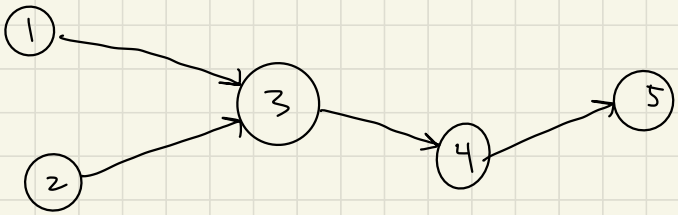
Kahn's Algorithm



adjList: 1 [3]
2 [3]
3 [4]
4 [5]
5 [ ]

1st: Count indegree via iteration

```
List numIn
for ( List subList : adjList)
    for ( Integer n: subList)
        numIn[n] ++
```

2nd: Use a queue, push the indegrees of 0, then pop them and everytime it pops, decrease the # of indegrees in nodes that it points at (essentially removing from graph)

Make our end list, queue

```
List sorted
Queue q

// Lets begin by adding all the 0 indegree to q
for ( i to numIn )
    if ( numIn[i] == 0)
        q.add(i)
```

Could look like this

q: (1) (2)

Kahns extended.

```
while( !q.isEmpty) {
    int curr = q.poll()      // take out the curr node
     sorted.add(curr)        // add to sorted list
                             // must dec. its connected
  for(int adj: adjList[curr] indegrees, can do this
    numIn[adj]--             iteratively through sublist
    if(numIn[adj]==0)        of curr, then check
        q.add(adj)           if it is 0
}
return sorted
```

*Note: We can determine if it is acyclic if our sorted > # vertices

Short Example:

q: ~~1~~ 2

numIn: 1: 0, 2: 0, 3: ~~2~~ 1

1. Remove 1

numIn: 1: 0, 2: 0, 3: ~~1~~ 0

2. Remove 2

q: ~~2~~

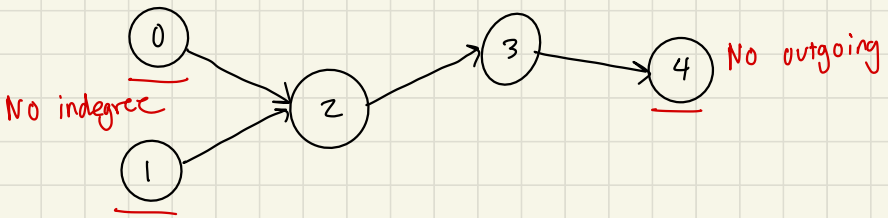
numIn: 1: 0, 2: 0, 3: 0

3. Add 3

q: 3

Using DFS

Using DFS, we must utilize a stack, visited list



We want the no outgoing to the bottom of the stack, and we want no indegrees to the top of our stack, Reasoning is because if we perform dfs on node 4, it will lead to nowhere because it doesn't point anywhere so beginning the traversal there is useless.

```
helperDFS (curr, adjList, numVert, stk, [] visited)
    visited[curr] = true // update visit
    for (int neighbors)
        if neighbor NOT visited
            helperDFS(neighbor) // visit neighbor
    stk.push(curr)
```

The recursive method will ensure we perform dfs on graph and will add nodes to stack

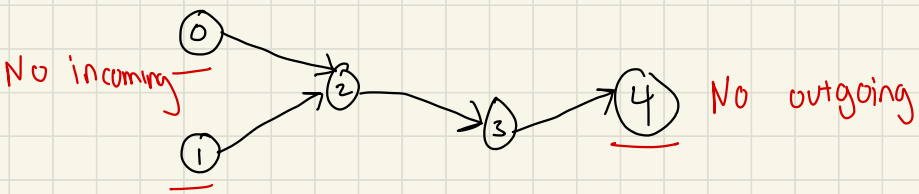
```
DFSTopological
    for(i to adjList)
        if visited[i] = false
            helperDFS(i, ..)
```

} ensures we visit all nodes

```
while ! stk.empty
    pop + add nodes to final list
```

Using BFS

Given: Find no outgoing edges, go last
Find no incoming edges, add first



Since dfs is w/ a queue

q: 0 1 4

Now we can apply bfs on the graph, making sure we keep track of visited and empty the queue

Very similar to Kahn, we just perform bfs BUT begin by adding the nodes w/0 any indegrees to our q first because they will otherwise be unreachable

← Kahn's indegree (stated before)

```
while !q.empty    //BFS
    curr = q.poll
    sorted.add(curr)  //Final list
     for (int neighbors)
         if  not visited
             visit[neighbor] = true
             q.add(neighbor)
```