

Course «Algorithms» · Lecture 10

The Greedy Technique and Prim’s Algorithm

Textbook: A. Levitin, Section 9.1

Today: a new technique

Always take what looks best **right now**, and never go back.

- when greedy works and when it fails
- minimum spanning trees and Prim’s algorithm
- a correctness proof with the cut property
- a fast implementation with a priority queue

Every greedy choice must be

Feasible

It satisfies the constraints, so the solution can still be completed.

Locally optimal

The best of the feasible choices by a simple rule.

Irrevocable

Once made, it is never changed.

Simple and fast. The catch: for many problems it gives the wrong answer.

Sometimes it works, sometimes not

Works

- change with coins 1, 5, 10, 25
- fractional knapsack by v/w

Fails

- coins 1, 3, 4 and amount 6
- 0/1 knapsack: 160 instead of 220

$$6 = 4 + 1 + 1 \quad (3 \text{ coins}), \qquad \text{optimum} \quad 6 = 3 + 3 \quad (2 \text{ coins})$$

The main message

For a greedy algorithm, the proof of correctness is part of the algorithm.

- **Greedy stays ahead:** each partial solution extends to an optimal one
- **Exchange argument:** turn any optimum into the greedy one, never worse

Minimum spanning tree

G = ⟨V, E⟩ connected, undirected, weights w(e). A **spanning tree** is a connected acyclic subgraph containing all vertices.

$$\text{a spanning tree has } \ |V| - 1 \ \text{ edges}$$

$$w(T) = \sum_{e \in T} w(e)$$

An **MST** has the smallest w(T). Uses: cables and pipelines, network backbones, clustering, a bound for TSP.

Why not try all spanning trees?

$$n^{n-2}$$

spanning trees in a complete graph on n vertices (Cayley's formula).

For n = 20 that is $20^{18} \approx 2.6 \cdot 10^{23}$. Hopeless.

Prim’s idea: grow one tree

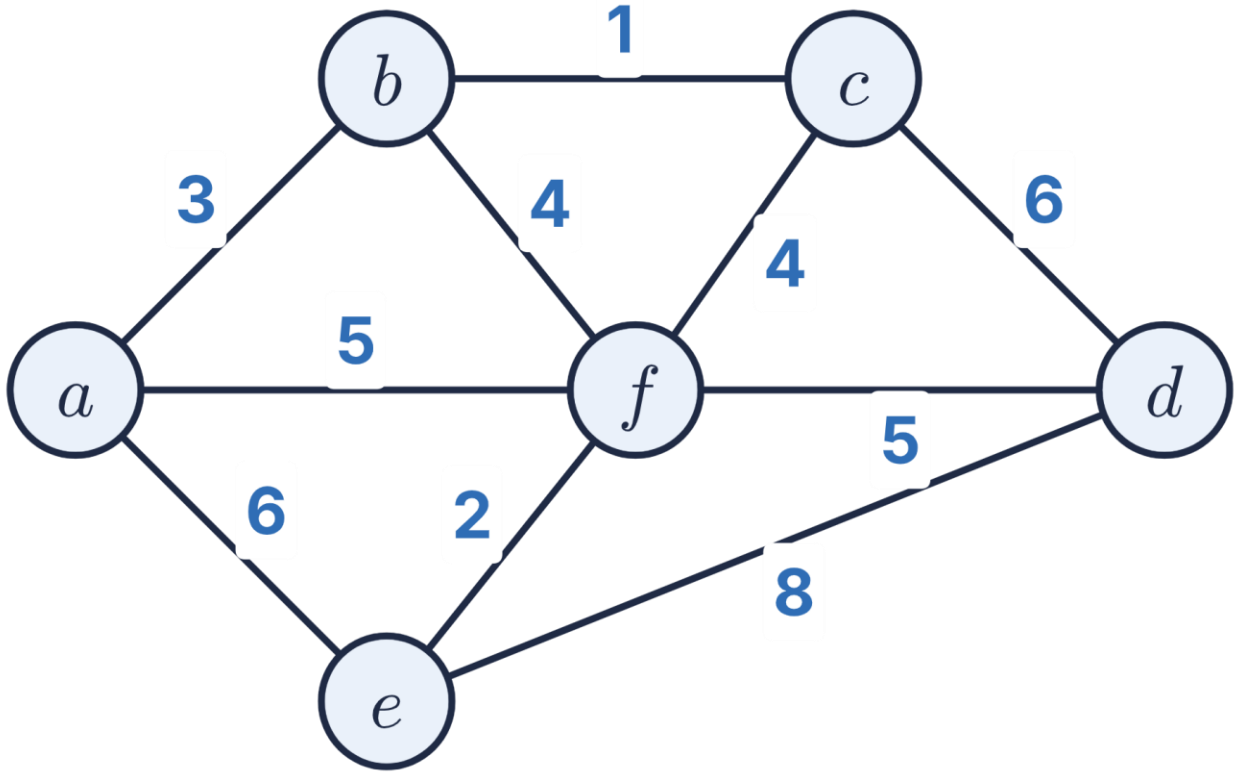
1. Start with any single vertex
2. Attach the **nearest** outside vertex: the one joined by the cheapest edge
3. Repeat until all |V| vertices are in

Tree vertices vs. the rest form a **cut**. Prim always takes the **lightest edge crossing the cut**.

The basic version

```
Prim(G)
  V_T ← {v0}    // any vertex
  E_T ← Ø
  for i ← 1 to |V| − 1 do
    find a lightest edge e* = (v*, u*)
      with v* in V_T and u* in V − V_T
    V_T ← V_T ∪ {u*}
    E_T ← E_T ∪ {e*}
  return E_T
```

Example: six vertices, start at a

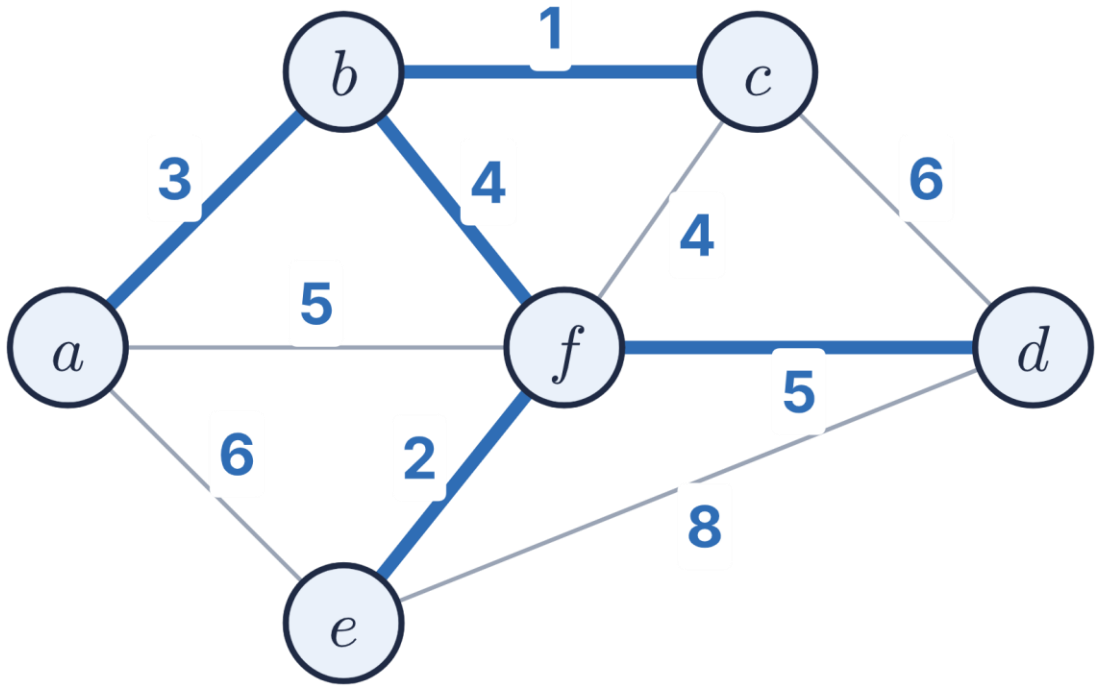


The run, step by step

Step	Tree	Candidates (key, via)	Chosen
0	a	b: 3 (a), e: 6 (a), f: 5 (a)	a–b (3)
1	a, b	c: 1 (b), e: 6 (a), f: 4 (b)	b–c (1)
2	a, b, c	d: 6 (c), e: 6 (a), f: 4 (b)	b–f (4)
3	a, b, c, f	d: 5 (f), e: 2 (f)	e–f (2)
4	a, b, c, f, e	d: 5 (f)	d–f (5)

Bold keys just improved: f from 5 to 4 when b joined; d and e when f joined.

The minimum spanning tree



$$w(T) = 3 + 1 + 4 + 2 + 5 = 15$$

The algorithm never looked back: once an edge was in, it stayed.

Why the minimum? The key tool

Cut property: a lightest edge crossing any cut (S, V – S) belongs to some minimum spanning tree.

S is any nonempty proper subset of V.

Proof: an exchange argument

1. Take any MST T. If e = (u, v) is in T, done.
2. Else T + e has one cycle. It crosses the cut again by some e′ ≠ e, and

$$w(e) \leq w(e')$$

Swap them:

$$T' = (T - \{e'\}) \cup \{e\}$$

$$w(T') = w(T) - w(e') + w(e) \leq w(T)$$

T′ is a spanning tree, no heavier than T, so it is an MST containing e. ■

Prim is correct: induction

Claim: after every step, the chosen edges E_T lie inside some MST.

- **Base:** $E_T = \emptyset$ is inside any MST
- **Step:** e^* is a lightest edge across the cut $(V_T, V - V_T)$; no edge of E_T crosses it, so the exchange keeps E_T
- After $|V| - 1$ steps E_T is a spanning tree inside an MST, so it **is** an MST ■

The same cut property will also prove Kruskal's algorithm.

Fast version: keys and a priority queue

For each outside vertex u: **key[u]** = cheapest edge to the tree, **parent[u]** = its other end.

$$w(u^*, u) < \text{key}[u] \quad \Rightarrow \quad \text{key}[u] \leftarrow w(u^*, u), \quad \text{parent}[u] \leftarrow u^*$$

```
for each u in V: key[u] ← ∞; parent[u] ← nil
key[v0] ← 0;  Q ← all vertices by key
while Q not empty do
  u* ← ExtractMin(Q)
  for each neighbor u of u* in Q with w(u*, u) < key[u] do
    parent[u] ← u*;  DecreaseKey(Q, u, w(u*, u))
```

Reminder: the binary min-heap

A complete binary tree in an array; each key ≤ its children's keys.

$$\text{children}: \; 2i, \; 2i+1, \qquad \text{parent}: \; \lfloor i/2 \rfloor$$

- Insert, ExtractMin, DecreaseKey: **O(log n)** each
- Height $\lfloor \log_2 n \rfloor$; building a heap: O(n)

How fast is it?

Implementation	ExtractMin	DecreaseKey	Total
Array, adjacency matrix	Θ(|V|)	Θ(1)	Θ(|V|²)
Binary heap, lists	Θ(log |V|)	Θ(log |V|)	Θ(|E| log |V|)
Fibonacci heap	Θ(log |V|) am.	Θ(1) am.	Θ(|E| + |V| log |V|)

$$T = O((|V| + |E|)\log|V|) = \Theta(|E|\log|V|)$$

|V| extractions, at most |E| decrease-keys. Dense graphs: array. Sparse graphs: heap.

Special cases

- **Ties:** different MSTs, same weight; distinct weights give a unique MST
- **Negative weights:** no problem, only comparisons matter
- **Disconnected graph:** restart to get a spanning forest
- **Start vertex:** does not change the weight
- **Maximum spanning tree:** use –w
- **Prim vs. Dijkstra:** key = one edge vs. a whole path (next lecture)

Common mistakes

- **“Greedy just works”:** you need a proof
- **Only edges of the last vertex:** the best edge may come from an old one (a–e stayed a candidate)
- **Updating tree vertices:** only vertices still in the queue
- **Wrong bound:** $\Theta(|E| \log |V|)$ needs a heap and lists; a matrix gives $\Theta(|V|^2)$

Summary

- Greedy: feasible, locally optimal, irrevocable; needs a proof
- MST: spanning tree of minimum total weight; brute force hopeless
- Prim grows one tree by the cheapest edge leaving it; example weight 15
- Correct by the cut property and an exchange argument
- $\Theta(|E| \log |V|)$ with a heap, $\Theta(|V|^2)$ with an array

Review questions

1. List the three properties of a greedy choice. Why is irrevocability both the speed and the danger?
2. Run Prim from vertex 1 on edges 1–2 (2), 1–3 (3), 2–3 (1), 2–4 (4), 3–4 (5), 3–5 (6), 4–5 (2). Show each step and the total weight.
3. State the cut property and prove it with the exchange argument.