С. В. МАТВІЙЧУК

НАВЧИТИ УЧНІВ ПРОГРАМУВАТИ

ПЕРЕДМОВА

Олімпіади — це той «зріз», що перевіряє не тільки володіння предметом, але й формує тенденції розвитку цього предмета, визначає вимоги до школи через цей предмет з боку суспільства, тобто те, що називають соціальним запитом. Олімпіади з інформатики (з програмування) як за змістом, так і за методикою проведення можна вважати сформованим явищем.

Спробуймо відповісти на таке просте запитання: яке головне завдання цих олімпіад? Безумовно, формальна мета — виявити на кожному з її етапів переможців в особистому заліку і відповідно визначити рейтинги територіальних одиниць — міст, районів, областей і навчальних закладів. Але не тільки це.

На мій погляд, головним завданням наших олімпіад будь-якого їх етапу є навчити учнів програмувати. Тобто під час проведення змагань учень потрапляє в екстремальні умови, коли за обмежений час потрібно:

- прочитати і правильно зрозуміти постановку задачі;
- знайти доцільний спосіб її розв'язання (це найвідповідальніший етап);
- описати ефективну математичну модель, використавши ідею;
- і нарешті, закодувати алгоритм, використавши відповідну мову програмування.

Чим це не науковий проект?

Учасник олімпіади повинен за 4–5 годин знайти алгоритми розв'язання декількох досить складних задач, написати і налагодити (перевірити за допомогою тестів) загалом від 100 до 500 рядків програмного коду. Чи можливо це? Практика показує, що так. Років 25–30 тому вважалося, що на 5–10 рядків коду день — це висока продуктивність професійного програміста. Критерій неоднозначний, але прогрес вражає.

Ми не замислюємося над побудовою фраз, спілкуючись рідною мовою. Так само учасник олімпіади виражає свої думки в системі програмування, автоматично створюючи програмний код без помилок. А отут уже без володіння структурною парадигмою мислення ніяк не обійтись.

Що стосується алгоритмів, то перелічимо лише основні розділи:

- арифметика цілих чисел;
- комбінаторика;
- пошук і сортування;
- математичне моделювання;
- довга арифметика цілих і дійсних чисел;
- жадібний алгоритм;

- алгоритми на графах (зв'язність, найкоротші шляхи, цикли, потоки в мережах тощо);
- перебір і методи його скорочення;
- рекурсивні алгоритми;
- динамічне програмування;
- метод гілок і меж;
- метод «хвилі»;
- аналітична геометрія (формули геометричних перетворень на площині, скалярний і векторний добутки, рівняння прямої, перетин прямих, належність точки многокутнику, опукла оболонка та ін.).

За кожною темою необхідно розв'язати визначену кількість задач, довести їх до рівня працюючих програм. Якщо, наприклад, задача, незалежно від змісту, зводиться після ряду перетворень до алгоритму пошуку найкоротшого шляху в графі, то її вже розв'язано. Учасник олімпіади, встановивши цей факт на свідомому рівні, всю іншу роботу виконує майже як автомат, вона не повинна вимагати значних зусиль на свідомому рівні — можна переходити до наступної задачі.

Останнім часом важливу допомогу в підготовці і проведенні олімпіад з програмування надають інтернет-ресурси спортивного програмування: www.e-olimp.com.ua, www.acm.lviv.ua, www.acmp.ru.

Перший із цих сайтів створений за ініціативою наших колег — учасників Всеукраїнського конкурсу «Вчитель року», відомих педагогічній громадськості вчителів інформатики Житомирщини Сергія Жуковського та Анатолія Присяжнюка, а програмний код інтернет-сайту належить колишньому учневі С. Жуковського ліцеїсту Сергію Колодяжному, призеру Всеукраїнської олімпіади.

На цих сайтах олімпіада не закінчується ніколи, тобто кожен такий сайт має архів задач різного рівня складності і будь-який зареєстрований користувач може, розв'язавши задачу, надіслати її на перевірку в будь-який момент часу протягом доби. Перевірка програми відбувається впродовж декількох секунд і здійснюється програмою за авторськими тестами. Якщо програма «проходить», тобто долає весь набір тестів, вона зараховується користувачеві і той отримує певну кількість балів, переходить на наступну сходинку в загальному рейтингу.

Тут наведені складені авторами задачі з обласних, районних та інтернет-олімпіад з інформатики Житомирської області за останні 10 років, багато з них розміщені в архіві задач сайту E-olimp. Коди програм та опис алгоритмів стануть у пригоді учасникам олімпіад та їх учителям.

СТОРІНКИ

(Р-2003)

Для нумерації M сторінок у книжці використано N цифр. Скільки сторінок у книжці?

Розв'язання

Для розв'язання цієї задачі не потрібен комп'ютер, а тим більше складна мова програмування. Достатньо мати товсту книжку, нехай навіть за того ж Паскаля. Головне, щоб сторінок було якнайбільше. Далі запасаємося терпінням і... гортаємо сторінки та рахуємо сумарну кількість використаних цифр для нумерації. Щойно набравши M цифр, дивимося, на якій ми сторінці, — це і є шукане N. Ось такий алгоритм.

Залишилося скласти програму, яка поставить усі значення на свої місця, і підручник з Паскаля знадобиться вже зовсім для іншого. Весь смак програмування полягає в можливості моделювати будь-який реальний або віртуальний процес і отримати відповіді на всі запитання, «не гортаючи сторінок».

Далі без компілятора якоїсь із мов програмування не обійтися (найбільш зручним у нашому випадку є Free Pascal). Дещо конкретизуємо алгоритм у змінних. N — кількість цифр, прочитана з файла даних; q — кількість цифр у запису поточної сторінки M; p — сумарна кількість цифр, використана на сторінках 1... M, d змінюється так: 10, 100, 1 000..., коли змінна M наздоганяє d, то d набуває наступного значення, а q збільшується на 1.

Про всяк випадок зупиняємо цикл за умови, якщо $p \ge N$ (може трапитись, що значення N некоректне і відповідного значення M просто не існує). Після закінчення роботи команд циклу виводимо у файл остаточне значення M або 0, якщо результат відсутній.

Домовимося ще про однотипне введення даних у цю та інші програми, які ми далі розглядатимо. Значення, що мають потрапити у відповідні змінні, спочатку записуються в текстовий файл з ім'ям input.txt, а результат буде записаний у текстовий файл output.txt. Для зручності в секції **Const** програми можна вказати імена **Fi** для вхідного і **Fo** для вихідного файлів та надати їм відповідні значення, тоді за необхідності змінити ім'я файла завжди знаєш, де його шукати. Якщо потрібно забезпечити введення з клавіатури і виведення на екран, тоді в програмі треба стерти команди для роботи з файлами типу: **Assign**, **Reset**, **Rewrite** і **Close**, а в командах **Read**, **Readln**, **Writeln** і **Write** видалити звертання до файлових змінних типу **fd** та **fs**.

Кінцевий вигляд відповідної програми такий:

```
const
  Fi = 'input.txt';
  Fo = 'output.txt';
var
  N,M,q,d,p:LongInt;
  fd,fs:Text;
begin
  assign(fd,Fi);
  reset(fd);
  readln(fd,N);
  close(fd);
  d:=10; q:=1;
  p:=0; M:=0;
  repeat
    inc(M);
    if M=d then begin inc(q); d:=d*10 end;
  p:=p+q;
  until p>=N;
  assign(fs,Fo);
  rewrite(fs);
  if p=N then writeln(Fs,M) else writeln(Fs,0);
  close(fs);
end.
```

СЕРЕДНЄ З ЧИСЕЛ

(Р-2008)

Дано три різних числа a, b, c. Вивести середнє з них.

Розв'язання

Ось така тривіальна, на перший погляд, задача. Добре відомі стандартні алгоритми знаходження найбільшого і найменшого з трьох чисел, а тут потрібно вказати число, яке таким не є. Отже, насамперед знайдемо $\mathrm{Max}(a,b,c)$ і $\mathrm{Min}(a,b,c)$, викликавши дві відповідні функції, які знаходять найбільше і найменше з двох чисел.

А далі можна було б визначити число, що не дорівнює крайнім значенням, тобто три команди такого типу:

```
if (a<>Max) and (a<>Min) then writeln(a).
```

Якось не дуже оригінально! А пам'ятаєте загадку?

«А» и «Б» сидели на трубе,
«А» упало, «Б» пропало,
что осталось на трубе?

Звичайно, можна вчинити так: якщо від суми заданих трьох чисел відняти найбільше і найменше число, то зрозуміло, залишиться середнє. Я вважаю, що програма буде зрозуміла без особливих пояснень.

```
const
Fi = 'input.txt';
fFo= 'output.txt';
var a,b,c,s : Integer;
  r,w: Text;
Function Max (x,y:Integer):Integer;
begin
  if x>y then Max:=x else Max:=y;
end;
Function Min (x,y:Integer):Integer;
begin
  if x<y then Min:=x else Min:=y;
end;
begin
  assign(r,Fi);
  reset(r);
  readln(r,a,b,c);
  close(r);
  s:=(a+b+c)-(Max(Max(a,b),c)+Min(Min(a,b),c));
  assign(w,Fo);
  rewrite(w);
  writeln(w,s);
  close(w);
end.
```

ОПТИМАЛЬНА КУПІВЛЯ CD

(P-2006)

Чисті компакт-диски продаються в різних упаковках: більша — на 100 дисків — коштує 100 грн, менша — на 20 дисків — 30 грн, а один окремий диск коштує 2 грн. Якої мінімальної суми має вистачити, щоб купити N таких дисків?

Розв'язання

Пригадайте, як ви вчиняли кожного разу, коли потрібно було купити декілька чистих компакт-дисків. Безумовно, найвигідніше купити відразу сто дисків, а ті диски, які виявляться зайвими, стануть у пригоді і заощадять ваші фінанси в майбутньому.

Проте не поспішатимемо і насамперед обчислимо змінну X — кількість грошей, які потрібно заплатити за диски, якщо купувати їх точно за правилами без найменшого «авантюризму». Для цього потрібно знайти, скільки буде упаковок кожного виду, послідовно розділивши значення N на 100 і на 20, а потім — просто обчислити відповідну кількість грошей.

Але... Уявіть, що диски в упаковці по одному вже закінчились, тоді ми будемо змушені замість них узяти додатково пачку на 20 дисків. Сумарно дисків буде точно не менше N, а грошей Y потрібно буде заплатити більше, а може й менше, ніж у першому випадку.

І нарешті, якщо диски купувати лише в коробках по 100 дисків, тобто просто взяти на одну пачку більше таких дисків, всього буде Z грн, а інших упаковок не брати взагалі, то ще не відомо, яка сума — X, Y чи Z — виявиться меншою. Звичайно, у двох останніх випадках ми маємо виграш не тільки в коштах (їх буде менше), а у в дисках (їх буде більше).

```
const
  Fi = 'input.txt';
  Fo = 'output.txt';
var n,a,b,c : LongInt;
  x,y,z : LongInt;
  fd,fs : Text;
begin
  assign(fd,Fi);
  reset(fd);
  readln(fd,n);
  close(fd);
  a:=n div 100;
  n:=n mod 100;
  b:=n div 20;
  c:=n mod 20;
  x:=a*100+b*30+c*2;
  y:=a*100+(b+1)*30;
  z:=(a+1)*100;
  if y<x then x:=y;
  if z<x then x:=z;
  assign(fs,Fo);
  rewrite(fs);
  writeln(fs,x);
  close(fs);
end.
```

КІЛЬКІСТЬ ДНІВ

(Р-2008)

Дано дві календарні дати. Знайти кількість днів між ними, включаючи початковий і кінцевий дні.

Розв'язання

Дуже корисною буде програма, адже якщо у вхідних даних задати дату народження і сьогоднішню дату, то програма видасть кількість прожитих днів!

Спочатку навчимося обчислювати кількість днів від 01.01.01 — уявної точки відліку до введеної дати D.M.Y. Думаєте, що це буде величезне число? Зовсім ні, на 20.10.2009 маємо всього 733 700 днів.

Загальну кількість днів поділимо на три частини:

1) дні в $D-1$ повних роках можна обчислити, врахувавши 366 для кожного високосного року і 365 — для кожного звичайного. В програмі 365 + year(y), де year — функція, що повертає 1 для високосного року і 0 — для звичайного. Функція цілком відповідає «Григоріанському календарю», тобто рік є високосним, якщо він кратний 4, але не кратний 100 або ж кратний 400;
2) дні в повних $M-1$ місяцях поточного року обчислені в програмі за допомогою функції month (j,y), де враховано місяці, які мають 30 або 31 день, а також окремо прописано кількість днів у лютому;
3) дні, що минули в поточному місяці поточного року, дорівнюватимуть D.

Різниця між результатами для двох заданих дат плюс 1 (щоб врахувати початковий день проміжку, який ми вже відняли) буде відповіддю до задачі.

```
const
Fi = 'input.txt';
Fo = 'output.txt';
var
  d1,m1,y1,s1 : LongInt;
  d2,m2,y2,s2 : LongInt;
  r,w: Text;
Function year(y:LongInt):LongInt;
begin
  if (y mod 4 =0) and
     (y mod 100 <>0) or (y mod 400=0)
  then year:=1 else year:=0;
end;
Function month (m,y : LongInt) : LongInt;
var w : LongInt;
begin
  w:=31;
  if m=2 then w:=28+year(y);
  if (m=4) or (m=6) or (m=9) or (m=11)
  then w:=30;
  month:=w;
end;
```

```
Function Count (d,m,y:LongInt):LongInt;
var i,j,s:LongInt;
begin
  s:=0;
  for i:=1 to y-1 do s:=s+365+year(i);
  for j:=1 to m-1 do s:=s+month(j,y);
  Count:=s+d;
end;
begin
  assign(r,Fi);
  reset(r);
  readln(r,d1,m1,y1);
  readln(r,d2,m2,y2);
  close(r);
  assign(w,Fo);
  rewrite(w);
  writeln(w,Count(d2,m2,y2)-Count(d1,m1,y1)+1);
  close(w);
end.
```

МОДЕЛЬ ГОДИННИКА З БОЄМ

(Р-2002)

Годинник з боєм відбиває кожної години таку кількість ударів, скільки їх є на циферблаті із цифрами від 1 до 12, та по одному разу тоді, коли хвилинна стрілка вказує на цифру 6. Знаючи початковий та кінцевий час однієї календарної доби (в годинах і хвилинах), обчислити загальну кількість ударів за цей проміжок часу.

Розв'язання

Цілком зрозуміла модель: потрібно емулювати годинник з боєм. Час у задачі потрібно перевести в одну одиницю виміру — хвилини, тоді ми зможемо просто оперувати з моментами часу в арифметичних формулах і логічних виразах. Нам знадобиться функція **Min**, щоб перевести значення годин і хвилин у хвилини.

Важливо правильно підібрати базову змінну, щоб генерувати ті моменти, коли годинник відбиває удари. Нехай змінна L буде рахувати події, коли годинник відбиває деяку кількість ударів. Змінюємо L від 0 до 47, тоді $T = L \cdot 30$ — момент часу в хвилинах, коли настає L-та серія ударів.

Кількість ударів годинника U також знаходимо за значенням змінної L:

- якщо значення L непарне, то кількість ударів U буде 1;
- якщо значення L парне, то обчислюємо U за такою формулою: `U=(L mod 24) div 2`.

Формула дає правильний результат для всіх моментів часу, за виключенням 00:00 та 12:00, де значення U дорівнює 12, тому коректуємо U в таких випадках:

L — номер серії	Час	T — час у хв	U ударів
0	00:00	0	12
1	00:30	30	1
2	01:00	60	1
3	01:30	90	1
4	02:00	120	2
5	02:30	150	1
...	...	...	...
22	11:00	660	11
23	11:30	690	1
24	12:00	720	12
25	12:30	750	1
26	13:00	780	1
27	13:30	810	1
28	14:00	840	2
...	...	...	...
46	23:00	1380	11
47	23:30	1410	1

```
const
  Fi = 'input.txt';
  Fo = 'output.txt';
var
  H1,M1,H2,M2,T1,T2,C,T,L,U:Integer;
  Fi,Fo:Text;
begin
  assign(Fi,Fi); reset(Fi);
  readln(Fi,H1,M1);
  readln(Fi,H2,M2);
  close(Fi);
  C:=0;
  T1:=H1*60+M1;
  T2:=H2*60+M2;
  for L:=0 to 47 do begin
    T:=L*30;
    if odd(L) then U:=1
    else begin
```

```
      U:=(L mod 24) div 2;
      if U=0 then U:=12;
    end;
    if (T1<=T) and (T<=T2) then inc(C,U);
  end;
  assign(Fo,Fo);
  rewrite(Fo);
  writeln(Fo,C);
  close(Fo);
end.
```

НУЛІ В КІНЦІ ЗАПИСУ *N*!

(P-2005)

Обчислити кількість нулів у кінці запису факторіала натурального числа N ($N \le 2 \cdot 10^9$).

input.txt	7	Значення N
output.txt	1	Кількість нулів у кінці запису $N!$

Розв'язання

Це перша, але не остання задача про факторіал числа, який поряд із такими математичними поняттями, як прості числа, степені двійки, послідовність Фібоначчі та ін. часто зустрічаються в умовах задач із програмування і, відповідно, алгоритмах. Очевидно, що такі поняття мають бути чітко сформовані, а програма, яка обчислює послідовність простих чисел, розкладає число на прості множники або друкує послідовність Фібоначчі, має бути написана менш ніж за 600 секунд без будь-яких шпаргалок. Тому для розминки пишемо програму, яка розв'язує задачу «в лоб» — просто обчислюємо $N!$ і ділимо це значення на 10 — результатом буде кількість таких ділень. Зауважимо, що навіть для $N!$ розміру **Int64** (20!= 2432902008176640000) нулів буде всього 4 — не багато. Оскільки програма створюється лише для досліджень, то не будемо себе обтяжувати роботою з файлами.

```
var
  n,i,k:LongInt;
  f:Int64;
begin
  readln(n);
  k:=0; f:=1;
```

```
  for i:=1 to n do f:=f*i;
  while f mod 10=0 do begin
    f:=f div 10;
    inc(k);
  end;
  writeln(k);
end.
```

Вочевидь, така програма не витримує будь-якої критики, і тому в наступній не будемо обчислювати значення $N!$ явно, а кількість нулів визначимо за кількістю дільників 5, що складають добуток $1*2*3*...*N$ — кожна така п'ятірка, помноживши на двійку, дасть один 0 у кінці запису $N!$. Зрозуміло, що дільників 2 у $N!$ (при $N>1$) буде значно більше, ніж дільників 5. Отже, висновок такий: нулів у кінці запису $N!$ буде рівно стільки, скільки буде сумарно дільників 5 у чисел $1, 2, 3, ...N$.

Функція Div5 обчислює, скільки дільників 5 має задане число m, а всі її виклики для чисел $1, 2, 3, ...N$ підсумовуються в основній частині до змінної k, яка містить кінцевий результат. Дуже часто вдало дібрана функція або процедура вносить до коду деяку прозорість і зрозумілість — програма читається, як формулювання математичного твердження.

```
var N,i,k : LongInt;
Function Div5 (m:LongInt):LongInt;
var d:LongInt;
begin
  d:=0;
  while (m mod 5=0) do begin
    m:=m div 5;
    inc(d);
  end;
  Div5:=d;
end;
begin
  readln(N);
  k:=0;
  for i:=1 to N do k:=k+Div5(i);
  writeln(k);
end.
```

Стало значно краще, але в разі спроби розмістити цю версію програми на www.acm.lviv.ua, ми б мали результат: ліміт часу на тесті 19. Це примушує нас замислитися над ефективністю алгоритму більш серйозно.

Висновок про те, що кількість нулів дорівнює кількості простих дільників 5, є цілком правильним, і це підтверджують 18 пройдених програмою тестів. Очевидно, нам не потрібно досліджувати подільність на 5 кожного з чисел від 1 до N, коли ми точно впевнені, що в крайньому випадку один дільник 5 має тільки кожне п'яте з них, тобто 5, 10, 15, 20,..., не менш ніж два мають — 25, 50, 100, ..., а числа, що діляться на 125, мають три або більше дільників 5.

Наприклад, як дізнатися, скільки нулів є в кінці запису 169!? Серед чисел від 1 до 169 по одному і більше дільнику 5 дадуть числа 5, 10, 15, 20, ... 165, не менш ніж два дільники 5 мають 25, 50, 75, 100, 125, 150, а три має лише число 125. Відповідно, чисел у кожній групі буде `169 div 5 = 33`; `33 div 5 = 6`; `6 div 5 = 1`.

Сумарну кількість дільників 5 (або нулів у кінці запису 169!) обчислюємо простим додаванням 33 + 6 + 1 = 40. Ось ця програма (з розміру коду помітно, що ми добре поміркували).

```
var n,i,k : LongInt;
begin
  readln(n);
  k:=0;
  repeat
    n:=n div 5;
    k:=k+n;
  until n=0;
  writeln(k);
end.
```

Для самостійного опрацювання можна запропонувати таку задачу XIX Всеукраїнської олімпіади з інформатики.

За заданим натуральним числом N необхідно обчислити кількість натуральних чисел, які є дільниками $N!$ (факторіалу числа N).

Наприклад, при $N = 4$, $N! = 4 \cdot 3 \cdot 2 \cdot 1 = 24$. Це число має такі дільники: 1, 2, 3, 4, 6, 8, 12, 24. Таким чином, шукана кількість дорівнює 8.

input.txt	4	Значення N
output.txt	8	Кількість дільників $N!$

СИМЕТРІЯ ЧИСЕЛ

(Р-2003)

Скільки існує N-значних паліндромів?

(Паліндром — це число, що читається однаково в обох напрямках). $N = 1...1000$.

Розв'язання

Математична модель цієї задачі очевидна — маючи значення N, знаходимо значення найменшого A та найбільшого B серед N-значних чисел, в діапазоні від A до B шукаємо паліндроми для того, щоб їх порахувати. Щоб перевірити число на «паліндромність» скористаємося функцією Back, яка переписує цифри числа в оберненому порядку, і якщо її значення збігається із числом, то ми знайшли паліндром.

Ось програма:

```
const
  Fi = 'input.txt';
  Fo = 'output.txt';
var
  N,M,A,B,i:Longint;
  fd,fs:Text;
Function back (x:LongInt):Longint;
var y:LongInt;
begin
  y:=0;
  repeat
    y:=y*10+(x mod 10);
    x:=x div 10;
  until x=0;
  back:=y;
end;
begin
  assign(fd,Fi);
  reset(fd);
  readln(fd,N);
  close(fd);
  B:=1;
  for i:=1 to N do B:=B*10;
  A:=B div 10; B:=B-1;
  M:=0;
  For i:=A to B do
    if i=back(i) then inc(M);
  assign(fs,Fo);
  rewrite(fs);
  writeln(fs,M);
  close(fs);
end.
```

Усе працює правильно, але вже для випадку $N=7$ час виконання програми на швидкому процесорі складає близько хвилини, а такий результат нас не влаштовує. Крім того, стандартний тип

Longint дасть можливість отримати результат тільки для $N \le 9$. А для більших значень N?

Керуючись аксіомою «не існує програми, яку не можна було б покращити», будемо шукати інші підходи. Спочатку запишемо результати, отримані на попередньому етапі.

N	M
1	9
2	9
3	90
4	90
5	900
6	900
7	9 000

Цілком впевнено можна стверджувати, що для $N = 8$ результат буде 9 000, а для $N = 9$ — 90 000. Цікаво, чому?

Будемо міркувати так: з будь-якого двозначного числа ми можемо утворити рівно один тризначний і один чотиризначний паліндром. Наприклад, із числа 72 маємо 727 і 7 227. Тому кількість тризначних і чотиризначних паліндромів дорівнює кількості двозначних чисел тощо. Отже, N-значних паліндромів рівно стільки, скільки існує $((N+1)$ div 2)-значних чисел, з яких кожен з паліндромів однозначно формується.

Декілька слів про наступну програму, яка долає будь-які обмеження. Результат будемо формувати в текстову змінну, він складатиметься з цифри 9 та $((N-1)$ div 2) нулів, тому обмеження можна розширити до $N \le 510$. Якщо результат до файла писати відразу, тобто цифру 9, а потім $((N-1)$ div 2) цифр 0, то значення N прямує до **LongInt**. Час виконання програми дорівнює часу виконання циклу з (N div 2) команд — дуже добре. Такий підхід схожий на динамічний підхід у програмах, коли наступний результат обчислюється за допомогою вже знайдених на попередніх етапах. Але не все відразу.

Маємо програму:

```
const
  Fi = 'input.txt';
  Fo = 'output.txt';
var
  N,i: Longint;
  fd,fs:Text;
begin
  assign(fd,Fi);
```

```
    reset(fd);
    readln(fd,N);
    close(fd);
    assign(fs,Fo);
    rewrite(fs);
    write(fs,'9');
    i:=0;
    while (i < (N-1) div 2) do begin
      inc(i);
      write(fs,0);
    end;
    writeln(fs);
    close(fs);
end.
```

А ось ще задача на аналогічну паліндромну тему.

Задано натуральне число M. Якщо це не паліндром, то записуємо його у зворотному порядку та додаємо до заданого. Кроки повторюємо, доки не отримаємо число-паліндром. Кількість виконаних операцій назвемо рівнем паліндромності заданого числа. Знайти рівень паліндромності числа $M = 1...9999$.

Математична модель дещо інша за змістом і методами розв'язування, але, безумовно зрозуміла. Спробуйте скласти програму самостійно.

ПОПУЛЯЦІЯ РОБОТІВ

(Р-2008)

A роботів за рік збирають C роботів, а B роботів — D роботів. На початку є N роботів. Яка найбільша кількість роботів може бути через K років?

Розв'язання

Популяція — сукупність осіб одного виду, що займають обмежений ареал, вільно схрещуються одна з одною, мають спільне походження, генетичну основу (так стверджує Вікіпедія).

Математична модель дещо схожа за формою на інтерпретацію чисел Фібоначчі, а за змістом це класична задача лінійного програмування. Розглянемо математичну модель: необхідно знайти такі цілі невід'ємні X і Y, що в разі виконання обмеження $AX + BY \le N$ значення виразу $CX + DY$ набуває максимального значення, де X, Y — кількість бригад роботів по A і B членів відповідно.

Для розв'язування не будемо згадувати відомий в інформатиці так званий «симплекс-метод», а застосуємо повний перебір, враховуючи, що змінна X може набувати значень від 0 до N div A, а змінна Y — від 0 до $(N - AX)$ div B. Процедуру визначення мак-

симального числа новостворених роботів у програмі можна описати окремою функцією, а для розв'язування задачі в межах K років потрібно виконати цю функцію декілька разів, змінивши тільки початкову кількість роботів, яка, зрозуміло, з кожним роком буде збільшуватися на кількість роботів, створених за попередній рік. Тому можна запропонувати таку програму:

```
var
  a,b,c,d,k,n,i:LongInt;
  f,s: Text;
  zFunction Robot (r:LongInt):LongInt;
var x,y,m:LongInt;
begin
  m:=0;
  for x:=0 to r div a do
    for y:=0 to (r-a*x) div b do
      if x*c+y*d>m then m:=x*c+y*d;
  robot:=m;
end;
begin
  assign(f,'input.txt');reset(f);
  readln(f,a,b,c,d,n,k);close(f);
  for i:=1 to k do n:=n+robot(n);
  assign(s,'output.txt');
  rewrite(s);
  writeln(s,n);
  close(s);
end.
```

КОМП'ЮТЕРИЗАЦІЯ

(О-2005)

За перший рік комп'ютеризації в школах встановили X нових комп'ютерів. За другий рік їх було завезено більше — Y. Міністерство освіти вирішило: кожного наступного року надавати школам стільки комп'ютерів, скільки їх було встановлено за два попередні роки разом.

N-го року встановили рекордну кількість — C комп'ютерів. Знаючи N і C, знайти значення X і Y ($N < 20$, $C < 2\,000\,000\,000$). Вказати відповідь із найменшим значенням X.

input.txt	6 37	Значення N і C
output.txt	4 5	Значення X, Y

Розв'язання

У цій задачі числа Фібоначчі фігурують вже не формально, а реально. Хоча не будемо поспішати і насамперед проаналізуємо кількість комп'ютерів, встановлених кожного року.

Рік, (N)	Комп'ютерів, (C)	Множник X, (A)	Множник Y, (B)
1	X	1	0
2	Y	0	1
3	$X+Y$	1	1
4	$X+2Y$	1	2
5	$2X+3Y$	2	3
6	$3X+5Y$	3	5
7	$5X+8Y$	5	8
8	$8X+11Y$	8	11

Очевидно, що множники біля X і Y утворюють послідовність Фібоначчі: для X — починаючи з третього, а для Y — з другого років. Отже, математична модель цієї задачі зводиться до пошуку розв'язку в натуральних числах діофантового рівняння $AX+BY=C$, де $A=F_{n-2}$, $B=F_{n-1}$ — відповідні члени послідовності Фібоначчі, які можуть бути елементарно обчислені за допомогою класичного циклу. З умови задачі необхідно уточнити, що $X<Y$ і перший знайдений розв'язок відповідатиме найменшому значенню X.

```
const
  Fi = 'input.txt';
  Fo = 'output.txt';
var
  X,Y,C,N,A,B:LongInt;
  f,s: Text;
Procedure Init;
begin
  assign(f,Fi);
  reset(f);
  readln(f,N);
  readln(f,C);
  close(f);
end;
Procedure Run_Done;
var i,p:LongInt;
```

```
begin
  assign(s,Fo);
  rewrite(s);
  A:=1; B:=1;
  for i:=4 to N do begin
    p:=A+B;
    A:=B;
    B:=p;
  end;
  For X:=1 to (C div a) div 2 do
    if ((C-X*a) mod b=0) then begin
      Y:=(C-X*a) div b;
      if (X<Y) then begin
        writeln(s,X,' ',Y);
        Break;
    end;
  end;
  close(s);
end;
begin
  Init; Run_Done;
end.
```

БУДІВНИЦТВО ПЛОТУ

(О-2006)

Використавши N дерев'яних заготовок довжиною $L[i]$ $(i=1\ldots N)$, потрібно побудувати пліт з M колод однакової довжини. Знайти найбільшу можливу довжину плота P. Усі числові значення натуральні, не більші за 1 000.

input.txt	3 4 20 70 25	Значення N і M Значення $L[i]$ $(i=1\ldots N)$
output.txt	23	Значення P

Розв'язання

Зауважимо, що для побудови плоту дерев'яні заготовки можна тільки розрізати на колоди якомога більшої однакової довжини, але не можна склеювати. Для побудови плоту необхідно не менше, ніж M колод.

Насамперед знайдемо обмеження максимальної довжини плоту. Якщо уявити, що всі колоди будуть вирізані з одного дерева, то в такому випадку відходів буде мінімально мало, а матеріал буде

використано максимально ефективно. У випадку N заготовок потрібно обчислити їх сумарну довжину, розділивши її націло на M (у програмі — змінна P).

З іншого боку, зрозуміло, що пліт не може бути довшим за довжину найбільшої заготовки (в програмі — змінна PP). Мінімальне з цих двох значень буде максимально можливою довжиною плоту P — початкове значення, з якого потрібно розпочати пошук, кожна ітерація якого буде описувати такі дії:

1) знаходимо, скільки колод довжиною P отримаємо з N заготовок;
2) якщо менше від M, то зменшуємо P на одиницю — ще одна ітерація.

Очевидно, можна запропонувати й інші алгоритми для побудови плоту максимальної довжини, хоча за заданих обмежень для змінних програма, наведена нижче, забезпечує непогану ефективність.

```
const
  FileIn= 'input.txt';
  FileOut = 'output.txt';
  _p = 1000;
var
  N,M: Integer;
  X:Array [1.._p] of Integer;
  P,PP : LongInt;
Procedure Init;
var
  f:Text;
  i:Integer;
begin
  assign(f,FileIn);
  reset(f);
  readln(f,N,M);
  for i:=1 to N do read(f,x[i]);
  close(f);
end;
Procedure Run;
var i,w:Integer;
begin
  P:=0; PP:=0;
  for i:=1 to N do begin
    P:=P+x[i];
    if x[i]>PP then PP:=x[i];
  end;
```

```
    P:=P div M+1;
    if PP<P then P:=PP;
    repeat
      Dec(P);
      w:=0;
      for i:=1 to N do inc(w,(x[i] div P));
    until w>=M;
  end;
  Procedure Done;
  var f:Text;
  begin
    assign(f,FileOut);
    rewrite(f);
    writeln(f,P);
    close(f);
  end;
  begin
    Init; Run; Done;
  end.
```

ОЛІМПІАДНА ЗАДАЧА ПРО ОЛІМПІАДУ

(Р-2003)

На обласну олімпіаду з інформатики прибуло N команд по $A[i]$ учасників у кожній $(i=1...N)$. Для проведення олімпіади виділили комп'ютерні кабінети по C комп'ютерів у кожному. Яку мінімальну кількість K кабінетів потрібно задіяти за умови, що в кожному кабінеті будуть представники лише різних команд?

Розв'язання

Традиційне завдання, яке розв'язує журі обласної олімпіади перед її проведенням, тепер запропоноване у вигляді задачі такої ж олімпіади — це своєрідна рекурсія. На жаль, для розв'язування нічого рекурсивного придумати не вдалося.

Базовими знаннями в цій задачі є одновимірні масиви та їх елементарна обробка, тобто сума, середнє, максимум тощо. Залишається тільки правильно застосувати ці базові алгоритми.

Уявімо, що обмежень на розміщення учасників немає. Тоді можна знайти кількість усіх учасників S і вибрати K так, щоб значення $S \le C * K$, тобто за необхідності збільшити кількість учасників, щоб S було кратним C.

Тепер задовольнити умову задачі ми не зможемо тільки у випадку, коли учасників в одній або декількох командах буде більше ніж K, а для їх розміщення кабінетів потрібно буде рівно стільки, скільки є в них учнів.

Маємо алгоритм:

1) знайти суму елементів масиву $A[i]$ і поділити її на C, попередньо збільшивши до кратного C;
2) знайти максимальний елемент у масиві $A[i]$;
3) відповідь до задачі — максимум із двох попередніх результатів.

```
const
  Fi = 'input.txt';
  Fo = 'output.txt';
  maxN = 100;
var
  C,N,K:Word;
  A:array [1..maxN] of Word;
  fd,fs:Text;
Procedure Init;
var i:Word;
begin
  assign(fd,Fi);
  reset(fd);
  readln(fd,N,C);
  For i:=1 to N do read(fd,A[i]);
  close(fd);
end;
Procedure Run;
var i,S,M:Word;
begin
  M:=0; S:=0;
  for i:=1 to N do begin
    if A[i]>M then M:=A[i];
    inc(S,A[i]);
  end;
  while not (S mod C = 0) do inc(S);
  S:=S div C;
  if M>S then K:=M else K:=S;
end;
Procedure Done;
begin
  assign(Fs,Fo);
  rewrite(Fs);
  writeln(Fs,K);
  close(Fs);
end;
begin
  init; Run; Done;
end.
```

Для самостійної роботи пропонується задача «Інтернет-кафе» з обласної олімпіади 2003 року. Розв'язавши її, зрозумієте, чому вона лишилася без пояснень. Ось її умова:

ІНТЕРНЕТ-КАФЕ

(О-2003)

До порожнього інтернет-кафе, у якому є C комп'ютерів, зайшла компанія з N друзів, щоб особисто кожному надіслати листи електронною поштою (по 1 хв на кожен лист). Кількість листів у членів компанії записані в масиві $L[1...N]$. Визначити мінімальний час (хв), впродовж якого всі листи будуть надіслані.

Усі числові значення натуральні і не перевищують 100.

input.txt	2 3 2 4 5	Значення C і N Один рядок з N числами: кількість листів у членів компанії
output.txt	6	Мінімальний час надсилання всіх листів

А тепер нова задача.

ОПТИМАЛЬНЕ ОБ'ЄДНАННЯ ФАЙЛІВ

(О-2002)

На жорсткому диску містяться N файлів, які необхідно об'єднати в один. Стандартна програма, отримавши на вхід два файли, об'єднує їх в один, витрачаючи по 1 с на кожен Мб від сумарного розміру вхідних файлів, кожен з яких не перевищує 1 Гб.

Знайти такий порядок об'єднання N файлів, щоб загальний час об'єднання був мінімальним. Розміри кожного з N файлів подано в текстовому файлі input.txt, відповіді записати у файл output.txt: перший рядок — мінімальний час об'єднання файлів, в інших $N-1$ рядках — розміри кожного з двох файлів, які об'єднуються ($N<100$).

input.txt	3 20 5 10	Значення N Розміри файлів
output.txt	50 10 5 15 20	Мінімальний час с, Об'єднали 10 і 5 Мб за 15 с. Об'єднали 15 і 20 Мб за 35 с

Розв'язання

Очевидно, що кількість операцій об'єднання для M файлів буде $N-1$. Для мінімізації загального часу необхідно мінімізувати

час кожної окремої операції, тобто щоразу вибирати для об'єднання два найменші файли з наявних на певний момент. Ось такий динамічний алгоритм.

Дещо конкретизуємо алгоритм для змінних.

Повторити $N-1$ разів такі дії:

1) відсортувати елементи масиву за спаданням, причому переглядати масив будемо від старших індексів;
2) замінити передостанній елемент сумою останнього і передостаннього;
3) врахувати час операції в загальний час;
4) зменшити M на одиницю.

Програма, що пропонується нижче, реалізує цей алгоритм, але знаходить тільки мінімальний час об'єднання файлів. Вивести послідовність об'єднання — справа техніки.

```
const
  Fi = 'input.txt';
  Fo = 'output.txt';
  maxN = 1000;
var N:Word;
  X : array [1..maxN] of Word;
  S : LongInt;
Procedure Init;
var i:Word; Fd:Text;
begin
  assign(Fd,Fi); reset(Fd); readln(Fd,N);
  For i:=1 to N do read(Fd,X[i]);
  close(Fd);
end;
Procedure Swap(var a,b:Word);
var c:Word;
begin
  c:=a; a:=b; b:=c;
end;
Procedure Sort(M:Word);
var
  i:Word;
  F : Boolean;
begin
  repeat
    F:=True;
    for i:=M-1 downto 1 do
      if X[i]<X[i+1] then begin
         swap(X[i],X[i+1]); F:=false
```

```
      end;
    until F;
  end;
  Procedure Run;
  var
    M:Word;
    R : LongInt;
  begin
    S:=0; M:=N;
    repeat
      Sort(M);
      R:=X[M-1]+X[M]; S:=S+R; X[M-1]:=R;
      Dec(M);
    until M=1;
  end;
  Procedure Done;
  var Fs:Text;
  begin
    assign(Fs,Fo);
    rewrite(Fs);
    writeln(Fs,S);
    close(Fs);
  end;
  begin
    init; Run; Done;
  end.
```

СТЕПІНЬ ДВІЙКИ

(О-2007)

У вхідному файлі послідовно записані N степенів двійки, тобто числа від 2 до 2^N без пропусків ($1 \le N \le 1000$). Знайдіть значення N.

input.txt	248163264128	Послідовність степенів двійки
output.txt	7	Значення N

Розв'язання

Справді програмістська задачка. Двійка — основа системи числення для кодування абсолютно всієї комп'ютерної інформації, степенями двійки вимірюються мегабайти оперативної пам'яті і гігабайти зовнішніх запам'ятовувальних пристроїв і ще багато іншого. Десь зустрічав таку головоломку для програмістів — потрібно продовжити послідовність двозначних чисел **24 81 63 26...** Ну як, помітили?

Але насправді у нас інша задача, хоча здається, що проблеми дещо схожі. Уявляєте у файлі таку послідовність з 55 цифр:

2481632641282565121024204840968192163843276865536131072, тобто

2 4 8 16 32 64 128 256 512 1024 2048 4096 8192 16384 32768 65536 131072?

Для такої послідовності потрібно розпізнати, яким показником степеня двійки закінчується цей числовий ряд.

Якби у файлі був другий варіант чисел, то можна було б порахувати кількість пропусків між ними і додати одиницю. А що можна придумати в першому випадку? Про всяк випадок, розмір файла даних для крайніх значень, тобто при $N = 1000$, становить 151 167 байтів. Файл пропускаємо, щоб вистачило місця для пояснення математичної моделі задачі.

Ідея полягає в тому, щоб послідовно читати з файла степені двійки і рахувати їх кількість, а потім вивести порядковий номер степеня, який був прочитаний останнім. Тому єдине запитання, на яке ми маємо відповісти, буде таким: скільки цифр має наступний степінь двійки, адже рівно стільки символів потрібно за кожною ітерацією прочитати з файла. Виявляється, контролювати такий процес дуже просто. Достатньо двох змінних — одна з них X буде типу **real** — теоретично це мантиса, а друга P — **Integer** — кількість цифр (або порядок плюс один) у наступного степеня двійки. Розпочинаємо значення цих величин з одиниці і змінюємо X множенням на два, але, щойно X перевищує 10, негайно ділимо значення X на 10, а значення P, тобто кількість цифр, прочитаних з файла наступного степеня, збільшуємо на 1. Дещо схожа ідея була використана в задачі «Нумерація сторінок книжки» і може застосовуватися в інших схожих моделях для виділення подібних об'єктів з «купи».

```
const
  Fi='input.txt';
  Fo='output.txt';
  _max=1000;
var
  N,p: Integer;
  X:Double;
  fD,fs:Text;
Procedure Init_Run;
var  i:Integer;
begin
  assign(fD,Fi);
  reset(fD);
```

```
    X:=1;
    p:=1;
    N:=0;
    repeat
    inc(N); X:=X*2;
    if X>10 then begin inc(p); X:=X/10 end;
    for i:=1 to p do read(fD,w);
    until EoF(fD) or EoLn(fD);
    close(fD);
  end;
  Procedure Done;
  var i:Integer;
  begin
    assign(fS,Fo);
    rewrite(fS);
    writeln(fS,N);
    for i:=1 to p do write(fS,C[i]);
    writeln(fS);
    close(fS);
  end;
  begin
    Init_Run; Done;
  end.
```

Мій співавтор пропонував дещо інше розв'язання цієї і наступної задачі: оскільки ціла частина десяткового логарифма натурального числа (до мантиси) на одиницю менша від кількості цифр цього числа, легко обчислити суму цифр чисел від 2 до 2^p для кожного натурального p і відшукати таке p, щоб ця сума збігалася з кількістю цифр числа, прочитаного з текстового файла. Це і буде потрібна нам відповідь (якщо вважати, що всі дані коректні).

Дуже схожа задача, що стосується довжини значення факторіала натурального числа, була запропонована на обласній олімпіаді 2009 року.

КІЛЬКІСТЬ ЦИФР ФАКТОРІАЛА

(О-2009)

Знайти кількість цифр у запису факторіала натурального числа N $(1 \le N \le 1000000)$.

input.txt	7	Значення N
output.txt	4	Кількість цифр у числі $N!$

Цю задачу залишаємо без пояснень на користь наступної і більш об'ємної задачі про факторіал.

УСІ ЦИФРИ ФАКТОРІАЛА, АБО ПРОЦЕДУРИ «ДОВГОЇ АРИФМЕТИКИ»

(О-2000)

Написати програму, яка за значенням $N!$ визначатиме значення N ($1 \le N \le 2000$).

Розв'язання

Вчитаємося ще раз в умову задачі: задано достатньо велике ціле число F (наскільки велике — поки що не відомо), яке дорівнює факторіалу деякого цілого числа N (вже не такого великого, $1 \le N \le 2000$). Тобто маємо задачу, обернену до задачі обчислення факторіала, яку розв'язують у шкільному курсі інформатики. Що ж до нашої задачі, то її розв'язання можливе лише із застосуванням інших методів — так званої «довгої арифметики» (про це — далі).

Задача на перший погляд здається нескладною. Алгоритм повинен забезпечити послідовне обчислення факторіалів цілих чисел до отримання потрібного результату. Він може бути таким:

1) прочитати значення $N!$ з файла в змінну F;
2) почати змінну N з нуля;
3) збільшити значення N на одиницю та обчислити $N!$;
4) якщо $N!$ не дорівнює F, то виконати пункт 3, інакше вивести N.

Розглянемо програму, написану мовою Pascal, яка реалізує цей алгоритм. Щодо обчислення $N!$, звичайно, можна створити відповідну функцію, в такому випадку для обчислення кожного факторіала буде виконуватись N множень, а вже обчислені на якомусь етапі факторіали будуть обчислюватись ще і ще раз для більших значень N. Кількість операцій такого алгоритму буде мати порядок N^2, що не є занадто ресурсоємким для сучасної комп'ютерної техніки, але ознакою добротного стилю програмування є саме економія ресурсів програми.

Тому використовуємо рекурентну формулу $N! = N \cdot (N-1)!$ й обчислюємо $N!$ за одну операцію множення. Значення поточного факторіала буде зберігатись у змінній R, у яку послідовно записуватимемо значення 1!, 2!, 3!, ... поки не з'явиться потрібний результат.

```
var
  N:Word;     {Шукане ціле число}
  F:LongInt;  {Потрібне значення N!}
  R:LongInt;  {Поточне значення N!}
  Fi:Text;    {Файлова змінна}
begin
```

```
  assign(Fi,'input.txt');reset(Fi);
  readln(Fi,F);  {Читаємо значення F з файла}
  close(Fi);
  N:=0; R:=1;          {Ыніціалізація N та R}
  repeat
  inc(N);           {Збільшити N на одиницю}
  R:=R*N;           {Обчислити N!}
  until R=F;    {Перевірка закінчення циклу}
  writeln(N);   {Виведення знайденого значення N}
end.
```

Тест, наведений в умові задачі, програма проходить відмінно. Також маємо абсолютно нормальні результати для всіх значень факторіалів чисел від 1 до 12, тобто алгоритм розв'язання задачі складений правильно і програма, якщо не зважати на обмеження $N \le 2000$, працює правильно.

Цікаво, як буде вести себе програма при більших значеннях N? Пропонуємо у файлі даних число 6 227 020 800, яке дорівнює факторіалу 13, — замість результату маємо повідомлення про хибний числовий формат. Справді, найбільше значення, яке підтримує стандартний **LongInt**, дорівнює 2 147 483 647, і тому Паскаль відмовляється читати в змінних типу **LongInt** більші значення.

Вихід із цього становища може бути достатньо простим: замінити в програмі тип змінних F та R на **Real**. Тепер програма працює нормально для всіх факторіалів чисел до 27 включно. Повідомлення про переповнення під час операції з дійсним числом виникає за спроби обробити 28!. Справа в тому, що стандартний тип **Real** коректно обслуговує 11-12 цифр мантиси дійсного числа. Тому, за відомими лише програмістам фірми Borland причинами, внутрішні процедури мови програмування Паскаль читають з файла записане там натуральне число 304888344611713860501504000000 як 30488834461162918300000000000, а обчислення цього значення факторіала 28! дає іншу величину:
30488834461191741400000000000. Значення F та R не збігаються, і алгоритм зациклюється, тобто обчислення R продовжується, виходить за межі стандартного типу **Real**, і Паскаль повідомляє про переповнення.

Як обійти цю проблему? Можна запропонувати порівнювати значення U та R наближено або порівнювати не самі числа, а їх порядки (порядок дійсного числа можна знайти, обчисливши його десятковий логарифм). Виправляючи рядок умови виходу з циклу на `until Abs(Ln(R)-Ln(F))<0.01`, матимемо позитивний результат для факторіалів чисел 28, 29, 30, 31, 32, 33. Якщо прочитати

з файла число 34! (найбільше значення стандартного типу **Real** `1.7e38<34!`), отримуємо знайоме повідомлення про переповнення під час виконання операції з дійсним числом.

У таблиці дійсних типів вибираємо найбільший можливий тип — **Extended** (найбільше значення — 1.1e4932) і вважаємо, що тепер успіх вже дуже близько. Нова версія програми читається так:

```
var
  N : Word;
  F,R: Extended;
  Fi : Text;
begin
  assign(Fi,'input.txt');
  reset(Fi);
  readln(Fi,F);
  close(Fi);
  N:=0; R:=1;
  repeat
  inc(N);
  R:=R*N;
  until Ln(R)/Ln(10)=Ln(F)/Ln(10);
  writeln(N);
end.
```

Ця програма ставить своєрідний рекорд і працює нормально для факторіалів чисел від 1 до 49. Програма сходить з дистанції при $N = 50$. Виявляється, процедура `readln(X : Extended)` читає з файла не більше ніж 64 знаки, а наступні цифри просто ігноруються. Число 50!, яке має 65 цифр, читається в змінну U як 3.04140932017134E+0063, тому знову маємо зациклювання і повідомлення про переповнення під час операції з дійсним числом.

N	$N!$
10	3628800
11	39916800
12	479001600
13	6227020800
14	87178291200
15	1307674368000
16	20922789888000
17	355687428096000
18	6402373705728000

19	121645100408832000
20	2432902008176640000
21	51090942171709440000
22	1124000727777607680000
23	25852016738884976640000
24	620448401733239439360000
25	15511210043330985984000000
26	403291461126605635584000000
27	10888869450418352160768000000
28	304888344611713860501504000000
29	8841761993739701954543616000000
30	265252859812191058636308480000000
31	8222838654177922817725562880000000
32	263130836933693530167218012160000000
33	8683317618811886495518194401280000000
34	295232799039604140847618609643520000000
35	10333147966386144929666651337523200000000
36	371993326789901217467999448150835200000000
37	13763753091226345046315979581580902400000000
38	523022617466601111760007224100074291200000000
39	20397882081197443358640281739902897356800000000
40	815915283247897734345611269596115894272000000000
41	33452526613163807108170062053440751665152000000000
42	1405006117752879898543142606244511569936384000000000
43	60415263063373835637355132068513997507264512000000000
44	2658271574788448768043625811014615890319638528000000000
45	119622220865480194561963161495657715064383733760000000000
46	5502622159812088949850305428800254892961651752960000000000
47	258623241511168180642964355153611979969197632389120000000000
48	12413915592536072670862289047373375038521486354677760000000000
49	608281864034267560872252163321295376887552831379210240000000000
50	30414093201713378043612608166064768844377641568960512000000000000

Настав час зробити деякі висновки: запропонований алгоритм розв'язування задачі працює правильно, але всі спроби реалізувати його на стандартних числових типах мови Паскаль не принесли потрібного результату, програма не працює навіть для 50!, не кажучи вже про 2 000!. Але витрачені зусилля не були марними — з'явився досвід, який підказує, що потрібно обійти стандартний числовий тип.

Для цього необхідно створити свій числовий тип даних та написати відповідні процедури оброблення змінних такого типу. Достатньо відомим в інформатиці є метод емуляції так званої «довгої арифметики», коли невід'ємне ціле число розглядається в програмі у вигляді масиву своїх цифр. Вводимо свій тип даних:

```
type
  TNumber = Record
  L:Word; {Кількість цифр у числі}
  {Масив цифр числа}
  C:Array [1..7000] of Byte;
end.
```

Наприклад, число 10! = 3 628 800 буде розміщено в змінній *A*: Tnumber у такий спосіб:

```
A.L:=7;
A.C[1]:=0;
A.C[2]:=0;
A.C[3]:=8;
A.C[4]:=8;
A.C[5]:=2;
A.C[6]:=6;
A.C[7]:=3.
```

Усі наступні елементи масиву цифр числа необхідно обнулити, що забезпечить більш комфортну роботу зі змінними типу **Tnumber** під час виконання арифметичних операцій додавання і множення. Конструктивна будова даних такого типу є очевидною, тому залишилося написати декілька процедур оброблення змінних типу **Tnumber** для того, щоб набути деякого досвіду.

У першу чергу навчимося читати змінні типу **Tnumber** з файла. Можна було б насамперед розглянути процедуру введення з клавіатури, але в нашому випадку такої необхідності немає (уявляєте введення з клавіатури такого собі числа 100!, яке складається з 158 цифр або 1 000! — 2 568 цифр?!). Тому вважатимемо, що всі цифри числа знаходяться в одному рядку текстового файла, який закінчується кодом **EoLn**. Для введення з файла будемо виконувати посимвольне читання цифр числа в змінну `w : char` та послідовно присвоювати їх елементам масиву в числовому вигляді, використовуючи той факт, що байтові значення символів цифр відрізняються від самих чисел на 48, тобто `Ord('0')=48, Ord('1')=49` тощо.

Процедура, яка прочитає значення факторіала в змінну *F* з файла, може бути такою:

```
Procedure InputTNumber (var F:TNumber);
var
  w:Char;
```

```
  Z:Array [1..7000] of Byte;
  i,U:Word;
begin
  FillChar(F.C,SizeOf(F.C),0);
  U:=0; {Обнулити кількість прочитаних цифр}
  repeat
  {Читаємо наступну цифру з файла та записуємо її}
  {Числове значення наступним елементом масиву Z }
  inc(U);
  read(Fd,w);
  Z[U]:=Byte(w)-48;
  until EoLn(Fd); {Цикл до кінця рядка у файлі}
  readln(Fd);
  {Переписуємо масив Z у масив цифр F.C}
  {У зворотному порядку}
  for i:=1 to U do F.C[i]:=Z[U-i+1];
  {Заповнюємо поле кількості цифр}
  F.L:=U;
end;
```

Тепер пишемо процедуру для збільшення на одиницю заданого числа `A:Tnumber`**, тобто аналог процедури Inc. Почнемо з наймолодшого розряду числа і спробуємо збільшити поточну цифру на одиницю. Якщо цього зробити неможливо, то обнулимо поточний розряд і повторимо аналогічну операцію для наступних розрядів.**

```
Procedure incTNumber (var A:Tnumber);
var i:Word;
begin
  i:=1; {Почнемо з наймолодшого розряду}
  while A.C[i]=9 do {Якщо неможливо збільшити}
  {Поточну цифру, то обнулити її і перейти}
  begin {До наступного розряду}
    A.C[i]:=0; inc(i);
  end;
  inc(A.C[i]); {Інкремент поточної цифри }
  if i>A.L then A.L:=i;{Скоригувати довжину числа}
end.
```

Лише для того, щоб закріпити набутий досвід, пишемо процедуру знаходження суми двох змінних типу Tnumber. Будемо користуватись алгоритмом обчислення суми двох багатоцифрових чисел, відомим з арифметики. Отже, послідовно рахуємо суми відповідних цифр операндів, починаючи з молодших розрядів до довжини найбільшого числа (незначущі цифри в старших розрядах меншого числа замінені нулями).

```
Procedure AddTNumber (A,B:Tnumber; var M:Tnumber);
var R,K,i:Word;
begin
  FillChar(M.C,7000,0); {Обнулити масив цифр M}
  If A.L>B.L then K:=A.L else K:=B.L;
  {K — кількість додавань}
  for i:=1 to K do begin{Цикл по K цифрах A та B}
    R:=M.C[i]+A.C[i]+B.C[i];
    {Знайти результат додавання двох поточних цифр}
    M.C[i]:=R mod 10;{та записати цифру одиниць}
    M.C[i+1]:=R div 10; {до поточного розряду}
  end;{результату,а цифру десятків — до наступного}
  if M.C[K+1]>0 then M.L:=K+1 else M.L:=K;
  {Заповнити поле кількості цифр результату}
  {Залежно від переносу до K+1 розряду}
end;
```

Наступною буде процедура множення двох змінних A, B типу **Tnumber** з отриманням результату M такого ж типу. Процедура **MultTnumber** використовує алгоритм множення двох багатоцифрових чисел у стовпчик.

```
Procedure MultTNumber (A,B:Tnumber; var M:Tnumber);
var R,i,j:Word;
begin
  FillChar(M.C,7000,0);
  for i:=1 to A.L do
  for j:=1 to B.L do begin
    R:=M.C[i+j-1]+A.C[i]*B.C[j];
    M.C[i+j-1]:=R mod 10;
    M.C[i+j]:=M.C[i+j]+R div 10;
  end.
  M.L:=A.L+B.L-1
  if M.C[A.L+B.L]>0 then M.L:=A.L+B.L;
end;
```

І нарешті, функція порівняння двох змінних типу **Tnumber**:

```
Function CompTNumber (X,Y:Tnumber):Boolean;
var
  i:Word;
  F:Boolean;
begin
  if X.L=Y.L then begin
    F:=true;
    for i:=1 to X.L do F:=F and (X.C[i]=Y.C[i]);
  end
```

```
  else F:=false;
  Comp:=F;
end.
```

Цього матеріалу вже достатньо, щоб написати головний модуль програми розв'язування задачі FACTOR, але автор цих рядків не вважає метод емуляції довгої арифметики найкращим у цьому випадку. Тому процес створення більш досконалої програми залишимо читачеві. Зате більш корисною буде програма для запису у файл $N!$ $(1 \le N \le 2000)$, адже нам необхідно створювати тестові файли, бо обчислювати факторіали в інший спосіб для великих N є достатньо проблематичним.

```
uses Crt;
type
TNumber=Record
  L:Word;
  C:Array [1..7000] of Byte;
end;
var
  R,A:Tnumber;
  N,i:Word;
Procedure incTNumber (var A:TNumber);
var i:Word;
begin
  i:=1;
  while A.C[i]=9 do begin
    A.C[i]:=0; inc(i);
  end;
  inc(A.C[i]);
  if i>A.L then A.L:=i;
end;
Procedure MultTnumber (A,B:Tnumber; var M:Tnumber);
var R,i,j:Word;
begin
  FillChar(M.C,7000,0);
  for i:=1 to A.L do
  for j:=1 to B.L do begin
    R:=M.C[i+j-1]+A.C[i]*B.C[j];
    M.C[i+j-1]:=R mod 10;
    M.C[i+j]:=M.C[i+j]+R div 10;
  end;
    if M.C[A.L+B.L]>0 then M.L:=A.L+B.L else M.L:=A.
    L+B.L-1;
end;
```

```
Procedure OutToFile (A:Tnumber);
var
  Fd:Text;
  i:Word;
begin
  assign(Fd,'input.txt'); rewrite(Fd);
  for i:=A.L downto 1 do write(Fd,A.C[i]);
  close(Fd);
end;
begin
  clrScr;
  write('N='); readln(N);
  FillChar(A.C,7000,0);
  A.C[1]:=0; A.L:=1;
  FillChar(R.C,7000,0);
  R.C[1]:=1; R.L:=1;
  For i:=1 to N do begin
  incTnumber(A); MultTnumber(A,R,R);
  end;
  OutToFile(R);
end.
```

Тепер, коли ми маємо можливість обчислити факторіал будь-якого з чисел $1 \le N \le 2000$, поставимо таке запитання: чим у першу чергу відрізняється 99! і 100!? Відповідь правильна — кількістю цифр. Оскільки у файлі дійсно знаходиться факторіал деякого числа, то є логічним наступне запитання: чи є необхідність перевіряти всі цифри факторіала, якщо порядки чисел збігаються. Ні — достатньо перевірити декілька перших цифр. То навіщо взагалі читати з файла всі цифри факторіала?

Настав час продемонструвати найкраще з точки зору автора розв'язання задачі FACTOR. Зрозуміло, що остання редакція програми буде використовувати новий тип даних, у нашому випадку це емуляція дійсного числового типу:

```
Type
TReal=Record
  L:Word; {Порядок числа}
  C:Real; {Мантиса числа}
end.
```

Перед виконанням арифметичних і логічних операцій з такими числами вони мають бути нормалізованими, тобто поле C має бути не меншим від 1 і меншим від 10. Відповідна процедура нормалізації може бути такою:

```
Procedure NormalTReal (var X:TReal);
begin
  while X.C>10 do {поки мантиса числа перевищує 10}
  begin
    X.C:=X.C/10; {Понизити порядок мантиси і}
    inc(X.L); {Збільшити на 1 порядок числа}
  end;
end.
```

Процедура читання з файла числа і розміщення його в змінній типу **TReal** є доволі цікавою. Нагадаємо, що нам потрібні декілька (наприклад, 8) перших цифр числа та кількість його цифр (навіть не враховуючи перші вісім, адже ми завжди зможемо нормалізувати змінну типу **TReal**). Але не будемо більше випробовувати терпіння читача і продемонструємо програму повністю. Вважатимемо, що набутий досвід допоможе читачеві розібратися в ній без пояснень. Зауважимо лише, що початковий алгоритм роз'язання задачі залишився незмінним, а кожен з етапів цього розв'язання випробовував інший числовий тип даних на цьому алгоритмі. Отже, правильно вибрана структура даних для програми — це запорука успіху.

```
type
TReal = Record
  L:Word;
  C:real;
end;
var
  A,F:Treal;
  N:Word;
  Fd:Text;
  T,k:Byte;
Function Comp (X,Y:TReal):Boolean;
var i:Word;
begin
  Comp:=(X.L=Y.L)and (Abs(X.C-Y.C)<0.000001);
end;

Procedure Normal (var X:TReal);
begin
  while X.C>10 do begin
    X.C:=X.C/10;
    inc(X.L);
  end;
end;
```

```
Procedure Input (var X:TReal);
var
  w:Char;
  i:Word;
begin
  X.L:=0; X.C:=0; i:=0;
  repeat
  read(Fd,w); inc(i);
  if i<9 then X.C:=X.C*10+Byte(w)-48
    else inc(X.L);
  until EoLn(Fd);
  readln(Fd);
  Normal(X);
end;

begin
  assign(Fd,'input.txt');
  reset(Fd);
  input(F);
  N:=0; A.C:=1; A.L:=0;
  repeat
  inc(N);
  A.C:=A.C*N;
  normal(A);
  until Comp(A,F);
  writeln(N);
  close(Fd);
end.
```

Наступна задача теж використовує методи «довгої арифметики».

ДВІЙКИ ТА П'ЯТІРКИ, АБО ПРОДОВЖЕННЯ «ДОВГОЇ АРИФМЕТИКИ»

(О-2002)

У вхідному файлі записане натуральне число, яке є значенням виразу $2^K \cdot 5^N$ при деяких цілих K і N ($0 \le K \le 10\,000$, $0 \le N \le 10\,000$). Знайдіть ці значення K і N та запишіть їх у файл.

input.txt	2000	
output.txt	4 3	Значення K і N

Розв'язання

Очевидне розв'язання цієї задачі можна запропонувати, використавши стандартний цілий тип **LongInt**. Зрозумілий алгоритм:

1) ділимо на 2 (поки ділиться), обчислюємо в K, скільки разів поділили;

2) ділимо на 5 (поки ділиться), обчислюємо в N, скільки разів поділили;
3) якщо журі нічого не переплутало, маємо отримати одиницю.

Ось програма:

```
const
  Fi = 'input.txt';
  Fo = 'output.txt';
var
  A:LongInt;
  N,K: Word;
  Fd,Fs:Text;
Procedure Init;
begin
  assign(Fd,Fi);
  reset(Fd);
  readln(Fd,A);
  close(Fd);
end;
Procedure Run;
begin
  K:=0; N:=0;
  while (A mod 2 = 0) do begin
    A:=A div 2; inc(K);
end;
  while (A mod 5 = 0) do begin
  A:=A div 5; inc(N);
  end;
end;
Procedure Done;
begin
  assign (Fs,Fo);
  rewrite(Fs);
  writeln(Fs,K,' ',N);
  close(Fs);
end;
begin
  Init; Run; Done;
end.
```

Така програма пройшла б на олімпіаді чотири тести та набрала лише 40 % від загальної кількості балів. Справа в обмеженнях для значень K і N ($0 \le K \le 10000, 0 \le N \le 10000$), тобто по максимуму десять тисяч цифр. Маємо справу з «довгою арифметикою», але будемо міркувати. Якщо просто розширити попередній алгоритм процедурами, що обслуговують арифметичні операції з довгими числами, то нам потрібно емулювати ділення на ціле число, а це не так просто.

Ділити просто лише на десять, зменшивши на одиницю кількість цифр числа, — і все. Тоді замість ділення на 2 потрібно множити на 5 і ділити на 10, а в разі ділення на 5 потрібно множити на 2 і знову ділити на 10.

Ще одна евристика: розглянемо число вигляду $2^K \cdot 5^N$, нехай буде $16\,000 = 2^7 \cdot 5^3$. Чому так багато нулів? Справді, кожна пара множників 2 та 5 утворює множник 10, і тому, порахувавши нулі в хвості числа, будемо знати менше з чисел K і N. Число, що залишилося, має множником або тільки 2, або тільки 5. Достатньо подивитися на цифру наймолодшого розряду, а далі ділимо до одиниці, щоб знайти найбільше з чисел K і N.

Тут пропонується текст програми, яка проходить усі запропоновані тести на 100 %:

```
Uses Crt;
const
  Fi = 'input.txt';
  Fo = 'output.txt';
  Max= 10000; {byteаксимальна кількість цифр}
Type
  Tnum = Record {Знайомий запис числа}
  P:Word;
  M:Фrray [0..Max] of Byte;
end;

var
  A,B:Tnum;
  N,K: Word;
  Fd,Fs:Text;

Procedure Init;
var
  i:Word;
  c:Char;
begin
  assign(Fd,Fi);
  reset(Fd);
  i:=0;
  repeat
  read(Fd,c); inc(i);
  B.M[i]:=Ord(c)-48;
  until EoLn(Fd) or EoF(Fd);
  B.P:=i;
  close(Fd);
end;
```

```
Procedure Mult (C:Byte; var A:Tnum);
var
  i,R:Word; {Процедура множить A:Tnum на C,}
  B:Tnum;
  {результат враховує зсув на 1 розряд}
begin {вліво, тобто симулюємо ділення на 10}
  FillChar(B.M,Max,0);
  for i:=1 to A.P do begin
    R:=B.M[i-1]+A.M[i]*C;
    B.M[i-1]:=R mod 10;
    B.M[i] :=R div 10;
  end;
  if B.M[A.P]>0 then B.P:=A.P
    else B.P:=A.P-1;
  A:=B;
end;

Procedure Run;
var
  i,j,l,h:Word;
  C : Byte;
begin
  l:=B.P;
  while B.M[l]=0 do dec(l);
  {Рахуємо і видаляємо нулі}
  h:=B.P-l;
  if Odd(B.M[l]) then C:=2 else C:=5;
  FillChar(A.M,Max,0);
  {Визначаємо множник, що лишився}
  j:=l; i:=0;
  repeat
    inc(i);{Переписуємо цифри з B в A,змінивши}
    A.M[i]:=B.M[j]; {Їх порядок на протилежний}
    Dec(j);
  until j=0;
  A.P:=l;
  i:=0; {Поки не 1, ділимо на 10}
while not ((A.P=1) and (A.M[1]=1)) do
  begin
    Mult(C,A); inc(i)
  end;
if C=5 then begin K:=h+i; N:=h end
  else begin N:=h+i; K:=h end;
  {Формуємо значення K і N }
```

```
end;
Procedure Done;
begin
  assign (Fs,Fo);
  rewrite(Fs);
  writeln(Fs,K,' ',N);
  close(Fs);
end;

begin
  Init; Run; Done;
end.
```

НАЙБІЛЬШИЙ СТЕПІНЬ ДВІЙКИ

(О-2005)

У файлі, розмір якого не перевищує 1 Кб, записані випадкові цифри 0...9 без пропусків. Який найбільший степінь двійки (тобто число 2^N) можна утворити, вилучивши деякі з цифр? Знайдіть значення N.

input.txt	6571825	Задане число
output.txt	9	Значення N

Пояснення

$6571825 \to 512 = 2^9$, тобто $N = 9$.

Розв'язання

І знову степінь двійки, хоча цього разу маємо просто технічну задачу, що розв'язується елементарними методами. Отже, перевіримо техніку.

Дані з файла читаємо в змінну C типу **Tnum**, точніше цифри в оберненому порядку заносимо до масиву $C.M$, а їх кількість у $C.P$. У змінній A типу **Tnum** формуємо найбільший степінь двійки, що має кількість цифр більше ніж $C.P$.

Одна ітерація цикла пошуку складається з таких дій:

1) отримуємо в змінній A попередній степінь двійки, це можна зробити у вже відомий спосіб — значення A помножити на 5 і закреслити 0 у найменшому розряді;
2) виконуємо перевірку, чи можна отримати цифри масиву $A.M$, видаливши деякі елементи масиву $C.M$.

У випадку успіху пошук закінчується, значення відповідного показника степеня двійки буде в змінній N, якщо ж відповіді немає, то значення N буде 1.

Програма легко прочитається і без коментарів.

```
const
  Fi = 'input.txt';
  Fo = 'output.txt';
  _p = 1025;
Type
  Tnum= Record
  P:Integer;
  M:Array [0.._p] of Byte;
end;
var
  A,C: Tnum;
  N : Integer;
Procedure Swap (var x,y:Byte);
var z:Byte;
begin
  z:=x; x:=y; y:=z;
end;
Procedure Init;
var
  f:Text;
  w:Char;
  i:Integer;
begin
  assign(f,Fi);
  reset(f);
  C.P:=0;
  repeat
  inc(C.P);
  read(f,w);
  C.M[C.P]:=ord(w)-48;
  until Eof(f) or EoLn(f);
  close(f);
  for i:=1 to C.P div 2 do Swap(C.M[i],C.M[C.P-i+1]);
end;
Procedure Mult (var X:Tnum; q:Byte);
var
  i,l,r:Integer;
  Y:Tnum;
begin
  for i:=0 to _p do Y.M[i]:=0;
  for i:=1 to X.P do begin
    r :=Y.M[i]+X.M[i]*q;
    Y.M[ i ] :=r mod 10;
    Y.M[i+1] :=r div 10;
  end;
  if Y.M[X.P+1]>0 then Y.P:=X.P+1
    else Y.P:=X.P;
```

```
  X:=Y;
end;
Procedure Muld (var X:Tnum; q:Byte);
var
  i,l,r:Integer;
  Y:Tnum;
begin
  for i:=0 to _p do Y.M[i]:=0;
  for i:=1 to X.P do begin
    r :=Y.M[i-1]+X.M[i]*q;
    Y.M[i-1] := r mod 10;
    Y.M[ i ] := r div 10;
  end;
  if Y.M[X.P]>0 then Y.P:=X.P
    else Y.P:=X.P-1;
  X:=Y;
end;
Procedure Run;
var
  i,j:Integer;
  w:Boolean;
begin
  A.P:=1; A.M[1]:=1;
  N:=0;
  repeat
  inc(N);
  mult(A,2);
  until A.P > C.P;
  repeat
  Dec(N);
  if N=-1 then break;
  Muld (A,5);
  i:=A.P;
  j:=C.P;
  repeat
  if A.M[i]=C.M[j] then dec(i);
  dec(j);
  until (i=0) or (j=0);
  until i=0;
  end;
Procedure Done;
var  f:Text;
begin
  assign(f,Fo);
  rewrite(f);
```

```
    writeln(f,N);
    close(f);
  end;
  begin
    Init; Run; Done;
  end.
```

N-ЗНАЧНІ ЧИСЛА

(О-2003)

Обчислити кількість N-значних натуральних чисел, сума цифр кожного з яких дорівнює M ($1 \le N \le 100$, $1 \le M \le 900$).

Приклад введення та виведення

input.txt	2 10	Значення N і M
output.txt	9	Кількість N-значних чисел

Розв'язання

Не будемо демонструвати перебірний розв'язок — у будь-якому випадку для його реалізації на максимальних даних не вистачить реального часу.

Спочатку розглянемо таблицю результатів для значень $N = 1...3$, $M = 1...27$.

1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
2	1	2	3	4	5	6	7	8	9	9	8	7	6	5	4	3	2	1
3	1	3	6	10	15	21	28	36	45	54	61	66	69	70	69	66	61	54

Вочевидь, для $N = 1$ чисел з відповідною сумою цифр M ($1 \le M \le 9$) буде по одному для кожної суми (перший рядок таблиці). Результати для більших N можна отримати рекуррентно:

`A[N,M]:= A[N-1,M]+A[N,M-1]` з поправкою у випадку $M > 10$ (тоді результат необхідно зменшити таким чином: $A[N-1, M-10]$). Фрагмент логіки:

```
Procedure Solut;
var
  i,j:Integer;
  A:Array [0..10,0..90] of LongInt;
begin
  for i:=1 to 9 do A[1,i]:=1;
  for j:=2 to N do
  for i:=1 to 9*j do begin
    A[j,i]:=A[j-1,i]+A[j,i-1];
    if i>10 then A[j,i]:=A[j,i]-A[j-1,i-10];
  end;
end.
```

Кінцева версія програми має використовувати довгу арифметику, оскільки стандартного типу **LongInt** вистачить щонайбільше для $N=10$. Відповідні процедури довгої арифметики **InitTnum**, **Add** (віднімання), **Sub** (додавання) ми розглянули раніше, тому текст програми пропонується без пояснень. На відміну від попереднього фрагмента, де для збереження результатів була використана матриця, тут використовуються два масиви C — вже знайдені результати у $(N-1)$-му рядку матриці, A — поточний результат (N-й рядок матриці).

```
const
  Fi = 'input.txt';
  Fo = 'output.txt';
  maxN = 100;
Type
  Tnum = Record
  P:Byte;
  M:Array [1..maxN] of ShortInt;
  end;
  Pnum = ^Tnum;
var
  N,M:LongInt;
  A,C: array [0..maxN*9] of Pnum;
  FD,FS:Text;
Procedure Init;
begin
  assign(FD,Fi);
  reset(FD);
  readln(FD,N,M);
  close(FD);
end;
Procedure Add (X,Y:Tnum; var Z:Tnum);
var i,l,q:Word;
begin
  FillChar(Z.M,100,0);
  if X.P>Y.P then l:=X.P else l:=Y.P;
  for i:=1 to l do begin
    q :=Z.M[i]+X.M[i]+Y.M[i];
    Z.M[ i ] := q mod 10;
    Z.M[i+1] := q div 10;
  end;
  if Z.M[l+1]>0 then Z.P:=l+1
    else Z.P:=l;
end;
Procedure Sub (X,Y:Tnum; var Z:Tnum);
var
```

```
  i,l:Word;
  q : ShortInt;
begin
  FillChar(Z.M,100,0);
  if X.P>Y.P then l:=X.P else l:=Y.P;
  for i:=1 to l do begin
    q := Z.M[i]+X.M[i]-Y.M[i];
    if q<0 then begin
      Z.M[ i ] := q + 10;
      Z.M[i+1] := -1;
    end
    else Z.M[ i ] := q;
  end;
  while Z.M[l]=0 do Dec(l);
  Z.P:=l;
end;
Procedure InitTnum (var B:Pnum);
var i:Byte;
begin
  GetMem (B,SizeOf(Tnum));
  for i:=1 to maxN do B^.M[i]:=0;
  B^.P:=0;
end;

Procedure Run;
var i,j : Word;
begin
  for i:=0 to MaxN*9 do InitTnum(A[i]);
    for i:=0 to MaxN*9 do InitTnum(C[i]);
      for i:=1 to 9 do begin C[i]^.M[1]:=1;
  C[i]^.P:=1 end;
    for j:=2 to N do begin
      for i:=1 to 9*j do begin
        Add(C[i]^,A[i-1]^,A[i]^);
        if i>10 then Sub(A[i]^,C[i-10]^,A[i]^);
      end;
      for i:=1 to j*9 do C[i]^:=A[i]^;
    end;
End;
Procedure Done;
var i :Word;
begin
  assign (FS,Fo);
  rewrite(FS);
  for i:=C[M]^.P downto 1 do write(FS,C[M]^.M[i]);
  writeln(FS);
```

```
  close(FS);
  for i:=0 to maxN*9 do FreeMem (A[i],SizeOf(Tnum));
  for i:=0 to maxN*9 do FreeMem (C[i],SizeOf(Tnum));
end;
begin
  Init; Run; Done;
end.
```

ОДИНИЦІ

(О-2008)

В арифметичному виразі дозволяється використовувати число 1, операції додавання і множення та дужки. Яку мінімальну кількість одиниць потрібно використати, щоб отримати задане натуральне число N $(1 \le N \le 5000)$?

input.txt	7	Значення N
output.txt	6	Мінімальна кількість одиниць

Розв'язання

Методи, які ми застосуємо для розв'язування цієї задачі, мають в інформатиці назву «динамічне програмування». Вочевидь, найбільш доцільна саме динамічна модель, тобто для того, щоб знайти відповідь для N, спочатку необхідно знайти найменшу кількість одиниць для значень $1 \ldots N-1$. Дослідимо відповідні вирази для декількох початкових значень N і занесемо результати до таблиці.

Значення N	Найменша кількість 1	Відповідний арифметичний вираз
1	1	1
2	2	$1+1$
3	3	$1+1+1$
4	4	$(1+1)\cdot(1+1)$
5	5	$1+(1+1)\cdot(1+1)$
6	5	$(1+1)\cdot(1+1+1)$
7	6	$1+(1+1)\cdot(1+1+1)$
8	6	$(1+1)\cdot(1+1)\cdot(1+1)$
9	6	$(1+1+1)\cdot(1+1+1)$
10	7	$(1+1)\cdot(1+(1+1)\cdot(1+1))$
11	8	$1+(1+1)\cdot(1+(1+1)\cdot(1+1))$
12	7	$(1+1)\cdot(1+1)\cdot(1+1+1)$
13	8	$1+(1+1)\cdot(1+1)\cdot(1+1+1)$

14	8	(1 + 1) · (1 + (1 + 1) · (1 + 1 + 1))
15	8	(1 + 1 + 1) · (1 + (1 + 1) · (1 + 1))
16	8	(1 + 1) · (1 + 1) · (1 + 1) · (1 + 1)
17	9	1 + (1 + 1) · (1 + 1) · (1 + 1) · (1 + 1)
18	8	(1 + 1) · (1 + 1 + 1) · (1 + 1 + 1)
19	9	1 + (1 + 1) · (1 + 1 + 1) · (1 + 1 + 1)
20	9	(1 + 1) · (1 + 1) · (1 + (1 + 1) · (1 + 1))
21	9	(1+1+1) · (1+(1+1) · (1+1+1))
22	10	(1 + 1) · (1 + (1 + 1) · (1 + (1 + 1) · (1 + 1)))
23	11	1 + (1 + 1) · (1 + (1 + 1) · (1 + (1 + 1) · (1 + 1)))
24	9	(1 + 1) · (1 + 1) · (1 + 1) · (1 + 1 + 1)
25	10	(1 + (1 + 1) · (1 + 1)) · (1 + (1 + 1) · (1 + 1))
26	10	(1 + 1) · (1 + (1 + 1) · (1 + 1) · (1 + 1 + 1))
27	9	(1 + 1 + 1) · (1 + 1 + 1) · (1 + 1 + 1)
28	10	(1 + 1) · (1 + 1) · (1 + (1 + 1) · (1 + 1 + 1))
29	11	1 + (1 + 1) · (1 + 1) · (1 + (1 + 1) · (1 + 1 + 1))
30	10	(1 + 1) · (1 + 1 + 1) · (1 + (1 + 1) · (1 + 1))
31	11	1 + (1 + 1) · (1 + 1 + 1) · (1 + (1 + 1) · (1 + 1))
32	10	(1 + 1) · (1 + 1) · (1 + 1) · (1 + 1) · (1 + 1)
33	11	(1 + 1 + 1) · (1 + (1 + 1) · (1 + (1 + 1) · (1 + 1)))
34	11	(1 + 1) · (1 + (1 + 1) · (1 + 1) · (1 + 1) · (1 + 1))
35	11	(1 + (1 + 1) · (1 + 1)) · (1+ (1 + 1) · (1 + 1 + 1))
36	10	(1 + 1) · (1 + 1) · (1 + 1 + 1) · (1 + 1 + 1)
37	11	1 + (1 + 1) · (1 + 1) · (1 + 1 + 1) · (1 + 1 + 1)
38	11	(1 + 1) · (1 + (1 + 1) · (1 + 1 + 1) · (1 + 1 + 1))
39	11	(1 + 1 + 1) · (1 + (1 + 1) · (1 + 1) · (1 + 1 + 1))
40	11	(1 + 1) · (1 + 1) · (1 + 1) · (1 + (1 + 1) · (1 + 1))
41	12	1 + (1 + 1) · (1 + 1) · (1 + 1) · (1 + (1 + 1) · (1 + 1))

Динамічна модель буде побудована таким чином:

- для $N = 1$ кількість одиниць буде $x[1] = 1$;
- кожне наступне $x[N]$ — це буде найменше з таких двох величин:

  $a = x[N-1] + 1$;

  $b = \min(x[i] + x[N \operatorname{div} i])$, для всіх дільників і числа N, більших за одиницю.

Нижче пропонується програма, яка обчислює лише мінімальну кількість одиниць. Спробуйте вдосконалити її, щоб для заданого N виводився ще й арифметичний вираз із третього стовпця таблиці.

```
const
  Fi = 'input.txt';
  Fo = 'output.txt';
  _N = 5000;
var
  N:Integer;
  X:Array [1.._N] of Byte;
  fD,fS:Text;
Procedure Init;
begin
  assign(fD,FileIn);
  reset(fD);
  readln(fD,N);
  close(fD);
end;
Procedure Run;
var i,a,b,m:Word;
begin
  FillChar(X,SizeOf(X),0);
  X[1]:=1;
  for M:=2 to N do begin
    a:=N;
    for i:=1 to M div 2 do
      if X[i]+X[M-i]< a then a:=X[i]+X[M-i];
    b:=N;
    for i:=2 to M div 2 do
      if (M mod i=0) and (X[i]+X[M div i]<b) then
    b:=X[i]+X[M div i];
    if a<b then X[M]:=a else X[M]:=b;
  end;
end;
Procedure Done;
begin
  assign (Fs,FileOut);
  rewrite(Fs);
  writeln(Fs,X[N]);
  close(Fs);
end;
begin
  Init; Run; Done
end.
```

ПІДНЕСЕННЯ ДО СТЕПЕНЯ

(О-2000)

Інтерпретатор деякої мови програмування вміє виконувати присвоювання типу `A: = B * C` (*A*, *B*, *C* — позначають імена будь-

яких змінних), але не вміє виконувати операцію піднесення до натурального степеня. Тому обчислення виразу типу $A \hat{} N$ (A в степені N) можна замінити серією команд присвоювання з операцією множення.

Наприклад, команду $X:=A^5$ можна записати серією з трьох команд:

```
R1:= A*A;
R2:= A*R1;
X:=R1*R2;
```

За заданим N ($2 \le N \le 2000$) знайдіть мінімальну кількість команд присвоювання з одним множенням у кожній для обчилення A^N.

input.txt	9	Значення N
output.txt	4	Мінімальна кількість команд

Розв'язання

Ця проблема є достатньо цікавою, тому опускаємо звичне повідомлення про очевидність задачі. Елементарне розв'язання дає програма, яка читає з файла число N та виводить у файл відповідь — число $N-1$. Звичайно, коли в першій команді одержати 2-й степінь (`R1:=A*A`), у другій — 3-й степінь (`R2:=A*R1`), то в ($N-1$)-й команді отримаємо N-й степінь. Таке роз'язання складається з максимальної кількості команд, і тому не може нас влаштовувати. Хоча, отримуючи на кожному етапі найменший можливий степінь, ми маємо максимальну кількість команд, то логічно, що в разі утворення в кожній з команд якомога більших степенів кількість команд буде мінімальною (або близькою до мінімальної).

Пропонувати алгоритм або навіть метод роз'язування задачі заранo, тому приступаємо до пошуку евристик, дослідивши набори команд для обчислення степенів, не більших за 10.

Показник степеня N	**Серія команд**	**Серія степенів**
2	`X:=A*A`	2
3	`R1:=A*A` `X:=A*R1`	2 3
4	`R1:=A*A` `X:=R1*R1`	2 4

5	`R1:=A*A`
	`R2:=R1*R1`
	`X:=R2*A`
		2
		4
		5

6	`R1:=A*A`
	`R2:=R1*R1`
	`X:=R1*R2`
		2
		4
		6

Показник степеня N	Серія команд	Серія степенів
7	R1:=A*A R2:=R1*R1 R3:=R1*R1 X:=A*R3	2 4 6 7
8	R1:=A*A R2:=R1*R1 X:=R2*R2	2 4 8
9	R1:=A*A R2:=R1*R1 R3:=R2*R2 X:=A*R3	2 4 8 9
10	R1:=A*A R2:=R1*R1 R3:=R2*R2 X:=R1*R3	2 4 8 10

Записуючи команди, дотримуватимемось сформульованого вище правила щодо отримання в кожній команді якомога вищого степеня. Якщо уважно розглянути таблицю, а саме її третій стовпець, дійдемо очевидних висновків.

1. Оскільки добуток степенів з однаковою основою дорівнює степеню з тією ж основою і показником, що дорівнює сумі показників степенів, то від операції множення відразу перейдемо до звичайного додавання степенів.
2. Кодувати показники степенів у серії команд достатньо лінійним масивом цілих чисел, де послідовно фіксувати показники степенів, отриманих у кожній наступній команді (нульовим елементом масиву буде 1).
3. Кожна наступна команда формується як сума двох раніше обчислених степенів так, щоб отримати якомога більший (але не більший за потрібний) степінь.

Алгоритм розв'язування задачі буде більш зрозумілим, якщо переформулювати умову таким чином.

У масиві цілих цисел X нульовий елемент дорівнює 1, необхідно сформувати наступні елементи так, щоб за мінімальну кількість кроків отримати останнім елементом масиву задане ціле число N. Наступний елемент масиву утворюється як сума двох уже заповнених елементів масиву.

Насамперед, випробуємо метод «жадібного» перебору для роз'язання цієї проблеми. Дещо конкретизуємо алгоритм:

1) ініціалізувати довжину та нульовий елемент масиву X;
2) почати перегляд масиву X з останього (найбільшого) елемента;
3) рухаючись назад по масиву, знайти такий поточний елемент, щоб він у сумі з останнім не перевищував N;
4) знайдену суму занести наступним елементом масиву X;
5) якщо останнім елементом X не з'явилося число N, виконати пункт 2).

У запропонованій нижче програмі цей алгоритм реалізовано у функції **Select**, яка для заданого N повертає відповідну кількість операцій.

```
var
  N:Word;  {Задане ціле число}
  C:Word;  {Шукана кількість операцій}
  X:Array [0..100] of Word;
  {Допоміжний масив}
  i,K: Byte;
  {Кількість та номер поточного тесту}
  Fi,Fo : Text; {Файлові змінні}
Function Select (N:Word):Word;
var i,j:Word;
begin
  X[0]:=1; j:=0;
  {Ініціалізуємо довжину та нульовий елемент}
  repeat
  i:=j;
  {Перегляд з кінця масиву; рухаючись назад}
  {Шукаємо суму елементів, яка не перевищує N}
  while X[i]+X[j]>N do Dec(i);
  inc(j); X[j]:=X[i]+X[j-1];
  {Заносимо суму останнім елементом}
  until N=X[j];
  {Пошук продовжуємо,поки не з'явиться значення N}
  Select:=j;
end;
begin {Головний модуль програми}
  assign(Fi,'input.txt');
  reset(Fi);
  readln(Fi,N);
  close(Fi);
  assign (Fo, 'output.txt');
  rewrite(Fo);
```

```
    writeln(Fo, Select(N));
    close(Fo);
  end.
```

Безумовно, «жадібний» алгоритм дає достатньо непоганий результат, але чи є цей результат мінімальним?

Спробуймо дослідити послідовності чисел для утворення деяких степенів. Оптимальний «жадібний» ланцюжок для числа 9: 2 4 8 9 (4 команди), але виявляється, що існують й інші оптимальні послідовності:

2 4 5 9

2 3 6 9 (кожного разу за 4 команди).

Степінь 9 може утворюватися з 5, 6 і 8 кожного разу за найменшу кількість операцій. Аналогічна ситуація для показника степеня 10.

Маємо:

2 4 8 10 («жадібний» алгоритм)

2 3 5 10

2 4 6 10 (кожного разу за 4 команди).

Отже, оптимальні набори команд можна отримати не тільки «жадібним» алгоритмом. Існують інші послідовності степенів, які врешті-решт можуть виявитися коротшими від знайдених.

Першим показником степеня, який заперечує «жадібний» алгоритм, є число 15. Виявляється, що цей степінь можна отримати лише за п'ять команд, тоді як «жадібний» ланцюжок 2 4 8 12 14 15 дає шість команд.

Оптимальні послідовності:

2 3 6 9 15

2 3 5 10 15 (кожного разу за 5 команд).

Аналогічні результати будуть для показників степенів 23, 27, 30, 31, 39 тощо, тому шукатимемо нові підходи для побудови відповідної мінімальної послідовності степенів.

Очевидно, однією командою можна утворити тільки другий степінь, тому за дві — тільки показники 3 і 4. На наступному етапі (тобто за три команди), можна отримати 5, 6, 8. Династію утворених степенів можна зобразити у вигляді деревоподібної структури. Очевидно, що перша поява показника степеня в цій структурі і буде мінімальною (за кількістю використаних команд). До кожного числа від вершини дерева буде вести ланцюжок чисел, які мають бути проміжними степенями для утворення шуканого степеня. Порядок вершини в цьому дереві дорівнює кількості команд, використаних для утворення заданого степеня (вершина 1 має порядок 0, вершина 2 — порядок 1, вершини 3, 4 — 2 тощо).

						1					
						2					
				3				4			
		5				6			8		
7		10				9				16	
14		13	20		11	15	18		17		32

Зрозуміло, що, сформувавши таку деревоподібну структуру за допомогою програми, ми отримаємо оптимальний результат, що дорівнює порядку вершини в такому дереві. Зобразимо деревоподібну структуру більш звичним способом, запам'ятовуючи для кожної вершини її порядок та предка (номер показника степеня, з якого утворилася ця вершина).

Маємо таблицю:

Степінь	**Предок**	**Порядок**	**Ланцюжок**
1	0	0	1
2	1	1	1 2
3	2	2	1 2 3
4	2	2	1 2 4
5	3	3	1 2 3 5
6	3	3	1 2 3 6
7	4	4	1 2 3 5 7
8	3	3	1 2 4 8
9	4	4	1 2 3 6 9
10	4	4	1 2 3 5 10
11	5	5	1 2 3 6 9 11
12	4	4	1 2 3 6 12
13	5	5	1 2 3 5 10 13
14	5	5	1 2 3 5 7 14
15	5	5	1 2 3 6 9 15
16	4	4	1 2 4 8 16
17	5	5	1 2 4 8 16 17
18	5	5	1 2 3 6 9 18
19	6	6	1 2 3 5 7 14 19
20	5	5	1 2 3 5 10 20
21	6	6	1 2 3 5 7 14 21
22	6	6	1 2 3 6 9 11 22

Степінь	Предок	Порядок	Ланцюжок
23	6	6	1 2 3 5 10 13 23
24	5	5	1 2 3 6 12 24
25	6	6	1 2 4 8 16 17 25
26	6	6	1 2 3 5 10 13 26
27	6	6	1 2 3 6 9 18 27
28	6	6	1 2 3 5 7 14 28
29	7	7	1 2 4 8 16 17 25 29
30	6	6	1 2 3 6 9 15 30

Заносити дані до такої таблиці можна, керуючись чергою або рекурсію. В такому випадку може виявитися проблематичним зберігати в пам'яті програми відповідні ланцюжки чисел для формування степенів.

Тому зупинимося на методі динамічного програмування, прямим завданням якого і є формування відповідної таблиці. Для знаходження оптимальної кількості команд для заданого M спочатку формуємо оптимальні розв'язки для степенів від 2 до $M-1$. Кожен наступний степінь N формуємо з утворених ланцюжків степенів, додавши до ланцюжка одне число, яке дорівнює сумі двох членів ланцюжка (останнього і ще якогось), при цьому базовим обирається ланцюжок із мінімальною кількістю використаних операцій.

Алгоритм, що реалізує метод динамічного програмування, буде таким:

1) ініціалізувати перший рядок таблиці;
2) пункти 3) і 4) виконати для степенів N від 2 до M;
3) за сформованими записами в таблиці (від 1 до $N-1$) розглядаємо базові ланцюжки для поточного N та вибираємо з них мінімальний за кількістю використаних команд;
4) занести в таблицю інформацію про знайдений базовий запис для поточного N, скоригувавши при цьому кількість використаних команд та номер предка.

Виявляється, у таблиці достатньо пам'ятати тільки поля кількості використаних команд і номер предка, знаючи якого можна однозначно відтворити ланцюжок попередніх степенів.

Залишилося лише навести текст програми, яка читає блок степенів з файла та виводить відповідні кількості команд для кожного степеня на екран. Під час зчитування з файла значення показників степеня заносяться до масиву X, паралельно визначається значення M — найбільший показник степеня в блоці даних. Описана вище таблиця будується для всіх N від 2 до M. Використовуючи елементи масиву X як індекси, обчислюємо відповідні кількості команд для кожного степеня з файла.

```
const
  MaxM = 2000;
  Fi = 'input.txt';
  Fo = 'output.txt';
type
  Textent = Record
  D:Byte;
  P:Word;
  end;
var
  M:Word;
  K:Byte;
  Z:Array [1..MaxM] of Textent;
  X:Array [1..100] of Word;
Procedure Init;
var
  f:Text;
  i:Byte;
begin
  assign(f,Fi);
  reset(f);
  readln(f,M);
  close(f);
end;
Function Test (W,A:Word):Boolean;
var B:Word;
begin
  B:=A;
  while (A+B > W) and (B > 0) do B:=Z[B].P;
  Test:=(A+B=W);
end;
Procedure Run;
var h,q,n,p,i:Word;
begin
  Z[1].D:=0; Z[1].P:=0;
  for n:=2 to m do begin
    h:=MaxM; q:=1;
    for p:=n div 2 to n-1 do
    if ((Z[p].D+1)<h) and Test(n,p) then begin
      h:=Z[p].D+1; q:=p
    end;
    Z[n].D:=h;
    Z[n].P:=q;
  end;
end;
```

```
Procedure Done;
var
  f:Text;
  i:Byte;
begin
  assign(f,Fo);
  rewrite(f);
  writeln(f,Z[M].D);
  close(f);
end;
begin
  Init; Run; Done;
end.
```

ТУК-ТУК, АБО МАТРИЦЯ З МНОГОКУТНИКІВ

(Р-2003)

На координатній площині задано N прямокутників — кожен парою протилежних вершин, сторони яких паралельні осям координат, а координати вершин — цілі числа з проміжку [– 20, 20]. Яку найбільшу кількість прямокутників можна прибити до площини одним цвяхом? Прямокутник вважається прибитим, якщо цвях забито у внутрішню точку прямокутника.

Розв'язання

Ця задача має за мету продемонструвати зв'язок між координатною площиною і матрицею. Хоча існує достатня кількість задач з аналітичної геометрії та двовимірних масивів (матриць), цим «цвяхом» ми зможемо «прибити» відразу два розділи.

Кожен елемент такої матриці обслуговує одиничну клітинку площини, його значення буде дорівнювати кількості прямокутників, яким належить відповідна клітинка.

Проілюструємо сказане рисунком, якщо на площині задано три прямокутники з такими координатами X і Y протилежних вершин: 1 1 6 5, 3 2 7 4, 2 3 4 6 (див. рис.):

6	0	0	0	0	0	0	0	0
5	0	0	1	1	0	0	0	0
4	0	1	2	2	1	1	0	0
3	0	1	2	3	2	2	1	0
2	0	1	1	2	2	2	1	0
1	0	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6	7

Декілька технічних зауважень:

- координати прямокутника для зручності можна оголосити своїм типом — запис із чотирьох полів;
- прочитавши з файла координати двох протилежних вершин, необхідно впорядкувати їх так, щоб прямокутник задавався нижньою лівою і верхньою правою вершинами;
- індексами відповідної клітинки в матриці буде нижня ліва координата клітинки на площині, тобто прямокутник із координатами L, D, R, U буде покривати діапазон $L \ldots R-1$ стовпців і $D \ldots U-1$ рядків у матриці;
- ініціалізувати елементи матриці потрібно 0, а потім для кожного поточного прямокутника збільшувати на 1 значення елементів матриці, які він покрив.

```
const
  _n=20;
  Fi='input.txt';
  Fo='output.txt';
Type
  TBar=Record
  L,D,R,U:Integer;
end;
var
  N,T,C:Byte;
  M:Array [-_n.._n,-_n.._n] of Integer;
  FD,FS:Text;
  {процедура впорядковує координати прямокутника
  A.L, A.D, A.R, A.U }
Procedure SortBar (var A:TBar);
var P:Integer;
begin
  if A.L>A.R then begin P:=A.R; A.R:=A.L; A.L:=P end;
  if A.D>A.U then begin P:=A.D; A.D:=A.U; A.U:=P end;
end;
{Процедура заповнює відповідні елементи матриці, які
покрив прямокутник A}
Procedure InCells (A:TBar); var i,j:Byte;
begin
  for i:=A.D to A.U-1 do
  for j:=A.L to A.R-1 do inc(M[i,j]);
end;
Procedure Init;
var
  i:Integer;
  B:Tbar;
begin
  assign(FD,Fi);
  reset(FD);
  readln(FD,C);
```

```
  FillChar(M,SizeOf(M),0);
  for i:=1 to C do begin
    readln(FD,B.L,B.D,B.R,B.U);
    SortBar(B);
    InCells(B);
  end;
  close(FD);
end;
Procedure Run;
var i,j : Integer;
begin
  T:=0;
  for i:=-_n to _n do {Знаходимо в максимум T}
  for j:=-_n to _n do
    if M[i,j]>T then T:=M[i,j];
  end;
Procedure Done;
var i,j : Byte;
begin
  assign (FS,Fo);
  rewrite(FS);
  writeln(FS,T);
  close (FS);
end;
begin
  Init; Run; Done;
end.
```

АРКУШ У КЛІТИНКУ

(О-2003)

На квадратному аркуші паперу розміром $N \times N$ клітинок накреслили K різних прямокутників зі сторонами на лініях сітки. Знаючи координати пар протилежних вершин кожного прямокутника у прямокутній системі координат, початок якої лежить в одній із крайніх точок аркуша, а осі збігаються із його сторонами, обчислити, на скільки частин розпадеться аркуш паперу, якщо зробити розрізи по кожній зі сторін прямокутника.

Усі числові значення цілі невід'ємні, не більші за 100.

input.txt	8 4 1 1 6 5 1 2 8 4 2 3 4 8 0 5 7 7	Значення N і K K рядків: координати пар протилежних вершин прямокутників
output.txt	14	Кількість частин аркушу паперу

Розв'язання

Задача, на перший погляд, схожа на попередню. Але це тільки на перший погляд — не той рівень.

Вочевидь, не розпадуться ті частинки аркуша, які не перетинає жодна зі сторін якогось прямокутника. Спочатку відмітимо клітинки матриці у схожий спосіб, але щоб уникнути повторів, діятимемо так:

1) ініціалізуємо елементи матриці нулями та враховуємо за перший прямокутник весь аркуш;
2) читаємо з файла поточний прямокутник;
3) знаходимо в матриці найбільше значення, потім для кожного поточного прямокутника збільшуємо на це значення елементи матриці, які він покрив;
4) обробивши всі прямокутники отримаємо матрицю, зображену на рисунку.

8	1	1	9	9	1	1	1	1
7	17	17	25	25	17	17	17	1
6	17	17	25	25	17	17	17	1
5	1	3	11	11	3	3	1	1
4	1	7	15	15	7	7	5	5
3	1	7	7	7	7	7	5	5
2	1	3	3	3	3	3	1	1
1	1	1	1	1	1	1	1	1
	1	2	3	4	5	6	7	8

Для розв'язування задачі залишається порахувати кількість зв'язаних областей у матриці, відмічених однаковим числом. Для цього згодиться будь-який алгоритм зафарбування. В наведеній нижче програмі він базується на рекурсії, хоча з метою безпеки можна запропонувати щось інше.

```
const
  u:Array [1..4] of Integer = (0,1,0,-1);
  v:Array [1..4] of Integer = (1,0,-1,0);
  Fi='input.txt';
  Fo='output.txt';
  maxN=100;
  maxB=100;
Type
  TBar = Record
  L,D,R,U:Integer;
```

```
  end;
  var  kB,kS:Word;
  MM,RR:Longint;
  N:Byte;
  M:Array [0..maxN,0..maxN] of LongInt;
  FD,FS:Text;
Procedure SortBar(var A:TBar);
var P:Integer;
begin
  if A.L>A.R then begin P:=A.R; A.R:=A.L; A.L:=P end;
  if A.D>A.U then begin P:=A.D; A.D:=A.U; A.U:=P end;
end;
Procedure InCells (A:TBar);
var
  i,j:Byte;
  p:LongInt;
begin
  p:=MM+1;
  for i:=A.D+1 to A.U do
  for j:=A.L+1 to A.R do begin
    M[i,j]:=M[i,j] + p;
    if M[i,j]>MM then MM:=M[i,j];
  end;
end;
Procedure Init;
var
  i,j:Byte;
  fD:Text;
  B:Tbar;
begin
  assign(fD,Fi);
  reset(fD);
  readln(fD,N,kB);
  FillChar(M,SizeOf(M),0);
  MM:=0;
  with B do begin
    L:=0; D:=0; R:=N; U:=N;
  end;
  InCells(B);
  for i:=1 to kB do begin
    readln(FD,B.L,B.D,B.R,B.U);
    SortBar(B);
    nCells(B);
  end;
```

```
  close(FD);
end;
Procedure Field (i,j:Byte);
var k:Byte;
begin
  M[i,j]:=-1;
  for k:=1 to 4 do
    if M[i+u[k],j+v[k]]=RR then Field(i+u[k],j+v[k]);
end;
Procedure Run;
var i,j:Byte;
begin
  kS:=0;
  for i:=1 to N do
    for j:=1 to N do
      if M[i,j]>0 then begin
        inc(kS);
      RR:=M[i,j];
      Field(i,j);
    end;
end;
Procedure Done;
var fS:Text;
begin
  assign (fS,Fo);
  rewrite(fS);
  writeln(fS,kS);
  close (fS);
end;
begin
  Init; Run; Done
end.
```

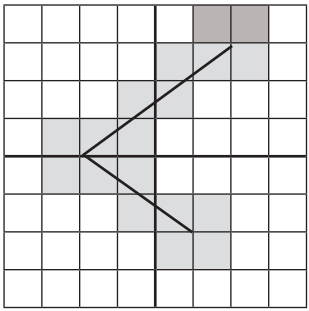
ЗАФАРБОВАНІ КЛІТИНКИ

(О-2001)

На аркуші паперу в клітинку накреслили ламану лінію з N вершин, які лежать на перетинах ліній сітки. Всі клітинки, які перетнула ламана, зафарбували (клітинка вважається перетнутою, якщо вона має з ламаною хоча б одну спільну точку). Обчислити кількість зафарбованих клітинок, якщо початок системи координат лежить на перетині ліній сітки, осі паралельні лініям сітки, а одиничний відрізок дорівнює стороні клітинки.

Усі числові значення цілі, по модулю не перевищують 100.

input.txt	3 2 3 −2 0 1 −2	Значення N Координати вершин ламаної
output.txt	18	Кількість зафарбованих клітинок



Розв'язання

Наочним обладнанням до задачі може бути старенький монітор із малим дозволом, наприклад 320×240, — це майже як у дисплея сучасного телефону. Тепер декілька зауважень безпосередньо щодо розв'язування задачі:

- координати N вершин ламаної читаємо в масив координат T;
- інформацію про «перетнутість» клітинок зберігатимемо в матриці C, де $C[i,j]$ кількість можливих перетинів комірки, що має вершини (i,j), $(i+1,j)$, $(i,j+1)$, $(i+1,j+1)$ з ланками ламаної;
- для кожної ланки з кінцями в точках $T[k]$, $T[k+1]$ переглядаємо всі клітинки, які можуть бути «перетнутими» цією ланкою;
- у випадку, якщо всі вершини клітинки лежать по один бік від відповідної ланки, то клітинка не перетинається ламаною;
- після перегляду всіх ланок кількість ненульових елементів у матриці C буде відповіддю до задачі.

Усе вищенаведене конкретизовано в наступній програмі.

```
var
  N:Integer;
  T:Array [1..mm] of Tpset;
  C:Array [-mm..mm,-mm..mm] of Integer;
  Mx,Lx,My,Ly,CC:Integer;
Procedure Init;
```

```
var
  i, j:Integer;
  f:Text;
begin
  assign(f,Fi);
  reset(f);
  for i:=-mm to mm do
    for j:=-mm to mm do C[i,j]:=0;
  Mx := -777; Lx := 777; My := -777; Ly := 777;
  readln(f,N);
  for i:=1 to N do const mm = 100;
    Fi='input.txt'; Fo='output.txt';
  Type
    Tpset = Record
    X,Y: Integer;
  end;
  begin
    readln(f,T[i].X,T[i].Y);
    if T[i].X>Mx then Mx:=T[i].X;
    if T[i].X<Lx then Lx:=T[i].X;
    if T[i].Y>My then My:=T[i].Y;
    if T[i].Y<Ly then Ly:=T[i].Y;
  end;
  close(f);
end;
Function ZZ (u:Integer):Integer;
begin
  if u>0 then ZZ:=1;
  if u<0 then ZZ:=-1;
  if u=0 then ZZ:=0;
end;
Function Cell (A,B:Tpset; x,y:Integer):Boolean;
var z1,z2,z3,z4:Integer;
begin
  z1:=ZZ((x-A.X)*(B.Y-A.Y)-(y-A.Y)*(B.X-A.X));
  z2:=ZZ((x+1-A.X)*(B.Y-A.Y)-(y-A.Y)*(B.X-A.X));
  z3:=ZZ((x-A.X)*(B.Y-A.Y)-(y+1-A.Y)*(B.X-A.X));
  z4:=ZZ((x+1-A.X)*(B.Y-A.Y)-(y+1-A.Y)*(B.X-A.X));
  Cell:=not (Abs(z1+z2+z3+z4)=4);
end;
Procedure Run;
var
  i,j,k:Integer;
  L,R,D,U:Integer;
begin
```

```
  for k:=1 to N-1 do begin
   if T[k].X<T[k+1].X
     then begin L:=T[k].X; R:=T[k+1].X end
     else begin L:=T[k+1].X; R:=T[k].X end;
   if T[k].Y<T[k+1].Y
     then begin D:=T[k].Y; U:=T[k+1].Y end
     else begin D:=T[k+1].Y; U:=T[k].Y end;
   for i:=L-1 to R do
     for j:=D-1 to U do
       if Cell(T[k],T[k+1],i,j) then inc(C[i,j]);
  end;
  CC := 0;
  for i:=Lx-1 to Mx do
    for j:=Ly-1 to My do
      if C[i,j]>0 then inc(CC);
end;
Procedure Done;
var f : Text;
begin
  assign (f,Fo);
  rewrite(f);
  writeln(f,CC);
  close(f);
end;
begin
  Init; Run; Done;
end.
```

АЛЬПІНІСТ, АБО МЕТОД ХВИЛІ

(О-2002)

Альпініст має план гірської місцевості у вигляді матриці $N \times M$, кожний елемент якої дорівнює висоті над рівнем моря у відповідній одиничній клітинці місцевості. Перехід (крок) можна здійснити в кожну із сусідніх клітинок через спільну сторону лише у випадку, якщо значення висот у цих клітинках відрізняються не більше ніж на 2.

У початковий момент альпініст знаходиться в клітинці $(1,1)$. Мета альпініста — дістатися найвищої точки місцевості (якщо це можливо). Яку мінімальну кількість кроків K необхідно зробити альпіністу до вершини?

Вхідні дані

У першому рядку вхідного файла записані натуральні числа N, M — розміри матриці. В наступних N рядках записано по M ці-

лих чисел — висоти над рівнем моря у відповідних клітинках місцевості. Всі числа не перевищують 100.

Вихідні дані

У вихідний файл необхідно записати мінімальну кількість кроків K або – 1, якщо дістатися вершини неможливо.

input.txt	5 4 3 4 2 2 6 4 5 4 6 7 7 8 3 8 6 7 3 7 7 6	Значення N і M План місцевості
output.txt	5	Мінімальна кількість кроків K

Розв'язання

Відповідь у подібних задачах можна отримати, використавши «метод хвилі». За допомогою цього методу можна просто знайти мінімальний шлях у лабіринті, поставити мат у два ходи в шахах, виявити зв'язні частини на карті місцевості тощо. Алгоритмічна (програмна) реалізація «методу хвилі» повністю відтворює перегляд вершин графа в ширину.

Працює цей метод приблизно так:

1) шукаємо клітинку (i), де ми знаходимося зараз — у початковий (поточний) момент часу;
2) всі клітинки, на які можна потрапити з поточних за один крок, відмічаємо, попередньо перевіривши, чи не досягли ми вершини (кінцевої клітинки);
3) якщо пошук не закінчено, то відмічені на цьому етапі клітинки набувають статус поточних і процес триває, аж поки не знайдемо шукану вершину або на деякій ітерації не знайдеться відмічених клітинок — у такому випадку результату немає.

Запишемо клітинки, знайдені на кожному з етапів.

Кількість кроків	**Координати клітинки**
0	1 1
1	1 2
2	1 3 2 2
3	1 4 2 1 2 3
4	2 4 3 1 3 3
5	3 4

Технічно реалізувати такий процес можна у два способи. Перший, простий і нераціональний — відмічати пройдені клітинки

безпосередньо в матриці і на кожному кроці спочатку відшукати відмічені клітинки, а потім відмітити сусідні. Або можна організувати пошук у ширину по черзі, коли координати відмічених клітинок поміщаються в масив координат за «головним» індексом P, а вибираються координати за «хвостовим» індексом Q.

Процес закінчується, коли результат знайдено або $P = Q$, тобто нових відмічених клітинок не з'явилося і задача не може бути розв'язана.

```
const
  _n = 100;
Fi = 'input.txt';
Fo = 'output.txt';
A:Array [1..4] of Integer=(-1,0,1,0);
B:Array [1..4] of Integer=(0,1,0,-1 );
Type
  Titem=Record
  H,R,C:Integer;
  end;
  var  N,M: Integer;
    Tur: Array [1.._n*_n] of Titem;
    Map: Array [0.._n+1,0.._n+1] of Integer;
    T:Titem;
    U,P,Q,G,Max:Integer;
    Fl:Boolean;
Procedure Init;
var
  i,j,p:Integer;
  f:Text;
begin
  assign(f,Fi);
  reset(f);
  readln(f,N,M);
  for i:=1 to N do begin
    for j:=1 to M do begin
      read(f,Map[i,j]);
    end;
    readln(f);
  end;
  close(f);
  Max:=0;
  for i:=1 to N do
    for j:=1 to M do
      if Map[i,j]>Max then Max:=Map[i,j];
  for j:=0 to M+1 do begin
    Map[0,j]:=7777; Map[N+1,j]:=7777;
```

```
  end;
  for i:=0 to N+1 do begin
    Map[i,0]:=7777; Map[i,M+1]:=7777;
  end;
  if N>M then p:=N else p:=M;
end;
Procedure PutChain (c,a,b:Integer);
begin
  inc(P);
  Tur[P].H := c;
  Tur[P].R := a;
  Tur[P].C := b;
  Map[a,b]:=-Map[a,b];
  Fl:=(Abs(Map[a,b])=Max);
end;
Procedure GetChain (var L:Titem);
begin
  inc(Q);
  L:=Tur[Q];
end;

Procedure Run;
var  k:Integer;
begin
  P:=0; Q:=0; G:=0;
  Fl:=False;
  PutChain(0,1,1);
  while not Fl and (P>Q) do begin
    GetChain(T);
    k:=0;
    repeat
    inc(k);
    if (Abs(Abs(Map[T.R,T.C])-
    Map[T.R+A[k],T.C+B[k]])<3) and
    (Map[T.R+A[k],T.C+B[k]]>0)
    then PutChain(Q,T.R+A[k],T.C+B[k]);
    until Fl or (k=4);
  end;
    if Fl then begin
    repeat
    T:=Tur[P]; inc(G); P:=T.H;
    until P=0;
    Dec(G);
  end
  else G:=-1;
end;
Procedure Done;
var f:Text;
```

```
begin
  assign (f,Fo);
  rewrite(f);
  writeln(f,G);
  close (f);
end;
begin
  Init; Run; Done;
end.
```

Схожа задача була на обласній олімпіаді 2004 року, і вона пропонується для самостійного розв'язання.

Нарешті Робінзон скачав з Інтернету фото свого острова і зможе побудувати хатину на південному узбережжі. Острів Робінзона омивається з усіх боків океаном і має внутрішні водойми. Фото — прямокутна матриця розміром $N \times M$ клітинок, де 0 — клітинка води, 1 — клітинка суші. Стара оселя Робінзона знаходиться в клітинці з координатами A, B. Допоможіть Робінзону вибрати клітинку C, D на південному узбережжі для побудови нової хатини так, щоб час T руху між хатинами був мінімальним. Якщо таких клітинок декілька, достатньо вказати одну з них. Рухається Робінзон до сусідніх клітинок через спільні сторони і витрачає 1 год на клітинку суші і 3 год — на клітинку води (час, затрачений на першу й останню клітинку маршруту, теж враховується). Клітинка південного узбережжя — остання 1 в кожному стовпці матриці.

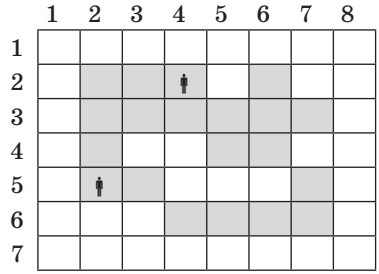
Числові значення N, M, A, B — натуральні, не більші за 50.

input.txt	7 8 0 0 0 0 0 0 0 0 0 1 1 1 0 1 0 0 0 1 1 1 1 1 1 0 0 1 0 0 1 1 0 0 0 1 1 0 0 0 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 2 4	Значення N і M Фото острова Координати A, B
output.txt	6 5 2	Мінімальний час T, год Координати C, D

ДЕКІЛЬКА ПІДХОДІВ ДО МАФІЇ

(О-2001)

Пропонуємо далі задачу на улюблену українську тему — про корупцію.



У державі Віртуляндії міністерство, що видає ліцензії, має N чиновників (включаючи міністра), причому справедливі правила:

- кожен із чиновників (крім міністра) має одного начальника;
- кожен із чиновників має від 0 до 4 підлеглих чиновників;
- начальник мого начальника не є моїм начальником;
- підлеглий мого підлеглого не є моїм підлеглим.

Для отримання ліцензії на відкриття фірми потрібно мати підпис міністра. Для отримання підпису будь-якого чиновника (в тому числі міністра) потрібно дати йому хабара, попередньо отримавши підпис будь-кого з його підлеглих, якщо вони є.

Визначте найдешевший шлях для отримання ліцензії.

Технічні вимоги

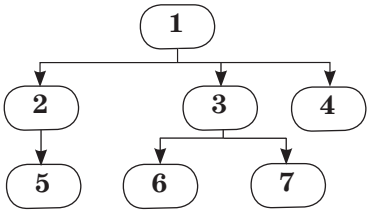
У першому рядку текстового файла input.txt записане число N — кількість чиновників у міністерстві. Далі йдуть N рядків. В i-му рядку ($i = 1 \ldots N$) записано дані про i-го чиновника в такій послідовності: $D[i]$ — сума хабара, $K[i]$ — кількість та номери його підлеглих. Номер міністра дорівнює одиниці. Всі числа натуральні, не перевищують 1 000 і відділені пропусками.

У перший рядок вихідного файла output.txt потрібно записати мінімальну суму, яку потрібно сплатити для отримання підпису. У другому рядку — послідовність номерів чиновників за найдешевшого шляху.

Приклад введення та виведення

input.txt	7 64 3 2 3 4 69 1 5 15 2 6 7 91 0 57 0 52 0 45 0	Значення N Розмір хабара та список підлеглих кожного чиновника
output.txt	124 7 3 1	Мінімальна сума та список найдешевшого шляху

Рисунок до файла input.txt



Розв'язання

Якщо вам вдалося дочитати нашу книжку до цього місця, то ви вже маєте значний досвід у розв'язуванні олімпіадних задач із програмування, а також оперуєте достатнім теоретичним багажем знань. Отже, розповідати зараз про графи, основні задачі й алгоритми на графах ми не будемо (*див. додатки*), а за умовчання припустимо, що читачеві відомі ці поняття.

Тепер про задачу. Вочевидь, задано дерево, коренем якого є міністр, а вершинами —інші чиновники, пов'язані між собою відповідно до гілок дерева.

Для розв'язування задачі потрібно знайти найдешевший шлях від якогось із листків до кореня. Це можна зробити, використавши стандартний пошук у глибину або ширину на заданому дереві від кореня до всіх листків. Також має місце і такий розв'язок:

1) прочитавши дані з файла, для кожного чиновника запишемо таке: номер його начальника та наявність у нього підлеглих, номер начальника в міністра буде 0;
2) переглядаючи всіх чиновників підряд, зупиняємось на тих, які не мають підлеглих, — це буде початком кожного зі шляхів до міністра;
3) піднявшись кожного разу службовими східцями від найнижчого чиновника до міністра, обчислимо відповідну суму та вибиремо з усіх найменшу;
4) задачу розв'язано, і вона вкладається в лінійний час.

У програмі пропонується переглядати дерево знизу — від листків до кореня:

```
const
  Fi = 'input.txt';
  Fo = 'output.txt';
Type
  TChin = Record
    R:Byte;
    F:Boolean;
    V:Word;
  end;
var
  N:Word;
  W : array [1..1000] of Tchin;
```

```
  Min: Word;
Procedure Init;
var
i,k,j,p:Byte; U:Word;
FD:Text;
begin
  Min:=0;
  assign(FD,Fi); reset(FD);
  readln(FD,N);
  for i:=1 to N do
    with W[i] do begin
      R:=0; V:=0; F:=false;
    end;
  for i:=1 to N do begin
    read(FD,U,k); inc(Min,U);
    W[i].V:=U;
    if k>0 then begin
      W[i].F:=true;
      for j:=1 to k do begin
        read(FD,p); W[p].R:=i;
      end;
    end;
    readln(FD);
  end;
  close(FD);
end;
Procedure Run;
var
i,j,l:Byte; U:Word; FS:Text;
begin
  assign(FS,Fo); rewrite(FS);
  for i:=1 to N do
  if not W[i].F then begin
    U:=0; j:=i;
    repeat
    U:=U+W[j].V;
    J:=W[j].R;
    until j=0;
    if U<Min then begin
      Min:=U; l:=i
    end;
  end;
  writeln(Min);
  repeat
  write(l,' ');
  l:=W[l].R;
```

```
  until l=0;
  close(FS);
end;
begin
  Init; Run;
end.
```

CD-БІРЖА

(О-2006)

У комп'ютерному клубі діє біржа з обміну CD-дисків, де можна обміняти будь-який диск на інший диск із каталогу, якщо не безпосередньо, то через проміжні обміни.

Каталог біржі містить список N різних дисків з номерами $1 \ldots N$. Для i-го диску каталогу ($i = 1 \ldots N$) зазначено список номерів дисків, які можна отримати в обмін, доплативши при цьому 1 грн за кожний обмін.

Маючи достатню кількість копій L перших дисків каталогу, потрібно отримати всі диски з каталогу за мінімальну кількість (D грн) доплат. Усі числові значення натуральні, не більші за 100.

input.txt	5 2 3 5 1 5 2 2 3 4	Значення N, L Диск № 1 можна обміняти на № 3 Диск № 2 можна обміняти на № 5 Диск № 3 можна обміняти на № 1 або № 5 Диск № 4 можна обміняти на № 2 Диск № 5 можна обміняти на № 2, № 3 або № 4
output.txt	4	Значення D

Пояснення

Диск № 1 міняємо на № 3 (1 грн), диск № 2 — на № 5 (1 грн), диск № 2 — на № 5, а за нього вимінюємо № 4 (2 грн). Витрачено 4 грн.

Розв'язання

Каталог CD-біржі вдало вписується в орієнтовний граф, де номери CD-дисків — це вершини, а ребра відповідають кожному з можливих обмінів дисків.

Вочевидь, вже на етапі введення даних необхідно побудувати таблицю зв'язності графа, причому ребрам, що відповідають можливим обмінам, надається значення 1, елементам головної діагоналі — 0, а всім іншим — достатньо велике значення, щоб унеможливити обмін.

Використавши алгоритм знаходження найкоротших шляхів у графі (алгоритм Флойда-Уоршела), знаходимо матрицю W — від-

стань для кожної пари дисків, тобто мінімальну суму, яку необхідно витратити, щоб обміняти диск і на диск j. Далі для кожного диску з номером j від $L+1$ до N знаходимо такий, щоб значення $W[i,j]$ було найменшим для всіх наявних дисків із номером i від 1 до L. Сума всіх таких $W[i,j]$ для j від $L+1$ до N буде відповіддю до задачі.

Для знаходження значень матриці W можна використати також алгоритм Дейкстри або пошук в ширину на графі.

```
const
  Fi = 'input.txt';
  Fo = 'output.txt ';
  _p = 100;
  _f = 300;

var
N,C,L:Integer;
M:Integer;
W:Array [1.._p,1.._p] of Integer;
f:Text;

Procedure Init;
var i,j,p:Integer;
begin
  FillChar(W,SizeOf(W),0);
  assign(f,Fi);
  reset(f);
  readln(f,N,L);
  for i:=1 to N do
    for j:=1 to N do
      if (i=j) then W[i,i]:=0
      else W[i,j]:=_f;
  N:=0; i:=0;
  repeat
    inc(i);
    while Not EoLn(f) do begin
      read(f,p);
      W[i,p]:=1;
      If p>N then N:=p;
    end;
    readln(f);
  until EoF(f);
  close(f);
  for i:=1 to N do
  for j:=1 to N do
  if(i<>j)and(W[i,j]=0)then W[i,j]:=_f;
```

```
end;
Procedure Run;
var
  i, j, k:Integer;
  s:Integer;
  u:boolean;
begin
  M:=0;
  repeat
  for i:=1 to N do
  for j:=1 to N do
  for k:=1 to N do
  If W[i,j]>W[i,k]+W[k,j]
  then W[i,j]:=W[i,k]+W[k,j];
  u:=true;
  for i:=1 to N do
    for j:=1 to N do
      for k:=1 to N do
        if W[i,j]=_f then u:=false;
  until u;
  M:=0;
  for j:=L+1 to N do begin
    s:=30000;
    for i:=1 to L do
    if W[i,j] < s then s:=W[i,j];
    inc(M,s);
  end;
  writeln(M);
end;
Procedure Done;
begin
  assign(f,Fo);
  rewrite(f);
  writeln(f,M);
  close(f);
end;

begin
  Init; Run; Done
end.
```

ЛИЖНІ МАРШРУТИ

(О-2007)

Через потепління деякі лижні туристичні маршрути, що з'єднують N пунктів, довелося відмінити або залишити лише в одному напрямку замість двох. Група лижників, що знаходиться в пункті A і має запас харчів лише на C днів, повинна дістатися до пункта B, не заходячи двічі в один пункт. Скільки існує таких маршрутів, якщо перехід між двома пунктами забирає 1 день? ($N = 2...50$).

input.txt	5 2 5 3 2 3 5 1 4 4 5 1 0	Значення N, A, B і C N рядків: у рядку $k = 1 \ldots N$ — список пунктів, у які можна потрапити з пункту k (або 0, якщо доріг немає)
output.txt	3	Кількість маршрутів

Розв'язання

Постановка задачі в графовій інтерпретації буде такою: на орієнтовному графі знайти кількість шляхів обмеженої довжини між двома заданими вершинами. Вірогідно, що правильно організований стандартний пошук у глибину або ширину на графі дасть можливість знайти правильну відповідь.

У наведеній нижче програмі пошук у глибину реалізовано засобами методу **BackTracking** (метод гілок і границь). Зауважимо, що більш зрозумілою може бути рекурсивна реалізація.

```
const
  Fi = 'input.txt';
  Fo = 'output.txt';
  _n = 50;
var
  N,L,A,B : Byte;
  K,M: LongInt;
  W:Arra [1.._n,1.._n] of Boolean;
  F:Set of Byte;
  Z:Array [1.._n] of Byte;
  f,s: Text;
Procedure Init;
var
  p,q: Byte;
  i,j: Integer;
begin
  for i:=1 to _n do
  for j:=1 to _n do W[i,j]:=false;
  assign(f,Fi);
  reset(f);
  readln(f,N,K);
  for i:=1 to K do begin
    readln(f,p,q);
    W[p,q]:=true;
  end;
  readln(f,A,B,L);
  close(f);
end;
```

```
Procedure Run;
var k,i,d:Byte;
begin
  M:=0;
  d:=1; z[d]:=A;
  F:=[A];
  k:=0;
  repeat
  while (k < N) and (z[d]<>B) and (d<L+1) do begin
    inc(k);
    if W[z[d],k] and not (k in F) then begin
      inc(d); z[d]:=k;
      F:=F+[k];
      k:=0;
    end;
  end;
  if (z[d]=B) then inc(M);
  k:=z[d]; dec(d); F:=F-[k];
  until d=0;
end;
Procedure Done;
begin
  assign(s,FileOut);
  rewrite(s);
  writeln(M);
  writeln(s,M);
  close(s);
end;
begin
  Init; Run; Done;
end.
```

У нашому будинку всі ліфти справні, тому що вони уявні. Не бійтесь клаустрофобії, погоджуйтесь на прогулянку.

ПОКАТАЙМОСЯ НА ЛІФТІ

(О-2004)

В N-поверховому будинку працює L ліфтів, кожен з яких обслуговує тільки окремі поверхи (не менше двох), на які вказує AB — код ліфту (A — найменший номер поверху, де зупиняється ліфт, B — крок).

Наприклад, ліфт із кодом 14 при $N=10$ зупиняється так: 1 5 9 5 1.

Знаючи значення N і L та коди всіх ліфтів, обчислити:

а) кількість K поверхів, де не зупиняється жоден ліфт;

б) кількість P поверхів, на які можна потрапити з першого поверху;

в) найменшу кількість M ліфтів, які потрібно використати, щоб дістатися з першого на останній поверх будинку (або вивести 0, якщо це неможливо).

Числові значення N, L, A і B — натуральні, N, L не перевищують 200.

input.txt	11 3 3 4 3 2 1 3	Значення N і L Коди L ліфтів
output.txt	3 8 2	Значення K Значення P Значення M

Розв'язання

Розміщено серед графових задач, тому що модель схожа на неорієнтований граф, а ідея алгоритму знаходиться в площині структури даних. Пояснення проведемо, використовуючи вхідний файл з умови. Вчиняємо так: прочитавши з файла код першого ліфту 3 4, формуємо множину $[3,7,11]$ — можливі поверхи, на яких зупиняється цей ліфт. Запишемо відповідні множини до таблиці.

Номер ліфту	Код ліфту	Множина поверхів
1	3 4	$[3,7,11]$
2	3 2	$[3,5,7,9,11]$
3	1 3	$[1,4,7,10]$
Об'єднання		$[1,3,4,5,7,9,10,11]$

Ймовірно, об'єднання множин міститиме номери поверхів, на які можна дістатися, якщо користуватися всіма ліфтами. Отже, якщо від кількості поверхів N відняти кількість елементів у множині об'єднання, отримаємо відповідь на перше запитання задачі.

Далі знаходимо ліфт із найменшим номером, який зупиняється на першому поверсі, і рекурсивно формуємо множину поверхів, які є доступними з першого. Знаходимо кількість елементів у такій множині і маємо відповідь на друге запитання задачі.

Щоб знайти відповідь на останнє запитання, потрібно послідовно побудувати множини поверхів, на які можна потрапити з першого, використавши один, два або більше ліфтів. Пошук зупиняється, щойно елементом однієї з множин з'явиться N-й поверх. Описані алгоритми конкретизуються в такій програмі:

```
const
  Fi = 'input.txt';
  Fo = 'output.txt';
```

```
  _N = 200;
Type Tset=Set of Byte;
var
  N,L: Byte;
  K,P,M : Byte;
  U,Z: Array [1.._N] of Tset;
  W,V: Tset;

Procedure Init;
var
  i,a,b:Byte;
  fD : Text;

begin
  assign(fD,FileIn);
  reset(fD);
  readln(fD,N,L);
  for i:=1 to L do begin
    readln(FD,a,b);
    U[i]:=[a];
    while (a+b<=N) do begin
      inc(a,b);
      U[i]:=U[i]+[a]
    end;
  end;
  close(FD);
end;

Function Count (S:Tset):Byte;
var
i,j: Byte;
begin
  j:=0;
  for i:=1 to N do
    if (i in S) then inc(j);
  count:=j;
end;
Procedure Floor (i:Byte);
var j:Byte;
begin
  V:=V+[i];
  W:=W+U[i];
  for j:=1 to N do
```

```
    if not (j in V) and (U[i]*U[j]<>[]) then Floor(j);
end;
Function Lift:Byte;
var i,j,q:Byte;
begin
  W:=[];
  for i:=1 to L do
    if (1 in U[i]) then W:=W+U[i];
  q:=1;
  while not (N in W) do begin
  inc(q); V:=W; W:=[];
  for i:=1 to L do
  if (V*U[i]<>[]) then W:=W+U[i];
  end;
  Lift:=q;
end;
Procedure Run;
var i:Byte;
begin
  W:=[];
  for i:=1 to L do W:=W+U[i];
  K:=N-Count(W);
  {---------------------------}
  W:=[]; V:=[];
  i:=0;
  repeat
  inc(i);
  until (i=L) or (1 in U[i]);
  if (1 in U[i]) then Floor(i);
  P:=Count(W);
  {---------------------------}
  If (N in W) then M:=Lift else M:=0;
end;
Procedure Done;
var
  i:Byte;
  fS:Text;
begin
  assign(fS,FileOut);
  rewrite(fS);
  writeln(fS,K);
  writeln(fS,P);
```

```
  writeln(fS,M);
  close(fS);
end;

begin
  Init; Run; Done;
end.
```

КОМАНДА МЕРА ЯК НАЙБІЛЬШИЙ ПОВНИЙ ПІДГРАФ

(О-2002)

Для забезпечення перемоги на виборах мер вирішив утворити команду зі своїх знайомих, у якій кожен є другом кожного з інших. Знаючи відносини між собою всіх N знайомих мера, утворити команду найбільшої чисельності M. За існування декількох розв'язків достатньо навести лише один з них.

Технічні вимоги

У першому рядку вхідного файла input.txt міститься єдине число N — кількість знайомих мера. У другому рядку — число K — кількість пар, що мають дружні відносини. У наступних K рядках записані через пропуск пари чисел a, b — номери друзів ($N < 50$).

У першому рядку вихідного файла output.txt записати число M — найбільшу чисельність команди мера. У наступному рядку записати склад команди.

Приклад введення та вивведення

input.txt	5 6 1 2 2 3 1 3 4 5 1 5 3 4	Значення N Значення K K рядків: номери пар друзів
output.txt	3 1 2 3	Найбільша чисельність команди мера Склад команди мера

Розв'язання

Сформулюємо задачу в графовій інтерпретації: на неорієнтовному графі знайти найбільший повний підграф. Тому задача може бути розв'язана звичайним переглядом графа в глибину. Щоб позбавитися повторень підграфів, вершини необхідно переглядати в порядку зростання їх номерів та включати в поточну «команду» тільки тих учасників, які «дружать» з усіма учасниками цієї команди.

Програмна реалізація цього алгоритму міститься у перегляді графа під керуванням стеку.

```
const
  Fi = 'input.txt';
  Fo = 'output.txt';
  Nmax = 50;
var
  N,K,mD,D:Byte;
  W : Array [1..Nmax,1..Nmax] of Boolean;
  F : Array [1..Nmax] of Boolean;
  Z,mZ : Array [1..Nmax] of Byte;
  Flag : Boolean;
Procedure Init;
var
  i,p,q:Byte;
  FD : Text;
begin
  FillChar(W,SizeOf(W),false);
  assign(FD,Fi);
  reset(FD);
  readln(FD,N);
  readln(FD,K);
  for i:=1 to K do begin
  readln(FD,p,q);
  W[p,q]:=true; W[q,p]:=true;
  end;
end;
Function Test(l:Byte):Boolean;
var
  f:Boolean;
  i : Byte;
begin
  f:=true;
  for i:=1 to D do
  if not W[l,Z[i]] then f:=false;
  Test:=f;
end;
Procedure Run;
var k,i : Byte;
begin
  Flag:=false;
  FillChar(f,SizeOf(f),true);
  d:=0; k:=0; Md:=0;
  repeat
```

```
    while (k < N) do begin
      inc(k);
      if (d=0) or Test(k) and f[k] then begin
        inc(d); z[d]:=k;
        f[k]:=false;
      end;
    end;
    if (d > Md) then begin mD:=d; mZ:=z; end;
      if d>0 then begin
        k:=z[d]; dec(d);
        F[k]:=true;
      end
        else Flag:=true;
      until Flag;
  end;
  Procedure Done;
  var
    i:Byte;
    Fs : Text;
  begin
    assign(Fs,Fo);
    rewrite(Fs);
    writeln(Fs,mD);
    for i:=1 to mD do write(Fs,mZ[i],' ');
    writeln(Fs);
    close(Fs);
  end;
  begin
    Init; Run; Done;
  end.
```

Розглянемо також дещо інший спосіб розв'язування задачі «про команду мера», використавши перегляд вершин графа в ширину.

Усі повні можливі підграфи запишемо до масиву множин W у порядку зростання кількості вершин у кожному підграфі. А саме:

1) спочатку записуємо в масив W усі одноелементні підграфи.
2) кожен крок алгоритму полягає в спробі розширити вже сформовану «команду» на одного учасника з номером, більшим від найбільшого номера команди;
3) якщо це вдалося, то нову «команду» записуємо до того ж масиву W;
4) процес відбувається поступово і зупиняється, коли збільшити жодну з «команд» неможливо;
5) останній елемент масиву W і буде найбільшою можливою «командою».

```
const
  Fi = 'input.txt';
  Fo = 'output.txt';
  Nmax = 50;
Type
Tset = set of Byte;
var
  N,M: Byte;
  Z : Array [1..Nmax] of Tset;
  W : Array [1..Nmax*Nmax div 2] of Tset;
  Q,P: Word;
Procedure Init;
var
  fd:Text;
  i,a,b:Byte;
begin
  assign(fd,Fi);
  reset(fd);
  read(fd,N,M);
  for i:=1 to M do begin
    read(fd,a,b);
    z[a]:=z[a]+[a,b];
    z[b]:=z[b]+[a,b];
  end;
  close(fd);
end;
Function MaxSet  (R:Tset):Byte;
vari,j: Byte;
begin
For i:=1 to n do
if i in R then j:=i;
MaxSet:=j;
end;
Function CountSet (R:Tset):Byte;
var i,j : Byte;
begin
  j:=0;
  for i:=1 to n do
    if i in R then inc(j);
  CountSet:=j;
end;
Procedure Run;
var i:Word;
```

```
begin
  for i:=1 to n do w[i]:=[i];
  P:=0; Q:=n;
  repeat
  inc(P);
  for i:=MaxSet(W[P])+1 to n do
    if not (i in W[P]) and (W[P]*Z[i]=W[P]) then
  begin
    inc(Q); W[Q]:=W[P]+[i]
  end;
  until P=Q;
end;
Procedure Done;
var
  i:Byte;
  fs:Text;
begin
  assign(fs,Fo);
  rewrite(fs);
  writeln(Fs,CountSet(W[P]));
  for i:=1 to N do
  if i in W[P] then write(Fs,i,' ');
  writeln(Fs);
  close(Fs);
end;

begin
  Init; Run; Done;
end.
```

БАГАТОПРОЦЕСОРНИЙ НАУКОВИЙ ПРОЕКТ

(О-2009)

Науковий проект складається з N задач і виконується на N-процесорному суперкомп'ютері, кожна задача міститься тільки на одному з процесорів. Задачі можуть виконуватись паралельно з іншими задачами, але декільком з них необхідно мати результати деяких інших задач проекту. Для кожної задачі відомий час, необхідний для її виконання, та список попередніх задач, які мають бути закінчені перед її запуском (цей список може бути й порожнім).

Знайти мінімальний час, за який можна виконати всі N задач проекту або вивести 1, якщо зробити це не можливо. Всі вхідні числові значення натуральні, не більші за 100.

Вхідні дані

Перший рядок — значення N, у наступних N рядках — час і список попередніх задач (якщо вони є) для кожної задачі.

Вихідні дані

Єдине число — найменший час виконання проекту або 1, якщо проект виконати неможливо.

input.txt	5 1 2 4 5 3 4 2 4 2 3 4	Значення N Час виконання і список попередніх задач для кожної задачі
output.txt	7	Найменший час виконання проекту

Розв'язання

Для розв'язання пропонується, на мій погляд, цікавий алгоритм, що за своїм стилем дуже нагадує відомий графовий алгоритм Дейкстри.

Алгоритм складається з N однакових ітерацій:

1) знаходимо задачу з номером m, що не має попередніх задач, тобто $y[m]=[\ ]$, і має найменший час виконання $x[m]$;
2) виконуємо задачу з номером m, тобто з усіх списків попередніх задач видаляємо цю задачу і, якщо це була остання задача в списку деякої i-ї задачі, яку вже можна виконувати, то час виконання i-ї задачі потрібно збільшити на час виконання m-ї.

Час виконання проекту дорівнюватиме найбільшому часу виконання деякої i-ї задачі, обчисленому описаним способом.

Якщо на якомусь кроці m-ї задачі не знайдено, то виконати проект не можливо. Описаний алгоритм реалізований у програмі.

```
const
  Fi = 'input.txt';
  Fo = 'output.txt';
  _n = 100;
var
  y:Array [1.._n] of set of Byte;
  x:Array [1.._n] of Integer;
  i,_m,n,l,m : Integer;
  r:set of Byte;
  fl : Boolean;
  f,s: Text;
begin
  assign(f,Fi);
```

```
  reset(f);
  readln(f,n);
  for i:=1 to n do begin
    read(f,x[i]);
    y[i]:=[];
    while not eoln(f) do begin
      read(f,l);
      y[i]:=y[i]+[l];
    end;
    readln(f);
  end;
  close(f);
  l:=0; r:=[];
  fl:=true;
  repeat
  inc(l);
  _m:=$7777;
  for i:=1 to n do
    if not (i in r) and (y[i]=[]) and (x[i]<_m)
  then begin _m:=x[i]; m:=i; end;
  if _m=$7777 then begin fl:=false; break end;
  r:=r+[m];
  for i:=1 to n do
    if not (i in r) and (m in y[i]) then begin
      y[i]:=y[i]-[m];
      if y[i]=[] then x[i]:=x[i]+x[m];
    end;
    until (l=n);
    _m:=-1;
    if fl then for i:=1 to n do if x[i]>_m then
  _m:=x[i];
  assign(s,Fo); rewrite(s);
  writeln(s,_m);
  close(s);
end.
```

АРИФМЕТИЧНИЙ РЕБУС

(О-2001)

Арифметичний ребус — це запис вигляду $M \,\&\, N = P$, де M, N, P — послідовності з цифр 0, 1, 2, …, 9 та/або латинських букв a, b, c, …, j, якими позначені невідомі цифри, а & — знак однієї з операцій + або *.

Щоб розв'язати ребус, потрібно кожну букву замінити на цифру, щоб отримати правильну рівність, за такими правилами:

- однакові букви в усьому ребусі замінюються однією цифрою, якої немає серед відомих цифр, різні букви — різними цифрами, яких немає серед відомих цифр;
- перша цифра у числах M, N, P — не нуль, а кожне з чисел M, N, P не перевищує 999 999.

Якщо розв'язку ребуса не існує, вказати відповідь 0.

input.txt	$ab8 + aba = 1b1b$	Ребус
output.txt	758 + 757 = 1 515	Будь-який розв'язок ребуса

Розв'язання

Достатньо громіздка перебірна задачка. Оскільки програмний код вже зайняв чотири сторінки, то будемо «економити» на поясненні.

Для кожної букви 'a', 'b', … 'j' латинського алфавіту заповнюємо шаблон у змінну типу **Tchar**. Для прикладу з умови отримаємо такі значення:

Елемент	W	F	A	B	C
V[0]	'a'	True	100	101	0
V[1]	'b'	True	10	10	101
V[2]..V[9]	'c'…'j'	False	0	0	0

Поле W містить один символ 'a', 'b',... або 'j', поле F дорівнює True, якщо відповідний символ поля W зустрічається в ребусі, і False — у протилежному випадку. Десяткові розряди в числових полях A, B, C встановлюються в 1, якщо відповідний символ поля W міститься у числах ребуса, причому A обслуговує перше число, B — друге, а C — результат.

У змінних Na, Nb, Nc знаходяться відомі числові частини чисел, що містяться в ребусі, а в змінних Ma, Mb, Mc мінімальні значення цих чисел, що служать для того, щоб виключити випадки з нулем у старшому розряді.

А далі перебираються всі можливі розміщення значень цифр, що не зустрічаються явно в числах (відповідне значення елемента масиву Y буде True) замість наявних у ребусі символів. Для кожної комбінації цифр після виконання операції в лівій частині перевіряється рівність із результатом у правій частині, і якщо значення збігаються, то ребус розгадано.

```
const
  Fi = 'input.txt';
  Fo = 'output.txt';
```

```
Type
  TChar = Record
  W:Char;
  F:Boolean;
  A,B,C:LongInt;
end;
var
  V:Array [0..9] of TChar;
  S:string;
  FD,FS:Text;
  Y,T: Array [0..9] of Boolean;
  N:Byte;
  Sa,Sb,Sc:String;
  Na,Nb,Nc:LongInt;
  Ma,Mb,Mc:LongInt;
  Z:Char;
  X:Array [0..9] of Byte;
Function Grab : String;
var u:string;
begin
  u:='';
  while S[1] in ['0'..'9','a'..'j'] do begin
    u:=u+s[1]; Delete(S,1,1);
  end;
  Grab:=u;
end;
Function Extr1 (p:Char; s:String):Longint;
var
  i:Byte;
  h:LongInt;
begin
  h:=0;
  For i:=1 to Length(s) do h:=h*10+Ord(p=s[i]);
  Extr1:=h;
end;
Function Extr2 (s:String):Longint;
var
  i:Byte;
  h:LongInt;
begin
  h:=0;
  for i:=1 to Length(s) do
```

```
  h:=h*10+Ord(s[i] in ['0'..'9'])*(Ord(s[i])-48);
  Extr2:=h;
end;
Function Size (s:string):Longint;
var
  i:Byte;
  h:LongInt;
begin
  h:=1;
  for i:=1 to Length(s)-1 do h:=h*10;
  Size:=h;
end;
Procedure Initvar;
var i:Byte;
begin
  N:=0;
  for i:=0 to 9 do
  with V[i] do begin
    W:=Chr(97+i);
    F:=(Pos(W,S)>0);
    A:=0; B:=0; C:=0;
    if F then inc(N);
  end;
    for i:=0 to 9 do Y[i]:=(Pos(Chr(48+i),S)=0);
    Sa:=Grab; Ma:=Size(Sa);
    Z :=S[1]; Delete(S,1,1);
    Sb:=Grab; Mb:=Size(Sb);
    Delete(S,1,1);
    Sc:=Grab; Mc:=Size(Sc);
    for i:=0 to 9 do
  if V[i].F then
  with V[i] do begin
    A:=Extr1(W,Sa);
    B:=Extr1(W,Sb);
    C:=Extr1(W,Sc);
  end;
  Na:=Extr2(Sa); Nb:=Extr2(Sb); Nc:=Extr2(Sc);
end;
Function Test (a1,b1,c1:LongInt):Boolean;
var
  k,i: Integer;
  f:Boolean;
begin
```

```
  k:=-1; f:=false;
  for i:=0 to 9 do
   if V[i].F then begin
     inc(k);
     a1:=a1+V[i].A*x[k];
     b1:=b1+V[i].B*x[k];
     c1:=c1+V[i].C*x[k];
   end;
  if (a1>=Ma) and (b1>=Mb) and (c1>=Mc) then
   Case Z of
     '+' : f:=(a1+b1=c1);
     '-' : f:=(a1-b1=c1);
     '*' : f:=(a1*b1=c1);
   end;
  if f then writeln(FS,a1,Z,b1,'=',c1);
  Test:=f;
end;
Procedure Solut;
var
  d,k,i:Integer;
  fl,Ok : Boolean;
begin
  Initvar;
  FillChar(T,10,true);
  d:=-1; k:=-1; fl:=false; Ok:=false;
  repeat
  while (k<9) and (d<N-1) do begin
   inc(k);
   if T[k] and Y[k] then begin
     inc(d); x[d]:=k; T[k]:=false; k:=-1;
   end;
  end;
  if (d=N-1) then if Test(Na,Nb,Nc) then Ok:=true;
  if d>-1 then begin k:=x[d]; Dec(d); T[k]:=true; end
  else fl:=true;
  until fl;
  if Not Ok then writeln('0');
end;
begin
  assign (FD,Fi); reset (FD);
  assign (FS,Fo); rewrite(FS);
  readln (FD,S); S:=S+'.';
  Solut;
  close(FD); close(FS);
end.
```

В. Л. ДІДКОВСЬКИЙ

СКЛАДНІ НА ПЕРШИЙ ПОГЛЯД

ПЕРЕДМОВА

Автор починав викладати інформатику в школі з середини 80-х років минулого століття, маючи 6 програмованих калькуляторів Б 3-34. У квітні 1987-го з Москви, за попередньою домовленістю з директором Ліанозовського заводу К. В. Агафоновим привезли чотири «АГАТИ», кошти на які виділив міськвиконком на чолі з А. А. Ринкою (тоді за цю суму можна було придбати 4 автомобілі «ЛАДА» — по 4 800 крб за кожний!). Далі до них додали ще 6 «АГАТІВ». Потім з'явилися три так звані «IBM-сумісні» машини дніпропетровського виробництва з вінчестерами, що важили по 5 кг, хоч і мали ємність лише 25 Мб. На Соросівську премію 1998 року вдалося дещо покращити (аж до 386-го процесора!) парк машин шкільного кабінету інформатики. Потім з'явилися перші CD-дисководи, потім ці дисководи почали не лише читати, а й писати диски. 2000 року, коли одинадцятикласник Іван Балабан, який минулого року на олімпіаді в Херсоні був другим, а цього року виборов у Києві перше місце і готувався до міжнародної олімпіади в Пекіні, виникла потреба мати комп'ютер з потужним процесором, на якому можна було б перевірити якість програм (чи відповідають вони критеріям «програма повинна працювати не довше за 0,5 секунди»). Таку машину подарував школі колишній мер М. О. Іванюк (це був «крутий» на той час Pentium 1)...

Впродовж цих років найбільш обдаровані учні, випереджаючи один одного, змагались у складанні програм, «окуповуючи» після уроків і у вихідні комп'ютерний кабінет, вдало виступали на міських і обласних олімпіадах, радуючи вчителя і школу успіхами на всеукраїнському рівні (десятеро учнів 8–11 класів стали переможцями Всеукраїнських олімпіад, із них троє посіли другі, а один — перше місце).

На відстані років добре видно, який подвиг здійснив у 80-ті роки минулого століття Андрій Петрович Єршов. Завдяки його наполегливості і невтомній праці «могучої кучки» його соратників — викладачів вузів, які навчали перших учителів шкільної інформатики, — кожен з нас став не тільки на голову вищим, а й змінив своє бачення світу. Бізнес, престижність і потреба в житті нової справи принесли ЕОМ у багато домівок, зробивши комп'ютер, подібно автомобілю, «не розкішшю, а засобом пересування». Завдяки «чорному» ринку вдалося за помірну ціну наповнити ці комп'ютери програмними засобами. Останніми роками з'явилася нова техніка, карколомні технології, просунувся інтернет...

Деякі із задач цього збірника, створених нами, на перший погляд здаються складними. Але це лише на перший погляд. Насправді більшість із них розв'язується за допомогою відомих алгоритмів інформатики, що стали так би мовити «класичними». Автор сподівається, що детальний розгляд їх повинен озброїти читачів засобами моделювання ситуацій, пробудити творче натхнення в учнів і їх викладачів та розвинути інтуїцію, слугувати становленню креативних можливостей. На жаль, через непоінформованість учителів, не всі з цих алгоритмів стають основою для позакласної роботи в школі. Ліквідувати цю прогалину, зробити авторські розв'язки складених нами задач не тільки інструкцією, а й «інформацією для роздумів» — така мета збірника «Олімпіадна інформатика». Є і більш глобальне сподівання — наблизити нас «до Європи», передових технологій, зробити «конкурентоспроможними» в сучасному світі.

РІЗНІ ЦИФРИ

(Р-2008)

В заданому N-значному числі M залишити найдовшу послідовність сусідніх різних цифр, а решту викреслити. ($N = 1...100$). На яких місцях у числі M стоять перша й остання цифра цієї послідовності? (Зауваження: задачу розв'язати за один перегляд цифр заданого числа.)

Приклад вхідних та вихідних даних

input.txt	202132911	Задане число
output.txt	3291 5 8	Послідовність Номери цифр

Розв'язання

Оскільки в Паскалі числа з кількістю цифр понад 10 «не поміщаються» у тип **LongInt**, будемо читати записане у файлі *f* число як літерну величину (типу **String**).

Нехай *sd* і *ss* (задана і шукана послідовність символів), а *SB* — будь-яка множина симолів (типу **Set of Char**). Вважатимемо спочатку $\mathbf{Max} := 0$; $pp := 0$; $pk := 0$; (найбільша довжина шуканої послідовності *ss* та позиції в *sd* її першого й останнього символів). Зазвичай на олімпіадах з інформатики не аналізують програми (а лише результати їх роботи), але це той випадок, коли журі ретельно переглянуло програми учнів, щоб упевнитися, що вони розв'язали задачу за один прохід циклу — один перегляд цифр записаного у файлі *f* числа, шукаючи потрібну найдовшу послідовність *ss* (учням, які не дотрималися цієї вимоги, хоч і знайшли правильні відповіді, оцінка була зменшена на 50 %).

Ось ця невеличка, зрозуміла програма:

```
uses Crt;
var
  k,r,pp,pk,max: Integer;
  c: Char;
  sb: set of Char;
  ss,sd,s: String;
  dat: Text;
  fl: Boolean;
begin
  clrscr;
  assign(dat,'c:\input.txt'); reset(dat);
  readln(dat,sd); close(dat);
  sb:=[ ]; max:=0;
  if length(sd)=1
```

```
  then begin ss:=sd; pp:=1; pk:=1;
end
else
for k:=1 to length(sd) do
 begin
  r:=k-1;s:=''; fl:=true;
  repeat
  if r<=length(sd)-1
  then
   begin
    c:=sd[r+1];
    if not(c in sb)
    then
     begin
      inc(r); sb:=sb+[c]; s:=s+c;
     end
     else
     begin
      fl:=false;
      if r+1-k>=length(ss)
      then
      begin
       pp:=k;pk:=r;sb:=[ ];
       max:=r+1-k; ss:=s;
      end
     end
   end
 else fl:=false;
 until(fl=false);
end;
writeln('Послідовність:',ss);
writeln('Позиції цифр: ',pp,' ',pk);
readkey;
end.
```

Наступна задача дещо унікальна (подібні задачі більше не розглядаються у цій книжці), її називають класичною, саме вона відкриває цикл задач геометричного характеру.

ОПЕРАЦІЯ

(О-2003)

Для блокади ракової пухлини на живій тканині було N разів застосоване ядерне точкове опромінення в N різних точках.

Лазерний скальпель вилучив ділянку тканини у формі многокутника найменшого периметра, якому належать усі точки опромінення. Обчислити кількість вершин цього многокутника та його площу ($N < 100$, координати точок — цілі числа з проміжку $[-100,100]$).

Приклади вхідного та вихідного файлів

input.txt	4 10 10 0 40 40 0 0 0	Значення N N рядків по два числа — координати точок
output.txt	3 800.00	Кількість вершин Площа многокутника

Розв'язання

Ця задача вважається в інформатиці класичною і належить до задач про так звані «опуклі оболонки».

Якщо маємо непорожню множину точок на площині, то завжди існує безліч опуклих многокутників, таких, що кожна із заданих точок належить цьому многокутнику. Серед них є той, вершинами якого будуть деякі (або всі) задані вершини (їх називають граничними точками), а решта (якщо такі залишилися!) є внутрішніми точками многокутника. Він єдиний з усіх можливих опуклих многокутників, і його називають «опуклою оболонкою» точкової множини. Можливі й так звані «вироджені» випадки, коли опукла оболонка містить одну або дві вершини.

Читач, мабуть, помітив, що невироджена опукла оболонка — це опуклий многокутник найменшої площі (і найменшого периметра), якому належать усі точки множини.

Тому задача «операція» вимагає відшукати опуклу оболонку заданої точкової множини. Як це робиться — розповімо докладно. Зробимо декілька зауважень:

- кожна пряма на площині може бути задана рівнянням типу $ax + by + c = 0$, де a, b, c — деякі числа, які легко визначити, знаючи дві різні точки $(x1, y1)$ і $(x2, y2)$, які належать прямій. Справді, з рівностей $a*x1 + b*y1 + c = 0$ та $a*x2 + b*y2 + c = 0$ матимемо: $a*(x1 - x2) = b*(y2 - y1)$. Тому достатньо вважати, що значення $a = y2 - y1$; $b = x1 - x2$; $c = -(a*x1 + b*y1)$;
- кожна пряма на площині розбиває множину її точок на три підмножини: D, V, Z, у яких ліва частина рівняння прямої $ax + by + c$ відповідно додатня (D), від'ємна (V) або перетво-

рюється на нуль (Z). Сторони опуклої оболонки — це відрізки тих прямих, що проходять через дві із заданих N точок, яких множини D або V порожні, тобто кількості точок у яких kd або kv дорівнюють нулю. Зауважимо, що деякі з цих прямих можуть містити понад дві задані N точки ($kz > 2$). Зрозуміло, що в такому випадку нас цікавитимуть лише крайні точки на цих прямих.

Позначимо через abc(u,v:integer) процедуру визначення параметрів a, b, c прямої, що проходить через точки з індексами u і v ($u \neq v$).

```
a:=y[v]-y[u]; b:=x[u]- x[v]; c:=-a*x[u]-b*y[u]
```

Тоді визначення прямих, на яких лежать сторони опуклої оболонки, виглядатиме таким чином:

```
Procedure obolon;
begin
  P:=0;{кількість сторін оболонки}
  for i: =1 to n-1 do
  for k:=i+1 to n do
  begin
    Abc(i,k);Z:=[];kd:=0;kv:=0;kz:=0;
    for t:=1 to n do
    begin
      W:=a*x[t]+b*y[t]+c;
      If w>0
      then inc(kd)
      else if w<0
           then inc(kv)
           else begin inc(kz);z:=z+[t] end;
    end;
    If kd*kv=0
    Then begin
           inc(p);s[p]:=z;
           If kz>2 then kraj;{крайні точки}
         end
    end;
end.
```

Ось як виглядатиме процедура *kraj*, яка в множині z, що містить понад дві точки, залишає лише крайні точки:

```
procedure kraj;
begin
  Min:=maxint;max:=-maxint;
  if y[i]<>y[k]
```

```
    then begin
           For j:=1 to n do
           If y[j]<min
           then begin min:=y[j];n1:=j end
           else if y[j]>max
                then begin max:=y[j];n2:=j end
         end
  else   if x[i]<>x[k]
         then begin
               For j:=1 to n do
               If x[j]<min
               then begin min:=y[j];n1:=j end
               else if y[j]>max
                    then begin max:=y[j];n2:=j end
             end;
    z:=[n1,n2];s[p]:=z;
  end.
```

Ми отримали p прямих опуклої оболонки, вершинами якої є крайні точки кожної з p прямих, записані у p множинах $S[1...p]$, кожна з яких містить лише дві крайні точки. Нескладно записати процедуру wersh, що визначає, у якому порядку $wer[1...p]$ розташовані вершини на опуклій оболонці. Спочатку перепишемо у два масиви L і PR кінці кожної зі сторін оболонки.

```
Pp:=0;
for j:=1 to p do
begin
  q:=0;inc(pp);
  for k:=1 to n do
  if k in s[j]
  then begin
         inc(q);
         if q=1 then l[pp]:=k else pr[kk]:=k
       end;
end,
```

Вважатимемо, що:

```
wer[1]:=l[1];wer[2]:=pr[1];ss:=[1];nom:=2;
repeat
  j:=1;
  while (j<=p) do
  begin
    if not(j in ss)
    then begin
           if l[j]=wer[nom]
```

```
            then  begin
                    inc(nom);wer[nom]:=pr[j]; ss:=ss+[j]
                  end
            else begin
                  if pr[j]=wer[nom]
                  then begin
                         inc(nom);wer[nom]:=l[j];
                         ss:=ss+[j]
                       end
                  else inc(j)
                end;
        end;
end;
until nom=p.
```

Тепер нескладно переписати в масиви $xx[1...p]$ та $yy[1...p]$ координати x, y всіх p послідовних вершин опуклої оболонки:

```
for k:=1 to p do
begin xx[k]:=x[wer[k]];yy[k]:=y[wer[k]] end.
```

Ми зробили це для того, щоб обчислити площу pl многокутника опуклої оболонки за відомим алгоритмом:

```
pl:=0;
for i:=1 to p do
pl:=pl+1/2*(xx[i+1]-x[i])*(yy[i+1]+yy[i]);
pl:=abs(pl).
```

Задачу розв'язано.

Відомо, що жовтий і синій кольори разом утворюють зелений. Саме ця особливість лежить в основі наступної задачі, і тому назва її така:

ЗЕЛЕНА ПЛЯМА

(О-1999)

Два прожектори утворюють на стіні дві плями трикутної форми, одну — блакитного, а другу — жовтого кольору. Визначити площу плями зеленого кольору, яка утворилася внаслідок накладання двох плям — блакитної і жовтої, та форму (кількість вершин) зеленої плями. (Площу обчислити з двома знаками після коми.)

input.txt	2 2 2 4 5 3 1 7 1 1 5 3	Два рядки — координати вершин двох плям
output.txt	3.00 3	1-й рядок — площа, 2-й рядок кількість вершин зеленої плями

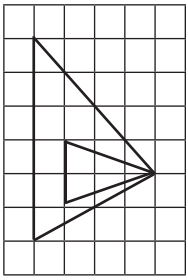
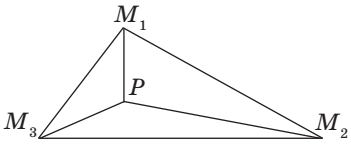


Рисунок до файла даних input.txt

Розв'язання

Зрозуміло, що пляма зеленого кольору матиме не більше ніж 6 вершин, якщо вона взагалі існує. Кожна із цих вершин — точка перетину сторін однієї плями зі сторонами іншої (якщо точка є внутрішньою точкою сторін, що перетинаються) або вершина однієї з плям, яка належить іншій. Вкажемо, як побудувати програму для задачі.

Спочатку розглянемо, як можна визначити, чи належить деяка точка $P(a,b)$ трикутнику з вершинами $M_1(p[1],q[1])$, $M_2(p[2],q[2])$ та $M_3(p[3],q[3])$.



Найпростіший спосіб такий. Розглянемо трикутники M_1PM_2, M_1PM_3, M_2PM_3 й обчислимо площу кожного з них.

Легко зрозуміти, що лише в тому випадку, коли P належить трикутнику, сума площ цих трьох трикутників дорівнюватиме площі заданого трикутника $M_1M_2M_3$.

Нагадаємо, що площу будь-якого трикутника легко обчислити, знаючи координати його вершин:

```
Function plo(u[1],v[1],u[2],v[2],u[3],v[3]:real):real;
var k:Integer;s:real;
begin
  u[4]:=u[1];v[4]:=v[1];s:=0;
  for k:=1 to 3 do
  s:=(u[k]-u[k+1])*(v[k]+v[k+1]);
  s:=abs(s)/2;
  plo:=s
end.
```

Зрозуміло, що площі трикутників M_1PM_2, M_1PM_3, M_2PM_3 та $M_1M_2M_3$ — це значення функції plo при таких значеннях аргументів відповідно:

$p[1]$, $q[1]$, $p[2]$, $q[2]$, $p[3]$, $q[3]$

$p[1]$, $q[1]$, $p[2]$, $q[2]$, a, b

$p[1]$, $q[1]$, $p[3]$, $q[3]$, a, b

Зауважимо, що, працюючи в Паскалі з величинами типу real, треба бути особливо уважним, перевіряючи рівність двох величин.

Через те що площі трикутників обчислені за формулою (а не задані!), враховуючи розрядну сітку, замість рівності слід перевіряти вимогу, щоб вони відрізнялися одна від одної на дуже малу величину (скажімо, меншу за 0,000001).

Тому, якщо `abs(plo(p[1],q[1],p[2],q[2],p[3],q[3]) - plo(p[1],q[1],p[2],q[2],a,b) - plo(p[1],q[1],p[3],q[3],a,b) - plo(p[2],q[2],p[3],q[3],a,b))<0.000001`, то це й означатиме, що точка $P(a,b)$ належить трикутнику $M_1M_2M_3$.

Як визначити точки перетину двох заданих відрізків M_1M_2 і M_3M_4, якщо їх кінці M_k мають координати відповідно $(p[k],q[k])$ $k=1...4$? Для цього складемо рівняння $ax+by=c$ прямих, яким належать відрізки, тобто визначимо коефіцієнти a, b, c. Для цього скористаємося такою процедурою:

```
procedure abc(u1,v1,u2,v2:real);
begin
  a:= v1-v2; b:=u2-u1; c:=a*u1+b*v1
end.
```

Застосовуючи *abc* двічі, отримаємо два рівняння типу:

$a1*x+b1*y=c1$ (для відрізка M_1M_2),

$a2*x+b2*y=c2$ (для відрізка M_3M_4).

Зрозуміло, що єдина точка перетину відрізків (якщо вона існує!), повинна задовольняти цим двом рівнянням і, крім того, бути внутрішньою (а не кінцевою!) точкою цих відрізків.

Для виконання першої з умов потрібно, щоб ці рівняння задовольняли умові: їх коефіцієнти при x і y не повинні бути пропорціональними (прямі не повинні бути паралельними), тобто $a1*b2\neq a2*b1$. Якщо ця вимога не виконується, можемо зробити висновок, що сторони не перетинаються (хоча, наприклад, одна зі сторін може містити кінець іншої сторони, а іноді — й обидва кінці; цей випадок буде зафіксований в іншій перевірці, адже тоді одна з вершин однієї плями належатиме трикутнику іншої плями!).

Координати єдиної точки перетину (u,v) в разі виконання умови $a1*b2\neq a2*b1$ можна обчислити, розв'язавши систему вказаних двох рівнянь:

$a1*x+b1*y=c1,$

$a2*x+b2*y=c2.$

Обчислимо величини: `d:=a1*b2-a2*b1, dx:=c1*b2-c2*b1, dy=a1*c2-a2*c1`. Оскільки $d\neq 0$, матимемо:

```
u:=dx/d; v:=dy/d.
```

Залишилося перевірити другу вимогу: точка (u,v) повинна бути внутрішньою точкою кожного з двох відрізків, тобто число u повинно бути між $p[1]$ і $p[2]$, а також між $p[3]$ і $p[4]$, а число v — між $q[1]$ і $q[2]$, а також між $q[3]$ і $q[4]$.

Щоб здійснити цю перевірку, складемо функцію vnutri типу boolean, яка визначає, чи є внутрішньою точка (xx,yy), що лежить на прямій, проведеній через задані дві точки:

```
function vnutri(u1,v1,u2,v2,xx,yy:real):Boolean;
var min,max:real;f:Boolean;
begin
  f:=true;
  if u1=u2
  then begin
        if v1<v2
        then begin min:=v1;max:=v2 end
        else begin min:=v2;max:=v1 end;
        f:=f and (yy<max) and (yy>min)
      end
  else begin
        if u1<u2
        then begin min:=u1;max:=u2 end
        else begin min:=u2;max:=u1 end;
        f:=f and (xx>min) and (xx<max)
      end;
  vnutri:=f
end.
```

Таким чином можна обчислити всі точки перетину кожної із шести сторін однієї плями з трьома сторонами іншої плями і визначити, які з точок перетину прямих є внутрішніми точками сторін.

Знайдені таким способом точки двох типів (вершини однієї плями, що потрапили на іншу, а також внутрішні точки перетину сторін) можна перевірити (хоч це і не обов'язково!), чи є вони різними. Кількість цих точок і є відповіддю задачі (числом між 0 і 6).

Але знайдені N вершин майбутньої зеленої плями нами не впорядковані (адже ми не впевнені в тому, що вони послідовні). Для цього потрібно буде відшукати (при $N>3$) опуклу оболонку знайденої нами множини точок площини і вказати порядок їх розташування на цій оболонці. Як це робиться, ретельно викладено нами в попередній задачі «операція».

Згадаймо, що нам потрібно також обчислити і площу зеленої плями, тобто площу деякого N-кутника (при $N>2$), користуючись схожим з plo алгоритмом:

```
function greenplo(u,v:mas):real;
var k:Integer;s:real;
begin
     u[N+1]:=u[1];v[N+1]:=v[1];s:=0;
   for k:=1 to N do
   s:=(u[k]-u[k+1])*(v[k]+v[k+1]);
   s:=abs(s)/2;
   plo:=s
end.
```

Тут $u[i]$, $v[i]$ ($i=1\ldots N$ і) — координати N послідовних вершин многокутника зеленої плями, а $N>2$ — їх кількість).

Тепер читач, сподіваємося, вже має можливість скласти відповідну програму мовою Паскаль.

ЛІНІЙКА

(О-1999)

Металеву лінійку завдовжки 2^N сантиметрів і шириною 1 см, розбиту на одиничні клітинки з позначками, загинають навпіл N разів, кожного разу загинаючи праву частину під низ. Який номер мала позначка у K-ій від початку лінійки клітинці, якщо після останнього загинання всі клітинки від верхньої до нижньої виявилися пронумерованими підряд числами $1\ldots2^N$ ($N=1\ldots6$)? (Товщина лінійки настільки мала, що нею можна знехтувати.)

input.txt	2 3	Значення N і K
output.txt	3	Номер позначки

Розв'язання

Позначки $a[1\ldots2^N]$, поставлені на лінійці, — це деяка перестановка чисел $1\ldots2^N$. Для невеликих значень $N=1\ldots3$ легко здогадатися, що позначки повинні бути такими:

```
N=1: 1 2
N=2: 1 4 3 2
N=3: 1 8 5 4 3 6 7 2
```

Вже для цих трьох значень N помічаємо дві цікаві закономірності:

Перша: якщо $a[k]=i$, то $a[i]=k$ (k, $i=1\ldots2^N$).

Друга: якщо позначки $a[1\ldots2^N]$ вибрати такими, щоб $a[k]=i$ ($k=1\ldots2^N$), то після N перегинань одне під одним опиняться саме числа 1 2, 1 4 3 2, 1 8 5 4 3 6 2… Ця друга властивість дає можливість

намітити шлях до розв'язання: щоб знайти потрібну нам нумерацію, треба спочатку в позначки вписати їх індекси, а потім N разів перегнути лінійку, слідкуючи за тим, які позначки опиняються одна під одною в кожному з рядків. Тож спробуємо перегинати N разів лінійку з позначками $1...2^N$ і кожного разу записувати у деяку нову матрицю $aa\left[1...2^t,1...2^{N-t}\right]$ позначки на лінійці, що розташовані одна під одною (тут $t=0...N$ — номер перегинання, перший індекс $y=1.2^N$).

Очевидно, при $t=0$ ми матимемо всього однорядкову «матрицю» із числами $aa[1,k]=a[k]$, де ($k=1...2^t$) — номер рядка, а другий ($x=1...2^{N-t}$) — номер стовпця.

Спробуємо прослідити, що відбувається з позначками в результаті перегинання, тобто встановимо закономірність, яка дасть змогу, знаючи значення $aa[y,x]$ на деякому t-ому кроці (тут $t=0...N-1$) до перегинання, встановити значення $aa[y,x]$ в новій матриці, яку отримаємо після перегинання.

Позначимо через rd і kl кількість рядків і стовпців (колонок) у матриці після t-ого перегинання. Якщо після перегинання елемент $aa[y,x]$ займе у новій матриці позицію $[yy,xx]$, то при $x=1...kl$ він залишиться в тому ж рядку і стовпці (тобто: $yy=y$, $xx=x$), а інакше індекси його зміняться за правилом: перший рядок стане останнім, але у зворотному порядку, другий — передостаннім у зворотному порядку, третій — третім від кінця у зворотному порядку тощо, тому y-ий рядок стане $(rd+1-y)$-им рядком, у цьому рядку його номер з кінця буде $(x-kl)$, а оскільки в цьому рядку всього kl елементів, то остаточно матимемо: $xx=2*kl+1-x$ і $yy=rd+1-y$.

Нехай $st(u)$ — функція, що обчислює значення 2^u:

```
Funcrion st(u:Integer):LongInt;
var s,k:LongInt;
begin s:=1; for k:=1 to u do s:=s*2; st:=s End;
```

Знаючи значення N, матимемо:

```
For k:=1 to st(N) do aa[1,k]:=st(k).
```

Процедура перегинання лінійки може бути записана алгоритмом:

```
Procedure krok(m:Integer);
begin
  rd:=st(m);{рядків після перегинання}
  kl:=st(N-m){колонок після перегинання};
    for y:=1 to rd div 2 do
```

```
    for x:=1 to kl*2 do
    begin
      if x<=kl
      then begin b[y,x]:=aa[y,x] end
      else begin
             b[rd+1-y,2*kl+1-x]:=aa[y,x]
           end;
    end;
    aa:=b;
end.
```

Тому основна програма виглядатиме дуже просто:

```
begin
  Clrscr;write('N,k=');read(N,kk);
  for k:=1 to st(N) do aa[1,k]:=st(k);
  for t=1 to N do krok(t);
  writeln('Позначка:',b[kk,1]);
  readkey
end.
```

КАБЕЛЬНЕ ТЕЛЕБАЧЕННЯ

(О-2000)

N будинків з'єднані мережею кабельного телебачення найкоротшої довжини. Вкажіть довжину мережі та номери всіх будинків, де можна розмістити студію так, щоб відстань від неї по мережі до найдальшого з будинків була найменшою. ($N < 100$, координати будинків — цілі числа з проміжку $[-100,100]$. Довжину мережі подати з точністю до 0.1).

input.txt	4 10 20 20 10 20 0 10 10	Значення N N рядків — по два числа: координати будинків
output.txt	30,0 2 4	Довжина мережі Номери будинків для студії

Розв'язання

Розіб'ємо задачу на дві частини: перша — пошук найкоротшої мережі, друга — пошук місця для студії. Перша розрахована на знання класичного алгоритму, відомого в інформатиці як алгоритм Прима-Краскала. Іноді його називають «дерев'яним». Ця назва

походить від слова «дерево» (саме так називають зв'язний граф без циклів). У цьому алгоритмі викладено побудову дерева найкоротшої довжини, що сполучає N заданих на площині вершин. Щоб докладніше ознайомитися з графами, радимо читачам спочатку переглянути розміщений у кінці книжки нарис про графи, який надалі можна використовувати, як довідковий матеріал.

Оскільки кожен будинок під'єднаний до мережі, почнемо будувати її з будинку № 1.

Зрозуміло, що мережа від цього будинку йде далі в будинок, найближчий до будинку № 1. Нехай це будинок із номером $k1$. Тож одне з ребер майбутнього дерева (від $i1=1$ до $k1$) ми вже побудували. Нагадаємо, що в дереві з N вершин всього $N-1$ ребер. Як визначити кінці ще $N-2$ ребер?

Позначимо через YES множину номерів будинків, уже під'єднаних до мережі. Після першого кроку YES = [1,k1]. Серед будинків, не під'єднаних до мережі, відшукаємо той (із номером $k2$), відстань від якого до одного з будинків з YES (з номером $i2$) найкоротша. Після цього множина YES зміниться: YES:=YES+[k2], а в графі з'явиться нове ребро – між $i2$ та $k2$. І так продовжуватимемо доти, доки YES не набере значення $[1...N]$.

Якщо через dov(u,v) позначити відстань між будинками з номерами u і v, а в 1/0-матриці d вважати $d[u,v]=1$, коли будинки u і v з'єднані ребром, і $d[u,v]=0$ — коли не з'єднані, тоді ця частина програми мовою Паскаль виглядатиме так:

```
For i:=1 to N do
For k:=1 to N do D[k,i]:=0;
YES:=[1];
repeat
  Min:=maxint;
  for i:=1 to N do
  for k:=1 to N do
  if (i in YES) and not(k in YES) and
  (dov(i,k)<min)
  then begin ii:=i;kk:=k;min:=dov[i,k];
             YES:=YES+[ii];
             D[ii,kk]:=1; D[kk,ii]:=1
       end;
until YES=[1..N].
```

Вказана в тексті функція dov(u,v) обчислює довжину відрізка між двома точками за теоремою Піфагора:

```
function dov(u,v:Integer):real;
var dx,dy:real;
```

```
begin
  Dx:=x[v]-x[u];dy:=y[v]-y[u];
  Dov:=sqrt(sqr(dx)+sqr(dy))
end.
```

Першу частину задачі ми вже розв'язали. Залишилося визначити місце для студії. Доведеться використати інший простий класичний алгоритм для графів, який належить Флойду і Уоршелу. У його основі лежить правило «Якщо в обхід ближче, ніж напряму, йди в обхід».

Але спочатку необхідно перетворити матрицю сполучень d типу (1/0) на матрицю відстаней по мережі dd, вважаючи, що при $i \neq k$ і $d[i,k]=0$ (тобто коли вершини i та k не сполучені ребром) ребро між i та k все-таки існує, але воно має нескінченно велику довжину. Оскільки нескінченність не є числом, достатньо в новій матриці dd (dd:=d) всі нулі, крім тих, що на головній діагоналі, замінити будь-яким великим числом (наприклад, maxint), а всі одиниці — відстанями dov(i,k), де dov(I,k) — довжина відрізка між точками з індексами i та k, яку легко обчислити за теоремою Піфагора, знаючи координати кожної з точок. Нагадаємо, що в новій матриці dd елементи матимуть тип real:

```
dd:=d;
for k:=1 to N do
for i:=1 to N do
if (k<>i)
then
  case dd[I,k] of
    0:dd[I,k]:=maxint;
    1:dd[I,k]:=vid(I,k)
  end.
```

До нової матриці dd вже можна застосувати алгоритм Флойда-Уоршела:

```
for k:=1 to N do
for i:=1 to N do
for j:=1 to N do
if ((i-j)*(i-k)*(j-k)<>0) {вершини різні} and
(dd[k,i]+dd[i,j]<dd[k,j]) (в обхід ближче)
Then begin
        dd[k,j]:=dd[k,i]+dd[i,j];
        dd[j,k]:=dd[k,j] {йди в обхід}
end.
```

Ми отримали матрицю *dd*, яка визначає найкоротші відстані по мережі між будь-якими двома будинками. Тому, щоб визначити місце для студії, залишилося зробити останній крок — вибрати такий номер будинку, відстань від якого до найдальшого з будинків є найменшою. Це й буде одним із можливих місць для студії.

У наведеній програмі легко зрозуміти, як визначити перелік усіх тих будинків, у яких можливо облаштувати студію:

```
uses crt;
var
     j,i,k,n,ii,kk: Integer;
     min,max,s: real;
     x,y,z: array[1..100] of real;
     d,dd: array[1..30,1..30] of real;
     yes,sw: set of Byte;
     dat: Text;

Function dov(u,v:Integer):real;
var dx,dy:real;
begin
  Dx:=x[v]-x[u];dy:=y[v]-y[u];
  Dov:=sqrt(sqr(dx)+sqr(dy))
end;
{--------------Основна програма---------------}
begin
clrscr;
assign(dat,'c:\tv1.dat'); reset(dat);
readln(dat,N);
for k:=1 to n do readln(dat,x[k],y[k]);
close(dat);
for i:=1 to N do
for k:=1 to N do D[k,i]:=0;
YES:=[1]; s:=0; min:=maxint;
repeat
  for i:=1 to N do
  for k:=1 to N do
  if (i in YES) and not (k in YES)
  and (dov(I,k)<min)
  then begin ii:=i; kk:=k; min:=dov(i,k); end;
  YES:=YES+[kk]; s:=s+min;
  min:=maxint;
  D[ii,kk]:=1; D[kk,ii]:=1
until YES=[1..N];
for k:=1 to n do
```

```
begin
  for i:=1 to n do
  write(d[k,i]:1:0,' ');
  writeln;
end;
dd:=d; {dd — матриця сполучень}
  for k:=1 to N do
  for i:=1 to N do
  if (k<>i)
  then
    begin
      if d[i,k]=0
      then dd[i,k]:=maxint
      else begin dd[i,k]:=dov(i,k);
            dd[k,i]:=dov(i,k)
           end
    end;
for k:=1 to N do
for i:=1 to N do
for j:=1 to N do
if ((i-j)*(i-k)*(j-k)<>0) {вершини різні} and
(dd[k,i]+dd[i,j]<dd[k,j]) {в обхід ближче)}
Then begin
       dd[k,j]:=dd[k,i]+dd[i,j];
       dd[j,k]:=dd[k,j] {йди в обхід}
     end;
min:=maxint;
for k:=1 to N do
for i:=1 to N do
if (i<>k) and (dd[i,k]<min)
Then begin kk:=k;min:=dd[i,k] end;
writeln;
writeln(s:5:1);
for k:=1 to n do
begin
  max:=0;
  for i:=1 to n do
  if (k<>i) and (dd[k,i]>max) then max:=dd[k,i];
  z[k]:=max;{відстані до найдальшого будинку}
end;
  min:=maxint;
  for k:=1 to n do
  if z[k]<=min then min:=z[k];
  sw:=[];{можливі місця для студії}
  for k:=1 to n do
```

```
  if z[k]=min then sw:=sw+[k];
for k:=1 to n do
if k in sw then write(k,' ');
readkey;
end.
```

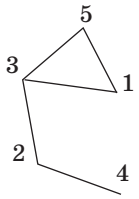
Далі ми розглянемо ще декілька задач на графи.

ВАЛЮТА

(О-2007)

У місті відкрито N відділень банку OLIMP, — одне від одного на відстані 1 км, але лише в одному з них (A) зберігаються валютні резерви, які за попереднім замовленням розвозять в інші відділення банку. Знаючи розташування цих банків, схему M шляхів сполучення та перелік банків, які замовили валюту, встановити довжину найкоротшого пробігу машини, що розвозить валюту, якщо після розвезення вона повертається в банк A. (N, $M=1...31$)

Приклади вхідного та вихідного файлів

input.txt			output.txt	
5 5 3 3 5 1 5 3 2 2 4 1 3 1 4 5	Значення N, M, A M рядків: у рядку $k=1...M$ номери двох банків, між якими є шлях сполучення Один рядок: список банків, які замовили валюту		7	Довжина найкоротшого пробігу
			 Схема до тесту input.txt	
Пояснення: Можливий маршрут найкоротшого пробігу 3, 5, 1, 3, 2, 4, 2				

Розв'язання

Це знову задача на графи.

Скориставшись відомим алгоритмом Флойда-Уоршела, визначимо найкоротші відстані між кожними двома банками.

У вказаному файлі input.txt цей алгоритм побудує таку матрицю відстаней:

0 2 1 3 1
2 0 1 1 2
1 1 0 2 1
3 1 2 0 3
1 2 1 3 0

Кожен маршрут розвезення валюти — це послідовність номерів банків із множини замовлень S (нехай цих замовлень Z). Тому спадає на думку (мабуть, не найпростіший) спосіб. Будуємо всі можливі перестановки номерів із множини S, які містять Z елементів, і пробуємо пройти цими шляхами, підраховуючи кожного разу довжини відстаней, доки не буде знайдено маршрут, який починається з банку L, проходить через всі зазначені банки та відстань на якому в сумі зі зворотним шляхом дає найменше число. До речі, цей спосіб не передбачає застосування алгоритма Дейкстри.

Ось як виглядатиме програма для цього останнього способу. (R — матриця відстаней після застосування алгоритму Флойда-Уоршела до масиву d; S — множина замовлень).

```
Uses crt;
var min,x,y,k,i,n,m,j,a,dl,q,qq:LongInt;
    d,r:array[1..50,1..50] of LongInt;
    p,pp:array[1..50] of Integer;
    s,ss:set of Byte;
    nt:string;
    f:Text;
begin
  clrscr;
  for i:=1 to n do for k:=1 to n do d[i,k]:=0;
  writeln('Номер тесту ');readln(nt);
  assign(f,'bank'+nt+'.dat');reset(f);
  readln(f,n,m,a);
  {n — банків, m — доріг, a — № банку валюти }
  for i:=1 to m do
  begin
    readln(f,x,y);d[x,y]:=1;d[y,x]:=1
  end;
  s:=[];{s — множина замовлень}
  repeat read(f,x);s:=s+[x] until eoln(f);
  close(f);
  s:=s-[a];
  {матриця доріг}
  writeln('Дороги:');
  for i:=1 to n do
  begin
  for k:=1 to n do write(d[i,k],' ');
  writeln
end;
  {алгоритм Флойда-Уоршела}
```

```
  {матриця відстаней}
writeln('Відстані:');
r:=d;
writeln;
for i:=1 to n do for k:=1 to n do
if (i<>k) and (r[i,k]=0) then r[i,k]:=1000;
for i:=1 to n do
for j:=1 to n do
for k:=1 to n do
if r[i,j]+r[j,k]<r[i,k]
then begin
      r[i,k]:=r[i,j]+r[j,k];
      r[k,i]:=r[i,k]
     end;
for i:=1 to n do
begin
  for k:=1 to n do write(r[i,k],' ');
  writeln
end;
writeln;
{Перебирання}
min:=maxint;
for i:=1 to n do
begin
  if i in s
  then begin {старт} q:=1;p[1]:=i;ss:=[i];
       dl:=r[a,i];k:=1;
       while q>0 do
       begin
         if s<>ss
         then begin
                if k<=n
                then begin
                       if (k in s) and
                          ([k]*ss=[])
                       then begin {крок}
                              inc(q);p[q]:=k;
                              ss:=ss+[k];
                              dl:=dl+r[k,p[q-1]];
                              k:=0
                            end
                     end
                else begin {крок назад}
                       k:=p[q];ss:=ss-[k];
                       dl:=dl-r[k,p[q-1]];
```

```
                        dec(q)
                      end
                  end
                else begin
                      if dl+r[p[q],a]<min
                      then begin
                            min:=dl+r[p[q],a];
                            pp:=p;qq:=q end;
                            k:=p[q];
                            ss:=ss-[k];
                            dl:=dl-r[k,p[q-1]];
                            dec(q)
                           end;
            inc(k)
          end;
        end;
    end;
    writeln('Мінімальний пробіг: ',min);
    write('Оптимальний шлях: ');
    for k:=1 to qq do write(pp[k],' ');writeln;
    readkey
end.
```

Читач зрозумів, що існує, мабуть, більш економне розв'язання, не пов'язане з перебиранням усіх перестановок чисел з множини замовлень, але для новачків у програмуванні за невеликої кількості замовлень вказаний вище спосіб теж дає хороший результат, тим більше що метод «віток і меж» відкине непотрібні та малоефективні перестановки.

ГРА В КОСТІ

(О-2002)

Двоє кидають N гральних костей, а потім, ходячи по черзі, перевертають одну з костей на сусідню грань так, щоб збільшити кількість очок на її верхній грані. Програв той, хто не зміг цього зробити. Вважайте, що ви ходите першим і граєте із суперником, який не помиляється. У заданій ситуації зробіть будь-який виграшний хід (якщо він існує) або здайтесь. ($N \le 100000$. Сума очок на протилежних гранях кості дорівнює 7).

Відповідь подайте у вигляді:

0 — якщо ситуація програшна;

X Y — якщо ситуація виграшна (тобто кость, на верхній грані якої X очок, повернути доверху гранню, на якій Y очок).

Приклади вхідного та вихідного файлів

input.txt	3 1 4 1	Значення N Один рядок: N чисел — кількості очок на верхніх гранях костей
output.txt	4 6	Відповідь

Розв'язання

Це досить цікава і проста задача, що відкриває цикл задач ігрового типу. Про методи розв'язування таких задач читач може прочитати у додатках в кінці книжки (див. «Хто не зміг, той програв»). Ми ж наведемо для цієї задачі просте і доступне розв'язання, яке не потребує спеціальних знань, окрім кмітливості та вміння аналізувати ситуацію.

Якщо на верхніх гранях усіх костей випало по 6 очок, ми програли (відповідь 0).

Нехай маємо усього одну кость, на верхній грані якої x очок ($x < 6$). Тоді при $x > 1$ матимемо виграшний хід x 6 (тобто повернути кость догори гранню 6). Якщо ж кость повернута догори гранню, на якій 1, тоді ми не можемо повернути її гранню 6, адже сума очок на паралельних гранях кості дорівнює 7. Тому ми зможемо повернути її догори гранню, на якій y очок, де $y = 2 \ldots 5$. Але це означатиме, що наш суперник має змогу повернути її догори гранню 6, тобто ми програли.

Нехай маємо дві кості, на яких однакові кількості очок, скажімо 2 і 2. Який би хід ми не зробили, суперник має можливість зробити з іншою костю той самий хід. У результаті за декілька пар ходів ми матимемо дві кості 6 і 6, що означатиме, що ми програли. Отже, пари однакових костей можна відкидати (це програші).

Але програшною для нас буде і ситуація, якщо на деяких костях 6 очок, а на решті 1.

Якщо ж не всі кості одиниці і шістки, можливі виграші, тобто всі шістки та одиниці можна відкинути (якщо при цьому не залишиться костей — ми програли). Зрозуміло, що випадок, коли ми маємо пару 2 і 5 (2 + 5 = 7) або 3 і 4 (3 + 4 = 7), — це теж програш. Тому слід відкидати пари костей типу 2 і 5, або 3 і 4, адже кожна така пара програшна.

До того ж взагалі не потрібно рахувати, скільки серед костей четвірок, п'ятірок (шістки та одиниці ми вже відкинули). Достатньо порахувати лише кількість двійок та трійок, хоча слід запам'ятати, чи були взагалі двійки, трійки, четвірки чи п'ятірки.

Величезна кількість костей (до 100 000) не дає змоги записати масив мовою Паскаль і завести масиву. А цього взагалі не потрібно

робити. Ось як обійтися без масиву. Подані нижче фрагменти допоможуть скласти програму мовою Паскаль, яка розв'язує нашу задачу.

```
Clrscr;
write('Номер тесту ');readln(nt);
{nt типу string}
assign(fd,'dice'+nt+'.dat);reset(fd);
readln(fd,N);{кількість костей}
k2:=0;k3:=0;{тип LongInt}
repeat
   read(fd,x); {x з діапазону 1..6}
   if x in[2,5]
   then inc(k2)
   else if x in[3,4] then inc(k3);
  F2:=(x=2);F3:=(x=3);F4:=(x=4);
  F5:=(x=5);
until eof(fd);
close(fd);
k2:=k2 mod 2; k3:=k3 mod 2;
{відкидаємо пари однакових костей та пари костей із
сумою 7}
assign(fs,'dice'+nt+'.sol');rewrite(fs);
Case k2+k3 of
     0: writeln(fs,'0');
     1: if k2=1
        then begin
               if F2 {були двійки}
               then writeln(fs,'2 6')
               else writeln(fs,'5 6')
                    {були п'ятірки}
            end
        else if k3=1
             then begin
                    if F3{були трійки}
                    then writeln(fs,'3 6')
                    else writeln(fs,'4 6')
                         {були четвірки}
                  end;
     2:if F2 and F3{були двійки і трійки}
       then writeln(fs,'2 3')
       else if F5 and F3
            {були п'ятірки і трійки}
            then writeln(fs,'3 5')
            else if F2 and F4
                 {були двійки і четвірки}
```

```
                then writeln(fs,'2 4')
                else writeln(fs,'4 5')
              {були четвірки і п'ятірки}
end;
close(fs).
```

Задачу розв'язано.

ДОМІНО

(0-2000)

Двоє робітників фабрики, що виготовляє доміно, маючи вдосталь кожної з 28 костей, грають в обідню перерву в таку гру. На початку гри вони навмання відкривають на столі декілька костей (не більше за 10 000) і ходять по черзі. За один хід можна зняти яку-небудь з костей (x,y), замінивши її на іншу (x,z), якщо $z>y$. Той, хто не зміг цього зробити, програв.

Яку б з костей зняли ви і якою її замінили б, якщо хочете виграти? (Якщо виграти неможливо, вивести 0.)

input.txt	1 3 6	Кількість костей N N рядків по 2 числа — кількості очок на кожній половині кожної з костей
output.txt	3 6 6 6	Яку кость зняти Якою замінити

Розв'язання

Той, хто переглянув розміщений у кінці книжки нарис автора задачі «Хто не зміг, той програв», у якому він пропонує свій погляд на задачі, пов'язані з іграми, зуміє зрозуміти викладені нижче міркування.

Назвемо кость x, y похідною для кості x, z, якщо $y>z$, і позначатимемо цей факт так: $x,z \to x,y$ (при $y>z$).

Кость 6,6 не має похідних. Тобто 6,6 — програш. Домовимося кожну програшну ситуацію вважати такою, що має стан 0. Усі ситуації, стан яких більший від нуля, вважатимемо виграшними. Покажемо, як обчислити стани всіх 28 костей доміно.

Для кості 5,6 є похідна 6,6 (вона має стан 0). Найменше невід'ємне ціле, якого немає серед станів похідних, — це число 1. Тому для кості 5,6 матимемо стан 1.

Який стан матиме кость 4,6? Похідні для кості 4,6 будуть:

5,6 (стан 1), 6,6 (стан 0). Найменше невід'ємне ціле, якого немає серед станів похідних, буде 2. Тому кость 4,6 має стан 2.

Аналогічно для кості 3,6 похідні будуть такими:

4,6 (стан 2), 5,6 (стан 1), 6,6 (стан 0). Тому стан 3,6 буде 3.

Легко отримати стани костей 2,6 (стан 4), 1,6 (стан 5), 0,6 (стан 6).

Обчислюючи кожного разу для кожної з інших костей їх похідні та визначаючи найменше з чисел 0, 1, 2,..., якого немає серед станів похідних, ми знатимемо стани кожної з костей. Матимемо таку таблицю (обчислення не наводяться):

6,6	5,6	4,6	3,6	2,6	1,6	0,6
0	1	2	3	4	5	6

5,5	4,5	3,5	2,5	1,5	0,5	4,4
0	3	2	5	4	7	0
3,4	2,4	1,4	0,4	3,3	2,3	1,3
1	6	7	4	0	7	6

0,3	2,2	1,2	0,2	1,1	0,1	0,0
5	0	1	2	0	3	0

Щоб похідні для кожної з 28 костей складати не «вручну», цю роботу можна доручити комп'ютеру, для чого доведеться скласти мовою Паскаль невеличку програму, яка обчислюватиме стани всіх костей.

Наводимо зразок такої програми.

```
Procedure stany;
var u,v: Integer;
begin
  For u:=6 downto 1 do
  For v:=u downto 0 do
  begin
    S:=[];
    if u<6
    then for k:=u+1 to 6 do s:=s+[st[k,v]];
    if v<6
    then for k:=v+1 to 6 do s:=s+[st[u,k]];
    k:=0; while k in s do inc(k);
    st[u,v]:=k;st[v,u]:=k;
  end;
end.
```

Стан ситуації, що складається з декількох костей, обчислюється як xor станів, у яких перебувають ці кості.

Розглянемо декілька прикладів.

Перший приклад

Дві кості 1,4 та 3,3. Їх стани 7 та 0. Спробуємо для кості 1,4 знайти похідну зі станом 0 або для кості 3,3 — похідну зі станом 7. Перший виграшний хід $1,4 \to 4,4$ (стан 0). Тоді ми залишимо супернику ситуацію зі станом 0 xor 0=0, тобто програш. Другий виграшний хід матимемо, якщо для кості 3,3 відшукаємо похідну зі станом 7. Але її немає.

Другий приклад

Три кості 2,1 0,5 4,6. Стани костей: 7 2 1. Стан ситуації дорівнює 7 xor 2 xor 1=5 xor 1=4>0. Ситуація виграшна, тому існує деякий виграшний хід. Який це хід? Його можна відшукати таким чином: для кожної з костей, що складають ситуацію (2,1 0,5 та 4,6 зі станами 1, 7 та 2), відшукаємо всі похідні. Для 2,1 маємо (в дужках зазначено стани похідних):

$2,1 \to 2,2(0) 2,1 \to 2,3(7) 2,1 \to 2,4(6) 2,1 \to 2,5(5) 2,1 \to 2,6(4) 2,1 \to$
$\to 3,1(6) 2,1 \to 4,1(7) 2,1 \to 5,1(4) 2,1 \to 6,1(5)$.

Оскільки стан кості 0,5 є 7, а 4,6 має стан 2, легко знайти 7 xor 2=5. Тому потрібний виграшний хід (навіть два таких ходи) повинен приводити до похідної зі станом 5 — це або $2,1 \to 2,5(5)$, або $2,1 \to 6,1(5)$. Чому ці ходи виграшні? В обох випадках ми замінимо кості на їх похідні зі станом 5. 5 xor 5=0, тому після кожного з цих ходів ми залишимо супернику ситуацію зі станом 0 (адже 5 xor 5=0), тобто програш. Аналогічно, вибираючи кость 0,5 (7), ми повинні замінити її похідною, що матиме стан 7 xor 1=6. Це можна зробити ходом $0,5 \to 0,6(6)$. Це ще один виграшний хід. Нарешті, для кості 4,6 потрібно знайти похідну, яка мала б стан 7 xor 2=5. Але такої похідної немає.

Підсумовуючи результати, матимемо відповідь: ситуація 2, 10, 56, 4 має три виграшних ходи —

$2,1 \to 2,5(5)$, або $2,1 \to 6,1(5)$, або $0,5 \to 0,6(6)$.

Третій приклад

5,3 2,1 6,0 3,0. Стани костей 2, 1, 6, 5. Оскільки 2 xor 1 xor 6 xor 5=0, ця ситуація програшна, тому виграшний хід відсутній. Нагадаємо, що в цьому випадку потрібно вивести відповідь 0.

Розглянуті приклади повинні надихнути нас на створення процедури пошуку виграшного ходу для ситуації, що складається з будь-якої кількості костей (у програмі позначено $st[u,v]$ — стан кості u, v, а в двох масивах a і b довжини N записані значення N костей, які складають ситуацію). Нагадаємо, що стан цієї ситуації stn обчислити дуже легко:

```
Stn:=0;
For i:=1 to N do stn:=stn xor st[a[i],b[i]].
```

А ось як виглядатиме пошук усіх виграшних ходів:

```
Procedure hid;
var u,v: Integer;
begin
  h:=0;
  for u:=1 downto N do
  begin
    {стан ситуації без кості з номером u=1..N}
    stnu:=stn xor st[a[u],b[u]];
    {Пошук похідних u-ої кості зі станом stnu}
  if a[u]<6
  then begin
        for k:=a[u]+1 to 6 do
        if st[k,b[u]]=stnu
        then begin
              inc(h);
              write(a[u],' ',b[u],'>');
              writeln(k,' ',b[u])
             end;
      end;
  if b[u]<6
  then begin
        for k:=b[u]+1 to 6 do
        if st[a[u],k]=stnu
        then begin
              inc(h);
              write(a[u],' ',b[u],'>');
              writeln(a[u],' ',k)
             end;
      end;
  end;
  writeln('Виграшних ходів ',h);
  readkey
end.
```

Задачу розв'язано.

Далі викладено розв'язання ще однієї задачі про ігри.

МОДЕЛІ

(О-2006)

Сашко спіткнувся, коли ніс із кабінету математики N однакових трикутних пірамід з рівними гранями, занумерованими числами 1, 2, 3, 4. Збираючи моделі з підлоги, він придумав нову гру, у яку можуть грати двоє, ходячи по черзі.

За один хід одну з фігур можна перевернути на грань із більшим номером. Програв той, хто не зміг зробити ходу. Тепер Сашко часто виграє у своїх друзів.

Спробуйте, якщо зможете, виграти в нього, або здайтесь, якщо виграти неможливо ($N < 10000$). Вкажіть будь-який виграшний хід, якщо ви ходите першим (або 0 0, якщо здаєтесь).

Приклади вхідного та вихідного файлів

input.txt	424	Один рядок: без пропусків числа з номерами граней, на яких стоять фігури
output.txt	2 4	Виграшний хід: фігуру, що стоїть на грані з номером 2, перевернути на грань 4

Розв'язання

Перше утруднення — як прочитати з текстового файла таку велику кількість чисел, які не можна записати навіть у масив? Виявляється, що жодного масиву взагалі не потрібно заводити, якщо зробити детальний аналіз ситуації.

Нехай $N = 1$. Якщо на нижній грані x ($x < 4$), то це виграш ходом $x \to 4$, а при $x = 4$ програш.

Нехай $N = 2$ й обидві фігурки стоять на однакових гранях x. Це програш. Справді, при $x = 4$ це очевидно, а при $x < 4$, якщо ми зробимо хід $x \to y$ ($y > x$), Сашко зробить такий самий хід з іншим тетраедром. Продовжуючи далі, прийдемо через парну кількість ходів до ситуації 4 4, у якій черга ходити нам. А це наш програш.

Зрозуміло, що парна кількість фігурок, що стоять на однакових гранях, — це теж програш. А це означає, що такі фігурки можна не враховувати (вилучити), якщо їх кількість парна. З тих самих міркувань слід не враховувати будь-яку кількість фігурок, що стоять на грані з номером 4.

Якщо не залишилося фігурок, то це програш.

Якщо ж деякі фігурки залишаться, то ми прийдемо до таких випадків: залишилось

- (1 фігура) 1, 2, 3 — це наші виграші (див. вище);
- (2 фігури) 1 2, 1 3, 2 3;
- (3 фігури) 1 2 3.

Видно, що коли фігур дві, ми виграємо ходами відповідно: $1 \to 2$, $1 \to 3$, $2 \to 3$, перевівши гру у дві однакових фігурки. Покажемо, що третій випадок для нас програшний. Для цього переберемо всі можливі ходи першого:

$1 \to 2$. Відповідь Сашка: $3 \to 4$. Ми програли.

$1 \to 3$. Відповідь Сашка: $2 \to 4$. Ми програли.

$1 \to 4$. Відповідь Сашка: $2 \to 3$. Ми програли.
$2 \to 3$. Відповідь Сашка: $1 \to 4$. Ми програли.
$2 \to 4$. Відповідь Сашка: $1 \to 3$. Ми програли.
$3 \to 4$. Відповідь Сашка: $1 \to 2$. Ми програли.
Ось і все.

А тепер повернімося до умови задачі.

Позначимо через $K1$, $K2$, $K3$ кількості фігур, що стоять на гранях із номерами відповідно 1, 2, 3, і прочитаємо текстовий файл.

Мовою Паскаль це виглядатиме так (c — змінна типу char):

```
K1:=0;K2:=0;K3:=0;
repeat
  read(f,c);
  Case c of
  '1':inc(K1);
  '2':inc(K2);
  '3':inc(K3)
  end;
until eoln(f).
```

Оскільки парну кількість будь-яких однакових пірамід можна вилучити, матимемо:

```
K1:=K1 mod 2;K2:=K2 mod 2;K3:=K3 mod 2;
Case (K1+K2+K3) of
0,3: write('0 0');
1:if K1=1 then write('1 4') else if K2=1 then
write('2 4') else write('3 4');
2:if K1=0 then write('2 3') else if K2=0 then
write('1 3') else write('1 2')
end.
```

Виграшний хід (або повідомлення про програш) виводиться на екран монітора. Потрібну програму мовою Паскаль легко складе читач.

Якщо на аркуші в клітинку зафарбувати деякі з клітинок, утворяться фігури. Вважатимемо дві зафарбовані клітинки, що мають спільну сторону, такими, що належать одній фігурі. Наступна задача належить до класичних задач інформатики. Вона теж рахує кількість трьох фігур.

КУБИКИ

(О -2001)

Куб із ребром 10 одиниць заповнений тисячею одиничних кубиків, серед яких деякі дерев'яні, а решта N — металеві

(намагнічені), які притягуються один до одного, якщо мають спільну грань, утворюючи одну фігуру. За наведеними координатами кожного з N металевих кубиків $X[i]$, $Y[i]$, $Z[i]$ визначити:

1) кількість утворених металевих фігур;
2) об'єм і площу поверхні кожної з них.

Вхідні дані записані у файлі input.txt ($N < 250$). Відповідь — у файлі output.txt.

Приклади вхідного та вихідного файлів

input.txt	3 1 1 1 4 1 5 1 1 2	Значення N N рядків по три числа — координати металевих кубиків
output.txt	2 2 1 10 6	Кількість фігур Об'єми фігур Поверхні фігур

Розв'язання

Ця задача, пов'язана з визначенням кількості фігур, належить до класичних задач інформатики і називається в російському варіанті «области». Зазвичай розглядаються фігури на площині, складені з квадратів одного кольору, що мають спільну сторону. Автор лише зробив задачу «тривимірною» і додав вимогу обчислити не тільки об'єм, а й площу поверхні кожної з таких фігур («областей»).

Вважатимемо, що:

1) кубики з'являються (і проявляють свої магнітні властивості) не всі одночасно, а по одному;
2) якщо виявиться, що новий металевий кубик є сусіднім (має спільну грань) з уже розглянутими металевими кубиками (зверху-знизу, зліва-справа або ззаду-спереду), тоді легко порахувати, що додавання цього кубика збільшить площу повної поверхні фігури:

- на 4 кв.одиниці, якщо кількість металевих сусідів дорівнює 1;
- на 2 кв.одиниці, якщо кількість металевих сусідів дорівнює 2;
- на 0 кв.одиниць, якщо кількість металевих сусідів дорівнює 3.

Легко помітити закономірність між зростанням площі (a) і кількістю металевих сусідів (b): $a + 2b = 6$. І дійсно, за кількості сусідніх металевих кубиків 4, 5 або 6 кожний такий кубик зменшуватиме площу повної поверхні, оскільки $a = 6 - 2b$:

- на 2 кв.одиниці, якщо кількість металевих сусідів дорівнює 4;
- на 4 кв.одиниці, якщо кількість металевих сусідів дорівнює 5;
- на 6 кв.одиниць, якщо кількість металевих сусідів дорівнює 6.

Якщо Fig — кількість вже знайдених фігур, то на початку роботи Fig = 0. Позначимо через yes множину номерів тих кубиків, для яких уже визначено, до якої фігури вони належать. Тоді спочатку yes:= [ ].

Будемо шукати, перебираючи номери, ті з кубиків (з номером k), які ще не належать множині yes, але є сусідами одного з кубиків (з номером i) цієї фігури (кубики з номерами i та k мають спільну грань). Тоді зрозуміло, що кубик із номером k теж потрібно віднести до цієї фігури, збільшивши на 1 кількість кубиків фігури kf[fig]. Якщо виявиться, що кубик із номером і має sp сусідніх металевих кубиків, які ще не належать yes, то зрозуміло, що площа поверхні цієї фігури повинна змінитися на величину 6-2*sp.

Повторюватимемо цю операцію доти, поки множина yes не дорівнюватиме $[1...N]$.

Нижче наводимо доволі просту програму, яка розв'язує поставлену задачу.

```
Uses crt;
var  k,i,nom,n,fig,sp:Integer;
     x,y,z,s,kf:array[1..250] of Integer;
     f:array[1..250] of set of Byte;
     dat:Text;
     yes:set of Byte;
Procedure data;
begin
  assign(dat,'c:\cube.dat');reset(dat);
  readln(dat,n);
  for k:=1 to n do readln(dat,x[k],y[k],z[k]);
  close(dat)
end;

{--------Перший кубик нової фігури--------}
function number:Integer;
var j:Integer;
begin
  j:=1; while (j<=n) and (j in yes) do inc(j);
  number:=j
end;
{------------Чи сусідні два кубики----------}
function susid(n1,n2:Integer):boolean;
begin
  if ((x[n1]=x[n2]) and (y[n1]=y[n2]) and
     (abs(z[n1]-z[n2])=1) {зверху/знизу}or
     (x[n1]=x[n2]) and (z[n1]=z[n2]) and
     (abs(y[n1]-y[n2])=1) {спереду/ззаду}or
```

```
      (z[n1]=z[n2]) and (y[n1]=y[n2]) and
      (abs(x[n1]-x[n2])=1)) {зліва/справа}and
      (n1 in yes) and not(n2 in yes) {n1 належить
       фігурі,а n2 ні}
  then susid:=true else susid:=false;
end;
{----------Основна програма----------------}
Begin
  clrscr;
  data;
  Fig:=0; yes:=[ ];
  repeat
    inc(Fig);f[fig]:=[number];kf[fig]:=1;
    yes:=yes+[number];S[fig]:=6;
    for i:=1 to n do
    begin
      sp:=0;
      for k:=1 to n do
      if susid(i,k)
      then begin
             inc(sp);f[fig]:=f[fig]+[k];
             yes:=yes+[k];inc(kf[fig])
           end;
      if sp>0 then s[fig]:=s[fig]+6-2*sp;
  end;
until yes=[1..n];
writeln('Фігур:',fig);
write('Об'єми фігур:');
for k:=1 to fig do write(kf[k]);writeln;
write('Поверхні фігур:');
for i:=1 to fig do write(s[k]);writeln;
readkey
end.
```

ВІРУС

(О-2001)

Жива тканина складена з однакових клітин у формі правильного шестикутника, які без пропусків заповнюють кожну її ділянку. Одну із цих клітин вразив вірус. Щохвилини кожна інфікована клітина може вразити одну зі здорових клітин, сусідніх із нею. Яку найбільшу кількість інфікованих клітин можна отримати за N хвилин (N — натуральне, не більше за 20 000)?

Приклади вхідного та вихідного файлів

virus1.dat	2	Значення N
virus1.sol	4	Кількість інфікованих клітин

Розв'язання

Здається, що це відома задача про ядерний розпад за формулою 2^N. І дійсно, протягом перших 5 хвилин може бути інфіковано саме стільки клітин. Зразок такого поширення вірусу наведений на рисунку внизу, де цифрами 0... 5 позначені вірусні клітини та час народження кожної нової вірусної клітини (наведено лише верхню половину рисунка).

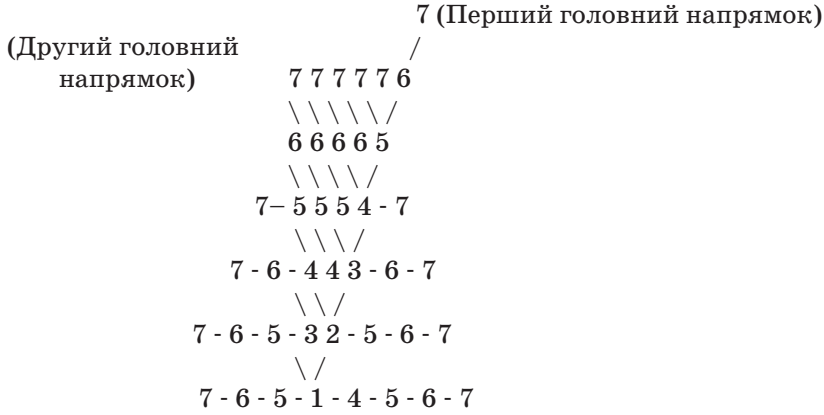
```
          5
          /
    5 5 5 4
    \ \ \ /
    4 4 3
    \ \ /
    5 - 3 2 - 5
    \ /
    5 - 1 - 4 - 5
```

Але вже на шостій хвилині всі намагання одержати навіть не 64 клітини, а хоча б 60 будуть марними. Автор переконався в цьому, втомившись від рисування. Тоді він написав програму, яка перебирає після 5-ї хвилини, коли вже є 32 клітини, всі можливі варіанти.

Їх виявилося чимало, бо програма працювала довго, і найкраще, що вона знайшла, — 54 клітини (програма тут не наводиться, пізніше читач зрозуміє, чому). Але результат роботи програми ми наведемо: ось так розташовані інфіковані клітини в одному з найкращих випадків (подано лише верхню половину рисунка).

```
              6
              /
        6 6 6 6 5
        \ \ \ \ /
        5 5 5 4
        \ \ \ /
6 - 4 4 3 - 6
        \ \ /
6 - 5 - 3 2 - 5 - 6
        \ /
6 - 5 - 1 - 4 - 5 – 6
```

Автор продовжив шукати найкращий варіант для $N=7$. Програма працювала ще довше, але знайшла 82 вражені вірусом клітини, і в одному з найкращих варіантів вони були розташовані так, як зображено на наведеному нижче рисунку:



Придивившись до рисунків, помітимо, що максимальні результати в розглянутих випадках $N=5,6,7$ одержані тоді, коли:

- початкова інфікована клітина і та, що від неї «народилась» на першій хвилині, «поділять» між собою площину навпіл, щоб не «заважати» одна одній;
- дії цих двох засновників колонії вірусних клітин і їх потомків симетричні одна одній відносно середини відрізка, що сполучає їх центри.

У півплощині інфікування у найкращому (для вірусу) варіанті відбувається за такими правилами:

- у колонії кожної хвилини «виживає молодняк», тобто всі клітини, народжені минулої хвилини, мають можливість вразити одну здорову клітину, потім «надається право» вибрати здорову клітину тим вірусним клітинам, які старші за них, тобто народилися $2, 3, \ldots, n$ хвилин тому;
- на N-й хвилині вражають по одній клітині лише дві з тих вірусних клітин, що народилися на $(N-2)$-й хвилині, лише одна — з народжених на $(N-3)$-й хвилині і жодна з народжених раніше;
- кожна вірусна клітина має «генетичну пам'ять» і намагається вразити здорову клітину в тому напрямку, у якому народилася сама; якщо цього зробити неможливо, вона шукає здорову в напрямку додатного повороту на $60°$, а потім — від'ємного.

Автор спробував написати формулу для обчислення кількості інфікованих клітин для всіх $N>5$, якщо інфікування відбувати-

меться далі саме за такими правилами, які вдалося помітити. Ось як він розмірковував.

Нехай $f(n)$ — кількість інфікованих клітин, які народились у півплощині на n-й хвилині. Всі ці клітини на наступній хвилині вразять по одній здоровій клітині, те саме станеться і з двома народженими на $(n-2)$-й хвилині, і з однією — на $(n-3)$-й. Іншим просто не вистачить здорових клітин для інфікування.

Тобто $f(n+1)=f(n)+3$. Тому різниця $f(n+1)-f(n)$ для $n>5$ завжди однакова і дорівнює 3. А це означає, що $f(n)$ виражається через n лінійно, тобто існує формула $f(n)=P*n+Q$, де P і Q — деякі константи, які легко знайти. По-перше, $P=3$, що ж до значення Q, то його легко знайти, знаючи $f(6)=11$ (*див. рис.*): $Q=-7$.

Тепер вже нескладно, знаючи загальну кількість вражених клітин після 5 хвилин (їх 16), написати формулу для чисельності колонії $S(n)=S(5)+f(6)+f(7)+\ldots+f(n)=16+(3*6-7)+$ $+(3*7-7)+\ldots+(3*n-7)$ — всього $n-5$ доданків у дужках.

Або $S(n)=\dfrac{(3n^2-11n+12)}{2}$ для півплощини, а для всієї площини — $S(n)=3n^2-11n+12$.

Коли автор увімкнув комп'ютер і спробував запустити програму для обчислення вірусних клітин для $n=8$, то це тривало ще довше, але максимум 116-й був знайдений, і він збігався з результатом за вказаною нами формулою. Ось чому автор не наводить програми, яка відшукує ці результати. Їх, як виявилося, можна знайти, розмірковуючи над трьома рисунками, наведеними в тексті.

Зменшення у порівнянні з формулою 2^N результату обумовлене «тіснотою» серед інфікованих клітин, тому всі інфіковані клітини намагаються «вирватись» якомога далі від джерела інфекції. А оскільки вони мають змогу це зробити лише в півплощині, то чітко визначаються:

- перший головний напрямок, з яким можна «втекти» якомога далі (див. останній рисунок);
- другий головний напрямок, з яким найдальша клітина на один крок ближча;
- два напрямки на горизонталі (лівий та правий) з відставанням ще на один-два кроки.

У першому варіанті клітина проживе 3 хв, в другому — 2 хв, в решті — лише одну. Звідси випливає той алгоритм, який ми склали (виживає увесь «молодняк» і ще 1 + 2 = 3 клітини зі старших за віком).

Цікаво, що збереження «молодняка» — це риса, притаманна майже всьому живому, в тому числі, можливо, і справжньому, а не придуманому нами для задачі вірусу. «Генетична пам'ять» у її дещо іншому варіанті — теж ознака живого. Насправді все простіше: триматися подалі від початкової клітини — ось надія на існування здорових клітин, якими можна «поживитись». Хтозна, може, саме так — через голод і боротьбу за виживання — формується справжня генетична пам'ять...

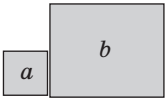
Зрозуміло, що ці сентенції в жодному разі не можуть замінити доведення того, що максимум колонії при значеннях $N > 3$ збігається з формулою $S(N) = 3N^2 - 11N + 12$, а для $N = 0 \ldots 3$ — з іншою 2^N. Тому автор, впевнений у своїх результатах, запрошує читачів довести (або спростувати) справедливість пропонованих формул.

Нижче наведено надзвичайно просту програму, яка виводить на екран кількість інфікованих клітин за вказаною формулою.

```
Uses crt;
var n,m,k:LongInt;
begin
  Clrscr;
  write('Кількість хвилин ');
  read(n);
  m:=n*(3*n-11)+12;
  шf n<4 then begin m:=1;
                for k:=1 to n do m:=m*2;
              end;
writeln(m);
readln;readln
end.
```

ПРЯМОКУТНИКИ З РІЗНИХ КВАДРАТІВ

(О-2001)

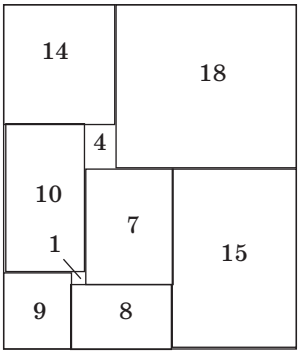


Деякі прямокутники можна розрізати на квадрати, сторони яких — різні натуральні числа. Знайдіть такі прямокутники, складені з N квадратів ($N \le 15$), серед яких є два сусідніх зі сторонами a і b ($a, b < 50$), що мають спільну вершину, а сторона меншого — частина сторони більшого (*див. рисунок*).

Приклади вхідного та вихідного файлів

rect1.dat	1 7 9	Значення a , b , N
rect1.sol	32 33 1 4 7 8 9 10 14 15 18	Два числа — розміри прямокутника Один рядок — сторони N квадратів

Розв'язання

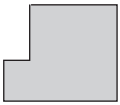


Ця задача виявилася несподіваною не тільки для учасників олімпіади, а й для тих членів журі, які ознайомилися з нею одночасно з учнями. Крім того, спантеличені текстом, вони виказали своє невдоволення автору, який буцімто припустився помилки (мовляв, неможливо скласти прямокутник з квадратів різного розміру). Вони просто не знали, що це досить відома ще з XVII століття задача, яка виникла неабиякі суперечки навіть серед відомих математиків і фактично була розв'язана лише у XX столітті.

Так, відомий радянський математик академік М. М. Лузін вважав, що вона не має розв'язків, а в книзі польського математика Гуго Штейнгауза, що вийшла у Львові 1930 р., було надруковано: «Невідомо, чи можна розбити прямокутник на попарно різні квадрати». І лише 1947 р. виявилося, що не тільки прямокутник, а й квадрат можна скласти з таких квадратів. Тоді ж вдалося з'ясувати, що задача має красиву графічну інтерпретацію, що поріднює її з відомими в електротехніці двома законами Кірхгофа, які фактично відкривають шлях до аналітичного обчислення сторін квадратів (див. далі). Автор ознайомився із цією задачею за студентських років, коли лише починалась ера комп'ютерів, тому розв'язання, про яке він прочитав, набуло лише аналітичного характеру. Оскільки задача сподобалась авторові, 1997 року йому вдалося скласти дві різні програми, які б обчислювали сторони таких квадратів, з яких можна скласти прямокутник. Задачу нарешті було запропоновано на Житомирській обласній олімпіаді.

Насправді, задача має розв'язки, але при $N > 8$. Так, наприклад, з дев'яти квадратів зі сторонами 1, 4, 7, 8, 9, 10, 14, 15, 18 можна скласти прямокутник 32×33 (див. рисунок).

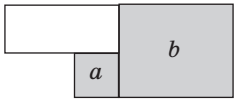
А тепер — про те, як розв'язати цю цікаву задачу.

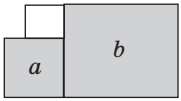


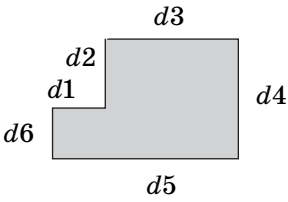
Розглянемо «сходинку» (див. рисунок), складену з двох неоднакових прямокутників.

Спочатку це два нерівних квадрати зі сторонами a і b, ($a<b$).

Добудуємо «сходинку» новим квадратом зі стороною $b-a$. Оскільки всі квадрати різні, можливі два випадки ($2a<b$ і $2a>b$):





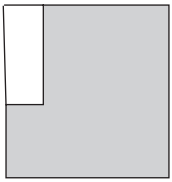


Взагалі кожну «сходинку» можна задати чотирма (з шести) параметрами $d1, d2, \ldots, d6$. Так, перша сходинка матиме такі значення параметрів:

$d1$	$d2$	$d3$	$d4$	$d5$	$d6$
a	$b-a$	b	b	$a+b$	a

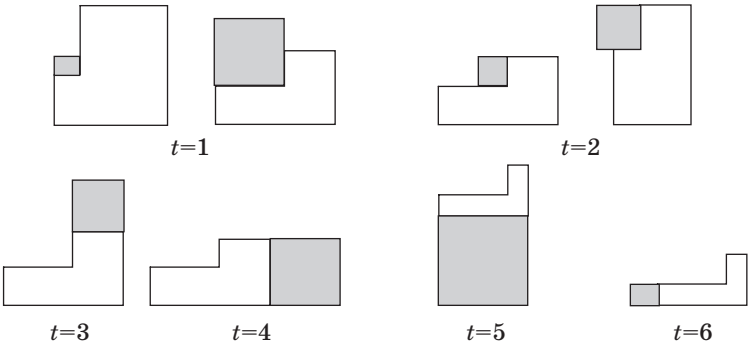
Сходинку можна добудувати і в інший спосіб: для цього вибрати одне з чисел $k=1\ldots6$ (за кількістю параметрів d) і побудувати зовні на стороні d_k квадрат зі стороною d_k. При цьому, якщо $k\neq1,2$, то кожного разу ми отримаємо знову сходинку, але з іншими параметрами $d_1\ldots d_6$. Те саме відбудеться при $k=1,2$ за умови, що d_1 і d_2 різні. Якщо ж вони однакові, то сходинка виродиться у прямокутник. Процедура добудови сходинки названа в програмі plus(k:integer).

Автор розраховував на те, що учні помітять це й інтуїція підкаже їм, як колись підказала авторові задачі, що, продовжуючи далі, ми врешті-решт «замкнемо» сходинку до прямокутника N-им квадратом, якого ще немає серед побудованих $N-1$ квадратів, і таким чином розв'яжемо задачу.



Параметри й орієнтація кожної нової «сходинки» суттєво залежатимуть від того, якими будуть для попередньої «сходинки» значення $d1$ і $d2$ ($d1<d2$, $d1>d2$ чи $d1=d2$, як на останньому кроці). Нагадаємо, що ми будемо добудовувати квадрати не лише зі сторонами $d1$ і $d2$, а й $d3$, $d4$, $d5$, $d6$, адже кожна така добудова пере-

творює сходинку на нову фігуру (теж сходинку — на перших $N-3$ кроках).



Тож розв'язування задачі зводиться до перебирання можливих варіантів розмірів $d1 \ldots d6$, таких, що після $N-3$ добудов ми отримаємо «сходинку», у якій $d1 = d2$, тобто останній квадрат зі стороною $d1$ «замкне» її до прямокутника. Перебирання легко здійснити для невеликих значень N за допомогою відомого бектрекінгу («метод віток і меж»).

Написання програми, що розв'язує задачу мовою Паскаль, наводиться нижче (в програмі параметри d — це числа двовимірного масиву $d[i,k]$ (i — номер сходинки $1 \ldots N$, $k = 1 \ldots 6$ — номер відповідного параметра 1... 6), а $c[1 \ldots n]$ — сторони квадратів, з яких складено прямокутник (перші дві з них задані в умові $c[1] = a$ і $c[2] = b$).

Відзначимо, що для будь-яких a, b і N може виявитися, що такого розрізання прямокутника на N квадратів не існує, тому тести в задачі дібрані були з уже відомих заготовок, щоб упевнено мати розв'язок. Для тих, хто захоче перевірити, як працює програма, наводимо ще кілька із цих заготовок (це можна зробити і в інший спосіб — задати будь-які a і b, а потім, перебираючи значення $N \geq 9$, вибрати таке, при якому програма видає відповідь).

N	a	b	**Сторони прямокутника**	**Сторони квадратів**
10	2	11	55 x 57	2 11 13 15 17 8 25 3 27 30
10	5	6	47 x 65	5 6 11 17 23 24 19 3 22 25
N	a	b	**Сторони прямокутника**	**Сторони квадратів**
10	3	35	79 x 130	3 35 38 41 44 12 23 11 34 45
10	4	11	98 x 111	4 11 15 26 41 7 44 3 54 57
10	4	15	115 x 94	4 15 19 34 23 11 60 16 39 55

N	a	b	Сторони прямокутника	Сторони квадратів
10	6	11	5 4 x 7 5	6 11 17 23 24 19 3 22 25
10	7	12	105 x 104	7 12 19 26 45 33 28 16 44 60
9	7	15	32 x 33	7 15 8 1 9 10 4 14 18
9	7	16	61 x 69	7 16 9 25 2 36 5 28 33
10	8	15	32 x 65	8 15 7 1 9 32 10 4 14 18
10	9	16	61 x 130	9 16 25 7 2 36 61 5 28 33

Для тих, хто пам'ятає з фізики розділ електрики (а точніше — закон Ома для повної мережі, або два закони Кірхгофа), можна порадити самому спробувати здійснити подібні розрізання прямокутника на різні квадрати.

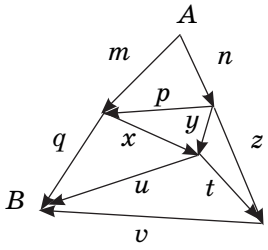


Рис. А

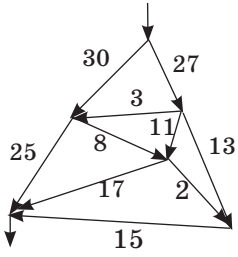


Рис. В

Для цього на проміжку електричної мережі від A до B (див. *рис. А*) накреслимо 10 відрізків (опорів) так, щоб із кожної вершини (крім A і B) виходило не менше від трьох відрізків, і розставимо на рисунку будь-яким чином стрілки — напрямки руху електричного струму на опорах (малими буквами позначені величини сил цих струмів, поки що нам не відомі).

Зрозуміло, що з *рис. А* легко скласти п'ять рівнянь $n+p=m$; $p+x=y$; $y+t=z$; $t+v=u$; $x+u=q$ (адже в кожному замкненому контурі без батарей сума сил струмів дорівнює нулю). Додамо ще чотири рівняння для вершин, де сходяться більше ніж два ребра: $m+p=q+x$; $n=p+y+z$; $x+y=u+t$; $t+z=v$ (у кожному вузлі сума сил вхідних струмів дорівнює сумі сил вихідних струмів). При цьому $q+x+u+v$ — сума сил струмів у вузлі A повинна дорівнювати сумі сил струмів у вузлі B, тобто $m+n$. Із системи цих дев'яти рівнянь із точністю до постійного множника можна отримати значення всіх опорів — найменші за модулем цілі числа, що і є значеннями сил струмів. (Якщо деякі з них виявляться від'ємними, на

рисунку слід поміняти стрілки на цих відрізках на протилежні та змінити струми на додатні.) Якщо виявиться, що всі струми різні, то це й буде потрібне розрізання прямокутника на 10 (за кількістю відрізків на схемі) різних квадратів. На *рис. В* зображені значення сил опорів (сторони розрізання), які отримаємо, розв'язавши систему дев'яти рівнянь для *рис. А*. Із зазначеного способу випливає інший алгоритм — через розв'язування систем рівнянь, який авторові вдалося втілити у значно складнішу програму 1997 року (програма не наводиться).

```
Clrscr;
write('a,b,N=');read(a,b,N);
q:=2;d[2,1]:=a;d[2,2]:=b-a;d[2,3]:=b;
d[2,4]:=b;d[2,5]:=b+a;d[2,6]:=a;k:=1;
s:=[a,b];c[1]:=a;c[2]:=b;
while q>1 do
begin
  if q<n-1
  then begin
        if k<=6
        then begin
              if not(d[q,k] in S)
              then begin
                    plus(k);inc(q);
                    p[q]:=k;c[q]:=d[q-1,k];
                    k:=0
                   end
             end
          else begin
                k:=p[q];s:=s-[c[q]];dec(q)
        end
    else if not(d[n-1,k] in s) and
           (d[n-1],1]=d[n-1,2])
        then begin
              plus(k);
              writeln(d[n,3],' ',d[n,6]);
              for j:=1 to n do write(c[j],' ');
              readkey;halt
             end
          else begin
                k:=p[q];s:=s-[c[q]];dec(q);
              end
    inc(k);
  end;
```

```
  {Розміри прямокутника - d[N,3] та d[N,6]}
  readkey
end.
```

Нагадаємо, що plus(t) — досить тривала процедура добудови нового квадрата — «сходинки» — за формулами, що визначають параметри d нової сходинки, а t — номер параметра d (з шести можливих 1... 6), до якого добудовується новий квадрат зі стороною $d[q,t]$. На поданих вище рисунках показано, який вигляд матиме нова сходинка в разі добудови $(q+1)$-го квадрата $(q=2...N-2)$ для різних значень $t=1...6$, а при $t=1,2$ — ще й у випадках $d[q,1]<d[q,2]$ та $d[q,1]>d[q,2]$.

```
Procedure plus(t:Integer);
var j:Byte;
begin
  for j:=1 to 6 do d[q+1,j]:=d[q,j];
  case t of
  1:begin
     if d[q,1]<d[q,2]
     then begin
           d[q+1,2]:=d[q,2]-d[q,1];
           d[q+1,6]:=d[q,6]+d[q,1];
         end else
     if d[q,1]>d[q,2]
     then begin
           d[q+1,6]:=d[q,1];
           d[q+1,1]:=d[q,1]-d[q,2];
           d[q+1,2]:=d[q,3];
           d[q+1,3]:=d[q,4];
           d[q+1,4]:=d[q,5];
           d[q+1,5]:=d[q,1]+d[q,6]
         end
    end;
2:begin
    if d[q,2]>d[q,1]
    then begin
          d[q+1,1]:=d[q,2]-d[q,1];
          d[q+1,2]:=d[q,6];d[q+1,3]:=d[q,5];
          d[q+1,5]:=d[q,3]+d[q,2];
          d[q+1,6]:=d[q,2]
        end else
    if d[q,2]<d[q,1]
    then begin
```

```
            d[q+1,1]:=d[q,1]-d[q,2];
            d[q+1,3]:=d[q,3]+d[q,2]
          end;
    end;
  3:begin
     d[q+1,2]:=d[q,3]+d[q,2];
     d[q+1,4]:=d[q,4]+d[q,3]
    end;
  4:begin
     d[q+1,3]:=d[q,3]+d[q,4];
     d[q+1,5]:=d[q,4]+d[q,5]
    end;
  5:begin
     d[q+1,6]:=d[q,5]+d[q,6];
     d[q+1,4]:=d[q,4]+d[q,5];
    end;
  6:begin
     d[q+1,5]:=d[q,5]+d[q,6];
     d[q+1,1]:=d[q,1]+d[q,6];
    end
    end;
  end.
```

Програма для цієї задачі, як і всі перебірні, працює не дуже швидко, але для значень $N = 9...12$ вже за декілька секунд дає потрібний результат.

Відзначимо, що добудова «сходинок» — це не універсальний спосіб отримання всіх можливих розрізань прямокутників на різні квадрати. Існують такі розрізання, які неможливо побудувати добудовою «сходинок». Докладніше про це можна прочитати у надзвичайно цікавій брошурі талановитого автора І. М. Яглома «Як розрізати квадрат» (М. : Наука, 1968).

Пригадую Всеукраїнську олімпіаду 1991 року у Хмельницькому, коли В. Бардадим, який вмів складати для олімпіад дуже цікаві задачі, запропонував учасникам задачу про електричні опори, для якої не мав не те що загального розв'язку, а навіть будь-якого прогнозу щодо можливого алгоритму, зате мав тести для деяких значень вхідних даних, відповіді на які були майже очевидними, якщо... спрацює інтуїція учасників олімпіади. Саме на це розраховував і автор цієї задачі, коли запропонував її на олімпіаді Житомирської області 2001 року.

Нижче наведено програму, складену за ідеями, близькими до міркувань, викладених вище.

```
Uses crt;
var  d:array[1..6,0..15] of Integer;
     p,c,h:array[1..15] of Integer;
     n,i,k,q,a,b,pr,z,j:Integer;
     x,y:array[1..50] of Integer;
     f,fnew:boolean;
{-----------------------------------------------}
procedure print;
var j,u,v:Integer;
begin u:=d[2,q];v:=d[3,q];j:=1;
  while(j<=pr) and (not((u=x[j]) and (v=y[j])))
       and (not((v=x[j]) and (u=y[j]))) do inc(j);
  if j=pr+1
  then begin inc(pr);x[pr]:=u;y[pr]:=v;
         writeln('Прямокутник ',u,'x',v);
         writeln(' Квадрати:');
         c[n]:=d[5,q];
         for j:=1 to n do
         write(c[j],' ');writeln;
       end;
end;
{-----------------------------------------------}
function new(u,v:Integer):boolean;
var j:Integer;
begin   j:=1;while(j<=v) and (c[j]<>u) do inc(j);
        fnew:=(j=v+1);new:=fnew;
end;
{-----------------------------------------------}
procedure krok(t:Integer);
var j,st:Integer;
begin for j:=1 to 6 do d[j,q]:=d[j,q-1];
  f:=true;st:=d[t,q-1];
  case t of
    1:begin inc(d[2,q],st);inc(d[6,q],st) end;
    2:begin inc(d[1,q],st);inc(d[3,q],st) end;
    3:begin inc(d[2,q],st);inc(d[4,q],st) end;
    4:begin inc(d[3,q],st);inc(d[5,q],st) end;
    5:if st<d[6,q-1]
      then begin inc(d[4,q],st);dec(d[6,q],st) end
      else
      if st>d[6,q-1]
      then begin
             d[1,q]:=st;
             d[2,q]:=d[4,q-1]+st;d[4,q]:=d[2,q-1];
```

```
            d[5,q]:=d[1,q-1];d[6,q]:=st-d[6,q-1]
          end else f:=false;
    6:if d[5,q-1]>st
      then begin
            inc(d[1,q],st);dec(d[5,q],st)
          end else
      if d[5,q-1]<st
      then begin
            d[1,q]:=d[3,q-1];
            d[3,q]:=d[1,q-1]+st;d[4,q]:=st;
            d[6,q]:=d[4,q-1];d[5,q]:=st-d[5,q-1]
          end else f:=false;
  end;
  if f then c[q+2]:=d[t,q-1];
end;
{-------------------------------------------}
procedure step;
begin inc(q);p[q]:=k;krok(k);
  if f=false then dec(q) else k:=0;
end;
{-------------------------------------------}
procedure back;
begin k:=p[q];dec(q) end;
{-------------------------------------------}
begin
  clrscr;write('N,a,b=');read(n,a,b);
  c[1]:=a;c[2]:=b;pr:=0;
  d[1,0]:=a;d[2,0]:=a+b;d[3,0]:=b;
  d[4,0]:=b;d[5,0]:=b-a;d[6,0]:=a;
  h[1]:=2;h[2]:=5;
       for i:=1 to 6 do
       begin if new(d[i,0],2)
             then begin
                  f:=true;p[1]:=i;
                  q:=1;k:=1;krok(i);
                  if f
                  then begin
                        while (q>0) do
                        begin
                          if q<n-3
                          then begin
                                if k<=6
                                then
                  begin
                    if new(d[k,q],q+2)
                    then step
```

```
                    end
                    else back;
                         end
                  else begin
                  if (q=n-3) and new(d[5,q],q+2)
                             and (d[5,q]=d[6,q])
                  then print;
                  back;
                 end;
              inc(k)
            end;
            end;
         end;
      end;
      writeln('Інших немає. кінець');
      readln;readln;
end.
```

АЕРОПОРТ

(О-2002)

Злітно-посадкова смуга аеропорту завдовжки L метрів має обладнання, яке фіксує з точністю до 0,1 м зліт і посадку лайнерів від старту до відриву від землі або від точки приземлення до точки зупинки. Оскільки лайнери злітають і сідають у різних місцях смуги, зношеність її покриття неоднакова в різних точках. Наприкінці тижня ділянки, де побували не менше ніж M лайнерів, потрібно ремонтувати. (N, $L < 3000$).

Маючи тижневі дані руху N лайнерів, обчисліть:

а) кількість P літаків, що побували на найбільш зношених ділянках;

б) загальну довжину R ділянок, які потребують ремонту;

в) кількість Q ділянок, де не було жодного з лайнерів.

Приклади вхідного та вихідного файлів

input.txt	4 2 443 5.5 27.6 7.7 16.9 6.6 121 415 222	Значення N і M Значення L N рядків з координатами точок приземлення та зупинки або старту і зльоту

output.txt	3 184.0 3	Значення P Значення R Значення Q

Розв'язання

Побудуємо всі N точок приземлення, зупинки, старту та зльоту лайнерів на злітно-посадковій смузі, тобто на відрізку $[0;L]$ осі Ox, і зберемо їх в один масив c, у якому не більше ніж $2N$ елементів. Якщо k-й лайнер сідав або злітав на відрізку $[a[k],b[k]]$, а функція new(u) зі значеннями true/false визначає справедливість твердження «точки u ще нема в масиві c», то наведений нижче фрагмент визначить положення $c[i]$ на смузі кожної з таких q ($q \le 2n$) точок.

```
m:=0;q:=2;c[1]:=a[1];c[2]:=b[1];
for k:=2 to n do
begin
  if new(a[k]) then begin inc(q);c[q]:=a[k] end;
  if new(b[k]) then begin inc(q);c[q]:=b[k] end;
end.
```

Відсортуємо масив c за зростанням:

```
for k:=1 to q-1 do
begin
  Min:=c[k];
  for i:=k+1 to q do
  if c[i]<min then begin min:=c[i];nom:=i end;
  x:=c[nom];c[nom]:=c[k];c[k]:=x
end.
```

Ми встановили, що на смузі всього q позначок, тобто маємо $q-1$ ділянок, на яких сідали та злітали N літаків.

Визначимо, скільки побувало літаків на кожній із цих ділянок, для цього порахуємо, скільком з N відрізків $[a[k],b[k]]$ належить середина $x=(c[i]+c[i+1])/2 (i=1\ldots q-1)$ кожної з $q-1$ ділянок.

```
for i:=1 to q-1 do
begin
  x:=(c[i]+c[i+1])/2;
  Lit[i]:=0;
  for k:=1 to N do
  if (x>a[k]) and (x<b[k]) then inc(lit{i});
end.
```

Читач вже зрозумів, що найбільш зношені ділянки — ті, що мають номери z, такі, що значення $\mathrm{Lit}(z)=\max$ (значення найбільшого елемента масиву $\mathrm{Lit}[1\ldots q-1]$), звідки вже легко порахувати

кількість літаків, що побували на цих ділянках, тобто кількість максимумів у масиві $\mathrm{Lit}[1\ldots q-1]$).

Потребують ремонту лише ті ділянки, де $\mathrm{Lit}(z)\ge M$. Тож загальну довжину цих ділянок теж визначити нескладно.

Дещо уважніше потрібно рахувати кількість ділянок, на яких не побувало жодного літака. По-перше, при $c[1]>0$ вже матимемо проміжок $[0,c[1])$ без літаків. А при $c[q]<L$ є ще один проміжок $(c[q],L]$ — теж без літаків. Далі можна піти таким шляхом: перевіряючи проміжки $(c[k],c[k+1])$ для $k=1\ldots q-1$, запишемо в новий масив $\mathrm{zero}[1\ldots z]$ номери k тих проміжків, для яких $\mathrm{Lit}[k]=0$. Якщо виявиться, що, наприклад, $\mathrm{zero}[5]=\mathrm{zero}[6]$ (на двох сусідніх проміжках не було літаків), то це означатиме, що ми маємо фактично один проміжок з номером 5 без літаків тощо. Сподіваємось, що читач вже зрозумів, у чому тут справа, і зможе скласти відповідну процедуру, що рахує кількість таких проміжків. Ось таким чином і розв'язується задача.

АТЕСТАЦІЯ

(О-2002)

Щоб атестувати учнів з теми «Всі дії з десятковими дробами», вчитель запропонував їм приклади різного рівня складності такого вигляду: $w=?$ (тут w — арифметичний вираз без дужок із цілими числами та десятковими дробами (з комами) і знаками операцій + – x : (додавання, віднімання, множення, ділення).

Знайдіть точну правильну відповідь у прикладі, записаному в текстовому файлі. Зразки файлів подаються. Всі приклади коректні, результати ділення — скінченні дроби.

Приклади вхідного та вихідного файлів

input.txt	24 : 15 x 6 + 13 – 2,9 x 2 = ?	Приклад
output.txt	16.8	Точна відповідь

Розв'язання

Вираз у лівій частині прикладу є сума доданків вигляду $M\,\&\,N\,\&\ldots\&\,P$, де & — знак множення (x) або ділення (:), а M, N, … P — деякі операнди (числа). Кожен доданок нескладно виділити з тексту прикладу і потім обчислити. Але для цього спочатку десяткову кому слід замінити точкою.

Якщо перший символ виразу без знака «+» чи «−», то доданків буде на один більше, ніж знаків «−» і «+».

Нехай в доданку d нараховується dz знаків множення (x) і ділення (:), а ці знаки записано в літерний масив $\mathrm{zn}[1\ldots\mathrm{dz}]$. Якщо записати в літерний масив $\mathrm{op}[1\ldots\mathrm{dz}+1]$ усі операнди (літерні величини), то нескдано визначити алгоритм обчислення одного доданка.

Обчислимо значення кожного з операндів і запишемо їх у числовий масив $\mathrm{zop}[1\ldots\mathrm{dz}+1]$, адже кожна з мов програмування високого рівня передбачає можливості перетворити літерний запис числа на число.

Тоді значення одного доданка нескладно обчислити.

```
 w:=zop[1]
для k від 1 до dz
пц
  якщо zn[k]="x" то w:=w*zop[k+1] все
  якщо zn[k]=":" то w:=w/zop[k+1] все
кц
```

Тепер можна обчислити з урахуванням знаків «+» та «−» суму всіх доданків, яка і буде відповіддю прикладу.

Є інший спосіб обчислення значення w — виразу без дужок. Це посимвольне читання виразу. Він дещо складніший, але й цікавіший. Викладемо цей спосіб.

Покладемо su:=0 (суматор усіх доданків), b:= «» (літерний буфер для зберігання цифр, десяткових крапок і знаків «+» або «−»), zb:=0 (значення чергового буфера, де zb:=знач(b)), mu:= «» (відкладена дія ділення або множення «:» або «x»), d:=0 (значення чергового доданка), p:=довж(w).

Уявімо собі, що нам показують лише один символ $s = w[k:k]$ з виразу, про який, крім того, відомо, чи не є цей символ останнім. Спробуємо визначити, що робити із цим символом s.

Можливі випадки:

- s — останній символ виразу (цифра);
- s — цифра «0»... «9», або крапка «.»;
- s — знак «+» або «−»;
- s — знак множення «x»;
- s — знак ділення «:».

У першому випадку:

```
якщо k=p
то b:=b+s;    (символ -- у буфер)
  zb:=знач(b) (значення буфера)
  ДОДАНОК  (виконати відкладені дії,
```

```
    su:=su+d   обчислити d і послати в su)
  все
```

У другому і третьому випадку:

```
  якщо s є в ".0123456789-+"
  то b:=b+s; zb:=знач(b)
    якщо s="+" або s="-"
    то доданок  (обчислення d)
      su:=su+d   (d — в суматор)
      b:="";zb:=знач(b);  (стартові значення
      mu:=""; d:=0   для нового d)
    все
  все
```

У третьому і четвертому випадках:

```
  якщо s="x" або s=":"
  то доданок (виконати відкладені дії, знайти d)
  a:=s
  все
```

Допоміжний алгоритм доданок обчислює d — числове значення прочитаної частини доданка.

```
  якщо mu="x" то d:=d*zb все
  якщо mu=":" то d:=d/zb все
  b:=""; zb:=0
  якщо s="x" або s=":" то mu:=s все
```

Легко скласти програму, що працює за цим алгоритмом.

Далі додаються:

1) програма обчислення одного виразу;
2) програма, що читає дані з файла й обчислює значення виразів.

{Додаток 2а: Обчислення одного виразу: вираз вводиться з клавіатури}

```
uses crt;
var w,s,ad,mu,b:string;
    p,n,k,j:Integer;
    d,su,zb:real;
    cc:Char;
{-----------Введення виразу---------------------}
procedure data;
var ww:string;
begin clrscr;write('Вираз: ');readln(ww);w:='';
  for j:=1 to length(ww) do
  begin if ww[j]=',' then ww[j]:='.';
    if pos(ww[j],'+-x:.0123456789')>0 then w:=w+ww[j];
  end;
```

```
  if pos(w[1],'+-')=0 then w:='+'+w;p:=length(w)
end;
{-----------Прочитана частина доданка-----------}
procedure dod;
begin if mu='' then d:=zb;
      if mu='x' then d:=d*zb;
      if mu=':' then d:=d/zb;
      b:='';if pos(s,'x:')>0 then mu:=s;
end;
{------------Значення суматора-----------------}
procedure sum;
begin   if ad='+' then su:=su+d;
        if ad='-' then su:=su-d;
        b:='';ad:=s;mu:='';
end;
{---------------Основна програма----------------}
begin data;b:='';zb:=0;ad:='';mu:='';d:=0;
  for k:=1 to p do
  begin s:=w[k];
    if k=p
    then begin b:=b+s;val(b,zb,j);dod;sum end
    else begin
          if pos(s,'+-')>0
          then begin dod;sum end
          else if pos(s,'x:')>0
               then dod
               else
               begin b:=b+s;val(b,zb,j) end
          end;
    end;
    writeln('Відповідь: ',su:5:5);
    cc:=readkey;
end.
{Додаток 2:Головна програма}
uses crt;
varw,s,ad,mu,b,ww:string;
     p,n,k,j:Integer;
     d,su,zb:real;
     cc:Char;
    f,fs:Text;

{----------------Наступний приклад----------------}
```

```
procedure start;
begin
  write('Вираз: ');readln(f,ww);w:='';
  for j:=1 to length(ww) do
  begin if ww[j]=',' then ww[j]:='.';
    if pos(ww[j],'+-x:.0123456789')>0 then w:=w+ww[j];
  end;
  if pos(w[1],'+-')=0 then w:='+'+w;p:=length(w);
  writeln(w);
end;
{---------------Читання даних------------------}
procedure data;
var nf:string;
begin clrscr;writeln('Назва файла на диску
A:');readln(f,nf);
  assign(f,'a:\'+nf+'.dat');reset(f);readln(f,n);
  assign(fs,'a:\'+nf+'.sol');rewrite(fs);
  writeln('Всього прикладів ',n);
end;
{--------Прочитана частина доданка--------------}
procedure dod;
begin   if mu='' then d:=zb;
        if mu='x' then d:=d*zb;
        if mu=':' then d:=d/zb;
        b:='';if pos(s,'x:')>0 then mu:=s;
end;
{--------------Значення суматора----------------}
procedure sum;
begin   if ad='+' then su:=su+d;
        if ad='-' then su:=su-d;
        b:='';ad:=s;mu:='';
end;
{-------------Основна програма-----------------}
begin data;
   for pr:=1 to n do
   begin start;b:='';zb:=0;ad:='';mu:='';d:=0;
      for k:=1 to p do
      begin s:=w[k];
        if k=p
        then begin b:=b+s;val(b,zb,j);dod;sum end
        else begin
               if pos(s,'+-')>0
```

```
                then begin dod;sum end else
                  if pos(s,'x:')>0
                  then dod else
                  begin b:=b+s;val(b,zb,j) end
              end;
    end;
    write('Відповідь: ');
    write(su:5:5);writeln(fs,su:5:5);
    cc:=readkey;
    close(f);close(fs);
end;
end.
```

ТЕТРАЕДР

(I-2001)

Трикутну піраміду, всі грані якої — рівносторонні трикутники (правильний тетраедр), позначені літерами A, B, C, D, перекотили по площині декілька разів без ковзання. Шлях тетраедра закодовано у вигляді слова S, складеного з літер на гранях, якими тетраедр торкався площини.

Скільки разів і на яких кроках тетраедр побував у тих самих місцях, де бував раніше?

Розв'язання

Виберемо на площині косокутну систему координат із кутом між віссю X та Y у $60°$ (*рис. 1*). Кожному ромбу зі стороною 1 поставимо у відповідність пару цілих чисел (u,v) — координати лівого нижнього кута ромба. Якщо провести менші діагоналі в ромбах, площина вкриється одиничними правильними трикутниками, кожному з яких поставимо у відповідність трійку чисел (u,v,w): перші два — координати відповідного ромбу, а w — одне з чисел 0 або 1; $w=1$, якщо трикутник обернений вершиною вниз, і $w=0$ — якщо вверх.

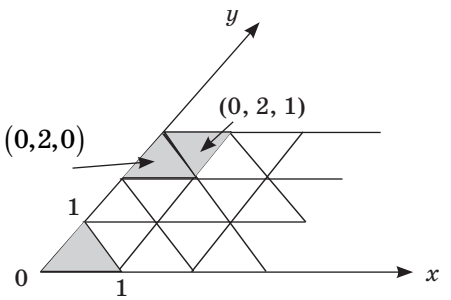


Рис. 1

Нехай трикутник з $u=0,\ v=0,\ w=0$ —перше місце, з якого тетраедр почав котитися по площині. Зрозуміло, що на кожному з подальших кроків основа тетраедра збігатиметься з одним із трикутників, якими вкрито площину.

Дві сторони кожного з цих трикутників паралельні осям X і Y, а третя — прямій Z, паралельній діагоналям ромбів. Кожне наступне положення основи тетраедра під час його перекочування по площині можна отримати симетрією попереднього відносно однієї зі сторін. Позначимо ці симетрії відповідно CX, CY, CZ.

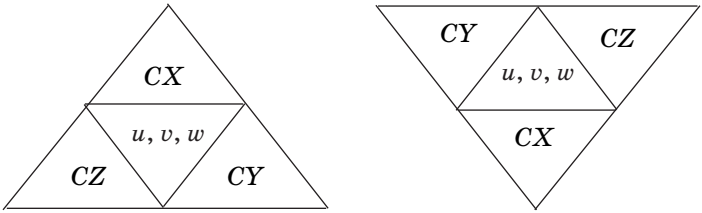


Рис. 2

Легко помітити, що трикутник (u,v,w):

- за симетрії CZ переходить у трикутник $(u,v,1-w)$;
- за симетрії CX — у трикутник $(u-1,v,1)$, якщо $w=0$, або у трикутник $(u+1,v,0)$, якщо $w=1$;
- за симетрії CY — у трикутник $(u,v-1,1)$, якщо $w=0$, або у трикутник $(u,v+1,0)$, якщо $w=1$.

У решті випадків можна записати більш вдалі формули:

- за симетрії CX — у трикутник $(u+2w-1,v,1-w)$;
- за симетрії CY — у трикутник $(u,v+2w-1,1-w)$.

Нехай літери A, B, C, D на гранях тетраедра позначені так, як на *рис. 3б*, де зображено вигляд зверху тетраедра, який торкається площини гранню D (читач розуміє, що порядок літер A, B, C чи A, C, B не має суттєвого значення). Легко помітити, що в разі перекочування через сторону, паралельну X, тетраедр торкнеться площини гранню C (симетрія CX), перекочування через сторону, паралельну Y (симетрія CY) — гранню A, а через сторону, паралельну Z (симетрія CZ), — гранню D.

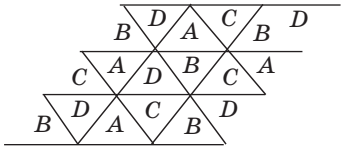


Рис. 3а

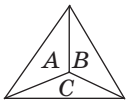


Рис. 3б

Розглядаючи інші випадки, можна скласти таблицю, у якій для кожної з літер на нижній грані вказана відповідна літера на нижній грані в новому положенні тетраедра (*рис.3а*).

Літера на нижній грані	**Літера на нижній грані наступного положення тетраедра**		
попереднього положення	**за симетрії** CX	**за симетрії** CY	**за симетрії** CZ
A	B	D	C
B	A	C	D
C	D	B	A
D	C	A	B

Тепер вже нескладно, використовуючи проведені дослідження, скласти програму, яка створює масиви значень u, v, w, починаючи з $u[0]=0$ $v[0]=0$ $w[0]=0$, аналізує літери, які належать до складу S, де закодований шлях тетраедра по площині і відповідно до вказаної таблиці визначає, якій із симетрій — CX, CY чи CZ — відповідає кожен з кроків тетраедра по площині. Для кожного нового положення програма визначає, чи не зустрічалося воно раніше.

Нижче наведено цю програму мовою Паскаль, яка ґрунтується на ідеях, близьких до викладених вище.

```
Uses crt;
var u,v,w : array[0..255] of Integer; {масиви коор-
динат}
     s : string;
     i,pw : Integer;{pw — кількість повторів}
     spw : set of Byte;{номери кроків, на яких був
повтор}
{----------------Наступний хід------------------}
procedure hid(k : Integer);
var c  : string; r,j : Integer;
begin  c:=s[k]+s[k+1];
       w[k]:=1-w[k-1];u[k]:=u[k-1];
       v[k]:=v[k-1];r:=2*w[k]-1;
```

```
        if (c='AB') or (c='BA') or
           (c='CD') or (c='DC')
        then inc(v[k],r)
        else if (c='CB') or (c='BC') or
                (c='AD') or (c='DA')
        then inc(u[k],r);
        j:=0;
        while (j<=k-1) and ((u[j]<>u[k]) or
              (v[j]<>v[k]) or (w[j]<>w[k])) do
              inc(j);
        if j<k then begin inc(pw);spw:=spw+[k] end;
end;
{----------Основна програма---------------------}
begin clrscr; write('Введіть слово S :');
readln(s);
     u[0]:=0; v[0]:=0; w[0]:=0; pw:=0; spw:=[ ];
     for i:=1 to length(s)-1 do hid(i);
     writeln('Повторів ',pw);
     if pw>0 then writeln('Номери кроків :');
     for i:=2 to length(s)-1 do
     if i in spw then write(i,' ');
     readln ; readln
end.
```

АВАРІЙНИЙ ВИЇЗД

(О-2003)

Місто має декілька кварталів, сполучених M шляхами. В одному з кварталів на станції водопостачання є лише одна машина з радіотелефоном, яка протягом доби здійснює виїзди на місця аварій у порядку надходження P викликів. Щоб зекономити пальне, машина почала виїжджати на новий виклик з пункту попереднього виклику, і лише після останнього виклику повертатися на станцію.

Знаючи довжини всіх шляхів і послідовність викликів, обчислити:

А) на скільки коротшим стане можливий пробіг машини за добу;

Б) якої найбільшої економії пробігу можна досягти, якщо дозволити водію змінювати порядок виїздів.

Усі числа невід'ємні цілі, не більші за 50, кількість викликів — не більша за 10, всі виклики здійсняються з різних кварталів, серед яких немає кварталу станції водопостачання.

Приклади вхідного та вихідного файлів

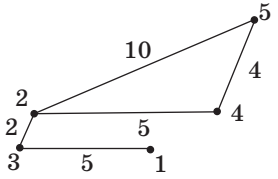
<table>
<tr><th colspan="2">input.txt</th><th colspan="2">output.txt</th></tr>
<tr><td>5 4
3
1 3 5
2 3 2
2 4 5
2 5 10
4 5 4
4 1 5 2</td><td>Шляхів M і викликів P
Номер кварталу станції
M рядків по 3 числа — номери кінців двох сусідніх кварталів і довжина шляху.
Один рядок — P пунктів виклику</td><td>6
18</td><td>Економія пробігу
Максимально можлива економія пробігу</td></tr>
<tr><td colspan="4"> 

Рисунок до файла help1.dat</td></tr>
</table>

Розв'язання

Коли б водій виїздив на виклики в тому порядку, який завжди існує (від станції до квартала виклику, потім назад, на станцію), то загальний пробіг машини дорівнював би подвоєній сумі найкоротших відстаней від станції до кварталів викликів. Нова схема виїздів машини дасть дещо інший пробіг.

Перший виїзд — до першого кварталу зі списку викликів, далі — сума найкоротших відстаней між кварталами (в порядку викликів), і нарешті — останній: повернення з останнього кварталу до станції. За такої схеми від першого до останнього виклику машина на станції відсутня й економія пального обернеться за звичай очікуванням наступного виклику.

Якщо порядок виїздів не був таким, що дає мінімальний пробіг, то, переставляючи у списку викликів номери кварталів, вдасться відшукати такий порядок викликів, який мінімізує довжину пробігу.

Видно, що пошук оптимального пробігу — це звичайна перебірна задача на перебирання всіх перестановок (їх не більше ніж), але перед цим у графі розташування кварталів доведеться визначити найкоротші відстані між кожними двома кварталами, а це легко зробити за допомогою відомого алгоритму Флойда-Уоршела (або алгоритму Дейкстри). Про ці алгоритми можна прочитати у вміщеному в кінці книжки нарисі про графи або переглянути розв'язання задач на графи.

Нижче з невеликими коментарями наведено ядро програми мовою Паскаль, що розв'язує обидві частини задачі. Зауважимо, що перебирання перестановок здійснено на основі класичного алгоритму «метод віток і меж» (бектрекінгу).

```
Procedure data;{читання даних з вхідного файла}
begin
  for i:=1 to 10 do for k:=1 to 10 do d[i,k]:=0;
  {обнулення матриці ребер графа:d[x,y]=0 означає:
  ребра між x і y нема}
  assign(fd,'input.txt');reset(fd);
  readln(fd,m,p); readln(fd,kd);
  N:=0;
  for i:=1 to m do
  begin
    readln(fd,x,y,z);d[x,y]:=z;d[y,x]:=z;
    if x>n then n:=x; if y>n then n:=y
  end; {n — кількість кварталів у місті}
  for i:=1 to p do read(fd,w[i]);
  readln(fd);close(fd);
  close(fd)
end;
Procedure vidst;
begin
  {Перетворення матриці d ребер на матрицю відстаней
  dd для застосування алгоритму Флойда-Уоршела}
  dd:=d;
  for i:=1 to n do for k:=1 to n do
   if (i<>k) and (d[i,k]=0) then dd[i,k]:=maxint;
  for i:=1 to n do
  for j:=1 to n do
  for k:=1 to n do
  if dd[i,j]+dd[j,k]<dd[i,k]
  then begin dd[i,k]:=dd[i,j]+dd[j,k] end;
end;
procedure probig1;{пробіг за звичайної схеми}
begin
  s1:=0;
  for i:=1 to p do s1:=s1+dd[kd,w[i]];
  s1:=2*s1
end;
procedure probig2;{пробіг за нової схеми}
begin
  s2:=0;
  for i:=1 to p-1 do s2:=s2+dd[w[i],w[i+1]];
  s2:=s2+dd[kd,w[1]]+dd[kd,w[p]];
end;
procedure probig3;{оптимальний пробіг}
```

```
begin
  {пошук оптимальної перестановки індексів 1..p}
  min:=maxint;
  {бектрекінг}
  for i:=1 to p do
  begin
     q:=1;u[1]:=i;s:=dd[kd,w[i]];ss:=[i];k:=1;
     while (q>0) do
     begin
       if k<=p
       then begin
              if not(k in ss)
              then begin
                     inc(q);u[q]:=k;ss:=ss+[k];
                       s:=s+dd[w[k],w[u[q-1]]];k:=0
                   end
              end
       else begin
              if (q=p) and (s+dd[kd,w[u[p]]<min)
              then min:=s+dd[kd,w[u[p]]];
              k:=u[q];dec(q);s:=s-dd[w[k],w[u[q]];
              ss:=ss-[k]
            end;
       inc(k);
  end;
end.
```

А основна програма має вигляд:

```
begin
  Data;vidst;
  Probig1;probig2;probig3;
  assign(fs,'output.txt');rewrite(fs);
  writeln(fs,s1-s2);writeln(fs,s1-min);
  close(fs);readkey
end.
```

ПАРКЕТ

(О-2001)

Жук шашель побував на P плитках паркету, викладеного однаковими правильними шестикутниками, і зіпсував їх, у результаті чого утворився «острівець» з M неушкоджених плиток, з усіх сторін оточений зіпсованими. За даним значенням P обчислити найбільше можливе значення M $(1 \le P < 120)$.

Приклади вхідного та вихідного файлів

input.txt	6	Значення P
output.txt	1	Максимальне значення M

Розв'язання

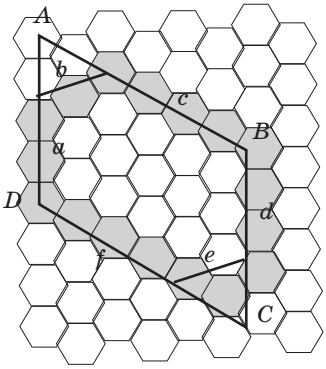
Це досить проста перебірна задача, яку можна розв'язати трьома вкладеними один в одного циклами, а єдиною складністю в ній є шестикутна форма плитки.

Спробуємо рисувати для невеликих значень P відповідні «острівці». Помічаємо, що при $P<6$ значення M дорівнює нулю.

Вважатимемо, що жук пересувається від центра однієї плитки до центра сусідньої, і далі центр плитки називатимемо просто «плиткою».

1^0. Щоб заподіяти найбільшої шкоди, жук повинен економно витрачати кожен з $P-1$ кроків, і тому на жодну з плиток не потрапить двічі. З очевидних міркувань можна зробити висновок, що маршрут жука — опукла замкнена лінія, кожна з ланок якої паралельна одній із трьох прямих, перпендикулярних сторонам плитки, тобто це опуклий многокутник із кількістю сторін не більше шість.

2^0. Вважатимемо його в загальному випадку шестикутником, але «дозволимо» йому мати сторони завдовжки 0.



На рисунку сторони його позначені буквами a, b, c, d, e, f, деякі з цих чисел можуть бути нулями, а за одиницю довжини прийнято відстань між центрами двох сусідніх плиток, яка в $\sqrt{3}$ разів довша за сторону плитки. Зауважимо, що числа a, b, ..., f — це довжини сторін шестикутника, а кількості плиток, розташованих на його сторонах, на одиницю більші.

3^0. Для кожного з таких шестикутників при $P>5$ нескладно побудувати мінімальний паралелограм $ABCD$ з гострим кутом $60°$ (називатимемо його «габаритним», далі — ГП), який покриває маршрут жука (*див. рис.*), тому площу многокутника S легко обчислити, якщо від площі ГП відняти площі двох правильних трикутників (на рисунку вони мають сторони b і e).

4^0. Нам доведеться виражати цю площу S невід'ємним цілим числом — у кількостях плиток. Тому площа кожного з двох трикутників зі сторонами b і e, які доповнюють маршрут жука до ГП, — це сума прогресій вигляду $b+(b-1)+(b-2)+\ldots+1$ та $e+(e-1)+\ldots+1$ (зрозуміло, при $b>0$ і $e>0$). Ці суми дорівнюють відповідно $\frac{b(b+1)}{2}$ і $\frac{e(e+1)}{2}$. Площа ж ГП (у кількостях плиток) — це добуток $(c+b+1)(a+b+1)$.

5^0. Легко помітити, що $a+b=d+e$ $a+f=d+c$ і $b+c=e+f$, тобто сума двох сусідніх сторін дорівнює сумі двох протилежних, паралельних їм.

Звідси, між іншим, випливає, що в кожній парі сусідніх сторін є принаймні одна ненульової довжини. Крім того, оскільки:

$P=a+b+c+d+e+f$, то $S=(c+b+1)(a+b+1)-\frac{b(b+1)}{2}-\frac{e(e+1)}{2}$.

Шукане нами число M отримаємо, віднявши від S кількість плиток на контурі, тобто $M=S-P$. Зрозуміло, що при сталому P найбільшому M відповідатиме найбільше S.

6^0. Легко помітити, що при сталому $P>5$ многокутник визначається лише трьома з шести параметрів $a\ldots f$ (довжини трьох сусідніх сторін) через існування залежностей, заданих поданими вище рівностями, тому пошук оптимального маршруту (з найбільшим S) зводиться до перебирання вкладених циклів. Якщо $x=a+b$ і $y=b+c$ — сторони ГП, тоді $x+y=b+c+d+e$, а тому $a+f=P-x-y$, звідки $a=P-x-y-f$.

Після того як було складено програму та обчислено доволі багато значень M для різних значень P, виявилося, що ця програма зайва. Автор помітив, що найбільша шкода, заподіяна жуком, може бути обчислена значно простіше, фактично — за формулою:

```
k:=p div 6;
if p mod 6=0
then write(' ',3*k*(k+1)+1-p)
else write(' ',(3*k+t)*(k+1)-p)
```

Тому в наведеній нижче програмі з дидактичних міркувань для $P>5$ значення M обчислюється двічі (за програмою, складеною на основі викладених вище міркувань, та за вказаною формулою), хоча, зрозуміло, достатньо було б залишити лише цю чудову формулу, яка робить розв'язання майже мініатюрним.

```
Uses crt;
var max,p,a,b,c,d,e,f,x,y,s,aa,bb,cc,dd,ee,ff,k,t:
    LongInt;
```

```
      ch:Char;
  begin
    clrscr;write('P=');read(p);
    max:=0;
    for x:=1 to p-1 do
    for y:=1 to p-1 do
    begin
      if p>x+y
      then for b:=0 to y do
        begin
          a:=p-x-y-b;c:=y-b;d:=x-c;f:=x-a;
          e:=y-f;
          s:=(y+1)*(x+1)-(c*(c+1)+f*(f+1)) div 2;
            if (a>=0) and (b>=0) and (c>=0) and
               (d>=0) and (e>=0) and (f>=0) and
               (s>max)
            then begin
                   max:=s;aa:=a;bb:=b;cc:=c;
                   dd:=d;ee:=e;ff:=f
                 end
      end;
    end;
    write('M=',max-p);k:=p div 6;t:=p mod 6;
    if t=0
    then write(' ',3*k*(k+1)+1-p)
    else write(' ',(3*k+t)*(k+1)-p);
    readkey;
  end.
```

Далі ми ознайомимо вас із задачею, яка працює з великими числами.

НАЙМЕНШЕ СПІЛЬНЕ КРАТНЕ

(О-2004)

У послідовності натуральних чисел C_1, C_2, C_3 ... будь-яке з чисел C_k — найменше натуральне, яке ділиться без остачі на кожне з перших k натуральних чисел 1, 2, 3, ..., k. Для заданого N вказати найменше M таке, що всі N чисел послідовності, починаючи з C_M, однакові. ($N < 100$)

Приклади вхідного та вихідного файлів

input.txt	2	Значення N
output.txt	5	Значення M

Пояснення

Послідовність C: 1, 2, 6, 12, 60, 60...

Розв'язання

Більшість з тих, хто пробував розв'язувати цю задачу, починали з того, що обчислювали значення C_k — найменшого спільного кратного чисел $1...k$. У результаті навіть при невеликих значеннях k отримували числа, що виходять за межі LongInt, і програма припиняла працювати. Спробуємо не обчислювати ці значення (і не розглядати взагалі масиву C).

Для цього відповімо на запитання: коли C_{k+1} відрізнятиметься від C_k? Нескладно зрозуміти, що це трапиться лише тоді, коли:

- або число $k+1$ є простим;
- або число $k+1$ має вигляд p^m, де p — простий дільник числа C_k, але кількість цих дільників m більша, ніж у розкладі числа C_k.

В обох випадках ця вимога може бути сформульована коротше: кількість простих дільників числа $k+1$ дорівнює 1.

Для цього розглянемо prost(x) — функцію типу boolean натурального x, що визначає справедливість твердження «x — просте число» (true/false).

```
{----------Чи є простим число--------------}
Function prost(x:LongInt):boolean;
var s,i,j:Integer;f:Boolean;
begin
     S:=0;i:=2;f:=true;
     while (i*i<=x) and f do
begin
        if x mod i=0 then f:=false;
        inc(i)
      end;
  prost:=f
end.
```

Щоб бути впевниним, що ми знайшли потрібне нам значення M, складемо функцію yes(t) типу boolean, яка відповідає (так/ні — відповідно true/false) на запитання: чи має число t лише один простий дільник?

```
Function yes(t: LongInt): boolean;
var j: LongInt;f: boolean;
begin
  j:=2; f:=true;
  while (j*j<=t) and f do
  begin
```

```
    while t mod j=0 do
    begin t:=t div j; f:=false end;
    inc(j);
  end;
  if f=false then yes:=(t=1) else yes:=f;
end.
```

Тоді основна програма виглядатиме так:

```
begin
  clrscr;
  write('n='); read(n);
  m:=1; ff:=false;
  repeat
    inc(m);k:=1;
    while yes(m)=false do inc(m);
    while (k<=n-1) and not(yes(m+k)) do inc(k);
    if k=n then ff:=true
  until ff;
  writeln(m);
  readkey;
end.
```

Ось так просто, не обчислюючи значень найменшого спільного кратного числа, ми розв'язали задачу. В наступній задачі нас чекають схожі труднощі. Щоб уникнути обчислень довгих чисел, ми використаємо іншу «фішку».

ДОВГЕ ЧИСЛО

(О-2005)

Послідовні натуральні числа від A до B виписали одне за одним без пропусків, при цьому утворилося число, яке без остачі ділиться на натуральне C. За даними A і C вказати два найменших значення B. (A, $C<10000$, $A \le B$).

Приклади вхідного та вихідного файлів

input.txt	40 3	Значення A і C
output.txt	41 42	Два рядки: найменші значення B

Пояснення

Числа 4 041 і 404 142 діляться на 3.

Розв'язання

Якщо x — многозначне число, то дописуванням до нього деякого натурального y ми отримаємо нове число z=x*10cif+y, де cif — кількість цифр у натуральному y, яку можна обчислити допоміжним алгоритмом cifr.

```
Procedure cifr;
var yy:Integer;
begin
  yy:=y;cif:=0;
  while yy>0 do
  begin yy:=yy div 10;inc(cif) end;
end.
```

Якщо A ділиться без остачі на C, тоді зрозуміло, що перше найменше значення B (позначимо його $B1$) дорівнює A.

Нехай A не кратне C. Зрозуміло, що спочатку $x = A$, а пошук двох найменших значень B ($B1$ і $B2$) можна провести алгоритмом:

```
x:=A; y:=x;
repeat
  inc(y);cifr; x:=x*st(y)+y;r:= ost(x)
until r=0;
B1:=y;
repeat
  inc(y);cifr; x:=x*st(y)+y;r:= ost(x)
until r=0;
B2:=y.
```

Тут величини $\mathrm{st}(y)$ та $\mathrm{ost}(x)$ — значення 10^{cif} та остача від ділення числа x на c відповідно.

Але цей алгоритм у програмі, написаній мовою Паскаль, працюватиме лише за умови, що число x не перевищує межі LongInt. Для вказаних у задачі діапазонів значень B і C алгоритм фактично не годиться.

Щоб розв'язати задачу, нам потрібно навчитись визначати не саме число x, а лише остачу від ділення довгого натурального числа x (навіть 10-значного і більше!) на натуральне c.

Звернімо увагу на формулу x:=x*st(y)+y. Позначимо остачу від ділення x на c через r, а через ry — остачу від ділення на c числа st(y), тоді: r:=r*ry+ost(y); r:=r mod c.

Але величину ry можна обчислити, знаючи значення cif (кількість цифр у числі y) таким чином:

```
Procedure st;
begin
  ry:=1;
  for k:=1 to cif do
  begin
    ry:=(ry mod c)*(10 mod c);
    ry:=ry mod c
  end;
end.
```

Отже, для визначення величин $B1$ і $B2$ ми можемо побудувати інший алгоритм:

```
If A mod c=0 then B1:=A
else begin
      y:=A;r:=A mod c;
      repeat
        inc(y);cifr;st;
        r:=r*ry+y;r:=r mod c
      until r=0;
     B1:=y;
   end;
repeat
  inc(y);cifr;st;
  r:=r*ry+y;r:=r mod c
  until r=0;
B2:=y.
```

Фактично готова програма, що розв'язує задачу.

PAINT

(О-2004)

Фігури на екрані графічного редактора Paint можна виділити, скопіювати (повністю), вставити, перетягти в інше місце.

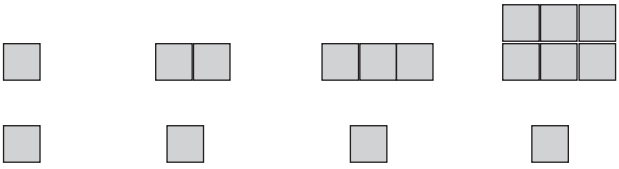
Яку найменшу кількість цих операцій потрібно виконати, щоб, маючи на екрані зображення одиничного квадрата K, побудувати ще і прямокутник розміром $A \times B$, розбитий на такі самі квадрати? (A, B — натуральні, не більші за 30).

Зразки текстових файлів

paint1.dat	3 2	Значення A і B
paint1.sol	12	Мінімум операцій

Пояснення

Послідовність побудови прямокутника 3×2:



виділити скопіювати вставити перетягти

виділити перетягти

вставити перетягти

виділити скопіювати вставити перетягти

Розв'язання

Цю задачу можна розв'язати в декілька способів, але для того, щоб побудувати прямокутник $A \times B$ (A — висота, B — ширина), доведеться спочатку за найменшу кількість операцій побудувати прямокутник $1 \times B$ (стрічку), а потім — за найменшу кількість операцій потрібний нам прямокутник $A \times B$ (або навпаки, почавши зі стрічки $1 \times A$).

Якби ми вміли обчислювати значення функції $\mathrm{str}(x)$, що визначає найменшу кількість операцій, потрібних для побудови однієї стрічки (прямокутника висотою 1 завширшки x), тоді для побудови потрібного прямокутника $A \times B$ при A, $B > 1$ доведеться виконати $\mathrm{str}(B) + \mathrm{str}(A) - 2$ операцій: $\mathrm{str}(B)$ для побудови однієї стрічки завширшки B, а потім ще $\mathrm{str}(A) - 2$ операцій для побудови A стрічок, накладених одна на одну, тобто наш прямокутник (на 2 менший, оскільки після $\mathrm{str}(b)$ операцій стрічку вже побудовано). Якщо $A = 1$ і $B > 1$, тоді вистачить $\mathrm{str}(B)$ операцій, а при $A = 1$ і $B = 1$ — лише 4 операції.

Помічаємо, що після виконанання операції «виділити», завжди наступною є «скопіювати», а після виконання операції «вставити» завжди наступна — «перетягти». Тобто фактично ми маємо не 4, а лише 2 перетворення (кожне — по 2 операції). Пронумеруємо їх відповідно 1 і 2. Домовимося, що перетворення 1 (якщо вже побудовано хоч один квадрат) означає виділити і скопіювати в буфер всю ту стрічку, що вже побудована. Якщо обмежитися лише цими двома перетвореннями, тоді кількість операцій $\mathrm{str}(x)$ навіть для невеликих значень x стрімко зростатиме. Справді, щоб побудувати стрічку довжиною 32, доведеться виконати одну за одною такі операції (далі «стрічка x» означає «стрічка довжиною x»): виділити, скопіювати в буфер, потім вставити і перетягти одиничний квадрат, ще раз вставити і перетягти квадрат (маємо стрічку 2), виділити і скопіювати цю стрічку, потім вставити і перетягти (маємо стрічку 4), виділити і скопіювати цю стрічку, потім вставити і перетягти (є стрічка 8), потім у таким самий спосіб отримати стрічки 16 і 32. Всього виконано 22 операції. Але якщо нам потрібно мати стрічку 33, тоді доведеться 33 рази вставляти одиничний квадрат, а це можна зробити лише за 4 + 32 * 2 = 68 операцій. І взагалі тоді при непарному x величина $\mathrm{str}(x) = 4 + 2 * (x - 1) 2 * x + 2$ операцій. Що тут можна зробити, щоб зменшити $\mathrm{str}(x)$? «Дозволити» ще одне перетворення (назвемо його 3): копіювати в буфер і потім вставляти і перетягувати одиничний квадрат, що увесь час є на

робочому столі Paint. Тоді, наприклад, стрічка 33 буде побудована такою послідовністю дій: 1 2 2 1 2 1 2 1 2 1 2 3 2 за 26 операцій.

Йдучи з кінця, легко вибрати таку стратегію створення стрічки x (для $x \geq 8$).

Покласти спочатку op:=0;

- поки x парне — виділити і скопіювати, потім вставити і перетягти стрічку довжиною x div 2, користуючись операціями 1 і 2 (при цьому x:=x div2), а op:=op+4;
- при x непарному, виконати операції 3 2 (тоді x:=x-1), а op:=op+4.

Так робити доти, доки не отримаємо x з діапазону 1... 7, а потім збільшити значення op на величину $2+2*x$.

Приклад: для стрічки довжиною 87 матимемо:

87 – 1 = 86 86 : 2 = 43 43 – 1 = 42 42 : 2 = 21 21 – 1 = 20

20 : 2 = 10 10 : 2 = 5 5 – 1 = 4. Але $\text{str}(4)=10$. Послідовність перетворень така: 1 2 2 2 2 (є стрічка 4), 2 (стрічка 5), 1 2 (стрічка 10), 1 2 (стрічка 20), 3 2 (стрічка 21), 1 2 (стрічка 42), 3 2 (стрічка 43), 1 2 (стрічка 86), 3 2 (стрічка 87).

Тому $\text{str}(87)=40$.

Відповідний алгоритм дуже простий:

```
Function str(x:Integer):LongInt;
begin
  op:=0;
  while x>=8 do
  begin
    if x mod 2=0
    then begin op:=op+4; x:=x div 2 end
    else begin op:=op+4; dec(x) end
  end;
  op:=op+2*x+2; str:=op
end.
```

А основна програма матиме вигляд:

```
begin
  Clrscr;wrtite('A,B=');read(A,B);
    if A>B then max:=A else max:=B;
    min:=A+B-max;
    if max=1
    then write(4)
    else if min=1
         then write(str(max))
         else write(srt(min)+str(max)-2)
  readkey;
end.
```

PAINT3D

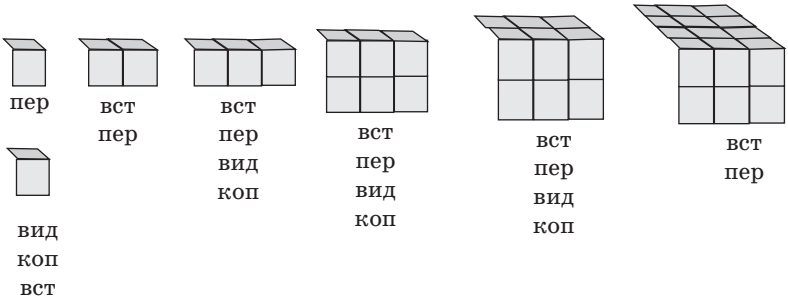
(О-2006)

Фігури на полі графічного редактора Paint можна виділити, скопіювати (повністю), вставити та перемістити (перетягти). Яку найменшу кількість цих операцій потрібно використати, щоб, маючи на полі зображення одиничного куба, побудувати ще одне зображення — прямокутного паралелепіпеда з вимірами $A \times B \times C$, складеного із цих кубів? (A, B, $C \le 100$ — натуральні числа).

Приклади вхідного та вихідного файлів

input.txt	2 3 4	Значення A, B і C
output.txt	20	Мінімум операцій

Пояснення до файла input.txt



Розв'язання

Цю задачу («тривимірну») було запропоновано на Житомирській обласній олімпіаді наступного року після задачі Paint. Зрозуміло, що це фактично та сама задача, що й попередня, але у випадку, коли a, b, $c > 1$, доведеться суму $\mathrm{str}(a)+\mathrm{str}(b)+\mathrm{str}(c)$ зменшувати на 4, а не на 2. Тому вкажемо лише, не змінюючи функції $\mathrm{str}(x)$, як виглядатиме основна програма:

```
begin
  ClrScr;
  write('A,B,C=');read(A,B,C);
  One:=0;
  if A=1 then inc(one);
  if B=1 then inc(one);
  if C=1 then inc(one);
  write(str(A)+str(B)+str(C)-2*one-4);
  readkey
end.
```

Далі ми пропонуємо читачеві задачу про вишиті на одежі малюнки, яка породжує деяку інтригу (що ж то за особливі прикраси, у яких ходить вождь племені?).

ВІЗЕРУНКИ

(О-2006)

Африканське плем'я Кібо, що живе біля підніжжя Кіліманджаро, вважає орла (О), лева (Л) та козла (К) священними тваринами, тому всі візерунки, якими жінки прикрашають одяг, нагадують лише цих тварин, але кожен з них такий, що не містить двох однакових сусідніх фрагментів (наприклад: ОЛКЛОКО, але не ОЛКОЛКЛ). Для вождя племені відшукують особливі візерунки, які не можуть бути фрагментами інших візерунків.

Визначте для вказаного значення N ($N < 30$):

а) скільки всього існує візерунків, які містять N цих тварин;

б) скільки з них таких, що можуть прикрасити одяг вождя.

Приклади вхідного та вихідного файлів

input.txt	3	Кількість тварин N
output.txt	12 0	Можливих візерунків Особливих візерунків

Пояснення

Усі можливі візерунки довжиною 3: ОЛК, ОЛО, ОКЛ, ОКО, ЛОК, ЛОЛ, ЛКО, ЛКЛ, КОЛ, КОК, КЛО, КЛК — всього 12 шт. Кожен з них може бути фрагментом іншого візерунка, наприклад: ОЛК, ЛКЛ, КЛО, ЛОК, ОКО — фрагменти для ОЛКЛОКО тощо.

Розв'язання

Авторові самому подобається ця задача не лише тому, що насправді біля підніжжя Кіліманджаро є схожі племена, а ще й тому, що в наведеному тесті кількість особливих візерунків дорівнює нулю. Це створює і справді інтригу: а що ж то таке — ці «особливі» візерунки? На думку автора, це повинно слугувати «інтенсифікації» роздумів учня, спонукаючи його до максимально можливого застосування добре розвинутої інтуїції.

Замінимо О, Л та К відповідно на A, B, C. Якщо L — один із таких візерунків, то з нього можна отримати ще п'ять візерунків, замінивши A, B, C на будь-яку іншу перестановку цих же букв (а їх усього $3! = 6$), тому загальна кількість їх повинна бути кратною числу 6.

Використовуючи алгоритм перебирання («метод віток і меж»), нескладно для заданого значення довжини візерунка N порахувати їх можливу кількість S. Щоб сформувати будь-який з них, знадобиться функція Viz із значеннями ТАК/НІ (true/false), яка для кожного візерунка L довжини N визначає, чи справді немає в ньому двох однакових сусідніх фрагментів.

Задати цю функцію можна таким чином (\ — знак ділення націло):

```
Viz:="ТАК";
Для j від 1 до N-1 {j - позиція початку фрагмента}
Пц
  Для d від 1 до N\2  {d - довжина фрагмента}
  Пц
    Якщо j+2*d-1≤N і
         L[j:j+d-1]=L[j+d:j+2*d-1]
    То Viz:="НІ"
    Все
  Кц
Кц
```

Але як визначити, чи є візерунок «особливим», яким можна прикрасити одежу вождя?

Рахуючи кількість візерунків S, коли знайдено черговий візерунок L довжини N, створимо ще 6 буквосполучень $R1$, $R2$, ..., $R6$, а саме:

```
R1:="А"+L; R2:="В"+L; R3:="С"+L; R4:=L+"А";
R5:=L+"В"; R6:=L+"С".
```

Якщо виявиться, що значення функції Viz для кожного з них є НІ, то це означатиме, що візерунок L не може бути фрагментом жодного з інших візерунків. Залишиться порахувати кількість SS таких (особливих) візерунків.

Програма, написана мовою Паскаль, наводиться нижче.

Відмітимо деякі результати роботи цієї програми.

Виявляється, що особливі візерунки існують не для кожного N, а лише при $N = 7,15,19,21,23,25,26,27,28,29,30$ (для $N = 1...30$).

Як виглядають ці візернунки?

Для $N = 7$ один з них такий: $ABACABA$.

Справді, якби він був фрагментом іншого візерунка P, то P містив би один із шести фрагментів, які можна отримати, «приклеївши» до $ABACABA$ зліва або справа A, B або C. Але виявляється, що цього саме не можна зробити, бо в утворених буквосполученнях, що претендують на те, щоб стати візерунками або їх фрагментами, тобто $AABACABA$, $BABACABA$, $CABACABA$, $ABACABAA$,

ABACABAB, *ABACABAC*, є сусідні однакові фрагменти (вони виділені жирно та підкреслені).

Зауважимо, що в авторській програмі мовою Паскаль символи *A*, *B*, *C* замінені відповідно на 1, 2, 3, а у вихідний файл можна надрукувати і всі можливі особливі візерунки.

Враховуючи той факт, що в задачі не вимагається наводити візерунки, а лише порахувати їх кількість, роботу програми можна пришвидшити втричі, якщо згадати про те, що кількість ланцюжків кратна $3! = 6$. Для цього в програмі слід лише змінити цикл for i:=1 to 3 do на for i:=1 to 1 do і потім відповіді *S* та *SS* збільшити втричі.

```
{-----------------------------------------}
uses crt;
var q,i,j,k,n,l,d:Integer;
     s,ss:LongInt;
     p:array[1..30] of 1..3;
     st,nt:string;
     f:boolean;
     ff:Text;
{----Чи є однакові сусіди-------------------}
function viz(u:string):boolean;
begin
  viz:=true;
  for l:=1 to length(u)-1 do
  for d:=1 to length(u) div 2 do
  if (l+2*d-1<=length(u))
     and (copy(u,l,d)=copy(u,l+d,d))
  then viz:=false;
end;
{------Основна програма---------------------}
begin
  clrscr;
  write('N=');read(N);s:=0;ss:=0;
  assign(ff,'pattern0.dat');rewrite(ff);
  for i:=1 to 3 do
  begin
    q:=1;p[1]:=i;st:=chr(48+i);k:=1;
    while q>0 do
    begin
      if q=N
      then begin
             inc(s);
      if not(viz (st+'1'))
```

```
          and not(viz (st+'2'))
          and not(viz (st+'3'))
          and not(viz ('1'+st))
          and not(viz ('2'+st))
          and not(viz ('3'+st))
       then begin {крок назад}
             inc(ss);write(ff,st,' ')
            end;
            {for j:=1 to q do
             write(p[j]);write(' ');}
             k:=p[q];dec(q);
             st:=copy(st,1,q);
       end
    else begin
          if k<=3
          then begin
                if viz (st+chr(48+k))
                then begin {крок}
                      inc(q);p[q]:=k;
                      st:=st+chr(48+k);
                      k:=0
                     end
               end
            else begin {крок назад}
                  k:=p[q];dec(q);
                  st:=copy(st,1,q)
                 end
               end;
         inc(k)
       end;
    end;
    writeln('Всього: ',s,' Особливих: ',ss);
    writeln(ff,ss);close(ff); readln;readln
  end.
```

ПУТІВКИ

(О-2006)

Туристична фірма не встигла через люті морози продати N ($N<15$) путівок на гірськолижні бази, термін дії яких вже настав. Щоб зменшити втрати, було вирішено з 1 лютого всі такі путівки, яким

залишилося $d[k]$ днів, продавати за номінальною вартістю — по $c[k]$ грн за день лише за ті дні, які залишилися з дня продажу ($k=1...N$).

На яку найбільшу суму можна реалізувати ці путівки, якщо кожного дня продавати по одній путівці?

($c[k]\le 100$, $d[k]\le 30$, $k=1...N$).

Приклади вхідного та вихідного файлів

pass3.dat			pass3.sol	
4 2 3 1 4	 37 45 46 30	Кількість путівок N N рядків по два числа — кількість днів $d[k]$ і вартість дня $c[k]$ ($k=1...N$)	232	Максимальна сума, грн

Пояснення

Найкращий варіант реалізації путівок такий: перший день — по 45 грн, другий — по 37 грн, третій — по 30 грн за день. Всього на суму 3 x 45 + 1 x 37 + 2 x 30 = 232 грн.

Розв'язання

Це звичайна перебірна задача. Справді, якщо в k-й день (тобто, k-го лютого) продано путівку на базу з номером i, то зрозуміло, що покупець платитиме лише за $d[i]-k+1$ днів, тому її вартість становитиме $c[i]*(d[i]-k+1)$ грн, при цьому величина $d[i]-k+1$ обов'язково повинна бути додатною. Це і буде основною вимогою для створення можливих варіантів продажу путівок і вибору з них оптимального, що набагато зменшує їх можливу кількість, що сягає $N!$. Програма, наведена нижче, з дидактичних міркувань залишена в такому вигляді, щоб можна було показати динаміку пошуку кращого варіанту продажу (тобто всі найкращі варіанти).

```
Uses crt;
var d,c,p: array[1..15] of Integer;
    n,k,q,su,i,j,max:Integer;
    s:set of Byte;

{-----------------Основна програма-------------}
begin
  Clrscr;write('N=');read(N);
  write('Днів та вартість одного дня:');
  Fot k:=1 to N do read(d[k],c[k]);
  {Перебирання варіантів реалізації backtracking}
  Max:=0;{Сума, яку потрібно обчислити}
  for i:=1 to n do {i - номер путівки}
```

```
begin
  q:=1;p[1]:=I;s:=[i];su:=d[i]*c[i];k:=1;
  while q>0 do
begin
  if k<=n
  then begin
        if([k]*s=[]) and (d[k]>=q+1)
        then begin {крок}
              inc (q); p[q]:=k;s:=s+[k];
              su:=su+(d[k]-q+1)*c[k];
              k:=0
            end
        end
     else begin
           if su>max
           then begin
                 max:=su;
`                writeln ('найкращий варіант:');
                 write ('Порядок:');
                 for j:=1 to q do
                 write (p[j],' '); writeln;
                 writeln ('Сума:',su,
                 ' путівок:',q);
               end;
           k:=p[q];s:=s-[k];
           su:=su-(d[k]-q+1)*c[k]; dec(q)
           {крок назад}
           end
     inc(k);
    end;
  end;
 readkey;
end.
```

РОЗМІН

(Р-2000)

Розміняти M гривень купюрами $a_1, a_2, \ldots, a_n$, використовуючи найменшу кількість видів купюр.

Приклади вхідного та вихідного файлів

input.txt	7	Значення M і N Один рядок — масив $a[1\ldots N]$
output.txt	6	Мінімальна кількість видів купюр

Розв'язання

Хоч цю задачу і вважають давно відомою, так би мовити «класичною», задачі подібного типу зустрічаються в нашому житті останнім часом дедалі частіше, як колись у 2000 році. І справа не в тому, що збільшилася кількість розмінів сум на купюри. Навпаки, розмінів стало, мабуть, менше (не вистачає готівки). Зате дедалі більше підприємств, як і в 2000-му році, розраховуються одне з одним не грішми, а так званим бартером (виробами на суму боргу), а підприємства і місцева влада з громадянами замість зарплати — товарами. Найбільш уживані номінали української валюти становлять 100, 50, 20, 10, 5, 2 та 1 гривню. Товари ж, якими доводиться розраховуватися, можуть мати інші ціни. Тому, щоб узагальнити задачу, можна було б сформулювати її дещо інакше, наприклад таким чином.

На M гривень купити найменшу кількість різних іграшок за ціною $a_1, a_2, \ldots, a_n$ гривень (всі числа — натуральні).

Розмін M гривень вказаними купюрами (або придбання іграшок) відбудеться лише за умови, що нам вдасться відшукати такі невід'ємні цілі $x_1, x_2, \ldots, x_n$, що сума добутків $a_1x_1 + a_2x_2 + \ldots + a_nx_n$ дорівнює M. При цьому потрібно використати найменшу кількість видів купюр (найменше різних іграшок), тобто найменшою повинна бути кількісь ненулів серед чисел $x_1, x_2, \ldots, x_n$.

Таким чином, потрібно перебрати всі можливі варіанти розміну, щоб вибрати з них оптимальний, який задовольняє вказану умову.

Перше, що спадає на думку, — це алгоритм із вкладеними один в одного n циклами, який перебирає всі x_i від 0 до $M \setminus a_i$ ($i = 1 \ldots n$), де $M \setminus a_i$ — ціла частина дробу M / a_i.

Але, на жаль, невідомо, як записати цей алгоритм у загальному випадку, до того ж він трудомісткий (неекономний), тому що перераховує і ті випадки, коли вказана сума, більша за M, а також сума остачі, більша за найбільше з чисел a_i ($i = 1 \ldots n$).

Потрібен інший алгоритм. Спосіб, що буде застосований, є різновидом перебирання і схожий із методом «динамічного програмування».

Ви помічали, як касири дають у магазині решту?

Спочатку найбільшу кількість найбільших купюр, потім — менших тощо.

Нехай S — сума, яку потрібно розміняти. Тоді:

```
для k від 1 до n
пц
  xk:=S\ak;
  S:=S mod ak
кц
```

Зрозуміло, що розмін буде вдалим лише тоді, коли після виконання цього циклу ми матимемо $S = 0$.

Такий спосіб в інформатиці названо «жадібним». На ньому і ґрунтується потрібний алгоритм. Спробуємо пояснити, як він виглядає.

Нехай потрібно розміняти решту RESH_k гривень купюрами $a_{k+1}, a_{k+2}, \ldots, a_n$.

Зрозуміло, що її можна заплатити купюрами номіналом a_{k+1}, взявши їх у кількості $x_{k+1} = \mathrm{RESH}_k \setminus a_{k+1}$. При цьому залишиться ще сплатити решту $\mathrm{RESH}_{k+1} = \mathrm{RESH}_k \bmod a_{k+1}$. Потім цю решту заплатити номіналами в a_{k+2} тощо до номіналу a_n включно.

Це нескладно зробити циклом:

```
для i від k +1 до n
пц
  xi:=RESHi-1 \ ai
  RESHi:=RESHi-1 mod ai
кц
```

Назвемо цю операцію $\mathrm{ROZMIN}(k)$. До речі, якщо покласти $\mathrm{RESH}_0 = M$, то $\mathrm{ROZMIN}(0)$ за умови, що $\mathrm{RESH}_n = 0$ дасть перший можливий варіант розміну «жадібним» способом потрібної суми M гривень.

Щоб перебрати всі можливі варіанти розміну, знайдемо, починаючи з кінця (точніше, з купюри a_{n-1}), такий найбільший індекс no, що $x_{\mathrm{no}} > 0$, тобто знайдемо серед чисел $x_1, x_2, \ldots, x_{n-1}$ ненуль із найбільшим індексом. Якщо всі числа $x_1, x_2, \ldots, x_{n-1}$ — нулі, вважатимемо, що інших варіантів розміну нема.

Тобто

```
no:=n-1; поки no>0 і xno =0 пц no:=no-1 кц
```

Вважатимемо, що $\mathrm{no} > 0$.

Зменшимо тепер число x_{no} на одиницю, тобто купимо на одну іграшку вартістю x_{no} менше.

При цьому величина $\mathrm{OST}_{\mathrm{no}}$ збільшиться на a_{no}. Назвемо цю операцію пошуку найдальшого ненуля серед чисел x_i, скажімо, NENUL. Нагадаємо, що вона відшукує значення індекса no.

Якщо $\mathrm{no} > 0$, то далі розміняємо цю решту $\mathrm{RESH}_{\mathrm{no}}$ купюрами вартістю $a_{\mathrm{no}+1}, \ldots, a_n$, виконавши операцію $\mathrm{ROZMIN}(\mathrm{no})$. При цьому у випадку вдалого розміну буде знайдено новий його варіант, який слід порівняти з першим, а якщо він до того ж оптимальніший за попередній, то слід запам'ятати кількість потрібних для розміну видів куп'юр min і відповідний масив x, позначивши його, наприклад, xx (назвемо цю операцію VYBIR).

Повторюючи операцію NENUL та VYBIR, ми переберемо всі можливі варіанти розміну і знайдемо найменшу кількість видів купюр min найбільш оптимальному з них.

Доцільно початкове значення min узяти таким, що дорівнює $n+1$ (заздалегідь неможливий варіант, адже всього купюр n видів, тобто значення min завжди менше від $n+1$). Однак $n+1$ — дуже зручне початкове значення для min. І ось чому. Перебравши всі варіанти, ми можемо отримати значення $\min = n+1$. А це означатиме, що жодного варіанту розміну знайти не вдалося.

Розв'язання задачі виглядатиме так:

```
min:=n+1;
ROZMIN(0);
повторити
  NENUL;
    якщо no>0
  то ROZMIN(no);
    VYBIR
  все
до no=0;
якщо min=n+1
то вивести("розмін неможливий")
інакш вивести("всього видів купюр",min);
      вивести("а саме:");
      для k від 1 до n
      пц
         якщо xxk>0
         то вивести (xxk," по ", ak)
         все
      кц
```

Тут VYBIR — операція, що має такий вигляд:

```
R:=0;
для k від 1 до n
пц якщо xk>0 то R:=R+1 все кц
якщо RESTk=0 і R<min
то min:=R;
   для k від 1 до n пц xxk:=xk кц
все
```

(У наведених вище операціях замість позначень елементів масивів $x[k]$, $xx[k]$, $a[k]$ та $\mathrm{RESH}[k]$ використані більш прийняті в математиці позначення для послідовностей відповідно x_k, xx_k, a_k та RESH_k. Крім того, застосовано неіснуючий в алгоритмічній мові цикл типу ДО, який краще записати мовою Паскаль).

Нижче наведено програму, написану мовою Паскаль, яка відшукує потрібний варіант розміну, побудований на тих самих ідеях.

```
uses crt;
type mas=array[1..100] of Integer;
var m,n,min,no,i, j, k: Integer;
     a,x,xx,ost: mas;

{---Розмін ost[t] купюрами a[t+1]...a[n]------}
procedure rozmin(t:Integer);
begin
  for k:=t+1 to n do
  begin
    x[k]:=ost[k-1] div a[k];
    ost[k]:=ost[k-1] mod a[k]
  end;
end;
{---Пошук найдальшого ненуля серед x[1..n-1]----}
procedure nozero;
begin
  no:=n-1;while(no>0) and (x[no]=0) do dec[no];
  if no>0
  then begin dec(x[no]);inc(ost[no],a[no]) end
end;
{-----Вибір найкращого варіанту---------------------}
procedure vybir;
begin
     R:=0;
     for k :=1 to n do if x[k]>0 then inc(R);
     if (ost[k]=0) and (R<min)
     then begin min:=R; xx:=x end;
end;
{--------Основна програма----------------------}
begin
   clrscr;
   write('Кількість видів купюр ');read(n);
   writeln('Які номінали ');
   for k:=1 to n do read(a[k]);
   rozmin(0);
  repeat
   nozero;
   if no>0
   then begin rozmin(no);vybir end;
  until no=0;
if min=n+1
```

```
then write('розмін неможливий')
else begin
      writeln('всього видів купюр ',min);
      writeln('а саме:');
      for k:=1 to n do
      if xx[k]>0
      then write(xx[k],' по ', a[k],' ');
    end;
   readln;readln
end.
```

А тепер нова, досить проста задача.

ВІДРІЗКИ

(Дідковський В. Л., Матвійчук С. В., О-2008)

Розріжте відрізок завдовжки L ($L = 4 \ldots 2000000000$) на найбільшу кількість частин, довжини яких — натуральні числа, щоб з них не можна було скласти жодного трикутника.

Приклади вхідного та вихідного файлів

input.txt	6	Значення L
output.txt	3	Найбільша кількість частин

Пояснення

Відрізок завдовжки 6 можна розрізати на 3 відрізки — 1, 2, 3 або 1, 1, 4, з яких неможливо скласти трикутника.

Розв'язання

Зрозуміло, що кількість відрізків N повинна бути не меншою за 3. Нехай $a[1 \ldots N]$ — неспадний масив, сума якого дорівнює заданому L, де $N \geq 3$. Щоб кількість частин N була якомога більшою, починати масив a потрібно з одиниці, навіть з двох: 1, 1, ... Тоді відрізок завдовжки $L = 4$ доведеться розрізати лише на три частини 1, 1, 2, з яких неможливо скласти трикутника. Для $L = 5$ матимемо три частини 1, 1, 3, для $L = 6$ — знову три частини (як у прикладі файла input1.dat). Для $L = 7$ крім трьох частин 1, 2, 4 є можливість розрізати відрізок на чотири частини 1, 1, 2, 3.

Можна помітити, що кількість частин збільшується на одиницю саме в тих випадках, коли кожний наступний відрізок, починаючи з третього, дорівнює сумі двох попередніх, тобто масив a є частиною відомого ряду Фібоначчі: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55...

(Якщо довжина L — сума деякої кількості M ($M > 3$) членів ряду Фібоначчі $S_M = F_1 + F_2 + \ldots + F_M$, тоді кількість частин розрі-

зання $P_M = P_{M-1} + 1$, де F_k — k-й член ряду. Для значень L з відрізка $[S_{M-1}, S_M - 1]$ вона залишається однаковою і дорівнює P_{M-1}).

Легко скласти й алгоритм розрізання відрізка завдовжки L на потрібні N частин. Його можна викласти, наприклад, таким чином:

```
a:=1;   b:=2;
N:=2; z:=3;
поки z<L
пц
     c:=a+b; a:=b; b:=c;
     z:=z+c; N:=N+1
кц
```

Нижче наводиться незвично проста програма мовою Паскаль, що розв'язує вказану задачу.

```
{-----------------------------------------------}
var a,b,c,L,N,z : LongInt;
     fs : Text;
begin
  clrscr;
  readln(L);
  a:=1; b:=2;
  N:=2; z:=3;
  while z<L do
  begin
    c:=a+b; a:=b; b:=c;
    z:=z+c; N:=N+1
  end;
  assign(fs,'output.sol');
  rewrite(fs);
  writeln(fs,N);
  close(fs);
end.
```

ВІДРІЗКИ-2

(Дідковський В. Л., Матвійчук С. В., О-2009)

Відрізок завдовжки L розрізали на найбільшу кількість частин, довжини яких — натуральні числа, з яких не можна скласти жодного трикутника. Обчисліть кількість можливих варіантів розрізання. (Два розрізання вважаються різними, якщо впорядковані масиви довжин їх частин відрізняються хоча б одним елементом.)

Вхідні дані

Значення $L = 4 \ldots 200$.

Вихідні дані

Єдине число — кількість варіантів розрізання.

Приклад введення	Приклад виведення
11	4

Пояснення

Відрізок завдовжки 11 можна в чотири способи розрізати на 4 відрізки, з яких неможливо скласти жодного трикутника: 1, 1, 2, 7, або 1, 1, 3, 6, або 1, 1, 4, 5, або 1, 2, 3, 5.

Розв'язання

Це, так би мовити, серійна задача: продовження задачі олімпіади 2008 року про найбільшу кількість N частин, на які можна розрізати відрізок завдовжки L (всі числа натуральні) так, щоб з них не можна було скласти жодного трикутника.

Нагадаємо, про що йдеться. Нехай маємо відрізок завдовжки $L = 11$. Тоді його можна розрізати щонайбільше на $N = 4$ частини, з яких неможливо скласти трикутника, наприклад такі: 1 1 2 7.

Ті, хто розв'язав задачу 2008 року, пам'ятають, що це дещо змінена задача про відомі числа Фібоначчі $f(k)$ (кожне число, починаючи з третього, дорівнює сумі двох попередніх): 1 1 2 3 5 8 13 21 34 55..., які легко обчислити. А наведене вище розрізання 1 1 2 7 легко отримується, якщо взяти $N-1$ перших чисел Фібоначчі, але останім вписати число $f(N)+L-\mathrm{sf}(N)$, де $\mathrm{sf}(k)$ — сума k перших чисел Фібоначчі: 1 2 4 7 12 20 33...

Але для значення $L = 11$ існують й інші довжини відрізків: 1 1 3 6, 1 1 4 5, 1 2 3 5. Тобто кількість можливих варіантів дорівнює 4.

Відрізок завдовжки $L = 19$ можна розрізати на п'ять частин таким чином: 1 1 2 3 12, 1 1 2 4 11, 1 1 2 5 10, 1 1 2 6 9, 1 1 3 4 10, 1 1 3 5 9, 1 2 3 5 8 (всього сім варіантів). Разом з тим відрізок, що має довжину $L = 7$, лише одним способом можна розрізати на чотири відрізка: 1 1 2 3.

Для випадку $L = 19$ розглянемо довжини частин. Перепишемо ці числа ще раз, при цьому в дужках між кожними двома числами впишемо величину, на яку наступне число більше за суму двох попередніх. Матимемо:

1(0)1(0)2(0)3(7)12
1(0)1(0)2(1)4(5)11
1(0)1(0)2(2)5(3)10
1(0)1(0)2(3)6(1)9

1(0)1(1)3(0)4(3)10
1(0)1(1)3(1)5(1)9
1(1)2(0)3(0)5(0)8

Числа, записані в дужках, якщо їх взяти у зворотному порядку, є не що інше, як кількості «монет» номіналом 1 2 4 7 (частина масиву сум перших чисел Фібоначчі 1 2 4 7 12 20 33 — див. вище), якими можна розміняти суму $L-\mathrm{sf}(N)$.

$$7*1+0*2+0*4+0*7=7$$
$$5*1+1*2+0*4+0*7=7$$
$$3*1+2*2+0*4+0*7=7$$
$$1*1+3*2+0*4+0*7=7$$
$$1*1+3*2+0*4+0*7=7$$
$$0*1+1*2+1*4+1*7=7$$
$$0*1+0*2+0*4+1*7=7$$

Тобто задача обчислення кількості варіантів розрізань звелася до обчислення кількості варіантів розміну суми $L-\mathrm{sf}(N)$ «купюрами» номіналом $\mathrm{sf}[1]$, $\mathrm{sf}[2]$, $\mathrm{sf}[3]$... А це давно відома класична задача «розмін», наведена нами раніше, яку знають усі, хто цікавиться інформатикою. Сподіваємося, тепер її знають і ті хто вперше ознайомився з про нею на сторінках цієї книжки.

Ось одна з можливих програм:

```
uses crt;
var f,sf,ost,z:array[0..45] of LongInt;
    m,i,j,war,nn,n:Integer;
    r,l:LongInt;
{---------------Жадібний алгоритм-------------}
Procedure jadn(u:Integer);
var i:Integer;
begin
  for i:=u+1 t   then begin
        inc(war);{write('war=',war,':');
        for i:=1 to m do write(z[i],' ' );
        writeln}
      end
end;
{-------Останній ненуль---------------------}
Procedure nenul;
begin
   nn:=m-1;
   while (nn>0) and (z[nn]=0) do dec(nn);
end;
{----------Числа Фібоначчі-----------------}
```

```
procedure fi;
var a,b,c,s:LongInt;
begin
  n:=2;a:=1;b:=1;s:=2;
  f[1]:=1;f[2]:=1;sf[1]:=1;sf[2]:=2;
  while l>=s do
  begin
    inc(n);c:=a+b;s:=s+c;f[n]:=c;sf[n]:=s;
    a:=b;b:=c;
  end;
  dec(n);
  {for i:=1 to n do
  if i<n
  then write(f[i],' ')
  else write(f[n]+l-sf[n]);writeln;}
end;
{------------Основна програма------------------}
begin
   Clrscr;write('L=');read(l);
   fi;
   r:=l-sf[n];writeln('R=',r);
   m:=1;while r>=sf[m] do inc(m); dec(m);
  { write('Купюри:');
   for i:=1 to m do write(sf[i],' ');writeln;}
   ost[0]:=r; jadn(0);
  repeat
  nenul;
  if nn>0 then begin
                 dec(z[nn]);
                 inc(ost[nn],sf[nn]);
                 jadn(nn)
               end;
  until nn=0;
  writeln('Варіантів: ',war);
  readkey
end.
```

ПАЛИВО

(О-2007)

N котелень однакової потужності сполучені системою трубопроводів з M труб для перекачування палива. На 9:00 ранку виявилося, що фактичні запаси палива $A[k]$ т ($k=1\ldots N$) такі, що в од-

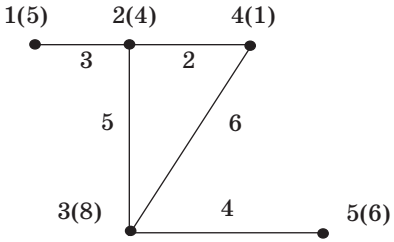
ній з котелень його значно менше за норми у Вт, а на інших — вдосталь або понад норму.

Сумарні запаси дозволяють виправити ситуацію, якщо перерозподілити паливо. У кожний момент часу з N насосів можуть працювати 0 або 2 (в сусідніх котельнях, що перекачують та приймають паливо), при цьому перекачування 1 т палива на 1 км забирає C хв.

За який найменший час T хв ця робота буде виконана? (Усі дані — невід'ємні цілі числа, не більші за 50.)

Приклади вхідного та вихідного файлів

input.txt	5 5 4 6 5 4 8 1 6 1 2 3 2 3 5 2 4 2 3 4 6 3 5 4	Значення N, M, B, C Один рядок — масив $a[1...N]$ M рядків по 3 числа — пари номерів котелень та довжини труб між ними
output.txt	102	Значення T, хв



Пояснення

Найменший час 102 хв буде досягнуто в разі перекачувань: з 1 на 2 (1 т) за 18 хв, з 2 на 4 (1 т) за 12 хв, з 3 на 4 (2 т) за 72 хв. Всього 102 хв.

Розв'язання

Ось зрозуміла програма мого співавтора Матвійчука С. В., що не потребує коментарів. Вона краща за авторське розв'язання цієї задачі на графи.

```
Uses crt;
const Fd = 'input.txt';
      Fs = 'output.txt';
      Nmax = 20;
var   N,M,B,R,T,D : Integer;
      ff : boolean;
      W : array [1..Nmax,1..Nmax] of LongInt;
      A,z,y : array [1..Nmax] of Integer;
{------------Обмін комірок---------------------}
Procedure Swap(var u,v : Integer);
var h : Integer;
```

```
begin h:=u; u:=v; v:=h end;
{---------Сортування масиву A-----------------}
Procedure Sort;
var i :Integer;
    F : boolean;
begin
  repeat
   F:=True;
   for i:=1 to N-1 do
    if A[i]>A[i+1]
    then begin
          Swap(A[i],A[i+1]);
          Swap(y[i],y[i+1]);
          F:=false
         end;
  until F;
end;
{---------Вхідні дані--------------------------}
Procedure Init;
var p,q,l,i,j : Integer;fi : Text;
begin
  assign(fi,FileIn);
  reset(fi);
  readln(fi,N,M,R,B);
  for i:=1 to N do
  begin read(fi,A[i]); y[i]:=i end;
  Sort;
  for i:=1 to N do z[y[i]]:=i;
  for i:=1 to N do
  for j:=1 to N do
  if i=j then W[i,j]:=0 else W[i,j]:=31999;
  for i:=1 to M do
begin
   readln(fi,p,q,l);
   W[z[p],z[q]]:=l; W[z[q],z[p]]:=l;
  end;
  close(fi);
end;
{-----------------------------------------------}
Function Check (var i : Integer) : boolean;
begin
  i:=0;
  repeat
```

```
    inc(i);
  until (i=N) or (a[i]<R);
  Check:=not(i=N);
end;
{---------Найкоротші відстані------------------}
Procedure Run;
var i,j,k,l,p : Integer;
begin
  for i:=1 to N do
    for j:=1 to N do
      for k:=1 to N do
        if W[i,j]>W[i,k]+W[k,j]
        then begin
               W[i,j]:=W[i,k]+W[k,j];
               W[j,i]:=W[i,k]+W[k,j];
             end;
  while Check(i) do
  begin
    l:=31999;
    for j:=i+1 to N do
      if (W[i,j]<l) and (a[j]>R)
      then begin l:=W[i,j]; k:=j end;
      if a[i]>a[k] then p:=R-a[i] else p:=a[k]-R;
      T:=T+l*p*B;
      inc(a[i],p);
      Dec(a[k],p);
  end;
end;
{-----------------------------------------------}
Procedure Done;
var fo : Text;
begin
  assign(fo,Fs);
  rewrite(fo);
  writeln(fo,T);
  close(fo);
end;
{-------------Основна програма-----------------}
begin
  Init; Run; Done
end.
```

ПОЛІГОН

(О-2009)

Військовий полігон має форму N-кутника. Якщо вилучити одну з вершин, то отримаємо опуклий многокутник. Встановити номер цієї вершини. ($N = 4\ldots20$).

Вхідні дані

В першому рядку — значення N, у наступних N рядках по два числа — координати послідовних вершин многокутника.

Вихідні дані

Єдине число — номер вершини.

Приклад введення
5
3 7
7 3
3 3
0 0
1 4.5

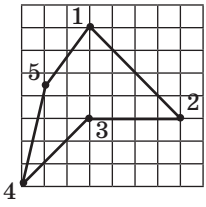


Рисунок до файла даних

Приклад виведення
3

Розв'язання

Очевидно, доведеться перебрати всі вершини $i = 1\ldots N$ і вибрати лише ті з них, в разі вилучення яких ми отримаємо опуклий $(N-1)$-кутник. Нехай $x[i]$, $y[i]$ — координати послідовних вершин заданого многокутника. Визначимо для кожного $u = 1\ldots N$, які координати $xx[k]$, $yy[k]$ ($k = 1\ldots N-1$) матиме новий многокутник, якщо вилучити вершину із заданим номером u.

```
Procedure koord(u:Integer;var xx,yy:mas);
begin
  for k:=1 to N-1 do
    if k<i then begin xx[k]:=x[i];yy[k]:=y[i] end
           else begin xx[k]:=x[k+1];yy[k]:=y[k+1] end
end.
```

Надалі нам потрібно мати надійний алгоритм, який перевіряє, чи є многокутник з вершинами $k = 1\ldots N-1$ і координатами ($xx[k]$, $yy[k]$) опуклим. Він повинен мати вигляд деякої функції, аргументами в якій є два масиви координат $xx[k]$, $yy[k]$ для $k = 1\ldots N-1$. Можна придумати декілька способів, як скласти такий алгоритм. Але найбільш вдалий з них той, який ми використали в задачі «операція», коли розглядали опуклі оболонки множини заданих

точок. Справді, многокутник опуклий лише тоді, коли він має таку властивість: усі його вершини (крім двох кінців кожної зі сторін) лежать по один бік від прямої, яка проходить через цю сторону.

Нагадаємо, що кожна пряма має рівняння вигляду $ax+by+c=0$, де коефіцієнти рівняння a, b, c визначаються процедурою abc, у якій $u=1...n$ — номер сторони.

```
Procedure abc(var u:Integer);
var dx,dy:real;
begin
  if u<n
  then begin dx:=xx[u+1]-xx[u];dy:=yy[u]-yy[u+1] end
  else begin dx:=xx[1]-xx[n];dy:=yy[n]-yy[1] end;
  a:=dy;b:=dx;c:=-(a*xx[u]+b*yy[u])
end.
```

Як правило, в задачах кількість сторін менша за 256, тому, позначивши через plus, minus — множини індексів тих вершин многокутника, у яких ліва частина рівняння прямої $ax+by+c$ має відповідно додатне і від'ємне значення, алгоритм у вигляді функції opuk (зі значеннями типу boolean) можна побудувати таким чином:

```
Function opuk(u:Integer):boolean;
var j:Integer;
begin
  abc(u);plus:=[ ];minus:=[ ];
  for j:=1 to n-1 do
  begin
    if a*xx[j]+b*yy[j]+c>0 then plus:=plus+[j]
    else if a*xx[j]+b*yy[j]+c<0 then minus:=minus+[j];
  end;
  Opuk:= (plus=[ ]) or (minus=[ ])
end.
```

Нижче подано програму, записану мовою Паскаль, що визначає множину номерів тих вершин заданого N-кутника, які задовольняють умову задачі (тобто в разі вилучення яких залишиться опуклий $(N-1)$-кутник).

```
Uses crt;
Type mas=array[1..250] of real;
var j,i,k,N:Integer;
     a,b,c:real;
     plus,minus,s:set of Byte;
     x,y,xx,yy:mas;
     Fd,fs:Text;
{--------Читання даних-------------------------}
```

```
procedure data;
begin
  assign(fd,'input.txt');reset(fd);
  readln(fd,n);
  for k:=1 to n do readln(fd,x[k],y[k]);
  close(fd);
end;
{----------Координати вершин (N-1)-кутника------}
Procedure koord(u:Integer);
begin
  for k:=1 to N-1 do
  if k<u
  then begin xx[k]:=x[k];yy[k]:=y[k] end
  else begin xx[k]:=x[k+1];yy[k]:=y[k+1] end
end;
{----------Коефіцієнти рівняння прямої---------}
Procedure abc(var u:Integer);
var dx,dy:real;
begin
  if u<n-1
  then
  begin dx:=xx[u+1]-xx[u];dy:=yy[u]-yy[u+1] end
  else
  begin dx:=xx[1]-xx[n-1];dy:=yy[n]-yy[1] end;
  a:=dy;b:=dx;c:=-(a*x[u]+b*y[u])
end;
{--Опуклість многокутника при вилученні u------}
Function opuk(u:Integer):boolean;
var j:Integer;
begin
  abc(u);plus:=[];minus:=[];
  for j:=1 to n-1 do
  begin
    if a*xx[j]+b*yy[j]+c>0
    then plus:=plus+[j]
    else if a*xx[j]+b*yy[j]+c<0
         then minus:=minus+[j];
  end;
  Opuk:= (plus=[]) or (minus=[])
end.
```

```
{----------------Основна програма-------------}
begin
  Data; S:=[];
  for k:=1 to N do
  begin
    Koord(k);
    if opuk(k) then s:=s+[k];
  end;
end.
```

Нагадаємо: у файлі відповіді fs повинно бути записане лише будь-яке одне з чисел, що належать множині S.

СУМА

(О-2010)

При якому найменшому натуральному N сума $1^1+2^2+3^3+\ldots+N^N$ без остачі ділиться на натуральне M ($M=2\ldots1000$)?

Приклад

Вхідні дані $M=3$. Вихідні дані $N=4$.

Розв'язання

Оскільки в умові не вимагається обчислювати значення вказаної в задачі суми, яка швидко зростає із зростанням N і не пізніше $N=11$ вже перевищуватиме LongInt, працюватимемо не з цими значеннями, а з остачами від ділення цієї суми на M.

Це зробить короткий, легко зрозумілий алгоритм, викладений у наведеній нижче програмі.

```
Uses crt;
var Sum,s,N,M,i:LongInt;
begin
   Clrscr;
     write('M=');read(M);
     Sum:=1;N:=1;
   while Sum mod M>0 do
   begin
     inc(N);s:=1;
     for i:=1 to N do
     begin s:=(s*(s mod M)) mod M End;
     Sum:=(sum+s) mod M end;
   writeln(N);
   readkey
end.
```

ПІДРУЧНИКИ

(I-2006)

У N шкіл району завезено пробні підручники з інформатики з розрахунку по b шт. на кожну. Через помилку водія в деякі школи завезено більше книжок, тому в деяких їх не вистачає. (Всі значення менші від 16.)

Обчисліть найменшу кількість рейсів R, які потрібно зробити, щоб виправити становище, якщо за один рейс можна перевезти зі школи в іншу школу до b книжок.

Для кожного з рейсів вкажіть три числа: з якої школи, у яку школу, скільки книжок перевезти. (Вкажіть один з можливих розв'язків.)

Приклади вихідних та вихідних файлів

input.txt	4 6 13 4 1	Значення N Один рядок: N чисел — кількості завезених у школи книжок
output.txt	2 2 3 2 2 4 5	Значення R R рядків по три числа: звідки, куди, скільки книжок перевезти

Розв'язання

Наведемо ґрунтовне розв'язання (з доволі довгими поясненнями) цієї цікавої задачі, яке потребує від читача зосередженості й особливої уваги.

Нехай a[1:N] — масив кількостей завезених у школи книжок. Зрозуміло, що сума його елементів дорівнює $N*b$. Вважатимемо, що дані в задачі коректні (загальна кількість книжок кратна числу N), тому легко визначаємо потрібне значення b.

Розглянемо окремо кожну з двох частин задачі.

Найменша кількість рейсів

Замість масиву a введемо більш зручний інший масив $c[1\ldots N]$, де $c[i]=a[i]-b$. Тоді сума елементів масиву c дорівнює нулю. Назвемо такий цілочисельний масив нормованим.

Зрозуміло, ще задача про вирівнювання кількостей книжок рівнозначна задачі про перетворення нормованого масиву на масив, складений з нулів (про обнулення масиву c). Тоді рейсом слід назвати операцію:

```
  c[i]:=c[i]-x
c[k]:=c[k]+x
```

для довільних натуральних i, k, x, таких що i, k, $c[i]>0$, $c[k]<0$; k=1..N: x<=c[i].

Нехай NP — кількість елементів масиву c, таких що $c[i]>0$, а NM — кількість елементів масиву c, таких, що $c[i]<0$.

(Домовимось нулі нормованого масиву не рахувати, вони не потребують рейсів.)

Встановимо деякі властивості нормованих масивів.

Теорема 1

У нормованому масиві з N елементами потрібно не менше ніж M рейсів, щоб усі елементи його стали нулями, де M — найбільше з двох чисел — NM і NP.

Доведення

Справді, потрібно зробити нулями всі додатні елементи, а для цього доведеться застосувати не менше ніж NP рейсів. Крім того, щоб усі від'ємні елементи стали нулями, доведеться застосувати не менше ніж NM рейсів.

Теорема 2

У нормованому масиві з N елементами вистачить $(N-1)$ рейсів, щоб обнулити усі його елементи.

Доведення

1. Нехай маємо масив з N елементів, у якому один елемент c_N від'ємний, а решта — додатні. Тривіально зрозуміло, що для обнулення масиву потрібно $(N-1)$ рейсів. (Відзначимо, що аналогічно для «дзеркально протилежного» випадку: один елемент додатний, решта від'ємні.)
2. Нехай у масиві з N елементів два c_{N-1} і c_N від'ємні, інші додатні.

   Обнулимо спочатку c_{N-1}.

   Для цього в сумі $(c_1+c_2+\ldots+c_k)+c_{N-1}$ беремо в дужках стільки доданків k ($k<N-2$), щоб ця сума стала невід'ємною. Тобто шукаємо найменше k таке, щоб $(c_1+c_2+\ldots+c_k)+c_{N-1}\ge 0$, але $(c_1+c_2+\ldots+c_k)+c_{N-1}+c_N<0$.

   Зрозуміло, що таке значення k існує, адже масив нормований.

   Можливі два випадки.

   Перший. Ця сума $(c_1+c_2+\ldots+c_k)+c_{N-1}$ дорівнює нулю.

   Щоб обнулити c_{N-1}, ми вже використали k рейсів, і потрібно ще $(n-2-k)$ рейсів, щоб перетворити на нуль число c_N.

   Таким чином, за $k+(N-2-k)=N-2$ рейсів (а це менше ніж $N-1$) масив c обнулився.

Другий. Ця сума $(c_1 + c_2 + \ldots + c_k) + c_{N-1}$ — додатне число.

Зрозуміло, що тоді останній доданок ck може бути розкладений на суму двох додатних доданків $c*$ і $c**$ таких, що:

$(c_1 + c_2 + \ldots + c_k + c*) + c_{N-1} = 0$ і $(c** + c_{k+1} + \ldots + c_{N-2}) + c_N = 0$.

У перших дужках сума містить k додатних доданків, у других — $(N-1-k)$ додатних доданків. Кожна з рівностей містить лише один від'ємний доданок. А випадок з одним від'ємним доданком нами розглянутий (нагадаємо: для нього вистачить стільки рейсів, скільки є додатних доданків).

Тобто для обнулення масиву c із двома від'ємними доданками потрібно використати $k + (N-1-k) = N-1$ рейсів.

Теорему для цього випадку доведено.

Звернімо увагу на те, що, «погасивши» c_{N-1}, ми звели задачу до випадку, коли в масиві лише один елемент від'ємний, а решта — додатні.

Нескладно зрозуміти, що далі теорема легко доводиться за індукцією. Справді, якщо три останні елементи масиву від'ємні, а решта — додатні, тоді, «погасивши» c_{N-2}, легко звести задачу до випадку, коли в масиві лише два елементи від'ємні тощо.

Введемо нове означення.

Назвемо нормований масив неспрощуваним, якщо з нього не можна вибрати жодного нормованого підмасиву з меншою кількістю елементів (зрозуміло, що спрощуваним слід назвати нормований масив, для якого існує нормований підмасив з меншою кількістю елементів).

Цікавим є наслідок з Теореми 2, а фактично нова, але зрозуміла

Теорема 3

Якщо нормований масив з N елементів неспрощуваний, то для його обнулення потрібно рівно $(N-1)$ рейсів.

Справді, в цьому масиві не може трапитися випадок перший, інакше масив був би спрощуваним.

І нарешті, ключовою для вказаної задачі є важливий і легко зрозумілий наслідок з теореми 3 —

Теорема 4

Якщо нормований масив з N елементів розпадається на M нормованих неспрощуваних підмасивів, то для обнулення такого масиву потрібно рівно $(N-M)$ рейсів.

Ця важлива теорема 4 фактично відповідає на запитання: яку найменшу кількість рейсів потрібно застосувати, щоб вирівняти кількості книжок (тобто обнулити масив c[1:N])?

У ній неявно записаний і алгоритм для розв'язування задачі.

Ось як він може виглядати:

1. Нормуємо масив a (створюємо новий масив c).
2. Відкидаємо з масиву c всі нулі. Нехай NN — кількість елементів нового масиву (без нулів), а індекси нулів масиву c записані в множині NZ.
3. Серед чисел c[1:N] знаходимо нормовану пару (пару таких, що в сумі дають нуль). Зрозуміло, що їх індекси не входять у NZ. Нехай таких пар знайдено $K2$.
4. Відшукуємо серед елементів, що не потрапили в пари і мають індекси, що не належать NZ, трійки нормованих. Нехай таких трійок $K3$.

Продовжуємо цей процес для нормованих четвірок, п'ятірок тощо.

5. В результаті $NN = K2 + K3 + \ldots + Ks$ (s — деяке натуральне).

Відкинувши всі нулі серед $K2, K3, \ldots, Ks$ (нехай таких нулів Z), за теоремою 4 легко зробити висновок: найменша кількість рейсів дорівнює $NN - (s - Z - 1)$.

Друга частина задачі (звідки, куди і скільки книжок перевозиться в кожному рейсі).

У першій частині ми вже розбили масив c на неспрощувані нормовані підмасиви. Для кожного з них (у програмі мовою Паскаль множину індексів його елементів позначено $s(h)$, де h — номер неспрощуваного підмасиву) легко обчислити:

SP — множину індексів додатних елементів, SM — множину індексів від'ємних елементів, NP — кількість елементів у підмасиві (ми домовилися, що в підмасиві немає нулів);

u — найменший з індексів додатних елементів;

v — найменший з індексів від'ємних елементів.

Ми вже знаємо, що потрібно $(NP - 1)$ рейсів, щоб обнулити цей підмасив. Обчислимо для кожного з рейсів, звідки, куди і скільки книжок слід перевезти, щоб обнулити підмасив.

Алгоритм цих перевезень простий і виглядатиме з використанням апарату множин таким чином:

(Нагадаємо, що перед першим рейсом t:=0)

```
поки (SP+SM<>[ ]) (поки не вирівняли кількість кни-
жок)
пц
      t:=t+1 (черговий рейс)
      x[t]:=u;y[t]:=v
```

(у масиви x і y запишемо для кожного рейсу номери, відповідні номерам школи, звідки перевозять книжки та куди перевозять)

```
якщо c[u ] +c[v] >= 0
то (в u-й школі вистачить зайвих книжок, щоб обнулити c[u])
     c[u]:=c[u]+c[v]; c[v]:=0; SM:=SM-[v];
     якщо c[u]=0
     то SP:=SP-[u]
     все (це можливо лише в останньому рейсі для цього підмасиву)
     якщо SM<>[ ]
     то v:=v+1
     все (якщо ще є від'ємні елементи в підмасиві)
   інакше (перевезти з u-ої школи в школу з номером v усі зайві книжки)
     c[v]:=c[u]+c[v];
     c[u]:=0;
     SP:=SP-[u];
     u:=u+1
  все
кц
```

Відповідна програма мовою Паскаль, яка розв'язує обидві частини задачі, додається нижче. Всі дані в ній вводяться з клавіатури.

```
{-------------Підручники-----------------------}
uses crt;
var n,nn,sum,I,k,b,h,m,q:Integer;
  a,c,p: array [ 1..16] of Integer;
  sn,st:set of Byte;
  s: array[1..16] of set of Byte;
{ Крок уперед -}
procedure step:
begin inc(p);p[q]:=k;st:=st+[k];sum:=sum+c[k] end;
{------------Крок назад-----------------------------}
procedure back;
begin k:=p[q];sum:=sum-c[k];st:=st-[k];dec(q) end;
{-----------Перша частина задачі------------------}
procedure part1;
begin
  clrscr;
  write('Шкіл:');
  read(n);
  sum:=0;
  write ('Завезено книжок у них:');
```

```
  {значення b}
  for i:=1 to n do
  begin
    read(a[i]);
    sum:=sum+a[i]
  end;
  b:=sum div n;
  nn:=0;
  sn:=[ ];
  for i:=1 to n do {ненулі масиву c}
  begin
    c[i]:=a[i]-b;
    if c[i]<>0 then begin
                      sn:=sn+[i];inc(nn)
                    end
  end;
  h:=0; {Розбивання sn на нормовані неспрощувані під
множини}
  for m:=2 to nn div 2 do
  begin
    for i:=1 to nn-m do
    begin
      if i in sn {старт}
      then begin
            q:=1;p[1]:=i;st:=[i];sum:=c[i];k:=i+1;
            while q>0 do
              begin
                if q<m
                then begin
                      if k<=nn
                      then begin
                            if k in sn then step
                            {крок вперед}
                           end
                      else back {крок назад}
                     end
                else begin
                      if sum=0
                      {знайдено підмножину st з m еле
              ментів}
                      then begin
                            sn:=sn-st;
                            inc(h);
                           end
```

```
                    else back {крок назад}
                  end;
               inc(k)
            end
         end
    end
end;
  if sn<>[ ]
  then begin inc(h);s[h]:=sn end;
  write('Рейсів: ',nn-h) end;

  {-------Друга частина задачі-----------------------}
procedure part2;
begin t:=0;
  {t - кількість потрібних рейсів для вирівнювання
кількостей книжок}
  for k:=1 to h do
  begin sp:=[ ];sm:=[ ]; end;
{визначення множин SP, SM та значень u і v для
кожного неспрощуваного підмасиву з номером k }
for i:=1 to nn do
if i in s(k)
then begin
      if c[i]>0
      then begin
            if sp=[ ] then u:=i;
            sp:=sp+[i]
           end
      else if c[i]<0
           then begin
                 if sm=[ ]
                 then v:=i; sm:=sm-[i]
                end;
    end;
{Алгоритм формування рейсів для неспрощуваного під-
масиву}
while (SP+SM<>[ ]) do {поки не вирівняли кількість
книжок}
begin
  inc(t); x[t]:=u;y[t]:=v;
  {у масиви x і y запишемо для кожного рейсу номери,
відповідні номерам школи, звідки перевозять книжки та
куди перевозять}
```

```
    if c[u]+c[v]>=0
    then begin {в u-й школі вистачить зайвих книжок,
          щоб обнулити c[v]}
           c[u]:=c[u]+c[v]; c[v]:=0; SM:=SM-[v];
            if c[u]=0 then SP:=SP-[u]; {це можливо
           лише в останньому рейсі цього підмасиву}
           if SM<>[ ] then v:=fm(v);
           {якщо ще є від'ємні елементи в підмасиві}
        end
    else begin
          {перевезти з u-оїї школи в школу з номером
     v всі зай-ві книжки}
           c[v]:=c[u]+c[v];c[u]:=0;
           SP:=SP-[u];u:=fp(u)
        end;
     end;
   end;
end:
{----------Основна програма-------------------------}
begin
  part1;part2;
  readln;readln
end.
```

ДОДАТОК 1

АЛГОРИТМИ НА ГРАФАХ

Визначення графа

Нехай задано P вершин, при цьому деякі з них сполучені лініями (ребрами).

Множину з P вершин і ребер, що їх сполучають, називають графом. Визначення графа містить, крім того, вимогу «без петель», тобто немає жодного ребра з A в A ($A=1\ldots P$). Граф не має подвійних ребер. Мережа шосейних або залізничних доріг є графом.

Граф з N вершин можна задати декількома способами.

Перший. Вказати число ребер M у цьому графі. Потім у два масиви U і V (завдовжки M кожен) записати номери вершин, що є кінцями ребер.

Другий. Матрицю D розміром $N X N$ заповнити нулями й одиницями за таким правилом: $D[A,B]=1$ лише тоді, коли вершини A і B сполучені ребром. В інші комірки матриці вписати нулі.

Графи можуть бути зв'язними (від кожної вершини можна дійти до кожної іншої, використовуючи ребра як дороги) і незв'язними.

Якщо в графі можна вибрати такі M вершин $W_1, W_2, \ldots, W_m$, що кожна наступна сполучена ребром з попередньою і остання — з першою, то кажуть, що в графі є цикл. Вважають, як правило, що в циклі не менше ніж три вершини.

Послідовність різних сусідніх вершин графа називають маршрутом графа.

Зв'язний граф без циклів називають деревом. (Рисунок його насправді нагадує дерево з коренем і гілками.) У вигляді дерев прокладають водопровідні мережі, мережі кабельного телебачення, електропроводку, каналізаційні, газові, радіо та інші мережі.

У вигляді дерев зображують родословну сім'ї по лінії батька чи матері. Коренем його буде найдальший родич (прабатько), далі підуть його сини та дочки, їх сини та дочки тощо.

Граф типу дерево має цікаву властивість: кожну його вершину можна вважати коренем.

Легко обчислити, що в дереві з P вершин кількість ребер $(P-1)$.

Ще одна важлива властивість дерева: між кожними двома вершинами дерева існує лише один маршрут.

Іноді розглядають орієнтовані графи (орграфи), у яких $D[A,B]$ не обов'язково дорівнює $D[B,A]$. (Наприклад, з Ялти в Сочі є поїзд, а із Сочі в Ялту — ні).

Всяка родословна фактично є орграфом: $D[A,B]=1$ означає, що A — батько для B і, зрозуміло, $D[B,A]=0$. Дерево каталогів диска — теж орграф.

Задачі про графи

Для задання графа з P вершин найчастіше використовують матрицю сполучень розміром $P\times P$, у якій P^2 параметрів $D[I,K]$ (I, $K=1\ldots P$), що набувають значень 1 (якщо вершини I і K сполучені ребром) або 0 (якщо ні). Задавання звичайного графа (не орграфа) матрицею містить зайві дані: зрозуміло, що коли вершини A і B сполучені ребром, то $D[A,B]=D[B,A]=1$. Крім того, для вершин A і B, між якими немає ребра, $D[A,B]=D[B,A]=0$. Тому у звичайному графі матриця ребер симетрична відносно головної діагоналі, яка зазвичай містить нулі (граф не має «петель»!).

Якщо задати граф з P вершин і M ребер двома масивами — індексами кінців ребер, — то знадобиться $2*M$ параметрів.

Але граф типу дерево можна задати масивом C довжини P, тобто використати лише P параметрів. Зрозуміло, що при $P>2$ це економніше навіть за другий спосіб, для якого потрібно $2*(P-1)$ параметрів. Як це зробити?

Домовимось одну з вершин (не важливо, яку саме, — наприклад, з індексом A) вважати коренем дерева й прабатьком усіх вершин, зокрема вершин, що виходять з A. Покладемо C[A]:=0, що означатиме: «у вершини A немає батьків». Вважаємо S:=[A] — множина вершин, для яких уже відомі їх батьки.

Перевіримо всі вершини, що не належать S, і для кожної з них з індексом K, що має ребро до вершини A, вважатимемо, що

```
C[K]=A;
для K від 1 до P
пц
  якщо D[K,A]=1 і (K не належить S)
  то C[K]:=A;S:=S+[K]
  все
кц
```

Нехай вже відомо, хто є батьком вершини I. Тоді подібну операцію доведеться виконати для кожної з вершин K, які не належать S, проте з'єднані ребром із вершиною I:

```
якщо (I належить S) і (K не належить S) і D[I,K]=1
то C[K]:=I;S:=S+[K]
все
```

Зрозуміло, що такий пошук доведеться проводити доти, доки у множину S не потраплять усі вершини.

Матимемо алгоритм:

```
S:=[1];C[1]:=0;
Поки S<>[1..P]
пц для I від 1 до P
   пц для K від 1 до P
      пц якщо (I in S) і not(K in S)
      і D[I,K]=1
        то C[K]:=I;S:=S+[K]
        все
      кц
   кц
кц
```

Його трудомісткість не перевищує P^3.

Тобто, ми розв'язали задачу № 0.

0) Дерево задано (1/0)-матрицею сполучень. Записати його у вигляді масиву батьків.

Спробуйте розв'язати прості (або не занадто складні) задачі про графи-дерева.

1) Дерево задано масивом батьків. Записати його у вигляді матриці сполучень, а також двома масивами кінців ребер.
2) Дерево з коренем задано масивом батьків. Поміняти в ньому корінь на B і записати новий масив батьків.
3) У дереві відомі довжини усіх ребер. Визначити довжину маршрута від A до B.
4) У дереві відомі довжини усіх ребер. Знайти дві найбільш віддалені вершини A і B.
5) У дереві відомі «маси» усіх вершин (наприклад, розміри файлів у директоріях диска). Визначити суму «мас» на маршруті від A до B.
6) Вибрати в графі вершину M таку, щоб відстань від неї до найбільш віддаленої вершини була найменшою.
7) Два графи з однаковими масивами батьків вважатимемо рівними. Перевірити, чи є рівними два графи з масивами батьків $Q1$ і $Q2$.
8) Два графи назвемо ізоморфними, якщо в одному з них можна перенумерувати вершини так, що він збігатиметься з іншим.

Перевірити, чи є ізоморфними два графи з масивами батьків $Q1$ і $Q2$.

9) На площині задано P будинків своїми координатами. Знайти найкоротшу мережу для проведення радіо (або кабельного ТВ) цих будинків (мережу у формі дерева).

Метод віток і меж

Є декілька стандартних задач про графи, що стали вже класичними. До них належить і задача про проведення кабельного телебачення, яка розв'язується так званим «дерев'яним» алгоритмом (її іноді формулюють так: знайти мінімальне остовне дерево для заданих P вершин). Алгоритм для подібних задач відомий як алгоритм Прима-Краскала.

До класичних належать і три задачі.

У даному графі з N вершин (не обов'язково — дереві) знайти:

а) всі маршрути з вершини A;

б) найкоротший (за кількістю вершин) маршрут від A до B;

в) найкоротший (за довжиною) маршрут від A до B (довжини усіх ребер відомі).

Ці задачі (і деякі інші) називають перебірними і розв'язують найчастіше з використанням алгоритма, який називають методом віток і меж. Інша назва — бектрекінг.

Детальніше роз'яснимо його суть на прикладі першої задачі (всі маршрути з вершини A).

В заданому графі з N вершин знайти всі маршрути з вершини з номером A.

Вважатимемо, що граф задано матрицею ребер D. Вершини шуканого маршруту будемо записувати в масив P (його довжина менша від N — поміркуйте, чому?). Зрозуміло, що:

```
Q:=1; {уже відомо Q вершин на маршруті}
P[1]:=A; {перша з вершин маршрута}
S:=[A]; {множина вже використаних вершин}
```

Почнемо шукати серед вершин з індексами $K=1...N$ таку, яка задовольняє двом умовам:

1) $1K$ не належить S;

2) $1K$ сполучена ребром з $P[Q]$.

(Видно, що задача перебірна! І перебирання слід почати з вершини $K=1$). Тому до трьох записаних команд додамо ще одну:

```
K:=1; {початок перебирання}
```

Таких вершин K може існувати декілька (якщо вони взагалі є). Кожна з них може бути додана у маршрут:

```
Q:=Q+1; P[Q]:=K; S:=S+[K].
```

Таку процедуру з трьох команд називають кроком (англійською — step).

Може трапитися, що при деякому Q (для кожного графа — обов'язково!) таких вершин немає (решта вершин або не мають ребер до $P[Q]$, або вже є на маршруті!). Це означає, що ми на кінці маршруту (в тупику). Якщо перебирати значення K у вигляді циклу поки $K \le N$, то ми прийдемо до значення $K = N+1$. Зрозуміло, що в цьому випадку слід ввести весь маршрут $P[1...Q]$ і шукати інші маршрути. Як вийти на тупика? Це роблять таким чином (знову процедура з трьох дуже схожих зі step команд, яку називають back — крок назад):

1) запам'ятати, де ми знаходимось: K:=P[Q];
2) вилучити K з множини S: S:=S-[K];
3) відступити крок назад: Q:=Q-1;

Зробивши крок назад, слід почати перебирання з номера на 1 більшого, ніж K (тобто K:=K+1). Зрозуміло, відступити крок назад можна лише за умови, що нове значення $Q \le 1$ ($Q > 0$). Тому алгоритм бектрекінгу для цієї задачі зручно записати таким чином:

Алгоритм МАРШРУТИ

```
Q:=1; P[Q]:=A; S:=[A]; K:=1
поки Q>0
пц
  якщо K<=N
  то якщо D[P[Q],K]=1  {вершини з'єднані ребром}
       і not(K in S)  {K немає на маршруті}
     то STEP; K:=0{щоб почати перебирання з K=1}
     все
    інакше ввести P[1..Q]; BACK
    все
    K:=K+1
кц
```

Зауважимо, що в бектрекінгу лише один цикл ПОКИ. Бектрекінг відсікає в перебиранні ті значення K, що не відповідають хоча б одній з двох умов (відсікає зайві гілки), обмежуючи діапазон для Q і K значеннями $1...N$.

За допомогою бектрекінгу легко розв'язати дві прості задачі:

а) знайти всі перестановки чисел $1...N$;
б) ввести всі випадки призначення N претендентів на M однакових посад ($N \ge M$).

Наведемо алгоритм для розв'язування обох задач. Задача про перебирання перестановок — це задача про призначення на N різних посад (тут важливий порядок!), тому слід вважати, що посад стільки, скільки претендентів: $M = N$, а в другій задачі про призначення команди K:=0 можна опустити (вилучити), скоротивши число перебирань (склад претендентів не зміниться, якщо змінити їх порядок: усі посади рівнозначні, тому усі масиви-маршрути йтимуть у порядку збільшення номерів претендентів). Зверніть увагу, що BACK зустрічається в алгоритмі двічі.

Алгоритм ПРИЗНАЧЕННЯ

```
Q:=1;P[Q]:=A;S:=[A];K:=1
поки Q>0
пц якщо Q<M
  то якщо K<=N
    то якщо not(K in S) {K немає на маршруті}
       то STEP;K:=0
       все
     інакше BACK
     все
    інакше ввести P[1..Q];BACK
    все
    K:=K+1
кц
```

Алгоритм Флойда-Уоршела

Задачу про довжину найкоротшого маршруту від A до B можна теж розв'язати за допомогою бектрекінгу. Але дуже просто й красиво виглядає розв'язання цієї задачі в алгоритмі Флойда-Уоршела.

Далі опишемо цей алгоритм детальніше.

Для того щоб використати його, потрібно відстані (довжини ребер) записати в матрицю R, схожу на $D[1...N,1...N]$, причому в комірках, де значення дорівнює нулю (немає ребер), вважати, що довжина ребра є ∞ (нескінченність). Практично нескінченність заміняють деяким великим числом, скажімо 10 000.

Тому складемо матрицю R ребер, замінивши одиниці в матриці D на відомі довжини ребер, а нулі в D — на 10 000. Нову матрицю скопіюємо в RR (щоб не псувати матрицю R).

Алгоритм виглядає майже граціозно:

```
для I від 1 до N
пц для J від 1 до N
  пц для K від 1 до N
     нц якщо RR[I,J]+RR[J,K]<RR[I,K]
       то RR[I,K]:= RR[I,J]+RR[J,K]
          RR[K,I]:= RR[I,J]+RR[J,K]
       все
    кц
  кц
кц
```

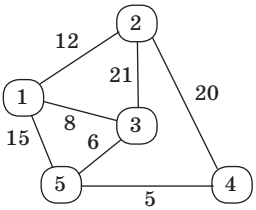
Словами його іноді формулюють так: «якщо в обхід ближче, ніж навпростець, то йди в обхід».

Нова матриця RR після виконання циклів ДЛЯ покаже відстані між кожними двома вершинами. Зауважимо, що алгоритм Флойда-Уоршела дає лише значення $RR[A,B]$ — найкоротшої відстані між кожними двома вершинами A, $B=1...N$, тобто довжину кожного найкоротшого маршруту, але не виводить самого маршруту.

Алгоритм Дейкстри

Той самий результат (найкоротшу довжину) покаже й алгоритм Дейкстри, зате в ньому буде визначено і власне маршрут (послідовність вершин), щоправда, лише від стартової вершини A до кожної з інших вершин.

Детальніше опишемо алгоритм Дейкстри.



Задано граф з N вершин, номер стартової вершини A і відома матриця ребер R (з урахуванням ∞ для пар вершин, між якими немає ребер). Знайти найкоротший маршрут з A до кожної з вершин і його довжину.

Проілюструємо алгоритм на прикладі конкретного графа (див. рисунок) з $N=5$ вершин. Нехай потрібно знайти довжини всіх маршрутів від вершини $A=2$ до вершин 1... 5. Виконаємо декілька операцій. Спочатку складаємо матрицю довжин ребер, користуючись рисунком.

	1	2	3	4	5
1	0	12	8	∞	15
2	12	0	21	20	∞
3	8	21	0	∞	6
4	∞	20	∞	0	5
5	15	∞	6	5	0

Заведемо три масиви X, Y, Z довжиною $N=5$ і складемо початкову таблицю (див. таблицю ниже) за такими правилами:

а) у перший масив X записуємо нулі у всі комірки, крім $X[2]=1$. Одиницею помічатимемо переглянуті вершини;

б) у другий масив впишемо довжини ребер від вершини 2 з матриці (або рисунка), тобто перенесемо другий рядок матриці;

в) у третій масив Z запишемо всюди 2 (номер стартової вершини), крім комірки 2, куди впишемо 0.

Індекс	**1**	**2**	**3**	**4**	**5**
X	0	1	0	0	0
Y	12	0	21	20	∞
Z	2	0	2	2	2

Тепер $N-1$ разів виконаємо процедуру ПОРІВНЯННЯ.

1) Знайдемо мінімум у масиві Y серед невідмічених вершин (серед тих $K=1\ldots N$, де $X_K=0$).
2) Нехай цей мінімум значення Y досягає на індексі W.
3) Міняємо на одиницю значення X_W.
4) Виконуємо цикл.

```
для K від 1 до N
пц якщо YK>YW+R[W,K] (*)
  то YK:=YW+R[W,K]; ZK:=W
  все
кц
```

Після першого виконання ПОРІВНЯННЯ матимемо:

Мінімум у масиві Y (серед нулів для X) є 12 і досягається на W=1: Y1=12.

Замінимо на 1 значення X_1.

Індекс	**1**	**2**	**3**	**4**	**5**
X	1	1	0	0	0
Y	12	0	21	20	∞
Z	2	0	2	2	2

Тепер переглянемо всі значення $K=1\ldots N$, щоб знайти ті, для яких виконується співвідношення $Y_K>12+R[1,K]$.

Співвідношення $Y_K>12+R[1,K]$ (в алгоритмі воно позначене *) виконується лише двічі: для $K=3$ $(21>12+8)$ і для $K=5$ $(\infty>12+15)$. Тому Y_3 замінимо на 20, а Z_3 на 1. Далі Y_5 замінимо

на 27, а Z_5 на 1. Ось якого вигляду набуде таблиця після першого кроку:

Індекс	1	2	3	4	5
X	1	1	0	0	0
Y	12	0	20	20	27
Z	2	0	1	2	1

Друге виконання процедури ПОРІВНЯННЯ.

Розглянемо таблицю після першого кроку.

Мінімум Y серед нулів для X буде при $W = 3$ і $Y_3 = 20$.

Замінимо на 1 значення X_3.

Співвідношення $Y_K > 20 + R[3, K]$ виконується для $K = 5 (27 > 20 + 6)$, тому слід замінити Y_5 на 26, а Z_5 на 3.

Ось якого вигляду набудуть три наших масиви:

Індекс	1	2	3	4	5
X	1	1	1	0	0
Y	12	0	20	20	26
Z	2	0	1	2	3

Ще два нулі в масиві X. Стільки разів ще доведеться повторити процедуру ПОРІВНЯННЯ.

Третє виконання процедури ПОРІВНЯННЯ.

Мінімум у середньому рядку серед невідмічених вершин при $W = 4,\ Y_4 = 20$. Замінимо на 1 значення X_4.

Індекс	1	2	3	4	5
X	1	1	1	1	0
Y	12	0	20	20	26
Z	2	0	1	2	3

Перевіримо, для яких значень K виконується нерівність $Y_K > 20 + R[4, K]$. Лише для $K = 5$ $(26 > 20 + 5)$.

Тому слід замінити Y_5 на 25, а Z_5 на 4.

Ось як зміниться таблиця:

Індекс	1	2	3	4	5
X	1	1	1	1	0
Y	12	0	20	20	25
Z	2	0	1	2	4

Останнє виконання процедури ПОРІВНЯННЯ.

$W = 5$, $Y_5 = 25$. Замінимо на 1 останній нуль у першому рядку.

Індекс	1	2	3	4	5
X	1	1	1	1	1
Y	12	0	20	20	25
Z	2	0	1	2	4

Перевіряємо для кожного $K = 1 \ldots N$, чи виконується співвідношення $Y_K > 20 + R[5, K]$. Таких значень немає. Більше в таблиці міняти нема чого. Вона остаточна.

А ось і відповіді: в середньому рядку записані найкоротші шляхи від вершини 2 до вершин 1... 5.

А де ж відповідні маршрути?

Доведеться виконати ще одну процедуру:

```
для K від 1 до N
пц вивести('вершина ',K)
     C:=ZK
       повторити
         вивести C
         C:=ZC
     до C=0
кц
```

Вона і виведе потрібні маршрути з усіх вершин (у зворотному порядку).

Існують інші задачі на графи, для багатьох із них вже складені алгоритми.

Ось одна із задач.

З'ясувати, чи є заданий граф планарним (усі його вершини розміщені на площині і жодні два ребра не перетинаються у внутрішніх точках). Алгоритм для цієї задачі склали Л. Понтрягін та С. Куратовський, які встановили критерій непланарності графа:

граф непланарний лише тоді, коли серед його підграфів є або повний граф з 5 вершин, або дводольний граф 3×3.

Роз'яснимо ці поняття:

- Підграфом графа називають граф, у якому вершин і ребер не більше, ніж у заданому (тобто деякі ребра і вершини «стерли» — вилучили).
- Повним називають граф, у якому кожна вершина сполучена ребром із кожною з інших вершин.
- Якщо множину вершин графа можно розбити на дві непорожні

неперетинні підмножини $M1$ і $M2$ так, що кожна з $P1$ вершин множини $M1$ сполучена ребром з кожною з $P2$ вершин множини $M2$, то кажуть, що задано дводольний («дві долі», тобто дві частини) граф $P1 \times P2$.

Спробуйте скласти алгоритми для кожної з наведених нижче досить складних задач:

а) чи містить заданий граф повний підграф з T вершин?

б) чи містить заданий граф дводольний граф $M \times P$?

в) відомо, які з B дівчат подобаються кожному з A хлопців. Яким може бути найбільша можлива кількість весіллів? (Задача про найбільше паросполучення.) Повне розв'язання цієї задачі про дводольний граф $A \times B$ можна знайти в цікавій книжці А. Бондарева і В. Рублинецького «Основы программирования» (Харьков : ФОЛИО, 1997).

г) На одній із Всеукраїнських олімпіад було запропоновано задачу:

Місто має декілька кварталів (вершин), сполучених вулицями (ребрами). У якому місці (на ребрі або у вершині) слід розмістити пункт поліції, щоб відстань від нього до найдальшого з кварталів була найменшою? (Ця задача належить до так званих мінімаксних задач: серед максимумів відстаней знайти мінімум.) Звертаємо увагу на те, що розв'язок задачі, наведений у згадуваній книжці А. Бондарева і В. Рублинецького (алгоритм Хакімі), містить суттєві помилки.

д) Ось ще схожа задача про графи з Всеукраїнських олімпіад:

Знайти остовне дерево в заданому графі (тобто треба «стерти» найбільше ребер у заданому графі, залишивши лише дерево так, щоб сума довжин його ребер була найменшою).

За допомогою графів розв'язують задачі про послідовність робіт на виробництві, про пошук виходу зі складних лабіринтів і багато інших.

ДОДАТОК 2

ХТО НЕ ЗМІГ, ТОЙ ПРОГРАВ…

Саме на такому принципі побудовані ігри двох учасників, які нерідко можна зустріти серед задач, що пропонуються на олімпіадах різного рівня з інформатики. Далі в умові таких задач пропонується оцінити (Виграш / Програш) кожну задану ситуацію, записану в деякому текстовому файлі, і зробити у випадку Виграшу хоча б один ВХ (виграшний хід). Іноді зустрічаються задачі, де потрібно перелічити всі можливі ВХ. Якщо ВХ не існує, вивести повідомлення «Програш» тощо.

Ці замітки адресовані в першу чергу тим учням, які беруть участь в олімпіадах. Тому в них ми спиратимемось на добре розвинену їх інтуїцію, не акцентуючи уваги на доведеннях. Для прискіпливого ж читача, обізнаного в тонкощах інформатики, ми іноді вимушені супроводжувати доведеннями досить зрозумілі учням, майже очевидні факти. Тому стиль цих заміток коливається від наукового до популярного, а найчастіше має дидактичний характер.

А тепер — про гру за згаданим у заголовку принципом.

Взагалі, кожна гра такого типу — це скінченна послідовність ситуацій, у яких перебувають об'єкти гри (гральні кості, карти, кості доміно, купки камінців тощо) або групи об'єктів. Ми розглянемо деякі з таких ігор, а саме ті, у яких можливий хід відбувається над одним об'єктом гри, і він не обмежений станами, у яких перебувають інші об'єкти.

Наведемо приклади таких ігор. У кожній з них умова починається однаково: «Грають двоє, ходять по черзі».

І закінчується теж однаково: «Той, хто не зміг цього зробити, програв».

Гра 1

Є декілька купок (не більше ніж 100 000) камінців, у найбільшій з яких до 100 камінців. За один хід можна одну з купок поділити на дві нерівні.

Гра 2

Робітники фабрики, що виготовляє доміно, маючи вдосталь кожної з 28 костей, грають в обідню перерву в таку гру. На початку гри вони навмання відкривають на столі декілька костей (не більше 10 000). За один хід можна зняти одну з костей (x,z), замінивши її іншою (x,z), якщо $z > y$.

Враховуючи задані межі для кількості купок, ці задачі взагалі здаються у граничних випадках такими, що неможливо розв'язати на ЕОМ (не вистачить пам'яті).

Спробуємо виявити деякі спільні властивості, притаманні кожній з таких ігор, а також вказати майже «універсальний» шлях для розв'язування таких задач.

Домовимося спочатку:

1^0. Якщо ситуація S' отримана із ситуації S через один хід, називатимемо S' похідною для S і позначатимемо цей факт таким чином: $S \to S'$.

1^0а. У кожній грі повинна бути одна, іноді — декілька нерухомих ситуацій, тобто таких, що не мають жодної похідної, інакше гра не мала б кінця. (У задачі про камінці — коли в кожній з купок один або два камінці, у задачі про доміно — коли всі кості 6 : 6.)

1^0б. У кожній грі повинні бути ситуації, що мають більше від однієї похідної, інакше в грі не було б варіантів, тобто вона була б нецікавою, а тому і нескладною.

2^0. Вважатимемо, що грають двоє суперників, які знають «секрет» гри (якого ми, на жаль, ще не знаємо, і якому сподіваємось навчитися до кінця цих заміток!) і не прощають один одному помилок. Тобто в кожній виграшній ситуації (далі В), вміють зробити ВХ, тобто знайти таку похідну S' для S, яка є програшною (далі П).

3^0. Враховуючи 10 та 20, вважатимемо, що у програшній ситуації гра відбуватиметься за сценарієм:

$\text{П} \to \text{В} \to \text{П} \to \text{В} \to \ldots \to \text{В} \to \text{П}$ (всього непарна кількість ситуацій). У цьому ланцюжку на парних місцях стоять В (виграшні) ситуації, на непарних — П (програшні).

У виграшній ситуації все навпаки:

$\text{В} \to \text{П} \to \text{В} \to \text{П} \ldots \to \text{В} \to \text{П}$ (всього парна кількість ситуацій), і на кожному парному місці — П, а на непарному — В. Тому випливає висновок: взагалі не розглядати ходів типу $\text{В} \to \text{В}$ (оскільки наші гравці таких помилок не роблять!).

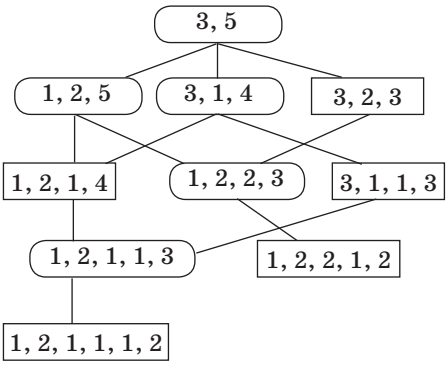
Легко сформулювати два постулати такої гри:

1) кожна похідна програшної ситуації є виграшною ($\text{П} \to \text{В}$) і навпаки: якщо всі похідні для S виграшні, то $S = \text{П}$ (програшна);
2) кожна виграшна ситуація має принаймні одну програшну похідну, і навпаки: якщо деяка похідна S' для S є програшною, то $S = \text{В}$ (виграшна).

А тому, щоб пересвідчитись, що $S = \text{В}$, достатньо знайти S' таку, що $S \to S' = \text{П}$.

І вже, здається, значно складніше довести, що $S = \text{П}$. Для цього потрібно переконатись у тому, що кожна похідна $S' = \text{В}$, тобто перебрати всі похідні. Зауважимо, що кожна з похідних може мати ще декілька похідних. Тому, якщо піти цим шляхом для встановлення того, якою є ситуація S — програшною, чи виграшною, — доведеться розглядати граф гри.

Так, для двох купок камінців 3,5 він матиме такий вигляд:



Для зручності читання всі П — у прямокутниках, всі В — в овалах, нерухомі П підкреслені.

Як визначити, коли має місце П, а коли В?

Йдучи з кінця, одержавши нерухомі 1, 1, 2, 2, 2 та 1, 1, 1, 1, 2, 2, робимо висновок, що $1,2,1,1,3 = \text{В}$ та $1,2,2,3 = \text{В}$.

А тому $3,2,3 = \text{П}, 1,2,1,4 = \text{П}$, $3,1,1,3 = \text{П}$ як такі, що мають лише одну похідну, яка яка є В. А тому $1,2,5 = \text{В}$ та $3,1,4 = \text{В}$ як такі, що мають серед похідних П. Тому зрозуміло, що $3,5 = \text{В}$ і, згідно з домовленістю 30, гра відбуватиметься фактично таким чином:

$3,5 \to 3,2,3 \to 1,2,2,3 \to 1,2,2,1,2$ (Виграш Першого).

Побудова дерева вимагає для кожної проміжної (а не лише початкової) ситуації шукати всі її похідні. А це неекономно. Легко уявити собі, як виглядатиме граф гри, коли купок камінців декілька сотень. Тоді просто може не вистачити пам'яті для навіть тимчасового запам'ятовування проміжних ситуацій, одержаних під час перебирання. Задача, у якій ідеться, наприклад, про 100 000 купок, тоді взагалі залишиться нерозв'язаною.

Нижче буде показано, що можна обійтись і без проміжних варіантів.

Розглянемо ситуацію, яку можна отримати з S приєднанням до неї такої ж ситуації і яку ми позначатимо S,S.

Теорема 1

$S,S = \text{П}$. Ситуація S,S завжди програшна.

Доведення проілюструємо на прикладі задачі про купки камінців.

Якщо S нерухома, то і S,S теж нерухома, тобто програшна.

Нехай Перший зробив хід $S \to S'$. Тоді у Другого є такий самий хід $S \to S'$. Через два ходи матимемо ситуацію S',S', яка складається з двох однакових. Але кількість купок в останній збільшилася на дві, хоч загальна кількість камінців не змінилася. Тому, якщо Другий продовжуватиме «копіювати» ходи Першого і далі, ми через парну кількість ходів прийдемо до ситуації So,So, яка є нерухомою і в якій черга ходити Першому. Тому він програв.

Означення 1

Назвемо дві ситуації $S1$ і $S2$ еквівалентними ($S1 \sim S2$), якщо ситуація $S1,S2 = \Pi$.

Легко довести властивості еквівалентності:

- Рефлексивність. Кожна S еквівалентна сама собі: $S \sim S$.
- Транзитивність. Дві ситуації, еквівалентні третій, еквівалентні одна одній: якщо $(S1 \sim S)$ і $(S2 \sim S)$, то ($S1 \sim S2$).
- Усі програшні ситуації еквівалентні одна одній.
- Ситуацію, у якій немає жодного камінця (вона не може зустрітися у грі), можна вважати програшною (адже той, чия черга ходити, не зможе зробити жодного ходу!).
- Якщо з довільної ситуації S вилучити всі програшні, отримаємо еквівалентну S ситуацію.
- Якщо до довільної ситуації S приєднати довільну кількість програшних, отримаємо еквівалентну S ситуацію.
- Якщо з довільної ситуації S вилучити всі пари однакових ситуацій, одержимо еквівалентну S ситуацію.
- Якщо до довільної ситуації S приєднати пари однакових ситуацій, одержимо еквівалентну S ситуацію.

Це вже дещо спрощує задачу про камінці, хоч і не радикально. Скажімо, для початкової ситуації 1, 3, 3, 3, 6, 2, 4, 13 матимемо:

$1,3,3,3,6,2,4,1,4 \sim 3,6,13$ (викреслені програші 1, 2, 4 та пара рівних 3, 3), щоправда, потрібно ще довести, що $4 = \Pi$.

Відзначимо дуже важливу властивість похідної.

Теорема 2

Якщо $S \to S'$, то $S,S' = \text{B}$,
або кожна ситуація нееквівалентна своїй похідній.

Доведемо її.

Нехай $S = \text{П}$. Тоді кожна її похідна $S' = \text{В}$. Але тоді П, $S' \sim S' = \text{В}$, тобто виграшна.

Нехай $S = \text{В}$. Якби $S, S' = \text{П}$, то кожна похідна для ситуації $\left(S, S'\right)$ була б виграшною. Але будь-яка похідна для $\left(S, S'\right)$ має вигляд або S, S'' (S'' — деяка похідна для S'), або вигляд $S1', S2'$ (деякі дві похідні ситуації S). Серед них виграшною була б і S', S' (йдеться про дві однакові похідні), а це хибно (теорема 1).

Спробуємо кожній ситуації поставити у відповідність невід'ємне ціле число — номер її стану.

Означення 2

Вважатимемо, що:

- Дві еквівалентні ситуації перебувають в одному стані. Кожна програшна ситуація має стан із номером нуль (коли камінців у купці 0, програв Перший, хоч такого варіанту в грі і не трапиться).
- Кожна виграшна ситуація має стан з номером, більшим від нуля.
- Якщо дві ситуації в одному стані, то вони еквівалентні.

Читач помітив, що деякі з цих означень фактично є теоремами, правда, легко зрозумілими. Ми ж вважатимемо їх означеннями.

Постає питання: як визначити номер стану виграшної ситуації? Це можна зробити індуктивним способом, знаючи стани всіх похідних.

Проілюструємо це на прикладі задачі про камінці, обмеживши лише випадком, коли маємо лише одну купку, у якій M камінців, і вкажемо спосіб, як визначити номер стану кожної зі 100 однокупкових ситуацій (в купці не більше ніж 100 камінців!).

При $M = 1$ та $M = 2$ маємо стан $C = 0$.

При $M = 3$ можливий лише один хід $3 \to 1,2 = \text{П}$ (стан 0), тому логічно вважати для $M = 3$, що $C = 1$.

При $M = 4$ теж можливий лише один хід $4 \to 1,3 = \text{В}$ (стан 1). Тому $4 = \text{П}$ і тому $C = 0$ (всі програші еквівалентні).

Продовжуючи далі, матимемо: $M = 5$. Можливі ходи $5 \to 1,4$ (стан 0) та $5 \to 2,3$ (стан 1). Тобто $5 = \text{В}$ (має програшну похідну, і тому $C \neq 0$). Але за теоремою 2 ситуація 5 не може мати й стану 1, інакше вона була б еквівалентна своїй похідній. А це означає, що логічно вважати стан 5 таким, що дорівнює 2. $C = 2$.

Продовжимо для інших M. Тут нас чекають труднощі.

Так, вже для $M=6$ ми зустрінемося з невизначеністю: яким вважати стан для 6 — $C=1$ чи $C=3$?

Результат $C=1$ можливий лише тоді, коли $6\sim 3$, тобто $3,6=\Pi$. І це дійсно так (доведення опущено).

Тому для 7 маємо: $7\to 1,6$ (стан 1) $7\to 2,5$ (стан 2) $7\to 3,4$ (стан 1).

Серед похідних усі виграші. Звідки $7=\Pi$ і $C=0$.

Для 8 маємо: $8\to 1,7$ ($C=0$) $8\to 2,6$ ($C=1$) $8\to 3,5$ (C не визначено). Шукаючи похідні для (3, 5), можна було б скористатися графом гри і врешті-решт довести, що $8\sim 5$, тому $C=2$. Але це інший варіант того ж пошуку всіх проміжних похідних.

Означення 3

Назвемо для кожної виграшної ситуації S первісною ту, множина станів похідних якої містить без пропусків кожне із цілих чисел діапазону $0\ldots k$ ($k\geq 0$). (Опустимо запитання: «Чи існують такі первісні?»)

Тоді справедливим буде твердження: для кожного k первісна має стан $k+1$ (пригадаймо індуктивний спосіб, яким ми користувалися для ситуацій при $M=3$, $M=5$). А тому доречно її позначити P_{k+1}, адже $k+1$ — номер її стану.

Ми вже зустрічалися з первісними:

$P_0=1$, $P_1=3$, $P_2=5$. Справедливою є теорема 3.

Стан будь-якої ситуації S — це найменше із чисел 0, 1, 2, …, якого нема серед станів похідних S'.

Зрозуміло, що на підставі теореми 2 стан похідної S' не може дорівнювати стану ситуації S. Але чому саме він — найменше з чисел? Дамо докладне доведення (на прикладі гри з купками камінців), позначивши через X найменше невід'ємне ціле, якого немає серед станів похідних, а через P_k — яку-небудь з первісних, що має стан k (k — невід'ємне ціле).

Нехай S — нерухома. Тоді зрозуміло, що вона програшна, тобто має стан 0. Але тоді множина номерів станів можливих її похідних порожня, і тому $X=0$.

Нехай $S=\Pi$. Тоді кожна її похідна виграшна, тобто стан кожної похідної більший за 0. І тому $X=0$. І справді, стан Π-ситуації за означенням дорівнює 0.

Нехай $S=\text{В}$. Якби для деякої S ми мали $C\neq X$, тоді ситуація S,P_X була б виграшною, тобто серед похідних ситуації (S,P_X) існував би програш. Але будь-яка похідна для (S,P_X) має вигляд або S',P_X, або вигляд $S,P_X{}'$. Перша не може бути програшною, адже X не міститься серед станів похідних. А друга має вигляд S,P_k, де

$k = 1,1,2,\ldots X-1$ (за означенням первісної), тобто теж не може бути програшною, інакше порушувалась би теорема 2.

І нарешті — найбільш несподіваний поворот у наших замітках. Виявляється, що для визначення стану будь-якої ситуації S потрібно знати лише стани, у яких перебувають ситуації з одним об'єктом гри (одна купка, одна кость доміно тощо), які не так вже складно й обчислити, спираючись на вказану теорему 3, якщо знати, що для двох і більше об'єктів гри побудована таким індуктивним способом система номерів станів базується на операції xor (ціле побітове виключаюче АБО).

Тобто справедливим буде твердження, яке ми назвемо

Теорема 4

Стан ситуації S_x, S_y дорівнює $x \operatorname{xor} y$.

Ця формула справедлива й для довільного числа об'єктів гри, адже xor допускає комутативність, дистрибутивність, асоціативність. (За браком місця залишаємо її без доведення, у якому немає потреби ані для обізнаного в інформатиці спеціаліста, якому це твердження майже очевидне, ані для учасника олімпіади, якому більше потрібен факт, а не його доведення.)

Ця теорема (мабуть, найважливіша!) дає можливість порахувати номер стану не лише для кожної однокупкової ситуації, а взагалі для довільної кількості купок.

Пам'ятаєте, ми зупинилися на стані ситуації 6. Похідні $6 \to 1,5 \sim 5$ (стан 2) $6 \to 2,4 \sim 1$ (стан 0). А тому за теоремою 5 маємо для 6 (стан 1), тобто $6 \sim 3$ (адже $3 = P_1$).

Далі мали проблеми зі станом для 8. Виявляється, після застосування теореми 5 немає жодних проблем:

похідні $8 \to 1,7 \sim 1$ (стан $0 \operatorname{xor} 0 = 0$) $8 \to 2,6 \sim 3$ (стан $0 \operatorname{xor} 1 = 1$) $8 \to 3,5$ (стан $1 \operatorname{xor} 2 = 3$). Тому маємо для 8 стан 2 (тобто $8 \sim 5$, а $5 = P_2$). Якщо продовжимо, знайдемо: для 9 стан 1, для 10 стан 0, для 11 стан 2, для 12 стан 1, для 13 стан 3 (нарешті знайдено однокупкову первісну; раніше її заміняла 3, 5) тощо.

Подібним чином можна знайти стани всіх 100 можливих однокупкових ситуацій.

Тому для ситуації S, яку записано в текстовому файлі, взагалі немає потреби створювати жодного масиву, крім тих, що вже були нами заведені до того, як ми почали читати дані з файла (в задачі про камінці це масив $C[1\ldots100]$, де записані стани кожної з можливих 100 однокупкових ситуацій: $C[k]$ — номер стану, у якому перебуває купка з k камінцями).

```
Покласти Z:=0; Q:=[ ];
     Поки не закінчився файл
     пц
        Читати з файла чергове A
        Z:=Z xor C[A]
        Q:=Q+[A]
     кц
```

Зрозуміло, що при $Z=0$ матимемо програш, в інших випадках — виграш.

Як вказати у випадку виграшу бодай один ВХ?

Для цього потрібно знайти серед купок, кількості камінців у яких записані в множині Q, таку (зі станом X), для якої існує похідна (зі станом Y), що задовольняє умову:

```
X xor Y xor Z=0.
```

Саме тоді, як нескладно порахувати, початкова ситуація S зі станом $Z>0$ перетвориться на gпрограшну.

Легко скласти і Pascal-програму, яка розв'язує за лічені секунди навіть задачу, у якій кількість купок камінців сягає 100 000.

ВІД АВТОРА

Автори задач, вміщених у книжці, — вчителі-практики звичайних шкіл з Новограда-Волинського та Ружина, добре відомі педагогічній громадськості області. Вони давно і плідно працюють в олімпіадній інформатиці, їх учні протягом десятків років є переможцями обласних олімпіад і успішно виступають на Всеукраїнських олімпіадах. У складі творчої групи Обласного інституту післядипломної педагогічної освіти вони впродовж багатьох років складають цікаві задачі для олімпіад, більшість з яких розв'язуються застосуванням відомих алгоритмів інформатики; готують команди області для участі в IV турі та очолюють ці команди.

Зараз у продажу чимало книжок, що допомагають опанувати операційні системи та прикладні програми, працювати у медійних засобах та інтернеті, але майже відсутні книжки для самоосвіти вчителя інформатики в галузі алгоритмізації програмування, які б давали змогу успішно працювати з учнями в позакласній роботі. Всі задачі цього збірника складені авторами і пропонувались на районних (Р) і обласних (О) олімпіадах Житомирської області, на п'яти обласних інтернет-олімпіадах (І), деякі з них використані на олімпіадах сайту E-olimp, створеного їх колегами С. С. Жуковським та А. В. Присяжнюком.

У цій книжці читачі знайдуть чимало класичних алгоритмів, необхідні поради, ґрунтовні пояснення авторських розв'язків та програми, що реалізують алгоритми для задач.

І. А. Лисогор, методист Обласного інституту післядипломної педагогічної освіти.

ЗМІСТ

С. В. Матвійчук. Навчити учнів програмувати 3
Передмова 4
Сторінки 6
Середнє з чисел 7
Оптимальна купівля CD 8
Кількість днів 9
Модель годинника з боєм 11
Нулі в кінці запису $N!$ 13
Симетрія чисел 15
Популяція роботів 18
Комп'ютеризація 19
Будівництво плоту 21
Олімпіадна задача про олімпіаду 23
Інтернет-кафе 25
Оптимальне об'єднання файлів 25
Степінь двійки 27
Кількість цифр факторіала 29
Усі цифри факторіала або Процедури «довгої арифметики» 30
Двійки та п'ятірки, або Продовження «довгої арифметики» 40
Найбільший степінь двійки 44
N-ЗНАЧНІ ЧИСЛА 47
Одиниці 50
Піднесення до степеня 52
Тук-тук, або Матриця з многокутників 60
Аркуш у лінійку 62
Зафарбовані клітинки 65
Альпініст, або Метод хвилі 68

Декілька підходів до мафії 72
CD-біржа 75
Лижні маршрути 78
Покатаймося на ліфті 80
Команда мера як найбільший повний підграф 83
Багатопроцесорний науковий проект 88
Арифметичний ребус 90
В. Л. Дідковський. Складні на перший погляд 95
Передмова 96
Різні цифри 98
Операція 99
Зелена пляма 103
Лінійка 107
Кабельне телебачення 109
Валюта 114
Гра в кості 117
Доміно 120
Моделі 123
Кубики 125
Вірус 128
Прямокутники з різних квадратів 132
Аеропорт 142
Атестація 144
Тетраедр 149
Аварійний виїзд 152
Паркет 154
Найменше спільне кратне 157
Довге число 159

Paint 161
Paint3D 164
Візерунки 165
Путівки 168
Розмін 170
Відрізки 175
Відрізки-2 176
Паливо 179
Полігон 183
Сума 186
Підручники 187
Додаток 1. Алгоритми на графах 195
Додаток 2. Хто не зміг, той програв... 205
ВІД АВТОРА .....