CSE 417
Algorithms and Complexity
Winter 2023
Lecture 13
Minimum Spanning Trees

Announcements
• Midterm, Wednesday, Feb 8

Minimum Spanning Tree

Greedy Algorithms for Minimum Spanning Tree
• Prim’s Algorithm: Extend a tree by including the cheapest outgoing edge
• Kruskal’s Algorithm: Add the cheapest edge that joins disjoint components

Greedy Algorithm 1
Prim’s Algorithm
• Extend a tree by including the cheapest outgoing edge

Greedy Algorithm 2
Kruskal’s Algorithm
• Add the cheapest edge that joins disjoint components
Why do the greedy algorithms work?

- For simplicity, assume all edge costs are distinct.

Edge inclusion lemma

- Let $S$ be a subset of $V$, and suppose $e = (u, v)$ is the minimum cost edge of $E$, with $u$ in $S$ and $v$ in $V - S$.
- $e$ is in every minimum spanning tree of $G$.
  - Or equivalently, if $e$ is not in $T$, then $T$ is not a minimum spanning tree.

Proof

- Suppose $T$ is a spanning tree that does not contain $e$.
- Add $e$ to $T$, this creates a cycle.
- The cycle must have some edge $e_1 = (u_1, v_1)$ with $u_1$ in $S$ and $v_1$ in $V - S$.

- $T_1 = T - \{e_1\} + \{e\}$ is a spanning tree with lower cost.
- Hence, $T$ is not a minimum spanning tree.

Optimality Proofs

- Prim’s Algorithm computes a MST.
- Kruskal’s Algorithm computes a MST.
- Show that when an edge is added to the MST by Prim or Kruskal, the edge is the minimum cost edge between $S$ and $V - S$ for some set $S$.

Prim’s Algorithm

```plaintext
S = {}; T = {};
while S != V
    choose the minimum cost edge $e = (u,v)$, with $u$ in $S$, and $v$ in $V - S$
    add $e$ to $T$
    add $v$ to $S$
```

Prove Prim’s algorithm computes an MST

- Show an edge $e$ is in the MST when it is added to $T$. 

Kruskal’s Algorithm

Let \( C = \{\{v_1\}, \{v_2\}, \ldots, \{v_n\}\}; \ T = \{\} \)
while \(|C| > 1\)
  Let \( e = (u, v) \) with \( u \) in \( C_i \) and \( v \) in \( C_j \) be the minimum cost edge joining distinct sets in \( C \)
  Replace \( C_i \) and \( C_j \) by \( C_i U C_j \)
  Add \( e \) to \( T \)

MST Implementation and runtime

• Prim’s Algorithm
  – Implementation, runtime: just like Dijkstra’s algorithm
  – Use a heap, runtime \( O(m \log n) \)
• Kruskal’s Algorithm
  – Sorting edges by cost: \( O(m \log n) \)
  – Managing connected components uses the Union-Find data structure
    • Amazing, pointer based data structure
    • Very interesting mathematical result

Disjoint Set ADT

• Data: set of pairwise disjoint sets.
• Required operations
  – Union – merge two sets to create their union
  – Find – determine which set an item appears in
• Check \( \text{Find}(v) \neq \text{Find}(w) \) to determine if \((v,w)\) joins separate components
• Do Union\((v,w)\) to merge sets

Application: Clustering

• Given a collection of points in an \( r \)-dimensional space and an integer \( K \), divide the points into \( K \) sets that are closest together

Observation: we will only traverse these trees upward from any given node to find the root.

Idea: reverse the pointers (make them point up from child to parent). The result is an up-tree.

Roots are the names of each set.
Distance clustering

- Divide the data set into $K$ subsets to maximize the distance between any pair of sets
  - $\text{dist}(S_1, S_2) = \min \{\text{dist}(x, y) \mid x \in S_1, y \in S_2\}$

Distance Clustering Algorithm

Let $C = \{\{v_1\}, \{v_2\}, \ldots, \{v_n\}\}$; $T = \{\}$

while $|C| > K$
  
  Let $e = (u, v)$ with $u$ in $C_i$ and $v$ in $C_j$ be the minimum cost edge joining distinct sets in $C$
  
  Replace $C_i$ and $C_j$ by $C_i \cup C_j$
Shortest paths in directed graphs vs undirected graphs

What about the minimum spanning tree of a directed graph?
- Must specify the root \( r \)
- Branching: Out tree with root \( r \)

Finding a minimum branching

Another MST Algorithm
- Choose minimum cost edge into each vertex
- Merge into components
- Repeat until done

Idea for branching algorithm
- Select minimum cost edge going into each vertex
- If graph is a branching then done
- Otherwise collapse cycles and repeat