**TH5:**

**Bài 1:**

Main idea: Tổ chức kiểu dữ liệu theo danh sách kề, có hàm bool kiểm tra đã thăm và hàm trace để dò ngược lại

Cài đặt hàm Dijkstra duyệt theo công thức **dist[v] = dist[u] + weight[u][v]** và cập nhập (không theo priority queue) và gán trace để đưa vào main và sử dụng hàm reverse để dò ngược lại và in kết quả

Code:

#include <iostream>

#include <queue>

#include <climits>

#include <stack>

#include <fstream>

#include <algorithm>

using *namespace* std;

#define FI "test.inp"

#define FO "test.out"

const *int* MAXN = 1005;

vector<pair<*int*, *int*>> adj[MAXN];

*int* dist[MAXN], trace[MAXN];

*bool* visited[MAXN];

*void* dijkstra(*int* *start*, *int* *n*)

{

    for (*int* i = 1; i <= *n*; i++)

    {

        dist[i] = INT\_MAX;

        visited[i] = false;

        trace[i] = -1;

    }

    dist[*start*]=0;

    for (*int* i = 1; i <= *n*; i++)

    {

*int* u = -1;

*int* minDist = INT\_MAX;

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

        {

            if (!visited[j] && dist[j] < minDist)

            {

                minDist = dist[j];

                u = j;

            }

        }

        if (u == -1)break;

        visited[u] = true;

        for (*auto* &[v, w] : adj[u])

        {

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

            {

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

                trace[v] = u;

            }

        }

    }

}

// cài bằng priority\_queue (nhanh hơn cho đồ thị lớn, nhưng không dò được phần tử âm)

// void dijkstra(int x, int n) // x là start

// {

//     for (int i = 1; i <= n; i++)

//     {

//         dist[i] = INT\_MAX;

//         visited[i] = false;

//         trace[i] = -1;

//     }

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

//     dist[x] = 0;

//     pq.push({0, x});

//     while (!pq.empty())

//     {

//         int u = pq.top().second;

//         pq.pop();

//         if (visited[u])

//             continue;

//         {

//             visited[u] = true;

//         }

//         for (auto &[v, w] : adj[u])

//         {

//             if (dist[u] + w < dist[v])

//             {

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

//                 trace[v] = u;

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

//             }

//         }

//     }

// }

*int* n, m, x, y;

*int* u, v, w;

*void* readInput(ifstream &*fi*)

{

*fi* >> n >> m >> x >> y;

    for (*int* i = 0; i < m; i++)

    {

*fi* >> u >> v >> w;

        adj[u].push\_back({v, w});

        adj[v].push\_back({u,w});

    }

}

*int* main()

{

    ifstream fi(FI);

    ofstream fo(FO);

    readInput(fi);

    dijkstra(x, n);

    vector<*int*> path;

*int* HTai = y;

    while (HTai != -1)

    {

        path.push\_back(HTai);

        HTai = trace[HTai];

    }

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

    fo << path.size() << "  " << dist[y] << endl;

    for (*auto* node : path)

    {

        fo << node << "  ";

    }

    fo << endl;

    return 0;

}

Input: Output:

6 7 1 4 6 15

1 2 1 1 2 3 6 5 4

1 6 10

2 3 2

3 4 20

3 6 3

6 5 4

5 4 5

**Bài 2:**

Main idea: tương tự như bài 1, nhưng lần này làm theo priority queue để queue khi đấy qua hàm great<pair<int,int>> sẽ tự động sắp xếp lại danh sách các đỉnh có khoảng cách ngắn nhất rồi đưa vào xử lý, và sau đó chỉnh sửa lại hàm nhập để có them giá trị x và chạy dijkstra 2 lần rồi tại main dùng insert để ghép mảng lại rồi in kết quả

Code:

#include <iostream>

#include <queue>

#include <climits>

#include <stack>

#include <fstream>

#include <algorithm>

using *namespace* std;

#define FI "test1.inp"

#define FO "test1.out"

const *int* MAXN = 1005;

vector<pair<*int*, *int*>> adj[MAXN];

*int* dist1[MAXN], dist2[MAXN], trace1[MAXN], trace2[MAXN];

*bool* visited[MAXN];

*void* dijkstra(*int* *start*, *int* *n*, *int* *dist*[], *int* *trace*[])

{

    for (*int* i = 1; i <= *n*; i++)

    {

*dist*[i] = INT\_MAX;

        visited[i] = false;

*trace*[i] = -1;

    }

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

*dist*[*start*] = 0;

    pq.push({0, *start*});

    while (!pq.empty())

    {

*int* u = pq.top().second;

        pq.pop();

        if (visited[u])

            continue;

        {

            visited[u] = true;

        }

        for (*auto* &[v, w] : adj[u])

        {

            if (*dist*[u] + w < *dist*[v])

            {

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

*trace*[v] = u;

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

            }

        }

    }

}

*int* n, m, s, t, x;

*int* u, v, w;

*void* readInput(ifstream &*fi*)

{

*fi* >> n >> m >> s >> t >> x;

    for (*int* i = 0; i < m; i++)

    {

*fi* >> u >> v >> w;

        adj[u].push\_back({v, w});

    }

}

vector<*int*> tracePath(*int* *start*, *int* *end*, *int* *trace*[])

{

    vector<*int*> path;

*int* current = *end*;

    while (current != -1)

    {

        path.push\_back(current);

        current = *trace*[current];

    }

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

    return path;

}

 vector<*int*>path;

*int* main()

{

    ifstream fi(FI);

    ofstream fo(FO);

    readInput(fi);

    dijkstra(s, n, dist1, trace1);

    dijkstra(x,n,dist2,trace2);

    vector<*int*> path1 = tracePath(s, x, trace1); // từ x đến z

    vector<*int*> path2 = tracePath(x, t, trace2); // từ z đến y

    path1.pop\_back(); // xóa z khỏi path1 để tránh lặp

    vector<*int*> fullPath = path1;

    fullPath.insert(fullPath.end(), path2.begin(), path2.end());

    fo << fullPath.size() << "  " << dist1[x]+dist2[t] << endl;

    for (*auto* node : fullPath)

    {

        fo << node << "  ";

    }

    fo << endl;

    return 0;

}

Input:

6 7 1 4 3

1 2 1

1 6 10

2 3 2

3 4 20

3 6 3

6 5 4

5 4 5

Output:

6 15

1 2 3 6 5 4

Bài 3:

Main idea: Cài đặt thuật toán Floyd-Warshall trên ý tưởng duyệt hết các phần tử trong ma trận kề và theo công thức lặp **dist[i][j] = min(dist[i][j],dist[i][k] + dist[k][j]);** và sửa lại hàm readinput để đọc từ danh sách kề sang ma trận kề

Code:

#include <iostream>

#include <algorithm>

#include <queue>

#include <climits>

#include <fstream>

#include <vector>

using *namespace* std;

#define FI "test2.inp"

#define FO "test2.out"

vector<vector<*int*>> adj;

*int* n, m;

*int* u, v, w;

*void* Floyd(vector<vector<*int*>> &*adj*)

{

*int* v =*adj*.size();

    for (*int* k = 0; k < v; k++)

    {

        for (*int* i = 0; i <v; i++)

        {

            for (*int* j = 0; j <v; j++)

            {

                if (*adj*[i][k] != INT\_MAX && *adj*[k][j] != INT\_MAX)

                {

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

                }

            }

        }

    }

}

*void* readInput(ifstream &*fi*)

{

*fi* >> n >> m;

    adj.assign(n + 1, vector<*int*>(n + 1, INT\_MAX));

    for (*int* i = 1; i <= n; i++)

    {

        adj[i][i] = 0;

    }

    for (*int* i = 0; i < m; i++)

    {

*fi* >> u >> v >> w;

        adj[u][v] = w;

        adj[v][u] = w;

    }

}

*int* main()

{

    ifstream fi(FI);

    ofstream fo{FO};

    readInput(fi);

    Floyd(adj);

    for (*int* i = 1; i <= n; i++)

    {

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

        {

            if (adj[i][j] == INT\_MAX)

            {

                fo << "INF ";

            }

            else

            {

                fo << adj[i][j] << " ";

            }

        }

        fo << endl;

    }

    return 0;

}

Input:

5 6

1 2 5

1 4 9

1 5 1

2 3 2

3 4 7

4 5 2

Output:

0 5 7 3 1

5 0 2 8 6

7 2 0 7 8

3 8 7 0 2

1 6 8 2 0

**Bài 4:**

Main idea: Tương tụ như bài 1, nhưng lần này cài theo **priority\_queue<pair<*int*, *int*>, vector<pair<*int*, *int*>>, greater<pair<*int*, *int*>>> pq;** (tương tự như hàm Dijkstra ở bài 2)

Code:

#include <iostream>

#include <queue>

#include <climits>

#include <stack>

#include <fstream>

#include<algorithm>

using *namespace* std;

#define FI "test3.inp"

#define FO "test3.out"

const *int* MAXN = 1005;

vector<pair<*int*, *int*>> adj[MAXN];

*int* dist[MAXN], trace[MAXN];

*bool* visited[MAXN];

*void* dijkstra(*int* *x*, *int* *n*) // x là start

{

    for (*int* i = 1; i <= *n*; i++)

    {

        dist[i] = INT\_MAX;

        visited[i] = false;

        trace[i] = -1;

    }

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

    dist[*x*] = 0;

    pq.push({0, *x*});

    while (!pq.empty())

    {

*int* u = pq.top().second;

        pq.pop();

        if (visited[u])

            continue;

        {

            visited[u] = true;

        }

        for (*auto* &[v, w] : adj[u])

        {

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

            {

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

                trace[v] = u;

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

            }

        }

    }

}

*int* n, m, x, y;

*int* u, v, w;

*void* readInput(ifstream &*fi*)

{

*fi* >> n>>m>> x>> y;

    for (*int* i = 0; i < m; i++)

    {

*fi* >> u>> v>> w;

        adj[u].push\_back({v, w});

    }

}

*int* main()

{

    ifstream fi(FI);

    ofstream fo(FO);

    readInput(fi);

    dijkstra(x,n);

    vector<*int*>path;

*int* HTai=y;

    while (HTai !=-1)

    {

        path.push\_back(HTai);

        HTai=trace[HTai];

    }

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

    fo<<path.size()<<"  "<<dist[y]<<endl;

    for (*auto* node:path)

    {

        fo <<node<<"  ";

    }

    fo<<endl;

    return 0;

}

input và output y hệt như bài 1.