CS4800: Algorithms & Data
Jonathan Ullman

Lecture 13:

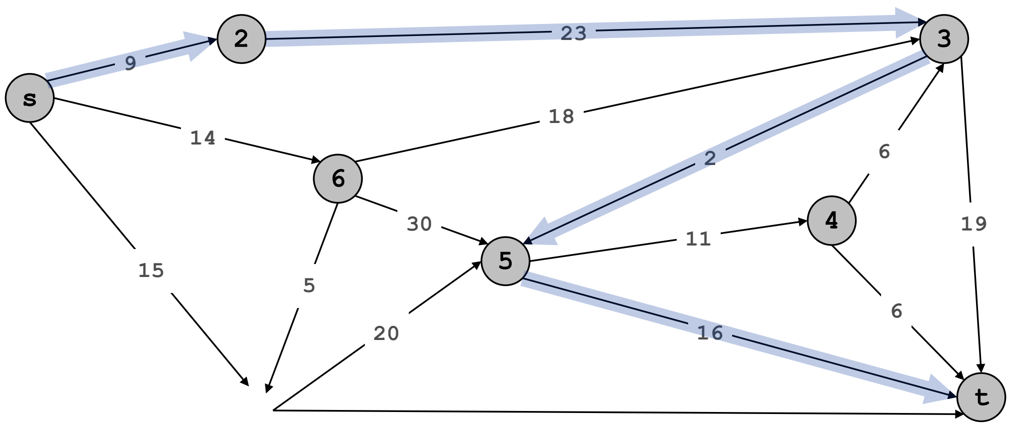
- Shortest Paths: Dijkstra's Algorithm, Heaps
- DFS(?)

Feb ~~20~~, 2018

22

Navigation

- Google Maps
- Internet Routing
- Slime



- Solving shortest paths in practice

Weighted Graphs

- A graph with edge lengths $G = (V, E, \{w_e\})$
  - V is the set of nodes/vertices
  - $E \subseteq V \times V$ is the set of edges
  - $w_e \in \mathbb{R}$ are edge weights
  - Can be directed or undirected

- Today:
  - ✓ Directed graphs (models one-way streets)
  - ✓ Non-negative edge lengths denoted $\ell_e \geq 0$

Shortest Paths

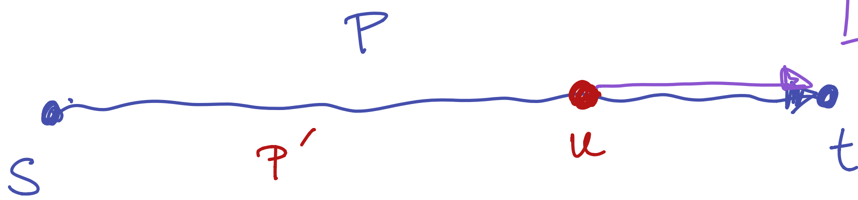
- The length of a path $P = v_1 \overset{6}{-} v_2 \overset{7}{-} \cdots \overset{11}{-} v_k$ is the sum of the edge lengths

$$\ell(P) = \sum_{i=2}^{k} \ell_{v_{i-1}, v_i}$$

- Shortest Path: given nodes $s, t \in V$, find the shortest path from s to t $P: s - v_1 - v_2 - \cdots - v_k - t$
- Single-Source Shortest Paths: given a node $s \in V$, find the shortest paths from s to every $t \in V$

Structure of Shortest Paths

d(v): distance from s to v



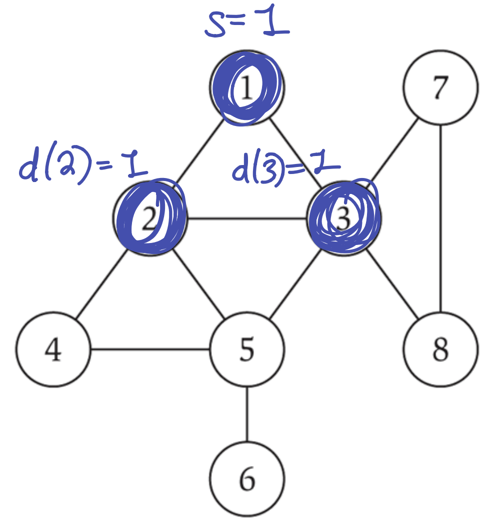
$d(s,t)$ = length of the shortest path from s to t

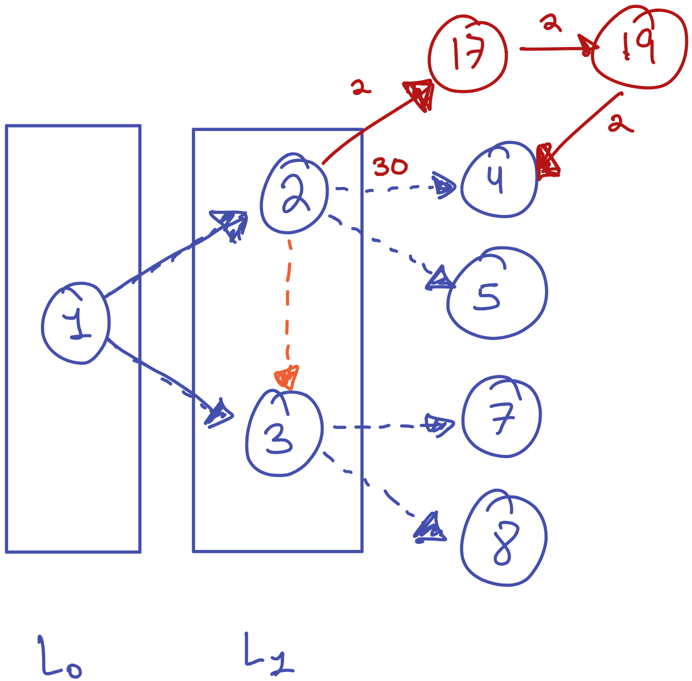
Observation: If the shortest path P from s to t passes through u, then P' must be a shortest path from s to u.

Observation: Suppose we know $d(s,u)$ and there is an edge $(u,t) \in E$. Then $d(s,t) \le d(s,u) + l_{u,t}$.

Last Week on *CS4800*

- If all edge lengths are 1 then BFS solves SSSP





$d(s) = 0$

$d(2) = d(3) = 1$

$d(4), d(5), d(7), d(8) \leq 2$

Last Week on *CS4800*

$\forall v$ s.t. $d(v) = i+1$

$\Rightarrow d(u) = i \Rightarrow L_i$

- Theorem: if all edge lengths are 1, then layer i is exactly the set of nodes at distance i from s

Proof: (Induction)

✓ Base Case ($i=0$): $L_0 = \{s\}$ which is the unique node reachable in 0 hops.

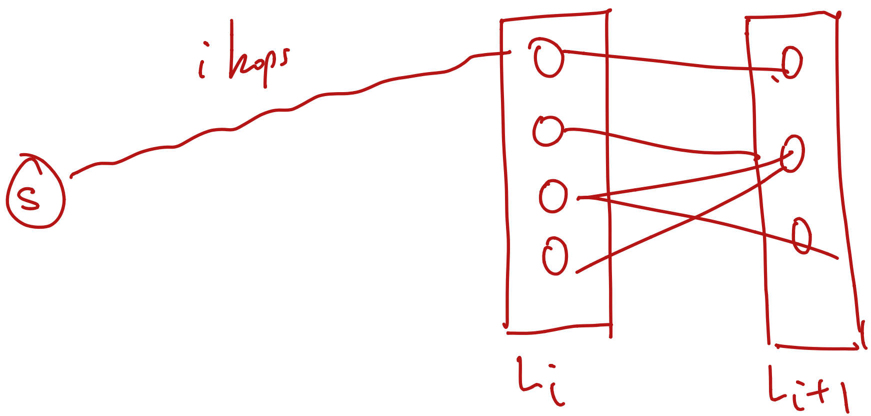
$\forall v \in L_{i+1}\ d(v) \leq i+1$

$(s \overset{i}{\rightsquigarrow} u \overset{1}{\rightarrow} v)$

$\forall v \in L_{i+1}\ d(v) \geq i+1$

$\Downarrow$

$v \in L_{i+1}\ d(v) = i+1$



Extending to Edge Lengths

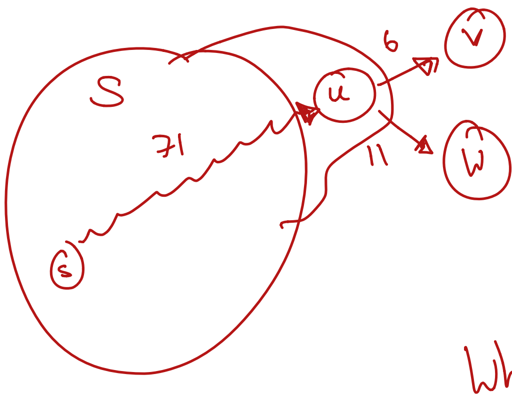
- Alternative interpretation of BFS:
  - Maintain a set of explored nodes S
  - Invariant: when we explore a node, we know its distance
  - Iteration: explore the nodes at smallest distance first

- Can we generalize this to graphs with edge lengths?

Dijkstra's Algorithm (Informal)

- Maintain a set of explored nodes S
  - Initially S is empty
- Maintain upper bound on distances $d(v)$
  - Initially $d(s) = 0, \forall v \neq s,\ d(v) = \infty$
- Until all nodes are explored:
  - Choose the node $u \notin S$ that minimizes $d(u)$
  - Update the distance of all neighbors of u

- Invariant: if $u \in S$, then $d(u)$ is the length of the shortest path from s to u
  - We'll talk about finding the shortest paths in a bit



$d(v) \le d(u) + 6 \le 77$

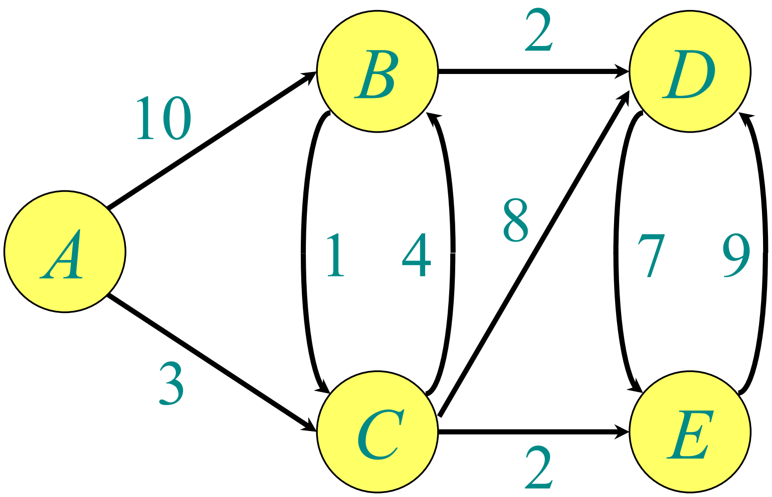
$d(w) \le d(u) + 11 \le 82$

$d(u) \le 71$

When we explore a node u we're going to update our upper bound on the distance from $s \rightsquigarrow v$ $\forall$ neighbors v of u.

Dijkstra's Algorithm: Demo

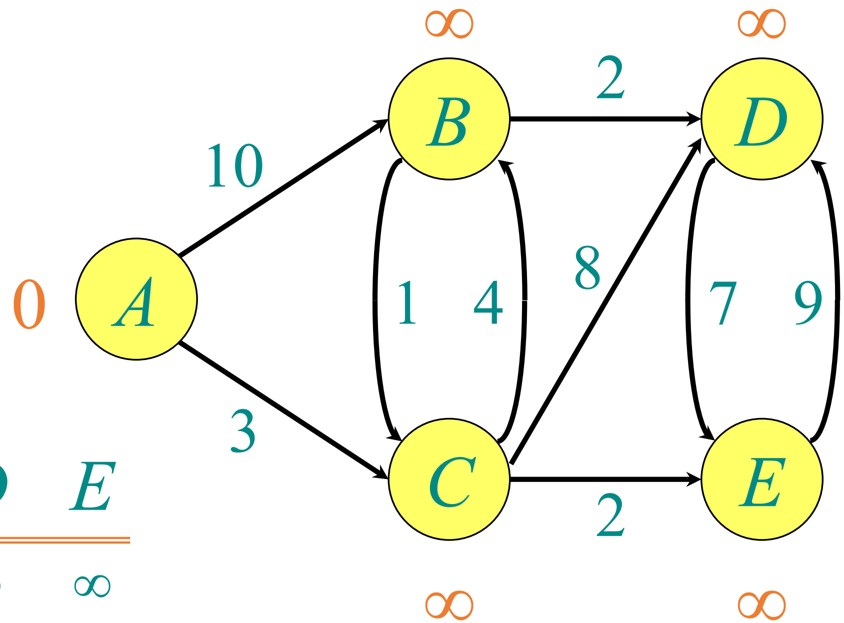
Graph with nonnegative edge lengths:



Dijkstra's Algorithm: Demo

s = A

Initialize:

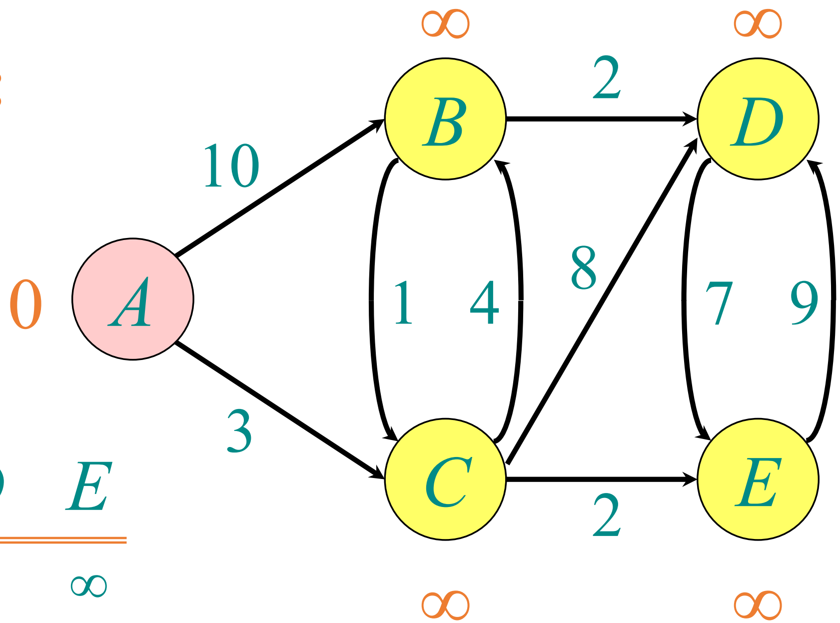


Q:	A	B	C	D	E
d(v):	0	∞	∞	∞	∞

S: {}

Dijkstra's Algorithm: Demo

Closest node is *A*:

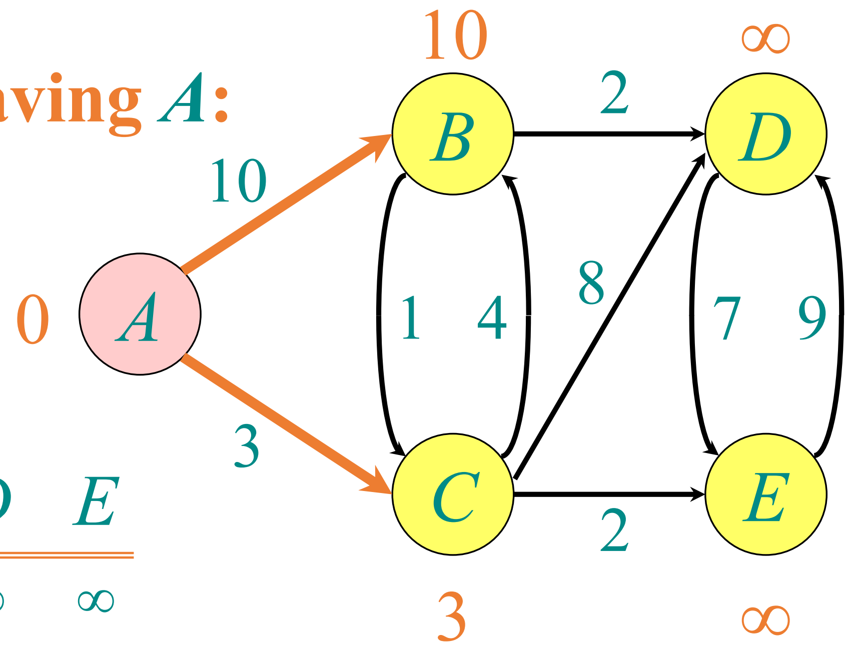


	A	*B*	*C*	*D*	*E*
d(v):	0	∞	∞	∞	∞

S: { *A* }

Dijkstra’s Algorithm: Demo

Explore edges leaving *A*:

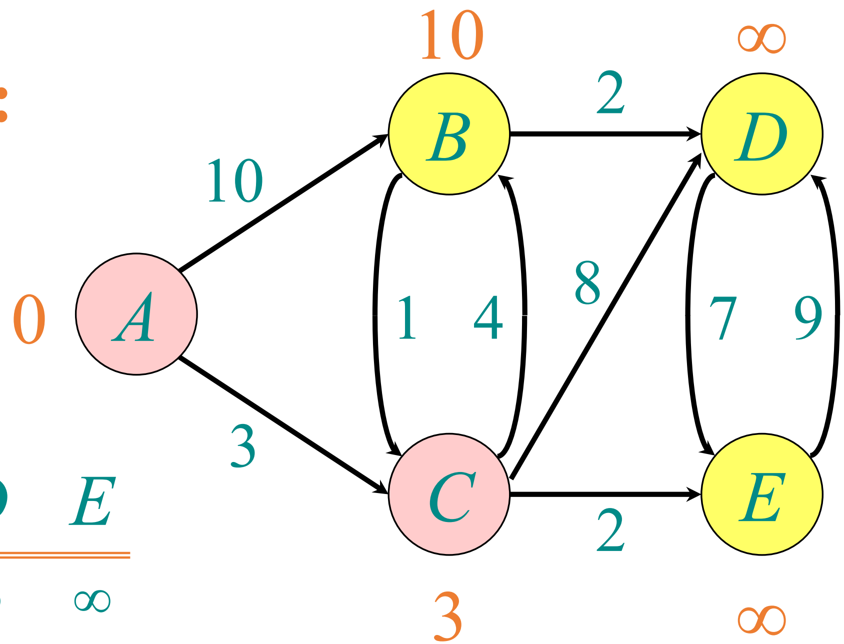


Q:	*A*	*B*	*C*	*D*	*E*
d(v):	0	∞	∞	∞	∞
		10	3	∞	∞

S: { *A* }

Dijkstra's Algorithm: Demo

Closest node is *C*:

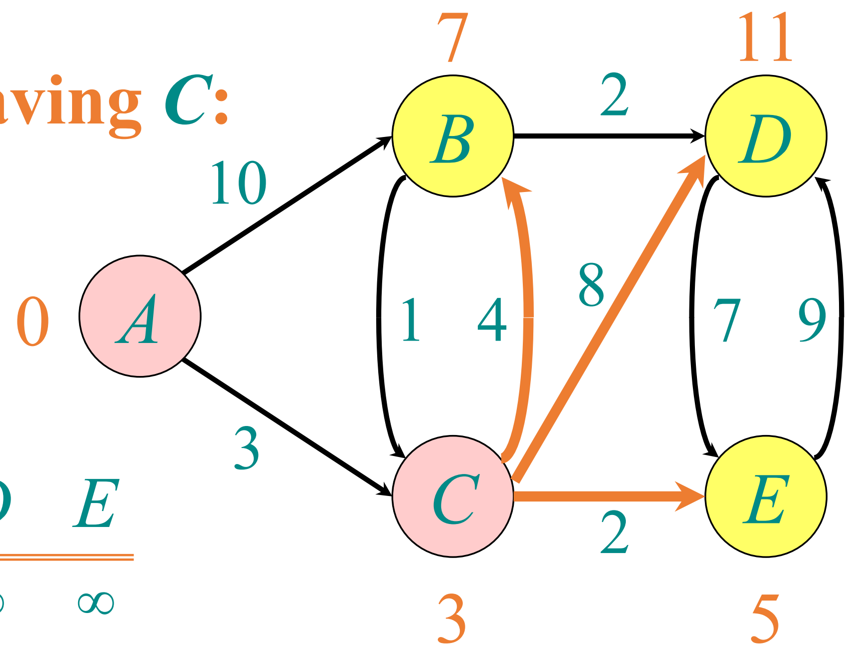


Q:	*A*	*B*	*C*	*D*	*E*
d(v):	0	∞	∞	∞	∞
		10	3	∞	∞

S: { *A*, *C* }

Dijkstra’s Algorithm: Demo

Explore edges leaving *C*:

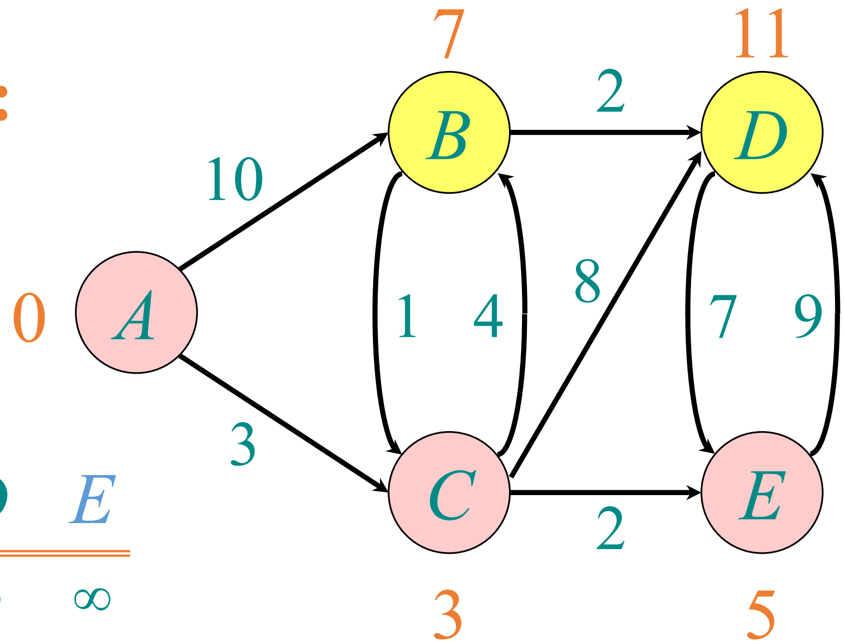


Q:	*A*	*B*	*C*	*D*	*E*
d(v):	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { *A*, *C* }

Dijkstra's Algorithm: Demo

Closest node is *E*:

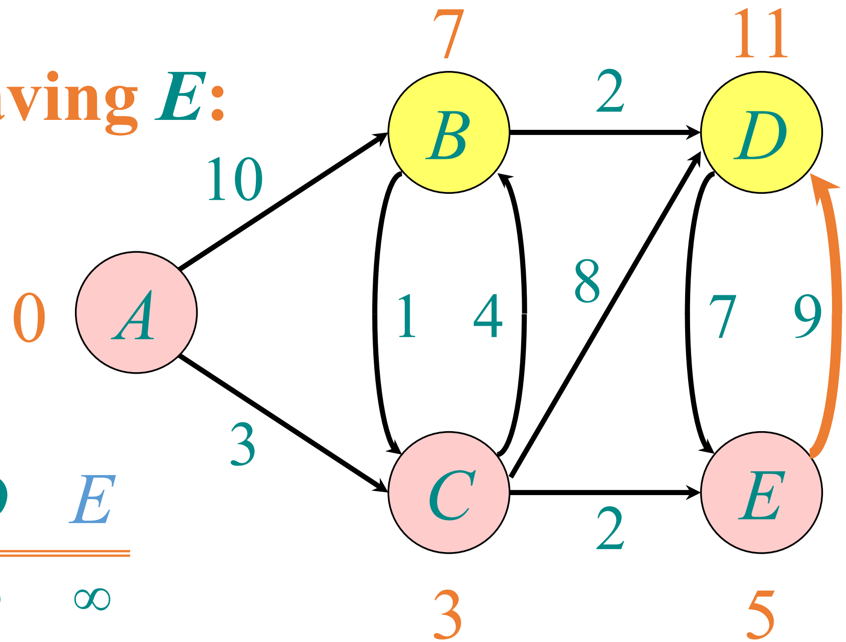


Q:	*A*	*B*	*C*	*D*	*E*
d(v):	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

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

Dijkstra's Algorithm: Demo

Explore edges leaving *E*:

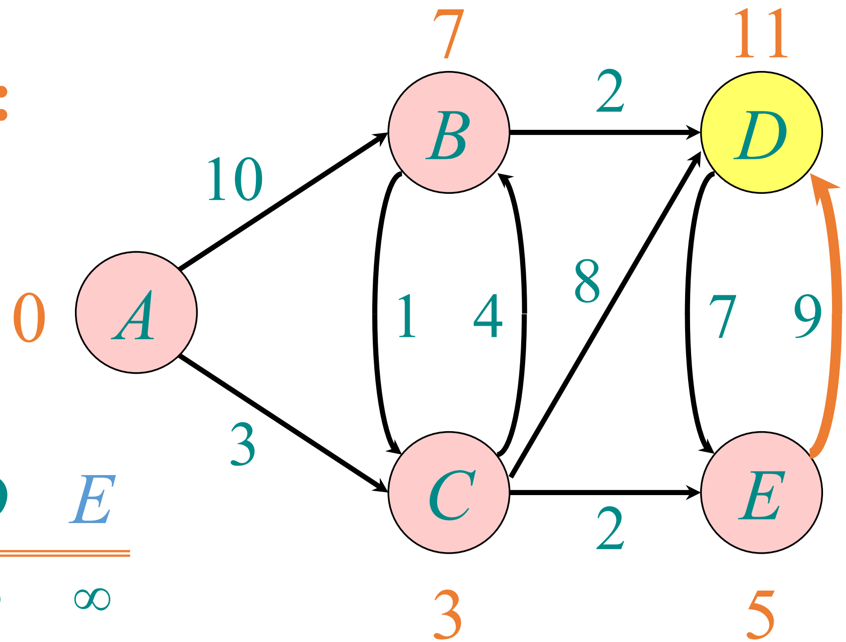


Q:	*A*	*B*	*C*	*D*	*E*
d(v):	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

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

Dijkstra’s Algorithm: Demo

Closest node is *B*:

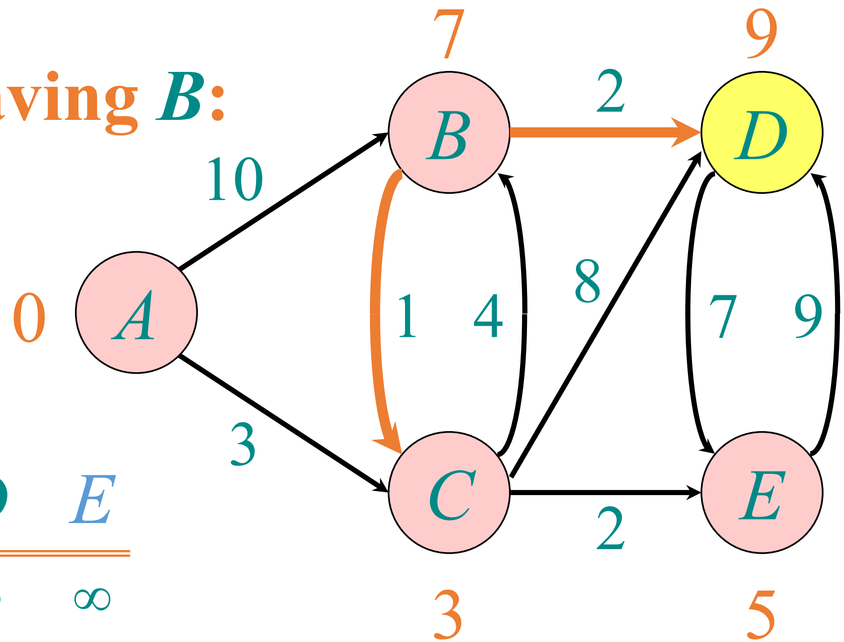


Q:	*A*	*B*	*C*	*D*	*E*
d(v):	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

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

Dijkstra's Algorithm: Demo

Explore edges leaving *B*:

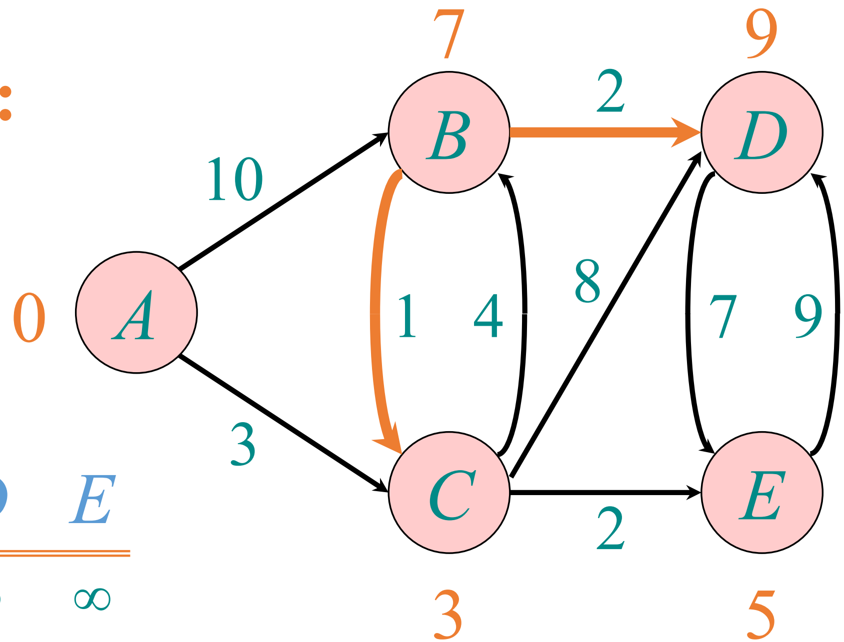


Q:	A	B	C	D	E
d(v):	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	

S: { A, C, E, B }

Dijkstra's Algorithm: Demo

Closest node is D:



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

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

Dijkstra's Algorithm (Informal)

- Invariant: if $u \in S$, then $d(u)$ is the length of the shortest path from s to u
- Proof by induction:

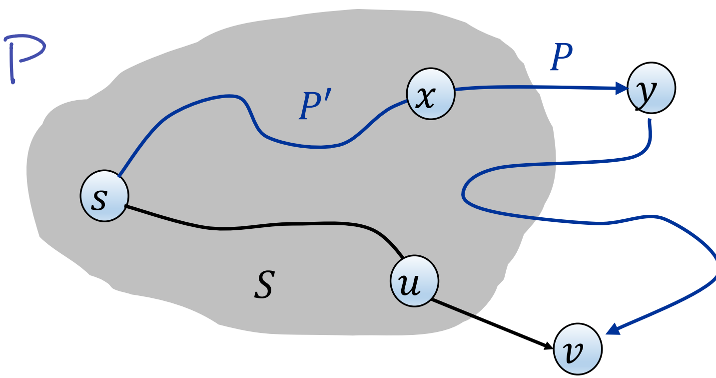
Base (explore s first): ✓

Dijkstra's Algorithm (Informal)

Since we chose v,
$d(v) \le d(y)$
$\le d(x) + l_{x,y} \le l_P$

- Invariant: if ~~u~~ v $\in S$, then $d($~~u~~ v$)$ is the length of the shortest path from s to ~~u~~ v
- Proof by induction: (Inductive Step)

- Consider what happens when I add v to S
- Clm: Any other $s \rightsquigarrow v$ path P is longer than $s \rightsquigarrow u \to v$ path I found. ($\geq d(v)$)



$$l_P \geq l_{P'} + l_{x,y} = d(x) + l_{x,y}$$

when we explored v
$d(y) \le d(x) + l_{x,y}$

Dijkstra Implementation

Dijkstra(G,s): last(v) ← ⊥

Set $d(s) \leftarrow 0$, set $d(v) \leftarrow \infty$ for $v \neq s$

Set $S \leftarrow \emptyset$, $Q \leftarrow V$

n times

While (Q is not empty):

Let $v \leftarrow \underset{w \in Q}{\operatorname{argmin}}\, d(w)$ ← Select the closest unexplored node — O(n) — O(n²) in total

Remove v from Q add it to S

For (neighbors $(v, u) \in E$):

If $(d(u) > d(v) + \ell_{v,u})$:

$d(u) \leftarrow d(v) + \ell_{v,u}$

last(u) ← v

O(deg(v))

O(m) in these steps

Assuming adjacency list

Dijkstra Implementation

Running Time:

n insert — m decrease key

n extract min

Dijkstra(G,s):

Set $d(s) \leftarrow 0$, set $d(v) \leftarrow \infty$ for $v \neq s$ — $O((n+m)\log n)$

Set $S \leftarrow \emptyset$, $Q \leftarrow V$ ← n Insert(x, ∞) — $O(m \log n)$

While (Q is not empty):

Let $v \leftarrow \underset{w \in Q}{\operatorname{argmin}}\, d(w)$ ← ExtractMin(Q)

Remove v from Q add it to S

For (neighbors $(v, u) \in E$):

If ($d(u) > d(v) + \ell_{v,u}$):

$d(u) \leftarrow d(v) + \ell_{v,u}$ ← m DecreaseKey(u, $d(u)$)

Implementing Dijkstra

- Every iteration we need to:
  - Find $v = \operatorname{argmin}_{w \notin S} d(w)$ to explore
  - Find all neighbors $(v, w) \in E$ and update d

Priority Queues (Heaps)

Priority Queues

- Want a data structure Q to store key-value pairs (k,v(k)) efficiently:
- Operations:
  - Insert(Q,k,v)
  - ExtractMin(Q)
  - DecreaseKey(Q,k,v)

Regular Queue:

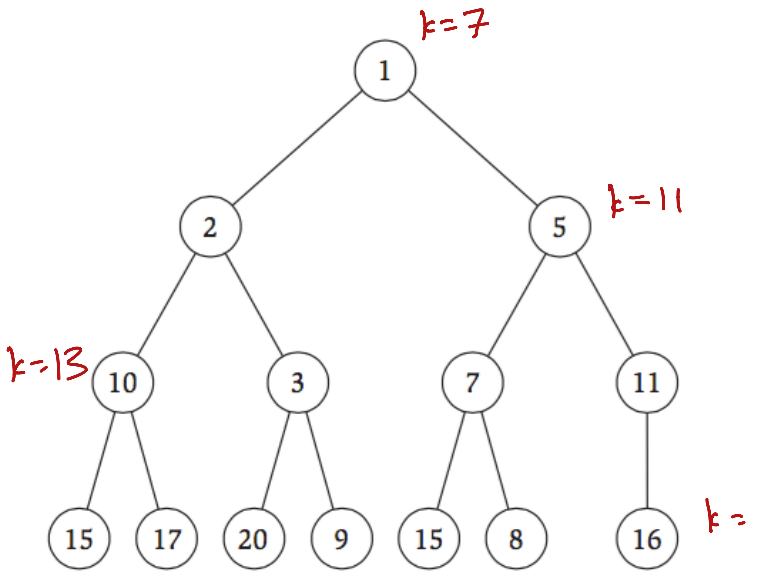
- No choice of value
- Cannot decrease

Possible Approaches?

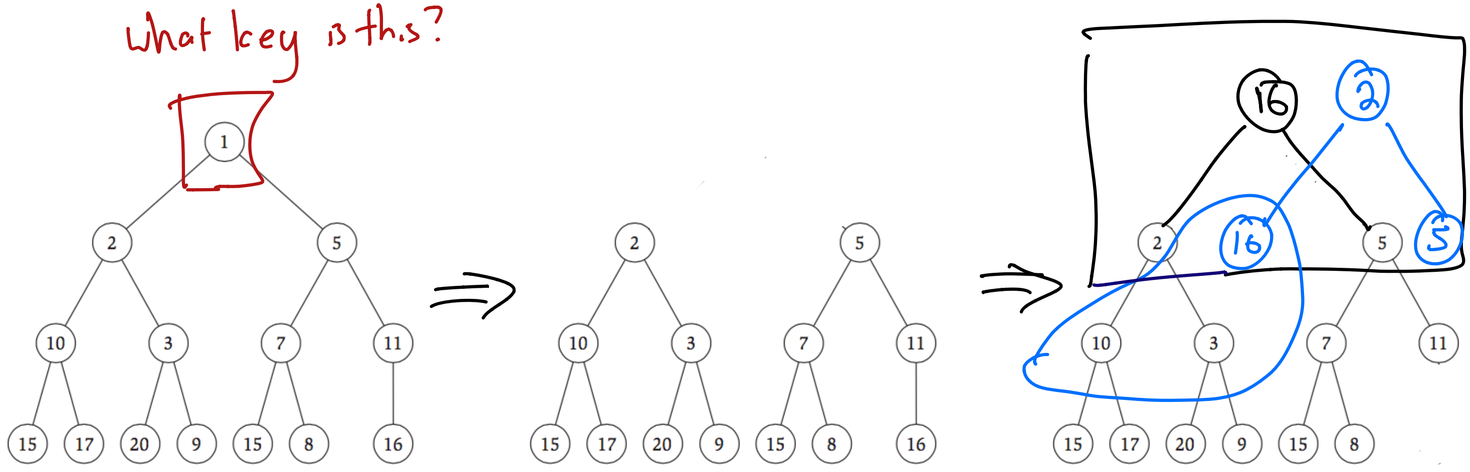
Heaps

nodes correspond to keys
labels of nodes correspond to values

- Store keys in a binary tree (can actually be an array)
- Heap Order: If k is the parent of k', $v(k) \leq v(k')$

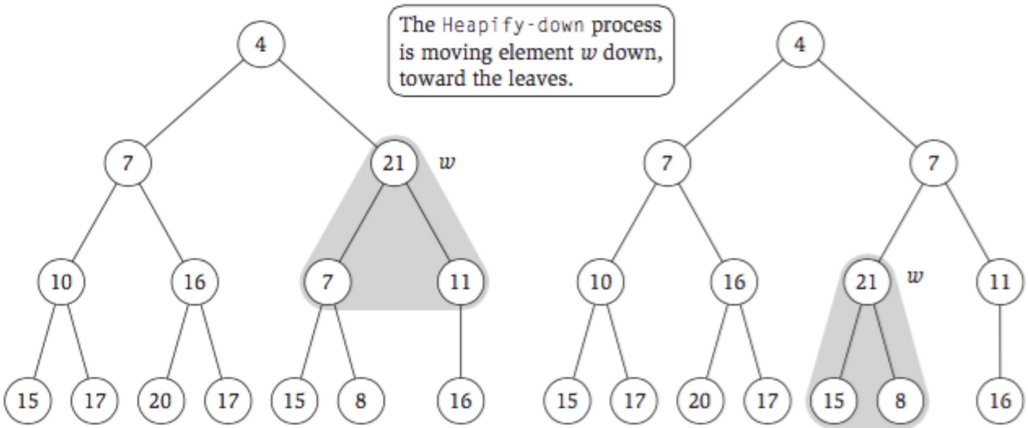


Implementing ExtractMin



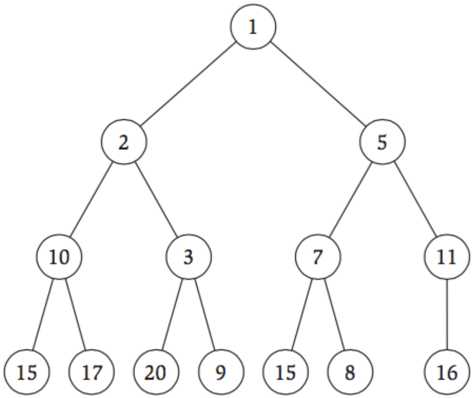
- After extracting the min I get 2 heaps
- Place the last elt at the root to get an "almost heap"
- "Repair an almost heap"

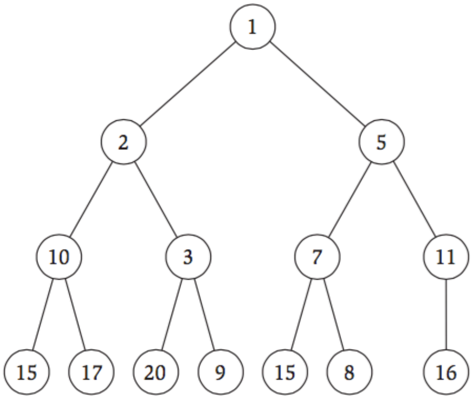
Implementing ExtractMin (HeapifyDown)



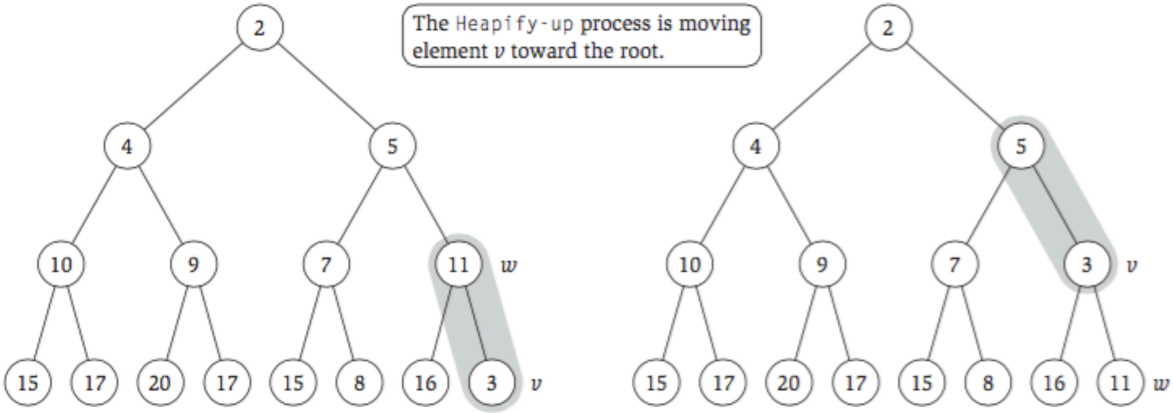
- Each swap takes $O(1)$ time
- Each swap pushes the problem down a layer
- At most $\lceil \log_2(n) \rceil$ layers $\Rightarrow$ $O(\log n)$ time

Implementing Insert

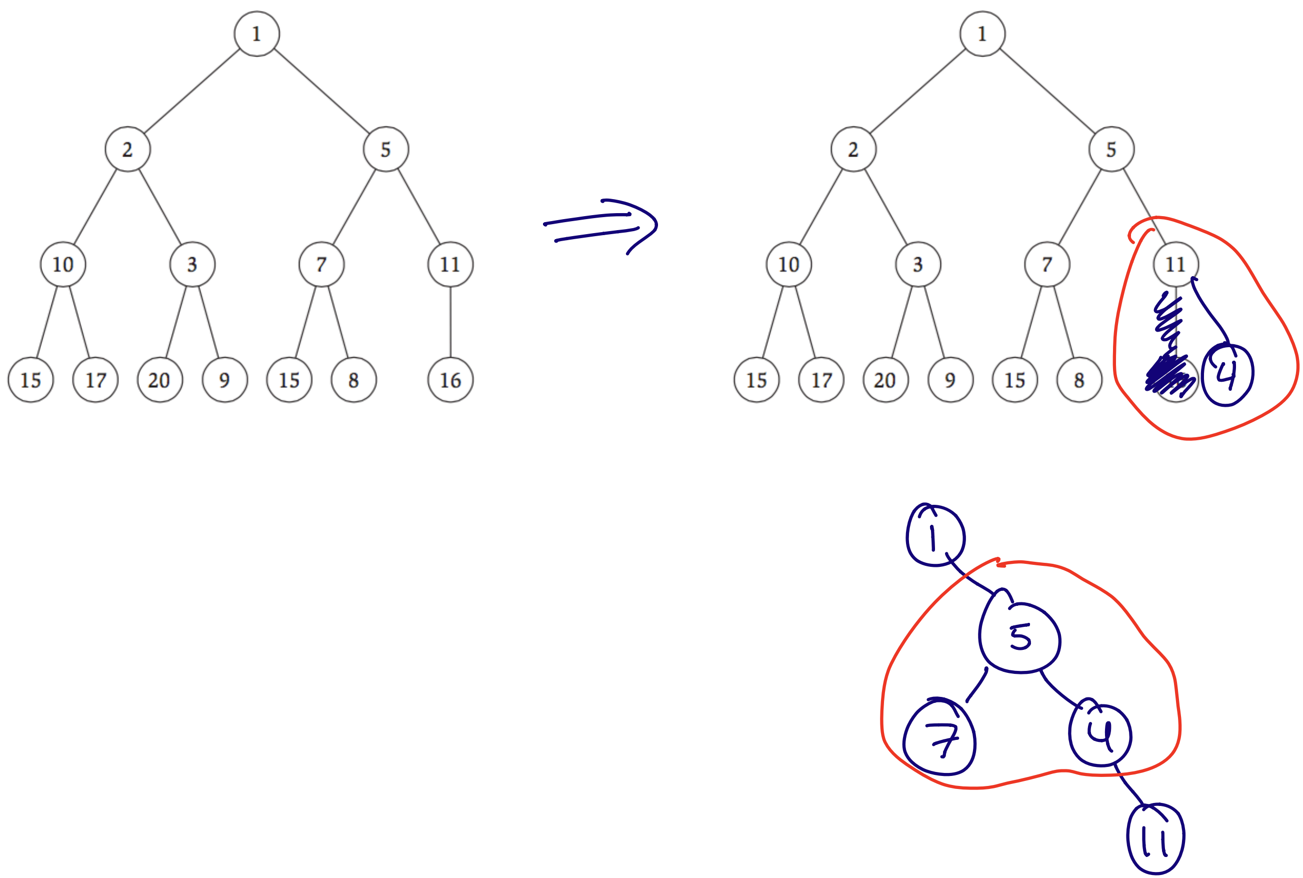




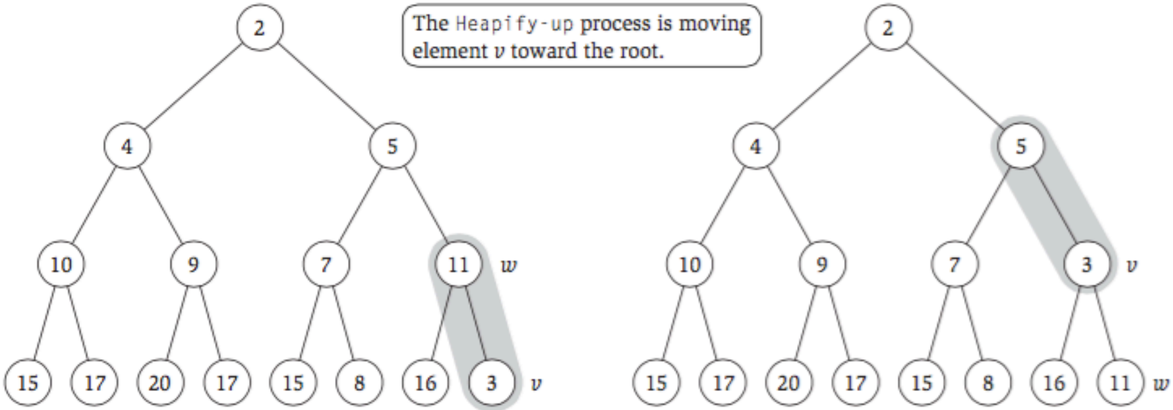
Implementing Insert (HeapifyUp)



Implementing DecreaseKey



Implementing Insert (HeapifyUp)



- Every swap takes O(1) time
- There will only be O(log n) swaps

Implementation Using Arrays

- Maintain an array H representing the nodes
- Maintain an array P mapping keys to nodes
  - Can look up the value of a given key in $O(1)$ time

- Index Arithmetic: for any node i (assuming array starts at A[1])
  - $\text{LeftChild}(i) = 2i$
  - $\text{RightChild}(i) = 2i + 1$
  - $\text{Parent}(i) = \lfloor i/2 \rfloor$

Heaps Summary

- Heapify operations take $O(\log n)$ time for n keys
- ExtractMin in $O(\log n)$:
  - Eliminate the root, move the last element to the root, restore the heap property using HeapifyDown
- Insert in $O(\log n)$:
  - Put new key in the last position, restore the heap property using HeapifyUp
- DecreaseKey in $O(\log n)$:
  - Lookup the location of the key, change its value, restore the heap property using HeapifyUp

Can find shortest paths from s to all other nodes in time $O(m \log n)$.

Back to Dijkstra's Algorithm

Dijkstra Implementation

Dijkstra(G,s):

Set $d(s) \leftarrow 0$, set $d(v) \leftarrow \infty$ for $v \neq s$

Set $S \leftarrow \emptyset$, $Q \leftarrow V$

While (Q is not empty):

Let $v \leftarrow \operatorname{argmin}_{v \in Q} d(v)$

Remove v from Q add it to S

For (neighbors $(v, u) \in E$):

If ($d(u) > d(v) + \ell_{v,u}$):

$d(u) \leftarrow d(v) + \ell_{v,u}$

Dijkstra Summary

- Can find the distance from s to all other nodes $v \in V$ in time $O(m \log n)$
  - Use a heap to implement the priority queue
  - Can get $O(m + n \log n)$ using fancier priority queues

- How do we find the shortest paths themselves?

Dijkstra Implementation

Dijkstra(G,s):

Set $d(s) \leftarrow 0$, set $d(v) \leftarrow \infty$ for $v \neq s$

Set $\text{last}(v) \leftarrow \perp$

Set $S \leftarrow \emptyset$, $Q \leftarrow V$

While (Q is not empty):

  Let $v \leftarrow \underset{v \in Q}{\text{argmin}}\, d(v)$

  Remove v from Q add it to S

  For (neighbors $(v, u) \in E$):

    If ($d(u) > d(v) + \ell_{v,u}$):

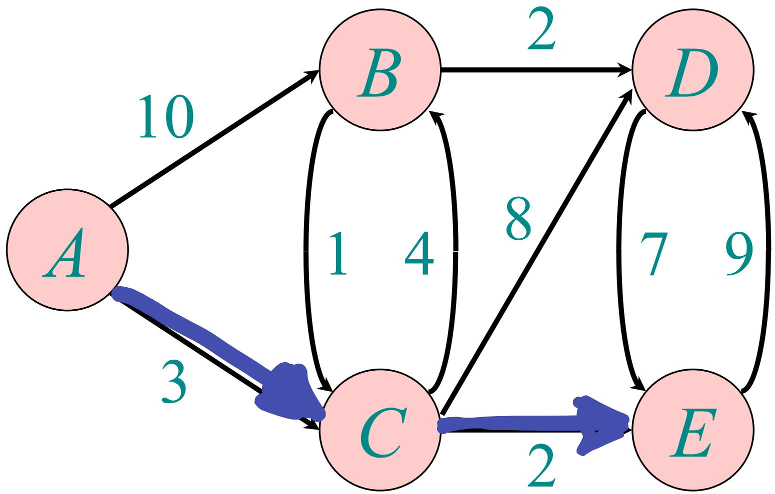
      $d(u) \leftarrow d(v) + \ell_{v,u}$

      $\text{last}(u) \leftarrow v$

Dijkstra's Algorithm: Demo

last(C) = A

last(E) = C



the actual shortest path from A to E is

last(last(E)) ↙ repeat until you get to A