Project 2 Report: Graph Algorithms and Data Structures

Introduction

This project demonstrates two core graph algorithms implemented in Python:

1. Dijkstra's Algorithm: Solves the single-source shortest path problem in both directed and undirected graphs with non-negative weights.

2. Kruskal's Algorithm: Computes a Minimum Spanning Tree (MST) for connected, undirected graphs with weighted edges.

Both algorithms accept input from a structured text file and print results to the console. Each algorithm is tested on four different graphs.

Dijkstra's Algorithm: Shortest Path

Description

Dijkstra's algorithm finds the shortest paths from a given source node to all other nodes in a weighted graph. It works by exploring the nearest unvisited node and updating the shortest distances to neighboring nodes iteratively.

Pseudocode

```
function Dijkstra(graph, source):
    dist[source] = 0
    for each vertex v in graph:
        if v != source:
            dist[v] = infinity
    prev = empty map
    Q = priority queue with all vertices and their dist values

    while Q is not empty:
        u = vertex in Q with smallest dist
        remove u from Q
        for each neighbor v of u:
            alt = dist[u] + weight(u, v)
            if alt < dist[v]:
                dist[v] = alt
                prev[v] = u
                update v in Q
    return dist, prev
```

Data Structures Used

- Adjacency list: To store graph edges efficiently.

- Min-heap (priority queue): To quickly retrieve the node with the lowest tentative distance.

- Dictionary: To track distances and reconstruct paths.

Runtime Analysis

- Inserting, updating, and removing from the heap takes O(log V).

- Each edge is checked at most once, giving total complexity O((V + E) log V).

Sample Input and Output

Input (undirected1.txt):

```
9 12 U
A B 4
A C 3
C D 1
...
A
```

Output:

```
Path to B: A -> B | Cost: 4

Path to D: A -> C -> D | Cost: 4
```

Kruskal's Algorithm: Minimum Spanning Tree

Description

Kruskal's algorithm builds a Minimum Spanning Tree by sorting all graph edges by weight and adding the smallest edge that does not form a cycle. Union-Find is used to manage disjoint sets of connected nodes.

Pseudocode

```
function Kruskal(graph):
    MST = empty set
    sort all edges by weight
    initialize Union-Find structure

    for each edge (u, v) in sorted edges:
        if find(u) != find(v):
            MST.add((u, v))
            union(u, v)

    return MST
```

Data Structures Used

- Edge list: To sort edges by weight.

- Union-Find: To detect and avoid cycles efficiently.

Runtime Analysis

- Sorting edges takes O(E log E).

- Union-Find operations take nearly constant time.

- Total complexity: O(E log E).

Sample Input and Output

Input (undirected1.txt):

```
9 12 U
A B 4
A C 3
C D 1
...
```

Output:

```
MST Edges:

A - C (weight 3)

C - D (weight 1)

...

Total cost: 20
```

How to Set Up and Run:

1. Prerequisites

Make sure you have Python 3 installed on your computer.

2. Input File Format

The corresponding file format would be:

```
6 10 U
A B 1
A C 2
B C 1
B D 3
B E 2
C D 1
C E 2
D E 4
D F 3
E F 3
A
```

3. How to Run shortest_path.py

To run Dijkstra's algorithm on a graph file:

```
python shortest_path.py undirected1.txt
```

This program will display the shortest path from the source node to all other nodes, along with the total cost of each path.

Example output:

```
Source: A Path to B: A -> B | Cost: 1 Path to C: A -> C | Cost: 2 Path to D: A -> B -> D |
Cost: 4 ...
```

4. How to Run mst.py

To run the Minimum Spanning Tree algorithm:

```
python mst.py undirected1.txt
```

Important: This program only works for undirected graphs. If you try to use a directed graph, the program will show an error.

Example output:

```
MST Edges: A - B (weight 1) B - C (weight 1) C - D (weight 1) ... Total cost: 12
```

5. Sample Commands

To run shortest_path.py with a directed graph:

```
python shortest_path.py directed1.txt
```

To run mst.py with a second undirected graph:

```
python mst.py undirected2.txt
```

Conclusion

This project successfully demonstrates the implementation of Dijkstra’s and Kruskal’s algorithms using appropriate data structures. Both algorithms are run on multiple graph configurations and tested using flexible input files. The report includes clear descriptions, pseudocode, runtime analysis, and instructions for running the programs.