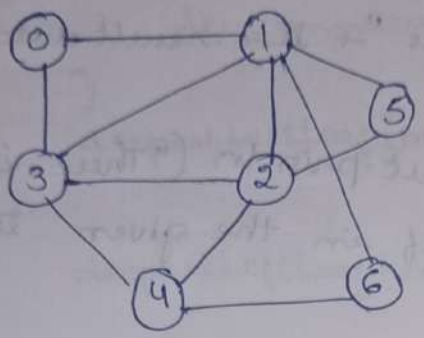
LAB SESSION - 6

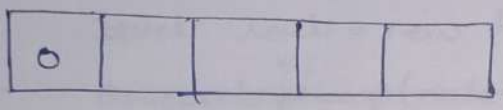
Pre-lab

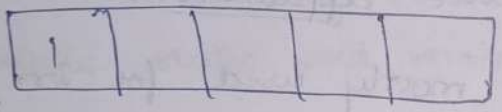
1.



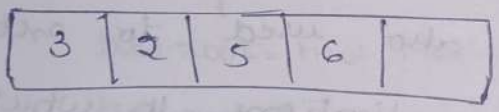
BFS : level order traversal

Consider Queue:

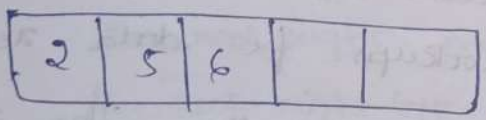




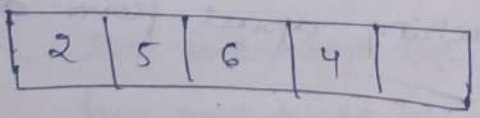
visited - 0

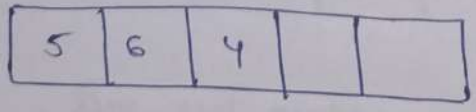


visited - 0, 1

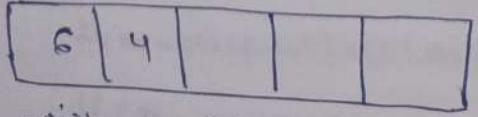


visited - 0, 1, 3

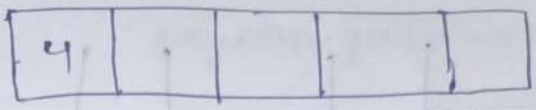




visited - 0, 1, 3, 2



visited - 0, 1, 3, 2, 5

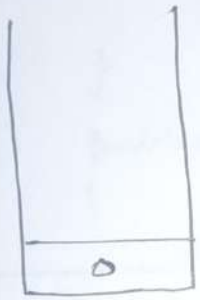


visited - 0, 1, 3, 2, 5, 6

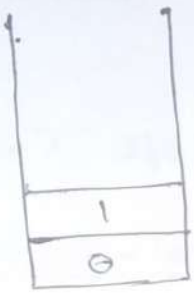
BFS: 0, 1, 3, 2, 5, 6, 4

DFS:

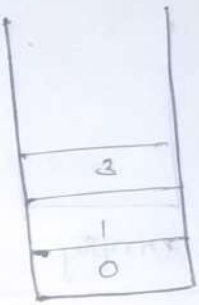
Consider stack.



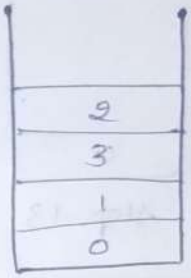
Step-1



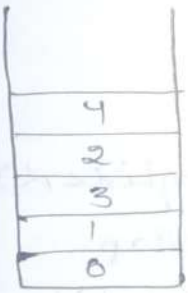
Step-2



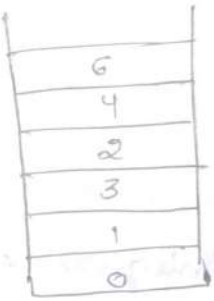
Step-3



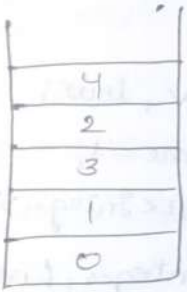
Step-4



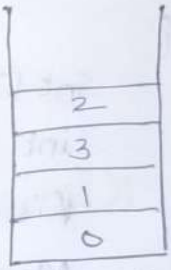
Step-5



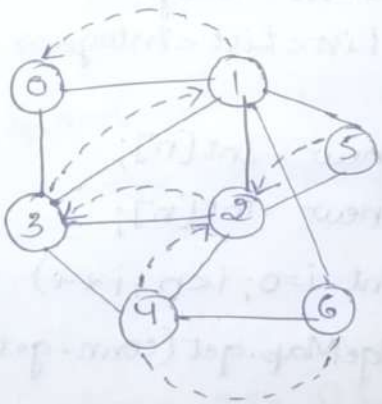
Step-6

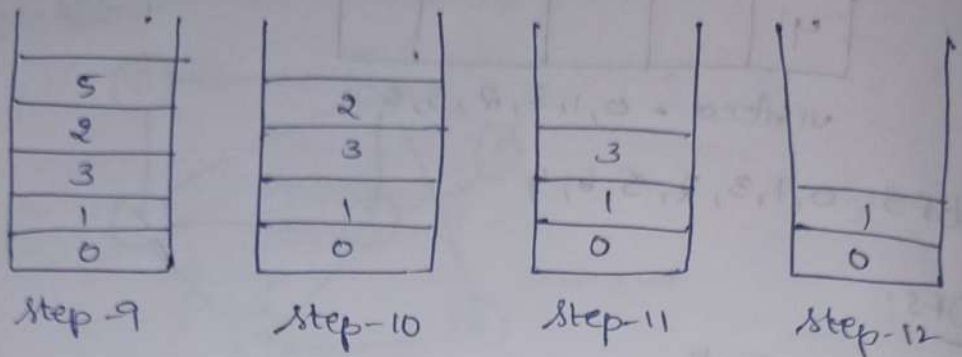


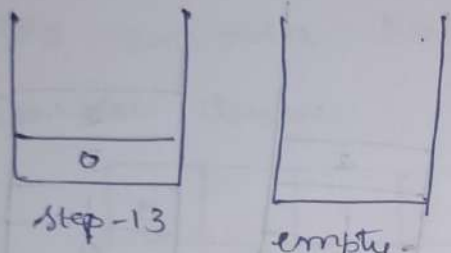
Step-7



Step-8







2. ~~DFS~~

```
class Solution
{
    int [] disc, low;
    int time = 1;
    List<List<Integer>> ans = new ArrayList<>();
    Map<Integer, List<Integer>> edgeMap
                = new HashMap<>();

    public List<List<Integer>> criticalConnections
        (int n, List<List<Integer>> connections)
    {
        disc = new int[n];
        low = new int[n];
        for (int i=0; i<n; i++)
            edgeMap.get(conn.get(0)).add
                (conn.get(1));

        for(List<Integer> conn: connections)
        {
```

```
            edgeMap.get(conn.get(0)).
                                add(conn.get(1));
            edgeMap.get(conn.get(1)).
                                add(conn.get(0));
        }
        dfs(0, -1);
        return ans;
    }
    public void dfs(int curr, int prev)
    {
        disc[curr] = low[curr] = time++;
        for (int next : edgeMap.get(curr))
        {
            if (disc[next] == 0) {
                dfs(next, curr);
                low[curr] = Math.min
                    (low[curr], low[next]);
            }
            else if (next != prev)
                low[curr] = Math.min(low[curr],
                                disc[next]);
            if(low[next] > disc[curr])
                ans.add(Arrays.asList(curr, next));
        }
    }
}
```

Input: 4 [[0,1],[1,2],[2,0],[1,3]]

Output: [[1,3]]

In-lab

1.

```
package Lab6;
public class Node {
    int x,y, dist;
    Node(int x, int y, int dist){
        this.x = x;
        this.y = y;
        this.dist = dist;
    }
}
```

```
package Lab6;
import java.util.*;
public class Main {
    private static final int[] row = {-1,0,0,1};
    private static final int[] col = {0,-1,1,0};
    private static boolean isSafe(int[][] field,
        boolean visited[][], int x, int y){
        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().x;
        int j = q.peek().y;
        int dist = q.peek().dist;
        q.poll();
        if (j == N-1) {
            return dist;
        }
        for (int k=0; k<row.length; k++) {
            if (isValid(i+row[k], j+col[k], M, N)
                && isSafe(field, visited, i+row[k], j
                    +col[k]))
            {
                visited[i+row[k]][j+col[k]] = true;
                q.add(new Node(i+row[k], j+
                    col[k], dist+1));
            }
        }
        return Integer.MAX_VALUE;
    }
public static int findShortestDistance(int[][]
                                    mat){
    if (mat==null || mat.length==0)
        return 0;
    int M = mat.length;
    int N = mat[0].length;
    int[] r = {-1,-1,-1,0,0,1,1,1};
    int[] c = {-1,0,1,-1,1,-1,0,1};
```

```
        for(int i=0; i<M; i++)
            for(int j=0; j<N; j++)
                if(mat[i][j] == Integer.MAX_VALUE
                    mat[i][j] = 0;

        return BFS(mat);

    }
    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, 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, 1, 0},
                { 1, 1, 1, 1, 1, 0, 1, 1, 1, 1},
                { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}
            };
        int dist = findShortestDistance(field);
        if(dist != Integer.MAX_VALUE)
            System.out.println("The shortest safe
                path has length of "+dist);
        else
        {
```

```
        - System.out.println("No route is safe
                to reach destination");
        }
    }
}
```

2.
```
class Solution
{
    public boolean btreeGameWinningMove
            (TreeNode root, int n, int x){
        count(root, x);
        return Math.max(Math.max(leftCount,
            rightCount), n-leftCount-rightCount-1)>n/2;
    }
    private int leftCount;
    private int rightCount;
    private int count(TreeNode root, int x)
    {
        if(root==null)
            return 0;
        final int l=count(root.left, x);
        final int r=count(root.right, x);
        if(root.val==x){
            leftCount=l;
            rightCount=r;
        }
        return 1+l+r;
    }
}
```

Input: {1,2,3,4,5,6,7,8,9,10,11} 11

Output: true

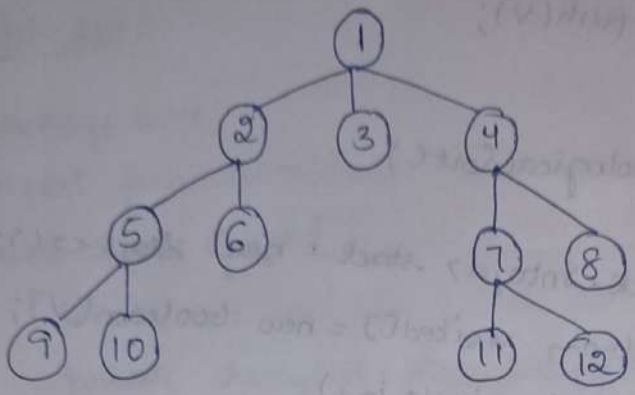
Post-lab:

1.
```
package Lab6;
import java.util.*;
public class Graph{
    private int V;
    private ArrayList<ArrayList<Integer>> adj;
    Graph(int V)
    {
        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 topologicalSortUtil(int v, boolean visited[],
        Stack<Integer> stack)
    {
        visited[v] = true;
        Integer i;
        Iterator<Integer> it =adj.get(v).
                                iterator();
        while (it.hasNext()){
            i=it.next();
            if(!visited[i])
                topologicalSortUtil(i, visited,
                                    stack);
        }
```

```
        stack.push(v);

    }
    void topologicalSort()
    {
        Stack<Integer> stack = new Stack<>();
        boolean visited[] = new boolean[V];
        for(int i=0;i<V; i++)
            visited[i] = false;
        for(int i=0; i<V; i++)
            if(visited[i] == false)
                topologicalSortUtil(i, visited, stack);
        while(stack.empty()==false)
            System.out.println(stack.pop() + " ");
    }
    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("Following is a
        Topological sort of given graph");
        g.topologicalSort();
    }
}
```

2.



BFS:

Consider queue:

1				

2	3	4		

visited - 1

3	4	5	6	

visited - 1, 2

4	5	6		

visited - 1, 2, 3

5	6	7	8	

visited - 1, 2, 3, 4

6	7	8	9	10

visited - 1, 2, 3, 4, 5

7	8	9	10	

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

8	9	10	11	12

visited - 1, 2, 3, 4, 5, 6, 7

9	10	11	12	

visited - 1, 2, 3, 4, 5, 6, 7, 8

10	11	12		

visited - 1, 2, 3, 4, 5, 6, 7, 8, 9

11	12			

visited - 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

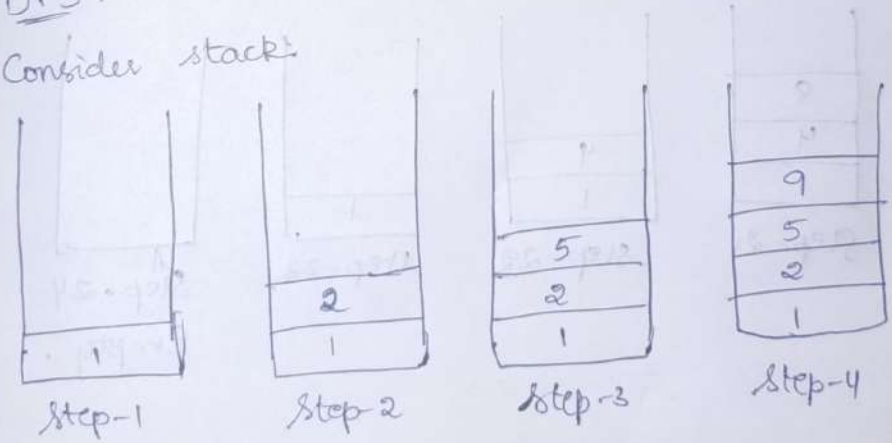
12				

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

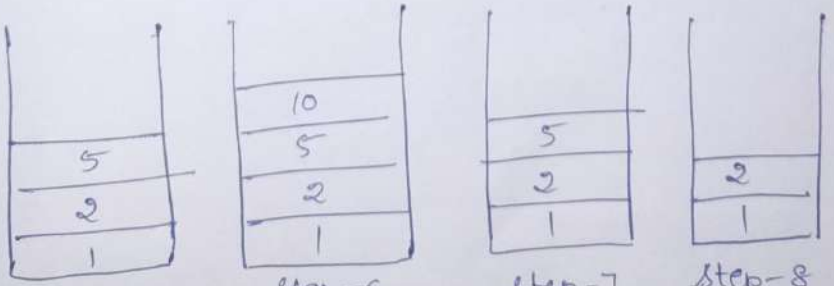
BFS: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12.

DFS:

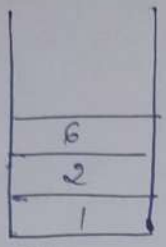
Consider stack:



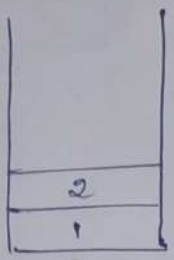
Step-1 Step-2 Step-3 Step-4



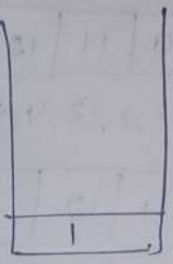
Step-5 Step-6 Step-7 Step-8



Step-9

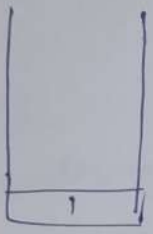


Step-10

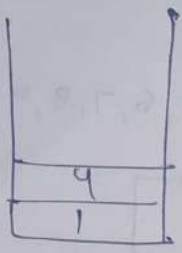


Step-11

Step-12

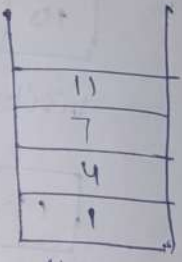


Step-13

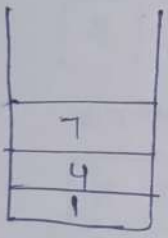


Step-14

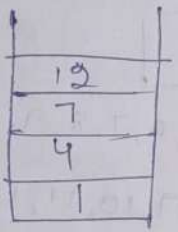
Step-15



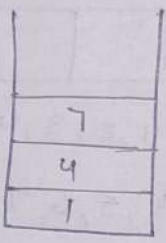
Step-16



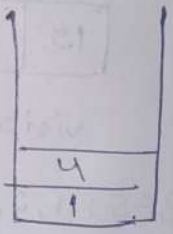
Step-17



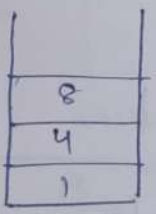
Step-18



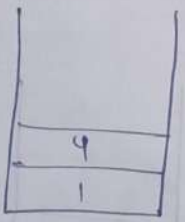
step-19



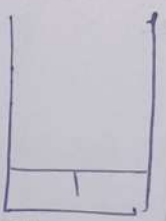
Step-20



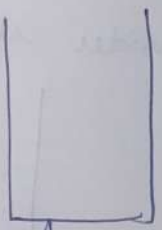
Step-21



step-22



Step-23


Step-24
Empty.