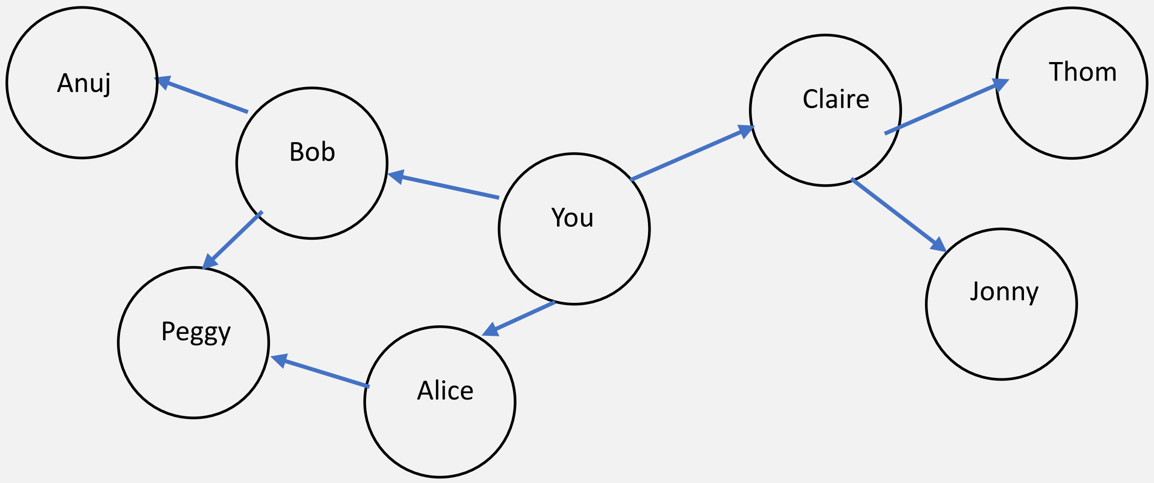
Chapter 6

Graph:

Graph is a way modeling connections for example when you are playing a card game the round can be represented using a graph like that player1 ---> player 2 -→player 3 and so the players are called nodes and the connections are called edges that's how a graph is built one node can be connected to many other nodes not only one the nodes that have a common edge are called neighbors also nodes doesn't have to have a common edge to be connected for example player1 is connected to player3 as you can get from player1 to player3 by passing through player2.

Breadth-first search

We have seen some search algorithms before but this one is a different type that runs on graphs. It answers two questions 1-is there a path from node A to B   2- what is the shortest path from node A to B let's have an example.



Let's suppose you have a mango farm, and you want to reach a seller to sell your mangoes so are you connected to a mango seller? The answer is easy first you'll make a list of all your direct friends and check each one of them if he is seller, you end your search if not you remove him/her from the list if you finished all your direct friends and none of them is a seller you begin checking friends of you friends how is that done once you check one of your friends and remove him/her from the list add all his friends to the list let's say you checked Alice so you remove her from the list then add Peggy to the end of the list

Finding the shortest path:

We are now able to determine if a path between two nodes exists or not. Let’s move to the other question, what is the shortest path to the node we found?

You’d prefer a first degree connection to a second degree connection to a third degree connection (first degree is your direct friend second degree is friend of your friend and third degree is the friend of a friend of your friend and so on) so you shouldn’t search any second degree connection before you finish all you first degree connections and fortunately this how breadth-first search works like we mentioned before you always add second degree connections to the end of the list so it’s guaranteed that you won’t check any second degree connection before first degree but you have to bear in mind that you have to check the nodes in the same order they were entered in in order not to search a second degree connection before a first degree one will there is a way to make sure that won’t happen.

Queues:

A queue works exactly like a real-life queue. When you add a new element, you add it to the end of the queue and when remove an element it’s removed from the beginning of the queue. This way we can make sure that nodes are checked in their correct order.

Implementing the graph

A graph is a relation between two nodes A->B and you happen to know a data structure that map relationships in a similar way in our case we want to map a node to its neighbors so we will make a hash table with a node as the key and an array of nodes as the value does it matter in which order we enter the nodes the answer is no because hash tables doesn’t have an order the order is important in the search that’s why we use the queue to ensure that elements entered first are checked first and that’s why while searching the graph make sure to insert the nodes to the queue in the right order.

Implementing the algorithm

You add all the first degree connections to the queue and check them one by one if one of them is a seller you are done if not pop the node and enqueue all its neighboring nodes and repeat you also need to mark the nodes that aren’t a seller to avoid adding them to the queue again and also to avoid entering an infinite loop so every time you check a node and it’s not a seller mark it so it’s not added to the queue again

Running time

If you search your entire network that means you followed all the edges so that’s O(number of edges) you also keep a queue of all the persons you search adding a person to queue is O(1) so adding all the people will be equal to number of vertices so that’s O(number of vertices) so the total is O(number of edges+ number of vertices) also written as O(V+E) and that’s all for this chapter.