Midterm II: T 3/27 in class
(Only graph algorithms)

CS4800: Algorithms & Data
Jonathan Ullman

Lecture 14:
• Minimum Spanning Trees

Feb 27, 2018
Minimum Spanning Trees
Network Design

- We have a set of locations $V = \{v_1, \ldots, v_n\}$
- Want to build a network to connect these locations
  - Every $v_i, v_j$ must be connected
  - Must be as cheap as possible
- Many variants
  - Build a “cheap” network that is “well connected”
  - Recall the bus routes problem from HW2
Minimum Spanning Trees (MST)

• **Input:** a graph \( G = (V, E, \{w_e\}) \)
  - Undirected, connected, weights may be negative
  - All edge weights are distinct (makes life much simpler)

• **A spanning tree** is a tree \( T \) that connects all of \( V \)
  - **Cost** of a tree \( T \) is the sum of the edge weights
    \[ \sum_{e \in T} w_e = \text{cost}(T) \]

• **Output:** A spanning tree \( T \) of minimum cost

**notation:** \( T^\ast \) is the MST
MST is not unique in general

If all $w_e$ are distinct then MST is unique
Minimum Spanning Trees (MST)

cannot be in MST

must be in the MST
Minimum Spanning Trees (MST)

\[
\text{Cost}(T^*) = 6 + 5 + 7 + 9 + 15 + 8 + 3 = 53
\]
MST Algorithms

• There are at least four reasonable MST algorithms
  • Borůvka’s Algorithm: start with $T = \emptyset$, in each round add cheapest edge out of each connected component
  • Prim’s Algorithm: start with some $s$, at each step add cheapest edge that grows the connected component
  • Kruskal’s Algorithm: start with $T = \emptyset$, consider edges in ascending order, adding edges unless they create a cycle
  • Reverse-Kruskal: start with $T = E$, consider edges in descending order, deleting edges unless it disconnects
Cycles and Cuts

• **Cycle**: a set of edges \((v_1, v_2), (v_2, v_3), \ldots, (v_k, v_1)\)

  ![Graph with a cycle](image)

  Cycle \(C = (1,2),(2,3),(3,4),(4,5),(5,6),(6,1)\)

• **Cut**: a subset of nodes \(S \subseteq V\)

  ![Graph with a cut](image)

  \[\text{Cut set}(S) : \forall e = (v, w) : u \notin S \lor v \notin S\]

  Cut \(S = \{4, 5, 8\}\)

  Cutset \(= \{5,6\}, \{5,7\}, \{3,4\}, \{3,5\}, \{7,8\}\)
Cycles and Cuts

• **Fact:** a cycle and a cutset intersect in an even number of edges

\[ V_1 \rightarrow V_2 \rightarrow V_3 \rightarrow V_4 \rightarrow V_5 \]

Every time the cycle leaves \( S \), it must come back to \( S \).
Properties of MSTs

• **Cut Property:** Let $S$ be a cut. Let $e$ be the minimum weight edge cut by $S$. Then the MST $T^*$ contains $e$
  • We call such an $e$ a **safe edge**

• **Cycle Property:** Let $C$ be a cycle. Let $f$ be the maximum weight edge in $C$. Then the MST $T^*$ does not contain $f$.
  • We call such an $f$ a **useless edge**
Proof of Cut Property

- **Cut Property**: Let $S$ be a cut. Let $e$ be the minimum weight edge cut by $S$. Then the MST $T^*$ contains $e$.

**Proof:**
- Let $T^*$ be the MST.
- Assume $e \not\in T^*$.
- Consider adding $e$ to $T^*$ (now there is a cycle containing $e$).
- There must be some $f \not\in \text{Cutset}(S) \cap \text{Cycle}$.
  
  $W_f > W_e$

  - $T$ is still a spanning tree.
  
  Let $T = T^* + (\{e\} - \{f\})$

  \[
  \text{cost}(T) = \text{cost}(T^*) + u_e - u_f < \text{cost}(T^*)
  \]

  **Contradiction!**
Proof of Cycle Property

- **Cycle Property**: Let $C$ be a cycle. Let $f$ be the maximum weight edge in $C$. Then the MST $T^*$ does not contain $f$.
  - Suppose $T^*$ contains $f$.
  - Suppose we delete $f$ from $T^*$, now $T^*$ has 2 connected components.
    - let $S$ be one of those components.
  - $f \in \mathcal{C} \cap \text{Cutset}(S)$, so there is $e \neq f \in \mathcal{C} \cap \text{Cutset}(S), w_e < w_f$.
  - Let $T = T^* - \{f\} + \{e\}$ - $T$ is spanning tree
    - $\text{cost}(T) < \text{cost}(T^*)$ contradiction.
Ask the Audience

• Assume $G$ has distinct edge weights

• **True/False?** If $e$ is the edge with the smallest weight, then $e$ is always in the MST $T^*$

• **True/False?** If $e$ is the edge with the largest weight, then $e$ is never in the MST $T^*$

> Cut Property

$e$ is the min ut.
edge in Cutset ($euv$)
The “Only” MST Algorithm

• **GenericMST:**
  • Let $T = \emptyset$
  • Until $T$ is connected:
    • Find one or more safe edges not in $T$
    • Add safe edges to $T$

• **Theorem:** **GenericMST** outputs an MST
  
  - $T \subseteq T^*$ (b/c $T$ only contains safe edges)
  - $T = T^*$ (if $T$ were not connected, there would be $\geq 2$ connected components $\Rightarrow$ a new safe edge)
Borůvka’s Algorithm

• **Borůvka:**
  • Let $T = \emptyset$
  • Until $T$ is connected:
    • Let $C_1, \ldots, C_k$ be the connected components of $(V, T)$
    • Let $e_1, \ldots, e_k$ be the safe edge for the cuts $C_1, \ldots, C_m$
    • Add $e_1, \ldots, e_k$ to $T$

• Correctness: every edge we add is safe
Borůvka’s Algorithm

Label Connected Components
Borůvka’s Algorithm

Add Safe Edges
Borůvka’s Algorithm

Label Connected Components
Borůvka’s Algorithm

Add Safe Edges
Borůvka’s Algorithm

Done!
Borůvka’s Algorithm (Running Time)

• **Borůvka**
  • Let $T = \emptyset$
  • Until $T$ is connected:
    • Let $C_1, \ldots, C_k$ be the connected components of $(V, T)$
    • Let $e_1, \ldots, e_k$ be the safe edge for the cuts $C_1, \ldots, C_m$
    • Add $e_1, \ldots, e_k$ to $T$

• How long to find safe edges?
• How many times through the main loop?
Borůvka’s Algorithm (Running Time)

**FindSafeEdges:**

- Find connected components $C_1, ..., C_k$
- Let $L[v]$ be the connected component of node $v$
- Let $S[i]$ be the safe edge of $C_i$

For each edge $(u,v)$:

- If $L[u] \neq L[v]$
  - If $w(u,v) < w(S[L[u]])$:
    - $S[L[u]] = (u,v)$
  - If $w(u,v) < w(S[L[v]])$:
    - $S[L[v]] = (u,v)$

Return \{S[1], ..., S[k]\}

**Total Time:** $O(m)$
Borůvka’s Algorithm (Running Time)

\( k \) components \( \rightarrow \frac{k}{2} \) components

• Claim: every iteration of the main loop halves the number of connected components.

  • Every time we add a safe edge, # of components decreases by 1.
  • Suppose there are \( k \) components \( C_1, \ldots, C_k \).
  • \( \exists S[1], \ldots, S[k/2] \) contains at least \( k/2 \)
divisor safe edges
    - any edge only is eligible for two components
  • \( \#CC \leq k - 1 \) \( \{S[1], \ldots, S[k/2]\} \leq k - \frac{k}{2} = \frac{k}{2} \).
Borůvka’s Algorithm (Running Time)

• **Borůvka**
  • Let $T = \emptyset$
  • Until $T$ is connected:
    • Let $C_1, \ldots, C_k$ be the connected components of $(V, T)$
    • Let $e_1, \ldots, e_k$ be the safe edge for the cuts $C_1, \ldots, C_m$
    • Add $e_1, \ldots, e_k$ to $T$

• How long to find safe edges? $O(m)$
• How many times through the main loop? $O(\log n)$

Running time: $O(m \log n)$
Prim’s Algorithm

- **Prim Informal**
  - Let $T = \emptyset$. Let $s$ be some arbitrary node and $S = \{s\}$.
  - Find the cheapest edge $e = (u, v)$ cut by $S$. Add $e$ to $T$ and add $v$ to $S$.

- **Correctness**: every edge we add is safe

[Diagram: Assuming $u \in S$ and $v \notin S$]
Prim’s Algorithm
Prim's Algorithm

Let $Q$ be a priority queue storing $V$

- key[$v$] $\leftarrow \infty$, last[$v$] $\leftarrow \perp$
- key[s] $\leftarrow 0$ for some arbitrary $s$

While $Q \neq \emptyset$:

- $u \leftarrow \text{ExtractMin}(Q)$ (assume $u$ has been found already)

For each edge $(u,v)$:

- If $v \in Q$ and $w(u,v) < \text{key}[v]$:
  - DecreaseKey($v$, $w(u,v)$) $m$ times $O(m \log n)$
  - last[$v$] $\leftarrow u$

Output $T = \{(1, \text{last}[1]), \ldots, (n, \text{last}[n])\}$ (excluding $s$)

$PQ$ (Heap): Stores key-value pairs

- Extract Min in $O(\log n)$ time
- Decrease Key

**Invariant:**
- $Q$ holds $S^c$
- $\text{value}[v]$: minimum edge from $S$ to $v$.

Running time is $O(m \log n)$
Kruskal’s Algorithm

- **Kruskal’s Informal**
  - Let \( T = \emptyset \)
  - Consider edges \( e \) in ascending order of weight:
    - If adding \( e \) would merge two connected components
      - Add \( e \) to \( T \)
  - \( O(n \log n) \) time to merge

- **Correctness**: every edge we add is safe

- **Running**: \( O(m \log n) \)
  - \( O(m) \) time to check
  - \( O(m \log m) \) to sort

Suppose \( e \) connects \( C_1 \) to \( C_2 \)

Suppose \( w_f < w_e \), then would have already added \( f \) to \( T \)
Kruskal’s Algorithm
Implementing Kruskal’s Algorithm

• **Union-Find Data Structure**

• Need to store the set of connected components in such a way that we can efficiently:
  - **Check** if u,v are in the same component (Find(u), Find(v))
  - **Merge** the connected components of u,v (Union(u,v))

• Can implement Union-Find so that
  - Find(u) takes $O(1)$ time
  - Any $k$ operations Union(u,v) take $O(k \log k)$ time
    - “Amortized Analysis”

• *Lots of fancier versions of this*
Kruskal’s Algorithm (Running Time)

• **Kruskal’s Informal**
  • Let $T = \emptyset$
  • Consider edges $e$ in ascending order of weight:
    • If adding $e$ would merge two connected components
      • Add $e$ to $T$

• Time to sort:
• Time to test edges:
• Time to add edges:
Comparison

• **Boruvka’s Algorithm:**
  • Only algorithm worth implementing
  • Low overhead, can be easily parallelized
  • Each iteration takes $O(m)$, very few iterations in practice

• **Prim’s/Kruskal’s Algorithms:**
  • Reveal useful structure of MSTs
  • Running time dominated by a single sort