1 Introduction

Given a connected weighted undirected graph $G$ with $V$ vertices, a spanning tree is a set of $V - 1$ edges of $G$, under which $G$ remains connected.

A minimum spanning tree is a spanning tree that minimizes the total weight of the edges in the tree.

2 Generic MST algorithm

```
CLRS 23

\[
\text{GENERICMST}(G, w) \\
T = \emptyset \\
\text{while } T \text{ is not a spanning tree do} \\
\hspace{1cm} \text{Find an edge } (u, v) \text{ that is safe to add.} \\
\hspace{1cm} T = T \cup \{(u, v)\} \\
\text{end while} \\
\text{return } T
\]
```

Invariant: Before each iteration, $T$ is a subset of some MST.

This is a greedy algorithm.

3 Why does the greedy approach work?

Corollary 23.2: Let $G = (V, E)$ be a connected, undirected graph with a real-valued weight function $w$ defined on $E$. Let $A$ be a subset of $E$ that is included in some minimum spanning tree for $G$, and let $C = (V_C, E_C)$ be a connected component (tree) in the forest $G_A = (V, A)$. If $u$ is a light edge connecting $C$ to some other component in $G_A$, then $u$ is safe for $A$.

Here the term “light edge” refers to the lowest-weight edge with that property.

Intuition: Think of the partially-completed tree as a set of connected components. If we pick one connected component, then the lightest edge that connects it to any another connected component is safe to add to the MST.

4 Kruskal’s algorithm

Idea: Add the lightest edge, across the entire graph, that does not create a cycle.

- First sort the edges by order of increasing weight.
- Use a disjoint sets data structure to test whether an edge creates a cycle.

Details: CLRS 631
5 Kruskal’s analysis

- Sorting the edges: $O(E \log E)$
- $E$ FIND operations: $O(E \alpha(V))$
- $V$ UNION operations: $O(V \alpha(V))$

Total run time:

$$T(n) = O(E \log E) + O(E \alpha(V)) + O(V \alpha(V))$$

$$= O(E \log E) + O(E \alpha(V))$$

$$= O(E \log V)$$

In the second step, we use the fact that $E \geq V - 1$, since the graph is connected. In the third step, we use the fact the $\alpha(V) = O(\log V) = O(\log E)$. In the final step, note that $\log E \leq \log V^2 = O(\log V)$.

6 Prim’s algorithm

**Idea:** Pick one node $v$ as the “root.” Add the lightest edge that connects an isolated node to the connected component containing $v$. Each node has two new attributes:

- A **parent** $v.\pi$, a pointer to another node:
  - For the root, $v.\pi = \text{nil}$.
  - For other nodes in the tree $v.\pi$ is the node that connects $v$ to the tree.
  - For nodes in the queue with finite keys, $v.\pi$ is the closest node in the tree to $v$.
  - For nodes in the queue with infinite keys, $v.\pi = \text{nil}$.

- A **key** $v.d$, the weight of the edge connecting to the parent.

Use a priority queue of all not-yet-added nodes, ordered by the $v.d$ values.

- When a node is added to the tree, perform the appropriate DECREASEKEY operations for its out-edges.

Details: CLRS 634

7 Analysis of Prim’s algorithm

With a binary heap:

- 1 **BUILDMinHEAP** operation: $O(V)$
- $V$ **EXTRACTMin** operations: $O(V \log V)$
- $E$ **DECREASEKEY** operations: $O(E \log V)$
Total run time:

\[
T(n) = O(V) + O(V \log V) + O(E \log V) \\
     = O(E \log V)
\]

With a Fibonacci heap:

- \text{V \textsc{Insert} operations: } O(V)
- \text{V \textsc{ExtractMin} operations: } O(V \log V)
- \text{E \textsc{DecreaseKey} operations: } O(E)

Total run time:

\[
T(n) = O(V) + O(V \log V) + O(E) \\
     = O(E + V \log V)
\]