#include <bits/stdc++.h>

using namespace std;

#define ll long long int

#define si(a) scanf("%d", &a)

#define sii(a, b) scanf("%d %d", &a, &b)

#define siii(a, b, c) scanf("%d %d %d", &a, &b, &c)

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

#define sll(a, b) scanf("%lld %lld", &a, &b)

#define slll(a, b, c) scanf("%lld %lld %lld", &a, &b, &c)

#define pf printf

#define dot(x) fixed<<setprecision(x)

#define max3(a, b, c) max(a, max(b, c))

#define min3(a, b, c) min(a, min(b, c))

#define vi vector<int>

#define vl vector<ll>

#define pb push\_back

#define PB pop\_back

#define mp make\_pair

#define vp vector<pair<ll, ll>>

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

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

#define FR(i, n) for (i = n; i > 0; i--)

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

#define eps 1e-9

#define mod2 1000007707

#define mod1 1000007909

#define P 150

#define file() \

freopen("input.txt", "r", stdin); \

freopen("output.txt", "w", stdout)

## 

## **String Algorithm:**

#### **// power up to n, Here m is the power value**

vl a;

void power(int n, int m)

{

int l = a.size();

if (l > n)

{

return;

}

if (l == 0)

{

a.pb(1);

l++;

}

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

{

a.pb((a[i - 1] \* P) % m);

}

}

#### **// Return a long long value which is the hash value of a string**

#### **// “s” is the string and m is the mod value**

ll hash\_value(string s, int m)

{

ll h = 0;

int l = s.size();

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

{

h = (h + (s[i] - 'a' + 1) \* a[i]) % m;

}

return h;

}

#### **// Return a vector<long long> which contain the hash value of prefix of a string**

vl prefix\_hash\_value(string s, int m)

{

vl h;

h.pb(0);

ll hv = 0;

int l = s.size();

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

{

hv = (hv + (s[i] - 'a' + 1) \* a[i]) % m;

h.pb(hv);

}

return h;

}

#### **// Return a vector<long> which contain the occurrences of string s in string t**

#### **// Here, m is the hash value of string s and hash is all prefix hash value of string t.**

vi occurrences(ll m, vl hash)

{

vi ocr;

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

{

ll x = (hash[i + n] - hash[i] + mod1) % mod1;

ll x1 = (m \* a[i]) % mod1;

if (x == x1)

{

ocr.pb(i);

}

}

return ocr;

}

#### **// Return the length of longest proper prefix**

ll prefix\_function(string s1, string s2)

{

int l = s1.size();

int m = 0, k = 0;

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

{

if (s1[k] == s2[i])

{

k++;

}

else

{

m = max(m, k);

k = 0;

if (s1[k] == s2[i])

{

k++;

}

}

}

m = max(m, k);

return m;

}

## 

## **LCS:**

#### **// lcs(0,0) will return the lcs string of string a,b;**

string a,b;

int n,m;

string ss[105][105]={};

int dp[105][105]={};

void lcs(int x, int y){

if(x==n || y==m){

dp[x][y]=0;

// ss[x][y]="";

return;

}

if(dp[x][y]!=-1){

return ;

}

if(a[x]==b[y]){

lcs(x+1,y+1);

if(ss[x+1][y+1].size()==0){

ss[x][y] = a[x];

}else{

ss[x][y] = a[x]+ss[x+1][y+1];

}

dp[x][y] = dp[x+1][y+1]+1;

return ;

}

lcs(x+1,y);

lcs(x,y+1);

if(dp[x+1][y]>=dp[x][y+1]){

ss[x][y] = ss[x+1][y];

dp[x][y] = dp[x+1][y];

}else{

ss[x][y] = ss[x][y+1];

dp[x][y] = dp[x][y+1];

}

return ;

}

#### **// lcs(0,0) will return the lcs of string a,b;**

string a,b;

int n,m;

int dp[1010][1010]={};

int lcs(int x, int y){

if(x==n || y==m){

return dp[x][y]=0;

}

if(dp[x][y]!=-1){

return dp[x][y];

}

if(a[x]==b[y]){

return dp[x][y] = lcs(x+1,y+1)+1;

}

return dp[x][y] = max(lcs(x+1,y),lcs(x,y+1));

}

## **Modular Arithmetic:**

#### **// Big Mod, Here n^p %m**

ll big\_mod(ll n, ll p, ll m){

if(p == 0)

return 1;

if(p%2==0){

ll a = big\_mod(n,p/2,m);

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

}

else{

ll a = big\_mod(n,p/2,m);

return ((n%m)\*(a%m)\*(a%m))%m;

}

}

#### **// Return sum of two big numbers as a string**

string sum(string a, string b){

if(a.size()<b.size()){

swap(a,b);

}

int x,y,z=0;

string s="";

reverse(all(a));

reverse(all(b));

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

if(j>=b.size()){

x = (a[i]-'0') + z;

s += ((x%10)+'0');

z = x/10;

}else{

x = (a[i]-'0') + (b[i]-'0')+z;

s += ((x%10)+'0');

z = x/10;

}

}

if(z<=0){

reverse(all(s));

return s;

}else{

s += (z+'0');

reverse(all(s));

return s;

}

}

#### **// Return product of two big numbers as a string**

string product(string s1, string s2){

if(s1.size()<s2.size()){

swap(s1,s2);

}

int l1 = s1.size();

int l2 = s2.size();

int x = 0;

string ans = "";

for(int i=l2-1;i>=0;i--){

string s="";

for(int k = 0;k<x;k++){

s+='0';

}

int a,b,c=0;

for(int j=l1-1;j>=0;j--){

a = ((s1[j]-'0') \* (s2[i]-'0')) + c;

s += ((a%10)+'0');

c = (a/10);

}

if(c>0){

s += (c+'0');

}

reverse(all(s));

ans = sum(s,ans);

x++;

}

return ans;

}

## **Base Conversion:**

#### **// Decimal to other base**

char a[] = {'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'};

string decimal\_to\_base(string s, int base){

int n=0;

string ans="";

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

n = (n\*10)+(s[i]-'0');

}

while(n){

ans+= a[n%base];

n/=base;

}

reverse(all(ans));

return ans;

}

#### **// Other base to Decimal**

int base\_to\_decimal(string s,int base){

int x = 0;

int ans = 0;

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

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

if(a[j]==s[i]){

ans+= (j\*pow(base,x));

}

}

x++;

}

return ans;

}

## **Sparse Table:**

#### **// logs(n) // build(a,n) // query(l,r)**

int st[100005][25],lg[100005]={0,0};

void logs(int n){

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

lg[i] = lg[i/2]+1;

}

}

void build(vi a, int n){

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

st[i][0] = a[i];

}

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

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

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

}

}

}

int query(int l,int r){

int x = lg[(r-l)+1];

return min(st[l][x],st[r-(1<<x)+1][x]);

}

## **DSU:**

int p[30] = {};

int sz[30] = {};

int findp(int n){

if(p[n]==n){

return n;

}

return p[n] = findp(p[n]);

}

void join(int u, int v){

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

swap(u,v);

}

p[u] = v;

sz[u] += sz[v];

}

void pre(){

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

p[i]=i;

sz[i]=1;

}

}

int main(){

//here x,y two connected node

int x,y;

int u = findp(x);

int v = findp(y);

if(u!=v){

join(u,v);

}

}

## **Segment Tree:**

vi a;

vi st;

void pre(){

for(int i=0;i<=4\*n;i++){

st.pb(0);

}

}

#### **// Build the segment tree … insert(1,1,array\_size);**

void insert(int n, int s, int e){

if(s==e){

st[n] = a[s-1];

return;

}

int l = 2\*n;

int r = 2\*n +1;

int m = s + (e-s)/2;

insert(l, s, m);

insert(r, m+1, e);

st[n] = min(st[l],st[r]);

}

#### **// query(1, 1, array\_size, left\_index, right\_index)**

ll query(int n, int s, int e, int i, int j){

if(j<s || i>e)

return 1000000000;

if(s>=i && e<=j){

return st[n];

}

int l = 2\*n;

int r = 2\*n +1;

int m = s + (e-s)/2;

int x = query(l, s, m,i,j);

int y = query(r, m+1, e,i,j);

return min(x,y);

}

#### **// index update(1,1,array\_size,index,value)**

void update(int n, int s, int e, int i, int v){

if(s==e){

st[n] = v;

return;

}

int l = 2\*n;

int r = 2\*n +1;

int m = s + (e-s)/2;

if(i<=m){

update(l, s, m, i, v);

}else{

update(r, m+1, e, i, v);

}

st[n] = (st[l]\*st[r]);

}

## **Lazy Segment Tree:**

vector<pair<ll, ll>> st;

void pre(){

for(i=0;i<=(4\*n)+1;i++){

st.pb(make\_pair(0,0));

}

}

ll query(int n, int s, int e, int i, int j,ll p=0){

if(j<s || i>e)

return 0;

if(s>=i && e<=j){

return st[n].first+p\*((e-s)+1);

}

int l = 2\*n;

int r = 2\*n +1;

int m = s + (e-s)/2;

ll x = query(l, s, m,i,j, p+st[n].second);

ll y = query(r, m+1, e,i,j,p+st[n].second);

return x+y;

}

void update(int n, int s, int e, int i, int j, ll v){

if(j<s || i>e)

return;

if(s>=i && e<=j){

st[n].first += ((e-s+1) \* v);

st[n].second += v;

return;

}

int l = 2\*n;

int r = 2\*n +1;

int m = s + (e-s)/2;

update(l, s, m,i,j,v);

update(r, m+1, e,i,j,v);

st[n].first = st[l].first + st[r].first + ((e-s+1) \* st[n].second);

}