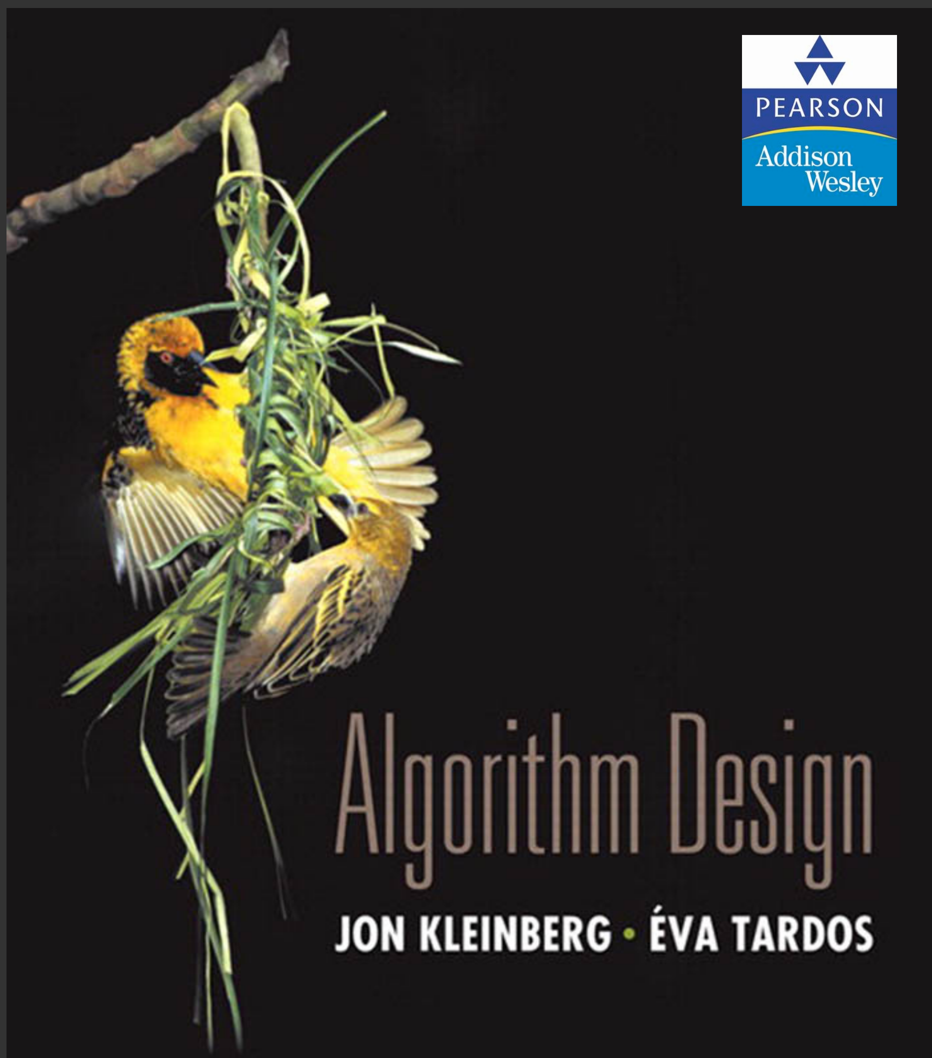
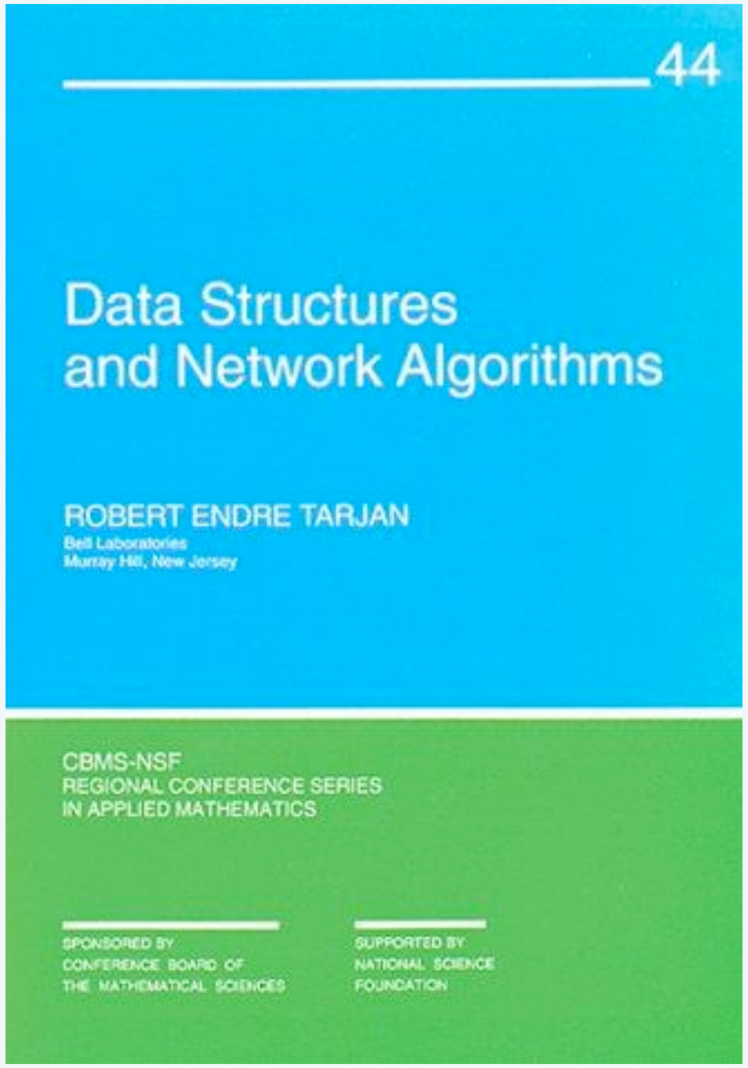


4. Greedy Algorithms II

- *minimum spanning trees*
- *Prim, Kruskal*

Lecture slides by Kevin Wayne

http://www.cs.princeton.edu/~wayne/kleinberg-tardos



SECTION 6.1

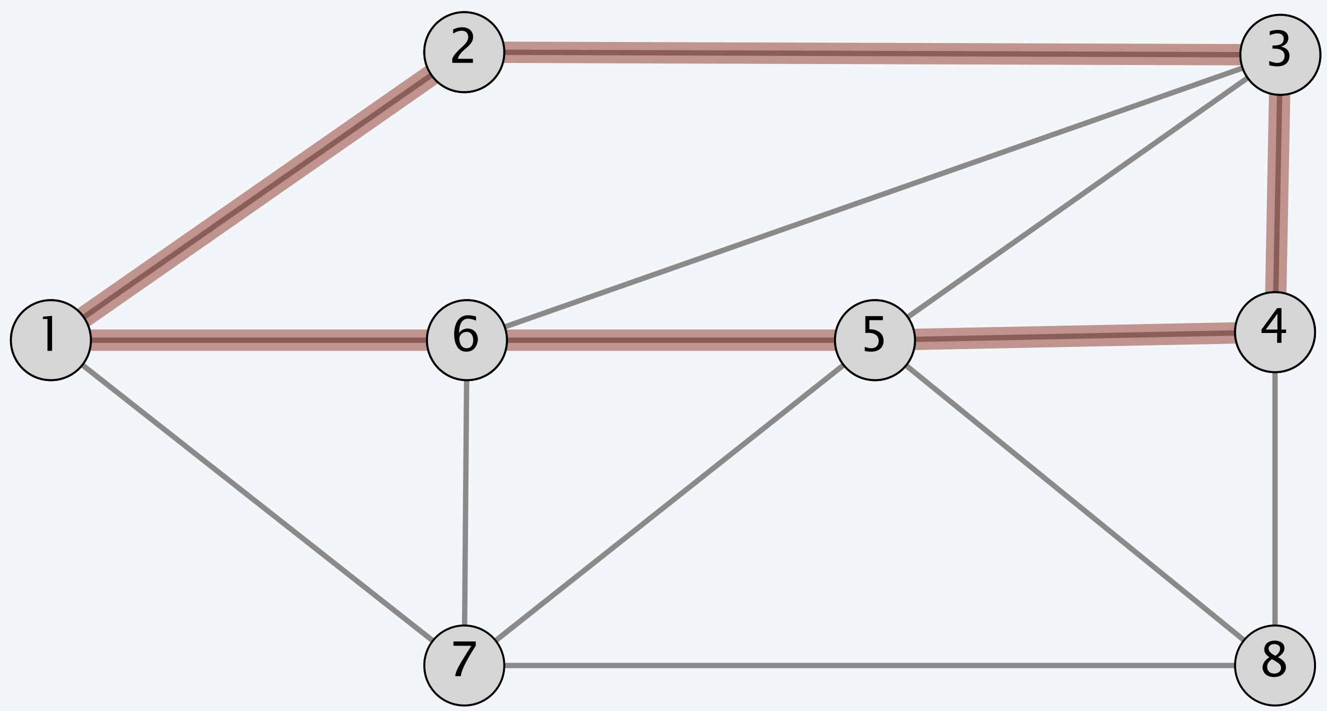
4. GREEDY ALGORITHMS II

- *minimum spanning trees*
- *Prim, Kruskal*

Cycles

Def. A path is a sequence of edges which connects a sequence of nodes.

Def. A cycle is a path with no repeated nodes or edges other than the starting and ending nodes.



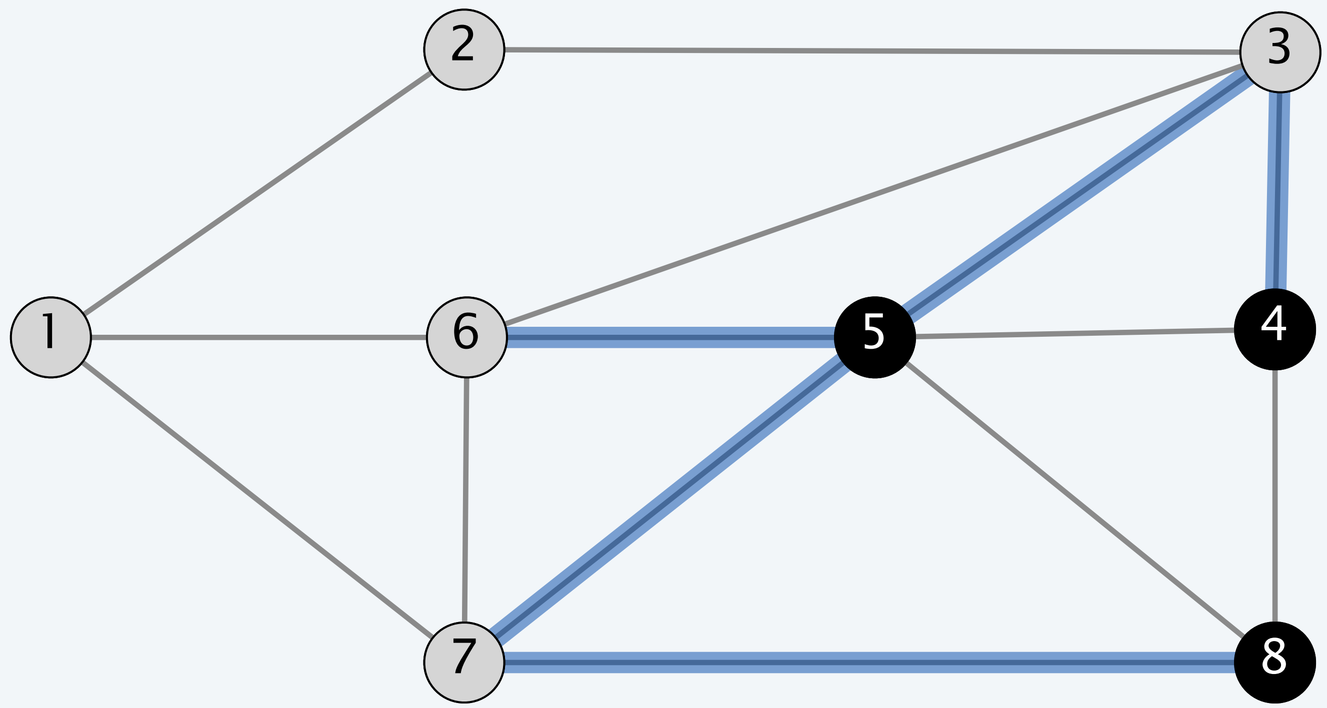
path P = { (1, 2), (2, 3), (3, 4), (4, 5), (5, 6) }

cycle C = { (1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 1) }

Cuts

Def. A cut is a partition of the nodes into two nonempty subsets S and $V - S$.

Def. The cutset of a cut S is the set of edges with exactly one endpoint in S.



cut S = { 4, 5, 8 }

cutset D = { (3, 4), (3, 5), (5, 6), (5, 7), (8, 7) }

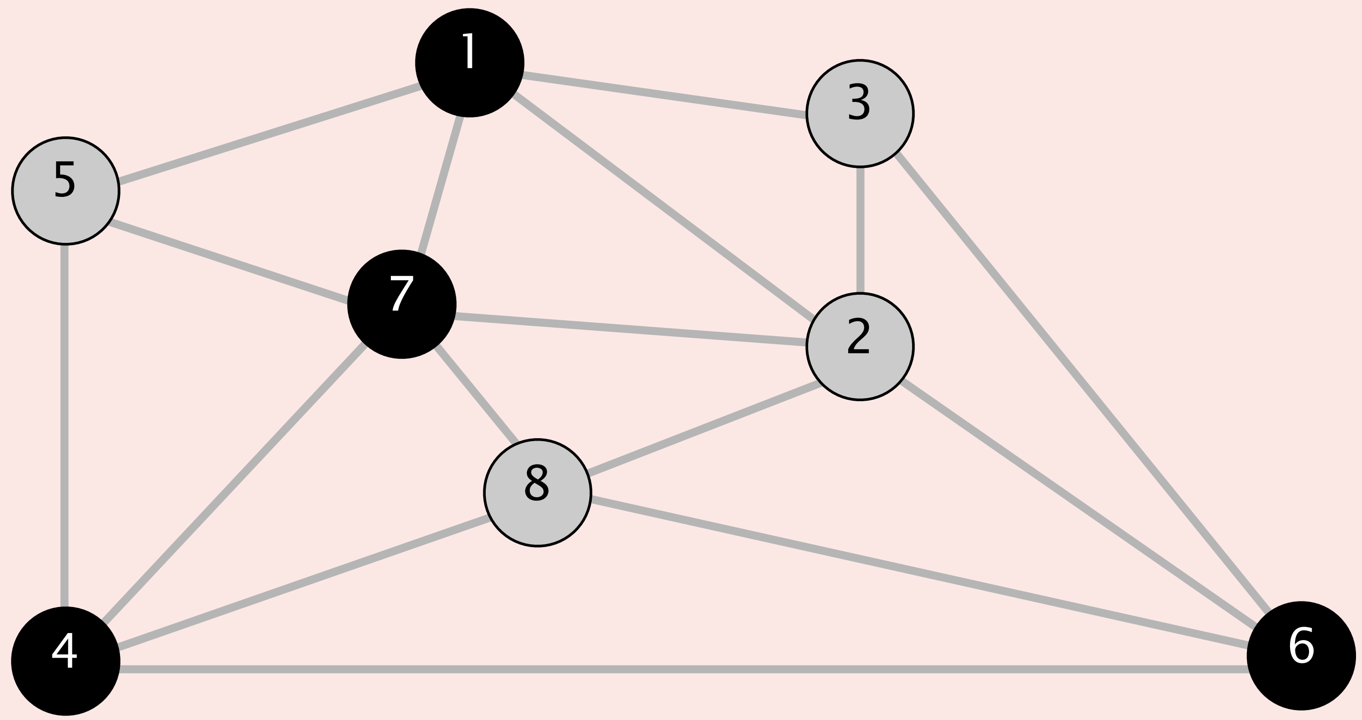
Consider the cut S = { 1, 4, 6, 7 }. Which edge is in the cutset of S?

A. S is not a cut (not connected)

B. 1–7

C. 5–7

D. 2–3



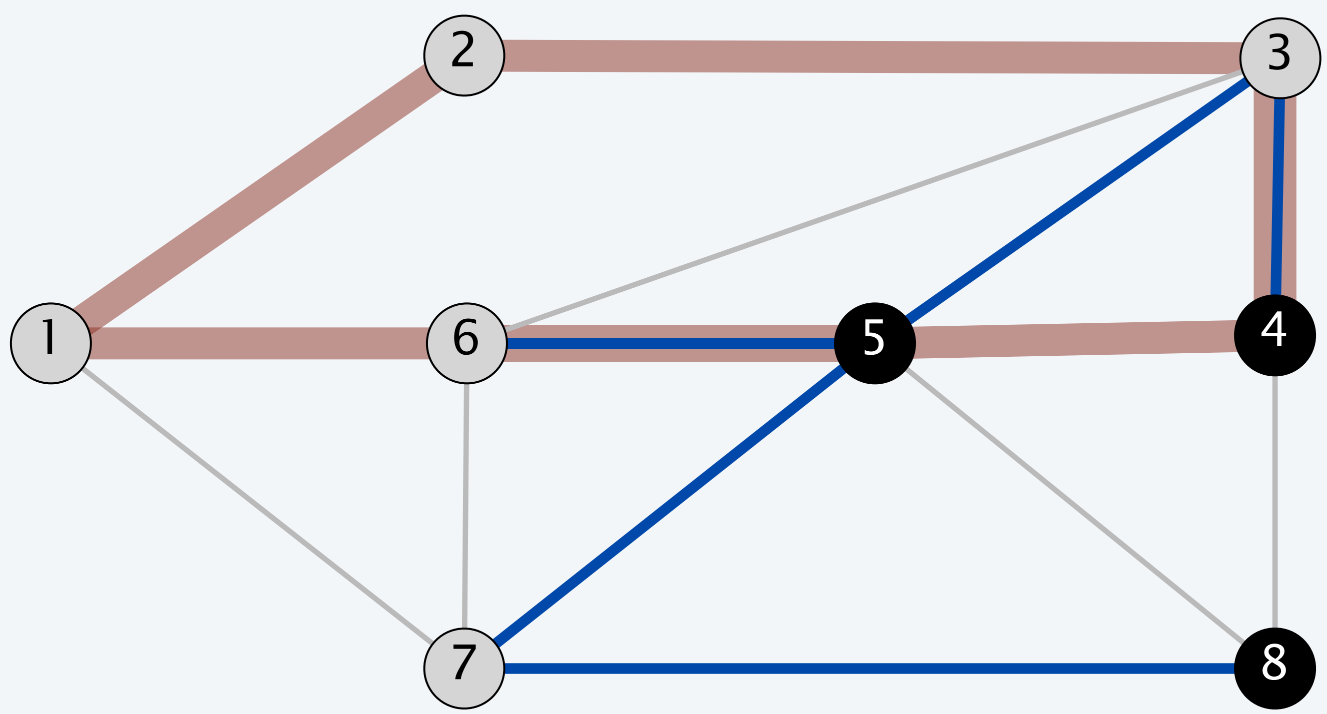
Minimum spanning trees: quiz 2

Let C be a cycle and let D be a cutset. How many edges do C and D have in common? Choose the best answer.

- **A.** 0
- **B.** 2
- **C.** not 1
- **D.** an even number

Cycle–cut intersection

Proposition. A cycle and a cutset intersect in an even number of edges.



cycle C = { (1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 1) }

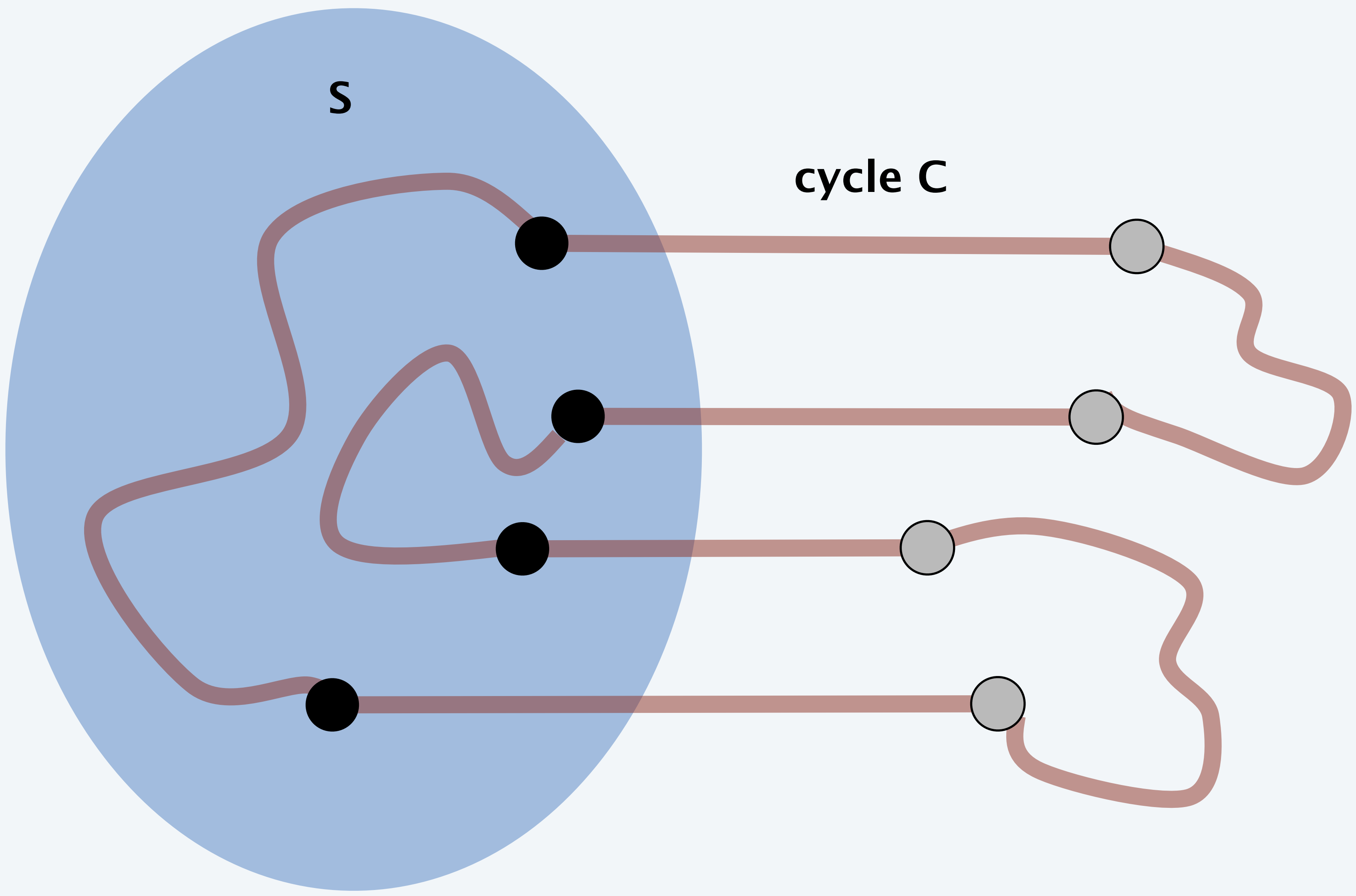
cutset D = { (3, 4), (3, 5), (5, 6), (5, 7), (8, 7) }

intersection $C \cap D$ = { (3, 4), (5, 6) }

Cycle-cut intersection

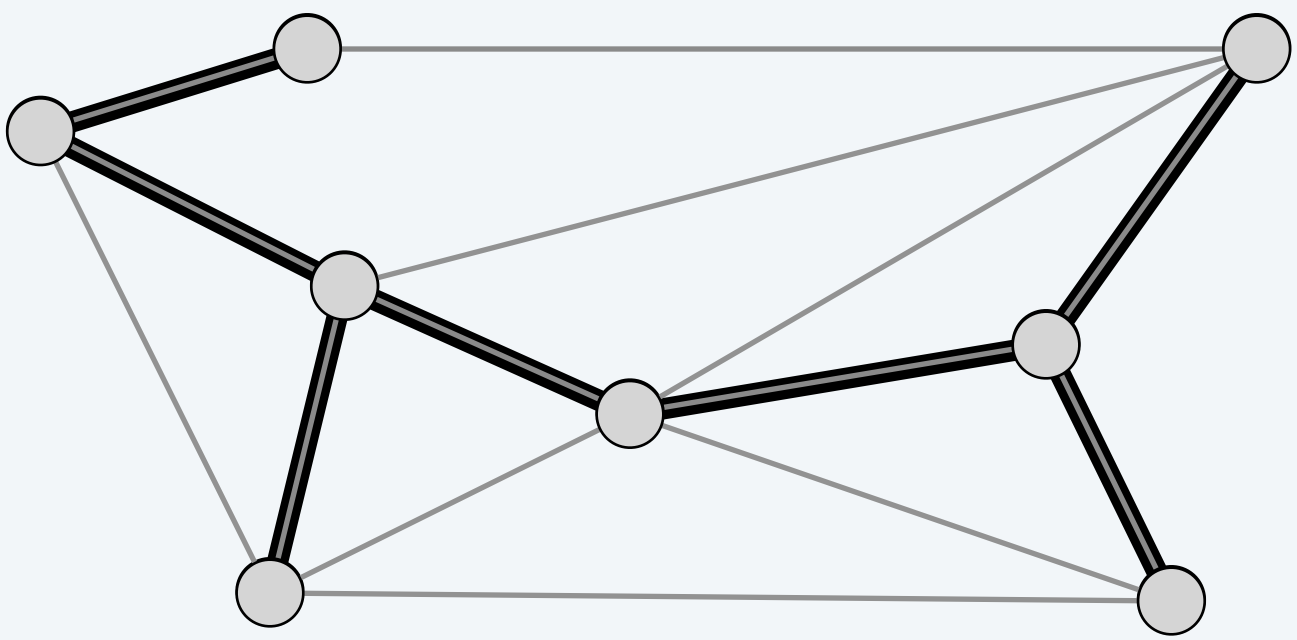
Proposition. A cycle and a cutset intersect in an even number of edges.

Pf. [by picture]



Spanning tree definition

Def. Let $H = (V, T)$ be a subgraph of an undirected graph $G = (V, E)$.
H is a spanning tree of G if H is both acyclic and connected.



graph G = (V, E)
spanning tree H = (V, T)

Minimum spanning trees: quiz 3

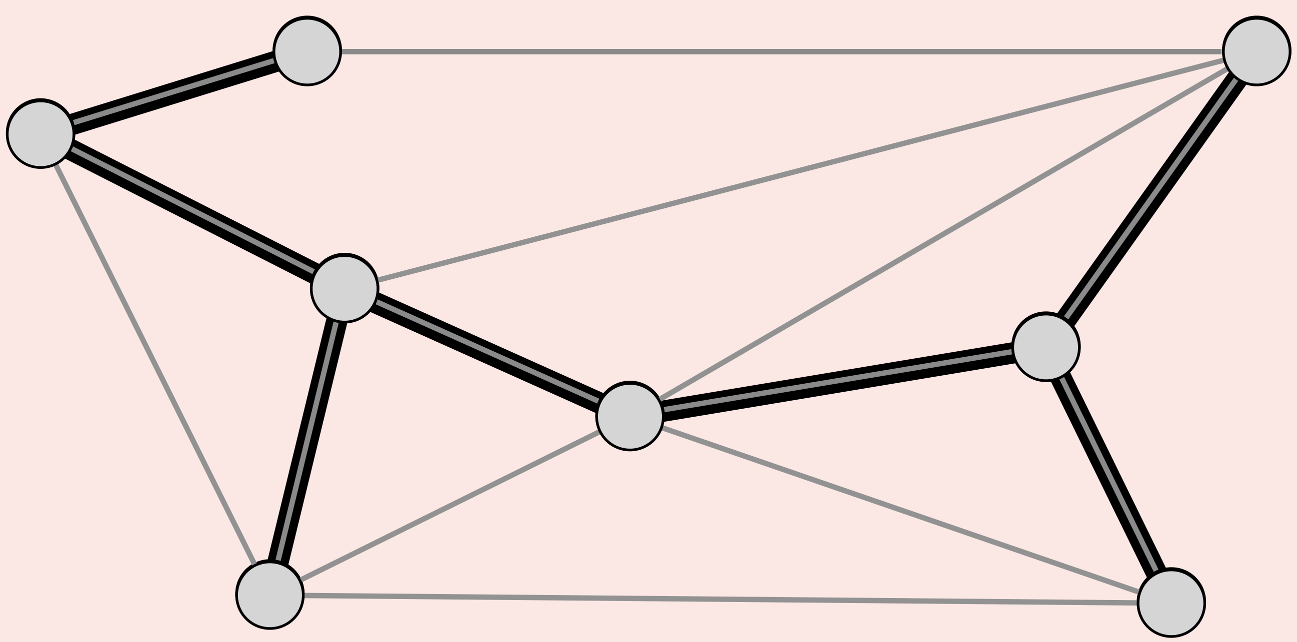
Which of the following properties are true for all spanning trees H?

A. Contains exactly $|V| - 1$ edges.

B. The removal of any edge disconnects it.

C. The addition of any edge creates a cycle.

D. All of the above.



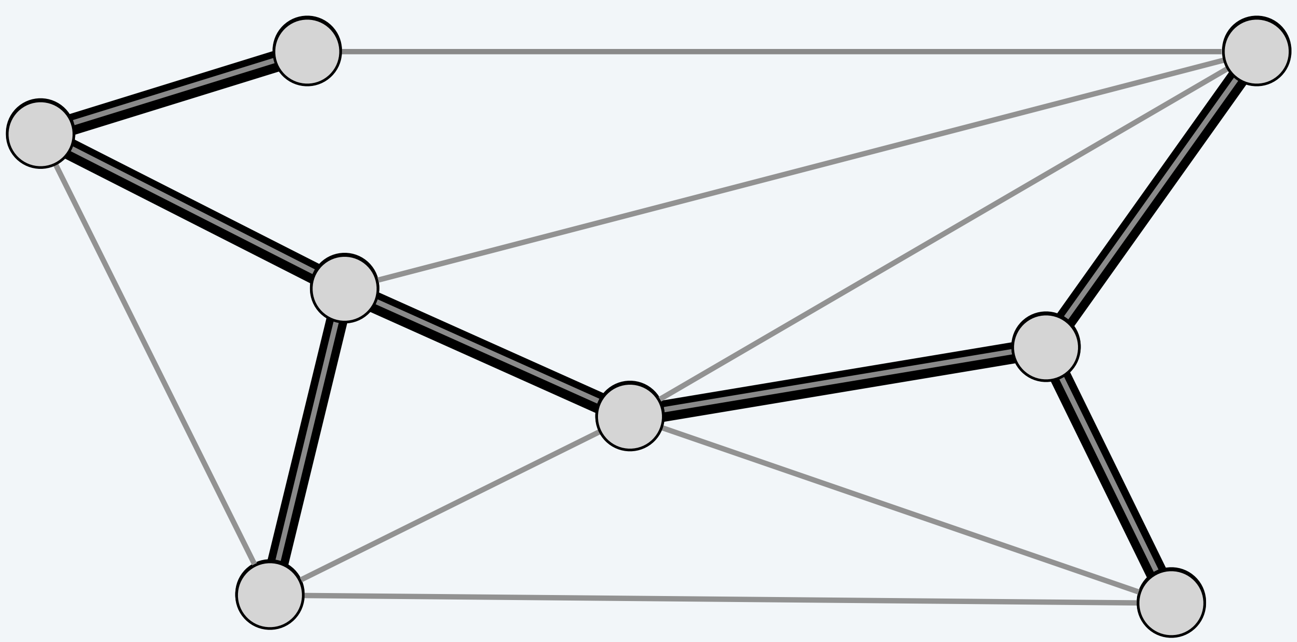
graph G = (V, E)

spanning tree H = (V, T)

Spanning tree properties

Proposition. Let $H = (V, T)$ be a subgraph of an undirected graph $G = (V, E)$.
Then, the following are equivalent:

- H is a spanning tree of G.
- H is acyclic and connected.
- H is connected and has $|V| - 1$ edges.
- H is acyclic and has $|V| - 1$ edges.
- H is minimally connected: removal of any edge disconnects it.
- H is maximally acyclic: addition of any edge creates a cycle.



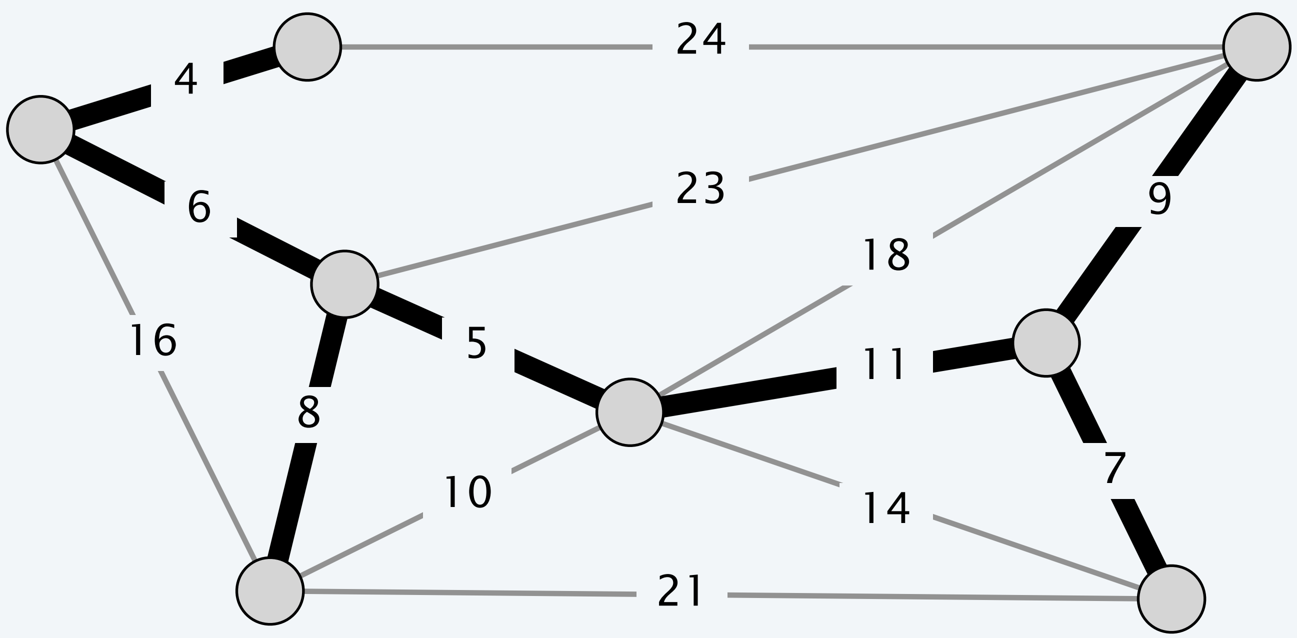
graph G = (V, E)
spanning tree H = (V, T)

A tree containing a cycle

https://maps.roadtrippers.com/places/46955/photos/374771356

Minimum spanning tree (MST)

Def. Given a connected, undirected graph $G = (V, E)$ with edge costs c_e, a minimum spanning tree (V, T) is a spanning tree of G such that the sum of the edge costs in T is minimized.



MST cost = 50 = 4 + 6 + 8 + 5 + 11 + 9 + 7

Cayley's theorem. The complete graph on n nodes has n^{n-2} spanning trees.

can't solve by brute force

Minimum spanning trees: quiz 4

Suppose that you change the cost of every edge in G as follows.
For which is every MST in G an MST in G′ (and vice versa)?
Assume c(e) > 0 for each e.

A. $c'(e) = c(e) + 17$.

B. $c'(e) = 17 \times c(e)$.

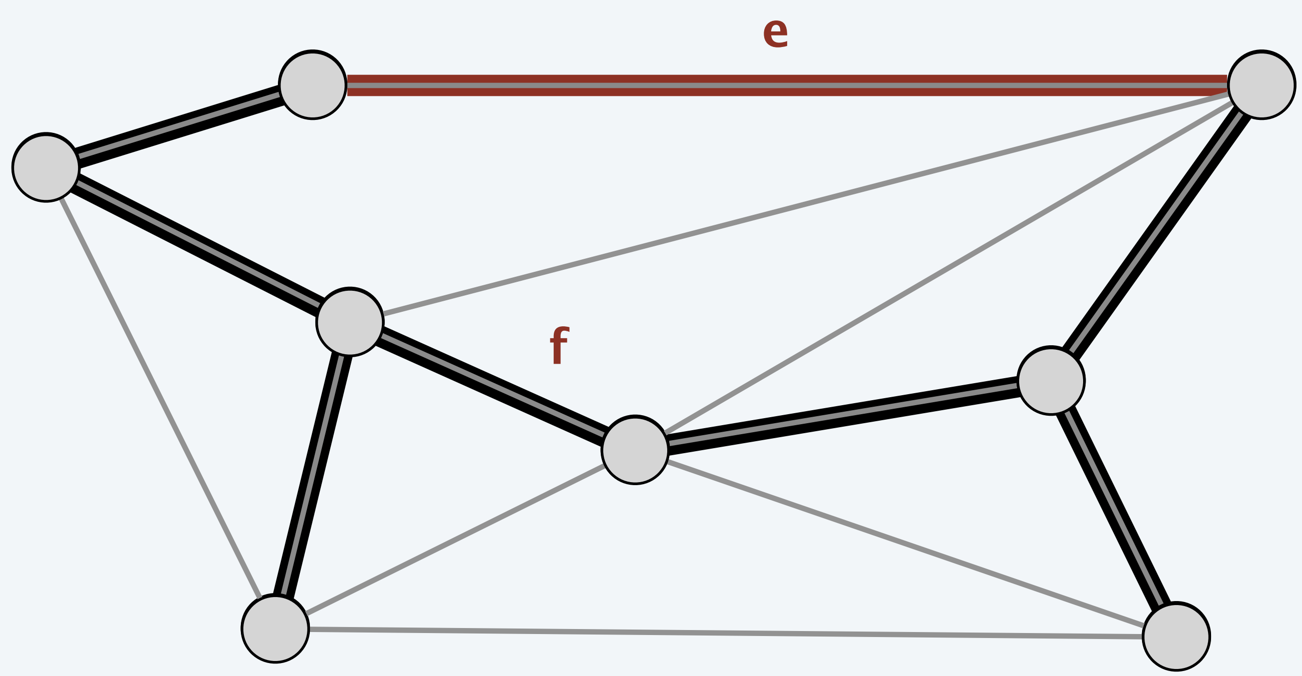
C. $c'(e) = \log_{17} c(e)$.

D. All of the above.

Fundamental cycle

Fundamental cycle. Let $H = (V, T)$ be a spanning tree of $G = (V, E)$.

- For any non tree-edge $e \in E$: $T \cup \{ e \}$ contains a unique cycle, say C.
- For any edge $f \in C$: $(V, T \cup \{ e \} - \{ f \})$ is a spanning tree.



graph G = (V, E)

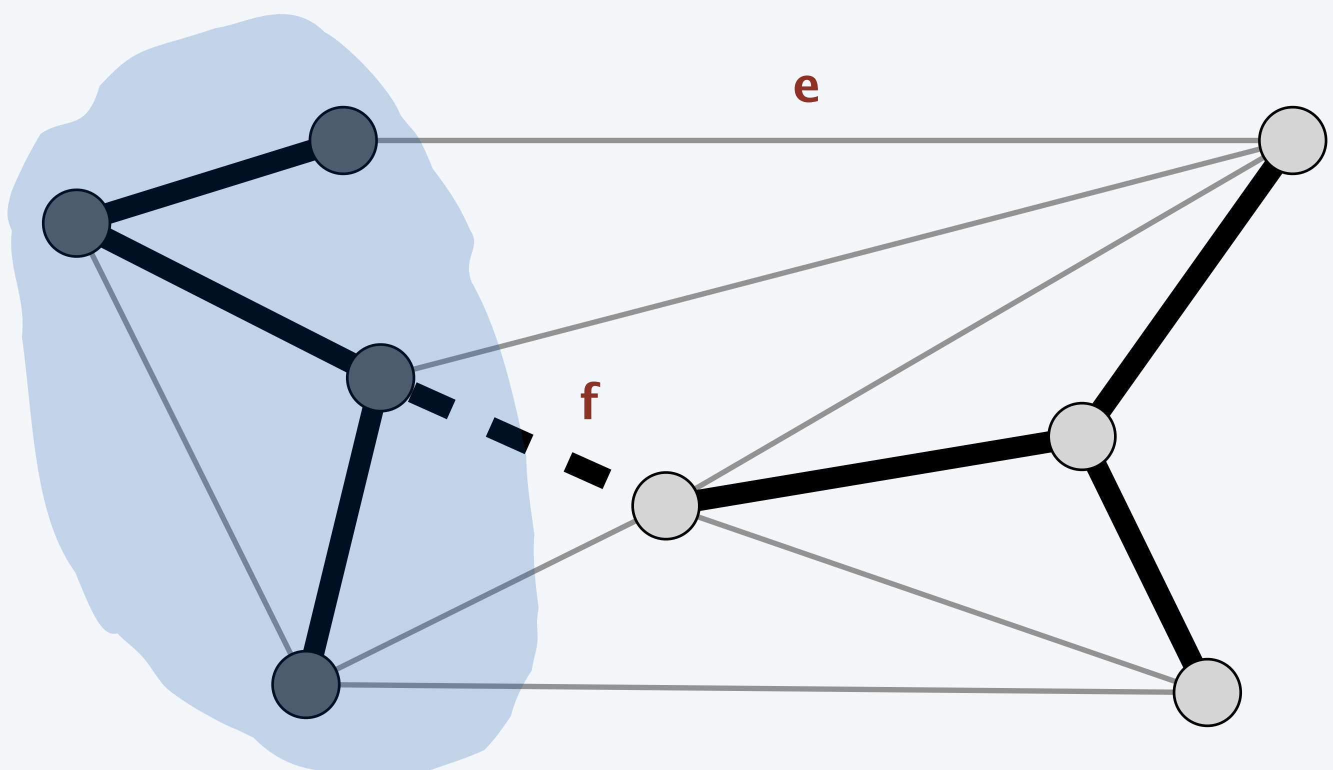
spanning tree H = (V, T)

Observation. If $c_e < c_f$, then (V, T) is not an MST.

Fundamental cutset

Fundamental cutset. Let $H = (V, T)$ be a spanning tree of $G = (V, E)$.

- For any tree edge $f \in T$: $(V, T - \{f\})$ has two connected components.
- Let D denote corresponding cutset.
- For any edge $e \in D$: $(V, T - \{f\} \cup \{e\})$ is a spanning tree.



graph G = (V, E)

spanning tree H = (V, T)

Observation. If $c_e < c_f$, then (V, T) is not an MST.

The greedy algorithm

Red rule.

- Let C be a cycle with no red edges.
- Select an uncolored edge of C of max cost and color it red.

Blue rule.

- Let D be a cutset with no blue edges.
- Select an uncolored edge in D of min cost and color it blue.

Greedy algorithm.

- Apply the red and blue rules (nondeterministically!) until all edges are colored. The blue edges form an MST.
- Note: can stop once $n - 1$ edges colored blue.

Greedy algorithm: proof of correctness

Color invariant. There exists an MST (V, T^*) containing every blue edge and no red edge.

Pf. [by induction on number of iterations]

Base case. No edges colored $\Rightarrow$ every MST satisfies invariant.

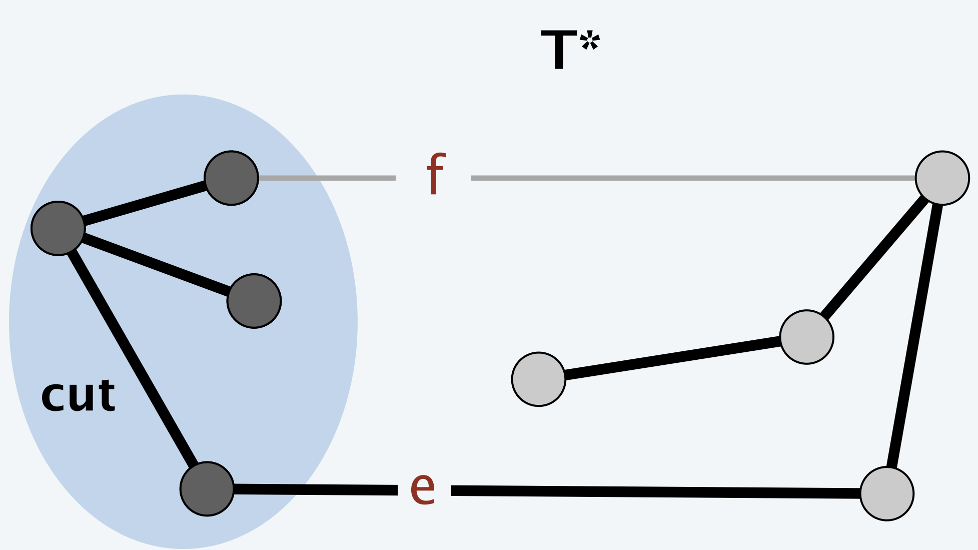
Greedy algorithm: proof of correctness

Color invariant. There exists an MST (V, T^*) containing every blue edge and no red edge.

Pf. [by induction on number of iterations]

Induction step (blue rule). Suppose color invariant true before blue rule.

- let D be chosen cutset, and let f be edge colored blue.
- if $f \in T^*$, then T^* still satisfies invariant.
- Otherwise, consider fundamental cycle C by adding f to T^*.
- let $e \in C$ be another edge in D.
- e is uncolored and $c_e \geq c_f$ since
  - $e \in T^* \Rightarrow e$ not red
  - blue rule $\Rightarrow e$ not blue and $c_e \geq c_f$
- Thus, $T^* \cup \{ f \} - \{ e \}$ satisfies invariant.



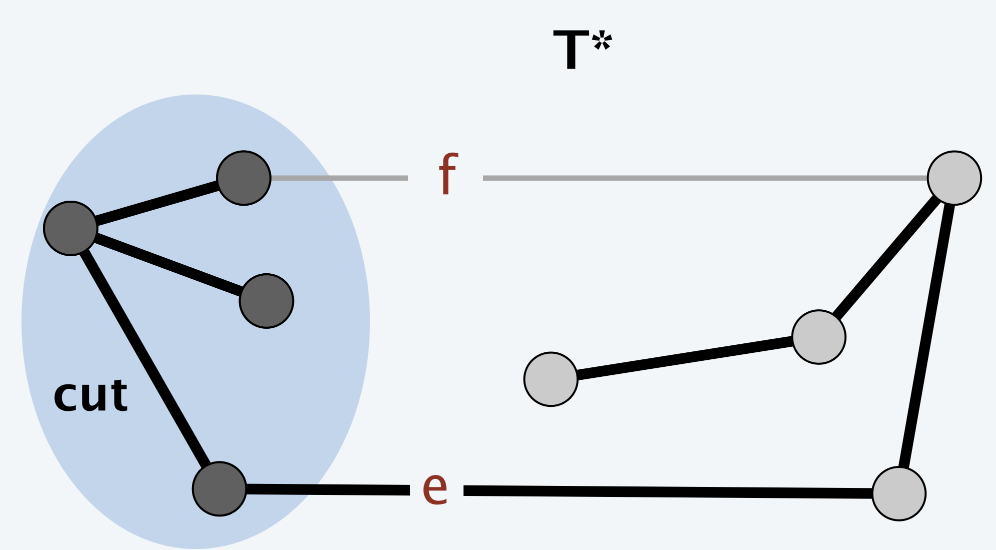
Greedy algorithm: proof of correctness

Color invariant. There exists an MST (V, T^*) containing every blue edge and no red edge.

Pf. [by induction on number of iterations]

Induction step (red rule). Suppose color invariant true before red rule.

- let C be chosen cycle, and let e be edge colored red.
- if $e \notin T^*$, then T^* still satisfies invariant.
- Otherwise, consider fundamental cutset D by deleting e from T^*.
- let $f \in D$ be another edge in C.
- f is uncolored and $c_e \geq c_f$ since
  - $f \notin T^* \Rightarrow f$ not blue
  - red rule $\Rightarrow f$ not red and $c_e \geq c_f$
- Thus, $T^* \cup \{ f \} - \{ e \}$ satisfies invariant. ▪



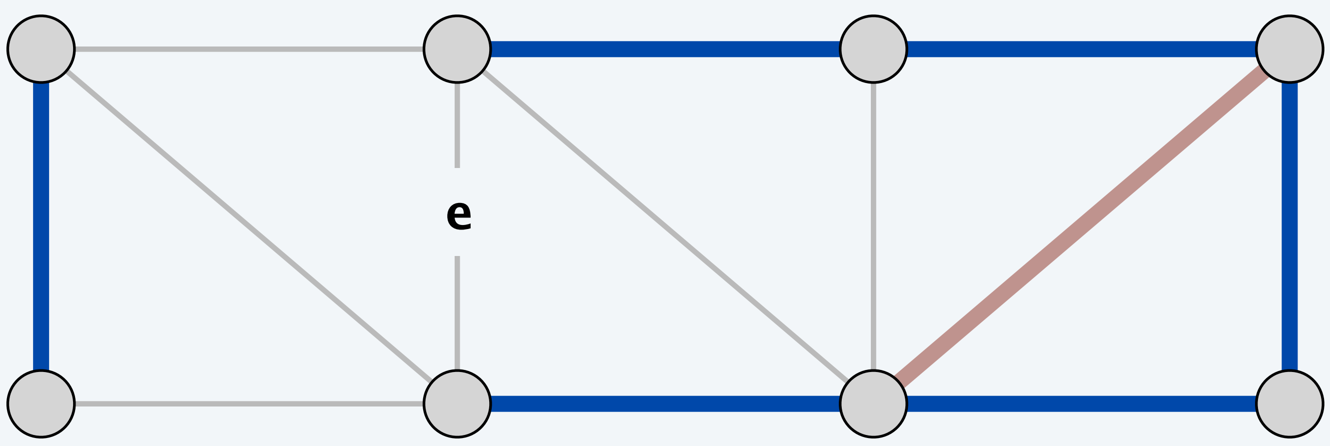
Greedy algorithm: proof of correctness

Theorem. The greedy algorithm terminates. Blue edges form an MST.

Pf. We need to show that either the red or blue rule (or both) applies.

- Suppose edge e is left uncolored.
- Blue edges form a forest.
- Case 1: both endpoints of e are in same blue tree.

  $\Rightarrow$ apply red rule to cycle formed by adding e to blue forest.



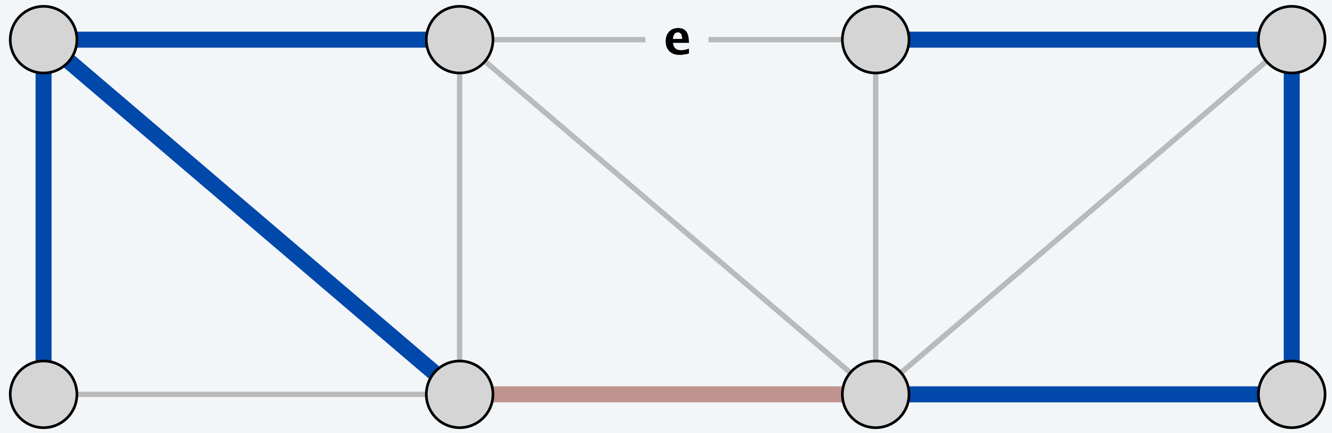
Case 1

Greedy algorithm: proof of correctness

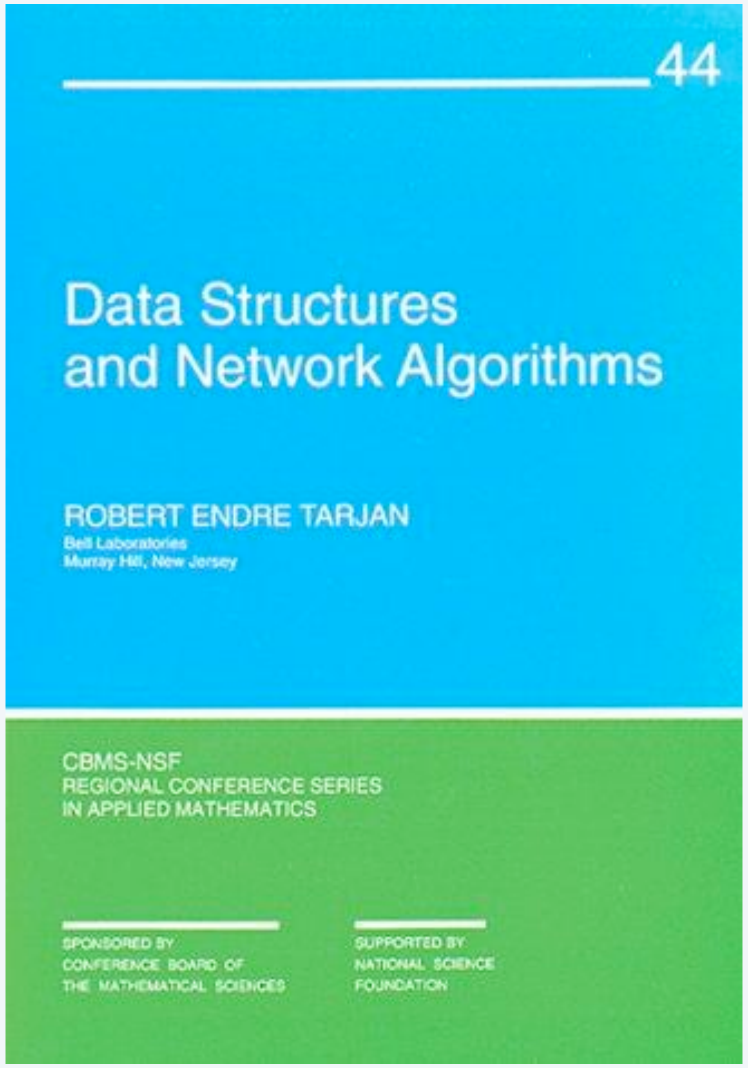
Theorem. The greedy algorithm terminates. Blue edges form an MST.

Pf. We need to show that either the red or blue rule (or both) applies.

- Suppose edge e is left uncolored.
- Blue edges form a forest.
- Case 1: both endpoints of e are in same blue tree.
  $\Rightarrow$ apply red rule to cycle formed by adding e to blue forest.
- Case 2: both endpoints of e are in different blue trees.
  $\Rightarrow$ apply blue rule to cutset induced by either of two blue trees. ▪



Case 2



SECTION 6.2

4. GREEDY ALGORITHMS II

- *minimum spanning trees*
- *Prim, Kruskal*

Prim's algorithm

Initialize $S = \{ s \}$ for any node s, $T = \varnothing$.

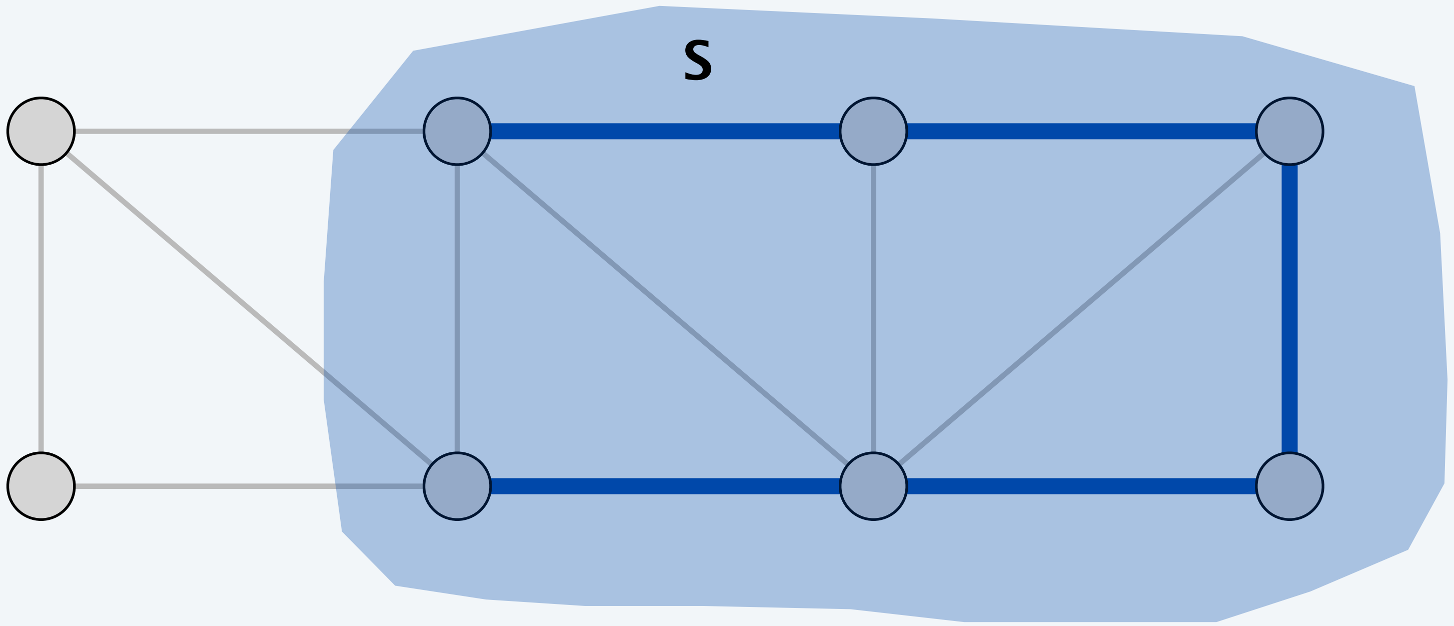
Repeat $n - 1$ times:

- Add to T a min-cost edge with exactly one endpoint in S.
- Add the other endpoint to S.

by construction, edges in cutset are uncolored

Theorem. Prim's algorithm computes an MST.

Pf. Special case of greedy algorithm (blue rule repeatedly applied to S). ▪



Prim's algorithm: implementation

Theorem. Prim's algorithm can be implemented to run in $O(m \log n)$ time.
Pf. Implementation almost identical to Dijkstra's algorithm.

PRIM (V, E, c)

$S \leftarrow \varnothing,\ T \leftarrow \varnothing.$

$s \leftarrow$ any node in V.

FOREACH $v \neq s$: $\pi[v] \leftarrow \infty, pred[v] \leftarrow null;\ \pi[s] \leftarrow 0.$

Create an empty priority queue pq.

FOREACH $v \in V$: INSERT$(pq, v, \pi[v])$.

WHILE (IS-NOT-EMPTY(pq))

  $u \leftarrow$ DEL-MIN(pq).

  $S \leftarrow S \cup \{ u \},\ T \leftarrow T \cup \{ pred[u] \}.$

  FOREACH edge $e = (u, v) \in E$ with $v \notin S$:

    IF $(c_e < \pi[v])$

      DECREASE-KEY(pq, v, c_e).

      $\pi[v] \leftarrow c_e;\ pred[v] \leftarrow e.$

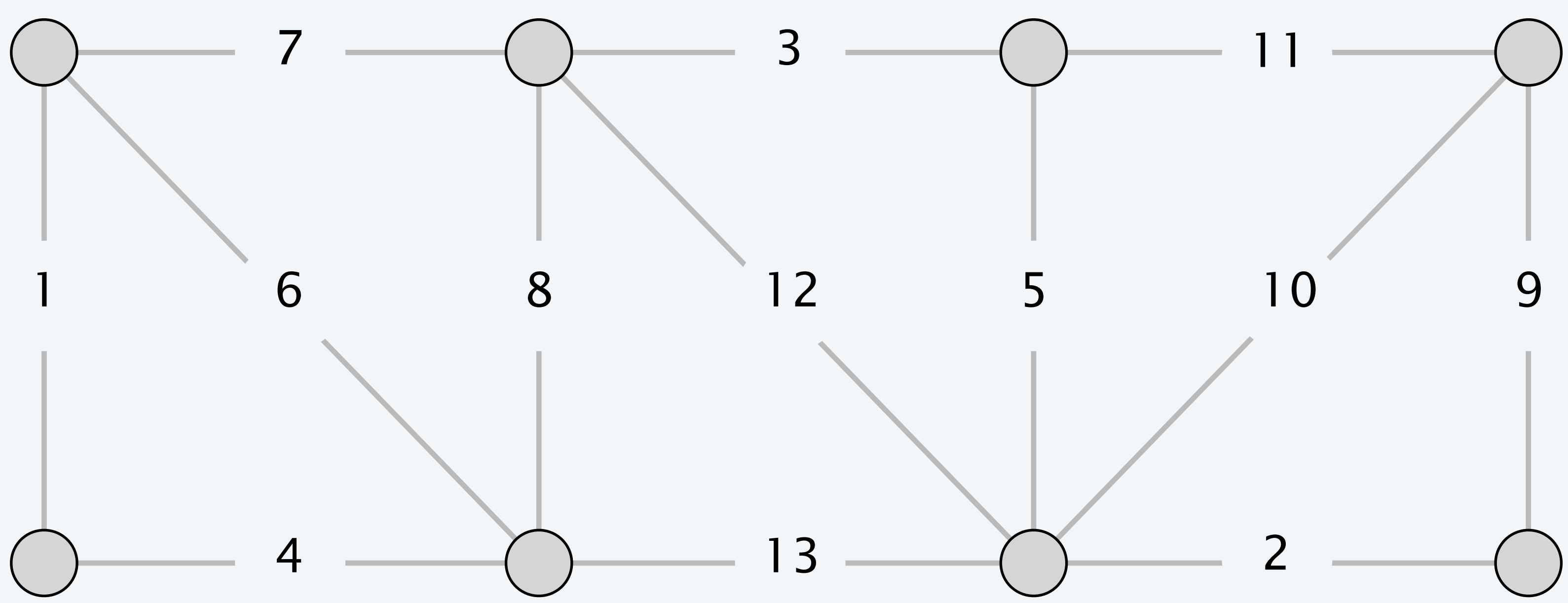
$\pi[v]$ = cost of cheapest known edge between v and S

Prim's algorithm demo

Initialize $S = \{ s \}$ for any node s, $T = \varnothing$.

Repeat $n - 1$ times:

- Add to T a min-weight edge with exactly one endpoint in S.
- Add the other endpoint to S.

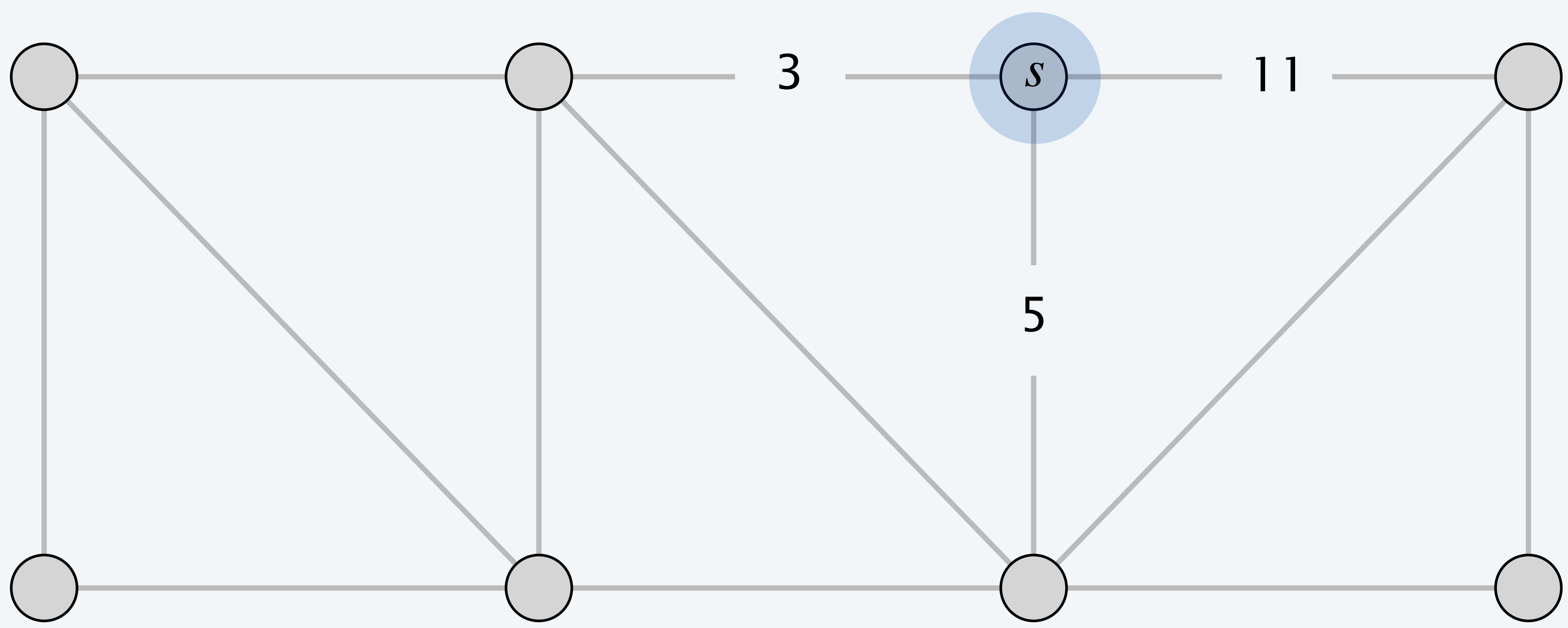


Prim's algorithm demo

Initialize $S = \{ s \}$ for any node s, $T = \varnothing$.

Repeat $n - 1$ times:

- Add to T a min-weight edge with exactly one endpoint in S.
- Add the other endpoint to S.

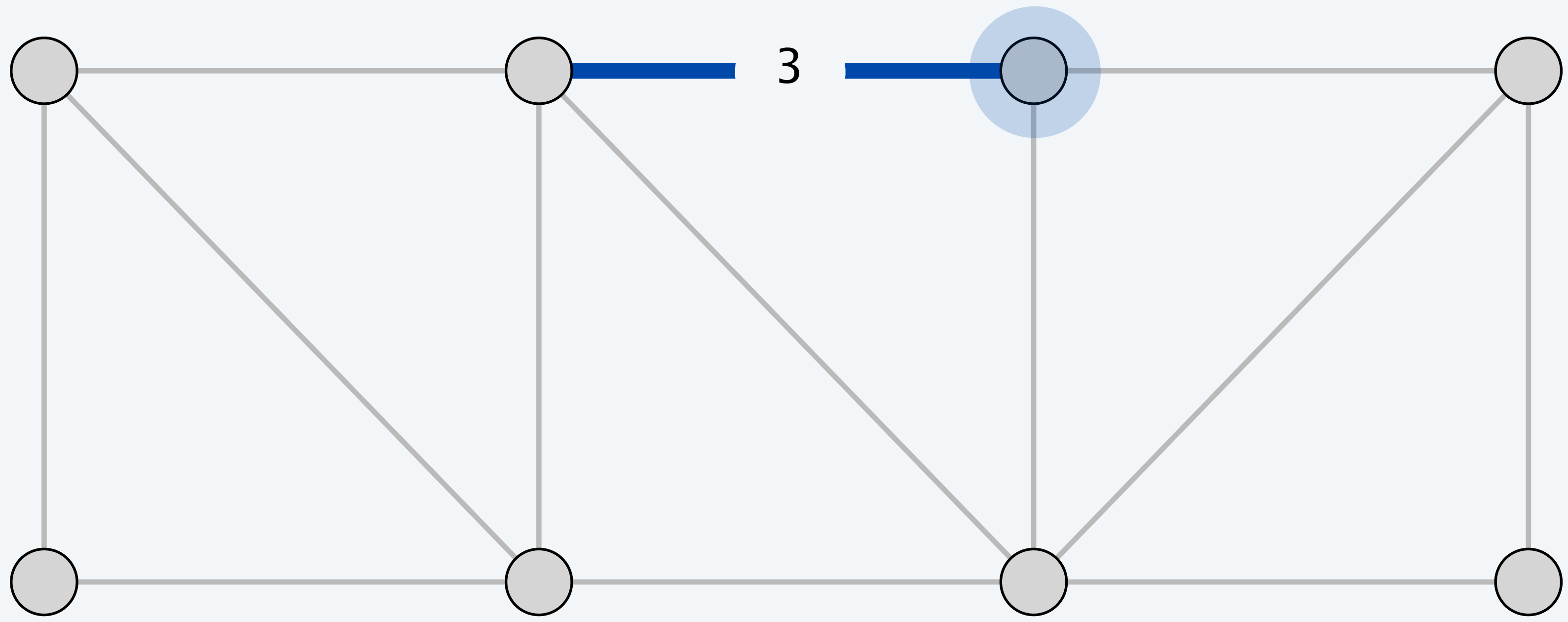


Prim's algorithm demo

Initialize $S = \{ s \}$ for any node s, $T = \varnothing$.

Repeat $n - 1$ times:

- Add to T a min-weight edge with exactly one endpoint in S.
- Add the other endpoint to S.

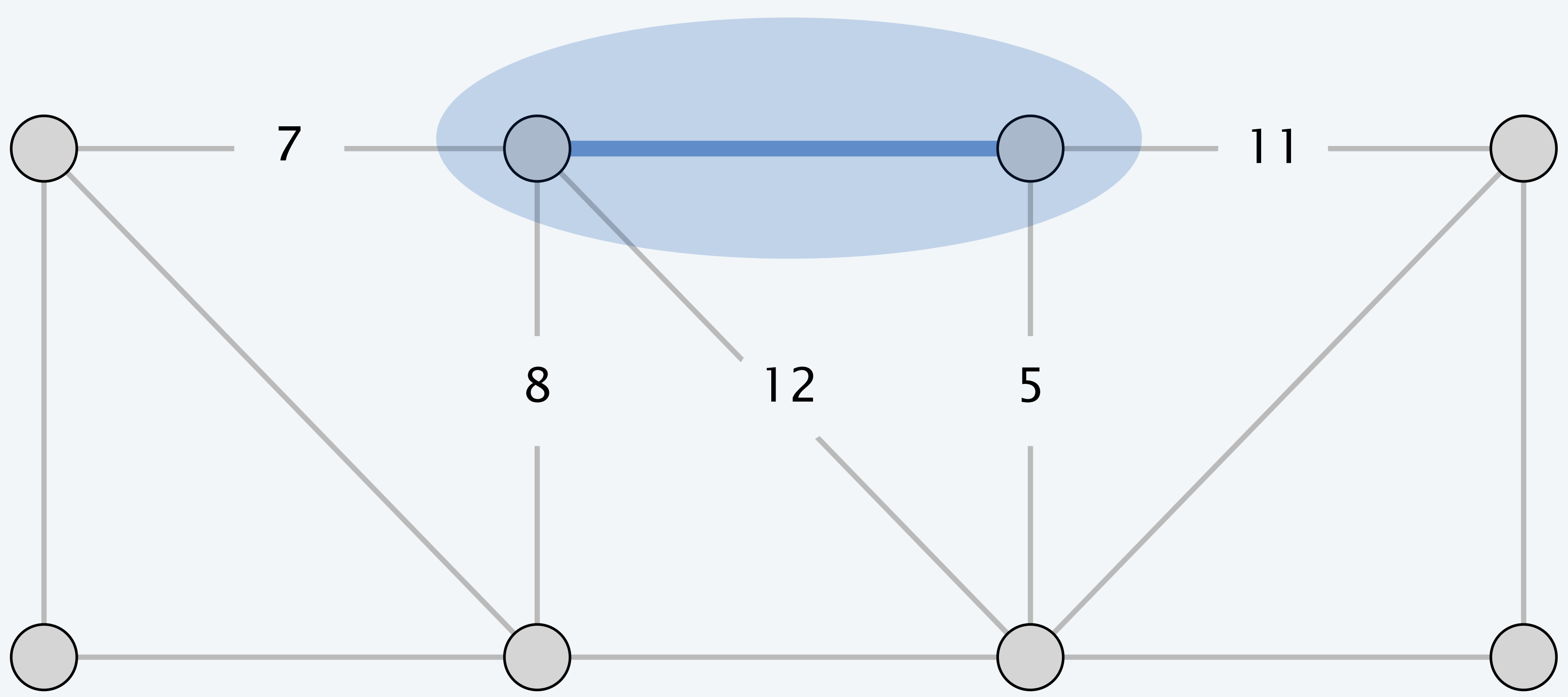


Prim's algorithm demo

Initialize $S = \{ s \}$ for any node s, $T = \varnothing$.

Repeat $n - 1$ times:

- Add to T a min-weight edge with exactly one endpoint in S.
- Add the other endpoint to S.

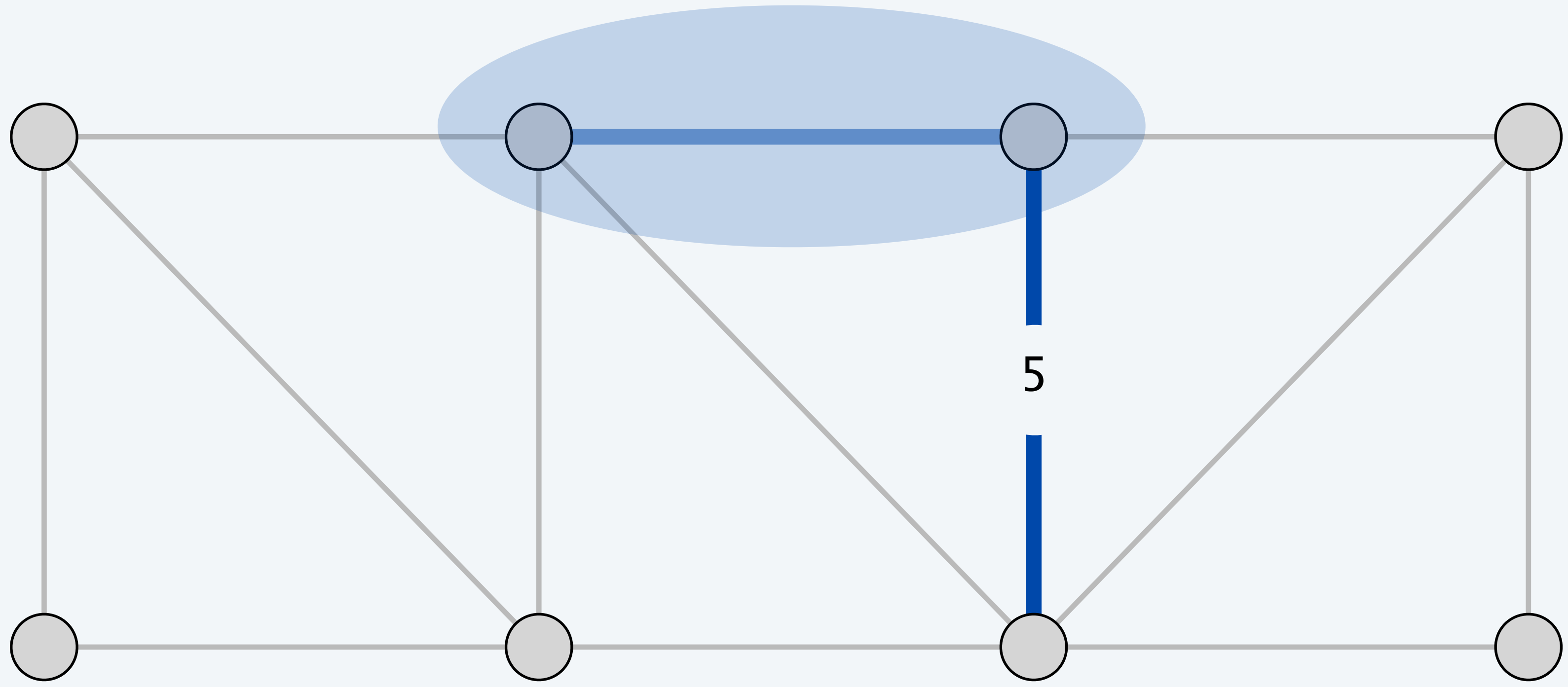


Prim's algorithm demo

Initialize $S = \{ s \}$ for any node s, $T = \varnothing$.

Repeat $n - 1$ times:

- Add to T a min-weight edge with exactly one endpoint in S.
- Add the other endpoint to S.

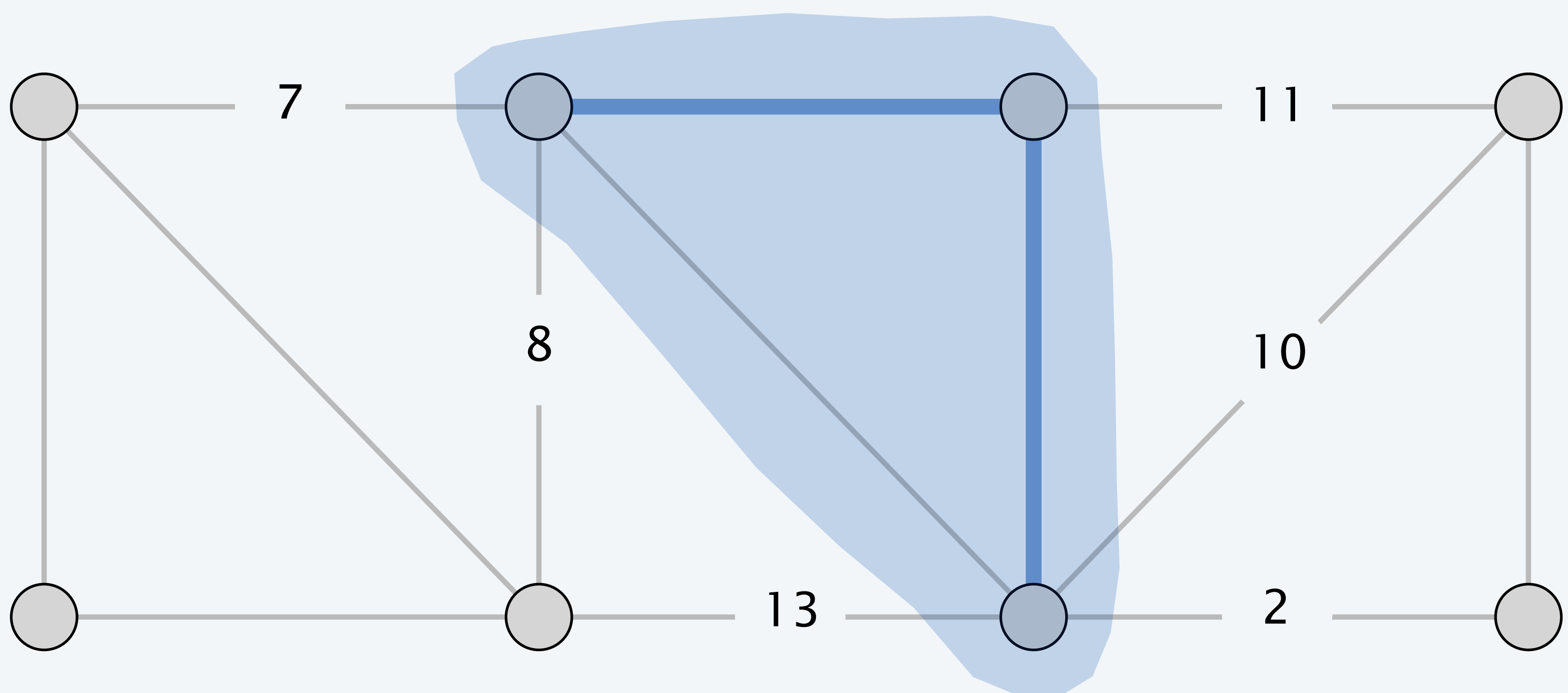


Prim's algorithm demo

Initialize $S = \{ s \}$ for any node s, $T = \varnothing$.

Repeat $n - 1$ times:

- Add to T a min-weight edge with exactly one endpoint in S.
- Add the other endpoint to S.

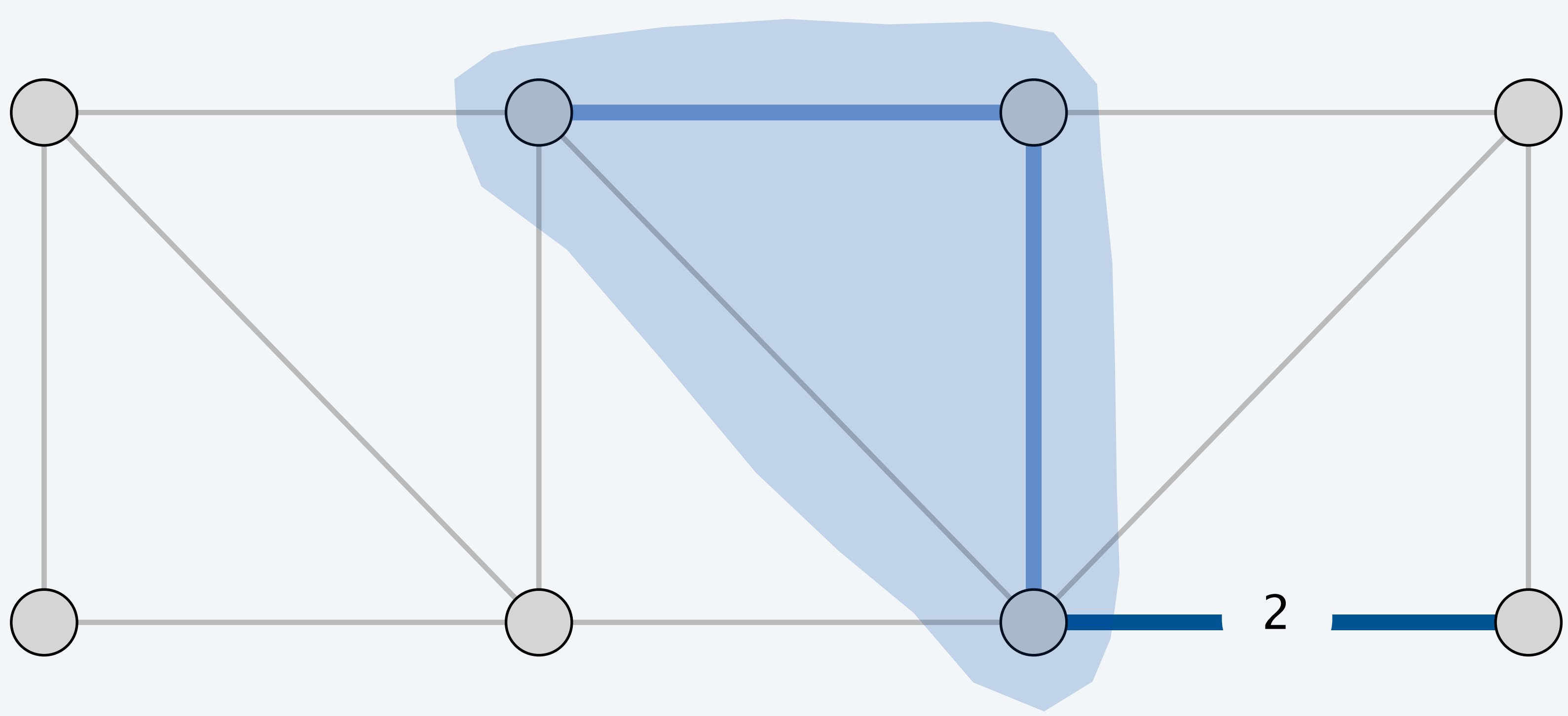


Prim's algorithm demo

Initialize $S = \{ s \}$ for any node s, $T = \varnothing$.

Repeat $n - 1$ times:

- Add to T a min-weight edge with exactly one endpoint in S.
- Add the other endpoint to S.

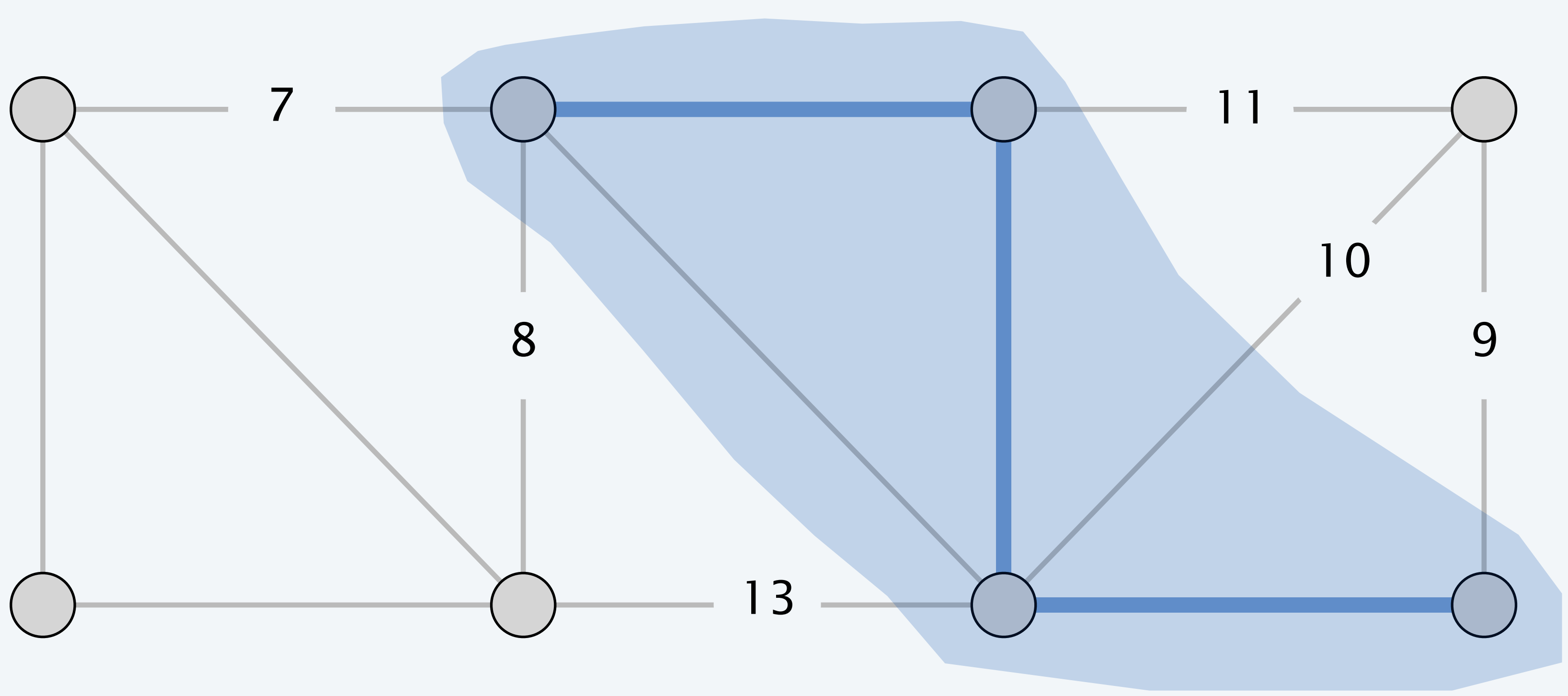


Prim's algorithm demo

Initialize $S = \{ s \}$ for any node s, $T = \varnothing$.

Repeat $n - 1$ times:

- Add to T a min-weight edge with exactly one endpoint in S.
- Add the other endpoint to S.

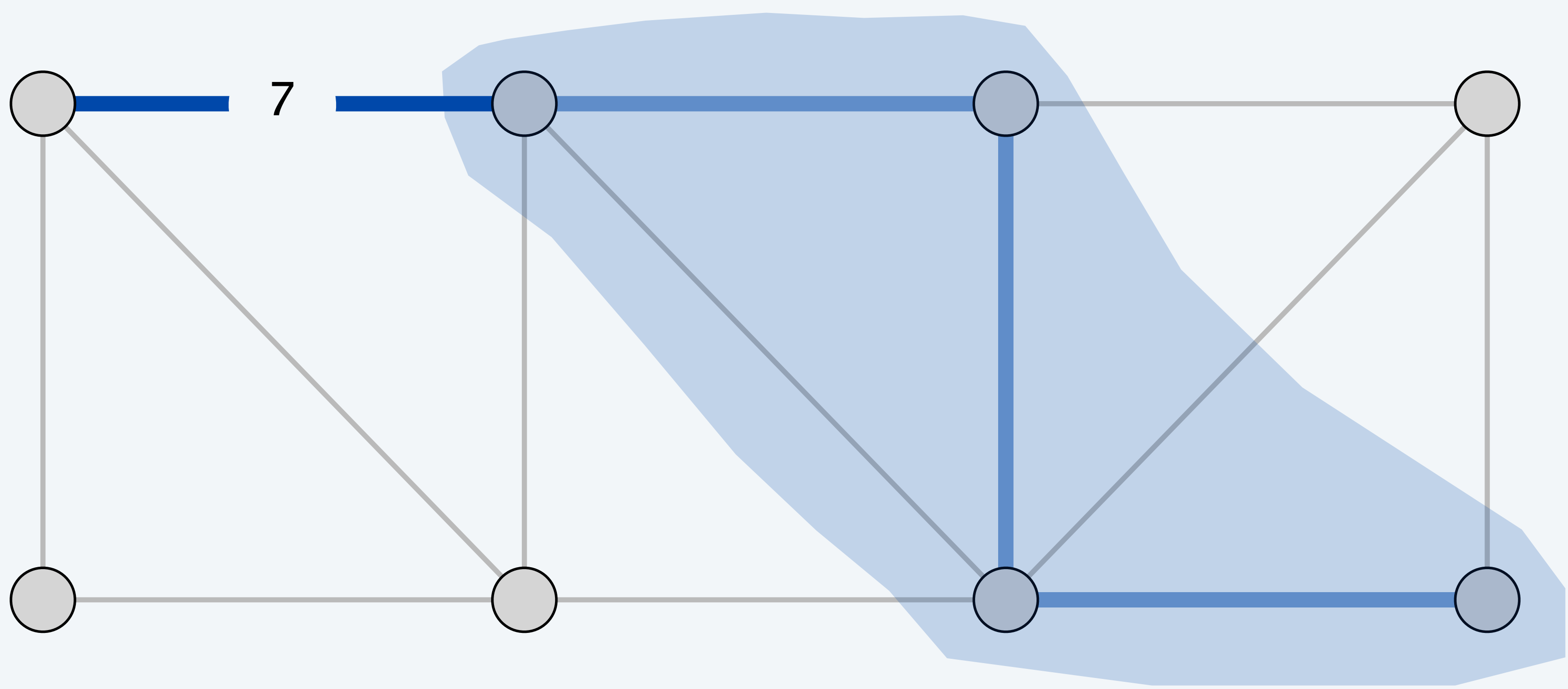


Prim's algorithm demo

Initialize $S = \{ s \}$ for any node s, $T = \varnothing$.

Repeat $n - 1$ times:

- Add to T a min-weight edge with exactly one endpoint in S.
- Add the other endpoint to S.

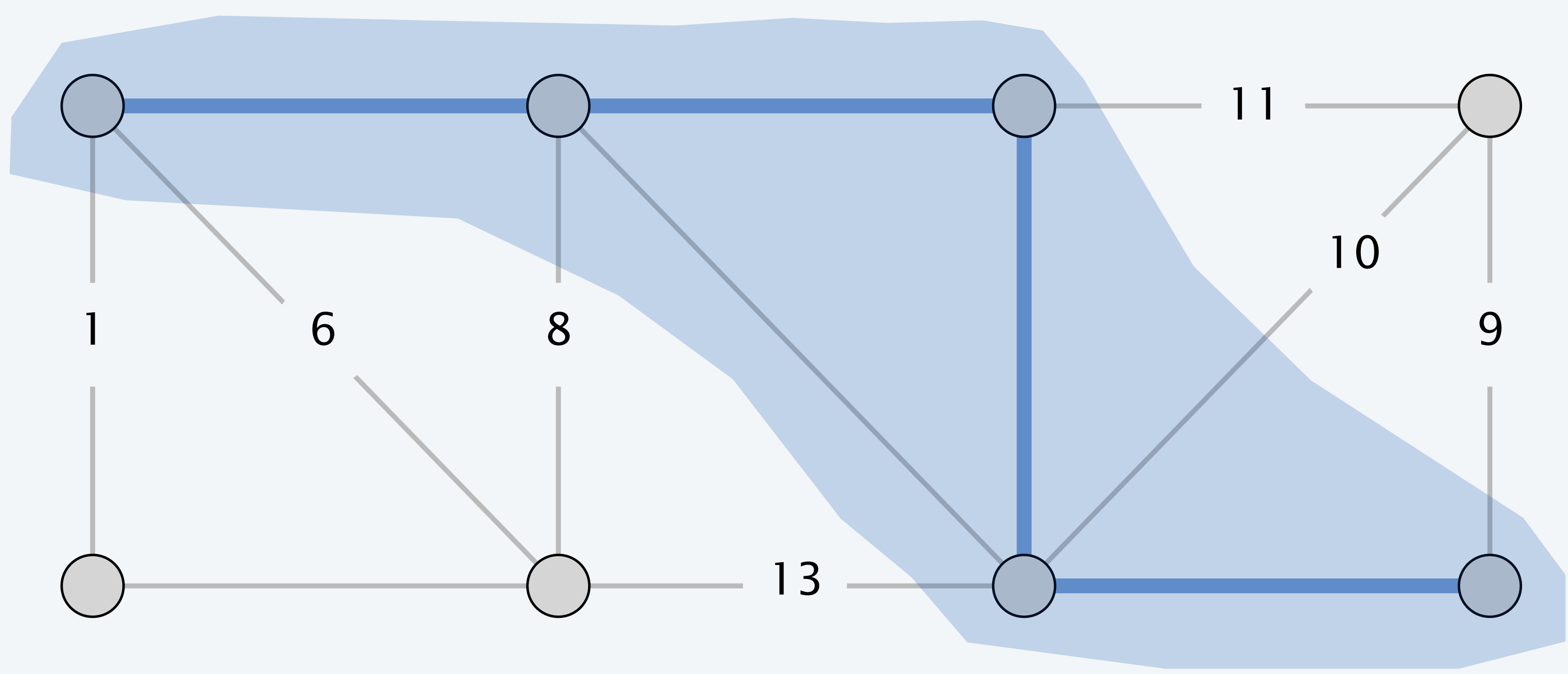


Prim's algorithm demo

Initialize $S = \{ s \}$ for any node s, $T = \varnothing$.

Repeat $n - 1$ times:

- Add to T a min-weight edge with exactly one endpoint in S.
- Add the other endpoint to S.

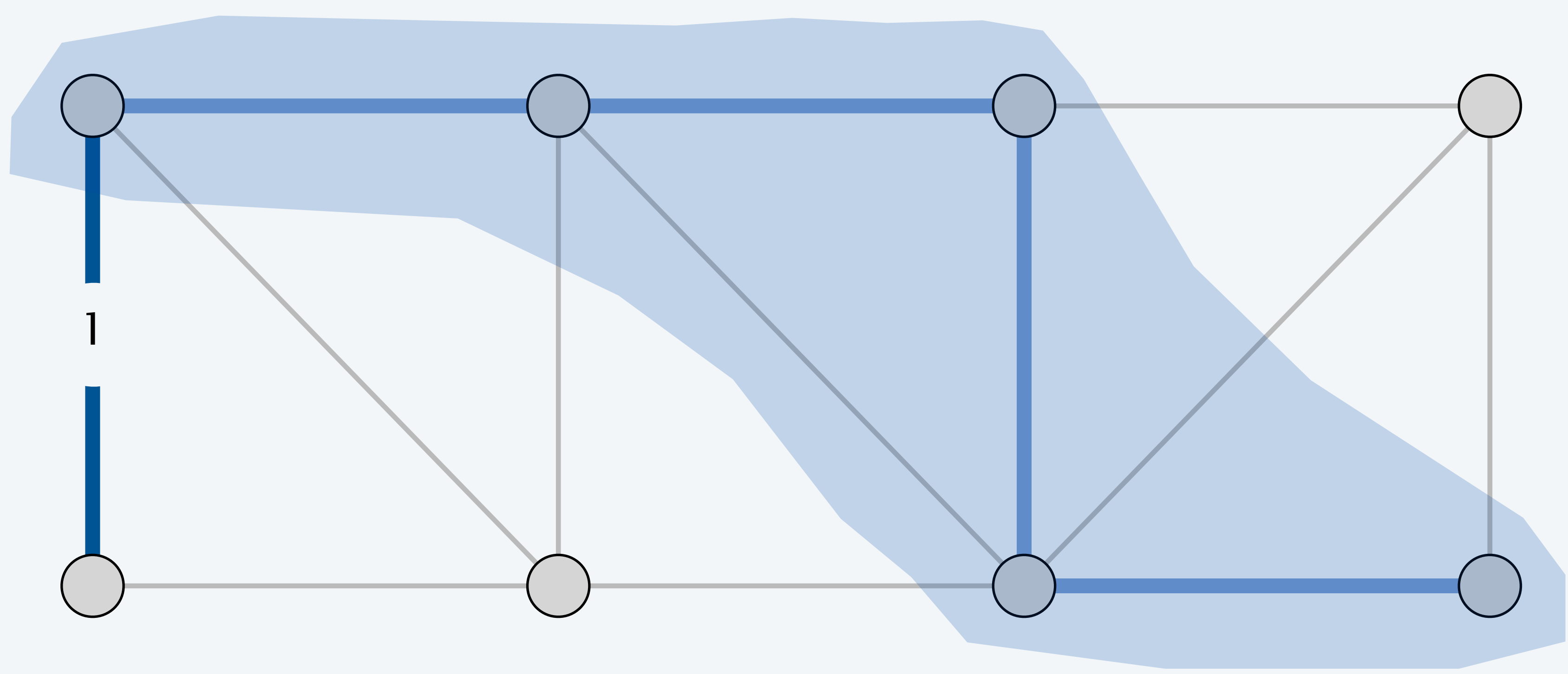


Prim's algorithm demo

Initialize $S = \{ s \}$ for any node s, $T = \varnothing$.

Repeat $n - 1$ times:

- Add to T a min-weight edge with exactly one endpoint in S.
- Add the other endpoint to S.

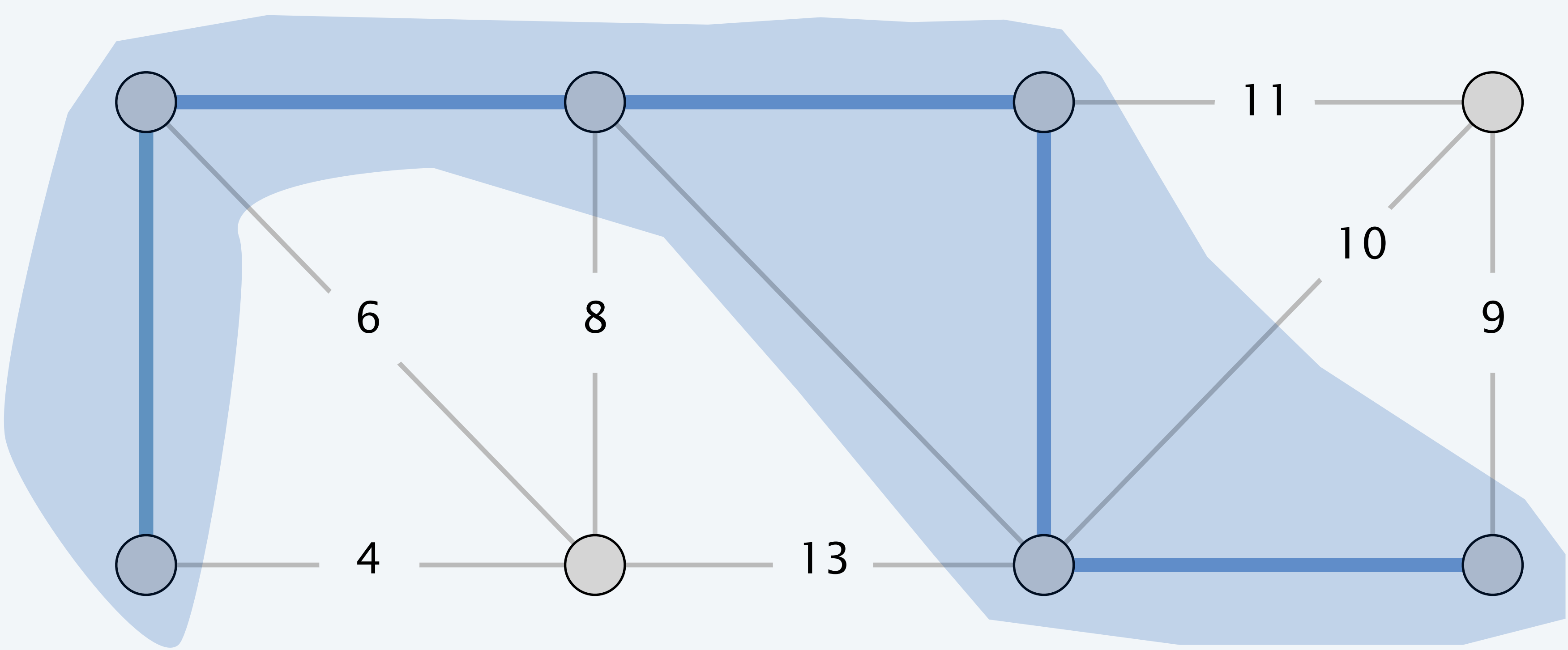


Prim's algorithm demo

Initialize $S = \{ s \}$ for any node s, $T = \varnothing$.

Repeat $n - 1$ times:

- Add to T a min-weight edge with exactly one endpoint in S.
- Add the other endpoint to S.

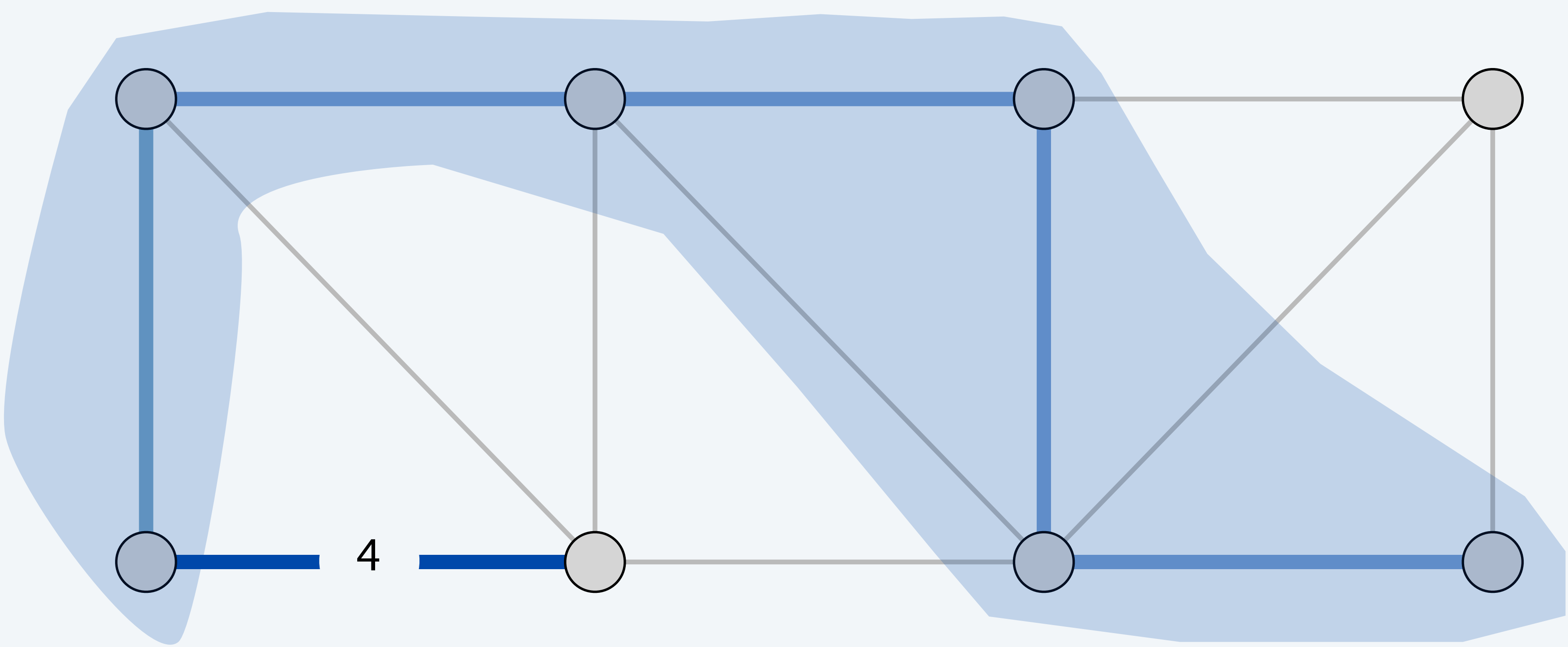


Prim's algorithm demo

Initialize $S = \{ s \}$ for any node s, $T = \varnothing$.

Repeat $n - 1$ times:

- Add to T a min-weight edge with exactly one endpoint in S.
- Add the other endpoint to S.

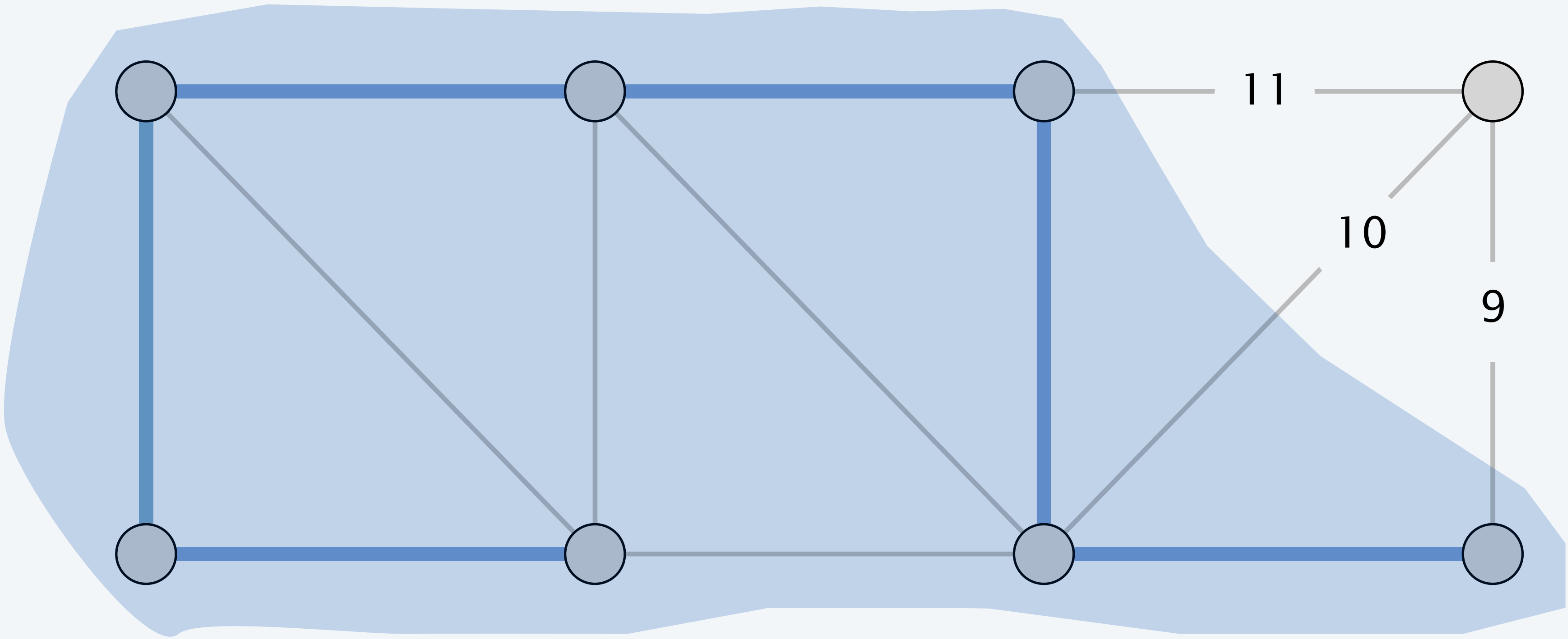


Prim's algorithm demo

Initialize $S = \{ s \}$ for any node s, $T = \varnothing$.

Repeat $n - 1$ times:

- Add to T a min-weight edge with exactly one endpoint in S.
- Add the other endpoint to S.

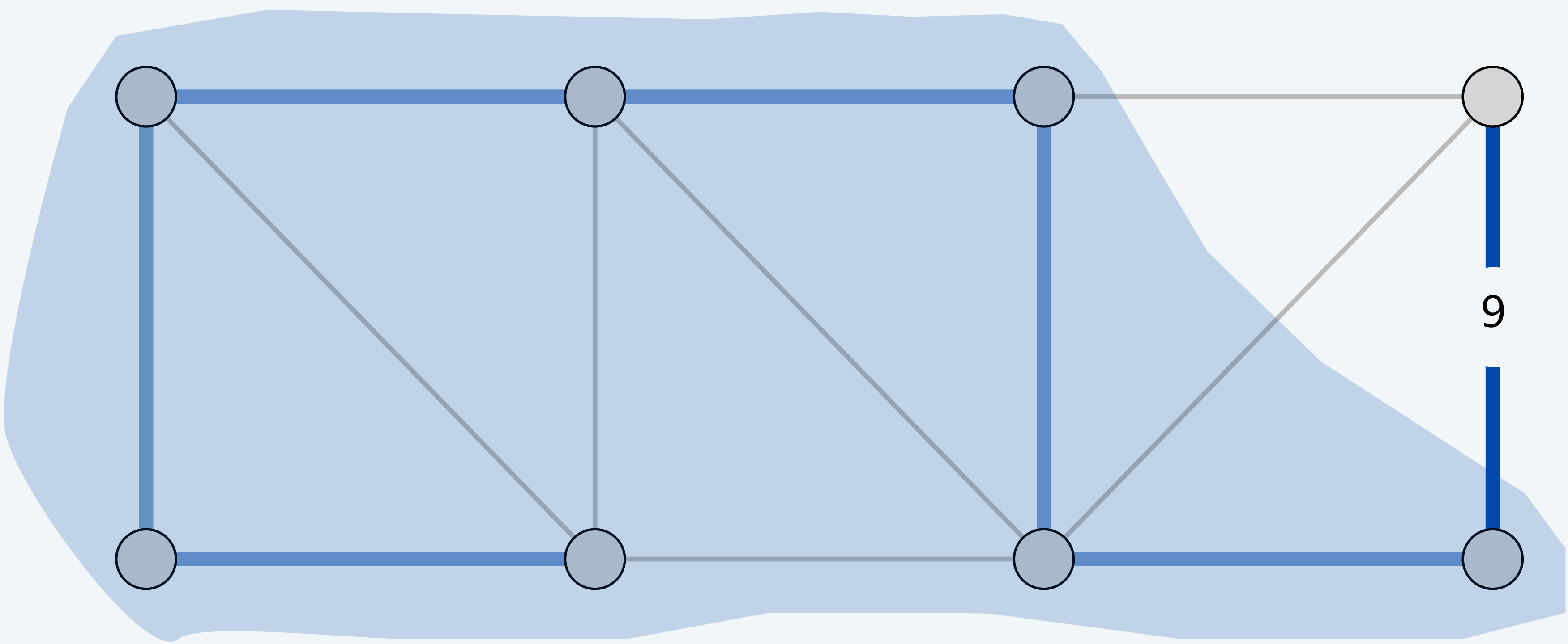


Prim's algorithm demo

Initialize $S = \{ s \}$ for any node s, $T = \varnothing$.

Repeat $n - 1$ times:

- Add to T a min-weight edge with exactly one endpoint in S.
- Add the other endpoint to S.

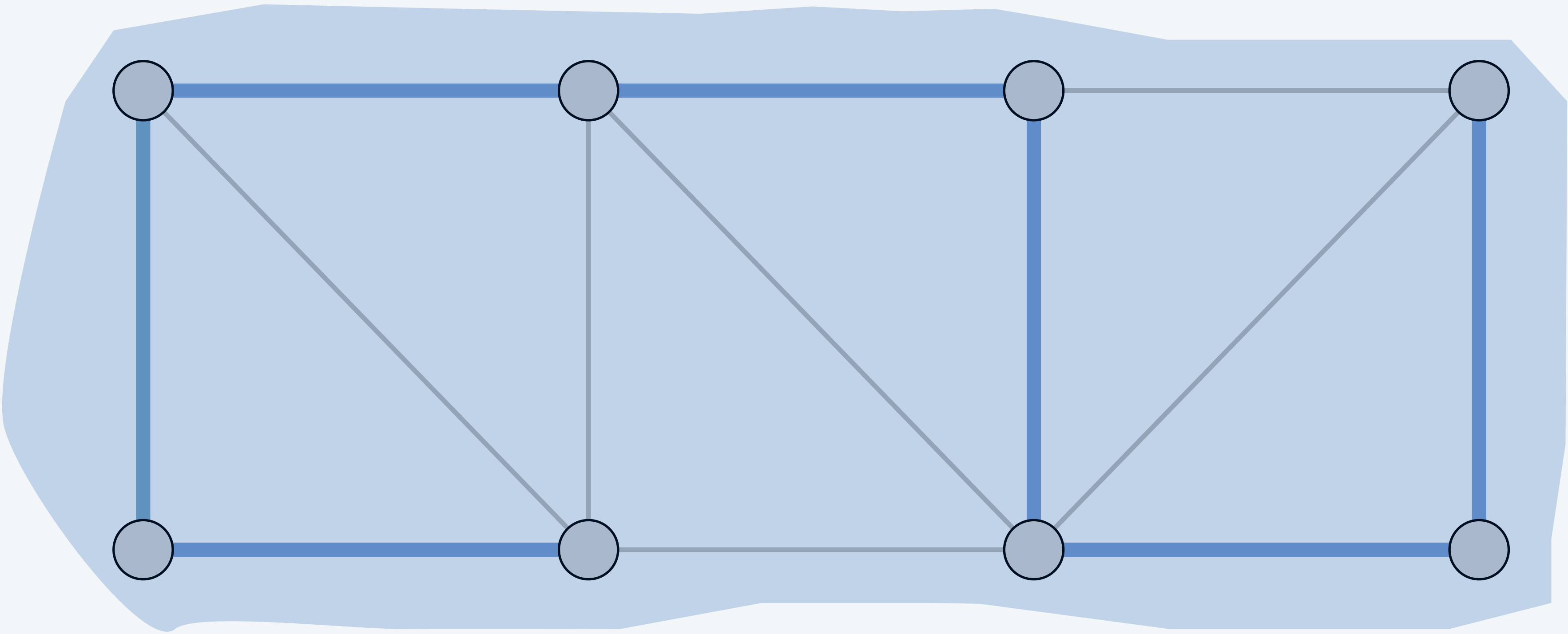


Prim's algorithm demo

Initialize $S = \{ s \}$ for any node s, $T = \varnothing$.

Repeat $n - 1$ times:

- Add to T a min-weight edge with exactly one endpoint in S.
- Add the other endpoint to S.

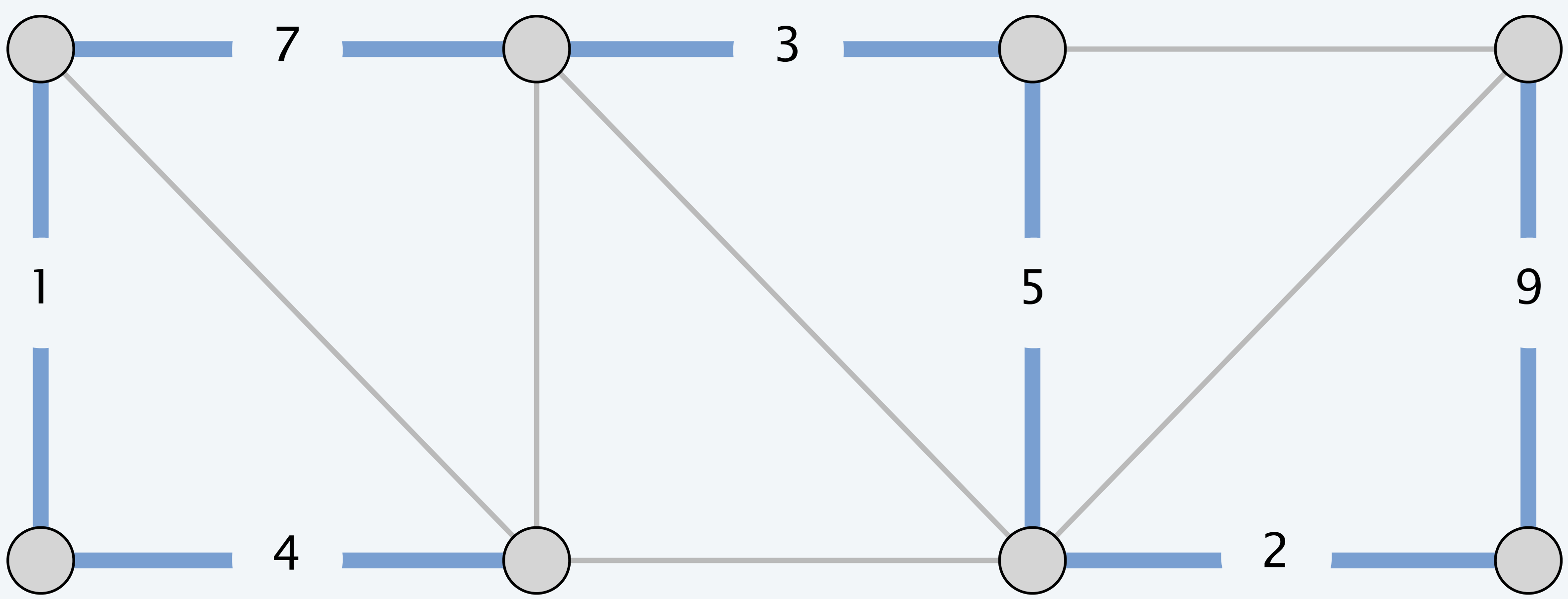


Prim's algorithm demo

Initialize $S = \{ s \}$ for any node s, $T = \varnothing$.

Repeat $n - 1$ times:

- Add to T a min-weight edge with exactly one endpoint in S.
- Add the other endpoint to S.



Kruskal's algorithm

Consider edges in ascending order of cost:

- Add to tree unless it would create a cycle.

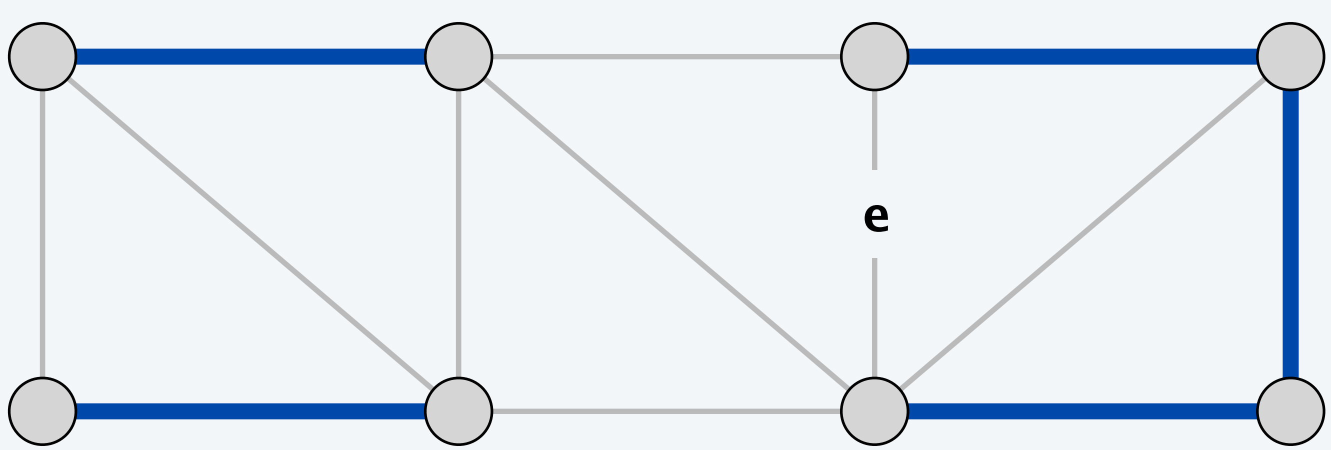
Theorem. Kruskal's algorithm computes an MST.

Pf. Special case of greedy algorithm.

- Case 1: both endpoints of e in same blue tree.
  $\Rightarrow$ color e red by applying red rule to unique cycle.
- Case 2: both endpoints of e in different blue trees.
  $\Rightarrow$ color e blue by applying blue rule to cutset defined by either tree. ▪

all other edges in cycle are blue

no edge in cutset has smaller cost
(since Kruskal chose it first)



Kruskal's algorithm: implementation

Theorem. Kruskal's algorithm can be implemented to run in $O(m \log m)$ time.

- Sort edges by cost.
- Use union–find data structure to dynamically maintain connected components.

KRUSKAL (V, E, c)

SORT m edges by cost and renumber so that $c(e_1) \le c(e_2) \le \ldots \le c(e_m)$.

$T \leftarrow \varnothing$.

FOREACH $v \in V$: MAKE-SET(v).

FOR $i = 1$ TO m

  $(u, v) \leftarrow e_i$.

  IF (FIND-SET(u) $\neq$ FIND-SET(v)) ← are u and v in same component?

    $T \leftarrow T \cup \{ e_i \}$.

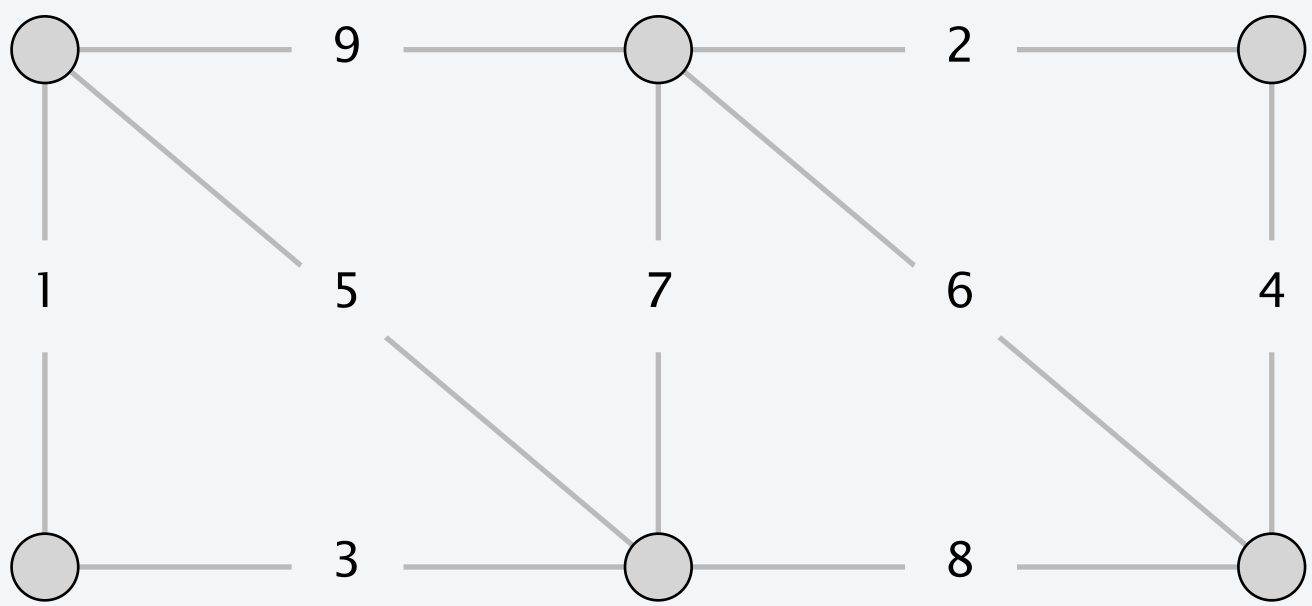
    UNION(u, v). ← make u and v in same component

RETURN T.

Kruskal's algorithm demo

Consider edges in ascending order of weight:

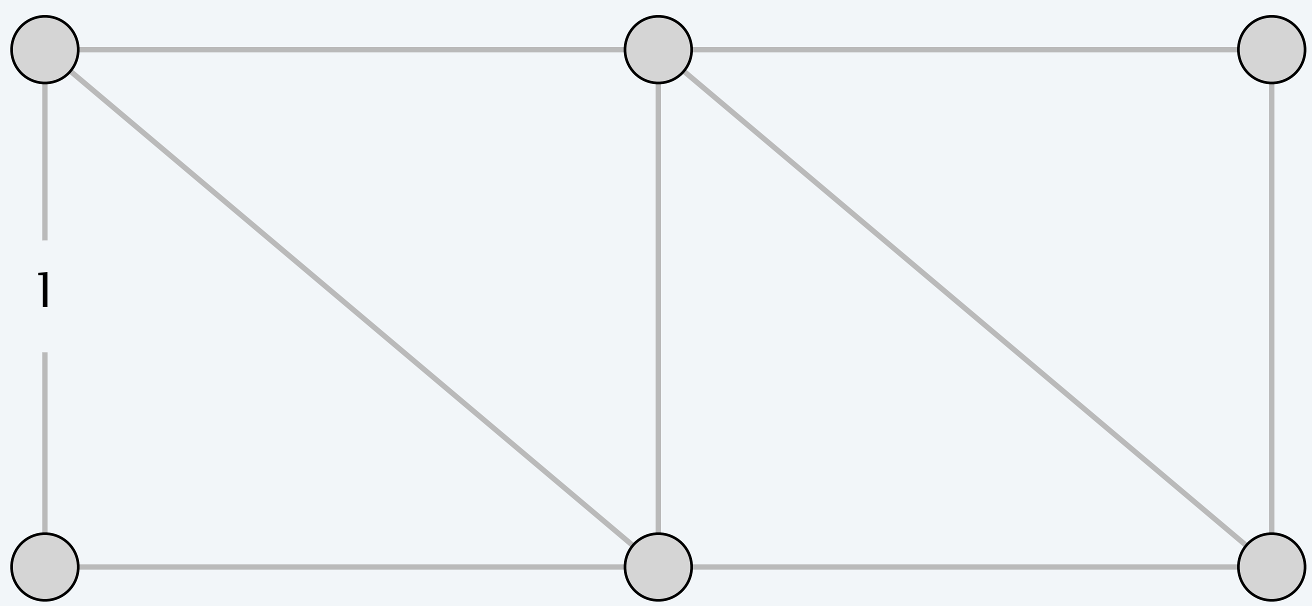
- Add to T unless it would create a cycle.



Kruskal's algorithm demo

Consider edges in ascending order of weight:

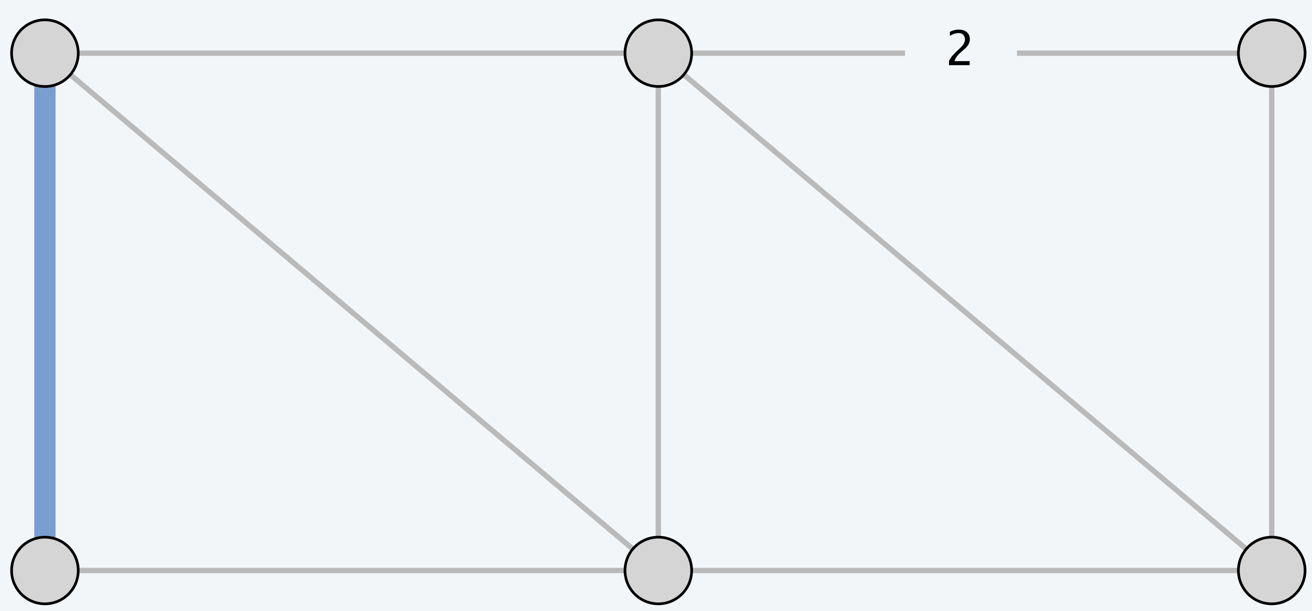
- Add to T unless it would create a cycle.



Kruskal's algorithm demo

Consider edges in ascending order of weight:

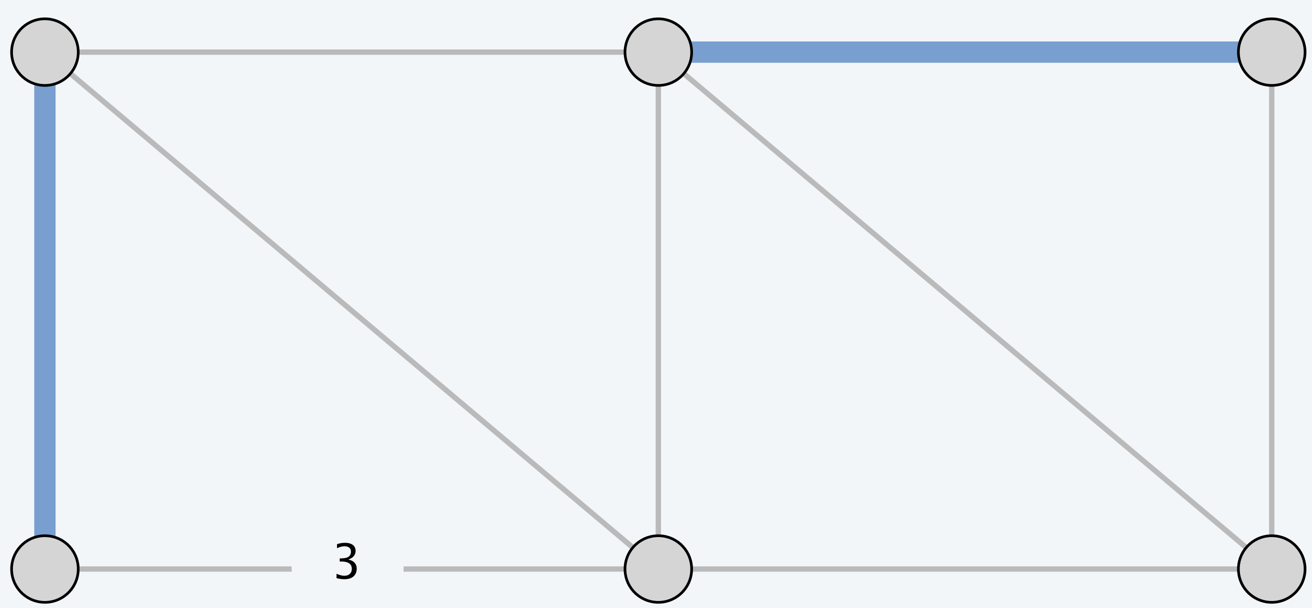
- Add to T unless it would create a cycle.



Kruskal's algorithm demo

Consider edges in ascending order of weight:

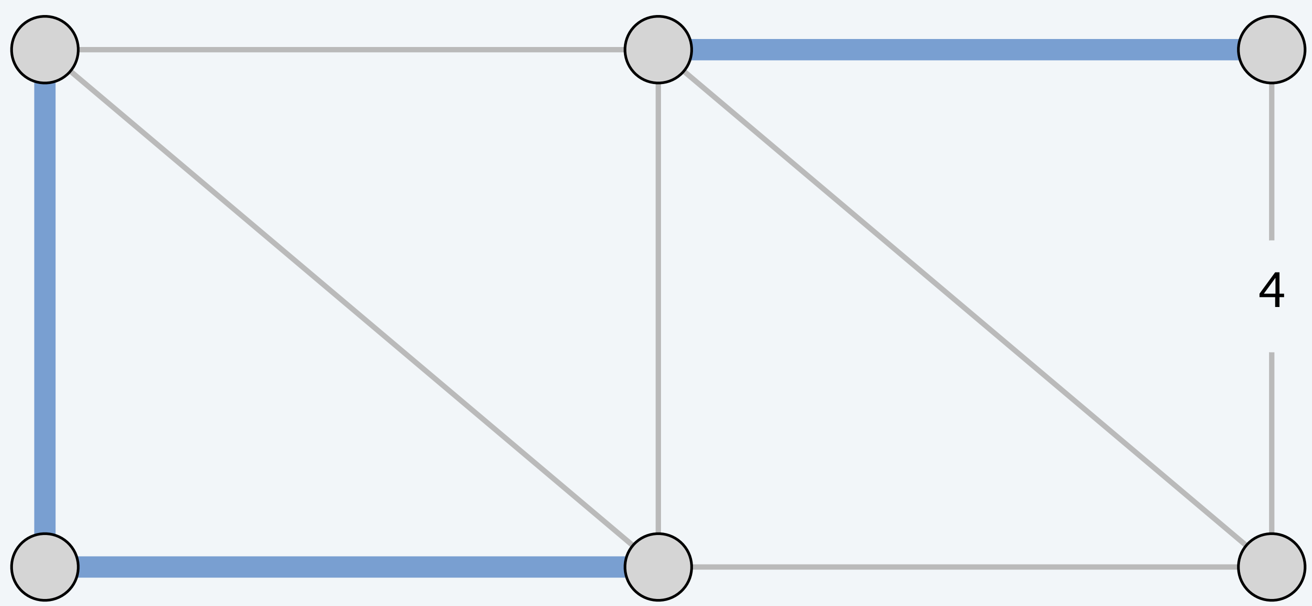
- Add to T unless it would create a cycle.



Kruskal's algorithm demo

Consider edges in ascending order of weight:

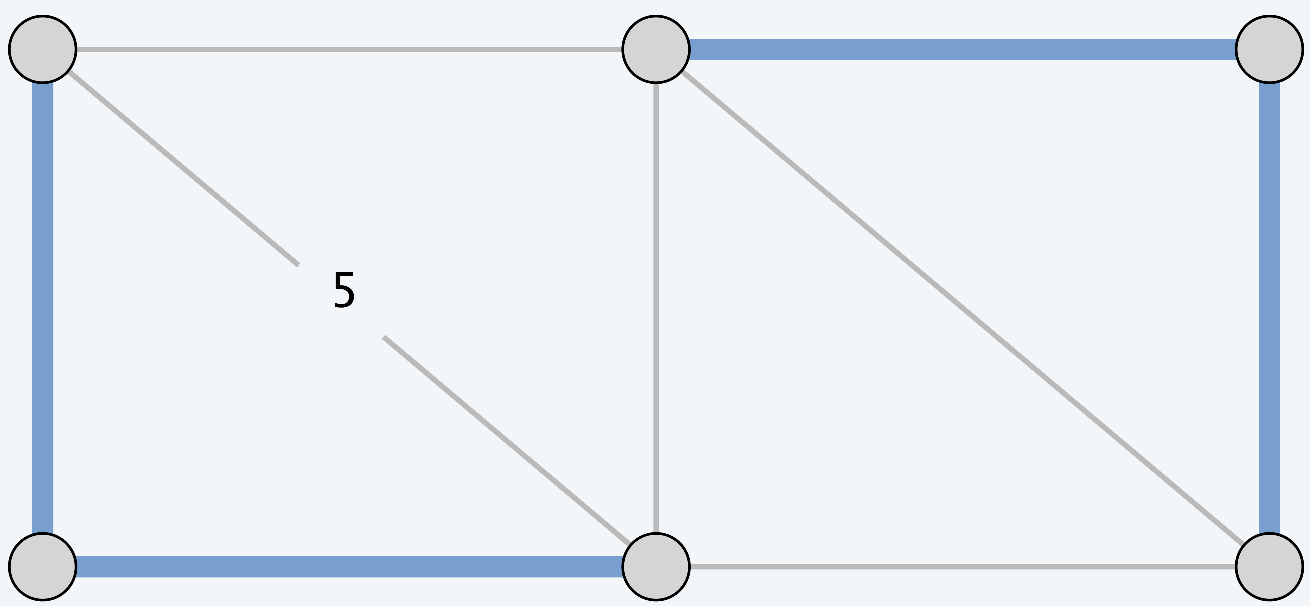
- Add to T unless it would create a cycle.



Kruskal's algorithm demo

Consider edges in ascending order of weight:

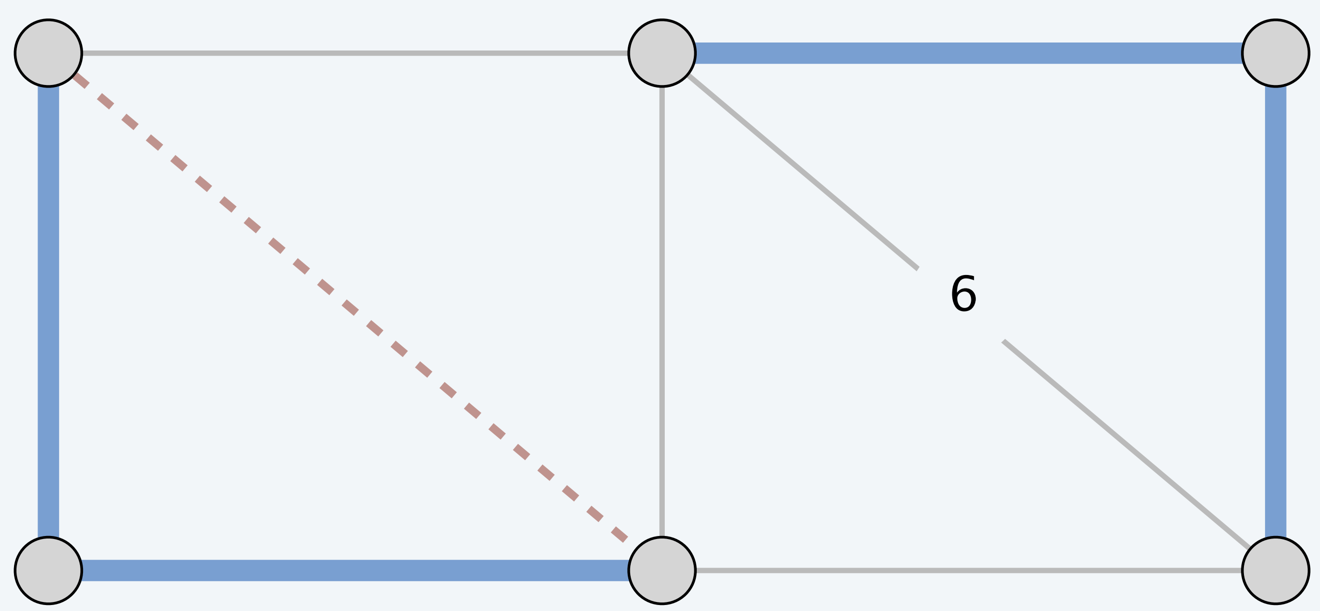
- Add to T unless it would create a cycle.



Kruskal's algorithm demo

Consider edges in ascending order of weight:

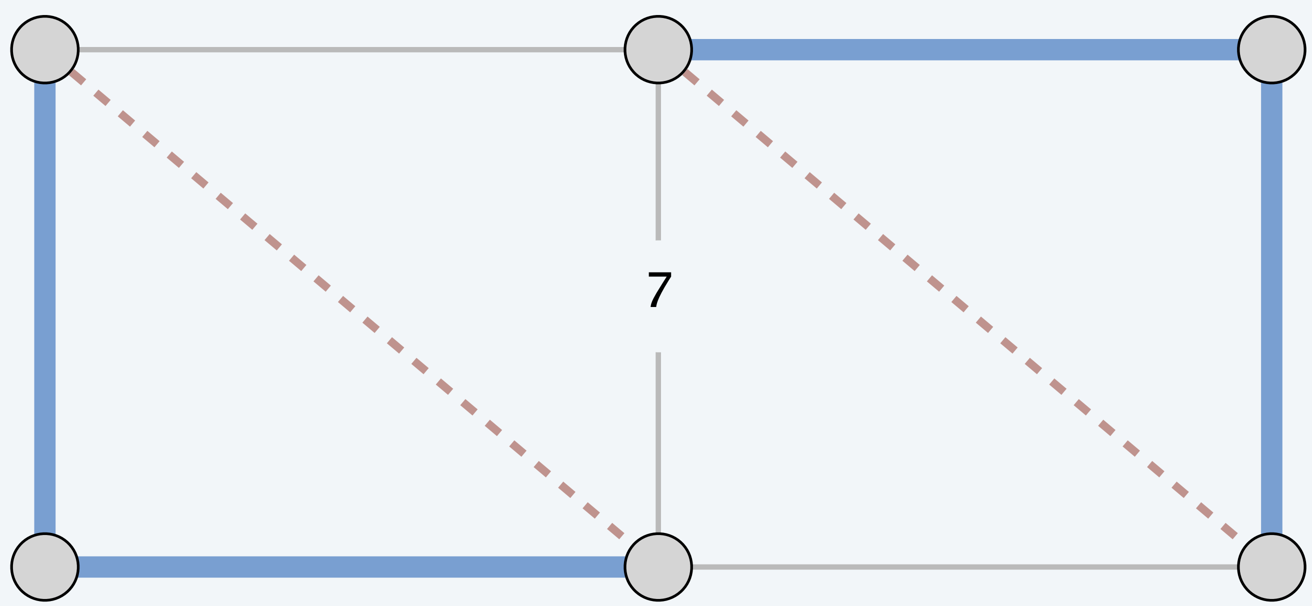
- Add to T unless it would create a cycle.



Kruskal's algorithm demo

Consider edges in ascending order of weight:

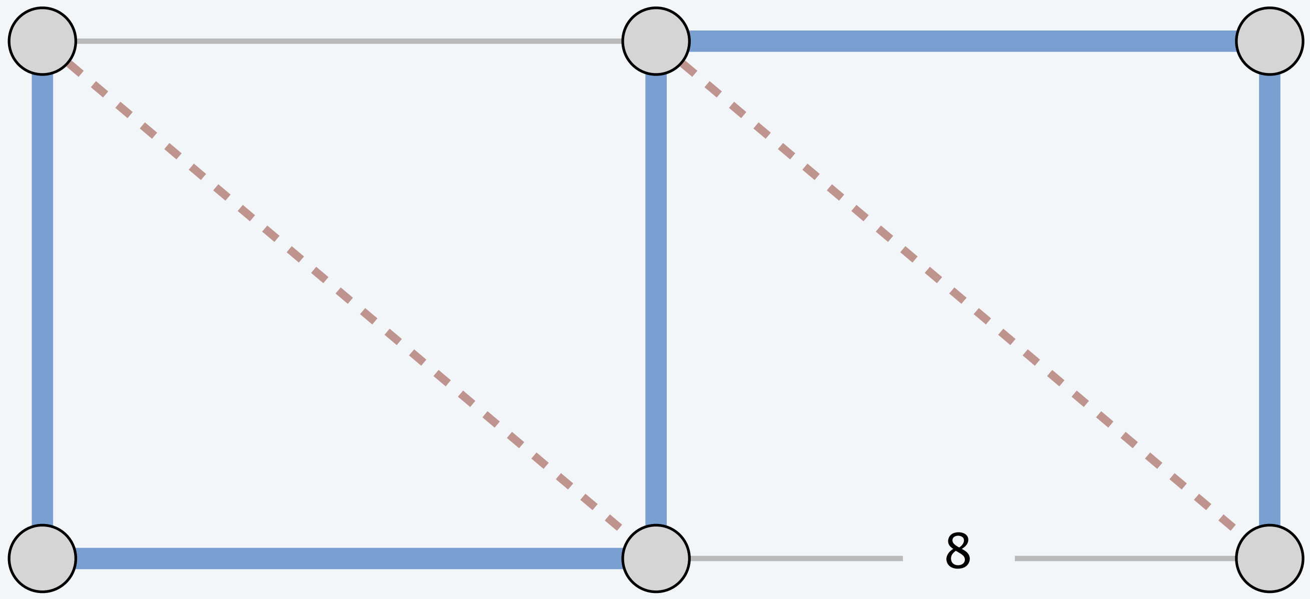
- Add to T unless it would create a cycle.



Kruskal's algorithm demo

Consider edges in ascending order of weight:

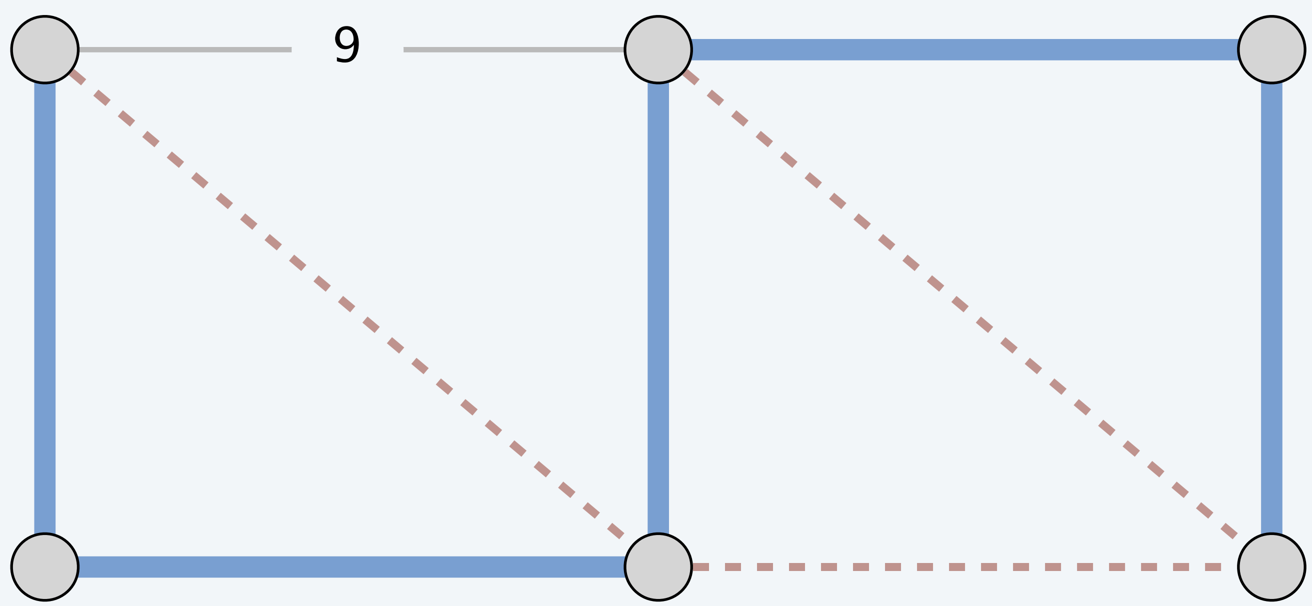
- Add to T unless it would create a cycle.



Kruskal's algorithm demo

Consider edges in ascending order of weight:

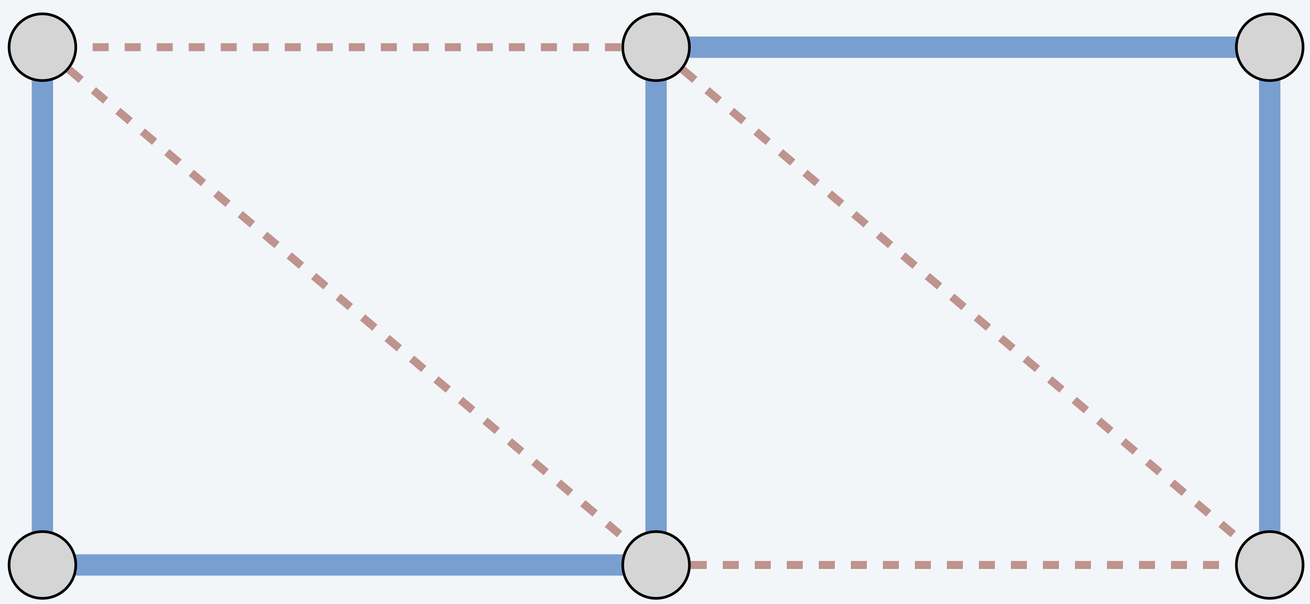
- Add to T unless it would create a cycle.



Kruskal's algorithm demo

Consider edges in ascending order of weight:

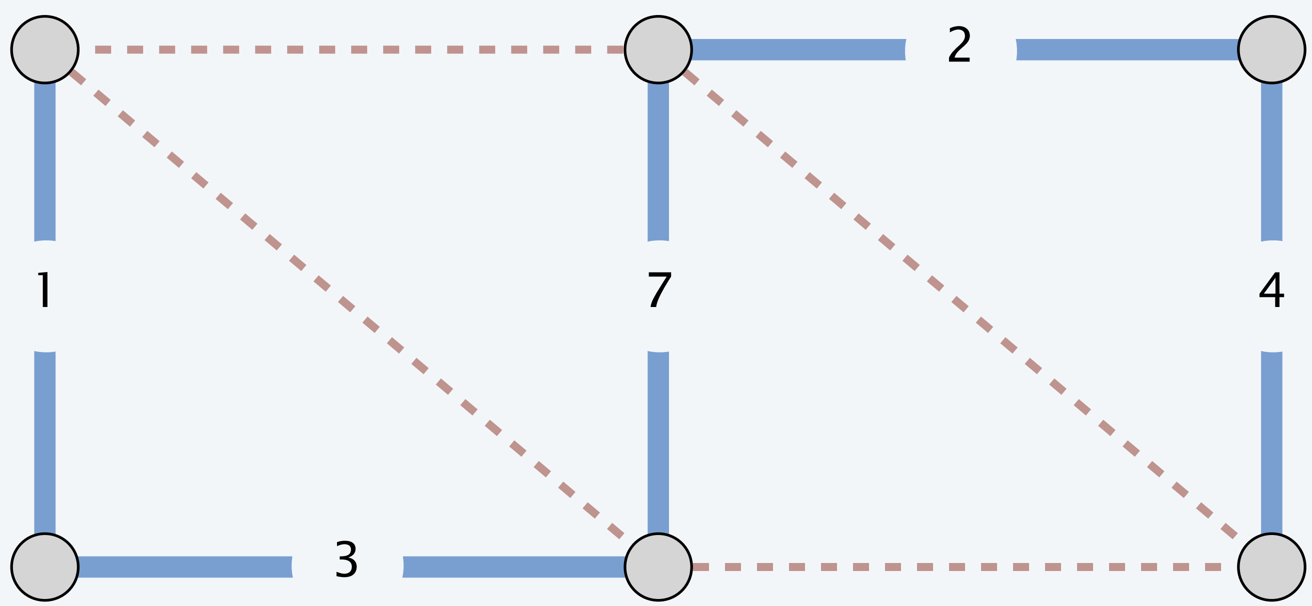
- Add to T unless it would create a cycle.



Kruskal's algorithm demo

Consider edges in ascending order of weight:

- Add to T unless it would create a cycle.



Reverse-delete algorithm

Start with all edges in T and consider them in descending order of cost:

- Delete edge from T unless it would disconnect T.

Theorem. The reverse-delete algorithm computes an MST.

Pf. Special case of greedy algorithm.

- Case 1. [deleting edge e does not disconnect T]
  $\Rightarrow$ apply red rule to cycle C formed by adding e to another path in T between its two endpoints

  no edge in C is more expensive
  (it would have already been considered and deleted)

- Case 2. [deleting edge e disconnects T]
  $\Rightarrow$ apply blue rule to cutset D induced by either component ▪

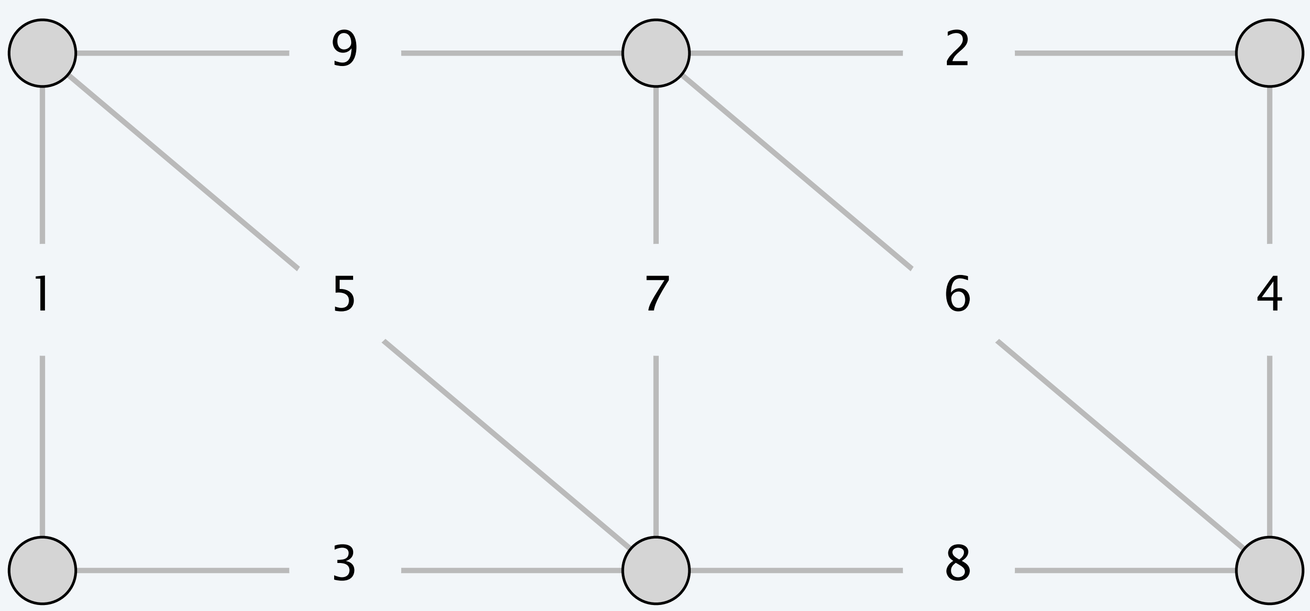
  e is the only remaining edge in the cutset
  (all other edges in D must have been colored red / deleted)

Fact. [Thorup 2000] Can be implemented to run in $O(m \log n (\log \log n)^3)$ time.

Reverse-delete algorithm demo

Start with all edges in T and consider them in descending order of weight:

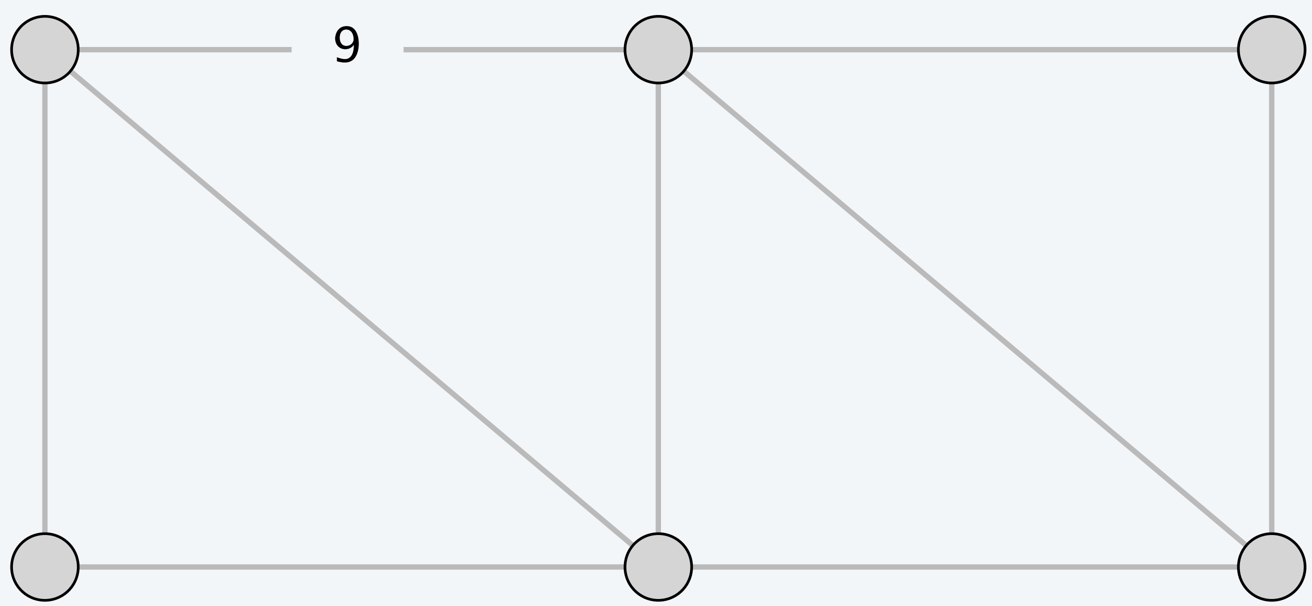
- Delete edge from T unless it would disconnect T.



Reverse-delete algorithm

Start with all edges in T and consider them in descending order of weight:

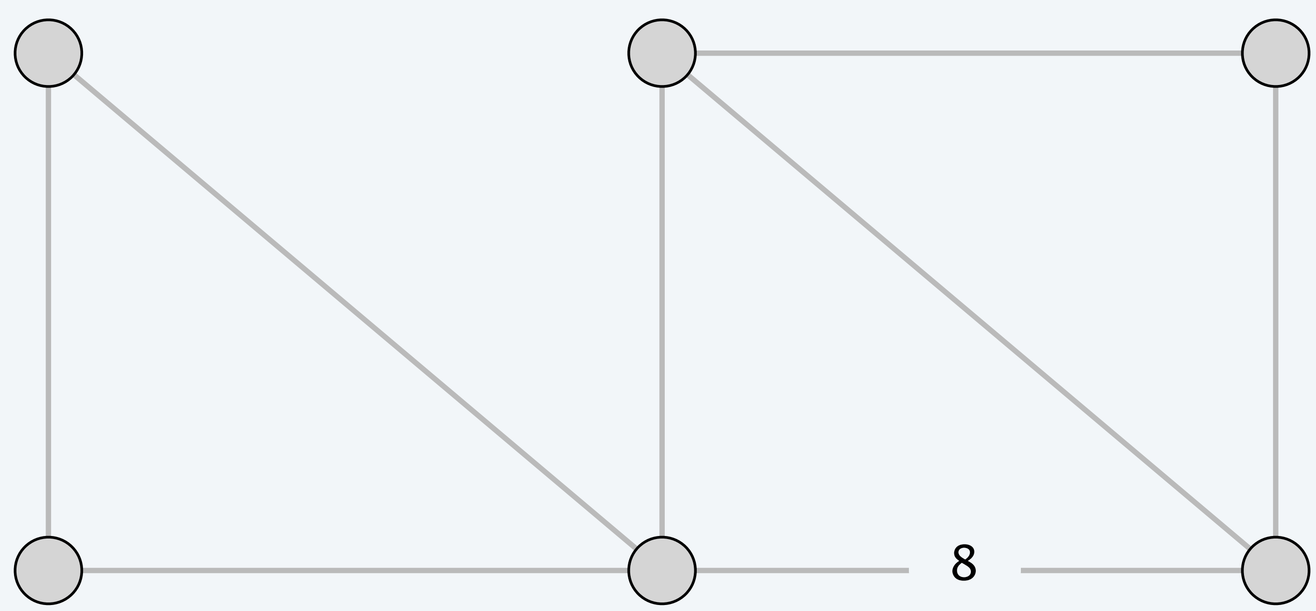
- Delete edge from T unless it would disconnect T.



Reverse-delete algorithm

Start with all edges in T and consider them in descending order of weight:

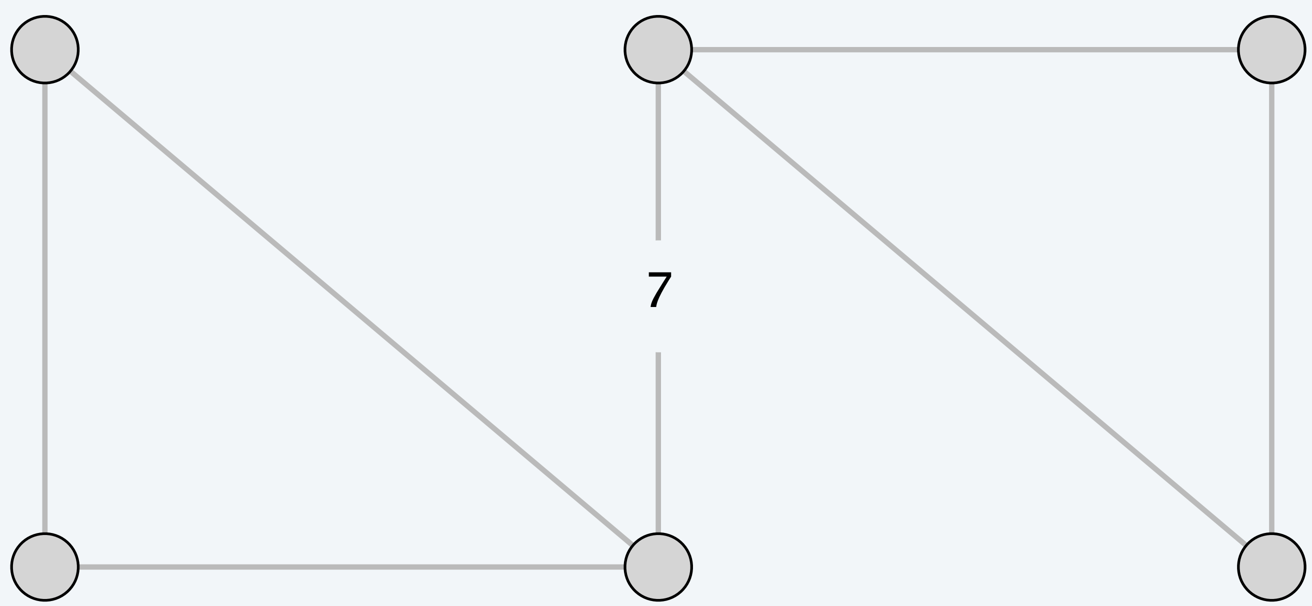
- Delete edge from T unless it would disconnect T.



Reverse-delete algorithm

Start with all edges in T and consider them in descending order of weight:

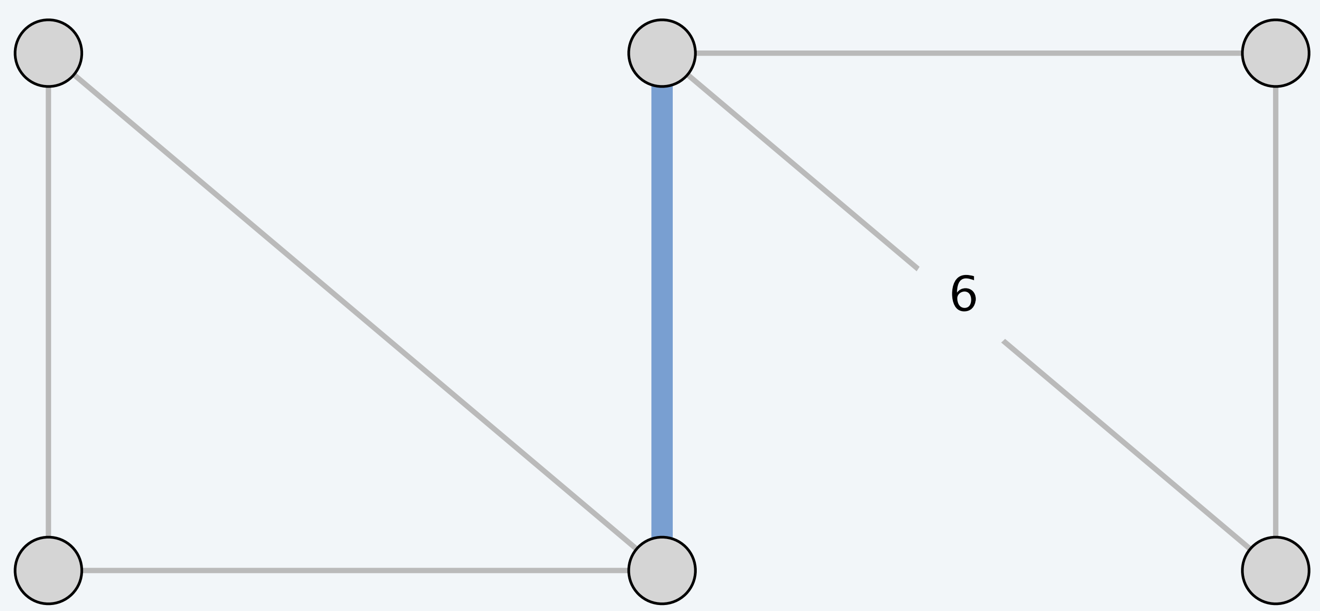
- Delete edge from T unless it would disconnect T.



Reverse-delete algorithm

Start with all edges in T and consider them in descending order of weight:

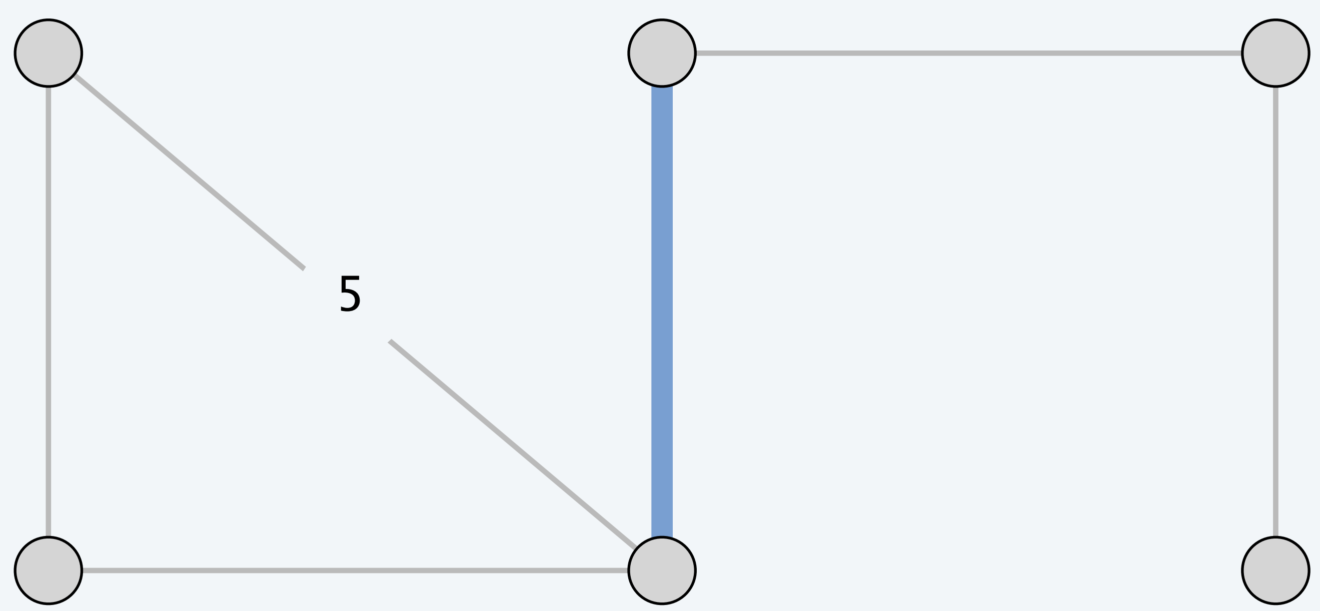
- Delete edge from T unless it would disconnect T.



Reverse-delete algorithm

Start with all edges in T and consider them in descending order of weight:

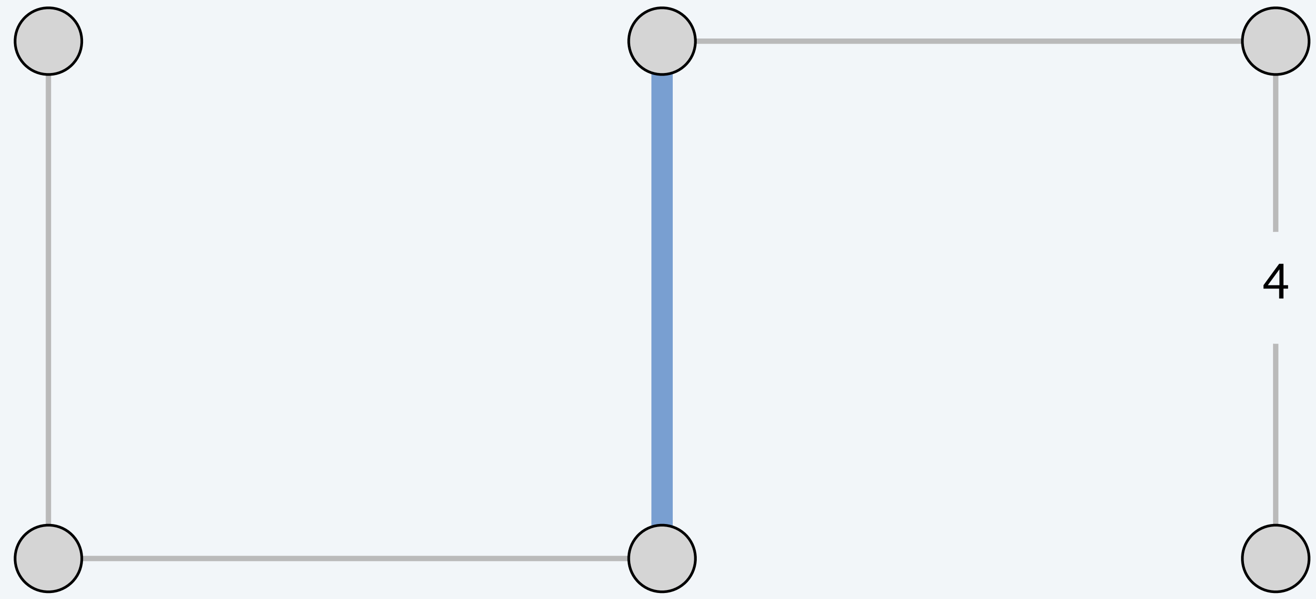
- Delete edge from T unless it would disconnect T.



Reverse-delete algorithm

Start with all edges in T and consider them in descending order of weight:

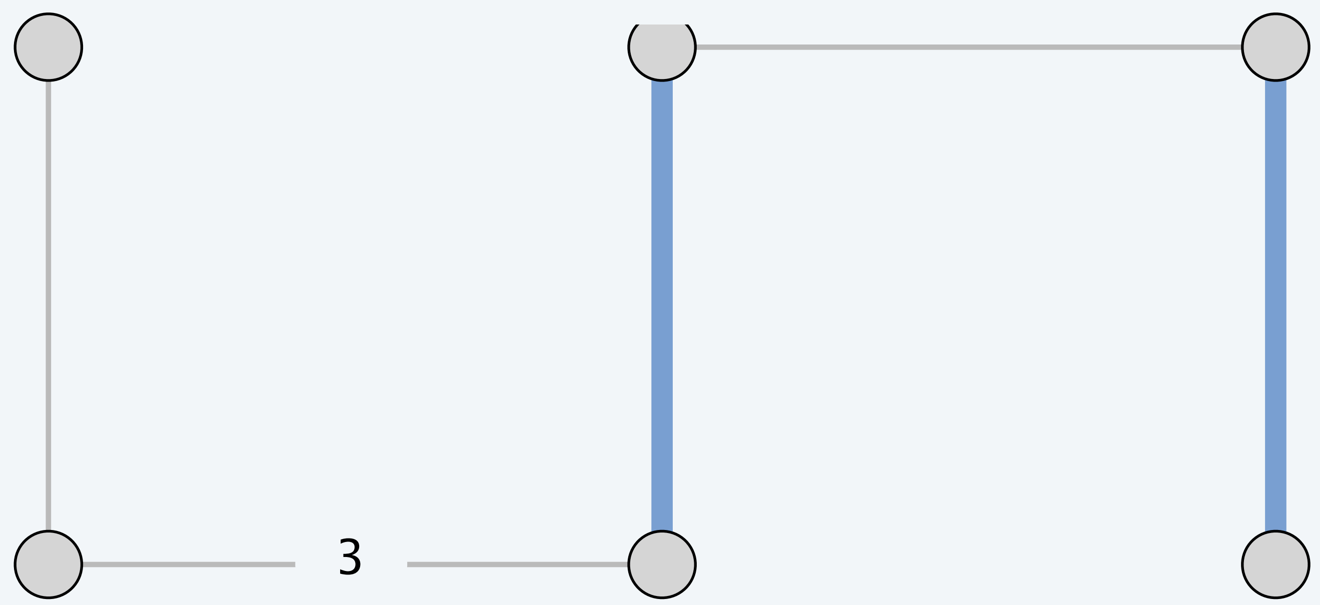
- Delete edge from T unless it would disconnect T.



Reverse-delete algorithm

Start with all edges in T and consider them in descending order of weight:

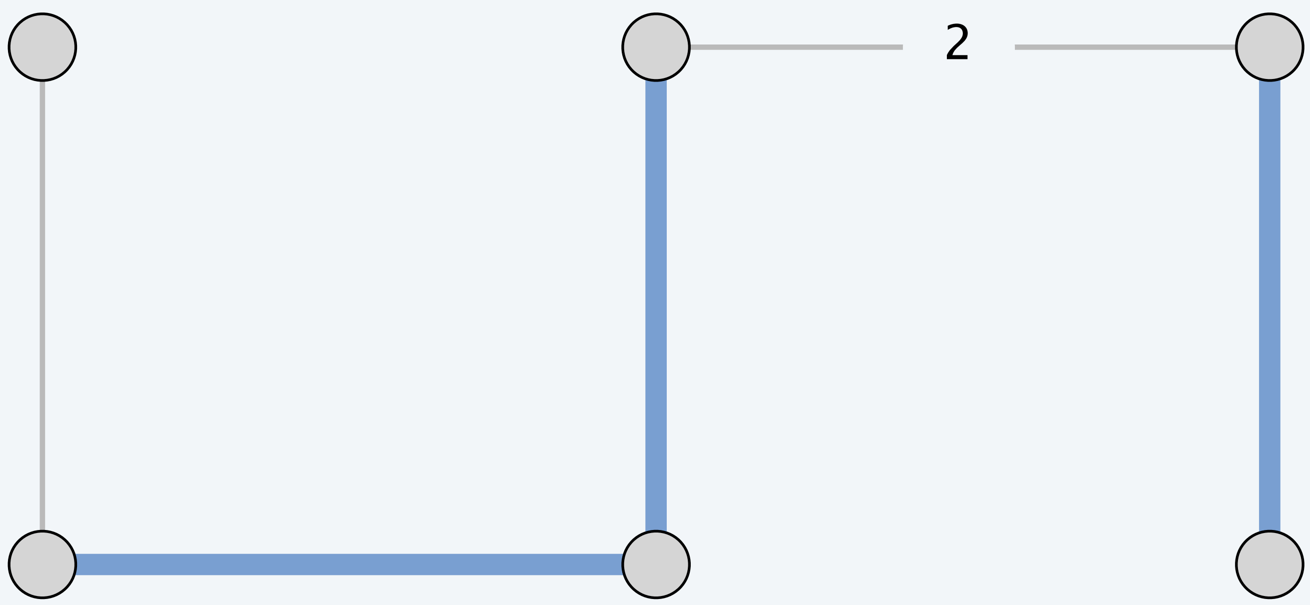
- Delete edge from T unless it would disconnect T.



Reverse-delete algorithm

Start with all edges in T and consider them in descending order of weight:

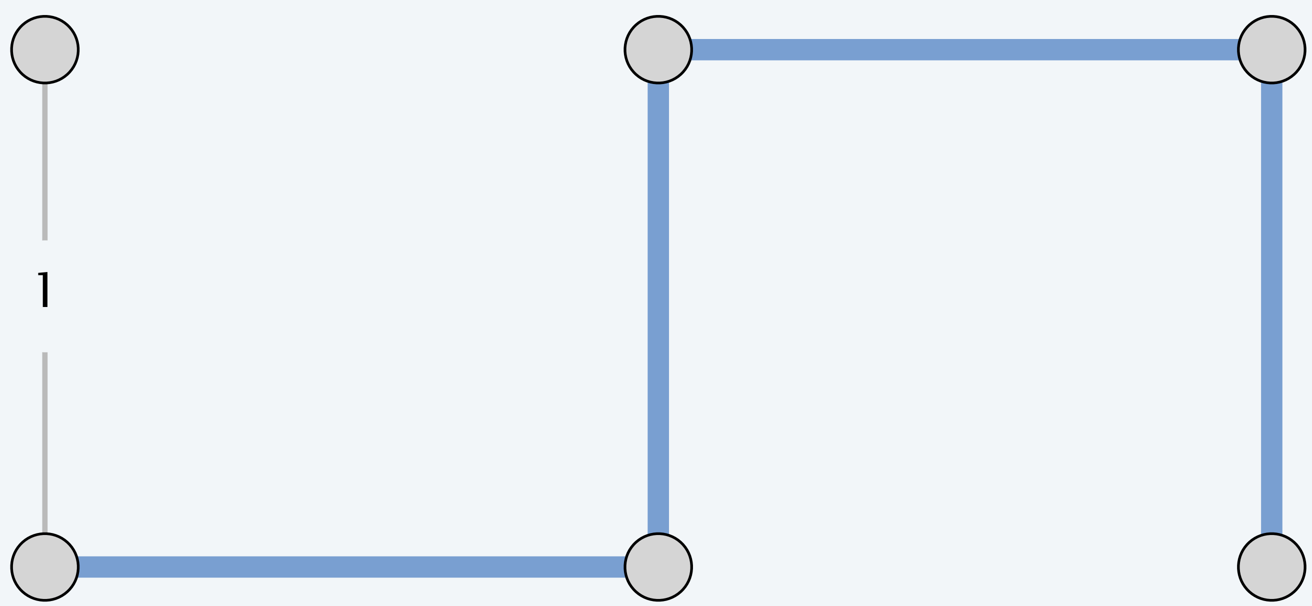
- Delete edge from T unless it would disconnect T.



Reverse-delete algorithm

Start with all edges in T and consider them in descending order of weight:

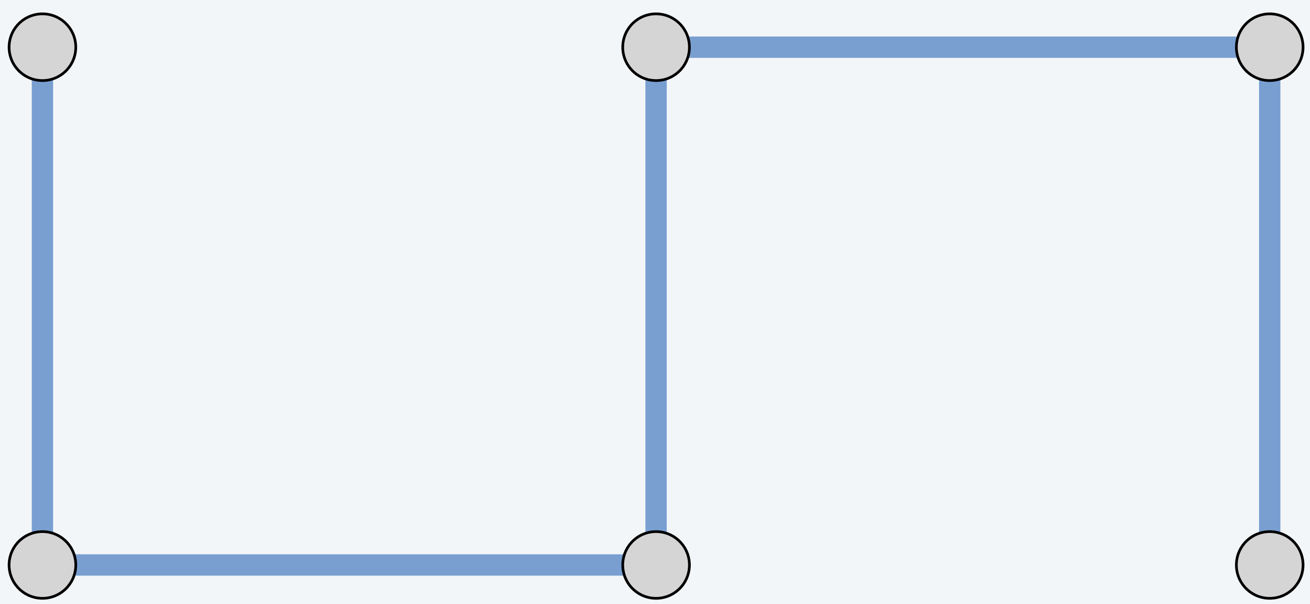
- Delete edge from T unless it would disconnect T.



Reverse-delete algorithm

Start with all edges in T and consider them in descending order of weight:

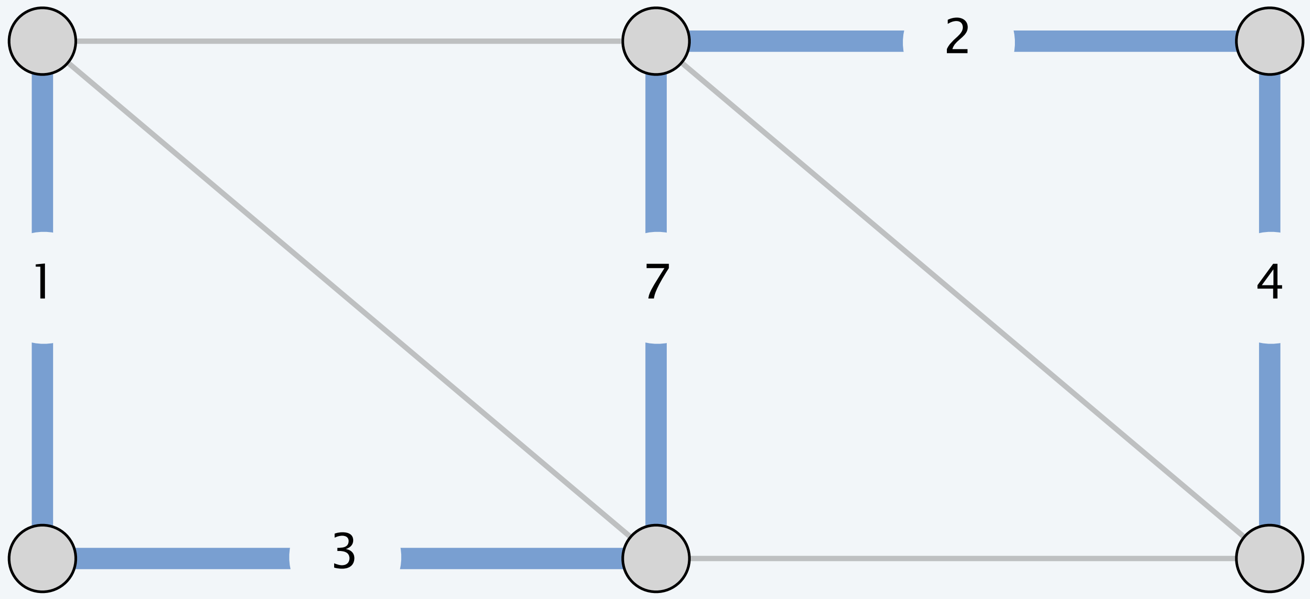
- Delete edge from T unless it would disconnect T.



Reverse-delete algorithm

Start with all edges in T and consider them in descending order of weight:

- Delete edge from T unless it would disconnect T.



Review: the greedy MST algorithm

Red rule.

- Let C be a cycle with no red edges.
- Select an uncolored edge of C of max cost and color it red.

Blue rule.

- Let D be a cutset with no blue edges.
- Select an uncolored edge in D of min cost and color it blue.

Greedy algorithm.

- Apply the red and blue rules (nondeterministically!) until all edges are colored. The blue edges form an MST.
- Note: can stop once $n - 1$ edges colored blue.

Theorem. The greedy algorithm is correct.

Special cases. Prim, Kruskal, reverse-delete, ...