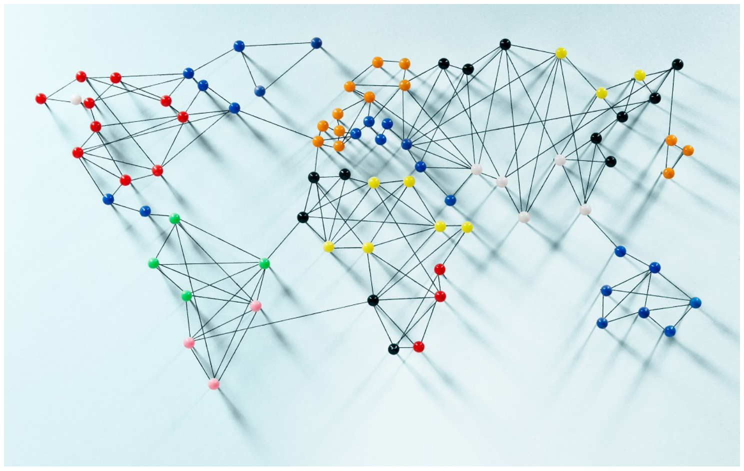
ECIE 3101 and ECIE 3312

MST Visualizer

Sem 2 2022/2023



Lecturer: Madam Rashidah Funke Olanrewaju

Group members:

1. Muhammad Azim Iskandar Bin Mohd Sofi (2010613)
2. Harith Irfan Bin Hashim (2013509)
3. Muhammad Adam Bin Mohd Ramzah (2014605)
4. Saifullah Bin Mohd Razali (2018385)
5. Muhammad Amin Bin Zulkarnain (2017167)

Objective

The objective of this project was to create a Graphical User Interface (GUI) that visualizes the application of two different algorithms – Kruskal's algorithm and Prim's algorithm – in finding the Minimum Spanning Tree (MST) of a graph. The primary goal was to make the understanding of these algorithms easier and interactive, aiming to promote better learning.

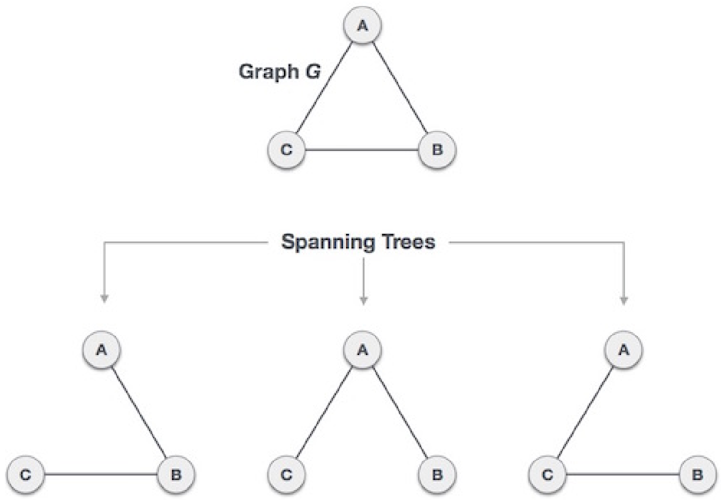
The project has been implemented in Python, using libraries such as NetworkX for handling and operating on the graph, and Tkinter for creating the GUI. An executable file was created to make the program accessible to users.

Introduction

Spanning Tree and Minimum Spanning Tree

A Spanning Tree is a subset of Graph G that has the least amount of connections feasible connecting every vertex. As a result, a spanning tree has no cycles and cannot be detached.

This concept leads us to the conclusion that any connected and undirected Graph G contains at least one spanning tree. A disconnected graph does not have any spanning tree, as it cannot be spanned to all its vertices.



A complete undirected graph can have a maximum n^{n-2} number of spanning trees, where n is the number of nodes. In the above addressed example, n is 3, hence 33−2 = 3 spanning trees are possible.

A Minimum Spanning Tree (MST) is a subset of the edges of a connected, edge-weighted undirected graph that connects all the vertices together, without any cycles, and with the minimum possible total edge weight. The MST has direct applications in network design, including telephone networks and computer networks.

Kruskal's Algorithm

Kruskal's algorithm is a minimum-spanning-tree algorithm that finds an edge of the least possible weight that connects any two trees in the forest. It is a greedy algorithm in graph theory, as it finds a minimum spanning tree for a connected weighted graph, adding increasing cost arcs at each step. We begin with the edges that have the lowest weight and keep adding edges until we accomplish our goal. The following are the stages for applying Kruskal's algorithm:

1. Sort all of the edges from light to heavy.
2. Add the edge to the spanning tree that has the lowest weight. Reject this edge if adding it resulted in a loop.
3. Keep adding the edges until all vertices have been reached.

Prim's Algorithm

Prim's algorithm is also a minimum-spanning-tree algorithm that finds an edge of the least possible weight that connects any two trees in the forest. It differs from Kruskal's algorithm, as it starts with an arbitrary node and grows the MST one edge at a time. Starting with one vertex, we continue to add edges with the lowest weight until we achieve our objective. Implementing Prim's algorithm involves the following steps:

1. Set up a random vertex to serve as the minimal spanning tree's first node.
2. Find the smallest edge between the tree's existing vertices and all other edges, then add it to the tree.
3. Repeat step 2 as necessary until a minimal spanning tree is formed.

Methodology

Our MST Visualization Tool was designed and developed leveraging the Python programming language, specifically employing the NetworkX and Tkinter libraries to fulfill the requirements of the project.

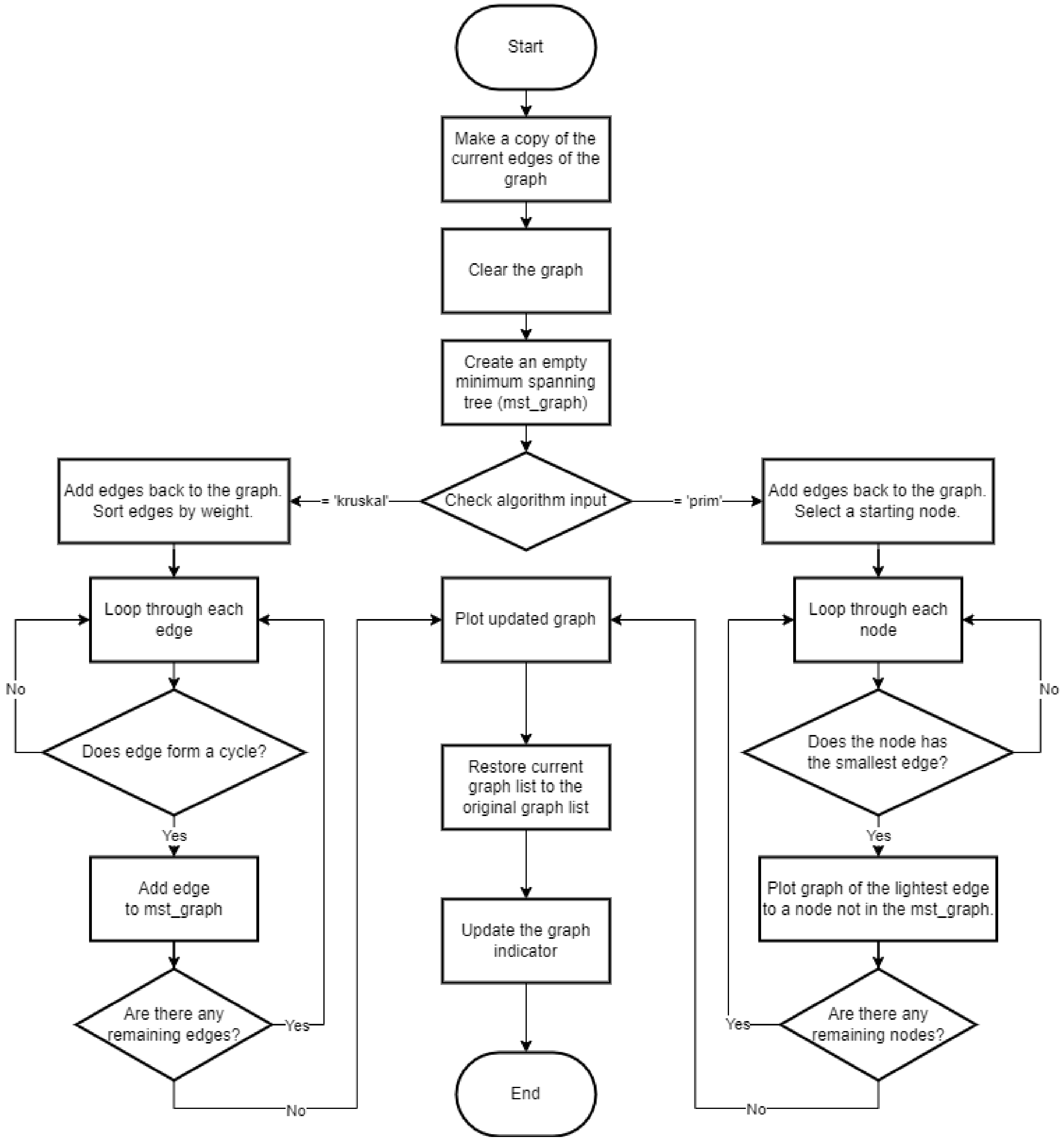
1. **NetworkX** is a robust library used to create, manipulate, and study the structure, dynamics, and functions of complex networks. In our tool, it plays a central role in performing all graph operations. This includes creating and managing graphs, adding nodes and edges, and implementing both Kruskal's and Prim's algorithms for Minimum Spanning Tree (MST) calculation.
2. **Tkinter**, on the other hand, is the standard Python interface to the Tk GUI toolkit. It offers a potent object-oriented interface to GUIs. We utilize Tkinter to design an intuitive and user-friendly interface where users can input graph data and observe the MST computation process and results.

The process of our tool unfolds through the following steps:

1. The tool first generates a GUI window using Tkinter. This window allows users to enter their graph details or opt for a randomly generated graph.
2. If the user chooses to manually input the graph, they input edges (node1, node2, weight) through the GUI interface. These edges are subsequently added to the graph structure maintained by NetworkX.
3. If the user opts for a randomly generated graph, the tool will randomly select nodes and edge weights and create a fully connected graph.
4. The user then selects the MST algorithm (either Kruskal or Prim) they wish to use for visualization purposes.
5. The selected algorithm is executed on the graph, and the process of constructing the MST is dynamically visualized on the GUI.
6. Upon completion, the resulting MST is displayed. The user can choose to add more edges, generate a new random graph, or run a different algorithm if they wish.

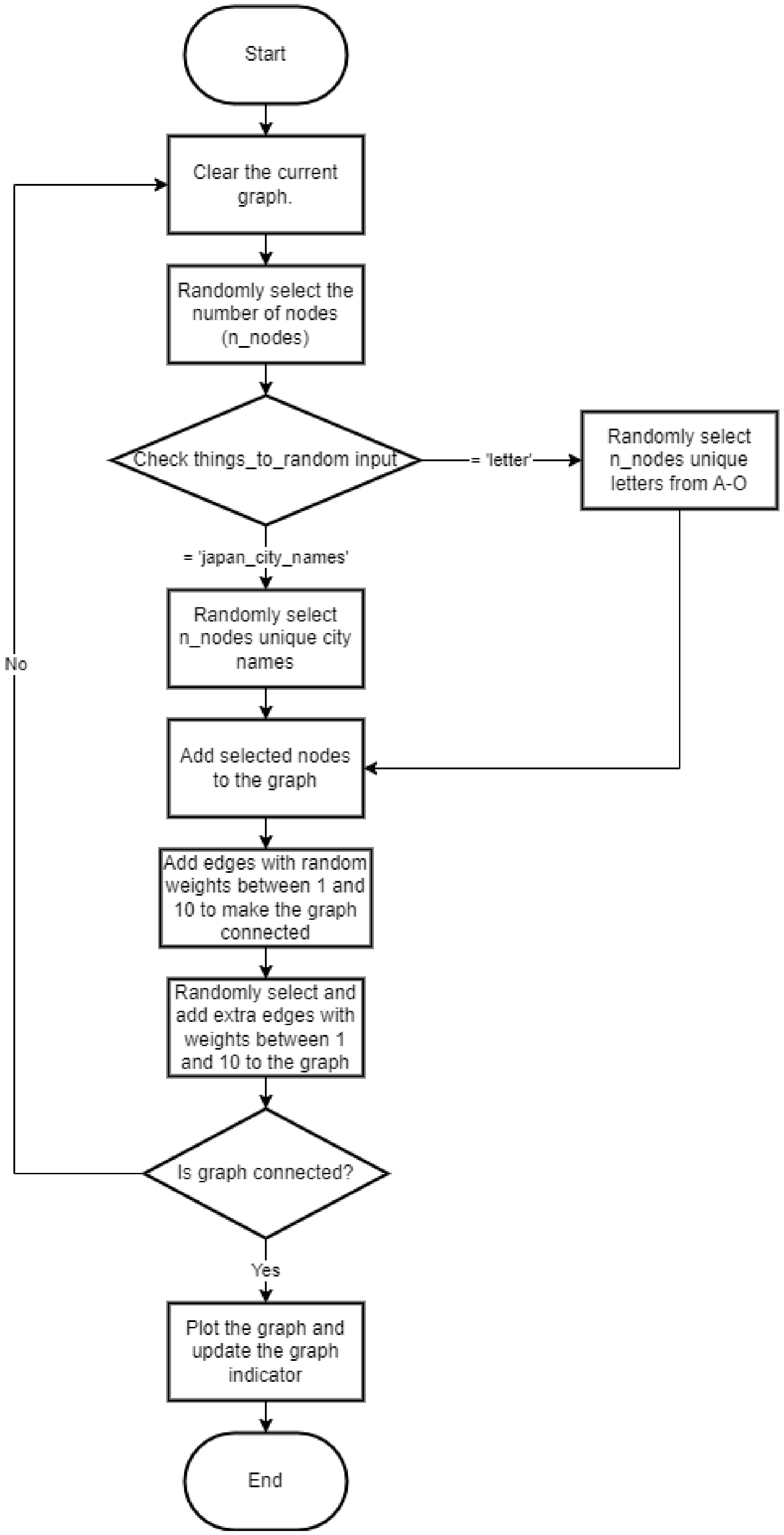
Flowchart

1. Kruskal and Prim's Algorithm



Start
Make a copy of the current edges of the graph
Clear the graph
Create an empty minimum spanning tree (mst_graph)
Check algorithm input
= 'kruskal'
= 'prim'
Add edges back to the graph. Sort edges by weight.
Add edges back to the graph. Select a starting node.
Loop through each edge
Does edge form a cycle?
No
Yes
Add edge to mst_graph
Are there any remaining edges?
Yes
No
Plot updated graph
Restore current graph list to the original graph list
Update the graph indicator
End
Loop through each node
Does the node has the smallest edge?
No
Yes
Plot graph of the lightest edge to a node not in the mst_graph.
Are there any remaining nodes?
Yes
No

2. Generating random graph



Start
Clear the current graph.
Randomly select the number of nodes (n_nodes)
Check things_to_random input
= 'letter'
Randomly select n_nodes unique letters from A-O
= 'japan_city_names'
Randomly select n_nodes unique city names
Add selected nodes to the graph
Add edges with random weights between 1 and 10 to make the graph connected
Randomly select and add extra edges with weights between 1 and 10 to the graph
Is graph connected?
No
Yes
Plot the graph and update the graph indicator
End

Pseudocode

1. Kruskal and Prim's Algorithm

```
function calculate_mst(algorithm):
    store a copy of the graph's edges in a variable "edges_copy"
    clear the current graph

    create an empty graph "mst_graph"

    if the algorithm is "kruskal":
        add the edges from "edges_copy" back to the graph
        sort the edges by weight in ascending order
        for each edge in the sorted edges:
            if adding the edge to "mst_graph" does not form a cycle:
                add the edge to "mst_graph"
                update and display the "mst_graph"
                pause for a moment before proceeding to the next edge

    else if the algorithm is "prim":
        add the edges from "edges_copy" back to the graph
        choose a starting node, typically the first node in the graph
        add the starting node to "mst_graph"
        while "mst_graph" has fewer nodes than the graph:
            find the minimum-weight edge that connects a node in "mst_graph" and a node not in "mst_graph"
            add the found edge to "mst_graph"
            update and display the "mst_graph"
            pause for a moment before proceeding to the next edge

    clear the graph and add the edges from "edges_copy" back to the graph
    update the graph connectivity indicator
```

2. **Generating random graph**

```
function random_graph(things_to_random):
    clear the current graph
    generate a random number of nodes between a minimum and maximum limit

    if things_to_random is 'letter':
        generate a list of unique random letters as node names
    else if things_to_random is 'japan_city_names':
        generate a list of unique random Japanese city names as node names

    until the graph is fully connected:
        clear the graph
        add the nodes to the graph
        connect each node to the next one with a random weight
        generate a random number of extra edges
        for each extra edge:
            choose two random nodes that are not already connected
            connect the chosen nodes with a random weight

    display the graph and update the graph connectivity indicator
```

Running Time

The time complexity of Kruskal’s and Prim’s algorithms differs based on the structures and characteristics of the graph they operate on.

Overall time complexity for Kruskal’s and Prim’s algorithms are O(E log E) and O((E+V) log V) respectively, where E is the number of edges and V is the number of vertices. Prim's algorithm performs well when it comes to dense graphs while Kruskal’s algorithm shines when it comes to sparse graphs since it uses a simpler data structure.

Both Kruskal’s and Prim’s algorithms have a time complexity of O(E log E), where E is the number of edges. This is because both algorithms require sorting of edges, which take O(E log E) amount of time to be executed. However, in terms of efficiency, Kruskal’s tends to be better for sparse graphs since it uses a simpler data structure, while Prim’s is more efficient for dense graphs.

Conclusion

The MST Visualization tool serves as a compelling and interactive resource for individuals aiming to understand the operational intricacies of Kruskal's and Prim's algorithms in constructing a Minimum Spanning Tree. By providing a detailed visualization of each step in the process, the tool enables users to effectively grasp these foundational graph theory concepts in action.

Successfully meeting its predetermined objectives, the project delivers an engaging and dynamic learning platform. It not only fosters comprehension of key theoretical ideas but also promotes interactive learning, enriching the user's grasp of these fundamental graph algorithms. In essence, the MST Visualization tool represents a significant stride in the realm of algorithmic learning tools.

Executable file download link

Version 1.0:

https://drive.google.com/file/d/1R5rQd9D2-bp4U-uEc7dTsKH-wmf8-bzc/view?usp=drive_link

Version 1.1:

https://drive.google.com/file/d/1MTNpcTC0UD0VMV09J79YMqJQ01pGAEsR/view?usp=sharing

Version 2.0:

https://drive.google.com/file/d/15u-Tg6aIrdUdc3qPF6z6Wr5KIO5RNPis/view?usp=sharing