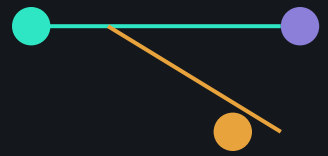


LECTURE 28

Minimum Spanning Trees

CSC211 · Algorithms and Data Structures

COMSATS University Islamabad, Lahore Campus

LECTURE 28

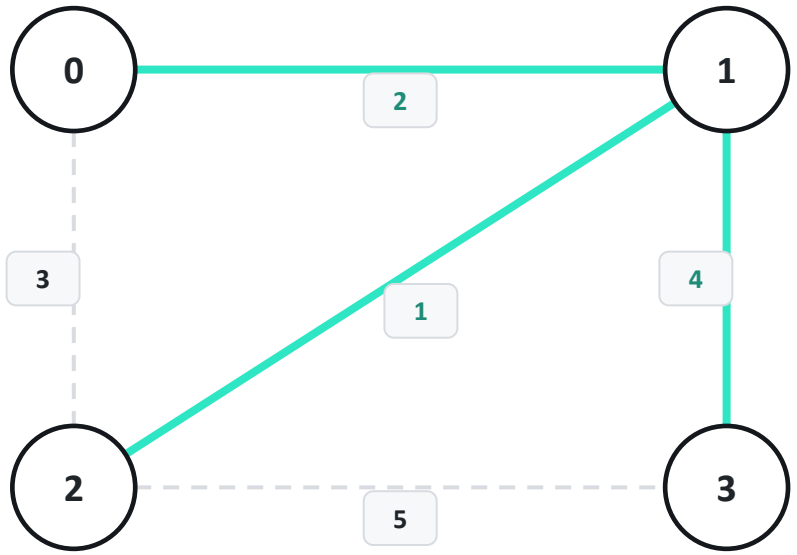
Agenda

1. The spanning tree concept, and what makes one “minimum”
2. Prim's algorithm — grow one connected tree, greedily
3. Kruskal's algorithm — sort all edges, greedily avoid cycles
4. Step-by-step traces of both algorithms, side by side, on the same graph
5. A direct comparison, and when each is the better fit

THE MST CONCEPT

Spanning Tree and Minimum Spanning Tree

A **spanning tree** connects every vertex, no cycles, exactly V-1 edges. The **MST** is the spanning tree with the smallest total weight.



MST: edges 0-1 (2), 1-2 (1), 1-3 (4) — total weight 7. No cheaper combination connects all 4 vertices.

V-1 edges

A 4-vertex spanning tree always has exactly 3 edges — the same edge-count fact from Lecture 16's binary trees.

Two classic algorithms find an MST:

P **Prim's**
Grow one tree outward.

K **Kruskal's**
Sort edges, avoid cycles.

GROW ONE TREE, GREEDILY

Prim's Algorithm

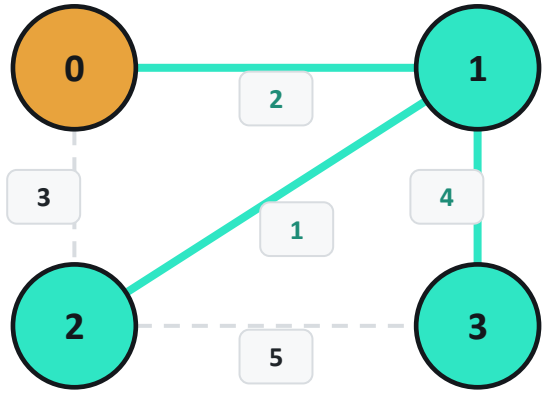
Start from any vertex; repeatedly add the **cheapest edge** that connects the growing tree to a vertex not yet in it — structurally close to Dijkstra's algorithm.

prims_algorithm.cpp — primMST()

```
int primMST() const {
  vector<bool> inMST(numVertices, false);
  priority_queue<...> pq;
  pq.push({0, 0});   // (weight, vertex)
  int totalWeight = 0;

  while (!pq.empty()) {
    auto [weight, current] = pq.top(); pq.pop();
    if (inMST[current]) continue;
  already added cheaper
    inMST[current] = true;
    totalWeight += weight;
    for (auto& [nb, w] : adjList[current])
      if (!inMST[nb]) pq.push({w, nb});
  }
  return totalWeight;
}
```

Output, starting from vertex 0



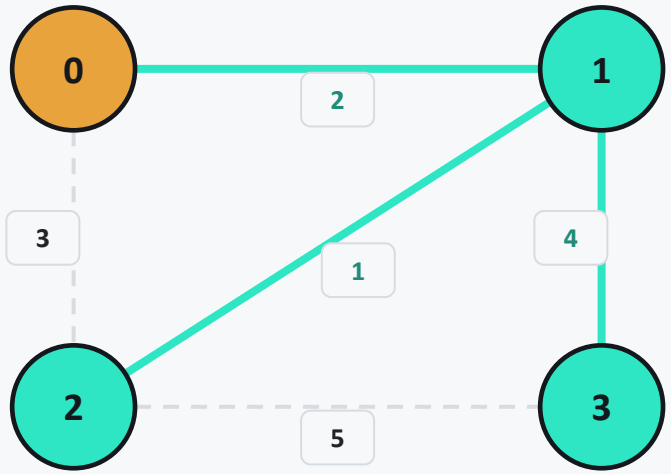
Total MST weight: 7

Dashed gray = considered but never won (a cheaper edge into the same vertex was tried first).

WATCHING THE TOTAL CLIMB

Tracing Prim's Algorithm

Step	Vertex added	Edge weight	Running total
1	0	0	0
2	1	2	2
3	2	1	3
4	3	4	7



Final MST weight: 7

Vertex 0 is the implicit start (step ①). The tree's first edge, 0-1 (weight 2), is step ②.

SORT EDGES, AVOID CYCLES

Kruskal's Algorithm

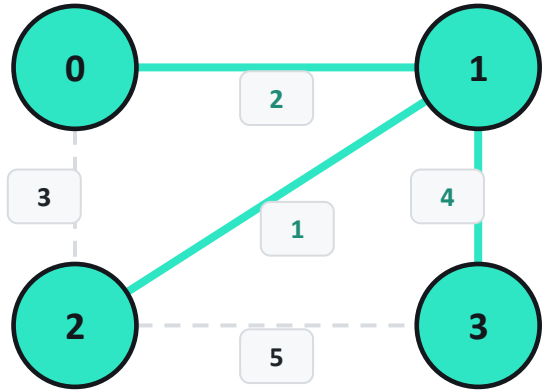
Sort every edge by weight; walk cheapest to most expensive, adding each edge unless it would create a cycle — checked with a **Disjoint Set (Union-Find)**.

kruskals_algorithm.cpp — excerpt

```
bool unite(int u, int v) {
  int rootU = find(u), rootV = find(v);
  if (rootU == rootV) return false;   // same set: cycle
  parent[rootU] = rootV;
  return true;
}

int kruskalMST(int V, vector<Edge> edges) {
  sort(edges.begin(), edges.end(), byWeight);
  DisjointSet ds(V); int total = 0;
  for (auto& e : edges)
    if (ds.unite(e.u, e.v)) total += e.weight;
  add if no cycle
  return total;
}
```

Sorted edges: 1-2(1), 0-1(2), 0-2(3), 1-3(4), 2-3(5)



Total MST weight: 7

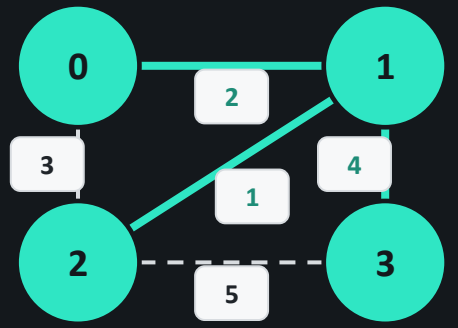
EVERY EDGE, SORTED ORDER

Tracing Kruskal's Algorithm

Edge	Weight	Decision
1–2	1	add — step 1, total 1
0–1	2	add — step 2, total 3
0–2	3	skip — would cycle
1–3	4	add — step 3, total 7
2–3	5	skip — would cycle

Why 0-2 and 2-3 are skipped

By the time 0-2 is considered, find(0) == find(2) already — 0 and 2 are connected indirectly through 0-1-2. The Disjoint Set answers this in near-constant time, no need to “see” the whole tree shape.



DIFFERENT ORDER, SAME TOTAL

Side by Side: Prim's and Kruskal's

Step	Prim's edge	Prim's total	Kruskal's edge	Kruskal's total
1	(start at vertex 0)	0	1-2 (weight 1)	1
2	0-1 (weight 2)	2	0-1 (weight 2)	3
3	1-2 (weight 1)	3	(skip 0-2, would cycle)	3
4	1-3 (weight 4)	7	1-3 (weight 4)	7
5			(skip 2-3, would cycle)	7

Both end at 7, using the exact same three edges (0-1, 1-2, 1-3) — arrived at in a completely different order, for completely different reasons.

COMPARISON AND APPLICATIONS

Prim's vs. Kruskal's, and Where MSTs Are Used

	Prim's	Kruskal's
Approach	Grows one tree outward	Sorts edges, adds cheapest first
Data structure	Min-heap priority queue	Sorted edges + Disjoint Set
Time complexity	O(E log V)	O(E log E), dominated by the sort
Best for	Dense graphs	Sparse graphs

N

Network design

Cheapest cable/pipeline layout, fully connected.

A

Approximation algos

A building block for e.g. traveling salesman.

C

Cluster analysis

Cut the priciest MST edges to form clusters.

KEY TAKEAWAYS

- A spanning tree connects every vertex with exactly V-1 edges and no cycles; a Minimum Spanning Tree is the cheapest one possible.
- Prim's algorithm grows one tree outward using a min-heap — structurally close to Dijkstra's algorithm, just optimizing edge weight instead of cumulative distance.
- Kruskal's algorithm sorts all edges and greedily adds the cheapest one that doesn't create a cycle, using a Disjoint Set (Union-Find) to detect cycles.
- Prim's and Kruskal's pick different edges in a different order but always land on the same total MST weight for a given graph.
- This closes Unit 6. Unit 7 returns to arrays and lists with a new lens: searching and sorting them efficiently.