Algorithm Design and Analysis

Lecture 10

• Implementing MST Algorithms

Adam Smith

A. Smith; based on slides by E. Demaine, C. Leiserson, S. Raskhodnikova, K. Wayne
Minimum spanning tree (MST)

Input: A connected undirected graph \( G = (V, E) \) with weight function \( w : E \to \mathbb{R} \).
- For now, assume all edge weights are distinct.

Output: A spanning tree \( T \) — a tree that connects all vertices — of minimum weight:

\[
w(T) = \sum_{(u,v) \in T} w(u, v).
\]
Greedy Algorithms for MST

• **Kruskal's:** Start with $T = \emptyset$. Consider edges in ascending order of weights. Insert edge $e$ in $T$ unless doing so would create a cycle.

• **Reverse-Delete:** Start with $T = E$. Consider edges in descending order of weights. Delete edge $e$ from $T$ unless doing so would disconnect $T$.

• **Prim's:** Start with some root node $s$. Grow a tree $T$ from $s$ outward. At each step, add to $T$ the cheapest edge $e$ with exactly one endpoint in $T$. 
Cut and Cycle Properties

- **Cut property.** Let $S$ be a subset of nodes. Let $e$ be the min weight edge with exactly one endpoint in $S$. Then the MST contains $e$.

- **Cycle property.** Let $C$ be a cycle, and let $f$ be the max weight edge in $C$. Then the MST does not contain $f$. 

$e$ is in the MST

$f$ is not in the MST
Prim's Algorithm: Correctness

- Prim's algorithm. [Jarník 1930, Prim 1959]
  - Initialize $S = \text{any node}$.  
  - Apply cut property to $S$.  
  - When edge weights are distinct, the edge that is added must be in the MST  
  - Thus, Prim’s alg. outputs the MST
Correctness of Kruskal

- [Kruskal, 1956]: Consider edges in ascending order of weight.
  - **Case 1**: If adding $e$ to $T$ creates a cycle, discard $e$ according to cycle property.
  - **Case 2**: Otherwise, insert $e = (u, v)$ into $T$ according to cut property where $S =$ set of nodes in $u$'s connected component.
Non-distinct edges?
Implementation of Prim(G,w)

**Idea:** Maintain $V - S$ as a priority queue $Q$ (as in Dijkstra). Key each vertex in $Q$ with the weight of the least-weight edge connecting it to a vertex in $S$.

$Q \leftarrow V$

$key[v] \leftarrow \infty$ for all $v \in V$

$key[s] \leftarrow 0$ for some arbitrary $s \in V$

while $Q \neq \emptyset$

- do $u \leftarrow \text{EXTRACT-MIN}(Q)$
  - for each $v \in \text{Adjacency-list}[u]$
    - do if $v \in Q$ and $w(u, v) < key[v]$
      - then $key[v] \leftarrow w(u, v)$ $\triangleright \text{DECREASE-KEY}$
      - $\pi[v] \leftarrow u$

At the end, $\{(v, \pi[v])\}$ forms the MST.
Analysis of Prim

Handshaking Lemma $\Rightarrow \Theta(m)$ implicit decrease-key’s.

Time: as in Dijkstra
### Analysis of Prim

While $Q \neq \emptyset$

1. Do $u \leftarrow \text{EXTRACT-MIN}(Q)$
2. For each $v \in \text{Adj}[u]$
   1. Do if $v \in Q$ and $w(u, v) < \text{key}[v]$
      1. Then $\text{key}[v] \leftarrow w(u, v)$
      2. $\pi[v] \leftarrow u$

Handshaking Lemma $\Rightarrow \Theta(m)$ implicit \textsc{Decrease-Key}'s.

<table>
<thead>
<tr>
<th>PQ Operation</th>
<th>Prim</th>
<th>Array</th>
<th>Binary heap</th>
<th>d-way Heap</th>
<th>Fib heap $^\dagger$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExtractMin</td>
<td>n</td>
<td>n</td>
<td>log n</td>
<td>HW3</td>
<td>log n</td>
</tr>
<tr>
<td>DecreaseKey</td>
<td>m</td>
<td>1</td>
<td>log n</td>
<td>HW3</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td>$n^2$</td>
<td>m log n</td>
<td>m log $\frac{m}{n}$</td>
<td>$m + n \log n$</td>
<td></td>
</tr>
</tbody>
</table>

$^\dagger$ Individual ops are amortized bounds

---

A. Smith; based on slides by E. Demaine, C. Leiserson, S. Raskhodnikova, K. Wayne
Greedy Algorithms for MST

• **Kruskal's**: Start with $T = \emptyset$. Consider edges in ascending order of weights. Insert edge $e$ in $T$ unless doing so would create a cycle.

• **Reverse-Delete**: Start with $T = E$. Consider edges in descending order of weights. Delete edge $e$ from $T$ unless doing so would disconnect $T$.

• **Prim's**: Start with some root node $s$. Grow a tree $T$ from $s$ outward. At each step, add to $T$ the cheapest edge $e$ with exactly one endpoint in $T$. 
## Union-Find Data Structures

### Operation \ Implementation

<table>
<thead>
<tr>
<th></th>
<th>Array + linked-lists and sizes</th>
<th>Balanced Trees</th>
</tr>
</thead>
<tbody>
<tr>
<td>Find (worst-case)</td>
<td>$\Theta(1)$</td>
<td>$\Theta(\log n)$</td>
</tr>
<tr>
<td>Union of sets A,B</td>
<td>$\Theta(\min(</td>
<td>A</td>
</tr>
<tr>
<td>Amortized analysis: k unions and k finds, starting from singletons</td>
<td>$\Theta(k \log k)$</td>
<td>$\Theta(k \log k)$</td>
</tr>
</tbody>
</table>

- With modifications, amortized time for tree structure is $O(n \text{ Ack}(n))$, where $\text{Ack}(n)$, the Ackerman function grows much more slowly than $\log n$.
- See KT Chapter 4.6