Section 6:
Dijkstra's Algorithm & HW7

CSE 331 STAFF

Homework 7

Modify your graph to use generics

- Will have to update HW #5 and HW #6 tests
- discussed in lecture on Friday (and Monday)

Implement Dijkstra's algorithm for shortest paths in *weighted* graphs

- Note: This should not change your implementation of Graph. Dijkstra's is performed on a Graph, not within a Graph.
- discussed in section (now)

Rest (test driver, etc.) as in HW6

HW7 Overview

Homework 7

Create a *weighted* graph between Marvel characters

- labels on the edges are numbers called “weights”

The weight of an edge is equal to the inverse of how many comic books the characters appeared in together

- Ex: if Amazing Amoeba and Zany Zebra appeared in 5 comic books together, the weight of their edge would be 1/5

The more well-connected two characters are, the lower the weight

- path in this graph will have shorter length (less total weight) if adjacent characters along the path appeared frequently together

Graph Activity

List the Characters set, the Books->Characters map, and draw the graph using these characters and “books”.

Harry	HP1
Harry	HP2
Harry	HP3
Harry	HP4
Quirrel	HP1
Scabbers	HP1
Scabbers	HP2
Voldemort	HP4
Voldemort	SharedAHead
Quirrel	SharedAHead

Graph Activity Answers

Characters

Harry, Quirrel, Scabbers

Books -> Characters

HP1 -> Harry, Quirrel, Scabbers

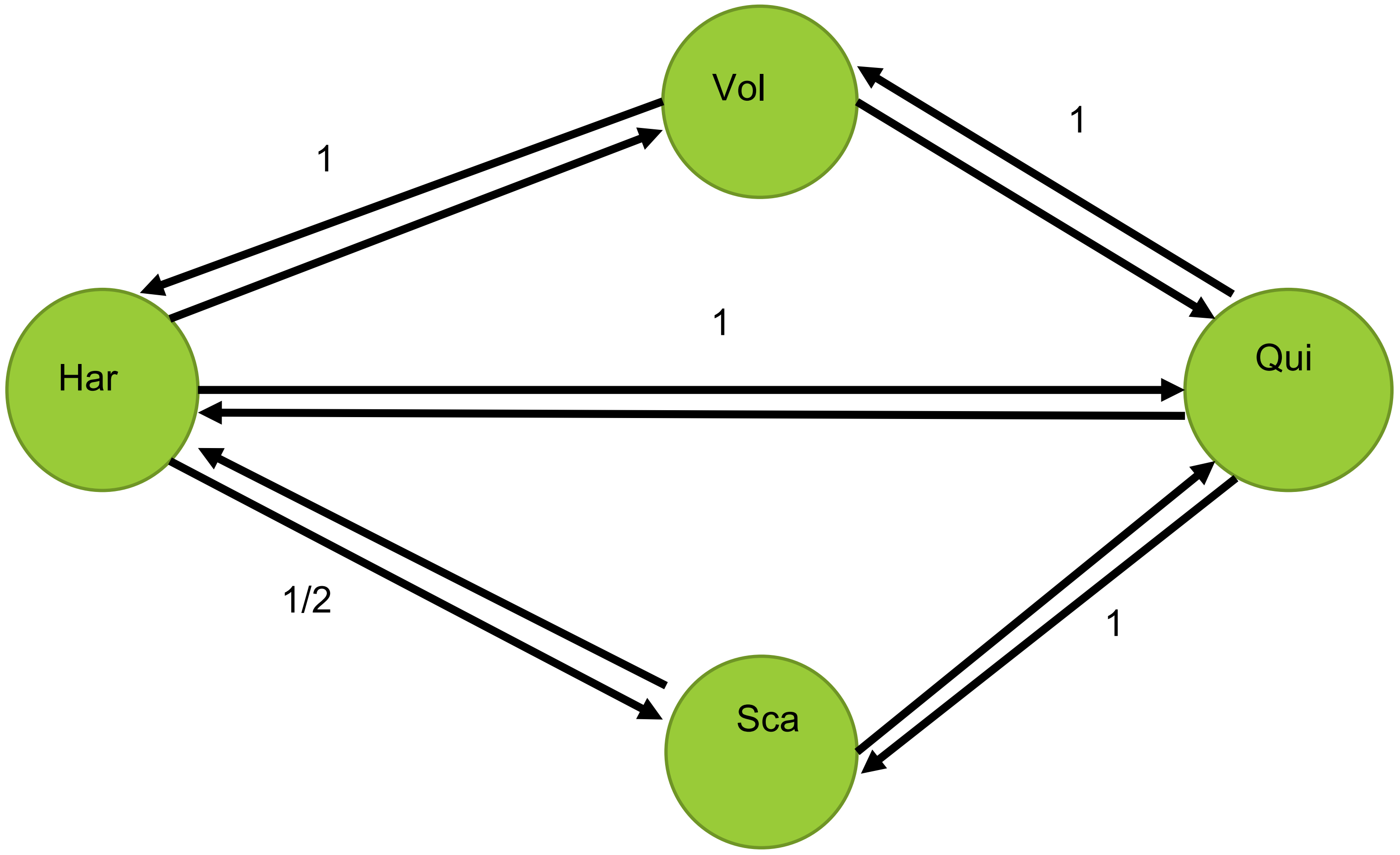
HP2 -> Harry, Scabbers,

HP3 -> Harry

HP4 -> Harry, Voldemort

SharedAHead -> Voldemort, Quirrel

Graph Activity Answers



Hw7 Test script Command Notes

HW7 ***LoadGraph*** command is slightly different from HW6

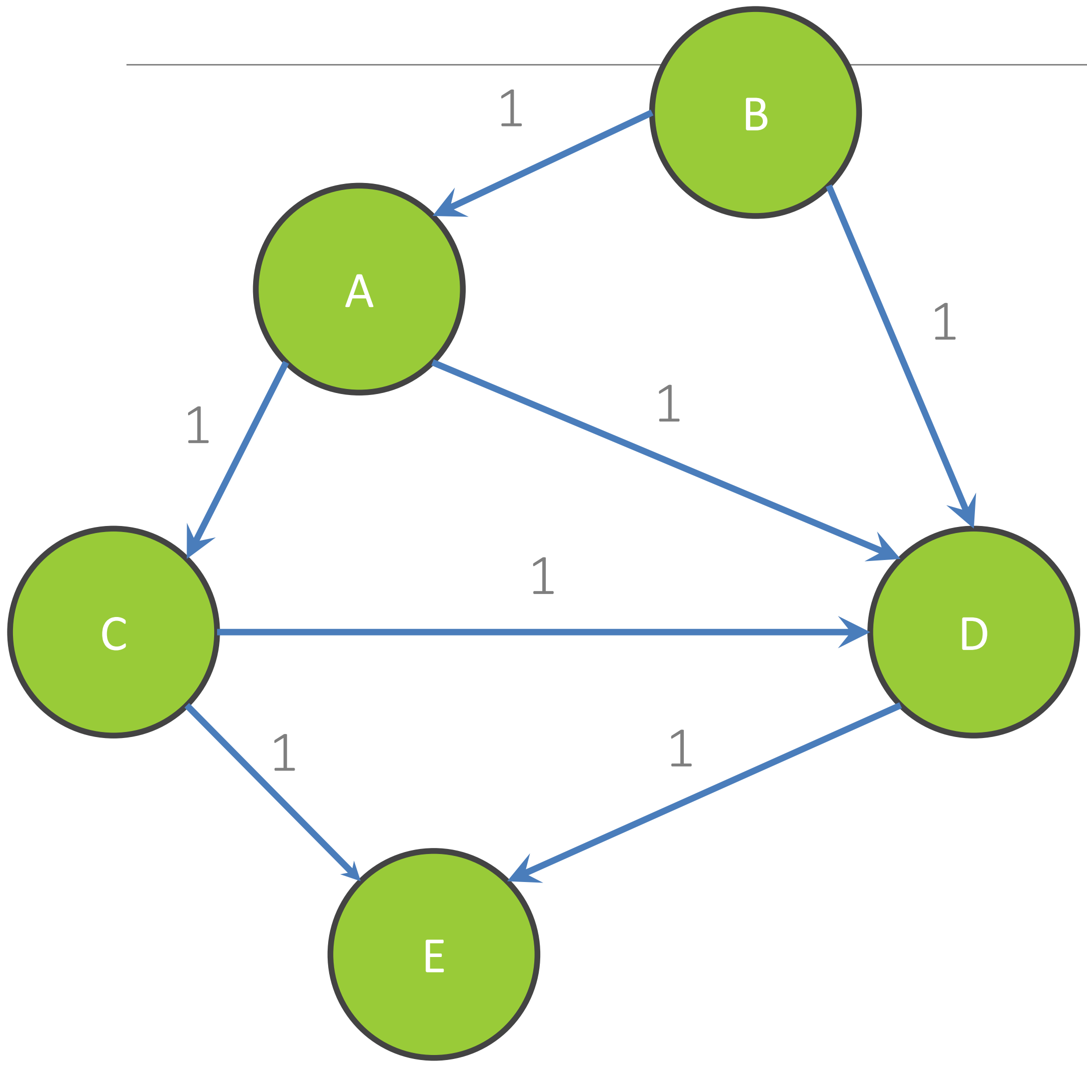
- After graph is loaded, there should be at most one directed edge from one node to another, with the edge label being the multiplicative inverse of the number of books shared

- 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

Dijkstra’s Algorithm

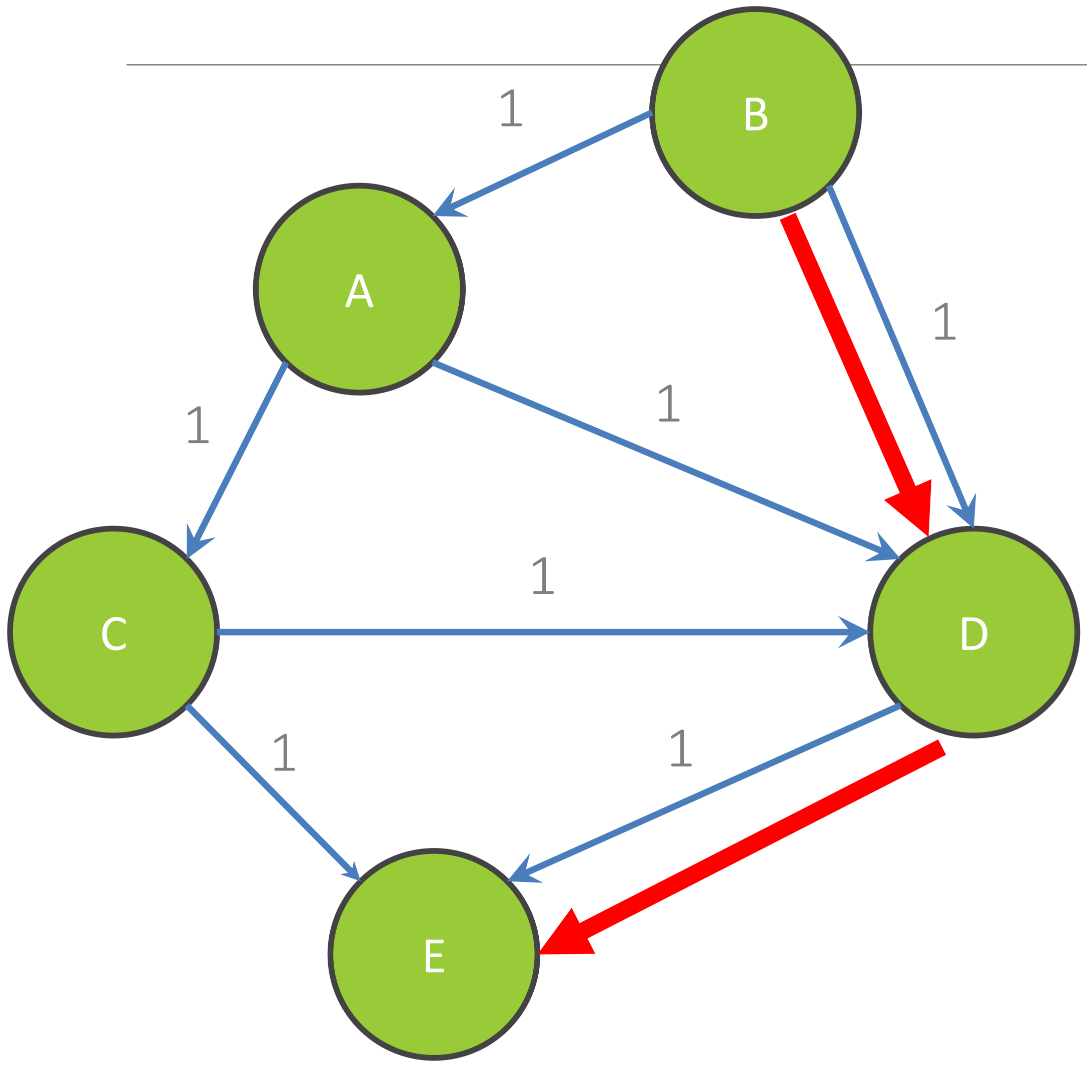
Review: Shortest Paths with BFS



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

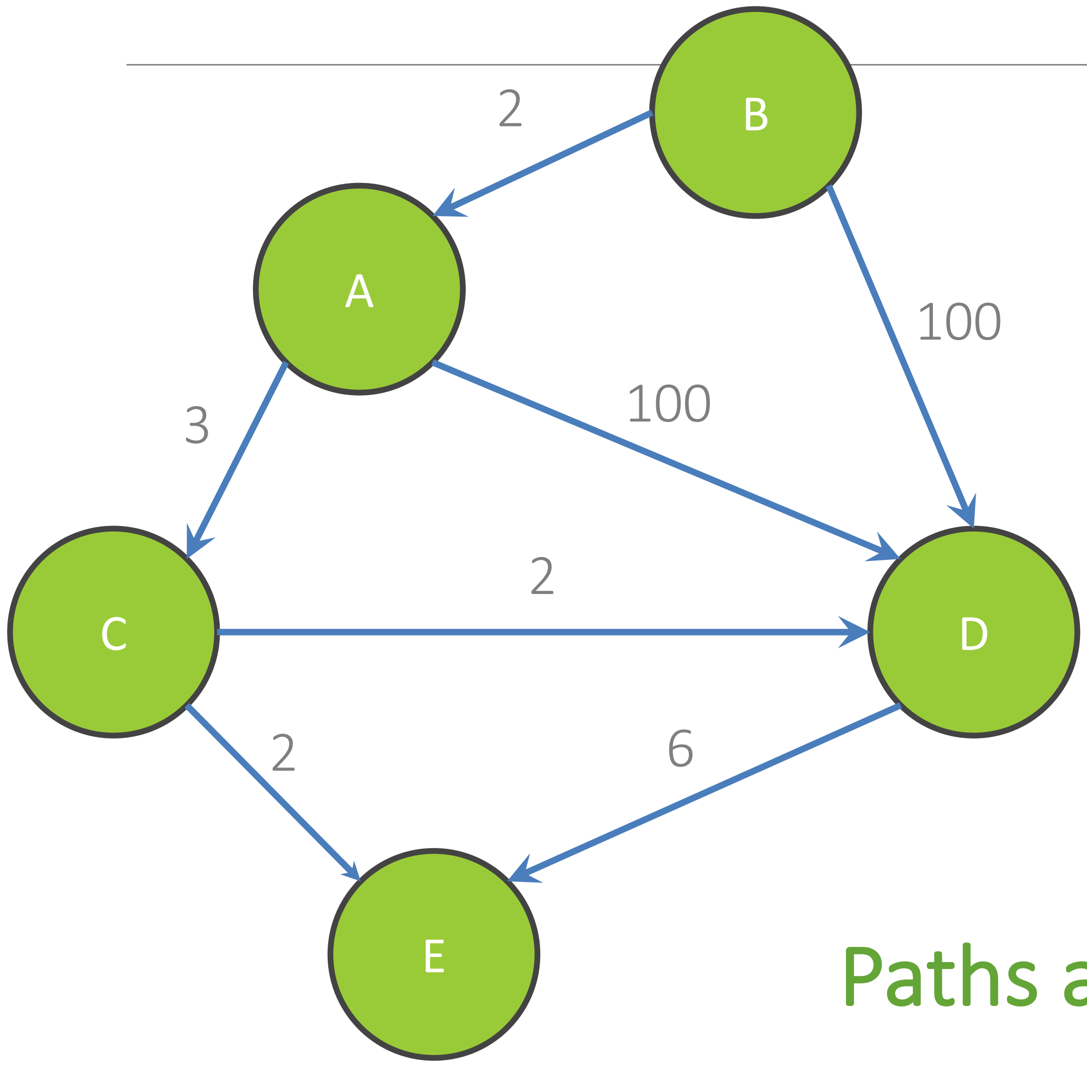
Review: Shortest Paths with BFS



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

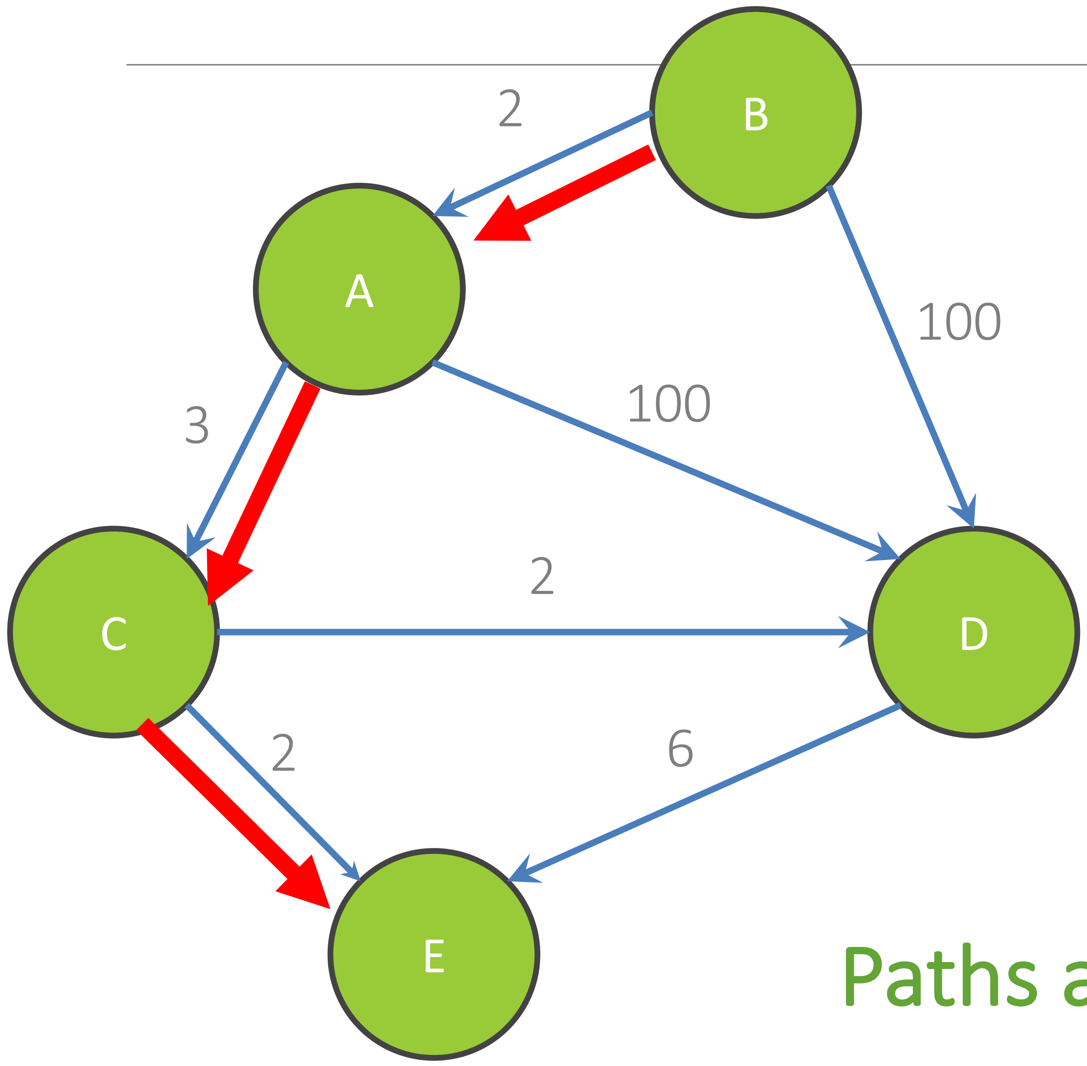


From Node B

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!

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!

Goal: Smallest cost? Or fewest edges?

BFS with weights

Why doesn’t BFS work correctly with weighted edges?

Consider the code again:

```
{{ Inv: list contains all nodes reachable from start
        by passing only through nodes list[0], ..., list[i-1]
        in order by their distance from start }}
while i < list.length:
        add children of list[i] not already in list to the end
        i = i + 1
```

Distance of new nodes is 1 + distance of list[i]
Distance of earlier nodes is 1 + distance of list[j] for some j < i
And we have distance to list[j] < distance to list[i] by Inv

BFS with weights

Why doesn’t BFS work correctly with weighted edges?

What happens if edges have weights?

Distance of new nodes is w + distance of list[i] for some weight w
Distance of earlier nodes is w’ + distance of list[j] for some j < i

distance to list[j] < distance to list[i]...

implies 1 + distance to list[j] < 1 + distance to list[i]

BUT does not imply w’ + distance to list[j] < w + distance to list[i]
since w’ could be much bigger than w

BFS with weights

How do we fix this?

Actually, the invariant is still fine! But the body doesn't preserve it:

```
{{ Inv: list contains all nodes reachable from start
            by passing only through nodes list[0], ..., list[i-1]
            in order by their distance from start }}
while i < list.length:
            add children of list[i] not already in list to the end
            i = i + 1
```

Problems:

1. new nodes do not necessarily go *at the end* of the list
2. could discover a shorter path to a node in the list already

BFS with Queue

In HW6 version of BFS:

- keep reachable elements in a set
  - only need the ability to see if we have found them already
- keep list[i], list[i+1], ... in a queue
  - only remove from front and add to end

Need to replace queue with something that can allow new elements to end up in the middle, not just at end...

Priority Queue

Priority queue is like a queue
BUT returns elements in order instead of FIFO

Two ways to provide order for Java’s PriorityQueue:

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())

Priority Queue Example

Suppose Q = [4, 8] is a priority queue

```
Q.add(3);
Q.add(5);
```

Q = [3, 4, 5, 8]

```
x = Q.remove();
```

Q = [4, 5, 8]

x = 3

Dijkstra’s Algorithm

- Uses a priority queue instead of a queue

- Elements in the priority queue are **paths** not nodes
  - different paths to different non-visited nodes
  - allow for multiple paths to each non-visited node

- Paths are ordered in queue by length (total weight)
  - shorter paths are removed earlier from the queue

Pseudocode with Priority Queue

```
Inputs: start = starting node

let Q = priority queue of paths ordered by length
let V = set of visited nodes (shortest paths already found)

add a path from start to itself to Q

// Inv: found shortest to each node in V
//      Q contains all paths of the form p + e, where
//        p is a shortest path to a visited node and
//        e is an edge to a non-visited node
while Q is non-empty:
   let path = Q.remove()
   let dest = final node of path
   if dest not in V:
       add dest to V
       for each edge e = <dest, child>:
           if child is not in V:
               let newPath = path + e
               add newPath to Q
```

Dijkstra's Algorithm

Reminder
Loop invariants get more important as the algorithms get more complex

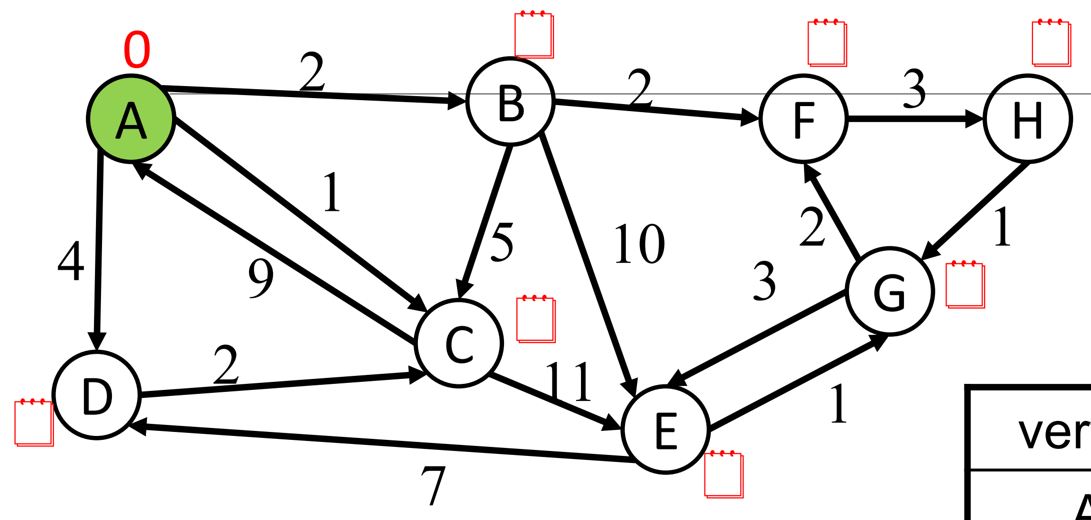
Is Inv preserved by loop body?

- invariant for Q requires adding each path of the form p + e
- invariant for V requires that path found is the shortest path

Claim: path found is a shortest path (out of scope)

- suppose there was a shorter path to the same node
- can assume it is of the form p' + e' where end of p' is visited
  - in general, the path must be of the form p' + e_1 + e_2 + ... + e_k, for some e_i's, but the path can only get shorter if we drop e_2 + ... + e_k
- thus, Inv tells us that p' + e' is also in the queue
- fact that p + e was removed before p' + e' means p + e is shorter
- contradiction

Example #1

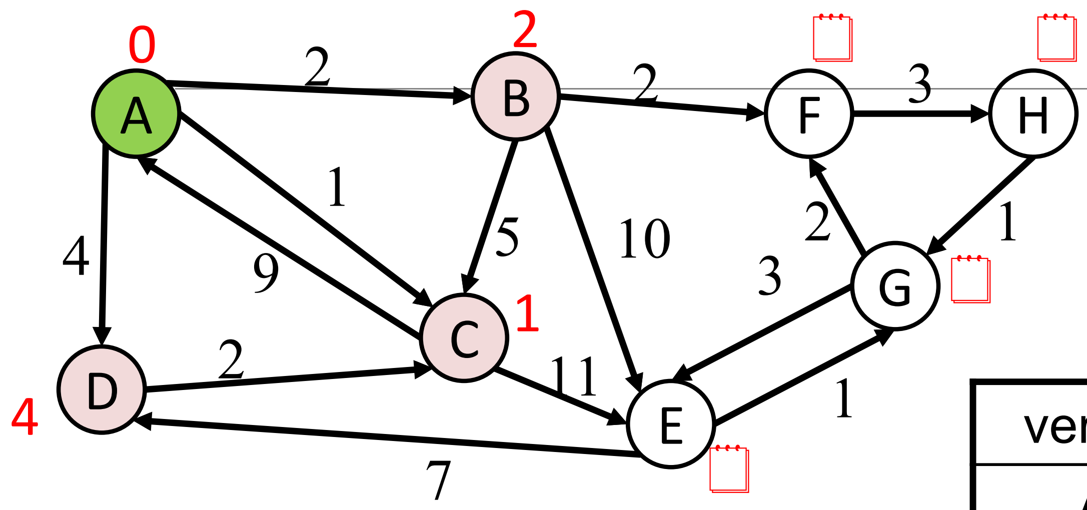


Goal: Fully explore the graph

Order Added to Visited Set:

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

Example #1

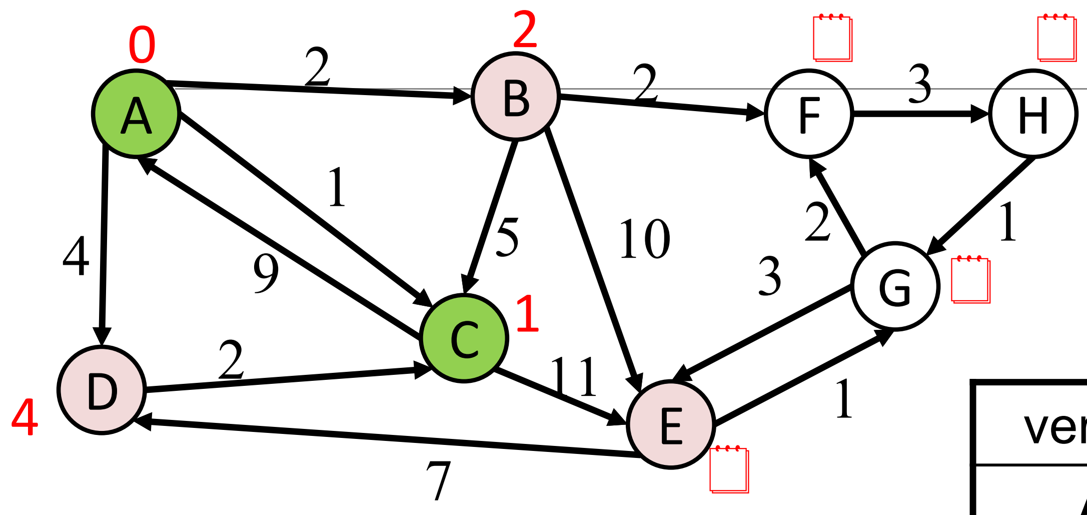


vertex	visited?	cost	path
A	Y	0	
B		≤ 2	A
C		≤ 1	A
D		≤ 4	A
E			
F			
G			
H			

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

A

Example #1

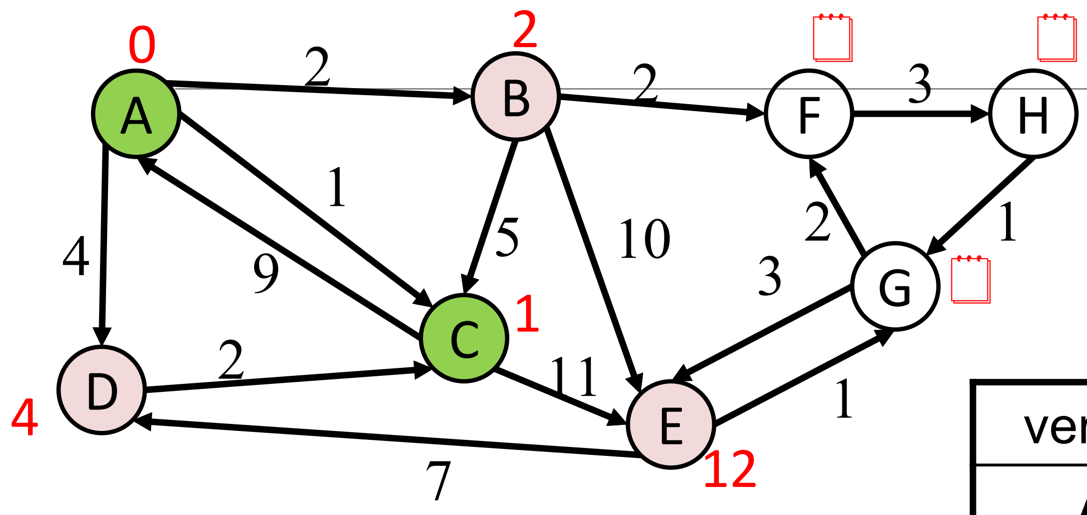


vertex	visited?	cost	path
A	Y	0	
B		≤ 2	A
C	Y	1	A
D		≤ 4	A
E			
F			
G			
H			

Order Added to Visited Set:

A, C

Example #1

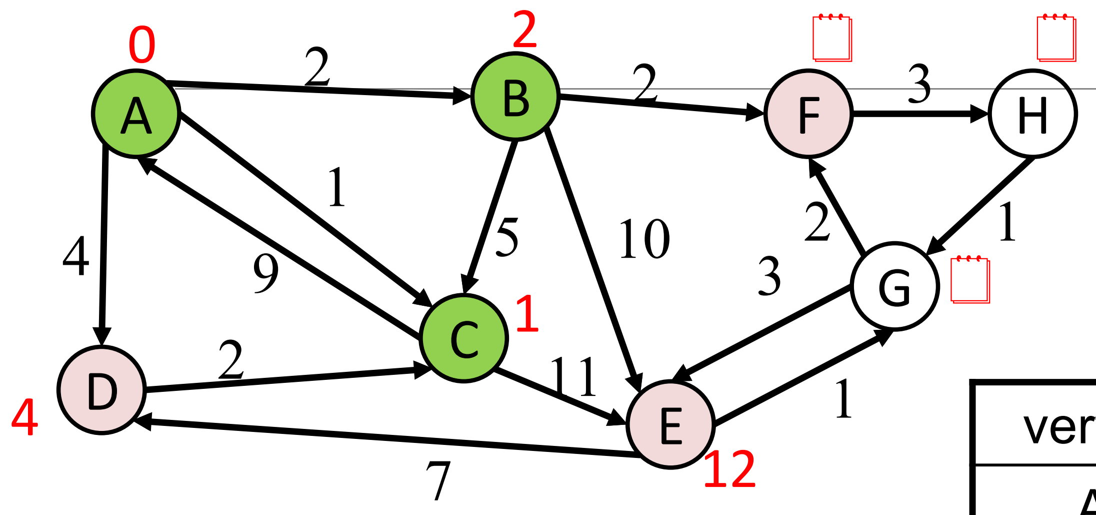


Order Added to Visited Set:

A, C

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

Example #1

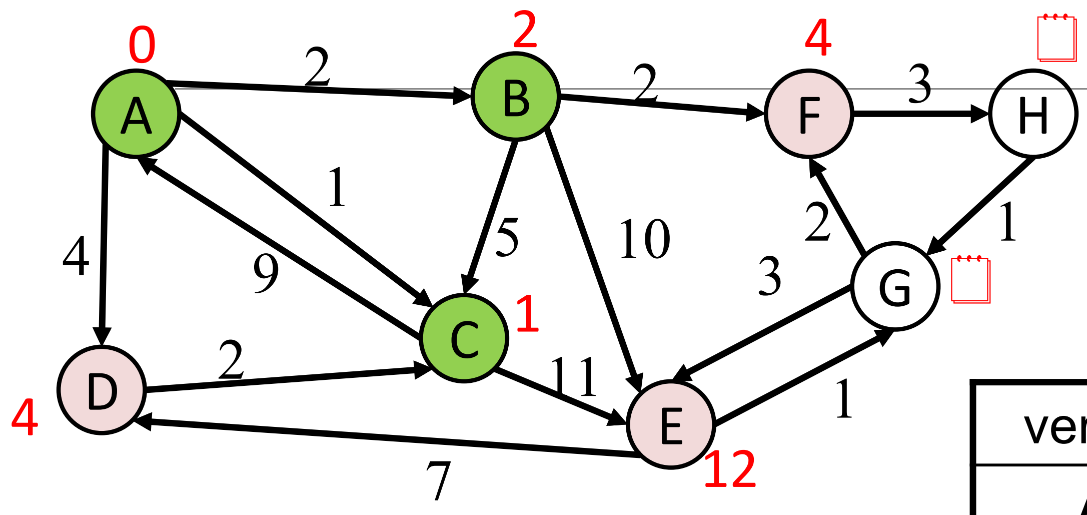


Order Added to Visited Set:

A, C, B

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

Example #1

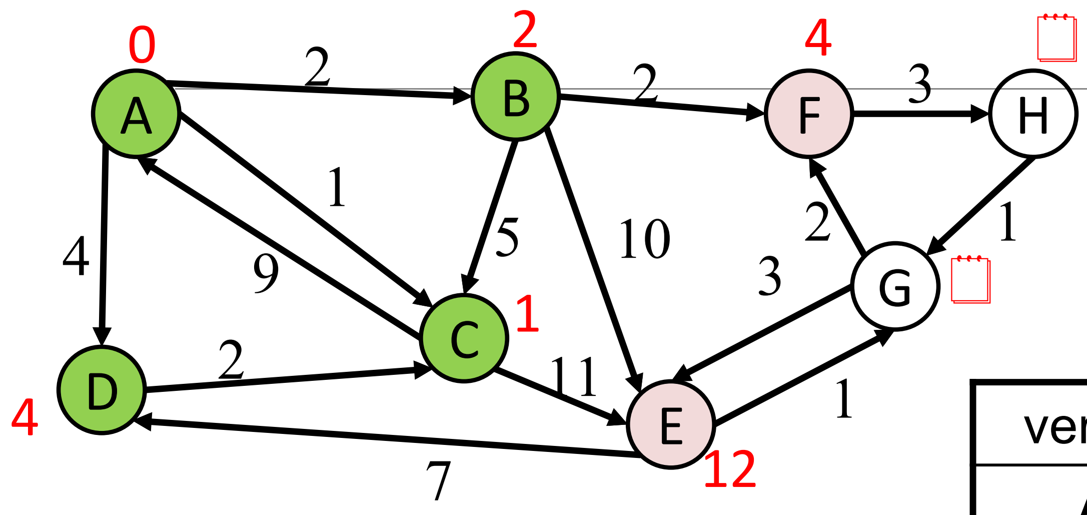


vertex	visited?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D		≤ 4	A
E		≤ 12	C
F		≤ 4	B
G			
H			

Order Added to Visited Set:

A, C, B

Example #1

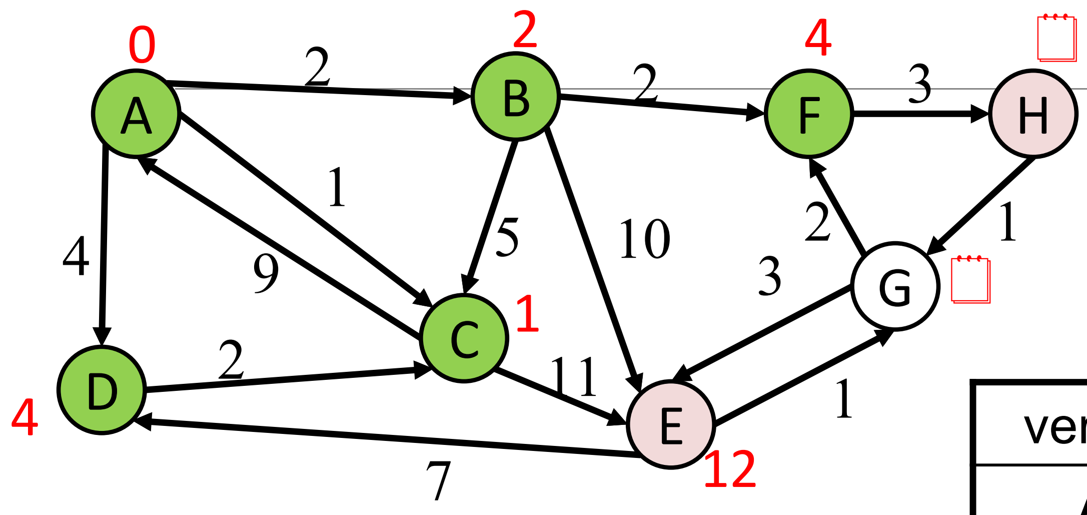


vertex	visited?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 12	C
F		≤ 4	B
G			
H			

Order Added to Visited Set:

A, C, B, D

Example #1

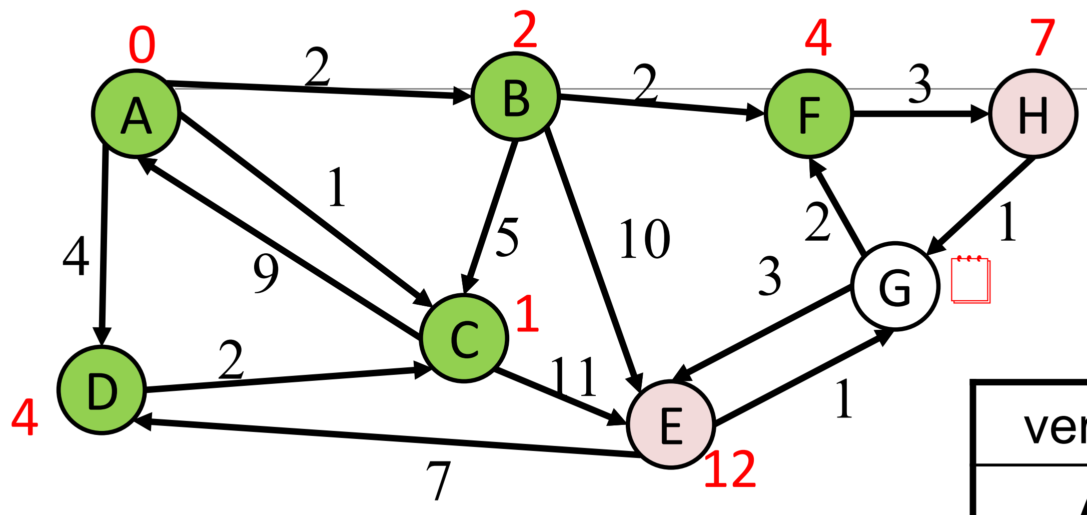


vertex	visited?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 12	C
F	Y	4	B
G			
H			

Order Added to Visited Set:

A, C, B, D, F

Example #1

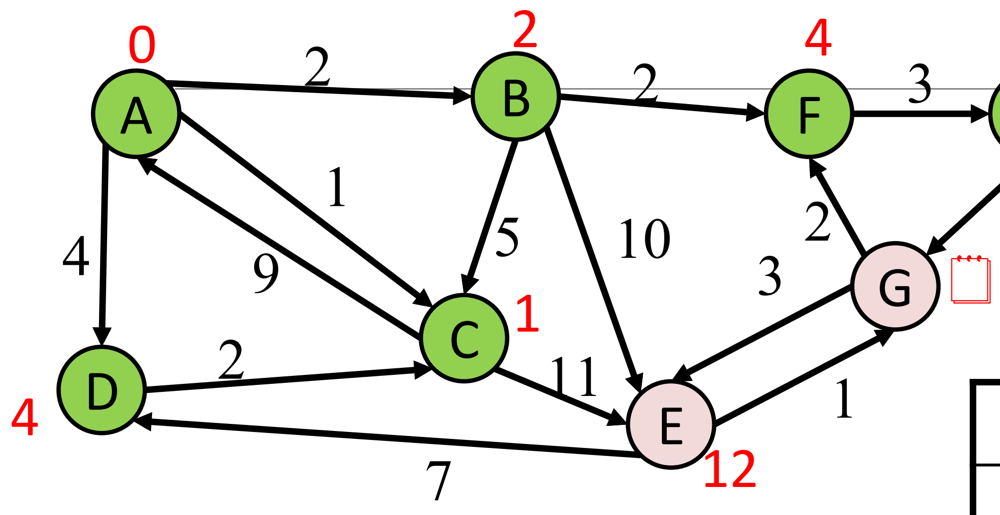


vertex	visited?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 12	C
F	Y	4	B
G			
H		≤ 7	F

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

A, C, B, D, F

Example #1

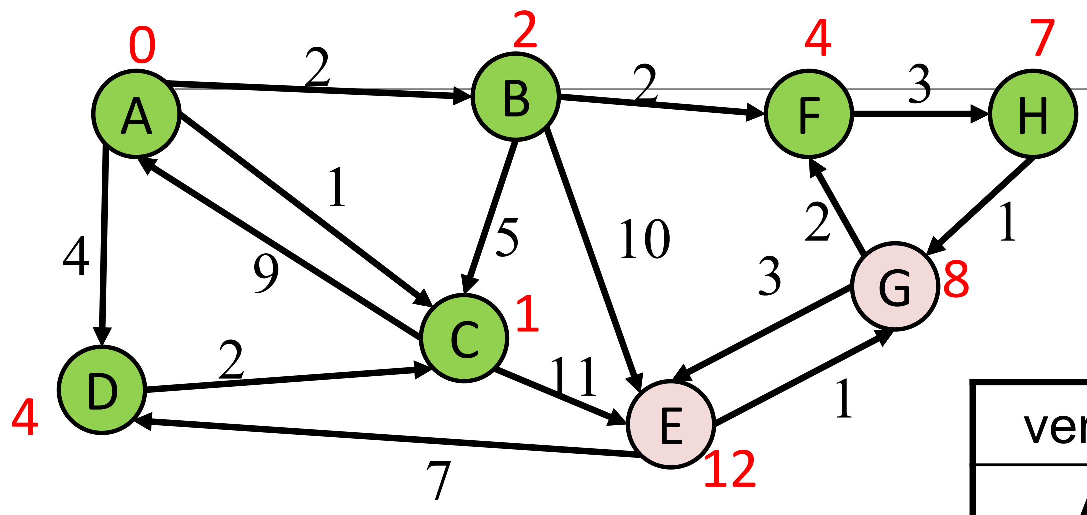


vertex	visited?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 12	C
F	Y	4	B
G			
H	Y	7	F

Order Added to Visited Set:

A, C, B, D, F, H

Example #1

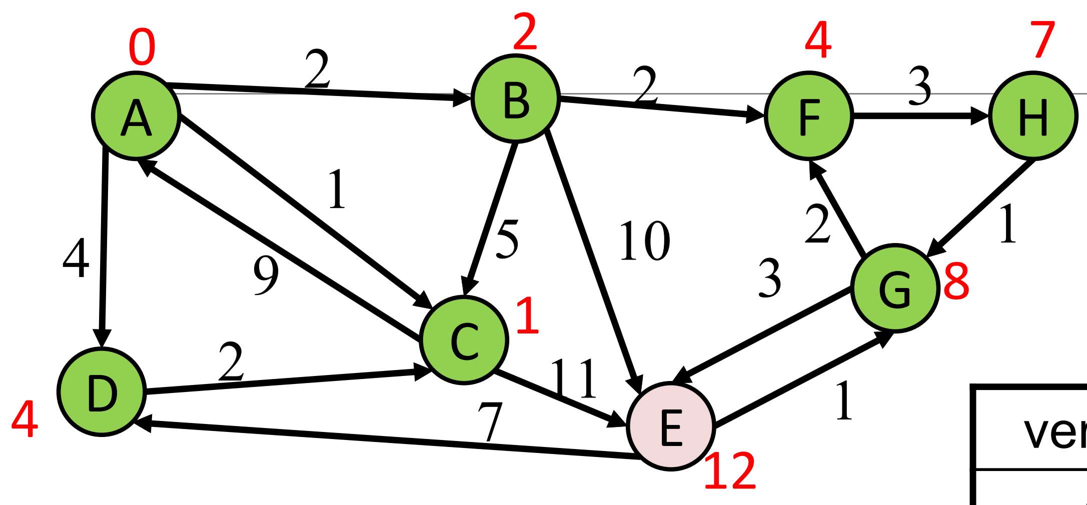


vertex	visited?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 12	C
F	Y	4	B
G		≤ 8	H
H	Y	7	F

Order Added to Visited Set:

A, C, B, D, F, H

Example #1

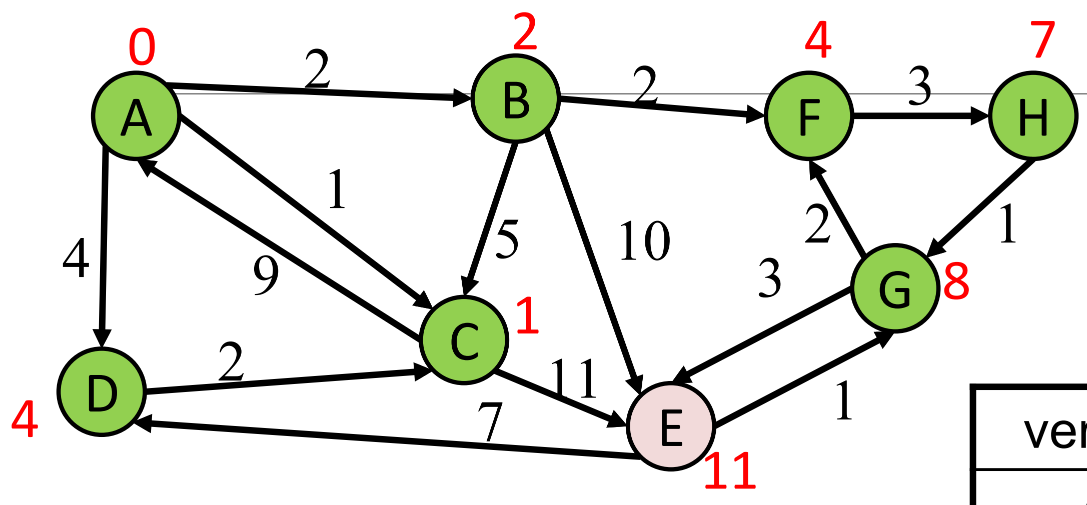


vertex	visited?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 12	C
F	Y	4	B
G	Y	8	H
H	Y	7	F

Order Added to Visited Set:

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

Example #1

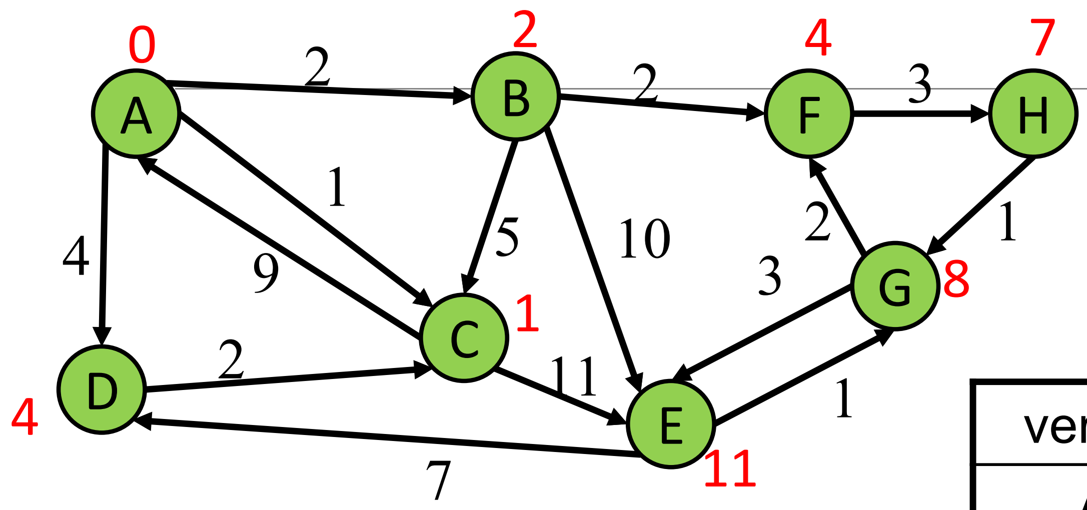


Order Added to Visited Set:

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

vertex	visited?	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

Example #1

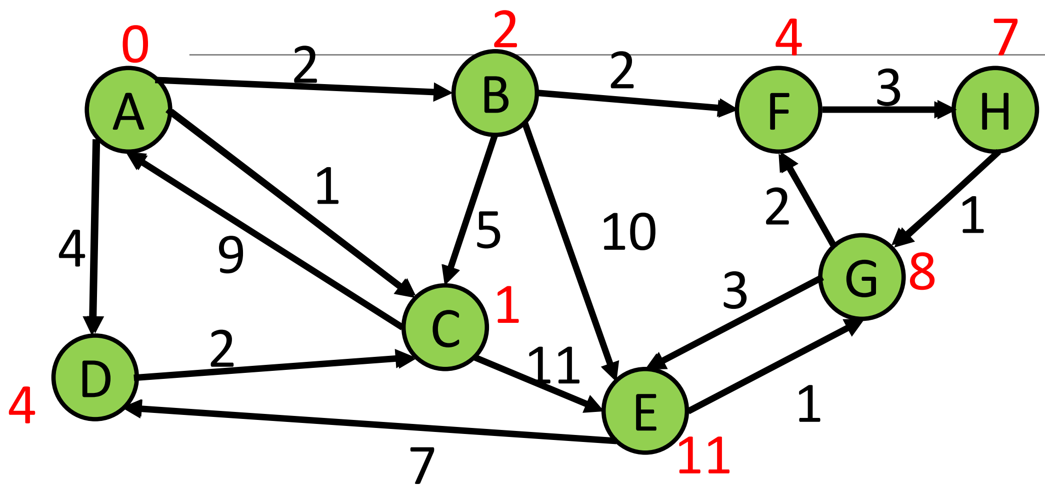


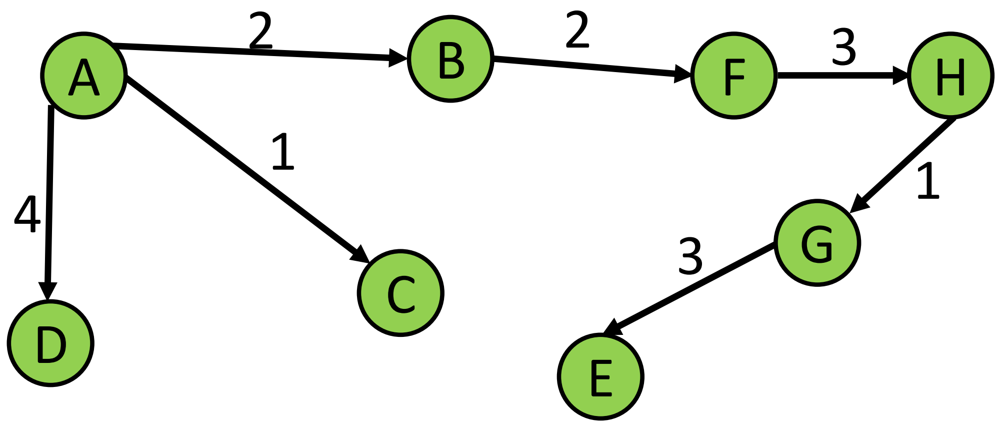
Order Added to Visited Set:

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

vertex	visited?	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

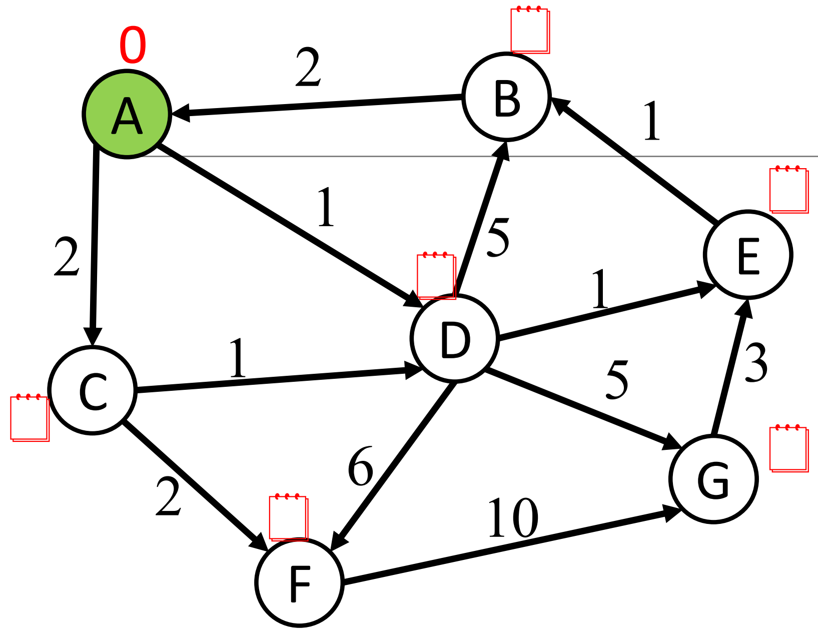
Interpreting the Results





vertex	visited?	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

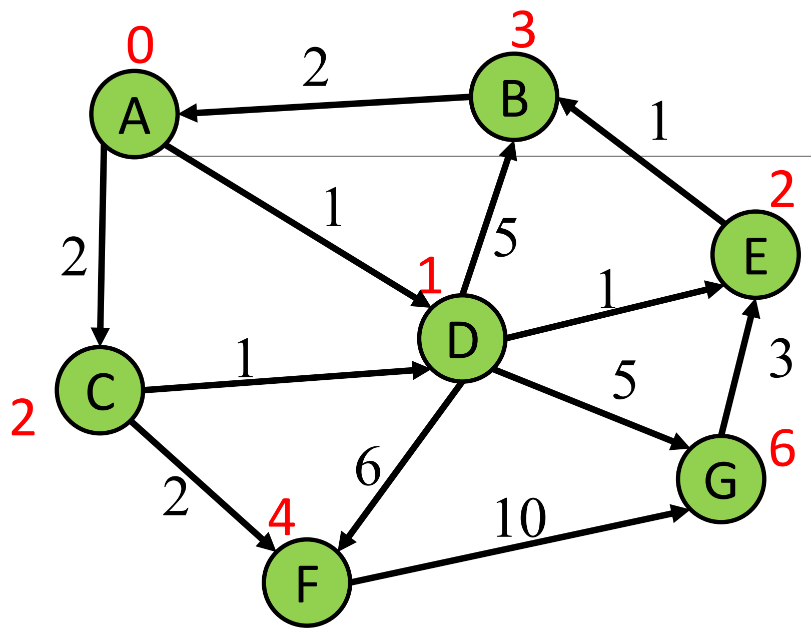
Example #2



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

Order Added to Visited Set:

Example #2



vertex	visited?	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 Visited Set:</u>

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