Symbiosis Institute of Technology

Department of Computer Science and Engineering

Academic Year 2025-26

Design Analysis of Algorithm– Lab

Batch 2023-27 - Sem V

Lab Assignment No:- 4	
Name of Student	Deepti Pal
PRN No.	23070122081
Batch	2023-27
Class	CSE A3
Academic Year & Semester	2025-26 TY, 5th Semester
Date of Submission	26 August 2025
Title of Assignment:	Assume that you are working as a software engineer for a logistics startup deploying delivery drones across Pune, India. The city's delivery hubs correspond to key localities, connected by roads with varying travel times due to traffic conditions. Your goal is to build a routing module that finds the fastest route between any two delivery hubs in Pune, minimizing delivery time. The input will be a list of delivery hubs in Pune and roads connecting them with estimated travel times in minutes. Given a **starting hub** and a **destination hub**, the program should compute: The shortest travel time between the two hubs. The exact route the drone should take (sequence of hubs)

Example:

hubs = ['Shivaji Nagar', 'FC Road', 'Kothrud', 'Viman Nagar', 'Hadapsar']

roads = [

    ('Shivaji Nagar', 'FC Road', 10),

    ('Shivaji Nagar', 'Kothrud', 15),

    ('FC Road', 'Kothrud', 5),

    ('FC Road', 'Viman Nagar', 20),

    ('Kothrud', 'Hadapsar', 30),

    ('Viman Nagar', 'Hadapsar', 10)

]

start = 'Shivaji Nagar'

end = 'Hadapsar'

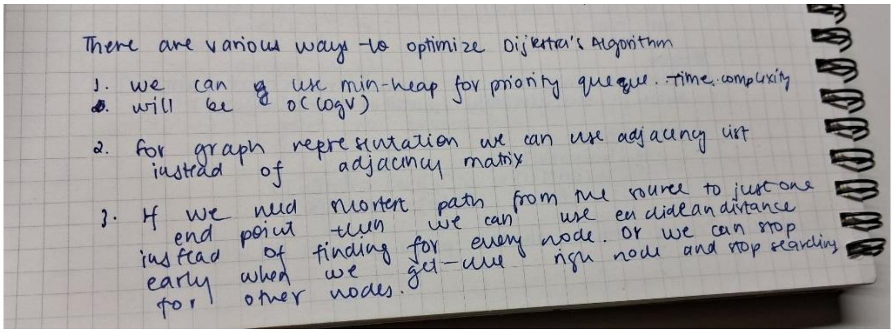
Shortest travel time: 40 minutes

Path: Shivaji Nagar → FC Road → Viman Nagar → Hadapsar

1.

Theory: (Handwritten)

1. How we can optimize Dijkstra's algorithm?



There are various ways to optimize Dijkstra's Algorithm

1. we can use min-heap for priority queque. Time complexity will be O(logV)

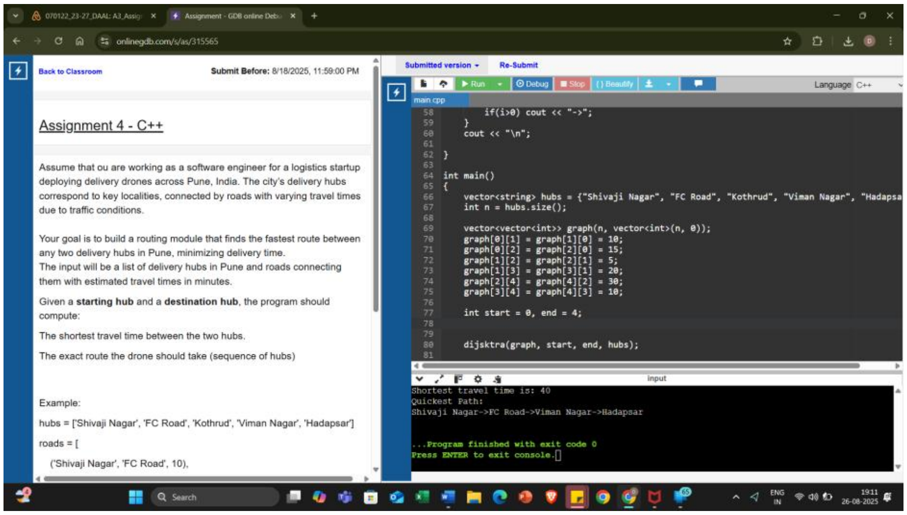
2. for graph representation we can use adjacency list instead of adjacency matrix

3. If we need shortest path from the source to just one end point then we can use euclidean distance instead of finding for every node. Or we can stop early when we get the righ node and stop searching for other nodes.

Source code

```
#include<iostream>
#include<climits>
#include<vector>
using namespace std;

//dijsktra algorithm

int minDist(vector<int> &tim, vector<bool>
&visited, int n) {
      int minValue = INT_MAX;
      int minIndex = -1;

      for(int i=0; i<n; i++) {
            if(!visited[i] && tim[i] < minValue) {
                  minValue = tim[i];
                  minIndex = i;
            }
      }
      return minIndex;
}

void dijsktra(vector<vector<int>> &graph, int src,
int dest, vector<string> &hubs) {
      int n = graph.size();
      vector<int> tim(n, INT_MAX);
      vector<bool> visited(n, false);
      vector<int> parent(n, -1);
      vector<int> path;

      //initialize
      tim[src] = 0;

      for(int count =0; count<n-1; count ++) {
            int u = minDist(tim, visited, n);
            if(u==-1) {
                  break;
            }
            visited[u] = true;

            for(int v=0; v<n; v++) {
                  if(graph[u][v] != 0 && !visited[v]
&& tim[u] + graph[u][v]< tim[v]) {
                        tim[v] = tim[u] +
graph[u][v];
                        parent[v] = u;
                  }
            }
      }
      if (tim[dest] == INT_MAX) {
        cout << "No path exists between " <<
hubs[src] << " and " << hubs[dest] << "\n";
        return;
    }

      for(int v=dest; v!= -1; v = parent[v]) {
            path.push_back(v);
      }

      cout << "Shortest travel time is: " << tim[dest] << endl;
      cout << "Quickest Path: " << endl;
      for(int i=path.size()-1; i>-1; i--) {
            cout << hubs[path[i]];
            if(i>0) cout << "->";
      }
      cout << "\n";

}

int main()
{
      vector<string> hubs = {"Shivaji Nagar", "FC Road", "Kothrud", "Viman Nagar", "Hadapsar"};
      int n = hubs.size();

      vector<vector<int>> graph(n, vector<int>(n, 0));
      graph[0][1] = graph[1][0] = 10;
      graph[0][2] = graph[2][0] = 15;
      graph[1][2] = graph[2][1] = 5;
      graph[1][3] = graph[3][1] = 20;
      graph[2][4] = graph[4][2] = 30;
      graph[3][4] = graph[4][3] = 10;

      int start = 0, end = 4;


      dijsktra(graph, start, end, hubs);

}
```

Output Screenshots (if applicable)



Submit Before: 8/18/2025, 11:59:00 PM

Assignment 4 - C++

Assume that ou are working as a software engineer for a logistics startup deploying delivery drones across Pune, India. The city's delivery hubs correspond to key localities, connected by roads with varying travel times due to traffic conditions.

Your goal is to build a routing module that finds the fastest route between any two delivery hubs in Pune, minimizing delivery time.
The input will be a list of delivery hubs in Pune and roads connecting them with estimated travel times in minutes.

Given a **starting hub** and a **destination hub**, the program should compute:

The shortest travel time between the two hubs.

The exact route the drone should take (sequence of hubs)

Example:

hubs = ['Shivaji Nagar', 'FC Road', 'Kothrud', 'Viman Nagar', 'Hadapsar']

roads = [

('Shivaji Nagar', 'FC Road', 10),

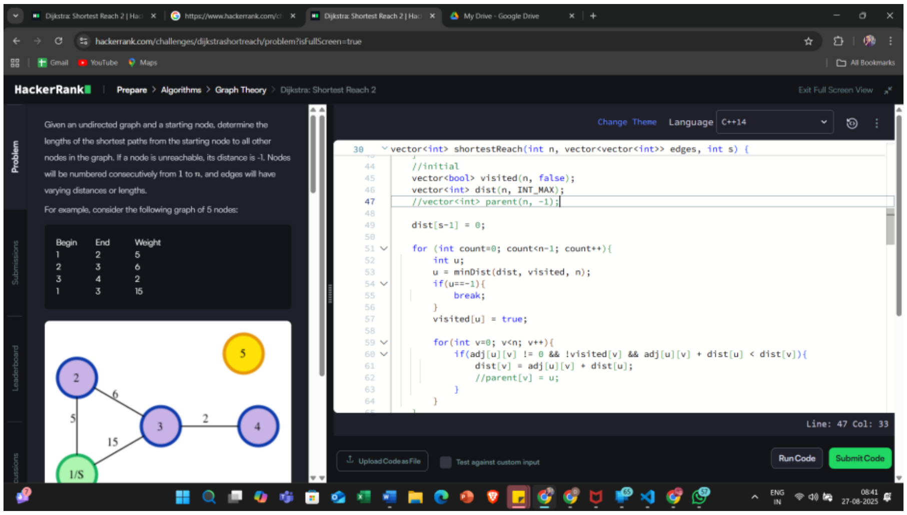
```
        if(i>0) cout << "->";
    }
    cout << "\n";

}

int main()
{
    vector<string> hubs = {"Shivaji Nagar", "FC Road", "Kothrud", "Viman Nagar", "Hadapsa
    int n = hubs.size();

    vector<vector<int>> graph(n, vector<int>(n, 0));
    graph[0][1] = graph[1][0] = 10;
    graph[0][2] = graph[2][0] = 15;
    graph[1][2] = graph[2][1] = 5;
    graph[1][3] = graph[3][1] = 20;
    graph[2][4] = graph[4][2] = 30;
    graph[3][4] = graph[4][3] = 10;

    int start = 0, end = 4;


    dijsktra(graph, start, end, hubs);
```

```
Shortest travel time is: 40
Quickest Path:
Shivaji Nagar->FC Road->Viman Nagar->Hadapsar


...Program finished with exit code 0
Press ENTER to exit console.
```

Problems Solved from Hacker Rank

1. https://www.hackerrank.com/challenges/dijkstrashortreach/problem?

Solution Screenshot:

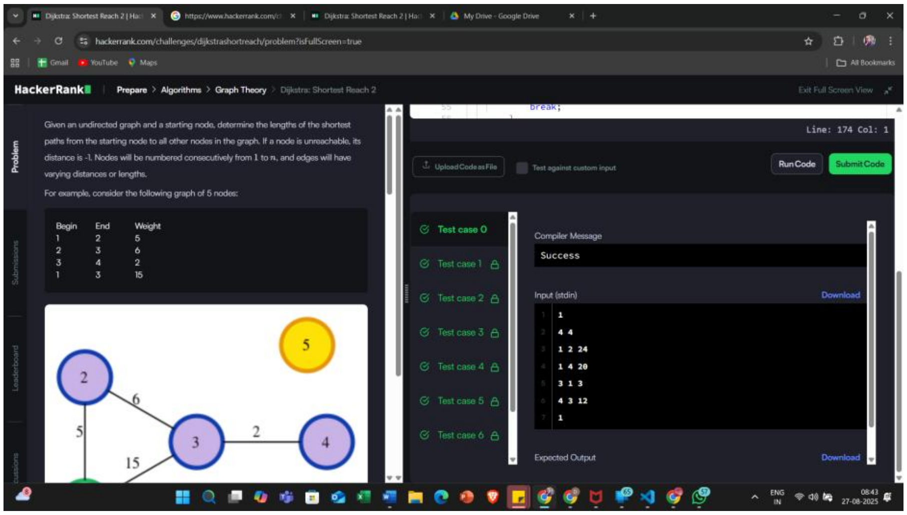


Given an undirected graph and a starting node, determine the lengths of the shortest paths from the starting node to all other nodes in the graph. If a node is unreachable, its distance is -1. Nodes will be numbered consecutively from 1 to n, and edges will have varying distances or lengths.

For example, consider the following graph of 5 nodes:

Begin	End	Weight
1	2	5
2	3	6
3	4	2
1	3	15

```
vector<int> shortestReach(int n, vector<vector<int>> edges, int s) {
    //initial
    vector<bool> visited(n, false);
    vector<int> dist(n, INT_MAX);
    //vector<int> parent(n, -1);

    dist[s-1] = 0;

    for (int count=0; count<n-1; count++){
        int u;
        u = minDist(dist, visited, n);
        if(u==-1){
            break;
        }
        visited[u] = true;

        for(int v=0; v<n; v++){
            if(adj[u][v] != 0 && !visited[v] && adj[u][v] + dist[u] < dist[v]){
                dist[v] = adj[u][v] + dist[u];
                //parent[v] = u;
            }
        }
```



Given an undirected graph and a starting node, determine the lengths of the shortest paths from the starting node to all other nodes in the graph. If a node is unreachable, its distance is -1. Nodes will be numbered consecutively from 1 to *n*, and edges will have varying distances or lengths.

For example, consider the following graph of 5 nodes:

Begin	End	Weight
1	2	5
2	3	6
3	4	2
1	3	15

```
break;
```

```
Success
```

```
1
4 4
1 2 24
1 4 20
3 1 3
4 3 12
1
```

2. Problem: *City Roads and Distance*

You are given a city with N intersections (nodes) and M roads (edges). Each road has a travel time. Find the shortest time from a given starting intersection S to every other intersection.
If an intersection is unreachable, output -1.

Input Format:

N M
u1 v1 t1
u2 v2 t2
...
um vm tm
S

Input Variables

Line	Variable (s)	Description
1	N M	- N: Number of nodes (intersections) - M: Number of edges (roads)
2 to M+1	ui vi ti	- A road between nodes ui and vi with travel time ti - The graph is **undirected**, so this road works in both directions
Last Line	S	- Starting node (from where to compute shortest distances)

Example Input

5 6
1 2 4
1 3 2
2 3 1
2 4 5
3 5 10
4 5 3
1

Constraints:

- 1 ≤ N ≤ 10^5
- 1 ≤ M ≤ 2×10^5
- 1 ≤ ti ≤ 10^3

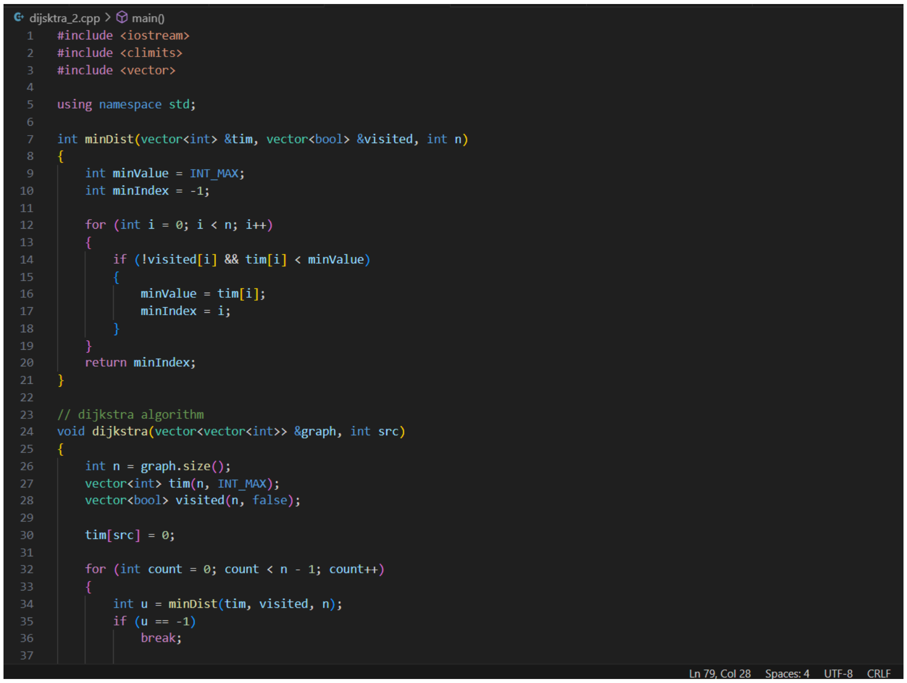
Output Format:

A single line with N-1 space-separated integers: the shortest distances from S to every other node, excluding S.

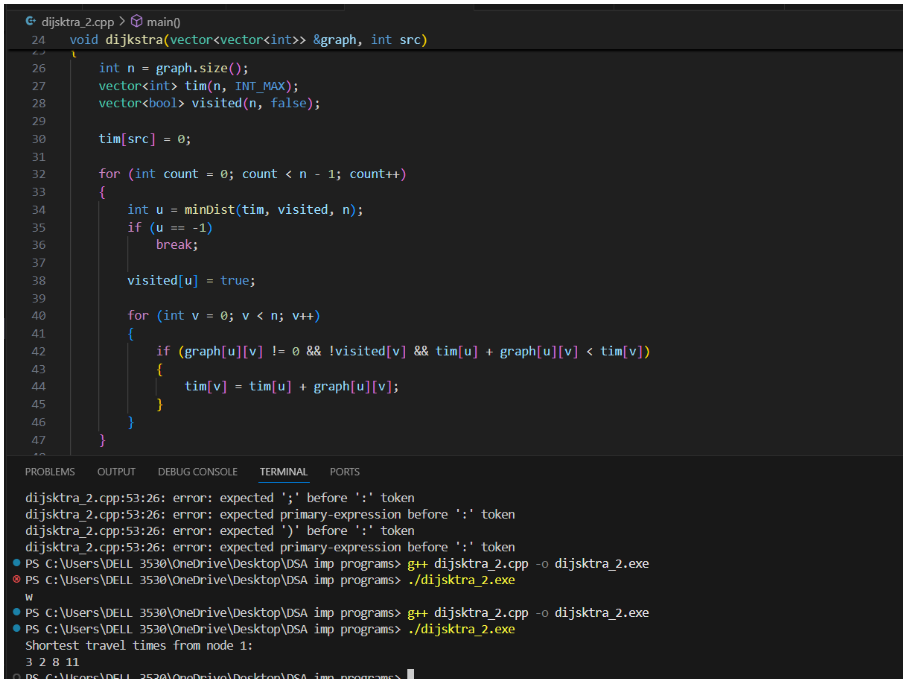
Example Output:

4 2 9 12

Solution Screenshot:

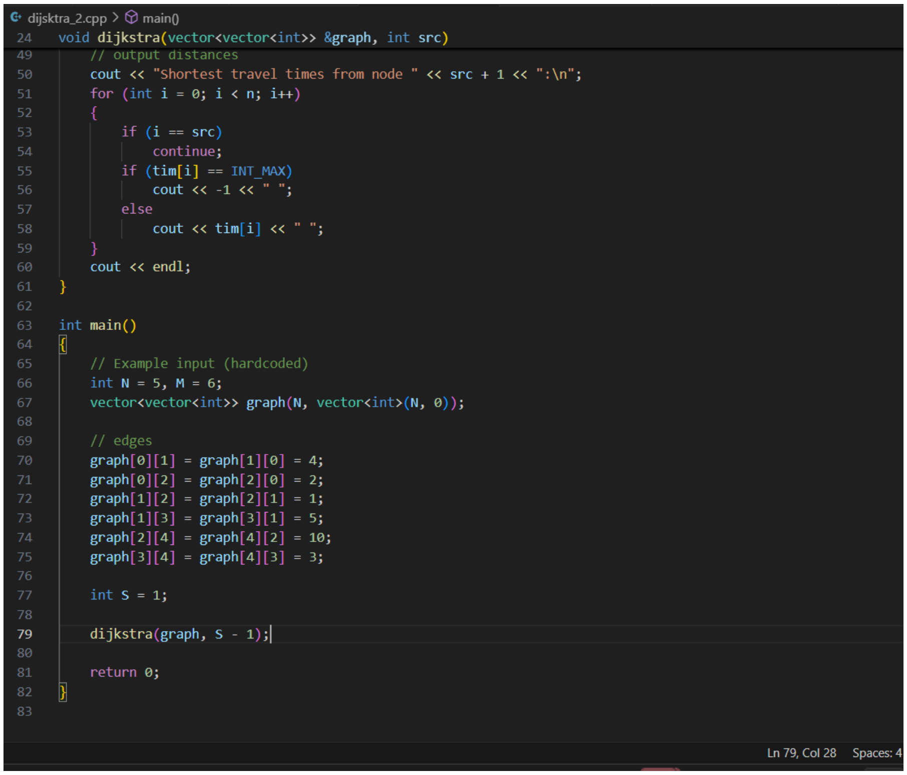


```cpp
#include <iostream>
#include <climits>
#include <vector>

using namespace std;

int minDist(vector<int> &tim, vector<bool> &visited, int n)
{
    int minValue = INT_MAX;
    int minIndex = -1;

    for (int i = 0; i < n; i++)
    {
        if (!visited[i] && tim[i] < minValue)
        {
            minValue = tim[i];
            minIndex = i;
        }
    }
    return minIndex;
}

// dijkstra algorithm
void dijkstra(vector<vector<int>> &graph, int src)
{
    int n = graph.size();
    vector<int> tim(n, INT_MAX);
    vector<bool> visited(n, false);

    tim[src] = 0;

    for (int count = 0; count < n - 1; count++)
    {
        int u = minDist(tim, visited, n);
        if (u == -1)
            break;

```

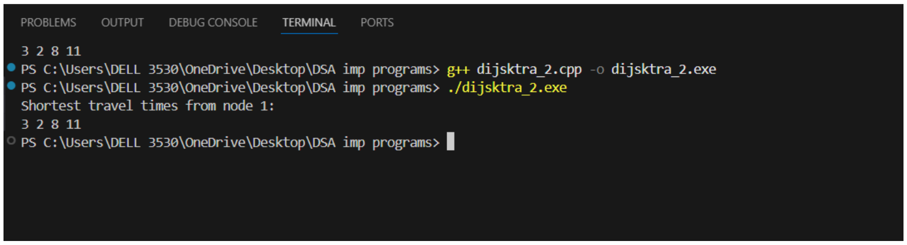


```cpp
void dijkstra(vector<vector<int>> &graph, int src)
{
    int n = graph.size();
    vector<int> tim(n, INT_MAX);
    vector<bool> visited(n, false);

    tim[src] = 0;

    for (int count = 0; count < n - 1; count++)
    {
        int u = minDist(tim, visited, n);
        if (u == -1)
            break;

        visited[u] = true;

        for (int v = 0; v < n; v++)
        {
            if (graph[u][v] != 0 && !visited[v] && tim[u] + graph[u][v] < tim[v])
            {
                tim[v] = tim[u] + graph[u][v];
            }
        }
    }
```

```
dijsktra_2.cpp:53:26: error: expected ';' before ':' token
dijsktra_2.cpp:53:26: error: expected primary-expression before ':' token
dijsktra_2.cpp:53:26: error: expected ')' before ':' token
dijsktra_2.cpp:53:26: error: expected primary-expression before ':' token
PS C:\Users\DELL 3530\OneDrive\Desktop\DSA imp programs> g++ dijsktra_2.cpp -o dijsktra_2.exe
PS C:\Users\DELL 3530\OneDrive\Desktop\DSA imp programs> ./dijsktra_2.exe
w
PS C:\Users\DELL 3530\OneDrive\Desktop\DSA imp programs> g++ dijsktra_2.cpp -o dijsktra_2.exe
PS C:\Users\DELL 3530\OneDrive\Desktop\DSA imp programs> ./dijsktra_2.exe
Shortest travel times from node 1:
3 2 8 11
PS C:\Users\DELL 3530\OneDrive\Desktop\DSA imp programs>
```



```cpp
void dijkstra(vector<vector<int>> &graph, int src)
    // output distances
    cout << "Shortest travel times from node " << src + 1 << ":\n";
    for (int i = 0; i < n; i++)
    {
        if (i == src)
            continue;
        if (tim[i] == INT_MAX)
            cout << -1 << " ";
        else
            cout << tim[i] << " ";
    }
    cout << endl;
}

int main()
{
    // Example input (hardcoded)
    int N = 5, M = 6;
    vector<vector<int>> graph(N, vector<int>(N, 0));

    // edges
    graph[0][1] = graph[1][0] = 4;
    graph[0][2] = graph[2][0] = 2;
    graph[1][2] = graph[2][1] = 1;
    graph[1][3] = graph[3][1] = 5;
    graph[2][4] = graph[4][2] = 10;
    graph[3][4] = graph[4][3] = 3;

    int S = 1;

    dijkstra(graph, S - 1);

    return 0;
}
```



```
3 2 8 11
PS C:\Users\DELL 3530\OneDrive\Desktop\DSA imp programs> g++ dijsktra_2.cpp -o dijsktra_2.exe
PS C:\Users\DELL 3530\OneDrive\Desktop\DSA imp programs> ./dijsktra_2.exe
Shortest travel times from node 1:
3 2 8 11
PS C:\Users\DELL 3530\OneDrive\Desktop\DSA imp programs>
```

3. Multiple Sources (Multi-Dijkstra)

A city has N intersections (nodes) connected by M bidirectional roads (edges). Due to an emergency, several safe zones (K of them) have been designated. Your task is to find, for **every node in the city**, the shortest distance to the **nearest safe zone**.

Key Points:

- Graph is **undirected** and weighted.
- You are given multiple source nodes (safe zones).
- For every node, you want the shortest distance to **any** of the safe zones.
- If a node is unreachable from all safe zones, its distance should be -1.

Input Format

```
N M K
u1 v1 w1
u2 v2 w2
...
uM vM wM
k1 k2 k3 ... kK
```

Variables Explained

Line	Variables	Description
1	N M K	- N: Number of nodes - M: Number of edges - K: Number of safe zones (source nodes)
2 to M+1	ui vi wi	- Roads connecting nodes ui and vi with weight wi (travel time or distance) - Undirected edges
M+2	k1 k2 ... kK	List of K safe zone node numbers

Output Format

- Print a single line with N space-separated integers.
- Each integer is the shortest distance from that node to the **nearest safe zone**.
- For nodes that are themselves safe zones, the distance is 0.
- For unreachable nodes, print -1.

Example

Input:

```
5 6 2
1 2 4
1 3 2
2 3 1
2 4 5
3 5 10
4 5 3
2 5
```

Explanation:

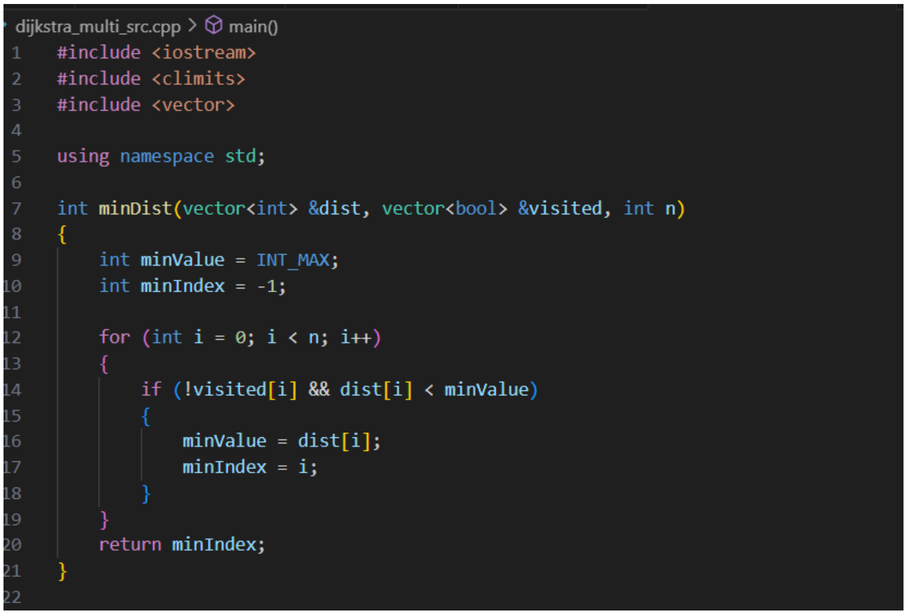
- Nodes: 5
- Edges: 6
- Safe zones: nodes 2 and 5
- Edges connect nodes as described.
- Need shortest distance from every node to closest node in {2, 5}.

Output:
3 0 1 4 0

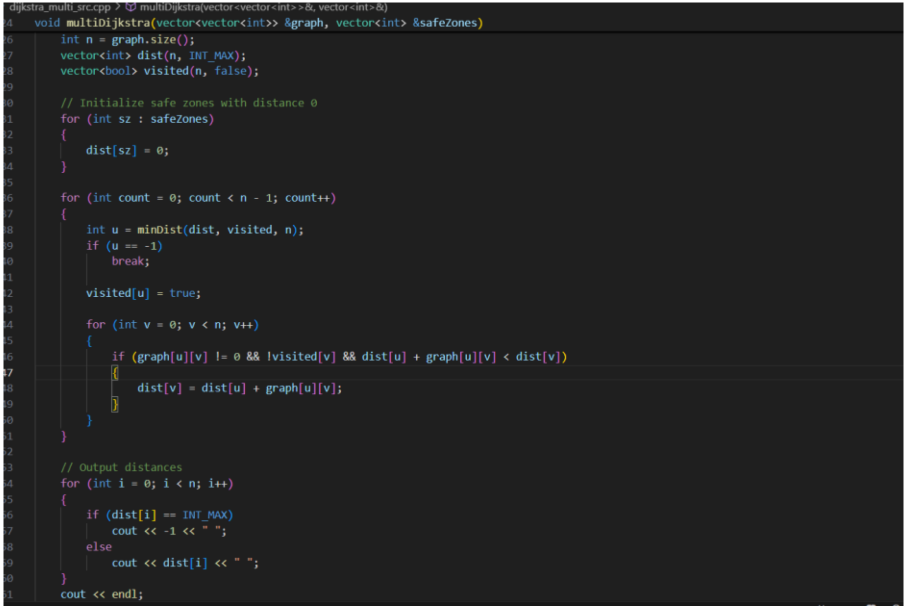
Interpretation:

- Node 1: nearest safe zone is 2, distance 3
- Node 2: safe zone, distance 0
- Node 3: nearest safe zone 2, distance 1
- Node 4: nearest safe zone 5, distance 4
- Node 5: safe zone, distance 0

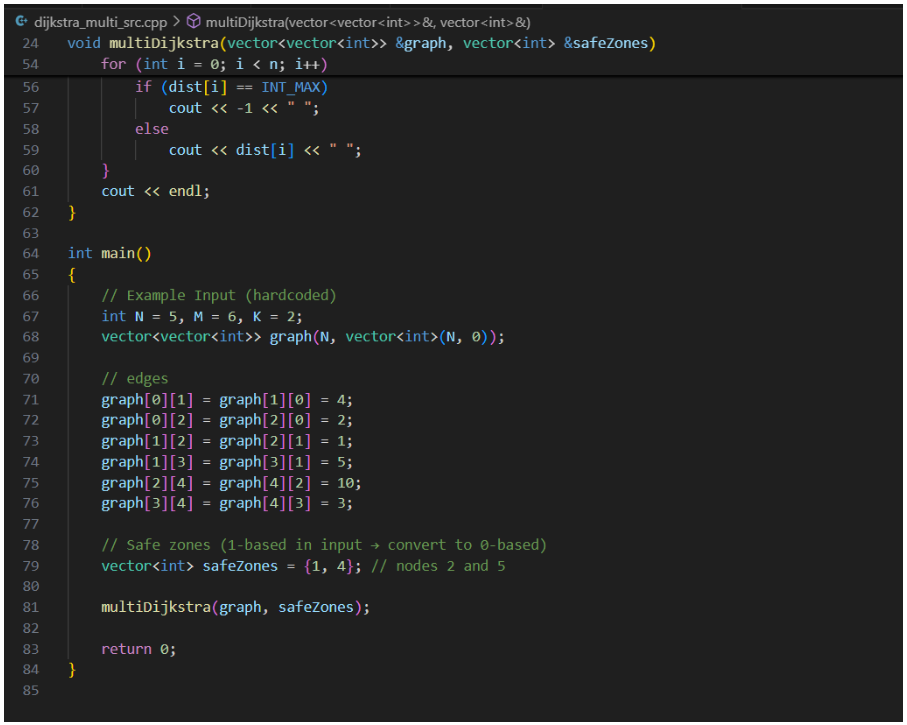
Solution Screenshot:



```cpp
#include <iostream>
#include <climits>
#include <vector>

using namespace std;

int minDist(vector<int> &dist, vector<bool> &visited, int n)
{
    int minValue = INT_MAX;
    int minIndex = -1;

    for (int i = 0; i < n; i++)
    {
        if (!visited[i] && dist[i] < minValue)
        {
            minValue = dist[i];
            minIndex = i;
        }
    }
    return minIndex;
}

```

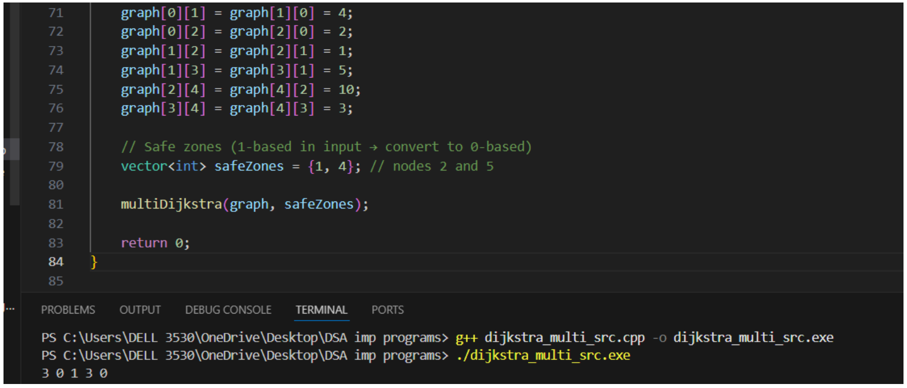

dijkstra_multi_src.cpp > multiDijkstra(vector<vector<int>>&, vector<int>&)

```cpp
void multiDijkstra(vector<vector<int>> &graph, vector<int> &safeZones)
    int n = graph.size();
    vector<int> dist(n, INT_MAX);
    vector<bool> visited(n, false);

    // Initialize safe zones with distance 0
    for (int sz : safeZones)
    {
        dist[sz] = 0;
    }

    for (int count = 0; count < n - 1; count++)
    {
        int u = minDist(dist, visited, n);
        if (u == -1)
            break;

        visited[u] = true;

        for (int v = 0; v < n; v++)
        {
            if (graph[u][v] != 0 && !visited[v] && dist[u] + graph[u][v] < dist[v])
            {
                dist[v] = dist[u] + graph[u][v];
            }
        }
    }

    // Output distances
    for (int i = 0; i < n; i++)
    {
        if (dist[i] == INT_MAX)
            cout << -1 << " ";
        else
            cout << dist[i] << " ";
    }
    cout << endl;
```



dijkstra_multi_src.cpp > multiDijkstra(vector<vector<int>>&, vector<int>&)

```cpp
void multiDijkstra(vector<vector<int>> &graph, vector<int> &safeZones)
    for (int i = 0; i < n; i++)
        if (dist[i] == INT_MAX)
            cout << -1 << " ";
        else
            cout << dist[i] << " ";
    }
    cout << endl;
}

int main()
{
    // Example Input (hardcoded)
    int N = 5, M = 6, K = 2;
    vector<vector<int>> graph(N, vector<int>(N, 0));

    // edges
    graph[0][1] = graph[1][0] = 4;
    graph[0][2] = graph[2][0] = 2;
    graph[1][2] = graph[2][1] = 1;
    graph[1][3] = graph[3][1] = 5;
    graph[2][4] = graph[4][2] = 10;
    graph[3][4] = graph[4][3] = 3;

    // Safe zones (1-based in input → convert to 0-based)
    vector<int> safeZones = {1, 4}; // nodes 2 and 5

    multiDijkstra(graph, safeZones);

    return 0;
}
```



```cpp
    graph[0][1] = graph[1][0] = 4;
    graph[0][2] = graph[2][0] = 2;
    graph[1][2] = graph[2][1] = 1;
    graph[1][3] = graph[3][1] = 5;
    graph[2][4] = graph[4][2] = 10;
    graph[3][4] = graph[4][3] = 3;

    // Safe zones (1-based in input → convert to 0-based)
    vector<int> safeZones = {1, 4}; // nodes 2 and 5

    multiDijkstra(graph, safeZones);

    return 0;
}
```

```
PS C:\Users\DELL 3530\OneDrive\Desktop\DSA imp programs> g++ dijkstra_multi_src.cpp -o dijkstra_multi_src.exe
PS C:\Users\DELL 3530\OneDrive\Desktop\DSA imp programs> ./dijkstra_multi_src.exe
3 0 1 3 0
```

4. Problem: *Escape the Island*

You're stranded on an island represented as a network of N areas (nodes) and M paths (edges). A **volcano** has erupted at node V, and lava begins to spread through the paths at a steady rate (one unit of time per unit edge weight).

You are at node S. You want to **escape to any other area** before the lava reaches it. You can move instantly after the eruption, but lava spreads simultaneously.

Your task is to find all nodes that you can reach **before the lava does** — meaning, the shortest path from S to a node must be strictly **less than** the time lava takes to reach that node from V.

Input Format

N M
u1 v1 w1
u2 v2 w2

```
...
uM vM wM
S V
```

Input Variables Explained

Line	Variables	Description
1	N M	- N: Number of nodes - M: Number of edges
2–M+1	ui vi wi	- Undirected edge between ui and vi with weight wi
M+2	S V	- S: Your starting node - V: Volcano node (lava source)

Output Format

- A list of all **nodes you can escape to**, including your current node S, if applicable.
- Nodes should be printed in **ascending order**.
- If no node is safe, print None.

Example Input

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

- Nodes: 6
- Edges: 7
- Start node: 1
- Volcano: 4

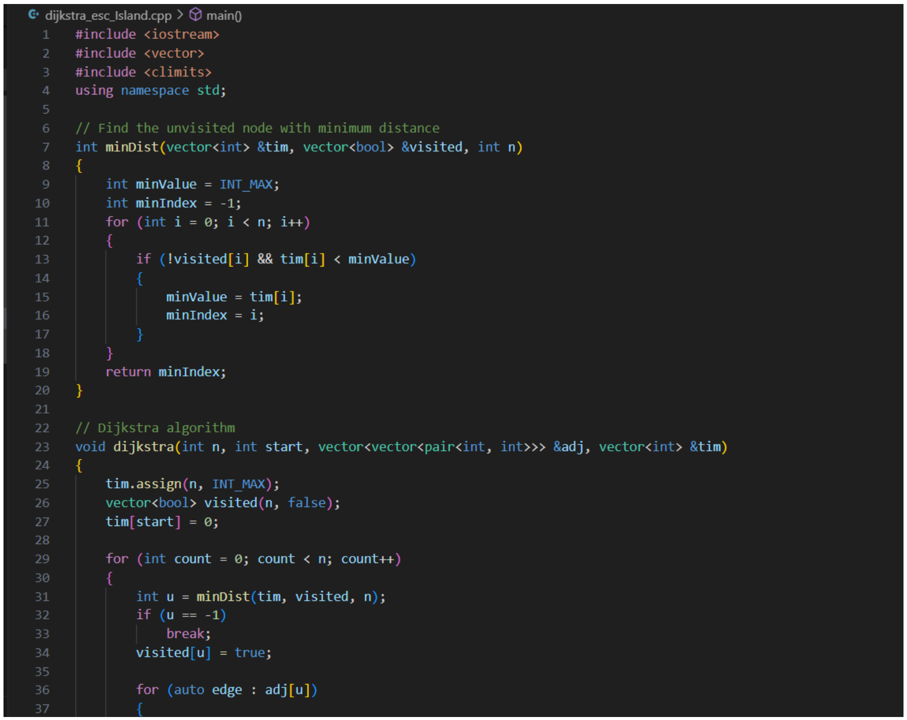
Example Output

```
1 2 5 6
```

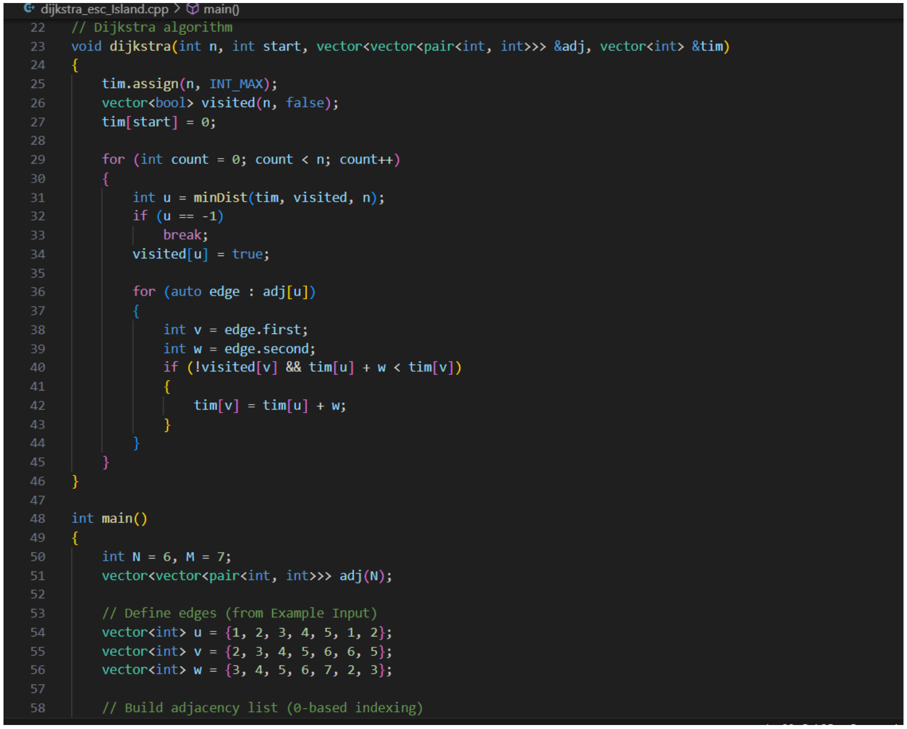
Explanation:

- From node 1, you can reach nodes 2, 5, and 6 **faster** than lava from node 4.
- Node 3 is unreachable before lava gets there.
- Node 4 is the volcano source, so it's instantly unsafe.

Solution Screenshot:

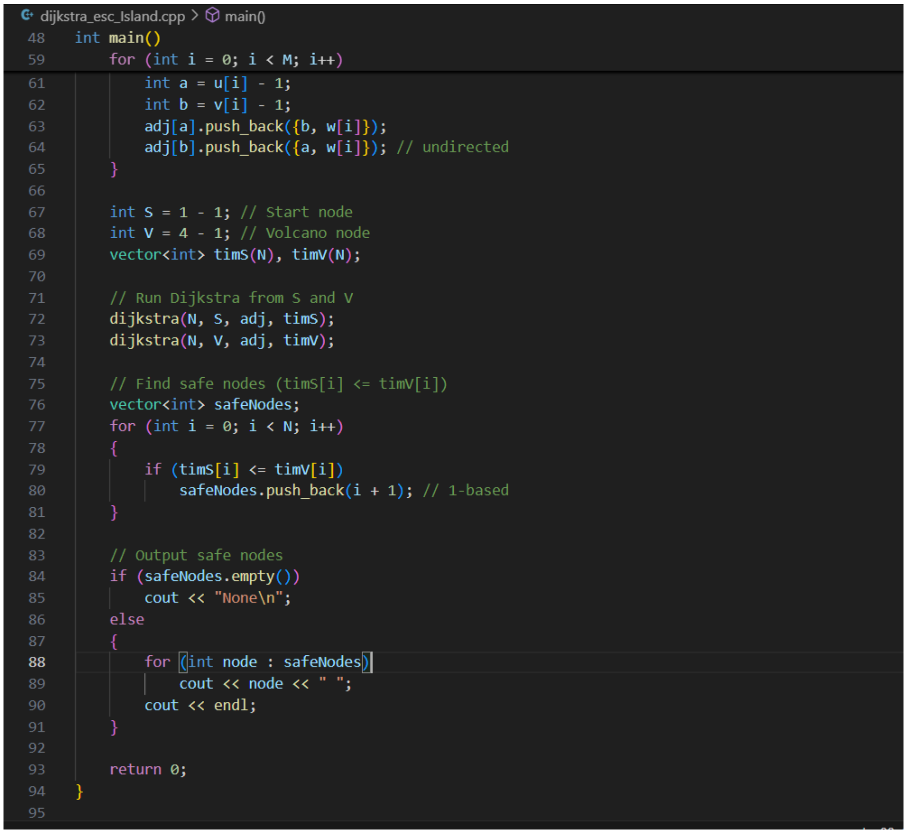


```cpp
#include <iostream>
#include <vector>
#include <climits>
using namespace std;

// Find the unvisited node with minimum distance
int minDist(vector<int> &tim, vector<bool> &visited, int n)
{
    int minValue = INT_MAX;
    int minIndex = -1;
    for (int i = 0; i < n; i++)
    {
        if (!visited[i] && tim[i] < minValue)
        {
            minValue = tim[i];
            minIndex = i;
        }
    }
    return minIndex;
}

// Dijkstra algorithm
void dijkstra(int n, int start, vector<vector<pair<int, int>>> &adj, vector<int> &tim)
{
    tim.assign(n, INT_MAX);
    vector<bool> visited(n, false);
    tim[start] = 0;

    for (int count = 0; count < n; count++)
    {
        int u = minDist(tim, visited, n);
        if (u == -1)
            break;
        visited[u] = true;

        for (auto edge : adj[u])
        {
```

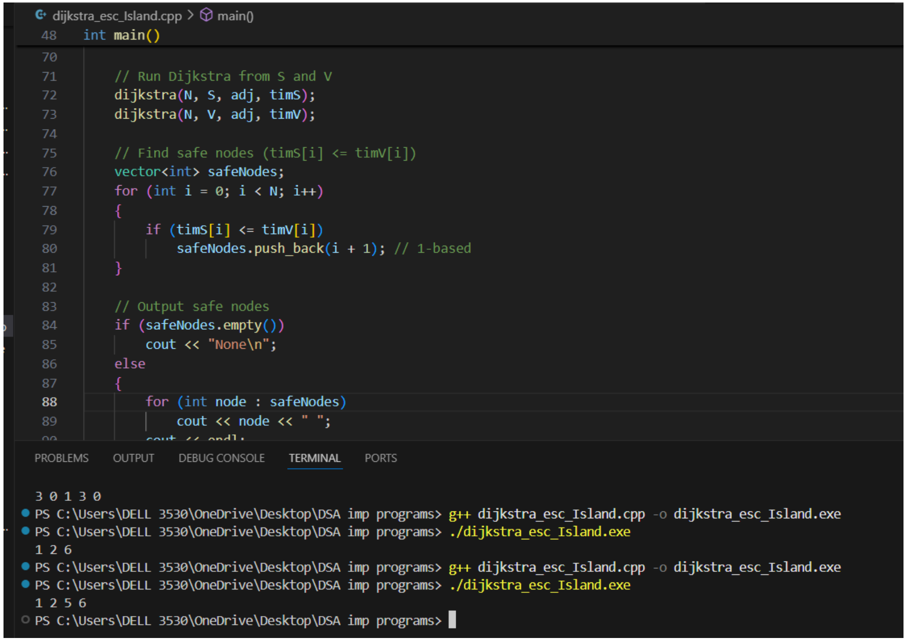


```cpp
// Dijkstra algorithm
void dijkstra(int n, int start, vector<vector<pair<int, int>>> &adj, vector<int> &tim)
{
    tim.assign(n, INT_MAX);
    vector<bool> visited(n, false);
    tim[start] = 0;

    for (int count = 0; count < n; count++)
    {
        int u = minDist(tim, visited, n);
        if (u == -1)
            break;
        visited[u] = true;

        for (auto edge : adj[u])
        {
```

```cpp
            int v = edge.first;
            int w = edge.second;
            if (!visited[v] && tim[u] + w < tim[v])
            {
                tim[v] = tim[u] + w;
            }
        }
    }
}

int main()
{
    int N = 6, M = 7;
    vector<vector<pair<int, int>>> adj(N);

    // Define edges (from Example Input)
    vector<int> u = {1, 2, 3, 4, 5, 1, 2};
    vector<int> v = {2, 3, 4, 5, 6, 6, 5};
    vector<int> w = {3, 4, 5, 6, 7, 2, 3};

    // Build adjacency list (0-based indexing)
```



```cpp
int main()
    for (int i = 0; i < M; i++)
        int a = u[i] - 1;
        int b = v[i] - 1;
        adj[a].push_back({b, w[i]});
        adj[b].push_back({a, w[i]}); // undirected
    }

    int S = 1 - 1; // Start node
    int V = 4 - 1; // Volcano node
    vector<int> timS(N), timV(N);

    // Run Dijkstra from S and V
    dijkstra(N, S, adj, timS);
    dijkstra(N, V, adj, timV);

    // Find safe nodes (timS[i] <= timV[i])
    vector<int> safeNodes;
    for (int i = 0; i < N; i++)
    {
        if (timS[i] <= timV[i])
            safeNodes.push_back(i + 1); // 1-based
    }

    // Output safe nodes
    if (safeNodes.empty())
        cout << "None\n";
    else
    {
        for (int node : safeNodes)
            cout << node << " ";
        cout << endl;
    }

    return 0;
}
```



```cpp
int main()

    // Run Dijkstra from S and V
    dijkstra(N, S, adj, timS);
    dijkstra(N, V, adj, timV);

    // Find safe nodes (timS[i] <= timV[i])
    vector<int> safeNodes;
    for (int i = 0; i < N; i++)
    {
        if (timS[i] <= timV[i])
            safeNodes.push_back(i + 1); // 1-based
    }

    // Output safe nodes
    if (safeNodes.empty())
        cout << "None\n";
    else
    {
        for (int node : safeNodes)
            cout << node << " ";
        cout << endl;
```

```
3 0 1 3 0
PS C:\Users\DELL 3530\OneDrive\Desktop\DSA imp programs> g++ dijkstra_esc_Island.cpp -o dijkstra_esc_Island.exe
PS C:\Users\DELL 3530\OneDrive\Desktop\DSA imp programs> ./dijkstra_esc_Island.exe
1 2 6
PS C:\Users\DELL 3530\OneDrive\Desktop\DSA imp programs> g++ dijkstra_esc_Island.cpp -o dijkstra_esc_Island.exe
PS C:\Users\DELL 3530\OneDrive\Desktop\DSA imp programs> ./dijkstra_esc_Island.exe
1 2 5 6
PS C:\Users\DELL 3530\OneDrive\Desktop\DSA imp programs>
```

Conclusion	Thus, we have studied Dijkstra’s algorithm and its time complexity