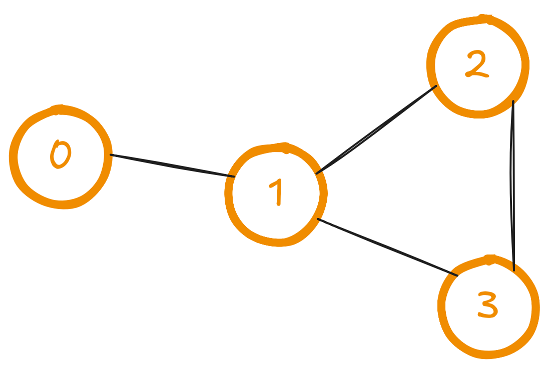
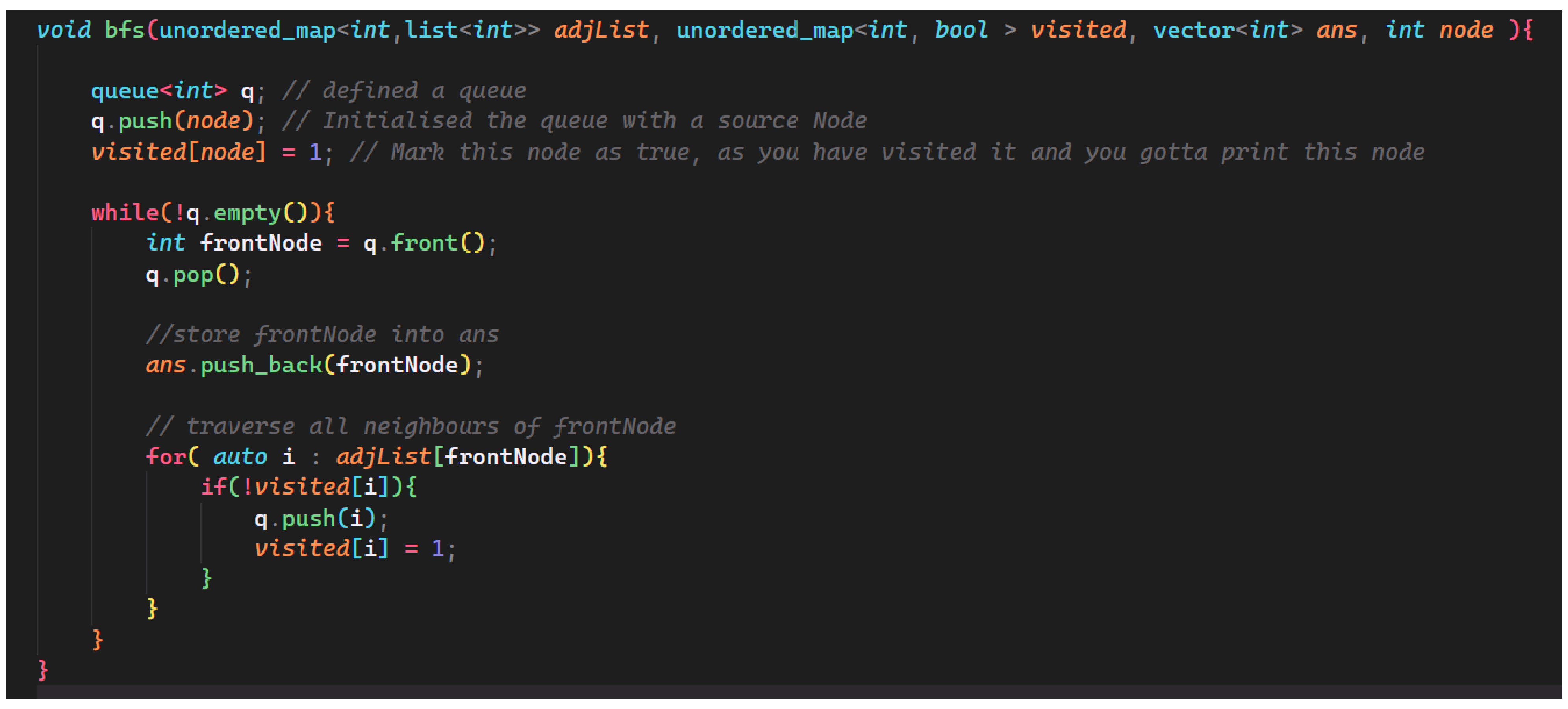
Breadth First Search



Adjacency List :
0 -> 1
1 -> 2,3
2-> 1,3
3 -> 1,2

You need to traverse the graph and print all the nodes in it.



```
void bfs(unordered_map<int,list<int>> adjList, unordered_map<int, bool > visited, vector<int> ans, int node ){

    queue<int> q; // defined a queue
    q.push(node); // Initialised the queue with a source Node
    visited[node] = 1; // Mark this node as true, as you have visited it and you gotta print this node

    while(!q.empty()){
        int frontNode = q.front();
        q.pop();

        //store frontNode into ans
        ans.push_back(frontNode);

        // traverse all neighbours of frontNode
        for( auto i : adjList[frontNode]){
            if(!visited[i]){
                q.push(i);
                visited[i] = 1;
            }
        }
    }
}
```

Visited

0	~~False~~ True
1	~~False~~ True
2	~~False~~ True
3	~~False~~ True

Explanation of the Code and Dry Run

- A queue is defined as q. $\boxed{\quad}$

- We push the source node into this $\boxed{0}$

- while(!q.empty()), this while loop would work till this queue becomes empty.
- frontNode is 0 and then we pop it to remove from the queue. $\boxed{\quad}$
- Answer array becomes $\boxed{0}$

- Now take a look at the adjacency list of 0, it is 1, auto i : adjList[frontNode], here i is the iterator of the elements of the list correspoding to the node, adjList[frontNode] is that particular list present in the index of frontNode and i is the element of that particular list.

- If we look at the visited table, 1 is still False. So push it into the queue and mark the 1 as visited

- The state of the queue is $\boxed{1}$

- We have completed the while loop, now the queue is still not empty, running the loop again.

- frontNode is 1 and then we pop this 1 out from the queue and queue becomes $\boxed{\quad}$

- we store this in the answer , Answer array becomes $\boxed{0\ 1}$

- refer to adjList[i] which is adjacency list of adjList[1] which is 2 , 3.
  - check 2, if it is visited in the list, No so add 2 in the queue
  - check 3, if it is visited in the list, No so add 3 in the queue.

  - The state of the queue is $\boxed{2\ \ 3}$

- Queue is not empty so doing the thing again for the queue , front Node is 2, store it and now pop it out from the queue, and add it into the answer
  - The state of the queue is $\boxed{3}$

  Answer array is $\boxed{0\ 1\ \ 2}$

- Now check, the adjacency list corresponding to this 2, it is 1 and 3.
- 1 is marked as True in the visiting list, so we do not push it again into the queue
- 3 is marked as True in the visiting list, so we do not push it again into the queue.

- The Queue is still not empty, it has 3 in it, so frontNode is 3 , pop it out from the Queue, and add it as answer.
  - The state of the queue is $\boxed{\quad}$

  Answer array is $\boxed{0\ 1\ \ 2\ \ 3}$

- for adjacency list[3], we got 1 and 2 , 1 is marked True, so we not push it into the queue, 2 is also marked true so we do not push it into the queue.

  - The state of the queue is $\boxed{\quad}$

  Answer array is $\boxed{0\ 1\ \ 2\ \ 3}$

_ The Queue has become empty now we can get out of the while loop, and return the answer array