Minimum Cost Spanning Trees

- Let $G = (V, E)$ be an edge weighted (undirected) graph, with a set of $n$ vertices and $m$ edges.

- Graph $G$ is connected iff there is a path between every pair of vertices.

  ![Connected](image1.png) ![Not Connected](image2.png)

- $H$ is a subgraph of $G$ iff the vertices and edges in $H$ are a subset of those in $G$.

- Cycle: Simple path with the same start and end vertex.

- Tree: Connected undirected graph with no cycles.

- Spanning tree of $G$: Subgraph of $G$ containing all the vertices in $G$ that forms a tree.
Min Cost Spanning Tree

Given a connected edge weighted graph, find a spanning tree such that the sum of the cost (weight) of the edges in it is least possible.

Edge Weighted Graph

Min Cost Spanning Tree
Kruskal’s Algorithm

Greedy Criterion: From the remaining edges, select a least cost edge that does not result in a cycle when added to the set of already selected edges.

Example

Edge Weighted Graph

Min Cost Spanning Tree
Kruskal’s Algorithm

// Find a min cost spanning tree of an n-vertex edge weighted (undirected) connected graph.

// K: Set of edges in MCST
K = Empty Set
Let E be the set of edges
while |K| not= n-1 do
    { Let \{u,v\} be a least cost edge in E
      E = E - \{u,v\}
      if \{u,v\} does not create a cycle in K
         then K = K Union \{u,v\}
    }

// K is a MCST
end
Correctness

Theorem: The spanning tree generated by Kruskal’s algorithm is a MCST.

- Clearly, $K$ constructed by the algorithm is a spanning tree.
- We now show that no matter which MCST $O$ we start with, it is possible to transform $O$ into $K$ without increasing its cost (weight).
  
  - Clearly, $|K| = |O| = n - 1$
  
  - If $K = O$ then there we are done as $K$ is a MCST.
  
  - Assume that $K \neq O$.
  
  - For some $1 \leq i \leq n - 1$ there are $i$ edges in $K$ that are not in $O$, and $i$ edges in $O$ that are not in $K$.
  
  - Perform the following transformation $i$ times.
Additional Notation

- Let $e$ be a least cost edge in $K$ that is not in $O$
- Let $f$ be any edge in the cycle $O \cup \{e\}$ that is not in $K$.

\[
\begin{array}{c}
\text{\includegraphics[width=0.5\textwidth]{additional_notation_diagram}}
\end{array}
\]

- Define $N = O - \{f\} \cup \{e\}$
- Clearly, $N$ is a spanning tree.
- We now prove that cost $f = \text{cost } e$.
- This implies that Cost $N = \text{Cost } O$
- Proof by contradiction (Cost $f \neq \text{Cost } e$)
Case 1: Cost $f \ > \ Cost \ e$

- Cost $f \ > \ Cost \ e$
- Implies Cost $O \ > \ Cost \ N$
- Implies that $O$ is not a MCST
- A Contradiction.
Case 2: Cost $f < \text{Cost } e$

- Edge $f$ was considered before edge $E$ by Kruskal algorithm.
- Since $f$ is not in $K$ then it must have been that Kruskal’s algorithm discarded it.
- So we know that edge $f$ together with all edges in $K$ with cost $\leq \text{Cost } f$ formed a cycle.
- But, all of these edges are also in $O$ because the way we selected edge $e$ (i.e., edge $e$ is the least cost edge in $K$ that is not in $O$).
- So MCST $O$ has a cycle.
- A Contradiction.
Therefore, Cost $f = \text{Cost } e$

- Implies that Cost $N = \text{Cost } O$
- So, $N$ is a MCST.
- So now there are only $i - 1$ edges in $K$ that are not in $N$ and $i - 1$ edges in $N$ that are not in $K$.
- Repeat this process with $O = N$ for a total of $i$ times
- The last time we apply this transformation $N$ will be $K$.
- This implies that $K$ is a MCST because its cost is equal to the cost of $O$ which is a MCST.
- End of Proof.
Implementation

- Graph $G$ is represented by adjacency lists ($n$ vertices and $m$ edges)
- Implementation using the code provided
- Edges organized in a min heap (ordered by their weight (cost))
- Detection of cycles is tricky!
Implemented via *Disjoint Set Union (Union-Find Operations)*. Each tree is a set.
Algorithm

// Find a min cost spanning tree of an n-vertex edge weighted (undirected) connected graph.
// K: Set of edges in MCST
K = Empty Set
Let E be the set of edges
Keep E in min heap H (ordered by weight)
initialize set i to {i} (for 1 <= i <= n)
while | K | not= n-1 do
  { Delete min edge from (H) and
    let {u,v} be that edge
   uset = find(u)
   vset = find(v)
    if(uset not= vset)
      {K = K Union {u,v}
       Union(uset,vset)
      }
  }
// K is a MCST
end
Time Complexity

- Building a heap of edges takes $O(m)$ time as there are $m$ edges.
- While loop is repeated at most $m$ times.
- Each time delete min takes $O(\log m)$ time.
- Total cost of union-find is *almost linear*.
- This implies that time complexity is $O(m \log m)$ which is also $O(m \log n)$ because $m < n^2$. 