Connectedness

Outline

We will use graph traversals to determine:

- Whether one vertex is connected to another
- The connected sub-graphs of a graph

Connected

First, let us determine whether one vertex is connected to another

- v_j is connected to v_k if there is a path from the first to the second

Strategy:

- Perform a breadth-first traversal starting at v_j
- While looping, if the vertex v_k ever found to be adjacent to the front of the queue, return true
- If the loop ends, return false

Connected

Consider implementing a breadth-first traversal on a graph:

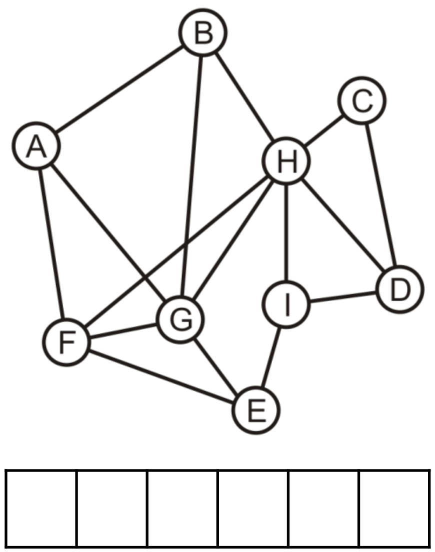
- Choose any vertex, mark it as visited and push it onto queue
- While the queue is not empty:
  - Pop to top vertex v from the queue
  - For each vertex adjacent to v that has not been visited:
    - Mark it visited, and
    - Push it onto the queue

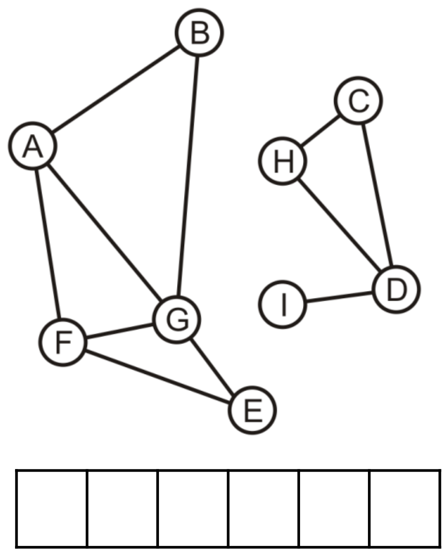
This continues until the queue is empty

- Note: if there are no unvisited vertices, the graph is connected,

Determining Connections

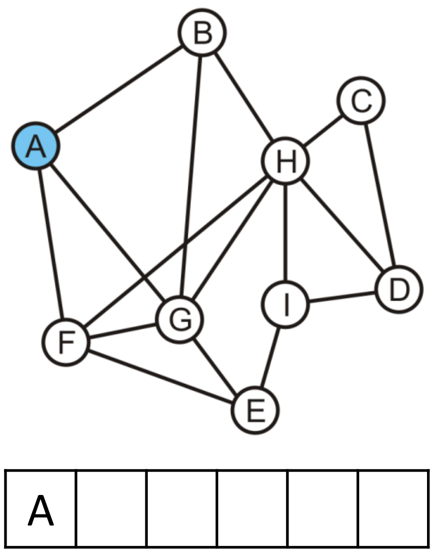
Is A connected to D?

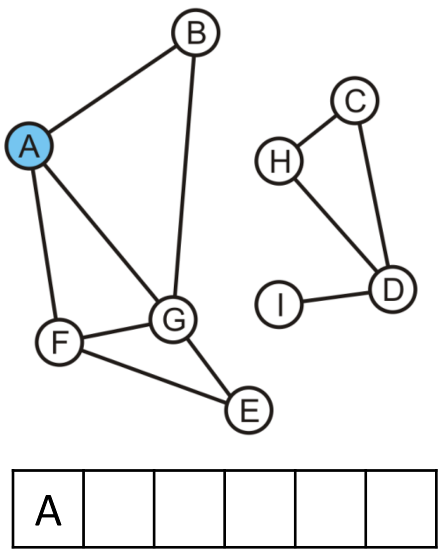




Determining Connections

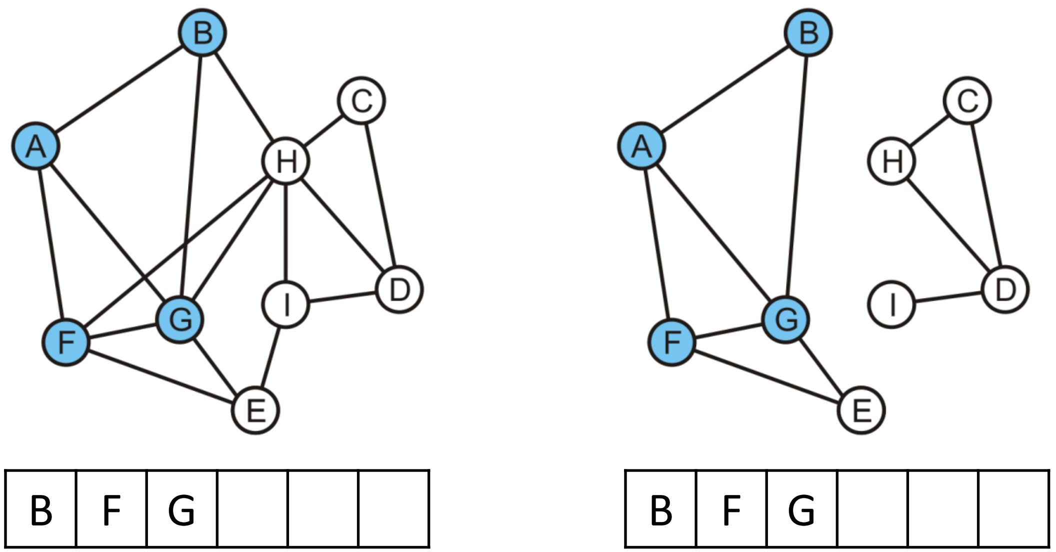
Vertex A is marked as visited and pushed onto the queue





Determining Connections

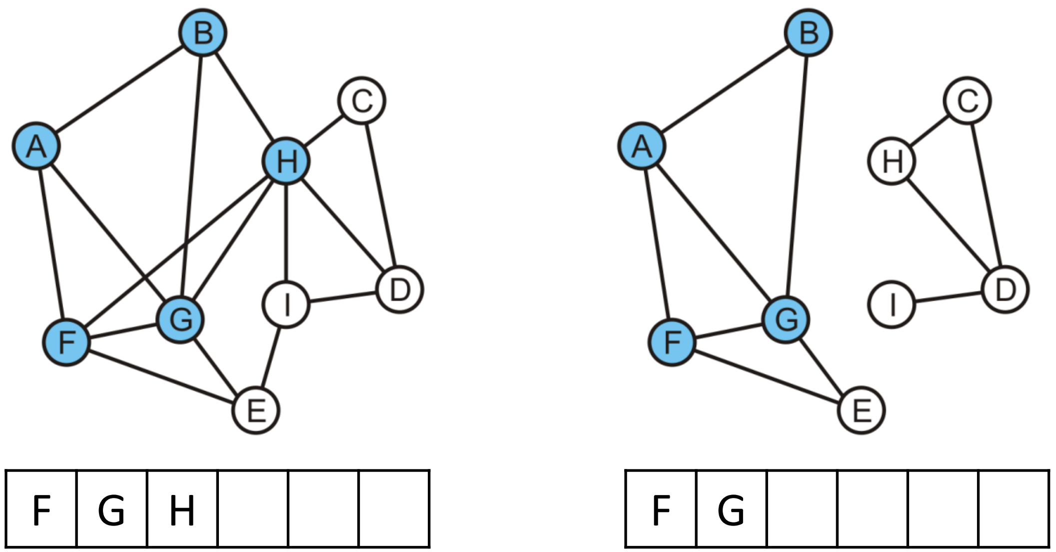
Pop the head, A, and mark and push B, F and G



Determining Connections

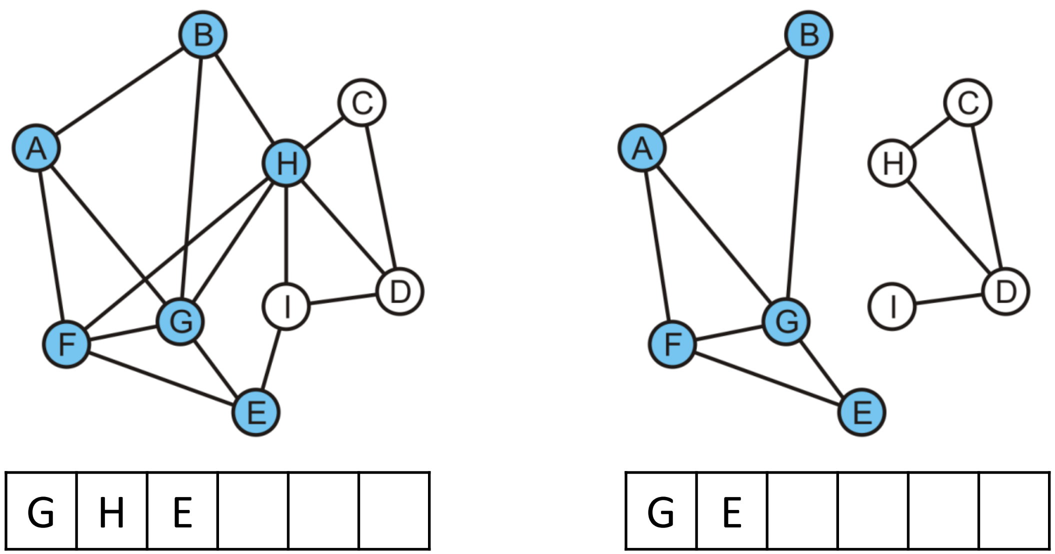
Pop B and mark and, in the left graph, mark and push H

- On the right graph, B has no unvisited adjacent vertices



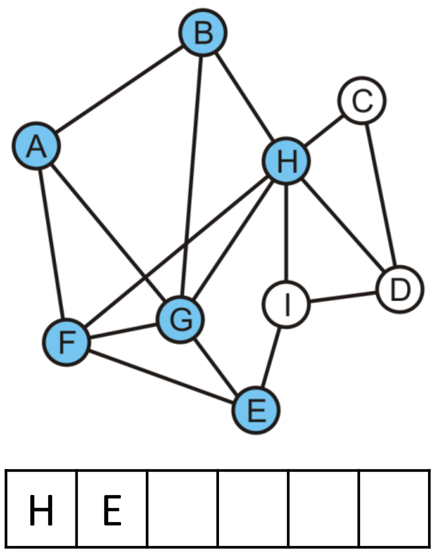
Determining Connections

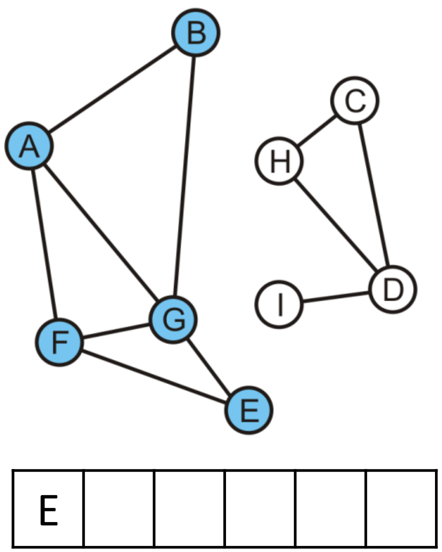
Popping F results in the pushing of E



Determining Connections

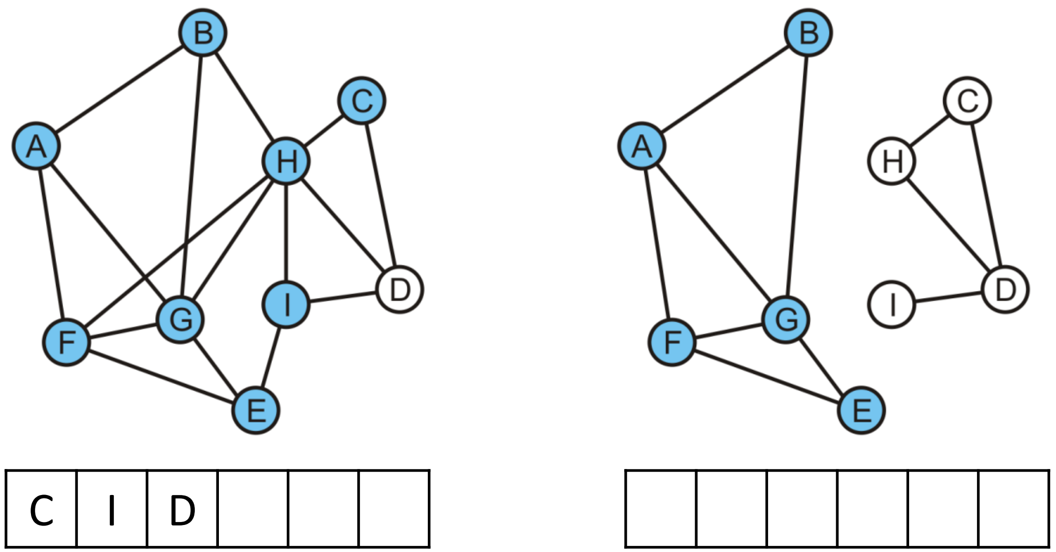
In either graph, G has no adjacent vertices that are unvisited





Determining Connections

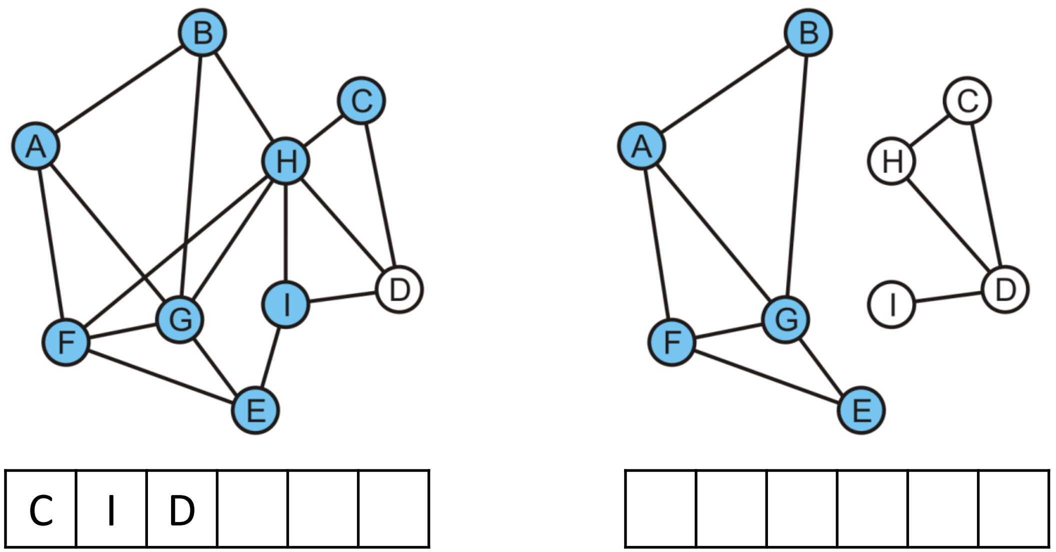
Popping H on the left graph results in C and I being pushed



Determining Connections

The queue op the right is empty

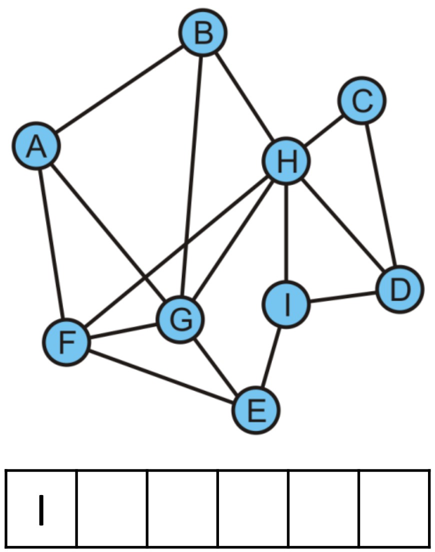
- We determine A is not connected to D

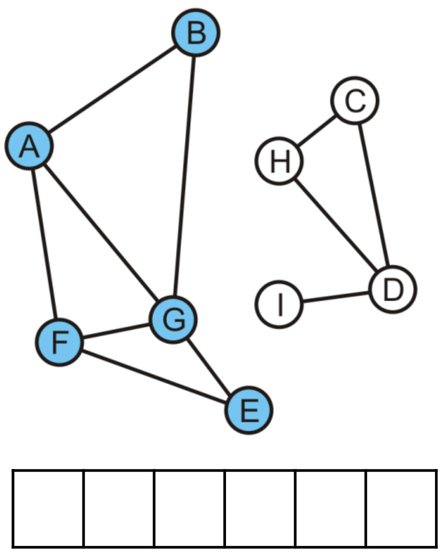


Determining Connections

On the left, we pop C and return `true` because D is adjacent to C

- In the left graph, A is connected to D

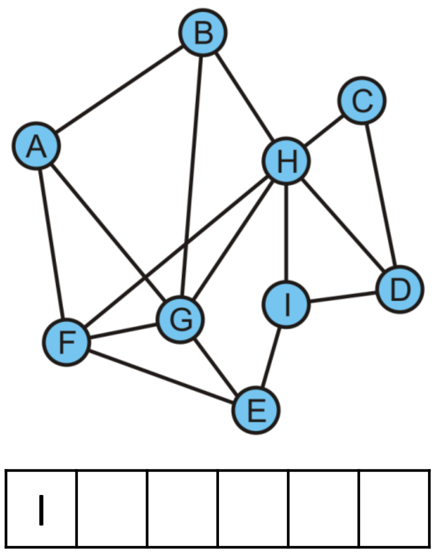


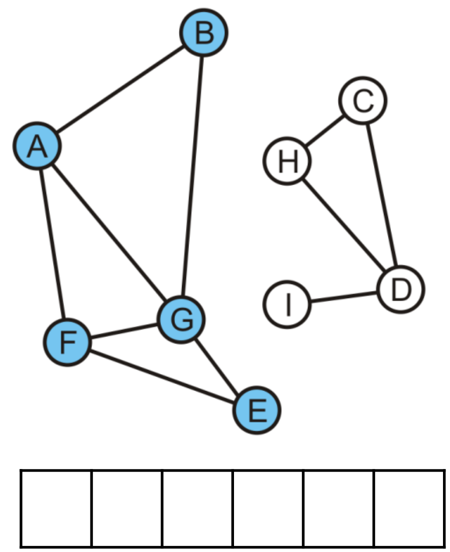


Determining Connections

On the left, we pop C and return `true` because D is adjacent to C

- In the left graph, A is connected to D





Connected Components

If we continued the traversal, we would find all vertices that are connected to A

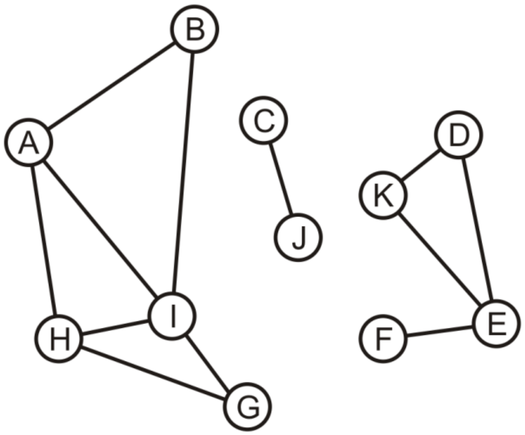
Suppose we want to partition the vertices into connected sub-graphs

- While there are unvisited vertices in the tree:
  - Select an unvisited vertex and perform a traversal on that vertex
  - Each vertex that is visited in that traversal is added to the set initially containing the initial unvisited vertex
- Continue until all vertices are visited

We would use a disjoint set data structure for maximum efficiency

Connected Components

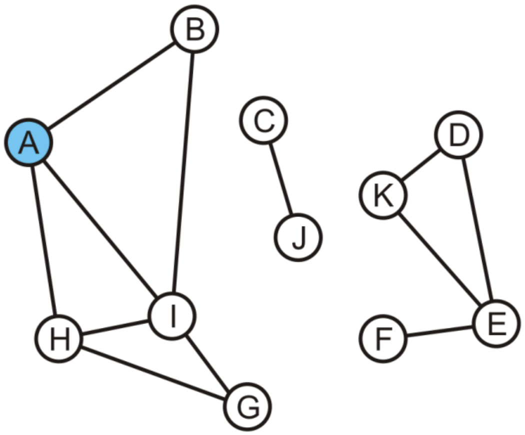
Here we start with a set of singletons



A	B	C	D	E	F	G	H	I	J	K
A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**

Connected Components

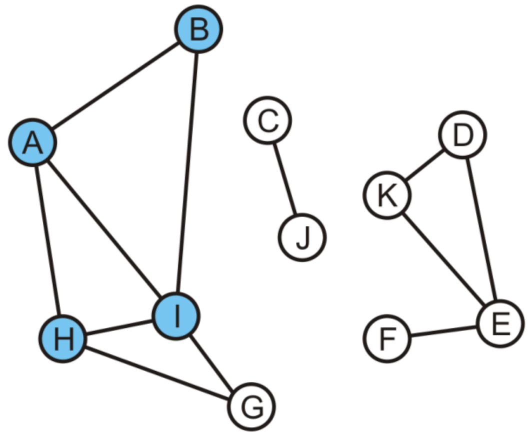
The vertex A is unvisited, so we start with it



A	B	C	D	E	F	G	H	I	J	K
A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**

Connected Components

Take the union of with its adjacent vertices: {A, B, H, I}

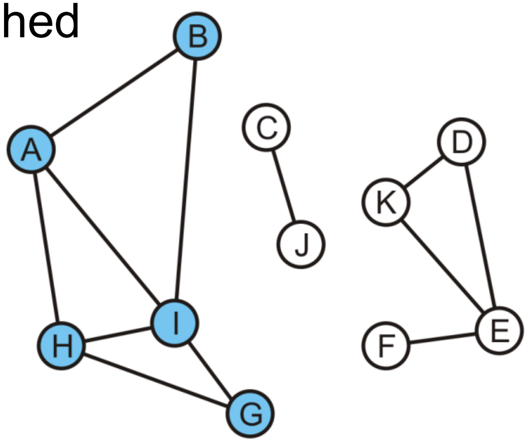


A	B	C	D	E	F	G	H	I	J	K
A	**A**	**C**	**D**	**E**	**F**	**G**	**A**	**A**	**J**	**K**

Connected Components

As the traversal continues, we take the union of the set {G} with the set containing H: {A, B, G, H, I}

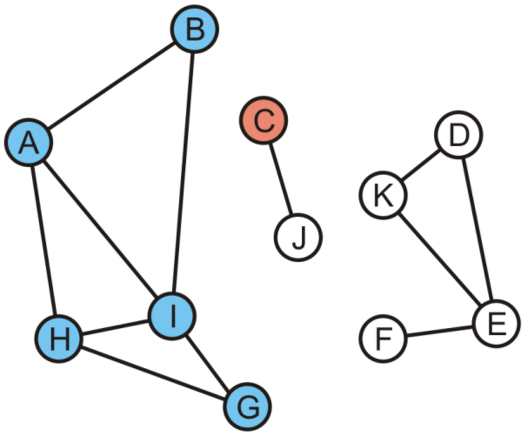
– The traversal is finished



A	B	C	D	E	F	G	H	I	J	K
A	**A**	**C**	**D**	**E**	**F**	**A**	**A**	**A**	**J**	**K**

Connected Components

Start another traversal with C: this defines a new set $\{C\}$

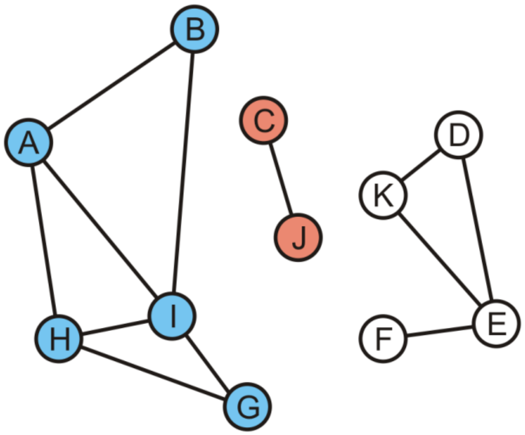


A	B	C	D	E	F	G	H	I	J	K
A	**A**	**C**	**D**	**E**	**F**	**A**	**A**	**A**	**J**	**K**

Connected Components

We take the union of {C} and its adjacent vertex J: {C, J}

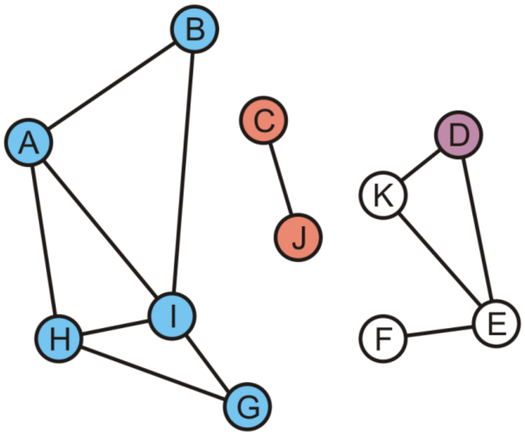
- This traversal is finished



A	B	C	D	E	F	G	H	I	J	K
A	**A**	**C**	**D**	**E**	**F**	**A**	**A**	**A**	**C**	**K**

Connected Components

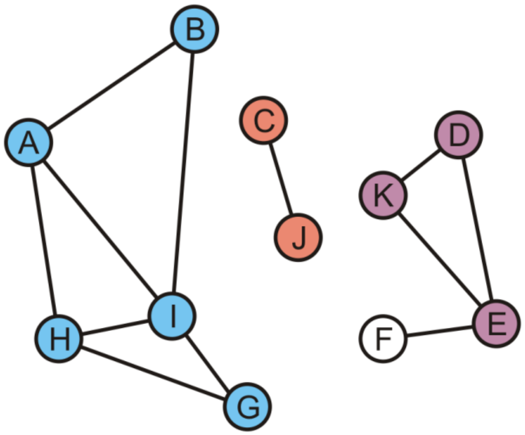
We start again with the set {D}



A	B	C	D	E	F	G	H	I	J	K
A	**A**	**C**	**D**	**E**	**F**	**A**	**A**	**A**	**C**	**K**

Connected Components

K and E are adjacent to D, so take the unions creating {D, E, K}

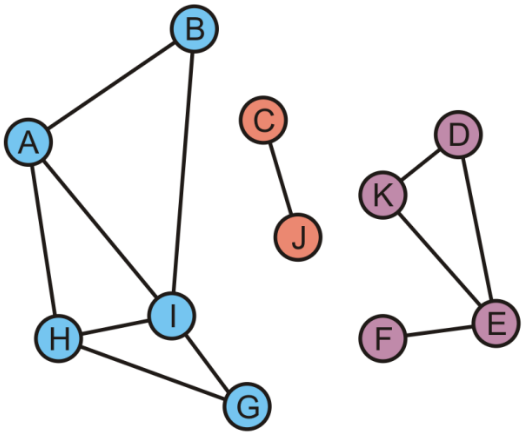


A	B	C	D	E	F	G	H	I	J	K
A	**A**	**C**	**D**	**D**	**F**	**A**	**A**	**A**	**C**	**D**

Connected Components

Finally, during this last traversal we find that F is adjacent to E

- Take the union of {F} with the set containing E: {D, E, F, K}

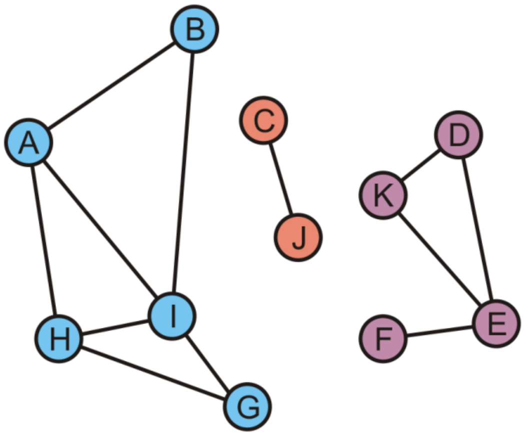


A	B	C	D	E	F	G	H	I	J	K
A	**A**	**C**	**D**	**D**	**D**	**A**	**A**	**A**	**C**	**D**

Connected Components

All vertices are visited, so we are done

– There are three connected sub-graphs {A, B, G, H, I}, {C, J}, {D, E, F, K}



A	B	C	D	E	F	G	H	I	J	K
A	A	C	D	D	D	A	A	A	C	D

Summary

This topic covered connectedness

- Determining if two vertices are connected
- Determining the connected sub-graphs of a graph
- Tracking unvisited vertices

References

Wikipedia, http://en.wikipedia.org/wiki/Connectivity_(graph_theory)