// **CRT**

ll modular(ll base, ll exp, ll mod)

{

ll res = 1;

while (exp > 0)

{

if (exp % 2 == 1)

res= (res \* base) % mod;

exp = exp >> 1;

base = (base \* base) % mod;

}

return res;

}

ll modinv(ll a,ll p)

{

return modular(a,p-2,p);

}

ll crt(vector<ll>& a,vector<ll>& m)

{

ll prod = 1;

for (auto i:m)

prod \*= i;

ll result = 0;

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

{

ll pp = prod / m[i];

result += a[i] \* modinv(pp, m[i]) \* pp;

}

return result % prod;

}

// **NPR , NCR**

#include<bits/stdc++.h>

using namespace std;

const int N = 1e6, mod = 1e9 + 7;

int power(long long n, long long k) {

int ans = 1 % mod; n %= mod; if (n < 0) n += mod;

while (k) {

if (k & 1) ans = (long long) ans \* n % mod;

n = (long long) n \* n % mod;

k >>= 1;

}

return ans;

}

int f[N], invf[N];

int nCr(int n, int r) {

if (n < r or n < 0) return 0;

return 1LL \* f[n] \* invf[r] % mod \* invf[n - r] % mod;

}

int nPr(int n, int r) {

if (n < r or n < 0) return 0;

return 1LL \* f[n] \* invf[n - r] % mod;

}

int32\_t main() {

f[0] = 1;

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

f[i] = 1LL \* i \* f[i - 1] % mod;

}

invf[N - 1] = power(f[N - 1], mod - 2);

for (int i = N - 2; i >= 0; i--) {

invf[i] = 1LL \* invf[i + 1] \* (i + 1) % mod;

}

cout << nCr(6, 2) << '\n';

cout << nPr(6, 2) << '\n';

return 0;

}

// **Grouping segment tree**

const int N=100001;

int n;

struct node{

   ll hsuff,hpref,subsum;

   ll totsum;

};

 vector<node>tree(4\*N);

ll arr[N+1];

node merge(node l,node r){

   if(l.subsum==INT\_MAX)return r;

   if(r.subsum==INT\_MAX)return l;

   node ans;

   ans.subsum=max(l.subsum,r.subsum);

   ans.subsum=max(ans.subsum,l.hsuff+r.hpref);

   ans.totsum=l.totsum+r.totsum;

   ans.hsuff=max(r.hsuff,r.totsum+l.hsuff);

   ans.hpref=max(l.hpref,l.totsum+r.hpref);

   return ans;

}

void built(int nodee,int l,int r){

   if(l==r){

      tree[nodee].hsuff=tree[nodee].hpref=tree[nodee].subsum=tree[nodee].totsum=arr[l];

   }else{

      int mid=(l+r)/2;

      int lft=2\*nodee,rgt=2\*nodee+1;

      built(lft,l,mid);

      built(rgt,mid+1,r);

      tree[nodee]=merge(tree[lft],tree[rgt]);

   }

}

void update(int nodee,int l,int r,int i,int j,int value){

  if(r<i || l>j)return;

  if(l>=i && r<=j){

    tree[nodee].hsuff=tree[nodee].hpref=tree[nodee].subsum=tree[nodee].totsum=value;

  }else{

     int mid=(l+r)/2;

      int lft=2\*nodee,rgt=2\*nodee+1;

      update(lft,l,mid,i,j,value);

      update(rgt,mid+1,r,i,j,value);

      tree[nodee]=merge(tree[lft],tree[rgt]);

  }

}

node search(int nodee,int l,int r,int i,int j){

   if(r<i || l>j){

      return {INT\_MAX,INT\_MAX,INT\_MAX,INT\_MAX};

   }

   if(l>=i && r<=j){

      return tree[nodee];

   }

   else{

      int mid=(l+r)/2;

      int lft=2\*nodee,rgt=2\*nodee+1;

      return merge(search(lft,l,mid,i,j),search(rgt,mid+1,r,i,j));

   }

}

int main()

{

cin>>n;

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

   cin>>arr[i];

}

built(1,0,n-1);

int q;

cin>>q;

while(q--){

   int type,a,b;

   cin>>type>>a>>b;

   if(type==1){

     node ans=search(1,0,n-1,a-1,b-1);

     cout<<ans.subsum<<endl;

   }else{

     update(1,0,n-1,a-1,a-1,b);

   }

}

}

**// persistent segment tree with binary search**

const int N=100001;

int arr[N+1];

vector<int>dp[4\*N+1];

vector<ll>dpr[4\*N+1];

void build(int node,int l,int r){

    if(l==r){

        dp[node].push\_back(arr[l]);

        dpr[node].push\_back(arr[l]);

        // cout<<arr[l]<<endl;

        // cout<<l<<" "<<r<<endl;

        // cout<<endl;

    }else{

        int mid=(l+r)/2;

       int left=2\*node,right=2\*node+1;

       build(left,l,mid);

       build(right,mid+1,r);

       int i=0,j=0;

       while(i<dp[left].size() && j<dp[right].size()){

        if(dp[left][i]<dp[right][j]){

            dp[node].push\_back(dp[left][i]);

            ++i;

        }else{

            dp[node].push\_back(dp[right][j]);

            ++j;

        }

       }

        while(j<dp[right].size()){

            dp[node].push\_back(dp[right][j]);

            ++j;

        }

        while(i<dp[left].size()){

            dp[node].push\_back(dp[left][i]);

            ++i;

       }

       dpr[node].push\_back(dp[node][0]);

       for(int i=1;i<dp[node].size();i++){

         ll c=dp[node][i]+dpr[node][i-1];

         dpr[node].push\_back(c);

       }

    }

}

ll ssearch(int node,int l,int r,int i,int j,ll value){

    if(l>j || r<i)return 0;

    if(l>=i && r<=j){

        ll ans=-1;

        int low=0,high=dp[node].size()-1;

        while(high>=low){

            int mid=(high+low)/2;

            if(dp[node][mid]<value){

                ans=mid;

                low=mid+1;

            }else high=mid-1;

        }

        if(ans==-1)return 0;

        else{

           return (value\*(ans+1))-dpr[node][ans];

        }

    }else{

        int mid=(r+l)/2;

        return 1ll\*(ssearch(2\*node,l,mid,i,j,value)+ssearch(2\*node+1,mid+1,r,i,j,value));

    }

}

int main()

{

   ios\_base:: sync\_with\_stdio(false);cin.tie(NULL);cout.tie(NULL);

int n,q;

cin>>n>>q;

for(int i=0;i<n;i++)cin>>arr[i];

build(1,0,n-1);

while(q--){

    ll value;

    int a,b;

    cin>>a>>b>>value;

    cout<<ssearch(1,0,n-1,a-1,b-1,value)<<endl;

}

}

// **Min segment tree**

vector<ll>ara(N),tree(4\*N),lazy[N];

void build(ll node,ll b,ll e)

{

if(b==e)

{

tree[node]= ara[b];

return;

}

ll left=node\*2;

ll right=node\*2+1;

ll mid=(b+e)/2;

build(left,b,mid);

build(right,mid+1,e);

tree[node]=min(tree[right],tree[left]);

}

ll query(ll node,ll b,ll e,ll i,ll j)

{

if(i>e || j<b)

return 1e9+7;

if(b>=i && e<=j)

return tree[node];

ll left=node\*2;

ll right=node\*2+1;

ll mid=(b+e)/2;

return min(query(left,b,mid,i,j),query(right,mid+1,e,i,j));

}

void update(ll node,ll b,ll e,ll i,ll value)

{

if(i>e || i<b)

return;

if(b>=i && e<=i)

{

tree[node]=value;

return;

}

ll left=node\*2;

ll right=node\*2+1;

ll mid=(b+e)/2;

update(left,b,mid,i,value);

update(right,mid+1,e,i,value);

tree[node]=min(tree[left],tree[right]);

}

int main()

{

int n,q,test,k,x,y;

cin>>n>>q;

rep(i,n)

{

cin>>ara[i];

}

build(1,1,n);

while(q--)

{

cin>>k>>x>>y;

if(!k)

update(1,1,n,x,y);

else

cout<<query(1,1,n,x,y)<<endl;

}

}

// **Centroid Decomposition**

const int N=2e5;

int n;

vector<int>tree[N+1];

vector<int>subtreesize(N+1,0);

vector<int>parent(N+1,0);

vector<int>segmentree(4\*N+1,0);

vector<int>nodeid(N+1,0);

vector<int>depth(N+1,0);

vector<int>boss(N+1,0);

void update(int node,int l,int r,int l1,int r1,int value)

{

if(r<l1 || r1<l)return;

if(l1<=l && r1>=r)

{

segmentree[node]=value;

}

else

{

int mid=(l+r)/2;

int left=2\*node;

int right=2\*node+1;

update(left,l,mid,l1,r1,value);

update(right,mid+1,r,l1,r1,value);

segmentree[node]=max(segmentree[left],segmentree[right]);

}

}

int query(int node,int l,int r,int l1,int r1)

{

if(r<l1 || r1<l)return INT\_MIN;

if(l1<=l && r1>=r)return segmentree[node];

int mid=(l+r)/2;

int left=2\*node;

int right=2\*node+1;

return max(query(left,l,mid,l1,r1),query(right,mid+1,r,l1,r1));

}

int dfs1(int node,int par,int dep)

{

subtreesize[node]++;

depth[node]=dep;

parent[node]=par;

for(int i=0; i<tree[node].size(); i++)

{

int ch=tree[node][i];

if(ch!=par)

{

subtreesize[node]+=dfs1(ch,node,dep+1);

}

}

return subtreesize[node];

}

void dfs2(int node,int parent,int top,int &ct)

{

boss[node]=top;

nodeid[node]=ct++;

int heavychild=-1,heavysize=-1;

for(int i=0; i<tree[node].size(); i++)

{

int ch=tree[node][i];

if(ch!=parent)

{

if(heavysize<subtreesize[ch])

{

heavychild=ch;

heavysize=subtreesize[ch];

}

}

}

if(heavychild==-1)return;

dfs2(heavychild,node,top,ct);

for(int i=0; i<tree[node].size(); i++)

{

int ch=tree[node][i];

if(ch!=parent && ch!=heavychild)

{

dfs2(ch,node,ch,ct);

}

}

}

int bahirkoro(int node1,int node2)

{

int mx=INT\_MIN;

while(boss[node1]!=boss[node2])

{

if(depth[boss[node1]]<depth[boss[node2]])swap(node1,node2);

mx=max(mx,query(1,0,n-1,nodeid[boss[node1]],nodeid[node1]));

node1=parent[boss[node1]];

}

if(depth[node1]>depth[node2])swap(node1,node2);

mx=max(mx,query(1,0,n-1,nodeid[node1],nodeid[node2]));

return mx;

}

int main()

{

int q;

cin>>n>>q;

vector<int>values(n+1,0);

for(int i=1; i<=n; i++)cin>>values[i];

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

{

int a,b;

cin>>a>>b;

tree[a].push\_back(b);

tree[b].push\_back(a);

}

int count=0;

int dep=0;

dfs1(1,1,dep);

dfs2(1,1,1,count);

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

{

update(1,0,n-1,nodeid[i],nodeid[i],values[i]);

}

while(q--)

{

int t;

cin>>t;

if(t==1)

{

int c,val;

cin>>c>>val;

update(1,0,n-1,nodeid[c],nodeid[c],val);

}

else

{

int a,b;

cin>>a>>b;

cout<<bahirkoro(a,b)<<" ";

}

}

}

**// Diakstra**

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

q.push({0,1});

vector<ll>dis(n+1,INT\_MAX);

dis[1]=0;

vector<pair<int,ll>>arr[n+1];

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

int a,b,w;

cin>>a>>b>>w;

arr[a].push\_back({b,w});

}

while(!q.empty()){

int node=q.top().second;

ll ww=abs(q.top().first);

q.pop();

if(dis[node]<ww)continue;

for(int i=0;i<arr[node].size();i++){

int ch=arr[node][i].first;

ll wt=arr[node][i].second;

if(dis[node]+wt<dis[ch]){

dis[ch]=dis[node]+wt;

q.push({-dis[ch],ch});

}

}

}

// **Floyd Warshall**

int nV;

// Implementing floyd warshall algorithm

void floydWarshall(vector<vector<ll>>&matrix) {

int i, j, k;

// Adding vertices individually

for (k = 0; k < nV; k++) {

for (i = 0; i < nV; i++) {

for (j = 0; j < nV; j++) {

if (matrix[i][k] + matrix[k][j] < matrix[i][j])

matrix[i][j] = matrix[i][k] + matrix[k][j];

}

}

}

}

int main() {

int m,q;

cin>>nV>>m>>q;

vector<vector<ll>>matrix(nV+1,vector<ll>(nV+1,INT\_MAX));

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

int a,b;

ll w;

cin>>a>>b>>w;

--a,--b;

ll val1=matrix[a][b],val2=matrix[b][a];

matrix[a][b]=min(val1,w);

matrix[b][a]=min(val2,w);

}

for(int i=0;i<nV;i++)matrix[i][i]=0;

floydWarshall(matrix);

while(q--){

int a,b;

cin>>a>>b;

--a,--b;

ll val=min(matrix[a][b],matrix[b][a]);

if(val==INT\_MAX)cout<<-1<<endl;

else

cout<<matrix[a][b]<<endl;

}

}

**// DSU**

void make(ll n)

{

par[n]=n;

sz[n]=1;

}

ll rep(ll n)

{

if(n==par[n])return n;

return par[n]=rep(par[n]);

}

void add(ll a,ll b)

{

a=rep(a);

b=rep(b);

if(a!=b)

{

if(sz[a]<sz[b])

{

swap(a,b);

}

par[b]=a;

sz[a]+=sz[b];

}

}

**// Lazy Propagation**

void updateRange(int node, int start, int end, int l, int r, int val)

{

if(lazy[node] != 0)

{

// This node needs to be updated

tree[node] += (end - start + 1) \* lazy[node]; // Update it

if(start != end)

{

lazy[node\*2] += lazy[node]; // Mark child as lazy

lazy[node\*2+1] += lazy[node]; // Mark child as lazy

}

lazy[node] = 0; // Reset it

}

if(start > end or start > r or end < l) // Current segment is not within range [l, r]

return;

if(start >= l and end <= r)

{

// Segment is fully within range

tree[node] += (end - start + 1) \* val;

if(start != end)

{

// Not leaf node

lazy[node\*2] += val;

lazy[node\*2+1] += val;

}

return;

}

int mid = (start + end) / 2;

updateRange(node\*2, start, mid, l, r, val); // Updating left child

updateRange(node\*2 + 1, mid + 1, end, l, r, val); // Updating right child

tree[node] = tree[node\*2] + tree[node\*2+1]; // Updating root with max value

}

int queryRange(int node, int start, int end, int l, int r)

{

if(start > end or start > r or end < l)

return 0; // Out of range

if(lazy[node] != 0)

{

// This node needs to be updated

tree[node] += (end - start + 1) \* lazy[node]; // Update it

if(start != end)

{

lazy[node\*2] += lazy[node]; // Mark child as lazy

lazy[node\*2+1] += lazy[node]; // Mark child as lazy

}

lazy[node] = 0; // Reset it

}

if(start >= l and end <= r) // Current segment is totally within range [l, r]

return tree[node];

int mid = (start + end) / 2;

int p1 = queryRange(node\*2, start, mid, l, r); // Query left child

int p2 = queryRange(node\*2 + 1, mid + 1, end, l, r); // Query right child

return (p1 + p2);

}

**// Lowest Common Ansestor**

#include <bits/stdc++.h>

using namespace std;

#define MAXN 100000

#define level 18

vector <int> tree[MAXN];

int depth[MAXN];

int parent[MAXN][level];

// pre-compute the depth for each node and their

// first parent(2^0th parent)

// time complexity : O(n)

void dfs(int cur, int prev)

{

depth[cur] = depth[prev] + 1;

parent[cur][0] = prev;

for (int i=0; i<tree[cur].size(); i++)

{

if (tree[cur][i] != prev)

dfs(tree[cur][i], cur);

}

}

// Dynamic Programming Sparse Matrix Approach

// populating 2^i parent for each node

// Time complexity : O(nlogn)

void precomputeSparseMatrix(int n)

{

for (int i=1; i<level; i++)

{

for (int node = 1; node <= n; node++)

{

if (parent[node][i1] != -1)

parent[node][i] =

parent[parent[node][i-1]][i-1];

}

}

}

**// Returning the LCA of u and v**

// Time complexity : O(log n)

int lca(int u, int v)

{

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

swap(u, v);

int diff = depth[v] - depth[u];

// Step 1 of the pseudocode

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

if ((diff>>i)&1)

v = parent[v][i];

// now depth[u] == depth[v]

if (u == v)

return u;

// Step 2 of the pseudocode

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

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

{

u = parent[u][i];

v = parent[v][i];

}

return parent[u][0];

}

void addEdge(int u,int v)

{

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

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

}

// driver function

int main()

{

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

int n = 8;

addEdge(1,2);

addEdge(1,3);

addEdge(2,4);

addEdge(2,5);

addEdge(2,6);

addEdge(3,7);

addEdge(3,8);

depth[0] = 0;

depth

dfs(1,0);

for every node

precomputeSparseMatrix(n);

cout << "LCA(4, 7) = " << lca(4,7) << endl;

cout << "LCA(4, 6) = " << lca(4,6) << endl;

return 0;

}

**// Hashcode with interface PDBS**

#include<bits/stdc++.h>

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

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

using namespace \_\_gnu\_pbds;

using namespace std;

template <typename T> using o\_set = tree<T, null\_type, less<T>, rb\_tree\_tag, tree\_order\_statistics\_node\_update>;

//variation= ll less\_equal<ll> (change according to need) ordered\_multiset

// ordered\_set os declare like this

// os.insert(1) insert like this

// \*os.find\_by\_order(k) returns an iterator to the k-th largest element (counting from zero)

// os.order\_of\_key(X) returns the number of items in a set that are strictly smaller than X

// works like set and complexity O(logN) and saves in ascending order with input order index

//if error occurs= c:\mingw\lib\gcc\mingw32\6.3.0\include\c++\ext\pb\_ds\detail\resize\_policy\hash\_standard\_resize\_policy\_imp.hpp0000644. Rename that file to remove the 0000644 from the end of it.

#define long long int ll

#define endl '\n'

int main(){

ios\_base:: sync\_with\_stdio(false);cin.tie(NULL);cout.tie(NULL);

**}**

#include<bits/stdc++.h>

using namespace std;

const int N = 1e6 + 9;

const int p1 = 137, mod1 = 127657753, p2 = 277, mod2 = 987654319;

int power(long long n, long long k, int mod) {

int ans = 1 % mod; n %= mod; if (n < 0) n += mod;

while (k) {

if (k & 1) ans = (long long) ans \* n % mod;

n = (long long) n \* n % mod;

k >>= 1;

}

return ans;

}

int ip1, ip2;

pair<int, int> pw[N], ipw[N];

void prec() {

pw[0] = {1, 1};

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

pw[i].first = 1LL \* pw[i - 1].first \* p1 % mod1;

pw[i].second = 1LL \* pw[i - 1].second \* p2 % mod2;

}

ip1 = power(p1, mod1 - 2, mod1);

ip2 = power(p2, mod2 - 2, mod2);

ipw[0] = {1, 1};

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

ipw[i].first = 1LL \* ipw[i - 1].first \* ip1 % mod1;

ipw[i].second = 1LL \* ipw[i - 1].second \* ip2 % mod2;

}

}

pair<int, int> string\_hash(string s) {

int n = s.size();

pair<int, int> hs({0, 0});

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

hs.first += 1LL \* s[i] \* pw[i].first % mod1;

hs.first %= mod1;

hs.second += 1LL \* s[i] \* pw[i].second % mod2;

hs.second %= mod2;

}

return hs;

}

pair<int, int> pref[N];

void build(string s) {

int n = s.size();

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

pref[i].first = 1LL \* s[i] \* pw[i].first % mod1;

if (i) pref[i].first = (pref[i].first + pref[i - 1].first) % mod1;

pref[i].second = 1LL \* s[i] \* pw[i].second % mod2;

if (i) pref[i].second = (pref[i].second + pref[i - 1].second) % mod2;

}

}

pair<int, int> get\_hash(int i, int j) {

assert(i <= j);

pair<int, int> hs({0, 0});

hs.first = pref[j].first;

if (i) hs.first = (hs.first - pref[i - 1].first + mod1) % mod1;

hs.first = 1LL \* hs.first \* ipw[i].first % mod1;

hs.second = pref[j].second;

if (i) hs.second = (hs.second - pref[i - 1].second + mod2) % mod2;

hs.second = 1LL \* hs.second \* ipw[i].second % mod2;

return hs;

}

int32\_t main() {

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

cin.tie(0);

prec();

string a, b; cin >> a >> b;

build(a);

int ans = 0, n = a.size(), m = b.size();

auto hash\_b = string\_hash(b);

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

ans += get\_hash(i, i + m - 1) == hash\_b;

}

cout << ans << '\n';

return 0;

}

**// Trie**

int trie[MAXN][27];

class WordDictionary {

int it = 0;

public:

/\*\* Initialize your data structure here. \*/

WordDictionary() {

memset(trie, -1, sizeof trie);

}

/\*\* Adds a word into the data structure. \*/

void addWord(string word) {

int ptr = 0;

for (char c : word) {

int d = c - 'a';

if (trie[ptr][d] == -1) {

trie[ptr][d] = ++it; // create new node

}

ptr = trie[ptr][d];

}

trie[ptr][26] = 1; // set flag to mark end of word

}

bool search(string word) {

vector<int> q{0};

int i = 0;

for (char c : word) {

int sz = q.size();

for (; i < sz; i++) {

int u = q[i];

if (c == '.') {

for (int d = 0; d < 26; d++) {

if (trie[u][d] != -1) {

q.push\_back(trie[u][d]);

}

}

} else {

int d = c - 'a';

if (trie[u][d] != -1) {

q.push\_back(trie[u][d]);

}

}

}

}

for (; i < q.size(); i++) {

if (trie[q[i]][26] != -1) {

return true;

}

}

return false;

}

};

**// Matrix expotention**

**const long long mod=1e9+7;**

vector<vector<long long>> mul(vector<vector<long long>> &a,vector<vector<long long>> &b)

{

long long n=a.size(),m=b[0].size();

vector<vector<long long>> c(n,vector<long long>(m,0));

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

{

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

{

for(long long k=0;k<a[0].size();k++)

c[i][j]=(c[i][j]+(1ll\*a[i][k]\*b[k][j])%mod)%mod;

}

}

return c;

}

vector<vector<long long>> p(vector<vector<long long>> &a,long n)

{

vector<vector<long long>> ans(a.size(),vector<long long> (a[0].size(),0));

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

ans[i][i]=1;

while(n)

{

if(n&1)

ans=mul(ans,a);

a=mul(a,a);

n=n>>1;

}

return ans;

}

int main()

{

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

cin.tie(0);

cout.tie(0);

long long n;

cin>>n;

long long m,k;

cin>>m>>k;

vector<vector<long long>> A(n,vector<long long> (n,0));

for(long long i=0;i<m;i++)

{

long long u,v;

cin>>u>>v;

u--;

v--;

A[u][v]+=1;

}

A=p(A,k);

cout<<A[0][n-1];

return 0;

}

**// Rafy’s code**

\_\_int128 read(string ra)

{

\_\_int128 x=0;

for(auto i:ra)

{

x\*=10;

x+=i-'0';

}

return x;

}

**//totient**

int phi(int n)

{

int result = n;

for(int p = 2; p \* p <= n; ++p)

{

if (n % p == 0)

{

while (n % p == 0)

n /= p;

result -= result / p;

}

}

if (n > 1)

result -= result / n;

return result;

}

int phi(int n) {

int result = n;

for (int i = 2; i \* i <= n; i++) {

if (n % i == 0) {

while (n % i == 0)

n /= i;

result -= result / i;

}

}

if (n > 1)

result -= result / n;

return result;}

**//Bride with DSU**

**const int N=1e5+10;**

**int n;**

**set<pair<int,int>>akk;**

**void IS\_BRIDGE(int v,int to){**

**akk.insert({v,to});**

**}**

**vector<int> adj[N];**

**vector<int>par(N+1),sz(N+1);**

**vector<bool> visited(N,0);**

**vector<int> tin(N,-1), low(N,-1);**

**int timer;**

**void dfs(int v, int p) {**

**visited[v] = true;**

**tin[v] = low[v] = timer++;**

**bool parent\_skipped = false;**

**for (int to : adj[v]) {**

**if (to == p && !parent\_skipped) {**

**parent\_skipped = true;**

**continue;**

**}**

**if (visited[to]) {**

**low[v] = min(low[v], tin[to]);**

**} else {**

**dfs(to, v);**

**low[v] = min(low[v], low[to]);**

**if (low[to] > tin[v])**

**IS\_BRIDGE(v, to);**

**}**

**}**

**}**

**void find\_bridges() {**

**timer = 0;**

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

**if (!visited[i])**

**dfs(i,-1);**

**}**

**}**

**void make(ll n1)**

**{**

**par[n1]=n1;**

**sz[n1]=1;**

**}**

**int rep(int n1)**

**{**

**if(n1==par[n1])return n1;**

**return par[n1]=rep(par[n1]);**

**}**

**void add(int a,int b)**

**{**

**a=rep(a);**

**b=rep(b);**

**if(a!=b)**

**{**

**if(sz[a]<sz[b])**

**{**

**swap(a,b);**

**}**

**par[b]=a;**

**sz[a]+=sz[b];**

**}**

**}**

* **//Rakib Code**

**//Bit tricks and modular snippet**

**//store it in a variable (i start from 1)**

**#define GET\_BIT(n,i) ((n & (1LL << ((i)-1))) >> ((i)-1))**

**#define SET\_BIT(n,i) ((n) | (1LL << ((i)-1)))**

**#define CLR\_BIT(n,i) ((n) & ~(1LL << ((i)-1)))**

**#define TGL\_BIT(n,i) ((n) ^ (1LL << ((i)-1)))**

**#define CLR\_MSB(n,i) ((n) & (1LL << ((i)-1))-1)//upto i+1 pos**

**#define CLR\_LSB(n,i) ((n) & ~((1LL << (i))-1))//ipto i pos**

**int firstSetMSB(int n){**

**return 32-(\_\_builtin\_clz(n));//32 for int, 64 for ll and clzll**

**}**

**int firstSetLSB(int n){**

**return log2(n & -n)+1;**

**}**

**bool isPowerOfTwo(int n) {**

**return ((n & (n - 1)) == 0);**

**}**

**bool isPowerOf\_2k(int n,int k) {**

**return (isPowerOfTwo(n) && (n%(k-1)==1)); //returns true if n=power of 2^k**

**}**

**void printBinary(int n){**

**for(int i=31; i>=0; i--)**

**cout<<((n>>i) & 1);**

**cout<<endl;**

**}**

**const int MOD1 = 127657753, MOD2 = 987654319;**

**const int p1 = 137, p2 = 277;**

**ll binExp(ll a,ll b,ll mod){//O(logn)**

**//a%=mod**

**ll ans=1;**

**while(b>0){**

**if(b&1)**

**ans=(ans\*a)%mod;**

**a=(a\*a)%mod;**

**b=b>>1;**

**}**

**return ans;**

**}**

**ll modAdd(ll a, ll b, ll m){return ((a % m) + (b % m)) % m;}**

**ll modSub(ll a, ll b, ll m){return ((a % m) - (b % m) + m) % m;}**

**ll modMul(ll a, ll b, ll m){return ((a % m) \* (b % m)) % m;}**

**ll modDiv(ll a, ll b, ll m){return ((a % m) \* binExp(b,m-2,m)) % m;}**

**//2d dfs**

**void dfs(vector<vector<int>>& grid, int sr, int sc, int n, int m,vector<vector<int>>& vis){**

**if(sr>=n || sc>=m || sr<0 || sc<0)**

**return;**

**if(vis[sr][sc]==1 || grid[sr][sc]==0)**

**return;**

**vis[sr][sc]=1;**

**dfs(grid,sr-1,sc,n,m,vis);**

**dfs(grid,sr+1,sc,n,m,vis);**

**dfs(grid,sr,sc-1,n,m,vis);**

**dfs(grid,sr,sc+1,n,m,vis);**

**}**

**int matrix\_DFS(vector<vector<int>>& grid){**

**int n=grid.size();**

**int m=grid[0].size();**

**vector<vector<int>> vis(n,vector<int> (m,0));**

**int i=0,j=0;**

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

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

**if(grid[i][j]==1 && vis[i][j]==0){**

**dfs(grid,i,j,n,m,vis);**

**}**

**}**

**}**

**return ans;**

**}**

**//2D bfs**

**const int N=1e3+10;**

**int vis[N][N],level[N][N];**

**int n,m;**

**vector<pair<int,int>> moves={{0,1},{0,-1},{1,0},{-1,0}};**

**//corner moves: {1,1},{1,-1},{-1,1},{-1,-1}**

**bool valid(int x,int y){**

**return (x>=0 && x<n && y>=0 && y<m);**

**}**

**void reset(){**

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

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

**vis[i][j]=0;**

**level[i][j]=INF;**

**}**

**}**

**}**

**void bfs(int srcx,int srcy){**

**reset();**

**queue<pair<int,int>> q;**

**vis[srcx][srcy]=1;**

**level[srcx][srcy]=0;**

**q.push({srcx,srcy});**

**while(!q.empty()){**

**pair<int,int> u=q.front();**

**q.pop();**

**int ux=u.first,uy=u.second;**

**for(auto it:moves){**

**int vx=it.first+ux;**

**int vy=it.second+uy;**

**if(!valid(vx,vy))**

**continue;**

**if(vis[vx][vy])**

**continue;**

**q.push({vx,vy});**

**vis[vx][vy]=1;**

**level[vx][vy]=level[ux][uy]+1;**

**}**

**}**

**}**

**//Kruskal**

**//add dsu code then:**

**const int N=1e5+10;**

**void kruskal(){**

**int n,m;//node,edge**

**cin>>n>>m;**

**vector<pair<int,pair<int,int>>> edges;//w,s,d**

**for(int i=0;i<m;i++){**

**int s,d,w;**

**cin>>s>>d>>w;**

**edges.pb({w,{s,d}});**

**}**

**sort(all(edges));**

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

**make\_set(i);**

**int cost=0;**

**vector<pair<int,int>> mst;**

**for(auto it:edges){**

**int w=it.first;**

**int u=it.second.first;**

**int v=it.second.second;**

**if(find\_set(u)==find\_set(v))//rep in rajiuns snippet**

**continue;**

**union\_sets(u,v);**

**cost+=w;**

**mst.pb({u,v});**

**}**

**cout<<cost<<endl;**

**for(auto it:mst)**

**cout<<it.first<<" "<<it.second<<endl;**

**}**

**//hash segment**

**//--------------const----------**

**#define pll pair<ll,ll>**

**const int N = 1e6+10;**

**const ll MOD[2] = {998244353, 1000000007};**

**ll BASE[2] = {0, 0};**

**ll POW[2][N];**

**ll IPOW[2][N];**

**vector<pair<pll,pll>> tree(4\*N+1);**

**//------------------HASH----------**

**ll power(ll a, ll p, ll m) {**

**ll ans = 1;**

**a %= m;**

**while (p) {**

**if (p & 1ll)**

**ans = (ans\*a) % m;**

**p >>= 1ll;**

**a = (a\*a) % m;**

**}**

**return ans;**

**}**

**void hash\_pre() {**

**ll b1, b2, i, j, inv;**

**mt19937\_64 rnd(chrono::steady\_clock::now().time\_since\_epoch().count());**

**b1 = (500 + (rnd() % (MOD[0]-500\*2+1)));**

**b2 = 0;**

**do {**

**b2 = (500 + (rnd() % (MOD[1]-500\*2+1)));**

**} while (b1 == b2);**

**BASE[0] = b1;**

**BASE[1] = b2;**

**for (i = 0; i < 2; ++i) {**

**ll \*pw = POW[i], \*ipw = IPOW[i], x = BASE[i], m = MOD[i];**

**pw[0] = 1;**

**ipw[0] = 1;**

**inv = power(x, m-2, m);**

**for (j = 1; j < N; ++j) {**

**pw[j] = (pw[j-1] \* x) % m;**

**ipw[j] = (ipw[j-1] \* inv) % m;**

**}**

**}**

**}**

**template<typename T>**

**struct DoubleHash {**

**int n;**

**T s;**

**vector<pll> h;**

**DoubleHash() {}**

**DoubleHash(T s) : s(s) {**

**n = s.size();**

**h.resize(n+1);**

**ll \*pw0 = POW[0], m0 = MOD[0];**

**ll \*pw1 = POW[1], m1 = MOD[1];**

**int i = 0;**

**h[i] = {0, 0};**

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

**h[i] = {**

**(h[i-1].first + (s[i-1] \* pw0[i]) % m0) % m0,**

**(h[i-1].second + (s[i-1] \* pw1[i]) % m1) % m1,**

**};**

**}**

**}**

**pair<ll,ll> get\_hash(int l, int r) {**

**assert((0 <= l) && (l <= r) && (r < n));**

**++l; ++r;**

**ll \*ipw0 = IPOW[0], m0 = MOD[0];**

**ll \*ipw1 = IPOW[1], m1 = MOD[1];**

**return {**

**((h[r].first - h[l-1].first + m0) \* ipw0[l]) % m0,**

**((h[r].second - h[l-1].second + m1) \* ipw1[l]) % m1,**

**};**

**}**

**pll get\_hash() {**

**return get\_hash(0, n-1);**

**}**

**pll merge\_hash(int l1, int r1, int l2, int r2) {**

**assert((0 <= l1) && (l1 <= r1) && (r1 < l2) && (l2 <= r2) && (r2 < n));**

**pll p1 = get\_hash(l1, r1);**

**pll p2 = get\_hash(l2, r2);**

**ll \*pw0 = POW[0], m0 = MOD[0];**

**ll \*pw1 = POW[1], m1 = MOD[1];**

**int len = r1-l1+1;**

**return {**

**(p1.first + (p2.first \* pw0[len]) % m0) % m0,**

**(p1.second + (p2.second \* pw1[len]) % m1) % m1,**

**};**

**}**

**};**

**/\***

**hash\_pre(); use it to calculate powers**

**DoubleHash<string> hs(s); pre compute hash of full string**

**hs.get\_hash(0,i-1); get hash of a sub string**

**hs.merge\_hash(0,i-1,i,n-1); get & merge hash of teo sub string**

**\*/**

**//------------Segment tree---------**

**/\***

**a and ra are string and reverse of string**

**build(1,1,n,a,ra) at node 1 build the tree from 1 to n**

**query(1,1,n,x,y) at node 1 query in x to y range for 1 to n array**

**update(1,1,n,x,y,up,a,ra) at node 1 update x to y range for 1 to n array if up=true update r else ra**

**merge\_node(a,b) adds pair a and b by modding;**

**everything o(4\*logN)**

**\*/**

**pair<pll,pll> merge\_node(pair<pll,pll> a,pair<pll,pll> b){**

**ll \*pw0 = POW[0], \*ipw0 = IPOW[0], m0 = MOD[0];**

**ll \*pw1 = POW[1], \*ipw1 = IPOW[1], m1 = MOD[1];**

**return {**

**{(a.first.first+b.first.first)%m0,(a.first.second+b.first.second)%m1},**

**{(a.second.first+b.second.first)%m0,(a.second.second+b.second.second)%m1}**

**};**

**}**

**void build(int node,int b,int e,string &a,string &ra)//node,begin,end**

**{**

**ll \*pw0 = POW[0], \*ipw0 = IPOW[0], m0 = MOD[0];**

**ll \*pw1 = POW[1], \*ipw1 = IPOW[1], m1 = MOD[1];**

**if(b==e){**

**tree[node].first={1ll\*a[b]\*pw0[b] % m0,1ll\*a[b]\*pw1[b] % m1};**

**tree[node].second={1ll\*ra[b]\*pw0[b] % m0,1ll\*ra[b]\*pw1[b] % m1};**

**return;**

**}**

**int left=node\*2;**

**int right=node\*2+1;**

**int mid=(b+e)/2;**

**build(left,b,mid,a,ra);**

**build(right,mid+1,e,a,ra);**

**tree[node]=merge\_node(tree[left],tree[right]);//change accordingly**

**}**

**pair<pll,pll> query(int node,int b,int e,int i,int j)//node,begin,end,range of query(i,j)**

**{**

**if(i>e || j<b)**

**return {{0,0},{0,0}};//return something that doesnt effect the result**

**if(b>=i && e<=j)**

**return tree[node];**

**int left=node\*2;**

**int right=node\*2+1;**

**int mid=(b+e)/2;**

**pair<pll,pll> p1=query(left,b,mid,i,j);**

**pair<pll,pll> p2=query(right,mid+1,e,i,j);**

**return merge\_node(p1,p2);//change accordingly**

**}**

**void update(int node,int b,int e,int i,int j,bool up,string &a,string &ra)//node,begin,end,updating(position,value)**

**{**

**ll \*pw0 = POW[0], \*ipw0 = IPOW[0], m0 = MOD[0];**

**ll \*pw1 = POW[1], \*ipw1 = IPOW[1], m1 = MOD[1];**

**if(i>e || j<b)**

**return;**

**if(b==e){**

**if(up){**

**tree[node].first.first=(1ll\*a[b]\*pw0[b]) % m0;**

**tree[node].first.second=(1ll\*a[b]\*pw1[b]) % m1;**

**}else{**

**tree[node].second.first=(1ll\*ra[b]\*pw0[b]) % m0;**

**tree[node].second.second=(1ll\*ra[b]\*pw1[b]) % m1;**

**}**

**return;**

**}**

**int left=node\*2;**

**int right=node\*2+1;**

**int mid=(b+e)/2;**

**update(left,b,mid,i,j,up,a,ra);**

**update(right,mid+1,e,i,j,up,a,ra);**

**tree[node]=merge\_node(tree[left],tree[right]);//change accordingly**

**}**

**int main() {**

**hash\_pre();**

**cin>>s;**

**a=s;**

**reverse(all(s));**

**ra=s;**

**build(1,0,n-1,a,ra);**

**while(q--){**

**cin>>t;**

**if(t==1){**

**int pos;**

**char c;**

**cin>>pos>>c;**

**pos--;**

**a[pos]=c;**

**ra[n-1-pos]=c;**

**update(1,0,n-1,pos,pos,true,a,ra);**

**update(1,0,n-1,n-1-pos,n-1-pos,false,a,ra);**

**}else{**

**cin>>l>>r;**

**l--,r--;**

**pair<pll,pll> ans=query(1,0,n-1,l,r);**

**pair<pll,pll> rans=query(1,0,n-1,n-1-r,n-1-l);**

**ans.first={(1ll\*ans.first.first\*ipw0[l]) % m0,(1ll\*ans.first.second\*ipw1[l]) % m1};**

**rans.second={(1ll\*rans.second.first\*ipw0[n-1-r]) % m0,(1ll\*rans.second.second\*ipw1[n-1-r]) % m1};**

**if(ans.first==rans.second)**

**cout<<"Yes"<<endl;**

**else**

**cout<<"No"<<endl;**

**}**

**}**