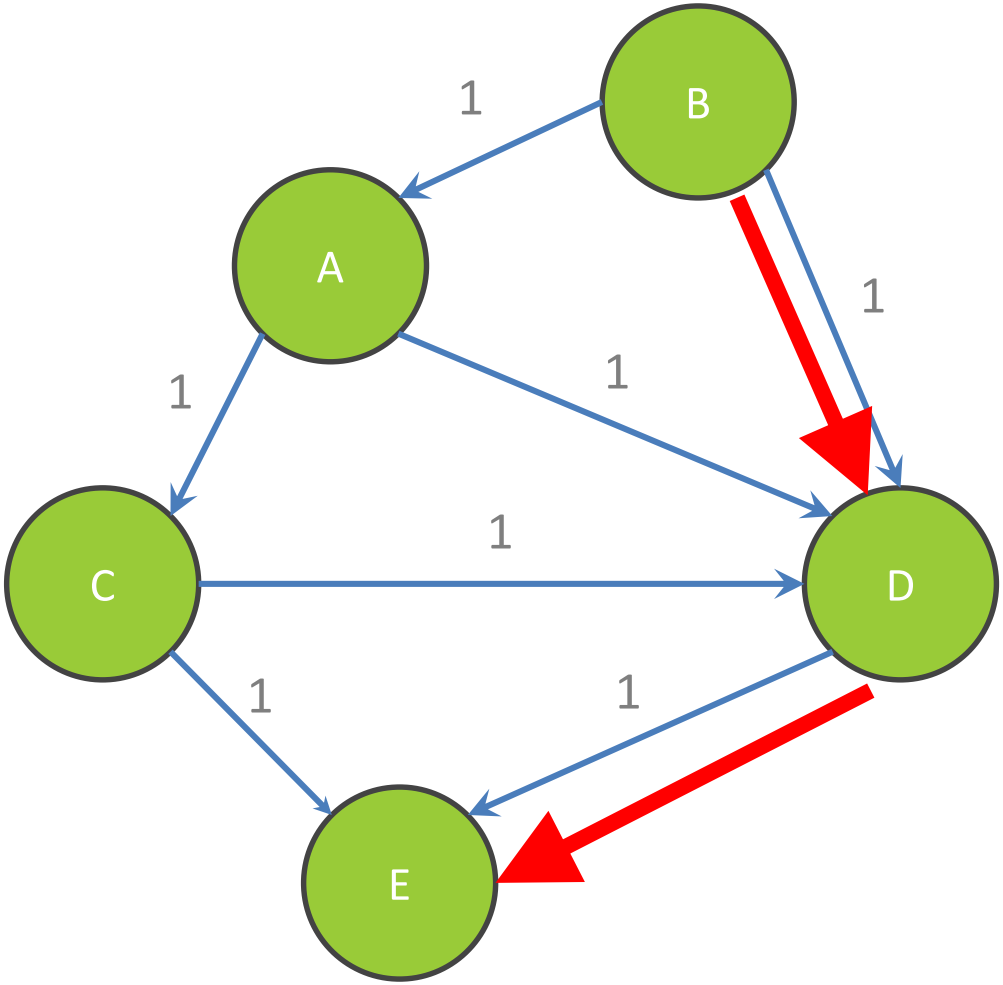
Section 7:
Dijkstra’s Algorithm

Slides adapted from Alex Mariakakis with material by Kellen Donohue, David Mailhot, Dan Grossman, Mike Ernst, Michael Hart, and Jacob Murphy

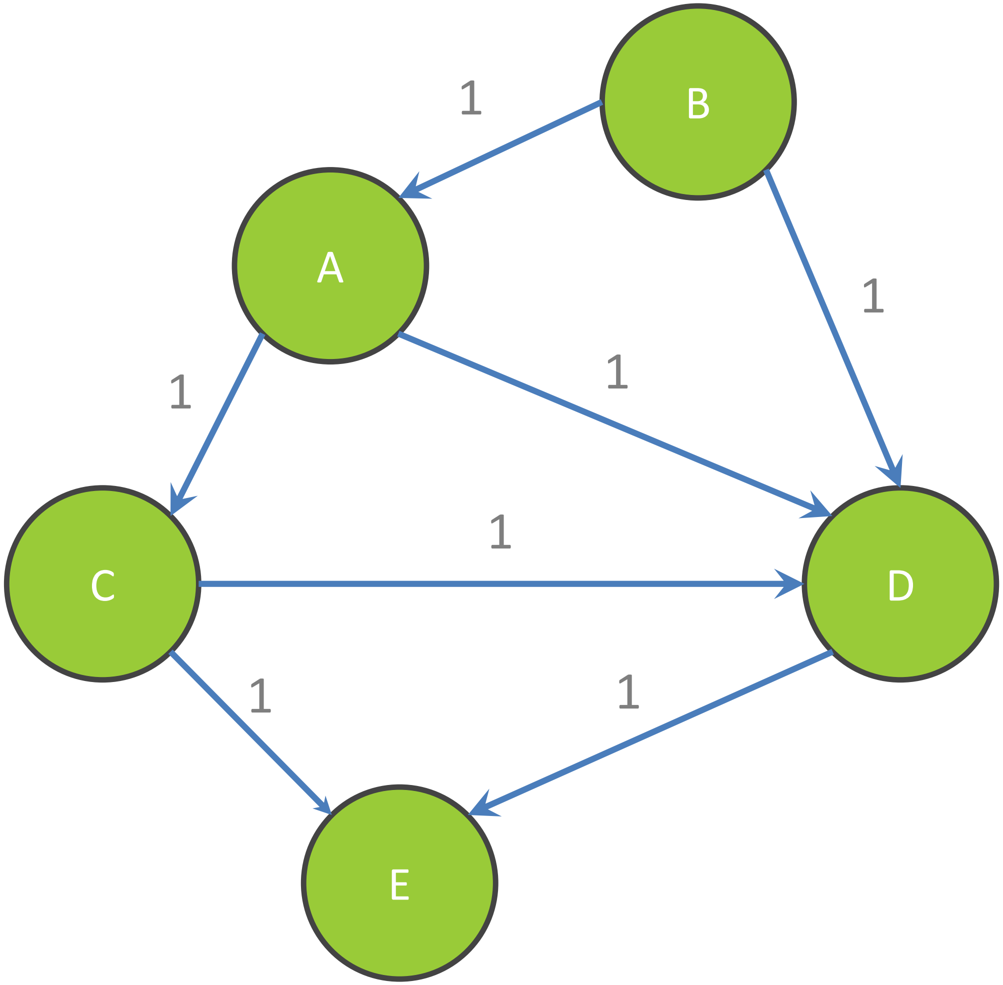
Review: Shortest Paths with BFS



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

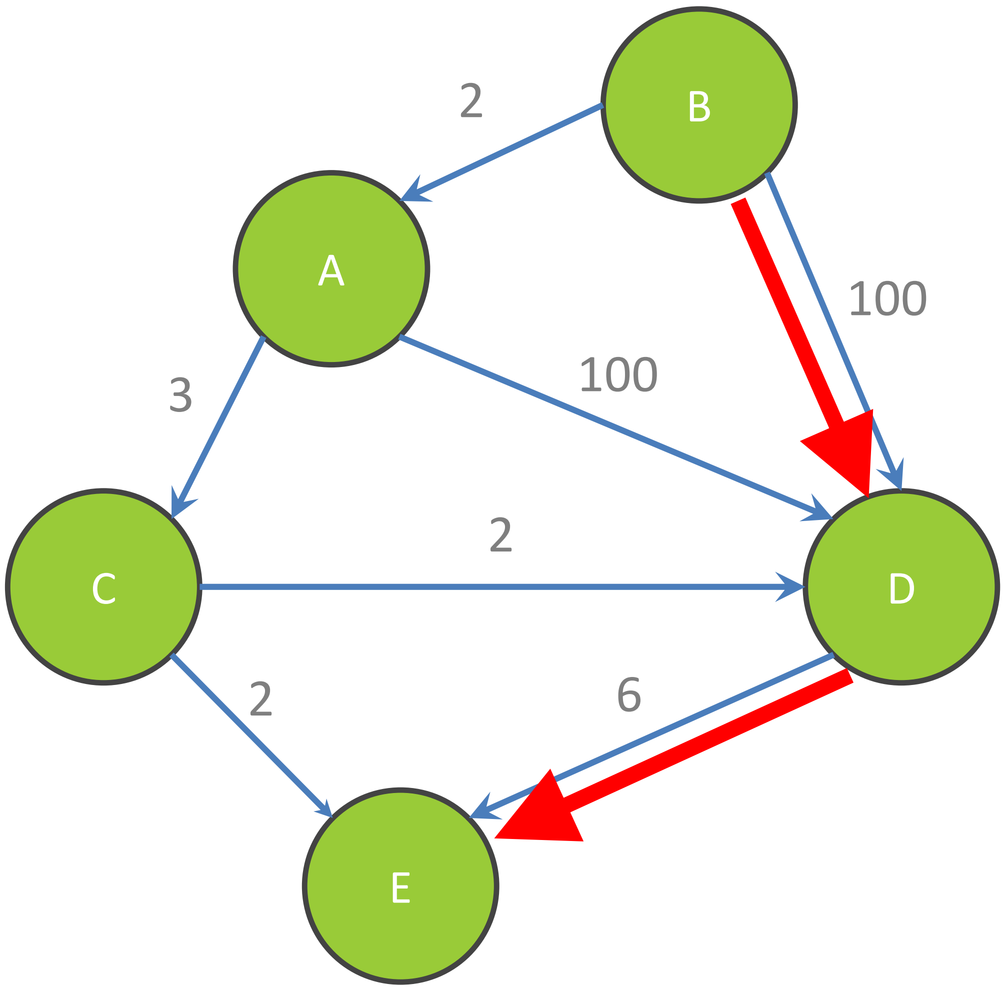
Review: Shortest Paths with BFS



From Node B

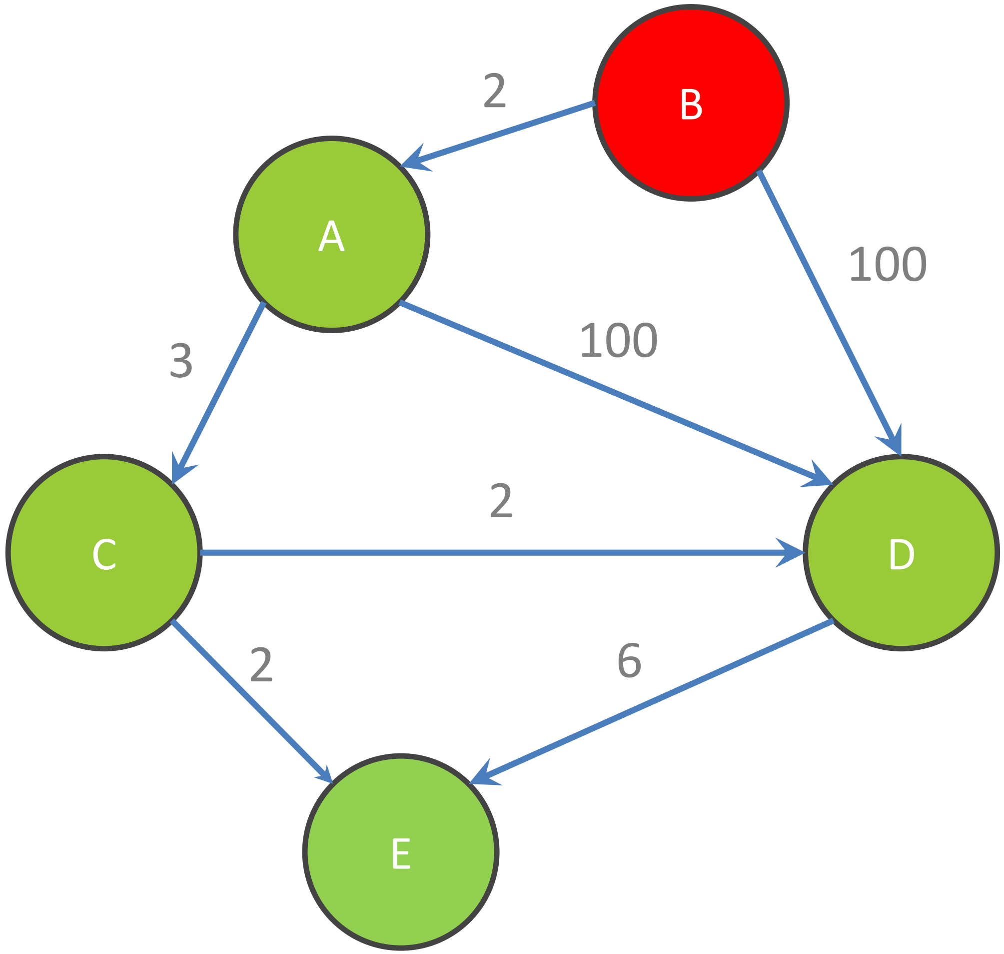
Destination	Path	Cost
A	<B,A>	1
B	<B>	0
C	<B,A,C>	2
D	<B,D>	1
E	<B,D,E>	2

Shortest Paths with Weights

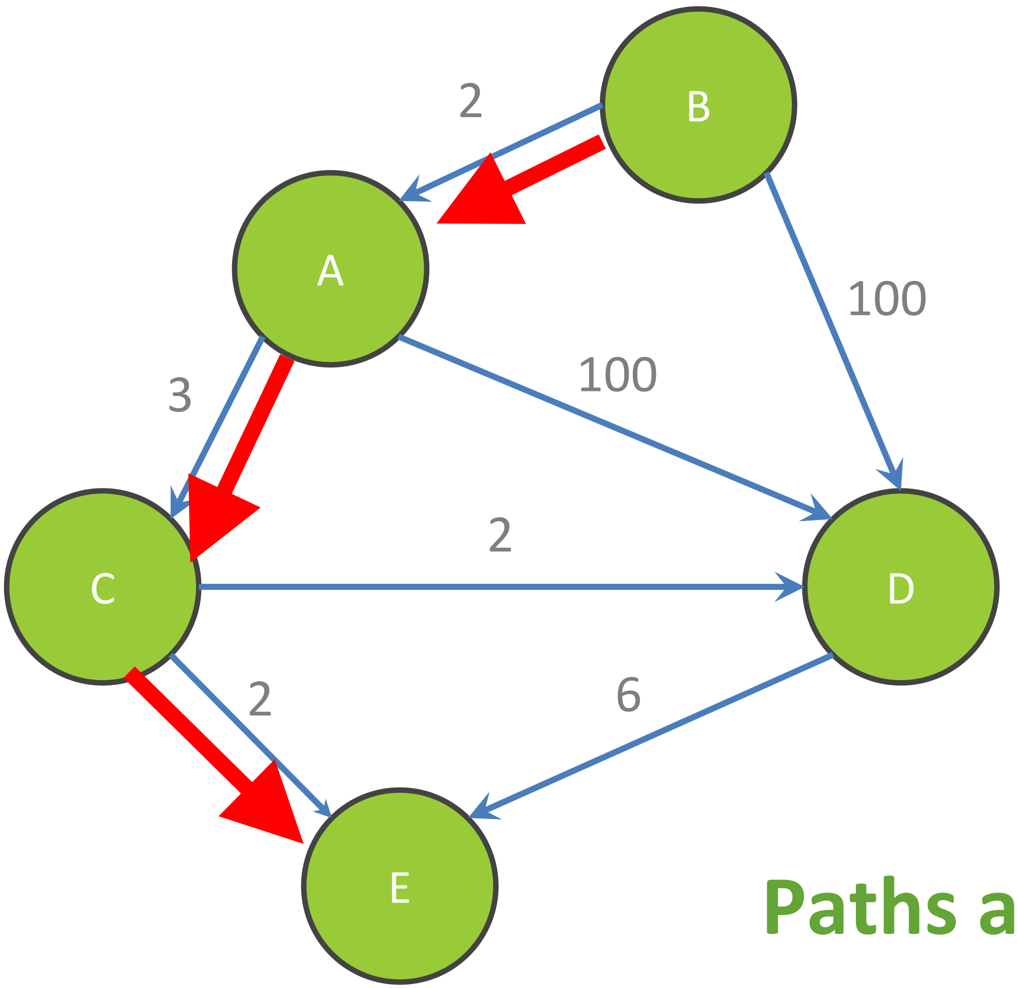


B -> E = 106?

How can we find the shortest path with weights?



Shortest Paths with Weights

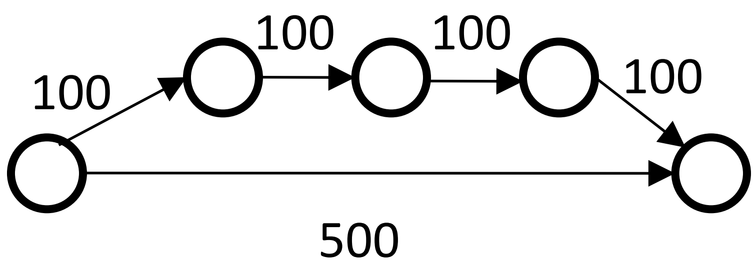


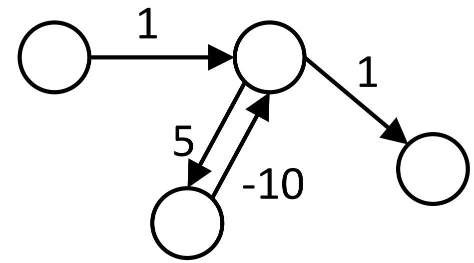
From Node B

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 Algorithm





BFS can find the most direct path, but not necessarily the shortest path!

Note that Dijkstra's Algorithm only works if there is not a negative cycle

Dijkstra’s Algorithm

Named after its inventor Edsger Dijkstra (1930-2002)

- Among his contributions to the growing CS community was his work on Operating Systems, in which he motivated the design and structure of a large project, not just the code

The Algorithm: similar to BFS, but incorporating weights

- Create a set of nodes to examine next, but instead of using the node that was next in line, use the node with the shortest distance
- How can you find the node with the shortest distance so far?

  Priority Queues (explained later)

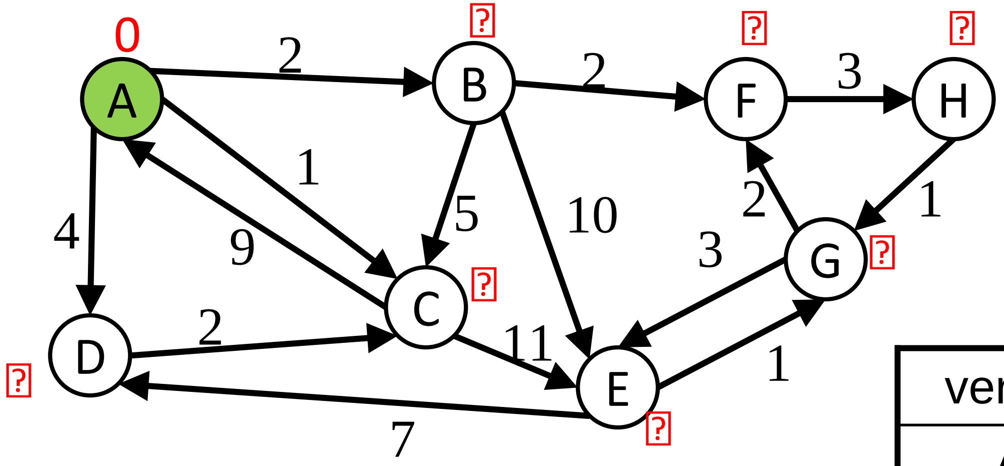
Dijkstra's Algorithm

Give a node two fields: `cost` and `finished`, `cost` gives an upper bound to the distance from the origin to that node, and `finished` describing if the cost of the node is the minimum cost of travelling to that node.

1. For each node `v`, set `v.cost = ∞` and `v.finished = false`
2. Set `source.cost = 0` (source is the starting node of our path)
3. While there are unknown nodes in the graph
   a) Select the unknown node `v` with lowest cost
   b) Mark `v` as finalized
   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 // add v to the nodes u has traversed
```

Example #1

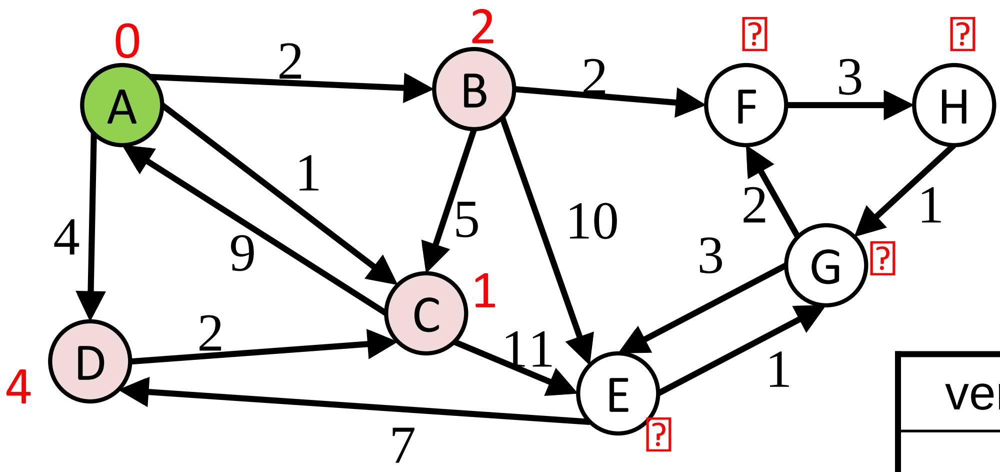


Goal: Find the best paths from A to the other nodes

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

Order Added to Known Set:

Example #1

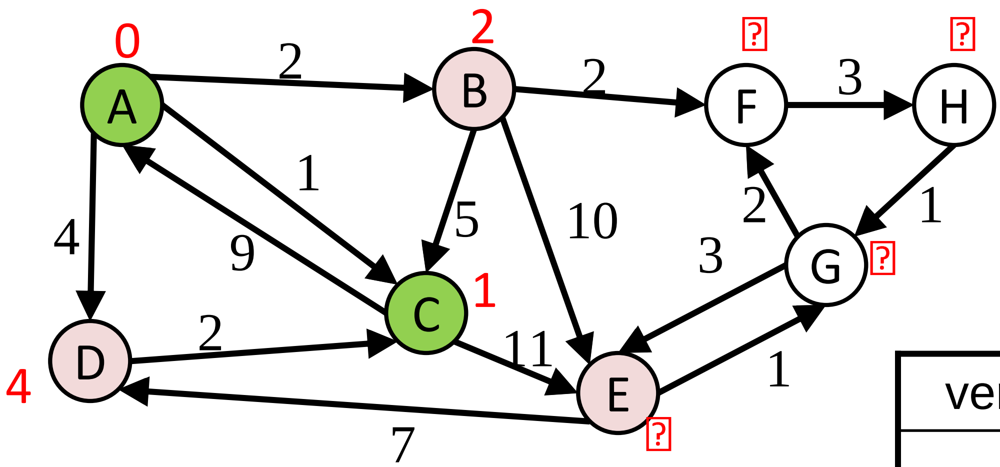


vertex	known?	cost	path
A	Y	0	
B		≤ 2	A
C		≤ 1	A
D		≤ 4	A
E		∞	
F		∞	
G		∞	
H		∞	

Order Added to Known Set:

A

Example #1

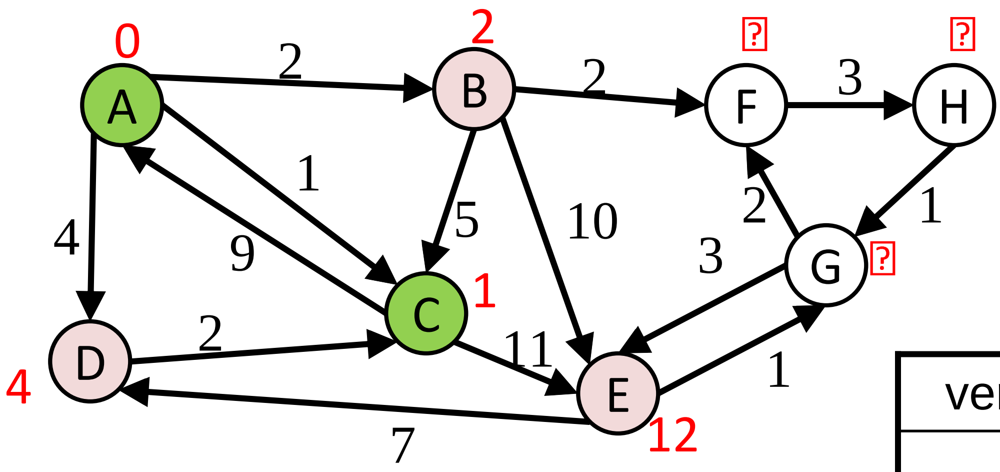


Order Added to Known Set:

A, C

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

Example #1

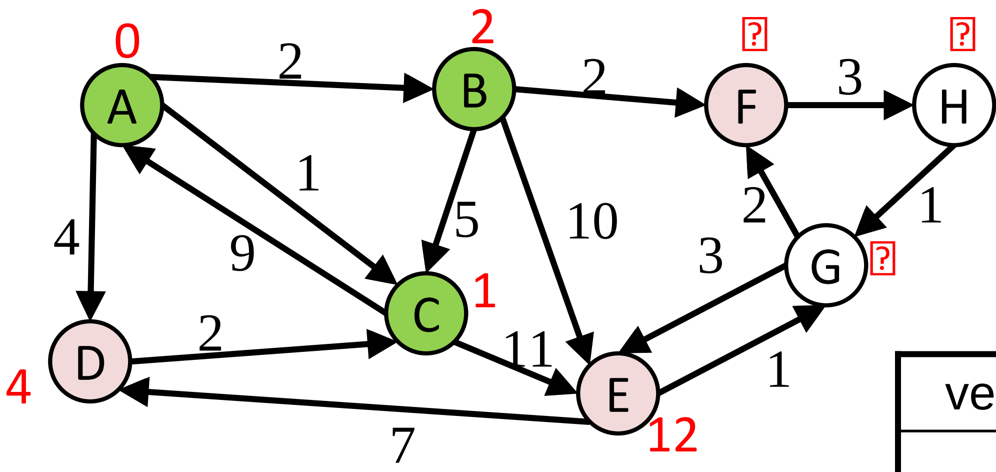


vertex	known?	cost	path
A	Y	0	
B		≤ 2	A
C	Y	1	A
D		≤ 4	A
E		≤ 12	C
F		∞	
G		∞	
H		∞	

Order Added to Known Set:

A, C

Example #1

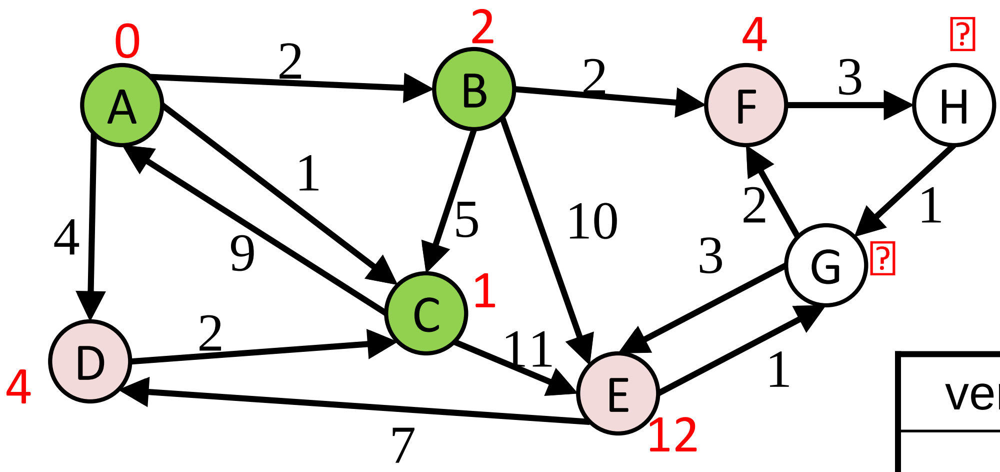


vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D		≤ 4	A
E		≤ 12	C
F		∞	
G		∞	
H		∞	

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

A, C, B

Example #1

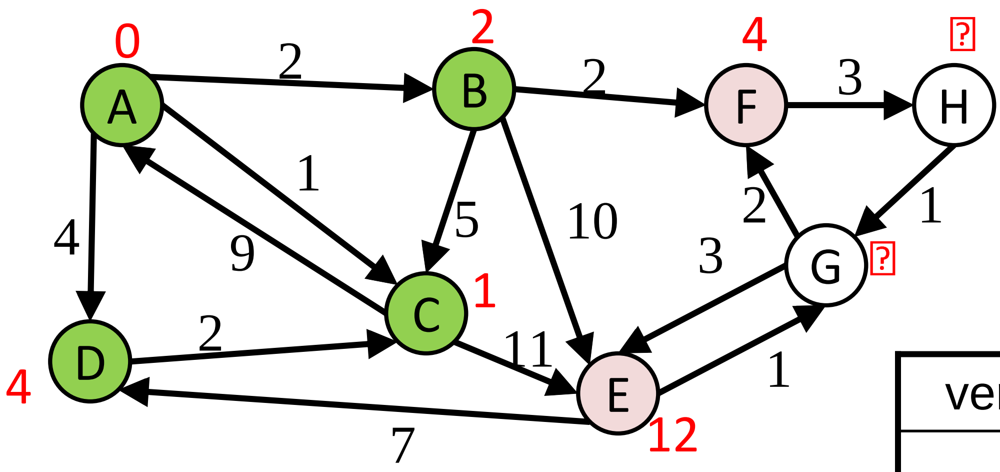


vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D		≤ 4	A
E		≤ 12	C
F		≤ 4	B
G		∞	
H		∞	

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

A, C, B

Example #1

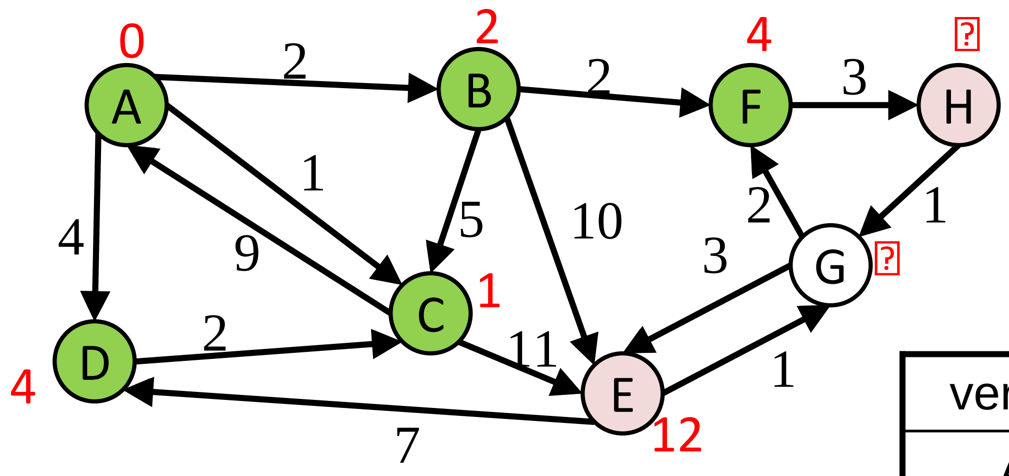


vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 12	C
F		≤ 4	B
G		∞	
H		∞	

Order Added to Known Set:

A, C, B, D

Example #1

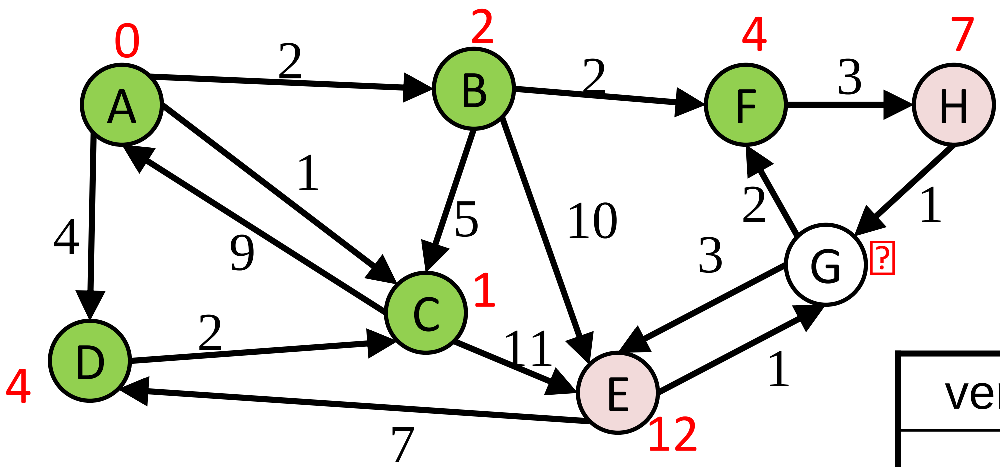


vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 12	C
F	Y	4	B
G		∞	
H		∞	

Order Added to Known Set:

A, C, B, D, F

Example #1

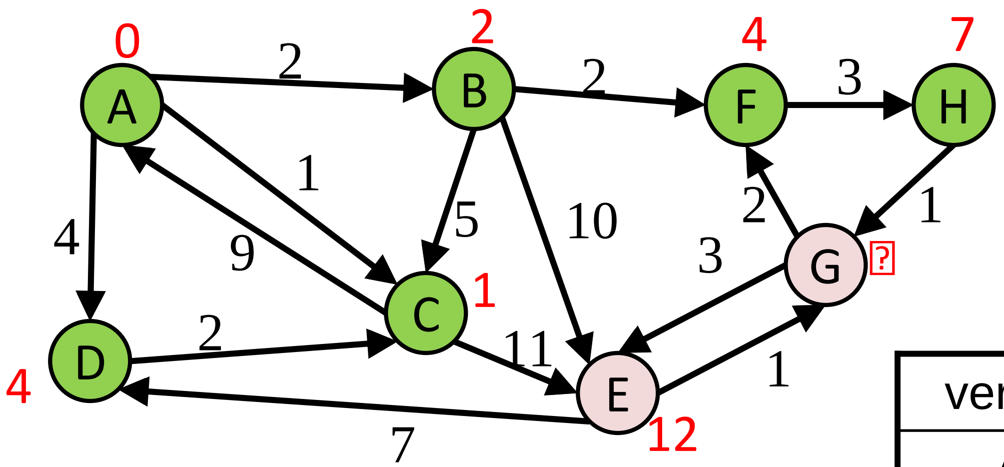


vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 12	C
F	Y	4	B
G		∞	
H		≤ 7	F

Order Added to Known Set:

A, C, B, D, F

Example #1

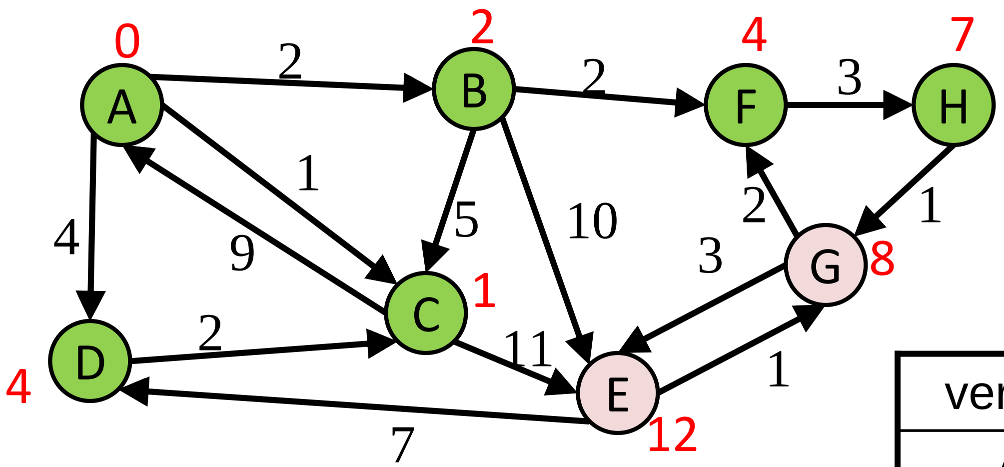


vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 12	C
F	Y	4	B
G		∞	
H	Y	7	F

Order Added to Known Set:

A, C, B, D, F, H

Example #1

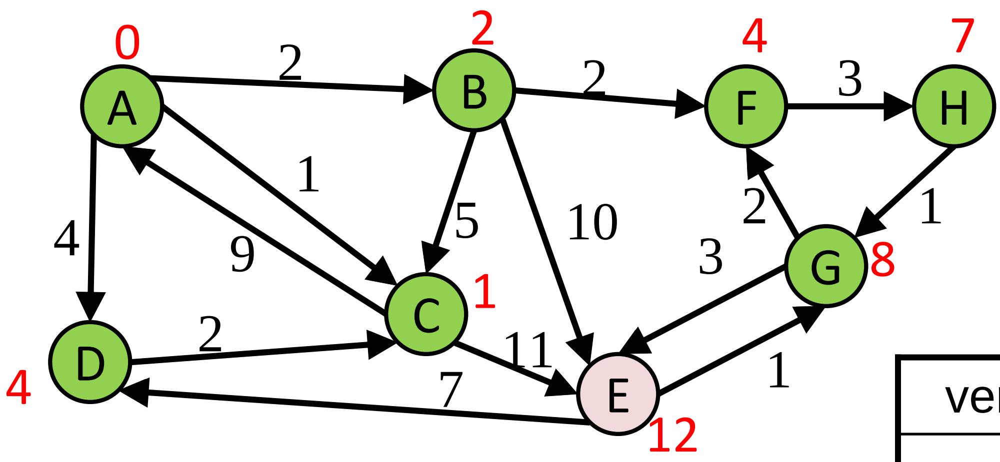


Order Added to Known Set:

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

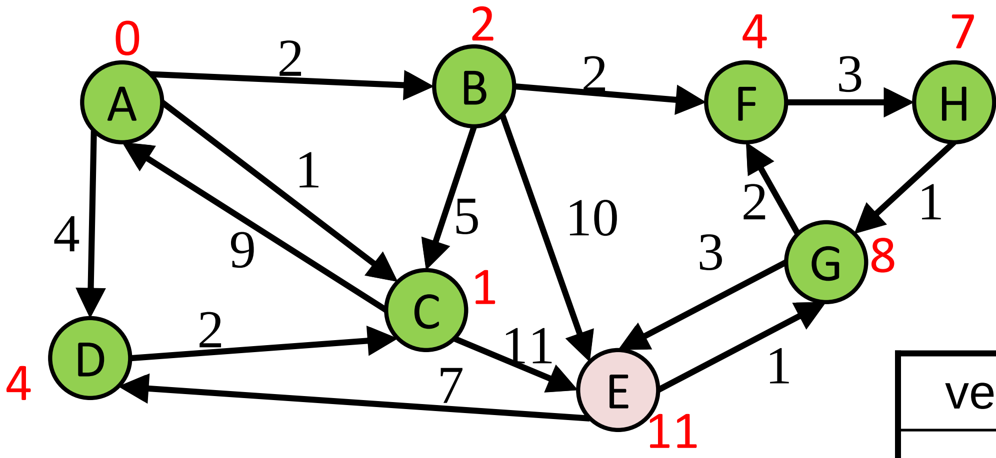


vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 12	C
F	Y	4	B
G	Y	8	H
H	Y	7	F

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

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

Example #1

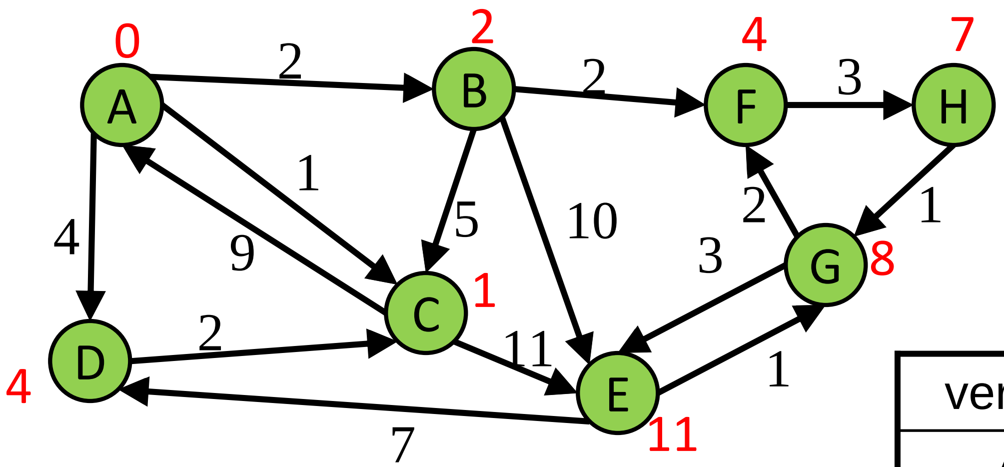


vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 11	G
F	Y	4	B
G	Y	8	H
H	Y	7	F

Order Added to Known Set:

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

Example #1

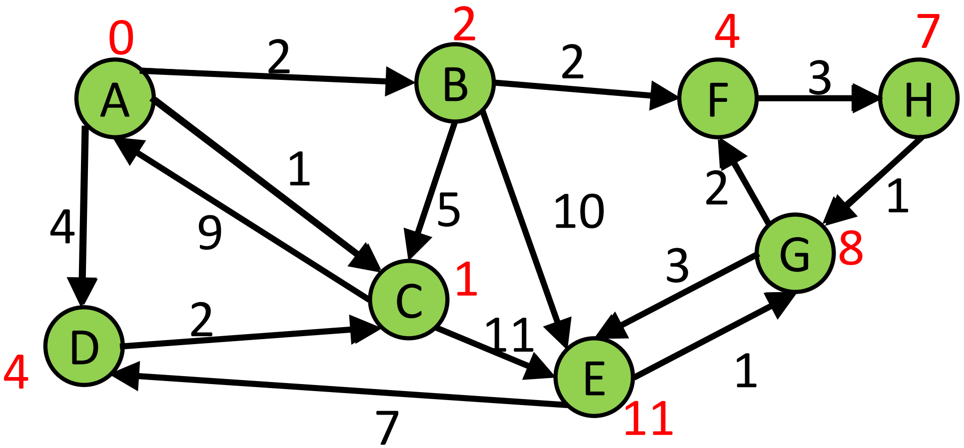


vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E	Y	11	G
F	Y	4	B
G	Y	8	H
H	Y	7	F

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

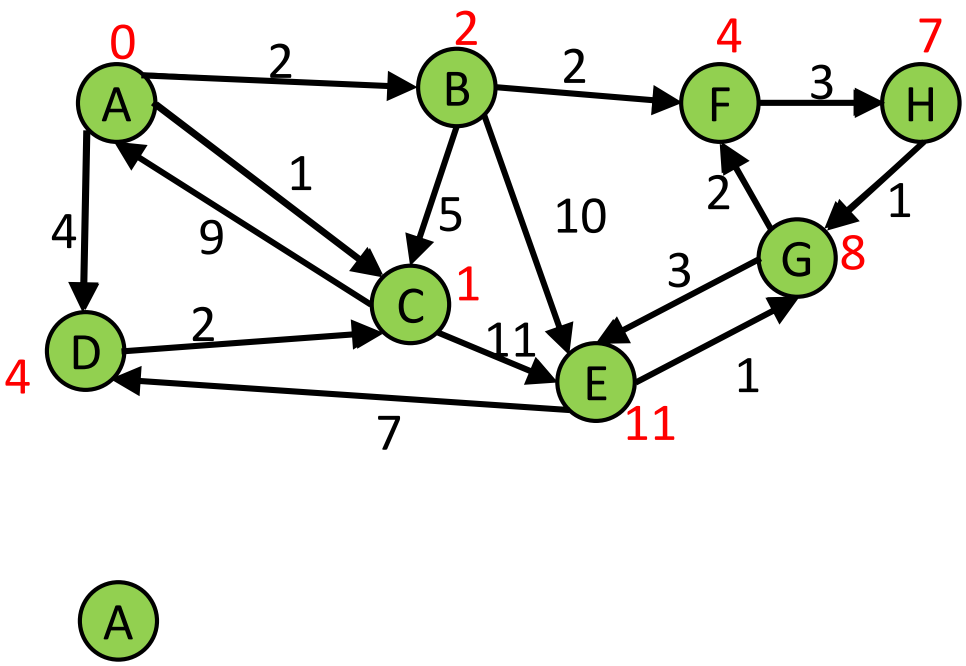
A, C, B, D, F, H, G, E

Interpreting the Results



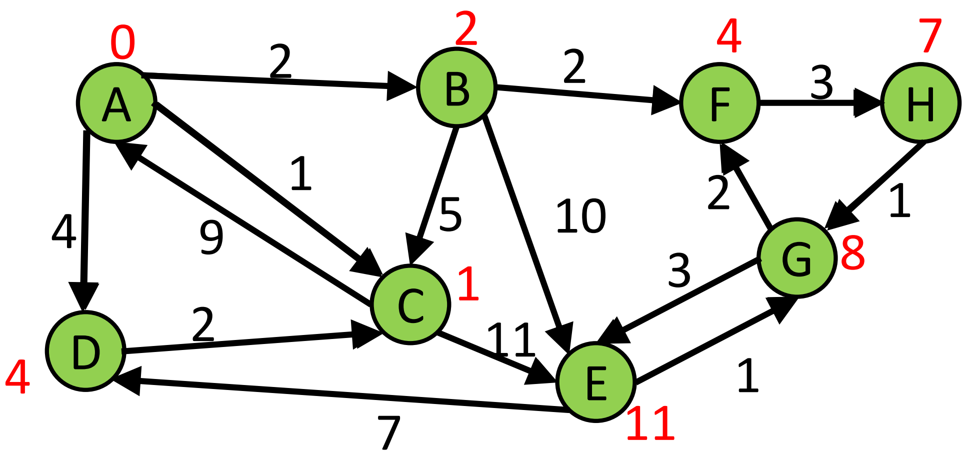
vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E	Y	11	G
F	Y	4	B
G	Y	8	H
H	Y	7	F

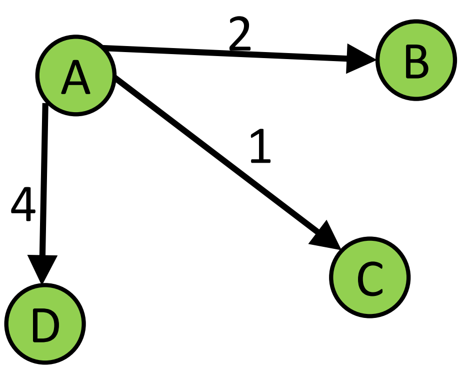
Interpreting the Results



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

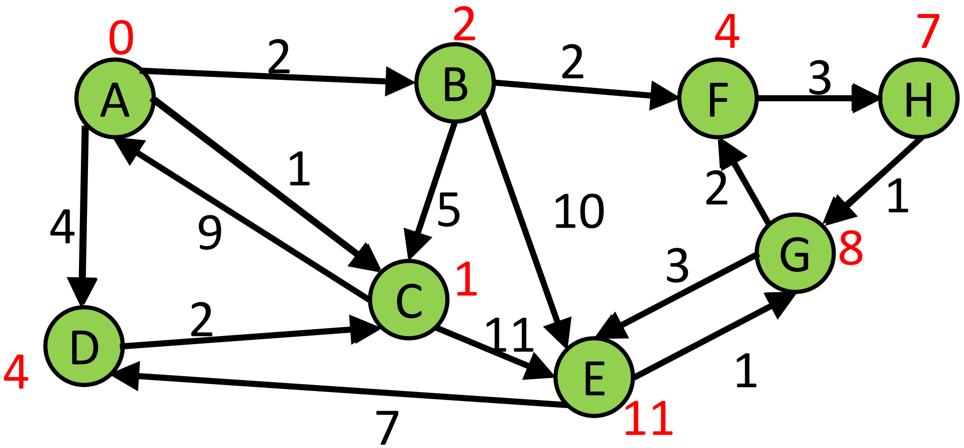
Interpreting the Results



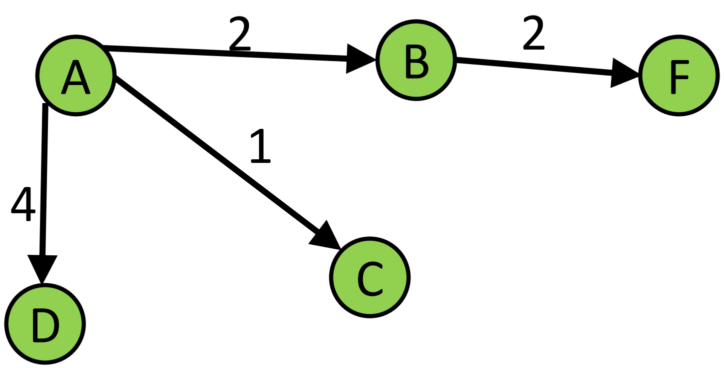


vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E	Y	11	G
F	Y	4	B
G	Y	8	H
H	Y	7	F

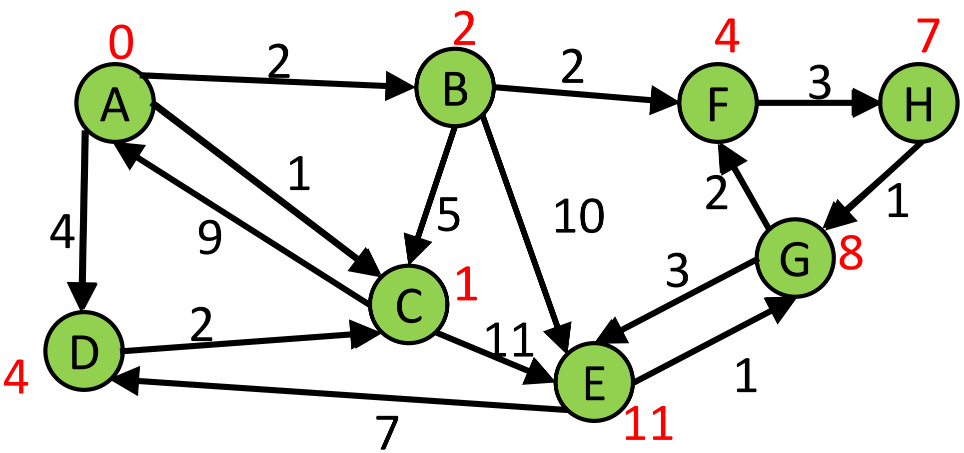
Interpreting the Results

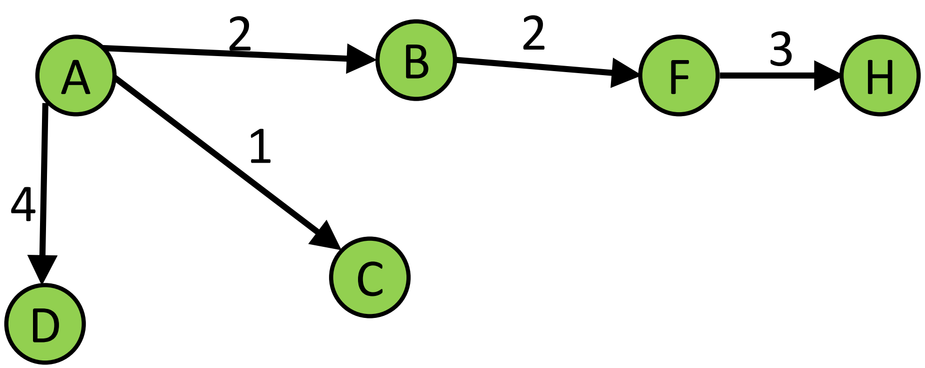


vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E	Y	11	G
F	Y	4	B
G	Y	8	H
H	Y	7	F



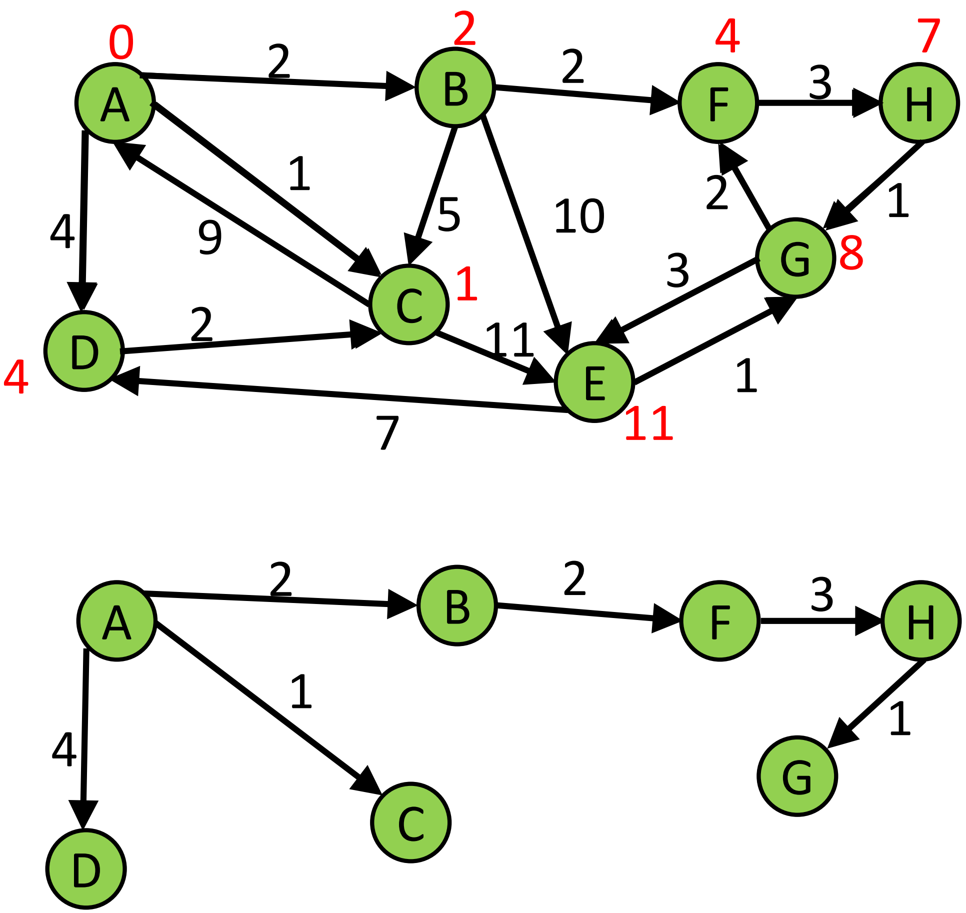
Interpreting the Results





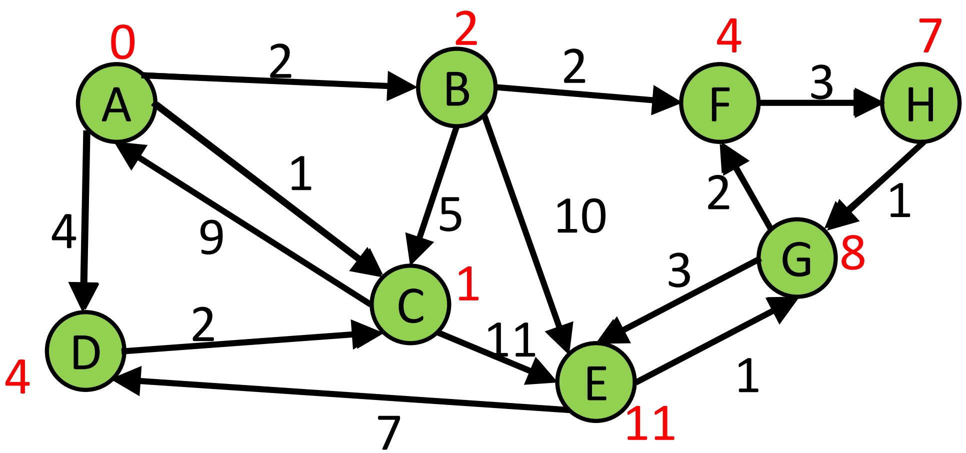
vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E	Y	11	G
F	Y	4	B
G	Y	8	H
H	Y	7	F

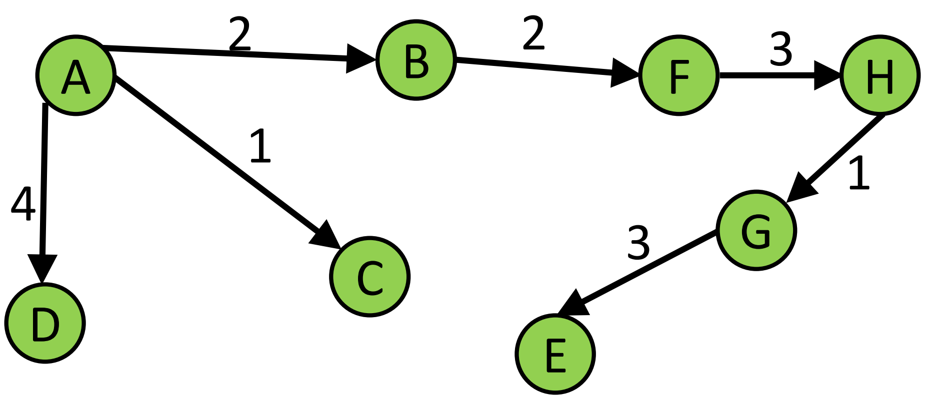
Interpreting the Results



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

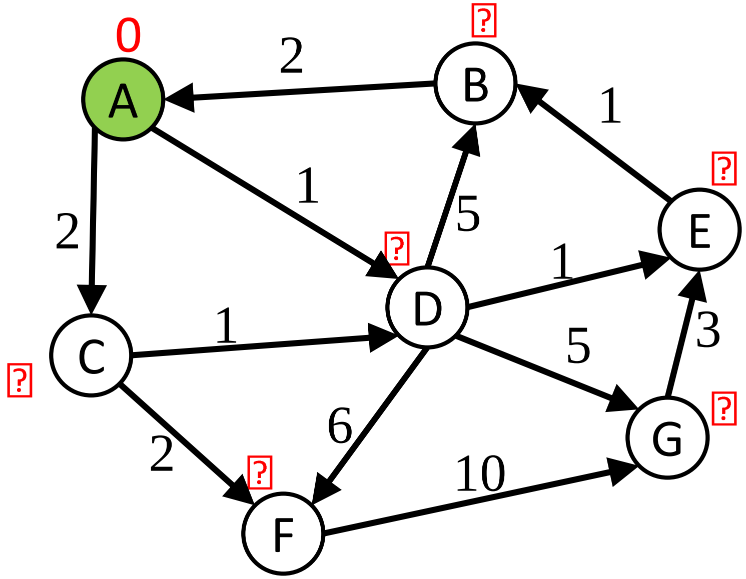
Interpreting the Results





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

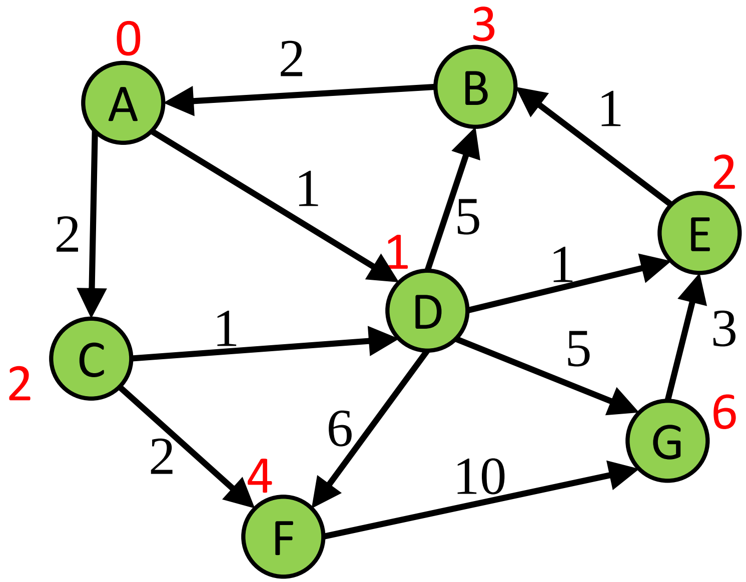
Example #2



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

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

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

Given a set of weighted paths, find the shortest path

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())