Algorithm Design and Analysis

CSE 565

Lecture 7

Greedy Graph Algorithms

- Shortest paths
- Minimum Spanning Tree

Sofya Raskhodnikova

The (Algorithm) Design Process

1. Work out the answer for some examples
2. Look for a general principle
   - Does it work on *all* your examples?
3. Write pseudocode
4. Test your algorithm by hand or computer
   - Does it work on *all* your examples?
   - Python is a great language for testing algorithms
5. Prove your algorithm is always correct
6. Check running time

Be prepared to go back to step 1!

Writing algorithms

- Clear and unambiguous
  - Test: You should be able to hand it to any student in the class, and have them convert it into working code.
- Homework pitfalls:
  - remember to specify data structures (list, stack, hash table,…)
  - For each function invocation, specify clearly what variables are passed to the function and what the function is returning.
  - writing recursive algorithms: don't confuse the recursive subroutine with the first call
  - label global variables clearly

Writing proofs

- State upfront the claim you are proving.
- Purpose
  - **Determine for yourself** that algorithm is correct
  - Convince reader
- Who is your audience?
  - **Yourself**
  - Your classmates
  - Not the TA/grader
- **Main goal: Find your own mistakes**

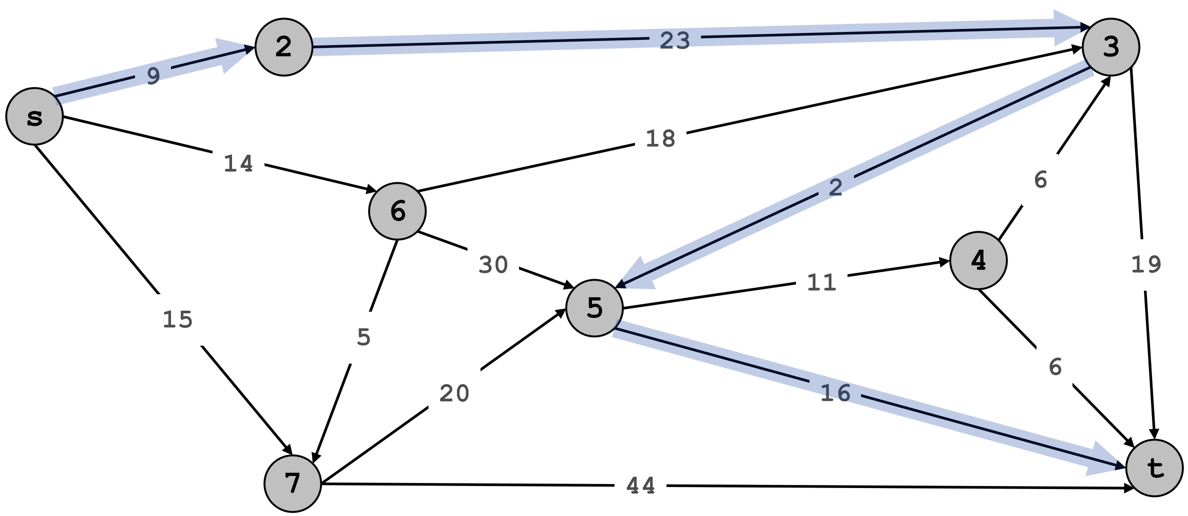
Homework

- Goals:
  - Reinforce and clarify material from lecture
  - Develop your skills
    - Problem-solving
    - Communication
- Make sure you understand the solution
- Use the feedback
- If you don't understand something, ask!
  - Me or the TA or on Piazza
- **Do not copy from other sources**

Shortest Paths

Shortest Path Problem

- **Input:**
  - Directed graph G = (V, E).
  - Source node s, destination node t.
  - for each edge e, length $\ell(e)$ = length of e.
  - length of a path = sum of lengths of edges on the path
- **Find:** shortest directed path from s to t.



Length of path (s,2,3,5,t) is 9 + 23 + 2 + 16 = 50.

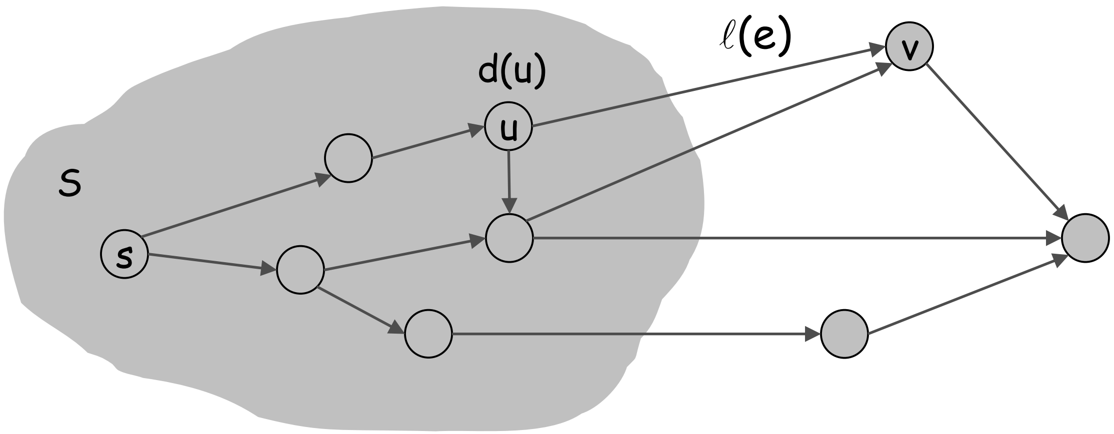
Dijksta's Algorithm: Overview

- Maintain a set of **explored nodes** S whose shortest path distance d(u) from s to u is known.
- Initialize S = { s }, d(s) = 0.
- Repeatedly choose unexplored node v which minimizes

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

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

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



Dijksta's Algorithm: Overview

- Maintain a set of **explored nodes** S whose shortest path distance d(u) from s to u is known.

Invariant: d(u) is known for all vertices in S

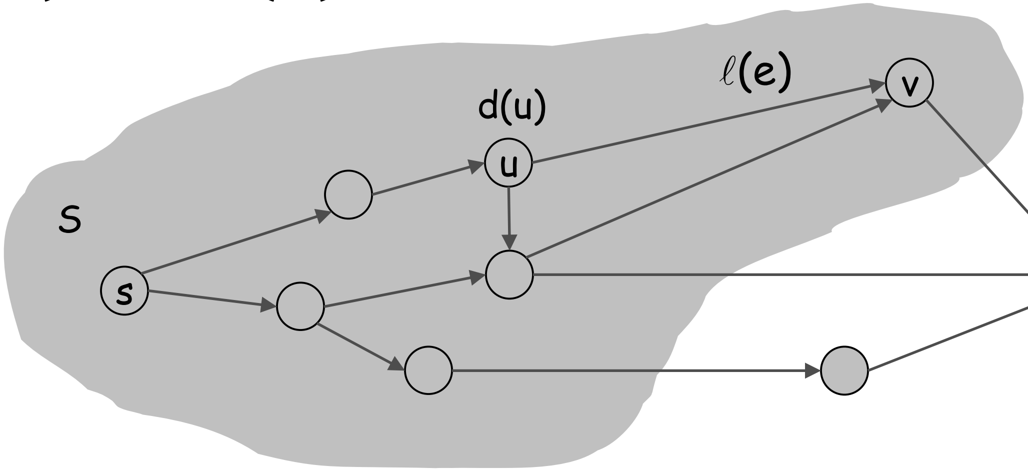
- Initialize S = { s }, d(s) = 0.
- Repeatedly choose unexplored node v which minimizes

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

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

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

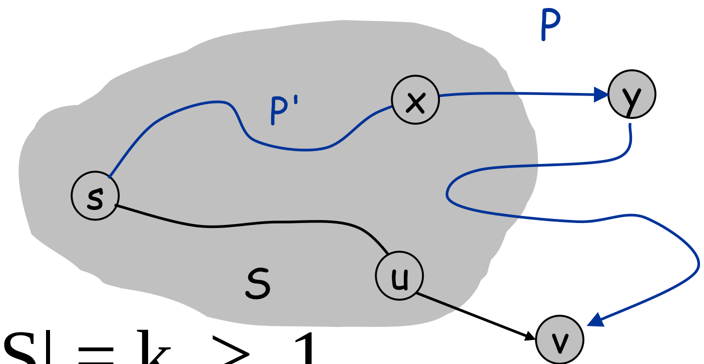
Intuition: like BFS, but with weighted edges



Correctness Proof of Dijkstra's

(Greedy Stays Ahead)

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



Proof: (by induction on $|S|$)

- **Base case:** $|S| = 1$; $d(s)=0$.
- **Inductive hypothesis:** Assume for $|S| = k \geq 1$.
  - 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. We'll see that it's 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' + (x,y) has length $\underbrace{\geq}_{\text{inductive hypothesis}} d(x)+ \ell(x,y) \underbrace{\geq}_{\text{defn of }\pi(y)} \pi(y) \underbrace{\geq}_{\text{Dijkstra's chose v instead of y}} \pi(v)$

Implementation

- For unexplored nodes, maintain

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

  - Next node to explore = node with minimum π(v).
  - When exploring v, for each edge e = (v,w), update

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

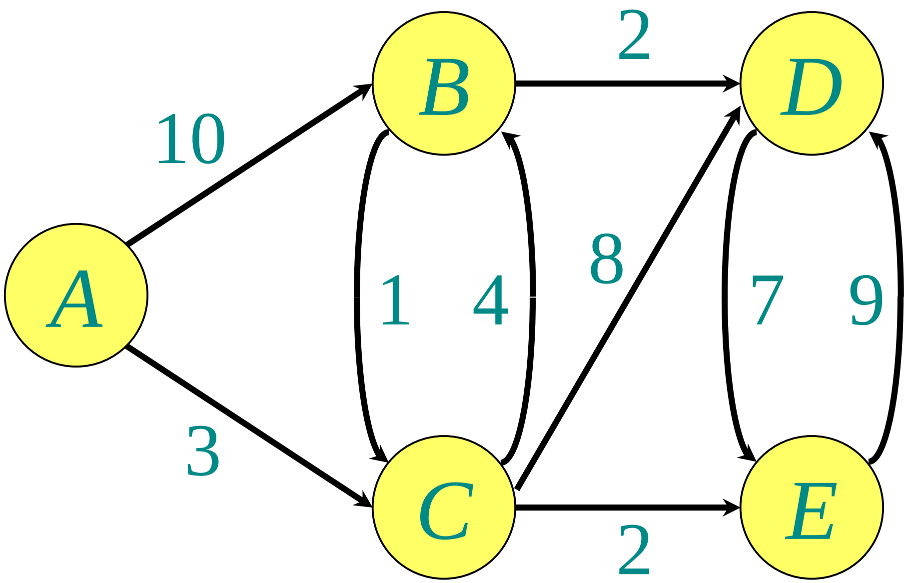
- **Efficient implementation:** Maintain a **priority queue** Q of unexplored nodes, prioritized by π(v).

Implementation: priority queues

- Maintain a set of items with priorities (= “keys”)
  - Example: jobs to be performed
- Operations:
  - INSERT
  - DECREASE-KEY
  - EXTRACT-MIN: find and remove item with least key
- Common data structure: heap
  - Time: O(log n) per operation

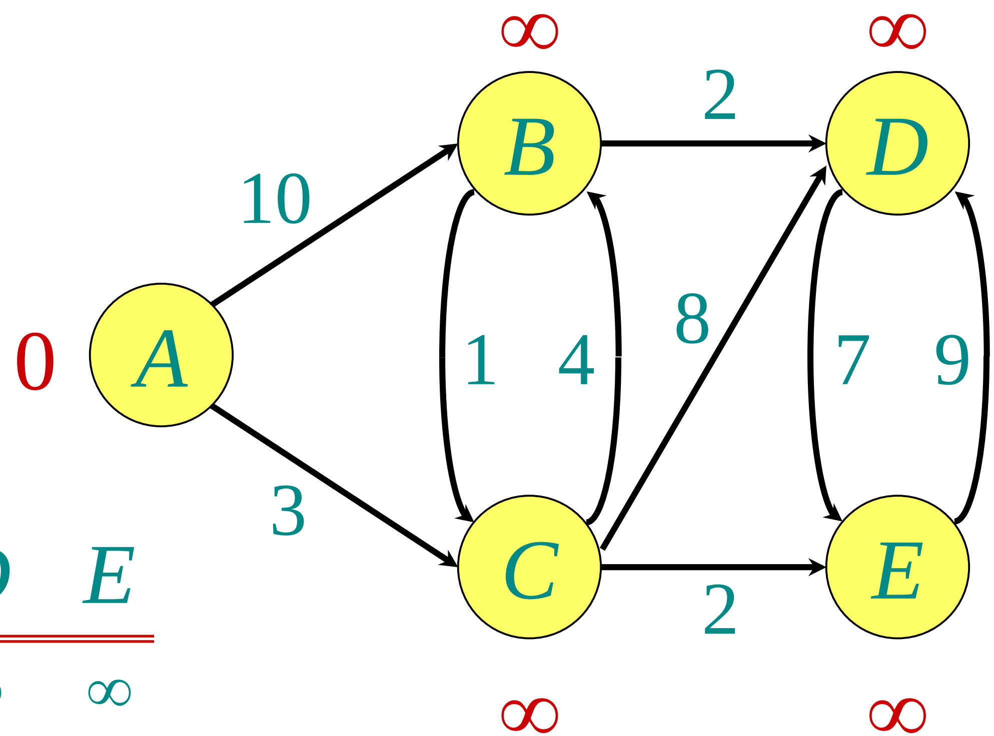
Demo of Dijkstra's Algorithm

Graph with nonnegative edge lengths:



Demo of Dijkstra's Algorithm

Initialize:

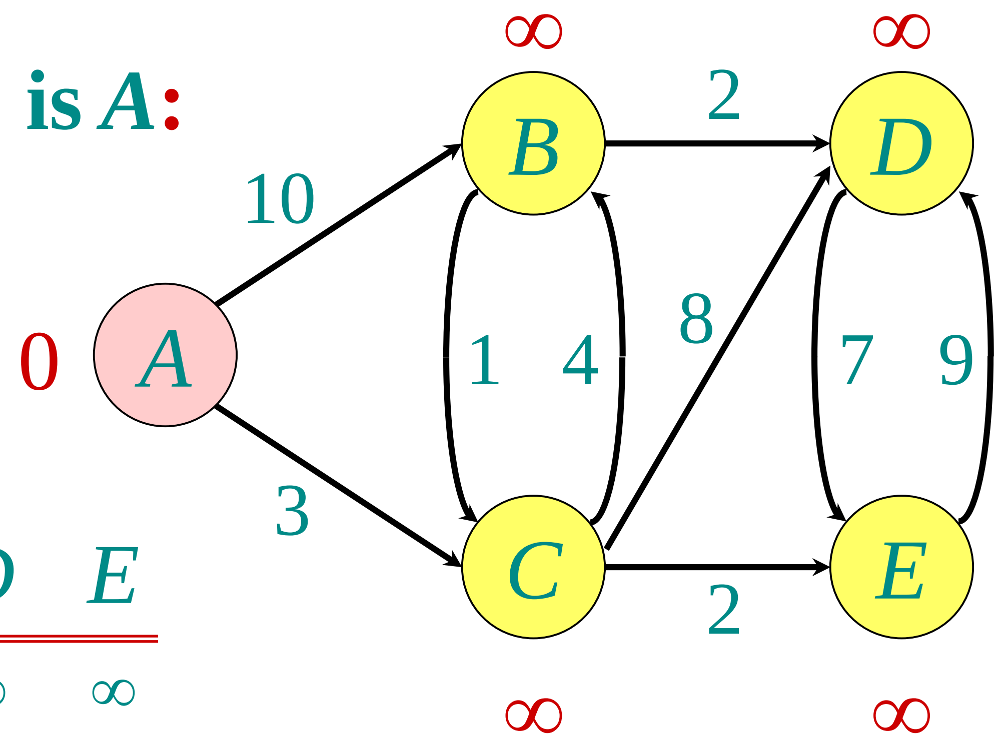


Q:	A	B	C	D	E
π(v):	0	∞	∞	∞	∞

S: { }

Demo of Dijkstra's Algorithm

EXTRACT-MIN(Q) is A:

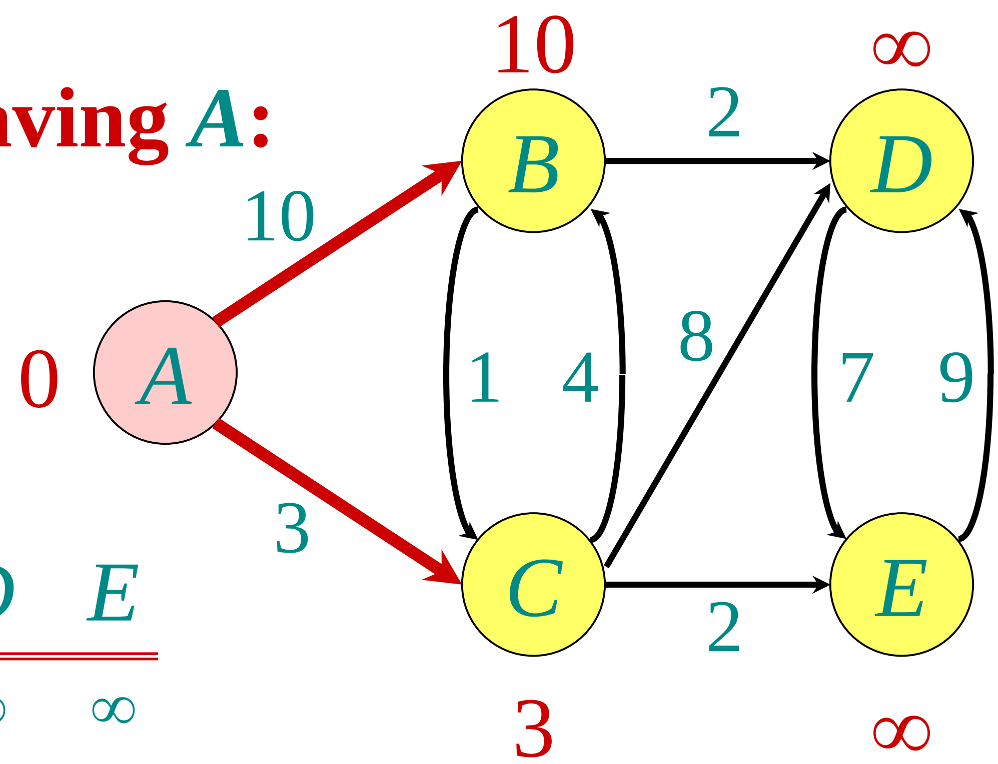


Q:	A	B	C	D	E
$\pi(v)$:	0	∞	∞	∞	∞

S: { A }

Demo of Dijkstra's Algorithm

Explore edges leaving *A*:

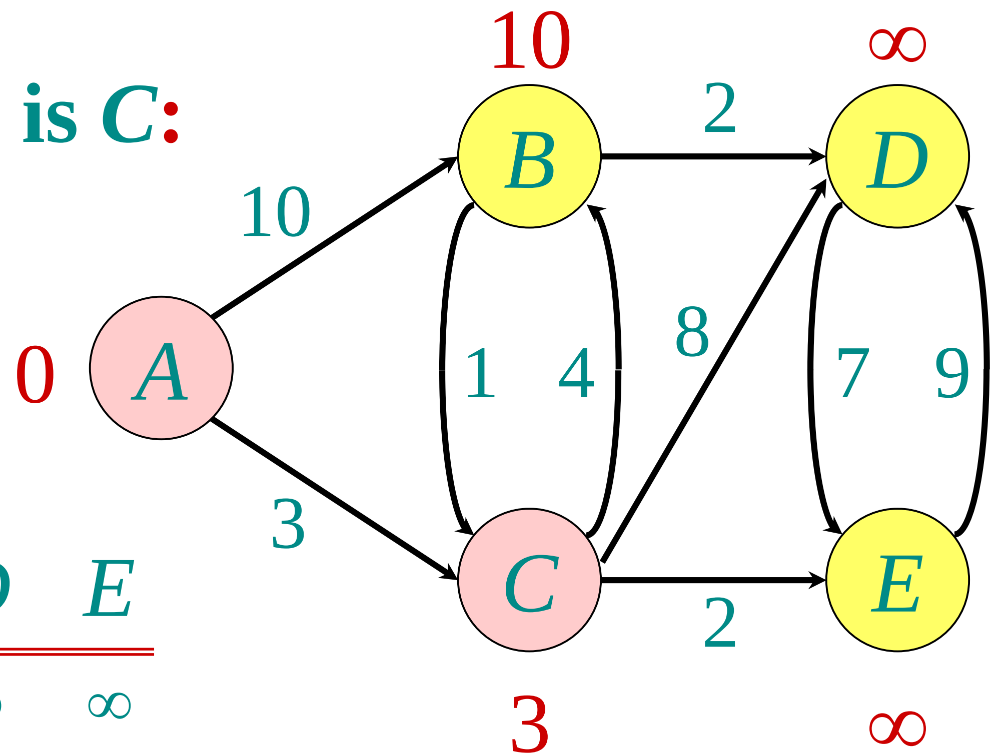


Q:	*A*	*B*	*C*	*D*	*E*
$\pi(v)$:	0	∞	∞	∞	∞
		10	3	∞	∞

S: { *A* }

Demo of Dijkstra's Algorithm

EXTRACT-MIN(Q) is C:

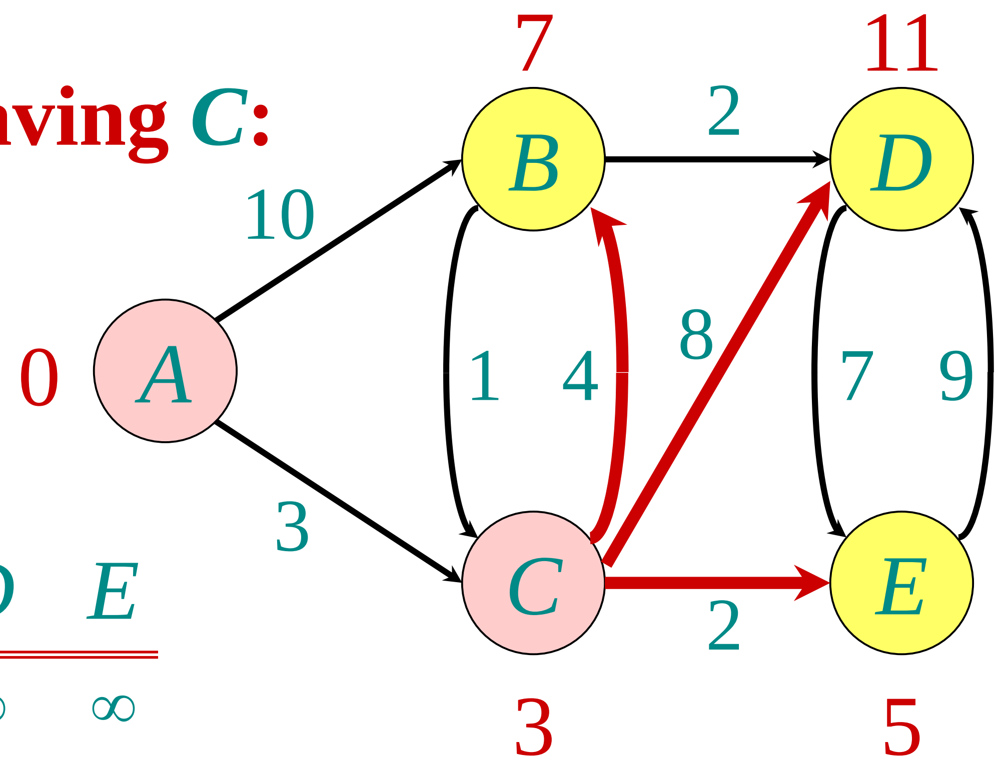


Q:	A	B	C	D	E
$\pi(v)$:	$\boxed{0}$	∞	∞	∞	∞
		10	$\boxed{3}$	∞	∞

S: { A, C }

Demo of Dijkstra's Algorithm

Explore edges leaving *C*:

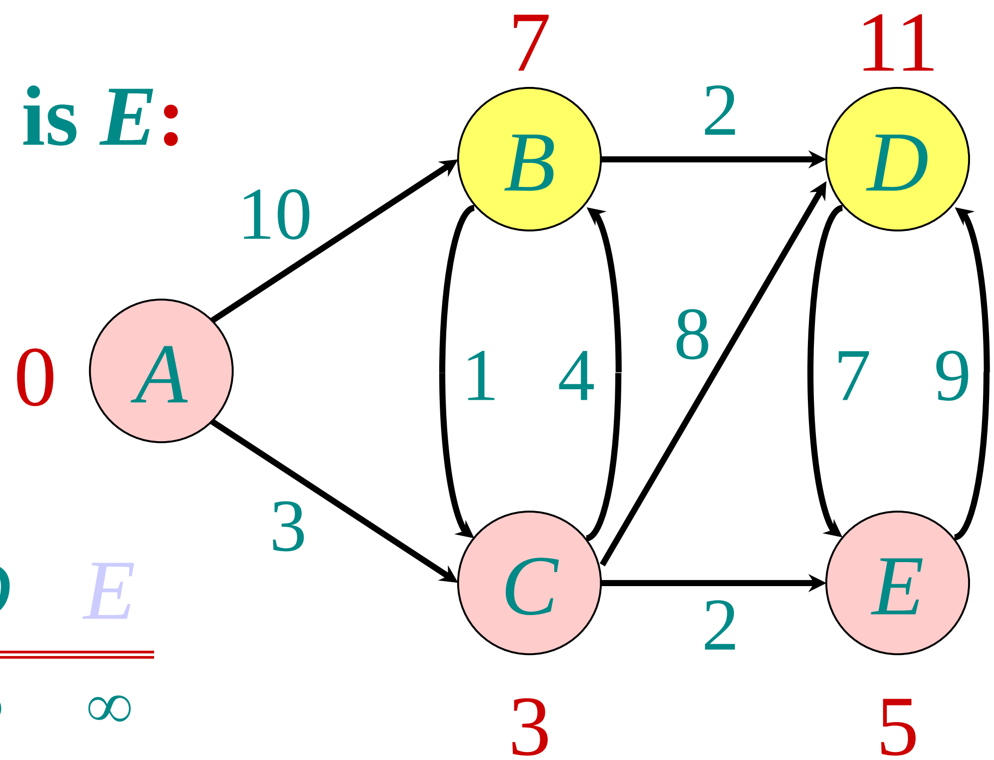


Q:	*A*	*B*	*C*	*D*	*E*
π(v):	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { *A*, *C* }

Demo of Dijkstra's Algorithm

EXTRACT-MIN(*Q*) is *E*:

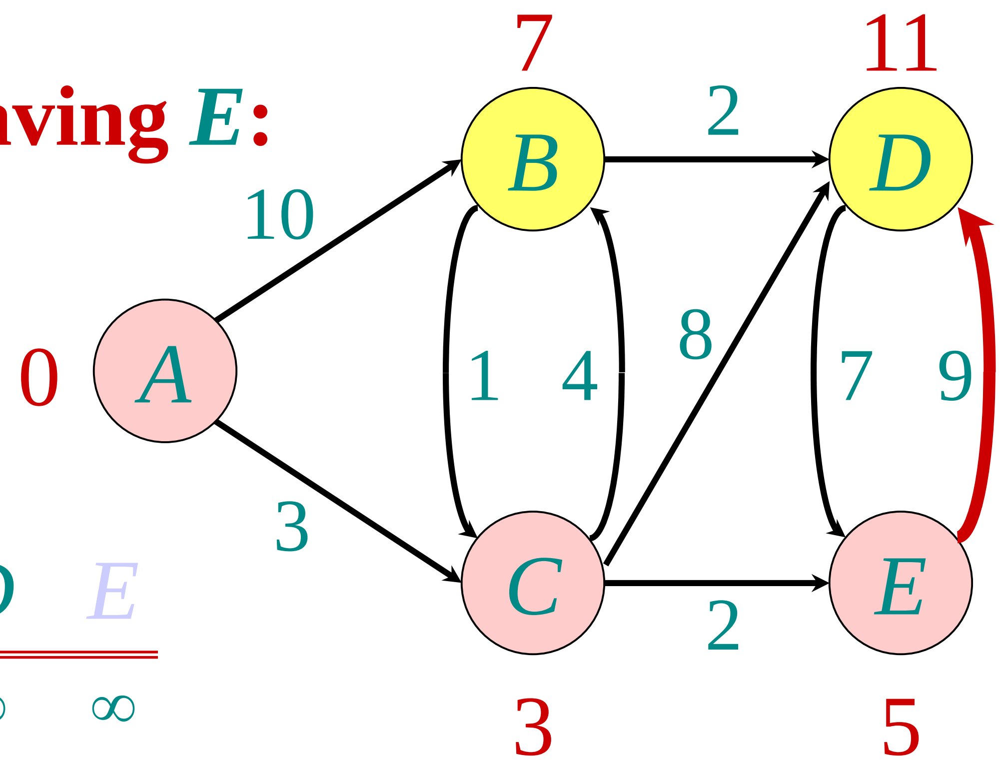


Q:	*A*	*B*	*C*	*D*	*E*
π(v):	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { *A*, *C*, *E* }

Demo of Dijkstra's Algorithm

Explore edges leaving *E*:

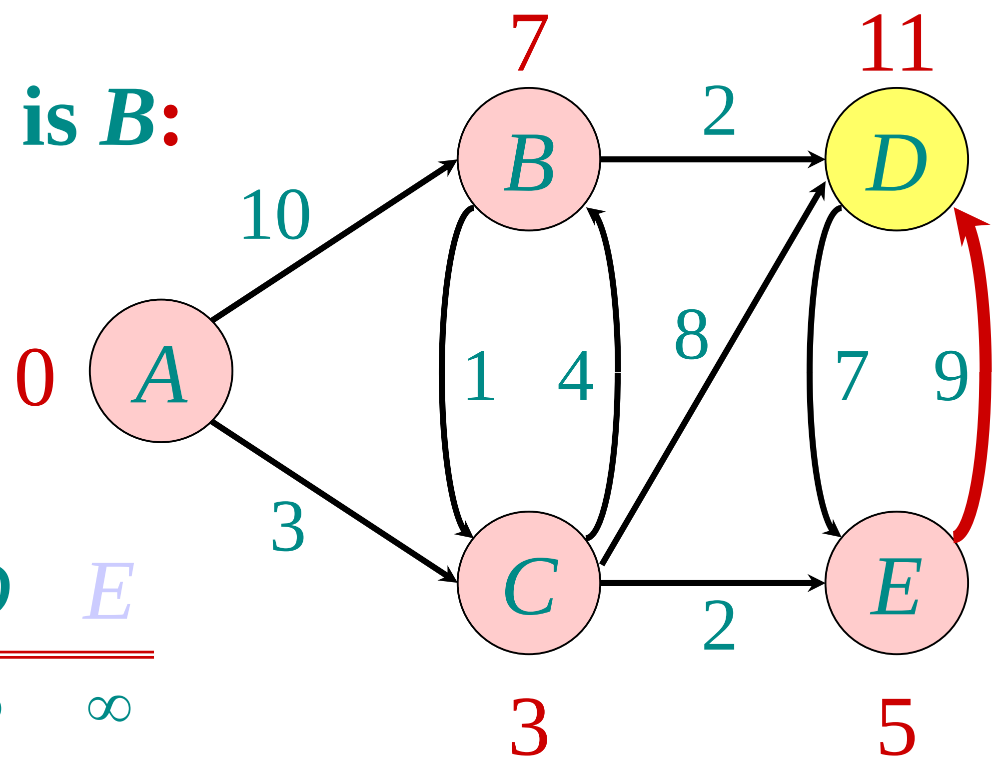


Q:	*A*	*B*	*C*	*D*	*E*
π(v):	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { *A*, *C*, *E* }

Demo of Dijkstra's Algorithm

EXTRACT-MIN(Q) is B:

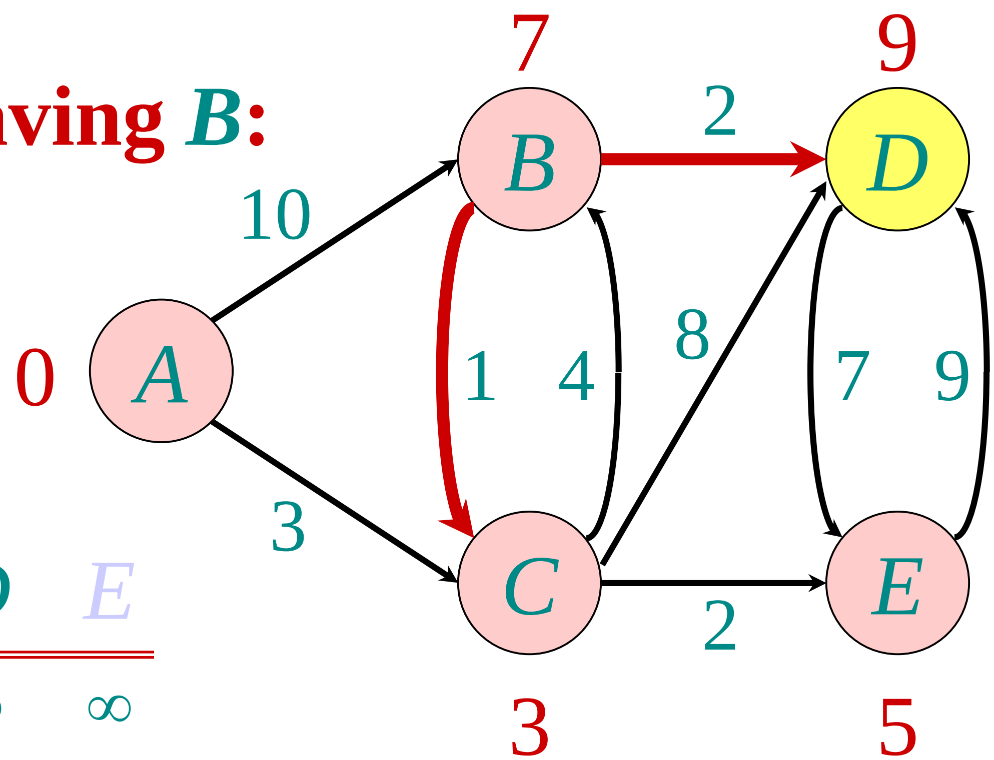


Q:	A	B	C	D	E
$\pi(v)$:	$\boxed{0}$	∞	∞	∞	∞
		10	$\boxed{3}$	∞	∞
		7		11	$\boxed{5}$
		$\boxed{7}$		11	

S: { A, C, E, B }

Demo of Dijkstra's Algorithm

Explore edges leaving *B*:

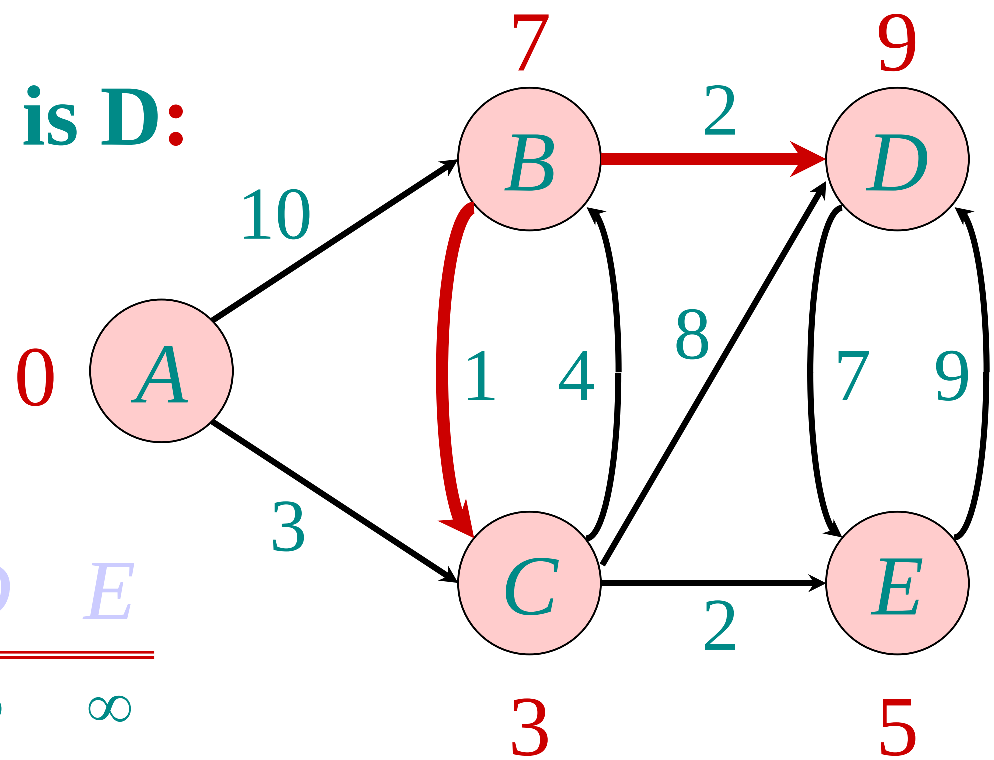


Q:	*A*	*B*	*C*	*D*	*E*
π(v):	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	

S: { *A*, *C*, *E*, *B* }

Demo of Dijkstra's Algorithm

EXTRACT-MIN(*Q*) is D:



Q:	*A*	*B*	*C*	*D*	*E*
π(v):	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	

S: { *A*, *C*, *E*, *B*, *D* }

Pseudocode for Dijkstra(G, ℓ)

$d[s] \leftarrow 0$
for each $v \in V - \{s\}$
  do $d[v] \leftarrow \infty$; $\pi[v] \leftarrow \infty$
$S \leftarrow \varnothing$
$Q \leftarrow V$ ⊳ Q is a priority queue maintaining $V - S$, keyed on $\pi[v]$
while $Q \neq \varnothing$
  do $u \leftarrow$ EXTRACT-MIN(Q)
    $S \leftarrow S \cup \{u\}$; $d[u] \leftarrow \pi[u]$
    for each $v \in Adjacency\text{-}list[u]$
      do if $\pi[v] > \pi[u] + \ell(u, v)$
        then $\pi[v] \leftarrow d[u] + \ell(u, v)$

explore edges leaving v

Implicit DECREASE-KEY

Analysis of Dijkstra

n times

while $Q \neq \emptyset$
    do $u \leftarrow$ EXTRACT-MIN(Q)
        $S \leftarrow S \cup \{u\}$
        for each $v \in Adj[u]$

explore an edge
            do if $d[v] > d[u] + \ell(u, v)$
                then $d[v] \leftarrow d[u] + \ell(u, v)$

\\ m implicit DECREASE-KEY's.

PQ Operation	Dijkstra	Array	Binary heap	d-way Heap	Fib heap †
ExtractMin	n	n	log n	HW	log n
DecreaseKey	m	1	log n	HW	1
Total		n^2	m log n	$m \log_{m/n} n$	m + n log n

† Individual ops are amortized bounds

Physical intuition

- System of pipes filling with water
  - Vertices are intersections
  - Edge length = pipe length
  - $d(v)$ = time at which water reaches v
- Balls and strings
  - Vertices $\mapsto$ balls
  - Edge e $\mapsto$ string of length $\ell(e)$
  - Hold ball s up in the air
  - $d(v) =$ (height of s)-(height of v)
- Nature uses greedy algorithms

Review

- Is Dijsktra's algorithm correct with negative edge weights?
  Give either
  - a proof of correctness, or
  - an example of a graph where Dijkstra fails

Further reading

- Erickson's lecture notes:
  http://web.engr.illinois.edu/~jeffe/teaching/algorithms/notes/21-sssp.pdf

Minimum Spanning Tree

Minimum spanning tree (MST)

Input: A connected undirected graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$.

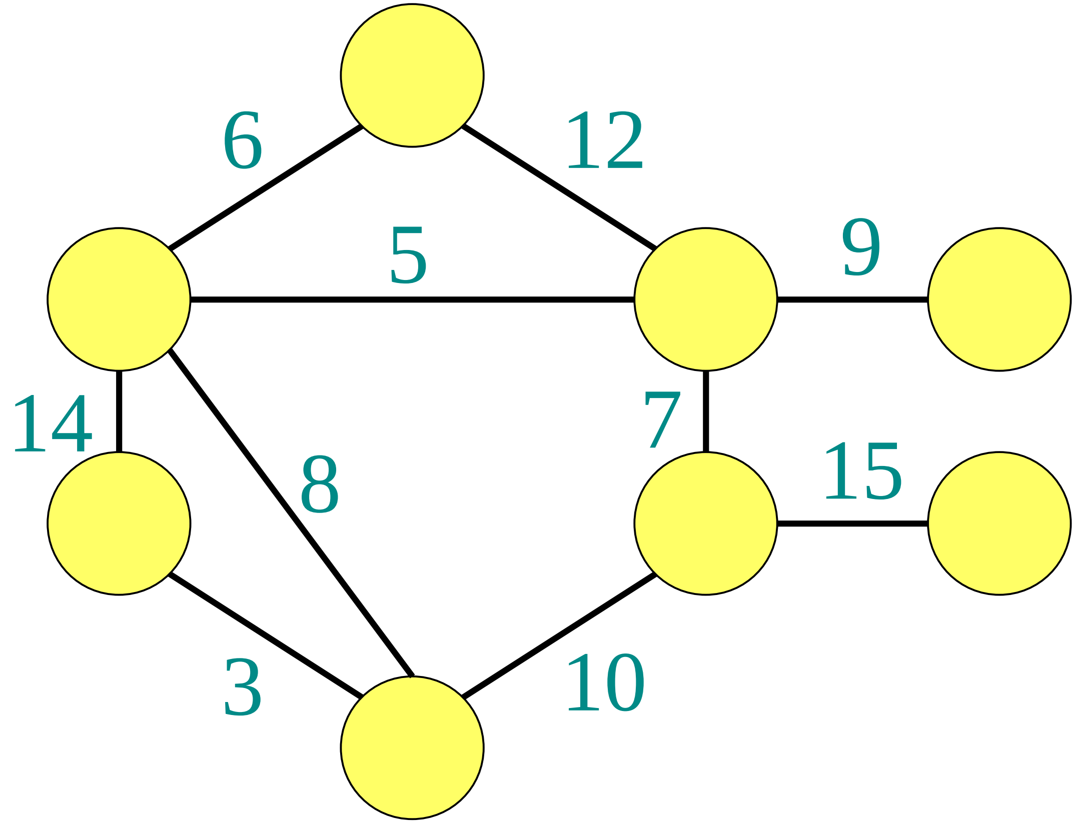
- For now, assume all edge weights are distinct.

Definition: A ***spanning tree*** is a tree that connects all vertices.

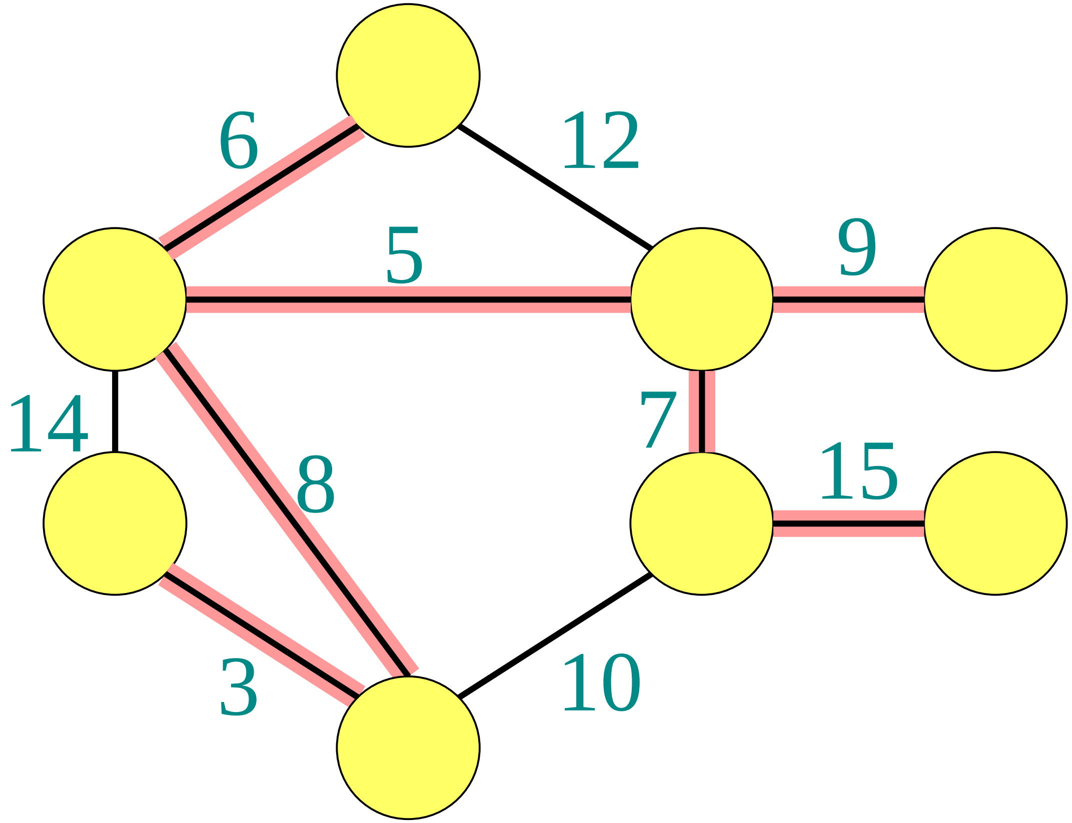
Output: A ***spanning tree*** T of minimum weight:

$$w(T) = \sum_{(u,v) \in T} w(u,v).$$

Example of MST



Example of MST

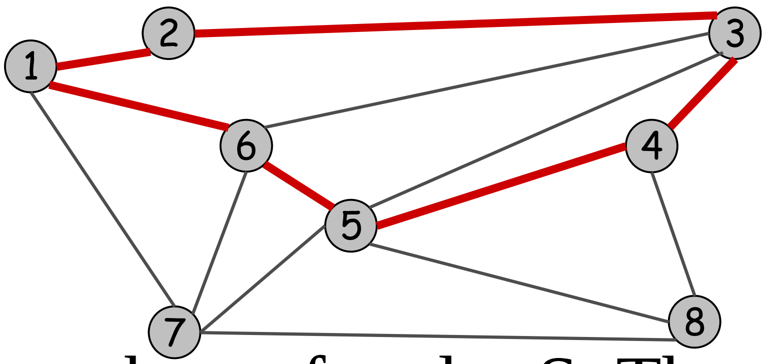


Greedy Algorithms for MST

- **Kruskal's:** Start with $T = \emptyset$. Consider edges in ascending order of weights. Insert edge e in T unless doing so would create a cycle.
- **Reverse-Delete:** Start with $T = E$. Consider edges in descending order of weights. Delete edge e from T unless doing so would disconnect T.
- **Prim's:** Start with some root node s. Grow a tree T from s outward. At each step, add to T the cheapest edge e with exactly one endpoint in 5.
- **Borůvka's:** Start with $T = \emptyset$. At each round, add the cheapest edge leaving each connected component of T.

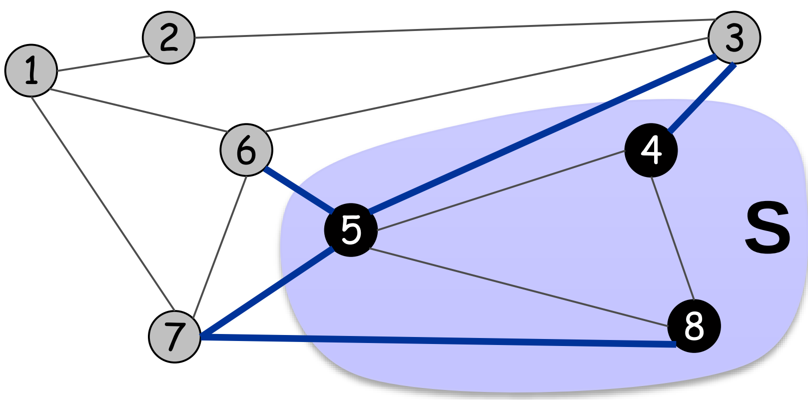
Cycles and Cuts

- **Cycle:** Set of edges of the form (a,b),(b,c),…,(y,z),(z,a).



Cycle C = (1,2),(2,3),(3,4),(4,5),(5,6),(6,1)

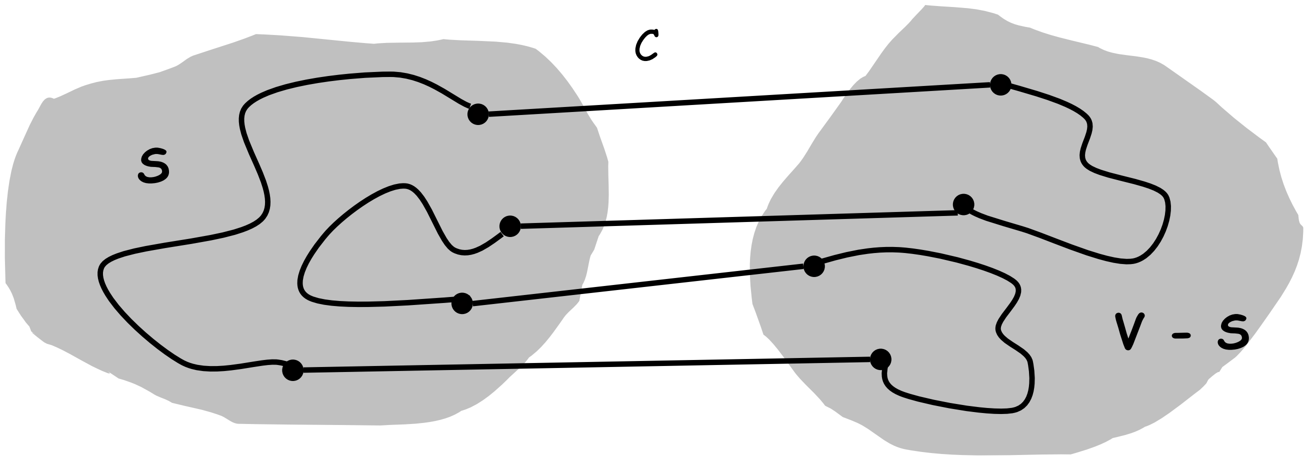
- **Cut:** a subset of nodes S. The corresponding **cutset** D is the subset of edges with exactly one endpoint in S.



Cut S = { 4, 5, 8 }
Cutset D = (5,6), (5,7), (3,4), (3,5), (7,8)

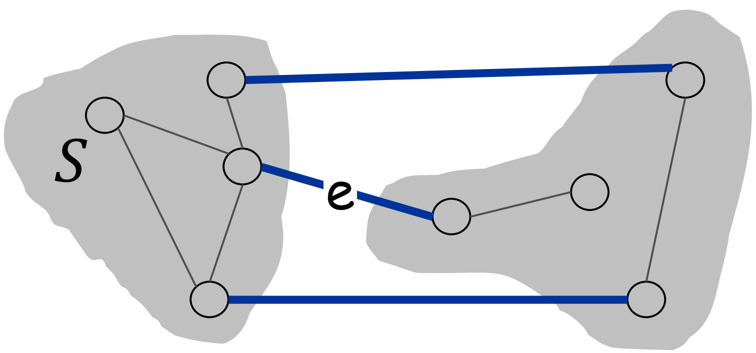
Cycle-Cut Intersection

- **Claim.** A cycle and a cutset intersect in an even number of edges.
- **Proof:** A cycle has to leave and enter the cut the same number of times.

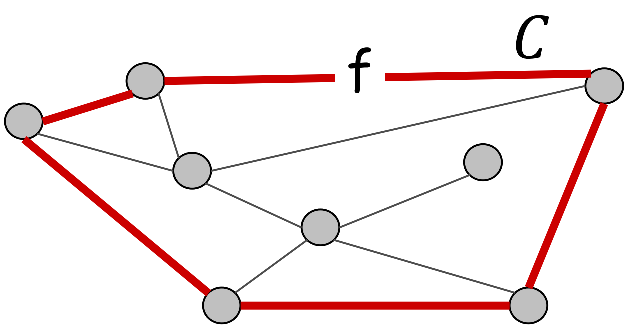


Cut and Cycle Properties

- **Cut property.** Let S be a subset of nodes. Let e be the min weight edge with exactly one endpoint in S. Then the MST contains e.
- **Cycle property.** Let C be a cycle, and let f be the max weight edge in C. Then the MST does not contain f.



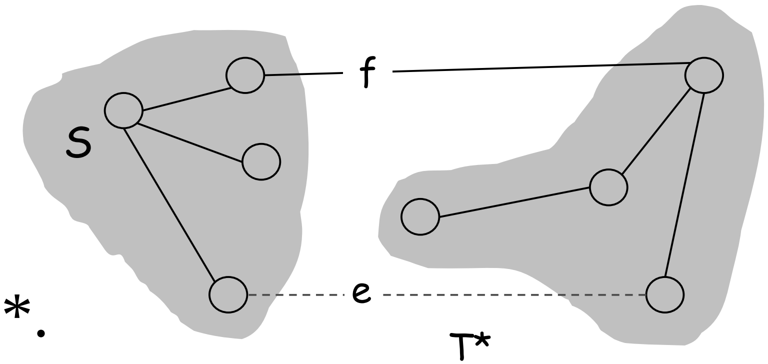
e is in the MST



f is not in the MST

Proof of Cut Property

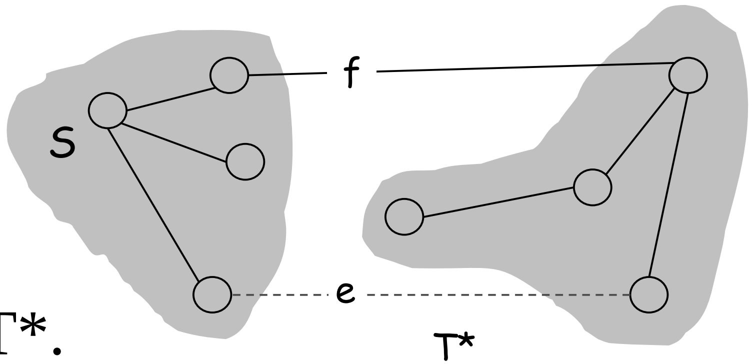
Cut property: Let S be a subset of nodes. Let e be the min weight edge with exactly one endpoint in S. Then the MST T* contains e.



- **Proof:** (exchange argument)
  - Suppose e does not belong to T*.
  - Adding e to T* creates a cycle C in T*.
  - Edge e is both in the cycle C and in the cutset D corresponding to S $\Rightarrow$ there exists another edge, say f, that is in both C and D.
  - T' = T* $\cup$ { e } - { f } is also a spanning tree.
  - Since $c_e < c_f$, cost(T') < cost(T*). Contradiction. ▪

Proof of Cycle Property

Cycle property: Let C be a cycle in G. Let f be the max weight edge in C. Then the MST T* does not contain f.



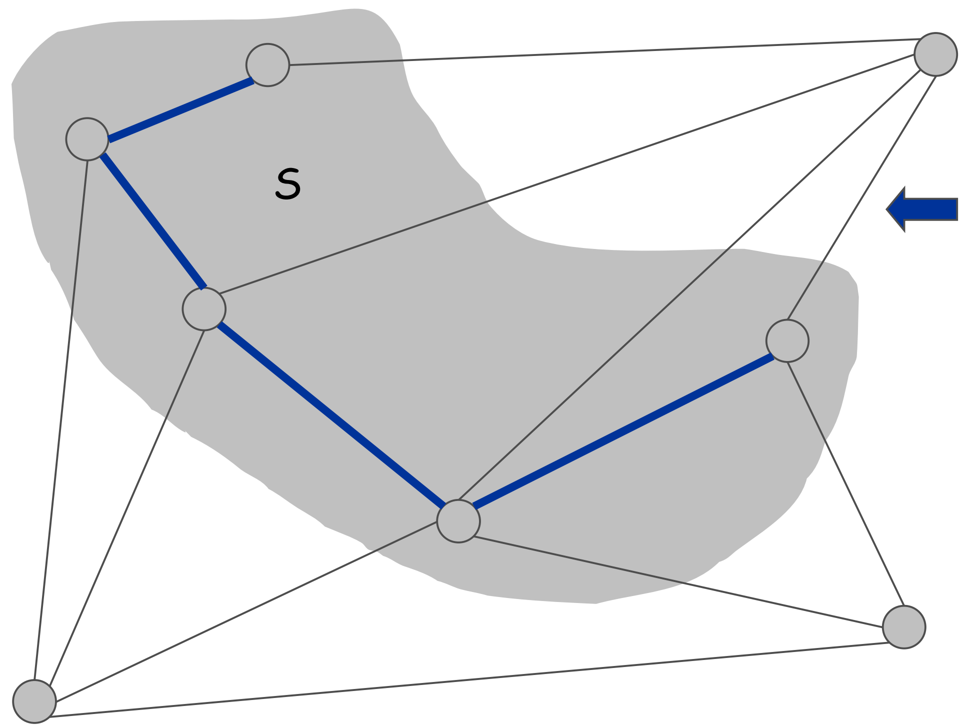
- **Proof:** (exchange argument)
  - Suppose f belongs to T*.
  - Deleting f from T* creates a cut S in T*.
  - Edge f is both in the cycle C and in the cutset D corresponding to S $\Rightarrow$ there exists another edge, say e, that is in both C and D.
  - T' = T* $\cup$ { e } - { f } is also a spanning tree.
  - Since $c_e < c_f$, cost(T') < cost(T*). Contradiction. ▪

Greedy Algorithms for MST

- **Kruskal's:** Start with $T = \emptyset$. Consider edges in ascending order of weights. Insert edge e in T unless doing so would create a cycle.
- **Reverse-Delete:** Start with $T = E$. Consider edges in descending order of weights. Delete edge e from T unless doing so would disconnect T.
- **Prim's:** Start with some root node s. Grow a tree T from s outward. At each step, add to T the cheapest edge e with exactly one endpoint in 5.
- **Borůvka's:** Start with $T = \emptyset$. At each round, add the cheapest edge leaving each connected component of T.

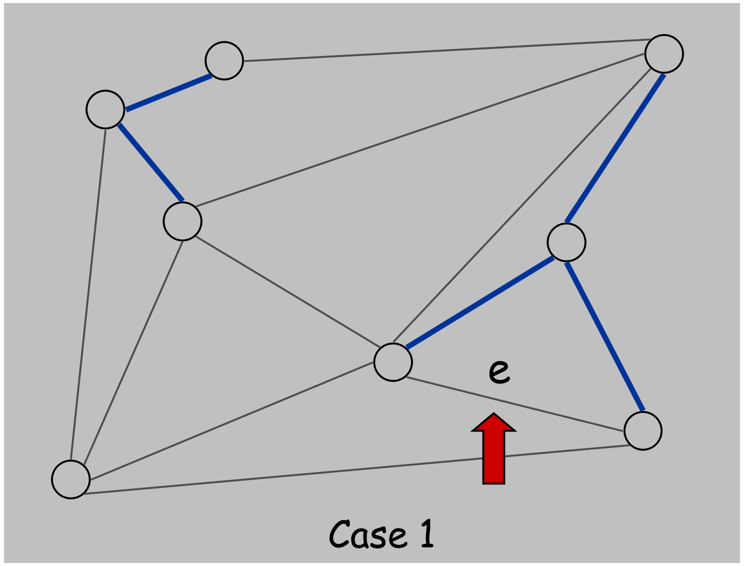
Prim's Algorithm: Correctness

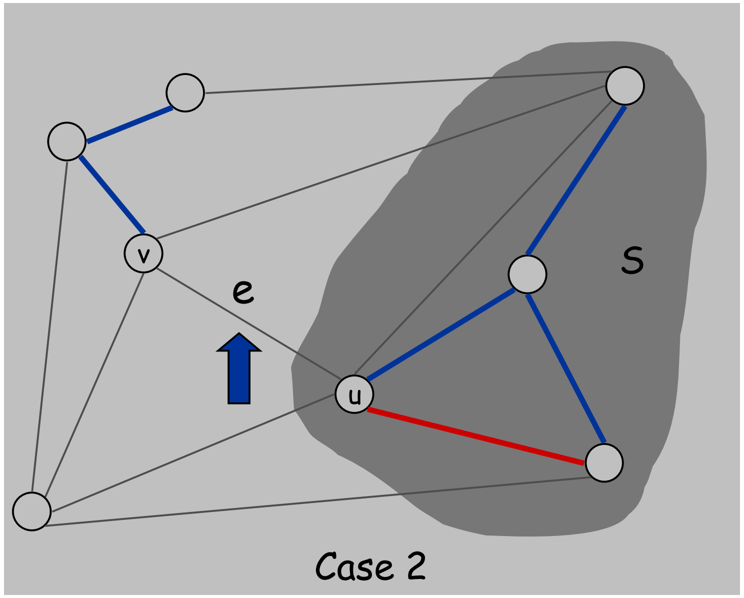
- Prim's algorithm. [Jarník 1930, Prim 1959]
  - Apply cut property to S.
  - When edge weights are distinct, every edge that is added must be in the MST
  - Thus, Prim’s algorithm outputs the MST



Correctness of Kruskal

- [Kruskal, 1956]: Consider edges in ascending order of weight.
  - **Case 1:** If adding e to T creates a cycle, discard e according to cycle property.





  - **Case 2:** Otherwise, insert e = (u, v) into T according to cut property where S = set of nodes in u's connected component.