CS 10: Problem solving via Object Oriented Programming

Winter 2017

Tim Pierson
260 (255) Sudikoff

Day 16 – Graph Traversals

Agenda

1. Depth first search

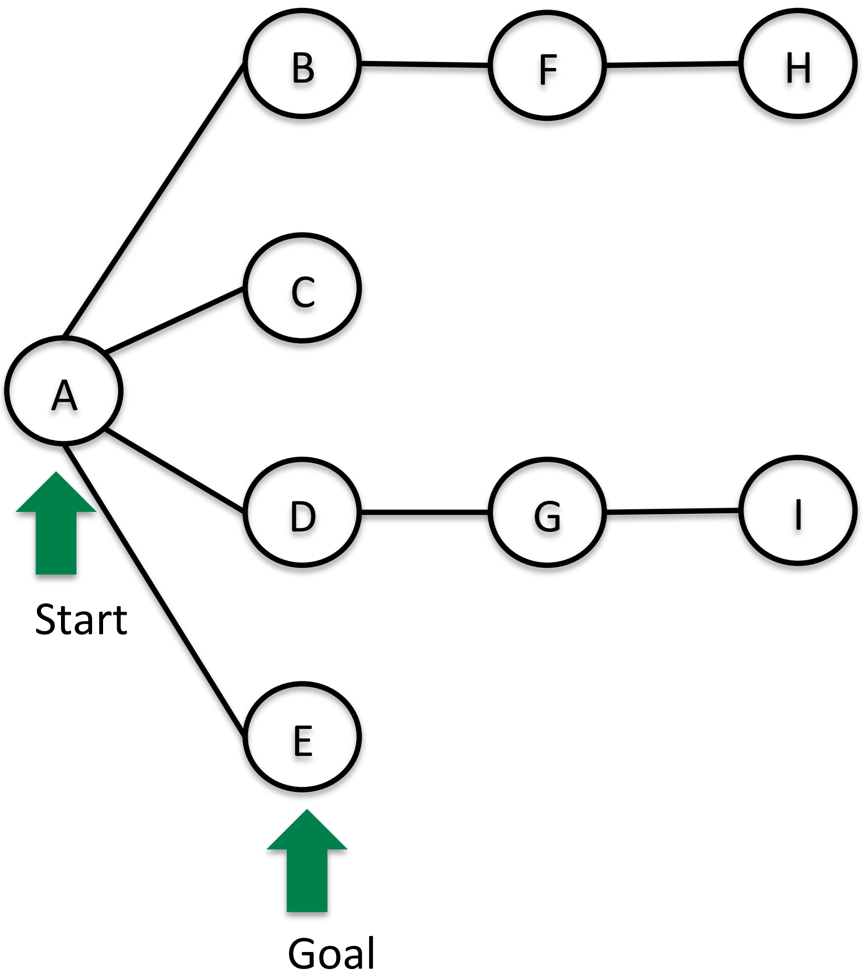
2. Breadth first search

Graph traversals are useful to answer questions about vertex relationships

Some Graph traversals uses

- Uses are typically 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 cycles (next class will consider them)

Depth First Search (DFS) uses a stack to explore as if in a maze



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

```
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

```
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

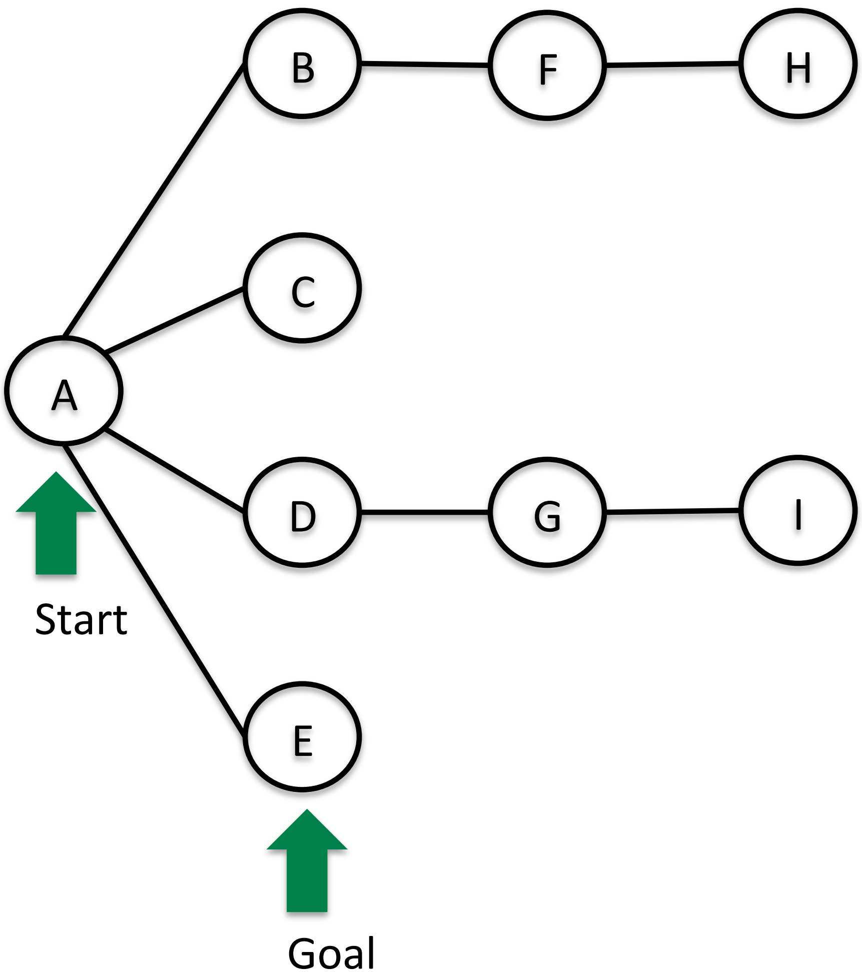
RegionFinder

```
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 pixel from end of list, you implemented a stack

If you added to end of list…

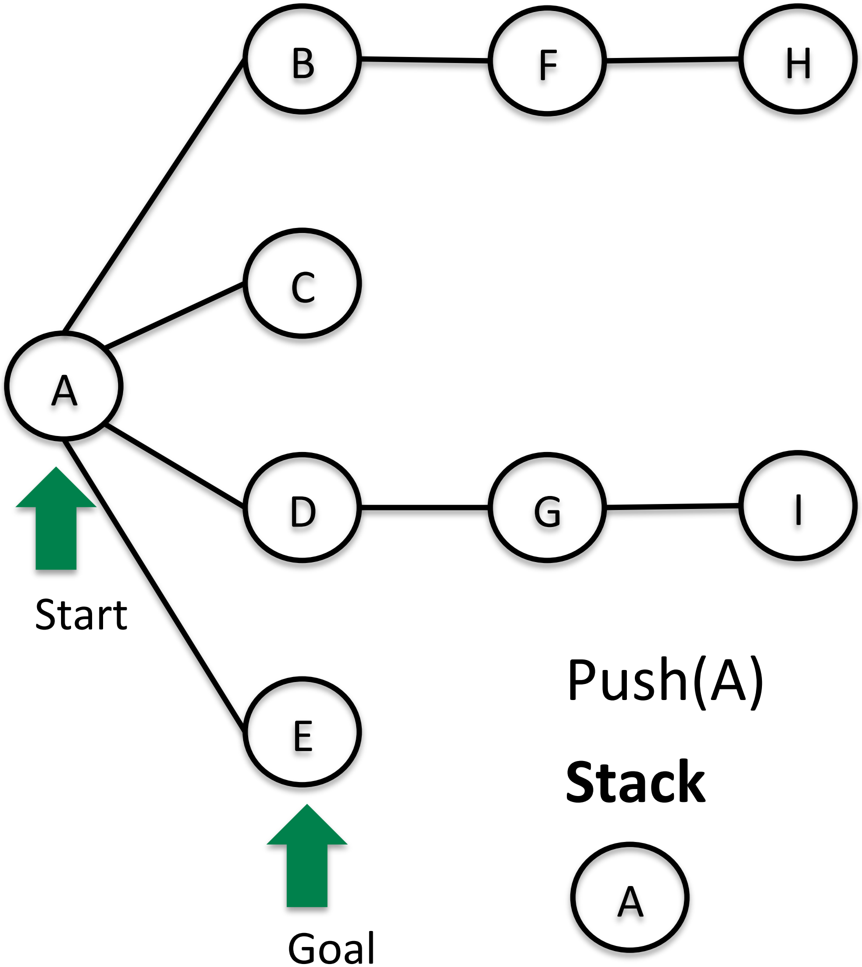
Depth First Search (DFS) is like exploring a maze



DFS algorithm

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

Depth First Search (DFS) is like exploring a maze



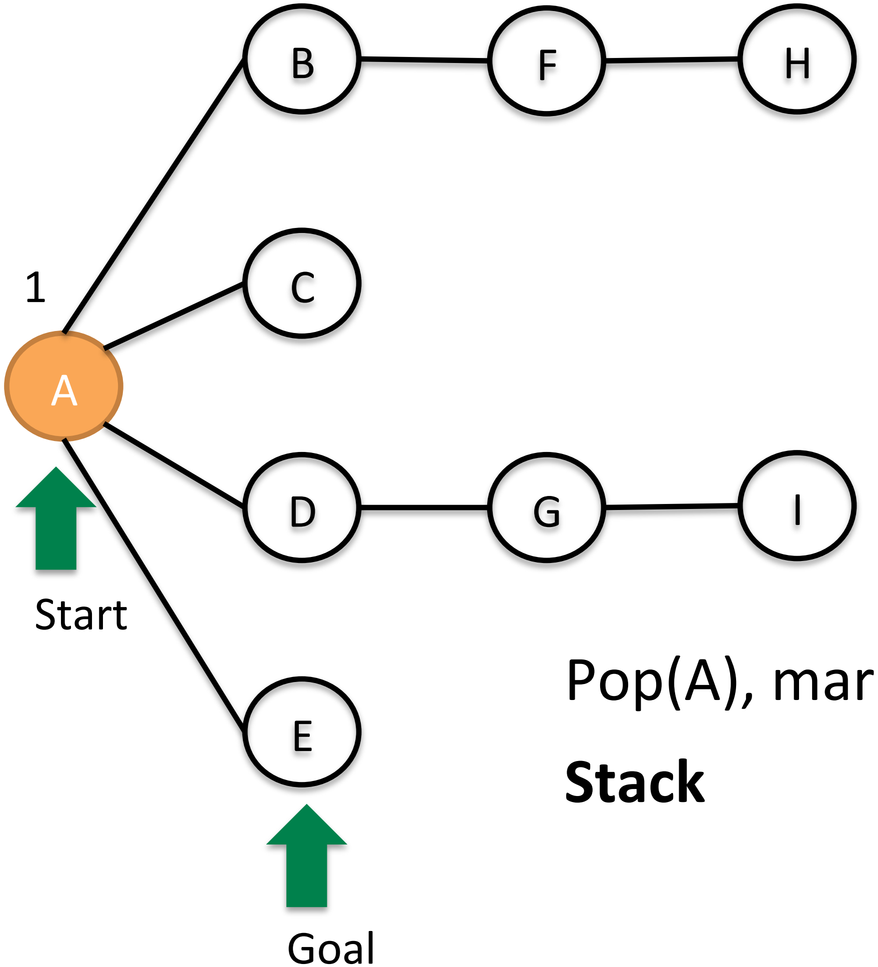
DFS algorithm

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

Push(A)

Stack

Depth First Search (DFS) is like exploring a maze



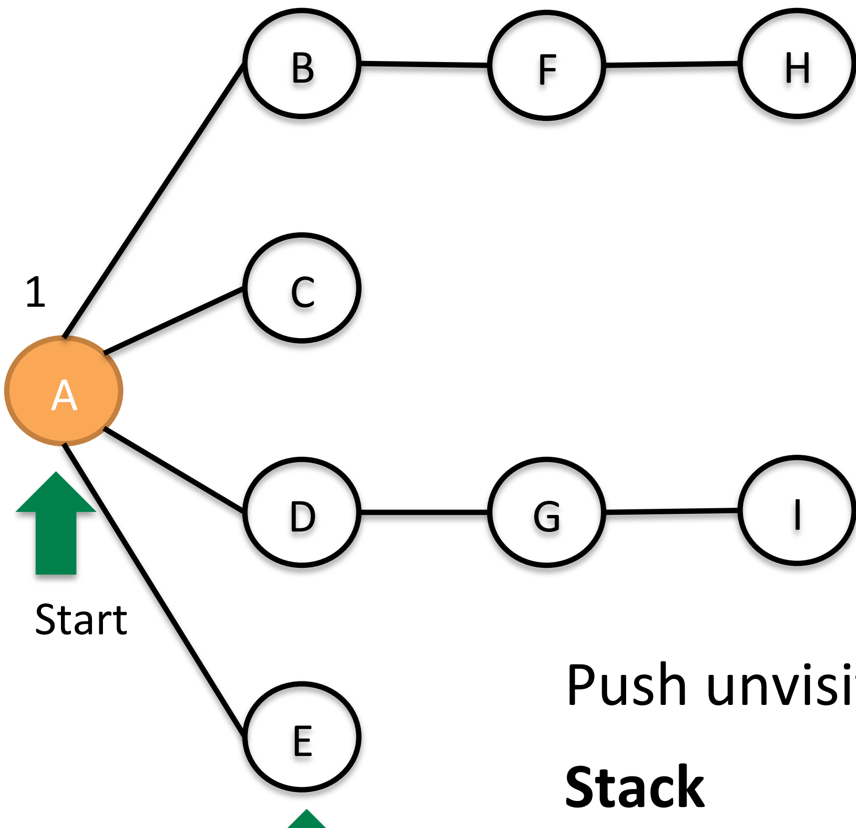
DFS algorithm

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

Pop(A), mark visited

Stack

Depth First Search (DFS) is like exploring a maze

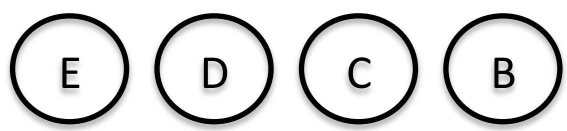


DFS algorithm

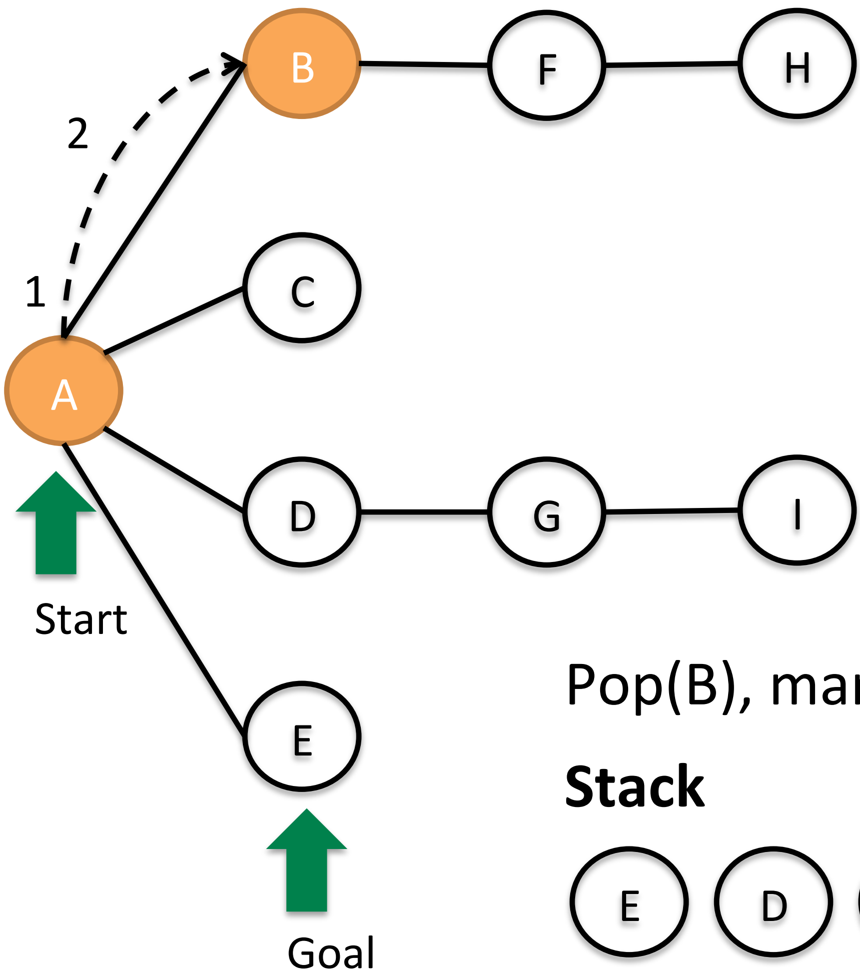
```
stack.push(s) //start node
repeat until find goal vertex or
stack empty:
    u = stack.pop()
    if !u.visited
        u.visited = true
        (maybe do something while here)
        for v ∈ u.adjacent
              if !v.visited
                stack.push(v)
```

Push unvisited adjacent

Stack



Depth First Search (DFS) is like exploring a maze



DFS algorithm

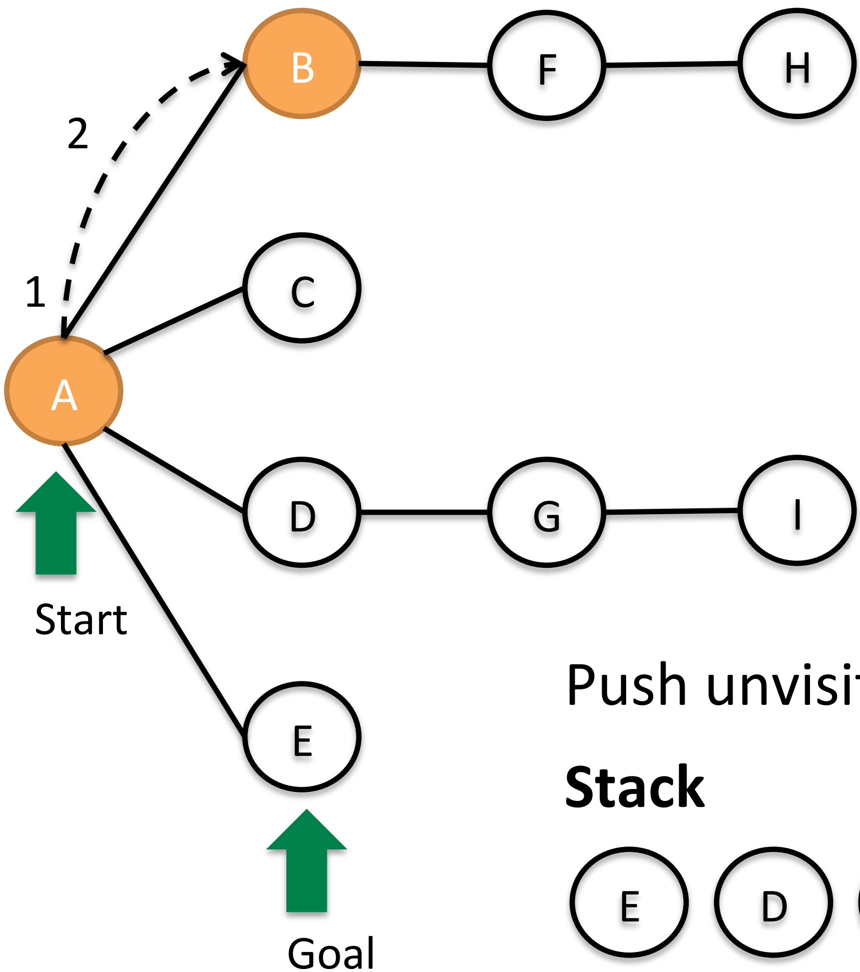
```
stack.push(s) //start node
repeat until find goal vertex or
stack empty:
    u = stack.pop()
    if !u.visited
        u.visited = true
        (maybe do something while here)
        for v ∈ u.adjacent
              if !v.visited
                stack.push(v)
```

Pop(B), mark visited

Stack

E D C

Depth First Search (DFS) is like exploring a maze



DFS algorithm

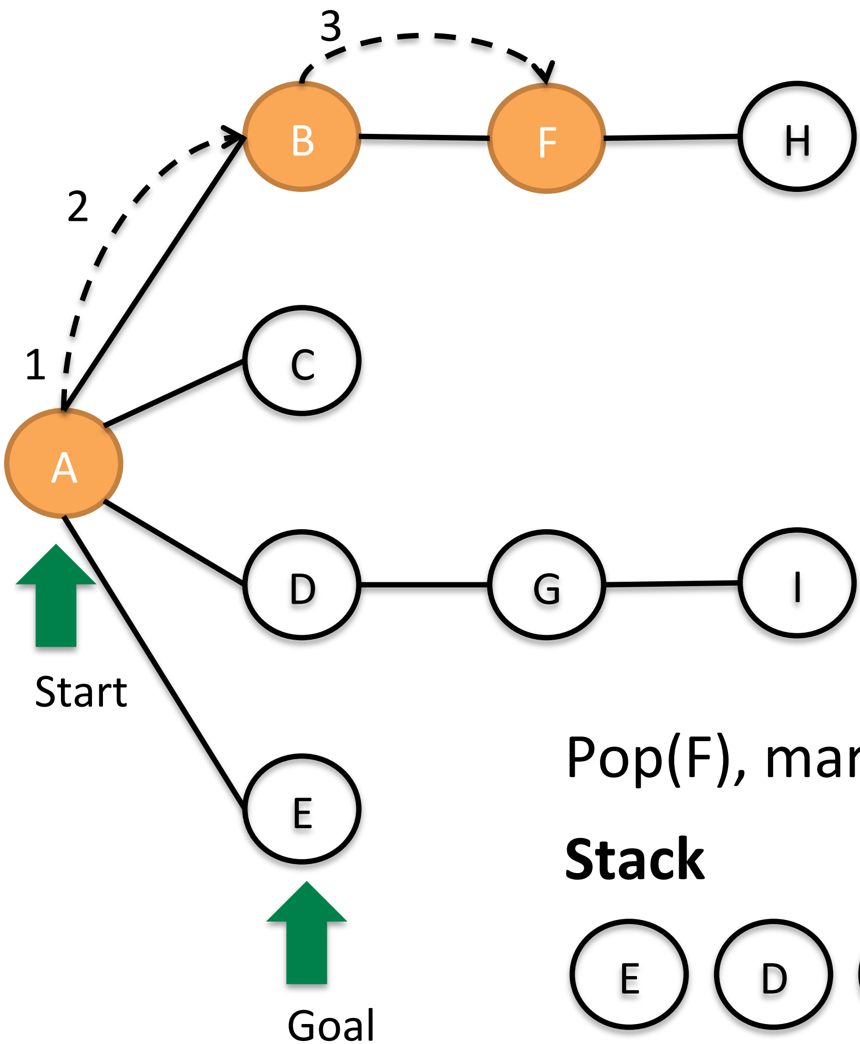
```
stack.push(s) //start node
repeat until find goal vertex or
stack empty:
    u = stack.pop()
    if !u.visited
        u.visited = true
        (maybe do something while here)
        for v ∈ u.adjacent
              if !v.visited
                stack.push(v)
```

Push unvisited adjacent (F, but not A)

Stack

E D C F

Depth First Search (DFS) is like exploring a maze



DFS algorithm

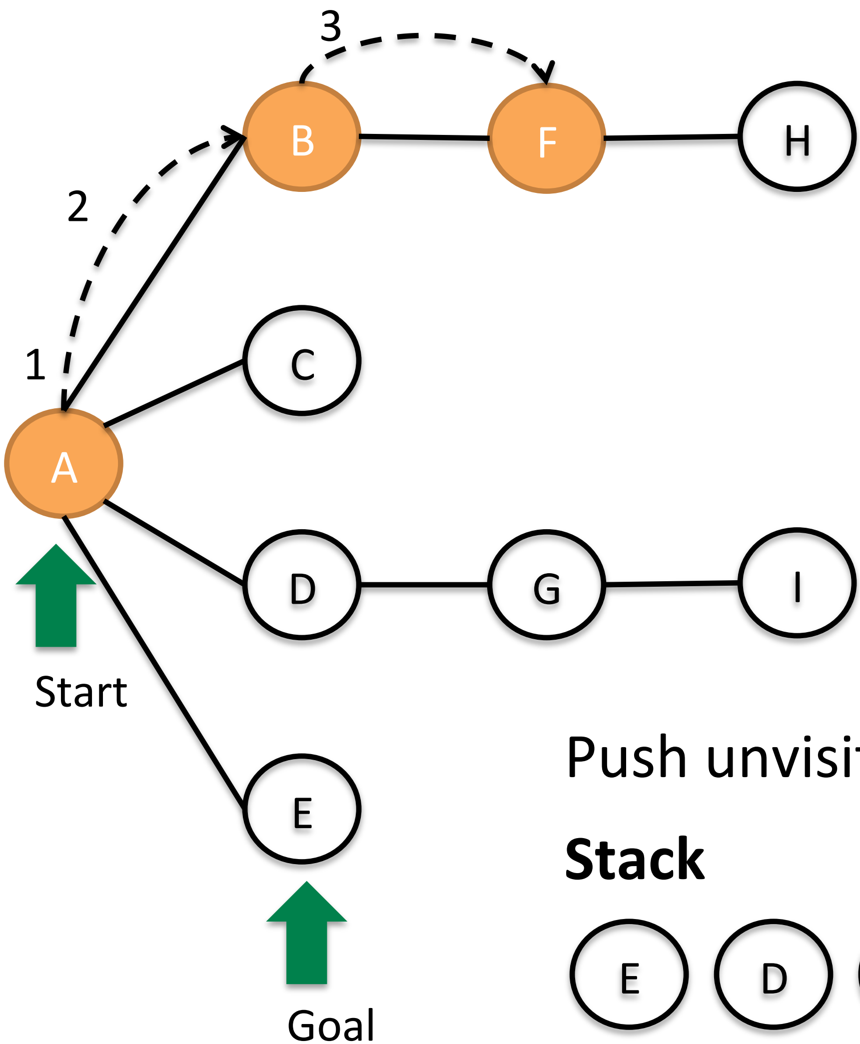
```
stack.push(s) //start node
repeat until find goal vertex or
stack empty:
    u = stack.pop()
    if !u.visited
        u.visited = true
        (maybe do something while here)
        for v ∈ u.adjacent
              if !v.visited
                stack.push(v)
```

Pop(F), mark visited

Stack

E D C

Depth First Search (DFS) is like exploring a maze



DFS algorithm

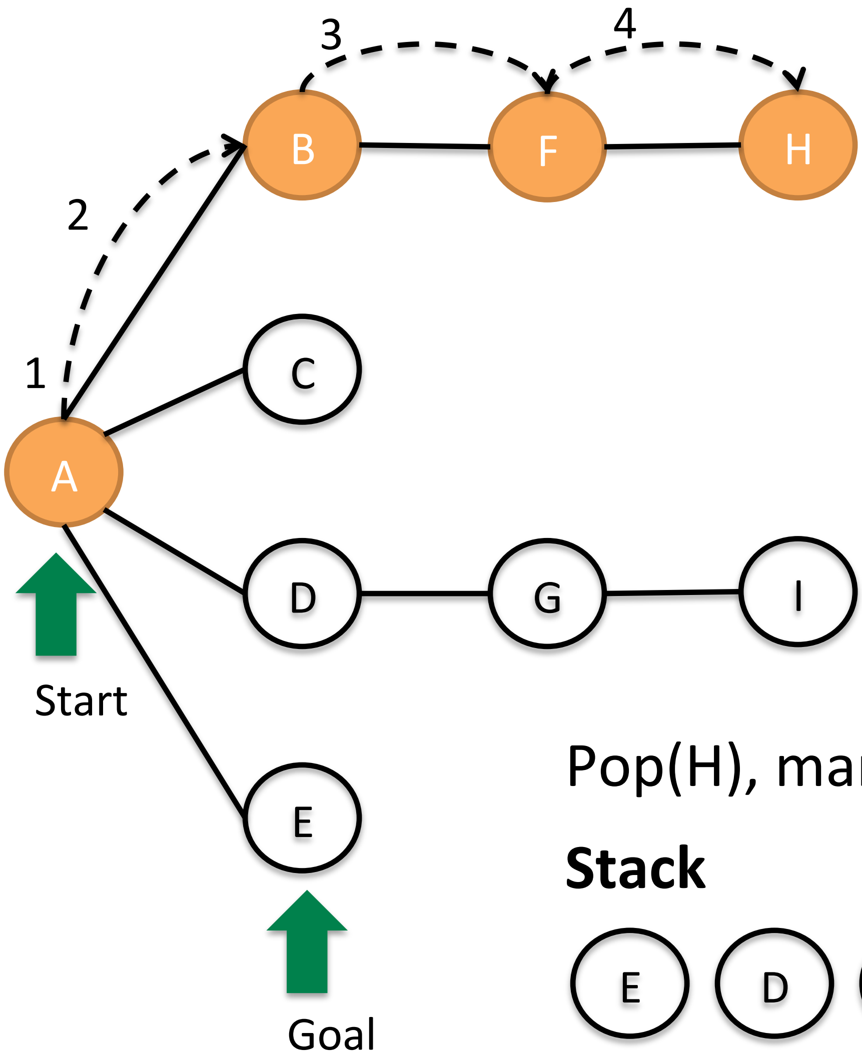
```
stack.push(s) //start node
repeat until find goal vertex or
stack empty:
    u = stack.pop()
    if !u.visited
        u.visited = true
        (maybe do something while here)
        for v ∈ u.adjacent
              if !v.visited
                stack.push(v)
```

Push unvisited adjacent (H, but not B)

Stack

E D C H

Depth First Search (DFS) is like exploring a maze



DFS algorithm

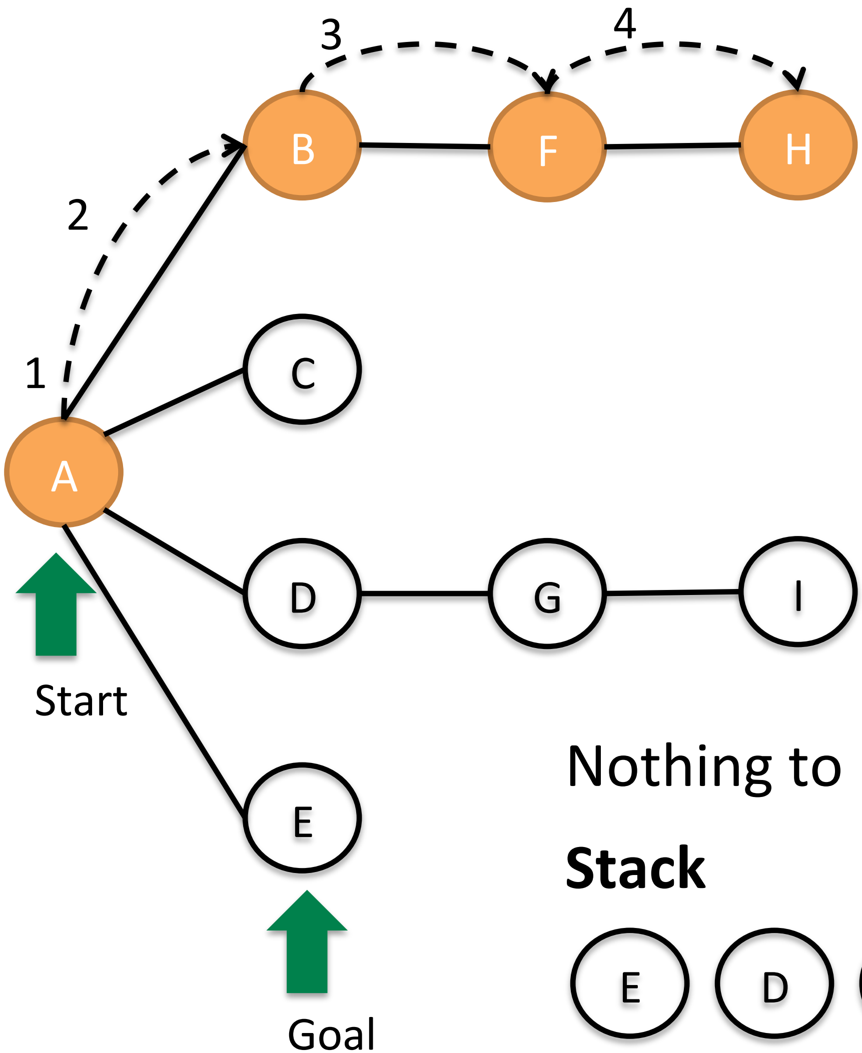
```
stack.push(s) //start node
repeat until find goal vertex or
stack empty:
    u = stack.pop()
    if !u.visited
        u.visited = true
        (maybe do something while here)
        for v ∈ u.adjacent
              if !v.visited
                stack.push(v)
```

Pop(H), mark visited

Stack

E D C

Depth First Search (DFS) is like exploring a maze



DFS algorithm

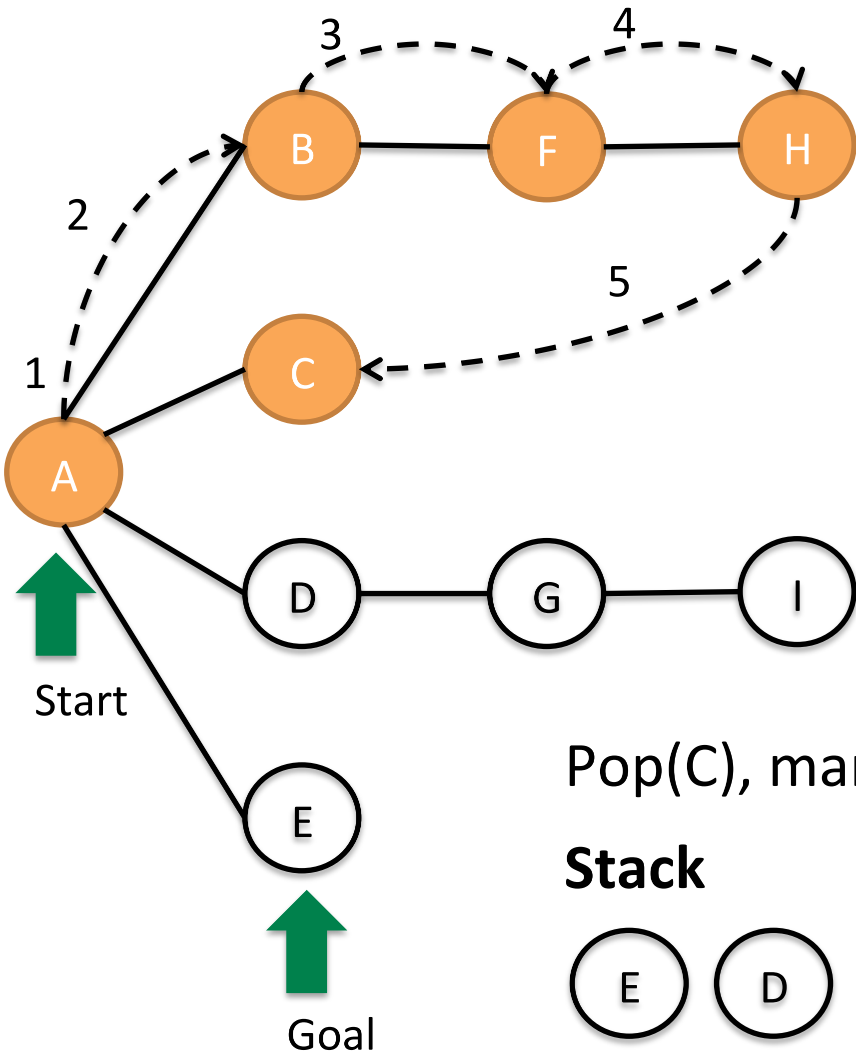
```
stack.push(s) //start node
repeat until find goal vertex or
stack empty:
    u = stack.pop()
    if !u.visited
        u.visited = true
        (maybe do something while here)
        for v ∈ u.adjacent
              if !v.visited
               stack.push(v)
```

Nothing to push, back up by popping C

Stack

E D C

Depth First Search (DFS) is like exploring a maze



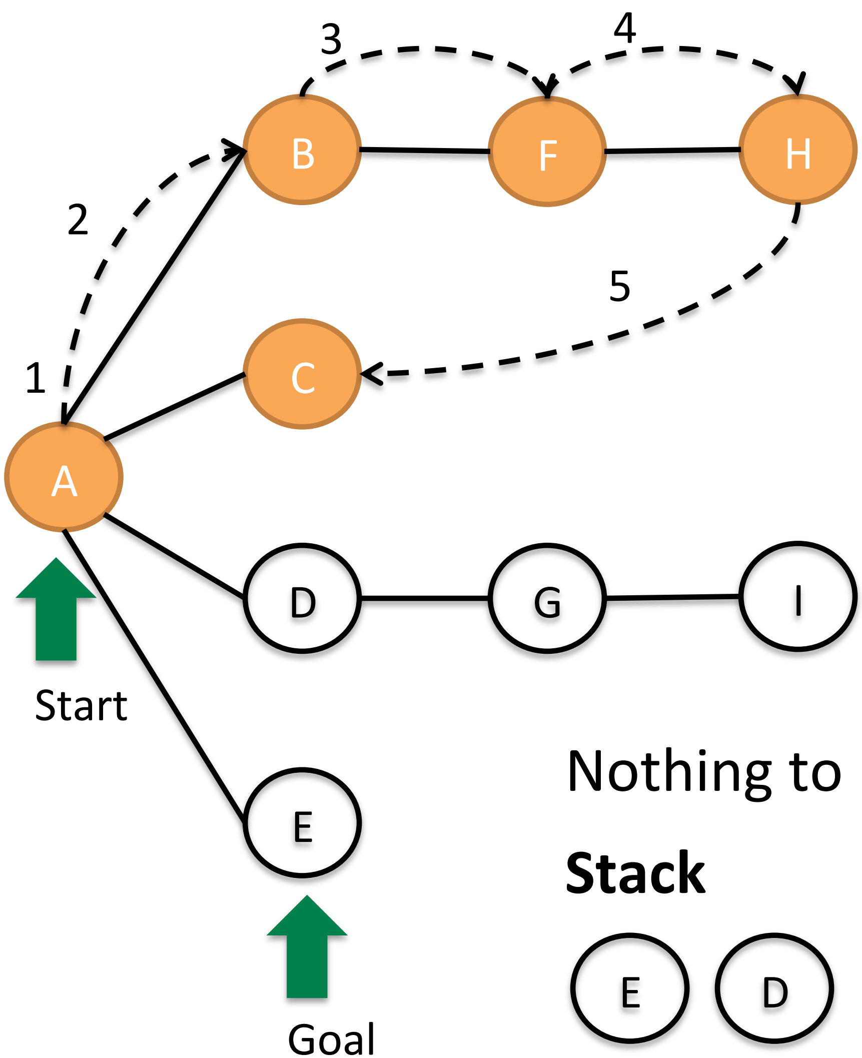
DFS algorithm

```
stack.push(s) //start node
repeat until find goal vertex or
stack empty:
    u = stack.pop()
    if !u.visited
        u.visited = true
        (maybe 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



DFS algorithm

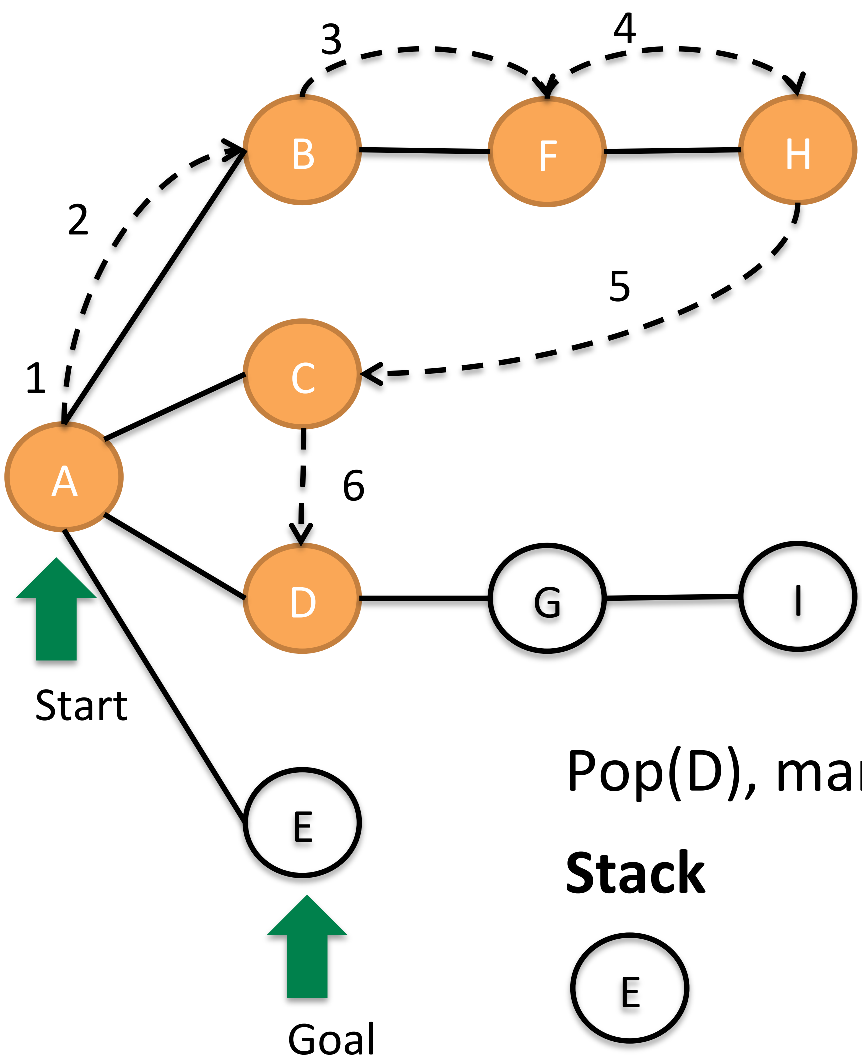
```
stack.push(s) //start node
repeat until find goal vertex or
stack empty:
    u = stack.pop()
    if !u.visited
        u.visited = true
        (maybe do something while here)
        for v ∈ u.adjacent
              if !v.visited
               stack.push(v)
```

Nothing to push, back up by popping D

Stack

E D

Depth First Search (DFS) is like exploring a maze



DFS algorithm

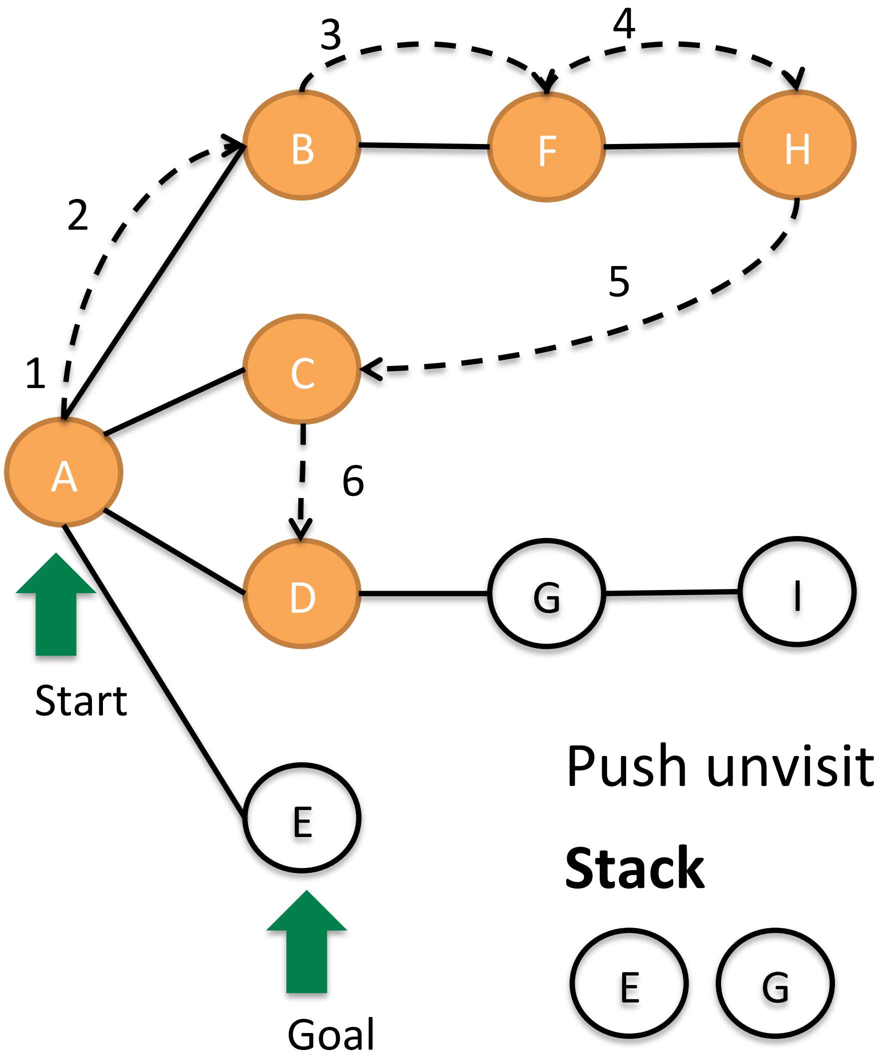
```
stack.push(s) //start node
repeat until find goal vertex or
stack empty:
    u = stack.pop()
    if !u.visited
        u.visited = true
        (maybe do something while here)
        for v ∈ u.adjacent
              if !v.visited
                stack.push(v)
```

Pop(D), mark visited

Stack

E

Depth First Search (DFS) is like exploring a maze



DFS algorithm

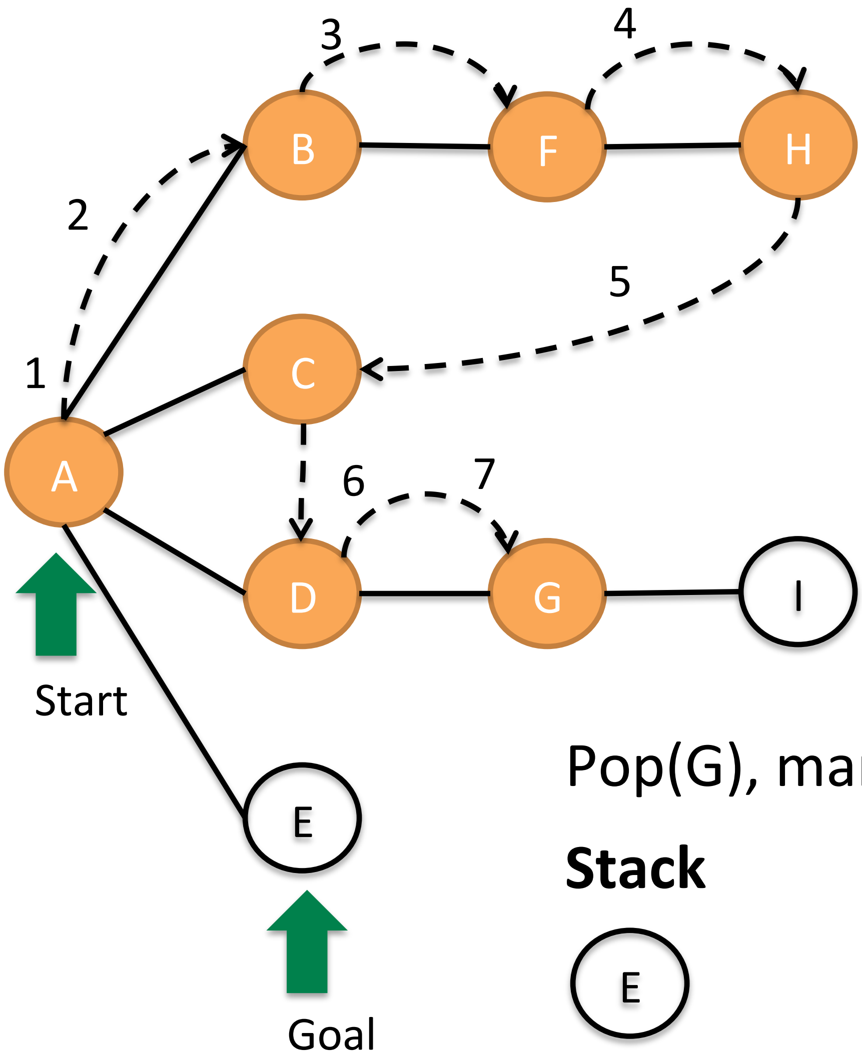
```
stack.push(s) //start node
repeat until find goal vertex or
stack empty:
    u = stack.pop()
    if !u.visited
        u.visited = true
        (maybe do something while here)
        for v ∈ u.adjacent
              if !v.visited
                stack.push(v)
```

Push unvisited adjacent (G, but not A)

Stack

E G

Depth First Search (DFS) is like exploring a maze



DFS algorithm

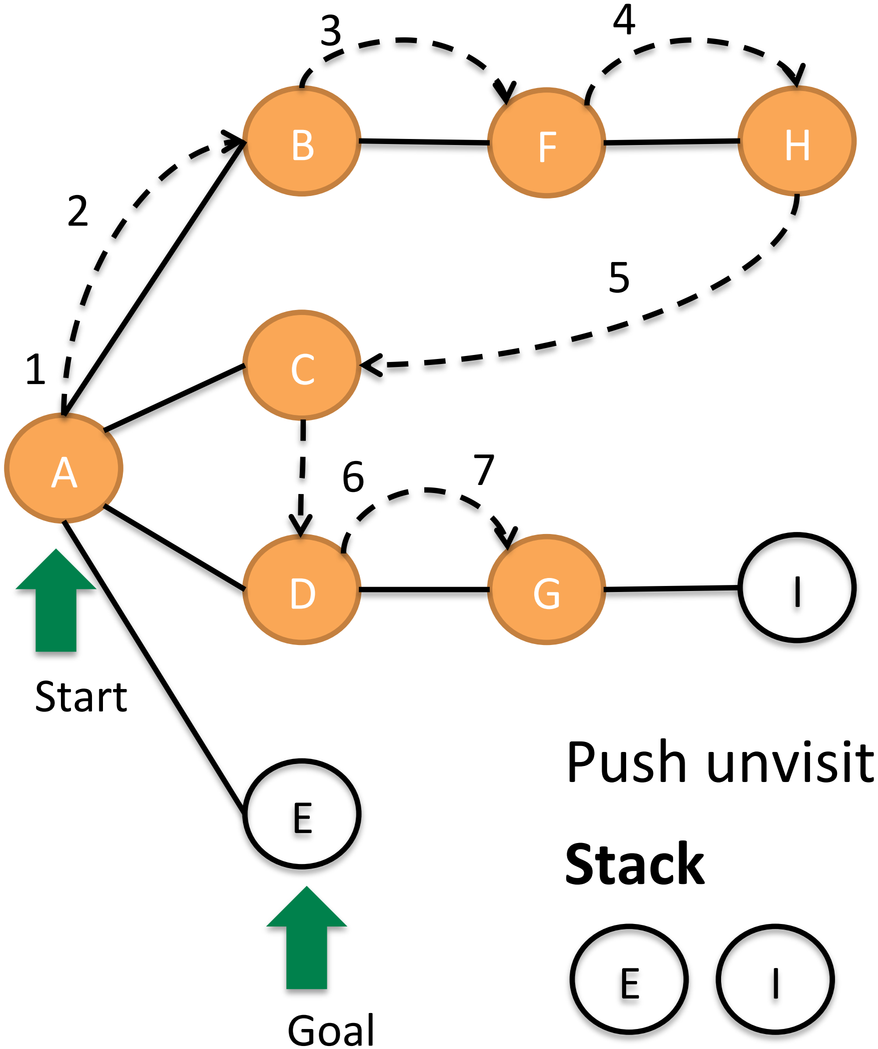
```
stack.push(s) //start node
repeat until find goal vertex or
stack empty:
    u = stack.pop()
    if !u.visited
        u.visited = true
        (maybe do something while here)
        for v ∈ u.adjacent
              if !v.visited
                stack.push(v)
```

Pop(G), mark visited

Stack

E

Depth First Search (DFS) is like exploring a maze



DFS algorithm

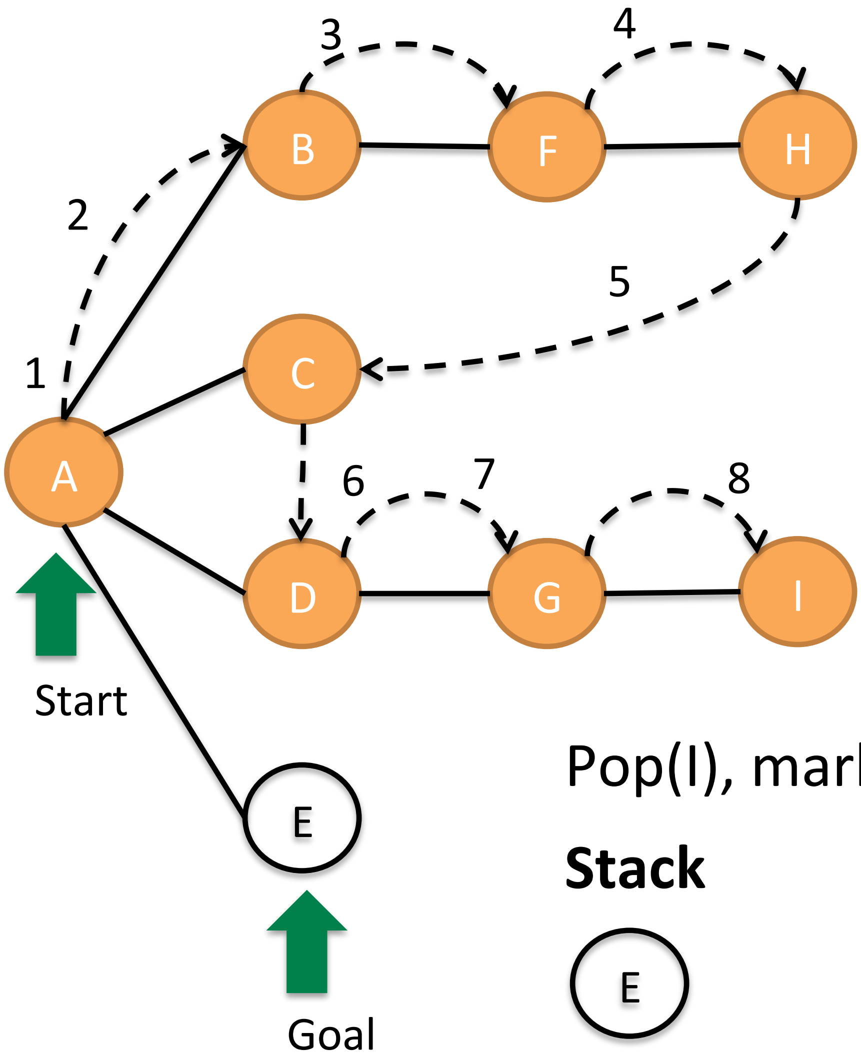
```
stack.push(s) //start node
repeat until find goal vertex or
stack empty:
    u = stack.pop()
    if !u.visited
        u.visited = true
        (maybe do something while here)
        for v ∈ u.adjacent
              if !v.visited
               stack.push(v)
```

Push unvisited adjacent (I, but not D)

Stack

E I

Depth First Search (DFS) is like exploring a maze



DFS algorithm

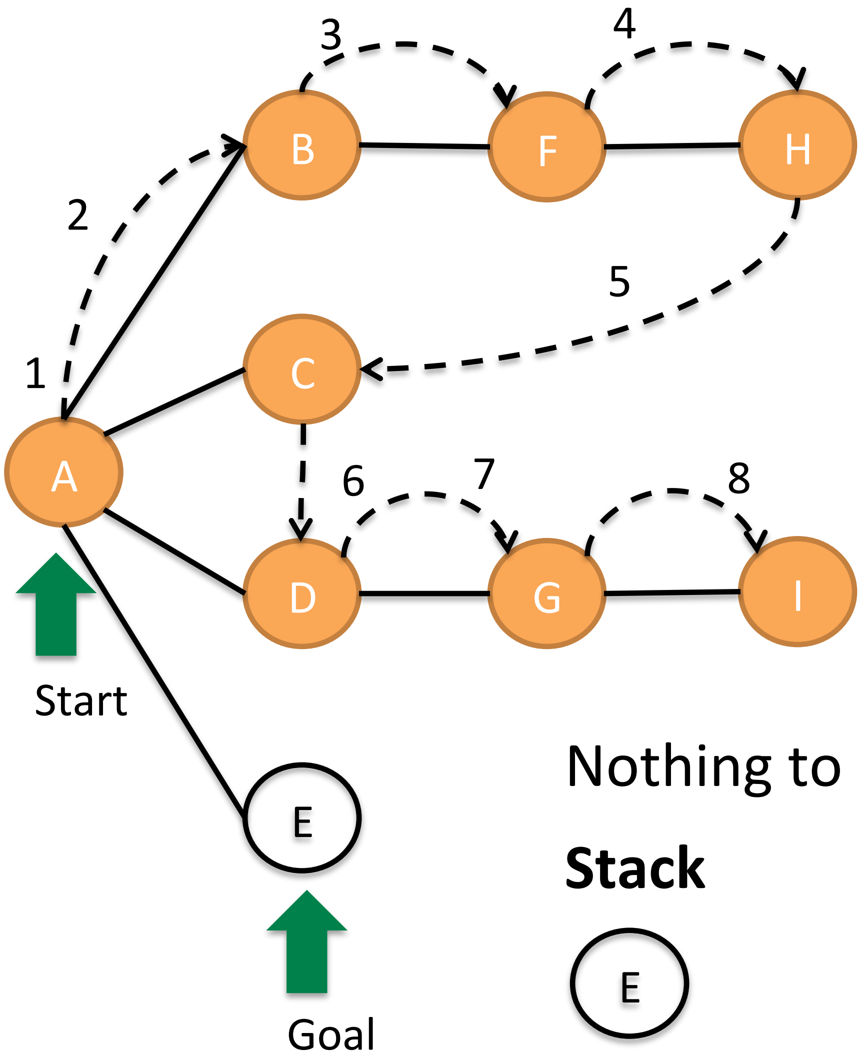
```
stack.push(s) //start node
repeat until find goal vertex or
stack empty:
    u = stack.pop()
    if !u.visited
        u.visited = true
        (maybe do something while here)
        for v ∈ u.adjacent
              if !v.visited
               stack.push(v)
```

Pop(I), mark visited

Stack

E

Depth First Search (DFS) is like exploring a maze



DFS algorithm

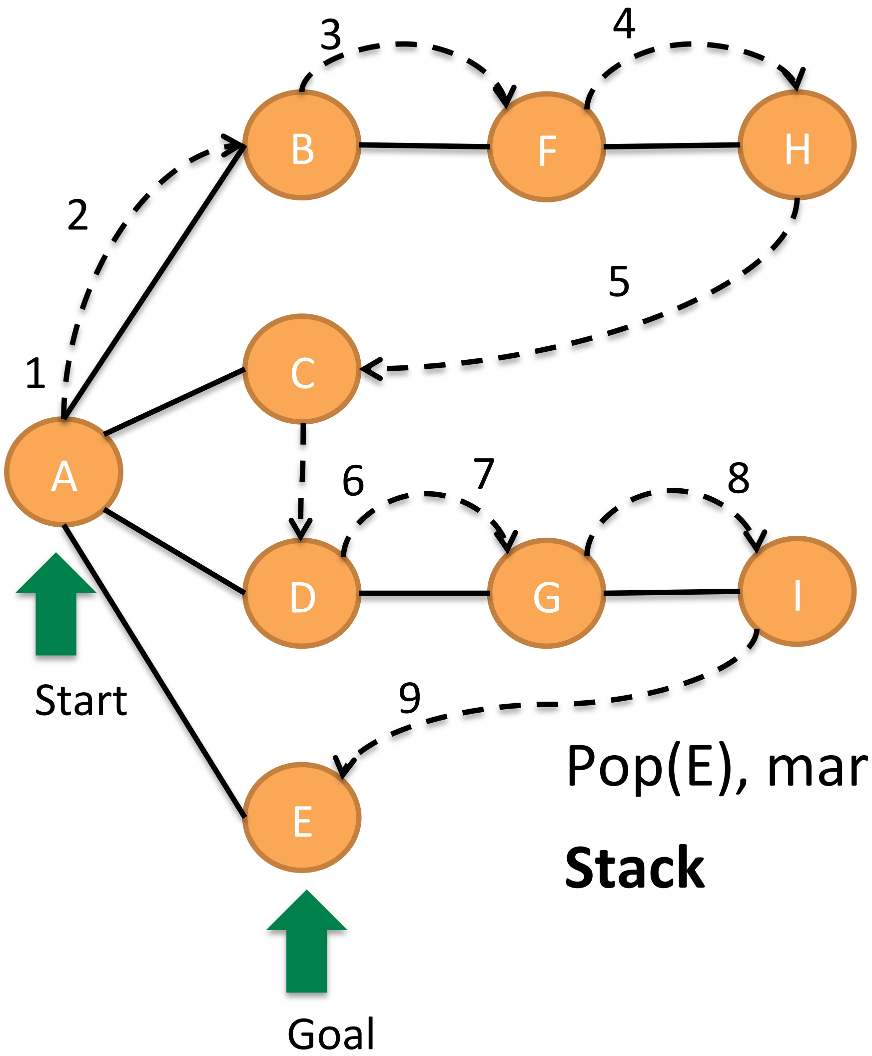
```
stack.push(s) //start node
repeat until find goal vertex or
stack empty:
    u = stack.pop()
    if !u.visited
        u.visited = true
        (maybe do something while here)
        for v ∈ u.adjacent
            if !v.visited
              stack.push(v)
```

Nothing to push, back up by popping E

Stack

E

Depth First Search (DFS) is like exploring a maze



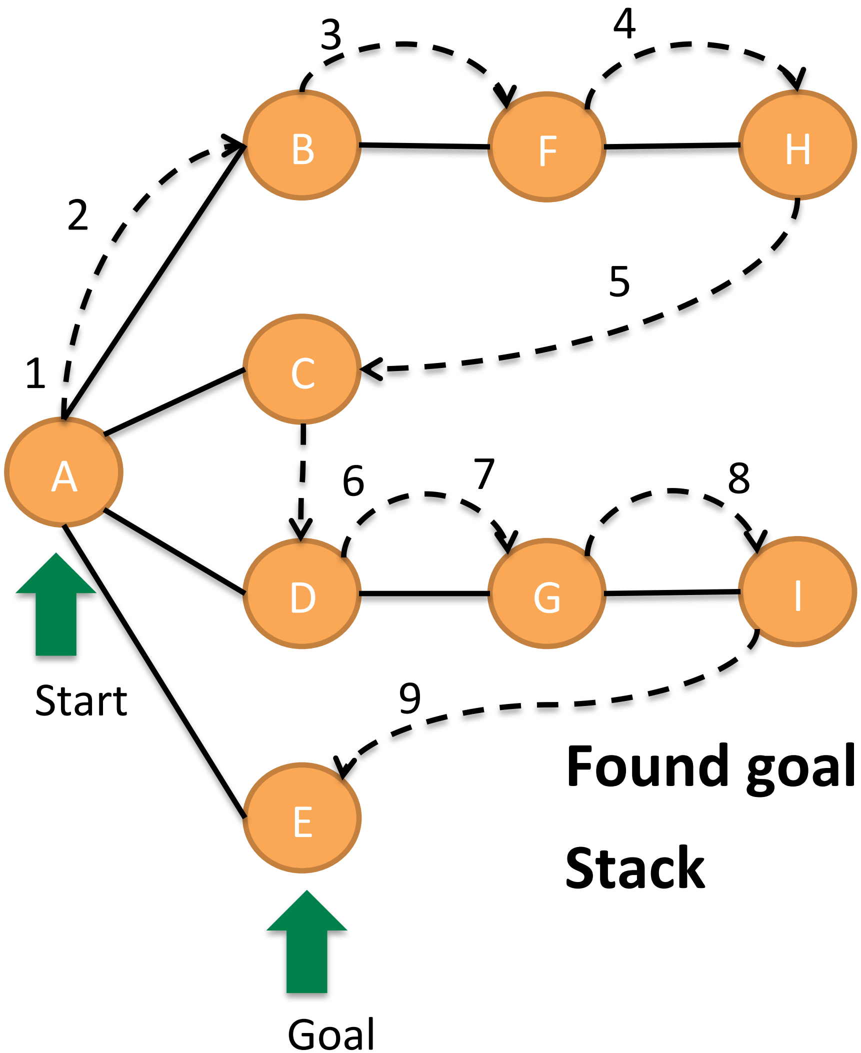
Pop(E), mark visited

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
        (maybe do something while here)
        for v ∈ u.adjacent
              if !v.visited
                stack.push(v)
```

Depth First Search (DFS) is like exploring a maze



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
        (maybe do something while here)
        for v ∈ u.adjacent
            if !v.visited
              stack.push(v)
```

Node discovery tells us something about the graph

Discovery edges

- Edges that lead to unvisited nodes
- Discovery edges form a tree on the graph
- 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 nodes)
- **Not guaranteed to be shortest path!**

Back edges

- Edges that lead to previously discovered nodes
- Lead to ancestor nodes in tree
- Indicate presence of a cycle in the graph

Run time is O(n+m)

Run time

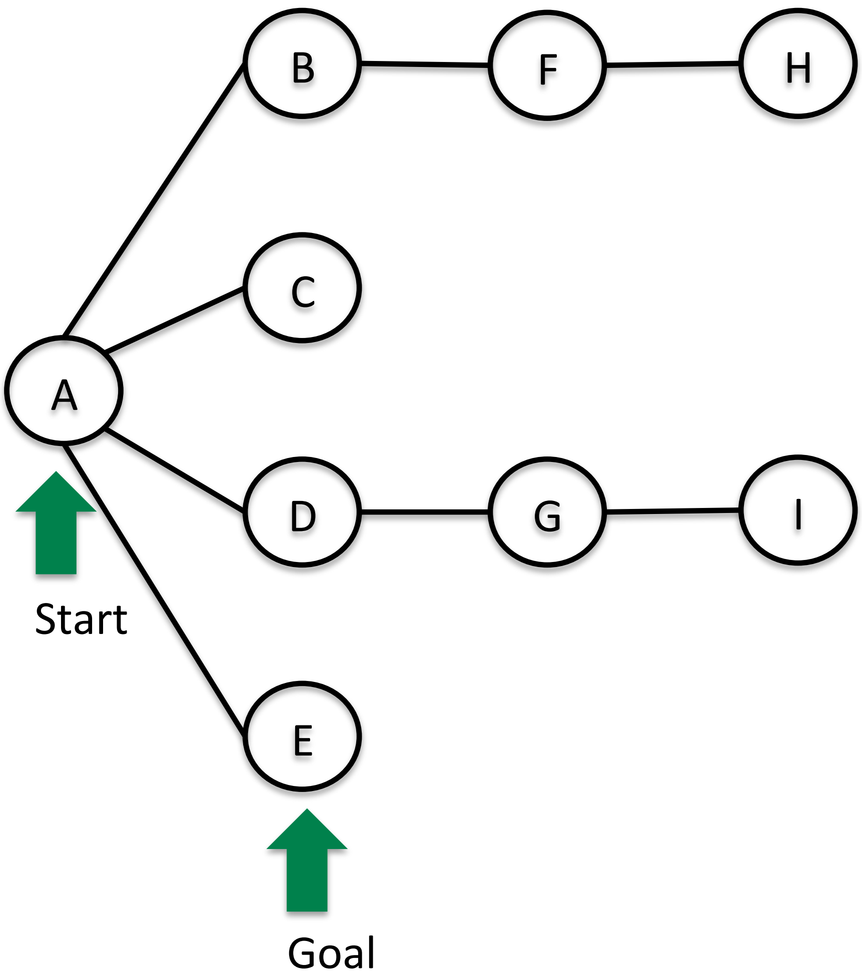
- Assume graph with *n* nodes and *m* edges
- Visit each node at most one time (due to visited indicator)
- Visit each edge at most one time

- Run time is O(n+m)

Agenda

1. Depth first search
2. Breadth first search

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 found 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

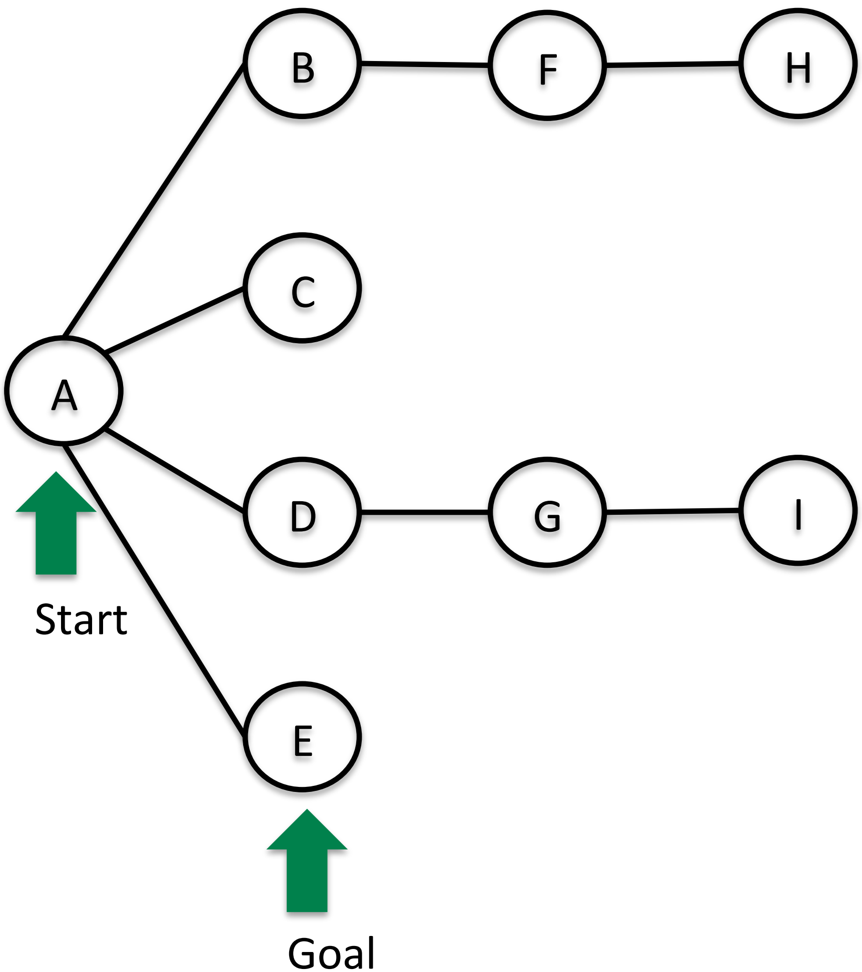
RegionFinder

```
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 pixel from front of list, you implemented a queue

If you added to end of list…

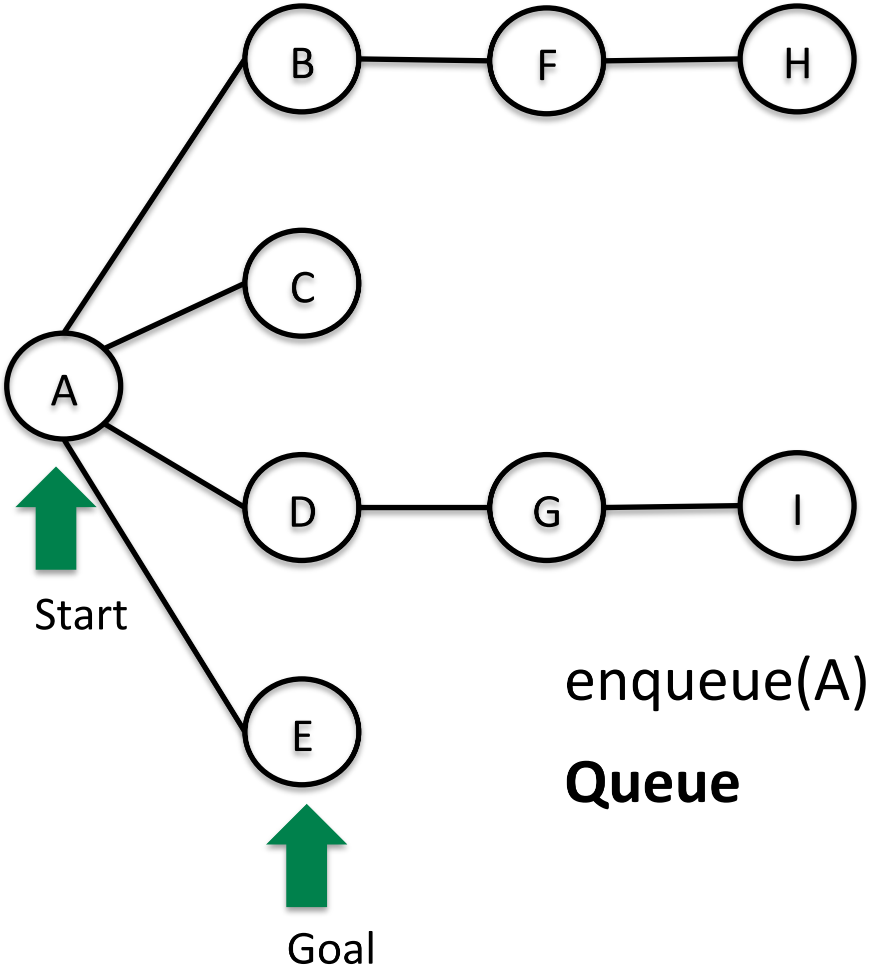
Breadth First Search (BFS) can find the shortest path between nodes



BFS algorithm

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

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



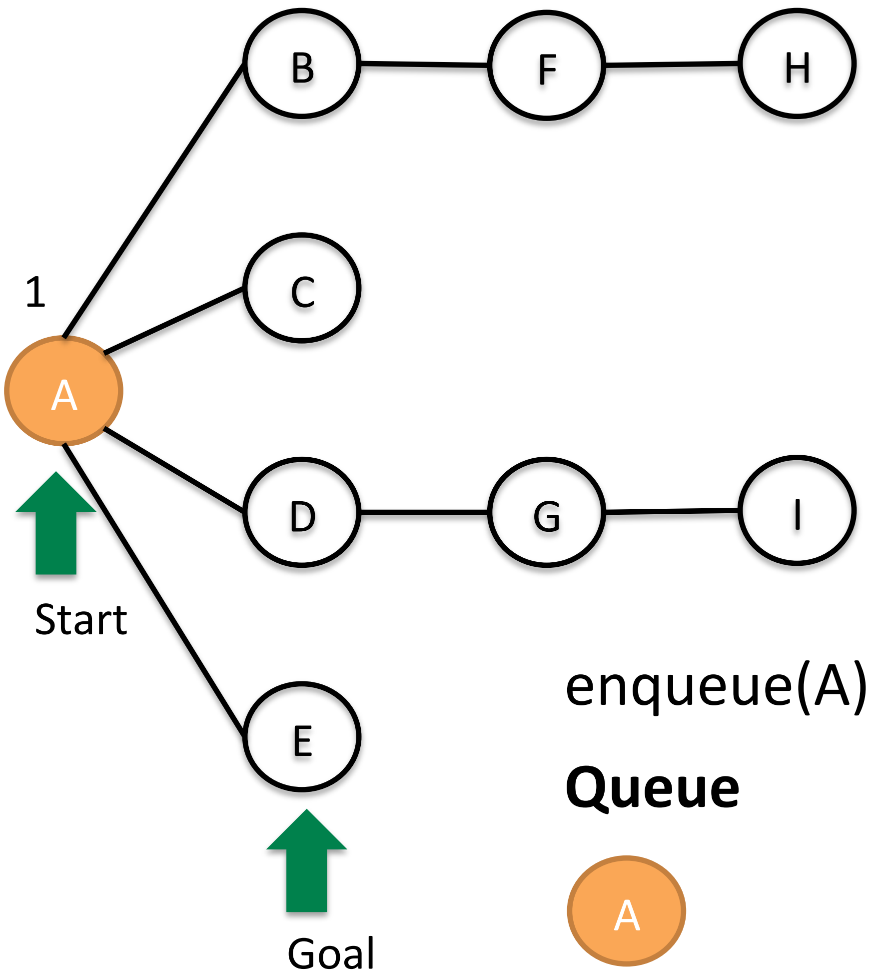
BFS algorithm

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

enqueue(A)

Queue

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



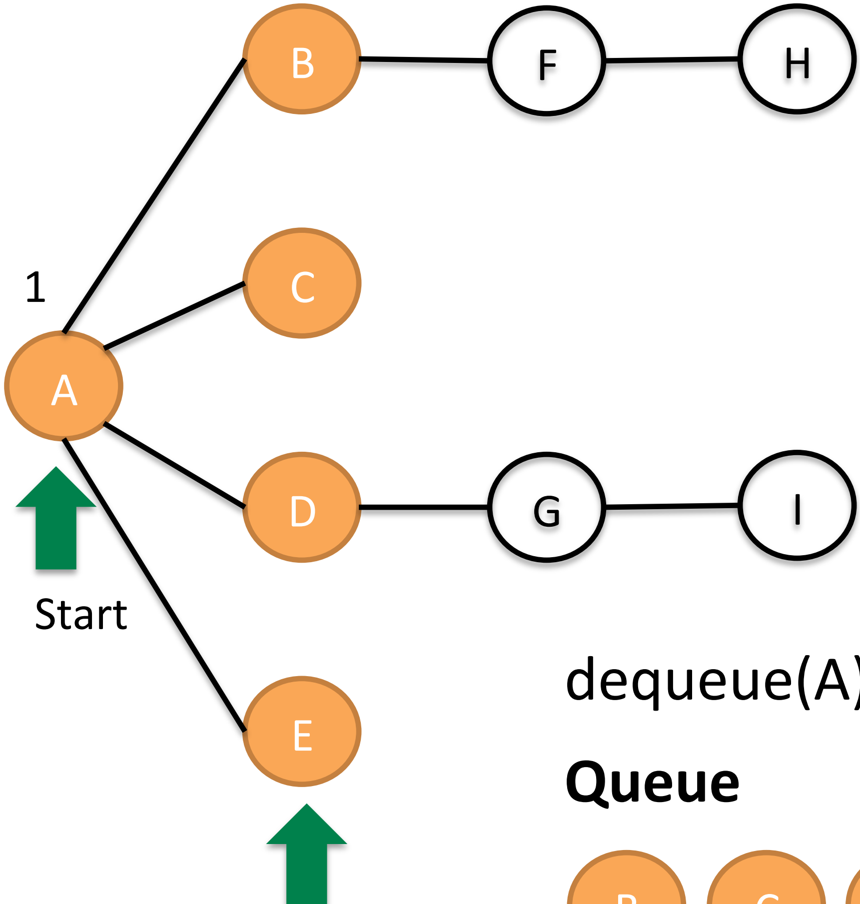
BFS algorithm

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

enqueue(A)

Queue

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

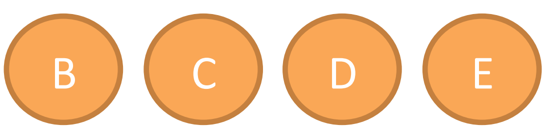


BFS algorithm

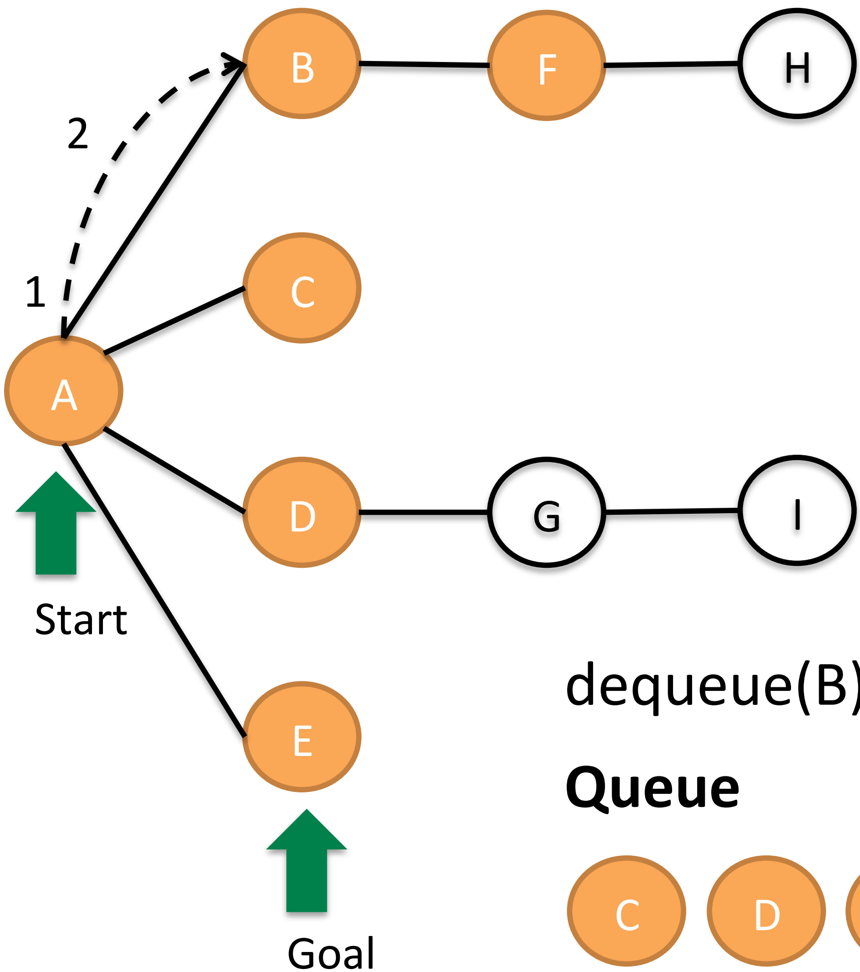
```
enqueue(s) //start node
s.visited = true
repeat until find goal vertex or
queue empty:
    u = dequeue()
    for v ∈ u.adjacent
        if !v.visited
            v.visited = true
            enqueue(v)
```

dequeue(A), unvisited enqueue adjacent

Queue



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



BFS algorithm

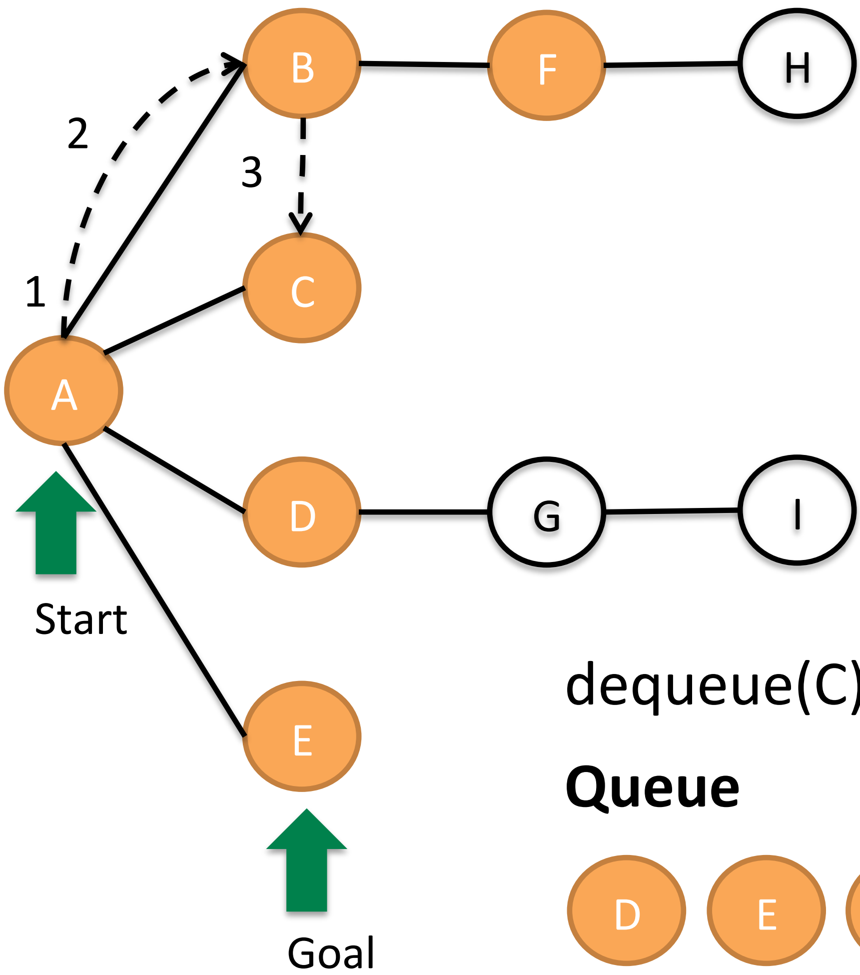
```
enqueue(s) //start node
s.visited = true
repeat until find goal vertex or
queue empty:
    u = dequeue()
    for v ∈ u.adjacent
        if !v.visited
            v.visited = true
            enqueue(v)
```

dequeue(B), enqueue unvisited adjacent F

Queue

C D E F

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



BFS algorithm

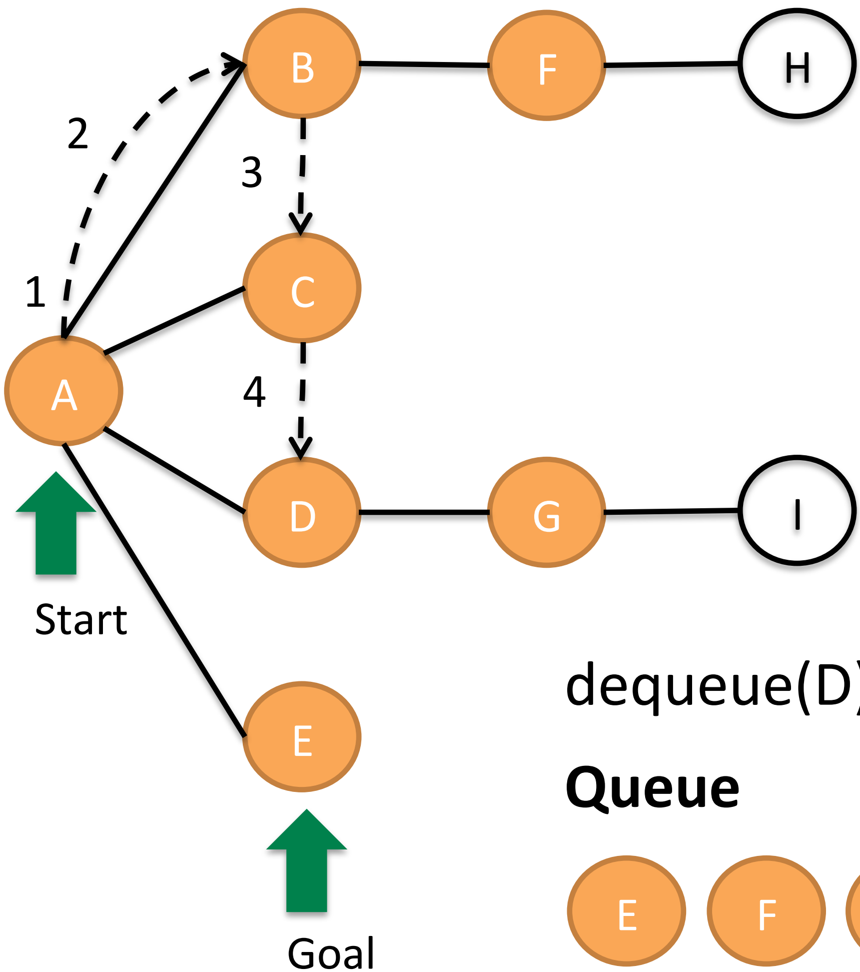
```
enqueue(s) //start node
s.visited = true
repeat until find goal vertex or
queue empty:
    u = dequeue()
    for v ∈ u.adjacent
        if !v.visited
            v.visited = true
            enqueue(v)
```

dequeue(C), enqueue unvisited adjacent (none)

Queue

D E F

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



BFS algorithm

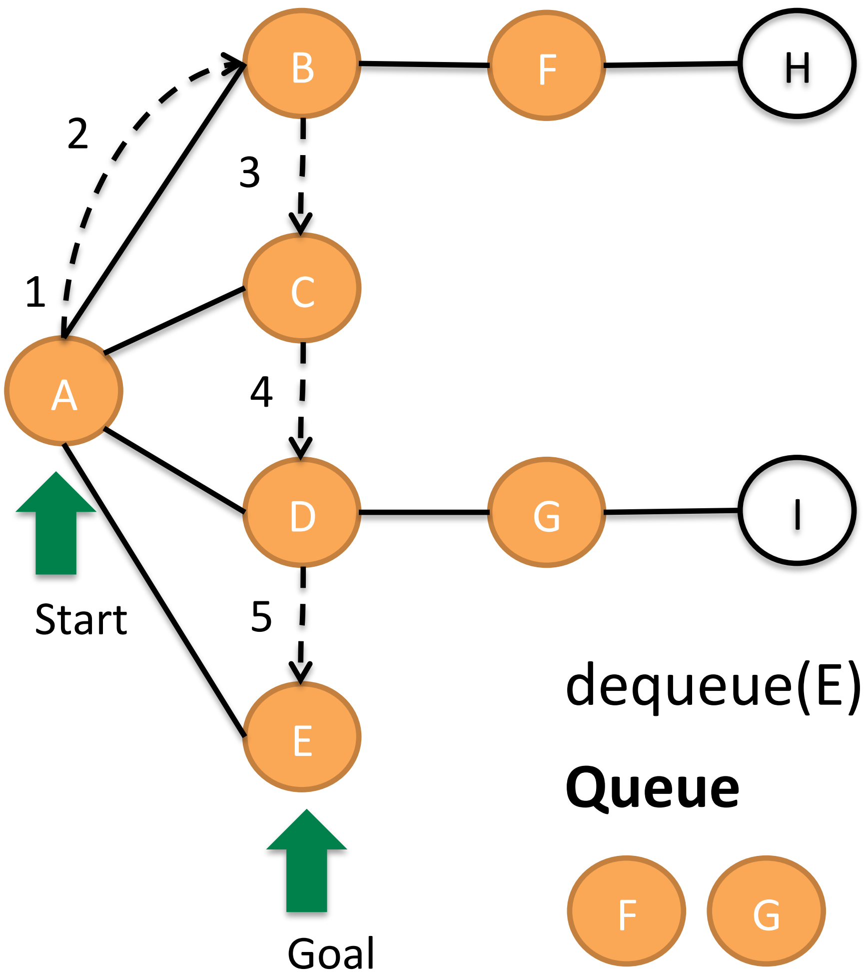
```
enqueue(s) //start node
s.visited = true
repeat until find goal vertex or
queue empty:
    u = dequeue()
    for v ∈ u.adjacent
        if !v.visited
            v.visited = true
            enqueue(v)
```

dequeue(D), enqueue unvisited adjacent G

Queue

E F G

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



BFS algorithm

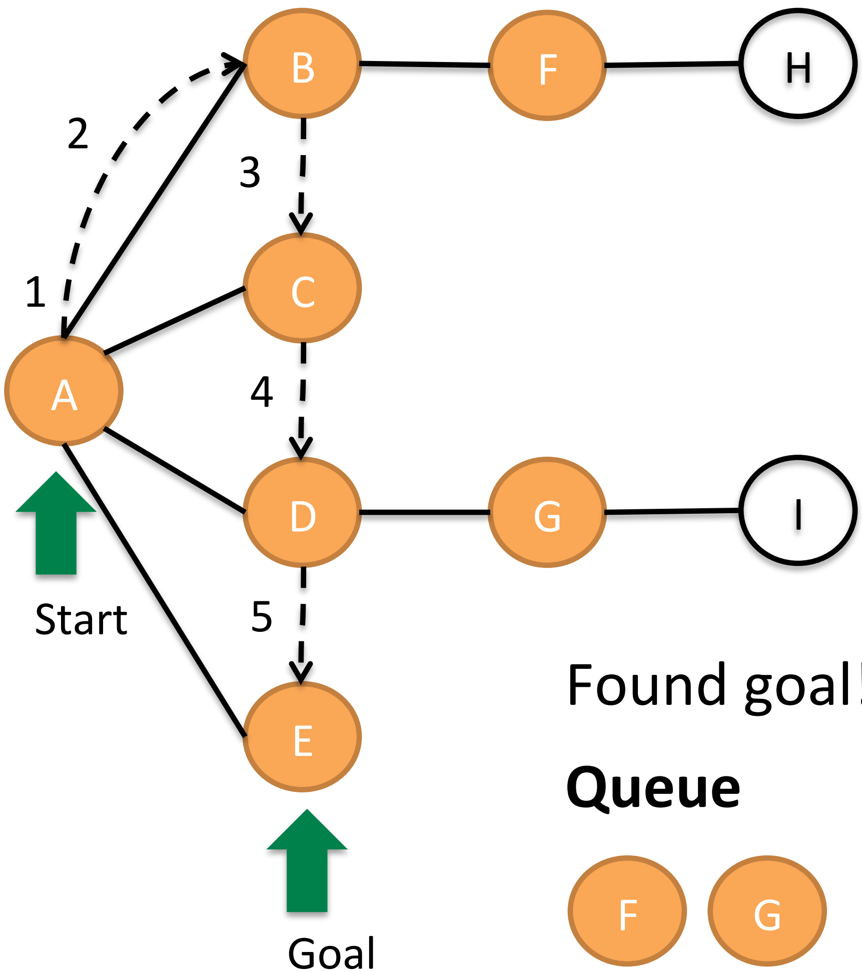
```
enqueue(s) //start node
s.visited = true
repeat until find goal vertex or
queue empty:
    u = dequeue()
    for v ∈ u.adjacent
        if !v.visited
            v.visited = true
            enqueue(v)
```

dequeue(E), enqueue unvisited adjacent

Queue

F G

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



Found goal!

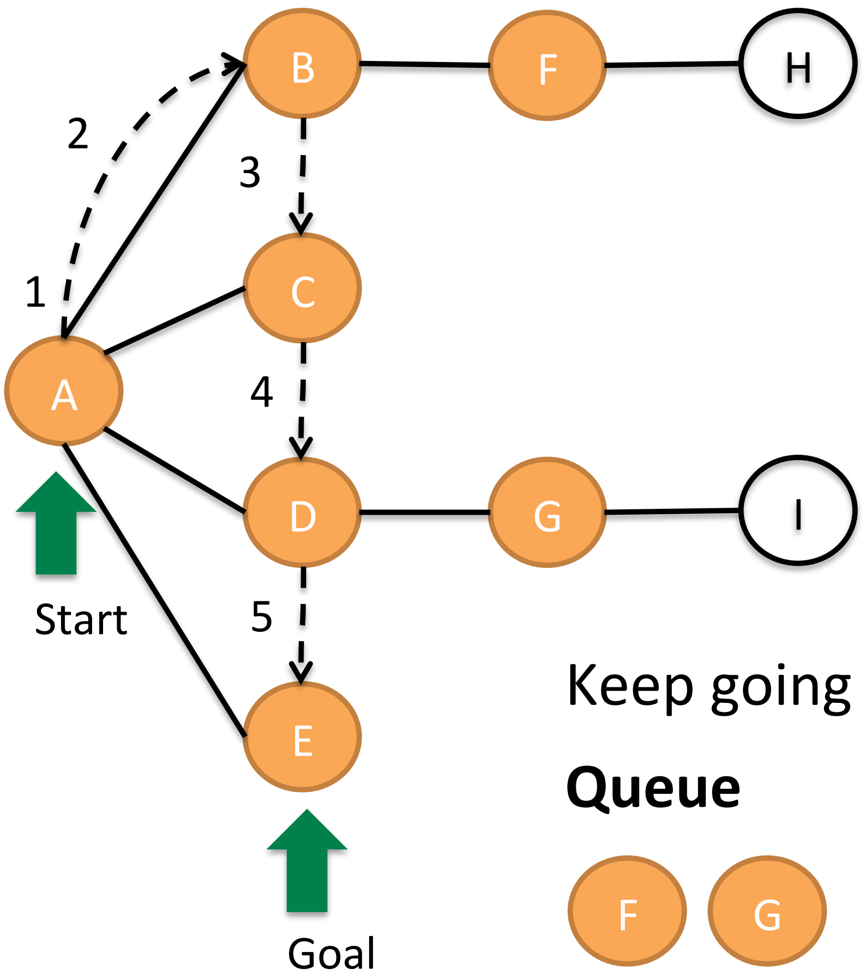
Queue

F G

BFS algorithm

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

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



BFS algorithm

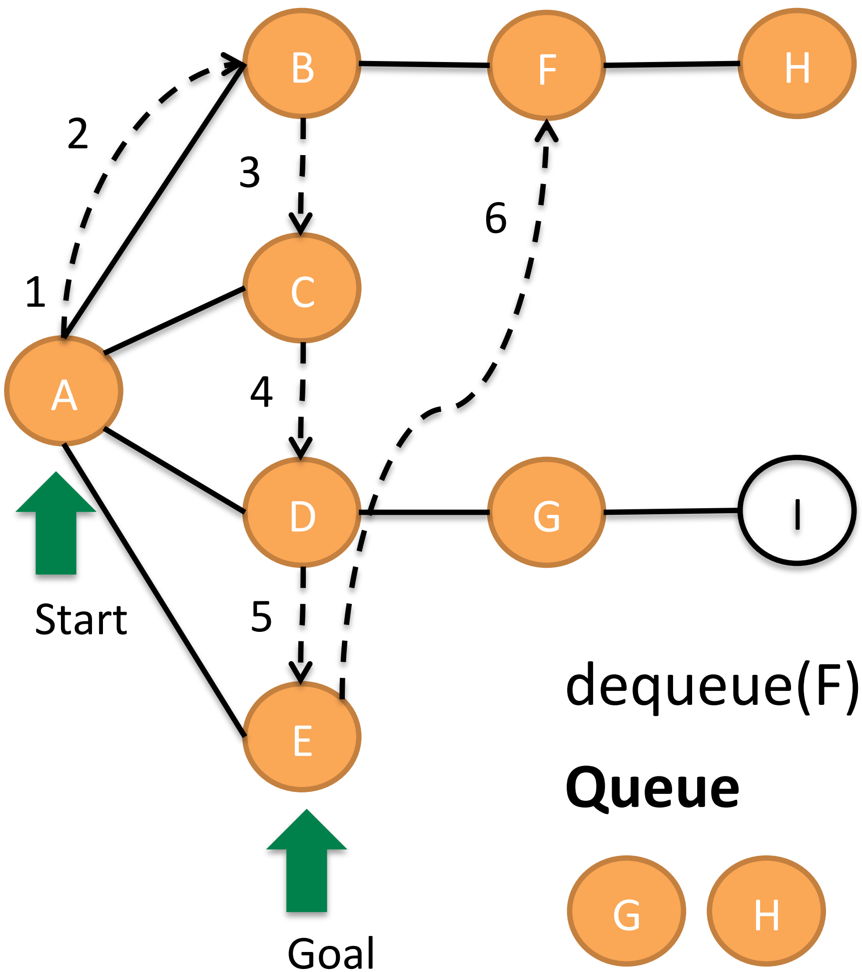
```
enqueue(s) //start node
s.visited = true
repeat until find goal vertex or
queue empty:
    u = dequeue()
    for v ∈ u.adjacent
        if !v.visited
            v.visited = true
            enqueue(v)
```

Keep going for fun

Queue

F G

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



BFS algorithm

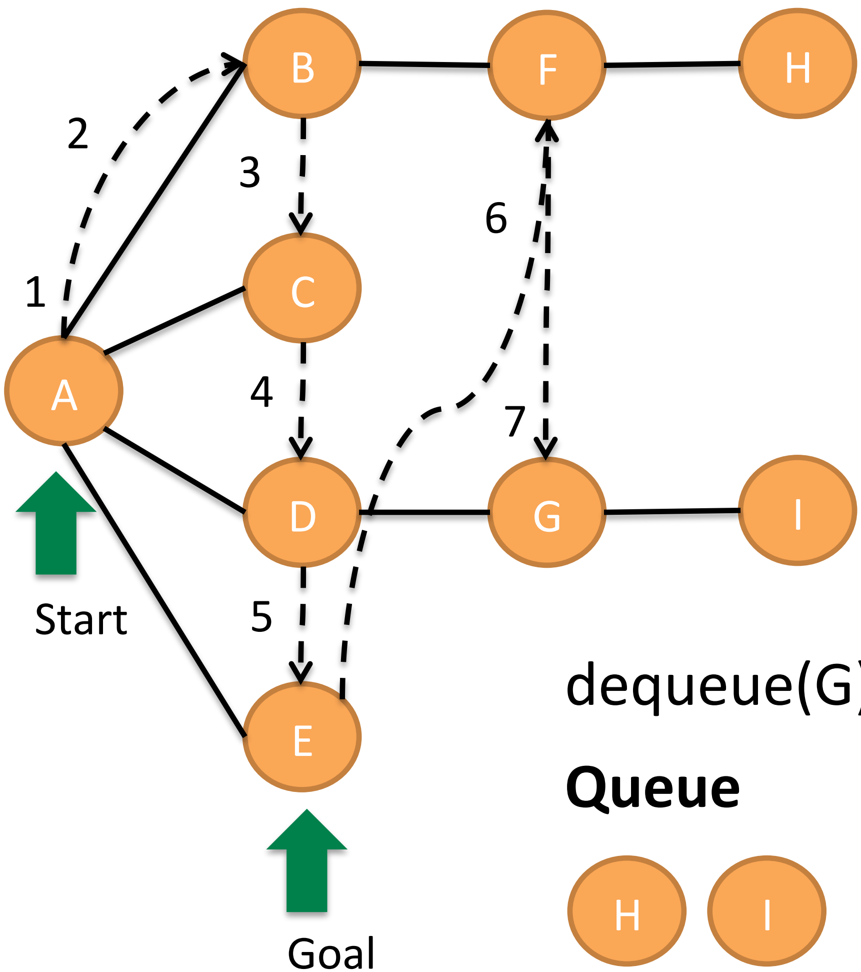
```
enqueue(s) //start node
s.visited = true
repeat until find goal vertex or
queue empty:
    u = dequeue()
    for v ∈ u.adjacent
        if !v.visited
            v.visited = true
            enqueue(v)
```

dequeue(F), enqueue unvisited adjacent H

Queue

G H

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



BFS algorithm

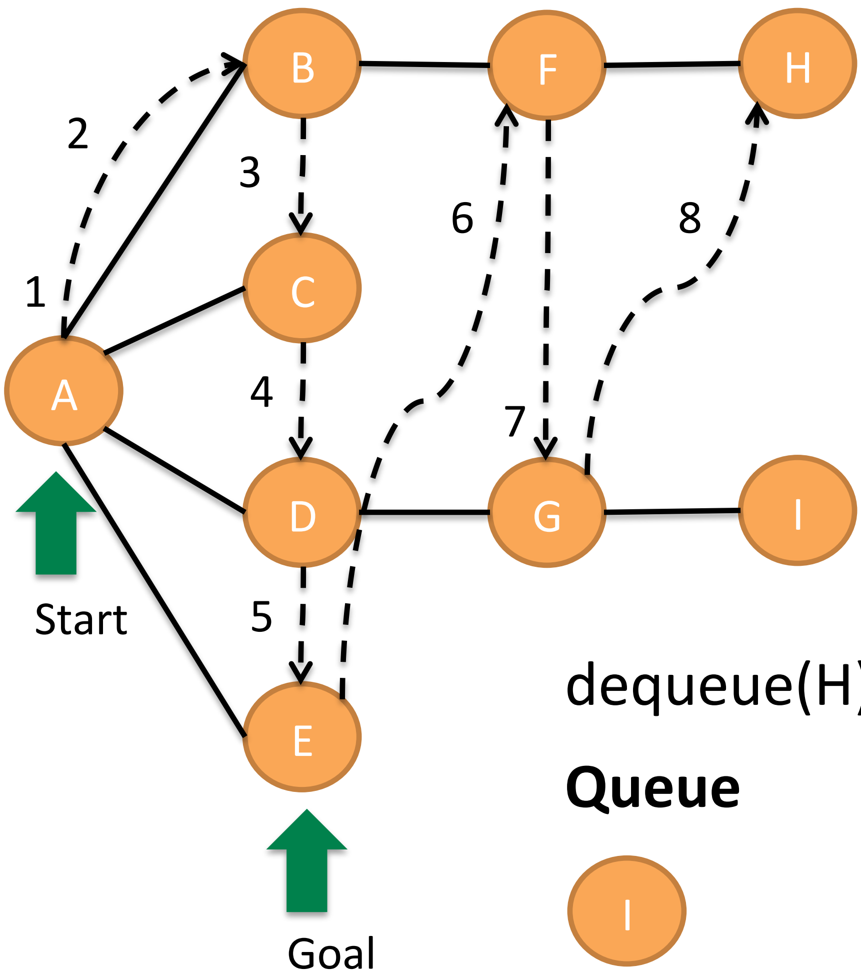
```
enqueue(s) //start node
s.visited = true
repeat until find goal vertex or
queue empty:
    u = dequeue()
    for v ∈ u.adjacent
        if !v.visited
            v.visited = true
            enqueue(v)
```

dequeue(G), enqueue unvisited adjacent I

Queue

H I

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



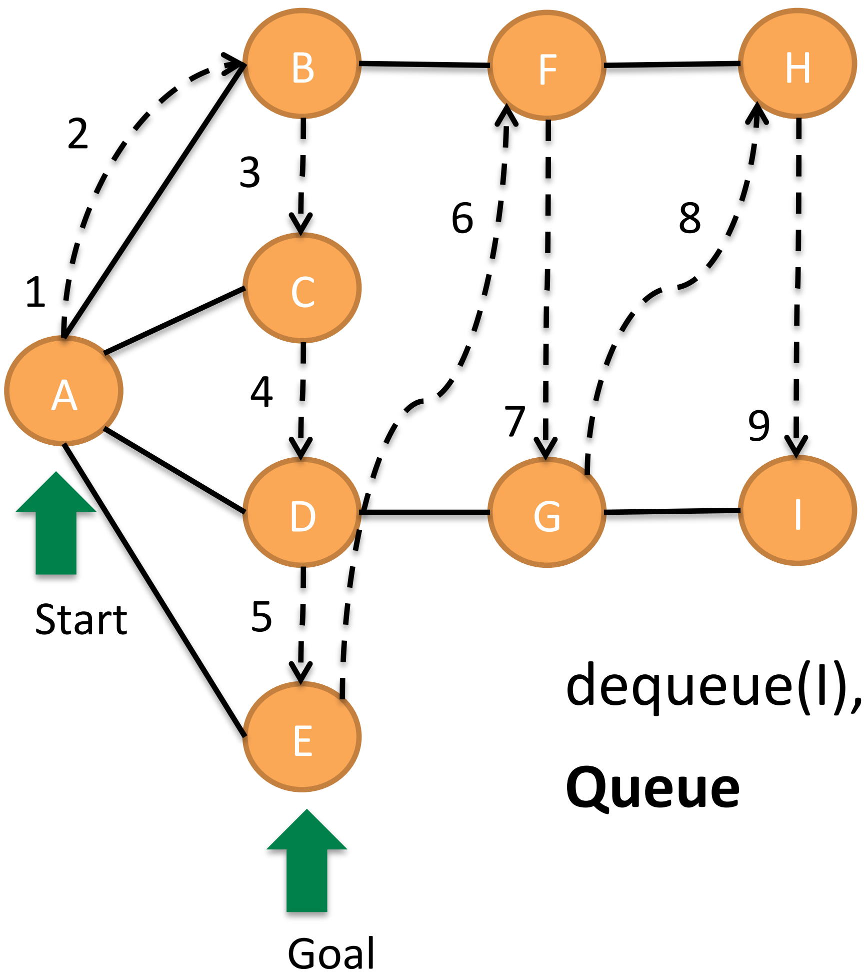
BFS algorithm

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

dequeue(H), enqueue unvisited adjacent (none)

Queue

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



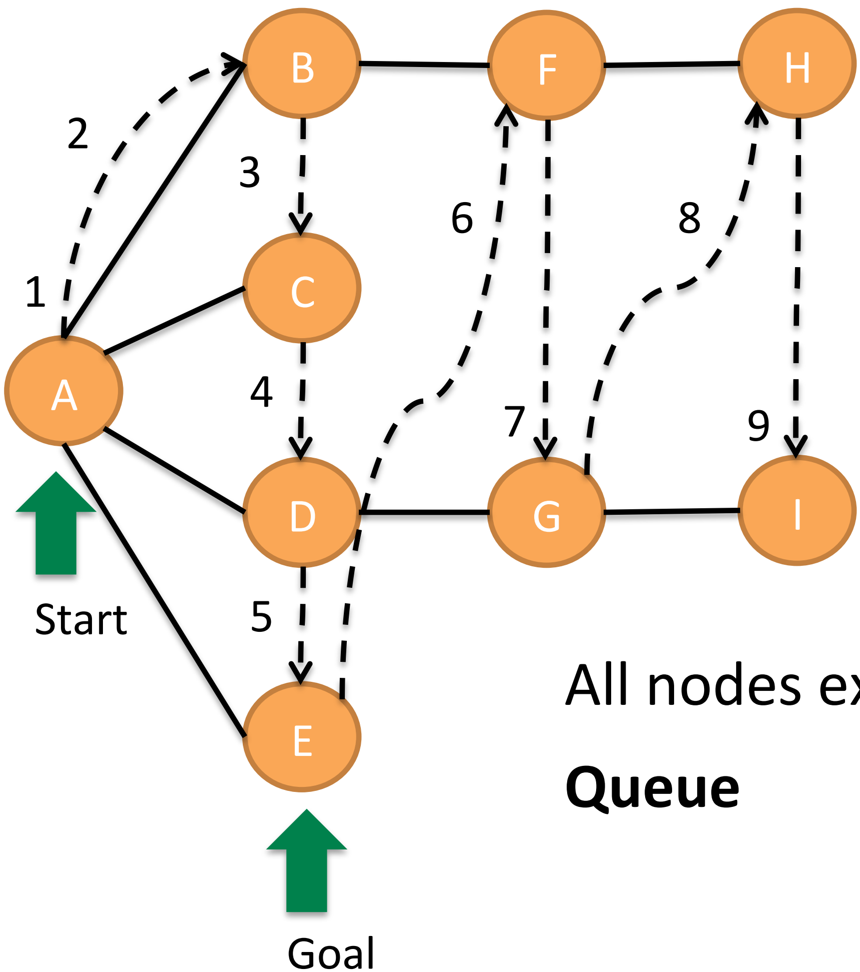
BFS algorithm

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

All nodes explored

Queue

Node discovery tells us something about the graph

Discovery edges

- Lead to unvisited nodes
- Form a tree on the graph
- Can traverse from start to goal (or any node)
- Can tell us which nodes are not reachable (not on path formed by discovery nodes)
- **Path guaranteed to have smallest number of edges**

Can track how we got to node to find shortest path

- Build vertex tree
- Parent of each vertex is vertex that discovered it
- Parent is unique because we don't visit vertices twice

Run time 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 O(n+m)
- Useful for the Kevin Bacon game!