Разбор задач Десятой Интернет-олимпиады

Автор: Федор Царев

Введение

В базовой номинации Десятой Интернет-олимпиады сезона 2006-2007 участникам было предложено для решения 8 задач. В олимпиаде приняло участие 92 команды, из них 70 решили хотя бы одну задачу.

Наиболее простой оказалась задача «H. Свадьба» — ее решили 55 команд. Наиболее сложной — задача «C. Голова на плечах» — ее не решила ни одна команда.

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

Задача A. Количество делителей

Рассмотрим разложение числа x на простые множители:

$$x = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot \ldots \cdot p_k^{\alpha_k}$$

Так все p_i — простые, то любой делитель y числа x имеет вид $y = p_1^{\beta_1} \cdot p_2^{\beta_2} \cdot \ldots \cdot p_k^{\beta_k}$, где $0 \le \beta_i \le \alpha_i$.

Таким образом, число делителей равно числу соответствующих им последовательностей β_1, β_2, …, β_k, которое равно $(\alpha_1 + 1) \cdot (\alpha_2 + 1) \cdot \ldots \cdot (\alpha_k + 1)$.

Задача B. Деление с остатком

Пусть запись числа n в десятичной системе счисления такова: $a_l a_{l-1} \ldots a_0$. Тогда $n = 10^l \cdot a_l + 10^{l-1} \cdot a_{l-1} + \ldots + 1 \cdot a_0$.

Таким образом задача свелась к вычислению остатка от деления значения указанного выражения на k. Как доказано в [4] (разбор задачи F), арифметические операции можно проводить непосредственно с остатками.

Значит,

$$n \bmod k = (10^l \cdot a_l + 10^{l-1} \cdot a_{l-1} + \ldots + 1 \cdot a_0) \bmod k =$$
$$= ((10^l \bmod k) \cdot (a_l \bmod k) + (10^{l-1} \bmod k) \cdot (a_{l-1} \bmod k) + \ldots + (1 \cdot a_0) \bmod k) \bmod k$$

Вычисление указанной суммы удобно вести справа налево, от слагаемого $(1 \cdot a_0) \bmod k$ к слагаемому $(10^l \bmod k) \cdot (a_l \bmod k)$, так как при этом не возникает необходимости заново вычислять необходимую степень десяти. Она пересчитывается по формуле: $10^i \bmod k = (10 \cdot (10^{i-1} \bmod k)) \bmod k$. Кроме этого, для вычисления этой суммы можно применить так называемую «схему Горнера» [3].

Задача C. Голова на плечах

Рассмотрим для начала более простую задачу. Задана последовательность чисел $a_1, a_2, \ldots, a_n$, в которой каждое из чисел от 1 до n встречается ровно один раз (такая последовательность называется *перестановкой порядка* n). Необходимо найти количество пар $i < j$, таких что $a_i > a_j$ (каждая такая пара называется *инверсией*).

При небольших n (например, $n \le 5000$) эту задачу можно решать перебором.

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

Однако, для больших n это решение будет работать слишком долго.

Применим для решения этой задачи метод «разделяй-и-властвуй» и модифицируем алгоритм сортировки слиянием [2]. Пусть необходимо найти количество инверсий в массиве $\{a_1, a_2, \ldots, a_n\}$. Разобьем его на две части: $\{a_1, a_2, \ldots, a_{n/2}\}$ и $\{a_{n/2+1}, a_{n/2+2}, \ldots, a_n\}$.

Обозначим как $IC(\{a_1, a_2, \ldots, a_n\})$ количество инверсий в массиве $\{a_1, a_2, \ldots, a_n\}$. Тогда $IC(\{a_1, a_2, \ldots, a_n\}) = IC(\{a_1, a_2, \ldots, a_{n/2}\}) + IC(\{a_{n/2+1}, a_{n/2+2}, \ldots, a_n\}) + C$, где как C обозначено количество инверсий, в которых один из элементов лежит в первой половине массива $\{a_1, a_2, \ldots, a_{n/2}\}$, а второй — во второй $\{a_{n/2+1}, a_{n/2+2}, \ldots, a_n\}$. Таким образом, количество инверсий можно найти с помощью следующей рекурсивной функции.

```
function countInversions(l, r : integer) : integer;
var
  m : integer;
begin
  if (l >= r) then begin
    result := 0;
    exit;
  end;
  m := (l + r) div 2;
  result := countInversions(l, m) + countInversions(m + 1, r);
  result := result + countC(l, m, m + 1, r);
end;
```

Осталось реализовать функцию `countC()`. Если ее удастся реализовать так, чтобы ее время работы составляло $O(r - l)$, то время работы всего алгоритма составит $O(n \log n)$, поскольку на каждом шаге массив делится пополам, и, значит, глубина рекурсии составляет $O(\log n)$.

Заметим, что C равно следующей сумме: $\sum_{i=n/2+1}^{n} more(a_i)$, где как $less(a_i)$ обозначено количество элементов из $\{a_{+1}, a_2, \ldots, a_{n/2}\}$, больших a_i.

Предположим, что массивы $\{a_1, a_2, \ldots, a_{n/2}\}$ (первый массив) и $\{a_{n/2+1}, a_{n/2+2}, \ldots, a_n\}$ (второй массив) уже отсортированы. «Научимся» строить из них один отсортированный массив, попутно вычисляя C. Какой элемент будет первым в отсортированном массиве? Поскольку $\{a_1, a_2, \ldots, a_{n/2}\}$ и $\{a_{n/2+1}, a_{n/2+2}, \ldots, a_n\}$ отсортированы по возрастанию, то это будет минимальный из a_1 и $a_{n/2+1}$. Аналогично далее, после выбора первого элемента, можно будет выбрать второй.

При этом если меньшим оказывается элемент из второго массива, то к числу C необходимо прибавить количество уже обработанных элементов из первого массива (все они меньше рассматриваемого элемента, а остальные элементы — больше, тут важно, что оба массива отсортированы).

Таким образом, функция `countC` не только осуществляет вычисление числа C, но и осуществляет слияние двух отсортированных массивов в один. Ее программная реализация остается в качестве упражнения читателю.

Вернемся теперь к исходной задаче и попробуем ее свести к рассмотренной только задаче вычисления количества инверсий. Это значит, что мы попробуем по входным данным исходной задачи построить входные данные для задачи вычисления количества инверсий, ответ для которой мы потом используем для нахождения ответа к исходной задаче.

Заметим, что все начальные и конечные точки волос различны. Присвоим волосам номера в порядке возрастания их полярных углов относительно центра окружности (за положительное направление выбирается направление «против часовой стрелки»), причем будем считать, что угол принимает значения из промежутка $(0; 2\pi)$ и нулевому углу соответствует направление «на север», т.е. в сторону увеличения ординаты. Теперь построим перестановку $p_1, p_2, \ldots, p_n$ следующим образом: p_i равно номеру волоса, у которого X-координата **конечной** точки i-ая по возрастанию. Заметим, что инверсия в этой перестановке соответствует паре секущихся волос — инверсия соответствует тому, что один волос из пары начинается раньше и заканчивается позже другого. С другой стороны, по той же причине каждой паре секущихся волос соответствует инверсия. Полное доказательство корректности описанного сведения остается читателю в качестве упражнения.

Таким образом, ответ на исходную задачу равен количеству инверсий в построенной перестановке p_i. Отметим также, что при вычислении этого числа следует использовать 64-битные целочисленные типы данных, поскольку максимальный ответ в этой задаче равен $\frac{10^5 \cdot (10^5-1)}{2}$, а такое число не «помещается» в 32-битный тип данных.

Упражнение. Какое максимальное и минимальное количество инверсий может быть в перестановке порядка n? Постройте соответствующие перестановки.

Задача D. Число-палиндром

Наиболее простой способ решения этой задачи — перебор основания системы счисления и проверка для каждого из них, является ли представление данного числа в соответствующей системе счисления палиндромом.

Для проверки, является ли строка палиндромом удобно использовать следующую функцию (текст функции приведен на языке программирования *Delphi*).

```
function isPalindrome(s : string) : boolean;
var
  i, l : integer;
begin
  result := true;
  l := length(s);
  for i := 1 to l do begin
    if (s[i] <> s[l - i + 1]) then begin
      result := false;
      exit;
    end;
  end;
end;
```

Далее, необходимо «научиться» переводить число в систему счисления с заданным основанием. Для этого также удобно написать функцию, осуществляющую перевод числа n в систему счисления с основанием k.

```
function convert(n, k : integer) : string;
var
  i, l : integer;
  digit : char;
  c : integer;
begin
  result := '';
  while (n > 0) do begin
    c := n mod k;
    if (c < 10) then begin
      digit := chr(ord('0') + c);
    end else begin
      digit := chr(ord('a') + c - 10);
    end;
    n := n div k;
    result := digit + result;
  end;
end;
```

Таким образом, для конкретного основания k системы счисления задача проверки того, является ли представление числа n в этой системе счисления палиндромом, решена. Теперь переберем все

значения k от 2 до 36 и для каждого из них отметим в массиве `good : array[2..36] of boolean`, является ли представление числа n в соответствующей системе счисления палиндромом.

Эту идею реализует следующий фрагмент программы.

```
for i := 2 to 36 do begin
  w[i] := isPalindrome(convert(n, i));
end;
```

Теперь перейдем к выводу ответа. Для этого найдем количество элементов, равных `true`, в массиве `good`. После этого необходимо вывести соответствующее слово (none, unique, multiple) и, при необходимости, все «хорошие» основания в порядке возрастания.

Это можно сделать, например, так.

```
cnt := 0;
for i := 2 to 36 do begin
  if (good[i]) then begin
    inc(cnt);
  end;
end;
if (cnt = 0) then begin
  writeln('none');
end else begin
  if (cnt = 1) then begin
    writeln('unique');
  end else begin
    writeln('multiple');
  end;
  for i := 2 to 36 do begin
    if (good[i]) then begin
      write(i, ' ');
    end;
  end;
  writeln;
end;
```

Задача E. Забавная последовательность

Так как по условию $a_{i+1} \le a_i + 3$, то $a_n \le 3 \cdot n$. По условию, $n \le 10^5$, следовательно, ответ на задачу не превосходит 300 000.

Для того, чтобы хранить числа, встречающиеся в последовательности, воспользуемся массивом логического типа (`was : array[1..300000] of boolean;`). Соответственно, `was[i] = true` будет значить, что число `i` встречается в последовательности, а `was[i] = false` — то, что не встречается.

Следующий фрагмент программы реализует описанный подход:

```
w[1] := true;
cur := 1;
for i := 2 to n do begin
  if (w[i]) then begin
    cur := cur + 3;
  end else begin
    cur := cur + 2;
  end;
  w[cur] := true;
```

```
end;
writeln(cur);
```

Задача F. Проверка орфографии

Общая идея решения состоит в том, что необходимо проверить, что каждое слово из текста встречается в словаре и каждое слово из словаря встречается в тексте.

Таким образом, необходимо разбить текст на слова и разработать некоторую структуру данных, которая позволяет выполнять операции «добавить слово» и «проверить наличие слова».

Решим для начала задачу построения необходимой структуры данных. В качестве такой структуры данных может выступать массив.

```
type dictionary = record
  a : array[1..MAXN] of string;
  size : integer;
end;
```

В массиве операция добавления может быть реализована следующим образом.

```
procedure add(s : string; var d : dictionary);
var
  found : boolean;
begin
  found := false;
  for i := 1 to d.size do begin
    if (d.a[i] = s) then begin
      found := true;
      break;
    end;
  end;
  if (not found) then begin
    inc(d.size);
    d.a[d.size] := s;
  end;
end;
```

Операция поиска реализуется так.

```
function find(s : string; const d : dictionary) : boolean;
var
  found : boolean;
begin
  found := false;
  for i := 1 to d.size do begin
    if (d.a[i] = s) then begin
      result := true;
      exit;
    end;
  end;
  result := false;
end;
```

Далее, необходимо реализовать разбиение строки на слова. Для этого необходимо при чтении пропускать все символы, не являющиеся буквами. Это можно сделать, например, так.

```
readln(s);
i := 1;
len := length(s);
while (i <= len) do begin
  while (i <= len) and (not isLetter(s[i])) do begin
    inc(i);
  end;
  curWord := '';
  while (isLetter(s[i])) do begin
    curWord := curWord + s[i];
    inc(i);
  end;
  add(toLowerCase(curWord), textD);
end;
```

Здесь функция `isLetter(c)` возвращает `true`, если символ `c` является буквой, и `false` — иначе, а функция `toLowerCase(s)` возвращает строку, содержащую те же буквы, что и `s`, только в нижнем регистре. Это необходимо, так как по условию задачи большие и маленькие буквы различать не требуется. Эти две функции могут быть реализованы, например, так.

```
function isLetter(c : char) : boolean;
begin
  result := (('a' <= c) and (c <= 'z')) or
            (('A' <= c) and (c <= 'Z'));
end;

function toLowerCase(s : string) : string;
var
  l : integer;
  i : integer;
  c : char;
begin
  l := length(s);
  result := '';
  for i := 1 to l do begin
    c := s[i];
    if (('A' <= c) and (c <= 'Z')) then begin
      result := result + chr(ord('A') + ord(c) - ord('a'));
    end else begin
      result := result + c;
    end;
  end;
end;
```

Написание полного текста программы, решающей задачу и использующей описанные в разборе процедуры и функции, остается читателю в качестве упражнения.

Задача G. Наручные часы

Общая идея решения задачи такова: будем моделировать течение времени, обновляя показания часов, и после каждой минуты проверять, верно ли, что все полоски во всех элементах уже были и белыми, и черными.

Для этого необходимо поддерживать два множества: B полосок, которые уже были черными, и W полосок, которые уже были белыми. Изначально оба эти множества пусты. Далее, на каждом

шаге по времени (на каждой секунде) будем обновлять эти множества: $B = B \cup CB$, $W = W \cup CW$, где CB — множество полосок, которые являются черными, когда часы показывают текущее время, а CW — множество полосок, которые являются белыми.

Осталось разработать способ хранения этих множеств и выполнения операции объединения. Будем хранить части множеств, соответствующие отдельным элементам индикатора, отдельно — с помощью битовых масок. Тогда объединение будет соответствовать побитовой операции `OR` (подробнее о битовых масках можно прочитать в [4]). Таким образом, каждое из множеств CB и CW будет храниться в массиве `array[1..4] of integer`.

Программа, реализующая эту идею, может выглядит, например, так:

```
readln(str);
h1 := ord(str[1]) - ord('0');
h2 := ord(str[2]) - ord('0');
m1 := ord(str[4]) - ord('0');
m2 := ord(str[5]) - ord('0');
ans := 0;
while (true) do begin
  done := true;
  done := done and (w[1] = ((1 shl 6) - 1));
  done := done and (b[1] = ((1 shl 6) - 1))
  for i := 2 to 4 do begin
    done := done and (w[i] = ((1 shl 7) - 1));
    done := done and (b[i] = ((1 shl 7) - 1));
  end;
  if (done) then begin
    break;
  end;
  w[1] := w[1] or cw1[h1];
  b[1] := b[1] or cb1[h1];

  w[2] := w[2] or cw2[h2];
  b[2] := b[2] or cb2[h2];

  w[3] := w[3] or cw2[m1];
  b[3] := b[3] or cb2[m1];

  w[4] := w[4] or cw2[m2];
  b[4] := b[4] or cb2[m2];


  inc(m2);
  if (m2 = 10) then begin
    m2 := 0;
    inc(m1);
  end;
  if (m1 = 6) then begin
    m1 := 0;
    inc(h2);
  end;
  if (h2 = 10) then begin
    h2 := 0;
    inc(h1);
  end;
```

```
  if (h1 = 2) and (h2 = 4) then begin
    h1 := 0;
    h2 := 0;
  end;
  inc(ans);
end;
writeln(ans);
```

Поясним принцип работы этого фрагмента программы. В массивах `cw1`, `cb1` хранятся множества полосок, являющихся соответственно белыми или черными, когда первый элемент индикатора часов показывает соответствующую цифру. Например, в `cw1[1]` хранится множество полосок, являющихся белыми, когда первый элемент показывает цифру 1.

В массивах `cw2`, `cb2` соответственно хранятся множества полосок, являющихся белыми или черными, когда другие элементы часов (со второго по четвертый) показывают соответствующую цифру. Например, в `cb2[2]` хранится множество полосок, являющихся черными, когда второй, третий или четвертый элемент часов показывает цифру 2.

Массивы `cw1`, `cw2`, `cb1`, `cb2` необходимо сформировать вручную. Разработка способа формирования этих массивов остается читателю в качестве упражнения.

Задача H. Свадьба

Для начала рассмотрим небольшой пример. Пусть $N = 2$, сумма денег на счету у Горгоны изначально равна 1, а у молодых людей на счету находится соответственно 2 и 3. Тогда возможны два варианта: либо Горгона выйдет замуж вначале за первого, а потом — за второго, или наоборот.

В первом случае Горгона получит $\frac{\frac{1+2}{2}+3}{2} = 2.25$, а во втором — $\frac{\frac{1+3}{2}+2}{2} = 2$. Таким образом, возникает мысль, что вначале необходимо выходить замуж за менее богатых, а затем — за более богатых.

Попробуем доказать это утверждение. Пусть у Горгоны на счету изначально A рублей, а у молодых людей: $x_1, x_2, \ldots, x_n$ ($x_i > A$). Тогда количество денег, которое Горгона получит после своей махинации равно

$$\frac{\frac{\frac{A+x_1}{2}+x_2}{2} + \ldots + x_n}{2} = \frac{A}{2^n} + \frac{x_1}{2^n} + \frac{x_2}{2^{n-1}} + \ldots + \frac{x_n}{2}$$

Покажем, что для того, чтобы это число было максимальным должны выполняться неравенства: $x_1 \le x_2 \le \ldots \le x_n$. Предположим обратное, то есть пусть существуют два индекса $i < j$, что $x_i > x_j$. Рассмотрим два варианта: когда эти два числа расположены на указанных позициях, и когда эти числа поменяны местами. Обозначим суммы, которые получатся в первом случае как S_1, во втором случае как S_2. Рассмотрим разность:

$$S_2 - S_1 = \frac{x_i}{2^{n+1-i}} + \frac{x_j}{2^{n+1-j}} - \frac{x_j}{2^{n+1-i}} - \frac{x_i}{2^{n+1-j}} =$$

$$= x_i\left(\frac{1}{2^{n+1-i}} - \frac{1}{2^{n+1-j}}\right) + x_j\left(\frac{1}{2^{n+1-j}} - \frac{1}{2^{n+1-i}}\right) =$$

$$= (x_i - x_j)\left(\frac{1}{2^{n+1-i}} - \frac{1}{2^{n+1-j}}\right) > 0$$

Таким образом, понятно, что если существуют два индекса i и j, обладающие указанным свойством, то такой порядок чисел $x_1, x_2, \ldots, x_n$ не является оптимальным. Значит, оптимальным является порядок возрастания.

Заметим также, что не имеет смысла выходить замуж за того, у кого денег меньше, чем у Горгоны. Это следует из того, что если $a < b$, то $\frac{a+b}{2} < b$. Более того, если есть возможность выйти замуж за того, у кого больше денег, то это выгодно сделать.

Таким образом, решение задачи таково: необходимо отсортировать суммы денег на счету у потенциальных женихов по неубыванию, а затем просматривать их в этом порядке, и, если у текущего молодого человека денег больше, чем у Горгоны на настоящий момент, то необходимо выйти на него замуж.

Этот подход реализует следующий фрагмент программы.

```
read(n);
for i := 1 to n do begin
  read(x[i]);
end;
read(a);
for i := 1 to n do begin
  for j := i + 1 to n do begin
    if (x[j] < x[i]) then begin
      t := x[i];
      x[i] := x[j];
      x[j] := t;
    end;
  end;
end;

for i := 1 to n do begin
  if (x[i] > a) then begin
    a := (x[i] + a) / 2;
  end;
end;

writeln(a : 0 : 8);
```

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.

[4] Разбор задач Восьмой Интернет-олимпиады, http://neerc.ifmo.ru/school/io/archive/20070203/solutions-basic-08.pdf