```
public class Striver_Graph_Playlist {
    class Striver{
    //*****************G5-Breadth First Search**********************************O(2E+V)
***| O(3N)********** */
     class Solution{
        public ArrayList<Integer> bfsOfGraph(int V,ArrayList<ArrayList<Integer>> adj){
        ArrayList<Integer> bfs=new ArrayList<>();
        boolean vis[]=new boolean[V];
        for(int i=0;i<V;i++){
            vis[i]=false;
        }
        Queue<Integer> q=new LinkedList<>();
        q.add(src);
        vis[src]=true;
        while(!q.isEmpty()){
            Integer node=q.poll();
            bfs.add(node);
            for(Integer it: adj.get(node)){
                if(vis[it]==false){
                    vis[it]=true;
                    q.add(it);
                }
            }
        }
        return bfs;
    }
    }
        //**************G6-Depth First Search**************************************O(2E+V)
***| O(3N)********* */
    class Solution{
        public void dfs(int src,boolean []vis,ArrayList<ArrayList<Integer>> adj,
ArrayList<Integer> dfs){
            vis[src]=true;
            dfs.add(src);
            for(Integer it: adj.get(src)){
                if(visited[it]==false){
                    dfs(it,vis,adj,dfs);
                }
            }
        }
        public ArrayList<Integer> dfsOfGraph(int V,ArrayList<ArrayList<Integer>> adj){
            ArrayList<Integer> dfs=new ArrayList<>();
            boolean vis[]=new boolean[V];
            for(int i=0;i<V;i++){
                vis[i]=false;
            }
            visited[src]=true;
            dfs(src,vis,adj,dfs);
            return dfs;
```

```
        }
        }
    //******************G7-Number of
Provinces********************************************** */
    class Solution {
        static void  dfs(int i,ArrayList<ArrayList<Integer>> adjLs,boolean []visited){
            visited[i]=true;
            for(Integer it: adjLs.get(i)){
                if(visited[it]==false){
                    dfs(it,adjLs,visited);
                }
            }
        }
        static int numProvinces(ArrayList<ArrayList<Integer>> adj, int V) {
            // code here
            ArrayList<ArrayList<Integer>> adjLs=new ArrayList<>();
            for(int i=0;i<V;i++){
                adjLs.add(new ArrayList<>());
            }

            for(int i=0;i<V;i++){
             for(int j=0;j<V;j++){
                 if(adj.get(i).get(j)==1&&i!=j){
                    adjLs.get(i).add(j);
                    adjLs.get(j).add(i);
                }
             }
            }

            boolean visited[]=new boolean[V];
            for(int i=0;i<V;i++){
                visited[i]=false;
            }
            int cnt=0;
            for(int i=0;i<V;i++){
                if(visited[i]==false){
                    dfs(i,adjLs,visited);
                    cnt++;
                }
            }
            return cnt;
    }
    }
    //******************G8-*Number of Islands************************************************
*/
    class Solution {
        class Pair{
            int row;
            int col;
```

```
        Pair(int _row,int _col){
        this.row=_row;
        this.col=_col;
        }
    }
  public void bfs(int i,int j,char [][]grid,boolean [][]visited,int n,int m){
    visited[i][j]=true;
    Queue<Pair> q=new LinkedList<>();
    q.add(new Pair(i,j));
    while(!q.isEmpty()){
        int r=q.peek().row;
        int c=q.peek().col;
        q.remove();
        for(int dx=-1;dx<=1;dx++){
            for(int dy=-1;dy<=1;dy++){
                int nr=r+dx;
                int nc=c+dy;

if(nr>=0&&nr<n&&nc>=0&&nc<m&&grid[nr][nc]=='1'&&visited[nr][nc]==false){
                 {
                   visited[nr][nc]=true;
                   q.add(new Pair(nr,nc));
                 }
            }
        }
    }
  }
  }
  public int numIslands(char[][] grid) {
        int n=grid.length;
        int m=grid[0].length;
        boolean visited[][]=new boolean[n][m];
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                visited[i][j]=false;
            }
        }
        int cnt=0;
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                if(visited[i][j]==false && grid[i][j]=='1'){
                cnt++;
                bfs(i,j,grid,visited,n,m);
            }
        }
    }
        return cnt;
  }
}
```

```
   //******************G9-Flood Fill********************************************************** */
   class Solution{
      public void dfs(int sr,int sc,int [][]ans,int [][]image,int []dx,int []dy,int iniColor,int
newColor,int n,int m){
         ans[sr][sc]=newColor;
         for(int i=0;i<4;i++){
            int nr=sr+dx[i];
            int nc=sc+dy[i];

if(nr>=0&&nr<n&&nc>=0&&nc<m&&image[nr][nc]==iniColor&&ans[nr][nc]!=newColor){
                  dfs(nr,nc,ans,image,dx,dy,iniColor,newColor,n,m);
            }
         }
      }
      public int[][] floodFill(int[][] image, int sr, int sc, int newColor)
      {
         // Code here
         int n=image.length;
         int m=image[0].length;
         int iniColor=image[sr][sc];
         int ans[][]=image;
         int dx[]={-1,0,1,0};
         int dy[]={0,1,0,-1};
         dfs(sr,sc,ans,image,dx,dy,iniColor,newColor,n,m);
         return ans;
      }
   }
   //******************G10-Rotten
Oranges*****************************************************/
   class Solution{
   class Pair{
      int rows;
      int cols;
      int time;
      Pair(int _x,int _y,int _z){
         this.rows=_x;
         this.cols=_y;
         this.time=_z;
      }
   }
   //Function to find minimum time required to rot all oranges.
   public int orangesRotting(int[][] grid)
   {
      // Code here
      int rows=grid.length;
      int cols=grid[0].length;

      Queue<Pair> q=new LinkedList<>();
      boolean visited[][]=new boolean[rows][cols];
```

```
        int cntFresh=0;
        for(int i=0;i<rows;i++){
            for(int j=0;j<cols;j++){
              if(grid[i][j]==2){
                  q.add(new Pair(i,j,0));
                  visited[i][j]=true;
              }
              else{
                  visited[i][j]=false;
              }
              if(grid[i][j]==1){
                  cntFresh++;
              }
            }
        }
        if(cntFresh==0){
            return 0;
        }
        int tm=0;
        int dx[]={-1,0,1,0};
        int dy[]={0,1,0,-1};
        int cnt=0;
        while(!q.isEmpty()){
            int r=q.peek().rows;
            int c=q.peek().cols;
            int t=q.peek().time;
            tm=Math.max(t,tm);
            q.remove();
            for(int i=0;i<4;i++){
                int nr=r+dx[i];
                int nc=c+dy[i];

if(nr>=0&&nr<rows&&nc<cols&&nc>=0&&grid[nr][nc]==1&&visited[nr][nc]==false){
                    q.add(new Pair(nr,nc,t+1));
                    visited[nr][nc]=true;
                    cnt++;
                    }
                }
            }
        if(cnt!=cntFresh)return -1;
        return tm;
    }
}
    //******************G11-Detect Cycle in Undirected Graph
BFS********************************************** */
    class Solution{
    public boolean checkforCycle(int src,int V,boolean vis[],
ArrayList<ArrayList<Integer>> adj,boolean vis[]){
        vis[src]=true;
```

```
        Queue<Pair> q=new LinkedList<>();
        q.add(new Pair(src,-1));
        while(!q.isEmpty()){
            int node=q.peek().first;
            int parent=q.peek().second;
            q.remove();
            for(Integer adjacentNode: adj.get(node))
            {
                if(vis[adjacentNode]==false){
                    vis[adjacentNode]=true;
                    q.add(new Pair(adjacentNode,node));
                }
                else if(parent!=adjacentNode){
                    return true;
                }
            }
        }
        return false;
    }
    public boolean isCyclic(int V,ArrayList<ArrayList<Integer>> adj){
    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]==false){
            if(checkCycle(i,V,adj,vis)==true)
            return true;
        }
    }
    return false;
    }
}
    //**********************G12-Detect Cycle in Undirected Graph
DFS************************************************** */
    class Solution {
        // Function to detect cycle in an undirected graph.
        public boolean isCycle(int V, ArrayList<ArrayList<Integer>> adj) {
            // Code here
            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] == false) {
                    if (dfs(i, -1, vis, adj) == true) {
                        return true;

                    }
```

```
            }
        }
        return false;
    }

    public boolean dfs(int node, int parent, boolean vis[],
ArrayList<ArrayList<Integer>> adj) {
        vis[node] = true;
        for (Integer adjacentNode : adj.get(node)) {
            if (vis[adjacentNode] == false) {
                if (dfs(adjacentNode, node, vis, adj) == true) {

                    return true;
                }
            } else if (adjacentNode != parent) {

                return true;
            }
        }
        return false;
    }
}
//*****************G13-Distance of nearest cell having 1|0/1
Matrix|*****************************/
class Solution  {
class Tuple{
    int row;
    int col;
    int steps;
    Tuple(int _row,int _col,int _steps){
        this.row=_row;
        this.col=_col;
        this.steps=_steps;
    }
}
public int[][] nearest(int[][] grid)
{
    // Code here
    int n=grid.length;
    int m=grid[0].length;
    Queue<Tuple> q=new LinkedList<>();
    boolean visited[][]=new boolean[n][m];
    int distance[][]=new int[n][m];
    for(int i=0;i<n;i++){
        for(int j=0;j<m;j++){
            if(grid[i][j]==1){
                q.add(new Tuple(i,j,0));
                visited[i][j]=true;
            }
```

```
                else{
                    visited[i][j]=false;
                }
            }
        }
        int dx[]={-1,0,1,0};
        int dy[]={0,1,0,-1};
        while(!q.isEmpty()){
            int r=q.peek().row;
            int c=q.peek().col;
            int st=q.peek().steps;
            q.remove();
            distance[r][c]=st;
            for(int i=0;i<4;i++){
                int nr=r+dx[i];
                int nc=c+dy[i];
                if(nr>=0&&nr<n&&nc>=0&&nc<m&&visited[nr][nc]==false&&grid[nr][nc]==0){
                    visited[nr][nc]=true;
                    q.add(new Tuple(nr,nc,st+1));
                }
            }
        }
        return distance;
    }
}
    //*****************G14-Replace 'O' with 'X'********************************** */
    class Solution{
        static void dfs(int row,int col,char [][]a,boolean [][]visited,int []dx,int []dy,int n,int m){
            visited[row][col]=true;

            for(int i=0;i<4;i++){
                    int nr=row+dx[i];
                    int nc=col+dy[i];
                    if(nr>=0&&nr<n&&nc>=0&&nc<m&&visited[nr][nc]==false&&a[nr][nc]=='O'){
                        dfs(nr,nc,a,visited,dx,dy,n,m);
                    }
                }
        }
        static char[][] fill(int n, int m, char a[][])
        {
            // code here
            int dx[]={-1,0,1,0};
            int dy[]={0,1,0,-1};

            boolean visited[][]=new boolean[n][m];
            for(int i=0;i<n;i++){
                for(int j=0;j<m;j++){
                    visited[i][j]=false;
                }
```

```
        }

        for(int j=0;j<m;j++){
            if(visited[0][j]==false&&a[0][j]=='O'){
                dfs(0,j,a,visited,dx,dy,n,m);
            }
             if(visited[n-1][j]==false&&a[n-1][j]=='O'){
                dfs(n-1,j,a,visited,dx,dy,n,m);
            }
        }
        for(int i=0;i<n;i++){
            if(visited[i][0]==false&&a[i][0]=='O'){
                dfs(i,0,a,visited,dx,dy,n,m);
            }
             if(visited[i][m-1]==false&&a[i][m-1]=='O'){
                dfs(i,m-1,a,visited,dx,dy,n,m);
            }
        }

        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                if(visited[i][j]==false&&a[i][j]=='O'){
                a[i][j]='X';
                }
            }
        }
        return a;

    }
}
//*****************G15-Number of Enclaves************************** */
class Solution {
    class Pair{
        int row;
        int col;
        Pair(int _row,int _col){
        this.row=_row;
        this.col=_col;
        }
    }
    int numberOfEnclaves(int[][] grid) {
        Queue<Pair> q=new LinkedList<>();
        int n=grid.length;
        int m=grid[0].length;
        boolean visited[][]=new boolean[n][m];
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
             visited[i][j]=false;
            }
```

```
        }

        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                if(i==0||i==n-1||j==0||j==m-1){
                    if(grid[i][j]==1)
                    {
                        q.add(new Pair(i,j));
                        visited[i][j]=true;
                    }
                }
            }
        }

        int dx[]={-1,0,1,0};
        int dy[]={0,1,0,-1};

        while(!q.isEmpty()){
            int r=q.peek().row;
            int c=q.peek().col;
            q.remove();
            for(int i=0;i<4;i++){
                int nr=r+dx[i];
                int nc=c+dy[i];
                if(nr>=0&&nr<n&&nc>=0&&nc<m&&grid[nr][nc]==1&&visited[nr][nc]==false)
{
                    {
                        visited[nr][nc]=true;
                        q.add(new Pair(nr,nc));
                    }
                }
            }
        }

        int cnt=0;
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
             if(visited[i][j]==false&&grid[i][j]==1){
                 cnt++;
                }
            }

        }
        return cnt;
    }
    }
    //*****************G16-Number of Distinct
Islands*****************************Constructive+DFS*************** */
    class Solution {
```

```
        class Pair{
            int row;
            int col;
            Pair(int _row,int _col){
            this.row=_row;
            this.col=_col;
            }
        }
        public void dfs(int row,int col,int [][]grid,boolean [][]visited,int []dx,int []dy,
ArrayList<String> res,int sr,int sc,int n,int m){
                visited[row][col]=true;
                res.add(toString(row-sr,col-sc));

                for(int i=0;i<4;i++){
                    int nr=row+dx[i];
                    int nc=col+dy[i];
                    if(nr>=0&&nr<n&&nc>=0&&nc<m&&visited[nr][nc]==false&&grid[nr][nc]==1)
{
                         dfs(nr,nc,grid,visited,dx,dy,res,sr,sc,n,m);
                    }
                }
            }
            public String toString(int r,int c){
                return Integer.toString(r)+" "+Integer.toString(c);
            }
        int countDistinctIslands(int[][] grid) {
            // Your Code here
                int n=grid.length;
                int m=grid[0].length;
                boolean visited[][]=new boolean[n][m];
                for(int i=0;i<n;i++){
                    for(int j=0;j<m;j++){
                        visited[i][j]=false;
                    }
                }
                int dx[]={-1,0,1,0};
                int dy[]={0,1,0,-1};
                HashSet<ArrayList<String>> hs=new HashSet<>();
                for(int i=0;i<n;i++){
                    for(int j=0;j<m;j++){
                        if(visited[i][j]==false && grid[i][j]==1){
                        ArrayList<String> res=new ArrayList<>();
                        dfs(i,j,grid,visited,dx,dy,res,i,j,n,m);
                        hs.add(res);
                    }
                }
            }
            return hs.size();
        }
```

```
    }
    //*****************G17-Bipartite
Graph******BFS************************************************************** */
    class Solution
{
    public boolean check(int start,int V,ArrayList<ArrayList<Integer>> adj,int []color){
        Queue<Integer> q=new LinkedList<>();
        q.add(start);
        color[start]=0;
        while(!q.isEmpty()){
            int node=q.peek();
            q.remove();

            for(Integer it: adj.get(node)){
                if(color[it]==-1){
                    color[it]=1-color[node];
                    q.add(it);
                }
                else if(color[it]==color[node]){
                    return false;
                }
            }
        }
        return true;
    }
    public boolean isBipartite(int V, ArrayList<ArrayList<Integer>>adj)
    {
        // Code here
        int color[]=new int[V];
        for(int i=0;i<V;i++)
        {
            color[i]=-1;
        }
        for(int i=0;i<V;i++){
            if(color[i]==-1){
                if(check(i,V,adj,color)==false){
                return false;
                }
            }
        }
        return true;
    }
}
    //*****************G18-Bipartite
Graph******DFS************************************************************** */
    class Solution{
    public boolean check(int start,int c,ArrayList<ArrayList<Integer>> adj,int []color){
        color[start]=c;
        for(Integer it: adj.get(start)){
```

```
            if(color[it]==-1){
                if(check(it,1-c,adj,color)==false){
                    return false;
                    }
                }
            else if(color[it]==color[start]){
                    return false;
                }
            }
        return true;
    }
    public boolean isBipartite(int V, ArrayList<ArrayList<Integer>>adj)
    {
        // Code here
        int color[]=new int[V];
        for(int i=0;i<V;i++)
        {
            color[i]=-1;
        }
        for(int i=0;i<V;i++){
            if(color[i]==-1){
                if(check(i,0,adj,color)==false){
                return false;
                }
            }
        }
        return true;
    }
}
    //*****************G19-Detect cycle in a directed graph using
DFS********************************************** */
    class Solution{
        private boolean dfsCheck(int node,ArrayList<ArrayList<Integer>> adj,boolean vis[],
boolean pathVis[]) {
        vis[node]=true;
        pathVis[node]=true;
        for(Integer it : adj.get(node)) {
            if(vis[it]==false){
                if(dfsCheck(it,adj,vis,pathVis)==true)
                    return true;
            }
            else if(pathVis[it]==true){
                return true;
            }
        }
        pathVis[node]=false;
        return false;
    }
    public boolean isCyclic(int V,ArrayList<ArrayList<Integer>> adj) {
```

```
        boolean vis[]=new boolean[V];
        boolean pathVis[] = new boolean[V];

        for(int i=0;i<V;i++){
            if(vis[i]==false) {
                if(dfsCheck(i,adj,vis,pathVis)==true)return true;
            }
        }
        return false;
    }
}
    //*****************G20-Find Eventual States
DFS************************************************************** */
    class Solution {
        private boolean dfsCheck(int node,List<List<Integer>> adj,int vis[],int pathVis[],int
check[]){
            vis[node]=1;
            pathVis[node]=1;
            check[node]=0;
            for(int it : adj.get(node)){
                if(vis[it]==0){
                    if(dfsCheck(it,adj,vis,pathVis,check)==true)
                        return true;
                    }
                else if(pathVis[it]==1){
                    return true;
                }
            }
            check[node]=1;
            pathVis[node]=0;
            return false;
        }
        List<Integer> eventualSafeNodes(int V,List<List<Integer>> adj){
            int vis[]=new int[V];
            int pathVis[]=new int[V];
            int check[]=new int[V];
            for(int i=0;i<V;i++){
                if(vis[i]==0) {
                    dfsCheck(i,adj,vis,pathVis,check);
                }
            }
            List<Integer> safeNodes=new ArrayList<>();
            for(int i=0;i<V;i++){
                if(check[i]==1)
                    safeNodes.add(i);
            }
            return safeNodes;
        }
    }
```

```
//*****************G21-Topological Sort DFS*************************************** */
class Solution{
    public void dfs(int node,boolean vis[],ArrayList<ArrayList<Integer>> adj,
Stack<Integer> st){
        vis[node]=true;
        for(Integer it: adj.get(node))
        {
            if(vis[it]==false){
                dfs(it,vis,adj,st);
            }
        }
        st.push(node);
    }
    public ArrayList<Integer> dfsOfGraph(int V,ArrayList<ArrayList<Integer>> adj){
    boolean vis[]=new boolean[V];
    Stack<Integer> st=new Stack<>();
    for(int i=0;i<V;i++){
        vis[i]=false;
    }
    for(int i=0;i<V;i++){
        if(vis[i]==false){
            dfs(i,vis,adj,st);
        }
    }
    int ans[]=new int[V];
    int i=0;
    while(!st.isEmpty()){
        ans[i++]=st.peek();
        st.pop();
    }
    return ans;
}
}
 //*****************G22-Topological Sort BFS*************************************** */
class Solution{
public boolean topoSort(int V,ArrayList<ArrayList<Integer>> adj){
    int indegree[]=new int[V];
    for(int i=0;i<V;i++){
        for(Integer it:adj.get(i)){
            indegree[it]++;
        }
    }
    Queue<Integer> q=new LinkedList<>();
    for(int i=0;i<V;i++){
        if(indegree[i]==0){
            q.add(i);
        }
    }
    int i=0;
```

```
        int topo[]=new int[V];
        while(!q.isEmpty()){
            int node=q.peek();
            q.remove();
            topo[i++]=node;
            for(Integer it: adj.get(node)){
                indegree[it]--;
                if(indegree[it]==0){q.add(it);}
            }
        }
        return topo;
    }
}
    //******************G-23-Detect a Cycle in Directed Graph Kahn's Algorithm
BFS***************/
    class Solution {
        public boolean isCyclic(int N,ArrayList<ArrayList<Integer>> adj){
            // int topo[] = new int[N];
            int indegree[]=new int[N];
            for(int i=0;i<N;i++){
                for(Integer it : adj.get(i)){
                    indegree[it]++;
                }
            }

            Queue<Integer> q=new LinkedList<Integer>();
            for(int i=0;i<N;i++){
                if(indegree[i]==0){
                    q.add(i);
                }
            }
            int cnt=0;
            while(!q.isEmpty()){
                Integer node=q.poll();
                cnt++;
                for (Integer it : adj.get(node)){
                    indegree[it]--;
                    if (indegree[it]==0){
                        q.add(it);
                    }
                }
            }
            if(cnt==N)
                return false;
            return true;
        }
    }
    //*****************G24-Course Schedule I and
II************************************************************** */
```

```
class Solution {
    public boolean isPossible(int N, int[][] prerequisites)
    {
        // Your Code goes here
        ArrayList<ArrayList<Integer>> adj=new ArrayList<>();
        for(int i=0;i<N;i++){
            adj.add(new ArrayList<>());
        }
        int m=prerequisites.length;
        for(int i=0;i<m;i++){
            adj.get(prerequisites[i][0]).add(prerequisites[i][1]);
        }

        int indegree[]=new int[N];
        for(int i=0;i<N;i++){
            for(Integer it:adj.get(i)){
                indegree[it]++;
            }
        }
        Queue<Integer> q=new LinkedList<>();
        for(int i=0;i<N;i++){
            if(indegree[i]==0){
                q.add(i);
            }
        }
        List<Integer> topo=new ArrayList<>();
            while(!q.isEmpty()){
                int node=q.peek();
                q.remove();
                topo.add(node);
                for(Integer it: adj.get(node)){
                    indegree[it]--;
                    if(indegree[it]==0){q.add(it);}
                }
            }
        if(topo.size()==N)return true;
        return false;}
}
//****************G25-Find Eventual Safe States BFS
Topo*********************************************************/
class Solution {

    List<Integer> eventualSafeNodes(int V, List<List<Integer>> adj) {

        // Your code here
         ArrayList<ArrayList<Integer>> adjRev=new ArrayList<>();
            for(int i=0;i<V;i++){
                adjRev.add(new ArrayList<>());
            }
```

```
            int indegree[]=new int[V];
            for(int i=0;i<V;i++){
                for(Integer it: adj.get(i)){
                adjRev.get(it).add(i);
                indegree[i]++;
                }
            }
            Queue<Integer> q=new LinkedList<>();
            List<Integer> safeNodes=new ArrayList<>();
            for(int i=0;i<V;i++){
                if(indegree[i]==0){
                    q.add(i);
                }
            }

            while(!q.isEmpty()){
                int node=q.peek();
                q.remove();
                safeNodes.add(node);
                for(Integer it: adjRev.get(node)){
                        indegree[it]--;
                        if(indegree[it]==0){q.add(it);}
                }
            }

            Collections.sort(safeNodes);
            return safeNodes;
        }
    }
    //****************G26-Alien Dictionary Topo
Sort************************************************************/
    class Solution{
    public List<Integer> topoSort(int V,List<List<Integer>> adj){
            int indegree[]=new int[V];
            for(int i=0;i<V;i++){
                for(Integer it:adj.get(i)){
                    indegree[it]++;
                }
            }
            Queue<Integer> q=new LinkedList<>();
            for(int i=0;i<V;i++){
                if(indegree[i]==0){
                    q.add(i);
                }
            }
            List<Integer> topo=new ArrayList<>();
            while(!q.isEmpty()){
                int node=q.peek();
                q.remove();
```

```
                topo.add(node);
                for(Integer it: adj.get(node)){
                    indegree[it]--;
                    if(indegree[it]==0){q.add(it);}
                }
            }
            return topo;
        }
    public String findOrder(String [] dict, int N, int K)
    {
        // Write your code here
        List<List<Integer>> adj=new ArrayList<>();
        for(int i=0;i<K;i++){
            adj.add(new ArrayList<>());
        }

        for(int j=0;j<N-1;j++){
            String s1=dict[j];
            String s2=dict[j+1];
            int len=Math.min(s1.length(),s2.length());
            for(int i=0;i<len;i++){
                if(s1.charAt(i)!=s2.charAt(i)){
                    adj.get(s1.charAt(i)-'a').add(s2.charAt(i)-'a');
                    break;
                }
            }
        }

        List<Integer> topo=topoSort(K,adj);
        String ans="";
        for(int it: topo){
            ans=ans+(char)(it+(int)('a'));
        }
        return ans;


    }
}
    //***************G27-Shortest Path in
DAG********************************************************************** */
    class Solution {
        class Pair{
            int destination;
            int edgeW;
            Pair(int d,int e){
                this.destination=d;
                this.edgeW=e;
            }
        }
```

```
        public void dfsTopo(int node,ArrayList<ArrayList<Pair>> adj,boolean vis[],
Stack<Integer> st){
            vis[node]=true;
            for(int i=0;i<adj.get(node).size();i++)
            {
                int v=adj.get(node).get(i).destination;
                if(vis[v]==false){
                    dfsTopo(v,adj,vis,st);
                }
            }
            st.push(node);
        }
      public int[] shortestPath(int N,int M, int[][] edges) {
        //Code here
         ArrayList<ArrayList<Pair>> adj=new ArrayList<>();
        for(int i=0;i<N;i++){
            ArrayList<Pair> temp=new ArrayList<>();
            adj.add(temp);
            }
        for(int i=0;i<M;i++){
            int src=edges[i][0];
            int dest=edges[i][1];
            int ew=edges[i][2];
            adj.get(src).add(new Pair(dest,ew));
        }

        Stack<Integer> st=new Stack<>();
        boolean vis[]=new boolean[N];
        for(int i=0;i<N;i++){
            vis[i]=false;
        }
        for(int i=0;i<N;i++){
            if(vis[i]==false){
                dfsTopo(i,adj,vis,st);
            }
        }

        int distance[]=new int[N];
        for(int i=0;i<N;i++){
        distance[i]=(int)(1e9);
        }
        distance[0]=0;
        while(!st.isEmpty()){
            int node=st.peek();
            st.pop();
            for(int i=0;i<adj.get(node).size();i++){
                int v=adj.get(node).get(i).destination;
                int wt=adj.get(node).get(i).edgeW;
                if(distance[node]+wt<distance[v]){
```

```
                    distance[v]=distance[node]+wt;
                }
            }
        }

        for(int i=0;i<N;i++){
        if(distance[i]==(int)(1e9))
        distance[i]=-1;
        }
        return distance;

    }
  }
  //****************G28-Shortest Path in Unidirected graph with unit
distance********************************** */
  class Solution {
  public int[] shortestPath(int[][] edges,int n,int m ,int src) {
      // Code here
      ArrayList<ArrayList<Integer>> adj=new ArrayList<>();
      for(int i=0;i<n;i++){
          adj.add(new ArrayList<>());
          }
      for(int i=0;i<m;i++){
          adj.get(edges[i][0]).add(edges[i][1]);
          adj.get(edges[i][1]).add(edges[i][0]);
          }
      int distance[]=new int[n];
      for(int i=0;i<n;i++){
      distance[i]=(int)(1e9);
      }
      distance[src]=0;
      Queue<Integer> q=new LinkedList<>();
      q.add(src);
      while(!q.isEmpty()){
          int node=q.peek();
          q.remove();
          for(Integer it: adj.get(node)){
              if(distance[node]+1<distance[it]){
                  distance[it]=distance[node]+1;
                  q.add(it);
              }
          }
      }
      for(int i=0;i<n;i++){
          if(distance[i]==(int)(1e9)){
              distance[i]=-1;
          }
      }
      return distance;
```

```
    }
}
    //***************G29-Word Ladder
|*****************************************************************************/
    class Solution{
    class Pair{
            String word;
            int steps;
            Pair(String _word,int _steps){
                this.word=_word;
                this.steps=_steps;
            }
        }
    public int wordLadderLength(String startWord, String targetWord, String[] wordList)
    {
        // Code here
        Queue<Pair> q=new LinkedList<>();
        q.add(new Pair(startWord,1));
        Set<String> st=new HashSet<String>();
        int n=wordList.length;
        for(int i=0;i<n;i++){
            st.add(wordList[i]);
        }
        st.remove(startWord);
        while(!q.isEmpty()){
            String curW=q.peek().word;
            int step=q.peek().steps;
            q.remove();
            if(curW.equals(targetWord)==true) return step;
            for(int i=0;i<curW.length();i++){
                for(char ch='a';ch<='z';ch++){
                char replacedArray[]=curW.toCharArray();
                replacedArray[i]=ch;
                String replacedWord=new String(replacedArray);

                if(st.contains(replacedWord)==true){
                    st.remove(replacedWord);
                    q.add(new Pair(replacedWord,step+1));
                }
            }
            }
        }
        return 0;
    }
}
    //***************G30-Word Ladder
||************************************************************** */
    class Solution{
    public ArrayList<ArrayList<String>> findSequences(String startWord, String
```

```
targetWord, String[] wordList)
    {
        // Code here
        Set<String> st=new HashSet<String>();
        int n=wordList.length;
        for(int i=0;i<n;i++){
            st.add(wordList[i]);
        }
        Queue<ArrayList<String>> q=new LinkedList<>();
        ArrayList<String> ls=new ArrayList<>();
        ls.add(startWord);
        q.add(ls);

        ArrayList<String> usedOnLevel=new ArrayList<>();
        ArrayList<ArrayList<String>> ans=new ArrayList<>();
        usedOnLevel.add(startWord);
        int level=0;

        while(!q.isEmpty()){
            ArrayList<String> vec=q.peek();
            q.remove();
            if(vec.size()>level){
                level++;
                for(String it: usedOnLevel){
                    st.remove(it);
                }
            }
            String word=vec.get(vec.size()-1);
            if(word.equals(targetWord)==true){
                    if(ans.size()==0){
                        ans.add(vec);
                    }
                    else if(ans.get(0).size()==vec.size()){
                        ans.add(vec);
                    }
                }

            for(int i=0;i<word.length();i++){
                for(char ch='a';ch<='z';ch++){
                char replacedArray[]=word.toCharArray();
                replacedArray[i]=ch;
                String replacedWord=new String(replacedArray);

                if(st.contains(replacedWord)==true){
                    vec.add(replacedWord);
                    ArrayList<String> temp=new ArrayList<>(vec);
                    q.add(temp);
                    usedOnLevel.add(replacedWord);
                    vec.remove(vec.size()-1);
```

```
                }
            }
        }
    }
    return ans;
    }
}
//*****************G32-Djikstra's Shortest Path Algorithm**************************************************** */
class Solution
{
    class Pair{
        int distance;
        int node;
        Pair(int _dis,int _node){
            this.distance=_dis;
            this.node=_node;
        }
    };
    //Function to find the shortest distance of all the vertices
    //from the source vertex S.
    static int[] dijkstra(int V, ArrayList<ArrayList<ArrayList<Integer>>> adj, int S)
    {
        // Write your code here
        PriorityQueue<Pair> pq=new PriorityQueue<Pair>((x,y)->x.distance-y.distance);
        int distance[]=new int[V];
        for(int i=0;i<V;i++){
            distance[i]=(int)(1e9);
        }
        distance[S]=0;
        pq.add(new Pair(0,S));
        while(pq.size()!=0){
            int dis=pq.peek().distance;
            int node=pq.peek().node;
            pq.remove();
            for(int i=0;i<adj.get(node).size();i++){
                int edgeW=adj.get(node).get(i).get(1);
                int adjNode=adj.get(node).get(i).get(0);

                if(dis+edgeW<distance[adjNode]){
                    distance[adjNode]=dis+edgeW;
                    pq.add(new Pair(dis+edgeW,adjNode));
                }
            }
        }
        return distance;
    }
}
```

```
    //*****************G35-Print Shortest Path
Djikstra************************************************************* */
    class Solution {
        class Pair{
            int first;
            int second;
            Pair(int _first,int _second){
                this.first=_first;
                this.second=_second;
            }
        }
        public static List<Integer> shortestPath(int n, int m, int edges[][]) {
            // code here
            ArrayList<ArrayList<Pair>> adj=new ArrayList<>();
            for(int i=0;i<=n;i++){
                adj.add(new ArrayList<>());
            }
            for(int i=0;i<m;i++){
                adj.get(edges[i][0]).add(new Pair(edges[i][1],edges[i][2]));
                adj.get(edges[i][1]).add(new Pair(edges[i][0],edges[i][2]));
            }
            PriorityQueue<Pair> pq=new PriorityQueue<Pair>((x,y)->x.first-y.first);
            int distance[]=new int[n+1];
            int parent[]=new int[n+1];
            for(int i=1;i<=n;i++){
                distance[i]=(int)(1e9);
                parent[i]=i;
            }
            distance[1]=0;
            pq.add(new Pair(0,1));
            while(pq.size()!=0){
                Pair it=pq.peek();
                int dis=it.first;
                int node=it.second;
                pq.remove();

                for(Pair iter: adj.get(node)){
                    int adjNode=iter.first;
                    int edgeWeight=iter.second;
                    if(dis+edgeWeight<distance[adjNode]){
                        distance[adjNode]=dis+edgeWeight;
                        pq.add(new Pair(dis+edgeWeight,adjNode));
                        parent[adjNode]=node;
                    }
                }
            }

            List<Integer> path=new ArrayList<>();
            if(distance[n]==1e9){
```

```
            path.add(-1);
            return path;
        }
        int node=n;
        while(parent[node]!=node){
            path.add(node);
            node=parent[node];
        }
        path.add(1);
        Collections.reverse(path);
        return path;
    }
}
    //*****************G36-Shortest Distance in a Binary Maze****************** */
    class Solution {
        class Tuple{
            int dis;
            int x;
            int y;
            Tuple(int _dis,int _x,int _y){
                this.dis=_dis;
                this.x=_x;
                this.y=_y;
            }
        }
        int shortestPath(int[][] grid, int[] source, int[] destination) {
            if(source[0]==destination[0]&&source[1]==destination[1]){
                return 0;
            }
            int sx=source[0],sy=source[1],Dx=destination[0],Dy=destination[1];
            int n=grid.length;
            int m=grid[0].length;
            int distance[][]=new int[n][m];
            for(int i=0;i<n;i++){
                for(int j=0;j<m;j++){
                    distance[i][j]=(int)(1e8);
                }
            }
            distance[sx][sy]=0;
            int dx[]={-1,0,1,0};
            int dy[]={0,1,0,-1};
            Queue<Tuple> q=new LinkedList<>();
            q.add(new Tuple(0,sx,sy));
            while(!q.isEmpty()){
                int d=q.peek().dis;
                int xc=q.peek().x;
                int yc=q.peek().y;
                q.remove();
                for(int i=0;i<4;i++){
```

```
                int nr=xc+dx[i];
                int nc=yc+dy[i];
                if(nr>=0&&nr<n&&nc>=0&&nc<m&&grid[nr][nc]==1&&d+1<distance[nr][nc])
{
                    distance[nr][nc]=d+1;
                    if(nr==Dx&&nc==Dy){
                        return d+1;
                    }
                    else{
                    q.add(new Tuple(d+1,nr,nc));
                    }
                }
             }
          }
          return -1;
       }
    }
    //*****************G37-Path with Minimum Effort************************************** */
    class Solution {
        class Tuple{
            int dis;
            int x;
            int y;
            Tuple(int _dis,int _x,int _y){
                this.dis=_dis;
                this.x=_x;
                this.y=_y;
            }
      }
      int MinimumEffort(int heights[][]) {
          PriorityQueue<Tuple> pq=new PriorityQueue<Tuple>((x,y)->x.dis-y.dis);
            int n=heights.length;
            int m=heights[0].length;
            int distance[][]=new int[n][m];
            for(int i=0;i<n;i++){
                for(int j=0;j<m;j++){
                    distance[i][j]=(int)(1e9);
                }
            }
            distance[0][0]=0;
            int dx[]={-1,0,1,0};
            int dy[]={0,1,0,-1};
            pq.add(new Tuple(0,0,0));
            while(!pq.isEmpty()){
                Tuple it=pq.peek();
                pq.remove();
                int diff=it.dis;
                int xc=it.x;
                int yc=it.y;
```

```
                if(xc==n-1&&yc==m-1){
                    return diff;
                }

                for(int i=0;i<4;i++){
                    int nr=xc+dx[i];
                    int nc=yc+dy[i];
                    if(nr>=0&&nr<n&&nc>=0&&nc<m){
                        int newEffort=Math.max(diff,Math.abs(heights[nr][nc]-heights[xc][yc]));
                        if(newEffort<distance[nr][nc]){
                            distance[nr][nc]=newEffort;
                            pq.add(new Tuple(newEffort,nr,nc));
                        }
                    }
                }
            }
            return 0;
            }
    }
    //******************G38-Cheapest Flights with at most K
stops************************************ */
    class Solution {
    class Pair{
        int first;
        int second;
        Pair(int _first,int _second){
            this.first=_first;
            this.second=_second;
        }
    }
    class Tuple{
        int first;
        int second;
        int third;
        Tuple(int _first,int _second,int _third){
            this.first=_first;
            this.second=_second;
            this.third=_third;
        }
    }
    public int CheapestFLight(int n,int flights[][],int src,int dst,int k) {
        // Code here
        ArrayList<ArrayList<Pair>> adj=new ArrayList<>();
        for(int i=0;i<n;i++){
            adj.add(new ArrayList<>());
        }
        int m=flights.length;
        for(int i=0;i<m;i++){
            adj.get(flights[i][0]).add(new Pair(flights[i][1],flights[i][2]));
```

```
        }
        int dist[]=new int[n];
        for(int i=0;i<n;i++){
            dist[i]=(int)(1e9);
        }
        dist[src]=0;
        Queue<Tuple> q=new LinkedList<>();
        q.add(new Tuple(0,src,0));
        while(!q.isEmpty()){
            Tuple it=q.peek();
            q.remove();
            int stops=it.first;
            int node=it.second;
            int cost=it.third;
            if(stops>k){
                continue;
            }
            for(Pair iter: adj.get(node)){
                int adjNode=iter.first;
                int edgeW=iter.second;

                if(cost+edgeW<dist[adjNode]){
                    dist[adjNode]=cost+edgeW;
                    q.add(new Tuple(stops+1,adjNode,cost+edgeW));
                }
            }
        }
        if(dist[dst]==(int)(1e9)){
            return -1;
        }
        return dist[dst];
    }
}
    //*****************G39-Minimum multiplication to reach end*************** */
    class Solution {
        class Pair{
            int first;
            int second;
            Pair(int _first,int _second){
                this.first=_first;
                this.second=_second;
            }
            }
        int minimumMultiplications(int[] arr, int start, int end) {
            Queue<Pair> q=new LinkedList<>();
            q.add(new Pair(start,0));
            int mod=100000;
            int dist[]=new int[mod];
            for(int i=0;i<mod;i++){
```

```
            dist[i]=(int)(1e9);
        }
        dist[start]=0;
        while(!q.isEmpty()){
            Pair it=q.peek();
            int num=it.first;
            int steps=it.second;
            q.remove();
            for(Integer i: arr){
                int fin=(num*i)%mod;
                if(steps+1<dist[fin]){
                    dist[fin]=steps+1;
                    if(fin==end){
                        return steps+1;
                    }
                    q.add(new Pair(fin,steps+1));
                }
            }
        }
        return -1;
        }
    }
    //*****************G40-Count Paths*No of Ways to Arrive at a
Destination********************************** */
    class Solution {
    class Pair{
        int first;
        int second;
        Pair(int _first,int _second){
            this.first=_first;
            this.second=_second;
        }
    }
static int countPaths(int n, List<List<Integer>> roads) {
    // Your code here
    ArrayList<ArrayList<Pair>> adj=new ArrayList<>();
    for(int i=0;i<n;i++){
        adj.add(new ArrayList<>());
    }
    int m=roads.size();
    for(int i=0;i<m;i++){
        adj.get(roads.get(i).get(0)).add(new Pair(roads.get(i).get(1),roads.get(i).get(2)));
        adj.get(roads.get(i).get(1)).add(new Pair(roads.get(i).get(0),roads.get(i).get(2)));
    }
    PriorityQueue<Pair> pq=new PriorityQueue<Pair>((x,y)->x.first-y.first);
    int dist[]=new int[n];
    int ways[]=new int[n];
    for(int i=0;i<n;i++){
        dist[i]=(int)(1e9);
```

```
        ways[i]=0;
    }
    dist[0]=0;
    ways[0]=1;
    int mod=(int)(1e9+7);
    pq.add(new Pair(0,0));
        while(!pq.isEmpty()){
        Pair it=pq.peek();
        int dis=it.first;
        int node=it.second;
        pq.remove();
        for(Pair iter: adj.get(node)){
            int adjNode=iter.first;
            int edgeW=iter.second;

            if(dis+edgeW<dist[adjNode]){
                dist[adjNode]=dis+edgeW;
                pq.add(new Pair(dis+edgeW,adjNode));
                ways[adjNode]=ways[node];
            }
            else if(dis+edgeW==dist[adjNode]){
                ways[adjNode]=(ways[node]+ways[adjNode])%mod;
        }
    }

}
return ways[n-1]%mod;
}
}
    //*****************G41-Bellman Ford - Negative Cycle |Base
Code********************************************** */
    class Solution {
        static int[] bellman_ford(int V, ArrayList<ArrayList<Integer>> edges, int S) {
            // Write your code here
            int[] distance=new int[V];
            for(int i=0;i<V;i++){
                distance[i]=(int)(1e8);
            }
            distance[S]=0;

            for(int i=0;i<V-1;i++) {
                for(ArrayList<Integer> it: edges){
                    int u=it.get(0);
                    int v=it.get(1);
                    int wt=it.get(2);
                    if(distance[u]!=(int)(1e8)&&distance[u]+wt<distance[v]){
                        distance[v]=distance[u]+wt;
                    }
                }
```

```
            }
            for(ArrayList<Integer> it: edges){
                    int u=it.get(0);
                    int v=it.get(1);
                    int wt=it.get(2);
                    if(distance[u]!=(int)(1e8)&&distance[u]+wt<distance[v]){
                        int temp[]=new int[1];
                        temp[0]=-1;
                        return temp;
                        }
                    }
            return distance;
        }
    }
    //*****************G42-Flyod Warshall|Multiple Source-Shortest
Path********************************************** */
    class Solution{
    public void shortest_distance(int[][] matrix)
    {
        // Code here
        int n=matrix.length;
        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                if(i==j){
                    matrix[i][j]=0;
                }
                else if(matrix[i][j]==-1){
                    matrix[i][j]=(int)(1e9);
                }
            }
        }
        for(int k=0;k<n;k++){
            for(int i=0;i<n;i++){
                for(int j=0;j<n;j++){
                    matrix[i][j]=Math.min(matrix[i][j],matrix[i][k]+matrix[k][j]);
                }
            }
        }

        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                if(matrix[i][j]==(int)(1e9)){
                    matrix[i][j]=-1;
                }
            }
        }
        return;
    }
}
```

```
   //***********G43-Find the City With the Smallest Number of Neighbours at a
Threshold Distance********** */
   class Solution {
       int findCity(int n, int m, int[][] edges,int distanceThreshold)
       {
           //code here
         int dist[][]=new int[n][n];

         for(int i=0;i<n;i++){
             for(int j=0;j<n;j++){
                 dist[i][j]=Integer.MAX_VALUE;
             }
         }
         for(int i=0;i<m;i++){
             int u=edges[i][0];
             int v=edges[i][1];
             int wt=edges[i][2];
             dist[u][v]=wt;
             dist[v][u]=wt;
         }

         for(int i=0;i<n;i++){dist[i][i]=0;}
         for(int k=0;k<n;k++){
             for(int i=0;i<n;i++){
                 for(int j=0;j<n;j++){
                     if(dist[i][k]==Integer.MAX_VALUE||dist[k][j]==Integer.MAX_VALUE)
{continue;}
                     dist[i][j]=Math.min(dist[i][j],dist[i][k]+dist[k][j]);
                 }
             }
         }

         int cntcity=n;
         int cityno=-1;
         for(int city=0;city<n;city++){
             int cnt=0;
             for(int adj=0;adj<n;adj++){
                 if(dist[city][adj]<=distanceThreshold){
                     cnt++;
                 }
             }
                 if(cnt<=cntcity){
                     cntcity=cnt;
                     cityno=city;
                 }
             }
         return cityno;
       }
    }
```

```
  //*****************G45-Prim's Algorithm for Minimum spanning
tree*************************** */
    class Solution{
    static class Pair{
        int node;
        int distance;
        Pair(int _first,int _second){
            this.node=_first;
            this.distance=_second;
        }
    }
    //Function to find sum of weights of edges of the Minimum Spanning Tree.
    static int spanningTree(int V, ArrayList<ArrayList<ArrayList<Integer>>> adj)
    {
        // Add your code here
        PriorityQueue<Pair> pq=new PriorityQueue<Pair>((x,y)-> x.distance-y.distance);
        boolean vis[]=new boolean[V];
        for(int i=0;i<V;i++){
            vis[i]=false;
        }
        pq.add(new Pair(0,0));
        int sum=0;
        while(!pq.isEmpty()){
            int wt=pq.peek().distance;
            int node=pq.peek().node;

            pq.remove();

            if(vis[node]==true)
                continue;
            vis[node]=true;
            sum+=wt;

            for(int i=0;i<adj.get(node).size();i++){
                int eW=adj.get(node).get(i).get(1);
                int adjNode=adj.get(node).get(i).get(0);

                if(vis[adjNode]==false){
                    pq.add(new Pair(adjNode,eW));
                }

            }
        }
        return sum;

    }
}
 //******************G46-Disjoint
Set************************************************************ */
```

```
    class DisjointSet {
    List<Integer> rank = new ArrayList<>();
    List<Integer> parent = new ArrayList<>();
    List<Integer> size = new ArrayList<>();
    public DisjointSet(int n) {
        for(int i = 0;i<=n;i++) {
            rank.add(0);
            parent.add(i);
            size.add(1);
        }
    }
    public int findUPar(int node) {
        if(node == parent.get(node)) {
            return node;
        }
        int ulp = findUPar(parent.get(node));
        parent.set(node, ulp);
        return parent.get(node);
    }
    public void unionByRank(int u, int v) {
        int ulp_u = findUPar(u);
        int ulp_v = findUPar(v);
        if(ulp_u == ulp_v) return;
        if(rank.get(ulp_u) < rank.get(ulp_v)) {
            parent.set(ulp_u, ulp_v);
        }
        else if(rank.get(ulp_v) < rank.get(ulp_u)) {
            parent.set(ulp_v, ulp_u);
        }
        else {
            parent.set(ulp_v, ulp_u);
            int rankU = rank.get(ulp_u);
            rank.set(ulp_u, rankU + 1);
        }
    }
    public void unionBySize(int u, int v) {
        int ulp_u = findUPar(u);
        int ulp_v = findUPar(v);
        if(ulp_u == ulp_v) return;
        if(size.get(ulp_u) < size.get(ulp_v)) {
            parent.set(ulp_u, ulp_v);
            size.set(ulp_v, size.get(ulp_v) + size.get(ulp_u));
        }
        else {
            parent.set(ulp_v, ulp_u);
            size.set(ulp_u, size.get(ulp_u) + size.get(ulp_v));
        }
    }
}
```

```
//*****************G47-Krusakal's Minimum Spanning Tree**************************/
class Solution{
class DisjointSet {
List<Integer> rank = new ArrayList<>();
List<Integer> parent = new ArrayList<>();
List<Integer> size = new ArrayList<>();
public DisjointSet(int n) {
    for(int i = 0;i<=n;i++) {
        rank.add(0);
        parent.add(i);
        size.add(1);
    }
}
public int findUPar(int node) {
    if(node == parent.get(node)) {
        return node;
    }
    int ulp = findUPar(parent.get(node));
    parent.set(node, ulp);
    return parent.get(node);
}
public void unionByRank(int u, int v) {
    int ulp_u = findUPar(u);
    int ulp_v = findUPar(v);
    if(ulp_u == ulp_v) return;
    if(rank.get(ulp_u) < rank.get(ulp_v)) {
        parent.set(ulp_u, ulp_v);
    }
    else if(rank.get(ulp_v) < rank.get(ulp_u)) {
        parent.set(ulp_v, ulp_u);
    }
    else {
        parent.set(ulp_v, ulp_u);
        int rankU = rank.get(ulp_u);
        rank.set(ulp_u, rankU + 1);
    }
}
public void unionBySize(int u, int v) {
    int ulp_u = findUPar(u);
    int ulp_v = findUPar(v);
    if(ulp_u == ulp_v) return;
    if(size.get(ulp_u) < size.get(ulp_v)) {
        parent.set(ulp_u, ulp_v);
        size.set(ulp_v, size.get(ulp_v) + size.get(ulp_u));
    }
    else {
        parent.set(ulp_v, ulp_u);
        size.set(ulp_u, size.get(ulp_u) + size.get(ulp_v));
    }
```

```
    }
}
class Edge implements Comparable<Edge>{
        int src,dest,weight;
        Edge(int _src,int _dest,int _weight){
            this.src=_src;this.dest=_dest;this.weight=_weight;
        }
        public int compareTo(Edge compareEdge){
            return this.weight-compareEdge.weight;
        }
    }
    //Function to find sum of weights of edges of the Minimum Spanning Tree.
    static int spanningTree(int V, ArrayList<ArrayList<ArrayList<Integer>>> adj)
    {
        List<Edge> edges=new ArrayList<Edge>();
        for(int i=0;i<V;i++){
            for(int j=0;j<adj.get(i).size();j++){
             int adjN=adj.get(i).get(j).get(0);
             int wt=adj.get(i).get(j).get(1);
             int node=i;
             Edge temp=new Edge(i,adjN,wt);
             edges.add(temp);
            }
        }
        DisjointSet ds=new DisjointSet(V);
        Collections.sort(edges);
        int mstwt=0;
        for(int i=0;i<edges.size();i++){
            int wt=edges.get(i).weight;
            int u=edges.get(i).src;
            int v=edges.get(i).dest;

            if(ds.findUPar(u)!=ds.findUPar(v)){
                mstwt+=wt;
                ds.unionBySize(u,v);
            }
        }
        return mstwt;
    }
}
    //****************G48-Number of Provinces************************************* */
    class Solution {
        class DisjointSet {
        List<Integer> rank = new ArrayList<>();
        List<Integer> parent = new ArrayList<>();
        List<Integer> size = new ArrayList<>();
        public DisjointSet(int n) {
            for(int i = 0;i<=n;i++) {
                rank.add(0);
```

```
            parent.add(i);
            size.add(1);
        }
    }
    public int findUPar(int node) {
        if(node == parent.get(node)) {
            return node;
        }
        int ulp = findUPar(parent.get(node));
        parent.set(node, ulp);
        return parent.get(node);
    }
    public void unionByRank(int u, int v) {
        int ulp_u = findUPar(u);
        int ulp_v = findUPar(v);
        if(ulp_u == ulp_v) return;
        if(rank.get(ulp_u) < rank.get(ulp_v)) {
            parent.set(ulp_u, ulp_v);
        }
        else if(rank.get(ulp_v) < rank.get(ulp_u)) {
            parent.set(ulp_v, ulp_u);
        }
        else {
            parent.set(ulp_v, ulp_u);
            int rankU = rank.get(ulp_u);
            rank.set(ulp_u, rankU + 1);
        }
    }
    public void unionBySize(int u, int v) {
        int ulp_u = findUPar(u);
        int ulp_v = findUPar(v);
        if(ulp_u == ulp_v) return;
        if(size.get(ulp_u) < size.get(ulp_v)) {
            parent.set(ulp_u, ulp_v);
            size.set(ulp_v, size.get(ulp_v) + size.get(ulp_u));
        }
        else {
            parent.set(ulp_v, ulp_u);
            size.set(ulp_u, size.get(ulp_u) + size.get(ulp_v));
        }
    }
}
    static int numProvinces(ArrayList<ArrayList<Integer>> adj, int V) {
        // code here
        DisjointSet ds=new DisjointSet(V);
        for(int i=0;i<V;i++){
            for(int j=0;j<V;j++){
                if(adj.get(i).get(j)==1){
                    ds.unionBySize(i,j);
```

```
                }
            }
        }
        int cnt=0;
        for(int i=0;i<V;i++){
            if(ds.parent.get(i)==i){
                cnt++;
            }
        }
        return cnt;
    }
}
//****************G-49-Number of Operations to Make Network Connected - DSU********/
class Solution {
class DisjointSet {
    List<Integer> rank = new ArrayList<>();
    List<Integer> parent = new ArrayList<>();
    List<Integer> size = new ArrayList<>();
    public DisjointSet(int n) {
        for(int i = 0;i<=n;i++) {
            rank.add(0);
            parent.add(i);
            size.add(1);
        }
    }
    public int findUPar(int node) {
        if(node == parent.get(node)) {
            return node;
        }
        int ulp = findUPar(parent.get(node));
        parent.set(node, ulp);
        return parent.get(node);
    }
    public void unionByRank(int u, int v) {
        int ulp_u = findUPar(u);
        int ulp_v = findUPar(v);
        if(ulp_u == ulp_v) return;
        if(rank.get(ulp_u) < rank.get(ulp_v)) {
            parent.set(ulp_u, ulp_v);
        }
        else if(rank.get(ulp_v) < rank.get(ulp_u)) {
            parent.set(ulp_v, ulp_u);
        }
        else {
            parent.set(ulp_v, ulp_u);
            int rankU = rank.get(ulp_u);
            rank.set(ulp_u, rankU + 1);
        }
```

```
        }
        public void unionBySize(int u, int v) {
            int ulp_u = findUPar(u);
            int ulp_v = findUPar(v);
            if(ulp_u == ulp_v) return;
            if(size.get(ulp_u) < size.get(ulp_v)) {
                parent.set(ulp_u, ulp_v);
                size.set(ulp_v, size.get(ulp_v) + size.get(ulp_u));
            }
            else {
                parent.set(ulp_v, ulp_u);
                size.set(ulp_u, size.get(ulp_u) + size.get(ulp_v));
            }
        }
    }
        public int Solve(int n, int[][] edge) {
            // Code here
            DisjointSet ds=new DisjointSet(n);
            int cntExtra=0;
            int m=edge.length;
            for(int i=0;i<m;i++){
                int u=edge[i][0];
                int v=edge[i][1];
                if(ds.findUPar(u)==ds.findUPar(v)){
                    cntExtra++;
                }
                else
                {
                    ds.unionBySize(u,v);
                }
            }
            int cntC=0;
            for(int i=0;i<n;i++){
                if(ds.parent.get(i)==i)
                {cntC++;}
            }

            int ans=cntC-1;
            if(cntExtra>=ans)
            {return ans;}
            return -1;

        }
    }
}
//************************************STRIVER GRAPH
PLAYLIST************************************ */
//Breadth First Search
//Depth First Search
```

```
//Number of Provinces
//Number of Islands
//Flood Fill
//Rotten Oranges
//Detect a Cycle DFS
//Detect a Cycle BFS
//Distance of nearest 1
//Surrounded Regions with X's
//Number of Enclaves
//Number of Distinct Islands
//Bipartite Graph DFS
//Bipartite Graph BFS
//Detect a Cycle in Directed Graph DFS
//Detect a Cycle in Directed Graph BFS(Kahn's Algo)
//Topological Sort BFS
//Topological Sort DFS
//Djikstra's Algo
//Printing Djikstra
//Shortest Distance in a Binary Maze
//BellMan Ford
//Flyod Warshal
//Count Number of ways to arrive at a destination
//Minimum Multiplication to  reach end
//Cheapest flight with at most K stops
//Path With Minimum Effort
//Find the city with max no of cities under threshold
//Course Schedule I and II
//Shortest Path in DAG
//Shortest Path in undirected graph with unit dist
//Alien Dictionary
//Word Ladder I
//Word Ladder II
//Eventual Safe States DFS
//Eventual Safe States BFS



//**************Snakes and Ladder************************ */
class Solution{
    class Pair{
    int dest,dist;
        Pair(int x,int y)
        {
            dest=x;
            dist=y;
        }
    }
    public int minThrow(int N,int arr[])
    {
```

```
        int moves[]=new int[35];
        boolean vis[]=new boolean[35];
        for(int i=0;i<35;i++){moves[i]=-1;vis[i]=false;}
        for(int i=0;i<2*N;i+=2){moves[arr[i]]=arr[i+1];}
        Queue<Pair> q=new LinkedList<>();
        Pair p=new Pair(0,0);
        q.add(new Pair(1,0));
        vis[1]=true;
        while(!q.isEmpty())
        {
            p=q.peek();
            int src=p.dest;
            int step= p.dist;
            if(src==30){break;}
            q.poll();
            for(int i=src+1;i<=src+6&&i<=30;i++)
            {
                Pair temp=new Pair(0,0);
                if(vis[i]==false)
                {
                    temp.dist=step+1;
                    vis[i]=true;
                    if(moves[i]!=-1){temp.dest=moves[i];}
                    else{temp.dest=i;}
                    q.add(temp);
                }
            }
        }
        return p.dist;
    }
}
//*************Knight Steps************************** */
class Solution{
    class Pair{
        int x,y,s;
        Pair(int X,int Y,int S){x=X;y=Y;s=S;}
    }
    class Solution
    {
        //Function to find out minimum steps Knight needs to reach target position.
        public int minStepToReachTarget(int KnightPos[], int TargetPos[], int N)
        {
            // Code here
            int sx=KnightPos[0],sy=KnightPos[1];
            int Dx=TargetPos[0],Dy=TargetPos[1];
            if(sx==Dx&&sy==Dy)return 0;
            int dx[]=new int[]{-2,-1,1,2,2,1,-1,-2};
            int dy[]=new int[]{-1,-2,-2,-1,1,2,2,1};
            boolean visited[][]=new boolean[N+1][N+1];
```

```
            for(boolean rows[]: visited){Arrays.fill(rows,false);}
            Queue<Pair> q=new LinkedList<>();
            q.add(new Pair(sx,sy,0));
            visited[sx][sy]=true;
            while(!q.isEmpty()){
                Pair it=q.remove();
                int xc=it.x;
                int yc=it.y;
                int steps=it.s;
                for(int i=0;i<8;i++){
                    int nr=xc+dx[i];
                    int nc=yc+dy[i];
                    if(nr>0&&nr<=N&&nc>0&&nc<=N&&visited[nr][nc]==false){
                        if(nr==Dx&&nc==Dy){return steps+1;}
                        visited[nr][nc]=true;
                        q.add(new Pair(nr,nc,steps+1));
                    }
                }
            }
            return -1;
        }
    }
}
//*********Walls and Gates********** */
class Solution  {
    class Tuple{
        int row;
        int col;
        int steps;
        Tuple(int _row,int _col,int _steps){
            this.row=_row;
            this.col=_col;
            this.steps=_steps;
        }
    }
    public int[][] nearest(int[][] grid)
    {
        // Code here
        int n=grid.length;
        int m=grid[0].length;
        Queue<Tuple> q=new LinkedList<>();
        boolean visited[][]=new boolean[n][m];
        int distance[][]=new int[n][m];
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                if(grid[i][j]==0){
                    q.add(new Tuple(i,j,0));
                    visited[i][j]=true;
                }
```

```
                else{
                    visited[i][j]=false;
                }
            }
        }
        int dx[]={-1,0,1,0};
        int dy[]={0,1,0,-1};
        while(!q.isEmpty()){
            int r=q.peek().row;
            int c=q.peek().col;
            int st=q.peek().steps;
            q.remove();
            distance[r][c]=st;
            for(int i=0;i<4;i++){
                int nr=r+dx[i];
                int nc=c+dy[i];
                if(nr>=0&&nr<n&&nc>=0&&nc<m&&visited[nr][nc]==false&&grid[nr][nc]==INF)
{
                    visited[nr][nc]=true;
                    q.add(new Tuple(nr,nc,st+1));
                }
            }
        }
        return distance;
    }
}
//***********Surrounded Regions****************** */
//***********Pacific Atlantic Water********* */
class Solution {
    class Tuple{
        int row;
        int col;
        int ho;
        Tuple(int _row,int _col,int _ho){
            this.row=_row;
            this.col=_col;
            this.ho=_ho;
        }
    }
    public List<List<Integer>> pacificAtlantic(int[][] heights) {
        int n=heights.length;
        int m=heights[0].length;
        Queue<Tuple> q1=new LinkedList<>();
        Queue<Tuple> q2=new LinkedList<>();
        boolean visited1[][]=new boolean[n][m];
        boolean visited2[][]=new boolean[n][m];
        for(int i=0;i<m;i++){
            q1.add(new Tuple(0,i,heights[0][i]));
            q2.add(new Tuple(n-1,i,heights[n-1][i]));
```

```
            visited1[0][i]=true;
            visited2[n-1][i]=true;
        }
        for(int i=0;i<n;i++){
            q1.add(new Tuple(i,0,heights[i][0]));
            q2.add(new Tuple(i,m-1,heights[i][m-1]));
            visited1[i][0]=true;
            visited2[i][m-1]=true;
        }
        int dx[]={-1,0,1,0};
        int dy[]={0,1,0,-1};
        while(!q1.isEmpty()){
            int r=q1.peek().row;
            int c=q1.peek().col;
            int hxy=q1.peek().ho;
            q1.remove();
            for(int i=0;i<4;i++){
                int nr=r+dx[i];
                int nc=c+dy[i];

if(nr>=0&&nr<n&&nc>=0&&nc<m&&visited1[nr][nc]==false&&heights[nr][nc]>=hxy){
                    visited1[nr][nc]=true;
                    q1.add(new Tuple(nr,nc,heights[nr][nc]));
                }
            }
        }
        while(!q2.isEmpty()){
            int r=q2.peek().row;
            int c=q2.peek().col;
            int hxy=q2.peek().ho;
            q2.remove();

            for(int i=0;i<4;i++){
                int nr=r+dx[i];
                int nc=c+dy[i];

if(nr>=0&&nr<n&&nc>=0&&nc<m&&visited2[nr][nc]==false&&heights[nr][nc]>=hxy){
                    visited2[nr][nc]=true;
                    q2.add(new Tuple(nr,nc,heights[nr][nc]));
                }
            }
        }
        List<List<Integer>> ans=new ArrayList<>();
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                if(visited1[i][j]&&visited2[i][j]){
                    List<Integer> ls=new ArrayList<>();
                    ls.add(i);
                    ls.add(j);
```

```
                ans.add(ls);
            }
          }
        }
        return ans;

    }
}
//********************Jump Game IV************************************************** */
class Solution {
    public int minJumps(int[] arr) {
      Map<Integer,ArrayList<Integer>> mp=new HashMap<>();
      int n=arr.length;
      boolean vis[]=new boolean[n];
      Queue<Integer> q=new LinkedList<>();
      for(int i=0;i<n;i++){
          if(mp.containsKey(arr[i])==false){
              ArrayList<Integer> newlist=new ArrayList<>();
              newlist.add(i);
              mp.put(arr[i],newlist);
          }
          else{
              ArrayList<Integer> oldlist=mp.get(arr[i]);
              oldlist.add(i);
              mp.put(arr[i],oldlist);
          }
      }
      q.add(0);
      vis[0]=true;
      int st=0;
      while(!q.isEmpty()){
          int s=q.size();
          for(int i=0;i<s;i++){
              Integer ind=q.remove();
              if(ind==n-1){return st;}
              int ele=arr[ind];
              if(ind+1<n&&vis[ind+1]==false){q.add(ind+1);vis[ind+1]=true;}
              if(ind-1>=0&&vis[ind-1]==false){q.add(ind-1);vis[ind-1]=true;}
              if(mp.containsKey(arr[ind])==true){
                  for(Integer k: mp.get(arr[ind])){
                      q.add(k);
                      vis[k]=true;
                  }
              }
              mp.remove(arr[ind]);
          }
          st++;
        }
        return n-1;
```

```
    }
}
//*****************Jump Game I********************************************** */
class Solution {
    public boolean canJump(int[] arr) {
      int n=arr.length;
      boolean vis[]=new boolean[n];
      Queue<Integer> q=new LinkedList<>();
      q.add(0);
      vis[0]=true;
      while(!q.isEmpty()){
        Integer ind=q.remove();
        if(ind==n-1){return true;}
        int steps=arr[ind];
        for(int ir=0;ir<=steps;ir++){
            if(ind+ir>=n-1){return true;}
            int nl=ind+ir;
                if(vis[nl]==false){
                    vis[nl]=true;
                    q.add(nl);
                }
        }
      }
      return false;
    }
}
//****************Jump Game II********************************************************** */
class Solution {
    public int jump(int[] arr) {
      if(arr.length==1){return 0;}
      int n=arr.length;
      boolean vis[]=new boolean[n];
      Queue<Integer> q=new LinkedList<>();
      q.add(0);
      vis[0]=true;
      int st=1;
      while(!q.isEmpty()){
       int s=q.size();
       for(int j=0;j<s;j++)
       {
        Integer ind=q.remove();
        if(ind==n-1){return st;}
        int steps=arr[ind];
        for(int ir=0;ir<=steps;ir++){
            if(ind+ir>=n-1){return st;}
            int nl=ind+ir;
                if(vis[nl]==false){
                    vis[nl]=true;
                    q.add(nl);
```

```
            }
          }
        }
        st++;
      }
      return -1;
    }
}
//****************Jump Game III********************************************************** */
class Solution {
    public boolean canReach(int[] arr, int start) {
      int n=arr.length;
      boolean vis[]=new boolean[n];
      Queue<Integer> q=new LinkedList<>();
      q.add(start);
      vis[start]=true;
      while(!q.isEmpty()){
        Integer ind=q.remove();
        if(arr[ind]==0){return true;}
        int steps=arr[ind];
        int nl1=ind+steps;
        int nl2=ind-steps;
            if(nl1>=0&&nl1<n){
                if(vis[nl1]==false){
                vis[nl1]=true;
                q.add(nl1);
                }
            }
            if(nl2>=0&&nl2<n){
                if(vis[nl2]==false){
                vis[nl2]=true;
                q.add(nl2);
                }
            }
        }
        return false;
    }
}
```