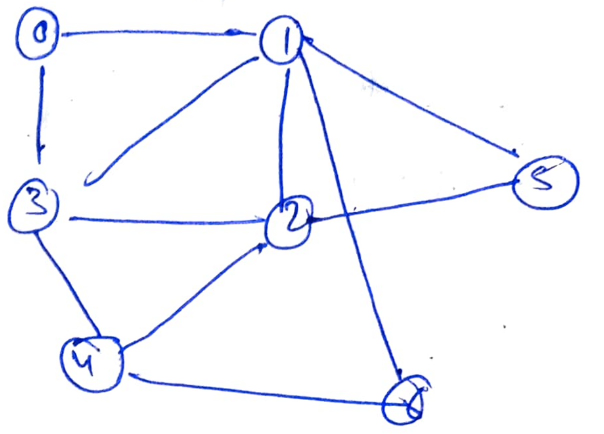
# LAB - 6   session 6   Application of BFS,

DFS, methods

## Pre lab

1. Give graph



BFS :-

starting point = 0

Queue

| | | | | | |
|---|---|---|---|---|---|
| 0 | | | | | |

visited

| | | | | | |
|---|---|---|---|---|---|
| 0 | | | | | |

add 0 to ~~the~~ visited and add adjacent nodes to queue

| | | | | | |
|---|---|---|---|---|---|
| 1 | 3 | | | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | | | | | | |

add 1 to visited and add adjacent nodes of 1 to queue

| | | | | |
|---|---|---|---|---|
| 3 | 2 | 6 | 5 | |

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 1 | | | | | |

| | | | | | |
|---|---|---|---|---|---|
| 2 | 6 | 5 | | | |

| | | | | | |
|---|---|---|---|---|---|
| 0 | 1 | 3 | | | |

add 2 to visited and add adjacent nodes to queue.

| | | | | | |
|---|---|---|---|---|---|
| 6 | 5 | 4 | | | |

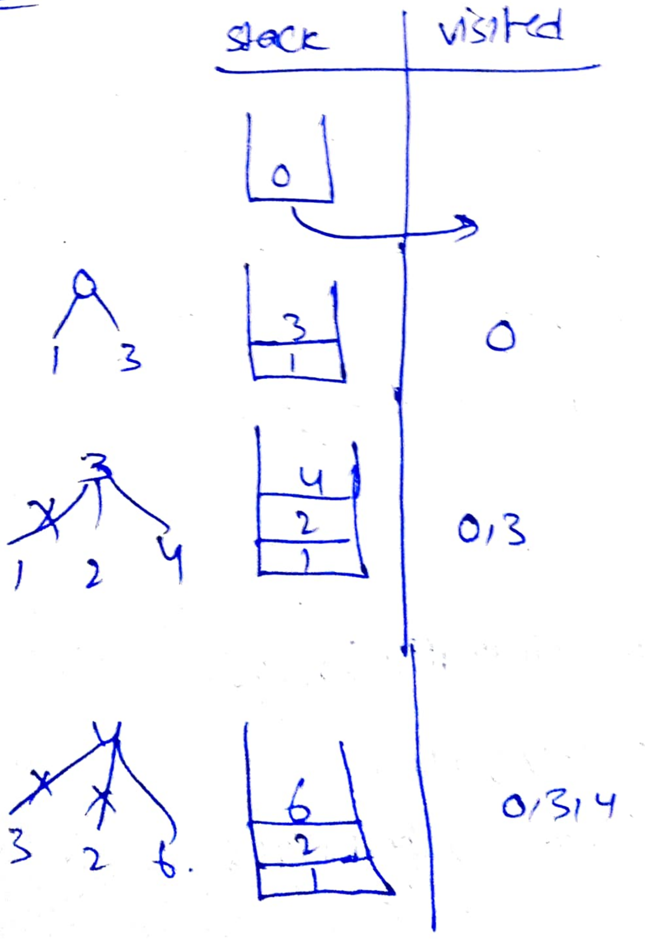
| | | | | | |
|---|---|---|---|---|---|
| 0 | 1 | 3 | 2 | | |

since all nodes are visited add all nodes in queue to visited.

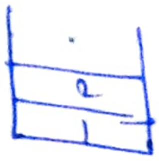
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 3 | 2 | 6 | 5 | 4 | |

BFS path is 0,1,3,2, 6,5,4

DFS

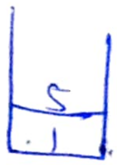


all adjacent nodes of '6' are visited

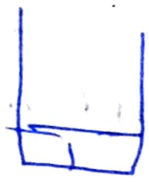


0,3,4,6

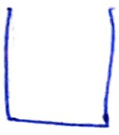
2
1
5



0,3,4,6,2



0,3,4,6,2,5



0,3,4,6,2,5,1

DFS path is 0,3,4,6,2,5,1

2)
```
package prelab2;
import java.util.*;
public class Graph {
        private int V;
        private LinkedList<Integer> adj[];
       Graph(int v) {
            V=v;
       adj = new LinkedList[v];
       for(int i=0; i<v; i++)
           adj[i]=new LinkedList();
}
```

```
void addEdge (int v, int w) {
        adj[v]. add(w);
}

void BFS (int s) {
    boolean visited[] = new boolean [V];
    LinkedList <Integer> queue = new LinkedList<>();
    visited [s] = true;
    queue . add(s);
    while (queue. size != 0) {
          s = queue . poll ();
          System. out. println( s+ " ")
          Iterator <Integer> i = adj[s]. listIterator
                                                  ();
          while (i. has~~e~~Next ()) {
               int n = i. next ();
               if(! visited [n]) {
                    visited [n] = true;
                    queue . add(n);
          }
       }
    }
  }
}

~~[illegible]~~ package pre lab2;
public class Demo {
      public static void main (String args[]) {
        Graph g = new  Graph(4);
            g.addEdge(0,1);
            g.addEdge(1,2);
            g.addEdge(2,0);
            g.addEdge(1,3);
            g.BFS(0).
    }
}
```

OUTPUT : [1,3]

AIn Lab2

```
1. package thlab1;
   import java.util.*;
   public class Demo {

       private static final int [] row = {-1,0,0,1};
                                 [] col = {0,-1,1,0};

     private static boolean isSafe(int [][] field
                          boolean visited[][], int x, int y){

       return (field[x][y] == 1 && !visited[x][y]);
     }
```

```
private static boolean isvalid (int u, int y, int m,
                                  int N) {

   return (x<M && y<N && x>=0 && y>=0);
 }

private static int BFS( int [][] field) {
      int M = field.length;
      int N = field[0].length;
     boolean[][] visited = new boolean[M][N];
    Queue<Node> q = new ArrayDeque<>();
    for( int r=0 ; r<m; r++) {
              if (field [r][0]==1) {
                   q.add(new node (r,0,0));
                   visited [r][0] = true;
          }
     }
   while (!q.isEmpty()) {

             int i = q.peek().u;
             int j = q.peek().y;
             int dist = q.peek().value;
             q.poll();
             if (j==N-1)
                      return dist;
             for (int k=0; k< raw.length; k++) {
                   q.add(new node (i+raw[k], j+col[k],
             }                         dist +1));
   }}
```

```
        return integer.MAX_VALUE;
}
public static void main(String args[]) {
    int [] [] field =
        {
            {0,1,1,1,0,1,1,1,1,1},
            {1,1,1,1,1,1,1,1,1,1},
            {1,1,1,1,1,1,1,1,0,1},
            {1,1,1,1,1,1,1,1,1,1},
            {1,1,1,1,1,0,1,1,1,1},
            {1,0,1,1,1,1,1,1,1,1},
            {1,1,1,1,1,1,1,1,1,0},
            {1,1,1,1,1,0,1,1,1,1},
            {1,1,1,1,1,1,1,1,1,1}
    };

  int dist =findShortest-Distance (field);
  if (dist != Integer MAX_VALUE) {
     System.out.println("shortest safe path" +dist);
  }
  else
      system.out.println("No route");
  }
}
```

```
package inLab1;
public class node {
        int  x, y, value;
        Node (int x, int y, int value) {
                this.x = x;
                this.y = y;
        }       this.value = value;
    }
```

output :- shortest safe path 11.

```
2. package  inlab2;
   public  class Demo {
          int l;
          int r;
          public  int size( TreeNode  node, int x) {
                if (node == null) {
                       return 0;
                }
              int ls = size(node.left, x);
              int rs =   size( node.right, x);
               if (node.val == x) {
                       l = ls;
                       r = rs;
               }
          int ts = ls + rs + 1;
   }      return ts;
Public   boolean btree GameWinningmove (TreeNode
                              root , int n , int v) {
      size( root, v);

    int  othersize - n - ( l+r+1)).
    int  max = Math.max(other size , Math.max(l,r))
   int   rest = n. Max ;
   if (max>rest)
          return  true;
   else
        return  false ;
  }
}
Package  inlab2 ;
Public   class  TreeNode {
          int  val;
          TreeNode  left ;
          TreeNode  right;
          TreeNode ( ){
           }
         TreeNode ( int  val ) {
               this. val = val ;
        }
        TreeNode ( int  val , TreeNode  left ,
                     TreeNode right ) {
          this. val = val;
          this. left = left;
          this. right = right;
    }
```

}

input    11   3

1 2 3 4 5 6 7 8 9 10 11

output   true

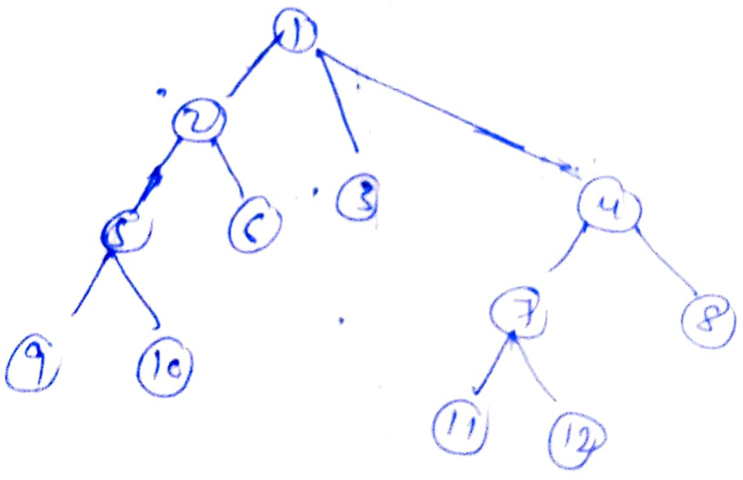
Post lab:

```
package Post lab1;
import java.util.*;
public class Graph {
    Private int v;
    Private ArrayList <ArrayList <Integer>> adj ;
   Graph(int v) {
      this.v = v;
      adj = new ArrayList < ArrayList <Integer>>(v);
      for (int i=0; i<v; i++)
           adj.add (new ArrayList <Integer>());
 }
void addEdge (int v, int w) {
   adj.get(v).add(w);
 }
void topological sort util (int v, boolean visited [],
               stack <Integer> stack) {
   visited [v] = true;
   Integer;
   Iterator <Integer> it = adj.get(v).iterator();
```

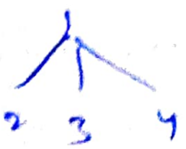
```
        while( it.hasNext()) {
              i=it.next();
              if (!visited [i])
                    topologicalsort-util (i, visited, stack);
           }
           stack.push (new Integer(v));
     }

   void  topological sort () {
         Stack<Integer> stack = new Stack<Integer>(
         boolean visited[] = new boolean[v];
         for (int i=0; i<v; i++)
                visited [i]= false;
         for (int i=0; i<v; i++) {
               topological sortutil (i, visited, stack);
         while (stack.empty()== false)
              System.out.Println( stack.pop() + "  "
         }
   }

package RSI-lab1;

public class Demo {

      public static void main(String args[]){
```

```
Graph g = new Graph(6);
    g.addEdge(5,2);
    g.addEdge(5,0);
    g.addEdge(4,0);
    g.addEdge(4,1);
    g.addEdge(2,3);
    g.addEdge(3,1);
  System.out.println("Topological sort");
    g.topologicalSort();
  }
}
```

2. given tree



BFS

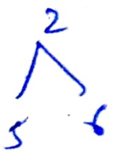
| | queue | visited |
|---|---|---|
|  | 1 | |
| | 2,3,4 | 1 |
| | 3,4,5,6 | 1,2 |

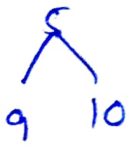
| | | |
|---|---|---|
| 3 | 4,5,6 | 1,2,3. |
| $$\begin{tikzcd}[column sep=small] & 4 \arrow[dl, dash] \arrow[dr, dash] & \\ 7 & & 8\end{tikzcd}$$ | 5,6,7,8 | 1,2,3,4 |
| $$\begin{tikzcd}[column sep=small] & 5 \arrow[dl, dash] \arrow[dr, dash] & \\ 1 & & 10\end{tikzcd}$$ | 6,7,8,9,10 | 1,2,3,4,5 |
| | 7,8,9,10 | 1,2,3,4,5,6 |
| | 8,9,10,11,12 | 1,2,3,4,5,6,7 |
| | 9,10,11,12 | 1,2,3,4,5,6,7,8 |
| | 10,11,12 | 1,2,3,4,5,6,7,8,9 |
| | 11,12 | 1,2,3,4,5,6,7,8,9,10 |
| | 12 | 1,2,3,4,5,6,7,8,9,11 |
| | | 1,2,3,4,5,6,7,8,9,10,11,12 |

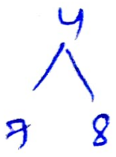
BFS path 1,2,3,4,5,6,7,8,9,10,11,12

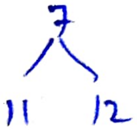
## DFS

| stack | visited |
|---|---|
| 1 | |
| 2,3,4 | 1 |
| 5,6,3,4 | 1,2 |
| 9,10,6,3,4 | 1,2,5 |
| 6,3,4 | 1,2,5,9,10 |
| 4 | 1,2,5,9,10,6,3 |
| 7,8 | 1,2,5,9,10, 6,3,4 |
| 11,12,8 | 1,2,5,9,10,6,3,4,7 |
| 12,8 | 1,2,5,9,10,6, 3,4,7,11 |
| 8 | 1,2,5,9,10,6,3,4,7,11,12 |
| | 1,2,5,9,10,6,3,4,7,11,12,8 |









DFS path 1,2,5,9,10,6,3,4,7,11,12,8