Kruskal's Algorithm

Minimum Spanning Trees & Union-Find

Yahya Efe Kuruçay 20220808005

May 17, 2025

Why Minimum Spanning Trees?

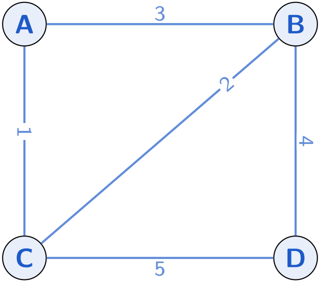
- Connect all nodes with **minimum total edge weight**.
- Applications:
  - Network design (telecom, electrical grids)
  - Cluster analysis
  - Approximation for NP-hard problems (e.g., TSP)

Kruskal's Approach

1. Sort edges by weight in ascending order.
2. Add edges one by one, **skipping cycles**.
3. Use Union-Find data structure for efficient cycle detection.

Time Complexity: $\mathcal{O}(E \log E)$

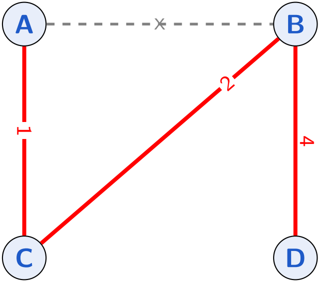
Step-by-Step Example: Initial Graph



Sorted edges by weight:

(A-C: 1), (B-C: 2), (A-B: 3), (B-D: 4), (C-D: 5)

Edge Selection Process



MST Edges: (A-C), (B-C), (B-D)

Total weight: $1 + 2 + 4 = 7$

Python Implementation: Union-Find (Core Logic)

```
class UnionFind:
    def __init__(self, size):
        self.parent = list(range(size))
        self.rank = [0] * size # For union by rank

    def find(self, i): # Path Compression
        if self.parent[i] == i: return i
        self.parent[i] = self.find(self.parent[i]) # Crucial line
        return self.parent[i]

    def union(self, i, j): # Union by Rank
        root_i = self.find(i)
        root_j = self.find(j)
        if root_i != root_j:
            # Attach smaller rank tree under root of higher rank tree
            if self.rank[root_i] < self.rank[root_j]: self.parent[root_i] =
            ↪  root_j
            elif self.rank[root_i] > self.rank[root_j]: self.parent[root_j] =
            ↪  root_i
            else:
                self.parent[root_j] = root_i; self.rank[root_i] += 1
            return True
        return False
```

Python Implementation: Kruskal's Algorithm (Core Logic)

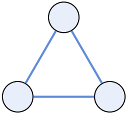
```python
# Assumes UnionFind class is defined (previous slide)
def kruskal_algorithm(num_nodes, edges):
    # edges: list of (weight, u, v)
    edges.sort() # 1. Sort edges by weight

    mst_edges = []
    uf = UnionFind(num_nodes) # Initialize UnionFind

    for weight, u, v in edges: # 2. Iterate through sorted edges
        if uf.union(u, v): # 3. Add edge if it doesn't form a cycle
            mst_edges.append((weight, u, v))
            # Optimization: stop if MST has (num_nodes - 1) edges
            if len(mst_edges) == num_nodes - 1:
                break

    return mst_edges

# Example (conceptual):
# mst = kruskal_algorithm(4, [(1,0,2), (2,1,2), ...])
```

Core steps: Sort edges, iterate, use Union-Find to check for cycles.

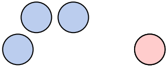
Time Complexity Breakdown

- Sorting edges: $\mathcal{O}(E \log E)$
- Union-Find operations (with path compression union by rank):
  - Initialization of Union-Find: $\mathcal{O}(V)$
  - For E edges, up to $2E$ 'find' and $V - 1$ 'union' operations.
  - Amortized time per operation: Nearly constant, $\mathcal{O}(\alpha(V))$, where α is the inverse Ackermann function.
- **Total Dominant Complexity:** $\mathcal{O}(E \log E)$ (due to edge sorting).

Real-World Applications of MSTs



Network Design



Cluster Analysis

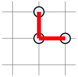


Image Segmentation

Key Takeaways

- Kruskal's is a **greedy algorithm** for finding Minimum Spanning Trees.
- Time complexity is primarily driven by edge sorting: $\mathcal{O}(E \log E)$.
- Efficiently detects cycles using the **Union-Find** data structure.
- Widely applicable in various network optimization and data analysis problems.