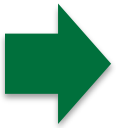
CS 10:
Problem solving via Object Oriented Programming

Graph Traversals

Main goals

- Implement graph traversals that does not take into account cost

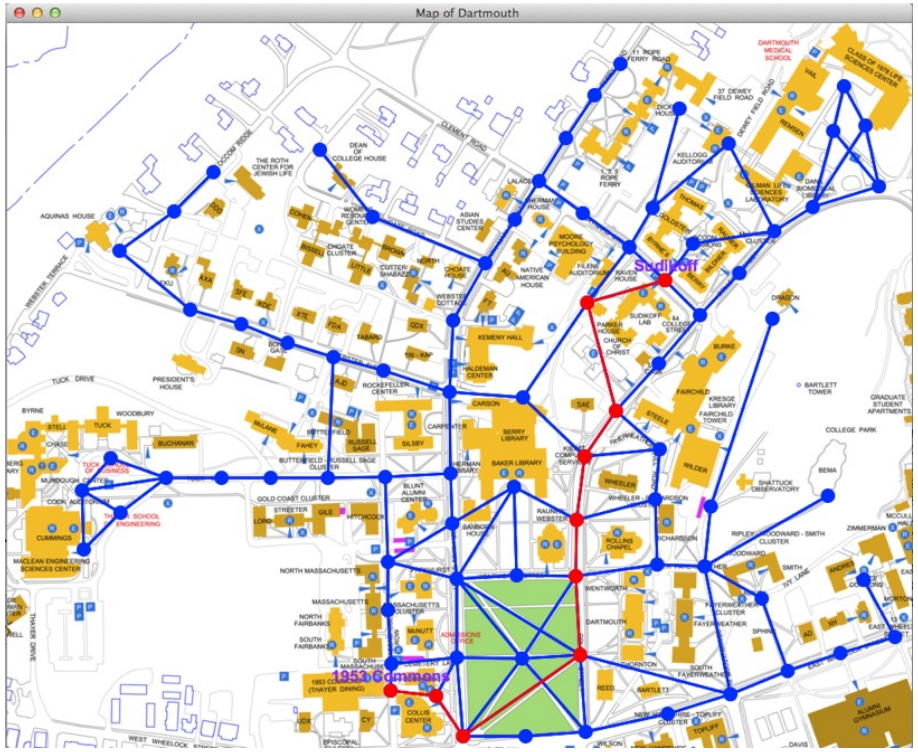
Agenda



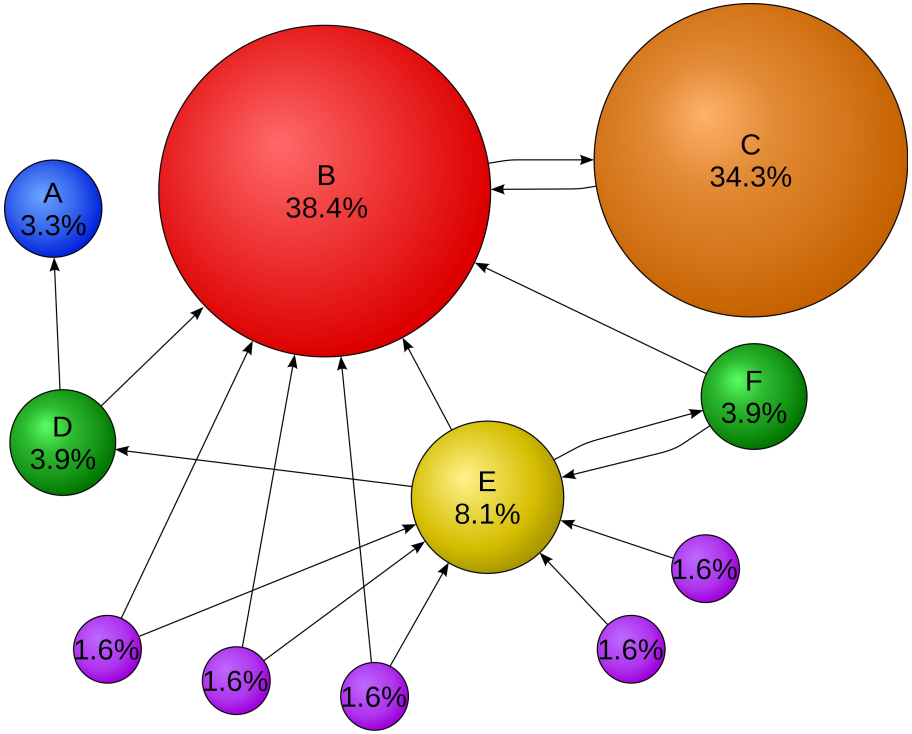
1. Depth first search (DFS)
2. Breadth first search (BFS)
3. Examples from last class and today

Graph traversals are useful to answer questions about relationships and reachability

Some examples

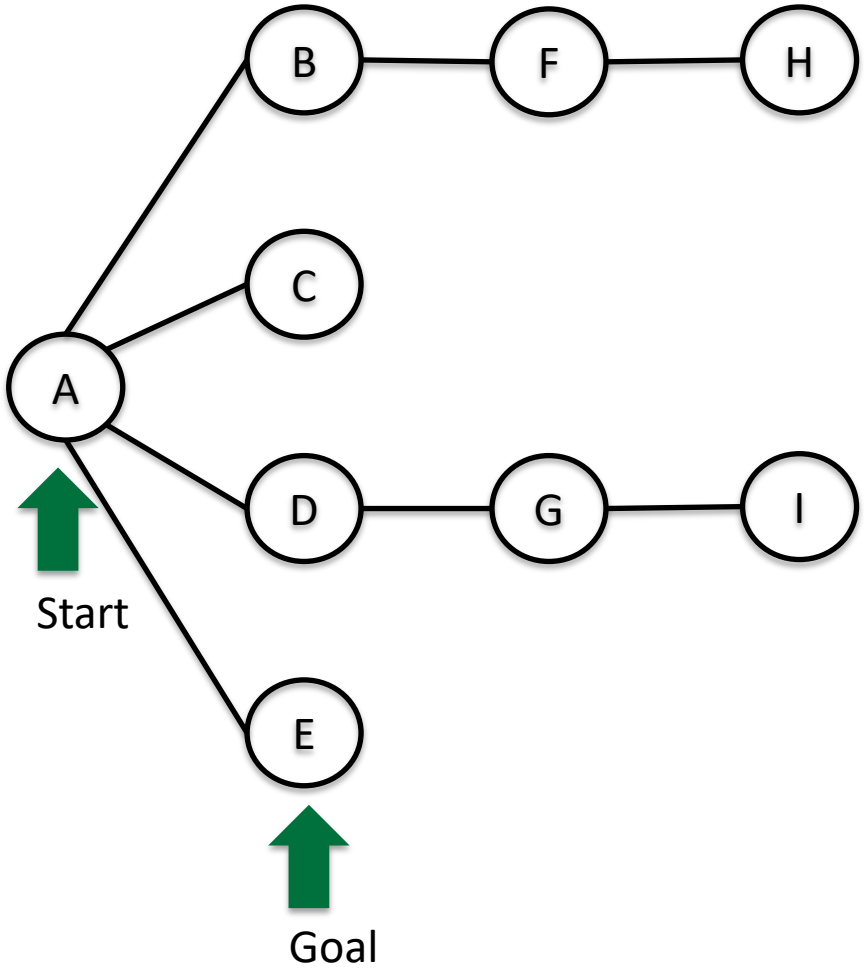


Map (CS1)



Page Rank, source: wikipedia

Depth First Search (DFS) basic idea



- Keep going until you can't go any further, then back track
- Relies on a **Stack** (implicit or explicit) to keep track of where you've been

Graph structure from http://stackoverflow.com/questions/687731/breadth-first-vs-depth-first

Some of you did Depth First Search on Problem Set 1

RegionFinder pseudo code

```
Loop over all the pixels
    If a pixel is unvisited and of the correct color
        Start a new region
        Keep track of pixels need to be visited, initially just one
        As long as there's some pixel that needs to be visited
            Get one to visit
            Add it to the region
            Mark it as visited
            Loop over all its neighbors
                If the neighbor is of the correct color
                    Add it to the list of pixels to be visited
        If the region is big enough to be worth keeping, do so
```

Some of you did Depth First Search on Problem Set 1

RegionFinder pseudo code

```
Loop over all the pixels
    If a pixel is unvisited and of the correct color
        Start a new region
        Keep track of pixels need to be visited, initially just one
        As long as there's some pixel that needs to be visited
            Get one to visit
            Add it to the region
            Mark it as visited
            Loop over all its neighbors
                If the neighbor is of the correct color
                    Add it to the list of pixels to be visited
        If the region is big enough to be worth keeping, do so
```

If you added to end of list…

Some of you did Depth First Search on Problem Set 1

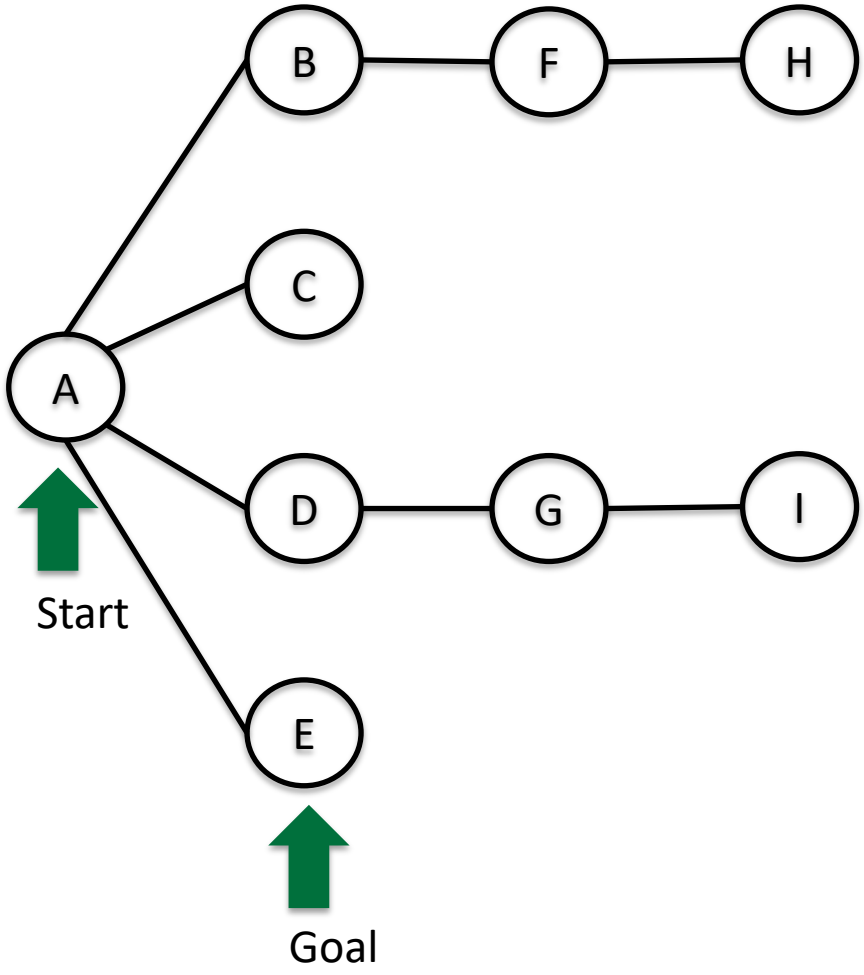
RegionFinder pseudo code

```
Loop over all the pixels
    If a pixel is unvisited and of the correct color
        Start a new region
        Keep track of pixels need to be visited, initially just one
        As long as there's some pixel that needs to be visited
            Get one to visit
            Add it to the region
            Mark it as visited
            Loop over all its neighbors
                If the neighbor is of the correct color
                    Add it to the list of pixels to be visited
        If the region is big enough to be worth keeping, do so
```

And if you get a pixel from end of list, you implemented a stack

If you added to end of list…

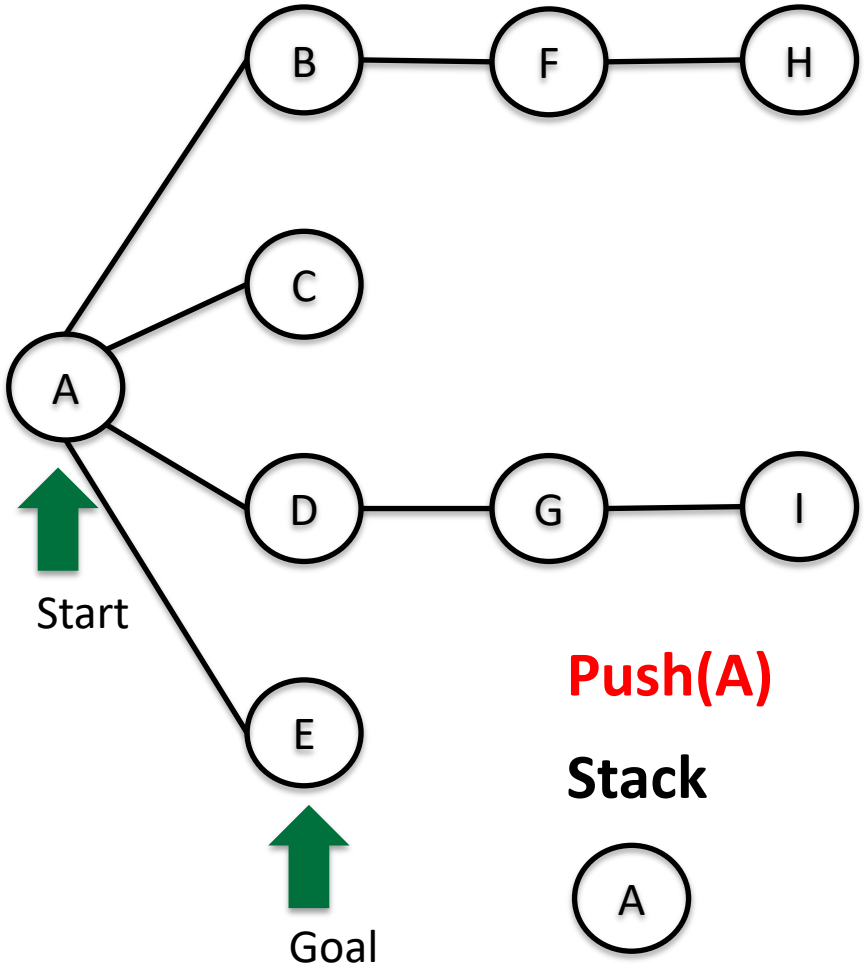
Depth First Search (DFS) is like exploring a maze



DFS algorithm

```
stack.push(s) //start node
repeat until find goal vertex or
stack empty:
    u = stack.pop()
    if !u.visited
        u.visited = true
        (do something while here)
        for v ∈ u.adjacent
              if !v.visited
                stack.push(v)
```

Depth First Search (DFS) is like exploring a maze



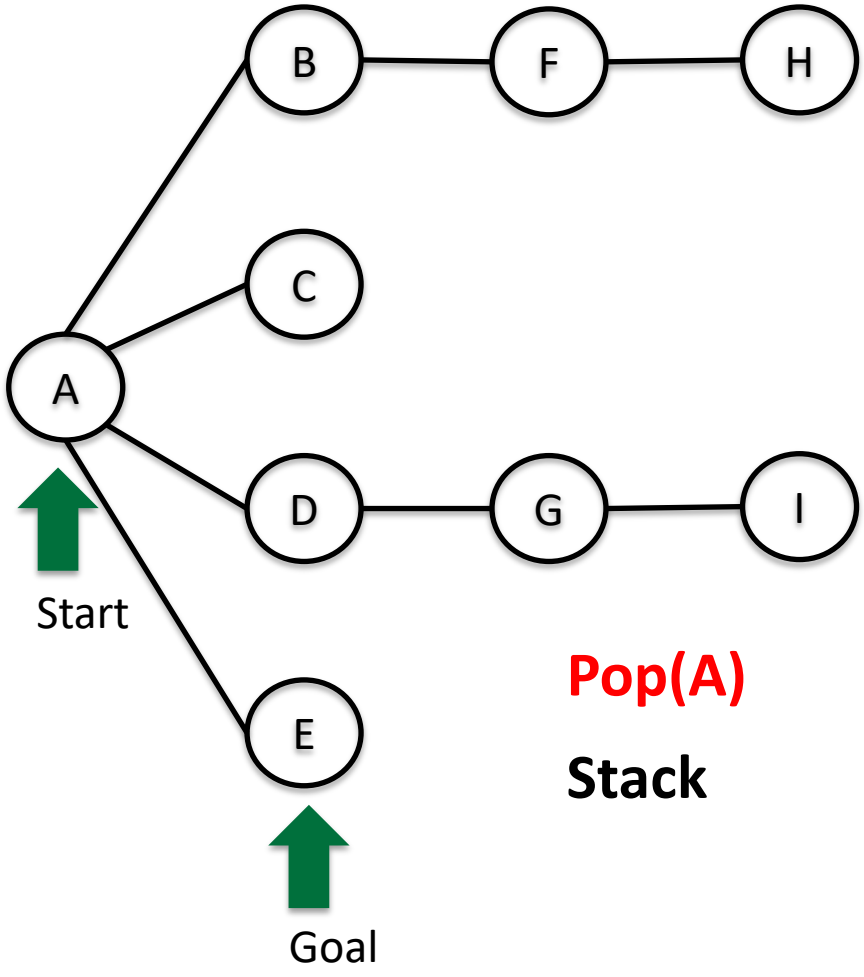
Push(A)

Stack

DFS algorithm

```
stack.push(s) //start node
repeat until find goal vertex or
stack empty:
    u = stack.pop()
    if !u.visited
        u.visited = true
        (do something while here)
        for v ∈ u.adjacent
              if !v.visited
                stack.push(v)
```

Depth First Search (DFS) is like exploring a maze



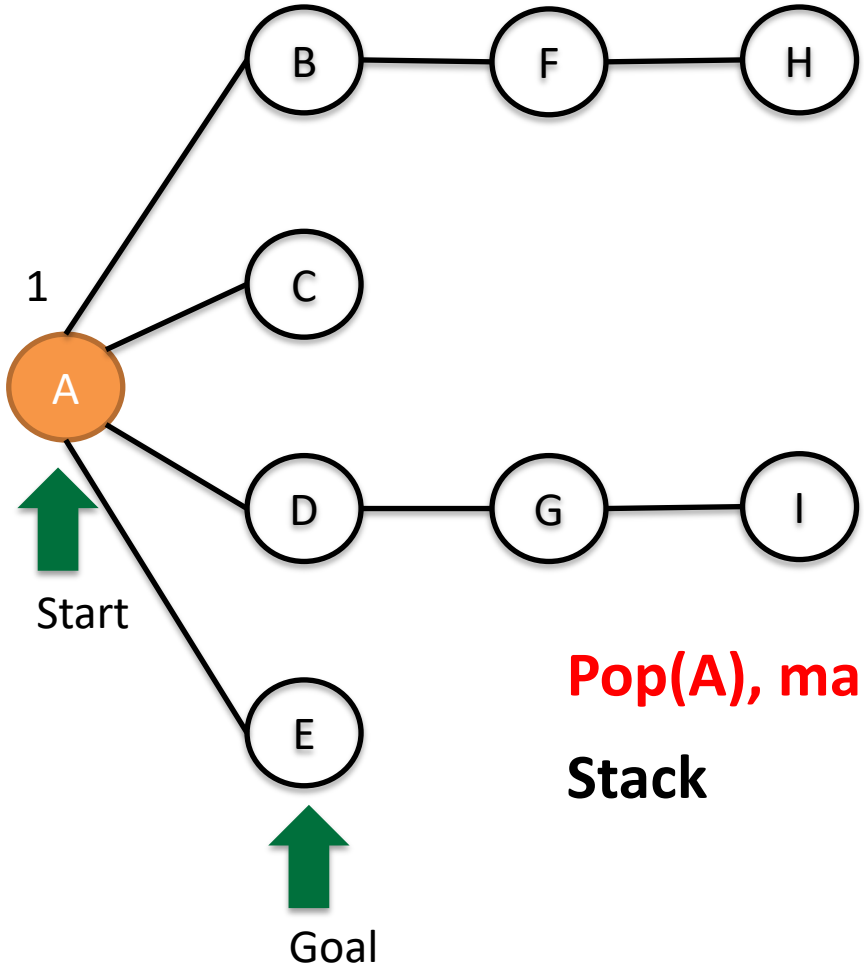
Pop(A)

Stack

DFS algorithm

```
stack.push(s) //start node
repeat until find goal vertex or
stack empty:
    u = stack.pop()
    if !u.visited
        u.visited = true
        (do something while here)
        for v ∈ u.adjacent
            if !v.visited
              stack.push(v)
```

Depth First Search (DFS) is like exploring a maze



DFS algorithm

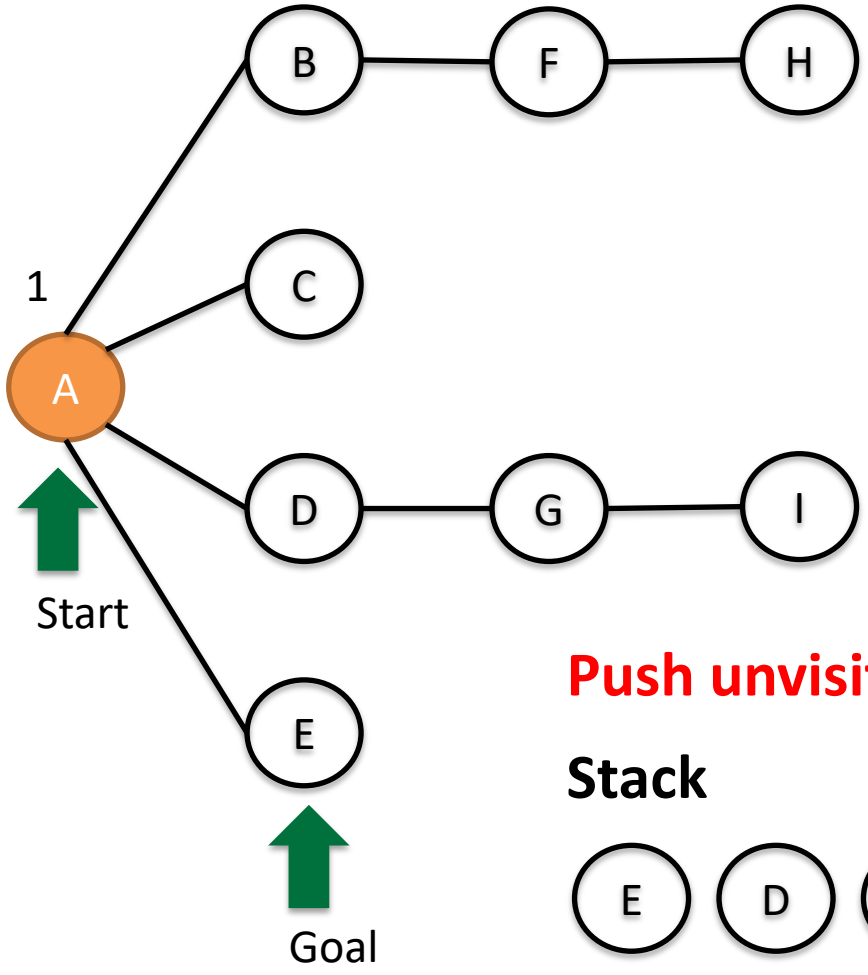
```
stack.push(s) //start node
repeat until find goal vertex or
stack empty:
    u = stack.pop()
    if !u.visited
      u.visited = true
      (do something while here)
      for v ∈ u.adjacent
            if !v.visited
              stack.push(v)
```

"Visit" the node

Pop(A), mark visited

Stack

Depth First Search (DFS) is like exploring a maze



DFS algorithm

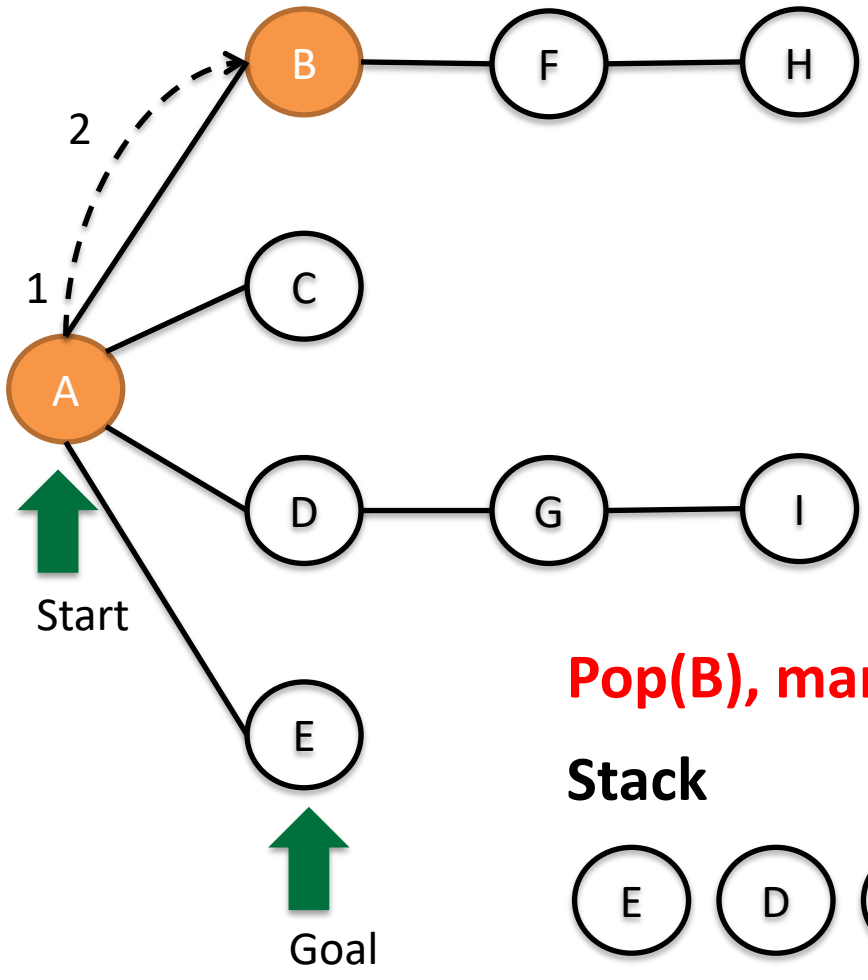
```
stack.push(s) //start node
repeat until find goal vertex or
stack empty:
    u = stack.pop()
    if !u.visited
        u.visited = true
        (do something while here)
        for v ∈ u.adjacent
            if !v.visited
              stack.push(v)
```

Push unvisited adjacent

Stack

- **What method would we use on our AdjacencyMapGraph?**
- ***graph.outNeighbors(u)***
- **Order pushed onto stack depends on order of nodes from outNeighbors iterator**

Depth First Search (DFS) is like exploring a maze



Order nodes visited

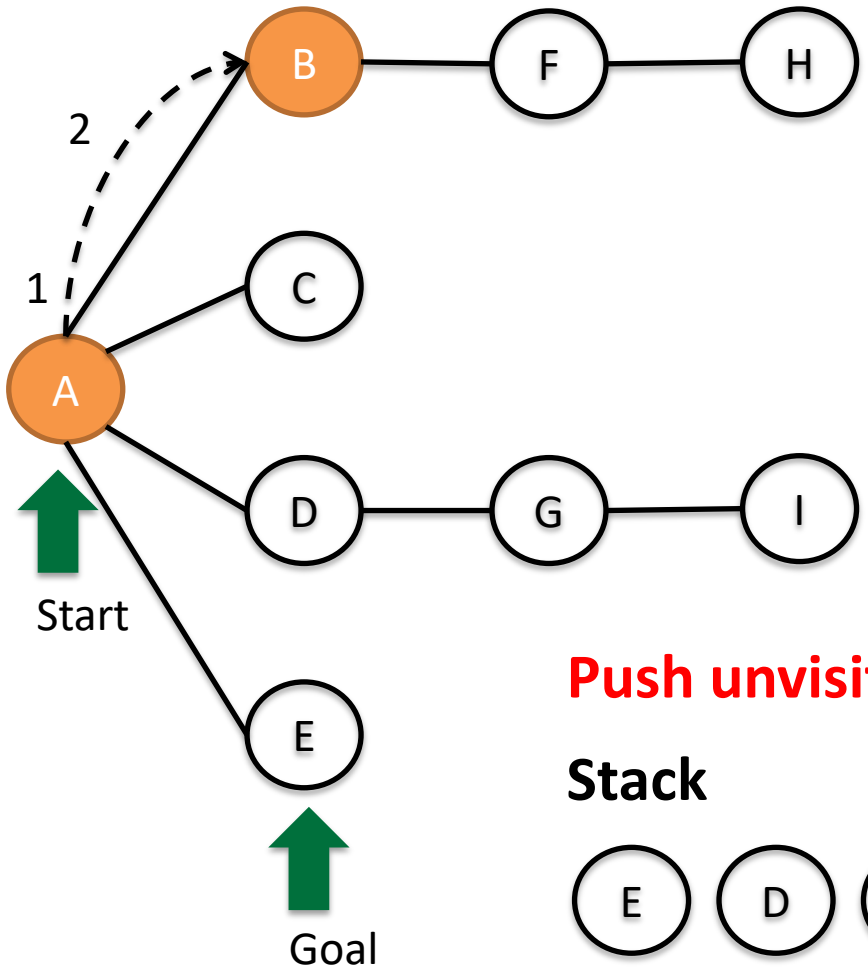
DFS algorithm

```
stack.push(s) //start node
repeat until find goal vertex or
stack empty:
    u = stack.pop()
    if !u.visited
        u.visited = true
        (do something while here)
        for v ∈ u.adjacent
            if !v.visited
              stack.push(v)
```

Pop(B), mark visited

Stack

Depth First Search (DFS) is like exploring a maze



Order nodes visited

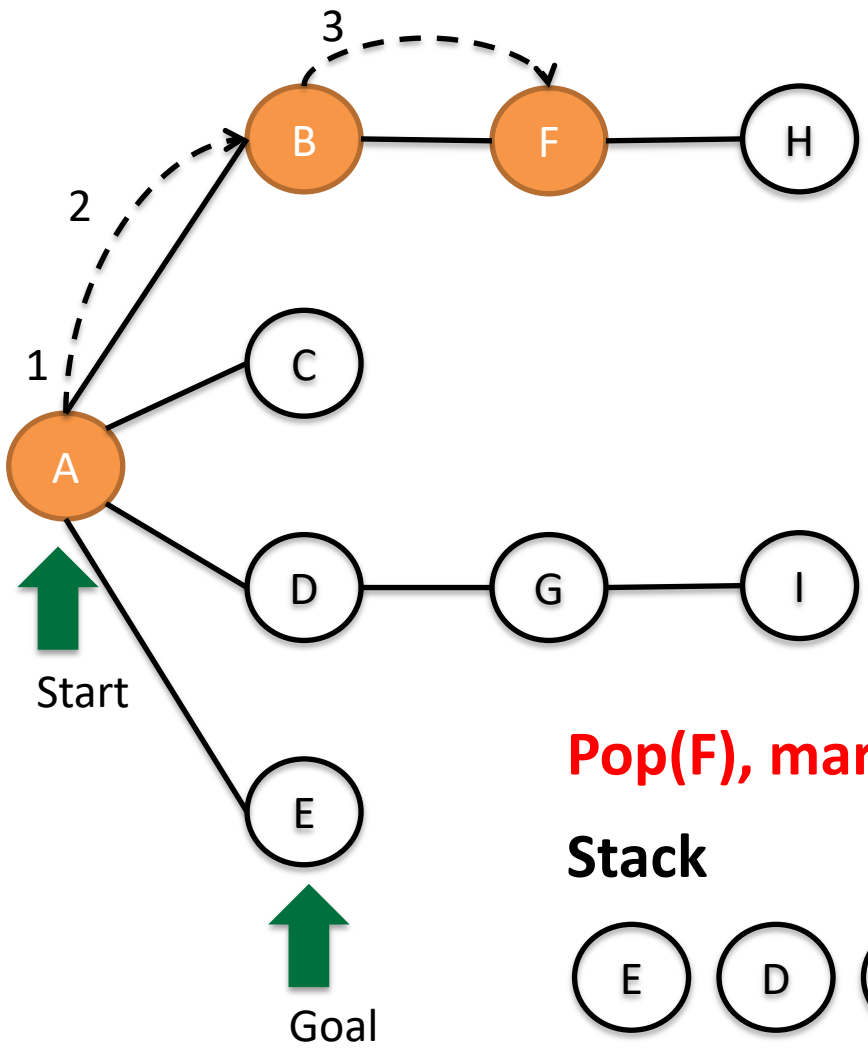
DFS algorithm

```
stack.push(s) //start node
repeat until find goal vertex or
stack empty:
    u = stack.pop()
    if !u.visited
        u.visited = true
        (do something while here)
        for v ∈ u.adjacent
            if !v.visited
              stack.push(v)
```

Push unvisited adjacent (F, but not A)

Stack

Depth First Search (DFS) is like exploring a maze



Order nodes visited

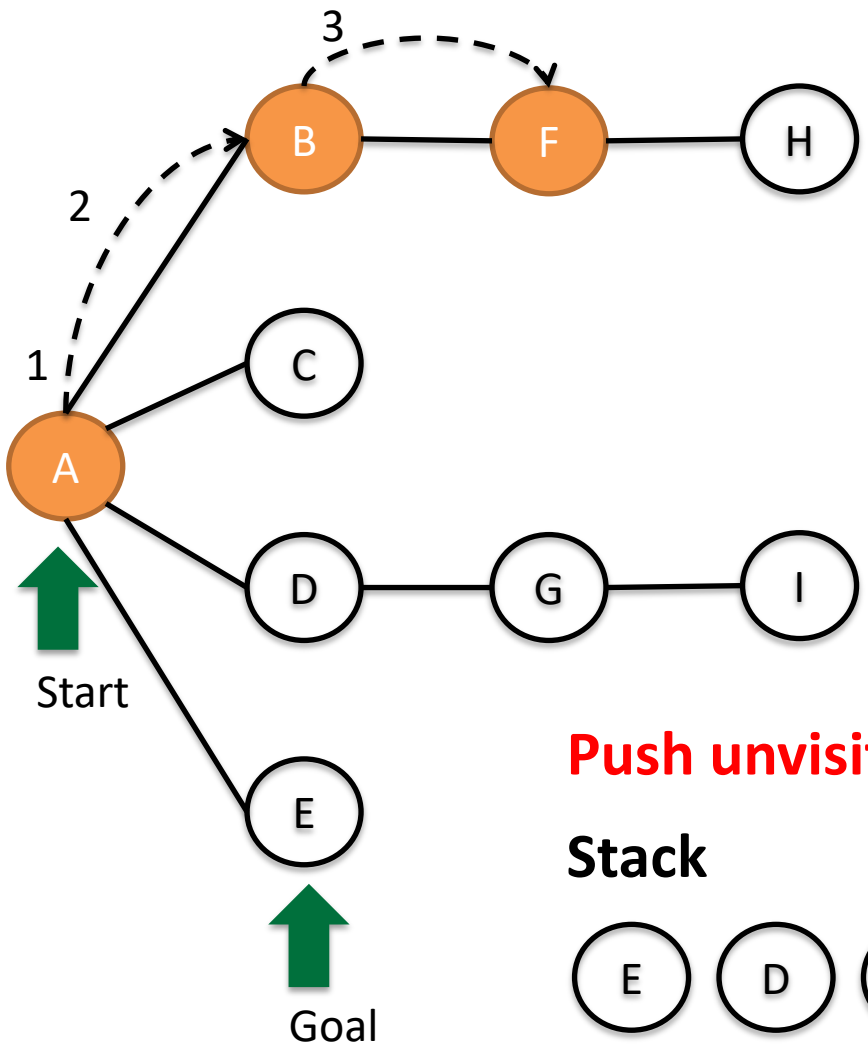
DFS algorithm

```
stack.push(s) //start node
repeat until find goal vertex or
stack empty:
    u = stack.pop()
    if !u.visited
        u.visited = true
        (do something while here)
        for v ∈ u.adjacent
            if !v.visited
              stack.push(v)
```

Pop(F), mark visited

Stack

Depth First Search (DFS) is like exploring a maze



Order nodes visited

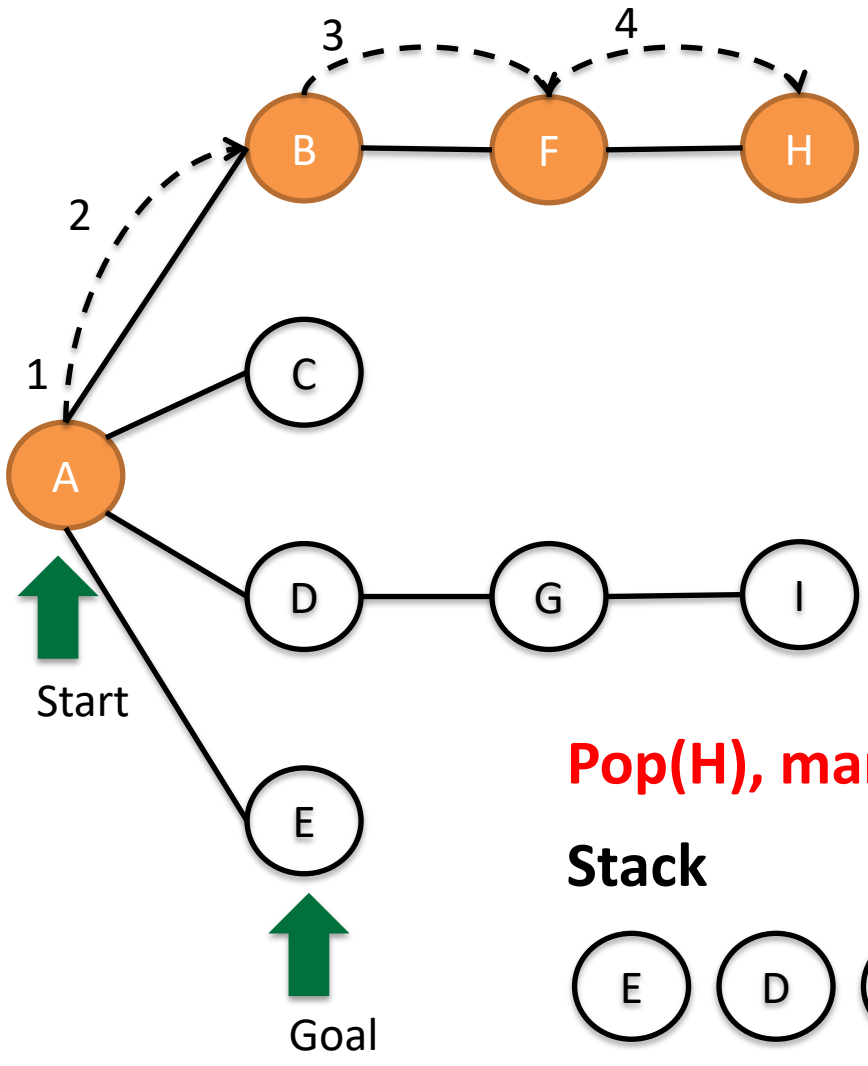
DFS algorithm

```
stack.push(s) //start node
repeat until find goal vertex or
stack empty:
    u = stack.pop()
    if !u.visited
        u.visited = true
        (do something while here)
        for v ∈ u.adjacent
            if !v.visited
              stack.push(v)
```

Push unvisited adjacent (H, but not B)

Stack

Depth First Search (DFS) is like exploring a maze



Order nodes visited

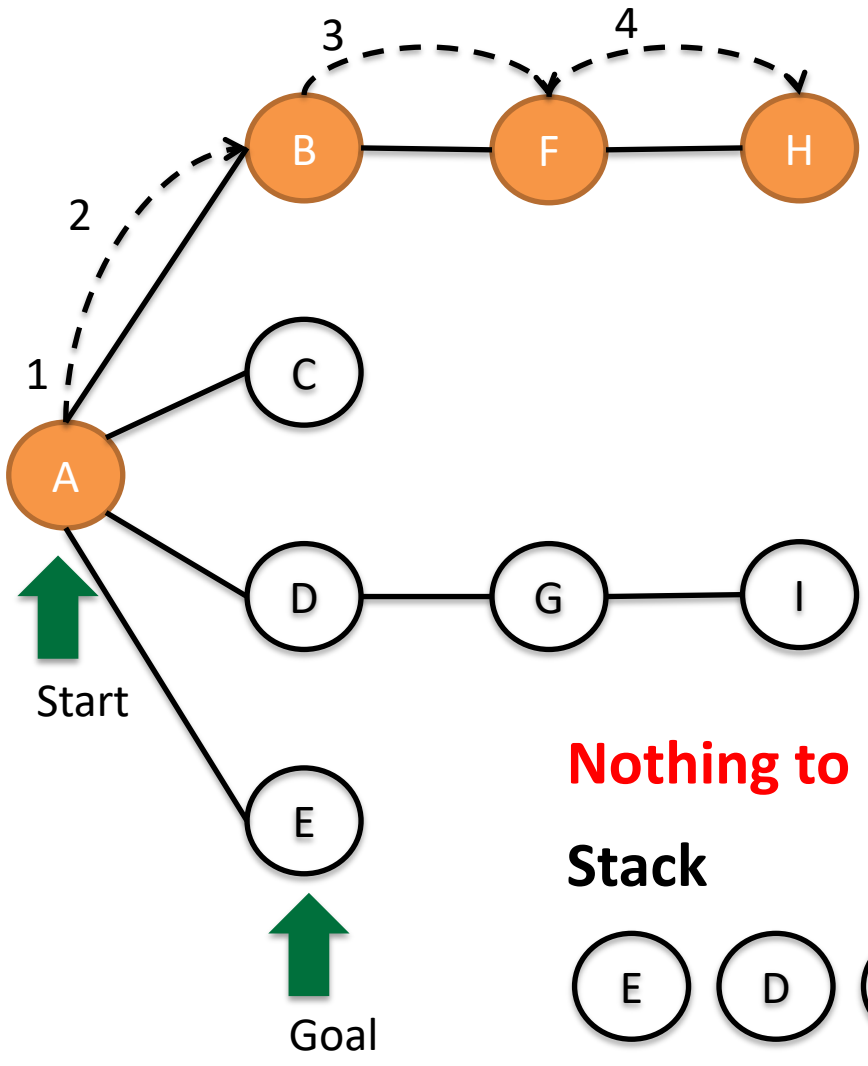
DFS algorithm

```
stack.push(s) //start node
repeat until find goal vertex or
stack empty:
    u = stack.pop()
    if !u.visited
        u.visited = true
        (do something while here)
        for v ∈ u.adjacent
            if !v.visited
              stack.push(v)
```

Pop(H), mark visited

Stack

Depth First Search (DFS) is like exploring a maze



Order nodes visited

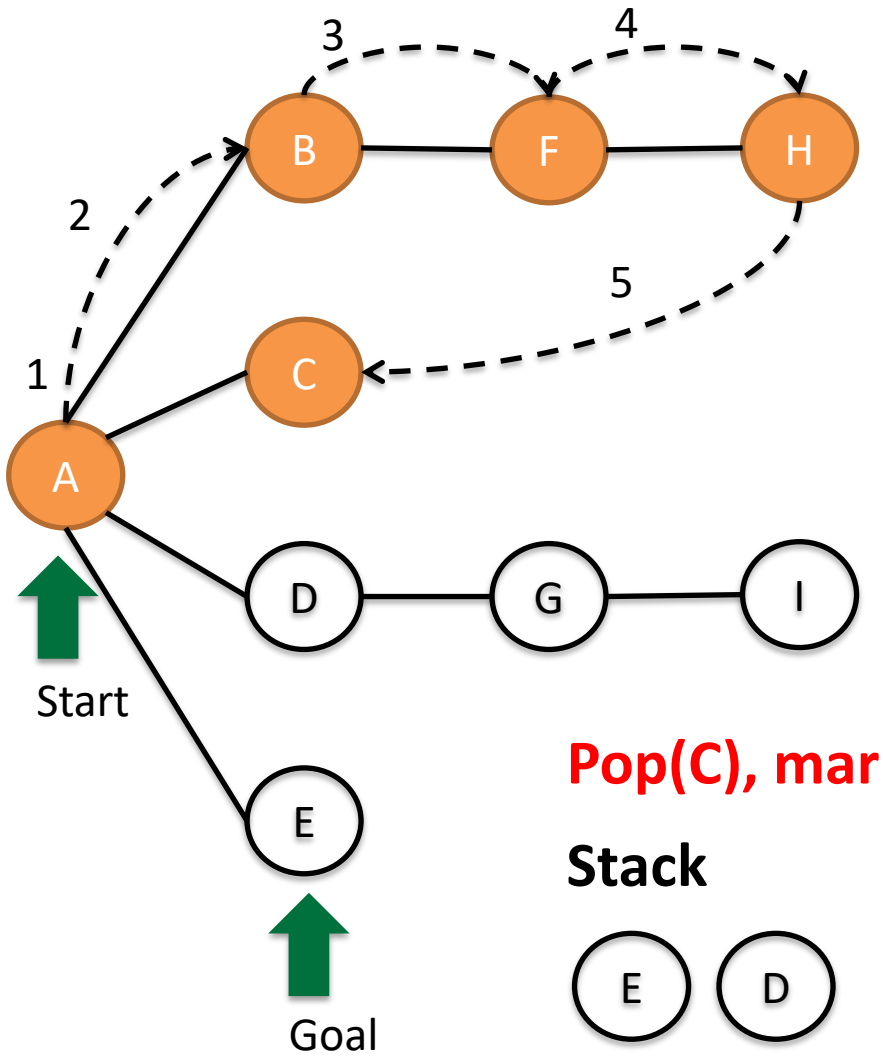
DFS algorithm

```
stack.push(s) //start node
repeat until find goal vertex or
stack empty:
    u = stack.pop()
    if !u.visited
        u.visited = true
        (do something while here)
        for v ∈ u.adjacent
            if !v.visited
              stack.push(v)
```

Nothing to push, back up by popping C

Stack

Depth First Search (DFS) is like exploring a maze



Order nodes visited

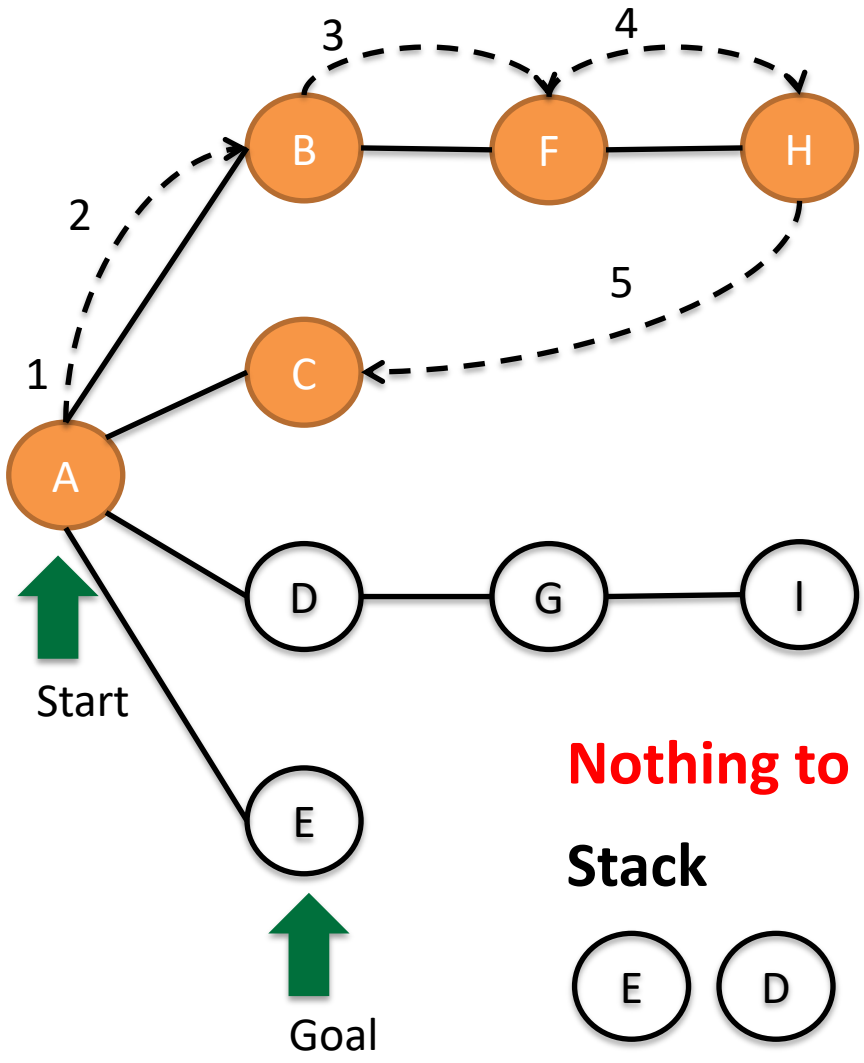
DFS algorithm

```
stack.push(s) //start node
repeat until find goal vertex or
stack empty:
    u = stack.pop()
    if !u.visited
        u.visited = true
        (do something while here)
        for v ∈ u.adjacent
            if !v.visited
              stack.push(v)
```

Pop(C), mark visited

Stack

Depth First Search (DFS) is like exploring a maze



Order nodes visited

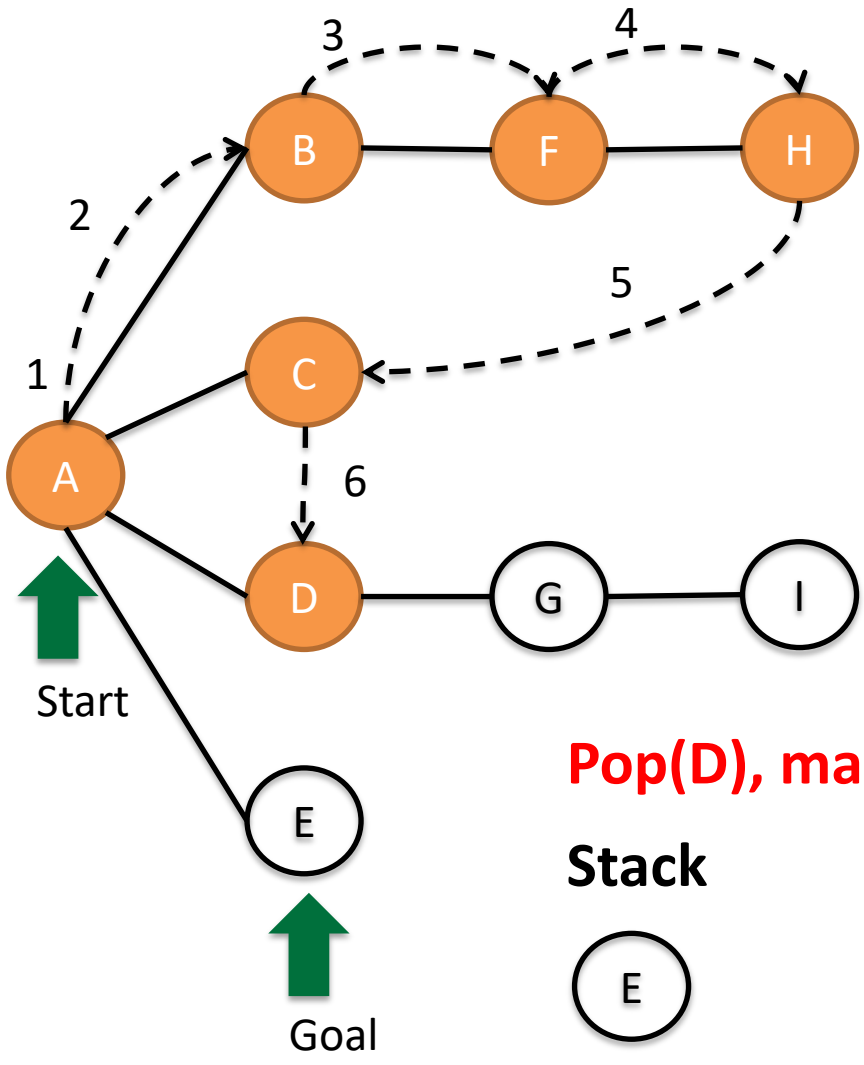
DFS algorithm

```
stack.push(s) //start node
repeat until find goal vertex or
stack empty:
    u = stack.pop()
    if !u.visited
        u.visited = true
        (do something while here)
        for v ∈ u.adjacent
            if !v.visited
              stack.push(v)
```

Nothing to push, back up by popping D

Stack

Depth First Search (DFS) is like exploring a maze



Order nodes visited

DFS algorithm

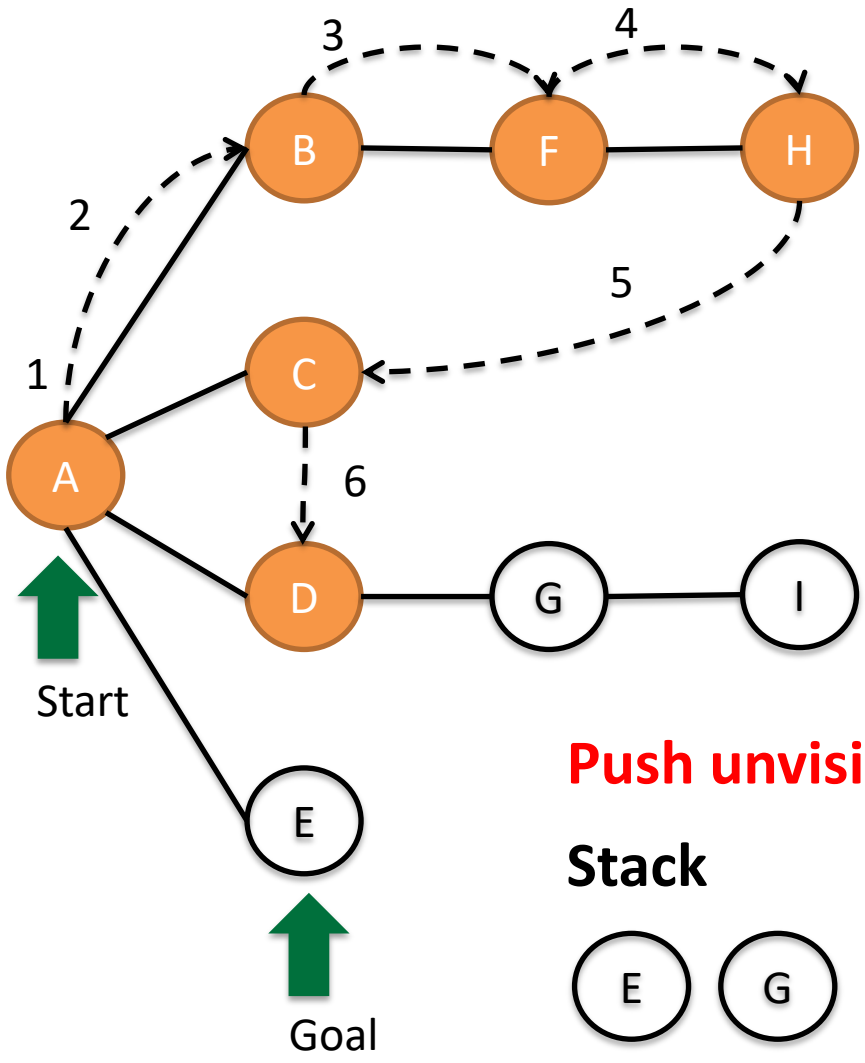
```
stack.push(s) //start node
repeat until find goal vertex or
stack empty:
    u = stack.pop()
    if !u.visited
        u.visited = true
        (do something while here)
        for v ∈ u.adjacent
            if !v.visited
              stack.push(v)
```

Pop(D), mark visited

Stack

E

Depth First Search (DFS) is like exploring a maze



Order nodes visited

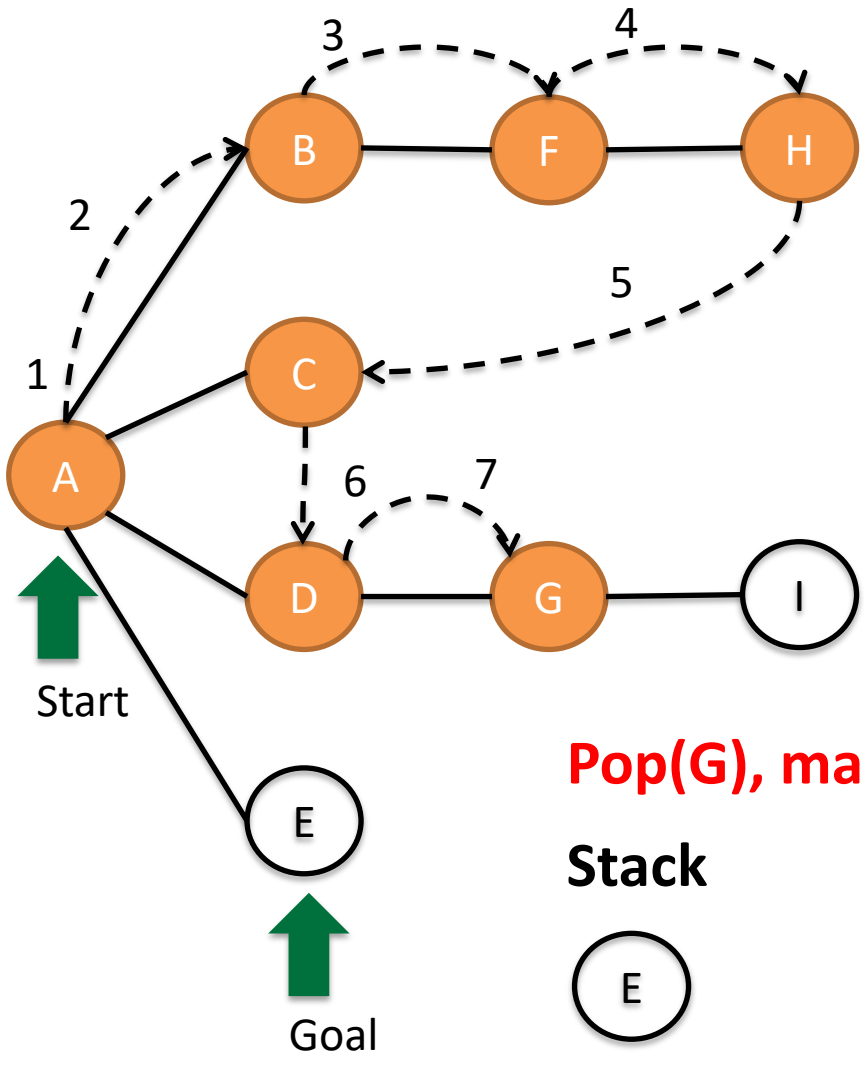
DFS algorithm

```
stack.push(s) //start node
repeat until find goal vertex or
stack empty:
    u = stack.pop()
    if !u.visited
        u.visited = true
        (do something while here)
        for v ∈ u.adjacent
            if !v.visited
              stack.push(v)
```

Push unvisited adjacent (G, but not A)

Stack

Depth First Search (DFS) is like exploring a maze



Order nodes visited

DFS algorithm

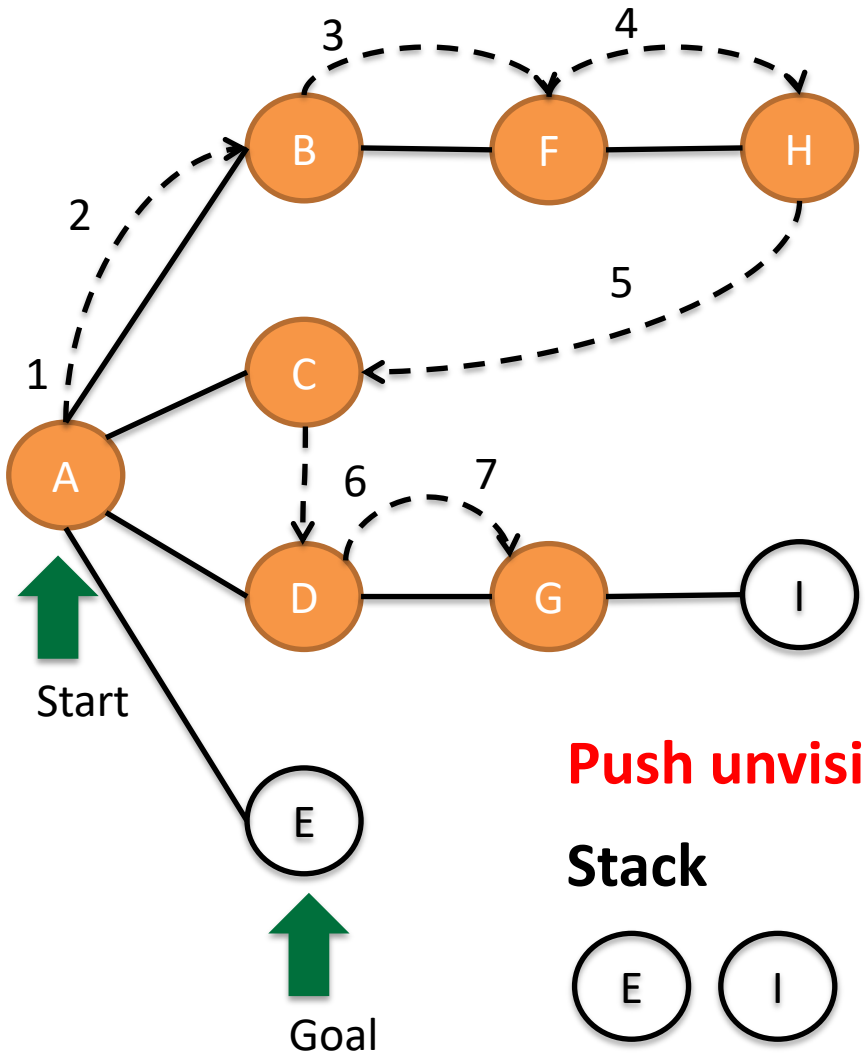
```
stack.push(s) //start node
repeat until find goal vertex or
stack empty:
    u = stack.pop()
    if !u.visited
        u.visited = true
        (do something while here)
        for v ∈ u.adjacent
            if !v.visited
              stack.push(v)
```

Pop(G), mark visited

Stack

E

Depth First Search (DFS) is like exploring a maze



Order nodes visited

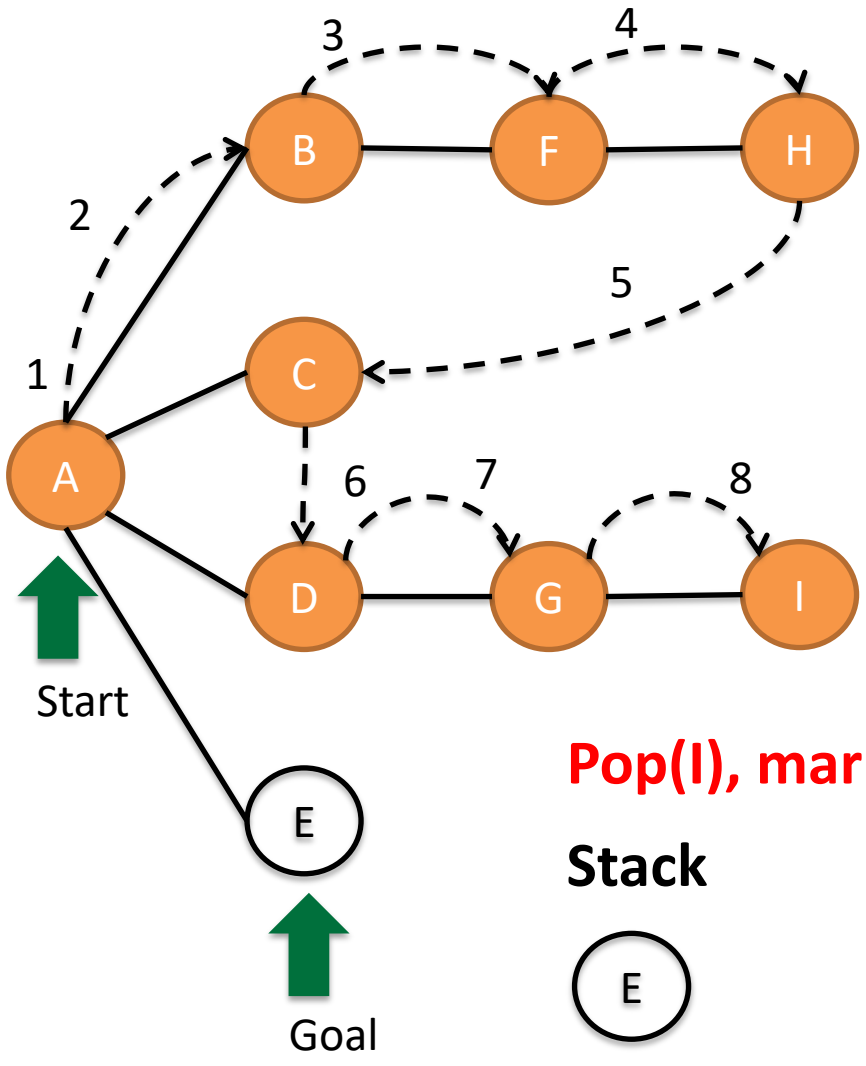
DFS algorithm

```
stack.push(s) //start node
repeat until find goal vertex or
stack empty:
    u = stack.pop()
    if !u.visited
        u.visited = true
        (do something while here)
        for v ∈ u.adjacent
            if !v.visited
              stack.push(v)
```

Push unvisited adjacent (I, but not D)

Stack

Depth First Search (DFS) is like exploring a maze



Order nodes visited

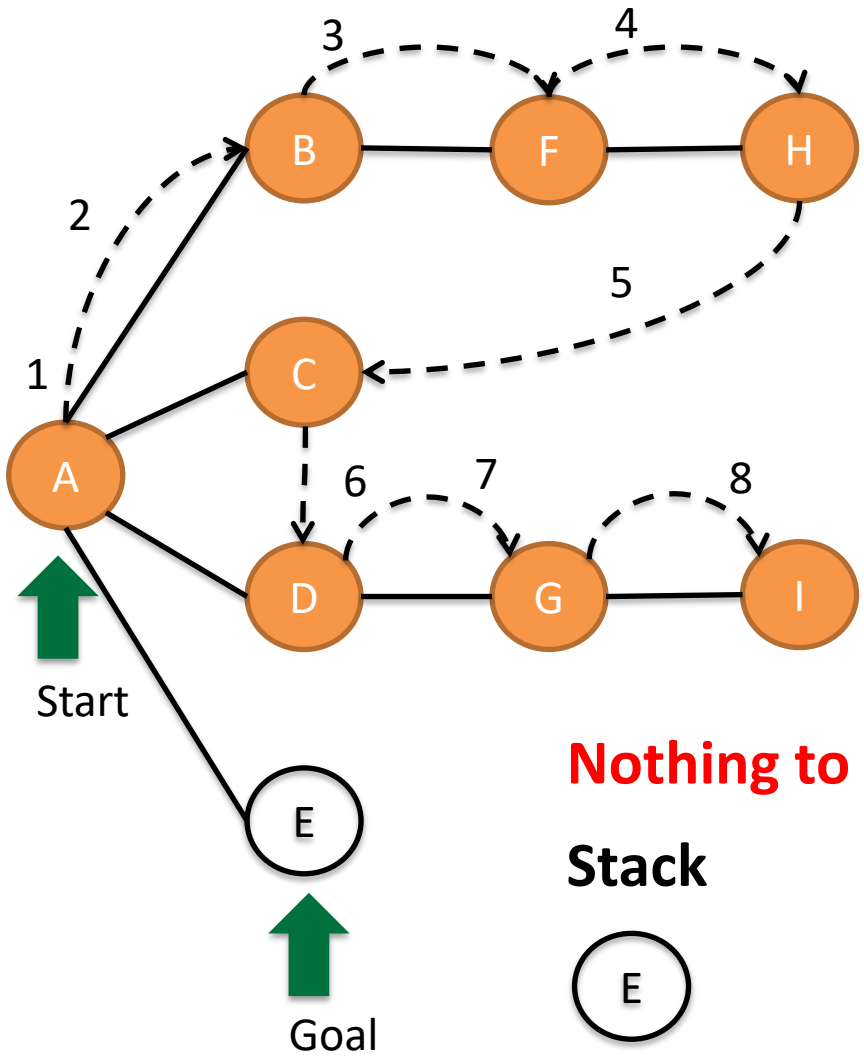
DFS algorithm

```
stack.push(s) //start node
repeat until find goal vertex or
stack empty:
    u = stack.pop()
    if !u.visited
        u.visited = true
        (do something while here)
        for v ∈ u.adjacent
            if !v.visited
              stack.push(v)
```

Pop(I), mark visited

Stack

Depth First Search (DFS) is like exploring a maze



Order nodes visited

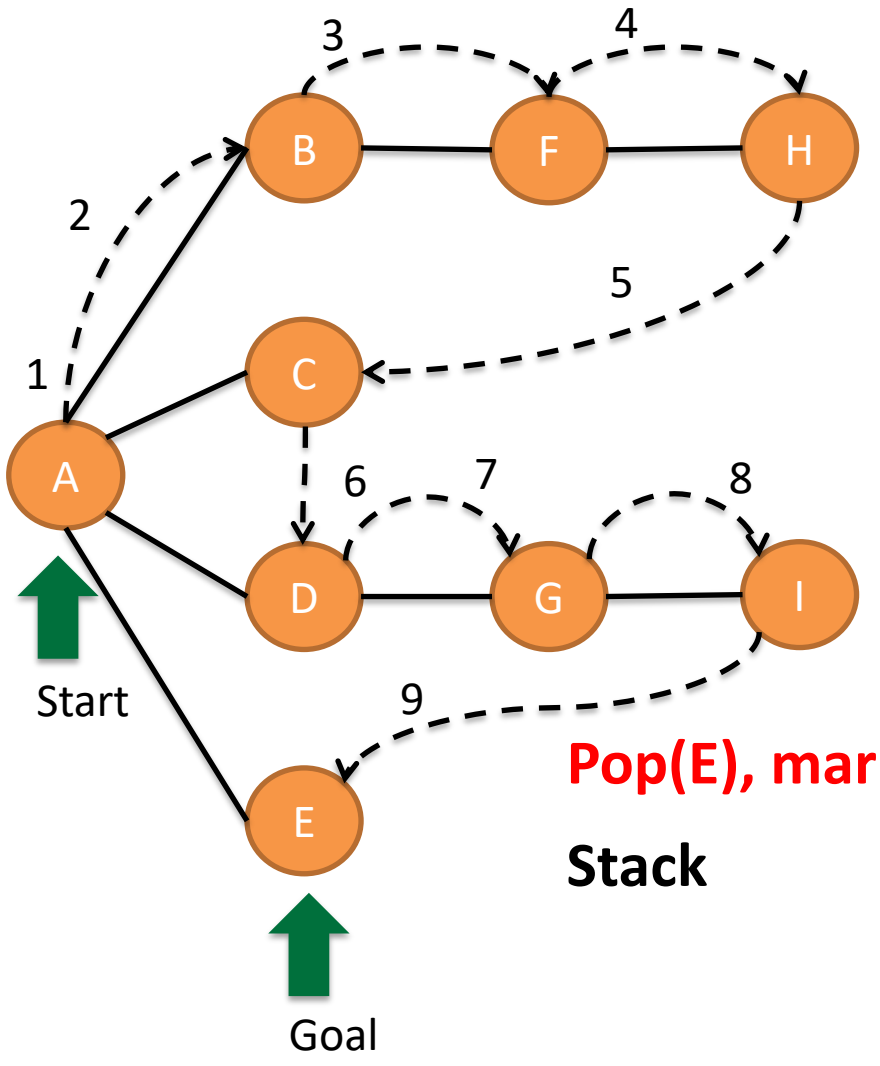
DFS algorithm

```
stack.push(s) //start node
repeat until find goal vertex or
stack empty:
    u = stack.pop()
    if !u.visited
        u.visited = true
        (do something while here)
        for v ∈ u.adjacent
            if !v.visited
              stack.push(v)
```

Nothing to push, back up by popping E

Stack

Depth First Search (DFS) is like exploring a maze



Order nodes visited

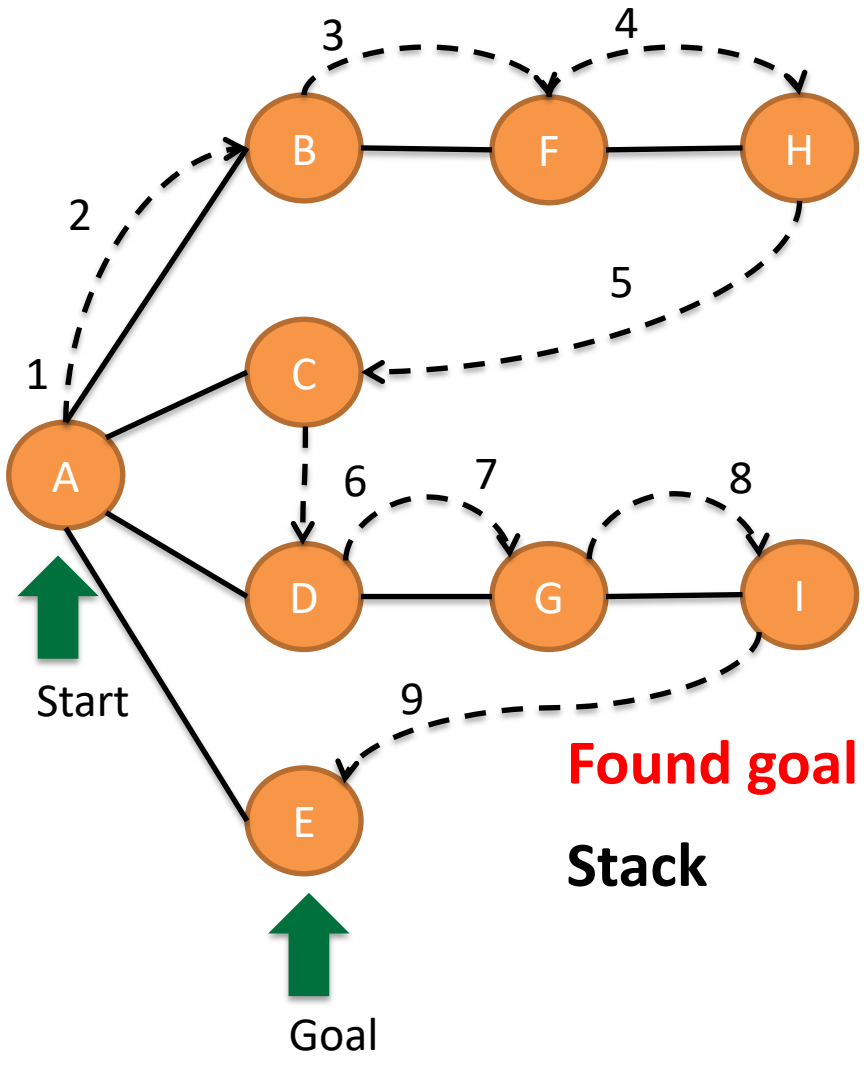
DFS algorithm

```
stack.push(s) //start node
repeat until find goal vertex or
stack empty:
    u = stack.pop()
    if !u.visited
        u.visited = true
        (do something while here)
        for v ∈ u.adjacent
            if !v.visited
              stack.push(v)
```

Pop(E), mark visited

Stack

Depth First Search (DFS) is like exploring a maze



Order nodes visited

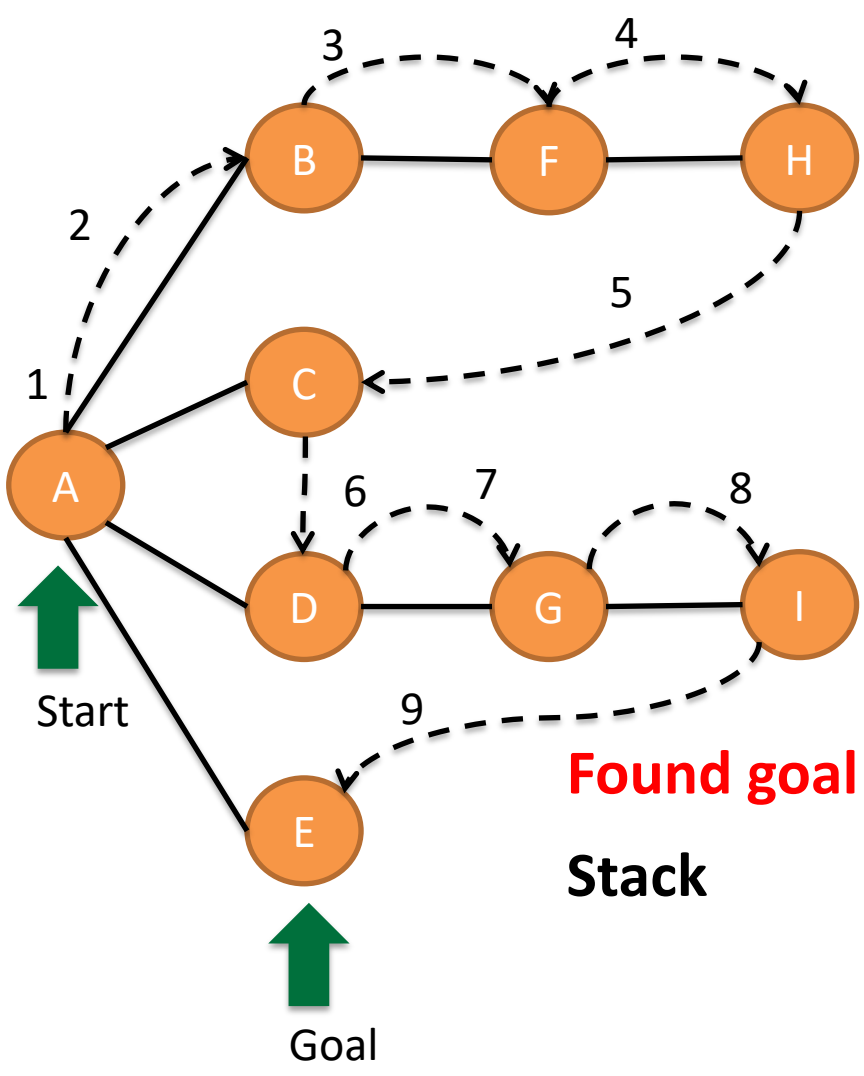
Found goal

Stack

DFS algorithm

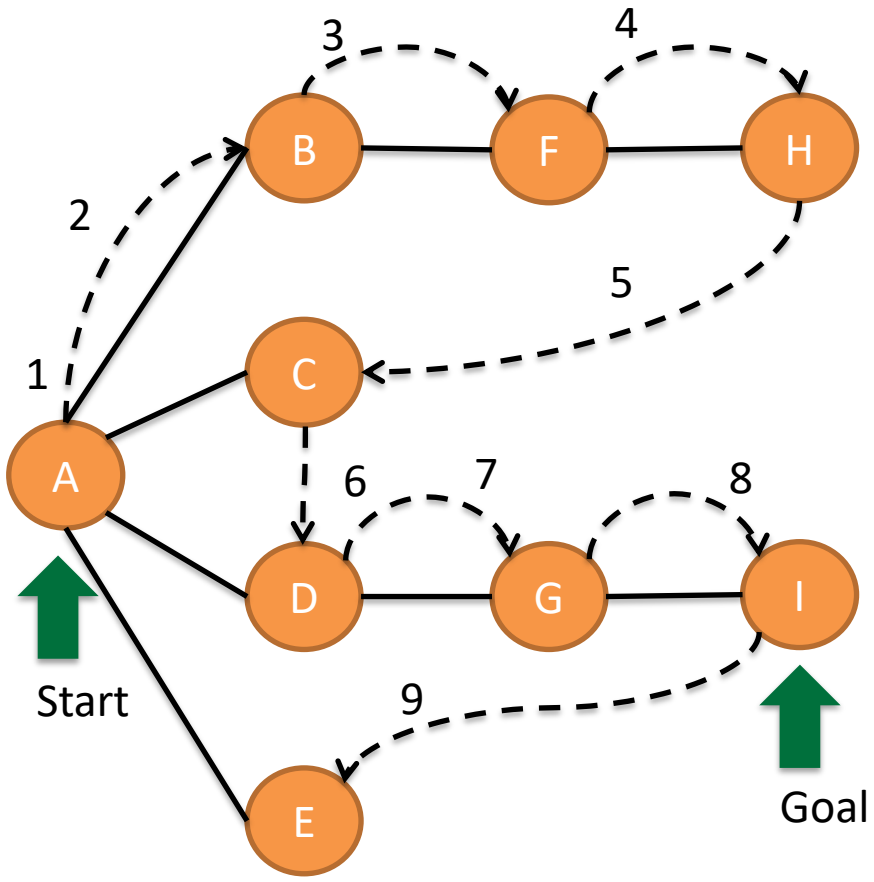
```
stack.push(s) //start node
repeat until find goal vertex or
stack empty:
    u = stack.pop()
    if !u.visited
        u.visited = true
        (do something while here)
        for v ∈ u.adjacent
              if !v.visited
                stack.push(v)
```

Node discovery tells us something about the graph



Order nodes visited

To find path from *start* to *v*, keep track of previous node as nodes are "discovered"



Order nodes visited

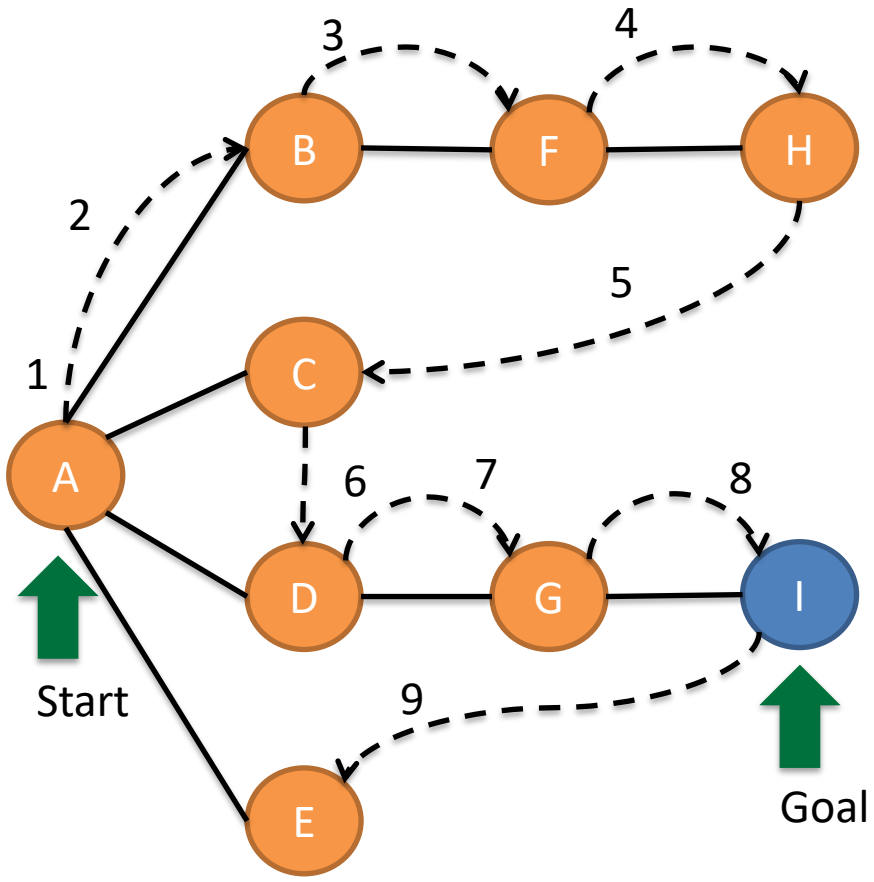
Path from start to *v*

- Each time node discovered, keep pointer to previous
- Could keep Map with node as Key and previous as Value
- Start at goal node (*v*)
- Track backward until find starting node
- Will find a path if it exists, but not necessarily the shortest path (wait for BFS)

Path A to I

Key	Value
A	Null
B	A
C	A
D	A
E	A
F	B
G	D
H	F
I	G

To find path from *start* to *v*, keep track of previous node as nodes are "discovered"



Order nodes visited

Path from *start* to *v*

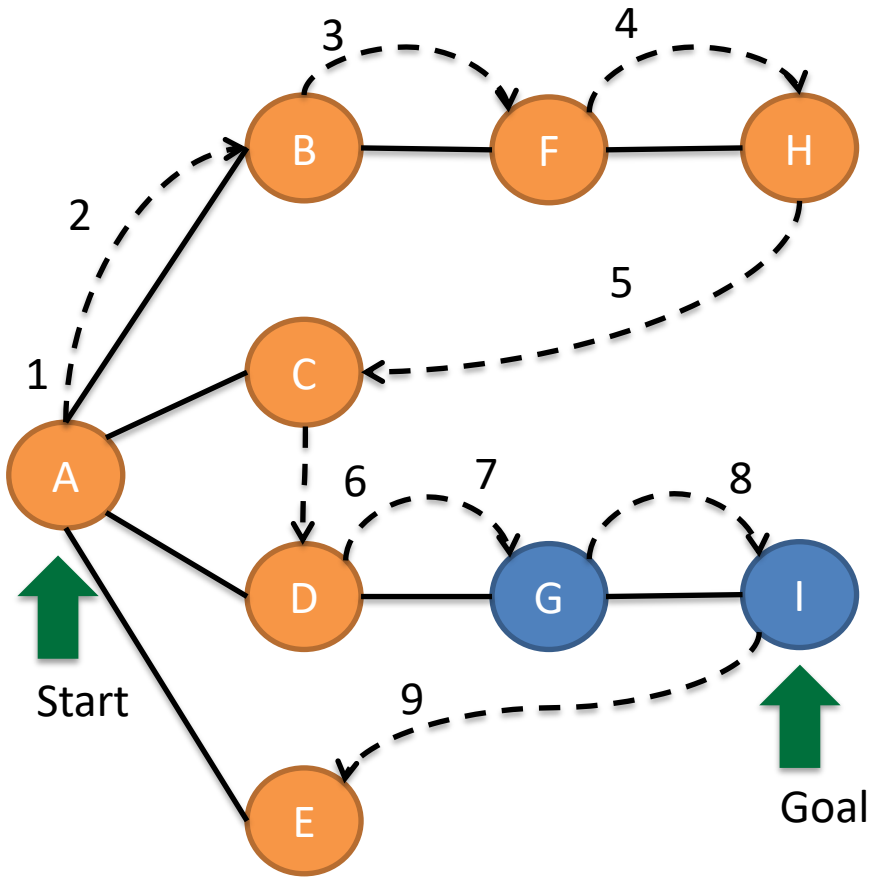
- Each time node discovered, keep pointer to previous
- Could keep Map with node as Key and previous as Value
- Start at goal node (*v*)
- Track backward until find starting node
- Will find a path if it exists, but not necessarily the shortest path (wait for BFS)

Path A to I

Key	Value
A	Null
B	A
C	A
D	A
E	A
F	B
G	D
H	F
I	**G**

Path G,I

To find path from *start* to *v*, keep track of previous node as nodes are "discovered"



Order nodes visited

Path from *start* to *v*

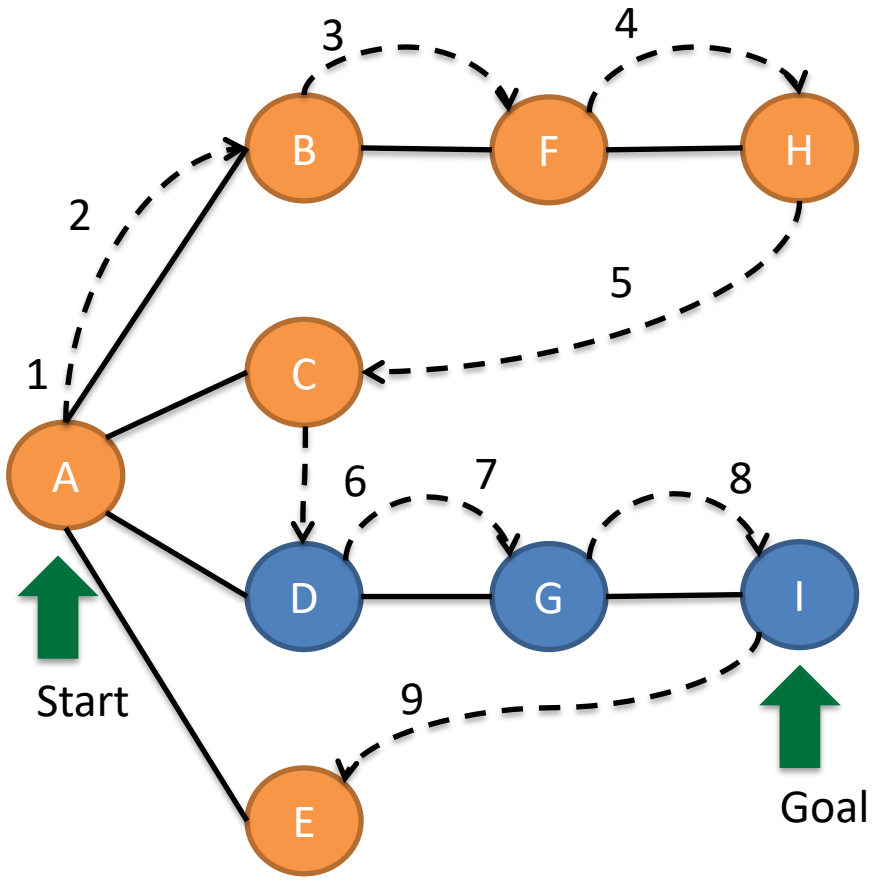
- Each time node discovered, keep pointer to previous
- Could keep Map with node as Key and previous as Value
- Start at goal node (*v*)
- Track backward until find starting node
- Will find a path if it exists, but not necessarily the shortest path (wait for BFS)

Path A to I

Key	Value
A	Null
B	A
C	A
D	A
E	A
F	B
G	**D**
H	F
I	G

Path D,G,I

To find path from *start* to *v*, keep track of previous node as nodes are "discovered"



Order nodes visited

Path from *start* to *v*

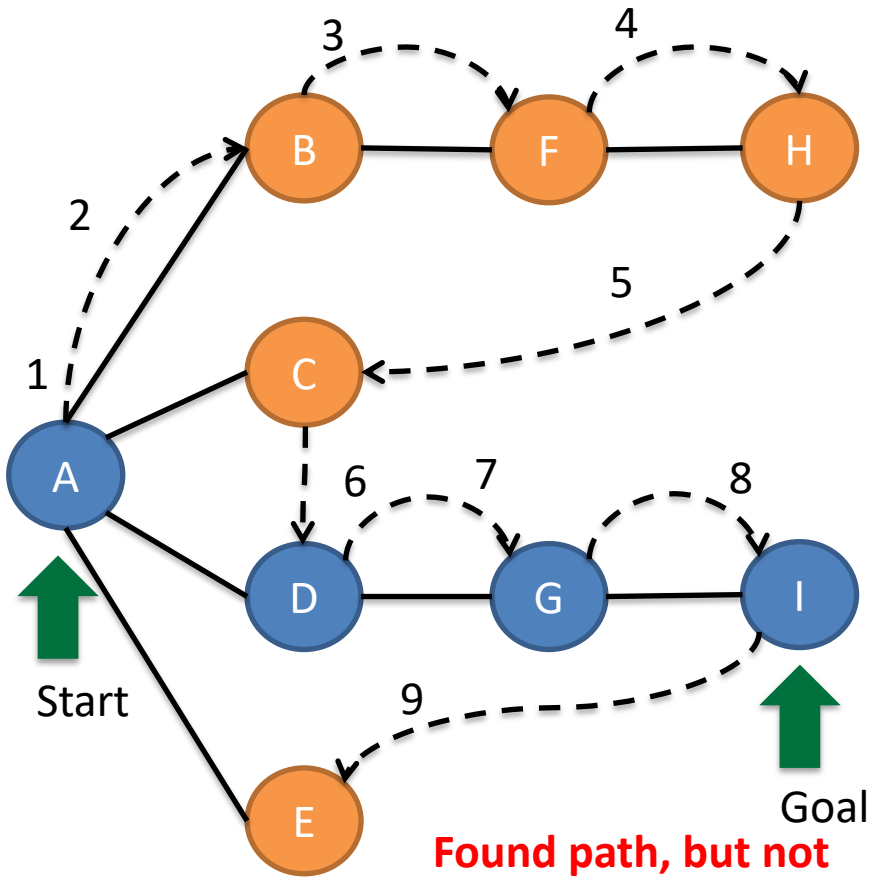
- Each time node discovered, keep pointer to previous
- Could keep Map with node as Key and previous as Value
- Start at goal node (*v*)
- Track backward until find starting node
- Will find a path if it exists, but not necessarily the shortest path (wait for BFS)

Path A to I

Key	Value
A	Null
B	A
C	A
D	**A**
E	A
F	B
G	D
H	F
I	G

Path A,D,G,I

To find path from *start* to *v*, keep track of previous node as nodes are "discovered"



Path from *start* to *v*

- Each time node discovered, keep pointer to previous
- Could keep Map with node as Key and previous as Value
- Start at goal node (*v*)
- Track backward until find starting node
- Will find a path if it exists, but not necessarily the shortest path (wait for BFS)

Path A to I

Key	Value
A	Null
B	A
C	A
D	**A**
E	A
F	B
G	D
H	F
I	G

Path A,D,G,I

Found path, but not necessarily shortest path

Order nodes visited

After DFS from *start* can find *a* path from *start* to any other reachable node

Could we start from node other than A? No

GraphTraversal.java: DFS code

```
17 public class GraphTraversal<V,E> {
18
19     public Map<V,V> backTrack; //keep track of prior vertex when v
20
21     /**
22      * Constructor.  Initialize backTrack to new HashMap.
23      */
24     public GraphTraversal() {
25         backTrack = new HashMap<V,V>();
26     }
27
28     /**
29      * Depth First Search
30      * @param G -- graph to search
31      * @param start -- starting vertex
32      */
33     public void DFS(AdjacencyMapGraph<V,E> G, V start) {
34         System.out.println("\nDepth First Search from " + start);
35         backTrack = new HashMap<V,V>(); //initialize backTrack
36         backTrack.put(start, null); //load start node with null pa
37         Set<V> visited = new HashSet<V>(); //Set to track which ve
38         Stack<V> stack = new Stack<V>(); //stack to implement DFS
39
40         stack.push(start); //push start vertex
41         while (!stack.isEmpty()) { //loop until no more vertices
42             V u = stack.pop(); //get most recent vertex
43             if (!visited.contains(u)) { //if not already visited
44                 visited.add(u); //add to visited Set
45                 for (V v : G.outNeighbors(u)) { //loop over out ne
46                     if (!visited.contains(v)) { //if neighbor not
47                         stack.push(v); //push non-visted neighbor
48                         backTrack.put(v, u); //save that this vert
49                     }
50                 }
51             }
52         }
53     }
```

DFS run time

Run time

- Assume graph with *n* nodes and *m* edges
- Visit each node at most one time (due to visited indicator)
- Examine each edge at most one time
- Run-time complexity is O(n+m)

DFS algorithm

```
stack.push(s) //start node
repeat until find goal vertex or
stack empty:
    u = stack.pop()
    if !u.visited
        u.visited = true
        (do something while here)
        for v ∈ u.adjacent
              if !v.visited
                stack.push(v)
```

After DFS (or BFS) *findPath()* finds a path from start to end if it exists

GraphTraversal.java

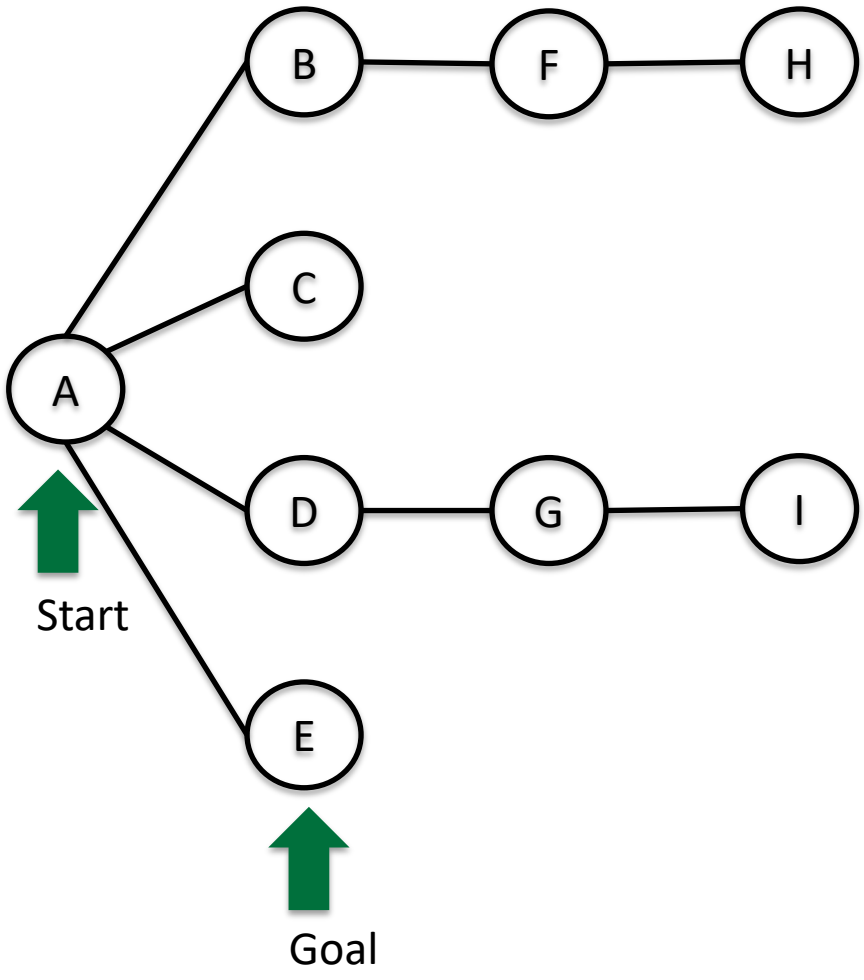
```
88      public ArrayList<V> findPath(V start, V end) {
89          //check that DFS or BFS have already been run from start
90          if (backTrack.isEmpty() || !backTrack.containsKey(start) ||
91                  (backTrack.containsKey(start) && backTrack.get(start) != null)) {
92              System.out.println("Run DFS or BFS on " + start + " before trying to find a path");
93              return new ArrayList<V>();
94          }
95          System.out.println("Path from " + start + " to " + end);
96          //make sure end vertex in backTrack
97          if (!backTrack.containsKey(end)) {
98              System.out.println("\tNo path found");
99              return new ArrayList<V>();
100         }
101         //start from end vertex and work backward to start vertex
102         ArrayList<V> path = new ArrayList<V>(); //this will hold the path from start to end verte
103         V current = end; //start at end vertex
104         //loop from end vertex back to start vertex
105         while (current != null) {
106             path.add(0,current); //add this vertex to front of arraylist path
107             current = backTrack.get(current); //get vertex that discovered this vertex
108         }
109         System.out.println(path);
110         return path;
111     }
112
```

- **Run time complexity?**
- **Length of path from *start* to *end***

Agenda

1. Depth first search (DFS)
2. Breadth first search (BFS)
3. Examples from last class and today

Breadth First Search (BFS) can find the shortest path between nodes



BFS basic idea

- Explore outward in "ripples"
- Look at all nodes 1 step away, then all nodes 2 steps away…
- Relies on a **Queue** (implicit or explicit) implementation
- Path from `s` to any other vertex is shortest

Some of you did Breadth First Search on Problem Set 1

RegionFinder

```
Loop over all the pixels
    If a pixel is unvisited and of the correct color
        Start a new region
        Keep track of pixels need to be visited, initially just one
        As long as there's some pixel that needs to be visited
            Get one to visit
            Add it to the region
            Mark it as visited
            Loop over all its neighbors
                If the neighbor is of the correct color
                    Add it to the list of pixels to be visited
        If the region is big enough to be worth keeping, do so
```

Some of you did Breadth First Search on Problem Set 1

RegionFinder

```
Loop over all the pixels
    If a pixel is unvisited and of the correct color
        Start a new region
        Keep track of pixels need to be visited, initially just one
        As long as there's some pixel that needs to be visited
            Get one to visit
            Add it to the region
            Mark it as visited
            Loop over all its neighbors
                If the neighbor is of the correct color
                    Add it to the list of pixels to be visited
        If the region is big enough to be worth keeping, do so
```

If you added to end of list…

Some of you did Breadth First Search on Problem Set 1

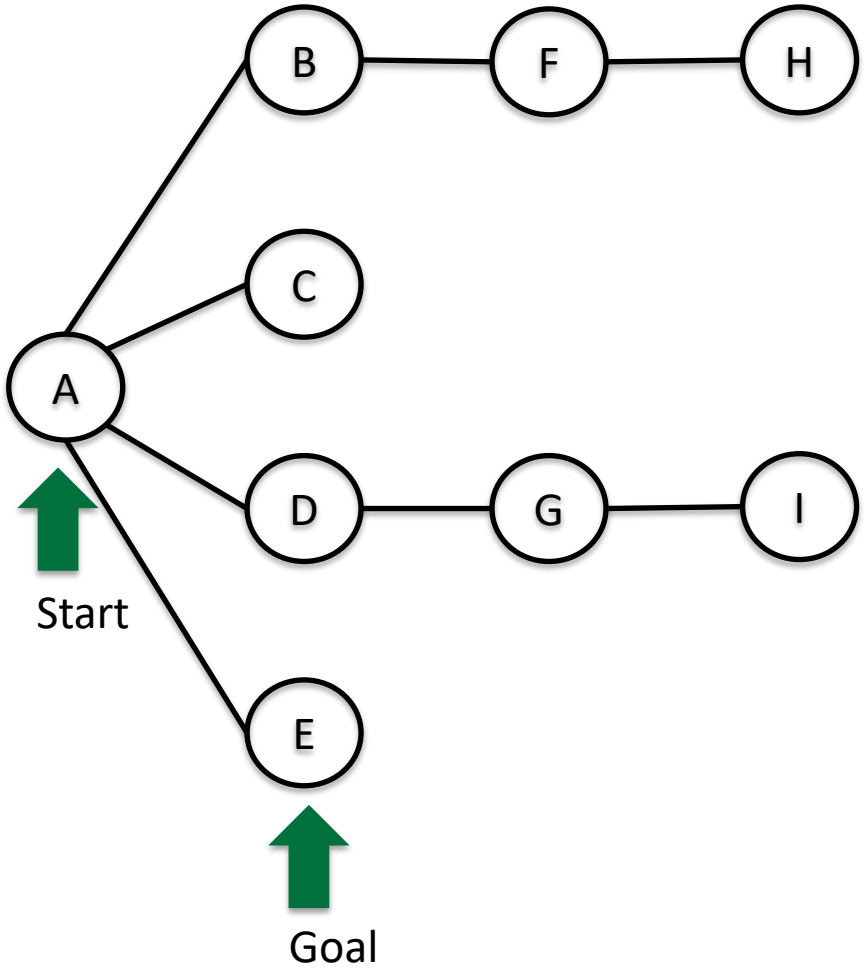
RegionFinder

```
Loop over all the pixels
    If a pixel is unvisited and of the correct color
        Start a new region
        Keep track of pixels need to be visited, initially just one
        As long as there's some pixel that needs to be visited
            Get one to visit
            Add it to the region
            Mark it as visited
            Loop over all its neighbors
                If the neighbor is of the correct color
                    Add it to the list of pixels to be visited
        If the region is big enough to be worth keeping, do so
```

And if you get a pixel from front of list, you implemented a Queue

If you added to end of list…

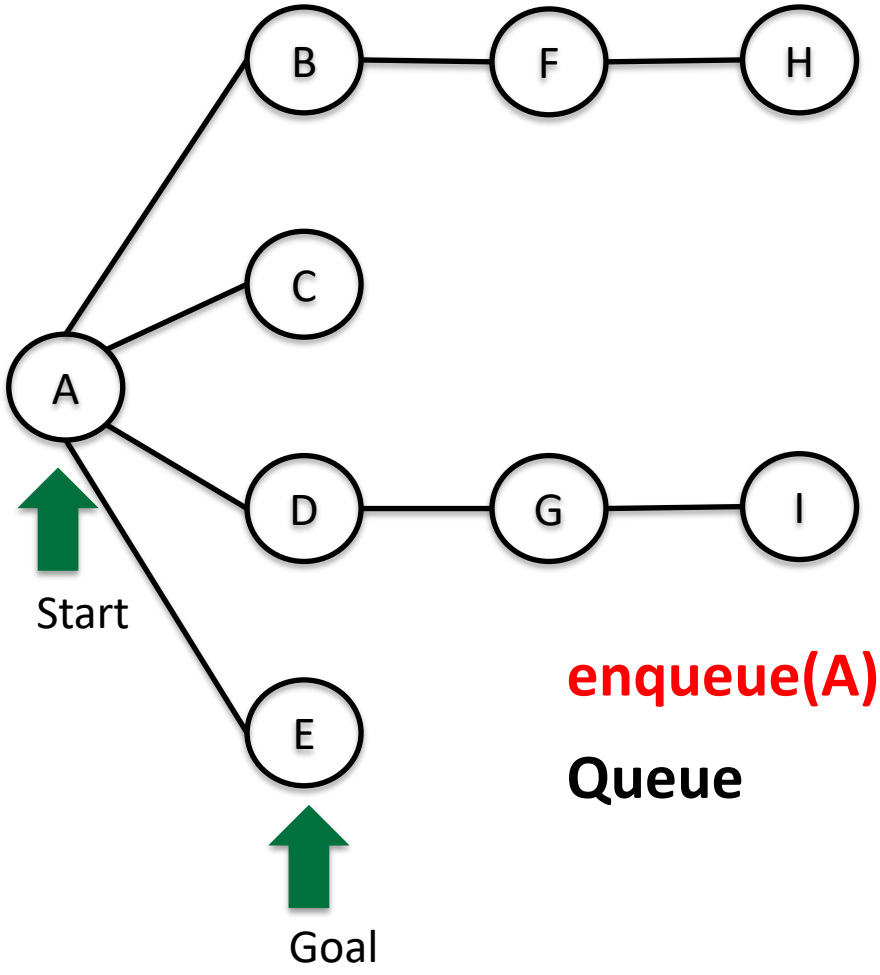
Breadth First Search (BFS) can find the shortest path between nodes



BFS algorithm

```
enqueue(s) //start node
s.visited = true
repeat until find goal vertex or
queue empty:
    u = dequeque()
    (do something here)
    for v ∈ u.adjacent
        if !v.visited
            v.visited = true
            enqueue(v)
```

Breadth First Search (BFS) can find the shortest path between nodes



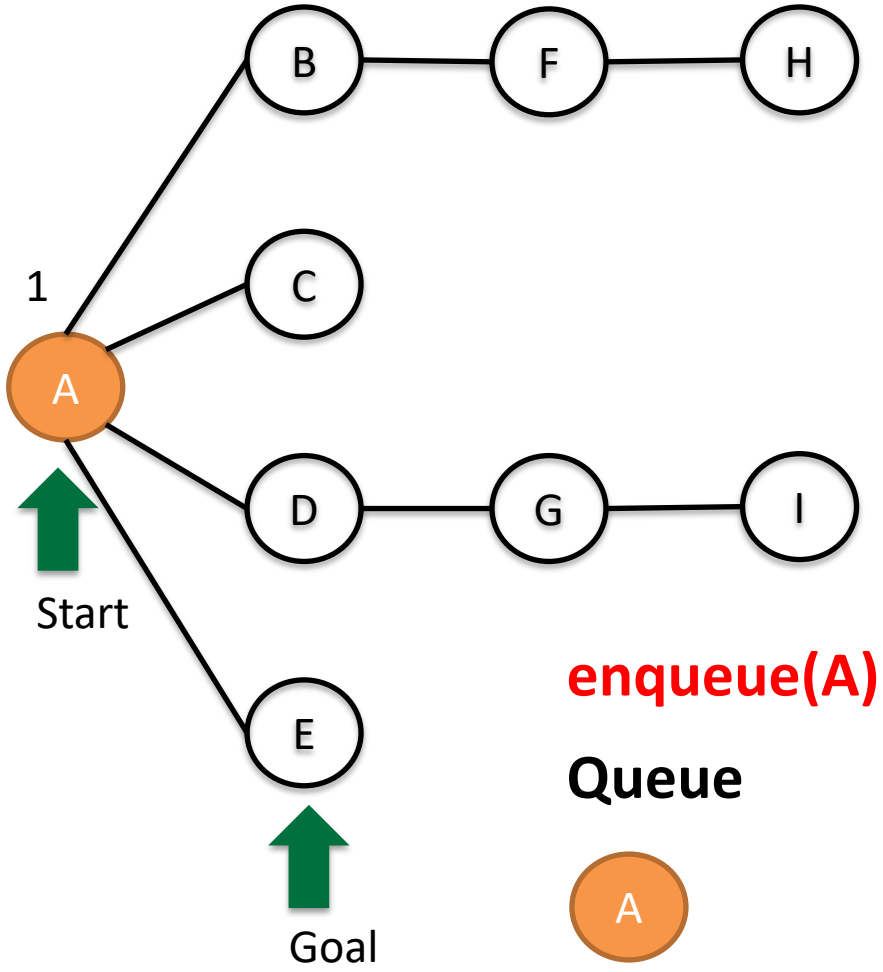
enqueue(A)

Queue

BFS algorithm

```
enqueue(s) //start node
s.visited = true
repeat until find goal vertex or
queue empty:
    u = dequeque()
    (do something here)
    for v ∈ u.adjacent
        if !v.visited
            v.visited = true
            enqueue(v)
```

Breadth First Search (BFS) can find the shortest path between nodes



enqueue(A)

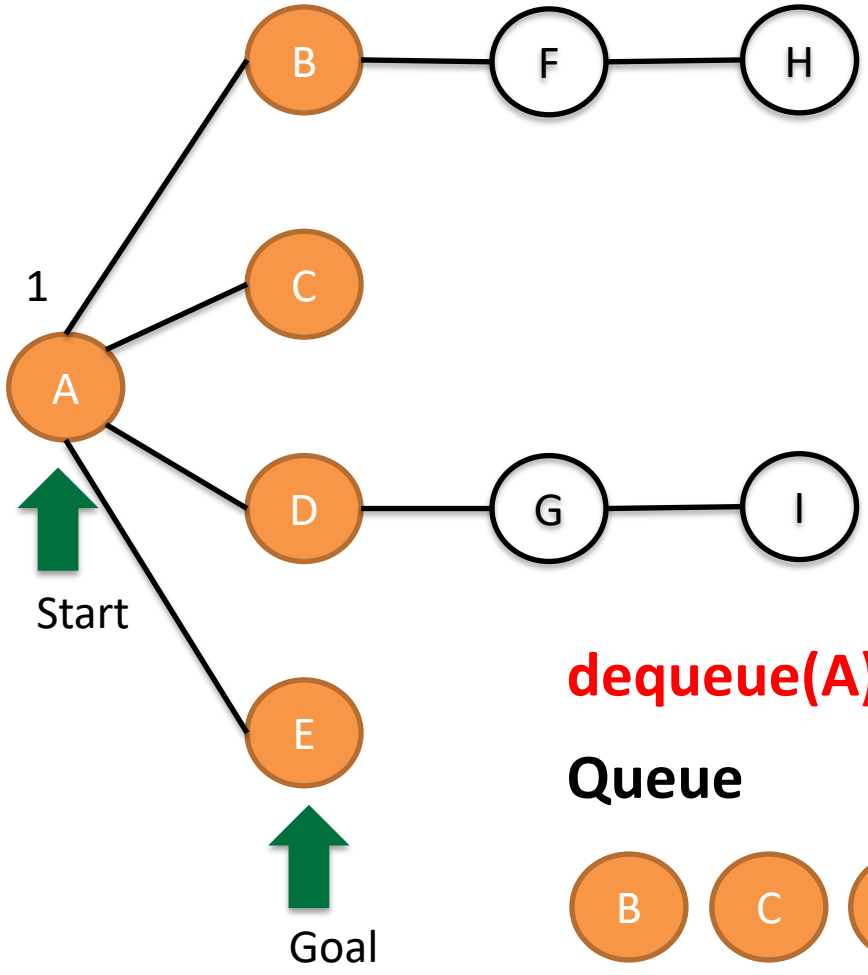
Queue

A

BFS algorithm

```
enqueue(s) //start node
s.visited = true
repeat until find goal vertex or
queue empty:
    u = dequeque()
    (do something here)
    for v ∈ u.adjacent
        if !v.visited
            v.visited = true
            enqueue(v)
```

Breadth First Search (BFS) can find the shortest path between nodes



"Visit" node *u*

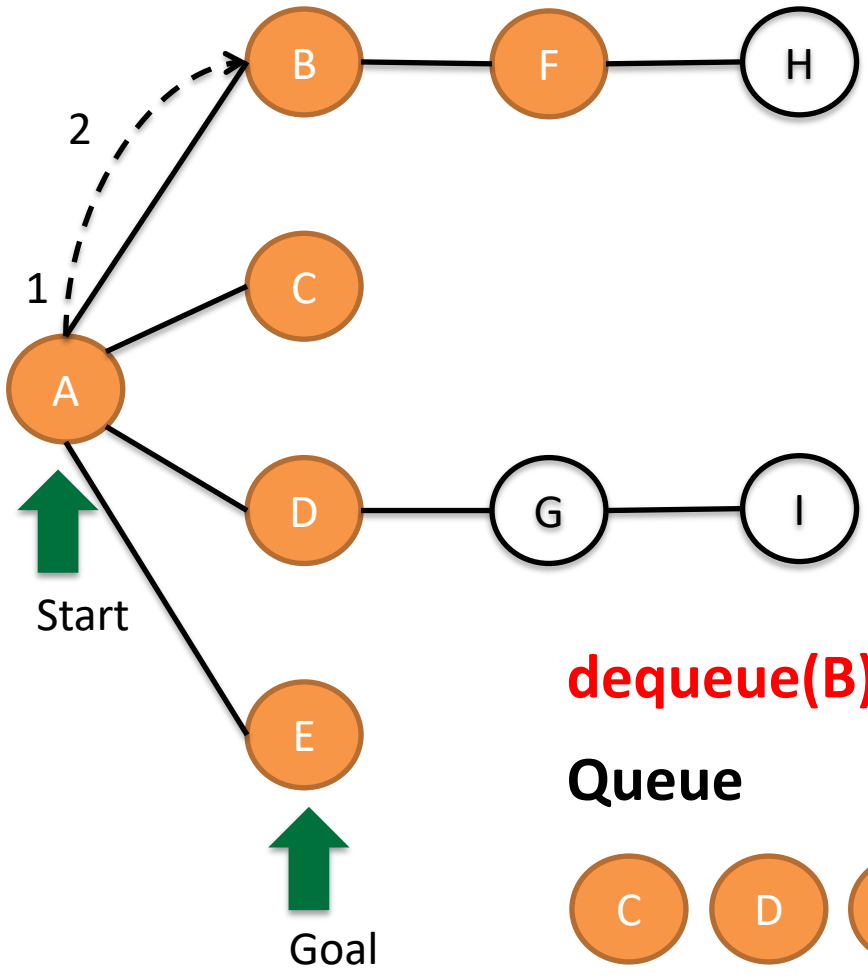
BFS algorithm

```
enqueue(s) //start node
s.visited = true
repeat until find goal vertex or
queue empty:
    u = dequeque()
    (do something here)
    for v ∈ u.adjacent
        if !v.visited
            v.visited = true
            enqueue(v)
```

dequeue(A), enqueue unvisited adjacent

Queue

Breadth First Search (BFS) can find the shortest path between nodes



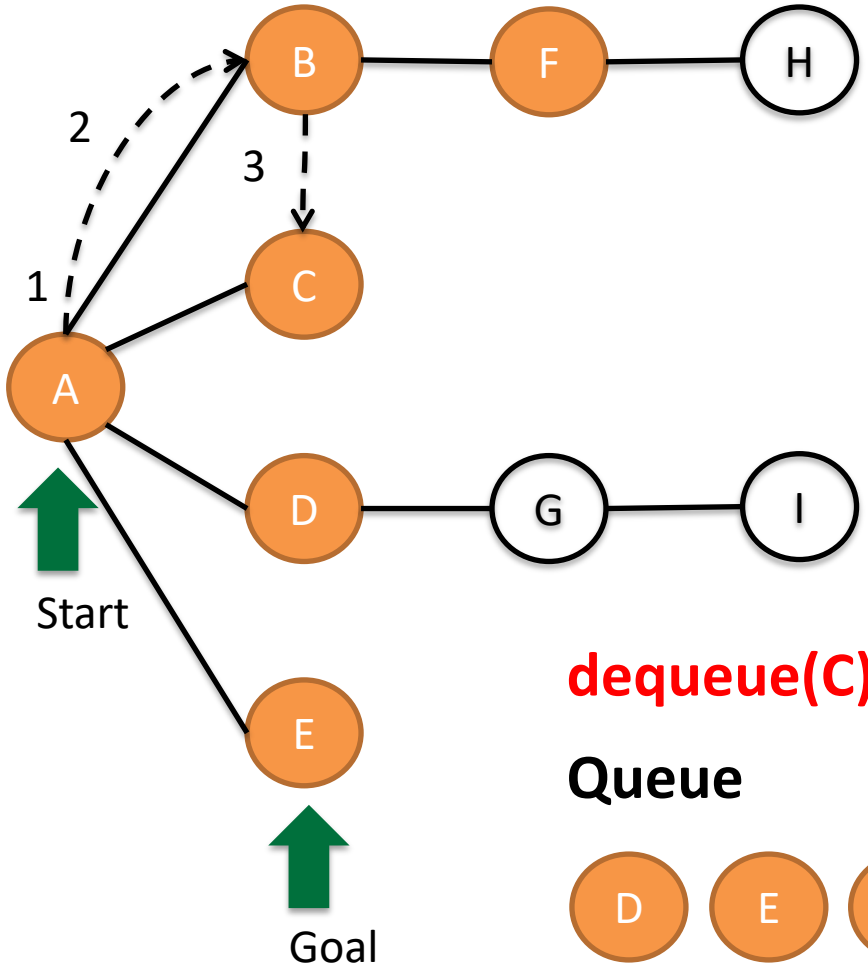
BFS algorithm

```
enqueue(s) //start node
s.visited = true
repeat until find goal vertex or
queue empty:
    u = dequeque()
    (do something here)
    for v ∈ u.adjacent
        if !v.visited
            v.visited = true
            enqueue(v)
```

dequeue(B), enqueue unvisited adjacent F (not A)

Queue

Breadth First Search (BFS) can find the shortest path between nodes



Order nodes visited

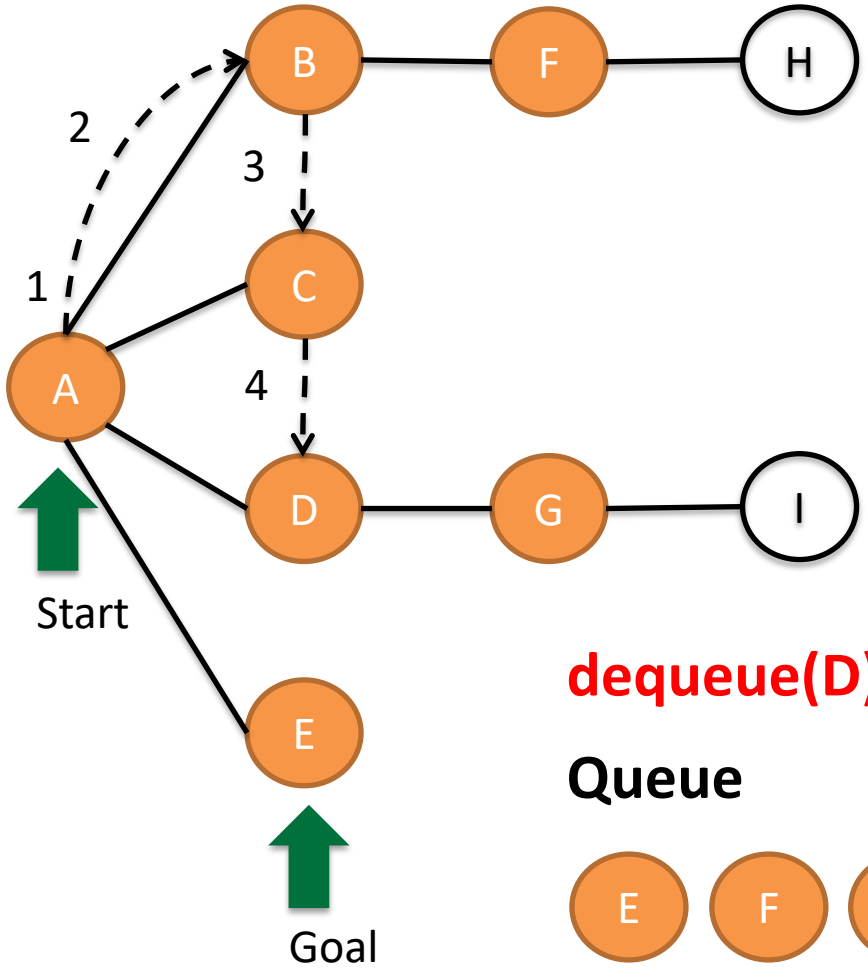
BFS algorithm

```
enqueue(s) //start node
s.visited = true
repeat until find goal vertex or
queue empty:
    u = dequeque()
    (do something here)
    for v ∈ u.adjacent
        if !v.visited
            v.visited = true
            enqueue(v)
```

dequeue(C), enqueue unvisited adjacent (none)

Queue

Breadth First Search (BFS) can find the shortest path between nodes



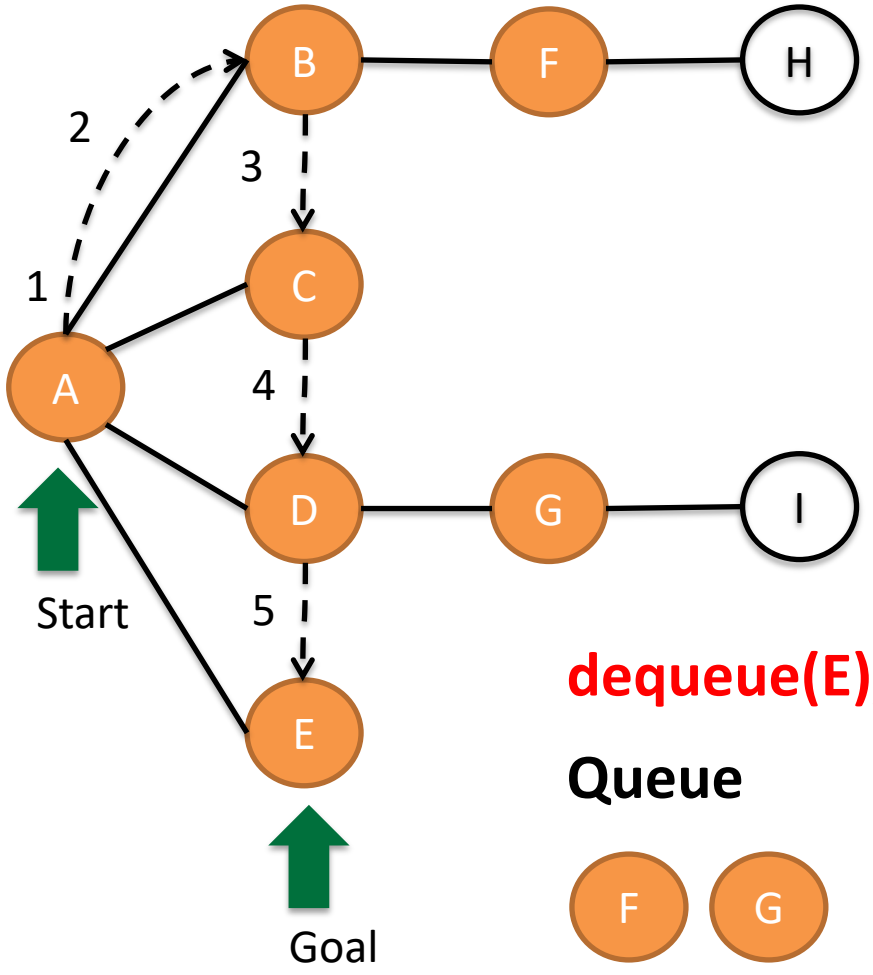
BFS algorithm

```
enqueue(s) //start node
s.visited = true
repeat until find goal vertex or
queue empty:
    u = dequeque()
    (do something here)
    for v ∈ u.adjacent
        if !v.visited
            v.visited = true
            enqueue(v)
```

dequeue(D), enqueue unvisited adjacent G (not A)

Queue

Breadth First Search (BFS) can find the shortest path between nodes



BFS algorithm

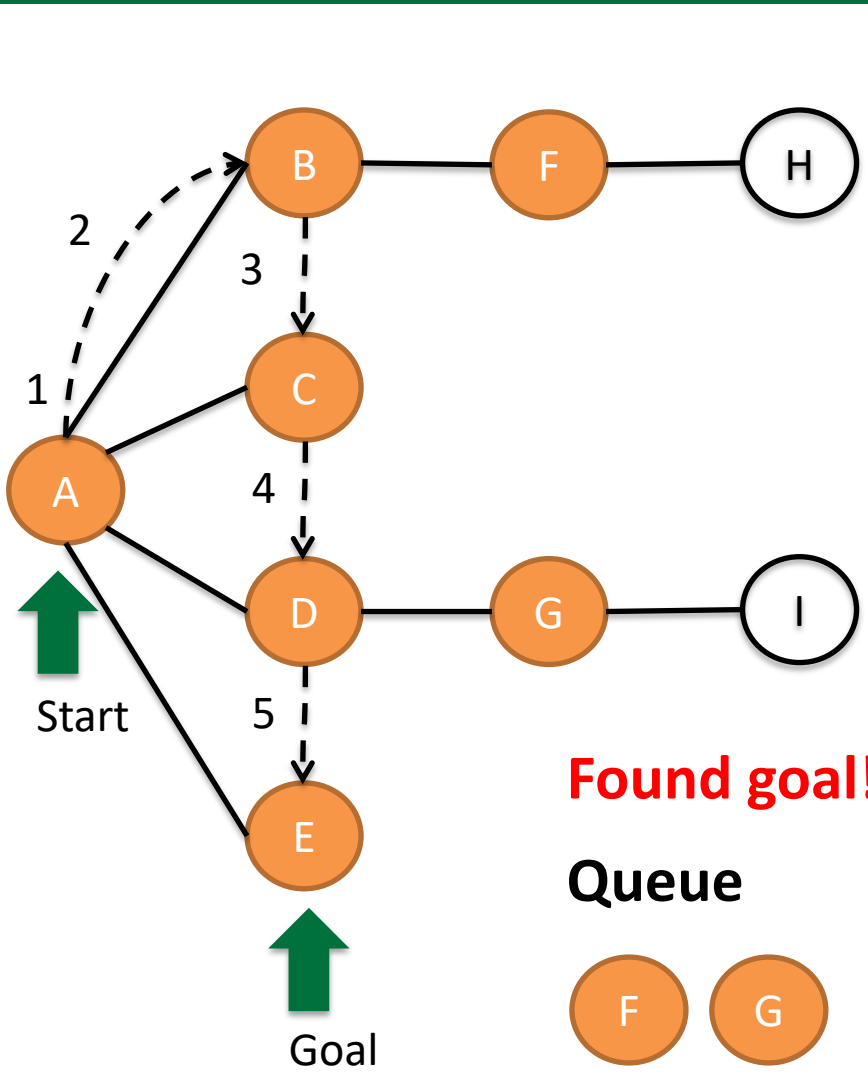
```
enqueue(s) //start node
s.visited = true
repeat until find goal vertex or
queue empty:
    u = dequeque()
    (do something here)
    for v ∈ u.adjacent
        if !v.visited
            v.visited = true
            enqueue(v)
```

dequeue(E), enqueue unvisited adjacent (none)

Queue

F G

Breadth First Search (BFS) can find the shortest path between nodes



BFS algorithm

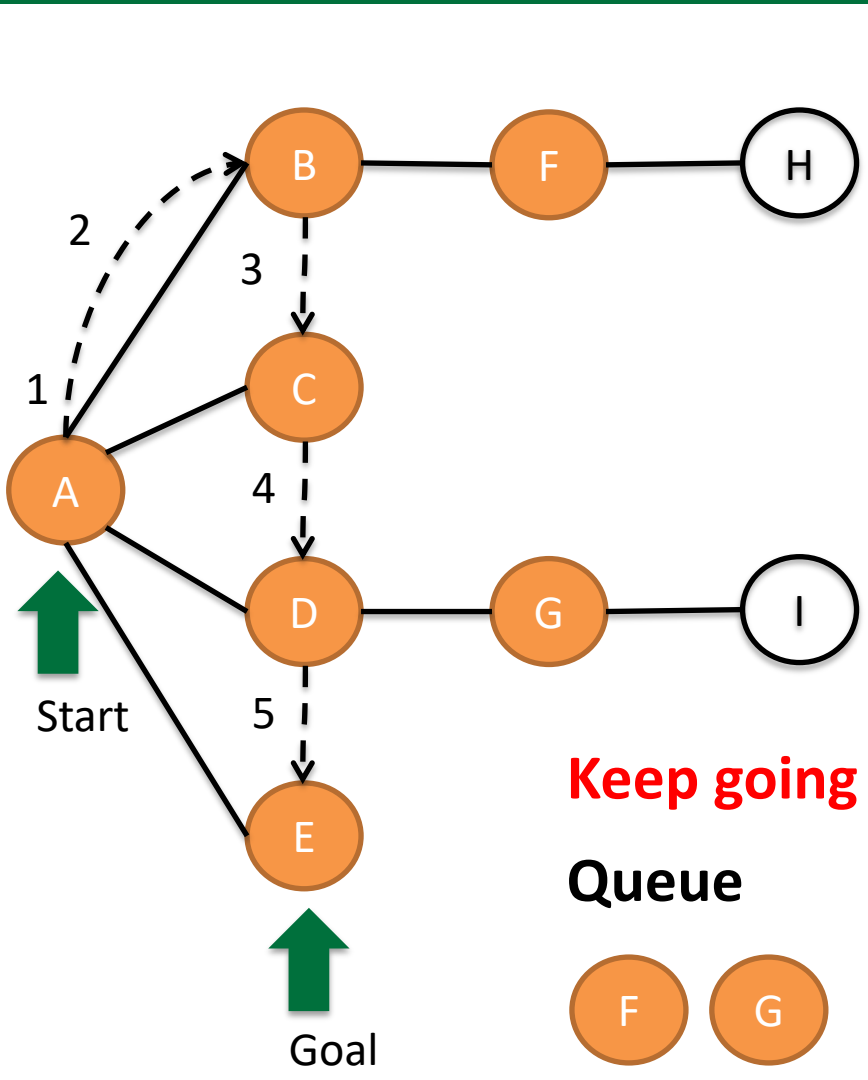
```
enqueue(s) //start node
s.visited = true
repeat until find goal vertex or
queue empty:
    u = dequeque()
    (do something here)
    for v ∈ u.adjacent
        if !v.visited
            v.visited = true
            enqueue(v)
```

Found goal! Can stop now

Queue

F G

Breadth First Search (BFS) can find the shortest path between nodes



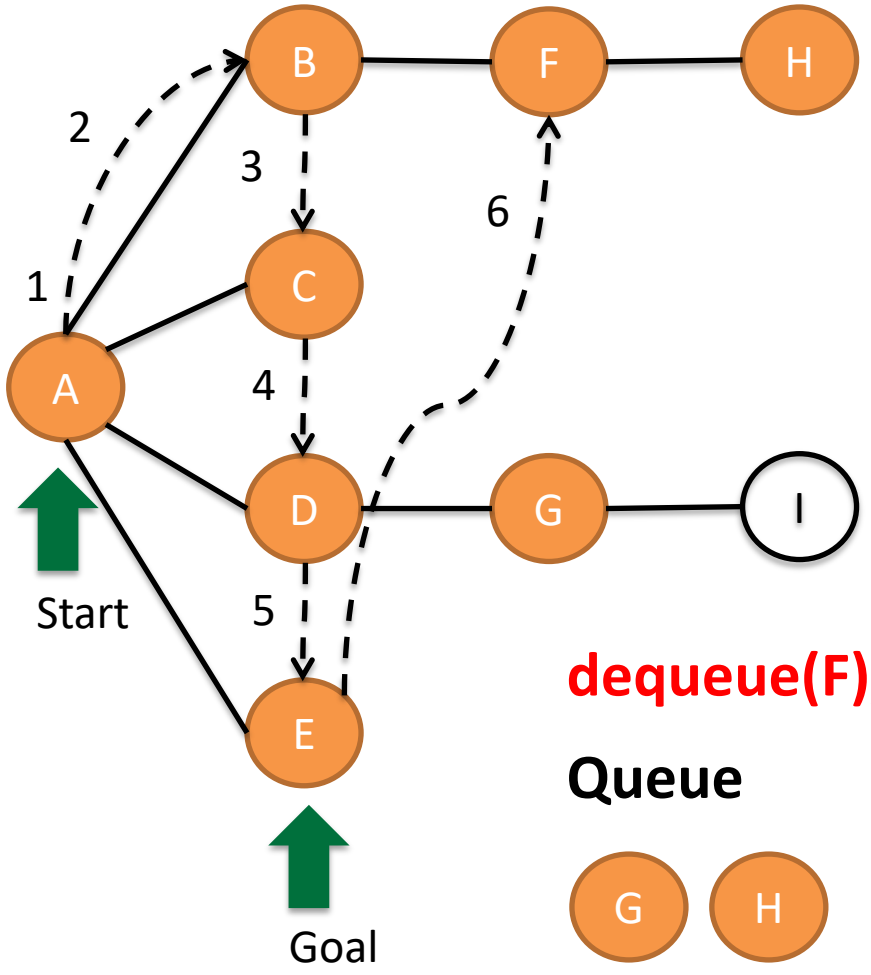
BFS algorithm

```
enqueue(s) //start node
s.visited = true
repeat until find goal vertex or
queue empty:
    u = dequeque()
    (do something here)
    for v ∈ u.adjacent
        if !v.visited
            v.visited = true
            enqueue(v)
```

Keep going to further illustrate BFS process

Queue

Breadth First Search (BFS) can find the shortest path between nodes



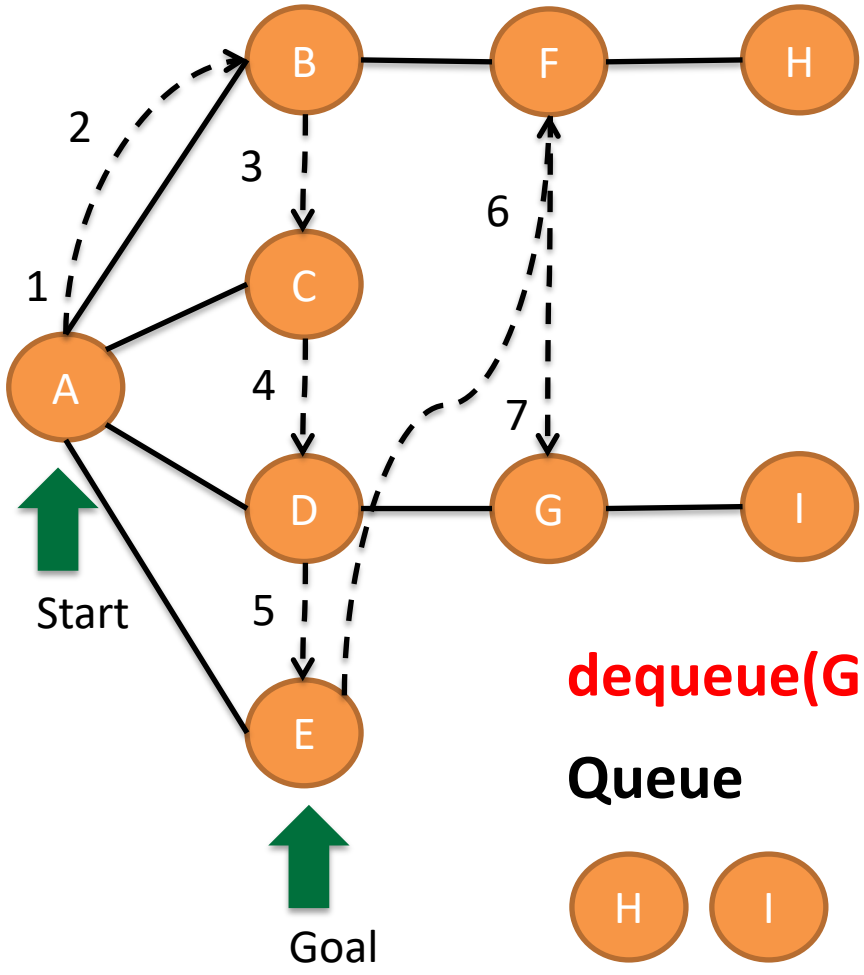
BFS algorithm

```
enqueue(s) //start node
s.visited = true
repeat until find goal vertex or
queue empty:
    u = dequeque()
    (do something here)
    for v ∈ u.adjacent
        if !v.visited
            v.visited = true
            enqueue(v)
```

dequeue(F), enqueue unvisited adjacent H

Queue

Breadth First Search (BFS) can find the shortest path between nodes



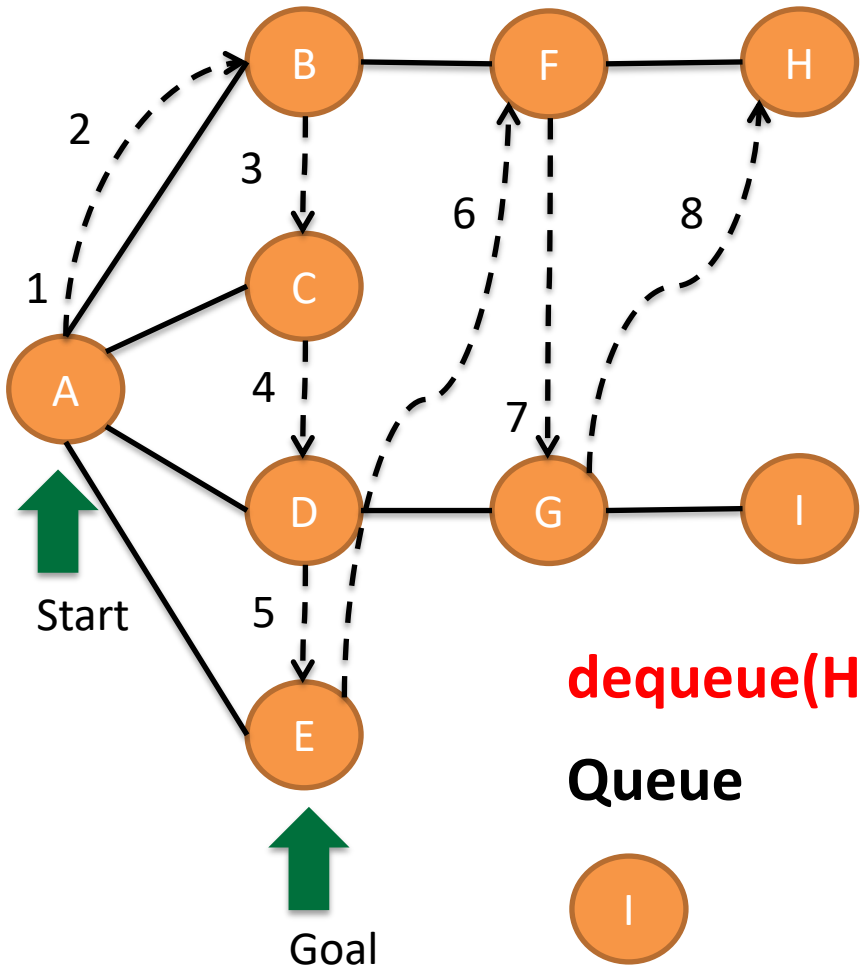
BFS algorithm

```
enqueue(s) //start node
s.visited = true
repeat until find goal vertex or
queue empty:
    u = dequeque()
    (do something here)
    for v ∈ u.adjacent
        if !v.visited
            v.visited = true
            enqueue(v)
```

dequeue(G), enqueue unvisited adjacent I

Queue

Breadth First Search (BFS) can find the shortest path between nodes



BFS algorithm

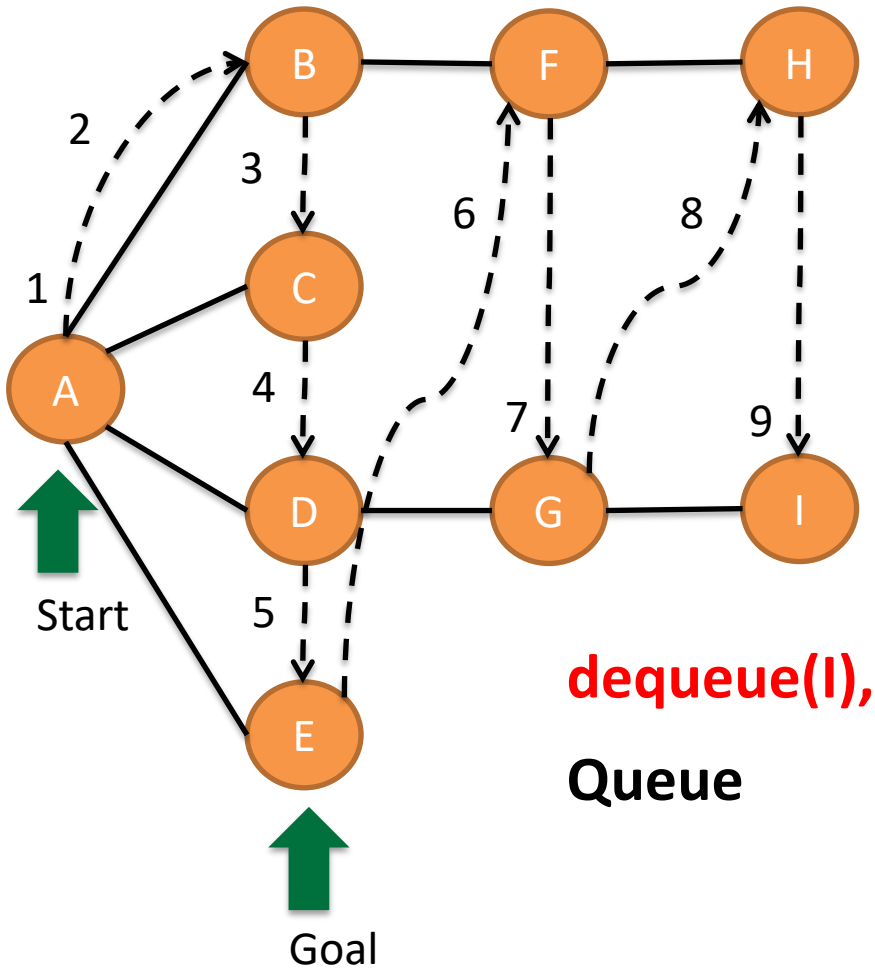
```
enqueue(s) //start node
s.visited = true
repeat until find goal vertex or
queue empty:
    u = dequeque()
    (do something here)
    for v ∈ u.adjacent
        if !v.visited
            v.visited = true
            enqueue(v)
```

dequeue(H), enqueue unvisited adjacent (none)

Queue

I

Breadth First Search (BFS) can find the shortest path between nodes



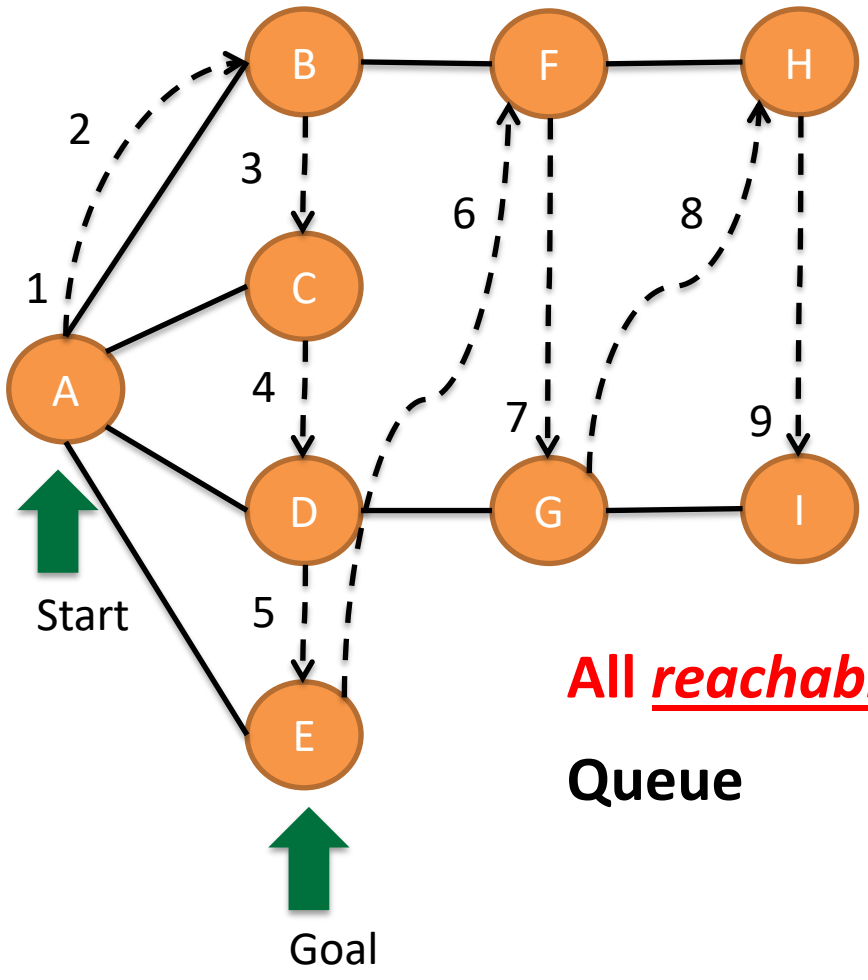
BFS algorithm

```
enqueue(s) //start node
s.visited = true
repeat until find goal vertex or
queue empty:
    u = dequeque()
    (do something here)
    for v ∈ u.adjacent
        if !v.visited
            v.visited = true
            enqueue(v)
```

dequeue(I), enqueue unvisited adjacent (none)

Queue

Breadth First Search (BFS) can find the shortest path between nodes



BFS algorithm

```
enqueue(s) //start node
s.visited = true
repeat until find goal vertex or
queue empty:
    u = dequeque()
    (do something here)
    for v ∈ u.adjacent
        if !v.visited
            v.visited = true
            enqueue(v)
```

All *reachable* nodes from *start* explored

Queue

Node discovery tells us something about the graph

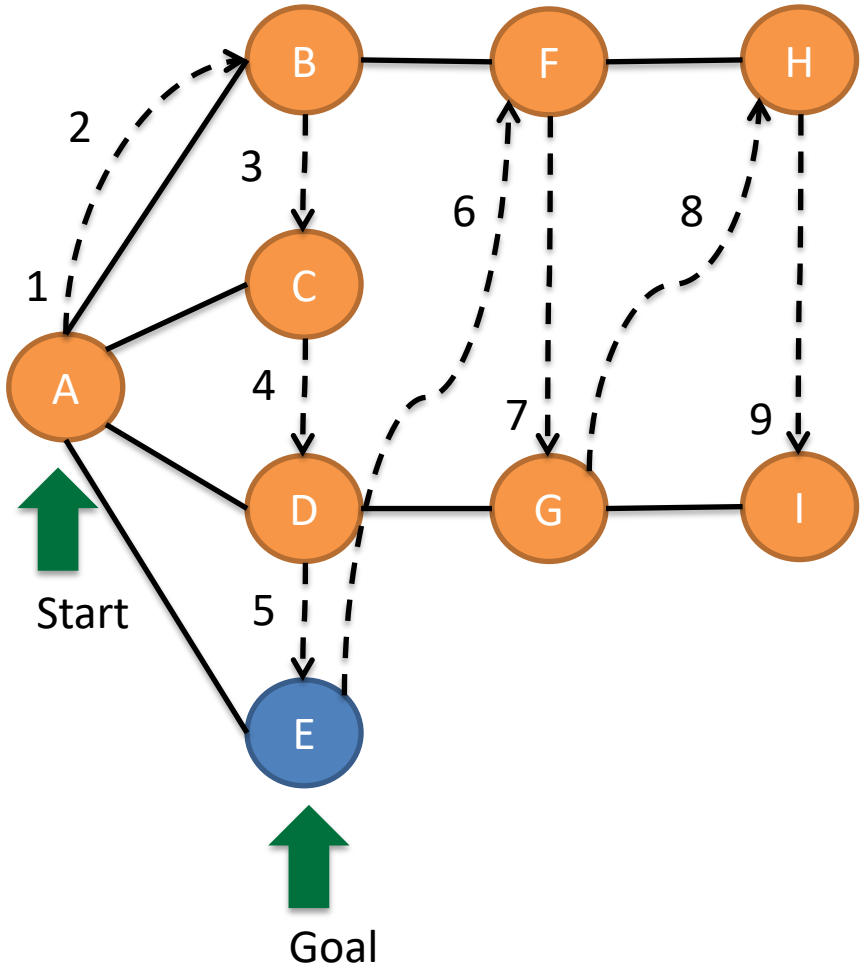
Discovery edges

- Edges that lead to unvisited nodes called *discovery edges*
- Discovery edges form a tree on the graph (root, no cycles)
- Can traverse from start to goal on tree (if goal reachable)
- Can tell us which nodes are not reachable (not on path formed by discovery edges)
- **Path guaranteed to have smallest number of edges**

Can track how we got to node to find shortest path

- Keep track of parent vertex
- Parent of each vertex is vertex that discovered it
- Parent is unique because we don't visit vertices twice

To find path from *start* to *v*, keep track of previous node as nodes are "discovered"



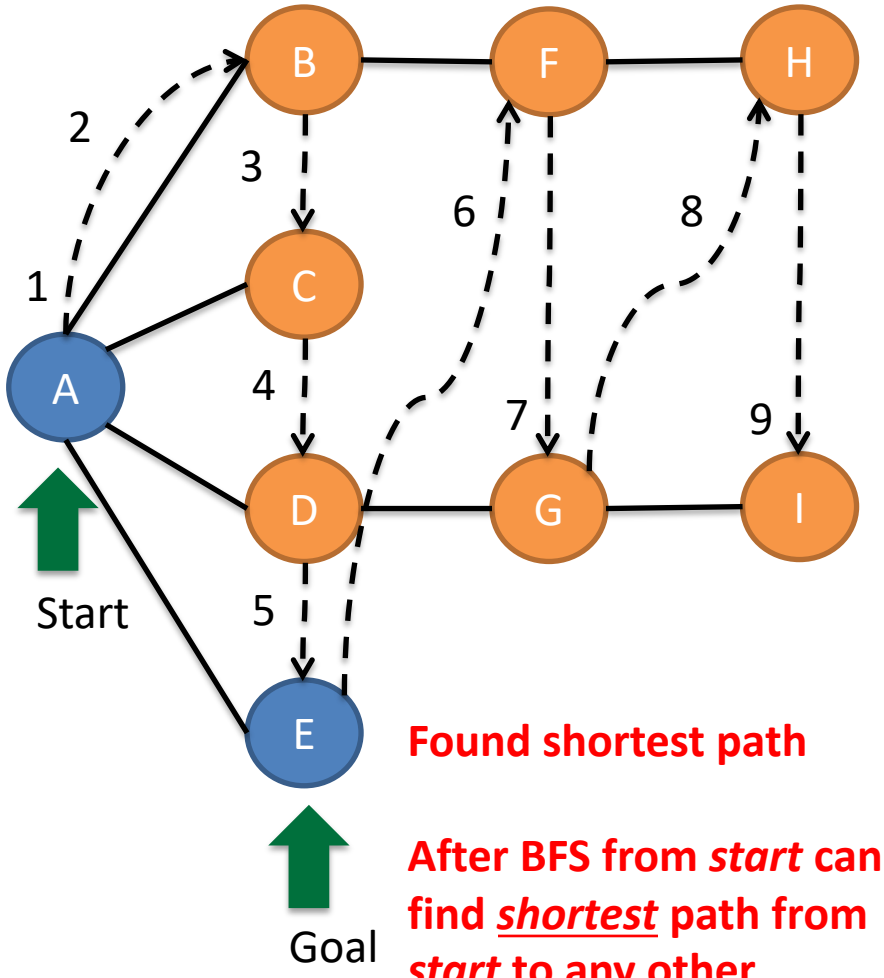
Path from *start* to *v*

- Each time node discovered, keep pointer to previous
- Could keep Map with node as Key and previous as Value
- Start at goal node (*v*)
- Track backward until find starting node
- **Will find shortest path if it exists**

Path A to E

Key	Value
A	Null
B	A
C	A
D	A
E	A
F	B
G	D
H	F
I	G

To find path from *start* to *v*, keep track of previous node as nodes are "discovered"



Found shortest path

After BFS from *start* can find *shortest* path from *start* to any other reachable node

Path from *start* to *v*

- Each time node discovered, keep pointer to previous
- Could keep Map with node as Key and previous as Value
- Start at goal node (*v*)
- Track backward until find starting node
- **Will find shortest path if it exists**

Path A to E

Key	Value
A	Null
B	A
C	A
D	A
E	**A**
F	B
G	D
H	F
I	G

Path A,E

BFS run-time complexity is O(n+m)

Run time

- Assume graph with *n* nodes and *m* edges
- Visit each node at most one time (due to visited indicator
- Visit each edge at most one time
- Run-time complexity O(n+m)

BFS algorithm

```
enqueue(s) //start node
s.visited = true
repeat until find goal vertex or
queue empty:
    u = dequeque()
    (do something here)
    for v ∈ u.adjacent
        if !v.visited
            v.visited = true
            enqueue(v)
```

GraphTraversal.java: BFS code

GraphTraversal.java

```
60    public void BFS(AdjacencyMapGraph<V,E> G, V start) {
61        System.out.println("\nBreadth First Search from " + start);
62        backTrack = new HashMap<V,V>(); //initialize backTrack
63        backTrack.put(start, null); //load start vertex with null parent
64        Set<V> visited = new HashSet<V>(); //Set to track which vertices have already been visited
65        Queue<V> queue = new LinkedList<V>(); //queue to implement BFS
66
67        queue.add(start); //enqueue start vertex
68        visited.add(start); //add start to visited Set
69        while (!queue.isEmpty()) { //loop until no more vertices
70            V u = queue.remove(); //dequeue
71            for (V v : G.outNeighbors(u)) { //loop over out neighbors
72                if (!visited.contains(v)) { //if neighbor not visited, then neighbor is discovered
73                    visited.add(v); //add neighbor to visited Set
74                    queue.add(v); //enqueue neighbor
75                    backTrack.put(v, u); //save that this vertex was discovered from prior vertex
76                }
77            }
78        }
79    }
```

Agenda

1. Depth first search (DFS)
2. Breadth first search (BFS)
3. Examples from last class and today

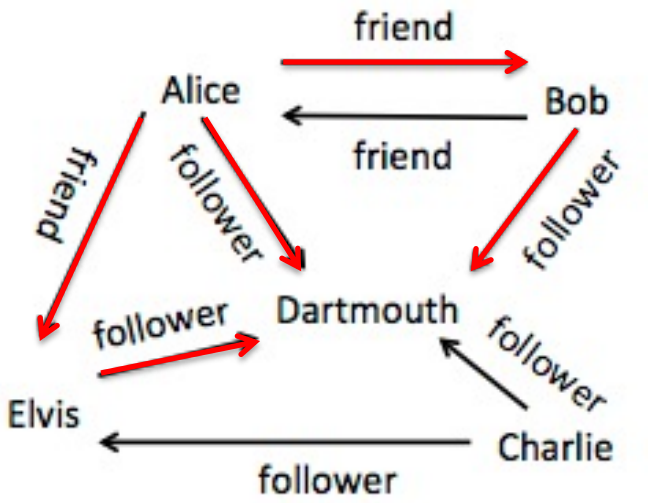
GraphTraversal.java: DFS and BFS on graph we looked at last class

```
113    public static void main(String[] args) {
114        //set up graph from class introducing Graphs
115        GraphTraversal<String,String> GT = new GraphTraversal<String,String>();
116        AdjacencyMapGraph<String,String> g = new AdjacencyMapGraph<String,String>();
117        g.insertVertex("Alice");
118        g.insertVertex("Bob");
119        g.insertVertex("Charlie");
120        g.insertVertex("Dartmouth");
121        g.insertVertex("Elvis");
122        g.insertDirected("Alice", "Dartmouth", "follower");
123        g.insertDirected("Bob", "Dartmouth", "follower");
124        g.insertDirected("Charlie", "Dartmouth", "follower");
125        g.insertDirected("Elvis", "Dartmouth", "follower");
126        g.insertUndirected("Alice", "Bob", "friend"); // symmetric, undirected edge
127        g.insertDirected("Alice", "Elvis", "friend"); // not symmetric, directed edge!
128        g.insertDirected("Charlie", "Elvis", "follower");
129
130        //run DFS from Alice
131        GT.DFS(g,"Alice");
132        //find path from start to end
133        GT.findPath("Bob", "Dartmouth"); //DFS wasn't run from Bob, should reject this
134        GT.findPath("Alice","Dartmouth");
135        GT.findPath("Alice","Charlie");
136        GT.findPath("Alice","Alice");
137
138        //run BFS
139        GT.BFS(g,"Alice");
140
141        //find path from start to end
142        GT.findPath("Alice","Dartmouth");
143        GT.findPath("Alice","Charlie");
144
```

Problems Javadoc Declaration Console Debug Expressions Error Log Call Hierarchy

<terminated> GraphTraversal [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Feb 12, 2018, 4:31:06 PM)

```
Depth First Search from Alice
Run DFS or BFS on Bob before trying to find a path
Path from Alice to Dartmouth
[Alice, Elvis, Dartmouth]
Path from Alice to Charlie
        No path found
Path from Alice to Alice
[Alice]

Breadth First Search from Alice
Path from Alice to Dartmouth
[Alice, Dartmouth]
Path from Alice to Charlie
        No path found
```



Key	Value
Alice	Null
Bob	Alice
Dartmouth	Elvis
Elvis	Alice

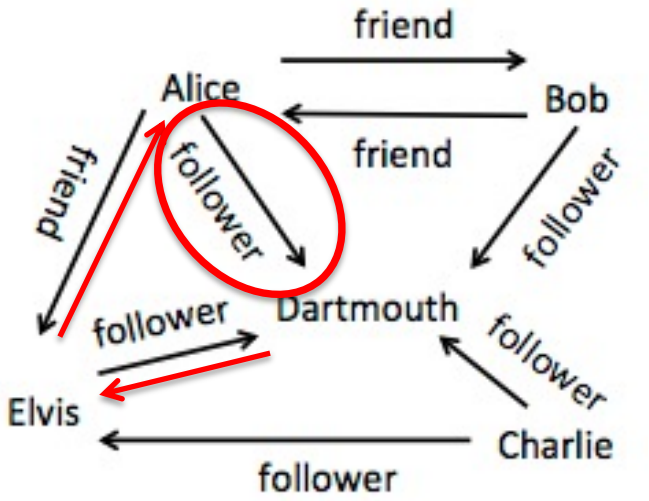
GraphTraversal.java: DFS and BFS on graph we looked at last class

```
113    public static void main(String[] args) {
114        //set up graph from class introducing Graphs
115        GraphTraversal<String,String> GT = new GraphTraversal<String,String>();
116        AdjacencyMapGraph<String,String> g = new AdjacencyMapGraph<String,String>();
117        g.insertVertex("Alice");
118        g.insertVertex("Bob");
119        g.insertVertex("Charlie");
120        g.insertVertex("Dartmouth");
121        g.insertVertex("Elvis");
122        g.insertDirected("Alice", "Dartmouth", "follower");
123        g.insertDirected("Bob", "Dartmouth", "follower");
124        g.insertDirected("Charlie", "Dartmouth", "follower");
125        g.insertDirected("Elvis", "Dartmouth", "follower");
126        g.insertUndirected("Alice", "Bob", "friend"); // symmetric, undirected edge
127        g.insertDirected("Alice", "Elvis", "friend"); // not symmetric, directed edge!
128        g.insertDirected("Charlie", "Elvis", "follower");
129
130        //run DFS from Alice
131        GT.DFS(g,"Alice");
132        //find path from start to end
133        GT.findPath("Bob", "Dartmouth"); //DFS wasn't run from Bob, should reject this
134        GT.findPath("Alice","Dartmouth");
135        GT.findPath("Alice","Charlie");
136        GT.findPath("Alice","Alice");
137
138        //run BFS
139        GT.BFS(g,"Alice");
140
141        //find path from start to end
142        GT.findPath("Alice","Dartmouth");
143        GT.findPath("Alice","Charlie");
144
```

Problems Javadoc Declaration Console Debug Expressions Error Log Call Hierarchy

<terminated> GraphTraversal [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Feb 12, 2018, 4:31:06 PM)

```
Depth First Search from Alice
Run DFS or BFS on Bob before trying to find a path
Path from Alice to Dartmouth
[Alice, Elvis, Dartmouth]
Path from Alice to Charlie
        No path found
Path from Alice to Alice
[Alice]

Breadth First Search from Alice
Path from Alice to Dartmouth
[Alice, Dartmouth]
Path from Alice to Charlie
        No path found
```



Key	Value
Alice	Null
Bob	Alice
Dartmouth	Elvis
Elvis	Alice

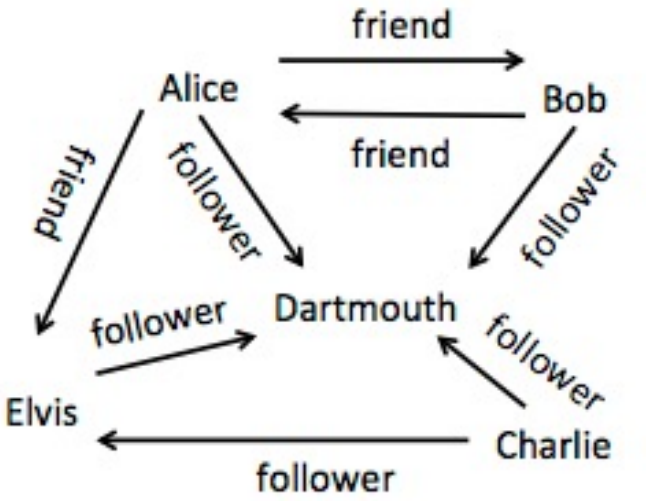
GraphTraversal.java: DFS and BFS on graph we looked at last class

```
113    public static void main(String[] args) {
114        //set up graph from class introducing Graphs
115        GraphTraversal<String,String> GT = new GraphTraversal<String,String>();
116        AdjacencyMapGraph<String,String> g = new AdjacencyMapGraph<String,String>();
117        g.insertVertex("Alice");
118        g.insertVertex("Bob");
119        g.insertVertex("Charlie");
120        g.insertVertex("Dartmouth");
121        g.insertVertex("Elvis");
122        g.insertDirected("Alice", "Dartmouth", "follower");
123        g.insertDirected("Bob", "Dartmouth", "follower");
124        g.insertDirected("Charlie", "Dartmouth", "follower");
125        g.insertDirected("Elvis", "Dartmouth", "follower");
126        g.insertUndirected("Alice", "Bob", "friend"); // symmetric, undirected edge
127        g.insertDirected("Alice", "Elvis", "friend"); // not symmetric, directed edge!
128        g.insertDirected("Charlie", "Elvis", "follower");
129
130        //run DFS from Alice
131        GT.DFS(g,"Alice");
132        //find path from start to end
133        GT.findPath("Bob", "Dartmouth"); //DFS wasn't run from Bob, should reject this
134        GT.findPath("Alice","Dartmouth");
135        GT.findPath("Alice","Charlie");
136        GT.findPath("Alice","Alice");
137
138        //run BFS
139        GT.BFS(g,"Alice");
140
141        //find path from start to end
142        GT.findPath("Alice","Dartmouth");
143        GT.findPath("Alice","Charlie");
144
```

Problems Javadoc Declaration Console Debug Expressions Error Log Call Hierarchy

<terminated> GraphTraversal [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Feb 12, 2018, 4:31:06 PM)

```
Depth First Search from Alice
Run DFS or BFS on Bob before trying to find a path
Path from Alice to Dartmouth
[Alice, Elvis, Dartmouth]
Path from Alice to Charlie
        No path found
Path from Alice to Alice
[Alice]

Breadth First Search from Alice
Path from Alice to Dartmouth
[Alice, Dartmouth]
Path from Alice to Charlie
        No path found
```



Key	Value
Alice	Null
Bob	Alice
Dartmouth	Elvis
Elvis	Alice

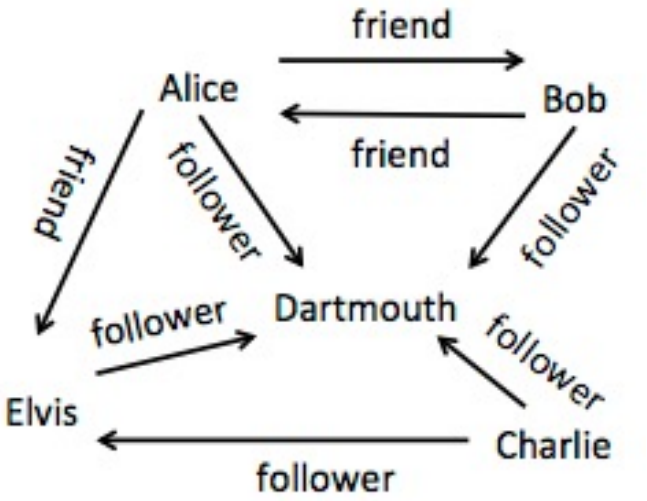
GraphTraversal.java: DFS and BFS on graph we looked at last class

```
113    public static void main(String[] args) {
114        //set up graph from class introducing Graphs
115        GraphTraversal<String,String> GT = new GraphTraversal<String,String>();
116        AdjacencyMapGraph<String,String> g = new AdjacencyMapGraph<String,String>();
117        g.insertVertex("Alice");
118        g.insertVertex("Bob");
119        g.insertVertex("Charlie");
120        g.insertVertex("Dartmouth");
121        g.insertVertex("Elvis");
122        g.insertDirected("Alice", "Dartmouth", "follower");
123        g.insertDirected("Bob", "Dartmouth", "follower");
124        g.insertDirected("Charlie", "Dartmouth", "follower");
125        g.insertDirected("Elvis", "Dartmouth", "follower");
126        g.insertUndirected("Alice", "Bob", "friend"); // symmetric, undirected edge
127        g.insertDirected("Alice", "Elvis", "friend"); // not symmetric, directed edge!
128        g.insertDirected("Charlie", "Elvis", "follower");
129
130        //run DFS from Alice
131        GT.DFS(g,"Alice");
132        //find path from start to end
133        GT.findPath("Bob", "Dartmouth"); //DFS wasn't run from Bob, should reject this
134        GT.findPath("Alice","Dartmouth");
135        GT.findPath("Alice","Charlie");
136        GT.findPath("Alice","Alice");
137
138        //run BFS
139        GT.BFS(g,"Alice");
140
141        //find path from start to end
142        GT.findPath("Alice","Dartmouth");
143        GT.findPath("Alice","Charlie");
144
```

Problems Javadoc Declaration Console Debug Expressions Error Log Call Hierarchy

<terminated> GraphTraversal [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Feb 12, 2018, 4:31:06 PM)

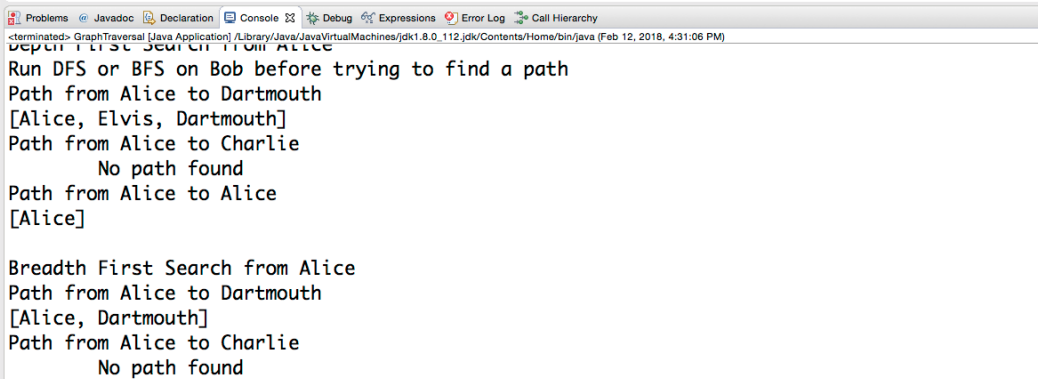
```
Depth First Search from Alice
Run DFS or BFS on Bob before trying to find a path
Path from Alice to Dartmouth
[Alice, Elvis, Dartmouth]
Path from Alice to Charlie
        No path found
Path from Alice to Alice
[Alice]

Breadth First Search from Alice
Path from Alice to Dartmouth
[Alice, Dartmouth]
Path from Alice to Charlie
        No path found
```

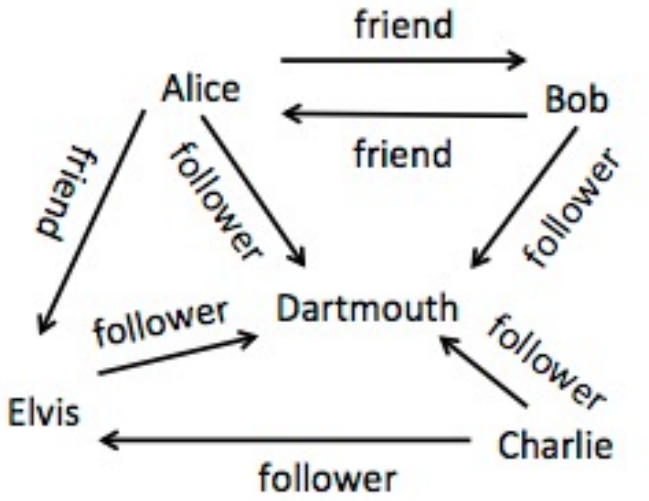


Key	Value
Alice	Null
Bob	Alice
Dartmouth	Elvis
Elvis	Alice

GraphTraversal.java: DFS and BFS on graph we looked at last class

```
113    public static void main(String[] args) {
114        //set up graph from class introducing Graphs
115        GraphTraversal<String,String> GT = new GraphTraversal<String,String>();
116        AdjacencyMapGraph<String,String> g = new AdjacencyMapGraph<String,String>();
117        g.insertVertex("Alice");
118        g.insertVertex("Bob");
119        g.insertVertex("Charlie");
120        g.insertVertex("Dartmouth");
121        g.insertVertex("Elvis");
122        g.insertDirected("Alice", "Dartmouth", "follower");
123        g.insertDirected("Bob", "Dartmouth", "follower");
124        g.insertDirected("Charlie", "Dartmouth", "follower");
125        g.insertDirected("Elvis", "Dartmouth", "follower");
126        g.insertUndirected("Alice", "Bob", "friend"); // symmetric, undirected edge
127        g.insertDirected("Alice", "Elvis", "friend"); // not symmetric, directed edge!
128        g.insertDirected("Charlie", "Elvis", "follower");
129
130        //run DFS from Alice
131        GT.DFS(g,"Alice");
132        //find path from start to end
133        GT.findPath("Bob", "Dartmouth"); //DFS wasn't run from Bob, should reject this
134        GT.findPath("Alice","Dartmouth");
135        GT.findPath("Alice","Charlie");
136        GT.findPath("Alice","Alice");
137
138        //run BFS
139        GT.BFS(g,"Alice");
140
141        //find path from start to end
142        GT.findPath("Alice","Dartmouth");
143        GT.findPath("Alice","Charlie");
144
```





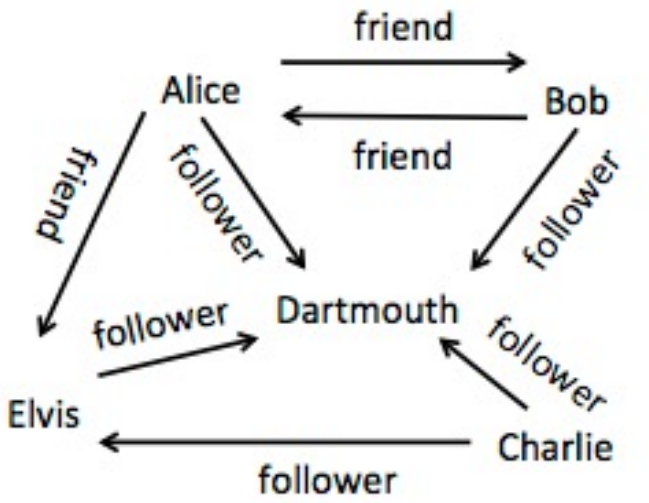
GraphTraversal.java: DFS and BFS on graph we looked at last class

```
113    public static void main(String[] args) {
114        //set up graph from class introducing Graphs
115        GraphTraversal<String,String> GT = new GraphTraversal<String,String>();
116        AdjacencyMapGraph<String,String> g = new AdjacencyMapGraph<String,String>();
117        g.insertVertex("Alice");
118        g.insertVertex("Bob");
119        g.insertVertex("Charlie");
120        g.insertVertex("Dartmouth");
121        g.insertVertex("Elvis");
122        g.insertDirected("Alice", "Dartmouth", "follower");
123        g.insertDirected("Bob", "Dartmouth", "follower");
124        g.insertDirected("Charlie", "Dartmouth", "follower");
125        g.insertDirected("Elvis", "Dartmouth", "follower");
126        g.insertUndirected("Alice", "Bob", "friend"); // symmetric, undirected edge
127        g.insertDirected("Alice", "Elvis", "friend"); // not symmetric, directed edge!
128        g.insertDirected("Charlie", "Elvis", "follower");
129
130        //run DFS from Alice
131        GT.DFS(g,"Alice");
132        //find path from start to end
133        GT.findPath("Bob", "Dartmouth"); //DFS wasn't run from Bob, should reject this
134        GT.findPath("Alice","Dartmouth");
135        GT.findPath("Alice","Charlie");
136        GT.findPath("Alice","Alice");
137
138        //run BFS
139        GT.BFS(g,"Alice");
140
141        //find path from start to end
142        GT.findPath("Alice","Dartmouth");
143        GT.findPath("Alice","Charlie");
144
```

Problems Javadoc Declaration Console Debug Expressions Error Log Call Hierarchy

<terminated> GraphTraversal [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Feb 12, 2018, 4:31:06 PM)

```
Depth First Search from Alice
Run DFS or BFS on Bob before trying to find a path
Path from Alice to Dartmouth
[Alice, Elvis, Dartmouth]
Path from Alice to Charlie
        No path found
Path from Alice to Alice
[Alice]

Breadth First Search from Alice
Path from Alice to Dartmouth
[Alice, Dartmouth]
Path from Alice to Charlie
        No path found
```



Key	Value
Alice	Null
Bob	Alice
Dartmouth	Alice
Elvis	Alice

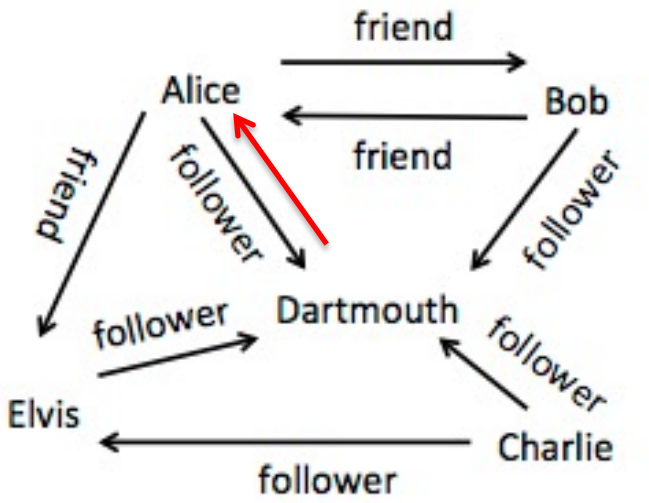
GraphTraversal.java: DFS and BFS on graph we looked at last class

```
113    public static void main(String[] args) {
114        //set up graph from class introducing Graphs
115        GraphTraversal<String,String> GT = new GraphTraversal<String,String>();
116        AdjacencyMapGraph<String,String> g = new AdjacencyMapGraph<String,String>();
117        g.insertVertex("Alice");
118        g.insertVertex("Bob");
119        g.insertVertex("Charlie");
120        g.insertVertex("Dartmouth");
121        g.insertVertex("Elvis");
122        g.insertDirected("Alice", "Dartmouth", "follower");
123        g.insertDirected("Bob", "Dartmouth", "follower");
124        g.insertDirected("Charlie", "Dartmouth", "follower");
125        g.insertDirected("Elvis", "Dartmouth", "follower");
126        g.insertUndirected("Alice", "Bob", "friend"); // symmetric, undirected edge
127        g.insertDirected("Alice", "Elvis", "friend"); // not symmetric, directed edge!
128        g.insertDirected("Charlie", "Elvis", "follower");
129
130        //run DFS from Alice
131        GT.DFS(g,"Alice");
132        //find path from start to end
133        GT.findPath("Bob", "Dartmouth"); //DFS wasn't run from Bob, should reject this
134        GT.findPath("Alice","Dartmouth");
135        GT.findPath("Alice","Charlie");
136        GT.findPath("Alice","Alice");
137
138        //run BFS
139        GT.BFS(g,"Alice");
140
141        //find path from start to end
142        GT.findPath("Alice","Dartmouth");
143        GT.findPath("Alice","Charlie");
144
```

Problems Javadoc Declaration Console Debug Expressions Error Log Call Hierarchy

<terminated> GraphTraversal [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Feb 12, 2018, 4:31:06 PM)

```
Depth First Search from Alice
Run DFS or BFS on Bob before trying to find a path
Path from Alice to Dartmouth
[Alice, Elvis, Dartmouth]
Path from Alice to Charlie
        No path found
Path from Alice to Alice
[Alice]

Breadth First Search from Alice
Path from Alice to Dartmouth
[Alice, Dartmouth]
Path from Alice to Charlie
        No path found
```

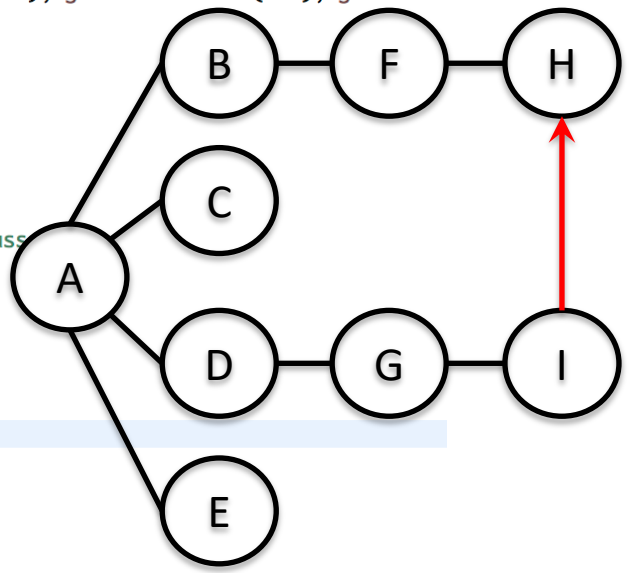


Key	Value
Alice	Null
Bob	Alice
Dartmouth	Alice
Elvis	Alice

DFS on today's graph

GraphTraversal.java

```
145         //set up graph from Graph Traversal class
146         AdjacencyMapGraph<String,String> g2 = new AdjacencyMapGraph<String,String>();
147         g2.insertVertex("A"); g2.insertVertex("B"); g2.insertVertex("C"); g2.insertVertex("D");
148         g2.insertVertex("E"); g2.insertVertex("F"); g2.insertVertex("G"); g2.insertVertex("H"); g2.insert
149         g2.insertUndirected("A", "B", "");
150         g2.insertUndirected("B", "F", "");
151         g2.insertUndirected("F", "H", "");
152         g2.insertUndirected("A", "C", "");
153         g2.insertUndirected("A", "D", "");
154         g2.insertUndirected("D", "G", "");
155         g2.insertUndirected("G", "I", "");
156         g2.insertUndirected("A", "E", "");
157         g2.insertDirected("I", "H", ""); //directed edge not from class
158
159         //run DFS from A and find path to H
160         GT.DFS(g2,"A");
161         GT.findPath("A", "B");
162
163         //run BFS from A and find path to H
164         GT.BFS(g2,"A");
165         GT.findPath("A", "B");
166
167     }
168
169 }
170
```



Problems Javadoc Declaration Console Debug Expressions Error Log Call Hierarchy

<terminated> GraphTraversal [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Feb 12, 2018, 5:14:16 PM)

```
Depth First Search from A
Path from A to B
[A, D, G, I, H, F, B]

Breadth First Search from A
Path from A to B
[A, B]
```

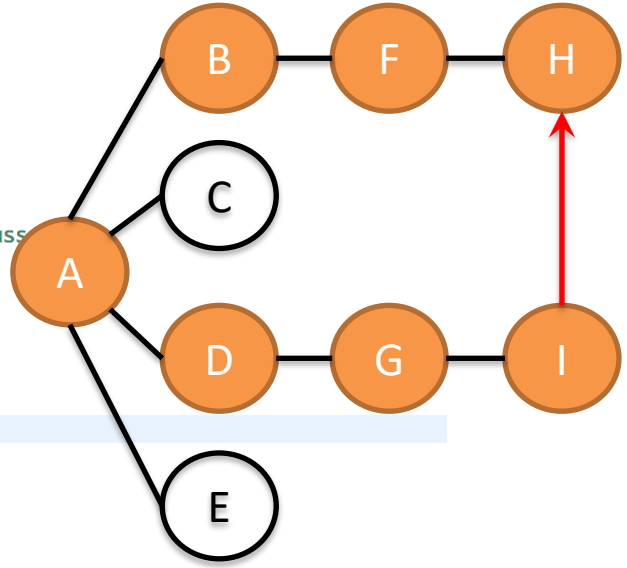
DFS

Key	Value
A	Null
B	F
C	A
D	A
E	A
F	H
G	D
H	I
I	G

DFS on today's graph

GraphTraversal.java

```
145         //set up graph from Graph Traversal class
146         AdjacencyMapGraph<String,String> g2 = new AdjacencyMapGraph<String,String>();
147         g2.insertVertex("A"); g2.insertVertex("B"); g2.insertVertex("C"); g2.insertVertex("D");
148         g2.insertVertex("E"); g2.insertVertex("F"); g2.insertVertex("G"); g2.insertVertex("H"); g2.insert
149         g2.insertUndirected("A", "B", "");
150         g2.insertUndirected("B", "F", "");
151         g2.insertUndirected("F", "H", "");
152         g2.insertUndirected("A", "C", "");
153         g2.insertUndirected("A", "D", "");
154         g2.insertUndirected("D", "G", "");
155         g2.insertUndirected("G", "I", "");
156         g2.insertUndirected("A", "E", "");
157         g2.insertDirected("I", "H", ""); //directed edge not from class
158
159         //run DFS from A and find path to H
160         GT.DFS(g2,"A");
161         GT.findPath("A", "B");
162
163         //run BFS from A and find path to H
164         GT.BFS(g2,"A");
165         GT.findPath("A", "B");
166
167     }
168
169 }
170
```



Problems Javadoc Declaration Console Debug Expressions Error Log Call Hierarchy

<terminated> GraphTraversal [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Feb 12, 2018, 5:14:16 PM)

```
Depth First Search from A
Path from A to B
[A, D, G, I, H, F, B]

Breadth First Search from A
Path from A to B
[A, B]
```

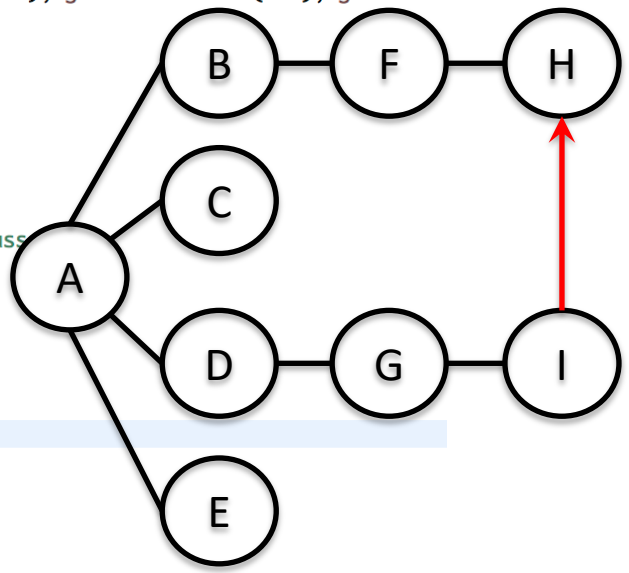
DFS

Key	Value
A	Null
B	F
C	A
D	A
E	A
F	H
G	D
H	I
I	G

BFS on today's graph

GraphTraversal.java

```
145         //set up graph from Graph Traversal class
146         AdjacencyMapGraph<String,String> g2 = new AdjacencyMapGraph<String,String>();
147         g2.insertVertex("A"); g2.insertVertex("B"); g2.insertVertex("C"); g2.insertVertex("D");
148         g2.insertVertex("E"); g2.insertVertex("F"); g2.insertVertex("G"); g2.insertVertex("H"); g2.insert
149         g2.insertUndirected("A", "B", "");
150         g2.insertUndirected("B", "F", "");
151         g2.insertUndirected("F", "H", "");
152         g2.insertUndirected("A", "C", "");
153         g2.insertUndirected("A", "D", "");
154         g2.insertUndirected("D", "G", "");
155         g2.insertUndirected("G", "I", "");
156         g2.insertUndirected("A", "E", "");
157         g2.insertDirected("I", "H", ""); //directed edge not from class
158
159         //run DFS from A and find path to H
160         GT.DFS(g2,"A");
161         GT.findPath("A", "B");
162
163         //run BFS from A and find path to H
164         GT.BFS(g2,"A");
165         GT.findPath("A", "B");
166
167     }
168
169 }
170
```



Problems Javadoc Declaration Console Debug Expressions Error Log Call Hierarchy

<terminated> GraphTraversal [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Feb 12, 2018, 5:14:16 PM)

```
Depth First Search from A
Path from A to B
[A, D, G, I, H, F, B]

Breadth First Search from A
Path from A to B
[A, B]
```

BFS

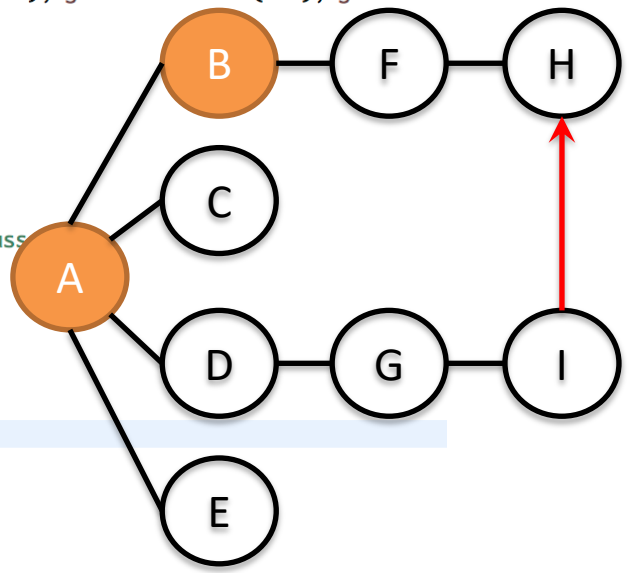
Key	Value
A	Null
B	**A**
C	A
D	A
E	A
F	B
G	D
H	F
I	G

BFS on today's graph

GraphTraversal.java

```
145         //set up graph from Graph Traversal class
146         AdjacencyMapGraph<String,String> g2 = new AdjacencyMapGraph<String,String>();
147         g2.insertVertex("A"); g2.insertVertex("B"); g2.insertVertex("C"); g2.insertVertex("D");
148         g2.insertVertex("E"); g2.insertVertex("F"); g2.insertVertex("G"); g2.insertVertex("H"); g2.insert
149         g2.insertUndirected("A", "B", "");
150         g2.insertUndirected("B", "F", "");
151         g2.insertUndirected("F", "H", "");
152         g2.insertUndirected("A", "C", "");
153         g2.insertUndirected("A", "D", "");
154         g2.insertUndirected("D", "G", "");
155         g2.insertUndirected("G", "I", "");
156         g2.insertUndirected("A", "E", "");
157         g2.insertDirected("I", "H", ""); //directed edge not from class
158
159         //run DFS from A and find path to H
160         GT.DFS(g2,"A");
161         GT.findPath("A", "B");
162
163         //run BFS from A and find path to H
164         GT.BFS(g2,"A");
165         GT.findPath("A", "B");
166
167     }
168
169 }
170
```

- **Run BFS**
- **Find path A to B**




```
Problems  Javadoc  Declaration  Console  Debug  Expressions  Error Log  Call Hierarchy
<terminated> GraphTraversal [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Feb 12, 2018, 5:14:16 PM)

Depth First Search from A
Path from A to B
[A, D, G, I, H, F, B]

Breadth First Search from A
Path from A to B
[A, B]
```

BFS

Key	Value
A	Null
B	**A**
C	A
D	A
E	A
F	B
G	D
H	F
I	G

PS-4

actors.txt

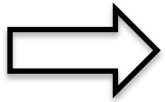
...
18|Loretta Devine
19|Whitney Houston
...

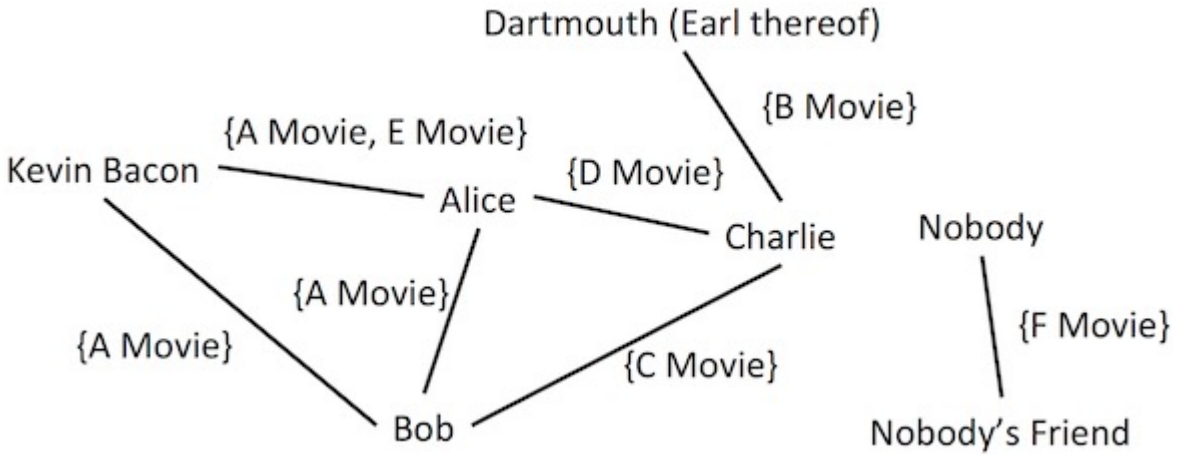
movies.txt

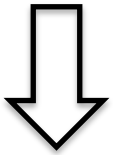
1|Toy Story (1995)
2|Jumanji (1995)
...

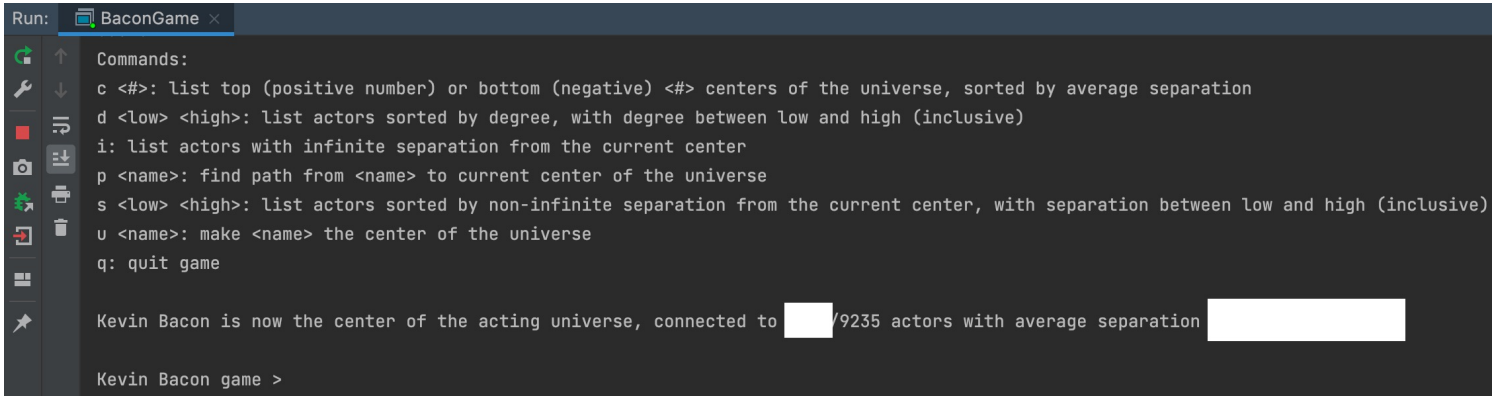
movie-actors.txt

1|2
1|3
...









https://www.cs.dartmouth.edu/cs10/PS-4.html

Summary

- To find paths from a start to another node
  - DFS uses stack to find paths from a start node to another node
  - BFS uses queue
- Both have a complexity of O(n+m) when keeping track of visited nodes
- Backtrack map to keep track of paths

Next

- Graph traversals that use cost

Additional Resources

GRAPH TRAVERSAL NOTES

Graph traversals are useful to answer questions about relationships

Some Graph traversals uses

- Uses are often around *reachability*
- Computing *path* from vertex `u` to vertex `v`
- Given start vertex `s` of Graph `G`, compute a path with the minimum number of edges between `s` and all other vertices (or report no such path exists)
- Testing whether `G` is fully connected (e.g., all vertices reachable)
- Identifying *cycles* in `G` (or reporting no cycle exists)
- Today's examples have ~~no~~ few cycles (next class)

DFS Node discovery tells us something about the graph

Discovery edges

- Edges that lead to unvisited nodes called *discovery edges*
- Discovery edges form a tree on the graph (root, no cycles)
- Can traverse from start to goal on tree (if goal reachable)
- Can tell us which nodes are not reachable (not on path formed by discovery edges)
- **With DFS, path not guaranteed to be shortest path!**

Back, cross, and forward edges

- Edges that lead to previously discovered nodes
- Back edges lead to ancestor nodes, forward edges to descendants, cross edges to non-ancestor or descendant
- These edges indicate presence of a cycle in the Graph
- Today's focus on graphs without cycles

BFS Node discovery tells us something about the graph

Discovery edges

- Edges that lead to unvisited nodes called *discovery edges*
- Discovery edges form a tree on the graph (root, no cycles)
- Can traverse from start to goal on tree (if goal reachable)
- Can tell us which nodes are not reachable (not on path formed by discovery edges)
- **Path guaranteed to have smallest number of edges**

Can track how we got to node to find shortest path

- Keep track of parent vertex
- Parent of each vertex is vertex that discovered it
- Parent is unique because we don't visit vertices twice

GraphTraversal.java: DFS

ANNOTATED SLIDES

GraphTraversal.java: DFS code

```
17 public class GraphTraversal<V,E> {
18
19     public Map<V,V> backTrack; //keep track of prior vertex when v
20
21     /**
22      * Constructor.  Initialize backTrack to new HashMap.
23      */
24     public GraphTraversal() {
25         backTrack = new HashMap<V,V>();
26     }
27
28     /**
29      * Depth First Search
30      * @param G -- graph to search
31      * @param start -- starting vertex
32      */
33     public void DFS(AdjacencyMapGraph<V,E> G, V start) {
34         System.out.println("\nDepth First Search from " + start);
35         backTrack = new HashMap<V,V>(); //initialize backTrack
36         backTrack.put(start, null); //load start node with null pa
37         Set<V> visited = new HashSet<V>(); //Set to track which ve
38         Stack<V> stack = new Stack<V>(); //stack to implement DFS
39
40         stack.push(start); //push start vertex
41         while (!stack.isEmpty()) { //loop until no more vertices
42             V u = stack.pop(); //get most recent vertex
43             if (!visited.contains(u)) { //if not already visited
44                 visited.add(u); //add to visited Set
45                 for (V v : G.outNeighbors(u)) { //loop over out ne
46                     if (!visited.contains(v)) { //if neighbor not
47                         stack.push(v); //push non-visited neighbor
48                         backTrack.put(v, u); //save that this vert
49                     }
50                 }
51             }
52         }
53     }
```

- When running DFS (or BFS), keep track of prior vertex when a vertex is discovered
- Map Key is current vertex, Value is prior vertex

DFS – given Graph *G* and *start* vertex

- Use *Set* to track visited vertices
- Use *Stack* to track vertices to visit
- Follow pseudo code from previous slides
- Add vertex to *backTrack* when discovered
- Only discovered vertices are added, non-reachable vertices not added to *backTrack*

After *DFS* can get from *start* to any reachable node in Graph using *backTrack*

GraphTraversal.java: findPath

ANNOTATED SLIDES

After DFS (or BFS) *findPath()* finds a path from start to end if it exists

GraphTraversal.java

```
88      public ArrayList<V> findPath(V start, V end) {
89          //check that DFS or BFS have already been run from start
90          if (backTrack.isEmpty() || !backTrack.containsKey(start) ||
91                  (backTrack.containsKey(start) && backTrack.get(start) != null)) {
92              System.out.println("Run DFS or BFS on " + start + " before trying to find a path");
93              return new ArrayList<V>();
94          }
95          System.out.println("Path from " + start + " to " + end);
96          //make sure end vertex in backTrack
97          if (!backTrack.containsKey(end)) {
98              System.out.println("\tNo path found");
99              return new ArrayList<V>();
100         }
101         //start from end vertex and work backward to start vertex
102         ArrayList<V> path = new ArrayList<V>(); //this will hold the path from start to end verte
103         V current = end; //start at end vertex
104         //loop from end vertex back to start vertex
105         while (current != null) {
106             path.add(0,current); //add this vertex to front of arraylist path
107             current = backTrack.get(current); //get vertex that discovered this vertex
108         }
109         System.out.println(path);
110         return path;
111     }
```

Make sure DFS or BFS has been previously run from *start* vertex

Make sure *end* vertex reachable

- **Run time complexity?**
- **Length of path from *start* to *end***

- **Loop backward from *end* to *start* (parent null)**
- **Add new vertices to front of *path***
- **Return *path* when done**

GraphTraversal.java: BFS

ANNOTATED SLIDES

After DFS (or BFS) *findPath()* finds a path from start to end if it exists

GraphTraversal.java

```
88      public ArrayList<V> findPath(V start, V end) {
89          //check that DFS or BFS have already been run from start
90          if (backTrack.isEmpty() || !backTrack.containsKey(start) ||
91                  (backTrack.containsKey(start) && backTrack.get(start) != null)) {
92              System.out.println("Run DFS or BFS on " + start + " before trying to find a path");
93              return new ArrayList<V>();
94          }
95          System.out.println("Path from " + start + " to " + end);
96          //make sure end vertex in backTrack
97          if (!backTrack.containsKey(end)) {
98              System.out.println("\tNo path found");
99              return new ArrayList<V>();
100         }
101         //start from end vertex and work backward to start vertex
102         ArrayList<V> path = new ArrayList<V>(); //this will hold the path from start to end verte
103         V current = end; //start at end vertex
104         //loop from end vertex back to start vertex
105         while (current != null) {
106             path.add(0,current); //add this vertex to front of arraylist path
107             current = backTrack.get(current); //get vertex that discovered this vertex
108         }
109         System.out.println(path);
110         return path;
111     }
```

Make sure DFS or BFS has been previously run from *start* vertex

Make sure *end* vertex reachable

- **Run time complexity?**
- **Length of path from *start* to *end***

- **Loop backward from *end* to *start* (parent null)**
- **Add new vertices to front of *path***
- **Return *path* when done**

GraphTraversal.java: From example of last class

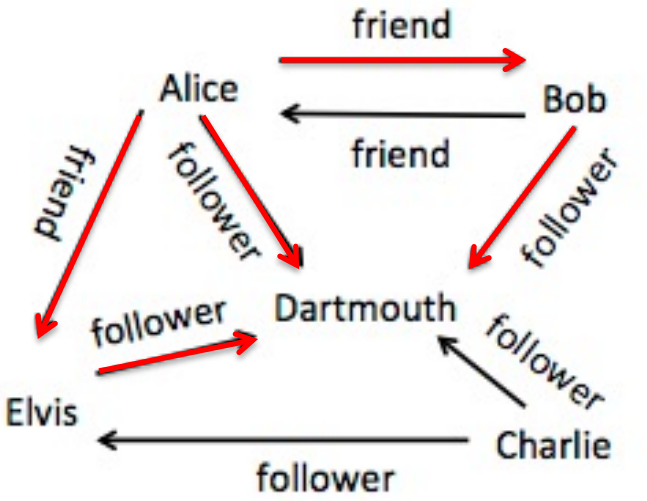
ANNOTATED SLIDES

GraphTraversal.java: DFS and BFS on graph we looked at last class

```
113    public static void main(String[] args) {
114        //set up graph from class introducing Graphs
115        GraphTraversal<String,String> GT = new GraphTraversal<String,String>();
116        AdjacencyMapGraph<String,String> g = new AdjacencyMapGraph<String,String>();
117        g.insertVertex("Alice");
118        g.insertVertex("Bob");
119        g.insertVertex("Charlie");
120        g.insertVertex("Dartmouth");
121        g.insertVertex("Elvis");
122        g.insertDirected("Alice", "Dartmouth", "follower");
123        g.insertDirected("Bob", "Dartmouth", "follower");
124        g.insertDirected("Charlie", "Dartmouth", "follower");
125        g.insertDirected("Elvis", "Dartmouth", "follower");
126        g.insertUndirected("Alice", "Bob", "friend"); // symmetric, undirected edge
127        g.insertDirected("Alice", "Elvis", "friend"); // not symmetric, directed edge!
128        g.insertDirected("Charlie", "Elvis", "follower");
129
130        //run DFS from Alice
131        GT.DFS(g,"Alice");
132        //find path from start to end
133        GT.findPath("Bob", "Dartmouth"); //DFS wasn't run from Bob, should reject this
134        GT.findPath("Alice","Dartmouth");
135        GT.findPath("Alice","Charlie");
136        GT.findPath("Alice","Alice");
137
138        //run BFS
139        GT.BFS(g,"Alice");
140
141        //find path from start to end
142        GT.findPath("Alice","Dartmouth");
143        GT.findPath("Alice","Charlie");
144
```

Create graph

- **Run DFS with start=Alice**
- **Find paths from Alice**



Key	Value
Alice	Null
Bob	Alice
Dartmouth	Elvis
Elvis	Alice

```
Problems  Javadoc  Declaration  Console  Debug  Expressions  Error Log  Call Hierarchy
<terminated> GraphTraversal [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Feb 12, 2018, 4:31:06 PM)
Depth First Search from Alice
Run DFS or BFS on Bob before trying to find a path
Path from Alice to Dartmouth
[Alice, Elvis, Dartmouth]
Path from Alice to Charlie
        No path found
Path from Alice to Alice
[Alice]

Breadth First Search from Alice
Path from Alice to Dartmouth
[Alice, Dartmouth]
Path from Alice to Charlie
        No path found
```

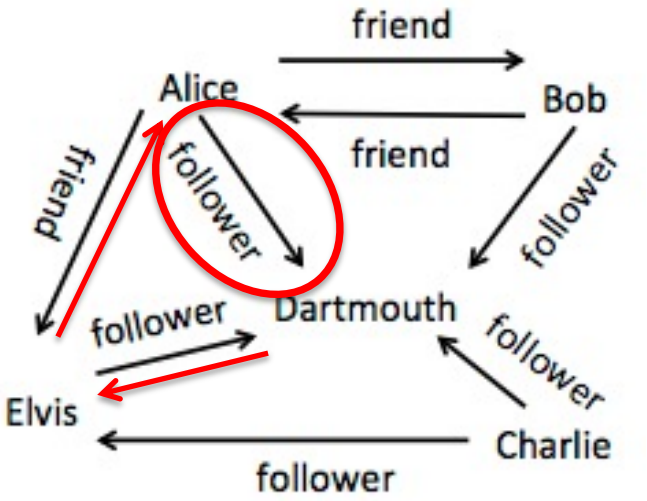
findPath("Bob", "Dartmouth")
DFS not run from Bob
DFS was run from Alice
So do not do this search (check *findPath* code)

GraphTraversal.java: DFS and BFS on graph we looked at last class

```
113    public static void main(String[] args) {
114        //set up graph from class introducing Graphs
115        GraphTraversal<String,String> GT = new GraphTraversal<String,String>();
116        AdjacencyMapGraph<String,String> g = new AdjacencyMapGraph<String,String>();
117        g.insertVertex("Alice");
118        g.insertVertex("Bob");
119        g.insertVertex("Charlie");
120        g.insertVertex("Dartmouth");
121        g.insertVertex("Elvis");
122        g.insertDirected("Alice", "Dartmouth", "follower");
123        g.insertDirected("Bob", "Dartmouth", "follower");
124        g.insertDirected("Charlie", "Dartmouth", "follower");
125        g.insertDirected("Elvis", "Dartmouth", "follower");
126        g.insertUndirected("Alice", "Bob", "friend"); // symmetric, undirected edge
127        g.insertDirected("Alice", "Elvis", "friend"); // not symmetric, directed edge!
128        g.insertDirected("Charlie", "Elvis", "follower");
129
130        //run DFS from Alice
131        GT.DFS(g,"Alice");
132        //find path from start to end
133        GT.findPath("Bob", "Dartmouth"); //DFS wasn't run from Bob, should reject this
134        GT.findPath("Alice","Dartmouth");
135        GT.findPath("Alice","Charlie");
136        GT.findPath("Alice","Alice");
137
138        //run BFS
139        GT.BFS(g,"Alice");
140
141        //find path from start to end
142        GT.findPath("Alice","Dartmouth");
143        GT.findPath("Alice","Charlie");
144
```

Create graph

- **Run DFS with start=Alice**
- **Find paths from Alice**



Key	Value
Alice	Null
Bob	Alice
Dartmouth	Elvis
Elvis	Alice

```
Problems  Javadoc  Declaration  Console  Debug  Expressions  Error Log  Call Hierarchy
<terminated> GraphTraversal [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Feb 12, 2018, 4:31:06 PM)
Depth First Search from Alice
Run DFS or BFS on Bob before trying to find a path
Path from Alice to Dartmouth
[Alice, Elvis, Dartmouth]
Path from Alice to Charlie
        No path found
Path from Alice to Alice
[Alice]

Breadth First Search from Alice
Path from Alice to Dartmouth
[Alice, Dartmouth]
Path from Alice to Charlie
        No path found
```

findPath("Alice", "Dartmouth") finds path Alice->Elvis->Dartmouth

Path yes, but not shortest path

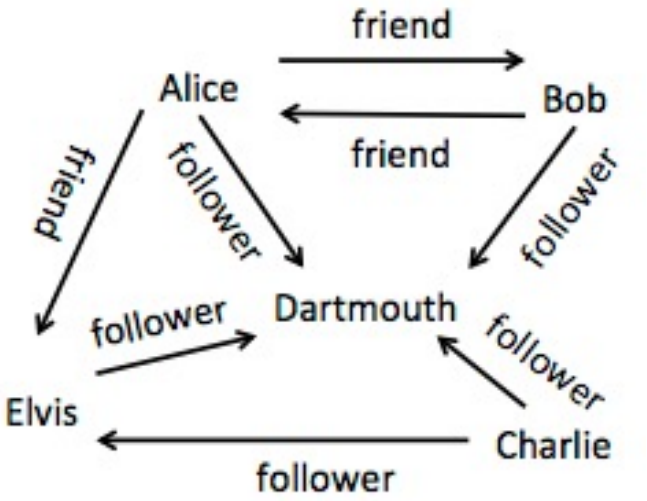
Shortest is Alice->Dartmouth

GraphTraversal.java: DFS and BFS on graph we looked at last class

```
113    public static void main(String[] args) {
114        //set up graph from class introducing Graphs
115        GraphTraversal<String,String> GT = new GraphTraversal<String,String>();
116        AdjacencyMapGraph<String,String> g = new AdjacencyMapGraph<String,String>();
117        g.insertVertex("Alice");
118        g.insertVertex("Bob");
119        g.insertVertex("Charlie");
120        g.insertVertex("Dartmouth");
121        g.insertVertex("Elvis");
122        g.insertDirected("Alice", "Dartmouth", "follower");
123        g.insertDirected("Bob", "Dartmouth", "follower");
124        g.insertDirected("Charlie", "Dartmouth", "follower");
125        g.insertDirected("Elvis", "Dartmouth", "follower");
126        g.insertUndirected("Alice", "Bob", "friend"); // symmetric, undirected edge
127        g.insertDirected("Alice", "Elvis", "friend"); // not symmetric, directed edge!
128        g.insertDirected("Charlie", "Elvis", "follower");
129
130        //run DFS from Alice
131        GT.DFS(g,"Alice");
132        //find path from start to end
133        GT.findPath("Bob", "Dartmouth"); //DFS wasn't run from Bob, should reject this
134        GT.findPath("Alice","Dartmouth");
135        GT.findPath("Alice","Charlie");
136        GT.findPath("Alice","Alice");
137
138        //run BFS
139        GT.BFS(g,"Alice");
140
141        //find path from start to end
142        GT.findPath("Alice","Dartmouth");
143        GT.findPath("Alice","Charlie");
144
```

Create graph

- **Run DFS with start=Alice**
- **Find paths from Alice**



Key	Value
Alice	Null
Bob	Alice
Dartmouth	Elvis
Elvis	Alice

```
Problems  Javadoc  Declaration  Console  Debug  Expressions  Error Log  Call Hierarchy
<terminated> GraphTraversal [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Feb 12, 2018, 4:31:06 PM)
Depth First Search from Alice
Run DFS or BFS on Bob before trying to find a path
Path from Alice to Dartmouth
[Alice, Elvis, Dartmouth]
Path from Alice to Charlie
        No path found
Path from Alice to Alice
[Alice]

Breadth First Search from Alice
Path from Alice to Dartmouth
[Alice, Dartmouth]
Path from Alice to Charlie
        No path found
```

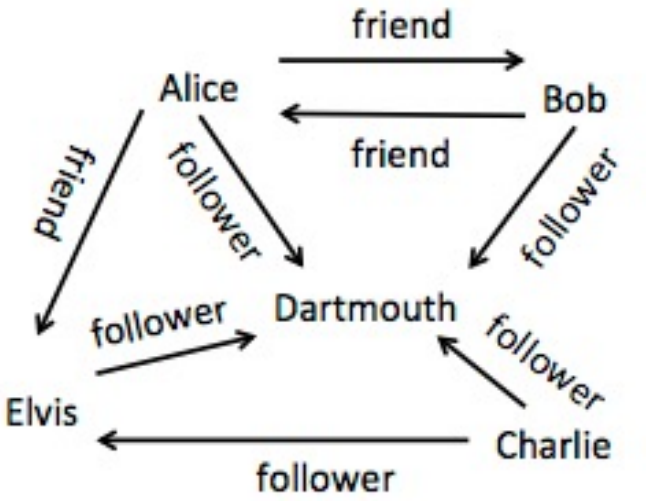
Alice can't reach Charlie in this graph
Charlie is not in *backTrack*

GraphTraversal.java: DFS and BFS on graph we looked at last class

```
113    public static void main(String[] args) {
114        //set up graph from class introducing Graphs
115        GraphTraversal<String,String> GT = new GraphTraversal<String,String>();
116        AdjacencyMapGraph<String,String> g = new AdjacencyMapGraph<String,String>();
117        g.insertVertex("Alice");
118        g.insertVertex("Bob");
119        g.insertVertex("Charlie");
120        g.insertVertex("Dartmouth");
121        g.insertVertex("Elvis");
122        g.insertDirected("Alice", "Dartmouth", "follower");
123        g.insertDirected("Bob", "Dartmouth", "follower");
124        g.insertDirected("Charlie", "Dartmouth", "follower");
125        g.insertDirected("Elvis", "Dartmouth", "follower");
126        g.insertUndirected("Alice", "Bob", "friend"); // symmetric, undirected edge
127        g.insertDirected("Alice", "Elvis", "friend"); // not symmetric, directed edge!
128        g.insertDirected("Charlie", "Elvis", "follower");
129
130        //run DFS from Alice
131        GT.DFS(g,"Alice");
132        //find path from start to end
133        GT.findPath("Bob", "Dartmouth"); //DFS wasn't run from Bob, should reject this
134        GT.findPath("Alice","Dartmouth");
135        GT.findPath("Alice","Charlie");
136        GT.findPath("Alice","Alice");
137
138        //run BFS
139        GT.BFS(g,"Alice");
140
141        //find path from start to end
142        GT.findPath("Alice","Dartmouth");
143        GT.findPath("Alice","Charlie");
144
```

Create graph

- **Run DFS with start=Alice**
- **Find paths from Alice**



Key	Value
Alice	Null
Bob	Alice
Dartmouth	Elvis
Elvis	Alice

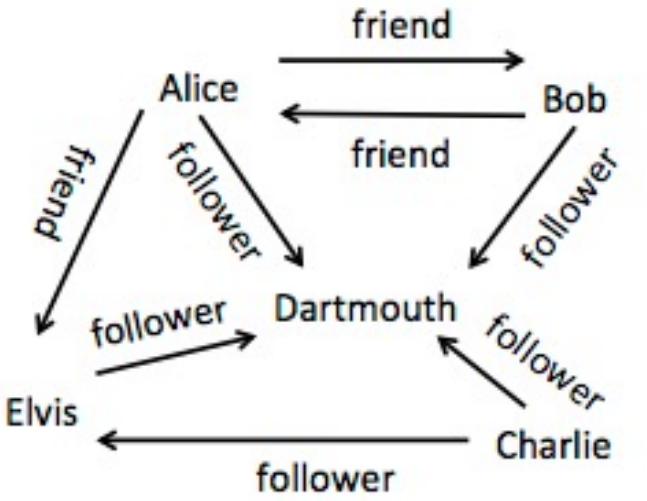
```
Problems  Javadoc  Declaration  Console  Debug  Expressions  Error Log  Call Hierarchy
<terminated> GraphTraversal [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Feb 12, 2018, 4:31:06 PM)
Depth First Search from Alice
Run DFS or BFS on Bob before trying to find a path
Path from Alice to Dartmouth
[Alice, Elvis, Dartmouth]
Path from Alice to Charlie
        No path found
Path from Alice to Alice
[Alice]

Breadth First Search from Alice
Path from Alice to Dartmouth
[Alice, Dartmouth]
Path from Alice to Charlie
        No path found
```

Alice can reach herself

GraphTraversal.java: DFS and BFS on graph we looked at last class

```
113    public static void main(String[] args) {
114        //set up graph from class introducing Graphs
115        GraphTraversal<String,String> GT = new GraphTraversal<String,String>();
116        AdjacencyMapGraph<String,String> g = new AdjacencyMapGraph<String,String>();
117        g.insertVertex("Alice");
118        g.insertVertex("Bob");
119        g.insertVertex("Charlie");
120        g.insertVertex("Dartmouth");
121        g.insertVertex("Elvis");
122        g.insertDirected("Alice", "Dartmouth", "follower");
123        g.insertDirected("Bob", "Dartmouth", "follower");
124        g.insertDirected("Charlie", "Dartmouth", "follower");
125        g.insertDirected("Elvis", "Dartmouth", "follower");
126        g.insertUndirected("Alice", "Bob", "friend"); // symmetric, undirected edge
127        g.insertDirected("Alice", "Elvis", "friend"); // not symmetric, directed edge!
128        g.insertDirected("Charlie", "Elvis", "follower");
129
130        //run DFS from Alice
131        GT.DFS(g,"Alice");
132        //find path from start to end
133        GT.findPath("Bob", "Dartmouth"); //DFS wasn't run from Bob, should reject this
134        GT.findPath("Alice","Dartmouth");
135        GT.findPath("Alice","Charlie");
136        GT.findPath("Alice","Alice");
137
138        //run BFS
139        GT.BFS(g,"Alice");
140
141        //find path from start to end
142        GT.findPath("Alice","Dartmouth");
143        GT.findPath("Alice","Charlie");
144
```

- **Run BFS start=Alice**



Problems Javadoc Declaration Console Debug Expressions Error Log Call Hierarchy

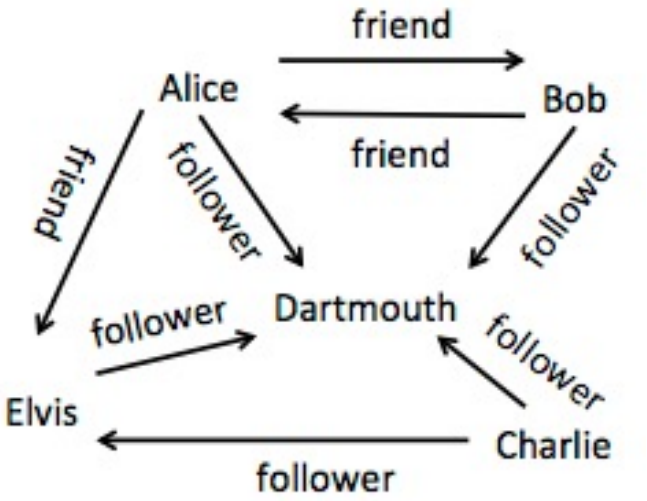
<terminated> GraphTraversal [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Feb 12, 2018, 4:31:06 PM)

```
Depth First Search from Alice
Run DFS or BFS on Bob before trying to find a path
Path from Alice to Dartmouth
[Alice, Elvis, Dartmouth]
Path from Alice to Charlie
        No path found
Path from Alice to Alice
[Alice]

Breadth First Search from Alice
Path from Alice to Dartmouth
[Alice, Dartmouth]
Path from Alice to Charlie
        No path found
```

GraphTraversal.java: DFS and BFS on graph we looked at last class

```
113    public static void main(String[] args) {
114        //set up graph from class introducing Graphs
115        GraphTraversal<String,String> GT = new GraphTraversal<String,String>();
116        AdjacencyMapGraph<String,String> g = new AdjacencyMapGraph<String,String>();
117        g.insertVertex("Alice");
118        g.insertVertex("Bob");
119        g.insertVertex("Charlie");
120        g.insertVertex("Dartmouth");
121        g.insertVertex("Elvis");
122        g.insertDirected("Alice", "Dartmouth", "follower");
123        g.insertDirected("Bob", "Dartmouth", "follower");
124        g.insertDirected("Charlie", "Dartmouth", "follower");
125        g.insertDirected("Elvis", "Dartmouth", "follower");
126        g.insertUndirected("Alice", "Bob", "friend"); // symmetric, undirected edge
127        g.insertDirected("Alice", "Elvis", "friend"); // not symmetric, directed edge!
128        g.insertDirected("Charlie", "Elvis", "follower");
129
130        //run DFS from Alice
131        GT.DFS(g,"Alice");
132        //find path from start to end
133        GT.findPath("Bob", "Dartmouth"); //DFS wasn't run from Bob, should reject this
134        GT.findPath("Alice","Dartmouth");
135        GT.findPath("Alice","Charlie");
136        GT.findPath("Alice","Alice");
137
138        //run BFS
139        GT.BFS(g,"Alice");
140
141        //find path from start to end
142        GT.findPath("Alice","Dartmouth");
143        GT.findPath("Alice","Charlie");
144
```

- **Run BFS start=Alice**



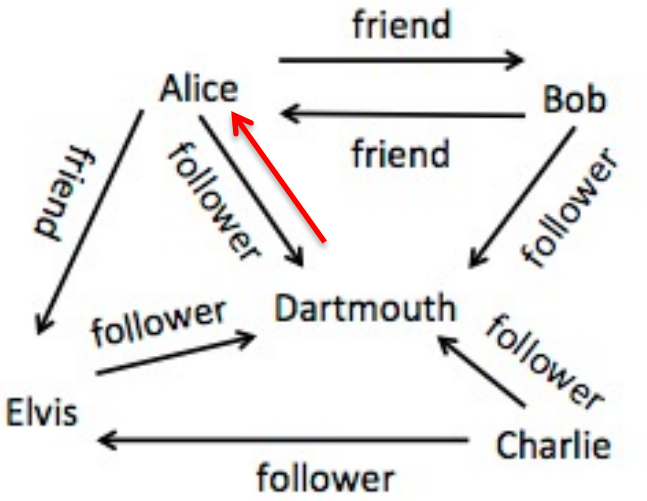
Key	Value
Alice	Null
Bob	Alice
Dartmouth	Alice
Elvis	Alice

```
Problems  Javadoc  Declaration  Console  Debug  Expressions  Error Log  Call Hierarchy
<terminated> GraphTraversal [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Feb 12, 2018, 4:31:06 PM)
Depth First Search from Alice
Run DFS or BFS on Bob before trying to find a path
Path from Alice to Dartmouth
[Alice, Elvis, Dartmouth]
Path from Alice to Charlie
        No path found
Path from Alice to Alice
[Alice]

Breadth First Search from Alice
Path from Alice to Dartmouth
[Alice, Dartmouth]
Path from Alice to Charlie
        No path found
```

GraphTraversal.java: DFS and BFS on graph we looked at last class

```
113    public static void main(String[] args) {
114        //set up graph from class introducing Graphs
115        GraphTraversal<String,String> GT = new GraphTraversal<String,String>();
116        AdjacencyMapGraph<String,String> g = new AdjacencyMapGraph<String,String>();
117        g.insertVertex("Alice");
118        g.insertVertex("Bob");
119        g.insertVertex("Charlie");
120        g.insertVertex("Dartmouth");
121        g.insertVertex("Elvis");
122        g.insertDirected("Alice", "Dartmouth", "follower");
123        g.insertDirected("Bob", "Dartmouth", "follower");
124        g.insertDirected("Charlie", "Dartmouth", "follower");
125        g.insertDirected("Elvis", "Dartmouth", "follower");
126        g.insertUndirected("Alice", "Bob", "friend"); // symmetric, undirected edge
127        g.insertDirected("Alice", "Elvis", "friend"); // not symmetric, directed edge!
128        g.insertDirected("Charlie", "Elvis", "follower");
129
130        //run DFS from Alice
131        GT.DFS(g,"Alice");
132        //find path from start to end
133        GT.findPath("Bob", "Dartmouth"); //DFS wasn't run from Bob, should reject this
134        GT.findPath("Alice","Dartmouth");
135        GT.findPath("Alice","Charlie");
136        GT.findPath("Alice","Alice");
137
138        //run BFS
139        GT.BFS(g,"Alice");
140
141        //find path from start to end
142        GT.findPath("Alice","Dartmouth");
143        GT.findPath("Alice","Charlie");
144
```

- **Run BFS start=Alice**
- **Find paths from Alice**



Key	Value
Alice	Null
Bob	Alice
Dartmouth	Alice
Elvis	Alice

```
Problems  Javadoc  Declaration  Console  Debug  Expressions  Error Log  Call Hierarchy
<terminated> GraphTraversal [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Feb 12, 2018, 4:31:06 PM)
Depth First Search from Alice
Run DFS or BFS on Bob before trying to find a path
Path from Alice to Dartmouth
[Alice, Elvis, Dartmouth]
Path from Alice to Charlie
        No path found
Path from Alice to Alice
[Alice]

Breadth First Search from Alice
Path from Alice to Dartmouth
[Alice, Dartmouth]
Path from Alice to Charlie
        No path found
```

BFS
findPath("Alice", "Dartmouth") finds shortest path
Alice->Dartmouth (DFS went through Elvis before Dartmouth)

GraphTraversal.java: From today's graph

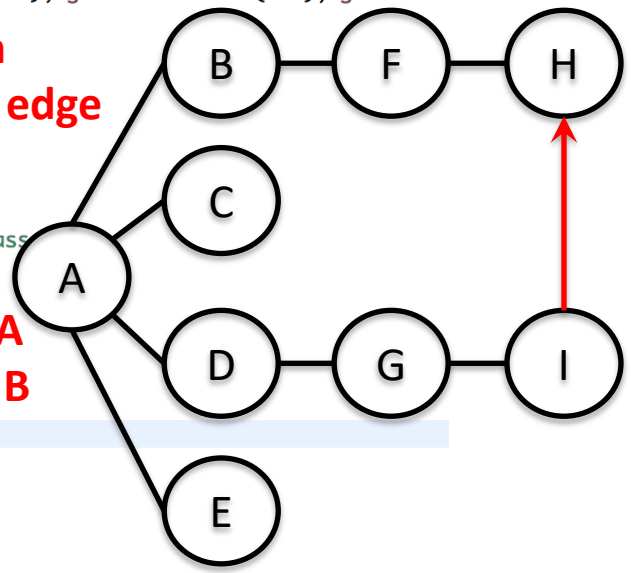
ANNOTATED SLIDES

DFS on today's graph

GraphTraversal.java

```
145         //set up graph from Graph Traversal class
146         AdjacencyMapGraph<String,String> g2 = new AdjacencyMapGraph<String,String>();
147         g2.insertVertex("A"); g2.insertVertex("B"); g2.insertVertex("C"); g2.insertVertex("D");
148         g2.insertVertex("E"); g2.insertVertex("F"); g2.insertVertex("G"); g2.insertVertex("H"); g2.insert
149         g2.insertUndirected("A", "B", "");
150         g2.insertUndirected("B", "F", "");
151         g2.insertUndirected("F", "H", "");
152         g2.insertUndirected("A", "C", "");
153         g2.insertUndirected("A", "D", "");
154         g2.insertUndirected("D", "G", "");
155         g2.insertUndirected("G", "I", "");
156         g2.insertUndirected("A", "E", "");
157         g2.insertDirected("I", "H", ""); //directed edge not from class
158
159         //run DFS from A and find path to H
160         GT.DFS(g2,"A");
161         GT.findPath("A", "B");
162
163         //run BFS from A and find path to H
164         GT.BFS(g2,"A");
165         GT.findPath("A", "B");
166
167     }
168
169 }
170
```

- Create graph
- Added extra edge
- Run DFS from A
- Find path A to B




```
Problems  Javadoc  Declaration  Console  Debug  Expressions  Error Log  Call Hierarchy
<terminated> GraphTraversal [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Feb 12, 2018, 5:14:16 PM)

Depth First Search from A
Path from A to B
[A, D, G, I, H, F, B]

Breadth First Search from A
Path from A to B
[A, B]
```

DFS

Key	Value
A	Null
B	F
C	A
D	A
E	A
F	H
G	D
H	I
I	G

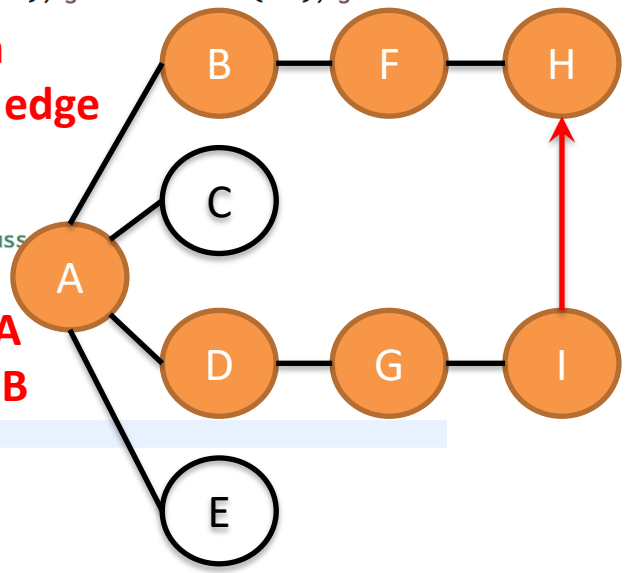
DFS on today's graph

GraphTraversal.java

```
145         //set up graph from Graph Traversal class
146         AdjacencyMapGraph<String,String> g2 = new AdjacencyMapGraph<String,String>();
147         g2.insertVertex("A"); g2.insertVertex("B"); g2.insertVertex("C"); g2.insertVertex("D");
148         g2.insertVertex("E"); g2.insertVertex("F"); g2.insertVertex("G"); g2.insertVertex("H"); g2.insert
149         g2.insertUndirected("A", "B", "");
150         g2.insertUndirected("B", "F", "");
151         g2.insertUndirected("F", "H", "");
152         g2.insertUndirected("A", "C", "");
153         g2.insertUndirected("A", "D", "");
154         g2.insertUndirected("D", "G", "");
155         g2.insertUndirected("G", "I", "");
156         g2.insertUndirected("A", "E", "");
157         g2.insertDirected("I", "H", ""); //directed edge not from class
158
159         //run DFS from A and find path to H
160         GT.DFS(g2,"A");
161         GT.findPath("A", "B");
162
163         //run BFS from A and find path to H
164         GT.BFS(g2,"A");
165         GT.findPath("A", "B");
166
167     }
168
169 }
170
```

- Create graph
- Added extra edge

- Run DFS from A
- Find path A to B



DFS

Key	Value
A	Null
B	F
C	A
D	A
E	A
F	H
G	D
H	I
I	G

Problems Javadoc Declaration Console Debug Expressions Error Log Call Hierarchy

<terminated> GraphTraversal [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Feb 12, 2018, 5:14:16 PM)

```
Depth First Search from A
Path from A to B
[A, D, G, I, H, F, B]

Breadth First Search from A
Path from A to B
[A, B]
```

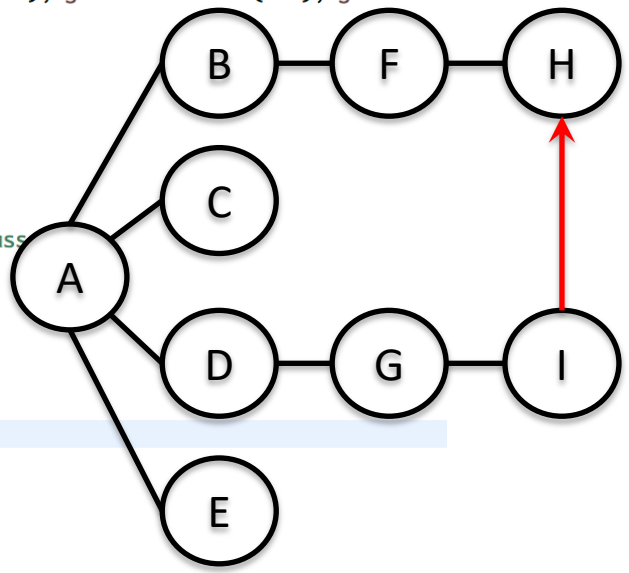
DFS findPath("A", "B") finds path, but not shortest path!
A->D->G->I->H->F->B
Why?
DFS explores as in a maze, as far as it can go before backing up
Here DFS popped D from Stack before it popped B and explored until B found

BFS on today's graph

GraphTraversal.java

```
145         //set up graph from Graph Traversal class
146         AdjacencyMapGraph<String,String> g2 = new AdjacencyMapGraph<String,String>();
147         g2.insertVertex("A"); g2.insertVertex("B"); g2.insertVertex("C"); g2.insertVertex("D");
148         g2.insertVertex("E"); g2.insertVertex("F"); g2.insertVertex("G"); g2.insertVertex("H"); g2.insert
149         g2.insertUndirected("A", "B", "");
150         g2.insertUndirected("B", "F", "");
151         g2.insertUndirected("F", "H", "");
152         g2.insertUndirected("A", "C", "");
153         g2.insertUndirected("A", "D", "");
154         g2.insertUndirected("D", "G", "");
155         g2.insertUndirected("G", "I", "");
156         g2.insertUndirected("A", "E", "");
157         g2.insertDirected("I", "H", ""); //directed edge not from class
158
159         //run DFS from A and find path to H
160         GT.DFS(g2,"A");
161         GT.findPath("A", "B");
162
163         //run BFS from A and find path to H
164         GT.BFS(g2,"A");
165         GT.findPath("A", "B");
166
167     }
168
169 }
170
```

- **Run BFS**
- **Find path A to B**



BFS

Key	Value
A	Null
B	**A**
C	A
D	A
E	A
F	B
G	D
H	F
I	G

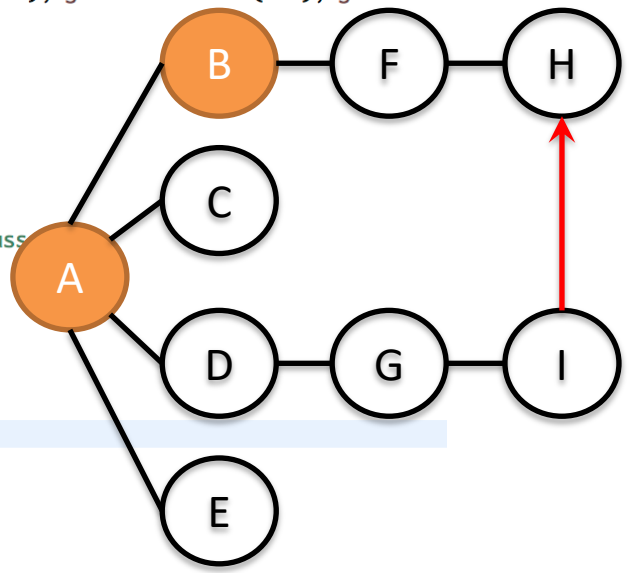
```
Problems  Javadoc  Declaration  Console  Debug  Expressions  Error Log  Call Hierarchy
<terminated> GraphTraversal [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Feb 12, 2018, 5:14:16 PM)

Depth First Search from A
Path from A to B
[A, D, G, I, H, F, B]

Breadth First Search from A
Path from A to B
[A, B]
```

BFS on today's graph

GraphTraversal.java

```
145         //set up graph from Graph Traversal class
146         AdjacencyMapGraph<String,String> g2 = new AdjacencyMapGraph<String,String>();
147         g2.insertVertex("A"); g2.insertVertex("B"); g2.insertVertex("C"); g2.insertVertex("D");
148         g2.insertVertex("E"); g2.insertVertex("F"); g2.insertVertex("G"); g2.insertVertex("H"); g2.insert
149         g2.insertUndirected("A", "B", "");
150         g2.insertUndirected("B", "F", "");
151         g2.insertUndirected("F", "H", "");
152         g2.insertUndirected("A", "C", "");
153         g2.insertUndirected("A", "D", "");
154         g2.insertUndirected("D", "G", "");
155         g2.insertUndirected("G", "I", "");
156         g2.insertUndirected("A", "E", "");
157         g2.insertDirected("I", "H", ""); //directed edge not from class
158
159         //run DFS from A and find path to H
160         GT.DFS(g2,"A");
161         GT.findPath("A", "B");
162
163         //run BFS from A and find path to H
164         GT.BFS(g2,"A");
165         GT.findPath("A", "B");
166
167     }
168
169 }
170
```

- **Run BFS**
- **Find path A to B**



BFS

Key	Value
A	Null
B	**A**
C	A
D	A
E	A
F	B
G	D
H	F
I	G

Problems Javadoc Declaration Console Debug Expressions Error Log Call Hierarchy

<terminated> GraphTraversal [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_112.jdk/Contents/Home/bin/java (Feb 12, 2018, 5:14:16 PM)

```
Depth First Search from A
Path from A to B
[A, D, G, I, H, F, B]

Breadth First Search from A
Path from A to B
[A, B]
```

BFS findPath("A", "B") finds shortest path A->B

Why?

BFS explores outward in ripples