Shortest Path Algorithms

Minimal Spanning Tree (MST) Algorithms
Prim and Kruskal

Graphs: Definitions

- Simple Path: sequence of edges leading from one vertex to another with no vertex appearing twice
- Connected Graph: a simple path exists between any pair of vertices
- Cycle: simple path, except that first and final nodes are the same
Definitions

- Edge-weighted graph $G = (V,E)$
- Subgraph of $G$ is $G' = (V',E')$, such that
  - $V \subseteq V$ and $E \subseteq E$
- Spanning Tree of $G$ is $T = (V',E')$
  - $V = V$
  - $T$ is connected
  - $T$ is acyclic

Cost/weight associated with each edge

- $C(\langle v_i, v_j \rangle)$

Total cost for tree

- For all $\langle v_i, v_j \rangle \in T$, $\sum_{i=1}^{n-1} C(\langle v_i, v_{i+1} \rangle)$

MinimalSpanning Tree (MST)

- Find $T = (V,E'$) with smallest total cost

The general problem

Given an edge-weighted undirected graph $G = (V,E)$

Find a tree $T$ that contains all nodes in $G$ and the sum of the costs of the edges in $T$ is minimal

That is, $T = (V,E')$, and $\Sigma C(E')$ is minimal
Prim’s Algorithm

- Greedy algorithm for finding MST on edge-weighted, connected, undirected graph
- Select edges one-by-one and add to spanning tree

**Given graph** $G = (V,E)$
- Init to 2 sets of vertices: ‘innies’ & ‘outies’
  - ‘innies’ are visited nodes (initially empty)
  - ‘outies’ are not yet visited (initially $V$)
- First innie is random node (root of MST)
- Iteratively (until no more outies)
  - choose outie ($v'$) with smallest distance from any innie
  - move $v'$ from outies to innies

**For Implementation Need:**

- For each vertex $v$, need to record:
  - $k_v$: has $v$ been visited? (initially false for all $v \in V$)
  - $d_v$: What is the minimal edge weight to $v$?
    (initially $\infty$ for all $v \in V$, except $v_v = 0$)
  - $p_v$: What vertex precedes (is parent of) $v$?
    (initially unknown for all $v \in V$)
<table>
<thead>
<tr>
<th>v</th>
<th>$k_v$</th>
<th>$d_v$</th>
<th>$p_v$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>F</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>F</td>
<td>∞</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>F</td>
<td>∞</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>F</td>
<td>∞</td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>F</td>
<td>∞</td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>F</td>
<td>∞</td>
<td></td>
</tr>
</tbody>
</table>

![Diagram](image)

<table>
<thead>
<tr>
<th>v</th>
<th>$k_v$</th>
<th>$d_v$</th>
<th>$p_v$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>F</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>F</td>
<td>13</td>
<td>a</td>
</tr>
<tr>
<td>c</td>
<td>F</td>
<td>8</td>
<td>a</td>
</tr>
<tr>
<td>d</td>
<td>F</td>
<td>1</td>
<td>a</td>
</tr>
<tr>
<td>e</td>
<td>F</td>
<td>∞</td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>F</td>
<td>∞</td>
<td></td>
</tr>
</tbody>
</table>

![Diagram](image)

<table>
<thead>
<tr>
<th>v</th>
<th>$k_v$</th>
<th>$d_v$</th>
<th>$p_v$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>T</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>F</td>
<td>13</td>
<td>a</td>
</tr>
<tr>
<td>c</td>
<td>F</td>
<td>5</td>
<td>d</td>
</tr>
<tr>
<td>d</td>
<td>T</td>
<td>1</td>
<td>a</td>
</tr>
<tr>
<td>e</td>
<td>F</td>
<td>4</td>
<td>d</td>
</tr>
<tr>
<td>f</td>
<td>F</td>
<td>5</td>
<td>d</td>
</tr>
</tbody>
</table>

![Diagram](image)
Prim’s Algorithm

Prim(G, s₀)
{
    // Initialize
    n = |V|;
    createtable(n); // stores k, d, p
    createpq(|E|); // empty pq
    table[s₀].d = 0;
    insertpq(0, s₀);

    // con’t
    while (!pq.isempty)
    {
        v₀ = getMin();
        if (!table[v₀].k) // not known
        {
            table[v₀].k = true;
            for each vᵢ ∈ Adj[v₀]
            {
                newd = weight(vᵢ, v₀);
            }
        }
    }
}
Prim’s Algorithm

// con’t
newd = weight(vi,v0);
if (table[vi].d > newd)
{
    table[vi].d = newd;
    table[vi].p = v0;
    insertpq(newd,vi);
}
}
}

Prim’s Algorithm

// con’t
for each vi ∈ G(V,E)
// build vertex set in T
    v ∈ T(V,E’);
for each vi ∈ G(V,E)
// build edge set in T
    (v,table[vi].p) ∈ T(V,E’);
}

Kruskal’s Algorithm

- Greedy algorithm for finding MST on edge-weighted, connected, undirected graph
- Select edges one-by-one and add to forest
Kruskal’s Algorithm

- Given graph $G = (V, E)$
- Iteratively (until MST found)
  - Select edge $(v_i, v_j) \in E$ s.t. $C(v_i, v_j)$ is min
  - However, $(v_i, v_j)$ must not create cycle
Kruskal's Algorithm

Kruskal(G)
{
    // Initialize
    n = |V|;
    for each v ∈ G(V,E) O( )
    // build vertex set in T
    v ∈ T(V,E'); O( )
    createpq(|E|); // empty heap O( )
}

Kruskal's Algorithm

// con't
// build priority queue
for each E ∈ G(V,E) O( )
insertpq(weight,(v₁,v₂)); O( )
while (!pq.isempty) O( )
{
    {v₁,v₂} = getMin(); O( )
    v₁ ∈ X; v₂ ∈ Y; O( )
    if X ≠ Y O( )
        Join(X,Y); O( )
    (v₁,v₂) ∈ T(V,E'); O( )
}
MST Summary

- MST is lowest cost tree that includes all nodes in a graph
- Two algorithms to find MST
  - Prim
    - Iteratively add closest node to current tree
  - Kruskal
    - Iteratively build forest with minimal edges