Разбор задач Восьмой Интернет-олимпиады

Автор: Федор Царев

Введение

В базовой номинации Восьмой Интернет-олимпиады сезона 2006-2007 участникам было предложено для решения 8 задач. В олимпиаде приняло участие 84 команды, из них 79 решили хотя бы одну задачу.

Наиболее простой оказалась задача «G. От перестановки чисел что-то меняется . . . » — ее решили 77 команд. Наиболее сложной — задача «A. Ничего и не терялось» — ее решили 4 команды.

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

Задача A. Ничего и не терялось

Решение задачи состоит из двух частей. Первая из них — чтение входного файла и выделение необходимой информации.

Вторая часть состоит в выполнении запросов. Заметим, что применение сортировки в лексикографическом порядке при выводе приведет в худшем случае к времени работы $O(n^2 logn)$, что не удовлетворяет ограничениям по времени. Заметим, что поиск десяти минимумов приведет к весьма близкой оценке времени работы, так как $10 = log_2 1024$.

Для того, чтобы избежать сортировки при выводе, необходимо хранить для каждого ключевого слова отсортированный список сайтов на которых оно встречается.

При этом вставка будет производиться за линейное время, вывод результатов запроса — за константное.

Таким образом, достигнута оценка времени работы $O(n^2)$, что уже укладывается в ограничения по времени.

Оценки $O(n\,logn)$ можно было добиться, применив сбалансированные деревья поиска, входящие в стандартные библиотеки языков *Java* и *C++*. При таких ограничениях этого, однако, не требовалось.

Приведем текст решения на языке *Delphi*.

```
program giggle_insertion;

uses sysutils;

type Tokenizer = object
    data: string;
    pos: integer;
    len: integer;

    constructor create(var s: string);
    function next: string;
end;

constructor Tokenizer.create(var s: string);
begin
    data := s;
    pos := 0;
    len := length(data);
end;

function Tokenizer.next: string;
```

```
var
    i: integer;
begin
    i := pos + 1;
    while (i <= len) and (data[i] <> ' ') do inc(i);
    result := copy(data, pos + 1, i - pos - 1);
    pos := i;
end;

var
    kwds:  array [1..2500] of string;
    sites: array [1..2500, 1..2500] of string;
    sizes: array [1..2501] of integer;

    line, command, keyword, site: string;
    ln: Tokenizer;

    i, j, k: integer;

    n: integer;
    kwdCnt: integer;
    cont: boolean;

function find(var s: string): integer;
var
    i: integer;
begin
    result := kwdCnt + 1;
    for i := 1 to kwdCnt do begin
        if kwds[i] = s then begin
            result := i;
            exit;
        end;
    end;
end;

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

    readln(n);
    kwdCnt := 0;

    for k := 1 to n do begin
        readln(line);
        ln.create(line);
        command := ln.next;
        if command = 'Add' then begin
            ln.next; //keyword
            keyword := ln.next;
            ln.next; //to
            site := ln.next;
```

```
        j := find(keyword);
        if j > kwdCnt then begin
            inc(kwdCnt);
            kwds[j] := keyword;
        end;

        cont := true;
        for i := 1 to sizes[j] do begin
            if sites[j][i] = site then begin
                cont := false;
                break;
            end;
        end;

        if cont then begin
            inc(sizes[j]);
            i := sizes[j];
            while (i > 1) and (sites[j][i - 1] > site) do begin
                sites[j][i] := sites[j][i - 1];
                dec(i);
            end;
            sites[j][i] := site;

            writeln('OK');
        end else begin
            writeln('Already exists');
        end;
    end else if command = 'Remove' then begin
        ln.next; //keyword
        keyword := ln.next;
        ln.next; //to
        site := ln.next;

        j := find(keyword);
        if j > kwdCnt then begin
            writeln('Not found');
        end else begin
            i := 1;
            while (i <= sizes[j]) and (sites[j][i] <> site) do inc(i);
            if i > sizes[j] then begin
                writeln('Not found');
            end else begin
                while (i < sizes[j]) do begin
                    sites[j][i] := sites[j][i + 1];
                    inc(i);
                end;
                dec(sizes[j]);

                writeln('OK');
            end;
        end;
```

```
        end else if command = 'Search' then begin
            keyword := ln.next;

            j := find(keyword);

            writeln('Results: ', sizes[j], ' site(s) found');
            i := 1;
            while (i <= 10) and (i <= sizes[j]) do begin
                writeln (i, ') ', sites[j][i]);
                inc(i);
            end;
        end;
        if k <> n then writeln('=======');
    end;
end.
```

Задача B. Мафия в городе

Построим граф, вершинами которого будут телефонные станции, а ребрами — каналы связи между ними.

Если переформулировать эту задачу на языке теории графов, получится так называемая задача о минимальном вершинном покрытии. *Вершинным покрытием* графа называется такое множество вершин, что у любого ребра графа хотя бы один из концов лежит в этом множестве [2].

Эта задача является NP-полной [2], так что, скорее всего, для нее не существует полиномиального решения. Однако ограничения на размер графа в рассматриваемом случае небольшие, поэтому применим перебор.

Рассмотрим все возможные подмножества множества вершин. Для каждого из них проверим, является ли оно вершинным покрытием. После чего найдем вершинное покрытие минимального размера, а подсчет количества таких покрытий будем вести параллельно.

Для начала решим более простую задачу. Пусть задано некоторое подмножество множества вершин данного графа. Необходимо проверить, является ли оно вершинным покрытием.

Для этого проверим, выполняется ли для заданного подмножества определение вершинного покрытия, то есть для каждого ребра графа проверим, лежит ли хотя бы один его конец в множестве.

Для представления множеств удобно применить битовые маски: пусть i-ый бит числа равен единице, если вершина номер i присутствует в множестве, и нулю в противном случае (биты нумеруются с единицы, начиная с младших). Например, число $1001_2 = 9_{10}$ соответствует множеству из первой и четвертой вершины, а число $110_2 = 6_{10}$ — множеству из второй и третьей вершины. Число 0 соответствует пустому множеству, а число $2^n - 1$ множеству, содержащему все n вершин.

Отметим также, что побитовая операция `and` ($\wedge$) соответствует пересечению множеств, а операция `or` ($\vee$) — объединению.

Множеству, содержащему только вершину номер i соответствует число 2^{i-1}. Для проверки наличия в множестве s вершины номер i можно проверить непустоту его пересечения с множеством, содержащим только вершину i. Это можно сделать, например, проверив условие $s \wedge 2^{i-1} \neq 0$.

Для удобства реализации будем использовать операцию циклического сдвига влево (`shl`, эквивалентна умножению на два) для получения степеней двоек. Понятно, что $2^i = 1 \cdot \underbrace{2 \cdot 2 \ldots \cdot 2}_{i \text{ раз}}$, поэтому 2^i равно `1 shl i`.

Этот подход может быть реализован, например, так.

```
function bitCount(x : longint) : longint;
begin
  result := 0;
```

```
  while (x > 0) do begin
    x := x and (x - 1);
    inc(result);
  end;
end;

begin
  readln(n, m);
  for i := 1 to m do begin
    readln(u[i], v[i]);
  end;
  ans1 := n + 1;
  ans2 := 0;
  for mask := 0 to (1 shl n) - 1 do begin
    curSize := bitCount(mask);
    curGood := true;
    for i := 1 to m do
    begin
      edgeGood := ((mask and (1 shl (u[i] - 1))) <> 0);
      edgeGood := edgeGood or ((mask and (1 shl (v[i] - 1))) <> 0);
      curGood := curGood and edgeGood;
      if (not curGood) then      begin
        break;
      end;
    end;
    if (curGood) then begin
      if (curSize < ans1) then begin
        ans1 := curSize;
        ans2 := 0;
      end;
      if (curSize = ans1) then begin
        inc(ans2);
      end;
    end;
  end;
end.
```

Задача C. Трубы

Задача решается моделированием описанного процесса. Пусть $Water(i)$ — суммарный напор воды в течение i-ой секунды. Значения $Water(i)$ легко вычислить в процессе чтения.

После этого остается промоделировать процесс, учитывая то, что вода не может выливаться, если бассейн пуст.

Приведем реализацию этого подхода:

```
  water[0] := 0;
  for i := 1 to n do begin
    readln(l, r, vv);
    for j := l + 1 to r do begin
      water[j] := water[j] + vv;
    end;
  end;
```

```
  readln(t);
  answer := 0;
  for i := 1 to t do begin
    answer := answer + water[i - 1];
    if (answer < 0) then begin
      answer := 0;
    end;
  end;

  writeln(answer);
```

Задача D. Несложная сортировка

Решение задачи состоит из двух шагов. Первый из них — вычисление последовательности b_i. Для этого удобно создать функцию, вычисляющую значение $s(x,k)$. На языке *Delphi* это можно сделать, например, так.

```
function s(x, k : longint) : longint;
begin
  result := 0;
  while (x > 0) do begin
    result := result + (x mod k);
    x := x div k;
  end;
end;
```

После этого необходимо отсортировать полученную последовательность. Для этого можно применить любую сортировку [2, 3], как работающую за $O(n\,logn)$, так и за $O(n^2)$.

Задача E. Строки

Заметим, что для всех $i = 1 \ldots n$ суммы $\sum_{j=1}^{|S_B|} d(S_A \hookrightarrow i, S_B \hookrightarrow j)$ равны. Обозначим их общее значение как S.

Таким образом, искомая сумма $\sum_{i=1}^{|S_A|} \sum_{j=1}^{|S_B|} d(S_A \hookrightarrow i, S_B \hookrightarrow j)$ равна $n \cdot S$.

Остается найти S. Для этого найдем значение следующей суммы: $\sum_{j=1}^{|S_B|} d(S_A \hookrightarrow n, S_B \hookrightarrow j) = \sum_{j=1}^{|S_B|} d(S_A, S_B \hookrightarrow j)$.

Перепишем ее следующим образом:

$$\sum_{j=1}^{|S_B|} d(S_A, S_B \hookrightarrow j) = \sum_{j=1}^{|S_B|} \sum_{k=1}^{|S_B|} d(S_A^{(k)}, (S_B \hookrightarrow j)^{(k)}) = \sum_{j=1}^{|S_B|} \sum_{k=1}^{|S_B|} d(S_A^{(k)}, S_B^{((k+j-1) mod |S_B|+1)})$$

Здесь $S^{(k)}$ обозначает k-ый символ строки S.

Поменяем теперь порядок суммирования:

$$\sum_{j=1}^{|S_B|} \sum_{k=1}^{|S_B|} d(S_A^{(k)}, S_B^{((k+j-1) mod |S_B|+1)}) = \sum_{k=1}^{|S_B|} \sum_{j=1}^{|S_B|} d(S_A^{(k)}, S_B^{((k+j-1) mod |S_B|+1)})$$

Так как при любом k числа вида $((k+j-1) mod |S_B|+1)$ «пробегают» множество $\{1, \ldots, |S_B|\}$, то сумму можно переписать следующим образом:

$$\sum_{k=1}^{|S_B|} \sum_{j=1}^{|S_B|} d(S_A^{(k)}, S_B^{((k+j-1) mod |S_B|+1)}) = \sum_{k=1}^{|S_B|} \sum_{j=1}^{|S_B|} d(S_A^{(k)}, S_B^{(j)})$$

Обозначим как $Count(S, c)$ количество вхождений символа c в строку S. Нетрудно видеть, что последняя сумма преобразуется следующим образом:

$$\sum_{k=1}^{|S_B|}\sum_{j=1}^{|S_B|} d(S_A^{(k)}, S_B^{(j)}) = \sum_{i='a'}^{'z'}\sum_{j='a'}^{'z'} d(i,j)\cdot Count(S_A, i)\cdot Count(S_B, j)$$

Последнюю сумму легко вычислить, найдя значения $Count(S_A, c)$ и $Count(S_B, c)$ для всех c. Это несложно сделать за время, пропорциональное $|S_A| + |S_B|$.

Задача F. Сверхстепень

Решение задачи основано на тождестве: $(2^{2^n})^2 = 2^{2^n}\cdot 2^{2^n} = 2^{2^n+2^n} = 2^{2^{n+1}}$.

Таким образом, для вычисления n-ой сверхстепени необходимо начать с числа 2 и выполнить n повторных возведений в квадрат.

Так как требуется вычислить остаток от деления результата на число m, то остановимся подробнее на вопросе вычисления по модулю. Пусть заданы три числа a, b и m. Докажем, что следующие формулы верны:

$$(a+b) \bmod m = ((a \bmod m) + (b \bmod m)) \bmod b$$

$$(a-b) \bmod m = ((a \bmod m) - (b \bmod m) + m) \bmod b$$

$$(a\cdot b) \bmod m = ((a \bmod m)\cdot(b \bmod m)) \bmod b$$

Дополнительное слагаемое $+m$ в формуле для разности необходимо из-за того, что операция взятия остатка в большинстве языков программирования некорректно работает с отрицательными числами. Заметим, что $(a \bmod m) - (b \bmod m) + m \geq 0$, так как $0 \leq (a \bmod m) \leq m-1$, $0 \leq (b \bmod m) \leq m-1$, поэтому $(a \bmod m) - (b \bmod m) + m \geq 0$.

Для доказательства всех трех формул представим числа a и b в следующем виде: $a = q_a\cdot m + r_a$, $b = q_b\cdot m + r_b$, где q_a, q_b — частные от деления a и b на m, а r_a и r_b — остатки.

Тогда

$$(a+b) \bmod m = (q_a\cdot m + r_a + q_b\cdot m + r_b) \bmod m =$$
$$= (r_a + r_b + (q_a+q_b)\cdot m) \bmod m =$$
$$= (r_a + r_b) \bmod m = ((a \bmod m) + (b \bmod m)) \bmod b$$

$$(a-b) \bmod m = (q_a\cdot m + r_a - q_b\cdot m - r_b) \bmod m =$$
$$= (r_a - r_b + (q_a-q_b)\cdot m) \bmod m =$$
$$= (r_a - r_b + m) \bmod m = ((a \bmod m) - (b \bmod m) + m) \bmod b$$

$$(a\cdot b) \bmod m = ((q_a\cdot m + r_a)\cdot(q_b\cdot m + r_b)) \bmod m =$$
$$= (q_a\cdot q_b\cdot m^2 + q_a\cdot m\cdot r_b + q_b\cdot m\cdot r_a + r_a\cdot r_b) \bmod m =$$
$$= ((q_a\cdot q_b\cdot m + q_a\cdot r_b + q_b\cdot r_a)\cdot m + r_a\cdot r_b) \bmod m =$$
$$= (r_a\cdot r_b) \bmod m = ((a \bmod m)\cdot(b \bmod m)) \bmod b$$

.

Описанный подход реализует следующий фрагмент программы:

```
res := 2 mod m;
for i := 1 to n do begin
  res := (res * res) mod m;
end;
```

Задача G. От перестановки чисел что-то меняется . . .

Решение этой задачи состоит в том, чтобы проверить все возможные случаи, которых всего 3: a_1 может быть равно $(a_2 + a_3)$, a_2 может быть равно $(a_1 + a_3)$, a_3 может быть равно $(a_1 + a_2)$.

Приведем фрагмент программы, реализующий этот подход:

```
if (a1 = a2 + a3) or (a2 = a1 + a3) or (a3 = a1 + a2) then begin
  writeln('YES')
end else begin
  writeln('NO')
end;
```

Задача H. Треугольник

Легко понять, что задача сводится с следующему: найти такие шесть целых чисел x_1, x_2, x_3, y_1, y_2, y_3, не превосходящих по модулю 10^9, что $x_1 + x_2 + x_3 = 3x$ и $y_1 + y_2 + y_3 = 3y$. При этом требуется, чтобы треугольник $(x_1, y_1) - (x_2, y_2) - (x_3, y_3)$ имел ненулевую площадь.

Будем вести рассуждения относительно чисел x_1, x_2, x_3. Для чисел y_1, y_2, y_3 рассуждения аналогичны.

Ясно, что хотя бы одно число из x_1, x_2, x_3 должно быть больше x, и хотя бы одно — меньше. Иначе, $x_1 = x_2 = x_3 = x$, и площадь соответствующего треугольника равна нулю. Отсюда ясно, что если $|x| = 10^9$, то решения не существует.

Пусть, например, $x_1 = x - 1, x_2 = x + 1, x_3 = x$.

Применяя аналогичные рассуждения к y_1, y_2, y_3, получаем шесть различных вариантов чисел y_1, y_2, y_3 (это перестановки чисел $y - 1, y + 1, y$).

Осталось из этих вариантов выбрать такой, чтобы получившийся треугольник имел ненулевую площадь. Этому условию, например, удовлетворяет такой вариант: $y_1 = y$, $y_2 = y + 1$, $y_3 = y - 1$.

Легко видеть, что полученное решение удовлетворяет всем ограничениям, если $|x|, |y| < 10^9$.

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.