Разбор задач Двенадцатой Интернет-олимпиады

Автор: Федор Царев

Введение

В базовой номинации Двенадцатой Интернет-олимпиады сезона 2006-2007 участникам было предложено для решения 8 задач. В олимпиаде приняло участие 58 команд, из них 56 решили хотя бы одну задачу.

Наиболее простой оказалась задача «H. Победитель» — ее решили 54 команды. Наиболее сложной — задача «A. Обобщенные числа Фибоначчи» — ее решили 5 команд.

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

Задача A. Обобщенные числа Фибоначчи

Наиболее простой способ решения задачи состоит в том, чтобы вычислить ответ с помощью двух вложенных циклов.

```
for i := 1 to k do begin
  f[i] := 1;
end;
for i := k + 1 to n do begin
  f[i] := 0;
  for j := 1 to k do begin
    f[i] := f[i] + f[i - j];
    f[i] := f[i] mod p;
  end;
end;
```

Основной недостаток этого решения состоит в том, что оно работает слишком медленно — время его работы составляет $O(k + (n-k)\,k) = O(nk)$. Например, при $n = 10^6, k = 5 \cdot 10^5$, это решение абсолютно точно не уложится в ограничение по времени.

Как получить более быстрое решение? Видимо, стоит найти какие-то свойства чисел Фибоначчи, позволяющие вычислять их быстрее.

Рассмотрим два последовательных обобщенных числа Фибоначчи.

$$F_i^{(k)} = F_{i-1}^{(k)} + \ldots + F_{i-k}^{(k)}$$

$$F_{i+1}^{(k)} = F_i^{(k)} + \ldots + F_{i+1-k}^{(k)}$$

Рассмотрим их разность.

$$F_{i+1}^{(k)} - F_i^{(k)} = F_i^{(k)} + \ldots + F_{i+1-k}^{(k)} - (F_{i-1}^{(k)} + \ldots + F_{i-k}^{(k)}) =$$

$$= F_i^{(k)} + (F_{i-1}^{(k)} + \ldots + F_{i+1-k}^{(k)}) - (F_{i-1}^{(k)} + \ldots F_{i+1-k}^{(k)}) - F_{i-k}^{(k)} =$$

$$= F_i^{(k)} - F_{i-k}^{(k)}$$

Таким образом, $F_{i+1}^{(k)} = 2 \cdot F_i^{(k)} - F_{i-k}^{(k)}$.

Воспользовавшись этим соотношением, можно написать более быструю программу.

```
for i := 1 to k do begin
  f[i] := 1;
end;
for i := k + 1 to n do begin
  f[i] := ((2 * f[i - 1] - f[i - k]) mod p + p) mod p;
end;
```

Эта программа работает за время, пропорциональное n. Таким образом, она легко укладывается в ограничение по времени.

Отметим одну важную деталь реализации. В большинстве языков программирования операция взятия остатка от деления (`mod`, `%`) некорректно работает с отрицательными числами. Например, `-1 mod 2 = -1`, в то время как остаток от деления всегда является неотрицательным числом.

Поэтому необходим способ вычисления остатка от деления произвольного числа a на положительное число p. Докажем, что искомый остаток равен $(a \bmod p + p) \bmod p$.

Для этого необходимо знать, как компьютер производит вычисления. Операции `div` и `mod` подчиняются следующим аксиомам:

1. $(-a)\ div\ p = -(a\ div\ p)$
2. $a = (a\ div\ p) \cdot p + (a \bmod p)$

Отсюда следует, что $a \bmod p$ лежит в пределе от $(-p+1)$ до $(p-1)$. Действительно, если a — неотрицательное число, то $a \bmod p$ — остаток от деления a на p. Если же $a < 0$, то $a\ div\ p = -(|a|\ div\ p)$ и верно неравенство (так как $-(|a| \bmod p) = -(|a| - (|a|\ div\ p) \cdot p)$):

$$a \bmod p = a - (a\ div\ p) \cdot p = -|a| + (|a|\ div\ p) \cdot p = -(|a| \bmod p) \geq (-m + 1)$$

Таким образом, число $(a \bmod p+p) \bmod p$ всегда лежит в отрезке $[0, p-1]$. Поэтому оно и является искомым остатком от деления a на p.

Задача B. Преобразователь строк

Для решения задачи необходимо написать программу, выполняющую описанные в условии действия. Как же это сделать наиболее просто?

Сразу после прочтения строки преобразуем ее в массив символов.

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

Напишем две процедуры, одна из которых будет сортировать заданный отрезок массива, а вторая — его разворачивать.

```
procedure reverse(l, r : integer);
var
  tmp : array[1..MAXM] of char;
  i : integer;
  len : integer;
begin
  len := r - l + 1;
  for i := 1 to len do begin
    tmp[i] := c[l + i - 1];
  end;
  for i := 1 to len do begin
```

```
    c[l + i - 1] := tmp[len - i];
  end;
end;

procedure sort(l, r : integer);
var
  i, j : integer;
  t : char;
begin
  for i := l to r do begin
    for j := i + 1 to r do begin
      if (c[j] < c[i]) then begin
        t := c[i];
        c[i] := c[j];
        c[j] := t;
      end;
    end;
  end;
end;
```

После того, как написаны эти две процедуры, решение задачи становится совсем несложным — необходимо прочитать из входного файла все операции и последовательно их применить к строке.

```
readln(n);
for i := 1 to n do begin
  read(op);
  readln(l, r);
  if (op = 'S') then begin
    sort(l, r);
  end;
  if (op = 'R') then begin
    reverse(l, r);
  end;
end;
for i := 1 to m do begin
  write(c[i]);
end;
writeln;
```

Оценим время работы написанной программы в худшем случае. Сортировка отрезка массива длиной l требует $O(l^2)$ операций, его разворот — $O(l)$ операций. В худшем случае, каждая операция — это сортировка всего массива. Таким образом, время работы программы составляет $O(n\,m^2)$, что укладывается в ограничение по времени при указанных в условии ограничениях на размер входных данных.

Задача C. Оптимизация сепарабельной функции

Первое, что необходимо для решения этой задачи — это не испугаться обилия в ее условии кажущихся на первый взгляд очень «умными» слов. Нередко бывает так, что достаточно несложная задача сформулирована так, что кажется очень сложной.

Итак, заданная функция представима в виде суммы $f(x_1, x_2, \ldots, x_n) = f_1(x_1)+f_2(x_2)+\ldots+f_n(x_n)$, где каждая f_i зависит только от x_i. Поэтому (в этом состоит главная идея решения) каждую из функций $f_i(x_i)$ можно минимизировать отдельно.

Таким образом, задача свелась к минимизации n квадратичных функций на n отрезках. Решим эту задачу.

Рассмотрим функцию $F(x) = Px^2 + Qx + R$ на отрезке $[A, B]$. Известно, что квадратичная функция на отрезке достигает своего минимума либо на его границе, либо в точке, соответствующей вершине параболы (если она лежит внутри рассматриваемого отрезка). Отметим, что это утверждение верно для квадратичной как с положительным коэффициентом при квадрате ($P > 0$), так и отрицательным ($P < 0$).

Таким образом, функция вычисления минимума на отрезке выглядит так.

```
procedure minimize(a, b, p, q, r : extended; var value, x : extended);
var
  x0 : extended;
  y0 : extended;
begin
  value := p * a * a + q * a + r;
  x := a;
  if (p * b * b + q * b + r < value) then begin
    value := p * b * b + q * b + r;
    x := b;
  end;
  x0 := -b / (2 * a);
  if (a <= x0) and (x0 <= b) then begin
    y0 := p * x0 * x0 + q * x0 + r;
    if (y0 < value) then begin
      value := y0;
      x := x0;
    end;
  end;
end.
```

Далее для решения задачи необходимо n раз вызвать эту функцию с соответствующими аргументами.

```
read(n);
ans := 0;
for i := 1 to n do
begin
  read(a, b, p, q, r);
  minimize(a, b, p, q, r, v[i], x[i]);
  ans := ans + v[i];
end;
writeln(ans : 0 : 10);
for i := 1 to n do begin
  write(x[i] : 0 : 10, ' ');
end;
```

Отметим, что в подобных задачах, как правило, лучше выводить достаточно большее количество знаков после десятичной точки (если, конечно, в условии не сказано: «Выводите ровно 3 знака после десятичной точки.»).

Отметим также, что термин *сепарабельная функция* действительно используется в теории методов оптимизации.

Задача D. Перестановка

Изучим более подробно структуру последовательности $a_n = 1, 0, 3, 2, 5, 4, \ldots$. Для этого разобьем

множество $\mathbb{Z}^+ = \{0, 1, 2, 3, \ldots\}$ на пары чисел: $(0, 1), (2, 3), (4, 5), \ldots$ и заметим, что в последовательности a_n эти пары идут в том же порядке, но в каждой из них числа поменяны местами.

Далее, заметим, что частные от деления чисел из одной пары на два совпадают: $0/2 = 1/2 = 0, 2/2 = 3/2 = 1, 4/2 = 5/2 = 2, \ldots$. Таким образом, по числу x легко найти номер пары, в котором оно находится — $n = x/2$.

Далее, отметим, что в последовательности a_n в каждой паре сначала стоит нечетное число, а потом — четное. Таким образом, если x нечетно, то ответ равен $2 \cdot n$ (позиция первого числа в n-ой паре), а если x четно, то ответ равен $2 \cdot n + 1$.

В качестве упражнения оставим доказательство того, что ответ на задачу также выражается формулой $x \oplus 1$, где как $\oplus$ обозначена операция побитового сложения по модулю два («исключающее или», xor).

Задача E. Космические путешествия

Прежде всего составим математическую модель задачи («переведем» условие на язык математики). Для решения этой задачи удобно применить аппарат теории графов.

Рассмотрим граф, вершинами которого являются планеты рассматриваемой звездной системы, а ребрами — телепортирующие туннели. По условию задачи этот граф является связным (то есть из каждой вершины можно по ребрам добраться в любую другую) и в нем нет циклов (так как, если бы в нем был цикл, на котором лежат вершины u и v, то между этими вершинами было бы как минимум два простых пути). Значит, рассматриваемый граф является деревом.

Вершины, соответствующие важным планетам, в теории графов называются *точками сочленения*. Итак, если переформулировать задачу на языке теории графов, получается: «Задано дерево. Необходимо найти количество его точек сочленения.»

Назовем *степенью* вершины количество ребер, концом которых она является. Докажем, что вершина дерева является точкой сочленения тогда и только тогда, когда ее степень больше единицы.

Пусть v — рассматриваемая вершина, и ее степень больше единицы. Тогда существуют две вершины u и w, соединенные с ней ребрами. Между u и w существует единственный путь, значит, это путь $u - v - w$. Этот путь проходит через вершину v. Если удалить из дерева вершину v, то этот путь перестанет соединять u и w, значит появится хотя бы одна пара вершин, несоединенных путем. Значит, v — точка сочленения.

Пусть теперь v является точкой сочленения. Предположим, что ее степень равна единице. Тогда она соединена ребром только с одной вершиной — обозначим ее x. Так как v — точка сочленения, то существуют такие две вершины u и w, что единственный путь между ними проходит через v, причем в этом пути каждая вершина встречается не более одного раза. Обозначим вершину, идущую перед v в этом пути, как y, а идущую сразу после — как z. Так как степень v равна единице, то $x = y = z$. Значит, вершина x дважды входит в рассматриваемый путь.

Если же степень v равна нулю, то через нее не проходит никакой путь. Значит, она не может быть точкой сочленения.

Таким образом, решение задачи состоит в том, чтобы найти степени каждой вершины и количество вершин со степенью, большей единицы. Ниже приведен один из вариантов реализации этого решения.

```
readln(n);
for i := 1 to n - 1 do begin
  readln(u, v);
  inc(deg[u]);
  inc(deg[v]);
end;
ans := 0;
for i := 1 to n do begin
  if (deg[i] > 1) then begin
    inc(ans);
```

```
  end;
end;
writeln(ans);
```

Задача F. Стабилизация последовательности

Решение задачи состоит в вычислении членов последовательности $a_1, a_2, a_3, \ldots$ до тех пор, пока не будет установлен факт стабилизации. При этом вычисления необходимо закончить, если вычислено более 1000 членов последовательности.

Напишем функцию, которая будет вычислять сумму цифр числа x.

```
function digSum(x : integer) : integer;
begin
  result := 0;
  while x > 0 do begin
    result := result + x mod 10;
    x := x div 10;
  end;
end;
```

Далее, напишем функцию, вычисляющую функцию $s(x)$, описанную в условии задачи, — сумму цифр делителей числа x.

```
function s(x : integer) : integer;
var
  i : integer;
begin
  result := 0;
  i := 1;
  while i * i <= x do begin
    if (x mod i = 0) then begin
      result := result + digSum(i);
      if (i <> x div i) then begin
        result := result + digSum(x div i);
      end;
    end;
  end;
end;
```

Решение, использующее описанный процедуры, может быть, например, таким.

```
readln(x);
a[1] := x;
ans := -1;
for i := 2 to 1000 do begin
  a[i] := s(a[i - 1]);
  if (a[i] = a[i - 1]) then begin
    ans := i - 1;
    break;
  end;
end;
writeln(ans);
if (ans <> -1) then begin
  for i := 1 to ans do begin
    write(a[i], ' ');
```

```
  end;
  writeln;
end;
```

Задача G. Расширение Вселенных

Заметим, что первый момент времени, в который у Вселенных будет общая точка — это момент времени t, в который сумма радиусов Вселенных будет равна расстоянию между их центрами в начальный момент времени.

Таким образом, t находится из следующего уравнения:

$$r_1 + v_1 \cdot t + r_2 + v_2 \cdot t = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Решая это уравнение, получаем:

$$t = \frac{\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} - r_1 - r_2}{v_1 + v_2}$$

Задача H. Победитель

Решение этой задачи состоит в программной реализации описанных в условии правил подсчета очков. Программа может быть, например, такой.

```
n := 11;
for i := 1 to n do begin
  read(place1[i]);
end;
for i := 1 to n do begin
  read(place2[i]);
end;
score1 := 0;
score2 := 0;
for i := 1 to n do begin
  if (place1[i] < place2[i]) then begin
    inc(score1);
  end;
  if (place2[i] < place1[i]) then begin
    inc(score2);
  end;
end;
if (score1 > score2) then begin
  writeln('First');
end else begin
  writeln('Second');
end;
```

Список литературы

[1] Романовский И.В. Дискретный анализ: Учебное пособие для студентов, специализирующихся по прикладной математике и информатике. - 3-е изд., перераб. и доп. - СПб: Невский Диалект; БХВ Петербург, 2003. - 320 с.: ил

[2] Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ. - М.: МЦНМО, 1999. - 960 с., 263 ил.

[3] Шень А. Программирование: теоремы и задачи. - М.: МЦНМО, 1995. - 264 с.