Graph and Types***

09 October 2024 21:15

```
class Solution {
   static long count(int n)
   {
      if(n==1)
      {
         return 1;
      }
      if(n==2)
      {
         return 2;

      }
      long b=4;
      long a=2;
      for(int i=3;i<=n;i++)
      {
         a=a*b;
         b=b*2;
      }
      return a;
   // code here

  }
}
```

#copied code

```
class Solution {
   static long count(int n)
   {

      return (long)Math.pow(2, (n*(n-1))/2);

  }
}
```

Graph Representation | C++

09 October 2024 21:45

#copied Code

```
vector<vector<int>> printGraph(int V, vector<pair<int,int>>edges)
  {
    vector<vector<int>>ans(V);

    for(int i=0;i<edges.size();i++)
    {
       int u=edges[i].first;
       int v=edges[i].second;

       ans[u].push_back(v);
       ans[v].push_back(u);
    }
    return ans;
    // Code here
  }
```

Graph Representation | Java

09 October 2024 21:45

#Copied code

```
class Solution {
   public List<List<Integer>> printGraph(int V, int edges[][])
   {
      List<List<Integer>> list=new ArrayList<List<Integer>>();


      for(int i=0;i<V;i++){
         list.add(new ArrayList<>());
      }

      for(int i=0;i<edges.length;i++){=-0=
         list.get(edges[i][0]).add(edges[i][1]);
          list.get(edges[i][1]).add(edges[i][0]);
      }
      return list;


   }
}
```

```
import java.util.*;

class Solution {

   public List<List<Pair<Integer, Integer>>> printGraph(int V, int edges[][]) {
      // Create an adjacency list with each entry being a list of pairs (neighbor, weight)
      List<List<Pair<Integer, Integer>>> list = new ArrayList<>();

      // Initialize the adjacency list with empty lists for each vertex
      for (int i = 0; i < V; i++) {
         list.add(new ArrayList<>());
      }

      // Add the edges to the adjacency list
      for (int i = 0; i < edges.length; i++) {
         int u = edges[i][0];  // Source vertex
         int v = edges[i][1];  // Destination vertex
         int w = edges[i][2];  // Weight of the edge

         // Since the graph is undirected, add the edge in both directions
         list.get(u).add(new Pair<>(v, w));
         list.get(v).add(new Pair<>(u, w));
      }

      return list;
   }
}
```

Connected Components | Logic Explanation

09 October 2024 21:45

#GIVING WRONG ANS

```
static int numProvinces(ArrayList<ArrayList<Integer>> adj, int V)
  {

     int n=adj.size();
     boolean[]vis=new boolean[];
     int cnt=0;
     for(int i=0;i<V;i++)
     {
        if(vis[i]==false)
        {
           dfs(i,adj,vis);
           cnt+=1;
        }
     }

     return cnt;

  }
  public static void dfs(int start,ArrayList<ArrayList<Integer>> adj,boolean[] vis)
  {

     vis[start]=true;

     for(int it:adj.get(start))
     {
        if(vis[it]==false)
        {
           dfs(it,adj,vis);
        }
     }
  }
```

#CORRECT ANS WHY

```
static int numProvinces(ArrayList<ArrayList<Integer>> adj, int V)
  {

     int n=adj.size();
     boolean[]vis=new boolean[n];
     int cnt=0;
     for(int i=0;i<V;i++)
     {
        if(vis[i]==false)
        {
           dfs(i,adj,vis,V);
           cnt+=1;
        }
```

```
    }

    return cnt;

}
public static void dfs(int start,ArrayList<ArrayList<Integer>> adj,boolean[] vis,int V)
{

    vis[start]=true;

    for(int it=0;it<V;it++)
    {
        if(vis[it]==false&&adj.get(start).get(it)==1)  //WHY THIS CODE
        {
            dfs(it,adj,vis,V);
        }
    }
}
```

BFS

09 October 2024 21:45

#C++

```
vector<int> bfsOfGraph(int V, vector<int> adj[])
{
    vector<bool>vis(V,false);
    queue<int>q;
    q.push(0);
    vis[0]=true;
    vector<int>ans;

    while(q.size()>0)
    {
        int y=q.front();
        ans.push_back(y);
        q.pop();
        for(auto x:adj[y])
        {
            if(!vis[x])
            {
                q.push(x);
                vis[x]=true;
            }
        }
    }


    return ans;
}
```

#JAVA CDE

```
public ArrayList<Integer> bfsOfGraph(int V, ArrayList<ArrayList<Integer>> adj)
  {
     ArrayList<Integer>ans=new ArrayList<>();
     int n = adj.size();
    boolean[]vis =new boolean[n];
    vis[0]=true;

     Queue<Integer>q=new LinkedList<>();
     q.add(0);
     while(q.size()>0)
     {
        int a=q.peek();
        q.remove();
        ans.add(a);

        for(int it:adj.get(a))
        {
```

```
            if(vis[it]==false)
            {
               vis[it]=true;
               q.add(it);
            }

         }
      }
      return ans;

   }
```

DFS

09 October 2024 21:46

#JAVA CODE

```
public ArrayList<Integer> dfsOfGraph(int V, ArrayList<ArrayList<Integer>> adj)
  {
    ArrayList<Integer>ans=new ArrayList<>();
    int n=adj.size();
    boolean[]vis=new boolean[n];
    for(int i=0;i<V;i++)
    {
      if(vis[i]==false)
      {
        dfs(i,adj,ans,vis);
      }
    }

    return ans;
    // Code here
  }

  public void dfs(int start,ArrayList<ArrayList<Integer>> adj,ArrayList<Integer> ans, boolean[] vis)
  {
    ans.add(start);
    vis[start]=true;

    for(int it:adj.get(start))
    {
      if(vis[it]==false)
      {
        dfs(it,adj,ans,vis);
      }
    }
  }
```

#C++ CODE

```
void dfs(vector<int>adj[],int src,vector<bool>&vis,vector<int>&ans)
  {
    vis[src]=true;
    ans.push_back(src);
    for(auto x:adj[src])
    {
      if(!vis[x])
      {
        dfs(adj,x,vis,ans);
      }
```

```
    }
}

vector<int> dfsOfGraph(int V, vector<int> adj[]) {
    vector<bool>vis(V,false);

    vector<int>ans;
    for(int i=0;i<V;i++)
    {
        if(!vis[i])
        {
            dfs(adj,i,vis,ans);
        }
    }
    return ans;
}
```

Number of provinces (leetcode)

10 October 2024 20:44

isConnected = [[1, 1, 0],
[1, 1, 0],
[0, 0, 1]]

```
class Solution {
    public void dfs(int start,int [][] isConnected, boolean[]vis)
    {
        vis[start]=true;
        for(int i=0;i<isConnected.length;i++)
        {
            if(vis[i]==false&&isConnected[start][i]==1)
            {
                dfs(i,isConnected,vis);
            }
        }
    }
    public int findCircleNum(int[][] isConnected)
    {
        int n=isConnected.length;
        boolean[]vis=new boolean[n];
        int cnt=0;
        for(int i=0;i<n;i++)
        {
            if(vis[i]==false)
            {
                dfs(i,isConnected,vis);
                cnt++;
            }
        }
        return cnt;

    }
}
```

```
import java.util.*;
class Solution {
   // dfs traversal function
   private static void dfs(int node,
      ArrayList<ArrayList<Integer>> adjLs ,
      int vis[]) {
       vis[node] = 1;
       for(Integer it: adjLs.get(node)) {
           if(vis[it] == 0) {
```

```
                dfs(it, adjLs, vis);
            }
        }
    }
    static int numProvinces(ArrayList<ArrayList<Integer>> adj, int V) {
        ArrayList<ArrayList<Integer>> adjLs = new ArrayList<ArrayList<Integer>>();
        for(int i = 0;i<V;i++) {
            adjLs.add(new ArrayList<Integer>());
        }

        // to change adjacency matrix to list
        for(int i = 0;i<V;i++) {
            for(int j = 0;j<V;j++) {
                // self nodes are not considered
                if(adj.get(i).get(j) == 1 && i != j) {
                    adjLs.get(i).add(j);
                    adjLs.get(j).add(i);
                }
            }
        }
        int vis[] = new int[V];
        int cnt = 0;
        for(int i = 0;i<V;i++) {
            if(vis[i] == 0) {
                cnt++;
                dfs(i, adjLs, vis);
            }
        }
        return cnt;
    }
    public static void main(String[] args)
    {
// adjacency matrix
        ArrayList<ArrayList<Integer> > adj = new ArrayList<ArrayList<Integer> >();
adj.add(new ArrayList<Integer>());
        adj.get(0).add(0, 1);
        adj.get(0).add(1, 0);
        adj.get(0).add(2, 1);
        adj.add(new ArrayList<Integer>());
        adj.get(1).add(0, 0);
        adj.get(1).add(1, 1);
        adj.get(1).add(2, 0);
        adj.add(new ArrayList<Integer>());
        adj.get(2).add(0, 1);
        adj.get(2).add(1, 0);
        adj.get(2).add(2, 1);

        Solution ob = new Solution();
        System.out.println(ob.numProvinces(adj,3));
    }
};
```

From <https://takeuforward.org/data-structure/number-of-provinces/>

Connected Components Problem in Matrix

10 October 2024 20:45

```
static int numProvinces(ArrayList<ArrayList<Integer>> adj, int V)
  {

     int n=adj.size();
     boolean[]vis=new boolean[n];
     int cnt=0;
     for(int i=0;i<V;i++)
     {
        if(vis[i]==false)
        {
           dfs(i,adj,vis,V);
           cnt+=1;
        }
     }

     return cnt;

  }
  public static void dfs(int start,ArrayList<ArrayList<Integer>> adj,boolean[] vis,int V)
  {

     vis[start]=true;

     for(int it=0;it<V;it++)
     {
        if(vis[it]==false&&adj.get(start).get(it)==1)
        {
           dfs(it,adj,vis,V);
        }
     }
  }
```

Rotten Oranges

10 October 2024 20:45

#code is okay only how should I pass queue in any func

```
class Solution {
    public class pair{
        int a;
        int b;
        pair(int a,int b)
        {
            this.a=a;
            this.b=b;
        }
    }
    public int orangesRotting(int[][] grid)
    {
        Queue<pair>q=new LinkedList<>();
        int r=grid.length;
        int c=grid[0].length;
        for(int i=0;i<grid.length;i++)
        {
            for(int j=0;j<grid[0].length;j++)
            {
                if(grid[i][j]==2)
                {
                    q.add(new pair(i,j));
                }
            }
        }
        int cnt=q.size();
        int time=0;
        while(cnt>0)
        {
            for(int it=0;it<cnt;it++)
            {
                pair p=q.poll();
                int x=p.a;
                int y=p.b;

            push_neighbour(x,y,q,grid,r,c);

            }
            cnt=q.size();
            if(cnt>0)
            {
                time++;
            }
        }
        for(int i=0;i<grid.length;i++)
        {
            for(int j=0;j<grid[0].length;j++)
            {
                if(grid[i][j]==1)
                {
                    return -1;
                }
            }
        }
        return time;

    }
```

```
    void push_neighbour(int x, int y, Queue<pair> q, int[][] grid, int r, int c)
    {
        int[]dx={-1,1,0,0};
        int[]dy={0,0,-1,1};
        for(int i=0;i<4;i++)
        {
            if(x+dx[i]<0||x+dx[i]>=r||y+dy[i]<0||y+dy[i]>=c||grid[x+dx[i]][y+dy[i]]!=1)
            {
                continue;
            }
            q.add(new pair(x+dx[i],y+dy[i]));
            grid[x+dx[i]][y+dy[i]]=2;
        }
    }
}
```

Flood fill

10 October 2024 20:45

```
class Solution {
    //int[][]ans;
    void dfs(int[][] image,int sr,int sc,int color,int newc,int r,int c)
    {
        if(sr<0||sc<0||sr>=r||sc>=c||image[sr][sc]==newc||image[sr][sc]!=color)
        {
            return;
        }
        image[sr][sc]=newc;
        dfs(image,sr-1,sc,color,newc,r,c);
        dfs(image,sr+1,sc,color,newc,r,c);
        dfs(image,sr,sc-1,color,newc,r,c);
        dfs(image,sr,sc+1,color,newc,r,c);
    }
    public int[][] floodFill(int[][] image, int sr, int sc, int color)
    {

        int r=image.length;
        int c=image[0].length;
        int old=image[sr][sc]; //here fetching oldc
        dfs(image,sr,sc,old,color,r,c);
        return image;

    }
}
```

Cycle Detection in unirected Graph (bfs)

10 October 2024 20:45

```
import java.util.*;
class Solution
{
  static boolean checkForCycle(ArrayList<ArrayList<Integer>> adj, int s,
        boolean vis[], int parent[])
  {
    Queue<Node> q =  new LinkedList<>(); //BFS
    q.add(new Node(s, -1));
    vis[s] =true;

    // until the queue is empty
    while(!q.isEmpty())
    {
      // source node and its parent node
      int node = q.peek().first;
      int par = q.peek().second;
      q.remove();

      // go to all the adjacent nodes
      for(Integer it: adj.get(node))
      {
        if(vis[it]==false)
        {
          q.add(new Node(it, node));
          vis[it] = true;
        }

         // if adjacent node is visited and is not its own parent node
        else if(par != it) return true;
      }
    }

    return false;
  }

  // function to detect cycle in an undirected graph
  public boolean isCycle(int V, ArrayList<ArrayList<Integer>> adj)
  {
     boolean vis[] = new boolean[V];
     Arrays.fill(vis,false);
     int parent[] = new int[V];
     Arrays.fill(parent,-1);

     for(int i=0;i<V;i++)
        if(vis[i]==false)
           if(checkForCycle(adj, i,vis, parent))
              return true;

     return false;
  }
```

```
    public static void main(String[] args)
    {
        ArrayList<ArrayList<Integer>> adj = new ArrayList<>();
        for (int i = 0; i < 4; i++) {
            adj.add(new ArrayList < > ());
        }
        adj.get(1).add(2);
        adj.get(2).add(1);
        adj.get(2).add(3);
        adj.get(3).add(2);

        Solution obj = new Solution();
        boolean ans = obj.isCycle(4, adj);
        if (ans)
            System.out.println("1");
        else
            System.out.println("0");
    }
}
class Node {
    int first;
    int second;
    public Node(int first, int second) {
        this.first = first;
        this.second = second;
    }
}
/
```

From <https://takeuforward.org/data-structure/detect-cycle-in-an-undirected-graph-using-bfs/>

Cycle Detection in undirected Graph (dfs)

10 October 2024 20:45

```
boolean dfs(ArrayList<ArrayList<Integer>> adj,int scr,int par,boolean[] vis)
  {
     vis[scr]=true;
     for(int it:adj.get(scr))
     {
        if(vis[it]==false)
        {
           if(dfs(adj,it,scr,vis))
           {
              return true;
           }
        }
        else if(it!=par||it==scr) //agar node vis hai aur it !=par ya self loop hai
        {
           return true;
        }
     }
     return false;
  }
  public boolean isCycle(int V, ArrayList<ArrayList<Integer>> adj)
  {
     int V=adj.size();
     boolean[]vis=new boolean[V];
     for(int i=0;i<V;i++)
     {
        if(vis[i]==false)
        {
           boolean flag=dfs(adj,i,-1,vis);
           if(flag==true)
           {
              return true;
           }
        }
     }
     return false;
  }
```

0/1 Matrix (Bfs Problem)

10 October 2024 20:46

```
class Solution {
    public class pair{
        int a;
        int b;
        pair(int a,int b)
        {
            this.a=a;
            this.b=b;
        }
    }
    public int[][] updateMatrix(int[][] mat)
    {
        Queue<pair>q=new LinkedList<>();
        int r=mat.length;
        int c=mat[0].length;
        boolean[][]vis=new boolean[r][c];
        for(int i=0;i<r;i++)
        {
            for(int j=0;j<c;j++)
            {
                if(mat[i][j]==0)
                {
                    q.add(new pair(i,j));
                    vis[i][j]=true;
                }
            }
        }
        //int cnt=q.size();
        int put=1;
        while(q.size()>0)
        {
            int cnt1=q.size();
            for(int k=0;k<cnt1;k++)
            {
                pair p=q.poll();
                int x=p.a;
                int y=p.b;

                int[]dx={-1,1,0,0};
                int[]dy={0,0,-1,1};
                for(int i=0;i<4;i++)
                {

                    if(x+dx[i]<0||x+dx[i]>=r||y+dy[i]<0||y+dy[i]>=c||vis[x+dx[i]]
[y+dy[i]]==true)
                    {
                      continue;
                    }

                    q.add(new pair(x+dx[i],y+dy[i]));
                    mat[x+dx[i]][y+dy[i]]=put;
                    vis[x+dx[i]][y+dy[i]]=true;
                }
            }

            put++;
        }
        return mat;
```

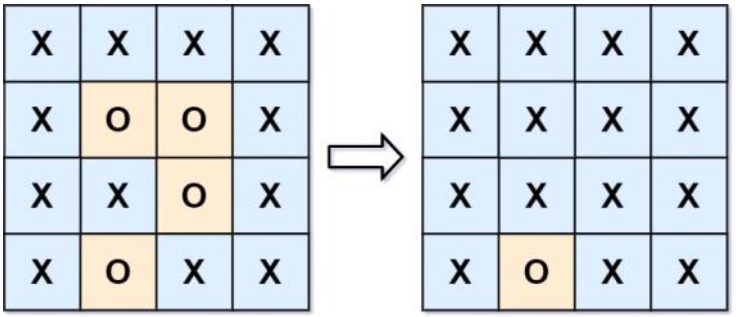
```
    }
}
```

Surrounded Regions (dfs)

10 October 2024 20:46

#NORMAL APPROCH CALL DFS IF YOU FOUND O IN i==0||i==R-1||j==0||j==C-1

```
class Solution {

    public void dfs(int x,int y,int r,int c,char[][]board)
    {
        board[x][y]='Y';
        int[]dx={-1,1,0,0};
        int[]dy={0,0,-1,1};
        for(int i=0;i<4;i++)
        {
            if(x+dx[i]<0||x+dx[i]>=r||y+dy[i]<0||y+dy[i]>=c||board[x+dx[i]]
[y+dy[i]]!='O')
            {
                continue;
            }
            dfs(x+dx[i],y+dy[i],r,c,board);
        }
    }
    public void solve(char[][] board)
    {
        int r=board.length;
        int c=board[0].length;
        for(int i=0;i<r;i++)
        {
            for(int j=0;j<c;j++)
            {
                if(i==0||j==0||i==r-1||j==c-1)
                {
                    if(board[i][j]=='O')
                    {
                        dfs(i,j,r,c,board);

                    }
                }
            }
        }
        for(int i=0;i<r;i++)
        {
            for(int j=0;j<c;j++)
            {
                if(board[i][j]=='O')
                {
                    board[i][j]='X';
                }
                else if(board[i][j]=='Y')
                {
                    board[i][j]='O';
                }
            }
        }
        return;
    }
}
```


X X X X
X O O X
X X O X
X O X X
X X X X
X X X X
X X X X
X O X X

Number of Enclaves [flood fill implementation - multisource]

10 October 2024 20:46

PLEASE READ THE QUESTION PROPERLY

```
class Solution {
    public void dfs(int x,int y,int r,int c,int[][]board)
    {
        board[x][y]=0;
        int[]dx={-1,1,0,0};
        int[]dy={0,0,-1,1};
        for(int i=0;i<4;i++)
        {
            if(x+dx[i]<0||x+dx[i]>=r||y+dy[i]<0||y+dy[i]>=c||board[x+dx[i]]
[y+dy[i]]!=1)
            {
                continue;
            }
            dfs(x+dx[i],y+dy[i],r,c,board);
        }
    }
    public int numEnclaves(int[][] board)
    {
        int r=board.length;
        int c=board[0].length;
        for(int i=0;i<r;i++)
        {
            for(int j=0;j<c;j++)
            {
                if(i==0||j==0||i==r-1||j==c-1)
                {
                    if(board[i][j]==1)
                    {
                        dfs(i,j,r,c,board);

                    }
                }
            }
        }
        int cnt=0;
        for(int i=0;i<r;i++)
        {
            for(int j=0;j<c;j++)
            {
                if(board[i][j]==1)
                {
                    cnt++;
                }
            }
        }
        return cnt;


    }
}
```

Word ladder - 1****

10 October 2024 20:46

Word ladder - 2****

10 October 2024 20:46

Number of Distinct Islands [dfs multisource]

10 October 2024 20:46

No of island

```
void mark_current_island(vector<vector<char>> &matrix,int x,int y,int r,int c)
  {
    if(x<0 || x>=r || y<0 || y>=c || matrix[x][y]!='1')  //Boundary case for matrix
       return;
    if(matrix[x][y]=='2')
    {
       return ;
    }
    //Mark current cell as visited
    matrix[x][y] = '2';

    //Make recursive call in all 4 adjacent directions
    mark_current_island(matrix,x+1,y,r,c);  //DOWN
    mark_current_island(matrix,x,y+1,r,c);  //RIGHT
    mark_current_island(matrix,x-1,y,r,c);  //TOP
    mark_current_island(matrix,x,y-1,r,c);  //LEFT
    mark_current_island(matrix,x+1,y+1,r,c); //SE
    mark_current_island(matrix,x+1,y-1,r,c); //SW
    mark_current_island(matrix,x-1,y-1,r,c); //NE
    mark_current_island(matrix,x-1,y+1,r,c); //NW
  }




  int numIslands(vector<vector<char>>& grid)
  {

    int rows = grid.size();
    if(rows==0)    //Empty grid boundary case
       return 0;
    int cols = grid[0].size();

    //Iterate for all cells of the array
    int no_of_islands = 0;
    for(int i=0;i<rows;++i)
    {
       for(int j=0;j<cols;++j)
       {
          if(grid[i][j]=='1')
          {
             mark_current_island(grid,i,j,rows,cols);
             no_of_islands += 1;
          }
       }
    }
    return no_of_islands;
  }
```

#MY CODE

```
public void dfs(int x,int y,int r,int c,int[][]board,StringBuilder curr)
  {
     if(x<0||x>=r||y<0||y>=c||board[x][y]!=1)
     {
        return;
     }
     board[x][y]=0;
     dfs(x+1,y,r,c,board,curr.append('D'));
     dfs(x-1,y,r,c,board,curr.append('U'));
     dfs(x,y-1,r,c,board,curr.append('L'));
     dfs(x,y+1,r,c,board,curr.append('R'));
     curr.append('Z');

  }


  int countDistinctIslands(int[][] board)
  {

     int r=board.length;
     int c=board[0].length;
     Set<String>s=new HashSet<>();
     for(int i=0;i<r;i++)
     {
        for(int j=0;j<c;j++)
        {

              if(board[i][j]==1)
              {
                 //String curr="";
                 //ArrayList<String> curr = new ArrayList<>();

                 StringBuilder curr= new StringBuilder();
                 dfs(i,j,r,c,board,curr);
                 //System.out.println(curr);
                 s.add(curr.toString());
              }

        }
     }


     return s.size();

  }
```

Bipartite Graph (DFS)

10 October 2024 20:47

BFS:

```
class Solution {
    private boolean check(int[][]graph,int start,int[]vis)
    {
        Queue<Integer>q=new LinkedList<Integer>();
        q.add(start);
        vis[start]=0;
        while(q.size()>0)
        {
            int scr=q.peek();
            q.remove();
            for(int it:graph[scr]) //matrix se element kaise pick kar rha hai
            {
                if(vis[it]==-1)
                {
                    vis[it]=1-vis[scr];
                    q.add(it);
                }
                else if(vis[it]==vis[scr])
                {
                    return false;
                }
            }
        }
        return true;
    }
    public boolean isBipartite(int[][] graph)
    {
        int V=graph.length;
        int[]vis=new int[V];
        for(int i=0;i<V;i++)
        {
            vis[i]=-1;
        }
        for(int i=0;i<V;i++)
        {
            if(vis[i]==-1)
            {
                if(check(graph,i,vis)==false)
                {
                    return false;
                }
            }
        }
        return true;
    }
}
```

DFS:

```
import java.util.*;

class Solution {

   private boolean dfs(int[][] graph, int node, int[] vis, int color) {
```

```
        // Color the current node
        vis[node] = color;

        // Traverse all neighbors of the current node
        for (int neighbor : graph[node]) {
            // If the neighbor has not been colored, color it with the opposite color and continue DFS
            if (vis[neighbor] == -1) {
                if (!dfs(graph, neighbor, vis, 1 - color)) {
                    return false;
                }
            }
            // If the neighbor has the same color as the current node, return false (not bipartite)
            else if (vis[neighbor] == color) {
                return false;
            }
        }
        return true;
    }

    public boolean isBipartite(int[][] graph) {
        int V = graph.length; // Number of vertices
        int[] vis = new int[V];

        // Initialize all vertices as unvisited (-1)
        Arrays.fill(vis, -1);

        // Check each component of the graph
        for (int i = 0; i < V; i++) {
            if (vis[i] == -1) {  // Not visited yet
                if (!dfs(graph, i, vis, 0)) {  // Start DFS with color 0
                    return false; // Graph is not bipartite
                }
            }
        }
        return true; // Graph is bipartite
    }
}
```

Cycle Detection in Directed Graph (DFS)

10 October 2024 20:47

```
import java.util.ArrayList;

public class Solution {

 // DFS function to detect a cycle in a directed graph
 private static boolean dfs(int start, ArrayList<ArrayList<Integer>> edges, int[] vis, int[] par) {
  // Mark the current node as visited and part of the current recursion stack
  vis[start] = 1;  // Visited state
  par[start] = 1;  // Recursion stack state

  // Explore all neighbors of the current node
  for (int it : edges.get(start)) {
   // If the neighbor is not visited, perform DFS on it
   if (vis[it] == 0) {
    if (dfs(it, edges, vis, par)) {
     return true; // Cycle detected
    }
   }
   // If the neighbor is currently in the recursion stack (par == 1), it indicates a cycle
   else if (par[it] == 1) {
    return true; // Cycle detected
   }
  }

  // Mark the current node as fully processed (finished processing)
  par[start] = 0;  // No longer in recursion stack
  return false;    // No cycle detected
 }

 // Function to detect cycle in a directed graph
 public static boolean detectCycleInDirectedGraph(int n, ArrayList<ArrayList<Integer>> edges) {
  // Initialize visited and recursion stack arrays
  int[] vis = new int[n];
  int[] par = new int[n];

  // Check for cycles in all nodes
  for (int i = 0; i < n; i++) {
   // If the node is unvisited, perform DFS from it
   if (vis[i] == 0) {
    if (dfs(i, edges, vis, par)) {
     return true; // Cycle detected
    }
   }
  }
  return false; // No cycle detected in the entire graph
 }

 public static void main(String[] args) {
  // Example test case
  int n = 6;  // Number of nodes
  ArrayList<ArrayList<Integer>> edges = new ArrayList<>();
```

```
    // Initialize the adjacency list for the graph (size n = 6)
    for (int i = 0; i < n; i++) {
      edges.add(new ArrayList<>());
    }

    // Add the directed edges
    edges.get(0).add(1); // 1 -> 2
    edges.get(1).add(2); // 2 -> 4
    edges.get(4).add(1); // 4 -> 1
    edges.get(2).add(4); // 3 -> 4
    edges.get(3).add(4); // 4 -> 2
    edges.get(5).add(2); // 5 -> 3
    edges.get(0).add(3); // 1 -> 3

    // Detect cycle
    boolean hasCycle = detectCycleInDirectedGraph(n, edges);
    System.out.println("Cycle detected: " + hasCycle);  // Expected output: false
  }
}
```

Topo Sort

18 October 2024 02:38

```
public static void dfs(int start,ArrayList<ArrayList<Integer>> adj,boolean[] vis,Stack<Integer>s)
  {

     vis[start]=true;

     for(int it:adj.get(start))
     {
        if(vis[it]==false)
        {
           dfs(it,adj,vis,s);
        }
     }
     s.add(start);
  }

  static ArrayList<Integer> topologicalSort(ArrayList<ArrayList<Integer>> adj)
  {

     ArrayList<Integer>ans=new ArrayList<>();
     Stack<Integer>s=new Stack<>();
     int V=adj.size();
     boolean[]vis=new boolean[V];
     for(int i=0;i<V;i++)
     {
        if(vis[i]==false)
        {
           dfs(i,adj,vis,s);
        }
     }

     while(s.size()>0)
     {
        int a=s.peek();
        ans.add(a);
        s.pop();
     }

     return ans;

  }
```

Kahn's Algorithm

18 October 2024 02:39

#KAHN ALGORITHM IS NOTHING BUT TOPOLOGY SORT

```
public static void dfs(int start,ArrayList<ArrayList<Integer>> adj,boolean[] vis,Stack<Integer>s)
  {

     vis[start]=true;

     for(int it:adj.get(start))
     {
        if(vis[it]==false)
        {
           dfs(it,adj,vis,s);
        }
     }
     s.add(start);
  }

  static ArrayList<Integer> topologicalSort(ArrayList<ArrayList<Integer>> adj)
  {

     ArrayList<Integer>ans=new ArrayList<>();
     Stack<Integer>s=new Stack<>();
     int V=adj.size();
     boolean[]vis=new boolean[V];
     for(int i=0;i<V;i++)
     {
        if(vis[i]==false)
        {
           dfs(i,adj,vis,s);
        }
     }

     while(s.size()>0)
     {
        int a=s.peek();
        ans.add(a);
        s.pop();
     }

     return ans;

  }
```

Cycle Detection in Directed Graph (BFS)

18 October 2024 02:39

```
boolean dfs(int scr,ArrayList<ArrayList<Integer>> adj,boolean[]vis,boolean[] par)
  {
     vis[scr]=true;
     par[scr]=true;

     for(int it:adj.get(scr))
     {
        if(vis[it]==false)
        {
           if(dfs(it,adj,vis,par))
           {
              return true;
           }
        }
        else if(it==scr||par[it]==true)
        {
           return true;
        }
     }
     par[scr]=false;
     return false;
  }


  public boolean isCyclic(int V, ArrayList<ArrayList<Integer>> adj)
  {
     boolean[]vis=new boolean[V];
     boolean[]par=new boolean[V];


     for(int i=0;i<V;i++)
     {
        if(vis[i]==false)
        {
           boolean flag=dfs(i,adj,vis,par);
           if(flag==true)
           {
              return true;
           }
        }
     }
     return false;
  }
```

Course Schedule - I

18 October 2024 02:39

#WRONG ANSWER ?

```
class Solution {
    public boolean dfs(int scr,List<List<Integer>>adj,boolean[]vis,boolean[] par)
    {
        vis[scr]=true;
        par[scr]=true;
        for(int it:adj.get(scr))
        {
            if(vis[it]==false)
            {
                if(dfs(it,adj,vis,par))
                {
                    return true;
                }
                else if(it==scr||par[it]==true)  //Ye kya bawasir hai ye if case ke bahar rahega
                {
                    return true;
                }
            }
        }
        par[scr]=false;
        return false;
    }
    public boolean canFinish(int k, int[][] arr)
    {
        List<List<Integer>>adj=new ArrayList<List<Integer>>();

        for(int i=0;i<k;i++)
        {
            adj.add(new ArrayList<>());
        }
        for(int i=0;i<arr.length;i++)
        {
            adj.get(arr[i][0]).add(arr[i][1]);
        }

        boolean[]vis=new boolean[k];
        boolean[]par=new boolean[k];
        for(int i=0;i<k;i++)
        {
            if(vis[i]==false)
            {
                if(dfs(i,adj,vis,par))
                {
                    return false;
                }
            }
        }
        return true;
    }
}
```

Correct ans

```java
public boolean dfs(int scr,List<List<Integer>>adj,boolean[]vis,boolean[] par)
    {
        vis[scr]=true;
        par[scr]=true;
        for(int it:adj.get(scr))
        {
            if(vis[it]==false)
            {
                if(dfs(it,adj,vis,par))
                {
                    return true;
                }
            }
            else if(it==scr||par[it]==true)
            {
                return true;
            }
        }
        par[scr]=false;
        return false;
    }
    public boolean canFinish(int k, int[][] arr)
    {
        List<List<Integer>>adj=new ArrayList<List<Integer>>();

        for(int i=0;i<k;i++)
        {
            adj.add(new ArrayList<>());
        }
        for(int i=0;i<arr.length;i++)
        {
            adj.get(arr[i][0]).add(arr[i][1]);
        }

        boolean[]vis=new boolean[k];
        boolean[]par=new boolean[k];
        for(int i=0;i<k;i++)
        {
            if(vis[i]==false)
            {
                if(dfs(i,adj,vis,par))
                {
                    return false;
                }
            }
        }
        return true;
    }
```

Course Schedule - II

18 October 2024 02:39

#WRONG ANSWER SAME AS COURSE SCHEDULE 2
Input
numCourses = 2
prerequisites = [[0,1],[1,0]]
Use Testcase
Output
[1,0]
Expected
[]

```
class Solution {
    public boolean dfs(int scr,List<List<Integer>>adj,boolean[]vis,boolean[]
par,Stack<Integer>s)
    {
        vis[scr]=true;
        par[scr]=true;
        for(int it:adj.get(scr))
        {
            if(vis[it]==false)
            {
                if(dfs(it,adj,vis,par,s))
                {
                    return true;
                }
                else if(it==scr||par[it]==true)
                {
                    return true;
                }
            }
        }
        s.add(scr);
        par[scr]=false;
        return false;
    }
    public int[] findOrder(int k, int[][] arr)
    {

        List<List<Integer>>adj=new ArrayList<List<Integer>>();
        Stack<Integer>s=new Stack<>();
        ArrayList<Integer>ans=new ArrayList<>();

        for(int i=0;i<k;i++)
        {
            adj.add(new ArrayList<>());
        }
        for(int i=0;i<arr.length;i++)
        {
            adj.get(arr[i][0]).add(arr[i][1]);
        }

        boolean[]vis=new boolean[k];
        boolean[]par=new boolean[k];
        for(int i=0;i<k;i++)
        {
            if(vis[i]==false)
            {
                if(dfs(i,adj,vis,par,s))
                {
                    int[]res=new int[1];
                    res[0]=0;
```

```
                    return res;
                }
            }
        }
        int[]res=new int[s.size()];
        int i=s.size()-1;
        while(s.size()>0)
        {
            int a=s.peek();
            res[i]=a;
            i--;
            s.pop();
        }

        return res;
    }
}
```

Correct Ans

```
class Solution {
   public boolean dfs(int scr,List<List<Integer>>adj,boolean[]vis,boolean[]
par,Stack<Integer>s)
    {
        vis[scr]=true;
        par[scr]=true;
        for(int it:adj.get(scr))
        {
            if(vis[it]==false)
            {
                if(dfs(it,adj,vis,par,s))
                {
                    return true;
                }
            }
            else if(it==scr||par[it]==true)
            {
                return true;
            }
        }
        s.add(scr);
        par[scr]=false;
        return false;
    }


    public int[] findOrder(int k, int[][] arr)
    {
        List<List<Integer>>adj=new ArrayList<List<Integer>>();
        Stack<Integer>s=new Stack<>();
        for(int i=0;i<k;i++)
        {
            adj.add(new ArrayList<>());
        }
        for(int i=0;i<arr.length;i++)
        {
            adj.get(arr[i][0]).add(arr[i][1]);
        }

        boolean[]vis=new boolean[k];
        boolean[]par=new boolean[k];
        for(int i=0;i<k;i++)
        {
```

```
            if(vis[i]==false)
            {
                if(dfs(i,adj,vis,par,s))
                {
                    int[]res=new int[0];
                    //res[0]=0;
                    return res;
                }
            }
        }
        int[]res=new int[s.size()];

        int i=s.size()-1;
        while(s.size()>0)
        {
            int a=s.peek();
            res[i]=a;
            i--;
            s.pop();
        }

        return res;

    }
}
```

Find eventual safe states****

18 October 2024 02:39

Alien dictionary*****

18 October 2024 02:39

Shortest Path in UG with unit weights

18 October 2024 14:06

```
class Solution {
   // Function to find the shortest path from a source node to all other nodes

   class Pair{
      int distance;
      int node;
      Pair(int distance,int node)
      {
         this.distance=distance;
         this.node=node;
      }
   }
   public int[] shortestPath(ArrayList<ArrayList<Integer>> adj, int src)
   {
      int n=adj.size();
      int[]dis=new int[n];
      for(int i=0;i<n;i++)
      {
         dis[i]=(int)(1e9); //distance store kare9ga
      }

      PriorityQueue<Pair>pq=new PriorityQueue<Pair>((x,y)->x.distance-y.distance);
      dis[src]=0;
      pq.add(new Pair(0,src));

      while(pq.size()>0)
      {
         int dist=pq.peek().distance;
         int node=pq.peek().node;
         pq.remove();

          // Traverse all adjacent nodes (neighbors)
         // Traverse all adjacent nodes (neighbors)
        for (int neighbor : adj.get(node)) {
            if (dis[node] + 1 < dis[neighbor]) {
               dis[neighbor] = dis[node] + 1;
               pq.add(new Pair(dis[neighbor],neighbor));
            }
         }
      }

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

```
class Solution {

    public int[] shortestPath(int[][] edges,int n,int m ,int scr)
    {
      //Distance array
      int[]dist=new int[n];
      Arrays.fill(dist,Integer.MAX_VALUE);
      dist[scr]=0;

      //Queue for BFS
      Queue<Integer>q=new LinkedList<>();
      q.add(scr);

      //Visited zaroori hai for dijkstra
      boolean[]vis=new boolean[n];

     //Creating graph from 2D array
      ArrayList<ArrayList<Integer>>adj=new ArrayList<ArrayList<Integer>>();
      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]);
      }

      while(q.size()>0)
      {
         int node=q.poll();
         vis[node]=true;
         for(int it:adj.get(node))
         {

            if(vis[it]==false&&dist[it]>dist[node]+1)
            {
               dist[it]=dist[node]+1;
               q.add(it);
            }
         }
      }

      for(int i=0;i<n;i++)
      {
         if(dist[i]==Integer.MAX_VALUE)
```

```
        {
            dist[i]=-1;
        }
      }
      return dist;
    }
}
```

Shortest Path in DAG

18 October 2024 14:06

```
class Solution {

    class Pair{
        int x;
        int y;
        Pair(int x,int y)
        {
            this.x=x;
            this.y=y;
        }
    }

    public void topo( ArrayList<ArrayList<Pair>>adj,int[]vis,Stack<Integer>s,int start)
    {
        vis[start]=1;

        for(int i=0;i<adj.get(start).size();i++)
        {
            int v=adj.get(start).get(i).x;
            if(vis[v]==0)
            {
                topo(adj,vis,s,v);
            }
        }
        s.add(start);
    }
    public int[] shortestPath(int V, int E, int[][] edges)
    {
        //Create graph
        ArrayList<ArrayList<Pair>>adj=new ArrayList<>();

        for(int i=0;i<V;i++)
        {
            //adj=new ArrayList<>()dj.add(new ArrayList<>());
            ArrayList<Pair>temp=new ArrayList<Pair>();
            adj.add(temp);
        }

        for(int i=0;i<E;i++)
        {
            int u=edges[i][0];
            int v=edges[i][1];
            int wt=edges[i][2];
            adj.get(u).add(new Pair(v,wt)); //???????????
        }
        //Graph created

        int[]vis=new int[V];
        Stack<Integer>s=new Stack<Integer>();

        //Topological sort
```

```
        for(int i=0;i<V;i++)
        {
            if(vis[i]==0)
            {
                topo(adj,vis,s,i);
            }
        }

        //Stack me saare element aa gae hai ab banao dijkstra ka algo
        int[]dist=new int[V];

        for(int i=0;i<V;i++)
        {
            dist[i]=(int)(1e9);
        }
        dist[0]=0;
        while(s.size()>0)
        {
            int node=s.peek();
            s.pop();

            for(int i=0;i<adj.get(node).size();i++)
            {
                int next=adj.get(node).get(i).x;
                int d=adj.get(node).get(i).y;

                if(dist[node]+d<dist[next])
                {
                    dist[next]=dist[node]+d;
                }
            }
        }

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

Djisktra's Algorithm

18 October 2024 14:06

```
class Solution {
    // Function to find the shortest distance of all the vertices
    // from the source vertex src.
    class Pair{
        int a;
        int b;
        Pair(int a,int b)
        {
            this.a=a;
            this.b=b;
        }
    }

    ArrayList<Integer> dijkstra(ArrayList<ArrayList<iPair>> adj, int src)
    {
        int n=adj.size();
        PriorityQueue<Pair>pq=new PriorityQueue<Pair>((x,y)->x.a-y.a);
        int[] dis=new int[n];
        for(int i=0;i<n;i++)
        {
            dis[i]=(int)(1e9);
        }
        dis[src]=0;

        pq.add(new Pair(0,src));

        while(pq.size()>0)
        {
            int d=pq.peek().a;
            int node=pq.peek().b;
            pq.remove();

           for(iPair neighbor:adj.get(node))
           {
               int dest=neighbor.first; //dest
               int dist=neighbor.second; //weight

               if(dis[node]+dist<dis[dest])
               {
                   dis[dest]=dis[node]+dist;
                   pq.add(new Pair(dis[dest],dest));
               }
           }
        }
        // Convert dis array into ArrayList for return
        ArrayList<Integer> result = new ArrayList<>();
        for (int i = 0; i < n; i++) {
            result.add(dis[i]);
        }

        return result;
    }
}
```

Why priority Queue is used in Djisktra's Algorithm

18 October 2024 14:06

Dijkstra's algorithm uses a priority queue (often implemented as a min-heap) to efficiently find the shortest path from a source node to all other nodes in a graph, b
3y always selecting the node with the smallest known distance for exploration.

Shortest path in a binary maze

18 October 2024 14:06

```
class Solution {
    //Direction array
    int[][] dir=new int[][]{{0,1},{0,-1},{1,0},{-1,0},{1,-1},{-1,1},{-1,-1},{1,1}};
    public int shortestPathBinaryMatrix(int[][] grid) +
    {
        int n=grid.length;
        int m=grid[0].length;
        //Base Condition
        if(grid[0][0]==1||grid[n-1][m-1]==1)
        {
            return -1;
        }
        //Vis array
        boolean[][]vis=new boolean[n][m];
        //Queue for traversal
        //we use queue not priority queue as the weight is 1 everyside
        Queue<int[]>q=new LinkedList<>();
        q.add(new int[]{0,0});
        int ans=0;
        while(q.size()>0)
        {
            int k=q.size();
            for(int i1=0;i1<k;i1++)
            {
                int[]a=q.peek();
                q.remove();
                int x=a[0];
                int y=a[1];
                if(x==n-1&&y==m-1)
                {
                    return ans+1;
                }
                for(int i=0;i<8;i++)
                {
                    int nextx=x+dir[i][0];
                    int nexty=y+dir[i][1];
                    if(nextx<0||nextx>=n||nexty<0||nexty>=m||vis[nextx][nexty]==true||grid[nextx][nexty]!=0)
                    {
                        continue;
                    }
                    else
                    {
                        q.add(new int[]{nextx,nexty});
                        vis[nextx][nexty]=true;
                    }
                }
            }
            ans++;
        }
        return -1;
    }

}
```

Path with minimum effort

18 October 2024 14:06

```
class Solution {
    //Direction array
    int[][] dir=new int[][]{{0,1},{0,-1},{1,0},{-1,0}};
    class Pair{
        int wt;
        int x;
        int y;
        Pair(int wt,int x,int y)
        {
            this.wt=wt;
            this.x=x;
            this.y=y;
        }
    }
    public int minimumEffortPath(int[][] grid)
    {
        int n=grid.length;
        int m=grid[0].length;

        //Vis array
        int[][]vis=new int[n][m];
        for(int i=0;i<n;i++)
        {
            for(int j=0;j<m;j++)
            {
                vis[i][j]=(int)(1e9);
            }
        }
        vis[0][0]=0;
        //Queue for traversal
        //we use queue not priority queue as the weight is 1 everyside
        PriorityQueue<Pair>pq=new PriorityQueue<Pair>((x,y)->x.wt-y.wt);
        pq.add(new Pair(0,0,0));

        while(pq.size()>0)
        {
                Pair a=pq.peek();
                pq.remove();
                int wt1=a.wt;
                int x1=a.x;
                int y1=a.y;
                if(x1==n-1&&y1==m-1)
                {
                    return wt1;
                }
                for(int i=0;i<4;i++)
                {
                    int nextx=x1+dir[i][0];
                    int nexty=y1+dir[i][1];
                    if(nextx<0||nextx>=n||nexty<0||nexty>=m)
                    {
                        continue;
                    }
                    else
                    {
                        int newwt=Math.max(Math.abs(grid[nextx][nexty]-grid[x1]
[y1]),wt1);
                        if(vis[nextx][nexty]>newwt)
                        {
```

```
                        vis[nextx][nexty]=newwt;
                        pq.add(new Pair(newwt,nextx,nexty));
                    }


                }
            }
        }
        return 0;

    }
}
```

Cheapest flights within k stops

18 October 2024 14:06

```
class Solution {
    class Pair{
        int wt;
        int x;
        Pair(int wt,int x)
        {
            this.wt=wt;
            this.x=x;
        }
    }
    class Pair1{
        int wt;
        int x;
        int stops;
        Pair1(int wt,int x,int stops)
        {
            this.wt=wt;
            this.x=x;
            this.stops=stops;
        }
    }
    public int findCheapestPrice(int n, int[][] flights, int src, int dst, int k)
    {
        //create a graph
        ArrayList<ArrayList<Pair>>adj=new ArrayList<>();
        for(int i=0;i<n;i++)
        {
            ArrayList<Pair>temp=new ArrayList<>();
            adj.add(temp);
        }
        int m=flights.length;
        for(int i=0;i<m;i++)
        {
            int w=flights[i][2];
            int scr=flights[i][0];
            int dest=flights[i][1];
            adj.get(scr).add(new Pair(w,dest));    // Source->wieght->destination
        }
        //Create Priority Queue
        //PriorityQueue<Pair1>pq=new PriorityQueue<Pair1>((x,y)->x.wt-y.wt);
        Queue<Pair1>pq=new LinkedList<>();
        pq.add(new Pair1(0,src,0));  //weight+ source+ ab tak kitne stops
        int[]vis=new int[n];
        for(int i=0;i<n;i++)
        {
            vis[i]=(int)(1e9);
        }
        vis[src]=0;
        while(pq.size()>0)
        {
            Pair1 temp=pq.peek();
            pq.remove();
            int weight=temp.wt;
            int x=temp.x;
            int stops=temp.stops;
            // if(x==dst)
            // {
            //     return weight;
            // }
```

```
            //Kyuki destination pahuchne ke baad bhi next min aa sakta hai
            if(stops>k)
            {
                continue;
            }
             for(Pair it:adj.get(x))
            {
                int nextw=it.wt;
                int nextd=it.x;

                if(weight+nextw<vis[nextd]&&stops<=k)
                {
                    vis[nextd]=weight+nextw;
                    pq.add(new Pair1(weight+nextw,nextd,stops+1));
                }


                // if(vis[x]+nextw<vis[nextd]&&stops<=k)
                // {
                //     vis[nextd]=vis[x]+nextw;
                //     pq.add(new Pair1(vis[x]+nextw,nextd,stops+1));
                // }
            }
        }
        if(vis[dst]==(int)(1e9))
        {
            return -1;
        }
        return vis[dst];

    }
}
```

Network Delay time

18 October 2024 14:06

Number of ways to arrive at destination

18 October 2024 14:06

Minimum steps to reach end from start by performing multiplication and mod operations with array elements

18 October 2024 14:06

Bellman Ford Algorithm

18 October 2024 14:06

Floyd Warshal Algorithm

18 October 2024 14:07

Find the city with the smallest number of neighbors in a threshold distance

18 October 2024 14:07

Minimum Spanning Tree

18 October 2024 14:07

1-> Prism Algorithm
2-> Kruskal Algorithm

C++

```
int spanningTree(int V, vector<vector<int>> adj[])
{
priority_queue <pair<int,int>,vector<pair<int,int>>,greater<pair<int,int>>> pq;
pq.push({0,0});
// vector<int> ans(V,0);
int ans=0;
vector<bool> visit(V,false);
while(!pq.empty()){
auto p = pq.top();
pq.pop();
int x = p.second;
int wt = p.first;
if(visit[x])continue;
visit[x] = true;
ans+=wt;
for(auto y:adj[x]){
if(!visit[y[0]]) pq.push({y[1],y[0]});
}
}
return ans;
}
```

JAVA:

```
import java.util.*;

class Solution {

    // Pair class to store weight and node
    class Pair {
        int wt;   // weight of the edge
        int node; // destination node
        Pair(int wt, int node) {
            this.wt = wt;
            this.node = node;
        }
    }

    static int spanningTree(int V, int E, List<List<int[]>> adj) {
        PriorityQueue<Pair> pq = new PriorityQueue<>((x, y) -> x.wt - y.wt);
        int[] vis = new int[V]; // visited array to mark visited nodes
```

```
        pq.add(new Pair(0, 0)); // Start from node 0 with weight 0

        int ans = 0;

        while (!pq.isEmpty()) {
            Pair temp = pq.poll(); // Get the edge with the minimum weight
            int wt = temp.wt;
            int scr = temp.node;

            // Skip the node if it is already visited
            if (vis[scr] == 1) {
                continue;
            }

            // Mark the node as visited
            vis[scr] = 1;
            ans += wt; // Add the weight of the current edge to the MST sum

            // Explore all neighbors of the current node
            for (int[] it : adj.get(scr)) {
                // it[0] is the neighbor, and it[1] is the weight of the edge
                if (vis[it[0]] == 0) {
                    pq.add(new Pair(it[1], it[0])); // Add neighbor to the priority queue
                }
            }
        }

        return ans; // Return the sum of the weights of the MST
    }
}
```

Prim's Algorithm

18 October 2024 14:07

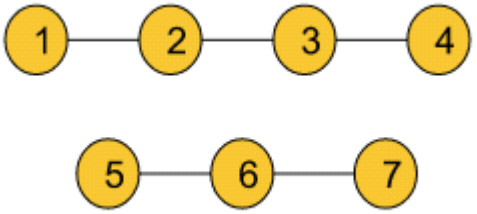
//AGAR get the MST question me dpuchha hoga to priority queue me ek or node store karenge jo parent store karega

```
import java.util.*;

class Solution {

    // Pair class to store weight and node
    class Pair {
        int wt;   // weight of the edge
        int node; // destination node
        Pair(int wt, int node) {
            this.wt = wt;
            this.node = node;
        }
    }

    static int spanningTree(int V, int E, List<List<int[]>> adj) {
        PriorityQueue<Pair> pq = new PriorityQueue<>((x, y) -> x.wt - y.wt);
        int[] vis = new int[V]; // visited array to mark visited nodes
        pq.add(new Pair(0, 0)); // Start from node 0 with weight 0

        int ans = 0;

        while (!pq.isEmpty()) {
            Pair temp = pq.poll(); // Get the edge with the minimum weight
            int wt = temp.wt;
            int scr = temp.node;

            // Skip the node if it is already visited
            if (vis[scr] == 1) {
                continue;
            }
            //ye baad me visied isiliye kar krhe haiki agar pehle kar denge to may be baad me koi node istak ja rha hoga jo kam weight ka hoga
            // Mark the node as visited
            vis[scr] = 1;
            ans += wt; // Add the weight of the current edge to the MST sum

            // Explore all neighbors of the current node
            for (int[] it : adj.get(scr)) {
                // it[0] is the neighbor, and it[1] is the weight of the edge
                if (vis[it[0]] == 0) {
                    pq.add(new Pair(it[1], it[0])); // Add neighbor to the priority queue
                }
            }
        }

        return ans; // Return the sum of the weights of the MST
    }
}
```

Disjoint Set [Union by Rank] ***

18 October 2024 14:07

Question: Given two components of an undirected graph



The question is whether node 1 and node 5 are in the same component or not.

```
import java.io.*;
import java.util.*;
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 Main {
    public static void main (String[] args) {
        DisjointSet ds = new DisjointSet(7);
        ds.unionByRank(1, 2);
        ds.unionByRank(2, 3);
        ds.unionByRank(4, 5);
        ds.unionByRank(6, 7);
        ds.unionByRank(5, 6);
// if 3 and 7 same or not
        if (ds.findUPar(3) == ds.findUPar(7)) {
            System.out.println("Same");
        } else
            System.out.println("Not Same");
ds.unionByRank(3, 7);
        if (ds.findUPar(3) == ds.findUPar(7)) {
            System.out.println("Same");
        } else
            System.out.println("Not Same");
    }
}
```

JAVA:

```
class GfG {

    int find(int par[], int x) {
        //while returnibg back with the recursion stack
        //sabke parent ke values ko ultimate parent se replace karte jao

        if(x==par[x]) return x;

        else{
            return par[x] = find(par, par[x]);
        }
    }

    void unionSet(int par[], int x, int y) {
        //RANK array
        int [] rank = new int[par.length]; //initially sabka rank 0 hoga

        //parnets of both
        int x_par = find(par, x);
        int y_par = find(par, y);

        //parents same => they are in same set
        if(x_par == y_par) return;

        //when not in same set
        //and
        //dont have same rank
        if(rank[x_par]>rank[y_par])
            par[y_par]= x_par;

        else if (rank[y_par]>rank[x_par])
            par[x_par]= y_par;

        //have same rank
```

```
        else{
            par[x_par]= y_par;
            rank[y_par]++;
        }
    }

    void unionBySize(int par[], int x, int y) {
        //RANK array
        int [] size = new int[par.length];
        for(int i=0;i<par.length;i++)
        {
            size[i]=1;
        }

        //parnets of both
        int x_par = find(par, x);
        int y_par = find(par, y);

        //parents same => they are in same set
        if(x_par == y_par) return;

        //when not in same set
        //and
        //dont have same rank
        if(size[x_par]>size[y_par])
        {
            par[y_par]= x_par;
            size[y_par]+=size[x_par];
        }
        else if (size[y_par]>size[x_par])
        {
            par[x_par]= y_par;
            size[x_par]+=size[y_par];
        }
        //have same rank
        else //we can club both elseif and else code please note
        {
            par[x_par]= y_par;
            size[x_par]+=size[y_par];
        }
    }
}
```

Disjoint Set [Union by Size]

18 October 2024 14:07

```
void unionBySize(int par[], int x, int y) {
    //RANK array
    int [] size = new int[par.length];
    for(int i=0;i<par.length;i++)
    {
       size[i]=1;
    }

    //parnets of both
    int x_par = find(par, x);
    int y_par = find(par, y);

    //parents same => they are in same set
    if(x_par == y_par) return;

    //when not in same set
    //and
    //dont have same rank
    if(size[x_par]>size[y_par])
    {
       par[y_par]= x_par;
       size[y_par]+=size[x_par];
    }
    else if (size[y_par]>size[x_par])
    {
       par[x_par]= y_par;
       size[x_par]+=size[y_par];
    }
    //have same rank
    else //we can club both elseif and else code please note
    {
       par[x_par]= y_par;
       size[x_par]+=size[y_par];
    }
  }
```

Kruskal's Algorithm***

18 October 2024 14:07

Dijkstra : from a single source, find shortest paths to all nodes.
Floyd-Warshall : shortest path from every node as a source.
Bellman-Ford : same as Dijkstra, but works for negative weights.
Topological Sort : print the nodes with no incoming edges first.
MST : connect all the nodes with minimum costs (n nodes // n-1 edges).
Prim's Algo : build the MST by starting from any node and expanding the tree one edge at a time.
Kruskal Algo : build the MST by sorting all edges and adding them one by one, ensuring no cycles are formed.

```
import java.io.*;
import java.util.*;
// User function Template for Java
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 _wt) {
        this.src = _src; this.dest = _dest; this.weight = _wt;
    }
    // Comparator function used for
    // sorting edgesbased on their weight
    public int compareTo(Edge compareEdge) {
        return this.weight - compareEdge.weight;
    }
};
class Solution {
    //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>();
        // O(N + E)
        for (int i = 0; i < V; i++) {
            for (int j = 0; j < adj.get(i).size(); j++) {
                int adjNode = adj.get(i).get(j).get(0);
                int wt = adj.get(i).get(j).get(1);
                int node = i;
                Edge temp = new Edge(i, adjNode, wt);
                edges.add(temp);
            }
        }
        DisjointSet ds = new DisjointSet(V);
        // M log M
        Collections.sort(edges);
        int mstWt = 0;
        // M x 4 x alpha x 2
        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;
    }
}
class Main {
    public static void main (String[] args) {
        int V = 5;
        ArrayList<ArrayList<ArrayList<Integer>>> adj = new ArrayList<ArrayList<ArrayList<Integer>>>
```

```
();
        int[][] edges =  {{0, 1, 2}, {0, 2, 1}, {1, 2, 1}, {2, 3, 2}, {3, 4, 1}, {4, 2, 2}};
for (int i = 0; i < V; i++) {
            adj.add(new ArrayList<ArrayList<Integer>>());
        }
for (int i = 0; i < 6; i++) {
            int u = edges[i][0];
            int v = edges[i][1];
            int w = edges[i][2];
ArrayList<Integer> tmp1 = new ArrayList<Integer>();
            ArrayList<Integer> tmp2 = new ArrayList<Integer>();
            tmp1.add(v);
            tmp1.add(w);
tmp2.add(u);
            tmp2.add(w);
adj.get(u).add(tmp1);
            adj.get(v).add(tmp2);
        }
Solution obj = new Solution();
        int mstWt = obj.spanningTree(V, adj);
        System.out.println("The sum of all the edge weights: " + mstWt);
}
}
```

From <https://takeuforward.org/data-structure/kruskals-algorithm-minimum-spanning-tree-g-47/>

Number of operations to make network connected

18 October 2024 14:07

```
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 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 makeConnected(int n, int[][] connections)
    {
        if (connections.length < n - 1) {
            return -1;  // Not enough connections to make all nodes connected
        }
        DisjointSet ds=new DisjointSet(n);
        // for(int i=0;i<connections.length;i++)
        // {
        //     for(int j=0;j<connections[0].length;j++)
        //     {
        //         if(connections[i][j]==1)
        //         {
        //             ds.unionBySize(i,j);
        //         }
        //     }
        // }
         for (int[] connection : connections) {
            int u = connection[0];
            int v = connection[1];
            ds.unionBySize(u, v);
        }
        // Count the number of connected components (i.e., number of distinct sets)
        int cnt = 0;
        for (int i = 0; i < n; i++) {
```

```
            if (ds.parent.get(i) == i) {
                cnt++;  // Each root represents a distinct connected component
            }
        }
        return cnt - 1;  // Return the number of operations to connect the
components (we need cnt-1 edges)
    }

}
```

Most stones removed with same rows or columns

18 October 2024 14:07

Accounts merge

18 October 2024 14:07

Number of island II

18 October 2024 14:07

Making a Large Island

18 October 2024 14:07

Swim in rising water

18 October 2024 14:08

Bridges in Graph

18 October 2024 14:08

Articulation Point

18 October 2024 14:08

Kosaraju's Algorithm

18 October 2024 14:08