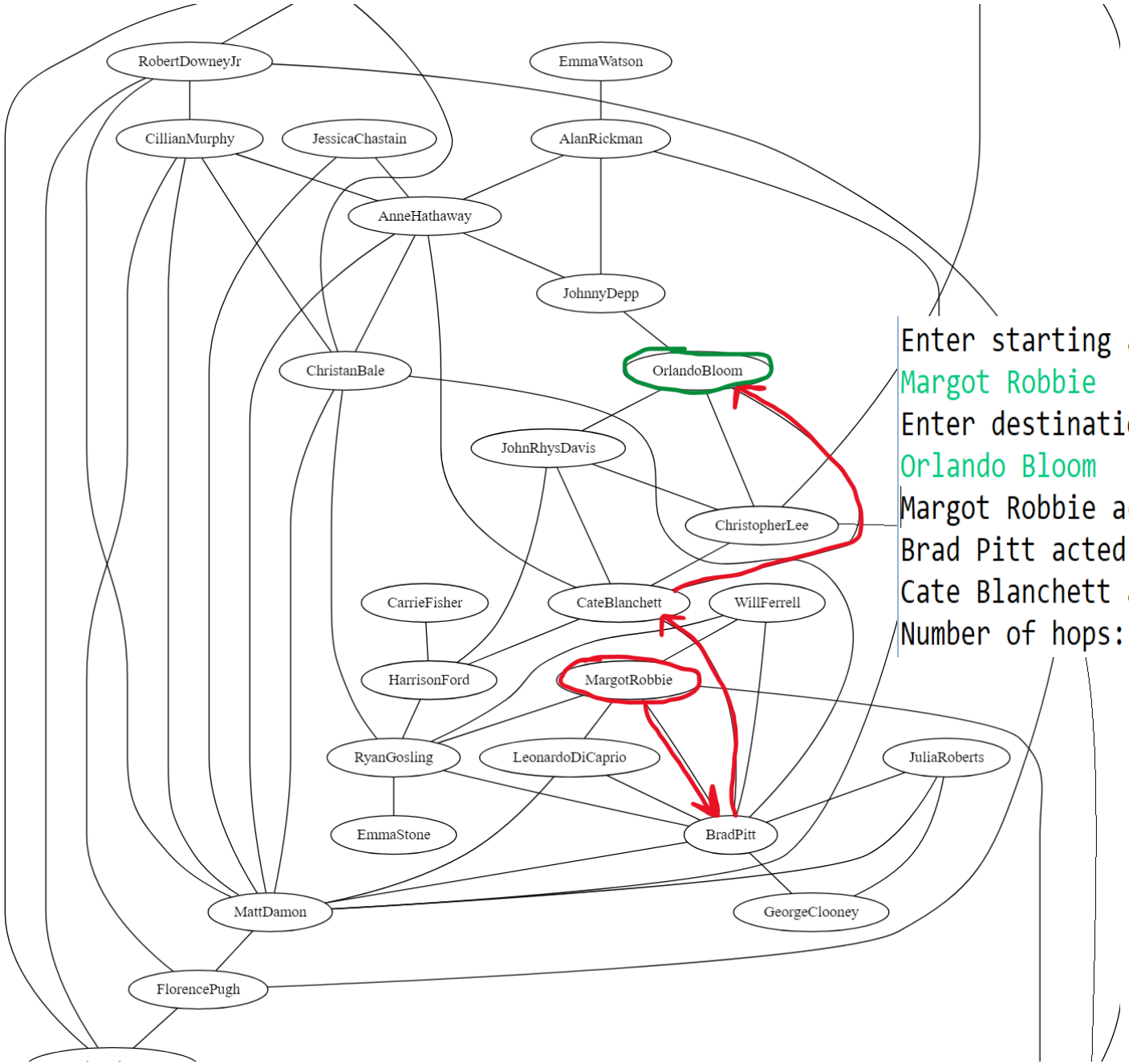
CSCI 232:
Data Structures and Algorithms

Program 3

Reese Pearsall

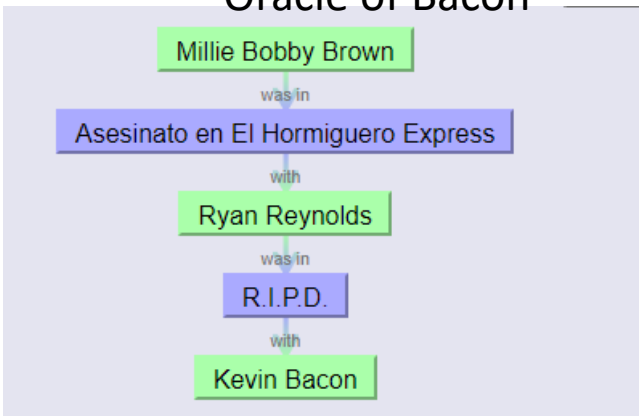
Summer 2025

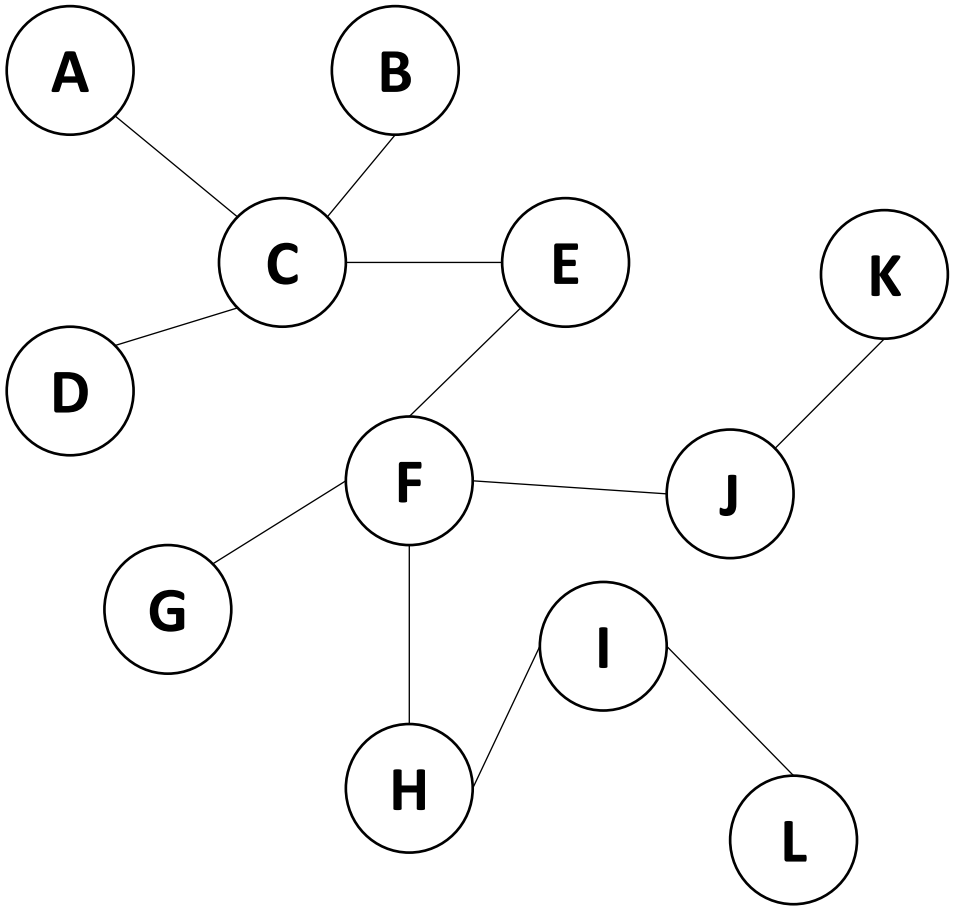


Finding Shortest Path between actors

```
Enter starting actor:
Margot Robbie
Enter destination actor:
Orlando Bloom
Margot Robbie acted with Brad Pitt in Once Upon a Time in Hollywood
Brad Pitt acted with Cate Blanchett in The Curious Case of Benjamin Button
Cate Blanchett acted with Orlando Bloom in Fellowship of the Ring
Number of hops: 3
```

Oracle of Bacon

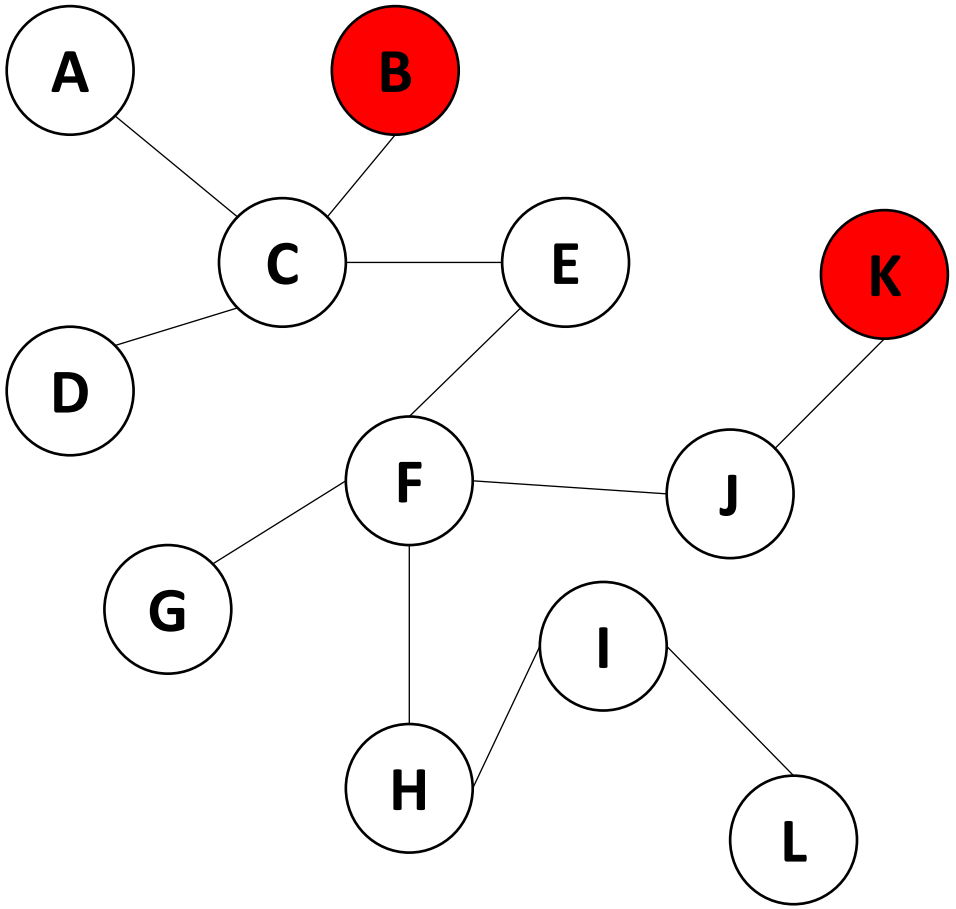




Consider an **Acyclic** graph (a graph with no cycles) (a “tree”)

Observation:

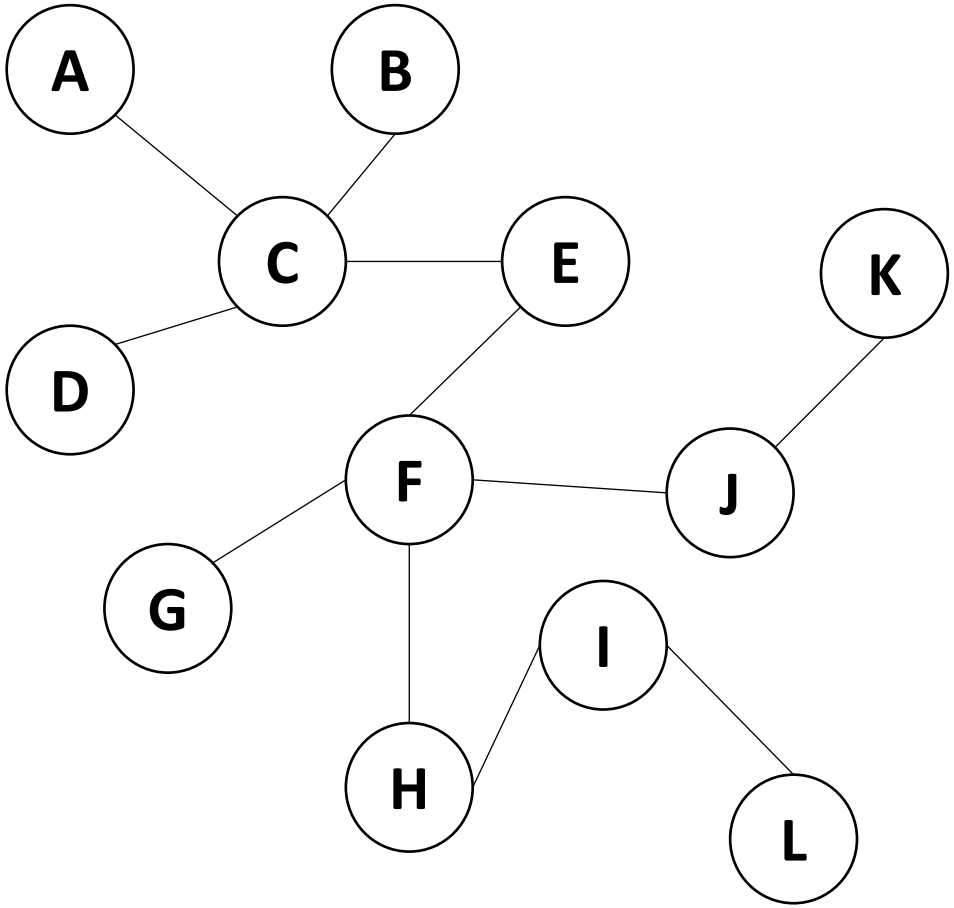
Pick any two vertices (V1, V2). There is **only one possible path** that goes from V1 to V2 (and vice versa)



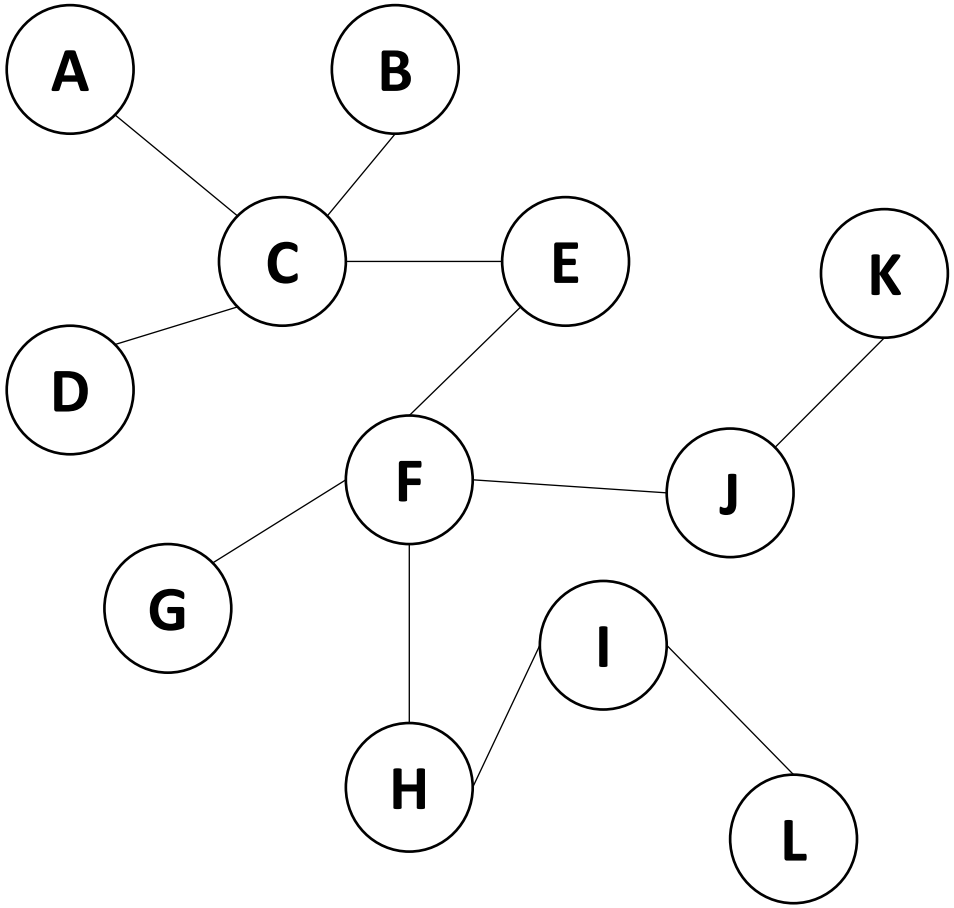
Consider an **Acyclic** graph (a graph with no cycles) (a “tree”)

Observation:

Pick any two vertices (V1, V2). There is **only one possible path** that goes from V1 to V2 (and vice versa)



Longest Path ?

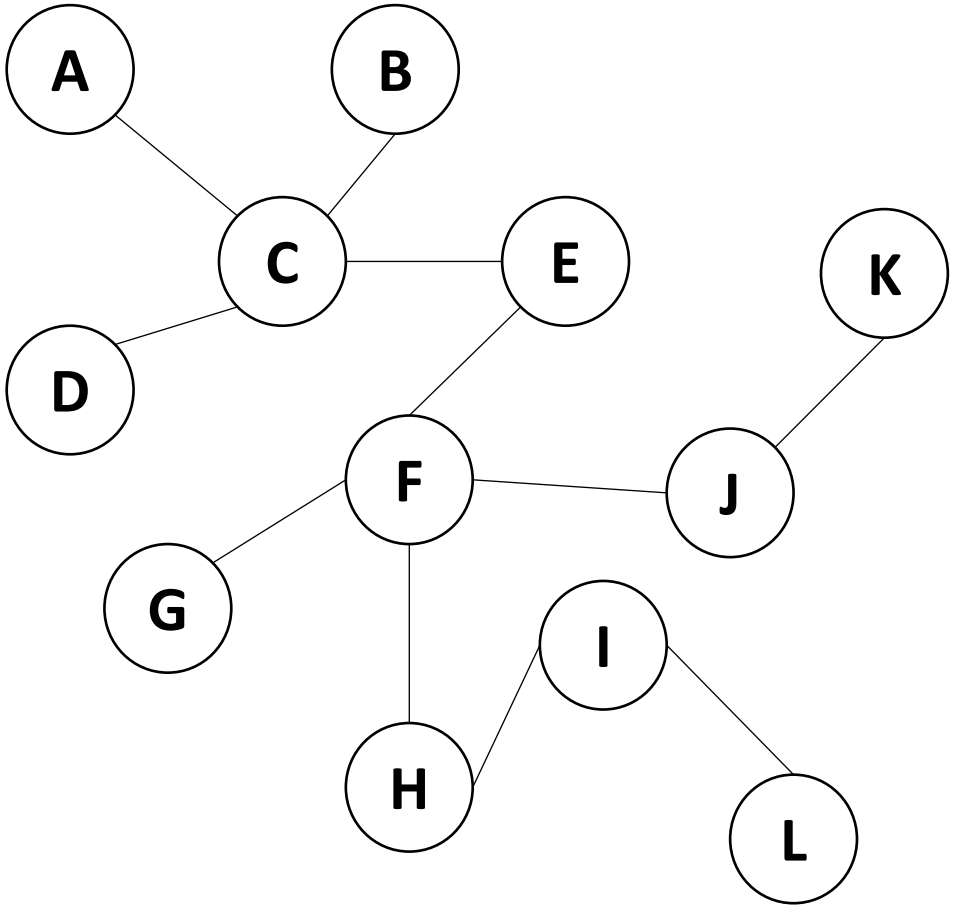


Select any vertex **v1**

```
HashMap<String, LinkedList<String>> adjList
```

```
{ J: [K], A:[C], C:[A,D,B], F: [E, G, H, J], … }
```

HashMaps are unordered, and there is no way to pick a key at an “index”



Select any vertex **v1**

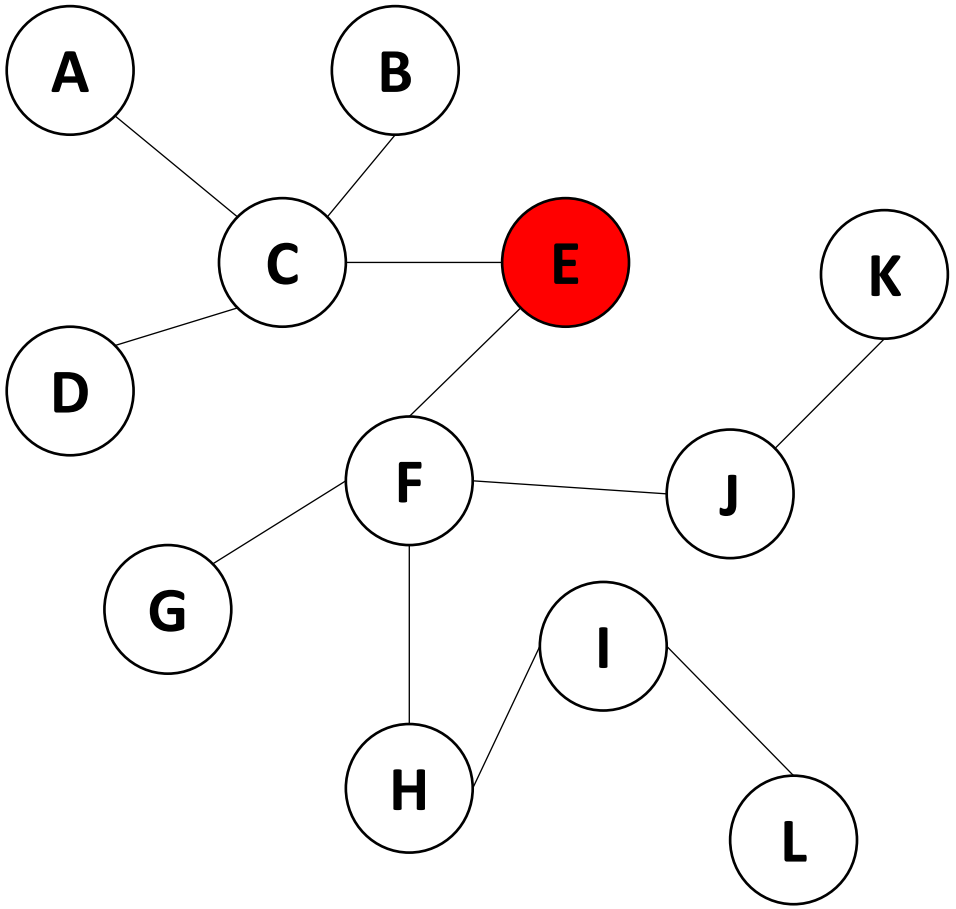
```
HashMap<String, LinkedList<String>> adjList
```

```
{ J: [K], A:[C], C:[A,D,B], F: [E, G, H, J], … }
```

HashMaps are unordered, and there is no way to pick a key at an "index"

Just return the first key when iterating over it

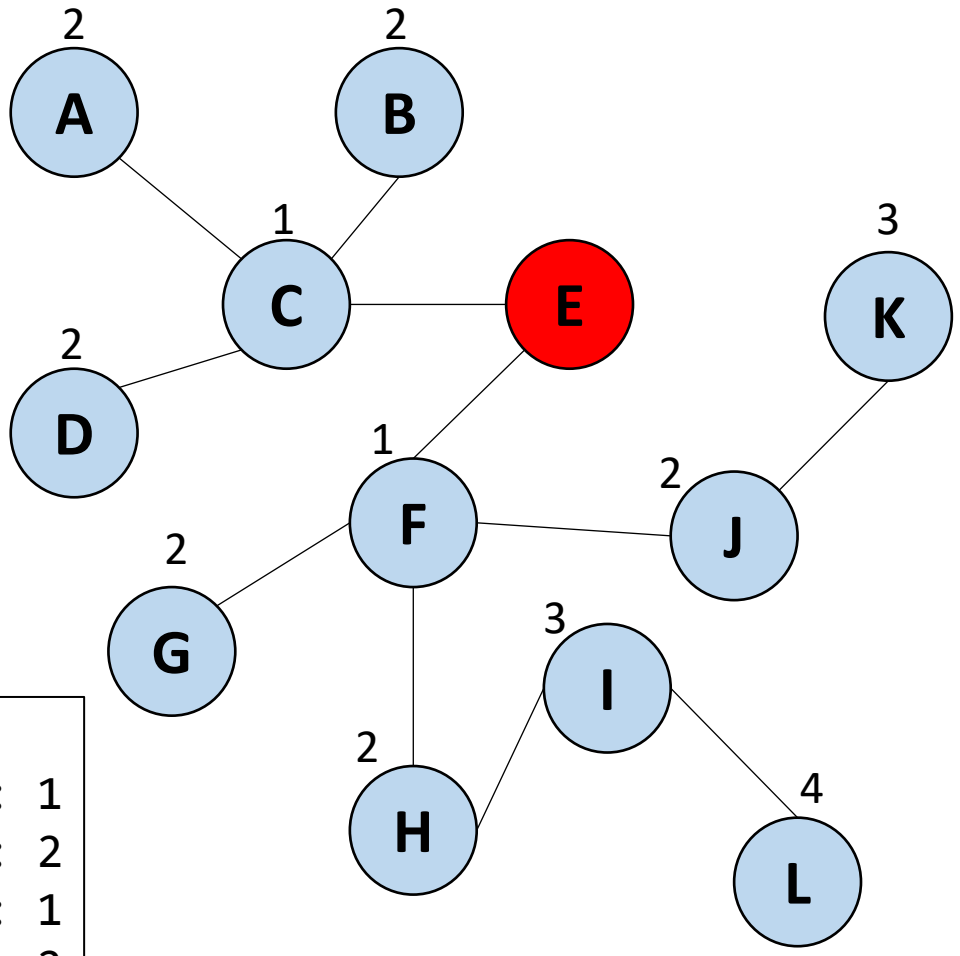
```
for(String key: adjList){
        return key;
}
```



Select any vertex **v1**

Do Breadth First Traversal from vertex **v1** to all other vertices

While doing breadth first, keep track of the distance from vertex **v1**

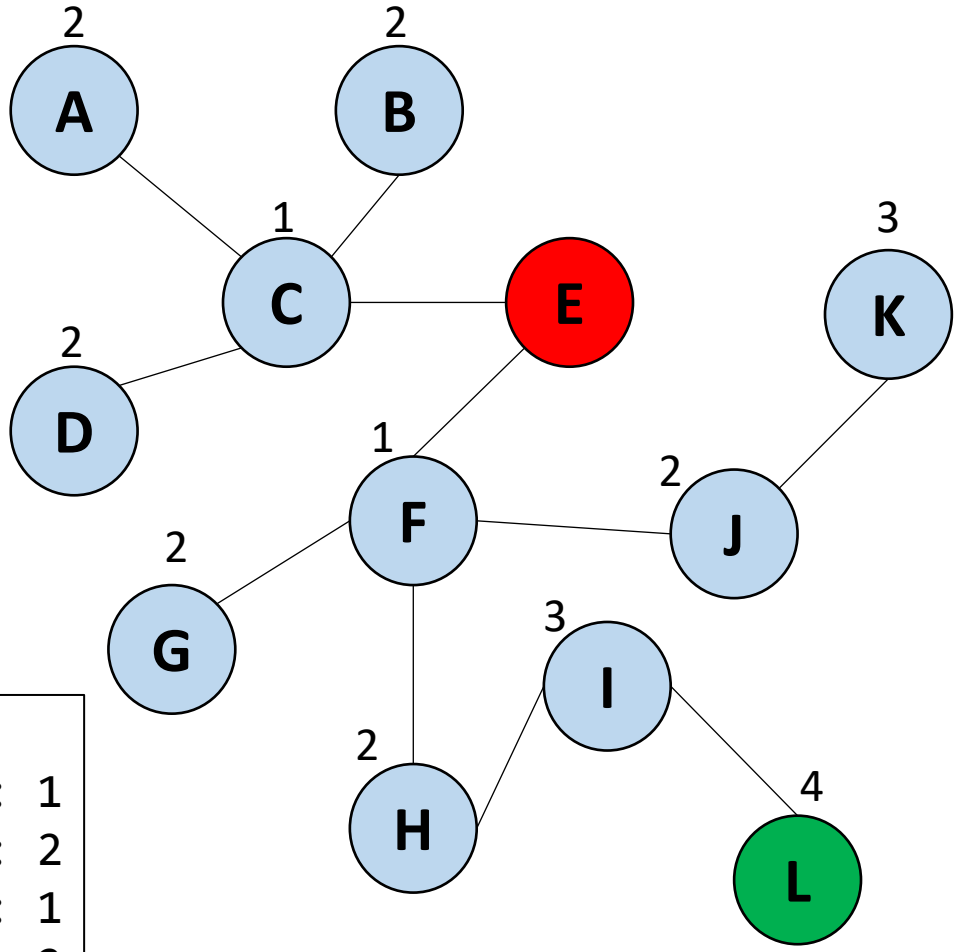



```
{
C: 1
J: 2
F: 1
D: 2
G: 2
L: 4
…

}
```

Select any vertex **v1**

Do Breadth First Traversal from vertex **v1** to all other vertices

While doing breadth first, keep track of the distance from vertex **v1** (store in some kind of data structure)



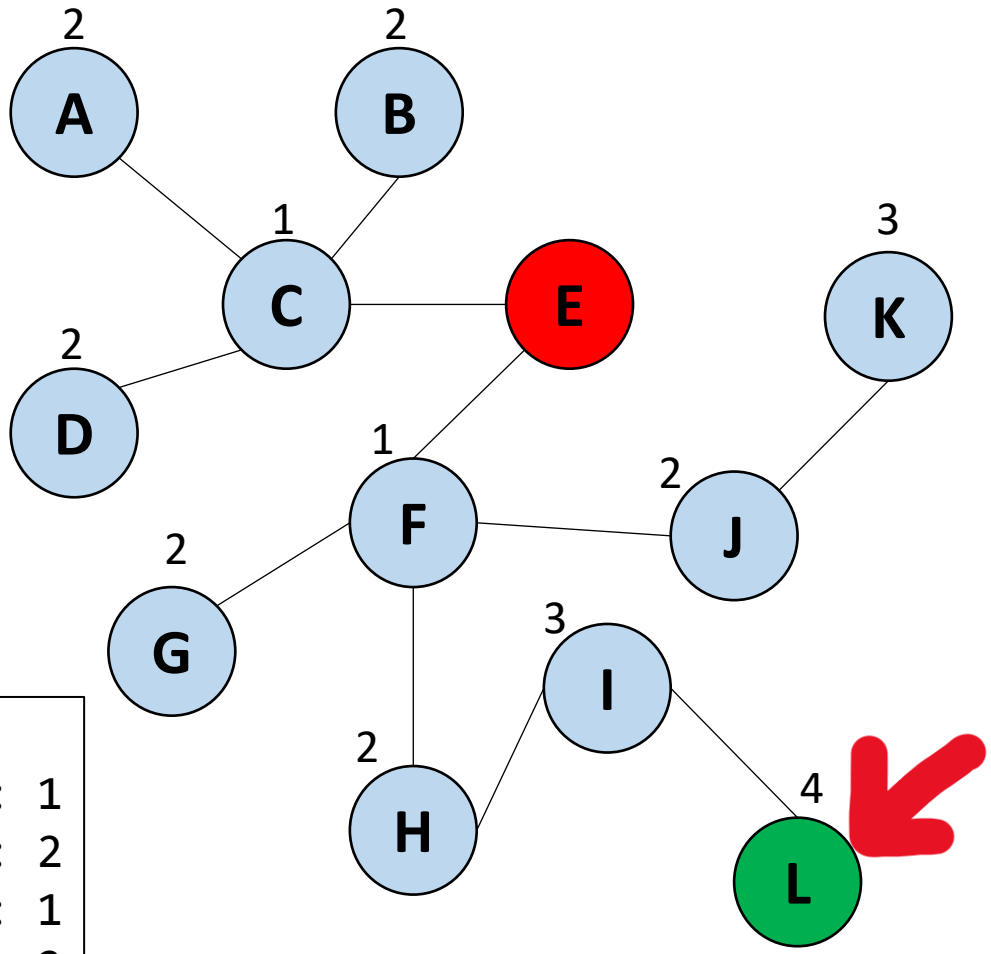

```
{
C: 1
J: 2
F: 1
D: 2
G: 2
L: 4
…

}
```

Select any vertex **v1**

Do Breadth First Traversal from vertex **v1** to all other vertices

While doing breadth first, keep track of the distance from vertex **v1** (store in some kind of data structure)

Select the node that was the furthest away, **v2**



Select any vertex **v1**

Do Breadth First Traversal from vertex **v1** to all other vertices

While doing breadth first, keep track of the distance from vertex **v1** (store in some kind of data structure)

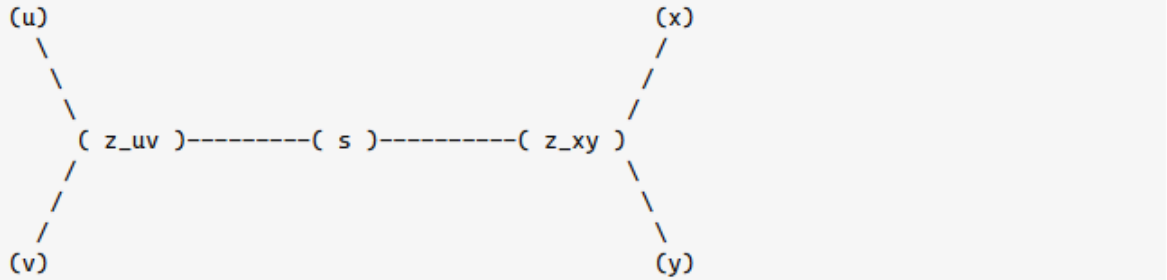
Select the node that was the furthest away, **v2**

```
{
C: 1
J: 2
F: 1
D: 2
G: 2
L: 4
…

}
```

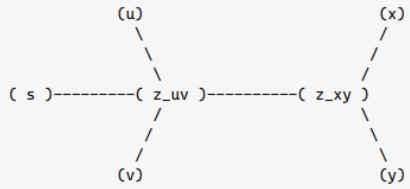
v2 must be an endpoint on the longest path because …

Choose an arbitrary tree node s. Assume $u, v \in V(G)$ are nodes with $d(u,v) = diam(G)$. Assume further that the algorithm finds a node x starting at s first, some node y starting at x next. wlog $d(s,u) \geq d(s,v)$. note that $d(s,x) \geq d(s,y)$ must hold, unless the algorithm's first stage wouldn't end up at x. We will see that $d(x,y) = d(u,v)$.

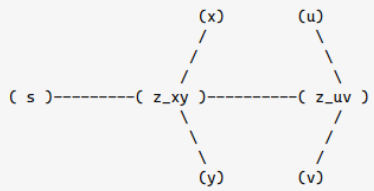
The most general configuration of all nodes involved can be seen in the following pseudo-graphics (possibly $s = z_{uv}$ or $s = z_{xy}$ or both):



analogue proofs hold for the alternative configurations



and



these are all possible configurations. in particular, $x \notin path(s,u), x \notin path(s,v)$ due to the result of stage 1 of the algorithm and $y \notin path(x,u), y \notin path(x,v)$ due to stage 2.

we know that:

1. $d(z_{uv}, y) \leq d(z_{uv}, v)$. otherwise $d(u,v) < diam(G)$ contradicting the assumption.
2. $d(z_{uv}, x) \leq d(z_{uv}, u)$. otherwise $d(u,v) < diam(G)$ contradicting the assumption.
3. $d(s, z_{xy}) + d(z_{xy}, x) \geq d(s, z_{uv}) + d(z_{uv}, u)$, otherwise stage 1 of the algorithm wouldn't have stopped at x.
4. $d(z_{xy}, y) \geq d(v, z_{uv}) + d(z_{uv}, z_{xy})$, otherwise stage 2 of the algorithm wouldn't have stopped at y.

1) and 2) imply

$$\begin{aligned} d(u,v) &= d(z_{uv},v) + d(z_{uv},u) \\ &\geq d(z_{uv},x) + d(z_{uv},y) = d(x,y) + 2\,d(z_{uv},z_{xy}) \\ &\geq d(x,y) \end{aligned}$$

3) and 4) imply

$$\begin{aligned} d(z_{xy},y) + d(s,z_{xy}) + d(z_{xy},x) \\ \geq d(s,z_{uv}) + d(z_{uv},u) + d(v,z_{uv}) + d(z_{uv},z_{xy}) \end{aligned}$$

equivalent to

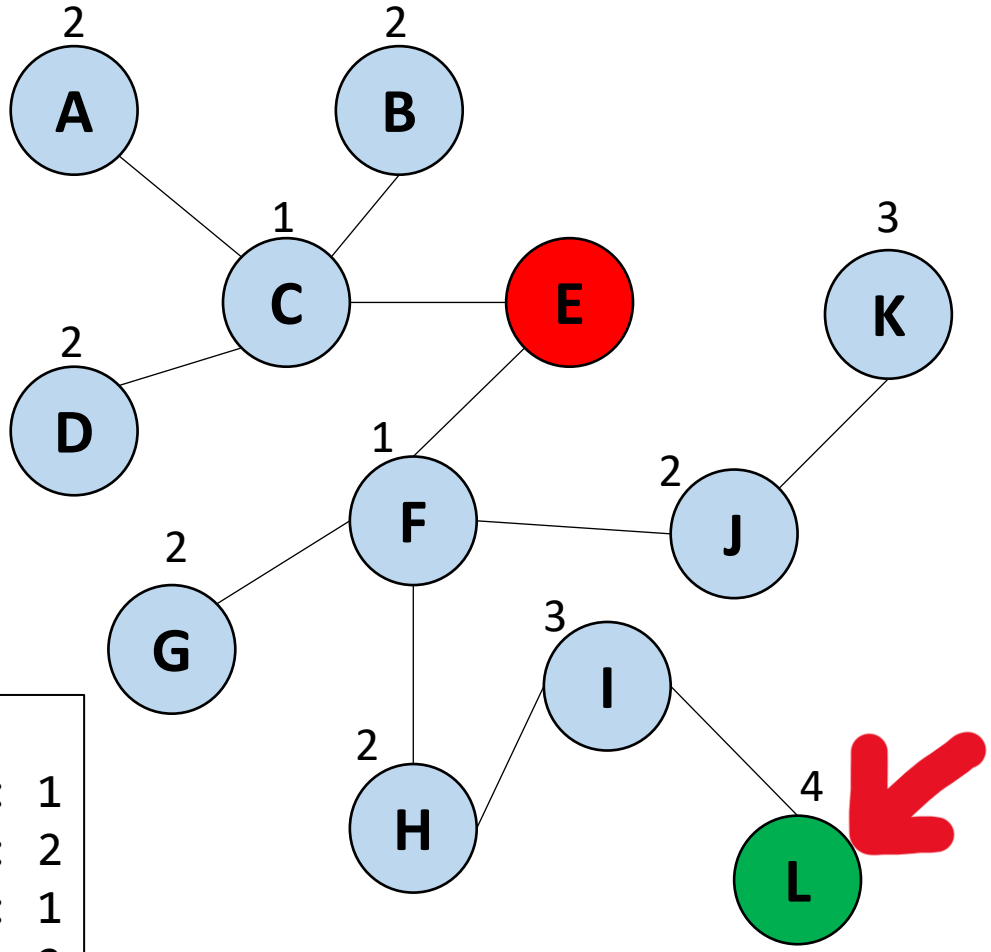
$$\begin{aligned} d(x,y) &= d(z_{xy},y) + d(z_{xy},x) \\ &\geq 2 * d(s,z_{uv}) + d(v,z_{uv}) + d(u,z_{uv}) \\ &\geq d(u,v) \end{aligned}$$

therefore $d(u,v) = d(x,y)$.

```
{
C: 1
J: 2
F: 1
D: 2
G: 2
L: 4
…

}
```

v2 mu … st path because reese told you so

ay, **v2**



Select any vertex **v1**

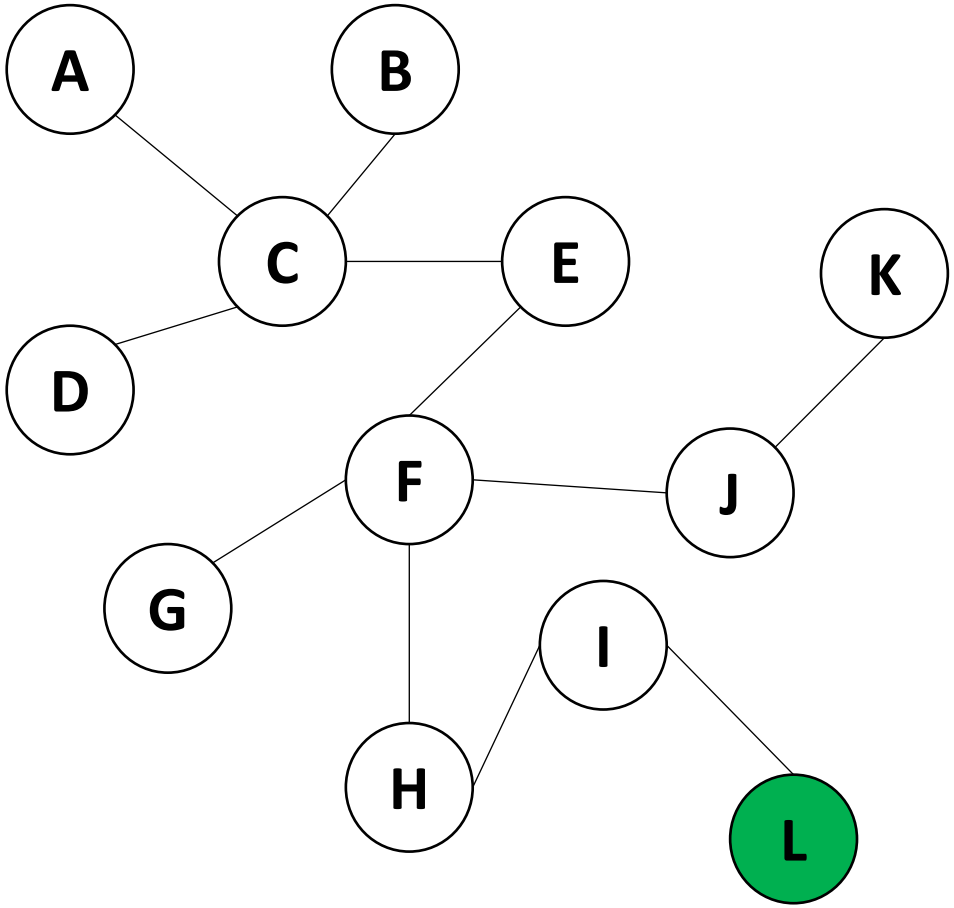
Do Breadth First Traversal from vertex **v1** to all other vertices

While doing breadth first, keep track of the distance from vertex **v1** (store in some kind of data structure)

Select the node that was the furthest away, **v2**

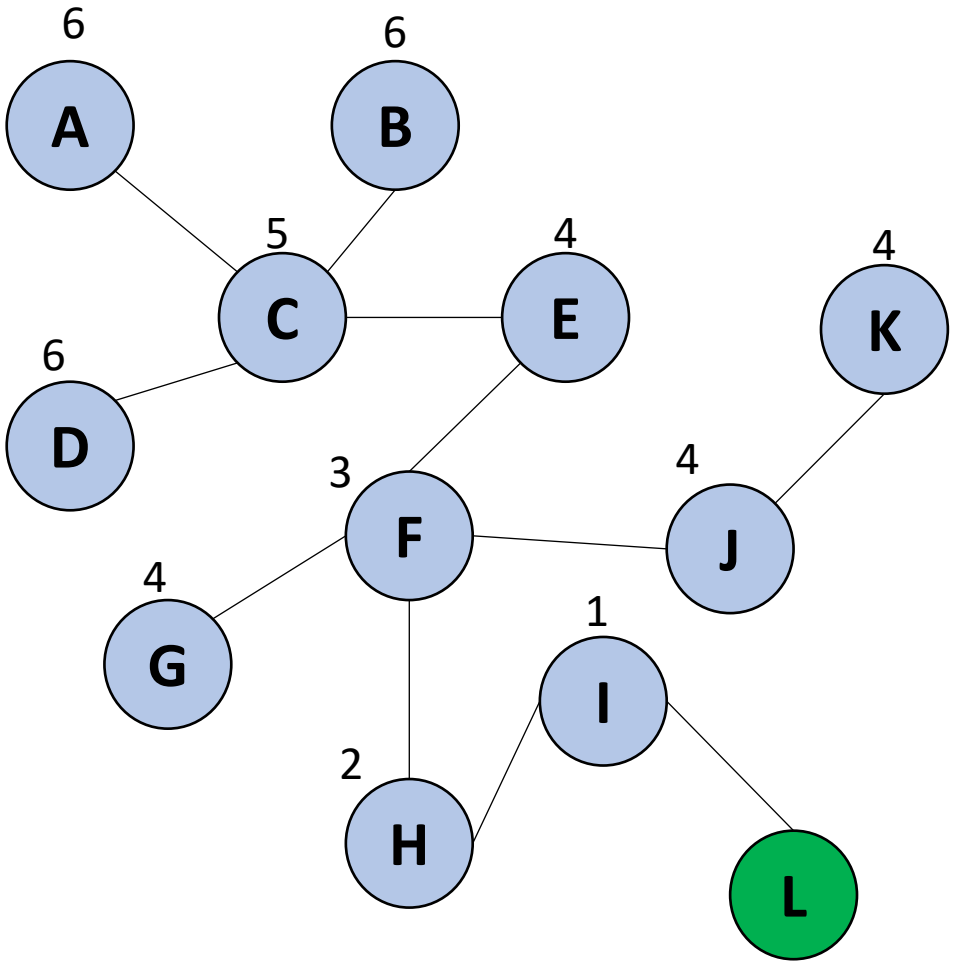
```
{
C: 1
J: 2
F: 1
D: 2
G: 2
L: 4
…

}
```

v2 must be an endpoint on the longest path because otherwise BFS would have found a deeper vertex



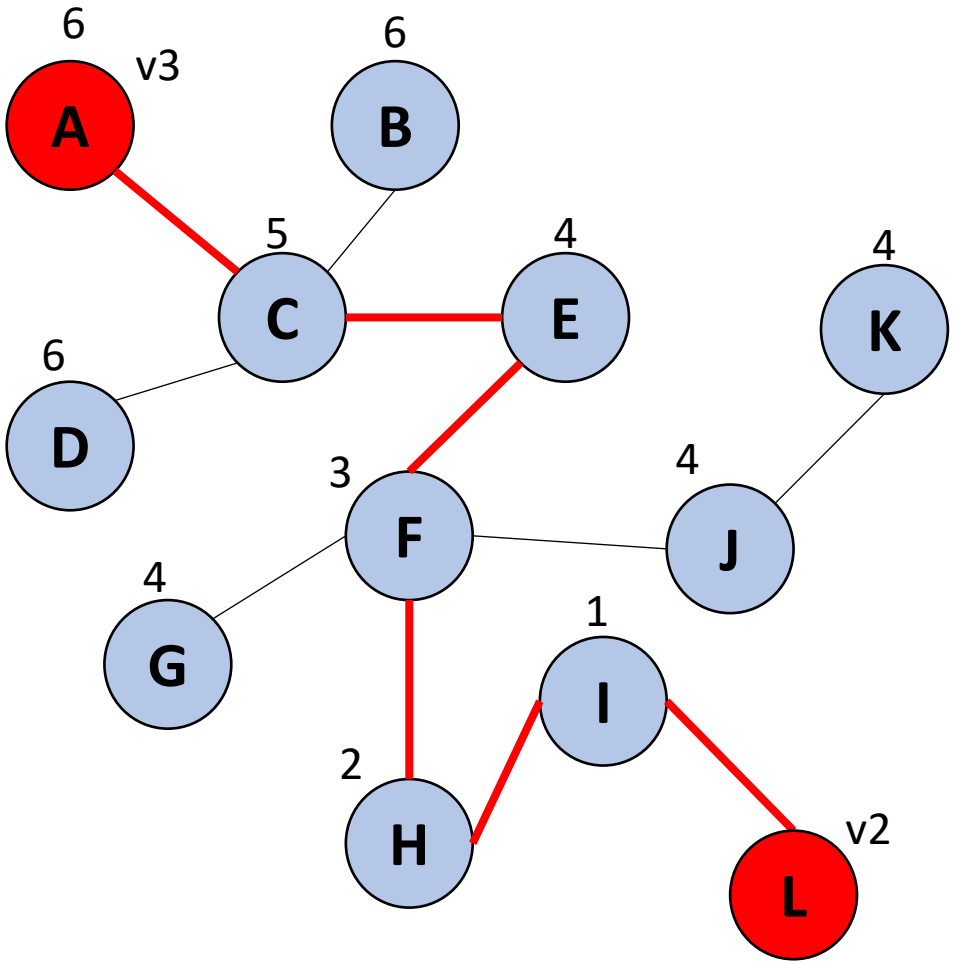
Do breadth first search again, but now starting from **v2**

Keep track of distances from **v2**



Do breadth first search again, but now starting from **v2**

Keep track of distances from **v2**



"Double pass BFS"

Will only work on an acyclic graph

Do breadth first search again, but now starting from **v2**

Keep track of distances from **v2**

Select the vertex with the longest distance, **v3**

(We have a tie for longest path, so just select one of them)

Breadth First will visit every node, and will always find the farthest away node from some starting point

[V2, V3] is the longest path