Разбор задач Второй Интернет-олимпиады

Введение

В базовой номинации Второй Интернет-олимпиады сезона 2008-2009 участникам было предложено для решения восемь задач. В олимпиаде приняло участие 175 команд, из них 164 решили хотя бы одну задачу.

Наиболее простой оказалась задача «C. Зайцы в клетках» — ее решили 154 команды. Наиболее сложной — задача «E. Точка и отрезок» — ее решили 38 команд.

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

Задача A. Движение с ускорением

Автор задачи: Федор Царев
Автор разбора: Федор Царев

Рассмотрим некоторый промежуток времени $[t_1, t_2]$, на котором автомобиль двигается с постоянным ускорением. Пусть начальная скорость равна V_0, а ускорение равно a. Тогда в любой момент времени $t_1 \le t \le t_2$ скорость автомобиля есть $V(t) = V_0 + a \cdot (t - t_1)$. Зададимся вопросом о поиске максимальной скорости на этом отрезке времени.

Так как зависимость скорости от времени линейна, то возможны три варианта:

- скорость возрастает ($a > 0$) — тогда скорость максимальна в момент времени t_2;
- скорость убывает ($a < 0$) — тогда скорость максимальна в момент времени t_1;
- скорость постоянна ($a = 0$) — тогда скорость максимальна в любой момент времени от t_1 до t_2.

Отсюда следует, что скорость максимальна либо в момент времени t_1, либо в момент времени t_2, то есть максимальная скорость $V_{max} = \max(V(t_1), V(t_2))$.

В рассматриваемой задаче заданы два временны́х отрезка изменения скорости: $[0, t_1]$ и $[t_1, t_2]$. Из сказанного выше следует, что на каждом из них максимальная скорость достигается в одном из его концов. Поэтому для нахождения ответа на задачу необходимо сравнить скорость V_0, скорость $V_1 = V_0 + a_1 \cdot t_1$ в момент времени t_1 и скорость $V_2 = V_1 + a_2 \cdot t_2$ в момент времени t_2.

Приведем программную реализацию.

```
uses
  SysUtils, Math;

var
  v0, t1, a1, t2, a2 : integer;
  v1, v2 : integer;

begin
  reset(input, 'accel.in');
  rewrite(output, 'accel.out');
  read(v0, t1, a1, t2, a2);

  v1 := v0 + a1 * t1;
  v2 := v1 + a2 * t2;

  writeln(max(max(v0, v1), v2));
end.
```

Время работы этого решения есть $O(1)$.

Задача B. Дигитация

Автор задачи: Максим Буздалов
Автор разбора: Максим Буздалов

Для решения этой задачи можно применить несколько подходов. Не все из них, однако, достаточно эффективны.

Первый возможный подход — это решение по определению. Оно подразумевает подсчет факториала в явном виде, используя длинную арифметику, и вычисление дигитации этого числа. Такое решение может работать лишь до значений n порядка 1000 и не является достаточно эффективным.

Для рассмотрения дальнейших подходов к решению получим некоторые интересные свойства дигитации.

Для начала заметим, что сумма цифр числа x сравнима с x по модулю 9. Докажем это утверждение по индукции. База индукции: сумма цифр числа 0 равна 0. Шаг индукции: пусть $s(x-1)$ сравнимо с $x-1$ по модулю 9. Прибавим к $x-1$ единицу, получим x. Если при этом переносов из младшего разряда не произошло, то сумма цифр также увеличится на 1. Иначе, если из m младших разрядов произошел перенос, то в числе $x-1$ все эти разряды содержали цифру 9, а в числе x эти разряды содержат 0, а в $m+1$-ом разряде цифра увеличилась на 1. Значит, и этот случай эквивалентен увеличению суммы цифр на 1 по модулю 9. Таким образом, из предположения индукции следует, что $s(x)$ сравнимо с x по модулю 9. Утверждение доказано.

Данное утверждение о сумме цифр можно распространить и на дигитацию, доказав его также по индукции. Из этого факта и из того, что из $x > 0$ вытекает $d(x) > 0$, можно получить следующие свойства дигитации:

- $d(x+y) = d(d(x)+d(y))$
- $d(x \cdot y) = d(d(x) \cdot d(y))$

Доказательство этих утверждений оставим читателю в качестве упражнения.

Отсюда, в частности, вытекает второй способ решения: рассмотрим $n!$ как произведение: $n! = 1 \cdot 2 \cdot \ldots \cdot n$ и вычислим $d(n!)$, используя второе из только что доказанных свойств дигитации. Фрагмент программы, производящей эти вычисления, может выглядеть так:

```
answer := 1;
for i := 1 to n do begin
    answer := d(answer * d(i));
end;
```

Заметим, что нет смысла вызывать `d(answer)` лишний раз, так как значение этой переменной в любой момент времени не превосходит 9.

Однако и этот способ для ограничений, данных в условии задачи, оказывается неэффективным. Его время работы оценивается как $O(n)$.

Существует и третий способ. Отметим, что если $n \geq 6$, то $n!$ делится на 9. Отсюда, учитывая сравнимость $d(x)$ и x по модулю 9, можно сразу доказать, что для $n \geq 6$ верно $d(n!) = 9$. Для маленьких значений n дигитацию можно вычислить и непосредственно. Этот способ работает за константное время и, бесспорно, достаточно эффективен, чтобы уложиться в ограничения, данные в условии.

Задача C. Зайцы в клетках

Автор задачи: Александр Торопов
Автор разбора: Федор Царев

Рассмотрим некоторое распределение зайцев по клеткам: пусть в первой клетке будут находиться a_1 зайцев, во второй — a_2, ..., в N-ой — a_N (при этом $\sum_{i=1}^{N} a_i = M$). Для этого распределения найдем максимальное число a_{max} зайцев, которое находится в одной клетке. Задача состоит в том, чтобы найти такое распределение, для которого это число a_{max} будет как можно меньше. При этом в качестве ответа в выходной файл следует вывести только минимальное значение min величины a_{max}.

Число min не может быть меньше, чем M/N (здесь и далее как / обозначена операция целочисленного деления). Докажем это методом «от противного». Предположим, что $min < M/N$. Рассмотрим распределение зайцев по клеткам, на котором $a_{max} = min$. Так как a_{max} есть максимальное число зайцев, находящихся в одной клетке, то для общего числа зайцев будет верна цепочка неравенств: $M \le a_{max} \cdot n = min \cdot n < M/N \cdot N \le M$. Полученное противоречие (получилось, что $M < M$) доказывает утверждение о том, что $min \ge M/N$.

Далее возможны два случая: либо M делится на N ($M \bmod N = 0$), либо не делится. В первом случае зайцев можно равномерно распределить по клеткам — в каждой из окажется M/N зайцев. Так как доказано, ответ не меньше этого числа, то оно и является ответом на задачу.

Во втором случае после того, как в каждую клетку было посажено M/N зайцев, останется $c = M \bmod N$ зайцев. Этих зайцев можно посадить по одному в некоторые c клеток (по свойствам деления $c < N$). Значит, в этом случае ответ равен $M/N + 1$.

Приведем фрагмент программы, реализующий описанное решение.

```
if (m mod n = 0) then begin
        writeln(m div n);
end
else begin
        writeln(m div n + 1);
end;
```

Задача D. Ленивый студент

Автор задачи: Антон Феськов
Автор разбора: Антон Феськов

Для решения этой задачи требовалось выписать все буквы, содержащие «полости», а именно: a, b, d, e, g, o, p и q. При внимательном прочтении условия оказывалось, что для этого достаточно рассмотреть последний пример из условия. После этого остается только считывать входной файл по букве и проверять на совпадение с выписанными выше.

Пример реализации на языке Паскаль:

```
ans := 0;
while (not SeekEof) do begin
    read(ch);
    if (ch = 'a') or (ch = 'b') or (ch = 'd') or (ch = 'e') or
       (ch = 'g') or (ch = 'o') or (ch = 'p') or (ch = 'q') then inc(ans);
end;
writeln(ans);
```

Здесь `ch` — переменная типа `char`, а `ans` — ответ. Алгоритм работает за время, пропорциональное длине входа с использованием $O(1)$ дополнительной памяти.

Задача E. Точка и отрезок

Автор задачи: Федор Царев
Автор разбора: Антон Феськов

Поскольку по условию отрезок параллелен одной из осей координат, то сразу возникают два случая. Разберем один из них: отрезок параллелен оси ординат. В этом случае $x_1 = x_2$. Второй случай аналогичен.

Из определения манхэттенского расстояния искомая величина есть сумма двух составляющих. Первая — перемещение по горизонтали ($|x - x_1|$), то есть расстояние от данной точки до прямой l, на которой лежит отрезок. Какую бы точку на отрезке мы ни выбрали, эта составляющая постоянна. Вторая — перемещение по вертикали, то есть расстояние от основания перпендикуляра, опущенного на прямую l, до данного отрезка. Другими словами, это расстояние от точки (x_1, y) до отрезка. Таким образом исходная задача сведена к случаю, когда и точка, и отрезок лежат на одной прямой.

Заметим, что если эта точка принадлежит отрезку (то есть $y_1 \le y \le y_2$ или $y_2 \le y \le y_1$), то расстояние от точки до отрезка равно нулю. Иначе ответ — это расстояние до одного из концов отрезка ($min(|y_1 - y|, |y_2 - y|)$). Получаем решение, состоящее из разбора нескольких случаев.

Приведем пример решения на языке Паскаль:

```
read(x, y, x1, y1, x2, y2);
if (x1 = x2) then begin
    if (y1 <= y) and (y <= y2) or (y2 <= y) and (y <= y1) then
        writeln(abs(x - x1))
    else
        writeln(abs(x - x1) + min(abs(y - y1), abs(y - y2)));
end else begin
    if (x1 <= x) and (x <= x2) or (x2 <= x) and (x <= x1) then
        writeln(abs(y - y1))
    else
        writeln(abs(y - y1) + min(abs(x - x1), abs(x - x2)));
end;
```

Здесь названия переменных соответствуют условию задачи.

Отметим, что в силу ограничений на координаты точек и концов отрезка (не превышают 10^9 по абсолютной величине), максимальный ответ может достигать 4×10^9, что превышает максимальное допустимое значение знакового 32-битного типа. Поэтому несмотря на простоту формул, необходимо было использовать либо беззнаковый 32-битный тип, либо 64-битный.

Задача F. Строчечки

Автор задачи: Максим Буздалов
Автор разбора: Максим Буздалов

Рассмотрим два случая: T является подстрокой S или не является.

В первом случае ответом будет S, так как S является подстрокой самой себя и является надстрокой T (так как T — подстрока S) и не существует других подстрок S, с длиной, не меньшей длины S.

Рассмотрим второй случай. Если T не является подстрокой S, то она не является подстрокой и произвольной подстроки S. Следовательно, множества подстрок S и надстрок T не пересекаются и ответа не существует.

Отсюда непосредственно следует решение. Необходимо проверить, является ли T подстрокой S. Если является, то вывести S; если же не является, то вывести «`NO SOLUTION`».

Поиск вхождения данной подстроки может выполняться за $O(|S| \cdot |T|)$ перебором начала вхождения подстроки. Ниже приведено решение, реализующее данный алгоритм:

```
uses sysutils;

var
    s, t: string;
    i, j: integer;
    sLen, tLen: integer;
begin
    reset(input, 'strings.in');
    rewrite(output, 'strings.out');
    readln(s);
    readln(t);
    sLen := length(s);
    tLen := length(t);
    for i := 0 to sLen - tLen do begin
        j := 1;
        while (j <= tLen) and (s[i + j] = t[j]) do begin
            inc(j);
        end;
        if j > tLen then begin
            writeln(s);
            halt;
        end;
    end;
    writeln('NO SOLUTION');
end.
```

Задача о поиске подстроки встречается достаточно часто в программировании, поэтому различные разновидности ее решений включены в стандартные библиотеки многих языков программирования. В качестве примеров приведем следующие:

- функция `strpos` в диалектах языка *Pascal*;
- функция `strstr` в языке *C*;
- методы `find` класса `std::string` в языке *C++*;
- методы `indexOf` и `contains` класса java.lang.String в языке *Java*.

Отметим, что задача о поиске подстроки (образца) в строке (тексте) может быть решена за время $O(|S| + |T|)$ с помощью алгоритма Кнута-Морриса-Пратта или ему подобных алгоритмов. Об этом и других алгоритмах для работы со строками можно прочитать в [1].

Задача G. Счастливый билетик - 2

Автор задачи: Владимир Ульянцев
Автор разбора: Владимир Ульянцев

Для решения задачи сначала посчитаем сумму цифр sum всего билетика. Затем рассмотрим последовательно каждую границу билетика и проверим, является ли она счастливой. По определению это значит, что сумма цифр $tsum$ до границы равна сумме цифр $sum - tsum$ после нее.

Заметим, что сумма цифр $tsum$ до текущей границы i отличается на a_i от суммы цифр до границы $i - 1$. Таким образом, пересчет суммы цифр до текущей границы выполняется за $O(1)$. Из этого следует, что проверка каждой границы на требуемое свойство выполняется за $O(1)$ действий, а время работы программы составляет $O(n)$.

Приведем программную реализацию на языке Pascal:

```
const
  maxn = 1000000;

var
  i, n, sum, tsum, ans : longint;
  a : array [1..maxn] of longint;

begin
  assign(input, 'ticket2.in');
  reset(input);
  assign(output, 'ticket2.out');
  rewrite(output);

  read(n);
  sum := 0;
  for i := 1 to n do begin
    read(a[i]);
    sum := sum + a[i];
  end;

  tsum := 0;
  ans := -1;
  for i := 1 to n do begin
    tsum := tsum + a[i];
    if ((tsum = sum - tsum) and (ans = -1)) then
      ans := i;
  end;

  writeln(ans);
end.
```

Задача H. Транспортные узлы

Автор задачи: Федор Царев
Автор разбора: Федор Царев

Формулировка этой задачи на языке теории графов такова. Задана граф, содержащий n вершин (соответствуют городам) и m ребер (соответствуют дорогам). Необходимо найти все вершины этого графа, степень которых больше либо равна k. Напомним, что степенью вершины называется число инцидентных ей ребер. Более подробно о графах можно прочитать в [2].

Несмотря на такую «графовую» формулировку задачи, для ее решения не требуется знание каких-либо специальных алгоритмов. Необходимо вычислить степени всех вершин, а затем найти вершины, удовлетворяющие указанном свойству.

Для вычисления степеней вершин создадим массив `deg`, i-ый элемент которого `deg[i]` будет содержать степень вершины i. Для заполнения этого массива при чтении очередного ребра `u v` необходимо увеличить на единицу значение `deg[u]` и `deg[v]`.

Далее, необходимо найти число вершин со степенью хотя бы k и номера таких вершин.

Приведем программную реализацию этого алгоритма.

```
read(n, m);
for i := 1 to m do begin
  read(u, v);
```

```
    inc(deg[u]);
    inc(deg[v]);
end;
read(k);
cnt := 0;
for i := 1 to n do begin
    if (deg[i] >= k) then begin
        inc(cnt);
    end;
end;
writeln(cnt);
for i := 1 to n do begin
    if (deg[i] >= k) then begin
        write(i, ' ');
    end;
end;
writeln;
```

Время работы этого алгоритма составляет $O(n + m)$.

Список литературы

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

[2] *Харари Ф.* Теория графов. Перевод с английского. — М.: УРСС, 2006.