Разбор задач Третьей Интернет-олимпиады

Введение

В базовой номинации Третьей Интернет-олимпиады сезона 2008-2009 участникам было предложено для решения восемь задач. В олимпиаде приняло участие 136 команд, из них 117 решили хотя бы одну задачу.

Наиболее простой оказалась задача «F. Счастливый билетик — 3» — ее решили 91 команда. Наиболее сложной — задача «B. Четырехугольник» — ее решили 34 команды.

Условия задач, результаты олимпиады, тесты и решения жюри можно найти на сайте интернет-олимпиад `http://neerc.ifmo.ru/school/io`.

Задача A. Морской бой — 3

Авторы задачи: Александр Торопов, Федор Царев
Автор разбора: Антон Феськов

Первая идея, которая приходит в голову при решении этой задачи — перебрать все возможные варианты расположения корабля. Попробуем ее формализовать.

Заметим, что положение корабля однозначно задается координатами левого верхнего его конца и направлением, которое может быть горизонтальным и вертикальным.

Таким образом, если перебрать все возможные положения левого верхнего конца корабля и для каждого — все направления, то будут рассмотрены все возможные положения корабля. После этого останется только для каждого положения проверить, не содержит ли такой корабль клеток, по которым были произведены выстрелы. Если таких клеток нет, то там может находиться корабль, иначе — не может.

Приведем программную реализацию этого подхода.

```
ans := 0;
for i := 1 to n do begin
    for j := 1 to m do begin
        good := true;
        for l := 0 to k - 1 do begin
            if (i + l > n) or (D[i + l][j] = '#') then begin
                good := false;
                break;
            end;
        end;
        if (good) then inc(ans);
        if (k > 1) then begin
            good := true;
            for l := 0 to k - 1 do begin
                if (j + l > m) or (D[i][j + l] = '#') then begin
                    good := false;
                    break;
                end;
            end;
            if (good) then inc(ans);
        end;
    end;
end;
writeln(ans);
```

Здесь матрица `D[i][j]` — матрица, заданная во входном файле, `ans` — ответ на задачу, `good` — переменная типа `boolean`, в которой хранится, можно ли поставить корабль текущим способом.

Отметим что при $k = 1$ корабль $1 \times k$ совпадает с кораблем $k \times 1$, поэтому если убрать строку `if (k > 1) then ...`, то решение (только при $k = 1$) будет работать неправильно и выдавать вдвое больший ответ.

Приведенное выше решение перебирает все клетки матрицы и для каждой совершает $O(k)$ операций. Таким образом время работы алгоритма $O(mnk)$, что при данных ограничениях легко укладывается в ограничения по времени.

Задача B. Четырехугольник

Авторы задачи: Александр Торопов, Федор Царев
Автор разбора: Антон Феськов

Договоримся обозначать многоугольник перечислением его вершин в порядке обхода по или против часовой стрелке. Тогда четырехугольник $ABCD$ совпадает с четырехугольниками $BCDA$ и $BADC$, но не совпадает с $BACD$ и $DCAB$.

Допустим, что мы каким-то образом узнали, в каком порядке должны следовать вершины во входном файле, чтобы они образовывали квадрат. Осталось проверить, является ли данный четырехугольник $ABCD$ квадратом. Это можно делать различными способами. Один из них — проверить, что все его стороны попарно равны ($|AB| = |BC| = |CD| = |DA|$, то есть $ABCD$ — ромб), и что диагонали также равны ($|AC| = |BD|$). Заметим, что не обязательно сравнивать непосредственно длины сторон, достаточно сравнить на равенство их квадраты ($|AB| = |BC| \Leftrightarrow |AB|^2 = |BC|^2$). Это замечание удобно при задании четырехугольника координатами вершин, так как при таком подходе все вычисления производятся в целых числах и не происходит потерь точности, связанных с использованием вещественной арифметики.

Выясним теперь, в каком порядке должны идти вершины, заданные во входном файле. Предположим, что заданный набор точек образует квадрат. Тогда любые три его вершины образуют прямоугольный треугольник. Пусть это треугольник ABC с прямым углом при вершине C. Тогда очевидно, что искомый квадрат — это четырехугольник $ADBC$. Но как по трем точкам понять, при какой из них прямой угол? Заметим, в прямоугольном треугольнике

$$\begin{cases} |AB| > |AC| \\ |AB| > |BC| \end{cases} \Leftrightarrow \begin{cases} |AB|^2 > |AC|^2 \\ |AB|^2 > |BC|^2 \end{cases}$$

Отсюда легко получаем алгоритм: рассматриваем первые три точки из входного файла, находим две наиболее удаленные из них, обозначаем их как A и B, а остальные две — как C и D. После этого остается проверить, является ли четырехугольник $ADBC$ квадратом — это описано выше.

Так как в описанном алгоритме часто используется вычисление квадрата расстояния между двумя вершинами, то эту операцию разумно выделить в функцию.

```
function sqLen(i, j : longint) : longint;
begin
    result := (x[i] - x[j]) * (x[i] - x[j]) + (y[i] - y[j]) * (y[i] - y[j]);
end;
```

Здесь (`x[i], y[i]`) — координаты `i`-й точки во входном файле.

Основная часть программы:

```
a := 1;
b := 3;
for i := 1 to 2 do begin
    if (sqLen(i, i + 1) > sqLen(a, b)) then begin
        a := i;
```

```
            b := i + 1;
        end;
end;
c := 6 - a - b;
d := 4;

if (sqLen(c, a) = sqLen(a, d)) and
   (sqLen(c, a) = sqLen(d, b)) and
   (sqLen(c, a) = sqLen(b, c)) and
   (sqLen(a, b) = sqLen(c, d)) then
    writeln('YES')
else
    writeln('NO');
```

Здесь `a`, `b`, `c` и `d` — переменные целого типа, соответствующие обозначению вершин в описании алгоритма.

Формула `c := 6 - a - b` вычисляет число, отсутствующее среди `a` и `b`. Действительно, если `a`, `b` и `c` - различные натуральные числа от 1 до 3, то `c = (1 + 2 + 3) - a - b`.

Время работы программы и используемая память есть $O(1)$.

Задача C. Робот

Автор задачи: Федор Царев
Автор разбора: Федор Царев

Для решения этой задачи можно применить несколько подходов. Все они основаны на моделировании выполнения программы роботом, а отличаются способом отслеживания клеток, которые робот посетил более одного раза.

Первый из методов решения основан на следующем наблюдении. Робот не может уйти ни в одну из сторон более чем на тысячу клеток, так как длина программы не превосходит тысячу инструкций, Поэтому все возможные клетки, в которых может находиться робот в процессе выполнения программы образуют квадрат со стороной 2001, центральная клетка которого соответствует начальному положению робота.

Можно создать двумерный массив целочисленного типа `field` (`field: array[1..2001, 1..2001] of integer`), который будет соответствовать полю. Каждый элемент этого массива соответствует некоторой клетке поля, а его значением будет число посещений роботом этой клетки.

Для вычисления ответа после окончания моделирования работы программы необходимо найти число элементов массива `field`, значения которых больше единицы.

Приведем фрагмент программы, реализующий этот подход.

```
readln(s);
r := 1001;
c := 1001;
n := length(s);
for i := 1 to n do begin
  inc(field[r][c]);
  ch := s[i];
  if (ch = 'L') then begin
    c := c - 1;
  end else if (ch = 'R') then begin
    c := c + 1;
  end else if (ch = 'U') then begin
    r := r - 1;
```

```
  end else if (ch = 'D') then begin
    r := r + 1;
  end;
end;
inc(field[r][c]);
ans := 0;
for i := 1 to 2001 do begin
  for j := 1 to 2001 do begin
    if (field[i][j] > 1) then begin
      inc(ans);
    end;
  end;
end;
writeln(ans);
```

Второй подход основан на идее, схожей с идеей решения задачи о нахождении числа различных элементов в массиве. Для этого создается массив посещенных клеток (каждая клетка, как и раньше задается двумя числами — ее координатами). После этого указанный массив упорядочиваются по возрастанию первых координат клеток (при их равенстве — по возрастанию вторых координат).

Приведем фрагмент программы, реализующий этот этап решения.

```
readln(s);
r := 0;
c := 0;
n := length(s);
p := 1;
for i := 1 to n do begin
  a[p][1] := r;
  a[p][2] := c;
  inc(p);
  ch := s[i];
  if (ch = 'L') then begin
    c := c - 1;
  end else if (ch = 'R') then begin
    c := c + 1;
  end else if (ch = 'U') then begin
    r := r - 1;
  end else if (ch = 'D') then begin
    r := r + 1;
  end;
end;
a[p][1] := r;
a[p][2] := c;
inc(p);

for i := 1 to p do begin
  for j := i + 1 to p do begin
    if (a[j][1] < a[i][1]) or
       ((a[j][1] = a[i][1]) and (a[j][2] < a[i][2])) then begin
      t := a[i][1];
      a[i][1] := a[j][1];
      a[j][1] := t;
```

```
        t := a[i][2];
        a[i][2] := a[j][2];
        a[j][2] := t;
      end;
    end;
end;
```

Здесь двумерный массив `a` хранит координаты клеток, посещенных роботом — элементы `a[i][1]` хранят номера строк, в которых находятся эти клетки, а `a[i][2]` — номера столбцов. Также отметим, что здесь выбор начального положения робота не важен, так как не происходит обращения к элементам массива с номерами, равными координатам клетки.

Заметим, что если робот k раз посетил некоторую клетку, то в этом массиве координаты этой клетки будут присутствовать также ровно k раз, причем они будут занимать несколько подряд идущих элементов. Используя это, можно найти ответ на задачу – для этого необходимо выделить такие отрезки одинаковых клеток, и найти число отрезков, длина которых превышает единицу.

Приведем программную реализацию этого этапа.

```
ans := 0;
i := 1;
while (i < p) do begin
  j := i;
  while (j < p) and (a[i][1] = a[j][1]) and (a[i][2] = a[j][2]) do begin
    inc(j);
  end;
  if (j - i > 1) then begin
    inc(ans);
  end;
  i := j;
end;
```

Здесь переменная `i` соответствует началу очередного отрезка, а переменная `j` используется для определения его конца. На каждой итерации она изначально равна `i`, а затем увеличивается до тех пор, пока не будет найден элемент, содержащий клетку, не равную `a[i]`. Его обнаружение соответсвует тому, что найдено начало нового отрезка. В этом случае длина текущего отрезка сравнивается с единицей, и, при необходимости, увеличивается значение переменной `ans`, используемой для вычисления ответа на задачу.

Задача D. Последовательность

Автор задачи: Сергей Поромов
Автор разбора: Федор Царев

Во-первых, заметим что так как по условию $n \leq 10^5$, то все элементы последовательности могут быть достаточно быстро вычислены. Это можно сделать, например, так.

```
read(x, y, z, w);
read(n, k);
a[1] := w;
for i := 2 to n do begin
  a[i] := (x * a[i - 1] + y) mod z;
end;
```

После выполнения этого фрагмента программы массив `a` будет заполнен теми числами, которые Вася записал на полях своей тетради. Теперь заметим, что последнее действие, которое выполняется

при вычислении очередного элемента `a[i]` (для `i > 1`), — это взятие остатка от деления на z. Это означает, что все элементы массива `a`, начиная со второго и до последнего, находятся в промежутке от 0 до $z-1$. Из этого и из того, что w по условию не превосходит 10000, следует, что все элементы массива `a` находятся в промежутке от 0 до 10000.

Воспользуемся идеей, сходной с идеей сортировки «подсчетом» [?] — для каждого из чисел от 0 до 10000 найдем, сколько раз оно встречается в массиве `a`.

```
for i := 1 to n do begin
  inc(cnt[a[i]]);
end;
```

После выполнения этого фрагмента программы в каждом элементе `cnt[i]` массива `cnt` будет содержаться количество вхождений числа `i` в массив `a`. Для того, чтобы теперь найти ответ на задачу, будем последовательно от 0 в возрастающем порядке перебирать натуральные числа (до 10000) и для каждого из них проверять, является оно k-ым в порядке неубывания или нет.

Рассмотрим некоторое целое число b ($0 \le b \le 10000$). Рассмотрим все числа b, такие что в массиве `a` количество чисел, не превосходящих b, больше либо равно k. Минимальное из них является ответом на задачу. Это же условие можно записать следующим образом.

$$\sum_{i=0}^{b} \texttt{cnt[i]} \le k, b \to \min$$

Непосредственная программная реализация этой формулы может оказаться недостаточно эффективной для решения задачи, так как будет требовать порядка $10000^2 = 10^8$ операций.

Заметим, что соответствующие суммы для b и для $(b+1)$ отличаются на одно слагаемое. Поэтому пересчет этой суммы можно производить за $O(1)$. Приведем фрагмент программы, реализующий этот этап решения задачи.

```
s := 0;
for b := 0 to 10000 do begin
  s := s + cnt[b];
  if (s >= k) then begin
    writeln(b);
    break;
  end;
end;
```

Заметим, что при такой реализации автоматически получается то, что будет найдено минимальное b. Это обеспечивается тем, что как только требуемое число b находится, так сразу происходит выход из цикла с помощью оператора `break`.

Задача E. Клавиша Shift

Автор задачи: Владимир Ульянцев
Автор разбора: Федор Царев

Заметим, что все символы, из которых состоит текст во входной файле, можно разбить на три типа:

1. Символы, для набора которых клавиша *Shift* обязательно должна быть нажата;
2. Символы, для набора которых клавиша *Shift* обязательно должна быть не нажата;
3. Символы, для набора которых клавиша *Shift* может быть нажата, а может быть и не нажата.

В первую группу входят заглавные буквы, во вторую — строчные буквы, а в третью — пробел и перевод строки.

Если в тексте подряд встречаются два символа, для набора одного из которых клавиша *Shift* должна обязательно нажата, а для набора другого — обязательно не нажата, то при наборе этой части текста клавишу *Shift* придется либо нажать (если второй из символов является заглавной буквой), либо отпустить (если второй из символов является строчной буквой). В других случаях нажимать или отпускать клавишу *Shift* не требуется.

На основе этого наблюдения можно сформулировать алгоритм решения задачи. Необходимо просматривать символы текста и хранить состояние клавиши *Shift* (нажата она или нет). Если клавиша *Shift* нажата, а очередной символ является строчной буквой, то ее необходимо отпустить. Если же клавиша *Shift* не нажата, а очередной символ является заглавной буквой, то ее необходимо нажать, при этом увеличив счетчик нажатий.

Приведем программную реализацию этого алгоритма.

```
ans := 0;
shift := false;
while (not eof) do begin
  read(ch);
  if (('A' <= ch) and (ch <= 'Z') and (not shift)) then begin
    shift := true;
    ans := ans + 1;
  end;
  if (('a' <= ch) and (ch <= 'z')) then begin
    shift := false;
  end;
end;
writeln(ans);
```

Задача F. Счастливый билетик — 3

Автор задачи: Федор Царев
Автор разбора: Сергей Мельников

Решение этой задачи можно разбить на три части, соответствующих проверке каждого из критериев, указанных в условии задачи.

Рассмотрим первый критерий — сумма цифр первой половины билетика равна сумме цифр второй половины ($\sum_{i=1}^{n} a_i = \sum_{i=n+1}^{2n} a_i$). Для его проверки необходимо перебрать все цифры и вычислить два числа: $A = \sum_{i=1}^{n} a_i$ и $B = \sum_{i=n+1}^{2n} a_i$.

Например, если цифры билетика хранятся в массиве a_i, то эти числа можно вычислить с помощью такого фрагмента программы.

```
A := 0;
for i := 1 to n do begin
  A := A + a[i];
end;
B := 0;
for i := n + 1 to 2 * n do begin
  B := B + a[i];
end;
```

После этого необходимо проверить равенство $A = B$.

Второй критерий — сумма цифр на нечетных позициях равна сумме цифр на четных позициях ($\sum_{i=1}^{n} a_{2i} = \sum_{i=1}^{n} a_{2i-1}$). Для его проверки аналогичным образом вычислим числа $C = \sum_{i=1}^{n} a_{2i}$ и $D = \sum_{i=1}^{n} a_{2i-1}$ и проверим их равенство.

```
C := 0;
D := 0;
for i := 1 to n do begin
  C := C + a[2 * i];
  D := D + a[2 * i - 1];
end;
```

Третий критерий — сумма четных цифр билетика равна сумме его нечетных цифр. Для его проверки вычислим сумму четных цифр и сумму нечетных цифр, а потом сравним их.

```
even := 0;
odd := 0;
for i := 1 to 2 * n do begin
  if a[i] mod 2 = 1 then begin
    odd := odd + a[i];
  end else begin
    even := even + a[i];
  end;
end;
```

В этом фрагменте программы переменная `even` хранит сумму четных цифр билетика, переменная `odd` — сумму нечетных цифр.

После этого выводим ответ.

```
if (A = B) or (C = D) or (even = odd) then begin
  writeln('YES');
end else begin
  writeln('NO');
end;
```

Задача G. Трехдиагональная матрица

Автор задачи: Федор Царев
Автор разбора: Антон Ахи

В данной задаче необходимо сделать именно то, что написано в условии. А именно, необходимо просмотреть все элементы матрицы A и для каждого $A_{ij} \neq 0$ проверить верно ли, что $|i - j| \leq 1$. Если хотя бы для одной ненулевой клетки матрицы данное условие не выполняется, то матрица, по определению, не является трехдиагональной. Во всех остальных случаях матрица трехдиагональна.

Фрагмент решения данной задачи на языке *Pascal*:

```
good := true;
for i := 1 to n do begin
  for j := 1 to n do begin
    if (A[i, j] <> 0 and abs(i - j) > 1) then begin
      good := false;
    end;
  end;
```

```
end;

if (good) then begin
  writeln('YES');
end else begin
  writeln('NO');
end;
```

В данном фрагменте переменная `good` отвечает за то, является ли матрица трехдиагональной, массив `A` — двумерный массив, содержащий данную матрицу.

Задача H. Две дуги

Автор задачи: Федор Царев
Автор разбора: Максим Буздалов

Сначала решим вспомогательную задачу. А именно, для данной дуги и данной точки на окружности определим, лежит ли точка на дуге.

Пусть дуга начинается в точке с градусной мерой S и заканчивается в точке с градусной мерой F, а данная точка имеет градусную меру X. Рассмотрим два случая: $S \le F$ и $S > F$.

В первом случае дуга содержит только точки с градусной мерой из отрезка $[S; F]$, поэтому данная точка принадлежит дуге тогда и только тогда, когда $S \le X$ *и* $X \le F$.

Во втором случае дуга содержит точку с градусной мерой 0, так как только в этой точке значение угла скачкообразно уменьшается. Так как во всех остальных точках окружности функция угла строго возрастает и непрерывна, то дуга содержит те и только те точки, градусные меры которых принадлежат следующему интервалу: $[S; 360) \cup [0; F]$. Поэтому данная точка принадлежит дуге тогда и только тогда, когда $S \le X$ *или* $X \le F$.

Отсюда следует первый способ решения. Так как значений возможных градусных мер концов дуг, немного, то их возможно перебрать и для каждого значения проверить, верно ли, что соответствующая точка лежит на обеих дугах одновременно. Следующая программа является одной из возможных реализаций такого решения:

```
uses sysutils;
var
    s1, f1, s2, f2, i: integer;
    ans, v1, v2: boolean;
begin
    reset(input, 'twoarcs.in');
    rewrite(output, 'twoarcs.out');
    readln(s1, f1);
    readln(s2, f2);
    ans := false;
    for i := 0 to 359 do begin
        if s1 <= f1 then begin
            v1 := (s1 <= i) and (i <= f1);
        end else begin
            v1 := (s1 <= i) or (i <= f1);
        end;
        if s2 <= f2 then begin
            v2 := (s2 <= i) and (i <= f2);
        end else begin
            v2 := (s2 <= i) or (i <= f2);
        end;
```

```
        ans := ans or (v1 and v2);
    end;
    if ans then begin
        writeln('YES');
    end else begin
        writeln('NO');
    end;
end.
```

Это решение работает за линейное время относительно количества возможных значений градусных мер. Для ограничений, данных в условии (где это количество равно 360), оно работает достаточно быстро.

Однако можно заметить, что дуги имеют хотя общую точку, когда верно хотя бы одно из четырех утверждений:

- первая дуга содержит начало второй дуги;
- первая дуга содержит конец второй дуги;
- вторая дуга содержит начало первой дуги;
- вторая дуга содержит конец первой дуги.

Это утверждение позволяет реализовать второй способ решения, работающее за константное время. Его возможная реализация приведена ниже:

```
uses sysutils;
var
    s1, f1, s2, f2, i: integer;
    v1, v2, v3, v4: boolean;
begin
    reset(input, 'twoarcs.in');
    rewrite(output, 'twoarcs.out');
    readln(s1, f1);
    readln(s2, f2);
    if s1 <= f1 then begin
        v1 := (s1 <= s2) and (s2 <= f1);
        v2 := (s1 <= f2) and (f2 <= f1);
    end else begin
        v1 := (s1 <= s2) or (s2 <= f1);
        v2 := (s1 <= f2) or (f2 <= f1);
    end;
    if s2 <= f2 then begin
        v3 := (s2 <= s1) and (s1 <= f2);
        v4 := (s2 <= f1) and (f1 <= f2);
    end else begin
        v3 := (s2 <= s1) or (s1 <= f2);
        v4 := (s2 <= f1) or (f1 <= f2);
    end;
    if v1 or v2 or v3 or v4 then begin
        writeln('YES');
    end else begin
        writeln('NO');
    end;
end.
```

Отметим, что второе решение работает, после незначительных модификаций, для достаточно широкого диапазона входных данных, например, для вещественных значений градусных мер, чего нельзя сказать о первом решении.

Список литературы

[1] *Кормен Т., Лейзерсон Ч., Ривест Р.* Алгоритмы: Построение и Анализ. — М.: МЦНМО, 1999.