Graphs

Minimum Spanning Trees
Problem: Laying Telephone Wire
Wiring: Naïve Approach

Expensive!
Wiring: Better Approach

Minimize the total length of wire connecting the customers
Minimum Spanning Tree (MST)

(see Weiss, Section 24.2.2)

A **minimum spanning tree** is a subgraph of an undirected weighted graph $G$, such that

- it is a tree (i.e., it is acyclic)
- it covers all the vertices $V$
  - contains $|V| - 1$ edges
- the total cost associated with tree edges is the minimum among all possible spanning trees
- not necessarily unique
How Can We Generate a MST?
Prim’s Algorithm

Initialization

a. Pick a vertex \( r \) to be the root
b. Set \( D(r) = 0, \text{parent}(r) = \text{null} \)
c. For all vertices \( v \in V, v \neq r \), set \( D(v) = \infty \)
d. Insert all vertices into priority queue \( P \), using distances as the keys

<table>
<thead>
<tr>
<th>Vertex</th>
<th>Parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>-</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>d</td>
<td>e</td>
</tr>
<tr>
<td>e</td>
<td></td>
</tr>
</tbody>
</table>

Distance Table:

<table>
<thead>
<tr>
<th></th>
<th>e</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>\infty</td>
<td>\infty</td>
<td>\infty</td>
<td>\infty</td>
<td></td>
</tr>
</tbody>
</table>
Prim’s Algorithm

While $P$ is not empty:

1. Select the next vertex $u$ to add to the tree
   $u = P$.deleteMin()

2. Update the weight of each vertex $w$ adjacent to $u$ which is not in the tree (i.e., $w \in P$)
   If $\text{weight}(u,w) < D(w)$,
   a. $\text{parent}(w) = u$
   b. $D(w) = \text{weight}(u,w)$
   c. Update the priority queue to reflect new distance for $w$
Prim’s algorithm

The MST initially consists of the vertex e, and we update the distances and parent for its adjacent vertices.

<table>
<thead>
<tr>
<th>Vertex</th>
<th>Parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>-</td>
</tr>
<tr>
<td>b</td>
<td>-</td>
</tr>
<tr>
<td>c</td>
<td>-</td>
</tr>
<tr>
<td>d</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>e</th>
<th>d</th>
<th>b</th>
<th>c</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>d</th>
<th>b</th>
<th>c</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>5</td>
<td>∞</td>
</tr>
</tbody>
</table>
Prim’s algorithm

```
<table>
<thead>
<tr>
<th>Vertex</th>
<th>Parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>-</td>
</tr>
<tr>
<td>b</td>
<td>e</td>
</tr>
<tr>
<td>c</td>
<td>e</td>
</tr>
<tr>
<td>d</td>
<td>e</td>
</tr>
<tr>
<td>a</td>
<td>d</td>
</tr>
</tbody>
</table>
```

![Graph with Prim's algorithm]
Prim’s algorithm

<table>
<thead>
<tr>
<th>Vertex</th>
<th>Parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>c</td>
</tr>
<tr>
<td>b</td>
<td>e</td>
</tr>
<tr>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>d</td>
<td>e</td>
</tr>
<tr>
<td>a</td>
<td>d</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Vertex</th>
<th>Parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>e</td>
</tr>
<tr>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>d</td>
<td>e</td>
</tr>
<tr>
<td>a</td>
<td>d</td>
</tr>
</tbody>
</table>
Prim’s algorithm
Prim’s algorithm

The final minimum spanning tree

<table>
<thead>
<tr>
<th>Vertex</th>
<th>Parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>-</td>
</tr>
<tr>
<td>b</td>
<td>e</td>
</tr>
<tr>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>d</td>
<td>e</td>
</tr>
<tr>
<td>a</td>
<td>d</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Vertex</th>
<th>Parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>e</td>
<td>-</td>
</tr>
<tr>
<td>b</td>
<td>e</td>
</tr>
<tr>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>d</td>
<td>e</td>
</tr>
<tr>
<td>a</td>
<td>d</td>
</tr>
</tbody>
</table>
Running time of Prim’s algorithm (without heaps)

**Initialization of priority queue** (array): $O(|V|)$

**Update loop**: $|V|$ calls

- Choosing vertex with minimum cost edge: $O(|V|)$
- Updating distance values of unconnected vertices: each edge is considered only once during entire execution, for a total of $O(|E|)$ updates

**Overall cost without heaps**: $O(|E| + |V|^2)$

When heaps are used, apply same analysis as for Dijkstra’s algorithm (p.469) (good exercise)
Prim’s Algorithm Invariant

- At each step, we add the edge \((u,v)\) s.t. the weight of \((u,v)\) is minimum among all edges where \(u\) is in the tree and \(v\) is not in the tree.

- Each step maintains a minimum spanning tree of the vertices that have been included thus far.

- When all vertices have been included, we have a MST for the graph!
Correctness of Prim’s

- This algorithm adds $n-1$ edges without creating a cycle, so clearly it creates a spanning tree of any connected graph \(\text{(you should be able to prove this)}\).

But is this a minimum spanning tree?

Suppose it wasn't.

- There must be point at which it fails, and in particular there must a single edge whose insertion first prevented the spanning tree from being a minimum spanning tree.
Correctness of Prim’s

- Let $G$ be a connected, undirected graph
- Let $S$ be the set of edges chosen by Prim’s algorithm before choosing an errorful edge $(x,y)$
- Let $V'$ be the vertices incident with edges in $S$
- Let $T$ be a MST of $G$ containing all edges in $S$, but not $(x,y)$. 
Correctness of Prim’s

- Edge \((x,y)\) is not in \(T\), so there must be a path in \(T\) from \(x\) to \(y\) since \(T\) is connected.

- Inserting edge \((x,y)\) into \(T\) will create a cycle

- There is exactly one edge on this cycle with exactly one vertex in \(V'\), call this edge \((v,w)\)
Correctness of Prim’s

- Since Prim’s chose \((x, y)\) over \((v, w)\), \(w(v, w) \geq w(x, y)\).
- We could form a new spanning tree \(T’\) by swapping \((x, y)\) for \((v, w)\) in \(T\) (prove this is a spanning tree).
- \(w(T’)\) is clearly no greater than \(w(T)\)
- But that means \(T’\) is a MST
- And yet it contains all the edges in \(S\), and also \((x, y)\)

...Contradiction
Another Approach

- Create a forest of trees from the vertices
- Repeatedly merge trees by adding “safe edges” until only one tree remains
- A “safe edge” is an edge of minimum weight which does not create a cycle

forest: \{a\}, \{b\}, \{c\}, \{d\}, \{e\}
Kruskal’s algorithm

Initialization

a. Create a set for each vertex \( v \in V \)
b. Initialize the set of “safe edges” \( A \)
   comprising the MST to the empty set
c. Sort edges by increasing weight

\[
F = \{a\}, \{b\}, \{c\}, \{d\}, \{e\} \\
A = \emptyset \\
E = \{(a,d), (c,d), (d,e), (a,c), \\
(b,e), (c,e), (b,d), (a,b)\}
\]
Kruskal’s algorithm

For each edge \((u,v) \in E\) in increasing order

while more than one set remains:

If \(u\) and \(v\), belong to different sets \(U\) and \(V\)

a. add edge \((u,v)\) to the safe edge set

\[ A = A \cup \{(u,v)\} \]

b. merge the sets \(U\) and \(V\)

\[ F = F - U - V + (U \cup V) \]

Return \(A\)

- Running time bounded by sorting (or findMin)
- \(O(|E|\log|E|)\), or equivalently, \(O(|E|\log|V|)\) (why???)
Kruskal’s algorithm

\[ E = \{(a,d), (c,d), (d,e), (a,c), (b,e), (c,e), (b,d), (a,b)\} \]

\[
\begin{array}{l}
\text{Forest} \\
\{a\}, \{b\}, \{c\}, \{d\}, \{e\} \\
\{a,d\}, \{b\}, \{c\}, \{e\} \\
\{a,d,c\}, \{b\}, \{e\} \\
\{a,d,c,e\}, \{b\} \\
\{a,d,c,e,b\}
\end{array}
\]

\[
\begin{array}{l}
\mathcal{A} \\
\emptyset \\
\{(a,d)\} \\
\{(a,d), (c,d)\} \\
\{(a,d), (c,d), (d,e)\} \\
\{(a,d), (c,d), (d,e), (b,e)\}
\end{array}
\]
Kruskal’s Algorithm Invariant

- After each iteration, every tree in the forest is a MST of the vertices it connects.

- Algorithm terminates when all vertices are connected into one tree.
Correctness of Kruskal’s Algorithm

- This algorithm adds \( n-1 \) edges without creating a cycle, so clearly it creates a spanning tree of any connected graph (\textit{you should be able to prove this}).

But is this a \textit{minimum} spanning tree?

Suppose it wasn't.

- There must be point at which it fails, and in particular there must a single edge whose insertion first prevented the spanning tree from being a minimum spanning tree.
Correctness of Kruskal’s

- Let $e$ be this first errorful edge.
- Let $K$ be the Kruskal spanning tree
- Let $S$ be the set of edges chosen by Kruskal’s algorithm before choosing $e$
- Let $T$ be a MST containing all edges in $S$, but not $e$. 
Correctness of Kruskal’s

Lemma: \( w(e') \geq w(e) \) for all edges \( e' \) in \( T - S \)

**Proof (by contradiction):**

- Assume there exists some edge \( e' \) in \( T - S \), \( w(e') < w(e) \)
- Kruskal’s must have considered \( e' \) before \( e \)
  - However, since \( e' \) is not in \( K \) (*why??*), it must have been discarded because it caused a cycle with some of the other edges in \( S \).
  - But \( e' + S \) is a subgraph of \( T \), which means it cannot form a cycle

...Contradiction
Correctness of Kruskal’s

- Inserting edge $e$ into $T$ will create a cycle
- There must be an edge on this cycle which is not in $K$ *(why??)*. Call this edge $e'$
- $e'$ must be in $T - S$, so (by our lemma) $w(e') \geq w(e)$
- We could form a new spanning tree $T'$ by swapping $e$ for $e'$ in $T$ (*prove this is a spanning tree*).
- $w(T')$ is clearly no greater than $w(T)$
- But that means $T'$ is a MST
- And yet it contains all the edges in $S$, and also $e$
  ...Contradiction
Greedy Approach

- Like Dijkstra’s algorithm, both Prim’s and Kruskal’s algorithms are **greedy algorithms**

- The greedy approach works for the MST problem; however, **it does not work for many other problems!**