Розв'язання задачі «Альтернативні шляхи»

Автор: Ілля Порубльов

Основні ідеї кількома словами: максимальна сума знаходиться динамічним програмуванням, кількості «гарних» шляхів зберігаються і обробляються окремо для різних «відставань» суми шляху від максимально можливої; запропоновано два різні способи такої обробки.

1. Знаходження максимально можливої суми

Знаходження (лише) максимально можливої суми — відома задача динамічного програмування.

Прочитаємо вхідні дані у двовимірний масив `mass` (*таблицю даних*) і заведемо масив `est` (*таблицю оцінок*), елементи котрого будуватимемо за правилами:

1. елементи 1-го рядка та 1-го стовпчика — за формулами

$$\texttt{est}[1,j] = \sum_{k=1}^{j} \texttt{mass}[1,k], \quad \texttt{est}[i,1] = \sum_{k=1}^{i} \texttt{mass}[k,1], \tag{1}$$

   тобто суми всіх елементів рядка (стовпчика) таблиці даних від першого по поточний включно;

2. решту елементів — за формулою

$$\texttt{est}[i,j] = \max\left(\texttt{est}[i-1,j], \texttt{est}[i,j-1]\right) + \texttt{mass}[i,j] \quad \text{при } i \geqslant 2,\ j \geqslant 2, \tag{2}$$

   тобто вибирається більший серед верхнього сусіднього та лівого сусіднього елементів таблиці оцінок, і до нього додається поточний елемент таблиці даних.

Наприклад, на рис. 1 $\texttt{est}[2,2] = 8$ будується як максимум з $\texttt{est}[2,1] = 6$ та $\texttt{est}[1,2] = 7$ плюс число самóї клітинки $\texttt{mass}[2,2] = 1$.

Лема 1. *Кожна клітинка таблиці оцінок, побудованої згідно (1)–(2), зберігає максимально можливу суму, яку можна набрати, дійшовши до даної клітинки.*

2	5	7	2
4	1	2	8
7	5	9	7

Таблиця даних

2	7	14	16
6	8	16	24
13	18	27	34

Таблиця оцінок

Рис. 1. Приклад

2. Знаходження кількості «гарних» шляхів

Очевидно, що: при $(i = 1)$ `or` $(j = 1)$ кількість «гарних» шляхів дорівнює 1; при $\texttt{est}[i-1,j] > \texttt{est}[i,j-1] + K$ усі «гарні» шляхи до (i,j)-ої клітинки проходять через $(i-1,j)$-шу, при $\texttt{est}[i-1,j] < \texttt{est}[i,j-1] - K$ — через $(i,j-1)$-шу; у випадку $|\texttt{est}[i-1,j] - \texttt{est}[i,j-1]| \leqslant K$, серед «гарних» шляхів є i шляхи через $(i-1,j)$-у, i через $(i,j-1)$-у.

Це наштовхує на думку, ніби кількість «гарних» шляхів рівна (відповідно) або кількості для $(i-1,j)$-ої клітинки, або $(i,j-1)$-ої, або сумі цих кількостей. Але насправді остання «гілка» цих міркувань неправильна.

На рис. 2, прийти у клітинку $(1;3)$ можна одним «гарним» шляхом (RR), у $(2;2)$ — двома (DR та RD). А якщо спробувати продовжити ці шляхи до клітинки $(2;3)$, то RRD (сума 15) та DRR (сума 13) виявляються «гарними», а RDR (сума 11) — ні: сумарне «відставання» суми шляху від максимально можливої склало $15-11 = 4$ $(> 3 = K)$.

знач. (`mass`) 1 оцінка (`est`) 1 шлях | сума · | 1	знач. (`mass`) 3 оцінка (`est`) 4 шлях | сума R | 4	знач. (`mass`) 8 оцінка (`est`) 12 шлях | сума RR | 12
знач. (`mass`) 5 оцінка (`est`) 6 шлях | сума D | 6	знач. (`mass`) 4 оцінка (`est`) 10 шлях | сума DR | 10 RD | 8	знач. (`mass`) 3 оцінка (`est`) 15 шлях | сума RRD | 15 DRR | 13

(при $K = 3$)

Рис. 2. Приклад переліку «гарних» шляхів. Вказані значення числа у клітинці, оцінка клітинки згідно (1)–(2) та повний перелік «гарних» шляхів до кожної клітинки (при K=3). "D" та "R" означають перехід на одну клітинку вниз та на одну клітинку праворуч відповідно

З'ясуємо, *звідки взялося* це «відставання». При побудові $\texttt{est}[2,2]$ маємо $\texttt{est}[1,2] < \texttt{est}[2,1]$; отже, шлях, що приходить до $(2;2)$ через $(1;2)$ буде гіршим за оптимальний на $\texttt{est}[2,1] - \texttt{est}[1,2] = 2$. Таким чином, шлях RD має «відставання» 2, а для шляхів DR та RR (всі переходи котрих відбуваються відповідно до (1) та (2)) «відставання» рівні 0.

Тепер розглянемо шлях RDR. На частині "RD" вже було «відставання» 2, до котрого додалася ще неоптимальність переходу $(2;2) \to (2;3)$ (величиною $\texttt{est}[1,3] - \texttt{est}[2,2] = 2$). Сумарно $2 + 2 = 4 > 3 = K$.

Тож введемо поняття *«затримки»*, що стосується кожного одиничного переходу ("R" або "D"):

1. якщо даний перехід єдиний (R у 1-му рядку або D у 1-му стовпчику), «затримка» рівна 0;
2. При $i \geqslant 2$, $j \geqslant 2$ для знаходження «затримок» достатньо обчислити:
   (a) більшу з попередніх оцінок $\texttt{max\_est} := \max(\texttt{est}[i-1,j], \texttt{est}[i,j-1])$,
   (b) нову оцінку $\texttt{est}[i,j] := \texttt{max\_est} + \texttt{mass}[i,j]$,
   (c) «затримки» $\texttt{add\_sh\_U} := \texttt{max\_est} - \texttt{est}[i-1,j]$ та $\texttt{add\_sh\_L} := \texttt{max\_est} - \texttt{est}[i,j-1]$.

Іншими словами, якщо даний перехід оптимальний (са́ме він вибирається згідно (2)), «затримка» рівна 0; якщо даний перехід *не*оптимальний (при виборі згідно (2) його оцінка строго менша оцінки іншого можливого переходу), «затримка» рівна різниці оцінок.

Цілком очевидна наступна лема:

Лема 2. *«Відставання» будь-якого шляху дорівнює сумі «затримок» усіх його переходів.*

2.1. Алгоритм підрахунку кількості «гарних» шляхів на основі рекурсії з запам'ятовуваннями

Введемо величину $T(i,j,d)$ — кількість шляхів з клітинки $(1;1)$ до клітинки $(i;j)$, «відставання» котрих не перевищує d (при $0 \leqslant d \leqslant K$).

При приході до клітинки згори, «відставання» шляхів збільшуються на `add_sh_U`, приході справа — на `add_sh_L`. Тому має місце співвідношення

$$T(i,j,d) = T(i-1,j,d-\texttt{add\_sh\_U}) + T(i,j-1,d-\texttt{add\_sh\_R}). \tag{3}$$

Звичайно, потрібно задати ще початкові умови:

1. «відставання» не може бути від'ємним, тому при $d<0$ вважаємо $T(i,j,d)=0$;
2. у 1-му рядку та 1-му стовпчику шлях єдиний, тому при $((i=1)\,\texttt{or}\,(j=1))\,\texttt{and}\,(d\geqslant 0)$ маємо $T(i,j,d)=1$.

Формулу (3) можна реалізувати просто рекурсивною функцією, але такий розв'язок, звичайно, не буде ефективним.[1] А от якщо зробити «мемоізовану» версію, тобто запам'ятовувати відповіді вже розв'язаних підзадач — щоб у разі повторного звернення до такої ж підзадачі не розв'язувати її заново, а підставляти готову відповідь, отримаємо *дуже* ефективний *за часом* алгоритм.

Звичайно, алгоритм, що працює з великим *три*вимірним масивом, не можна вважати ефективним за об'ємом пам'яті. Але виявляється, що (при вказаних в умові задачі та пам'ятці учасника обмеженнях) пам'яті все-таки вистачає.

Тож далі ми розглянемо алгоритм, значно ефективніший за пам'яттю і не менш ефективний за часом роботи, але вважатимемо, що це просто інший спосіб, котрий *був би* істотно кращим, *якби* мало місце значно «жорстіше» співвідношення між розмірами задачі та об'ємом доступної пам'яті.

2.2. Ітеративний алгоритм підрахунку кількості «гарних» шляхів

На основі леми 2 вже неважко сформулювати наступну, котра дає ключ до ефективного підрахунку кількостей «гарних» шляхів.

Лема 3. *Нехай для $(i-1,j)$-ої та $(i,j-1)$-ої клітинок відомі оцінки `st[·,·]` та «розподіли» кількостей «гарних» шляхів: скільки мають «відставання» 0, скільки — «відставання» 1, і т. д., до «відставання» K включно*[2] *(позначимо ці послідовності як `NWay_U` та `NWay_L`). Тоді наступні дії:*

1. *обчислити величи́ни `add_sh_U` та `add_sh_L` (див. правила перед лемою 2)*
2. *зсунути «розподіли» відповідно до розрахованих «затримок», тобто для елементів `NWay_U` збільшити «відставання» на `add_sh_U`, елементів `NWay_L` — на `add_sh_L`;*
3. *«з'єднати» зсунуті `NWay_U` та `NWay_L`, пододававши кількості шляхів з однаковими затримками та повідкидавши відставання, котрі після збільшення стали строго більшими K*

призведуть до знаходження «розподілу» кількостей «гарних» шляхів для (i,j)-ої клітинки.

Тепер ми знаємо *і* «розподіли» «гарних» шляхів для 1-го рядка та 1-го стовпчика, *і* спосіб побудови «розподілів» при $i\geqslant 2$, $j\geqslant 2$. А значить, можемо побудувати «розподіли» для усіх клітинок 2-го рядка (зліва направо), потім для усіх клітинок 3-го рядка, і т. д.[3] Насамкінець, після обчисленння «розподілу» для останньої (M,N)-ої клітинки, лишається просто пододавати кількості шляхів для усіх «відставань» ($\leqslant K$).

2.2.1. Використання «зсунутого злиття» Ситуація, коли до клітинки справді ведуть шляхи з усіма значеннями «відставань» (від 0 до K), можлива. Але для значної частини клітинок буде інакше: не рівними 0 будуть кількості шляхів лише для досить малої частини «відставань».

Тому краще зберігати `NWay_U` та `NWay_L` *не* у вигляді масивів, де «відставання» виступає індексом, кількість шляхів — значенням, *а* у вигляді структури з рис. 3.

У `CellEst` поле `est` зберігає оцінку клітинки (в смислі формул (1)–(2)), `num` — кількість значень «відставань», для котрих є «гарні» шляхи, а масив `nn` (елементи з 0-го по (`num`−1)-й) — самі́ кількості шляхів, у форматі «до клітинки ведуть `v` шляхів з «відставанням» `sh`», масив впорядкований за зростанням `sh`.

(Наприклад, для клітинки $(2;2)$ з рис. 2 `est`$=10$, `num`$=2$, `nn[0]`$=$ $=$(`sh:0`, `v:1`), `nn[1]`$=$(`sh:2`, `v:1`).)

```
type Ways=record
            sh:byte;
            v:longint;
          end;
    WaysArr=array[0..MAXK] of Ways;
    CellEst=record
              est:longint;
              num:byte;
              nn:WaysArr;
            end;
```

Рис. 3. Стуктури даних для зберігання кількостей шляхів та «відставань»

При застосуванні до цих структур даних, алгоритм з леми 3 набуває вигляду, зображеного на рис. 4. Цей алгоритм спирається на відому ідею злиття (слияние, merge) впорядкованих послідовностей, але злиття проходить з особливостями: по-перше, зливаються не самі́ значення вхідних послідовностей, а їхні «зсуви» на `add_sh_U` та `add_sh_L`;

[1] час роботи буде приблизно пропорційний добутку кількості *(!!!)* «гарних» шляхів на розміри поля

[2] в даному алгоритмі рахується кількість шляхів, «відставання» яких *в точності дорівнює* вказаному

[3] причому, нема потреби зберігати «розподіли» для всіх клітинок вхідної таблиці, а достатньо для двох сусідніх рядків, зсуваючи поточну позицію по мірі обробки

```
if CU.est>=CL.est then begin
  add_sh_U:=0; add_sh_L:=longint(CU.est)-longint(CL.est);                    (* будуємо <<затримки>>  *)
  cres.est:=CU.est+the_v;                                                    (* будуємо оцінку        *)
end else begin
  add_sh_U:=longint(CL.est)-longint(CU.est); add_sh_L:=0;
  cres.est:=CL.est+the_v;
end;
the_eee:=the_est+the_est2-the_v;
Delta:=est2[1,1]-the_eee;
if Delta>K then begin
  cres.num:=0; exit                  (* див. лему 4 *)
end;
local_K:=THE_K-Delta;                (* див. лему 5 *)
i:=0; j:=0; k:=0;
while((i<CU.num)and(j<CL.num)and    (* жодна з вхідних послідовностей не закінчилася, і значення       *)
(longint(CU.nn[i].sh)+add_sh_U<=local_K)and  (* <<відставань>> (включно з новими <<затримкамии>>)      *)
(longint(CL.nn[j].sh)+add_sh_L<=local_K)) do begin  (* не перевищили <<запасу>> на <<гарні>> шляхи     *)
  if longint(CU.nn[i].sh)+add_sh_U=longint(CL.nn[j].sh)+add_sh_L then begin
    cres.nn[k].sh:=CU.nn[i].sh+add_sh_U; (* значення <<відставань>> (включно з новими <<затримкамии>>) *)
    cres.nn[k].v := CU.nn[i].v + CL.nn[j].v; (* однакові, кількості шляхів додаються                  *)
    inc(i); inc(j); inc(k);
  end else
    if longint(CU.nn[i].sh)+add_sh_U < longint(CL.nn[j].sh)+add_sh_L then begin
      cres.nn[k].sh:=CU.nn[i].sh+add_sh_U; (* <<відставання>> плюс <<затримка>> верхньої клітинки      *)
      cres.nn[k].v:=CU.nn[i].v;     (* менші, ніж лівої, записуємо у результат поточний елемент CU     *)
      inc(i); inc(k);
    end else begin
      cres.nn[k].sh:=CL.nn[j].sh+add_sh_L; (* те саме для лівої *)
      cres.nn[k].v:=CL.nn[j].v;
      inc(j); inc(k);
    end;
end;
while((i<CU.num)and(longint(CU.nn[i].sh)+add_sh_U<local_K)) do begin
  cres.nn[k].sh:=CU.nn[i].sh+add_sh_U;     (* для випадку, коли послідовність nn для лівої клітинки    *)
  cres.nn[k].v:=CU.nn[i].v;      (* вже закінчилася, а у послідовності верхньої ще є <<гарні>> шляхи   *)
  inc(i); inc(k);
end;
while((j<CL.num)and(longint(CL.nn[j].sh)+add_sh_L<=local_K)) do begin
  cres.nn[k].sh:=CL.nn[j].sh+add_sh_L;     (* аналогічно *)
  cres.nn[k].v:=CL.nn[j].v;
  inc(j); inc(k);
end;
cres.num:=k;
```

Рис. 4. Алгоритм побудови послідовності кількостей шляхів поточної (i,j)-ої клітинки (`cres`) за відомими послідовностями $(i-1,j)$-ої (`CU`) та $(i,j-1)$-ої (`CL`) за допомогою «зсунутого злиття», з використанням оптимізації відтинань по «зустрічним» оцінкам

по-друге, причиною завершення результуючої послідовності слід брати не лише закінчення обох вхідних послідовностей, а ще й випадок, коли («зсунуте») значення «відставання» перевищує K, тобто шляхи перестають бути «гарними».[4]

2.2.2. Відтинання гарантовано не «гарних» шляхів за допомогою «зустрічних» оцінок За умовою, «у більшості випадків *мінімальнo* можлива сума ... *суттєво менша* за максимально можливу». Отже, значна частина клітинок «безперспективна» в тому смислі, що через них не може проходити жодного «гарного» шляху (так буде не завжди, але, для відповідних вхідних даних, досить часто).

В умові сказано, що кількість «гарних» шляхів гарантовано не перевищує 10^9. Напрошується виновок, що це зроблено задля спрощення, щоб не писати «довгу арифметику». Але насправді звідси можна зробити значно «потужніший» висновок. Кількість *взагалі всіх* шляхів з $(1;1)$ до $(M;N)$ складає C^M_{M+N-1}, що при $M=N=200$ буде $\approx 5{,}1\cdot 10^{118}$. А це (при великих M та N) означає, що значна частина шляхів далеко *не* «гарні». Алгоритм, що користується *лише* ідеями розділу 2.2.1, кінець кінцем відкидає їх, але робить це лише тоді, коли вони «з'єднуються» з «більш обнадійливими»[5] (і з'являється велика «затримка» переходу). А можна «розрізняти» їх *значно раніше*.

Для цього «розвернемо» формули (1)–(2), щоб рахувати «зустрічні» оцінки (смисл котрих — «яку найкращу суму можна назбирати, прийшовши з (i,j)-ої клітинки до останньої (M,N)-ої?»):

$$\texttt{est2}[M,j]=\sum_{k=j}^{N}\texttt{mass}[M,k],\quad \texttt{est2}[i,N]=\sum_{k=i}^{M}\texttt{mass}[k,N], \tag{1'}$$

$$\texttt{est2}[i,j]=\max\left(\texttt{est2}[i+1,j],\texttt{est2}[i,j+1]\right)+\texttt{mass}[i,j]\quad \text{при } i<M,\ j<M. \tag{2'}$$

[4] Алгоритм рис. 4 включає також і оптимізацію, викладену в розділі 2.2.2, тому все виявляєтьвся трохи складніше

[5] Тому можливо (і в окремих тестах так і є), що у ітеративної програми, котра не користується даною оптимізацією, з'являтимуться (насправді непотрібні) проміжні величи́ни, які *не* поміщаються у `longint`.

Розглянемо величину $\texttt{eee}[i,j] := (\texttt{est}[i,j] + \texttt{est2}[i,j] - \texttt{mass}[i,j])$. Вона задає максимальну суму, котру можна назбирати, йдучи з $(1;1)$-ої клітинки до (M,N)-ої *через* (i,j)-*у*. Тоді $\texttt{eee}[i,j] \leqslant \texttt{est}[M,N]$, причому рівність досягається лише для тих клітинок, через які проходить(ять) максимальний(і) шлях(и).

Тепер позначимо $\Delta_{ij} := (\texttt{est}[M,N] - \texttt{eee}[i,j])$. Щодо цієї величини виконуються наступні твердження.

Лема 4. *При $\Delta_{ij} > K$ шляхи через (i,j) можна не досліджувати, бо серед них нема «гарних».*

Лема 5. *При $\Delta_{ij} \leqslant K$, достатньо проводити «зсунуте злиття» для (i,j)-ої клітинки лише для значень «відставань» $\leqslant K - \Delta_{ij}$.*

Доведення. Найкраща сума, котру можна назбирати, йдучи через дану клітинку, гірша взагалі найкращої на Δ_{ij}. Значить, далі будуть «неоптимальні переходи», сума «затримок» котрих якраз і складе Δ_{ij}. А отже, шляхи, «відставання» котрих зараз більші за $K - \Delta_{ij}$, все одно будуть відкинуті. ■

Ці спостереження дають можливість скоротити об'єм роботи при «злиттях» і зменшити час роботи програми (щоправда, для різних вхідних даних виграш буде різний, від дуже значного до майже непомітного).

3. Оцінки часу роботи та об'єму пам'яті

Як уже згадувалося, час роботина різних вхідних даних однакового розміру може сильно відрізнятися. Тому доведеться сказати, що для найшвидших алгоритмів («мемоізованої» реалізації формули (3) та ітераційної реалізації, що використовує всі ідеї розділу 2.2.2) кількість дій у гіршому випадку оцінюється як $O(M{\cdot}N{\cdot}K)$, але для багатьох вхідних даних оцінка не набагато перевищує $O(M{\cdot}N)$.

Об'єм пам'яті мемоізованої рекурсії становить $O(M{\cdot}N{\cdot}K)$; для ітеративних алгоритмів[6] — $\theta(N{\cdot}(M{+}K))$.

[6] завдяки спостереженню з виноски 3 на стор. 2