

```
for(int i = 0; i < V; i++){
    if(vis[i]) continue;
    boolean resOfThisComponent = bfs(graph, V, i, vis);
    if(resOfThisComponent) return true;
}
return false;
```

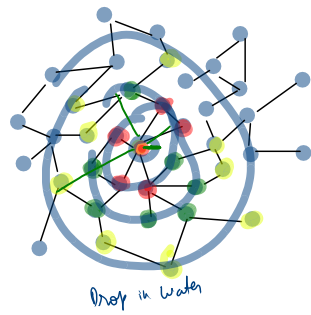
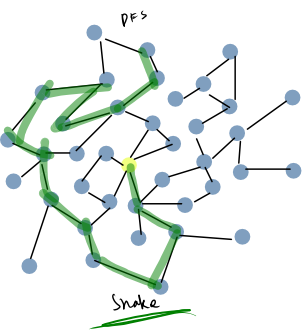
0 1
2 3
4 5
6 7

T T T T T T T

0 1 2 3 4 5 6 7

```
void bfs(int x) {
    // your code here
    Queue<Integer> q = new LinkedList<>();
    q.add(x);
    boolean[] visited = new boolean[vertices];
    //start dfs
    while(q.size() > 0){
        //remove
        int currNode = q.remove();
        //mark
        if(visited[currNode]) continue; //abstraction / cycle
        visited[currNode] = true;
        //work
        System.out.println(currNode+" ");
        //add unvisited nbrs
        for(int nbr:graph[currNode]){
            if(!visited[nbr]) q.add(nbr);
        }
    }
}
```

2 2



0 1

0 1 2 3

0: {1,3}

1: {0,2}

2: {3,4}

3: {4,5}

4: {5,6}

5: {6}

6: { }

0 1 2 3 4 5 6

T T T T T T T

2

3

Remove
Mark visited
Add unvisited

```
void bfs(int x) {
    // your code here
    Queue<Integer> q = new LinkedList<>();
    q.add(x);
    boolean[] visited = new boolean[vertices];
    //start dfs
    while(q.size() > 0){
        //remove
        int currNode = q.remove();
        //mark
        if(visited[currNode]) continue; //abstraction / cycle
        visited[currNode] = true;
        //work
        System.out.println(currNode+" ");
        //add unvisited nbrs
        for(int nbr:graph[currNode]){
            if(!visited[nbr]) q.add(nbr);
        }
    }
}
```

Graph → BFS → Queue
→ DFS → Recursion

1 Mark visited
2 Work
3 Visit unvisited nbr

0 1 2 3 4 5 6

0: { }

1: {0,2}

2: {1,3}

3: {2,4}

4: {3,5}

5: {4,6}

6: {5}

0 1 2 3 4 5 6

T T T T T T T

2 2

3



Graphs

Vertices/Node

Edges

Directed graph

Undirected graph

Cycle

Acyclic

Self Loop

Components

0 1 2 3 4 5 6

0: {1,3}

1: {0,2}

2: {3,4}

3: {4,5}

4: {5,6}

5: {6}

6: { }

0 1 2 3 4 5 6

T T T T T T T

2 2

3

