Course: Design and Analysis of Algorithm

(CT-363)

GUI Application for Minimum Spanning Tree Construction and Analysis

Prepared by:

Muhammad Hamza (DT - 22048)

Submitted To:

Dr Usman Amjad

Contents

Abstract 3

Introduction 3

Objectives 3

Tools and Technologies 3

Methodology 4

Features 4

Time Complexity Analysis 4

- Prim's Algorithm 4
- Kruskal's Algorithm 4

Code: 5

Results and Observations 6

Limitations 8

Future Work 9

Real-Time Applications of Minimum Spanning Trees 9

- Network Design 9
- Transportation Networks 9
- Clustering in Data Mining 9
- Image Processing 9
- Civil Infrastructure Planning 9
- Wireless Sensor Networks 9
- Approximation Algorithms 10

System Architecture Overview 10

Conclusion 10

Abstract
This project presents a Python-based GUI application for constructing and analyzing Minimum Spanning Trees (MST) using Prim's and Kruskal's algorithms. The application enables users to input graph data, visualize MSTs, compare algorithm performance, and handle CSV data import/export. The tool enhances understanding of graph theory and algorithm analysis, making it valuable for both educational and practical purposes.

Introduction
Graph theory is a cornerstone of computer science with applications in networking, logistics, clustering, and circuit design. Among graph algorithms, MST algorithms like Prim's and Kruskal's play a crucial role in optimizing connectivity at minimal cost. This project aims to implement these algorithms within a user-friendly GUI, allowing users to interactively construct, visualize, and analyze MSTs.

Objectives
- Develop an intuitive graphical interface for graph construction.
- Implement Prim's and Kruskal's algorithms for MST generation.
- Visualize MSTs to aid understanding and analysis.
- Compare the outputs and performance of both algorithms.
- Provide options to import/export graph data via CSV files.

Tools and Technologies
Language: Python

Libraries:

- Tkinter (GUI)
- NetworkX (graph operations)
- Matplotlib (visualization)
- CSV module (file handling)

Algorithms:

- Prim's

- Kruskal's

Methodology

Users can add vertices, edges, and weights via GUI input. Prim's Algorithm selects the minimum weight edge from a visited node to an unvisited node. Kruskal's Algorithm sorts all edges and adds the smallest edge that does not form a cycle. NetworkX and Matplotlib render the input graph and the resulting MSTs. Users can import and export edge lists for reproducibility. The tool compares the total weight and time complexity of both algorithms. Users can save the generated MST visualizations (Prim's and Kruskal's) as image files for documentation or presentation.

Features

- Add vertices, edges, and weights dynamically.
- Display list of edges and weights.
- Execute Prim's and Kruskal's algorithms.
- Show total weight of the computed MST.
- Visualize both MSTs side by side.
- Import and export graphs using CSV files.
- Reset graph for fresh input.
- Compare accuracy and total weight of Prim's and Kruskal's MSTs.
- Handle invalid inputs and provide user-friendly error messages.
- Allow users to save the MST visualizations as image files.
- Provide clear success or error alerts for CSV operations.

Time Complexity Analysis

Prim's Algorithm

Prim's algorithm utilizes a priority queue to select the edge with the smallest weight at each step. Each vertex is inserted into the heap once, and each edge is relaxed at most once. The logarithmic term arises from heap operations. Therefore, the total time complexity is efficient for dense graphs where the number of edges is large relative to the number of vertices.

Kruskal's Algorithm

Kruskal's algorithm sorts all the edges based on weight, which requires O(E log E) time. The Union-Find operations (to check and merge connected components) are performed in near constant time $O(\alpha(V))$, where α is the inverse Ackermann function, which grows very slowly. Hence, the sorting step dominates the overall complexity. This algorithm performs well on sparse graphs where the number of edges is significantly less than the number of possible edges.

Code:

```
def prim_mst(self, n, graph):
        visited = set()
        heap = [(0, 0, -1)]
        mst = []
        total_weight = 0

        while len(visited) < n and heap:
            w, u, prev = heapq.heappop(heap)
            if u in visited:
                continue
            visited.add(u)
            if prev != -1:
                mst.append((prev, u, w))
                total_weight += w
            for weight, v in graph[u]:
                if v not in visited:
                    heapq.heappush(heap, (weight, v, u))

        return (mst, total_weight) if len(mst) == n - 1 else (None, None)

    def kruskal_mst(self, n, edges):
        parent = list(range(n))

        def find(u):
            while parent[u] != u:
                parent[u] = parent[parent[u]]
                u = parent[u]
            return u

        def union(u, v):
            u_root = find(u)
            v_root = find(v)
            if u_root == v_root:
                return False
            parent[v_root] = u_root
            return True

        mst = []
        total_weight = 0
        count = 0

        for w, u, v in sorted(edges):
```

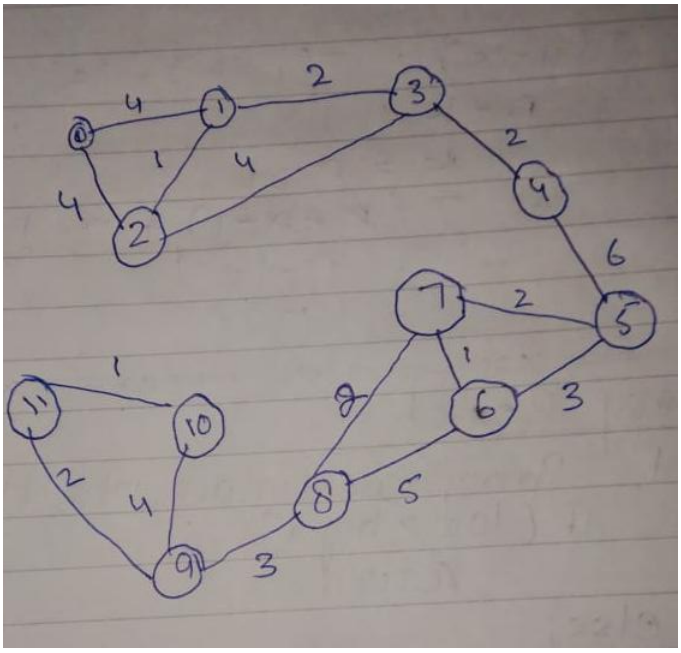
```
            if union(u, v):
                mst.append((u, v, w))
                total_weight += w
                count += 1
                if count == n - 1:
                    break

        return total_weight, mst if count == n - 1 else (None, None)
```

Results and Observations

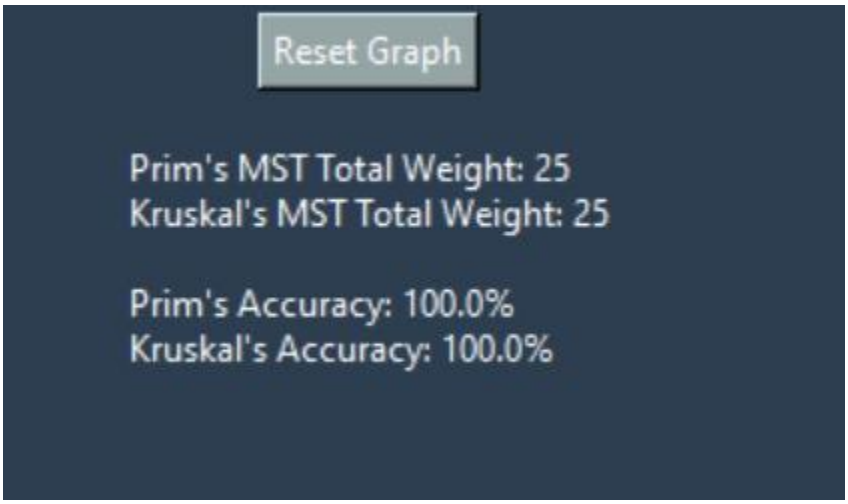
After successful implementation of both Prim's and Kruskal's algorithms, the application generates and visualizes the corresponding Minimum Spanning Trees (MSTs) based on user-defined graphs. The results are presented in the form of graphical diagrams, showing how edges are selected to form the MST while minimizing total weight.

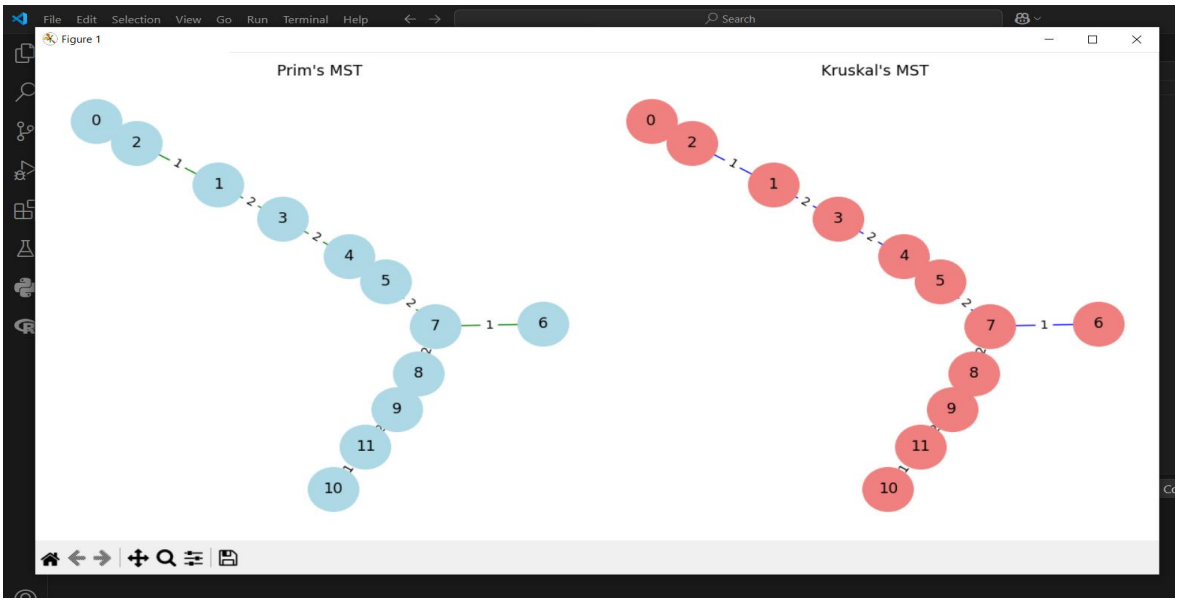
Input Graph:



Output:


Number of Vertices:
12
Start Vertex:
End Vertex:
Weight:
Edges Added (u, v, w):
(2, 3, 4)
(3, 4, 2)
(4, 5, 6)
(5, 6, 3)
(5, 7, 2)
(6, 7, 1)
Add Edge
Compute MST
Import Edges
Export Edges
Reset Graph





Observations:

- Both algorithms produce an MST with the same total weight (as expected) but may select different edges depending on graph structure.
- Prim's algorithm starts from a root node and grows the MST, while Kruskal's algorithm builds the MST edge by edge in increasing order of weight.
- The visual comparison helps in understanding the working of each algorithm intuitively.

Limitations

- The application currently supports only undirected and connected graphs.
- Graph input must be done manually through the GUI, which may be inefficient for large graphs.

- No support for real-time step-by-step animation, which could further aid educational understanding.
- Negative edge weights are not handled explicitly and may cause unpredictable behavior.
- The visualization layout (spring layout) may become cluttered with a high number of vertices.

Future Work

- Add support for directed graphs and disconnected components.
- Integrate Boruvka's algorithm for additional MST comparison.
- Provide step-by-step animation of each algorithm’s progress for better learning.
- Introduce performance benchmarking for large graphs.
- Optimize the GUI for better layout control and responsiveness.

Real-Time Applications of Minimum Spanning Trees

Network Design

MSTs are widely used in designing least-cost networks, such as telecommunication, electrical grids, and computer networks. They help ensure all nodes are connected using the minimal amount of cable or wire.

Transportation Networks

Transportation planners use MSTs to design road systems, railways, and airline routes that minimize construction costs while maintaining full connectivity.

Clustering in Data Mining

In hierarchical clustering algorithms, MSTs are used to find natural groupings in data based on minimum connection paths.

Image Processing

MSTs are applied in image segmentation, pattern recognition, and shape analysis to connect pixels or features with minimal edge weights.

Civil Infrastructure Planning

Urban planners use MSTs for laying down pipelines, water distribution systems, and power grids to optimize material usage and reduce expenses.

Wireless Sensor Networks

MSTs assist in optimizing communication between sensors while minimizing energy usage, extending the overall lifetime of the network.

Approximation Algorithms

MSTs serve as foundational tools in approximation algorithms for NP-hard problems such as the Traveling Salesman Problem (TSP).

System Architecture Overview

The architecture of the application is modular, with separation between GUI logic, algorithm implementation, and file handling mechanisms. Key components include:

- Graph Module: Handles internal data structures using NetworkX for vertices, edges, and weights.
- Algorithm Module: Implements Prim's and Kruskal's logic independently for modular testing.
- Visualization Module: Uses Matplotlib to plot graphs and MSTs with clear node and edge labels.
- CSV Handler: Manages import and export operations for user graph data.
- Main GUI: Integrates all modules using Tkinter with input fields, buttons, and canvas for visualization.

Conclusion

This project successfully demonstrates the application of two classical graph algorithms—Prim's and Kruskal's—for constructing Minimum Spanning Trees. By integrating algorithm logic with GUI and visualization tools, it provides an intuitive and interactive learning platform for understanding MSTs. The comparison between algorithms through both code and graphical output enhances conceptual clarity and encourages further exploration into graph theory and optimization.