Graphs: Minimum Spanning Trees
(Chapter 9)

CSE 373
Data Structures and Algorithms

Today’s Outline

• Admin:
  – Midterm #2 – Friday Nov 18th, topic list has been posted
  – HW #5 – Graphs, partners allowed, due after Thanksgiving

• Graphs
  – Shortest Paths
  – Minimum Spanning Trees

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
- 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

Two Different Approaches

Prim’s Algorithm
Almost identical to Dijkstra’s

Kruskals’s Algorithm
Completely different!

Student Activity

Find the MST

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.”

A node-based greedy algorithm
Builds MST by greedily adding nodes
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
- Compare to slides in Dijkstra lecture!

Prim's Algorithm for MST

1. For each node v, set v.cost = ∞ 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
### 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>E</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>2</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>3</td>
<td>E</td>
<td></td>
</tr>
</tbody>
</table>

### Primitive Analysis

- **Correctness**
  - Intuitively similar to Dijkstra
- **Run-time**
  - $O(|E| \log |V|)$ using a priority queue

---

### Student Activity

**Find MST using Prim's**

<table>
<thead>
<tr>
<th>V</th>
<th>Kwn</th>
<th>Distance</th>
<th>path</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V7</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Order Declared Known:**

- $V_1$

---

**Prim's Analysis**

- Correctness ??
  - Intuitively similar to Dijkstra
- 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**

```java
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);
    }
  }
}
```

**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: \((E,G)\)
5: \((D,G),(B,D)\)
6: \((D,F)\)
10: \((F,G)\)

Output:

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)

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