Section 7:
Dijkstra's

JUSTIN BARE AND DERIC PANG

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

Things to Discuss

Late days

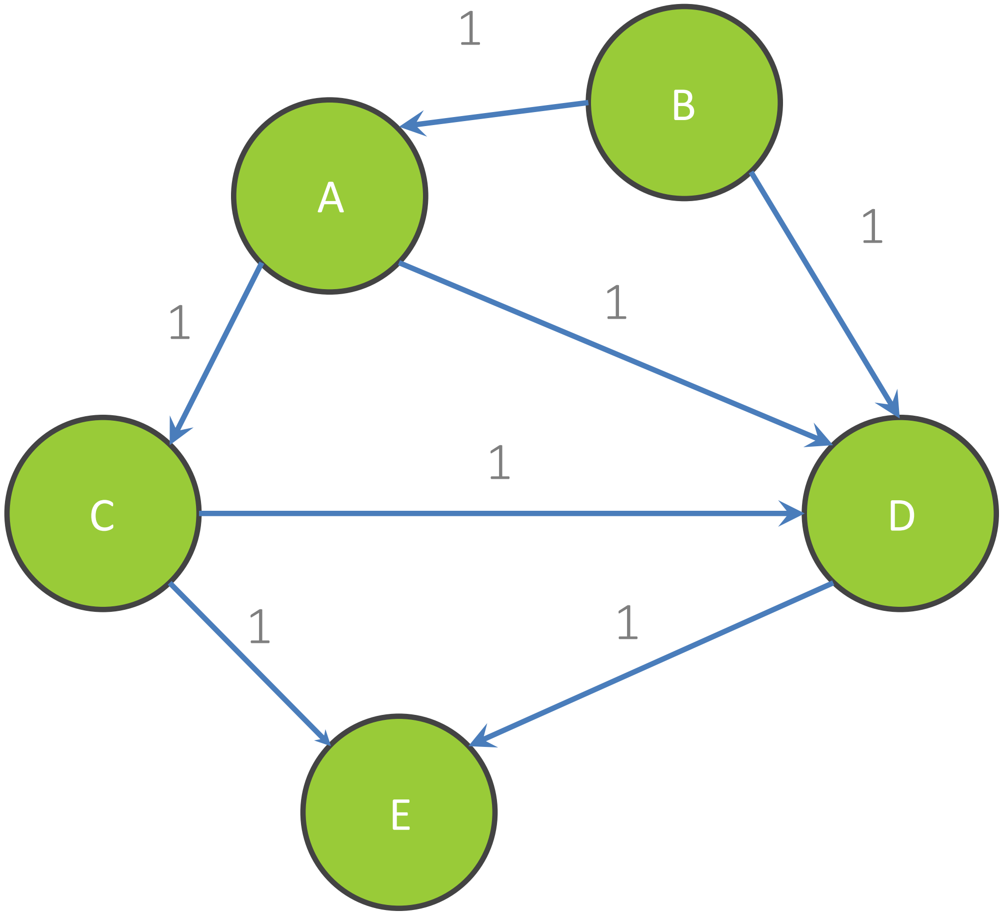
- 4 assignments left, including Homework 6
- Late Days are marked in Catalyst

Homework 5 grades are published. Woot!

Hints from a Homework-Grader:

- Answers.txt – Hint:
  - “Name: <replace this with your name>”
- Abstraction functions and representation invariants
  - Understand what they are and the difference between them
- JavaDoc Comments – Should contain a general overview of the method
- Minimize the number of asserts per test
  - One domain per test – a specific functionality and input

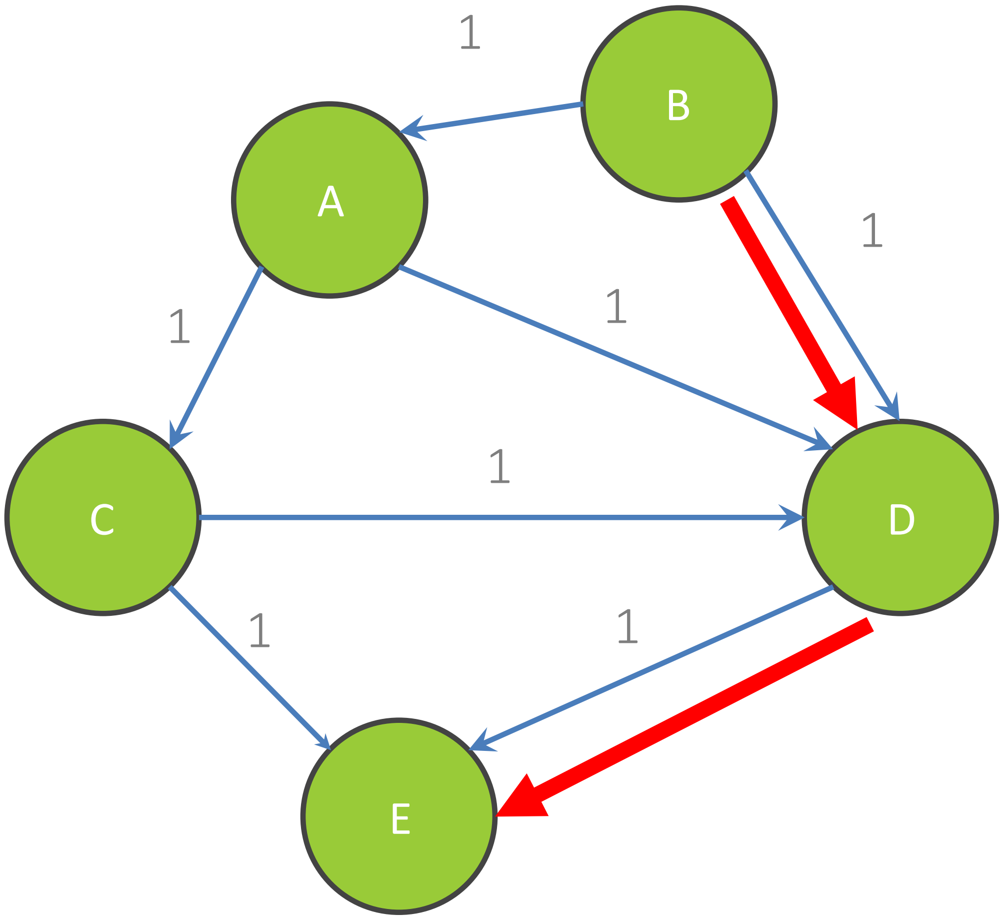
Review: Shortest Paths with BFS



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

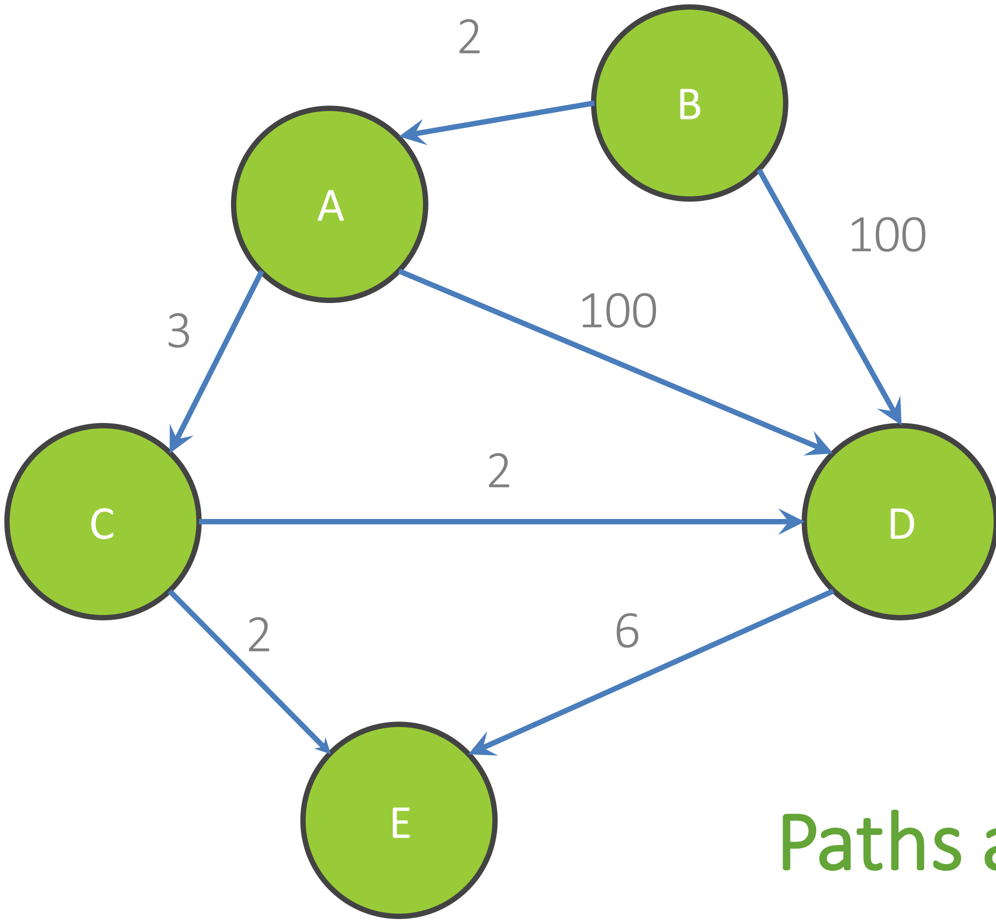
Review: Shortest Paths with BFS



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

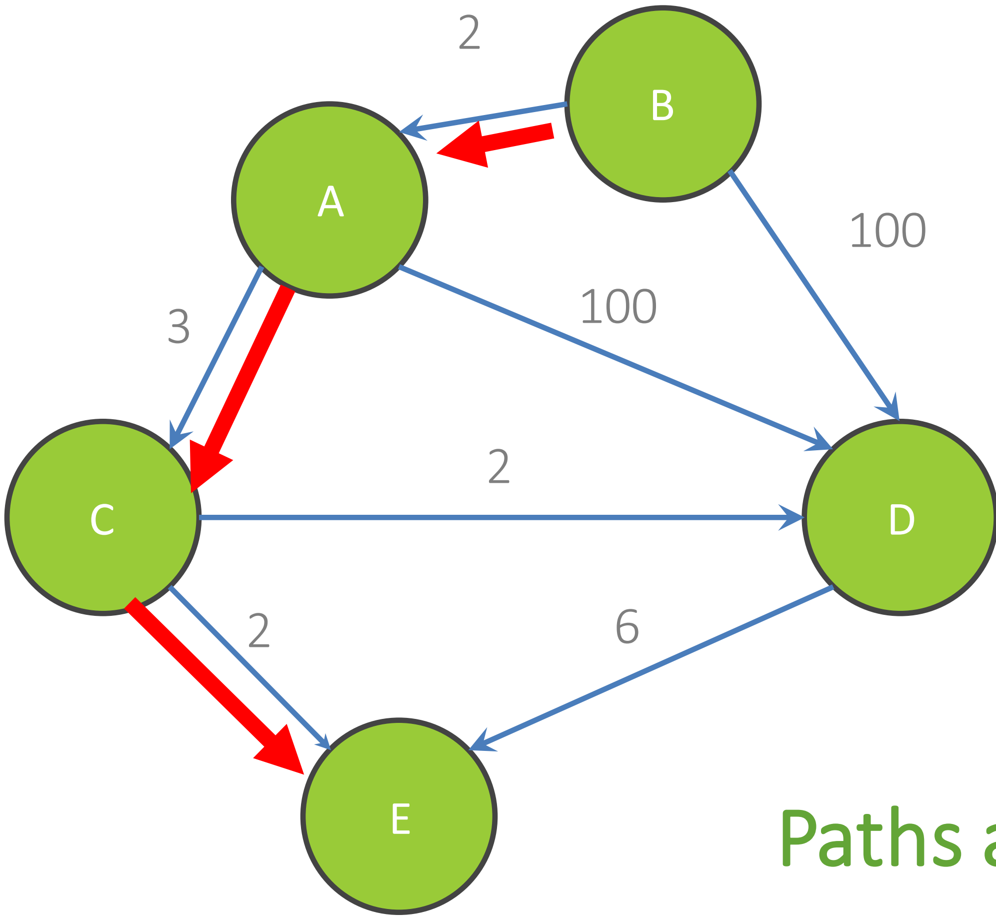


From Node B

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



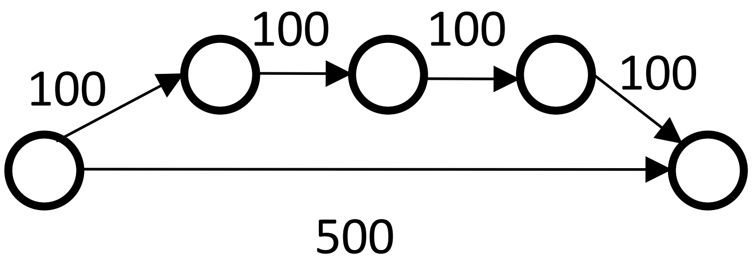
From Node B

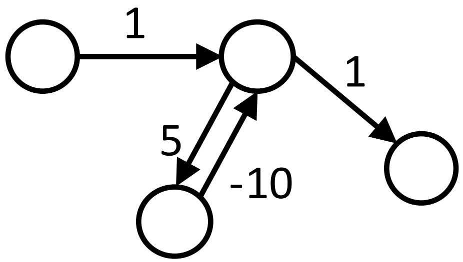
Destination	Path	Cost
A	<B,A>	2
B	<B>	0
C	<B,A,C>	5
D	<B,A,C,D>	7
E	<B,A,C,E>	7

Paths are not the same!

Goal: Smallest cost? Or fewest edges?

BFS vs. Dijkstra's Algorithm





BFS doesn't work when the path with minimal cost ≠ path with fewest edges

Dijkstra's Algorithm works in this case, but only 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

Wikipedia is a good resource: https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

The idea: reminiscent of BFS, but adapted to handle weights

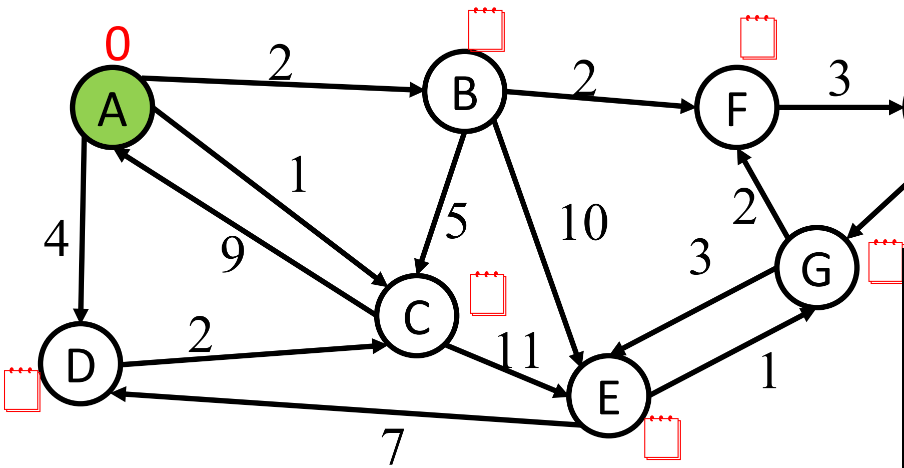
- Grow the set of nodes whose shortest distance has been computed
- Nodes not in the set will have a "best distance so far"
- A **PRIORITY QUEUE** will turn out to be useful for efficiency – We'll cover this later in the slide deck

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

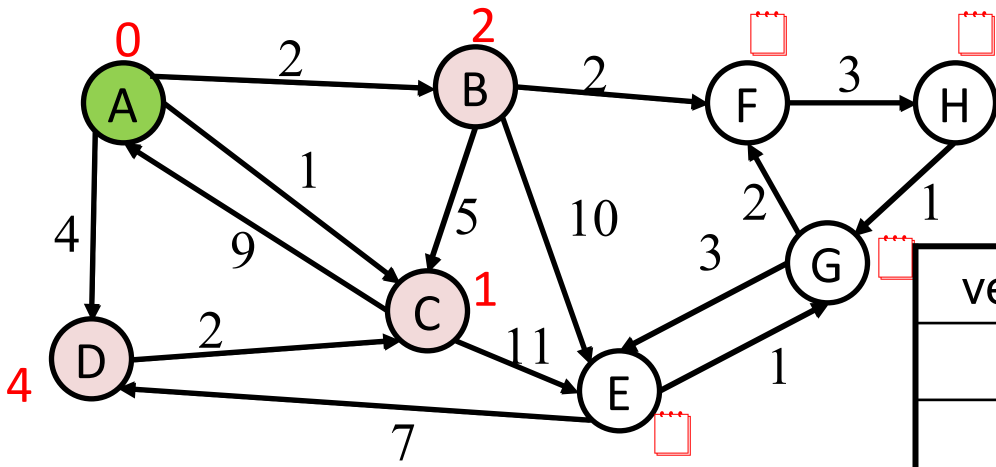


Goal: Fully explore the graph

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

Order Added to Known Set:

Example #1

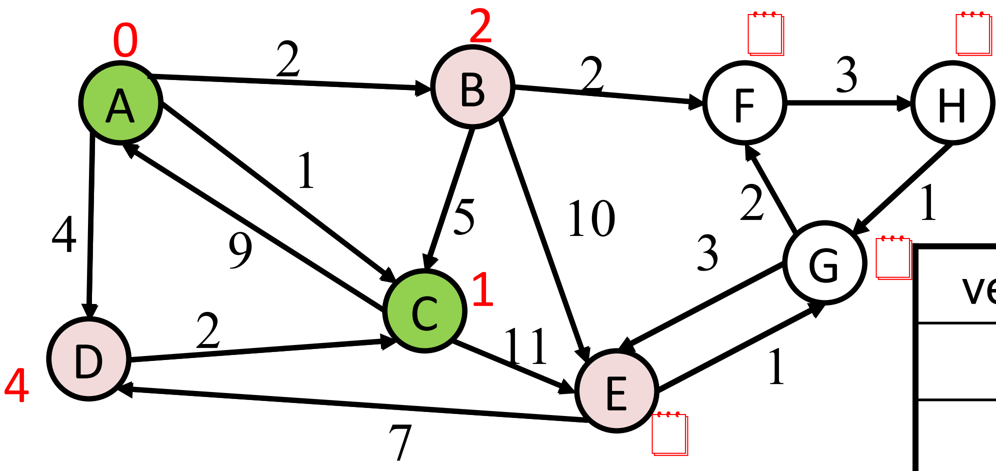


vertex	known?	cost	path
A	Y	0	
B		≤ 2	A
C		≤ 1	A
D		≤ 4	A
E		∞	
F		∞	
G		∞	
H		∞	

Order Added to Known Set:

A

Example #1

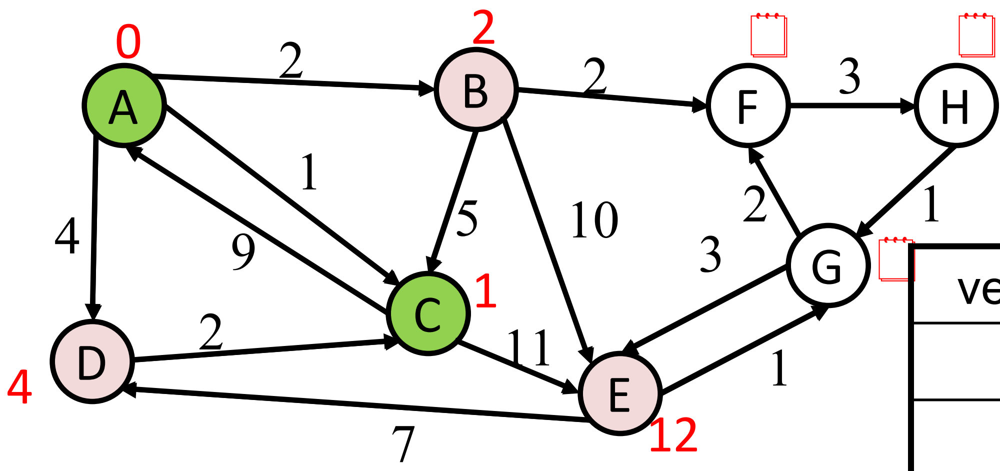


vertex	known?	cost	path
A	Y	0	
B		≤ 2	A
C	Y	1	A
D		≤ 4	A
E		∞	
F		∞	
G		∞	
H		∞	

Order Added to Known Set:

A, C

Example #1

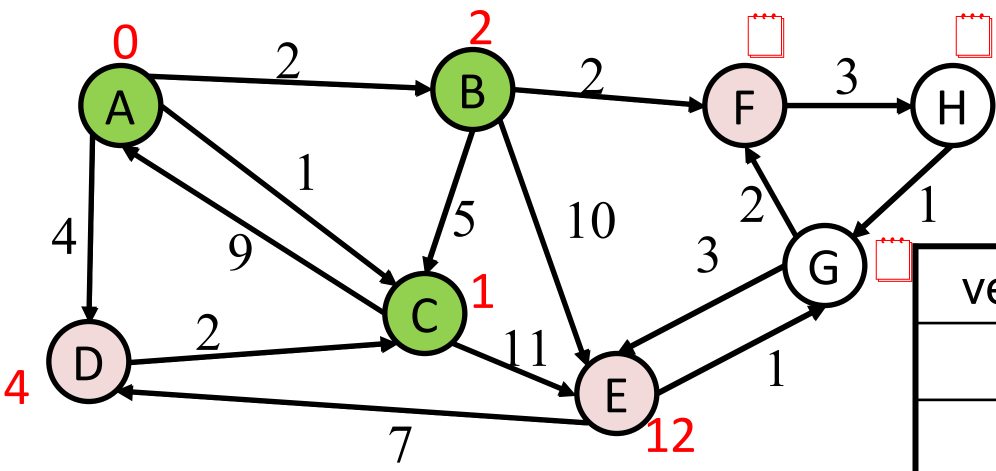


vertex	known?	cost	path
A	Y	0	
B		≤ 2	A
C	Y	1	A
D		≤ 4	A
E		≤ 12	C
F		∞	
G		∞	
H		∞	

Order Added to Known Set:

A, C

Example #1

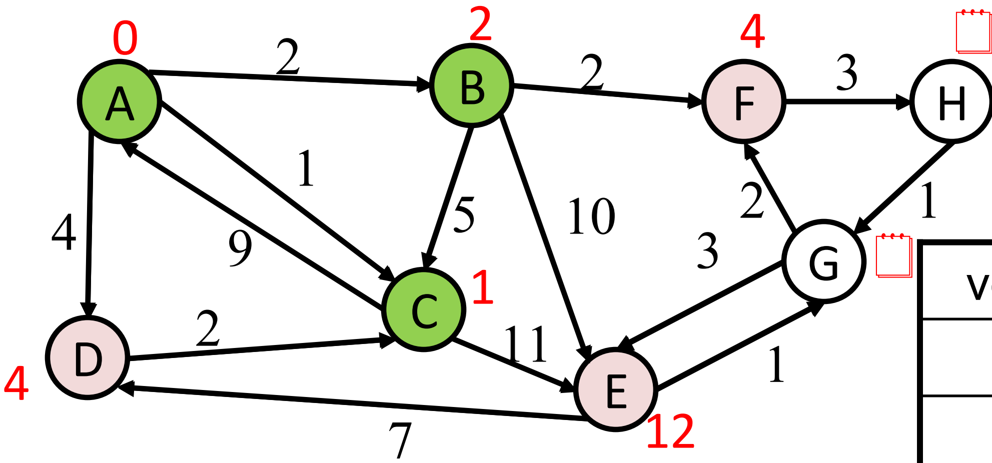


vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D		≤ 4	A
E		≤ 12	C
F		∞	
G		∞	
H		∞	

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

A, C, B

Example #1

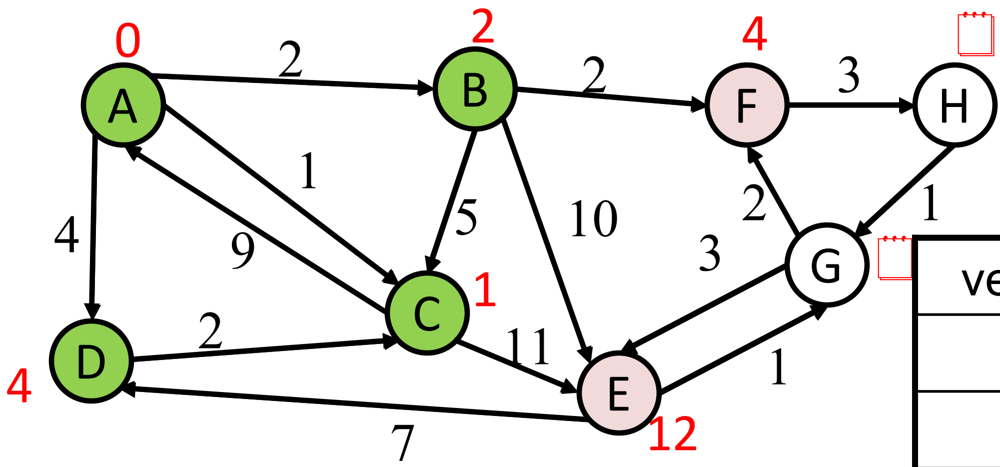


vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D		≤ 4	A
E		≤ 12	C
F		≤ 4	B
G		∞	
H		∞	

Order Added to Known Set:

A, C, B

Example #1

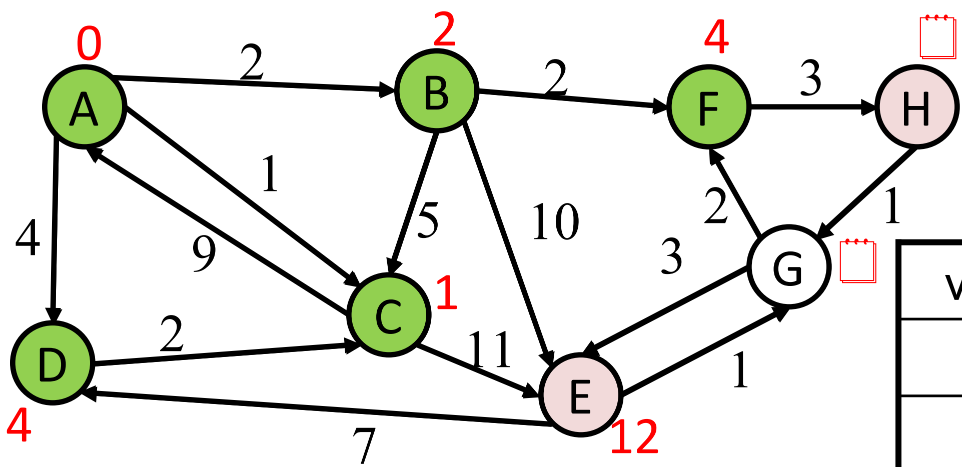


vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 12	C
F		≤ 4	B
G		∞	
H		∞	

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

A, C, B, D

Example #1

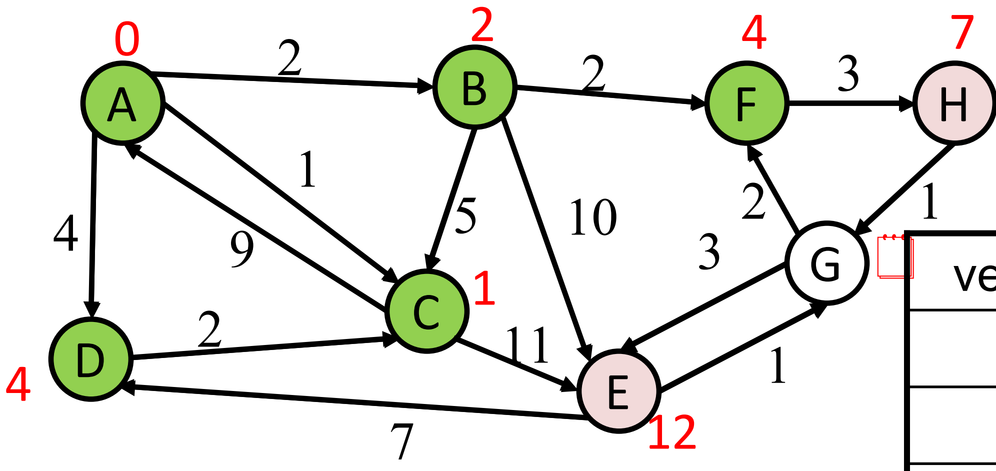


vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 12	C
F	Y	4	B
G		∞	
H		∞	

Order Added to Known Set:

A, C, B, D, F

Example #1

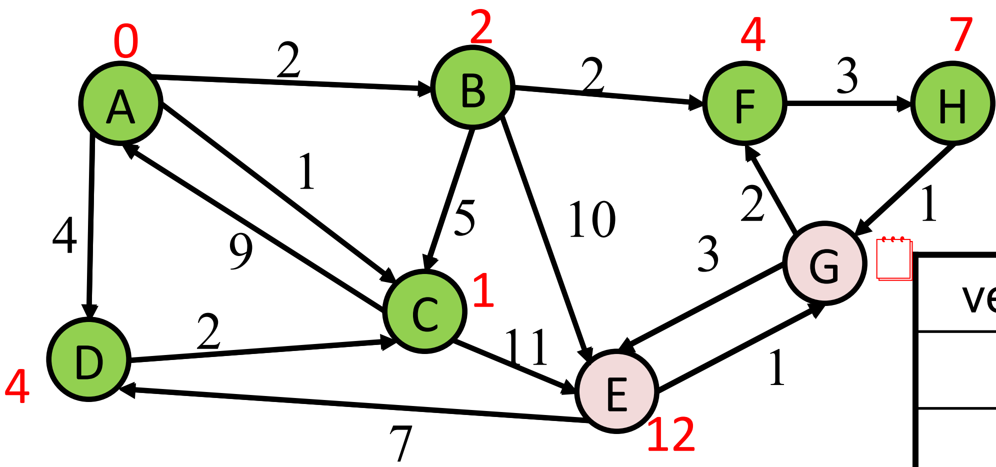


vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 12	C
F	Y	4	B
G		∞	
H		≤ 7	F

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

A, C, B, D, F

Example #1

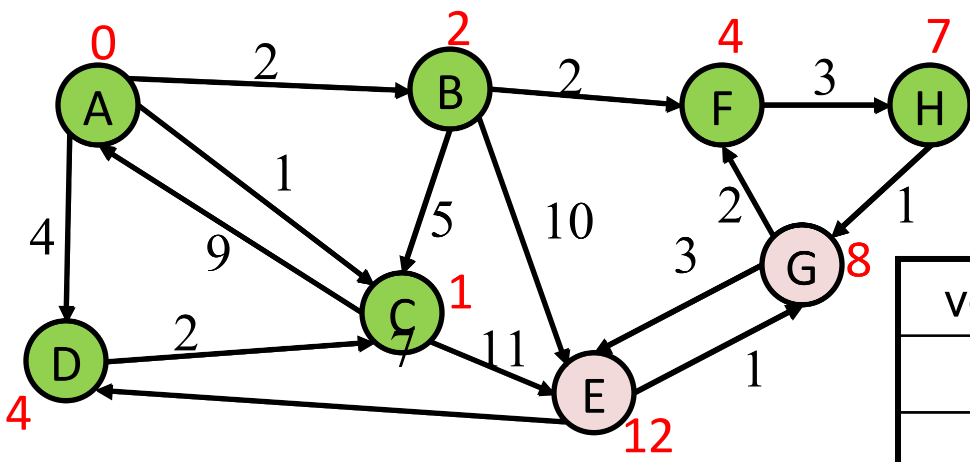


vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 12	C
F	Y	4	B
G		∞	
H	Y	7	F

Order Added to Known Set:

A, C, B, D, F, H

Example #1

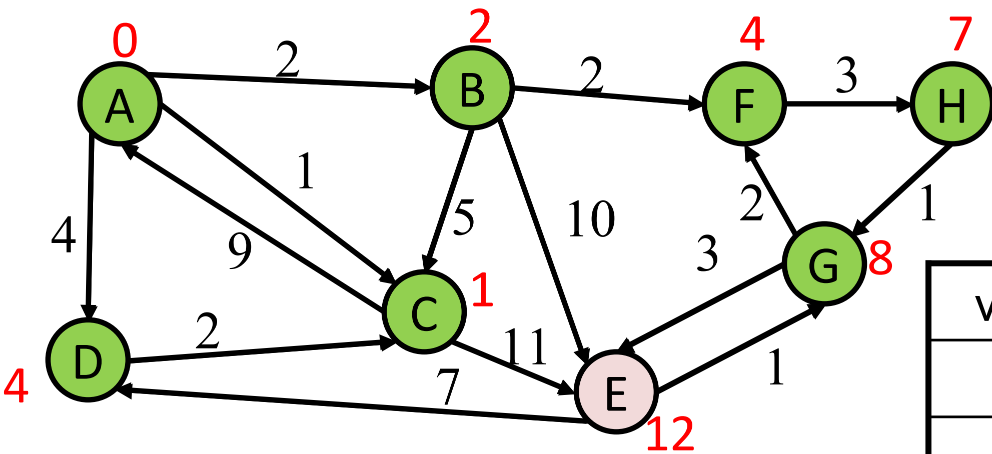


vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 12	C
F	Y	4	B
G		≤ 8	H
H	Y	7	F

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

A, C, B, D, F, H

Example #1

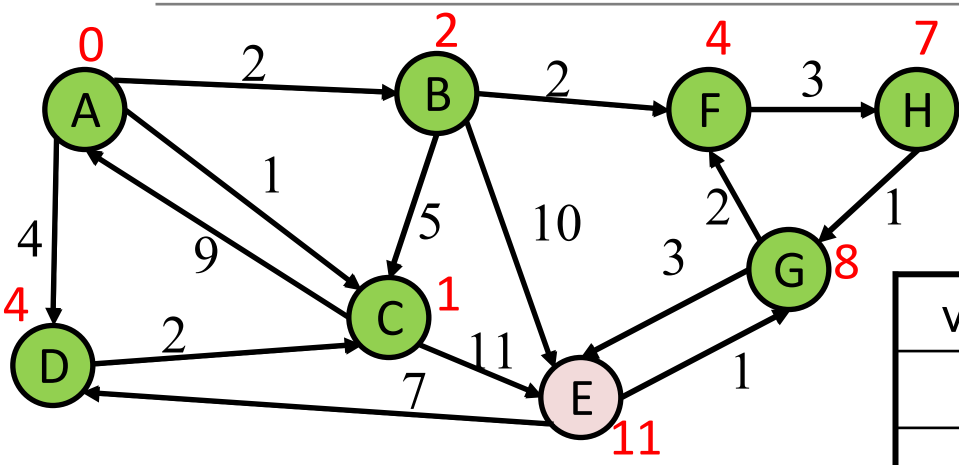


vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 12	C
F	Y	4	B
G	Y	8	H
H	Y	7	F

Order Added to Known Set:

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

Example #1

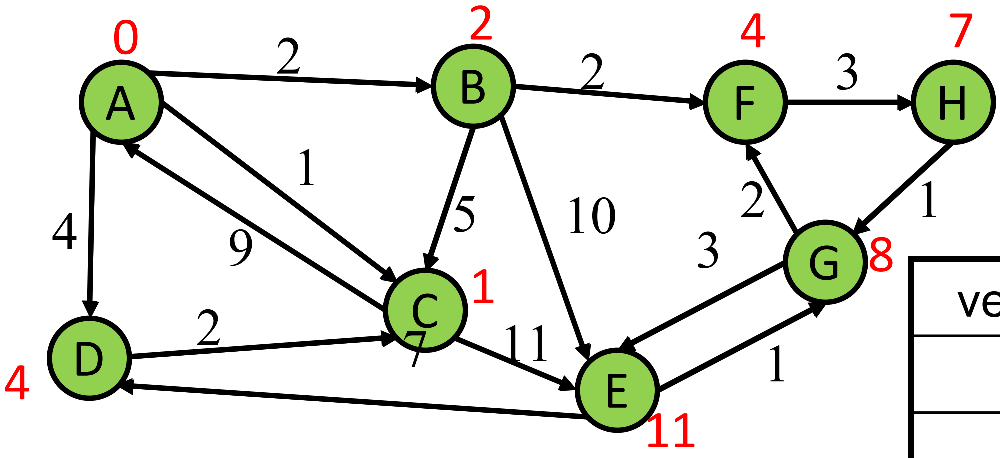


Order Added to Known Set:

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

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

Example #1

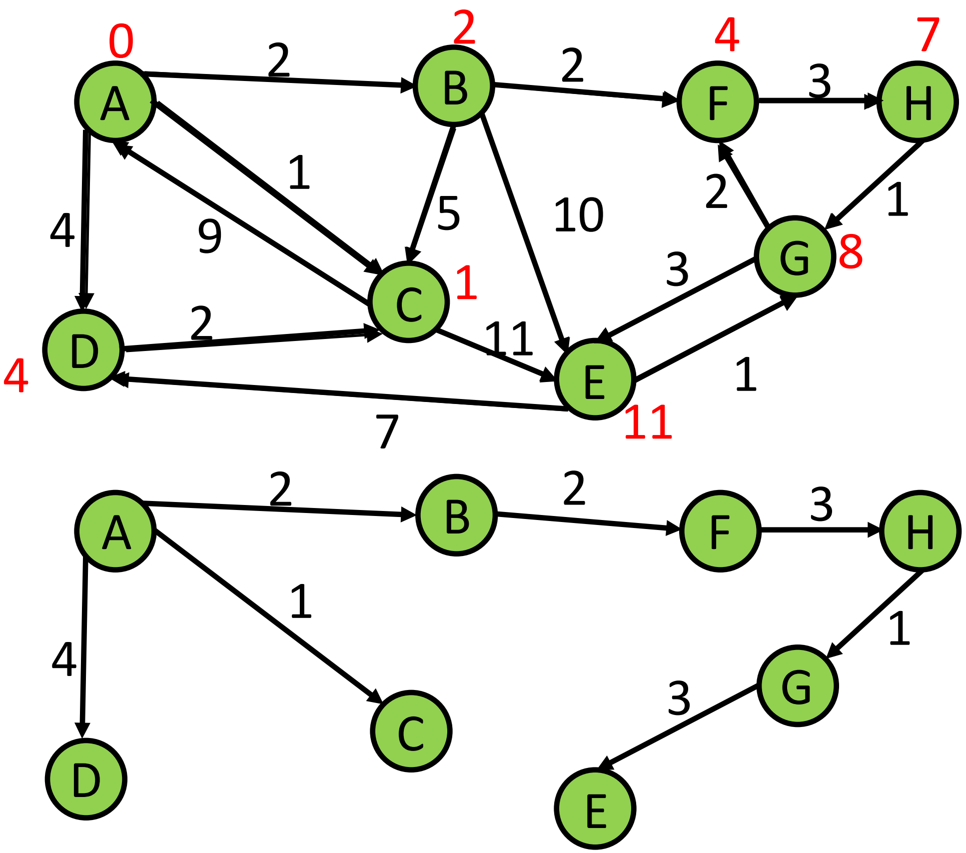


vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E	Y	11	G
F	Y	4	B
G	Y	8	H
H	Y	7	F

Order Added to Known Set:

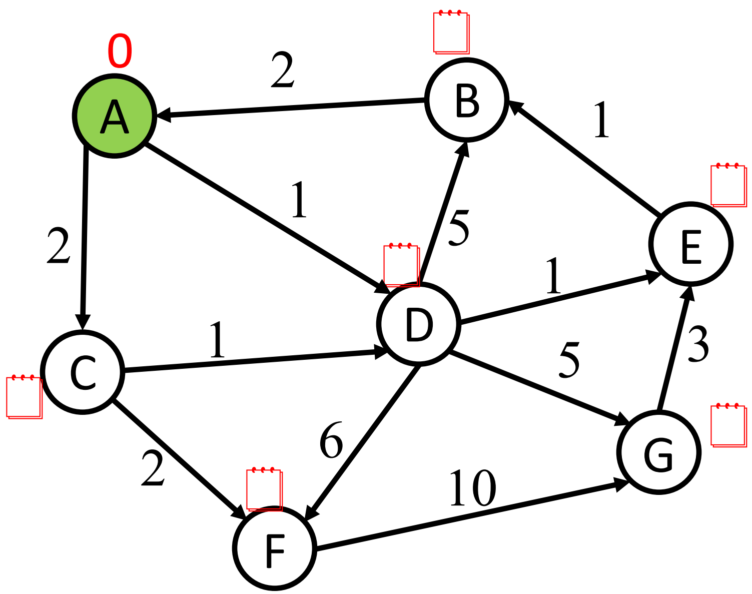
A, C, B, D, F, H, G, E

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

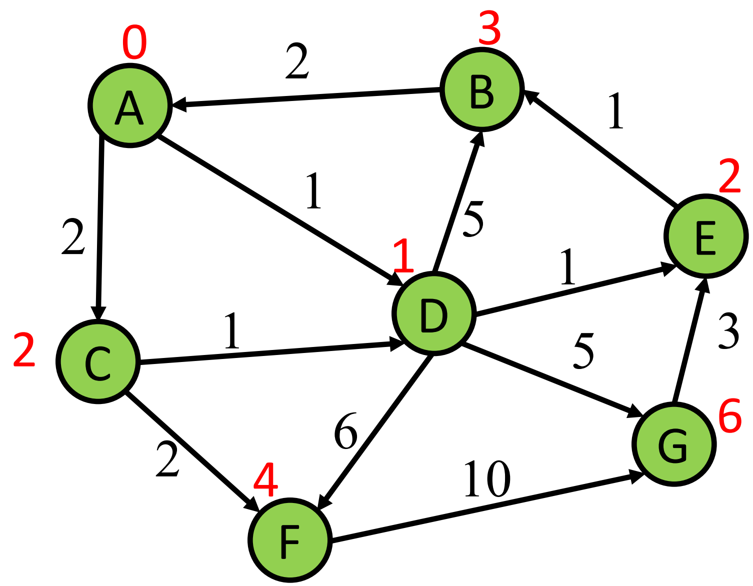
Example #2



Order Added to Known Set:

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

Example #2



Order Added to Known Set:

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

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

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(b.cost + weight((b,a)) < a.cost){
        a.cost = b.cost + weight((b,a))
        a.path = b
      }
   …
```

Can We Do Better?

Increase efficiency by considering lowest cost unknown vertex with sorting instead of looking at all vertices

Priority queue is like a queue, but returns elements by lowest value instead of FIFO

Insert and remove run in **O(log n)**

Naïve Dijkstra's: **O(|# of nodes|^2)**

With priority queue: **O(|# of edges| + |# of nodes| * log|# of nodes|)**

Priority Queue

Increase efficiency by considering lowest cost unknown vertex with sorting instead of looking at all vertices

Priority queue 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 H with all nodes
  while(heap is not empty) {
    b = H.extractMin()
    b.known = true
    for each edge (b,a) in G {
      if(b.cost + weight((b,a)) < a.cost){
        a.cost = b.cost + weight((b,a))
        a.path = b
      }
    …
```

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

Hw7 Important Notes!!!

<u>DO NOT</u> access data from hw6/src/data

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

<u>DO NOT</u> 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