

Незнайка и рогейн. 11 класс

Незнайка побывал в Австралии, где услышал про рогейн – вид спорта, изобретённый в этой стране. В рогейне от участника требуется быстрое передвижение бегом по пересечённой местности, навыки ориентирования с помощью карты и компаса, а также точное планирование своего маршрута. На карте соревнований по рогейну обозначены контрольные пункты, за посещение каждого из которых начисляется некоторое фиксированное количество очков. Разные пункты могут приносить разные количества очков, или одинаковые, – это решают организаторы соревнований до начала рогейна. Каждый пункт участник может посетить столько раз, сколько пожелает, но очки начисляются только при первом посещении пункта. Цель каждого участника рогейна, набрать наибольшую сумму очков, затратив на передвижение между контрольными пунктами время, не превышающее максимальный предел, установленный организаторами. Передвигаться между любыми двумя контрольными пунктами можно напрямик через лес, либо по дороге, соединяющей два контрольных пункта, если такая дорога имеется.

Незнайка уже было решил заняться рогейном, но призадумался над тем, как он будет планировать свой маршрут. Поэтому Незнайка составил программу, которая поможет ему. Входные данные программы содержат сведения об N контрольных пунктах, у каждого из которых известно количество очков за его посещение – p_i . Маршрут Незнайки начинается в пункте 1. За время, не превышающее заданное время T , Незнайка должен посетить выбранные им контрольные пункты и успеть вернуться в пункт 1. Выбирать контрольные пункты для маршрута Незнайка должен так, чтобы набрать максимально возможную сумму очков. Про любые два контрольных пункта u и v Незнайка имеет сведения о том, сколько времени он потратит на передвижение от пункта u до пункта v напрямик через лес и о том, есть ли дорога, соединяющая пункты u и v , и если она есть, то сколько времени занимает передвижение по ней. Программа находит набор контрольных пунктов, которые можно успеть посетить и набрать при этом наибольшую сумму очков. При выводе результата номера пунктов сортируются по возрастанию. Набор пунктов всегда начинается с 1. В наборе пункты не повторяются, даже если пункт при передвижении был посещён неоднократно, его номер в ответе выводится один раз. В частности, набор из более чем одного пункта заканчивается не на 1, так как никакой номер пункта не повторяется. Если программа находит несколько наборов пунктов, то она оставляет лишь те, которые имеют наименьший размер. Если после этого остаётся один вариант, то он и является ответом. Если остаётся несколько вариантов, то программа упорядочивает наборы пунктов лексикографически и считает ответом набор пунктов, оказавшийся на первом месте.

Составьте свою программу, дающую те же ответы, что и программа, составленная Незнайкой. Рассмотрим один пример. Пусть имеется три пункта. Пусть передвижение по лесу между любой парой пунктов занимает 9 минут. Пусть между любой парой пунктов есть дорога, движение по которой занимает 1 минуту. Пусть предельное время составляет 10 минут. Пусть количества очков за посещения пунктов равны 1, 2 и 3, соответственно. Программа найдёт два набора пунктов: 1) 1 2 3 и 2) 1 3 2. После сортировки номеров пунктов по возрастанию оба набора запишутся одинаково: 1 2 3. Эта запись и будет ответом.

Рассмотрим ещё один пример. Пусть, как и ранее, имеется три пункта. Пусть передвижение по лесу между любой парой пунктов занимает 9 минут. Пусть между первым и вторым пунктом есть дорога, движение по которой занимает 1 минуту. Пусть между вторым и третьим пунктом есть дорога, движение по которой занимает 1 минуту. Пусть предельное время составляет 4 минуты. Пусть количества очков за посещения пунктов равны 1, 3 и 5, соответственно. Программа найдёт один маршрут подходящей длительности, передвигаясь по которому удастся посетить все три пункта, успеть вернуться в первый и заработать 9 очков: $1 \rightarrow 2 \rightarrow 3 \rightarrow 2 \rightarrow 1$. Единственный набор пунктов этого маршрута будет ответом: 1 2 3.

Рассмотрим последний пример. Пусть, как и ранее, имеется три пункта. Пусть передвижение по лесу между любой парой пунктов занимает 9 минут. Пусть между первым и вторым пунктом есть дорога, движение по которой занимает 1 минуту. Пусть между первым и третьим пунктом есть дорога, движение по которой занимает 1 минуту. Пусть предельное время составляет 2 минуты. Пусть количества очков за посещения пунктов равны 1, 5 и 5, соответственно. В таких условиях есть два маршрута подходящей длительности, передвигаясь по которым удастся посетить два пункта, успеть вернуться в первый и заработать 6 очков: $1 \rightarrow 2 \rightarrow 1$ и $1 \rightarrow 3 \rightarrow 1$. Разных наборов пунктов у этих маршрутов будет два: 1) 1 2 и 2) 1 3. Лексикографически первый набор предшествует второму. Он и будет ответом.

Формат ввода:

В первой строке вводятся 3 натуральных числа N T M ($N \leq 10$, $T \leq 10^9$, $M \leq \frac{N(N-1)}{2}$) – количество контрольных пунктов, максимальный предел времени и количество дорог. Во второй строке даны N натуральных чисел – количества очков за посещение контрольных пунктов – p_1, \dots, p_N ($p_i \leq 10^9$). В следующих N строках дана таблица G размера $N \times N$ из натуральных чисел, где $G[i][j] = G[j][i] \leq 10^9$, и $G[i][j]$ задаёт время, которое потребуется для перемещения между пунктами i и j напрямик через лес. Гарантируется, что $G[i][i] = 0$. Затем в M строках даны сведения о дорогах в формате u_i v_i d_i ($1 \leq u_i, v_i \leq N$, $u_i \neq v_i$, $1 \leq d_i \leq 10^9$). Любая из этих строк сообщает о том, что из пункта u_i можно добежать до пункта v_i по дороге за d_i минут. Любая дорога допускает движение в обоих направлениях (из u_i до v_i и наоборот), и время, затраченное на движение по ней, не зависит от направления движения. Любая пара пунктов может быть соединена не более чем одной дорогой. Если проложена дорога из u_i до v_i , то ещё одной дороги из v_i до u_i быть не может.

Формат вывода:

В первой строке программа выводит K – количество пунктов наборе, составленном так, как описано выше. Во второй строке программа выводит номера пунктов по возрастанию, разделяя одиночными пробелами, не ставя пробел после последнего пункта в наборе: n_1 n_2 ... n_K , где $n_1 = 1$ и $n_K \neq 1$ при $K \neq 1$.

Ввод примера №1:

```
3 10 2
1 4 3
0 9 9
9 0 9
9 9 0
1 2 1
2 3 1
```

Вывод примера №1:

```
3
1 2 3
```

Ввод примера №2:

```
3 10 1
1 4 5
0 9 9
9 0 9
9 9 0
1 2 1
```

Вывод примера №2:

```
2
1 2
```

Ввод примера №3:

```
3 10 1
1 4 5
0 9 9
9 0 9
9 9 0
1 2 6
```

Вывод примера №3:

```
1
1
```

Пояснения по решению

Задача состоит в том, чтобы составить маршрут во взвешенном неориентированном графе длины меньше параметра T , который пройдет через вершины с максимальной суммой очков.

Заметим, что дороги можно использовать только, если она укорачивает передвижение между пунктами, а значит, для каждой дороги (u, v, w) , где u, v — это номера пунктов, а w — это время передвижения между u и v по дороге можно обновить значение матрицы смежности $G[u][v] = \min(G[u][v], w)$.

Далее отметим, что параметр N довольно мал, что позволяет нам перебрать все перестановки $P = (a_1, \dots, a_{n-1})$ посещения вершин, где $a_i \in \overline{2, n}$ и $a_i \neq a_j$ при $i \neq j$. Каждая такая перестановка будет описывать последовательность первых посещений соответствующих пунктов. Тогда ответ на вопрос, сколько времени потребуется, чтобы посетить вершины (a_1, \dots, a_k) находится, как $d[1][a_1] + d[a_1][a_2] + \dots + d[a_k][1]$, где $d[i][j]$ — это минимальное время, необходимое для передвижения от i пункта до j . Подсчет суммы можно вести, пока ее значение меньше T . Итого каждая перестановка может лишь один раз попытаться обновить максимальное количество очков за пункты. Алгоритм достаточно эффективен, если заранее посчитать $d[i][j]$.

Вычислить $d[i][j]$ можно несколькими способами:

1. Для каждого пункта запустить алгоритм Дейкстры.
2. Сразу найти все кратчайшие расстояния с помощью алгоритма Флойда - Уоршелла.

В конце необходимо вывести вершины, встретившиеся в лучшей перестановке в порядке возрастания.

Код возможного решения

```
#include <bits/stdc++.h>

using namespace std;

const int inf = 1'000'000'001;

int main() {
    // read data
    int N, T, M;
    cin >> N >> T >> M;
    vector<int> p(N);
    for (int i = 0; i < N; ++i) cin >> p[i];
    vector<vector<int>> G(N, vector<int>(N));
    for (int i = 0; i < N; ++i)
        for (int j = 0; j < N; ++j) cin >> G[i][j];
    int u, v, d;
    for (int i = 0; i < M; ++i) {
        cin >> u >> v >> d;
        u--;
        v--;
        G[u][v] = min(G[u][v], d);
        G[v][u] = min(G[v][u], d);
    }
    // preprocessing
    for (int i = 0; i < N; ++i)
        for (int j = 0; j < N; ++j) {
            if (i == j) continue;
            if (G[i][j] == 0) G[i][j] = inf;
        }

    for (int k = 0; k < N; ++k)
        for (int i = 0; i < N; ++i)
            for (int j = 0; j < N; ++j)
                G[i][j] = min(G[i][j], G[i][k] + G[k][j]);
```

```

// main algorithm
vector<int> route;
for (int i = 1; i < N; ++i) route.push_back(i);
long long max_score = p[0];
vector<int> max_route;
do {
    long long iter_score = p[0];
    long long iter_time = 0;
    int iter_pos = 0;
    for (int checkpoint : route) {
        if (G[iter_pos][checkpoint] >= inf) continue;
        if (G[checkpoint][0] >= inf) continue;
        if (iter_time + G[iter_pos][checkpoint] + G[checkpoint][0] <= T) {
            iter_time += G[iter_pos][checkpoint];
            iter_score += p[checkpoint];
            iter_pos = checkpoint;
        } else
            break;
    }
    if (max_score < iter_score) {
        max_score = iter_score;
        max_route.clear();
        for (int checkpoint : route) {
            if (checkpoint == iter_pos) {
                max_route.push_back(checkpoint);
                break;
            }
            max_route.push_back(checkpoint);
        }
    }
} while (next_permutation(route.begin(), route.end()));
// print answer
set<int> ans = {1};
for (int checkpoint : max_route) ans.insert(checkpoint + 1);
cout << ans.size() << endl;
for (int checkpoint : ans) cout << checkpoint << ' ';
cout << endl;
}

```