

# Project 4 Report: Minimum Spanning Tree

CSE 5311 – Fall 2025

Student: Md Mahedi Hasan Rigan

Student ID : 1002310545

---

## 1. Introduction

This project implements and visualizes two Minimum Spanning Tree (MST) algorithms:

- **Prim's Algorithm**
- **Kruskal's Algorithm**

The system provides:

1. An easy-to-use GUI tool for visualizing MSTs
2. Support for entering your own graph using an edge list
3. A feature to create random graphs and test how long they take to run
4. A tool to run experiments and plot running time against input size
5. Automatic runtime comparisons for different graph sizes

The code can be found in this link : [Github](#)

---

## 2. System Design and Implementation

The project consists of four main python files:

- **prims.py** – Implementation of Prim's MST algorithm
- **kruskal.py** – Implementation of Kruskal's MST algorithm
- **gui.py** – Full GUI for input, visualization, and experiments feature
- **utils.py** – Helper functions for parsing input and drawing graphs

The GUI lets you:

- Load your own graph input
  - Run either Prim's or Kruskal's algorithm
  - See the MST visually
  - Check the runtime in milliseconds
  - Generate random graphs
  - Run experiments by changing the number of nodes and edges
  - View charts that compare the runtimes
- 

### 3. Data Structures Used

#### 3.1 Graph Representation

The graph is stored as:

- **Node list:** A list of labels ("A", "B", "C")
- **Edge list:** A list of tuples ( $u, v, \text{weight}$ )

#### 3.2 Prim's Algorithm Data Structures

- **Min-Heap (Priority Queue) :** Used to select the cheapest outgoing edge
- **Visited Set :** Tracks nodes already included in the MST.
- **Adjacency List :** For faster neighbor access.

#### 3.3 Kruskal's Algorithm Data Structures

- **Disjoint Set Union (Union-Find)**
  - **parent[]** array
  - **rank[]** array
- **Sorted Edge List :** Edges sorted by weight.

#### 3.4 GUI Data Structures

- **Matplotlib Figure + Axes** for graph drawing
  - **Tkinter Widgets** for user input and controls
-

## 4. Features

### 4.1 Input Parser

The function `parse_edge_list()` reads input such as:

A B 4

A C 3

B C 2

It converts it into a structured node and edge list.

### 4.2 Minimum Spanning Tree Computation

The user selects an algorithm using a dropdown menu:

- "prim"
- "kruskal"

The selected algorithm is executed and the MST is returned as:

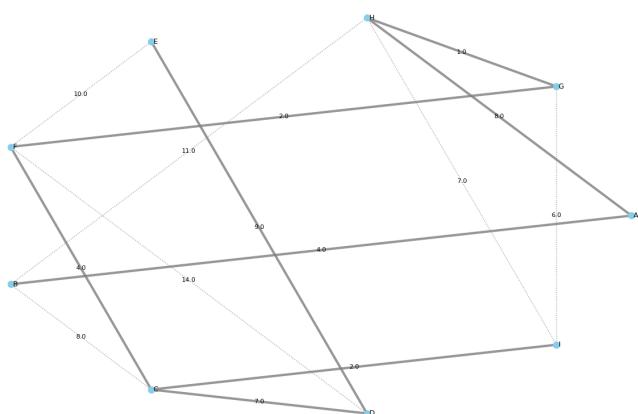
`mst_edges: [(u, v, w), ...]`

`total_weight: float`

### 4.3 Graph Visualization

Using `matplotlib` python library, the system:

- Draws nodes arranged in a circle
- Draws all graph edges
- Highlights MST edges in thicker lines



## 4.4 Random Graph Generator

Users enter:

- Number of nodes
- Number of edges

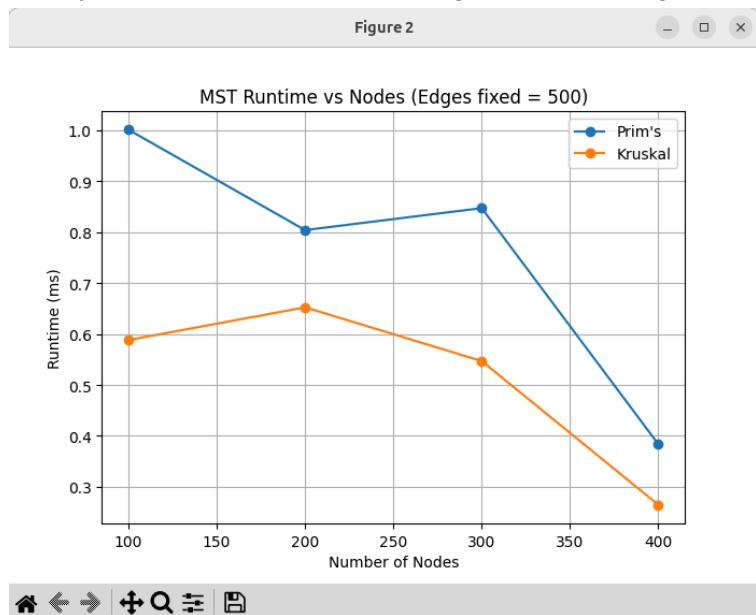
The program creates random graphs and runs MST on them.

## 4.5 Series Experiment Tool

Two experiment modes:

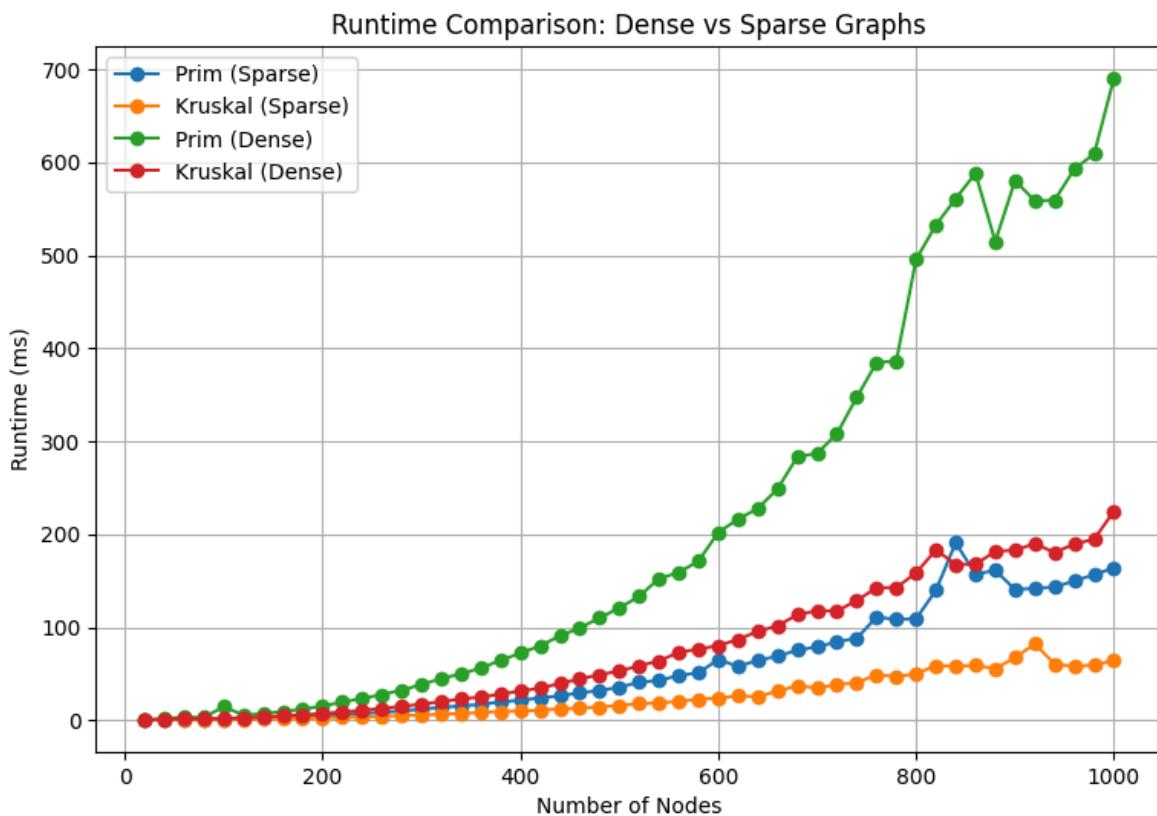
1. **Node Variable Mode**
  - Nodes change (10, 20, 50, 80)
  - Edges fixed
  - Runtime vs Node Count
2. **Edge Variable Mode**
  - Edges change (50, 100, 200, 300)
  - Nodes fixed
  - Runtime vs Edge Count

The system produces a matplotlib graph comparing:



## 4.6 Sparse and Dense Graph Visualization

Users can also enter maximum node number to visualize the dense and sparse graph runtime in a plot.



## 5. Time Complexity

Algorithm	Time Complexity	Notes
Prim's	$O(E \log V)$ using min-heap	Good for dense graphs
Kruskal's	$O(E \log E)$	Good for sparse graphs

## 6. Discussion

### 6.1 Algorithm Comparison Summary

Condition	Better Algorithm
Sparse Graphs	Kruskal
Dense Graphs	Prim
Graphs represented as adjacency list	Prim
Graphs represented as edge list	Kruskal
When sorting edges is expensive	Prim
When Union-Find operations are cheap	Kruskal

---

### 6.2 How to improve their runtime?

- Using **Fibonacci heap** reduces Prim to:  $O(E + V \log V)$
  - Using **path compression + union by rank** (already done) gives Kruskal:  $O(E \cdot \alpha(V))$
  - Storing adjacency lists in highly cache-friendly structures improves Prim further.
  - Parallelizing edge sorting improves Kruskal's practical runtime.
- 

## 7. Code Quality

The project includes:

- Clean modular code
  - Proper function names
  - Clear GUI separation (logic vs visual layer)
  - Project divided into multiple files for better readability
  - Random generators instead of hard-coded inputs
- 

## 8. Files Included

1. **gui.py** — Main application with GUI

2. **prims.py** — Prim's algorithm
  3. **kruskal.py** — Kruskal's algorithm
  4. **utils.py** — Input parser and graph drawing tools
  5. **mst\_gui.py** — Entry script
  6. **README.md** — Instructions
  7. **runtime\_dense\_sparse\_graph.jpg** — Dense and Sparse graph runtime graph
-