



南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Algorithm Design and Analysis (H)

CS216

Instructor: Shan CHEN (陈杉)

chens3@sustech.edu.cn

(slides edited from Prof. Shiqi Yu)



南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Greedy Algorithms



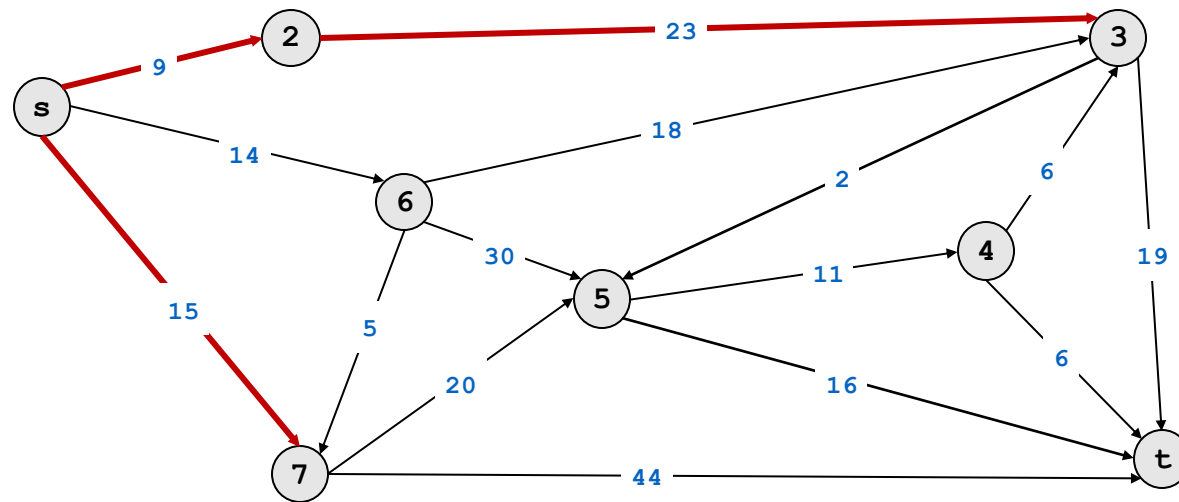
4. Shortest Paths in a Graph



Single-Source Shortest Path Problem

- **Single-source shortest path problem.**

- **Directed** graph $G = (V, E)$ with **non-negative** edge costs.
- Source s . *undirected graph is usually easier*
- ℓ_e = length of edge e . *path length = sum of edge lengths on path*
- Goal: find a **shortest** directed path from s to every node.



shortest path from s to 3: $9 + 23 = 32$
shortest path from s to 7: 15



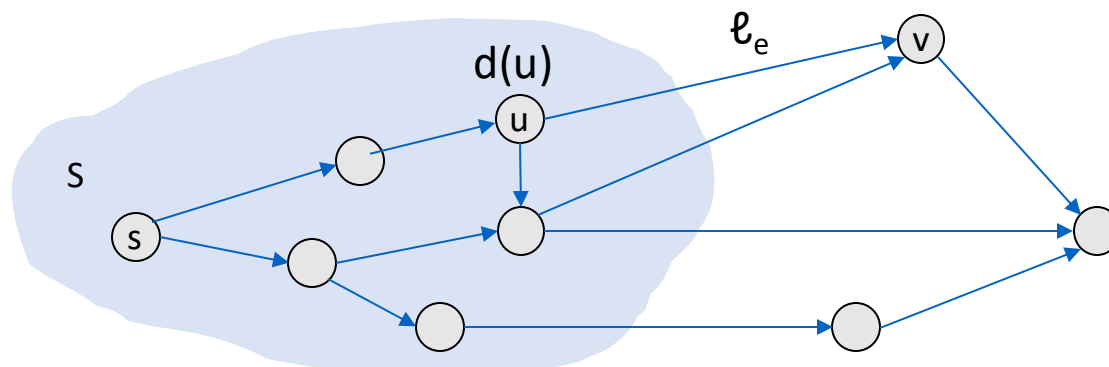
Dijkstra's Algorithm

- **Greedy approach.**

- Maintain a set S of **explored nodes** for which we have determined the shortest path distance $d(u)$ from s to u .
- Initialize $S = \{s\}$, $d(s) = 0$.
- Repeatedly choose unexplored node v which minimizes

$$\pi(v) = \min_{e = (u, v) : u \in S} d(u) + \ell_e \quad \leftarrow \text{shortest path to some } u \text{ in explored part, followed by a single edge } (u, v)$$

- Add v to S and set $d(v) = \pi(v)$.





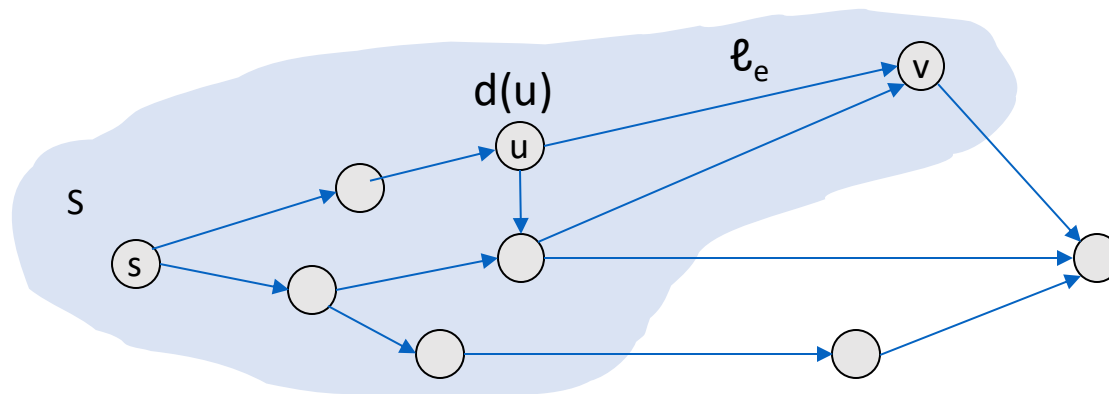
Dijkstra's Algorithm

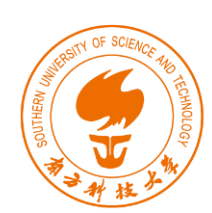
- **Greedy approach.**

- Maintain a set S of **explored nodes** for which we have determined the shortest path distance $d(u)$ from s to u .
- Initialize $S = \{s\}$, $d(s) = 0$.
- Repeatedly choose unexplored node v which minimizes

$$\pi(v) = \min_{e = (u, v) : u \in S} d(u) + \ell_e \quad \leftarrow \text{shortest path to some } u \text{ in explored part, followed by a single edge } (u, v)$$

- Add v to S and set $d(v) = \pi(v)$.





Dijkstra's Algorithm: Proof of Correctness

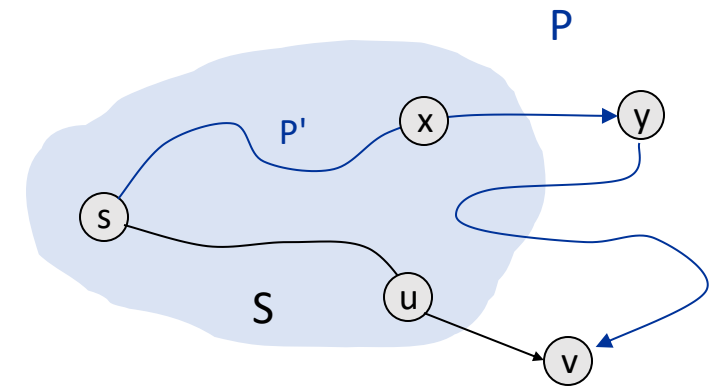
- **Invariant.** For each $u \in S$, $d(u)$ is the length of the shortest s - u path.
- Pf. (by induction on $|S|$)
 - Base case: $|S| = 1$ is trivial.
 - Inductive hypothesis: Assume true for $|S| \geq 1$.
 - Let v be next node added to S and let u - v be the chosen edge.
 - The shortest s - u path plus (u, v) is an s - v path of length $\pi(v)$.
 - Consider any s - v path P . 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 leaves S .
$$\ell(P) \geq \ell(P') + \ell(x, y) \geq d(x) + \ell(x, y) \geq \pi(y) \geq \pi(v)$$

↑
nonnegative
weights

↑
inductive
hypothesis

↑
definition
of $\pi(y)$

↑
definition
of $\pi(v)$





Dijkstra's Algorithm: Implementation

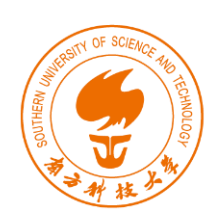
- For each unexplored node, explicitly maintain $\pi(v) = \min_{e=(u,v): u \in S} d(u) + \ell_e$.
 - Next node to explore = node with minimum $\pi(v)$.
 - When exploring v , for each incident edge $e = (v, w)$, update

$$\pi(w) = \min \{ \pi(w), \pi(v) + \ell_e \}.$$

- Maintain a priority queue of unexplored nodes, prioritized by $\pi(v)$.

PQ Operation	Dijkstra	Array	Binary heap	d-way Heap	Fibonacci heap [†]
Insert	n	n	log n	$d \log_d n$	1
ExtractMin	n	n	log n	$d \log_d n$	log n
ChangeKey	m	1	log n	$\log_d n$	1
IsEmpty	n	1	1	1	1
Total		n^2	$m \log n$	$m \log_{m/n} n$	$m + n \log n$

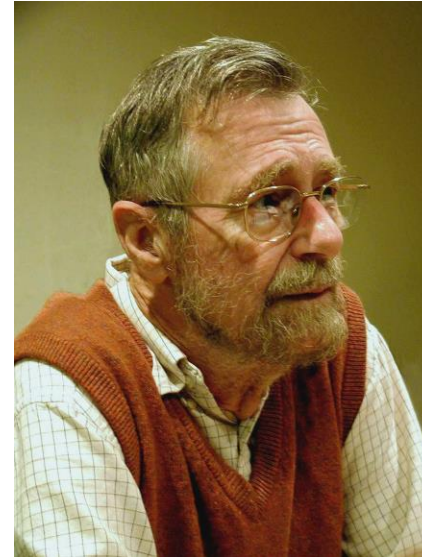
[†] Individual ops are amortized bounds

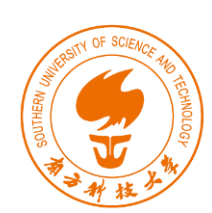


Dijkstra's Algorithm: History

" 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 W. Dijkstra



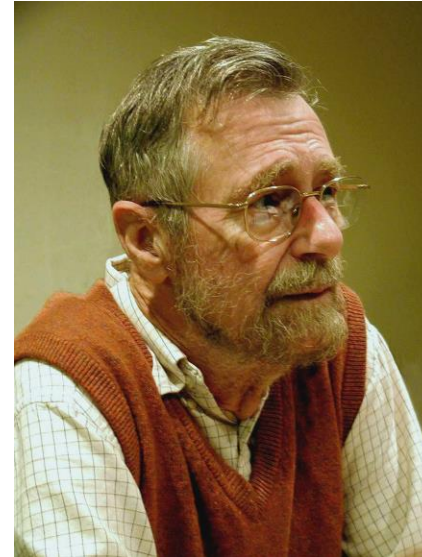


More about Edsger W. Dijkstra

Dijkstra was immensely influential in many fields of computing: compilers, operating systems, concurrent programming, software engineering, programming languages, algorithm design, and teaching (among others!)

It would be hard to pin down what he is most famous for because he has influenced so much CS.

Dijkstra was also influential in making programming more structured -- he wrote a seminal paper titled, "Goto Considered Harmful" where he lambasted the idea of the "goto" statement (which exists in C++ -- you will rarely, if ever, use it!)



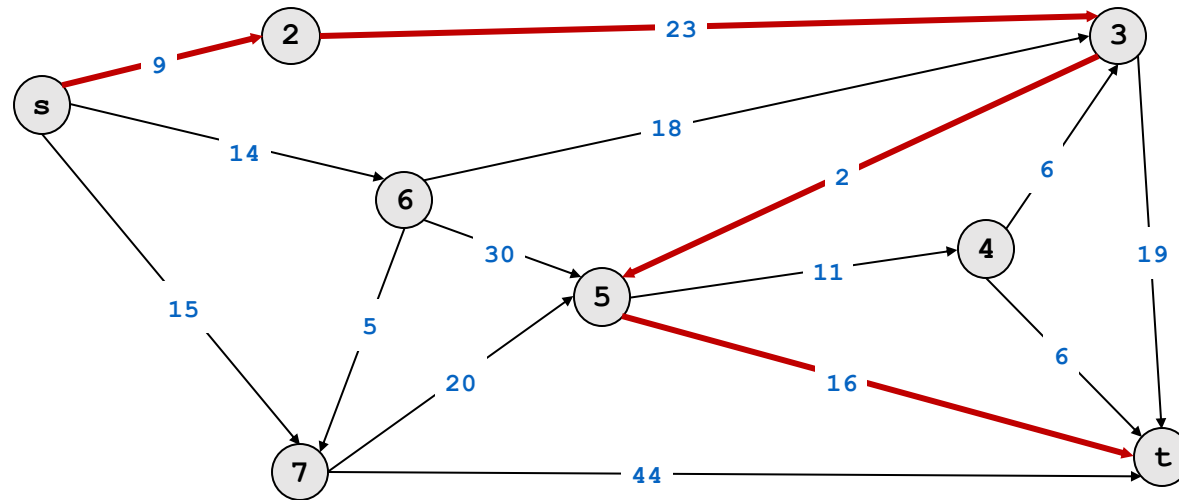


Single-Pair Shortest Path Problem

- **Single-pair shortest path problem.**

- Directed graph $G = (V, E)$ with non-negative edge costs.
- Source s , destination t .
- ℓ_e = length of edge e .
- Goal: find a shortest directed path **from s to t** .

only one destination is considered



shortest path from s to t :
 $9 + 23 + 2 + 1 = 50$



Single-Pair Shortest Path Problem

- **Single-pair shortest path problem.**

- Directed graph $G = (V, E)$ with non-negative edge costs.
- Source s , destination t .
- ℓ_e = length of edge e .
- Goal: find a shortest directed path **from s to t** . ← only one destination is considered

- **Q.** Can we beat Dijkstra when we have only one destination?

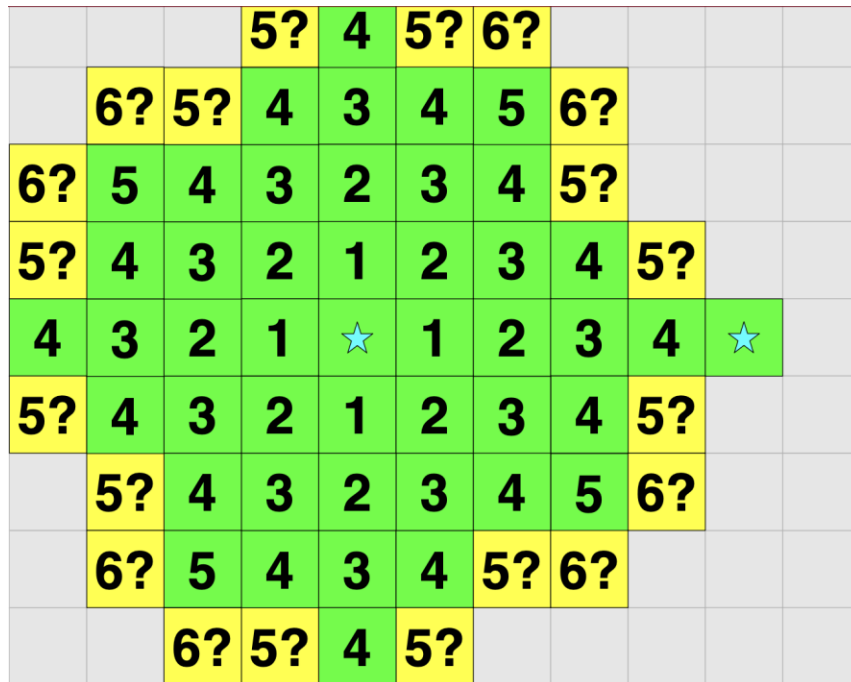
- E.g., if traveling from Beijing to Shanghai, we will go south.
- Solution: add some **heuristic information** to guide the search, which could be direction in the case of a street map.



Single-Pair Shortest Path Problem

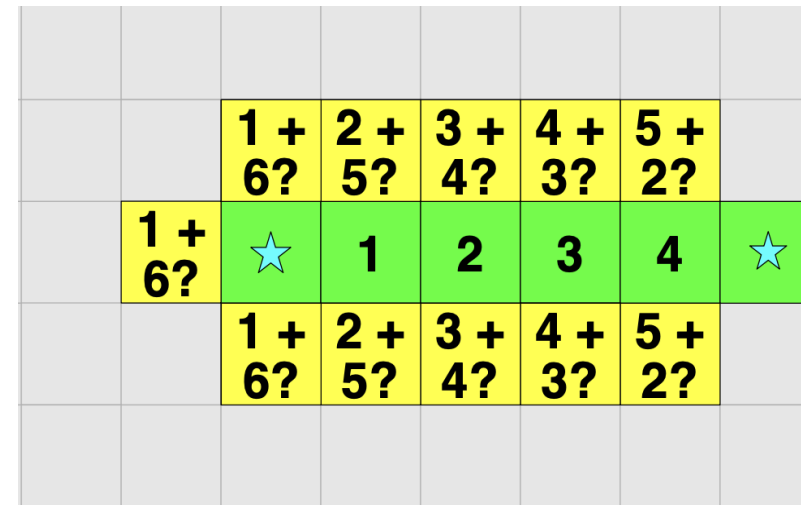
- **Dijkstra's priority:**

➤ s-v distance $\pi(v)$



- **Ideal priority:**

➤ $\pi(v) + v-t$ distance $d(v, t)$



we should prioritize
search to the right





A* Search Algorithm

- **A* priority:**

- s-v distance $\pi(v)$ (same as Dijkstra) + **heuristic v-t distance $h(v, t)$**

- **Q.** How do we estimate the true v-t distance $d(v, t)$?

- E.g., hamming distance, Euclidean distance, etc.

- **Choice of $h(v, t)$:**

- $h(v, t) = 0$: same as Dijkstra
- **$h(v, t) < d(v, t)$: same or faster than Dijkstra (and shortest path guaranteed)**
- $h(v, t) = d(v, t)$: fastest, but requires perfect knowledge (the shown example)
- $h(v, t) > d(v, t)$: won't necessarily find the shortest path (but might run faster)

- **Takeaway:** Always **underestimate** the true future cost $d(v, t)$ for A*.





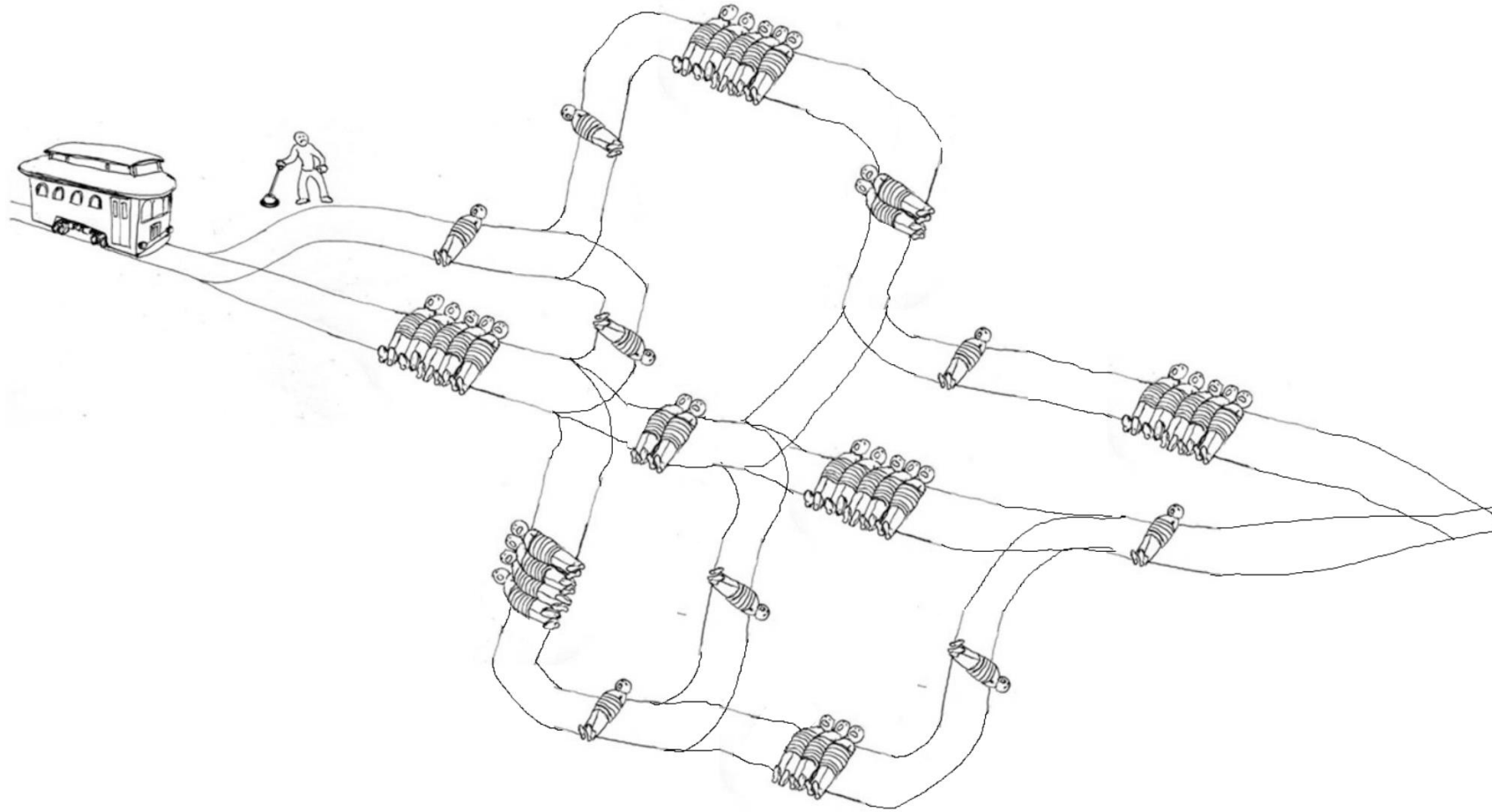
- ## Dijkstra:

A*: ($h(v, t)$ = Hamming distance)

[illegible]

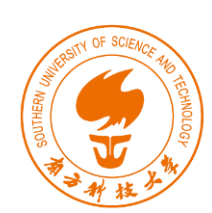


Shortest Path Problem: Moral Implications





5. Minimum Spanning Trees



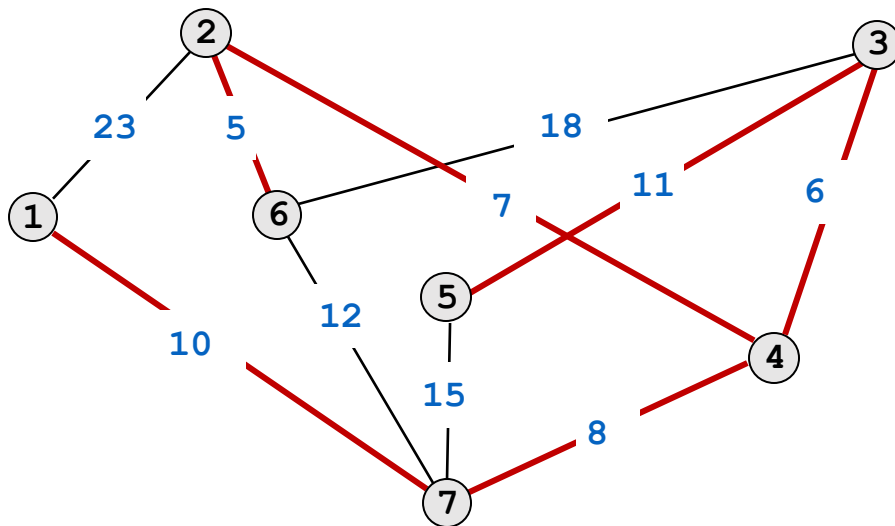
Spanning Trees

- **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.
- **Property.** All 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.



Minimum Spanning Trees (MSTs)

- **Def.** Given a connected, **undirected** graph $G = (V, E)$ with edge costs, a **minimum spanning tree (MST)** (V, T) is a spanning tree of G such that the sum of the edge costs in T is minimized.
- **Cayley's theorem:** K_n has n^{n-2} spanning trees. ($|V| = n$, $|E| = m$)
cannot solve by brute-force



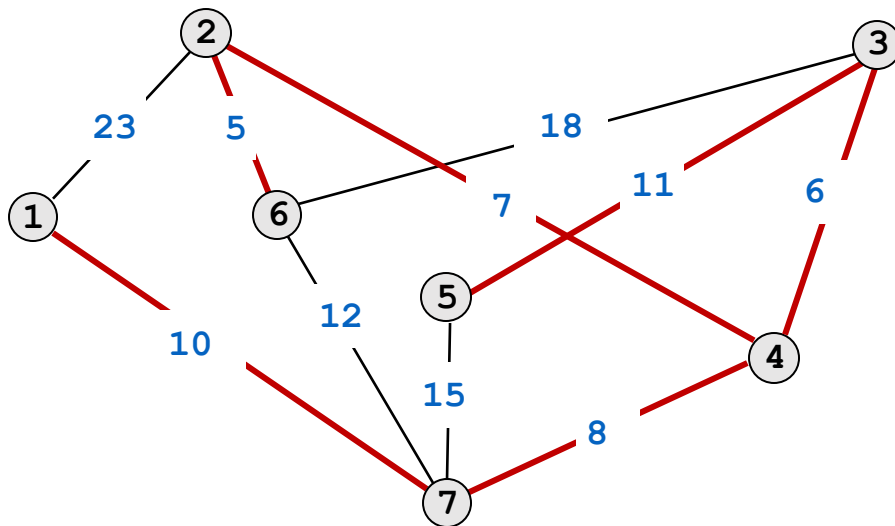
MST cost: $5 + 6 + 7 + 8 + 10 + 11 = 47$



Prim's Algorithm

- **Greedy approach.**

- Initialize $S = \{s\}$ for any node s and initialize edge set $T = \emptyset$.
- Repeat $n - 1$ times:
 - ✓ Add to T a **min-cost edge** ← extract-min with exactly one endpoint in S .
 - ✓ Add the other endpoint to S . ← change-key



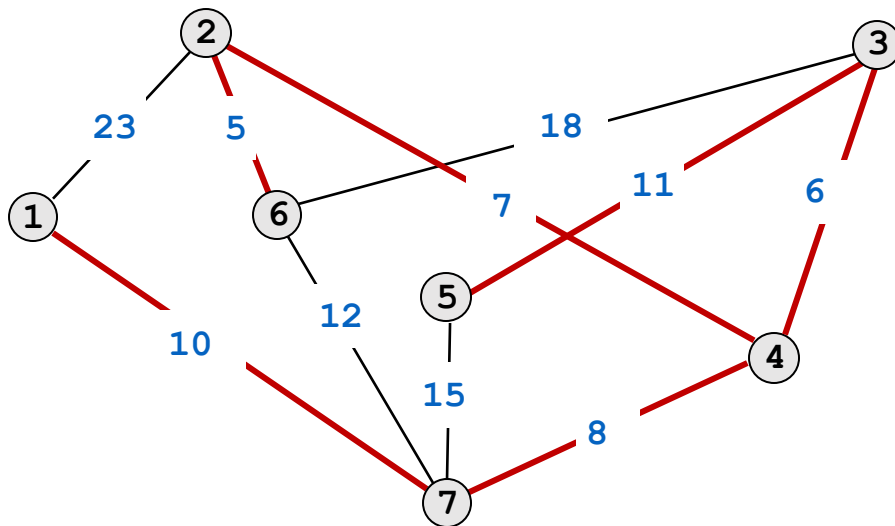
time complexity: $O(m \log n)$



Kruskal's Algorithm

- **Greedy approach.**

- Initialize edge set $T = \emptyset$.
- Sort edges in ascending order of cost.
- Repeat m times:
 - ✓ Add to T the considered edge unless it would create a cycle. ← union-find



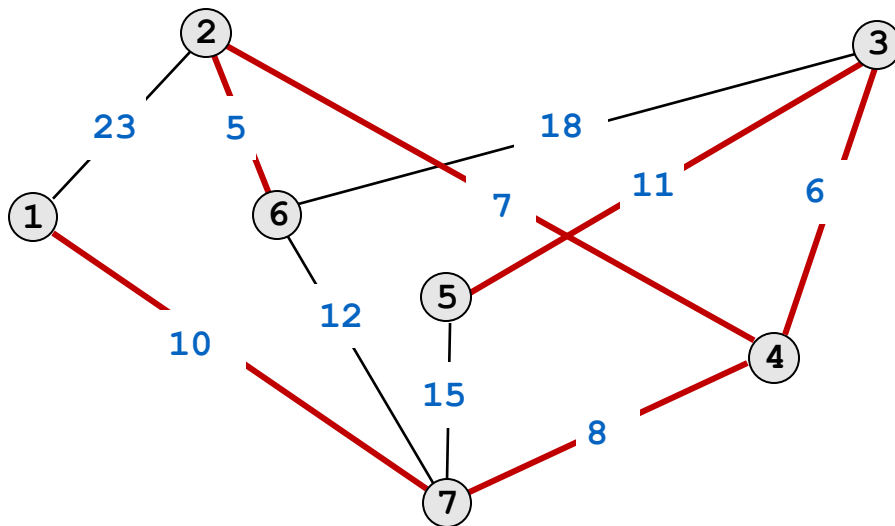
time complexity: $O(m \log m)$



Reverse-Delete Algorithm

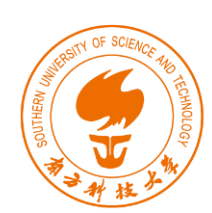
- **Greedy approach.**

- Initialize edge set $T = E$.
- Sort edges in descending order of cost.
- Repeat m times:
 - ✓ Delete from T the considered edge unless it would disconnect T .



time complexity: $O(m \log n (\log \log n)^3)$

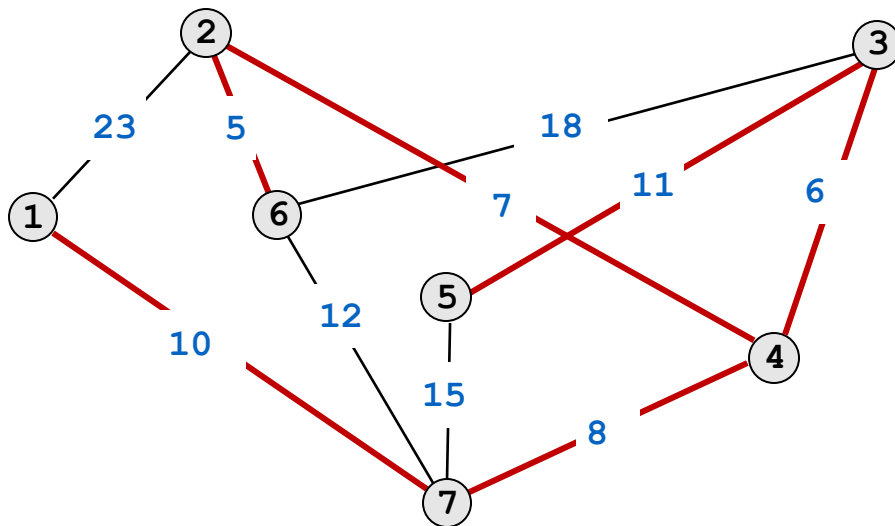
[Thorup 2000]



Borůvka's Algorithm

- **Greedy approach.**

- Initialize edge set $T = \emptyset$ (T has n connected components, one for each node).
- Repeat until only one connected component is left: $\leftarrow O(\log n)$ rounds, no need to sort
 - ✓ For each edge (u, v) , if u, v are in different components, use the cost of (u, v) to update the min-cost edge for both components.
 - ✓ Add to T the min-cost edge for each component.

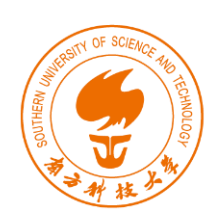


time complexity: $O(m \log n)$




Minimum Spanning Trees: Summary

- The learned MST greedy algorithms follow similar ideas and share roughly the same time complexity $O(m \log n)$.
 - Prim: extend a single connected component with a min-cost edge
 - Kruskal: extend connected components with a min-cost edge
 - Reverse-delete: remove a max-cost edge and maintain connected
 - Borůvka: extend connected components with a min-cost edge (without sorting)
- **Remark.** The above greedy algorithms can be extended to find **minimum spanning forests**.
- **Q.** Are there linear MST algorithms?



Minimum Spanning Trees: Linear Algorithms

- Linear **randomized** MST algorithms do exist! [Karger-Klein-Tarjan 1995]
- It is still open for **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	<i>asymptotically optimal</i>	Pettie–Ramachandran
20xx	$O(m)$??? 



6. Clustering

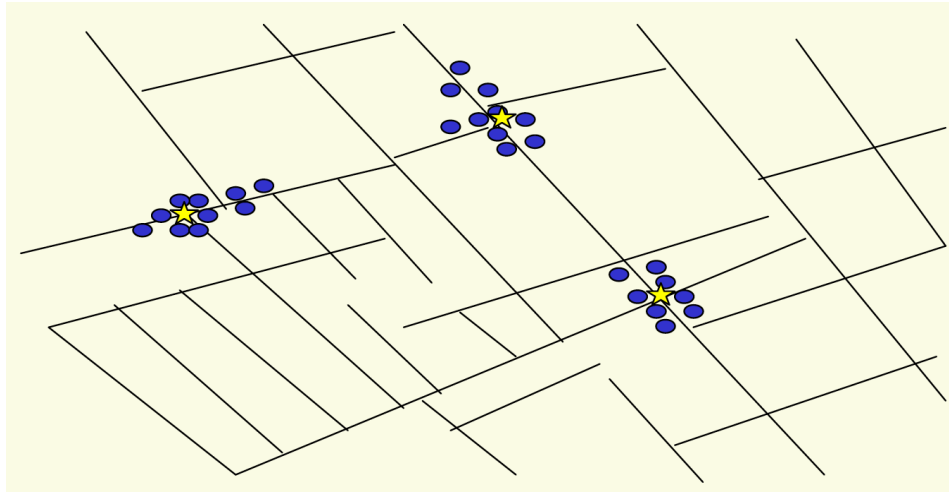


Clustering

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

e.g., photos, documents, etc.

w.r.t. some distance, e.g.,
number of pixels that differ
by some threshold



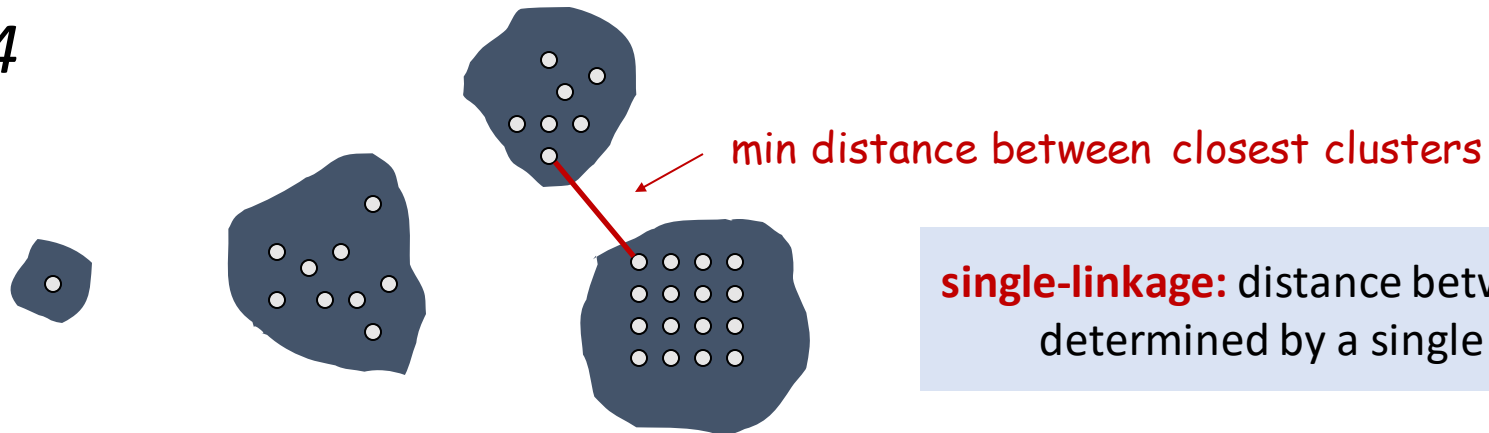
Outbreak of cholera deaths in London in 1850s. Reference: Nina Mishra, HP Labs

- **Applications.** Routing, categorization, similarity searching, etc.



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)
 - $d(p_i, p_j) \geq 0$ (nonnegativity)
 - $d(p_i, p_j) = d(p_j, p_i)$ (symmetry)
- **Spacing.** Min distance between any pair of points in different clusters.
- **Goal.** Given an integer k , find a k -clustering of maximum spacing.
- Ex. $k = 4$



single-linkage: distance between two clusters is determined by a single pair of objects



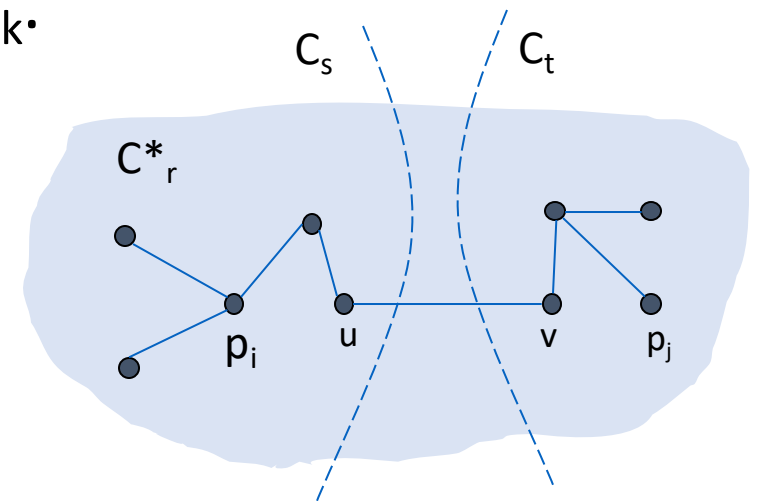
Single-Link k -Clustering: Greedy Algorithm

- **Greedy approach.**
 - Form a graph on the object set U , with n clusters in the beginning.
 - 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**, with a complete graph K_n where edge costs are distances (except we stop when there are k connected components).
- **Remark.** Equivalent to finding an MST and deleting the $k - 1$ most expensive edges.



Single-Link k -Clustering: Analysis

- **Theorem.** Let C^* be the clustering C_1^*, \dots, C_k^* formed by deleting the $k-1$ most expensive edges of an MST. C^* is a k -clustering of max spacing.
- Pf. Let C denote any other k -clustering C_1, \dots, C_k .
 - Let p_i, p_j be in the same cluster in C^* , say C_r^* , but in different clusters in C , say C_s and C_t .
 - Some edge (u, v) on p_i – p_j path in C_r^* spans two different clusters in C .
 - Spacing of $C^* = \text{length } d^*$ of the $(k-1)$ -st longest edge in the corresponding MST.
 - All edges on p_i – p_j path have length $\leq d^*$ since Kruskal already added them.
 - Spacing of C is $\leq d^*$ since u and v are in different clusters in C and $d(u, v) \leq d^*$. ■



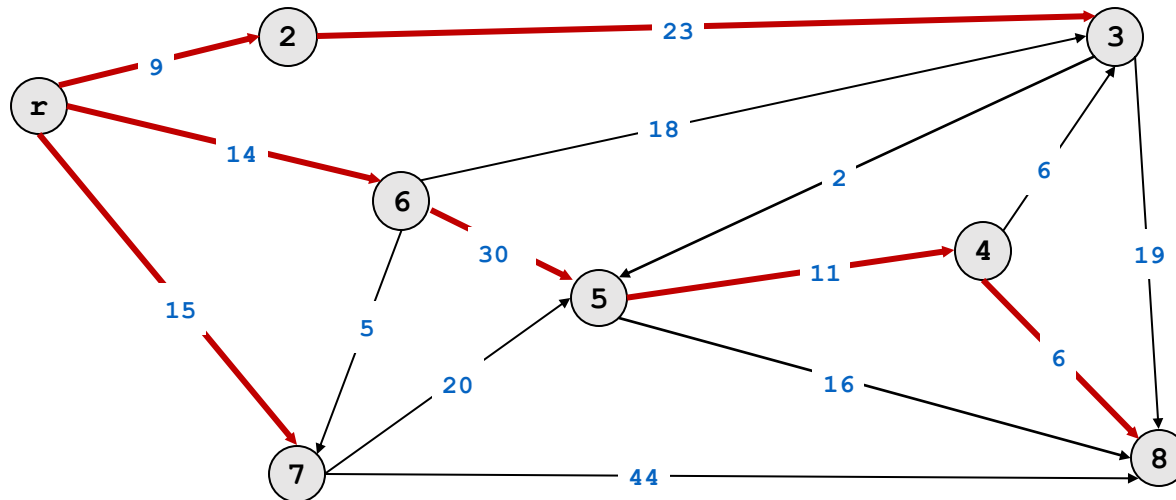


7. Min-Cost Arborescences



Arborescences

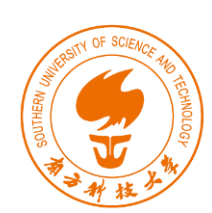
- **Def.** Given a **directed** graph $G = (V, E)$ and a root $r \in V$, an **arborescence** (rooted at r) is a subgraph $T = (V, F)$ such that
 - T is a spanning tree of G if we ignore the direction of edges.
 - There is a (unique) directed path in T from r to each other node $v \in V$.
- **Observation.** Arborescences are essentially directed spanning trees.





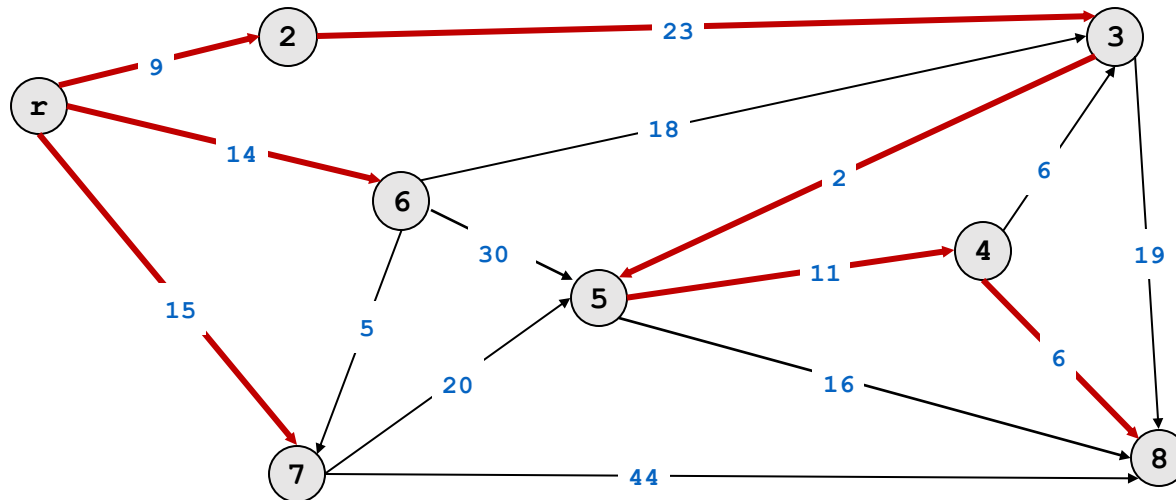
Arborescences

- **Claim.** A subgraph T 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. We need prove both the "if" part and the "only if" part.
 - "only if" part \Rightarrow :
 - ✓ A spanning tree has no cycles.
 - ✓ The last edge on the **unique** $r-v$ path is the only entering edge.
 - "if" part \Leftarrow :
 - ✓ Suppose T has no cycles and each node $v \neq r$ has exactly one entering edge.
 - ✓ To construct an $r-v$ path, start at v and 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. ■



Min-Cost Arborescences

- **Goal.** Given a directed graph G with a root node r and nonnegative edge costs, find an arborescence rooted at r of **minimum cost**.
- **Observation.** Min-cost arborescences are essentially **directed MSTs**.
- **Assumptions** (w.l.o.g.): All nodes are reachable from r . No edge enters r .



min-cost arborescence cost:
 $9 + 14 + 15 + 23 + 2 + 11 + 6 = 80$



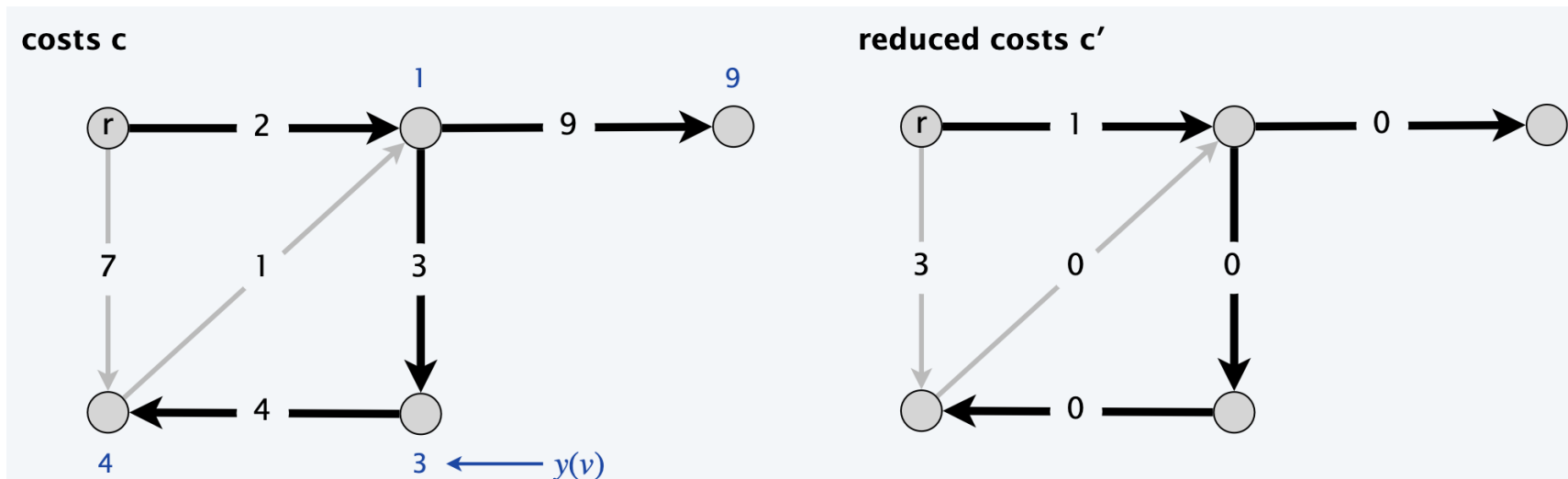
Min-Cost Arborescences

- **Goal.** Given a directed graph G with a root node r and nonnegative edge costs, find an arborescence rooted at r of **minimum cost**.
- **Observation.** Min-cost arborescences are essentially **directed MSTs**.
- **Assumptions** (w.l.o.g.): All nodes are reachable from r . No edge enters r .
- **Property.** For each node $v \neq r$, choose a cheapest edge entering v . If such $n - 1$ edges form an arborescence, then it is a min-cost arborescence.
 - What would happen when it is not an arborescence? There are directed cycles.
- **Q.** How can we remove such directed cycles?



Min-Cost Arborescences

- **Def.** For each $v \neq r$, let $y(v)$ denote the min cost of any edge entering v . The **reduced cost** of an edge (u, v) is $c'(u, v) = c(u, v) - y(v) \geq 0$.
- **Claim.** 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. Recall that any arborescence T has exactly one edge entering $v \neq r$, so the cost difference in c and c' is independent of the chosen T . ▀

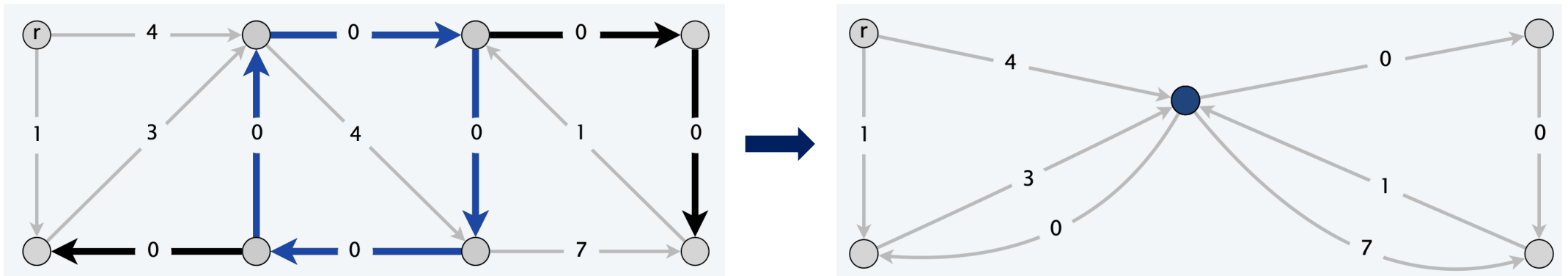




Chu-Liu's Algorithm (aka. Edmonds' Algorithm)

- **Greedy approach.**

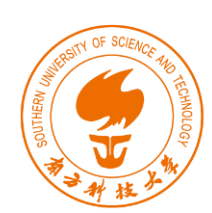
- For each $v \neq r$, choose a **cheapest edge entering v** and form an edge set E^* .
- Then, all edges in E^* have **0 cost** with respect to reduced costs $c'(u, v)$.
- If E^* does not contain a cycle, then we find a min-cost arborescence.
- If E^* contains a cycle C , can afford to **use as many such 0-cost edges in C** .
- Therefore, we can **contract C to a supernode** (and remove any self-loops).
- Recursively solve problem in contracted graph G' with costs $c'(u, v)$.





Chu-Liu's Algorithm: Analysis

- **Theorem.** [Chu–Liu 1965, Edmonds 1967] The greedy algorithm finds a min-cost arborescence.
- Pf. (by strong induction on number of nodes $|V|$)
 - If the edges of E^* 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).
 - There exists a min-cost arborescence T in G that corresponds to T' . ■
- **Q.** How do we prove the last step?
 - Min-cost arborescence in G' has 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 .



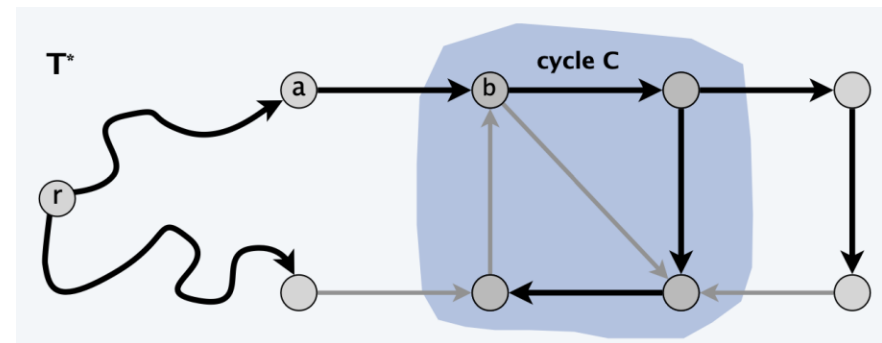
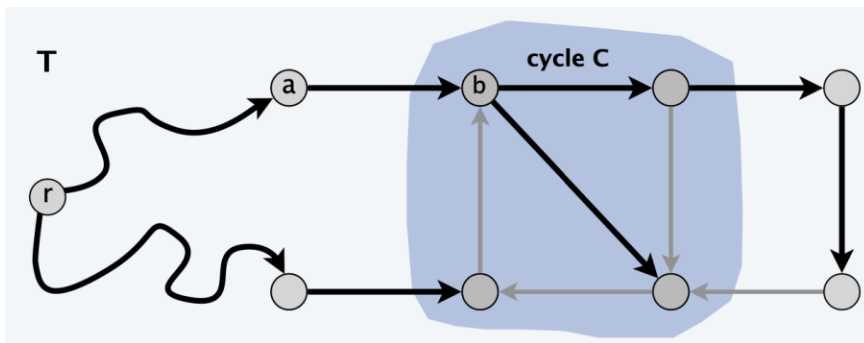
Chu-Liu's Algorithm: Analysis

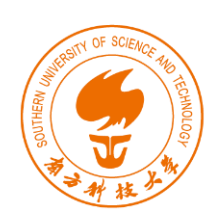
- **Claim.** Let C be a cycle in G containing only 0-cost edges. There exists a min-cost arborescence T rooted at r with exactly one edge entering C .
- Pf. (by cases)
 - Case 0. T has no edges entering C . Impossible!
 - ✓ Arborescence T has an $r-v$ path for each node $v \Rightarrow$ at least one edge enters C .
 - Case 1. T has exactly one edge entering C . Nothing to prove.
 - Case 2. T has **two (or more) edges** entering C . We construct another min-cost arborescence T^* that has exactly one edge entering C .



Chu-Liu's Algorithm: Analysis

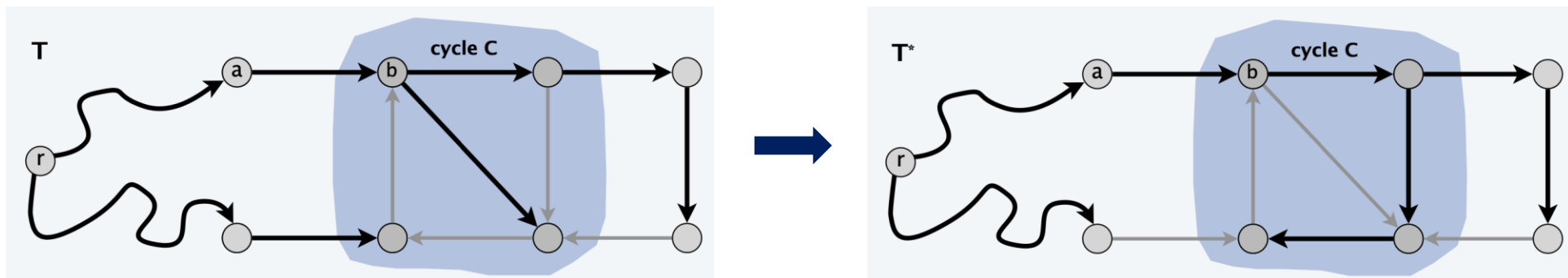
- **Claim.** Let C be a cycle in G containing only 0-cost edges. There exists a min-cost arborescence T rooted at r with exactly one edge entering C .
- Pf. (by cases)
 - Case 2. T has **two (or more) edges** entering C . We construct another min-cost arborescence T^* that has exactly one edge entering C .
 - ✓ 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) . ↖ only one node in C
 - ✓ We add in all edges of C except the one that enters b .





Chu-Liu's Algorithm: Analysis

- **Claim.** Let C be a cycle in G containing only 0-cost edges. There exists a min-cost arborescence T rooted at r with exactly one edge entering C .
- Pf. (by cases)
 - Case 2. T^* is a min-cost arborescence.
 - ✓ The cost of T^* is no more than that of T since we add only 0-cost edges.
 - ✓ T^* is an arborescence, i.e., it has exactly one edge entering each node $v \neq r$ and has no directed cycles.
 - T had no cycles before; no cycles within C ; now only (a, b) enters C .





Chu-Liu's Algorithm: Analysis

- **Theorem.** [Chu–Liu 1965, Edmonds 1967] The greedy algorithm finds a min-cost arborescence.
- Pf. (by strong induction on number of nodes $|V|$)
 - If the edges of E^* 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).
 - **Proved:** There exists a min-cost arborescence T in G that corresponds to T' . ■
- **Time complexity.** $O(mn)$.
 - 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. ■



More on Min-Cost Arborescences

- Chu-Liu's algorithm can be extended to find **directed minimum spanning forests**.
- There exists an $O(m + n \log n)$ time algorithm to compute a min-cost arborescence. [\[Gabow–Galil–Spencer–Tarjan 1985\]](#)