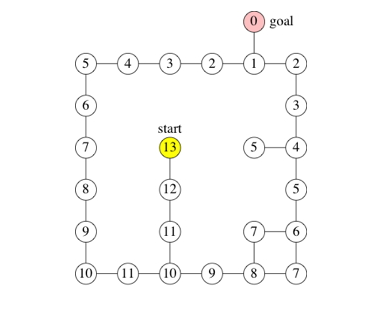
We wanna make the littel dude in the red circle reach his destination. The path he should take is actually a directional graph.



This is the route it has to take (imagine the background is a labyrinth with walls and shit.) The book tells us that this is the shortest route possible, through using breadth first search

Breadth first Search, or BFS, is an algorithm that finds the shortest path (with the fewest edges) in a graph between two vertices. Pretty cool.

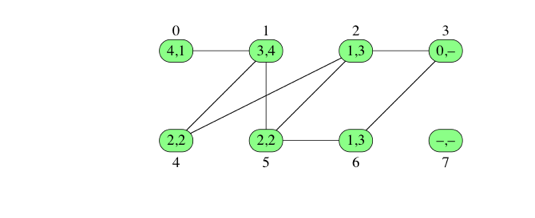
How does it work

The algorithm assigns two values to each vertex:

A **distance**, the minimum distance from the source vertex to this one (min number of edges)

A **predecessor** of this vertex along the shortest path from the source vertex. The source vertex has a predecessor value of null.

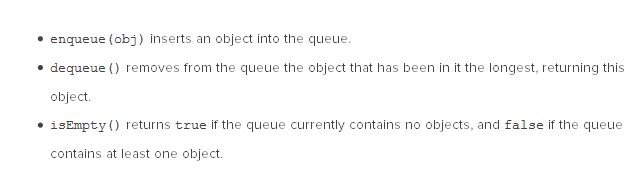
Basically, it shows the distance from the source, and the value of the previous vertex on the shortest path to source. In this example, we start at vertex 3 (that’s why its second parameter has a value of null). seven is not connected to 3 by any set of edges, so both its parameters are null.



As you guessed, we work recursively, starting from source. We start by giving the source a distance of 0 (it’s the base case). Then we go to all the vertices that are in the immediate neighborhood of the source and give them a distance of 1, a predecessor of source. Then we go to the vertices that are distance 1 from those vertices but only if they have not been visited before and give them a distance of 2. How do you know whether a vertex has been visited already? Well, that’s easy: if it hasn’t, its distance parameter is null, otherwise it’s a value.

But maybe we are looking for routes from several sources. Maybe we already used breadth search on another vertex, and now some vertices have received distance and predecessor values. In that case, we are going to use a **queue**, which is a data structure that allows us to insert and remove items, were the item removed is always the one that has been in the queue the longest. So it’s **First in, first out.**

A queue has three operations:



Here is how it works. First we enqueue() the source. From there, the source leads to two more vertices. We dequeue source and enqueue the two neighboring vertices. then we dequeue one of them and look for the vertices that are connected, but not visited (so the enqueued one wont be taken into account), then we dequeue the second one and enqueue its neighbors.

basically, you proceed step by step, vertex by vertex, and everytime you ‘launch’ yourself from a vertex to find its neighbors, you dequeue it, before enqueueing simultaneously the neighbours found.

When the queue is empty, breadth search terminates. You may wanna save the vertices as they go in the queue in another array perhaps to record the route taken.

