## 68. Metro

The metro system consists of *n* stations located on *l* lines. Each station belongs to one or several lines (in this case, a passenger can transfer to any of the lines passing through this station). Each line includes at least two stations and intersects with at least one other line. The metro network is connected.

Travel between two adjacent stations on the same line is possible in both directions and takes 2 minutes. A transfer from one line to another within the same station takes 1 minute. Other time costs can be neglected.

Determine the minimum time required for the manager of the company “Diez-Product” to travel from station *a* to the company’s office located near station *b*.

**Input.** The first line contains two positive integers *n* and *l* (1 ≤ *l* ≤ 10). Each of the next *l* lines lists the consecutive station numbers of each metro line. The last line contains the numbers and line indices of the starting and destination stations. All numbers are positive and do not exceed  70.

**Output.** Print the minimum travel time between the specified stations.

|  |  |
| --- | --- |
| **Sample input** | **Sample output** |
| 7 32 1 36 1 4 55 72 1 7 3 | 10 |

**SOLUTION**

**graphs – Dijkstra**

**Анализ алгоритма**

Кратчайшее расстояние от источника до вершины *v*, принадлежащей линии *l*, храним в dist[*v*][*l*], где 1 ≤ *v* ≤ 70, 1 ≤ *l* ≤ 10. Граф храним в виде списков смежности. Вместе с направлением и стоимостью ребра храним номер линии метро, которой принадежит вершина куда направлено ребро.

Стартуем со станции *s*, находящейся на линии *line*1. Ответ на задачу – минимальная стоимость, за которую можно добраться до вершины *f* на линии *line*2. При движении между вершинами следует совершить проверку: если вершины принадлежат разным линиям, то следует добавить 1 (совершить пересадку).

При помощи алгоритма Дейкстры находим кратчайшее расстояния из сосотяния (*s*, *line*1) в (*f*, *line*2).

**Example**

In the given example, it is required to find the shortest route from station 2 on line 1 to station 7 on line 3.



* Move along line 1 from station 2 to station 1: 2 minutes.
* Transfer at station 1 from line 1 to line 2: 1 minute.
* Move along line 2 from station 1 to station 5: 4 minutes.
* Transfer at station 5 from line 2 to line 3: 1 minute.
* Move along line 3 from station 5 to station 7: 2 minutes.

Thus, the entire trip takes 10 minutes.

**Реализация алгоритма**

Количество вершин в графе не более MAX = 71. Количество линий не более LINES = 11.

#define MAX 71

#define LINES 11

Кратчайшее расстояние от источника до вершины *v*, принадлежащей линии *l*, храним в dist[*v*][*l*]. Объявим массив кратчайших расстояний.

int dist[MAX][LINES];

Структура node содержит информацию о ребре:

* *to* – направление ребра;
* *cost* – стоимость ребра;
* *line* – какой линии метро ребро принадлежит;

struct node

{

 int to, cost, line;

 node(int to, int cost, int line) : to(to), cost(cost), line(line) {}

};

Структура node также будет использоваться в очереди с приоритетами, где

* *to* – текущая вершина;
* *cost* – минимальная стоимость проезда до вершины *to* (на линии *line*);
* *line* – номер линии метро, на которой находится вершина *to*;

В очереди с приоритетами вершины сортируются по возрастанию стоимости проезда.

bool operator< (node a, node b)

{

 return a.cost > b.cost;

}

Список смежности вершин графа.

vector<vector<node> > g;

Основная часть программы. Читаем входные данные.

scanf("%d %d", &n, &l);

g.resize(n + 1);

Читаем *i*-ую линию метро. Время движения между соседними станциями *u* и *v* равно 2.

for (i = 1; i <= l; i++)

{

Читаем станцию *u* – первую станцию *i*-ой линии метро. За станцией *u* идет станция *v*.

 scanf("%d ", &u);

 while (scanf("%d%c", &v, &ch))

 {

Движение возможно в обоих направлениях. Стоимость ребра равна 2.

 g[u].push\_back(node(v, 2, i));

 g[v].push\_back(node(u, 2, i));

 u = v;

Как только доходим до конца строки – заканчиваем читать данные очередной линии метро.

 if (ch == '\n') break;

 }

}

Инициализируем массив кратчайших расстояний.

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

 for (j = 1; j <= l; j++)

 dist[i][j] = 2000000000;

Читаем информацию о начальной и конечной станции. Для каждой станции задается линия, которой она принадлежит.

 scanf("%d %d %d %d", &s, &line1, &f, &line2);

Стартуем со станции *s*, находящейся на линии *line*1.

 dist[s][line1] = 0;

 priority\_queue<node, vector<node> > pq;

Заносим в очередь структуру node, содержащую начальную вершину *s* на линии *line*1. Стоимость попадания в начальную вершину равна 0.

pq.push(node(s, 0, line1)); // (v, cost, line)

while (!pq.empty())

{

Извлекаем из очереди вершину *v* с наименьшей стоимостью.

 int cost = pq.top().cost;

 int v = pq.top().to;

 int l1 = pq.top().line;

 pq.pop(); // dist[v][l1] = cost

Перебираем вершины, связанные с *v*.

 for (i = 0; i < g[v].size(); i++)

 {

Имеется ребро *v* – *to* стоимостью *w*, принадлежащее линии *l*2.

 int to = g[v][i].to;

 int w = g[v][i].cost;

 int l2 = g[v][i].line;

Если двигаться в *to* по ребру *v* – *to*, то стоимость составит *cost* + *w*.

 int tot\_cost = cost + w;

Если линии у вершин *v* и *to* разные, то следует совершить пересадку с линии *l*1 на линию *l*2.

 if (l1 != l2) tot\_cost += 1;

Если движение по ребру *v* – *to* дает лучший результат, то улучшаем значение dist[*to*][*l*2] и заносим вершину *to* в очередь (стоимость попадания в нее составляет *tot\_cost* и она находится на линии *l*2).

 if (tot\_cost < dist[to][l2])

 {

 dist[to][l2] = tot\_cost;

 pq.push(node(to, tot\_cost, l2));

 }

 }

}

Вывод ответа – стоимости добраться до вершины *f* на линии *line*2.

printf("%d\n", dist[f][line2]);