```cpp
/** GRAPH THEORY **/
/** GT.01 - Dijkstra Template **/
struct node{
    int u,w;
    node(int a,int b){
        u=a;    w=b;
    }
};
bool operator < (node A,node B){
    return A.w>B.w;
}
void dijkstra(int s){
    for(int i=0;i<sz;i++){
        d[i]=intlimit;  par[i]=-1;
    }
    priority_queue<node>pq;
    pq.push(node(s,0)); d[s]=0;
    while(!pq.empty()){
        node top=pq.top(); pq.pop();
        int u=top.u;
        if(top.w > d[u])    continue;
        for(int i=0;i<graph[u].size();i++){
            int v=graph[u][i];
            if(top.w+cost[u][i] < d[v]){
                d[v]=top.w+cost[u][i];
                pq.push(node(v,d[v]));
                par[v]=u;
            }   }   }   }
/** GT.02 - 0/1 BFS **/
void BFS(int start)
{
    deque<int>q;
    q.push_back(start); dis[start] = 0;
    while(!q.empty()){
        int u = q.front();  q.pop_front();
        for(int i=0 ;
i<(int)graph[u].size() ; i++){
            int v = graph[u][i];
            if(dis[v] > dis[u]+cost[u][i]){
                dis[v] = dis[u]+cost[u][i];
                (cost[u][i] == 0)?
q.push_front(v) : q.push_back(v);
            }
        }
    }
}
/** GT.03 - TopSort **/
struct node{
    int x;
    node(int a){    x=a;    }
};
bool operator < (node A,node B){
    return A.x>B.x;
}
vector<int>res,graph[10005];
int in[10005];
void topBFS(int n){
    priority_queue<node>q;
    for(int i=1;i<=n;i++){
        if(in[i] == 0)  q.push(node(i));
    }
    while(!q.empty()){
        int u=q.top().x;    res.pb(u);
q.pop();
        for(int i=0;i<graph[u].size();i++){
            in[graph[u][i]]--;
            if(in[graph[u][i]] == 0)
                q.push(node(graph[u][i]));
        }
    }
}
/** GT.04 - TopSort **/
vector<int>graph[1005];
bool vis[1005];     stack<int>ans;
void topDFS(int start){
    vis[start]=1;
    for(int i=0 ; i<graph[start].size() ;
i++){
        int u=graph[start][i];
        if(!vis[u]) topDFS(u);
    }
    ans.push(start);
}
/** GT.05 - Minimum Spanning Tree **/
#define sz 10005
struct edge{
    int u,v,cost;
    bool operator < (const edge& p)const
        return cost < p.cost;
};
vector<edge>e;  int par[sz];
void makeset(int n){
    for(int i=1;i<=n;i++)
        par[i]=i;
}
int Find(int x){
    if(par[x] == x)
        return x;
    return par[x]=Find(par[x]);
}
int MST(int n){
    sort(e.begin(),e.end());
    makeset(n);
    int cnt=0,s=0;
    for(int i=0 ; i<e.size() ; i++){
        int u=Find(e[i].u);
        int v=Find(e[i].v);
        if(u != v){
            par[u]=v;   cnt++;
            s += e[i].cost;
            if(cnt == n-1)  break;
        }
    }
    if(cnt == n-1)  return s;   /** All
the nodes are connected **/
    else    return -1;  /** There are
twice or more component **/
}
/** GT.06 - SCC Template **/
vi graph[1005],rgraph[1005],comp[1005];
bool vis[1005]; stack<int>stk;
void DFS1(int s){
    vis[s]=1;
    for(int i=0;i<graph[s].size();i++){
        int u=graph[s][i];
        if(!vis[u]) DFS1(u);
    }
    stk.push(s);    return;
}
void DFS2(int s,int x){
    vis[s]=x;
    comp[x].pb(s);
    for(int i=0;i<rgraph[s].size();i++){
        int u=rgraph[s][i];
        if(!vis[u]) DFS2(u,x);
    }   return;
}
int main()
{
    int n,e,i,j,k,u,v,x;
    while(scin2(n,e) != EOF){
        for(i=0;i<1005;i++){
            graph[i].clear();
rgraph[i].clear();
            comp[i].clear();    vis[i]=0;
        }
        for(i=1;i<=e;i++){
            scin2(u,v);
            graph[u].pb(v);
            rgraph[v].pb(u);
        }
        for(i=1;i<=n;i++){
            if(!vis[i]) DFS1(i);
        }
        x=0;    ms(vis,0);
        while(!stk.empty()){
            int top=stk.top();  stk.pop();
            if(!vis[top])   DFS2(top,++x);
        }
        for(i=1;i<=x;i++){
            pf("%d->",i);
            for(j=0;j<comp[i].size();j++){
                pf(" %d",comp[i][j]);
            }   pf("\n");
        }
    }
    return 0;
}
```

```cpp
/** GT.07 - Articulation Point **/
#define node 10005
vector<int>graph[node];
int tim=0,par[node],disc[node],low[node];
bool vis[node],AP[node];
void DFSAP(int u){
    vis[u]=true;
    low[u]=disc[u]=++tim;
    int child=0;
    for(int i=0;i<graph[u].size();i++){
        int v=graph[u][i];
        if(vis[v] == false){
            child++;    par[v]=u;
DFSAP(v);
            low[u]=min(low[u],low[v]);
            if(par[u]!=-1 &&
low[v]>=disc[u])   AP[u]=1;
            if(par[u]==-1 && child>1)
AP[u]=1;
        }
        else if(v != par[u])
low[u]=min(low[u],disc[v]);
    }
}
int main()
{
    int n,e,u,v,i,j,cnt=0;
    scin2(n,e);
    for(i=1;i<=e;i++){
        scin2(u,v); graph[u].pb(v);
graph[v].pb(u);
    }
    ms(par,-1);
    DFSAP(1);
    for(i=1;i<=n;i++){
        if(AP[i])   cnt+=1;
    }
    cout<<cnt<<endl;
    return 0;
}
/** GT.08 - Articulation Bridge **/
vl graph[100005];
ll
visited[100005],started[100005],parent[10000
5],low[100005];
vector< pair<ll,ll> > rasta; /// in this
vector all bridges are stored from
rasta[x].ff to rasta[x].ss
bool mark[100005];
ll Time;
void make_graph(ll u, ll v){
    graph[u].pb(v);
}
void DFS(ll u, ll p = -1){
    if(visited[u]==1) return ;
    visited[u] = 1; ll child = 0;
parent[u] = p;
    started[u] = (++Time); low[u] =
started[u];
    for(ll i = 0; i < graph[u].size();
i++){
        if(visited[graph[u][i]]==0){
            child++; DFS(graph[u][i],u);
            low[u] =
min(low[u],low[graph[u][i]]);

if(low[graph[u][i]]>started[u]) {mark[u]
=
1;if(u<graph[u][i])rasta.pb(mp(u,graph[u][i
]));else rasta.pb(mp(graph[u][i],u));}
        }
        else if(parent[u]!=graph[u][i])
low[u] = min(low[u],started[graph[u][i]]);
    }
}
int main(){
    int t,T; scin(T);
    RUN_CASE(t,T){
        ll n, m;
        ms(visited,0); ms(started,0);
ms(mark,0);
        ms(parent,0); ms(low,0); scln(n);
        ll u , v, sum = 0;
        for(ll i = 0; i < n ; i++){
            sc("%lld (%lld)",&u,&m);
            for(ll j = 0; j < m; j++){
                scln(v); make_graph(u,v);
            }
        }
        for(ll i = 0; i < n; i++)
if(visited[i]==0) DFS(i);
        sort(all(rasta));
        CASEL(t);
        pf("%lld critical
links\n",(ll)rasta.size());
        for(ll i = 0; i < rasta.size();
i++) pf("%lld - %lld\n",rasta[i].ff,
rasta[i].ss);
        rasta.clear();
        for(ll i = 0; i < n; i++)
graph[i].clear();
    }
    return 0;
}
/** GT.09 - MaxFlow Template(Ford
Fulkerson's Algorithm)
    FAQ:
১) একাধিক সোর্স বা সিংক থাকলে কি করতে হবে?
-> একটি সুপার সোর্স এবং একটি সুপার সিংক বানিয়ে নিতে হবে। সুপার সোর্স
এর সাথে মূল গ্রাফের সোর্সগুলোর এজ ক্যাপাসিটি হবে ইনফিনিটি এবং সুপার
সিংক এর সাথে মূল গ্রাফের সিংক গুলোর এজ ক্যাপাসিটি হবে ইনফিনিটি। ফ্লো
কল দিতে হবে সুপার সোর্স থেকে সুপার সিংক এ । (উদাঃ সুপার সোর্স ০ এবং
সুপার সিংক n+1)
২) নোড ক্যাপাসিটি দেয়া থাকলে কি করতে হবে?
-> নোডটা কে দুই ভাগে ভাগ করে ফেলবো। ভাগ দুইটি কে নতুন এজ দিয়ে
সংযুক্ত করবো। যেই এজের ক্যাপাসিটি হবে ওই নোডের ক্যাপাসিটির সমান।
যেমনঃ n সংখ্যক নোডের একটা গ্রাফে একটা নোড যদি I হয় এবং তার
ক্যাপাসিটি যদি c হয় তবে নোডটাকে ভাঙ্গলে দুইটা নোড পাবো
I এবং n+I । এবং এদের এজ ক্যাপাসিটি হবে c .
আমরা A নোডটা Ain এবং Aout এই দুটি নোডে ভাগ করেছি। এখন
আসল গ্রাফ যতগুলো এজ A তে প্রবেশ করেছে সেগুলো প্রবেশ করবে Ain
এ এবং আসল গ্রাফে যতগুলো এজ A থেকে বাইরে গিয়েছে সেগুলো এখন
বাইরে যাবে Aout থেকে। Ain থেকে Aout এ একটা এজ প্রবেশ করবে
যেটার ক্যাপাসিটি হবে এজ এর ক্যাপাসিটির সমান।
একটা নোডকে স্প্লিট করলে তাদের মধ্যকার এজ অবশ্যই ডিরেক্টেড হবে।
গ্রাফ ডিরেক্টেড হোক কিংবা আনডিরেক্টেড হোক।
৩) দুই বন্ধু একই নোড থেকে যাত্রা শুরু করে একই গন্তব্যে পৌছাতে চায়
কিন্তু দুইজনেই চায় ভিন্ন ভিন্ন রাস্তা ব্যবহার করে যেতে, তারমানে একই এজ
কখনো ২জন ব্যবহার করতে পারবে না। এধরণের পথকে এজ ডিসজয়েন্ট পাথ
বলে। তোমাকে বলতে হবে কোনো একটা গ্রাফে দুটি এজ ডিসজয়েন্ট পাথ আছে
নাকি?
-> সাধারণ ম্যাক্স-ফ্লো ব্যবহার করেই এজ ডিসজয়েন্ট পাথ বের করা যায়।
শুরুর নোডকে সোর্স এবং গন্তব্য নোডকে সিংক ধরবে। এবার সবগুলো এজ এর
ক্যাপাসিটি বানিয়ে দাও ১ এর সমান। এখন যদি তুমি সোর্স থেকে সিংকে দুই ফ্লো
পাঠাতো পারো সেটার মানে হলো দুটি ডিসজয়েন্ট পাথ আছে। প্রতিটা এজের
ক্যাপাসিটি ১ হওয়াতে ২ ফ্লো যে দুটি পথে গিয়েছে তাদের মধ্যে কমন এজ থাকা
সম্ভব না।
ঠিক একই ভাবে তুমি একটা গ্রাফে সর্বোচ্চ কয়টা ডিসজয়েন্ট পাথ থাকা
সম্ভব অথবা দুই বন্ধুর জায়গায় K টা বন্ধু থাকলে কি হতো বের করে ফেলতে
পারবে।
**/
#define sz 105
bool vis[sz];
int n,e,cap[sz][sz],flow[sz][sz],par[sz];
bool BFS(int src,int des)   /** Return
true if there is a path from source to
destination.Also,fills parent[] to store
the path **/
{
    ms(vis,0);  queue<int>q;
q.push(src);    vis[src] = 1;   par[src]
= -1;
    while(!q.empty()){  /** Standard BFS
Code using adjacency matrix **/
        int u = q.front();
        q.pop();
        for(int v=1 ; v<=n ; v++){
            if(!vis[v] &&
(cap[u][v]-flow[u][v])>0){
                q.push(v);  par[v] = u;
vis[v] = 1;
            }
        }
    }
    return vis[des];
}
int FordFulkerson(int src,int des){
    int u,v,mxFlow=0;
```

```
        while(BFS(src,des)){
            int mnPathFlow = infinity;
            for(v=des ; v!=src ; v=par[v]){
/** Traverse the path & find out minimum edge capacity **/
                u = par[v];
                mnPathFlow = min(mnPathFlow , cap[u][v]-flow[u][v]);
            }
            for(v=des ; v!=src ; v=par[v]){
/** Update the flow for every edge of the path **/
                u = par[v];
                flow[u][v] += mnPathFlow;
                flow[v][u] -= mnPathFlow;
            }
            mxFlow += mnPathFlow;
        }
        return mxFlow;
}
int main()
{
        int i,j,k,u,v,c,src,des,ans;
        ms(cap,0);  ms(flow,0);
        cin>>n>>src>>des>>e;
        for(i=1 ; i<=e ; i++){
            cin>>u>>v>>c;   cap[u][v] += c;
            cap[v][u] += c;  /** This line means that the graph is bidirectional.If the graph is unidirectional & there're no edge v-u the cap[v][u]=0  **/
        }
        ans = FordFulkerson(src,des);
        cout<<ans<<endl;
        return 0;
}
/** GT.10 - MaxFLow Template using Adjacency List(Ford Fulkerson's Algorithm)
    এডজাসেন্ট লিস্ট ব্যাবহার করলে গ্রাফ ইউনিডিরেকশনাল হোক কিংবা বাইডিরেকহনাল, এজ দুইদিকেই যুক্ত করতে হবে। কারণ, ম্যাক্স ফ্লো তে ব্যাক এজ ব্যাবহার করে অপ্টিমাল সল্যুশন বের করতে হয়। ইউনিডিরেকশনাল হলে ক্যাপাসিটি এরে একমুখী হবে। তার মানে গ্রাফ একমুখী। এজন্য, লিস্টে দুইদিকে এজ যুক্ত করলেও সমস্যা হবে না। বরং দুইদিকে যুক্ত না করলে কাজ করবে না।
    Theory of BPM:
    1. MaxBPM = MaxFlow
    2. Max Size of Independent Set = Total Node - Max BPM(here, edge of a graph is the dependency)
    3. Vertex Cover = Max BPM ( Vertex Cover: You have to select minimum number of vertex so that each & every edges r connected to these vertex)
    4. Edge cover = n - Max BPM ( Edge cover: You have to select minimum number of edges so that they cover all the vertex)
**/
#define sz 205
bool vis[sz];
int n,e,par[sz],cap[sz][sz],flow[sz][sz];
vector<int>graph[sz];
bool BFS(int src,int des)    /** Return true if there is a path from source to destination.Also,fills parent[] to store the path **/
{
    ms(vis , 0);    queue<int>q;    q.push(src);    vis[src] = 1;   par[src] = -1;
    while(!q.empty()){  /** Standard BFS using adjacency list **/
        int u = q.front();
        q.pop();
        for(int i=0 ; i<(int)graph[u].size() ; i++){
            int v = graph[u][i];
            if(!vis[v] && (cap[u][v]-flow[u][v])>0){
                q.push(v);
                par[v] = u;
                vis[v] = 1;
            }
        }
    }
    return vis[des];
}
int FordFulkerson(int src,int des){
    int u,v,mxFlow=0;
    while(BFS(src,des)){
        int mnPathFlow = infinity;
        for(v=des ; v!=src ; v=par[v]){
/** Traverse the path & find out minimum edge capacity **/
            u = par[v];
            mnPathFlow = min(mnPathFlow , cap[u][v]-flow[u][v]);
        }
        for(v=des ; v!=src ; v=par[v]){
/** Update the flow for every edge of the path **/
            u = par[v];
            flow[u][v] += mnPathFlow;
            flow[v][u] -= mnPathFlow;
        }
        mxFlow += mnPathFlow;
    }
    return mxFlow;
}
void addEdge(int u,int v){
    graph[u].pb(v);
    graph[v].pb(u);
}
int main()
{
    int i,j,k,u,v,c,src,des,ans;
    scin2(n,e); scin2(src,des);
    for(i=1 ; i<=e ; i++){
        cin>>u>>v>>c;   cap[u][v] += c;
        cap[v][u] += c;     /** This line means that the graph is bidirectional.If the graph is unidirectional & there're no edge v-u the cap[v][u]=0  **/
        addEdge(u,v);
    }
    ans = FordFulkerson(src,des);
cout<<ans<<endl;
    return 0;
}
/** GT.11 - BPM using Kuhn's Algorithm ; Complexity - O(VE) **/
int Left[1005],Right[1005],vis[1005];
vi graph[1005];
bool Kuhn(int u){
    for(int i=0 ; i<(int)graph[u].size() ; i++){
        int v = graph[u][i];
        if(vis[v])continue;
        vis[v] = 1;
        if(Right[v]==-1 || Kuhn(Right[v])){
            Right[v] = u;   Left[u] = v;
            return true;    ///Matching possible
        }
    }
    return false;   ///Matching not possible
}
int BPM(int setA){
    ms(Left , -1); ms(Right , -1); int m=setA;  int cnt = 0;
    for(int i=1 ; i<=m ; i++){  /// m-total node of left part(set A) of bipartite graph ; We can calculate the number of node of set A by bi-coloring the graph with two color white & black OR we can put total number of node of the given graph here
        ms(vis , 0);
        if(Kuhn(i)) cnt++;
    }
    return cnt;
}
int main()
{
    int i,j,k,u,v,n,e,bpm;
    scin2(n , e);
    for(i=1 ; i<=e ; i++){
        scin2(u , v);   graph[u].pb(v); graph[v].pb(u);
    }
```

```
    bpm = BPM(5);
    cout<<bpm<<endl;
    return 0;
}
/** DS.01 - Order Set Template **/
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
template<typename T> using orderset =
tree<T,null_type,less<T>,rb_tree_tag,tree_or
der_statistics_node_update>;
orderset<int> X ; orderset<int>::iterator
it ;
X.insert(1);
cout<<*X.find_by_order(1)<<endl;
cout<<X.order_of_key(-5)<<endl;
/** DS.02 - DSU Template By Component Size
**/
#define sz 100005
int par[sz],Size[sz];
void makeset(int n){
    for(int i=1 ; i<=n ; i++){
        par[i] = i; Size[i] = 1;
    }
}
int FindSet(int node){
    if(par[node] == node)    return node;
    return par[node] = FindSet(par[node]);
}
void UnionSet(int nodeA,int nodeB){
    int a = FindSet(nodeA);
    int b = FindSet(nodeB);
    if(a != b){
        if(Size[a] < Size[b])    swap(a,b);
        par[b] = a;
        Size[a] += Size[b];
    }
}
int main()
{
    int i,j,k,n,e,u,v;
    scin2(n,e); makeset(n);
    for(i=1 ; i<=e ; i++){
        scin2(u,v); UnionSet(u,v);
    }
    for(i=1 ; i<=n ; i++)
        pf("Parent of %d = %d\tSize of
this component =
%d\n",i,FindSet(i),Size[FindSet(i)]);
    return 0;
}
/** DS.03 - MO's Algorithm **/
#define sz 200005
int blocksz,ans[sz],ara[sz],sum;
struct info{
    int id,l,r;
    bool operator < (const info &p)const{
        int blockno = l/blocksz , pblockno
= p.l/blocksz;
        if(blockno == pblockno) return
r<p.r;
        else    return blockno<pblockno;
    }
}query[sz];
void Add(int index){
    sum += ara[index];
}
void Remove(int index){
    sum -= ara[index];
}
int main()
{
    int n,q;
    scin(n);
    blocksz = sqrt(n);
    for(int i=0 ; i<n ; i++)scin(ara[i]);
    scin(q);
    for(int i=0 ; i<q ; i++){
        scin2(query[i].l , query[i].r);
        query[i].id = i;
    }
    sort(query , query+q);
    int l=0,r=-1;
    /** Calculate answer for all query in
OFFline **/
    for(int i=0 ; i<q ; i++){
        while(r < query[i].r)    Add(++r);
        while(l < query[i].l)
Remove(l++);
        while(r > query[i].r)
Remove(r--);
        while(l > query[i].l)    Add(--l);
        ans[query[i].id] = sum;
    }
    for(int i=0 ; i<q ; i++)
        cout<<ans[i]<<endl;
    return 0;
}
/** DS.04 - Square Root Decomposition
Template(Point update ; Sum query) **/
int n,numBlock,blockSize;
/** size of the blocks and the number of
blocks r equal**/
int ara[100005],block[320];
void Build(){
    for(int i=0 ; i<n ; i++)
        block[i/blockSize] += ara[i];
}
int Query(int lft,int rgt){
    int
sum=0,sourceBlock,destBlock,nxtBlock,prevBlo
ck;
    sourceBlock = lft/blockSize;
    destBlock = rgt/blockSize;
    if(sourceBlock == destBlock)    /** IF
left and right r existing in same block **/
    {
        for(int i=lft ; i<=rgt ; i++)
            sum += ara[i];
    }
    else{
        nxtBlock = sourceBlock+1;
        prevBlock = destBlock-1;
        for(int i=lft ;
i<=nxtBlock*blockSize-1 ; i++)
            sum += ara[i];      /** Add
1st partially relevant block **/
        for(int i=nxtBlock ; i<=prevBlock
; i++)
            sum += block[i];    /** Add
all fully relevant block **/
        for(int i=destBlock*blockSize ;
i<=rgt ; i++)
            sum += ara[i];      /** Add
last partially relevant block **/
    }
    return sum;
}
void update(int id,int val){
    int blockID = id/blockSize;
block[blockID] += (val - ara[id]);
ara[id] = val;
}
int main()
{
    int i,j,k,temp,l,r,q,ans,type,id,val;
    scin(n);
    for(i=0 ; i<n ; i++)
        scin(ara[i]);
    numBlock = (int)sqrt(n*1.0)+1;
    blockSize = numBlock;
    Build();
    scin(q);
    for(i=1 ; i<=q ; i++){
        scin(type);
        if(type == 0){
            scin2(id,val);  update(id,val);
        }
        else{
            scin2(l,r); ans = Query(l,r);
            cout<<ans<<endl;
        }
    }
    return 0;
}
/** DS.05 - Standard Segment Tree(Lazy) **/
void push_down(ll node,ll b,ll e){
    tree[node] += lazy[node];
    if(b != e){
```

```
            lazy[node*2] += lazy[node];
            lazy[node*2+1] += lazy[node];
        }
        lazy[node] = 0;
    }
    void Build(ll node,ll b,ll e){    /** Build Tree for Range Minimum Query **/
        if(b == e){
            tree[node] = ara[b]; return;
        }
        ll mid = (b+e)/2;
        Build(node<<1 , b , mid);
        Build(1+(node<<1) , mid+1 , e);
        tree[node] = min(tree[node<<1] , tree[1+(node<<1)]);
    }
    void Update(ll node,ll b,ll e,ll i,ll j,ll val)      /** Increase all the element in a range **/
    {
        if(lazy[node] != 0) push_down(node , b , e);
        if(i>e || j<b)  return;
        if(b>=i && e<=j){
            tree[node] += val;
            if(b != e){
                lazy[node*2] += val;
                lazy[node*2+1] += val;
            }
            return;
        }
        ll mid = (b+e)/2;
        Update(node<<1 , b , mid , i , j , val);
        Update(1+(node<<1) , mid+1 , e , i , j , val);
        tree[node] = min(tree[lft] , tree[rgt]);
    }
    ll Query(ll node,ll b,ll e,ll i,ll j)    /** Range Minimum Query **/
    {
        if(i>e || j<b)  return longlimit;
        if(lazy[node] != 0) push_down(node , b , e);
        if(b>=i && e<=j)     return tree[node];
        ll mid = (b+e)/2;
        ll p1 = Query(node<<1 , b , mid , i , j);
        ll p2 = Query(1+(node<<1) , mid+1 , e , i , j);
        return min(p1 , p2);
    }
    /** DS.06 - Find the maximum value & number of times it appears **/
    int ara[sz];    pii tree[sz*4];
    pii combine(pii a,pii b){
        if(a.first > b.first)
            return a;                                   /** First-value , Second-frequency**/
        if(b.first > a.first)
            return b;
        return make_pair(a.first , a.second+b.second);    /** When value is same,then sum the frequency**/
    }
    void init(int node,int b,int e){
        if(b == e){
            tree[node] = make_pair(ara[b] , 1); return;
        }
        int mid = (b+e)/2;
        init(node<<1 , b , mid);
        init(1+(node<<1) , mid+1 , e);
        tree[node] = combine(tree[left] , tree[right]);
    }
    pii query(int node,int b,int e,int i,int j){
        if(i>e || j<b)  return make_pair(-infinity , 0);
        if(b>=i && e<=j)     return tree[node];
        int mid = (b+e)/2;
        pii p1 = query(node<<1 , b , mid , i , j);
        pii p2 = query(right , mid+1 , e , i , j);
        return combine(p1 , p2);
    }
    void update(int node,int b,int e,int pos,int val){
        if(pos>e || pos<b)  return;
        if(b>=pos && e<=pos){
            tree[node] = make_pair(val , 1);    return;
        }
        int mid = (b+e)/2;
        update(node<<1 , b , mid , pos , val);
        update(1+(node<<1) , mid+1 , e , pos , val);
        tree[node] = combine(tree[left] , tree[right]);
    }
    int main()
    {
        int i,j,n,lb,ub,pos,val,q,cmd;
        pii ans;
        scin2(n,q);
        for(i=1;i<=n;i++)    scin(ara[i]);
        init(1 , 1 , n);
        for(i=1 ; i<=q ; i++){
            scin(cmd);       /** 0-update; 1-query **/
            if(cmd == 0){
                scin2(pos , val);
                update(1 , 1 , n , pos , val);
            }
            else{
                scin2(lb , ub);
                ans = query(1 , 1 , n , lb , ub);
                pf("Maximum value=%d\tFrequency=%d\n" , ans.first , ans.second);
            }
        }
        return 0;
    }
    /** DS.07 - Maximum Sub Array Sum with point update **/
    struct data
    {
        ll sum,pref,suff,ans;
    }tree[4*sz];
    int ara[sz];
    data combine(data left,data right){
        data res;
        res.sum=left.sum+right.sum;
        res.pref=max(left.pref,left.sum+right.pref);
        res.suff=max(right.suff,right.sum+left.suff);
        res.ans=max(max(left.ans,right.ans),left.suff+right.pref);
        return res;
    }
    data make_data(int val){
        data res;
        res.sum=val;     res.pref=val;
        res.suff=val;    res.ans=val;
        /** You must take minimum one element,thats why res.pref=val;res.suff=val;
        res.ans=val; otherwise,
        it will res.pref=max(0,val);res.suff=max(0,val);res.ans=max(0,val);**/
        return res;
    }
    void init(int node,int b,int e){
        if(b == e){
            tree[node]=make_data(ara[b]);
    return;
        }
        int mid=(b+e)/2;
```

```
    init(node<<1 , b , mid);
    init(1+(node<<1) , mid+1 , e);
    tree[node] = combine(tree[node<<1] , tree[1+(node<<1)]);
}
void update(int node,int b,int e,int idx,int val){
    if(idx>e || idx<b)  return;
    if(b>=idx && e<=idx){
        tree[node]=make_data(val);  return;
    }
    int mid=(b+e)/2;
    update(node<<1 , b , mid , idx , val);
    update(1+(node<<1) , mid+1 , e , idx , val);
    tree[node] = combine(tree[node<<1] , tree[1+(node<<1)]);
}
data query(int node,int b,int e,int i,int j){
    if(i>e || j<b)  return make_data(-infinity);
    if(b>=i && e<=j)    return tree[node];
    int mid=(b+e)/2;
    data p1=query(node , b , mid , i , j);
    data p2=query(1+(node<<1) , mid+1 , e , i , j);
    return combine(p1,p2);
}
int main()
{
    int i,j,x,y,n,fg,q,val;
    data res;
    scin(n);
    for(i=1;i<=n;i++)   scin(ara[i]);
    init(1,1,n);    scin(q);
    for(i=1;i<=q;i++){
        scin(fg);
        if(fg == 0){
            scin2(x,val);
update(1,1,n,x,val);
        }
        else{
            scin2(x,y);
res=query(1,1,n,x,y);
            pf("%lld\n",res.ans);
        }
    }
    return 0;
}
/** DS.08 - Maximum SubRectangle Area using segmentTree + BS ; We can calculate maximum rectangle area using init,query & calculate function**/
#define sz 2005
char s[sz][sz];
int n,m,ara[sz];
pii tree[3*sz];   /** 1st element-min value of range ; 2nd element-index of min value **/
void init(int node,int b,int e){
    if(b == e){
        tree[node] = make_pair(ara[b] , b); return;
    }
    int mid=(b+e)/2;
    init(node<<1 , b , mid);
    init(1+(node<<1) , mid+1 , e);
    tree[node].first = min(tree[node<<1].first , tree[1+(node<<1)].first);
    tree[node].second = (tree[node<<1].first == tree[node].first)?tree[node<<1].second:tree[1+(node<<1)].second;
}
pii query(int node,int b,int e,int i,int j){
    if(b>e || b>j || e<i)
        return make_pair(infinity,infinity);
    if(b>=i && e<=j)
        return tree[node];
    int mid=(b+e)/2;
    pii ret1 = query(node<<1 , b , mid , i , j);
    pii ret2 = query(1+(node<<1) , mid+1 , e , i ,j);
    pii ret;    ret.first = min(ret1.first , ret2.first);
    ret.second = (ret.first==ret1.first)?ret1.second:ret2.second;
    return ret;
}
int Calculate(int b,int e)  /** BS for calculating the area of simple histogram **/
{
    if(b > e)
        return 0;
    if(b == e)
        return ara[b];
    int ret1,ret2,ret3;
    pii p = query(1,1,n,b,e);
    ret1 = p.first*(e-b+1);
    ret2 = Calculate(b , p.second-1);
    ret3 = Calculate(p.second+1 , e);
    return max(ret1 , max(ret2 , ret3));
}
int main()
{
    int i,j,k,t,T,ans,cnt;
    scin(T);
    RUN_CASE(t,T){
        ans=0;  ms(ara,0);  scin2(m,n);
        for(i=1 ; i<=m ; i++)
        {
            sc("%s",s[i]);
            for(j=0 ; j<n ; j++)
/** Here,0 means space , 1 means block ; so, we r trying to create histogram **/
            {
                if(s[i][j] == '0')
/** Height of histogram means number of consecutive 0 from this row to upper most row **/
                    ara[j+1] += 1;
                else
                    ara[j+1] = 0;
/** Here,consecutive 0 from this row is 0 **/
            }
            init(1,1,n);    /** Building histogram from row 1 to i **/
            cnt = Calculate(1,n);   /** Maximum sub rectangle area of this histogram **/
            ans = max(ans,cnt);
        }
        pf("Case %d: %d\n",t,ans);
    }
    return 0;
}
/** DS.09 - Segment Tree + Offline Query
How many horizontal lines are completely lies in segment l to r?
You're given an array with n distinct elements & q queries. In each query you'll given l[i],r[i]. You have to print the number of pairs of integers q,w(l[i]<=q,w<=r[i]) such that ara[q] is the divisor of ara[w].
Solution:
Suppose, we have a pair of index(i,j) where ara[i] is the divisor of ara[j]. We say it a segment on a 2D plane. Now we have to count the number of segment those are completely lies in range [x,y] on that 2D plane.
1.  For each number between [1,n] we have to store its multiples between [1,n]. Actually here in divisor's id, we stored all its multiple's id.
2.  Store the query according to l[i] of each query. (see the solution for details)
3.  Now, iterate from n to 1. For each i, increase the value of endpoint of the segment starting from i. Calculate the sum of total segment lies in range i to
```

```
    qry[i].first **/
int ara[sz],id[sz],tree[4*sz],ans[sz];
void Update(int node,int b,int e,int
pos,int val){
    if(pos>e || pos<b) return;
    if(b>=pos && e<=pos){
        tree[node] += val; return;
    }
    int mid = (b+e)>>1;
    if(pos <= mid)
        Update(node<<1 , b , mid , pos ,
val);
    else
        Update(1+(node<<1) , mid+1 , e ,
pos , val);
    tree[node] =
tree[node<<1]+tree[1+(node<<1)];
}
int Query(int node,int b,int e,int i,int
j){
    if(i>e || j<b) return 0;
    if(b>=i && e<=j) return tree[node];
    int mid = (b+e)>>1;
    return (Query(node<<1 , b , mid , i ,
j) + Query(1+(node<<1) , mid+1 , e , i ,
j));
}
vi segment[sz];     /** segment[i][j]
means i is the divisor's id & j is the
multiple's id **/
vector<pii>qry[sz]; /** qry[i][j] means
l=i && qry[i][j]={r , query's SL} **/
int main()
{
    int
i,j,k,n,q,l,r,lft,rgt,x,y,totseg,cnt=0,bads
z=0;
    scn2(n,q);
    for(i=1 ; i<=n ; i++){
        scin(ara[i]); id[ara[i]] = i;
///ID of each distinct element
    }
    for(i=1 ; i<=q ; i++){
        scn2(lft,rgt);
        qry[lft].pb(make_pair(rgt , i));
    }
    for(i=1 ; i<=n ; i++){
        for(j=i ; j<=n ; j+=i){
            x = id[i];      ///Divisor's ID
            y = id[j];      ///Multiple's ID
            segment[min(x,y)].pb(max(x,y));
        }
    }
    /// How OFFline Query works
    for(i=n ; i>=1 ; i--){
        for(auto it : segment[i])
            Update(1 , 1 , n , it ,
1);     ///Increase the value of ending
point of the segment starting from this
node. Here, i is the divisor's address ,
it is the multiple's address.
        for(auto it : qry[i])
            ans[it.second] = Query(1 , 1
, n , i , it.first);   ///Calculate the
sum of total segment completely lies in
range i to it.first
    }
    for(i=1 ; i<=q ; i++)
        pf("%d\n",ans[i]);
    return 0;
}
/** DS.10 - Merge Sort Tree Template
Problem:
You're given an array a with n integers.
You've to calculate the total number
of subArrays with sum less than val.
More formally, you are required to calculate
the number of pairs l,r (l≤r) such that
al+al+1+⋯+ar−1+ar<val.
Solution:
1.  Calculate the cumulative sum of the
given array.
2.  Build a merge sort tree on cumulative
sum array.
3.  Then call query from each position to
calculate the number of segment started
    on that position and have sum less
than val.    **/
#define sz 200005
ll ara[sz],sum[sz];
vector<ll>tree[4*sz];
void Init(ll node,ll b,ll e){
    if(b == e){
        tree[node].pb(sum[b]);  return;
    }
    ll mid = (b+e)>>1;
    Init(node<<1 , b , mid);
    Init(1+(node<<1) , mid+1 , e);
    merge(tree[node<<1].begin() ,
tree[node<<1].end() ,
tree[1+(node<<1)].begin() ,
tree[1+(node<<1)].end() ,
back_inserter(tree[node]));
}
ll Query(ll node,ll b,ll e,ll i,ll j,ll
val){
    if(i>e || j<b)  return 0;
    if(b>=i && e<=j)
        return
upper_bound(tree[node].begin() ,
tree[node].end() ,
val-1)-tree[node].begin();
    ll mid = (b+e)>>1;
    ll p1 = Query(node<<1 , b , mid , i ,
j , val);
    ll p2 = Query(1+(node<<1) , mid+1 , e
, i , j , val);
    return p1+p2;
}
int main()
{
    ll i,j,k,n,val,ans=0,x;
    scln2(n , val);
    rep(i,1,n)scln(ara[i]);
    rep(i,1,n)sum[i] = sum[i-1]+ara[i];
    Init(1 , 1 , n);
    for(i=1 ; i<=n ; i++){
        x = Query(1 , 1 , n , i , n ,
sum[i-1]+val);
        ans += x;
    }
    cout<<ans<<endl;
    return 0;
}
/** DS.11 - 2D Segment Tree template
    Sum query of a sub rectangle of the
matrices and update an element of given
matrices **/
int n,m,mat[sz][sz],tree[4*sz][4*sz];
void initY(int nodeX,int bX,int eX,int
nodeY,int bY,int eY){
    if(bY == eY){
        if(bX == eX)     /** We r working
on a single row ; we can directly generate
segTree for row = bX **/
            tree[nodeX][nodeY] =
mat[bX][bY];
        else             /** We r working
on multiple row ; We r creating parent
segTree from two child segTree which
contains sum from row(bX to eX) **/
            tree[nodeX][nodeY] =
tree[nodeX*2][nodeY] +
tree[nodeX*2+1][nodeY];
        return;
    }
    int midY = (bY+eY)/2;
    initY(nodeX , bX , eX , 2*nodeY , bY
, midY);
    initY(nodeX , bX , eX , 2*nodeY+1 ,
midY+1 , eY);
    tree[nodeX][nodeY] =
tree[nodeX][nodeY*2] +
tree[nodeX][nodeY*2+1];
}
void initX(int nodeX,int bX,int eX){
    if(bX == eX){
```

```
891             initY(nodeX, bX, eX, 1, 1, m);
892             return;
893         }
894         int midX=(bX+eX)/2;
895         initX(2*nodeX , bX , midX);
896         initX(2*nodeX+1, midX+1 , eX);
897         initY(nodeX , bX , eX , 1 , 1 ,m);
    /** Merge segment trees of two rows to
    create parent segment tree **/
898     }
899     int sumQueryY(int nodeX,int nodeY,int
    bY,int eY,int ly,int ry){
900         if(ly > ry) return 0;
901         if(ly==bY && ry==eY)     return
    tree[nodeX][nodeY];
902         int midY = (bY+eY)/2;
903         int ret1 = sumQueryY(nodeX , nodeY*2
    , bY , midY , ly , min(ry,midY));
904         int ret2 = sumQueryY(nodeX ,
    nodeY*2+1 , midY+1 , eY , max(ly,midY+1)
    , ry);
905         return ret1+ret2;
906     }
907     int sumQueryX(int nodeX,int bX,int eX,int
    lx,int rx,int ly,int ry){
908         if(lx > rx) return 0;
909         if(lx==bX && rx==eX)     return
    sumQueryY(nodeX , 1 , 1 , m , ly , ry);
910         int midX = (bX+eX)/2;
911         int ret1 = sumQueryX(nodeX*2 , bX ,
    midX , lx , min(rx,midX) , ly , ry);
912         int ret2 = sumQueryX(nodeX*2+1 ,
    midX+1 , eX , max(lx,midX+1) , rx , ly ,
    ry);
913         return ret1+ret2;
914     }
915     void updateY(int nodeX,int bX,int eX,int
    nodeY,int bY,int eY,int x,int y,int val){
916         if(bY == eY){
917             if(bX == eX)
918                 tree[nodeX][nodeY] = val;
919             else
920                 tree[nodeX][nodeY] =
    tree[nodeX*2][nodeY] +
    tree[nodeX*2+1][nodeY];
921             return;
922         }
923         int midY = (bY+eY)/2;
924         if(y <= midY)
925             updateY(nodeX , bX , eX , nodeY*2
    , bY , midY , x , y , val);
926         else
927             updateY(nodeX , bX , eX ,
    nodeY*2+1 , midY+1 , eY , x , y , val);
928         tree[nodeX][nodeY] =
    tree[nodeX][nodeY*2]+tree[nodeX][nodeY*2+1];
       ///Merge two columns
929     }
930     void updateX(int nodeX,int bX,int eX,int
    x,int y,int val){
931         if(bX == eX){
932             updateY(nodeX , bX , eX , 1 , 1 ,
    m , x , y , val);
933             return;
934         }
935         int midX = (bX+eX)/2;
936         if(x <= midX)
937             updateX(nodeX*2 , bX , midX , x ,
    y , val);
938         else
939             updateX(nodeX*2+1 , midX+1 , eX ,
    x , y , val);
940         updateY(nodeX , bX , eX , 1 , 1, m ,
    x , y , val);       ///Merge two rows
941     }
942     int main()
943     {
944         int i,j,k,q,x1,y1,x2,y2,ans,val,id;
945         scin2(n,m);
946         for(i=1 ; i<=n ; i++){
947             for(j=1 ; j<=m ; j++)
948                 scin(mat[i][j]);
949         }
950         initX(1,1,n);    scin(q);
951         for(i=1 ; i<=q ; i++){
952             scin(id);
953             if(id == 0){ ///Update
954                 scin2(x1,y1);    scin(val);
955                 updateX(1 , 1 ,n , x1, y1 ,
    val);
956             }
957             else{     ///Sum Query
958                 scin2(x1,y1);    scin2(x2,y2);
959                 ans = sumQueryX(1 , 1 , n ,
    x1 , x2 , y1 , y2);
960                 cout<<ans<<endl;
961             }
962         }
963         return 0;
964     }
965     /** DS.12 - Trie Template
966     ট্রাই এর কিছু ব্যবহার:
967     ১. একটা ডিকশনারিতে অনেকগুলো শব্দ আছে, কোনো একটা শব্দ আছে
    নাকি নাই খুজে বের করতে হবে। এই প্রবলেমটা আমরা উপরের কোডেই সলভ
    করেছি।
968     ২. ধরো তোমার ৩ বন্ধুর টেলিফোন নম্বর হলো "৫৬৭৮", "৪৩২২",
    "৫৬৭"। তুমি যখন প্রথম বন্ধুকে ডায়াল করবে তখন ৫৬৭ চাপার সাথে সাথে
    ৩য় বন্ধুর কাছে ফোন চলে যাবে কারণ ৩য় বন্ধুর নাম্বার প্রথম জনের প্রিফিক্স!
    অনেকগুলো ফোন নম্বর দেয়া আছে, বলতে হবে এরকম কোনো নম্বর আছে
    নাকি যেটা অন্য নম্বরের প্রিফিক্স। (UVA 11362)।
969     ৩. একটা ডিকশনারিতে অনেকগুলো শব্দ আছে। এখন কোনো একটা শব্দ
    কয়বার "prefix" হিসাবে এসেছে সেটা বের করতে হবে। যেমন "al"
    শব্দটা উপরের ডিকশনারিতে ৩বার প্রিফিক্স হিসাবে এসেছে (algo,
    algea, also এই সবগুলো শব্দের প্রিফিক্স "al")। এটা বের করার
    জন্য প্রতিটা নোডে একটা কাউন্টার ভ্যারিয়েবল রাখতে হবে, কোনো নোডে
    যতবার যাবে ততবার কাউন্টারের মান বাড়িয়ে দিবে। সার্চ করার সময় প্রিফিক্সটা
    খুজে বের করে কাউন্টারের মান দেখবে।
970     ৪. মোবাইলের ফোনবুকে সার্চ করার সময় তুমি যখন কয়েকটা লেটার লিখো
    তখন সেই প্রিফিক্স দিয়ে কি কি নাম শুরু হয়েছে সেগুলো সাজেশন বক্সে
    দেখায়। এটা তুমি ট্রাই দিয়ে ইমপ্লিমেন্ট করতে পারবে?
971     ৪. দুটি স্ট্রিং এর "longest common substring" বের করতে
    হবে। (subsequence হলে ডিপি দিয়ে সহজে করা যায়, এখানে
    substring চেয়েছি)।
972     (হিন্টস: একটা স্ট্রিং এর শেষ থেকে এক বা একাধিক ক্যারেকটার নেয়া হলে
    সেটাকে স্ট্রিংটার সাফিক্স বলে, যেমন blog এর সাফিক্স
    g,og,log,blog। আর প্রতিটা substring ই কিন্তু কোনো না কোনো
    সাফিক্স এর প্রিফিক্স!! তাই সবগুলো সাফিক্সকে ট্রাইতে ইনসার্ট করলে কাজটা
    সহজ হয়ে যায়!)
973     ৫. (অ্যাডভান্সড) সম্ভবত ২০১১তে ডেফোডিল ইউনিভার্সিটির ন্যাশনাল
    কনটেস্টে এসেছিলো প্রবলেমটা। একটা ডিকশনারি ইনপুট দেয়া থাকবে।
    প্রতিবার ডিকশনারির ২টা শব্দ কুয়েরি দিবে, বলতে হবে তাদের মধ্যে
    common prefix এর দৈর্ঘ্য কত। যেমন algo আর algea এর কমন
    প্রিফিক্স alg, দৈর্ঘ্য ৩। ট্রাইতে ডিকশনারিতে ইনসার্ট করে প্রতি কুয়েরিতে
    শব্দদুটি এন্ড-মার্ক থেকে LCA (lowest common ancestor) বের
    করে প্রবলেমটা সলভ করা যায়।
974     **/
975     struct node{
976         bool endmark;
977         node *next[26+1];
978         node(){
979             endmark = false;
980             for(int i=0 ; i<26 ; i++)
981                 next[i] = NULL;
982         }
983     };
984     node *root;
985     void Insert(char* str , int len){
986         node* curr = root;
987         for(int i=0 ; i<len ; i++){
988             int id = str[i]-'a';
989             if(curr->next[id] == NULL)
990                 curr->next[id] = new node();
991             curr = curr->next[id];
992         }
993         curr->endmark = true;
994     }
995     bool Search(char *str,int len){
996         node* curr = root;
997         for(int i=0 ; i<len ; i++){
998             int id = str[i]-'a';
999             if(curr->next[id] == NULL)
1000                return false;
1001            curr = curr->next[id];
1002        }
1003        return curr->endmark;
1004    }
1005    void Destroy(node *curr)     /** If we pass
    root then the whole tree will be deleted **/
```

```
{
    for(int i=0 ; i<26 ; i++){
        if(curr->next[i] != NULL)
            Destroy(curr->next[i]);
    }
    delete(curr);
}
int main()
{
    int i,j,k,n,q;
    root = new node();
    pf("Enter the number of word: ");
    scin(n);
    for(i=0 ; i<n ; i++){
        char str[50];
        sc("%s",str);
        Insert(str , strlen(str));
    }
    pf("Enter the number of query: ");
    scin(q);
    for(i=1 ; i<=q ; i++){
        char str[50];
        sc("%s",str);
        if(Search(str , strlen(str)))
            pf("The string u searched for is found\n");
        else
            pf("The string u searched for is not found\n");
    }
    return 0;
}
/** DS.13 - LCA template using sparse table **/
/** complexity of build: O(nlogn) ; complexity of query: O(logn)    **/
#define sz 100005
bool vis[sz];
int level[sz],parent[sz],st[sz][22];
vi graph[sz];
void BFS(int start)     /** Pre-calculate the parent & level of each node **/
{
    ms(parent,-1);  ms(level,0);  ms(vis,0);
    queue<int>q;
    vis[start]=1;   level[start]=0;  q.push(start);
    while(!q.empty()){
        int u=q.front();    q.pop();
        for(int i=0 ; i<graph[u].size() ; i++){
            int v=graph[u][i];
            if(!vis[v]){
                vis[v]=1;   parent[v] = u;  level[v] = 1+level[u];  q.push(v);
            }
        }
    }
}
void LCA_Init(int n)            /** Here, n is the number of node **/
{
    ms(st,-1);                  /** Initially, 2^j th parent of each node is -1 **/
    for(int i=1 ; i<=n ; i++)
        st[i][0] = parent[i];   /** 2^0 th parent of each node is its original parent **/
    for(int j=1 ; (1<<j)<=n ; j++){
        for(int i=1 ; i<=n ; i++){
            if(st[i][j-1] != -1){
                st[i][j] = st[st[i][j-1]][j-1];     /** kono node er 2^j th parent holo
                    oi node er 2^j-1 th parent er 2^j-1 th parent. For example, 1ta node er
                    2^4=16th parent holo tar 2^3=8th parent er 2^3=8th parent **/
            }
        }
    }
}
int LCA_Query(int n,int p,int q)    /** Here, n is the number of node. p & q are two nodes whose LCA we've to found out **/
{
    if(level[p] < level[q])
        swap(p,q);                  /** P will always be more deeper node than q **/
    int Log=1;
    while(1){
        int nxt = Log+1;
        if((1<<nxt) > level[p])
            break;
        Log++;
    }
    for(int i=Log ; i>=0 ; i--){    /** Took p & q in same level **/
        if(level[p]-(1<<i) >= level[q])
            p = st[p][i];
    }
    if(p == q)
        return p;   /** Here,q is the ancestor of p **/
    for(int i=Log ; i>=0 ; i--)     /** Now,both nodes r in same level. Here we'll try to take both of them in a level together where parent of both of them will be equal **/
    {
        if(st[p][i]!=-1 && st[p][i]!=st[q][i]){
            p=st[p][i];
            q=st[q][i];
        }
    }
    return parent[p];
}
int main()
{
    int i,j,k,n,e,u,v,q;
    scin2(n,e);
    rep(i,1,e){
        scin2(u,v);
        graph[u].pb(v);
        graph[v].pb(u);
    }
    BFS(1);
    LCA_Init(n);
    scin(q);
    rep(i,1,q){
        scin2(u,v);
        pf("Lowest common ancestor of %d & %d = %d\n",u,v,LCA_Query(n,u,v));
    }
    return 0;
}
/** DS.14 - Path Cost In a Tree
    You're given a graph contains n node & m edges. You've to operate two different queries:
    q1. Update the cost of node x to v
    q2. Answer the total node cost between two nodes not necessary to be ancestor , decendend
    You've to perform 10^5 queries
**/
#define sz 30005
bool vis[sz];
int level[sz],parent[sz],sptable[sz][22],st[sz],ed[sz],cost[sz],ara[2*sz],tree[6*sz],curr_time;
/** level-level of each node ; parent-parent of each node ; stable-sparse table ;
    st-discovering time of each node ; ed-finishing time of each node ;
    cost-number of genies in each node ; ara-reforming cost by discovering & finishing time
    tree-build segment tree **/
vi graph[sz];
/** LCA part **/
void BFS(int start)         /** Pre-calculate the parent & level of each
```

```
node **/
{
    ms(parent,-1);  ms(vis,0);  ms(level,0);
    queue<int>q;
    vis[start]=1;   level[start]=0;
q.push(start);
    while(!q.empty()){
        int u=q.front();    q.pop();
        for(int i=0 ; i<graph[u].size() ;
i++){
            int v=graph[u][i];
            if(!vis[v]){
                vis[v] = 1; parent[v] = u;
                level[v] = 1+level[u];
q.push(v);
            }
        }
    }
}
void LCA_Init(int n)            /** Here,
n is the number of node **/
{
    ms(sptable,-1);                 /**
Initially, 2^j th parent of each node is
-1 **/
    for(int i=1 ; i<=n ; i++)
        sptable[i][0] = parent[i];   /**
2^0 th parent of each node is its original
parent **/
    for(int j=1 ; (1<<j)<=n ; j++){
        for(int i=1 ; i<=n ; i++){
            if(sptable[i][j-1] != -1){
                sptable[i][j] =
sptable[sptable[i][j-1]][j-1];     /**
kono node er 2^j th parent holo
                oi node er 2^j-1 th parent
er 2^j-1 th parent. For example, 1ta node er
                2^4=16th parent holo tar
2^3=8th parent er 2^3=8th parent **/
            }
        }
    }
}
int LCA_Query(int n,int p,int q)    /**
Here, n is the number of node. p & q are
two nodes whose LCA we've to found out **/
{
    if(level[p] < level[q])
        swap(p,q);                  /** P
will always be more deeper node than q **/
    int Log=1;
    while(1){
        int nxt = Log+1;
        if((1<<nxt) > level[p])
            break;
        Log++;
    }
    for(int i=Log ; i>=0 ; i--){    /**
Took p & q in same level **/
        if(level[p]-(1<<i) >= level[q])
            p = sptable[p][i];
    }
    if(p == q)
        return p;   /** Here,q is the
ancestor of p **/
    for(int i=Log ; i>=0 ; i--)     /**
Now,both nodes r in same level. Here we'll
try to take both of them in a level
together where parent of both of them will
be equal **/
    {
        if(sptable[p][i]!=-1 &&
sptable[p][i]!=sptable[q][i]){
            p=sptable[p][i];
q=sptable[q][i];
        }
    }
    return parent[p];
}/** Segment Tree Part **/
void DFS(int start){
    vis[start]=1;
    st[start] = ++curr_time;
    for(int i=0 ; i<graph[start].size() ;
i++){
        int v=graph[start][i];
        if(!vis[v]) DFS(v);
    }
    ed[start] = ++curr_time;
}
void init(int node,int b,int e){
    if(b == e){
        tree[node] = ara[b];    return;
    }
    int mid=(b+e)/2;
    init(node<<1 , b , mid);
    init(1+(node<<1) , mid+1 , e);
    tree[node] =
tree[node<<1]+tree[1+(node<<1)];
}
void update(int node,int b,int e,int
i,int val){
    if(i>e || i<b)  return;
    if(i==b && i==e){
        tree[node]=val;
        return;
    }
    int mid=(b+e)/2;
    update(node<<1 , b , mid , i , val);
    update(1+(node<<1) , mid+1 , e , i ,
val);
    tree[node] =
tree[node<<1]+tree[1+(node<<1)];
}
int query(int node,int b,int e,int i,int
j){
    if(i>e || j<b)  return 0;
    if(b>=i && e<=j)    return tree[node];
    int mid=(b+e)/2;
    int p1 = query(node<<1 , b , mid , i
, j);
    int p2 = query(1+(node<<1) , mid+1 ,
e , i , j);
    return p1+p2;
}
void Clean(int n){
    curr_time=0;
    for(int i=0 ; i<=n ; i++){
        graph[i].clear();   vis[i]=0;
parent[i]=-1;
    }
}
int main()
{
    int
i,j,k,t,T,n,u,v,q,id,lca,tot,start,finish,p
1,p2;
    scin(T);
    RUN_CASE(t,T){
        scin(n);
        rep(i,1,n)scin(cost[i]);
        rep(i,2,n){
            scin2(u,v);
            graph[u+1].pb(v+1);
graph[v+1].pb(u+1);
        }
        DFS(1);
        for(i=1 ; i<=n ; i++){
            start = st[i];  finish = ed[i];
            ara[start] = cost[i];
ara[finish] = -cost[i];
        }
        init(1,1,2*n);  BFS(1);
LCA_Init(n);
        scin(q);
        pf("Case %d:\n",t);
        rep(i,1,q){
            sc("%d %d %d",&id,&u,&v);
            if(id){
                u += 1;   cost[u] = v;
                update(1,1,2*n,st[u],v);
                update(1,1,2*n,ed[u],-v);
            }
            else{
                u+=1; v+=1;
                if(u > v)swap(u,v);
                lca = LCA_Query(n,u,v);
                start = min(st[u] ,
st[lca]);
```

```
                finish = max(st[u] , st[lca]);
                p1 = query(1,1,2*n,start,finish);
                start = min(st[v] , st[lca]);
                finish = max(st[v] , st[lca]);
                p2 = query(1,1,2*n,start,finish);
                tot = p1+p2-cost[lca];
                pf("%d\n",tot);
            }
        }Clean(n);
    }
    return 0;
}
/** DS.15 - LCA Problem - QTrREE2
    Problem: You r given a tree with n nodes & n-1 edges with w cost
    We will ask you to perfrom some instructions of the following form:
    DIST a b : ask for the distance between node a and node b
    or
    KTH a b k : ask for the k-th node on the path from node a to node b
    Solution:   dis(a,b) = dis(root,a)+dis(root,b) - 2*dis(root,LCA)    **/
#define sz 100005
bool vis[sz];
ll level[sz],parent[sz],sptable[sz][25],dis[sz];
vector<ll> graph[sz],cost[sz];
void BFS(ll start)  /** Pre-calculate the parent,level & distance of each node from root node **/
{
    ms(level,0);    ms(vis,0);    ms(parent,-1);
    queue<ll>q; vis[start] = 1;
    level[start] = dis[start] = 0;
    q.push(start);
    while(!q.empty()){
        ll u=q.front(); q.pop();
        for(ll i=0 ; i<(ll)graph[u].size() ; i++){
            int v=graph[u][i];
            if(!vis[v]){
                vis[v] = 1; dis[v] = cost[u][i]+dis[u];
                level[v] = 1+level[u]; parent[v] = u;  q.push(v);
            }   }   }   }

void LCA_Init(ll n)
/** Here, n is the number of node **/
{
    ms(sptable,-1);
/** Initially, 2^j th parent of each node is -1 **/
    for(int i=1 ; i<=n ; i++)
        sptable[i][0] = parent[i];
/** 2^0 th parent of each node is its original parent **/
    for(int j=1 ; (1<<j)<=n ; j++){
        for(int i=1 ; i<=n ; i++){
            if(sptable[i][j-1] != -1){
                sptable[i][j] = sptable[sptable[i][j-1]][j-1];  /** kono node er 2^j th parent holo
                oi node er 2^j-1 th parent er 2^j-1 th parent. For example, 1ta node er 2^4=16th parent holo tar 2^3=8th parent er 2^3=8th parent **/
            }   }   }   }

ll LCA_Query(ll n,ll p,ll q)
/** Here, n is the number of node. p & q are two nodes whose LCA we've to found out **/
{
    if(level[p] < level[q])
        swap(p,q);
/** P will always be more deeper node than q **/
    ll Log=1;
    while(1){
        ll nxt=1+Log;
        if((1<<nxt) > level[p]) break;
        Log++;
    }
    for(ll i=Log ; i>=0 ; i--)
/** Took p & q in same level **/
    {
        if(level[p]-(1<<i) >= level[q]) p = sptable[p][i];
    }
    if(p == q)  return p;
    for(ll i=Log ; i>=0 ; i--)
/** Now,both nodes r in same level. Here we'll try to take both of them in a level together where parent of both of them will be equal **/
    {
        if(sptable[p][i]!=-1 && sptable[p][i]!=sptable[q][i]){
            p=sptable[p][i]; q=sptable[q][i];
        }
    }
    return parent[p];
}
ll KthNodeInPath(ll n,ll u,ll v,ll k)       /** We've to found out the k-th node on the path from node u to node v **/
{

k-=1;
/** u is the 1st node in the path which is already discovered,so decrease k by 1 **/
    ll Log=0,dist,lca;
    lca = LCA_Query(n,u,v);                 /** To move on the path from u to v we've to know the LCA for turning up **/
    dist = abs(level[u]-level[lca]);        /** From u to LCA how many level I can move **/
    while(1){
        ll nxt=1+Log;
        if((1<<nxt) > dist) break;
        Log++;
    }
    for(ll i=Log ; i>=0 ; i--)
    {   /** level[sptable[u][i]]>=level[lca] means that I can't go to upper level from LCA ; I've to turn to v from LCA **/
        if((1<<i)<=k && sptable[u][i]!=-1 && level[sptable[u][i]]>=level[lca])
/** One kind of BS **/
        {   /**     move u to maximum(LCA,kth node on the path)     **/
            k -= (1<<i);
            u = sptable[u][i];
        }
    }
    if(k > 0)
    {   /** K>0 means we didn't go k th node yet ; So,we've to turn up through LCA and go deeper to node v  **/
        k = level[v]-k-level[lca];  /** We've to go upper of k level from node v for found out kth node from u **/
        Log=0;
        dist = abs(level[v]-level[lca]);    /** From v to LCA how many level I can move **/
        while(1){
            ll nxt=1+Log;
            if((1<<nxt) > dist) break;
            Log++;
        }
        for(ll i=Log ; i>=0 ; i--)
/** One kind of BS **/
```

```
1365            {
1366                if((1<<i)<=k &&
    sptable[v][i]!=-1 &&
    level[sptable[v][i]]>=level[lca]){
1367                    k -= (1<<i);
1368                    v = sptable[v][i];
1369                }
1370            }
1371            return v;
1372        }
1373        else
1374            return u;
1375    }
1376    void Clean(ll n){
1377        for(ll i=0 ; i<=n ; i++){
1378            graph[i].clear();   cost[i].clear();
1379        }
1380    }
1381    int32_t main()
1382    {
1383        ll i,j,k,t,T,u,v,d,n,ans,lca;
1384        scln(T);
1385        RUN_CASE(t,T){
1386            scln(n);
1387            rep(i,2,n){
1388                scln2(u,v); scln(d);
1389                graph[u].pb(v); graph[v].pb(u);
1390                cost[u].pb(d);  cost[v].pb(d);
1391            }
1392            BFS(1); LCA_Init(n);     string cmd;
1393            while(1){
1394                cin>>cmd;
1395                if(cmd == "DONE")
1396                    break;
1397                else if(cmd == "DIST"){
1398                    scln2(u,v);
1399                    lca = LCA_Query(n,u,v);
1400                    ans =
    dis[u]+dis[v]-2*dis[lca];
1401                    pf("%lld\n",ans);
1402                }
1403                else{
1404                    scln2(u,v);
1405                    scln(k);
1406                    ans =
    KthNodeInPath(n,u,v,k);
1407                    pf("%lld\n",ans);
1408                }
1409            }
1410            pf("\n");   Clean(n);
1411        }
1412        return 0;
1413    }
1414    /** DS.16 - LCA problem
1415    You're given a rooted tree .Find the
    length of the shortest and the length
1416  of the longest road on the path between
    the two given nodes. **/
1417    #define sz 100005
1418    bool vis[sz]; int parent[sz],level[sz];
1419    vi graph[sz]; map<pii , int>cost;
1420    void BFS(int start)
1421    {
1422        ///BFS FOR LCA
1423    }
1424    struct info
1425    {
1426        int par,lowest,highest;
1427        info() {}
1428        info(int a,int b,int c){
1429            par=a; lowest=b; highest=c;
1430        }
1431    }sptable[sz][20];
1432    void Init(int n){
1433        for(int i=0 ; i<=n ; i++){
1434            for(int j=0 ; j<20 ; j++)
1435                sptable[i][j] =
    info(-1,infinity,0);
1436        }
1437    }
1438    void LCA_Init(int n){
1439        Init(n);
1440        for(int i=1 ; i<=n ; i++){
1441            sptable[i][0].par = parent[i];
1442            if(parent[i] != -1){
1443                sptable[i][0].lowest =
    min(sptable[i][0].lowest,
    cost[make_pair(i,parent[i])]);
1444                sptable[i][0].highest =
    max(sptable[i][0].highest,
    cost[make_pair(i,parent[i])]);
1445            }
1446        }
1447        for(int j=1 ; (1<<j)<=n ; j++){
1448            for(int i=1 ; i<=n ; i++){
1449                if(sptable[i][j-1].par != -1){
1450                    sptable[i][j].par =
    sptable[sptable[i][j-1].par][j-1].par;
1451                    sptable[i][j].lowest =
    min(min(sptable[i][j].lowest,sptable[i][j-1]
    .lowest) ,
    sptable[sptable[i][j-1].par][j-1].lowest);
1452                    sptable[i][j].highest =
    max(max(sptable[i][j].highest,sptable[i][j-1
    ].highest) ,
    sptable[sptable[i][j-1].par][j-1].highest);
1453                }
1454            }
1455        }
1456    }
1457    pii LCA_Query(int n,int p,int q){
1458        if(p == q)return make_pair(0,0);
1459        pii ret = make_pair(infinity,0);
1460        if(level[p] < level[q]) swap(p,q);
1461        int Log=0;
1462        while(1){
1463            int nxt = Log+1;
1464            if((1<<nxt) > level[p])
1465                break;
1466            Log++;
1467        }
1468        for(int i=Log ; i>=0 ; i--){
1469            if(level[p]-(1<<i) >= level[q]){
1470                ret.first = min(ret.first ,
    sptable[p][i].lowest);
1471                ret.second = max(ret.second ,
    sptable[p][i].highest);
1472                p = sptable[p][i].par;
1473            }
1474        }
1475        if(p == q) return ret;
1476        for(int i=Log ; i>=0 ; i--){
1477            if(sptable[p][i].par!=-1 &&
    sptable[p][i].par!=sptable[q][i].par){
1478                ret.first = min(ret.first ,
    min(sptable[p][i].lowest ,
    sptable[q][i].lowest));
1479                ret.second = max(ret.second ,
    max(sptable[p][i].highest ,
    sptable[q][i].highest));
1480                p=sptable[p][i].par;
1481                q=sptable[q][i].par;
1482            }
1483        }
1484        ret.first = min(ret.first ,
    cost[make_pair(p,parent[p])]);
1485        ret.first = min(ret.first ,
    cost[make_pair(q,parent[q])]);
1486        ret.second = max(ret.second ,
    cost[make_pair(p,parent[p])]);
1487        ret.second = max(ret.second ,
    cost[make_pair(q,parent[q])]);
1488        return ret;
1489    }
1490    int main()
1491    {
1492        int i,j,k,n,small,big,u,v,w,q;
1493        scin(n);
1494        rep(i,2,n){
1495            scin2(u,v); scin(w);
1496            graph[u].pb(v); graph[v].pb(u);
1497            cost[make_pair(u,v)]=w;
    cost[make_pair(v,u)]=w;
1498        }
1499        BFS(1); LCA_Init(n); scin(q);
1500        rep(i,1,q){
1501            scin2(u,v);
1502            pii ans = LCA_Query(n,u,v);
```

```
        pf("%d %d\n",ans.first,ans.second);
    }
    return 0;
}
/** DS.18 - LCA Problem **/
/** Problem: you are given a rooted tree, where each node contains an integer value. And the value of a node is strictly greater than the value of its parent. Now you
are given a node and an integer query. You have to find the greatest possible parent of this node (may include the node itself), whose value if greater than or equal to the given query integer.    **/
bool vis[sz]; vi graph[sz];
int level[sz],parent[sz],sptable[sz][25],cost[sz];
void BFS(int start)
{
    ///BFS FOR LCA
}
void LCA_Init(int n){
    ms(sptable,-1);
    for(int i=1 ; i<=n ; i++)
        sptable[i][0]=parent[i];
    for(int j=1 ; (1<<j)<=n ; j++){
        for(int i=1 ; i<=n ; i++){
            if(sptable[i][j-1] != -1)
                sptable[i][j] = sptable[sptable[i][j-1]][j-1];
        }
    }
}
int LCA_Query(int node,int val){
    int Log=1,retnode=node;
    while(1){
        int nxt=Log+1;
        if((1<<nxt)>level[node]) break;
        Log++;
    }
    for(int i=Log ; i>=0 ; i--){     /** One kind of BS **/
        int v=sptable[retnode][i];
        if(sptable[retnode][i]!=-1 && cost[v]>=val)
            retnode=sptable[retnode][i];
    }
    return retnode;
}
void Clean(int n){
    for(int i=0 ; i<=n ; i++)
        graph[i].clear();
}
int main(){
    int i,j,k,t,T,n,q,u,v,ans,par,val;
    scin(T);
    RUN_CASE(t,T){
        scin2(n,q); cost[1]=1;
        rep(i,2,n){ /** Nodes denotes from 1 to n **/
            scin2(par,val);
graph[i].pb(par+1);
            graph[par+1].pb(i); cost[i]=val;
        }
        BFS(1); LCA_Init(n);
        pf("Case %d:\n",t);
        rep(i,1,q){
            scin2(u,val); ans = LCA_Query(u+1,val);
            pf("%d\n",ans-1);
        }
        Clean(n);
    }
    return 0;
}
/** DP রদীক্ষঃ 1.In DP, dp[2][12] is more efficient than dp[12][2]
    2.  In DP, calculating from bigger to smaller state is more efficient.(In that case we haven't memset dp array in every case.) **/
/** DP.01 - 0-1 KnapSack Template(Iterative) **/
int n,CAP,weight[105],cost[105],dp[105][1005];
/** State-1:Number of element ; State-2:Maximum size of knapsack **/
bool take[105];
void KnapSack(){
    for(int i=1 ; i<=n ; i++)/** i th element **/   {
        dp[i][0]=0;
        for(int j=1 ; j<=CAP ; j++) /** Present knapsack size is j **/  {
            if(j >= weight[i])
                dp[i][j] = max(cost[i]+dp[i-1][j-weight[i]] , dp[i-1][j]);
            else
                dp[i][j] = dp[i-1][j];
        }
    }
}
int main()
{
    int i,j;
    ms(dp,0);   scin2(n,CAP);
    for(i=1;i<=n;i++)
        scin2(weight[i],cost[i]);
    KnapSack();
    pf("\nOptimal answer is = %d\n",dp[n][CAP]);
    /** Identify the selected ITEM **/
    j=CAP;
    for(i=n ; i>=1 ; i--){
        if(dp[i][j] == dp[i-1][j])  ;
        else{
            take[i]=1;  j -= (weight[i]);
        }
    }
    /** Print the selected ITEM **/
    for(i=1 ; i<=n ; i++){
        if(take[i] == 1)    pf("ITEM-%d is selected\n",i);
    }
    }
    return 0;
}
/** DP.02 - Space Optimized knapsack **/
int n,cap,wt[505],val[505],dp[2][2000005];
/// State-1: ith and (i-1)th element ; State-2:Maximum size of knapsack
/// Constraint: cap <= 2000000 ; n <= 500 ; val[i] <= 10^7 ; wt[i] <= 10^7
void SpaceOptimizedKnapsack(){
    int present,past;
    for(int i=1 ; i<=n ; i++){
        if(i%2 == 0)
            present=0, past=1;
        else
            present=1, past=0;
        for(int j=1 ; j<=cap ; j++){
            if(wt[i] <= j)
                dp[present][j] = max(val[i]+dp[past][j-wt[i]] , dp[past][j]);
            else
                dp[present][j] = dp[past][j];
        }
    }
    pf("%d\n",dp[present][cap]);
}
int main()
{
    scin2(cap , n);
    for(int i=1 ; i<=n ; i++)
        scin2(val[i] , wt[i]);
    SpaceOptimizedKnapsack();
    return 0;
}
/** DP.03 - LCS Template **/
string s1,s2,res;
int len1,len2,dp[1005][1005]; /** State-1: Size of s1 ; State-2: Size of s2**/
int LCS(int pos1,int pos2){
    if(pos1>=len1 || pos2>=len2)
```

```
    return 0;
1637    if(dp[pos1][pos2] != -1)    return dp[pos1][pos2];
1638    else{
1639        if(s1[pos1] == s2[pos2])
1640            return dp[pos1][pos2] = 1+LCS(pos1+1,pos2+1);
1641        else{
1642            int res1=LCS(pos1,pos2+1);
1643            int res2=LCS(pos1+1,pos2);
1644            return dp[pos1][pos2] = max(res1,res2);
1645        }
1646    }
1647 }
1648 void Print(int pos1,int pos2) /** Print the Longest Common Subsequence **/{
1649    if(pos1>=len1 || pos2>=len2)    return;
1650    if(s1[pos1] == s2[pos2]){   /** Take the character **/
1651        res += s1[pos1] ;
1652        Print(pos1+1,pos2+1);
1653    }
1654    else{
1655        int res1=dp[pos1][pos2+1];
1656        int res2=dp[pos1+1][pos2];
1657        if( res1 >= res2 ) Print( pos1 , pos2 + 1 ) ; /** Go,where the result is max **/
1658        else Print( pos1 + 1 , pos2 ) ;
1659    }
1660 }
1661 void PrintAll(int pos1,int pos2) /** Print all LCS **/{
1662    if(pos1>=len1 || pos2>=len2){
1663        cout<<res<<endl;    return;
1664    }
1665    if(s1[pos1] == s2[pos2]){
1666        res += s1[pos1];
1667        PrintAll(pos1+1,pos2+1);
1668        res.erase(res.end()-1);     /** Delete the last Character **/
1669    }
1670    else{
1671        int res1=dp[pos1][pos2+1];
1672        int res2=dp[pos1+1][pos2];
1673        if(res1 > res2)
1674            PrintAll(pos1,pos2+1);
1675        else if(res1 < res2)
1676            PrintAll(pos1+1,pos2);
1677        else    /** If both call return same ans,then go both side **/
1678        {
1679            PrintAll(pos1,pos2+1);
1680            PrintAll(pos1+1,pos2);
1681        }
1682    }
1683 }
1684 int main()
1685 {
1686    ms(dp,-1);
1687    int i,j,ans;
1688    cin>>s1; cin>>s2;
1689    len1=s1.size(); len2=s2.size();
1690    ans = LCS(0,0);
1691    pf("Size of LCS : %d\n",ans);
1692    pf("LCS:\n");   res = "" ; Print(0,0) ;
1693    cout<< res << endl ;
1694    pf("ALL LCS:\n");   res = ""; PrintAll(0,0);
1695    return 0;
1696 }
1697 /** DP.04 - LIS Template - O(n^2) **/
1698 int n=7,value[]={-1000000,5,0,9,2,7,3,4};
1699 int dp[1000],dir[1000];
1700 /**Printing Soln**/
1701 void solution(int start){
1702    while(dir[start] != -1){
1703        pf("Index %d ; value = %d\n",start,value[start]);
1704        start=dir[start];
1705    }
1706 }
1707 int longest(int u){
1708    if(dp[u] != -1)
1709        return dp[u];
1710    int maxi=0;
1711    for(int v=u+1;v<=n;v++){
1712        if(value[v] > value[u]){
1713            if(longest(v) > maxi){
1714                maxi=longest(v);
1715                dir[u]=v;
1716            }
1717        }
1718    }
1719    return dp[u]=1+maxi;
1720 }
1721 int main()
1722 {
1723    ms(dp,-1); ms(dir,-1);
1724    int LIS=0,start,i;
1725    for(int i=1;i<=n;i++){
1726        if(longest(i) > LIS){
1727            LIS=longest(i);
1728            start=i;
1729        }
1730    }
1731    pf("LIS = %d , starting point %d\n\n\n",LIS,start);
1732    solution(start);
1733    return 0;
1734 }
1735 /** DP.05 - LIS Template - O(n log (k)) **/
1736 const int inf = 2000000000; ///Infinity
1737 int n,ara[2000005],L[2000005],I[2000005],sequence[2000005];
1738 void takeInput(){
1739    scin(n);
1740    for(int i=0 ; i<n ; i++)
1741        scin(ara[i]);
1742 }
1743 int LisNlogK(){
1744    int i;
1745    I[0]=-inf;
1746    for(i=1 ; i<=n ; i++){
1747        I[i] = inf; L[i]=n;
1748    }
1749    int LisLen=0; /** Keeps the maximum position where a data is inserted **/
1750    for(i=0 ; i<n ; i++){
1751        int lo=0,hi=LisLen,mid;
1752        while(lo <= hi) /** Find the suitable position for ara[i] **/
1753        {
1754            mid=(lo+hi)/2;
1755            if(ara[i] > I[mid])
1756                lo=mid+1;
1757            else
1758                hi=mid-1;
1759        }
1760        /** Observe the BS carefully , when the BS ends low>hi and we put our item in I[lo] **/
1761        I[lo]=ara[i];   L[i]=lo;
1762        if(LisLen < lo) /** LisLen contains maximum position **/
1763            LisLen=lo;
1764    }
1765    return LisLen;
1766 }
1767 void findSequence(int mxlen){
1768    int i=0,j;
1769    /** at first find the position of the item whose L[] is maximum **/
1770    for(j=1 ; j<n ; j++){
1771        if(L[j] > L[i]) i=j;
1772    }
1773    /** initialize the position in sequence where the items can be added ; observe that the
1774    data r saving from right to left **/
1775    int top=L[i]-1;
1776    /** insert the item in ith position to sequence **/
1777    sequence[top--] = ara[i];
1778    for(j=i-1 ; j>=0 ; j--){
```

```
            if(ara[j]<ara[i] && L[j]==L[i]-1){
                /** we have found a valid item. so we'll save it **/
                i=j;    sequence[top--]=ara[i];
            }
        }
        pf("LIS is : \n");
        for(i=0 ; i<mxlen ; i++)
            pf("%d ",sequence[i]);
        pf("\n");
    }
    int main()
    {
        takeInput();    int res=LisNlogK();
    pf("The LIS length is %d\n",res);
        findSequence(res);
        return 0;
    }
    /** DP.06 - Matrix Chain Multiplication Template **/
    int row[105],col[105],dp[105][105];
    bool vis[105][105];
    int FuN(int BEG,int END){
        if(BEG >= END)  return 0;
        if(vis[BEG][END])   return dp[BEG][END];
        int ret=1<<30;
        for(int MID=BEG ; MID<END ; MID++){
            int LeftRet = FuN(BEG , MID);
            int RightRet = FuN(MID+1 , END);
            int MulLeftRight = row[BEG]*col[MID]*col[END];
            int tot = LeftRet+RightRet+MulLeftRight;
            ret = min(ret , tot);
        }
        vis[BEG][END]=1;
        return dp[BEG][END] = ret;
    }
    int main()
    {
        int n;  cin>>n;
        for(i=0 ; i<n ; i++)
            cin>>row[i]>>col[i];
        cout<<FuN(0 , n-1)<<endl;
        return 0;
    }
    /** DP.07 - Maximum Sum of K-subArrays
        Problem: You've given an array of n elements. You've to select k non-empty
        subArrays so that total sum of all sub arrays will be maximize. **/
    ll n,k,ara[5005],dp[2][5005][5005];
    bool vis[2][5005][5005];
    ll FuN(bool isStart,ll pos,ll subArrayNo){
        if(pos > n){
            if(subArrayNo == k) return 0LL;
            else    return -longlimit;
        }
        if(vis[isStart][pos][subArrayNo])
            return dp[isStart][pos][subArrayNo];
        ll ret1=-longlimit,ret2=-longlimit,ret3=-longlimit;
        if(isStart){
            ret1 = ara[pos] + FuN(0 , pos+1 , subArrayNo+1);
            ret3 = ara[pos] + FuN(1 , pos+1 , subArrayNo+1);
            ret2 = FuN(1 , pos+1 , subArrayNo);
        }
        else{
            ret1 = ara[pos] + FuN(0 , pos+1 , subArrayNo);
            ret2 = FuN(1 , pos+1 , subArrayNo);
            ret3 = ara[pos] + FuN(1 , pos+1 , subArrayNo);
        }
        vis[isStart][pos][subArrayNo] = 1;
        return dp[isStart][pos][subArrayNo] = max(ret1 , max(ret2 , ret3));
    }
    int main()
    {
        ll i,j,ans;
        ms(vis , 0);    scln2(n , k);
        rep(i , 1 , n)  scln(ara[i]);
        ans = FuN(1 , 1 , 0);
    cout<<ans<<endl;
        return 0;
    }
    /** DP.08 - Digit DP-1  : How many zeros?
    Problem: Jimmy writes down the decimal representations of all natural numbers
    between and including m and n, (m ≤ n).
    How many zeroes will he write down? **/
    ll dp[2][2][15][15];
    /** State-1: is_less_than_given_value ; State-2:
    is_current_position_is_starting_position
        State-3:
    current_position_from_left_side ; State-4:
    total_zero_so_far_in_this_created_number
    **/
    vector<int>num;
    ll DP(ll isSmall , ll isStart , ll pos , ll totZero){
        if(pos >= (ll)num.size())
            return totZero;     /** Return number of zero used in this number so far **/
        if(dp[isSmall][isStart][pos][totZero] != -1)
            return dp[isSmall][isStart][pos][totZero];
        ll ret=0LL,can_be_taken=0;
        if(isSmall)
            can_be_taken=9; /** If it already small,then any digit can be taken **/
        else
            can_be_taken=num[pos];  /** If its prefix is equal then digit can be taken till num[pos] **/
        if(!isStart){    /** If its not a starting digit , then 0 can be taken **/
            for(int i=0 ; i<=can_be_taken ; i++){
                ret += DP(isSmall | (i<num[pos]) , 0 , pos+1 , (i==0)+totZero); /** If i<num[pos] then isSmall OR 1 = 1  and if i==0 the number of totzero in this number will increased by 1**/
            }
        }
        else{
            for(int i=1 ; i<=can_be_taken ; i++){    /** we can't put zero in starting position **/
                ret += DP(isSmall | (i<num[pos]) , 0 , pos+1 , (i==0)+totZero);
            }
            ret += DP(1 , 1 ,  pos+1 , totZero); /** IF its a starting position , then we can put nothing in that position.
                Then the position will be starting position again , and it will be smaller than max range **/
        }
        return dp[isSmall][isStart][pos][totZero] = ret;
    }
    ll Calculate(ll x){
        if(x < 0)
            return 0LL;
        else if(x <= 9)
            return 1LL;
        num.clear();    ms(dp , -1);
        while(x){
            num.pb(x%10);   x /= 10;
        }
        reverse(num.begin() , num.end());
        return DP(0 , 1 , 0 , 0) + 1; /** 0 can't be taken ; So,add 1 by default **/
    }
    int main()
    {
        ll t,T,n,m,ret1,ret2,ans;
        scln(T);
```

```
    RUN_CASE(t,T){
        scln2(m,n);
        ans = Calculate(n) - Calculate(m-1);
        pf("Case %lld: %lld\n",t,ans);
    }
    return 0;
}
/** DP.09 - Digit DP-2: Digit Sum
Problem: Given the numbers a and b, calculate the digit sum(sum of all digit between a and b) of [a,b]. **/
ll dp[2][2][16][16][145];
/** State-1: Is this number smaller than the range
    State-2: Is this position is starting position or not
    State-3: Position
    State-4: Total number of digit used in this number
    State-5: Sum of digit of this number    **/
vector<int>num;
ll FuN(bool isSmall,bool isStart,int pos,int tot_digit,int sum_digit){
    if(pos >= (int)num.size())
        return (ll)sum_digit;

if(dp[isSmall][isStart][pos][tot_digit][sum_digit] != -1)
        return dp[isSmall][isStart][pos][tot_digit][sum_digit];
    ll ret=0,can_be_taken=0;
    if(isSmall)
        can_be_taken=9;
    else
        can_be_taken=num[pos];
    if(!isStart){
        for(int i=0 ; i<=can_be_taken ; i++)
            ret += FuN(isSmall|(i<num[pos]) , 0 , pos+1 , tot_digit+1 , sum_digit+i);
    }
    else{
        for(int i=1 ; i<=can_be_taken ; i++)
            ret += FuN(isSmall|(i<num[pos]) , 0 , pos+1 , tot_digit+1 , sum_digit+i);
        ret += FuN(1 , 1 , pos+1 , 0 , 0);
    }
    return dp[isSmall][isStart][pos][tot_digit][sum_digit] = ret;
}
ll Calculate(ll x){
    if(x <= 0)  return 0;
    num.clear();    ms(dp , -1);
    while(x){
        num.pb(x%10);   x /= 10;
    }
    reverse(num.begin() , num.end());
    return FuN(0 , 1 , 0 , 0, 0);
}
int main()
{
    ll t,T,a,b,ans;
    scln(T);
    RUN_CASE(t,T){
        scln2(a,b);
        if(a > b)   swap(a,b);
        ans = Calculate(b)-Calculate(a-1);
        pf("%lld\n",ans);
    }
    return 0;
}
/** DP.10 - Digit DP 3
    Problem: Count the number of integers in the range [A, B] which are divisible by K and the sum of its digits is also divisible by K. **/
int dp[2][2][15][85][85];

/** State-1: is_less_than_given_value ;
State-2: is_current_position_starting_position
State-3: current_position_from_left_side ;
State-4: remainder_of_value_i_have_made  ;
State-5: remainder_of_sum_of_digit  **/
vector<int>num; int K;
int FuN(int isSmall , int isStart , int pos , int val_remainder , int sum_remainder)
{
    if(pos >= (int)num.size()){
        if(isStart)
            return 0;   /** If its a starting position even now! the value is zero which is invalid **/
        if(val_remainder%K==0 && sum_remainder%K==0) return 1;
        else return 0;
    }

if(dp[isSmall][isStart][pos][val_remainder][sum_remainder] != -1)
        return dp[isSmall][isStart][pos][val_remainder][sum_remainder];
    int ret=0 , can_be_taken=0;
    if(isSmall)
        can_be_taken = 9;   /** If the number already small,then any digit can be taken **/
    else
        can_be_taken = num[pos];    /** If its prefix is equal then digit can be taken till num[pos] **/
    if(!isStart){
        for(int i=0 ; i<=can_be_taken ; i++){    /** If its not a starting digit , then 0 can be taken **/
            ret += FuN(isSmall|(i<num[pos]) , 0 , pos+1 , (val_remainder*10+i)%K , (sum_remainder+i)%K);
        }
    }
    else{
        for(int i=1 ; i<=can_be_taken ; i++){    /** we can't put zero in starting position **/
            ret += FuN(isSmall|(i<num[pos]) , 0 , pos+1 , (val_remainder*10+i)%K , (sum_remainder+i)%K);
        }
        ret += FuN(1 , 1 , pos+1 , 0 , 0);  /** IF its a starting position , then we can also put nothing in that position.
        Then the position will be starting position again , and it will be smaller than max range **/
    }
    return dp[isSmall][isStart][pos][val_remainder][sum_remainder] = ret;
}
int Calculate(int x)
{
    if(x <= 0) return 0;
    num.clear(); ms(dp , -1);
    while(x){
        num.pb(x%10);
        x /= 10;
    }
    reverse(num.begin() , num.end());
    return FuN(0 , 1 , 0 , 0 , 0);
}
int main(){
    int t,T,ans,A,B;
    scin(T);
    RUN_CASE(t,T){
        sc("%d %d %d",&A,&B,&K);
        if(A>B) swap(A,B);
        if(K == 1) ans = B-A+1;    /** All number in the range will fulfill the given condition **/
```

```
        else if(K > 81) ans = 0;
/** None of the number will fulfill the condition digit_sum%K==0; because maximum digit sum for 9 position can be 81 **/
        else ans = Calculate(B)-Calculate(A-1);
        pf("Case %d: %d\n",t,ans);
    }
    return 0;
}
/** DP.11 - BitMask template ; complexity (2^n)*(n^2) **/
int Set(int N,int pos){return N=N | (1<<pos);}
int reset(int N,int pos){return N= N & ~(1<<pos);}
bool check(int N,int pos){return (bool)(N & (1<<pos));}
int w[20][20],n,dp[(1<<15)+2];
int call(int mask){
    if(mask==(1<<n)-1) return 0;    /** Bought everything **/
    if(dp[mask]!=-1) return dp[mask];
    int mn=1<<28;
    for(int i=0; i<n; i++){
        if(check(mask,i)==0){   /** ith item haven't bought yet **/
            int price=w[i][i];  /** Base price of ith item **/
            for(int j=0; j<n; j++){
                if(i!=j and check(mask,j)!=0){  /** Increase the price if I've already bought jth item **/
                    price+=w[i][j];
                }
            }
            int ret=price+call(Set(mask,i));
            mn=min(mn,ret);
        }
    }
    return dp[mask]=mn;
}
int FuN(int isParTaken,int node){
    if(dp[isParTaken][node] != -1)
        return dp[isParTaken][node];
    int ret1=0,ret2=0;
    if(isParTaken || par[node]==0){
///IF parent of this node is taken,then we can both take this node or ignore this node
        ret1 = 0;
        for(int i=0 ; i<(int)graph[node].size() ; i++){
            int v = graph[node][i];
            if(v != par[node])
                ret1 += FuN(0 , v);
        }
        ret2 = 1;
        for(int i=0 ; i<(int)graph[node].size() ; i++){
            int v = graph[node][i];
            if(v != par[node])
                ret2 += FuN(1 , v);
        }
        return dp[isParTaken][node] = min(ret1 , ret2);
    }
    else{       ///IF parent of this node was not taken, then we have to take this node
        ret2 = 1;
        for(int i=0 ; i<(int)graph[node].size() ; i++){
            int v = graph[node][i];
            if(v != par[node])
                ret2 += FuN(1 , v);
        }
        return dp[isParTaken][node] = ret2;
    }
}
int main()
{
    int i,j,k,n,u,v,ans;
    scin(n);
    for(i=1 ; i<n ; i++){
        scin2(u,v);
        graph[u].pb(v); graph[v].pb(u);
    }
    ms(vis , 0);    par[1] = 0; DFS(1);
    ms(vis , 0);    ms(dp , -1);
    ans = FuN(0 , 1);   cout<<ans<<endl;
    return 0;
}
/** DP.12 - Graph + DP - ICPC'19
Problem: You're given a wighted tree. A tree is said to be beautiful if the summation of all pair distance of the vertex of the tree is non-negative(if n=5 , then
d(1,2)+d(1,3)+d(1,4)+d(1,5)+d(2,3)+d(2,4)+d(2,5)+d(3,4)+d(3,5)+d(4,5) >= 0 )
You have to determine whether the given tree is beautiful or not? IF the tree is not beautiful, you have to perform a series of operating to make the given tree beautiful.
The operation is: Select an edge whose weight is negative and increase its weight by 1.
Now, u have to determine the minimum number of times u have to perform the operation to make the given tree beautiful.  **/
bool vis[sz]; vi graph[sz];
int dp[sz],level[sz];
void make_graph(int u,int v){
    graph[u].pb(v); graph[v].pb(u);
}
void DFS(int start){
    vis[start] = 1;
    int sum = 0;
    for(int i=0 ; i<(int)graph[start].size() ; i++){
        ll v = graph[start][i];
        if(!vis[v]){
            level[v] = 1+level[start];
            DFS(v); sum += (1+dp[v]);
```

```
2132            }
2133        }
2134        dp[start] = sum;
2135    }
2136    pii edges[sz];
2137    map<pii , int>edgecost,repeat;
2138    struct info{
2139        int u,v,w,rept;
2140        info(){}
2141        info(int a,int b,int c,int d){
2142            u=a; v=b; w=c; rept=d;
2143        }
2144    };
2145    vector<info>q;
2146    bool cmp(info p1,info p2){
2147        return p1.rept<p2.rept;
2148    }
2149    ll BS(ll sum){
2150        if(sum >= 0LL)
2151            return 0LL;
2152        sort(q.begin() , q.end() , cmp);
2153        ll
        i,j,k,u,v,w,tempsum=fabs(sum),cnt=0LL,lo,hi
        ,mid,res,bad;
2154        while(!q.empty() && tempsum>0){
2155            info top = q.back(); q.pop_back();
2156            u = top.u; v = top.v; w =
        fabs(top.w);
2157            if(level[v] > level[u]) swap(u,v);
2158            lo=0;hi=fabs(w);res=-1;
2159            while(lo <= hi){
2160                mid = (lo+hi)/2;
2161                bad = top.rept*mid;
2162                if(bad >= tempsum){
2163                    res = mid; hi = mid-1;
2164                }
2165                else
2166                    lo = mid+1;
2167            }
2168            if(res == -1){
2169                tempsum -= (top.rept * w);
        cnt += w;
2170            }
2171            else{
2172                tempsum -= (top.rept * res);
2173                cnt += res;
2174                break;
2175            }
2176        }
2177        return cnt;
2178    }
2179    void Clean(ll n){
2180        for(ll i=0 ; i<=n ; i++){
2181            vis[i] = 0; dp[i] = 0;
2182            level[i] = 0; graph[i].clear();
2183        }
2184        q.clear(); edgecost.clear();
        repeat.clear();
2185    }
2186    int main(){
2187        int i,j,k,n,u,v,w,t,T;
2188        ll ans,sum,temp;
2189        scin(T);
2190        RUN_CASE(t,T)
2191        {
2192            sum = 0LL;
2193            scin(n);
2194            for(i=1 ; i<=n-1 ; i++){
2195                scin2(u,v);
2196                scin(w);
2197                edges[i] = make_pair(u,v);
2198                edgecost[make_pair(u,v)] = w;
2199                edgecost[make_pair(v,u)] = w;
2200                make_graph(u,v);
2201            }
2202            DFS(1);
2203            for(i=1 ; i<=n-1 ; i++){
2204                u = edges[i].first; v =
        edges[i].second;
2205                if(level[v] > level[u])
        swap(u,v);
2206                ll under = dp[u];
2207                ll upper = n-(dp[u]+1);
2208                temp = (under+1)*upper;
2209                repeat[make_pair(u,v)] = temp;
2210                if(edgecost[make_pair(u,v)] <
        0)
2211                    q.push_back(info(u , v ,
        edgecost[make_pair(u,v)] , temp));
2212                sum +=
        (temp*edgecost[make_pair(u,v)]);
2213            }
2214            ans = BS(sum); pf("Case %d:
        %lld\n",t,ans); Clean(n);
2215        }
2216        return 0;
2217    }
2218    /** DP.13 - Palindrome Partitioning
2219    Problem: A palindrome partition is the
        partitioning of a string such that each
        separate substring is a palindrome.
2220    Count minimum possible number of
        substrings in a palindrome partition of a
        string **/
2221    int minPalPartition(char str[]){
2222        int n=strlen(str),i,j,k,L;
2223        bool isPal[n][n];   /**
        isPal[i][j]=true if str[i..j] is a
        palindrome **/
2224        int dp[n];       /** Minimum number of
        cuts needed for palindrome partitioning of
        substring str[0..i] **/
2225        for(i=0 ; i<n ; i++)
2226            isPal[i][i]=1; /** Every one
        length substring is a palindrome **/
2227        for(L=2 ; L<=n ; L++){   /** Build
        solution for L size substring **/
2228            for(i=0 ; i<n-L+1 ; i++){    /**
        From index i,take a L length substring **/
2229                j=i+L-1;     /** Here j is the
        ending index **/
2230                if(L == 2)
2231                    isPal[i][j] =
        (str[i]==str[j]); /** If two character
        matched,then its a palindrome **/
2232                else
2233                    isPal[i][j] =
        (str[i]==str[j]) && isPal[i+1][j-1];  /**
        IF str[i+1...j-1] is a palindrome and
        str[i]==str[j] **/
2234            }
2235        }
2236        for(i=0 ; i<n ; i++){
2237            if(isPal[0][i] == true)
2238                dp[i]=0;
2239            else{
2240                dp[i] = infinity;
2241                for(j=0 ; j<i ; j++){
2242                    /** If str[j+1...i] is a
        palindrome & min cut of str[0..j]+1 < min
        cut of str[0..i] then update min cut of
        str[0..i] **/
2243                    if(isPal[j+1][i]==true &&
        1+dp[j]<dp[i])
2244                        dp[i] = 1+dp[j];
2245                }
2246            }
2247        }
2248        return 1+dp[n-1]; /** Return the min
        cut value for complete string **/
2249    }
2250    int main()
2251    {
2252        int t,T,ans;
2253        scin(T);
2254        RUN_CASE(t,T){
2255            char s1[1005];  sc("%s",s1);
2256            ans = minPalPartition(s1);
2257            pf("Case %d: %d\n",t,ans);
2258        }
2259        return 0;
2260    }
2261    /** DP.14 - Longest Palindromic
        Subsequence **/
2262    int dp[1005][1005]; string s;
2263    int LPS(int pos1,int pos2){
2264        if(pos1 == pos2)     /** If there is
        only one character **/
```

```
2265            return 1;
2266        if(s[pos1]==s[pos2] &&
    pos1+1==pos2)     /** If there r only two
    characters & both r same **/
2267            return 2;
2268        if(dp[pos1][pos2] != -1)     return
    dp[pos1][pos2];
2269        else{
2270            if(s[pos1] == s[pos2])
2271                return dp[pos1][pos2] =
    2+LPS(pos1+1, pos2-1);
2272            else
2273                return dp[pos1][pos2] =
    max(LPS(pos1+1,pos2), LPS(pos1,pos2-1));
2274        }
2275    }
2276    /** Print the solution **/
2277    vector<char>v1,v2;
2278    void Print(int pos1,int pos2){
2279        if(pos1 == pos2){
2280            v1.pb(s[pos1]); return;
2281        }
2282        if(s[pos1]==s[pos2] && pos1+1==pos2){
2283            v1.pb(s[pos1]); v2.pb(s[pos2]);
2284            return;
2285        }
2286        if(s[pos1] == s[pos2]){
2287            v1.pb(s[pos1]); v2.pb(s[pos2]);
2288            Print(pos1+1 , pos2-1);
2289        }
2290        else{
2291            if(dp[pos1+1][pos2] >
    dp[pos1][pos2-1]) Print(pos1+1,pos2);
2292            else    Print(pos1,pos2-1);
2293        }
2294    }
2295    int main()
2296    {
2297        int i,j,lps,len;
2298        cin>>s; ms(dp,-1);
2299        lps = LPS(0,s.size()-1);
    cout<<lps<<endl;
```

```
2300        /** Solution Print **/
2301        Print(0,s.size()-1);
2302        reverse(v2.begin() , v2.end());
2303        for(i=0 ; i<v1.size() ; i++)
2304            cout<<v1[i];
2305        for(i=0 ; i<v2.size() ; i++)
2306            cout<<v2[i];
2307        cout<<endl;
2308        return 0;
2309    }
2310    /** DP.15 - Longest Palindromic Substring
    Template
2311        We can find the longest palindrome
    substring in (n^2) time with O(1) extra
    space. The idea is to generate all even
    length and odd length palindromes and keep
    track of the longest palindrome seen so far.
2312        Step to generate odd length palindrome:
2313        Fix a centre and expand in both
    directions for longer palindromes.
2314        Step to generate even length palindrome
2315        Fix two centre ( low and high ) and
    expand in both directions for longer
    palindromes.    **/
2316    void PrintSubstring(string str,int
    low,int high){
2317        pf("Longest palindromic substring : ");
2318        for(int i=low ; i<=high ; i++)
2319            cout<<str[i];
2320        cout<<endl;
2321    }
2322    int LongestPalSubstr(string str){
2323        int
    len=str.size(),start=0,low,high,mxlen=1;
2324        for(int i=1 ; i<len ; i++){
2325            /** Find longest even length
    palindrome with center point i-1 and i **/
2326            low=i-1;     high=i;
2327            while(low>=0 && high<len &&
    str[low]==str[high]){
2328                if(high-low+1 > mxlen){
2329                    start=low;
```

```
2330                    mxlen=high-low+1;
2331                }
2332                --low;  ++high;
2333            }
2334            /** Find longest even length
    palindrome with center point i **/
2335            low=i-1;     high=i+1;
2336            while(low>=0 && high<len &&
    str[low]==str[high]){
2337                if(high-low+1 > mxlen){
2338                    start=low;
    mxlen=high-low+1;
2339                }
2340                --low;  ++high;
2341            }
2342        }
2343        PrintSubstring(str,start,start+mxlen-1);
2344        return mxlen;
2345    }
2346    int main()
2347    {
2348        int mx;
2349        string s1;
2350        cin>>s1;
2351        mx = LongestPalSubstr(s1);
2352        pf("Length of longest palindromic
    substring is = %d\n",mx);
2353        return 0;
2354    }
2355    /** DP.16 - DP on grid (Baker Bro) **/
2356    int m,n,grid[105][105],dp[105][105][105][2];
2357    /** Approach:    এই প্রব্লেমে ধরে নেই যে দুইজন লোক গ্রিডের ১ম
    সেল থেকে শেষ সেলে যাবে এবং পয়েন্টগুলো কালেক্ট করবে। মনে রাখতে
    হবে, এই দুইজন লোকের
2358        রাস্তা শুধুমাত্র গ্রিডের ১ম এবং শেষ সেলেই মিলিত হতে পারবে। আর
    উভয় ব্যাক্তিই হয় নিচে নাহয় ডানে মুভ করতে পারবে। মানে রো এবং কলাম এর
    মান শুধু বাড়তেই পারবে।
2359        আমাদের এপ্রোচ হচ্ছে , দুইজন লোক একইসাথে নতুন কলামে যেতে
    পারবে। এবং একজন একজন করে রো
2360        পরিবর্তন করতে পারবে। ১ম ব্যাক্তি সর্বদা ২য় ব্যাক্তির থেকে নিচের কোন
    রো তে থাকবে। এক্ষেত্রে এমন হতে পারে তারা ২য় কলামের ১ম এবং ২য় রো তে
    তারা আছে। এখন ১ম ব্যাক্তি ২য় রো থেকে একে একে ৫ম রো তে যাওয়ার পর
```

```
    ২য় ব্যাক্তি ২য়,৩য় রো
2361        ভিজিট করতেছে। মানে ১ম ব্যাক্তির ভিজিট করা পথেই যাচ্ছে! যা অসম্ভব।
2362        এজন্য একটা স্টেট রাখতে হবে যে বর্তমান কলামে ১ম ব্যাক্তি ১ বারও
    নিচে গেছে কি না? ১ম ব্যাক্তি যদি নিচে যায়,তাহলে আমি আর জানি না সে এই
    কলাম এর কোন কোন
2363        সেল ভিজিট করছে।সো,২য় ব্যাক্তি নিচে যেতে পারবে না এই কলাম
    বরাবর।
2364        আর যদি ১ম ব্যাক্তি এখনো নিচে যায় নি এই কলাম বরাবর, তাহলে ২য়
    ব্যাক্তি প্রথম ব্যাক্তির রো এর আগের রো পর্যন্ত নিচে যেতে পারবে ১ রো ১ রো করে।
2365        সর্বশেষে, আমরা জানি যে শুরুর এবং শেষের সেলের পয়েন্ট দুইবার
    যোগ হয়েছে। তাই ১বার বিয়োগ করে দিতে হবে। স্টেটগুলো হবেঃ
2366        State-1: বর্তমান কলাম বরাবর ১ম ব্যাক্তি কি নিচের দিকে মুভ
    করছে?
2367        State-2: ১ম ব্যাক্তি বর্তমানে কোন রো তে আছে?
2368        State-3: ২য় ব্যাক্তি বর্তমানে কোন রো তে আছে?
2369        State-4: উভয়ে বর্তমানে কোন কলামে আছে?
2370        বেইস কেস কি হবে নিজে চিন্তা করে বের কর।  **/
2371    int FuN(int row1,int row2,int col,int
    p1_not_moved_in_this_col){
2372        if(row1!=m && col==n)    return
    -infinity;
2373        else if(row1==m && row2==m && col==n
    && p1_not_moved_in_this_col)
2374            return
    grid[row1][col]+grid[row2][col];
2375
    if(dp[row1][row2][col][p1_not_moved_in_this_
    col] != -1)
2376            return
    dp[row1][row2][col][p1_not_moved_in_this_col
    ];
2377        int ret1=0,ret2=0,ret3=0;
2378        if((col==n && row2<row1) || (col<n &&
    row2+1<row1 && p1_not_moved_in_this_col &&
    col>1))
2379            ret1 = grid[row2][col] +
    FuN(row1,row2+1,col,p1_not_moved_in_this_col
    );
2380        if(row1<m)
2381            ret3 = grid[row1][col] +
    FuN(row1+1,row2,col,0);
2382        if(col<n && row1!=row2)
```

```
        ret2 = grid[row1][col] + grid[row2][col] + FuN(row1,row2,col+1,1);
    return dp[row1][row2][col][p1_not_moved_in_this_col] = max(ret1, max(ret2,ret3));
}
int main()
{
    int i,j,t,T,ans;
    scin(T);
    RUN_CASE(t,T){
        scin2(m,n);
        for(i=1 ; i<=m ; i++){
            for(j=1 ; j<=n ; j++)
                scin(grid[i][j]);
        }
        ms(dp,-1);  ans = FuN(1,1,1,1);
        ans -= (grid[1][1]+grid[m][n]);
        pf("Case %d: %d\n",t,ans);
    }
    return 0;
}
/** DP.17 - Problem: LOJ - 1092 - Lighted Panels
এই প্রবলেম এ আমাদের একটা লাইট প্যানেল এর অবস্থা দেওয়া হয়েছে, কোন অবস্থার '*' এর মানে হচ্ছে লাইট জ্বলে আছে, '.' মানে হচ্ছে লাইটা নিভে আছে। আমাদেরকে মিনিমাম মুভে প্যানেল এর সবগুলা লাইট জ্বালাতে হবে। এইখানে যখন কোন পয়েন্ট toggle করা হয় (মানে এইটা যে অবস্থায় আছে জ্বলা থাকলে নিভা, নিভা থাকলে জ্বলে উঠবে) ঐ পয়েন্ট এর সাথে adjacent যে পয়েন্টগুলা থাকে (diagonal সহ) তারাও toggle করবে।
row and column of the grid <= 8
Solution:
1. এই প্রবলেমটা যদি দেখি আমরা যখন কোন পয়েন্ট toggle করি তাহলে কি হবে আমরা যে row তে আছি তার আগের row এবং পরের row এর আমরা যে column এ আছি তার আগের column এবং পরের column নিয়ে কাজ করব।
2. আমি এখন যে row তে আসি তার কিঅবস্থা (মানে কোন কোন পয়েন্ট জ্বলে আসে কি নিভে আসে) যে row থেকে আসলাম তার কি অবস্থা ছিল এবং যে row তে যাব তার কি অবস্থা আছে আমাদের তা জানা থাকা দরকার।
3. State-1: number_of_row ; State-2: bit_representation_of_previous_row ;
State-3: bit_representation_of_current_row
4. আমাদের ক্যালকুলেশন এর জন্য আমাদের পরের row ও দরকার হবে এইটা আমরা কোথায় পাব। আমরা প্রথমেই ইনপুট নেওয়ার সময় একটা array তে রেখে দিতে পারি কোন কোন লাইট এখন জ্বলে আসে।
5. যে row তে আসি এর প্রতিটা combination করে আমরা আমাদের store value গুলা change করে দেখব। এর জন্য আমরা subset mask use করতে পারি। যেহেতু column এর highest limit 8 .তাই (1 << 8) == 256 খুব সহজেই আমরা আমাদের dp এর ভিতর তা চালাতে পারি।
6. যখন আমরা নেক্সড row তে যাব আমাদের sure করতে হবে আমাদের আগের row এর সবগুলা লাইটা জ্বালানো আছে (যদি না থাকে তাহলে আর কোন ভাবেই ঐ লাইটাকে আর জ্বালানো সম্ভব হবে না।) **/
int Set(int N, int pos) {return  N = N | (1<<pos);}
int Reset(int N, int pos) {return  N = N & ~(1<<pos);}
bool Cheek(int N, int pos) {return (bool)(N & (1<<pos));}

bool vis[10][(1<<8)+5][(1<<8)+5];
int row,col,rowcondition[10],dp[10][(1<<8)+5][(1<<8)+5];
char grid[10][10];
int FuN(int idx,int currmask,int prevmask){
    if(idx >= row){
        if(prevmask == (1<<col)-1) return 0;
        else return infinity;
    }
    if(vis[idx][currmask][prevmask])
        return dp[idx][currmask][prevmask];
    int ret = infinity;
    for(int i=0 ; i <= (1<<col)-1 ; i++){
        int cnt = 0;
        int rowrep[3] = {prevmask , currmask , rowcondition[idx+1]};    /** manipulated bit representation of previous row , manipulated bit representation of current row(manipulated from previous row) , original bit representation of next row **/
        for(int j=0 ; j<col ; j++){
            ///Check all possible combination by subset mask
            if(Cheek(i , j) == 0)
                continue;   ///No need to toggle
            ///Need to toggle
            cnt++;
            for(int k=0 ; k<3 ; k++){ ///Toggle the jth light of prevrow,currrow,nxtrow
                rowrep[k] ^= (1<<j);
            }
            if(j+1 < col){  ///Toggle the (j+1)th light of prevrow,currrow,nxtrow
                for(int k=0 ; k<3 ; k++)
                    rowrep[k] ^= (1<<(j+1));
            }
            if(j-1 >= 0){   ///Toggle the (j-1)th light of prevrow,currrow,nxtrow
                for(int k=0 ; k<3 ; k++)
                    rowrep[k] ^= (1<<(j-1));
            }
        }
        if(idx == 0)    ///IF its 1st row,then go to 2nd row. Because,if any light isn't on yet, we can on that light from next row.
            ret = min(ret , cnt+FuN(idx+1 , rowrep[2] , rowrep[1]));
        else if(rowrep[0] == (1<<col)-1)   ///Only IF all the lights are on in our previous row,then we can go to next row
            ret = min(ret , cnt+FuN(idx+1 , rowrep[2] , rowrep[1]));
    }
    vis[idx][currmask][prevmask] = 1;
    return dp[idx][currmask][prevmask] = ret;
}

int main(){
    int i,j,k,t,T,ans,mask;
    scin(T);
    RUN_CASE(t,T){
        scin2(row , col);
        for(i=0 ; i<row ; i++){
            sc("%s",grid[i]);
            mask = 0;
            for(j=0 ; j<col ; j++){
                if(grid[i][j] == '*')
                    mask = Set(mask , j);
            }
            rowcondition[i] = mask;
        }
        ms(vis , 0); ans = FuN(0 , rowcondition[0] , 0);
        if(ans < infinity)
            pf("Case %d: %d\n",t,ans);
        else
            pf("Case %d: impossible\n",t);
    }
    return 0;
}
/** NT.01 - Segmented Sieve Template **/
vector<int>Prime;
bool mark[10000009];
void sieve(int n){
    u know it well-_-
}
ll segmentedSieve(ll L,ll R){
    ll cnt=0;
    bool isPrime[R-L+2];
    for(int i=0 ; i<=R-L+1 ; i++)
        isPrime[i]=true;            /** Initially mark all as prime **/
    if(L == 1)
        isPrime[0]=false;
    for(int i=0 ; i<Prime.size() && Prime[i]*Prime[i]<=R ; i++){
        ll cutPrime=Prime[i];
        ll base=cutPrime*cutPrime;
        if(base < L){
```

```
            base = ((L+cutPrime-1)/cutPrime)*cutPrime; /** Increase base if it already cut by normal sieve **/
        }
        for(ll j=base ; j<=R ; j+=cutPrime)
            isPrime[j-L]=false;     /** Mark the index as composite **/
    }
    for(int i=0 ; i<=R-L ; i++){
        if(isPrime[i] == true){
            /// cout<<L+i<<endl; /**Print the Prime Numbers **/
            cnt += 1;
        }
    }
    return cnt;
}
int main(){
    sieve(10000000);    segmentedSieve(1e9, 1e9+1e5);
    return 0;
}
/** NT.02 - Large BigMOD **/
ll BigMul(ll a,ll b,ll m){
    if(b == 0)  return 0;
    if(b == 1)  return a%m;
    if(b%2 == 0){
        ll temp=b/2;
        ll res=BigMul(a,temp,m);
        return ((res%m)+(res%m))%m;
    }
    else{
        ll temp=BigMul(a,b-1,m);
        return ((a%m)+(temp%m))%m;
    }
}
ll BigMod(ll a,ll b,ll m){
    if(b == 0)  return 1%m;
    if(b%2 == 0){
        ll temp=BigMod(a,b/2,m);
        return BigMul(temp,temp,m)%m;}
    else{
        ll temp=BigMod(a,b-1,m);
        return BigMul(temp,a,m)%m;}
}
int main()
{
    cout<<BigMod(a , b , m)<<endl;
    return 0;
}
/** NT.03 - SievePhi **/
int phi[1000006], mark[1000006];
void sievephi(int n){
    int i,j;
    for(int i=1;i<=n;i++) phi[i] = i;
    phi[1] =1;  mark[1] = 1;
    for(int i=2;i<=n;i++){
        if(!mark[i]){
            for(j=i;j<=n;j+=i){
                mark[j] = 1;
                phi[j] = phi[j] / i *(i-1);
            }
        }
    }
}
/** NT.04 - Prime Factorization in nlog(n) **/
int SF[1000005]; /** Here we store smallest factor for each number **/
void FuN(){
    for( int i = 2 ; i * i <= 10000000 ; i ++ ){
        if( !SF[i] ){
            for( int j = i ; j <= 10000000 ; j += i ){
                if( !SF[j] ) SF[j] = i ;
            }
        }
    }
    for( int i = 2 ; i <= 1e7 ; i ++ ) if( !SF[i] ) SF[i] = i ;
}
vector<int>Factor;
void getFactor(int n){
    Factor.clear();
    while(n != 1){
        Factor.push_back(SF[n]);
        n/=SF[n];
    }
}
/** NT.05 - nCr in nlogn **/
/** nCr in nlogn complexity **/
int SF[1000005];
void Smallest_Factor(){
    U know me-_-
}
ll Power(ll a,ll n){
    ll ret=1;
    for(ll i=1 ; i<=n ; i++)
        ret = ((ret%MOD)*(a%MOD))%MOD;
    return ret;
}
map<int,int>nom,denom;
map<int,int>::iterator it;
void FactorizeNom(int n){
    while(n != 1){
        nom[SF[n]]++;
        n /= SF[n];
    }
}
void FactorizeDenom(int n){
    while(n != 1){
        denom[SF[n]]++;
        n /= SF[n];
    }
}
ll nCr(ll n,ll r)
{
    nom.clear();
    denom.clear();
    ll mn=min(r, n-r), mx=max(r, n-r), i, temp , ans=1 , x , pow;
    for(i=mx+1 ; i<=n ; i++)
        FactorizeNom(i);
    for(i=2 ; i<=mn ; i++)
        FactorizeDenom(i);
    for(it=nom.begin() ; it!=nom.end() ; it++){
        x = it->first; pow = nom[x]-denom[x];
        temp = Power(x , pow); ans *= temp;
    }
    return ans;
}
int main()
{
    Smallest_Factor();
    cout<<nCr(10 , 1)<<endl;
    return 0;
}
/** Z-algorithm Template **/
vi z_fun(string s){
    int i , l , r , n = s.size();
    vi z(n);
    for(i=1,l=0,r=0 ; i<n ; i++){
        if(i <= r)
            z[i] = min(r-i+1 , z[i-l]);
        while(i+z[i]<n && s[z[i]]==s[i+z[i]])
            ++z[i];
        if(i+z[i]-1 > r){
            l=i;
            r=i+z[i]-1;
        }
    }
    return z;
}
int main()
{
    int i,j,n;
    string s;
    cin>>s;
    vi Z = z_fun(s);
    for(i=0;i<Z.size();i++)
        pf("%d ",Z[i]);
    pf("\n");
    return 0;
```

```
}
/** String Hashing Problem:
 You are given a string M and N other strings smaller in length than M. You have to find whether each of these N strings is a substring of M. All strings consist of only alphanumeric characters. **/
 #define szz 100005
string s,sub;   const int p=331;
ll p_pow[szz],inv[szz],hash_sum[szz];
void powER(){
    p_pow[0] = 1;
    for(ll i=1 ; i<szz ; i++)
        p_pow[i] = (p_pow[i-1]*p)%MOD;
}
void modINV(){
    inv[0] = 1;
    inv[1] = BigMod(p , MOD-2);
    for(ll i=2 ; i<szz ; i++)
        inv[i] = (inv[i-1]*inv[1])%MOD;
}
void Hash_Table(){
    ll len = (ll)s.size();
    for(ll i=0 ; i<len ; i++)
        hash_sum[i+1] = (hash_sum[i] + (s[i]-'0'+1)*p_pow[i])%MOD;
}
ll compute_hash(){
    ll hash_value=0LL , len = (ll)sub.size();
    for(ll i=0 ; i<len ; i++){
        hash_value = (hash_value + (sub[i]-'0'+1)*p_pow[i])%MOD;
    }
    return hash_value;
}
map<ll,ll>st[2005];
void subStringStore(ll len){
    ll lb,ub,val,sz = (ll)s.size();
    for(ll i=0 ; i<=sz-len ; i++){
        lb = i+1;
        ub = i+len;
        val = (hash_sum[ub]-hash_sum[lb-1]+MOD)%MOD;
        val = (val*inv[i])%MOD;
        st[len][val]++;
    }
}
bool vis[2005];
int main(){
    ll i,j,k,n,val,len,sz;
    powER();
    modINV();
    cin>>s;
    sz = (ll)s.size();
    Hash_Table();
    scln(n);
    rep(i,1,n){
        cin>>sub;
        len = (ll)sub.size();
        val = compute_hash();
        if(!vis[len]){
            subStringStore(len);
            vis[len] = 1;
        }
        if(st[len][val] > 0)
            pf("Y\n");
        else
            pf("N\n");
    }
    return 0;
}
/** Aho Corasic Template - SPOJ-SUB_PROB **/
const int MX = 2005;
int n;  /// n is the number of queries
char text[1000006],dictionary[MX][MX];
const int goes = 127; /// If MLE comes, set it to 27 for only lowercase letters
struct NODE{
    int cnt; bool vis; NODE *next[goes];
    vector<NODE *> out;
    NODE(){
        for(int i = 0; i < goes; i++)
            next[i] = NULL;
        out.clear(); vis = false; cnt = 0;
    }
    ~NODE(){
        for(int i = 1; i < goes; i++)
            if(next[i]!=NULL && next[i] != this)
                delete next[i];
    }
};
NODE *root;
void buildtrie(){
    root = new NODE();
    for(int i = 0; i < n; i++){
        NODE *p = root;
        for(int j = 0; dictionary[i][j]; j++){
            char c = dictionary[i][j]; /// - 'a' + 1; ///uncomment it for only lowercase letters
            if(!p->next[c])
                p->next[c] = new NODE();
            p = p->next[c];
        }
    }
    queue<NODE *> q;
    for(int i = 0; i < goes; i++){
        if(!root->next[i]) root->next[i] = root;
        else{
            q.push(root->next[i]);
            root->next[i]->next[0] = root;  /// next[0] is back pointer
        }
    }
    while(!q.empty()){
        NODE *u = q.front(); q.pop();
        for(int i = 1; i < goes; i++){
            if(u->next[i]){
                NODE *v = u->next[i];
                NODE *w = u->next[0];
                while(!w->next[i])
                    w = w->next[0];
                v->next[0] = w = w->next[i];
                w->out.pb(v); q.push(v);
            }
        }
    }
}
void aho_corasick(){
    NODE *p = root;
    for(int i = 0; text[i]; i++){
        char c = text[i]; /// - 'a' + 1; ///uncomment it for only lowercase letters
        while(!p->next[c])
            p = p->next[0];
        p = p->next[c];
        p->cnt++;
    }
}
int DFS(NODE *p){
    if(p->vis) return p->cnt;
    for(int i = 0; i < (int)p->out.size(); i++)
        p->cnt += DFS(p->out[i]);
    p->vis = true;
    return p->cnt;
}
int main(){
    sc("%s",text); scin(n);
    for(int i = 0; i < n; i++){
        sc("%s",dictionary[i]);
    }
    buildtrie(); aho_corasick();
    for(int i = 0; i < n; i++){
        NODE *p = root;
        for(int j = 0; dictionary[i][j]; j++){
            char c = dictionary[i][j];
            p = p->next[c];
        }
        int x = DFS(p);
        if(x==0) pf("N\n");
        else pf("Y\n");
```

```
    }
    delete root;
    return 0;
}
/** Ternary Search Template **/
double ternary_search(double l, double r) {
    double eps = 1e-9;  ///set the error limit here
    while (r - l > eps) {
        double m1 = l + (r - l) / 3;
        double m2 = r - (r - l) / 3;
        double f1 = f(m1);  ///evaluates the function at m1
        double f2 = f(m2);  ///evaluates the function at m2
        if (f1 < f2)
            l = m1;
        else
            r = m2;
    }
    return f(l);    ///return the maximum of f(x) in [l, r]
}
/** Binary Search + Meet In the Middle + Backtracking **/
/** For each coin we've three coins. We can take this coin for 0,1 or 2 times **/
/** So,we'll divide the coins into two portion & generating all possible permutation of both portion **/
/** Then,sort 2nd portion B. And search for every x-A[i] in the 2nd portion. If yes the ans will be yes or No otherwise **/
int coin[20]; vi A;
map<int,int>B; int n;
void SubSet(int idx,int val){
    if(idx == (n/2)){
        B[val]++;
        return;
    }
    SubSet(idx+1,val);
    SubSet(idx+1,val+coin[idx]);
    SubSet(idx+1,val+(2*coin[idx]));
}
void SubSet2(int idx,int val){
    if(idx == n){
        A.pb(val);
        return;
    }
    SubSet2(idx+1,val);
    SubSet2(idx+1,val+coin[idx]);
    SubSet2(idx+1,val+(2*coin[idx]));
}
int main(){
    int i,j,k,t,T,mx,temp,cnt;
    scin(T);
    RUN_CASE(t,T){
        A.clear(); B.clear(); scin2(n,k);
        for(i=0 ; i<n ; i++) scin(coin[i]);
        SubSet(0,0); SubSet2(n/2,0);
        bool fg=0;
        for(i=0 ; i<A.size() && !fg ; i++){
            temp = k-A[i];
            if(B[temp] > 0) fg=1;
        }
        if(fg)
            pf("Case %d: Yes\n",t);
        else
            pf("Case %d: No\n",t);
    }
    return 0;
}
/** Matrix Exponential Template **/
struct Matrix{
    ll v[5][5]; /** Sizes may vary **/ ll row, col;
    Matrix operator = (const Matrix X){
        row = X.row; col = X.col;
        for(ll i = 0; i < row; i++){
            for(ll j = 0; j < col; j++)
                v[i][j] = X.v[i][j];
        }
    }
};  ll mod;
Matrix multiply(const Matrix &A, const Matrix &B){
    assert(A.col==B.row);
    Matrix R;
    R.row = A.row; R.col = B.col;
    for(ll i = 0; i < R.row; i++){
        for(ll j = 0; j < R.col; j++){
            ll sum = 0;
            for(ll k = 0; k < A.col; k++){
                sum += (A.v[i][k] * B.v[k][j]);
                sum %= mod;
            }
            R.v[i][j] = sum;
        }
    }
    return R;
}
Matrix power(const Matrix &M, ll p){
    assert(p >= 1);
    if(p == 1) return M;
    if(p%2 == 1) return multiply(M, power(M, p-1));
    Matrix ret = power(M, p / 2);
    Matrix bet = multiply(ret,ret);
    return bet;
}
int main(){
    int t,T;
    Matrix base;
    base.row = 2LL; base.col = 2LL;
    base.v[0][0] = base.v[0][1] = base.v[1][0] = 1LL;
    base.v[1][1] = 0; scin(T);
    RUN_CASE(t,T){
        ll a,b,n,m;
        scln(a); scln(b); scln(n); scln(m);
        mod = POW(10LL,m);
        Matrix MAT, Res, M;
        MAT.row = 2LL; MAT.col = 1LL;
        MAT.v[0][0] = b; MAT.v[1][0] = a;
        M = power(base, n - 1); Res = multiply(M, MAT);
        CASE(t); pf("%lld\n",Res.v[0][0]);
    }
    return 0;
}
/** Solve of a given mathematical expression **/
map<char, int> M;
int main(){
    M['-'] = 1; M['+'] = 1; M['*'] = 3; M['/'] = 4; M['#'] = 0;
    int t,T; scin(T);
    RUN_CASE(t,T){
        string str,tmp; cin>>str;
        str+='#'; tmp = "";
        stack<char> chr; stack<ll> num;
        str = "0" + str;
        stringstream ss;
        int n = (int)str.size();
        for(int i = 0; i < n; i++){
            if(str[i]=='+'||str[i]=='-'||str[i]=='*'||str[i]=='/'||str[i]=='#'){
                ss<<tmp; ll x; ss>>x; tmp = ""; ss.clear();
                if(chr.empty() || M[str[i]] > M[chr.top()]){
                    num.push(x);
                    chr.push((char)str[i]);
                }
                else{
                    while(!chr.empty() && M[str[i]] <= M[chr.top()]){
                        ll y = num.top();
                        int z = M[chr.top()];
                        char ch = chr.top();
                        num.pop();
                        chr.pop();
                        if(ch=='-') y = y-x;
                        else if(ch=='+') y = y+x;
                        else if(ch=='*') y
```

```
= x * y;
                        else if(ch=='/') y
= y/x;
                        x = y;
                    }
                    num.push(x);
chr.push((char)str[i]);
                }
            }
            else tmp += ((char)str[i]);
        }
        CASE(t); printf("%lld\n",num.top());
    }
    return 0;
}
/** 10+2*3,100*2+500/10,10+20/2-5   */
/** MOD INVERSE Template **/
#define x first
#define y second
pii egcd(int a,int b)
{
    if(b == 0) return pii(1,0);
    else{
        pii d = egcd(b , a%b);
        return pii(d.y , d.x-d.y*(a/b));
    }
}
int modInv(int a,int b){
    pii ret = egcd()
    return ;
}
/**COIN CHANGE 1 **/
/**In a strange shop there are n types of
coins of value A1, A2 ... An. C1, C2, ...
Cn denote the number of coins of value A1,
A2 ... An respectively. You have to find
the number of ways you can make K using
the coins.**/
int main()
{
    int n,current_limit, ans,coin[105],
t, T, k, limit[105],tmp,tmp1;
```

```
        scin(T);int combination[1005];
        RUN_CASE(t,T){
            ms(coin,0); ms(limit,0);
            scin(n); scin(k);
            for(int i=0; i<n; i++)
scin(coin[i]);
            for(int i=0; i<n; i++)
scin(limit[i]);
            CASE(t);
            ms(combination,0); combination[0]
= 1;
            for(int i=0; i<n; i++){
                current_limit = limit[i];
                for(int j = k; j >= 0; j--){
                    for(int x = 1; x <=
current_limit; x++){
                        if((j-x*coin[i])>=0)
combination[j]+= combination[ j -
(x*coin[i])];
                    }
                }
                for(int j=0;j<=k;j++)
combination[j]%=MOD;
            }
            pf("%d\n",combination[k]);
        }return 0;
}
/**COIN CHANGE 2 **/
/**In a strange shop there are n types of
coins of value A1, A2 ... An. You have to
find the number of ways you can make K
using the coins. You can use any coin at
most K times.**/
ll combination[10002] ,tmp, tmp1;
int main()
{
    int t, T, n, k, coin[102];
    scin(T);
    RUN_CASE(t,T){
        scin(n);scin(k);
        combination[0] = 1;
        for(int i = 0; i < n; i++){
```

```
            scin(coin[i]);
            for(int j = coin[i]; j <= k ;
j++){
                if(coin[i]<=j){
                    tmp = combination[ j
- coin[i] ]%MOD;
                    tmp1 = combination[j]
+ tmp;
                    combination[j]
=tmp1%MOD;
                }}}
        CASE(t);
        pf("%lld\n", combination[k]);
        ms(combination,0);
    }return 0;
}
/**COIN CHANGE 3 **/
/**In a strange shop there are n types of
coins of value A1, A2 ... An. C1, C2, ...
Cn denote the number of coins of value A1,
A2 ... An respectively. You have to find
the number of different values (from 1 to
m), which can be produced using these
coins**/
int main()
{
    int n,current_limit, ans;
    int coin[105],cnt[100005];
    int t, T, k, limit[105],tmp,tmp1;
    scin(T);int combination[100005];
    RUN_CASE(t,T){
        ms(coin,0); ms(limit,0);
        scin(n); scin(k);
        for(int i=0; i<n; i++)
scin(coin[i]);
        for(int i=0; i<n; i++)
scin(limit[i]);
        CASE(t);
        combination[0] = 1; ans = 0;
        for(int i=0; i<n; i++){
            current_limit = limit[i];
ms(cnt,0);
```

```
            for(int j = coin[i]; j <= k;
j++){

if(cnt[j-coin[i]]<current_limit &&
combination[j]==0 &&
combination[j-coin[i]]){
                    ans++;
                    cnt[j] =
cnt[j-coin[i]]+1;
                    combination[j] = 1;
                }}}
        pf("%d\n",ans);
        ms(combination,0);
    }return 0;
}
/**nCr using inverse mod **//// x must be
relatively prime to p
const int Maxn = 1e5 + 1;
template<typename T, typename T1>
T mod(T x, T1 p){
    x %= p;
    if (x < 0) x += p;
    return x;
}
int fact[Maxn], inv[Maxn], ifact[Maxn];
void factorial(){
    fact[0] = 1;
    for(int i = 1; i < Maxn; i++){
        fact[i] = 1LL * fact[i - 1] * i %
MOD;
    }
}
template<typename T>
T inverse(T x, T p){
    x = mod(x, p);
    if (x == 1) return x;
    return mod((1LL * (-p / x) * (inv[p %
x] % p)), p);
    /// Since inverse of p % x is already
calculated.
}
void inverse_fact(){
```

```
3062        ifact[0] = 1;
3063        for(int i = 1; i < Maxn; i++){
3064            inv[i] = inverse(i, MOD);
3065            ifact[i] = (1LL * ifact[i - 1] *
    inv[i]) % MOD;
3066        }
3067    }
3068    int nCr(int n, int r){
3069        int ret = (1LL * ifact[n - r] *
    ifact[r]) % MOD ;
3070        ret = (1LL * ret * fact[n]) % MOD;
3071        return ret;
3072    }
3073    int main(){
3074        factorial(); inverse_fact();
3075        while(t--){
3076            cin >> n >> r;
3077            cout << nCr(n,r) << endl;
3078        }return 0;
3079    }
3080    /** Default Template **/
3081    #define Input
        freopen("in.txt","r",stdin)
3082    #define Output
        freopen("out.txt","w",stdout)
3083    #define ll                  long long int
3084    #define ull                 unsigned long
        long int
3085    #define pii                 pair<int,int>
3086    #define pll                 pair<ll,ll>
3087    #define sc                  scanf
3088    #define scin(x)             sc("%d",&(x))
3089    #define scin2(x,y)          sc("%d
        %d",&(x),&(y))
3090    #define scln(x)             sc("%lld",&(x))
3091    #define scln2(x,y)          sc("%lld
        %lld",&(x),&(y))
3092    #define pf                  printf
3093    #define all(a)
        (a.begin()),(a.end())
3094    #define UNIQUE(X)
        (X).erase(unique(all(X)),(X).end())
3095    #define SORT_UNIQUE(c)
        (sort(c.begin(),c.end()),
        c.resize(distance(c.begin(),unique(c.begin()
        ,c.end()))))
3096    #define ms(a,b)
        memset(a,b,sizeof(a))
3097    #define pb(a)               push_back(a)
3098    #define mp                  make_pair
3099    #define db                  double
3100    #define ff                  first
3101    #define ss                  second
3102    #define sqr(x)              (x)*(x)
3103    #define vi                  vector<int>
3104    #define vl                  vector<ll>
3105    #define CIN
        ios_base::sync_with_stdio(0);cin.tie(0);cout
        .tie(0)
3106    #define RUN_CASE(t,T)       for(__typeof(t)
        t=1;t<=T;t++)
3107    #define CASE(t)             printf("Case %d:
        ",t)
3108    #define CASEl(t)            printf("Case
        %d:\n",t)
3109    #define intlimit            2147483690
3110    #define longlimit           92233720368547758
3111    #define infinity            (1<<28)
3112    #define gcd(a,b)            __gcd(a,b)
3113    #define lcm(a,b)            ((a)*(b))/gcd(a,b)
3114    #define mxx                 123456789
3115    #define PI                  2*acos(0.0)
3116    #define rep(i,a,b)          for(__typeof(i)
        i=a;i<=b;i++)
3117    using namespace std;
3118    ///------------------Graph
        Moves-------------------
3119    ///const int fx[] = {+1,-1,+0,+0};
3120    ///const int fy[] = {+0,+0,+1,-1};
3121    ///const int fx[] =
        {+0,+0,+1,-1,-1,+1,-1,+1}; ///King's move
3122    ///const int fy[] =
        {-1,+1,+0,+0,+1,+1,-1,-1}; ///King's move
3123    ///const int fx[] =
        {-2,-2,-1,-1,+1,+1,+2,+2}; ///Knight's move
3124    ///const int fy[] =
        {-1,+1,-2,+2,-2,+2,-1,+1}; ///Knight's move
3125
```