CS3000: Algorithms & Data
Jonathan Ullman

Jonathan Ullman

Lecture 11:

- Shortest Paths: BFS, Start Dijkstra

Feb 24, 2020

Shortest Paths:
Breadth-First Search

Exploring a Graph

- **Problem:** Is there a path from s to t?
- **Idea:** Explore all nodes reachable from s.

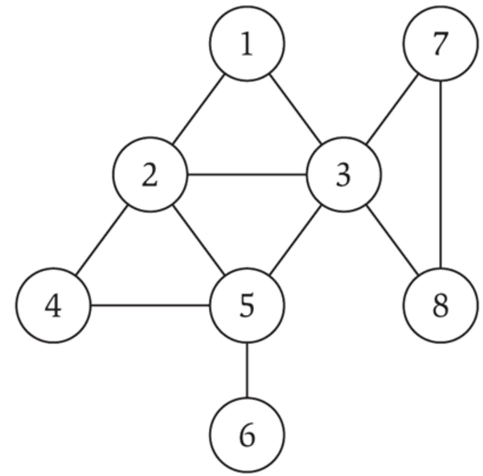
- Two different search techniques:
  - **Depth-First Search:** follow a path until you get stuck, then go back
  - **Breadth-First Search:** explore all nearby nodes before moving on to farther away nodes
    - Finds the shortest path from s to t!

Breadth-First Search (BFS)

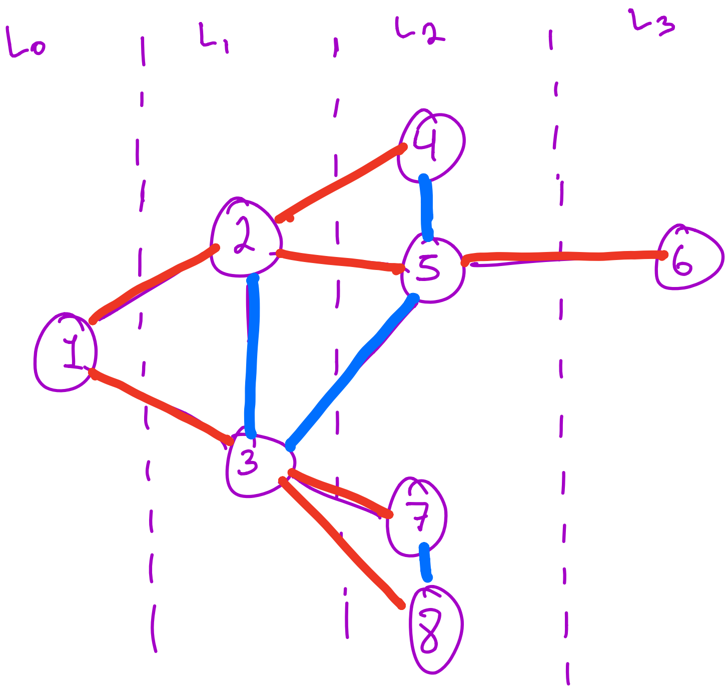
- **Informal Description:** start at s, find neighbors of s, find neighbors of neighbors of s, and so on…

- BFS Tree:
  - $L_0 = \{s\}$
  - $L_1 =$ all neighbors of L_0
  - $L_2 =$ all neighbors of L_1 that are not in L_0, L_1
  - $L_3 =$ all neighbors of L_2 that are not in L_0, L_1, L_2
  - …
  - $L_d =$ all neighbors of L_{d-1} that are not in $L_0, \ldots, L_{d-1}$
  - Stop when L_{d+1} is empty

Example



- BFS this graph from $\boldsymbol{s = 1}$



- Red edges are "tree edges"
- Red edges give paths from s to t
- Blue edges are either $L_i \leftrightarrow L_i$ or $L_i \leftrightarrow L_{i+1}$

Breadth-First Search Implementation

≈ NULL

```
BFS(G = (V,E), s):
  Let explored[v] ← false ∀v, explored[s] ← true
  Let layer[v] ← ∞ ∀v, layer[s] ← 0
  Let parent[v] ← ⊥ ∀v
  Let i ← 0, L_0 = {s}, T ← ∅

  While (L_i is not empty):
    Initialize new layer L_{i+1}
    For (u in L_i):
      For ((u,v) in E):
        If (explored[v] = false):
          explored[v] ← true,
          layer[v] ← i+1
          parent[v] ← u    (Add (u,v) to T)
          Add v to L_{i+1}
    i ← i+1
```

BFS Running Time (Adjacency List)

```
BFS(G = (V,E), s):
  Let explored[v] ← false ∀v, explored[s] ← true
  Let layer[v] ← ∞ ∀v, layer[s] ← 0
  Let parent[v] ←⊥ ∀v
  Let i ← 0, L_0 = {s}, T ← ∅

  While (L_i is not empty):
    Initialize new layer L_{i+1}
    For (u in L_i):
      For ((u,v) in E):
        If (explored[v] = false):
          explored[v] ← true,
          layer[v] ← i+1
          parent[v] ← u
          Add v to L_{i+1}
    i ← i+1
```

$O(n)$

each node occurs once

$\deg(u)$ times

$O(1)$ per edge

$$\sum_{u \in V} O(\deg(u)+1) = O(n) + \sum_{u \in V} O(\deg(u)) = \underline{\underline{O(n+m)}}$$

Shortest Paths via BFS

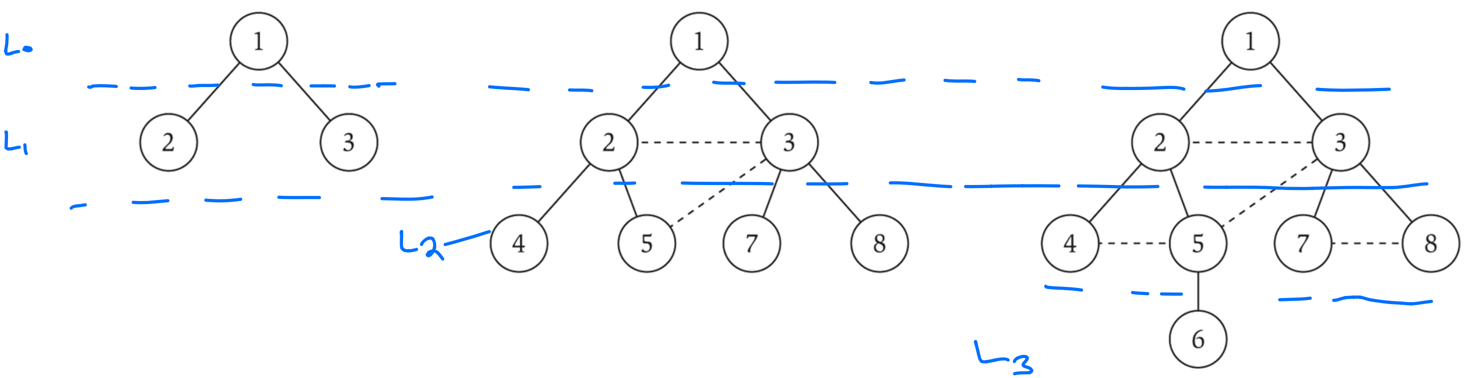
$d(s,t)$ or $d(s \to t)$

or $dist(s,t)$ or $dist(s \to t)$

- **Definition:** the distance between s, t is the number of edges on the shortest path from s to t

  • If t not reachable from s then $d(s,t) = \infty$

- **Thm:** BFS finds distances from s to other nodes
  - L_i contains all nodes at distance i from s



Shortest Paths via BFS

- **Definition:** the distance between s, t is the number of edges on the shortest path from s to t
- **Thm:** BFS finds distances from s to other nodes
  - $\forall i$ L_i contains all nodes at distance i from s

Base Cases: L_0 is obvious

L_1 is obvious (L_1 contains all neighbors of s)

Induction: If true for $L_0, L_1, \ldots, L_i$ then true for L_{i+1}

Suppose u is such that $d(s,u) = i+1$

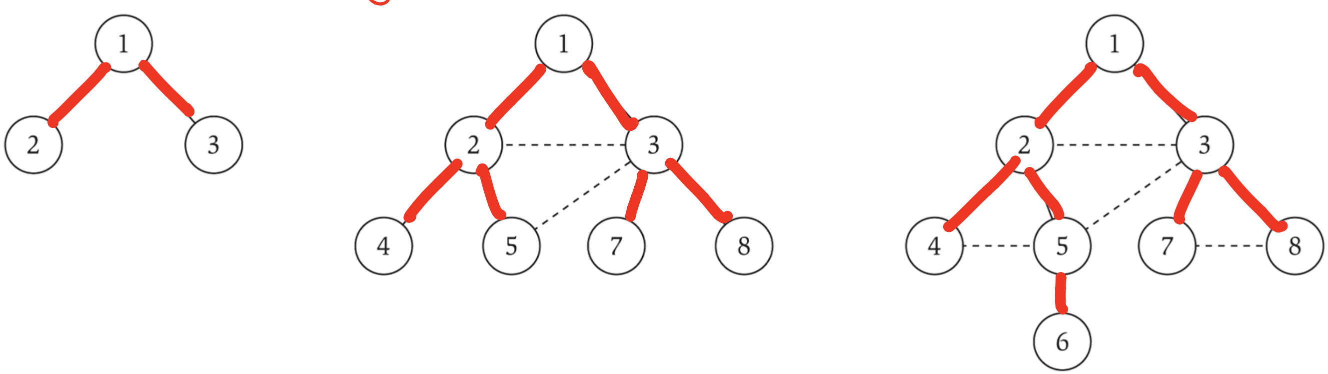
$$s \xrightarrow{i \text{ hops}} v \longrightarrow u$$

By induction, v is in L_i. Therefore u is in L_{i+1}

Shortest Paths via BFS

- **Definition:** the distance between s, t is the number of edges on the shortest path from s to t
- **Thm:** BFS finds distances from s to other nodes and the tree edges give the shortest s to t path
  - Can find distances and shortest path tree in time $O(n+m)$… then can find a shortest path in time $O(n)$

Tree edges give shortest paths

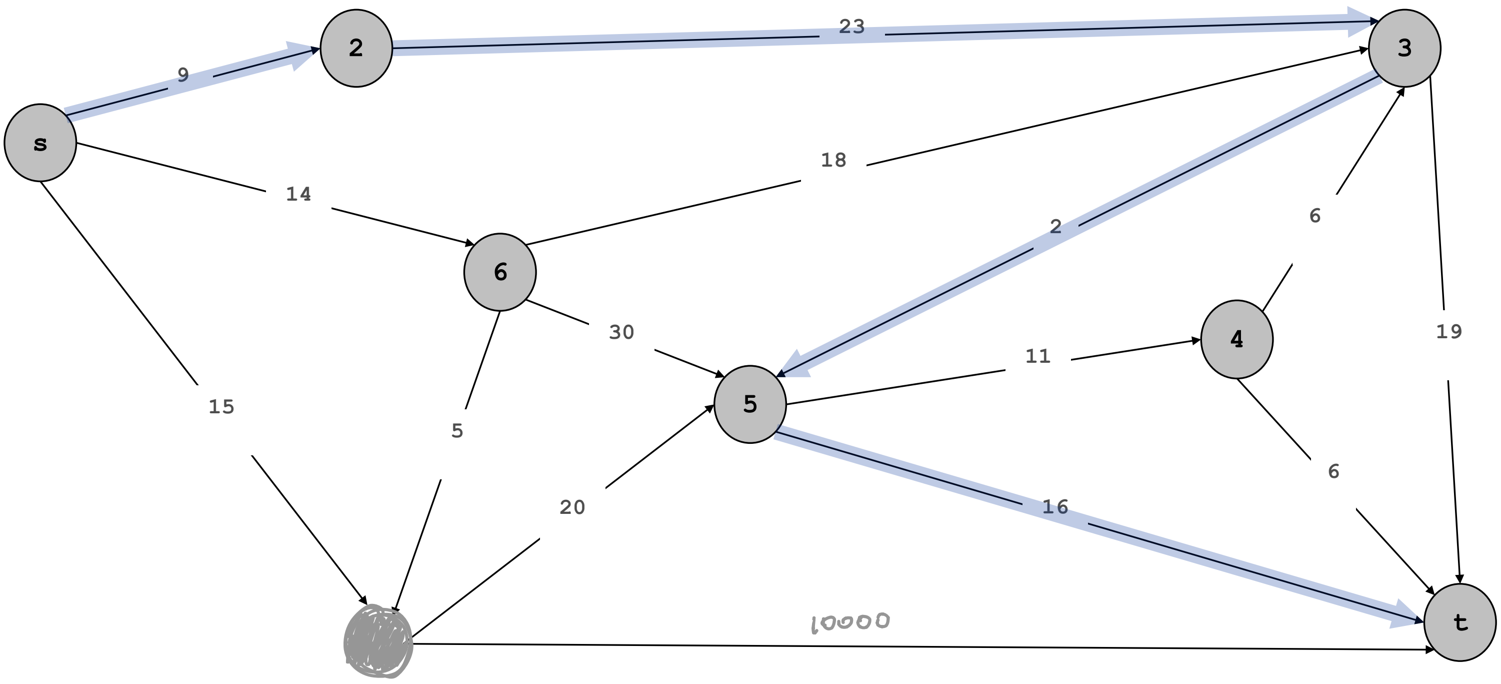


Shortest Paths via BFS

- **Definition:** the distance between s, t is the number of edges on the shortest path from s to t
- **Thm:** BFS finds distances from s to other nodes and the tree edges give the shortest s to t path
  - Can find distances and shortest path tree in time $O(n+m)$… then can find a shortest path in time $O(n)$

Shortest Paths: Dijkstra

Navigation



Weighted Graphs

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

- **Today:**
  - Directed graphs (one-way streets)
  - Strongly connected (there is always some path)
  - Non-negative edge lengths ($\ell(e) \geq 0$)

Shortest Paths

- The length of a path $P = v_1 - v_2 - \cdots - v_k$ is the sum of the edge lengths

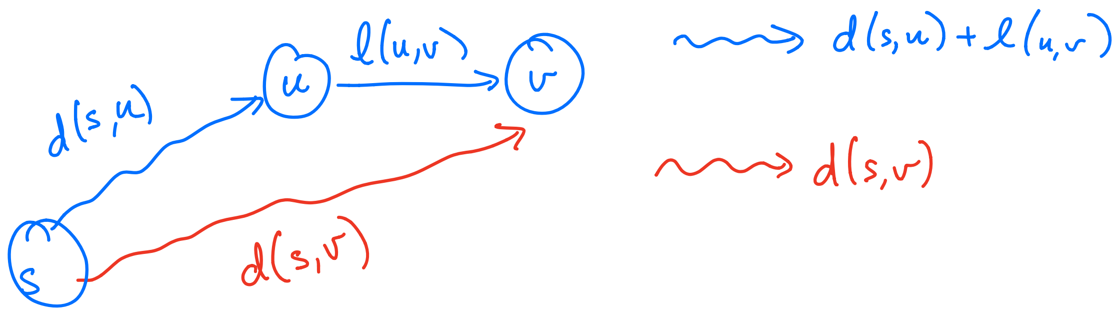
$$\ell(P) = \sum_{e \in P} \ell(e)$$

- The distance $d(s, t)$ is the length of the shortest path from s to t
- **Shortest Path:** given nodes $s, t \in V$, find the shortest path from s to 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

- If $(u, v) \in E$, then $d(s, v) \leq d(s, u) + \ell(u, v)$ for every node $s \in V$

- If $(u, v) \in E$, and $d(s, v) = d(s, u) + \ell(u, v)$ then there is a shortest $s \rightsquigarrow v$-path ending with (u, v)

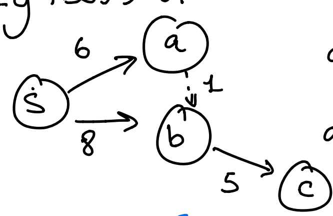


Dijkstra's Algorithm

- Maintain an upper bound on $d(s,t)$ $\forall t$

  $d[s]=0 \qquad d[t]=\infty$ for $t \neq s$

- Explore neighbors of s



$d[a] = 6$

$d[b] = 8$

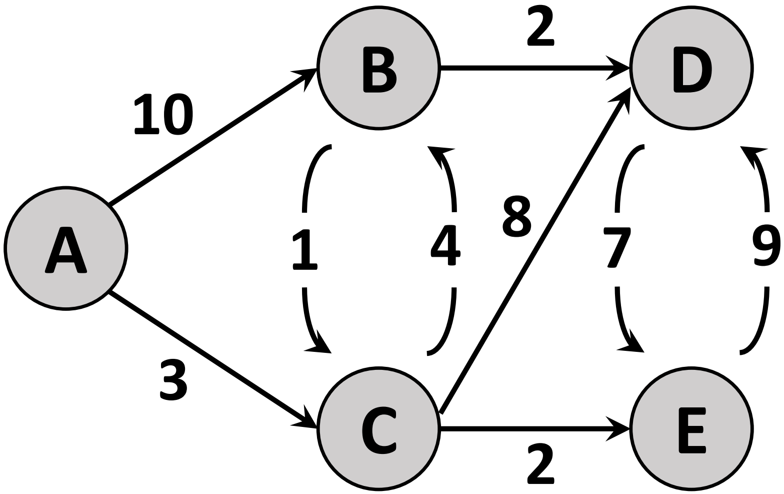
New upper bounds

- Find another node [with the smallest $d[u]$ of all unexplored nodes]

  Explore neighbors of that node

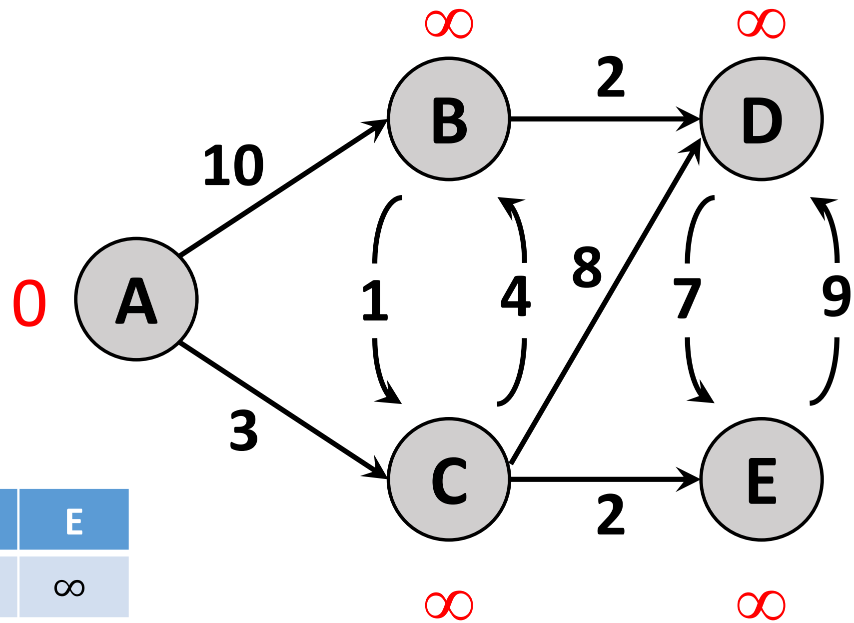
- Repeat until all nodes are explored

Dijkstra’s Algorithm: Demo



Dijkstra's Algorithm: Demo

Initialize



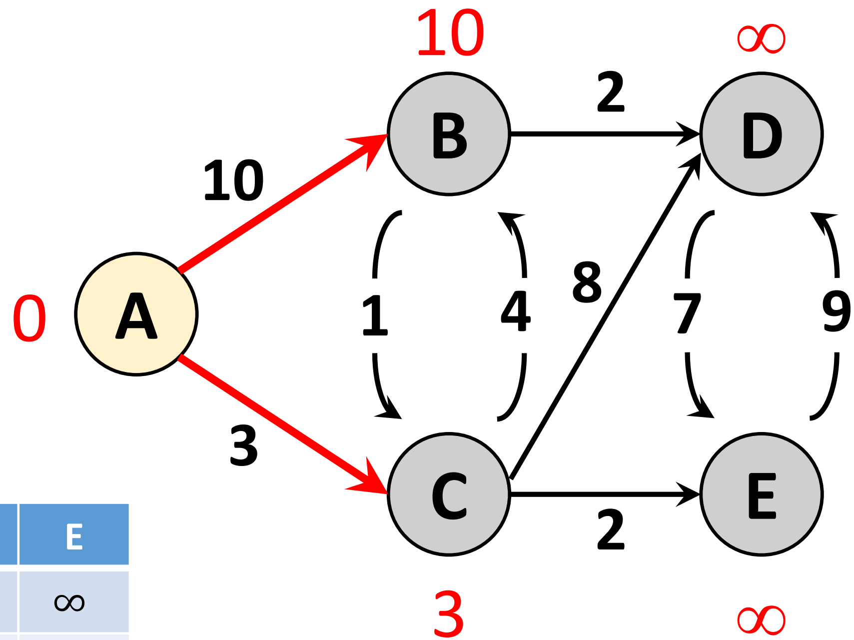
	A	B	C	D	E
$d_0(u)$	**0**	∞	∞	∞	∞

$$S = \{\}$$

Set of explored nodes

Dijkstra's Algorithm: Demo

Explore A

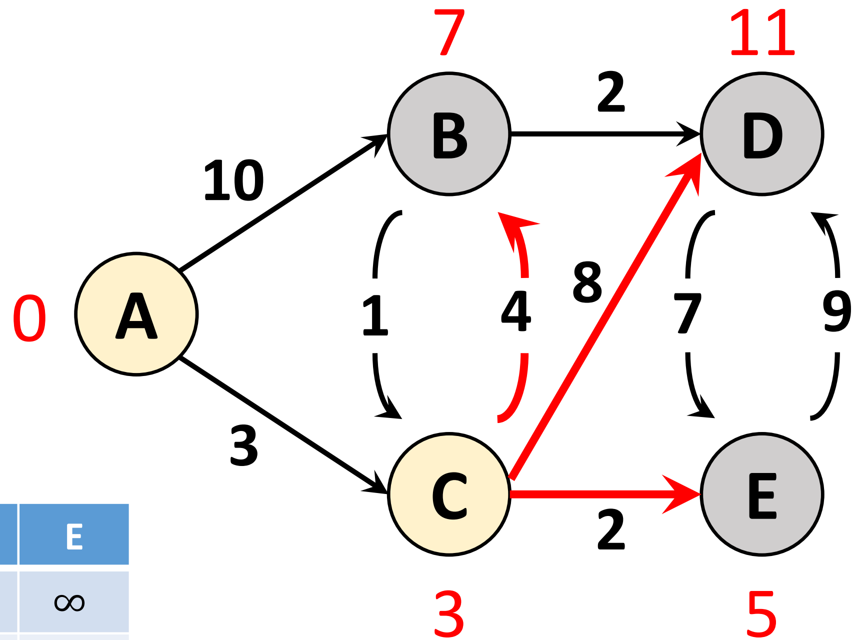


	A	B	C	D	E
$d_0(u)$	**0**	∞	∞	∞	∞
$d_1(u)$	**0**	**10**	**3**	∞	∞

$$S = \{A\}$$

Dijkstra's Algorithm: Demo

Explore C

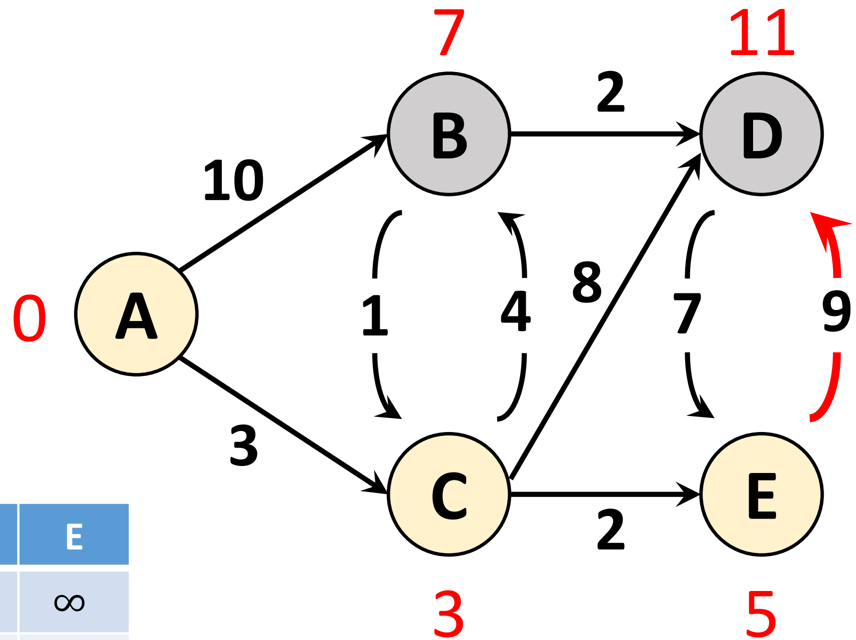


	A	B	C	D	E
$d_0(u)$	0	∞	∞	∞	∞
$d_1(u)$	0	10	3	∞	∞
$d_2(u)$	0	7	3	11	5

$$S = \{A, C\}$$

Dijkstra's Algorithm: Demo

Explore E

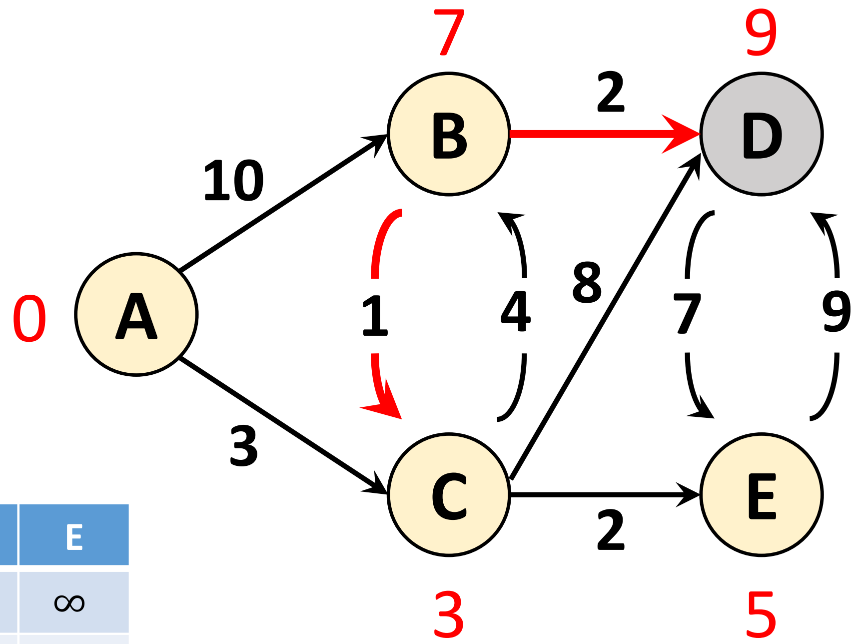


	A	B	C	D	E
$d_0(u)$	**0**	∞	∞	∞	∞
$d_1(u)$	**0**	**10**	**3**	∞	∞
$d_2(u)$	**0**	**7**	**3**	**11**	**5**
$d_3(u)$	**0**	**7**	**3**	**11**	**5**

$$S = \{A, C, E\}$$

Dijkstra's Algorithm: Demo

Explore B

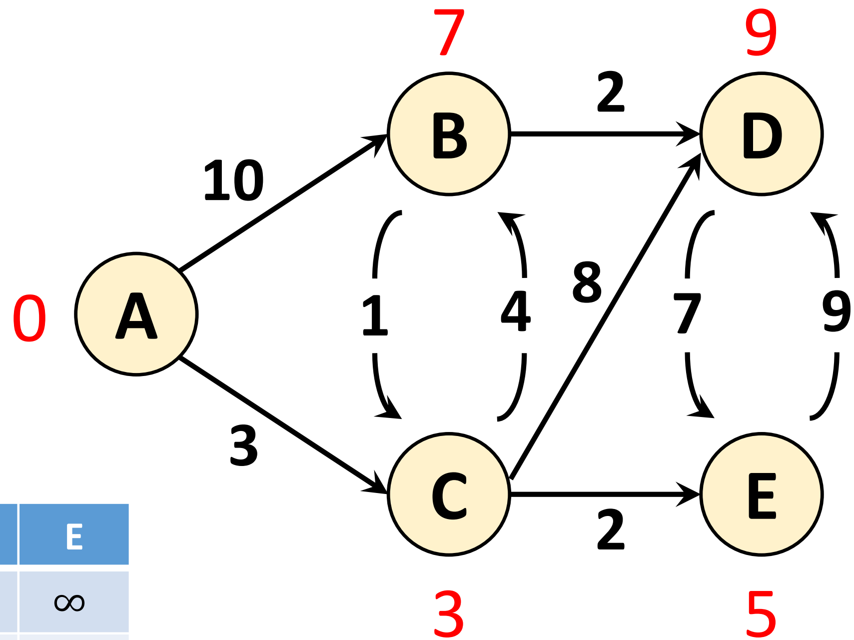


	A	B	C	D	E
$d_0(u)$	0	∞	∞	∞	∞
$d_1(u)$	0	10	3	∞	∞
$d_2(u)$	0	7	3	11	5
$d_3(u)$	0	7	3	11	5
$d_4(u)$	0	7	3	9	5

$$S = \{A, C, E, B\}$$

Dijkstra's Algorithm: Demo

Don't need to explore D

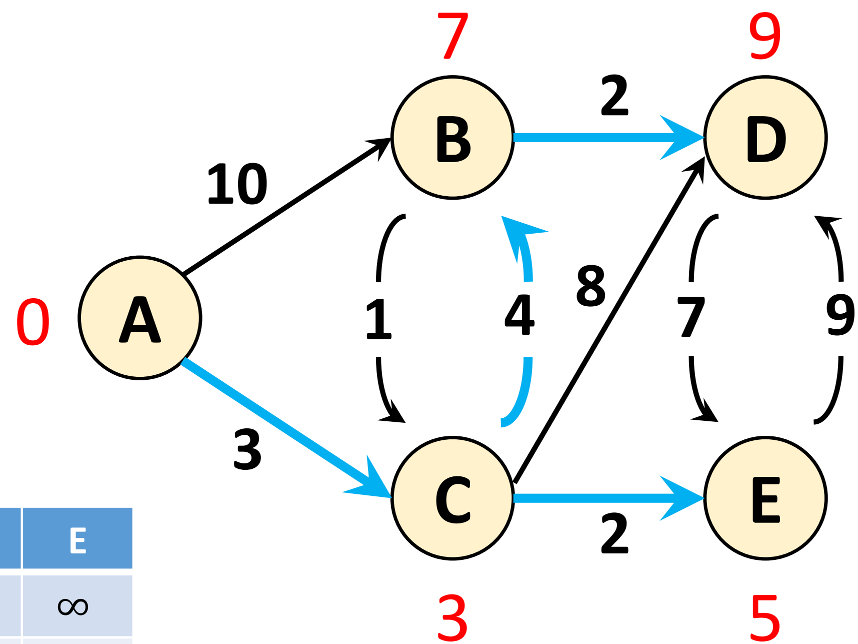


	A	B	C	D	E
$d_0(u)$	0	∞	∞	∞	∞
$d_1(u)$	0	10	3	∞	∞
$d_2(u)$	0	7	3	11	5
$d_3(u)$	0	7	3	11	5
$d_4(u)$	0	7	3	9	5

$$S = \{A, C, E, B, D\}$$

Dijkstra’s Algorithm: Demo

Maintain parent pointers so we can find the shortest paths



	A	B	C	D	E
$d_0(u)$	0	∞	∞	∞	∞
$d_1(u)$	0	10	3	∞	∞
$d_2(u)$	0	7	3	11	5
$d_3(u)$	0	7	3	11	5
$d_4(u)$	0	7	3	9	5

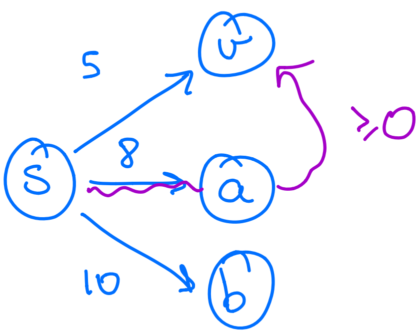
Correctness of Dijkstra

- **Warmup 0:** initially, $d_0(s)$ is the correct distance

- **Warmup 1:** after exploring the ~~first~~ second node v, $d_1(v)$ is the correct distance

If (s,v) is the shortest edge starting at s. Then

$d(s,v) = \ell(s,v)$



Any other $s \rightsquigarrow v$ path has length ≥ 5, so it's not a shorter path

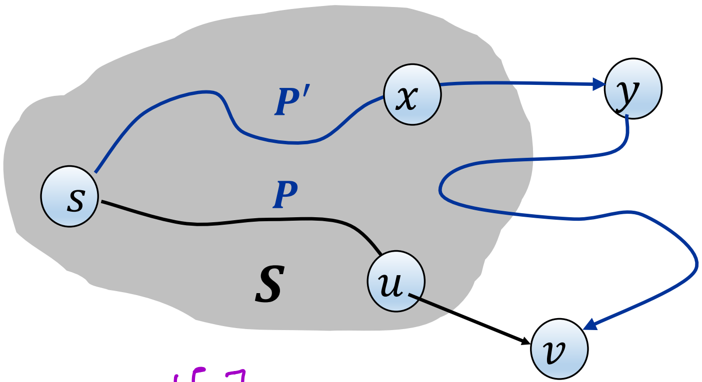
Correctness of Dijkstra

shortest path we've found after exploring i nodes

- **Invariant:** after we explore the i-th node, $\hat{d}_i(v)$ is correct for every $v \in S$

- We just argued the invariant holds after we've explored the 1st and 2nd nodes

Correctness of Dijkstra



- **Invariant:** after we explore the i-th node, $d_i(v)$ is correct for every $v \in S$
- **Proof:**

parent[v] = u

next node we explore

Want to show that $d_i(v) = d_i(u) + \ell(u,v)$ is the shortest path

$$\ell(P') = \ell(P_{s,x}) + \ell(x \to y) + \ell(P_{y,v})$$

$$\geq \ell(P_{s,x}) + \ell(x \to y) \quad [\ell(e) \geq 0]$$

Not totally obvious

$$\geq d_i(x) + \ell(x \to y) \quad [x \text{ is explored}]$$

$$\geq d_i(y) \quad [x \text{ is explored}]$$

$$\geq d_i(v) \quad [\text{b/c I chose } v, \text{ not } y]$$

$$= \ell(P)$$

Implementing Dijkstra

```
Dijkstra(G = (V,E,{ℓ(e)}, s):
  d[s] ← 0, d[u] ← ∞ for every u != s
  parent[u] ←⊥ for every u
  Q ← V                // Q holds the unexplored nodes

  While (Q is not empty):
    u ← argmin_{w∈Q} d[w]     //Find closest unexplored
    Remove u from Q

    // Update the neighbors of u
    For ((u,v) in E):
      If (d[v] > d[u] + ℓ(u,v)):
        d[v] ← d[u] + ℓ(u,v)
        parent[v] ← u

  Return (d, parent)
```

Implementing Dijkstra (Naïvely)

1
- Need to explore all n nodes
- Each exploration requires:
  - 2a Finding the unexplored node u with smallest distance
  - 2b Updating the distance for each neighbor of u

Find the node w/ minimum value

Decrease the value associated with a given node

2a $O(n)$ time

2b $O(\deg(u) + 1)$

$$\sum_{u \in V} O(n + \deg(u) + 1) = O(n^2 + m)$$