Minimum Spanning Trees

Based on Mahesh Viswanathan’s notes
MST: the problem

- Input: Connected graph $G = (V, E)$ with edge costs
- Goal: find $T \subseteq E$ such that $(V, T)$ is connected and the total cost of all edges in $T$ is the smallest
  - $T$ is called the Minimum spanning tree of $G$
Greedy Template

$T$ is empty (*$T$ will store edges of a MST*)

While $T$ is not spanning tree yet (*$|T| \leq |V| - 1$*)

select $e \in E$ to add to $T$ according to a greedy criterion

Return $T$

How should we choose the edge to add to $T$?
Kruskal’s algorithm

Process edges in the order of their costs (starting from the least) and add edges to $T$ as long as they don’t form a cycle.

Figure: Graph $G$

Figure: MST of $G$
Kruskal’s algorithm

Process edges in the order of their costs (starting from the least) and add edges to $T$ as long as they don’t form a cycle.

Figure: Graph $G$

Figure: MST of $G$
Kruskal’s algorithm

Process edges in the order of their costs (starting from the least) and add edges to $T$ as long as they don’t form a cycle.

Figure: Graph $G$

Figure: MST of $G$
Kruskal’s algorithm

Process edges in the order of their costs (starting from the least) and add edges to $T$ as long as they don’t form a cycle.

Figure: Graph $G$

Figure: MST of $G$
Kruskal’s algorithm

Process edges in the order of their costs (starting from the least) and add edges to $T$ as long as they don’t form a cycle.
Kruskal’s algorithm

Process edges in the order of their costs (starting from the least) and add edges to $T$ as long as they don’t form a cycle.

Figure: Graph $G$

Figure: MST of $G$
Prim’s algorithm

$T$ maintained by algorithm will be a tree. In each iteration, pick edge with least attachment cost to $T$.

Figure: Graph $G$

Figure: MST of $G$
Prim’s algorithm

$T$ maintained by algorithm will be a tree. In each iteration, pick edge with least attachment cost to $T$.

Figure: Graph $G$

Figure: MST of $G$
Prim’s algorithm

The tree maintained by the algorithm will be a tree. In each iteration, pick the edge with the least attachment cost to $T$.

Figure: Graph $G$

Figure: MST of $G$
Prim’s algorithm

\( T \) maintained by algorithm will be a tree. In each iteration, pick edge with least attachment cost to \( T \).

Figure: Graph \( G \)

Figure: MST of \( G \)
Prim’s algorithm

$T$ maintained by algorithm will be a tree. In each iteration, pick edge with least attachment cost to $T$.

**Figure:** Graph $G$

**Figure:** MST of $G$
Prim’s algorithm

$T$ maintained by algorithm will be a tree. In each iteration, pick edge with least attachment cost to $T$. 

Figure: Graph $G$

Figure: MST of $G$
Prim’s algorithm

$T$ maintained by algorithm will be a tree. In each iteration, pick edge with least attachment cost to $T$.

Figure: Graph $G$

Figure: MST of $G$
Correctness

• For now, we will assume that all edge costs are distinct
Cut property

Lemma

• Let $S \subset V (\neq \emptyset \text{ and } \neq V)$. Let $e = (v, w)$ be the minimum cost edge with one end in $S$ and the other end in $V \setminus S$. Then every MST contains $e$.

Let’s try this proof:

• Suppose (for contradiction) that $e$ is not in MST $T$

• Since $T$ is connected (being a tree), there must be some edge $f$ with one end in $S$ and the other in $V \setminus S$

• Since $e$ is cheaper than $f$, $T' = (T \setminus \{f\}) \cup \{e\}$ is a cheaper spanning tree

\textit{T'} might not be a spanning tree
Error in proof:

Figure: Problematic example. $S = \{1, 2, 7\}$, $e = (7, 3)$, $f = (1, 6)$
Proof of cut property

Proof.

Suppose minimum \((S, V \setminus S)\)-cut edge \(e = (v, w)\) is not in MST \(T\).
Proof of cut property

Proof.

- Suppose minimum $(S, V \setminus S)$-cut edge $e = (v, w)$ is not in MST $T$.
- Since $T$ is connected, there is some path (say $P$) from $v$ to $w$ in $T$.
Proof of cut property

Proof.

1. Suppose minimum \((S, V \setminus S)\)-cut edge \(e = (v, w)\) is not in MST \(T\).
2. Since \(T\) is connected, there is some path (say \(P\)) from \(v\) to \(w\) in \(T\).
3. Let \(w'\) be the first vertex in \(P\) belonging to \(V \setminus S\); let \(v'\) be the vertex just before it on \(P\), and let \(e' = (v', w')\).
Proof of cut property

Proof.

- Suppose minimum $(S, V \setminus S)$-cut edge $e = (v, w)$ is not in MST $T$.
- Since $T$ is connected, there is some path (say $P$) from $v$ to $w$ in $T$.
- Let $w'$ be the first vertex in $P$ belonging to $V \setminus S$; let $v'$ be the vertex just before it on $P$, and let $e' = (v', w')$.
- $T' = (T \setminus \{e'\}) \cup \{e\}$ is spanning tree of lower cost.
Show that $T'$ is a spanning tree

- $T'$ is connected
  - If path uses $e' = (v', w')$, then go from $v'$ to $v$, use edge $(v, w)$, then go from $w$ to $w'$

- $T'$ is acyclic
  - $T \cup \{e\}$ has only one cycle, one that involves both $e$ and $e'$
  - Removing $e'$ removes the cycle
Correctness of the greedy algorithms

• Prim’s algorithm
  – Pick edge with minimum attachment cost to current tree, and add to current tree.

• Proof of correctness
  – If $e$ is added to $T$, it belongs to every MST
    • Let $S$ be the vertices connected by edges in $T$ when $e$ is added
    • $e$ is the lowest cost edge with one end in $S$ and the other in $V \setminus S$.
  – Set of edges forms a spanning tree
    • No cycles are created because we always add an edge from an unattached vertex
Prim’s algorithm

$T$ maintained by algorithm will be a tree. In each iteration, pick edge with least attachment cost to $T$.

Figure: Graph $G$

Figure: MST of $G$
Prim’s algorithm

The set $T$ maintained by the algorithm will be a tree. In each iteration, pick an edge with least attachment cost to $T$.

Figure: Graph $G$

Figure: MST of $G$
Prim’s algorithm

$T$ maintained by algorithm will be a tree. In each iteration, pick edge with least attachment cost to $T$.

Figure: Graph $G$

Figure: MST of $G$
Prim’s algorithm

$T$ maintained by algorithm will be a tree. In each iteration, pick edge with least attachment cost to $T$.

**Figure:** Graph $G$

**Figure:** MST of $G$
Prim’s algorithm

The set $T$ maintained by the algorithm will be a tree. In each iteration, pick the edge with the least attachment cost to $T$.

Figure: Graph $G$

Figure: MST of $G$
Prim’s algorithm

The tree $T$ maintained by the algorithm will be a tree. In each iteration, pick the edge with the least attachment cost to $T$.

Figure: Graph $G$

Figure: MST of $G$
Prim’s algorithm

$T$ maintained by algorithm will be a tree. In each iteration, pick edge with least attachment cost to $T$.

**Figure:** Graph $G$

**Figure:** MST of $G$
Correctness of the greedy algorithms

• Kruskal’s
  – Pick edge of lowest cost and add if it does not form a cycle with existing edges

• Proof of correctness:
  – When an edge $e = (v, w)$ is added, let $S$ be the connected components in $T$ that contains $v$
  – $e$ is the minimum cost edge with one end in $S$ and one end in $V \setminus S$
Kruskal’s algorithm

Process edges in the order of their costs (starting from the least) and add edges to $T$ as long as they don’t form a cycle.

Figure: Graph $G$

Figure: MST of $G$
Kruskal’s algorithm

Process edges in the order of their costs (starting from the least) and add edges to $T$ as long as they don’t form a cycle.

Figure: Graph $G$

Figure: MST of $G$
Kruskal’s algorithm

Process edges in the order of their costs (starting from the least) and add edges to $T$ as long as they don’t form a cycle.

Figure: Graph $G$

Figure: MST of $G$
Kruskal’s algorithm

Process edges in the order of their costs (starting from the least) and add edges to $T$ as long as they don’t form a cycle.

Figure: Graph $G$

Figure: MST of $G$
When edge costs are not distinct

• Order edges lexicographically to break ties
• Both Prim’s, kruscal’s are optimal with respect to the lexicographical ordering
Implementation & Data structure
Prim’s algorithm

- Idea: Pick edge with minimum attachment cost to current tree, and add to current tree

Implementing Prim’s

- \( E \) is the set of all edges in \( G \)
- randomly pick a node \( u_0 \) and \( S = \{u_0\} \)
- \( T \) is empty (*\( T \) stores edge of a MST*)
- while \( S \neq V \)
  - pick \( e = (v, w) \in E \) such that
    - \( v \in S, w \notin S \)
    - \( e \) has minimum cost
  - \( T = T \cup e; S = S \cup w \)
- return set \( T \)

Analysis:
- Number of iterations: \( O(|V|) \)
- Picking \( e \) in each iteration is \( O(|E|) \)
- Total time \( O(|V||E|) \)
More efficient implementation

procedure prim(G, w)
Input: A connected undirected graph $G = (V, E)$ with edge weights $w_e$
Output: A minimum spanning tree defined by the array prev

for all $u \in V$:
    cost($u$) = $\infty$
    prev($u$) = nil
Pick any initial node $u_0$
cost($u_0$) = 0

$H =$ makequeue($V$) (priority queue, using cost-values as keys)
while $H$ is not empty:
    $v =$ deletemin($H$)
    for each $\{v, z\} \in E$:
        if cost($z$) > $w(v, z)$:
            cost($z$) = $w(v, z)$
            prev($z$) = $v$
            decreasekey($H, z$)
    $S = S \cup \{v\}$, $T = T \cup (prev(v), v)$

Analysis: $O((|V| + |E|) \log |V|)$ - same as Dijkstra’s
- each node is inserted and deleted once from the priority queue ($O(|V| \log |V|)$)
- Each edge is checked once, leading one possible decreasekey ($O(|E| \log |V|)$)
Kruskal’s algorithm

• Idea: pick edge of lowest cost and add if it does not form a cycle with existing edges

Implementing Kruskal’s

sort the edges in $E$ in increasing order based on cost
$T$ is empty (*$T$ stores edge of a MST*)
for each edge $e$ in sorted order
  if $T \cup e$ does not creates cycle
    $T = T \cup e$

return set $T$

Analysis:

• Sorting the edges: $O(|V|\log |V|)$
• For loop executes $O(|E|)$ times
• Each time, deciding if adding an edge $e = (u, v)$ leads to cycle:
  – Run DFS or BFS on $T$ to see if $u$ and $v$ are connected $O(|V| + |E|)$
• Total time $O(|V| \log |V| + |E|(|V| + |E|))$
Efficient Implementation of Kruskal’s

- Use Union-by-rank data structure to maintain a set of disjoint sets
- Each set contains the nodes of a particular connected component
- Initially each node is in a component by itself

```plaintext
procedure kruskal(G, w)
Input: A connected undirected graph \( G = (V, E) \) with edge weights \( w_e \)
Output: A minimum spanning tree defined by the edges \( X \)

for all \( u \in V \):
    makeset(u)

\( X = {} \)
Sort the edges \( E \) by weight
for all edges \( \{u, v\} \in E \), in increasing order of weight:
    if \( \text{find}(u) \neq \text{find}(v) \):
        add edge \( \{u, v\} \) to \( X \)
        \( \text{union}(u, v) \)
```

- Every node is connected to only itself
- If \( u \) and \( v \) are not in the same connected component
- Merge the two connected components of \( u \) and \( v \)
Union-by-rank data structure

- Maintain the nodes in a component as a tree
- The root of the tree represents the component
- `makeset(u)` creates a single node tree rooted at `u`
- `find(x)` traverse the tree from `x` to its root, and return the root
- `union(x, y)` merges two trees into a single one
  - Keep track of the rank of each node (the height of subtree rooted at that node)
  - When merging two trees, pick the root that has higher rank values to be the root
After makeset(A), makeset(B), ..., makeset(G):

After union(A, D), union(B, E), union(C, F):

After union(C, G), union(E, A):

After union(B, G):

procedure union(x, y)
    \( r_x = \text{find}(x) \)
    \( r_y = \text{find}(y) \)
    if \( r_x = r_y \): return
    if rank\( (r_x) > \) rank\( (r_y) \):
        \( \pi(r_x) = r_y \)
    else:
        \( \pi(r_y) = r_x \)
        if rank\( (r_x) = \) rank\( (r_y) \): rank\( (r_y) = \) rank\( (r_y) + 1 \)
• Property: any root node of rank $k$ has at least $2^k$ nodes in the tree
  – Try to work out the proof (by induction on $k$) by yourself

• There are maximum of $|V|$ nodes in the tree
  – The maximum rank is thus $O(\log|V|)$
  – $find(x)$ runs in $O(\log |V|)$ time
procedure kruskal\((G, w)\)
Input: A connected undirected graph \(G = (V, E)\) with edge weights \(w_e\)
Output: A minimum spanning tree defined by the edges \(X\)

for all \(u \in V\):
    makeset\(u\)

\(X = \{\}\)
Sort the edges \(E\) by weight
for all edges \(\{u, v\} \in E\), in increasing order of weight:
    if find\(u\) \neq \text{find}(v):
        add edge \(\{u, v\}\) to \(X\)
        union\(u, v\)  

\(\text{Total running time:} \quad O(|V|) + O(|E| \log |E|) + O(|E| \log |V|)\)