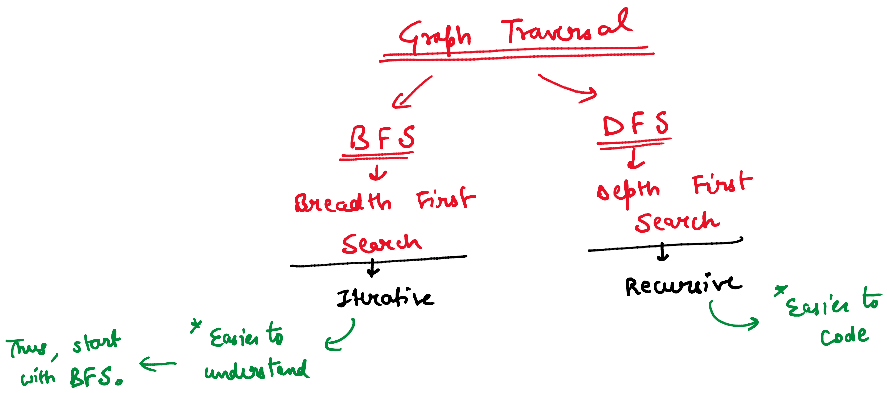
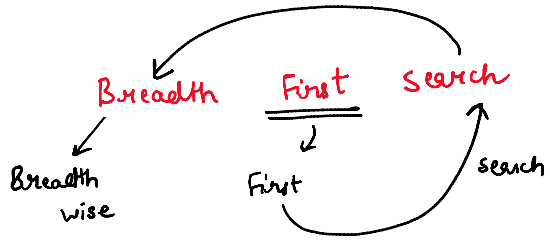
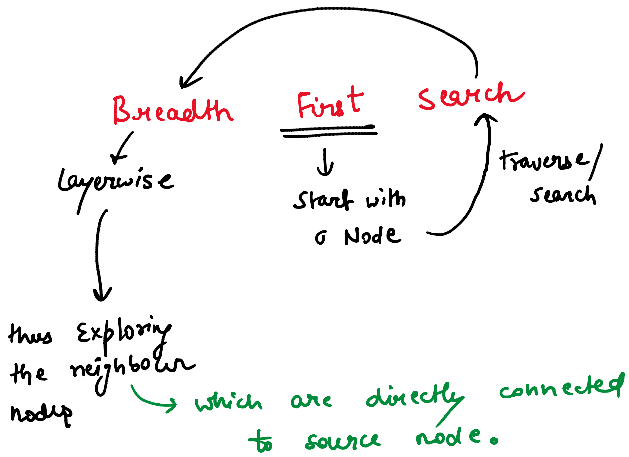
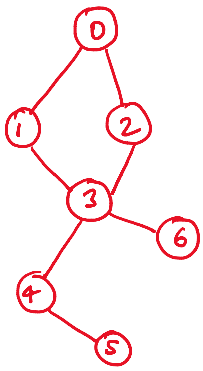
G-05 Breadth First Search (BFS) - Graph Traversal Technique - C++/Java

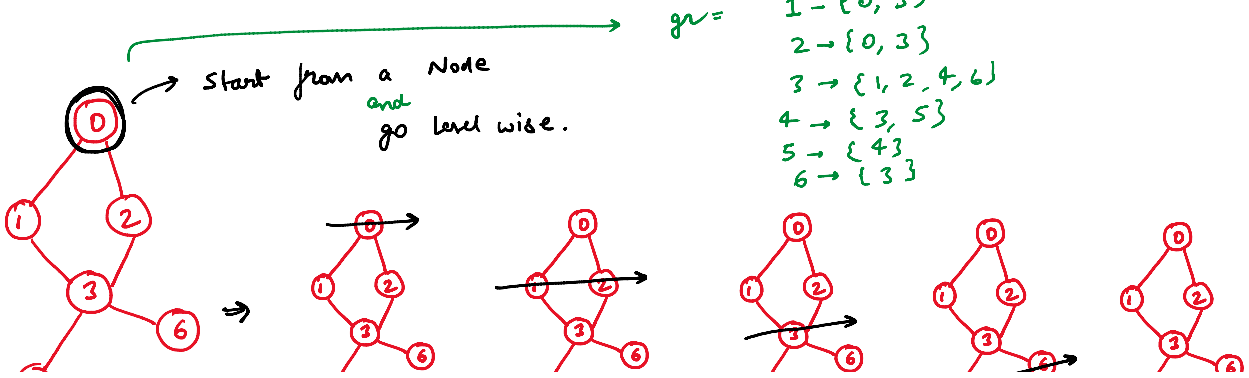
02 April 2023 08:45 PM

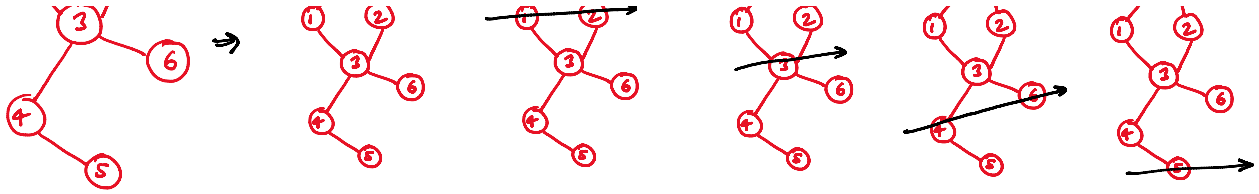




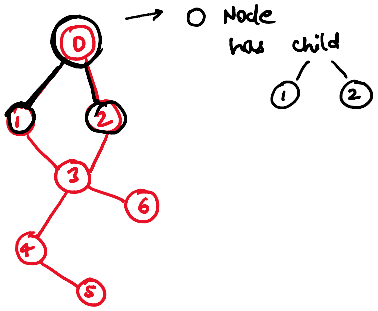








↳ Now, if we are standing at the start/root node.
we want to go to neighbour.



→ 0 Node has child 1 2

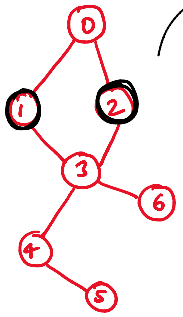
thus in adj list of

Neighbours of 0

0 → {1, 2}
1 → {0, 3}
2 → {0, 3}

↓

go to both of {1 and 2}
then simultaneously (Push in Queue & take from Front)
↳ we can use a Queue for this.



Now, we are at 1 & 2, we want to go to its Neighbours i.e. 3

1 → {0, 3}
2 → {0, 3}

But But But
Neighbour of 1 & 2 is 0 also

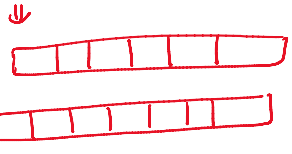
↓

But it was already visited right?

↓

Thus to monitor which Node was visited → we take a Visited Array

Thus, with visited array :: queue ::



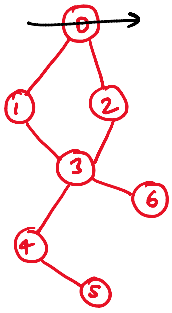
→ we can do BFS ☺

Dry Run Breadth First search :-

```cpp
// src - Source Node / Starting Node
// vector<int> vis(n, 0);
// vector<vector<int>> gr(n);
void bfsOfGraph(int src, vector<int>& vis, vector<vector<int>>& gr) {
    queue<int> q;
    q.push(src);

    vis[src] = 1;
    while (!q.empty()) {
        int cur = q.front();
        q.pop();

        for (int child : gr[cur]) {
            if (vis[child] == 0) {
                q.push(child);
                vis[child] = 1;
            }
        }
    }
}
```

src node is 0

while the queue has something

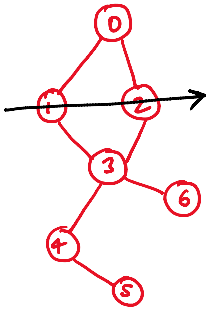


queue:- 0

visited : 0 1 2 3 4 5 6 — 1

remove it & push its child that are Not visited in queue

grab that value and Push in its child



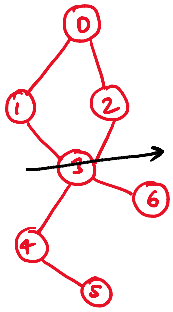
queue:- 1 2

visited : 0 1 2 3 4 5 6 — 1 1 1

```cpp
// src - Source Node / Starting Node
// vector<int> vis(n, 0);
// vector<vector<int>> gr(n);
void bfsOfGraph(int src, vector<int>& vis, vector<vector<int>>& gr) {
    queue<int> q;
    q.push(src);

    vis[src] = 1;
    while (!q.empty()) {
        int cur = q.front();
        q.pop();

        for (int child : gr[cur]) {
            if (vis[child] == 0) {
                q.push(child);
                vis[child] = 1;
            }
        }
    }
}
```

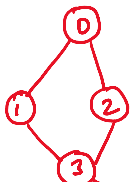
Pushing child only when child is NOT VISITED

also mark it visited



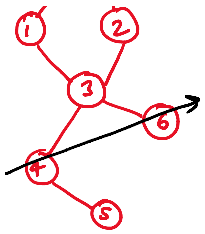
queue:- 3

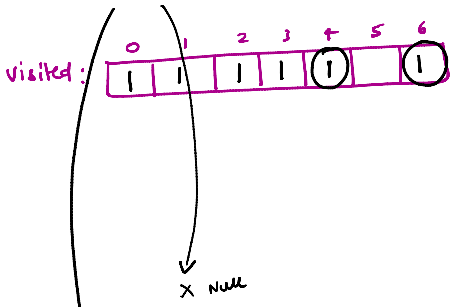
visited : 0 1 2 3 4 5 6 — 1 1 1 1

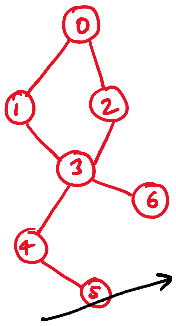


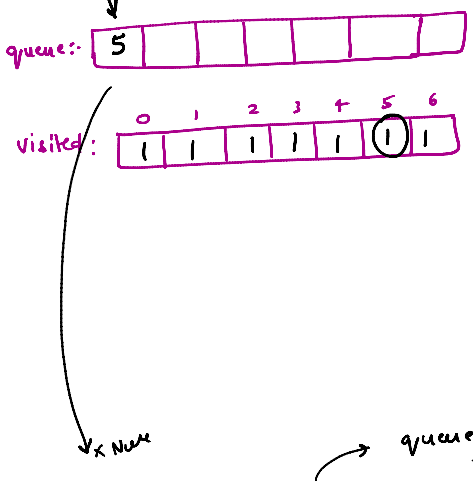
queue:- 4 6

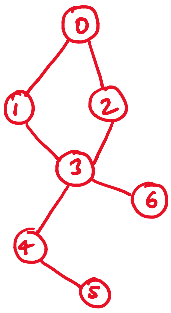
visited : 0 1 2 3 4 5 6 — 1 1 1 1 1 1

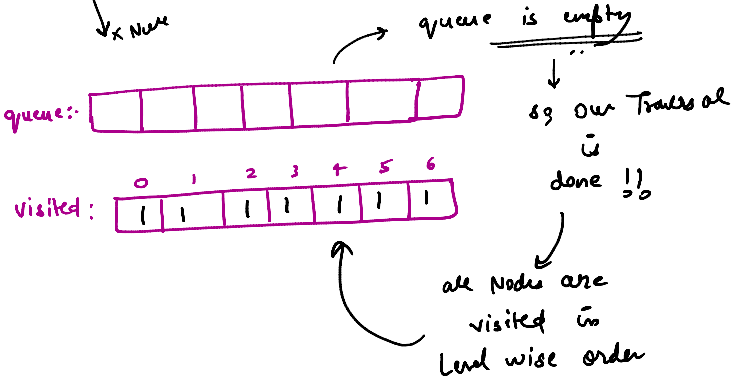












⊙ C++

```cpp
#include<bits/stdc++.h>
using namespace std;


// src - Source Node / Starting Node
// vector<int> vis(n, 0);
// vector<vector<int>> gr(n);
void bfsOfGraph(int src, vector<int>& vis, vector<vector<int>>& gr) {
    queue<int> q;
    q.push(src);

    vis[src] = 1;
    while (!q.empty()) {
        int cur = q.front();
        q.pop();

        for (int child : gr[cur]) {
            if (vis[child] == 0) {
                q.push(child);
                vis[child] = 1;
```

→ Have a queue to do levelwise traversal.

→ Push in queue & visit it.

→ Process until queue is empty / all elements are visited.

⊙ Java

```java
import java.util.*;

public class Main {
    // src - Source Node / Starting Node
    // vector<int> vis(n, 0);
    // vector<vector<int>> gr(n);
    public static void bfsOfGraph(int src, List<Integer> vis, List<List<Integer>> gr) {
        Queue<Integer> q = new LinkedList<>();
        q.add(src);

        vis.set(src, 1);
        while (!q.isEmpty()) {
            int cur = q.poll();

            for (int child : gr.get(cur)) {
                if (vis.get(child) == 0) {
                    q.add(child);
                    vis.set(child, 1);
                }
            }
        }
    }
```

```
        for (int child : gr[cur]) {
            if (vis[child] == 0) {
                q.push(child);
                vis[child] = 1;
            }
        }
    }
}

int main()
{
    int n, m;
    cin >> n >> m;

    // Graph Nodes are from 0 to n-1
    vector<vector<int>> gr(n);
    for (int i = 0; i < m; i++) {
        int u, v;
        cin >> u >> v;

        gr[u].push_back(v);
        gr[v].push_back(u);
    }

    vector<int> vis(n, 0);
    bfsOfGraph(0, vis, gr);
}
```

empty / all elements are visited.

goes children → if not visited → Push in queue.

→ vertices & edges

→ Initializing Undirected graph - gr

→ Having visited array.

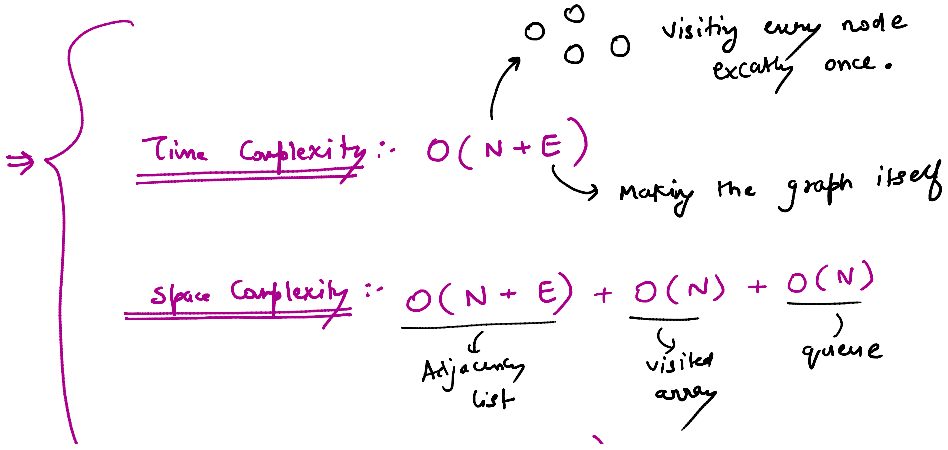
→ bfs function will do bfs.

```
                vis.set(child, 1);
            }
        }
    }
}
```

```
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int m = sc.nextInt();

        // Graph Nodes are from 0 to n-1
        List<List<Integer>> gr = new ArrayList<>();
        for (int i = 0; i < n; i++) {
            gr.add(new ArrayList<>());
        }

        for (int i = 0; i < m; i++) {
            int u = sc.nextInt();
            int v = sc.nextInt();

            gr.get(u).add(v);
            gr.get(v).add(u);
        }

        List<Integer> vis = new ArrayList<>();
        for (int i = 0; i < n; i++) {
            vis.add(0);
        }

        bfsOfGraph(0, vis, gr);
    }
}
```

```
#include<bits/stdc++.h>
using namespace std;


// src - Source Node / Starting Node
// vector<int> vis(n, 0);
// vector<vector<int>> gr(n);
void bfsOfGraph(int src, vector<int>& vis, vector<vector<int>>& gr) {
    queue<int> q;
    q.push(src);
    vis[src] = 1;
    while (!q.empty()) {
        int cur = q.front();
        q.pop();
        for (int child : gr[cur]) {
            if (vis[child] == 0) {
                q.push(child);
                vis[child] = 1;
            }
        }
    }
}
int main()
{
    int n, m;
    cin >> n >> m;
    // Graph Nodes are from 0 to n-1
    vector<vector<int>> gr(n);
    for (int i = 0; i < m; i++) {
        int u, v;
        cin >> u >> v;
        gr[u].push_back(v);
        gr[v].push_back(u);
    }
    vector<int> vis(n, 0);
    bfsOfGraph(0, vis, gr);
}
```

```
import java.util.*;
public class Main {
    // src - Source Node / Starting Node
    // vector<int> vis(n, 0);
    // vector<vector<int>> gr(n);
    public static void bfsOfGraph(int src, List<Integer> vis,
List<List<Integer>> gr) {
        Queue<Integer> q = new LinkedList<>();
        q.add(src);
        vis.set(src, 1);
        while (!q.isEmpty()) {
            int cur = q.poll();
            for (int child : gr.get(cur)) {
                if (vis.get(child) == 0) {
                    q.add(child);
                    vis.set(child, 1);
                }
            }
        }
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int m = sc.nextInt();
        // Graph Nodes are from 0 to n-1
        List<List<Integer>> gr = new ArrayList<>();
        for (int i = 0; i < n; i++) {
            gr.add(new ArrayList<>());
        }
        for (int i = 0; i < m; i++) {
            int u = sc.nextInt();
            int v = sc.nextInt();
            gr.get(u).add(v);
            gr.get(v).add(u);
        }
        List<Integer> vis = new ArrayList<>();
        for (int i = 0; i < n; i++) {
            vis.add(0);
        }
        bfsOfGraph(0, vis, gr);
    }
}
```



Adjacency list visited array queue

$= O(N+E)$

Questions for practice :- ⇒...

BFS problems

- Flood Fill: https://leetcode.com/problems/flood-fill/
- 01 Matrix: https://leetcode.com/problems/01-matrix/
- As Far from Land as Possible: https://leetcode.com/problems/as-far-from-land-as-possible/
- Rotting Oranges: https://leetcode.com/problems/rotting-oranges/
- Shortest Path in Binary Matrix: https://leetcode.com/problems/shortest-path-in-binary-matrix/
- Number of Islands: https://leetcode.com/problems/number-of-islands/
- Word Ladder I: https://leetcode.com/problems/word-ladder/
- Word Ladder II: https://leetcode.com/problems/word-ladder-ii/
- Evaluate Division: https://leetcode.com/problems/evaluate-division/
- Get Watched Videos by Your Friends: https://leetcode.com/problems/get-watched-videos-by-your-friends/
- Cut Off Trees for Golf Event: https://leetcode.com/problems/cut-off-trees-for-golf-event/