Bài 1:

1. Đầu vào:

* Đồ thị có hướng với:
  + n: số đỉnh (1 ≤ n ≤ 1000)
  + m: số cạnh (1 ≤ m ≤ 10^6)
  + x, y: hai đỉnh cần tìm đường đi ngắn nhất từ x đến y
* Mỗi cạnh (u, v) có trọng số w (0 ≤ w ≤ 10^5).

2. Yêu cầu:

Tìm độ dài đường đi ngắn nhất từ x → y và liệt kê các đỉnh trên đường đi đó.

3. Ý tưởng thuật toán Dijkstra:

* Dijkstra là thuật toán tìm đường đi ngắn nhất từ 1 đỉnh nguồn s đến các đỉnh còn lại trong đồ thị có trọng số không âm.
* Dựa trên chiến lược tham lam (Greedy):
  + Luôn chọn đỉnh chưa xét có khoảng cách nhỏ nhất tạm thời.
  + Cập nhật khoảng cách của các đỉnh kề.

4. Cấu trúc dữ liệu cần:

* vector<pair<int,int>> adj[n+1]; → lưu danh sách kề và trọng số.  
  (Ví dụ: adj[u].push\_back({v, w});)
* vector<int> dist(n+1, INF); → lưu độ dài ngắn nhất hiện tại.
* vector<int> parent(n+1, -1); → lưu đường đi (để truy vết ngược).
* priority\_queue<pair<int,int>, vector<pair<int,int>>, greater<pair<int,int>>> pq; → hàng đợi ưu tiên.

Phần code:

#include <iostream>

#include <vector>

#include <queue>

#include <utility>

#include <algorithm>

#include <fstream>

using namespace std;

#define INF 1000000000

int main() {

ifstream fin("dijkstra.in");

ofstream fout("dijkstra.out");

if (!fin.is\_open()) {

cout << "Khong mo duoc file dau vao!" << endl;

return 0;

}

int n, m, s, t;

fin >> n >> m >> s >> t;

vector< vector< pair<int,int> > > adj(n + 1);

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

int u, v, w;

fin >> u >> v >> w;

adj[u].push\_back(make\_pair(v, w));

}

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

vector<int> parent(n + 1, -1);

priority\_queue< pair<int,int>, vector< pair<int,int> >, greater< pair<int,int> > > pq;

dist[s] = 0;

pq.push(make\_pair(0, s));

while (!pq.empty()) {

pair<int,int> top = pq.top();

pq.pop();

int du = top.first;

int u = top.second;

if (du != dist[u]) continue;

for (size\_t i = 0; i < adj[u].size(); i++) {

int v = adj[u][i].first;

int w = adj[u][i].second;

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

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

parent[v] = u;

pq.push(make\_pair(dist[v], v));

}

}

}

if (dist[t] == INF) {

fout << "Khong co duong di" << endl;

return 0;

}

vector<int> path;

int v = t;

while (v != -1) {

path.push\_back(v);

v = parent[v];

}

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

fout << path.size() << " " << dist[t] << endl;

for (size\_t i = 0; i < path.size(); i++)

fout << path[i] << " ";

fout << endl;

fin.close();

fout.close();

return 0;

}

Bài 2:

Ta cần tìm đường đi ngắn nhất từ s → t mà phải đi qua x.  
Khi đó, ta tách bài toán thành hai phần:

Vì đồ thị có hướng và trọng số không âm → dùng Dijkstra là hợp lý.

⚙️ Các bước làm

1. Chạy Dijkstra(s) → tìm khoảng cách từ s đến mọi đỉnh.  
   → lấy dist1[x]: độ dài ngắn nhất từ s → x.
2. Chạy Dijkstra(x) → tìm khoảng cách từ x đến mọi đỉnh.  
   → lấy dist2[t]: độ dài ngắn nhất từ x → t.
3. Tổng độ dài:
4. Truy vết đường đi:
   * Dùng mảng parent1[] từ Dijkstra(s) để lấy đường đi s → x
   * Dùng mảng parent2[] từ Dijkstra(x) để lấy đường đi x → t
   * Ghép 2 đường lại (bỏ trùng đỉnh x ở giữa)

Phần code:

#include <iostream>

#include <vector>

#include <queue>

#include <algorithm>

#include <fstream>

using namespace std;

const int INF = 1000000000;

struct Edge {

int v, w;

};

vector<int> dijkstra(int n, int start, const vector<vector<Edge> > &adj, vector<int> &parent) {

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

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

priority\_queue<pair<int,int>, vector<pair<int,int> >, greater<pair<int,int> > > pq;

dist[start] = 0;

pq.push(make\_pair(0, start));

while (!pq.empty()) {

int du = pq.top().first;

int u = pq.top().second;

pq.pop();

if (du != dist[u]) continue;

for (int i = 0; i < (int)adj[u].size(); i++) {

int v = adj[u][i].v;

int w = adj[u][i].w;

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

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

parent[v] = u;

pq.push(make\_pair(dist[v], v));

}

}

}

return dist;

}

int main() {

ifstream fin("ngannhat.in");

ofstream fout("ngannhat.out");

if (!fin) {

cout << "Khong mo duoc file dau vao!" << endl;

return 0;

}

int n, m, s, t, x;

fin >> n >> m >> s >> t >> x;

vector<vector<Edge> > adj(n + 1);

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

int u, v, w;

fin >> u >> v >> w;

Edge e;

e.v = v;

e.w = w;

adj[u].push\_back(e);

}

vector<int> parent1, parent2;

vector<int> dist1 = dijkstra(n, s, adj, parent1);

vector<int> dist2 = dijkstra(n, x, adj, parent2);

if (dist1[x] == INF || dist2[t] == INF) {

fout << "Khong co duong di" << endl;

return 0;

}

int tong = dist1[x] + dist2[t];

fout << tong << endl;

vector<int> path1;

for (int v = x; v != -1; v = parent1[v]) path1.push\_back(v);

reverse(path1.begin(), path1.end());

vector<int> path2;

for (int v = t; v != -1; v = parent2[v]) path2.push\_back(v);

reverse(path2.begin(), path2.end());

vector<int> path = path1;

for (int i = 1; i < (int)path2.size(); i++)

path.push\_back(path2[i]);

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

fout << path[i] << " ";

fout << endl;

fin.close();

fout.close();

return 0;

}

Bài 3:

Ý tưởng thuật toán Floyd-Warshall:

1. Khởi tạo bảng khoảng cách dist[i][j]:
   * Nếu i == j → dist[i][j] = 0
   * Nếu có cạnh (i, j) → dist[i][j] = w
   * Ngược lại → dist[i][j] = INF (vô cực, tức là chưa có đường đi)
2. Cập nhật dần dần qua từng đỉnh trung gian k:

Ý tưởng cốt lõi là:

Nếu đi từ i → j mà qua một đỉnh trung gian k thì tổng quãng đường có thể ngắn hơn.

Ta xét:

dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j])

Nghĩa là:

* + Nếu đi thẳng i → j tốn dist[i][j]
  + Nhưng nếu đi vòng i → k → j mà ngắn hơn thì cập nhật lại giá trị mới.

1. Lặp cho tất cả các đỉnh trung gian k:

for (k = 1 → n)

for (i = 1 → n)

for (j = 1 → n)

dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);

Sau 3 vòng lặp này, dist[i][j] chính là độ dài ngắn nhất từ i đến j.

Phần code:

#include <iostream>

#include <fstream>

#include <vector>

#include <algorithm>

using namespace std;

const int INF = 1000000000;

int main() {

ifstream fin("floyd.in");

ofstream fout("floyd.out");

if (!fin) {

cout << "Khong mo duoc file dau vao!" << endl;

return 0;

}

int n, m;

fin >> n >> m;

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

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

dist[i][i] = 0;

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

int u, v, w;

fin >> u >> v >> w;

if (w < dist[u][v]) {

dist[u][v] = w;

dist[v][u] = w; // d? th? vô hu?ng

}

}

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

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

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

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

dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);

}

}

}

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

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

if (dist[i][j] == INF) fout << "0 ";

else fout << dist[i][j] << " ";

}

fout << endl;

}

fin.close();

fout.close();

return 0;

}

Bài 4:

Ý tưởng:

Bước 1. Biểu diễn đồ thị

* Dùng danh sách kề để lưu cạnh:
* vector<pair<int, int>> adj[n + 1];

Mỗi phần tử là (đỉnh\_kề, trọng\_số).

Bước 2. Khởi tạo

* Mảng dist[i] = INF (vô cực)
* Mảng pre[i] để lưu đỉnh trước đó (phục hồi đường đi)
* dist[s] = 0
* Dùng hàng đợi ưu tiên:
* priority\_queue<pair<int,int>, vector<pair<int,int>>, greater<pair<int,int>>> pq;

Mỗi phần tử trong pq là (khoảng cách hiện tại, đỉnh)  
→ đảm bảo đỉnh gần nhất được lấy ra trước.

Bước 3. Vòng lặp chính

Trong khi hàng đợi chưa rỗng:

1. Lấy đỉnh u có khoảng cách nhỏ nhất.
2. Nếu dist[u] đã tối ưu → bỏ qua.
3. Với mỗi cạnh (u, v, w):
   * Nếu dist[v] > dist[u] + w thì:
     + Cập nhật dist[v] = dist[u] + w
     + Ghi nhớ pre[v] = u
     + Đưa (dist[v], v) vào hàng đợi.

Bước 4. Phục hồi đường đi

Sau khi chạy xong:

* Nếu dist[t] == INF → không có đường đi.
* Ngược lại:
* vector<int> path;
* for (int cur = t; cur != -1; cur = pre[cur])
* path.push\_back(cur);

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

Phần code:

#include <iostream>

#include <vector>

#include <queue>

#include <algorithm>

#include <fstream>

using namespace std;

const int INF = 1000000000;

struct Edge {

int v, w;

};

vector<int> dijkstra(int n, int start, vector<vector<Edge> > &adj, vector<int> &parent) {

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

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

// priority\_queue <pair<khoang cach, dinh>, ... >

priority\_queue<pair<int, int>, vector<pair<int, int> >, greater<pair<int, int> > > pq;

dist[start] = 0;

pq.push(make\_pair(0, start));

while (!pq.empty()) {

int du = pq.top().first;

int u = pq.top().second;

pq.pop();

if (du != dist[u]) continue;

for (int i = 0; i < (int)adj[u].size(); i++) {

int v = adj[u][i].v;

int w = adj[u][i].w;

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

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

parent[v] = u;

pq.push(make\_pair(dist[v], v));

}

}

}

return dist;

}

int main() {

ifstream fin("priorityqueue.in");

ofstream fout("priorityqueue.out");

if (!fin.is\_open()) {

cout << "Khong mo duoc file dau vao!\n";

return 0;

}

int n, m, s, t, x;

fin >> n >> m >> s >> t >> x;

vector<vector<Edge> > adj(n + 1);

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

int u, v, w;

fin >> u >> v >> w;

Edge e;

e.v = v;

e.w = w;

adj[u].push\_back(e);

}

vector<int> parentS, parentX;

vector<int> distS = dijkstra(n, s, adj, parentS);

vector<int> distX = dijkstra(n, x, adj, parentX);

if (distS[x] == INF || distX[t] == INF) {

fout << "Khong co duong di" << endl;

fin.close();

fout.close();

return 0;

}

int total = distS[x] + distX[t];

fout << total << endl;

vector<int> path1;

for (int v = x; v != -1; v = parentS[v])

path1.push\_back(v);

reverse(path1.begin(), path1.end());

vector<int> path2;

for (int v = t; v != -1; v = parentX[v])

path2.push\_back(v);

reverse(path2.begin(), path2.end());

vector<int> path = path1;

for (int i = 1; i < (int)path2.size(); i++)

path.push\_back(path2[i]);

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

fout << path[i] << " ";

fout << endl;

fin.close();

fout.close();

return 0;

}