CSE373: Data Structures & Algorithms
Lecture 20: Minimum Spanning Trees

Linda Shapiro
Spring 2016
Announcements

• HW 4 due Wednesday, May 18
• HW 5 will be due June 1. Ben Jones is our expert in Ezgi’s absence. benjones@cs
Minimum Spanning Trees

The minimum-spanning-tree problem
– Given a weighted undirected graph, compute a spanning tree of minimum weight

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

$G’$ is a minimum spanning tree.
Two different approaches

Prim’s Algorithm
Almost identical to Dijkstra’s

Kruskals’s Algorithm
Completely different!
**Prim’s Algorithm Idea**

**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 vs. Dijkstra’s

Recall:

Dijkstra picked the unknown vertex with smallest cost where
\[ \text{cost} = \text{distance to the source}. \]

Prim’s pick the unknown vertex with smallest cost where
\[ \text{cost} = \text{distance from this vertex to the known set} \]
(in other words, the cost of the smallest edge connecting this vertex
to the known set)

Otherwise identical 😊
Prim’s Algorithm

1. For each node \( v \), set \( v\.cost = \infty \) and \( v\.known = \text{false} \)
2. Choose any node \( v \)
   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
   c) For each edge \((v, u)\) with weight \( w \),
      \[
      \text{if}(w < u\.cost) \{
      u\.cost = w; \\
      u\.prev = v;
      \}
      \]
Prim’s Example

Select the unknown node $v$ with lowest cost
Mark $v$ as known and add $(v, v.\text{prev})$ to output
For each edge $(v,u)$ with weight $w$,
    
    if ($w < u.\text{cost}$) {
        $u.\text{cost} = w$;
        $u.\text{prev} = v$;
    }

<table>
<thead>
<tr>
<th>vertex</th>
<th>known?</th>
<th>cost</th>
<th>prev</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>??</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>??</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>??</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>??</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>??</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>??</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>??</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Prim’s Example**

Select the unknown node $v$ with lowest cost.
Mark $v$ as known and add $(v, v.\text{prev})$ to output.

For each edge $(v,u)$ with weight $w$,
if ($w < u.\text{cost}$) {
    $u.\text{cost} = w$;
    $u.\text{prev} = v$;
}
Prim’s Example

Select the unknown node \( v \) with lowest cost
Mark \( v \) as known and add \((v, \text{v.prev})\) to output
For each edge \((v, u)\) with weight \( w \),
\[
\text{if}(w < u.\text{cost}) \{
    u.\text{cost} = w;
    u.\text{prev} = v;
\}
\]

<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></td>
<td>2</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td></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></td>
<td>1</td>
<td>D</td>
</tr>
<tr>
<td>F</td>
<td></td>
<td>6</td>
<td>D</td>
</tr>
<tr>
<td>G</td>
<td></td>
<td>5</td>
<td>D</td>
</tr>
</tbody>
</table>
Prim’s Example

Select the unknown node v with lowest cost
Mark v as known and add (v, v.prev) to output
For each edge (v,u) with weight w,
    if(w < u.cost) {
        u.cost = w;
        u.prev = v;
    }
**Prim’s Example**

Select the unknown node v with lowest cost
Mark v as known and add (v, v.prev) to output
For each edge (v,u) with weight w,
  \[
  \text{if}(w < u.\text{cost}) \{
  \quad u.\text{cost} = w;
  \quad u.\text{prev} = v;
  \}\]

<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></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></td>
<td>2</td>
<td>C</td>
</tr>
<tr>
<td>G</td>
<td></td>
<td>3</td>
<td>E</td>
</tr>
</tbody>
</table>
Prim's Example

Select the unknown node \( v \) with lowest cost
Mark \( v \) as known and add \((v, v.\text{prev})\) to output
For each edge \((v, u)\) with weight \( w \),
\[
\text{if}(w < u.\text{cost}) \{
\quad u.\text{cost} = w;
\quad u.\text{prev} = v;
\}
\]

<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></td>
<td>2</td>
<td>C</td>
</tr>
<tr>
<td>G</td>
<td></td>
<td>3</td>
<td>E</td>
</tr>
</tbody>
</table>
Prim’s Example

Select the unknown node v with lowest cost
Mark v as known and add (v, v.prev) to output
For each edge (v, u) with weight w,
    if (w < u.cost) {
        u.cost = w;
        u.prev = v;
    }
**Prim’s Example**

Select the unknown node $v$ with lowest cost
Mark $v$ as known and add $(v, v.prev)$ to output
For each edge $(v,u)$ with weight $w$,
if($w < u.cost$) {
    $u.cost = w$;
    $u.prev = v$;
}

<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>
Analysis

• Correctness
  – A bit tricky
  – Intuitively similar to Dijkstra

• Run-time
  – Same as Dijkstra
  – $O(|E| \log |V|)$ using a priority queue
    • Costs/priorities are just edge-costs, not path-costs
A cable company wants to connect five villages to their network which currently extends to the town of Avonford. What is the minimum length of cable needed?
Prim’s Algorithm

Model the situation as a graph and find the MST that connects all the villages (nodes).
Prim’s Algorithm

Select any vertex

A

Select the shortest edge connected to that vertex (since it’s the only known one)

AB 3
Prim’s Algorithm

Select the shortest edge that connects an unknown vertex to any known vertex.

AE  4
Prim’s Algorithm

Select the shortest edge that connects an unknown vertex to any known vertex.

ED 2
Prim’s Algorithm

Select the shortest edge that connects an unknown vertex to any known vertex.

DC  4
Prim’s Algorithm

Select the shortest edge that connects an unknown vertex to any known vertex.

EF 5
Prim’s Algorithm

All vertices have been connected.

The solution is

AB 3
AE 4
ED 2
DC 4
EF 5

Total weight of tree: 18
Start with A. Make it Known.
HINT: Choose the least cost edge out of it. Add that node to the Known.
A connects to B, C, E
Lowest cost is to E
Now we consider anything connected to A or E.

Nodes considered are B, C, D. Lowest cost to B.
Now we consider anything connected to A, B, or E.

Nodes considered are C, D.

Lowest cost to D.
Now we consider anything connected to A, B, D, or E.

Nodes considered are C.

Lowest cost is from D to C.

DONE
Minimum Spanning Tree Algorithms

• **Prim’s Algorithm** for Minimum Spanning Tree
  - Similar idea to Dijkstra’s Algorithm but for MSTs.
  - Both based on expanding cloud of known vertices
    (basically using a priority queue instead of a DFS stack)

• **Kruskal’s Algorithm** for Minimum Spanning Tree
  - Another, but different, greedy MST algorithm.
  - Uses the Union-Find data structure.
Kruskal’s Algorithm

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

An edge-based greedy algorithm
Builds MST by greedily adding edges

\[ G = (V, E) \]
Kruskal’s Algorithm Pseudocode

1. Sort edges by weight (min-heap)
2. Each node in its own set (up-trees)
3. While output size < |V| - 1
   - Consider next smallest edge \((u, v)\)
   - if \(\text{find}(u)\) and \(\text{find}(v)\) indicate \(u\) and \(v\) are in different sets
     • output \((u, v)\)
     • \(\text{union}(\text{find}(u), \text{find}(v))\)

Recall invariant:
- \(u\) and \(v\) in same set if and only if connected in output-so-far
**Kruskal’s Example**

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
Kruskal’s Example

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)

Note: At each step, the union/find sets are the trees in the forest
Kruskal’s Example

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
Kruskal’s Example

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
Kruskal’s Example

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
Kruskal’s Example

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
Kruskal’s Example

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
Kruskal’s Example

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
Kruskal’s Example

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
Kruskal’s Algorithm Analysis

Idea: Grow a forest out of edges that do not grow a cycle, just like for the spanning tree problem.
  – But now consider the edges in order by weight

So:
  – Sort edges: $O(|E| \log |E|)$ (next course topic)
  – Iterate through edges using union-find for cycle detection almost $O(|E|)$

Somewhat better:
  – Floyd’s algorithm to build min-heap with edges $O(|E|)$
  – Iterate through edges using union-find for cycle detection and deleteMin to get next edge $O(|E| \log |E|)$
  – Not better worst-case asymptotically, but often stop long before considering all edges.
Kruskal’s Algorithm

List the edges in order of size:

- ED 2
- AB 3
- AE 4
- CD 4
- BC 5
- EF 5
- CF 6
- AF 7
- BF 8
- CF 8
Kruskal’s Algorithm

Select the edge with min cost

ED 2
Kruskal’s Algorithm

Select the next minimum cost edge that does not create a cycle

ED  2
AB  3
Kruskal’s Algorithm

Select the next minimum cost edge that does not create a cycle

ED 2
AB 3
CD 4 (or AE 4)
Select the next minimum cost edge that does not create a cycle

ED  2
AB  3
CD  4
AE  4
Select the next minimum cost edge that does not create a cycle

ED 2
AB 3
CD 4
AE 4
BC 5 – forms a cycle
EF 5
Kruskal’s Algorithm

All vertices have been connected.

The solution is

ED  2
AB  3
CD  4
AE  4
EF  5

Total weight of tree: 18
Practice

Order the edges.

(C,D)
(E,B)
(A,E)
(B,D)
(A,B)
(E,D)
(E,C)
(A,C)
Order the edges.

(C,D) ✔
(E,B)
(A,E)
(B,D)
(A,B)
(E,D)
(E,C)
(A,C)
Order the edges.

(C,D) ✔
(E,B) ✔
(A,E)
(B,D)
(A,B)
(E,D)
(E,C)
(A,C)
Order the edges.

(C,D) ✓
(E,B) ✓
(A,E) ❌
(B,D) ✓
(A,B) ✓
(E,D) ✓
(E,C) ✓
(A,C) ✓
Order the edges.

(C,D) ✔
(E,B) ✔
(A,E) ✔
(B,D) ✔
(A,B) ✔
(E,D) ✔
(E,C) ✔
(A,C) ✔
Order the edges.

(C,D) ✔
(E,B) ✔
(A,E) ✔
(B,D) ✔
(A,B) ✔
(E,D) ✔
(E,C) ✔
(A,C) ✔
Done with graph algorithms!

Next lecture…
• Sorting