Разбор задач Первой Интернет-олимпиады

Введение

В базовой номинации Первой Интернет-олимпиады сезона 2008-2009 участникам было предложено для решения 9 задач. В олимпиаде приняло участие 211 команд, из них 195 решили хотя бы одну задачу.

Наиболее простой оказалась задача «F. Сбор черники» — ее решили 180 команд. Наиболее сложной — задача «D. Отрезок и окружности» — ее решили 17 команд.

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

Задача A. Антипалиндром

Автор задачи: Владимир Ульянцев
Автор разбора: Федор Царев

Заметим, что число подстрок строки длины n равно $1+2+\ldots+n = \frac{n\,(n+1)}{2}$. Это объясняется тем, что строка длины n содержит одну подстроку длины n, две подстроки длины $n-1$, ..., n подстрок длины 1. Таким образом, время работы решения, основанного на переборе всех подстрок будет пропорционально n^2 или требовать большего времени. Такое решение не укладывается в ограничения по времени.

Для того, чтобы получить более быстрое решение, необходимо более детальное изучение свойств подстрок, не являющихся палиндромами. Во-первых, сама строка s может не быть палиндромом. Тогда она и является ответом для задачи. Во-вторых, может оказаться так, что все подстроки строки s являются палиндромами. Такое может быть только в том случае, если все символы s равны между собой.

Докажем теперь, что если ни один из этих двух случаев не реализуется, то строка, получаемая из s отбрасыванием первого символа не является палиндромом. Итак, строка s является палиндромом, но не все ее символы равны между собой. Обозначим как t строку, получающуюся из s отбрасыванием первого символа. Докажем с помощью метода «от противного», что t не является палиндромом.

Для этого предположим, что $t = s_2 s_3 \ldots s_n$ является палиндромом. Из этого следует, что ее первый и последний символ равны между собой, то есть $s_2 = s_n$. Из того, что s равно палиндромом следует, что $s_1 = s_n$. Таким образом, получаем, что $s_1 = s_2 = s_n$. Далее, из того, что s является палиндромом так же следует, что $s_{n-1} = s_2$. Значит, $s_1 = s_2 = s_n = s_{n-1}$. Из того, что t является палиндромом следует, что $s_3 = s_{n-1}$. Значит, цепочка равенств расширяется далее: $s_1 = s_2 = s_n = s_{n-1} = s_3$. Продолжая это рассуждение, получаем, что все символы строки s равны между собой, что противоречит предположению. Значит, доказано, что строка t не является палиндромом.

Это рассуждение приводит нас к следующему решению. Возможны три случая:

- все символы строки s равны между собой — тогда ответ `NO SOLUTION`;
- строка s не является палиндромом — тогда ответом является сама строка s;
- иначе ответом является строка t, получающаяся из s отбрасыванием первого символа.

Приведем фрагмент программы, реализующий это решение.

```
read(s);
n := length(s);

all := false;
```

```
pali := false;

for i := 2 to n do begin
  if (s[i] <> s[i - 1]) then begin
    all := true;
  end;
  if (s[i] <> s[n - i + 1]) then begin
    pali := true;
  end;
end;

if (not all) then begin
  writeln('NO SOLUTION')
end else if (not pali) then begin
  for i := 1 to n - 1 do begin
    write(s[i]);
  end;
end else begin
  writeln(s);
end;
```

Задача B. Сближение с целью

Автор задачи: Федор Царев
Автор разбора: Антон Феськов

Первая идея при решении этой задачи — вычислить координаты цели через время t. Если мы будем считать, что цель изначально находится в этой точке и не двигается, то ответ не изменится. Таким образом считаем, что цель находится в точке $(x_1, y_1) = (x_0 + t \times V_x, y_0 + t \times V_y)$.

Заметим, что за время t РК-2008-2009 может попасть в любую точку на расстоянии не более $V \times t$ от его начального положения (от начала координат), значит геометрическое место точек, в которые РК-2008-2009 может попасть за время t есть круг радиуса $V \times t$ с центром в начале координат.

Множество точек, находящихся на расстоянии d от положения цели (а именно в одной из таких точек РК-2008-2009 должен оказаться), есть окружность радиуса d с центром в точке (x_1, y_1). Если эти две фигуры пересекаются, то крейсер может выполнить поставленную задачу, иначе нет. Задача сведена к вопросу о том, пересекается ли данный круг и окружность.

Необходимое условие пересечения: расстояние между центрами фигур не превосходит суммы их радиусов.

$$\sqrt{x_1^2 + y_1^2} \le d + V \times t \Leftrightarrow x_1^2 + y_1^2 \le (d + V \times t)^2$$

Однако, это условие не является достаточным: если круг лежит целиком внутри окружности, то эти фигуры не пересекаются. Отрицание этого условия можно сформулировать так: сумма расстояния между центрами фигур и радиуса круга не меньше радиуса окружности.

$$\sqrt{x_1^2 + y_1^2} + V \times t \ge d \Leftrightarrow \left[\begin{array}{l} x_1^2 + y_1^2 \ge (d - V \times t)^2 \\ d \le V \times t \end{array} \right.$$

Доказательство того, что были рассмотрены все случаи, оставляется читателю.

Отметим, что все условия, которые нужно проверять, записаны в виде, удобном для вычисления в целых числах, чтобы избежать потерь точности при использовании вещественной арифметики. Однако при данных ограничениях при использовании менее чем 64-битного типа приводило к переполнению. Таким образом, в языке *Delphi* необходимо было использовать тип `int64`, в *Java* — `long`, а в *C++* — `long long`.

Приведем программную реализацию этого подхода.

```
read(x0, y0, vx, vy, v, t, d);
x1 := x0 + vx * t;
y1 := y0 + vy * t;
if (x1 * x1 + y1 * y1 <= (d + v * t) * (d + v * t)) and
  ((x1 * x1 + y1 * y1 >= (d - v * t) * (d - v * t)) or (d <= v * t)) then
  writeln('YES')
else
  writeln('NO');
```

Здесь `x0`, `y0`, `vx`, `vy`, `v`, `t` и `d` — переменные типа `int64`, соответствующие величинам x_0, y_0, V_x, V_y, V, t и d из условия задачи.

Время работы алгоритма — $O(1)$.

Задача C. Морской бой — 2

Автор задачи: Александр Торопов
Автор разбора: Александр Торопов

Для решения задачи переберем все клетки поля, и для каждой из них проверим, можно ли в нее поставить корабль. Рассмотрим клетку с координатами (i, j), находящуюся на пересечении i-ой строки и j-ого столбца. Поставить однопалубный корабль в эту клетку можно только в том случае, если она и ее соседи не заняты кораблями.

Соседями клетки являются две соседние клетки по вертикали (номер строки отличается на единицу) и две соседние клетки по горизонтали (номер столбца отличается на единицу). Таким образом, координаты этих клеток есть: $(i-1, j)$, $(i+1, j)$, $(i, j-1)$ и $(i, j+1)$. Отметим, что соседей у клетки не всегда четыре — например, у угловой клетки всего два соседа. Приведем программную реализацию этого подхода.

Пример программы на Delphi:

```
readln(n, m);
for i := 1 to n do begin
  readln(s[i]);
end;

ans := 0;

for i := 1 to n do begin
  for j := 1 to m do begin
    if (s[i][j] = '*') then begin
          continue;
        end;
        t := 0;
        if ((i = 1) or (s[i - 1][j] = '.')) then begin
      inc(t);
        end;
        if ((i = n) or (s[i + 1][j] = '.')) then begin
          inc(t);
        end;
        if ((j = 1) or (s[i][j - 1] = '.')) then begin
      inc(t);
    end;
```

```
        if ((j = m) or (s[i][j + 1] = '.')) then begin
      inc(t);
    end;
        if (t = 4) then begin
          inc(ans);
        end;
  end;
end;

writeln(ans);
```

Задача D. Отрезок и окружности

Автор задачи: Владимир Ульянцев
Автор разбора: Антон Ахи

Для начала рассмотрим более простую задачу. Пусть наш отрезок таков, что при движении от одного края к другому, расстояние до центра координат возрастает. Для такого отрезка ответ очевиден — это количество целых чисел между расстояниями от центра координат до обоих концов отрезка.

Теперь необходимо свести данную задачу к рассмотренной выше. Для этого необходимо найти на отрезке точку, ближайшую к началу координат. Таким образом исходный отрезок разбивается на два новых, для которых выполнено условие из простой задачи. Также следует рассмотреть крайний случай, а именно, если ближайшая к $(0;0)$ точка находится на целом расстоянии от начала координат. В этом случае мы посчитаем это пересечение дважды, поэтому необходимо уменьшить ответ на единицу.

Стоит заметить, что находить саму ближайшую точку нет необходимости. Достаточно найти лишь расстояние до нее.

```
Const
 eps = 1e-9;
Var
 x1, y1, x2, y2 : longint;
 a, b, d : double;
 ans : longint;

begin
 reset(input, 'circles.in');
 rewrite(output, 'circles.out');
 read(x1, y1, x2, y2);
 d := abs((y2 * x1 - x2 * y1) /
      sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2)));
 a := sqrt(x1 * x1 + y1 * y1);
 b := sqrt(x2 * x2 + y2 * y2);
 ans := 0;
 if (x1 * x1 + y1 * y1 - x2 * x2 - y2 * y2
  + (x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2) < 0) or
   (-x1 * x1 - y1 * y1 + x2 * x2 + y2 * y2
  + (x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2) < 0) then
 begin
  ans := ans + trunc(max(a, b) + eps) - trunc(min(a, b) - eps);
 end else
```

```
 begin
  ans := ans + trunc(a + eps) - trunc(d - eps);
  ans := ans + trunc(b + eps) - trunc(d - eps);
  if (abs(round(d) - d) < eps) and (abs(d) > eps) then
  begin
   dec(ans);
  end;
 end;
 writeln(ans);
 close(Input);
 close(Output);
end.
```

Здесь `d` — расстояние до прямой, которая содержит отрезок, `a` и `b` — расстояния до концов отрезка.

Задача E. Двойственная ломаная

Автор задачи: Федор Царев
Автор разбора: Владимир Ульянцев

Найдем для заданной замкнутой ломаной P двойственную для неё ломаную k-ого порядка. Для этого достаточно k раз подряд найти ломаную, двойственную для ломаной, полученной на предыдущем шаге (изначально P). Затем посчитаем ответ — длину получившейся ломаной.

Приведем программную реализацию.

```
var
    i, n, j, k : longint;
    x, y, xt, yt : array [0..100] of extended;
    ans : extended;

begin
    reset(input, 'dual.in');
    rewrite(output, 'dual.out');

    read(n);
    for i := 0 to n - 1 do begin
        read(x[i], y[i]);
    end;
    read(k);

    for j := 1 to k do begin
        for i := 0 to n - 1 do begin
            xt[i] := (x[i] + x[(i + 1) mod n]) / 2;
            yt[i] := (y[i] + y[(i + 1) mod n]) / 2;
        end;
        x := xt;
        y := yt;
    end;

    ans := 0;
    for i := 0 to n - 1 do begin
        ans := ans + sqrt(sqr(x[(i + 1) mod n] - x[i])
```

```
                + sqr(y[(i + 1) mod n] - y[i]));
    end;

    write(ans);
end.
```

Данный алгоритм k раз выполняет нахождение двойственной ломаной, итого суммарное время работы программы составляет $O(kn)$, что позволяет уложиться в ограничения по времени при указанных в условии ограничениях на n и k.

Задача F. Сбор черники

Автор задачи: Александр Торопов
Автор разбора: Сергей Поромов

В этой задаче необходимо найти три рядом расположенных куста таких, что сумма количества ягод на них максимальна. При этом необходимо учитывать то, что грядка круглая, и поэтому можно взять, например, последний и два первых куста. Заметим также, что число ягод невелико и для его хранения можно использовать 32-битный целочисленный тип данных (`longint` в языке *Pascal*, `int` в языке *C*, `int` в языке *Java*).

Решение состоит в том, чтоб последовательно перебрать первый куст из тех, что мы берем, посчитать сумму количества ягод на нем и следующих двух. Из таких сумм необходимо найти максимум.

Этот алгоритм работает за линейное от n время. Один из способов обрабатывать то, что грядка является круглой, — использовать массив длиной $n+2$ и в последние два элемента записать первые два.

Приведем программную реализацию этого алгоритма.

```
for i := 1 to n do
  read(a[i]);
a[n + 1] := a[1];
a[n + 2] := a[2];
ans := 0;
for i := 1 to n do begin
  if (a[i] + a[i + 1] + a[i + 2] > ans) then begin
    ans = a[i] + a[i + 1] + a[i + 2];
  end;
end;
write(ans);
```

Задача G. Индикатор

Автор задачи: Федор Царев
Автор разбора: Федор Царев

Решение этой задачи, в целом, будет таким: в обоих случаях цифры будут приписываться к числу по одной, начиная со старших разрядов.

Для получения наименьшего числа необходимо обеспечить, чтобы в старших разрядах стояли как можно меньшие цифры (при этом необходимо следить за тем, что старшей цифрой не может быть ноль). Для получения наибольшего числа необходимо обеспечить, чтобы в старших разрядах стояли большие цифры.

Опишем более подробно алгоритм построения наименьшего числа. В каждый момент времени будут поддерживаться следующие величины: уже полученная часть числа num (некоторые его старшие разряды), число k полосок, которые должны стать черными в оставшихся разрядах, и число n цифр, которые необходимо дописать.

За один шаг к числу будет приписываться одна цифра. Для этого необходимо перебрать очередную цифру d. Обозначим число черных полосок в изображении цифры d как `black[d]`. После приписывания цифры d к существующему числу останется дописать к нему $n-1$ цифру, а число полосок, которые должны быть черными в оставшихся разрядах будет равно $k-$`black[d]`.

Тут необходимо проверить, хватит ли черных полосок на то, чтобы написать оставшиеся цифры. Например, если исходно $n=2, k=8$, а в качестве первой цифры d мы пробуем поставить восьмерку, то на оставшуюся одну цифру остается одна полоска. Так как ни одной цифры, изображаемой с помощью одной полоски, не существует, то восьмерку в качестве старшей цифры поставить нельзя. С другой стороны, если $n=2, k=10$, а в качестве первой цифры выбрать единицу, то на оставшуюся цифру остается восемь полосок. Таких цифр также не существует, поэтому единица первой цифрой быть не может.

Поэтому возникает задача проверки того, может ли быть составлено число, содержащее n цифр и k черных полосок. Для того, чтобы решить эту подзадачу, заметим, что в изображении одной цифры могут участвовать две, три, четыре, пять, шесть или семь полосок. Наиболее важно тут то, что эти числа заполняют собой отрезок целых чисел от двух до семи. Значит, n чисел из этого множества могут давать в сумме любое число от $2 \cdot n$ до $7 \cdot n$. Таким образом, для решения указанной подзадачи достаточно проверить двойное неравенство $2 \cdot n \le k \le 7 \cdot n$.

Приведем фрагмент программы, реализующий описанный алгоритм.

```
min := '';
ck := k;
cn := n;
for i := 1 to n do begin
  st := 0;
  if (i = 1) then begin
    st := 1;
  end;
  for j := st to 9 do begin
    nk := ck - cnt[j];
    if (nk >= 2 * (cn - 1)) and (nk <= 7 * (cn - 1)) then begin
      min := min + chr(ord('0') + j);
      ck := nk;
      cn := cn - 1;
      break;
    end;
  end;
end;
```

Здесь переменная `st` обозначает цифру, с которой начинается перебор — для старшего разряда `st = 1`, для всех остальных — `st = 0`.

В случае построения максимального числа алгоритм такой же, только вместо перебора цифр по возрастанию (начиная от меньших) цифры перебираются по убыванию.

```
max := '';
ck := k;
cn := n;
for i := 1 to n do begin
  st := 0;
  if (i = 1) then begin
    st := 1;
```

```
  end;
  for j := 9 downto st do begin
    nk := ck - cnt[j];
    if (nk >= 2 * (cn - 1)) and (nk <= 7 * (cn - 1)) then begin
      max := max + chr(ord('0') + j);
      ck := nk;
      cn := cn - 1;
      break;
    end;
  end;
end;
```

Задача H. K-перестановки

Автор задачи: Сергей Мельников
Автор разбора: Антон Феськов

Заметим, что перестановок из девяти чисел немного, всего $9! = 362880$. Поэтому можно просто перебрать все перестановки и из n чисел и проверить каждую, не является ли она k-перестановкой.

Приведем фрагмента программы на языке Паскаль.

```
procedure go(pos : integer);
var
  i : integer;
begin
  if (pos = n + 1) then begin
    for i := 1 to n - 1 do begin
      if abs(p[i] - p[i + 1]) > k then begin
        exit;
      end;
    end;
    inc(ans);
    exit;
  end;
  for i := 1 to n do begin
    if (not w[i]) then begin
      w[i] := true;
      p[pos] := i;
      go(pos + 1);
      w[i] := false;
    end;
  end;
end;
```

Процедура `go(pos)` перебирает все возможные еще не использованные числа, которые можно поставить на позицию `pos` в перестановке `p`, если мы знаем все предыдущие числа. Для того, чтобы эффективно проверять, было использовано число или нет, поддерживается булевый массив `w`, причем `w[i] = true` если `i` уже использовано и `false` иначе. Если `pos > n`, то мы построили какую-то перестановку, осталось проверить, что она является k-перестановкой. Если это так, то увеличиваем ответ — переменную `ans`.

Задача I. Турист

Автор задачи: Федор Царев
Автор разбора: Сергей Поромов

Эта задача является одним из вариантов так называемой *задачи о рюкзаке*. Более подробно об этой задаче и ее вариантах можно прочитать, например, в [1] и [2].

Эту задачу можно решать методом перебора. В ней необходимо перебрать всевозможные варианты палаток, которые берет Гена (их всего 8, если считать и когда он ничего не берет) и проверить, что суммарная масса палаток не превышает w, а общая вместимость не меньше k. Если хоть один вариант возможен, вывести YES, иначе NO.

Приведем программную реализацию этого алгоритма.

```
can := false;
if ((b1 + b2 + b3 >= k) and (a1 + a2 + a3 <= w)) then begin
  can := true;
end;
if ((b2 + b3 >= k) and (a2 + a3 <= w)) then begin
  can := true;
end;
if ((b1 + b3 >= k) and (a1 + a3 <= w)) then begin
  can := true;
end;
if ((b1 + b2 >= k) and (a1 + a2 <= w)) then begin
  can := true;
end;
if ((b1 >= k) and (a1 <= w)) then begin
  can := true;
end;
if ((b2 >= k) and (a2 <= w)) then begin
  can := true;
end;
if ((b3 >= k) and (a3 <= w)) then begin
  can := true;
end;
if (can) then begin
  writeln('YES');
end else begin
  writeln('NO');
end;
```

Перебор можно написать куда короче, если использовать двоичное преставление чисел от 0 до 7 и считать, что i-я палатка берется, когда i-й бит равен 1.

```
read(k, w);
for i := 1 to 3 do begin
  read(a[i], b[i]);
end;
can := false;
for i := 0 to 7 do begin
  ww := 0;
  kk := 0;
  for j := 1 to 3 do begin
    if (((1 shl (j - 1)) and i) > 0) then begin
      ww := ww + a[j];
      kk := kk + b[j];
```

```
      end;
    end;
    if (ww <= w) and (kk >= k) then begin
      can := true;
    end;
  end;
  if (can) then begin
    writeln('YES');
  end else begin
    writeln('NO');
  end;
```

Список литературы

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

[2] *Шень А.* Программирование: теоремы и задачи.— М.: МЦНМО, 1995.