Lecture 16:
Graphs Shortest Paths

CSE 332: Data Structures & Parallelism

Yafqa Khan

Summer 2025

Announcements

- EX06 due today
- EX07 due next Monday
- Exam 2 information posted here:
  - https://courses.cs.washington.edu/courses/cse332/25su/exams/final.html
  - **Note: it will be hard to accommodate makeups; only four days to grade**
  - If you can't make proposed makeup dates (e.g., sickness/emergency), some options:
  - Option 1: Exam 1 is worth 40% instead of 20% of overall grade
  - Option 2: Take the final exam in the next CSE 332 offering

Today

- Graph Terminologies
    - Paths vs Cycles
    - Connected vs Unconnected
    - Sparse vs dense
- Graph Datastructures
    - Adjacency Matrix
    - Adjacency List
- Graph Traversals
    - DFS (Iterative + Recursive)
    - BFS
- Graph Shortest Paths
    - Dijkstra's

Today

- Graph Traversals
  - DFS (Iterative + Recursive)
  - BFS
- Graph Shortest Paths
  - Dijkstra's

Today

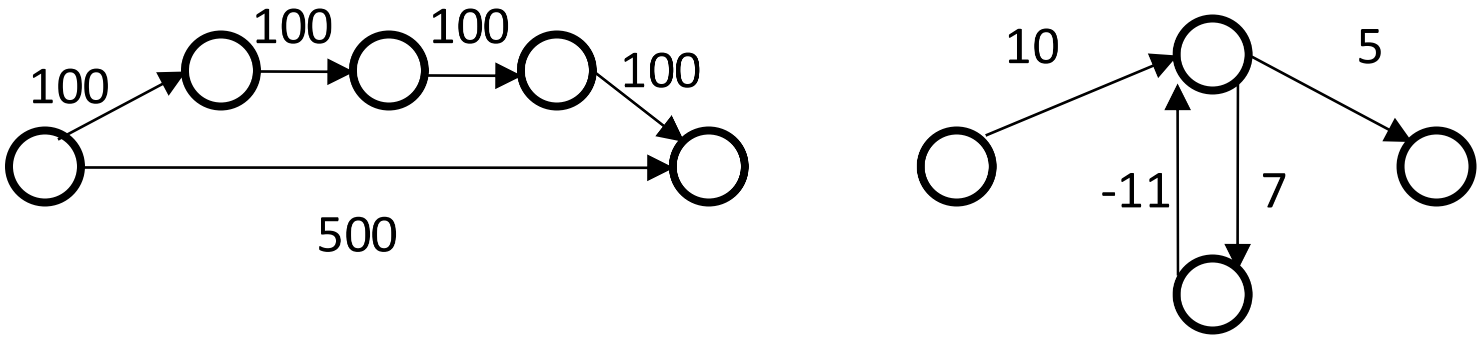
- Graph Traversals
  - DFS (Iterative + Recursive)
  - BFS
- Graph Shortest Paths
  - Dijkstra's

Shortest Path: Applications

- Google Maps
- Network routing
- Driving directions
- Cheap flight tickets
- Critical paths in project management (see textbook)
- etc.

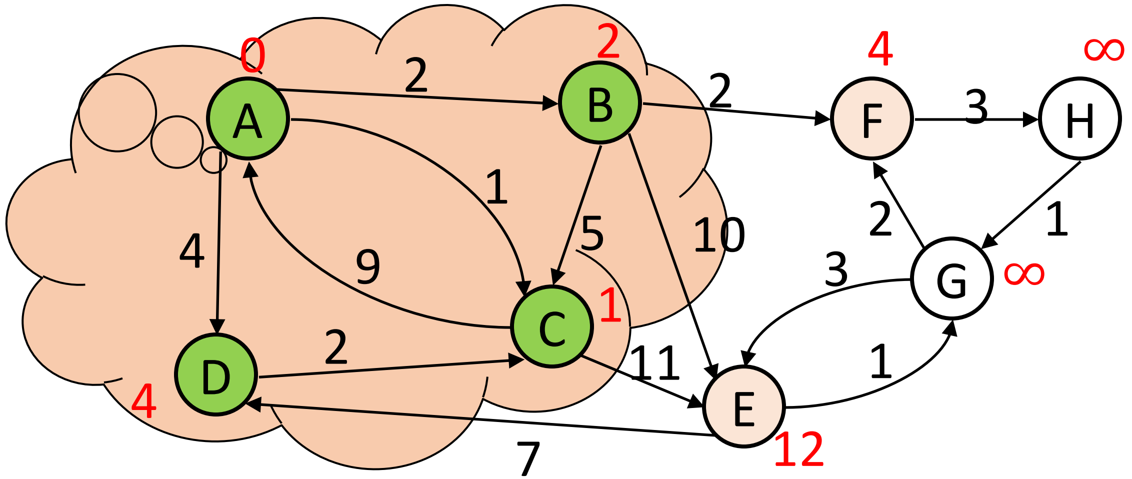
Shortest Path: Weighted Graphs

New Problem: What is the shortest path from `src` to specific nodes in a weighted graph?



- Why BFS won’t work: Shortest path may not have the fewest edges
  - Annoying when this happens with costs of flights
- We will assume there are no negative weights
  - Problem is ill-defined if there are negative-cost cycles
  - Some algorithms are wrong (e.g, Dijkstra's Algorithm) if edges can be negative

Shortest Path: Dijkstra's Algorithm

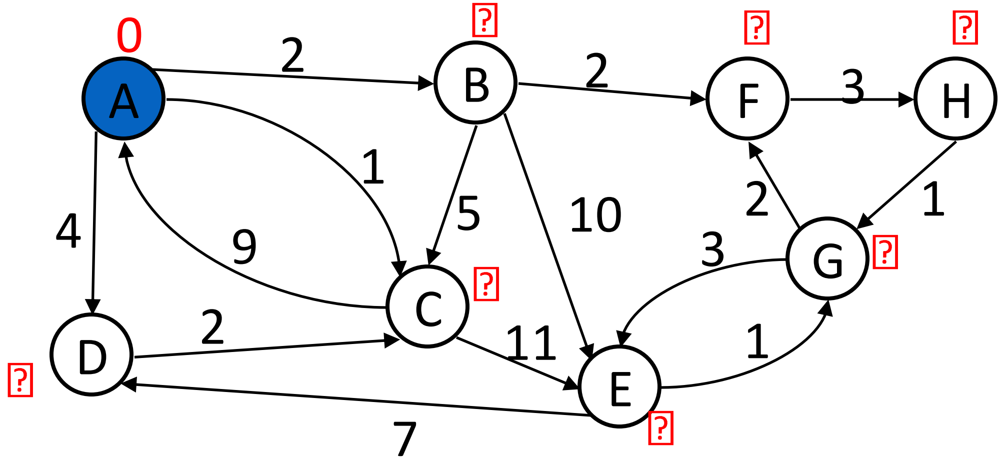


- Initially, start node (A) has cost 0 and marked “visited”
- At each step:
  - Pick cheapest visited vertex v not in the cloud
  - Add v to the "cloud" of known vertices
  - Visit and update distances for nodes with edges from v
- That’s it! (Have to prove it produces correct answers)

Dijkstra's: The Algorithm

```
Dijkstras(Graph G, Node src):
  src.cost = 0 // all other costs uninitialized / implicitly "infinity"
  mark src as visited
  while (there are unknown nodes in G)
      v = unknown, visited node with lowest cost
      mark v as known
      for each edge (v, u) with weight w in G:
            potentialBest = v.cost + w // cost of potential best path
                                          to u (through v)
            if (u is not visited):
                  u.cost = potentialBest
                  u.pred = v
                  mark u as visited
            else if (potentialBest < u.cost):
                  u.cost = potentialBest
                  u.pred = v
```

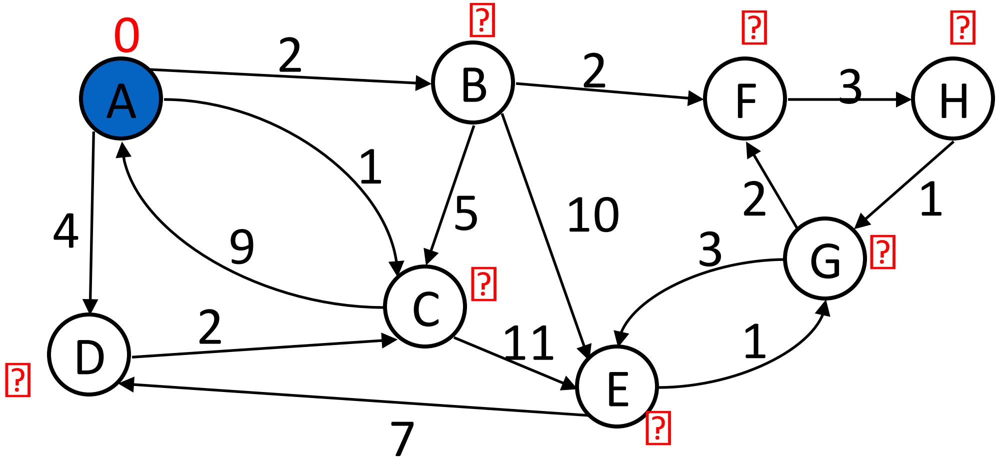
Dijkstra's: Example



Order Added to known Set:

vertex	known?	cost	pred
A			
B			
C			
D			
E			
F			
G			
H			

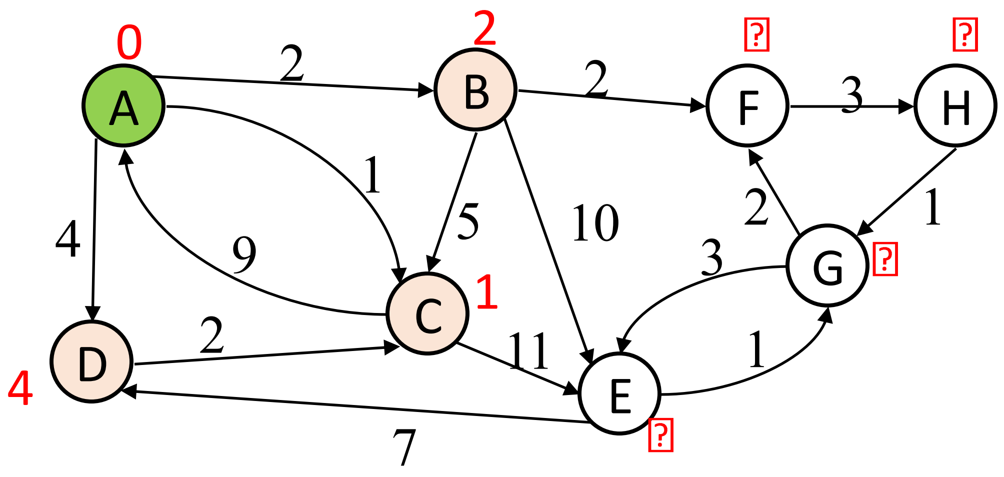
Dijkstra's: Example



Order Added to known Set:

vertex	known?	cost	pred
A		0	
B		∞	
C		∞	
D		∞	
E		∞	
F		∞	
G		∞	
H		∞	

Dijkstra's: Example

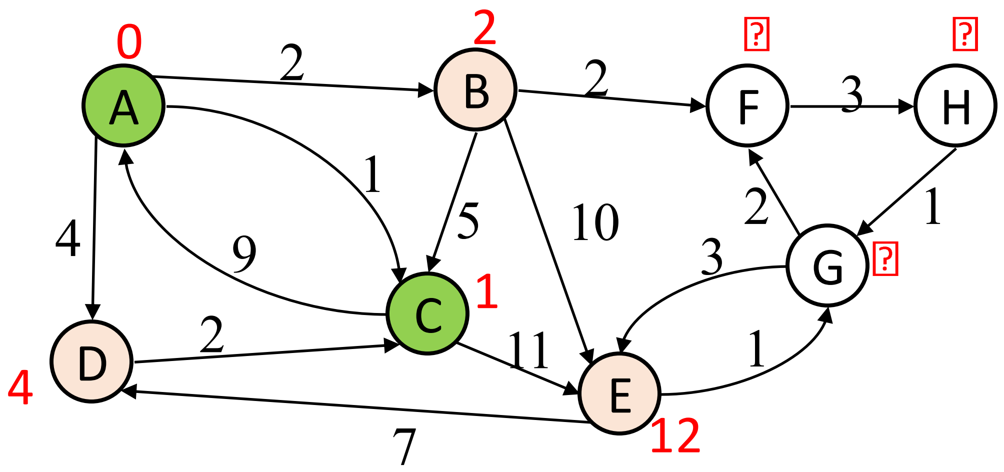


<u>Order Added to known Set:</u>

A

vertex	known?	cost	pred
A	Yes	0	
B		2	A
C		1	A
D		4	A
E		∞	
F		∞	
G		∞	
H		∞	

Dijkstra's: Example

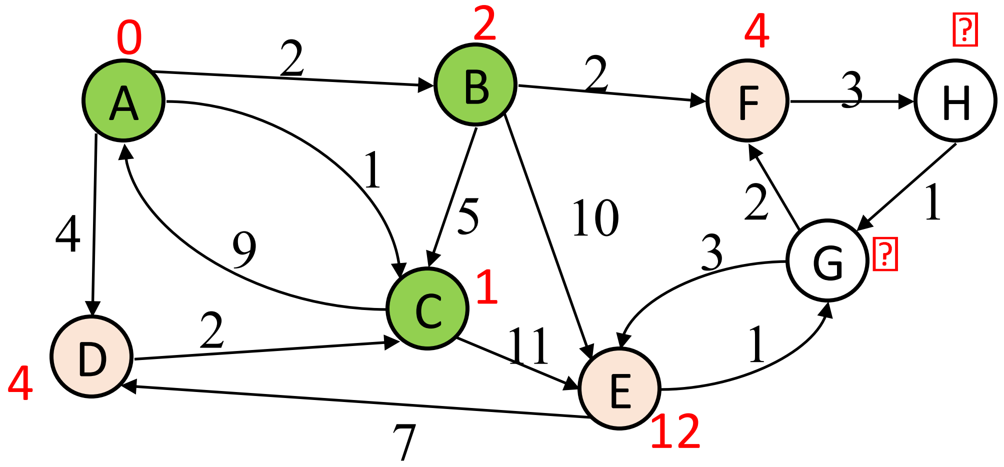


Order Added to known Set:

A C

vertex	known?	cost	pred
A	Yes	0	
B		2	A
C	Yes	1	A
D		4	A
E		12	C
F		∞	
G		∞	
H		∞	

Dijkstra's: Example

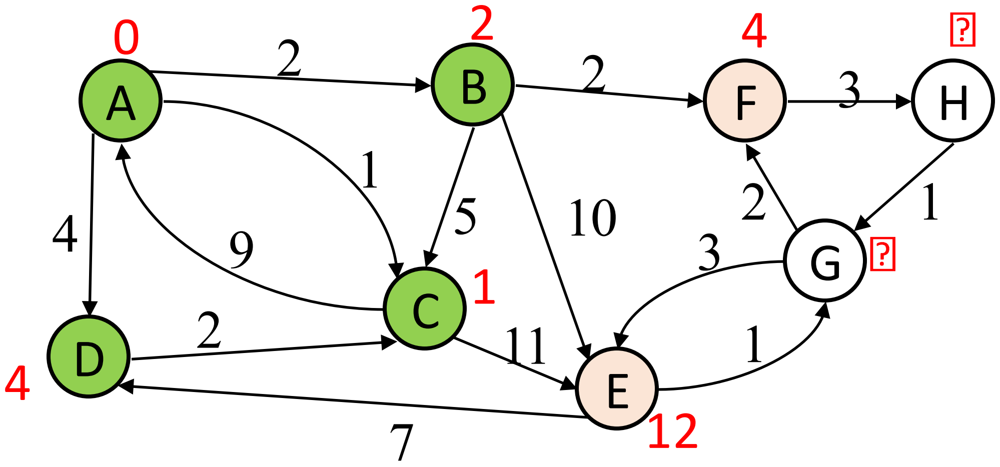


Order Added to known Set:

A C B

vertex	known?	cost	pred
A	Yes	0	
B	Yes	2	A
C	Yes	1	A
D		4	A
E		12	C
F		4	B
G		∞	
H		∞	

Dijkstra's: Example

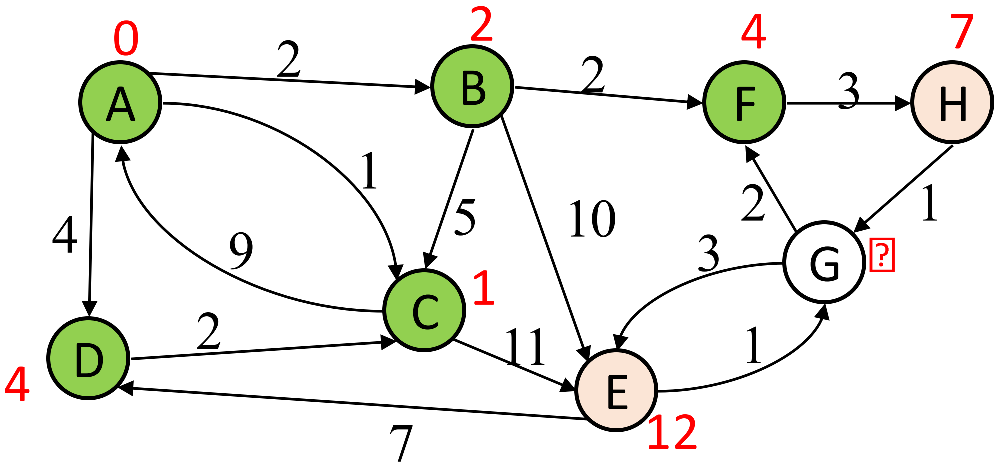


Order Added to known Set:

A C B D

vertex	known?	cost	pred
A	Yes	0	
B	Yes	2	A
C	Yes	1	A
D	Yes	4	A
E		12	C
F		4	B
G		∞	
H		∞	

Dijkstra's: Example

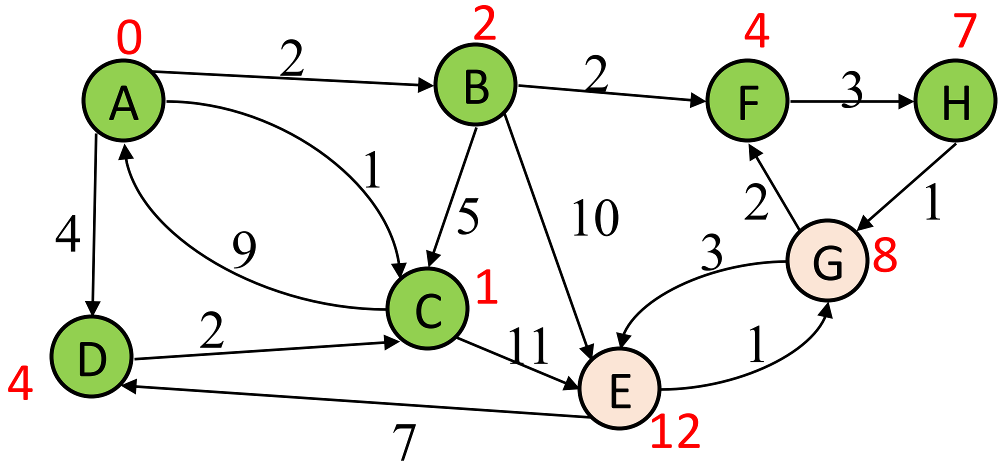


Order Added to known Set:

A C B D F

vertex	known?	cost	pred
A	Yes	0	
B	Yes	2	A
C	Yes	1	A
D	Yes	4	A
E		12	C
F	Yes	4	B
G		∞	
H		7	F

Dijkstra's: Example

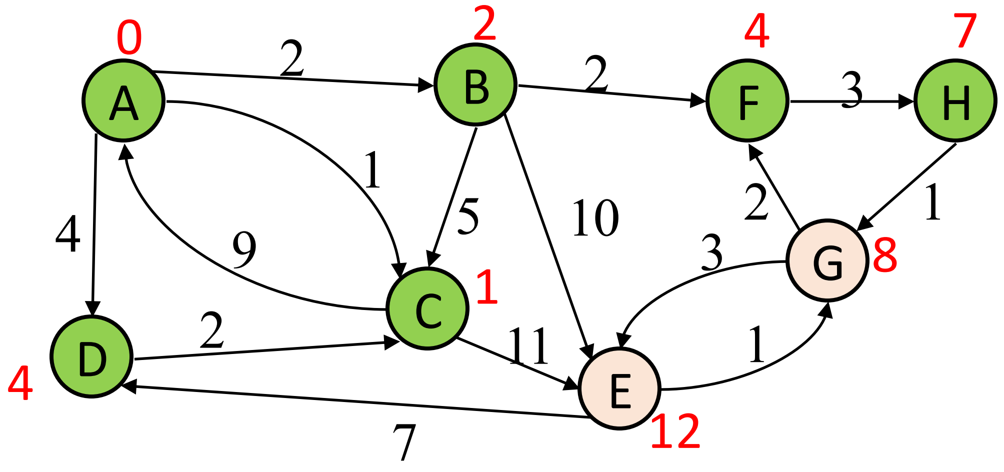


Order Added to known Set:

A C B D F H

vertex	known?	cost	pred
A	Yes	0	
B	Yes	2	A
C	Yes	1	A
D	Yes	4	A
E		12	C
F	Yes	4	B
G		8	H
H	Yes	7	F

Dijkstra's: Example

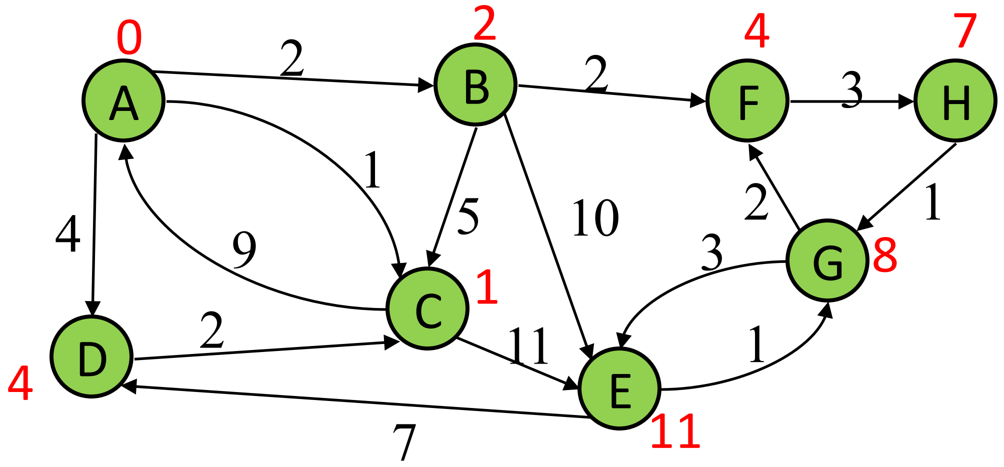


Order Added to known Set:

A C B D F H G

vertex	known?	cost	pred
A	Yes	0	
B	Yes	2	A
C	Yes	1	A
D	Yes	4	A
E		~~12~~ 11	~~C~~ G
F	Yes	4	B
G	Yes	8	H
H	Yes	7	F

Dijkstra's: Example



Order Added to known Set:

A C B D F H G E

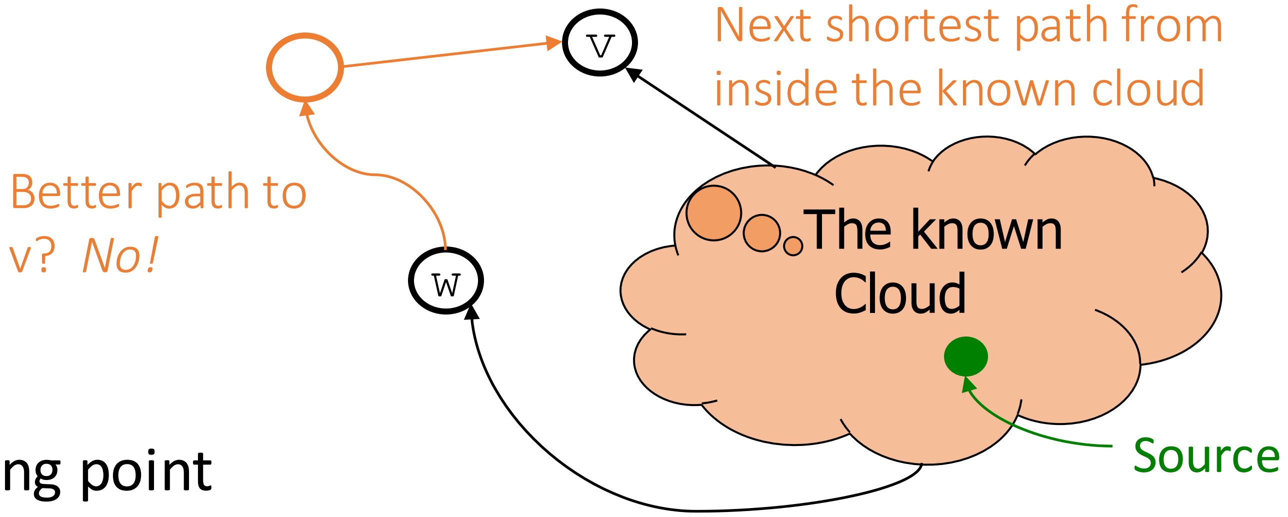
vertex	known?	cost	pred
A	Yes	0	
B	Yes	2	A
C	Yes	1	A
D	Yes	4	A
E	Yes	~~12~~ 11	~~C~~ G
F	Yes	4	B
G	Yes	8	H
H	Yes	7	F

Dijkstra's: A Greedy Algorithm

- Dijkstra’s algorithm
  - For single-source shortest paths in a weighted graph (directed or undirected) with no negative-weight edges

- An example of a greedy algorithm:
  - At each step, irrevocably does what seems best at that step
    - A locally optimal step, not necessarily globally optimal
  - Once a vertex is known, it is not revisited
    - Turns out to be globally optimal

Dijkstra's: Correctness



1. Greedy Approach
   - Prioritizes nodes closer to the starting point
2. Optimality of Selected Nodes
   - When Dijkstra’s marks a node as known, it has seen all possible paths to that node based on the visited nodes. Since it picks the smallest current cost, it is optimal.
3. Convergence
   - Dijkstra's explores every nodes, so it never accidently picks e.g., the 2nd best path

Dijkstra's: Unoptimal Efficiency

```
Dijkstras(Graph G, Node src):
   src.cost = 0 // all other costs implicitly "infinity"
   mark src as visited
   while (there are unknown nodes in G)
        v = unknown, visited node with lowest cost
        mark v as known
        for each edge (v, u) with weight w in G:
                potentialBest = v.cost + w // cost of potential best path
                                              to u (through v)
                if (u is not visited):
                        u.cost = potentialBest
                        u.pred = v
                        mark u as visited
                else if (potentialBest < u.cost):
                        u.cost = potentialBest
                        u.pred = v
```

$\mathcal{O}(1)$

$\mathcal{O}(|V|^2)$

$\mathcal{O}(|E|)$

$\mathcal{O}(|V|^2 + |E|)$

Dijkstra's: Optimal Efficiency

```
Dijkstras(Graph G, Node src):
   src.cost = 0 // all other costs implicitly "infinity"
   mark src as visited
   heap = {src}
   while (heap is not empty)
        v = heap.deleteMin()
        mark v as known
        for each edge (v, u) with weight w in G:
               potentialBest = v.cost + w // cost of potential best path
                                          to u (through v)
               if (u is not visited):
                       u.cost = potentialBest
                       u.pred = v
                       mark u as visited
                       heap.insert(u)
               else if (potentialBest < u.cost):
                       u.cost = potentialBest
                       u.pred = v
                       heap.changePriority(u, potentialBest)
```

$\mathcal{O}(1)$

$\mathcal{O}(|V|\log(|V|))$

$\mathcal{O}(|E|\log(|V|))$

$\mathcal{O}(|V|\log|V| + |E|\log|V|)$

Heap: Other operations

- `decreaseKey(idx, Δ)` or `increaseKey(idx, Δ)`
  1. `arr[idx] -= Δ` or `arr[idx] += Δ`
  2. `percolateUp()` or `percolateDown()`

  Worst Case $\Theta(\log n)$

- `delete(idx)`
  1. `decreaseKey(idx, ∞)`
  2. `deleteMin()`

  Worst Case $\Theta(\log n)$

Heap: Note on `decrease/increaseKey`

- MORE COMMONLY CALLED `changePriority(key, prio)`
  1. Uses a map to go from `key` -> `idx`
  2. `arr[idx] = prio`
  3. `percolateUp()` or `percolateDown()`

  (Same as `decrease/increaseKey`)

Any Questions?