Глава 14

Игры двух лиц

В этой главе…

- Примеры антагонистических игр с полной информацией
- Игры с известной стратегией
- Выигрышные и проигрышные позиции
- Использование рекурсии и таблиц ходов
- Числовое оценивание позиций

Игры, в которых участвуют два игрока, знакомы нам с детства — от крестиков-ноликов на поле 3×3 до шашек и шахмат. Как правило, такие игры являются *антагонистическими* — выигрыш одного игрока означает проигрыш другого. Игры также бывают *с полной* и *с неполной информацией* — в зависимости от того, все ли известно игроку о позиции своей и соперника. Примеры игр с полной информацией — варианты крестиков-ноликов, шахматы или шашки, с неполной — домино или карточные игры, в которых соперник скрывает, что у него "на руках".

Серьезные игры вроде шахмат с самых первых ходов имеют очень много вариантов продолжения. Из-за этого для них очень трудно сформулировать "алгоритм игры", придерживаясь которого, игрок может выиграть. Именно отсутствие таких алгоритмов делает эти игры интересными для соревнования людей и чрезвычайно сложными для программирования. Хотя нельзя не отметить, что уже в 2005 и 2006 годах наилучшие шахматные программы в матчах против чемпионов мира убедительно переигрывали людей.

В процессе игры ее участники, как правило, по очереди совершают *ходы*, благодаря которым образуются *позиции* в игре. Варианты процесса любой игры обычно можно представить корневым ориентированным деревом (иногда — направленным графом). Узлы в этом дереве представляют позиции (корневой узел — исходную позицию, узлы-листья — заключительные позиции, из которых нельзя сделать ход). Дуги дерева представляют ходы.

В реальных играх деревья позиций разветвляются очень широко, поэтому поиск хода с помощью перебора, т.е. обхода такого дерева, оказывается очень длительным. Для сокращения перебора существуют общие способы, аналогичные методу ветвей и границ, однако не они будут предметом нашего внимания.

Игры с полной информацией, представленные в данной главе, являются играми *с известной стратегией*. Это значит, что один из игроков, выбирая право первого хода и каждый свой ход некоторым не очень сложным способом, может гарантировать себе выигрыш при любой игре соперника (естественно, в пределах правил). Главное

в этих играх — найти методику выбора ходов, гарантирующих выигрыш, и реализовать ее в программе по возможности эффективно.

В решениях задач данной главы отсутствует защита от ходов игрока, нарушающих правила игры. Добавьте такую защиту к решению каждой задачи самостоятельно — в большинстве ситуаций это довольно просто. Защита должна, как минимум, игнорировать ход соперника, нарушающий правила игры.

14.1. Анализ позиций и выбор хода

14.1.1. Выигрышные и проигрышные позиции

Задача 14.1. Касса содержит C копеек. Два игрока поочередно забирают из кассы от 1 до M копеек. Выигрывает игрок, после хода которого касса станет пустой. Программа должна читать положительные числа C и M (от 1 до 10^4), выбирать право первого хода и выигрывать.

Пример. При $C=17$ и $M=9$ последовательность ходов (первый — 5, второй — 6, первый — 6) означает выигрыш первого игрока, поскольку $17-5-6-6=0$.

Анализ и решение задачи. В любой игре присутствует такое понятие, как *позиция*. Неформально говоря, это совокупность текущих параметров игры, которую изменяют игроки своими ходами и по которой в конце концов определяется выигрыш; кроме того, позиция должна однозначно задавать возможные ходы из нее.

В данной задаче позицией естественно считать *остаток* в кассе R и номер игрока, имеющего право хода.

Предположим, что сейчас наш ход. Начнем с ситуаций, в которых остаток R мал. Если $R=0$, мы уже проиграли. Если $R=1$, $R=2$, …, $R=M$, у нас есть победный ход — число R, после которого сумма станет равной C. Если $R=M+1$, то любой наш ход сделает R равным одному из чисел 1, 2, …, M, и тогда победный ход сделает соперник.

Итак, остатки $R=0$ и $R=M+1$ характеризуют *проигрышную позицию*. Каким бы ни был наш ход, соперник выиграет (конечно, если не допустит ошибки, но не стоит на это надеяться…). В то же время, в позициях с $R=1, 2, \ldots$ или M есть победный ход.

Продолжим анализ. Если $R=M+2, M+3, \ldots$ или $M+M+1$, то наш соответствующий ход 1, 2, …, M приведет соперника к проигрышной позиции с $R=M+1$. Значит, в этих позициях можно выиграть. Но если $R=2M+2$, любой наш ход приведет в позицию с R в пределах от $M+2$ до $2M+1$, начиная с которой выиграет соперник…

По этим наблюдениям нетрудно догадаться, что позиции с $R=0$, $M+1$, $2(M+1)$, $3(M+1)$, … являются проигрышными, а во всех остальных существуют ходы, "загоняющие" соперника в одну из этих проигрышных позиций, т.е. остальные позиции — *выигрышные*.

Итак, при $C \bmod (M+1) \neq 0$ исходная позиция является выигрышной, и наш первый ход — число $C \bmod (M+1)$. Иначе позиция проигрышная, и пусть начинает соперник. На каждый ход A соперника нужно отвечать ходом $M+1-A$. Программа, реализующая эти соображения, приведена в листинге 14.1.

Листинг 14.1. Простейшая игра в сумму

```
program SimpleGame;
  var c,              { исходная сумма }
```

```
      m    : integer; { максимальный ход }
      r,              { текущий остаток }
      move : integer; { ход }
      m1   : integer; { m+1 }
begin
  write('Введите сумму и максимальный ход (integer)>');
  readln(c, m);
  r := c; m1 := m+1; { инициализация }
  writeln('Макс. ход: ', m, ', остаток = ', r);
  if r mod m1 <> 0 then begin
    move := r mod m1;
    dec(r, move);
    writeln('Мой ход>', move);
    writeln('Макс. ход: ', m, ', остаток = ', r);
  end;
  while r > 0 do begin
    write('Ваш ход>'); readln(move);
    dec(r, move);
    writeln('Макс. ход: ', m, ', остаток = ', r);
    move := m1 - move;
    dec(r, move);
    writeln('my move>', move);
    writeln('Макс. ход: ', m, ', остаток = ', r);
  end;
  writeln('Простите, вы проиграли.');
end.
```

Уточним понятия, возникшие в рассмотренной задаче. Они относятся к общей теории, помогающей в построении выигрышных стратегий для довольно широкого класса игр.

> *Выигрышная позиция* — это позиция, начиная с которой можно, играя правильно, гарантированно выиграть при любой игре соперника. *Проигрышная позиция* — позиция, начиная с которой выиграть невозможно (если соперник не допустит ошибки). Иными словами, в выигрышной позиции либо игрок уже выиграл, либо ***хотя бы один*** ход приводит в проигрышную позицию, обеспечивая выигрыш ***при любой игре соперника***. В проигрышной же позиции либо игра уже проиграна, либо ***нет ни одного хода***, обеспечивающего выигрыш (при правильной игре ***соперник***а), т.е. ***любой*** ход приводит в выигрышную позицию.

Наконец, рассмотрим общую схему, которую может иметь программа или подпрограмма для любой антагонистической игры с известными выигрышными и проигрышными позициями (сравните ее с листингом 14.1).

```
создать начальную позицию;
отобразить позицию;
if позиция выигрышная then begin
  найти и выполнить свой ход;
  отобразить ход;
  отобразить позицию;
end;
```

```
while позиция незаключительная do begin
  получить ход от игрока;
  выполнить ход игрока;
  отобразить позицию;
  найти и выполнить свой ход;
  отобразить свой ход;
  отобразить позицию;
end;
отобразить сообщение о своей победе.
```

Главное — уметь определять, выигрышна ли текущая позиция, и находить ход, после которого сопернику каждый раз достается проигрышная позиция. А уточнить пункты этой схемы конкретными операторами и подпрограммами совсем несложно.

Проверять, правилен ли ход игрока, необходимо при его задании игроком. Отображать заданный игроком ход, как правило, не нужно — это происходит во время задания. В некоторых задачах ход программы определяется как результат анализа, является ли позиция выигрышной. Тогда пункт *найти и выполнить свой ход* должен включать анализ позиции. Примеры приведены в следующих подразделах.

Еще одно замечание. Если вам все-таки досталась проигрышная позиция, а возможности отдать ход сопернику нет — как ходить? В игре с идеальным соперником любой ход приведет к проигрышу, но можно пытаться "продержаться как можно дольше". Если же соперник не является идеальным, то перед каждым ходом нужно проверять, не досталась ли нам выигрышная позиция, и в этой ситуации "перехватывать инициативу".

Однако представленной выше теории проигрышных и выигрышных позиций *не всегда достаточно*. Существуют антагонистические игры (в том числе и с "бинарным" результатом "выиграл/проиграл"!), анализ которых требует оценивать позиции точнее, чем просто как выигрышные или проигрышные. В частности, могут оказаться полезными так называемые числа Спрэга–Гранди (Sprag, Grundy). Определение и примеры применения этих чисел можно найти, например, в работе [2], однако в ней подробно рассмотрены только игры, для анализа которых в действительности достаточно "бинарной" проигрышности или выигрышности.

В данной книге этот материал также не представлен, да и источников, где он был бы разъяснен достаточно обширно и понятно, на момент написания книги авторы, к сожалению, указать не могут. Можем лишь посоветовать сначала изучить игры, разобранные тут, а затем поискать материалы с более глубокой теорией в Интернете...

14.1.2. Золотое сечение

Задача 14.2. Есть две кучки спичек. Два игрока берут из них спички по очереди. За один ход игрок берет из неменьшей кучки ненулевое число спичек, кратное числу спичек в другой кучке. Выигрывает тот, кто взял последнюю спичку в одной из кучек. Программа должна реализовывать выигрышную стратегию, в частности, решать, кто должен ходить первым.

Пример. Если в кучках `1` и `3` спички, выигрывает первый игрок, забрав `3` спички. Если в кучках `2` и `3` спички, то первый игрок может взять только `2` спички из второй кучки, после чего в позиции `(2,1)` второй игрок забирает `2` спички и выигрывает.

Вход и выход. На клавиатуре задаются два положительных целых чисел (типа `integer`), обозначающих количества спичек в кучках. Ход задается числом, кратным меньшему из чисел в текущей позиции. После каждого хода выводится полученная позиция (два числа).

Анализ задачи. Поищем способ определения, является ли заданная позиция выигрышной. Пусть в позиции (a, b) первое число не меньше второго. Если $b = 0$, то позиция является проигрышной, иначе, если a кратно b, — выигрышной.

Пусть a не кратно b, т.е. $a = kb+r$, где $r = a \bmod b > 0$. Из позиции (a, b) можно перейти в позиции (r, b), $(b+r, b)$, $(2b+r, b)$, …, $((k-1)\cdot b+r, b)$. По определению, позиция (a, b) является выигрышной тогда и только тогда, когда хотя бы одна из этих позиций проигрышная. Но это можно определить рекурсивно! Поскольку во всех этих позициях первое число строго меньше a, рекурсия обязательно достигнет "дна", на котором одно из чисел равно 0 или одно из них кратно другому.

Однако такая рекурсия ***очень неэффективна*** из-за многократных рекурсивных вызовов. Можно оптимизировать время работы за счет памяти, используя табличную технику (см. главу 13 и дальнейшие задачи в данной главе). Но для этой игры есть способ получше. Докажем, что позиции $(2b+r, b)$, …, $((k-1)\cdot b+r, b)$ обязательно выигрышные, т.е. их можно не анализировать.

▸ Предположим, что одна из указанных позиций, скажем, $(l\cdot b+r, b)$, где $2 \le l < k$ и $r < b$, проигрышная. Но из проигрышной позиции ***любой*** ход ведет в выигрышную позицию. Значит, и (r, b), и $(b+r, b)$ являются выигрышными. Но тогда из выигрышной позиции $(b+r, b)$ единственный допустимый ход b приводит в выигрышную же позицию (r, b), что невозможно по определению выигрышной позиции. Полученное противоречие доказывает, что позиция вида $(l\cdot b+r, b)$ при $l \ge 2$ является выигрышной. Отсюда следует: при $a/b \ge 2$ позиция (a, b) выигрышная. Кроме того, из двух позиций (r, b) и $(b+r, b)$ одна, и только одна, является выигрышной. ◂

Итак, если $b < a < 2\cdot b$ (неравенства строгие), то достаточно только рекурсивно выяснить, является ли позиция (r, b), где $r = a \bmod b > 0$, проигрышной. Если является, то позиция (a, b) выигрышная, иначе проигрышная.

Нужно еще правильно выбрать ход при $a/b \ge 2$. Если позиция (r, b) проигрышная, то в нее ведет ход $a - a \bmod b$, иначе проигрышной обязательно будет позиция $(b+r, b)$, в которую ведет ход $a - a \bmod b - b$.

Эти соображения избавляют от повторных рекурсивных вызовов и существенно сокращают работу.

Решение задачи. Оформим решение задачи с помощью трех подпрограмм.

Процедура `makeMove` реализует собственно изъятие спичек из большей кучки.

Функция `detWin` получает при вызове количества `a` и `b` спичек в кучках, возвращает признак того, что данная позиция выигрышная, и в параметре-переменной сохраняет выигрышный ход (или наименьший возможный ход, если позиция проигрышная).

Процедура `game` получает исходную позицию и вызывает функцию `detWin`, чтобы определить, является ли эта позиция выигрышной. Если это так, она делает первый ход. Далее в цикле соперник и программа делают по одному ходу. Поскольку соперник все время загоняется в проигрышную позицию, в которой у него есть только один допустимый ход, дальнейшие вызовы `detWin` не нужны.

Решение представлено в листинге 14.2.

Листинг 14.2. Решение, основанное на рекурсивном анализе

```
var a, b : integer; { исходные числа }

procedure makeMove(var a, b, c : integer);
Begin
  if a > b then dec(a, c) else dec(b, c);
End;

function detWin(a, b : integer; var c : integer) : boolean;
   var t : integer;
Begin
  if a < b then begin
    t := a; a := b; b := t
  end;
  { гарантированно a >= b }
  if b = 0 then begin
    detWin := false; c := 0
  end else
  if (a mod b = 0) then begin
    detWin := true; c := a
  end else
  if (a div b >= 2) then begin
    detWin := true;
    if not detWin(b, a mod b , c)
      then c := a - a mod b
    else c := a - a mod b - b
  end else
  if not detWin(b, a mod b , c) then begin
    c := a - a mod b; detWin := true
  end else begin
    c := b; detWin := false;
  end;
End;

procedure game(a, b : integer);
  var c : integer;
Begin
  writeln('Позиция: ', a, ' ', b);
  if detWin(a, b, c) then begin
    makeMove(a, b, c);
    writeln('Мой ход>', c);
    writeln('Позиция: ', a, ' ', b);
  end;
  while (a <> 0) and (b <> 0) do begin
    write('Ваш ход>'); readln(c);
    { Здесь нужно добавить проверку правильности хода }
    makeMove(a, b, c);
    writeln('Позиция: ', a, ' ', b);
    if detWin(a, b, c) then begin
      makeMove(a, b, c);
      writeln('Мой ход>', c);
```

```
      writeln('Позиция: ', a, ' ', b);
    end;
  end;
  writeln('Простите, вы проиграли.');
End;

Begin
  readln(a, b); game(a, b);
End.
```

⏩ Добавьте проверку правильности хода соперника.

Решение без рекурсии. Рассмотрим выигрышную позицию (2,1); большее число будем указывать в позиции первым. Эта позиция достигается ходом 2 из проигрышной позиции (3,2), а та, в свою очередь, ходом 3 из выигрышной позиции (5,3). Следующей в этом движении назад будет проигрышная позиция (8,5), затем выигрышная (13,8). Стоп! Каждая из них, имея вид (a,b), достигается ходом a из позиции $(a+b,a)$. Числа 2 и 1 — это числа Фибоначчи, а для любых последовательных пар (c,a) и (a,b) верно, что $c=b+a$. Значит, в парах находятся последовательные числа Фибоначчи.

Нетрудно убедиться, что отношение соседних чисел Фибоначчи через раз больше и через раз меньше *золотого сечения* Ф[1], равного

$\frac{\sqrt{5}+1}{2} \approx 1{,}618034$: $2/1 > \Phi$, $3/2 < \Phi$, $5/3 > \Phi$, $8/5 < \Phi$ и т.д.

Значит, в выигрышных позициях с соседними числами Фибоначчи отношение чисел больше Ф, а в проигрышных — меньше Ф. Этот пример подводит нас к такому предположению.

- Позиция (a,b), где $a \ge b$, является выигрышной тогда и только тогда, когда $a=b$ или $a/b > \Phi$.

Докажем это утверждение.

▸ ($\Rightarrow$) Вспомним первое решение. На "дне" рекурсии находилась выигрышная позиция (a,b), в которой $a/b \ge 2$, т.е. $a/b > \Phi$ (ситуация $a=b$ в качестве выигрышной позиции невозможна в ***рекурсивных*** вызовах). За один ход a до этой позиции, т.е. в предыдущем вызове рассматривалась проигрышная позиция $(a+b,a)$, за два хода — выигрышная и т.д. Заметим: если $a/b > \Phi$, то $(a+b)/a < \Phi$. Действительно,

$(a+b)/a = 1+b/a < 1+1/\Phi = \Phi$.

Аналогично, если $a/b < \Phi$, то $(a+b)/a > \Phi$. Отсюда для всех проигрышных позиций (a,b) выполняется $a/b < \Phi$, а для выигрышных — $a/b > \Phi$.

($\Leftarrow$) Пусть $a/b > \Phi$ в позиции (a,b). Если $a/b \ge 2$, то позиция выигрышная (это доказано в первом решении). Пусть $a=b+r$, где $r = a \bmod b > 0$. В позиции (a,b) есть только один ход b, и он приводит в позицию (b,r) с $b/r < \Phi$. Из такой позиции (b,r) есть только один ход в позицию $(b-r,r)$, в которой $(b-r)/r > \Phi$. Таким образом, из позиции с отношением чисел больше Ф переходим в позицию с отношением меньше Ф, а еще через шаг — в позицию с отношением больше Ф.

---

[1] Обозначение Ф связано с именем древнегреческого скульптора Фидия — он использовал золотое сечение в пропорциях многих своих скульптур.

Каждый раз одно из чисел уменьшается, поэтому на некотором шаге получим позицию вида $(k \cdot b, b)$ с $k \geq 2$. У нее отношение чисел больше Ф, поэтому она получена через четное число шагов. Но она выигрышная, поэтому и исходная выигрышная. ◂

Для реализации описанного способа не обязательно использовать само число Ф. Нетрудно убедиться, что для позиции (a, b) условие $a/b > \Phi$ равносильно тому, что $a/b > b/(a \bmod b)$ или $a/b > b/(b + a \bmod b)$. В программе от неравенств с дробями перейдем к неравенствам с произведениями. Эти соображения реализованы в следующей процедуре.

```
function detWin(a, b : integer; var c : integer) : boolean;
  var t : integer;
begin
  toSwap := false;
  if a < b then begin
    t := a; a := b; b := t; toSwap := true;
  end;
  { a >= b }
  if b = 0 then begin
    detWin := false; c := 0
  end
  else
  if a mod b = 0 then begin
    detWin := true; c := a
  end
  else
  if (a*(a mod b) > b*b) or (a*b > b*(b + a mod b)) then begin
    detWin := true;
    if (a mod b)*(b + a mod b) > b*b then
      c := a - a mod b
    else c := a - a mod b - b;
  end
  else begin
    c := b; detWin := false;
  end;
end;
```

⏩ Напишите программу с учетом того, что в первой строке текста `'heapprop.txt'` записано количество тестов `nTest`, а следующие `nTest` строк содержат по паре целых положительных целых чисел (типа `integer`), обозначающих количества спичек в кучках. Для каждого теста выводится исходная позиция, а далее после каждого хода выводится текущая позиция (два числа).

14.1.3. Ним

Полный анализ этой игры в 1901 году опубликовал профессор Гарвардского университета Чарлз Бутон. По одной из версий, он назвал эту игру в честь старинного французского города. Однако авторам кажется более правдоподобным, что Ч. Бутон использовал устаревший английский глагол *nim* (брать, красть) или повелительное наклонение *nimm* немецкого глагола *nehmen* (брать).

Задача 14.3. На столе лежат несколько кучек камешков. Два игрока берут из них камешки по очереди. За один ход игрок выбирает любую кучку и берет

из нее любое ненулевое число камешков. Выигрывает тот, кто взял самый последний камешек. Программа должна реализовывать выигрышную стратегию, в частности, решать, кто должен ходить первым.

Пример. Если в кучках `1` и `3` камешка, выиграет первый игрок. Он берет `2` камешка из второй кучки и оставляет позицию (`1,1`). Второй игрок может взять из любой кучки только `1` камешек, после чего второй заберет последний камешек из другой и выиграет. Если в кучках по `2` камешка, то выиграет второй игрок. Если первый возьмет `2` камешка из какой-нибудь кучки, то второй заберет `2` из другой и выиграет. Если же первый возьмет `1` камешек из какой-нибудь кучки, то второй — `1` камешек из другой кучки, а с позицией (`1, 1`) уже все ясно.

Вход и выход. От клавиатуры вводится количество кучек n, $1 \le n \le 10$, затем n положительных чисел (типа `integer`) — количества камешков в кучках. Ход задается номером кучки и количеством забираемых камешков. После каждого хода выводится полученная позиция (n чисел).

Анализ задачи. Анализ позиций с двумя равными кучками прост: сколько бы камешков ни брал первый игрок из какой-либо кучки, второй будет брать столько же из другой кучки и в конце концов выиграет.

Если же кучки не равны, то цель ясна — своим ходом сделать их равными. В этой ситуации выигрывает первый игрок — первым ходом он заберет из большей кучки разницу количеств камешков и оставит второму игроку две равные кучки.

В общем случае решение связано с двоичным представлением чисел. Заметим: если числа равны, то их двоичные представления одинаковы, иначе они отличаются хотя бы в одном разряде. Поэтому поразрядная сумма по модулю `2` ("исключающее или") двух равных чисел имеет во всех разрядах `0`, т.е. равна `0`, а разных чисел — отлична от `0`.

Результат поразрядного сложения по модулю `2` для краткости будем называть *ним-суммой*, а само это сложение — *ним-сложением*. В языке Turbo Pascal оно реализовано операцией `xor`: `10 xor 5 = 15`, `5 xor 6 = 3`. Итак, позиция с двумя числами a_1 и a_2 является выигрышной тогда и только тогда, когда a_1 `xor` $a_2 \neq 0$. Но если чисел не два, а больше, то этот критерий остается в силе!

- Позиция с числами a_1, a_2, …, a_n является выигрышной тогда и только тогда, когда a_1 `xor` a_2 `xor` … `xor` $a_n \neq 0$.

К сожалению, доказательство критерия выходит за рамки этой книги.

Рассмотрим примеры. Позиция (`1,2,3`) проигрышная — двоичные представления чисел `01`, `10`, `11` дают в каждом разряде четное число единиц, из-за чего их ним-сумма равна `0`. Позиция (`5,2,4`) выигрышная — эти числа с двоичными представлениями `101`, `010`, `100` дают ним-сумму `3` с представлением `011`.

Итак, чтобы определить, является ли позиция выигрышной, достаточно вычислить ним-сумму чисел в позиции и убедиться, что она не равна `0`. Но *нужно еще определить, из какой кучки и сколько камешков взять, чтобы получилась проигрышная позиция*.

Вначале найдем, сколько камешков нужно взять. Если выбрать одно из чисел, скажем, a_1, и ним-прибавить к нему ним-сумму всех чисел $S = a_1$ `xor` a_2 `xor` … `xor` a_n, то ним-сумма (a_1 `xor` S) `xor` a_2 `xor` … `xor` a_n, очевидно, равна `0`. Значит, чтобы новая ним-сумма стала равной 0, из a_1 камешков ***должно остаться*** a_1 `xor` S. Например, в позиции (`5,2,4`) с ним-суммой `3` выберем число `2`; `2 xor 3 = 1` — это количество камешков,

которые должны остаться во второй кучке. Значит, возьмем из нее 2−1=1 камешек и получим позицию (5,1,4) с ним-суммой 0.

Но как выбрать число для ним-сложения с *S*? Вообще говоря, ним-прибавление *S* может увеличить число, например 4 xor 3 = 7 или 5 xor 3 = 6. Однако число ***должно уменьшиться*** — ведь камешки нужно не добавлять к кучке, а брать из нее! Но оказывается, что среди чисел a_1, a_2, …, a_n обязательно найдется хотя бы одно, которое при ним-прибавлении *S* уменьшится. Убедимся в этом.

Заметим: если двоичные представления двух ним-слагаемых имеют 1 в одном и том же разряде, то в их ним-сумме этот разряд равен 0. Например, 2 xor 3 = 1 (2, 3 и 1 имеют двоичные представления 10, 11 и 01 соответственно).

Найдем разряд со ***старшей единицей*** ним-суммы *S*. Тогда, по построению *S*, среди a_1, a_2, …, a_n есть нечетное количество чисел, у которых в этом разряде 1. Обозначим любое из них через *x*. В результате ним-прибавления *S* к *x* в указанном разряде будет 0, старшие разряды *x*, если есть, останутся без изменений, а возможные изменения в младших разрядах гарантированно "не перекроют" уменьшения в указанном разряде. Следовательно, ним-сумма *x* и *S* окажется меньше *x*.

Итак, искомым может быть любое из чисел, имеющих 1 в том разряде, в котором находится старшая 1 ним-суммы *S*. Чтобы найти его, переберем числа a_1, a_2, …, a_n, пока не найдем a_i, для которого a_i xor $S < a_i$.

Решение задачи. Приведем только функцию detWin, которая возвращает признак выигрышности позиции, сохраняя в параметрах-переменных номер кучки и количество камешков, представляющих ход из выигрышной позиции. Если позиция проигрышная, ход вычисляется так же, как и для выигрышной позиции, но программа в такой позиции ходить не должна.

Размеры кучек представим массивом num типа aInt = **array** [1..maxN] **of** integer, где maxN — имя константы 10. Количество кучек представим параметром n, ход — параметрами-переменными nHeap и decrem. Функция приведена в листинге 14.3.

Листинг 14.3. Функция определения позиции и хода в игре ним

```
function detWin(var num : aInt; n : byte;
    var nHeap : byte; var decrem  : integer) : boolean;
  var sum,
    numCopy : integer;
    k : byte;
begin
  sum := 0;
  for k := 1 to n do sum := sum xor num[k];
  detwin := (sum <> 0);
  k := 1;
  while num[k] xor sum >= num[k] do inc(k);
  nHeap := k;
  numCopy := num[nHeap];
  num[nHeap] := num[nHeap] xor sum;
  decrem := numCopy - num[nHeap];
end;
```

⏩ Напишите программу, аналогичную программе в листинге 14.2.

14.1.4. Таблица ходов

Задача 14.4. Касса содержит C копеек ($C \geq 3$). Два игрока по очереди берут из кассы по целому числу копеек. За ход можно взять не меньше одной копейки и не больше удвоенного числа копеек, взятых соперником на предыдущем ходу. Первый ход — одна или две копейки. Выигрывает тот, кто своим ходом опустошит кассу. Программа должна читать C (от `1` до `300`), выбирать право первого хода и выигрывать.

Пример. При $C=3$ выигрывает второй игрок — если первый взял `1` копейку, второй берет `2`, если первый взял `2` копейки, второй берет `1`. При $C=4$ выиграет первый игрок — взяв 1 копейку, он оставит в кассе 3 копейки перед ходом второго игрока.

Анализ задачи. Ясно, что позиция в данной задаче — это не просто остаток в кассе, а пара (***остаток***, ***последний ход***). Обозначим эти величины соответственно r и m.

Позиция (r,m) является выигрышной, если существует ход k (от 1 до $2m$) в проигрышную позицию $(r–k,k)$, и является проигрышной, если при любом k от 1 до $2m$ позиция $(r–k,k)$ выигрышная. Это рекурсивное определение имеет "дно": позиции $(1,k)$ и $(2,k)$ при любом возможном k являются выигрышными, $(0,k)$ — проигрышными.

Чтобы решить, является ли текущая позиция выигрышной или проигрышной, нетрудно реализовать это определение "в лоб" с помощью рекурсивной функции. Однако из-за кратных рекурсивных вызовов такое решение неэффективно. Рекурсия с запоминанием может улучшить его, но проще один раз заполнить таблицу вариантов ходов и использовать ее во время игры.

Используем таблицу `tab`, строки которой индексированы остатками, столбцы — последними ходами. Значением элемента, соответствующего выигрышной позиции, является один из возможных ходов, которые ведут к выигрышу. Если позиция проигрышная, значением элемента должно быть то, что не может быть ходом — положим его равным `0`. Тогда `0` будет признаком проигрышной позиции, а положительное значение — признаком выигрышной.

Построение таблицы. Очевидно, все элементы первой строки должны иметь значение `1`, второй — `2`. Заполним остальные строки, инициализировав все их элементы значением `0`. Пусть теперь $n \geq 3$ и $m \geq 1$. Если $n \leq 2m$, то в позиции возможен ход n, опустошающий кассу, поэтому `tab[n, m] = n`. При $n > 2m$ переберем все возможные ходы k от 1 до $2m$ и используем элементы предыдущих строк таблицы. Если `tab[n-k, k] = 0` хотя бы для одного значения `k`, значит, ход `k` приводит в проигрышную позицию $(n–k,k)$; тогда `tab[n, m] = k`. При этом естественно выбрать максимальное значение `k`, ускоряющее игру. Если же все `tab[n-k, k]` $\neq$ `0` при всех k от 1 до $2m$, `tab[n, m]` представляет проигрышную позицию и остается равным `0`.

Выбор хода с помощью таблицы. Предположим, текущей является позиция (r,m). Если $r \leq 2m$, то победный ход r является допустимым, и для его определения ***таблица вообще не нужна***. При $r > 2m$, если `tab[r, m] > 0`, ходом будет `tab[r, m]`, иначе `1` (такой ход в действительности программа никогда ***не выполняет***). Если оформить выбор хода в виде функции, возвращающей булев признак выигрышности позиции, то в первых двух ситуациях возвращается `true`, в третьей — `false`.

Ширина таблицы. Остается уточнить, какой должна быть ширина таблицы, т. е. каким может быть максимальный ход. Заметим: если последний ход равен m, то все

выполненные ходы уменьшили исходную сумму C ***не менее чем*** на $2m-1$ копеек (касса уменьшается в точности на $2m-1$, если первый ход равен 1 и каждый следующий ход вдвое больше предыдущего). Отсюда, если достигнута позиция (r, m), то $r \le C-2m+1$. Но при $r \le 2m$ для определения следующего хода (одного из чисел от 1 до $2m$) таблица не нужна. Но таблица нужна, поэтому $r > 2m$. Тогда $2m < C-2m+1$, или $2m < (C+1)/2$. Итак, достаточно таблицы шириной $(C+1)/2$.

⏩ Реализуйте представленное решение.

14.2. Оценивание позиций: максимальная сумма

Задача 14.5. N золотых слитков с различными ценами разложены в ряд. Два игрока по очереди берут несколько слитков в левом конце ряда. Первый ход — один или два слитка, а далее за каждый ход можно взять не меньше одного слитка и не больше удвоенного числа слитков, взятых перед этим соперником. Цель каждого игрока — получить как можно б*о*льшую сумму цен взятых слитков. Программа должна вычислять, какие суммы могут обеспечить себе первый и второй игроки, как бы ни играл их соперник. Затем она должна играть за игрока, максимальная сумма которого больше, и набирать слитки с суммой не меньше этой максимальной.

Вход. В строке текста записано число N ($1 \le N \le 200$), затем N цен слитков от `1` до `200` в порядке их расположения в ряду. Числа разделены пробелами.

Примеры. *Вход*: `4 1 2 4 8`. Первый игрок не позволит набрать второму сумму больше `6`, если сделает первый ход `1`. Тогда второй игрок возьмет два слитка ценой `2` и `4`. Первый игрок заберет последний слиток и получит сумму `9`. Если первый игрок сделает первый (ошибочный!) ход `1 2`, то второй заберет слитки `4 8`. Таким образом, играя правильно, первый набирает не меньше `9`, второй — не меньше `6`, поэтому программа должна ходить первой и набирать сумму не меньше `9`. *Вход*: `4 1 1 1 1`. Каждый игрок может набрать не меньше `2`, поэтому не важно, за какого игрока будет играть программа.

Анализ задачи. В данной задаче главной является ***сумма***, которую можно набрать, исходя из позиции. Сумма зависит как от оставшихся слитков, так и от последнего хода, поэтому под позицией будем понимать пару (r, m), где r — номер первого из оставшихся слитков, m — последний ход.

Максимум суммы, которую можно набрать, начиная с позиции (r, m), обозначим через $S(r, m)$, а сумму цен всех слитков, начиная с r-го, — $T(r)$. Оба игрока стремятся увеличить свою сумму, поэтому игрок получит ***все, что не сможет забрать его соперник***. Значит, ходить нужно так, чтобы соперник мог набрать как можно меньше. Иными словами, первый ход k, где $1 \le k \le 2m$, приводящий к позиции $(r+k, k)$, нужно выбрать так, чтобы ***минимизировать*** максимум суммы, набираемой начиная с позиции $(r+k, k)$. Таким образом,

$$S(r, m) = T(r) - \min_{1 \le k \le 2m} S(r+k, k), \tag{14.1}$$

а ход — это значение k, при котором достигается $\min\limits_{1 \le k \le 2m} S(r+k, k)$.

Как видим, максимальная сумма $S(r,m)$ рекурсивно выражается с помощью максимальной суммы с бо́льшим аргументом $r+k$. На "дне" рекурсии находятся ситуации, в которых игрок может взять все оставшиеся слитки. Естественно, он должен их взять, иначе часть возьмет соперник и уменьшит сумму игрока, поэтому $S(r,m)=T(r)$ при $2m \ge N+1-r$.

Первым ходом первого игрока может быть 1 или 2, поэтому можно считать, что он начинает в позиции $(1,1)$. Вначале нужно определить, кто делает первый ход. Для этого нужно знать $S(1,1)$ и $T(1)$. Если $S(1,1) \ge T(1)/2$, программа будет ходить первой, иначе отдаст это право сопернику.

Решим задачу, не используя рекурсию. Значения $T(r)$ для $r=1, 2, \ldots, N$ вычислим сразу и запомним в массиве `T`, поскольку они понадобятся в дальнейшем. Для вычисления $S(1,1)$ используем таблицу `S` размерами $N \times L$, где L — ширина таблицы, которую определим ниже. Заполним таблицу построчно, начав с ***последней*** строки — все значения в ней равны цене последнего слитка $T(N)$. Затем в очередной строке r для каждого возможного значения m, если $2m < N+1-r$, то $S(r,m)$ вычисляется по формуле (14.1), иначе $S(r,m)=T(r)$.

Определив $S(1,1)$ и право первого хода, программа должна играть, т.е. для каждой позиции (r,m) находить значение k, при котором достигается ***минимум*** $S(r+k,k)$ по k от 1 до $2m$. Для решения этой задачи достаточно построенной таблицы `S`.

Найдем необходимую ширину L таблицы при $N \le 200$. На первом шаге ход — 1 или 2, на втором — от 1 до 4, на третьем — от 1 до 8 и т.д. С каждым ходом необходимая ширина таблицы удваивается и образуются позиции с максимальной шириной $(3,2)$, $(7,4)$, …, $(63,32)$. Затем возможные ходы m от 1 до 64 приводят в позиции вида $(63+m,m)$. Из позиции $(63+m,m)$ возможен ход k от 1 до $2m$, но при этом $63+m+k$ должно быть не больше 200. Найдем наибольшее возможное k из того, что

$$63 + m + 2m \ge 200 \text{ è } 63 + (m-1) + 2(m-1) < 200.$$

Отсюда $m=46$, $63+m=109$, т.е. $k_{max}=200-109=91$.

Итак, при $N \le 200$ достаточно таблицы шириной $L=91$. Поскольку суммарная цена слитков не больше $200 \times 200 = 40000$, элементы таблицы могут иметь тип `word` и занимать по два байта, а вся таблица уместится в пределах 40 Кбайт.

⏩ Реализуйте представленное решение.

В задаче 14.5 позиции не являются выигрышными или проигрышными. Вместо этого позиция (r,m) имеет числовую *оценку* $S(r,m)$. Очередным ходом игрок "загоняет соперника" в позицию с ***минимальной*** оценкой ***максимального*** выигрыша, которого может достичь соперник своим ходом. Такой способ оценивания позиций и выбора ходов является примером применения *метода минимакса*, предложенного в 1945 году О. Моргенштерном и Дж. фон Нейманом (см., например, [25, 28]).

Упражнения

14.1. Предположим, что в игре в сумму (см. задачу 14.1) игрок проигрывает, если своим ходом опустошает кассу. Реализовать выигрышную стратегию.

14.2. Шахматная доска имеет m горизонталей и n вертикалей; координаты ее нижней левой клетки — $(1,1)$, верхней правой — (m,n). На нижней левой клетке

стоит король. Игроки по очереди перемещают его на одно поле вправо, вверх или по диагонали вправо-вверх. Выигрывает тот, кто поставит короля в верхний правый угол доски. Программа должна получать числа m и n, определять право первого хода и выигрывать.

Вход и выход. Вначале на клавиатуре задаются количества горизонталей и вертикалей на доске (два числа от 1 до 20) и выводится начальное положение короля. Ход задается числом 1, 2 или 3 — соответственно вправо, вверх или по диагонали. После каждого хода выводится текущее положение короля (два числа).

14.3. Первый игрок задает начальное значение переменной A, второй — переменной B (чи*с*ла 0 или 1). Затем игроки заполняют 10 строк программы, по очереди выбирая номер свободной строки и записывая в нее один оператор следующего вида:

```
1. A:=1-A                    4. B:=1-B
2. if B=1 then A:=1-A        5. if A=1 then B:=1-B
3. if B=0 then A:=1-A        6. if A=0 then B:=1-B
```

После заполнения строк операторы выполняются. Если в результате $A \neq B$, выигрывает первый игрок, иначе — второй. Программа должна играть за второго игрока и выигрывать.

Вход и выход. Игрок указывает номер свободной строки (от 1 до 10) и номер оператора (от 1 до 6), ответный ход программы отображается в таком же виде. После заполнения строк вся программа вместе с начальными присваиваниями выводится на экран и имитируется; при этом отображаются выполненные операторы и значения переменных A и B.

14.4. Начиная от заданной даты, игроки по очереди называют число и месяц, каждый раз увеличивая или число в месяце, или месяц. Считается, что в феврале 28 дней. Кто назвал 31 декабря, тот: *а*) выиграл; *б*) проиграл.

14.5. *Игра в произведение*. Два игрока по очереди называют натуральные числа от 2 до 9; на это число умножается произведение всех ранее названных игроками чисел. Вначале произведение равно 1. Выигрывает игрок, после хода которого произведение станет больше заданного целого числа C. Программа должна читать C ($2 \le C \le 10^{12}$), определять право первого хода и выигрывать.

14.6. *Игра в произведение–2.* В условия предыдущей игры (*игра в произведение*) внесены два изменения. Первое: для выигрыша нужно получить произведение, лежащее в диапазоне от C до R (включительно), а произведение больше R означает проигрыш. Второе: число очередного хода должно быть взаимно простым с числом предыдущего хода. Программа должна читать два целых числа C и R ($2 \le C \le R \le 10^3$), определять право первого хода и выигрывать.

14.7. *Игра Норткотта*. Мост состоит из m дорожек по n клеток в каждой. В левой и правой клетках каждой дорожки находятся два соперничающих барана. За один ход баран проходит вперед по своей дорожке или отступает на любое число клеток, но не может пройти своего соперника или сойти с моста. Бараны ходят по очереди слева и справа. Проиграет та шеренга баранов, у которой нет хода.

14.8. *Игра "Нимбы"*. Игровое поле разделено на клетки, образующие прямоугольник. Вначале все клетки поля свободны. Игрок по имени Горз своими ходами занимает клетки по горизонтали (в строках), по имени Верт — по вертикали (в столбцах). На каждом ходу Горз выбирает любую свободную клетку и занимает ее и все клетки в строке, достижимые без пересечения вертикали, занятой Вертом. Верт аналогично занимает клетки в столбцах, не пересекая занятые горизонтали. Начинает Горз. Выигрывает тот, кто займет последнюю свободную клетку.

Программа должна получать размеры поля (количество строк и затем количество столбцов), выбирать игрока, за которого она играет, и выигрывать.

Пример. На следующем рисунке представлены два варианта игры на поле 2×3. В клетках указаны номера ходов, на которых эти клетки были заняты (нечетные обозначают ходы Горза, четные — Верта). На рис. *а* третьим ходом выиграл Горз, на рис. *б* — четвертым ходом Верт.

1	1	1
2	3	3

а

1	1	1
3	2	4

б

14.9. В ряд положены N спичек. Два игрока по очереди каждым своим ходом берут K спичек, расположенных рядом ($K \le N$). Игрок, который не может сделать очередной ход, проигрывает. Программа должна определять право первого хода и выигрывать.

Вход и выход. Вначале задаются N и K ($1 \le K \le N \le 40$). Ход обозначается минимальным из номеров забираемых спичек. На экране отображаются исходная позиция, затем ходы соперников и позиции после ходов. Позиция задается в виде двух строк — первая представляет спички и состоит из символов `1` и `0`, обозначающих спички и пустые позиции, вторая состоит из цифр `0`, `1`, …, `9`, указывающих младшие цифры номеров позиций в строке. Например, при $N=4$, $K=2$ первый игрок, забрав две средние спички, оставляет второму позицию `1001`, в которой хода нет.

14.10. *Тик-так-ту*. У первого игрока есть три знака "×", у второго — три "`0`". На первом этапе игры игроки по очереди ставят по три своих знака на клетки поля 3×3. На втором этапе игроки по очереди перемещают свои знаки на одну клетку по вертикали или горизонтали. Выигрывает тот, кто первым займет тремя своими знаками одну из горизонталей, вертикалей или диагоналей. Повторение позиции рассматривается как ничейный результат. Программа должна играть крестиками и выигрывать.