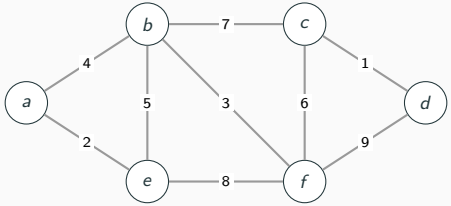
Minimum Spanning Trees

The Cut Property, Prim's Algorithm, Kruskal's Algorithm

The Problem

Let $G = (V, E)$ be a connected, undirected graph with edge weights $w(e)$.

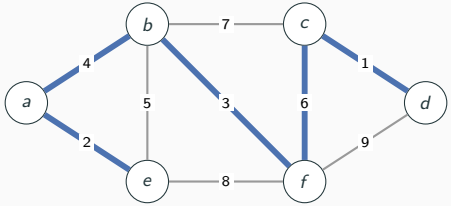
- A spanning tree is a subset $T \subseteq E$ that connects all vertices and has no cycle.
- Its weight is $w(T) = \sum_{e \in T} w(e)$.
- A minimum spanning tree (MST) has the smallest weight among all spanning trees.



The Problem

Let $G = (V, E)$ be a connected, undirected graph with edge weights $w(e)$.

- A spanning tree is a subset $T \subseteq E$ that connects all vertices and has no cycle.
- Its weight is $w(T) = \sum_{e \in T} w(e)$.
- A minimum spanning tree (MST) has the smallest weight among all spanning trees.

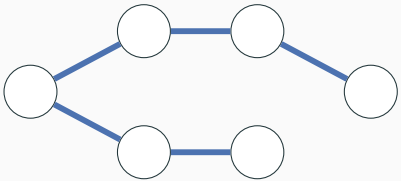


The MST (blue) has weight $2 + 4 + 3 + 6 + 1 = 16$.

Facts About Trees

For a graph on n vertices:

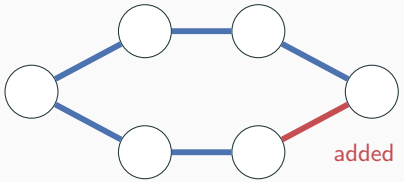
- Every spanning tree has exactly $n - 1$ edges.
- Adding any edge to a spanning tree creates exactly one cycle.
- Removing any edge of that cycle gives a spanning tree again.



Facts About Trees

For a graph on n vertices:

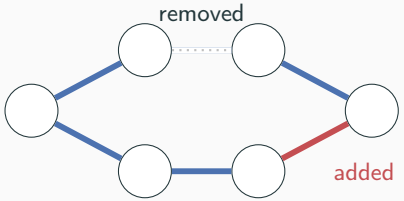
- Every spanning tree has exactly $n - 1$ edges.
- Adding any edge to a spanning tree creates exactly one cycle.
- Removing any edge of that cycle gives a spanning tree again.



Facts About Trees

For a graph on n vertices:

- Every spanning tree has exactly $n - 1$ edges.
- Adding any edge to a spanning tree creates exactly one cycle.
- Removing any edge of that cycle gives a spanning tree again.



This swap is the main tool in every MST proof.

Applications

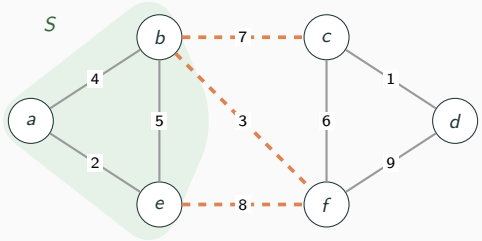
- **Network design.** Connect cities, offices, or circuit pins with cables, pipes, or roads as cheaply as possible.
- **Clustering.** Build an MST of the data points and delete its $k-1$ heaviest edges: the k remaining pieces are the single-linkage clusters.
- **Approximation.** An MST gives a tour at most twice the optimum for the metric travelling salesman problem.
- **Image segmentation.** Pixels are vertices, edge weights are colour differences; MST-based methods split the image into regions.
- **Bottleneck paths.** The path between u and v in an MST minimises the heaviest edge used.

Throughout, we assume all edge weights are distinct. Then the MST is unique, which makes the proofs cleaner. Everything still works with ties, broken consistently.

The Cut Property

Cuts

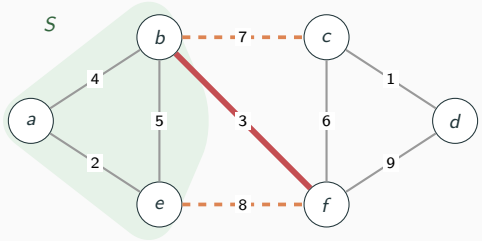
A cut is a partition of the vertices into S and $V \setminus S$, both nonempty. An edge crosses the cut if it has one endpoint on each side.



Here $S = \{a, b, e\}$. The crossing edges (orange) have weights $7, 3, 8$.

Cuts

A cut is a partition of the vertices into S and $V \setminus S$, both nonempty. An edge crosses the cut if it has one endpoint on each side.



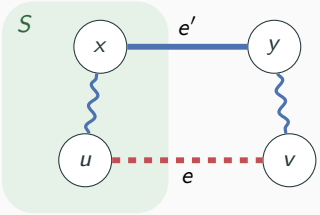
Here $S = \{a, b, e\}$. The crossing edges (orange) have weights $7, 3, 8$. The lightest crossing edge is b–f.

The Cut Property

Cut property

For any cut $(S, V \setminus S)$, the lightest edge crossing the cut belongs to the MST.

Proof (exchange). Let $e = \{u, v\}$ be the lightest crossing edge, $u \in S$, and suppose the MST T does not contain e.

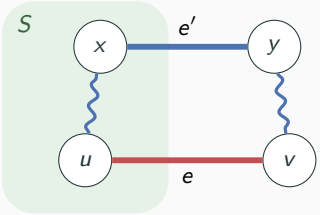


The Cut Property

Cut property

For any cut $(S, V \setminus S)$, the lightest edge crossing the cut belongs to the MST.

Proof (exchange). Let $e = \{u, v\}$ be the lightest crossing edge, $u \in S$, and suppose the MST T does not contain e.



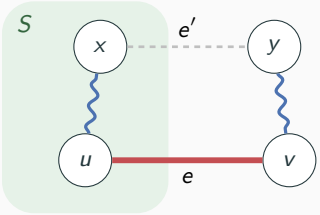
- The path from u to v in T starts in S and ends outside, so it uses some crossing edge $e' = \{x, y\}$.

The Cut Property

Cut property

For any cut $(S, V \setminus S)$, the lightest edge crossing the cut belongs to the MST.

Proof (exchange). Let $e = \{u, v\}$ be the lightest crossing edge, $u \in S$, and suppose the MST T does not contain e.



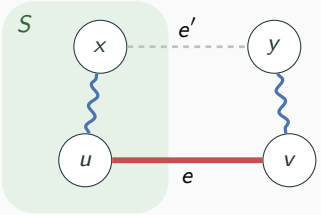
- The path from u to v in T starts in S and ends outside, so it uses some crossing edge $e' = \{x, y\}$.
- $T' = T - e' + e$ is a spanning tree (swap on the cycle formed by e).

The Cut Property

Cut property

For any cut $(S, V \setminus S)$, the lightest edge crossing the cut belongs to the MST.

Proof (exchange). Let $e = \{u, v\}$ be the lightest crossing edge, $u \in S$, and suppose the MST T does not contain e.



- The path from u to v in T starts in S and ends outside, so it uses some crossing edge $e' = \{x, y\}$.
- $T' = T - e' + e$ is a spanning tree (swap on the cycle formed by e).
- $w(e) < w(e')$, so $w(T') < w(T)$. Contradiction. □

A Generic Greedy Algorithm

The cut property gives a recipe for building the MST one edge at a time.

Generic MST

Start with $F = \emptyset$. Repeat $n - 1$ times:

1. choose a cut that no edge of F crosses,
2. add the lightest edge crossing it to F.

- Every added edge is in the MST, by the cut property.
- The chosen cut separates two different pieces of F, so no cycle is ever created.

A Generic Greedy Algorithm

The cut property gives a recipe for building the MST one edge at a time.

Generic MST

Start with $F = \emptyset$. Repeat $n - 1$ times:

1. choose a cut that no edge of F crosses,
2. add the lightest edge crossing it to F.

- Every added edge is in the MST, by the cut property.
- The chosen cut separates two different pieces of F, so no cycle is ever created.

Prim and Kruskal are two ways of choosing the cut:

- **Prim**: S = the vertices of the single tree grown so far.
- **Kruskal**: S = the component of one endpoint of the next lightest useful edge.

Prim's Algorithm

Prim's Algorithm

Grow a single tree from a start vertex s.

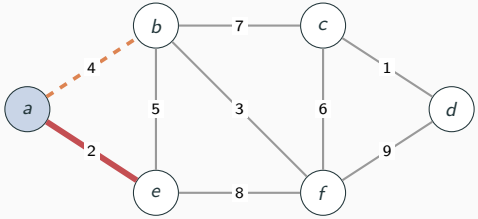
Prim

$S = \{s\}$, $T = \emptyset$. While $S \neq V$: add the lightest edge $\{u, v\}$ with $u \in S$, $v \notin S$ to T, and add v to S.

- Each step uses the cut $(S, V \setminus S)$. No edge of T crosses it, so the cut property says the new edge is in the MST.
- After $n - 1$ steps, T is a spanning tree made of MST edges: it is the MST.

It looks just like Dijkstra, except the priority of v is the weight of the single edge into S, not the distance from s.

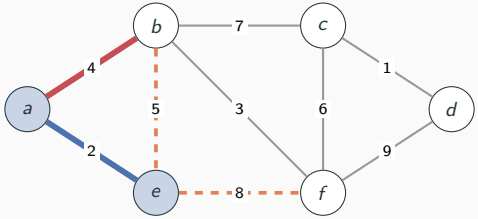
Prim in Action (start at a)



$S = \{a\}$. Crossing: $4, 2$.
Add a–e (2).

dashed: crossing the cut
red: lightest crossing edge
blue: tree so far

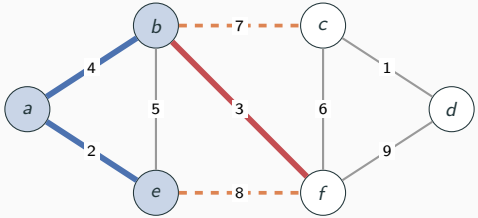
Prim in Action (start at a)



$S = \{a, e\}$. Crossing: $4, 5, 8$.
Add a–b (4).

dashed: crossing the cut
red: lightest crossing edge
blue: tree so far

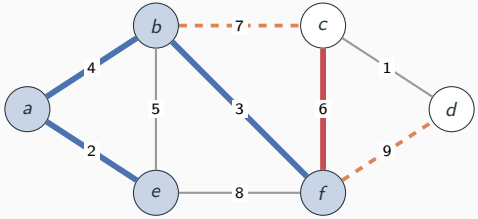
Prim in Action (start at a)



$S = \{a, b, e\}$. Crossing: $3, 7, 8$.
Add b–f (3).

dashed: crossing the cut
red: lightest crossing edge
blue: tree so far

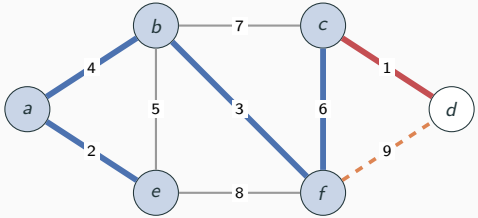
Prim in Action (start at a)



$S = \{a, b, e, f\}$. Crossing: $7, 6, 9$.
Add c–f (6).

dashed: crossing the cut
red: lightest crossing edge
blue: tree so far

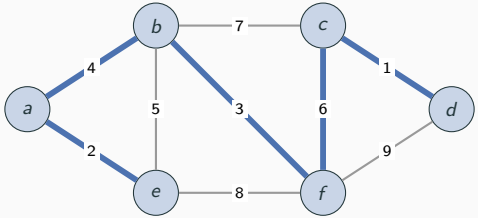
Prim in Action (start at a)



$S = \{a, b, c, e, f\}$. Crossing: $1, 9$.
Add c–d (1).

dashed: crossing the cut
red: lightest crossing edge
blue: tree so far

Prim in Action (start at a)



$S = V$. Done.
Weight $2 + 4 + 3 + 6 + 1 = 16$.

dashed: crossing the cut
red: lightest crossing edge
blue: tree so far

Implementing Prim

For each $v \notin S$ keep

$$\text{key}[v] = \text{weight of the lightest edge from } S \text{ to } v \quad (\infty \text{ if none}),$$

in a min-priority queue, together with parent$[v]$, the other end of that edge.

1. Extract the v with smallest key; add v and $\{\text{parent}[v], v\}$ to the tree.
2. For each neighbour $x \notin S$ of v: if $w(v, x) < \text{key}[x]$, set $\text{key}[x] = w(v, x)$, $\text{parent}[x] = v$.

priority queue	time
array, linear scan	$O(n^2)$ dense graphs
binary heap	$O((n + m) \log n)$

Prim in Julia

```
using DataStructures: PriorityQueue, dequeue_pair!

function prim(n, adj, s = 1) # adj[u]: list of (v, weight) pairs
    key, parent, intree = fill(Inf, n), zeros(Int, n), falses(n)
    key[s] = 0.0
    pq = PriorityQueue{Int, Float64}(s => 0.0)
    while !isempty(pq)
        u, _ = dequeue_pair!(pq)
        intree[u] = true
        for (v, w) in adj[u]
            if !intree[v] && w < key[v] # lighter edge from the tree to v
                key[v] = w
                parent[v] = u
                pq[v] = w # insert, or lower its key
            end
        end
    end
    return parent # MST edges: (parent[v], v) for every v != s
end
```

Kruskal's Algorithm

Kruskal's Algorithm

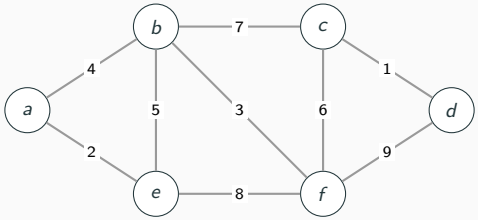
Consider the edges from lightest to heaviest.

Kruskal

Sort the edges by weight. For each edge $\{u, v\}$ in order: if u and v are in different components of T, add the edge to T; otherwise skip it (it would close a cycle).

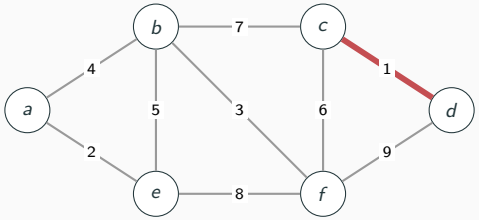
- T is a forest that slowly merges into one tree.
- "Are u and v in the same component?" and "merge the two components" are exactly `find` and `union`: use union-find.

Kruskal in Action



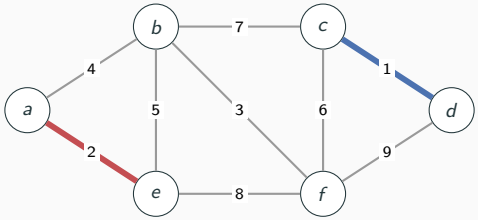
edge	w
c–d	1
a–e	2
b–f	3
a–b	4
b–e	5
c–f	6
b–c	7
e–f	8
f–d	9

Kruskal in Action



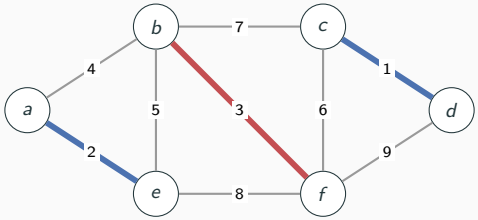
edge	w	
c–d	1	add
a–e	2	
b–f	3	
a–b	4	
b–e	5	
c–f	6	
b–c	7	
e–f	8	
f–d	9	

Kruskal in Action



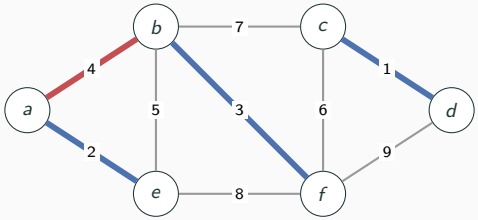
edge	w	
c–d	1	add
a–e	2	add
b–f	3	
a–b	4	
b–e	5	
c–f	6	
b–c	7	
e–f	8	
f–d	9	

Kruskal in Action



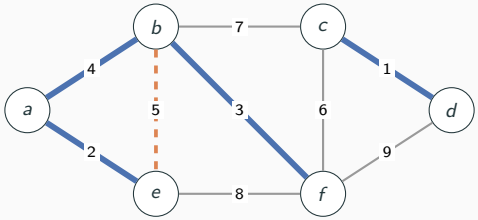
edge	w	
c–d	1	add
a–e	2	add
b–f	3	add
a–b	4	
b–e	5	
c–f	6	
b–c	7	
e–f	8	
f–d	9	

Kruskal in Action



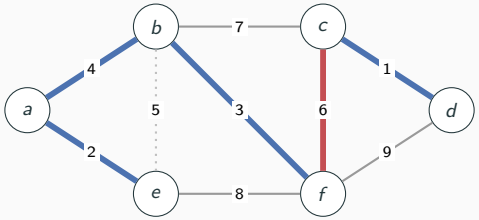
edge	w	
c–d	1	add
a–e	2	add
b–f	3	add
a–b	4	add
b–e	5	
c–f	6	
b–c	7	
e–f	8	
f–d	9	

Kruskal in Action



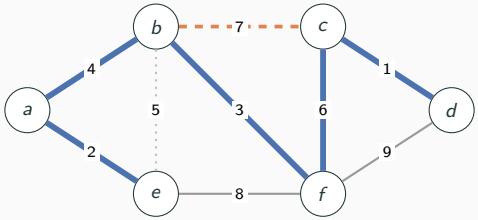
edge	w	
c–d	1	add
a–e	2	add
b–f	3	add
a–b	4	add
b–e	5	skip: b, e joined
c–f	6	
b–c	7	
e–f	8	
f–d	9	

Kruskal in Action



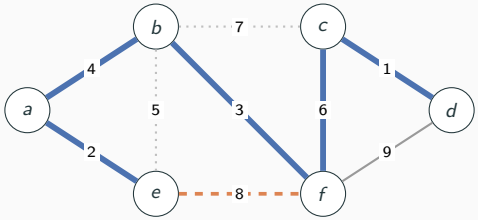
edge	w	
c–d	1	add
a–e	2	add
b–f	3	add
a–b	4	add
b–e	5	skip: b, e joined
c–f	6	add
b–c	7	
e–f	8	
f–d	9	

Kruskal in Action



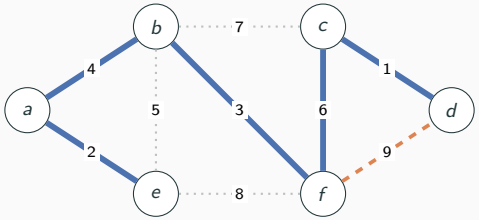
edge	w	
c–d	1	add
a–e	2	add
b–f	3	add
a–b	4	add
b–e	5	skip: b, e joined
c–f	6	add
b–c	7	skip: b, c joined
e–f	8	
f–d	9	

Kruskal in Action



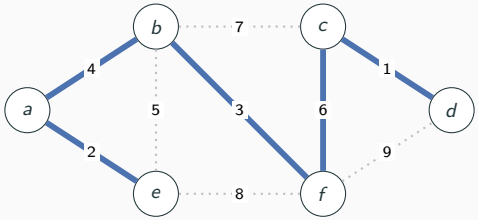
edge	w	
c–d	1	add
a–e	2	add
b–f	3	add
a–b	4	add
b–e	5	skip: b, e joined
c–f	6	add
b–c	7	skip: b, c joined
e–f	8	skip
f–d	9	

Kruskal in Action



edge	w	
c–d	1	add
a–e	2	add
b–f	3	add
a–b	4	add
b–e	5	skip: b, e joined
c–f	6	add
b–c	7	skip: b, c joined
e–f	8	skip
f–d	9	skip

Kruskal in Action

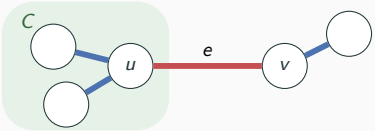


edge	w	
c–d	1	add
a–e	2	add
b–f	3	add
a–b	4	add
b–e	5	skip: b, e joined
c–f	6	add
b–c	7	skip: b, c joined
e–f	8	skip
f–d	9	skip

Same tree as Prim, weight 16.

Why Kruskal Is Correct

Suppose Kruskal adds $e = \{u, v\}$. Let C be the component of T containing u at that moment, and use the cut $(C, V \setminus C)$.



- No edge of T crosses the cut: C is a whole component.
- $v \notin C$, so e crosses the cut.
- Every lighter edge was already considered and now lies inside a component (it was added, or skipped because its endpoints were joined). So e is the lightest crossing edge.

By the cut property e is in the MST; since G is connected, T ends as a spanning tree. □

Kruskal in Julia

```
# edges: list of (weight, u, v)
# UnionFind, union! from the union-find lecture
function kruskal(n, edges)
    uf = UnionFind(n)
    mst = Tuple{Int, Int, Float64}[]
    for (w, u, v) in sort(edges)
        if union!(uf, u, v) # different components
            push!(mst, (u, v, w))
        end
    end
    return mst
end
```

Running time

- Sorting: $O(m \log m) = O(m \log n)$.
- m union-find operations: $O(m\,\alpha(n))$.
- Total $O(m \log n)$, dominated by the sort.

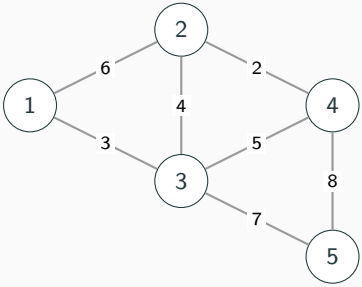
If the edges arrive already sorted, Kruskal runs in nearly linear time.

Prim vs. Kruskal

	Prim	Kruskal
grows	one tree from s	a forest that merges
cut used	tree vs. rest	component vs. rest
data structure	priority queue	sorting + union-find
time	$O((n+m)\log n)$, or $O(n^2)$	$O(m \log n)$
best for	dense graphs (array version)	sparse graphs, sorted edges

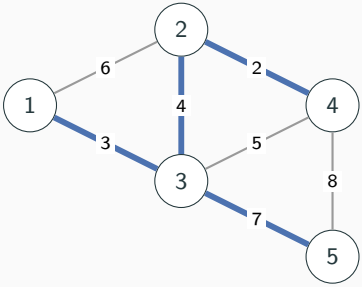
Both are greedy, and both are correct for the same reason: every edge they add is the lightest edge across some cut that the current edges do not cross.

Exercise



1. Run Kruskal. List the edges in the order they are added and the edges that are skipped.
2. Run Prim from vertex 1. List the edges in the order they are added.
3. The weight of edge 3–5 rises from 7 to 10. Which edge replaces it, and what is the new MST weight?

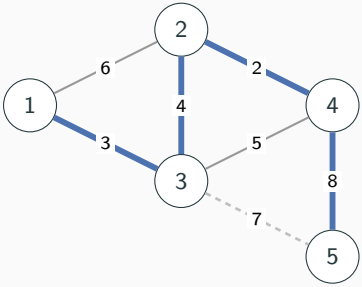
Solution



(a) Add 2–4 (2), 1–3 (3), 2–3 (4). Skip 3–4 (5) and 1–2 (6), whose endpoints are already joined. Add 3–5 (7). Skip 4–5 (8).

(b) 1–3 (3), 2–3 (4), 2–4 (2), 3–5 (7). Same tree, weight 16.

Solution



(a) Add 2–4 (2), 1–3 (3), 2–3 (4). Skip 3–4 (5) and 1–2 (6), whose endpoints are already joined. Add 3–5 (7). Skip 4–5 (8).

(b) 1–3 (3), 2–3 (4), 2–4 (2), 3–5 (7). Same tree, weight 16.

(c) Vertex 5 has only two edges. With 3–5 at 10, the cut $(\{5\}, \text{rest})$ has lightest edge 4–5 (8), so it replaces 3–5. New weight $2 + 3 + 4 + 8 = 17$.

Summary

- An MST connects all vertices with the least total weight, using $n - 1$ edges.
- Cut property: the lightest edge across any cut is in the MST. Proof: swap it with the tree edge that crosses the cut.
- Prim: grow one tree, always adding the lightest edge leaving it. Priority queue, $O((n + m) \log n)$.
- Kruskal: scan edges by weight, add those joining two components. Sorting plus union-find, $O(m \log n)$.