Университет ИТМО

Факультет ПИиКТ

Алгоритмы и структуры данных

Лабораторная работа №3

Выполнила: Наумова Надежда Александровна

Группа P3201

Преподаватель: Тропченко Андрей Александрович

Санкт-Петербург

2020 г.

## **1080. Раскраска карты**

# Постановка проблемы

Рассмотрим географическую карту с *N* странами, занумерованными от 1 до *N* (0 < *N* < 99). Для каждой страны известны номера соседних стран, т.е. имеющих общую границу с данной. Из каждой страны можно попасть в любую другую, перейдя некоторое количество границ. Напишите программу, которая определит, возможно ли покрасить карту только в два цвета — красный и синий — так, что если две страны имеют общую границу, их цвета различаются. Цвет первой страны — красный. Ваша программа должна вывести одну возможную раскраску для остальных стран или сообщить, что такая раскраска невозможна.

### Исходные данные

В первой строке записано число *N*. Из следующих *N* строк *i*-я строка содержит номера стран, с которыми *i*-я страна имеет границу. Каждое целое число в *i*-й строке больше, чем *i*, кроме последнего, которое равно 0 и обозначает конец списка соседей *i*-й страны. Если строка содержит 0, это значит, что *i*-я страна не соединена ни с одной страной с б*o*льшим номером.

### Результат

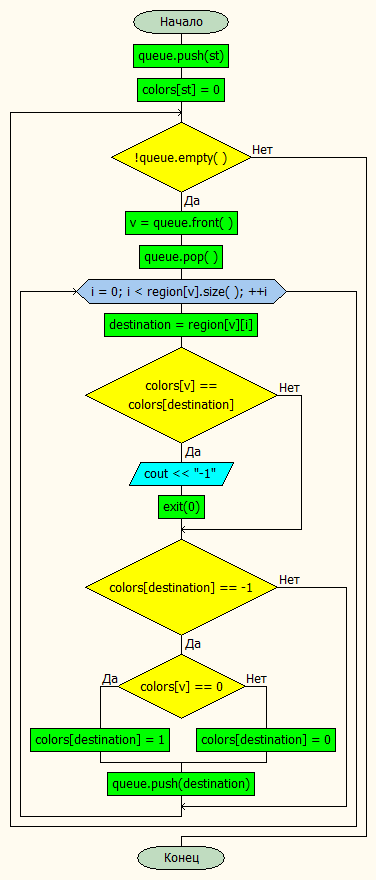
Вывод содержит ровно одну строку. Если раскраска возможна, эта строка должна содержать список нулей и единиц без разделителей между ними. *i*-я цифра в этой последовательности обозначает цвет *i*-й страны. 0 соответствует красному цвету, единица — синему. Если раскраска невозможна, выведите целое число –1.

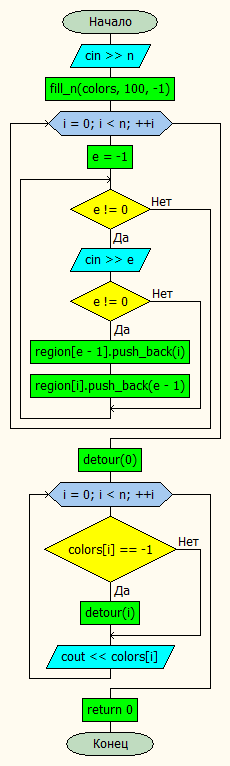
# Описание решения

Обойдем граф в ширину. Если город можно покрасить в синий или красный, не нарушив при этом условие, то красим и переходим к следующему. Если условие нарушается и встречаются хотя бы два соседних города одного цвета, то делаем вывод, что такую карту покрасить в 2 цвета невозможно.

Если по завершении обхода графа остаются непосещенные вершины (граф не связный), то повторяем алгоритм для них (обойдем эти вершины в ширину отдельно).

# Схема решения





# Исходный код

// импорт необходимых библиотек  
#include <bits/stdc++.h>  
  
// использование пространства имен std  
using namespace std;  
  
// массив, где каждый номер пункта будет иметь  
// соотв. цвет  
int colors[100];  
  
//вектор для хранения графа  
vector<int> region[100];  
  
// объявление переменной, хранящей кол-во вершин графа  
int n;  
  
// ф-ия, реализующая алгоритм проверки графа на двудольность  
void detour(int st) {  
 queue<int> queue;  
 queue.push(st);  
 colors[st] = 0;  
  
 while (!queue.empty()) {  
 int v = queue.front();  
 queue.pop();

// проверка всех, с кем соединена текущая вершина  
 for (int i = 0; i < region[v].size(); ++i) {

// одна такая вершина, с которой соединена текущая  
 int destination = region[v][i];

// если она уже покрашена в тот же цвет, то  
 //нельзя покрасить в 2 цвета совсем  
 if (colors[v] == colors[destination]) {  
 cout << "-1";  
 exit(0);  
 }

// если она еще не покрашена, красим в отличный от текущего цвет  
 // и кладем в очередь, чтобы продолжить покраску с нее  
 if (colors[destination] == -1) {  
 if (colors[v] == 0) {  
 colors[destination] = 1;  
 } else {  
 colors[destination] = 0;  
 }  
 queue.push(destination);  
 }  
 }  
 }  
}  
  
int main(){  
 // считываем кол-во городов  
 cin >> n;  
 // заполняем массив ответа -1-ами  
 fill\_n(colors, 100, -1);  
 // "соединяем вершины на графе"  
 for (int i = 0; i < n; i++) {  
 int e = -1;  
 while (e != 0) {  
 cin >> e;  
 if (e != 0) {  
 region[e - 1].push\_back(i);  
 region[i].push\_back(e - 1);  
 }  
 }  
 }  
  
 // начинаем раскраску графа  
 detour(0);  
  
 for(int i = 0; i < n; i++){  
 // раскраска, если граф оказался не связным  
 if(colors[i] == -1) {  
 detour(i);  
 }  
 // вывод результата  
 cout << colors[i];  
 }  
  
 return 0;  
}

# Вывод:

Я научилась имитировать граф в программе, а также смогла реализовать алгоритм проверки графа на двудольность.

## **1450. Российские газопроводы**

# Постановка проблемы

### Вступление

Большими неприятностями обернулся прошедший год для Государства Российского. То неурожай, то птичий грипп, то вечные споры хозяйствующих субъектов... A тут ещё и Президент задумал, наконец, собрать средства на покупку новой балалайки и ручного медведя для своего двоюродного племянника. Все эти факторы (в особенности, конечно, последний) сильно ударили по экономике государства. Посовещавшись со своими друзьями в валенках и ушанках, Президент решил воспользоваться традиционным методом укрепления национального бюджета - увеличением налога на транспортировку газа.

### Задача

Сеть российских газопроводов представляет собой N перекачивающих станций, некоторые из которых соединены газопроводами. Для каждого из M газопроводов известны номера станций A[i] и B[i], которые он соединяет, и его прибыльность C[i], т.е. то количество долларов, которое будет ежесуточно приносить в виде налогов перекачка газа по этому газопроводу. Каждая пара станций соединена не более чем одним газопроводом.

Сеть была построена советскими инженерами, которые точно знали, что газ поставляется из месторождений Украины в Сибирь, а не наоборот. Поэтому все газопроводы являются однонаправленными, т.е. для каждого газопровода перекачка газа возможна только в направлении из станции с номером A[i] на станцию с номером B[i]. Более того, для любых двух станций X и Y верно, что если возможна перекачка газа из X на Y (возможно, через промежуточные станции), то обратная перекачка из Y на X невозможна. Известно, что газ поступает на начальную станцию с номером S и отгружается потребителям на конечной станции с номером F.

Президент потребовал от Правительства указать маршрут (т.е. линейную последовательность попарно соединённых газопроводами станций) перекачки газа из начальной станции на конечную, причём прибыльность этого маршрута должна быть максимальной. Под прибыльностью маршрута понимается суммарная прибыльность входящих в него газопроводов.

К сожалению, Президент не учёл того факта, что многие газопроводы изначальной сети уже давно прекратили существование, в результате чего может оказаться, что перекачка газа из начальной станции на конечную вообще невозможна...

### Исходные данные

Первая строка содержит целые числа N (2 ≤ N ≤ 500) и M (0 ≤ M ≤ 124750). Каждая из следующих M строк содержит целые числа A[i], B[i] (1 ≤ A[i], B[i] ≤ N) и C[i] (1 ≤ C[i] ≤ 10000) для соответствующего газопровода. Последняя строка содержит целые числа S и F (1 ≤ S, F ≤ N; S ≠ F).

### Результат

Если искомый маршрут существует, выведите его прибыльность. Иначе выведите "No solution".

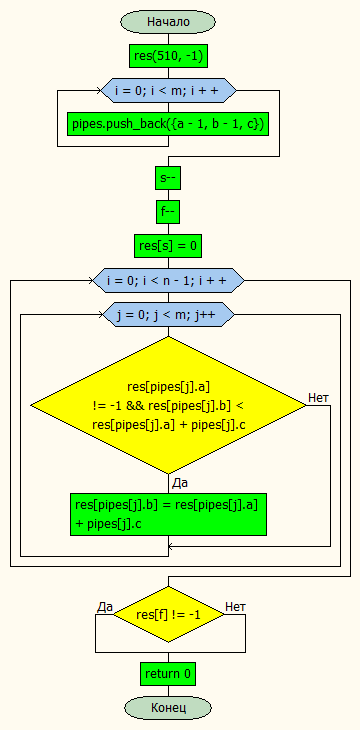
## Описание решения

Для решения данной задачи воспользуемся алгоритмом Беллмана-Форда.

Создадим матрицу res, в которую будем записывать максимальную газопроводность на нынешнем шаге. На каждой итерации будем рассматривать все возможные пути из каждой посещённой вершины. Если мы находим новое максимальное значение газопропускной способности в вершину i, то обновляем значение в res[i].

В итоге получаем, что в векторе res находятся максимальные значения от s. В случае, если какой-то элемент res остался равен -1, можем сделать вывод, что такого пути не существует.

# Схема решения



# Исходный код

// импорт необходимых библиотек

#include <iostream>  
#include <vector>  
  
// создаем структуру "трубы"  
struct pipe {  
 int a, b, c;  
};  
  
// создаем вектор, в котором будем хранить информацию о трубах  
std::vector<pipe> pipes;  
  
int main() {  
 //инициализируем и считываем количество станций и газопроводов  
 int n, m;  
 std::cin >> n >> m;  
  
 //создаем вектор, в который будем записывать результат  
 std::vector<int> res(510, -1);  
  
 for (int i = 0; i < m; i ++) {  
 // считываем и записываем информацию о трубах  
 // а-1 и b-1 сделано для удобства отсчета с нуля  
 int a, b, c;  
 std::cin >> a >> b >> c;  
 pipes.push\_back({a - 1, b - 1, c});  
 }  
 // получаем на вход и сохраняем номер начальной и конечной точки маршрута  
 int s, f;  
 std::cin >> s >> f;  
 // для удобства отсчета с нуля вычтем по единице  
 s--;  
 f--;  
  
 //обнуляем результат  
 res[s] = 0;  
  
 //поиск наивыгоднейшего маршрута (Алгоритм Беллмана-Форда)  
 for (int i = 0; i < n - 1; i ++) {  
 for (int j = 0; j < m; j++) {  
 if (res[pipes[j].a] != -1 && res[pipes[j].b] < res[pipes[j].a] + pipes[j].c) {  
 res[pipes[j].b] = res[pipes[j].a] + pipes[j].c;  
 }  
 }  
 }  
  
 //выводим результат (или "No solution" если такой газопровод нельзя построить)  
 if (res[f] != -1) {  
 std::cout << res[f];  
 } else {  
 std::cout << "No solution";  
 }  
  
 return 0;  
}

# Вывод:

в данной задаче я применила алгоритм Беллмана-Форда поиска кратчайшего пути во взвешенном графе.

## **1160. Network**

# Постановка проблемы

Andrew is working as system administrator and is planning to establish a new network in his company. There will be N hubs in the company, they can be connected to each other using cables. Since each worker of the company must have access to the whole network, each hub must be accessible by cables from any other hub (with possibly some intermediate hubs).

Since cables of different types are available and shorter ones are cheaper, it is necessary to make such a plan of hub connection, that the maximum length of a single cable is minimal. There is another problem - not each hub can be connected to any other one because of compatibility problems and building geometry limitations. Of course, Andrew will provide you all necessary information about possible hub connections.

You are to help Andrew to find the way to connect hubs so that all above conditions are satisfied.

### Исходные данные

The first line contains two integer: N - the number of hubs in the network (2 ≤ N ≤ 1000) and M — the number of possible hub connections (1 ≤ M ≤ 15000). All hubs are numbered from 1 to N. The following M lines contain information about possible connections - the numbers of two hubs, which can be connected and the cable length required to connect them. Length is a positive integer number that does not exceed 106. There will be no more than one way to connect two hubs. A hub cannot be connected to itself. There will always be at least one way to connect all hubs.

### Результат

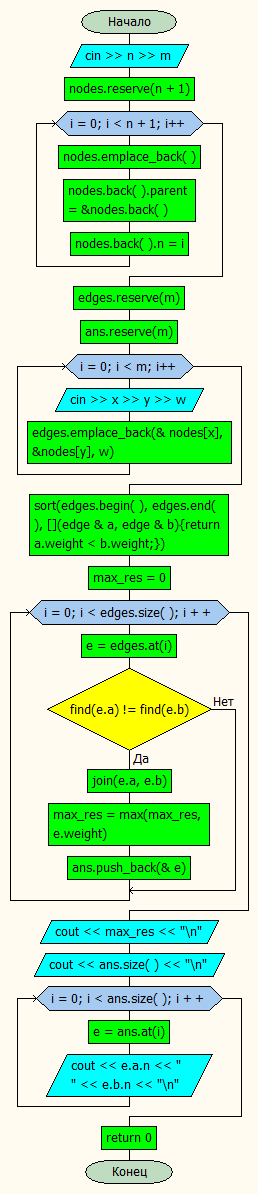
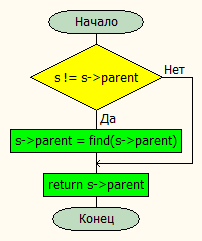
Output first the maximum length of a single cable in your hub connection plan (the value you should minimize). Then output your plan: first output P - the number of cables used, then output P pairs of integer numbers - numbers of hubs connected by the corresponding cable. Separate numbers by spaces and/or line breaks.

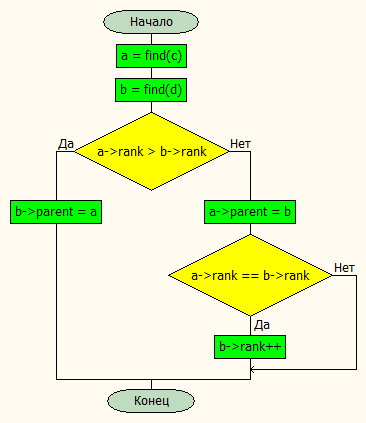
## Описание решения

Задача на нахождение кратчайшего пути (Алгоритм Краскала).

Отсортируем ребра по возрастанию. Добавляем в граф ребра, начиная с минимального. Если при добавлении следующего ребра получается цикл, то это ребро не нужно. В итоге получается минимальное остовное дерево.

# Схема решения





# Исходный код

// импорт необх. библиотек

#include <bits/stdc++.h>  
// исп. простр. имен std

using namespace std;  
  
struct node {  
 node\* parent;  
 int rank; // для объединения по рангу  
 int n{0}; // номер этого узла  
 node() : parent(this), rank(0) { }  
};  
  
// структура для описания провода - хранит  
// "откуда" и "куда" его тянут и длину  
struct edge {  
 node\* a, \*b;  
 int weight;  
 edge(node\* a, node\* b, int weight) : a(a), b(b), weight(weight) {}  
};  
  
// функция поиска ребра с таким весом  
node\* find(node\* s) {  
 if (s != s->parent) {  
 s->parent = find(s->parent);  
 }  
 return s->parent;  
}  
  
// метод для присоединения  
void join(node\* c, node\* d) {  
 node\* a = find(c);  
 node\* b = find(d);  
 if(a->rank > b->rank) {  
 b->parent = a;  
 } else {  
 a->parent = b;  
  
 if(a->rank == b->rank) {  
 b->rank++;  
 }  
 }  
}

// векторы для хранения узлов, проводов, ответа соотв.  
vector<node> nodes;  
vector<edge> edges;  
vector<edge\*> ans;  
  
int main() {  
  
 // объявление и считывание зн-й переменных, хранящих  
 // кол-во хабов и максимальное допустимое кол-во соединений  
 int n, m;  
 cin >> n >> m;  
 nodes.reserve(n + 1);  
  
 // заполнение вектора с узлами  
 for (int i = 0; i < n + 1; i++) {  
 nodes.emplace\_back();  
 nodes.back().parent = &nodes.back();  
 nodes.back().n = i;  
 }  
  
 // устанавливаем минимально возможное кол-во эл-тов в векторах  
 edges.reserve(m);  
 ans.reserve(m);  
  
 // заполнение вектора с проводами, сортировка по длине провода  
 for (int i = 0; i < m; i++) {  
 int x, y, w;  
 cin >> x >> y >> w;  
 edges.emplace\_back(&nodes[x], &nodes[y], w);  
 }  
 sort(edges.begin(), edges.end(), [] (edge& a, edge& b) { return a.weight < b.weight; });  
 int max\_res = 0;  
  
 // применяем алгоритм Краскала  
 for (int i = 0; i < edges.size(); i ++) {  
 edge& e = edges.at(i);  
 // если нет цикла  
 if(find(e.a) != find(e.b)) {  
 join(e.a, e.b);  
 max\_res = max(max\_res, e.weight);  
 ans.push\_back(&e);  
 }  
 }  
 // вывод результатов  
 cout << max\_res << "\n";  
 cout << ans.size() << "\n";  
 for (int i = 0; i < ans.size(); i ++) {  
 auto e = ans.at(i);  
 cout << e->a->n << " " << e->b->n << "\n";  
 }  
 return 0;  
}

# Вывод:

в данной задаче я запрограммировала алгоритм поиска минимального остовного дерева.

## **1162. Currency Exchange**

# Постановка проблемы

Several currency exchange points are working in our city. Let us suppose that each point specializes in two particular currencies and performs exchange operations only with these currencies. There can be several points specializing in the same pair of currencies. Each point has its own exchange rates, exchange rate of A to B is the quantity of B you get for 1A. Also each exchange point has some commission, the sum you have to pay for your exchange operation. Commission is always collected in source currency.

For example, if you want to exchange 100 US Dollars into Russian Rubles at the exchange point, where the exchange rate is 29.75, and the commission is 0.39 you will get (100 - 0.39) \* 29.75 = 2963.3975RUR.

You surely know that there are N different currencies you can deal with in our city. Let us assign unique integer number from 1 to N to each currency. Then each exchange point can be described with 6 numbers: integer A and B - numbers of currencies it exchanges, and real RAB, CAB, RBA and CBA - exchange rates and commissions when exchanging A to B and B to A respectively.

Nick has some money in currency S and wonders if he can somehow, after some exchange operations, increase his capital. Of course, he wants to have his money in currency S in the end. Help him to answer this difficult question. Nick must always have non-negative sum of money while making his operations.

### Исходные данные

The first line contains four numbers: N - the number of currencies, M - the number of exchange points, S - the number of currency Nick has and V - the quantity of currency units he has. The following M lines contain 6 numbers each - the description of the corresponding exchange point - in specified above order. Numbers are separated by one or more spaces. 1 ≤ S ≤ N ≤ 100, 1 ≤ M ≤ 100, V is real number, 0 ≤ V ≤ 103.

For each point exchange rates and commissions are real, given with at most two digits after the decimal point, 10-2 ≤ rate ≤ 102, 0 ≤ commission ≤ 102.

Let us call some sequence of the exchange operations simple if no exchange point is used more than once in this sequence. You may assume that ratio of the numeric values of the sums at the end and at the beginning of any simple sequence of the exchange operations will be less than 104.

### Результат

If Nick can increase his wealth, output YES, in other case output NO.

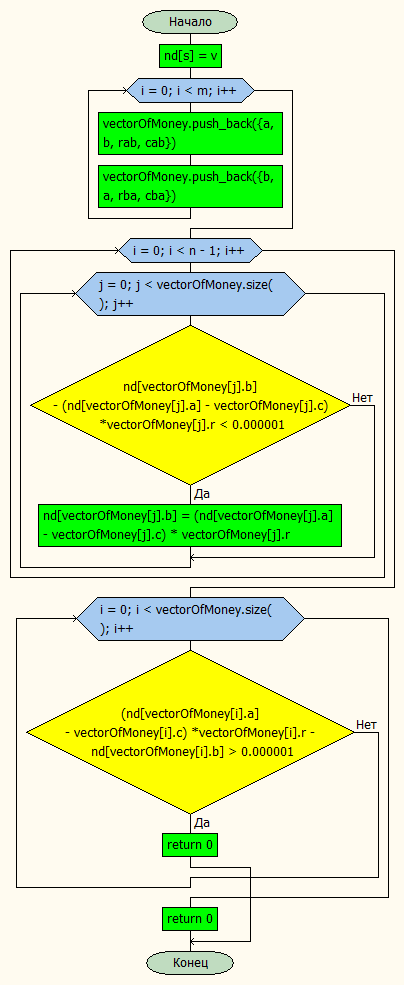
## Описание решения

Задача аналогична задаче 1450, однако при поиске максимума теперь необходимо учитывать, что при каждом переводе взымается комиссия.

Представим все возможные переводы в виде графа и воспользуемся алгоритмом Беллмана-Форда.

На каждой итерации будем рассматривать все возможные пути из каждой посещённой вершины. При нахождении нового максимального значения будем обновлять значение nd. И в итоге мы получим вектор, в котором будут находиться максимальные значения от начальной вершины.

# Схема решения



# Исходный код

#include <iostream>  
#include <vector>  
  
struct money {  
 int a, b;  
 double r, c;  
};  
  
std::vector<money> vectorOfMoney;  
// массив для поиска максимальных значений  
double nd[101];  
  
int main() {  
 // переменные для хранения кол-ва валют, пунктов,  
 // валют у Ника, денег у Ника  
 int n, m, s;  
 double v;  
 std::cin >> n >> m >> s >> v;  
 // инициализация начального значения  
 nd[s] = v;  
 for (int i = 0; i < m; i++) {  
 // переменные с номерами валют, которые обменивает данный обменник  
 int a, b;  
 // переменные с обменными курсами и комиссиями  
 double rab, cab, rba, cba;  
 std::cin >> a >> b >> rab >> cab >> rba >> cba;  
 // заполняем граф  
 vectorOfMoney.push\_back({a, b, rab, cab});  
 vectorOfMoney.push\_back({b, a, rba, cba});  
 }  
  
 // применяем алгоритм Беллмана-Форда  
 for (int i = 0; i < n - 1; i++) {  
 for (int j = 0; j < vectorOfMoney.size(); j++) {  
 if (nd[vectorOfMoney[j].b] - (nd[vectorOfMoney[j].a] - vectorOfMoney[j].c) \* vectorOfMoney[j].r < 0.000001) {  
 nd[vectorOfMoney[j].b] = (nd[vectorOfMoney[j].a] - vectorOfMoney[j].c) \* vectorOfMoney[j].r;  
 }  
 }  
 }  
  
 // вывод результата  
 for (int i = 0; i < vectorOfMoney.size(); ++i) {  
 if ((nd[vectorOfMoney[i].a] - vectorOfMoney[i].c) \* vectorOfMoney[i].r - nd[vectorOfMoney[i].b] > 0.000001) {  
 std::cout << "YES";  
 return 0;  
 }  
 };  
 std::cout << "NO";  
 return 0;  
}

# Вывод:

в данной задаче я применила алгоритм Беллмана-Форда поиска кратчайшего пути во взвешенном графе.

## **1806. Мобильные телеграфы**

# Постановка проблемы

Каждому бойцу 25-й стрелковой дивизии выдали новейшее средство связи — мобильный телеграф. С его помощью можно отправлять телеграммы командованию и боевым товарищам прямо на поле битвы. К сожалению, конструкция телеграфов ещё далека от совершенства — передавать сообщения можно только между некоторыми парами телеграфов.

Каждому устройству присвоен уникальный номер — строка из десяти десятичных цифр. С телеграфа *a* можно отправить сообщение на телеграф *b* только в том случае, если из номера *a* можно получить номер *b*, изменив в нём ровно одну цифру либо поменяв в нём две цифры местами. Время передачи сообщения с телеграфа *a* на телеграф *b* зависит от длины наибольшего общего префикса их номеров — чем больше его длина, тем быстрее передаётся сообщение.

Во время очередного сражения Анка из своей хорошо замаскированной позиции увидела небольшую группу белых, пытающуюся обойти обороняющихся красноармейцев с тыла. Какое минимальное время понадобится на доставку этой информации от Анки до Чапаева по телеграфу, возможно, с помощью других красноармейцев?

### Исходные данные

В первой строке записано целое число *n* (2 ≤ *n* ≤ 50000) — количество бойцов в дивизии. Во второй строке через пробел в порядке невозрастания записаны десять целых чисел в пределах от 1 до 10000 — время передачи сообщения с одного телеграфа на другой при длине общего префикса их номеров, равной нулю, единице, двум, …, девяти. Далее идут *n* строк, содержащие номера телеграфов, выданных бойцам дивизии. Номер телеграфа Анки указан первым, а номер телеграфа Чапаева — последним. Все номера телеграфов попарно различны.

### Результат

Если передать Чапаеву сообщение нельзя, выведите в единственной строке «-1». В противном случае в первой строке выведите минимальное время, за которое можно доставить сообщение. Во второй строке выведите количество бойцов, которые поучаствуют в его доставке, а в третьей строке выведите через пробел их номера в порядке от Анки к Чапаеву. Бойцы 25-й дивизии занумерованы числами от 1 до *n* в том порядке, в котором описаны номера их мобильных телеграфов на входе. Если существует несколько способов передать сообщение за минимальное время, выведите любой из них.

## Описание решения

**Пояснение к примененному алгоритму:**

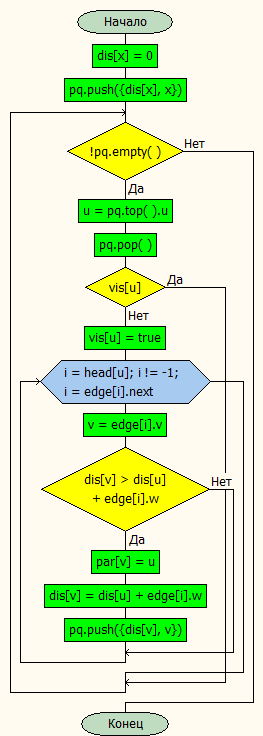
Разделим задачу на 2 подзадачи:

* построить граф;

Чтобы построить граф, необходимо при добавлении нового телеграфа проверять все возможные комбинации для определения всех связей. Т.е. будем менять каждую цифру на другую и попробуем переставить каждую пару чисел в номере телеграфа. Если получается номер уже существующего телеграфа, то будем добавлять связь между вершинами (вес ребра будет составлять длину общего префикса).

* найти кратчайший путь (алгоритм Дейкстры).

# Схема решения



# Исходный код

#include <bits/stdc++.h>

using namespace std;

const int N = 50001;

const int M = 20000001;

const int max\_ = 0x3f3f3f3f;

int n;

int valForTransition[15];

string telegraphNumber[N];

map<long long, int> node;

struct edge {

int v, w, next;

} edge[M];

int head[N], ec;

int dis[N], par[N];

bool vis[N];

vector<int> vec;

struct pp{

int d, u;

bool operator < (const pp &cmp) const {

return d > cmp.d;

}

};

// метод для добавления связи в граф

void add\_edge(int u, int v, int w) {

edge[ec] = {v, w, head[u]};

head[u] = ec++;

}

vector<long long> py;

//метод, определяет все связи между телеграфами

void deal(int id) {

long long tmp = 0;

// восстановление номера телеграфа

for (int i = 0; i < 10; ++i) {

tmp = tmp \* 10 + telegraphNumber[id][i] - '0';

}

const long long tt = tmp;

//поиск связи при замене одной цифры

for (int i = 0; i < 10; ++i) {

for (int j = 0; j < 10; ++j) {

tmp = tmp - (telegraphNumber[id][i] - '0' - j) \* py[i];

auto it = node.find(tmp);

if (it != node.end()) {

int len = 0;

int idx = it->second;

while (len < 9 && telegraphNumber[id][len] == telegraphNumber[idx][len]) {

len++;

}

add\_edge(id, idx, valForTransition[len]);

add\_edge(idx, id, valForTransition[len]);

}

tmp = tt;

}

}

//поиск связи при смене 2-х цифр местами

for (int i = 0; i < 10; ++i) {

for (int j = i + 1; j < 10; ++j) {

int t1 = telegraphNumber[id][i] - '0';

int t2 = telegraphNumber[id][j] - '0';

tmp = tmp - (t1 - t2) \* py[i] - (t2 - t1) \* py[j];

auto it = node.find(tmp);

if (it != node.end()){

int len = 0;

int idx = it->second;

while (len < 9 && telegraphNumber[id][len] == telegraphNumber[idx][len]){

len++;

}

add\_edge(id, idx, valForTransition[len]);

add\_edge(idx, id, valForTransition[len]);

}

tmp = tt;

}

}

node.insert({tt, id});

}

// Алгоритм Дейкстры

void Dijkstra(int x){

priority\_queue<pp> pq;

dis[x] = 0;

pq.push({dis[x], x});

while (!pq.empty()){

int u = pq.top().u;

pq.pop();

if (vis[u]) {

continue;

}

vis[u] = true;

for (int i = head[u]; i != -1 ; i = edge[i].next) {

int v = edge[i].v;

if (dis[v] > dis[u] + edge[i].w){

par[v] = u;

dis[v] = dis[u] + edge[i].w;

pq.push({dis[v], v});

}

}

}

}

int main() {

long long tmp = 1;

// заполняем вектор необходимыми значениями

for (int i = 0; i < 10; ++i) {

py.push\_back(tmp);

tmp \*= 10;

}

reverse(py.begin(), py.end());

// считываем кол-во бойцов в дивизии

cin >> n;

memset(head, -1, sizeof(head));

ec = 0; // счетчик узлов

// считываем время передачи сообщений

for (int i = 0; i < 10; ++i) {

cin >> valForTransition[i];

}

// считываем номера телеграфов, ищем связи

for (int i = 1; i <= n; ++i) {

cin >> telegraphNumber[i];

deal(i);

}

memset(dis, max\_, sizeof(dis));

par[1] = -1;

Dijkstra(1); // применяем алгоритм Дейкстры

// если сообщение передать нельзя

if (dis[n] == max\_){

cout << "-1";

return 0;

}

// вывод минимального времени доставки сообщения

cout << dis[n] << "\n";

// собираем наш кратчайший путь с конца - от Чапаева

int x = n;

while(x != -1){

vec.push\_back(x);

x = par[x];

}

reverse(vec.begin(), vec.end());

// вывод кол-ва солдат, участвовавших в доставке

cout << vec.size() << "\n";

// вывод номеров этих солдат

for (int i = 0; i < vec.size(); ++i){

cout << vec[i] << (i == vec.size() - 1? "\n" : " ");

}

return 0;

}

# Вывод:

в данной задаче я применила алгоритм Дейкстры для поиска кратчайшего пути.