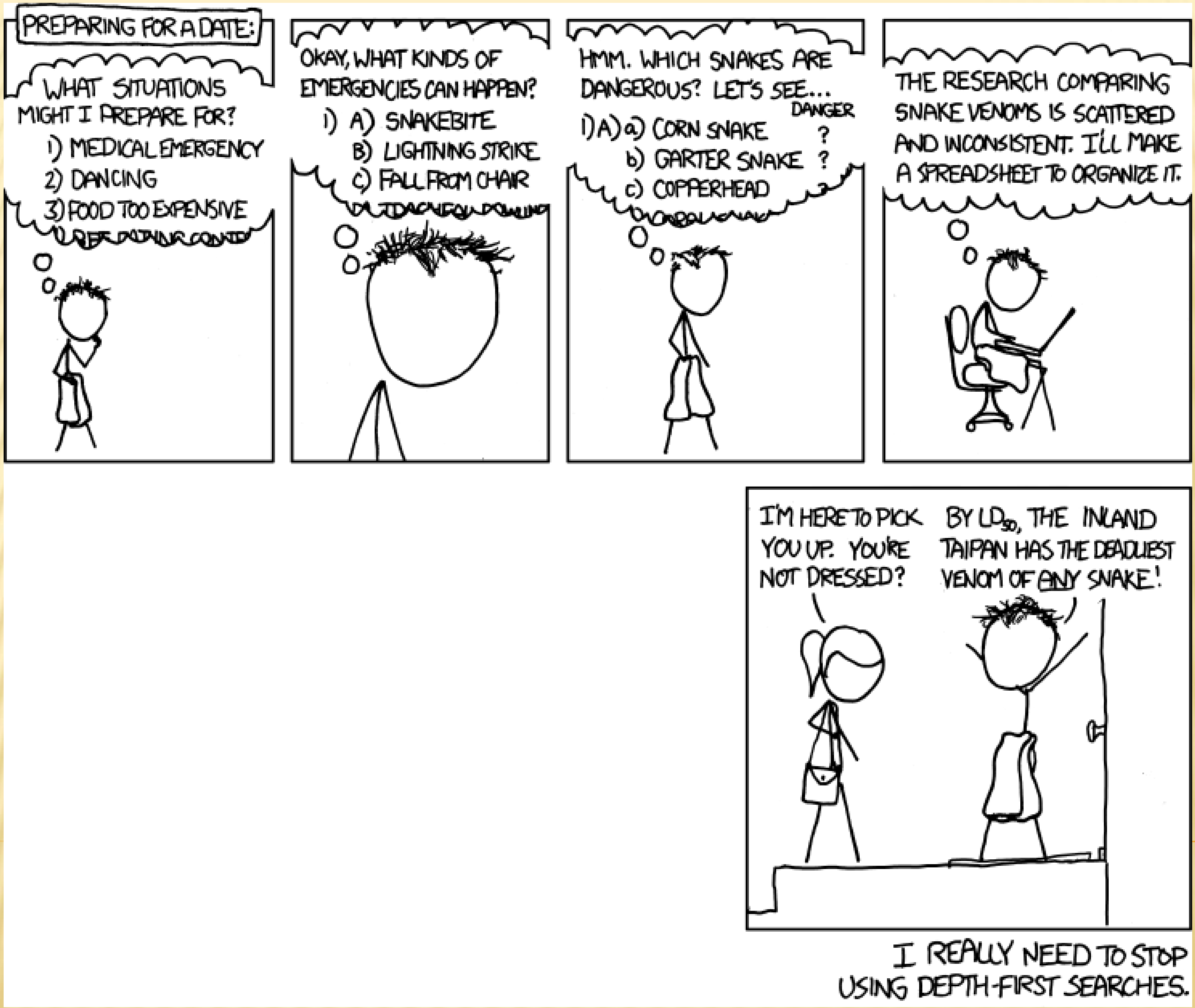

PREPARING FOR A DATE:
WHAT SITUATIONS MIGHT I PREPARE FOR? 1) MEDICAL EMERGENCY 2) DANCING 3) FOOD TOO EXPENSIVE
OKAY, WHAT KINDS OF EMERGENCIES CAN HAPPEN? 1) A) SNAKEBITE B) LIGHTNING STRIKE C) FALL FROM CHAIR
HMM. WHICH SNAKES ARE DANGEROUS? LET'S SEE... DANGER 1) A) a) CORN SNAKE ? b) GARTER SNAKE ? c) COPPERHEAD ?
THE RESEARCH COMPARING SNAKE VENOMS IS SCATTERED AND INCONSISTENT. I'LL MAKE A SPREADSHEET TO ORGANIZE IT.
I'M HERE TO PICK YOU UP. YOU'RE NOT DRESSED?
BY LD50, THE INLAND TAIPAN HAS THE DEADLIEST VENOM OF ANY SNAKE!
I REALLY NEED TO STOP USING DEPTH-FIRST SEARCHES.

Section 7: Dijkstra's

Slides adapted from Alex Mariakakis

with material Kellen Donohue, David Mailhot, and Dan Grossman

THINGS TO DISCUSS

- Late days
  - 3 assignments left
  - Can use 2 late days max per assignment

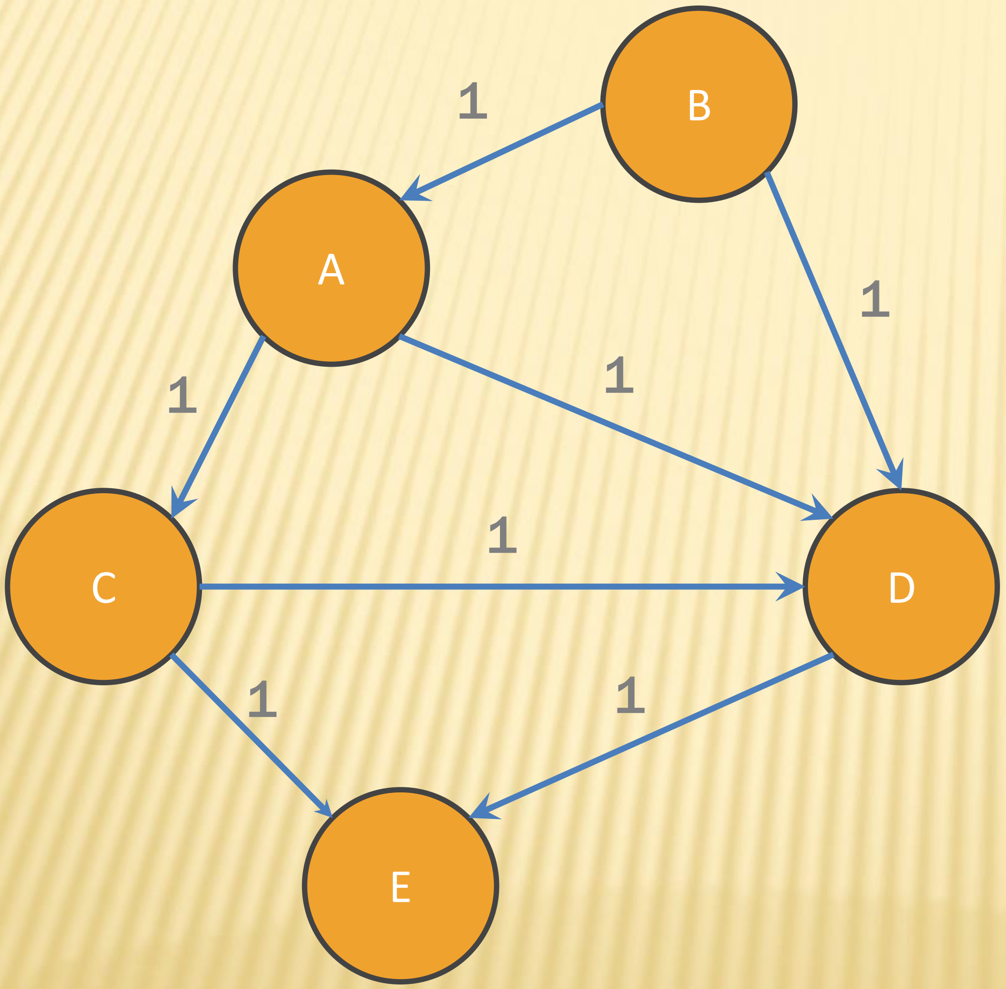
HOMEWORK 7

- Modify your graph to use generics
  - Will have to update HW #5 and HW #6 tests
- Implement Dijkstra's algorithm
  - Search algorithm that accounts for edge weights
  - Note: This should not change your implementation of Graph. Dijkstra's is performed <u>on</u> a Graph, not <u>within</u> a Graph.

HOMEWORK 7

- The more well-connected two characters are, the lower the weight and the more likely that a path is taken through them
  - The weight of an edge is equal to the inverse of how many comic books the two characters share
  - Ex: If Amazing Amoeba and Zany Zebra appeared in 5 comic books together, the weight of their edge would be 1/5

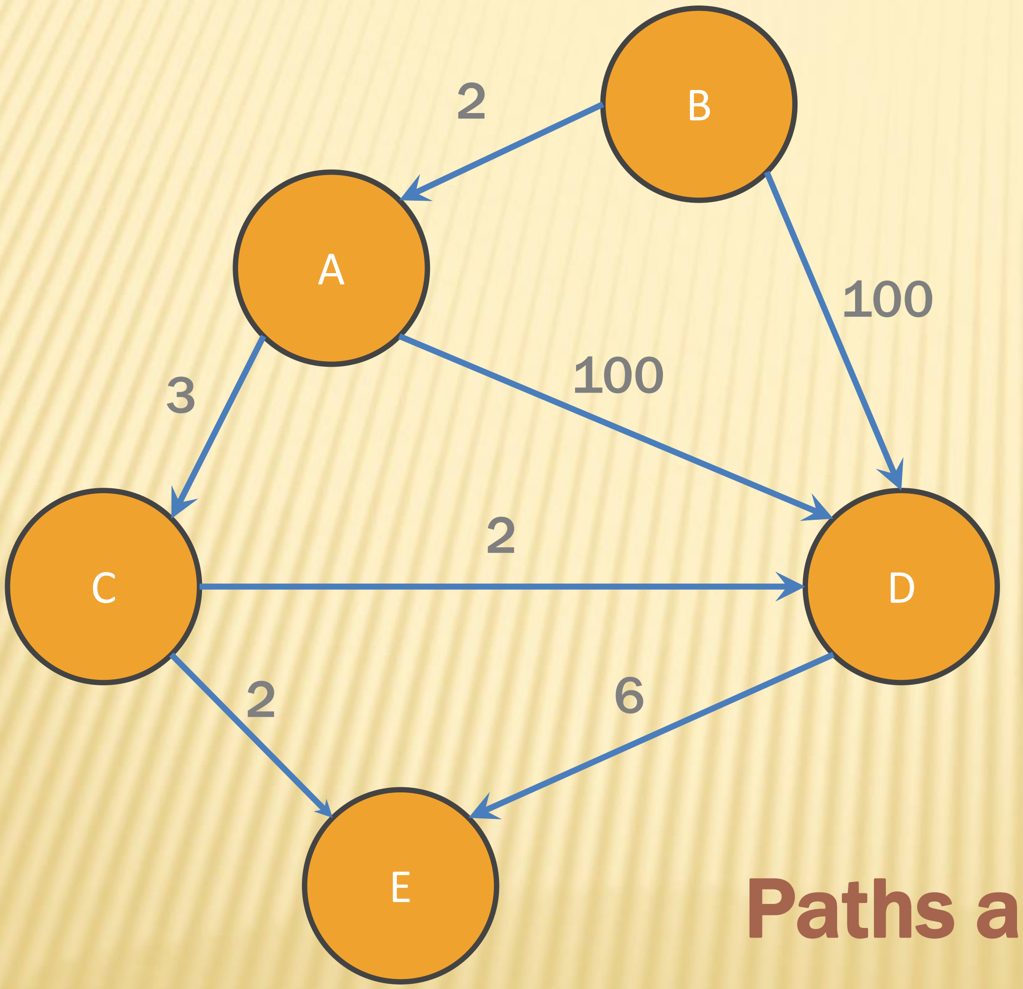
REVIEW: SHORTEST PATHS WITH BFS



From Node B

Destination	Path	Cost
A	<B,A>	1
B	<B>	0
C	<B,A,C>	2
D	<B,D>	1
E	<B,D,E>	2

SHORTEST PATHS WITH WEIGHTS

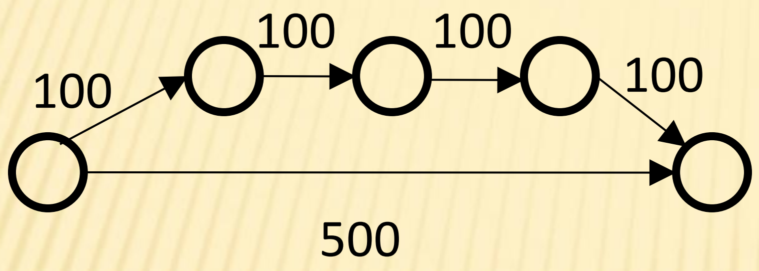


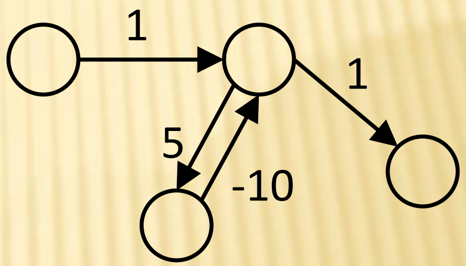
From Node B

Destination	Path	Cost
A	<B,A>	2
B	<B>	0
C	<B,A,C>	5
D	<B,A,C,D>	7
E	<B,A,C,E>	7

Paths are not the same!

BFS VS. DIJKSTRA'S





- BFS doesn't work because path with minimal cost ≠ path with fewest edges
- Dijkstra's works if the weights are non-negative
- What happens if there is a negative edge?
  - Minimize cost by repeating the cycle forever

DIJKSTRA'S ALGORITHM

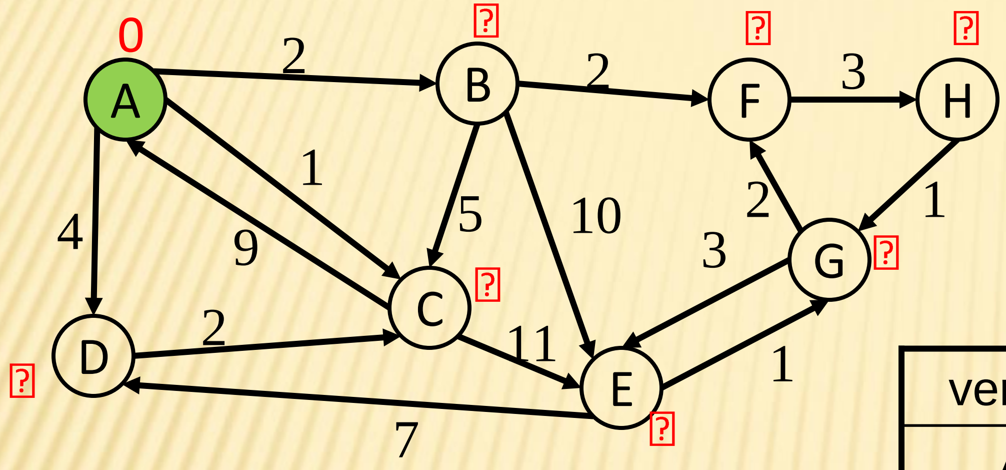
- Named after its inventor Edsger Dijkstra (1930-2002)
  - Truly one of the "founders" of computer science; this is just one of his many contributions
- The idea: reminiscent of BFS, but adapted to handle weights
  - Grow the set of nodes whose shortest distance has been computed
  - Nodes not in the set will have a "best distance so far"
  - A priority queue will turn out to be useful for efficiency

DIJKSTRA'S ALGORITHM

1. For each node `v`, set `v.cost = ∞` and `v.known = false`
2. Set `source.cost = 0`
3. While there are unknown nodes in the graph
   a) Select the unknown node `v` with lowest cost
   b) Mark `v` as known
   c) For each edge `(v,u)` with weight `w`,

```
c1 = v.cost + w          // cost of best path through v to u
c2 = u.cost              // cost of best path to u previously known
if(c1 < c2)              // if the new path through v is better, update
    u.cost = c1
    u.path = v
```

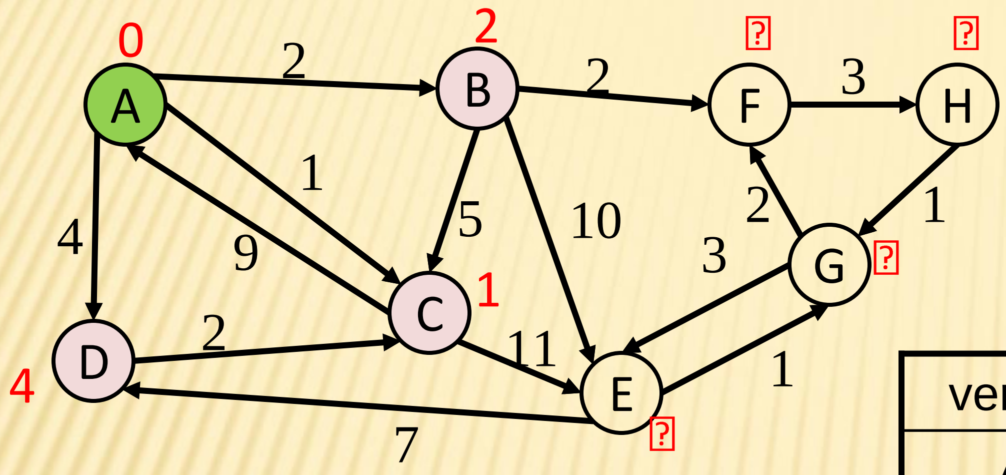
Example #1



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

Order Added to Known Set:

EXAMPLE #1

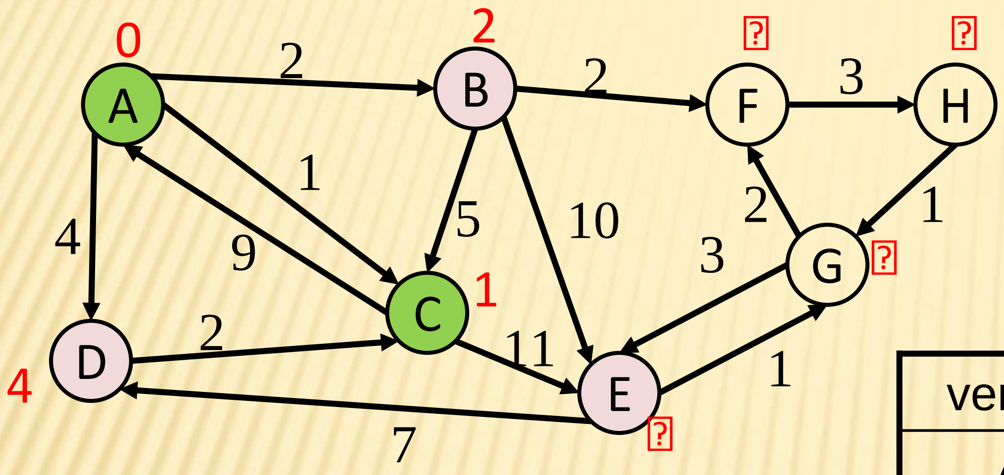


vertex	known?	cost	path
A	Y	0	
B		≤ 2	A
C		≤ 1	A
D		≤ 4	A
E		∞	
F		∞	
G		∞	
H		∞	

Order Added to Known Set:

A

EXAMPLE #1

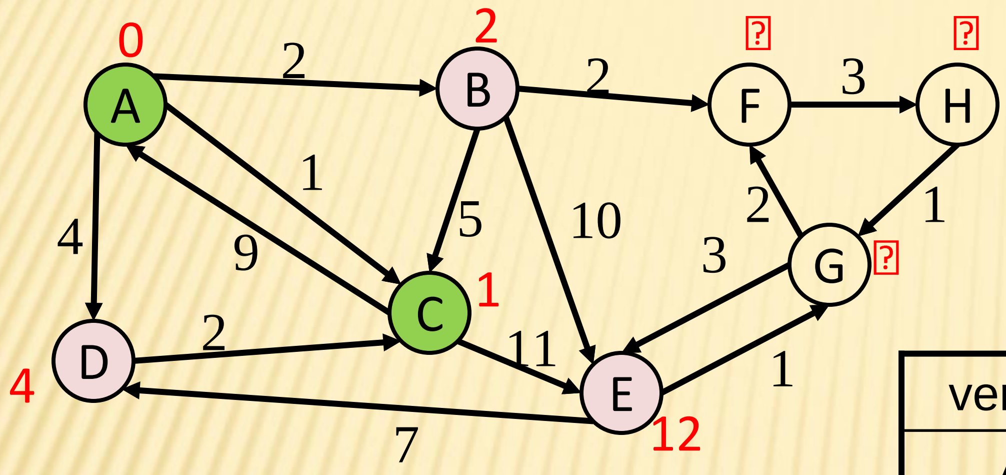


vertex	known?	cost	path
A	Y	0	
B		≤ 2	A
C	Y	1	A
D		≤ 4	A
E		∞	
F		∞	
G		∞	
H		∞	

Order Added to Known Set:

A, C

EXAMPLE #1

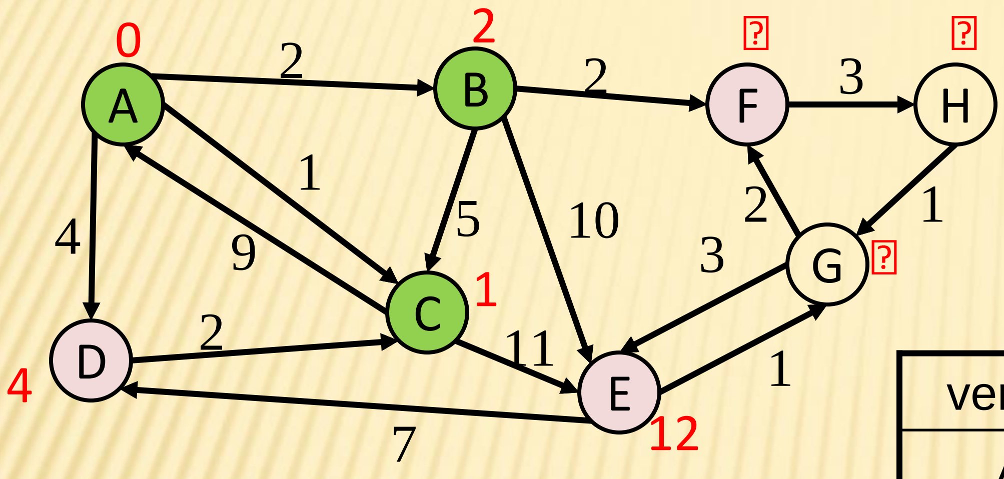


vertex	known?	cost	path
A	Y	0	
B		≤ 2	A
C	Y	1	A
D		≤ 4	A
E		≤ 12	C
F		∞	
G		∞	
H		∞	

Order Added to Known Set:

A, C

Example #1

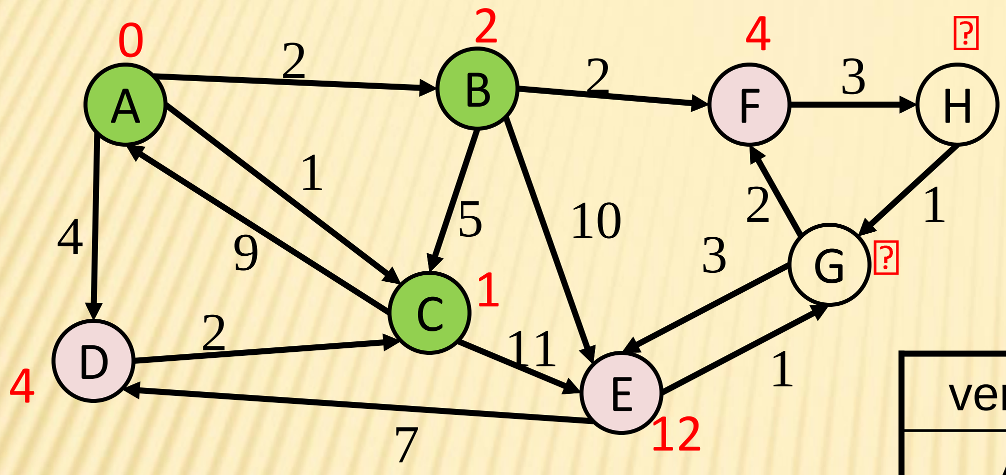


vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D		≤ 4	A
E		≤ 12	C
F		∞	
G		∞	
H		∞	

Order Added to Known Set:

A, C, B

EXAMPLE #1

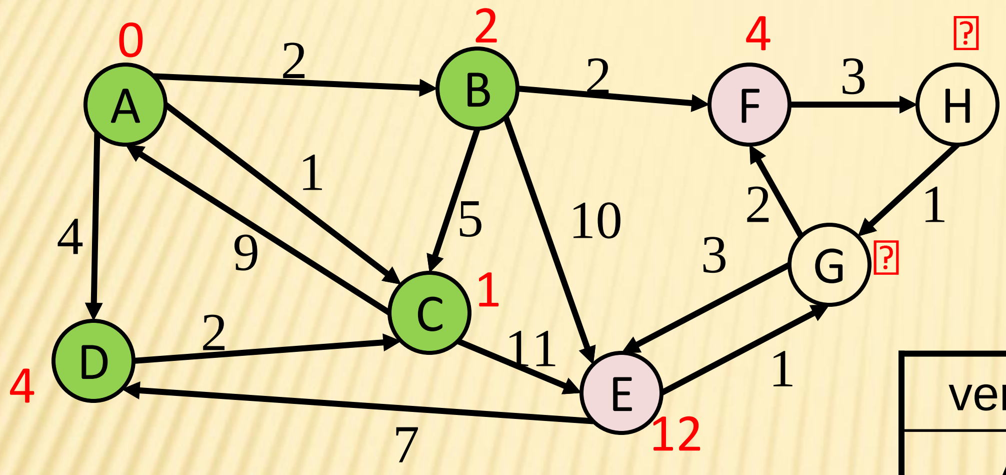


vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D		≤ 4	A
E		≤ 12	C
F		≤ 4	B
G		∞	
H		∞	

Order Added to Known Set:

A, C, B

EXAMPLE #1

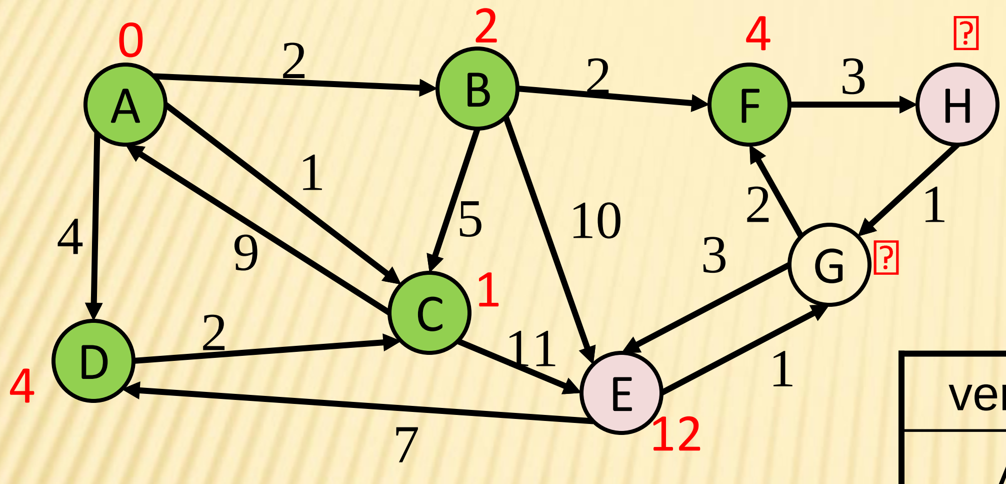


vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 12	C
F		≤ 4	B
G		∞	
H		∞	

Order Added to Known Set:

A, C, B, D

EXAMPLE #1

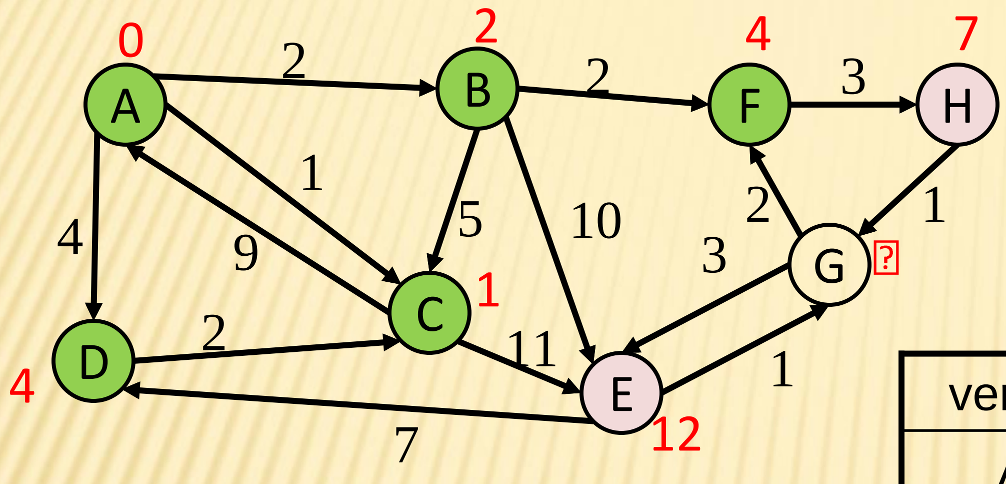


vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 12	C
F	Y	4	B
G		∞	
H		∞	

Order Added to Known Set:

A, C, B, D, F

EXAMPLE #1

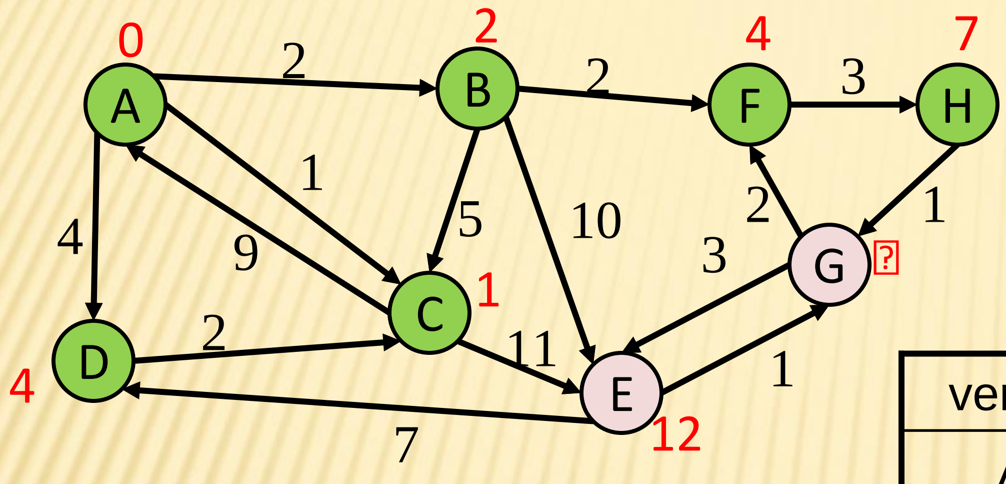


vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 12	C
F	Y	4	B
G		∞	
H		≤ 7	F

Order Added to Known Set:

A, C, B, D, F

Example #1

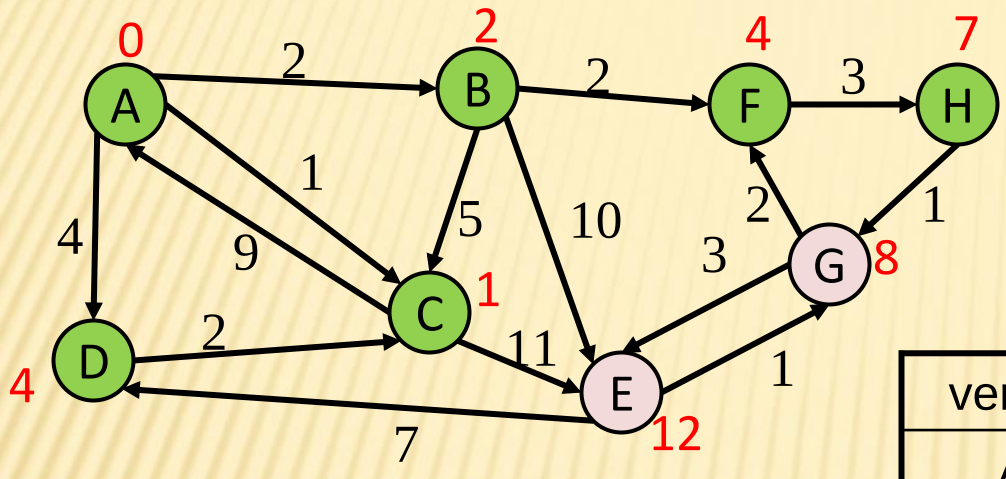


vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 12	C
F	Y	4	B
G		∞	
H	Y	7	F

Order Added to Known Set:

A, C, B, D, F, H

Example #1

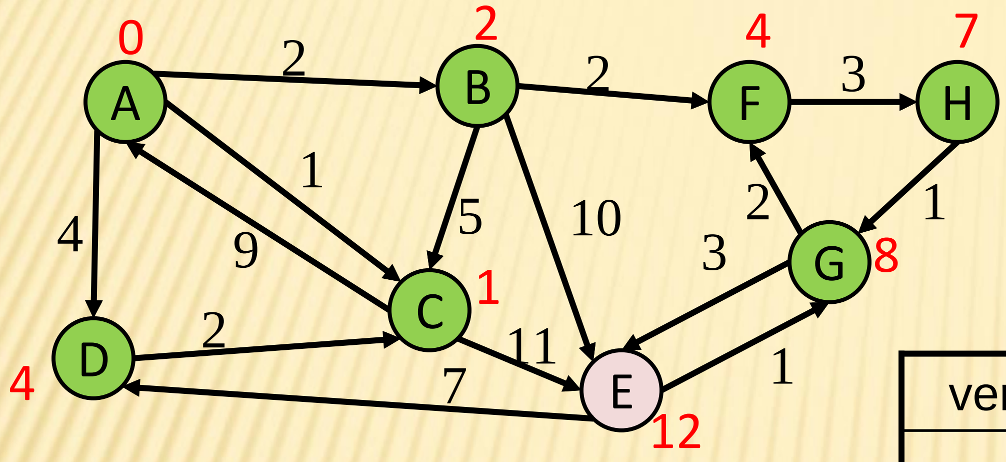


vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 12	C
F	Y	4	B
G		≤ 8	H
H	Y	7	F

Order Added to Known Set:

A, C, B, D, F, H

EXAMPLE #1

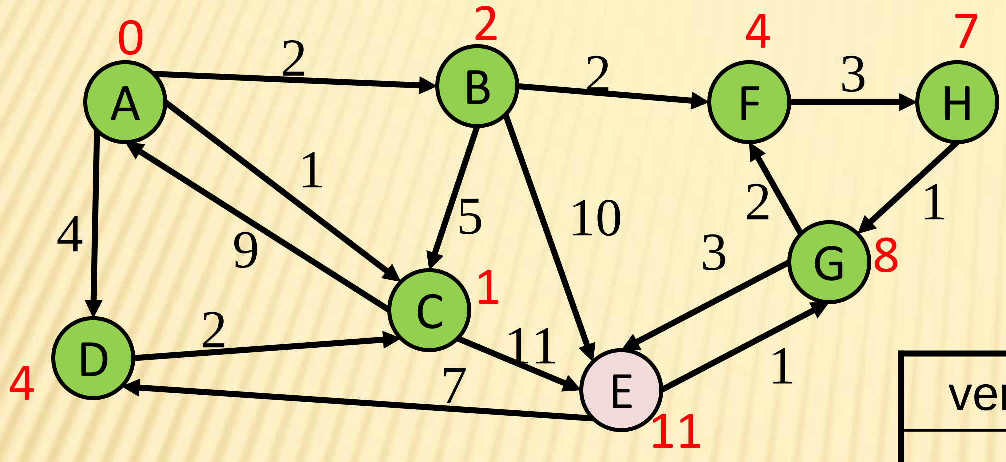


vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 12	C
F	Y	4	B
G	Y	8	H
H	Y	7	F

Order Added to Known Set:

A, C, B, D, F, H, G

EXAMPLE #1

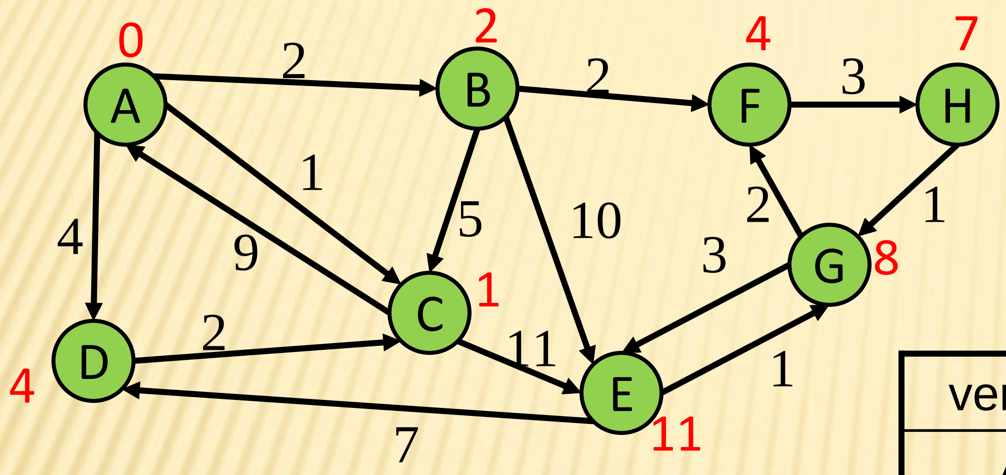


vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 11	G
F	Y	4	B
G	Y	8	H
H	Y	7	F

Order Added to Known Set:

A, C, B, D, F, H, G

Example #1

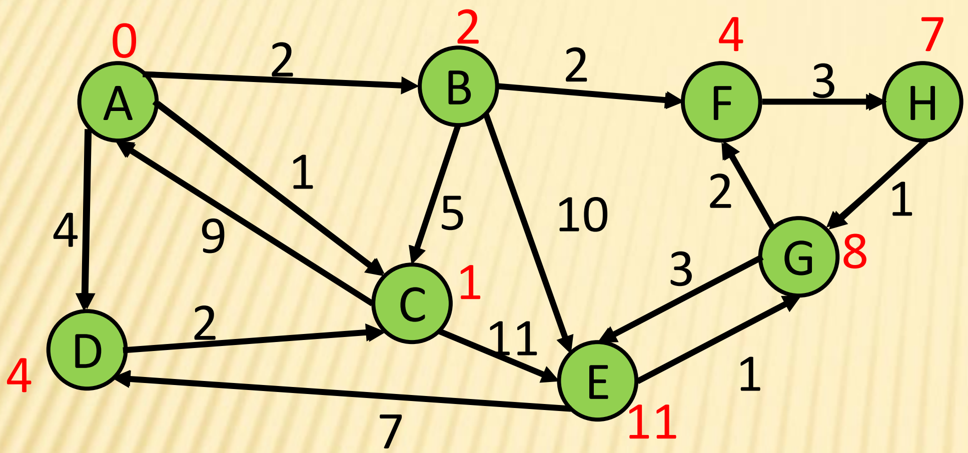


Order Added to Known Set:

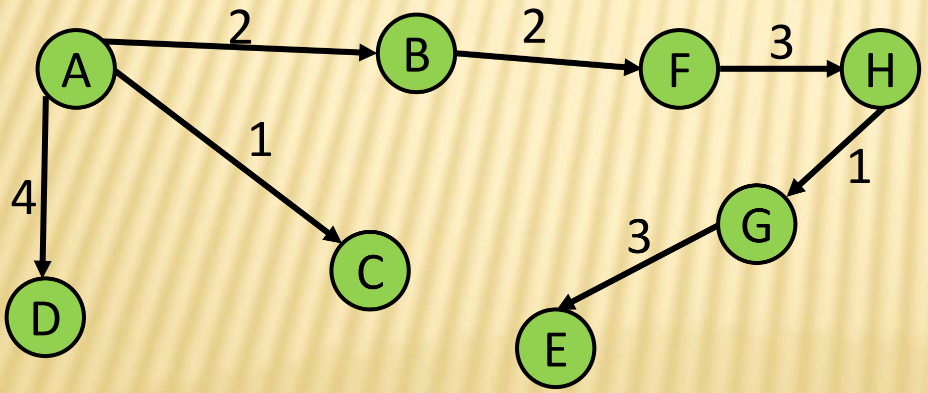
A, C, B, D, F, H, G, E

vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E	Y	11	G
F	Y	4	B
G	Y	8	H
H	Y	7	F

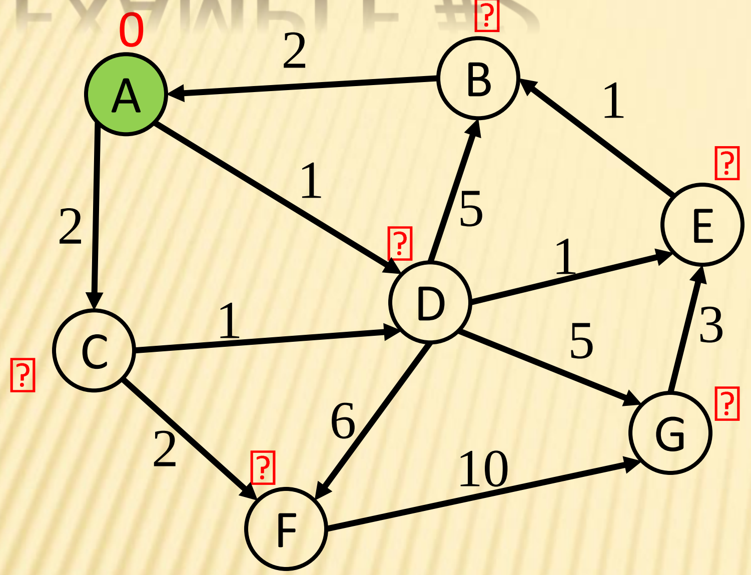
INTERPRETING THE RESULTS



vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E	Y	11	G
F	Y	4	B
G	Y	8	H
H	Y	7	F



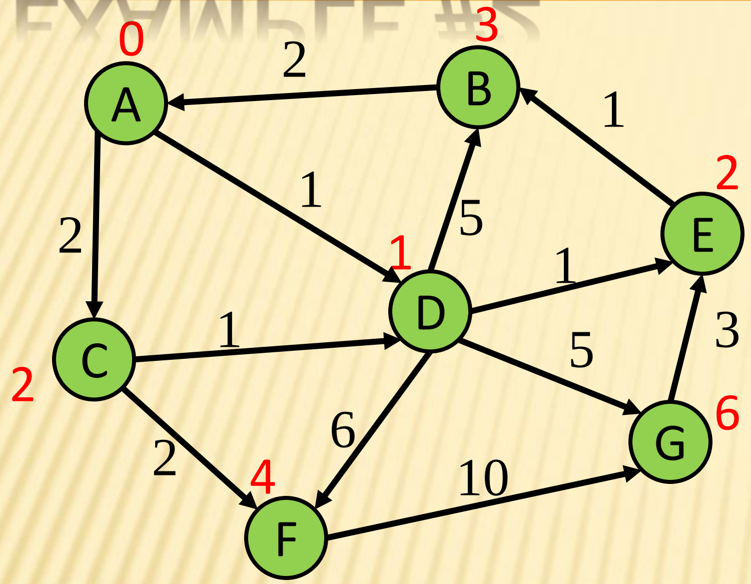
EXAMPLE #2



Order Added to Known Set:

vertex	known?	cost	path
A	Y	0	
B		∞	
C		∞	
D		∞	
E		∞	
F		∞	
G		∞	

Example #2



vertex	known?	cost	path
A	Y	0	
B	Y	3	E
C	Y	2	A
D	Y	1	A
E	Y	2	D
F	Y	4	C
G	Y	6	D

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

A, D, C, E, B, F, G

PSEUDOCODE ATTEMPT #1

```
dijkstra(Graph G, Node start) {
  for each node: x.cost=infinity, x.known=false
  start.cost = 0
  while(not all nodes are known) {
    b = dequeue
    b.known = true
    for each edge (b,a) in G {
      if(!a.known) {
        if(b.cost + weight((b,a)) < a.cost){
          a.cost = b.cost + weight((b,a))
          a.path = b
        }
      }
  …
```

$O(|V|)$

$O(|V|^2)$

$O(|E|)$

$O(|V|^2)$

CAN WE DO BETTER?

- Increase efficiency by considering lowest cost unknown vertex with sorting instead of looking at all vertices
- PriorityQueue is like a queue, but returns elements by lowest value instead of FIFO

PRIORITY QUEUE

- Increase efficiency by considering lowest cost unknown vertex with sorting instead of looking at all vertices
- PriorityQueue is like a queue, but returns elements by lowest value instead of FIFO
- Two ways to implement:
  1. Comparable
     a) class Node implements Comparable<Node>
     b) public int compareTo(other)
  2. Comparator
     a) class NodeComparator extends Comparator<Node>
     b) new PriorityQueue(new NodeComparator())

PSEUDOCODE ATTEMPT #2

```
dijkstra(Graph G, Node start) {
  for each node: x.cost=infinity, x.known=false
  start.cost = 0
  build-heap with all nodes
  while(heap is not empty) {
    b = deleteMin()
    if (b.known) continue;
    b.known = true
    for each edge (b,a) in G {
     if(!a.known) {
       add(b.cost + weight((b,a)) )
     }
  …
```

O(|V|)

O(|V|log|V|)

O(|E|log|V|)

O(|E|log|V|)

PROOF OF CORRECTNESS

- All the “known” vertices have the correct shortest path through induction
  - Initially, shortest path to start node has cost 0
  - If it stays true every time we mark a node “known”, then by induction this holds and eventually everything is “known” with shortes path
- Key fact: When we mark a vertex “known” we won’t discover a shorter path later
  - Remember, we pick the node with the min cost each round
  - Once a node is marked as “known”, going through another path will only add weight
  - Only true when node weights are positive

HW7 TEST SCRIPT COMMAND NOTES

- HW7 ***LoadGraph*** command is slightly different from HW6
  - After graph is loaded, there should be at most one directed edge from one node to another, with the edge label being the multiplicative inverse of the number of books shared
  - Example: If 8 books are shared between two nodes, the edge label will be 1/8
  - Since the edge relationship is symmetric, there would be another edge going the other direction with the same edge label

HW7 TEST SCRIPT COMMAND NOTES

- Let’s say **AddEdge** is called by the client after **LoadGraph**
- Now, two things may happen
  - There is a directed edge from one node to another, but not in the other direction (no longer symmetric)
    - Need not worry about this. It will be up to the client to run another **AddEdge** command if they want the symmetry
  - There are two directed edges from one node to the other
    - Make sure you choose the edge with the least cost in your pathfinding algorithm
    - Do not overwrite an existing edge or combine edge values in any way