**//Mushfiq Code**

**//NCR**

**const int N = 1e6 + 9,**

**mod = 1e9 + 7;**

**int f[N], inv[N], finv[N];**

**void prec() {**

**f[0] = 1;**

**for (int i = 1; i < N; i++) f[i] = 1 LL \* i \* f[i - 1] % mod;**

**inv[1] = 1;**

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

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

**inv[i] = (inv[i] + mod) % mod;**

**}**

**finv[0] = 1;**

**for (int i = 1; i < N; i++) finv[i] = 1 LL \* inv[i] \* finv[i - 1] % mod;**

**}**

**int ncr(int n, int r) {**

**if (n < r || n < 0 || r < 0) return 0;**

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

**15**

**}**

**void brute() {**

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

**C[i][0] = 1;**

**}**

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

**for (int j = 1; j <= i; j++) {**

**C[i][j] = (C[i - 1][j] + C[i - 1][j - 1]) % mod;**

**}**

**}**

**}**

**//Lazy**

**class SegmentTree {**

**public: int n;**

**vector < int > a,**

**lazy,**

**tree;**

**SegmentTree(vector < int > arr) {**

**a = arr;**

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

**lazy.assign(4 \* n, 0);**

**tree.assign(4 \* n, 0);**

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

**}**

**void update(int l, int r, int val) {**

**update(1, 0, n - 1, l, r, val);**

**}**

**int query(int l, int r) {**

**return query(1, 0, n - 1, l, r);**

**}**

**void print() {**

**for (auto it: tree) cout << it << endl;**

**}**

**private: void propagate(int node, int start, int end) {**

**if (start == end) tree[node] += lazy[node];**

**else {**

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

**/\*if you need sum use it else for other things remove (end-start+1) part\*/**

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

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

**}**

**lazy[node] = 0;**

**}**

**void build(int node, int start, int end) {**

**if (start == end) tree[node] = a[start];**

**else {**

**int mid = (start + end) / 2;**

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

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

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

**}**

**}**

**void update(int node, int start, int end, int l, int r, int val) {**

**propagate(node, start, end);**

**if (end < l or start > r) return;**

**if (start == end) tree[node] += val;**

**else if (l <= start and end <= r) {**

**lazy[node] += val;**

**propagate(node, start, end);**

**} else {**

**int mid = (start + end) / 2;**

**update(2 \* node, start, mid, l, r, val);**

**3**

**update(2 \* node + 1, mid + 1, end, l, r, val);**

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

**}**

**}**

**int query(int node, int start, int end, int l, int r) {**

**if (end < l or start > r) return 0;**

**propagate(node, start, end);**

**if (start == end) {**

**return tree[node];**

**} else if (l <= start and end <= r) return tree[node];**

**else {**

**int mid = (start + end) / 2;**

**int left = query(2 \* node, start, mid, l, r);**

**int right = query(2 \* node + 1, mid + 1, end, l, r);**

**return left + right;**

**}**

**}**

**};**

**//PBDS**

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

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

using namespace \_\_gnu\_pbds;

typedef tree<int, null\_type, less<int>, rb\_tree\_tag,

tree\_order\_statistics\_node\_update> ordered\_set;//sorted ascending

typedef tree<int, null\_type, greater<int>, rb\_tree\_tag,

tree\_order\_statistics\_node\_update> ordered\_rset;//sorted descending

typedef tree<int, null\_type, less\_equal<int>, rb\_tree\_tag,

tree\_order\_statistics\_node\_update> ordered\_multiset;//sorted ascending

typedef tree<int, null\_type, greater\_equal<int>, rb\_tree\_tag,

tree\_order\_statistics\_node\_update> ordered\_rmultiset;//sorted descending

/\*

\*\*\*Don’t use long long as defined when using it\*\*\*

1>declare ==> ordered\_set A

2>insert ==> A.insert(val);

3>finding kth element (0 based - index) ==> \*A.find\_by\_order(k);

4>finding the index in which val is located

if not found returns its relative position(position if inserted) ==>

A.order\_of\_key(val);

5>lower Bound ==> \*A.lower\_bound(val);

6>Upper Bound ==> \*A.upper\_bound(val);

7>Erase ==> A.erase(val or pointer);

For multiset use custom delete function!

\*/

void myErase(ordered\_set &t, int v){

int rank = t.order\_of\_key(v);

ordered\_set::iterator it = t.find\_by\_order(rank);4

if(\*it == v)t.erase(it);

}

**//Geometry**

const double PI=3.141592653589793;//acos(-1)

struct TwoDGeometry{

static double triangleArea(double base,double height){

return 0.5\*base\*height;

}

static double rectanglePerimeter(double length,double breadth){

return 2\*(length+breadth);

}

static double rectangleArea(double length,double breadth){

return length\*breadth;7

}

static double rectangleDiagonal(double length,double breadth){

return sqrt(length\*length+breadth\*breadth);

}

static double squarePerimeter(double side){

return 4\*side;

}

static double squareArea(double side){

return side\*side;

}

static double squareDiagonal(double side){

return sqrt(2)\*side;

}

static double parallelogramArea(double base,double height){

return base\*height;

}

static double rhombusArea(double diagonal1,double diagonal2){

return 0.5\*diagonal1\*diagonal2;

}

static double trapeziumArea(double height,double parallelSide1,double

parallelSide2){

return 0.5\*height\*(parallelSide1+parallelSide2);

}

static double circlePerimeter(double radius){

return 2\*PI\*radius;

}

static double circleArea(double radius){

return PI\*radius\*radius;

}

};

// 3D Shape Formulas

struct ThreeDGeometry{

static double cubeSurfaceArea(double side){

return 6\*side\*side;

}

static double cubeVolume(double side){

return pow(side,3);

}

static double cuboidSurfaceArea(double length,double breadth,double

height){

return 2\*(length\*breadth+breadth\*height+height\*length);

}

static double cuboidVolume(double length,double breadth,double height){

return length\*breadth\*height;

}

static double cylinderSurfaceArea(double radius,double height){

return 2\*PI\*radius\*(radius+height);

}

static double cylinderVolume(double radius,double height){

return PI\*radius\*radius\*height;

}

static double coneSurfaceArea(double radius,double height){

double slantHeight=sqrt(radius\*radius+height\*height);

return PI\*radius\*(radius+slantHeight);

}

static double coneVolume(double radius,double height){

return (1.0/3)\*PI\*radius\*radius\*height;

}

static double sphereSurfaceArea(double radius){

return 4\*PI\*radius\*radius;

}

static double sphereVolume(double radius){

return (4.0/3)\*PI\*pow(radius,3);

}

static double hemisphereSurfaceArea(double radius){

return 3\*PI\*radius\*radius;

}

static double hemisphereVolume(double radius){

return (2.0/3)\*PI\*pow(radius,3);

}

};

//ThreeDGeometry::sphereVolume(3)

**//2d prefix sum**

**void build() {**

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

**for (int j = 1; j <= m; j++)**

**pref[i][j] = pref[i - 1][j] + pref[i][j - 1] -**

**pref[i - 1][j - 1] + a[i][j];**

**}**

**int query(int x1, y1, x2, y2) {**

**return pref[x2][y2] - pref[x1 - 1][y2] -**

**pref[x2][y1 - 1] + pref[x1 - 1][y1 - 1];**

**}**

**void update(int a1, b1, a2, b2) {**

**pref[a1][b1]++;**

**pref[a2 + 1][b2 + 1]++;**

**pref[a1][b2 + 1]--;**

**pref[a2][b1 + 1]--;**

**}**

**//COD & SOD**

**int count\_of\_divisors(int n) {**

**int ans = 1;**

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

**if (n % i == 0) {**

**int e = 0;**

**while (n % i == 0) {**

**e++;**

**n /= i;**

**}**

**ans \*= (e + 1);**

**}**

**}**

**if (n > 1) ans \*= 2;**

**return ans;**

**}**

**int sum\_of\_divisor(int n) {**

**int ans = 1;**

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

**if (n % i == 0) {**

**int e = 0, gg = 1, sum = 1;**

**while (n % i == 0) {**

**n /= i, e++;**

**gg \*= i;**

**sum += gg;**

**}**

**ans \*= sum;**

**}**

**}**

**if (n > 1)**

**ans \*= (1 + n);**

**return ans;**

**}**

**// CSOD**

**int CSOD(int n) {**

**int i = 1;**

**int ans = 0;**

**while (i <= n) {**

**int q = n / i;**

**int j = (n / q) + 1;**

**int s = sum(i, j - 1); //i + (i+1) + ...(j-1)**

**ans += s \* q;**

**}**

**return ans;**

**}**

**// Mobius**

**mobb[1] = 1;**

**for (int i = 1; i < m; i++) {**

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

**if (j != i) mobb[j] -= mobb[i];**

**if (i > 1) divs[j].pb(i);**

**}**

**}**

**// Trie**

**struct TrieNode {**

**TrieNode \* childNode[26];**

**bool wordEnd;**

**TrieNode() {**

**wordEnd = 0;**

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

**childNode[i] = NULL;**

**}**

**}**

**};**

**void insert\_key(TrieNode \* root, string & key) {**

**TrieNode \* currNode = root;**

**for (auto c: key) {**

**if (currNode -> childNode[c - ’**

**a’] == NULL) {**

**TrieNode \* newNode = new TrieNode();**

**currNode -> childNode[c - ’**

**a’] = newNode;**

**currNode = currNode -> childNode[c - ’**

**a’];**

**}**

**currNode = currNode -> childNode[c - ’**

**a’];**

**}**

**currNode -> wordEnd = 1;**

**}**

**bool search\_key(TrieNode \* root, string & key) {**

**TrieNode \* currNode = root;**

**for (auto c: key) {**

**if (currNode -> childNode[c - ’**

**a’] == NULL) return 0;**

**currNode = currNode -> childNode[c - ’**

**a’];**

**}**

**return (currNode -> wordEnd);**

**}**

**//Shuvos Code**

**//Knapsack bigpower**

**const int mx = 1e6;**

**struct item {**

**int weight;**

**int value;**

**};**

**int Knapsack(int w, vector < item > it) {**

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

**vector < vector < int >> dp(n + 1, vector < int > (w + 1, 0));**

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

**for (int j = 1; j <= w; j++) {**

**if (it[i - 1].weight <= j) {**

**dp[i][j] = max(dp[i - 1][j], it[i - 1].value + dp[i - 1][j - it[i - 1].weight]);**

**} else {**

**dp[i][j] = dp[i - 1][j];**

**}**

**}**

**}**

**return dp[n][w];**

**}**

**//Big power**

**ll power(ll base, ll n, ll mod) {**

**ll result = 1;**

**while (n) {**

**if (n % 2 == 1) {**

**result = (result \* base) % mod**

**n--;**

**} else {**

**base = (base \* base) % mod**

**n /= 2;**

**}**

**}**

**return result;**

**}**

**//Inverse mod**

**ll power(ll base, ll p) {**

**ll res = 1;**

**while (p) {**

**if (p % 2 == 1) {**

**res = (res \* base) % mod;**

**p--;**

**} else {**

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

**p /= 2;**

**}**

**}**

**return res % mod;**

**}**

**//Articulation point**

**const int mx = 2e4 + 3;**

**ll low[mx], dic[mx], arti[mx], par[mx], tim;**

**bool vis[mx];**

**vector < ll > g[mx];**

**void init() {**

**tim = 0;**

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

**g[i].clear();**

**vis[i] = 0;**

**low[i] = 0;**

**dic[i] = 0;**

**arti[i] = -1;**

**}**

**}**

**void dfs(ll node, ll p) {**

**vis[node] = 1;**

**low[node] = dic[node] = ++tim;**

**ll child = 0;**

**for (auto x: g[node]) {**

**if (x == p) {**

**continue;**

**}**

**if (!vis[x]) {**

**child++;**

**dfs(x, node);**

**low[node] = min(low[x], low[node]);**

**if (dic[node] <= low[x] && p != -1) {**

**arti[node] = 1;**

**}**

**} else {**

**low[node] = min(dic[x], low[node]);**

**}**

**}**

**if (child > 1 && p == -1) {**

**arti[node] = 1;**

**}**

**}**

**Main()**

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

**if (!vis[i]) {**

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

**}**

**}**

**ll ans = 0;**

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

**if (arti[i] != -1) {**

**ans++;**

**}**

**}**

**//Finding bridge**

**bool vis[100];**

**vector < int > adj[100];**

**int timer;**

**int in [100];**

**int out[100];**

**void dfs(int node, int par) {**

**vis[node] = 1;**

**timer++;**

**in [node] = timer;**

**out[node] = timer;**

**for (int x: adj[node]) {**

**if (par == x) {**

**continue;**

**}**

**if (vis[x] == 1) {**

**out[node] = min(out[node], in [x]);**

**} else {**

**dfs(x, node);**

**if (in [node] > out[x]) {**

**cout << "There exit a bridege\n";**

**}**

**}**

**}**

**}**

**//Rakib Code**

**//stringstream example**

**string s;**

**getline(cin,s);**

**istringstream is(s);**

**int x,y;**

**string a,b;**

**is>>a>>x>>b>>y;**

**// subsetgen**

**vector<vector<int>> genSubsets(vector<int>& v){**

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

**int subcnt=(1<<n);**

**vector<vector<int>> all\_subset;**

**for(int mask=0; mask<subcnt; mask++){**

**vector<int> cur\_subset;**

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

**if((mask&(1<<bit))!=0){**

**cur\_subset.push\_back(v[bit]);**

**}**

**}**

**if(cur\_subset.size())**

**all\_subset.push\_back(cur\_subset);**

**}**

**return all\_subset;**

**}**

**//pbds**

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

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

**#include <ext/pb\_ds/detail/standard\_policies.hpp>**

**using namespace \_\_gnu\_pbds;**

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

**//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.**

**//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;}**

**//Number theory**

**map<long long,long long> freq;**

**void factorization\_of\_factorial(long long val){**

**int i=0;**

**ll ans=1;**

**while(prime[i]<=val){**

**ll x=0;**

**ll t=floor(log2(val)/log2(prime[i]));**

**for(ll j=1;j<=t;j++){**

**ll z=floor(val/pow(prime[i],j));**

**x+=z;**

**}**

**freq[prime[i]]=x;**

**i++;**

**}**

**}**

**//finding all divisors of all the numbers from 1 to 10^6**

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

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

**long long divSum[N];**

**void divisors(){//O(NlogN)**

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

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

**divs[j].pb(i);**

**divSum[j]+=i;**

**}**

**divSum[0]=0;**

**}**

**ll modexpo(ll x, ll n, ll m){**

**if(n == 0){**

**return 1;**

**}**

**ll y = x \* x % m; ///y = x^2**

**if(n % 2 == 0){**

**return modexpo(y, n/2, m);**

**}**

**else{**

**return (modexpo(x, n-1, m) \* x) % m;**

**}**

**}**

**int N=1e7+10;**

**vector<bool> isPrime(N,true);**

**vector<int> lp(N,0);//lowest prime of n**

**vector<int> hp(N,0);//highest prime of n**

**void sieveNorm(){//O(n\*log(log(N)))**

**isPrime[0]=isPrime[1]=false;**

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

**if(isPrime[i]==true){**

**lp[i]=hp[i]=i;**

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

**isPrime[j]=false;**

**hp[j]=i;**

**if(lp[j]==0)**

**lp[j]=i;**

**}**

**}**

**}**

**}**

**vector<int> p\_freq;**

**void p\_factorization(long long val){//O(logN)**

**while(val>1){**

**int pf=hp[val];**

**while(val%pf==0){**

**val/=pf;**

**p\_freq.push\_back(pf);**

**}**

**}**

**}**

**//bitwise sieve**

**long long N=1e7+10;**

**vector<int> mark(N/32+1);**

**vector<long long> prime;**

**void bit\_sieve(){//O(n\*log(log(N)))**

**// x=x|(1<<i) means setting 1 in ith bit of x**

**// x&(1<<i) means checking if ith bit of x is 1**

**for(long long i=4; i<N; i+=2){**

**long long index=i/32;**

**long long bitNo=i%32;**

**mark[index]=mark[index]|(1<<bitNo);**

**}**

**prime.push\_back(2);**

**for(long long i=3; i<N; i++){**

**long long index=i/32;**

**long long bitNo=i%32;**

**bool f=mark[index]&(1<<bitNo);**

**if(f==true)**

**continue;**

**prime.push\_back(i);**

**for(long long j=i\*i; j<N; j+=i+i){**

**long long index=j/32;**

**long long bitNo=j%32;**

**mark[index]=mark[index]|(1<<bitNo);**

**}**

**}**

**}**

**//segmented sieve**

**vector<long long> segmented\_sieve(long long L,long long R){// 1<=(L,R)<=1e12 and L-R<=1e6**

**//need to pre compute all prime till 1e6 or sqrt(R) using sieve**

**if(L==1)**

**L++;**

**vector<long long> seg\_prime;**

**vector<bool> seg\_mark(R-L+1,false);**

**for(auto pr : prime){**

**long long j=1LL\*pr\*pr;**

**if(j>R)**

**break;**

**if(j<L)**

**j=((L+pr-1)/pr)\*1LL\*pr;**

**while(j<=R){**

**seg\_mark[j-L]=true;**

**j+=pr;**

**}**

**}**

**for(long long i=L; i<=R; i++){**

**if(seg\_mark[i-L]==false)**

**seg\_prime.pb(i);**

**}**

**return seg\_prime;**

**}**

**//bfs & dfs**

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

**vector<int> g[N],vis(N);**

**void DFS(int source){**

**int u = source;**

**vis[u]=1;**

**for(auto v : g[u]){**

**if(vis[v]==0)**

**DFS(v);**

**}**

**}**

**void BFS(int source){**

**queue<int> q;**

**q.push(source);**

**vis[source]=1;**

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

**int u=q.front();**

**q.pop();**

**for(auto v: g[u]){**

**if(vis[v]==0){**

**q.push(v);**

**vis[v]=1;**

**}**

**}**

**}**

**}**

**//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;**

**}**

**}**

**}**

**//bipartite bfs**

**bool bipartiteBfs(int src, vector<int> adj[], int color[]) {**

**queue<int>q;**

**q.push(src);**

**color[src] = 1;**

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

**int node = q.front();**

**q.pop();**

**for(auto it : adj[node]) {**

**if(color[it] == -1) {**

**color[it] = 1 - color[node];**

**q.push(it);**

**} else if(color[it] == color[node]) {**

**return false;**

**}**

**}**

**}**

**return true;**

**}**

**bool checkBipartite(vector<int> adj[], int n) {**

**int color[n];**

**memset(color, -1, sizeof color);**

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

**if(color[i] == -1) {**

**if(!bipartiteBfs(i, adj, color)) {**

**return false;**

**}**

**}**

**}**

**return true;**

**}**

**//Cycle detect**

**bool detect(int src, vector<int> adj[], int vis[]) {**

**vis[src] = 1;**

**// store <source node, parent node>**

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

**q.push(make\_pair(src, -1));**

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

**int node = q.front().first;**

**int parent = q.front().second;**

**q.pop();**

**// go to all adjacent nodes**

**for(auto adjacentNode: adj[node]) {**

**// if adjacent node is unvisited**

**if(!vis[adjacentNode]) {**

**vis[adjacentNode] = 1;**

**q.push(make\_pair(adjacentNode, node));**

**}**

**// if adjacent node is visited and is not it's own parent node**

**else if(parent != adjacentNode) {**

**// yes it is a cycle**

**return true;**

**}**

**}**

**}**

**// there's no cycle**

**return false;**

**}**

**bool isCycle(int V, vector<int> adj[]) {**

**int vis[V];**

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

**vis[i]=0;**

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

**if(!vis[i]) {**

**if(detect(i, adj, vis)) return true;**

**}**

**}**

**return false;**

**}**

**//DSU**

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

**int parent[N],sizes[N];**

**void make\_set(int v) {**

**parent[v] = v;**

**sizes[v] = 1;**

**}**

**int find\_set(int v) {**

**if (v == parent[v])**

**return v;**

**return parent[v] = find\_set(parent[v]);//path compression**

**}**

**void union\_sets(int a, int b) {**

**a = find\_set(a);**

**b = find\_set(b);**

**if (a != b) {**

**if (sizes[a] < sizes[b])//union by size**

**swap(a, b);**

**parent[b] = a;**

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

**}**

**}**

**void dsu(int n,int q){**

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

**make\_set(i);**

**while(q--){**

**int a,b;**

**cin>>a>>b;**

**union\_sets(a,b);**

**}**

**int components=0;**

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

**if(find\_set(i)==i)**

**components++;**

**}**

**cout<<components<<endl;**

**}**

**//Kruskal**

**//add dsu code then:**

**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))**

**continue;**

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

**cost+=w;**

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

**}**

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

**for(auto it:mst)**

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

**}**

**//Floyd warshell**

**vector<vector<int>> pre(100, vector<int>(100, -1));**

**vector<vector<int>> dist(100, vector<int>(100, INF));**

**void go(int src,int dest){**

**if (src==dest)**

**cout<<src<<" ";**

**else if (pre[src][dest]==-1)**

**cout<<"There is no path from "<< src <<"->"<<dest<< endl;**

**else{**

**go(src, pre[src][dest]);**

**cout<<dest<<" ";**

**}**

**}**

**void floyd\_warshell(int n){**

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

**dist[i][i]=0;**

**}**

**for (int k = 1; k <= n; k++){**

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

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

**if(dist[i][k]==INF && dist[j][k]==INF) continue;**

**if (dist[i][j] > dist[i][k] + dist[k][j]){**

**dist[i][j] = dist[i][k] + dist[k][j];**

**pre[i][j] = pre[k][j];**

**}**

**}**

**}**

**}**

**}**

**//Natrix expo**

**const int nmax = 100+5; //max row or col size**

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

**/\***

**Create matrix: Matrix F0(2,1); F0.val[0][0] = 0, F0.val[1][0] = 1;**

**Print matrix: M.print();**

**multiplicatiopn: Fn = M \* F0;**

**exponentiation: M = Matexpo(M, n);**

**\*/**

**struct Matrix{**

**// after constructing val contains garbage**

**long long int val[nmax][nmax];**

**int row, col;**

**Matrix(int \_r, int \_c){**

**row = \_r;**

**col = \_c;**

**/\* memset \*/**

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

**for(int j = 0; j<col; j++)**

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

**}**

**Matrix operator\*(Matrix other){**

**Matrix result(row, other.col); // O(nmax\*nmax)**

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

**for(int j = 0; j<other.col; j++){**

**for(int k = 0; k<col; k++){**

**result.val[i][j] += val[i][k] \* other.val[k][j];**

**result.val[i][j] %= mod;**

**}**

**}**

**}**

**// O(row \* col \* other.col)**

**return result;**

**}**

**void print(){**

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

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

**cout<<val[i][j]<<" ";**

**}**

**cout<<endl;**

**}**

**}**

**};**

**// 1)if n=0 return In, 2)if n=even return (X^n/2)^2, 3)if n=odd return X\*(X^n/2)^2**

**Matrix Matexpo(Matrix X, long long int n){// X^n**

**Matrix Y(X.row, X.col);**

**if(n == 0){**

**for(int i = 0; i<X.row; i++)**

**Y.val[i][i] = 1;**

**return Y;**

**}**

**Y = Matexpo(X, n/2);**

**Y = Y \* Y;**

**if(n % 2 == 1){**

**Y = Y \* X;**

**}**

**return Y;**

**}**

**//double hash**

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

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

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

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

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

**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<pair<ll,ll>> 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,**

**};**

**}**

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

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

**}**

**pair<ll,ll> merge\_hash(int l1, int r1, int l2, int r2) {**

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

**pair<ll,ll> p1 = get\_hash(l1, r1);**

**pair<ll,ll> 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**

**\*/**

**//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)**

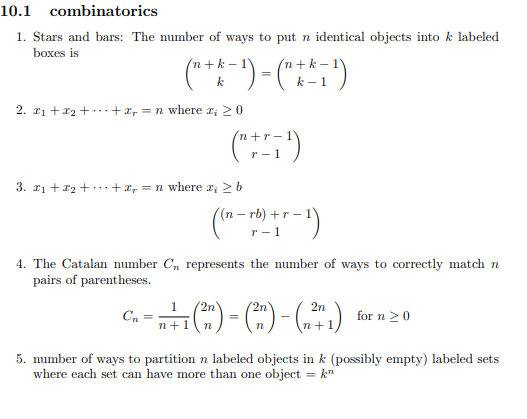
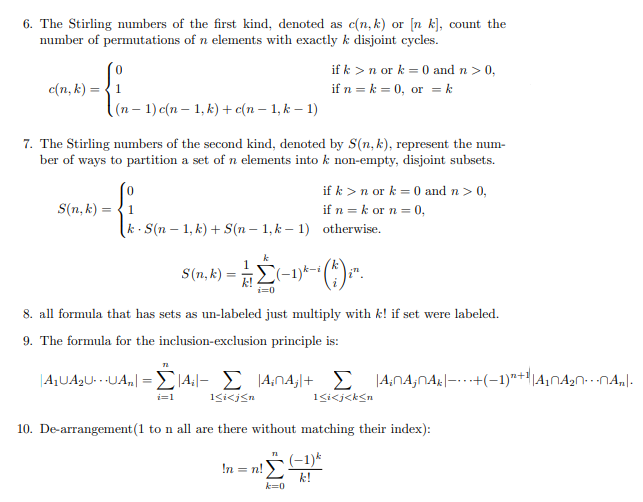
**cout<<"Yes"<<endl;**

**else**

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

**}**

**}**

****