Section 7:
Dijkstra's Algorithm and Generics

BRYAN VAN DRAANEN

SLIDES ADAPTED FROM ALEX MARIAKAKIS

WITH MATERIAL KELLEN DONOHUE, DAVID MAILHOT, AND DAN GROSSMAN

Java String Pair Class

```
 public class Pair {
     private String key;
     private String value;

     public Pair(String key, String value) {
          this.key = key;
          this.value = value;
     }

     public String getKey() {
          return this.key;
     }
}
```

Java Generic Pair Class

```
 public class Pair<K, V> {
     private K key;
     private V value;

     public Pair(K key, V value) {
          this.key = key;
          this.value = value;
     }

     public K getKey() {
          return this.key;
     }
}
```

Java Generic Pair Class (Comparable Generic Types)

```
 public class Pair<K extends Comparable<K>,
                  V extends Comparable<V>> {
     private K key;
     private V value;

     public Pair(K key, V value) {
           this.key = key;
           this.value = value;
     }

     public K getKey() {
           return this.key;
     }
}
```

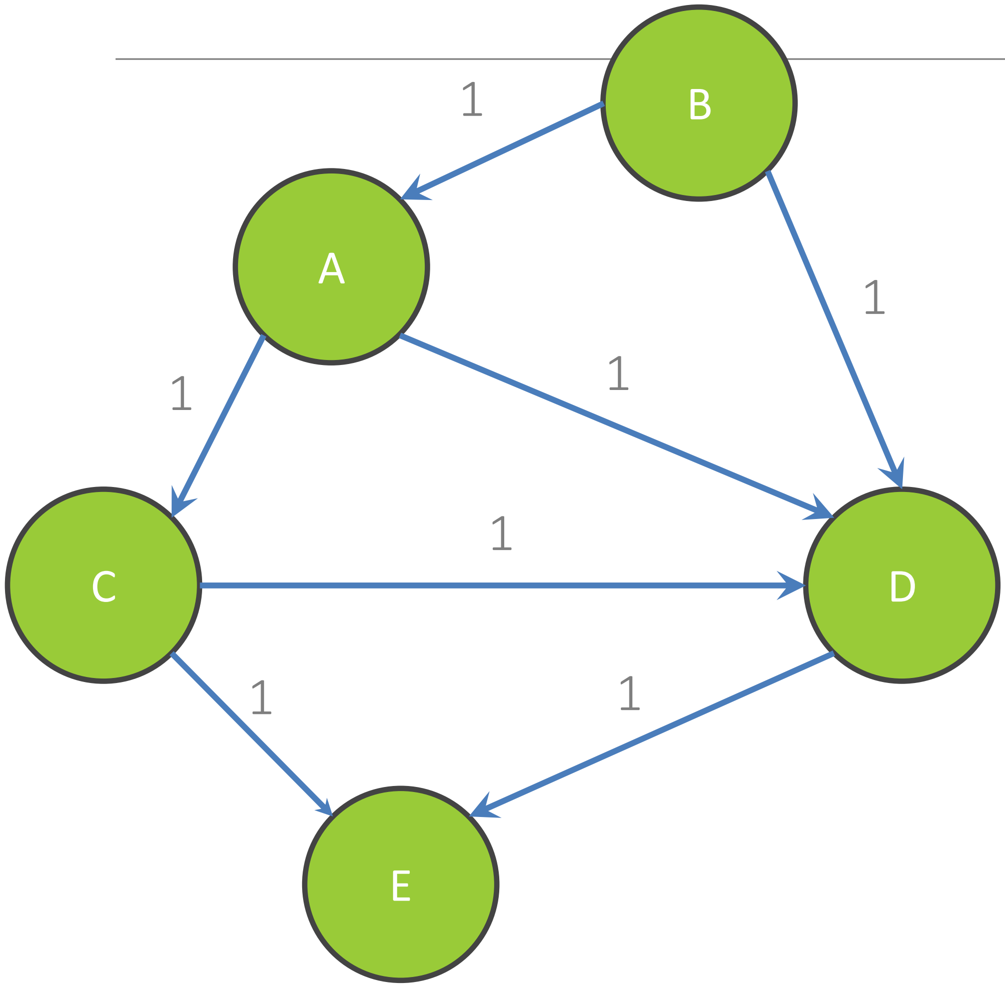
Java Generic Pair Class (Comparable Generic Types and Comparable Generic Pair)

```
public class Pair<K extends Comparable<K>,
                    V extends Comparable<V>>
                    implements Comparable<Pair<K,V>>{
      private K key;
      private V value;

      public Pair(K key, V value) {
             this.key = key;
             this.value = value;
      }

      public int compareTo(Pair<K, V> other) {
             int r = key.compareTo(other.key);
             return r != 0 ? r : value.compareTo(other.value);
      }
}
```

Using Generic Java Pair

```
Pair<String, String> pair = new Pair<String,String>(“hey”,“listen!”);

String key = pair.getKey();

Pair<Integer, Integer> intPair = new Pair<Integer, Integer>(1, 2);

Integer key2 = intPair.getKey();

List<Pair<String,String>> lst = new ArrayList<Pair<String,String>>();

lst.add(pair); // Add your generic pair to a generic collection!
```

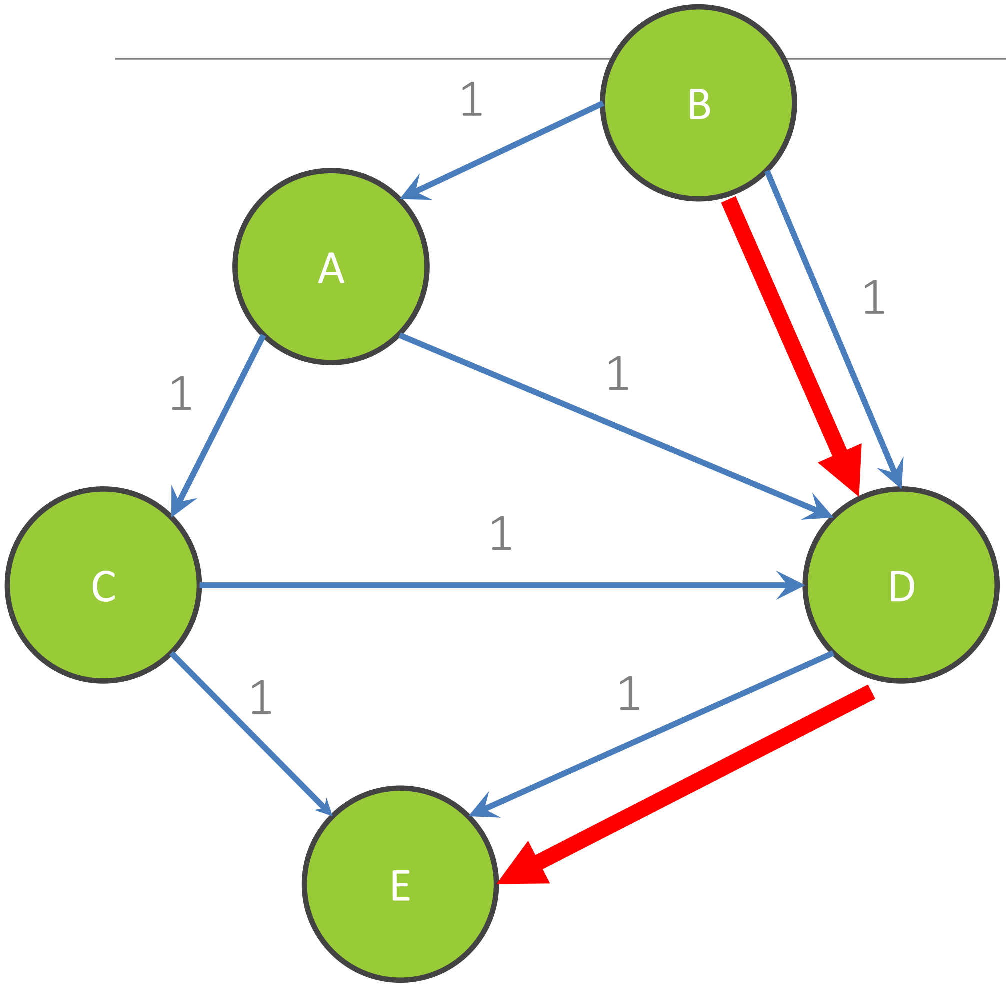
Review: Shortest Paths with BFS



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

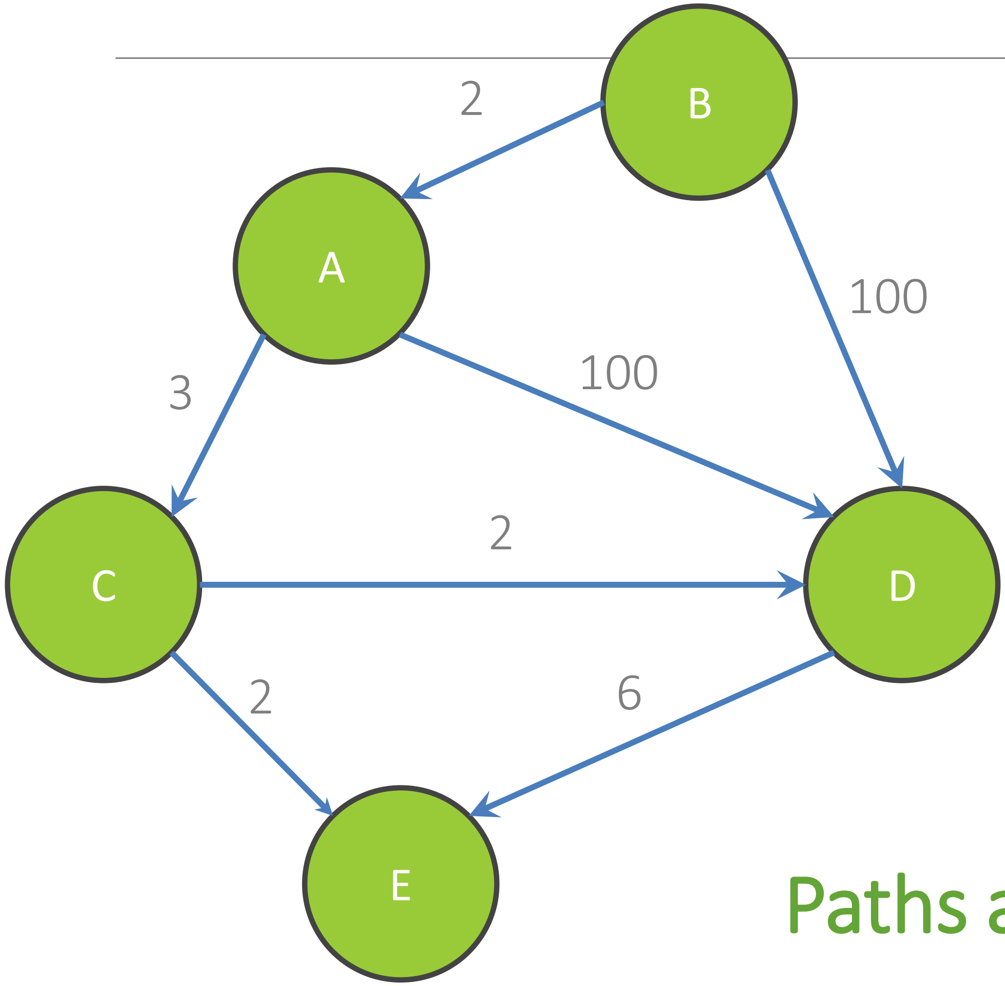
Review: Shortest Paths with BFS



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

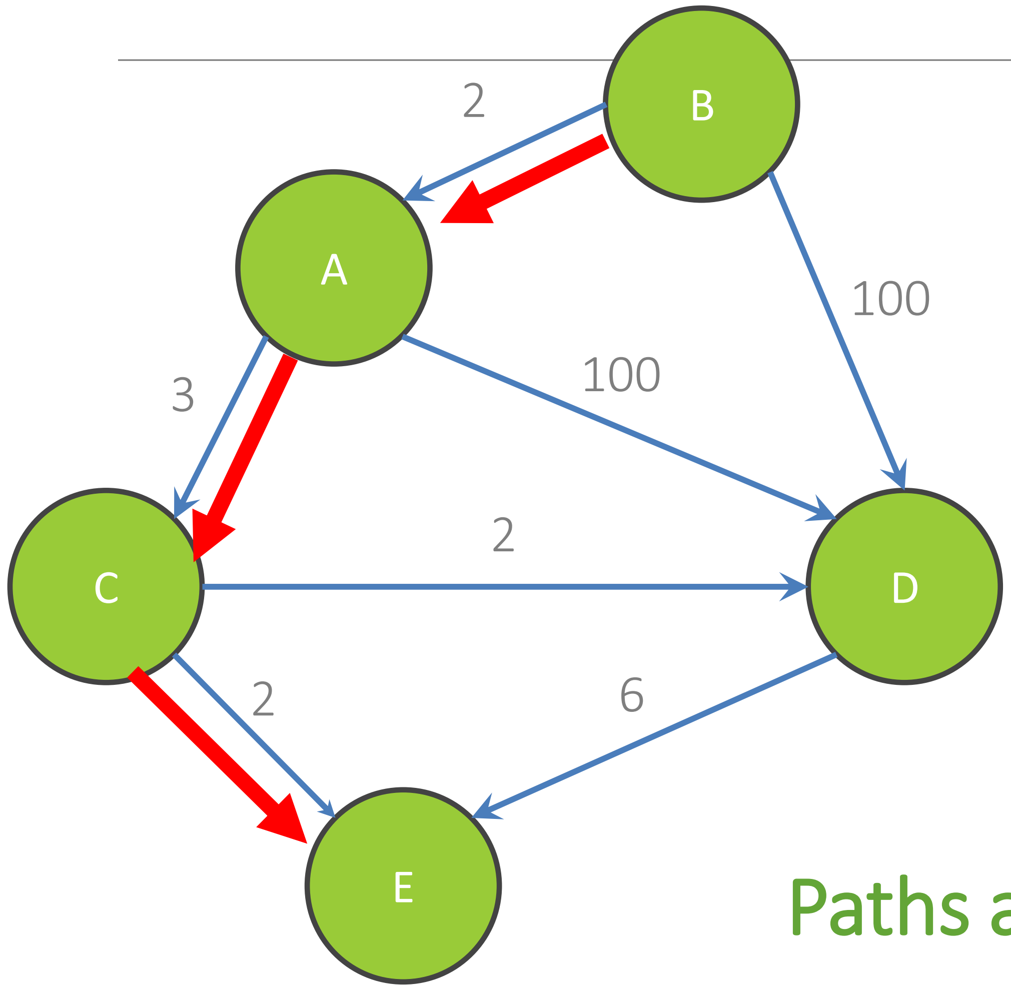


From Node B

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



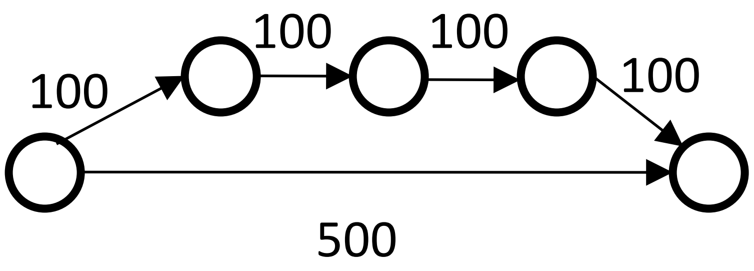
From Node B

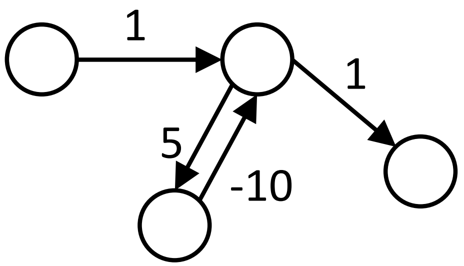
Destination	Path	Cost
A	<B,A>	2
B	<B>	0
C	<B,A,C>	5
D	<B,A,C,D>	7
E	<B,A,C,E>	7

Paths are not the same!

Goal: Smallest cost? Or fewest edges?

BFS vs. Dijkstra’s





BFS doesn’t work because path with minimal cost ≠ path with fewest edges

Note: Dijkstra’s only 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

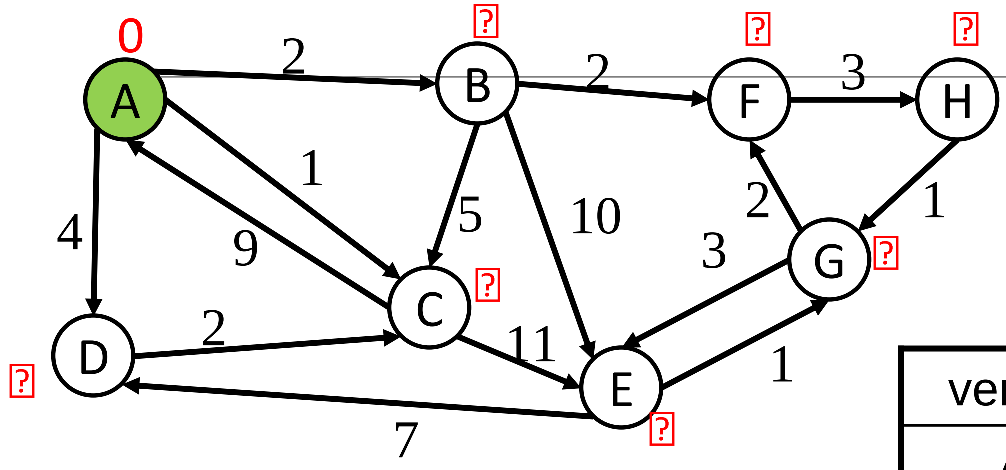
- Grow the set of nodes whose shortest distance has been computed
- Nodes not in the set will have a “best distance so far”
- A **PRIORITY QUEUE** will turn out to be useful for efficiency – We’ll cover this later in the slide deck

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

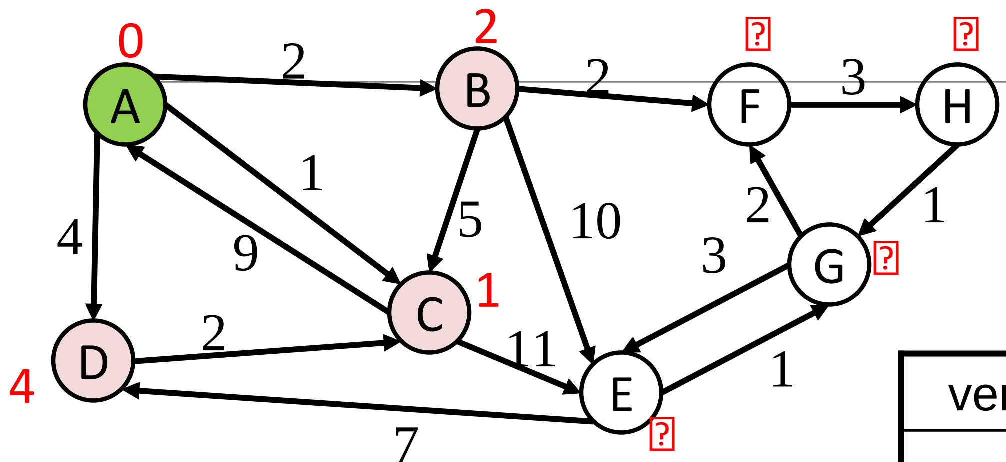


Goal: Fully explore the graph

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

Order Added to Known Set:

Example #1

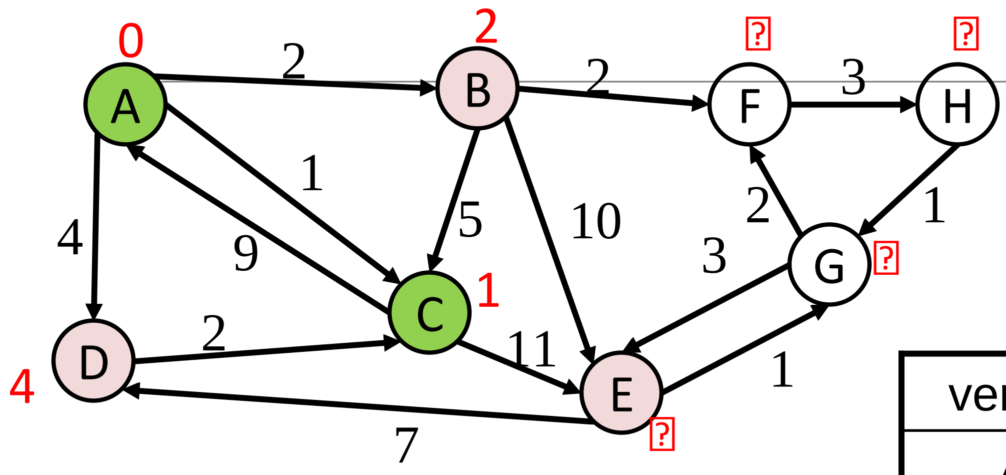


Order Added to Known Set:

A

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

Example #1

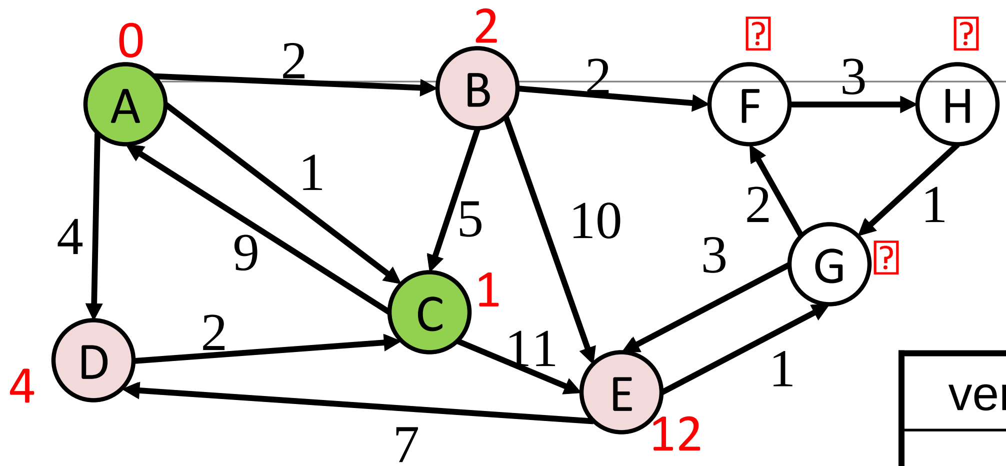


Order Added to Known Set:

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

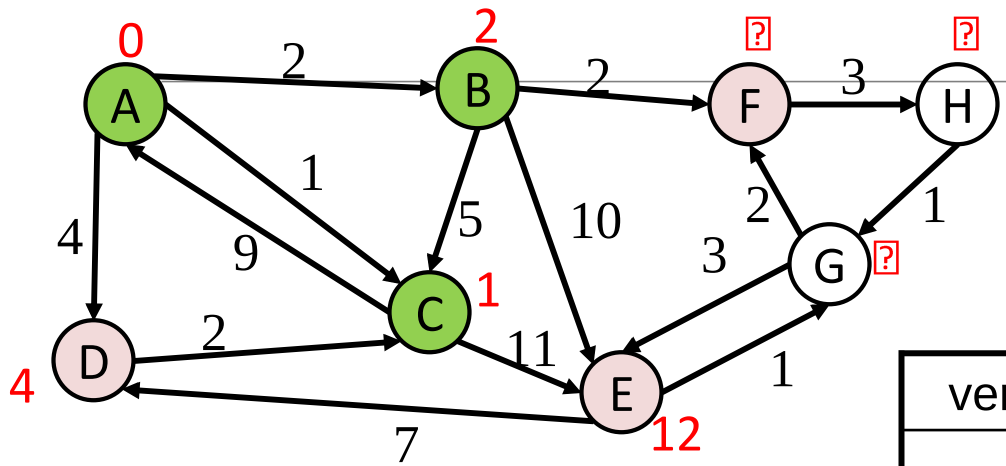


vertex	known?	cost	path
A	Y	0	
B		≤ 2	A
C	Y	1	A
D		≤ 4	A
E		≤ 12	C
F		∞	
G		∞	
H		∞	

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

A, C

Example #1

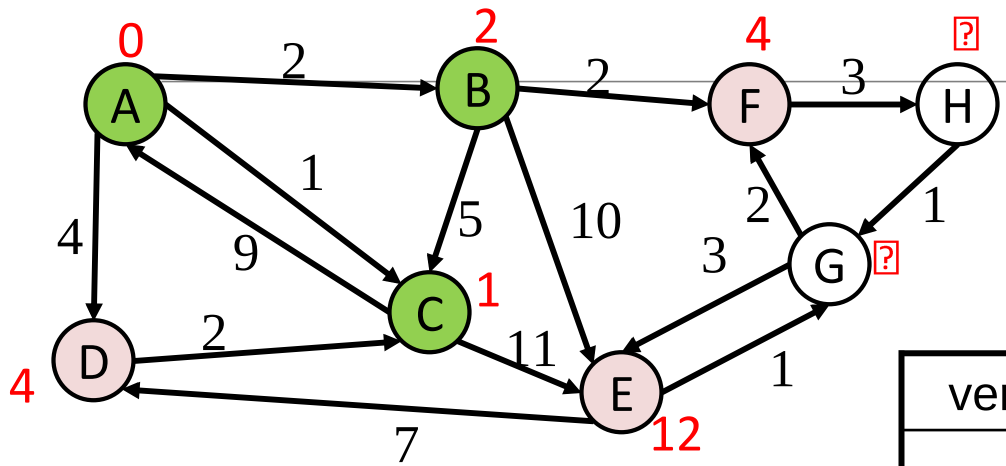


vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D		≤ 4	A
E		≤ 12	C
F		∞	
G		∞	
H		∞	

Order Added to Known Set:

A, C, B

Example #1

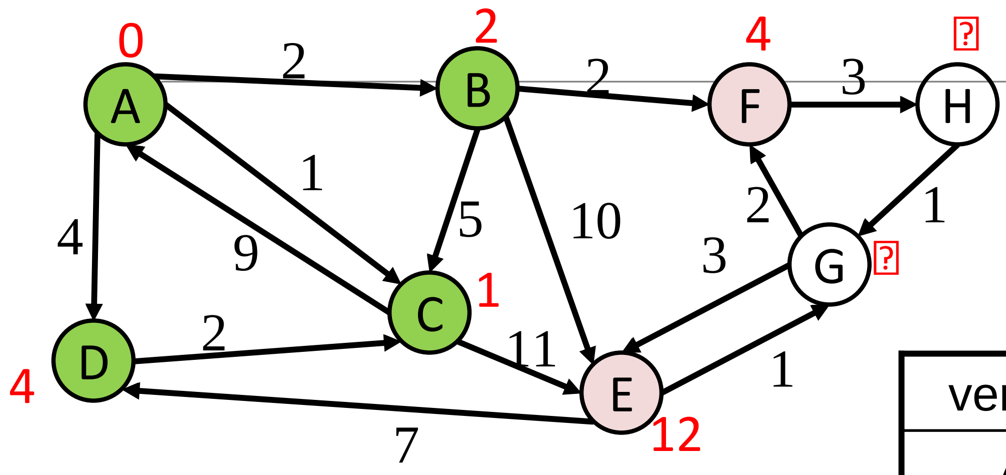


vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D		≤ 4	A
E		≤ 12	C
F		≤ 4	B
G		∞	
H		∞	

Order Added to Known Set:

A, C, B

Example #1

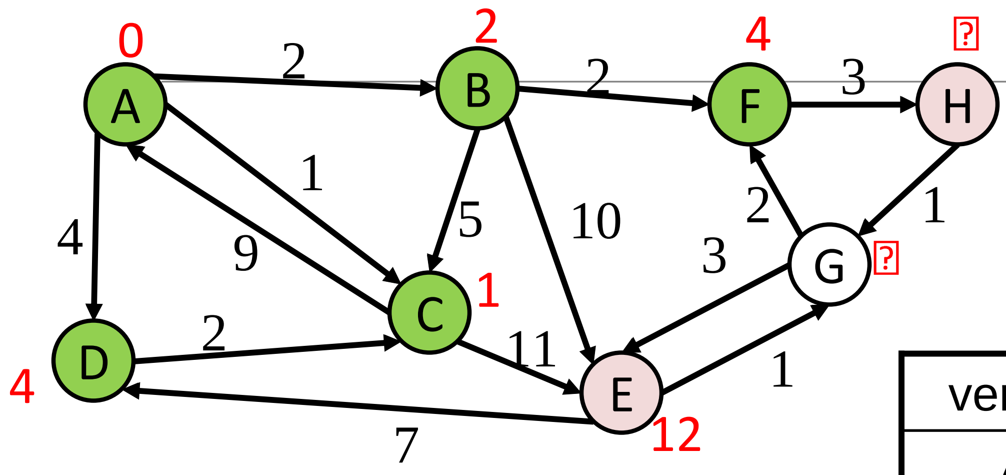


vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 12	C
F		≤ 4	B
G		∞	
H		∞	

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

A, C, B, D

Example #1

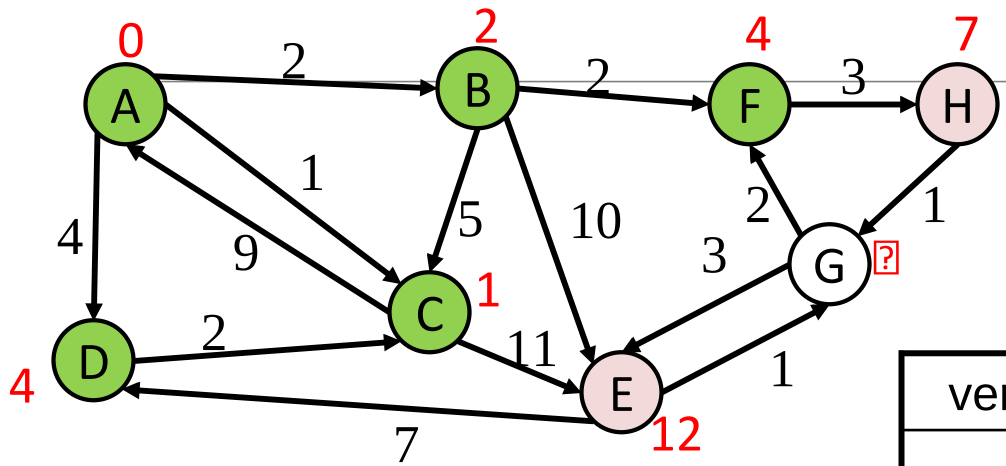


vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 12	C
F	Y	4	B
G		∞	
H		∞	

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

A, C, B, D, F

Example #1

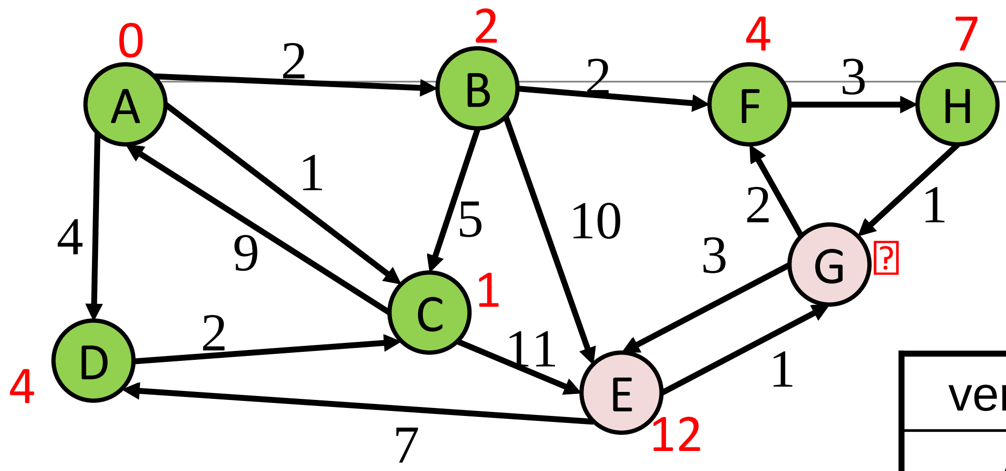


vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 12	C
F	Y	4	B
G		∞	
H		≤ 7	F

Order Added to Known Set:

A, C, B, D, F

Example #1

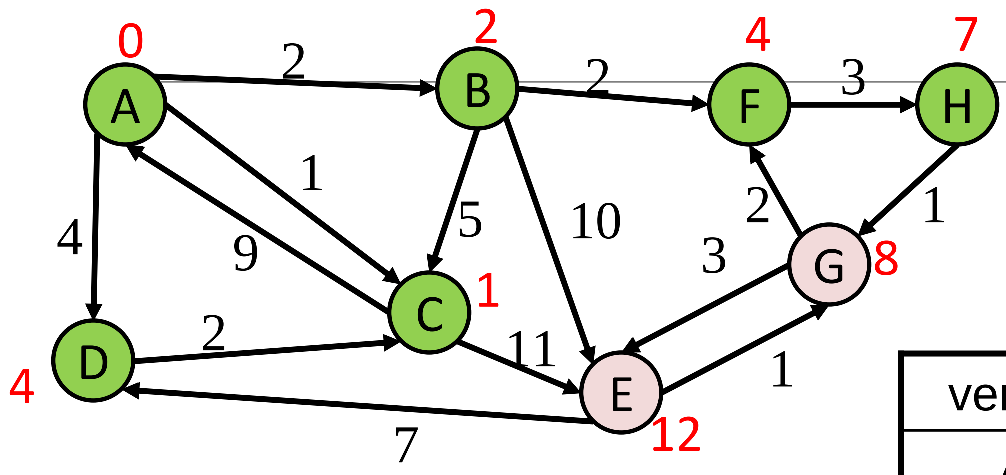


vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 12	C
F	Y	4	B
G		∞	
H	Y	7	F

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

A, C, B, D, F, H

Example #1

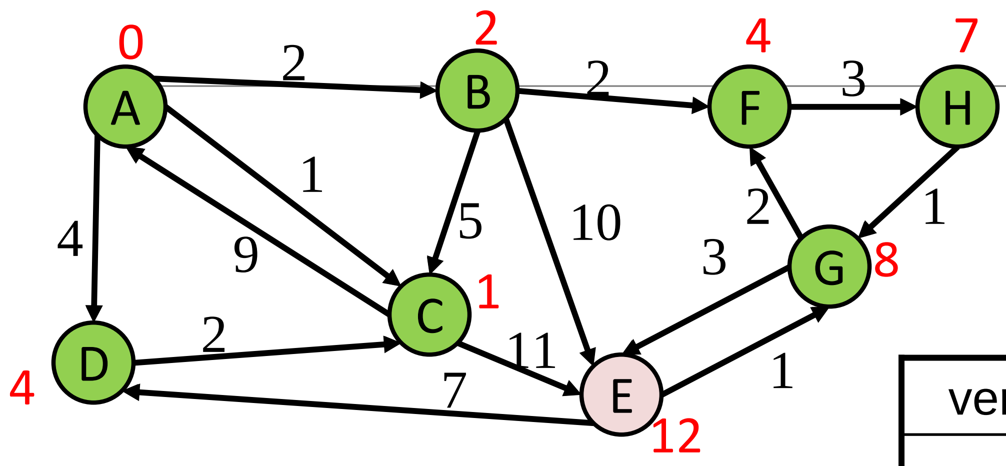


vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 12	C
F	Y	4	B
G		≤ 8	H
H	Y	7	F

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

A, C, B, D, F, H

Example #1

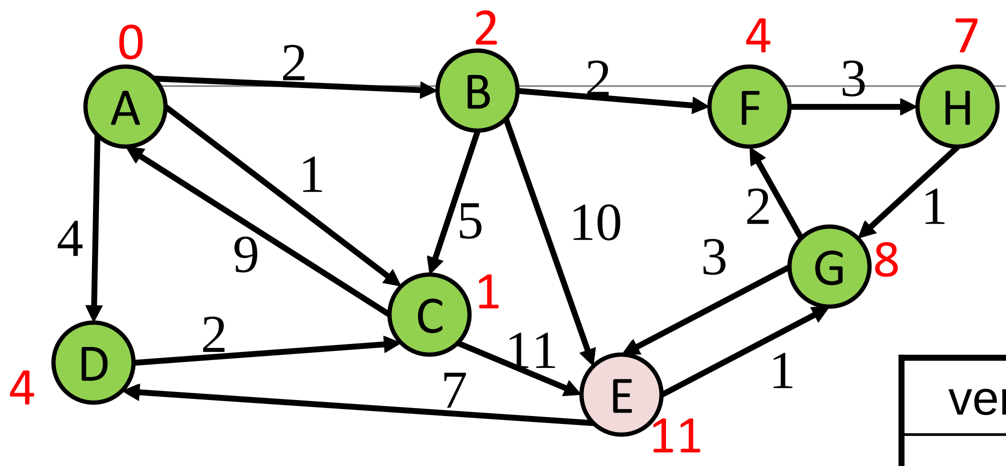


vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 12	C
F	Y	4	B
G	Y	8	H
H	Y	7	F

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

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

Example #1

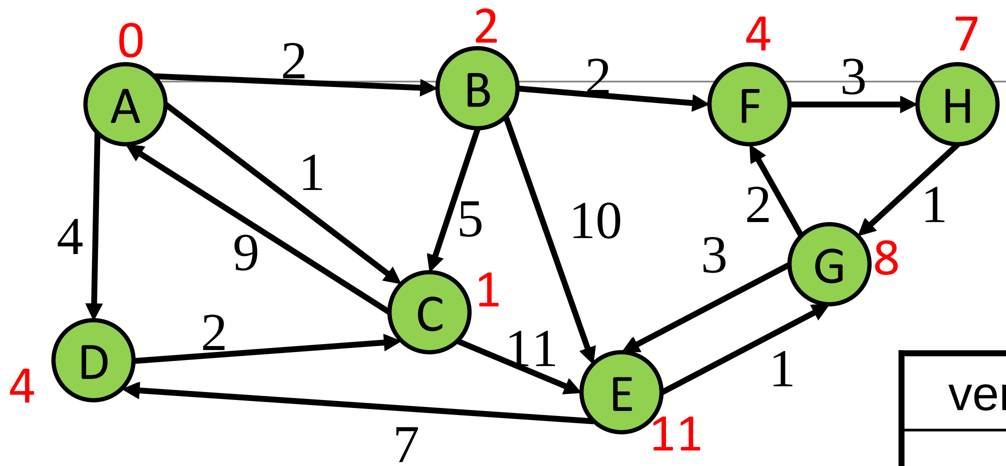


vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E		≤ 11	G
F	Y	4	B
G	Y	8	H
H	Y	7	F

Order Added to Known Set:

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

Example #1

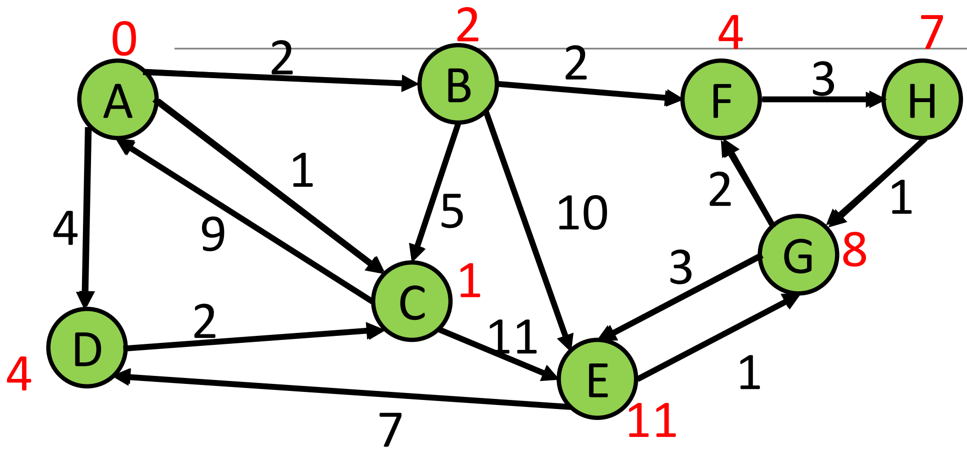


vertex	known?	cost	path
A	Y	0	
B	Y	2	A
C	Y	1	A
D	Y	4	A
E	Y	11	G
F	Y	4	B
G	Y	8	H
H	Y	7	F

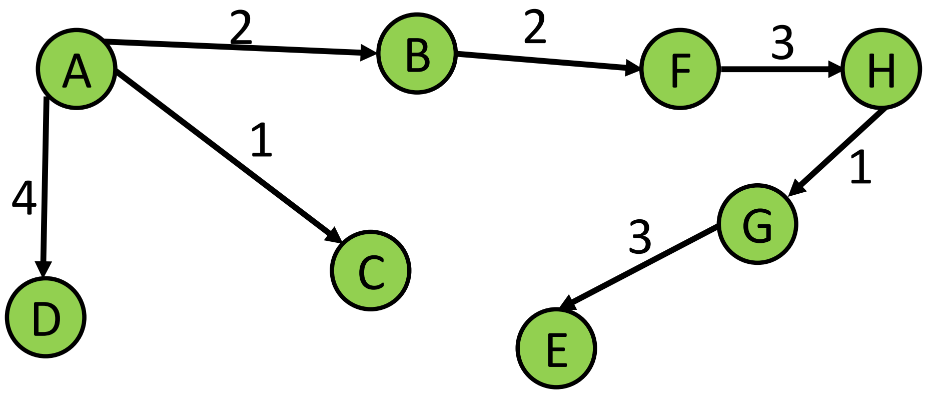
Order Added to Known Set:

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

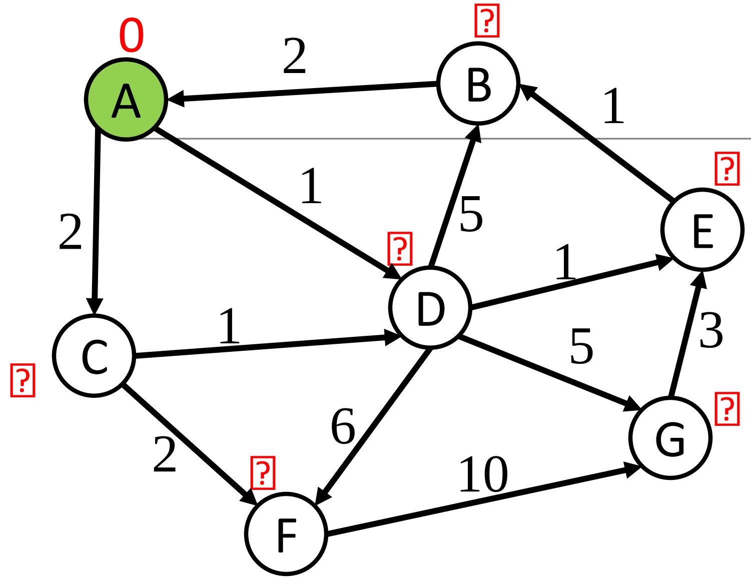
Interpreting the Results



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



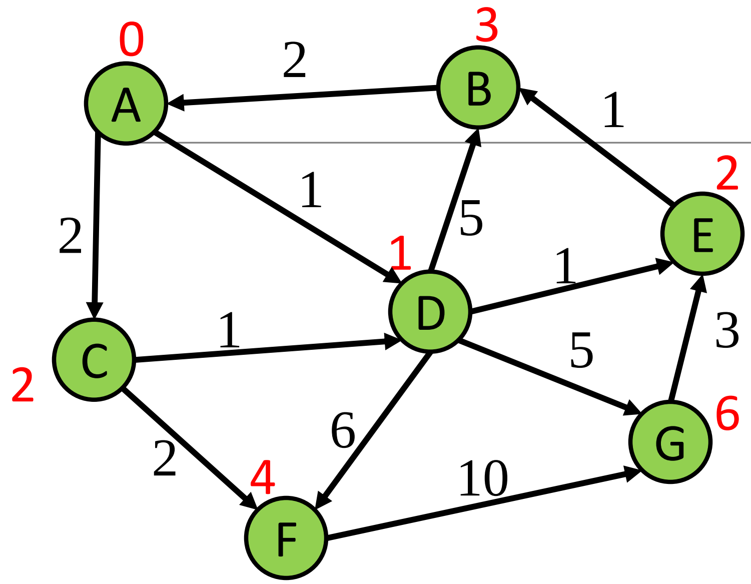
Example #2



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(!a.known) {
        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

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())

Priority Queue Example

Priority Queue q compared by path weights (lower weights higher priority)

Suppose q = [2, 4, 5]

```
        q.add(3); - path with weight 3
        q.add(8); - path with weight 8

q = [2, 3, 4, 5, 8]

        x = q.remove();

q = [3, 4, 5, 8]
x = 2
```

Pseudocode with PriorityQueue

```
start = starting node
dest = destination node
active = priority queue -(each element is a path from start to a given node -
                          Priority based on total cost of individual path)
finished = set of nodes - where least-cost path is known

add a path from start to itself to active


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

   if minDest is dest:
        return minPath
   if minDest is in finished:
        continue
   add minDest to finished

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

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!!!

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

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

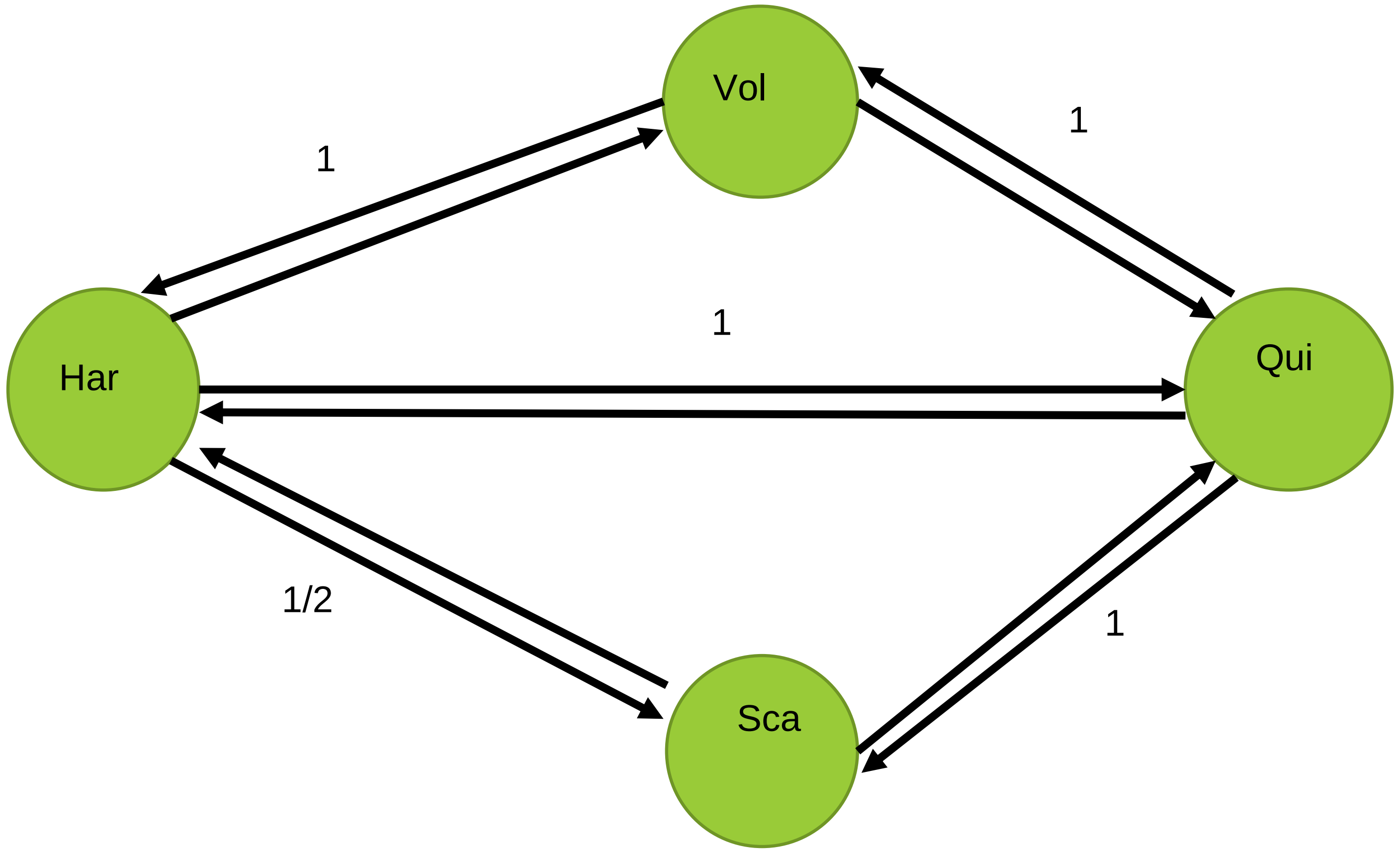
HP2 -> Harry, Scabbers,

HP3 -> Harry

HP4 -> Harry, Voldemort

SharedAHead -> Voldemort, Quirrel

Graph Activity Answers


Vol
1
1
1
Har
Qui
1/2
1
Sca