Shortest Paths and Spanning Trees

A network can support different optimization questions. The objective determines which algorithm to use.

Shortest paths: minimize the cost of traveling from a source to a destination.

Minimum spanning tree: connect every node while minimizing the sum of selected edge costs.

- Dijkstra: nonnegative travel costs
- Bellman–Ford: negative travel costs
- Prim and Kruskal: connecting all nodes
- Proofs and further explanation

Recall the network model

Term	Meaning
Node	A location or state
Directed arc (i,j)	Travel is allowed from i to j
Undirected edge {i,j}	Travel is allowed in both directions
Path	A sequence of adjacent nodes with no repeated node
Path cost	The sum of costs of its arcs or edges

For shortest paths we pay for one route. For a spanning tree we pay for every selected edge once.

A shortest path is a one-unit flow problem

Let E be the directed-arc set and $x_{ij} \geq 0$ the flow on arc (i, j), with cost c_{ij}. Send one unit from source s to destination t:

$$\begin{aligned} \min \quad & \sum_{(i,j)\in E} c_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_{j:(i,j)\in E} x_{ij} - \sum_{j:(j,i)\in E} x_{ji} = \begin{cases} 1 & i = s, \\ -1 & i = t, \\ 0 & \text{otherwise}, \end{cases} \\ & x_{ij} \geq 0. \end{aligned}$$

> With a reachable destination and no negative-cost directed cycle, a shortest path gives an optimal flow. Other optimal flows may split across equally short paths.

Represent an undirected edge by two opposite arcs with the same cost.

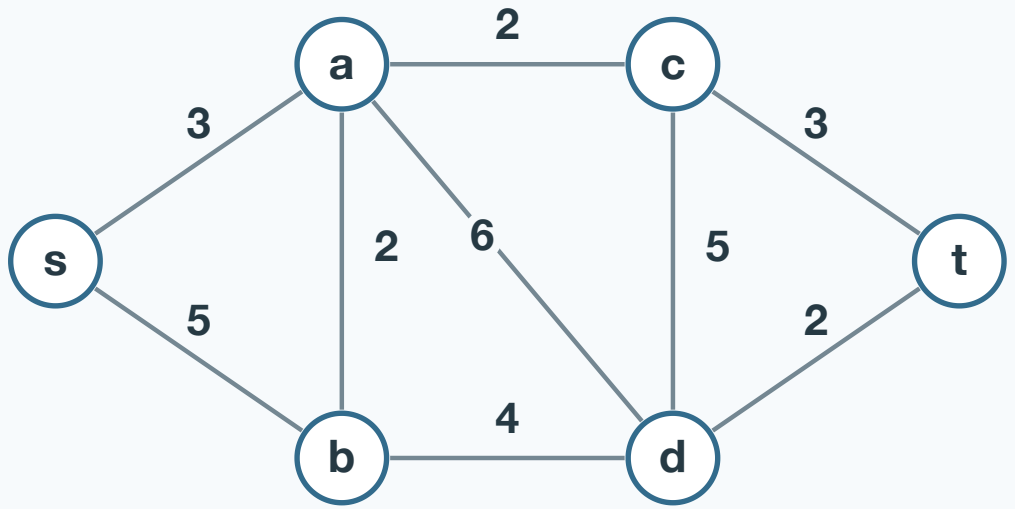
Find the shortest routes starting at s

Every edge can be traversed in either direction. Its number is the travel cost.

> Find the minimum cost from s to every node, especially t.

All nine edge costs are nonnegative. We will use Dijkstra's algorithm.

When distance labels tie, select nodes in this fixed order: s, a, b, c, d, t.



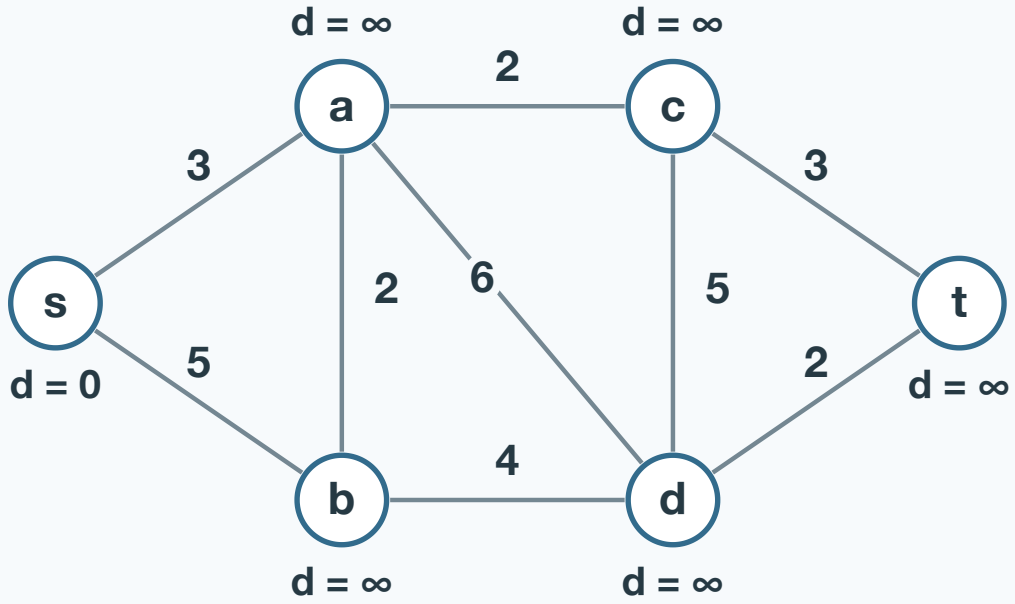
Full undirected network. The source is s; the destination of particular interest is t.

A distance label records the best route found so far

Write $d(v)$ for the cost of the best route found from s to node v. Initially $d(s) = 0$; all other labels are ∞.

> Along an arc or edge from u to v, compare $d(v)$ with $d(u) + c_{uv}$. If the new value is smaller, replace $d(v)$ and record u as the predecessor of v.

This comparison is called **relaxing** the arc or edge. A predecessor records the last step of the improved route.



Undirected edges; labels are costs. Filled nodes have final distances. Bold edges record current predecessors.

Dijkstra: finalize the smallest temporary label

1. Initialize the labels; no node is final yet.
2. Choose an unfinalized node u with the smallest finite label.
3. Mark u final. Relax each edge from u to an unfinalized neighbor.
4. Repeat. Stop when every reachable node is final.

> Keep the existing predecessor when a candidate route has the same cost. This fixes a reproducible choice among equally short routes.

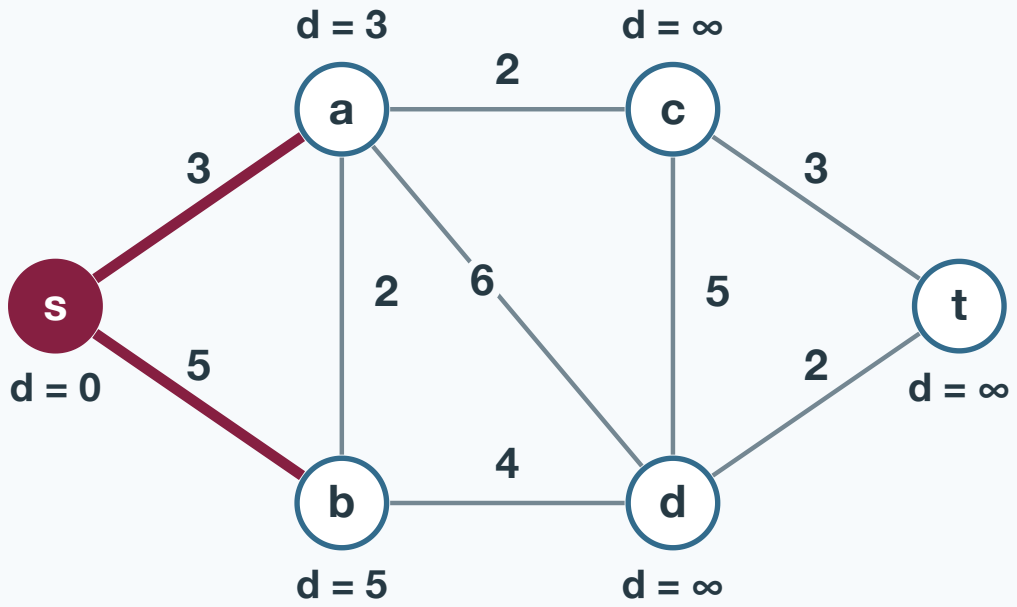
If only one destination matters, stop when that destination becomes final. A label first assigned to a node is not necessarily its final distance.

Select s: discover the first two routes

$$d(a) = 0 + 3 = 3, \qquad d(b) = 0 + 5 = 5.$$

Make s final. Nodes c, d and t have not yet been reached.

Predecessors: a ← s and b ← s. Next choose a because 3 is the smallest temporary label.



Undirected edges; labels are costs. Filled nodes have final distances. Bold edges record current predecessors.

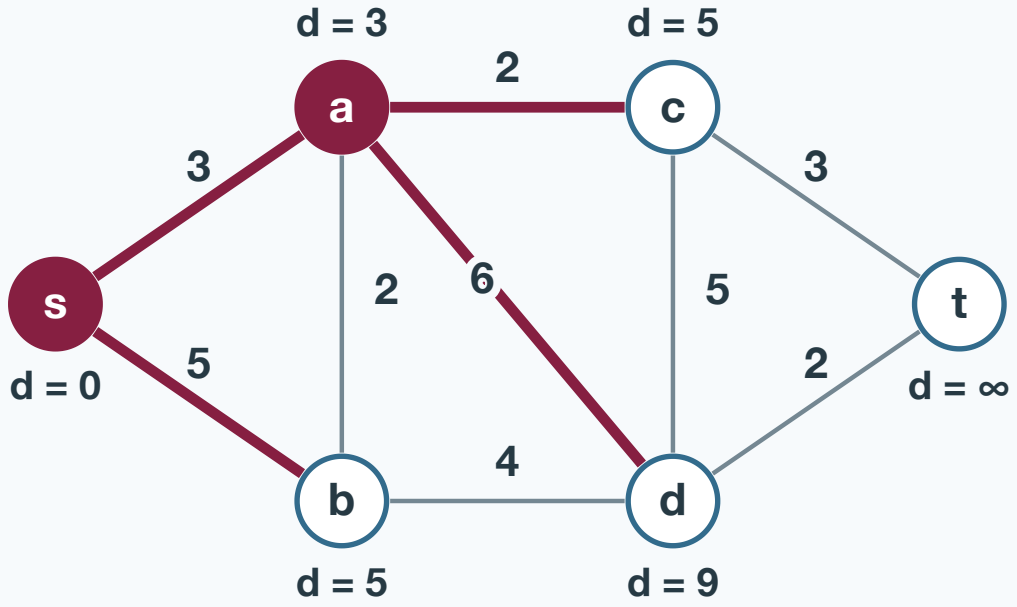
Select a: improve the routes to c and d

$$d(c) = 3 + 2 = 5, \qquad d(d) = 3 + 6 = 9.$$

The route through a to b costs $3 + 2 = 5$, equal to its current label. Keep predecessor s for b.

> Now b and c both have label 5. The stated node order chooses b next.

The label 9 at d is temporary; the algorithm will compare later routes to d.



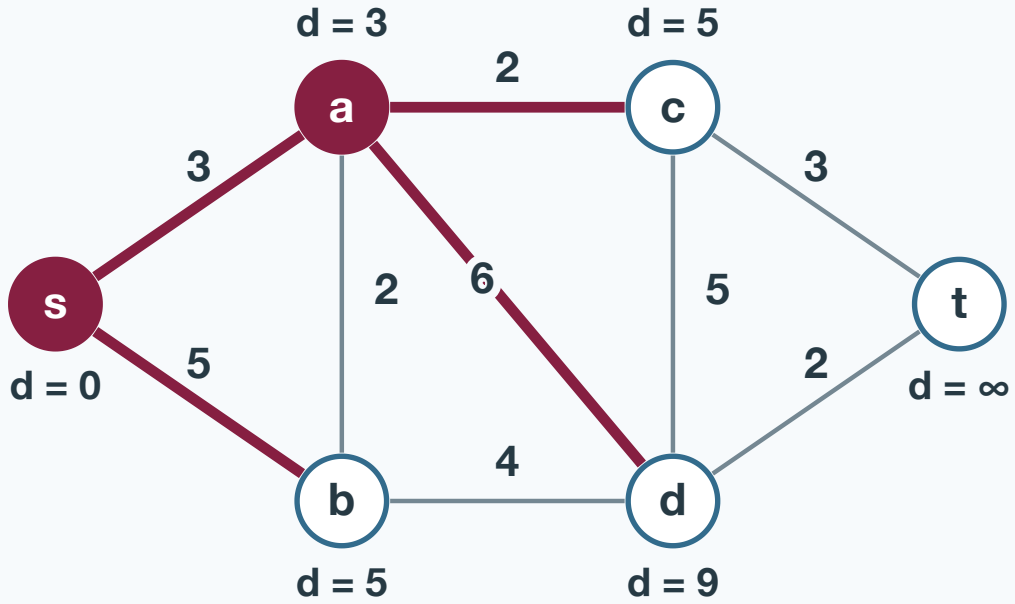
Undirected edges; labels are costs. Filled nodes have final distances. Bold edges record current predecessors.

Your turn: finish the distance table

Work for about 6 minutes. Node a has just become final; the current labels and predecessors are shown.

1. Select the next node and update its neighbors.
2. Continue until every node is final.
3. Follow predecessors backward from t to recover a shortest route.

Use node order s, a, b, c, d, t for ties. Keep the old predecessor when a route only ties the old cost.



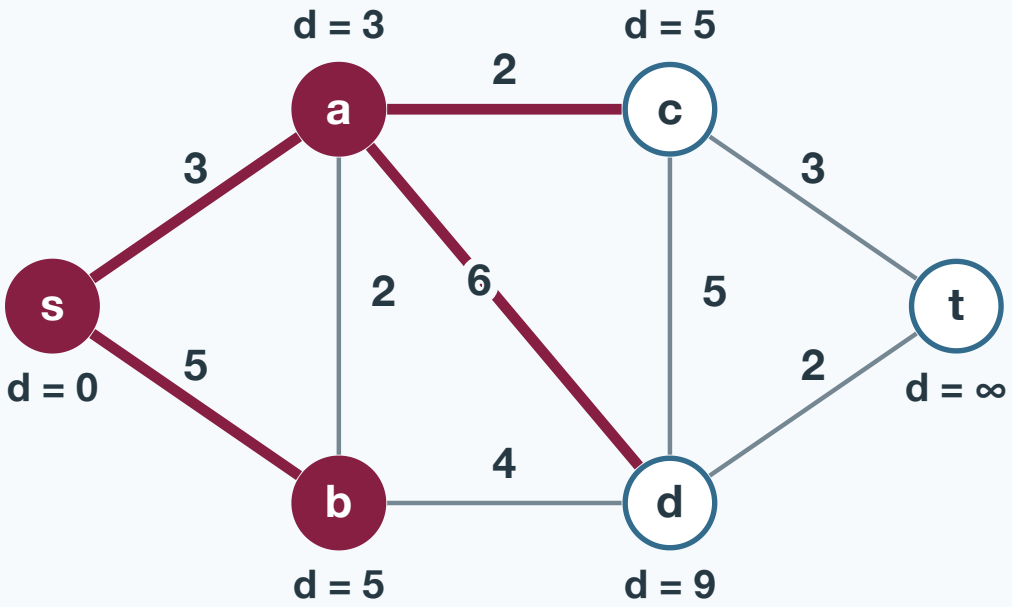
Undirected edges; labels are costs. Filled nodes have final distances. Bold edges record current predecessors.

Select b: no label improves

Node b has label 5. The candidate route to d costs $5 + 4 = 9$, equal to its existing label.

> Keep predecessor a for d. No label changes.

Node a is already final. Among the remaining nodes, c has the smallest label: 5.



Undirected edges; labels are costs. Filled nodes have final distances. Bold edges record current predecessors.

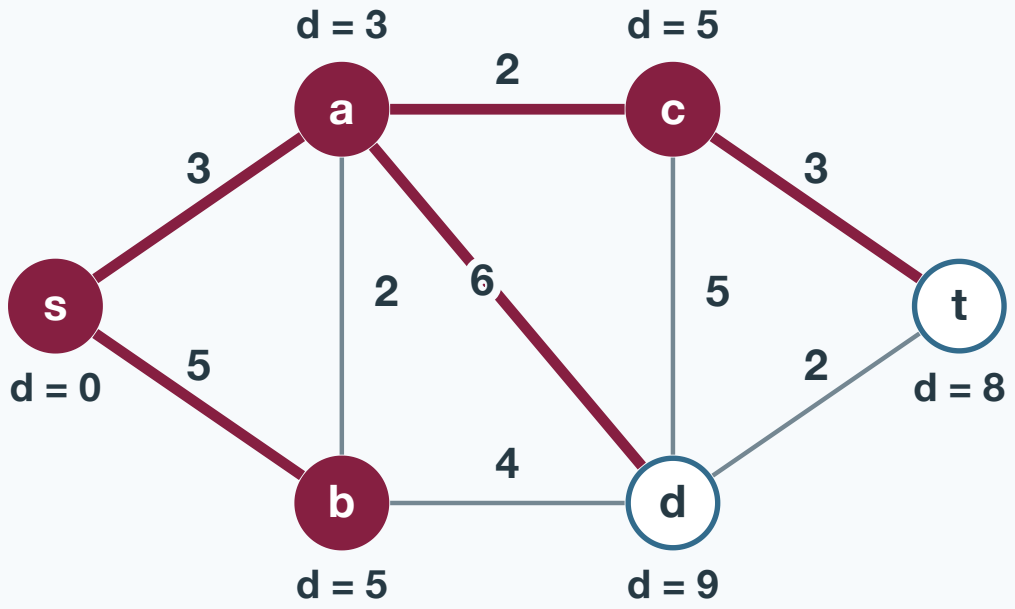
Select c: reach t with cost 8

$$d(t) = 5 + 3 = 8.$$

The candidate route to d costs $5 + 5 = 10 > 9$, so retain $d(d) = 9$.

> Next select t, with label 8, before d, with label 9.

The newly found route to t is s → a → c → t.



Undirected edges; labels are costs. Filled nodes have final distances. Bold edges record current predecessors.

Select t, then d: the full trace is complete

Selected	s	a	b	c	d	t
Initial	0	∞	∞	∞	∞	∞
s	0	3	5	∞	∞	∞
a	0	3	5	5	9	∞
b	0	3	5	5	9	∞
c	0	3	5	5	9	8
t	0	3	5	5	9	8
d	0	3	5	5	9	8

From t, the candidate route to d costs $8 + 2 = 10 > 9$. Selecting d afterward produces no further change.

> Final distances from s: (0, 3, 5, 5, 9, 8), in node order s, a, b, c, d, t.

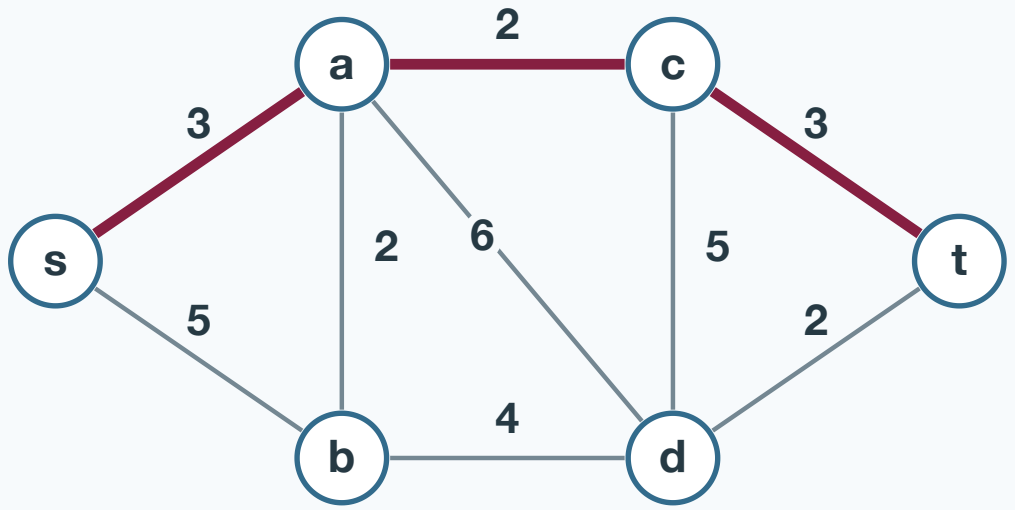
Recover the shortest path using predecessors

Start at t and read its predecessor c, then a, then s. Reverse this sequence.

$$s \to a \to c \to t, \qquad 3 + 2 + 3 = 8.$$

The route s → b → d → t costs 5 + 4 + 2 = 11, so it is longer.

> The highlighted route reaches t. The other predecessor edges still give shortest routes to the remaining nodes.



Bold edges form a shortest s-to-t route, with total cost 8.

Why nonnegative costs justify finalizing a node

> Suppose u has the smallest temporary label. Any undiscovered competing route must first reach some unfinalized node.

That first unfinalized node already has a label no larger than the cost of the competing route’s prefix. Its label cannot be smaller than the label selected at u.

> If every remaining edge cost is nonnegative, continuing along that route cannot lower its cost below the selected label at u.

Appendix: the complete argument. A negative edge can break exactly the “continuing cannot lower cost” step.

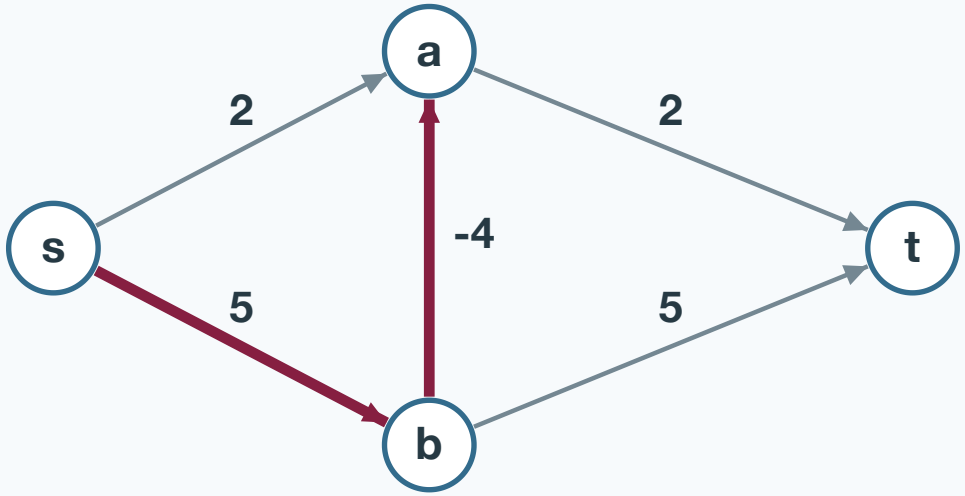
A negative arc can invalidate a final label

Arcs are now directed. From s, Dijkstra would label a as 2 and b as 5, then make a final.

$$s \to b \to a: \quad 5 + (-4) = 1 < 2.$$

> The later route through b improves a. Finalizing a at 2 was incorrect. This graph has no directed cycle.

We need an algorithm that allows later improvements instead of finalizing nodes.



Directed graph. The highlighted route to a costs 1; the direct arc costs 2.

Bellman–Ford: allow successively more arcs

Row k stores $D_k(v)$: the minimum cost from source s to node v using **at most k arcs**. We allow a **walk**, which may repeat nodes.

For an incoming arc $(u, v) \in E$, u is the preceding node and c_{uv} is that arc's cost. E is the set of directed arcs.

$$D_k(v) = \min \left\{ D_{k-1}(v), \quad \min_{(u,v)\in E} \left(D_{k-1}(u) + c_{uv} \right) \right\}.$$

Formula term	What to compare
Keep $D_{k-1}(v)$	Keep the best walk already found with at most k – 1 arcs.
Try $D_{k-1}(u) + c_{uv}$	Reach u using the previous row, then append u → v. Try every incoming arc; take the smallest cost.

One pass = one new row. Copy the previous row, compare candidates for every node, then finish the row before using its new values. Unlike Dijkstra, no node is permanently finalized.

Pass 1: use at most one arc

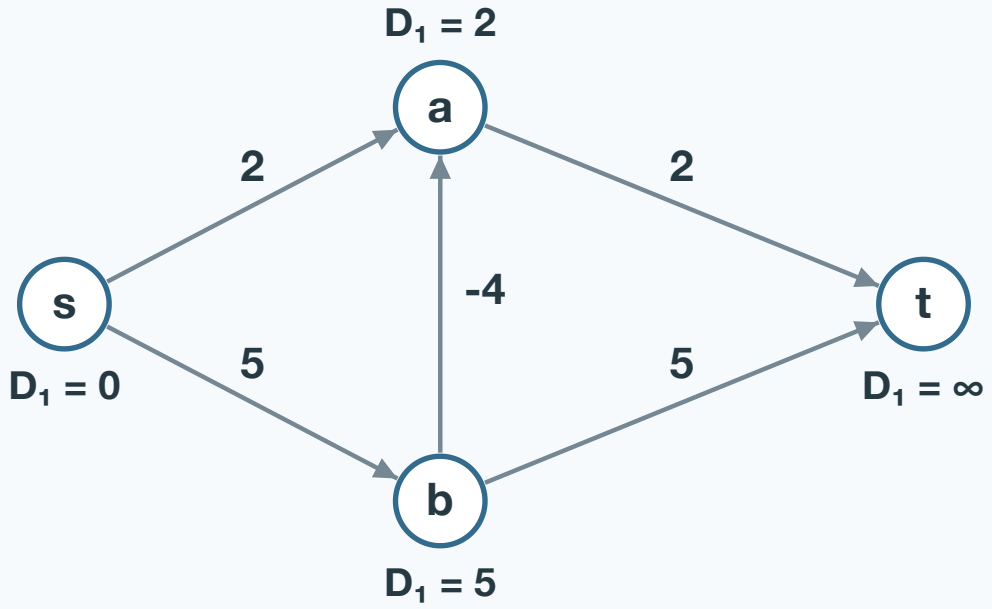
Start with $D_0(s) = 0$; every other entry is ∞, meaning no walk with zero arcs reaches that node.

Pass k	s	a	b	t
0	0	∞	∞	∞
1	0	2	5	∞

For a, compare **keep**, **s → a**, **b → a**:

$$D_1(a) = \min\{\infty, 0 + 2, \infty - 4\} = 2.$$

For b, the only incoming arc is s → b: $D_1(b) = \min\{\infty, 0 + 5\} = 5$.



Node labels: row 1 (≤ 1 arc). Arc labels: costs.

For t, compare **keep**, **a → t**, **b → t**: $D_1(t) = \min\{\infty, \infty + 2, \infty + 5\} = \infty$. Use row 0 at a and b. No walk from s to t uses at most one arc.

Pass 2: the route to a improves

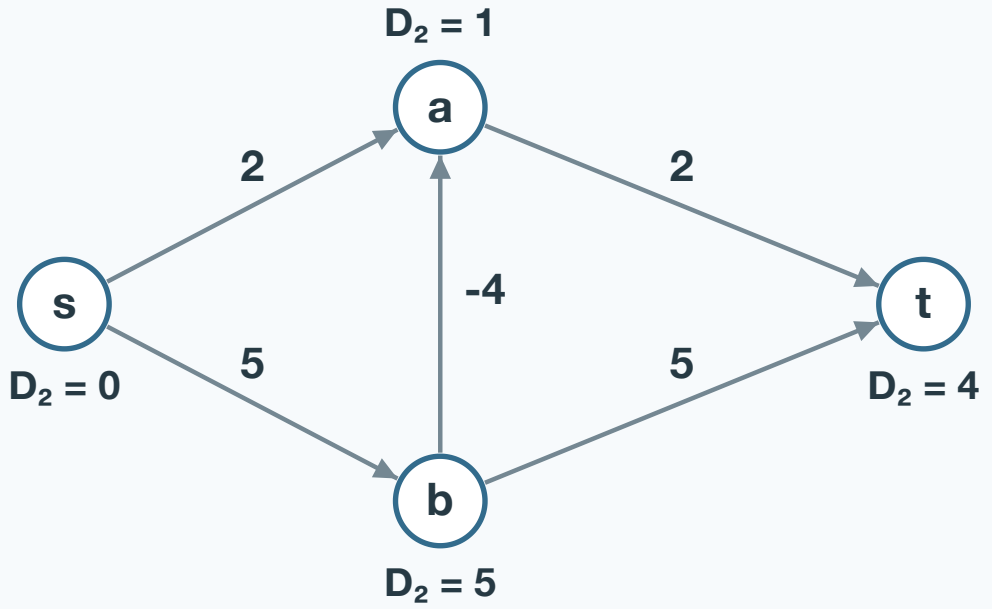
Pass k	s	a	b	t
1	0	2	5	∞
2	0	1	5	4

For a, compare **keep**, **s → a**, **b → a**:

$$D_2(a) = \min\{2, 0+2, 5-4\} = 1.$$

For t, compare **keep**, **a → t**, **b → t**:

$$D_2(t) = \min\{\infty, 2+2, 5+5\} = 4.$$



Node labels: row 2 (≤ 2 arcs). Arc labels: costs.

Why not $1 + 2 = 3$ at t yet? The new $D_2(a) = 1$ uses s → b → a, already two arcs. Appending a → t uses three arcs, so it belongs in row 3.

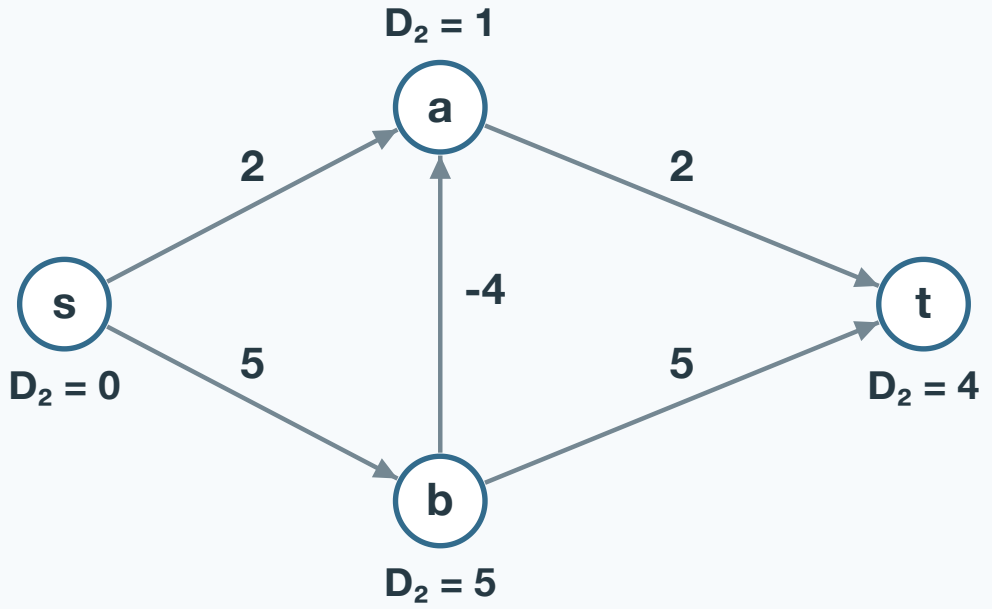
Your turn: complete pass 3

Work for about 4 minutes. Use only row 2 to compute every entry of row 3.

Pass k	s	a	b	t
2	0	1	5	4

At a, compare $D_2(a), D_2(s)+2, D_2(b)-4$.
At t, compare $D_2(t), D_2(a)+2, D_2(b)+5$.

1. Write row 3 for s, a, b and t.
2. Which final arc gives the smallest cost at t? Write the complete route from s.
3. Using row 3, would any entry improve in row 4?



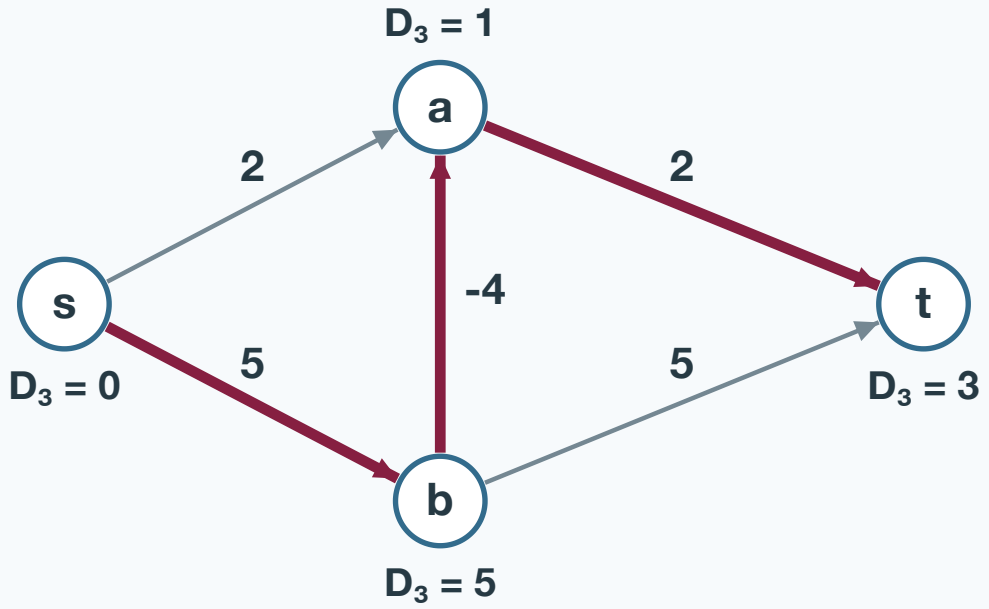
Node labels: row 2 (≤ 2 arcs). Arc labels: costs.

Pass 3 finds the shortest route; pass 4 confirms it

Pass k	s	a	b	t
0	0	∞	∞	∞
1	0	2	5	∞
2	0	1	5	4
3	0	1	5	3
4	0	1	5	3

At t: **keep 4**, **via a: 1 + 2**, **via b: 5 + 5**.

$$D_3(t) = \min\{4, 1+2, 5+5\} = 3.$$



Node labels: row 3 (≤ 3 arcs). Arc labels: costs.

Shortest route: s → b → a → t, with cost 5 – 4 + 2 = 3.

Stop: row 4 = row 3. All comparisons are unchanged, so no later row can improve.

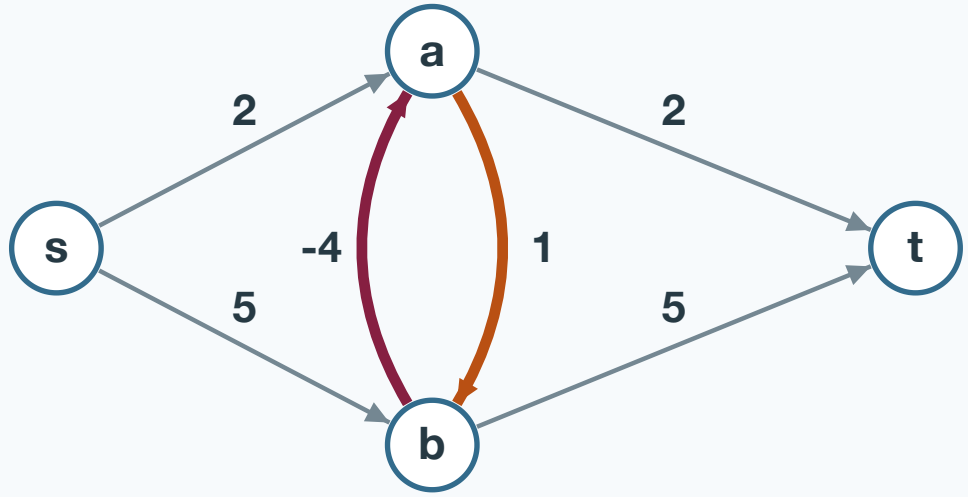
A negative cycle makes the cost unbounded below

Add a → b with cost 1 to the previous graph; all other arcs stay unchanged.

Negative-cost directed cycle: a → b → a has total cost 1 – 4 = –3.

Start: s → b → a → t, cost 3.
One repetition: s → b → a → b → a → t, cost 0.

Cycle repetitions r	Walk cost
0	3
1	3 – 3 = 0
2	3 – 6 = –3
$r = 0, 1, 2, \ldots$	3 – 3r



Orange: new a → b arc. Arc labels are costs.

Repeat the cycle r times before going to t: $3 - 3r \to -\infty$. There is **no finite minimum walk cost**; every finite walk still has a finite cost.

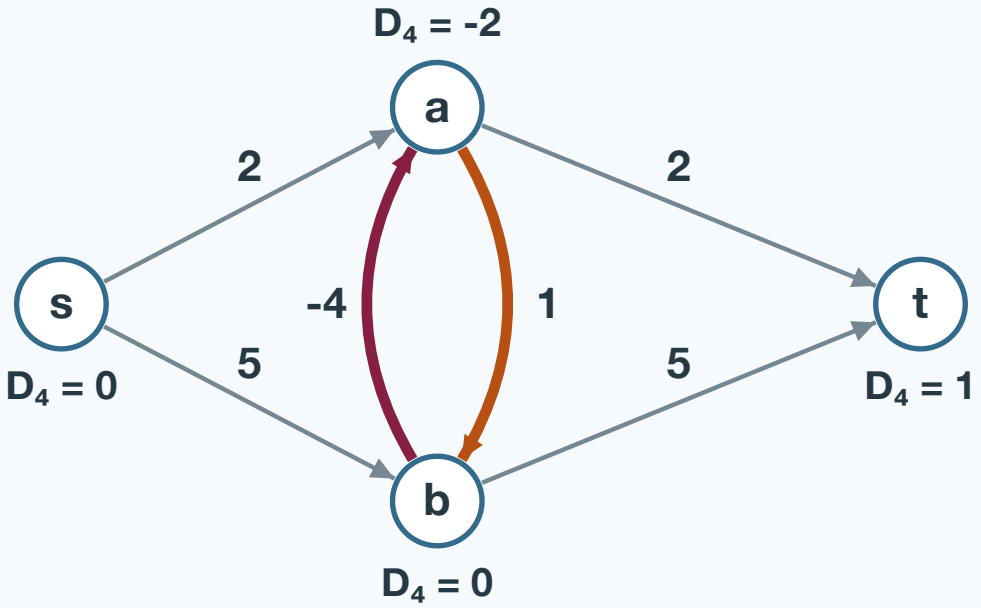
How Bellman–Ford detects the negative cycle

Four nodes, $n = 4$. Compare rows 3 and 4:

Pass k	s	a	b	t
3	0	-1	2	3
4	0	-2	0	1

$$D_4(t) = \min\{3, -1 + 2, 2 + 5\} = 1 < 3.$$

> If **any entry decreases on pass** n, a negative-cost cycle is reachable from s.



Node labels: row 4. Orange: new a → b arc.

> For the value at t to be $-\infty$, **both connections** must exist: s → cycle and cycle → t. Here s → b reaches the cycle, and a → t reaches the destination.

Appendix: why the pass-n test works.

Choose the shortest-path algorithm by its assumptions

	Dijkstra	Bellman–Ford
Arc costs	All nonnegative	May be negative
Progress	Finalize a node	Allow one more arc per pass
Can a label improve later?	Not after it is final	Yes, on later passes
Typical work	$O(n^2 + m)$ by simple scans	$O(nm)$ for n passes
Negative-cycle information	Not applicable	Detects cycles reachable from s

Here n is the number of nodes and m the number of arcs; represent undirected edges by opposite arcs. A binary-heap Dijkstra implementation takes $O((n + m) \log n)$.

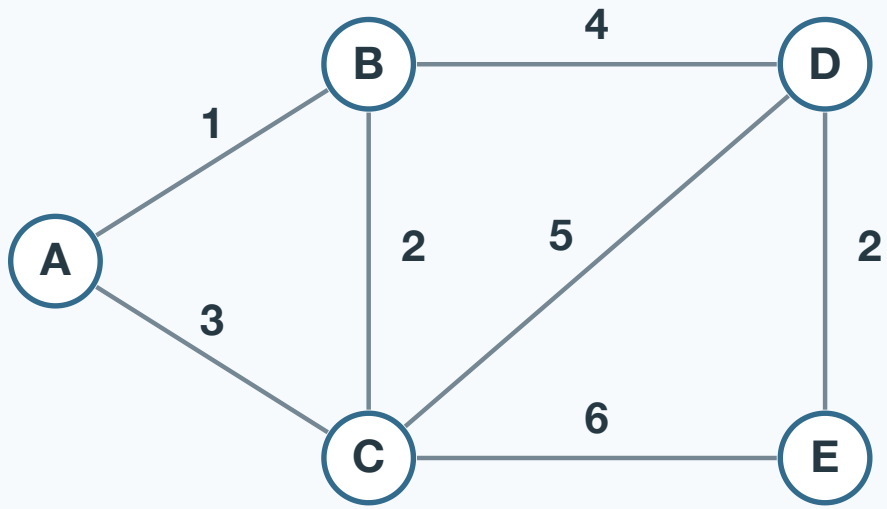
Which cables should connect five buildings?

A university wants to connect buildings A–E with fiber-optic cables.

Each undirected edge is a possible cable connection. Its label is the one-time installation cost, in **thousands of dollars**.

> Every building must be able to communicate with every other building, possibly through intermediate buildings.

Assume positive installation costs. Backup connections after a cable failure are not required.



Possible connections; none selected yet. A–B costs $1,000.

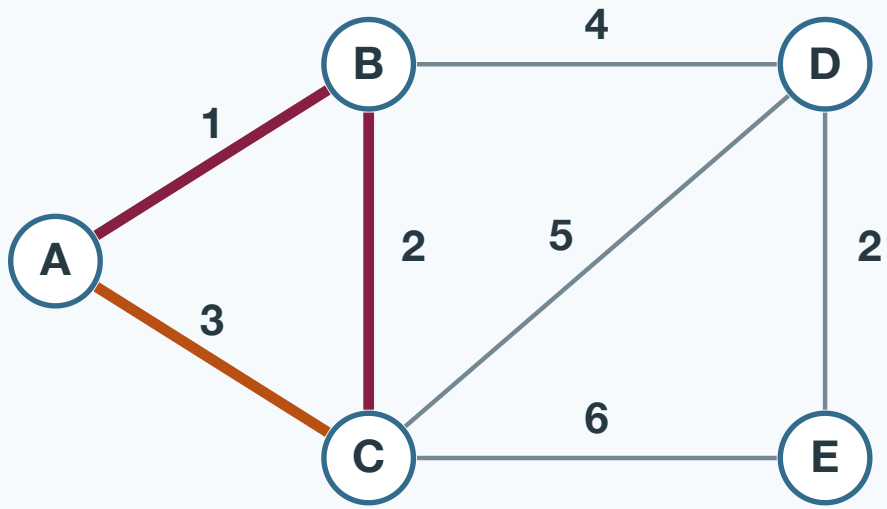
Objective: minimize the total cost of all installed cables, paying for each once. A shortest-path problem instead adds costs along one route.

Why a cheapest connected network is a tree

Suppose installed cables include A–B–C–A. Remove A–C: A and C stay connected through B, saving **$3,000**.

Deleting an edge from any cycle preserves connectivity. With positive installation costs, a cheapest connected network has no cycles.

> A **tree** is connected and has no cycle. A **spanning tree** includes every node of the original graph.



Orange: remove A–C. Costs in $1,000.

A **minimum spanning tree (MST)** minimizes $\sum_{e \in T} c_e$, where T is the chosen spanning tree and c_e is the cost of edge e.

Five buildings need four cables: a tree on n nodes has $n - 1$ edges.

Prim: grow one connected tree

1. Start from one node; here start at A.
2. Consider every edge with one endpoint already reached and the other not yet reached.
3. Select a cheapest such edge and add its new node.
4. Repeat until every node has been reached.

> The reached and unreached node sets form a **cut**: a partition of the nodes into two groups. A crossing edge has one endpoint in each group.

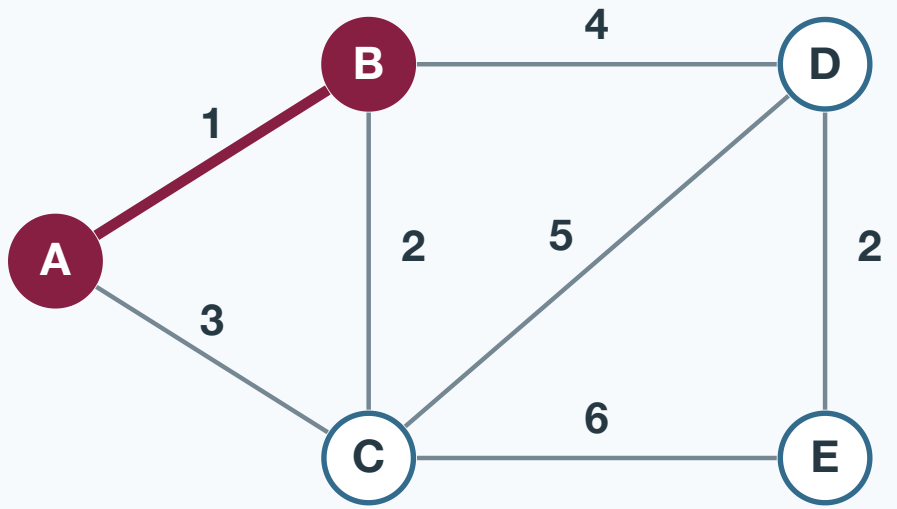
Prim compares single crossing-edge costs, not distances from A. Both Prim and Kruskal break cost ties by alphabetic endpoint pairs: B–C before D–E.

Prim step 1: connect A to B

Crossing edge	Cost
A–B	1
A–C	3

> Choose A–B. Reached nodes: A, B. Total selected cost: 1.

The next candidates must connect {A,B} to {C,D,E}.



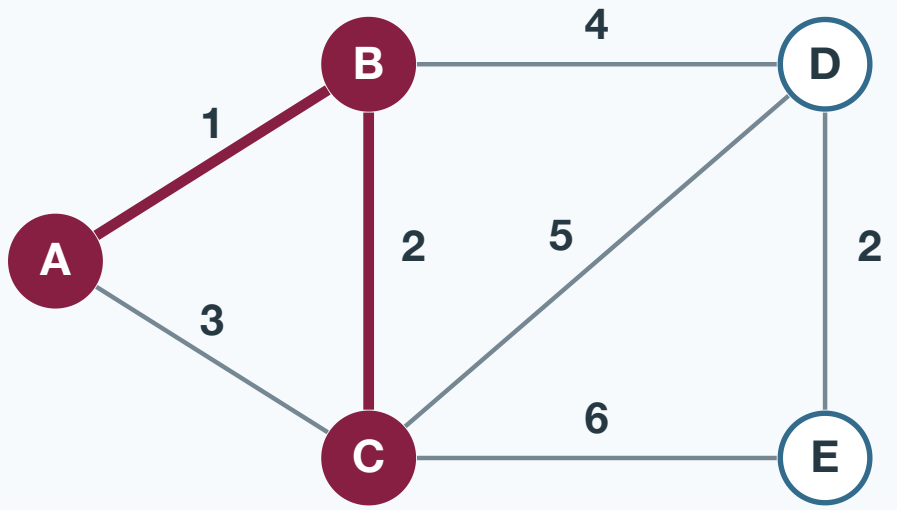
Maroon: selected cables. Costs are in $1,000.

Prim step 2: connect C through B

Crossing edge	Cost
A–C	3
B–C	2
B–D	4

> Choose B–C. Reached nodes: A, B, C. Total selected cost: 3.

Edge A–C now has both endpoints reached, so it is no longer a crossing edge.



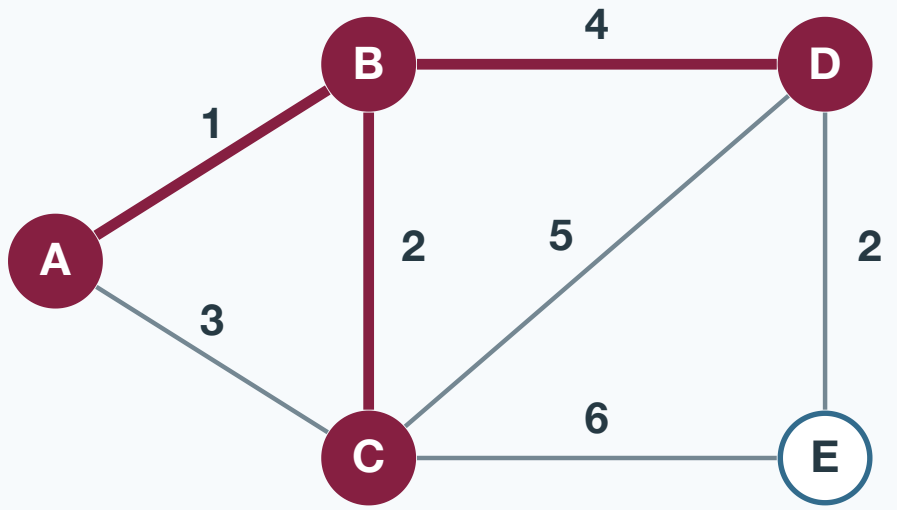
Maroon: selected cables. Costs are in $1,000.

Prim step 3: connect D through B

Crossing edge	Cost
B–D	4
C–D	5
C–E	6

> Choose B–D. Reached nodes: A, B, C, D. Total selected cost: 7.

Edge D–E has cost 2, but it could not be chosen earlier: both endpoints were outside the growing tree.



Maroon: selected cables. Costs are in $1,000.

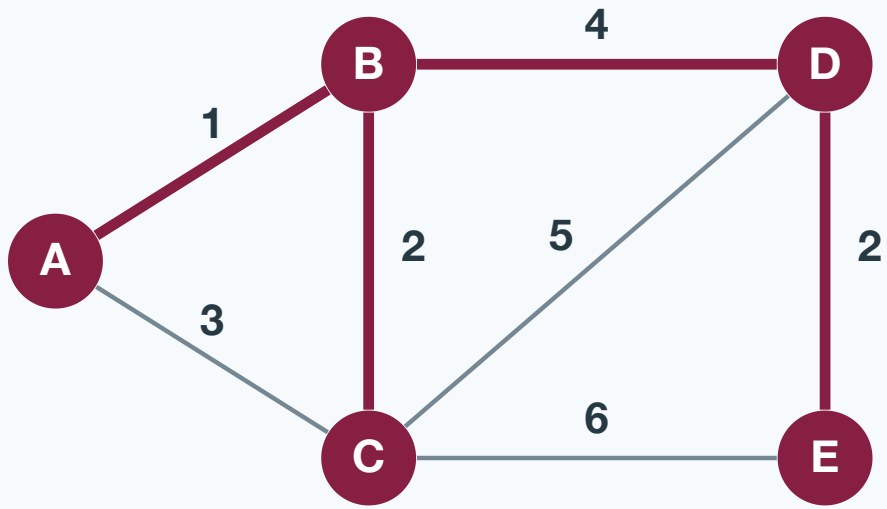
Prim step 4: connect E through D

Crossing edge	Cost
C–E	6
D–E	2

$$\boxed{1 + 2 + 4 + 2 = 9.}$$

All five nodes are connected by four edges. The selected tree has total cost 9.

Every step added one new node, so the selected edges never formed a cycle.



Maroon: selected cables. Costs are in $1,000.

Kruskal: add cheap edges without making a cycle

1. Sort every edge by increasing cost.
2. Start with no selected edges: each node is its own connected component.
3. Consider the next edge. Add it if its endpoints lie in different selected components; otherwise reject it.
4. Stop after selecting n – 1 edges.

> A **connected component** is a maximal group of nodes joined by paths using selected edges. Joining two different components cannot create a cycle.

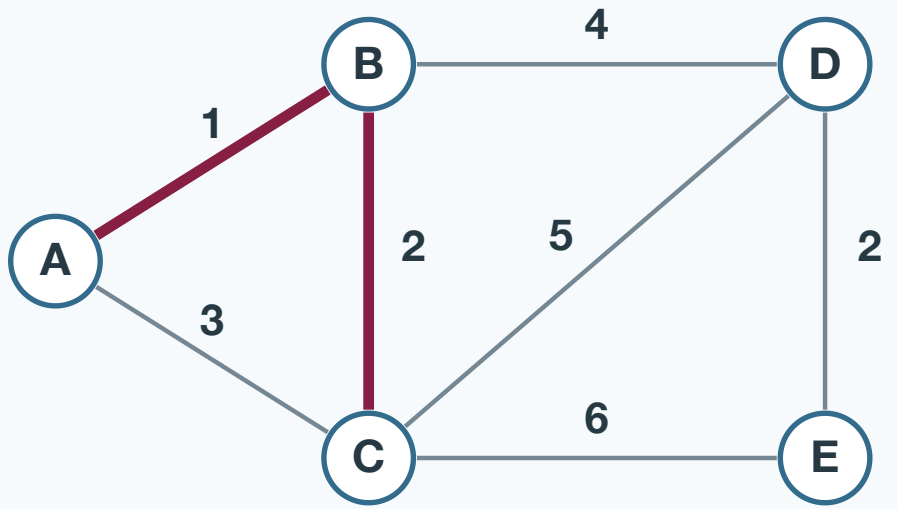
Unlike Prim, Kruskal may build several separate trees before joining them.

Kruskal steps 1–2: accept A–B, then B–C

Consider	Decision	Total cost
A–B, cost 1	Accept	1
B–C, cost 2	Accept	3

> Selected components are now {A,B,C}, {D}, and {E}.

The tied cost-2 edges are processed alphabetically, so B–C comes before D–E.



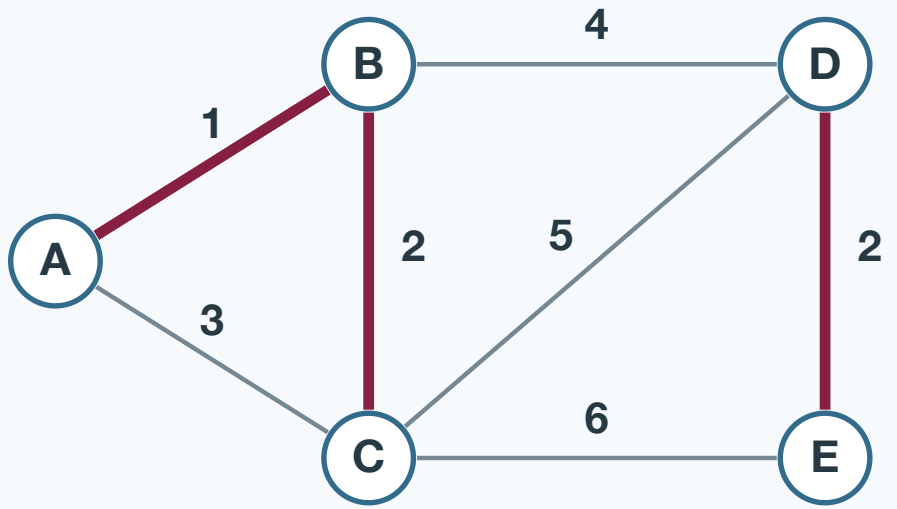
Maroon: selected cables. Costs are in $1,000.

Kruskal step 3: accept D–E

D and E lie in different selected components, so adding D–E creates no cycle.

> Components: {A,B,C} and {D,E}. Selected cost: 1 + 2 + 2 = 5.

The selected edges are not connected yet. Kruskal permits this intermediate state.



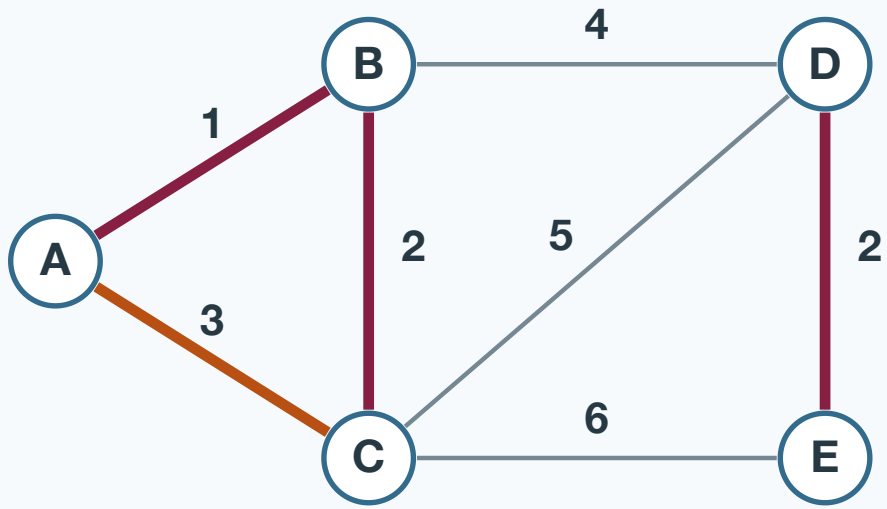
Maroon: selected cables. Costs are in $1,000.

Kruskal step 4: reject A–C

A and C are already connected by A–B–C. Adding A–C would create the cycle A–B–C–A.

> Reject A–C, even though its cost 3 is smaller than the next candidate cost 4.

Components and selected cost stay unchanged: {A,B,C}, {D,E}, and cost 5.

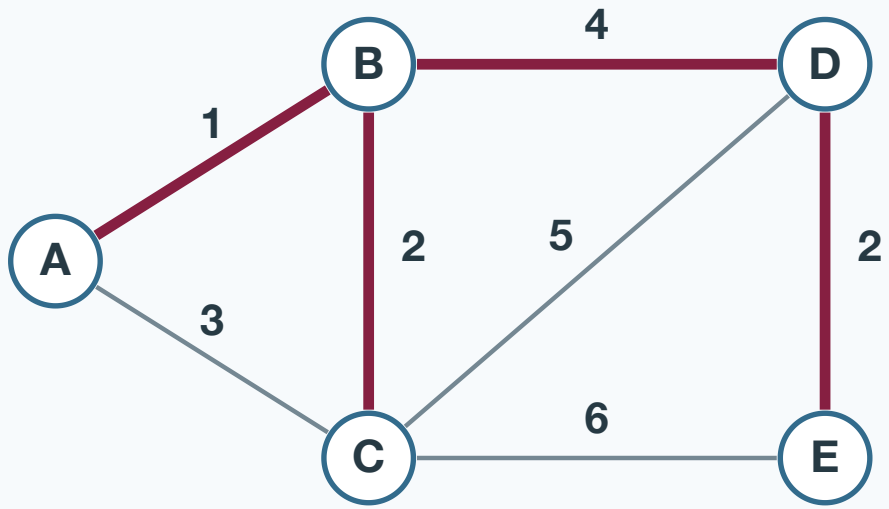


Orange A–C is being tested and rejected. Bold maroon edges remain selected. Costs are in $1,000.

Kruskal step 5: join the two components

Processed edge	Decision
A–B (1)	Accept
B–C (2)	Accept
D–E (2)	Accept
A–C (3)	Reject: cycle
B–D (4)	Accept

Four selected edges connect all five nodes.
Stop with total cost 9, the same result as Prim.



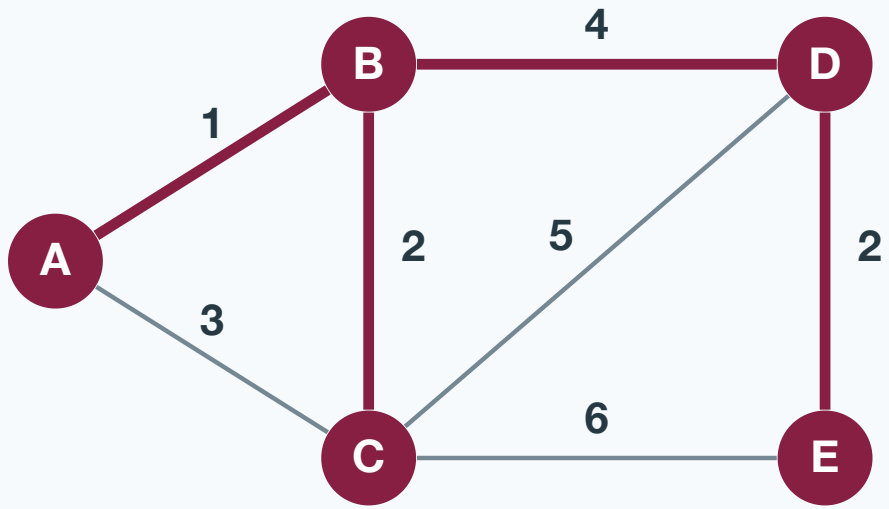
Maroon: selected cables. Costs are in $1,000.

Your turn: does this tree give every shortest route?

Work for about 4 minutes. The highlighted tree has minimum total edge cost 9.

1. What is the cost of the tree route from C to E?
2. Is that a shortest C-to-E route in the original graph?
3. Explain why an MST need not give shortest routes between every pair of nodes.

Compare the cost of a route with the sum of all selected tree-edge costs.



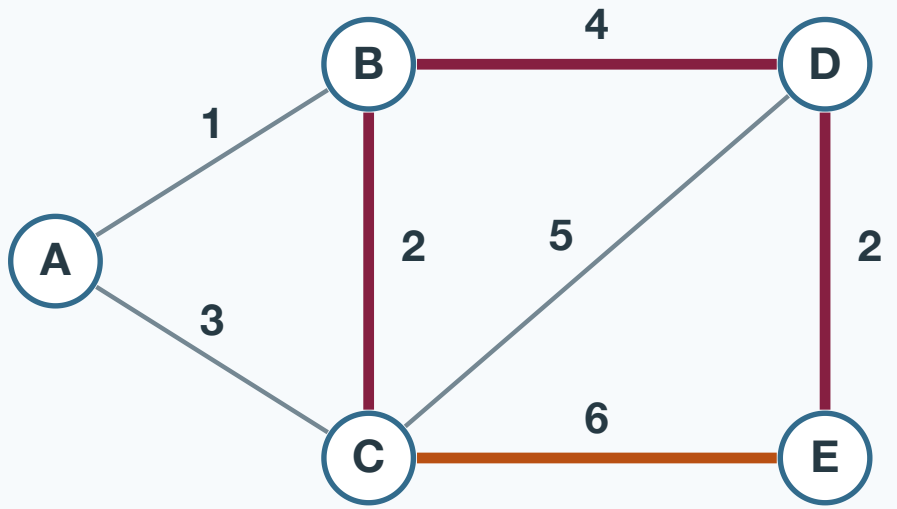
Maroon: selected cables. Costs are in $1,000.

Minimum tree cost and minimum route cost differ

$$C \to B \to D \to E: \quad 2 + 4 + 2 = 8.$$

The original graph also has the direct edge C–E of cost 6, so the tree route is not a shortest route.

> An MST minimizes the sum of all selected edge costs. A shortest-path problem minimizes the cost along one route. These are different objectives.



Maroon: MST route, cost 8. Orange: direct C–E, cost 6. Costs in $1,000.

Why cheapest crossing edges are safe

Cut property: in a connected undirected graph, a cheapest edge across a cut belongs to at least one MST. If it is uniquely cheapest, it belongs to every MST.

An optimal tree omitting that edge must use another crossing edge to connect the two groups. Exchange a suitable crossing edge for the cheapest one; the cost cannot increase.

Cycle property: an edge strictly more expensive than every other edge on a cycle belongs to no MST. With tied costs, do not conclude that every tied edge is excluded.

Appendix: complete exchange arguments and the condition needed for repeated choices.

Four algorithms, two different optimization goals

Algorithm	Objective	Selection/update
Dijkstra	Shortest routes; nonnegative costs	Smallest temporary distance
Bellman–Ford	Shortest routes; allows negative costs	Relax all arcs on each pass
Prim	Minimum total tree-edge cost	Cheapest edge crossing the reached-node cut
Kruskal	Minimum total tree-edge cost	Cheapest next edge that creates no cycle

> Practice naming the objective, checking the assumptions, and recording each algorithm state. A small label is meaningful only after we say what it measures.

Read BT §§7.9–7.10. Appendix: proofs, stopping conditions, and LP connections.

Appendix: why the algorithms work

These pages supply the complete arguments behind the main examples.

- Dijkstra finalization
- Bellman–Ford recurrence and stopping
- Detecting reachable negative cycles
- Shortest paths and the flow LP
- Why a tree has n – 1 edges
- Cut, cycle, Prim, and Kruskal proofs

Dijkstra: prove that a selected label is final

Assume all edge costs are nonnegative. Inductively, every previously finalized node has its true shortest distance. Every finite temporary label is the cost of a discovered route.

Let u be the next selected node. Suppose a shorter route to u existed. On that route, let v be the first unfinalized node and w its finalized predecessor. When w was finalized, relaxing w→v gave d(v) no larger than the cost of the route's prefix to v.

$$d(u) \le d(v) \le \text{prefix cost to } v \le \text{full competing route cost to } u.$$

> The first inequality follows from selecting the smallest temporary label. The last follows from nonnegative remaining costs. This contradicts a competing route costing less than d(u).

The base case is s, with distance 0. Induction proves every finalized label is correct.

Bellman–Ford: prove the recurrence

For $k = 0$, only the empty walk at s is allowed. Suppose row $k - 1$ gives the minimum costs using at most $k - 1$ arcs.

Any walk to v with at most k arcs either already uses at most $k - 1$ arcs, or ends with an arc (u, v). Removing that last arc leaves a walk to u with at most $k - 1$ arcs.

$$D_k(v) = \min\{D_{k-1}(v),\ \min_{(u,v)\in E}(D_{k-1}(u) + c_{uv})\}.$$

> Every candidate in this minimum is attained by an allowed walk whenever finite. Every allowed walk has cost at least one of these candidate minima. The formula is therefore exact.

If no negative cycle is reachable from s, removing cycles cannot increase route cost. A shortest route can be simple and uses at most n – 1 arcs.

Why improvement on pass n detects a negative cycle

If $D_n(v) < D_{n-1}(v)$, a walk attaining $D_n(v)$ must use n arcs. A walk with n arcs visits $n + 1$ node positions, so some node repeats.

If every cycle on that walk had nonnegative cost, deleting a cycle would give a walk of at most n – 1 arcs costing no more. This contradicts the strict improvement. Thus a negative cycle is reachable from s and can reach v.

> Conversely, if no label changes on a full pass, every reachable arc satisfies $d(v) \leq d(u) + c_{uv}$. Summing these inequalities around a reachable directed cycle implies its total cost is nonnegative.

Thus a reachable negative cycle forces some strict improvement on pass n. To find every affected node, start from improved nodes and follow directed arcs forward.

Why a shortest route solves the one-unit flow LP

Assume t is reachable from s and there is no negative-cost directed cycle. Decompose a feasible one-unit flow into path amounts $\alpha_P \geq 0$ and cycle amounts $\beta_C \geq 0$, with $\sum_P \alpha_P = 1$. Here P is an s-to-t path and C a directed cycle.

Cycle contributions have nonnegative cost. Each path has cost at least L, the shortest s-to-t path cost. Therefore every feasible flow has cost at least L.

$$\sum_P \alpha_P \operatorname{cost}(P) + \sum_C \beta_C \operatorname{cost}(C) \;\geq\; L \sum_P \alpha_P = L.$$

> Sending one unit along a shortest path attains L, proving equality of the two optima.

This proves that an optimal path flow exists, not that every optimal flow is a single path. A negative cycle anywhere in the uncapacitated LP would allow an unbounded circulation, even if it is unreachable from s.

Why a tree on n nodes has n – 1 edges

For n = 1, the tree has no edges. For n ≥ 2, take a longest simple path. An endpoint has only one neighbor: any extra neighbor would extend the path or create a cycle. Such a node is called a leaf.

Remove that leaf and its single incident edge. The remaining graph is connected and has no cycle, so it is a tree on n – 1 nodes.

$$|E(T)| = |E(T \setminus \{\text{leaf}\})| + 1 = (n-2) + 1 = n - 1.$$

> Removing any tree edge disconnects the tree. Adding a new edge joins nodes already connected by a unique path, creating one cycle. Deleting an edge on that cycle gives another spanning tree.

Every connected graph has a spanning tree by cycle deletion; a disconnected graph has none. A connected graph with n – 1 edges must be a tree: deleting a cycle edge would leave too few edges for connectivity.

The cut exchange argument preserves earlier choices

Let F be selected edges contained in some MST T. Choose a cut that no edge of F crosses. Let e be a cheapest edge crossing that cut.

If T already contains e, nothing is needed. Otherwise add e to T, creating a unique cycle. The path in T between the endpoints of e must cross the cut at an edge f.

$$c_e \le c_f, \qquad T' = T + e - f, \qquad \operatorname{cost}(T') \le \operatorname{cost}(T).$$

> Because no selected edge crosses the cut, f is not in F. Thus T′ is an MST containing every earlier choice and e. This is the repeated-choice form of the cut property.

For a uniquely cheapest e, any MST omitting e would strictly improve under the exchange. Hence every MST contains e.

The cycle exchange argument

Let e be the unique most expensive edge on an undirected cycle. Suppose an MST T contains e. Removing e from T creates two components.

The rest of the original cycle contains an edge f crossing between those components. That edge is cheaper than e. Replacing e by f reconnects T with no cycle and strictly lowers its cost, a contradiction.

> If e only ties for the maximum, the replacement may preserve cost. Then there exists an MST omitting e; it is incorrect to conclude that every MST omits e.

Kruskal’s rejection has a simpler immediate reason: the endpoints are already connected by selected edges. Adding the current edge would violate the requirement that selected edges have no cycle.

Why Prim and Kruskal finish at an MST

Prim: let F be the current tree. Its reached-node cut has no selected edge crossing it. The next cheapest crossing edge is safe by the cut exchange argument.

Kruskal: when an edge joins two selected components, take the cut around one component. Any cheaper crossing edge would already have been processed and would have joined components, so the accepted edge is cheapest across that cut.

Starting from the empty set, induction keeps the chosen edges inside at least one MST. When n – 1 edges have been selected without a cycle, they form a spanning tree and hence an MST.

Simple implementations: Prim $O(n^2 + m)$ by scans; Kruskal $O(m \log m)$ to sort, with efficient component tracking. Both permit negative edge costs: their proofs compare edges, not cumulative distances.