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 $w : E \to \mathbb{R}$
• 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?
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.

![Diagram of a graph with edges and vertices labeled]

**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 \(A\)
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: $(u, v) \not\in T$.

There is a simple path $p$ in $T$ from $u$ to $v$ since $T$ is a min spanning tree.

$p$ forms a cycle.

$u$ and $v$ are on opposite sides of the cut $(S, V - S)$.

Hence, at least one other edge in $p$, say $(x, y)$, must cross the cut.

$(x, y)$ is not in $T$, since the cut respects $A$.

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 the same as $(x, y)$.

So $T'$ will also be a min spanning tree.

---

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?
- Will it be greedy?

- Any algorithm that follows Theorem 23.1.

- Will find minimum spanning tree.
Another way to look at the Algorithm

\begin{itemize}
  \item Define \( G_A = (V, A) \) (the graph with the edges \( A \))
  \begin{itemize}
    \item \( G_A \) is a forest
    \item Each connected component in \( G_A \) is a tree (no cycles, connected)
  \end{itemize}
  \item Algorithm
  \begin{itemize}
    \item Start with \( A = \emptyset \). \( G_A \) has \( |V| \) trees
    \item Any safe edge \((u, v)\) will connect two distinct components of \( G_A \)
      \begin{itemize}
        \item regardless of how a cut is chosen
      \end{itemize}
    \item Each iteration reduces the number of components by 1
    \item Ends when there is just one component
  \end{itemize}
\end{itemize}

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 \( A \)

\begin{itemize}
  \item Questions
    \begin{itemize}
      \item What is the cut that the light edge is crossing?
      \item How does this restrict the previous algorithm
      \begin{itemize}
        \item How does this restrict what cut is used?
      \end{itemize}
      \item What choices are left?
    \end{itemize}
\end{itemize}
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?
    + 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 \(C_1\)
Example

Code

MST-KRUSKAL(G, w)
1 $A = \emptyset$
2 for each vertex $v \in G.V$
3 \hspace{1em} MAKE-SET(v)
4 \hspace{1em} 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 \hspace{1em} if $\text{FIND-SET}(u) \neq \text{FIND-SET}(v)$
7 \hspace{2em} $A = A \cup \{(u, v)\}$
8 \hspace{1em} UNION(u, v)
9 \hspace{1em} 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
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 $O(E \log E) = O(E \log V)$
- $|E| < |V|^2$

Overview

• Growing a Min Spanning Tree
• Kruskal’s Algorithm
  $\Rightarrow$ 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
**Code**

\[
\text{MST-PRIM}(G, w, r)
\]

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{2em} \( v.\pi = u \)
11. \hspace{2em} \( v.key = w(u, v) \)

---

**Example**

![Graph](image)
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. \hspace{1em} if $v \in Q$ and $w(u, v) \lt v.key$
10. \hspace{2em} $v.\pi = u$
11. \hspace{1em} $v.key = w(u, v)$

$O(V)$ to build heap. Why?