# What do you call 8 Hobbits?

# What do you call 8 HobBITs?

A HobBYTE!

# Section 7:
# Dijkstra's Algorithm

SLIDES ADAPTED FROM ALEX MARIAKAKIS
WITH MATERIAL KELLEN DONOHUE, DAVID
MAILHOT, AND DAN GROSSMAN

# Agenda

- Reminders
  - HW 6 due tonight (7/26)
  - HW 7 due next Thursday (8/2)
- Regression Testing
- Subtyping
- Dijkstra’s Algorithm
- HW 7

# Regression testing

- It's important to make sure software still performs well after changes!
- Regression Testing is the practice of running existing tests after making changes
  - Your HW5 tests should still pass after you update your graph for HW6!
    - You will be graded on this!
  - You may update your HW5 tests

# Subtyping: Liskov Substitution Principle

- Refer to your worksheet…

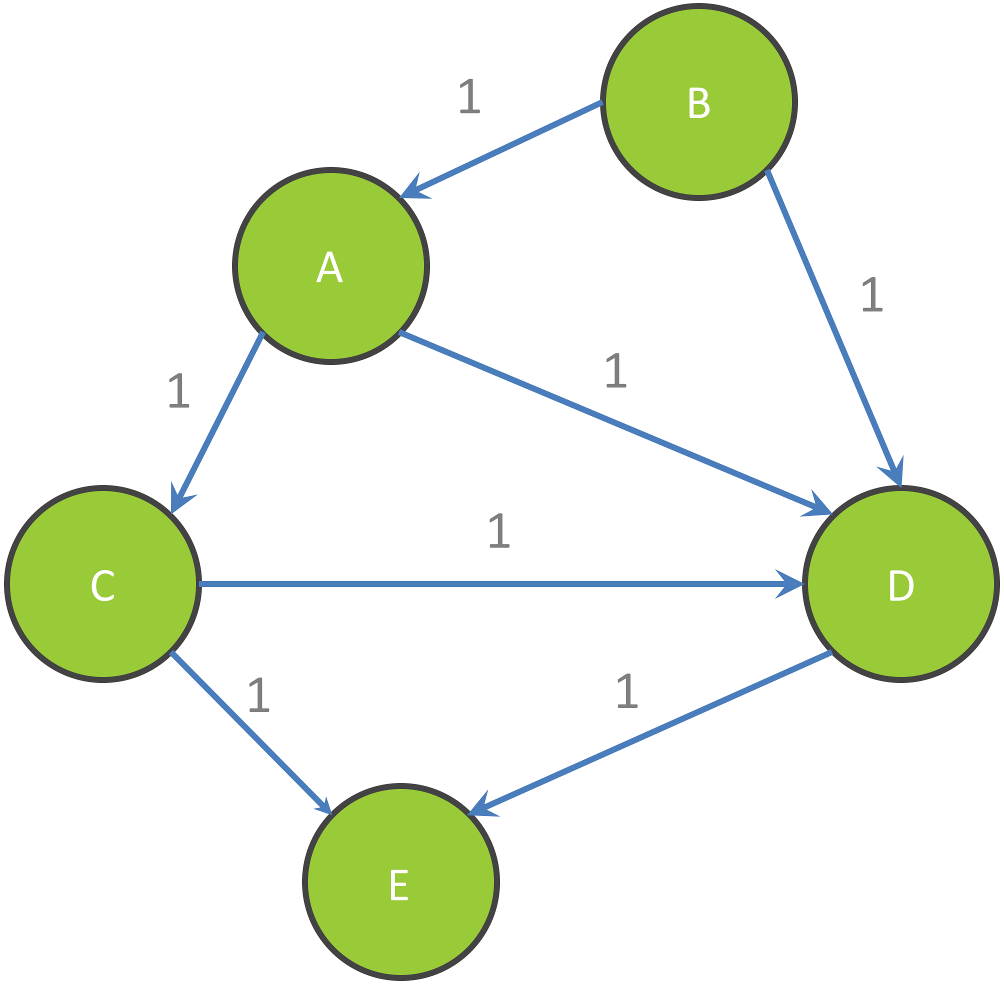
- LSP: If B is a true subtype of A, then B is substitutable for A.
  - Follows Principle of Least Surprise
- Is it enough to say "every B is an A"?
  - This is necessary but not sufficient for B to be a true subtype of A


Barbara Liskov

# Subtyping: Specifications

- If B is a true subtype of A, then B must have a stronger specification than A
  - Each method of B must have a stronger specification than the corresponding method of A (if any)
  - Stronger method spec means:
    - Requires less (weaker precondition)
    - Promises more (stronger postcondition)
- Worksheet time!

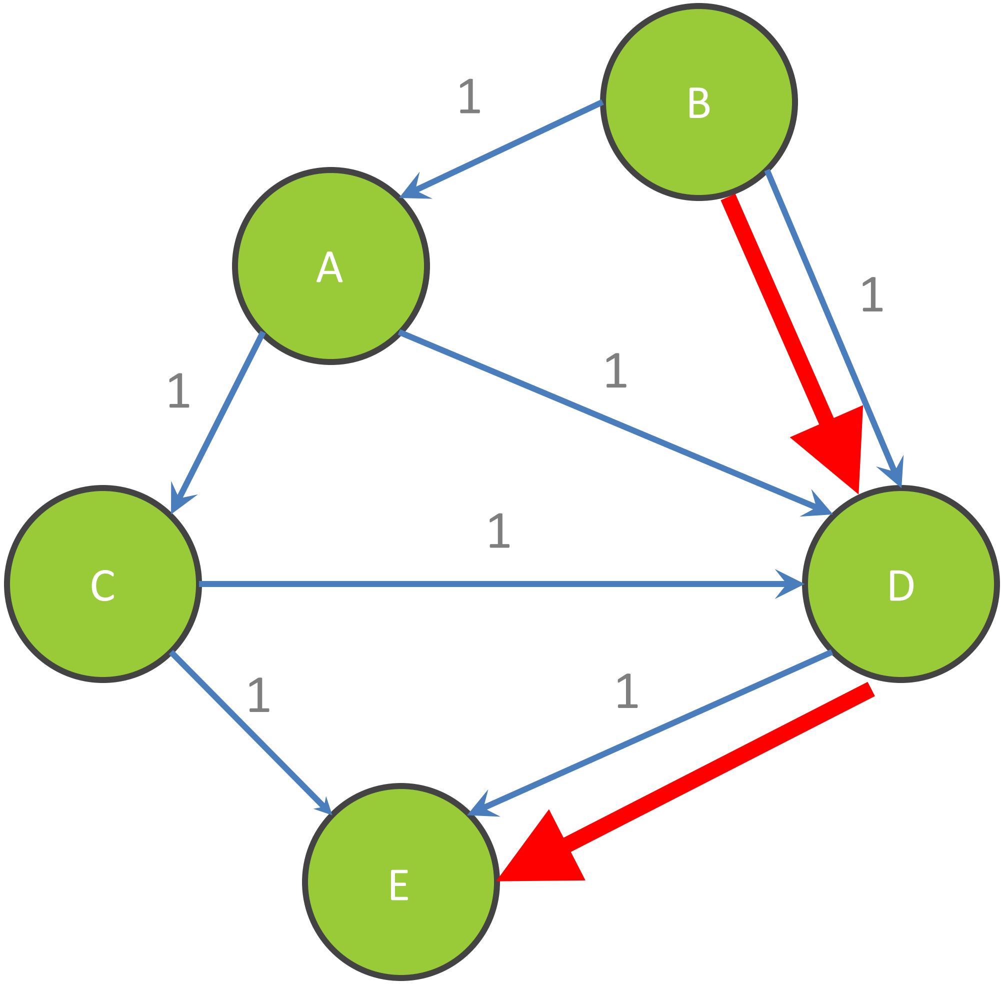
# Review: Shortest Paths with BFS




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 |

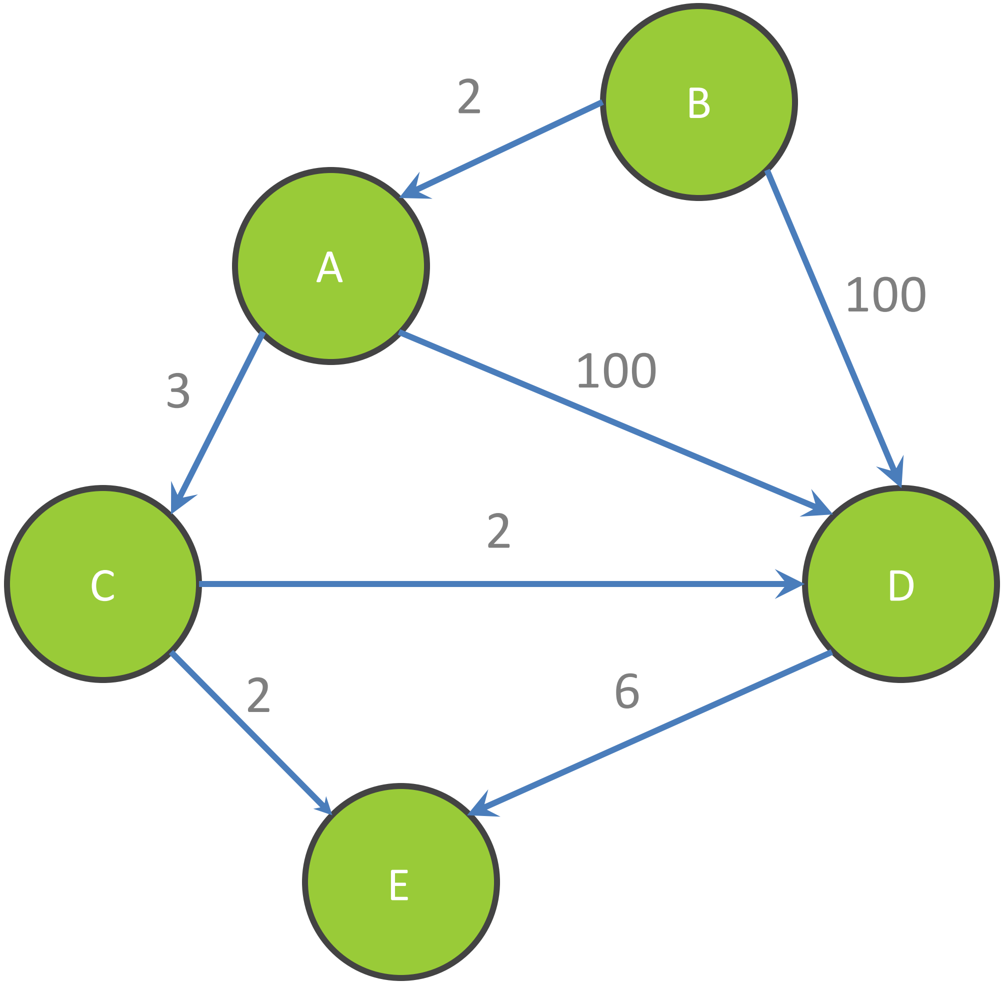
# Review: Shortest Paths with BFS



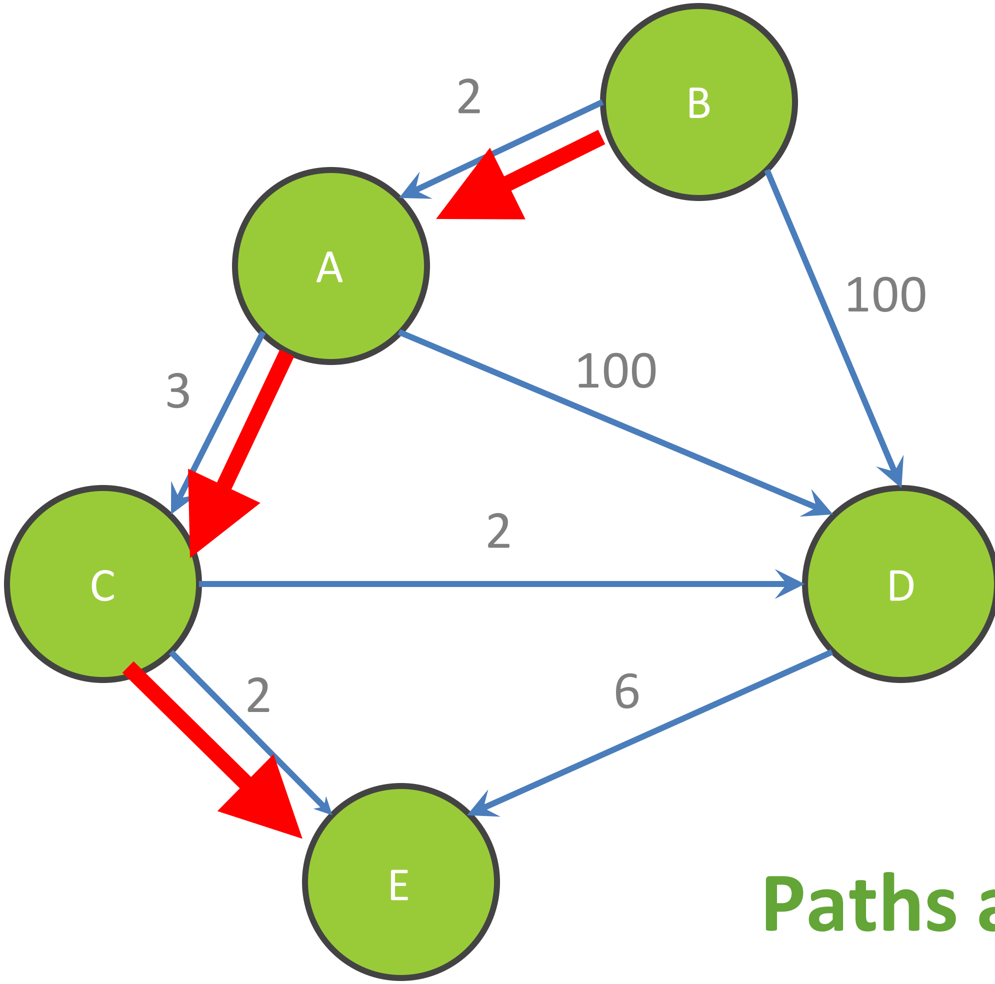

From Node B

| 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



# Shortest Paths with Weights

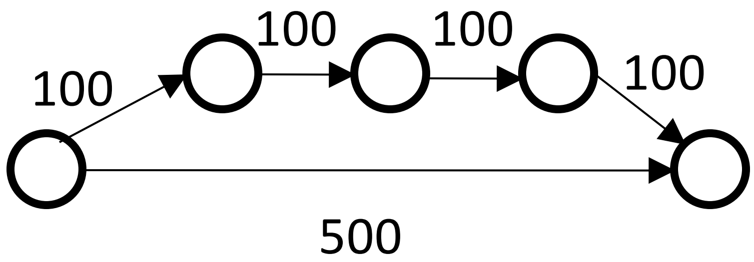


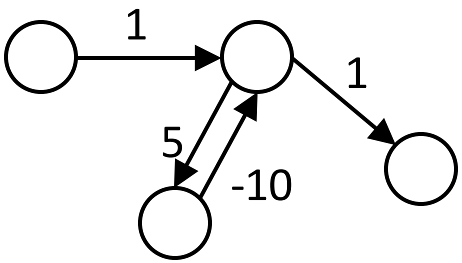

From Node B

| 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

Also, Dijkstra’s works if the weights are non-negative

What happens if there is a negative edge?

- Minimize cost by repeating the cycle forever (this is bad)
- How could we fix this?

# 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

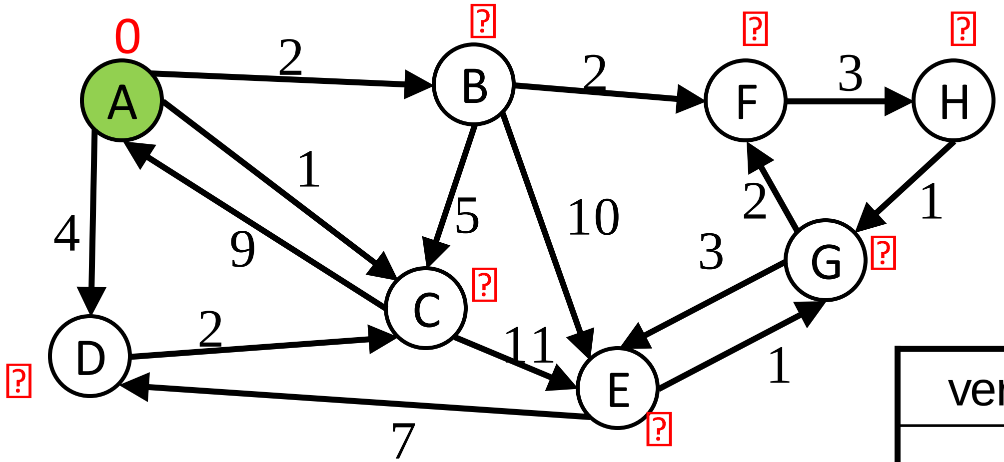
- Grow the set of nodes whose shortest distance has been computed
- Nodes not in the set will have a “best distance so far”
- A **PRIORITY QUEUE** will turn out to be useful for efficiency – We’ll cover this later in the slide deck

# 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

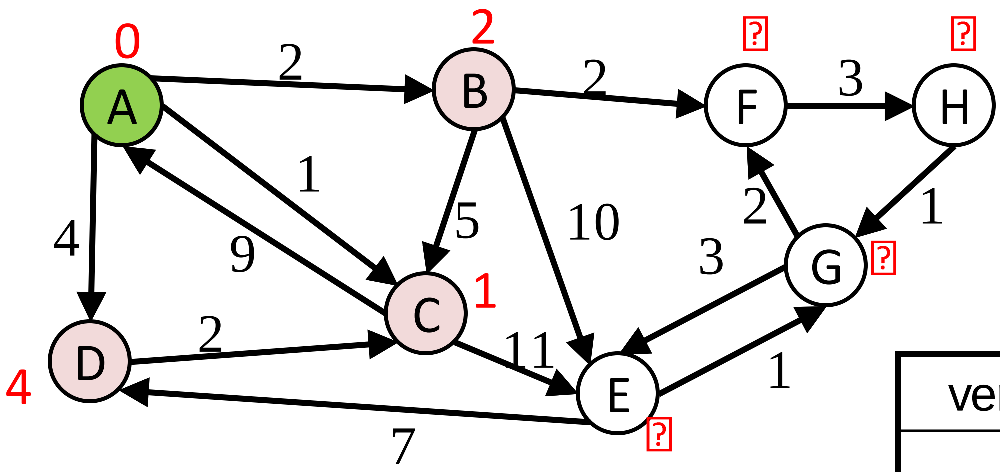



Goal: Fully explore the graph

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

Order Added to Known Set:

# Example #1

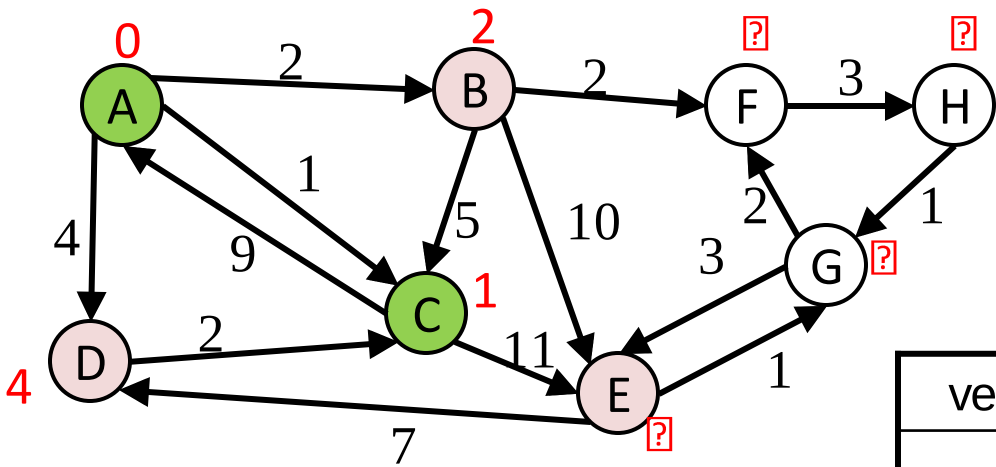



| vertex | known? | cost | path |
|---|---|---|---|
| A | Y | 0 | |
| B | | ≤ 2 | A |
| C | | ≤ 1 | A |
| D | | ≤ 4 | A |
| E | | ∞ | |
| F | | ∞ | |
| G | | ∞ | |
| H | | ∞ | |

Order Added to Known Set:

A

# Example #1

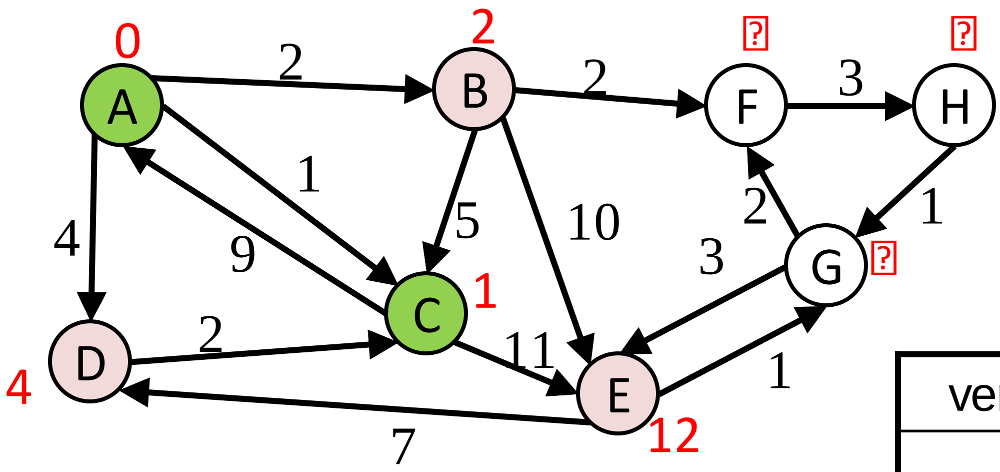



| vertex | known? | cost | path |
|---|---|---|---|
| A | Y | 0 | |
| B | | ≤ 2 | A |
| C | Y | 1 | A |
| D | | ≤ 4 | A |
| E | | ∞ | |
| F | | ∞ | |
| G | | ∞ | |
| H | | ∞ | |

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

A, C

# Example #1

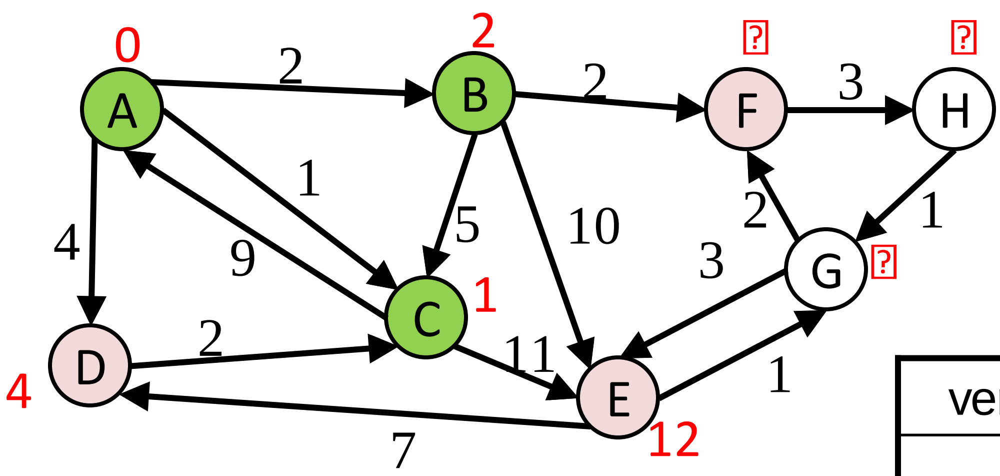



Order Added to Known Set:

A, C

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

# Example #1

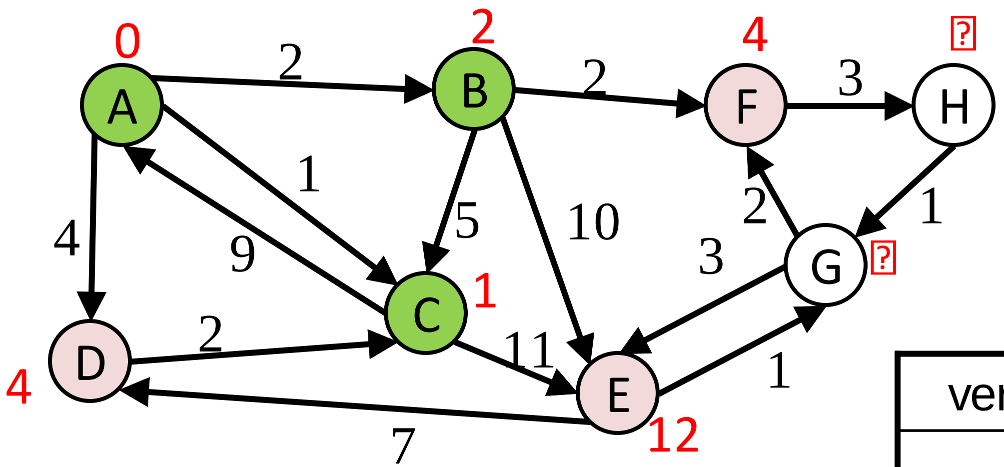



| vertex | known? | cost | path |
|---|---|---|---|
| A | Y | 0 | |
| B | Y | 2 | A |
| C | Y | 1 | A |
| D | | ≤ 4 | A |
| E | | ≤ 12 | C |
| F | | ∞ | |
| G | | ∞ | |
| H | | ∞ | |

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

A, C, B

# Example #1

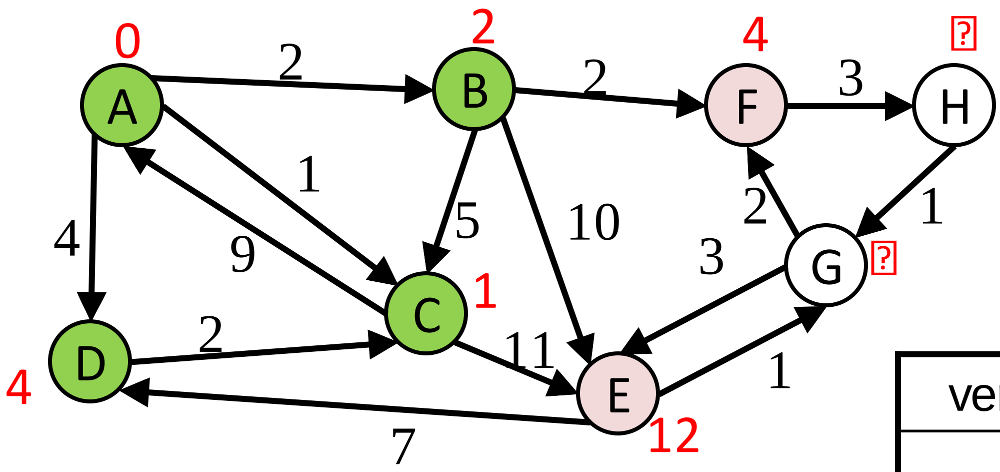



| vertex | known? | cost | path |
|---|---|---|---|
| A | Y | 0 | |
| B | Y | 2 | A |
| C | Y | 1 | A |
| D | | ≤ 4 | A |
| E | | ≤ 12 | C |
| F | | ≤ 4 | B |
| G | | ∞ | |
| H | | ∞ | |

Order Added to Known Set:

A, C, B

# Example #1

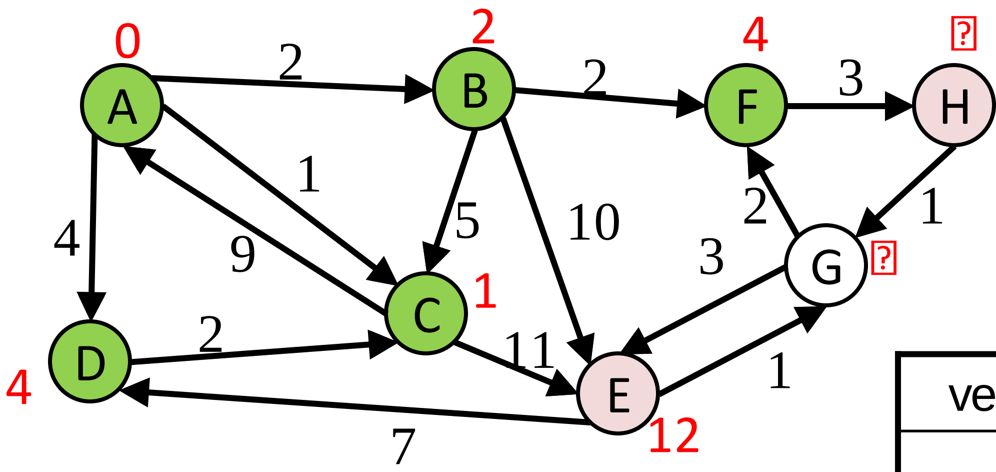



| vertex | known? | cost | path |
|---|---|---|---|
| A | Y | 0 | |
| B | Y | 2 | A |
| C | Y | 1 | A |
| D | Y | 4 | A |
| E | | ≤ 12 | C |
| F | | ≤ 4 | B |
| G | | ∞ | |
| H | | ∞ | |

Order Added to Known Set:

A, C, B, D

# Example #1

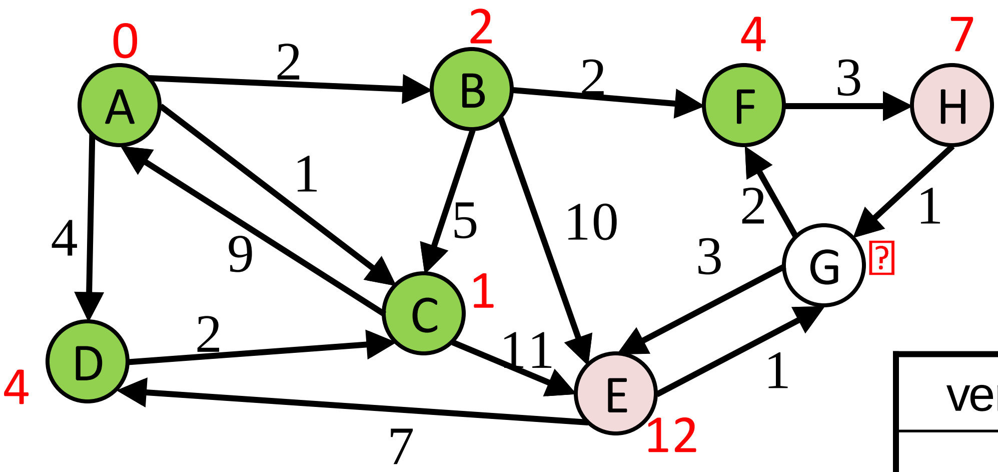



| vertex | known? | cost | path |
|---|---|---|---|
| A | Y | 0 | |
| B | Y | 2 | A |
| C | Y | 1 | A |
| D | Y | 4 | A |
| E | | ≤ 12 | C |
| F | Y | 4 | B |
| G | | ∞ | |
| H | | ∞ | |

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

A, C, B, D, F

# Example #1

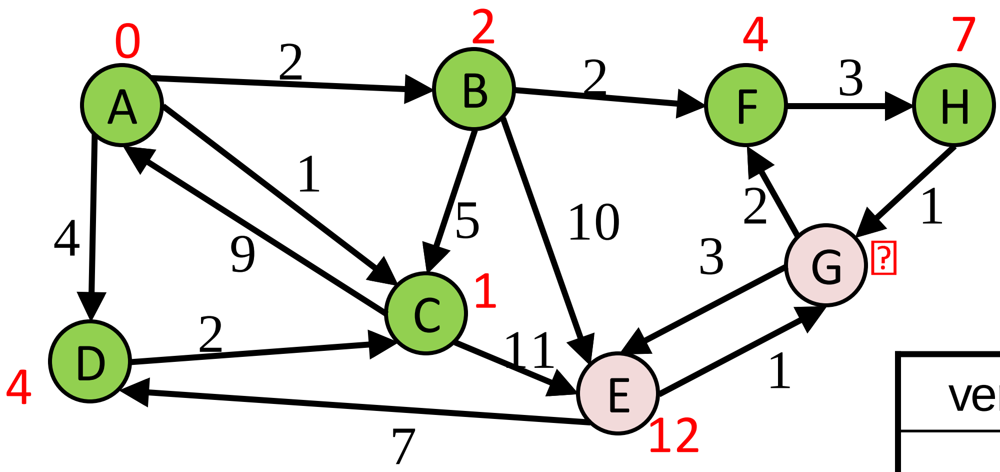



| vertex | known? | cost | path |
|---|---|---|---|
| A | Y | 0 | |
| B | Y | 2 | A |
| C | Y | 1 | A |
| D | Y | 4 | A |
| E | | ≤ 12 | C |
| F | Y | 4 | B |
| G | | ∞ | |
| H | | ≤ 7 | F |

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

A, C, B, D, F

# Example #1

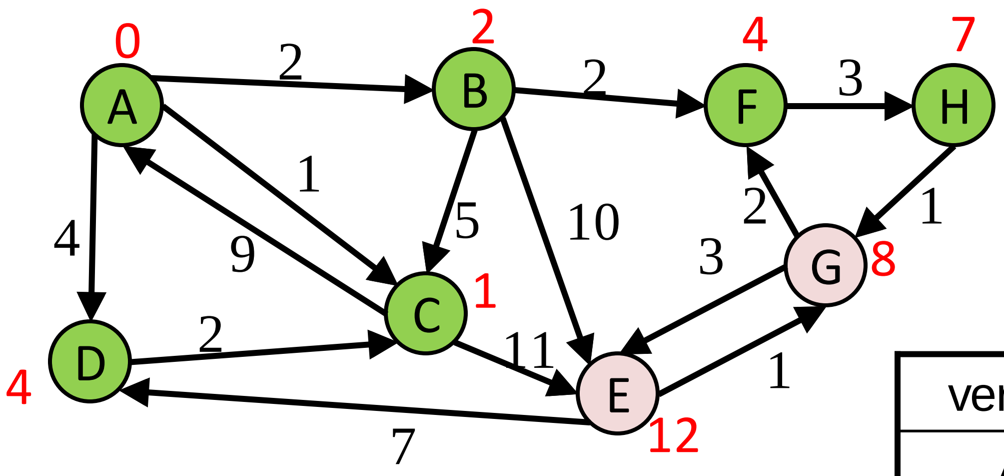



| vertex | known? | cost | path |
|---|---|---|---|
| A | Y | 0 | |
| B | Y | 2 | A |
| C | Y | 1 | A |
| D | Y | 4 | A |
| E | | ≤ 12 | C |
| F | Y | 4 | B |
| G | | ∞ | |
| H | Y | 7 | F |

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

A, C, B, D, F, H

# Example #1

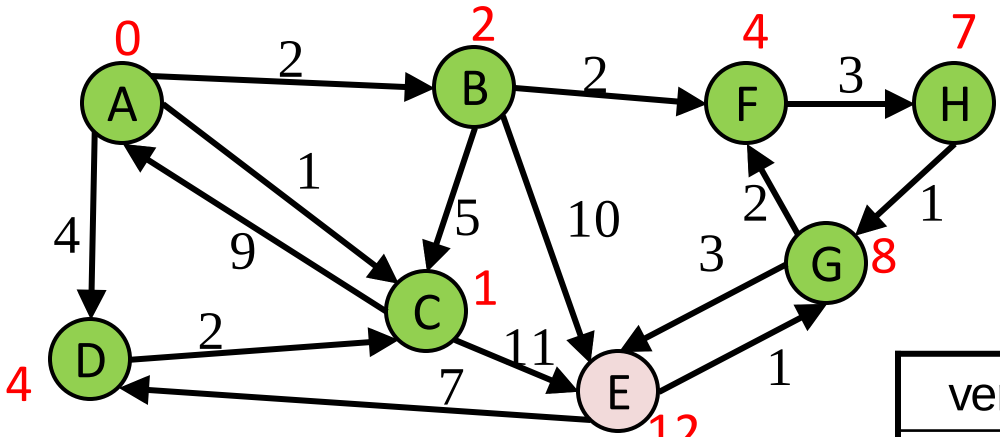



Order Added to Known Set:

A, C, B, D, F, H

| 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 |

# Example #1

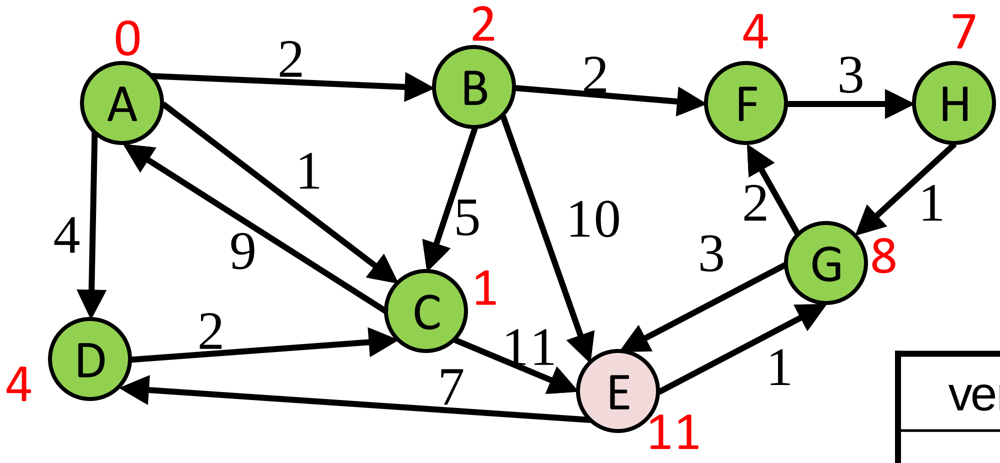



| vertex | known? | cost | path |
|---|---|---|---|
| A | Y | 0 | |
| B | Y | 2 | A |
| C | Y | 1 | A |
| D | Y | 4 | A |
| E | | ≤ 12 | C |
| F | Y | 4 | B |
| G | Y | 8 | H |
| H | Y | 7 | F |

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

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

# Example #1

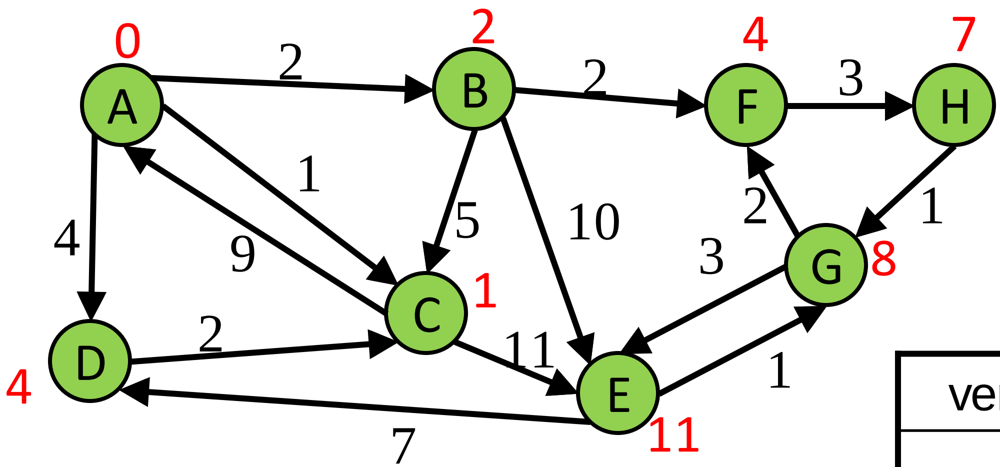



| vertex | known? | cost | path |
|---|---|---|---|
| A | Y | 0 | |
| B | Y | 2 | A |
| C | Y | 1 | A |
| D | Y | 4 | A |
| E | | ≤ 11 | G |
| F | Y | 4 | B |
| G | Y | 8 | H |
| H | Y | 7 | F |

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

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

# Example #1

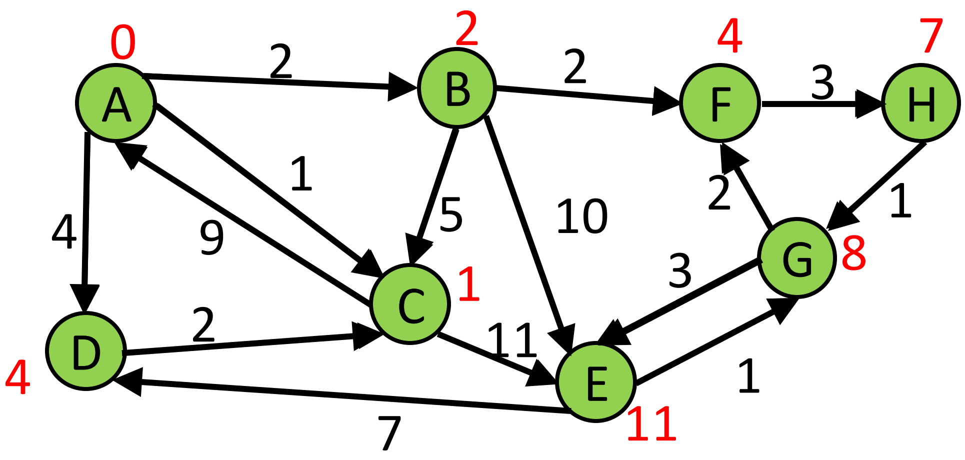



| vertex | known? | cost | path |
|---|---|---|---|
| A | Y | 0 | |
| B | Y | 2 | A |
| C | Y | 1 | A |
| D | Y | 4 | A |
| E | Y | 11 | G |
| F | Y | 4 | B |
| G | Y | 8 | H |
| H | Y | 7 | F |

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

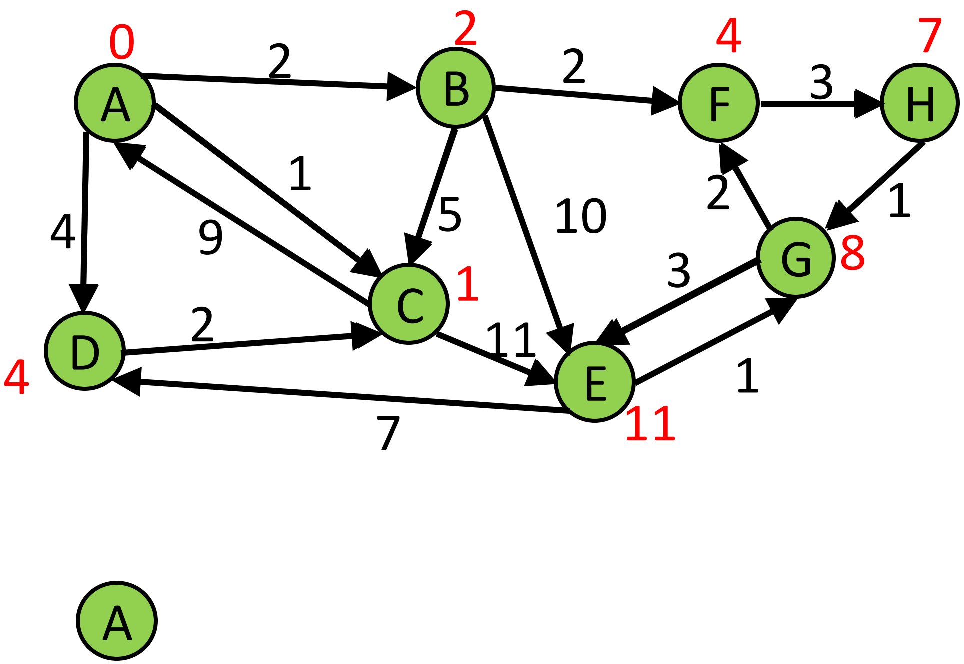
A, C, B, D, F, H, G, E

# Interpreting the Results



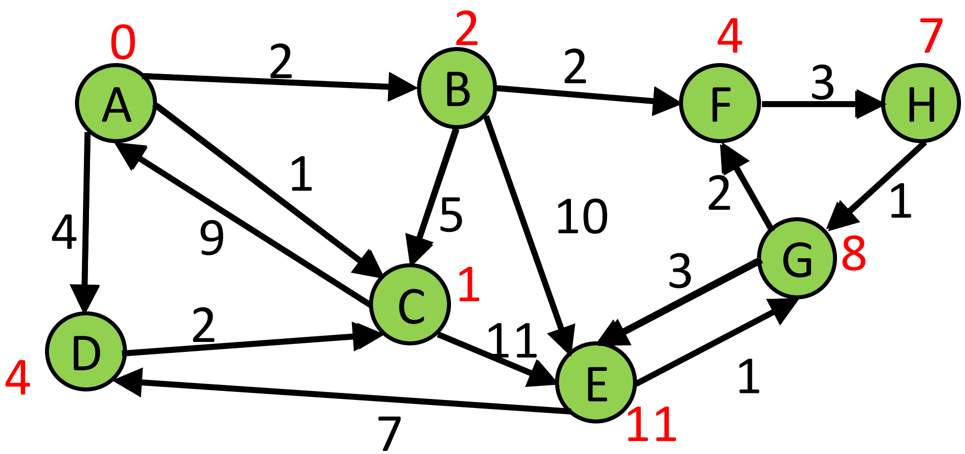

| vertex | known? | cost | path |
|---|---|---|---|
| A | Y | 0 | |
| B | Y | 2 | A |
| C | Y | 1 | A |
| D | Y | 4 | A |
| E | Y | 11 | G |
| F | Y | 4 | B |
| G | Y | 8 | H |
| H | Y | 7 | F |

# Interpreting the Results

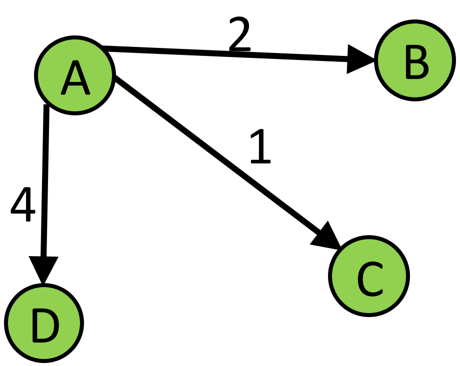



| vertex | known? | cost | path |
|---|---|---|---|
| A | Y | 0 | |
| B | Y | 2 | A |
| C | Y | 1 | A |
| D | Y | 4 | A |
| E | Y | 11 | G |
| F | Y | 4 | B |
| G | Y | 8 | H |
| H | Y | 7 | F |

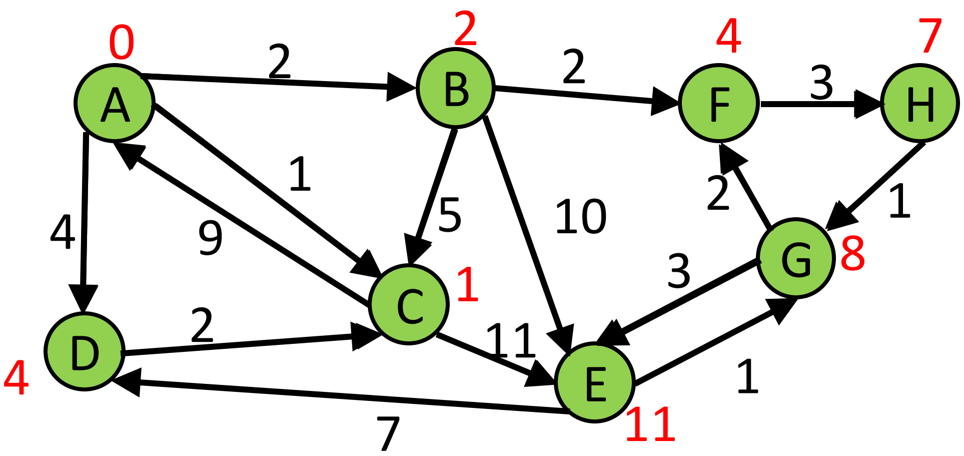
# Interpreting the Results



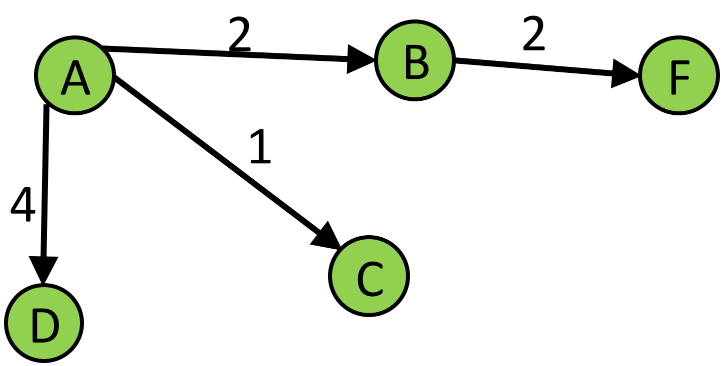

| vertex | known? | cost | path |
|---|---|---|---|
| A | Y | 0 | |
| B | Y | 2 | A |
| C | Y | 1 | A |
| D | Y | 4 | A |
| E | Y | 11 | G |
| F | Y | 4 | B |
| G | Y | 8 | H |
| H | Y | 7 | F |



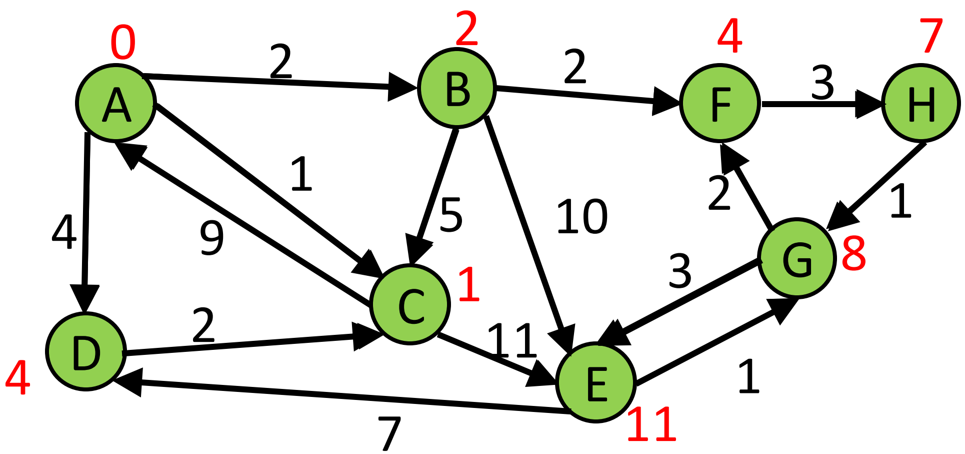
# Interpreting the Results

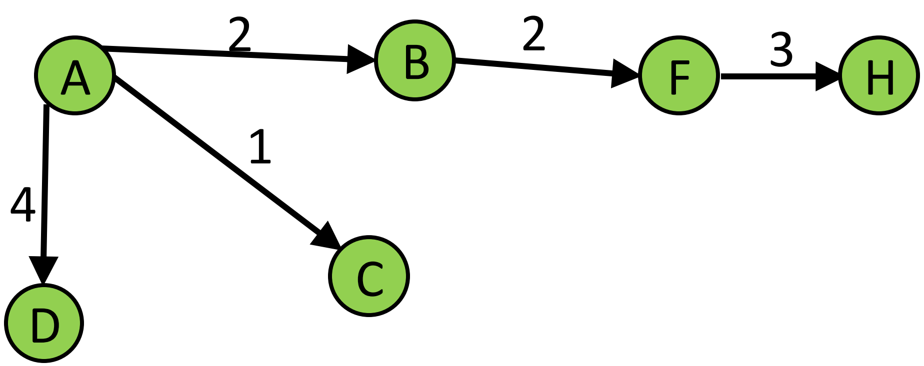



| vertex | known? | cost | path |
|---|---|---|---|
| A | Y | 0 | |
| B | Y | 2 | A |
| C | Y | 1 | A |
| D | Y | 4 | A |
| E | Y | 11 | G |
| F | Y | 4 | B |
| G | Y | 8 | H |
| H | Y | 7 | F |



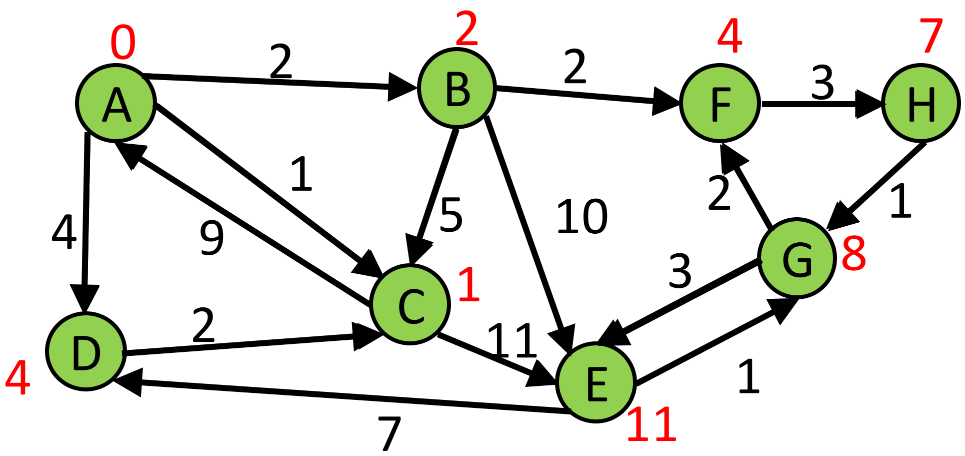
# Interpreting the Results

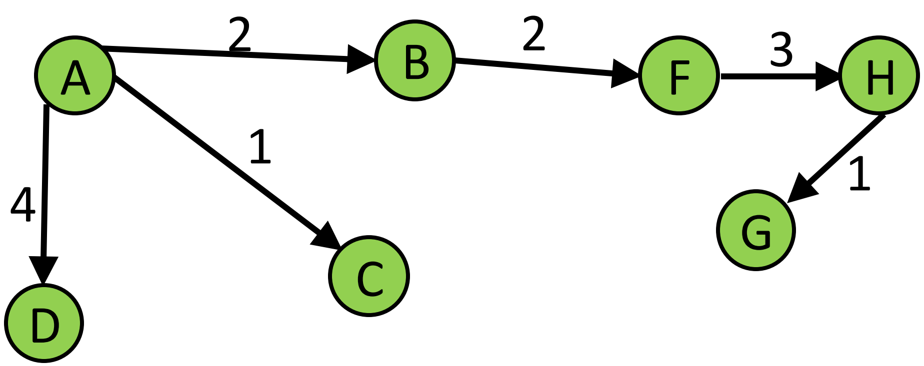



| vertex | known? | cost | path |
|---|---|---|---|
| A | Y | 0 | |
| B | Y | 2 | A |
| C | Y | 1 | A |
| D | Y | 4 | A |
| E | Y | 11 | G |
| F | Y | 4 | B |
| G | Y | 8 | H |
| H | Y | 7 | F |

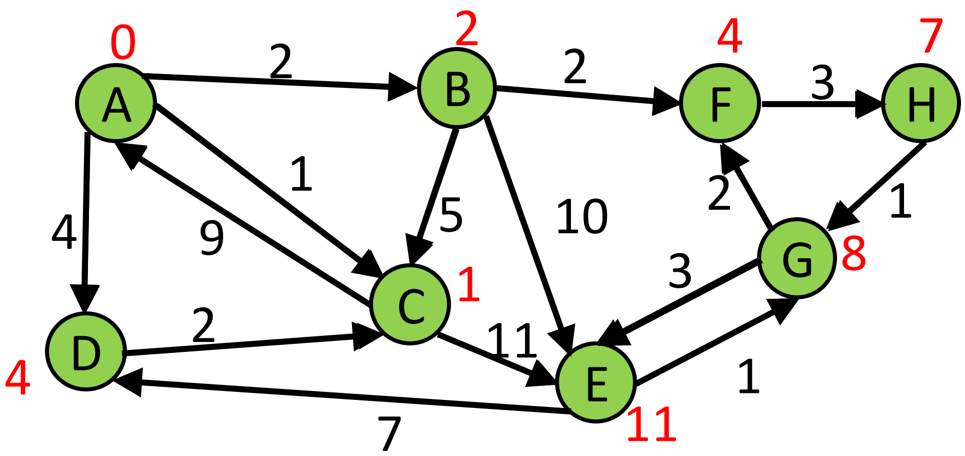
# Interpreting the Results

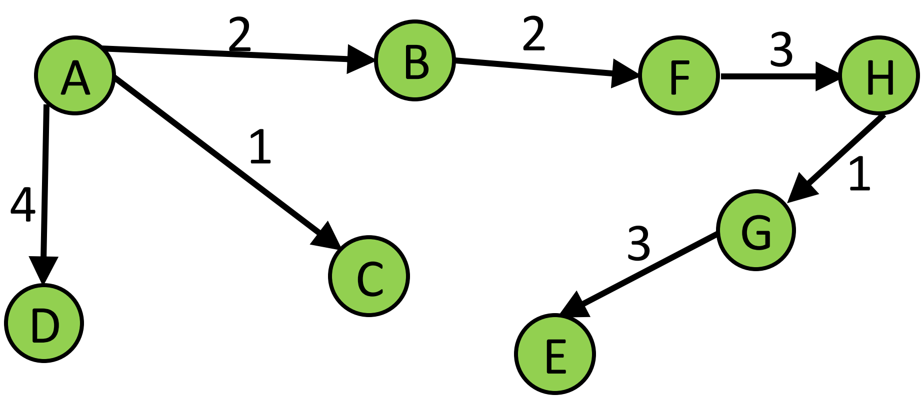



| vertex | known? | cost | path |
|---|---|---|---|
| A | Y | 0 | |
| B | Y | 2 | A |
| C | Y | 1 | A |
| D | Y | 4 | A |
| E | Y | 11 | G |
| F | Y | 4 | B |
| G | Y | 8 | H |
| H | Y | 7 | F |

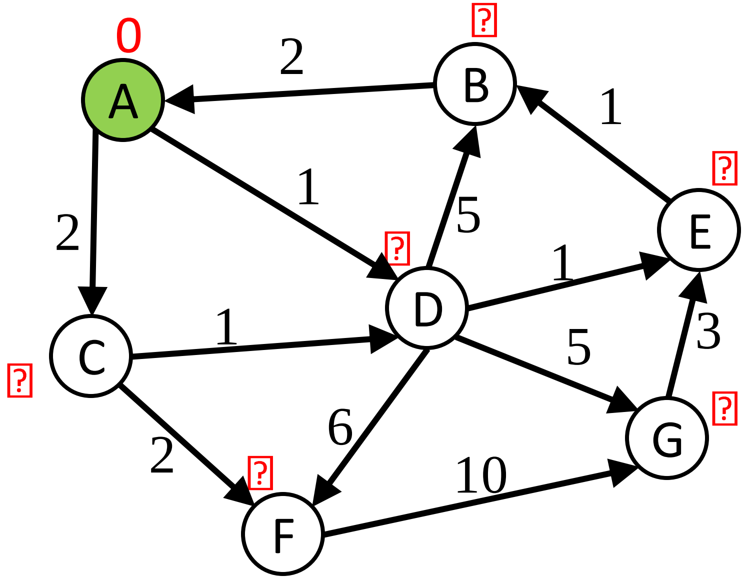
# Interpreting the Results




| 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 |

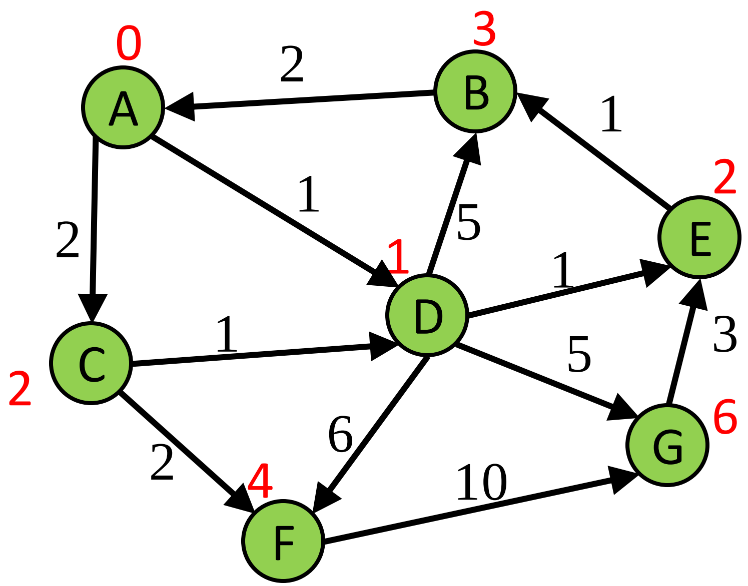
# Example #2




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 |

Order Added to Known Set:

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

# Pseudocode

```
// pre-condition: start is the node to start at
// initialize things
active = new empty priority queue of paths
    from start to a given node
    // A path's “priority” in the queue is the total
    // cost of that path.

finished = new empty set of nodes
    // Holds nodes for which we know the
    // minimum-cost path from start.

// We know path start->start has cost 0
Add a path from start to itself to active
```

# Pseudocode (cont.)

```
while active is non-empty:
      minPath = active.removeMin()
      minDest = destination node in minPath

      if minDest is in finished:
          continue

      for each edge e = ⟨minDest, child⟩:
          if child is not in finished:
              newPath = minPath + e
              add newPath to active

      add minDest to finished
```

# 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

# 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())

# Homework 7

Modify your graph to use generics

- **Will have to update graph and old tests!**
- (**<u>all</u>** of your old tests should still pass)

Implement Dijkstra's algorithm

- 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

# Hw7 Important Notes!!!

DO NOT access data from hw6/src/data

- Copy over data files from hw6/src/data into hw7/src/data, and access data in hw7 from there instead
- Remember to do this! Or tests will fail when grading.

DO NOT modify ScriptFileTests.java

# 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