Exercise 3: Good 'ol Kruskal

ENSF 338 - Lab 8

Group 12

1. What is a Minimum Spanning Tree (MST)?

A **minimum spanning tree (MST)** of a connected, weighted, undirected graph is a subset of the edges that:

- Connects *all* the vertices together (i.e., it is a spanning tree).
- Has the *minimum possible total edge weight.*
- Contains no cycles (it is a tree).

In other words, if we consider all possible spanning trees of the original graph, the MST is one whose sum of edge weights is as small as possible.

2. Example with a Graph of 5 Nodes

Below is a small graph with 5 nodes: A, B, C, D, and E. The edges have the following weights (for instance):

$$A - B = 2, \quad A - C = 3, \quad B - C = 2, \quad B - D = 4,$$
$$C - E = 6, \quad D - E = 5, \quad C - D = 3 \quad \ldots$$

Full Graph

(*If you don't have an external image, you can draw the graph in Overleaf/Google Docs, or replace this figure with a text-based ASCII diagram.*)

Chosen MST

One possible MST for this graph is formed by the edges:

$$\{\, A - B,\, B - C,\, C - D,\, D - E \}.$$

Assuming the weights are $2, 2, 3, 5$ respectively, the total cost is $2 + 2 + 3 + 5 = 12$.

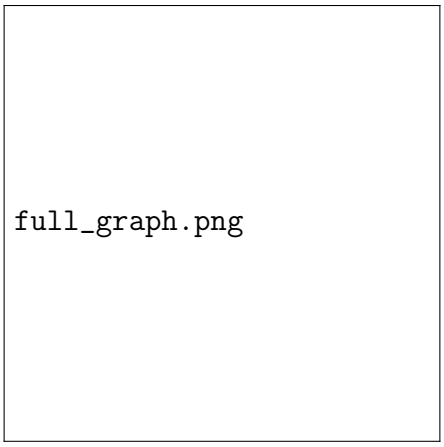

Figure 1: Full graph with 5 nodes and various weighted edges. (Insert your own diagram.)

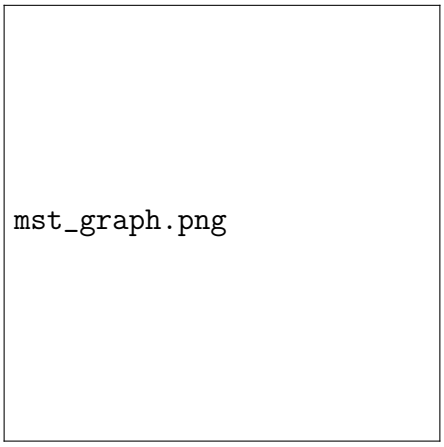

Figure 2: Highlighted MST edges (in bold or color). (Insert your own diagram.)

3. Union-Find and Kruskal's MST Algorithm

3.1. UNION-FIND Data Structure

To efficiently detect cycles while building the MST, we use the **Union-Find** (also known as Disjoint Set) data structure. Key operations:

- `make_set(x)`: initializes a new set containing element x.
- `find(x)`: returns the representative (root) of the set that x belongs to.
- `union(x, y)`: merges the sets containing x and y (if they are different).

3.2. Kruskal's Algorithm

1. Sort all edges in order of **increasing weight**.
2. Initialize Union-Find structure so each node is in its own set.

3. Iterate over edges from smallest to largest weight:
    - If an edge connects two nodes that are in **different** sets (i.e., `find(u) != find(v)`), add the edge to the MST and `union` their sets.
    - Otherwise (if they are in the same set), adding the edge would create a cycle, so skip it.
4. Continue until you have $n - 1$ edges in the MST (where n is the number of nodes).

Because the union-find structure quickly tells us whether adding an edge will form a cycle, Kruskal's algorithm can efficiently find the MST.

4. Implementation and Sample Usage

We provide a Python implementation in `ex3.py`, which includes:

- A **UnionFind** class for cycle detection.
- An **mst()** method inside a **MSTGraph** class (extended from `Graph`) that:
    1. Sorts edges by weight
    2. Uses Union-Find to build the MST
    3. Returns a new `Graph` object that contains only the MST edges

Example usage:

```
from ex3 import MSTGraph

g = MSTGraph()
g.importFromFile("random.dot")

mst_g = g.mst()
print("MST edges:")
print(mst_g)
```

5. Conclusion

In this exercise, we:

- Learned about MSTs and why they're useful.
- Implemented **Kruskal's algorithm** using a **Union-Find** data structure.
- Constructed and tested an MST for a sample graph and a random input (`random.dot`).

End of ex3.pdf