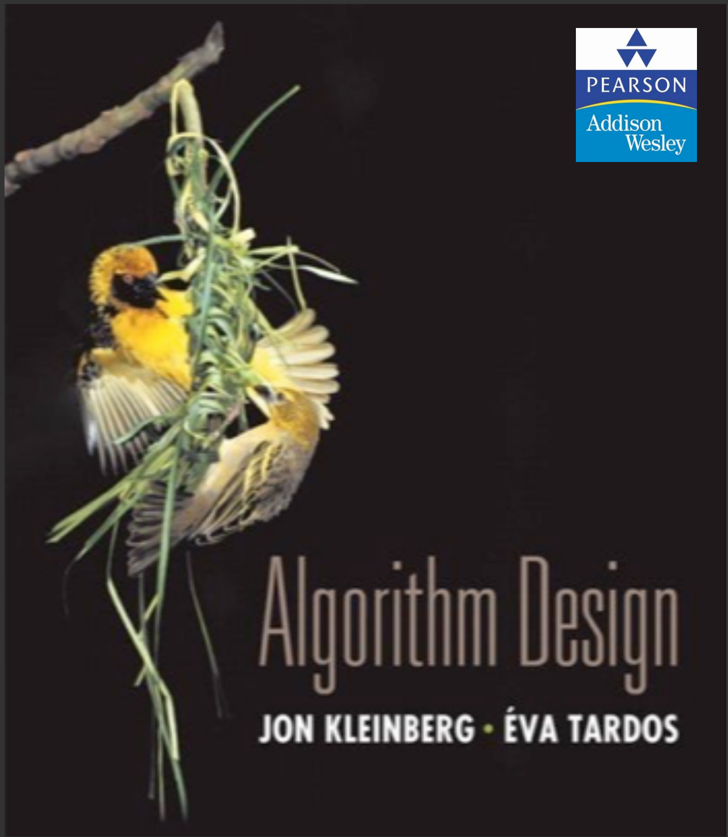


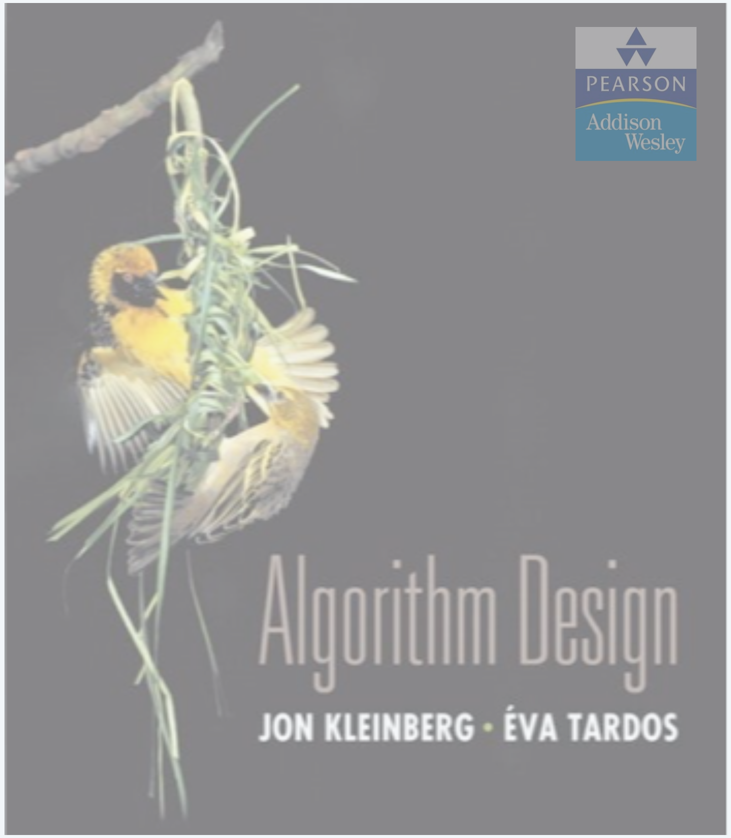
4. GREEDY ALGORITHMS II

- *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 15/9/12 下午10:40

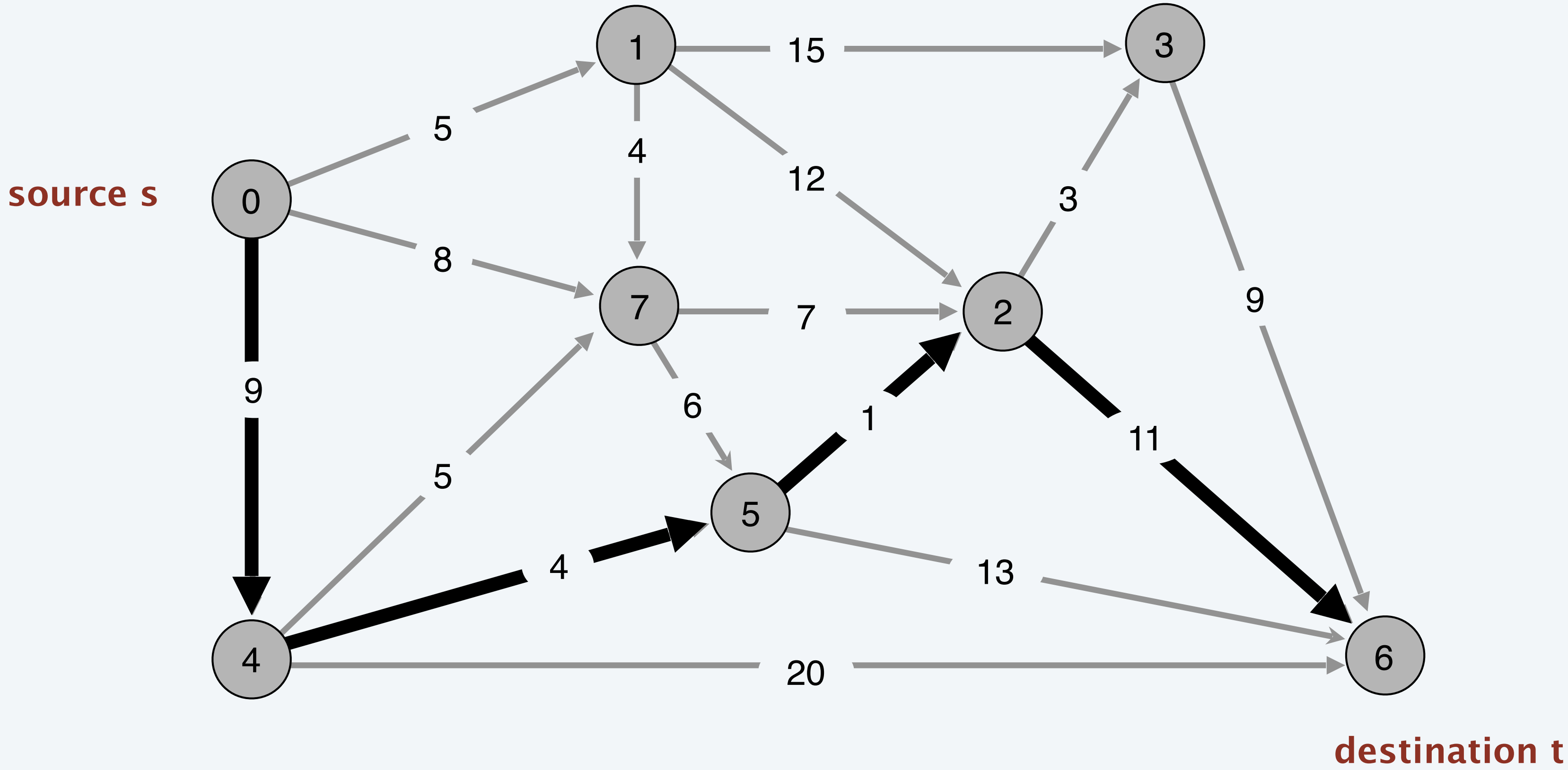


4. GREEDY ALGORITHMS II

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

Shortest-paths problem

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



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

Car navigation

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.

Reference: Network Flows: Theory, Algorithms, and Applications, R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, Prentice Hall, 1993.

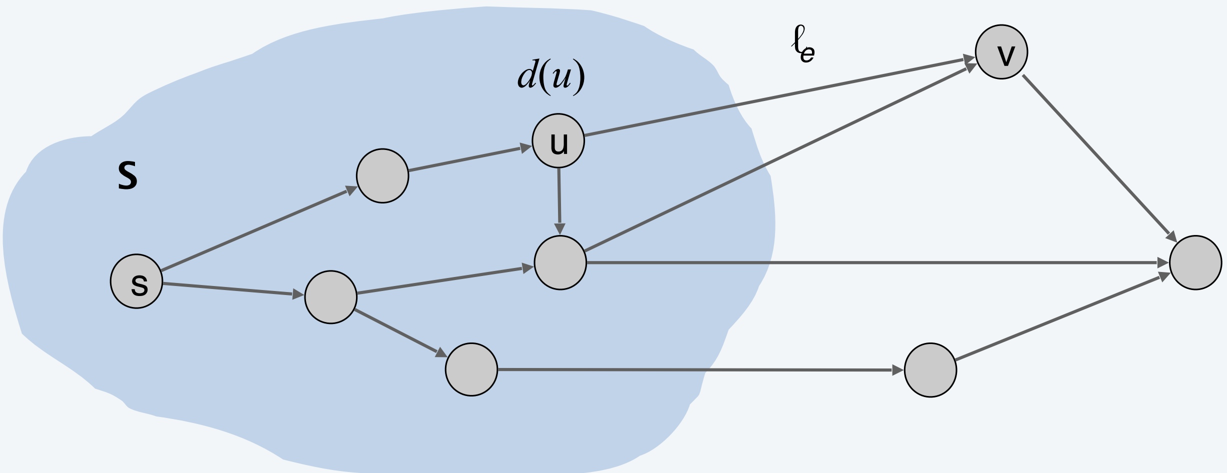
Dijkstra's algorithm

Greedy approach. Maintain a set of explored nodes S for which algorithm has determined the shortest path distance $d(u)$ from s to u.

- Initialize $S = \{ s \},\ d(s) = 0$.
- Repeatedly choose unexplored node v which minimizes

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

shortest path to some node u in explored part, followed by a single edge (u, v)



Dijkstra's algorithm

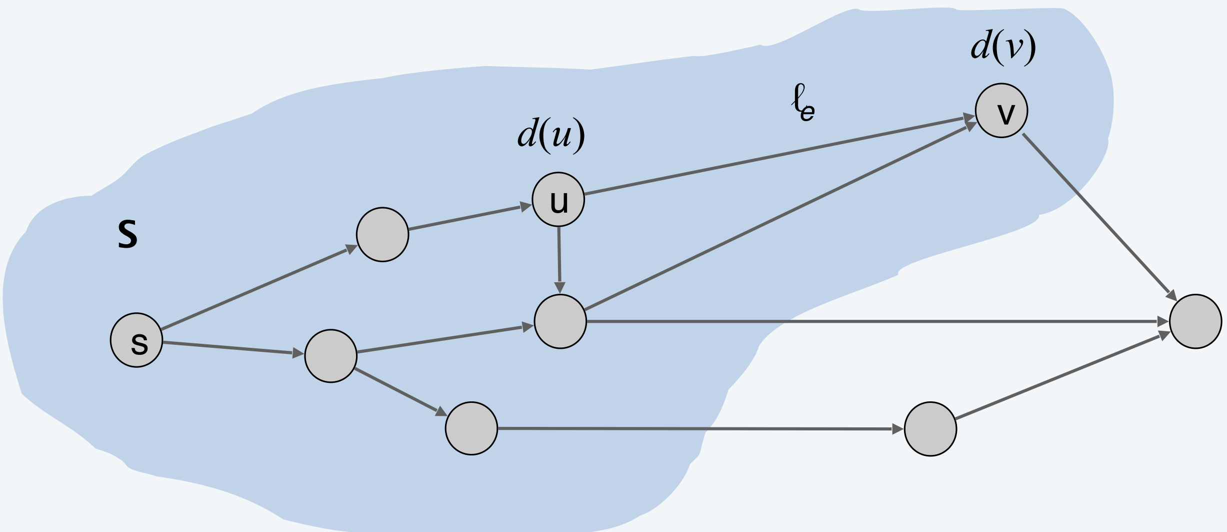
Greedy approach. Maintain a set of explored nodes S for which algorithm has determined the shortest path distance $d(u)$ from s to u.

- Initialize $S = \{ s \},\ d(s) = 0$.
- Repeatedly choose unexplored node v which minimizes

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

add v to S, and set $d(v) = \pi(v)$.

shortest path to some node u in explored part, followed by a single edge (u, v)



Dijkstra's algorithm: proof of correctness

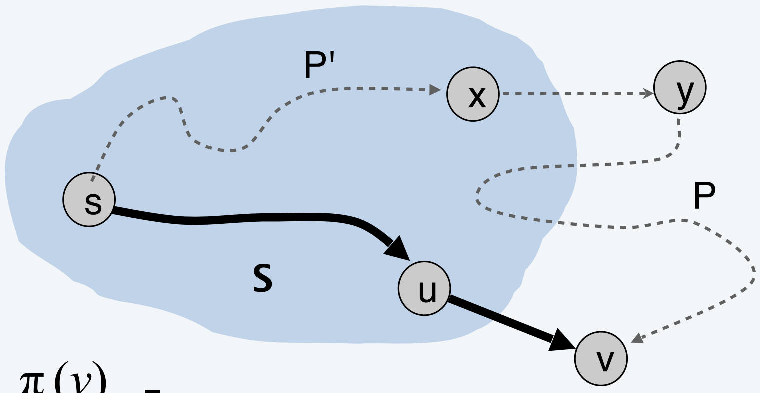
Invariant. For each node $u \in S$, $d(u)$ is the length of the shortest $s \rightsquigarrow u$ path.

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

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

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

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



$$\ell(P) \geq \ell(P') + \ell(x, y) \geq d(x) + \ell(x, y) \geq \pi(y) \geq \pi(v) \quad \blacksquare$$

nonnegative lengths

inductive hypothesis

definition of $\pi(y)$

Dijkstra chose v instead of y

Simpler proof?

l(s,u)+l(u,v) >= l(s,x)+l(x,y)

Triangle inequality

Dijkstra's algorithm: efficient implementation

Critical optimization 1. For each unexplored node v, explicitly maintain $\pi(v)$ instead of computing directly from formula:

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

- For each $v \notin S$, $\pi(v)$ can only decrease (because S only increases).
- More specifically, suppose u is added to S and there is an edge (u, v) leaving u. Then, it suffices to update:

$$\pi(v) = \min \{ \pi(v),\ d(u) + \ell(u, v) \}$$

Critical optimization 2. Use a priority queue to choose the unexplored node that minimizes $\pi(v)$.

Dijkstra's algorithm: efficient implementation

Implementation.

- Algorithm stores $d(v)$ for each explored node v.
- Priority queue stores $\pi(v)$ for each unexplored node v.
- Recall: $d(u) = \pi(u)$ when u is deleted from priority queue.

DIJKSTRA (V, E, s)

Create an empty priority queue.

FOR EACH $v \neq s$: $d(v) \leftarrow \infty$; $d(s) \leftarrow 0$.

FOR EACH $v \in V$: *insert* v with key $d(v)$ into priority queue.

WHILE (the priority queue *is not empty*)

  $u \leftarrow$ *delete-min* from priority queue.

  FOR EACH edge $(u, v) \in E$ leaving u:

    IF $d(v) > d(u) + \ell(u, v)$

      decrease-key of v to $d(u) + \ell(u, v)$ in priority queue.

      $d(v) \leftarrow d(u) + \ell(u, v)$.

Dijkstra's algorithm: which priority queue?

Maximum m edges, so maximum m decreasing-keys

Performance. Depends on PQ: n insert, n delete-min, m decrease-key.

- Array implementation optimal for dense graphs.
- Binary heap much faster for sparse graphs.
- 4-way heap worth the trouble in performance-critical situations.
- Fibonacci/Brodal best in theory, but not worth implementing.

most frequent operation

Potential project!
DNN pruning
Dense matrix (training)
->
Sparse matrix
(inference)

PQ implementation	insert	delete-min	decrease-key	total
unordered array	$O(1)$	$O(n)$	$O(1)$	$O(n^2)$
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)	$O(1)$	$O(\log n)$ †	$O(1)$ †	$O(m + n \log n)$
Brodal queue (Brodal 1996)	$O(1)$	$O(\log n)$	$O(1)$	$O(m + n \log n)$

process nodes in increasin g order of distance from source

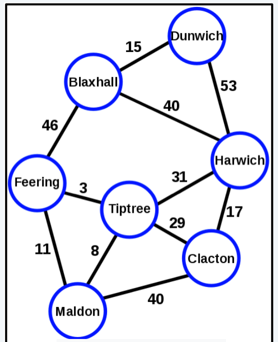
† amortized

Extensions of Dijkstra's algorithm

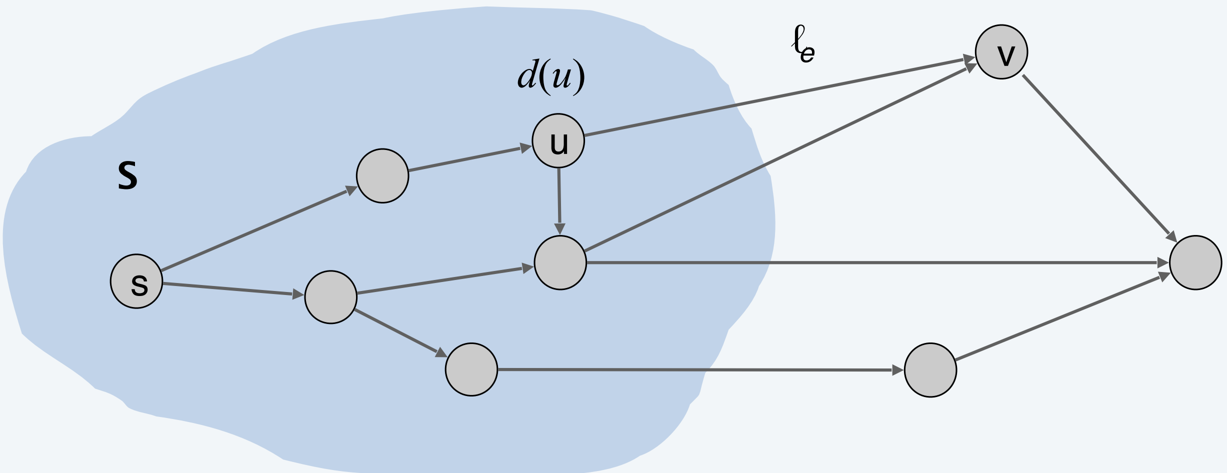
Dijkstra's algorithm and proof extend to several related problems:

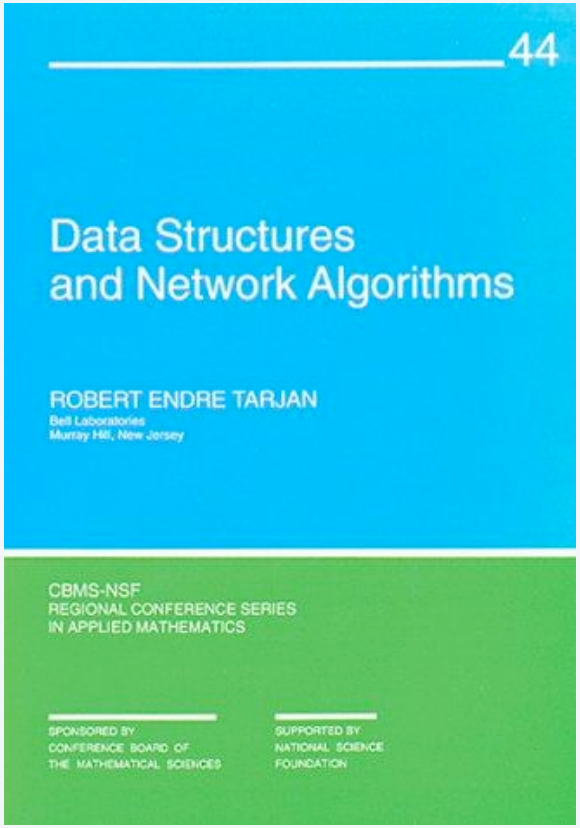
- Shortest paths in undirected graphs: $d(v) \le d(u) + \ell(u, v)$.
- Maximum capacity paths: $d(v) \ge \min \{ d(u), c(u, v) \}$.
- Maximum reliability paths: $d(v) \ge d(u) \times \gamma(u, v)$.
- …

Key algebraic structure. Closed semiring (tropical, bottleneck, Viterbi).



The maximum capacity path from Maldon to Feering has bandwidth 29, and passes through Clacton, Tiptree, Harwich, and Blaxhall.





SECTION 6.1

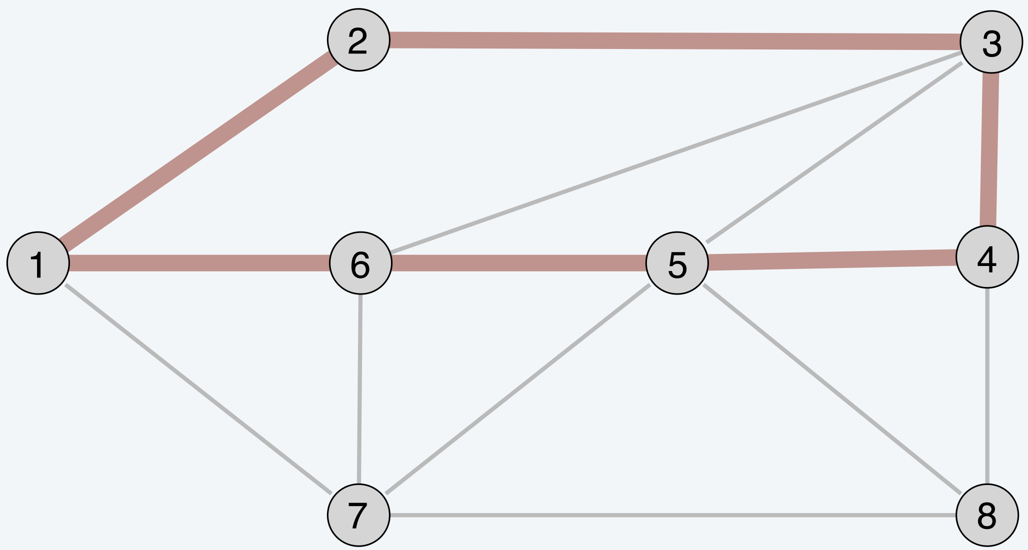
4. GREEDY ALGORITHMS II

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

Cycles and cuts

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.

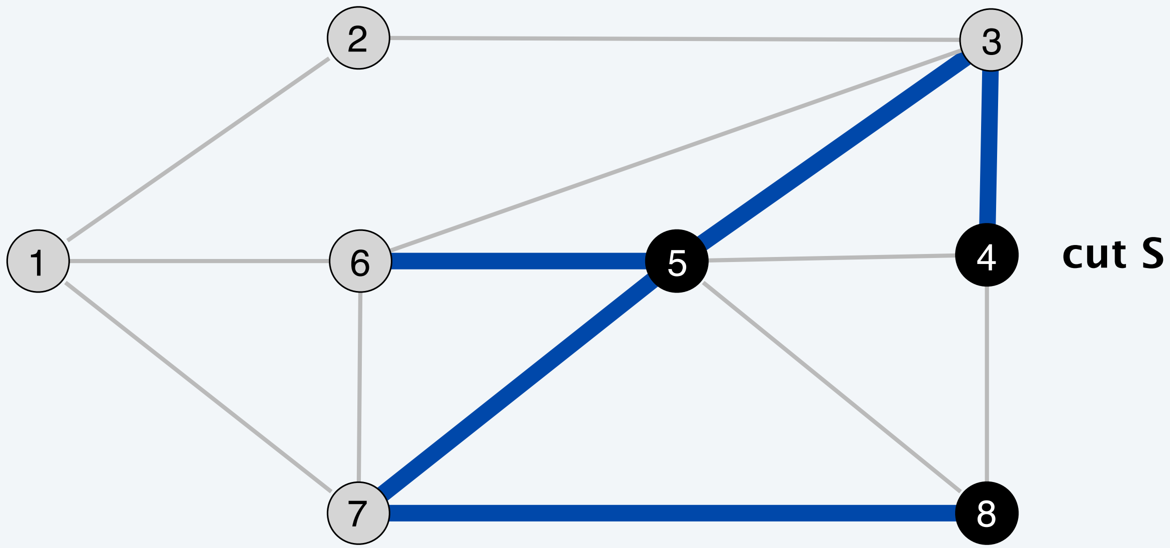


cycle C = { (1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 1) }

Cycles and cuts

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

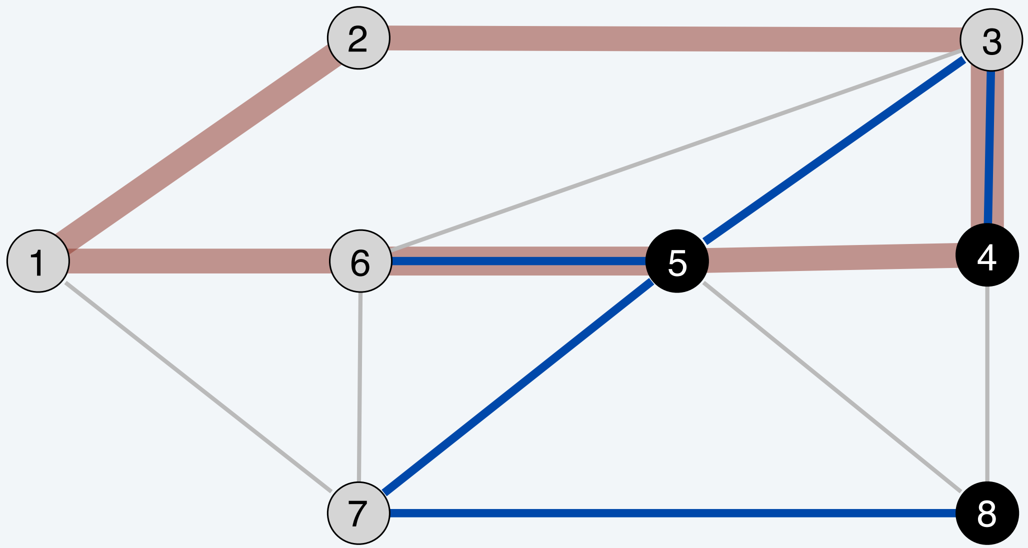
Def. The cutset of a cut S is the set of edges with exactly one endpoint in S.



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

Cycle-cut intersection

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



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

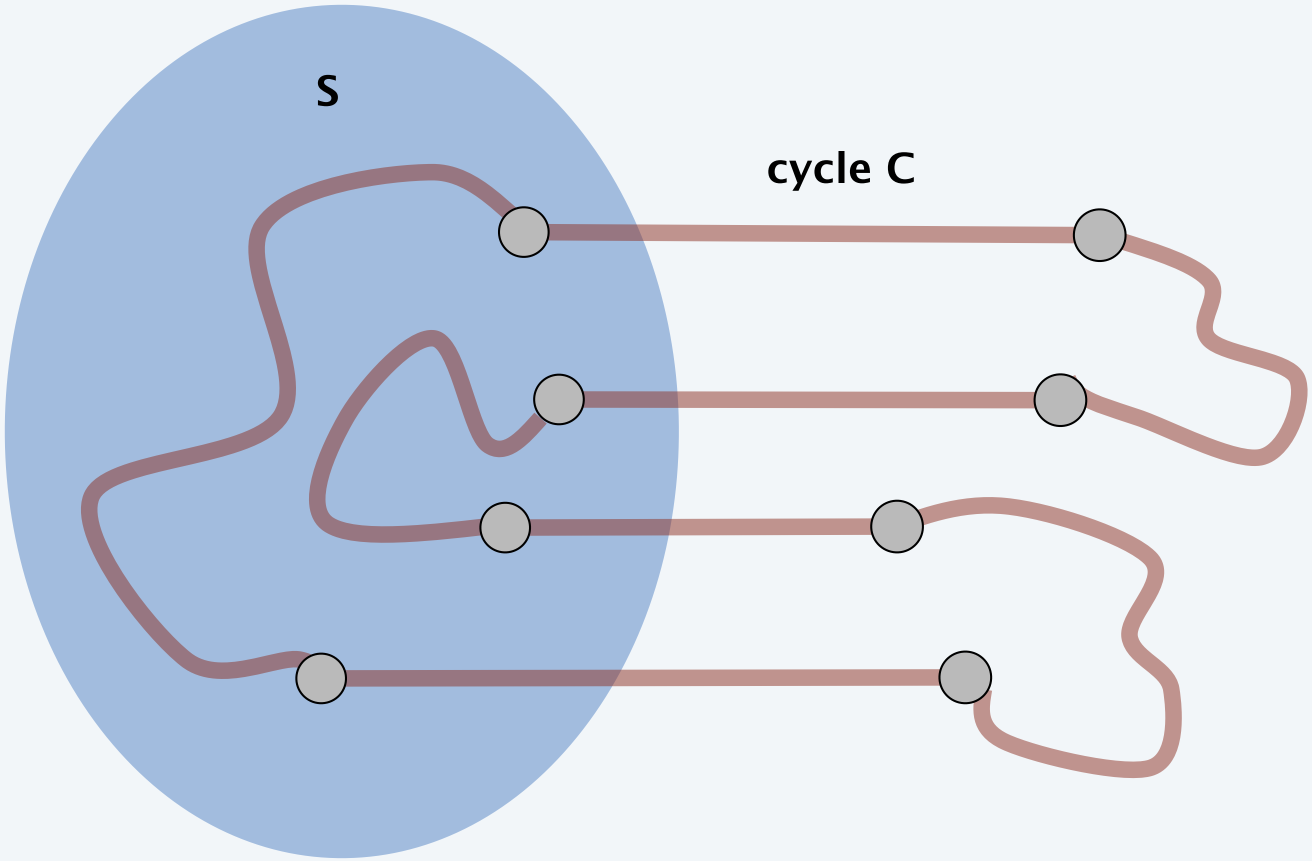
cycle C = { (1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 1) }

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

Cycle-cut intersection

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

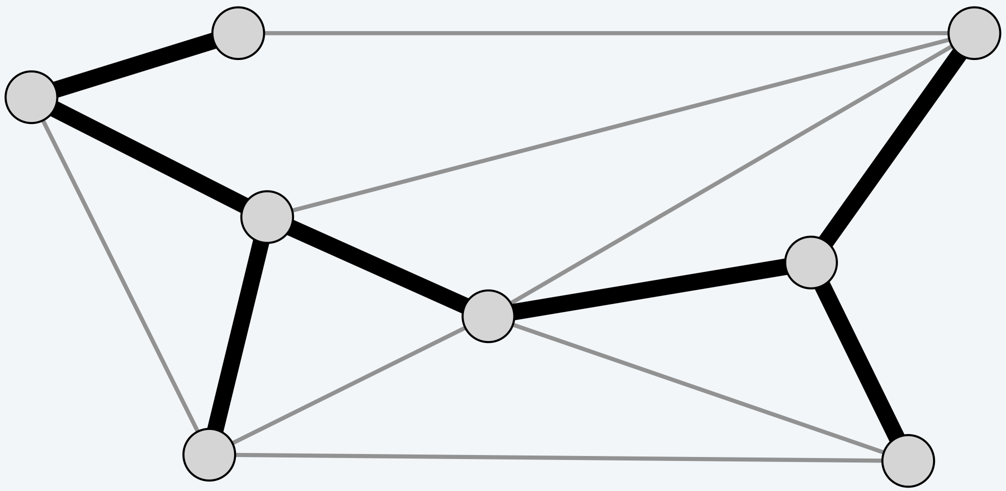
Pf. [by picture]



Spanning tree properties

Proposition. Let $T = (V, F)$ be a subgraph of $G = (V, E)$. TFAE:

- T is a spanning tree of G.
- T is acyclic and connected.
- T is connected and has $n - 1$ edges.
- T is acyclic and has $n - 1$ edges.
- T is minimally connected: removal of any edge disconnects it.
- T is maximally acyclic: addition of any edge creates a cycle.
- T has a unique simple path between every pair of nodes.



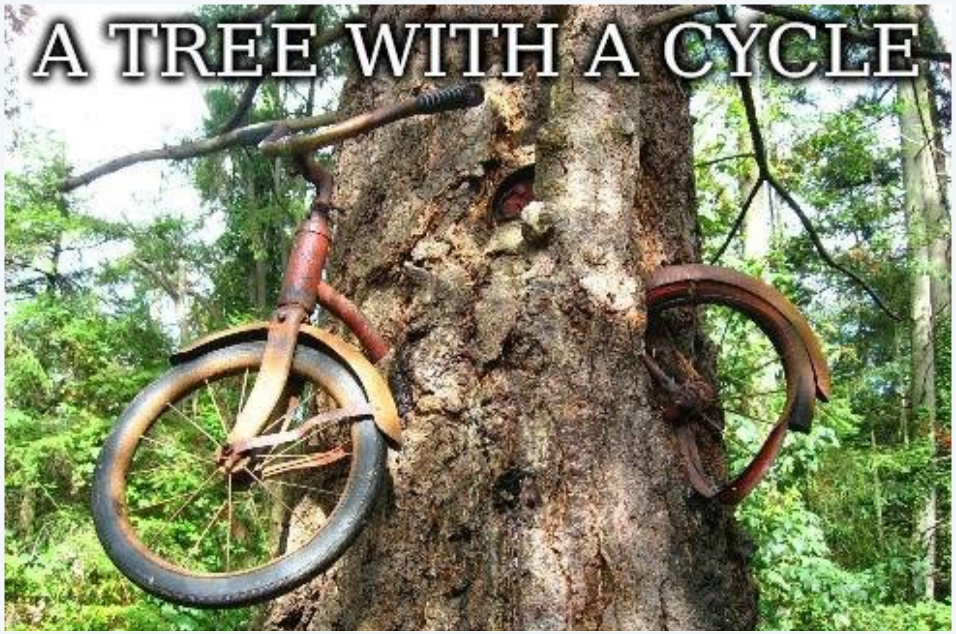
A Tree has no cycle EXCEPT!!!

T = (V, F)

Spanning tree properties

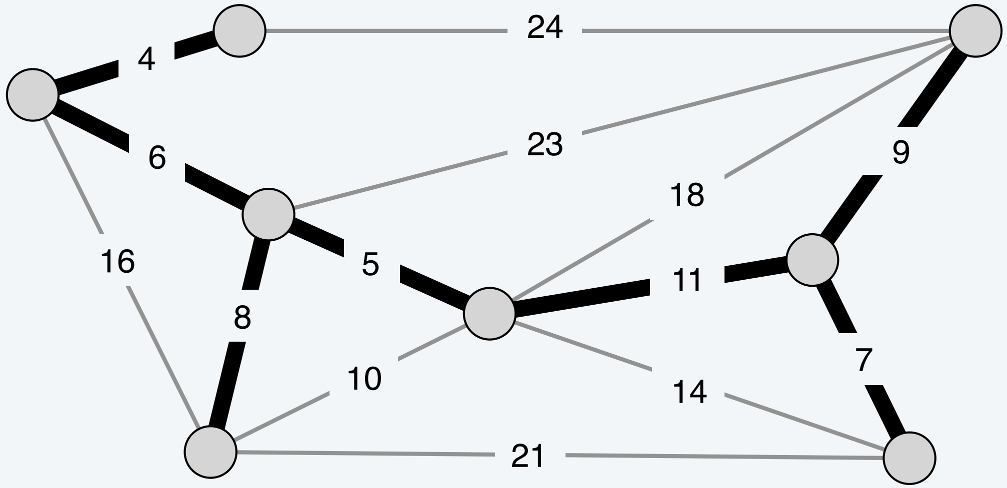
Proposition. Let $T = (V, F)$ be a subgraph of $G = (V, E)$. TFAE:

- T is a spanning tree of G.
- T is acyclic and connected.
- T is connected and has $n - 1$ edges.
- T is acyclic and has $n - 1$ edges.
- T is minimally connected: removal of any edge disconnects it.
- T is maximally acyclic: addition of any edge creates a cycle.
- T has a unique simple path between every pair of nodes.



Minimum spanning tree

Given a connected graph $G = (V, E)$ with edge costs c_e, an MST is a subset of the edges $T \subseteq E$ such that T is a spanning tree whose sum of edge costs is minimized.



Many simple greedy algorithms that you first try may be correct!

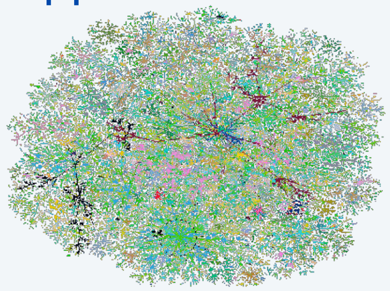
MST cost = 50 = 4 + 6 + 8 + 5 + 11 + 9 + 7

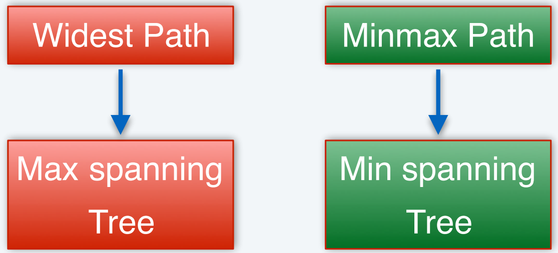
Cayley's theorem. There are n^{n-2} spanning trees of K_n.

← can't solve by brute force

Applications

MST is fundamental problem with diverse applications.



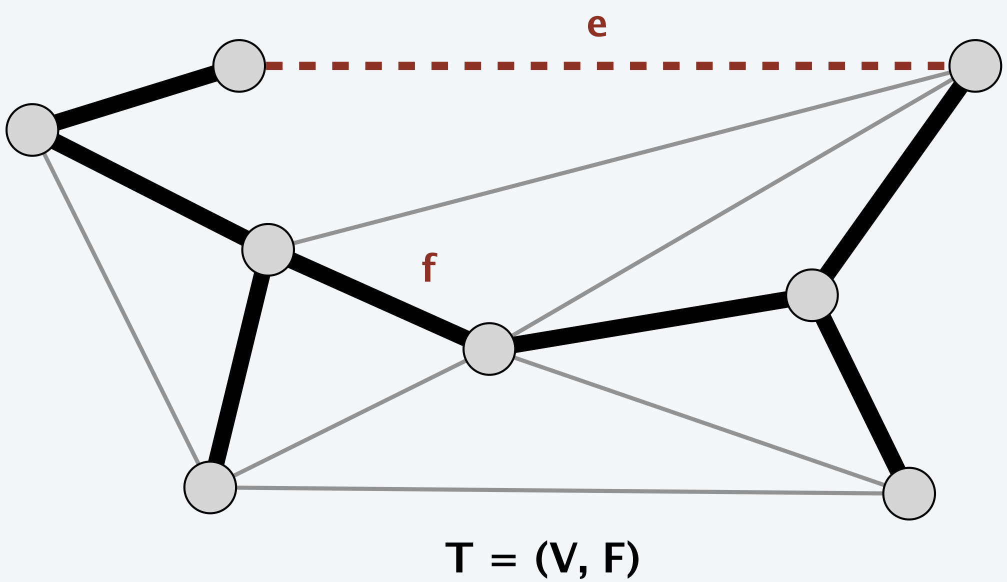


- Dithering.
- Cluster analysis.
- Max bottleneck paths.
- Real-time face verification.
- LDPC codes for error correction.
- Image registration with Renyi entropy.
- Find road networks in satellite and aerial imagery.
- Reducing data storage in sequencing amino acids in a protein.
- Model locality of particle interactions in turbulent fluid flows.
- 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).

Fundamental cycle

Fundamental cycle.

- Adding any non-tree edge e to a spanning tree T forms unique cycle C.
- Deleting any edge $f \in C$ from $T \cup \{ e \}$ results in new spanning tree.

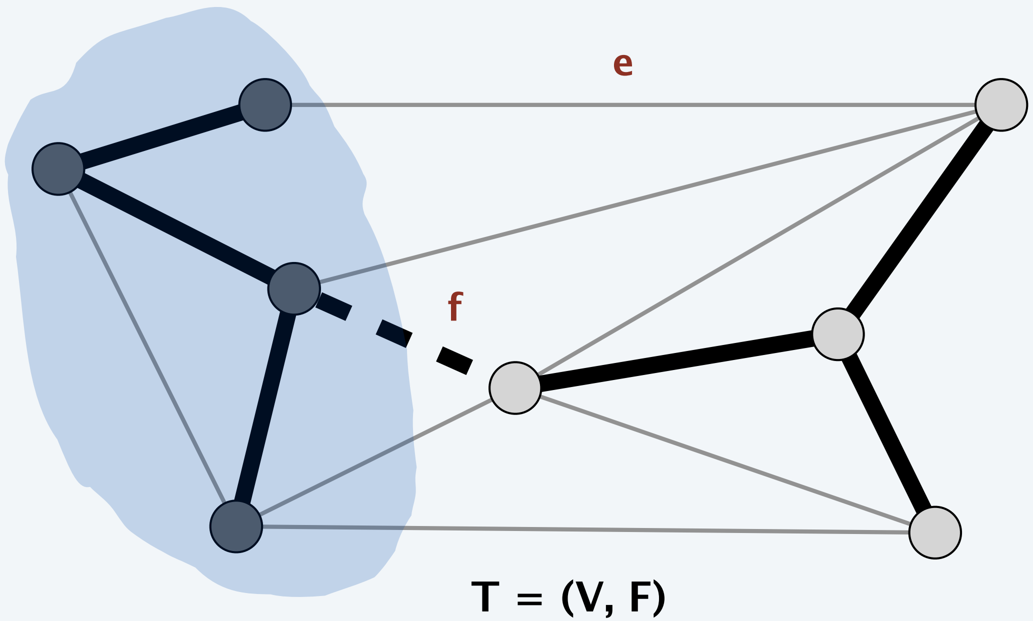


Observation. If $c_e < c_f$, then T is not an MST.

Fundamental cutset

Fundamental cutset.

- Deleting any tree edge f from a spanning tree T divide nodes into two connected components. Let D be cutset.
- Adding any edge $e \in D$ to $T - \{ f \}$ results in new spanning tree.



Observation. If $c_e < c_f$, then 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 weight and color it red.

Blue rule.

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

Greedy algorithm.

- Apply the red and blue rules (non-deterministically!) 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 T^* containing all of the blue edges and none of the red edges.

Pf. [by induction on number of iterations]

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

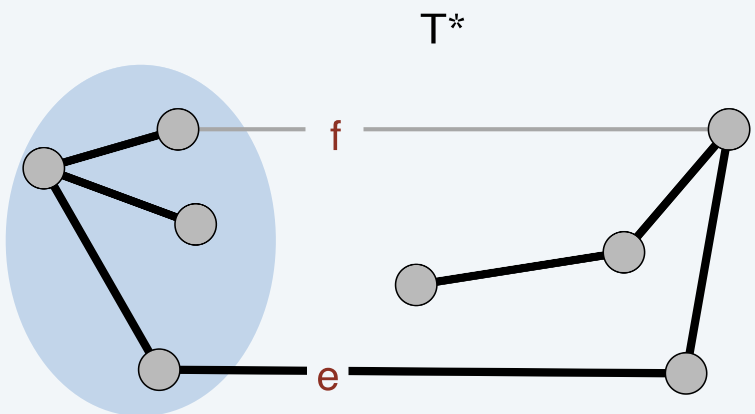
Greedy algorithm: proof of correctness

Color invariant. There exists an MST T^* containing all of the blue edges and none of the red edges.

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 $f \in T^*$, 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.



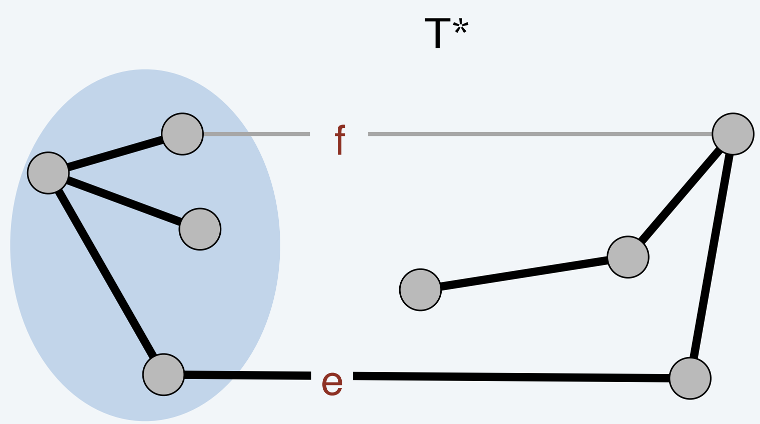
Greedy algorithm: proof of correctness

Color invariant. There exists an MST T^* containing all of the blue edges and none of the red edges.

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 $e \notin T^*$, 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. ▪



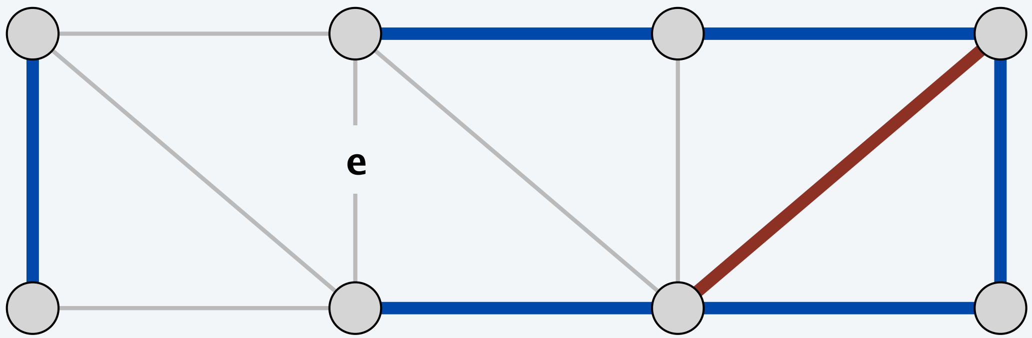
Greedy algorithm: proof of correctness

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

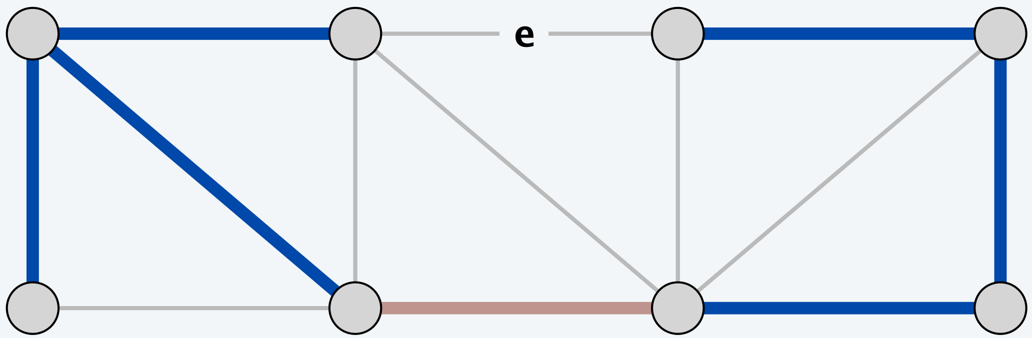
Theorem. The greedy algorithm terminates. Blue edges form an MST.

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

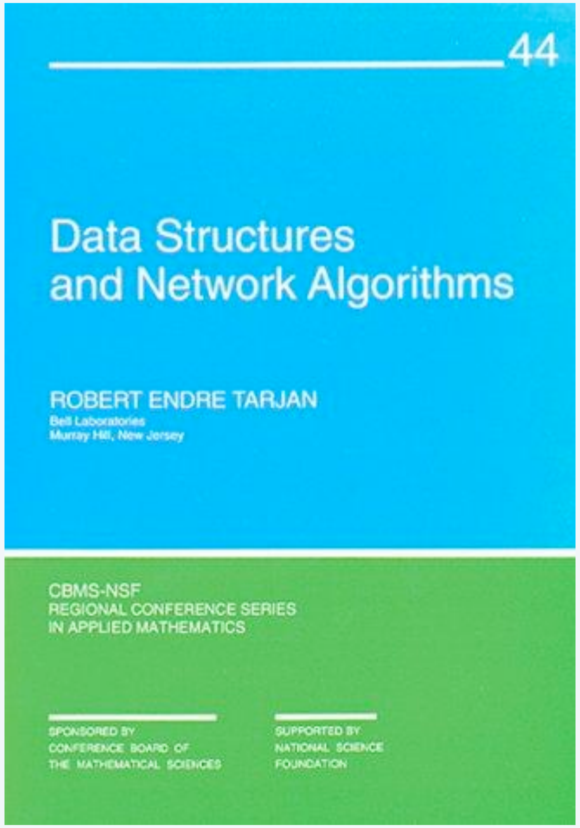
- Suppose edge e is left uncolored.
- Blue edges form a forest.
- Case 1: both endpoints of e are in same blue tree.

  ⇒ 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

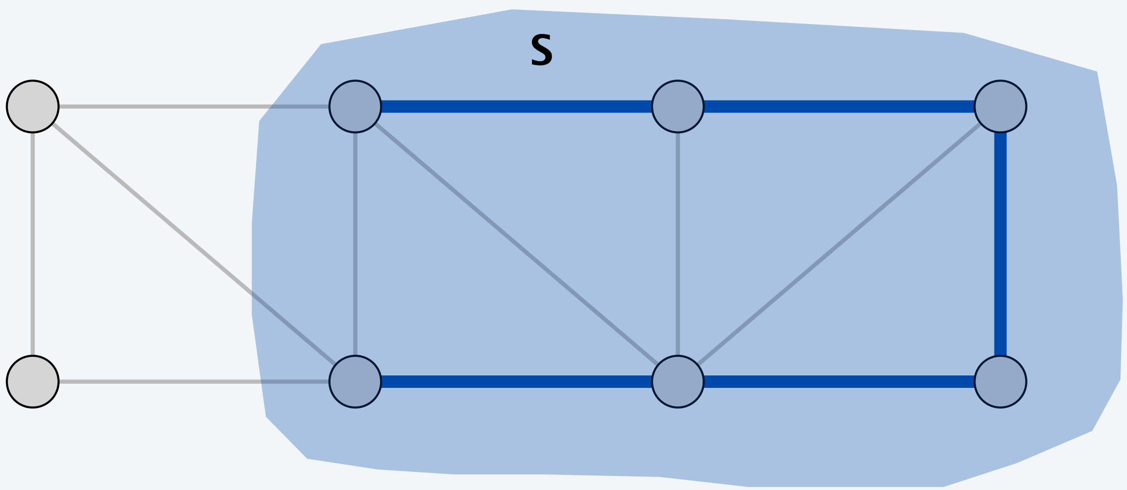
Initialize S = any node.

Repeat $n - 1$ times:

- Add to tree the min weight edge with one endpoint in S.
- Add new node to S.

Theorem. Prim's algorithm computes the MST.

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



Prim's algorithm: implementation

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

Pf. Implementation almost identical to Dijkstra's algorithm.

[$d(v)$ = weight of cheapest known edge between v and S]

PRIM (V, E, c)

Create an empty priority queue.

$s \leftarrow$ any node in V.

FOR EACH $v \neq s$: $d(v) \leftarrow \infty$; $d(s) \leftarrow 0$.

FOR EACH v : *insert* v with key $d(v)$ into priority queue.

WHILE (the priority queue *is not empty*)

  $u \leftarrow$ *delete-min* from priority queue.

  FOR EACH edge $(u, v) \in E$ incident to u:

    IF $d(v) > c(u, v)$

      decrease-key of v to $c(u, v)$ in priority queue.

      $d(v) \leftarrow c(u, v)$.

Kruskal's algorithm

Consider edges in ascending order of weight:

- Add to tree unless it would create a cycle.

Theorem. Kruskal's algorithm computes the MST.

Pf. Special case of greedy algorithm.

- Case 1: both endpoints of e in same blue tree.

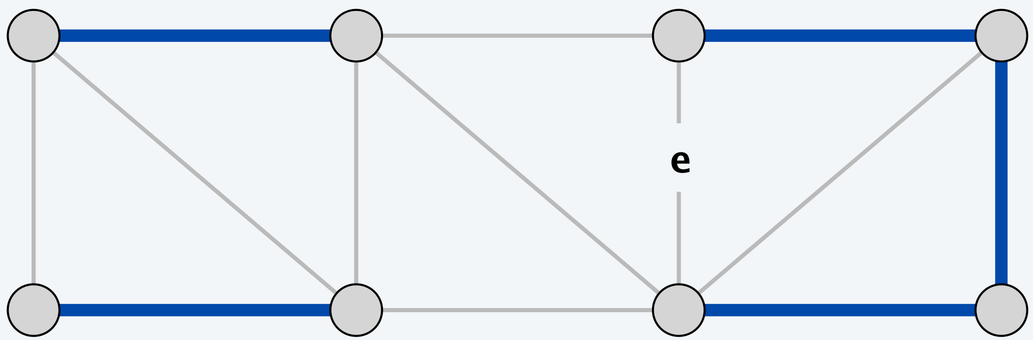
  ⇒ color red by applying red rule to unique cycle.

  all other edges in cycle are blue

- Case 2. If both endpoints of e are in different blue trees.

  ⇒ color blue by applying blue rule to cutset defined by either tree. ▪

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



Kruskal's algorithm: implementation

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

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

KRUSKAL (V, E, c)

SORT m edges by weight so that $c(e_1) \leq c(e_2) \leq \ldots \leq c(e_m)$

$S \leftarrow \phi$

FOREACH $v \in V$: MAKESET(v).

FOR $i = 1$ TO m

  $(u, v) \leftarrow e_i$

  IF FINDSET$(u) \neq$ FINDSET(v) ← are u and v in same component?

    $S \leftarrow S \cup \{ e_i \}$

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

RETURN S

Reverse-delete algorithm

Consider edges in descending order of weight:

- Remove edge unless it would disconnect the graph.

Theorem. The reverse-delete algorithm computes the MST.

Pf. Special case of greedy algorithm.

- Case 1: removing edge e does not disconnect graph.

  $\Rightarrow$ apply red rule to cycle C formed by adding e to existing path between its two endpoints

  any edge in C with larger weight would have been deleted when considered

- Case 2: removing edge e disconnects graph.

  $\Rightarrow$ apply blue rule to cutset D induced by either component. ▪

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

Fact. [Thorup 2000] Can be implemented 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 weight and color it red.

Blue rule.

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

Greedy algorithm.

- Apply the red and blue rules (non-deterministically!) 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, ...

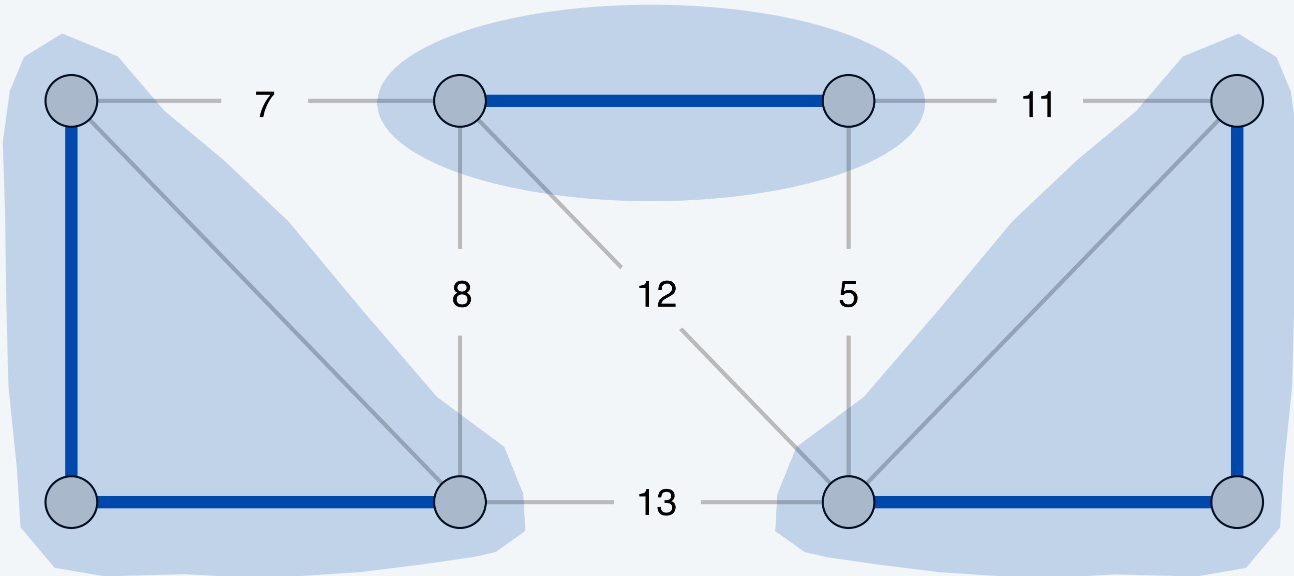
Borůvka's algorithm

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). ▪

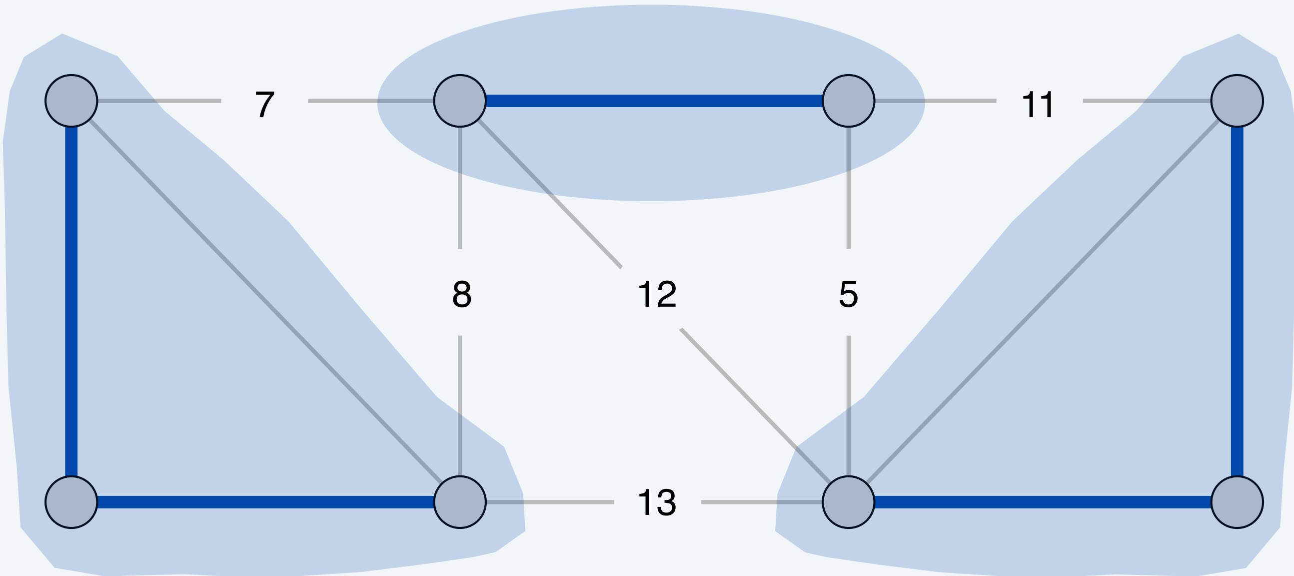


Borůvka's algorithm: implementation

Theorem. Borůvka's algorithm can be implemented 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
- At most $\log_2 n$ phases since each phase (at least) halves total # trees. ▪

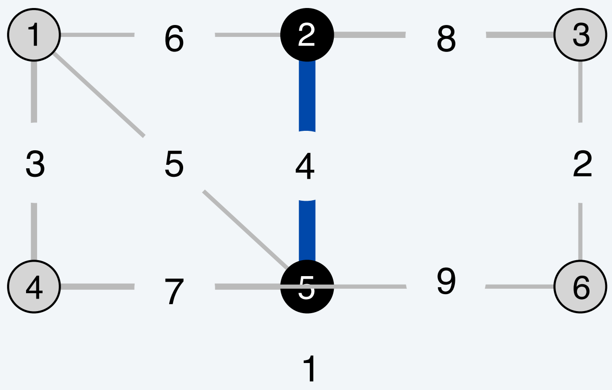


Borůvka's algorithm: implementation

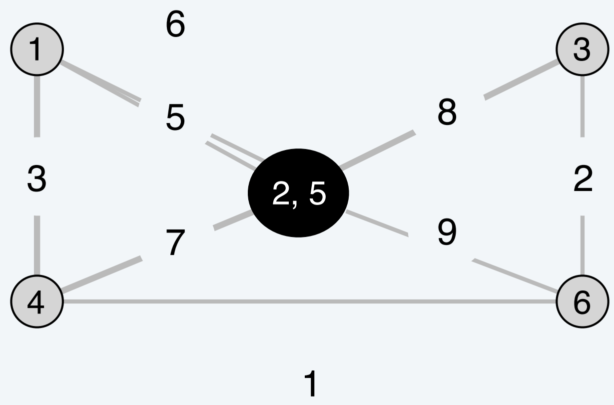
Node contraction version.

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

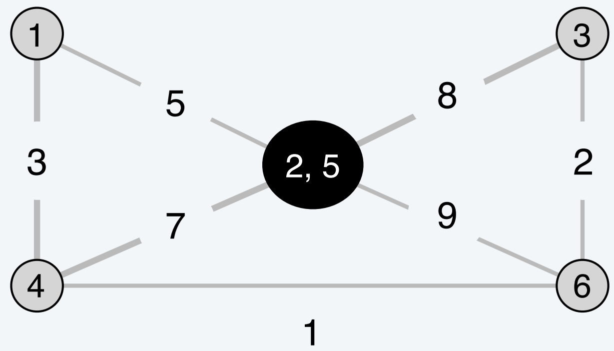
graph G



contract nodes 2 and 5



delete parallel edges and self loops

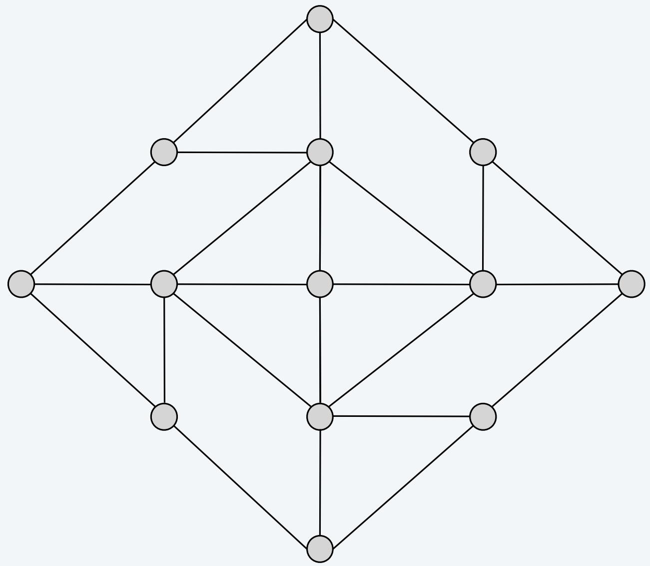


Borůvka's algorithm on planar graphs

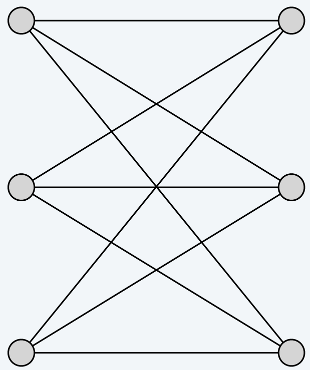
Theorem. Borůvka's algorithm runs in $O(n)$ time on planar graphs.

Pf.

- To implement a Borůvka phase in $O(n)$ time:
  - use contraction version of algorithm
  - in planar graphs, $m \leq 3n - 6$.
  - graph stays planar when we contract a blue tree
- Number of nodes (at least) halves.
- At most $\log_2 n$ phases: $cn + cn/2 + cn/4 + cn/8 + \ldots = O(n)$. ▪



planar



not planar

Borůvka-Prim algorithm

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

Theorem. The Borůvka-Prim algorithm computes an MST and can be implemented in $O(m \log \log n)$ time.

Pf.

- Correctness: special case of the greedy algorithm.
- The $\log_2 \log_2 n$ phases of Borůvka's algorithm take $O(m \log \log n)$ time; resulting graph has at most $n / \log_2 n$ nodes and 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. ▪

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

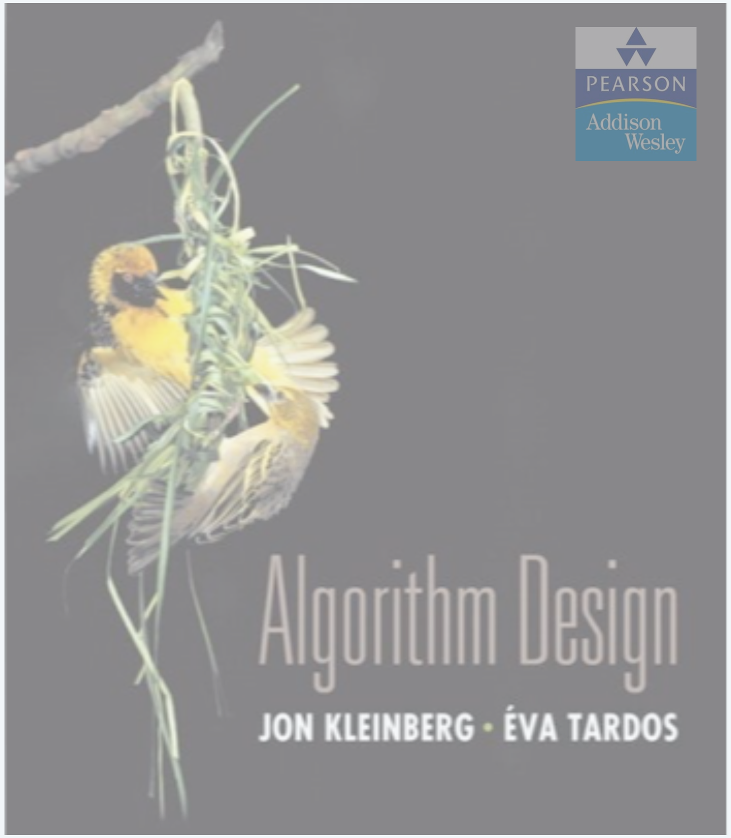
Does a linear-time MST algorithm exist?

deterministic compare-based MST algorithms

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	*optimal*	Pettie-Ramachandran
20xx	$O(m)$	???

Remark 1. $O(m)$ randomized MST algorithm. [Karger-Klein-Tarjan 1995]

Remark 2. $O(m)$ MST verification algorithm. [Dixon-Rauch-Tarjan 1992]



4. GREEDY ALGORITHMS II

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