CSCI311-cityConnectionProject-Team1

Team 1
Cole Zehe, Kyle Light, Tung Tran, Jake Etzler

Dec 2, 2021

1 Introduction:

Minimum Spanning Tree is one of most well-known, classical problem in computer science community. This problem is attributed to Kruskal and Prim in 1950s as its first efficient solutions, but its first appearance can date back to Boruvka in 1926. The problem is defined as follows: Given a graph $G = (V, E, w)$. $T \subseteq E$ is a spanning tree of G if (V, T) is a tree. The function $w : E \rightarrow$ indicates the weight for each edge. The MST is then a spanning tree whose weight is minimum among all spanning tree.

Our task is to fully connect the city by laying the cable along roads in a way that it connects to every road intersection. This problem is equivalent to solving an MST with road intersections for nodes and roads for edges.

We discuss three types of algorithms: our solution, Kruskal's and Prim's in order to solve this problem. The following reports the approaches and empirical analyses of the latter two more efficient algorithm. The code used in our experiments are published on https://github.com/colezehe/CSCI311$_{C}ity_{C}onnection$

2 Methods:

Generally, the MST problem exhibits optimal substructures: olution of a graph can be found from MST solutions of smaller subgraphs.

Furthermore, solving MSTs can also lead to overlapping subprolems. This is since MST substructures can be identical, leading to computational waste. Obviously, with these two mentioned properties, dynamic programming solutions exist. We'll discuss one such solution in detail in subsection 2.1.

Most notably, MSTs have the greedy choice property. Starting from anywhere, suppose (u, v) is the lightest edge crossing some cut S with $w(u, v) = s$. Then $P \cup (u, v)$ is part of some MST T, where P is a partial MST contained in S. This allows for the design of efficient algorithms such as Prim's and Kruskal's.

2.1 Dynamic Programming solution:

Originally, without the knowledge of the MST's greedy choice, the problem can be solved by comparing with the recursive structure: $MST(G(V, E)) = min_{v \in V}(\textit{cost to connect } v \textit{ with } MST(G'(V \setminus \{v\}), E))$, skipping over v's that are disconnected from $V \setminus \{v\}$. Suppose we're using Adjacency Matrix to index edges. Since for each subset S we need to compute the product $|S| - 1$ times the amount of computation for subsets of S with size $|S - 1|$. This implies a and, which implies a solution of time complexity $O(|V|!)$
However, since many subgraphs with the set of vertices $V \setminus \{v\}$ share the same subproblems $(V \setminus \{v\} \bigcap V1 \setminus \{w\} = V \setminus \{\})$.
We propose using memoization by storing MST solutions to a table of size $2^{|}V|$. This way, we'll only need to compute each subset S using $|S| - 1$ subsets of S of size $|S| - 1$, implying a solution of time complexity $T(|V|) = O(\sum_{S \subset V} |S| - 1) = O(\sum_i (i - 1)C(i, |V|)) = O(|V|2^{|}V|) < O(|V|!)$.

2.2 HashSet:

In the implementation of our algorithms, we also use different data structures. The HashSet data structure for this project is a basic HashSet that has been optimized for implementation with Prim's algorithm. We decided on this data structure because both the ADD(X) and the CONTAINS(X) methods have a constant runtime of O(1). Normally, a hash set would have a REMOVE(X) method as well, however this method would be unused during the run time of Prim's, so to further optimize the data structure this method has not been included. On top of all of that, this is a data structure that we were all familiar with from Java, so it was an easy choice to implement this structure. The only problem with this data structure is that we've implemented it using Python. Being Python, this implementation of HashSet is hindered by the native slow speeds.

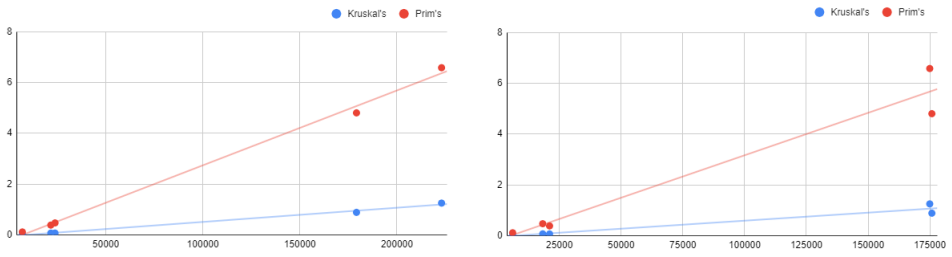

Figure 1: Comparison between Prim's and Kruskal's w.r.t the number of edges (left) and number of vertices (right) on the 5 datasets.

2.3 Priority Queue:

We used the data structure Priority Queue to keep track of minimum weight edges in Prim's algorithm. In our implementation, two types of Priority Queues were used. Our first type simply stored all edges then search for minimum edge in linear time. Our other implementation of Priority Queue used Binary Heap, which reduced the time complexity of the POP() operation from $O(n)$ to $O(log\ n)$.

2.4 Graph representations:

We implemented multiple representations of input graph. A total of 3 were used: simple list, adjacency matrix and adjacency list. Lists were simple to implement but can cost linear time complexity to find elements. Adjacency matrix has the most efficient search time asymtotically or real time, but has a high space complexity overhead of $O(|V|^2)$. Adjacency lists had a worst case of $O(n)$, but in Prim's algorithm implementation, it only costs an amortized $O(1)$, making it highly suited for Prim's algorithm.

2.5 Prim's:

Having established the supporting data structures, we now go into the actual solutions to MST problem. The expected, theoretical time for Prim's implemented with a Priority Queue is $O((|V| + |E|)log|V|)$. To extract and insert all of the nodes would take $|V|log|V|$. It takes $|E|log|V|$ to update our distance. In real world application, our algorithm takes a bit longer than this. This is due to our use of Python, which is a naturally slower language, and our importation of other files that we have written.

2.6 Kruskal's:

Kruskal's idea was simple: always add the lightest edge in the graph, though we need to check for cycles. In our implementation, we used disjoint sets, where each set is a collection of connected nodes. Since both checking and adding an edge has the time complexity of $O(log * n)$, the algorithm had a total runtime of $O(|E|log|E|) + O(|E|log * |V|) = O(|E|log|V|)$. This is since $log|E|$ and $log|V|$ are asymtotically equivalent as $E \leq |V|^2$.

2.7 Comparison:

Asymtotically, in our implementation, Kruskal's algorithm was better than Prim's algorithm ($O((|V| + |E|)log|V|) > O(|E|log|V|)$). This is also reflected in our real-world runtimes on 5 different dataset: California, City of Oldenburg, San Francisco, San Joaquin, and North America (**Fig.** 1)

3 Conclusion:

In this paper, we discussed the methods to solve the MST problem, which in turn solves the problem of connecting the city with internet cable. We demonstrated theoretically and empirically that Kruskal's algorithm is a better match for this problem than Prim's.