Algorithm Design and Analysis

CSE 565

LECTURE 6

Greedy Graph Algorithms

- Shortest paths

Adam Smith

A. Smith; based on slides by E. Demaine, C. Leiserson, S. Raskhodnikova, K. Wayne

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, etc)
  - writing recursive algorithms: don’t confuse the recursive subroutine with the first call
  - label global variables clearly

Writing proofs

- 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

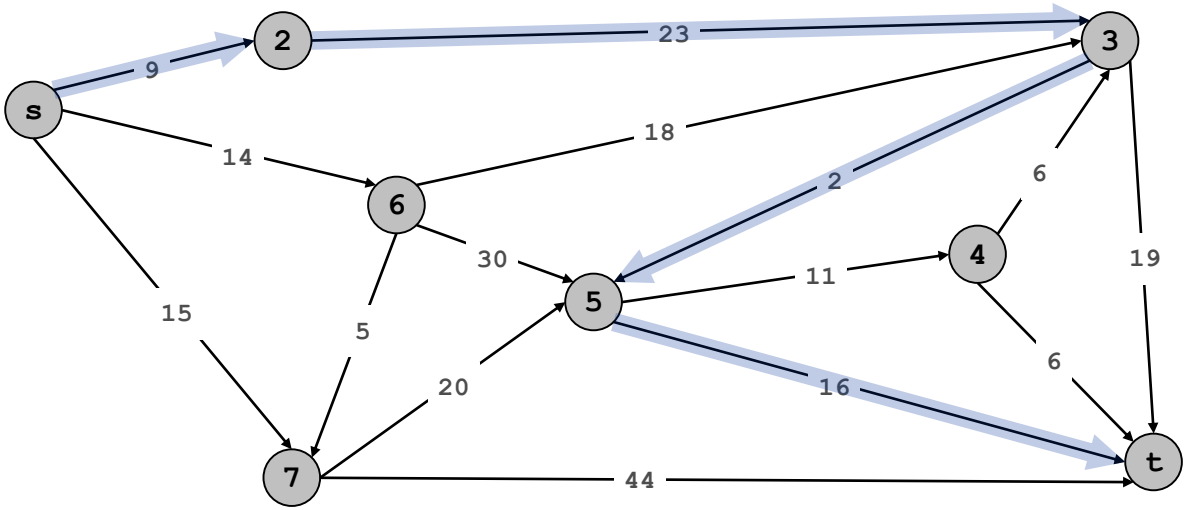
- Goals:
  - Reinforce and clarify material from lecture
  - Develop your skills
    - problem-solving and
    - Communication
- Make sure you understand the solution
- Use the feedback
- If you don’t understand something, ask!
  - Me or the TA or on Piazza

Shortest Paths

A. Smith; based on slides by E. Demaine, C. Leiserson, S. Raskhodnikova, K. Wayne

Single-source 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 path = sum of edge lengths
- **Find:** shortest directed path from s to t.



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

When is there a shortest path?

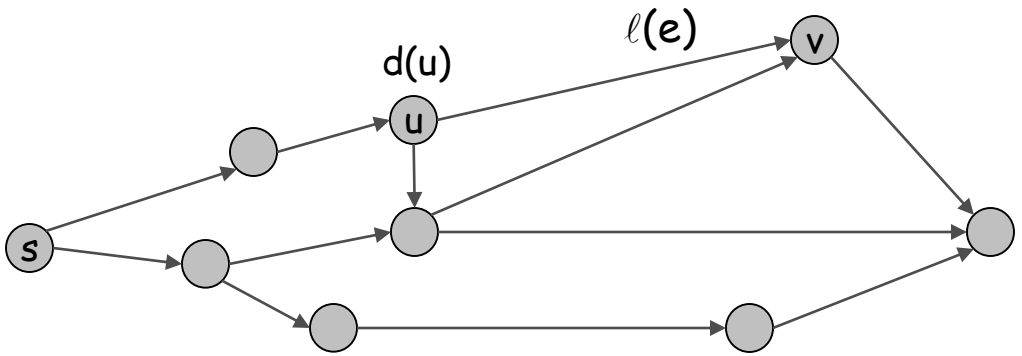
- Edge weights nonnegative:
  - shortest path always exists

- Negative weight edges:
  - shortest path **may** exist or not
  - If *no negative cycles in G*, then shortest path exists
  - If *negative cycles in G*, then no shortest path

Generic idea: relaxing edges

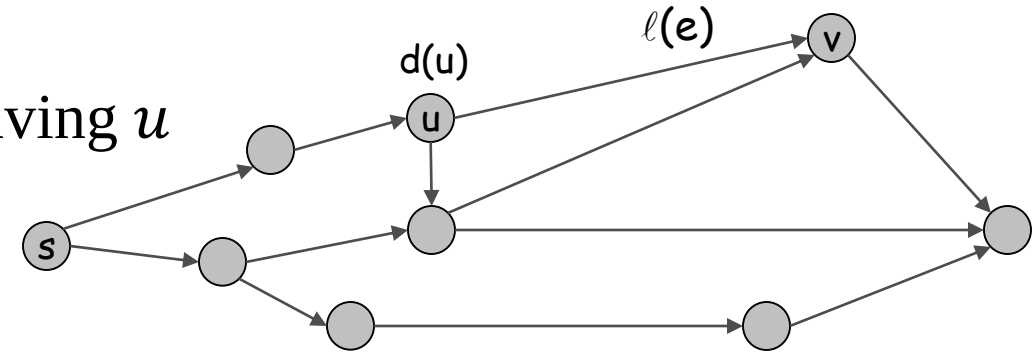
- For each vertex $v \in V$, maintain
  - Distance estimate $d(v)$ from s to v (initially ∞)
  - Predecessor $\pi(v)$ on shortest path from s to v (initially NIL)
- Invariant: $d(v)$ is the length of some path from s to v that ends with edge $\pi(v) \rightarrow v$.
- An edge $u \rightarrow v$ is **tense** if $d(v) > d(u) + \ell(u, v)$
  - This means that *d(v)* is too big!
- Relax(u, v)
  - $d(v) = d(u) + \ell(u, v)$
  - $\pi(v) = u$
- Note: edges leaving v may now be tense

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



Generic algorithm

- Maintain a "bag" of **nodes T** to be considered.
- Initialize $T = \{s\}, d(s) = 0$.
- $d(v) = \infty$ for all $v \neq s$
- While T is not empty
  - Take a vertex u from T
  - For each edge $u \rightarrow v$ leaving u
    - If $u \rightarrow v$ is tense,
      - **Relax(u,v)**
      - Put or update v in T



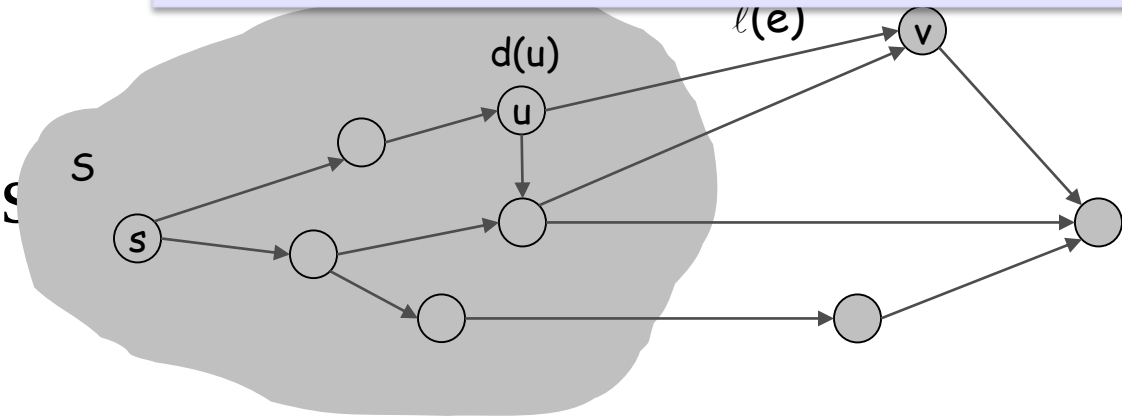
Proof of correctness: If no tense edges, then all $d(u)$ are correct.

(Proof on board.)

Dikstra's aglorithm

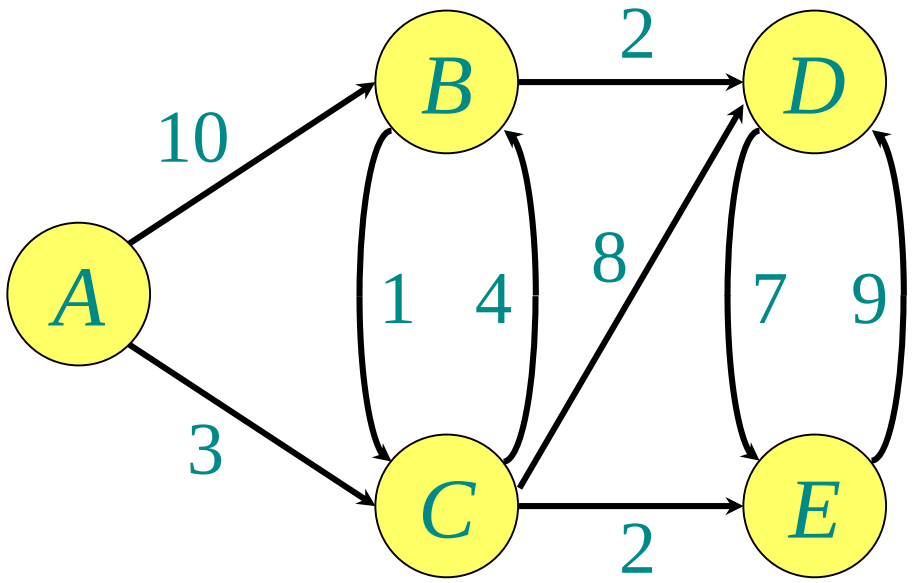
- Priority queue to maintain the bag T
  - Key = d(v)
- Process unexplored node in ascending order of estimate $d(v)$
- Usual implementation:
  - Never process a vertex twice!
  - Mark vertices as done (using set ***S***)
- Generic correctness proof breaks down

- While T is not empty
  - Take a vertex u from T
  - **Add u to S**
  - For each edge $u \rightarrow v$ leaving u
    - If $v \notin S$ and (u, v) is tense,
      - **Relax(u,v)**
      - Put (or update) v in T



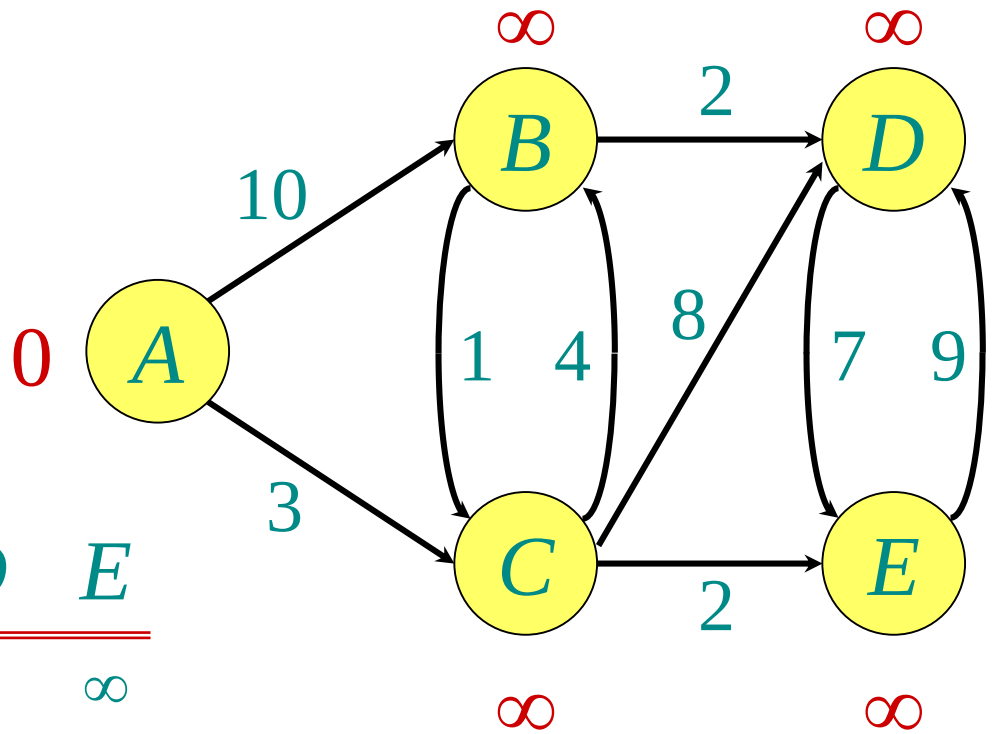
Demo of Dijkstra's Algorithm

Graph with nonnegative edge lengths:



Demo of Dijkstra's Algorithm

Initialize:

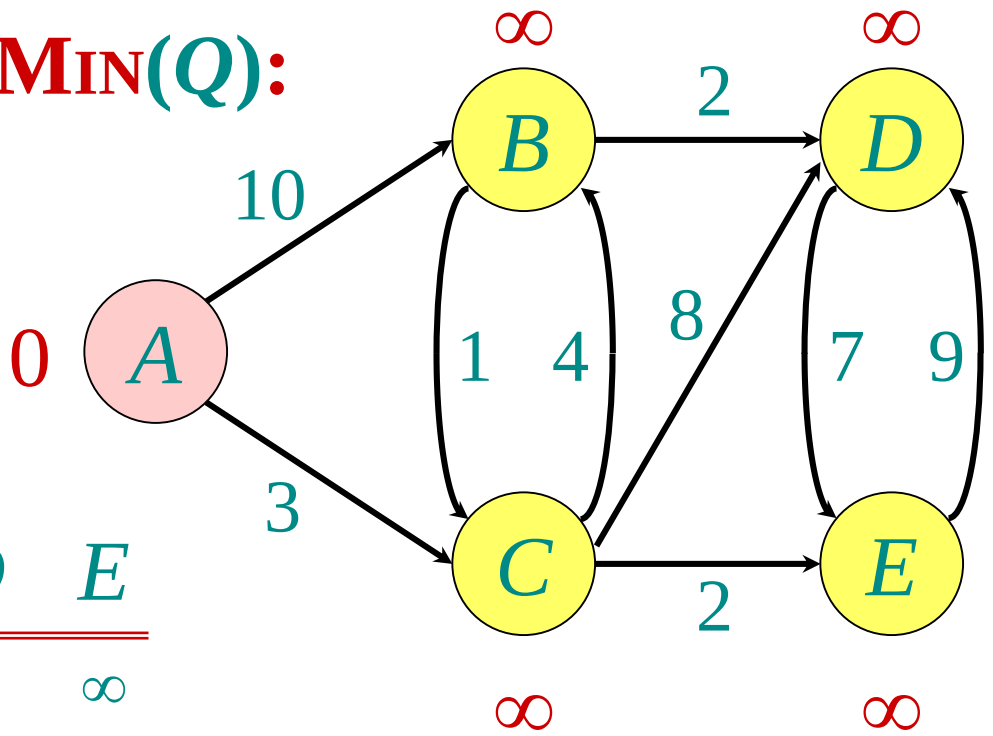


Q:	A	B	C	D	E
	0	∞	∞	∞	∞

S: {}

Demo of Dijkstra's Algorithm

"*A*" ← EXTRACT-MIN(*Q*):

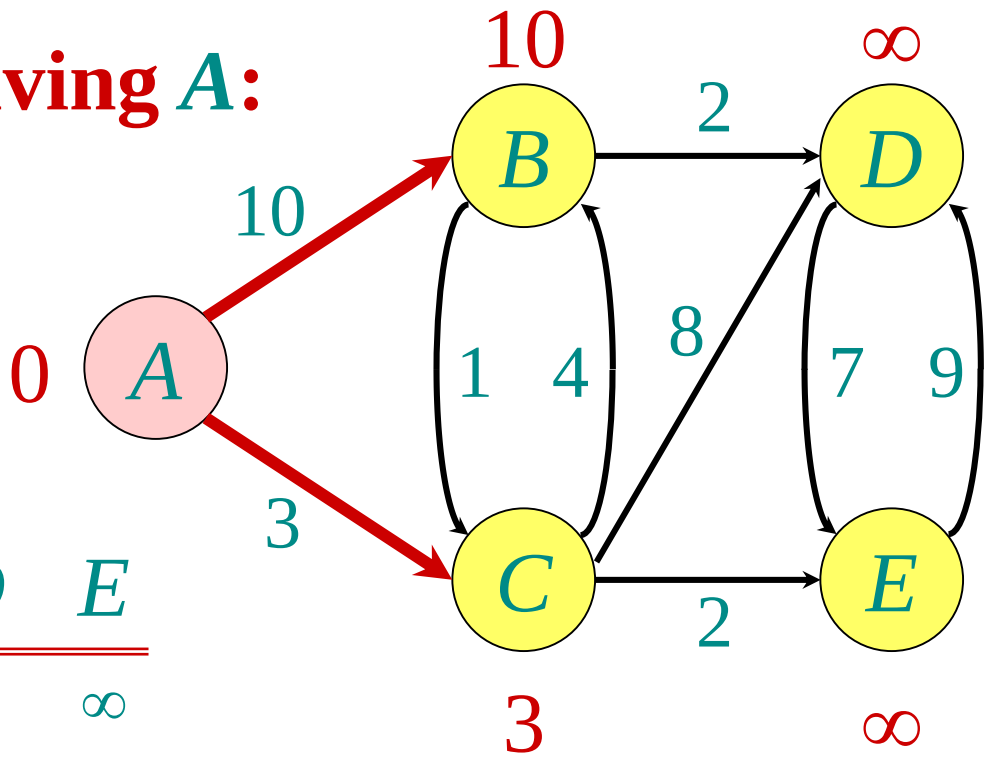


Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞

S: { *A* }

Demo of Dijkstra's Algorithm

Explore edges leaving *A*:

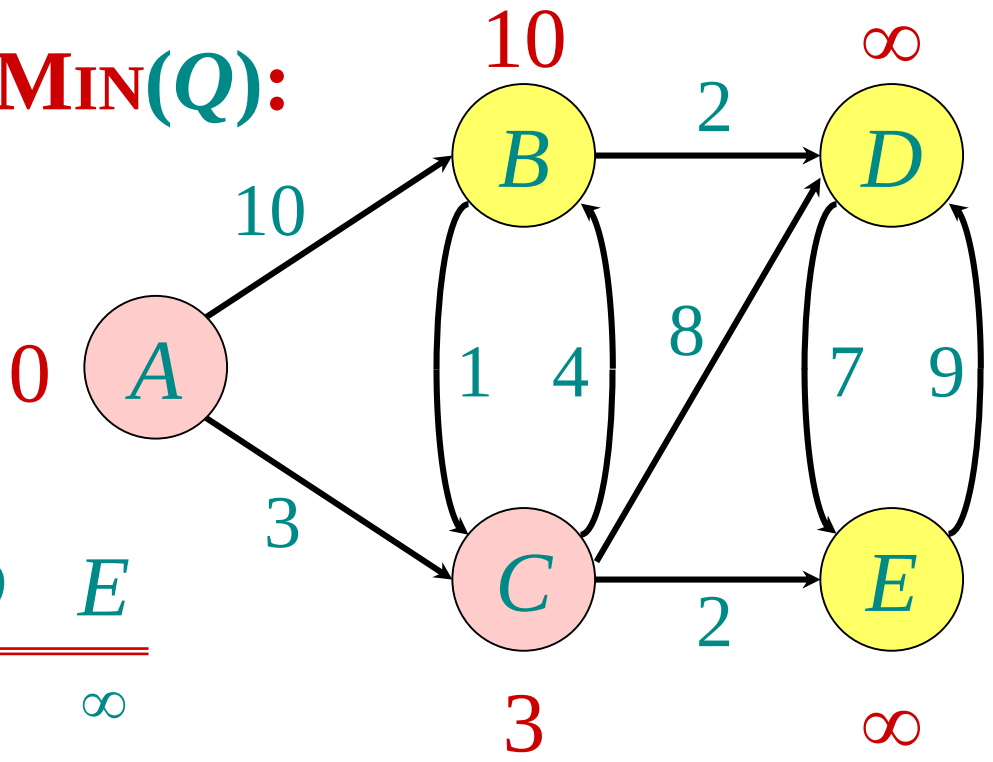


Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { *A* }

Demo of Dijkstra's Algorithm

"*C*" ← EXTRACT-MIN(*Q*):

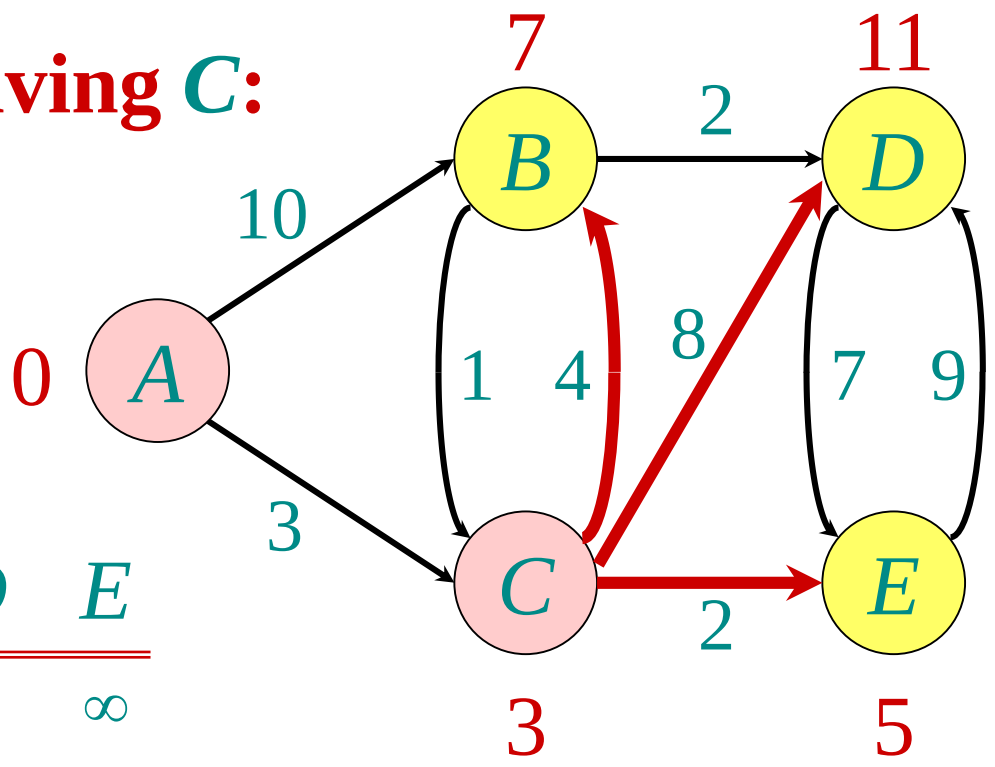


Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { *A*, *C* }

Demo of Dijkstra's Algorithm

Explore edges leaving *C*:

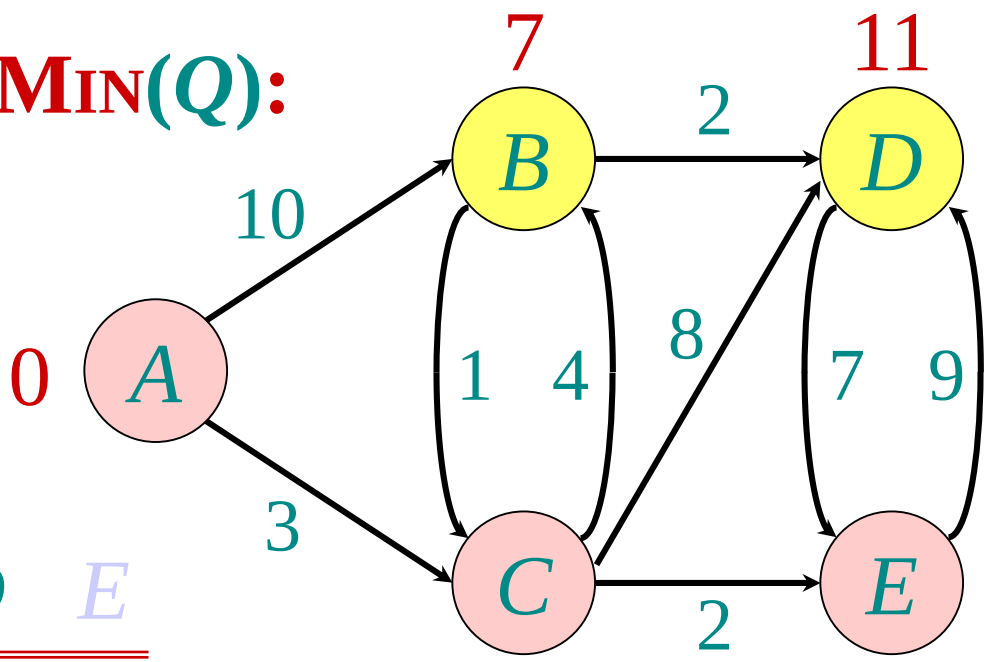


Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { *A*, *C* }

Demo of Dijkstra's Algorithm

"E" ← EXTRACT-MIN(*Q*):

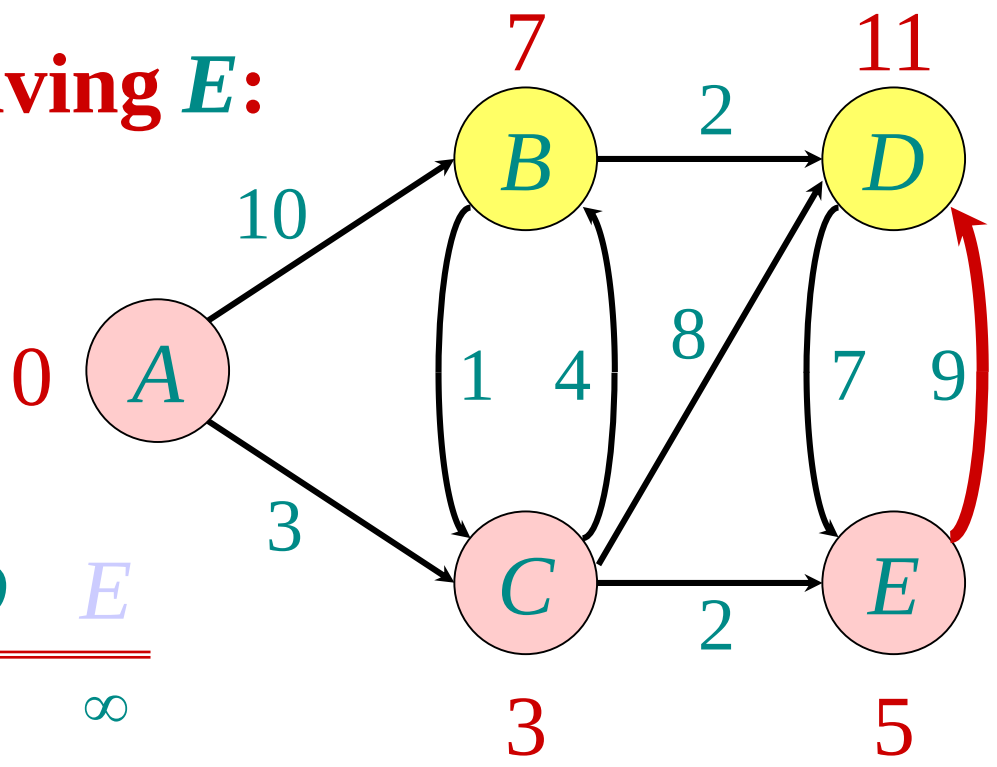


Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

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

Demo of Dijkstra's Algorithm

Explore edges leaving *E*:

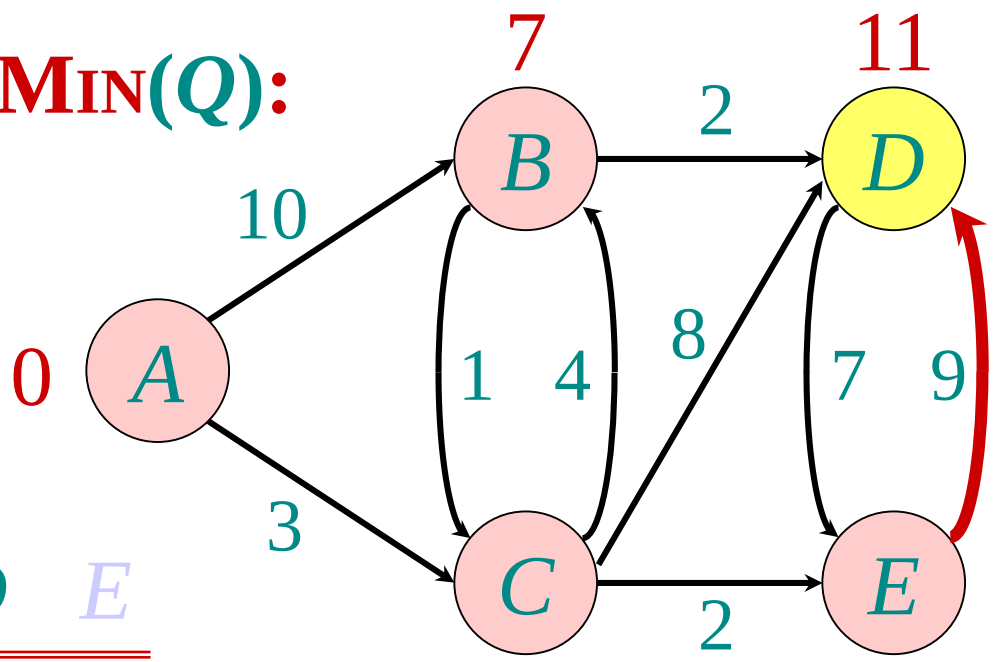


Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

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

Demo of Dijkstra's Algorithm

"B" ← EXTRACT-MIN(*Q*):

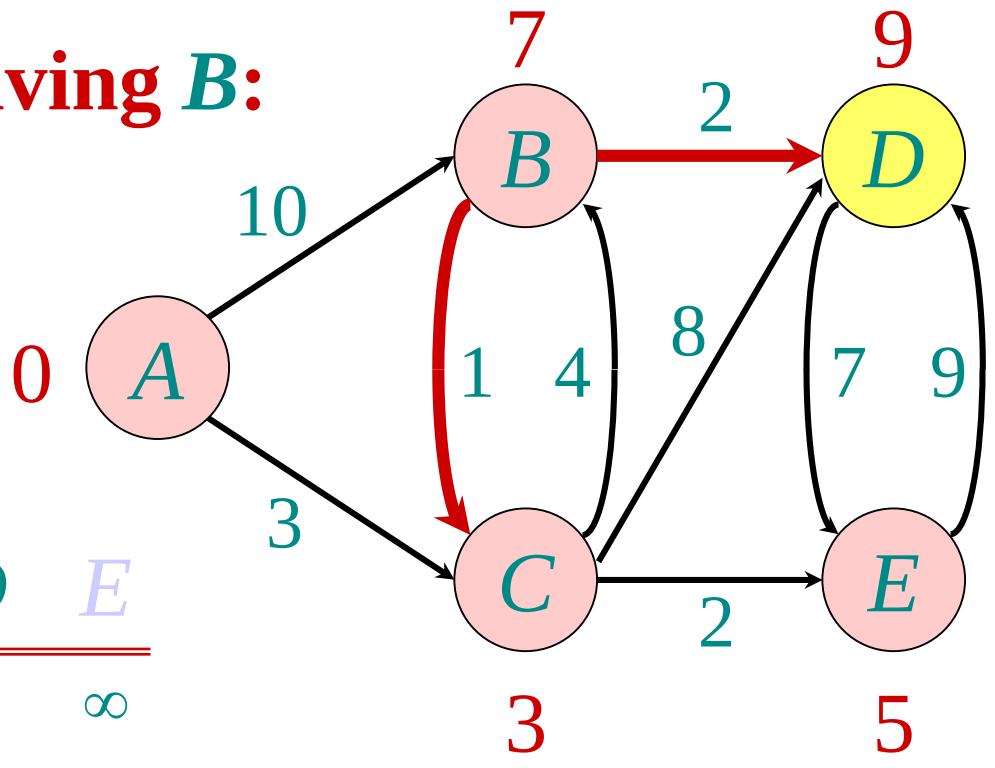


Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

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

Demo of Dijkstra's Algorithm

Explore edges leaving *B*:

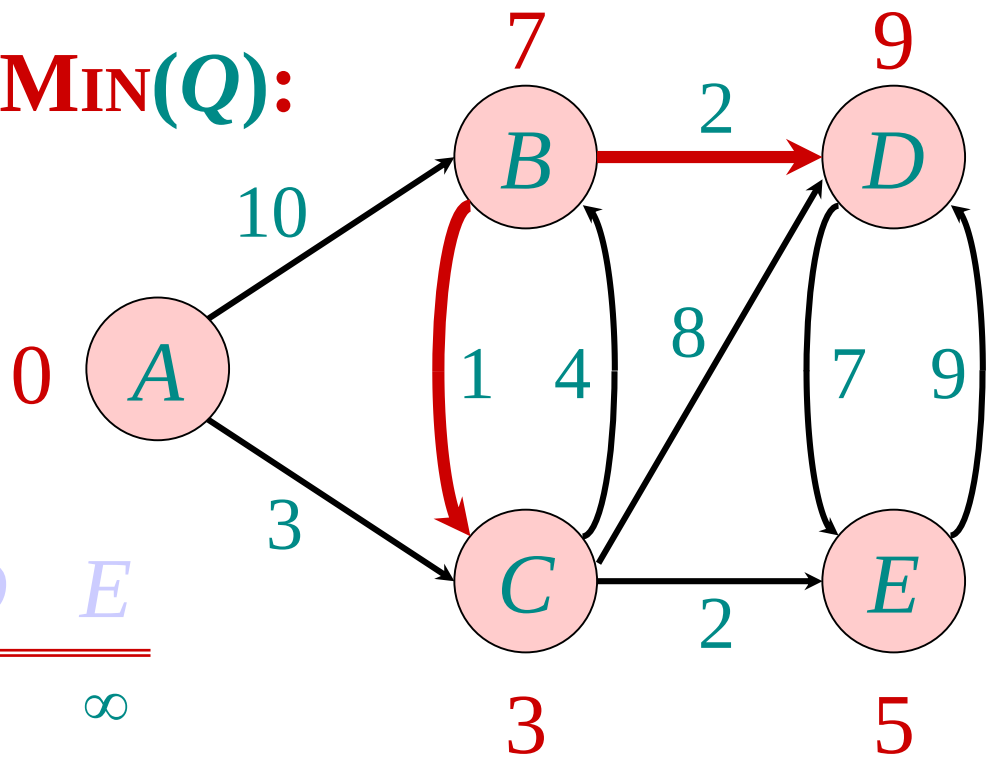


Q:	*A*	*B*	*C*	*D*	*E*
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	

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

Demo of Dijkstra's Algorithm

"D" ← EXTRACT-MIN(*Q*):



Q:

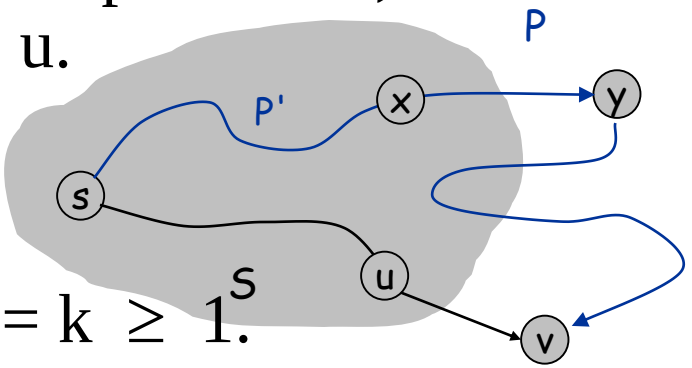
A	*B*	*C*	*D*	*E*
0	∞	∞	∞	∞
	10	3	∞	∞
	7		11	5
	7		11	
			9	

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

New Proof of Correctness

(Greedy Stays Ahead)

- **Invariant.** For each node u that has been processed, d(u) = length of shortest path from s to u.
- **Proof:** (by induction on |S|)
- **Base case:** |S| = 1 is trivial.
- **Inductive hypothesis:** Assume for |S| = k ≥ 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 $d(v)$.
  - Consider any s-v path P.
    - We will show that it's no shorter than $d(v)$.
  - 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 $= d(x) + \ell(x, y) = d(y) \geq d(v)$
  - So P has length at least $d(v)$.

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

Implementation: Priority Queue

- Maintain a set of items with priorities (= “keys”)
  - Example: jobs to be performed
- Operations:
  - Insert
  - Increase key
  - Decrease key
  - Extract-min: find and remove item with least key
- Common data structure: binary heap
  - Time: O(log n) per operation

Pseudocode for Dijkstra(G, ℓ, s)

```
d[s] ← 0
for each v ∈ V − {s}
    do d[v] ← ∞
S ← ∅
Q ← V          ▷ Q is a priority queue maintaining V − S,
                 keyed on d[v]
while Q ≠ ∅
    do u ← EXTRACT-MIN(Q)
       S ← S ∪ {u}
       for each v ∈ Adjacency-list[u]
           do if d[v] > d[u] + ℓ(u, v)
                 then d[v] ← d[u] + ℓ(u, v)
```

explore edges leaving v

Implicit DECREASE-KEY

Analysis of Dijkstra

n times

while $Q \neq \varnothing$
  do $u \leftarrow$ EXTRACT-MIN(Q)
    $S \leftarrow S \cup \{u\}$

$degree(u)$ times

for each $v \in Adj[u]$
  do if $d[v] > d[u] + w(u, v)$
    then $d[v] \leftarrow d[u] + w(u, v)$

$\leq m$ implicit DECREASE-KEY's.

PQ Operation	Dijkstra	Array	Binary heap	d-way Heap	Fib heap †
ExtractMin	n	n	log n	HW3	log n
DecreaseKey	m	1	log n	HW3	1
Total		n^2	m log n	m log $_{m/n}$ n	m + n log n

† Individual ops are amortized bounds

Further reading

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