



LECTURE 12 – MINIMUM SPANNING TREES

Prepared by: Izbassar Assylzhan

Course: Algorithms & Data Structures – Fall 2025

LECTURE 12



In today's lecture, we'll look at the notion of spanning trees, and implement an algorithm to find Minimum Spanning Tree (MST) using Prime's and Kruskal's algorithms.

TOPICS WE'LL COVER:

Spanning Tree

Kruskal's Algorithm

Prim's Algorithm

Practice Problems

GOALS FOR THIS LECTURE:

- Understand what a Spanning Tree and Minimum Spanning Tree are;
- Differentiate between Kruskal's and Prim's algorithms;
- Implement optimal algorithms of both.

SPANNING TREE

Spanning Tree

Kruskal's Algorithm

Prim's Algorithm

Practice Problems

A Spanning Tree of a graph is as subset of edges that connects all vertices without cycles as shown in Figure 2.

Key properties:

- Unique only when all edges have distinct weights.
- For a graph with N nodes:
→ Every spanning tree has exactly $N - 1$ edges

Minimum Spanning Tree:

A spanning tree where the sum of edge weights is minimized.

We can find the MST with Kruskal's & Prim's algorithms.

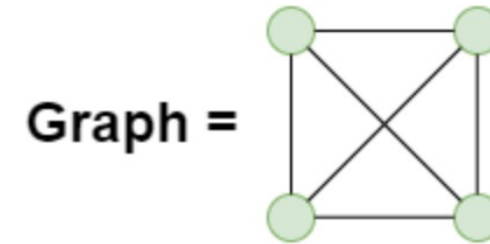


Figure 1 – Example graph [1]

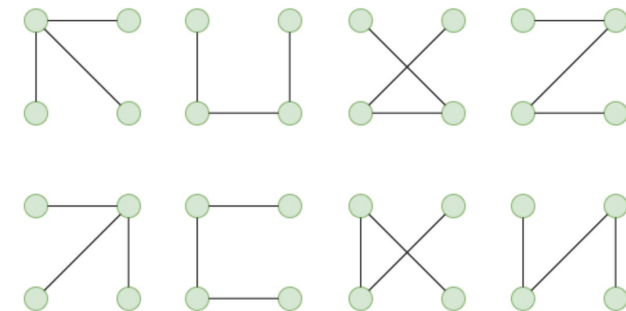


Figure 2 – Some of the spanning graphs of Fig. 1

[1] <https://www.geeksforgeeks.org/dsa/spanning-tree/>

KRUSKAL'S ALGORITHM

Spanning Tree

Kruskal's Algorithm

Prim's Algorithm

Practice Problems

A greedy algorithm that builds the MST by repeatedly taking the smallest weight edge that does not form a cycle.

Steps:

- **Sort** all edges by weight;
- **Iterate** through edges (smallest → largest);
- For each edge (u, v):
 - *If u and v are in different sets* → **include** the edge;
 - *Otherwise skip* (it forms a cycle);
- Use *Disjoint Set Union* (DSU / Union-Find) to detect cycles.

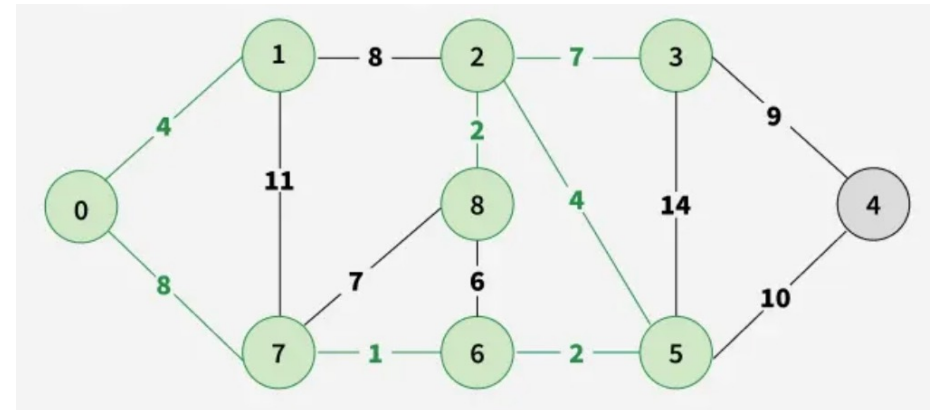


Figure 3 – Example graph for MST [2]

Complexity:

- Sorting edges: $O(E \log E)$. Union-Find operations: almost $O(1)$ → **Total**: $O(E \log E)$.

KRUSKAL'S ALGORITHM (CONT.)

Spanning Tree

Kruskal's Algorithm

Prim's Algorithm

Practice Problems

```
class DisjointSetUnion {
private:
    vector<int> parent, rank;

public:
    DisjointSetUnion(int n) {
        parent.resize(n);
        rank.resize(n, 0);
        for (int i = 0; i < n; i++)
            parent[i] = i;
    }

    int find(int v) {
        if (v == parent[v]) return v;
        return parent[v] = find(parent[v]);
    }

    void unite(int a, int b) {
        a = find(a);
        b = find(b);
        if (a != b) {
            if (rank[a] < rank[b]) swap(a, b);
            parent[b] = a;
            if (rank[a] == rank[b]) rank[a]++;
        }
    }
};
```

Figure 4 – Implementation of DSU

```
struct Edge {
    int u, v, w;
};

bool comparator(const Edge &a, const Edge &b) {
    return a.w < b.w;
}

int kruskal(int n, vector<Edge> &edges) {
    sort(edges.begin(), edges.end(), comparator);
    DisjointSetUnion DisjointSetUnion(n);

    int mst_weight = 0;
    vector<Edge> mst_edges;

    for (Edge &e : edges) {
        if (DisjointSetUnion.find(e.u) != DisjointSetUnion.find(e.v)) {
            DisjointSetUnion.unite(e.u, e.v);
            mst_weight += e.w;
            mst_edges.push_back(e);
        }
    }

    cout << "Edges in MST:\n";
    for (auto &e : mst_edges)
        cout << e.u << " -- " << e.v << " (weight=" << e.w << ")" << endl;

    return mst_weight;
}
```

Figure 5 – Kruskal's Algorithm

PRIM'S ALGORITHM

Spanning Tree

Kruskal's Algorithm

Prim's Algorithm

Practice Problems

A greedy algorithm that grows the MST like BFS, i.e. it starts from any vertex, always add the minimum-weight edge from the tree to a new vertex.

Steps:

1. Start at any vertex;
2. Use a **priority queue** to pick the smallest outgoing edge;
3. Add vertex to MST and push its outgoing edges;
4. Repeat until all vertices are included.

Complexity:

- With priority queue (min-heap): $O(E \log V)$
- Using adjacency list representation

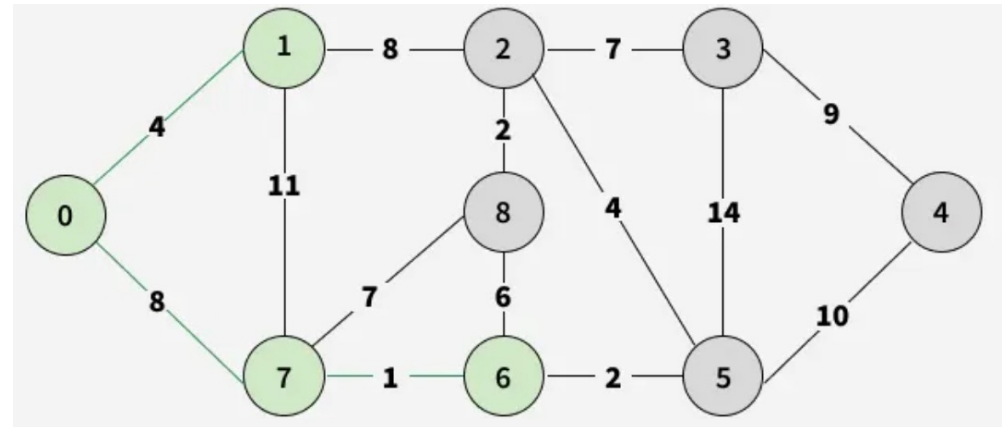


Figure 6 – Representation of expanding the nodes [3]

PRIM'S ALGORITHM (CONT.)

We can see the Prim's algorithm below in Figures 7-9.

Spanning Tree

Kruskal's Algorithm

Prim's Algorithm

Practice Problems

```
typedef pair<int, int> PII;

int prim(int n, vector<vector<PII>>& adj, int start = 0) {

    vector<bool> inMST(n, false);
    priority_queue<PII, vector<PII>, greater<PII>> pq;

    pq.push({0, start});

    int mst_weight = 0;
    vector<pair<int, int>> mst_edges;
```

Figure 7 – Initialization of needed DS

```
    cout << "Edges selected in Prim's MST:\n";
    for (auto &e : mst_edges) {
        cout << "Vertex: " << e.first << " | Edge weight = " << e.second << "\n";
    }

    return mst_weight;
}
```

Figure 9 – Printing the MST

```
while (!pq.empty()) {
    auto [w, v] = pq.top();
    pq.pop();

    if (inMST[v]) continue;

    inMST[v] = true;
    mst_weight += w;

    if (w != 0) mst_edges.push_back({v, w});

    for (auto &edge : adj[v]) {
        int to = edge.first;
        int weight = edge.second;

        if (!inMST[to])
            pq.push({weight, to});
    }
}
```

Figure 8 – The core implementation of Prim's algorithm

Note: The Kruskal's algorithm best for sparse graphs, where E is large but manageable, and the Prim's algorithm for dense graphs, or when using adjacency lists & priority queues.

PRACTICE PROBLEMS

Spanning Tree

Kruskal's Algorithm

Prim's Algorithm

Practice Problems

We have one task to solve:

- [1584. Min Cost to Connect All Points.](#)

Q & A