CSE 373: Data Abstractions
Minimum Spanning Trees
Ruth Anderson
Autumn 2012

Minimum Spanning Trees
Given an undirected graph $G=(V,E)$, find a graph $G'=(V,E')$ such that:
- $E'$ is a subset of $E$
- $|E'| = |V| - 1$
- $G'$ is connected
- $\sum_{(u,v) \in E'} c_{uv}$ is minimal

Applications:
- Example: Electrical wiring for a house or clock wires on a chip
- Example: A road network if you cared about asphalt cost rather than travel time

Student Activity
Find the MST

Two Different Approaches
Prim's Algorithm
Almost identical to Dijkstra's
Kruskals's Algorithm
Completely different!

Prim's algorithm
Idea: Grow a tree by picking a vertex from the unknown set that has the smallest cost. Here cost = cost of the edge that connects that vertex to the known set. Pick the vertex with the smallest cost that connects “known” to “unknown.”

Prim's Algorithm vs. Dijkstra's
Recall:
Dijkstra picked the unknown vertex with smallest cost where cost = distance to the source.
Prim's pick the unknown vertex with smallest cost where cost = distance from this vertex to the known set (in other words, the cost of the smallest edge connecting this vertex to the known set)
- Otherwise identical
Prim's Algorithm for MST

1. For each node $v$, set $v.cost = \infty$ and $v\_known = false$
2. Choose any node $v$, (this is like your “start” vertex in Dijkstra)
   a) Mark $v$ as known
   b) For each edge $(v, u)$ with weight $w$
      set $u.cost = w$ and $u\_prev = v$
3. While there are unknown nodes in the graph
   a) Select the unknown node $v$ with lowest $cost$
   b) Mark $v$ as known and add $(v, v\_prev)$ to output (the MST)
   c) For each edge $(v, u)$ with weight $w$
      if ($w < u.cost$) {
         $u.cost = w$
         $u\_prev = v$
      }

Example: Find MST using Prim's

\[
\begin{array}{c|c|c|c}
\text{vertex} & \text{known?} & \text{cost} & \text{prev} \\
\hline
A & Y & 0 & \\
B & 2 & A & \\
C & 2 & A & \\
D & 1 & A & \\
E & ? & ? & \\
F & ? & ? & \\
G & ? & ? & \\
\end{array}
\]
Example: Find MST using Prim's

<table>
<thead>
<tr>
<th>vertex</th>
<th>known?</th>
<th>cost</th>
<th>prev</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Y</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>Y</td>
<td>1</td>
<td>E</td>
</tr>
<tr>
<td>C</td>
<td>Y</td>
<td>1</td>
<td>D</td>
</tr>
<tr>
<td>D</td>
<td>Y</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>E</td>
<td>Y</td>
<td>1</td>
<td>D</td>
</tr>
<tr>
<td>F</td>
<td>Y</td>
<td>2</td>
<td>C</td>
</tr>
<tr>
<td>G</td>
<td>Y</td>
<td>3</td>
<td>E</td>
</tr>
</tbody>
</table>

Prim's Analysis

- Correctness ??
  - A bit tricky
  - Intuitively similar to Dijkstra
  - Might return to this time permitting (unlikely)

- Run-time
  - Same as Dijkstra
  - $O(|E| \log |V|)$ using a priority queue

Kruskal's MST Algorithm

Idea: Grow a forest out of edges that do not create a cycle. Pick an edge with the smallest weight.

$G = (V, E)$
**Kruskal's Algorithm for MST**

**An edge-based greedy algorithm**

Builds MST by greedily adding edges

1. Initialize with
   - empty MST
   - all vertices marked unconnected
   - all edges unmarked
2. While there are still unmarked edges
   a. Pick the lowest cost edge \((u, v)\) and mark it
   b. If \(u\) and \(v\) are not already connected, add \((u, v)\) to the MST and mark \(u\) and \(v\) as connected to each other

---

**Kruskal's pseudo code**

```c
void Graph::kruskal()
{
    int edgesAccepted = 0;
    DisjSet s(NUM_VERTICES);
    while (edgesAccepted < NUM_VERTICES - 1)
    {
        e = smallest weight edge not deleted yet;
        // edge e = (u, v)
        uset = s.find(u);
        vset = s.find(v);
        if (uset != vset)
        {
            edgesAccepted++;
            s.unionSets(uset, vset);
        }
    }
}
```

<table>
<thead>
<tr>
<th>Time complexity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2|E</td>
<td>) heap ops</td>
</tr>
</tbody>
</table>

---

**Student Activity**

Find MST using Kruskal's

Now find the MST using Prim's method.

Under what conditions will these methods give the same result?

---

**Example: Find MST using Kruskal's**

Edges in sorted order:
1. \((A,D)\), \((C,D)\), \((B,E)\), \((D,E)\)
2. \((A,B)\), \((C,F)\), \((A,C)\)
3. \((E,G)\)
4. \((D,G)\), \((B,D)\)
5. \((D,F)\)
6. \((F,G)\)
7. \((2,1)\)
8. \((2,5)\)
9. \((2,3)\)
10. \((2,6)\)

Output:

\((A,D)\)

Note: At each step, the union/find sets are the trees in the forest
Example: Find MST using Kruskal's

Edges in sorted order:
1: (A,D), (C,D), (B,E), (D,E)
2: (A,B), (C,F), (A,C)
3: (E,G)
5: (D,G), (B,D)
6: (D,F)
10: (F,G)

Output: (A,D), (C,D), (B,E)

Note: At each step, the union/find sets are the trees in the forest

Example: Find MST using Kruskal's

Edges in sorted order:
1: (A,D), (C,D), (B,E), (D,E)
2: (A,B), (C,F), (A,C)
3: (E,G)
5: (D,G), (B,D)
6: (D,F)
10: (F,G)

Output: (A,D), (C,D), (B,E), (D,E)

Note: At each step, the union/find sets are the trees in the forest

Example: Find MST using Kruskal's

Edges in sorted order:
1: (A,D), (C,D), (B,E), (D,E)
2: (A,B), (C,F), (A,C)
3: (E,G)
5: (D,G), (B,D)
6: (D,F)
10: (F,G)

Output: (A,D), (C,D), (B,E), (D,E), (C,F)

Note: At each step, the union/find sets are the trees in the forest

Example: Find MST using Kruskal's

Edges in sorted order:
1: (A,D), (C,D), (B,E), (D,E)
2: (A,B), (C,F), (A,C)
3: (E,G)
5: (D,G), (B,D)
6: (D,F)
10: (F,G)

Output: (A,D), (C,D), (B,E), (D,E), (C,F), (E,G)

Note: At each step, the union/find sets are the trees in the forest
**Correctness**

Kruskal’s algorithm is clever, simple, and efficient

- But does it generate a minimum spanning tree?
- How can we prove it?

First: it generates a spanning tree

- Intuition: Graph started connected and we added every edge that did not create a cycle
- Proof by contradiction: Suppose u and v are disconnected in Kruskal’s result. Then there’s a path from u to v in the initial graph with an edge we could add without creating a cycle. But Kruskal would have added that edge. Contradiction.

Second: There is no spanning tree with lower total cost...

**The inductive proof set-up**

Let $F$ (stands for “forest”) be the set of edges Kruskal has added at some point during its execution.

Claim: $F$ is a subset of one or more MSTs for the graph

(Therefore, once $|F|=|V|-1$, we have an MST.)

Proof: By induction on $|F|

Base case: $|F|=0$: The empty set is a subset of all MSTs

Inductive case: $|F|=k+1$: By induction, before adding the $(k+1)$th edge (call it $e$), there was some MST $T$ such that $F-e \subseteq T$...