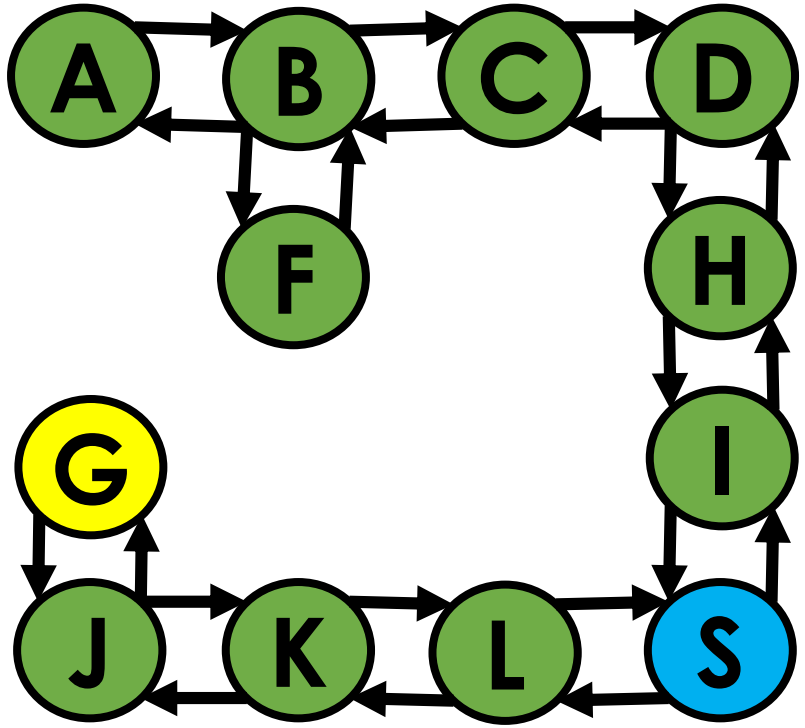
Graph Search

Part 2: Breadth-first Search

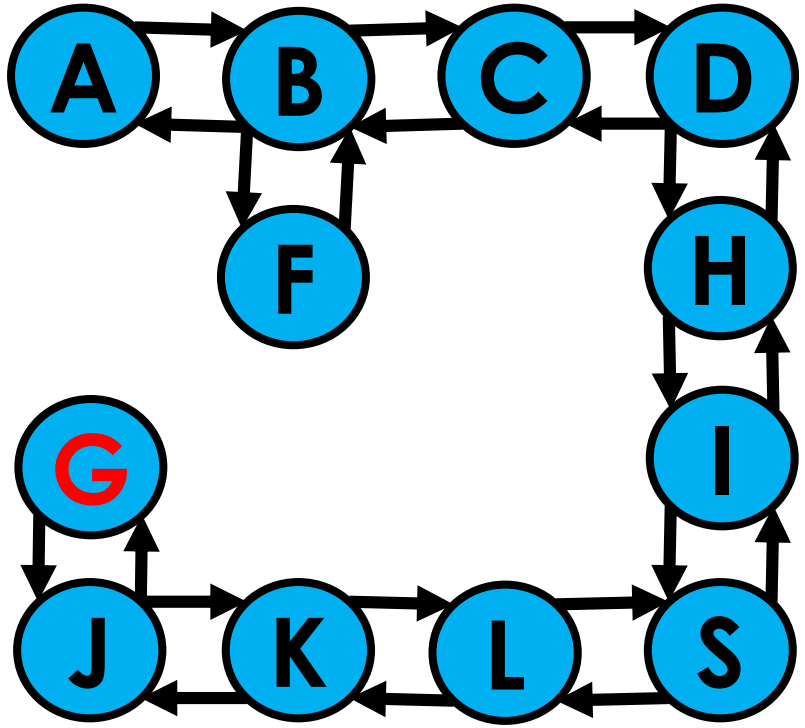
By the end of this video you will be able to…

- Perform Breadth-first Search (BFS) on a graph
- Implement the code for BFS
- Describe how ADT Queue works
- Describe how Queues are used in BFS



DFS: Algorithm

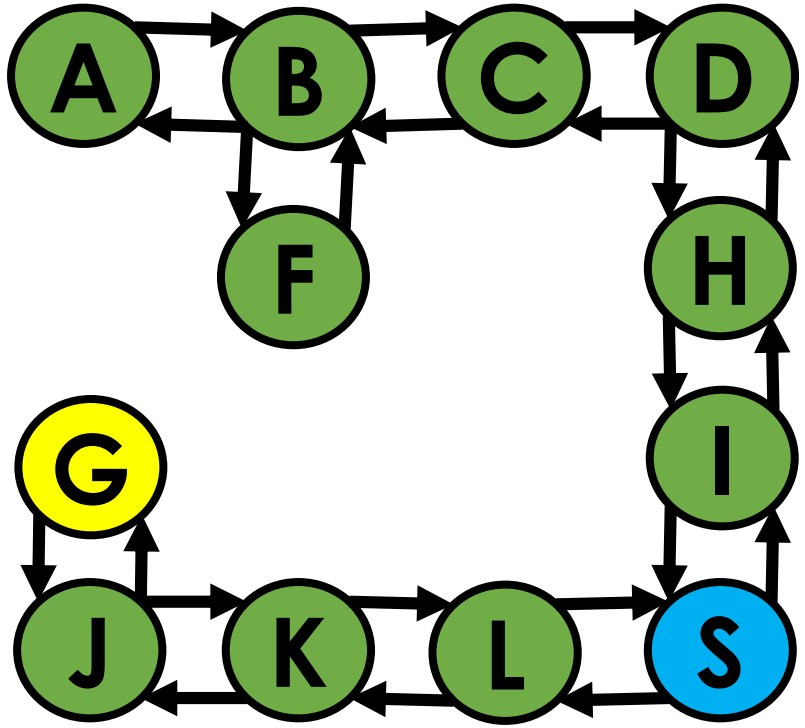
```
DFS(S, G):
    Initialize: stack, visited HashSet and parent HashMap
    Push S onto the stack
    while stack is not empty:
        pop node curr from top of stack
        if curr == G return parent map
        for each of curr's neighbors, n, not in visited set:
            add n to visited set
            add curr as n's parent in parent map
            push n onto the stack
    // If we get here then there's no path
```



Depth-first Search (DFS)

How to keep track of where to search next?

Stack: List where you add and remove from one end only:
push → add an element
pop → remove an element

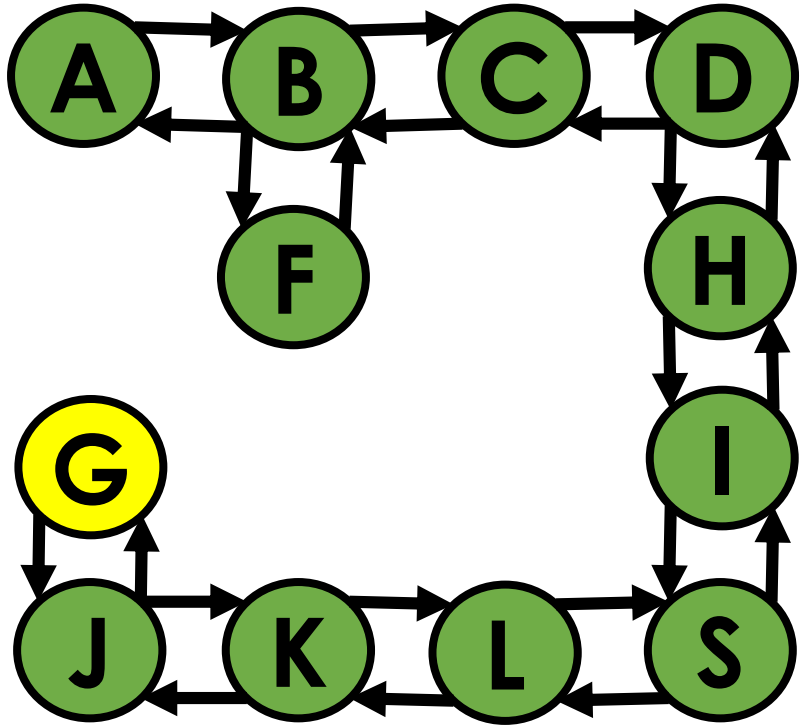


Breadth-first Search (BFS)

How to keep track of where to search next?

Queue: List where you add elements to one end and remove them from the other
enqueue → add an element
dequeue → remove an element

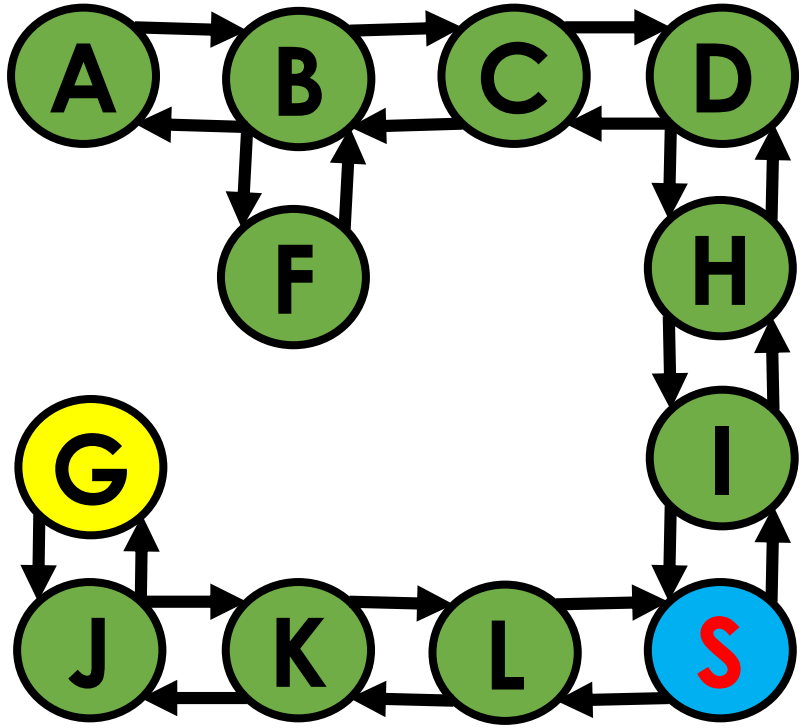
BFS: Algorithm




```
BFS(S, G):
    Initialize: queue, visited HashSet and parent HashMap
    Enqueue S onto the queue and add to visited
    while queue is not empty:
        dequeue node curr from top of queue
        if curr == G return parent map
        for each of curr's neighbors, n, not in visited set:
            add n to visited set
            add curr as n's parent in parent map
            enqueue n onto the queue
    // If we get here then there's no path
```

queue: S
curr:
visited: S

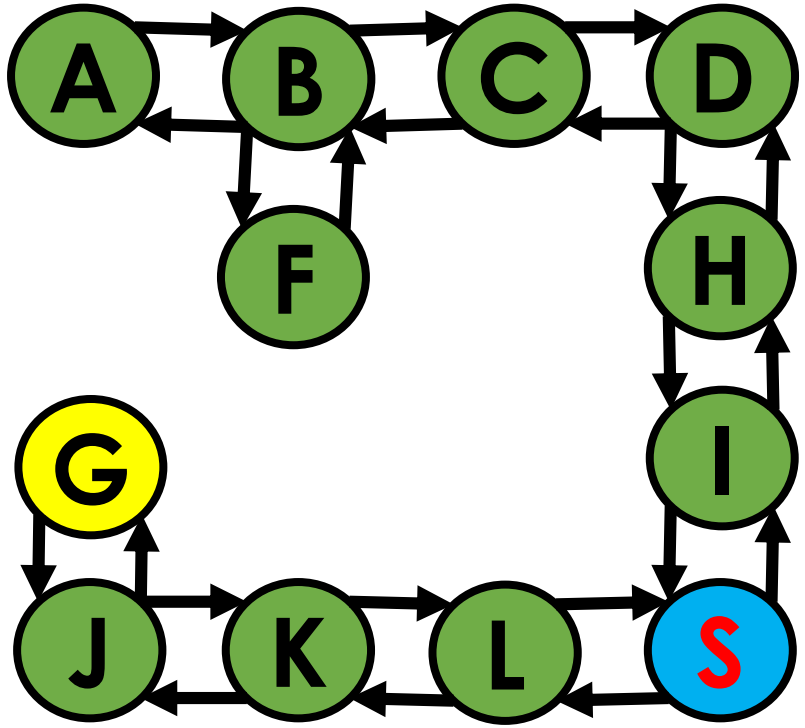
BFS: Algorithm




```
BFS(S, G):
    Initialize: queue, visited HashSet and parent HashMap
    Enqueue S onto the queue and add to visited
    while queue is not empty:
        dequeue node curr from top of queue
        if curr == G return parent map
        for each of curr's neighbors, n, not in visited set:
            add n to visited set
            add curr as n's parent in parent map
            enqueue n onto the queue
    // If we get here then there's no path
```

queue:
curr: S
visited: S

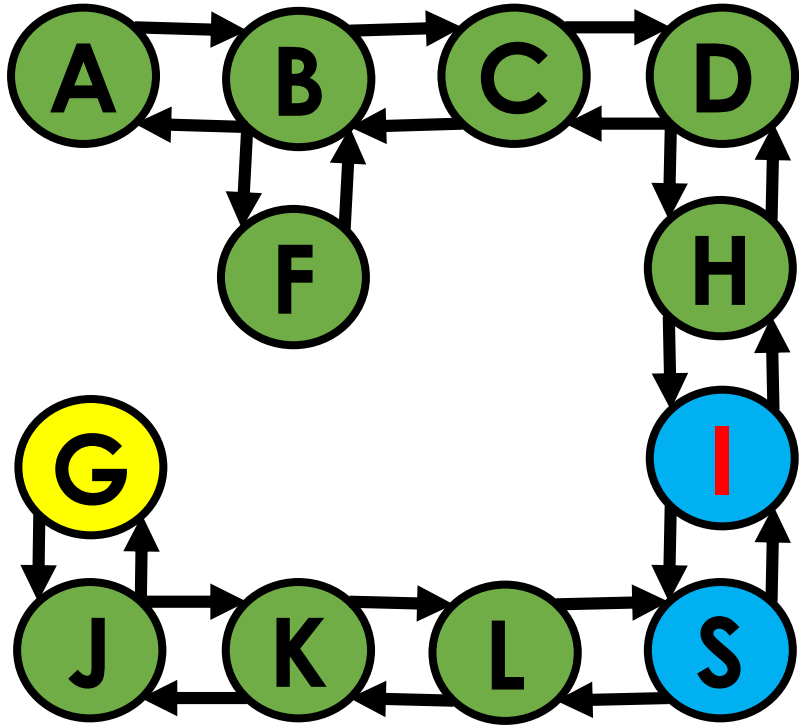
BFS: Algorithm




```
BFS(S, G):
    Initialize: queue, visited HashSet and parent HashMap
    Enqueue S onto the queue and add to visited
    while queue is not empty:
        dequeue node curr from top of queue
        if curr == G return parent map
        for each of curr's neighbors, n, not in visited set:
            add n to visited set
            add curr as n's parent in parent map
            enqueue n onto the queue
    // If we get here then there's no path
```

queue: I, L
curr: S
visited: S, I, L

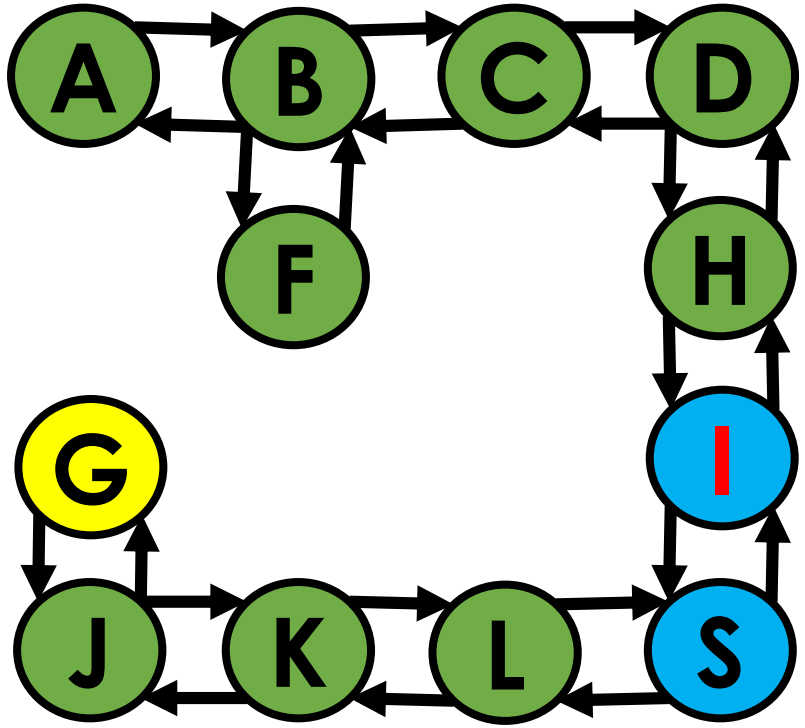
BFS: Algorithm




```
BFS(S, G):
    Initialize: queue, visited HashSet and parent HashMap
    Enqueue S onto the queue and add to visited
    while queue is not empty:
        dequeue node curr from top of queue
        if curr == G return parent map
        for each of curr's neighbors, n, not in visited set:
            add n to visited set
            add curr as n's parent in parent map
            enqueue n onto the queue
    // If we get here then there's no path
```

queue: L
curr: I
visited: S, I, L

BFS: Algorithm

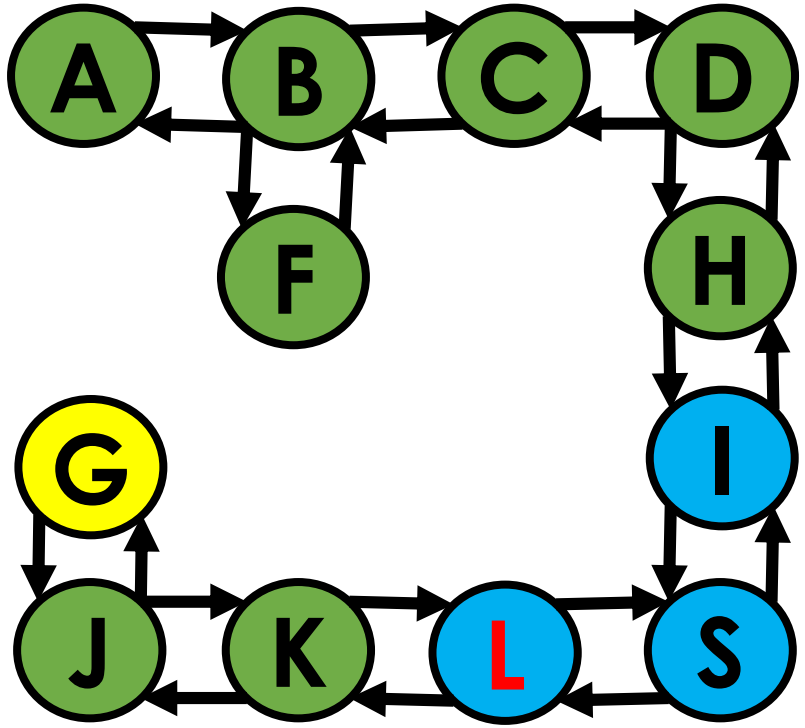



```
BFS(S, G):
    Initialize: queue, visited HashSet and parent HashMap
    Enqueue S onto the queue and add to visited
    while queue is not empty:
        dequeue node curr from top of queue
        if curr == G return parent map
        for each of curr's neighbors, n, not in visited set:
            add n to visited set
            add curr as n's parent in parent map
            enqueue n onto the queue
    // If we get here then there's no path
```

queue: L, H
curr: I
visited: S, I, L, H

What node will we explore next?

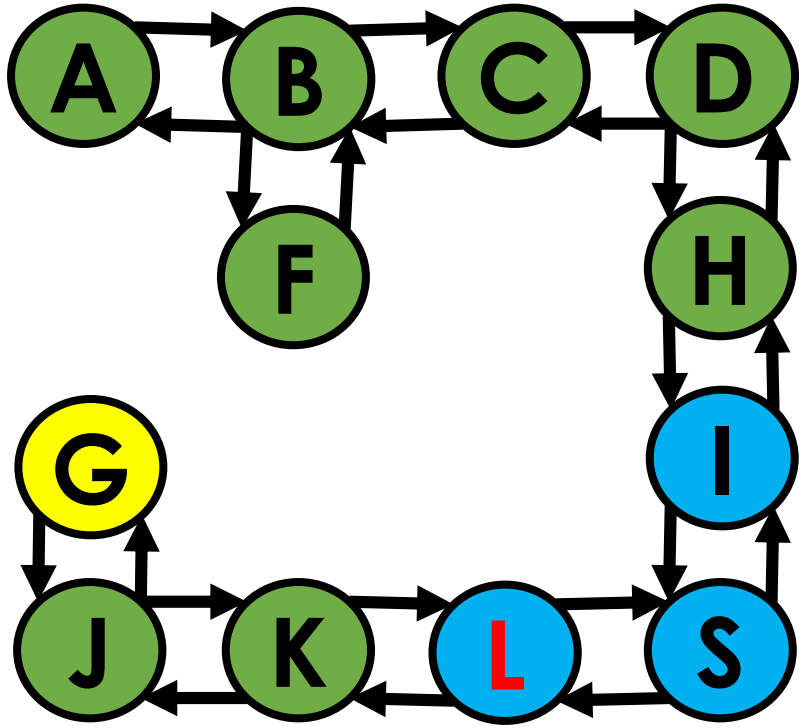
BFS: Algorithm




```
BFS(S, G):
    Initialize: queue, visited HashSet and parent HashMap
    Enqueue S onto the queue and add to visited
    while queue is not empty:
        dequeue node curr from top of queue
        if curr == G return parent map
        for each of curr's neighbors, n, not in visited set:
            add n to visited set
            add curr as n's parent in parent map
            enqueue n onto the queue
    // If we get here then there's no path
```

queue: H
curr: L
visited: S, I, L, H

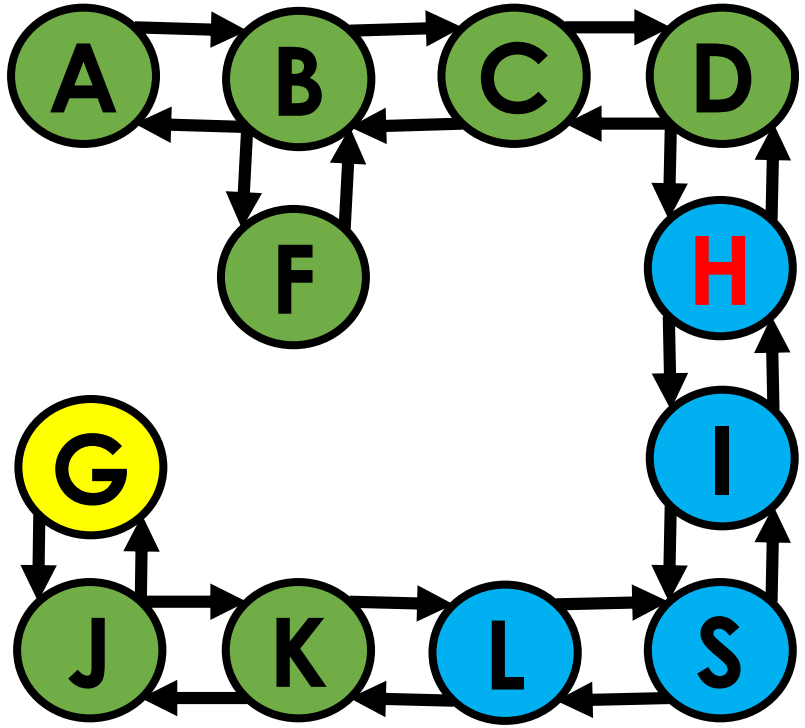
BFS: Algorithm




```
BFS(S, G):
    Initialize: queue, visited HashSet and parent HashMap
    Enqueue S onto the queue and add to visited
    while queue is not empty:
        dequeue node curr from top of queue
        if curr == G return parent map
        for each of curr's neighbors, n, not in visited set:
            add n to visited set
            add curr as n's parent in parent map
            enqueue n onto the queue
    // If we get here then there's no path
```

queue: H, K
curr: L
visited: S, I, L, H, K

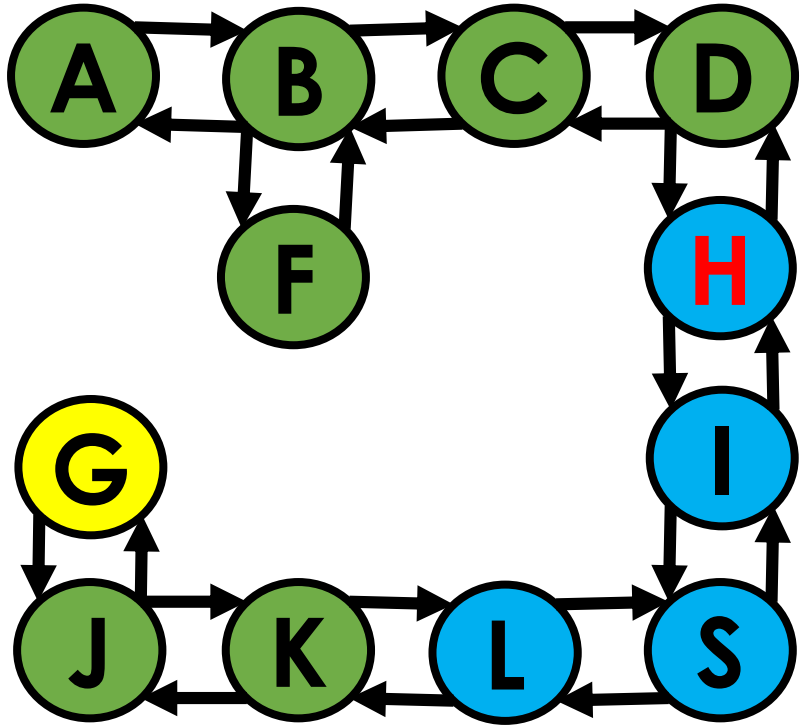
BFS: Algorithm




```
BFS(S, G):
    Initialize: queue, visited HashSet and parent HashMap
    Enqueue S onto the queue and add to visited
    while queue is not empty:
        dequeue node curr from top of queue
        if curr == G return parent map
        for each of curr's neighbors, n, not in visited set:
            add n to visited set
            add curr as n's parent in parent map
            enqueue n onto the queue
    // If we get here then there's no path
```

queue: K
curr: H
visited: S, I, L, H, K

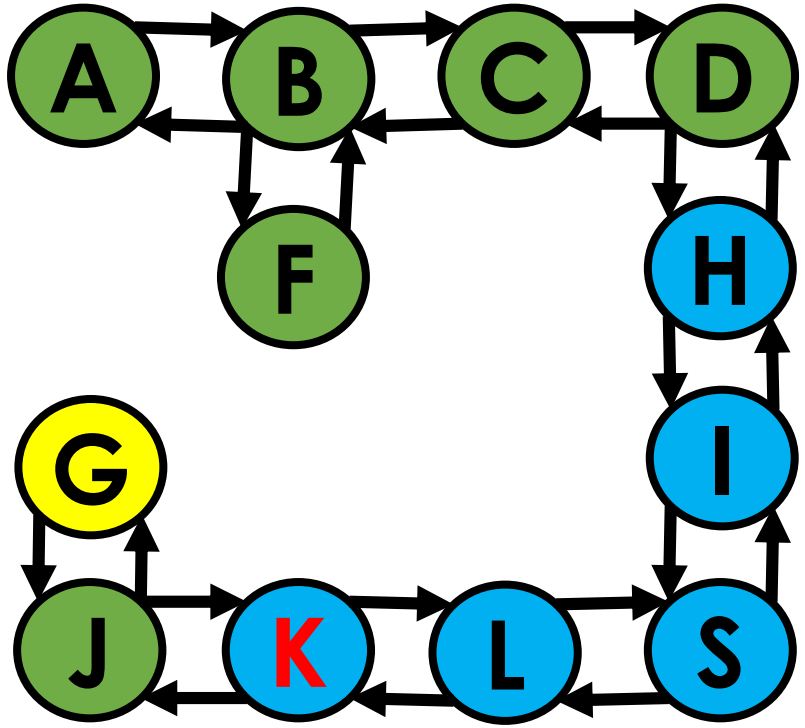
BFS: Algorithm




```
BFS(S, G):
    Initialize: queue, visited HashSet and parent HashMap
    Enqueue S onto the queue and add to visited
    while queue is not empty:
        dequeue node curr from top of queue
        if curr == G return parent map
        for each of curr's neighbors, n, not in visited set:
            add n to visited set
            add curr as n's parent in parent map
            enqueue n onto the queue
    // If we get here then there's no path
```

queue: K, D
curr: H
visited: S, I, L, H, K, D

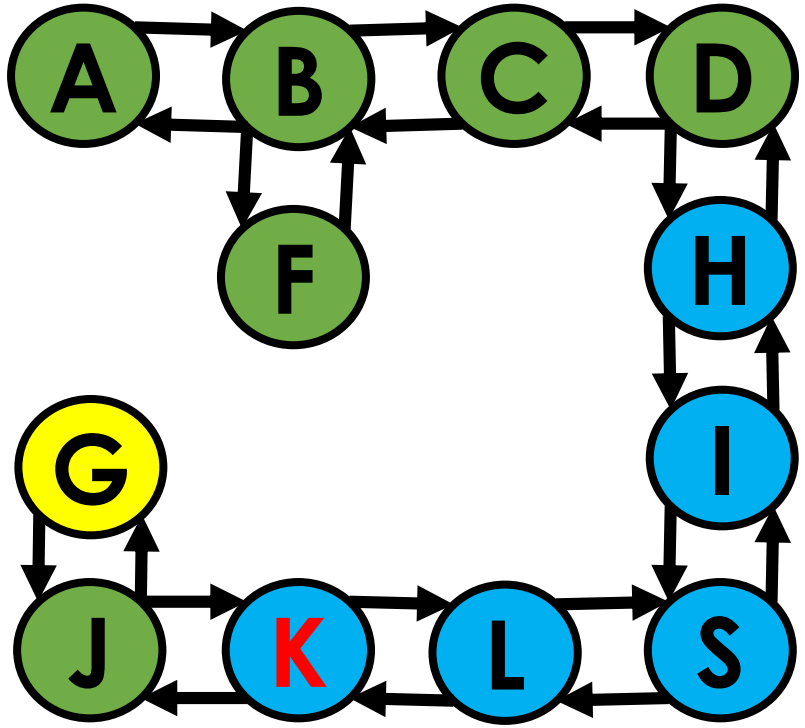
BFS: Algorithm




```
BFS(S, G):
    Initialize: queue, visited HashSet and parent HashMap
    Enqueue S onto the queue and add to visited
    while queue is not empty:
        dequeue node curr from top of queue
        if curr == G return parent map
        for each of curr's neighbors, n, not in visited set:
            add n to visited set
            add curr as n's parent in parent map
            enqueue n onto the queue
    // If we get here then there's no path
```

queue: D
curr: K
visited: S, I, L, H, K, D

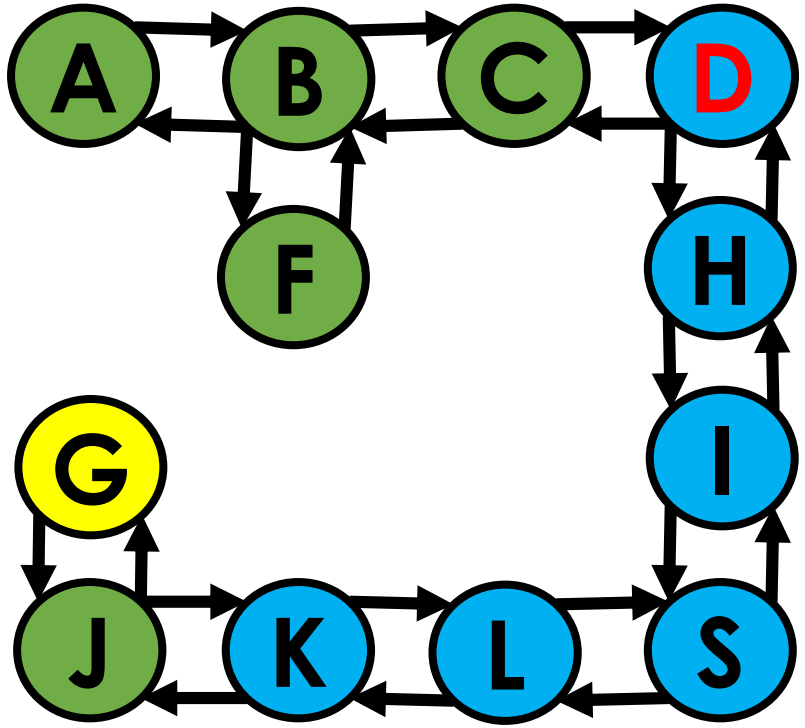
BFS: Algorithm




```
BFS(S, G):
    Initialize: queue, visited HashSet and parent HashMap
    Enqueue S onto the queue and add to visited
    while queue is not empty:
        dequeue node curr from top of queue
        if curr == G return parent map
        for each of curr's neighbors, n, not in visited set:
            add n to visited set
            add curr as n's parent in parent map
            enqueue n onto the queue
    // If we get here then there's no path
```

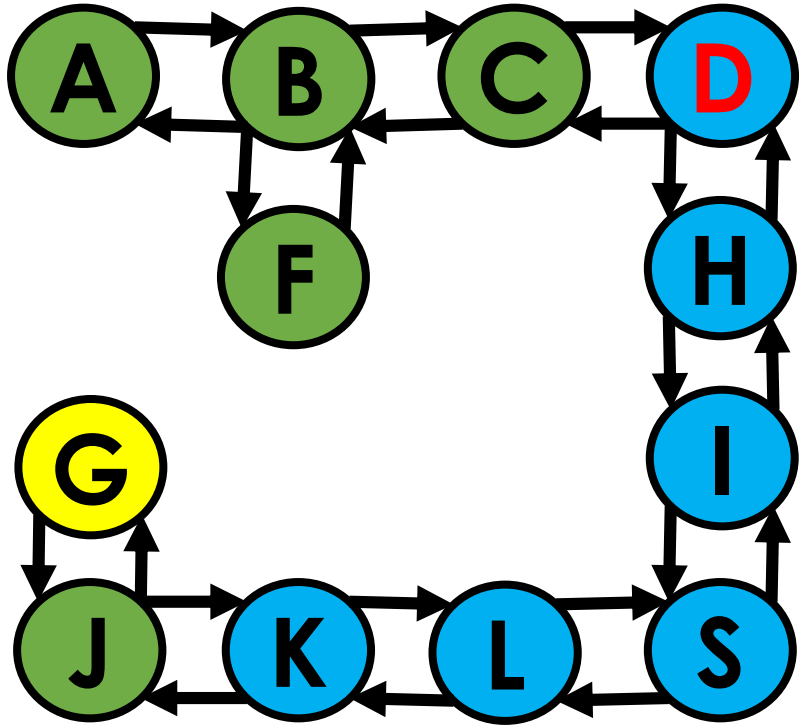
queue: D, J
curr: K
visited: S, I, L, H, K, D, J

BFS: Algorithm




```
BFS(S, G):
    Initialize: queue, visited HashSet and parent HashMap
    Enqueue S onto the queue and add to visited
    while queue is not empty:
        dequeue node curr from top of queue
        if curr == G return parent map
        for each of curr's neighbors, n, not in visited set:
            add n to visited set
            add curr as n's parent in parent map
            enqueue n onto the queue
    // If we get here then there's no path
```

queue: J
curr: D
visited: S, I, L, H, K, D, J

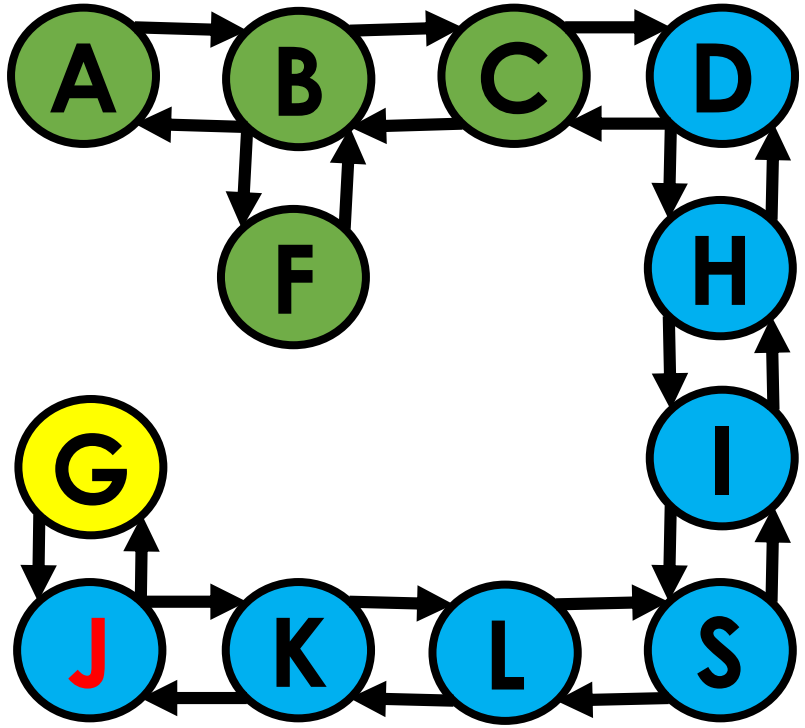


BFS: Algorithm

```
BFS(S, G):
    Initialize: queue, visited HashSet and parent HashMap
    Enqueue S onto the queue and add to visited
    while queue is not empty:
        dequeue node curr from top of queue
        if curr == G return parent map
        for each of curr's neighbors, n, not in visited set:
            add n to visited set
            add curr as n's parent in parent map
            enqueue n onto the queue
    // If we get here then there's no path
```

queue: J, C
curr: D
visited: S, I, L, H, K, D, J, C

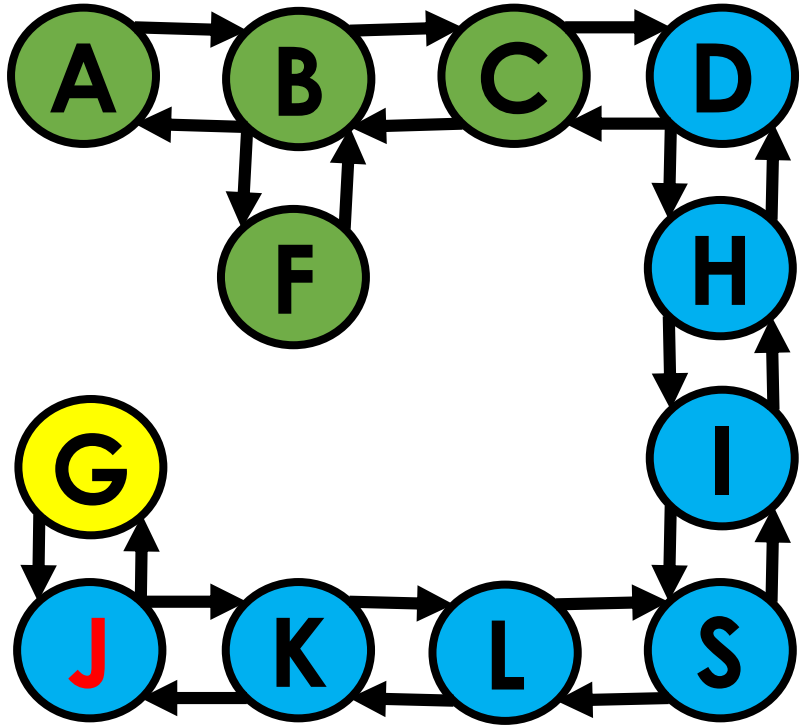
BFS: Algorithm




```
BFS(S, G):
    Initialize: queue, visited HashSet and parent HashMap
    Enqueue S onto the queue and add to visited
    while queue is not empty:
        dequeue node curr from top of queue
        if curr == G return parent map
        for each of curr's neighbors, n, not in visited set:
            add n to visited set
            add curr as n's parent in parent map
            enqueue n onto the queue
    // If we get here then there's no path
```

queue: C
curr: J
visited: S, I, L, H, K, D, J, C

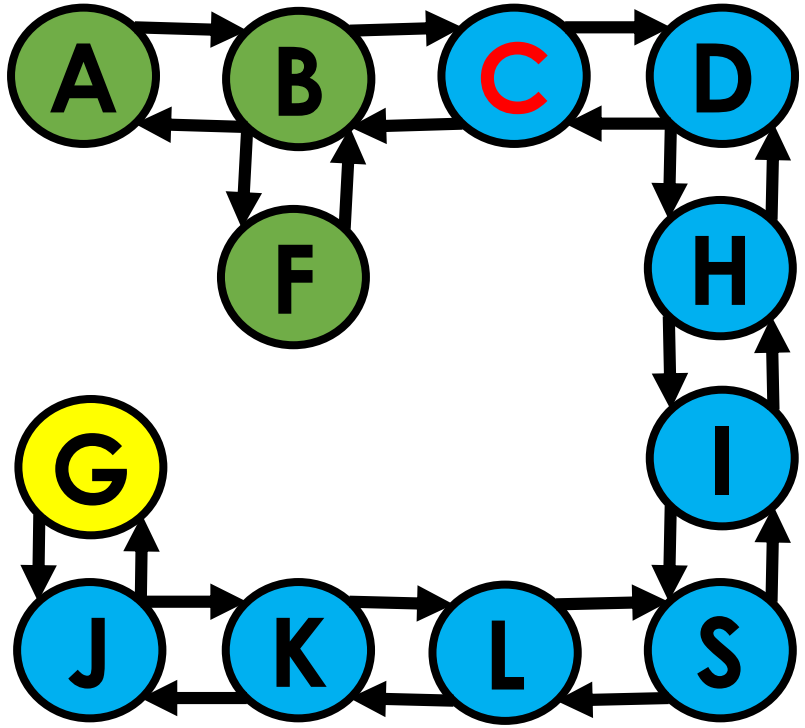
BFS: Algorithm




```
BFS(S, G):
    Initialize: queue, visited HashSet and parent HashMap
    Enqueue S onto the queue and add to visited
    while queue is not empty:
        dequeue node curr from top of queue
        if curr == G return parent map
        for each of curr's neighbors, n, not in visited set:
            add n to visited set
            add curr as n's parent in parent map
            enqueue n onto the queue
    // If we get here then there's no path
```

queue: C, G
curr: J
visited: S, I, L, H, K, D, J, C, G

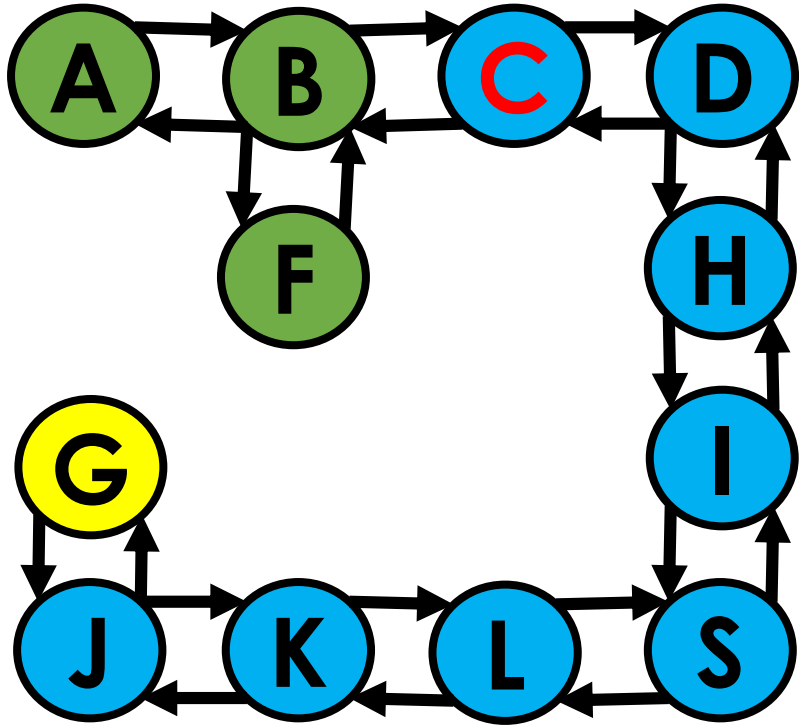
BFS: Algorithm




```
BFS(S, G):
    Initialize: queue, visited HashSet and parent HashMap
    Enqueue S onto the queue and add to visited
    while queue is not empty:
        dequeue node curr from top of queue
        if curr == G return parent map
        for each of curr's neighbors, n, not in visited set:
            add n to visited set
            add curr as n's parent in parent map
            enqueue n onto the queue
    // If we get here then there's no path
```

queue: G
curr: C
visited: S, I, L, H, K, D, J, C, G

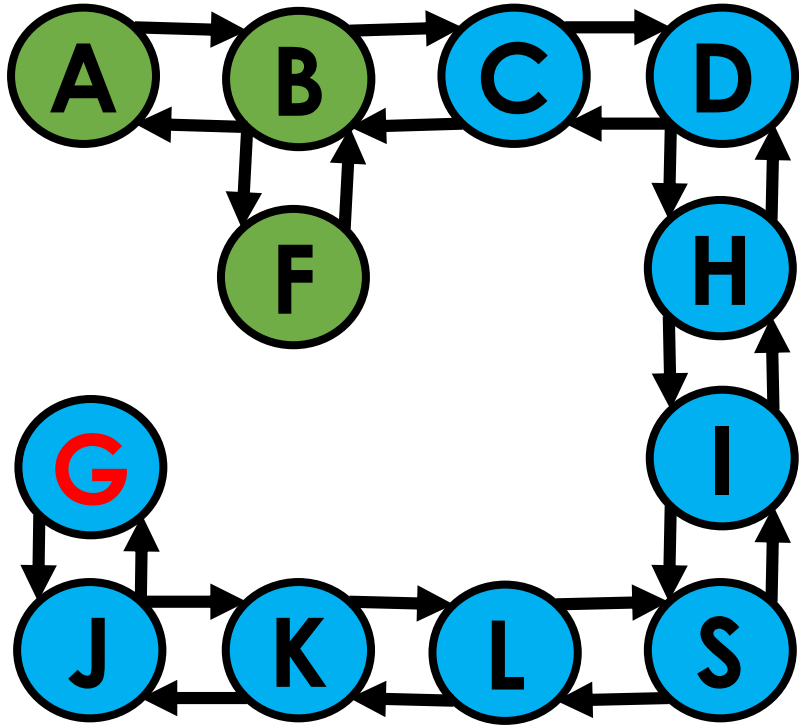
BFS: Algorithm



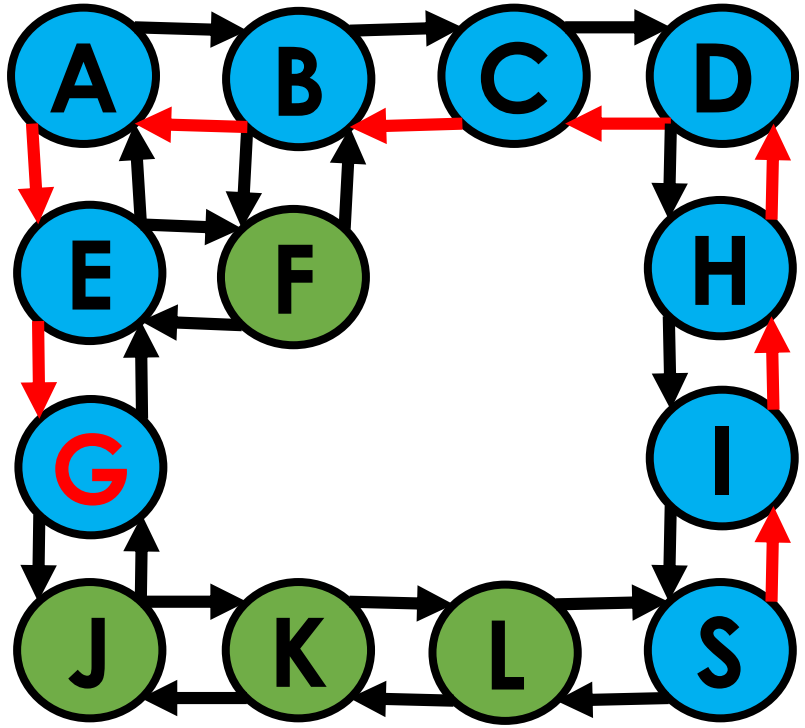

```
BFS(S, G):
    Initialize: queue, visited HashSet and parent HashMap
    Enqueue S onto the queue and add to visited
    while queue is not empty:
        dequeue node curr from top of queue
        if curr == G return parent map
        for each of curr's neighbors, n, not in visited set:
            add n to visited set
            add curr as n's parent in parent map
            enqueue n onto the queue
    // If we get here then there's no path
```

queue: G, B
curr: C
visited: S, I, L, H, K, D, J, C, G, B

BFS: Algorithm

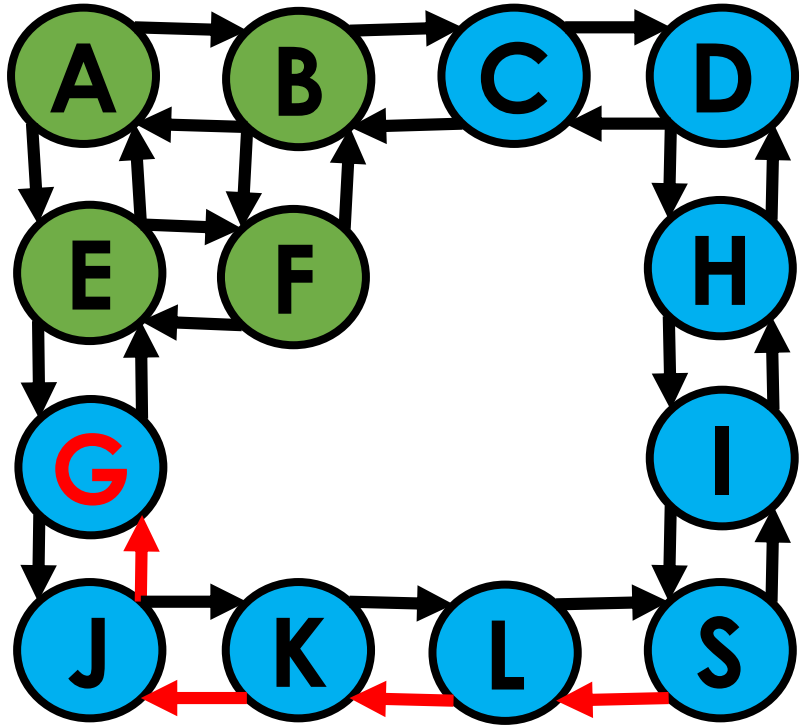


queue: B
curr: G
visited: S, I, L, H, K, D, J, C, G, B



Depth-first Search (DFS)

Problem: This is a very long path!



Breadth-first Search (BFS)

BFS finds a shorter path!