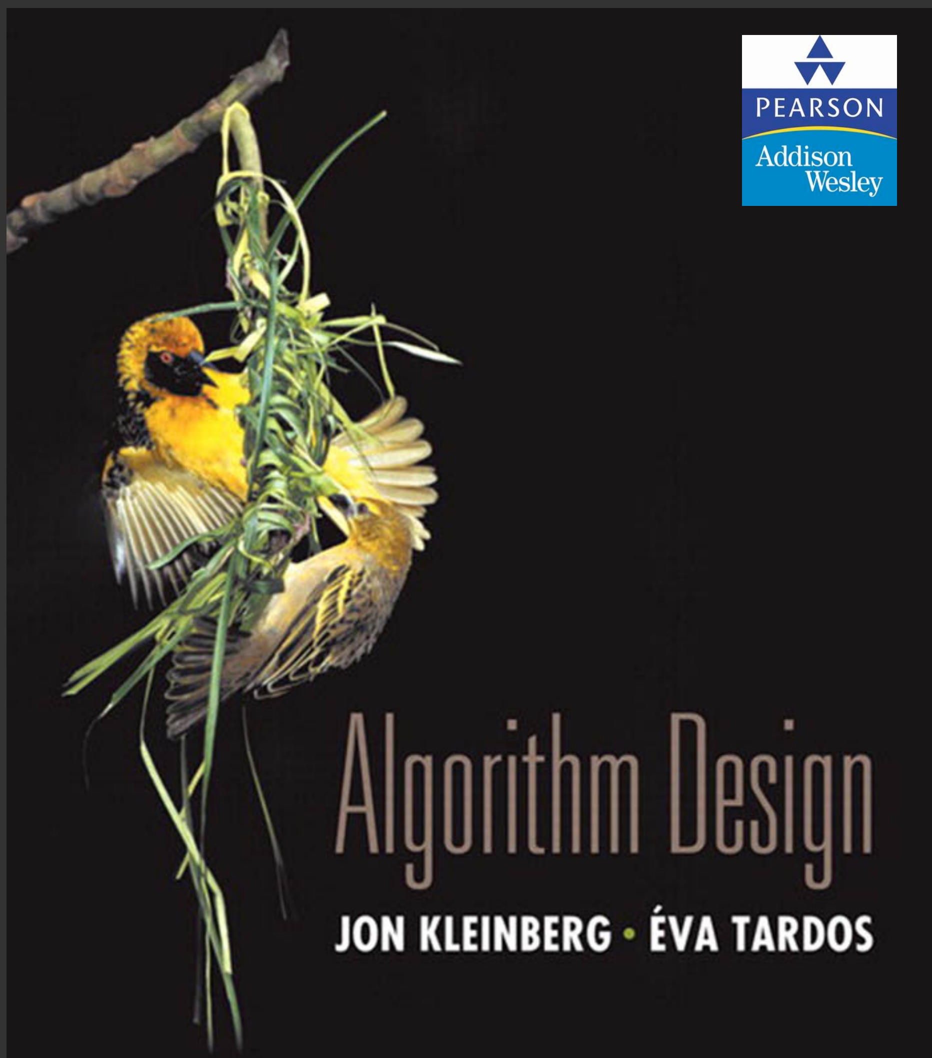


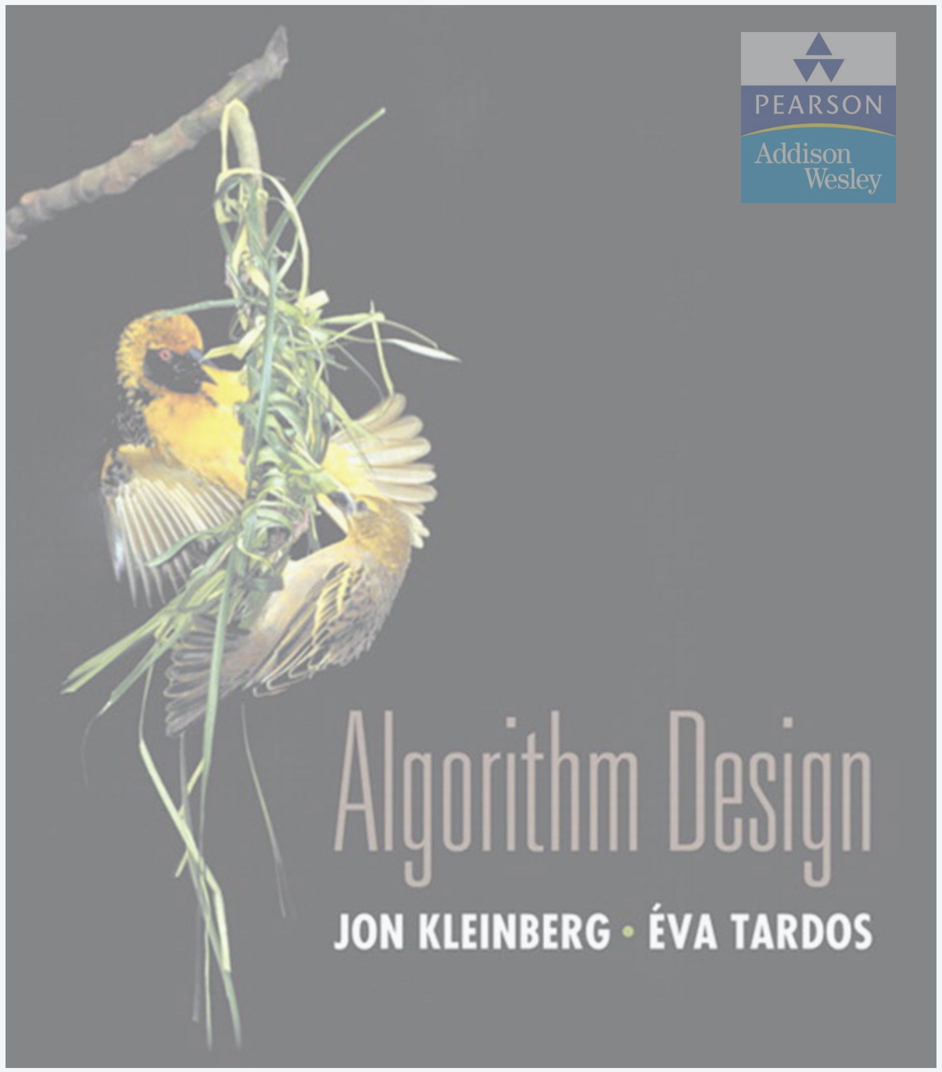
4. Greedy Algorithms II

- *Dijkstra's algorithm*
- *minimum spanning trees*
- *Prim, Kruskal, Boruvka*
- *single-link clustering*
- *min-cost arborescences*

Lecture slides by Kevin Wayne

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

Last updated on 2/25/20 4:16 PM



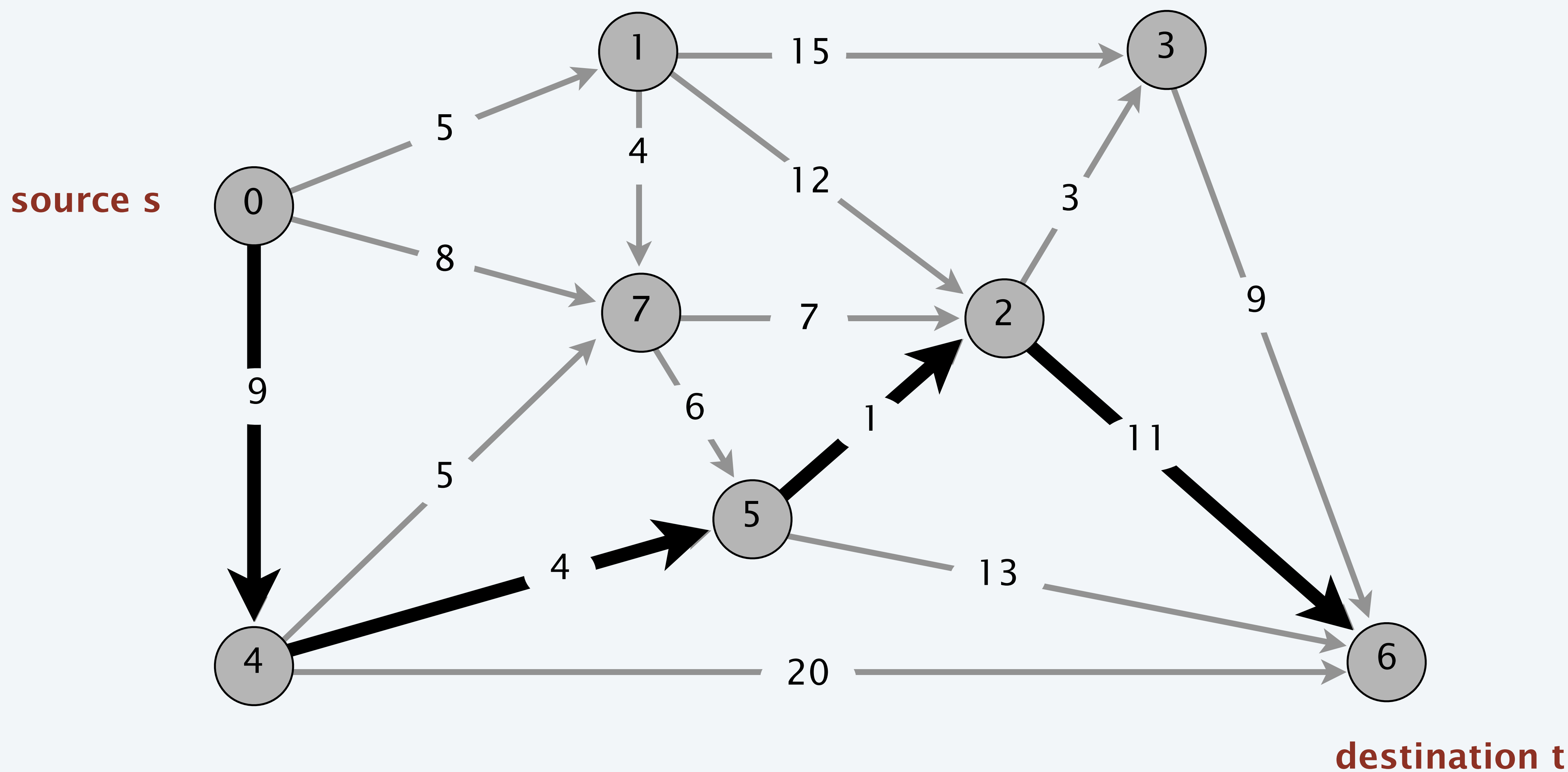
SECTION 4.4

4. GREEDY ALGORITHMS II

- *Dijkstra's algorithm*
- *minimum spanning trees*
- *Prim, Kruskal, Boruvka*
- *single-link clustering*
- *min-cost arborescences*

Single-pair shortest path problem

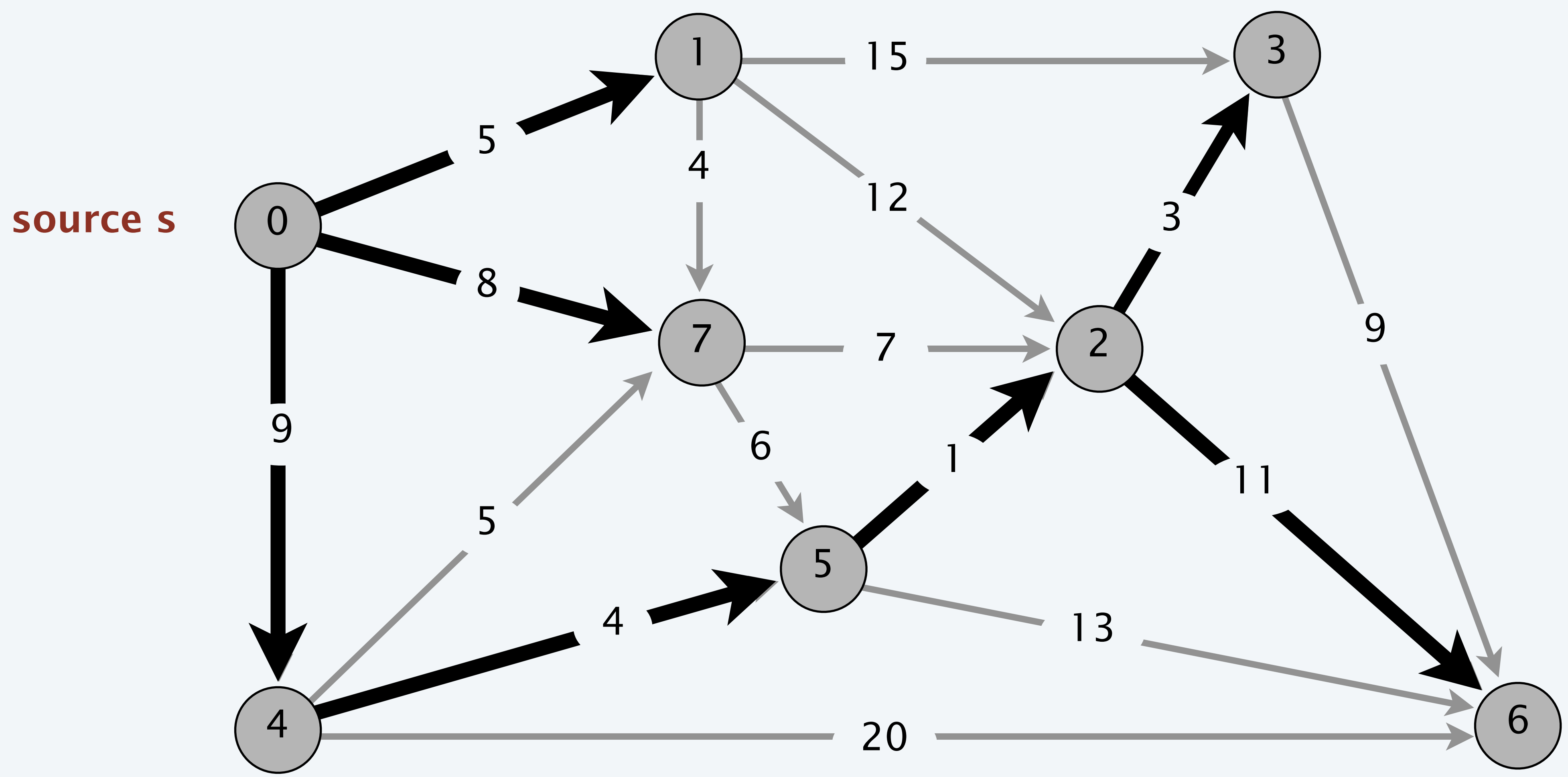
Problem. Given a digraph $G = (V, E)$, edge lengths $\ell_e \geq 0$, source $s \in V$, and destination $t \in V$, find a shortest directed path from s to t.



length of path = 9 + 4 + 1 + 11 = 25

Single-source shortest paths problem

Problem. Given a digraph $G = (V, E)$, edge lengths $\ell_e \geq 0$, source $s \in V$, find a shortest directed path from s to every node.



shortest-paths tree

Shortest paths: quiz 1

Suppose that you change the length of every edge of G as follows. For which is every shortest path in G a shortest path in G′?

A. Add 17.

B. Multiply by 17.

C. Either A or B.

D. Neither A nor B.

Shortest paths: quiz 2

Which variant in car GPS?

A. Single source: from one node s to every other node.

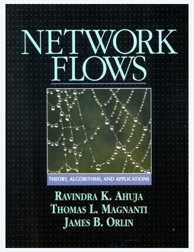
B. Single sink: from every node to one node t.

C. Source–sink: from one node s to another node t.

D. All pairs: between all pairs of nodes.

Shortest path applications

- PERT/CPM.
- Map routing.
- Seam carving.
- Robot navigation.
- Texture mapping.
- Typesetting in LaTeX.
- Urban traffic planning.
- Telemarketer operator scheduling.
- Routing of telecommunications messages.
- Network routing protocols (OSPF, BGP, RIP).
- Optimal truck routing through given traffic congestion pattern.



Network Flows: Theory, Algorithms, and Applications,
by Ahuja, Magnanti, and Orlin, Prentice Hall, 1993.

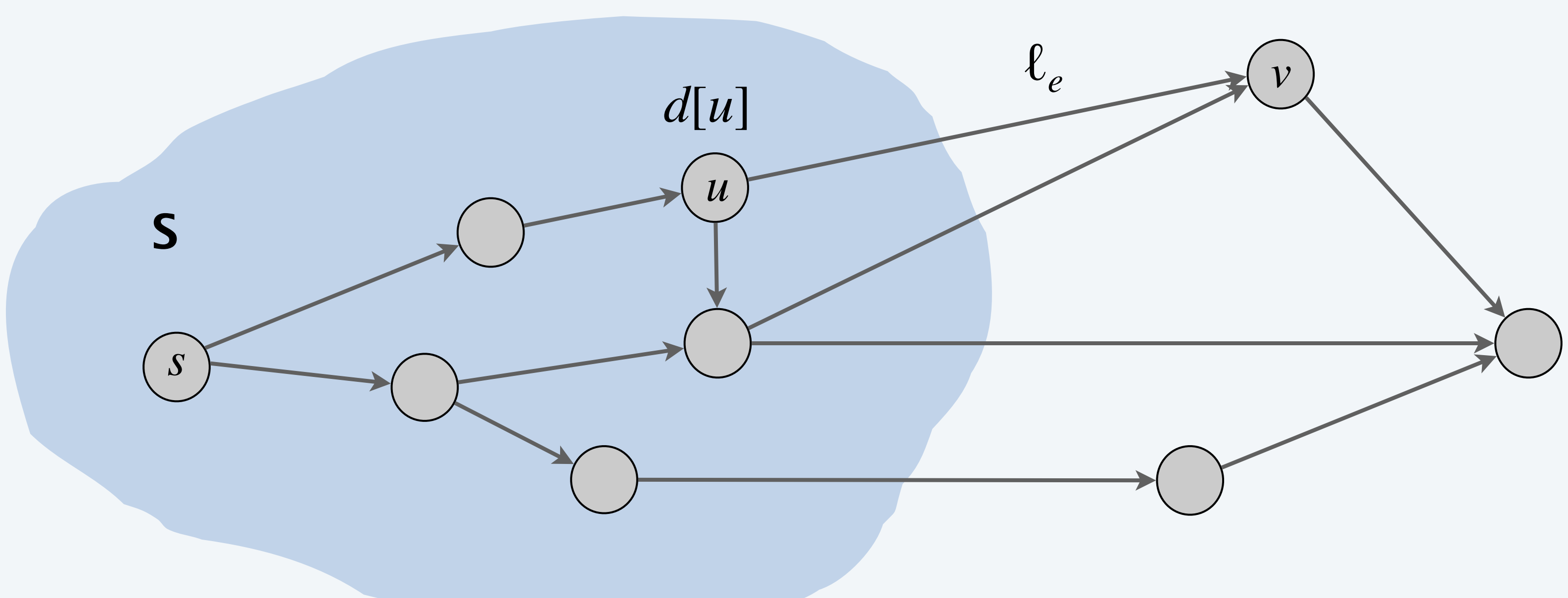
Dijkstra's algorithm (for single-source shortest paths problem)

Greedy approach. Maintain a set of explored nodes S for which algorithm has determined $d[u]$ = length of a shortest $s \rightsquigarrow u$ path.

- Initialize $S \leftarrow \{ s \}$, $d[s] \leftarrow 0$.
- Repeatedly choose unexplored node $v \notin S$ which minimizes

$$\pi(v) = \min_{e = (u,v) \,:\, u \in S} \boxed{d[u] + \ell_e}$$

the length of a shortest path from s to some node u in explored part S, followed by a single edge $e = (u, v)$



Dijkstra's algorithm (for single-source shortest paths problem)

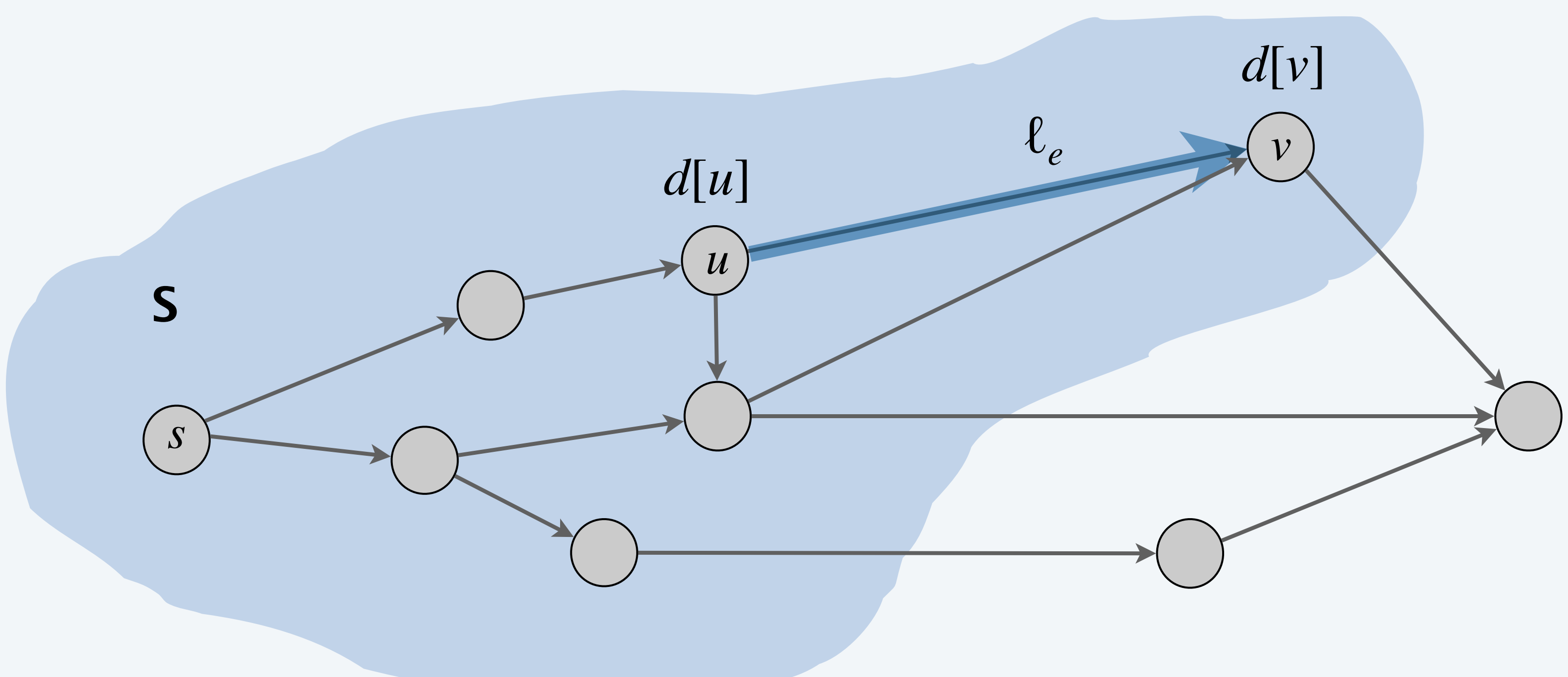
Greedy approach. Maintain a set of explored nodes S for which algorithm has determined $d[u]$ = length of a shortest $s \leadsto u$ path.

- Initialize $S \leftarrow \{ s \}$, $d[s] \leftarrow 0$.
- Repeatedly choose unexplored node $v \notin S$ which minimizes

$$\pi(v) = \min_{e = (u,v) \,:\, u \in S} \boxed{d[u] + \ell_e}$$

the length of a shortest path from s to some node u in explored part S, followed by a single edge $e = (u, v)$

  add v to S, and set $d[v] \leftarrow \pi(v)$.
- To recover path, set $pred[v] \leftarrow e$ that achieves min.



Dijkstra's algorithm: proof of correctness

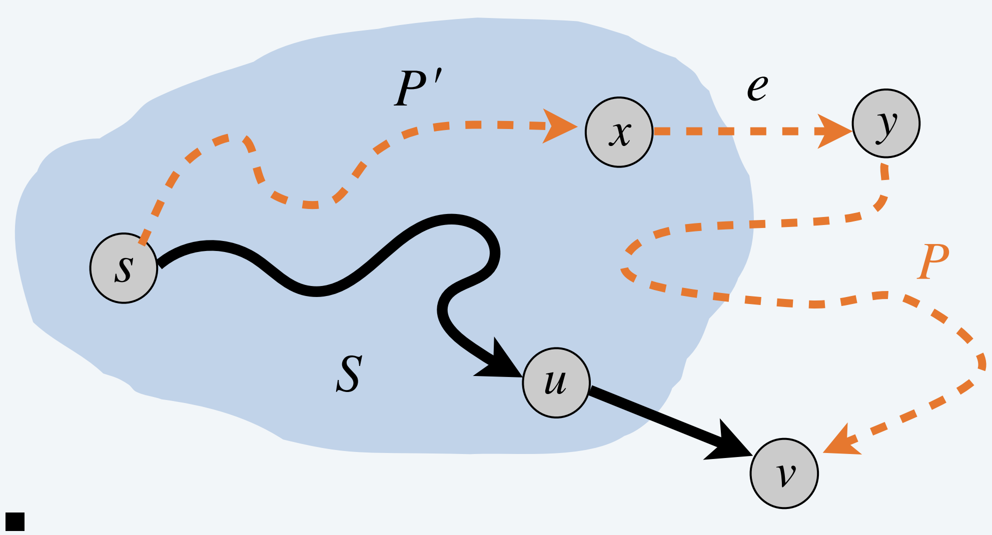
Invariant. For each node $u \in S$: $d[u]$ = length of a shortest $s \leadsto u$ path.

Pf. [by induction on $|S|$]

Base case: $|S| = 1$ is easy since $S = \{ s \}$ and $d[s] = 0$.

Inductive hypothesis: Assume true for $|S| \geq 1$.

- Let v be next node added to S, and let (u, v) be the final edge.
- A shortest $s \leadsto u$ path plus (u, v) is an $s \leadsto v$ path of length $\pi(v)$.
- Consider any other $s \leadsto v$ path P. We show that it is no shorter than $\pi(v)$.
- Let $e = (x, y)$ be the first edge in P that leaves S, and let P' be the subpath from s to x.
- The length of P is already $\geq \pi(v)$ as soon as it reaches y:



$$\ell(P) \underbrace{\geq}_{\text{non-negative lengths}} \ell(P') + \ell_e \underbrace{\geq}_{\text{inductive hypothesis}} d[x] + \ell_e \underbrace{\geq}_{\text{definition of } \pi(y)} \pi(y) \underbrace{\geq}_{\text{Dijkstra chose } v \text{ instead of } y} \pi(v) \quad \blacksquare$$

Dijkstra's algorithm: efficient implementation

Critical optimization 1. For each unexplored node $v \notin S$:
explicitly maintain $\pi[v]$ instead of computing directly from definition

$$\pi(v) \;=\; \min_{e\,=\,(u,v)\;:\;u \in S} d[u] + \ell_e$$

- For each $v \notin S$: $\pi(v)$ can only decrease (because set S increases).

- More specifically, suppose u is added to S and there is an edge $e = (u, v)$ leaving u. Then, it suffices to update:

$$\pi[v] \leftarrow \min \{\, \pi[v],\; \pi[u] + \ell_e) \,\}$$

recall: for each $u \in S$,
$\pi[u] = d[u]$ = length of shortest $s \rightsquigarrow u$ path

Critical optimization 2. Use a min-oriented priority queue (PQ)
to choose an unexplored node that minimizes $\pi[v]$.

Dijkstra's algorithm: efficient implementation

Implementation.

- Algorithm maintains $\pi[v]$ for each node v.
- Priority queue stores unexplored nodes, using $\pi[\cdot]$ as priorities.
- Once u is deleted from the PQ, $\pi[u]$ = length of a shortest $s \rightsquigarrow u$ path.

```
DIJKSTRA (V, E, ℓ, s)

FOREACH v ≠ s : π[v] ← ∞, pred[v] ← null; π[s] ← 0.
Create an empty priority queue pq.
FOREACH v ∈ V : INSERT(pq, v, π[v]).
WHILE (IS-NOT-EMPTY(pq))
    u ← DEL-MIN(pq).
    FOREACH edge e = (u, v) ∈ E leaving u:
        IF (π[v] > π[u] + ℓe)
            DECREASE-KEY(pq, v, π[u] + ℓe).
            π[v] ← π[u] + ℓe ; pred[v] ← e.
```

Dijkstra's algorithm: which priority queue?

Performance. Depends on PQ: n INSERT, n DELETE-MIN, $\leq m$ DECREASE-KEY.

- Array implementation optimal for dense graphs. ← $\Theta(n^2)$ edges
- Binary heap much faster for sparse graphs. ← $\Theta(n)$ edges
- 4-way heap worth the trouble in performance-critical situations.

priority queue	INSERT	DELETE-MIN	DECREASE-KEY	total
node-indexed array (A[i] = priority of i)	$O(1)$	$O(n)$	$O(1)$	$O(n^2)$
binary heap	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(m \log n)$
d-way heap (Johnson 1975)	$O(d \log_d n)$	$O(d \log_d n)$	$O(\log_d n)$	$O(m \log_{m/n} n)$
Fibonacci heap (Fredman–Tarjan 1984)	$O(1)$	$O(\log n)$ †	$O(1)$ †	$O(m + n \log n)$
integer priority queue (Thorup 2004)	$O(1)$	$O(\log \log n)$	$O(1)$	$O(m + n \log \log n)$

assumes $m \geq n$ † amortized

Shortest paths: quiz 3

How to solve the the single-source shortest paths problem in undirected graphs with positive edge lengths?

- **A.** Replace each undirected edge with two antiparallel edges of same length. Run Dijkstra's algorithm in the resulting digraph.
- **B.** Modify Dijkstra's algorithms so that when it processes node u, it consider all edges incident to u (instead of edges leaving u).
- **C.** Either A or B.
- **D.** Neither A nor B.

Shortest paths: quiz 3

Theorem. [Thorup 1999] Can solve single-source shortest paths problem in undirected graphs with positive integer edge lengths in $O(m)$ time.

Remark. Does not explore nodes in increasing order of distance from s.

Undirected Single Source Shortest Paths with Positive Integer Weights in Linear Time

Mikkel Thorup
AT&T Labs—Research

The single source shortest paths problem (SSSP) is one of the classic problems in algorithmic graph theory: given a positively weighted graph G with a source vertex s, find the shortest path from s to all other vertices in the graph.

Since 1959 all theoretical developments in SSSP for general directed and undirected graphs have been based on Dijkstra's algorithm, visiting the vertices in order of increasing distance from s. Thus, any implementation of Dijkstra's algorithm sorts the vertices according to their distances from s. However, we do not know how to sort in linear time.

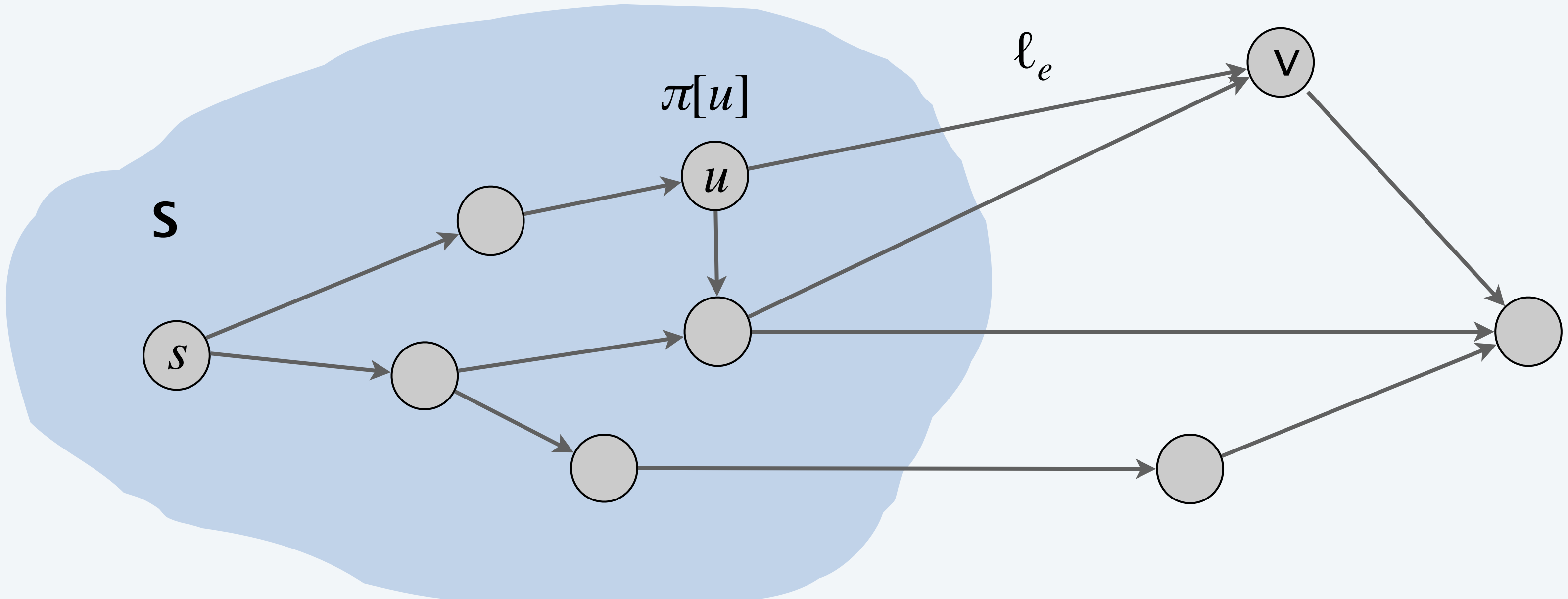
Here, a deterministic linear time and linear space algorithm is presented for the undirected single source shortest paths problem with positive integer weights. The algorithm avoids the sorting bottle-neck by building a hierarchical bucketing structure, identifying vertex pairs that may be visited in any order.

Extensions of Dijkstra's algorithm

Dijkstra's algorithm and proof extend to several related problems:

- Shortest paths in undirected graphs: $\pi[v] \le \pi[u] + \ell(u, v)$.
- Maximum capacity paths: $\pi[v] \ge \min \{ \pi[u], c(u, v) \}$.
- Maximum reliability paths: $\pi[v] \ge \pi[u] \times \gamma(u, v)$.
- ...

Key algebraic structure. Closed semiring (min-plus, bottleneck, Viterbi, ...).

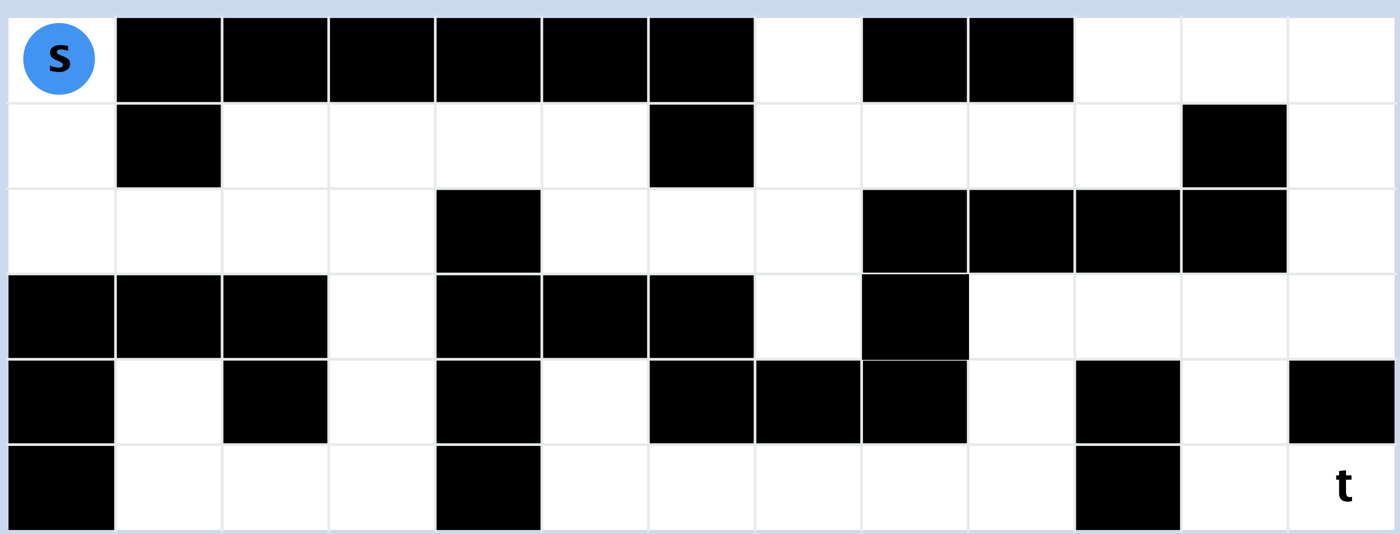


$$
\begin{aligned}
a + b &= b + a \\
a + (b + c) &= (a + b) + c \\
a + 0 &= a \\
a \cdot (b \cdot c) &= (a \cdot b) \cdot c \\
a \cdot 0 &= 0 \cdot a = 0 \\
a \cdot 1 &= 1 \cdot a = a \\
a \cdot (b + c) &= a \cdot b + a \cdot c \\
(a + b) \cdot c &= a \cdot c + b \cdot c \\
a^* &= 1 + a \cdot a^* = 1 + a^* \cdot a
\end{aligned}
$$

Google's foo.bar Challenge

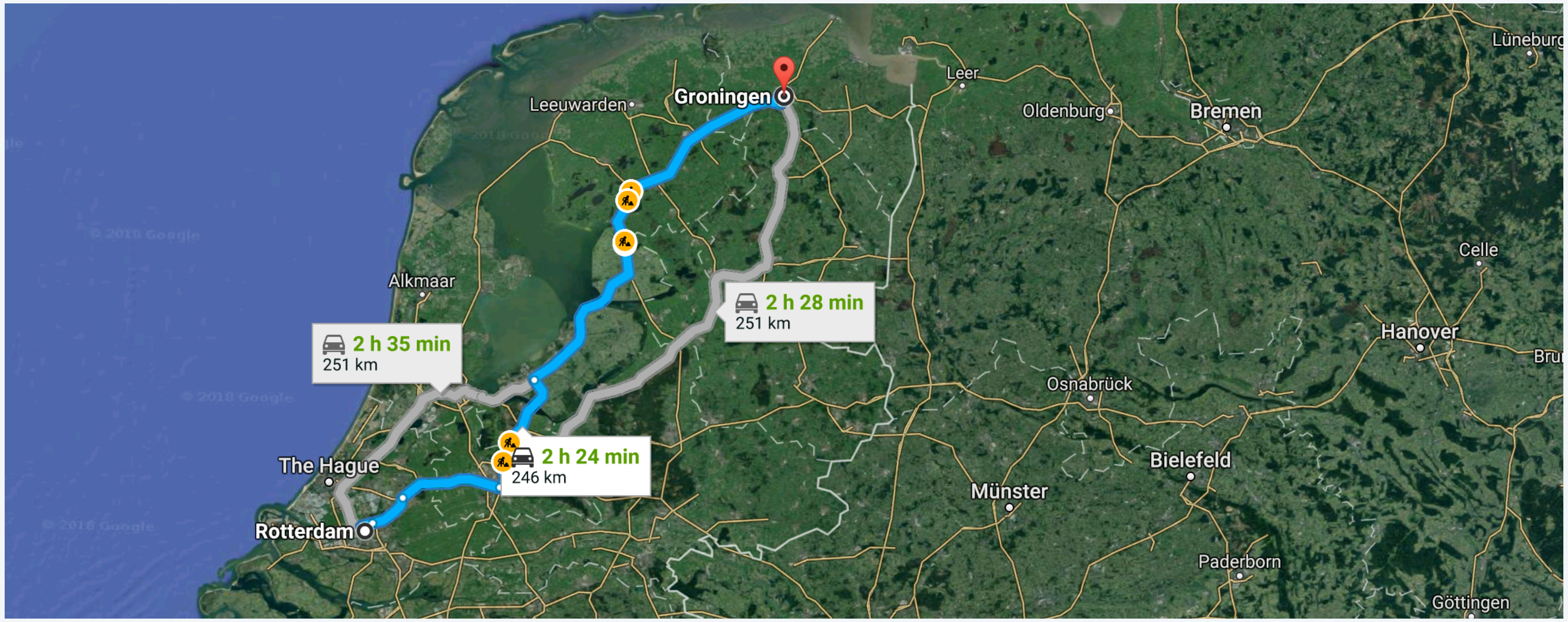
You have maps of parts of the space station, each starting at a prison exit and ending at the door to an escape pod. The map is represented as a matrix of 0s and 1s, where 0s are passable space and 1s are impassable walls. The door out of the prison is at the top left $(0, 0)$ and the door into an escape pod is at the bottom right $(w{-}1, h{-}1)$.

Write a function that generates the length of a shortest path from the prison door to the escape pod, where you are allowed to remove one wall as part of your remodeling plans.

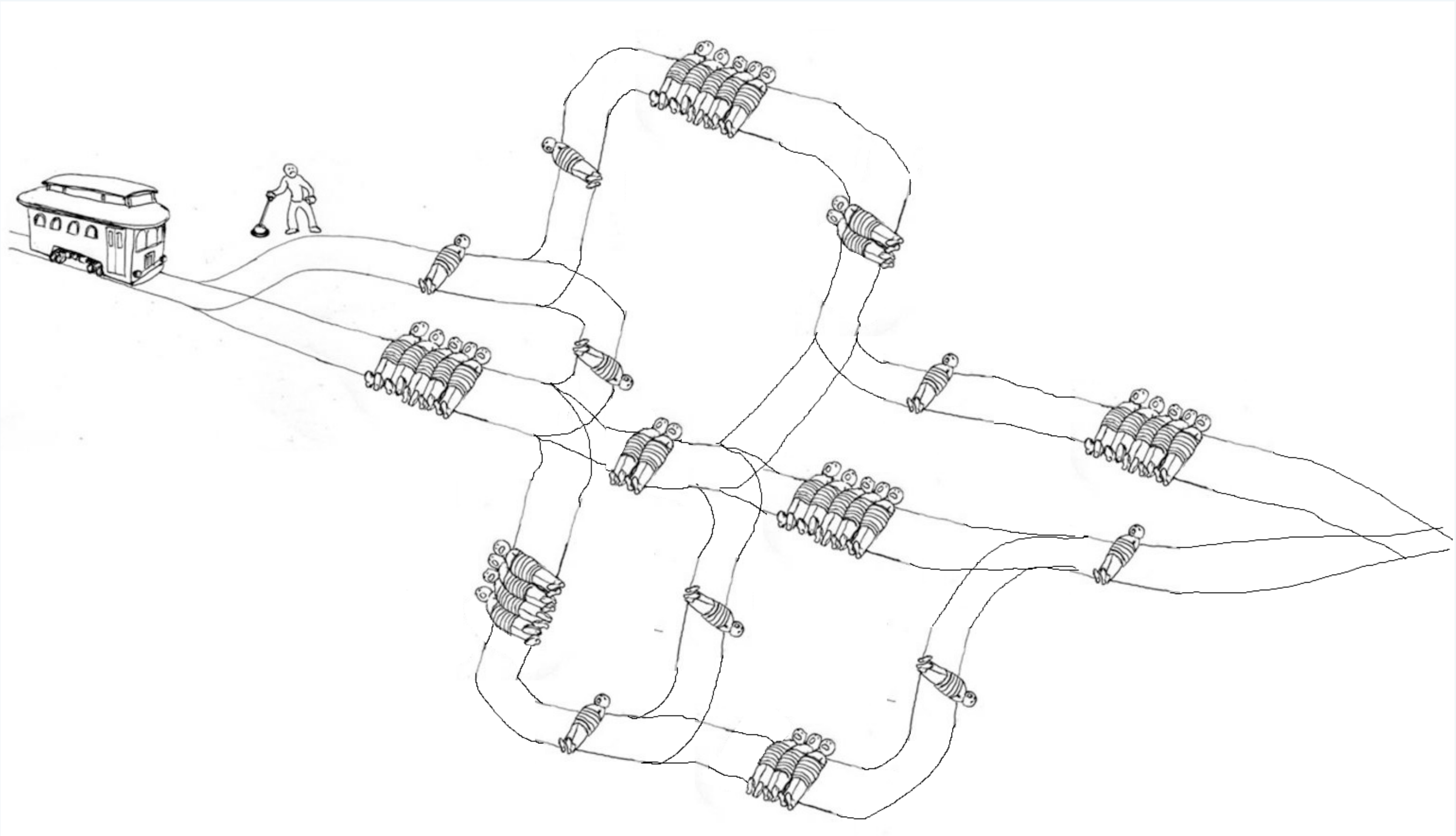


Edsger Dijkstra

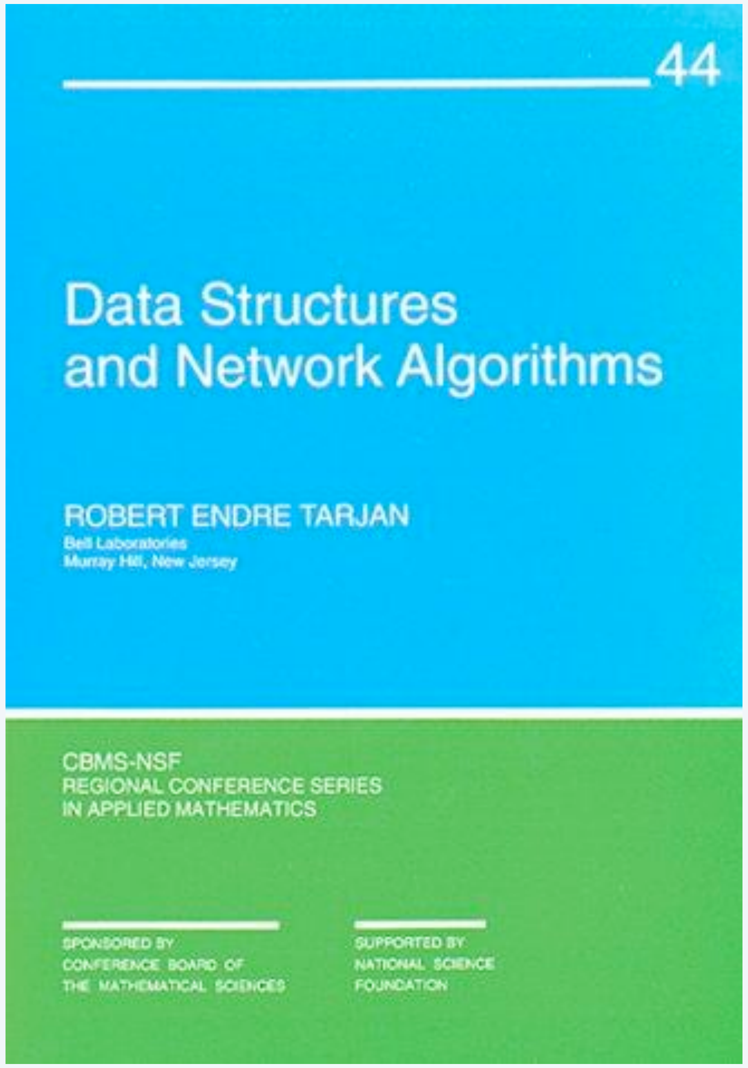
> “ *What’s the shortest way to travel from Rotterdam to Groningen? It is the algorithm for the shortest path, which I designed in about 20 minutes. One morning I was shopping in Amsterdam with my young fiancée, and tired, we sat down on the café terrace to drink a cup of coffee and I was just thinking about whether I could do this, and I then designed the algorithm for the shortest path.* ” — Edsger Dijsktra



The moral implications of implementing shortest-path algorithms



https://www.facebook.com/pg/npcompleteteens



SECTION 6.1

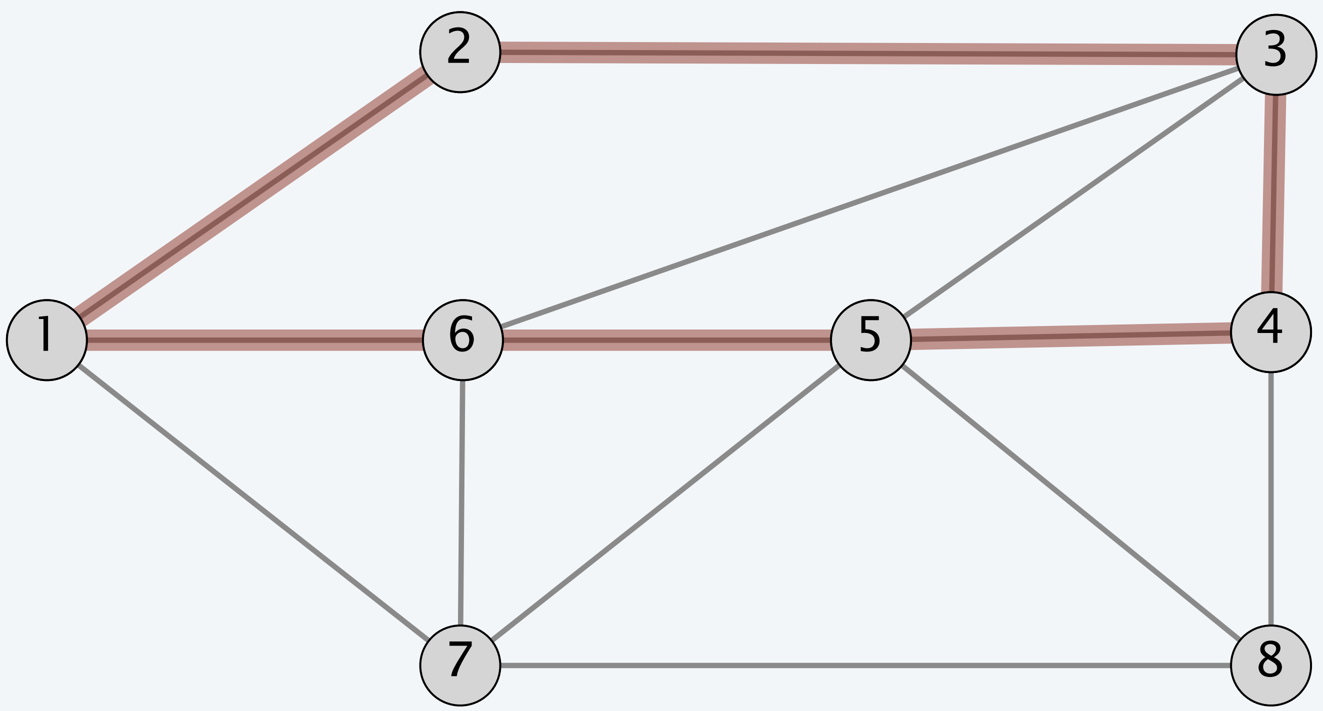
4. GREEDY ALGORITHMS II

- *Dijkstra's algorithm*
- *minimum spanning trees*
- *Prim, Kruskal, Boruvka*
- *single-link clustering*
- *min-cost arborescences*

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.



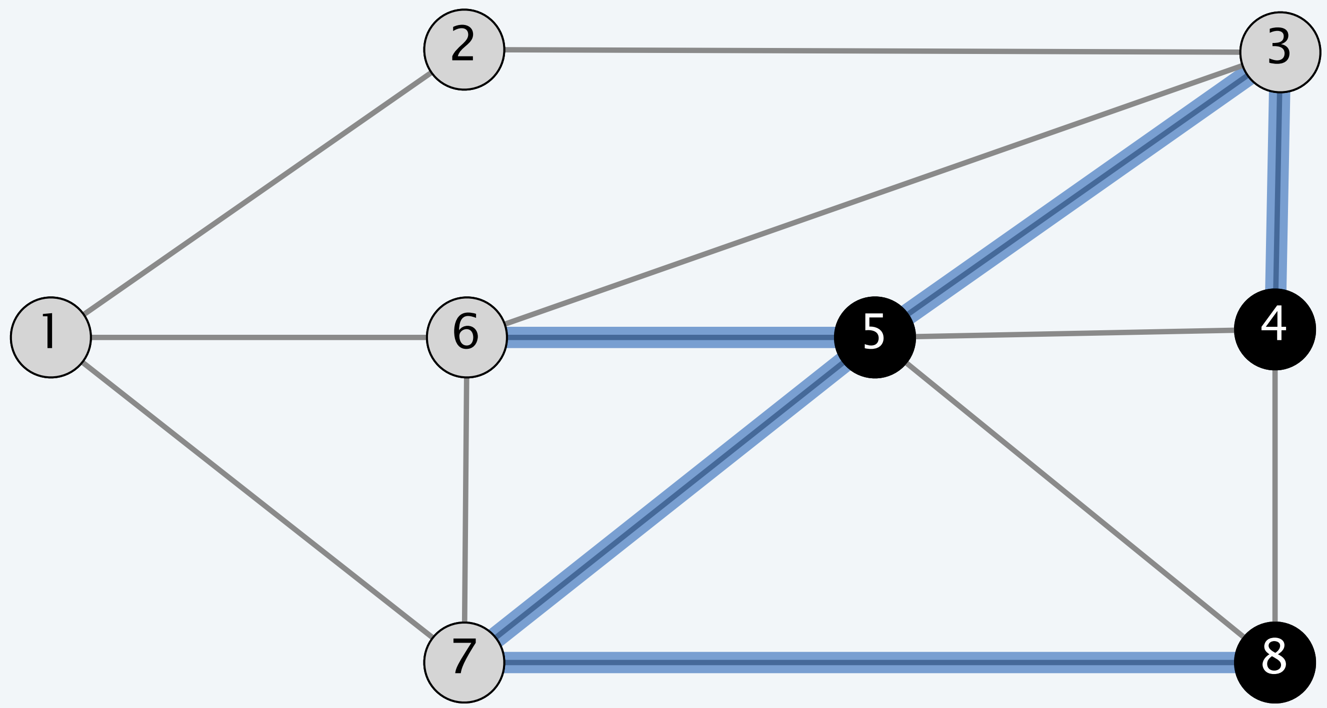
path P = { (1, 2), (2, 3), (3, 4), (4, 5), (5, 6) }

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) }

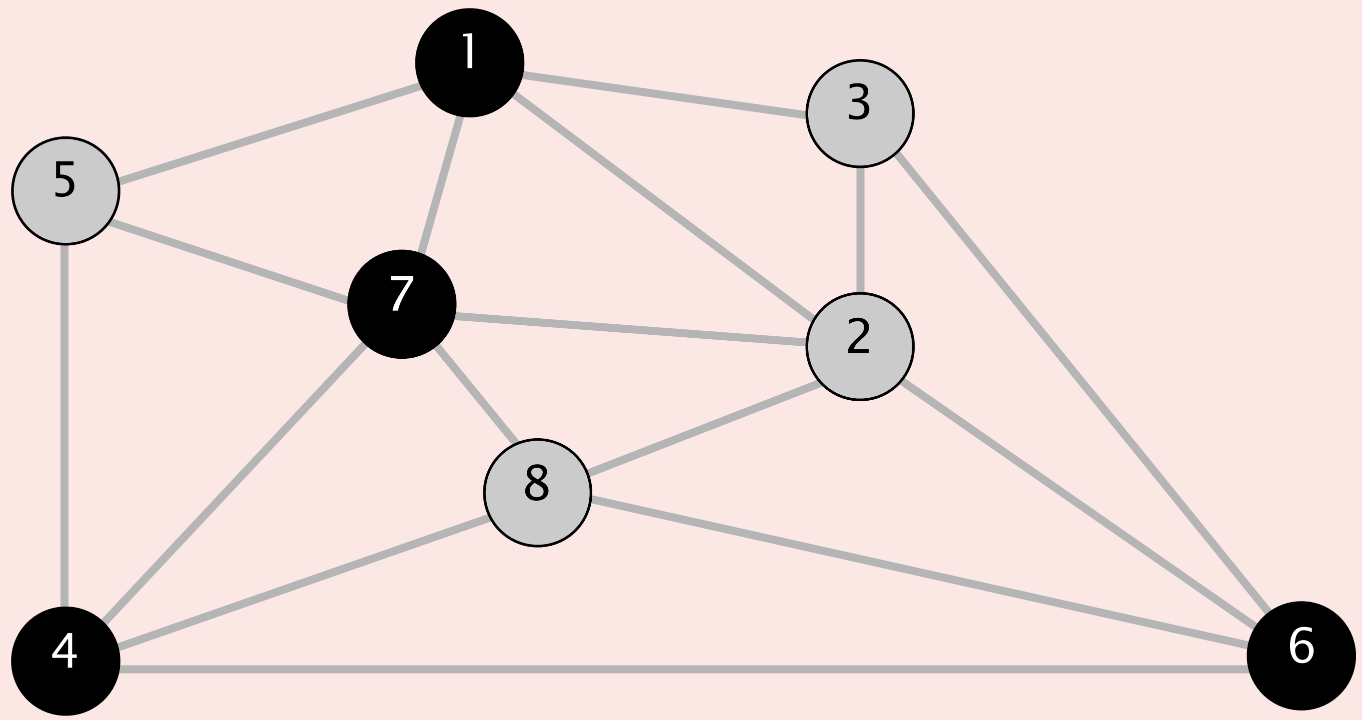
Consider the cut S = { 1, 4, 6, 7 }. Which edge is in the cutset of S?

A. S is not a cut (not connected)

B. 1–7

C. 5–7

D. 2–3



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

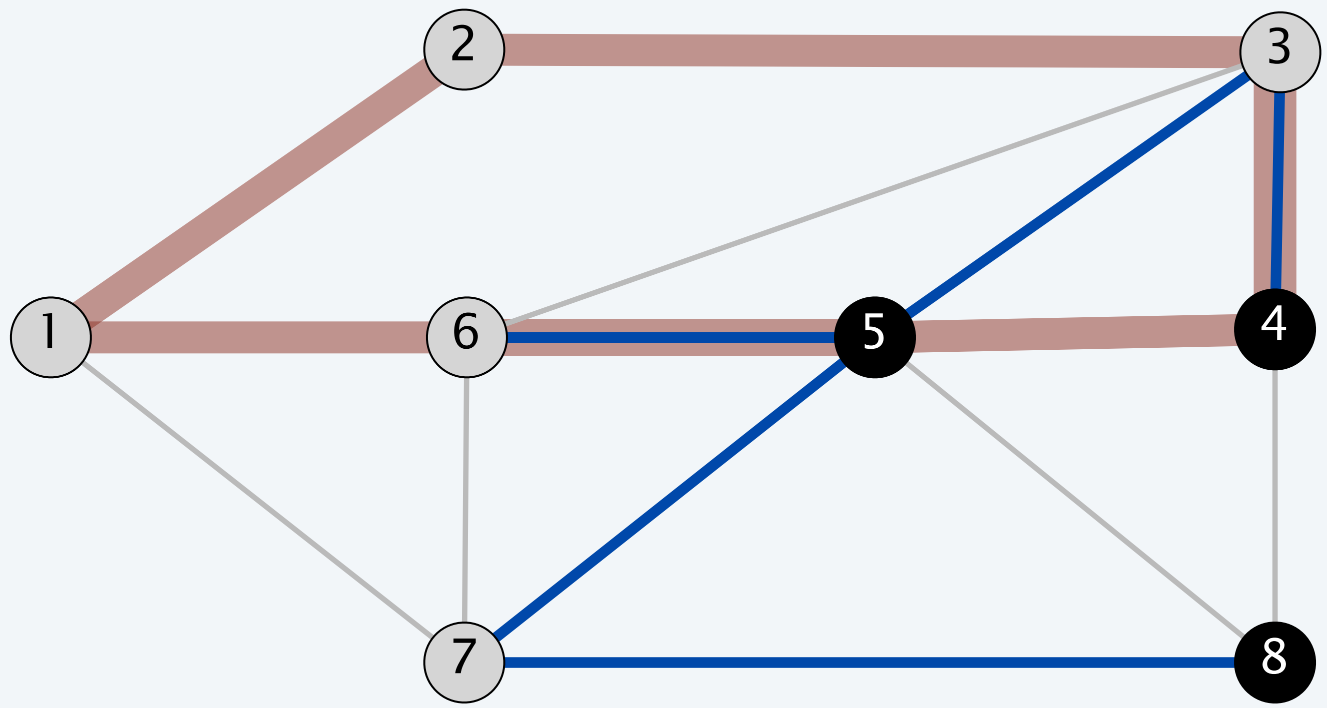
B. 2

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) }

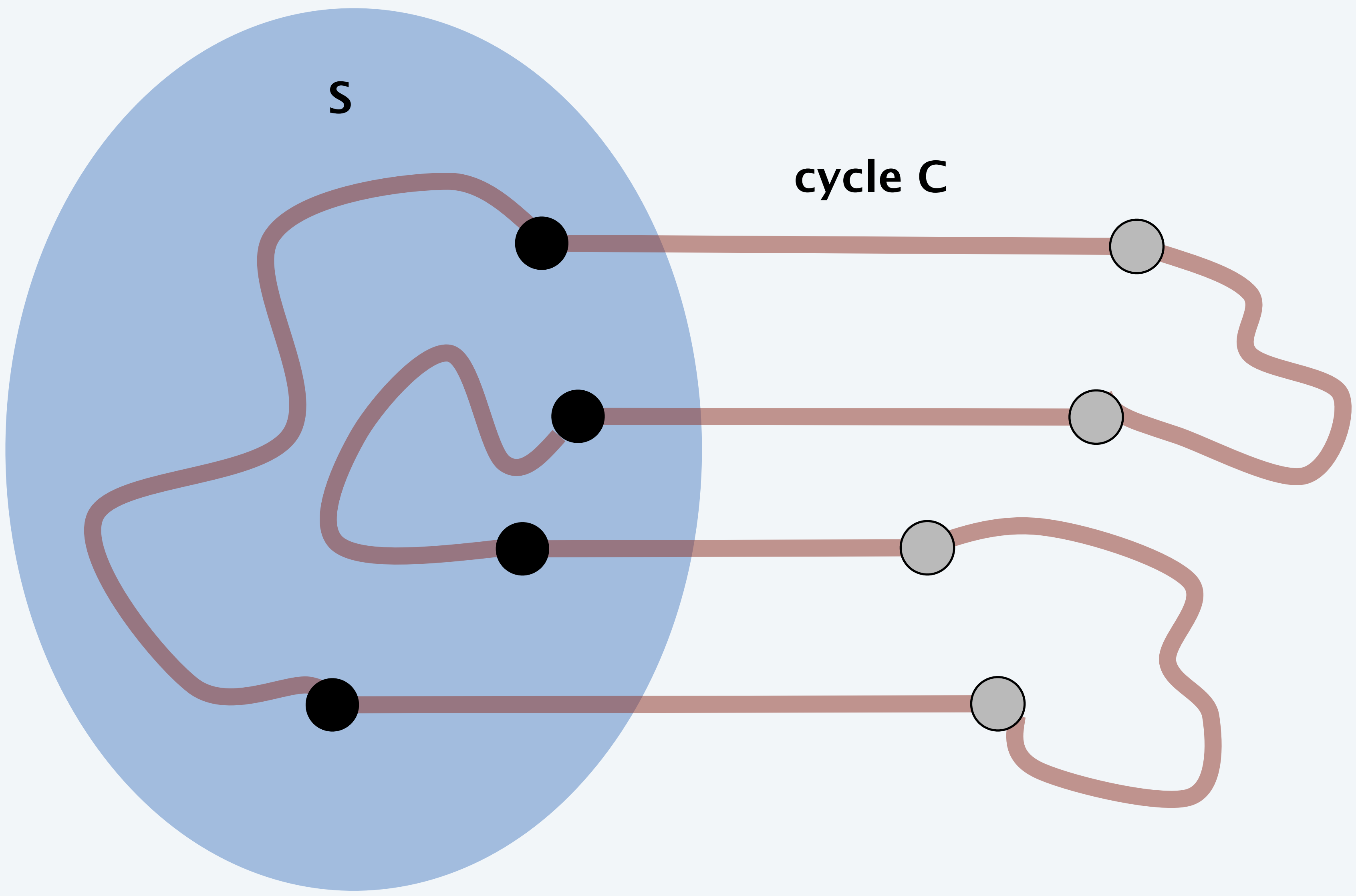
cutset D = { (3, 4), (3, 5), (5, 6), (5, 7), (8, 7) }

intersection C ∩ D = { (3, 4), (5, 6) }

Cycle-cut intersection

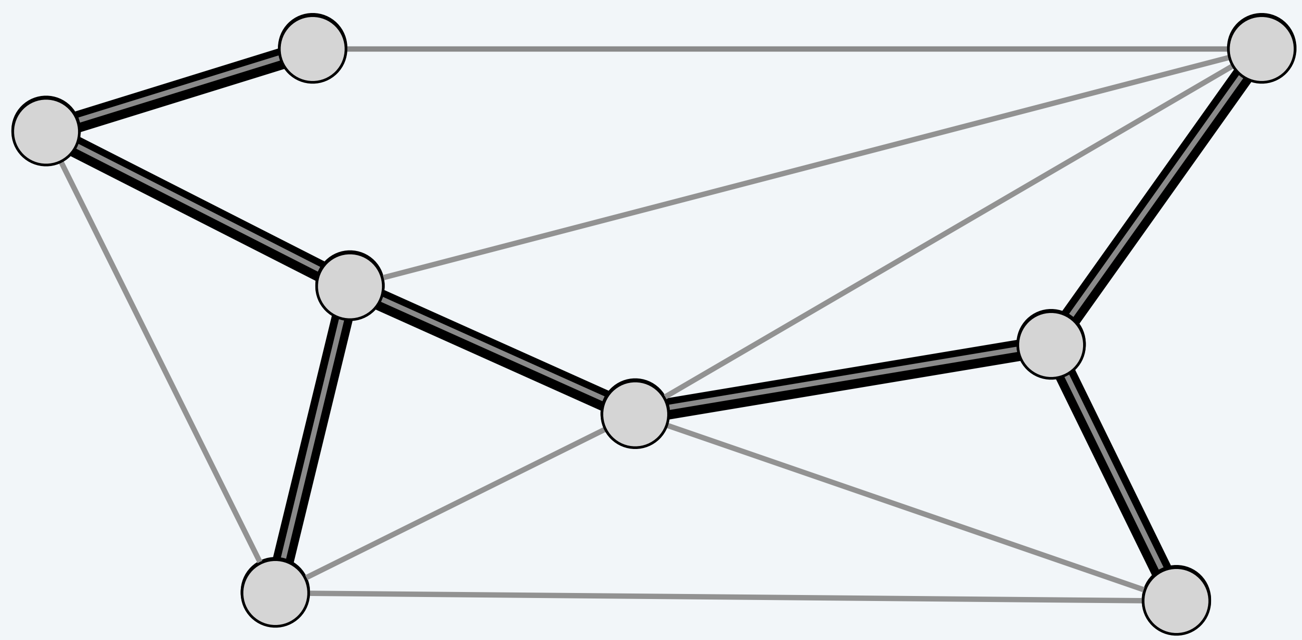
Proposition. A cycle and a cutset intersect in an even number of edges.

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

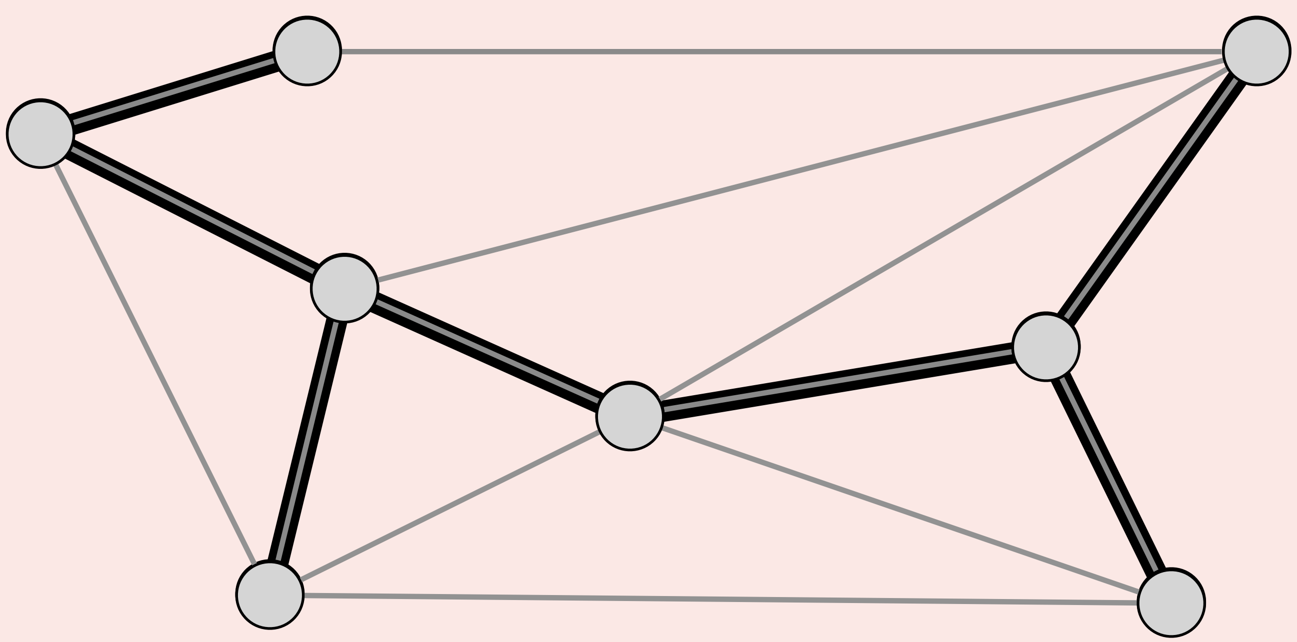
Which of the following properties are true for all spanning trees H?

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.



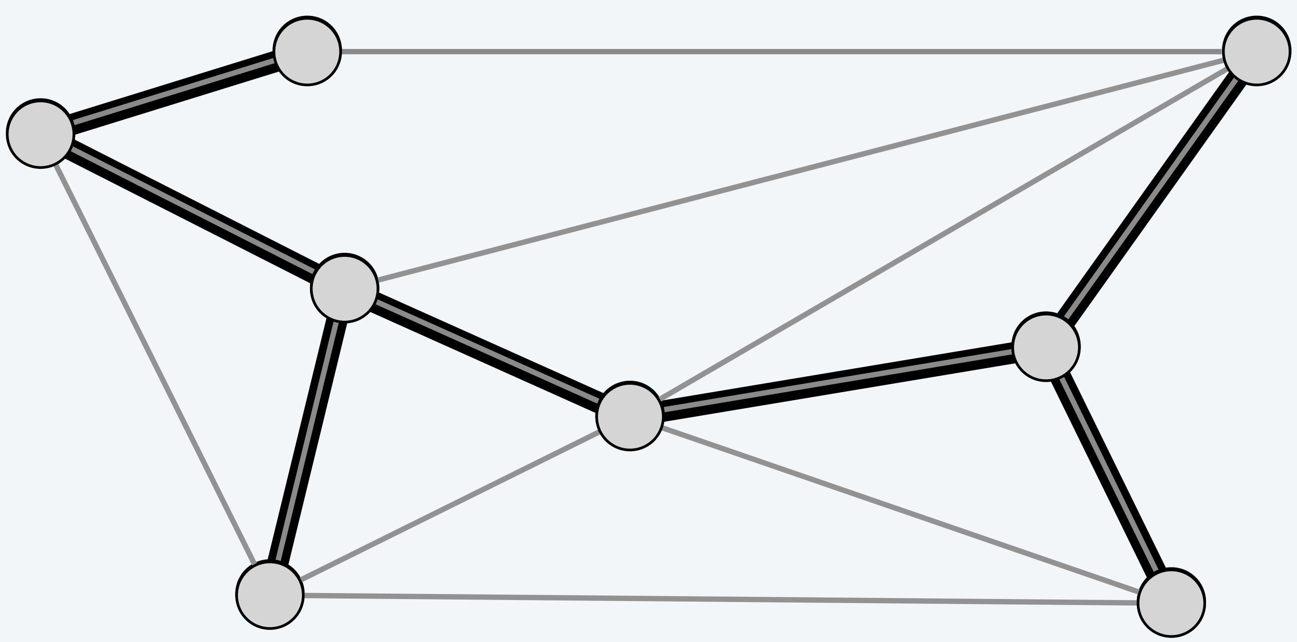
graph G = (V, E)

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)

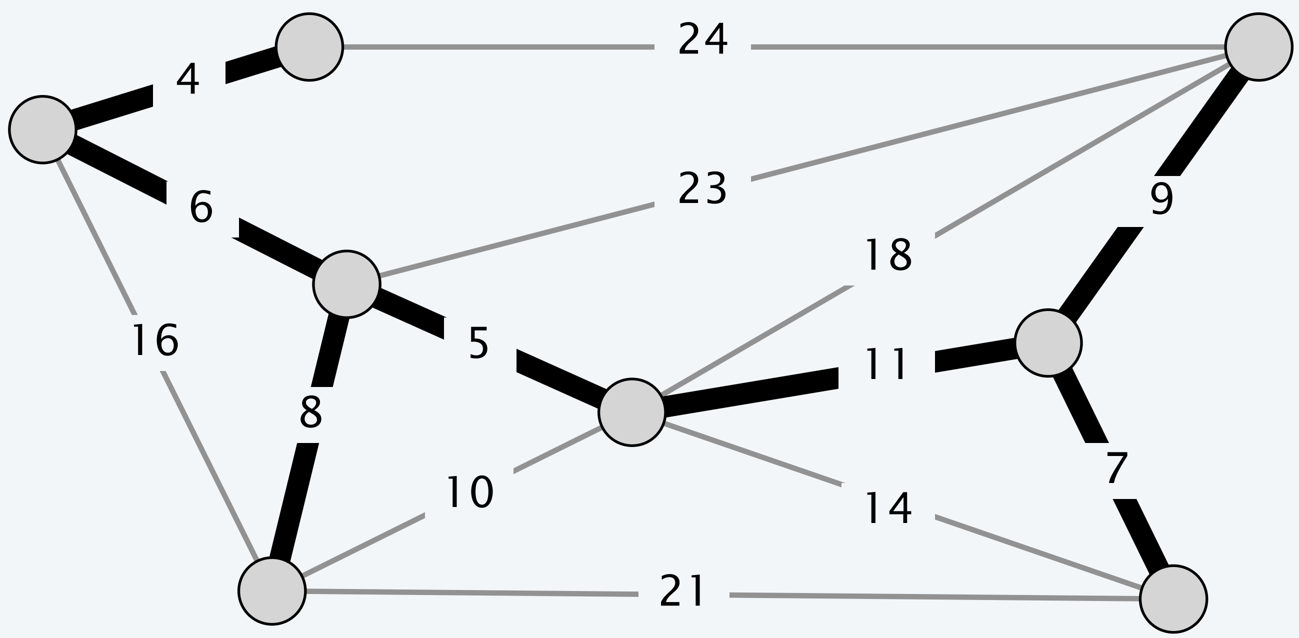
spanning tree H = (V, T)

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

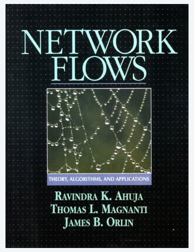
C. $c'(e) = \log_{17} c(e)$.

D. All of the above.

Applications

MST is fundamental problem with diverse applications.

- Dithering.
- Cluster analysis.
- Max bottleneck paths.
- Real-time face verification.
- LDPC codes for error correction.
- Image registration with Renyi entropy.
- Find road networks in satellite and aerial imagery.
- Model locality of particle interactions in turbulent fluid flows.
- Reducing data storage in sequencing amino acids in a protein.
- Autoconfig protocol for Ethernet bridging to avoid cycles in a network.
- Approximation algorithms for NP-hard problems (e.g., TSP, Steiner tree).
- Network design (communication, electrical, hydraulic, computer, road).

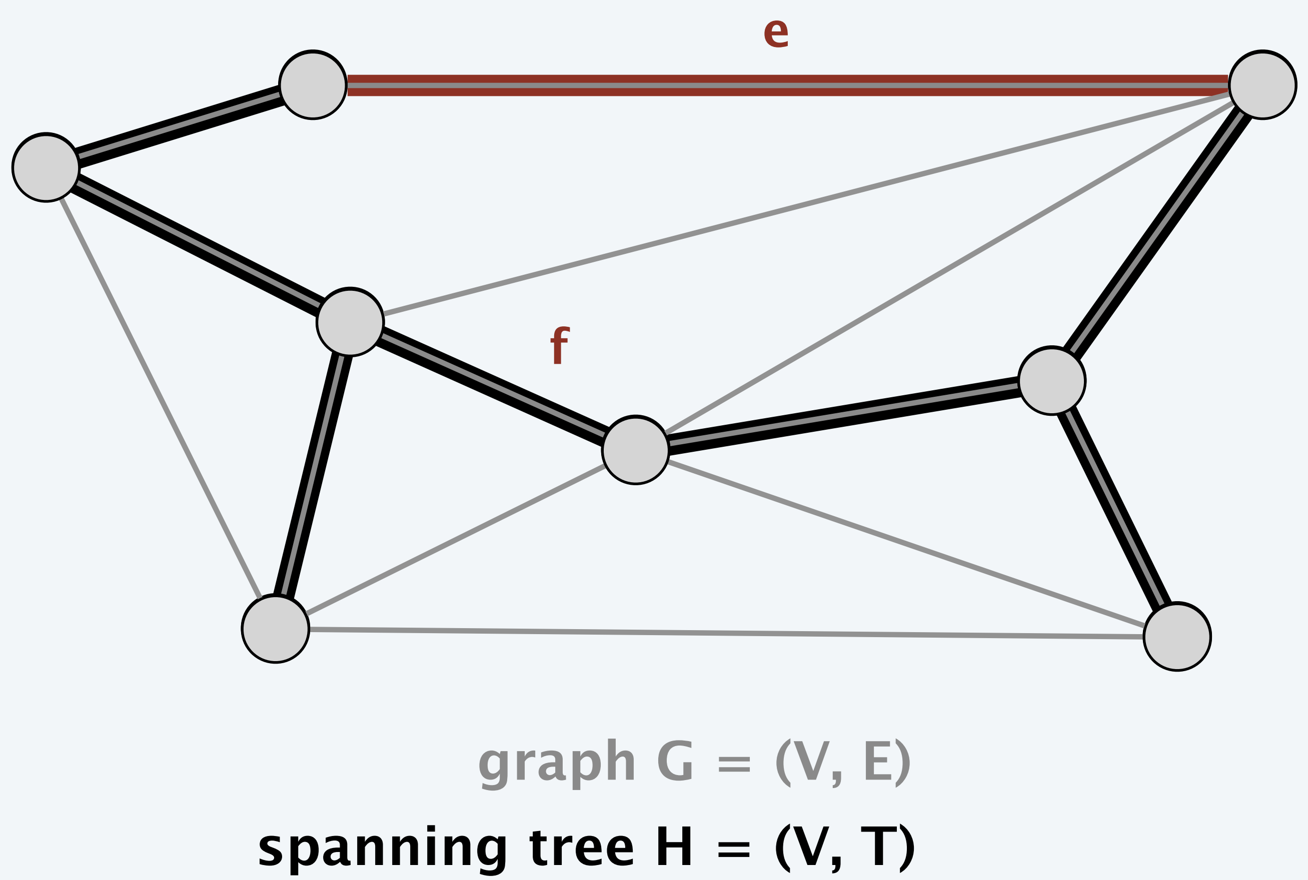


Network Flows: Theory, Algorithms, and Applications,
by Ahuja, Magnanti, and Orlin, Prentice Hall, 1993.

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.

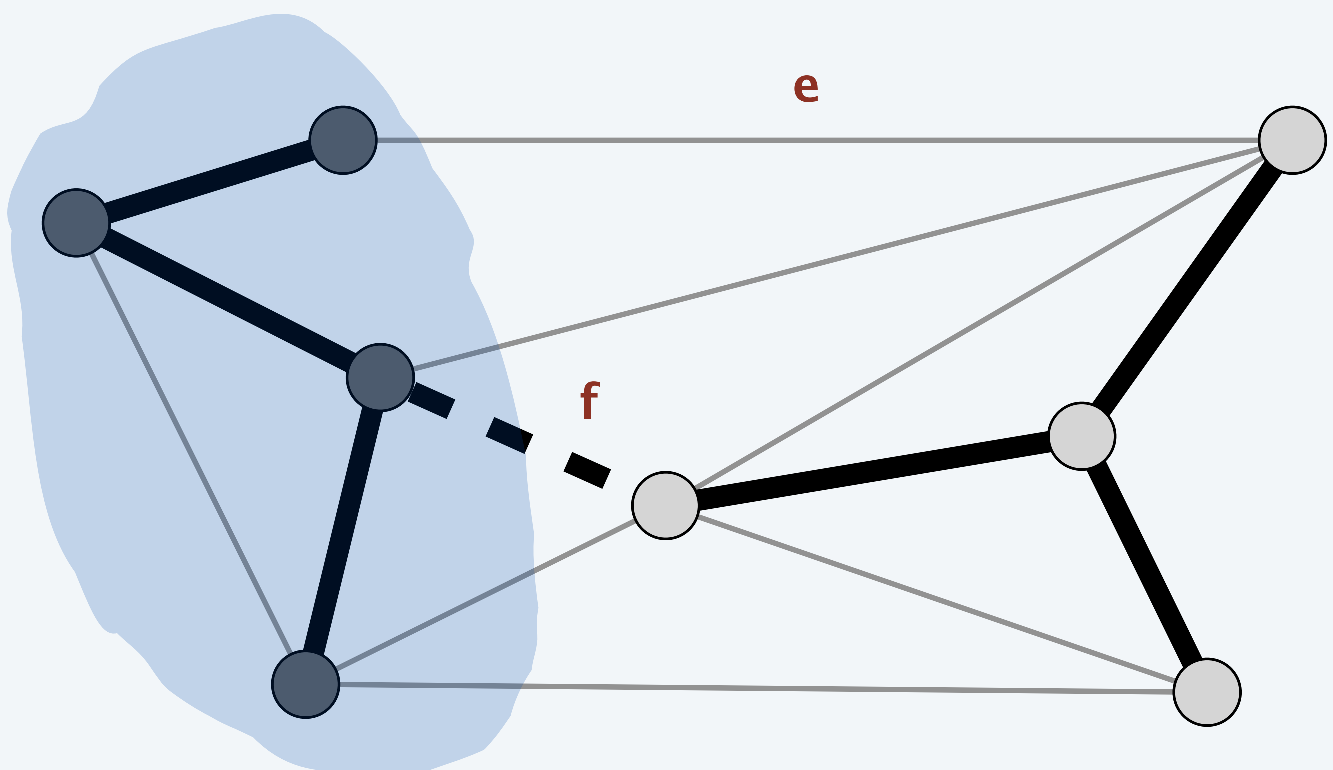


Observation. If $c_e < c_f$, then (V, T) is not an MST.

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.

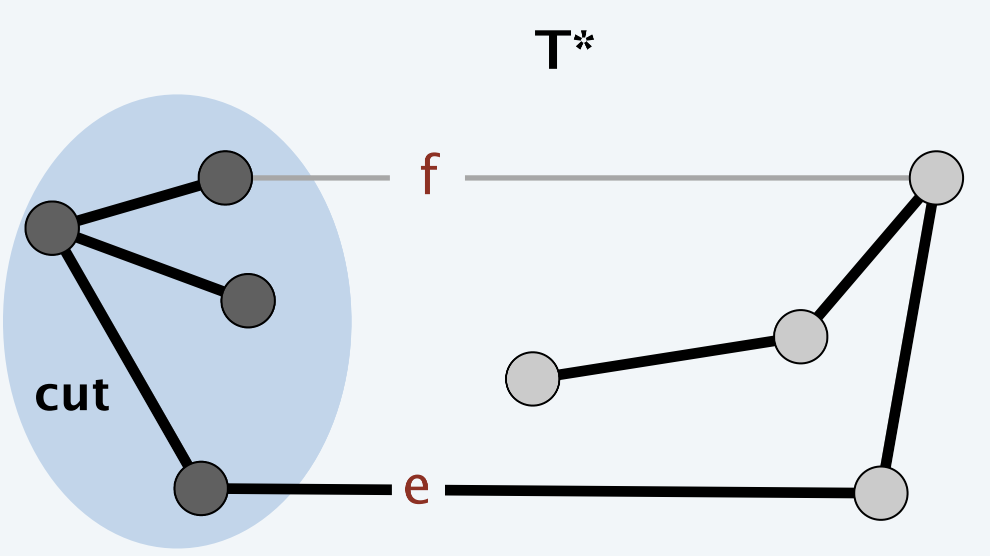
Greedy algorithm: proof of correctness

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.



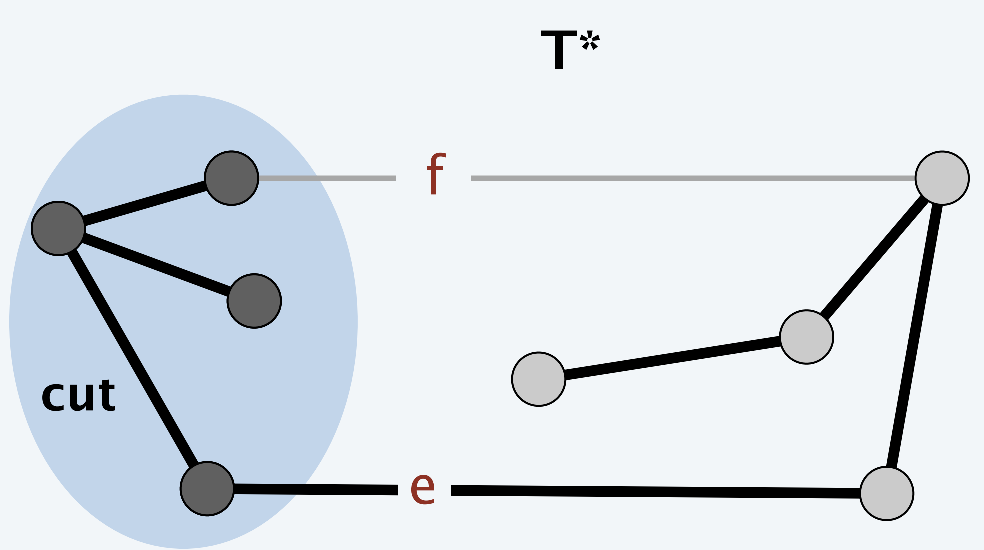
Greedy algorithm: proof of correctness

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. ▪

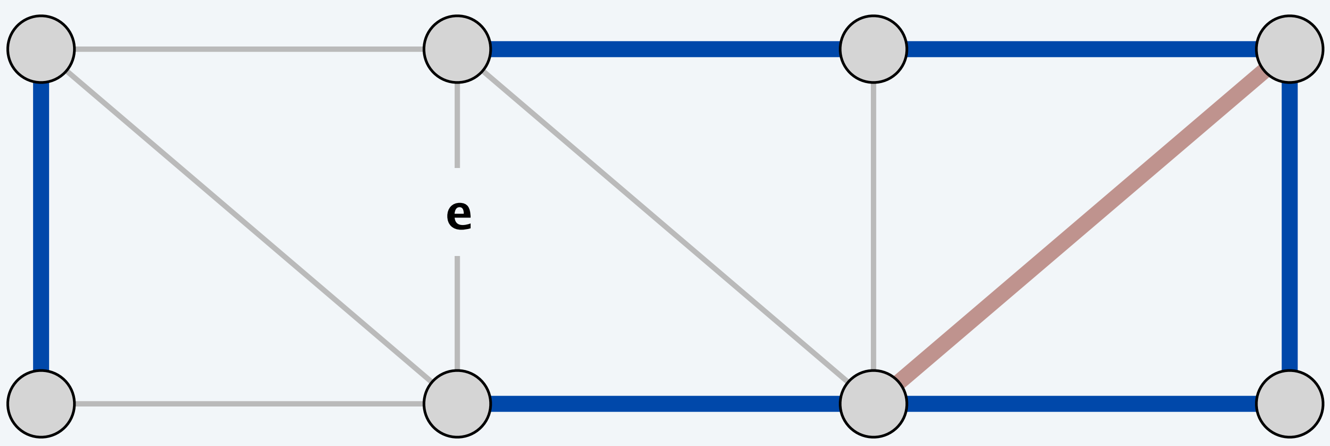


Greedy algorithm: proof of correctness

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

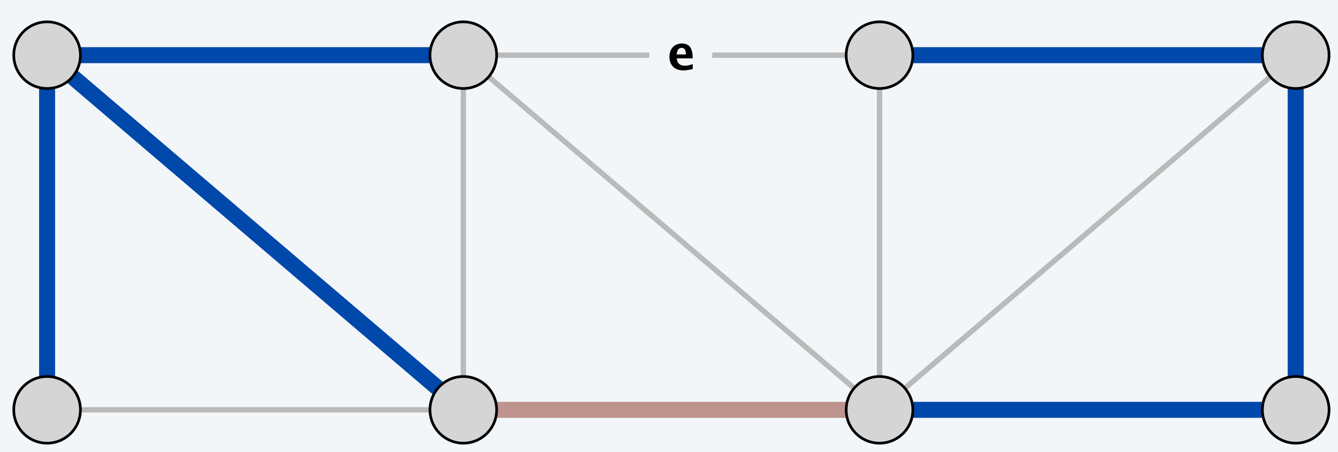
Theorem. The greedy algorithm terminates. Blue edges form an MST.

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

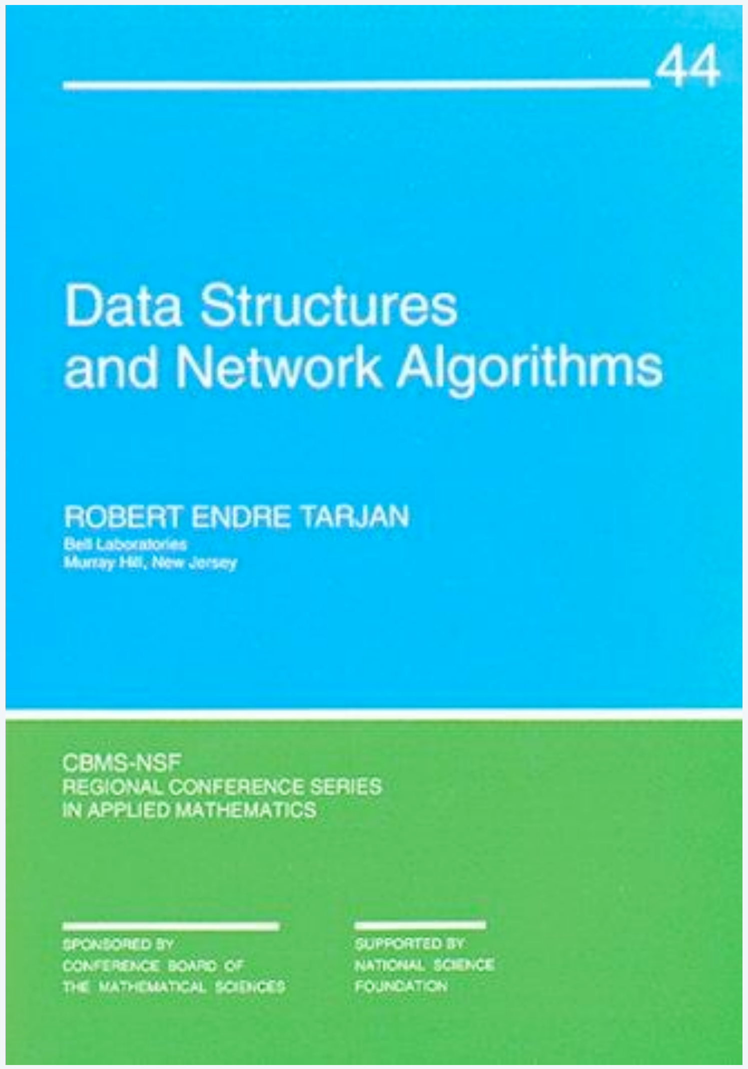
- Suppose edge e is left uncolored.
- Blue edges form a forest.
- Case 1: both endpoints of e are in same blue tree.

  ⇒ apply red rule to cycle formed by adding e to blue forest.
- Case 2: both endpoints of e are in different blue trees.

  ⇒ apply blue rule to cutset induced by either of two blue trees. ▪



Case 2



SECTION 6.2

4. GREEDY ALGORITHMS II

- *Dijkstra's algorithm*
- *minimum spanning trees*
- *Prim, Kruskal, Boruvka*
- *single-link clustering*
- *min-cost arborescences*

Prim's algorithm

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

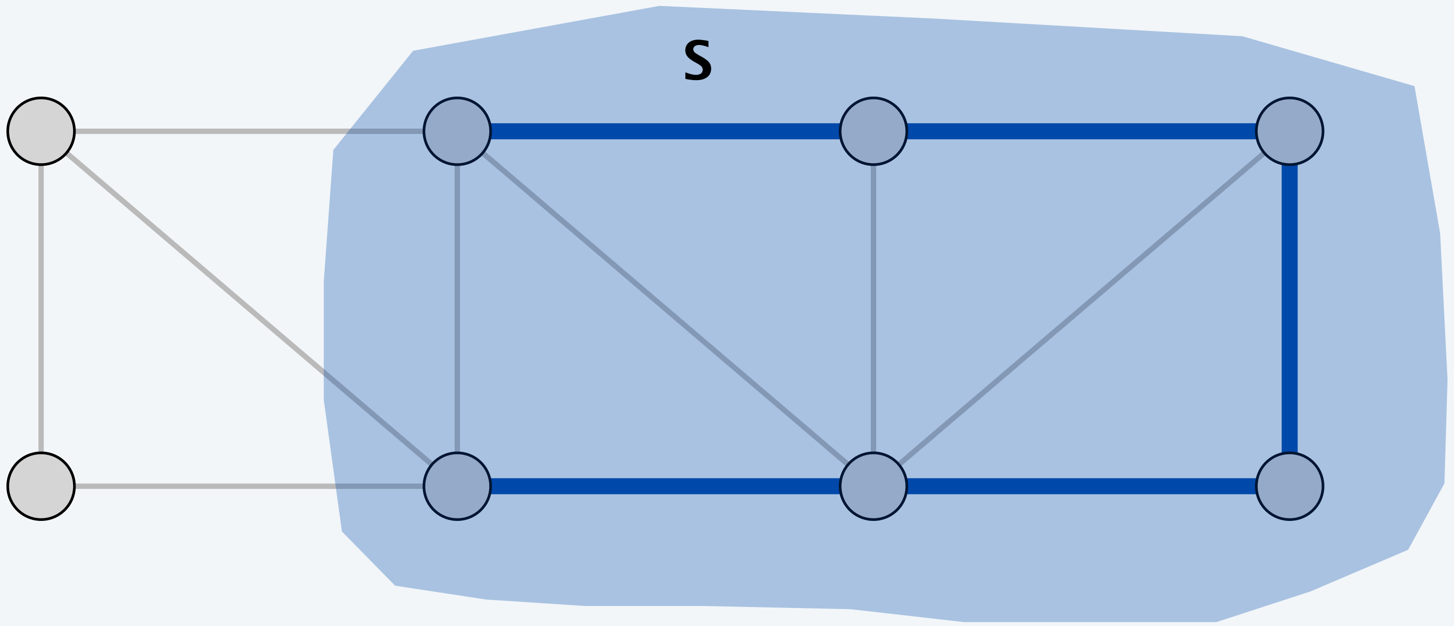
Repeat $n - 1$ times:

- 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

Kruskal's algorithm

Consider edges in ascending order of cost:

- Add to tree unless it would create a cycle.

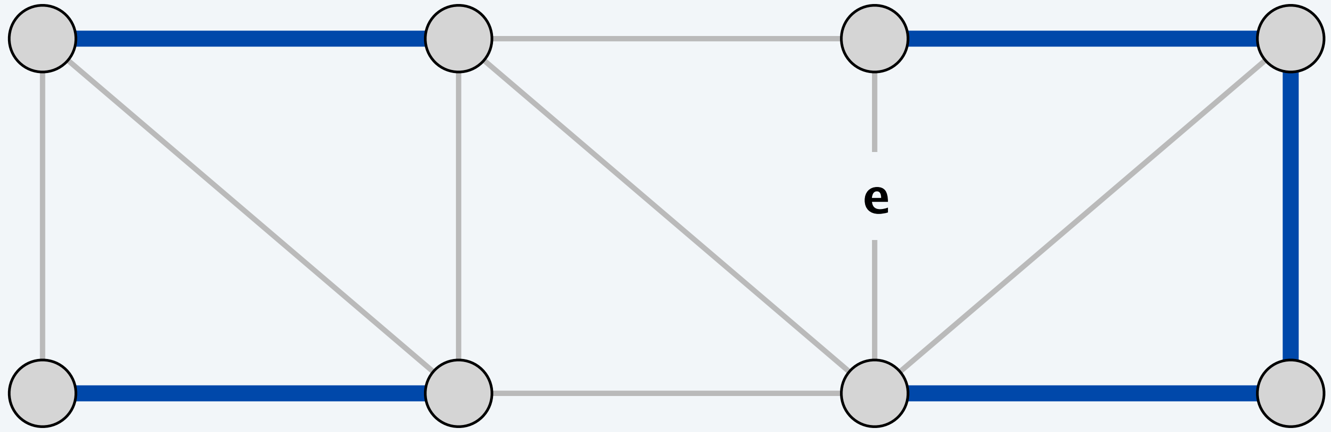
Theorem. Kruskal's algorithm computes an MST.

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 $c(e_1) \leq c(e_2) \leq \ldots \leq c(e_m)$.

$T \leftarrow \varnothing$.

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.

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.

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, ...

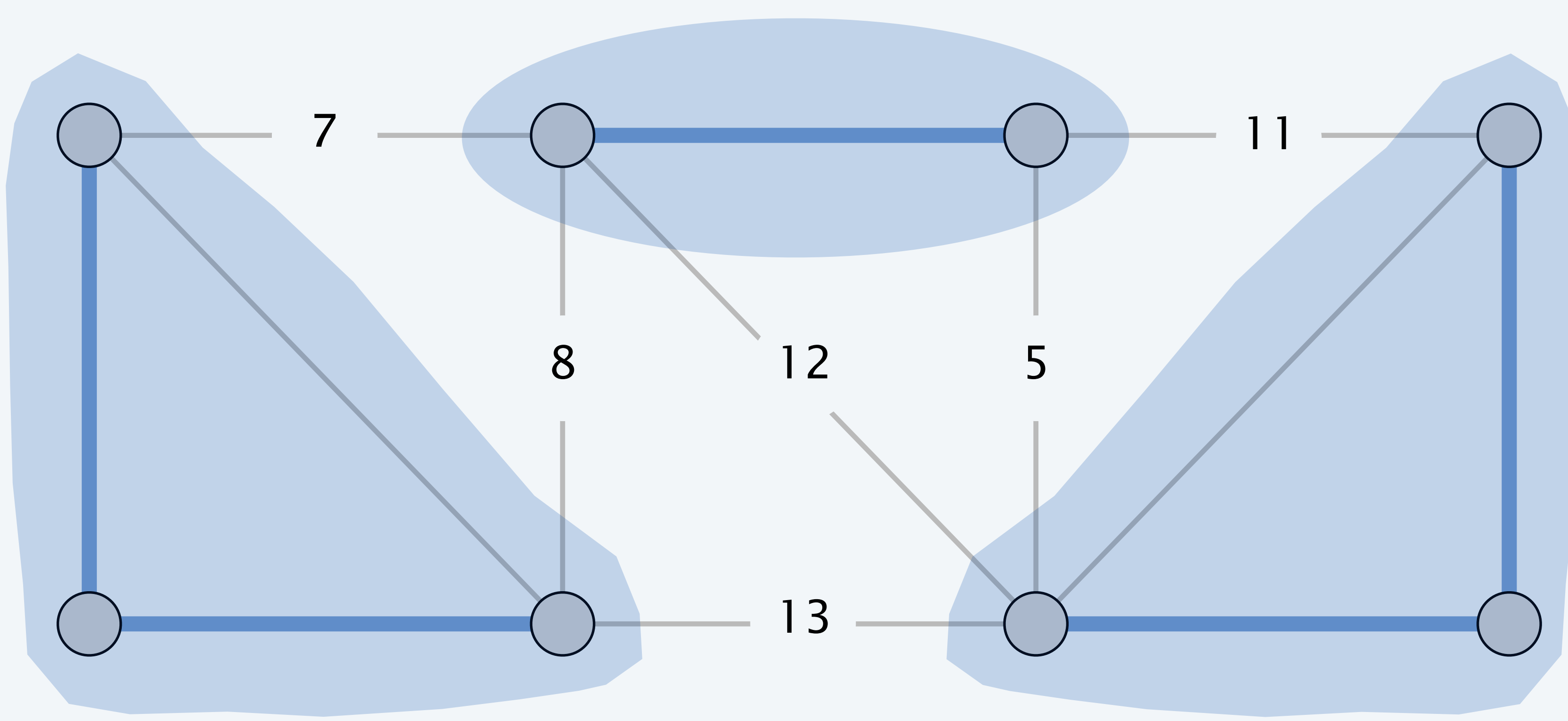
Borůvka's algorithm

Repeat until only one tree.

- Apply blue rule to cutset corresponding to each blue tree.
- Color all selected edges blue.

Theorem. Borůvka's algorithm computes the MST. ← assume edge costs are distinct

Pf. Special case of greedy algorithm (repeatedly apply blue rule). ▪

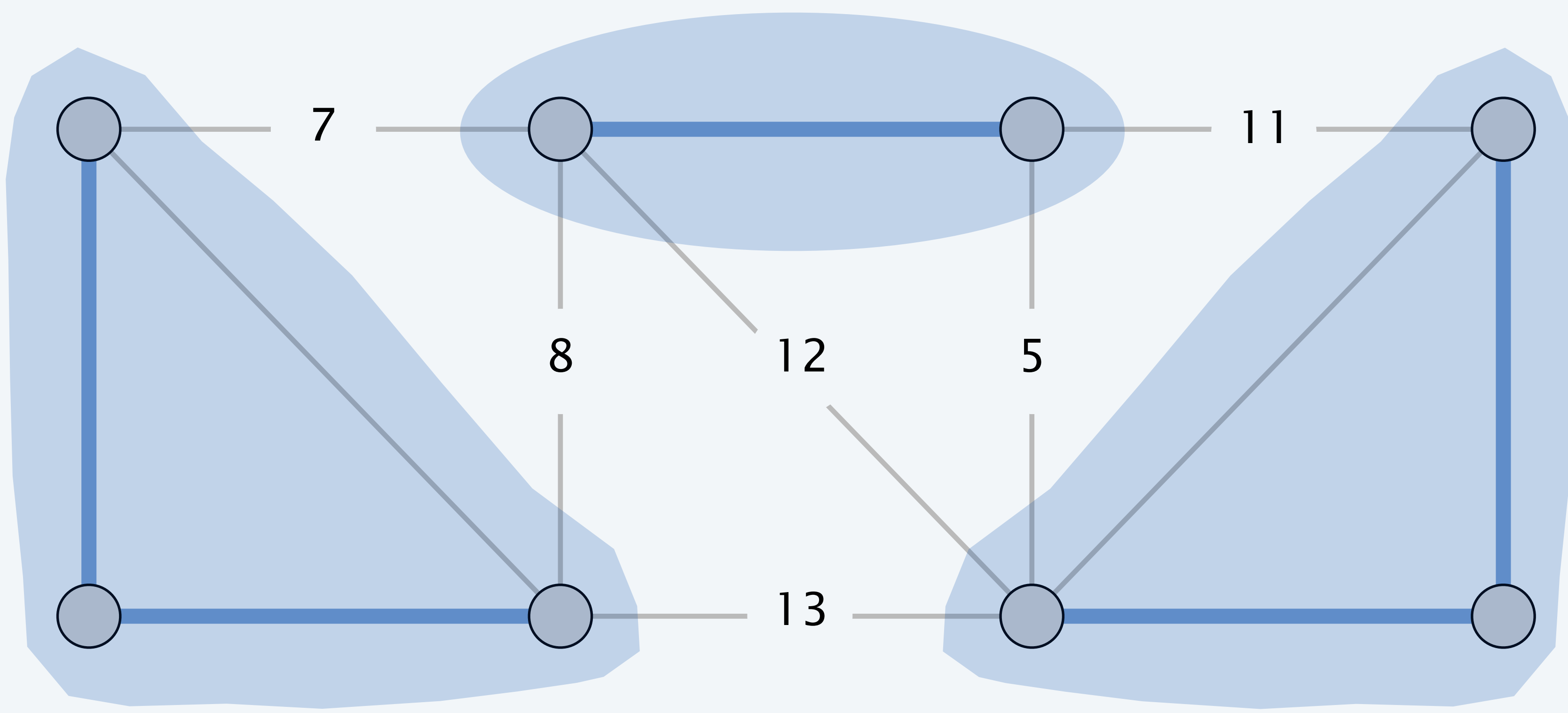


Borůvka's algorithm: implementation

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

Pf.

- To implement a phase in $O(m)$ time:
  - compute connected components of blue edges
  - for each edge $(u, v) \in E$, check if u and v are in different components; if so, update each component's best edge in cutset
- $\leq \log_2 n$ phases since each phase (at least) halves total # components. ▪

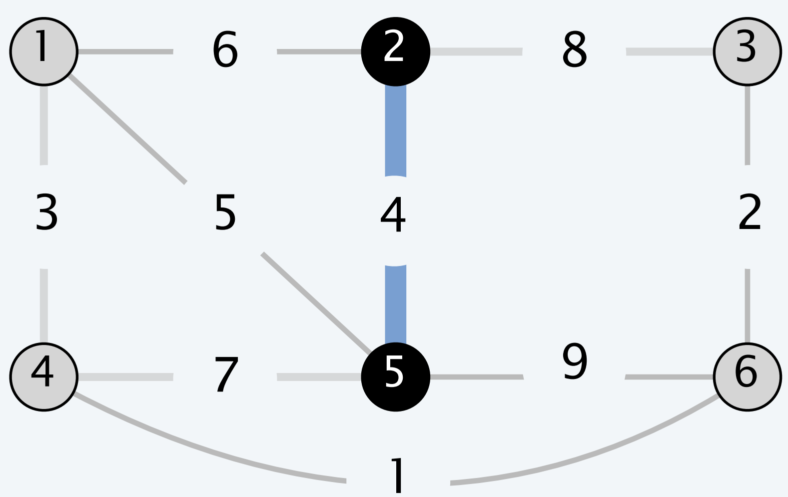


Borůvka's algorithm: implementation

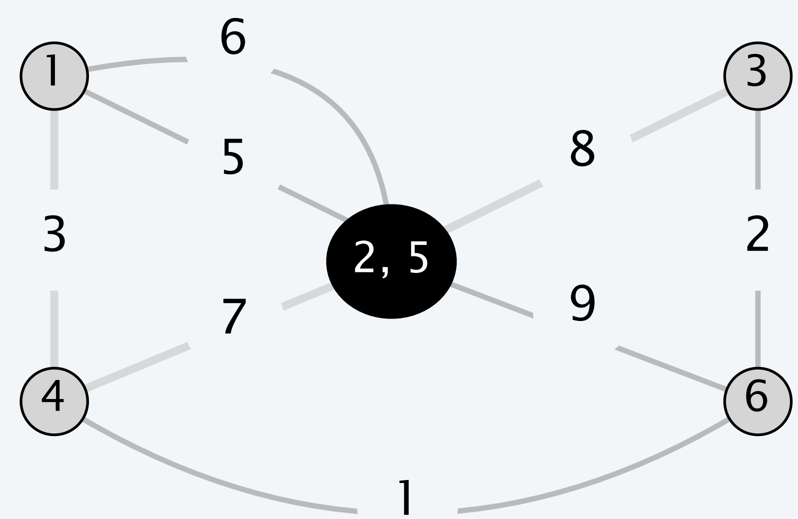
Contraction version.

- After each phase, contract each blue tree to a single supernode.
- Delete self-loops and parallel edges (keeping only cheapest one).
- Borůvka phase becomes: take cheapest edge incident to each node.

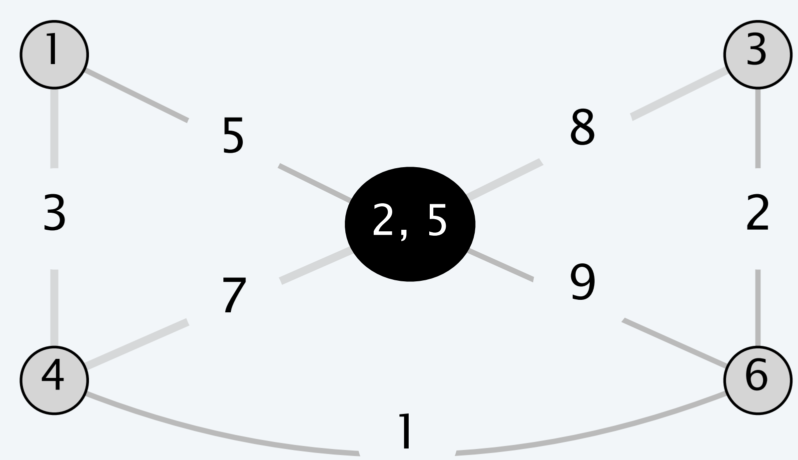
graph G



contract edge 2–5



delete self-loops and parallel edges

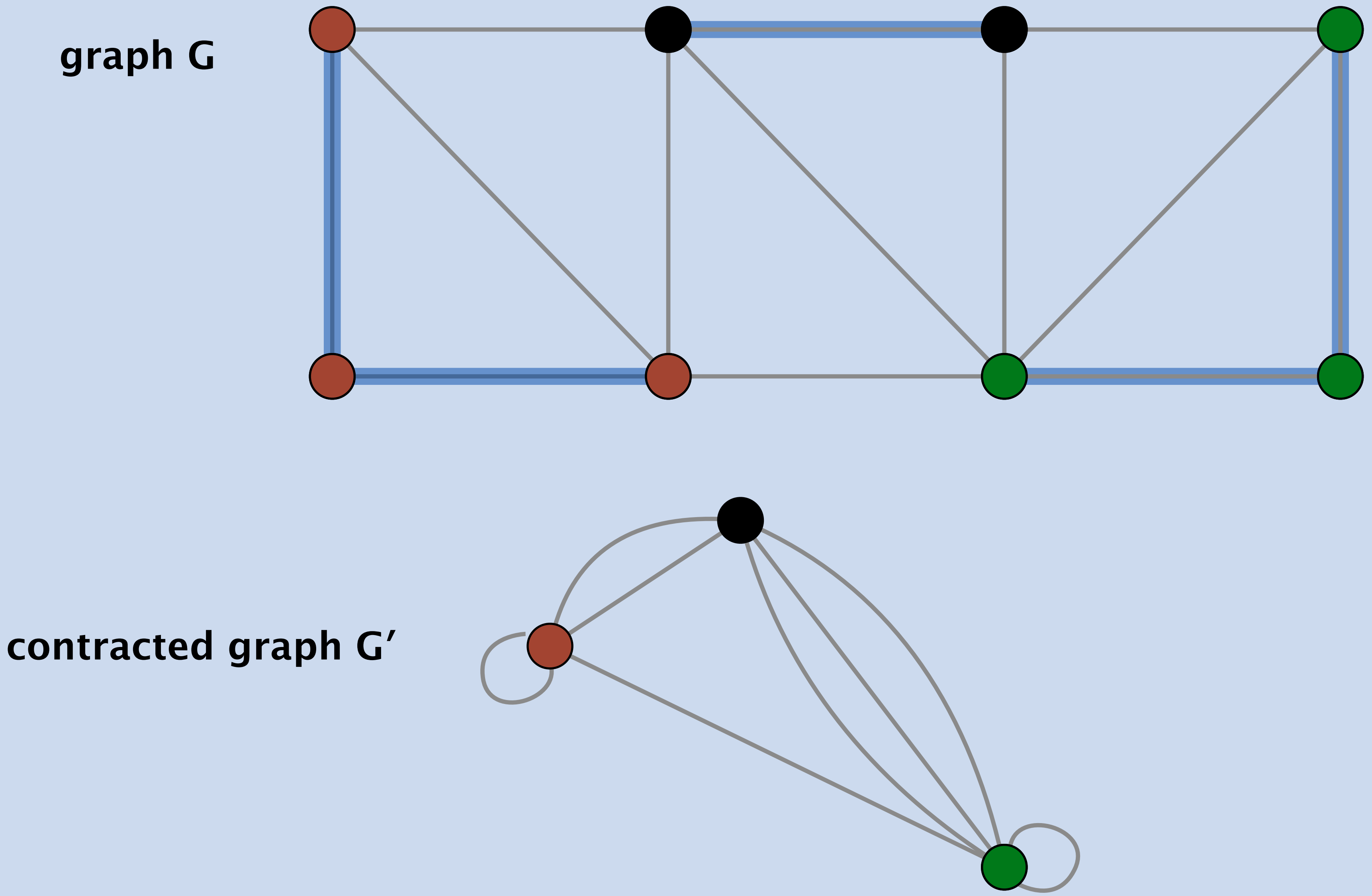


Q. How to contract a set of edges?

CONTRACT A SET OF EDGES

Problem. Given a graph $G = (V, E)$ and a set of edges F, contract all edges in F, removing any self-loops or parallel edges.

Goal. $O(m + n)$ time.



Contract a set of edges

Problem. Given a graph $G = (V, E)$ and a set of edges F, contract all edges in F, removing any self-loops or parallel edges.

Solution.

- Compute the n' connected components in (V, F).
- Suppose $id[u] = i$ means node u is in connected component i.
- The contracted graph G' has n' nodes.
- For each edge $u–v \in E$, add an edge $i–j$ to G', where $i = id[u]$ and $j = id[v]$.

Removing self loops. Easy.

Removing parallel edges.

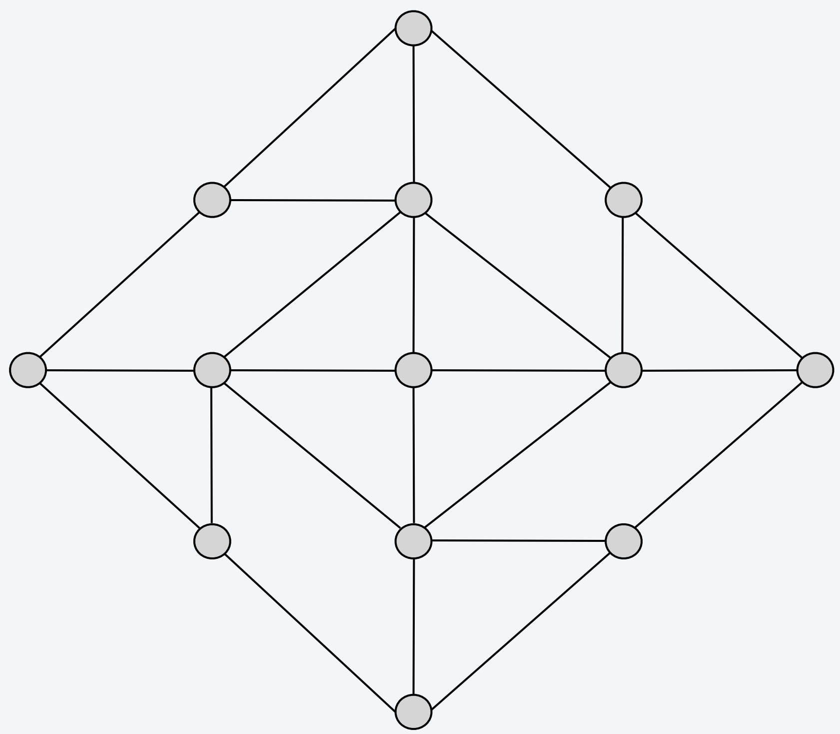
- Create a list of edges $i–j$ with the convention that $i < j$.
- Sort the edges lexicographically via LSD radix sort.
- Add the edges to the graph G', removing parallel edges.

Borůvka's algorithm on planar graphs

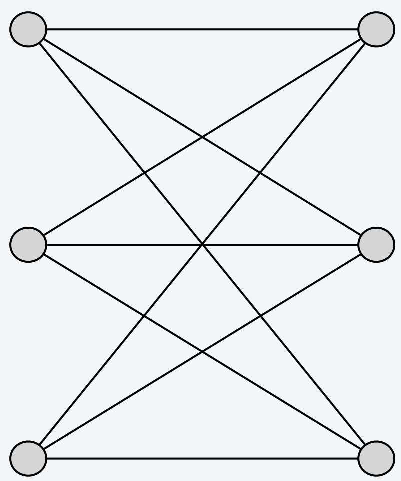
Theorem. Borůvka's algorithm (contraction version) can be implemented to run in $O(n)$ time on planar graphs.

Pf.

- Each Borůvka phase takes $O(n)$ time:
  - Fact 1: $m \le 3n$ for simple planar graphs.
  - Fact 2: planar graphs remains planar after edge contractions/deletions.
- Number of nodes (at least) halves in each phase.
- Thus, overall running time $\le cn + cn/2 + cn/4 + cn/8 + \ldots = O(n)$. ▪



planar



$K_{3,3}$ not planar

A hybrid algorithm

Borůvka–Prim algorithm.

- Run Borůvka (contraction version) for $\log_2 \log_2 n$ phases.
- Run Prim on resulting, contracted graph.

Theorem. Borůvka–Prim computes an MST.

Pf. Special case of the greedy algorithm.

Theorem. Borůvka–Prim can be implemented to run in $O(m \log \log n)$ time.

Pf.

- The $\log_2 \log_2 n$ phases of Borůvka's algorithm take $O(m \log \log n)$ time; resulting graph has $\le n / \log_2 n$ nodes and $\le m$ edges.
- Prim's algorithm (using Fibonacci heaps) takes $O(m + n)$ time on a graph with $n / \log_2 n$ nodes and m edges. ▪

$$O\left(m + \frac{n}{\log n} \log\left(\frac{n}{\log n}\right)\right)$$

Does a linear-time compare-based MST algorithm exist?

year	worst case	discovered by
1975	$O(m \log \log n)$	Yao
1976	$O(m \log \log n)$	Cheriton–Tarjan
1984	$O(m \log^* n)$, $O(m + n \log n)$	Fredman–Tarjan
1986	$O(m \log (\log^* n))$	Gabow–Galil–Spencer–Tarjan
1997	$O(m\, \alpha(n) \log \alpha(n))$	Chazelle
2000	$O(m\, \alpha(n))$	Chazelle
2002	*asymptotically optimal*	Pettie–Ramachandran
20xx	$O(m)$	

deterministic compare-based MST algorithms

iterated logarithm function

$$\lg^* n = \begin{cases} 0 & \text{if } n \le 1 \\ 1 + \lg^*(\lg n) & \text{if } n > 1 \end{cases}$$

n	$\lg^* n$
$(-\infty, 1]$	0
$(1, 2]$	1
$(2, 4]$	2
$(4, 16]$	3
$(16, 2^{16}]$	4
$(2^{16}, 2^{65536}]$	5

Theorem. [Fredman–Willard 1990] $O(m)$ in word RAM model.

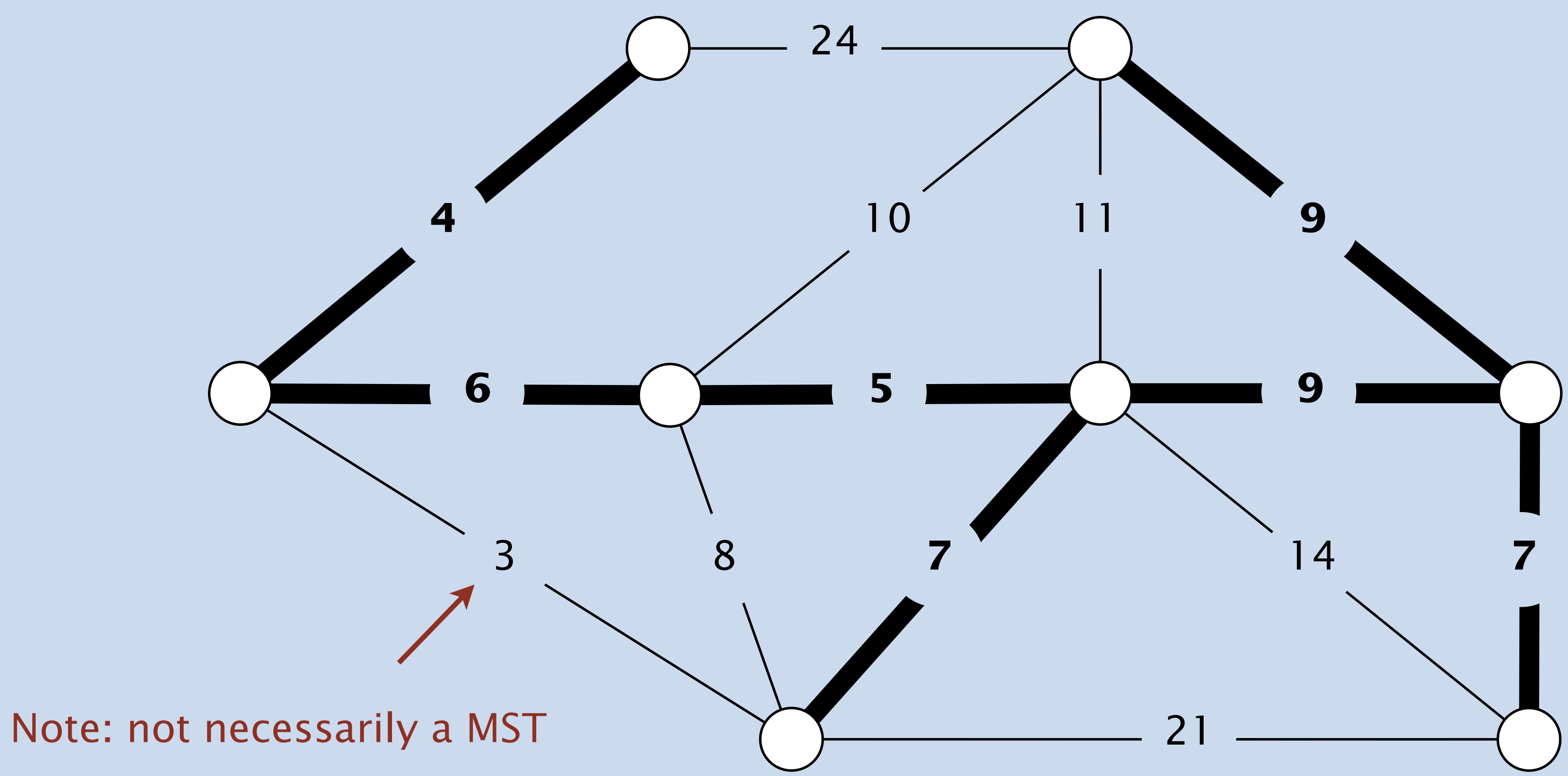
Theorem. [Dixon–Rauch–Tarjan 1992] $O(m)$ MST verification algorithm.

Theorem. [Karger–Klein–Tarjan 1995] $O(m)$ randomized MST algorithm.

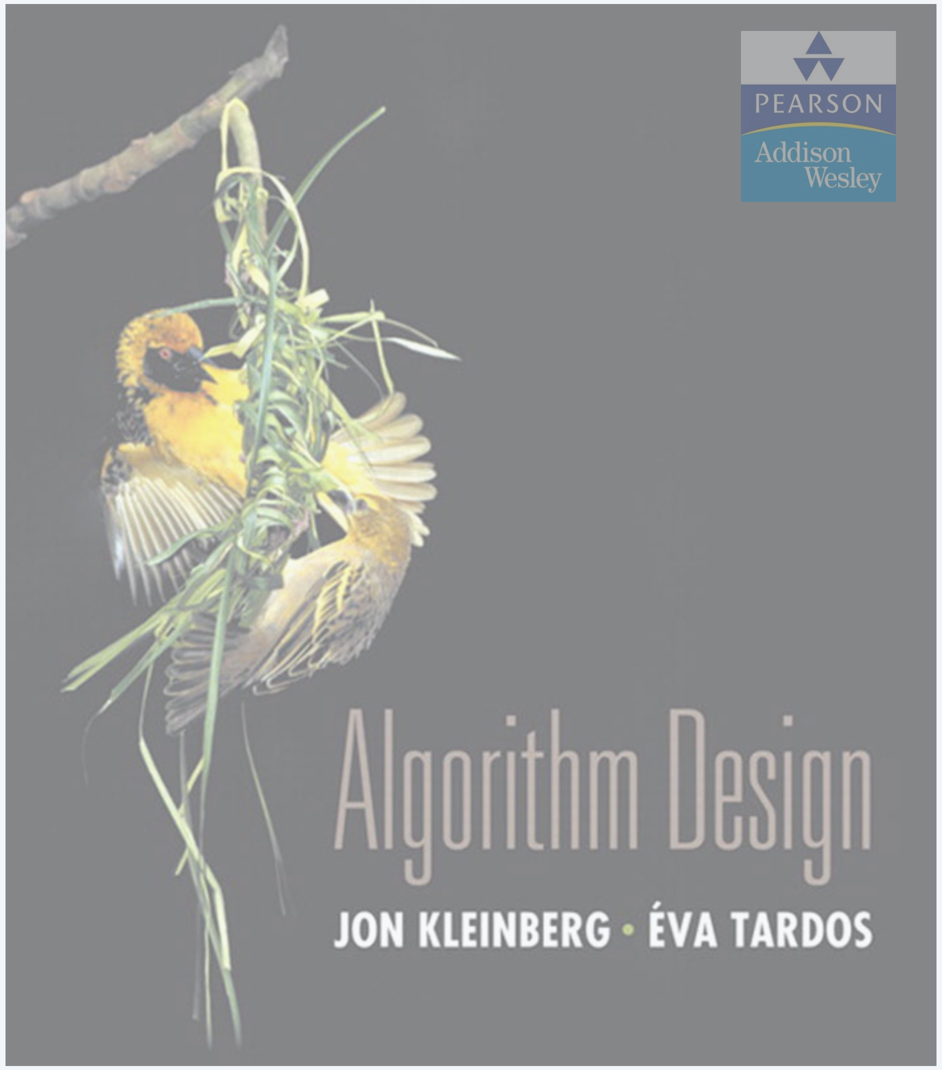
Minimum Bottleneck Spanning Tree

Problem. Given a connected graph G with positive edge costs, find a spanning tree that minimizes the most expensive edge.

Goal. $O(m \log m)$ time or better.



minimum bottleneck spanning tree T (bottleneck = 9)



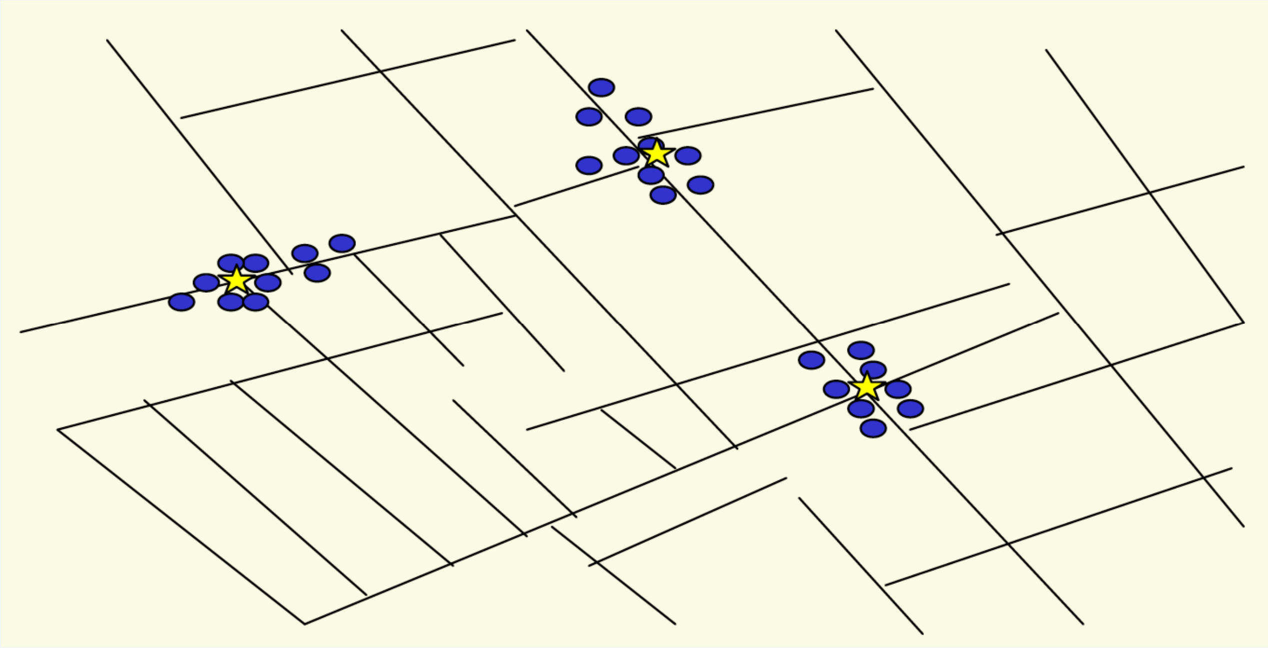
SECTION 4.7

4. GREEDY ALGORITHMS II

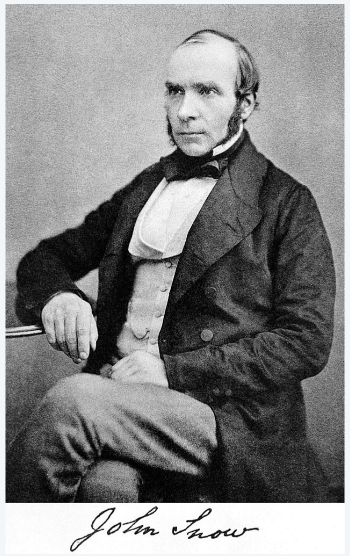
- *Dijkstra's algorithm*
- *minimum spanning trees*
- *Prim, Kruskal, Boruvka*
- *single-link clustering*
- *min-cost arborescences*

Clustering

Goal. Given a set U of n objects labeled $p_1, \ldots, p_n$, partition into clusters so that objects in different clusters are far apart.



outbreak of cholera deaths in London in 1850s (Nina Mishra)



Applications.

- Routing in mobile ad-hoc networks.
- Document categorization for web search.
- Similarity searching in medical image databases
- Cluster celestial objects into stars, quasars, galaxies.
- ...

Clustering of maximum spacing

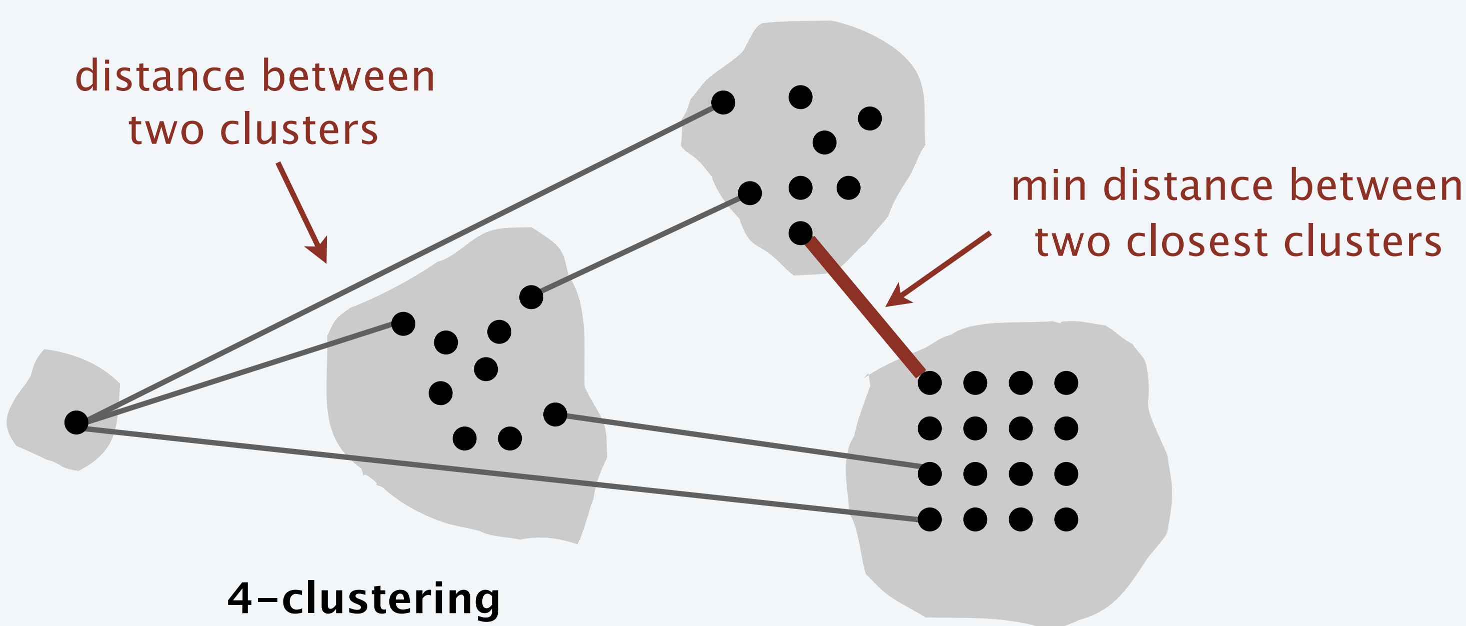
k-clustering. Divide objects into k non-empty groups.

Distance function. Numeric value specifying "closeness" of two objects.

- $d(p_i, p_j) = 0$ iff $p_i = p_j$ [identity of indiscernibles]
- $d(p_i, p_j) \geq 0$ [non-negativity]
- $d(p_i, p_j) = d(p_j, p_i)$ [symmetry]

Spacing. Min distance between any pair of points in different clusters.

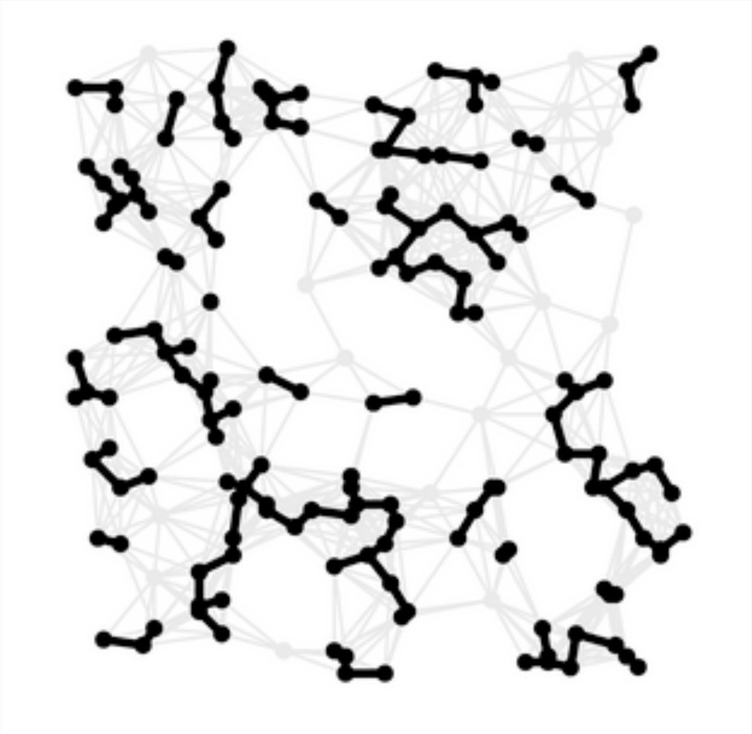
Goal. Given an integer k, find a k-clustering of maximum spacing.



Greedy clustering algorithm

"Well-known" algorithm in science literature for single-linkage k-clustering:

- Form a graph on the node set U, corresponding to n clusters.
- Find the closest pair of objects such that each object is in a different cluster, and add an edge between them.
- Repeat $n - k$ times (until there are exactly k clusters).



Key observation. This procedure is precisely Kruskal's algorithm (except we stop when there are k connected components).

Alternative. Find an MST and delete the $k - 1$ longest edges.

Greedy clustering algorithm: analysis

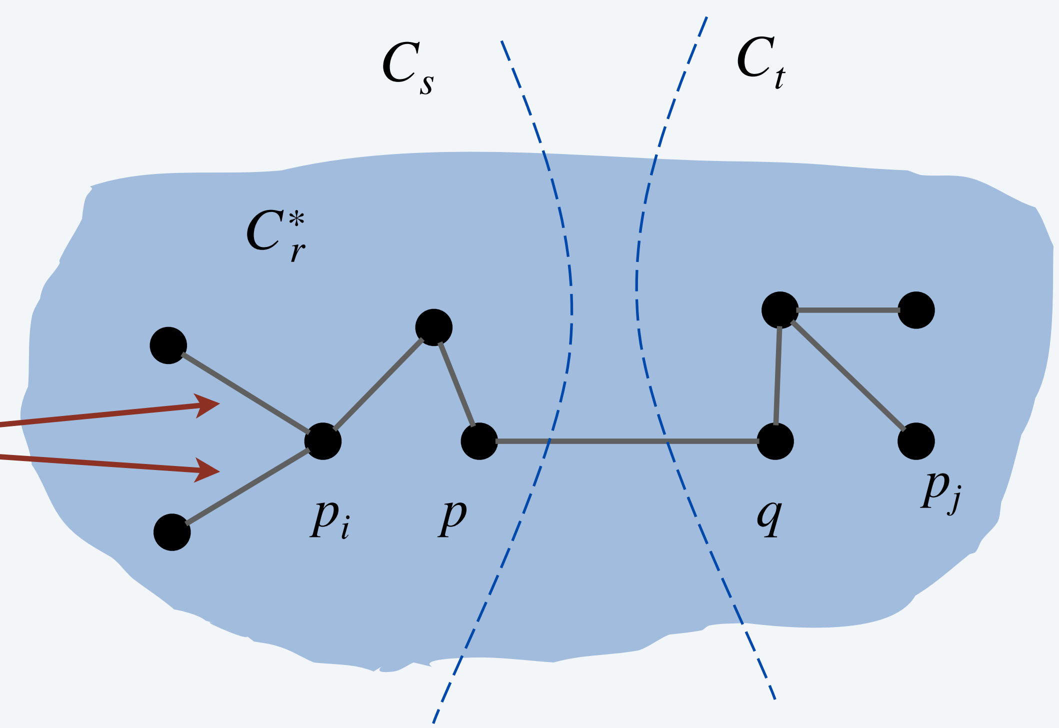
Theorem. Let C^* denote the clustering $C^*_1, \ldots, C^*_k$ formed by deleting the $k-1$ longest edges of an MST. Then, C^* is a k-clustering of max spacing.

Pf.

- Let C denote any other clustering $C_1, \ldots, C_k$.
- Let p_i and p_j be in the same cluster in C^*, say C^*_r, but different clusters in C, say C_s and C_t.
- Some edge (p, q) on $p_i - p_j$ path in C^*_r spans two different clusters in C.
- Spacing of C^* = length d^* of the $(k-1)^{\text{st}}$ longest edge in MST.
- Edge (p, q) has length $\leq d^*$ since it was added by Kruskal.
- Spacing of C is $\leq d^*$ since p and q are in different clusters. ▪

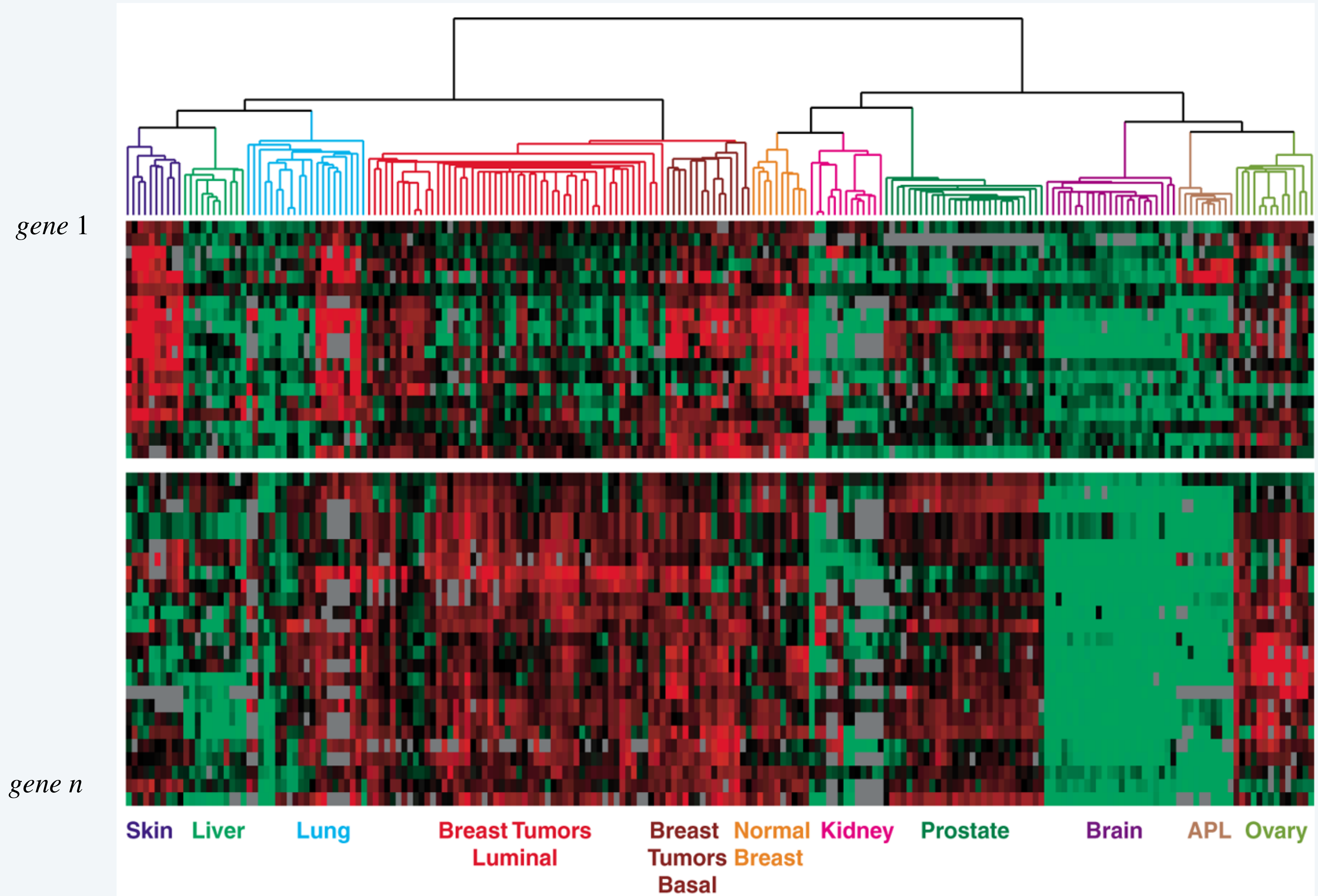
this is the edge Kruskal would have added next had we not stopped it

edges left after deleting $k-1$ longest edges from a MST



Dendrogram of cancers in human

Tumors in similar tissues cluster together.



Reference: Botstein & Brown group

gene expressed

gene not expressed

Which MST algorithm should you use for single-link clustering?

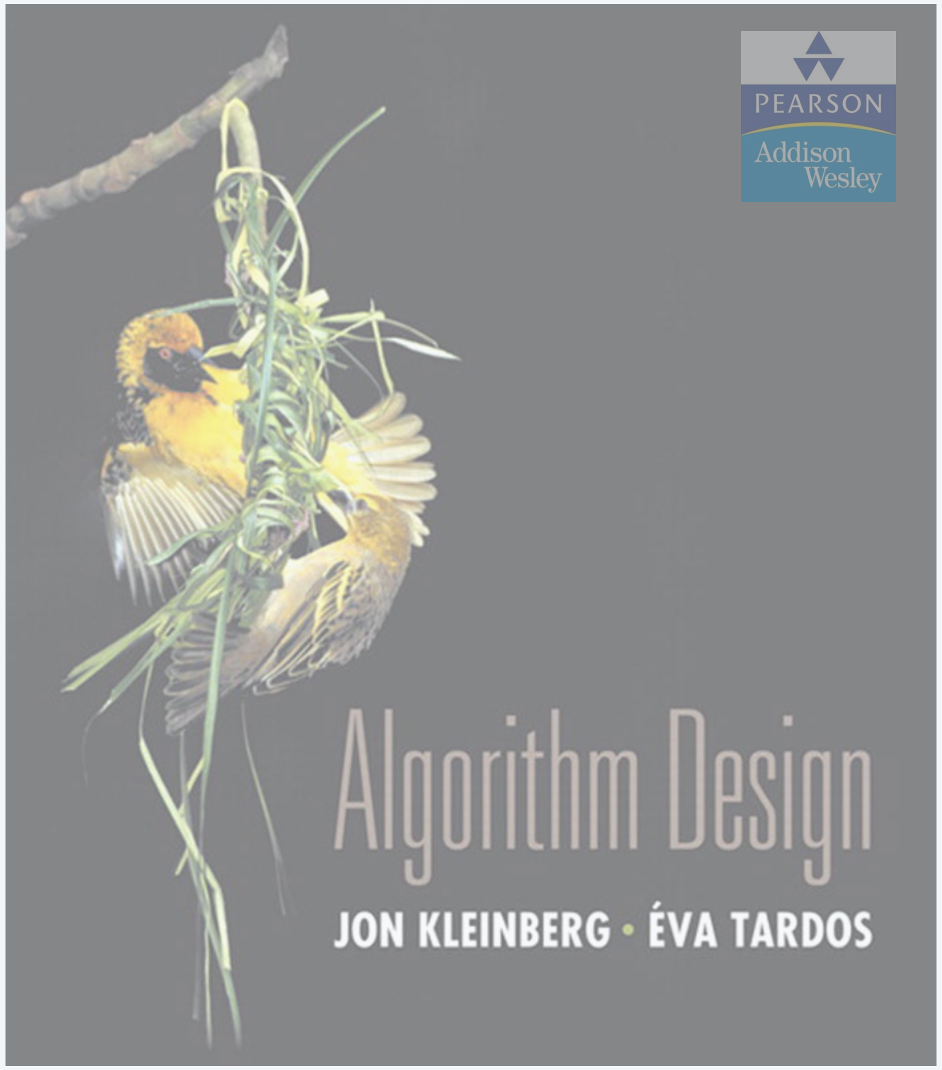
number of objects n
can be very large

A. Kruskal (stop when there are k components).

B. Prim (delete $k - 1$ longest edges).

C. Either A or B.

D. Neither A nor B.



SECTION 4.9

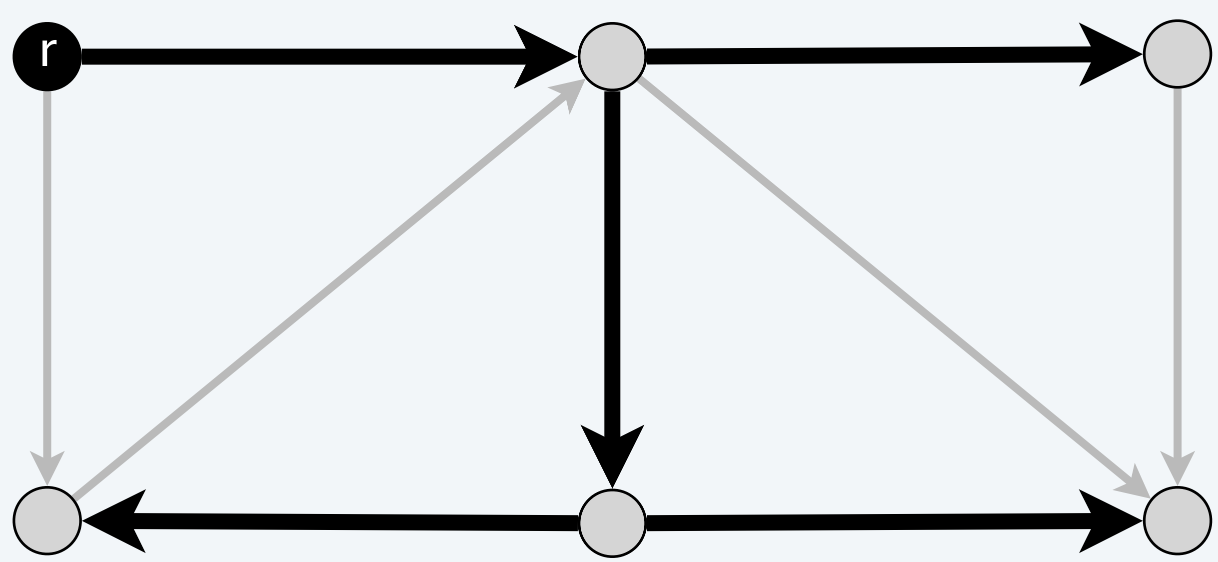
4. GREEDY ALGORITHMS II

- *Dijkstra's algorithm*
- *minimum spanning trees*
- *Prim, Kruskal, Boruvka*
- *single-link clustering*
- *min-cost arborescences*

Arborescences

Def. Given a digraph $G = (V, E)$ and a root $r \in V$, an arborescence (rooted at r) is a subgraph $T = (V, F)$ such that

- T is a spanning tree of G if we ignore the direction of edges.
- There is a (unique) directed path in T from r to each other node $v \in V$.



Minimum spanning arborescence: quiz 1

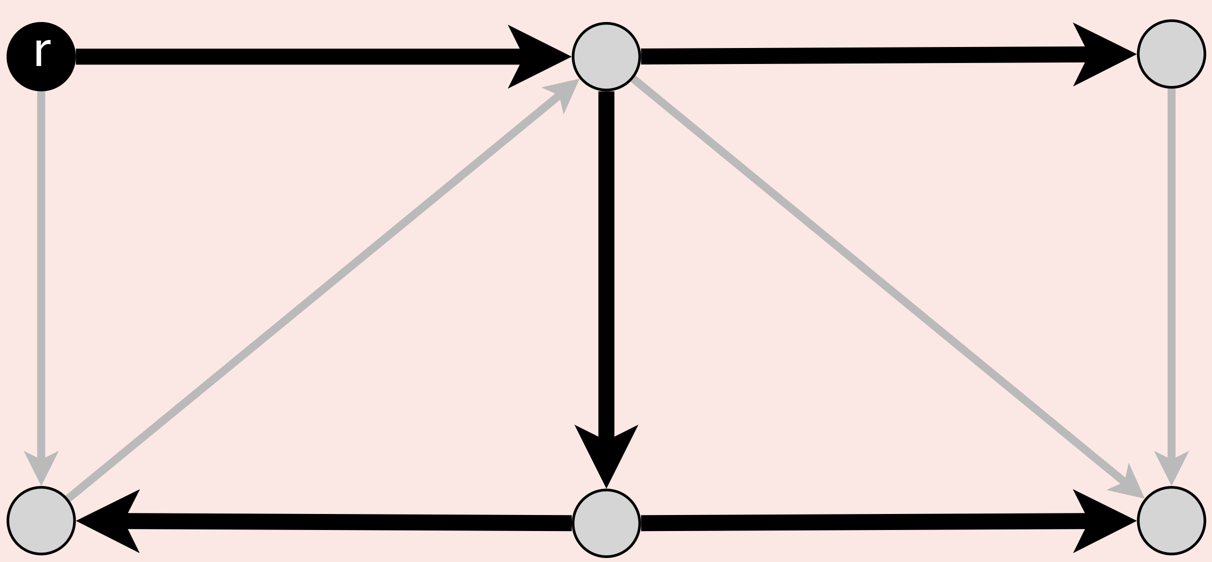
Which of the following are properties of arborescences rooted at r?

A. No directed cycles.

B. Exactly $n - 1$ edges.

C. For each $v \neq r$: $indegree(v) = 1$.

D. All of the above.



Arborescences

Proposition. A subgraph $T = (V, F)$ of G is an arborescence rooted at r iff T has no directed cycles and each node $v \neq r$ has exactly one entering edge.

Pf.

$\Rightarrow$ If T is an arborescence, then no (directed) cycles and every node $v \neq r$ has exactly one entering edge—the last edge on the unique $r \leadsto v$ path.

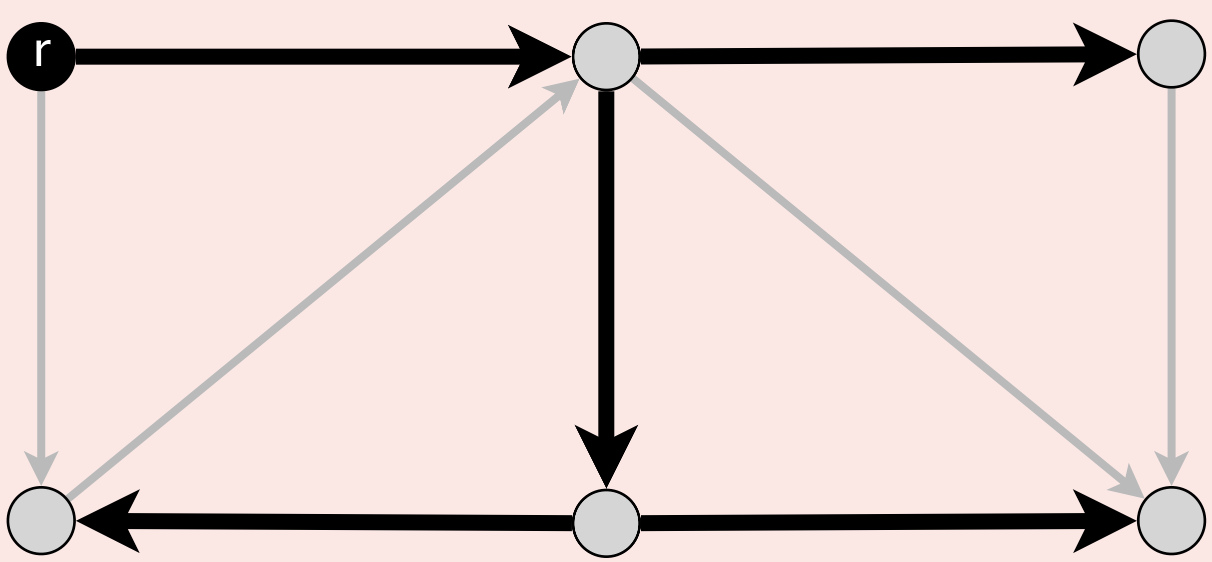
$\Leftarrow$ Suppose T has no cycles and each node $v \neq r$ has one entering edge.

- To construct an $r \leadsto v$ path, start at v and repeatedly follow edges in the backward direction.
- Since T has no directed cycles, the process must terminate.
- It must terminate at r since r is the only node with no entering edge. ▪

Minimum spanning arborescence: quiz 2

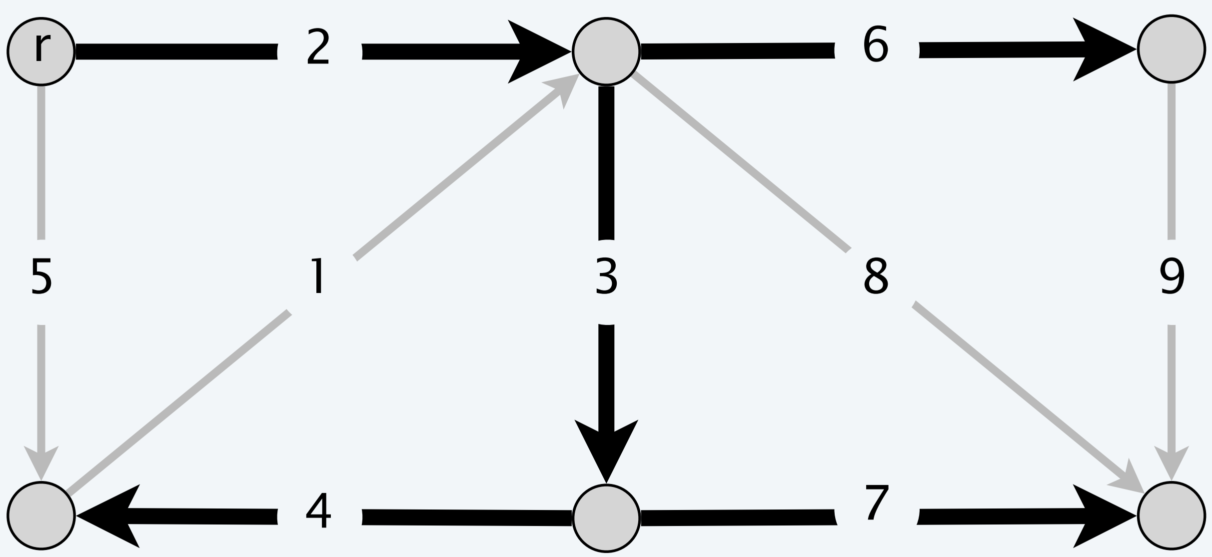
Given a digraph G, how to find an arborescence rooted at r?

- **A.** Breadth-first search from r.
- **B.** Depth-first search from r.
- **C.** Either A or B.
- **D.** Neither A nor B.



Min-cost arborescence problem

Problem. Given a digraph G with a root node r and edge costs $c_e \geq 0$, find an arborescence rooted at r of minimum cost.



Assumption 1. All nodes reachable from r.

Assumption 2. No edge enters r (safe to delete since they won't help).

Minimum spanning arborescence: quiz 3

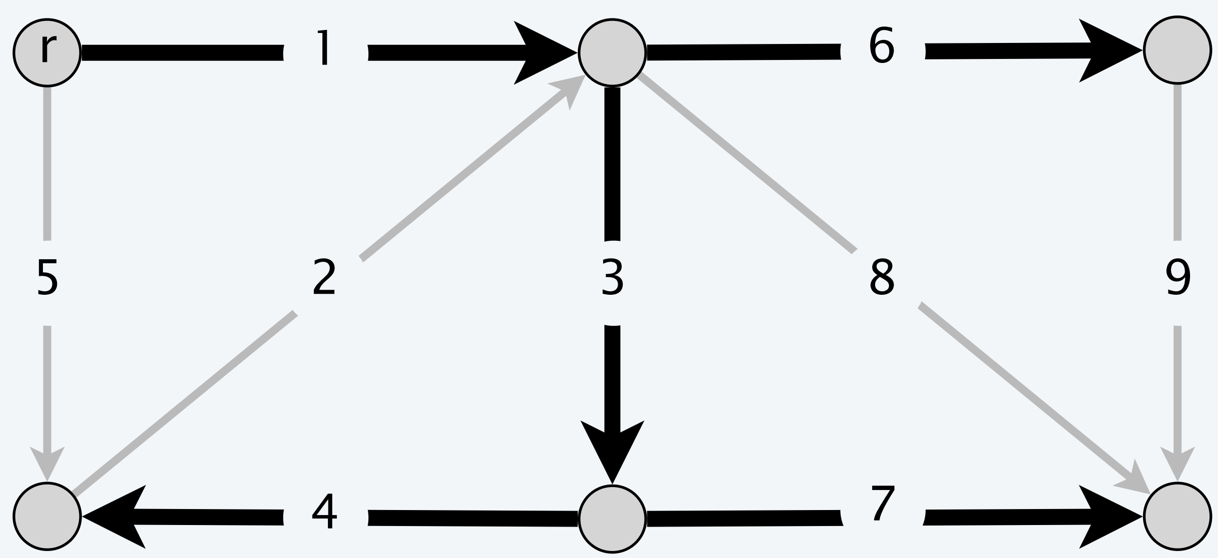
A min-cost arborescence must…

- **A.** Include the cheapest edge.
- **B.** Exclude the most expensive edge.
- **C.** Be a shortest-paths tree from r.
- **D.** None of the above.

A sufficient optimality condition

Property. For each node $v \neq r$, choose a cheapest edge entering v and let F^* denote this set of $n - 1$ edges. If (V, F^*) is an arborescence, then it is a min-cost arborescence.

Pf. An arborescence needs exactly one edge entering each node $v \neq r$ and (V, F^*) is the cheapest way to make each of these choices. ▪

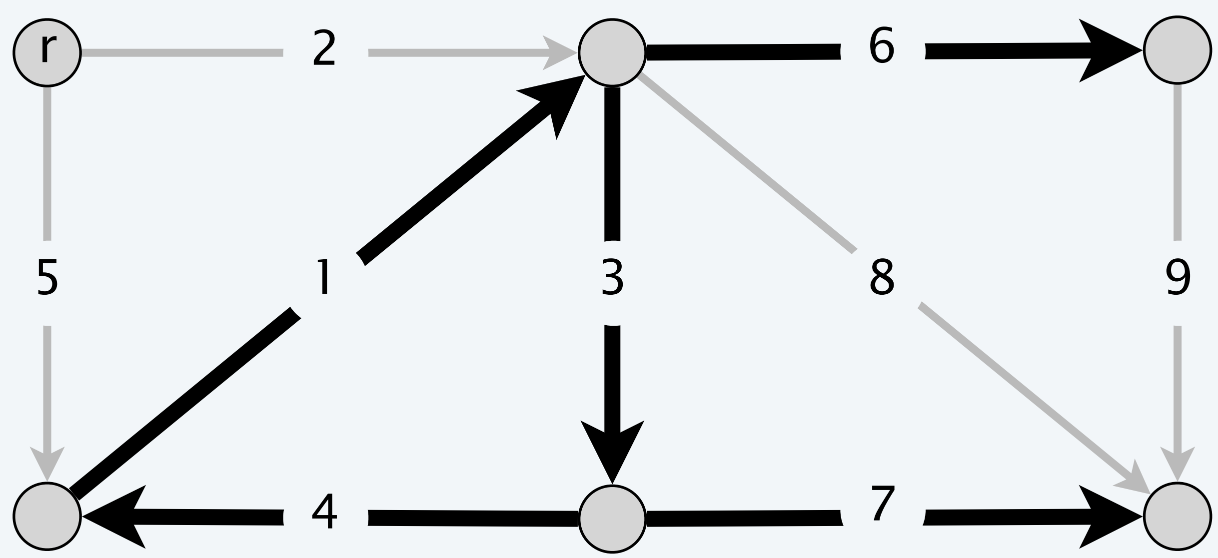


F* = thick black edges

A sufficient optimality condition

Property. For each node $v \neq r$, choose a cheapest edge entering v and let F^* denote this set of $n - 1$ edges. If (V, F^*) is an arborescence, then it is a min-cost arborescence.

Note. F^* may not be an arborescence (since it may have directed cycles).

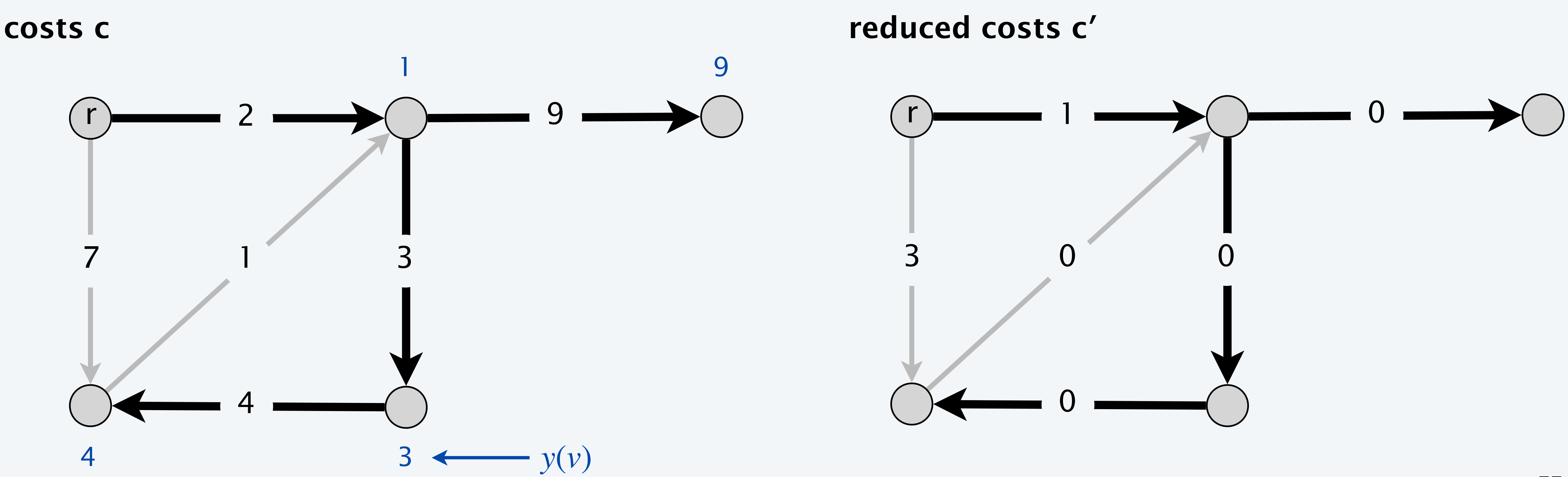


F* = thick black edges

Reduced costs

Def. For each $v \neq r$, let $y(v)$ denote the min cost of any edge entering v.
Define the reduced cost of an edge (u, v) as $c'(u, v) = c(u, v) - y(v) \geq 0$.

Observation. T is a min-cost arborescence in G using costs c iff
T is a min-cost arborescence in G using reduced costs c'.
Pf. For each $v \neq r$: each arborescence has exactly one edge entering v. ▪

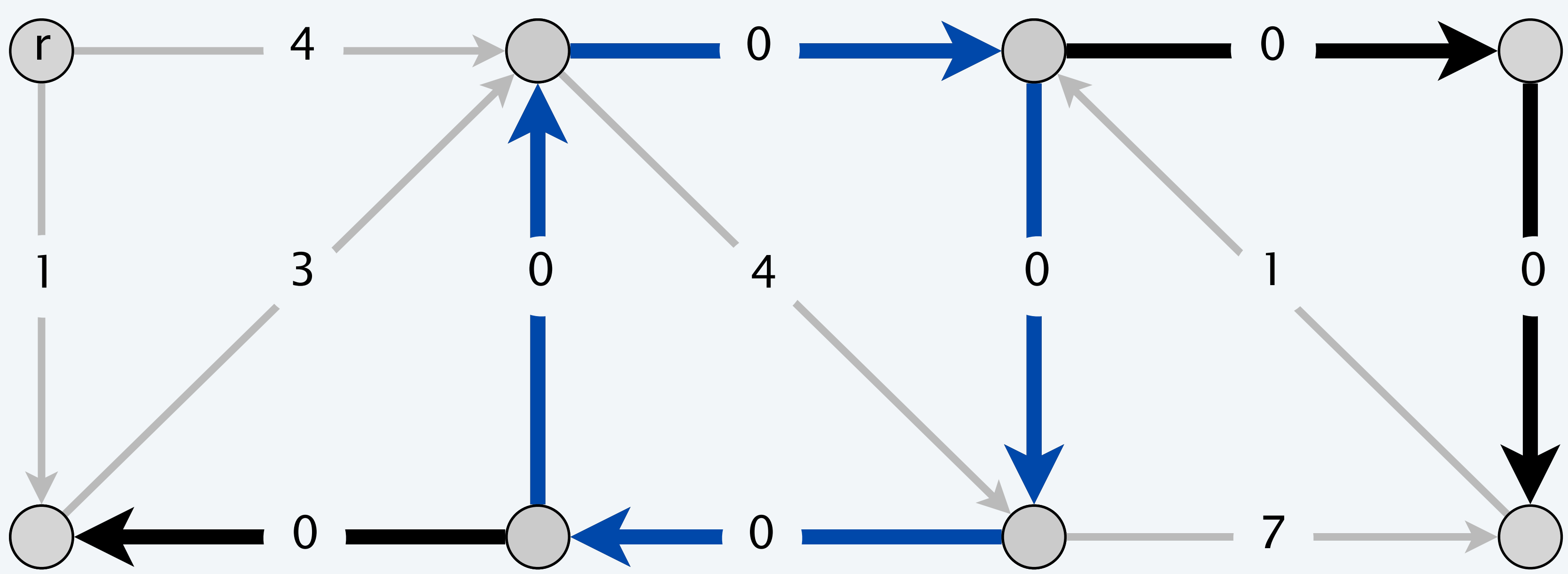


Edmonds branching algorithm: intuition

Intuition. Recall F^* = set of cheapest edges entering v for each $v \neq r$.

- Now, all edges in F^* have 0 cost with respect to reduced costs $c'(u, v)$.
- If F^* does not contain a cycle, then it is a min-cost arborescence.
- If F^* contains a cycle C, can afford to use as many edges in C as desired.
- Contract edges in C to a supernode (removing any self-loops).
- Recursively solve problem in contracted network G' with costs $c'(u, v)$.

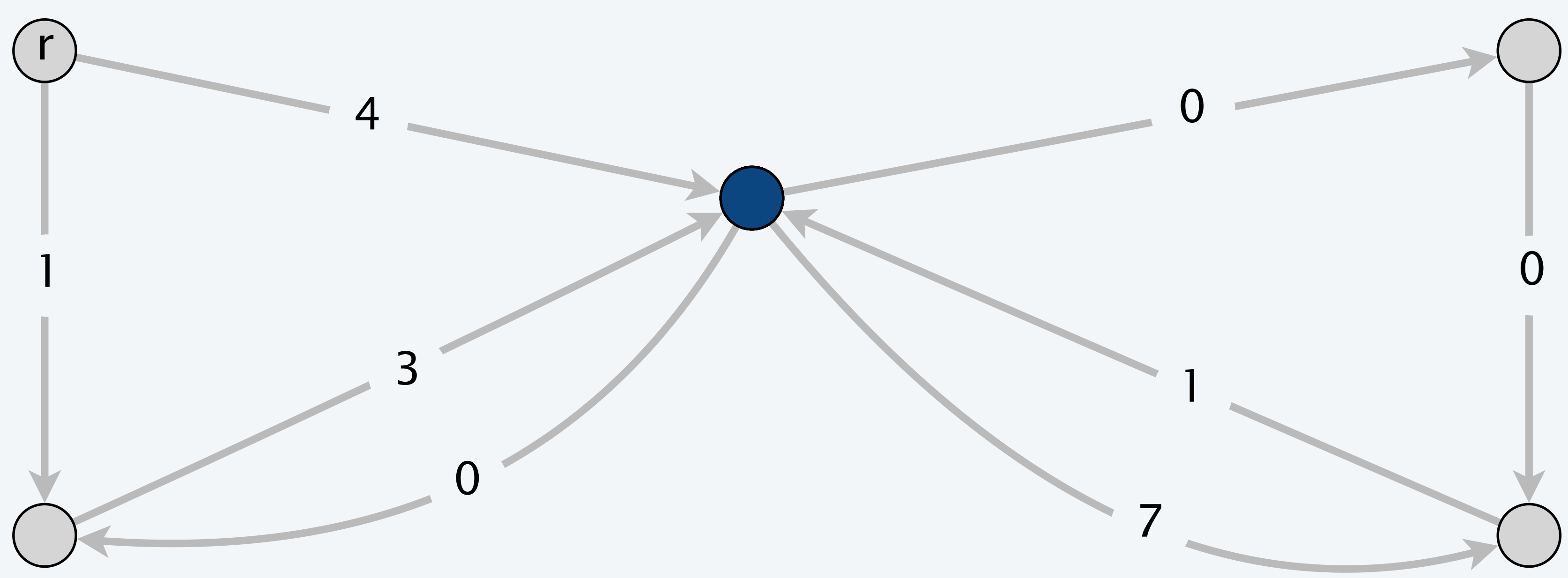
F* = thick edges



Edmonds branching algorithm: intuition

Intuition. Recall F^* = set of cheapest edges entering v for each $v \neq r$.

- Now, all edges in F^* have 0 cost with respect to reduced costs $c'(u, v)$.
- If F^* does not contain a cycle, then it is a min-cost arborescence.
- If F^* contains a cycle C, can afford to use as many edges in C as desired.
- Contract edges in C to a supernode (removing any self-loops).
- Recursively solve problem in contracted network G' with costs $c'(u, v)$.



Edmonds branching algorithm

EDMONDS-BRANCHING (G, r, c)

FOREACH $v \neq r$:

  $y(v) \leftarrow$ min cost of any edge entering v.

  $c'(u, v) \leftarrow c'(u, v) - y(v)$ for each edge (u, v) entering v.

FOREACH $v \neq r$: choose one 0-cost edge entering v and let F^* be the resulting set of edges.

IF (F^* forms an arborescence) RETURN $T = (V, F^*)$.

ELSE

  $C \leftarrow$ directed cycle in F^*.

  Contract C to a single supernode, yielding $G' = (V', E')$.

  $T' \leftarrow$ EDMONDS-BRANCHING(G', r, c').

  Extend T' to an arborescence T in G by adding all but one edge of C.

  RETURN T.

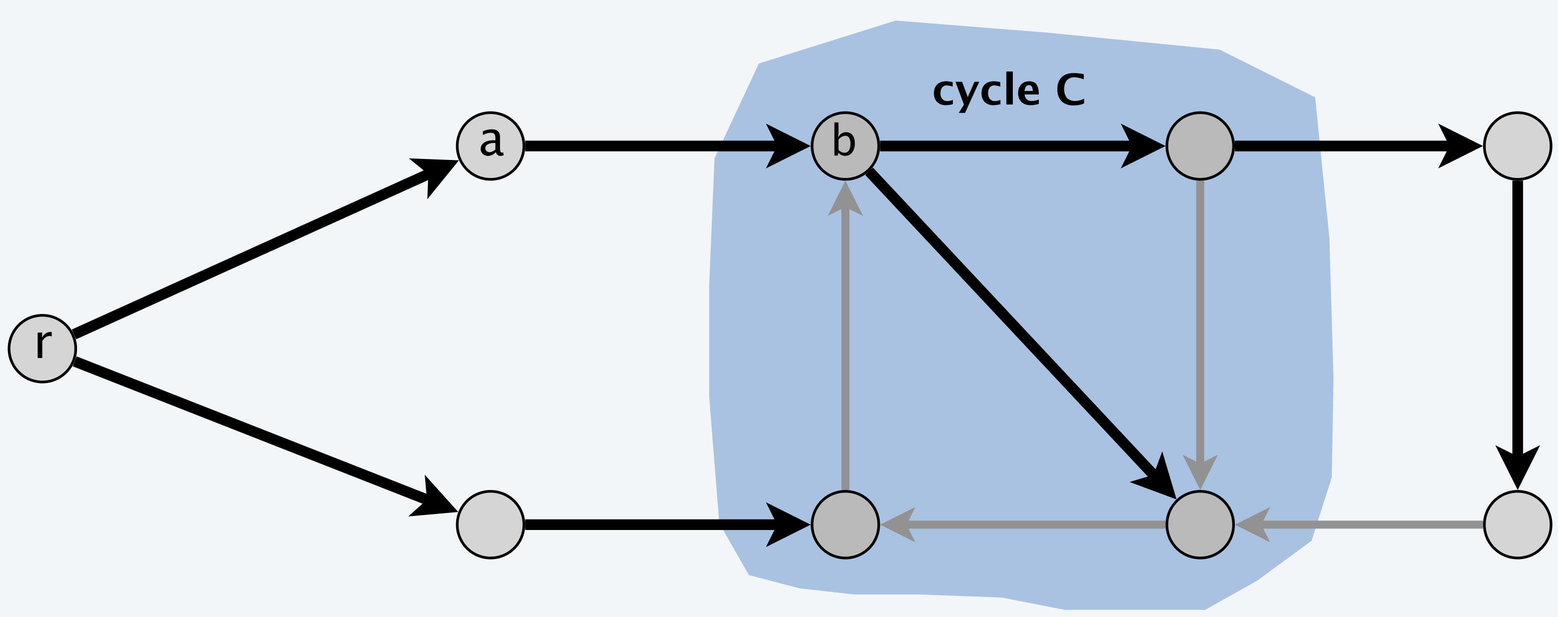
Edmonds branching algorithm

Q. What could go wrong?

A. Contracting cycle C places extra constraint on arborescence.

- Min-cost arborescence in G' must have exactly one edge entering a node in C (since C is contracted to a single node)
- But min-cost arborescence in G might have several edges entering C.

min-cost arborescence in G



Edmonds branching algorithm: key lemma

Lemma. Let C be a cycle in G containing only 0-cost edges. There exists a min-cost arborescence T rooted at r that has exactly one edge entering C.

Pf.

Case 0. T has no edges entering C.
Since T is an arborescence, there is an $r \rightsquigarrow v$ path for each node v $\Rightarrow$ at least one edge enters C. ※

Case 1. T has exactly one edge entering C.
T satisfies the lemma.

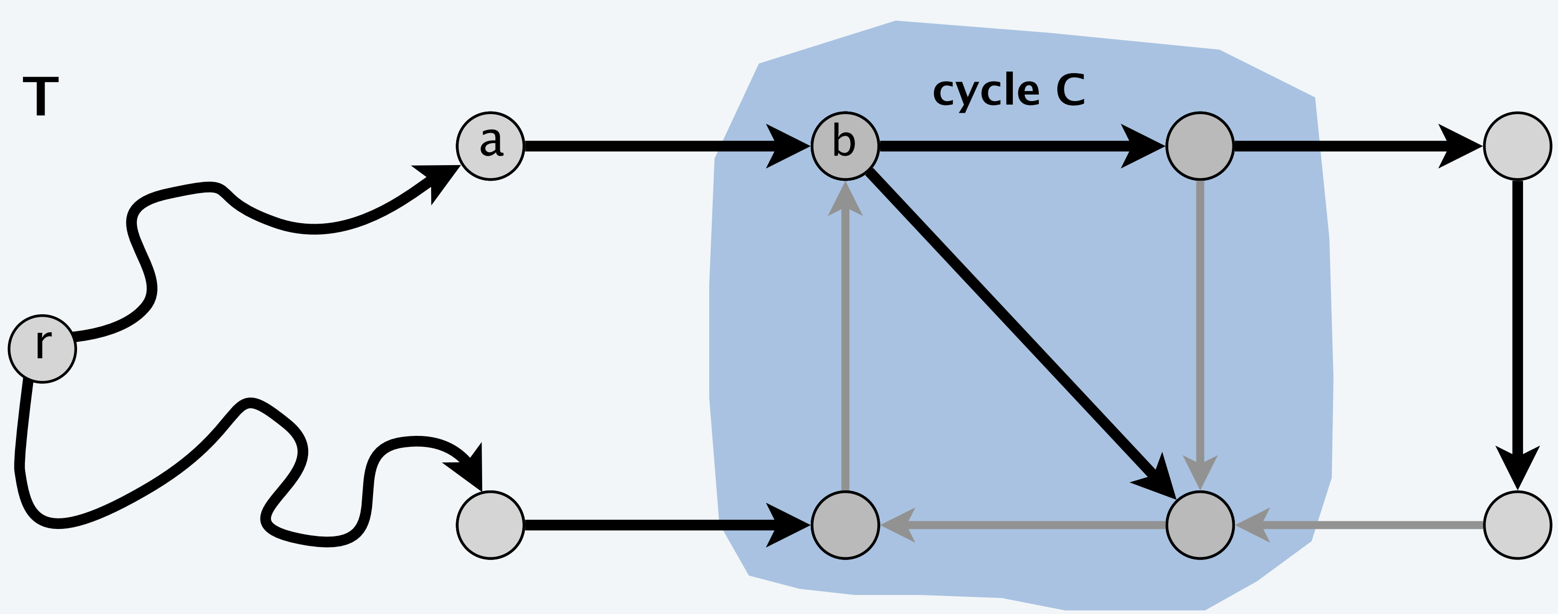
Case 2. T has two (or more) edges entering C.
We construct another min-cost arborescence T^* that has exactly one edge entering C.

Edmonds branching algorithm: key lemma

Case 2 construction of T^*.

- Let (a, b) be an edge in T entering C that lies on a shortest path from r.
- We delete all edges of T that enter a node in C except (a, b).
- We add in all edges of C except the one that enters b.

this path from r to C uses only one node in C



Edmonds branching algorithm: key lemma

Case 2 construction of T^*.

- Let (a, b) be an edge in T entering C that lies on a shortest path from r.
- We delete all edges of T that enter a node in C except (a, b).
- We add in all edges of C except the one that enters b.

this path from r to C uses only one node in C

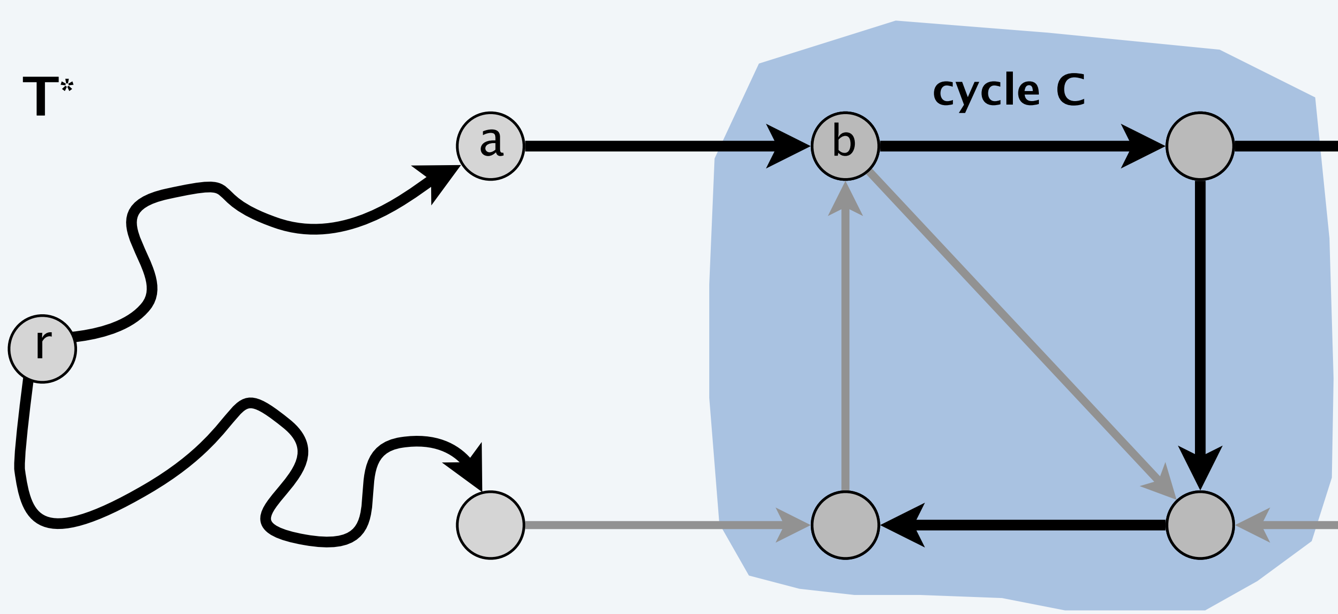
Claim. T^* is a min-cost arborescence.

- The cost of T^* is at most that of T since we add only 0-cost edges.
- T^* has exactly one edge entering each node $v \neq r$.
- T^* has no directed cycles.

T is an arborescence rooted at r

(T had no cycles before; no cycles within C; now only (a, b) enters C)

and the only path in T^* to a is the path from r to a (since any path must follow unique entering edge back to r)



Edmonds branching algorithm: analysis

Theorem. [Chu–Liu 1965, Edmonds 1967] The greedy algorithm finds a min-cost arborescence.

Pf. [by strong induction on number of nodes]

- If the edges of F^* form an arborescence, then min-cost arborescence.
- Otherwise, we use reduced costs, which is equivalent.
- After contracting a 0-cost cycle C to obtain a smaller graph G', the algorithm finds a min-cost arborescence T' in G' (by induction).
- Key lemma: there exists a min-cost arborescence T in G that corresponds to T'. ▪

Theorem. The greedy algorithm can be implemented to run in $O(m\,n)$ time.

Pf.

- At most n contractions (since each reduces the number of nodes).
- Finding and contracting the cycle C takes $O(m)$ time.
- Transforming T' into T takes $O(m)$ time. ▪

un-contracting cycle C,
remove all but one edge entering C,
taking all but one edge in C

Min-cost arborescence

Theorem. [Gabow–Galil–Spencer–Tarjan 1985] There exists an $O(m + n \log n)$ time algorithm to compute a min-cost arborescence.

COMBINATORICA 6 (2) (1986) 109—122

EFFICIENT ALGORITHMS FOR FINDING MINIMUM SPANNING TREES IN UNDIRECTED AND DIRECTED GRAPHS

H. N. GABOW*, Z. GALIL**, T. SPENCER*** and R. E. TARJAN

Received 23 January 1985
Revised 1 December 1985

Recently, Fredman and Tarjan invented a new, especially efficient form of heap (priority queue). Their data structure, the *Fibonacci heap* (or F-heap) supports arbitrary deletion in $O(\log n)$ amortized time and other heap operations in $O(1)$ amortized time. In this paper we use F-heaps to obtain fast algorithms for finding minimum spanning trees in undirected and directed graphs. For an undirected graph containing n vertices and m edges, our minimum spanning tree algorithm runs in $O(m \log \beta(m, n))$ time, improved from $O(m\beta(m, n))$ time, where $\beta(m, n) = \min \{i \mid \log^{(i)} n \leqq m/n\}$. Our minimum spanning tree algorithm for directed graphs runs in $O(n \log n + m)$ time, improved from $O(n \log n + m \log \log \log_{(m/n+2)} n)$. Both algorithms can be extended to allow a degree constraint at one vertex.