Разбор задач Седьмой Интернет-олимпиады

Введение

В базовой номинации Седьмой Интернет-олимпиады сезона 2008-2009 участникам было предложено для решения восемь задач. В олимпиаде приняло участие 49 команд, из них 38 решили хотя бы одну задачу.

Наиболее простой оказалась задача «G. Тетрис» — ее решили 32 команды. Наиболее сложной — задача «A. Красивые даты» — ее решили 8 команд.

В связи с техническими проблемами на сервере олимпиада длилась восемь часов. По ее результатам перевод из одной номинации в другую не производился.

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

Задача A. Красивые даты

Автор задачи: Владимир Ульянцев
Автор разбора: Антон Ахи

Посчитаем, сколько дней в Вовином календаре с первого января первого года по 31 декабря 9999 года. 365×9999 дней без учета високосности и еще $\frac{9999}{4} = 2499$ високосных лет. Итого 3652134 дней. Решение, которое будет перебирать все дни в заданном интервале и проверять их на красоту подходит по временным ограничениям.

Для удобства создадим тип данных `date`, представляющий дату:

```
date = record
  d, m, y : integer;
end;
```

Создадим массив с числом дней в месяцах в невисокосном году:

```
  days : array[1..12] of integer =
                        (31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31);
```

Также необходимы функции перевода числа в дату, сравнения дат, проверки, является ли дата красивой, и увеличение даты на один день.

Функция перевода числа в дату:

```
function inttodate(a : integer) : date;
begin
  result.d := a div 1000000;
  result.m := a div 10000 mod 100;
  result.y := a mod 10000;
end;
```

Функция сравнения дат:

```
function equals(a, b : date) : boolean;
begin
  result := (a.d = b.d) and (a.m = b.m) and (a.y = b.y);
end;
```

Процедура увеличения даты на один день (случай февраля високосного года в ней рассматривается отдельно):

```
procedure incdate(var a : date);
begin
  a.d := a.d + 1;
  if (a.m = 2) and (a.y mod 4 = 0) then begin
    if (a.d = 30) then begin
      a.d := 1;
      a.m := a.m + 1;
    end;
  end else begin
    if (a.d > days[a.m]) then begin
      a.d := 1;
      a.m := a.m + 1;
    end;
  end;
  if (a.m > 12) then begin
    a.m := 1;
    a.y := a.y + 1;
  end;
end;
```

Для определения «красоты» даты необходима функция проверки того, является ли развернутая дата корректной:

```
function gooddate(a : date) : boolean;
begin
  if (a.m > 12) or (a.m < 1) or (a.d < 1) then begin
    result := false;
    exit;
  end;
  if (a.m = 2) and (a.y mod 4 = 0) then begin
    result := a.d <= 29;
  end else begin
    result := a.d <= days[a.m];
  end;
end;
```

Тогда функция проверки даты на красоту будет выглядеть следующим образом:

```
function get(a : date) : integer;
var
  b : date;
begin
  b.d := a.y div 10 mod 10 + a.y mod 10 * 10;
  b.m := a.y div 1000 mod 10 + a.y div 100 mod 10 * 10;
  b.y := a.d div 10 +
                 10 * (a.d mod 10 + 10 * (a.m div 10 + 10 * (a.m mod 10)));
  if (gooddate(b)) then begin
    result := 1;
  end else begin
    result := 0;
  end;
end;
```

Для удобства эта функция возвращает единицу, если дата красивая, и ноль в противном случае.

С таким набором функций код программы выглядит следующим образом:

```
begin
  reset(input, 'dates.in');
  rewrite(output, 'dates.out');
  read(st, fin);
  a := inttodate(st);
  b := inttodate(fin);
  ans := get(b);
  while not(equals(a, b)) do begin
    ans := ans + get(a);
    incdate(a);
  end;
  writeln(ans);
end.
```

Задача B. Производство деталей

Автор задачи: Владимир Ульянцев
Автор разбора: Антон Банных

Предположим, что нам известна оптимальная стратегия использования капсул. Заметим, что если в какой-то момент часть капсул нужно пустить на переработку, то детали из них можно производить непосредственно перед переработкой. Применяя это рассуждение можно объединить процесс переработки и изготовления деталей и получить неделимую операцию: взять m полных капсул, произвести m деталей и получить k полных капсул. Кроме того, из полной капсулы можно произвести деталь, а использованную капсулу выбросить.

Применим метод динамического программирования. Пусть $d[i]$ — максимальное количество деталей, которое можно получить из i капсул при фиксированных m и k. Очевидно, что $d[0] = 0$. Из предыдущих рассуждений следует, что при $i < m$ $d[i] = d[i-1] + 1$, а при $i >= m$ $d[i] = max(d[i-1]+1, d[i-m+k]+m)$.

Вычислим значения массива d для всех возможных a_i (по условию задачи $a_i \leq 10^6$), после чего ответим на все запросы просто обратившись к $d[a_i]$.

Фрагмент решения, на языке Паскаль:

```
read(n, m, k);
d[0] := 0;
for i := 1 to 1000000 do begin
        d[i] := d[i - 1] + 1;
        if (i >= m) then
                d[i] := d[i - m + k] + m;
end;
for i := 1 to n do begin
        read(a);
        write(d[a], ' ');
end;
```

Время работы алгоритма есть $O(max(a_i))$ (в данном случае $max(a_i) = 10^6$).

Особо отметим, что при некоторых сочетаниях m и k ответ для больших a_i превышает максимальное допустимое значение 32-битного типа. Например, при $m = 100000$, $k = 99999$ из 10^6 капсул можно получить 90000199999 деталей. Оценить сверху ответ можно как $m \times max(a_i)$, что при данных ограничениях составляло порядка 10^{12}. Это показывает, что для хранения ответа необходимо использовать 64-битный тип данных.

Задача C. Футбол

Автор задачи: Владимир Ульянцев
Автор разбора: Антон Банных

Первое на что стоит обратить внимание в этой задаче — возможность сведения ее к меньшим подзадачам. Действительно, для того, чтобы решить задачу для фиксированных n и k достаточно знать ответы на задачи для пар $(n, k-1)$, $(n-1, k-1)$ и $(n-3, k-1)$, которые соответствуют различным возможным результатам последнего матча (поражение, ничья и победа соответственно). Действительно, предположим, нам известно, что в последнем матче команда проиграла. Тогда количество вариантов набрать n очков за k матчей равно количеству вариантов набрать n очков за $k-1$ матч, так как за последний k-й матч не изменил суммарный счет команды. Аналогично разбираются случаи ничьей и победы.

Одно из возможных решений состоит в использовании метода динамического программирования. Заведем массив d, в котором в ячейке $d[k, n]$ будет храниться ответ на задачу для n очков и k матчей. Тогда при $k \geq 1$ и $n \geq 3$ очков $d[k, n] = d[k-1, n] + d[k-1, n-1] + d[k-1, n-3]$.

Фрагмент решения, на языке Паскаль:

```
read(n, k);
d[0, 0] := 1;
for i := 1 to k do
        for j := 0 to 3 * k do begin
                inc(d[i, j], d[i - 1, j]);
                if (j >= 1) then inc(d[i, j], d[i - 1, j - 1]);
                if (j >= 3) then inc(d[i, j], d[i - 1, j - 3]);
        end;
writeln(d[k, n]);
```

Время работы алгоритма есть $O(nk)$.

Существуют и другие подходы к решению данной задачи. Например, идею о том, что для нахождения ответа достаточно знать ответ для трех меньших задач, можно было использовать для построения переборного решения, которое также укладывалось в ограничения по времени. Поскольку для каждого состояния есть три перехода в состояние с меньшим k, то время работы такого алгоритма есть $O(3^k)$.

Приведем рекурсивную функцию на языке Паскаль, соответствующую изложенным идеям. Отдельно разбирается случай, когда не сыграно ни одного матча.

```
function count(n, k : integer) : int64;
begin
        if (k = 0) then begin
                if (n = 0) then
                        result := 1
                else
                        result := 0;
        end else begin
                result := count(n, k - 1);
                inc(result, count(n - 1, k - 1));
                inc(result, count(n - 3, k - 1));
        end;
end;
```

Еще один подход состоит в том, чтобы перебрать количество побед и поражений, после чего для каждого допустимого варианта вычислить количество различных расстановок из комбинаторных рассуждений. Приведем конечную формулу. Пусть в k матчах было одержано x побед, а y матчей

было сыграно вничью. Тогда количество различных расстановок исходов матчей есть $C_k^x \times C_{k-x}^y$. Время работы перебора числа побед и поражений есть $O(k^2)$. Построение треугольника Паскаля [2] для вычисления используемых в формулах биномиальных коэффициентов выполняется за такое же время, то есть суммарное время работы алгоритма есть $O(k^2)$.

Задача D. Бумага

Автор задачи: Владимир Ульянцев
Автор разбора: Роман Сатюков

Пусть лист бумаги имеет размеры n на m. Заметим, что при сгибе бумаги размер одной из сторон не меняется, а размер другой стороны уменьшается не более чем в два раза. Пусть $2^{k-1} < n \le 2^k$ и $2^{q-1} < m \le 2^q$. Тогда лист надо сгибать как минимум $k+q$ раз чтобы получить лист 1×1. Таким образом, ответ не меньше $k+q$.

Покажем, как за $k+q$ действий согнуть лист в 1×1. Для этого сначала лист $n \times m$ согнем в лист $1 \times m$ за k операций следующим образом: если текущие размеры листа имеют вид $(2t+1) \times m$, то согнем его в лист $(t+1) \times m$, а если $2t \times m$, то согнем пополам в $t \times m$. В результате за k операций получится лист $1 \times m$. Теперь аналогичным образом согнем лист $1 \times m$ в лист 1×1 за q операций.

Фрагмент решения, на языке Паскаль:

```
read(m, n);
ans := 0;
// Считаем k и сразу прибавляем его к ответу
i := 1;
while (i < n) do begin
    i := 2 * i;
    ans := ans + 1;
end;
// Считаем q и сразу прибавляем его к ответу
i := 1;
while (i < m) do begin
    i := 2 * i;
    ans := ans + 1;
end;
writeln(ans);
```

Задача E. Трафарет

Автор задачи: Владимир Ульянцев
Автор разбора: Владимир Ульянцев

Эта задача относится к категории задач, для решения которых требуется аккуратно реализовать то, что описано в условии.

Сначала прочитаем из входного файла и сохраним состояния листочка Оли и трафарета Тани в двумерных массивах. В программной реализации будут использоваться следующие массивы:

- `tab[1..a, 1..b]` хранит числа, записанные на Олином листочке;
- `s[1..h, 1..w]` хранит Танин трафарет.

Теперь вычислим ответ на поставленную задачу. Для этого рассмотрим каждую клетку Олиного листочка, на которой лежал Танин, и прибавим к ответу те числа, клетки над которыми были вырезаны.

Фрагмент программы на Delphi:

```
  ans := 0;
  for i := 1 to h do begin
    for j := 1 to w do begin
      if (s[i][j] = '.') then begin
        ans := ans + tab[x - 1 + i][y - 1 + j];
      end;
    end;
  end;
  writeln(ans);
```

Время работы этой программы составляет $O(a \cdot b)$.

Задача F. Сумма

Автор задачи: Владимир Ульянцев
Автор разбора: Антон Ахи

Эта задача решается разбором случаев. Во-первых, если k четно, то и сумма будет четной. Теперь рассмотрим случай нечетного k. Как известно, сумма чисел от 1 до n составляет $\frac{n(n+1)}{2}$. Таким образом $\sum_{i=m}^{n} i = \frac{n(n+1)-m(m-1)}{2}$. Получается, что сумма будет четной тогда и только тогда, когда $n(n+1) - m(m-1)$ делится на 4.

Таким образом задача свелась к необходимости посчитать данное выражение по модулю 4.

Фрагмент решения, на языке Паскаль:

```
read(m, n, k);
if (k mod 2 = 0) or
  (((n mod 4) * ((n + 1) mod 4) -
                (m mod 4) * ((m - 1) mod 4)) mod 4 = 0) then
  writeln('YES')
else
  writeln('NO');
```

Задача G. Тетрис

Автор задачи: Федор Царев
Автор разбора: Роман Сатюков

В этой задаче требовалось аккуратно реализовать функцию подсчета очков. Для этого будем просматривать строки сверху вниз. Для каждой очередной строки будем проверять, является ли она полностью заполненной. Кроме того, будем поддерживать переменную c — количество полностью заполненных подряд идущих строк до нашей строки. Если очередная строка целиком заполнена, то увеличиваем c на единицу, иначе обнуляем c. Параллельно ведем подсчет очков.

Фрагмент решения, на языке Паскаль:

```
readln(n, m);
c := 0;
ans := 0;
for i := 1 to n do begin
  readln(s);
  // good - является ли текущая строка целиком заполненной
  good := true;
```

```
  for j := 1 to m do begin
    good := good and (s[j] = '*');
  end;
  if good then begin
    // если целиком заполнена - то увеличиваем c
    c := c + 1;
  end else begin
    // если нет - обнуляем c и прибавляем очки к ответу
    ans := ans + c * (c + 1) div 2;
    c := 0;
  end;
end;
// не забываем прибавить к ответу очки за последние строки
ans := ans + c * (c + 1) div 2;
writeln(ans);
```

Задача H. Кладоискатель

Автор задачи: Владимир Ульянцев
Автор разбора: Владимир Ульянцев

В задаче требуется выяснить, сколько элементов последовательности $a, 2a, \ldots, ba$ делится на b. Докажем, что искомое число равно наибольшему общему делителю чисел a и b.

Пусть d — наибольший общий делитель чисел a и b. Тогда $a = dr$, $b = ds$, где r и s — два взаимно простых числа.

Если все числа $a, 2a, \ldots, ba$ разделить на b, то частные можно записать в виде $\frac{r}{s}, \frac{2r}{s}, \frac{3r}{s}, \ldots, \frac{(ds-1)r}{s}, \frac{(ds)r}{s}$. Так как r и s — взаимно простые, среди частных целыми числами могут быть лишь те, у которых в числителе коэффициент при r (равный $1, 2, \ldots, ds$) делится на s. Несложно видеть, что число таких коэффициентов равно d, что и требовалось доказать.

Пример программы на Pascal:

```
var
  a, b : integer;

function gcd(a, b : integer) : integer;
begin
  if (b = 0) then
    result := a
  else
    result := gcd(b, a mod b);
end;

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

  read(a, b);
  writeln(gcd(a, b));
end.
```

Данная программа для нахождения наибольшего общего делителя использует алгоритм Евклида. Дополнительную информацию об этом алгоритме можно найти в книге [1].

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.