GRAPH DATA STRUCTURE

Content

10.1. BFS of graph

Statement : Given a directed graph. The task is to do Depth First Search of this graph.

```
static void bfs(int src, ArrayList<ArrayList<Integer>> list, boolean vis[]){
        LinkedList<Integer> queue = new LinkedList<>();
        queue.add(src);
        vis[src] = true;
        while(!queue.isEmpty()){
                int s = queue.poll();
                System.out.print(s+" ");
                ArrayList<Integer> temp = list.get(s);
                for(int i=0;i<temp.size();i++){
                        int n = temp.get(i);
                        if(!vis[n]){
                                vis[n] = true;
                                queue.add(n);
                        }
                }
        }
}
```

10.2. DFS of Graph

Statement : Given a directed graph. The task is to do Breadth First Search of this graph.

```
static void dfs(int src, ArrayList<ArrayList<Integer>> list, boolean vis[]){
        vis[src] = true;
        System.out.print(src+" ");
        ArrayList<Integer> temp = list.get(src);
        for(int i=0;i<temp.size();i++){
                int n = temp.get(i);
                if(!vis[n]){
                        dfs(n,list,vis);
                }
        }
}
```

10.3. Topological Sort

Statement : Given a Directed Graph. Find any Topological Sorting of that Graph.

```
static void topoSortUtil(int n,ArrayList<ArrayList<Integer>> list, Boolean vis[],
Stack<Integer> stack){
        vis[n] = true;
        int no;
        ArrayList<Integer> adjacencyList = list.get(n);
        for(int i=0;i<adjacencyList.size();i++){
                no = adjacencyList.get(i);
                if(!vis[no]){
                        topoSortUtil(no,list,vis,stack);
                }
        }
        stack.push(n);
}
static int[] topoSort(ArrayList<ArrayList<Integer>> list, int N){
        Stack<Integer> stack = new Stack<>();
        boolean vis[] = new boolean[list.size()];
        for(int i=0;i<list.size();i++){
                vis[i] = false;
        }
        for(int i=0;i<list.size();i++){
                if(!vis[i]){
                        topoSortUtil(i,list,vis,stack);
                }
        }
        int res[] = new int[list.size()];
        int i = 0;
        while(!stack.empty()){
                res[i++] = stack.pop();
        }
        return res;
}
```

10.4. Detect cycle in an undirected graph O(V+E)

Statement : Given a Undirected Graph. Check whether it contains a cycle or not.

```
static boolean isCyclicUtil(int n,ArrayList<ArrayList<Integer>> list,boolean vis[],int
parent){
        vis[n] = true;
        ArrayList<Integer> temp = list.get(n);
```

```
        for(int i=0;i<temp.size();i++){
                int d = temp.get(i);
                if(!vis[d]){
                        if(isCyclicUtil(d,list,vis,n)){
                                return true;
                        }
                }else if(d!=parent){
                        return true;
                }
        }
        return false;
}
static boolean isCyclic(ArrayList<ArrayList<Integer>> list, int V){
        boolean vis[] = new boolean[V];
        for(int i=0;i<V;i++)
                vis[i] = false;
        for(int i=0;i<V;i++){
                if(!vis[i]){
                        if(isCyclicUtil(i,list,vis,-1)){
                                return true;
                        }
                }
        }
        return false;
}
```

10.5. Detect cycle in a directed graph

Statement : Given a Directed Graph. Check whether it contains any cycle or not.\

```
static boolean isCyclicUtil(int i,ArrayList<ArrayList<Integer>> list,boolean
vis[],boolean recStack[]){
        if(recStack[i]){
                return true;
        }
        if(vis[i]){
                return false;
        }
        vis[i] = true;
        recStack[i] = true;
```

```
        ArrayList<Integer> l = list.get(i);
        for(int j=0;j<l.size();j++){
                if(isCyclicUtil(l.get(j), list, vis, recStack )){
                        return true;
                }
        }

        recStack[i] = false;
        return false;
}
static boolean isCyclic(ArrayList<ArrayList<Integer>> list, int V){
        boolean vis[] = new boolean[V];
        boolean recStack[] = new boolean[V];
        for(int i=0;i<V;i++){
                vis[i] = false;
                recStack[i] = false;
        }
        for(int i=0;i<V;i++){
                if(isCyclicUtil(i,list,vis,recStack)){
                        return true;
                }
        }
        return false;
}
```

10.6. Find the number of islands

Statement : Given a Matrix consisting of 0s and 1s. Find the number of islands of connected 1s present in the matrix.
Note: A 1 is said to be connected if it has another 1 around it (either of the 8 directions).

```
// A function to check if a given cell (row, col) can be included in DFS
boolean isSafe(int M[][], int row, int col,
                boolean visited[][]){
        // row number is in range, column number is in range and value is 1 and not yet
        //visited
        return (row >= 0) && (row < ROW) && (col >= 0) && (col < COL)        &&
(M[row][col] == 1 && !visited[row][col]);

}
```

```
// A utility function to do DFS for a 2D boolean matrix.
// It only considers the 8 neighbors as adjacent vertices
void DFS(int M[][], int row, int col, boolean visited[][]){
        // These arrays are used to get row and column numbers
        // of 8 neighbors of a given cell
        int rowNbr[] = new int[] { -1, -1, -1, 0, 0, 1, 1, 1 };
        int colNbr[] = new int[] { -1, 0, 1, -1, 1, -1, 0, 1 };

        // Mark this cell as visited
        visited[row][col] = true;

        // Recur for all connected neighbours
        for (int k = 0; k < 8; ++k)
                if (isSafe(M, row + rowNbr[k], col + colNbr[k], visited))
                        DFS(M, row + rowNbr[k], col + colNbr[k], visited);
        }

        int countIslands(int M[][]){
                // Make a bool array to mark visited cells.
                // Initially all cells are unvisited
                boolean visited[][] = new boolean[ROW][COL];

                // Initialize count as 0 and travese through the all cells of given matrix
                int count = 0;
                for (int i = 0; i < ROW; ++i)
                        for (int j = 0; j < COL; ++j)
                                if (M[i][j] == 1 && !visited[i][j]) // If a cell with
                                { // value 1 is not
                                        // visited yet, then new island found, Visit all
                                        // cells in this island and increment island count
                                        DFS(M, i, j, visited);
                                        ++count;
                                }

                return count;
        }
}
```

10.7. Implementing Dijkstra (Adjacency Matrix)

Statement : Given a graph of **V** nodes represented in the form of the adjacency matrix. The task is to find the shortest distance of all the vertex's from the source vertex.

```
class Implementation{
	private static int minDis(int dis[],boolean vis[], int V){
		int min = Integer.MAX_VALUE;
		int min_index = -1;
		for(int i=0;i<V;i++){
			if(vis[i]==false && dis[i]<min){
				min = dis[i];
				min_index = i;
			}
		}
		return min_index;
	}
	static void dijkstra(ArrayList<ArrayList<Integer>> list, int src, int V){
		int dis[] = new int[V];
		boolean vis[] = new boolean[V];
		for(int i=0;i<V;i++){
			dis[i] = Integer.MAX_VALUE;
			vis[i] = false;
		}
		dis[src] = 0;
		for(int i=0;i<V-1;i++){
			int u = minDis(dis,vis,V);
			vis[u] = true;
			ArrayList<Integer> temp = list.get(u);
			for(int v=0;v<V;v++){
				if(temp.get(v)!=0 && vis[v]==false && dis[u]+temp.get(v) <=
dis[v]){
					dis[v] = dis[u]+temp.get(v);
				}
			}
		}
		for(int i=0;i<V;i++){
			System.out.print(dis[i]+" ");
		}
	}
}
```

10.8. Find whether path exist

Statement : Given a **N** X **N** matrix (**M**) filled with 1, 0, 2, 3. The task is to find whether there is a path possible from source to destination, while traversing through blank cells only. You can traverse up, down, right and left.

- A value of cell **1** means Source.
- A value of cell **2** means Destination.
- A value of cell **3** means Blank cell.
- A value of cell **0** means Blank Wall.

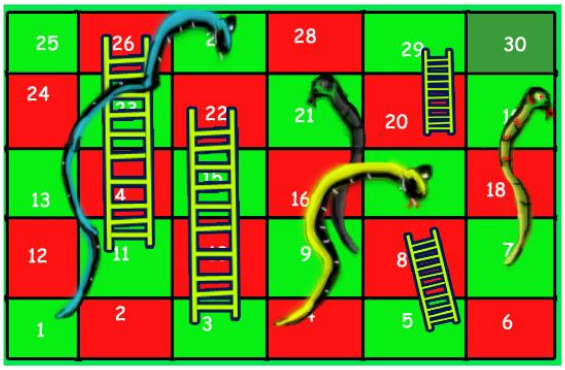
Note: there is only single source and single destination.

```
class GFG {
	private static boolean isSafe(int r,int c,int n,int arr[][],boolean vis[][]){
		return (r>=0) && (c>=0) && (r<n) && (c<n) && (arr[r][c]!=0) &&
(vis[r][c]==false);
	}
	private static int isPath(int r,int c,int n,int arr[][],boolean vis[][]){
		if(arr[r][c]==2){
			return 1;
		}
		int row[] = {-1,0,0,1};
		int col[] = {0,-1,1,0};
		vis[r][c] = true;
		int ans = 0;
		for(int i=0;i<4;i++){
			if(isSafe(r+row[i],c+col[i],n,arr,vis)){
				ans = isPath(r+row[i],c+col[i],n,arr,vis);
				if(ans==1)
					return 1;
			}
		}
		return ans;
	}
	public static int isPathExists(int arr[][],int n){
		boolean vis[][] = new boolean[n][n];
		for(int i=0;i<n;i++){
			for(int j=0;j<n;j++){
				if(arr[i][j]==1){
					return isPath(i,j,n,arr,vis);
				}
```

```
                }
            }
            return 0;
        }
        public static void main (String[] args) {
            Scanner sc = new Scanner(System.in);
            int test = sc.nextInt();
            for(int t=0;t<test;t++){
                int n = sc.nextInt();
                int arr[][] = new int[n][n];
                for(int i=0;i<n;i++){
                    for(int j=0;j<n;j++){
                        arr[i][j] = sc.nextInt();
                    }
                }
                System.out.println(isPathExists(arr,n));
            }
        }
}
```

10.9. Snake and Ladder Problem

Statement : Given a snake and ladder board of order 5x6, find the minimum number of dice throws required to reach the destination or last cell (30th cell) from source (1st cell) .



For the above board output will be 3
For 1st throw get a 2
For 2nd throw get a 6
For 3rd throw get a 2

```
class Node{
```

```
    int cell,throwNo;
    public Node(int cell, int throwNo){
        this.cell = cell;
        this.throwNo = throwNo;
    }
}
class GFG {
    public static void main (String[] args) {
        Scanner sc = new Scanner(System.in);
        int test = sc.nextInt();
        for(int t=0;t<test;t++){
            int n = sc.nextInt(); //snake or ladder count
            HashMap<Integer, Integer> map = new HashMap<>();
            for(int i=0;i<n;i++){
                int key = sc.nextInt();
                int value = sc.nextInt();
                map.put(key, value);
            }
            //processing
            //make a visited array
            boolean vis[] = new boolean[31];
            Queue<Node> q = new LinkedList<>();
            q.add(new Node(1,0)); //add first cell with 0 throw count
            //BFS
            int res = 0;
            while(!q.isEmpty()){
                Node node = q.remove();
                int cell = node.cell;
                for(int i=1;i<=6;i++){
                    int newCell = cell + i;
                    //to check for any snake or ladder
                    if(map.containsKey(newCell)){
                        newCell = map.get(newCell);
                    }
                    if(newCell==30){  //game over
                        res = node.throwNo + 1;
                        break;
                    }
                    if(!vis[newCell]){  //if cell is not visited yet
                        vis[newCell] = true;
                        q.add(new Node(newCell, node.throwNo + 1));
                    }
```

```
                }
                if(res!=0){
                    break;
                }
            }
            System.out.println(res);
        }
    }
}
```

10.10. Shortest Source to Destination Path

Statement : Given a boolean 2D matrix (0-based index), find whether there is path from (0,0) to (x,y) and if there is one path, print the minimum no of steps needed to reach it, else print -1 if the destination is not reachable. You may move in only four direction ie up, down, left and right. The path can only be created out of a cell if its value is 1.

```
class Node{
    int x,y,count;
    public Node(int x, int y, int count){
        this.x = x;
        this.y = y;
        this.count = count;
    }
}
class GFG {
    static int n,m;
    private static int sTD(int dX,int dY,int cX,int cY,int arr[][]){
        if(arr[dX][dY]==0 || arr[cX][cY]==0){
            return -1;
        }

        //BFS
        Queue<Node> q = new LinkedList<>();
        boolean vis[][] = new boolean[n][m];
        q.add(new Node(cX,cY,0));
        vis[cX][cY] = true;
        while(!q.isEmpty()){
            Node node = q.remove();
            int x = node.x;
```

```
                int y = node.y;
                if(x==dX && y==dY){
                        return node.count;
                }
                //traverse all four side
                //up
                if(x-1>=0 && !vis[x-1][y] && arr[x-1][y]==1){
                        vis[x-1][y] = true;
                        q.add(new Node(x-1,y,node.count+1));
                }
                //down
                if(x+1<n && !vis[x+1][y] && arr[x+1][y]==1){
                        vis[x+1][y] = true;
                        q.add(new Node(x+1,y,node.count+1));
                }
                //left
                if(y-1>=0 && !vis[x][y-1] && arr[x][y-1]==1){
                        vis[x][y-1] = true;
                        q.add(new Node(x,y-1,node.count+1));
                }
                //right
                if(y+1<m && !vis[x][y+1] && arr[x][y+1]==1){
                        vis[x][y+1] = true;
                        q.add(new Node(x,y+1,node.count+1));
                }
        }
        return -1;
}
public static void main (String[] args) {
        Scanner sc = new Scanner(System.in);
        int test = sc.nextInt();
        for(int t=0;t<test;t++){
                n = sc.nextInt();
                m = sc.nextInt();
                int arr[][] = new int[n][m];
                for(int i=0;i<n;i++){
                        for(int j=0;j<m;j++){
                                arr[i][j] = sc.nextInt();
                        }
                }
                int x = sc.nextInt();
                int y = sc.nextInt();
```

```
                int ans = sTD(x,y,0,0,arr);
                System.out.println(ans);
            }
        }
}
```

10.11. Minimum Cost Path (Using Dijkstra)

Statement : Given a square grid of size **N**, each cell of which contains integer cost which represents a cost to traverse through that cell, we need to find a path from top left cell to bottom right cell by which total cost incurred is minimum. You can move in 4 directions : up, down, left an right.

Note : It is assumed that negative cost cycles do not exist in input matrix.

```
class GFG {
    static class Vertex{
        int x,y;
        public Vertex(int x,int y){
            this.x = x;
            this.y = y;
        }
    }
    private static Vertex min(int dis[][],boolean vis[][],int n){
        int min = Integer.MAX_VALUE;
        Vertex p = new Vertex(0,0);
        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                if(vis[i][j]==false && dis[i][j]<min){
                    min = dis[i][j];
                    p.x = i;
                    p.y = j;
                }
            }
        }
        return p;
    }
    private static boolean isSafe(int x,int y,int n){
        return (x>=0) && (y>=0) && (x<n) && (y<n);
    }
    private static int minCost(int arr[][],int n){
        boolean vis[][] = new boolean[n][n];
```

```
		int dis[][] = new int[n][n];
		for(int i=0;i<n;i++){
			for(int j=0;j<n;j++){
				dis[i][j] = Integer.MAX_VALUE;
			}
		}
		dis[0][0] = arr[0][0];
		for(int i=0;i<n*n-1;i++){
			Vertex u = min(dis,vis,n);
			vis[u.x][u.y] = true;
			int row[] = {-1,0,0,1};
			int col[] = {0,-1,1,0};
			for(int k=0;k<4;k++){
				int x = u.x+row[k];
				int y = u.y+col[k];
				if(isSafe(x,y,n) && !vis[x][y] &&
dis[u.x][u.y]+arr[x][y]<=dis[x][y]){
					dis[x][y] = dis[u.x][u.y]+arr[x][y];
				}
			}
		}
		return dis[n-1][n-1];
	}
	public static void main (String[] args) {
		Scanner sc = new Scanner(System.in);
		int test = sc.nextInt();
		for(int t=0;t<test;t++){
			int n = sc.nextInt();
			int arr[][] = new int[n][n];
			for(int i=0;i<n;i++){
				for(int j=0;j<n;j++){
					arr[i][j] = sc.nextInt();
				}
			}
			System.out.println(minCost(arr,n));
		}
	}
}
```

10.12. Circle of strings

Statement : Given an array of strings **A[]**, determine if the strings can be chained together to form a circle. A
string **X** can be chained together with another string **Y** if the last character of **X** is same as first
character of **Y.** If every string of the array can be chained, it will form a circle.

For eg for the array arr[] = {"for", "geek", "rig", "kaf"} the answer will be Yes as the given strings can be chained as "for", "rig", "geek" and "kaf".

```
class GFG {
	static int M = 26;
	private static void dfs(ArrayList<ArrayList<Integer>> list,boolean vis[],int u){
		vis[u] = true;
		ArrayList<Integer> temp = list.get(u);
		for(int i=0;i<temp.size();i++){
			if(!vis[temp.get(i)]){
				dfs(list,vis,temp.get(i));
			}
		}
	}

	private static int isStronglyConnected(ArrayList<ArrayList<Integer>> g,boolean
mark[],int s){
		boolean vis[] = new boolean[M];
		dfs(g,vis,s);
		//After dfs is still any vertex left to visit means it is no strongly connected
graph
		for(int i=0;i<M;i++){
			if(mark[i] && !vis[i])
				return 0;
		}
		//It is stronglly connected
		return 1;
	}

	private static int isCircleOfString(String arr[],int n){
		ArrayList<ArrayList<Integer>> g = new ArrayList<>();
		for(int i=0;i<M;i++){
			ArrayList<Integer> list = new ArrayList<>();
			g.add(list);
		}
		boolean mark[] = new boolean[M];
```

```
            int in[] = new int[M];
            int out[] = new int[M];
            for(int i=0;i<n;i++){
                int f = arr[i].charAt(0) - 'a';
                int l = arr[i].charAt(arr[i].length()-1) - 'a';
                mark[f] = true;
                mark[l] = true;
                in[l]++;
                out[f]++;
                //add edge in graph
                g.get(f).add(l);

            }
            for(int i=0;i<M;i++){
                if(in[i]!=out[i])
                    return 0;
            }
            return isStronglyConnected(g,mark,arr[0].charAt(0)-'a');
        }
        public static void main (String[] args) {
            Scanner sc = new Scanner(System.in);
            int test = sc.nextInt();
            for(int i=0;i<test;i++){
                int n = sc.nextInt();
                String arr[] = new String[n];
                for(int j=0;j<n;j++){
                    arr[j] = sc.next();
                }
                System.out.println(isCircleOfString(arr,n));
            }
        }
}
```

10.13. Floyd Warshall : O(V^3)

Statement : The problem is to find shortest distances between every pair of vertices in a given edge weighted directed Graph. The Graph is represented as Adjancency Matrix, and the Matrix denotes the weight of the edegs (if it exists) else INF (1e7).

```
class GFG {
    private static int[][] floydWarshal(int arr[][],int n){
```

```
        int dist[][] = new int[n][n];
        //copy arr to dist
        for(int i=0;i<n;i++){
                for(int j=0;j<n;j++){
                        dist[i][j] = arr[i][j];
                }
        }
        //processing
        for(int k=0;k<n;k++){
                //chosse each vertex as source
                for(int i=0;i<n;i++){
                        // Pick all vertices as destination for the above picked source
                        for(int j=0;j<n;j++){

                                if(dist[i][k] + dist[k][j] < dist[i][j]){
                                        dist[i][j] = dist[i][k] + dist[k][j];
                                }
                        }
                }
        }
        return dist;
}
public static void main (String[] args) {
        Scanner sc = new Scanner(System.in);
        int test = sc.nextInt();
        for(int t=0;t<test;t++){
                int n = sc.nextInt();
                int arr[][] = new int[n][n];
                for(int i=0;i<n;i++){
                        for(int j=0;j<n;j++){
                                String s = sc.next();
                                if(s.equals("INF")){
                                        arr[i][j] = Integer.MAX_VALUE;
                                }else{
                                        arr[i][j] = Integer.parseInt(s);
                                }
                        }
                }
                int dist[][] = floydWarshal(arr,n);
                for(int i=0;i<n;i++){
                        for(int j=0;j<n;j++){
                                if(dist[i][j]==10000000)
```

```
                        System.out.print("INF"+" ");
                    else
                        System.out.print(dist[i][j]+" ");
                }
                System.out.println();
            }
        }
    }
}
```

10.14. Alien Dictionary

Statement : Given a sorted dictionary of an alien language having N words and k starting alphabets of standard dictionary. Find the order of characters in the alien language.
Note: Many orders may be possible for a particular test case, thus you may return any valid order.

```
class Graph {
    //An array representing the graph as an adjacency list
    private final LinkedList<Integer>[] adjacencyList;

    Graph(int nVertices){
        adjacencyList = new LinkedList[nVertices];
        for (int vertexIndex = 0; vertexIndex < nVertices; vertexIndex++){
            adjacencyList[vertexIndex] = new LinkedList<>();
        }
  }
    void addEdge(int startVertex, int endVertex){
        adjacencyList[startVertex].add(endVertex);
    }

    private int getNoOfVertices(){
        return adjacencyList.length;
    }

    // A recursive function used by topologicalSort
    private void topologicalSortUtil(int currentVertex, boolean[] visited,
                        Stack<Integer> stack) {

        visited[currentVertex] = true;
```

```
            for (int adjacentVertex : adjacencyList[currentVertex]){
                if (!visited[adjacentVertex]){
                    topologicalSortUtil(adjacentVertex, visited, stack);
                }
            }

            // Push current vertex to stack which stores result
            stack.push(currentVertex);
        }

        // prints a Topological Sort of the complete graph
        void topologicalSort(){
            Stack<Integer> stack = new Stack<>();

            // Mark all the vertices as not visited
            boolean[] visited = new boolean[getNoOfVertices()];
            for (int i = 0; i < getNoOfVertices(); i++){
                visited[i] = false;
            }

            // Call the recursive helper function to store Topological
            // Sort starting from all vertices one by one
            for (int i = 0; i < getNoOfVertices(); i++){
                if (!visited[i]){
                    topologicalSortUtil(i, visited, stack);
                }
            }

            // Print contents of stack
            while (!stack.isEmpty()){
                System.out.print((char)('a' + stack.pop()) + " ");
            }
        }
}

public class OrderOfCharacters{
        // This function finds and prints order of characer from a sorted array of words.
        // alpha is number of possible alphabets starting from 'a'. For simplicity, this
        // function is written in a way that only first 'alpha' characters can be there
        // in words array. For example if alpha is 7, then words[] should contain words
        // having only 'a', 'b','c' 'd', 'e', 'f', 'g'
        private static void printOrder(String[] words, int alpha){
```

```
            // Create a graph with 'aplha' edges
            Graph graph = new Graph(alpha);

            for (int i = 0; i < words.length - 1; i++){
                // Take the current two words and find the first mismatching char.
                String word1 = words[i];
                String word2 = words[i+1];
                for (int j = 0; j < Math.min(word1.length(), word2.length()); j++){
                    // If we find a mismatching character, then add an edge
                    // from character of word1 to that of word2
                    if (word1.charAt(j) != word2.charAt(j)){
                        graph.addEdge(word1.charAt(j) - 'a', word2.charAt(j)-
'a');
                        break;
                    }
                }
            }

            // Print topological sort of the above created graph
            graph.topologicalSort();
        }

        // Driver program to test above functions
        public static void main(String[] args){
            String[] words = {"caa", "aaa", "aab"};
            printOrder(words, 3);
        }
}
```