Project Worksheet

**Student Name:** Yash Singh **UID:** 24MCI10030

**Student Name:**Vishvam Parihar **UID**:24MCI10010

**Branch:** MCA(AIML) **Section/Group:**24MAM1(A)

**Semester:** I **Date of Performance:**

**Subject Name:** DAA LAB **Subject Code:** 24CAP-612

* **Aim/Overview of the practical:**

To design and implement a Graph class in a programming language (such as Python, C++, or Java) that uses Kruskal's algorithm to find the Minimum Spanning Tree (MST) of a given weighted, undirected graph. The goal is to understand the workings of Kruskal's algorithm, how it efficiently finds the MST, and to deepen understanding of graph theory concepts like edge sorting, union-find structures, and tree formation

* **Task to be done:**

1. **Graph Class Structure**
2. **Edge Representation**
3. **Union-Find Implementation**
4. **Sorting Edges by Weight**
5. **Kruskal’s MST Method**

* **Programming languages used:**

**Python**:-

**Pros**: Simple syntax, easy to understand and implement algorithms, extensive libraries (like networkx for graph manipulation).

**Cons**: Slower execution speed compared to compiled languages like C++ and Java.

Block diagram:

[Graph Initialization]

|

v

[Add Edge]

|

v

[Sort Edges]

|

v

[Union-Find Initialization]

|

v

[Kruskal’s MST]

|

v

[Output MST]

**Pseudo Code:-**

Kruskal(graph):

Initialize an empty list MST

Create a union-find data structure for the graph

Sort all the edges in the graph by weight

for each edge (u, v, w) in sorted edges:

if Find(u) != Find(v): # If u and v are in different sets

MST.append((u, v, w)) # Add edge to MST

Union(u, v) # Union the sets

return MST

Code:-

import tkinter as tk

from tkinter import messagebox

from tkinter import ttk

# Graph class implementing Kruskal's MST algorithm

class Graph:

    def \_\_init\_\_(self, vertices):

        self.V = vertices

        self.graph = []

    def addEdge(self, u, v, w):

        if u >= self.V or v >= self.V or u < 0 or v < 0:

            raise ValueError(f"Vertices {u} and {v} are out of range for a graph with {self.V} vertices.")

        self.graph.append([u, v, w])

    def find(self, parent, i):

        if parent[i] != i:

            parent[i] = self.find(parent, parent[i])

        return parent[i]

    def union(self, parent, rank, x, y):

        if rank[x] < rank[y]:

            parent[x] = y

        elif rank[x] > rank[y]:

            parent[y] = x

        else:

            parent[y] = x

            rank[x] += 1

    def KruskalMST(self):

        result = []

        i = 0

        e = 0

        self.graph = sorted(self.graph, key=lambda item: item[2])

        parent = [node for node in range(self.V)]

        rank = [0] \* self.V

        while e < self.V - 1:

            u, v, w = self.graph[i]

            i += 1

            x = self.find(parent, u)

            y = self.find(parent, v)

            if x != y:

                e += 1

                result.append([u, v, w])

                self.union(parent, rank, x, y)

        minimumCost = sum([weight for u, v, weight in result])

        return result, minimumCost

# GUI Application

class KruskalApp:

    def \_\_init\_\_(self, root):

        self.root = root

        self.root.title("Kruskal's MST Algorithm")

        self.root.geometry("550x600")

        self.edge\_list = []

        # Title

        title\_label = tk.Label(root, text="Kruskal's MST Algorithm", font=("Helvetica", 18, "bold"), fg="#003366")

        title\_label.pack(pady=10)

        # Frame for Vertices and Edges input

        input\_frame = tk.Frame(root)

        input\_frame.pack(pady=10, padx=20, fill="x")

        # Number of Vertices

        tk.Label(input\_frame, text="Number of Vertices:", font=("Helvetica", 12)).grid(row=0, column=0, padx=5, pady=5)

        self.vertices\_entry = tk.Entry(input\_frame, font=("Helvetica", 12), width=5)

        self.vertices\_entry.grid(row=0, column=1, padx=5, pady=5)

        # Edge Input

        tk.Label(input\_frame, text="Edge (u, v, weight):", font=("Helvetica", 12)).grid(row=1, column=0, padx=5, pady=5)

        self.u\_entry = tk.Entry(input\_frame, font=("Helvetica", 12), width=5)

        self.u\_entry.grid(row=1, column=1, padx=5, pady=5)

        self.v\_entry = tk.Entry(input\_frame, font=("Helvetica", 12), width=5)

        self.v\_entry.grid(row=1, column=2, padx=5, pady=5)

        self.w\_entry = tk.Entry(input\_frame, font=("Helvetica", 12), width=5)

        self.w\_entry.grid(row=1, column=3, padx=5, pady=5)

        # Buttons to add edge and clear edges

        button\_frame = tk.Frame(root)

        button\_frame.pack(pady=10)

        add\_edge\_button = tk.Button(button\_frame, text="Add Edge", command=self.add\_edge, bg="#4CAF50", fg="white", font=("Helvetica", 12))

        add\_edge\_button.grid(row=0, column=0, padx=10)

        clear\_edges\_button = tk.Button(button\_frame, text="Clear Edges", command=self.clear\_edges, bg="#f44336", fg="white", font=("Helvetica", 12))

        clear\_edges\_button.grid(row=0, column=1, padx=10)

        # Edge List Display

        self.edge\_list\_display = tk.Text(root, height=8, width=40, font=("Helvetica", 12))

        self.edge\_list\_display.pack(pady=10)

        self.edge\_list\_display.insert(tk.END, "Edges:\n")

        self.edge\_list\_display.config(state="disabled")

        # Calculate MST Button

        calculate\_mst\_button = tk.Button(root, text="Calculate MST", command=self.calculate\_mst, bg="#2196F3", fg="white", font=("Helvetica", 14))

        calculate\_mst\_button.pack(pady=15)

        # Result Display

        self.result\_label = tk.Label(root, text="", font=("Helvetica", 12), fg="#003366")

        self.result\_label.pack(pady=10)

    def add\_edge(self):

        try:

            u = int(self.u\_entry.get())

            v = int(self.v\_entry.get())

            w = int(self.w\_entry.get())

            vertices = int(self.vertices\_entry.get()) if self.vertices\_entry.get().isdigit() else 0

            if u >= vertices or v >= vertices or u < 0 or v < 0:

                messagebox.showerror("Invalid Edge", f"Vertices {u} and {v} are out of range for a graph with {vertices} vertices.")

                return

            self.edge\_list.append((u, v, w))

            self.display\_edges()

            self.u\_entry.delete(0, tk.END)

            self.v\_entry.delete(0, tk.END)

            self.w\_entry.delete(0, tk.END)

        except ValueError:

            messagebox.showerror("Invalid Input", "Please enter valid integer values for u, v, and weight.")

    def clear\_edges(self):

        self.edge\_list = []

        self.display\_edges()

    def display\_edges(self):

        self.edge\_list\_display.config(state="normal")

        self.edge\_list\_display.delete(1.0, tk.END)

        self.edge\_list\_display.insert(tk.END, "Edges:\n")

        for u, v, w in self.edge\_list:

            self.edge\_list\_display.insert(tk.END, f"{u} -- {v} == {w}\n")

        self.edge\_list\_display.config(state="disabled")

    def calculate\_mst(self):

        try:

            vertices = int(self.vertices\_entry.get())

            self.graph = Graph(vertices)

            for u, v, w in self.edge\_list:

                self.graph.addEdge(u, v, w)

            mst, minimumCost = self.graph.KruskalMST()

            result\_text = "Edges in the MST:\n"

            for u, v, weight in mst:

                result\_text += f"{u} -- {v} == {weight}\n"

            result\_text += f"\nMinimum Cost of MST: {minimumCost}"

            self.result\_label.config(text=result\_text)

        except ValueError:

            messagebox.showerror("Invalid Input", "Please enter a valid number of vertices.")

        except IndexError:

            messagebox.showerror("MST Error", "Ensure the graph has enough edges to form a spanning tree.")

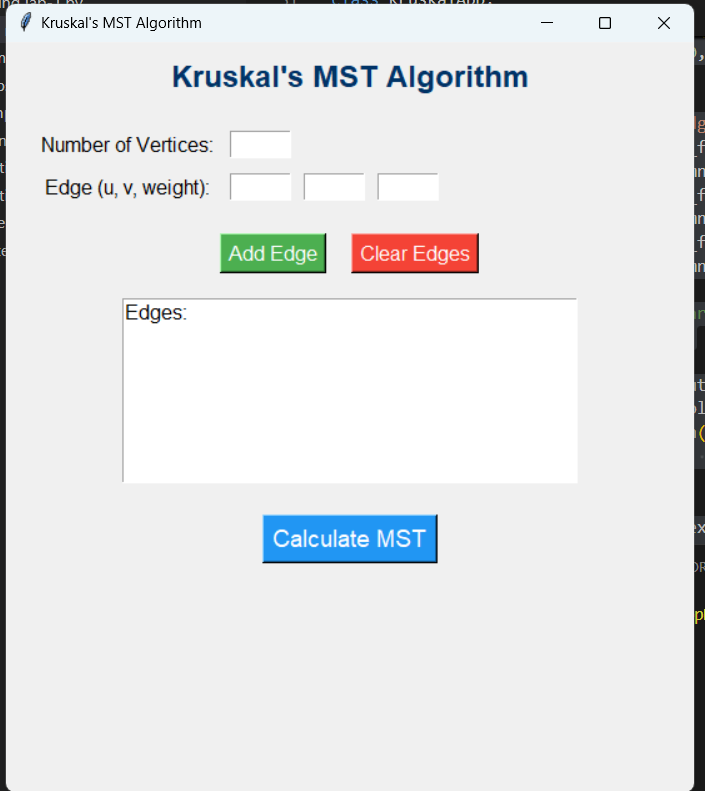
# Run the app

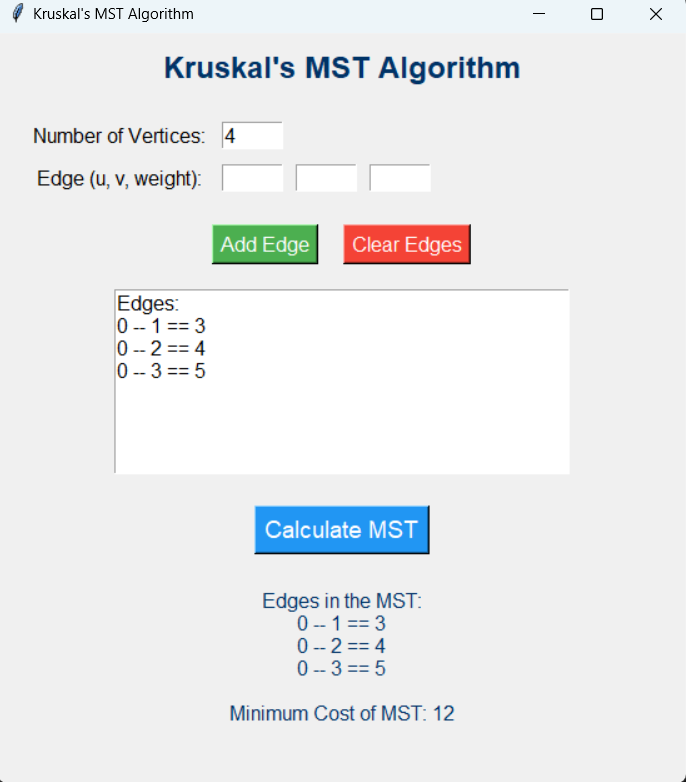
root = tk.Tk()

app = KruskalApp(root)

root.mainloop()

Output:-





Learning Outcomes:-

 **Understanding Graph Theory**:

* Gain a foundational understanding of graph theory concepts, including vertices, edges, and weighted graphs.
* Learn about different types of spanning trees and their properties.

 **Kruskal's Algorithm**:

* Comprehend the steps and logic behind Kruskal's algorithm for finding the Minimum Spanning Tree.
* Analyze how the algorithm leverages sorting and union-find data structures to efficiently construct the MST.

 **Data Structures**:

* Familiarize oneself with essential data structures such as lists, arrays, and the union-find structure, including path compression and union by rank techniques.

 **Algorithm Implementation**:

* Develop skills in implementing algorithms programmatically in a chosen programming language (e.g., Python).
* Enhance debugging and problem-solving skills by resolving issues that arise during the implementation.

 **Graph Representation**:

* Learn different ways to represent graphs in code, particularly edge lists and adjacency lists, and understand their respective advantages and disadvantages.

 **GUI Development**:

* Acquire knowledge in developing user-friendly graphical user interfaces (GUIs) using libraries such as Tkinter.
* Understand event-driven programming concepts and how to handle user inputs effectively.

Project Links and References –

* https://github.com/Mrhacker6/DAA