Making Friends Report

Adam Tovatt

May 7, 2025

1 Results

All test cases passed when running the provided `check_solution.sh` script. The program successfully solved even the largest test cases with 100,000 nodes and 3000,000 edges in around a second on a modern machine. The majority of the time is spent sorting the edges at the start of Kruskal's algorithm, which has a time complexity of $O(M \log M)$. For example, for the largest input the time spent on different parts of the solution is as follows:

Section	**Time (ms)**	**Percent of Total**
readInput	633	44.8%
sortConnections	755	53.4%
createTree	24.5	1.7%
fullSolve	1413	100.0%

2 Implementation details

I implemented Kruskal's algorithm to compute the Minimum Spanning Tree (MST). The core data structures are:

- A list of edges (called `Connection`), each containing two node IDs and a weight.
- A `UnionFind` data structure that uses path compression and union by rank to efficiently track connected components.

The algorithm works by sorting all edges by weight, and then iteratively connecting nodes if they are not already in the same group (i.e., have different roots in the union-find structure). This guarantees the MST is built with the minimal total weight and no cycles.

The overall running time is $O(M \log M)$, where M is the number of edges.

Discussion Questions

Why does the algorithm produce an MST?
Kruskal's algorithm always picks the smallest available edge that doesn't create a cycle. This ensures minimal total weight and guarantees all nodes are connected without cycles — which defines a Minimum Spanning Tree.

What is the time complexity, and why?
The time complexity is $O(M \log M)$. This is because fastest possible sorting algorithm works works by repeatedly splitting the list in the middle, which takes $\log M$ steps, and at each step it processes all M edges — so the total work is $M \cdot \log M$. Sorting takes the majority of the time and the union-find operations are nearly constant time due to path compression and union by rank. This leaves us with $O(M \log M)$.

What happens if an included edge collapses in real applications?
The network becomes disconnected. This is because the tree is already made to be the most minimal amount of edges possible, if an edge dissappears the the tree doesn't work anymore. In practical systems, this could break communication or connectivity. To handle this, real systems should probably use redundant connections or dynamic algorithms to recompute an MST.

What are real applications of MST?
Examples include network design (minimizing cost of wiring or cables), clustering, and road planning. For MST to be the right solution, the problem must involve connecting all items at minimal total cost with no cycles required.
Simply put: any scenario where minimizing "cost" or "distance" between points that are connected in a way such that there is exactly one unique path between any two nodes is a good place to use MST.