Chapter 24: Minimum Spanning Tree

\( G = (V, E) \): a connected (undirected) graph
\( w \): an integer edge weight function
compute a **minimum-weight spanning tree** of \( G \), where the weight of a tree \( T \), denoted \( w(T) \), is \( \sum_{e \in T} w(e) \)

![Graph with weights](image)

**Theorem A** \( G = (V, E) \): a connected (undirected) graph
\( w \): an integer edge weight function
\( A \): expandable to an MST
\( (S, V - S) \): a cut respecting \( A \)
\( e = (u, v) \): a light edge crossing the cut
Then \( e \) is safe for \( A \).

**Proof** Let \( T \) be an MST containing \( A \). There is a unique path in \( T \) from \( u \) to \( v \). Pick from the path an edge crossing the cut, say \( d \). Replacing \( d \) by \( e \) generates a spanning tree \( T' \). Here \( w(T') = w(T) \) (because \( e \) is light) so \( T' \) is an MST, and \( A \cup \{e\} \) is expandable to \( T' \). Thus \( e \) is safe for \( A \).

**Corollary B** Every light edge connecting two distinct components in \( G_A = (V, A) \) is safe for \( A \).

**Safe edges and cuts**
\( e \in E - A \) is **safe** for \( A \) if \( A \cup \{e\} \) is expandable to an MST or an MST already

a **cut** of \( G \): a partition \((S, V - S)\) of \( V \)
an edge \( e \) **crosses** \((S, V - S)\) if \( e \) connects a node in \( S \) and one in \( V - S \)
\((S, V - S)\) **respects** \( A \subseteq E \) if no edges in \( A \) cross the cut

For any edge property \( Q \), an **light edge** w.r.t. \( Q \) is one with the smallest weight among those with the property \( Q \)

**Kruskal’s Algorithm**

Maintain a collection of connected components and construct an MST \( A \).

Initially, each node is a connected component and \( A = \emptyset \).

Examine all the edges in the **nondecreasing order of weights**.

- If the current edge connects two different components, add \( e \) to \( A \) to unite the two components.

The added edge is a light edge; otherwise, an edge with smaller weight should have already united the two components.
Implementation with “disjoint-sets”

1. $A \leftarrow \emptyset$
2. for each vertex $v \in V$ do
   3. Make-Set($v$)
   4. reorder the edges so there weights are in nondecreasing order
   5. for each edge $(u, v) \in E$ in the order do
      6. if Find-Set($u$) $\neq$ Find-Set($v$) then
         7. $A \leftarrow A \cup \{(u, v)\}$
         8. Union($u$, $v$)
3. return $A$

The number of disjoint-set operations that are executed is $2E + 2V - 1 = O(E)$, out of which $V$ are Make-Set operations. So, the total cost for disjoint-set operations is: cost:

$$O(E \lg^* V) = O(E \lg V).$$

Q3. So, the total running time is?

Q1. Assuming that the length of $w$ is a word-length, what is the complexity of sorting?

Q2. What is the complexity of initialization?
**Prim's algorithm**

Maintain a set of edges $\mathcal{A}$ and a set of nodes $\mathcal{B}$. Fix any node $r$ as the root and set $\mathcal{B}$ to $\{r\}$. Set $\mathcal{A}$ to $\emptyset$. Then repeat the following until $\mathcal{A} = \mathcal{V} - 1$.

- Find a light edge $e$ connecting a node in $\mathcal{B}$ to one $\mathcal{V} - \mathcal{B}$. Let $e = (u, v)$ with $u \in \mathcal{B}$ and $v \notin \mathcal{B}$.
- Put $e$ in $\mathcal{A}$ and $v$ in $\mathcal{B}$.

Implement the algorithm with a priority queue $\mathcal{Q}$ of nodes in $\mathcal{V} - \mathcal{B}$, based on the key $\text{key}[v]$, where $\text{key}[v]$ is the minimum edge weight connecting $v$ to a node in $\mathcal{B}$.

By convention, $\text{key}[v] = \infty$ if there is no such edge.

Each node $v$ has a field $\pi[v]$ as the “parent,” that is, the node $u$ such that $(u, v)$ was a light edge when $v$ is added.

An implicit definition of $\mathcal{A}$ is

\[ \{(v, \pi[v]) \mid v \in \mathcal{V} - \{r\} - \mathcal{Q} \}. \]

1. $Q \leftarrow V$
2. \textbf{for} each $u \in Q$ \textbf{do} $\text{key}[u] \leftarrow \infty$
3. $\text{key}[r] \leftarrow 0$
4. $\pi[r] \leftarrow \text{NIL}$
5. \textbf{while} $Q \neq \emptyset$ \textbf{do}
6. \hspace{1em} $u \leftarrow \text{Extract-Min}(Q)$
7. \hspace{1em} \textbf{for} each $v \in \text{Adj}[u]$ \textbf{do}
8. \hspace{2em} \textbf{if} $v \in Q$ and $w(u, v) < \text{key}[v]$ \textbf{then}
9. \hspace{3em} $\pi[v] \leftarrow u$
10. \hspace{3em} $\text{key}[v] \leftarrow w(u, v)$

Line 3 forces to select $r$ first. Lines 7-10 are for updating the keys.

Implement $Q$ using a heap. The running time is

\[ V \cdot \text{the cost of Build-Heap} \]
\[ + (\mathcal{V} - 1) \cdot \text{the cost of Extract-Min} \]
\[ + E \cdot \text{the cost of Decrease-Key}. \]
If a binary heap or a binomial heap is used, the running time is:

\[
V \cdot O(1) \\
+ (V - 1) \cdot O(|\lg V|) \\
+ E \cdot O(\lg V) \\
= O((E + V) \lg V) = O(E \lg E),
\]

which is the same as the running time of Kruskal's algorithm.

If a Fibonacci heap is used, the running time is:

\[
V \cdot O(1) \\
+ (V - 1) \cdot O(\lg V) \\
+ E \cdot O(1) \\
= O(V \lg V + E),
\]

which is better than the running time of Kruskal's algorithm.