Разбор задач Восьмой Интернет-олимпиады

Автор: Федор Царев

Введение

В базовой номинации Восьмой Интернет-олимпиады сезона 2007-2008 участникам было предложено для решения 8 задач. В олимпиаде приняло участие 100 команд, из них 86 решили хотя бы одну задачу.

Наиболее простой оказалась задача «A. Красивые числа» — ее решили 83 команды. Наиболее сложной — задача «G. К коду Грея» — ее решили 2 команды.

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

Задача A. Красивые числа

Для начала напишем функцию, определяющую, является ли поступающее ей на вход число x красивым. Для этого воспользуемся алгоритмом перевода в k-ичную систему счисления при $k = 10$.

```
function isBeautiful(x : integer) : boolean;
var
  cnt : integer;
  sum : integer;
begin
  cnt := 0;
  sum := 0;
  while (x > 0) do begin
    inc(cnt);
    sum := sum + (x mod 10);
    x := x div 10;
  end;
  result := (sum mod cnt = 0);
end;
```

Далее, напишем программу, последовательно проверяющую все натуральные числа, и находящую таким образом n-ое красивое число.

```
readln(n);
i := 1;
while (true) do begin
  if (isBeautiful(i)) then begin
    n := n - 1;
  end;
  if (n = 0) then begin
    writeln(i);
    break;
  end;
  i := i + 1;
end;
```

Время работы этой программы составляет $O(x \log x)$, где x — n-ое красивое число. Поэтому она будет укладываться в ограничения по времени только, если максимально возможный ответ не очень большой. Эксперимент, проведенный с помощью этой программы показывает, что при $n = 100000$ ответ равен 576396, а такое решение укладывается в две секунды с достаточно большим запасом.

Задача B. Числа Фибоначчи

Ключом к решению данной задачи является формула $GCD(F_n, F_m) = F_{GCD(n,m)}$, где как $GCD(a, b)$ обозначается наибольший общий делитель чисел a и b. Для доказательства (которое проводится методом, указанным в [2]) этой формулы мы воспользуемся тремя леммами, которые могут быть доказаны по индукции. Их доказательство остается читателю в качестве упражнения.

Лемма 1. $GCD(F_n, F_{n+1}) = 1$, то есть два последовательных числа Фибоначчи взаимно просты.

Лемма 2. $F_{m+n} = F_{m+1}F_n + F_mF_{n-1}$.

Лемма 3. Если n делится на m, то F_n делится на F_m.

В соответствии с алгоритмом Евклида поиска наибольшего общего делителя чисел n и m $GCD(n, m) = GCD(m, r)$, где как r обозначен остаток от деления n на m ($n = qm + r$, где q — частное от деления).

Тогда:

$$GCD(F_m, F_n) = GCD(F_m, F_{qm+r}) = GCD(F_n, F_{qm+1}F_r + F_{qm}F_{r-1}) =$$
$$= GCD(F_m, F_{qm+1}F_r) = GCD(F_m, F_r).$$

Второе равенство справедливо по лемме 2, третье — по лемме 3 (так как qm делится на m), а четвертое — по лемме 1 (F_{qm} и F_{qm+1} взаимно просты).

Таким образом, доказана формула $GCD(F_n, F_m) = GCD(F_m, F_{n \bmod m})$, аналогичная основной формуле алгоритма Евклида. Если продолжить такое же рассуждение далее (аналогично алгоритму Евкида), то придем к доказываемой формуле $GCD(F_n, F_m) = F_{GCD(n,m)}$.

Задача C. Развлечения с измерителем — 2

Для решения этой задачи можно, например, записать все попарные расстояния между точками в массив, упорядочить его по возрастанию, удалить повторяющиеся элементы и вывести этот массив. Для того, чтобы понять, какой метод сортировки применим, найдем максимальное количество попарных расстояний между точками. Если задано n точек, то пар точек $\frac{n(n-1)}{2}$, что при заданных в условии задачи ограничениях не превосходит $\frac{50 \cdot 49}{2} = 1225$.

Таким образом, понятно, что даже использование метода сортировки за квадратичное (например, сортировки «пузырьком» или вставками [1]) время позволяет уложиться в установленные ограничения по времени.

Также отметим, что в этой задаче удобно оперировать не с расстояниями, а с их квадратами (они являются целыми числами), а квадратный корень извлекать только при выводе ответа. Более того, при менее жестких ограничениях на координаты точек (например, если разрешить координаты до 10^9 по абсолютному значению) существует наборы входных данных, на которых решение, работающее непосредственно с расстояниями дает неправильный ответ, а решение, оперирующее с квадратами, работает правильно. Примером такого набора данных является

```
3
-1000000000 -1000000000
977545001 977544900
997320350 997320450
```

На этом примере квадраты расстояний от первой точки до второй и третьей равны 7821368062496100001 и 7978577560512325000. Такие числа «помещаются» в 64-битные типы данных, существующие в языках программирования. Если же вычислить (с использованием даже 10-байтного вещественного типа данных) квадратные корни из этих чисел, то они из-за погрешности, возникающей вследствие округления, будут равны с точки зрения компьютера.

Задача D. Генерация тестов

Для решения этой задачи могут применяться различные методы. Опишем один из них. Для начала, напишем функцию, проверяющую, есть ли среди поступившего ей на вход множества точек три точки, лежащие на одной прямой. Для этого просмотрим все тройки точек P_i, P_j, P_k и проверим верно ли, что векторы P_iP_j и P_iP_k коллинеарны. Для этого необходимо проверить пропорциональность координат этих векторов, то есть равенство

$$\frac{x_j - x_i}{y_j - y_i} = \frac{x_k - x_i}{y_k - y_i}.$$

Однако при непосредственной проверке этого равенства может произойти деление на ноль, поэтому вместо него лучше проверять равенство

$$(x_j - x_i) \cdot (y_k - y_i) = (x_k - x_i) \cdot (y_j - y_i).$$

Для того, чтобы решить теперь исходную задачу, сгенерируем набор из n точек со случайными координатами, а потом их проверим с помощью указанного выше метода на отсутствие трех точек, лежащих на одной прямой. Если это условие выполняется, то выведем этот набор в качестве ответа. В противном случае повторим процесс с начала — сгенерируем новый набор и его проверим.

Если реализовать этот алгоритм и запустить его с максимально возможным значением $n = 300$, то можно видеть, что искомый набор точек будет достаточно быстро построен. Так как любой поднабор этого набора также удовлетворяет требованию отсутствия трех точек на одной прямой, то искомый набор существует при любом n от 1 до 300, а `NO` выводить никогда не надо.

Читатели, заинтересовавшиеся оценкой вероятности успеха этого алгоритма, могут найти дополнительную информацию в [3], [4], [5], [6].

Задача E. Левая рекурсия

В этой задаче требуется читать строки по одной из входного файла и считать количество строк, в которых первый символ совпадает с четвертым. Рассматривать случай, когда в строке нет четвертого символа, не нужно, так как по условию правые части всех продукций непусты. Этот алгоритм реализует следующий фрагмент программы.

```
readln(n);
ans := 0;
for i := 1 to n do begin
  readln(s);
  if (s[1] = s[4]) then begin
    inc(ans);
  end;
end;
writeln(ans);
```

Задача F. Матрица

Самый простой алгоритм, который приходит в голову при прочтении этой задачи, состоит в том, чтобы перебрать все элементы матрицы и для каждого из них проверить верно ли, что он не больше остальных элементов своей строки и не меньше остальных элементов своего столбца. Однако время работы этого алгоритма есть $O(nm(n+m))$, поэтому при заданных в условии задачи ограничениях он не укладывается в ограничения по времени.

Поэтому необходимо придумать более быстрый алгоритм. Заметим, что для того, чтобы проверить, является ли элемент матрицы минимальным в своей строке, не требуется его сравнивать

со всеми элементами строки, а достаточно сравнить только с минимальным из них (на равенство). Для столбцов можно действовать аналогично. На поиск минимумов в каждой строке и максимумов в каждом столбце требуется время, равное $O(nm)$. После их нахождения каждая операция проверки того, является ли элемент седловой точкой, занимает $O(1)$ времени. Таким образом, суммарное время работы составляет $O(nm)$, что позволяет уложиться в ограничения по времени при указанных в условии ограничениях.

Задача G. К коду Грея

Первым шагом к решению задачи является написание функции, определяющей, могут ли два числа a и b стоять рядом в коде Грея. Для этого необходимо проверить, что они отличаются ровно в одном бите. Заметим, что операция `xor` (побитовое «или», $\oplus$) выдает единицу, если ее аргументы различны, и ноль, если совпадают. Таким образом, необходимо проверить, что в побитовом «или» чисел a и b только один бит является единичным. Иными словами, $a \oplus b$ является степенью двойки. Поскольку в задаче приходится иметь дело максимум с шестнадцати-битными числами, то эту проверку можно организовать с помощью массива `isTwoPower[0..65355]`, i-ый элемент которого содержит значение булевого типа, обозначающее, является ли число i степенью двойки. Заполнить его можно, например, таким образом.

```
t := 1;
for i := 0 to 16 do begin
  isTwoPower[t] := true;
  t := t * 2;
end;
```

Самый простой способ решения этой задачи (после каждого изменения проверять все пары соседних) элементов приводит к времени работы равному $O(m2^n)$, что не позволяет уложиться в указанные в условии ограничения по времени. Поэтому необходимо придумать более быстрый алгоритм.

Для этого найдем число пар соседних элементов, в которых нарушается свойство кода Грея (два соседних элемента отличаются в одном бите), и обозначим его как K. Если $K = 0$, то последовательность чисел является кодом Грея (обратное утверждение тоже верно). Для исходной последовательности K можно вычислить за $O(2^n)$ операций.

Предложим способ пересчитывать K при смене местами двух чисел: a_i и a_j. Пусть функция `diff(a, b)` возвращает ноль, если a и b отличаются ровно в одном бите, и ноль — иначе. Тогда формула для пересчета K будет такой:

$$K = K - diff(a_i, a_{i-1}) - diff(a_i, a_{i+1}) - diff(a_j, a_{j-1}) - diff(a_j, a_{j+1}) +$$

$$+ diff(a_i, a_{j-1}) + diff(a_i, a_{j+1}) + diff(a_j, a_{i-1}) + diff(a_j, a_{i+1}).$$

Слагаемые со второго по пятое соответствуют тому, что числа a_i и a_j убираются с i-ого и j-ого места, соответственно, а с шестого по девятое — тому, что a_i ставится в j-ую позицию, а a_j — в i-ую. Как a_{i-1} обозначено число, являющееся левым соседом a_i, а как a_{i+1} — число, являющееся его правым соседом. Для a_0 левым соседом является a_{2^n-1}, и, наоборот, для числа a_{2^n-1} правым соседом является число a_0.

Задача H. Треугольная область

Для решения этой задачи необходимо найти расстояния от заданной точки до каждой из сторон треугольника, и выбрать из них минимальное. Так как треугольник не содержит тупых углов, то расстояние от точки, находящейся внутри этого треугольника, до его стороны равно расстоянию от этой точки до прямой, содержащей сторону (перпендикуляр из точки на сторону падает внутрь этой стороны).

Поэтому достаточно научится вычислять расстояние от точки до прямой. Пусть заданы две точки на прямой: $A(x_1, y_1)$ и $B(x_2, y_2)$ и точка $C(x_0, y_0)$, расстояние от которой до прямой требуется найти. Искомое расстояние можно вычислить несколькими способами. Рассмотрим два из них.

Первый из них состоит в том, что искомое расстояние равно длине высоты, проведенной из точки C, в треугольнике ABC. Эту высоту можно найти по формуле $h = \frac{2S}{a}$, где как S обозначена площадь треугольника, а как a — длина стороны AB. Саму площадь можно найти, например, по формуле Герона $S = \sqrt{p(p-a)(p-b)(p-c)}$, где $p = \frac{a+b+c}{2}$ — полупериметр треугольника, а a, b и c — соответственно длины сторон AB, BC и CA.

Второй заключается в использовании формулы для расстояния от точки до прямой, известной из аналитической геометрии — расстояние от прямой, заданной уравнением $Ax + By + C = 0$, до точки с координатами (x_0, y_0) равно

$$\frac{|Ax_0 + By_0 + c|}{\sqrt{A^2 + B^2}}.$$

Осталось научиться строить прямую, проходящую через две точки (x_1, y_1) и (x_2, y_2). Ее коэффициенты вычисляются по формулам

$$A = y_2 - y_1,$$

$$B = x_1 - x_2,$$

$$C = -Ax_1 - By_1.$$

Список литературы

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

[2] Su, Francis E., et al. «Fibonacci GCD's, please.» Mudd Math Fun Facts. http://www.math.hmc.edu/funfacts

[3] Number of lines through at least 2 points of an n X n grid of points in The On-Line Encyclopedia of Integer Sequences http://www.research.att.com/~njas/sequences/A018808

[4] No-Three-in-a-Line-Problem in Wolfram Mathworld. http://mathworld.wolfram.com/No-Three-in-a-Line-Problem.html

[5] A. Flammenkamp, Progress in the no-three-in-line problem. http://wwwhomes.uni-bielefeld.de/achim/no3in/readme.html

[6] No-3-in-line problem: number of ways of placing 2n points on n X n grid so no 3 are in a line in The On-Line Encyclopedia of Integer Sequences. http://www.research.att.com/~njas/sequences/A000769