

## CÁCH ĐÁNH GIÁ ĐIỂM THỰC HÀNH

### HỌC PHẦN: IT3150 – Project 1- 2023.1

#### I. Quy định, yêu cầu:

- Tài liệu và nội dung thực hành chấm điểm trên hệ thống:  
<https://lab.soict.hust.edu.vn/>
- Bài tập trên lớp chấm điểm tự động (các bài không chấm trên hệ thống làm vào máy tính → làm báo cáo thực hành – Theo mẫu).
- Hạn nộp báo cáo trên Teams (Bài tập trên lớp + Bài tập về nhà): 1 tuần.

#### II. Đánh giá điểm thực hành

- Chuyên cần (đúng giờ, nghiêm túc trong giờ học) - Điểm danh trên Teams: 10%
- Báo cáo thực hành (bài tập trên lớp + Về nhà) theo mẫu nộp trên Teams: 40%
- Trắc nghiệm – Form trên Teams: 10%
- Kiểm tra thực hành: 40%. (Tiết 2,3 buổi thực hành thứ 5).

**Điểm thưởng: 5% → 10% (Cho Mục 1,2 điểm TB từ 9-10).**

Tham gia thực hành đúng giờ đầy đủ theo thời khóa biểu (nếu có lý do không đi thực hành đúng giờ được thì gửi mail xin phép thực hành bù trước 1 ngày qua mail [hoalt@soict.hust.edu.vn](mailto:hoalt@soict.hust.edu.vn), Tiêu đề: đăng ký học bù – IT3040 – MaLopTH.

Các kíp có thể bù:

TT	Thời gian, địa điểm, Tuần học	Mã nhóm	Mã lớp
1			
2			
3			
4			
5			
6			
7			

**Nếu nghỉ không có lý do 3 buổi, không thực hành bù thì điểm chuyên cần, báo cáo và BTVN coi như 0 điểm thực hành.**

## Contents

<b>Bài tập 1:</b> Lưu lượng tối đa .....	3
<b>Bài tập 2:</b> Đường dẫn ngắn nhất giữa 2 nút trên biểu đồ có hướng có trọng số không âm .....	11
<b>Bài tập 3:</b> Tất cả các cặp đường đi ngắn nhất .....	16

## Table of figures

Figure 1 code bài 1 .....	4
Figure 2 test bài 1 .....	7
Figure 3 code bài 2 .....	11
Figure 4 test bài 2 .....	12
Figure 5 code bài 3 .....	17
Figure 6 test bài 3 .....	18

## Báo cáo tuần 6

### Bài tập 1: Lưu lượng tối đa

---

Cho mạng  $G = (V, E)$  là đồ thị có trọng số có hướng. Nút  $s$  là nguồn và nút  $t$  là đích.  $c(u,v)$  là dung lượng của cung  $(u,v)$ . Tìm luồng cực đại trên  $G$ .

#### Đầu vào

- Dòng 1: hai số nguyên dương  $N$  và  $M$  ( $1 \leq N \leq 104$ ,  $1 \leq M \leq 106$ )
- Dòng 2: chứa 2 số nguyên dương  $s$  và  $t$
- Dòng  $i+2$  ( $i = 1, \dots, M$ ): chứa hai số nguyên dương  $u$  và  $v$  là điểm cuối của  $i$  quàn què vòng cung

#### đầu ra

Viết giá trị của luồng cực đại được tìm thấy

Figure 1 code bài 1

## Source code

```

1 //Mai Minh Hoàng
2 //20215381
3 #include <iostream>
4 #include<vector>
5 #define MAX_N 10000
6 using namespace std;
7 class Edge {//định nghĩa cạnh chứa đỉnh tới và trọng số
8 public:
9     int v, w;
10    Edge(int v, int w) {
11        this->v = v;
12        this->w = w;
13    }
14 };
15 int visited[MAX_N];//lưu lại các cạnh đã thăm trong lần xét đó
16 int trace[MAX_N];//mảng lưu lại đỉnh được duyệt trước đó( để lấy đường đi từ nguồn tới đích)
17 vector<Edge> listEdge[MAX_N];//đồ thị thật
18 vector<Edge> f[MAX_N];//đồ thị với trọng số là giá trị đã sử dụng
19 int n,m,s, t;
20 void BFS(int u, int sink) {//duyet bắt đầu từ đỉnh nguồn
21     visited[u] = 1;// đánh dấu nguồn đã được thăm
22
23     vector<int> queue;// hàng đợi các đỉnh đã thăm
24     queue.push_back(u);// thêm nguồn vào hàng đợi
25
26     while (!queue.empty()) {// cho tới khi hàng đợi còn phần tử
27         int t = queue[0];//lấy ra phần tử ở trước
28         queue.erase(queue.begin());// xóa nó khỏi hàng đợi
29         for (const Edge& edge : f[t]) {// với mỗi cạnh có đường đi từ đỉnh đang được lấy ra tới đỉnh khác
30             int w = edge.w;//trọng số của cạnh

```

```

31     int v = edge.v; //đỉnh đi tới
32     if (!visited[edge.v] && edge.w < listEdge[t][edge.v].w) { //nếu đỉnh đi tới chưa được thăm và tr
33         queue.push_back(v); //thêm đỉnh đó vào hàng đợi
34         visited[v] = 1; // đánh dấu đã thăm
35         trace[v] = t; //đỉnh ở trước v là t
36         if (v == t) return; //nếu v là t tức là đã tìm được đường đi tới đích
37     }
38 }
39 }
40
41 }
42 int find_augment_from_to(int source, int sink) { //kiểm tra xem còn có đường tăng từ nguồn tới đích không
43     for (int i = 0; i <= n; i++) { // mỗi lần dùng hàm kiểm tra này thì reset lại trạng thái của mảng đã thăm
44         visited[i] = 0;
45         trace[i] = 0;
46     }
47     BFS(source, sink); //dùng BFS để duyệt lần lượt từ nguồn tới đích
48     return visited[sink]; //nếu đỉnh đích đã được thăm tức là còn đường tăng luồng sẽ trả về 1 nếu không thì
49 }
50 }
51 void increase_flow( int source, int sink) { //tăng luồng khi còn có thể tăng
52
53     int minCapacity = 999999; //gán giá trị vô cùng
54     int u = sink; //đầu tiên là gán u là đích
55     while (u != source) { // khi u còn khác nguồn
56         int previousNode = trace[u]; //lấy đỉnh được thăm trước trước của đỉnh đó
57         minCapacity = (minCapacity < (listEdge[previousNode][u].w -
58             f[previousNode][u].w)) ? minCapacity:
59             (listEdge[previousNode][u].w - f[previousNode][u].w); //nếu giá trị đường tăng lớn hơn trọng số
60         u = previousNode; //gán u là đỉnh trước đó để vào vòng lặp mới
61     }
62 }

```

```

62
63     while (sink != source) {//vòng lặp khác để sửa giá trị trên đồ thị
64
65         int previousNode = trace[sink];
66         f[previousNode][sink].w += minCapacity;//cạnh xuôi dòng thì cộng thêm minCapacity
67         f[sink][previousNode].w -= minCapacity;//cạnh ngược dòng thì trừ đi giá trị đó
68         sink = previousNode;
69     }
70
71 }
72 int main()
73 {
74     cin >> n >> m;
75     cin >> s >> t;
76     for (int i = 0; i <= n; i++) {//khởi tạo các cạnh ban đầu trọng số đều -1
77         for (int j = 0; j <= n; j++) {
78             listEdge[i].push_back(Edge(j, -1));
79             f[i].push_back(Edge(j, -1));
80         }
81     }
82     for (int i = 0; i < m; i++) {//với mỗi cạnh được nhập vào thì sửa trọng số của từng cạnh
83         int u, v, w;
84         cin >> u >> v >> w;
85         listEdge[u][v].w = w;
86         f[u][v].w=0;// cạnh xuôi dòng giá trị ban đầu là 0
87         f[v][u].w = w;//cạnh ngược dòng giá trị ban đầu là trọng số tối đa cạnh
88     }
89
90     while (find_augment_from_to(s, t)) {//khi còn đường tăng luồng thì gọi hàm tăng luồng
91         increase_flow(s, t);
92     }
93 }
94
95 int max_flow = 0;//biến lưu giá trị luồng cực đại
96 for (const Edge& edge : listEdge[s]) {//cộng tất cả các luồng phát ra từ nguồn
97     if (edge.w >= 0) {
98         max_flow += f[s][edge.v].w;
99     }
100 }
101 cout << max_flow << "\n";
102 return 0;
103 }

```

Figure 2 test bài 1

Input

```
7 12
1 7
1 6 8
1 2 9
2 5 1
2 4 6
3 7 6
4 3 5
4 7 7
5 3 2
5 4 4
6 2 2
6 3 3
6 5 4
```



Correct output

13



User output

13



Input

```
8 14
1 4
1 2 7
1 3 6
1 8 2
2 3 5
2 5 2
3 6 9
3 7 8
3 8 4
5 4 8
6 4 2
7 5 1
7 6 4
8 6 3
8 4 5
```



Correct output

10



User output

10



Input

```
10 21
1 10
1 6 8
1 2 9
1 6 4
2 5 1
2 4 6
3 7 6
4 3 5
4 7 7
5 3 2
5 4 4
6 2 2
6 3 3
6 5 4
6 7 5
7 8 2
8 9 8
```



```
9 10 2
7 10 4
7 9 2
8 10 1
5 10 4
```

Correct output

11



User output

11



Code:

```
//Mai Minh Hoàng
//20215381
#include <iostream>
#include<vector>
#define MAX_N 10000
using namespace std;
```



```

class Edge { //định nghĩa cạnh chứa đỉnh tới và trọng số
public:
    int v, w;
    Edge(int v, int w) {
        this->v = v;
        this->w = w;
    }
};

int visited[MAX_N]; //lưu lại các cạnh đã thăm trong lần xét đó
int trace[MAX_N]; //mảng lưu lại đỉnh được duyệt trước đó( để lấy đường đi từ nguồn tới đích)
vector<Edge> listEdge[MAX_N]; //đồ thị thật
vector<Edge> f[MAX_N]; //đồ thị với trọng số là giá trị đã sử dụng
int n, m, s, t;
void BFS(int u, int sink) { //duyet bắt đầu từ đỉnh nguồn
    visited[u] = 1; // đánh dấu nguồn đã được thăm

    vector<int> queue; // hàng đợi các đỉnh đã thăm
    queue.push_back(u); // thêm nguồn vào hàng đợi

    while (!queue.empty()) { // cho tới khi hàng đợi còn phần tử
        int t = queue[0]; //lấy ra phần tử ở trước
        queue.erase(queue.begin()); // xóa nó khỏi hàng đợi
        for (const Edge& edge : f[t]) { // với mỗi cạnh có đường đi từ đỉnh đang được lấy ra tới đỉnh khác
            int w = edge.w; //trọng số của cạnh
            int v = edge.v; //đỉnh đi tới
            if (!visited[edge.v] && edge.w < listEdge[t][edge.v].w) { //nếu đỉnh đi tới chưa được thăm và trọng số tối
                đa của cạnh lớn hơn giá trị đã được dùng
                queue.push_back(v); //thêm đỉnh đó vào hàng đợi
                visited[v] = 1; // đánh dấu đã thăm
                trace[v] = t; //đỉnh ở trước v là t
                if (v == t) return; //nếu v là t tức là đã tìm được đường đi tới đích
            }
        }
    }
}

int find_augment_from_to(int source, int sink) { //kiểm tra xem còn có đường tăng từ nguồn tới đích không
    for (int i = 0; i <= n; i++) { // mỗi lần dùng hàm kiểm tra này thì reset lại trạng thái của mảng đã thăm và mảng
        đường đi
        visited[i] = 0;
        trace[i] = 0;
    }
    BFS(source, sink); //dùng BFS để duyệt lần lượt từ nguồn tới đích
    return visited[sink]; //nếu đỉnh đích đã được thăm tức là còn đường tăng luồng sẽ trả về 1 nếu không thì trả về
    0
}

void increase_flow( int source, int sink) { //tăng luồng khi còn có thể tăng

    int minCapacity = 999999; //gán giá trị vô cùng
    int u = sink; //đầu tiên là gán u là đích
    while (u != source) { // khi u còn khác nguồn

```

**Mai Minh Hoàng – 20215381**

```
int previousNode = trace[u]; //lấy đỉnh được thăm trước trước của đỉnh đó
minCapacity = (minCapacity < (listEdge[previousNode][u].w -
    f[previousNode][u].w)) ? minCapacity:
    (listEdge[previousNode][u].w - f[previousNode][u].w); //nếu giá trị đường tăng lớn hơn trọng số của cạnh
đang được xét thì gán giá trị mới là giá trị cạnh đang được xét
u = previousNode; //gán u là đỉnh trước đó để vào vòng lặp mới
}

while (sink != source) { //vòng lặp khác để sửa giá trị trên đồ thị

    int previousNode = trace[sink];
    f[previousNode][sink].w += minCapacity; //cạnh xuôi dòng thì cộng thêm minCapacity
    f[sink][previousNode].w -= minCapacity; //cạnh ngược dòng thì trừ đi giá trị đó
    sink = previousNode;
}

}
int main()
{
    cin >> n >> m;
    cin >> s >> t;
    for (int i = 0; i <= n; i++) { //khởi tạo các cạnh ban đầu trọng số đều -1
        for (int j = 0; j <= n; j++) {
            listEdge[i].push_back(Edge(j, -1));
            f[i].push_back(Edge(j, -1));
        }
    }
    for (int i = 0; i < m; i++) { //với mỗi cạnh được nhập vào thì sửa trọng số của từng cạnh
        int u, v, w;
        cin >> u >> v >> w;
        listEdge[u][v].w = w;
        f[u][v].w = 0; // cạnh xuôi dòng giá trị ban đầu là 0
        f[v][u].w = w; //cạnh ngược dòng giá trị ban đầu là trọng số tối đa cạnh
    }

    while (find_augment_from_to(s, t)) { //khi còn đường tăng luồng thì gọi hàm tăng luồng
        increase_flow(s, t);
    }
    int max_flow = 0; //biến lưu giá trị luồng cực đại
    for (const Edge& edge : listEdge[s]) { //cộng tất cả các luồng phát ra từ nguồn
        if (edge.w >= 0) {
            max_flow += f[s][edge.v].w;
        }
    }
    cout << max_flow << "\n";
    return 0;
}
```

## Bài tập 2: Đường dẫn ngắn nhất giữa 2 nút trên biểu đồ có hướng có trọng số không âm

Cho đồ thị có hướng  $G = (V, E)$  trong đó  $V = \{1, 2, \dots, n\}$  là tập hợp các nút. Mỗi cung  $(u, v)$  có trọng số không âm  $w(u, v)$ . Cho hai nút  $s$  và  $t$  của  $G$ . Tìm đường đi ngắn nhất từ  $s$  đến  $t$  trên  $G$ .

### Đầu vào

- Dòng 1: chứa hai số nguyên  $n$  và  $m$  là số nút và số cung của  $G$  ( $1 \leq n \leq 100000$ )
- Dòng  $i + 1$  ( $i = 1, 2, \dots, m$ ): chứa 3 số nguyên  $u, v, w$  trong đó  $w$  là trọng số của cung  $(u, v)$  ( $0 \leq w \leq 100000$ )
- Dòng  $m+2$ : chứa hai số nguyên  $s$  và  $t$

### Đầu ra

Viết trọng số của đường đi ngắn nhất được tìm thấy hoặc viết -1 nếu không tìm thấy đường đi nào từ  $s$  đến  $t$

Figure 3 code bài 2

Source code

```

1 //Mai Minh Hoàng
2 //20215381
3 #include <iostream>
4 #include <vector>
5 #include <queue>
6 #include <limits>
7
8 #define MAX_N 100100 //định nghĩa hằng số max
9
10 using namespace std;
11
12 class Edge { //định nghĩa lớp Edge là cạnh chứa đỉnh tới và trọng số
13 public:
14     int v, w;
15     Edge(int v, int w) {
16         this->v = v; //v là đỉnh tới
17         this->w = w; //w là trọng số
18     }
19 };
20
21 vector<Edge> listEdge[MAX_N]; //mảng chứa vector, mỗi vector i là một tập các cạnh đi từ đỉnh thứ i
22 int dist[MAX_N]; //khoảng cách từ đỉnh thứ i tới đỉnh nguồn
23
24 void dijkstra(int start, int n) { // hàm dijkstra tìm đường nhỏ nhất
25     priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq; //hàng đợi ưu tiên chứa cặp khóa và giá trị, k
26     pq.push({0, start}); //đặt vào hàng đợi đỉnh đầu tiên là nguồn
27     for (int i = 1; i <= n; i++) {
28         dist[i] = INT_MAX; //khởi tạo tất cả khoảng cách các đỉnh tới nguồn là vô cùng
29     }
30     dist[start] = 0; //khoảng cách từ nguồn tới nguồn là 0
31
32     while (!pq.empty()) { //khi hàng đợi còn chưa rỗng
33         int u = pq.top().second; //lấy ra phần tử ở đầu hàng đợi, u là đỉnh
34         int d = pq.top().first; //d là khoảng cách từ đỉnh đó tới nguồn
35         pq.pop();
36         for (int i = 0; i < listEdge[u].size(); i++) {
37             int v = listEdge[u][i].v;
38             int w = listEdge[u][i].w;
39             if (dist[v] > d + w) {
40                 dist[v] = d + w;
41                 pq.push({dist[v], v});
42             }
43         }
44     }
45 }
46
47 int main() {
48     int n, m;
49     cin >> n >> m;
50     for (int i = 1; i <= m; i++) {
51         int u, v, w;
52         cin >> u >> v >> w;
53         listEdge[u].push_back(Edge(v, w));
54     }
55     int s, t;
56     cin >> s >> t;
57     dijkstra(s, n);
58     if (dist[t] == INT_MAX) {
59         cout << -1 << endl;
60     } else {
61         cout << dist[t] << endl;
62     }
63     return 0;
64 }
```

```

34     int d = pq.top().first; //d là khoảng cách từ đỉnh đó tới nguồn
35     pq.pop();
36
37     if (d > dist[u]) continue; //nếu khoảng cách tới nguồn lớn hơn khoảng cách nhỏ nhất hiện tại từ đỉnh đó tới nguồn thì bỏ qua
38
39     for (const Edge& edge : listEdge[u]) { //nếu không thì với mỗi cạnh bắt đầu bởi đỉnh u tới đỉnh khác
40         int v = edge.v; //lấy ra đỉnh còn lại và trọng số cạnh
41         int w = edge.w;
42         if (dist[u] + w < dist[v]) { //nếu trọng số cộng với khoảng cách từ u tới nguồn mà bé hơn khoảng cách nhỏ nhất hiện tại t
43             dist[v] = dist[u] + w; // thì cập nhật khoảng cách nhỏ nhất từ v tới nguồn
44             pq.push({ dist[v], v }); //thêm đỉnh vào hàng đợi
45         }
46     }
47 }
48 }
49
50 int main() { //bắt đầu chương trình
51     ios_base::sync_with_stdio(0); //tắt đồng bộ nhập xuất để tăng tốc độ
52     cin.tie(0);
53     cout.tie(0);
54     int n, m, s, t;
55     cin >> n >> m; //nhập n và m
56
57     for (int i = 0; i < m; i++) { //lần lượt nhập các cạnh
58         int u, v, w;
59         cin >> u >> v >> w;
60         listEdge[u].push_back(Edge(v, w)); //lưu cách cạnh vào hàng đợi
61     }
62     cin >> s >> t;
63     dijkstra(s, n); //tìm đường ngắn nhất
64     cout << dist[t]; //in ra giá trị đường ngắn nhất tới đích
65     return 0;
66 }

```

Figure 4 test bài 2

## Input

```

5 7
2 5 87
1 2 97
4 5 78
3 1 72
1 4 19
2 3 63
5 1 18
1 5

```

## Correct output

97

## User output

97

Input

```
100 500
76 83 55
86 83 78
43 83 23
35 86 96
30 43 84
64 86 66
62 30 52
74 76 70
89 74 43
95 35 40
53 43 11
17 30 19
16 89 7
57 16 38
60 64 36
81 35 13
90 53 85
```

...

Correct output

87

User output

87

Input

```
1000 10000
324 798 4
467 324 19
175 324 60
7 175 8
632 798 66
729 324 90
934 175 75
297 798 53
560 798 35
953 632 38
735 934 99
138 735 98
151 953 53
229 632 30
787 151 86
831 953 47
```

...

```
315 537 14
589 397 29
38 885 85
399 631 31
560 680 87
63 224 24
349 429 1
374 895 73
360 131 79
171 437 53
217 781 42
92 309 56
785 150 5
1 1000
```

Correct output

86



User output

86



Input

```
10000 100000
170 1301 22
6477 1301 32
7306 170 35
6652 1301 4
7642 6477 74
6458 7306 2
1861 7306 31
5707 7642 12
6568 7306 57
8071 7642 72
1983 6477 87
6910 8071 58
2721 6458 42
9019 6910 84
5202 6568 64
1061 1301 2
```



....

```
6370 462 27
2705 5473 95
6194 6133 75
8524 1929 42
9238 8743 97
509 9236 37
7897 3027 57
127 9919 23
1545 9527 96
1698 7538 73
8645 6359 78
4458 4766 1
6175 2405 8
5305 5856 50
2401 5512 55
758 4614 99
6401 8143 68
8211 1421 36
1 10000
```

Correct output

108



User output

108



## Code:

```
//Mai Minh Hoàng
//20215381
#include <iostream>
#include <vector>
#include <queue>
#include <climits>

#define MAX_N 100100 //định nghĩa hằng số max

using namespace std;

class Edge { //định nghĩa lớp Edge là cạnh chứa đỉnh tới và trọng số
public:
    int v, w;
    Edge(int v, int w) {
        this->v = v; //v là đỉnh tới
        this->w = w; //w là trọng số
    }
};

vector<Edge> listEdge[MAX_N]; //mảng chứa vector, mỗi vector i là một tập các cạnh đi từ đỉnh thứ i
```

```

int dist[MAX_N]; // khoảng cách từ đỉnh thứ i tới đỉnh nguồn

void dijkstra(int start, int n) { // hàm dijkstra tìm đường nhỏ nhất
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq; // hàng đợi ưu tiên chứa
    cặp khóa và giá trị, khóa nhỏ hơn thì ra trước (khóa là khoảng cách tới nguồn)
    pq.push({ 0, start }); // đặt vào hàng đợi đỉnh đầu tiên là nguồn
    for (int i = 1; i <= n; i++) {
        dist[i] = INT_MAX; // khởi tạo tất cả khoảng cách các đỉnh tới nguồn là vô cùng
    }
    dist[start] = 0; // khoảng cách từ nguồn tới nguồn là 0

    while (!pq.empty()) { // khi hàng đợi còn chưa rỗng
        int u = pq.top().second; // lấy ra phần tử ở đầu hàng đợi, u là đỉnh
        int d = pq.top().first; // d là khoảng cách từ đỉnh đó tới nguồn
        pq.pop();

        if (d > dist[u]) continue; // nếu khoảng cách tới nguồn lớn hơn khoảng cách nhỏ nhất hiện tại từ đỉnh đó tới
        nguồn thì bỏ qua

        for (const Edge& edge : listEdge[u]) { // nếu không thì với mỗi cạnh bắt đầu bởi đỉnh u tới đỉnh khác
            int v = edge.v; // lấy ra đỉnh còn lại và trọng số cạnh
            int w = edge.w;
            if (dist[u] + w < dist[v]) { // nếu trọng số cộng với khoảng cách từ u tới nguồn mà bé hơn khoảng cách
            nhỏ nhất hiện tại từ v tới nguồn
                dist[v] = dist[u] + w; // thì cập nhật khoảng cách nhỏ nhất từ v tới nguồn
                pq.push({ dist[v], v }); // thêm đỉnh vào hàng đợi
            }
        }
    }
}

int main() { // bắt đầu chương trình
    ios_base::sync_with_stdio(0); // tắt đồng bộ nhập xuất để tăng tốc độ
    cin.tie(0);
    cout.tie(0);
    int n, m, s, t;
    cin >> n >> m; // nhập n và m

    for (int i = 0; i < m; i++) { // lần lượt nhập các cạnh
        int u, v, w;
        cin >> u >> v >> w;
        listEdge[u].push_back(Edge(v, w)); // lưu cách cạnh vào hàng đợi
    }
    cin >> s >> t;
    dijkstra(s, n); // tìm đường ngắn nhất
    cout << dist[t]; // in ra giá trị đường ngắn nhất tới đích
    return 0;
}

```

## Bài tập 3: Tất cả các cặp đường đi ngắn nhất



## Mai Minh Hoàng – 20215381

Cho đồ thị có hướng  $G = (V, E)$  trong đó  $V = \{1, 2, \dots, n\}$  là tập hợp các nút và  $w(u,v)$  là trọng số (độ dài) của cung  $(u,v)$ . Tính  $d(u,v)$  - độ dài đường đi ngắn nhất từ  $u$  đến  $v$  trong  $G$ , với mọi  $u,v$  trong  $V$ .

### Đầu vào

- Dòng 1: chứa 2 số nguyên dương  $n$  và  $m$  ( $1 \leq n, m \leq 10000$ )
- Dòng  $i+1$  ( $i = 1, 2, \dots, m$ ): chứa 3 số nguyên dương  $u, v, w$  trong đó  $w$  là trọng số của cung  $(u,v)$  ( $1 \leq w \leq 1000$ )

### Đầu ra

- Dòng  $i$  ( $i = 1, 2, \dots, n$ ): ghi dòng thứ  $i$  của ma trận  $d$  (nếu không có đường đi từ nút  $i$  đến nút  $j$  thì  $d(i,j) = -1$ )

Figure 5 code bài 3

Source code

```
1 //Mai Minh Hoàng
2 //20215381
3 #include <iostream>
4 #include <vector>
5 #include <queue>
6 #include <climits>
7
8 #define MAX_N_81 100 //định nghĩa hằng số max
9
10 using namespace std;
11
12 class Edge_81 { //định nghĩa lớp Edge là cạnh chứa đỉnh tới và trọng số
13 public:
14     int v_81, w_81;
15     Edge_81(int v_81, int w_81) {
16         this->v_81 = v_81; //v là đỉnh tới
17         this->w_81 = w_81; //w là trọng số
18     }
19 };
20
21 vector<Edge_81> listEdge_81[MAX_N_81]; //mảng chứa vector, mỗi vector i là một tập các cạnh đi từ đỉnh thứ i
22 int dist_81[MAX_N_81][MAX_N_81]; //khoảng cách từ đỉnh thứ i tới đỉnh j
23
24 void dijkstra_81(int start_81, int n_81) { // hàm dijkstra tìm đường nhỏ nhất
25     priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq_81; //hàng đợi ưu tiên chứa cặp khóa và giá trị
26     pq_81.push({ 0, start_81 }); //đặt vào hàng đợi đỉnh đầu tiên là nguồn
27     for (int i_81 = 1; i_81 <= n_81; i_81++) {
28         dist_81[start_81][i_81] = INT_MAX; //khởi tạo tất cả khoảng cách các đỉnh tới nguồn là vô cùng
29     }
30     dist_81[start_81][start_81] = 0; //khoảng cách từ nguồn tới nguồn là 0
31
32     while (!pq_81.empty()) { //khi hàng đợi còn chưa rỗng
33         int u_81 = pq_81.top().second; //lấy ra phần tử ở đầu hàng đợi, u là đỉnh
34         int d_81 = pq_81.top().first; //d là khoảng cách từ đỉnh đó tới nguồn
```

```

35     pq_81.pop();
36
37     if (d_81 > dist_81[start_81][u_81]) continue; //nếu khoảng cách tới nguồn lớn hơn khoảng cách nhỏ nhất hiện tại từ đỉnh đó tới
38
39     for (const Edge_81& edge_81 : listEdge_81[u_81]) { //nếu không thì với mỗi cạnh bắt đầu bởi đỉnh u tới đỉnh khác
40         int v_81 = edge_81.v_81; //lấy ra đỉnh còn lại và trọng số cạnh
41         int w_81 = edge_81.w_81;
42         if (dist_81[start_81][u_81] + w_81 < dist_81[start_81][v_81]) { //nếu trọng số cộng với khoảng cách từ u tới nguồn mà bé
43             dist_81[start_81][v_81] = dist_81[start_81][u_81] + w_81; // thì cập nhật khoảng cách nhỏ nhất từ v tới nguồn
44             pq_81.push({ dist_81[start_81][v_81], v_81 }); //thêm đỉnh vào hàng đợi
45         }
46     }
47 }
48 }
49
50
51 int main() {
52     int n_81, m_81, s_81, t_81;
53     cin >> n_81 >> m_81;
54     for (int i_81 = 0; i_81 < m_81; i_81++) { //lần lượt nhập các cạnh
55         int u_81, v_81, w_81;
56         cin >> u_81 >> v_81 >> w_81;
57         listEdge_81[u_81].push_back(Edge_81(v_81, w_81)); //lưu cách cạnh vào hàng đợi
58     }
59     for (int i_81 = 0; i_81 < n_81; i_81++) {
60         dijkstra_81(i_81 + 1, n_81); //lần lượt tìm đường đi ngắn nhất từ 1 đỉnh tới tất cả các đỉnh
61     }
62     for (int i_81 = 1; i_81 <= n_81; i_81++) {
63         for (int j_81 = 1; j_81 <= n_81; j_81++) {
64             cout << dist_81[i_81][j_81] << " "; //in ma trận khoảng cách nhỏ nhất
65         }
66         cout << "\n";
67     }
68     return 0;
69 }

```

Figure 6 test bài 3

## Input

```

4 9
1 2 9
1 3 7
1 4 2
2 1 1
2 4 5
3 4 6
3 2 2
4 1 5
4 2 8

```

## Correct output

```

0 9 7 2
1 0 8 3
3 2 0 5
5 8 12 0

```

## User output

```

0 9 7 2
1 0 8 3
3 2 0 5
5 8 12 0

```

## Input

```
6 17
1 29
1 37
1 42
1 63
2 11
2 45
2 58
3 46
3 22
3 69
4 15
4 28
4 51
5 21
5 63
6 31
6 52
```

## Correct output

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

## User output

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

## Code:

```
//Mai Minh Hoàng
//20215381
#include <iostream>
#include <vector>
#include <queue>
#include <climits>

#define MAX_N_81 100 //định nghĩa hằng số max

using namespace std;

class Edge_81 { //định nghĩa lớp Edge là cạnh chứa đỉnh tới và trọng số
public:
    int v_81, w_81;
    Edge_81(int v_81, int w_81) {
```

```

    this->v_81 = v_81;//v là đỉnh tới
    this->w_81 = w_81;//w là trọng số
}
};

vector<Edge_81> listEdge_81[MAX_N_81]; //mảng chứa vector, mỗi vector i là một tập các cạnh đi từ đỉnh thứ i
int dist_81[MAX_N_81][MAX_N_81]; //khoảng cách từ đỉnh thứ i tới đỉnh j

void dijkstra_81(int start_81, int n_81) { // hàm dijkstra tìm đường nhỏ nhất
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq_81; //hàng đợi ưu tiên
    chứa cặp khóa và giá trị, khóa nhỏ hơn thì ra trước (khóa là khoảng cách tới nguồn)
    pq_81.push({ 0, start_81 }); //đặt vào hàng đợi đỉnh đầu tiên là nguồn
    for (int i_81 = 1; i_81 <= n_81; i_81++) {
        dist_81[start_81][i_81] = INT_MAX; // khởi tạo tất cả khoảng cách các đỉnh tới nguồn là vô cùng
    }
    dist_81[start_81][start_81] = 0; //khoảng cách từ nguồn tới nguồn là 0

    while (!pq_81.empty()) { //khi hàng đợi còn chưa rỗng
        int u_81 = pq_81.top().second; //lấy ra phần tử ở đầu hàng đợi, u là đỉnh
        int d_81 = pq_81.top().first; //d là khoảng cách từ đỉnh đó tới nguồn
        pq_81.pop();

        if (d_81 > dist_81[start_81][u_81]) continue; //nếu khoảng cách tới nguồn lớn hơn khoảng cách nhỏ nhất
        hiện tại từ đỉnh đó tới nguồn thì bỏ qua

        for (const Edge_81& edge_81 : listEdge_81[u_81]) { //nếu không thì với mỗi cạnh bắt đầu bởi đỉnh u tới đỉnh
        khác
            int v_81 = edge_81.v_81; //lấy ra đỉnh còn lại và trọng số cạnh
            int w_81 = edge_81.w_81;
            if (dist_81[start_81][u_81] + w_81 < dist_81[start_81][v_81]) { //nếu trọng số cộng với khoảng cách từ u tới
            nguồn mà bé hơn khoảng cách nhỏ nhất hiện tại từ v tới nguồn
                dist_81[start_81][v_81] = dist_81[start_81][u_81] + w_81; // thì cập nhật khoảng cách nhỏ nhất từ v tới
            nguồn
                pq_81.push({ dist_81[start_81][v_81], v_81 }); //thêm đỉnh vào hàng đợi
            }
        }
    }
}

int main() {
    int n_81, m_81, s_81, t_81;
    cin >> n_81 >> m_81;
    for (int i_81 = 0; i_81 < m_81; i_81++) { //lần lượt nhập các cạnh
        int u_81, v_81, w_81;
        cin >> u_81 >> v_81 >> w_81;
        listEdge_81[u_81].push_back(Edge_81(v_81, w_81)); //lưu cách cạnh vào hàng đợi
    }
    for (int i_81 = 0; i_81 < n_81; i_81++) {
        dijkstra_81(i_81 + 1, n_81); //lần lượt tìm đường đi ngắn nhất từ 1 đỉnh tới tất cả các đỉnh
    }
    for (int i_81 = 1; i_81 <= n_81; i_81++) {

```

```
for (int j_81 = 1; j_81 <= n_81; j_81++) {  
    cout << dist_81[i_81][j_81] << " "; //in ma trận khoảng cách nhỏ nhất  
}  
cout << "\n";  
}  
return 0;  
}
```