Problem Set

sharif

May 28, 2023

1 Wormholes

Link

This problem is based on finding a **negative cycle** in a graph.
ALgorithm used: **Bellman ford**

```cpp

#include<bits/stdc++.h>
using namespace std ;


 bool find_negative_cycle(vector<pair<int,int>>adj[],int noVertices ,int source)
{
    vector<int>distance(noVertices,numeric_limits<int>::max());
    distance[source]=0 ;

    //Relaxation
    for(int i=0 ;i<noVertices-1;i++)
    {
        for(int u=0;u<noVertices;u++)
        {
            for(auto edges: adj[u])
            {
                int v=edges.first ;
                int weight=edges.second ;
                if(distance[u]!=numeric_limits<int>::max() && distance[u]+weight<distance[v])
                {
                    distance[v]=distance[u]+weight ;
                }
            }
        }
    }

    //additional iteration to check negative cycle
    for(int i=0 ;i<noVertices ;i++)
    {
        for(auto edges: adj[i])
            {
                int v=edges.first ;
                int weight=edges.second ;
                if(distance[i]!=numeric_limits<int>::max() && distance[i]+weight<distance[v])
                {
                    return true ;
                }
            }
    }
    return false ;

}

```

```

int main()
{

int t ,n,m;//star system , wormholes
scanf("%d",&t);
while(t--)
{
 scanf("%d%d",&n,&m);
 vector<pair<int,int>>adj[n+10] ;
 for(int i=0 ;i<m ;i++)
 {
     int u,v,w ;
     scanf("%d%d%d",&u,&v,&w);
     adj[u].push_back({v,w});

 }

 if(find_negative_cycle(adj,n,0))
 {
     printf("possible\n");
 }
 else
 {
     printf("not possible\n");
 }

}

return 0 ;
}
```

2 King Escape

Link

ALgorithm used: **dfs**

```cpp

#include<bits/stdc++.h>
using namespace std;
const int N=1e3+10 ;

int visited[N][N] ;
int n,ax,ay,bx,by,cx,cy ;

bool checkValidMove(int nx,int ny) //new x and new y
{
    if(nx==ax || ny==ay) // check horizontal and vertical
    {
        return true ;
    }
    if(abs(nx-ax)==abs(ny-ay)) //check queen and king are same diagonal or not
    {
        return true ;
    }

    return false ;
}

void dfs(int x,int y) {

    int dx[8] = {1, 1, 1, 0, 0, -1, -1, -1};
    int dy[8] = {1, 0, -1, 1, -1, 1, 0, -1};

        if(x<1 || x>n || y<1 || y>n)
        {
            return ;
        }
        if(visited[x][y])
        {
            return  ;
        }

        if(checkValidMove(x,y))
        {
            return ;
        }
    visited[x][y]=1 ;

        for (int i = 0; i < 8; ++i) {
            int nx = x + dx[i];
            int ny = y + dy[i];

            dfs(nx,ny);//check for new cell

        }
}

int main() {
cin >> n;
   // queen location
    cin >> ax >> ay;
    // king location
    cin >> bx >> by;
    // target location
    cin >> cx >> cy;

    dfs(bx,by);
    if (visited[cx][cy])
        cout << "YES" << endl;
    else
        cout << "NO" << endl;

    return 0;
}
```

3 leetCode-Find if Path Exists in Graph

Link

This problem is based on finding a **connected component** in a graph.
ALgorithm used: **dfs**

```

class Solution {

     void dfs(int source ,vector<int>adj[], vector<bool>&visit )
    {
        visit[source]=true ;
        for(auto child: adj[source]) //traverse adjacent nodes  of source
        {
            if(!visit[child])
            {
                visit[child]=true ;
                dfs(child,adj,visit);
            }
        }
    }
public:
    bool validPath(int n, vector<vector<int>>& edges, int source, int destination) {

        int x=edges.size() ;
        vector<int>adj[n];
        vector<bool>visit(n,false) ;

        for(int i=0 ;i<x ;i++)
        {
            adj[edges[i][0]].push_back(edges[i][1]);
            adj[edges[i][1]].push_back(edges[i][0]);

        }


                dfs(source,adj,visit); // traverse from source if found destination return true

                if( (visit[source] && !visit[destination]) ||  (!visit[source] &&visit[destination]))
                {
                    return false ;
                }


         return true ;


    }
};
```

4 CodeForces-Bmail Computer Network

Link

This problem is based on finding a **Connected path from source to destination** in a graph.
Algorithm used: **BFS**

```
#include<bits/stdc++.h>
using namespace std ;
const int N=200000 ;
int visited[N];

void bfs(vector<int>graph[],int parent[],int source)
{
    queue<int>q ;
    q.push(source);
    visited[source]=1 ;
    parent[source]=-1 ;
    while(!q.empty())
    {
        int v=q.front();
        for(auto u: graph[v])
        {
            if(!visited[u])
            {
                parent[u]=v ;
                visited[u]=true ;
                q.push(u);

            }
        }
           q.pop() ;
    }

}

void path(int parent[],int src) // using recursion for finding parent ( destination to source)
{

    if(parent[src]==-1){
            printf("%d ",1);
        return ;
    }


   path(parent ,parent[src]);
   printf("%d ",src);
}

int main()
{
 int n ;
 cin>>n ;
 vector<int>graph[n+5];

 int parent[n]={} ;
 //8
 //1 1 2 2 3 2 5

 for(int i=2 ;i<=n ;i++)
 {
     int a;
     cin>>a ;
     graph[i].push_back(a);
     graph[a].push_back(i);
 }

 bfs(graph,parent,1) ;
 path(parent,n);


 //** you can use this path(parent,n) method  or this block of code to print path **

 /*
 int src=n ;
 set<int>path ;
 while(src!=-1)
 {
     path.insert(src);
     src=parent[src] ;
 }
 for(auto u: path)
    cout<<u<<" " ;*/


}
```

5 LeetCode - Convert Sorted Array to Binary Search Tree

Link

This problem is based on **convert a array to binary search tree**
Algorithm used: **Binary search Tree**

```
class Solution {
public:
    TreeNode* sortedArrayToBST(vector<int>& nums) {
        int size = nums.size();
        int half = size/2;

        if(size == 0)
            return NULL; // if no child


        TreeNode* root = new TreeNode(nums[half]);
        //create a root node
        if(size == 1)
            return root;

          //left child ( start to half of nums vector )
        vector<int>left(nums.begin(), nums.begin() + half);
        //right child( half to end  of nums vector )
        vector<int>right(nums.begin() + half + 1, nums.end());
        root->left = sortedArrayToBST(left);
        root->right = sortedArrayToBST(right);

        return root;
    }
};
```

6 Uva - Dividing coins

Link

This problem is based on **optimization technique**
Algorithm used: **0/1 knapSack**

```

#include<bits/stdc++.h>
using namespace std ;
const int N = 110;
const int MAXN = 102*500;
int weight[N];
int dp[N][MAXN] ;

int sum ;

//used 2D array

int knapSack(int m)
{
    int max_weight=sum/2 ; // here max capacity will be half of  total weight

    for(int i=0; i <=max_weight; i++)
     dp[0][i] = 0;


    for(int i=1 ;i<=m;i++)
    {
        for(int w=0 ;w<=max_weight ;w++)
        {
            if(weight[i]<=w)
            {
               dp[i][w]=max(dp[i-1][w],(dp[i-1][w-weight[i]]+weight[i]));
            }
            else
            {
                dp[i][w]=dp[i-1][w] ;
            }

        }
    }


    return dp[m][max_weight];
}

//using 1D Array (memory optimization)
//int  knapSack(int m)
//{
//    int w=sum/2 ;
//  for(int i=1 ;i<=m ;i++)
//  {
//      for(int j=w ;j>0 ;j--)
//      {
//          if(weight[i]<=j)
//          {
//              dp[j]=max(dp[j],weight[i]+dp[j-weight[i]]);
//          }
//      }
//  }
//    return dp[w];
//}
//

int main()
{

int t ;
scanf("%d",&t);
while(t--)
{
int m ;
scanf("%d",&m);

sum=0 ;
for(int i=1 ;i<=m ;i++)
{
    scanf("%d",&weight[i]) ;
    sum+=weight[i] ;
}

  printf("%d\n", sum - 2*knapSack(m) );
}
return 0 ;
}
```

7 Codeforces-KnapSack

Link

This problem is based on **greedy technique**
Algorithm used:

```
#include<bits/stdc++.h>
using namespace std ;


long long  total_weight ;
long long  oneItem ;
long long  weight ;
vector<int>items ;

int main()
{

    int t ;
    cin>>t ;
    while(t--)
    {
        long long n,w ;
        cin>>n>>w ;


        total_weight=0 ;
        oneItem=0 ;

        for(int i=1 ; i<=n ; i++)
        {
            cin>>weight ;
            if(weight <= w )
            {
                if(weight>=(w+1)/2)
                {
                    oneItem=i ;
                }
                else if ( total_weight < (w+1)/2)
                {
                    items.push_back(i);
                    total_weight+=weight ;
                }
            }

        }

        if(oneItem>0)
        {
            cout<<1<<'\n'<<oneItem ;

        }
        else
        {
            if(total_weight>=(w+1)/2)
            {
                cout<<items.size()<<endl ;
                for(auto u: items)
                    cout<<u<<" " ;

            }
            else
            {
                cout<<-1 ;
            }
        }

        cout<<endl;
        items.clear();

    }

}

\begin{figure}[H]
\end{figure}
```

8 LeetCode-Is Subsequence

Link

This problem is based on **Longest common subsequence**
Algorithm used: **LCS**

```
class Solution {
public:
    bool isSubsequence(string s, string t) {

        int sub=s.size();
        int  original=t.size();

        int dp[sub+1][original+1];

        for(int i=0 ;i<=sub ;i++)
        dp[i][0]=0 ; //fill first row and first column with zero
        for(int i=0 ; i<=original;i++)
        dp[0][i]=0 ;

        for(int i=1 ;i<=sub;i++)
        {
            for(int j=1 ;j<=original ;j++)
            {
                if(s[i-1]==t[j-1])
                {
                    dp[i][j]=1+dp[i-1][j-1];//if match subsequence with original string
                }
                else
                {
                    dp[i][j]=max(dp[i][j-1],dp[i-1][j]);// if not match
                }
            }
        }

        if(dp[sub][original]==sub)
        return true ;
        else
        return false ;

    }
};
```

9 Uva-dominos2

Link

This problem is based on **visited node number from given node**
Algorithm used: **dfs**

```

#include<bits/stdc++.h>
using namespace std ;
const int N=10005 ;
bool vis[N];
int c ;
vector<int>adj[N];

void dfs(int src)
{
    c++ ;
    vis[src]=true ;
    //cout<<src<<endl ;
    for(auto u: adj[src])
    {
        if(!vis[u])
        {
            vis[u] =true ;
            dfs(u);
        }
    }

}

int main()
{


    int t ;
    scanf("%d",&t);
    while(t--)
    {
        int n,m,l ;
        scanf("%d%d%d",&n,&m,&l);
        for(int i=0; i<m ; i++)
        {
            int x,y ;
            scanf("%d%d",&x,&y);
            adj[x].push_back(y);
        }
        int fallDominos[l];
        for(int i=0 ; i<l ; i++)
        {
            int a ;
            scanf("%d",&a);
            fallDominos[i]=a ;
        }
        for(int i=0 ; i<l ; i++)
        {
            if(!vis[fallDominos[i]])
                dfs(fallDominos[i]);
        }


        printf("%d\n",c);
        c=0 ;
        fill(vis,vis+N,false);
        for(int i=0; i<n ; i++)
        {
            adj[i].clear();
        }



    }

}
```

10 codeforces-: Rumor

Link

This problem is based on **finding connected component**
Algorithm used: **dfs**

```
#include<bits/stdc++.h>
using namespace std ;
#define ll long long
const int N=1e5+10 ;
vector<ll>adj[N] ;
vector<bool>visited(N,false) ;
ll cost[N] ;
ll ans=0 ;

ll dfs( ll src)
{

    ll m=cost[src];
    visited[src]=true ;
    for(auto u:adj[src])
    {
        if(!visited[u])
        {
            m=min(m,dfs(u));
        }
    }
    return m ;
}



int main()
{
    ll n,m ;
    cin>>n>>m ;
    for(ll i=1; i<=n ; i++)
    {
        ll a ;
        cin>>a ;
        cost[i]=a ;
    }
    for(ll i=0 ; i<m ; i++)
    {
        ll x,y ;
        cin>>x>>y ;
        adj[x].push_back(y);
        adj[y].push_back(x);
    }

    for(ll i=1 ; i<=n ; i++)
    {
        if(!visited[i])
        {
            ans+=dfs(i);
        }
    }
    cout<<ans<<endl ;

}
```

11 leetCode- countCompleteComponents

Link

This problem is based on **finding count of complete component in a graph**
Algorithm used: **dfs**

```
class Solution {
    bool vis[6000];
    int c=0 ;
    int e=0,v=0 ;
    vector<int>adj[6000];
    void dfs(int src)
    {
        vis[src]=true ;
        v++ ;
        for(auto u: adj[src])
        {
            e++ ;
            if(!vis[u])
            {
                dfs(u);
            }
        }
    }
public:
    int countCompleteComponents(int n, vector<vector<int>>& edges) {
       int x=edges.size();
       for(int i=0 ;i<x ;i++)
       {
           adj[edges[i][0]].push_back(edges[i][1]);
           adj[edges[i][1]].push_back(edges[i][0]);
       }
       for(int i=0 ;i<=n-1;i++)
       {
           if(!vis[i])
           {

               dfs(i);
               //cout<<v<<" "<<e<<endl ;
               if(e/2==(v*(v-1))/2)
               {
                   c++ ;
               }
               v=0 ;
               e=0 ;

           }
       }
        return c ;

    }
};
```

12 uva-binary search tree

Link

This problem is based on **binary search tree build**
Algorithm used: **BST**

```

#include<bits/stdc++.h>
using namespace std ;

struct node
{
    int value=0 ;
    node *left=NULL  ;
    node *right=NULL ;
};

node *create(node *current,int value)
{
    node *newnode=new node();
    newnode->value=value ;
    newnode->left=NULL ;
    newnode->right=NULL ;

    if(current==NULL)
        return newnode ;

    node *temp=current;
    node *next=current ;
    while(next!=NULL)
    {
        temp=next ;
        if(value > next->value )
        {
            next=next->right ;
        }
        else
        {
            next=next->left ;
        }
    }
    if(value > temp->value)
    {
        temp->right=newnode ;

    }
    else
    {
        temp->left=newnode ;
    }

    return current ;
}
void post_order(node *t)
{
    if(t==NULL)
        return ;
    post_order(t->left);
    post_order(t->right);
    printf("%d\n",t->value );
}

int main()
{
    node *current=NULL ;
    int a  ;
    while(scanf("%d",&a)!=EOF)
    {
        current= create(current,a);
    }
    node *t=current ;
    post_order(t);


}
```

13 cf-Omkar and Heavenly Tree

Link

This problem is based on **observation**

Algorithm used:

```
#include<bits/stdc++.h>
using namespace std ;
int main()
{
     int t ;
    cin>>t ;
    while(t--)
    {
        int n,m ;
        cin>>n>>m ;
        int a,b,c ;
        bool res[m];
        for(int i=1 ; i<=m ; i++)
        {
            cin>>a>>b>>c ;
            res[b]=true ;
        }
        int root ;
        for(int i=1 ; i<=n ; i++)
        {
            if(!res[i])
            {
                root=i ;
                break ;
            }
        }
        for(int i=1 ; i<=n ; i++)
        {
            if(i==root)
                continue ;
            else
            {
                cout<<root<<" "<<i<<endl ;
            }
        }
    }
}
```

14 leetcode-Invert Binary Tree

Link

This problem is based on **binary search tree**
Algorithm used: **bst**

```

class Solution {
    void pt(TreeNode * temp)
    {
        if(temp==NULL)
        return  ;
        TreeNode * t=temp->left ;
        temp->left=temp->right ;
        temp->right=t ;

        invertTree(temp->left);
        invertTree(temp->right);
    }
public:
    TreeNode* invertTree(TreeNode* root) {
         TreeNode * temp=root ;
        pt(temp);
        return temp ;
    }
};
```

15 uva-wedding shoping

Link

This problem is based on **DP**
Algorithm used:

```
#include<bits/stdc++.h>
using namespace std ;
int price[25][25];
int dp[205][25];
int M,C ;
int shop(int money,int g)
{
    //cout<<"shop("<<money<<","<<g<<")"<<endl ;
    if(money<0)
        return -1e9 ;
    if(g==C)
        return M-money ;
    if(dp[money][g]!=-1)
        return dp[money][g];
    int ans=-1 ;
    for(int k=1; k<=price[g][0]; k++)
    {
        ans=max(ans,shop(money-price[g][k],g+1)) ;
    }
    return dp[money][g]=ans ;


}

int main()
{

    int t ;
    scanf("%d",&t);
    while(t--)
    {
        scanf("%d%d",&M,&C);
        for(int i=0 ; i<C ; i++)
        {
            scanf("%d",&price[i][0]);
            for(int j=1; j<=price[i][0]; j++)
            {
                scanf("%d",&price[i][j]);
            }
        }
        memset(dp,-1,sizeof(dp));
        int ans=shop(M,0);
        if(ans<0)
        {
            printf("no solution\n");
        }
        else
            printf("%d\n",ans);

    }
    return 0 ;
}
```

16 leetcode-LCS

Link

This problem is based on **LCS**
Algorithm used:

```
class Solution {
public:
    int longestCommonSubsequence(string text1, string text2) {
      int s=text1.size();
      int t=text2.size();
      int dp[s+1][t+1];

      for(int i=0 ;i<=s;i++)
      dp[i][0]=0 ;
      for(int i=0;i<=t ;i++)
      dp[0][i]=0 ;

      for(int i=1 ;i<=s ;i++)
      {
          for(int j=1 ;j<=t ;j++)
          {
              if(text1[i-1]==text2[j-1])
              {
                  dp[i][j]=1+dp[i-1][j-1];
              }
              else
              dp[i][j]=max(dp[i-1][j],dp[i][j-1]);
          }
      }
      return dp[s][t];

    }
};
```

17 codeforces-DZY Loves Chemistry

Link

This problem is based on **connected component**
Algorithm used: **dfs**

```
#include<bits/stdc++.h>
using namespace std ;
vector<int>adj[60];
vector<bool>vis(60,false);
void optimize()
{
ios_base::sync_with_stdio(false);
cin.tie(NULL);
}
int c=0 ;
void dfs(int src)
{
  vis[src]=true ;
   c++ ;
  for(auto u: adj[src])
  {
      if(!vis[u])
      {
          dfs(u);
      }
  }
}


int main()
{
optimize() ;

int t=1 ;

while(t--)
{
int n,m ;
cin>>n>>m ;


for(int i=0 ;i<m ;i++)
{
    int x,y ;
    cin>>x>>y ;
    adj[x].push_back(y);
    adj[y].push_back(x);
}

long long  ans=1 ;
for(int i=1;i<=n ;i++)
{
    if(!vis[i])
    {
        dfs(i) ;

    ans*=pow(2,c-1);

    }
    c=0 ;
}


cout<<ans<<endl ;

}

}
```

18 leetcode-Network Delay Time

Link

This problem is based on **SSSP**
Algorithm used: **Dijkstra**

```cpp
class Solution {
    vector<pair<int, int>> adj[110];
    vector<bool> vis{vector<bool>(110, false)};
    vector<int> dist{vector<int>(110, 1e9)};

    void dijkstra(int src, int n) {
        vis[src] = true;
        priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;
        pq.push({0, src});
        dist[src] = 0;
        while (!pq.empty()) {
            int u = pq.top().second;
            pq.pop();
            for (auto x : adj[u]) {
                int v = x.first;
                int w = x.second;
                if (dist[u] + w < dist[v]) {
                    dist[v] = dist[u] + w;
                    pq.push({dist[v], v});
                }
            }
        }
    }

public:
    int networkDelayTime(vector<vector<int>>& times, int n, int k) {
        int x = times.size();
        for (int i = 0; i < x; i++) {
            adj[times[i][0]].push_back({times[i][1], times[i][2]});
        }
        dijkstra(k, n);
        int mx = 0;
        for (int i = 1; i <= n; i++) {
            if (dist[i] == 1e9)
                return -1;
            mx = max(mx, dist[i]);
        }
        return mx;
    }
};
```

19 leetcode-Min Cost to Connect All Points

Link

This problem is based on **MST**
Algorithm used: **kruskal**

```cpp
class Solution {

    struct Edge{
    int src,des,w ;
};

struct Compare{
    bool operator()(const Edge &a,const Edge &b){
        return a.w>b.w ;
    }
};
int findParent(int node,vector<int>&parent)
{
    if(parent[node]==-1)
        return node ;

    return findParent(parent[node],parent);

}
int kruskalMst(vector<Edge>&edges ,int numVertex)
{
   int ans=0,e=0 ;
    vector<int>parent(numVertex,-1);

    priority_queue<Edge ,vector<Edge>,Compare>pq(edges.begin(),edges.end());
    while(!pq.empty())
    {
        Edge nextEdge=pq.top();
        pq.pop();
        int x=findParent(nextEdge.src,parent);
         int y=findParent(nextEdge.des,parent);
        if(x!=y)
        {
            ans+=nextEdge.w;
             e++ ;
            parent[x]=y ;
        }
        if(e==numVertex-1)
        break ;
    }
   return ans ;
}
public:
    int minCostConnectPoints(vector<vector<int>>& points) {

        vector<Edge>point;
        int x=points.size();
        for(int i=0 ;i<x ;i++)
        {
            for(int j=i+1 ;j<x ;j++)
            {
               int x1=points[i][0];
               int y1=points[i][1] ;
               int x2=points[j][0];
               int y2=points[j][1] ;
               point.push_back({i,j,abs(x1-x2)+abs(y1-y2)});
            }
        }
      return  kruskalMst(point,x);

    }
};
```

20 uva-10405 - Longest Common Subsequence

Link

This problem is based on **lcs**
Algorithm used: **lcs**

```
#include<bits/stdc++.h>
using namespace std ;

void optimize()
{
ios_base::sync_with_stdio(false);
cin.tie(NULL);
}
int dp[1001][1001];
int lcs(string text1,string text2)
{
    int s=text1.size();
    int o=text2.size();

    for(int i=0 ;i<=s ;i++)
        dp[i][0]=0 ;
    for(int i=0 ;i<=o ;i++)
        dp[0][i]=0 ;

    for(int i=1 ;i<=s ;i++)
    {
        for(int j=1 ;j<=o ;j++)
        {
            if(text1[i-1]==text2[j-1])
            {
                dp[i][j]=1+dp[i-1][j-1];
            }
            else
            {
                dp[i][j]=max(dp[i-1][j],dp[i][j-1]);
            }
        }


    }

    return dp[s][o];
}

int main()
{
optimize() ;
char s[1010],t[1010];

while(gets(s) && gets(t))
{

 printf("%d\n",lcs(s,t));

}

}
```

21 uva-graph connectivity

Link

This problem is based on **dfs**
Algorithm used:

```cpp
#include<bits/stdc++.h>
using namespace std;

bool vis[30];
vector<int>adj[30];

int charToNumber(char c) {
    return c - 'A' + 1;
}


void dfs(int node) {
    vis[node] = true;
    for (int neighbor : adj[node]) {
        if (!vis[neighbor]) {
            dfs(neighbor);
        }
    }
}

int main() {
int TC, V;
  char c, a, b;
  char input[10];

  scanf("%d", &TC);
  while(TC--)
  {
    memset(vis, false, sizeof(vis));
    cin >> c;
    V = c - '0' - 16;
    while(getchar() != '\n');
    while(gets(input) && sscanf(input, "%c%c", &a, &b) == 2)
    {
      int u, v;
      u = a - '0' - 16;
      v = b - '0' - 16;
      adj[u].push_back(v);
      adj[v].push_back(u);
    }
    int ans=0 ;
    for(int i=1 ;i<=V ;i++)
        {
            if(!vis[i])
            {
                ans++ ;
                dfs(i);
            }
        }
        printf("%d\n",ans);
    if(TC)
      printf("\n");
      for(int i=0; i<30; i++)
      adj[i].clear();

  }

  return 0;
}
```

22 leetcoe-Find the Town Judge

Link

This problem is based on **dfs**
Algorithm used:

```
class Solution {
public:


    int findJudge(int n, vector<vector<int>>& trust) {

        int x=trust.size();
        vector<pair<int,int>>trustCount(n+1,{0,0});

        for(auto u:trust)
        {
            int a=u[0];
            int b=u[1] ;
           trustCount[a].first++ ;
            trustCount[b].second++ ;

        }

       for(int i=1 ;i<=n ;i++)
       {
           if(trustCount[i].first==0 && trustCount[i].second==n-1)
           return i ;
       }
       return -1 ;





    }
};
```

23 leetcode-Search in a Binary Search Tree

Link

This problem is based on
Algorithm used:

```

 TreeNode * print(TreeNode * root,int val)
 {
     TreeNode * ans ;

     if(root==NULL)
     return root;

    if(root->val==val){
        return root ;
    }

        if(root->val > val)
     return print(root->left,val);
     else
      return print(root->right,val);
 }
class Solution {
public:
int f=0 ;
    TreeNode* searchBST(TreeNode* root, int val) {
       TreeNode *t=root ;
       return print(root,val);

    }
};
```

24 leetcode-Minimum cost for ticket

Link

This problem is based on **dp**
Algorithm used:

```
class Solution {
public:
    int mincostTickets(vector<int>& days, vector<int>& costs) {

      int n=days.size(),lastDay=days.back();
      vector<int>dp(lastDay+1),isTravelDay(lastDay+1);
     // memset(dp,0,lastDay+1);
      for(auto day:days)
      {
          isTravelDay[day]=true;
      }

      for(int i=1 ;i<=lastDay ;i++)
      {
          if(!isTravelDay[i])
          {
              dp[i]=dp[i-1];
              continue ;
          }
          dp[i]=dp[i-1]+costs[0];
          dp[i]=min(dp[i],dp[max(i-7,0)]+costs[1]);
          dp[i]=min(dp[i],dp[max(i-30,0)]+costs[2]);

      }

      for(int i=1;i<=lastDay ;i++)
      {
          cout<<dp[i]<<" ";
      }

         return dp.back();
    }
};
```

25 cf-forever winter

Link

This problem is based on
Algorithm used:

```

#include<bits/stdc++.h>
using namespace std ;
const int N=210 ;

//vector<int>vis(210,0);
//vector<int>parent(210,-1);
```

```
//void bfs(int src)
//{
//    vis[src]=1 ;
//    queue<int>q ;
//    q.push(src);
//    while(!q.empty())
//    {
//        int u=q.front();
//        q.pop();
//        for(auto v:adj[u])
//        {
//            if(!vis[v])
//            {
//                vis[v]=1 ;
//                parent[v]=u ;
//                q.push(v);
//            }
//        }
//    }
//}
//void dfs(int src)
//{
//    vis[src]=1 ;
//    for(auto u: adj[src])
//    {
//        if(!vis[u])
//        {
//            parent[u]=src ;
//            dfs(u);
//        }
//    }
//}
void optimize()
{
ios_base::sync_with_stdio(false);
cin.tie(NULL);
}

int main()
{
optimize() ;

int t ;
cin>>t ;
while(t--)
{
    int adj[N]={0};
int n,m ;
cin>>n>>m ;

for(int i=0 ;i<m ;i++)
{
    int x,y ;
    cin>>x>>y ;
     adj[x]++ ;
     adj[y]++ ;

}

int c=0 ;

for(int i=1;i<=n;i++)
{
   if(adj[i]==1)
   {
       c++ ;
   }
}
cout<<m-c<<" "<<c/(m-c)<<endl ;


}

}
```

26 uva-heavy cargo

Link

This problem is based on **kruskal algorithm**
Algorithm used:

```

#include<bits/stdc++.h>
using namespace std ;

struct city
{
    string src,des ;
    int w ;
} edges[20000];
long long n,m ;
string x,y ;
map<string,string >parent ;

bool cmp(city a, city b)
{
    return a.w>b.w ;
}

string find_root(string r)
{
    if(parent[r]=="")
    {
        return r ;
    }
    return parent[r]=find_root(parent[r]);
}
long long kruskal()
{
    sort(edges,edges+m,cmp);
    long long ans=9999999999;
    for(long long  i=0 ; i<m ; i++)
    {
        string u=find_root(edges[i].src);
        string v=find_root(edges[i].des);
        if(u!=v)
        {
            parent[u]=v ;
            if(ans>edges[i].w)
                ans=edges[i].w ;
        }
        if(find_root(x)==find_root(y))
        {
            return ans ;
        }
    }
    return ans ;
}

int main()
```

```
{

    long long  caseno=0 ;
    while(scanf("%lld%lld",&n,&m)==2 &&(n+m)!=0)
    {
        parent.clear();
        for(long long  i=0 ; i<m; i++)
        {
            cin>>edges[i].src>>edges[i].des>>edges[i].w ;

        }
        cin>>x>>y ;
        printf("Scenario #%lld\n%lld tons\n\n",++caseno,kruskal());
    }

}
```

27

Link

This problem is based on
Algorithm used: