

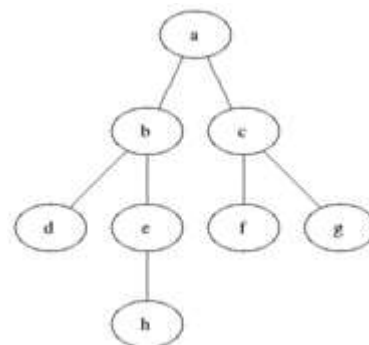
Breadth-First Search

So far, we've learned different ways of creating representations of graphs. What now? One natural question to have is what are the distances between any two nodes in the graph? Enter *graph traversal algorithms*.

Traversal algorithms are algorithms to traverse or visit nodes in a graph. One type of traversal algorithm is the breadth-first search algorithm.

This algorithm starts at one node and visits all its neighbors that are one edge away. It then goes on to visit each of their neighbors and so on until all nodes have been reached.

An important data structure that will help implement the breadth-first search algorithm is the queue. This is an array where you can add elements to one end and remove elements from the other end. This is also known as a *FIFO* or *First-In-First-Out* data structure.



Visually, this is what the algorithm is doing.

(visit URL [Data Structures: Breadth-First Search | freeCodeCamp.org](https://www.freecodecamp.org/data-structures/breadth-first-search))

The grey shading represents a node getting added into the queue and the black shading represents a node getting removed from the queue. See how every time a node gets removed from the queue (node turns black), all their neighbors get added into the queue (node turns grey).





To implement this algorithm, you'll need to input a graph structure and a node you want to start at.

First, you'll want to be aware of the distances from, or number of edges away from, the start node. You'll want to start all your distances with some large

number, like `Infinity`. This prevents counting issues for when a node may not be reachable from your start node. Next, you'll want to go from the start node to its neighbors. These neighbors are one edge away and at this point you should add one unit of distance to the distances you're keeping track of.

Write a function `bfs()` that takes an adjacency matrix graph (a two-dimensional array) and a node label `root` as parameters. The node label will just be the integer value of the node between `0` and `n - 1`, where `n` is the total number of nodes in the graph.

Your function will output a JavaScript object key-value pairs with the node and its distance from the root. If the node could not be reached, it should have a distance of `Infinity`.

	The input graph <code>[[0, 1, 0, 0], [1, 0, 1, 0], [0, 1, 0, 1], [0, 0, 1, 0]]</code> with a start node of <code>1</code> should return <code>{0: 1, 1: 0, 2: 1, 3: 2}</code>
	The input graph <code>[[0, 1, 0, 0], [1, 0, 1, 0], [0, 1, 0, 1], [0, 0, 0, 0]]</code> with a start node of <code>1</code> should return <code>{0: 1, 1: 0, 2: 1, 3: Infinity}</code>
	The input graph <code>[[0, 1, 0, 0], [1, 0, 1, 0], [0, 1, 0, 1], [0, 0, 1, 0]]</code> with a start node of <code>0</code> should return <code>{0: 1, 1: 1, 2: 2, 3: 3}</code>
	The input graph <code>[[0, 1], [1, 0]]</code> with a start node of <code>0</code> should return <code>{0: 0, 1: 1}</code>