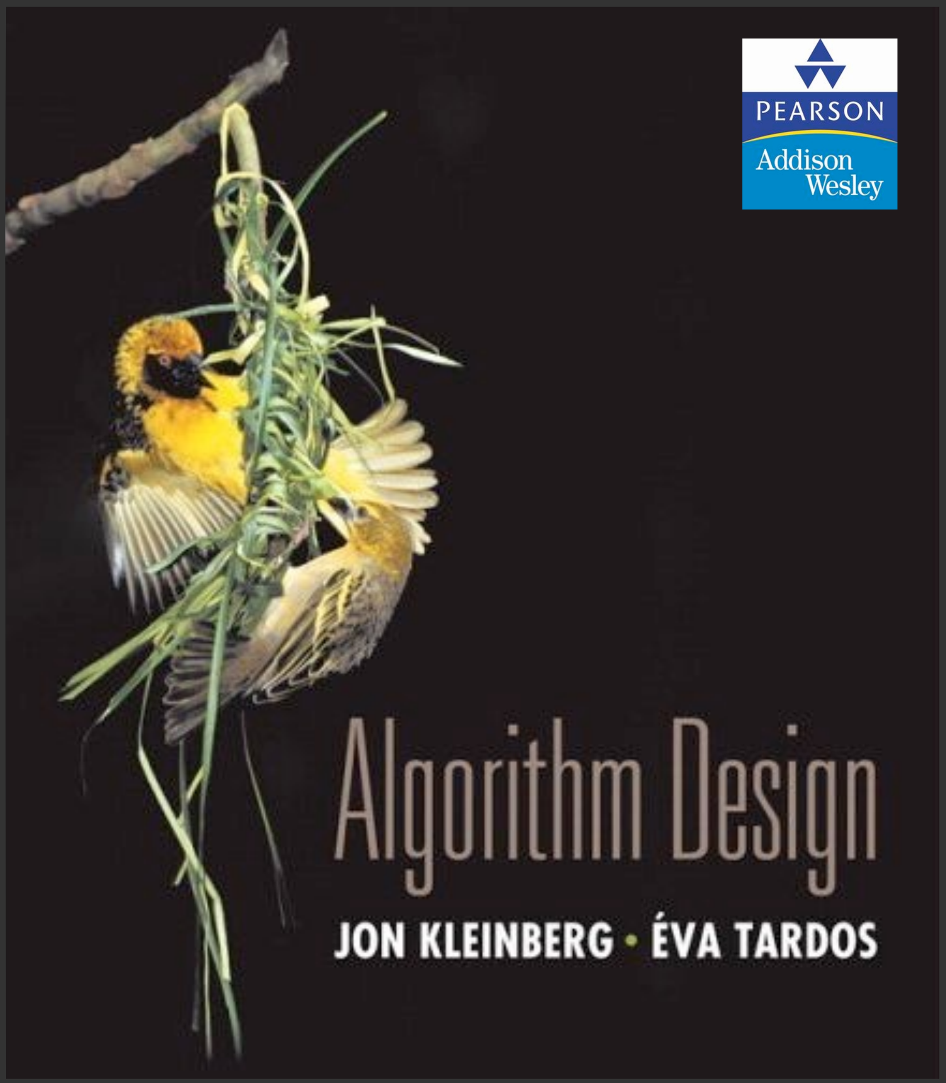


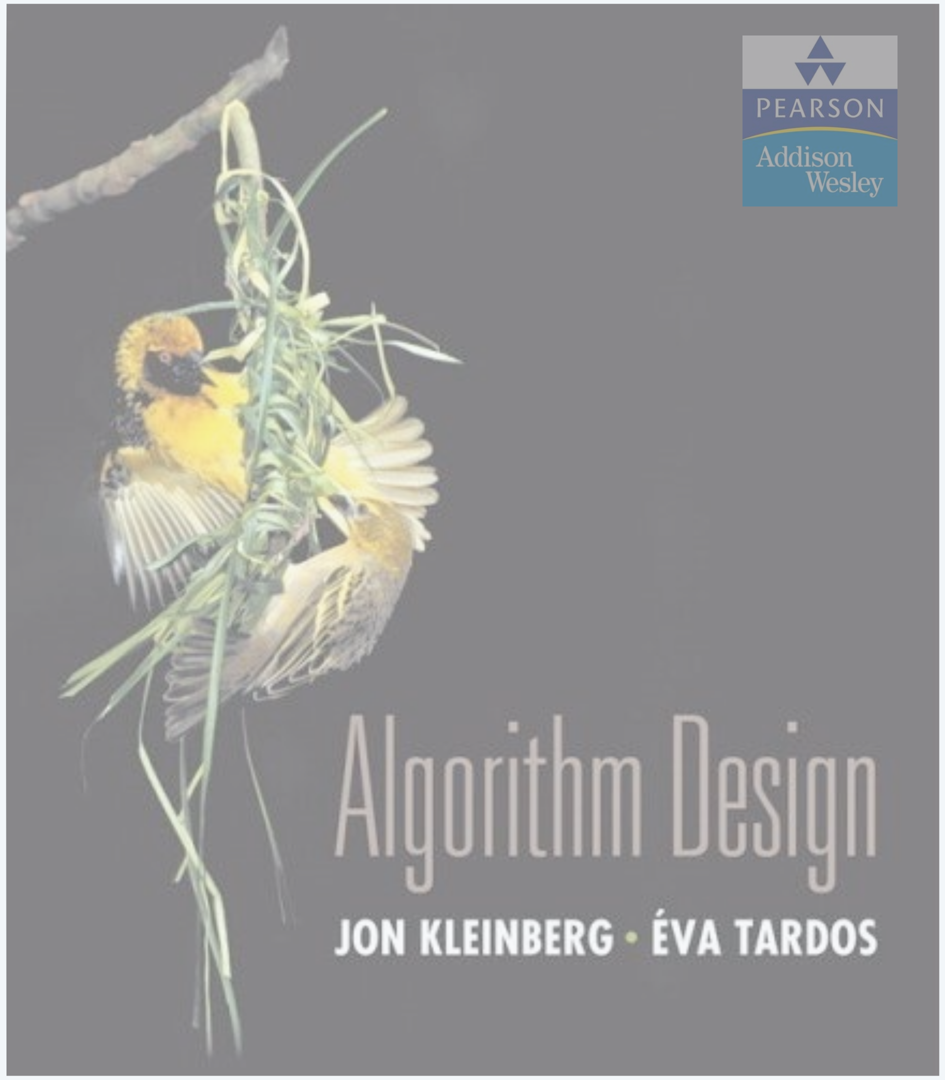
Lecture slides by Kevin Wayne

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

4. Greedy Algorithms II

- *Dijkstra′s algorithm*
- *minimum spanning trees*
- *Prim, Kruskal*
- *Union-Find Structure*

Last updated on 10/9/24 11:03AM

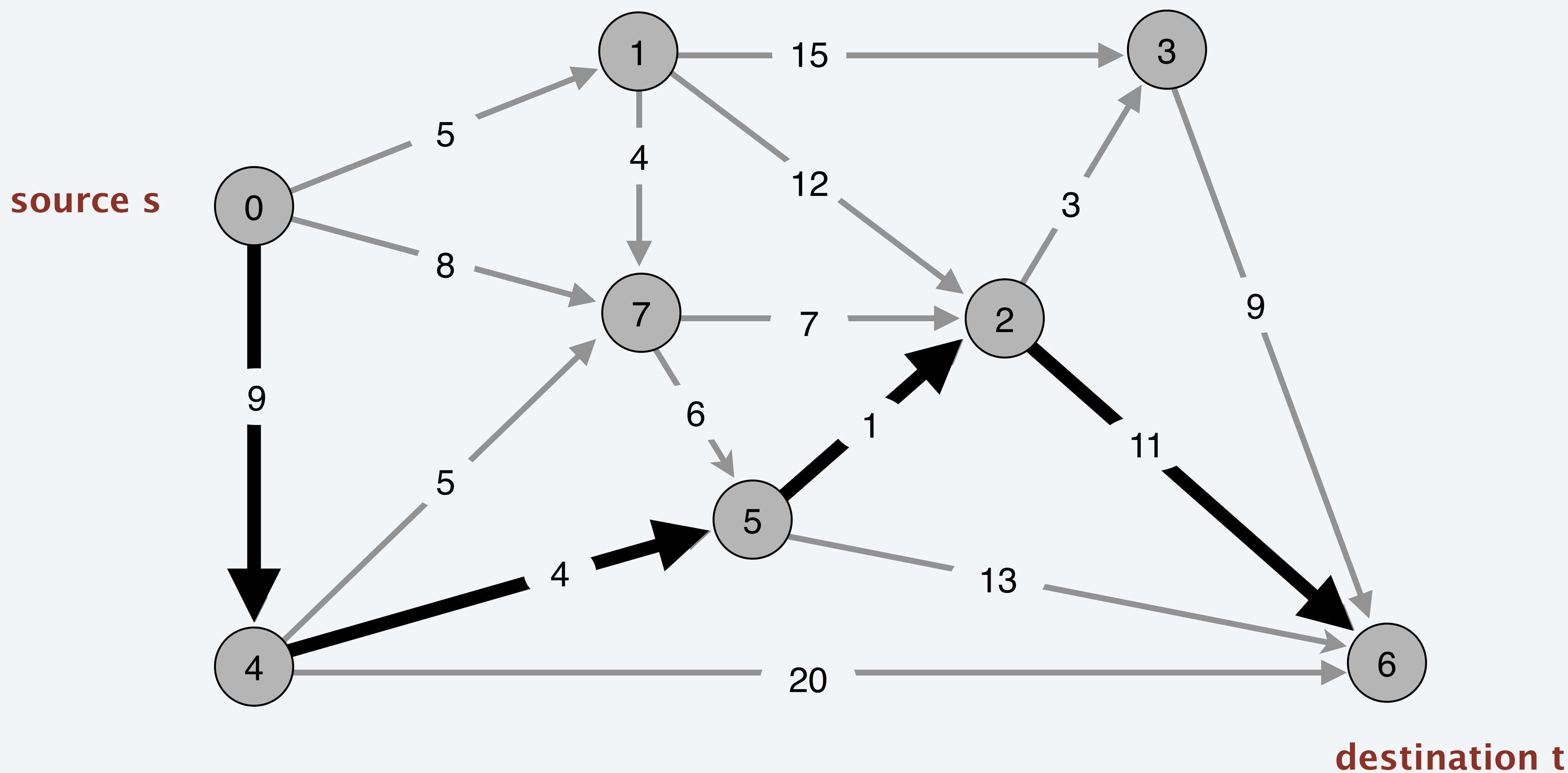


4. Greedy Algorithms II

- *Dijkstra′s algorithm*
- *minimum spanning trees*
- *Prim, Kruskal*
- *Union-Find Structure*

Single-pair shortest path problem

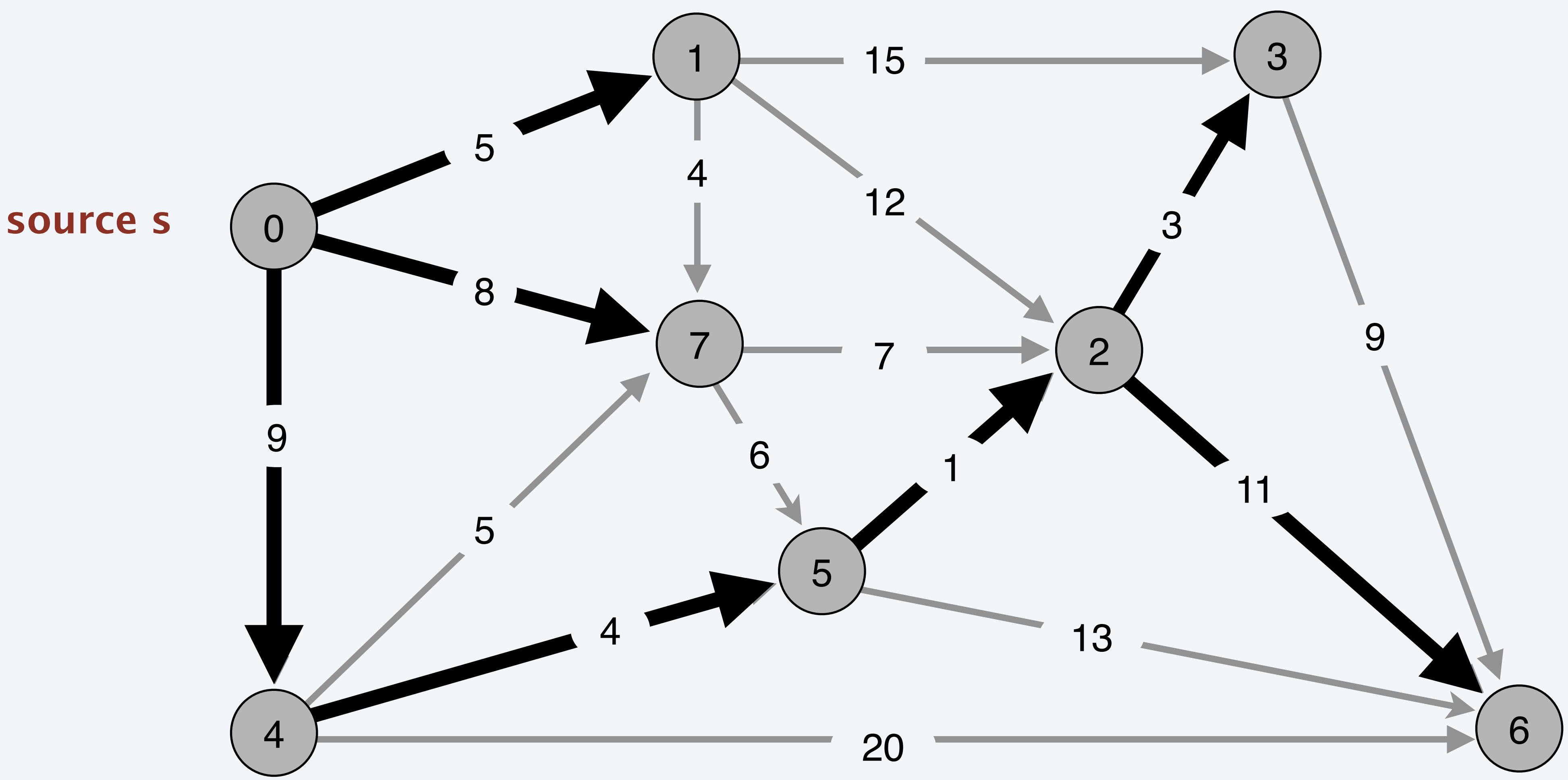
Problem. Given a digraph $G = (V, E)$, edge lengths $\ell_e \geq 0$, source $s \in V$, and destination $t \in V$, find a shortest directed path from s to t.



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

Car navigation

Q. Which kind of shortest path problem?

A. Single-destination shortest paths problem.

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.

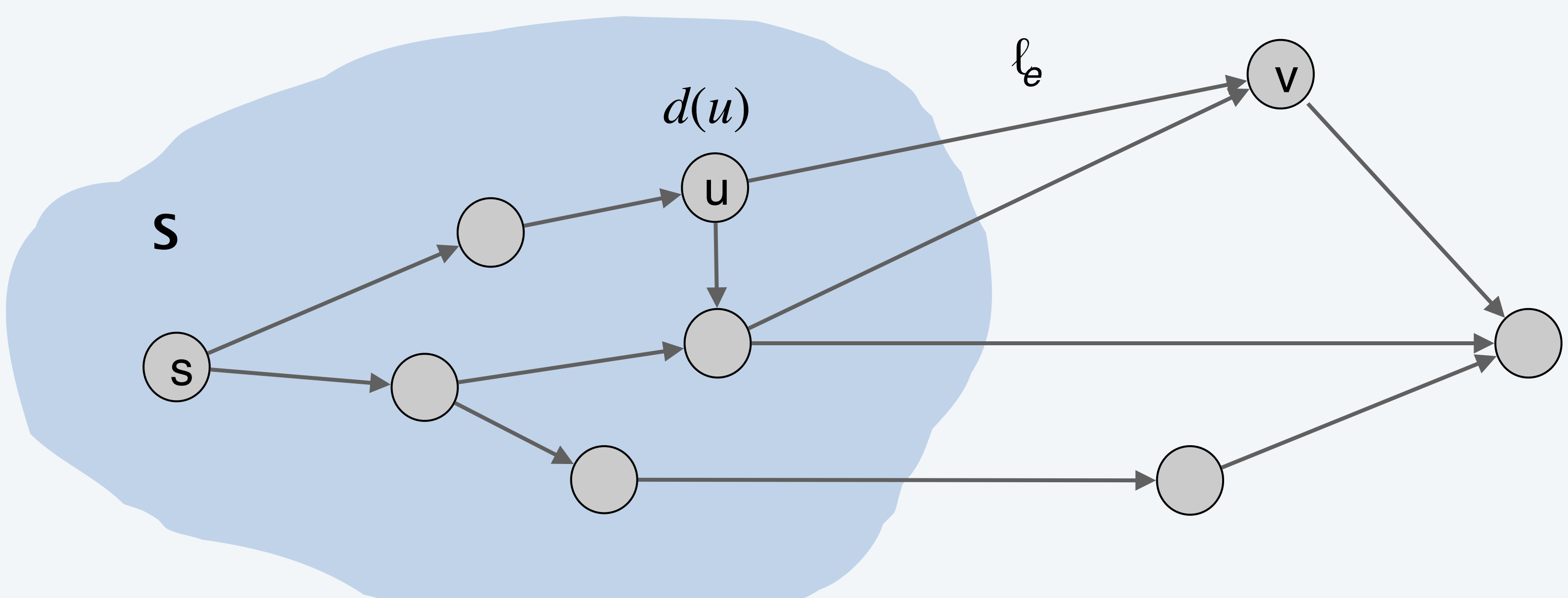
Dijkstra's algorithm

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.

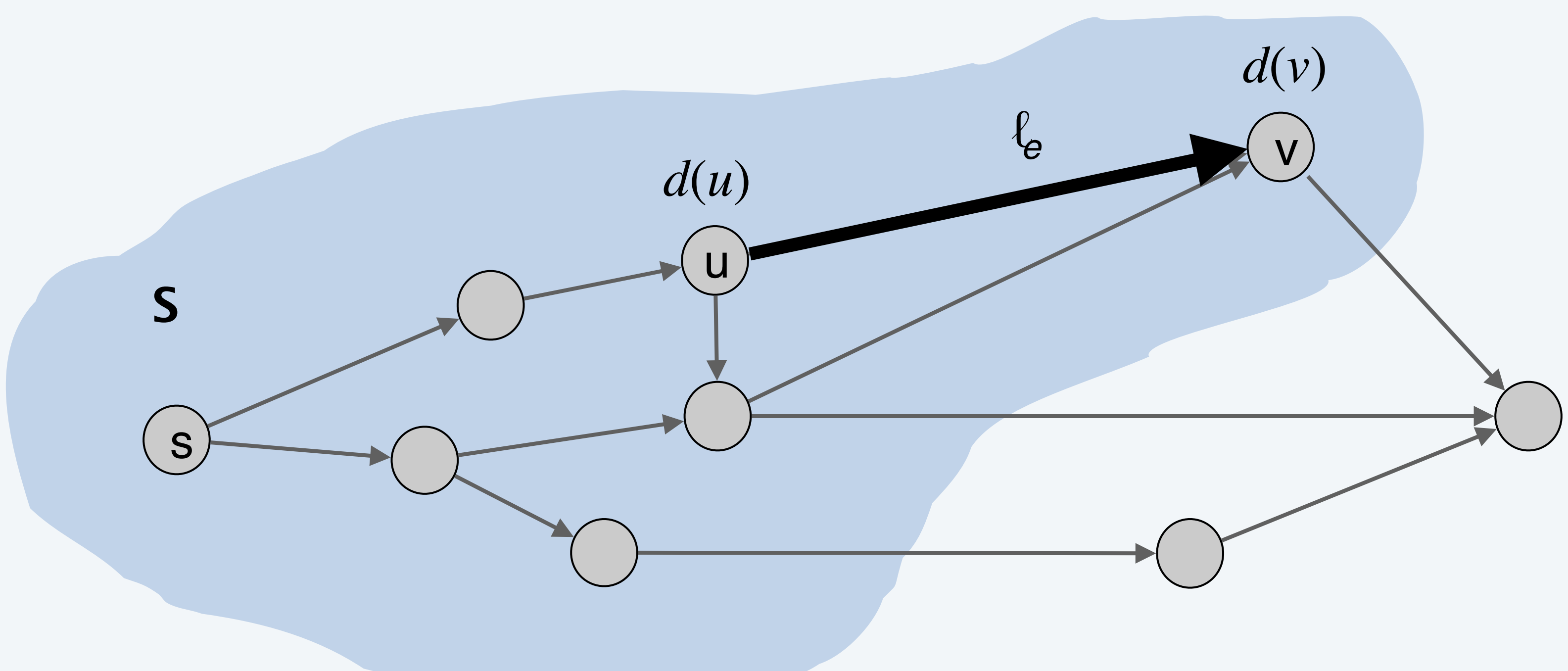
- Initialize $S = \{ s \}$, $d(s) = 0$.
- Repeatedly choose unexplored node v which minimizes

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

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

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

- To recover path, set $pred(v) = (u, v)$ that achieves min.



Dijkstra's algorithm: proof of correctness

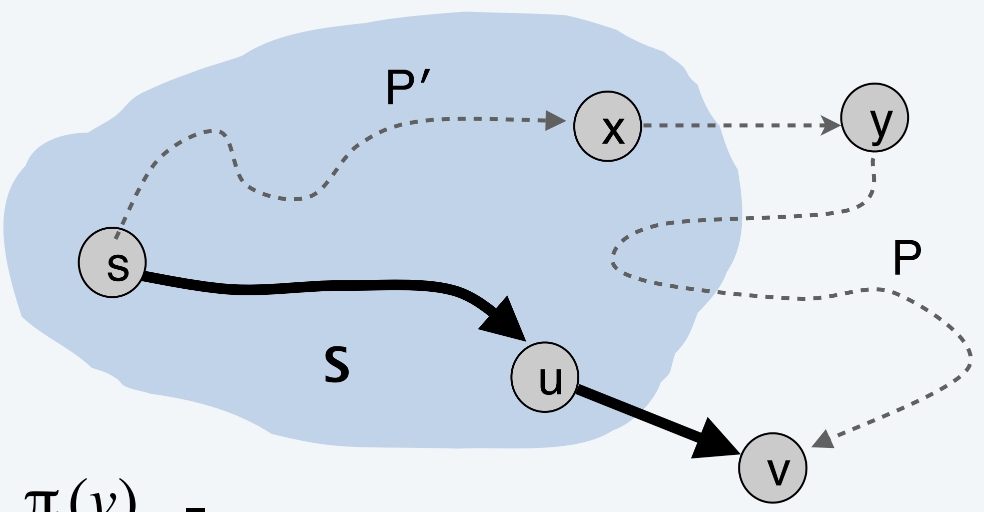
Invariant. For each node $u \in S$, $d(u)$ is the length of a shortest $s \rightsquigarrow u$ path.

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.
- A 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.



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

- $\ell(P) \geq \ell(P') + \ell(x, y)$: non-negative lengths
- $\geq d(x) + \ell(x, y)$: inductive hypothesis
- $\geq \pi(y)$: definition of $\pi(y)$
- $\geq \pi(v)$: Dijkstra chose v instead of y

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 an unexplored node that minimizes $\pi(v)$.

Dijkstra's algorithm: efficient implementation

Implementation.

- Algorithm stores $\pi(v)$ for each node v.
- Priority queue stores $\pi(v)$ for each unexplored node v.
- Recall that $\pi(v) = d(v)$ once vertex is deleted from priority queue.

DIJKSTRA (V, E, ℓ, s)

$pq \leftarrow$ CREATE-PRIORITY-QUEUE().

FOREACH $v \neq s$: $\pi(v) \leftarrow \infty$; $\pi(s) \leftarrow 0$.

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

WHILE (IS-NOT-EMPTY(pq))

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

  FOREACH *edge* $(u, v) \in E$ *leaving* u:

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

      DECREASE-KEY$(pq, v, \pi(u) + \ell(u, v))$.

      $\pi(v) \leftarrow \pi(u) + \ell(u, v)$; $pred(v) \leftarrow (u, v)$.

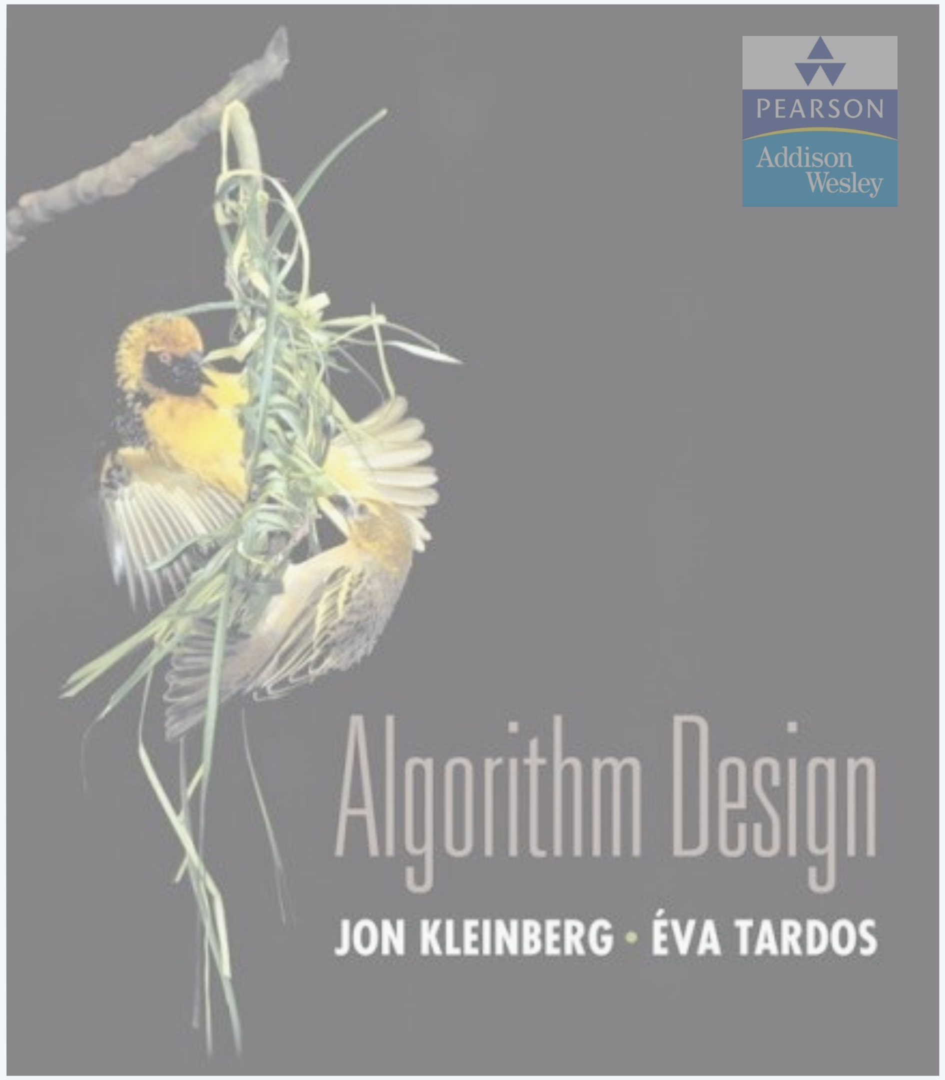
Dijkstra's algorithm: which priority queue?

Performance. Depends on priority queue: n INSERT, n DELETE-MIN, m DECREASE-KEY.

- Array implementation optimal for dense graphs.
- Binary heap much faster for sparse graphs.

priority queue implementation	INSERT	DELETE-MIN	DECREASE-KEY	total
unordered array	$O(1)$	$O(n)$	$O(1)$	$O(n^2)$
binary heap	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(m \log n)$

† amortized



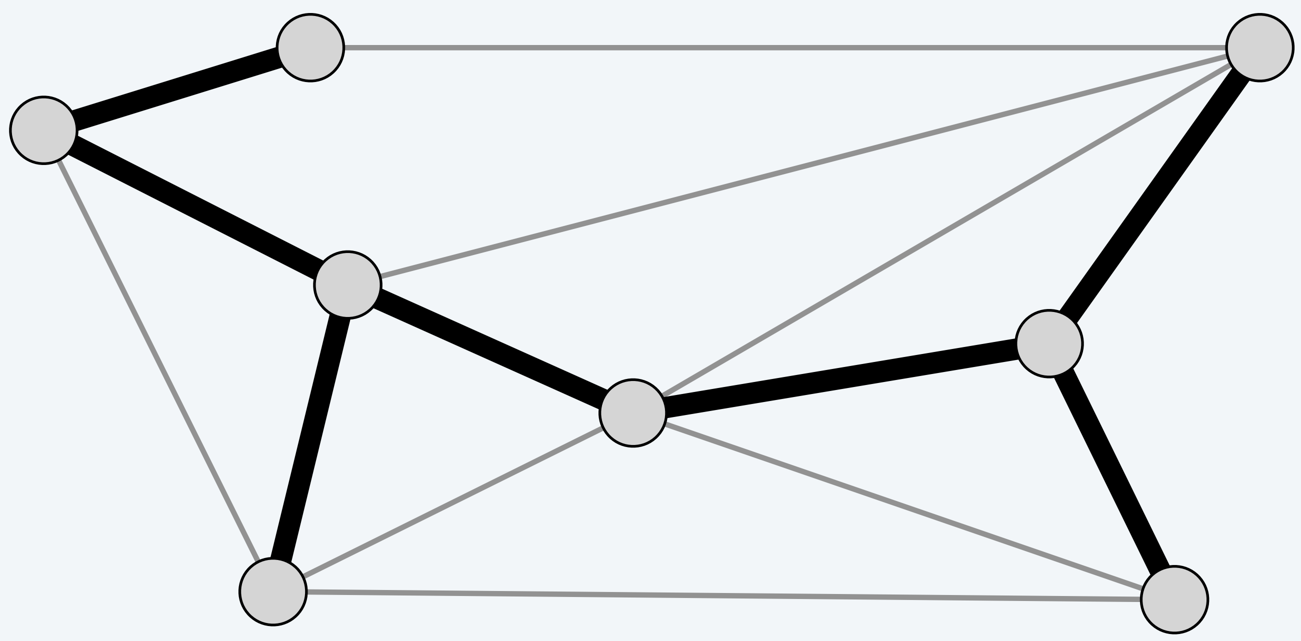
4. Greedy Algorithms II

- *Dijkstra's algorithm*
- *minimum spanning trees*
- *Prim, Kruskal*
- *Union-Find Structure*

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.

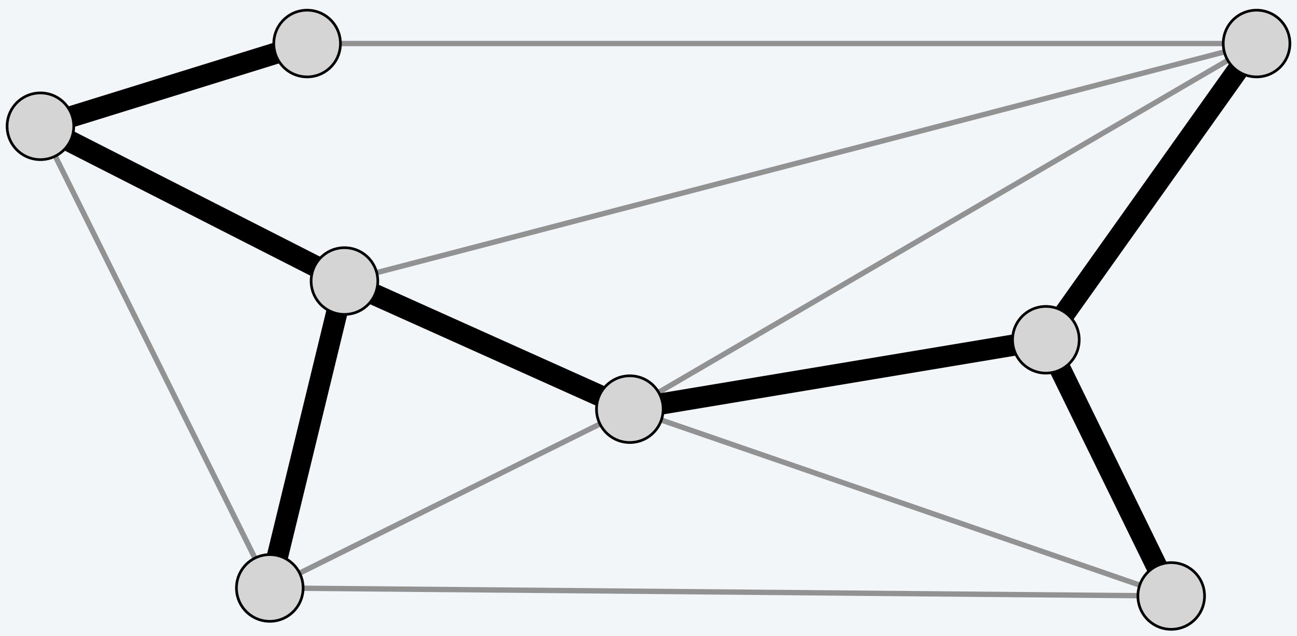


H = (V, T) is a spanning tree of G = (V, E)

Spanning tree properties

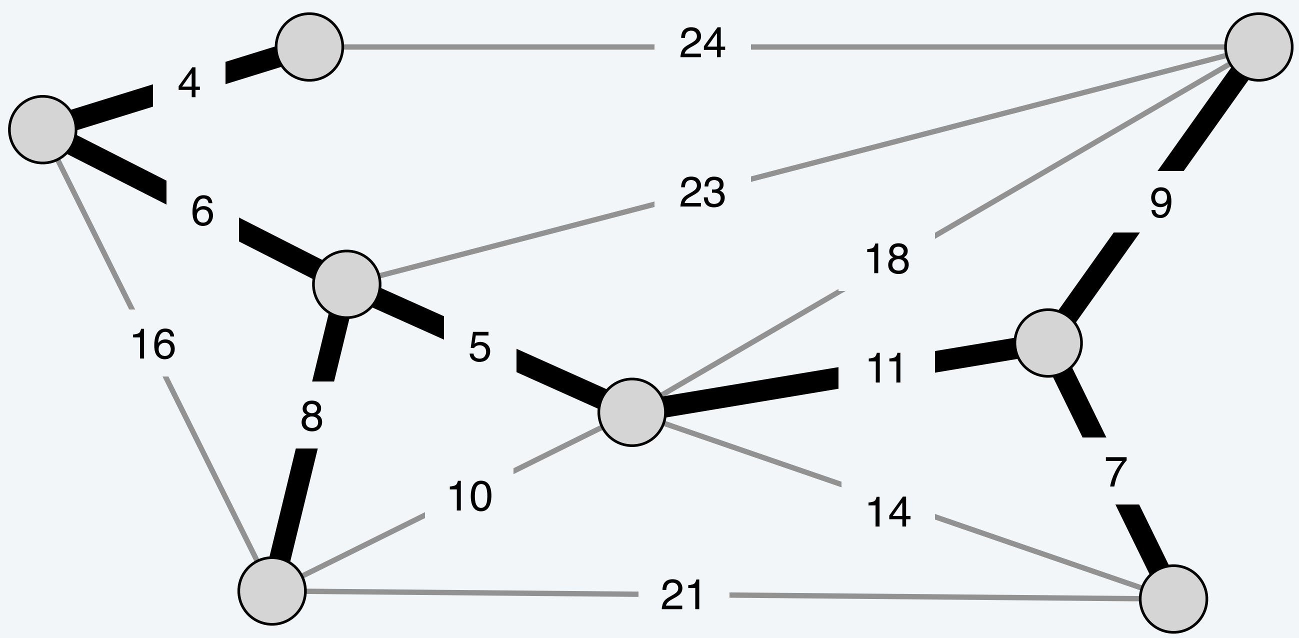
Proposition. Let $H = (V, T)$ be a subgraph of an undirected graph $G = (V, E)$. Then, the following are equivalent:

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



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.



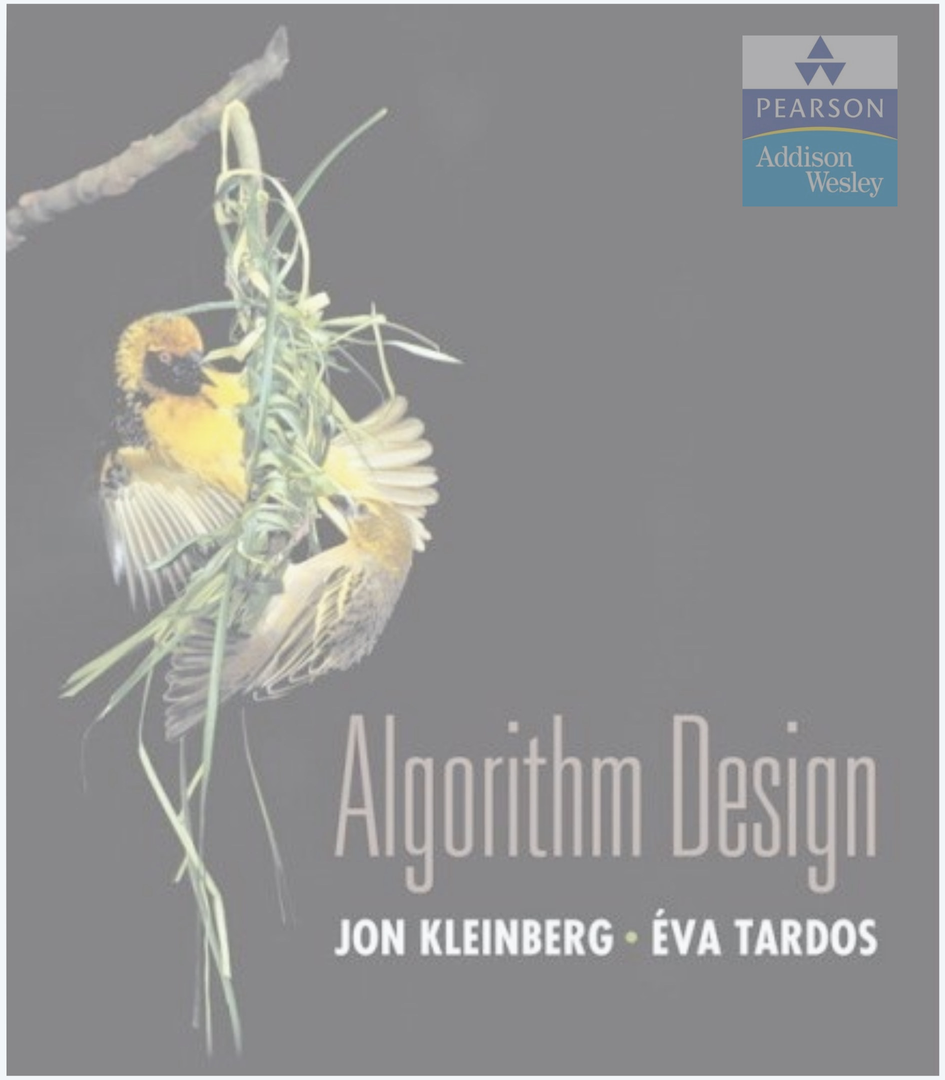
MST cost = 50 = 4 + 6 + 8 + 5 + 11 + 9 + 7

Cayley's theorem. There are n^{n-2} spanning trees of complete graph on n vertices. ← 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).



4. Greedy Algorithms II

- *Dijkstra's algorithm*
- *minimum spanning trees*
- *Prim, Kruskal*
- *Union-Find Structure*

Prim's algorithm

Initialize S = any node, $T = \varnothing$.

Repeat $n - 1$ times:

- Add to T a min-weight edge with one endpoint in S.
- Add new node 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)

Create an empty priority queue pq.

$T \leftarrow \varnothing$.

$s \leftarrow$ *any node in* V.

FOREACH $v \neq s$: $\pi(v) \leftarrow \infty$; $\pi(s) \leftarrow 0$.

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

WHILE (IS-NOT-EMPTY(pq))

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

  $T \leftarrow T \cup pred(u)$.

  FOREACH *edge* $(u, v) \in E$ *incident to* u:

    IF $\pi(v) > c(u, v)$

      DECREASE-KEY$(pq, v, c(u, v))$.

      $\pi(v) \leftarrow c(u, v)$; $pred(v) \leftarrow (u, v)$.

π(v) = weight of cheapest known edge between v and S

Kruskal's algorithm

Consider edges in ascending order of weight:

- Add to tree unless it would create a cycle.

Kruskal's algorithm: implementation

Theorem. Kruskal's algorithm can be implemented to run 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 $c(e_1) \leq c(e_2) \leq \ldots \leq c(e_m)$.

$T \leftarrow \varnothing$.

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

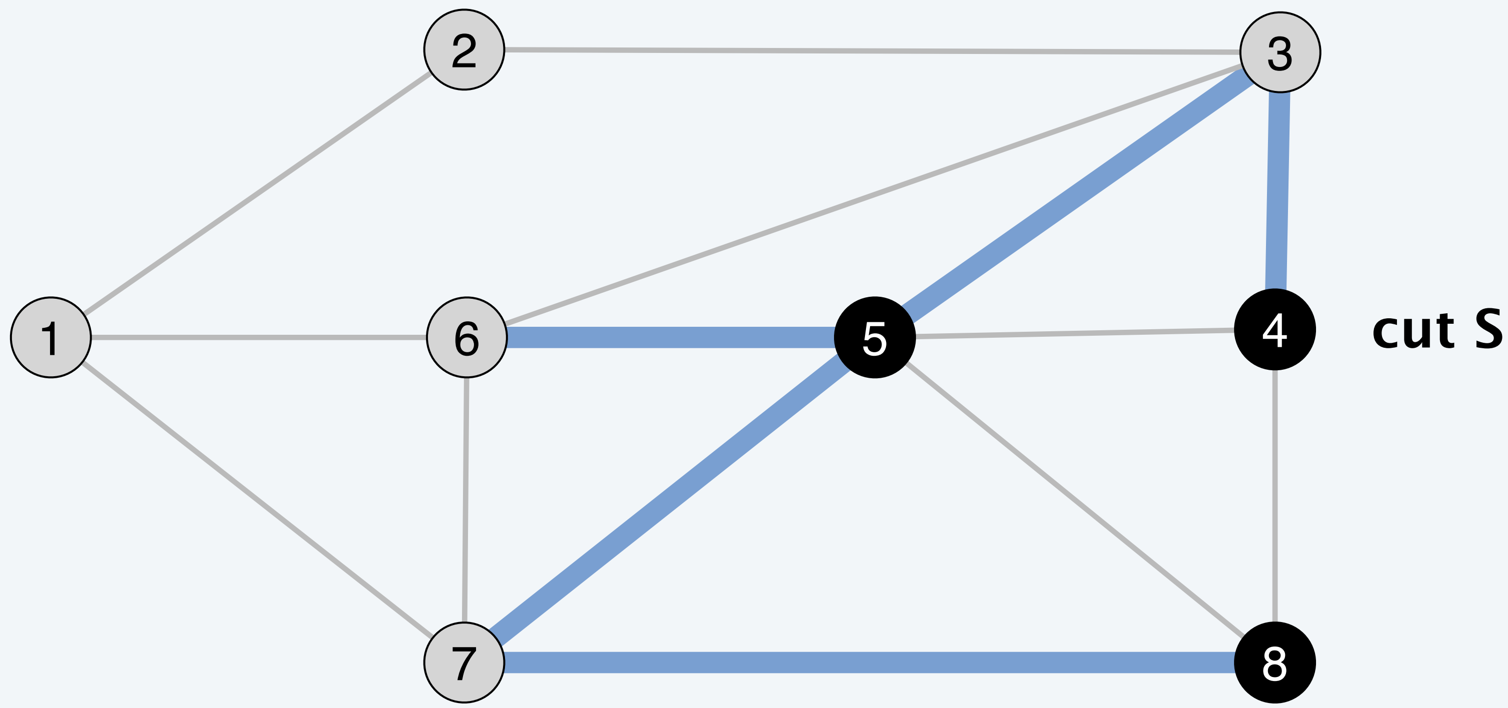
Consider edges in descending order of weight:

- Remove edge unless it would disconnect the graph.

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.

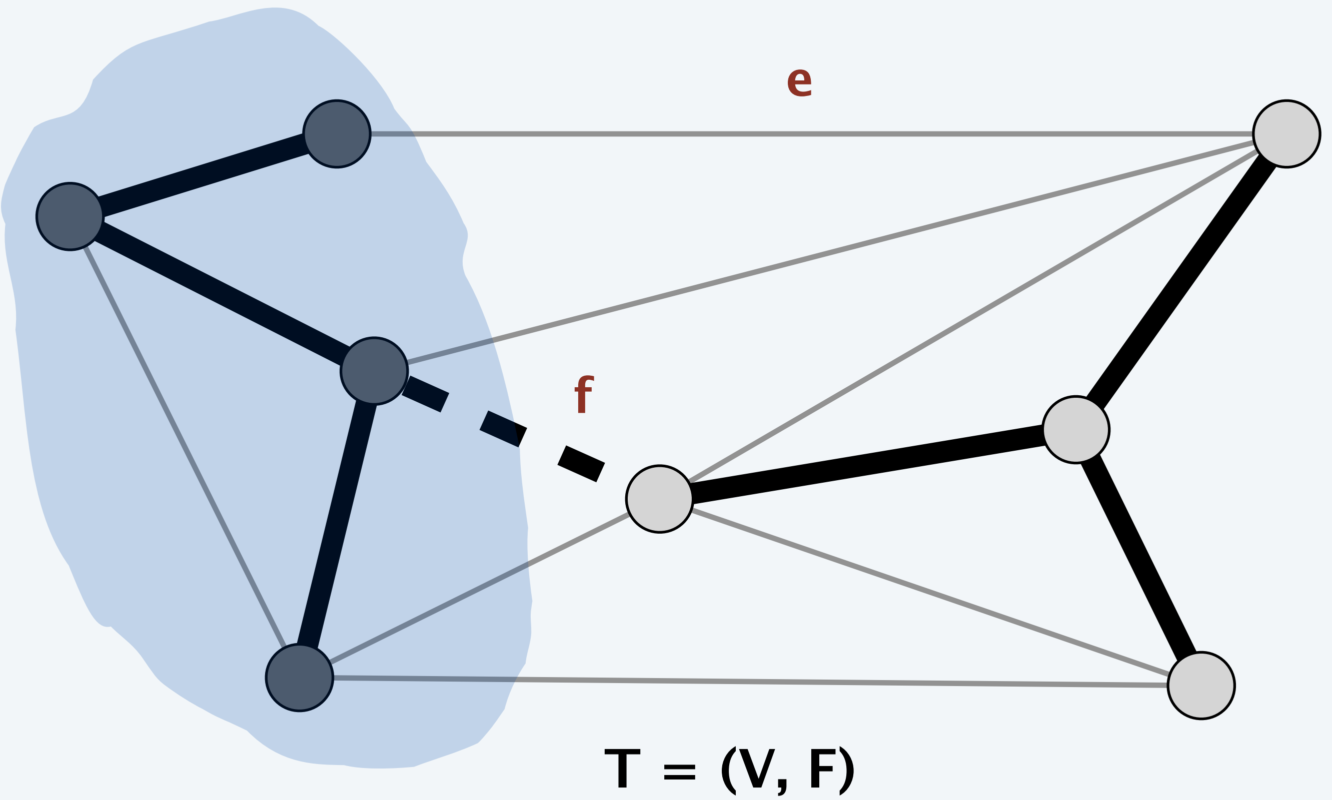


cutset D = { (3, 4), (3, 5), (5, 6), (5, 7), (8, 7) }

Fundamental cutset

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

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

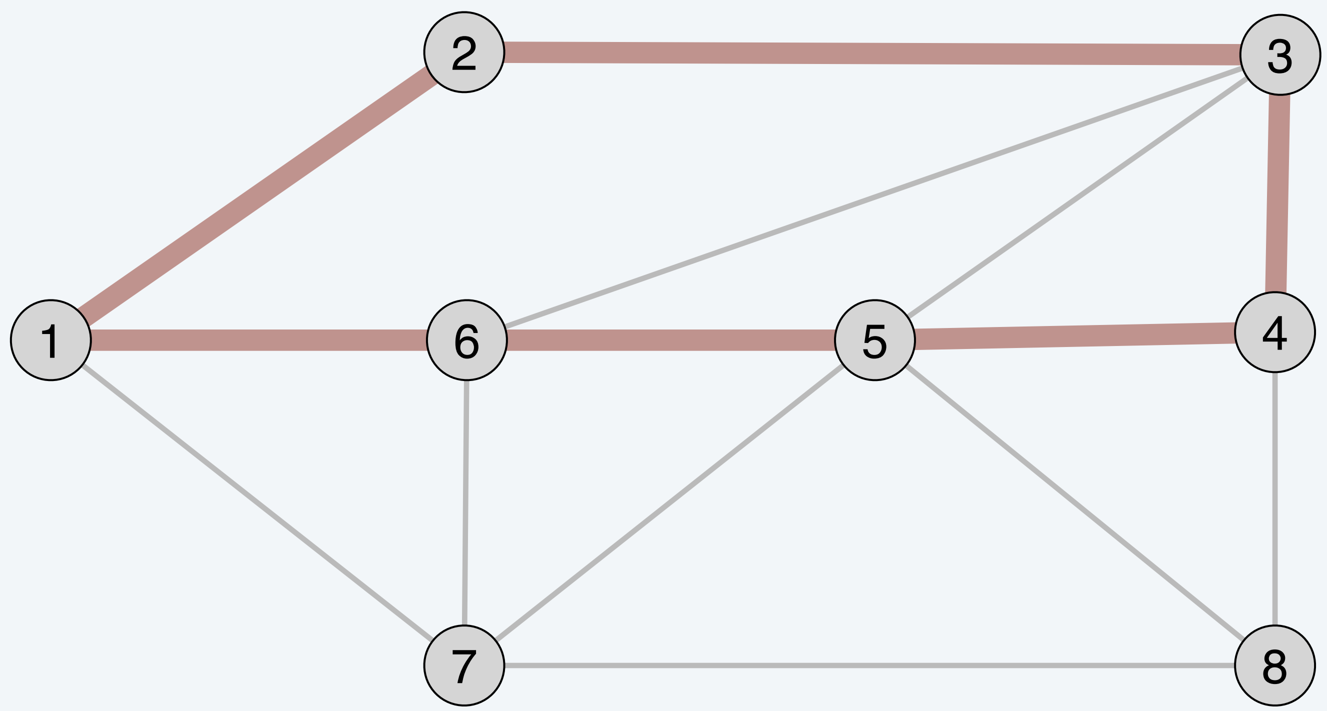


Observation. If $c_e < c_f$, then (V, T) is not an MST.

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.

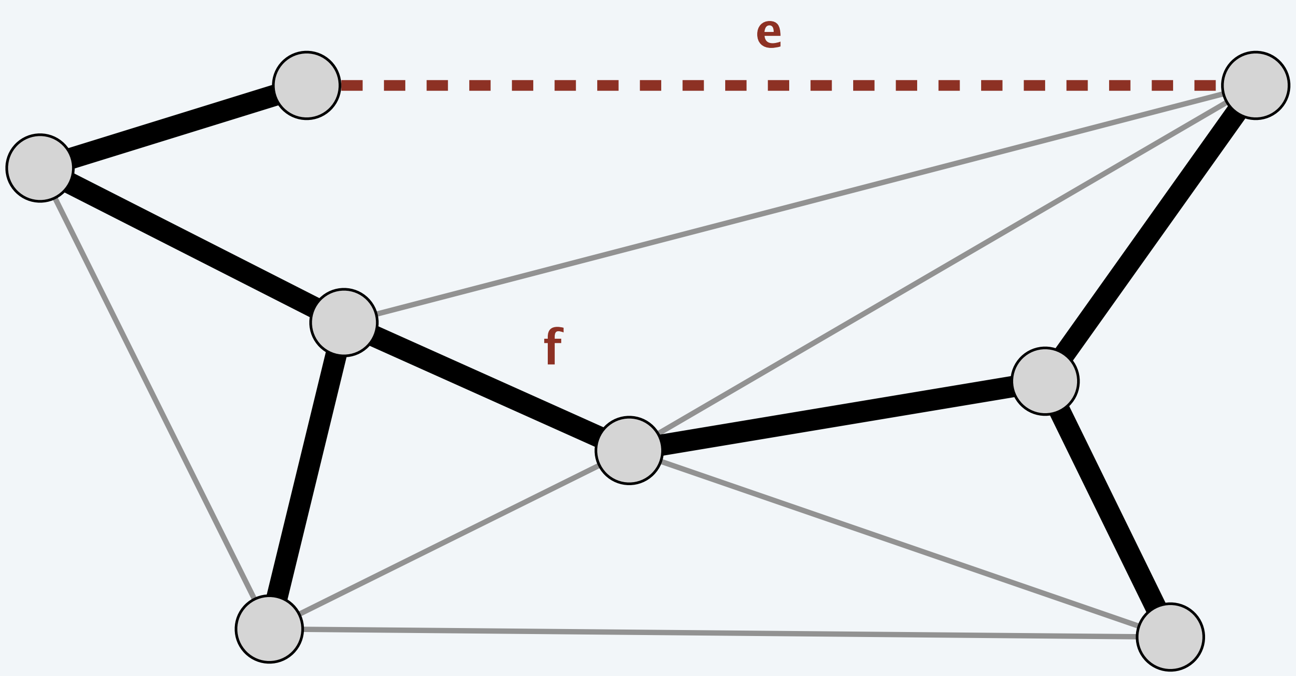


cycle C = { (1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 1) }

Fundamental cycle

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

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



T = (V, F)

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