Minimum Spanning Trees (MST)

A tree in graph theory is defined as an undirected graph `G = (V, E)` that is both:

- Connected: There is a path between all pairs of nodes.
- Acyclic: There are no cycles of edges.

An undirected, acyclic graph is also known as a forest of trees.

Tree Properties

Theorem (B.2): For an undirected graph `G = (V, E)`, the following statements are equivalent:

1. G is a tree (connected and acyclic).
2. G is connected, but removing any edge disconnects it.
3. G is connected and `m = n - 1` (where `m` is the number of edges and `n` is the number of nodes).
4. G is acyclic, but adding any edge creates a cycle.
5. G is acyclic and `m = n - 1`.
6. There is exactly one path between every pair of nodes.

Minimum Spanning Tree (MST) Definition

> A spanning tree for an undirected, connected graph `G = (V, E)` is a subgraph `T = (V, E')` where `E' ⊆ E`, which forms a tree.

Note that the node set `V` is the same for both the original graph and the spanning tree. Per Theorem B.2, all spanning trees have `n - 1` edges.

> A Minimum Spanning Tree (MST) for a weighted, undirected, connected graph `G` is a spanning tree of `G` with the minimum possible sum of edge weights.

The original motivation for MST algorithms was to find the most cost-effective way to connect points in a supply network (e.g., electricity, oil). The edges in `G` represent possible connections, and the edge weights represent the cost of establishing that connection. The first algorithm for this problem was developed by Borůvka in 1926.

MST Algorithms

The basic idea behind MST algorithms is a greedy approach: build the MST by selecting edges one by one based on a specific rule.

Correctness Invariant

The correctness of these algorithms relies on the invariant:

> What we have built so far is part of an optimal solution.

Specifically, if $A \subseteq E$ represents the edges selected so far, then the invariant is:

> There exists an MST that contains A.

A safe edge for A is an edge that can be added to A without violating the invariant. At least one safe edge must exist as long as the invariant holds and $|A| < n - 1$.

MST Algorithm Invariant

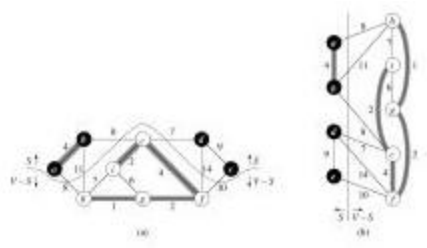
Invariant: There exists an MST that contains A.

- Initialization: Any connected graph has at least one spanning tree and thus an MST. The empty set $\varnothing$ trivially satisfies the invariant.
- Maintenance: Ensuring the selected edge is a safe edge maintains the invariant.
- Termination: Any (M)ST contains exactly $n - 1$ edges. Since A grows by one edge per iteration, the invariant ensures the algorithm terminates with A being an MST.

Finding a Safe Edge

> A cut is a subset $S \subseteq V$ of the nodes, effectively dividing the nodes into two sets: S and $V - S$.

A light edge across a cut is the one with the minimum weight.



In the image, the cut is visualized as dividing the graph into two sets of nodes (black and white). The edges crossing between these sets are candidates for the lightest edge.

Cut Theorem

- If there exists an MST that contains `A`,
- `S` is a cut such that `A` has no edges crossing it, and
- `e` is a lightest edge across the cut,
- then `e` is safe for `A` (i.e., there exists an MST that contains `A u {e}`).

Proof:

1. Let `T` be an MST that contains `A`.
2. We want to create an MST `T'` that contains `A u {e}`.

Let `e = (u, v)` be a lightest edge across the cut `S`.

Since `T` is connected, there must be a path in `T` between `u` and `v`, with at least one edge `(x, y)` crossing the cut `S`.

Let `T'` be `T` with `(x, y)` replaced by `e = (u, v)`. Since `T'` is still connected and has `n` nodes and `n - 1` edges, `T'` is also a tree. Because `e` is a lightest edge across the cut, `T'` can only be lighter than `T`. Therefore, `T'` is also an MST.

`T'` contains `A u {e}` because the removed edge `(x, y)` is not in `A` (since `A` has no edges crossing the cut).

Using the Cut Theorem in MST Algorithms

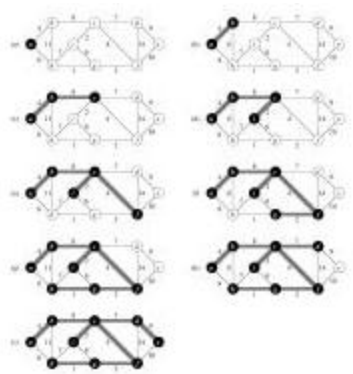
Invariant: There exists an MST that contains the selected edges `A`.

- A new edge `(u, v)` with both endpoints in the same connected component in `G' = (V, A)` will introduce a cycle and thus violate the invariant. Such edges are never safe.
- A new edge `(u, v)` with endpoints in different connected components `C1` and `C2` in `G' = (V, A)` is safe if it is a lightest edge out of `C1`: use the cut theorem on the cut `C1`.

Adding an edge with endpoints in different connected components `C1` and `C2` in `G'` merges `C1` and `C2` into a single connected component.

Prim-Jarnik MST Algorithm

The Prim-Jarnik algorithm (Prim 1957, Jarnik 1930) starts from an arbitrary starting node `r` and continuously expands `r`'s connected component.



Each node `v ∈ V − C` stores information about its shortest edge across the cut `C` in the fields `v.key` and `v.π`. The set `A` is `{(v, v.π) | v ∈ C − {r}}`. The nodes in `V − C` are stored in a min-priority queue `Q`.

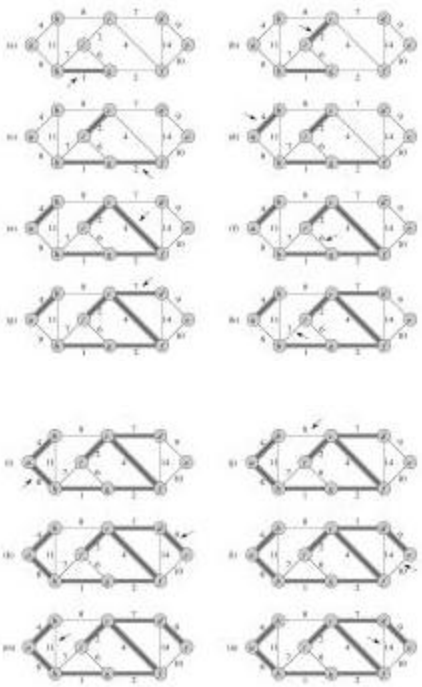
- **Correctness**: via the cut theorem and the invariant.
- **Running Time**: `n` Insert, `n` ExtractMin, `m` DecreaseKey on a priority queue of size `O(n)`, totaling `O(m log n)` since `m ≥ n − 1` (`G` is connected).

Kruskal MST Algorithm

Kruskal's algorithm (1956) attempts to add edges to `A` in globally lightest-first order.

Recap:

1. An edge `(u, v)` can never be added to `A` if `u` and `v` lie in the same connected component in `G' = (V, A)`.
2. If an edge `(u, v)` between two different connected components is added to `A`, these two connected components become one.



The algorithm maintains the connected components in `G' = (V, A)` using a disjoint-set data structure on `V`:

```
Make-Set(x)
Union(x, y)
Find-Set(x)
```

Properties

1. The data structure maintains the connected components in `G' = (V, A)`.
2. An edge examined (in the IF statement) has both endpoints in the same connected component after the examination, regardless of the outcome of the test in the IF statement. Since connected components in `G'` only merge along the way, this also applies to the edge in the rest of the algorithm.

Kruskal Correctness

At the time the algorithm adds an edge `(u, v)` to `A`, `u` and `v` lie in different connected components `C1` and `C2` in `G' = (V, A)`. This follows from property 1 and the test in the IF statement.

Consider the cut given by `u`'s connected component `C1`. All lighter edges have already been examined and therefore have both endpoints in the same connected component in `G'` (property 2). Therefore, `(u, v)` is a lightest edge across this cut, and we can use the cut theorem.

When the algorithm stops, all edges have been examined. Every edge in the input graph `G = (V, E)` therefore has both endpoints in the same connected component in `G' = (V, A)` [property 2]. Such edges cannot be added to `A` without introducing a cycle.

So `A` is itself the MST (from the invariant) that contains `A`, since no edges can be added to `A`. Thus, the algorithm is correct.

Note that there are exactly `n - 1` Union operations along the way, since each adds one edge to `A`, and since an MST has `n - 1` edges.

Kruskal Running Time

- Work: Sort `m` edges, perform `n` Make-Set, `n - 1` Union, `m` Find-Set.
- From previous knowledge, there exists a data structure for disjoint-sets where

```
Make-Set(x)
n - 1 Union(x, y)
m Find-Set(x)
```

takes a total of `O(m + n log n)` time. The overall running time for Kruskal is `O(m log m)` since `m ≥ n - 1`, as the input graph is connected.