Станкевич Андрей Сергеевич

РАЗБОР ЗАДАЧ ПЯТОЙ ВСЕРОССИЙСКОЙ КОМАНДНОЙ ОЛИМПИАДЫ ШКОЛЬНИКОВ ПО ПРОГРАММИРОВАНИЮ

ВВЕДЕНИЕ

С 26 по 28 ноября 2004 года в Санкт-Петербурге и Барнауле прошла Пятая Всероссийская командная олимпиада школьников по программированию. В олимпиаде приняло участие 110 команд. Используя открытый статус олимпиады, на нее прибыли школьники из Беларуси, одновременно соревнования прошли в Ереване и Ташкенте.

Победителем олимпиады стала команда СУНЦ МГУ, второе место заняла команда ФМЛ № 30 города Санкт-Петербурга, подготовка которой шла также в олимпиадном центре СПбГУ, третье место досталось команде ФТЛ города Саратова.

Вашему вниманию предлагается разбор задач олимпиады. Разбор подготовлен председателем жюри олимпиады Андреем Станкевичем. В разборе использованы фрагменты программ-решений жюри, написанных Андреем Станкевичем, Георгием Корнеевым, Павлом Мавриным и Сергеем Оршанским. Для каждой задачи указан автор, предложивший ее на олимпиаду. Текст условий задач подготовлен Андреем Станкевичем, при этом оригинальная легенда, предложенная автором, была заменена в задачах A, F, H и I. Идея легенды в задаче H предложена Сергеем Оршанским.

ЗАДАЧА A. В ШКОЛУ НА ВЕЛОСИПЕДЕ

Автор: *Антон Попович, СПбГЭТУ «ЛЭТИ».*

Петя любит ездить в школу на велосипеде. Но ездить на велосипеде по тротуарам запрещено, а ездить по дороге опасно. Поэтому Петя ездит только по специальным велосипедным дорожкам. К счастью, и Петин дом, и Петина школа находятся в непосредственной близости от таких дорожек.

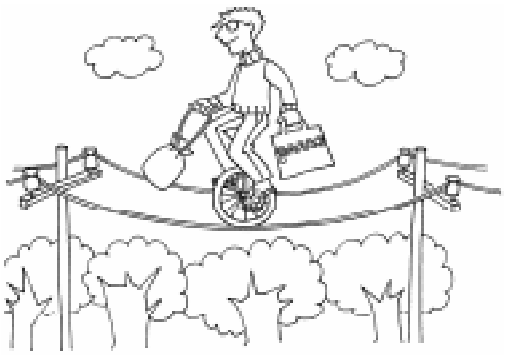
В городе, где живет Петя, есть ровно две велосипедных дорожки. Каждая дорожка имеет форму окружности. В точках их пересечения можно переехать с одной дорожки на другую.

Петя знает точку, в которой он заезжает на дорожку, и точку, в которой следует съехать, чтобы попасть в школу. Петю заинтересовал вопрос: какое минимальное расстояние ему следует проехать по дорожкам, чтобы попасть из дома в школу.

Формат входного файла

Будем считать, что в городе введена прямоугольная декартова система координат. Первые две строки входного файла описывают велосипедные дорожки. Каждая из них содержит по три целых числа – координаты центра окружности, которую представляет собой соответствующая дорожка, и ее радиус. Координаты и радиус не превышают 200 по абсолютной величине, радиус – положительное число. Гарантируется, что дорожки не совпадают.

Следующие две строки содержат по два вещественных числа – координаты точ-



ки, где Петя заезжает на дорожку, и точки, в которой Петя съезжает с дорожки. Гарантируется, что каждая из точек с высокой точностью лежит на одной из дорожек (расстояние от точки до центра одной из окружностей отличается от ее радиуса не более чем на 10^{-8}). Точки могут лежать как на одной дорожке, так и на разных.

Формат выходного файла

Выведите в выходной файл минимальное расстояние, которое следует проехать Пете по велосипедным дорожкам, чтобы попасть из дома в школу. Ответ должен отличаться от правильного не более чем на 10^{-4}.

Если доехать из дома до школы по велосипедным дорожкам невозможно, выведите в выходной файл число –1.

Примеры (рисунок 1)

Входной файл	Выходной файл
0 0 5 4 0 3 3.0 4.0 1.878679656440357 -2.121320343559643	8.4875540166
0 0 5 4 0 3 4.0 3.0 4.0 -3.0	6.4350110879
0 0 4 10 0 4 4.0 0.0 6.0 0.0	-1

Решение

Первое решение, которое приходит в голову для этой задачи, – попытаться разобрать способы, как Петя может добраться от дома до школы, и выбрать среди возможных путей кратчайший. Рассмотрим эти способы.

Если точки, где Петя заезжает на дорожки и где он съезжает с них, находятся на разных дорожках, то Петя должен переехать с одной дорожки на другую. Поскольку две окружности могут пересекаться в двух различных точках, у Пети есть выбор, где осуществить переезд. Также, после выбора точки переезда с одной дорожки на другую у Пети есть два варианта добраться до нее от начальной точки (по часовой стрелке и

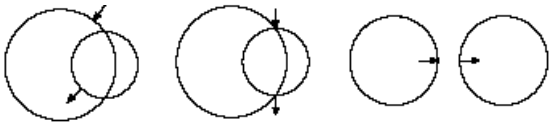


Рисунок 1.

против) и два аналогичных варианта добраться от нее до конечной точки. Таким образом в случае, если конечные точки лежат на разных окружностях, у Пети есть восемь вариантов добраться от одной точки до другой.

Кажется, что, в случае, если точки лежат на одной окружности, у Пети всего два варианта проехать от старта до финиша. Но это не так: оказывается, если окружности имеют разный радиус, Пете может быть выгодно иногда заехать по дороге на другую окружность. Такой пример показан на рисунке 2.

Получается, что у Пети в этом случае семь вариантов добраться от одной точки до другой: заезжать или не заезжать на другую дорожку, а также исходное направление движения от начальной точки и направление движения по вспомогательной дорожке, если он на нее заезжает. Кроме того, следует еще следить за тем, в какой точке пересечения следует заехать на вспомогательную окружность и в какой вернуться обратно.

Ясно, что разбор столь огромного количества случаев представляет собой значительную техническую сложность. Кроме того, трудно не допустить ошибки ни в одном из случаев, поскольку длина пути в каждом из них записывается достаточно запутанной формулой. Возникает идея вместо разбора случаев применить какой-либо более общий метод.

Когда речь заходит о поиске кратчайшего пути, сразу возникают ассоциации с теорией графов. Построим вспомогательный

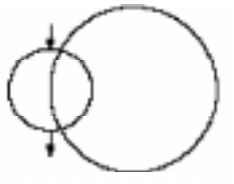


Рисунок 2.

взвешенный граф, который отражает структуру велосипедных дорожек, и сведем нашу задачу к задаче о поиске кратчайшего пути в этом графе.

Вершинами графа будут начальная и конечная точки, а также точки пересечения дорожек. Соединим две вершины ребром, если соответствующие им точки лежат на одной окружности. В качестве длины ребра установим минимум из расстояний между соответствующими точками при проходе от одной к другой по и против часовой стрелки соответственно.

Здесь есть один тонкий момент: расстояние между вершинами, которые соответствуют точкам пересечения, является на самом деле минимумом не из двух, а из четырех чисел – от одной точки пересечения до другой можно добраться двумя способами по каждой из окружностей. Впрочем, разбор этого случая происходит автоматически при аккуратном написании. Для каждой из окружностей проверяем все пары точек, не лежат ли обе точки из пары на окружности. Если это так, проводим релаксацию – для каждого из направлений перехода между точками по рассматриваемой окружности проверяем: если расстояние между точками пока не определено или больше посчитанной величины, то обновляем его.

Чтобы найти кратчайшее расстояние между вершинами, соответствующими начальной и конечной точкам, можно применить любой способ поиска кратчайшего пути в графе, например алгоритм Флойда или Дейкстры. Алгоритм Дейкстры подробно рассмотрен в разборе задачи С.

Остались лишь технические моменты: как пересечь две окружности, как проверить, что две точки лежат на одной окружности, и как найти расстояние между точками вдоль окружности.

Существует несколько методов пересечения окружностей. Изложим полностью алгебраический метод. Как известно, окружность радиуса r с центром в точке $(x_c; y_c)$ представляет собой множество точек, удовлетворяющих уравнению

$$(x - x_c)^2 + (y - y_c)^2 = r^2.$$

Пусть требуется пересечь окружности с центрами в точках $(x_1; y_1)$ и $(x_2; y_2)$ и радиусами r_1 и r_2, соответственно. Запишем систему уравнений, которой должны удовлетворять точки пересечения:

$$\begin{cases}(x-x_1)^2+(y-y_1)^2=r_1^2\\(x-x_2)^2+(y-y_2)^2=r_2^2\end{cases}.$$

Вычитая из первого уравнения второе, получим линейное уравнение:

$$2(x_1 - x_2)x + 2(y_1 - y_2)y = (x_1^2 - x_2^2) + (y_1^2 - y_2^2) - (r_1^2 - r_2^2).$$

Это уравнение прямой. Если окружности пересекаются, то это уравнение прямой, проходящей через точки пересечения окружностей – они удовлетворяют уравнению, а через две точки можно провести ровно одну прямую. Несложно также показать, что если окружности касаются, то эта прямая касается обеих окружностей, а если они не имеют общих точек, то эта прямая не имеет общих точек ни с одной из окружностей (эта прямая называется радикальной осью). Поэтому достаточно пересечь эту прямую с любой из окружностей.

Для этого удобно записать ее уравнение в параметрической форме:

$$(x; y) = (x_0; y_0) + (\Delta x; \Delta y)t.$$

Чтобы перейти от уравнения прямой вида $ax + by = c$, в котором мы его получили, к параметрическому, требуется найти произвольную точку $(x_0; y_0)$, лежащую на прямой, и ее направляющий вектор $(\Delta x; \Delta y)$.

Чтобы найти точку на прямой, можно пересечь прямую с произвольной прямой, которая ей не параллельна. Чтобы уменьшить потери точности, удобно воспользоваться перпендикулярной прямой.

Уравнение прямой, перпендикулярной прямой $ax + by = c$, имеет вид

$$bx - ay = \tilde{c},$$

где $\tilde{c}$ может быть выбрано произвольно. Положим, к примеру, $\tilde{c} = 0$ (это соответствует проведению прямой через начало координат). Тогда для пересечения прямых необходимо решить систему уравнений

$$\begin{cases}ax+by=c\\bx-ay=0\end{cases}$$

Ее решением является точка (*ac/d*; *bc/d*), где $d = a^2 + b^2$. Заметим, что по построению эта точка является ближайшей точкой к началу координат на рассматриваемой прямой, следовательно, использование этой точки позволяет уменьшить потенциальные потери точности.

Осталось найти направляющий вектор прямой. Для этого нам надо найти такую пару (Δx; Δy), чтобы изменение вектора (x; y) на вектор, кратный (Δx; Δy), сохраняло величину $ax + by$. Иначе говоря, $ax + by = a(x + \Delta x) + b(y + \Delta y)$, то есть $a\Delta x = -b\Delta y$. Ясно, что направляющий вектор определен с точностью до константы, в частности, подходит вектор ($-b$; a). Итак, параметрическое уравнение для прямой $ax + by = c$ имеет вид

$$(x; y) = \left(\frac{ac}{a^2 + b^2}; \frac{bc}{a^2 + b^2} \right) + (-b; a)t .$$

Чтобы найти точку пересечения прямой, заданной в параметрической форме, с окружностью, достаточно подставить в уравнение окружности координаты точки на прямой, зависящие от параметра t, удовлетворяющие получившемуся уравнению. После этого, подставив полученные значения t в выражение для кооринат, получим координаты самих точек. Отметим, что приведенным методом можно пересекать окружность также с лучом или отрезком. Достаточно записать соответствующее уравнение в параметрической форме и рассматривать только допустимые значения параметра.

Альтернативный, полностью геометрический метод пересечения двух окружностей, приведен в статье Е.В. Андреевой и Ю.Е. Егорова [1].

Проверить, что точка лежит на окружности, можно, подставив ее координаты в уравнение окружности. Чтобы найти расстояние между точками на окружности, можно, например, найти полярный угол каждой точки относительно центра окружности. Расстояние вдоль окружности радиуса r между точками с полярными углами α_1 и α_2 в одном из направлений есть $r|\alpha_1 - \alpha_2|$, а в другом $r|2\pi - (\alpha_1 - \alpha_2)|$.

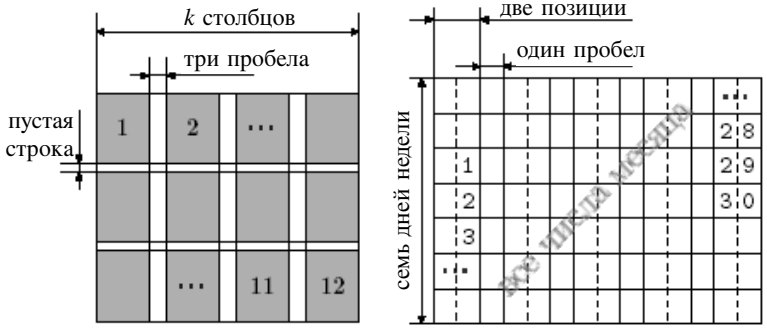
ЗАДАЧА B. КАЛЕНДАРЬ

Автор: *Виктор Матюхин, МГУ.*

Неудовлетворенный стандартным календарем своей операционной системы, Витя решил написать программу, которая будет выводить красиво отформатированный календарь. Он разработал формальные требования к календарю, но понял, что сам не способен написать соответствующую программу. Помогите ему. Календарь состоит из блоков, каждый из которых соответствует одному месяцу. Блоки расположены в виде таблицы из k столбцов и $12/k$ строк (k выбирается делителем числа 12). Месяцы выводятся в следующем порядке: первая строка содержит блоки, соответствующие месяцам с первого по k-ый, следующая – с $(k + 1)$-го по $2k$-ый, и т. д.

Ширина всех блоков в столбце должна быть одинакова, высота всех блоков равна семи (числу дней в неделе). Между блоками различных строк таблицы выводится пустая строка, в каждой строке между соседними блоками из разных столбцов выводится три пробела.

Блок, соответствующий месяцу, устроен следующим образом. Каждой (в том числе неполной) неделе данного месяца в блоке соответствует столбец, имеющий ширину, равную двум. Между двумя соседними столбцами в каждой строке выводится один пробел. Если несколько блоков располагаются в одном столбце в календаре, то для выравнивания ширины в те блоки, которые содержат меньше недель, в конец добавляется необходимое число пустых столбцов-недель. При этом разные столбцы календаря могут иметь разную ширину.



Общая схема календаря Схема блока месяца

Рисунок 3.

Все числа месяца заносятся в блок, соответствующий этому месяцу. Число заносится в ту строку блока, которая соответствует дню недели, на который приходится число в этом месяце. Число заносится в столбец блока, соответствующий неделе, в которой находится данное число. Однозначные числа дополняются слева одним пробелом. Таким образом, числа в столбце оказываются выровнены по правому краю (рисунок 3).

Формат входного файла

Входной файл содержит описание года, календарь для которого следует вывести – три числа: d – день недели, на который приходится первое января ($1 \le d \le 7$), l – является ли год високосным ($l = 1$ означает, что год является високосным, $l = 0$ – что не является) и k – количество столбцов в календаре (k – одно из чисел 1; 2; 3; 4; 6; 12).

Напомним, что високосный год отличается от обычного тем, что в високосном году февраль содержит 29 дней.

Формат выходного файла

Выведите календарь, отформатированный, как описано в условии задачи. Проверяющая программа в этой задаче игнорирует пробелы в конце строк. В остальном календарь должен быть отформатирован в точности в соответствии с условием.

Пример

Для наглядности в примере, вместо пробелов, выведены точки. Ваша программа должна выводить пробелы.

Входной файл

```
4 1 4
```

Выходной файл

```
....5.12.19.26.........2..9.16.23....1..8.15.22.29.......5.12.19.26...
....6.13.20.27.........3.10.17.24....2..9.16.23.30.......6.13.20.27...
....7.14.21.28.........4.11.18.25....3.10.17.24.31.......7.14.21.28...
.1..8.15.22.29.........5.12.19.26....4.11.18.25.......1..8.15.22.29...
.2..9.16.23.30.........6.13.20.27....5.12.19.26.......2..9.16.23.30...
.3.10.17.24.31.........7.14.21.28....6.13.20.27.......3.10.17.24......
.4.11.18.25.........1..8.15.22.29....7.14.21.28.......4.11.18.25......
........................................................................
....3.10.17.24.31.......7.14.21.28.......5.12.19.26.......2..9.16.23.30
....4.11.18.25.......1..8.15.22.29.......6.13.20.27.......3.10.17.24.31
....5.12.19.26.......2..9.16.23.30.......7.14.21.28.......4.11.18.25...
....6.13.20.27.......3.10.17.24.......1..8.15.22.29.......5.12.19.26...
....7.14.21.28.......4.11.18.25.......2..9.16.23.30.......6.13.20.27...
.1..8.15.22.29.......5.12.19.26.......3.10.17.24.31.......7.14.21.28...
.2..9.16.23.30.......6.13.20.27.......4.11.18.25.......1..8.15.22.29...
........................................................................
....6.13.20.27.........4.11.18.25....1..8.15.22.29.......6.13.20.27...
....7.14.21.28.........5.12.19.26....2..9.16.23.30.......7.14.21.28...
.1..8.15.22.29.........6.13.20.27....3.10.17.24.......1..8.15.22.29...
.2..9.16.23.30.........7.14.21.28....4.11.18.25.......2..9.16.23.30...
.3.10.17.24.........1..8.15.22.29....5.12.19.26.......3.10.17.24.31...
.4.11.18.25.........2..9.16.23.30....6.13.20.27.......4.11.18.25......
.5.12.19.26.........3.10.17.24.31....7.14.21.28.......5.12.19.26......
```

Решение и программа

Все, что требовалось в этой задаче, – аккуратно реализовать построение календаря. Приведем ключевые моменты программы.

Особую важность в задачах такого рода представляет выбор подходящих структур данных.

Во-первых, будем использовать константный массив, содержащий количество дней в месяце в не високосном и в високосном году, соответственно.

```
const
days: array [0..1, 1..12] of longint = (
(31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31),
(31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31));
```

Прочитав входные данные, подсчитаем для каждого месяца количество недель в нем (массив `nw`). Переменная `c` содержит текущий день недели.

```
read(d, w, k);

c := d;
for i := 1 to 12 do begin
  nw[i] := 1;
  for j := 1 to days[w][i] do begin
    inc(c);
    if c > 7 then begin
      c := 1;
      if j < days[w][i] then
        inc(nw[i]);
    end;
  end;
end;
```

Посчитаем количество недель в каждом из столбцов (массив `colw`) – это максимум количества недель во всех месяцах данного столбца. Переменная `c` содержит текущий столбец.

```
c := 1;
for i := 1 to 12 do begin
  colw[c] := max(colw[c], nw[i]);
  inc(c);
  if c > k then
    c := 1;
end;
```

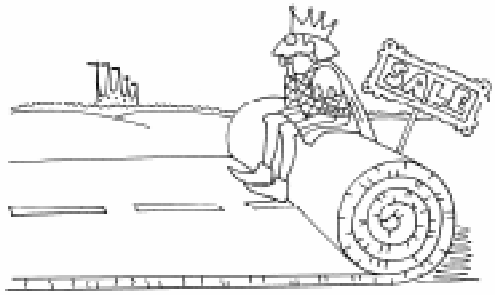
Теперь построим массив, содержащий для каждого месяца, каждой недели и каждого дня этой недели число, которое приходится на этот день. Если этот день отсутствует в месяце (например, понедельник первой недели месяца, если месяц начинается со среды), положим соответствующий элемент массива равным нулю.

```
c := d;
for i := 1 to 12 do begin
  week := 1;
  for j := 1 to days[w][i] do begin
    day[i][week][c] := j;

    inc(c);
    if c > 7 then begin
      c := 1;
      inc(week);
    end;
  end;
end;
```

После этого остается всего лишь аккуратно вывести результат. Для простоты мы запускаем цикл по дням недели до 8 – при значении 8 будет выведена пустая строка, соответствующая разделителю между строками календаря. Последняя конструкция соответствует тому, что после последней строки календаря выводить разделитель не надо.

```
for i := 1 to 12 div k do begin
  for d := 1 to 8 do begin
    for j := 1 to k do begin
      mnth := (i - 1) * k + j;
      for l := 1 to colw[j] do begin
        if day[mnth][l][d] = 0 then
            write(' ')
        else if day[mnth][l][d] < 10 then
            write(' ', day[mnth][l][d])
        else
            write(day[mnth][l][d]);
        if l < colw[j] then
            write(' ');
      end;

      if j < k then
        write(' ');
      end;
      writeln;
      if (d = 7) and (i = 12 div k) then
        break;
  end;
end;
```

ЗАДАЧА C. ВОЕННЫЙ ПОХОД



Автор: *Сергей Оршанский, СПбГУ ИТМО.*

В одном феодальном государстве средневековой Европы было n городов и m дорог, каждая из которых соединяла некоторые два города. Каждая дорога принадлежала правителю одного из городов (не обязательно одного из тех, которые она соединяла). Однажды правитель города S решил объявить войну правителю города T. Перед ним сразу же встала нелегкая задача: как довести армию до города T.

Правитель каждого города требует плату за проход войск через его город. Кроме того, правитель города S может перемещать войска только по дорогам, которые принадлежат ему. При этом он может купить любую дорогу у ее владельца за определенную плату (даже если владелец – правитель города T). К сожалению, все деньги из казны города S были потрачены на предыдущий неудачный военный поход, поэтому сначала правителю придется продать некоторые свои дороги (разумеется, после этого он не сможет провести по ним войска).

Помогите правителю выяснить, какие дороги следует продать, а какие купить, чтобы довести войска от города S до города T и оплатить проход через все промежуточные города. Все операции продажи и покупки дорог надо осуществить до начала похода, пока правители других городов не догадались о воинственных намерениях правителя города S.

Формат входного файла

Первая строка входного файла содержит целые числа n и m – количество городов и дорог соответственно ($2 \le n \le 2\,000$, $1 \le m \le 50\,000$). Города нумеруются от 1 до n, города S и T имеют номера 1 и n соответственно. Следующие n строк содержат под одному целому числу r_i – плату за проезд через город i ($0 \le r_i \le 10\,000$, $r_1 = r_n = 0$).

Следующие m строк содержат описания дорог. Дорога описывается четырьмя целыми числами: a_i, b_i, p_i и c_i. Здесь a_i и b_i – номера городов, которые соединяет дорога, p_i – номер города, правителю которого она принадлежит, c_i – ее стоимость ($a_i \ne b_i$, $1 \le c_i \le 10\,000$). По дороге можно перемещаться в обоих направлениях. Любые два города соединены не более чем одной дорогой.

Формат выходного файла

На первой строке выходного файла выведите список дорог, которые нужно продать в следующем формате: сначала число дорог, а затем их номера. Дороги нумеруются с единицы в том порядке, в котором они заданы во входном файле. На второй строке выведите список дорог, которые нужно купить, в том же формате. На третьей строке выведите маршрут похода – номера городов в порядке следования войска. Если решений несколько, выведите любое. Если решения нет, выведите в выходной файл число –1.

Пример

Входной файл	Выходной файл
3 3 0 1 0 1 2 1 10 2 3 1 10 3 1 2 2	1 1 1 3 1 3

Решение

Эта задача может быть сведена к задаче о поиске кратчайшего пути во взвешенном графе. Опишем метод сведения, а также расскажем об одном из наиболее эффективных алгоритмов решения получившейся задачи – алгоритме Дейкстры.

Заметим сначала, что по условию все операции продажи и покупки дорог осуществляются до военного похода, поэтому нельзя пройти по дороге и затем продать ее. Кроме того, исходно у нас нет денег, а за проход через промежуточные города надо

платить. Поставим себе сначала альтернативную задачу – получить максимальное количество денег. Для этого надо просто продать все свои дороги.

Будем считать, что после этого мы можем покупать их за ту же цену, которую мы получили от ее продажи. Такая операция будет соответствовать тому, что мы просто не продаем эту дорогу вообще.

Итак, мы свели задачу к следующей: есть дороги, каждая из которых имеет некоторую стоимость, следует купить некоторые из них, чтобы можно было добраться от города S до города T. При этом у нас имеется некоторая максимальная сумма денег, которую мы можем потратить (сумма стоимостей проданных дорог). Кроме того, по-прежнему за проход через каждый промежуточный город следует заплатить некоторую сумму.

Попытаемся избавиться от последнего требования. А именно, присоединим сумму, которую следует заплатить при проходе через город, к стоимости дороги, по которой мы приходим в этот город.

Здесь, правда, возникает проблема: по дороге можно двигаться в двух направлениях, стоимость прохода через города на ее концах может быть различной. Решим эту проблему следующим образом: превратим каждую дорогу в две односторонних той же стоимости. К стоимости каждой из получившихся дорог прибавим стоимость прохода через город, в который она ведет. Теперь можно считать, что нам просто надо купить некоторое множество дорог, чтобы добраться из города S до города T, потратив не более некоторой суммы. И дороги теперь ориентированы – по ним можно перемещаться только в одном направлении.

Найдем минимальное по стоимости множество дорог, которое мы можем купить, чтобы добраться из города S до города T. Если их суммарная стоимость не превышает суммы, которой мы располагаем, то мы получаем решение задачи, иначе очевидно, что задача не имеет решения.

Получившуюся задачу можно рассматривать в терминах теории графов следующим образом. Задан ориентированный взвешенный граф. Требуется выбрать минимальное по суммарному весу множество ребер, чтобы оно содержало путь от вершины S до вершины T. Ясно, что в качестве такого множества следует выбирать ребра S–T пути минимального веса. Поскольку, по построению, веса всех ребер нашего графа неотрицательны, такой путь существует, если существует вообще способ добраться из S до T.

Рассмотрим алгоритм построения пути минимального веса в графе, который обычно называют алгоритмом Дейкстры. Данный алгоритм применим в случае, когда веса ребер графа неотрицательны. Мы покажем, как можно естественным образом получить алгоритм Дейкстры из алгоритма, основанного на динамическом программировании (алгоритма Форда-Беллмана). Читатели, знакомые с алгоритмом Дейкстры, могут в принципе пропустить оставшуюся часть разбора.

Итак, рассмотрим взвешенный граф с неотрицательными весами ребер. Будем искать кратчайшие пути от вершины s до всех остальных вершин. Используем динамическое программирование – обозначим как $d_{i,j}$ длину кратчайшего пути от вершины s до вершины i, содержащего не более j ребер.

Ясно, что $d_{s,0} = 0$ и $d_{i,0} = \infty$ для $i \neq s$. Несложно выписать рекуррентное соотношение:

$$d_{i,j} = \min\left(d_{i,j-1}, \min_{(k,i)\in E} \left(d_{k,j-1} + w(k;i)\right)\right).$$

Здесь E означает множество ребер графа, а $w(k; i)$ – вес ребра из вершины k в вершину i. Длина кратчайшего s–i пути равна $d_{i,n}$.

Посмотрим на возможную реализацию этого алгоритма. Заметим, что вместо двумерного массива d, можно использовать одномерный, выполняя n раз для всех i присваивание

$$d_i \leftarrow \min\left(d_i, \min_{(k,i)\in E} \left(d_{k,j-1} + w(k,i)\right)\right).$$

В этом случае после j итераций d_i не будет превышать длины кратчайшего пути от вершины s до вершины i, содержащего не более j ребер (это несложно доказать по индукции). Следовательно, в конце d_i будет содержать длину кратчайшего пути от s до i.

Время работы получившегося алгоритма есть $O(nm)$, где n – число вершин в графе, а m – число ребер. Действительно, на каждой итерации выполняется просмотр каждого ребра ровно по одному разу: когда происходит обновление значения d для вершины, в которую это ребро ведет. Рассматриваемый алгоритм обычно называют алгоритмом Форда-Беллмана.

Укажем теперь прием, который позволяет вывести из алгоритма Форда-Беллмана алгоритм Дейкстры, сократив время работы с $O(nm)$ до $O(n^2)$. Сначала заметим, что обновление массива d можно проводить, перебирая не концы ребер, а их начала.

А именно, выполним n итераций: каждый раз просмотрим все вершины, и для каждой вершины k и каждого ребра $(k; i)$ выполним присваивание:

$$d_i \leftarrow \min (d_i; d_k + w(k; i)).$$

Эта операция называется *релаксацией* по ребру $(k; i)$.

Сделаем следующее ключевое наблюдение: если перед j-ой итерацией значение d_k является окончательным, то есть уже содержит длину кратчайшего s–k пути, то на следующих итерациях проводить релаксации по ребрам, начинающимся в вершине k, не имеет смысла – обновления значений массива d для их концов не произойдет.

И наоборот, если значение d_k строго больше длины соответствующего кратчайшего пути, то проводить релаксацию не имеет смысла на j-ой итерации – все равно на последующих итерациях значения будут улучшены. Таким образом, для каждой вершины на самом деле имеет смысл проводить релаксацию по ребрам, из нее исходящим, не более одного раза – после того, как для нее становится известно кратчайшее расстояние. Вопрос лишь в том, как понять, что текущее расстояние до вершины действительно является кратчайшим.

Научимся определять этот момент. Рассмотрим состояние процесса перед очередной итерацией алгоритма. Пусть для некоторых вершин уже известно, что в массиве d находится кратчайшее расстояние до этих вершин от стартовой, и проведена релаксация по выходящим из них ребрам. Назовем эти вершины *черными*. Остальные вершины будем называть *серыми*. Заметим, что в силу отсутствия ребер отрицательного веса, стартовая вершина всегда является черной. Ясно, что если кратчайший путь до некоторой серой вершины использует в качестве промежуточных вершин только черные, то в настоящее время расстояние до этой вершины посчитано правильно. Отметим, что такая вершина обязательно найдется, поскольку начало кратчайшего пути также является кратчайшим путем (возьмем, например, серую вершину, путь до которой содержит минимальное число ребер). Как же найти эту вершину? Возьмем серую вершину, текущее расстояние до которой является минимальным. Ясно, что кратчайший путь до нее от стартовой проходит только по черным вершинам (иначе рассмотрим первую серую вершину на пути, который лучше, текущее расстояние до нее должно быть меньше). Следовательно, его длина в настоящий момент точно известна и находится в массиве d. Именно для этой вершины и следует провести релаксацию по исходящим из нее ребрам, одновременно сделав ее черной.

Итак, получаем следующий алгоритм. Стандартная инициализация: $d_s = 0$, $d_i = \infty$ для $i \neq s$, все вершины не помечены как черные. Затем проводим n итераций.

На каждой итерации выбрать непомеченную вершину, для которой значение d_i минимально. Пометить ее как черную и провести релаксацию по ребрам, исходящим из вершины i.

Оценим время работы алгоритма. Если для выбора минимума на каждом шаге использовать проход по всему массиву, то на это требуется $O(n)$ действий на каждой итерации, что дает суммарное время $O(n^2)$ на поиск минимума на всех итерациях. При этом релаксация по каждому ребру выполняется ровно один раз, и время релаксации есть $O(1)$, поскольку требуется лишь обновить значение в массиве.

Таким образом, суммарное время на все релаксации составляет $O(m)$, и время работы алгоритма определяется первым компонентом и составляет $O(n^2)$.

Использование альтернативных структур данных позволяет изменить соотноше-

ние времени работы двух составляющих алгоритма. Например, организовав вершины в виде двоичной кучи, используя текущее расстояние в качестве ключа, можно получить время $O(\log n)$ для выбора минимума (общее время $O(n \log n)$ на все выборы минимума), но при этом время на релаксацию также увеличивается до $O(\log n)$. Это ведет к тому, что время работы определяется теперь вторым слагаемым и составляет $O(m \log n)$, что выгодно в разреженных графах. Более подробное описание алгоритма Дейкстры и его реализации можно найти, например, в книге [2].

Вернемся к нашей задаче. Помимо кратчайшего расстояния, которое в нашей задаче позволяет выяснить, существует ли достаточно дешевый путь, требуется указать и сам набор дорог, которые следует купить, то есть сам кратчайший путь.

Алгоритм Дейкстры позволяет найти его вместе с его длиной. Для этого достаточно для каждой вершины хранить вместе с длиной текущего кратчайшего пути указатель на вершину, из которой следует придти в заданную, чтобы получить путь данной длины. Каждый раз, когда при проведении релаксации по ребру текущее значение кратчайшего пути обновляется, следует обновлять и значение предшествующей вершины. Тогда, возвращаясь из вершины назначения по этим указателям, мы и получим кратчайший путь.

Программа

Приведем ключевые моменты программы. В этой задаче количество вершин в графе достаточно мало, поэтому можно использовать для представления графа матрицу весов. Будем использовать значение $2^{30} - 1$ в качестве бесконечности – из ограничений в задаче следует, что это значение заведомо больше всех чисел, которые могут возникать в решении. С другой стороны, такое значение позволяет складывать два бесконечных значения в стандартном 32-битном знаковом типе, не вызывая его переполнения.

Проведем инициализацию. Массив `z` будет содержать матрицу весов.

```
read(n, m);
for i := 1 to n do begin
   read(r[i]);
end;
for i := 1 to n do begin
   for j := 1 to n do begin
     z[i][j] := maxlongint div 2;
   end;
end;
```

Теперь прочитаем ребра и «продадим» все наши дороги. Массив `s` будет содержать значение –1, если следует продать дорогу, 0, если не следует проводить с ней операций, и 1, если ее следует купить. Массив `q` содержит номер дороги, которая соединяет города (эта информация потребуется при выводе ответа).

```
cash := 0;
for i := 1 to m do begin
  read(a[i], b[i], p[i], c[i]);
  if p[i] = 1 then begin
    s[i] := -1;
    cash := cash + c[i];
  end;
  q[a[i], b[i]] := i;
  q[b[i], a[i]] := i;
  z[a[i], b[i]] := c[i] + r[b[i]];
  z[b[i], a[i]] := c[i] + r[a[i]];
end;
```

Теперь выполняем алгоритм Дейкстры. Массив `u` содержит пометки вершин, массив `prev` – ссылки на предшественника для поиска самого пути.

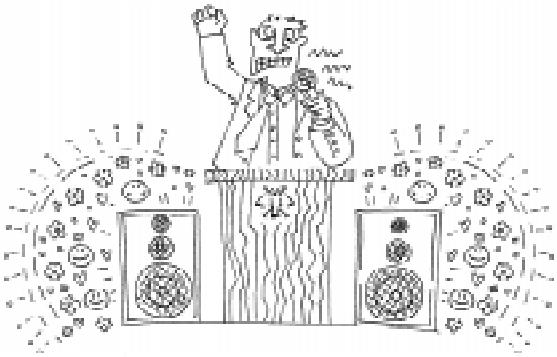
```
for i := 0 to n do begin
  d[i] := maxlongint div 2;
end;
d[0] := maxlongint div 2;
d[1] := 0;
for i := 1 to n do begin
  mi := 0;
  for j := 1 to n do begin
   if not u[j] and (d[j] < d[mi]) then begin
     mi := j;
   end;
  end;
  u[mi] := true;
  for j := 1 to n do begin
   if d[j] > d[mi] + z[mi][j] then begin
     d[j] := d[mi] + z[mi][j];
     prev[j] := mi;
   end;
  end;
end;
```

Осталось найти ответ.

```
if cash < d[n] then begin
  writeln(-1);
  exit;
end;
i := n;
j := prev[n];
while j <> 0 do begin
  inc(s[q[i][j]]);
  i := prev[i];
  j := prev[j];
end;
```

Вывод ответа по массиву s не представляет собой никакой сложности, оставляем его для самостоятельной реализации.

ЗАДАЧА D. ЦЕНЗУРА



Автор: *Андрей Станкевич, СПбГУ ИТМО.*

Телевидение Флатландии готовится показать в вечернем эфире выступление одного известного политика. Поскольку политик известен своей несдержанностью, решено было написать специальную программу, которая вырезала бы из речи политика некоторые фразы, запуская в этот момент рекламу. Поскольку ввести все неполиткорректные слова в программу представляется довольно сложным, решено было использовать эвристический алгоритм определения адекватности фразы.

Речь политика в реальном времени оцифровывается, распознается и подается на вход программе как последовательность фраз. Каждая фраза состоит из слов, записанных в одну строку. Слово представляет собой последовательность символов, ограниченную с обеих сторон границами фразы, пробелами или знаками препинания (символами «.», «!», «?», «:», «-», «,», «;», «(» или «)»).

Слово считается подозрительным, если в него входит не более трех различных букв (любой символ, кроме пробелов и знаков препинания считается буквой, большие и маленькие буквы считаются различными). Например, слова «дом», «мама» или «шалаш» являются подозрительными, а слова «привет», «Шалаш» или «hello» – нет.

Фраза считается подозрительной, если не менее половины слов в ней подозрительны (каждое вхождение слова во фразу считается отдельно).

Напишите программу, которая процензурирует речь политика, удалив из нее все подозрительные фразы.

Формат входного файла

Входной файл содержит не более 1000 фраз, каждая из которых представляет собой строку не длиннее 250 символов. Фраза содержит только символы с ASCII кодами от 32 до 255.

Гарантируется, что в каждой фразе есть хотя бы одно слово.

Формат выходного файла

Выведите в выходной файл все фразы из входного файла, которые не являются подозрительными. Фразы следует выводить в том же порядке, в котором они заданы во входном файле.

Примеры

Входной файл
Наша система власти достаточно стабильна. Флатландия - наша родина. Ура! Пятая Всероссийская олимпиада школьников по программированию. Ну вы это, того... We are the champions! She loves you! Yeah, Yeah, Yeah! Хрен редьки не слаще Спасибо за внимание.
Выходной файл
Наша система власти достаточно стабильна. Пятая Всероссийская олимпиада школьников по программированию. She loves you! Yeah, Yeah, Yeah! Хрен редьки не слаще Спасибо за внимание.

Решение и программа

Эта несложная задача на обработку текста не стала препятствием для большинства команд на олимпиаде. При решении задачи нужно было последовательно решить следующие подзадачи: разбиение фразы на слова, подсчет количества различных символов в слове, проверка того, что слово является подозрительным.

Приведем фрагмент программы, который осуществляет решение задачи. Переменная `g` будет хранить число «хороших» слов, переменная `b` – число «плохих». Сначала указателем `i` находим начало слова, затем указателем `j` – конец. Попутно считаем в переменной `c` количество различных символов в слове. Массив `u`, заиндексированный символами, хранит информацию, встречался ли уже такой символ в текущем слове.

```
while not seekeof do begin
  readln(s);
  inc(i);
  g := 0;
  b := 0;
  s := s + ' ';
  i := 1;
  while true do begin
    while (i <= length(s)) and (s[i]
                in delim) do inc(i);
    if i > length(s) then break;
    j := i;
    c := 0;
    while not (s[j] in delim) do begin
      if not u[s[j]] then begin
        u[s[j]] := true;
        inc(c);
      end;
      inc(j);
    end;
    while i < j do begin
      u[s[i]] := false;
      inc(i);
    end;
    if c > 3 then inc(g) else inc(b);
  end;
  if g > b then writeln(s);
end;
```

ЗАДАЧА E. ПРИКЛЮЧЕНИЯ НА ШАХМАТНОЙ ДОСКЕ

Автор: *Виктор Матюхин, МГУ.*

Ваня очень любит шахматы. Причем он не только любит просто играть в шахматы, но часто придумывает разные головоломки и просто забавные задачки с использованием шахматных фигур. Также, вместо стандартной шахматной доски 8×8, Ваня часто использует в своих задачах доски другого размера.

Недавно он придумал новую головоломку и рассказал ее своим друзьям. Суть головоломки заключается в следующем. На одно из полей доски размером $m \times n$ записывается некоторое положительное целое число и затем на него ставится ферзь.

После этого Ваня делает k ходов ферзем, каждый раз перемещая его по шахматным правилам на одно из полей, на котором он еще не был. При этом каждый раз, перед тем как поставить ферзя на некоторое поле, он записывает на это поле целое число, причем это число всегда больше всех чисел, уже записанных на доске.

Задача друзей Вани – по числам, записанным на доске, восстановить маршрут ферзя или выяснить, что Ваня где-то ошибся. Поскольку Ваня часто выбирает достаточно большие m, n и k, друзья устали решать эту головоломку вручную и решили написать для ее решения программу. Помогите им!

Напомним, что по шахматным правилам ферзь может пойти на любое поле доски, находящееся на одной вертикали, горизонтали или диагонали с тем полем, на котором он находится.

Формат входного файла

Первая строка входного файла содержит числа m, n и k ($1 \le m$; $n \le 300$, $0 \le k < mn$). Следующие m строк содержат по n целых чисел и описывают поля доски, пустому полю соответствует число 0, а полю, на котором записано число, – это число. Все числа, записанные на доске, положительные, целые и не превышают 109.

Формат выходного файла

Если Ваня ошибся при построении головоломки, выведите в выходной файл сообщение «Wrong Board». В противном случае выведите m строк по n чисел – для каждого поля выведите номер хода, перед которым ферзь побывал на этом поле, а для последнего поля, на котором он побывал, – число $k + 1$.

Для полей, на которых ферзь не побывал, выведите число 0.

Примеры

Входной файл	Выходной файл
4 4 7 10 20 0 100 30 0 0 40 0 0 0 0 45 42 0 70	1 2 0 8 3 0 0 4 0 0 0 0 6 5 0 7
2 4 4 10 20 30 40 0 50 0 0	Wrong Board
2 2 2 1 2 4 3	Wrong Board

Решение

Отсортируем числа, находящиеся на доске, вместе с координатами клеток, на которых они находятся, по возрастанию. Если среди чисел встречаются одинаковые, то доска некорректна. Также доска некорректна, если ненулевых чисел на доске не $k + 1$. В противном случае мы получили последовательность клеток в предполагаемом порядке посещения их ферзем. Остается проверить, что ферзь действительно мог перейти между последовательными клетками. Поскольку ферзь может перейти из одной клетки в другую, только если они лежат на одной горизонтали, вертикали или диагонали, то получаем следующие условия на координаты последовательных клеток $(r_1; c_1)$ и $(r_2; c_2)$. Должно выполняться одно из четырех условий: $r_1 = r_2$ (клетки лежат на одной горизонтали), $c_1 = c_2$ (клетки лежат на одной вертикали), $|r_1 - r_2| = |c_1 - c_2|$ (клетки лежат на одной диагонали, параллельной главной) или $|r_1 + r_2| = |c_1 + c_2|$ (клетки лежат на одной диагонали, параллельной побочной).

Замечание: для всех задач ограничение по времени – 2 секунды, ограничение по памяти – 64 мегабайта.

Продолжение разбора (задачи F, G, H, I, J) будет опубликовано в первом номере журнала за 2005 год.

Литература

1. *Андреева Е.В., Егоров Ю.Е.* Вычислительная геометрия на плоскости // Информатика, 2002. С. 39.

2. *Кормен Т., Лейзерсон Ч., Ривест Р.* Алгоритмы: построение и анализ. М.: МЦНМО, 1999.

Станкевич Андрей Сергеевич, преподаватель кафедры Компьютерных технологий СПбГУ ИТМО, лауреат премии Президента Российской Федерации.