CS204(2025)

Graph-II

Instructions for Implementation

- Write the program in C or C++ using standard input/output.
- Follow the input/output format strictly.
- Ensure your code: Handles edge cases correctly.
- Do not use STL containers like vector, set, or map in C++. Use raw arrays unless explicitly allowed.
- You are encouraged to write clean, modular, and well-documented code.

Q1. Shortest Delivery Routes in a City

Problem Statement:

A logistics company operates in a city having E number of the roads and V number of the intersection point. These intersection points are acting like both the delivery point as well as the warehouse. Each **road** between intersections has a **travel time**.

You are hired as a software engineer to design a system that helps the company find the **shortest delivery routes** from a given warehouse to all other delivery points in the city.

Write a **C/C++ program** to compute the shortest travel time from the warehouse (source vertex) to every other intersection using **Dijkstra's Algorithm**.

Input Format:

- The first line contains two integers `V` and `E` — the number of intersections and roads.
- The next `E` lines each contain three integers `u v w`, representing a road between intersections `u` and `v` with travel time `w`.
- The last line contains an integer `S` — the intersection number where the warehouse (source) is located.

Output Format:

Print the **minimum travel time** from the warehouse to each intersection in the following format:

Intersection_point Time

NOTE: Print the intersection point in ascending order and the values should be separated by space**.**

Sample Test Case:

Input:

```
5 6
0 1 2
0 2 4
1 2 1
1 3 7
2 4 3
3 4 1
0
```

Output:

```
0 0
1 2
2 3
3 9
4 6
```

Q2. Currency Exchange Rate Analyzer
Problem Statement:
You are working as a data scientist for a **global finance company** that analyzes **currency exchange rates** between different countries.
Each currency is represented as a **vertex** in a graph, and each **directed edge** from currency `u` to `v` with weight `w` represents the **exchange rate cost** (which can even be negative due to conversion fees, commissions, or market fluctuations).
Your task is to design a **C/C++ program** that finds the **minimum conversion cost** from a given base currency to all other currencies.

Input Format:

- The first line contains two integers `V` and `E` — the number of currencies and exchange rates.
- The next `E` lines each contain three integers `u v w`, representing a directed edge from currency `u` to currency `v` with exchange cost `w`.
- The last line contains an integer `S` — the source currency (base).

Output Format:
Print the **minimum conversion cost** from the base currency to each other currency in the following format:
Target_Currency Cost

NOTE : Print the Target_Currency in **ascending order** and the values should be separated by space**.**
NOTE : If the system detects a **negative weight cycle**, print : **nwc**

Sample Test Case:
Input:
5 8
0 1 -1
0 2 4
1 2 3
1 3 2
1 4 2
3 2 5

```
3 1 1
4 3 -3
0
```
Output:
```
0 0
1 -1
2 2
3 -2
4 1
```

Q3. City Traffic Navigation System

Problem Statement:

You are working for a **smart city transportation department** that aims to design an intelligent **traffic navigation system**.

The city's has V number of **crossing points** and E number of **roads**. There is travel time associated between two crossing points.

To optimize route planning for emergency vehicles and public transport, you need to compute the **shortest travel time between every pair of crossing points** in the city.

Write a **C/C++ program** that calculates the shortest path distances between all pairs of crossing using the **adjacency matrix** representation of the graph.

Input Format:

- The first line contains an integer `V` — the number of crossing points.
- The next `V` lines each contain `V` integers representing the **adjacency matrix** of the graph, where:
  - `matrix[i][j]` = travel time from intersection `i` to `j`
  - Use a large value (e.g., `99999`) to represent **no direct road** between intersections.

Output Format:

Print the **shortest travel time matrix**, where each entry `(i, j)` represents the **minimum travel time** from intersection `i` to intersection `j`.

If there is no path between two intersections, print `INF` for that entry.

NOTE: The values in the output should be separated by space**.**

Sample Test Case:

Input:

```
4
0 5 99999 10
99999 0 3 99999
99999 99999 0 1
99999 99999 99999 0
```

Output:

```
0 5 8 9
INF 0 3 4
```

INF INF 0 1
INF INF INF 0

Q4. Network Latency Optimization Between Global Data Centers
Problem Statement:
You are a **network engineer** for a multinational cloud company that operates several **data centers** across the globe.
Each data center is connected to others through high-speed fiber links, but the **latency** (in milliseconds) between centers may vary depending on routing paths and network load.
Your goal is to determine the **minimum network latency** between every pair of data centers and also identify the **optimal intermediate data centers** used along those paths.
Write a **C/C++ program** to:

1. Compute the **shortest latency** between all pairs of data centers and display the **path matrix**, indicating the **intermediate data centers** used in each shortest path.

Input Format:

- The first line contains an integer `V` — the number of data centers.
- The next `V` lines each contain `V` integers representing the **latency matrix**, where:
  - `matrix[i][j]` = latency (in milliseconds) from data center `i` to `j`
  - Use a large value (e.g., `99999`) to represent **no direct connection** between two centers.

Output Format:

1. Print the **Path Matrix**, showing intermediate data centers used in each shortest path.

NOTE: The values in the output should be separated by space**.**

Sample Test Case:
Input:
4
0 5 99999 10
99999 0 3 99999

```
99999 99999 0 1
99999 99999 99999 0
```

Output:

```
- - 1 1
- - - 2
- - - -
- - - -
```

Here,

- `matrix[i][j] = k` means the **path from i → j** passes through data center **k** as an intermediate node.
- A dash (`-`) indicates a **direct connection** or **no available path**.

Q5. Building a Cost-Effective Fiber Network

Problem Statement:

You are working as a **network infrastructure engineer** for a telecom company that is setting up a **fiber-optic network** to connect several **office buildings** in a new smart business district.

Each building represents a **node**, and the **cost of laying fiber cables** between two buildings is given (in lakhs of rupees).

Since the goal is to **connect all buildings with minimum total cost** while ensuring that **every building is reachable**, you decide to use **Prim's Algorithm** to design the **Minimum Spanning Tree (MST)** of the network.

Your task is to write a **C/C++ program** that takes the cost adjacency matrix of the graph and outputs:

1. The **edges** included in the MST.
2. The **total minimum cost** of connecting all buildings.

Assume that the graph is **connected, undirected, and weighted**, ensuring a **unique MST** when no two edges have the same weight.

Input Format:

- The first line contains an integer `V` — the number of buildings.
- The next `V` lines contain `V` integers, representing the **adjacency matrix** of cable-laying costs between buildings.
  - `matrix[i][j]` = cost to connect building `i` and building `j`.
  - Use `0` to represent **no self-connection** (i.e., `i == j`).

Output Format:

- Print the edges that form the **Minimum Spanning Tree**, along with their costs.
- Finally, print the **total minimum cost (X)**.

Format:

u v w

X

Note:

1. Assume all edge weights are distinct, ensuring a unique Minimum Spanning Tree (MST).

2. Always start Prim's Algorithm from **building 0** (vertex 0).
3. After computing the MST, **print all MST edges in ascending order of vertices** — i.e., sort edges by the smaller vertex first, and if equal, by the larger vertex.

Sample Test Case:

Input:

```
5
0 2 0 6 0
2 0 3 8 5
0 3 0 0 7
6 8 0 0 9
0 5 7 9 0
```

Output:

```
0 1 2
1 2 3
1 4 5
0 3 6
16
```

Explanation:

- The algorithm connects all five buildings (`0` to `4`) using the least expensive set of fiber links.
- Since all edge weights are distinct, the resulting **MST is unique**.
- The **total minimum cost** of connecting all buildings is **16 lakhs**.

Q6. Designing a Power Distribution Grid

Problem Statement:

You are an **electrical engineer** working for a renewable energy company that is planning to set up a **power distribution network** connecting several **solar power stations** across a rural region.

Each power station is represented as a **node**, and the **cost of laying transmission lines** between two stations (in crores of rupees) is represented as an **edge weight**.

Your objective is to **connect all the power stations** such that:

- Every station is connected directly or indirectly,
- The **total construction cost** is minimized, and
- No cycles are formed (ensuring efficiency and no redundant connections).

To achieve this, you decide to use **Kruskal's Algorithm** — a **greedy approach** for finding the **Minimum Spanning Tree (MST)** of the network.

Write a **C/C++ program** to implement Kruskal's Algorithm to determine:

1. The **edges** included in the MST, and
2. The **total minimum cost** of building the power grid.

Assume all edge weights are **distinct**, guaranteeing a **unique MST**.

Input Format:

- The first line contains two integers `V` and `E` — the number of power stations (vertices) and possible connections (edges).
- The next `E` lines each contain three integers `u v w`, representing:
  - A possible connection between stations `u` and `v`,
  - And its associated cost `w` (in crores).

Output Format:

Print the **edges** that form the MST and the **total minimum cost(X)** of the network.

Format:

u v w

X

Note:

1. Assume all edge weights are **distinct**, ensuring a **unique MST**.
2. While processing edges, **sort them first by weight** in ascending order.

3. If multiple edges have the same weight (hypothetically), break ties by the **lower vertex index first**, then by the **next vertex index**.
4. After constructing the MST, **print all edges in ascending order of (u, v)** — i.e., sort MST edges first by the smaller vertex number, then by the larger one.
5. The program output must exactly match in both **edges** and **order** for uniform evaluation.

Sample Test Case:

Input:

```
6 9
0 1 4
0 2 4
1 2 2
1 0 4
2 3 3
2 5 2
2 4 4
3 4 3
5 4 3
```

Output:

```
1 2 2
2 5 2
2 3 3
3 4 3
0 1 4
14
```

Explanation:

- The algorithm selects the **lowest-cost connections** first, ensuring that no cycles are created.
- All **six stations** are connected with a **total cost of 14 crores**.
- Since all edge weights are unique, the resulting **MST is unique**.

Q7. Find the City With the Smallest Number of Reachable Cities Within Threshold

Problem Statement:

There are n cities numbered from 0 to n-1. Given the array edges where edges[i] = [fromi, toi, weighti] represents a bidirectional and weighted edge between cities fromi and toi, and given the integer distanceThreshold. Return the city with the smallest number of cities that are reachable through some path and whose distance is at most distanceThreshold. If multiple such cities exist, return the city with the greatest number.

Input Format:

Three integers n, m, and distanceThreshold.
Followed by m lines where each line contains three integers fromi, toi, and weighti representing an edge.

Output Format:

Print one integer — the city satisfying the conditions.

Test Cases:

Input:

```
4 4 4
 0 1 3
 1 2 1
 2 3 4
 0 3 7
```

Output:

```
3
```

Input:

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

Output:

```
3
```

Q8. Swim in Rising Water

Problem Statement:

You are given an n x n integer matrix grid where each value grid[i][j] represents the elevation at that point (i, j).

It starts raining, and water gradually rises over time. At time t, the water level is t, meaning any cell with elevation less than or equal to t is submerged or reachable.

You can swim from a square to another 4-directionally adjacent square if and only if the elevation of both squares individually are at most t. You can swim infinite distances in zero time. You must stay within the boundaries of the grid during your swim.

Return the minimum time until you can reach the bottom right square (n - 1, n - 1) if you start at the top left square (0, 0).

Input Format:

An integer n — the size of the matrix.

Followed by n lines of space-separated integers representing the elevation grid.

Output Format:

Print one integer — the minimum time required to reach the bottom-right cell.

Test Cases:

Input:

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

Output:

```
4
```

Input:

```
2
0 1
2 3
```

Output:

```
3
```