**Depth First Search**

*Definition:* Explore as far as possible down one branch before “backtracking” (choose the lowest item as our next item to visit)

*Pseudocode for Depth First Search:*

DFS\_recursive:

* The function should accept a starting node
* Create a list to store the end result, to be returned at the very end
* Create an object to store visited vertices
* Create a helper function which accepts a vertex
  + The helper function should return early if the vertex is empty
  + The helper function should place the vertex it accepts into the visited object and push that vertex into the result array
  + Loop over all of the values in the adjacent list for that vertex
  + If any of those values have not been visited, recursively invoke the helper function with that vertex
* Invoke the helper function with starting vertex
* Return the result array

DFS\_iterative:

* The function should accept a starting node
* Create a stack to help use keep track of vertices (use a list/array)
* Create a list to store the end result, to be returned at the very end
* Create an object to store the visited vertices
* Add the starting vertex to the stack, and mark it visited
* While the stack has something in it:
  + Pop the next vertex from the stack
  + If that vertex hasn’t been visited yet
    - Mark it as visited
    - Add it to the result list
    - Push all of its neighbors into the stack

*Solution for Depth First Search:*

function DFS\_recursive(start) {

const result = [];

const visited = {};

const adjacentList = this.adjacentList;

function dfs(vertex) {

if(!vertex) return null;

visited[vertex] = true;

result.push(vertex);

adjacentList[vertex].forEach(neighbor => {

if (!visited[neighbor]) {

return dfs(neighbor);

}

});

}

dfs(start);

return result;

}

function DFS\_iterative(start) {

const result = [];

const visited = {};

var currentVertex;

const stack = [start];

visited[start] = true;

while(stack.length) {

currentVertex = stack.pop();

result.push(currentVertex);

this.adjacentList[currentVertex].forEach(neighbor => {

if (!visited[neighbor]){

visited[neighbor] = true;

stack.push(neighbor);

}

});

}

return result;

}

**Breadth First Search**

*Definition:* Visit neighbors at current depth first

*Pseudocode for Breadth First Search:*

BFS:

* This function should accept a starting vertex
* Create a queue (you can use an array) and place the starting vertex in it
* Create an array to store the nodes visited
* Make the starting vertex as visited
* Loop as long as there is anything in the queue
* Remove the first vertex from the queue and push it into the array that stores nodes visited
* Loop over each vertex in the adjacency list for the vertex you are visiting
* If it is not inside the object that stores nodes visited, mark it as visited and enqueue that vertex
* Once you have finished looping, return the array of visited nodes

*Solution for Depth First Search:*

function BFS(start) {

const queue = [start];

const result = [];

const visited = {};

visited[start] = true;

var currentVertex;

while (queue.length) {

currentVertex = queue.shift();

result.push(currentVertex);

this.adjacentList[currentVertex].forEach(neighbor => {

if (!visited[neighbor]) {

visited[neighbor] = true;

queue.push(neighbor);

}

});

}

return result;

}