Разбор задач Девятой Интернет-олимпиады

Авторы: Федор Царев, Антон Феськов, Владимир Ульянцев, Антон Ахи

Введение

В базовой номинации Девятой Интернет-олимпиады сезона 2007-2008 участникам было предложено для решения 8 задач. В олимпиаде приняло участие 90 команд, из них 75 решили хотя бы одну задачу.

Наиболее простой оказалась задача «E. Произведение цифр» — ее решили 67 команд. Наиболее сложной — задача «A. Взлом хеш-функции» — ее решили 6 команд.

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

Задача A. Взлом хеш-функции

Для решения этой задачи могут применяться различные методы. Опишем один из них. Для начала, напишем функцию, вычисляющую значение полиномиальной хеш-функции заданной строки.

```
function hash(s : string) : int;
var
  i : int;
  cx : int;
begin
  result := 0;
  cx := 1;
  for i := 1 to length(s) do begin
    result := (result + (ord(s[i]) - ord('0')) * cx) mod m;
    cx := (cx * x) mod m;
  end;
end;
```

Для того, чтобы решить теперь исходную задачу, сгенерируем случайную строку из цифр от 0 до 9 длины l. Если значение хеш-функции от нее равно v, то выведем ее в ответ, иначе — повторим процесс с начала.

Ключевым фактом, обосновывающим работоспособность описанного алгоритма, является то, что полиномиальная хеш-функция при взаимно простых m и x обладает свойством *равномерности хеширования*. Это свойство заключается в том, что количества различных строк длины l, для которых значения хеш-функции равны i, примерно равны при различных i. Обсуждение этого вопроса читатель может найти в [1].

Оценим вероятность того, что искомая строка будет найдена в течение k шагов. Вероятность того, что строка не будет найдена за один шаг равна $1 - \frac{1}{m}$, вероятность того, что на каждом из k шагов она не будет найдена равна $(1-\frac{1}{m})^k$. Значит, искомая вероятность равна $1-(1-\frac{1}{m})^k$. Оценим количество попыток, которое можно успеть сделать за две секунды. Поскольку каждый шаг состоит в вычислении одного значения хеш-функции, то он занимает $O(L)$ операций. Если предположить, что компьютер может за две секунды выполнить 50 миллионов арифметических операций, то для максимального значения $L = 100$ получаем $k = 500000$. Подставляя в выведенную выше формулу максимальное значение $m = 256$, получаем $1 - (1 - \frac{1}{m})^k = 1 - (1 - \frac{1}{256})^{500000}$. Так как $(1 - \frac{1}{x})^x \le \frac{1}{e}$, то

$$1 - (1 - \frac{1}{256})^{500000} \ge 1 - ((1 - \frac{1}{256})^{256})^{1953} \ge 1 - (\frac{1}{e})^{1953}.$$

Таким образом, вероятность того, что ответ не будет найден за две секунды не превышает $(\frac{1}{e})^{1953}$, а столь малой вероятностью вполне можно пренебречь.

Задача B. Минимальный лексикографически циклический сдвиг перестановки

Заметим, что из всех циклических сдвигов любой перестановки минимальным лексикографически является тот, который начинается с единицы. Поэтому задача сводится к тому, чтобы найти позицию единицы в перестановке и вывести соответствующий циклический сдвиг.

Это можно сделать, например, так.

```
onePos := -1;
for i := 1 to n do begin
  if (p[i] = 1) then begin
    onePos := i;
    break;
  end;
end;
for i := onePos to n do begin
  write(p[i], ' ');
end;
for i := 1 to onePos - 1 do begin
  write(p[i], ' ');
end;
```

Отметим также, что существует алгоритм поиска лексикографически минимального циклического сдвига за линейное время, который работает для строк, в которых каждый символ встречается произвольное число раз (а не один раз, как в перестановке). Этот алгоритм основан на алгоритме Дюваля, с которым можно ознакомиться в [3].

Задача C. Плейлист

Прежде всего, в этой задаче необходимо занумеровать песни, так как работа со строками является крайне неудобной. Занумеровать композиции можно как в порядке появления во входном файле, так и в лексикографическом порядке. Так же необходимо научиться по названию песни быстро узнавать её номер, для чего можно использовать хэш-функцию или искать название двоичным поиском в предварительно отсортированном массиве, содержащем названия всех песен.

Далее требуется с помощью двух указателей (один на массив, содержащий времена переключений, другой на массив, содержащий порядок исполнения песен) установить на каких песнях произошло переключение и, зная их номера, отнять от рейтинга единицу. К рейтингу всех песен, во время которых не было переключения, следует не забывать прибавлять единицу.

Оценка времени работы при использовании хэш-функции $O(n+k+m+s)$, где s — суммарная длина всех названий. При использовании сортировки и двоичного поиска время работы составляет $O(pn\log n + pm\log n + k)$, где p — максимальная длина названия.

Пример программы на языке Паскаль:

1. Функция двоичного поиска в массиве с названиями песен в лексикографическом порядке:

```
function BinarySearch(s : string) : longint;
var
  l, r, m : integer;
begin
  l := 0;
  r := n;
  while (r - l > 1) do
  begin
    m := (l + r) div 2;
    if (a[m] <= s) then
```

```
      l := m
    else
      r := m;
  end;
  Result := l;
end;
```

2. Основная часть программы:

```
Begin
  ...
  i := 0;
  j := 0;
  time := 0;
  while (i < m) do
  begin
    if ((j < k) and (time + length[song[i]] > skip[j])) then
    begin
      time := skip[j];
      Dec(rating[song[i]]);
      Inc(i);
      Inc(j);
    end else
    begin
      Inc(time, length[song[i]]);
      Inc(rating[song[i]]);
      Inc(i);
    end;
  end;
  ...
End.
```

Задача D. Апельсины

Обозначим количество долек, на которое будет разрезан каждый апельсин как x. Тогда суммарное количество долек будет равно mx. Для того, чтобы эти дольки можно было поровну разделить между n людьми mx должно делиться на n. Таким образом, mx должно быть минимальным числом, которое делится на n и на m. Таким числом является наименьшее общее кратное чисел n и m. Таким образом, ответ на задачу равен $x = \frac{LCM(n,m)}{m}$, где как $LCM(n,m)$ обозначено наименьшее общее кратное (least common multiple) чисел n и m.

Докажем, что наименьшее общее кратное чисел n и m равно частному от деления nm на их наибольший общий делитель. Рассмотрим все простые делители числа nm: p_1, p_2, ..., p_k. Обозначим степень, с которой p_i входит в число n как α_i, а в число m — как β_i. Тогда наибольший общий делитель чисел n и m равен $GCD(n,m) = \Pi_{i=1}^{k} p_i^{min(\alpha_i,\beta_i)}$, а их наименьшее общее кратное — $LCM(n,m) = \Pi_{i=1}^{k} p_i^{max(\alpha_i,\beta_i)}$.

Так как для любых двух чисел a и b верно равенство $a + b = min(a,b) + max(a,b)$, то

$$LCM(n,m) \cdot GCD(n,m) = \Pi_{i=1}^{k} p_i^{min(\alpha_i,\beta_i)+max(\alpha_i,\beta_i)} = \Pi_{i=1}^{k} p_i^{\alpha_i+\beta_i} = nm.$$

Задача E. Произведение цифр

Напишем функцию, определяющую, делится ли число x на произведение его цифр.

```
function divides(x : integer) : boolean;
var
  product, tmp : integer;
begin
  tmp := x;
  product := 0;
  while (tmp > 0) do begin
    product := product * (tmp mod 10);
    tmp := tmp div 10;
  end;
  if (product = 0) then begin
    result := false;
  end else begin
    result := x mod product = 0;
  end;
end;
```

Время работы этой функции пропорционально количеству цифр в десятичной записи числа x, которое примерно равно его десятичному логарифму. Для чисел не превосходящих 10^9 количество цифр не превосходит 10. Так как по условию задачи $r-l \leq 10^5$, то алгоритм, основанный на переборе всех чисел заданного отрезка будет совершать порядка 10^6 операций, и, значит, программа, его реализующая должна укладываться в ограничения по времени.

Задача F. Точки и отрезки

Первая идея, которая приходит в голову после прочтения задачи — проверить каждую точку на принадлежность всем отрезкам. Тогда время работы программы составит $O(nm)$, и такое решение не будет укладываться в ограничение по времени. Поэтому необходимо придумать более быстрый алгоритм. Возможны несколько различных подходов, опишем два из них.

Первый подход заключается в обработке каждой точки отдельно от остальных. Определим для точки x количество отрезков, в которых она содержится. Для этого достаточно знать количество отрезков, начала которых находятся не правее x, а концы — не левее. Отсортировав массивы, содержащие координаты начал и концов отрезков, двоичным поиском (описанным в [2]) можно найти эти величины. Заметим, что для сортировки массивов в рамках ограничений задачи не подходят алгоритмы, работающие за $O(n^2)$, поэтому необходимо использовать более быстрый алгоритм, например, алгоритм сортировки слиянием. Таким образом программа работает за $O((m+n)\log n)$ (время сортировки и время двоичного поиска для m точек).

Для удобства описания второго подхода введем некоторые обозначения. Пусть B — совокупность точек, являющихся началами отрезков. E — совокупность точек, являющихся концами отрезков. P — совокупность точек, для каждой из которых необходимо определить количество отрезков, в которых она содержится. Переберем все целые точки числовой прямой в порядке возрастания координаты. Если в любой момент времени для текущей точки мы сможем определить, в скольких отрезках она содержится (обозначим эту величину k), то сможем это сделать и для любой точки из P. Встретив в процессе перебора точку из B, увеличиваем k на единицу, встретив из E — уменьшаем. При этом не имеет смысла рассматривать точки, не входящие в B, E или P, так как их обработка не повлечет изменение числа k. Таким образом, рассматривая точки из B, E или P по порядку возрастания координаты, можем решить поставленную задачу. Самый быстрый способ это сделать — отсортировать массив, состоящий из всех точек совокупностей B, E и P. Оценка времени работы программы в данном случае равна оценке времени сортировки, то есть $O((n+m)\log(n+m))$.

Сравним изложенные выше решения. Время работы первого алгоритма меньше, однако, при данных в условии ограничениях на n и m, это превосходство неощутимо.

Однако, у первого решения есть преимущество перед вторым. Для наглядности введем понятия «онлайн» и «оффлайн». Пусть есть какие-то данные и какие-то запросы к этим данным. *Онлайн*

алгоритм отвечает на каждый запрос, не используя информацию об остальных запросах. *Оффлайн* алгоритм заранее получает информацию обо всех запросах и может это использовать, чтобы быстро решить все задачи сразу.

В данной задаче запрос — определение для точки количества отрезков, в которых она содержится, данные — координаты начал и концов отрезков. Первый алгоритм решения задачи — онлайн, так как в первоначальных расчетах (сортировке массивов) не использует свойства исходных точек. Второй алгоритм — оффлайн, так как все исходные точки задействованы в сортировке массива, и невозможно будет решить задачу, добавив хотя бы одну исходную точку.

Задача G. Дана строка...

Первое, что приходит в голову при решении этой задачи — перебрать все подстроки и для каждой вычислить ее качество. Несложно видеть, что если n — длина данной строки, то время работы такого алгоритма — $O(n^3)$, что не укладывается в ограничения по времени. Поэтому для решения этой задачи необходимо исследовать структуру искомой подстроки. Сделаем несколько наблюдений.

Наблюдение 1. Качество подстроки не превосходит качества самой строки.

Мы можем заранее узнать, какие именно символы будут задавать качество строки: этими символами будут максимальный и минимальный символы в строке. Назовем эти символы «особыми». Разобьем их на два типа: минимальные отнесем к одному, а максимальные — к другому. В искомой строке должны содержаться символы обоих типов.

Исходя из этого можно составить более быстрый алгоритм. Переберем позицию начала искомой подстроки. Затем переберем в порядке возрастания длины всех подстрок, начинающихся в данной позиции, храня при этом максимальный и минимальный элементы в текущей подстроке. При добавлении очередного символа сравниваем его с текущим максимальным и минимальным символами и в случае необходимости обновляем их. Разность между номерами в алфавите максимального и минимального символов в текущей подстроке и есть ее качество. Время работы такого алгоритма $O(n^2)$, что также не укладывается в ограничения по времени. Поэтому необходимо сделать еще одно наблюдение.

Наблюдение 2. В искомой подстроке особые символы — первый и последний, а другие ее символы особыми не являются.

Рассмотрим подстроку, в которой содержатся оба типа особых символа. Пусть первый символ не является особым. Если мы его отбросим, то качество подстроки не уменьшится, так как разность между номерами особых символов заведомо не меньше разности между номерами любых двух символов, содержащихся в строке. Таким образом находим более короткую подстроку с таким же качеством. Кроме того, если особый символ находится не на краю строки, то либо слева либо справа от него найдется особый символ второго типа. Таким образом, взяв подстроку, концы которой являются этими элементами, получаем более короткую строку с таким же качеством.

Итак, достаточно перебрать все подстроки, концы которых являются особыми символами разных типов, внутри которых не содержатся другие особые символы.

Приведенных рассуждений достаточно, чтобы построить эффективный алгоритм. Сначала ищем максимальный и минимальный символы в строке. Затем перебираем подряд символы в строке, запоминая номера последнего максимального и последнего минимального символов. Пусть мы взяли максимальный символ. Обновляем номер последнего максимального символа. Если до этого уже встречались минимальные символы, сравниваем длину подстроки, образованной текущим символом и последним минимальным символом, с текущим ответом. Аналогично разбирается случай, когда текущий символ — минимальный. Остальные символы игнорируются.

При просмотре каждого символа совершается $O(1)$ операций, поэтому суммарное время работы программы $O(n)$, где n — количество элементов в строке.

Фрагмент программы, перебирайщий нужные подстроки:

```
lastMin := -1;
lastMax := -1;
ansLen := 1000000;
```

```
ansPos := -1;
for i := 1 to n do begin
  if (s[i] = max) then begin
    if (lastMin <> -1) then begin
      sl := i - lastMin + 1;
      if (sl < ansLen) then begin
        ansLen := sl;
        ansPos := lastMin;
      end;
    end;
    lastMax := i;
  end;
  if (s[i] = min) then begin
    if (lastMax <> -1) then begin
      sl := i - lastMax + 1;
      if (sl < ansLen) then begin
        ansLen := sl;
        ansPos := lastMax;
      end;
    end;
    lastMin := i;
  end;
end;
```

Здесь `s` - строка, `min` - минимальный символ в строке, а `max` - максимальный.

Задача H. Сумма

Попробуем решить задачу методом перебора. Первая мысль, которая приходит в голову при обдумывании такого подхода, состоит в том, что придется перебирать четверки чисел, каждое из которых может быть порядка 1500, поэтому придется выполнить порядка 1500^4 операций, что точно не уложится в ограничения по времени.

Заметим, что поскольку $x = a+b+c+d$, то $d = x-a-b-c$, то есть если зафиксировать числа a, b и c, то значение d будет определено однозначно. Таким образом, достаточно перебирать только тройки чисел a, b, c. Приведем фрагмент программы, реализующий этот подход, и оценим время его работы.

```
ans := 0;
for a := 1 to x do begin
  for b := a to x - a do begin
    for c := b to x - a - b do begin
      d := x - a - b - c;
      if (d >= c) then begin
        inc(ans);
      end else begin
        break;
      end;
    end;
  end;
end;
```

Оценим сверху количество четверок чисел вида $(a, b, c, x-a-b-c)$, в которых все числа положительны, не превосходят x и идут в неубывающем порядке. Всего различных четверок вида $(a, b, c, x-a-b-c)$ не более x^3. Рассмотрим только четверки, в которых все числа различны. В

каждой такой четверке числа можно переставить $4! = 24$ разными способами, однако только один из них (в котором они упорядочены по возрастанию) нам подходит. Таким образом, количество требуемых нам четверок не превосходит $\frac{x^3}{24}$.

Рассмотрим теперь четверки, в которых есть одинаковые числа. На каждую такую четверку (a, b, c, d) кроме ограничения $x = a + b + c + d$ будет наложено еще несколько ограничений (например, $a = b$). Таким образом, уменьшается количество переменных, значения которых можно варьировать («свободных» переменных). Значит, таких четверок порядка x^2.

Время работы описанного метода пропорционально количество четверок чисел вида $(a, b, c, x - a - b - c)$, в которых все числа положительны, не превосходят x и идут в неубывающем порядке, которое не превосходит $\frac{x^3}{24}$. Для $x = 1500$ это число составляет 140625000, а такое количество достаточно простых операций (прибавление единицы) укладывается в ограничение по времени.

Список литературы

[1] *Кнут Д.* Искусство программирования, том 3. Сортировка и поиск, 2-е изд. : Пер. с англ. — М.: Вильямс, 2007.

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

[3] *Смит Б.* Методы и алгоритмы вычислений на строках: Пер. с англ. — М.: Вильямс, 2006.