CSE332: Data Abstractions

Lecture 23: Minimum Spanning Trees

Ruth Anderson
Winter 2013
Announcements

- Homework 7 – due NOW at the BEGINNING of lecture!
- Homework 8 – coming soon!

- Project 3 – the last programming project!
  - **ALL Code - Tues March 12, 2013 11PM** - (65% of overall grade):
  - Writeup - Thursday March 14, 2013, 11PM - (25% of overall grade)
“Scheduling note”

- “We now return to our interrupted program” on graphs
  - Last “graph lecture” was lecture 16
    - Shortest-path problem
    - Dijkstra’s algorithm for graphs with non-negative weights

- Why this strange schedule?
  - Needed to do parallelism and concurrency in time for project 3 and homeworks 6 and 7
  - But cannot delay all of graphs because of the CSE312 co-requisite

- So: not the most logical order, but hopefully not a big deal

3/04/2011
Trees

A tree is a graph with exactly one path between any two nodes.

No need for a root, hierarchy, or ordered children.
Spanning Trees

- A simple problem: Given a connected graph $G = (V, E)$, find a minimal subset of the edges such that the graph is still connected.
- A graph $G_2 = (V, E_2)$ such that $G_2$ is connected and removing any edge from $E_2$ makes $G_2$ disconnected.
Observations

1. Any solution to this problem is a tree
   - Recall a tree does not need a root; just means acyclic
   - For any cycle, could remove an edge and still be connected

2. Solution not unique unless original graph was already a tree

3. Problem ill-defined if original graph not connected

4. A tree with |V| nodes has |V|-1 edges
   - So every solution to the spanning tree problem has |V|-1 edges
Minimum Spanning Trees

Given an undirected graph $G=(V,E)$, find a graph $G'=(V, E')$ such that:

- $G'$ is a spanning tree.
- Sum of edge weights in $G'$ 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.

$G'$ is a **minimum spanning tree**.
An application

Bell systems was the telephone company for 100 years.

They want to connect everyone in the US to their telephone network as cheaply as possible.
Find the MST

Student Activity
Two Different Approaches

- **Prim’s Algorithm**
  - Almost identical to Dijkstra’s
  - One node, grow greedily

- **Kruskals’s Algorithm**
  - Completely different!
  - Forest of MSTs, *Union* them together.
  - I wonder how to union…

3/04/2011
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 = \textit{distance to the source}.
Prim’s pick the unknown vertex with smallest cost where
  cost = \textit{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 lecture 16!
Prim’s Algorithm for MST

1. For each node \( v \), set \( v.\text{cost} = \infty \) and \( v.\text{known} = \text{false} \)

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 \( u.\text{cost}=w \) and \( u.\text{prev}=v \)

3. While there are unknown nodes in the graph
   c. Select the unknown node \( v \) with lowest \( \text{cost} \)
   d. Mark \( v \) as known and add \((v, v.\text{prev})\) to output (the MST)
   e. For each edge \((v, u)\) with weight \( w\),
      
      if \( w < u.\text{cost} \) {
         \[ u.\text{cost} = w; \]
         \[ u.\text{prev} = v; \]
      }

Example: Find MST using Prim’s

![Graph with vertices and edges labeled]

<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>

3/04/2011
Example: Find MST using Prim’s

- **Graph**:
  - Vertices: A, B, C, D, E, F, G
  - Edges with weights:
    - A-B: 2
    - A-C: 2
    - A-D: 1
    - A-F: 2
    - B-D: 1
    - B-E: 1
    - C-D: 5
    - C-E: 3
    - C-G: 10
    - D-G: 6

- **Table**:
  - | vertex | known? | cost | prev |
  - |-------|-------|------|------|
  - | A     | Y     | 0    |      |
  - | B     |       | 2    | A    |
  - | C     |       | 2    | A    |
  - | D     |       | 1    | A    |
  - | E     |       | ??   |      |
  - | F     |       | ??   |      |
  - | G     |       | ??   |      |
Example: Find MST using Prim’s

Graph:

- A: 0
- B: 2
- C: 2
- D: 2
- E: 1
- F: 1
- G: 1

Costs:

- AB: 2
- BC: 1
- CD: 1
- DE: 1
- DF: 6
- DG: 5
- EF: 5
- FG: 10

Table:

<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>
Example: Find MST using Prim’s

```
vertex  known?  cost  prev
A       Y       0
B       2       A
C       Y       1   D
D       Y       1   A
E       1       D
F       2       C
G       5       D
```

3/04/2011
Example: Find MST using Prim’s

```
vertex   known? | cost | prev
---|---|---|---
A   | Y   | 0  |   
B   |     | 1  | E  
C   | Y   | 1  | D  
D   | Y   | 1  | A  
E   | Y   | 1  | D  
F   |     | 2  | C  
G   |     | 3  | E  ```
Example: Find MST using Prim’s

- **Graph**: A network of points (vertices) connected by lines (edges) with associated costs.
- **Table**: Shows the vertices, whether they are known, their cost, and their previous vertex.

<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>
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></td>
<td>3</td>
<td>E</td>
</tr>
</tbody>
</table>
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>
```

3/04/2011
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>

Start with $V_1$

Order Declared Known:

$V_1$

Total Cost:
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 heap

3/04/2011
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

Maze construction used random edge order.
Otherwise the same!
Aside: Union-Find aka Disjoint Set ADT

- **Union(x,y)** – take the union of two sets named x and y
  - Given sets: {3, 5, 7}, {4, 2, 8}, {9}, {1, 6}
  - **Union(5,1)**
    - Result: {3, 5, 7, 1, 6}, {4, 2, 8}, {9}.
    - To perform the union operation, we replace sets x and y by (x ∪ y)

- **Find(x)** – return the name of the set containing x.
  - Given sets: {3, 5, 7, 1, 6}, {4, 2, 8}, {9},
  - **Find(1)** returns 5
  - **Find(4)** returns 8

- We can do Union in constant time.
- We can get Find to be *amortized* constant time
  (worst case O(log n) for an individual Find operation).
Kruskal’s pseudo code

```cpp
void Graph::kruskal(){
    int edgesAccepted = 0;
    DisjSet s(NUM_VERTICES);

    Sort of ignore this loop in calc run-time...
    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);
        }
    }
}
```

**On heap of edges**

- `Deletemin = log |E|`
- `|E| heap ops`
- `2|E| finds`
- `One for each vertex in the edge`
- `Find = log |V|`
- `|V| unions`

**Union = O(1)**

\[
|E| \log |E| + 2|E|\log|V|+|V|
\]

**Sort of ignore this loop in calc run-time...**

\[
O(|E|\log|E|) = O(|E|\log|V|)
\]

\[
b/c \log |E| < \log|V|^2 = 2\log|V|
\]
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

3/04/2011
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)

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

3/04/2011
**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: \((A,D), (C,D), (B,E)\)

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

3/04/2011
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

3/04/2011
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)

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

3/04/2011
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 $F\setminus\{e\} \subseteq T$ ...
Staying a subset of some MST

Claim: \( F \) is a subset of one or more MSTs for the graph

So far: \( F \setminus \{e\} \subseteq T \):

Two disjoint cases:

- If \( \{e\} \subseteq T \): Then \( F \subseteq T \) and we’re done
- Else \( e \) forms a cycle with some simple path (call it \( p \)) in \( T \)
  - Must be since \( T \) is a spanning tree
**Staying a subset of some MST**

Claim: \( F \) is a subset of *one or more* MSTs for the graph

So far: \( F - \{e\} \subseteq T \) and

- There must be an edge \( e_2 \) on \( p \) such that \( e_2 \) is not in \( F \)
  - Else Kruskal would not have added \( e \)

- Claim: \( e_2 \text{.weight} == e \text{.weight} \)
Staying a subset of some MST

Claim: \( F \) is a subset of one or more MSTs for the graph

So far: \( F-\{e\} \subseteq T \)
- \( e \) forms a cycle with \( p \subseteq T \)
- \( e_2 \) on \( p \) is not in \( F \)

- Claim: \( e_2.\text{weight} = e.\text{weight} \)
  - If \( e_2.\text{weight} > e.\text{weight} \), then \( T \) is not an MST because \( T-\{e_2\}+\{e\} \) is a spanning tree with lower cost: contradiction
  - If \( e_2.\text{weight} < e.\text{weight} \), then Kruskal would have already considered \( e_2 \). It would have added it since \( T \) has no cycles and \( F-\{e\} \subseteq T \). But \( e_2 \) is not in \( F \): contradiction

3/04/2011
Staying a subset of *some* MST

Claim: \( F \) is a subset of *one or more* MSTs for the graph

So far: \( F-\{e\} \subseteq T \)
- \( e \) forms a cycle with \( p \subseteq T \)
- \( e2 \) on \( p \) is not in \( F \)
- \( e2.weight == e.weight \)

- Claim: \( T-\{e2\}+\{e\} \) is an MST
  - It’s a spanning tree because \( p-\{e2\}+\{e\} \) connects the same nodes as \( p \)
  - It’s minimal because its cost equals cost of \( T \), an MST
- Since \( F \subseteq T-\{e2\}+\{e\} \), \( F \) is a subset of one or more MSTs

Done.

3/04/2011