Minimum Spanning Trees (Chapter 23)

• Connected Undirected graph
• Edges have weights
• Find minimum weight subset of edges that connects every vertex and is acyclic
  - Must be connected
  - Does not need to be a path
    + Hence the use of the term spanning tree

*Do we need to require the min spanning tree to be acyclic?
*What about an unweighted graph?
*What might this be used for?
*How fast might this be?
Overview of Chapter

• Lay ground work
  - Generic minimum spanning tree method

• Contrast two algorithms
  - Kruskal’s algorithm
  - Prim’s algorithm

• Both make use of the generic method

• Both are greedy algorithms
  - *Greedy strategy advocates making the choice that is best at the moment*
  - Can prove their greedy strategy is optimal
Overview

⇒ Growing a Min Spanning Tree
• Kruskal’s Algorithm
• Prim’s Algorithm
Growing a Min-Spanning Tree

• Let $G = (V, E)$ with a weight function $w : E \rightarrow \mathbb{R}$

• Generic Method for growing a tree
  - Grow set $A$ (a set of edges) starting with $A = \emptyset$
  - Loop invariant:
    $A$ is a subset of some minimum spanning tree
  - At each step, determine an edge $(u, v)$ that we can add to $A$ that maintains loop invariant:
    $+ \{(u, v)\} \cup A$ is a subset of a min spanning tree
  - Such an edge is called a *safe edge*

*Greedy?*  
*Optimal?*  
*Difficulty?*
• Definitions:
  - **Cut** \((S, V-S)\) of an undirected graph \(G = (V, E)\) is a partition of \(V\)
  - Edge \((u, v) \in E\) **crosses** cut \((S, V-S)\) if one vertex in \(S\) other in \(V-S\)
  - A cut **respects** a set of edges \(A\) if no edge in \(A\) crosses the cut
  - Edge is a **light edge** crossing a cut if its weight is min of any edge crossing cut. Can be ties.

© P. Heeman, 2017
Finding Safe Edges

- Definitions:
  - **Cut** \((S, V-S)\) of an undirected graph \(G = (V, E)\) is a partition of \(V\)
  - Edge \((u, v) \in E\) crosses cut \((S, V-S)\) if one vertex in \(S\) other in \(V-S\)
  - A cut respects a set of edges \(A\) if no edge in \(A\) crosses the cut
  - Edge is a **light edge** crossing a cut if its weight is min of any edge crossing cut. Can be ties.

**Theorem 23.1**
Let \(G = (V, E)\) be a connected, undirected graph with \(w : E \rightarrow \mathbb{R}\).
Let \(A\) be subset of \(E\) that is included in some min-spanning tree for \(G\).
Let \((S, V-S)\) be any cut of \(G\) that respects \(A\).
Let \((u, v)\) be a light edge crossing \((S, V-S)\).
Then edge \((u, v)\) is safe for \(A\)
Theorem 23.1: In Plain English

• If you have $A$, a partial min-spanning tree for $G$
  - Pick some vertices to make a cut that respects $A$
  - Find light edge: edge of min weight that crosses cut
  - It is a safe edge

• Questions:
  - Is it hard to find such a cut?
  - How should we pick cut?
  - Do we evolve the cut, or pick a fresh cut each time?

• Any algorithm that follows Theorem 23.1
  - Will find minimum spanning tree
  - Will it be greedy?
Proof (Not by contradiction)

Let $T$ be a min-spanning tree that includes $A$
Let $(S, V- S)$ be a cut that respects $A$
Let $(u, v)$ be a light edge that crosses the cut

**Case 1:** $(u, v) \in T$. Done

**Case 2:** $(u, b) \notin T$
There is a simple path $p$ in $T$ from $u$ to $v$ since $T$ is a min spanning tree $(u, v)$ and $p$ forms a cycle
$u$ and $v$ are on opposite sides of the cut $(S, V - S)$ that respects $A$
Hence, at least one other edge in $p$, say $(x, y)$, must cross cut.
$(x, y)$ is not in $A$, since the cut respects $A$ (so it was added later).
Create $T'$ by removing $(x, y)$ and adding $(u, v)$.
$T'$ will be a spanning tree.
Since $(u, v)$ is light, its weight must be same as $(x, y)$.
So $T'$ will also be a min spanning tree.
Another way to look at the Algorithm

\[ G = (V, E) \text{ and } A \subseteq E \]

• Define \( G_A = (V, A) \) (the graph with the edges \( A \))
  - \( G_A \) is a forest
  - Each connected component in \( G_A \) is a tree (no cycles, connected)

• Algorithm
  - Start with \( A = \emptyset \). \( G_A \) has \(|V| \) trees
  - Any safe edge \((u, v)\) will connect two distinct components of \( G_A \)
    + regardless of how a cut is chosen
  - Each iteration reduces the number of components by 1
  - Ends when there is just one component
Corollary 23.2

Let $G = (V, E)$ be a connected, undirected graph with $w : E \rightarrow \mathbb{R}$
Let $A$ be a subset of $E$ that is in some min spanning tree of $G$
Let $C = (V_C, E_C)$ be a connected component (tree) in forest $G_A = (V, A)$.
If $(u, v)$ is a light edge between $C$ and some other component in $G_A$, then $(u, v)$ is safe for $A$

• Questions
- What is the cut that the light edge is crossing?
- How does this restrict the previous algorithm
  + How does this restrict what cut is used?
- What choices are left?
Overview

• Growing a Min Spanning Tree
  ⇒ Kruskal’s Algorithm
• Prim’s Algorithm
Kruskal’s Algorithm

• Pick edge \((u, v)\) of min weight connecting any two trees in \(G_A\)
  - Can this be proved correct by Corollary 23.2?
Kruskal’s Algorithm

• Pick edge \((u, v)\) of min weight connecting any two trees in \(G_A\)
  - Can this be proved correct by Corollary 23.2?
  
    + You actually pick the edge first, say \((u, v)\)
    + Then you specify what \(C_1\) is: component that \(u\) is in
    + Since \((u, v)\) is minimum, it is a min edge coming out of \(C_1\)
MST-KRUSKAL\((G, w)\)

1. \(A = \emptyset\)
2. for each vertex \(v \in G.V\)
   3. MAKE-SET\((v)\)
4. sort the edges of \(G.E\) into nondecreasing order by weight \(w\)
5. for each edge \((u, v) \in G.E\), taken in nondecreasing order by weight
   6. if FIND-SET\((u) \neq FIND-SET(v)\)
      7. \(A = A \cup \{(u, v)\}\)
      8. UNION\((u, v)\)
6. return \(A\)

- What data structures does it need?
MST-Kruskal(G, w)
1  \( A = \emptyset \)
2  \textbf{for} each vertex \( v \in G.V \)
3  \textbf{MAKE-SET}(v)
4  sort the edges of \( G.E \) into nondecreasing order by weight \( w \)
5  \textbf{for} each edge \( (u, v) \in G.E \), taken in nondecreasing order by weight
6  \textbf{if} \textbf{ FIND-SET}(u) \neq \textbf{ FIND-SET}(v)
7  \hspace{1em} A = A \cup \{(u, v)\}
8  \hspace{1em} \textbf{UNION}(u, v)
9  \textbf{return} A

• What data structures does it need?
  + Components: disjoint sets (forest implementation is faster)
  + Edges: sort it into an array (min priority queue not needed)
  + Min-spanning edges: array
Example

© P. Heeman, 2017
Running Time

- **Lines 4: $O(E \log E)$**
- **Set operations**
  - $O(V)$ make-set
  - $O(E)$ find-set
  - $O(V)$ union
  - $O(V + E)$ operations
- Graph is fully connected so $|E| \geq |V| - 1$
- So $O(E)$ operations
- From Chapter 21.4: $O(E \log E)$

- **Overall $O(E \log E) = O(E \log V)$**
  - $|E| < |V|^2$

---

MST-KRUSKAL($G, w$)

```python
1   A = ∅
2   for each vertex $v \in G.V$
3       MAKE-SET($v$)
4   sort the edges of $G.E$ into nondecreasing order by weight $w$
5   for each edge $(u, v) \in G.E$, taken in nondecreasing order by weight
6       if FIND-SET($u$) $\neq$ FIND-SET($v$)
7         $A = A \cup \{(u, v)\}$
8       UNION($u, v$)
9   return $A$
```
Overview

• Growing a Min Spanning Tree
• Kruskal’s Algorithm
⇒ Prim’s Algorithm
Prim’s Algorithm

• Edges in $A$ always form a single tree
  - Start with an arbitrary vertex
  - Add a light edge that connects $A$ to a new vertex

• To determine next vertex to add
  - Go through all edges to find min connecting $A$ to a vertex not in $A$
  - Can easily be $O(VE)$
A Better Way

• For each vertex not in A
  - Keep its min weight/edge to any vertex in A
  - No edge to A: use $\infty$

• When new vertex $u$ added to A
  - For all vertices $v$ adjacent to $u$, see if $u$ provides better way to $A$ (via $u$)
  - If it does, update its weight/edge

• Need to update weights: min-priority queue
MST-PRIM\((G, w, r)\)

1. \textbf{for} each \(u \in G.V\)
2. \(u.key = \infty\)
3. \(u.\pi = \text{NIL}\)
4. \(r.key = 0\)
5. \(Q = G.V\)
6. \textbf{while} \(Q \neq \emptyset\)
7. \(u = \text{EXTRACT-MIN}(Q)\)
8. \textbf{for} each \(v \in G.Adj[u]\)
9. \hspace{1em} \textbf{if} \(v \in Q\) and \(w(u, v) < v.key\)
10. \hspace{1em} \(v.\pi = u\)
11. \hspace{1em} \(v.key = w(u, v)\)
Example
Running Time

• Use min priority queue
  - $O(V)$ to build heap. Why?

- For each vertex, for each edge

- Updating key

MST-PRIM($G, w, r$)

1. for each $u \in G.V$
2. $u.key = \infty$
3. $u.\pi = \text{NIL}$
4. $r.key = 0$
5. $Q = G.V$
6. while $Q \neq \emptyset$
7. $u = \text{EXTRACT-MIN}(Q)$
8. for each $v \in G.\text{Adj}[u]$
9. if $v \in Q$ and $w(u, v) < v.key$
10. $v.\pi = u$
11. $v.key = w(u, v)$