**Subrroted tree sum(graph flatten and segment tree)**

void dfs(ll u)

{

sttime[u]=++tim;

vist[u]=1;

for(auto v:g[u])

{

if(!vist[v])

dfs(v);

}

fintime[u]=tim;

}

void build(ll cur,ll left,ll right)

{

if(left==right)

{

st[cur]=valu[a[left]];

return;

}

ll mid=(left+right)/2;

build(cur\*2,left,mid);

build(cur\*2+1,mid+1,right);

st[cur]=st[cur\*2]+st[cur\*2+1];

}

void update(ll cur,ll left,ll right,ll pos,ll val)

{

if(pos<left||pos>right)return;

if(pos<=left&&pos>=right)

{

st[cur]=val;

return;

}

ll mid=(left+right)/2;

update(cur\*2,left,mid,pos,val);

update(cur\*2+1,mid+1,right,pos,val);

st[cur]=st[cur\*2]+st[cur\*2+1];

}

ll query(ll cur,ll left,ll right,ll l,ll r)

{

if(r<left||l>right)return 0;

if(l<=left&&r>=right)

{

return st[cur];

}

ll mid=(left+right)/2;

ll x=query(cur\*2,left,mid,l,r);

ll y=query(cur\*2+1,mid+1,right,l,r);

return (x+y);

}

LCA + flat tree + MO’s

vll adj[N];

ll a[N\*3];

ll in[N];

ll out[N];

ll x;

struct query

{

ll l,r,id;

ll typ;

};

ll val[N];

query queries[N+4];

**struct LowestCommonAncestor**

{

int N, root = 0, po;

vector <vector <int> > g;

vector <vector <int> > sptab;

vector <int> depth;

vector <int> parent;

void Init(int \_n)

{

N = \_n;

po = log2((N)) + 1;

g.assign(N, {});

depth.resize(N);

parent.resize(N);

sptab.assign(N, {});

}

void AddEdge(int u, int v)

{

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

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

}

void Dfs(int u, int par = -1)

{

if(par == -1)

{

depth[u] = 0;

parent[u] = -1;

}

for(int v : g[u])

{

if (v == par) continue;

parent[v] = u;

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

Dfs(v, u);

}

}

void SparceTable()

{

for(int i = 0 ; i < N ; i++) sptab[i][0] = parent[i];

for(int j = 1 ; (1 << j) < N ; j++)

{

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

{

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

{

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

}

}

}

}

void Build()

{

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

{

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

{

sptab[i].push\_back(-1);

}

}

Dfs(root);

SparceTable();

}

**int Lca(int u, int v)**

{

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

int log;

for(log = 1 ; (1 << log) <= depth[u] ; log++);

log--;

for(int i = log ; i >= 0 ; i--)

{

if(depth[u] - (1 << i) >= depth[v])

{

u = sptab[u][i];

}

}

if(u == v) return u;

for(int i = log ; i >= 0 ; i--)

{

if(sptab[u][i] != -1 && sptab[u][i] != sptab[v][i])

{

u = sptab[u][i];

v = sptab[v][i];

}

}

return parent[u];

}

int KthAncestor(int u, int k)

{

int log;

for(log = 1 ; (1 << log) <= depth[u] ; log++);

log--;

for(int i = log ; i >= 0 ; i--)

{

if(k - (1 << i) >= 0)

{

u = sptab[u][i];

k -= (1 << i);

}

}

return u;

}

int Getdist(int u, int v)

{

return (depth[u] + depth[v] - (2 \* (depth[Lca(u, v)]))) ;

}

**bool IsAnsector(int u, int v)**

{

int cur = Lca(u, v);

if(cur == u) return 1;

return 0;

}

} lca;

void dfs(ll u,ll p)

{

in[u]=x;

a[x++]=u;

for(ll j : adj[u])

{

if(j==p) continue;

dfs(j,u);

}

out[u]=x;

a[x++]=u;

}

ll block=320;

bool cmp(query p1, query p2)

{

ll b\_a=p1.l/block;

ll b\_b=p2.l/block;

if(b\_a!=b\_b) return b\_a < b\_b;

return p1.r < p2.r;

}

ll uni=0;

ll freq[N+11];

ll node[N+1];

void add(ll idx)

{

ll n1 = a[idx];

node[n1]++;

if(node[n1]==1)

{

freq[val[n1]]++;

if(freq[val[n1]]==1) uni++;

}

else

{

freq[val[n1]]--;

if(freq[val[n1]]==0) uni--;

}

}

void del(ll idx)

{

ll n1 = a[idx];

node[n1]--;

if(node[n1]==1)

{

freq[val[n1]]++;

if(freq[val[n1]]==1) uni++;

}

else

{

freq[val[n1]]--;

if(freq[val[n1]]==0) uni--;

}

}

**two types query: 1. change s to x 2 . calc dis be U to V.**

const ll mxn=3\*10e4;

llsttime[mxn+3],fintime[mxn+3],vist1[mxn+3],valu[mxn\*2+4]a[mxn\*2+4];

vector<ll>g[mxn+4];

ll tim=0;

void dfs\_for\_timeing(ll u)

{

vist1[u]=1;

sttime[u]=++tim;

for(auto v:g[u])

{

if(!vist1[v])

{

dfs\_for\_timeing(v);

}

}

fintime[u]=++tim;

}

ll st[mxn\*4];

void build(ll cur,ll left,ll right)

{

if(left==right)

{

st[cur]=valu[a[left]];

return;

}

ll mid=(left+right)/2;

build(cur\*2,left,mid);

build(cur\*2+1,mid+1,right);

st[cur]=st[cur\*2]+st[cur\*2+1];

}

void update(ll cur,ll left,ll right,ll pos,ll val)

{

if(pos<left||pos>right)return;

if(pos<=left&&pos>=right)

{

st[cur]=val;

valu[a[pos]]=val;

return;

}

ll mid=(left+right)/2;

update(cur\*2,left,mid,pos,val);

update(cur\*2+1,mid+1,right,pos,val);

st[cur]=st[cur\*2]+st[cur\*2+1];

}

ll query(ll cur,ll left,ll right,ll l,ll r)

{

if(r<left||l>right)return 0;

if(l<=left&&r>=right)

{

return st[cur];

}

ll mid=(left+right)/2;

ll x=query(cur\*2,left,mid,l,r);

ll y=query(cur\*2+1,mid+1,right,l,r);

return (x+y);

}

ll vist2[mxn+3],sptb[mxn+3][30],level[mxn+3],pa[mxn+3];

void dfs\_for\_parent(ll u)

{

vist2[u]=1;

for(auto v:g[u])

{

if(!vist2[v])

{

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

pa[v]=u;

dfs\_for\_parent(v);

}

}

}

ll sparsetable(ll n)

{

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

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

{

sptb[i][0]=pa[i];

}

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

{

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

{

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

}

}

}

ll lca(ll u,ll v)

{

if(level[u]<level[v])

swap(u,v);

ll log=log2(level[u]);

for(ll i=log; i>=0; i--)

{

if(level[u]-(1<<i)>=level[v])

{

u=sptb[u][i];

}

}

if(u==v)return u;

for(ll i=log; i>=0; i--)

{

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

{

u=sptb[u][i];

v=sptb[v][i];

}

}

return pa[u];

}

class dsuClass

{

public:

ll n;

vector<ll> parent;

vector<ll> size;

ll sccCount;

void init(ll n)

{

this->n = n;

parent.resize(n);

size.resize(n);

reset();

}

void reset()

{

sccCount = n;

for(ll i=0; i<n; i++) parent[i] = i;

for(ll i=0; i<n; i++) size[i] = 1;

}

ll update(ll a)

{

if (parent[a] == a) return a;

return parent[a] = update(parent[a]);

}

void join(ll a, ll b)

{

a = update(a);

b = update(b);

if (size[b] > size[a]) swap(a, b);

if (a == b) return;

size[a] += size[b];

parent[b] = a;

sccCount--;

}

ll \_size(ll a)

{

a = update(a);

return size[a];

}

bool isMaster(ll v)

{

return parent[v] == v;

}

void updateAll()

{

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

}

};

**//<for policy based data structure>**

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

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

using namespace \_\_gnu\_pbds;

Trie - basic

struct node

{

bool endmark;

node \*next[27];

node()

{

endmark=false;

for(int i=0; i<26; i++)

next[i]=NULL;

}

}\*root;

void insert(char \*str,int len)

{

node \*cur=root;

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

{

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

if(cur->next[id]==NULL)

cur->next[id]=new node();

cur=cur->next[id];

}

cur->endmark=true;

}

bool search(char \*str,int len)

{

node \*cur=root;

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

{

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

if(cur->next[id]==NULL)

return false;

cur=cur->next[id];

}

return cur->endmark;

}

void del(node \*cur)

{

for(int i=0; i<26; i++)

{

if(cur->next[i])

del(cur->next[i]);

}

delete(cur);

}

**Longest palindromic substring**

**#include <bits/stdc++.h>**

**using namespace std;**

**#define ll long long**

#define fast \

ios\_base::sync\_with\_stdio(false); \

cin.tie(NULL);

const ll mxn=1e6+10;

const ll base = 1949313259;

const ll mod = 2117566807;

ll prehash[mxn+10],sufhash[mxn+10],pw[mxn+10],inv[mxn+10];

ll big\_mod(ll a,ll b)

{

if(b==0)return 1;

else if(b==1)return a;

if(b%2==0)

{

ll tm=big\_mod(a,b/2);

return ((tm%mod)\*(tm%mod))%mod;

}

else return ((a%mod)\*(big\_mod(a,b-1)%mod))%mod;

}

void Compute\_PreHash\_SufHash(string s)

{

ll len=s.size();

pw[0]=pw[1]=1;

inv[0]=inv[1]=1;

ll inmod=big\_mod(base,mod-2);

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

{

pw[i]=(pw[i-1]\*base)%mod;

inv[i]=(inv[i-1]\*inmod)%mod;

}

prehash[0]=0;

for(ll i=1; i<=s.size(); i++)

{

prehash[i]=(prehash[i-1]+(s[i-1]-'a'+1)\*pw[i])%mod;

}

sufhash[0]=0;

for(ll i=s.size(); i>=1; i--)

{

sufhash[len+1-i]=(sufhash[len-i]+(s[i-1]-'a'+1)\*pw[len+1-i])%mod;

}

}

ll GetPreHash(ll left,ll right)

{

ll Hash=((prehash[right]-prehash[left-1])\*inv[left])%mod;

if(Hash<0)

Hash+=mod;

return Hash;

}

ll GetSufHash(ll left,ll right)

{

ll Hash=((sufhash[right]-sufhash[left-1])\*inv[left])%mod;

if(Hash<0)

Hash+=mod;

return Hash;

}

int main()

{

fast

string s;

cin>>s;

ll len=s.size();

Compute\_PreHash\_SufHash(s);

ll leftans=1;

ll rightans=1;

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

{

ll left=i;

ll right=min(len-left,left-1)+left;

while(left<=right)

{

ll mid=(left+right)/2;

if(GetPreHash(i,mid)==GetSufHash(len+1-i,len+1-i+(mid-i)))

{

if(abs(leftans-rightans)<abs(mid-(i-(mid-i))))

{

leftans=i-(mid-i);

rightans=mid;

}

left=mid+1;

}

else

{

right=mid-1;

}

}

left=i;

right=min(len+1-left,left-1)+left;

while(left<=right)

{

ll mid=(left+right)/2;

if(GetPreHash(i,mid)==GetSufHash(len+2-i,len+2-i+(mid-i))&&len+2-i>0&&len+2-i+(mid-i)<=len)

{

if(abs(leftans-rightans)<abs(mid-(i-(mid-i+1))))

{

leftans=i-(mid-i+1);

rightans=mid;

}

left=mid+1;

}

else

{

right=mid-1;

}

}

}

for(ll i=leftans; i<=rightans; i++)

cout<<s[i-1];

cout<<endl;

return 0;

}

**// <Suffix array blackbox>**

char str[MM];

int s0[(MM / 3) + 10], sa0[(MM / 3) + 10];

int n, ar[MM], sa[MM], lcp[MM], bucket[MM], mem[MM << 2];

void radixsort(int \*source, int \*dest, int \*val, int n, int lim)

{

int s = 0, x;

memset(bucket, 0, lim << 2);

ff(i, n)

bucket[val[source[i]]]++;

ff(i, lim)

{

x = bucket[i];

bucket[i] = s, s += x;

}

ff(i, n)

dest[bucket[val[source[i]]]++] = source[i];

}

void DC3(int \*ar, int \*sa, int n, int lim, int ptr)

{

int \*s12, \*sa12;

int allc = (n / 3) << 1, n0 = (n + 2) / 3;

int i, j, k, l, c, d, p, t, m, r, counter;

s12 = &mem[ptr], ptr += (allc + 5), sa12 = &mem[ptr], ptr += (allc + 5);

c = 0, m = 0, r = n + ((n % 3) == 1);

for (i = 0; i < r; i++, m++)

{

if (m == 3)

m = 0;

if (m)

s12[c++] = i;

}

s12[c] = sa12[c] = s12[c + 1] = sa12[c + 1] = s12[c + 2] = sa12[c + 2] = 0;

radixsort(s12, sa12, ar + 2, c, lim + 1);

radixsort(sa12, s12, ar + 1, c, lim + 1);

radixsort(s12, sa12, ar, c, lim + 1);

counter = 0, j = -1;

for (i = 0; i < c; i++)

{

if ((ar[sa12[i]] != j) || (ar[sa12[i] + 1] != k) || (ar[sa12[i] + 2] != l))

{

counter++;

j = ar[sa12[i]], k = ar[sa12[i] + 1], l = ar[sa12[i] + 2];

}

if ((sa12[i] % 3) == 1)

s12[sa12[i] / 3] = counter;

else

s12[(sa12[i] / 3) + n0] = counter;

}

if (counter == c)

{

for (i = 0; i < c; i++)

sa12[s12[i] - 1] = i;

}

else

{

DC3(s12, sa12, c, counter, ptr);

for (i = 0; i < c; i++)

s12[sa12[i]] = i + 1;

}

for (i = 0, d = 0; i < c; i++)

{

if (sa12[i] < n0)

s0[d++] = (sa12[i] \* 3);

}

radixsort(s0, sa0, ar, d, lim + 1);

for (k = 0, l = ((n % 3) == 1), r = 0; r < n; r++)

{

j = sa0[k];

i = ((sa12[l] < n0) ? (sa12[l] \* 3) + 1 : ((sa12[l] - n0) \* 3) + 2);

if (l == c)

sa[r] = sa0[k++];

else if (k == d)

sa[r] = i, l++;

else

{

if (sa12[l] < n0)

{

if ((ar[i] < ar[j]) || (ar[i] == ar[j] && s12[sa12[l] + n0] <= s12[j / 3]))

sa[r] = i, l++;

else

sa[r] = j, k++;

}

else

{

if ((ar[i] < ar[j]) || (ar[i] == ar[j] && ar[i + 1] < ar[j + 1]) || (ar[i] == ar[j] && ar[i + 1] == ar[j + 1] && s12[sa12[l] - n0 + 1] <= s12[(j / 3) + n0]))

sa[r] = i, l++;

else

sa[r] = j, k++;

}

}

}

}

void LcpArray()

{

int i, j, k;

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

ar[sa[i]] = i;

for (k = 0, i = 0; i < n; i++, k ? k-- : 0)

{

if (ar[i] == (n - 1))

k = 0;

else

{

j = sa[ar[i] + 1];

while (((i + k) < n) && ((j + k) < n) && (str[i + k] == str[j + k]))

k++;

}

lcp[ar[i]] = k;

}

}

void genSuffixArray()

{

int i, j, lim = 0;

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

{

ar[i] = str[i];

if (ar[i] > lim)

lim = ar[i];

}

ar[n] = ar[n + 1] = ar[n + 2] = 0;

DC3(ar, sa, n, lim, 0);

}

// **preprocesssing the prefix index of the suffix till ith index of the pattern**

**int lps[MM];**

void CalcLPS(string &pat)

{

// i is for prefix pointer, starts at 0

int i = 0;

lps[0] = 0;

// suffix pointer starts from 1

ff2(j, 1, pat.size())

{

// if prefix index matches keep matching

if (pat[i] == pat[j])

{

lps[j] = lps[j - 1] + 1;

i++;

}

// otherwise reset i to 0

// and its easy to check the current index right here

// than trying to j--, trust me i have tested it

// sometimes you just have to write more code to be understandable

else

{

i = 0;

if (pat[i] == pat[j])

{

lps[j] = 1;

i++;

}

else

lps[j] = 0;

}

}

}

string s, pat;

void answer()

{

cin >> s >> pat;

CalcLPS(pat);

int j = 0;

int m = pat.size();

int cnt = 0;

int i = 0;

while (i < s.size())

{

// if they match keep going

if (pat[j] == s[i])

{

i++;

j++;

}

// if we can stil go back step by step, keep going

// keep comparing eith the same index of the main index

else if (j > 0)

{

j = lps[j - 1];

}

// if j==0, just increase the main-string index

else

{

i++;

}

// found a match, increase the counter

// can also keep the matching starting index, i-j

// then start matching after the prefix that is the suffix till j-1

if (j == m)

{

cnt++;

j = lps[j - 1];

}

}

outln(cnt);

}

Direction array

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

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

**Variation of DFS Cycle Undirected Graph**

void dfs(ll u,ll p)

{

colur[u]=1;

for(auto v: g[u])

{

if(colur[v]==0&&f==0)

{

pa[v]=u;

dfs(v,u);

}

else if(v!=p&&f==0)

{

st\_city=u;

en\_city=v;

f=1;

}

}

colur[u]=2;

}

void cycle\_print()

{

cycle.pb(en\_city);

while(st\_city!=en\_city)

{

cycle.pb(st\_city);

st\_city=pa[st\_city];

}

cycle.pb(en\_city);

}

**Variation of DFS Cycle Directed Graph**

void dfs(ll u,ll p)

{

colur[u]=1;

for(auto v: g[u])

{

if(colur[v]==0&&f==0)

{

pa[v]=u;

dfs(v,u);

}

else if(colur[v]==1&&f==0)

{

st\_city=u;

en\_city=v;

f=1;

}

}

colur[u]=2;

}

void cycle\_print()

{

cycle.pb(en\_city);

while(st\_city!=en\_city)

{

cycle.pb(st\_city);

st\_city=pa[st\_city];

}

cycle.pb(en\_city);

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

}

Topological sort

void dfs(ll u)

{

visit[u]=1;

for(auto v: g[u])

{

if(visit[v]==0)

dfs(v);

else if(vi[v]==1)

f=1;

}

visit[u]=2;

ve.pb(u);

}

void dijstra(ll u,ll n)

{

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

cost[i]=1e18;

cost[1]=0;

priority\_queue<pair<ll,ll>>q;

q.push({-cost[u],u});

while(!q.empty())

{

ll wt=-q.top().first;

ll u=q.top().second;

q.pop();

if(cost[u]<wt)continue;

for(auto i:g[u])

{

ll v=i.first;

ll wtt=i.second;

if(cost[v]>wtt+wt)

{

cost[v]=wt+wtt;

q.push({-cost[v],v});

}

}

}

}

Variation Dijstra kth Sortest path

// maximum k=10;

void dijstra(ll n,ll k)

{

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

{

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

{

if(i==1)

{

cost[i][j]=0;

}

cost[i][j]=INF;

}

}

priority\_queue<pair<ll,ll>>q;

q.push({0,1});

while(!q.empty())

{

ll wt=-q.top().first;

ll u=q.top().second;

q.pop();

if(cost[u][k-1]<wt)continue;

for(auto i:g[u])

{

ll wtch=i.second;

ll v=i.first;

if(wt+wtch<cost[v][k-1])

{

cost[v][k-1]=wt+wtch;

q.push({-cost[v][k-1],v});

sort(cost[v],cost[v]+k);

}

}

}

}

**Variation of Dijsta Topological Sort**

void toplogical\_sort(ll n)

{

vector<ll>vv;

queue<ll>q;

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

{

if(cost[i]==0)

q.push(i);

}

while(!q.empty())

{

ll a=q.front();

q.pop();

vv.pb(a);

for(ll i=0; i<v[a].size(); i++)

{

ll m=v[a][i];

cost[m]--;

if(cost[m]==0)

{

q.push(m);

}

}

}

if(vv.size()==n)

{

for(ll i=0; i<vv.size(); i++)cout<<vv[i]<<in;

cout<<endl;

}

else cout<<"IMPOSSIBLE"<<endl;

}

**Variation Dijstra**

**>> what is the minimum price of such a route?**

**>> how many minimum-price routes are there? (modulo 109+7)**

**>> what is the minimum number of flights in a minimum-price route?**

**>> what is the maximum number of flights in a minimum-price route?**

**vector<pair<ll,ll>>g[mxn+3];**

ll cost[mxn+3],FlightMx[mxn+3],FlightMn[mxn+3],FlightRoute[mxn+3];

void dijstra(ll n)

{

for(ll i=1; i<=n+5; i++)

{

cost[i]=INF;

FlightRoute[i]=1;

}

cost[1]=0;

priority\_queue<pair<ll,ll>>q;

q.push({-0,1});

while(!q.empty())

{

ll wtp=-q.top().first; //wtp weight of parent

ll u=q.top().second;

q.pop();

if(cost[u]<wtp)

continue;

for(auto ch:g[u])

{

ll wtch=ch.second; //wtch weight of child

ll v=ch.first;

if(wtch+wtp==cost[v])

{

cost[v]=wtch+wtp;

FlightRoute[v]+=FlightRoute[u];

FlightRoute[v]%=mod;

FlightMx[v]=max(FlightMx[v],FlightMx[u]+1);

FlightMn[v]=min(FlightMn[v],FlightMn[u]+1);

}

else if(wtch+wtp<cost[v])

{

cost[v]=wtch+wtp;

FlightRoute[v]=FlightRoute[u];

FlightRoute[v]%=mod;

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

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

q.push({-cost[v],v});

}

}

}

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Floyad Warshal \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

void Floyad\_warshal(ll u,ll n)

{

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

{

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

{

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

{

cost[i][j]=min(cost[i][j],cost[i][k]+cost[k][j]);

}

}

}

}

**Bellmand Ford and Negetive Cycle print**

struct edge

{

ll a, b, wet;

} g[mxn];

void bellmand(ll n, ll m)

{

ll cost[mxn];

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

cost[i] = 1e18;

ll pa[mxn], f;

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

{

f = -1;

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

{

if (cost[g[j].a] + g[j].wet < cost[g[j].b])

{

f = g[j].b;

pa[g[j].b] = g[j].a;

cost[g[j].b] = cost[g[j].a] + g[j].wet;

}

}

}

if (f == -1)

cout << "NO" << endl;

else

{

cout << "YES" << endl;

ll y = pa[f];

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

{

y = pa[y];

}

vector<ll> ans;

ans.push\_back(y);

ll x=y;

while (1)

{

if (ans.size() > 1 && y == pa[x])

break;

x = pa[x];

ans.push\_back(x);

}

ans.push\_back(y);

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

for (ll i = 0; i < ans.size(); i++)

cout << ans[i] << " ";

cout << endl;

}

**}**

**Minimum Spaning Tree**

void make\_set(ll n)

{

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

{

parent[i] = i;

}

}

ll find(ll x)

{

if (x == parent[x])

return x;

else

return parent[x] = find(parent[x]);

}

ll unio(ll u, ll v)

{

u = find(u);

v = find(v);

if (u == v)

return 0;

else

{

parent[u] = v;

return 1;

}

}

**void MST() // Minimum Spaning Tree**

**{**

sort(g.begin(),g.end()); // sort on weight

for (ll i = 0; i < g.size(); i++)

{

ll u = g[i].second.first;

ll v = g[i].second.second;

ll cost = g[i].first;

if (unio(u, v))

{

st.insert(u);

st.insert(v);

ans += cost;

}

}

}

**\*\*\*\*\*\*\*\*\*\*\*\* Articulation point / bridge \*\*\*\*\*\*\*\***

Bridge:

\_\_\_\_\_\_\_\_\_\_\_

const ll N = 1e5+7;

vector<ll>g[N];

bool vis[N];

ll timer=0;

vector<pair<ll,ll> >bridgEdge;

ll dfs(ll u,ll p)

{

    timer++;

    vis[u]=true;

    in[u]=timer;

    low[u]=timer;

    for(auto v : g[u] )

    {

        if(p==v)continue;

        if(!vis[v])

        {

            dfs(v,u);

            if(low[v]>in[u])

            {

               bridgEdge.push\_back({v,u});

            }

            low[u]=min(low[u],low[v]);

        }

        else

        {

            low[u]=min(low[u],in[v]);

        }

    }

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Articulation Point :**

ll n,m;

ll timer=0,ans=0;

ll dfs(ll u,ll p=-1)

{

    timer++;

    vis[u]=true;

    in[u]=timer;

    low[u]=in[u];

    ll child=0;

    for(auto v : g[u] )

    {

        if(p==v)continue;

        if(!vis[v])

        {

            dfs(v,u);

            low[u]=min(low[u],low[v]);

            if(low[v]>=in[u]&&p!=-1&&!point[u])

            {

                point[u]=true;

                ans++;

            }

            child++;

        }

        else

            low[u]=min(low[u],in[v]);

    }

    if(p==-1&&child>1)// if u is a root node

        ans++;

}

Sieve :

const int N = 1e8 + 9;

bool f[N];

int32\_t main()

{

    int n = N - 9;

    vector<int> primes;

    f[1] = true;

    for (int i = 2; i \* i <= n; i++)

    {

        if (!f[i])

        {

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

            {

                f[j] = true;

            }

        }

    }

    for (int i = 2; i <= n; i++)

    {

        if (!f[i])

        {

            primes.push\_back(i);

        }

    }

    cout << primes.size() << '\n';

    return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Largest of powr Of P :**

#define ll long long

ll legend(ll n,ll p)

{

    ll cnt=0;

    while(n>1)

    {

        n/=p;

        cnt+=n;

    }

    return cnt;

}

vector<pair<ll,ll> >primefact(ll b)

{

   vector<pair<ll,ll> >fact;

   for(ll i=2;b!=1;i++)

   {

       if(b%i==0)

       {

           ll cnt=0;

           while(b%i==0)

           {

               cnt++;

               b/=i;

           }

           fact.push\_back({i,cnt});

       }

   }

   return fact;

}

ll lpox(ll n,ll b)

{

    vector<pair<ll,ll> >bf=primefact(b);

    ll ans =  INT\_MAX;

    for(ll i=0;i<bf.size();i++)

    {

        ll a = bf[i].first;

        ll b = bf[i].second;

        ans=min(ans,legend(n,a)/b);

    }

    return ans;

}

int main()

{

    ll n,p;

    cin>>n>>p;

    cout<<(lpox(n,p))<<endl;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

String s;

getline(cin,s);

Stringstream ss;

ss<<s;

String word;

while(ss>>word)

{

    cout<<word<<’\n’;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

struct cmp{

bool operator()(int a,int b) const {

return a\*a < b\*b;

}

priority\_queue<int>q;

priority\_queue<int,vector<int>,cmp >customq;

priority\_queue<int,vector<int>,greater<int> >minq;

**NUMBER THEORY**

**1. if n is odd all the divisor of n will be odd.**

**2. if d divide n than d divide also (n-d).**

**3. if n is even and is not a power of 2, it means that n has an odd divisor.**

// Permutation..

1 . for n objects how many distinct permutations exists ?

n (n - 1) (n - 2) (n - 3)...(1) = n!

2. if an integer n >= 0 , n factorial denoted n! .. is defined as

0! = 1

n! = n (n - 1) (n - 2)..(1) , for n >= 1

3. if there are n distinct objects , the number of permutations of size k ,

with 1 <= k <= n , for the n objects is

P(n , k) = n (n - 1) (n - 2) (n - 3) ... (n - k + 1)

or

P(n , k) = n! / (n - k)!

4. P(n , n) = n!

5 . permutations with repeatation :

If we have n1 indistingushable objects of a first type , n2 indistingushable objects

of a second type ,, and Nr indistingushable objects of kth type , where n1 + n2 + .. + nr = n ,

then there are ,

P(n , n) = n! / (n1!n2!..nk!)

// GCD

1 : GCD property :

For non-negative integers a and b, where a and b are not both zero, provable by considering the Euclidean algorithm in base n :

// gcd((n^a) − 1, (n^b) − 1) = (n^gcd(a,b)) − 1

// If gcd(x,n)=1, then gcd(n−x,n)=1

2 : GCD and LCM relations , It is based on the formula that ,

// LCM(A , B) × GCD( A , B) = A × B

3 : Number of divisors (NOD) of a number N can be calculated using Prime power factorization.

Let , N = P1^a1 \* P2^a2 \* P3^a3 \* ... \* Pn^ak , is the prime power factorization of a number N , where P is the prime number and

a is number of times occurs that prime number.

Then , NOD(N) defines as :

// NOD(N) = (a1 + 1) \* (a2 + 1) \* (a3 + 1) \* ... \* (ak + 1)

4 : Sum of divisors (SOD) of a number N can be calculated using prime power factorization.

Let , N = P1^a1 \* P2^a2 \* P3^a3 \* ... \* Pn^ak , s the prime power factorization of a number N , where P is the prime number and

a is number of times occurs that prime number.

Then , SOD(N) defines as :

// SOD(N) = (P1^0 + P1^1 + p1^2 + ... + P1^a1) \* (P2^0 + P2^1 + P2^2 + ... + P2^a2) \* (Pk^0 + Pk^1 + ... + Pk^ak)

5 : Logarithm base calculation :

//logB(x)=logC(x) / logC(B)

6 : Trailing zeros in N! in decimal number system ,

Let , a is frequency of 2 in N! prime factorisation and b is frequency of 5 in N! prime factorisation..

Then ,

// Number of Trailing zeros = min(a , b) ;

7 : Trailing zeros in N! in different base system :

We find number of trailing zero using the following steps:

Factorize the base B

If B = pa11 × pa22…× pakk, then find out occurance of xi=factorialPrimePower(pi).

But we can’t use xi directly. In order to create B we will need to combine each pi into paii. So we divide each xi by ai.

Number of trailing zero is MIN(x1,x2,…,xk).

8 : Leading Numbers :

We need to execute the following steps to find the first K leading digits of a number x ( in our problem x=N! ):

Find the log value of the number whose leading digits we are seeking. y=log10(x).

Decompose y into two parts. Integer part p and fraction part q.

The answer is ⌊10q×10K−1⌋.

9 : Euler Phi Extension Theorem

Theorem:

Given a number N, let d be a divisor of N. Then the number of pairs a,N, where 1≤a≤N and gcd(a,N)=d, is ϕ(N / d).

10 : Euler Phi Divisor Sum Theorem

Theorem:

For a given integer N, the sum of Euler Phi of each of the divisors of N equals to N, i.e, N = ∑d | N ϕ(d)

11 : Eulers Totient Function (Eulers Phi):

Given an integer N, how many numbers less than or equal N are there such that they are coprime to N?

A number X is coprime to N if gcd(X,N) = 1.

// ϕ(n) = n × ((p1 − 1) / p1) × ((p2 − 1) / p2) … × ((pk − 1) / pk)

12 : Given two sequences of numbers A=[a1,a2,…,an] and M=[m1,m2,…,mn], find the smallest solution to the following linear congruence equations if it exists:

x ≡ a1(mod m1)

x ≡ a2(mod m2)

…

x ≡ an(mod mn)

13 : GCD Sum Function – g(n)

Given a positive integer N, find the value of g(N), where

g(n) = gcd(1,n) + gcd(2,n) + gcd(3,n) +⋯+ gcd(n,n) = i=1 to n ∑ gcd(i,n)

// there is a direct formula for calculating the value of g(n).

If the prime factorization of n is p1^a1 × p2^a2 × … \*pk^ak, then

g(n) = i=0 to k ∏ ((ai + 1)pi^ai) – (ai \* pi^(ai−1))

14 : Sum of Co-prime Numbers of an Integer

Problem

Given a number N, find the sum of all numbers less than or equal to N that are co-prime with N.

Let us define a function f(n), which gives us sum of all numbers less than or equal to n that are co-prime to n.

Then we can calculate the value of f(n) with the following formula:

// f(n)= (phi(n) \* n) / 2

Eulers Phi Properties..

1. If p is a prime number, then gcd(p,q)=1 for all 1≤q<p. Therefore we have:

ϕ(p)=p−1.

2. If p is a prime number and k≥1, then there are exactly pk/p numbers between 1 and pk that are divisible by p. Which gives us:

ϕ(p^k)=p^k−p^k−1.

3. If a and b are relatively prime, then:

ϕ(ab)=ϕ(a)⋅ϕ(b).

4. In general, for not coprime a and b, the equation

ϕ(ab)=ϕ(a)⋅ϕ(b)⋅dϕ(d)

with d=gcd(a,b) holds.

5.

**Fibonacci properties...**

**1 . Cassini's identity:**

**(Fn−1)\*(Fn+1) − F2n= (−1)^n**

**2 . The "addition" rule:**

**Fn+k=(Fk \* Fn+1) + (Fk−1 \* Fn)**

**3 . Applying the previous identity to the case k=n, we get:**

**F2n= Fn \* (Fn+1 + Fn−1)**

**4 . From this we can prove by induction that for any positive integer k, Fnk is multiple of Fn.**

**5 . The inverse is also true: if Fm is multiple of Fn, then m is multiple of n**

**6 . GCD identity:**

GCD(Fm,Fn) = FGCD(m,n)

Codes

ll POW(ll a, ll b, ll mod)

{

a %= mod;

ll r = 1;

for(ll i = b; i > 0; i >>= 1)

{

if(i & 1)

r = (r \* a) % mod;

a = (a \* a) % mod;

}

return r;

}

ll f[N];

ll nCr(ll n, ll r)

{

if(n < r)

return 0;

return f[n] \* POW(f[n - r] \* f[r], mod - 2, mod) % mod;

}

ll nPr(ll n, ll r)

{

return nCr(n, r) \* f[r] % mod;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*Number of divisor\*\*\*\*\*\*\*\*

void NOD(ll n)

{

ll nod=1;

for(ll i=0; prime[i]\*prime[i]<=n; i++)

{

ll cnt=1;

if(n%prime[i]==0)

{

while(n%prime[i]==0)

{

n/=prime[i];

cnt++;

}

}

nod\*=cnt;

}

if(n!=1)nod\*=2;

return nod;

}

**\*\*\*\*\*\*\*\*\*\*\*Sum of Divisor\*\*\*\*\*\*\*\*\*\***

void SOD(ll n)

{

ll ans=1;

for(ll i=0; prime[i]\*prime[i]<=n; i++)

{

if(n%prime[i]==0)

{

ll temsum=1,p=1;

while(n%prime[i]==0)

{

n/=prime[i];

primefac.pb(prime[i]);

p\*=prime[i];

temsum+=p;

}

ans\*=temsum;

}

}

if(n!=1)

ans\*=(n+1);

return ans;

}

\*\*\*\*\*\*\*\*\*\*Eluers Phi\*\*\*\*\*\*\*\*\*\*\*\*

void EluersPhiSum()

{

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

phi[i] = i;

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

{

if(phi[i]==i)

{

for(int j=i; j<=mxn; j+=i)

phi[j]-=phi[j]/i;

}

}

prephi[1]=0;

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

{

prephi[i]=prephi[i-1]+(phi[i]\*phi[i]);

}

}

\*\*\*\*\*\*\*\*\*\*Eluers Phi\*\*\*\*\*\*\*\*\*\*\*\*

void Eluersphi(ll n)

{

for(ll i=0; i<=mxn; i++)isp[i]=0;

for(ll i=4; i<=mxn; i+=2)isp[i]=1;

isp[0] = isp[1] = 1;

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

{

if(isp[i]==0)

{

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

isp[j]=1;

}

}

prime.pb(2);

for(ll i=3; i<=mxn; i+=2)

{

if(isp[i]==0)

prime.pb(i);

}

ll ans=n;

for(ll i=0; prime[i]\*prime[i]<=n&&i<prime.size(); i++)

{

if(n%prime[i]==0)

{

while(n%prime[i]==0)

{

n/=prime[i];

}

ans/=prime[i];

ans\*=(prime[i]-1);

}

}

if(n>1)

{

ans/=n;

ans\*=(n-1);

}

return ans;

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Bit Masking\*\*\*\*\*\*\*\*\*\*\*\*\***

void BitMasking\_InclusionExclusion(ll n)

{

ll mask=1<<m;

vector<pair<ll,ll>>v;

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

{

ll lcm=1,cnt=0;

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

{

if(i&(1<<j))

{

cnt++;

lcm=LCM(lcm,a[j]);

}

}

v.pb({cnt,lcm});

}

ll ans=n;

for(ll i=0; i<v.size(); i++)

{

if(v[i].first%2!=0)

ans-=n/v[i].second;

else

ans+=n/v[i].second;

}

**}**

**Number of co-prime pairs from an array using inclu-eclu**

**const ll mx=1e6;**

ll isp[mx+5];

vector<ll>prime;

vector<ll>v[100010];

void siv()

{

for(ll i=4; i<=mx; i+=2)isp[i]=1;

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

{

if(isp[i]==0)

{

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

isp[j]=1;

}

}

prime.pb(2);

for(ll i=3; i<=mx; i+=2)

{

if(isp[i]==0)

prime.pb(i);

}

}

ll fr[mx+2];

int main()

{

fast

siv();

ll n;

long long ans=0;

cin>>n;

ll a[n+2];

for(ll i=0; i<n; i++)cin>>a[i];

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

{

for(ll j=0; prime[j]\*prime[j]<=a[i]&&j<prime.size(); j++)

{

if(a[i]%prime[j]==0)

{

v[i].pb(prime[j]);

while(a[i]%prime[j]==0)

{

a[i]/=prime[j];

}

}

}

if(a[i]>1)

v[i].pb(a[i]);

}

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

{

if(v[i].size()==0)continue;

ll k=1<<v[i].size();

for(ll mask=0; mask<=k; mask++)

{

long long x=1;

for(ll j=0; j<v[i].size(); j++)

{

if(mask&(1<<j))

{

x\*=v[i][j];

}

}

if(x!=1)

fr[x]++;

}

}

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

{

if(v[i].size()==0)

{

ans+=(n-1);

continue;

}

ll k=1<<v[i].size();

long long an=n;

for(ll mask=0; mask<=k; mask++)

{

ll x=1,cnt=0;

for(ll j=0; j<v[i].size(); j++)

{

if(mask&(1<<j))

{

cnt++;

x\*=v[i][j];

}

}

if(cnt%2!=0)

an-=fr[x];

else an+=fr[x];

}

ans+=an;

}

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

// for(ll j=0;j<v[i].size();j++)

// cout<<i<<in<<v[i][j]<<endl;

cout<<ans/2<<endl;

return 0;

**}**

**\*\*\*\*\*\*\*\*\*\*\*\* String sum \*\*\*\*\*\*\*\*\*\*\*\*\*\***

// works in O(n) ..

string BigintSum(string &str1, string &str2)

{

if((int)str1.size() < (int)str2.size())

swap(str1, str2) ;

int len1 = (int)str1.size() ;

int len2 = (int)str2.size() ;

int n = len2 ;

int carry = 0 ;

string sum ;

for(int i = len1 - 1, j = len2 - 1 ; i >= 0 ; i--, j--)

{

int x ;

if(n)

{

x = str1[i] - '0' + str2[j] - '0' + carry ;

n-- ;

}

else

x = str1[i] - '0' + carry ;

carry = x / 10 ;

x = x % 10 ;

sum.push\_back(x +'0') ;

}

if(carry) sum.push\_back(carry + '0') ;

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

return sum ;

}

**\*\*\*\*\*\*\*\* String Multiplication \*\*\*\*\*\*\*\*\*\*\***

**//take to string as numbers and returns the product of these two numbers to strings**

**// On worst case BigintProd take O(n^2) complexity**

string BigintProd(string &str1, string &str2)

{

if((int)str1.size() < (int)str2.size())

swap(str1, str2) ;

int len1 = (int)str1.size() ;

int len2 = (int)str2.size() ;

int carry = 0, cnt = 1 ;

string prod, pro ;

for(int i = len2 - 1 ; i >= 0 ; i--)

{

carry = 0 ;

for(int j = len1 - 1 ; j >=0 ; j--)

{

int x = (str2[i] - '0') \* (str1[j] - '0') + carry ;

carry = x / 10 ;

x = x % 10 ;

prod.push\_back( x+'0') ;

}

if(carry) prod.push\_back(carry +'0') ;

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

if(cnt == 1)

{

pro = prod ;

cnt++ ;

prod.clear() ;

}

else

{

pro = BigintSum(pro, prod) ;

prod.clear() ;

}

for(int k = len2 - 1 ; k >= i ; k--)

prod.push\_back('0') ;

}

//return "0" if any multiply apply with 0

if(pro[0] == '0' && pro.size() > 1) pro = "0" ;

return pro ;

}

**\*\*\*\*\*\*\*\*\*\* all base to decimal \*\*\*\*\*\*\*\***

**long long kick(string s,long long m)**

{

long long ans=0;

for(char v:s)

{

ans\*=m;

ans+=v-'0';

}

return ans;

}

// Returns the maximum value that

// can be put in a knapsack of capacity W

int knapSack(int W, int wt[], int val[], int n)

{

// Base Case

if (n == 0 || W == 0)

return 0;

// If weight of the nth item is more

// than Knapsack capacity W, then

// this item cannot be included

// in the optimal solution

if (wt[n - 1] > W)

return knapSack(W, wt, val, n - 1);

// Return the maximum of two cases:

// (1) nth item included

// (2) not included

else

return max(

val[n - 1]

+ knapSack(W - wt[n - 1],

wt, val, n - 1),

knapSack(W, wt, val, n - 1));

}

TEMPLATES

#include<bits/stdc++.h>

using namespace std;

typedef long long int ll;

typedef long double ld;

typedef vector<ll> vll;

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

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

using namespace \_\_gnu\_pbds;

typedef tree<ll, null\_type, less<ll>, rb\_tree\_tag, tree\_order\_statistics\_node\_update> ordered\_set;

#define fi first

#define se second

#define pb push\_back

#define Fin freopen("input.txt","r",stdin)

#define Fout freopen("output.txt","w",stdout)

#define nn "\n"

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

#define zz(v) (ll)v.size()

#define ss ' '

#define pii pair<ll,ll>

#define gcd(a,b) \_\_gcd(a,b)

#define lcm(a,b) (a\*b)/gcd(a,b)

\_\_builtin\_popcount(x)

\_\_builtin\_clz(x)

\_\_builtin\_ctz(x)

int main()

{

// ios::sync\_with\_stdio(0);

// cin.tie(0);

/\* you should actually read the stuff at the bottom \*/

}

\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**#define ll long long**

**const ll N = 1e5+7;**

ll ar[N];

struct st

{

#define ln (n<<1)

#define rn ((n<<1)+1)

ll t[4\*N],lazy[4\*N];

st()

{

memset(t,0,sizeof(t));

memset(lazy,0,sizeof(lazy));

}

inline void push(ll n,ll l,ll r)

{

if(lazy[n]==0)

return;

t[n] = t[n] + lazy[n]\*(r-l+1);

if(l!=r)

{

lazy[ln] = lazy[ln] + lazy[n];

lazy[rn] = lazy[rn] + lazy[n];

}

lazy[n]=0;

}

inline ll combine(ll a,ll b)

{

return a+b;

}

inline void pull(ll n)

{

t[n] = t[ln] + t[rn];

}

void build(ll n,ll l,ll r)

{

if(l==r)

{

t[n] = ar[l];

lazy[n] =0;

return ;

}

ll mid = (l+r)>>1;

build(ln,l,mid);

build(rn,mid+1,r);

pull(n);

}

void update(ll n,ll l,ll r,ll lin,ll rin,ll val)

{

push(n,l,r);

if(l>rin||r<lin)

return;

if(l>=lin&&r<=rin)

{

lazy[n] = val;

push(n,l,r);

return ;

}

ll mid = (l+r)>>1;

update(ln,l,mid,lin,rin,val);

update(rn,mid+1,r,lin,rin,val);

pull(n);

}

ll query(ll n,ll l,ll r,ll lin,ll rin)

{

push(n,l,r);

if(l>rin||r<lin)

return 0;

if(l>=lin&&r<=rin)

{

return t[n] ;

}

ll mid = (l+r)>>1;

return combine(query(ln,l,mid,lin,rin),query(rn,mid+1,r,lin,rin));

}

}t;

int main()

{

ios\_base::sync\_with\_stdio(0);

cin.tie(0);

ll tst;

// cin>>tst;

ll ts=0;

while(tst)

{

ts++;

ll n,q;

cin>>n>>q;

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

cin>>ar[i];

ll f,a,b,c;

t.build(1,0,n-1);

while(q--)

{

cin>>f;

if(f==1)

{

cin>>a>>b>>c;

t.update(1,0,n-1,a,b-1,c);

}

else

{

cin>>a;

cout<< t.query(1,0,n-1,a-1,a-1)<<endl;

}

}

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Belman ford :

const int N = 3e5 + 9;

struct st {

int a, b, cost;

} e[N];

const int INF = 2e9;

int32\_t main() {

int n, m;

cin >> n >> m;

for(int i = 0; i < m; i++) cin >> e[i].a >> e[i].b >> e[i].cost;

int s;

cin >> s;//is there any negative cycle which is reachable from s?

vector<int> d (n, INF);//for finding any cycle(not necessarily from s) set d[i] = 0 for all i

d[s] = 0;

vector<int> p (n, -1);

int x;

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

x = -1;

for (int j=0; j<m; ++j) {

if (d[e[j].a] < INF) {

if (d[e[j].b] > d[e[j].a] + e[j].cost) {

d[e[j].b] = max (-INF, d[e[j].a] + e[j].cost);//for overflow

p[e[j].b] = e[j].a;

x = e[j].b;

}

}

}

}

if (x == -1) cout << "No negative cycle from "<<s;

else {

int y = x; //x can be on any cycle or reachable from some cycle

for (int i=0; i<n; ++i) y = p[y];

vector<int> path;

for (int cur=y; ; cur=p[cur]) {

path.push\_back (cur);

if (cur == y && path.size() > 1) break;

}

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

cout << "Negative cycle: ";

for (int i=0; i<path.size(); ++i) cout << path[i] << ' ';

}

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Smallest Prime Factor(SPF):**

const int N = 1e6 + 9;

int spf[N];

int32\_t main() {

ios\_base::sync\_with\_stdio(0);

cin.tie(0);

for (int i = 2; i < N; i++) {

spf[i] = i;

}

for (int i = 2; i < N; i++) {

for (int j = i; j < N; j += i) {

spf[j] = min(spf[j], i);

}

}

int q; cin >> q; // queries q <= 1e6

while (q--) {

int n; cin >> n; // find prime factorization of n <= 1e6

vector<int> ans;

while (n > 1) {

ans.push\_back(spf[n]);

n /= spf[n];

}

for (auto x: ans) cout << x << ' '; cout << '\n';

}

return 0;

}

**LCAAAAAA**

**#include <bits/stdc++.h>**

using namespace std;

// (neighbor, weight)

vector <pair <int, int>> g[100010];

int h[100010], par[20][100010], sub[100010];

// par[i][j] --> node j er 2^i tomo ancestor ke?

// 2^0 tomo == 1st

void go (int u = 1, int p = -1, int far = 0) {

par[0][u] = p, h[u] = far, sub[u] = 1;

for (auto [v, w] : g[u]) if (v != p) {

go(v, u, far + 1);

sub[u] += sub[v];

ans += w \* sub[v] \* (n - sub[v]);

}

}

int LCA (int u, int v) {

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

for (int i = 19; i >= 0; --i) {

// u er 2^i tomo parent ke

int anc = par[i][u];

if (anc != -1 and h[anc] >= h[v]) {

u = anc;

}

}

// ekhon u ar v same height

for (int i = 19; i >= 0; --i) {

int u\_anc = par[i][u];

int v\_anc = par[i][v];

if (u\_anc != -1 and u\_anc != v\_anc) {

u = u\_anc, v = v\_anc;

}

}

// ekhon u ar v duitai lca er next level tay

return par[0][u];

}

int maxOnPath (int u, int v) {

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

int result = INT\_MIN;

for (int i = 19; i >= 0; --i) {

// u er 2^i tomo parent ke

int anc = par[i][u];

if (anc != -1 and h[anc] >= h[v]) {

result = max(result, mx[i][u]);

u = anc;

}

}

// ekhon u ar v same height

for (int i = 19; i >= 0; --i) {

int u\_anc = par[i][u];

int v\_anc = par[i][v];

if (u\_anc != -1 and u\_anc != v\_anc) {

result = max(result, mx[i][u]);

result = max(result, mx[i][v]);

u = u\_anc, v = v\_anc;

}

}

// ekhon u ar v duitai lca er next level tay

result = max(result, mx[0][u]);

result = max(result, mx[0][v]);

return result;

}

int main() {

memset(par, -1, sizeof par);

go();

for (int i = 1; i < 20; ++i) {

// par[i][j]

// par[0][j]...

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

// node j er 2^i tomo parent ke?

// node j er 2^(i-1) tomo parent je, tar 2^(i-1) tomo parent

if (par[i - 1][j] != -1) {

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

mx[i][j] = max(mx[i-1][j], mx[i-1][par[i-1][j]]);

}

}

}

return 0;

}

**Graph implemented from array**

#include <bits/stdc++.h>

using namespace std;

vector <int> g[100010];

int arr[100010], par[20][100010];

int LCA (int u, int v) {

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

for (int i = 19; i >= 0; --i) {

// u er 2^i tomo parent ke

int anc = par[i][u];

if (anc != -1 and h[anc] >= h[v]) {

u = anc;

}

}

// ekhon u ar v same height

for (int i = 19; i >= 0; --i) {

int u\_anc = par[i][u];

int v\_anc = par[i][v];

if (u\_anc != -1 and u\_anc != v\_anc) {

u = u\_anc, v = v\_anc;

}

}

// ekhon u ar v duitai lca er next level tay

return par[0][u];

}

bool vis[100010];

void dfs () {

}

int main() {

stack <int> stk;

// i theke kothay jawa jay

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

while (!stk.empty() and arr[stk.top()] <= arr[i]) stk.pop();

if (stk.empty()) par[0][i] = -1;

else {

par[0][i] = stk.top();

g[par[0][i]].emplace\_back(i);

}

stk.push(i);

}

memset(par, -1, sizeof par);

go();

for (int i = 1; i < 20; ++i) {

// par[i][j]

// par[0][j]...

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

// node j er 2^i tomo parent ke?

// node j er 2^(i-1) tomo parent je, tar 2^(i-1) tomo parent

if (par[i - 1][j] != -1) {

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

}

}

}

return 0;

}

**#fairy**

#include <bits/stdc++.h>

using namespace std;

int ans, numberOfBadEdges, height[101010], counter[101010];

void go (int u = 1, int par = -1) {

for (int v : g[u]) if (v != par) {

go(v, u);

counter[u] += counter[v];

if (counter[v] == numberOf

BadEdges) {

++ans;

}

}

}

int main() {

for (auto [u, v] : bad\_edges) {

if (height[u] > height[v]) swap(u, v);

// u upore, v niche

++counter[v], --counter[u];

}

go();

return 0;

}

**/////////////////////////////////////////////////////////////////////////////////////////////**

**Mo’s algorithm by nirjhor vai**

#include <bits/stdc++.h>

using namespace std;

const int B = 450;

const int N = 200005;

tuple <int, int, int, int> queries[N];

int n, q, a[N], ans[N], freq[N], distinct;

void add (int pos) {

int x = a[pos];

if (freq[x] == 0) ++distinct;

++freq[x];

}

void remove (int pos) {

int x = a[pos];

--freq[x];

if (freq[x] == 0) --distinct;

}

int main() {

cin >> n >> q;

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

scanf("%d", a + i);

}

vector <int> coo(a, a + n);

sort(coo.begin(), coo.end());

coo.erase(unique(coo.begin(), coo.end()), coo.end());

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

a[i] = lower\_bound(coo.begin(), coo.end(), a[i]) - coo.begin();

}

for (int i = 0; i < q; ++i) {

int l, r;

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

--l, --r;

int block = l / B;

queries[i] = make\_tuple(block, (block & 1) ? -r : r, l, i);

}

// Mo

sort(queries, queries + q);

int L = 0, R = -1;

for (int it = 0; it < q; ++it) {

auto [block, r, l, i] = queries[it];

r = abs(r);

// block --> [l, r] jetar id i

while (R < r) add(++R);

while (L > l) add(--L);

while (L < l) remove(L++);

while (R > r) remove(R--);

// L == l, R == r

ans[i] = distinct;

}

for (int i = 0; i < q; ++i) printf("%d\n", ans[i]);

return 0;

}

**////////////////////////////////////////**

**Hasing code :**

#include <bits/stdc++.h>

using namespace std;

const int B = 29;

const int N = 100010;

const int MOD = 1e9 + 7;

// power[i] = B^i

// inverse[i] = B^(-i)

// 29^(-1) mod 1e9 + 7

long long power[N], inverse[N], INV\_B;

// a^e (mod MOD)

long long bigMod (long long a, long long e) {

if (e == 0) return 1;

if (e == 1) return a;

if (e % 2 == 0) {

long long temp = bigMod(a, e / 2);

return temp \* temp % MOD;

}

return a \* bigMod(a, e - 1) % MOD;

}

int main() {

power[0] = 1;

for (int i = 1; i < N; ++i) {

power[i] = power[i - 1] \* B % MOD;

}

INV\_B = bigMod(B, MOD - 2);

inverse[0] = 1;

for (int i = 1; i < N; ++i) {

inverse[i] = inverse[i - 1] \* INV\_B % MOD;

}

string text, pattern;

cin >> text >> pattern;

long long pattern\_hash = 0;

// p[0] \* power[0] + p[1] \* power[1] + p[2] \* power[2] + p[3] \* power[3] + ...

for (int i = 0; i < pattern.size(); ++i) {

pattern\_hash = (pattern\_hash + (pattern[i] - 'a' + 1) \* power[i]) % MOD;

}

long long text\_hash = 0;

for (int i = 0; i < pattern.size(); ++i) {

text\_hash = (text\_hash + (text[i] - 'a' + 1) \* power[i]) % MOD;

}

// starting position of pattern in text

for (int i = 0; i <= (int) text.size() - (int) pattern.size(); ++i) {

// text[i.....i+pattern\_size-1]

if (text\_hash == pattern\_hash) {

cout << "match found starting at " << i << '\n';

}

text\_hash -= text[i] - 'a' + 1;

text\_hash = text\_hash \* INV\_B % MOD;

if (i + pattern.size() < text.size()) {

text\_hash = (text\_hash + (text[i + pattern.size()] - 'a' + 1) \* power[pattern.size() - 1]) % MOD;

}

if (text\_hash < 0) text\_hash += MOD;

}

return 0;

}

**Hasing code 2:**

#include <bits/stdc++.h>

using namespace std;

const int B = 29;

const int N = 100010;

const int MOD = 1e9 + 7;

// power[i] = B^i

// inverse[i] = B^(-i)

// 29^(-1) mod 1e9 + 7

long long power[N], inverse[N], prefix[N], INV\_B;

// text[l] \* B^l + text[l+1] \* B^(l+1) + ... + text[r] \* B^r

long long rangeHash (int l, int r) {

long long ret = prefix[r + 1] - prefix[l];

ret = ret \* inverse[l] % MOD;

if (ret < 0) ret += MOD;

return ret;

}

// a^e (mod MOD)

long long bigMod (long long a, long long e) {

if (e == 0) return 1;

if (e == 1) return a;

if (e % 2 == 0) {

long long temp = bigMod(a, e / 2);

return temp \* temp % MOD;

}

return a \* bigMod(a, e - 1) % MOD;

}

int main() {

power[0] = 1;

for (int i = 1; i < N; ++i) {

power[i] = power[i - 1] \* B % MOD;

}

INV\_B = bigMod(B, MOD - 2);

inverse[0] = 1;

for (int i = 1; i < N; ++i) {

inverse[i] = inverse[i - 1] \* INV\_B % MOD;

}

string text;

cin >> text;

prefix[0] = 0;

for (int i = 0; i < text.size(); ++i) {

prefix[i + 1] = (prefix[i] + (text[i] - 'a' + 1) \* power[i]) % MOD;

}

int q; cin >> q;

while (q--) {

int i, j, l;

cin >> i >> j >> l;

if (rangeHash(i, i + l - 1) == rangeHash(j, j + l - 1)) {

cout << "MATCH\n";

} else {

cout << "nope\n";

}

}

return 0;

}

**Co\_ordinate Compression :**

#include<bits/stdc++.h>

using namespace std;

int32\_t main() {

ios\_base::sync\_with\_stdio(0);

cin.tie(0);

vector<int> a({100, 9, 10, 10, 9});

vector<int> v = a;

sort(v.begin(), v.end());

// [9, 9, 10, 10, 100]

v.resize(unique(v.begin(), v.end()) - v.begin());

// unique -> [9, 10, 100, 9, 10]

// v.resize(3) -> [9, 10, 100]

for (int i = 0; i < a.size(); i++) {

a[i] = lower\_bound(v.begin(), v.end(), a[i]) - v.begin() + 1;

cout << a[i] << ' ';

}

// v[a[i] - 1] is the original value of a[i]

return 0;

}