Breadth-First Search (BFS)

1. Definition:
   You start at vertex v, go to all of its unvisited neighbours, and repeat for all of v's neighbours, and so on. BFS will find the shortest distance between 2 vertices.

2. Algorithm:

   1. Start at v. Visit v and mark it as visited.

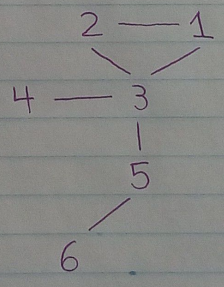
   2. Visit every unmarked neighbour of v and mark each neighbour as visited.

   3. Mark v as finished.

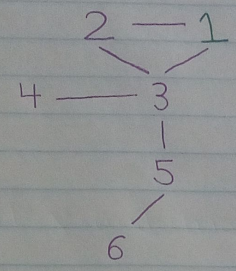
   4. Recurse on each vertex marked as visited in the order they were visited.

3. Examples:

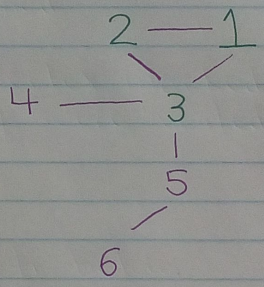
   1. Consider the graph below.



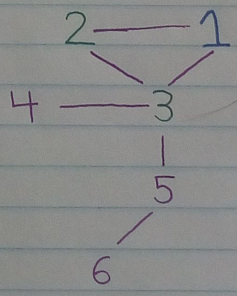
I will start at 1, and mark it visited by writing it in green.



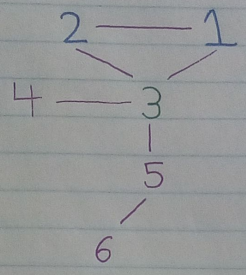
Next, I will visit all the unmarked neighbours of 1 in the following order: 2, 3.



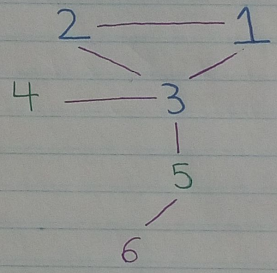
I will mark 1 as finished by writing it in blue.



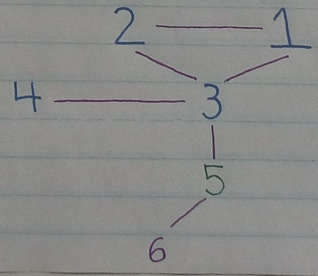
Since I visited 2 first, I will visit every unmarked neighbour of 2 and then mark 2 as finished.



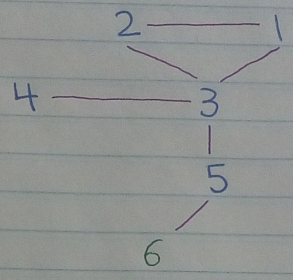
I will now visit every unmarked neighbour of 3 and then mark it as finished. I will visit 4, then 5.



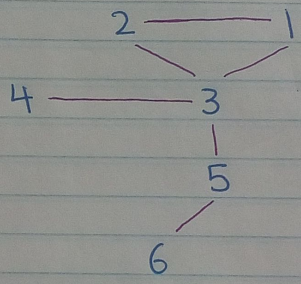
Since I visited 4 first, I will visit every unmarked neighbour and then mark 4 as finished.



I will visit all the unmarked neighbours of 5 and mark 5 as finished.



Finally, I will visit all the unmarked neighbours of 6 and mark 6 as finished.

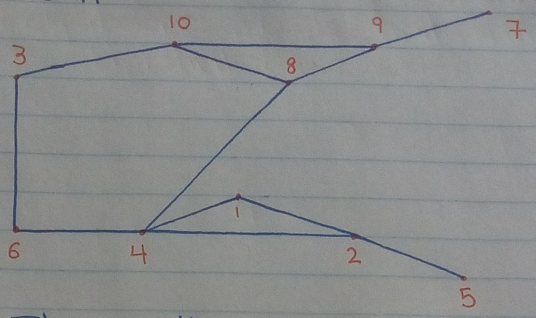


2. Given an adjacency list, draw the graph and do a BFS.

Adjacency List:

1: 2, 4, 10
2: 1, 4, 5, 9
3: 6, 10
4: 1, 2, 6, 8
5: 2
6: 3, 4
7: 9
8: 4, 9, 10
9: 2, 7, 8, 10
10: 1, 3, 8, 9

Soln:



This is the graph.

Now, to do a BFS using the graph and the adjacency list, I will use a queue to store the unmarked neighbours.

1. Starting at 1, I see that it has 3 neighbours 2, 4 and 10.

   Queue: 2, 4, 10
   Visited: 1

2. Visiting 2, I see that it has 4 neighbours, but I've already visited 1 and 4 is already in the queue, so I only add 5 and 9 to it. I will pop 2 now.

   Queue: ~~2~~, 4, 10, 5, 9
   Visited: 1, 2

4

Visiting , I add its unmarked neighbours, 6 and 8 to the queue. Then, I remove 4.

Queue: ~~2~~, ~~4~~, 10, 5, 9, 6, 8
Visted: 1, 2, 4

Visiting 10, I add 3 to the queue. Then, I remove 10.

Queue: ~~2~~, ~~4~~, ~~10~~, 5, 9, 6, 8, 3
Visited: 1, 2, 4, 10

Visiting 5, I don't add anything to the queue because I've already visited 2. I remove 5 from the queue.

Queue: ~~2~~, ~~4~~, ~~10~~, ~~5~~, 9, 6, 8, 3
Visited: 1, 2, 4, 10, 5

Visiting 9, I add 7 to the queue. Then, I remove 9 from the queue.

Queue: ~~2~~, ~~4~~, ~~10~~, ~~5~~, ~~9~~, 6, 8, 3, 7
Visited: 1, 2, 4, 10, 5, 9

Visiting 6, I don't add anything to the queue. Then, I remove 6 from the queue.

Queue: ~~2~~, ~~4~~, ~~10~~, ~~5~~, ~~9~~, ~~6~~, 8, 3, 7
Visited: 1, 2, 4, 10, 5, 9, 6

Visiting 8, I don't add anything to the queue. Then, I remove 8.

Queue: ~~2~~, ~~4~~, ~~10~~, ~~5~~, ~~9~~, ~~6~~, ~~8~~, 3, 7
Visited: 1, 2, 4, 10, 5, 9, 6, 8

Visiting 3, I don't add anything to the queue. Then, I remove 3.

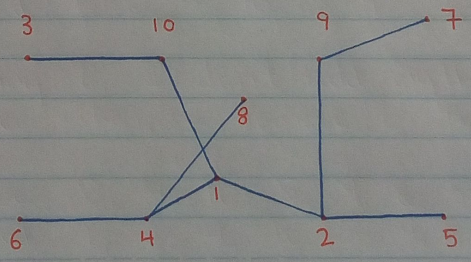
Queue: ~~2~~, ~~4~~, ~~10~~, ~~5~~, ~~9~~, ~~6~~, ~~8~~, ~~3~~, 7
Visited: 1, 2, 4, 10, 5, 9, 6, 8, 3

Visiting 7, I don't add anything to the queue. Then, I remove 7.

Queue: ~~2~~, ~~4~~, ~~10~~, ~~5~~, ~~9~~, ~~6~~, ~~8~~, ~~3~~, ~~7~~
Visited: 1, 2, 4, 10, 5, 9, 6, 8, 3, 7

Since there are no more items on the queue, we are finished.

Note: A BFS will produce a non-unique spanning tree. In our case, the tree looks like:



4. Information Given From A BFS

1. The shortest distance/path from any 2 vertices.
2. Whether the graph is connected.
3. The number of connected components.
4. A BFS constructs a spanning tree that visits every node connected to the starting node.

Note: Because a BFS follows from an adjacency list, the spanning tree is not unique.

Note: BFS is slow and takes a lot of work to get the solution.

5. Implementing BFS:

- We can use a queue (FIFO) to implement a BFS given an adjacency list representation of a graph.
- A queue has the following operations:
  1. Enqueue(Q,v)
  2. Dequeue(Q)
  3. Isempty(Q)

- Furthermore, we will need to store the following information for each node, v:

  1. The current node, u, and its state. I.e. Visited, Not Visited, Finished
  2. The predecessor, p[u].
  3. The distance from u to v.
  4. The order of discovery.

6. Complexity:

   - Since each node is enqueued at most once, the adjacency list of each node is examined at most once. Therefore, the total running time of BFS is O(m+n) or linear in the size of the adj list.
   - Each node is enqueued when it is not visited, at which point it is marked visited.

7. Other:

   - BFS will only visit the nodes that are reachable from the starting node.
   - If the graph is connected / strongly connected, then this is all nodes. If not, we may need to do BFS multiple times.