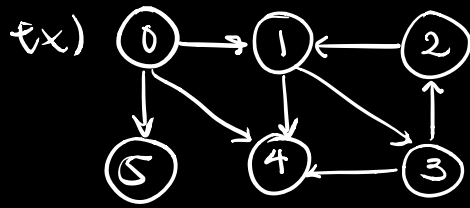


Route Between Nodes: given a directed graph, design an algorithm to find out whether there is a route between two nodes.



input: 0, 3
output: true

input: 2, 5
output: false

Adjacency List Representation

0: 1, 4, 5

1: 2, 3, 4

2: 1

3: 4

4:

5:

? are the nodes specified as start or destination? If so,

0 → 3 is true but 3 → 0 is false

BFS or DFS?

Breadth-first search since we are interested in quickly finding any path that exists between two nodes.
It isn't necessary to visit every node.

Implementation

```
function hasPath(p, q) {  
  if (p === null || q === null) {  
    return false;  
  }  
  
  let nodeQueue = [];  
  p.visited = true;  
  
  nodeQueue.push(p);  
  while (!nodeQueue.isEmpty()) {  
    let currNode = nodeQueue.shift();  
    for (adjNode in currNode.adjacent) {  
      if (adjNode === q) {  
        return true;  
      }  
    }  
  }  
}
```

```

    if (adjNode.visited === false) {
        adjNode.visited === true;
        nodeQueue.push(adjNode);
    } // we don't want to add an already
      // visited node back into the queue
  }
}
return false;
}

```

if source/destination do not matter

- use bidirectional sort instead
- call the function again but swap p & q
use a flag (checked both) to prevent an infinite swapping loop

Test it out:

p = 0, q = 3

- 1) queue = [0] → currNode = 0, queue = []
- 2) queue = [1, 4, 5], currNode = 1, queue = [4, 5]
- 3) queue = [4, 5, 3], currNode = 4, queue = [5, 3]
- 4) queue = [5, 3], currNode = 5, queue = [3]
- 5) queue = [3], currNode = 3 = q = return true

Complexity

$O(n)$ time, we could visit every node in the worst case

$O(n)$ space because of the usage of the queue to contain nodes