|  |  |
| --- | --- |
|  | **2018** |
|  | COOK BOOK  NOBEL  DOUBLE COLUMN |

|  |
| --- |
| **[TYPE THE DOCUMENT TITLE]** |
| [Type the abstract of the document here. The abstract is typically a short summary of the contents of the document. Type the abstract of the document here. The abstract is typically a short summary of the contents of the document.] |

Template:

#include <bits/stdc++.h>

using namespace std;

/\*\*\*template\*\*\*/

#define ll long long

#define LL long long

#define dd double

#define scl(n) scanf("%lld",&n)

#define sl(n) scanf("%lld",&n)

#define scd(n) scanf("%lf",&n)

#define pi pair<ll,ll>

#define pll pair<ll,ll>

#define pb push\_back

#define mp make\_pair

#define uu first

#define vv second

#define fr first

#define sc second

#define FOR(i,n) for(ll i=1;i<=n;i++)

#define LOOP(i,n) for(ll i=0;i<n;i++)

#define FOOR(i,a,b) for(ll i=a;i<=b;i++)

#define LOP(i,a,b) for(ll i=a;i<b;i++)

#define sorted(s) sort(s.begin(),s.end())

#define sortedd(s) sort(s.begin(),s.end(),cmp)

#define reversed(s) reverse(s.begin(),s.end())

#define contains(a,b) (find((a).begin(), (a).end(), (b)) != (a).end())

#define MAXN 100005

#define pii 3.1415926536

#define mod 1000000007

#define eps 0.0000000001

#define inf 9000000000000000

#define INF (1LL<<62)

#define mstt(a,b) memset((a),(b),sizeof (a))

#define sz(x) (ll)x.size()

#define mx 92681+10

struct node

{

ll nod,d;

node(ll a,ll b)

{

nod=a;

d=b;

}

bool operator<(const node &p)const

{

return p.d<d;

}

};

ll Set(ll n,ll pos)

{

return n=n|(1LL<<pos);

}

ll reset(ll n,ll pos)

{

return n=n&~(1LL<<pos);

}

bool check(ll n,ll pos)

{

return (bool)(n&(1LL<<pos));

}

ll power(ll x,ll y)

{

ll res = 1;

x = x;

while (y > 0)

{

if (y & 1)

res = (res\*x);

y = y>>1;

x = (x\*x);

}

return res;

}

/\*\*\*template\*\*\*/

int main()

{

return 0;

}

Aho Corasik:

ll backedge[maxn];///here maxn is the sum of the length of all pattern

ll nextt[maxn][26];

ll sz;

vector<ll>out[maxn];

ll endd[510];

ll cnt[maxn];

bool vis[maxn];

void cle()

{

mstt(backedge,-1);

mstt(nextt,-1);

mstt(cnt,0);

LOOP(i,maxn)

{

out[i].clear();

vis[i]=0;

}

sz=0;

}

void build\_matching\_machine(const vector<string>&words)

{

cle();

LOOP(i,words.size())

{

const string &s = words[i];

ll v = 0;

LOOP(j,s.size())

{

ll id = s[j] - 'a';

if (nextt[v][id] == -1) /// Allocate a new node

{

nextt[v][id] = ++sz;

}

v = nextt[v][id];

}

endd[i]=v;

}

deque<ll>q;

LOOP(i,26)

{

if (nextt[0][i] != -1)

{

backedge[nextt[0][i]] = 0;

out[0].pb(nextt[0][i]);

q.pb(nextt[0][i]);

}

}

while (!q.empty())

{

ll src = q.front();

q.pop\_front();

LOOP(i,26)

{

ll adj=nextt[src][i];

if (adj != -1)

{

ll v = backedge[src];

while (1)

{

if(nextt[v][i]!=-1)

{

v=nextt[v][i];

break;

}

if(v==0)

{

break;

}

v = backedge[v];

}

backedge[adj] = v;

out[v].pb(adj);

q.push\_back(adj);

}

}

}

}

ll tes;

ll n;

vector<string> patterns;

ll ans[510];

string text,pat;

void search()

{

ll v=0;

LOOP(i,text.size())

{

ll id=text[i]-'a';

while(1)

{

if(nextt[v][id]!=-1)

{

v=nextt[v][id];

break;

}

if(v==0)

{

break;

}

v=backedge[v];

}

cnt[v]++;

}

}

ll dfs(ll src)

{

if(vis[src])return cnt[src];

LOOP(i,out[src].size())

{

ll adj=out[src][i];

cnt[src]+= dfs(adj);

}

vis[src]=1;

return cnt[src];

}

int main()

{

\_\_\_

cin>>n;

cin>>text;

patterns.clear();

FOR(i,n)

{

cin>>pat;

patterns.pb(pat);

}

build\_matching\_machine(patterns);

search();

cout<<"Case "<<cas<<":"<<endl;

LOOP(i,n)

{

cout<<dfs(endd[i])<<endl;

}

return 0;

}

BIT TREE:

ll BITree[MAXN+10],arr[MAXN+10],n;

ll query(ll x)

{

ll sum = 0;

while (x>0)

{

sum += BITree[x];

x-=x&(-x);

}

return sum;

}

void update(ll x,ll val)

{

while (x <= n)

{

BITree[x] += val;

x+=x&(-x);

}

}

void build()

{

FOR(i,n)BITree[i] = 0;

LOOP(i,n)update(i+1,arr[i]);/// to update position update(i+1)

}

int main()

{

scl(n);

LOOP(i,n)scl(arr[i]);

build();

ll q;

scl(q);

FOR(i,q)

{

ll typ;

scl(typ);

if(typ==1)///query

{

ll l,r;

scl(l);

scl(r);

if(l==0)printf("%lld\n",query(r+1));

else printf("%lld\n",query(r+1)-query(l));///to find sum[0....x] do query(x+1)

}

else///update

{

ll x,val;

scl(x);

scl(val);

update(x+1,val);

}

}

return 0;

}

DIGIT DP:

include<bits/stdc++.h>

using namespace std;

typedef long long int ll;

typedef unsigned long long int ull;

ll aa,bb,la,lb;

ll dp[20][2][2];

int visited[20][2][2];

char a[20],b[20];

ll cas;

ll k;

ll go(ll pos,ll sf,ll gf)

{

if(pos==lb)

{

return 1;//do something;

}

if(visited[pos][sf][gf]==cas)

return dp[pos][sf][gf];

ll st,ed,ret,i;

if(sf) /// upperlimit theke choto ase kina cheak kortesi,jodi choto thake tahole end hobe 9 otherwise b[pos]

ed = k-1;

else

ed = min(k-1,(ll)b[pos]-'0');

if(gf)

st = 0;

else

st = a[pos]-'0'; ///lower limit theke boro ase kina chek kortesi,jodi boro hoi tahole start hobe 0 otherwise a[pos]

ret = 0;

for(i=st; i<=ed; i++)

ret+=go(pos+1,sf or (i<(b[pos]-'0')),gf or (i>(a[pos]-'0')));

visited[pos][sf][gf]=cas;

return dp[pos][sf][gf] = ret;

}

int main()

{

ll tes;

scanf("%lld",&tes);

for(cas=1;cas<=tes;cas++)

{

scanf("%lld %lld %lld",&aa,&bb,&k);

sprintf(a,"%lld",aa);

sprintf(b,"%lld",bb);

la=strlen(a);

lb=strlen(b);

reverse(a,a+la);

for(ll i=la; i<lb; i++)

a[i]='0';

reverse(a,a+lb);

ll ans = go(0,0,0);

printf("%lld\n",ans);

}

return 0;

}

DINIC:

#include<bits/stdc++.h>

#include<iostream>

#define ll long long int

#define rep(i,a,b) for(int i=a; i<=b; i++)

#define sci(a) scanf("%d",&a);

#define scl(a) scanf("%ld",&a);

#define scd(a) scanf("%lf",&a);

#define all(x) x.begin(),x.end()

#define sz size()

#define mem(x,a) memset(x,a,sizeof(x))

#define Max 1010

using namespace std;

vector<int>graph[Max+1];

int cap[Max+1][Max+1],f[Max+1][Max+1];

int d[Max+1];

int vis[Max+1];

bool bfs(int src,int sink)

{

mem(d,-1);

d[src]=0;

queue<int>q;

q.push(src);

while(!q.empty())

{

int u=q.front();

q.pop();

for(int i=0; i<graph[u].size() ; i++)

{

int v=graph[u][i];

if(d[v]==-1 && (f[u][v]<cap[u][v]))

{

q.push(v);

d[v]=d[u]+1;

}

}

}

return d[sink]!=-1;

}

int ind[Max+1]; /\*\*\* ind is the next operational stariting index of vertex\*\*\*/

int dfs(int src,int flow,int sink)

{

if(!flow) return 0;

if(src==sink) return flow;

for( ; ind[src]<graph[src].sz ; ++ind[src] )

{

int v=graph[src][ind[src]];

if(d[v]!=d[src]+1) continue;

int curr\_flow=dfs(v, min(flow, cap[src][v]-f[src][v]),sink);

if(curr\_flow)

{

f[src][v]+=curr\_flow;

f[v][src]-=curr\_flow;

return curr\_flow;

}

}

return 0;

}

ll dinic(int src,int sink)

{

ll flow=0;

int INF=111111111;

//cout<<"starting ok"<<endl;

while(1)

{

if(!bfs(src,sink)) break;

// cout<<"bfs ok"<<endl;

mem(ind,0);

while(int curr\_flow=dfs(src,INF,sink))

flow+=curr\_flow;

// cout<<"dfs"<<endl;

}

return flow;

}

void Inp()

{

for(int i=0; i<Max; i++)

graph[i].clear();

int node,edge;

mem(cap,0); mem(f,0);

sci(node);

sci(edge);

for(int i=0; i<edge; i++)

{

int u,v,w;

sci(u); sci(v); sci(w);

graph[u].push\_back(v);

graph[v].push\_back(u);

cap[u][v]+=w;

}

}

DP ON KMP :

ll Next[100010][28];

ll next\_j(ll j, ll c)

{

ll &ret = Next[j][c];

if (ret != -1)

return ret;

char ch = c + 'a';

while (j > 0 && ch != t[j])

j = lps[j-1];

if (ch == t[j])

j++;

return ret = j;

}

DSU ON TREE :

vector<ll>graph[MAXN];

ll sz[MAXN],depth[MAXN],big[MAXN];

ll n;

void getsize(ll src,ll par,ll d)

{

depth[src]=d;

sz[src]=1;

ll maxi=-1;

LOOP(i,graph[src].size())

{

ll adj=graph[src][i];

if(adj!=par)

{

getsize(adj,src,d+1);

sz[src]+=sz[adj];

if(sz[adj]>maxi)

{

maxi=sz[adj];

big[src]=adj;

}

}

}

}

bool is\_big[500005];

void add(ll src,ll par,ll val)

{

LOOP(i,graph[src].size())

{

ll adj=graph[src][i];

if(!is\_big[adj] && adj!=par)

{

add(adj,src,val);

}

}

}

void dfs(ll src,ll par,bool keep)

{

LOOP(i,graph[src].size())

{

ll adj=graph[src][i];

if(adj!=big[src] && adj!=par)

{

dfs(adj,src,0);

}

}

if(big[src]!=-1)

{

dfs(big[src],src,1);

is\_big[big[src]]=1;

}

add(src,par,1);

if(big[src]!=-1)is\_big[big[src]]=0;

if(keep==0)add(src,par,-1);

}

void cle()

{

FOR(i,n)

{

graph[i].clear();

big[i]=-1;

}

}

int main()

{

scl(n);

cle();

FOR(i,n-1)

{

ll u,v;

scl(u);

scl(v);

graph[u].pb(v);

graph[v].pb(u);

}

getsize(1,-1,0);

dfs(1,-1,1);

return 0;

}

EULAR PHI :

for(ll i=1; i<MAXN; i++)/// find the template at some website

{

phi[i]=i;

}

for(ll i=2; i<MAXN; i++)

{

if(phi[i]==i)

{

for(ll j=i; j<MAXN; j+=i)

{

phi[j]/=i;

phi[j]\*=i-1;

}

}

}

KMP :

#include <bits/stdc++.h>

using namespace std;

#define ll long long

#define LOOP(i,n) for(ll i=0;i<n;i++)

int lps[1000005];

void make\_lps(string &s)

{

ll i=1;

ll len=0;

lps[0]=0;

ll len2=s.size();

while(i<len2)

{

if(s[i]==s[len])

{

lps[i]=++len;

i++;

}

else if(s[i]!=s[len])

{

if(len>0)

{

len=lps[len-1];

}

else

{

lps[i]=0;

i++;

}

}

}

}

string text,pattern;

int compute()

{

ll cnt=0;

ll i=0;

ll j=0;

ll len=text.size();

while(i<len)

{

if(pattern[j]==text[i])

{

i++;

j++;

}

else if(pattern[j]!=text[i])

{

if(j>0)

{

j=lps[j-1];

}

else

{

i++;

}

}

if(j==pattern.size())

{

cout<<i-(pattern.size())+1<<" "<<i<<endl;

j=lps[j-1];

}

}

return cnt;

}

int main()

{

cin>>text;

cin>>pattern;

make\_lps(pattern);

ll ret=compute();

cout<<ret<<endl;

return 0;

}

MO :

#include <bits/stdc++.h>

using namespace std;

/\*\*\*template\*\*\*/

#define ll long long

#define scl(n) scanf("%lld",&n)

#define pi pair<ll,ll>

#define pb push\_back

#define uu first

#define vv second

#define FOR(i,n) for(ll i=1;i<=n;i++)

#define LOOP(i,n) for(ll i=0;i<n;i++)

#define FOOR(i,a,b) for(ll i=a;i<=b;i++)

#define LOP(i,a,b) for(ll i=a;i<b;i++)

#define MAXN 209990

#define mstt(a,b) memset((a),(b),sizeof (a))

#define BLOCK 555

/\*\*\*template\*\*\*/

ll arr[MAXN],cnt[MAXN];

ll l,r,sum,ans[MAXN];

struct query

{

ll L, R, id;

} q[MAXN];

bool cmp(query x,query y)

{

if(x.L/BLOCK != y.L/BLOCK)

{

return x.L/BLOCK < y.L/BLOCK;

}

return x.R < y.R;

}

ll N,Q;

void cle()

{

mstt(cnt,0);

l=0;

r=-1;

sum=0;

}

void in\_querys()

{

LOOP(i,Q)

{

scl(q[i].L);

scl(q[i].R);

q[i].L--;

q[i].R--;

q[i].id=i;

}

sort(q,q+Q,cmp);

}

void add(ll pos)

{

}

void del(ll pos)

{

}

int main()

{

scl(N);

scl(Q);

cle();

in\_querys();

LOOP(i,Q)

{

while(l>q[i].L)add(--l);

while(r<q[i].R)add(++r);

while(l<q[i].L)del(l++);

while(r>q[i].R)del(r--);

ans[q[i].id]=sum;

}

LOOP(i,Q)

{

printf("%lld\n",ans[i]);

}

return 0;

}

LCA AND TREE LINEAR :

ll starting\_time[maxi];

ll finishing\_time[maxi];

ll level[maxi];

ll parent[17][maxi];

ll start\_time[2\*maxi];

ll end\_time[2\*maxi];

vector<ll>graph[maxi];

ll timee,n;

void in\_clear()

{

for(ll i=0; i<=n; i++)

{

starting\_time[i]=0;

finishing\_time[i]=0;

graph[i].clear();

for(ll j=0;j<17;j++)

{

parent[j][i]=-1;

}

}

for(ll i=1; i<=2\*n; i++)

{

start\_time[i]=0;

end\_time[i]=0;

}

timee=1;

}

void bfs0(ll par,ll source,ll count1)

{

starting\_time[source]=timee;

timee++;

level[source]=count1;

parent[0][source]=par;

for(ll i=0; i<graph[source].size(); i++)

{

if(graph[source][i]!=par)

{

bfs0(source,graph[source][i],count1+1);

}

}

finishing\_time[source]=timee;

timee++;

}

void lca\_init()

{

for(ll i=1; i<17; i++)

{

for(ll j=0; j<n; j++)

{

if(parent[i-1][j]!=-1)

{

parent[i][j]=parent[i-1][parent[i-1][j]];

}

}

}

}

ll lca(ll u, ll v)

{

if(level[u] < level[v]) swap(u,v);

ll diff = level[u] - level[v];

for(ll i=0; i<17; i++)

if( (diff>>i)&1 )

u = parent[i][u];

if(u == v) return u;

for(ll i=17-1; i>=0; i--)

if(parent[i][u] != parent[i][v])

{

u = parent[i][u];

v = parent[i][v];

}

return parent[0][u];

}

void making\_starting\_and\_finishing\_array()

{

for(ll i=0; i<n; i++)

{

start\_time[starting\_time[i]]=value[i];

end\_time[finishing\_time[i]]=value[i];

}

// for(ll i=1;i<=2\*n;i++)

// {

// printf("index %d start\_time %d fin time %d\n",i,start\_time[i],end\_time[i]);

// }

}

PALINDROME GENERATOR :

ll createPalindrome(ll input,bool isOdd)

{

ll n = input;

ll palin = input;

if(isOdd)

n /= b;

while (n > 0)

{

palin = palin \* b + (n % b);

n /= b;

}

return palin;

}

void generatePaldindromes(ll n)

{

ll number;

for (ll j = 0; j < 2; j++)

{

int i = 1;

while ((number = createPalindrome(i, 10, j % 2)) < n)

{

i++;

}

}

}

int main()

{

ll n = 104;

generatePaldindromes(n);

return 0;

}

SPARSE TABLE :

ll arr[MAXN+10];

ll sparse[MAXN+10][26];

ll close[MAXN+10];

ll n;

void lca\_init()

{

FOR(i,n)sparse[i][0]=arr[i];

for(ll j=1;(1LL<<j)<=n;j++)

{

for(ll i=1;(i+(1LL<<j)-1)<=n;i++)

{

sparse[i][j]=min(sparse[i][j-1],sparse[i + (1LL<<(j - 1))][j - 1]);

}

}

close[1] = 0;

for(ll i=2;i<=MAXN;i++)

{

close[i]=close[i - 1];

if ((1LL<<(close[i]+1))==i)

close[i]++;

}

}

ll query(ll l,ll r)

{

ll diff=close[r-l+1];

return min(sparse[l][diff],sparse[r-(1LL<<diff)+1][diff]);

}

int main()

{

scl(n);

FOR(i,n)scl(arr[i]);

lca\_init();

cout<<query(6,6)<<endl;

return 0;

}

STRONGLY CONNECTED COMPONENT :

#include <bits/stdc++.h>

using namespace std;

#define ll long long

vector<ll>graph[100];

vector<ll>reverse\_graph[100];

vector<ll>components[100];

stack<ll>finished\_nodes;

ll visited[100];

ll colour[100];

ll nodes,edges,node1,node2,mark;

void cle()

{

for(ll i=0;i<=nodes;i++)

{

graph[i].clear();

reverse\_graph[i].clear();

components[i].clear();

}

}

void in\_edges()

{

for(ll i=0;i<edges;i++)

{

scanf("%lld%lld",&node1,&node2);

graph[node1].push\_back(node2);

reverse\_graph[node2].push\_back(node1);

}

}

void clear\_stack()

{

while(!finished\_nodes.empty())

{

finished\_nodes.pop();

}

}

void clear\_colour\_vis()

{

for(ll i=1;i<=nodes;i++)

{

visited[i]=-1;

colour[i]=-1;

}

}

void dfs(ll source)

{

ll adj;

colour[source]=1;

for(ll i=0;i<graph[source].size();i++)

{

adj=graph[source][i];

if(colour[adj]==-1)

{

dfs(adj);

}

}

finished\_nodes.push(source);

}

void dfs2(ll source,ll mark)

{

ll adj;

visited[source]=1;

for(ll i=0;i<reverse\_graph[source].size();i++)

{

adj=reverse\_graph[source][i];

if(visited[adj]==-1)

{

dfs2(adj,mark);

}

}

components[mark].push\_back(source);

}

void find\_scc()

{

clear\_stack();

clear\_colour\_vis();

for(ll i=1;i<=nodes;i++)

{

if(colour[i]==-1)

{

dfs(i);

}

}

while(!finished\_nodes.empty())

{

cout<<finished\_nodes.top()<<endl;

finished\_nodes.pop();

}

ll source;

mark=1;

while(!finished\_nodes.empty())

{

source=finished\_nodes.top();

finished\_nodes.pop();

if(visited[source]==-1)

{

dfs2(source,mark);

mark++;

}

}

}

int main()

{

scanf("%lld%lld",&nodes,&edges);

cle();

in\_edges();

find\_scc();

bool f=0;

for(ll i=1;i<mark;i++)

{

if(components[i].size()>1)

{

f=1;

break;

}

}

if(f)

{

}

return 0;

}

CENTROID TREE :

vector<pi>graph[MAXN];

ll son\_size[MAXN],mxson[MAXN];

bool vis[MAXN];

ll n,root;

ll parent[MAXN];

void cle()

{

FOR(i,n)

{

graph[i].clear();

vis[i]=0;

}

}

void in\_edges()

{

ll u,v,c;

FOR(i,n-1)

{

scl(u);

scl(v);

scl(c);

graph[u].pb(mp(v,c));

graph[v].pb(mp(u,c));

}

}

void dfssize(ll src,ll par)

{

son\_size[src]=1;

mxson[src]=0;

LOOP(i,graph[src].size())

{

ll adj=graph[src][i].uu;

if(adj!=par && vis[adj]==0)

{

dfssize(adj,src);

son\_size[src]+=son\_size[adj];

mxson[src]=max(mxson[src],son\_size[adj]);

}

}

}

void dfsroot(ll main\_source,ll source,ll par)

{

if(son\_size[main\_source]-son\_size[source]>mxson[source])

{

mxson[source]=son\_size[main\_source]-son\_size[source];

}

if(mxson[source]<mxson[root])

{

root=source;

}

LOOP(i,graph[source].size())

{

ll adj=graph[source][i].uu;

if(adj!=par && vis[adj]==0)

{

dfsroot(main\_source,adj,source);

}

}

}

ll dfs(ll source)

{

dfssize(source,-1);

root=0;

mxson[0]=n;

dfsroot(source,source,-1);

vis[root]=1;

ll now\_root=root;

LOOP(i,graph[now\_root].size())

{

ll adj=graph[now\_root][i].uu;

ll adj\_cost=graph[now\_root][i].vv;

if(vis[adj]==0)

{

parent[dfs(adj)]=now\_root;

}

}

return now\_root;

}

int main()

{

scl(n);

cle();

in\_edges();

dfs(1);

return 0;

}

Segmented sieve :

#include <bits/stdc++.h>

using namespace std;

#define ll long long

#define maxi 46350

vector<ll>segment\_prime;

vector<ll>sq\_prime;

bool status[maxi];

void normal\_sieve()

{

for(ll i=2; i<maxi; i++)

{

if(status[i]==0)

{

for(ll j=i\*2; j<maxi; j+=i)

{

status[j]=1;

}

}

}

for(ll i=2; i<maxi; i++)

{

if(status[i]==0)

{

sq\_prime.push\_back(i);

}

}

}

void segmentedsieve(ll lower,ll upper)

{

ll limit=upper-lower+2;

bool mark[limit+1];

memset(mark,true,sizeof(mark));

ll upper\_prime=floor(sqrt(upper));

for(ll i=0;i<sq\_prime.size()&&sq\_prime[i]<=upper\_prime;i++)

{

ll now\_prime=sq\_prime[i];

ll lower\_limit\_in\_range=floor(lower/now\_prime)\*now\_prime;

if(lower\_limit\_in\_range<lower)

{

lower\_limit\_in\_range+=now\_prime;

}

if(lower\_limit\_in\_range==now\_prime)

{

lower\_limit\_in\_range+=now\_prime;

}

for(ll j=lower\_limit\_in\_range;j<upper;j+=now\_prime)

{

mark[j-lower]=false;

}

}

for(ll i=lower;i<upper;i++)

{

if(mark[i-lower]==1)

{

segment\_prime.push\_back(i);

}

}

// for(ll i=0;i<segment\_prime.size();i++)

// {

// cout<<segment\_prime[i]<<endl;

// }

}

int main()

{

normal\_sieve();

ll tes,left,right;

scanf("%lld",&tes);

for(ll i=1; i<=tes; i++)

{

scanf("%lld%lld",&left,&right);

segment\_prime.clear();

segmentedsieve(left,right+1);

printf("Case %lld: %d\n",i,segment\_prime.size());

}

return 0;

}

Bitmask sieve :

#include <iostream>

#include <stdio.h>

#include <math.h>

#include <algorithm>

using namespace std;

#define size 100000100

long long a[size / 64 + 100];

unsigned prime[5761558];

unsigned int dp[5761558];

int cnt;

int sieve\_with\_bit\_masking()

{

for (long long i = 3; i <= sqrt(size); i += 2)

{

if(!(a[i/64]&(1LL<<(i%64))))

{

for(long long j = i \* i; j <= size; j += 2 \* i)

{

a[j/64] |= (1LL<<(j%64));

}

}

}

prime[cnt++] = 2;

for (long long i = 3; i <= size; i += 2)

{

if(!(a[i / 64] & (1LL << (i % 64))))

{

prime[cnt++] = i;

}

}

}

TRIE :

#include <bits/stdc++.h>

using namespace std;

#define ll long long

#define maxn 5000050

struct Trie

{

ll nextt[maxn][26];

ll endmark[maxn];

ll sz;

Trie()

{

clear();

}

void clear()

{

sz=0;

memset(nextt[0],-1,sizeof(nextt[0]));

memset(endmark,0,sizeof endmark);

}

void add(const char\* str)

{

ll v=0;

ll len=strlen(str);

for(ll i = 0; i< len; ++i)

{

int id = str[i] - 'a';

if(nextt[v][id] == -1)

{

nextt[v][id]=++sz;

memset(nextt[sz],-1,sizeof(nextt[sz]));

}

v=nextt[v][id];

}

endmark[v]++;

}

ll dfs(const char\* s)

{

ll v=0;

ll len=strlen(s);

for(ll i=0; i<len ; i++)

{

ll id=s[i]-'a';

if(nextt[v][id]==-1)

{

return 0;

}

v=nextt[v][id];

}

return endmark[v];

}

};

Trie tree;

char s[100005];

int main()

{

tree.clear();

scanf(" %s",s);

tree.add(s);

tree.add(s);

cout<<tree.dfs(s)<<endl;

return 0;

}

PRIME FACTORIZATION (LOGN)

#include "bits/stdc++.h"

using namespace std;

#define MAXN 100001

int spf[MAXN];

// Calculating SPF (Smallest Prime Factor) for every

// number till MAXN.

// Time Complexity : O(nloglogn)

void sieve()

{

spf[1] = 1;

for (int i=2; i<MAXN; i++)

spf[i] = i;

for (int i=4; i<MAXN; i+=2)

spf[i] = 2;

for (int i=3; i\*i<MAXN; i++)

{

if (spf[i] == i)

{

for (int j=i\*i; j<MAXN; j+=i)

if (spf[j]==j)

spf[j] = i;

}

}

}

vector<int> getFactorization(int x)

{

vector<int> ret;

while (x != 1)

{

ret.push\_back(spf[x]);

x = x / spf[x];

}

return ret;

}

**Bit :**

struct bit

{

ll MAXN;

vector<lll>BIT;

void build()

{

for(ll i=0;i<MAXN;i++)BIT.push\_back(0LL);

}

void update(ll x,ll v)

{

for(; x<MAXN; x+=x&-x)

{

BIT[x]+=v;

}

}

ll query(ll x)

{

lll sum = 0;

for(; x > 0; x -= x & -x)

sum+=BIT[x];

return sum;

}

void update(ll l,ll r,ll v)

{

update(l,v);

update(r+1,-v);

}

}everylevel[maxn];

**Divide and Conqueror optimization:**

void fill\_dp(ll rem,ll l,ll r,ll p1,ll p2)

{

if(l>r)

{

return ;

}

ll mid=(l+r)/2;

ll &a=dp[mid][rem];

ll &b=p[mid][rem];

for(ll i=p1;i<=p2;i++)

{

ll now\_cost=dp[i][rem-1]+cost(i+1,mid);

if(i==p1)

{

a=now\_cost;

b=i;

}

else if(now\_cost<a)

{

a=now\_cost;

b=i;

}

}

fill\_dp(rem, l, mid-1, p1, p[mid][rem]);

fill\_dp(rem, mid+1, r, p[mid][rem], p2);

}

int main()

{

scl(n);

scl(k);

FOR(i,n)

{

scl(arr[i]);

}

for(ll i=1; i<=n; i++)

{

dp[i][1]=cost(1,i);

}

for(ll i=2;i<=k;i++)

{

fill\_dp(i,1,n,1,n);

}

cout<<dp[n][k]<<endl;

return 0;

}

# Dynamic trie :

#define lim 200010

/// NODE = trie nodes

struct NODE

{

LL f,num;

NODE \*nxt[2];

NODE()

{

f = num = 0;

nxt[0] = nxt[1] = NULL;

}

};

NODE \*roots[lim]; /// Head of each state

LL a[lim]; /// information array

void Insert( LL pos )

{

roots[pos] = new NODE;

NODE \*p = roots[pos];

p->nxt[0] = (roots[pos-1])->nxt[0];

p->nxt[1] = (roots[pos-1])->nxt[1];

for( LL i = 20 ; i>=0 ; i-- )

{

LL b = CHK(a[ pos ],i)?1:0;

if( p->nxt[b] == NULL )

{

p->nxt[b] = new NODE;

}

else

{

NODE \*q = new NODE;

q->f = (p->nxt[b])->f;

q->nxt[0] = (p->nxt[b])->nxt[0];

q->nxt[1] = (p->nxt[b])->nxt[1];

p->nxt[b] = q;

}

p = p->nxt[b];

(p->f)++;

}

p->num = a[ aof[pos] ];

}

/// Finds maximum a[i] xor K

LL Search( LL L, LL R, LL K )

{

NODE \*LP = roots[L-1], \*RP = roots[R];

for( LL i = 20 ; i>=0 ; i-- )

{

LL b = CHK(K,i)?1:0;

LL lagbe = b^1;

if( LP == NULL || LP->nxt[lagbe] == NULL )

{

if( RP->nxt[lagbe] != NULL )

{

RP = RP->nxt[lagbe];

LP = NULL;

}

else

{

if(LP!=NULL)

LP = LP->nxt[b];

RP = RP->nxt[b];

}

}

else

{

if( RP->nxt[lagbe] != NULL )

{

if( (LP->nxt[lagbe])->f < (RP->nxt[lagbe])->f )

{

LP = LP->nxt[lagbe];

RP = RP->nxt[lagbe];

}

else

{

LP = LP->nxt[b];

RP = RP->nxt[b];

}

}

else

{

LP = LP->nxt[b];

RP = RP->nxt[b];

}

}

}

return (RP->num) ^ K;

}

Submask enumeration :

for (int s=m; s; s=(s-1)&m)

**Sos dp:**

**for**(**int** mask = 0; mask < (1<<N); ++mask)

{

dp[mask][-1] = A[mask]; **//handle base case separately (leaf states)**

**for**(**int** i = 0;i < N; ++i)

{

**if**(mask & (1<<i))

dp[mask][i] = dp[mask][i-1] + dp[mask^(1<<i)][i-1];

**else**

dp[mask][i] = dp[mask][i-1];

}

F[mask] = dp[mask][N-1];

}

**Palindromic tree:**

const int MAXN = 105000;

struct node

{

int next[26],len,sufflink,num;

}tree[MAXN];

int len,num,suff;

char s[MAXN];

void add(int pos)

{

int cur = suff, curlen = 0;

int let = s[pos] - 'a';

while (1)

{

curlen = tree[cur].len;

if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos])break;

cur = tree[cur].sufflink;

}

if (tree[cur].next[let])

{

suff = tree[cur].next[let];

return ;

}

suff = ++num;

tree[num].len = tree[cur].len + 2;

tree[cur].next[let] = num;

if (tree[num].len == 1)

{

tree[num].sufflink = 2;

tree[num].num = 1;

return ;

}

while (1)

{

cur = tree[cur].sufflink;

curlen = tree[cur].len;

if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos])

{

tree[num].sufflink = tree[cur].next[let];

break;

}

}

tree[num].num = 1 + tree[tree[num].sufflink].num;

}

void initTree()

{

num = 2;

suff = 2;

tree[1].len = -1;

tree[1].sufflink = 1;

tree[2].len = 0;

tree[2].sufflink = 1;

for(int i=1; i<=len+3; i++)

{

for(int j=0; j<=25; j++)

{

tree[i].next[j]=0;

}

}

}

**Suffix automata:**

struct state

{

ll len,link,Next[26];

bool is\_clone;

vector<ll>inv\_link;

state() {}

state(ll \_len,ll \_link)

{

is\_clone=0;

len=\_len;

link=\_link;

inv\_link.clear();

for(ll i=0; i<=25; i++)Next[i]=-1;

}

};

struct suffix\_automation

{

ll MAXLEN;

vector<state>st;

vector<ll>d,ans,cnt;

ll sz,last;

void clear()

{

st.clear();

d.clear();

for(ll i=1; i<=2\*MAXLEN+3; i++)

{

st.push\_back(state(0,-1));

d.push\_back(-1);

ans.push\_back(-1);

cnt.push\_back(-1);

}

st[0].len=0;

st[0].link=-1;

sz=1;

last = 0;

}

void suffix\_automation\_insert(ll c)

{

ll cur,p,q;

cur = sz++;

st[cur].len = st[last].len + 1;

p = last;

while (p != -1 && st[p].Next[c]==-1)

{

st[p].Next[c] = cur;

p = st[p].link;

}

if (p == -1)st[cur].link = 0;

else

{

q = st[p].Next[c];

if (st[p].len + 1 == st[q].len)st[cur].link = q;

else

{

ll clone = sz++;

st[clone]=st[q];

st[clone].len=st[p].len+1;

st[clone].is\_clone=1;

while (p != -1 && st[p].Next[c] == q)

{

st[p].Next[c] = clone;

p = st[p].link;

}

st[q].link = st[cur].link = clone;

}

}

last = cur;

}

void building\_suffix\_automata(const char\* s)

{

ll len=strlen(s);

for(ll i=0; i<len; i++)suffix\_automation\_insert((ll)(s[i]-'a'));

for (ll v = 1; v < sz; v++)

{

st[st[v].link].inv\_link.push\_back(v);

}

}

ll search(const char \*s)

{

ll v=0;

ll len=strlen(s);

for(ll i=0; i<len; i++)

{

ll now=s[i]-'a';

if(st[v].Next[now]!=-1)v=st[v].Next[now];

else return i;

}

return len;

}

ll solve1(ll v)

{

if(d[v]!=-1)return d[v];

d[v]=1;

for(ll i=0; i<=25; i++)if(st[v].Next[i]!=-1)d[v]+=solve1(st[v].Next[i]);

return d[v];

}

ll solve\_len(ll v)

{

if(ans[v]!=-1)return ans[v];

ans[v]=0;

for(ll i=0; i<=25; i++)if(st[v].Next[i]!=-1)ans[v]+=(solve\_len(st[v].Next[i])+solve1(st[v].Next[i]));

return ans[v];

}

void solve\_k(ll v,ll rem)

{

if(rem==0)

{

printf("\n");

return;

}

ll sum=0;

for(ll i=0; i<=25; i++)

{

if(st[v].Next[i]!=-1)

{

if(sum+solve1(st[v].Next[i])<rem)sum+=solve1(st[v].Next[i]);

else

{

printf("%c",i+'a');

solve\_k(st[v].Next[i],rem-sum-1);

break;

}

}

}

}

ll solve\_occur(ll v)

{

if(cnt[v]!=-1)return cnt[v];

if(v==0 || st[v].is\_clone==1)cnt[v]=0;

else cnt[v]=1;

LOOP(i,st[v].inv\_link.size())

{

cnt[v]+=solve\_occur(st[v].inv\_link[i]);

}

return cnt[v];

}

ll cnt\_occur(const char \*s)

{

ll v=0;

ll len=strlen(s);

for(ll i=0; i<len; i++)

{

ll now=s[i]-'a';

if(st[v].Next[now]!=-1)v=st[v].Next[now];

else return 0;

}

return solve\_occur(v);

}

};

suffix\_automation tr;

char str[100005];

int main()

{

scanf(" %s",str);

tr.MAXLEN=strlen(str);

tr.clear();

tr.building\_suffix\_automata(str);

cout<<tr.solve\_len(0)<<endl;

// for(ll i=1; i<=12; i++)tr.solve\_k(0,i);

scanf("%s",str);

cout<<tr.cnt\_occur(str)<<endl;

return 0;

}

**Convex HULL:**

#define ll long long

#define siz 100009

struct point {

ll x, y;

};

point p[siz], hull[2 \* siz];

ll sz = 0;

bool cmp(point a, point b) {

if(a.x != b.x)

return a.x < b.x;

return a.y < b.y;

}

ll cross (point a, point b, point c) {

return (b.x - a.x) \* (c.y - a.y) - (b.y - a.y) \* (c.x - a.x);

}

void ConvexHull(ll n) {

sz = 0;

sort(p, p + n, cmp);

/// Building upper hull

for(ll i = 0; i < n; i++) {

while (sz > 1 && cross(hull[sz - 2], hull[sz - 1], p[i]) <= 0) --sz; /// use < 0 for taking co-linear points

hull[sz++] = p[i];

}

/// Building lower hull

for(int i = n - 2, j = sz + 1; i >= 0; i--) {

while (sz >= j && cross(hull[sz - 2], hull[sz - 1], p[i]) <= 0) --sz; /// use < 0 for taking co-linear points

hull[sz++] = p[i];

}

/// last point is same as first point. so, sz--

/// sz is the size of convex hull

sz--;

}

**Digit DP without memset:**

///How many zeros in the numbers' digits. Range of the numbers is (l, r)

#include <bits/stdc++.h>

using namespace std;

#define ll long long

#define pb push\_back

ll dp[2][2][12][12];

vector <ll> num;

ll solve(ll isStart, ll isSmall, ll pos, ll val)

{

if(pos == 0)

return val;

ll &ret = dp[isStart][isSmall][pos][val];

if(ret != -1 && isSmall)

return ret;

ll lim, pos2 = num.size() - pos;

if(isSmall)

lim = 9;

else

lim = num[pos2];

ll rt = 0;

if(!isStart) {

for(ll i = 0; i <= lim; i++)

rt += solve(0, isSmall | i < num[pos2], pos - 1, (i == 0) + val);

}

else {

for(ll i = 1; i <= lim; i++)

rt += solve(0, isSmall | i < num[pos2], pos - 1, val);

rt += solve(1, 1, pos - 1, 0);

}

return ret = rt;

}

ll calc(ll n)

{

if(n < 0)

return 0;

if(n < 10)

return 1;

ll tmp = n;

num.clear();

while(tmp) {

num.pb(tmp % 10);

tmp /= 10;

}

reverse(num.begin(), num.end());

return solve(1, 0, num.size(), 0) + 1; /// + 1 is for the number "0". We are not calculating this number in solve function.

}

int main()

{

ll t, caseno = 0;

memset(dp, -1, sizeof(dp));

cin >> t;

while(t--) {

ll l, r;

scanf("%lld %lld", &l, &r);

ll ans = calc(r);

ans -= calc(l - 1);

printf("Case %lld: %lld\n", ++caseno, ans);

}

return 0;

}

**Order Statitics tree :**

#include <bits/stdc++.h>

#include <ext/pb\_ds/tree\_policy.hpp>

#include <ext/pb\_ds/assoc\_container.hpp>

using namespace std;

using namespace \_\_gnu\_pbds;

template <typename T> using orderedSet =

tree<T, null\_type, less<T>, /// greater<T> for sorting decreasingly

rb\_tree\_tag, tree\_order\_statistics\_node\_update>;

int main()

{

orderedSet <int> os;

os.insert(5);

os.insert(2);

os.insert(6);

os.insert(4);

os.insert(2);

os.insert(7);

cout << \*os.find\_by\_order(0)<<endl; // 2

cout << \*os.find\_by\_order(1)<<endl; // 4

cout << \*os.find\_by\_order(2)<<endl; // 5

cout << \*os.find\_by\_order(3)<<endl; // 6

cout << \*os.find\_by\_order(4)<<endl; // 7

cout << os.order\_of\_key(5) << endl; // 2

cout << os.order\_of\_key(2) << endl; // 0

cout << os.order\_of\_key(6) << endl; // 3

cout << os.order\_of\_key(4) << endl; // 1

cout << os.order\_of\_key(7) << endl; // 4

}

**Miller Rabin :**

/\* Miller Rabin Primality Test for <= 10^18 \*/

#define ll long long

ll mulmod(ll a, ll b, ll c)

{

ll x = 0, y = a % c;

while (b)

{

if (b & 1) x = (x + y) % c;

y = (y << 1) % c;

b >>= 1;

}

return x % c;

}

ll fastPow(ll x, ll n, ll MOD)

{

ll ret = 1;

while (n)

{

if (n & 1) ret = mulmod(ret, x, MOD);

x = mulmod(x, x, MOD);

n >>= 1;

}

return ret % MOD;

}

bool isPrime(ll n)

{

if(n == 2 || n == 3) return true;

if(n == 1 || !(n & 1)) return false;

ll d = n - 1;

int s = 0;

while (d % 2 == 0)

{

s++;

d /= 2;

}

int a[9] = { 2, 3, 5, 7, 11, 13, 17, 19, 23 };

for(int i = 0; i < 9; i++)

{

if(n == a[i]) return true;

bool comp = fastPow(a[i], d, n) != 1;

if(comp) for(int j = 0; j < s; j++)

{

ll fp = fastPow(a[i], (1LL << (ll)j)\*d, n);

if (fp == n - 1)

{

comp = false;

break;

}

}

if(comp) return false;

}

return true;

}

**FFT :**

/\*\*\*

\* Multiply (7x^2 + 8x^1 + 9x^0) with (6x^1 + 5x^0)

\* ans = 42x^3 + 83x^2 + 94x^1 + 45x^0

\* A = {9, 8, 7}

\* B = {5, 6}

\* V = multiply(A,B)

\* V = {45, 94, 83, 42}

\*\*\*/

/\*\*\* Tricks

\* Use vector < bool > if you need to check only the status of the sum

\* Use bigmod if the power is over same polynomial && power is big

\* Use long double if you need more precision

\* Use long long for overflow

\*\*\*/

typedef vector <int> vi;

const double PI = 2.0 \* acos(0.0);

using cd = complex<double>;

void fft(vector<cd> & a, bool invert = 0)

{

int n = a.size();

for (int i = 1, j = 0; i < n; i++) {

int bit = n >> 1;

for (; j & bit; bit >>= 1)

j ^= bit;

j ^= bit;

if (i < j)

swap(a[i], a[j]);

}

for (int len = 2; len <= n; len <<= 1) {

double ang = 2 \* PI / len \* (invert ? -1 : 1);

cd wlen(cos(ang), sin(ang));

for (int i = 0; i < n; i += len) {

cd w(1);

for (int j = 0; j < len / 2; j++) {

cd u = a[i+j], v = a[i+j+len/2] \* w;

a[i+j] = u + v;

a[i+j+len/2] = u - v;

w \*= wlen;

}

}

}

if (invert) {

for (cd & x : a)

x /= n;

}

}

void ifft(vector <cd> & p)

{

fft(p, 1);

}

vi multiply(vi const& a, vi const& b)

{

vector<cd> fa(a.begin(), a.end()), fb(b.begin(), b.end());

int n = 1;

while (n < a.size() + b.size())

n <<= 1;

fa.resize(n);

fb.resize(n);

fft(fa);

fft(fb);

for (int i = 0; i < n; i++)

fa[i] \*= fb[i];

ifft(fa);

vi result(n);

for (int i = 0; i < n; i++)

result[i] = round(fa[i].real());

return result;

}

**HLD with value in edges:**

//Bob likes trees and XOR (Exclusive OR) very much. So he created a tree with each edge having a weight associated with it.

//One day his friend Alice came to his house, after playing video games for some time they got bored and decided to play something else.

//Bob showed the tree to Alice and told her that given any 2 vertices in the tree he can tell the XOR of all the weights on the unique path joining these 2 vertices.

//Alice thought it was a little easy and wanted to twist this a little bit. Now she proposed a different version involving q queries:

//

//Each query can be of 2 types.

//1 x y change the weight of xth edge to y.

//2 u v find the xor of the weights on the path joining vertices u and v.

//Can you help Bob solve this problem? For every query of 2nd type output the XOR value.

#include <bits/stdc++.h>

using namespace std;

#define siz 100009

#define ll int

#define pb push\_back

vector <ll> graph[siz], cost[siz], edge[siz];

ll start[siz], stop[siz], TM = 1, tree1[8 \* siz], tree2[8 \* siz], to[siz], val[siz];

void dfs(ll n, ll p)

{

start[n] = TM++;

for(ll i = 0; i < graph[n].size(); i++) {

if( graph[n][i] == p)

continue;

to[ edge[n][i] ] = graph[n][i];

val[ graph[n][i] ] = cost[n][i];

dfs(graph[n][i], n);

}

stop[n] = TM++;

}

void update1(ll lo, ll hi, ll indx, ll val, ll node)

{

if(lo > indx || hi < indx)

return;

if(lo == indx && hi == indx) {

tree1[node] = val;

return;

}

ll mid = (lo + hi) / 2;

update1(lo, mid, indx, val, 2 \* node);

update1(mid + 1, hi, indx, val, 2 \* node + 1);

tree1[node] = tree1[2 \* node] ^ tree1[2 \* node + 1];

}

void update2(ll lo, ll hi, ll indx, ll val, ll node)

{

if(lo > indx || hi < indx)

return;

if(lo == indx && hi == indx) {

tree2[node] = val;

return;

}

ll mid = (lo + hi) / 2;

update2(lo, mid, indx, val, 2 \* node);

update2(mid + 1, hi, indx, val, 2 \* node + 1);

tree2[node] = tree2[2 \* node] ^ tree2[2 \* node + 1];

}

ll query1(ll lo, ll hi, ll left, ll right, ll node)

{

if(lo > right || hi < left)

return 0;

if(lo >= left && hi <= right)

return tree1[node];

ll mid = (lo + hi) / 2;

ll p1 = query1(lo, mid, left, right, 2 \* node);

ll p2 = query1(mid + 1, hi, left, right, 2 \* node + 1);

return p1 ^ p2;

}

ll query2(ll lo, ll hi, ll left, ll right, ll node)

{

if(lo > right || hi < left)

return 0;

if(lo >= left && hi <= right)

return tree2[node];

ll mid = (lo + hi) / 2;

ll p1 = query2(lo, mid, left, right, 2 \* node);

ll p2 = query2(mid + 1, hi, left, right, 2 \* node + 1);

return p1 ^ p2;

}

int main()

{

ll n;

cin >> n;

for(ll i = 1; i < n; i++) {

ll u, v, w;

scanf("%lld %lld %lld", &u, &v, &w);

graph[u].pb(v);

graph[v].pb(u);

cost[u].pb(w);

cost[v].pb(w);

edge[u].pb(i);

edge[v].pb(i);

}

dfs(1, -1);

for(ll i = 1; i < n; i++) {

update1(1, TM, start[ to[i] ], val[ to[i] ], 1);

update2(1, TM, stop[ to[i] ], val[ to[i] ], 1);

}

ll q;

cin >> q;

for(ll i = 1; i <= q; i++) {

ll state, u, v;

scanf("%d %d %d", &state, &u, &v);

if(state == 1) {

update1(1, TM, start[ to[u] ], v, 1);

update2(1, TM, stop[ to[u] ], v, 1);

}

else {

ll ans = 0;

ans ^= query1(1, TM, 1, start[u], 1);

ans ^= query2(1, TM, 1, start[u], 1);

ans ^= query1(1, TM, 1, start[v], 1);

ans ^= query2(1, TM, 1, start[v], 1);

printf("%d\n", ans);

}

}

return 0;

}

**HLD value In Nodes:**

//Finally the Great Magical Lamp was in Aladdin's hand. Now he wanted to return home. But he didn't want to take any help from the Genie

// because he thought that it might be another adventure for him. All he remembered was the paths he had taken to reach there. But since he took the lamp,

// all the genies in the cave became angry and they were planning to attack. As Aladdin was not afraid, he wondered how many genies were there. He summoned the Genie from the lamp and asked this.

//

//Now you are given a similar problem. For simplicity assume that, you are given a tree (a connected graph with no cycles) with n nodes, nodes represent places, edges represent roads.

//In each node, initially there are an arbitrary number of genies. But the numbers of genies change in time. So, you are given a tree, the number of genies in each node and several queries of two types.

//They are:

//

//1) 0 i j, it means that you have to find the total number of genies in the nodes that occur in path from node i to j (0 ≤ i, j < n).

//

//2) 1 i v, it means that number of genies in node i is changed to v (0 ≤ i < n, 0 ≤ v ≤ 1000).

#include <bits/stdc++.h>

using namespace std;

#define sz 30009

#define tsz 60009

#define ll int

#define pb push\_back

vector <ll> graph[sz];

ll node[sz], tree1[4 \* tsz], tree2[4 \* tsz], ltree[4 \* tsz], start[sz], stop[sz], frst[sz], TM = 1;

ll nodeOf[tsz], euler\_tour[tsz], el = 0;

void dfs(ll u, ll p)

{

nodeOf[TM] = u;

start[u] = TM++;

frst[u] = el;

euler\_tour[ el++ ] = start[u];

for(ll i = 0; i < graph[u].size(); i++) {

if(graph[u][i] == p)

continue;

dfs(graph[u][i], u);

euler\_tour[ el++ ] = start[u];

}

stop[u] = TM++;

}

void update1(ll lo, ll hi, ll indx, ll val, ll node)

{

if(lo > indx || hi < indx)

return;

if(lo == hi && hi == indx) {

tree1[node] = val;

return;

}

ll mid = (lo + hi) / 2;

update1(lo, mid, indx, val, 2 \* node);

update1(mid + 1, hi, indx, val, 2 \* node + 1);

tree1[node] = tree1[2 \* node] + tree1[2 \* node + 1];

}

void update2(ll lo, ll hi, ll indx, ll val, ll node)

{

if(lo > indx || hi < indx)

return;

if(lo == hi && hi == indx) {

tree2[node] = val;

return;

}

ll mid = (lo + hi) / 2;

update2(lo, mid, indx, val, 2 \* node);

update2(mid + 1, hi, indx, val, 2 \* node + 1);

tree2[node] = tree2[2 \* node] + tree2[2 \* node + 1];

}

ll query1(ll lo, ll hi, ll left, ll right, ll node)

{

if(lo > right || hi < left)

return 0;

if(lo >= left && hi <= right)

return tree1[node];

ll mid = (lo + hi) / 2;

ll p1 = query1(lo, mid, left, right, 2 \* node);

ll p2 = query1(mid + 1, hi, left, right, 2 \* node + 1);

return p1 + p2;

}

ll query2(ll lo, ll hi, ll left, ll right, ll node)

{

if(lo > right || hi < left)

return 0;

if(lo >= left && hi <= right)

return tree2[node];

ll mid = (lo + hi) / 2;

ll p1 = query2(lo, mid, left, right, 2 \* node);

ll p2 = query2(mid + 1, hi, left, right, 2 \* node + 1);

return p1 + p2;

}

void build(ll lo, ll hi, ll node)

{

if(lo == hi) {

ltree[node] = euler\_tour[lo];

return;

}

ll mid = (lo + hi) / 2;

build(lo, mid, 2 \* node);

build(mid + 1, hi, 2 \* node + 1);

ltree[node] = min( ltree[2 \* node], ltree[2 \* node + 1] );

}

ll lcaquery(ll lo, ll hi, ll left, ll right, ll node)

{

if(lo > right || hi < left)

return INT\_MAX;

if(lo >= left && hi <= right)

return ltree[node];

ll mid = (lo + hi) / 2;

ll p1 = lcaquery(lo, mid, left, right, 2 \* node);

ll p2 = lcaquery(mid + 1, hi, left, right, 2 \* node + 1);

return min(p1, p2);

}

int main()

{

ll t, caseno = 0;

cin >> t;

while(t--) {

el = 0, TM = 1;

memset(tree1, 0, sizeof(tree1));

memset(tree2, 0, sizeof(tree2));

ll n, u, v;

scanf("%d", &n);

for(ll i = 0; i < n; i++)

scanf("%d", &node[i]);

for(ll i = 0; i < n - 1; i++) {

scanf("%d %d", &u, &v);

graph[u].pb(v);

graph[v].pb(u);

}

dfs(0, -1);

for(ll i = 0; i < n; i++) {

update1(1, TM, start[i], node[i], 1);

update2(1, TM, stop[i], node[i], 1);

}

build(0, el, 1);

ll q;

scanf("%d", &q);

printf("Case %d:\n", ++caseno);

for(ll i = 1; i <= q; i++) {

ll state, u, v;

scanf("%d %d %d", &state, &u, &v);

if(state == 0) {

if(u == v) {

printf("%d\n", node[u]);

continue;

}

ll lca = -1;

ll ans1 = query1(1, TM, 1, start[u], 1);

ans1 -= query2(1, TM, 1, start[u], 1);

ll ans2 = query1(1, TM, 1, start[v], 1);

ans2 -= query2(1, TM, 1, start[v], 1);

if(start[u] < start[v]) {

if(stop[u] > stop[v])

lca = u;

}

if(start[v] < start[u]) {

if(stop[v] > stop[u])

lca = v;

}

if(lca == -1) {

ll left = frst[u], right = frst[v];

if(left > right)

swap(left, right);

lca = nodeOf[ lcaquery(0, el, left, right, 1) ];

}

ll ans3 = query1(1, TM, 1, start[ lca], 1);

ans3 -= query2(1, TM, 1, start[ lca], 1);

ll ans = ans1 + ans2 - (2 \* ans3) + node[lca];

printf("%d\n", ans);

}

else {

node[u] = v;

update1(1, TM, start[u], v, 1);

update2(1, TM, stop[u], v, 1);

}

}

for(ll i = 0; i < sz; i++)

graph[i].clear();

}

return 0;

}