**Bài 1. Đường đi ngắn nhất (Dijkstra.\*)**

**Ý tưởng:**

* **Ban đầu, gán dist[s] = 0, các đỉnh khác dist[i] = ∞.**
* **Mỗi bước, chọn đỉnh u có dist[u] nhỏ nhất chưa được duyệt.**
* **Với mỗi cạnh (u → v, trọng số w):**
  + **Nếu dist[v] > dist[u] + w, thì cập nhật dist[v] = dist[u] + w,  
    và ghi nhớ pre[v] = u (để truy vết đường đi sau này).**
* **Lặp lại cho đến khi xét xong tất cả các đỉnh, hoặc khi đã xác định được dist[t].**

**Code:**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

struct Vertex {

int v;

int w;

};

vector<vector<Vertex>> adjList;

vector<int> dist;

vector<bool> label;

vector<int> path;

vector<int> pre;

const int INF = 1000000000;

void Dijkstra(int s) {

int n = static\_cast<int>(adjList.size()) - 1;

dist.resize(n + 1, INF);

dist[s] = 0;

pre.resize(n+1, -1);

label.resize(n+1, false);

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

int u = -1;

int minDist = INF;

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

if (!label[i] && dist[i] < minDist) {

minDist = dist[i];

u = i;

}

}

if (u == -1) break; // không còn đỉnh có thể cập nhật

label[u] = true;

// --- Cập nhật các đỉnh kề v của u ---

for (auto edge : adjList[u]) {

int v = edge.v;

int w = edge.w;

if (!label[v] && dist[v] > dist[u] + w) {

dist[v] = dist[u] + w;

pre[v] = u; // truy vết

}

}

}

}

// ======== HÀM TRUY VẾT ĐƯỜNG ĐI ========

void TimDuong(int s, int t) {

path.clear();

if (dist[t] == INF) return; // không có đường đi

for (int u = t; u != s; u = pre[u]) {

if (u == -1) return; // tránh vòng lặp vô hạn nếu không có đường đi

path.push\_back(u);

}

path.push\_back(s);

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

}

// ======== HÀM CHÍNH ========

void solve() {

int n, m, s, t;

cin >> n >> m >> s >> t;

adjList.assign(n + 1, {}); // đánh số từ 1 -> n

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

int u, v, w;

cin >> u >> v >> w;

adjList[u].push\_back({ v, w });

}

Dijkstra(s);

TimDuong(s, t);

if (dist[t] == INF) {

cout << "Khong co duong di tu " << s << " den " << t << "\n";

return;

}

cout << path.size() << " " << dist[t] << "\n";

for (int v : path) cout << v << " ";

cout << "\n";

}

int main() {

solve();

return 0;

}

**Bài 2. Đường đi ngắn nhất qua đỉnh trung gian (NganNhatTrungGian.\*)**

**Ý tưởng:**

* **Đường đi sẽ gồm 2 đoạn:**
  1. **Từ s → x**
  2. **Từ x → t**

**Vậy độ dài tổng = dist(s, x) + dist(x, t) và đường đi = (đường s→x) + (đường x→t, bỏ trùng x ở đầu).**

**Code:**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

struct Vertex {

int v;

int w;

};

vector<vector<Vertex>> adjList;

vector<int> dist;

vector<bool> label;

vector<int> path;

vector<int> pre;

const int INF = 1000000000;

void Dijkstra(int s) {

int n = static\_cast<int>(adjList.size()) - 1;

dist.assign(n + 1, INF);

dist[s] = 0;

pre.assign(n + 1, -1);

label.assign(n + 1, false);

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

int u = -1;

int minDist = INF;

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

if (!label[i] && dist[i] < minDist) {

minDist = dist[i];

u = i;

}

}

if (u == -1) break; // không còn đỉnh có thể cập nhật

label[u] = true;

// --- Cập nhật các đỉnh kề v của u ---

for (auto edge : adjList[u]) {

int v = edge.v;

int w = edge.w;

if (!label[v] && dist[v] > dist[u] + w) {

dist[v] = dist[u] + w;

pre[v] = u; // truy vết

}

}

}

}

// ======== HÀM TRUY VẾT ĐƯỜNG ĐI ========

void TimDuong(int s, int t) {

path.clear();

if (dist[t] == INF) return; // không có đường đi

for (int u = t; u != s; u = pre[u]) {

if (u == -1) return; // tránh vòng lặp vô hạn nếu không có đường đi

path.push\_back(u);

}

path.push\_back(s);

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

}

// ======== HÀM CHÍNH ========

void solve() {

int n, m, s, t, x;

cin >> n >> m >> s >> t>>x;

adjList.assign(n + 1, {}); // đánh số từ 1 -> n

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

int u, v, w;

cin >> u >> v >> w;

adjList[u].push\_back({ v, w });

}

Dijkstra(s);

TimDuong(s, x);

if (dist[x] == INF) {

cout << "Khong co duong di tu " << s << " den " << x << "\n";

return;

}

vector<int>path1 = path;

int k = dist[x];

Dijkstra(x);

TimDuong(x, t);

if (dist[t] == INF) {

cout << "Khong co duong di tu " << x << " den " << t << "\n";

return;

}

k += dist[t];

for (int i = 1; i < (int)path.size(); i++) {

path1.push\_back(path[i]);

}

cout << path1.size() << " " << k<< "\n";

for (int v : path1) cout << v << " ";

cout << "\n";

}

int main() {

solve();

return 0;

}

**Bài 3. Đường đi ngắn nhất mọi đỉnh (Floyd.\*)**

**Ý tưởng:**

**Thuật toán Floyd:**

**• Input: G=(V, E)**

**• Output:**

**– Ma trận đường đi ngắn nhất giữa các cặp đỉnh**

**▪ d[i, j]: độ dài đường đi ngắn nhất từ đỉnh i đến đỉnh j**

**– Ma trận ghi nhận đường đi**

**▪ p[i, j]: ghi nhận đỉnh đi trước đỉnh j trong đường đi**

**ngắn nhất từ đỉnh i đến đỉnh j**

**Thuật toán Floyd:**

**• Bước 1: [Khởi tạo]**

**– d[i, j]=a[i, j]**

**– p[i, j] = i;**

**• Bước 2: [Lặp]**

**– Cho k, i, j chạy từ 0 đến n-1**

**Nếu (d[i, j] > d[i, k] + d[k, j]) thì**

**d[i, j] = d[i, k] + d[k, j]**

**p[i, j] = p[k, j]**

**Code:**

#include <iostream>

#include <vector>

using namespace std;

vector<vector<int>> dist;

//vector<int>path;

const int INF = 1000000000;

void Floyd(int n) {

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

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

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

if (dist[i][k] < INF && dist[k][j] < INF) {

if (dist[i][j] > (dist[i][k] + dist[k][j]))

dist[i][j] = dist[i][k] + dist[k][j];

}

}

}

}

}

void solve() {

int n, m;

cin >> n >> m;

dist.assign(n + 1, vector<int>(n + 1, INF));

/\*path.resize(n + 1);\*/

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

dist[i][i] = 0;

}

for (int j = 0; j < m; j++)

{

int u, v, w;

cin >> u >> v >> w;

dist[u][v] = w;

dist[v][u] = w;

}

Floyd(n);

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

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

if (dist[i][j] == INF)

cout << "INF "; else cout << dist[i][j] << " ";

}

cout << "\n";

}

}

int main() {

solve();

return 0;

}

**Bài 4. Dijkstra với hàng đợi ưu tiên (DijkstraPriorityQueue.\*)**

**Ý tưởng:**

 **Khởi tạo ban đầu:**

* dist[s] = 0, các đỉnh khác INF.
* pre[] để lưu **đỉnh trước đó** trong đường đi.
* Đưa đỉnh s vào hàng đợi.

 **Xét từng đỉnh trong hàng đợi:**

* Lấy đỉnh u ra.
* Với mỗi cạnh (u, v) có trọng số w:
  + Nếu dist[v] > dist[u] + w → **cập nhật** dist[v] = dist[u] + w và pre[v] = u.
  + Nếu đỉnh v chưa có trong hàng đợi → thêm v vào để tiếp tục xét.

 **Sau khi kết thúc:**

* dist[t] là **độ dài ngắn nhất** từ s đến t.
* Dựa vào mảng pre[] → **truy vết ngược** để tìm **đường đi**.

**Code:**

#include <iostream>

#include <queue>

#include <vector>

using namespace std;

struct Vertex {

int v;

int w;

};

vector<vector<Vertex>> adjList;

vector<int> dist;

vector<bool> label;

vector<int> path;

vector<int> pre;

queue<int> q;

const int INF = 1000000000;

void Dijkstra(int s) {

int n = static\_cast<int>(adjList.size()) - 1;

dist.resize(n + 1, INF);

dist[s] = 0;

pre.resize(n+1, -1);

label.resize(n+1, false);

q.push(s);

label[s] = true;

while (!q.empty()) {

int u = q.front();

q.pop();

label[u] = false;

for (auto e : adjList[u]) {

int v = e.v;

int w = e.w;

if (dist[v] > dist[u] + w) {

dist[v] = dist[u] + w;

pre[v] = u;

if (!label[v]) {

q.push(v);

label[v] = true;

}

}

}

}

}

void TimDuong(int s, int t) {

path.clear();

if (dist[t] == INF) return; // không có đường đi

for (int u = t; u != s; u = pre[u]) {

if (u == -1) return; // tránh vòng lặp vô hạn nếu không có đường đi

path.push\_back(u);

}

path.push\_back(s);

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

}

void solve() {

int n, m, s, t;

cin >> n >> m >> s >> t;

adjList.assign(n + 1, {}); // đánh số từ 1 -> n

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

int u, v, w;

cin >> u >> v >> w;

adjList[u].push\_back({ v, w });

}

Dijkstra(s);

TimDuong(s, t);

if (dist[t] == INF) {

cout << "Khong co duong di tu " << s << " den " << t << "\n";

return;

}

cout << path.size() << " " << dist[t] << "\n";

for (int v : path) cout << v << " ";

cout << "\n";

}

int main() {

solve();

return 0;

}