

# 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**

```
1
2 #include<bits/stdc++.h>
3 using namespace std ;
4
5
6 bool find_negative_cycle(vector<pair<int,int>>adj[],int noVertices ,int source)
7 {
8     vector<int>distance(noVertices,numeric_limits<int>::max());
9     distance[source]=0 ;
10
11     //Relaxation
12     for(int i=0 ;i<noVertices-1;i++)
13     {
14         for(int u=0;u<noVertices;u++)
15         {
16             for(auto edges: adj[u])
17             {
18                 int v=edges.first ;
19                 int weight=edges.second ;
20                 if(distance[u]!=numeric_limits<int>::max() && distance[u]+weight<distance[v])
21                 {
22                     distance[v]=distance[u]+weight ;
23                 }
24             }
25         }
26     }
27
28     //additional iteration to check negative cycle
29     for(int i=0 ;i<noVertices ;i++)
30     {
31         for(auto edges: adj[i])
32         {
33             int v=edges.first ;
34             int weight=edges.second ;
35             if(distance[i]!=numeric_limits<int>::max() && distance[i]+weight<distance[v])
36             {
37                 return true ;
38             }
39         }
40     }
41     return false ;
42 }
43
44
```

```

45
46 int main()
47 {
48
49 int t ,n,m;//star system , wormholes
50 scanf ("%d",&t);
51 while (t--)
52 {
53     scanf ("%d%d",&n,&m);
54     vector<pair<int,int>>adj[n+10] ;
55     for(int i=0 ;i<m ;i++)
56     {
57         int u,v,w ;
58         scanf ("%d%d%d",&u,&v,&w);
59         adj[u].push_back({v,w});
60
61     }
62
63     if(find_negative_cycle(adj,n,0))
64     {
65         printf("possible\n");
66     }
67     else
68     {
69         printf("not possible\n");
70     }
71 }
72 }
73
74 return 0 ;
75 }

```

## 2 King Escape

[Link](#)

Algorithm used: **dfs**

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  const int N=1e3+10 ;
4
5
6  int visited[N][N] ;
7  int n,ax,ay,bx,by,cx,cy ;
8
9  bool checkValidMove(int nx,int ny) //new x and new y
10 {
11     if(nx==ax || ny==ay) // check horizontal and vertical
12     {
13         return true ;
14     }
15     if(abs(nx-ax)==abs(ny-ay)) //check queen and king are same diagonal or not
16     {
17         return true ;
18     }
19
20     return false ;
21 }
22
23 void dfs(int x,int y) {
24
25     int dx[8] = {1, 1, 1, 0, 0, -1, -1, -1};
26     int dy[8] = {1, 0, -1, 1, -1, 1, 0, -1};
27
28     if(x<1 || x>n || y<1 || y>n)
29     {
30         return ;
31     }
32     if(visited[x][y])
33     {
34         return ;
35     }
36
37     if(checkValidMove(x,y))
38     {
39         return ;
40     }
41     visited[x][y]=1 ;
42
43     for (int i = 0; i < 8; ++i) {
44         int nx = x + dx[i];
45         int ny = y + dy[i];
46
47         dfs(nx,ny); //check for new cell
48     }
49 }
50
51
52 int main() {
53     cin >> n;
54     // queen location
55     cin >> ax >> ay;
56     // king location
57     cin >> bx >> by;
58     // target location
59     cin >> cx >> cy;
60
61     dfs(bx,by);
62     if (visited[cx][cy])
63         cout << "YES" << endl;
64     else
65         cout << "NO" << endl;
66
67     return 0;
68 }

```

### 3 leetCode-Find if Path Exists in Graph

[Link](#)

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

```
1
2 class Solution {
3
4     void dfs(int source ,vector<int>adj[], vector<bool>&visit )
5     {
6         visit[source]=true ;
7         for(auto child: adj[source]) //traverse adjacent nodes of source
8         {
9             if(!visit[child])
10            {
11                visit[child]=true ;
12                dfs(child,adj,visit);
13            }
14        }
15    }
16 public:
17     bool validPath(int n, vector<vector<int>>& edges, int source, int destination) {
18
19         int x=edges.size() ;
20         vector<int>adj[n];
21         vector<bool>visit(n,false) ;
22
23         for(int i=0 ;i<x ;i++)
24         {
25             adj[edges[i][0]].push_back(edges[i][1]);
26             adj[edges[i][1]].push_back(edges[i][0]);
27
28         }
29
30
31         dfs(source,adj,visit); // traverse from source if found destination return true
32
33         if( (visit[source] && !visit[destination]) || (!visit[source] &&visit[destination
34             ]))
35         {
36             return false ;
37         }
38
39         return true ;
40
41     }
42 };
```

### 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**

```

1 #include<bits/stdc++.h>
2 using namespace std ;
3 const int N=200000 ;
4 int visited[N];
5
6 void bfs(vector<int>graph[],int parent[],int source)
7 {
8     queue<int>q ;
9     q.push(source);
10    visited[source]=1 ;
11    parent[source]=-1 ;
12    while(!q.empty())
13    {
14        int v=q.front();
15        for(auto u: graph[v])
16        {
17            if(!visited[u])
18            {
19                parent[u]=v ;
20                visited[u]=true ;
21                q.push(u);
22            }
23        }
24        q.pop() ;
25    }
26 }
27
28 }
29
30 void path(int parent[],int src) // using recursion for finding parent ( destination to source)
31 {
32
33     if(parent[src]==-1){
34         printf("%d ",1);
35         return ;
36     }
37
38
39     path(parent ,parent[src]);
40     printf("%d ",src);
41 }
42
43 int main()
44 {
45     int n ;
46     cin>>n ;
47     vector<int>graph[n+5];
48
49     int parent[n]={} ;
50     //8
51     //1 1 2 2 3 2 5
52
53     for(int i=2 ;i<=n ;i++)
54     {
55         int a;
56         cin>>a ;
57         graph[i].push_back(a);
58         graph[a].push_back(i);
59     }
60
61     bfs(graph,parent,1) ;
62     path(parent,n);
63
64

```

```

65  /** you can use this path(parent,n) method or this block of code to print path **
66
67  /*
68  int src=n ;
69  set<int>path ;
70  while(src!=-1)
71  {
72      path.insert(src);
73      src=parent[src] ;
74  }
75  for(auto u: path)
76      cout<<u<<" " ;*/
77
78
79  }

```

## 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**

```

1  class Solution {
2  public:
3      TreeNode* sortedArrayToBST(vector<int>& nums) {
4          int size = nums.size();
5          int half = size/2;
6
7          if(size == 0)
8              return NULL; // if no child
9
10
11         TreeNode* root = new TreeNode(nums[half]);
12         //create a root node
13         if(size == 1)
14             return root;
15
16         //left child ( start to half of nums vector )
17         vector<int>left(nums.begin(), nums.begin() + half);
18         //right child( half to end of nums vector )
19         vector<int>right(nums.begin() + half + 1, nums.end());
20         root->left = sortedArrayToBST(left);
21         root->right = sortedArrayToBST(right);
22
23         return root;
24     }
25 };

```

## 6 Uva - Dividing coins

[Link](#)

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

```

1
2 #include<bits/stdc++.h>
3 using namespace std ;
4 const int N = 110;
5 const int MAXN = 102*500;
6 int weight[N];
7 int dp[N][MAXN] ;
8
9 int sum ;
10
11 //used 2D array
12
13 int knapSack(int m)
14 {
15     int max_weight=sum/2 ; // here max capacity will be half of total weight
16
17     for(int i=0; i <=max_weight; i++)
18         dp[0][i] = 0;
19
20
21     for(int i=1 ;i<=m;i++)
22     {
23         for(int w=0 ;w<=max_weight ;w++)
24         {
25             if(weight[i]<=w)
26             {
27                 dp[i][w]=max(dp[i-1][w],(dp[i-1][w-weight[i]]+weight[i]));
28             }
29             else
30             {
31                 dp[i][w]=dp[i-1][w] ;
32             }
33         }
34     }
35
36
37
38     return dp[m][max_weight];
39 }
40
41 //using 1D Array (memory optimization)
42 //int knapSack(int m)
43 //{
44 //    int w=sum/2 ;
45 //    for(int i=1 ;i<=m ;i++)
46 //    {
47 //        for(int j=w ;j>0 ;j--)
48 //        {
49 //            if(weight[i]<=j)
50 //            {
51 //                dp[j]=max(dp[j],weight[i]+dp[j-weight[i]]);
52 //            }
53 //        }
54 //    }
55 //    return dp[w];
56 //}
57 //
58
59 int main()
60 {
61
62     int t ;
63     scanf("%d",&t);
64     while(t--)
65     {
66         int m ;

```

```

67 scanf("%d",&m);
68
69 sum=0 ;
70 for(int i=1 ;i<=m ;i++)
71 {
72     scanf("%d",&weight[i]) ;
73     sum+=weight[i] ;
74 }
75
76 printf("%d\n", sum - 2*knapSack(m) );
77 }
78 return 0 ;
79 }

```

## 7 Codeforces-KnapSack

[Link](#)

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

```

1  #include<bits/stdc++.h>
2  using namespace std ;
3
4
5  long long  total_weight ;
6  long long  oneItem ;
7  long long  weight ;
8  vector<int>items ;
9
10 int main()
11 {
12
13     int t ;
14     cin>>t ;
15     while(t--)
16     {
17         long long n,w ;
18         cin>>n>>w ;
19
20
21         total_weight=0 ;
22         oneItem=0 ;
23
24         for(int i=1 ; i<=n ; i++)
25         {
26             cin>>weight ;
27             if(weight <= w )
28             {
29                 if(weight>=(w+1)/2)
30                 {
31                     oneItem=i ;
32                 }
33                 else if ( total_weight < (w+1)/2)
34                 {
35                     items.push_back(i);
36                     total_weight+=weight ;
37                 }
38             }
39
40         }
41
42         if(oneItem>0)
43         {

```



```

44         cout<<1<<'\n'<<oneItem ;
45
46     }
47     else
48     {
49         if(total_weight>=(w+1)/2)
50         {
51             cout<<items.size()<<endl ;
52             for(auto u: items)
53                 cout<<u<<" " ;
54
55             }
56         else
57         {
58             cout<<-1 ;
59         }
60     }
61
62     cout<<endl;
63     items.clear();
64
65 }
66
67 }
68
69 \begin{figure}[H]
70 \end{figure}

```

## 8 LeetCode-Is Subsequence

[Link](#)

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

```

1  class Solution {
2  public:
3      bool isSubsequence(string s, string t) {
4
5          int sub=s.size();
6          int original=t.size();
7
8          int dp[sub+1][original+1];
9
10         for(int i=0 ;i<=sub ;i++)
11             dp[i][0]=0 ; //fill first row and first column with zero
12         for(int i=0 ; i<=original;i++)
13             dp[0][i]=0 ;
14
15         for(int i=1 ;i<=sub;i++)
16         {
17             for(int j=1 ;j<=original ;j++)
18             {
19                 if(s[i-1]==t[j-1])
20                 {
21                     dp[i][j]=1+dp[i-1][j-1]; //if match subsequence with original string
22                 }
23                 else
24                 {
25                     dp[i][j]=max(dp[i][j-1],dp[i-1][j]); // if not match
26                 }
27             }
28         }
29     }

```

```

30     if(dp[sub][original]==sub)
31         return true ;
32     else
33         return false ;
34
35 }
36 };

```

## 9 Uva-dominos2

[Link](#)

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

```

1
2 #include<bits/stdc++.h>
3 using namespace std ;
4 const int N=10005 ;
5 bool vis[N];
6 int c ;
7 vector<int>adj[N];
8
9 void dfs(int src)
10 {
11     c++ ;
12     vis[src]=true ;
13     //cout<<src<<endl ;
14     for(auto u: adj[src])
15     {
16         if(!vis[u])
17         {
18             vis[u] =true ;
19             dfs(u);
20         }
21     }
22 }
23
24
25 int main()
26 {
27
28
29     int t ;
30     scanf("%d",&t);
31     while(t--)
32     {
33         int n,m,l ;
34         scanf("%d%d%d",&n,&m,&l);
35         for(int i=0; i<m ; i++)
36         {
37             int x,y ;
38             scanf("%d%d",&x,&y);
39             adj[x].push_back(y);
40         }
41         int fallDominos[l];
42         for(int i=0 ; i<l ; i++)
43         {
44             int a ;
45             scanf("%d",&a);
46             fallDominos[i]=a ;
47         }
48         for(int i=0 ; i<l ; i++)
49         {

```

```

50         if(!vis[fallDominos[i]])
51             dfs(fallDominos[i]);
52     }
53
54
55     printf("%d\n",c);
56     c=0 ;
57     fill(vis,vis+N,false);
58     for(int i=0; i<n ; i++)
59     {
60         adj[i].clear();
61     }
62
63
64
65 }
66
67 }

```

## 10 codeforces-: Rumor

[Link](#)

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

```

1  #include<bits/stdc++.h>
2  using namespace std ;
3  #define ll long long
4  const int N=1e5+10 ;
5  vector<ll>adj[N] ;
6  vector<bool>visited(N,false) ;
7  ll cost[N] ;
8  ll ans=0 ;
9
10 ll dfs( ll src)
11 {
12
13     ll m=cost[src];
14     visited[src]=true ;
15     for(auto u:adj[src])
16     {
17         if(!visited[u])
18         {
19             m=min(m,dfs(u));
20         }
21     }
22     return m ;
23 }
24
25
26
27 int main()
28 {
29     ll n,m ;
30     cin>>n>>m ;
31     for(ll i=1; i<=n ; i++)
32     {
33         ll a ;
34         cin>>a ;
35         cost[i]=a ;
36     }
37     for(ll i=0 ; i<m ; i++)
38     {

```

```

39     ll x,y ;
40     cin>>x>>y ;
41     adj[x].push_back(y);
42     adj[y].push_back(x);
43 }
44
45 for(ll i=1 ; i<=n ; i++)
46 {
47     if(!visited[i])
48     {
49         ans+=dfs(i);
50     }
51 }
52 cout<<ans<<endl ;
53
54 }

```

## 11 leetCode- countCompleteComponents

[Link](#)

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

```

1  class Solution {
2      bool vis[6000];
3      int c=0 ;
4      int e=0,v=0 ;
5      vector<int>adj[6000];
6      void dfs(int src)
7      {
8          vis[src]=true ;
9          v++ ;
10         for(auto u: adj[src])
11         {
12             e++ ;
13             if(!vis[u])
14             {
15                 dfs(u);
16             }
17         }
18     }
19 public:
20     int countCompleteComponents(int n, vector<vector<int>>& edges) {
21         int x=edges.size();
22         for(int i=0 ;i<x ;i++)
23         {
24             adj[edges[i][0]].push_back(edges[i][1]);
25             adj[edges[i][1]].push_back(edges[i][0]);
26         }
27         for(int i=0 ;i<=n-1;i++)
28         {
29             if(!vis[i])
30             {
31
32                 dfs(i);
33                 //cout<<v<<" "<<e<<endl ;
34                 if(e/2==(v*(v-1))/2)
35                 {
36                     c++ ;
37                 }
38                 v=0 ;
39                 e=0 ;
40

```

```

41         }
42     }
43     return c ;
44 }
45 }
46 };

```

## 12 uva-binary search tree

[Link](#)

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

```

1
2 #include<bits/stdc++.h>
3 using namespace std ;
4
5 struct node
6 {
7     int value=0 ;
8     node *left=NULL ;
9     node *right=NULL ;
10 };
11
12 node *create(node *current,int value)
13 {
14     node *newnode=new node();
15     newnode->value=value ;
16     newnode->left=NULL ;
17     newnode->right=NULL ;
18
19     if(current==NULL)
20         return newnode ;
21
22     node *temp=current;
23     node *next=current ;
24     while(next!=NULL)
25     {
26         temp=next ;
27         if(value > next->value )
28         {
29             next=next->right ;
30         }
31         else
32         {
33             next=next->left ;
34         }
35     }
36     if(value > temp->value)
37     {
38         temp->right=newnode ;
39     }
40     else
41     {
42         temp->left=newnode ;
43     }
44
45     return current ;
46 }
47
48 void post_order(node *t)
49 {

```

```

50     if(t==NULL)
51         return ;
52     post_order(t->left);
53     post_order(t->right);
54     printf("%d\n",t->value );
55 }
56
57 int main()
58 {
59     node *current=NULL ;
60     int a ;
61     while(scanf("%d",&a)!=EOF)
62     {
63         current= create(current,a);
64     }
65     node *t=current ;
66     post_order(t);
67
68
69 }

```

## 13 cf-Omkar and Heavenly Tree

[Link](#)

This problem is based on **observation**

Algorithm used:

```

1  #include<bits/stdc++.h>
2  using namespace std ;
3  int main()
4  {
5      int t ;
6      cin>>t ;
7      while(t-->0)
8      {
9          int n,m ;
10         cin>>n>>m ;
11         int a,b,c ;
12         bool res[m];
13         for(int i=1 ; i<=m ; i++)
14         {
15             cin>>a>>b>>c ;
16             res[b]=true ;
17         }
18         int root ;
19         for(int i=1 ; i<=n ; i++)
20         {
21             if(!res[i])
22             {
23                 root=i ;
24                 break ;
25             }
26         }
27         for(int i=1 ; i<=n ; i++)
28         {
29             if(i==root)
30                 continue ;
31             else
32             {
33                 cout<<root<<" "<<i<<endl ;
34             }
35         }
36     }
37 }

```

## 14 leetcode-Invert Binary Tree

[Link](#)

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

```
1
2 class Solution {
3     void pt(TreeNode * temp)
4     {
5         if(temp==NULL)
6             return ;
7         TreeNode * t=temp->left ;
8         temp->left=temp->right ;
9         temp->right=t ;
10
11         invertTree(temp->left);
12         invertTree(temp->right);
13     }
14 public:
15     TreeNode* invertTree(TreeNode* root) {
16         TreeNode * temp=root ;
17         pt(temp);
18         return temp ;
19     }
20 };
```

## 15 uva-wedding shoping

[Link](#)

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

```
1 #include<bits/stdc++.h>
2 using namespace std ;
3 int price[25][25];
4 int dp[205][25];
5 int M,C ;
6 int shop(int money,int g)
7 {
8     //cout<<"shop("<<money<<","<<g<<")"<<endl ;
9     if(money<0)
10         return -1e9 ;
11     if(g==C)
12         return M-money ;
13     if(dp[money][g]!=-1)
14         return dp[money][g];
15     int ans=-1 ;
16     for(int k=1; k<=price[g][0]; k++)
17     {
18         ans=max(ans,shop(money-price[g][k],g+1)) ;
19     }
20     return dp[money][g]=ans ;
21
22 }
23
24
25 int main()
26 {
27
28     int t ;
```

```

29     scanf("%d",&t);
30     while(t-->0)
31     {
32         scanf("%d",&M,&C);
33         for(int i=0 ; i<C ; i++)
34         {
35             scanf("%d",&price[i][0]);
36             for(int j=1; j<=price[i][0]; j++)
37             {
38                 scanf("%d",&price[i][j]);
39             }
40         }
41         memset(dp,-1,sizeof(dp));
42         int ans=shop(M,0);
43         if(ans<0)
44         {
45             printf("no solution\n");
46         }
47         else
48             printf("%d\n",ans);
49     }
50     return 0 ;
51 }
52

```

## 16 leetcode-LCS

[Link](#)

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

```

1  class Solution {
2  public:
3      int longestCommonSubsequence(string text1, string text2) {
4          int s=text1.size();
5          int t=text2.size();
6          int dp[s+1][t+1];
7
8          for(int i=0 ;i<=s;i++)
9              dp[i][0]=0 ;
10         for(int i=0;i<=t ;i++)
11             dp[0][i]=0 ;
12
13         for(int i=1 ;i<=s ;i++)
14         {
15             for(int j=1 ;j<=t ;j++)
16             {
17                 if(text1[i-1]==text2[j-1])
18                 {
19                     dp[i][j]=1+dp[i-1][j-1];
20                 }
21                 else
22                     dp[i][j]=max(dp[i-1][j],dp[i][j-1]);
23             }
24         }
25         return dp[s][t];
26     }
27 }
28

```



## 17 codeforces-DZY Loves Chemistry

[Link](#)

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

```
1 #include<bits/stdc++.h>
2 using namespace std ;
3 vector<int>adj[60];
4 vector<bool>vis(60,false);
5 void optimize()
6 {
7     ios_base::sync_with_stdio(false);
8     cin.tie(NULL);
9 }
10 int c=0 ;
11 void dfs(int src)
12 {
13     vis[src]=true ;
14     c++ ;
15     for(auto u: adj[src])
16     {
17         if(!vis[u])
18         {
19             dfs(u);
20         }
21     }
22 }
23
24
25 int main()
26 {
27     optimize() ;
28
29     int t=1 ;
30
31     while(t--)
32     {
33         int n,m ;
34         cin>>n>>m ;
35
36
37         for(int i=0 ;i<m ;i++)
38         {
39             int x,y ;
40             cin>>x>>y ;
41             adj[x].push_back(y);
42             adj[y].push_back(x);
43         }
44
45         long long ans=1 ;
46         for(int i=1;i<=n ;i++)
47         {
48             if(!vis[i])
49             {
50                 dfs(i) ;
51
52                 ans*=pow(2,c-1);
53
54             }
55             c=0 ;
56         }
57
58
59     cout<<ans<<endl ;
```

```
60 }
61 }
62
63 }
```

## 18 leetcode-Network Delay Time

[Link](#)

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

```
1 class Solution {
2     vector<pair<int, int>> adj[110];
3     vector<bool> vis{vector<bool>(110, false)};
4     vector<int> dist{vector<int>(110, 1e9)};
5
6     void dijkstra(int src, int n) {
7         vis[src] = true;
8         priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> pq;
9         pq.push({0, src});
10        dist[src] = 0;
11        while (!pq.empty()) {
12            int u = pq.top().second;
13            pq.pop();
14            for (auto x : adj[u]) {
15                int v = x.first;
16                int w = x.second;
17                if (dist[u] + w < dist[v]) {
18                    dist[v] = dist[u] + w;
19                    pq.push({dist[v], v});
20                }
21            }
22        }
23    }
24
25 public:
26     int networkDelayTime(vector<vector<int>>& times, int n, int k) {
27         int x = times.size();
28         for (int i = 0; i < x; i++) {
29             adj[times[i][0]].push_back({times[i][1], times[i][2]});
30         }
31         dijkstra(k, n);
32         int mx = 0;
33         for (int i = 1; i <= n; i++) {
34             if (dist[i] == 1e9)
35                 return -1;
36             mx = max(mx, dist[i]);
37         }
38         return mx;
39     }
40 };
```

## 19 leetcode-Min Cost to Connect All Points

[Link](#)

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

```
1 class Solution {
```

```

2
3     struct Edge{
4         int src,des,w ;
5     };
6
7     struct Compare{
8         bool operator()(const Edge &a,const Edge &b){
9             return a.w>b.w ;
10        }
11    };
12    int findParent(int node,vector<int>&parent)
13    {
14        if(parent[node]==-1)
15            return node ;
16
17        return findParent(parent[node],parent);
18    }
19
20    int kruskalMst(vector<Edge>&edges ,int numVertex)
21    {
22        int ans=0,e=0 ;
23        vector<int>parent(numVertex,-1);
24
25        priority_queue<Edge ,vector<Edge>,Compare>pq(edges.begin(),edges.end());
26        while(!pq.empty())
27        {
28            Edge nextEdge=pq.top();
29            pq.pop();
30            int x=findParent(nextEdge.src,parent);
31            int y=findParent(nextEdge.des,parent);
32            if(x!=y)
33            {
34                ans+=nextEdge.w;
35                e++ ;
36                parent[x]=y ;
37            }
38            if(e==numVertex-1)
39                break ;
40        }
41        return ans ;
42    }
43    public:
44        int minCostConnectPoints(vector<vector<int>>& points) {
45
46            vector<Edge>point;
47            int x=points.size();
48            for(int i=0 ;i<x ;i++)
49            {
50                for(int j=i+1 ;j<x ;j++)
51                {
52                    int x1=points[i][0];
53                    int y1=points[i][1] ;
54                    int x2=points[j][0];
55                    int y2=points[j][1] ;
56                    point.push_back({i,j,abs(x1-x2)+abs(y1-y2)});
57                }
58            }
59            return kruskalMst(point,x);
60
61        }
62    };

```

## 20 uva-10405 - Longest Common Subsequence

[Link](#)

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

```
1 #include<bits/stdc++.h>
2 using namespace std ;
3
4 void optimize()
5 {
6     ios_base::sync_with_stdio(false);
7     cin.tie(NULL);
8 }
9 int dp[1001][1001];
10 int lcs(string text1,string text2)
11 {
12     int s=text1.size();
13     int o=text2.size();
14
15     for(int i=0 ;i<=s ;i++)
16         dp[i][0]=0 ;
17     for(int i=0 ;i<=o ;i++)
18         dp[0][i]=0 ;
19
20     for(int i=1 ;i<=s ;i++)
21     {
22         for(int j=1 ;j<=o ;j++)
23         {
24             if(text1[i-1]==text2[j-1])
25             {
26                 dp[i][j]=1+dp[i-1][j-1];
27             }
28             else
29             {
30                 dp[i][j]=max(dp[i-1][j],dp[i][j-1]);
31             }
32         }
33     }
34
35     }
36
37     return dp[s][o];
38 }
39
40 int main()
41 {
42     optimize() ;
43     char s[1010],t[1010];
44
45     while(gets(s) && gets(t))
46     {
47
48         printf("%d\n",lcs(s,t));
49     }
50 }
51
52 }
```

## 21 uva-graph connectivity

[Link](#)

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

```

1  #include<bits/stdc++.h>
2  using namespace std;
3
4  bool vis[30];
5  vector<int>adj[30];
6
7  int charToNumber(char c) {
8      return c - 'A' + 1;
9  }
10
11
12 void dfs(int node) {
13     vis[node] = true;
14     for (int neighbor : adj[node]) {
15         if (!vis[neighbor]) {
16             dfs(neighbor);
17         }
18     }
19 }
20
21 int main() {
22     int TC, V;
23     char c, a, b;
24     char input[10];
25
26     scanf("%d", &TC);
27     while(TC--)
28     {
29         memset(vis, false, sizeof(vis));
30         cin >> c;
31         V = c - '0' - 16;
32         while(getchar() != '\n');
33         while(gets(input) && sscanf(input, "%c%c", &a, &b) == 2)
34         {
35             int u, v;
36             u = a - '0' - 16;
37             v = b - '0' - 16;
38             adj[u].push_back(v);
39             adj[v].push_back(u);
40         }
41         int ans=0 ;
42         for(int i=1 ;i<=V ;i++)
43         {
44             if(!vis[i])
45             {
46                 ans++ ;
47                 dfs(i);
48             }
49         }
50         printf("%d\n",ans);
51         if(TC)
52             printf("\n");
53         for(int i=0; i<30; i++)
54             adj[i].clear();
55     }
56
57     return 0;
58 }
59

```

## 22 leetcoe-Find the Town Judge

[Link](#)

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

```
1 class Solution {
2 public:
3
4
5     int findJudge(int n, vector<vector<int>>& trust) {
6
7         int x=trust.size();
8         vector<pair<int,int>>trustCount(n+1,{0,0});
9
10        for(auto u:trust)
11        {
12            int a=u[0];
13            int b=u[1] ;
14            trustCount[a].first++ ;
15            trustCount[b].second++ ;
16
17        }
18
19        for(int i=1 ;i<=n ;i++)
20        {
21            if(trustCount[i].first==0 && trustCount[i].second==n-1)
22                return i ;
23        }
24        return -1 ;
25
26
27
28
29    }
30 };
```

## 23 leetcode-Search in a Binary Search Tree

[Link](#)

This problem is based on  
Algorithm used:

```
1
2  TreeNode * print(TreeNode * root,int val)
3  {
4      TreeNode * ans ;
5
6      if(root==NULL)
7          return root;
8
9      if(root->val==val){
10         return root ;
11     }
12
13     if(root->val > val)
14         return print(root->left,val);
15     else
16         return print(root->right,val);
17 }
18 class Solution {
19 public:
20     int f=0 ;
21     TreeNode* searchBST(TreeNode* root, int val) {
```

```

22     TreeNode *t=root ;
23     return print(root,val);
24
25 }
26 };

```

## 24 leetcode-Minimum cost for ticket

[Link](#)

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

```

1  class Solution {
2  public:
3      int mincostTickets(vector<int>& days, vector<int>& costs) {
4
5          int n=days.size(),lastDay=days.back();
6          vector<int>dp(lastDay+1),isTravelDay(lastDay+1);
7          // memset(dp,0,lastDay+1);
8          for(auto day:days)
9          {
10             isTravelDay[day]=true;
11         }
12
13         for(int i=1 ;i<=lastDay ;i++)
14         {
15             if(!isTravelDay[i])
16             {
17                 dp[i]=dp[i-1];
18                 continue ;
19             }
20             dp[i]=dp[i-1]+costs[0];
21             dp[i]=min(dp[i],dp[max(i-7,0)]+costs[1]);
22             dp[i]=min(dp[i],dp[max(i-30,0)]+costs[2]);
23
24         }
25
26         for(int i=1;i<=lastDay ;i++)
27         {
28             cout<<dp[i]<<" ";
29         }
30
31         return dp.back();
32     }
33 };

```

## 25 cf-forever winter

[Link](#)

This problem is based on  
Algorithm used:

```

1
2 #include<bits/stdc++.h>
3 using namespace std ;
4 const int N=210 ;
5
6 //vector<int>vis(210,0);
7 //vector<int>parent(210,-1);

```

```

8 //void bfs(int src)
9 //{
10 //    vis[src]=1 ;
11 //    queue<int>q ;
12 //    q.push(src);
13 //    while(!q.empty())
14 //    {
15 //        int u=q.front();
16 //        q.pop();
17 //        for(auto v:adj[u])
18 //        {
19 //            if(!vis[v])
20 //            {
21 //                vis[v]=1 ;
22 //                parent[v]=u ;
23 //                q.push(v);
24 //            }
25 //        }
26 //    }
27 //}
28 //void dfs(int src)
29 //{
30 //    vis[src]=1 ;
31 //    for(auto u: adj[src])
32 //    {
33 //        if(!vis[u])
34 //        {
35 //            parent[u]=src ;
36 //            dfs(u);
37 //        }
38 //    }
39 //}
40 void optimize()
41 {
42 ios_base::sync_with_stdio(false);
43 cin.tie(NULL);
44 }
45
46 int main()
47 {
48 optimize() ;
49
50 int t ;
51 cin>>t ;
52 while(t--)
53 {
54     int adj[N]={0};
55     int n,m ;
56     cin>>n>>m ;
57
58     for(int i=0 ;i<m ;i++)
59     {
60         int x,y ;
61         cin>>x>>y ;
62         adj[x]++ ;
63         adj[y]++ ;
64     }
65
66     int c=0 ;
67
68     for(int i=1;i<=n;i++)
69     {
70         if(adj[i]==1)
71         {
72             c++ ;
73         }
74     }

```



```

75 }
76 cout<<m-c<<" "<<c/(m-c)<<endl ;
77
78
79 }
80
81 }

```

## 26 uva-heavy cargo

[Link](#)

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

```

1
2 #include<bits/stdc++.h>
3 using namespace std ;
4
5 struct city
6 {
7     string src,des ;
8     int w ;
9 } edges[20000];
10 long long n,m ;
11 string x,y ;
12 map<string,string >parent ;
13
14 bool cmp(city a, city b)
15 {
16     return a.w>b.w ;
17 }
18
19 string find_root(string r)
20 {
21     if(parent[r]=="")
22     {
23         return r ;
24     }
25     return parent[r]=find_root(parent[r]);
26 }
27 long long kruskal()
28 {
29     sort(edges,edges+m,cmp);
30     long long ans=9999999999;
31     for(long long i=0 ; i<m ; i++)
32     {
33         string u=find_root(edges[i].src);
34         string v=find_root(edges[i].des);
35         if(u!=v)
36         {
37             parent[u]=v ;
38             if(ans>edges[i].w)
39                 ans=edges[i].w ;
40         }
41         if(find_root(x)==find_root(y))
42         {
43             return ans ;
44         }
45     }
46     return ans ;
47 }
48
49 int main()

```

```

50 {
51
52     long long    caseno=0 ;
53     while(scanf("%lld%lld",&n,&m)==2  &&(n+m)!=0)
54     {
55         parent.clear();
56         for(long long    i=0 ; i<m; i++)
57         {
58             cin>>edges[i].src>>edges[i].des>>edges[i].w ;
59
60         }
61         cin>>x>>y ;
62         printf("Scenario  %lld\n%lld  tons\n\n",++caseno,kruskal());
63     }
64
65 }

```

## 27

Link

This problem is based on  
Algorithm used: