অ্যারে কম্প্রেশন

❖ map এ Integer কে integer দিয়ে ম্যাপিং করা যায়:-

```
#include<bits/stdc++.h>
using namespace std;
int n;
int input[100];///-102 1 134565589 134565589 -102 66666668 134565589 66666668 -102 1 -2
void compress(){
    map<int,int>element;
    int assign=0,c=0,compressed[n];
    for(int i=0;i<n;i++){
            int x=input[i];
            if(element.find(x)==element.end()){///x not yet compressed
                    element[x]=assign;
                     printf("Mapping %d with %d\n",x,assign);
                    assign++;
            }
            x=element[x];
            compressed[c++]=x;
    }
    printf("Compressed
    Array:\n");
    for(int i=0;i<n;i++){
    printf("%d ",compressed[i]);
    }
    printf("\n");
}
int main(){
    cin>>n;
    for(int i=0;i<n;i++){
            scanf("%d",&input[i]);
    }
    compress();
    return 0;
}
```

* map এ string কে integer দিয়ে ম্যাপিং করা যায়:-

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    string s1,s2;
    map<string,int>mymap;
    int edge,assign=0;
    cin>>edge;
    for(int i=0;i<edge;i++){
         string s1,s2;
         cin>>s1>>s2;
         if(mymap.find(s1)==mymap.end()){
printf("Maping %s with %d\n",s1.c_str(),assign);
              mymap[s1]=assign++;
         }
         if(mymap.find(s2)==mymap.end()){
 printf("Maping %s with %d\n",s2.c_str(),assign);
              mymap[s2]=assign++;
     }
    int u=mymap[s1];
    int v=mymap[s2];
    printf("Edge: %d %d\n",u,v);
}
return 0;
}
```

Articulation Points and Bridges

# Suppose V is a vertex in the graph. Remove the vertex V and the edges joining it. Now if the graph gets disconnected or the number of disconnected component of the graph increases then V is the cut vertex of the graph.

Sample Code: Kingdom Unity **Problem Code:** KINGCON

```
using namespace std;
vector<int>edges[5000000];
int parent[mx],child[mx],low[mx],temp[mx];
bool vis[mx],is_cut[mx];
//int c;
void dfs(int u){
    static int c=0;
    vis[u]=true;
    low[u]=temp[u]=++c;
    for(int i=0;i<edges[u].size();i++){
        int v=edges[u][i];
        if(v==parent[u]) continue;
        else if(!vis[v]){
            child[u]++;
            parent[v]=u;
            dfs(v);
            low[u]=min(low[u],low[v]);
            if(low[v]>=temp[u]) is_cut[u]=true;
        }
        else{
            low[u]=min(low[u],temp[v]);
        }
    }
}
```

```
int main(){
    int t;
    scanf("%d",&t);
    while(t--){
        memset(vis,false,sizeof(vis));
        memset(is_cut,false,sizeof(vis));
        memset(parent,-1,sizeof(parent));
        memset(child,0,sizeof(child));
        memset(low,-1,sizeof(low));
        memset(temp,-1,sizeof(temp));
        int n,e;
        ll k;
        scanf("%d %d %lld",&n,&e,&k);

        for(int i=0;i<e;i++){
            int x,y;
            scanf("%d %d",&x,&y);
            edges[x].push_back(y);
            edges[y].push_back(x);
        }
//          parent[0]=-1;
        dfs(0);
//          is_cut[0]=child[0]>1?true:false;
        if(child[0]>1) is_cut[0]=true;
        else is_cut[0]=false;
        ll ans=0;
        for(int i=0;i<n;i++){
            if(is_cut[i]) ans+=k;
        }
        printf("%lld\n",ans);
        for(int i=0;i<n;i++) edges[i].clear();
    }
    return 0;
}
```

Input:

```
1
7 6 5
0 1
1 2
3 4
2 4
2 6
5 2
```

Output:

```

15
```

Strongest Connected Component-SCC

```
using namespace std;
vector<int>m[mx];
vector<int>mm[mx];
map<int,int>finish;
bool vis[mx];
int cnt1=0,cnt2=0;
void dfs(int u){
    vis[u]=true;
    ++cnt1;
    for(int i=0;i<m[u].size();i++){
        int v=m[u][i];
        if(!vis[v]) dfs(v);
    }
    finish[u]=++cnt1;
}
void printcircle(int u){
    vis[u]=true;
    for(int i=0;i<mm[u].size();i++){
        int v=mm[u][i];
        if(!vis[v]) printcircle(v);
    }
    cout<<u<<" ";
}
bool comp(const
pair<char,int>&a,const
pair<char,int>&b){
    return a.second>b.second;
}
```

```
int main(){
    int n,e;
    cin>>n>>e;
    for(int i=0;i<e;i++){
        int a,b;
        cin>>a>>b;
        m[a].push_back(b);
        mm[b].push_back(a);
    }
    memset(vis,false,sizeof(vis));
    dfs(1);
    vector<pair<int,int> >v(finish.begin(),finish.end());
    sort(v.begin(),v.end(),comp);
    memset(vis,false,sizeof(vis));
    for(int i=0;i<v.size();i++){
        if(!vis[v[i].first]){
            cnt2++;
            cout<<cnt2<<"-";
            printcircle(v[i].first);
            cout<<'\n';
        }
    }
    return 0;
}
```

Input:

```
7 8
1 2 2 3 3 1 3 4 4 5 5 6 6 7 7 5
```

Output:

```
1-2 3 1
2-4
3-6 7 5
```

T_Sort

Two way of calculating t-sort:-
First one is DFS logic:
Second one is Normal logic:

```
int main(){
//    freopen("Input.txt","r",stdin);
//    freopen("Output.txt","w",stdout);

while(scanf("%d%d",&n,&e)==2&&n!=0&&e!=0){

memset(visited,false,sizeof(visited));
        for(int i=1;i<=e;i++){
            int u,v;scanf("%d%d",&u,&v);
            edges[u].push_back(v);
        }
        for(int i=1;i<=n;i++) s.insert(i);

for(__typeof(s.begin())it=s.begin();it!=s.end();it++){
            if(!visited[*it]) dfs(*it);
        }
        while(!st.empty()){
            printf("%d ",st.top());
            st.pop();
        }
        printf("\n");
        for(int i=1;i<=n;i++) edges[i].clear();
        s.clear();
    }
    return 0;
}
```

```
using namespace std;
int n,e;
map<int,vector<int> >edges;
stack<int>st;
set<int>s;
bool visited[mx];
void dfs(int start){
    visited[start]=true;

for(__typeof(edges[start].begin())it=edges[start].begin();it!=edges[start].end();it++){
        if(!visited[*it]) dfs(*it);
    }
    st.push(start);
}
```

```
Sample Input
5 4
1 2
2 3
1 3
1 5
0 0
Sample Output
1 4 2 5 3
```

```
int main(){
//    freopen("Input.txt","r",stdin);
//    freopen("Output.txt","w",stdout);

while(scanf("%d%d",&n,&m)==2&&n!=0&&m!=0){
//        if(n==0&&m==0) break;
        for(int i=1;i<=n;i++){
            indegree[i]=0;
            taken[i]=0;
        }
        for(int i=1;i<=m;i++){
            int u,v;scanf("%d%d",&u,&v);
            order[u][v]=1;
            indegree[v]++;
        }
        T_SORT();
        memset(order,0,sizeof(order));
        indegree.clear();
        taken.clear();
    }
    return 0;
}
```

```
using namespace std;
int n,m;
int order[mx][mx];
map<int,int>indegree;
map<int,int>taken;
void T_SORT(){
    vector<int>task;
    int j;
    bool flag=true;
    for(int i=1;i<=n;i++){
        for(j=1;j<=n;j++){
            if(!taken[j] && !indegree[j]){
                taken[j]=1;
                task.push_back(j);
                for(int k=1;k<=n;k++){
                    if(!taken[k]&&order[j][k]) indegree[k]--;
                }
                break;
            }
        }
        if(j==n+1) flag=false;
    }
```

```
Sample Input
5 4
1 2
2 3
1 3
1 5
0 0
Sample Output
1 4 2 5 3
```

```
    if(flag==true){
        vector<int>::iterator it;
        for(it=task.begin();it!=task.end();it++){
            cout<<*it<<" ";
        }
        printf("\n");
        task.clear();
    }
    else{
        printf("This is no solution\n");
        task.clear();
    }
}
```

<u>Bellman Ford_sample_code:- 558 Wormholes</u>

Which works in negative cycle:- গ্রাফে নেগেটিভ সাইকেল থাকলে বেলম্যান ফোর্ড ব্যবহার করতে হবে।

```
using namespace std;
vector<pair<int,int> >Pair;
int Distance[mx],cost[mx][mx];
int n,m;
bool B_F(){
    for(int i=0;i<n;i++) Distance[i]=10000001;
    Distance[0]=0;
    for(int i=0;i<n-1;i++){
        for(int j=0;j<m;j++){
            int u=Pair[j].first,v=Pair[j].second;

if(Distance[u]+cost[u][v]<Distance[v]){

Distance[v]=Distance[u]+cost[u][v];
            }
        }
    }
    for(int i=0;i<m;i++){
        int u=Pair[i].first,v=Pair[i].second;
        if(Distance[u]+cost[u][v]<Distance[v]){
            return true;
        }
    }
    return false;
}
```

```
int main(){
    int t;
    scanf("%d",&t);
    for(int i=1;i<=t;i++){
        int s1,s2,w;
        memset(cost,0,sizeof(cost));
        scanf("%d %d",&n,&m);
        for(int j=0;j<m;j++){
            scanf("%d %d %d",&s1,&s2,&w);
            Pair.push_back(make_pair(s1,s2));
            cost[s1][s2]=w;
        }
        bool flag=B_F();
        if(flag) cout<<"possible\n";
        else cout<<"not possible\n";
        Pair.clear();
    }
    return 0;
}
```

If there any cycle in a graph

#When the graph is directed:-

```
#include<bits/stdc++.h>
using namespace std;
int n,e;
map<int,vector<int> >edges;
map<int,bool>visited;
map<int,bool>Stack;
using namespace std;
bool DFS(int u){
    if(!visited[u]){
        visited[u]=true;
        Stack[u]=true;
        for(int i=0;i<edges[u].size();i++){
            int v = edges[u][i];
            if(!visited[v]&&DFS(v)){
                return true;
            }
            else if(Stack[v]){
                return true;
            }
        }
    }
    Stack[u]=false;
    return false;
}
```

```
bool IsCyclic(){
    for(int i=0;i<n;i++){
        visited[i]=false;
        Stack[i]=false;
    }
    for(int i=0;i<n;i++){
        if(DFS(i)){
            return true;
        }
    }
    return false;
}
int main(){
    int u,v;
    cin>>n>>e;
    for(int i=0;i<e;i++){
        cin>>u>>v;
        edges[u].push_back(v);
    }
    IsCyclic()?
cout<<"Yes"<<endl:cout<<"No"<<endl;
    return 0;
}
```

#When the graph is undirected:-

```
int n,e;
map<int,vector<int> >edges;
map<int,bool>visited;
bool DFS(int vertex,int parent){
    visited[vertex]=true;
    for(int i=0;i<edges[vertex].size();i++){
        int c=edges[vertex][i];
        if(!visited[c]){
            if(DFS(c,vertex)){
                return true;
            }
        }
        else if(c!=parent)  return true;
    }
    return false;
}
```

```
bool isCyclic(){
    for(int i=0;i<n;i++){
        visited[i]=false;
    }
    for(int i=0;i<n;i++){
        if(!visited[i]){
            if(DFS(i,-1)){
                return true;
            }
        }
    }
    return false;
}
```

```
int main(){
    int u,v;
    while(cin>>n>>e){
    for(int i=0;i<e;i++){
        cin>>u>>v;
        edges[u].push_back(v);
        edges[v].push_back(u);
    }
    isCyclic()? cout<<"Yes"<<endl:cout<<"No"<<endl;
    edges.clear();
    visited.clear();
    }
    return 0;
}
```

<u>Dijkastra_Sample_Code</u>

Dijkastra_Sample_Code_Using_P_Queue

```
using namespace std;
int n;
vector<p >edges[mx];
vector<int>Distance;
int Dijkstra(int start,int End){
    Distance.assign(n,inf);
    priority_queue<p,vector<p >,greater<p > >q;
    q.push(make_pair(start,0));
    Distance[start]=0;
    while(!q.empty()){
        p u=q.top();
        q.pop();
        for(int i=0;i<edges[u.first].size();i++){
            p v = edges[u.first][i];
            int d = Distance[u.first]+v.second;
            if(d<Distance[v.first]){
                Distance[v.first] = d;
                q.push(make_pair(v.first,d));
            }
        }
    }
    return Distance[End];
}
```

```
int main(){
    int t;
    scanf("%d",&t);
    for(int tc=1;tc<=t;tc++){
        int e,start,End;
     scanf("%d %d %d
%d",&n,&e,&start,&End);
        for(int i=1;i<=e;i++){
            int u,v,w;
            scanf("%d %d
%d",&u,&v,&w);

edges[u].push_back(make_pair(v,w));

edges[v].push_back(make_pair(u,w));
        }
        int d = Dijkstra(start,End);
        if(d!=inf) printf("Case #%d:
%d\n",tc,d);
        else printf("Case #%d:
unreachable\n",tc);
        for(int i=0;i<=n;i++)
edges[i].clear();
        Distance.clear();
    }
    return 0;
}
```

Sample Input

```
4
2 1 0 1
0 1 100
3 3 2 0
0 1 100
0 2 200
1 2 50
2 0 0 1
5 5 1 5
1 2 3
2 3 4
3 5 4
1 4 10
4 5 20
```

Sample Output

```
Case #1: 100
Case #2: 150
Case #3: unreachable
Case #3: 11
```

Dijkastra_Sample_Code_Using_Class

```
using namespace std;
vector<int>edges[mx];
int cost[mx][mx];
int n;
vector<int>Distance;
class data{
public:
    int node,w;
    bool operator<(const data& p)const{
        return w>p.w;
    }
};
int Dijkstra(int start,int End){
    Distance.assign(n+1,inf);
    priority_queue<data>q;
    data u,v; u.node=start,u.w=0;
    q.push(u);
    Distance[start]=0;
    while(!q.empty()){
        u=q.top();
        q.pop();
        for(int i=0;i<edges[u.node].size();i++){
            v.node = edges[u.node][i];
            v.w = cost[u.node][v.node]+u.w;
            if(Distance[v.node]>v.w){
                Distance[v.node] = v.w;
                q.push(v);
            }
        }
    }
    return Distance[End];
}
```

```
int main(){
    int t;
    scanf("%d",&t);
    for(int tc=1;tc<=t;tc++){
        int e,start,End;
     scanf("%d %d %d
%d",&n,&e,&start,&End);
        for(int i=1;i<=e;i++){
            int u,v,w;
            scanf("%d %d
%d",&u,&v,&w);
            edges[u].push_back(v);
            edges[v].push_back(u);
            cost[u][v]=w;
            cost[v][u]=w;
        }
        int d = Dijkstra(start,End);
        if(d!=inf) printf("Case #%d:
%d\n",tc,d);
        else printf("Case #%d:
unreachable\n",tc);
        for(int i=0;i<=n;i++)
edges[i].clear();
        Distance.clear();
    }
    return 0;
}
```

This code also follow previous input and output…………….!

Dijkastra_Sample_Code_If any loop or multiple edge

```
using namespace std;
vector<p >edges[mx];
llu Distance[mx];
priority_queue<p,vector<p>,greater<p> >q;
int par[mx];
llu n;
llu Dijkstra(int start,int End){
    p u,v;
    for(int i=1;i<=n;i++) Distance[i]=inf;
    memset(par,-1,sizeof(par));
    q.push(make_pair(0,start));
    Distance[start]=0;
    while(!q.empty()){
        u=q.top();
        q.pop();
        if(u.first>Distance[u.second]) continue;
        for(int i=0;i<edges[u.second].size();i++){
            v = edges[u.second][i];
             if(Distance[u.second]+v.second<Distance[v.first]){
                Distance[v.first] = Distance[u.second]+v.second;
                q.push(make_pair(Distance[v.first],v.first));
                par[v.first]=u.second;
            }
        }
    }
    return Distance[End];
}
```

```
void printpath(int s){
    if(s==-1) return;
    printpath(par[s]);
    cout<<s <<" ";
}
int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    llu e,start,End;
    cin>>n>>e>>start>>End;
    while(e--){
        llu u,v,w;
        cin>>u>>v>>w;

edges[u].push_back(make_pair(v,w));

edges[v].push_back(make_pair(u,w));
     }
    llu d = Dijkstra(start,End);
    if(par[n]!=-1){
        cout<<"Shortest Path:
"<<d<<'\n';
        printpath(n);
        cout<<"\n";
    }
    else cout<<"-1\n";
    return 0;
}
```

Input:

```
6
7 8 1 7
1 2 2
2 3 3
2 3 4
3 4 5
4 5 7
5 6 6
4 6 15
6 7 1
```

Output:

```
24
1 2 3 4 5 6 7
```

Flood Fill Algorithm

```
#include<iostream>
using namespace std;
// Dimentions of paint screen
#define M 8
#define N 8
// A recursive function to replace previous color 'prevC' at  '(x, y)'
// and all surrounding pixels of (x, y) with new color 'newC' and
void floodFillUtil(int screen[][N], int x, int y, int prevC, int newC){
    // Base cases
    if (x < 0 || x >= M || y < 0 || y >= N) return;
    if (screen[x][y] != prevC) return;
    // Replace the color at (x, y)
    screen[x][y] = newC;
    // Recur for north, east, south and west
    floodFillUtil(screen, x+1, y, prevC, newC);
    floodFillUtil(screen, x-1, y, prevC, newC);
    floodFillUtil(screen, x, y+1, prevC, newC);
    floodFillUtil(screen, x, y-1, prevC, newC);
}
// It mainly finds the previous color on (x, y) and
// calls floodFillUtil()
void floodFill(int screen[][N], int x, int y, int newC) {
    int prevC = screen[x][y];
    floodFillUtil(screen, x, y, prevC, newC);
}
int main() {
    int screen[M][N] = {{1, 1, 1, 1, 1, 1, 1, 1},
                        {1, 1, 1, 1, 1, 1, 0, 0},
                        {1, 0, 0, 1, 1, 0, 1, 1},
                        {1, 2, 2, 2, 2, 0, 1, 0},
                        {1, 1, 1, 2, 2, 0, 1, 0},
                        {1, 1, 1, 2, 2, 2, 2, 0},
                        {1, 1, 1, 1, 1, 2, 1, 1},
                        {1, 1, 1, 1, 1, 2, 2, 1},
                      };
    int x = 4, y = 4, newC = 3;
    floodFill(screen, x, y, newC);
    cout << "Updated screen after call to floodFill: n";
    for (int i=0; i<M; i++){
        for (int j=0; j<N; j++) cout << screen[i][j] << " ";
        cout << endl;
    }
}
```

Floyd-Warshall sample code

```
#include<bits/stdc++.h>/// Solves the all-pairs shortest path problem using Floyd Warshall algorithm
#define INF 999999
int distance[101][101];
int graph[101][101];
int V,E;
void Print_Solution(){
    printf("Following matrix shows the shortest distances between every pair of vertices \n");
    for(int i=0;i<V;i++){
        for(int j=0;j<V;j++){
            if(distance[i][j]==INF) printf("%7s","INF");
            else printf ("%7d", distance[i][j]);
        }
        printf("\n");
    }
}
void Floyd_Warshall (){
    for(int i=0;i<V;i++){
        for(int j=0;j<V;j++){
            distance[i][j]=graph[i][j];
        }
    }
    for(int k=0;k<V;k++){
        for(int i=0;i<V;i++){
            for(int j=0;j<V;j++){
                if(distance[i][k]+distance[k][j]<distance[i][j]){/// If vertex k is on the shortest path from i to j,...
                    distance[i][j]=distance[i][k]+distance[k][j];///...then update the value of distance[i][j]
                }
            }
        }
    }
    Print_Solution();
}
int main(){
    scanf("%d %d",&V,&E);
    for(int i=0;i<V;i++){
        for(int j=0;j<V;j++){
            if(i==j) graph[i][j]=0;
            else graph[i][j]=INF;
        }
    }
    for(int i=1;i<=E;i++){
        int u,v,w;
        scanf("%d %d %d",&u,&v,&w);
        graph[u][v]=w;
    }
    Floyd_Warshall();
    return 0;
}
```

Lowet Common Ansector:- 10938 Flea circus

Code:

```
using namespace std;
vector<int >edges[mx];
int level[mx];
int sparse[mx][20];
int parent[mx];
void DFS(int from,int u,int dep){
    parent[u]=from;
    level[u]=dep;
    for(int i=0;i<edges[u].size();i++){
        int v=edges[u][i];
        if(v!=from){
            DFS(u,v,dep+1);
        }

    }
}
void Sparse_Table(int n){
    for(int i=1;i<=n;i++){
        sparse[i][0]=parent[i];
    }
    sparse[1][0]=-1;
    for(int j=1;(1<<j)<n;j++){
        for(int i=1;i<=n;i++){
            if(sparse[i][j-1]!=-1){
                sparse[i][j]=sparse[sparse[i][j-1]][j-1];
            }
        }
    }
}
int LCA_query(int n,int p,int q){
    if(level[p]<level[q]) swap(p,q);
    int c;
    for(c=1;(1<<c)<=level[p];c++);
    c--;
    for(int i=c;i>=0;i--){
        if(level[p]-(1<<i)>=level[q]){
            p=sparse[p][i];
        }
    }
    if(p==q) return p;
    for(int i=c;i>=0;i--){
        if(sparse[p][i]!=-1&&sparse[p][i]!=sparse[q][i]){
            p=sparse[p][i],q=sparse[q][i];
        }
    }
    return parent[p];
}
int no_node(int p,int q){
    int c;
    for(c=0;(1<<c)<=level[p];c++);
    c--;
    for(int i=c;i>=0;i--){  if(level[p]-(1<<i)>=q) p=sparse[p][i];
    }
    return p;
}
int main(){
    int n,m,a,b;
    while(scanf("%d",&n)&&n){
        mem(parent,-1);
        mem(level,0);
        mem(sparse,-1);
        for(int i=0;i<mx;i++) edges[i].clear();
        for(int i=1;i<n;i++){
            scanf("%d%d",&a,&b);
            edges[a].push_back(b);
            edges[b].push_back(a);
        }
        DFS(-1,1,0);
        Sparse_Table(n);
        scanf("%d",&m);
        while(m--){
            scanf("%d%d",&a,&b);
            int lca=LCA_query(n,a,b);
            int dis=level[a]+level[b]-2*level[lca];
            if(dis%2==0){
                dis=dis>>1;
                if(level[a]-level[lca]>=dis){
                    int ans=no_node(a,level[a]-dis);///Tricky part
                    printf("The fleas meet at %d.\n",ans);
                }
                else{
                    int dd=level[a]-level[lca];
                    dd=dis-dd;
                    dd=dd+level[lca];
                    int ans=no_node(b,dd);///Tricky part
                    printf("The fleas meet at %d.\n",ans);
                }
            }
            else{
                dis=dis>>1;
                vector<int>point;
                if(level[a]-level[lca]>=dis){
                    int ans=no_node(a,level[a]-dis);///Tricky part
                    point.push_back(ans);
                }
                else{
                    int dd=level[a]-level[lca];
                    dd=dis-dd;
                    dd=dd+level[lca];
                    int ans=no_node(b,dd);///Tricky part
                    point.push_back(ans);
                }
                if(level[a]-level[lca]>=dis+1){
                    int ans=no_node(a,level[a]-dis-1);///Tricky part
                    point.push_back(ans);
                }
                else{
                    int dd=level[a]-level[lca];
                    dd=dis+1-dd;
                    dd=dd+level[lca];
                    int ans=no_node(b,dd);///Tricky part
                    point.push_back(ans);
                }
                sort(point.begin(),point.end());
                printf("The fleas jump forever between %d and %d.\n",point[0],point[1]);
                point.clear();
            }
        }
    }
    return 0;
}
```

MST&DisjointSET

First best MST:-

```
using namespace std;
class edge{
public:
    int w;
    char u,v;
    bool operator<(const edge &p)const{
        return w<p.w;
    }
};
vector<edge>e;
map<char,int>representative;
int find_par(int x){
    if(representative[x]!=x)
representative[x]=find_par(representative[x]);
    return representative[x];
}
int MST(int n){
    sort(e.begin(),e.end());
    for(char i='a';i<=97+n;i++){
        representative[i]=i;
    }
    int c=0,ans=0;
    for(int i=0;i<(int)e.size();i++){
        int pu=find_par(e[i].u);
        int pv=find_par(e[i].v);
        if(pu!=pv){
            representative[pu]=pv;
            c++;
            ans+=e[i].w;
            if(c==n-1){
                break;
            }
        }
    }
    return ans;
}
```

```
int main(){
//    freopen("Input.txt","r",stdin);
//    freopen("Output.txt","w",stdout);
    int n,m;
    cin>>n>>m;
    for(int i=0;i<m;i++){
        int w;
        char u,v;
        cin>>u>>v>>w;
        edge ob;
        ob.u=u;
        ob.v=v;
        ob.w=w;
        e.push_back(ob);
    }
    cout<<MST(n)<<endl;
    return 0;
}
```

Second best MST:- 10600 ACM contest and Blackout

```
using namespace std;
int ans1;
vector<int>ans2;
map<int,int>parent;
vector<pair<int,int> >ms;
class node{
public:
    int u,v,w;
    bool operator<(const node &p)const{
        return w<p.w;
    }
};
vector<node>edges;
int find_par(int x){
    if(parent[x]!=x)
parent[x]=find_par(parent[x]);
    else return parent[x];
}
void MST(int n){
    sort(edges.begin(),edges.end());
    for(int i=0;i<=n;i++) parent[i]=i;
    int c=0;
    for(int i=0;i<edges.size();i++){
        int pu=find_par(edges[i].u);
        int pv=find_par(edges[i].v);
        if(pu!=pv){
            c++;
            parent[pu]=pv;
            ans1+=edges[i].w;

ms.push_back(make_pair(edges[i].u,edges[i].v));
            if(c==n-1) break;
        }
    }
}
```

```
int second_mst(int n,int a,int b){
    int ans=0,c=0;
    for(int i=0;i<=n;i++) parent[i]=i;
    for(int i=0;i<edges.size();i++){
        if(edges[i].u==a&&edges[i].v==b)
continue;
        else{
            int pu=find_par(edges[i].u);
            int pv=find_par(edges[i].v);
            if(pu!=pv){
                c++;
                ans+=edges[i].w;
                parent[pu]=pv;
                if(c==n-1){
                    ans2.push_back(ans);
                    break;
                }
            }
        }
    }
}
int main(){
    int t;  scanf("%d",&t);
    while(t--){
        ans1=0;
        ans2.clear(); ms.clear(); edges.clear();
        parent.clear(); int n,m;
        scanf("%d %d",&n,&m);
        for(int i=0;i<m;i++){
            int a,b,d;
            scanf("%d %d %d",&a,&b,&d);
            node ob;  ob.u=a; ob.v=b; ob.w=d;
            edges.push_back(ob);
        }
        MST(n);
        for(int i=0;i<ms.size();i++){
         second_mst(n,ms[i].first,ms[i].second);
        }
        sort(ans2.begin(),ans2.end());
        printf("%d %d\n",ans1,ans2[0]);
    }
```

	return 0;
	}

Sample Input
```
2
5 8
1 3 75      3 4 51      2 4 19      3 2 95      2 5 42      5 4 31      1 2 9       3 5 66
9 14
1 2 4       1 8 8       2 8 11      3 2 8       8 9 7       8 7 1       7 9 6       9 3 2
3 4 7       3 6 4       7 6 2       4 6 14      4 5 9       5 6 10
```
Sample Output
```
110 121
37 37
```

Tower of Hanoi

The main logic behind the tower of Hanoi is, from any pile a with n disks, in order to move all the n disks to pile c following all constraints, we'll first need to move n-1 disks to the middle pile b, then the rest nth disk directly to destination, then move the previously moved n-1 disks from middle pile b to destination pile c using pile a as middle pile now. Similarly, for n-1 disks, we'll work for n-2 disks first, then work up to n... and so on. The following program demonstrates it all:

```
#include <stdio.h>

// s source, m middle, d destination
void tower(int n, char s, char m, char d)
{
    if(n>0)
    {
        // first move n-1 disks from source to middle,
        // using destination as middle pillar
        tower(n-1, s, d, m);
        // the only left nth disk can be moved directly from source to
destination
        printf("%c -> %c\n", s, d);
        // on first step we've sent n-1 disks to the middle,
        // and now, from the middle pillar, we'll transfer them back to
destination
        // using the source as the middle pillar
        tower(n-1, m, s, d);
    }
}

int main()
{
    int n;
    scanf("%d", &n);
    tower(n, 'a', 'b', 'c');
    return 0;
}
```

ট্রি ডায়ামিটার

ট্রি হলো এমন একটা আনডিরেক্টেড গ্রাফ যেটার সব নোড থেকে সব নোডে যাওয়া যায় এবং কোনো সাইকেল নেই। এখন আমাদের ট্রি এর সবথেকে দূরের দুটা নোড খুজে বের করতে হবে, একেই বলা হয় ট্রি এর ডায়ামিটার।

Sample Code:- **POJ 1383 Labyrinth**

```
using namespace std;
const int fx[]={+1,-1,+0,+0};
const int fy[]={+0,+0,+1,-1};
int r,c,ans,x,y;
char arr[mx][mx];
bool visited[mx][mx];
int dis[mx][mx];
void BFS(){
    queue<pair<int,int> >q;
    q.push(make_pair(x,y));
    memset(visited,false,sizeof(visited));
    memset(dis,0,sizeof(dis));
    visited[x][y]=true;
    while(!q.empty()){
        pair<int,int>p;
        p=q.front();
        q.pop();
        for(int i=0;i<4;i++){
            int tx=p.first+fx[i];
            int ty=p.second+fy[i];
           if(arr[tx][ty]=='.'&&valid(tx,ty)&&!visited[tx][ty]){
                visited[tx][ty]=true;
                q.push(make_pair(tx,ty));
                dis[tx][ty]=dis[p.first][p.second]+1;
                if(dis[tx][ty]>ans){
                    ans=dis[tx][ty];
                    x=tx;y=ty;
                }
            }
        }
    }
}
```

```
int main(){
    int t;
    scanf("%d",&t);
    for(int i=1;i<=t;i++){
        scanf("%d %d",&c,&r);
        for(int j=0;j<r;j++){
            scanf("%s",&arr[j]);
        }
        x=y=-1;
        for(int j=0;j<r;j++){
            for(int k=0;k<c;k++){
                if(arr[j][k]=='.'){
                    x=j,y=k;
                    break;
                }
            }
            if(x!=-1) break;
        }
        ans=0;
        BFS();
        ans=0;
        BFS();
 printf("Maximum rope length is %d.\n",ans);
    }
}
```

```
Input:
2
3 3
###
#.#
###
7 6
#######
#.#.###
```

Output:

```
Maximum rope length is 0.
Maximum rope length is 8.
```

```
#.#.###
#.#.#.#
#.....#
#######
```

Farthest Nodes in a Tree:- *LightOJ - 1094*

Given a tree (a connected graph with no cycles), you have to find the farthest nodes in the tree. The edges of the tree are weighted and undirected. That means you have to find two nodes in the tree whose distance is maximum amongst all nodes.

Solution:

```
int main(){
    int t;
    scanf("%d",&t);
    for(int tc=1;tc<=t;tc++){
        int n;
        scanf("%d",&n);
        for(int i=0;i<n-1;i++){
            int u,v,w;
            scanf("%d %d %d",&u,&v,&w);
            edges[u].push_back(make_pair(v,w));
            edges[v].push_back(make_pair(u,w));
        }
        ans=0;
        memset(visited,false,sizeof(visited));
        DFS(0,0);
        ans=0;
        memset(visited,false,sizeof(visited));
        DFS(st,0);
        printf("Case %d: %d\n",tc,ans);
        for(int i=0;i<n;i++){
            edges[i].clear();
        }
    }
    return 0;
}
```

```
using namespace std;
vector<pair<int,int> >edges[mx];
int st;
long long int ans;
bool visited[mx];
void DFS(int u,long long int dis){
    visited[u]=true;
    if(dis>ans){
        ans=dis;
        st=u;
    }
    for(int i=0;i<edges[u].size();i++){
        int v=edges[u][i].first;
        int w=edges[u][i].second;
        if(!visited[v]) DFS(v,dis+w);
    }
}
```

Sample Input

```
2
4
0 1 20
1 2 30
2 3 50
5
0 2 20
2 1 10
0 3 29
0 4 50
```

Output for Sample Input

```
Case 1: 100
Case 2: 80
```

Marge Sort

```
using namespace std;
void merge(int arr[],int l,int m,int r){
    int i,j,k;
    int n1=m-l+1;
    int n2=r-m;
    int L[n1],R[n2];
    for(i=0;i<n1;i++){
        L[i]=arr[l+i];
    }
    for(j=0;j<n2;j++){
        R[j]=arr[m+1+j];
    }
    i=0;j=0;k=l;
    while(i<n1&&j<n2){
        if(L[i]<=R[j]){
            arr[k]=L[i];
            i++;
        }
        else{
            arr[k]=R[j];
            j++;
        }
        k++;
    }
    while(i<n1){
        arr[k]=L[i];
        i++;
        k++;
    }
    while(j<n2){
        arr[k]=R[j];
        j++;
        k++;
    }
}
```

```
void mergeSort(int arr[],int l,int r){
    if(l<r){
        int m=l+(r-l)/2;
        mergeSort(arr,l,m);
        mergeSort(arr,m+1,r);
        merge(arr,l,m,r);
    }
}
void print_array(int arr[],int n){
    for(int i=0;i<n;i++){
        cout<<arr[i]<<" ";
    }
    cout<<endl;
}
int main(){
    int n;
    cin>>n;
    int arr[100];
    for(int i=0;i<n;i++){
        cin>>arr[i];
    }
    mergeSort(arr,0,n-1);
    print_array(arr,n);
    return 0;
}
```

Segment Tree and Lazy Propagation

# *Segment Tree_Basic Code:*

```
using namespace std;
int arr[mx];
int tree[3*mx];
void input(int node,int l,int r){
    if(l>r) return;
    if(l==r){
        tree[node]=arr[l];
        return;
    }
    int left=node<<1,right=(node<<1)+1,
         mid=(l+r)>>1;
    input(left,l,mid);
    input(right,mid+1,r);
    tree[node]=tree[left]+tree[right];
}
int query(int node,int l,int r,int i,int j){
    if(i>r||j<l) return 0;
    if(i<=l&&j>=r) return tree[node];
    int
left=node<<1,right=(node<<1)+1,mid=(l+r)>>1;
    int p1 = query(left,l,mid,i,j);
    int p2 = query(right,mid+1,r,i,j);
    return p1+p2;
}
```

```
void update(int node,int l,int r,int i,int
newvalue){
    if(i>r||i<l) return;
    if(i<=l&&i>=r){///when l==r==i !!!!!!!!!!
        tree[node] = newvalue;
        return;
    }
    int
left=node<<1,right=(node<<1)+1,mid=(l+r)>>1;
    update(left,l,mid,i,newvalue);
    update(right,mid+1,r,i,newvalue);
    tree[node] = tree[left]+tree[right];
}
int main(){
    int n;  cin>>n;
    for(int i=1;i<=n;i++)  cin>>arr[i];
    input(1,1,n);
    update(1,1,n,2,0);
    cout<<query(1,1,n,1,3)<<endl;
    update(1,1,n,2,2);
    cout<<query(1,1,n,1,3)<<endl;
    return 0;
}
```

*Find the maximum consecutive similar number from **i** to **j**:- 1339 - Strongest Community*

Solution:

```
using namespace std;
int ans;
int arr[mx],value[4*mx];
p Left[4*mx],Right[4*mx];
void build_tree(int node,int l,int r){
    if(l>r) return;
    if(l==r){
        value[node]=1;
        Left[node]=Right[node]=p(arr[l],1);
        return;
//        left[node].first=right[node].first=arr[l];
//        left[node].second=right[node].second=1;
    }
    int ll=node<<1,rr=(node<<1)+1,mid=(l+r)>>1;
    build_tree(ll,l,mid);
    build_tree(rr,mid+1,r);
    value[node]=max(value[ll],value[rr]);
    Left[node]=Left[ll]; Right[node]=Right[rr];
    if(Left[rr].first==Right[ll].first)
        value[node]=max(value[node],Left[rr].second+Right[ll].second);
    if(Left[ll].first==Left[rr].first) Left[node].second+=Left[rr].second;
    if(Right[ll].first==Right[rr].first) Right[node].second+=Right[ll].second;
}
pair<p,p>query(int node,int l,int r,int i,int j){
    if(l>j||r<i) return make_pair(p(0,0),p(0,0));
    if(l>=i&&r<=j){
        ans=max(ans,value[node]);
        return make_pair(Left[node],Right[node]);
    }
    if(l==r) return make_pair(p(0,0),p(0,0));
    int ll=node<<1,rr=(node<<1)+1,mid=(l+r)>>1;
    pair<p,p>lll=query(ll,l,mid,i,j);
    pair<p,p>rrr=query(rr,mid+1,r,i,j);
    p le1=lll.first,ri1=lll.second;
    p le2=rrr.first,ri2=rrr.second;
    if(ri1.first==le2.first) ans=max(ans,ri1.second+le2.second);
    if(le1.first==le2.first) le1.second+=le2.second;
    if(ri1.first==ri2.first) ri2.second+=ri1.second;
    return make_pair(le1,ri2);
}
int main(){
    int t; scanf("%d",&t);
    for(int tc=1;tc<=t;tc++){
        printf("Case %d:\n",tc);
        int n,c,q;
        scanf("%d%d%d",&n,&c,&q);
        for(int i=1;i<=n;i++)
              scanf("%d",&arr[i]);
        build_tree(1,1,n);
        for(int i=1;i<=q;i++){
            int x,y;
            scanf("%d%d",&x,&y);
            ans=1;
            query(1,1,n,x,y);
            printf("%d\n",ans);
        }
    }
    return 0;
```

Input:

```
2
10 3 4
1 1 1 3 3 3 3 2 2 2
1 5
1 6
1 7
7 9
3 3 1
3 2 1
1 1
```

Output:

```
Case 1:
3
3
4
2
Case 2:
1
```

```
}
```

# *Segment Tree with Lazy Propagation_Basic Code:-*

```
using namespace std;
long long arr[mx];
long long tree[4*mx];
long long lazy[4*mx];
void create_tree(int node,int l,int r){
    if(l>r){
        return;
    }
    if(l==r){
        tree[node] = arr[l];
        return;
    }
    int left = node*2;
    int right = node*2+1;
    int mid = (l+r)/2;
    create_tree(left,l,mid);
    create_tree(right,mid+1,r);
    tree[node] = tree[left]+tree[right];
}
void update(int node,int l,int r,int i,int j,int val){
    int left = node*2;
    int right = node*2+1;
    int mid = (l+r)/2;
    if(lazy[node]!=0){
        tree[node]+= lazy[node]*(r-l+1);
        if(l!=r){
            lazy[left]+=lazy[node];
            lazy[right]+=lazy[node];
        }
        lazy[node] = 0;
    }
    if(i>r||j<l){
        return;
    }
    if(i<=l&&j>=r){
        tree[node]+=(r-l+1)*val;
        if(l!=r){
            lazy[left]+=val;
            lazy[right]+=val;
        }
        return;
    }
    update(left,l,mid,i,j,val);
    update(right,mid+1,r,i,j,val);
    tree[node] = tree[left]+tree[right];
}
```

```
long long query(int node,int l,int r,int i,int j){
    if(l>r||i>r||j<l){
        return 0;
    }
    int left = node*2;
    int right = node*2+1;
    int mid = (l+r)/2;
    if(lazy[node]!=0){
        tree[node]+= lazy[node]*(r-l+1);
        if(l!=r){
            lazy[left]+=lazy[node];
            lazy[right]+=lazy[node];
        }
        lazy[node] = 0;
    }
    if(i<=l&&j>=r){
        return tree[node];
    }
    long long p1=query(left,l,mid,i,j);
    long long p2=query(right,mid+1,r,i,j);
    return p1+p2;
}
int main(){
    int t,n,q,a,b,c,val;
    scanf("%d",&t);
    for(int i=1;i<=t;i++){
        printf("Case %d:\n",i);
        scanf("%d %d",&n,&q);
        memset(arr,0,sizeof(arr));
        memset(tree,0,sizeof(tree));
        memset(lazy,0,sizeof(lazy));
        create_tree(1,0,n-1);
        for(int j=1;j<=q;j++){
            scanf("%d",&a);
            if(a==1){
                scanf("%d %d",&b,&c);
                printf("%lld\n",query(1,0,n-1,b,c));
            }
            else{
                scanf("%d %d %d",&b,&c,&val);
                update(1,0,n-1,b,c,val);
            }
        }
    }
    return 0;
}
```

Binary Indexed Tree or Fenwick Tree

Basic Code

```
using namespace std;
vector<int>elements;
vector<int>BITree;
int N;
int getSum(int index){
    int sum=0;
    index=index+1;
    while(index>0){
        sum+=BITree[index];
        index-=index&(-index);
    }
    return sum;
}
int rangeQuery(int u,int v){
    return getSum(v)-getSum(u-1);
}
void updateBIT(int index,int value){
    index=index+1;
    while(index<=N){
        BITree[index]+=value;
        index+=index&(-index);
    }
}
void constructBITree(){
    for(int i=1;i<=N;i++){
        BITree.push_back(0);
    }
    for(int i=0;i<N;i++){
        updateBIT(i,elements[i]);
    }
}
int main(){
    scanf("%d",&N);
    for(int i=1;i<=N;i++){
        int k; scanf("%d",&k);
        elements.push_back(k);///2 1 1 3 2 3 5 6 7 8 9
    }
    constructBITree();
    printf("Sum of array....is %d\n",getSum(5));
    updateBIT(3,6);
    printf("Sum of array....is %d\n",getSum(5));
    printf("Sum of array....is %d\n",rangeQuery(0,6));
    return 0;
}
```

1112 - Curious Robin Hood

```
#define ui unsigned int
using namespace std;
int elements[100001];
int BITree[100001];
int n;
int getSum(ui index){
    int sum=0;
    index=index+1;
    while(index>0){
        sum+=BITree[index];
        index-=index&(-index);
    }
    return sum;
}
int rangeQuery(ui u,ui v){
    return getSum(v)-getSum(u-1);
}
void updateBIT(ui index,int value){
    index=index+1;
    while(index<=n){
        BITree[index]+=value;
        index+=index&(-index);
    }
}
void updateBIT_decress(ui index,int value){
    index=index+1;
    while(index<=n){
        BITree[index]-=value;
        index+=index&(-index);
    }
}
void constructBITree(){
    for(int i=1;i<=n;i++){
        BITree[i]=0;
    }
    for(int i=0;i<n;i++){
        updateBIT(i,elements[i]);
    }
}
int main(){
    int t; scanf("%d",&t);
    for(int i=1;i<=t;i++){
        memset(elements,0,sizeof(elements));
        memset(BITree,0,sizeof(BITree));
        printf("Case %d:\n",i);
        int q; scanf("%d %d",&n,&q);
        for(int i=0;i<n;i++){
            int k;  scanf("%d",&k);
            elements[i]=k;
```

Input and Output of *1112 - Curious Robin Hood*

Input:

```
1
5 6
3 2 1 4 5
1 4
2 3 4
3 0 3
1 2
3 0 4
1 1
```

Output:

```
Case 1:
5
14
1
13
2
```

```
        }
        constructBITree();
        while(q--){
            int a;
            scanf("%d",&a);
            if(a==1){
                ui h;
                scanf("%d",&h);
                updateBIT_decress(h,elements[h]);
                printf("%d\n",elements[h]);
                elements[h]=0;
            }
            else if(a==2){
                ui h,k;
                scanf("%d %d",&h,&k);
                updateBIT(h,k);
                elements[h]=elements[h]+k;
            }
            else if(a==3){
                ui h,k;
                scanf("%d %d",&h,&k);
                printf("%d\n",rangeQuery(h,k));
            }
        }
    }
    return 0;
}
```

Suffix or Prefix tree

#This Prefix tree works as a dictionary:

```
using namespace std;
class node{
public:
    bool endmark;
    node *next[26+1];
    node(){
        endmark = false;
        for(int i=0;i<26;i++){
            next[i] = NULL;
        }
    }
};
node *root;
void Insert(char *str,int len){
    node *current_node = root;
    for(int i=0;i<len;i++){
        int id = str[i]-'a';
        if(current_node->next[id]==NULL){
            current_node->next[id]=new node();
```

```
void del(node *current_node){
    for(int i=0;i<26;i++){
        if(current_node->next[i]){
            del(current_node->next[i]);
        }
    }
    delete(current_node);
}
int main(){
    root = new node();
    cout<<"Enter Number of Words:"<<endl;
    int n; cin>>n;
    for(int i=1;i<=n;i++){
        char str[50];
        scanf("%s",str);
        Insert(str,strlen(str));
    }
    cout<<"Number of Query:"<<endl;
    int q; cin>>q;
```

```
        }
        current_node = current_node->next[id];
    }
    current_node->endmark = true;
}
bool Search(char *str,int len){
    node *current_node = root;
    for(int i=0;i<len;i++){
        int id = str[i]-'a';
        if(current_node->next[id]==NULL){
            return false;
        }
        current_node=current_node->next[id];
    }
    return current_node->endmark;
}
```

```
    for(int i=1;i<=q;i++){
        char str[50];
        scanf("%s",str);
        if(Search(str,strlen(str))){
            cout<<"found"<<endl;
        }
        else{
            cout<<"Not Found"<<endl;
        }
    }
    del(root);
    return 0;
}
```

#Some Other Example: *Short names are good-UVA 12506(UVA 11488 – Hints)*

```
using namespace std;
ll MPG;
class node{
public:
    ll endmark;
    node *next[30];
    node(){
        endmark = 0;
        for(int i=0;i<30;i++){
            next[i]=NULL;
        }
    }
};
node *root;
void insert(string s){
    node *current_node=root;
    for(int i=0;i<s.length();i++){
        int id = s[i]-'a';
        if(current_node->next[id]==NULL){
            current_node->next[id]=new node();
        }
        current_node=current_node->next[id];
        current_node->endmark=current_node-
>endmark+1;
//        MPG=max(MPG,current_node-
>endmark*(i+1));
    }
}
void del(node *current_node){
    for(int i=0;i<26;i++){ if(current_node->next[i])
            delete(current_node->next[i]);
    }
    delete(current_node);
}
```

```
void dfs(node *current_node,int endmark){
    if(current_node==NULL) return;
    if(current_node->endmark==1){
        MPG+=endmark;
        return;
    }
    for(int i=0;i<26;i++){
        if(current_node->next[i]!=NULL){
            dfs(current_node->next[i],endmark+1);
        }
    }
}
int main(){
    int T,N;
    cin>>T;
    while(T--){
        MPG=0;
        string s;
        root = new node();
        cin>>N;
        for(int i=0;i<N;i++){
            cin>>s;
            insert(s);
        }
        node *current_node=root;
        dfs(current_node,0);
        cout<<MPG<<endl;
        del(root);
    }
    return 0;
}
```

Articulation Points and Bridges_Sample Code

```
using namespace std;
vector<int>edges[mx];
int parent[mx],child[mx],low[mx],temp[mx];
bool vis[mx],is_cut[mx];
int c;
void dfs(int u){
    vis[u]=true;
    low[u]=temp[u]=++c;
//    cout<<low[u]<<endl;
    for(int i=0;i<edges[u].size();i++){
        int v=edges[u][i];
        if(v==parent[u]) continue;
        else if(!vis[v]){
            child[u]++;
            parent[v]=u;
            dfs(v);
            low[u]=min(low[u],low[v]);
//            if(parent[u]==-1&&child[u]>1)
is_cut[u]=true;
            if(low[v]>=temp[u]) is_cut[u]=true;
        }
        else{
            low[u]=min(low[u],temp[v]);
        }
    }
}
int main(){
    int n,e;
    scanf("%d %d",&n,&e);
    for(int i=1;i<=e;i++){
        int x,y;scanf("%d%d",&x,&y);
        edges[x].push_back(y);
        edges[y].push_back(x);
    }
    c=0; parent[1]=-1;
    dfs(1);
//    if(child[1]>1) is_cut[1]=true;
//    else is_cut[1]=false;
    is_cut[1]=child[1]>1?true:false;
    for(int i=1;i<=n;i++){
        if(is_cut[i]) printf("%d\n",i);
    }
    return 0;
}
```

Minimum Vertex Cover Sample Code

```
vector<int>edge[mx];
int dp[mx][mx];
int par[mx];
int func(int u,int isgurd){
    if(edge[u].size()==0) return 0;
    int &ret=dp[u][isgurd];
    if(ret!=-1) return ret;
    int sum=0;
//    cout<<"BIJOY"<<endl;

    for(int i=0;i<edge[u].size();i++){
        int v=edge[u][i];
        if(v!=par[u]){
            par[v]=u;
            if(isgurd==0){
                sum+=func(v,1);
            }
            else{
                sum+=min(func(v,0),func(v,1));

            }
        }
    }
    return ret=sum+isgurd;
}
int main(){
    int n,e;
    while(scanf("%d%d",&n,&e)==2){
        if(n==0) break;
        for(int i=1;i<mx;i++) edge[i].clear();
        mem(dp,-1);
        mem(par,0);
        for(int i=1;i<=e;i++){
            int u,v; scanf("%d%d",&u,&v);
            edge[u].push_back(v);
            edge[v].push_back(u);
        }
        int ans=min(func(1,0),func(1,1));
        printf("%d\n",ans);
    }
    return 0;
}
```

Max Flow - Ford Fulkerson Sample Code

```
using namespace std;
int capacity[mx][mx],par[mx];
bool vis[mx];
vector<int>edges[mx];
//vector<int>path;
//vector<vector<int> >paths;
int bfs(int n,int start,int finish){
   bool flag=false;
   mem(vis,false);
   mem(par,-1);
   queue<int>q;
   q.push(start);
   vis[start]=true;
   while(!q.empty()){
      int u=q.front(); q.pop();
      for(int i=0;i<edges[u].size();i++){
         int v=edges[u][i];
         if(!vis[v]&&capacity[u][v]>0){
            vis[v]=true;
            par[v]=u;
            q.push(v);
            if(v==finish){
               flag=true;
               break;
            }
         }
      }
   }
   return flag;
}
int FordFulkarson(int n,int start,int finish){
//   path.clear();
//   paths.clear();
   int max_flow=0;
   while(bfs(n,start,finish)){
      int v=finish,flow=1e9;
//       cout<<v<<" ";
      while(v!=start){
   //       path.push_back(v);
         int u = par[v];
//           cout<<v<<" -> "<<u<<" ";
         flow=min(flow,capacity[u][v]);
```

Max Flow -Dinic Sample Code

```
using namespace std;
ll n,m;
ll level[mx],car[mx];
class data{
public:
   ll from,to,cap,flow;
};
vector<data>edge;
vector<ll>graph[mx];
bool bfs(){
   mem(level,-1);
   queue<ll>q;
   q.push(1);
   level[1]=0;
   while(!q.empty()){
      if(level[n]!=-1) return true;
      ll u=q.front(); q.pop();
      for(ll i=0;i<(ll)graph[u].size();i++){
         ll id=graph[u][i];
         ll v=edge[id].to;
         if(level[v]==-
1&&edge[id].flow<edge[id].cap){
            q.push(v);
            level[v]=level[u]+1;
         }
      }
   }
   return false;
}
ll dfs(ll u,ll flow){
   if(u==n) return flow;
   if(flow==0) return 0;
   for(;car[u]<graph[u].size();car[u]++){
      ll id=graph[u][car[u]];
      ll v=edge[id].to;

if(level[v]==level[u]+1&&edge[id].flow<edge[i
d].cap){
         ll cur_flow=min(flow,(ll)(edge[id].cap-
edge[id].flow));
```

```
            v=u;
        }
//        cout<<endl;
//        cout<<"BIJOY "<<flow<<endl;
    //    path.push_back(start);
        max_flow+=flow;
        v=finish;
        while(v!=start){
            int u = par[v];
            capacity[u][v]-=flow;
//            cout<<u<<" "<<v<<" 
"<<capacity[u][v]<<endl;
            capacity[v][u]+=flow;
//            cout<<v<<" "<<u<<" 
"<<capacity[v][u]<<endl;
            v=u;
        }
//        reverse(path.begin(),path.end());
//        paths.push_back(path);
//        path.clear();
    }
    return max_flow;
}
int main(){
    int t; scanf("%d",&t);
    for(int cs=1;cs<=t;cs++){
        for(int i=0;i<mx;i++) edges[i].clear();
        mem(capacity,0);
        int n; scanf("%d",&n);
        int start=1,finish=n,e;
        scanf("%d",&e);
        for(int i=1;i<=e;i++){
            int u,v,w;
            scanf("%d%d%d",&u,&v,&w);
            edges[u].push_back(v);
            edges[v].push_back(u);
            capacity[u][v]+=w;
            capacity[v][u]+=w;
        }
        int ans = FordFulkarson(n,start,finish);
        printf("Case %d: %d\n",cs,ans);
    }
    return 0;
}
```

```
            ll temp_flow=dfs(v,cur_flow);
            if(temp_flow){
                edge[id].flow+=temp_flow;
                edge[id^1].flow-=temp_flow;
                return temp_flow;
            }
        }
    }
    return 0;
}
ll dinic(){
    ll max_flow=0;
    while(bfs()){
        mem(car,0);
        while(ll temp_flow=dfs(1,INT_MAX)){
            max_flow+=temp_flow;
            if(temp_flow==0) break;
        }
    }
    return max_flow;
}
int main(){
    scanf("%lld%lld",&n,&m);
    for(ll i=1;i<=m;i++){
        ll u,v,w;
        scanf("%lld%lld%lld",&u,&v,&w);
        if(u!=v){
            data ob1={u,v,w,0};
            data ob2={v,u,w,0};
            graph[u].push_back((ll)edge.size());
            edge.push_back(ob1);
            graph[v].push_back((ll)edge.size());
            edge.push_back(ob2);
        }
    }
    ll ans=dinic();
    printf("%lld\n",ans);
    return 0;
}
```