# Minimum Spanning Trees (MST)

## 1 Definitions

### 1.1 Path

A path is a sequence of edges which connects a sequence of nodes.

### 1.2 Cycle

A cycle is a path with no repeated nodes or edges other than the start and end nodes.

### 1.3 Tree

A **undirected graph** is a tree if it is connected and does not contain a cycle. The basic tree theorem is stated below: Let $G$ be an undirected graph on $n$ nodes (may not be the tree), then **any two of the following statements imply the third**:

1. $G$ is connected.
2. $G$ does not contain a cycle.
3. $G$ has $n-1$ edges

### 1.4 Spanning Tree

An **undirected graph** is a **spanning tree** if it is a tree and touches **every vertex** in G. The definition of **Minimum Spanning Tree (MST)** is slightly different. Given a connected graph $G = (V, E)$ with real-valued edge weights $c_e$, an MST is a **subset of the edges** $T$ such that $T$ is a spanning tree **whose sum of edge weights is minimized**.

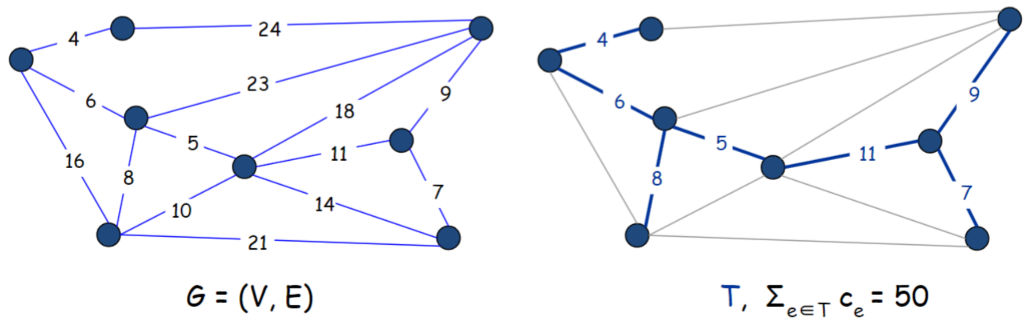


Figure 1: MST Overview

Based on **Cayley's Theorem**, there are $n^{n-2}$ spanning trees with $n$ nodes.

# 2 Properties

## 2.1 Cut

### 2.1.1 Definition

A **cut** is a partition of the nodes into two **nonempty subsets** $S$ and $V - S$. The **cutset** $D$ of a cut $S$ is the **set of edges with exactly one endpoint in** $S$.

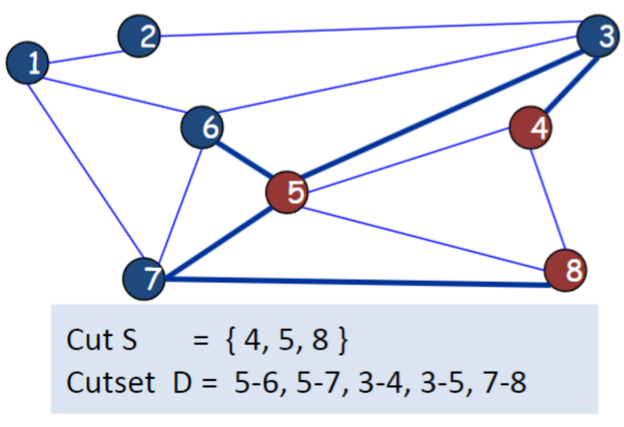


Figure 2: Cut and Cutset Definitions

### 2.1.2 Property

Assume all edge costs $c_e$ are distinct. Then consider a cut in a graph that divides the vertices into two disjoint subsets. Among the edges that cross the cut (those that have one endpoint in each subset), if the weight of an edge $e$ is the smallest, then **every MST** contains $e$.

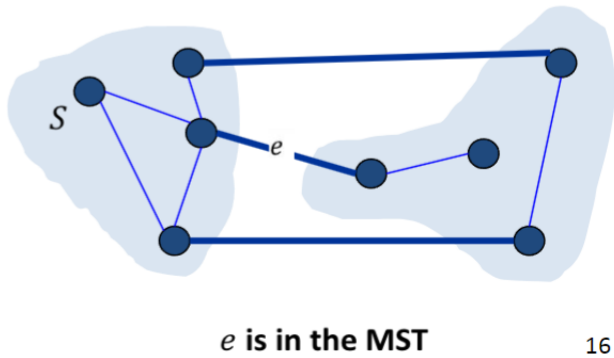


Figure 3: Cut Property

#### 2.1.3 Proof

1. Define $e$ as the minimum cost edge in the cutset for a set $S$, and define $T^*$ as an MST.
2. **Suppose** $e$ does not belong to $T^*$
3. Because $T^*$ is an MST, so the endpoints $u$ and $v$ must be connected by a path in $T^*$
4. Adding $e$ to $T^*$ will create a cycle $C$ in $T^*$, because $T^*$ is a tree, adding any edge connecting two vertices will create a cycle.
5. This cycle must contain another edge $f$ that crosses the cut $S$, because the cycle must enter and leave each part of the cut, and $e$ was one such edge crossing. We can remove $f$ to break the cycle and still have a spanning tree.
6. By the assumption of the cut property, the weight of $e$ is less than or equal to the weight of $f$, because $e$ is the minimum weight edge crossing the cut. Therefore replacing $f$ with $e$ **can not increase the total weight of the tree.**
7. This replacement results in another spanning tree $T'$ with a weight less than or equal to that of $T$. If $T$ was a MST, then $T'$ is either another MST or has a smaller weight, which contradicts the assumption that $T$ was the MST without $e$.
8. Therefore, every MST contains $e$.

## 2.2 Cycle

#### 2.2.1 Definition

Again, we assume all edge costs $c_e$ are distinct. We define cycle as a set of edges that form $a-b, b-c, c-d, ..., y-z, z-a$, as shown below:

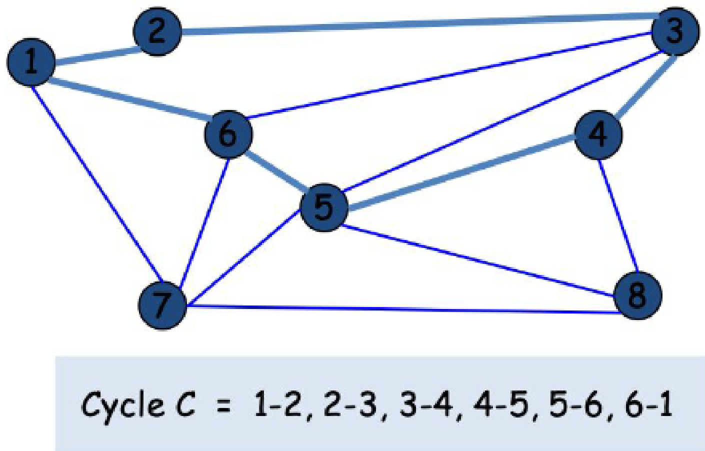


Figure 4: Cycle Definition

Now, define $C$ as any cycle, and let $f$ be the max cost edge belonging to $C$. **Then every MST does not contain $f$.**

#### 2.2.2 Proof

1. Define $f$ as the maximum weight in a cycle $C$ is included in a MST $T$.
2. Suppose $f$ is part of some cycle in the graph $G$, and $T$ includes $f$.
3. Because $f$ is part of a cycle, we can remove $f$ from $T$ without disconnecting the graph. Removing $e$ breaks the cycle and leaves a spanning tree because the rest of the graph remains connected.
4. By removing $f$, which has the maximum weight in the cycle, the total weight of the spanning tree is reduced. This new spanning tree $T'$ has a lower total weight than $T$, assuming $f$ was part of $T$.
5. This contradicts the assumption that $T$ was a MST of $G$, as $T$ included an edge $e$ that could be removed to reduce its total weight.

# 3 Prim's Algorithm

## 3.1 Definition

The Prim's Algorithm includes the following steps:

1. Start with a set 'MST' that contains a chosen starting vertex (arbitrary)
2. While 'MST' does not yet include all vertices:
   (a) Select and remove the edge with the smallest weight that **connects a vertex in MST to a vertex outside MST**

(b) Add the selected edge and vertex not in MST to MST

(c) Update the priority queue by adding the new edges that connect the newly added vertex to any vertex not yet in MST

## 3.2 Proof

### 3.2.1 Simple Version

Prim's algorithm utilizes the cut property, that the min cost edge is always inside the MST. So if we keep adding the min cost edge, we can get a MST.

## 3.3 Implementation

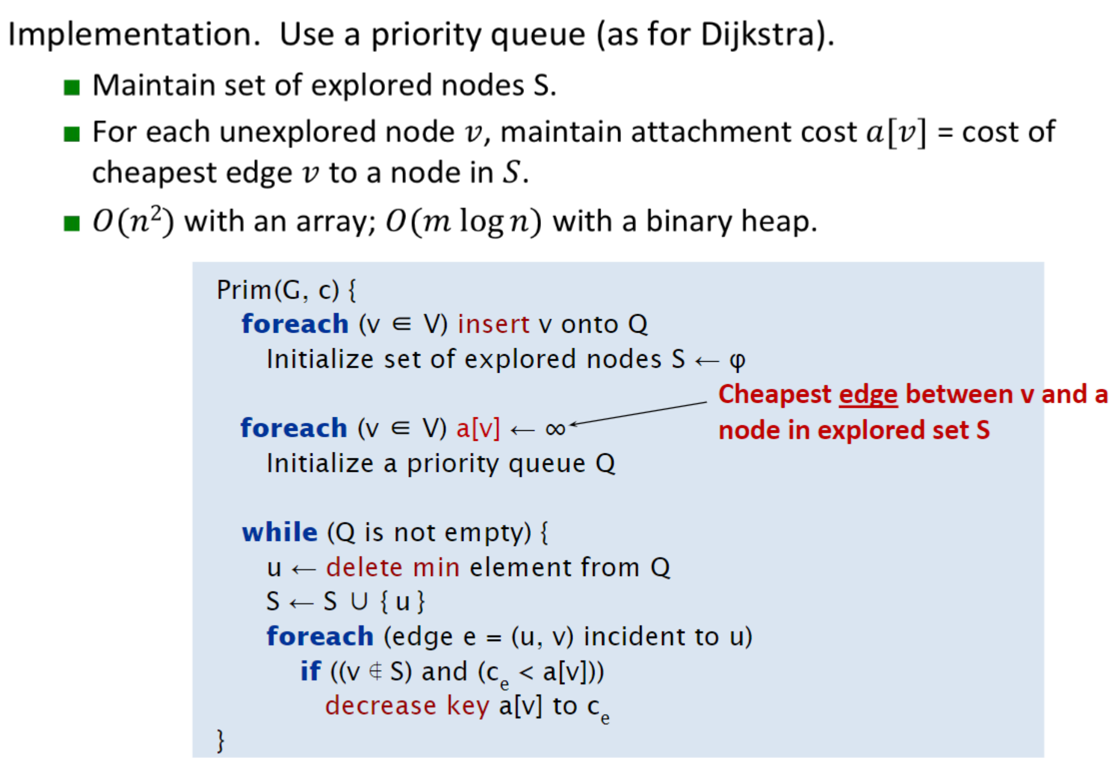


Figure 5: Prim's Algorithm Implementation

# 4 Kruskal's Algorithm

## 4.1 Definition

1. First sort all the edges of the graph in **non-decreasing** order of their weights.

2. Start with $T = \emptyset$. Insert edge $e$ in ascending order in $T$ unless doing so would create a cycle.

## 4.2 Proof

### 4.2.1 Simple Version

There are two situations when adding an edge $e$:

1. If adding $e$ creates a cycle, then skip $e$. This is valid by the cycle property.
2. If adding $e$ does not create a cycle, then add $e$. It is valid by the cut property.

## 4.3 Implementation

Implementation. Use the union-find data structure.

- Build set T of edges in the MST.
- Maintain a set for each connected component.
- $O(m \log n)$ for sorting and $O(m\, \underbrace{\alpha\,(m, n)}_{\text{essentially a constant}})$ for union-find.

$m \leq n^2 \Rightarrow \log m$ is $O(\log n)$

```
Kruskal(G, c) {
  Sort edges weights so that c_1 ≤ c_2 ≤ ... ≤ c_m.
  T ← φ

  foreach (u ∈ V) make a set containing singleton u

  for i = 1 to m                are u and v in different connected components?
    (u,v) = e_i
    if (u and v are in different sets) {
      T ← T ∪ {e_i}
      merge the sets containing u and v
    }                           merge two components
  return T
}
```

Figure 6: Kruskal's Algorithm Implementation

# 5 Reverse-Delete Algorithm

## 5.1 Definition

Start with $T = E$, where $E$ is the edge array. Then consider edges in descending order of cost. Delete edge $e$ from $T$ unless doing so would disconnect $T$.

## 5.2 Proof

If removing edge $e$ does not disconnect the graph, then $e$ is part of a cycle. Therefore the edges removed do not belong to any MST. Finally, the output is a spanning tree, connected and no cycle.