Contenido

[Math 2](#_Toc493888436)

[Shanks' Algorithm(a^x = b(m)) 2](#_Toc493888437)

[Algoritmo Shanka-Tonelli 3](#_Toc493888438)

[Discrete\_logarithm 3](#_Toc493888439)

[Discrete roots 3](#_Toc493888440)

[Gauss 4](#_Toc493888441)

[Big Number Class 5](#_Toc493888442)

[Extended GCD 7](#_Toc493888443)

[Inverso Modular mcd(n,m) 7](#_Toc493888444)

[Linear Congruences 7](#_Toc493888445)

[Matrix Exponeciation 7](#_Toc493888446)

[Modular Multiplication 8](#_Toc493888447)

[Notacion Infija-Postfija 8](#_Toc493888448)

[PHI(N) 9](#_Toc493888449)

[Phi Sieve 9](#_Toc493888450)

[Pholard-Rho 9](#_Toc493888451)

[Sum\_of\_divs\_Nfact 11](#_Toc493888452)

[FFT Complex 12](#_Toc493888453)

[Teorema del Resto Chino 13](#_Toc493888454)

[Ternary Search 13](#_Toc493888455)

[Geometry 13](#_Toc493888456)

[Area de Union de rectangulos 16](#_Toc493888457)

[Closest pair points 17](#_Toc493888458)

[Convex Hull 18](#_Toc493888459)

[Perimeter of the union of rectangles 18](#_Toc493888460)

[Pick Theorem 19](#_Toc493888461)

[Par de puntos más lejanos (Manhatan) 20](#_Toc493888462)

[Mayor ctdad de ptos dento de un circulo de radio R 20](#_Toc493888463)

[Diameter of a planar point set 21](#_Toc493888464)

[String 22](#_Toc493888465)

[AHO\_CORASICK 22](#_Toc493888466)

[2D PATTERN MATCHING 23](#_Toc493888467)

[HASHING 24](#_Toc493888468)

[Knuth Morris Pratt 25](#_Toc493888469)

[Longest Square (TANDEMS) 25](#_Toc493888470)

[Lyndon Descomposition 26](#_Toc493888471)

[MANACHER 26](#_Toc493888472)

[MAXIMUN-MINIMUN-SUFFIX 26](#_Toc493888473)

[Period 27](#_Toc493888474)

[SMALLEST CYCLIC SHIFT 27](#_Toc493888475)

[SUFFIX ARRAY (N log^2 N) 27](#_Toc493888476)

[SUFFIX ARRAY (N log N) 28](#_Toc493888477)

[Z and Prefix function 29](#_Toc493888478)

[Palindromic Tree 30](#_Toc493888479)

[Grafos 31](#_Toc493888480)

[Componentes Biconexas 31](#_Toc493888481)

[Bridges and Articulation Points 32](#_Toc493888482)

[Centroid descomposition 33](#_Toc493888483)

[Centroid Descomposition (centroid tree) 34](#_Toc493888484)

[Dinic 36](#_Toc493888485)

[Edmonds 37](#_Toc493888486)

[Edmond-Karp 39](#_Toc493888487)

[Hungarian 40](#_Toc493888488)

[Khun 41](#_Toc493888489)

[Max Flow Min Cost 42](#_Toc493888490)

[Stable marriage 43](#_Toc493888491)

[Stoer Wagner 43](#_Toc493888492)

[Tarjan SCC 44](#_Toc493888493)

[Minimun Vertex Cover Bipartite 45](#_Toc493888494)

[Bellman Ford 46](#_Toc493888495)

[DSU on tree 46](#_Toc493888496)

[Camino Euleriano 47](#_Toc493888497)

[Steiner Tree 48](#_Toc493888498)

[Tarjan (Vertex) BCC 48](#_Toc493888499)

[Data Structure 48](#_Toc493888500)

[AVL 48](#_Toc493888501)

[Heavy-Light Descomposition 50](#_Toc493888502)

[KD-Tree 54](#_Toc493888503)

[SQRT Descomposition 55](#_Toc493888504)

[Order Statistic Set 56](#_Toc493888505)

[Persistent Segment Tree 56](#_Toc493888506)

[RMQ 57](#_Toc493888507)

[Splay Tree 58](#_Toc493888508)

[Treap 60](#_Toc493888509)

[Treap Persistente 62](#_Toc493888510)

[Dynamic Programming 64](#_Toc493888511)

[Convex Hull Trick 64](#_Toc493888512)

[Divide and Conquer 64](#_Toc493888513)

[From Divide and Conquer to Knuth 65](#_Toc493888514)

[Knuth Optimization 66](#_Toc493888515)

[SQRT 67](#_Toc493888516)

[Mo’ s Algorithm 67](#_Toc493888517)

[Mo’s Algorithm on tree 67](#_Toc493888518)

[SQRT Descomposition 69](#_Toc493888519)

# Math

### Shanks' Algorithm(a^x = b(m))

O(sqrt(m)), return x such that a^x = b mod m

int solve ( int a, int b, int m )

{

int n = (int)sqrt( m + .0 )+1, an = 1;

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

an = (an \* a)%m;

map<int, int>vals;

for ( int i = 1, cur = an; i <= n; ++ i )

{

if ( ! vals. count ( cur ) )

vals [ cur ] = i ;

cur = (cur \* an)%m;

}

for ( int i = 0, cur = b; i <= n; ++ i )

{

if ( vals. count ( cur ) )

{

int ans = vals [ cur ] \* n - i ;

if ( ans < m )return ans;

}

cur = (cur \* a)%m;

}

return -1;

}

### Algoritmo Shanka-Tonelli

//devuelve x (mod p) tal que x^2 = a (mod p) //using: powmod()

long long solve\_quadratic( long long a, int p )

{

if( a == 0 ) return 0;

if( p == 2 ) return a;

if( powMod(a,(p-1)/2, p) != 1 ) return -1;

int phi = p-1, n = 0, k = 0, q = 0;

while( phi%2 == 0 ) phi/=2, n ++;

k = phi;

for( int j = 2; j < p; j ++ )

if( powMod( j, (p-1)/2, p ) == p-1 )

{

q = j;

break;

}

long long t = powMod( a, (k+1)/2, p );

long long r = powMod( a, k, p );

while( r != 1 )

{

int i = 0, v = 1;

while( powMod( r, v, p ) != 1 ) v \*= 2, i ++;

long long e = powMod( 2, n-i-1, p );

long long u = powMod( q, k\*e, p );

t = (t\*u)%p;

r = (r\*u\*u)%p;

}

return t;

}

### Discrete\_logarithm

ll dlog(ll a, ll b, ll M)

{

map<ll, ll> \_hash;

ll n = euler\_phi(M), k = sqrt(n);

for(ll i = 0, t = 1; i < k; ++i)

{

\_hash[t] = i;

t = mul(t, a, M);

}

ll c = pow(a, n - k, M);

for(ll i = 0; i \* k < n; i++)

{

if(\_hash.find(b) != \_hash.end())

return i \* k + \_hash[b];

b = mul(b, c, M);

}

return -1;

}

### Discrete roots

/\*

Solve x^k=a (mod n)

\*/

vector<ll> discrete\_root(ll k, ll a, ll n)

{

if (a == 0)

return {0};

ll g = primitive\_root(n);

ll sq = (ll) sqrt(n + .0) + 1;

vector<pair<ll, ll>> dec(sq);

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

dec[i - 1] = {pow(g, ll(i \* sq \* 1ll \* k % (n - 1)), n), i};

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

ll any\_ans = -1;

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

{

ll my = ll(pow(g, ll(i \* 1ll \* k % (n - 1)), n) \* 1ll \* a % n);

auto it = lower\_bound(dec.begin(), dec.end(), make\_pair(my, 0ll));

if (it != dec.end() && it->first == my)

{

any\_ans = it->second \* sq - i;

break;

}

}

if (any\_ans == -1)

return {};

ll delta = (n - 1) / \_\_gcd(k, n - 1);

vector<ll> ans;

for (ll cur = any\_ans % delta; cur < n - 1; cur += delta)

ans.push\_back(pow(g, cur, n));

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

return ans;

}

### Gauss

// Gauss-Jordan elimination with full pivoting.

// Uses:

// (1) solving systems of linear equations (AX=B)

// (2) inverting matrices (AX=I)

// (3) computing determinants of square matrices

// Running time: O(n^3)

// INPUT: a[][] = an nxn matrix

// b[][] = an nxm matrix

// OUTPUT: X = an nxm matrix (stored in b[][])

// A^{-1} = an nxn matrix (stored in a[][])

// returns determinant of a[][]

const double EPS = 1e-10;

typedef vector<int> VI;

typedef double T;

typedef vector<T> VT;

typedef vector<VT> VVT;

int GaussJordan(vector<vector<T> > &a, vector<vector<T> > &b) {

const int n = a.size();

const int m = b[0].size();

vector<int> irow(n), icol(n), ipiv(n);

int det = 1;

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

int pj = -1, pk = -1;

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

if (!ipiv[j])

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

if (!ipiv[k])

if (pj == -1 || fabs(a[j][k]) > fabs(a[pj][pk])) {

pj = j;

pk = k;

}

if (fabs(a[pj][pk]) < EPS) {

cerr << "Matrix is singular." << endl;

exit(0);

}

ipiv[pk]++;

swap(a[pj], a[pk]);

swap(b[pj], b[pk]);

if (pj != pk)

det \*= -1;

irow[i] = pj;

icol[i] = pk;

int c = 1.0 / a[pk][pk];

det \*= a[pk][pk];

a[pk][pk] = 1.0;

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

a[pk][p] \*= c;

for (int p = 0; p < m; p++)

b[pk][p] \*= c;

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

if (p != pk) {

c = a[p][pk];

a[p][pk] = 0;

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

a[p][q] -= a[pk][q] \* c;

for (int q = 0; q < m; q++)

b[p][q] -= b[pk][q] \* c;

}

}

for (int p = n - 1; p >= 0; p--)

if (irow[p] != icol[p]) {

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

swap(a[k][irow[p]], a[k][icol[p]]);

}

return det;

}

### Big Number Class

#define iszero(t) (t.len==1&&t.s[0]==0)

#define setlen(l,t) t.len=l;while(t.len>1&&t.s[t.len-1]==0) t.len--

const int maxlen=100;

struct bigint

{

int len,s[maxlen];

bigint()

{

\*this=0;

}

bigint(int a)

{

\*this=a;

}

bigint(const char \*a)

{

\*this=a;

}

bigint operator=(int);

bigint operator=(const char\*);

bigint operator=(const bigint&); //Optional

friend ostream& operator<<(ostream&,const bigint&);

bigint operator+(const bigint&);

bigint operator-(const bigint&);

bigint operator\*(const bigint&);

bigint operator/(const bigint&); //Require - cmp

bigint operator%(const bigint&); //Require - cmp

static int cmp(const bigint&,const bigint&);

static bigint sqrt(const bigint&); //Require - \* cmp

};

bigint bigint::operator=(int a)

{

len=0;

do

{

s[len++]=a%10;

a/=10;

}

while(a>0);

return \*this;

}

bigint bigint::operator=(const char \*a)

{

len=strlen(a);

for(int i=0; i<len; i++) s[i]=a[len-i-1]-'0';

return \*this;

}

bigint bigint::operator=(const bigint&a)

{

len=a.len;

memcpy(s,a.s,sizeof(\*s)\*len);

return \*this;

}

ostream& operator<<(ostream&os,const bigint&a)

{

for(int i=a.len-1; i>=0; i--) os<<a.s[i];

return os;

}

bigint bigint::operator+(const bigint&a)

{

bigint b;

b.s[b.len=max(len,a.len)]=0;

for(int i=0; i<b.len; i++) b.s[i]=(i<len?s[i]:0)+(i<a.len?a.s[i]:0);

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

if(b.s[i]>=10)

{

b.s[i]-=10;

b.s[i+1]++;

}

if(b.s[b.len]) b.len++;

return b;

}

//Make sure \*this>=a

bigint bigint::operator-(const bigint&a)

{

bigint b;

for(int i=0; i<len; i++) b.s[i]=s[i]-(i<a.len?a.s[i]:0);

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

if(b.s[i]<0)

{

b.s[i]+=10;

b.s[i+1]--;

}

setlen(len,b);

return b;

}

bigint bigint::operator\*(const bigint&a)

{

bigint b;

memset(b.s,0,sizeof(\*s)\*(len+a.len+1));

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

for(int j=0; j<a.len; j++) b.s[i+j]+=s[i]\*a.s[j];

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

{

b.s[i+1]+=b.s[i]/10;

b.s[i]%=10;

}

setlen(len+a.len+1,b);

return b;

}

bigint bigint::operator/(const bigint&a)

{

bigint b,c;

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

{

if(!iszero(b))

{

for(int j=b.len; j>0; j--) b.s[j]=b.s[j-1];

b.len++;

}

b.s[0]=s[i];

c.s[i]=0;

while(cmp(b,a)>=0)

{

b=b-a;

c.s[i]++;

}

}

setlen(len,c);

return c;

}

bigint bigint::operator%(const bigint&a)

{

bigint b;

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

{

if(!iszero(b))

{

for(int j=b.len; j>0; j--) b.s[j]=b.s[j-1];

b.len++;

}

b.s[0]=s[i];

while(cmp(b,a)>=0) b=b-a;

}

return b;

}

int bigint::cmp(const bigint&a,const bigint&b)

{

if(a.len<b.len) return -1;

else if(a.len>b.len) return 1;

for(int i=a.len-1; i>=0; i--)

if(a.s[i]!=b.s[i]) return a.s[i]-b.s[i];

return 0;

}

bigint bigint::sqrt(const bigint&a)

{

int n=(a.len-1)/2,p;

bigint b,d;

b.len=n+1;

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

{

if(!iszero(d))

{

for(int j=d.len+1; j>1; j--) d.s[j]=d.s[j-2];

d.s[0]=a.s[i\*2];

d.s[1]=a.s[i\*2+1];

d.len+=2;

}

else d=a.s[i\*2]+(i\*2+1<a.len?a.s[i\*2+1]\*10:0);

bigint c;

c.s[1]=0;

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

{

c.s[j]+=b.s[i+j]<<1;

if(c.s[j]>=10)

{

c.s[j+1]=1;

c.s[j]-=10;

}

else c.s[j+1]=0;

}

c.len=n-i+1+c.s[n-i+1];

for(p=1;; p++)

{

c.s[0]=p;

if(cmp(d,c\*p)<0) break;

}

b.s[i]=c.s[0]=p-1;

d=d-c\*(p-1);

}

return b;

}

### Extended GCD

devuelve x,y tal que ax+by = gcd(a,b)

int64 extended\_euclid( int64 a, int64 b, int64& x, int64& y )

{

int64 g = a;

x = 1;

y = 0;

if ( b != 0 )

{

g = extended\_euclid( b, a % b, y, x );

y -= ( a / b ) \* x;

}

return g;

}

### Inverso Modular mcd(n,m)

using: Extended GCD

int inverso\_mod(int n,int m)

{

int s, t, d;

d = extended\_euclid( n, m, s, t );

return ((s % m)+m)% m;

}

### Linear Congruences

/\*

Solve x=ai(mod mi), for any i and j, (mi,mj)|ai-aj

Return (x0,M) M=[m1..mn]. All solutions are x=x0+t\*M

Note: be carful with the overflow in the multiplication

Tested: LIGHTOJ 1319

\*/

pair<ll, ll> linear\_congruences(const vector<ll> &a, const vector<ll> &m)

{

int n = a.size();

ll u = a[0], v = m[0], p, q;

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

{

ll r = gcd(v, m[i], p, q);

ll t = v;

if ((a[i] - u) % r)

return {-1, 0}; // no solution

v = v / r \* m[i];

u = ((a[i] - u) / r \* p \* t + u) % v;

}

if (u < 0)

u += v;

return {u, v};

}

### Matrix Exponeciation

//Matrix Exponentiation O( n^3\*log(n) )

typedef vector <int> vect;

typedef vector < vect > matrix;

matrix identity (int n) {

matrix A(n, vect(n));

for (int i = 0; i <n; i++) A[i][i] = 1;

return A;

}

matrix mul(const matrix &A, const matrix &B) {

matrix C(A.size(), vect(B[0].size()));

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

for (int j = 0; j < C[i].size(); j++)

for (int k = 0; k < A[i].size(); k++)

C[i][j] += A[i][k] \* B[k][j];

return C;

}

matrix pow(const matrix &A, int e) {

return ( e == 0 ) ? identity(A.size()) :

( e % 2 == 0 ) ? pow(mul(A, A), e/2) : mul(A, pow(A, e-1));

}

### Modular Multiplication

// Modular Multiplication of big numbers

inline ll mulmod(ll a, ll b, ll m) {

ll x = 0, y = a % m;

while (b > 0) {

if (b % 2 == 1) x = (x + y) % m;

y = (y \* 2) % m;

b /= 2;

}

return x;

}

### Notacion Infija-Postfija

char E[1000], I[1000], x;

stack<char>S;

stack<double>P;

int v, V[30];

int val( char c )

{

if( c == '+' || c == '-' )

return 1;

if( c == '\*' || c == '/' )

return 2;

if( c == '^' )

return 3;

return 4;

}

int main()

{

scanf("%s\n",E);

int le = strlen(E);

E[le] = ')';

while( scanf("%c=%d\n",&x,&v) != EOF )

V[x-'a'] = v;

int pos = 0;

S.push('(');

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

{

if( E[i] >= 'a' && E[i] <= 'z' )

I[pos++] = E[i];

else if( E[i] == '(' )

S.push(E[i]);

else if( E[i] == ')' )

{

while( S.top() != '(' )

{

I[pos++] = S.top();

S.pop();

}

S.pop();

}

else

{

while( !S.empty() && val(S.top()) >= val(E[i]) && val(S.top()) <= 3 )

{

I[pos++] = S.top();

S.pop();

}

S.push(E[i]);

}

}

printf("%s",I);

double a,b,c;

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

{

if( I[i] >= 'a' && I[i] <= 'z' )

P.push(V[I[i]-'a']);

else

{

b = P.top();

P.pop();

a = P.top();

P.pop();

if( I[i] == '+' )

c = a+b;

if( I[i] == '-' )

c = a-b;

if( I[i] == '\*' )

c = a\*b;

if( I[i] == '/' )

c = a/b;

if( I[i] == '^' )

c = pow(a,b);

P.push(c);

}

}

printf("\n%.2lf",P.top());

return 0;

}

### PHI(N)

// computes the number of coprimes of p^k, being p prime

//int phi(int p, int k) { return pow(p, k) - pow(p, k-1); } // phi(p^k)

int phi(int p, int pk) { return pk - (pk/p); } // phi(p^k), where pk=p^k

// computes the number of coprimes of n

// phi(n) = (p\_1 - 1) \* p\_1 ^ (k\_1 - 1) \* (p\_2 - 1) \* p\_2 ^ (k\_2-1)

int phi(int n){

int coprimes = (n != 1); // phi(1) = 0

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

if (n%i == 0){

int pk = 1;

while (n%i == 0)

n /= i, pk \*= i;

coprimes \*= phi(i, pk);

}

if (n > 1) coprimes \*= phi(n, n); // n is prime

return coprimes;

}

### Phi Sieve

#define MAXN 100000

int phi[MAXN + 1],factor[MAXN + 1];

void phi\_sieve(){

memset(factor,-1,sizeof factor);

phi[1] = 1;

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

if(factor[i] == -1){

phi[i] = i - 1;

if(i <= MAXN / i)

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

factor[j] = i;

}else{

int p = factor[i];

int aux = i / p;

if(aux % p == 0) phi[i] = phi[aux] \* p;

else phi[i] = (p - 1) \* phi[aux];

}

}

}

### Pholard-Rho

typedef unsigned long long ull;

typedef long long ll;

//return a \* b % m

ull mul\_mod(ull a, ull b, ull m)

{

ull res = 0, tmp = a % m;

while (b)

{

if (b & 1)

{

res = res + tmp;

res = (res >= m ? res - m : res);

}

b >>= 1;

tmp <<= 1;

tmp = (tmp >= m ? tmp - m : tmp);

}

return res;

}

//return a ^ b % m

ll exp\_mod(ll a, ll b, ll m)

{

ll res = 1 % m, tmp = a % m;

while (b)

{

if (b & 1)

{

//res = (res \* t) % m;

res = mul\_mod(res, tmp, m);

}

//t = t \* t % m;

tmp = mul\_mod(tmp, tmp, m);

b >>= 1;

}

return res;

}

bool Miller\_Rabin(ll n)

{

int a[5] = { 2, 3, 7, 61, 24251 };

if (n == 2) return true;

if (n == 1 || (n & 1) == 0) return false;

ll b = n - 1;

for (int i = 0; i < 5; i++)

{

if (a[i] >= n) break;

while ((b & 1) == 0) b >>= 1;

ll t = exp\_mod(a[i], b, n);

while (b != n - 1 && t != 1 && t != n - 1)

{

t = mul\_mod(t, t, n);

b <<= 1;

}

if (t == n - 1 || (b & 1)) continue;

else return false;

}

return true;

}

ll factor[100];

ll nfactor, minfactor;

ll gcd(ll a, ll b)

{

return b ? gcd(b, a%b) : a;

}

void Factor(ll n);

void pollard\_rho(ll n)

{

if (n <= 1) return;

if (Miller\_Rabin(n))

{

factor[nfactor++] = n;

if (n < minfactor) minfactor = n;

return;

}

ll x = 2 % n, y = x, k = 2, i = 1;

ll d = 1;

while (true)

{

i++;

x = (mul\_mod(x, x, n) + 1) % n;

d = gcd((y - x + n) % n, n);

if (d > 1 && d < n)

{

pollard\_rho(d);

pollard\_rho(n / d);

return;

}

if (y == x)

{

Factor(n);

return;

}

if (i == k)

{

y = x;

k <<= 1;

}

}

}

void Factor(ll n)

{

ll d = 2;

while (n % d != 0 && d \* d <= n) d++;

pollard\_rho(d);

pollard\_rho(n / d);

}

// return true if n is prime

// nfactor is the size

bool factorize(ll n)

{

nfactor = 0;

minfactor = (1LL << 63);

pollard\_rho(n);

if (nfactor == 1 && factor[0] == n)

{

return true;

}

sort(factor, factor + nfactor);

return false;

}

int main()

{

ll n;

while (cin >> n && n != -1)

{

factorize(n);

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

{

cout << " ";

cout << factor[i] << endl;

}

cout << endl;

}

}

### Sum\_of\_divs\_Nfact

const int MAXN = 5e4 + 1;

const LL MOD = 1e9 + 7;

bool comp[MAXN];

vector<int> primes;

LL fact[MAXN];

void Criba(){

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

comp[i] = true;

for (int i = 3 ; i \* i < MAXN ; i += 2)

if (!comp[i])

for (int j = i \* i ; j < MAXN ; j += 2 \* i)

comp[j] = true;

primes.push\_back(2);

for (int i = 3 ; i < MAXN ; i += 2)

if (!comp[i])

primes.push\_back(i);

primes.push\_back(MAXN);//centinela

}

LL E(int N, int p){

LL r = 0;

while (N / p > 0){

r += N / p;

N /= p;

}

return r;

}

LL ModExp(int b, int e){

if (e == 0)

return 1LL;

if (e & 1)

return ModExp(b, e - 1) \* b % MOD;

LL r = ModExp(b, e >> 1);

return r \* r % MOD;

}

LL InvMod(int p){

LL r = ModExp(p, MOD - 2);

return r;

}

int main(){

Criba();

fact[0] = 1;

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

fact[i] = (fact[i - 1] \* i) % MOD;

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

cin.tie(0);

int N;

cin >> N;

LL sol = 1;

for (int i = 0 ; primes[i] <= N ; i++){

LL e = E(N, primes[i]);

LL f = ModExp(primes[i], e + 1) - 1;

sol \*= f;

sol %= MOD;

sol \*= InvMod(primes[i] - 1);

sol %= MOD;

}

sol -= fact[N];

sol += MOD;

sol %= MOD;

cout << sol << '\n';

return 0;

}

### FFT Complex

const int MN = 1e5+2;

typedef long long ll;

typedef complex<double> base;

double PI = acos(-1);

void fft(vector<base> &a, bool invert)

{

int n = (int) a.size();

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

{

ll bit = n >> 1;

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

j += bit;

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

}

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

{

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

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

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

{

base w(1);

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

{

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

a[i + j] = u + v;

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

w \*= wlen;

}

}

}

if(invert)

for(int i = 0; i < n; ++i) a[i] /= n;

}

void multiply(const vector<ll> &a, const vector<ll> &b, vector<ll> &res)

{

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

size\_t n = 1;

while(n < max(a.size(), b.size())) n <<= 1;

n <<= 1;

fa.resize(n), fb.resize(n);

fft(fa, false), fft(fb, false);

for(size\_t i = 0; i < n; ++i) fa[i] \*= fb[i];

fft(fa, true);

res.resize(n);

for(size\_t i = 0; i < n; ++i) res[i] = ll(fa[i].real() + 0.5);

}

ll mod(ll k)

{

if (k < 0) return -k;

return k;

}

int main()

{

cin.tie(0);

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

int N,M;

cin >> N >> M;

vector<ll> a(2\*MN,0), b(2\*MN,0), c;

ll tmp;

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

{

cin >> tmp;

a[tmp+MN]++;

}

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

{

cin >> tmp;

b[MN-(tmp+1)]++;

}

multiply(a,b,c);

ll sol = 0;

ll NN = c.size();

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

if (c[i])

sol += c[i]\*mod(2\*MN-1-i);

cout << sol << "\n";

return 0;

}

### Teorema del Resto Chino

using:Inverso Modular

int resto\_chino (int x[], int m[], int k)

{

int i, tmp, MOD = 1, RES = 0;

for (i=0; i <k ; i++) MOD \*= m[i];

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

{

tmp = MOD/m[i];

tmp \*= inverso\_mod(tmp, m[i]);

RES += (tmp\*x[i]) % MOD;

}

return RES % MOD;

}

### Ternary Search

double TS()

{

double left = minimo, right = maximo;

while(right-left > EPS)

{

double lt = (2.\*left + right) / 3;

double rt = (left + 2.\*right) / 3;

if(f(lt) > f(rt))//minimo (< si maximo)

left = lt;

else

right = rt;

}

return (left+right)/2;

}

# Geometry

const double EPS = 1e-8;

const double oo = 1e12;

const double PI = 3.141592653589793;

#define X real()

#define Y imag()

typedef complex<double> P;

typedef vector<P> Pol;

struct circle

{

P p;

double r;

circle() {}

circle(P x,double rr)

{

p=x, r = rr;

}

};

struct L: public vector <P> //Linea

{

L (P a, P b)

{

push\_back(a);

push\_back(b);

}

};

inline bool operator<(const P a, const P b)

{

return a.X!=b.X ?a.X<b.X :a.Y <b.Y;

}

double cross(P a, P b) //1

{

return imag(conj(a) \* b);

}

double dot(P a, P b) //2

{

return (conj(a)\*b).X;

}

//Orientacion de 3 puntos

int ccw(P a, P b, P c) //3,1 2

{

b-=a;

c-=a;

if(cross(b,c)>0) return +1;

if(cross(b,c)<0) return -1;

if(dot(b,c)<0) return +2;//c-a-b line

if(norm(b)<norm(c)) return -2;//a-b-c line

return 0;

}

//Interseccion de 2 rectas

bool intersectLL (L l, L m) //4,1

{

//non-parallel

return abs(cross(l[1]-l[0], m[1]-m[0])) > EPS

|| abs(cross(l[1]-l[0], m[0]-l[0])) < EPS;

} //same-line

//Punto interseccion recta recta

P crosspoint(L l, L m) //5,1

{

double A = cross( l[1]-l[0], m[1]-m[0]);

double B = cross( l[1]-l[0], l[1]-m[0]);

if(abs(A)<EPS && abs(B)<EPS)

return m[0]; //Same line

if(abs(A)<EPS) return P(0,0);//parallels

return m[0] + B / A \* (m [1] - m [0]);

}

//Interseccion recta y segmento

bool intersectLS (L l, L s) //6, 1

{

//s[0] is left of l

return cross(l[1]-l[0], s[0]-l[0]) \*

cross(l[1]-l[0],s[1]-l[0])<EPS;

} //s[1] is right of l

//Interseccion recta y punto

bool intersectLP (L l, P p) //7,1

{

return abs(cross(l[1]-p, l[0]-p))<EPS;

}

//Interseccion de 2 segmento

bool intersectSS (L s, L t) //8,3

{

FOR(i,2)FOR(j,2) if(abs(s[i]-t[j])<EPS)

return 1; // same point

return ccw(s[0],s[1],t[0])\*ccw(s[0],s[1],t[1])<=0

&& ccw(t[0],t[1],s[0])\*ccw(t[0],t[1],s[1])<=0;

}

//Interseccion segmento y punto

bool intersectSP (L s,P p) //9

{

double a=abs(s[0]-p)+abs(s[1]-p);

return a-abs(s[1]-s[0])<EPS;

}

//Interseccion circulo circulo

pair<P, P> intersectCC(circle a,circle b)

{

P x= b.p - a.p;

P A= conj(x), C = a.r\*a.r\*(x);

P B= (b.r\*b.r-a.r\*a.r-(x)\*conj(x));

P D= B\*B-4.0\*A\*C;

P z1= (-B+sqrt(D)) / (2.0\*A) +a.p;

P z2= (-B-sqrt(D)) / (2.0\*A) +a.p;

return pair<P, P>(z1, z2);

}

//Proyeccion punto recta

P projection(L l,P p) //10,2

{

double t=dot(p-l[0], l[0]-l[1])/norm(l[0]-l[1]);

return l[0] + t\*(l[0]-l[1]);

}

//Refleccion punto recta

P reflection(L l, P p) //11, 10

{

return p +(P(2,0) \*(projection(l,p)-p));

}

//Distancia recta punto

double distanceLP(L l,P p) //12, 10

{

return abs(p - projection(l,p));

}

//Distancia recta recta

double distanceLL(L a, L b) //13,4 12

{

if(intersectLL(a,b)) return 0;

return distanceLP(a,b[0]);

}

//Distancia recta segmento

double distanceLS(L l, L s) //14,7 12

{

if(intersectLS(l,s)) return 0;

return min(distanceLP(l,s[0]),distanceLP(l,s[1]));

}

//Distancia segmento punto

double distanceSP(L s, P p) //15, 10 9

{

const P r = projection(s,p);

if (intersectSP(s,r)) return abs(r-p);

return min( abs(s[0]-p), abs(s[1]-p) );

}

//distancia segmento segmento

double distanceSS (L s, L t) //16,8 15

{

if (intersectSS(s, t)) return 0;

double a=oo,b=oo;

FOR(i,2) a=min(a, distanceSP(s,t[i]));

FOR(i,2) b=min(b, distanceSP(t,s[i]));

return min(a,b);

}

//Centro de circunferencia dado 3 puntos

P circunferenceCenter(P a, P b, P c) //17

{

P x =1.0/conj(b-a), y=1.0/conj(c-a);

return (y-x)/(conj(x)\*y-x\*conj(y)) +a;

}

double anguloEjeX(P a) //18,1 2

{

P b = P(1,0);

if(dot(b,a)/(abs(a)\*abs(b))==1) return 0;

if(dot(b,a)/(abs(a)\*abs(b))==-1) return PI;

double aux=asin(cross(b,a)/(abs(a)\*abs(b)));

if(a.X<0 && a.Y>0) aux+=PI/2;

if(a.X<0 && a.Y<0) aux-=PI/2;

if(aux<0) aux += 2\*PI;

return aux;

}

double anguloEntreVectores(P a, P b) //19,18

{

double aa = anguloEjeX(a);

double bb = anguloEjeX(b);

double r = bb - aa;

if (r<0) r+=2\*PI;

return r;

}

double anguloEntre3Puntos(P a, P b, P c) //20,19

{

a-=b;

c-=b;

return anguloEntreVectores(a,b);

}

int pointInPolygon(Pol pol, P p) //22, 1 2

{

bool in = false;

int n=pol.size();

FOR(i,n)

{

P a= pol[i] - p, b= pol[(i+1)%n]-p;

if(a.Y > b.Y) swap(a,b);

if(a.Y<=0 && 0 < b.Y)

if (cross(a,b)<0) in = !in;

if(abs(cross(a,b))<=EPS &&dot(a,b)<=0)

return true; // ON

}

return in; // IN | OUT

}

double area(Pol pol) //25, 1

{

double A=0;

int n=pol.size();

FOR(i,n)

A+=cross(pol[i],pol[(i+1)%n]);

return A/2.;

}

### Area de Union de rectangulos

struct T

{

int x,y1,y2,IoF;

T(int a=0,int b=0,int c=0,int d=0)

{

x=a;

y1=b;

y2=c;

IoF=d;

}

} L[200005];

int B[200005],r1;

int B1[200005],r2;

bool com(const T &s,const T &p)

{

return s.x<p.x;

}

int MAXY;

int Stree[3000005];

long long cant[3000005];

int r;

void update(int node,int ini,int fin,int y1,int y2,int IoF)

{

if(ini>y2 || fin<y1)

return;

if(ini>=y1 && fin<=y2)

Stree[node]+=IoF;

else

{

int piv=(ini+fin)/2;

update(2\*node,ini,piv,y1,y2,IoF);

update(2\*node+1,piv+1,fin,y1,y2,IoF);

}

if(Stree[node]==0)

{

if(ini==fin)

cant[node]=0;

else

cant[node]=(long long)cant[2\*node]+cant[2\*node+1];

}

else

cant[node]=(long long)B[fin]-B[ini-1];

}

int main()

{

int N;

scanf("%d",&N);

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

{

int x1,y1,x2,y2;

scanf("%d%d%d%d",&x1,&x2,&y1,&y2);

if(x1>x2)

swap(x1,x2);

if(y1>y2)

swap(y1,y2);

L[++r]=T(x1,y1,y2,1);

L[++r]=T(x2,y1,y2,-1);

B1[++r2]=y1;

B1[++r2]=y2;

}

B1[0]=-1;

sort(B1+1,B1+r2+1);

for(int i=1; i<=r2; i++)

if(B1[i]!=B1[i-1])

B[++r1]=B1[i];

sort(L+1,L+r+1,com);

int last=L[1].x;

long long area=0;

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

{

long long temp=(long long)L[i].x-last;

temp=(long long)temp\*cant[1];

area=(long long)area+temp;

last=L[i].x;

int I=lower\_bound(B+1,B+r1+1,L[i].y1)-B;

int F=lower\_bound(B+1,B+r1+1,L[i].y2)-B;

update(1,1,r1,I+1,F,L[i].IoF);

}

cout << area;

return 0;

}

### Closest pair points

int square(int n)

{

return n\*n;

}

struct T

{

int x,y,id;

T(int a=0,int b=0)

{

x=a;

y=b;

}

bool operator <(const T &p)const

{

return x<p.x;

}

} P[100005];

double dist(T a,T b)

{

return sqrt(square(a.x-b.x)+square(a.y-b.y));

}

struct compy

{

bool operator()(const T &s,const T &p)const

{

return s.y<p.y;

}

};

multiset<T,compy>MS;

multiset<T,compy>::iterator I,F;

int main()

{

int N;

scanf("%d",&N);

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

scanf("%d%d",&P[i].x,&P[i].y),P[i].id=i;

sort(P+1,P+N+1);

double min\_dist=1<<30;

int s1,s2;

int p=1;

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

{

while(p<i && P[i].x-P[p].x>=min\_dist)

{

MS.erase(MS.find(P[p]));

p++;

}

I=MS.lower\_bound(T(P[i].x,P[i].y-min\_dist-1));

F=MS.upper\_bound(T(P[i].x,P[i].y+min\_dist+1));

while(I!=F)

{

//min\_dist=min(min\_dist,dist(P[i],\*I));

T x=\*I;

if(min\_dist>dist(P[i],x))

{

min\_dist=dist(P[i],x);

s1=P[i].id;

s2=x.id;

}

I++;

}

MS.insert(P[i]);

}

printf("%d %d",s1,s2);

return 0;

}

### Convex Hull

struct pt

{

double x, y;

};

bool cmp (pt a, pt b)

{

return (a.x < b.x) || (a.x == b.x && a.y < b.y);

}

bool cw (pt a, pt b, pt c)

{

return a.x\*(b.y-c.y)+b.x\*(c.y-a.y)+c.x\*(a.y-b.y) <= 0;

}

bool ccw (pt a, pt b, pt c)

{

return a.x\*(b.y-c.y)+b.x\*(c.y-a.y)+c.x\*(a.y-b.y) >= 0;

}

void convex\_hull (vector<pt> & a)

{

if (a.size() == 1) return;

sort (a.begin(), a.end(), &cmp);

pt p1 = a[0], p2 = a.back();

vector<pt> up, down;

up.push\_back (p1);

down.push\_back (p1);

for (size\_t i=1; i<a.size(); ++i)

{

if (i==a.size()-1 || cw (p1, a[i], p2))

{

while (up.size()>=2 && !cw (up[up.size()-2], up[up.size()-1], a[i]))

up.pop\_back();

up.push\_back (a[i]);

}

if (i==a.size()-1 || ccw (p1, a[i], p2))

{

while (down.size()>=2 && !ccw (down[down.size()-2], down[down.size()-1], a[i]))

down.pop\_back();

down.push\_back (a[i]);

}

}

a.clear();

for (size\_t i=0; i<up.size(); ++i)

a.push\_back (up[i]);

for (size\_t i=down.size()-2; i>0; --i)

a.push\_back (down[i]);

}

typedef double ll;

ll area( pt a, pt b, pt c )

{

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

}

### Perimeter of the union of rectangles

#define MAXC 1 << 17

#define LEFT( v ) 2 \* ( v ) + 1

#define RIGHT( v ) 2 \* ( v ) + 2

int N, last, size;

long long perimeter;

struct event {

int start, lo, hi, flag;

bool operator < ( const event &e ) const { return start < e.start; }

};

vector< event > events[2];

int amount[ 3 \* MAXC ], times[ 3 \* MAXC ];

void update( int node, int lo, int hi, int& start, int& end, int& value ) {

if ( lo > end || hi < start ) return;

if ( start <= lo && hi <= end )

times[node] += value;

else {

int mid = ( lo + hi ) / 2;

update( LEFT( node ), lo, mid, start, end, value );

update( RIGHT( node ), mid + 1, hi, start, end, value );

}

if ( !times[node] )

amount[node] = ( lo == hi ) ? 0 :

amount[ LEFT( node ) ] + amount[ RIGHT( node ) ];

else amount[node] = hi - lo + 1;

}

void lineSweep( vector< event >& ls ) {

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

last = 0; size = ls.size();

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

update( 0, 0, MAXC, ls[i].lo, ls[i].hi, ls[i].flag );

perimeter += abs( last - amount[0] );

last = amount[0];

}

}

int main() {

scanf( "%d", &N );

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

int x1, x2, y1, y2;

scanf( "%d %d %d %d", &x1, &y1, &x2, &y2 );

if ( x1 > x2 ) swap( x1, x2 );

if ( y1 > y2 ) swap( y1, y2 );

events[0].push\_back( ( event ) { y1, x1, x2 - 1, +1 } );

events[0].push\_back( ( event ) { y2, x1, x2 - 1, -1 } );

events[1].push\_back( ( event ) { x1, y1, y2 - 1, +1 } );

events[1].push\_back( ( event ) { x2, y1, y2 - 1, -1 } );

}

lineSweep( events[0] );

lineSweep( events[1] );

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

return 0;

}

### Pick Theorem

/\*

Pick's theorem

A = I + B/2 - 1:

A = Area of the polygon

I = Number of integer coordinates points inside

B = Number of integer coordinates points on the boundary

Polygon's vertex must have integer coordinates

Tested: LightOJ 1418

Complexity: O(n)

\*/

typedef long long ll;

typedef complex<ll> point;

struct segment { point p, q; };

ll points\_on\_segment(const segment &s)

{

point p = s.p - s.q;

return \_\_gcd(abs(p.real()), abs(p.imag()));

}

// <Lattice points (not in boundary), Lattice points on boundary>

pair<ll, ll> pick\_theorem(polygon &P)

{

ll A = area2(P), B = 0, I = 0;

for (int i = 0, n = P.size(); i < n; ++i)

B += points\_on\_segment({P[i], P[NEXT(i)]});

A = abs(A);

I = (A - B) / 2 + 1;

return {I, B};

}

### Par de puntos más lejanos (Manhatan)

int N, mays = -1 << 30, mayd = -1 << 30, mens = 1 << 30, mend = 1 << 30;

int main()

{

scanf("%d", &N);

int x, y;

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

{

scanf("%d%d", &x, &y);

mens = min( mens, x + y );

mays = max( mays, x + y );

mend = min( mend, x - y );

mayd = max( mayd, x - y );

}

printf("%d", max( mays - mens, mayd - mend ));

return 0;

}

### Mayor ctdad de ptos dento de un circulo de radio R

//Dado un radio R encontrar la mayor ctdad de ptos dentro de un circulo con ese radio

#include <bits/stdc++.h>

using namespace std;

const int maxn = 2003;

typedef double db;

int N;

db R;

db X[maxn], Y[maxn];

db sq( db x ){

return x\*x;

}

db dist( int a, int b ){

return sqrt( sq(X[a]-X[b]) + sq(Y[a]-Y[b]) );

}

db area( db a, db b, db c ){

db sp = (a+b+c)/2.;

return sqrt( sp\*(sp-a)\*(sp-b)\*(sp-c) );

}

struct dos{

db ang;

int id, iof;

}I[2\*maxn];

set<int>S;

int solve( ){

db pi = 2\*acos(0);

int ret = 0;

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

int c = 0;

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

if( i == j )continue;

db d = dist(i,j);

if( d > 2\*R )continue;

db alpha = atan2( Y[j]-Y[i], X[j]-X[i] );

db A = area(R,R,d);

db beta = asin((2\*A)/(R\*d));

db ini = alpha-beta;

//if( ini < 0 )ini += 2\*pi;

db fin = alpha+beta;

//if( fin > 2\*pi )fin -= 2\*pi;

I[++c] = dos{ini,j,1};

I[++c] = dos{fin,j,-1};

}

S.clear();

sort(I+1,I+c+1,[]( const dos &s1, const dos &s2){

if( s1.ang != s2.ang )return s1.ang < s2.ang;

return s1.iof > s2.iof;

});

int mc = 0;

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

if( I[j].iof == 1 ){

S.insert(I[j].id);

}

mc = max(mc, (int)S.size());

if( I[j].iof == -1 ){

if( S.find(I[j].id) != S.end() )

S.erase(S.find(I[j].id));

}

}

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

if( I[j].iof == 1 ){

S.insert(I[j].id);

}

mc = max(mc, (int)S.size());

if( I[j].iof == -1 ){

if( S.find(I[j].id) != S.end() )

S.erase(S.find(I[j].id));

}

}

ret = max(ret,mc+1);

}

return ret;

}

int main()

{

cin.tie(0);

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

#ifdef acm

freopen("a.in","r",stdin);

#endif // acm

while( cin >> N >> R && (N+R) ){

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

cin >> X[i] >> Y[i];

cout << "It is possible to cover " << solve() << " points.\n";

}

return 0;

}

### Diameter of a planar point set

O(N \* logN) with rotating calipers method

typedef pair<double, double> point;

bool cw(const point &a, const point &b, const point &c)

{

return (b.first - a.first) \* (c.second - a.second) - (b.second - a.second) \* (c.first - a.first) < 0;

}

vector<point> convexHull(vector<point> p)

{

int n = p.size();

if (n <= 1)

return p;

int k = 0;

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

vector<point> q(n \* 2);

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

for (; k >= 2 && !cw(q[k - 2], q[k - 1], p[i]); --k)

;

for (int i = n - 2, t = k; i >= 0; q[k++] = p[i--])

for (; k > t && !cw(q[k - 2], q[k - 1], p[i]); --k)

;

q.resize(k - 1 - (q[0] == q[1]));

return q;

}

double area(const point &a, const point &b, const point &c)

{

return abs((b.first - a.first) \* (c.second - a.second) - (b.second - a.second) \* (c.first - a.first));

}

double dist(const point &a, const point &b)

{

return hypot(a.first - b.first, a.second - b.second);

}

double diameter(const vector<point> &p)

{

vector<point> h = convexHull(p);

int m = h.size();

if (m == 1)

return 0;

if (m == 2)

return dist(h[0], h[1]);

int k = 1;

while (area(h[m - 1], h[0], h[(k + 1) % m]) > area(h[m - 1], h[0], h[k]))

++k;

double res = 0;

for (int i = 0, j = k; i <= k && j < m; i++)

{

res = max(res, dist(h[i], h[j]));

while (j < m && area(h[i], h[(i + 1) % m], h[(j + 1) % m]) > area(h[i], h[(i + 1) % m], h[j]))

{

res = max(res, dist(h[i], h[(j + 1) % m]));

++j;

}

}

return res;

}

int main()

{

vector<point> points(4);

points[0] = point(0, 0);

points[1] = point(3, 0);

points[2] = point(0, 3);

points[3] = point(1, 1);

double d = diameter(points);

cout << d << endl;

}

# String

### AHO\_CORASICK

const int MAXL = 100005;

int NODO[MAXL];

struct node

{

int dic[27], parent, link, l;

node( int p )

{

parent = p;

link = l = 0;

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

dic[i] = -1;

}

};

typedef pair<int,int>par;

queue<par>Q;

vector<node>tree;

char W[MAXL], S[MAXL], SOL[MAXL];

int f( char c )

{

return c-'a';

}

void add\_word( )

{

int l = strlen(W), nod = 0;

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

{

if( tree[nod].dic[f(W[i])] == -1 )

tree[nod].dic[f(W[i])] = tree.size(), tree.push\_back(node(nod));

nod = tree[nod].dic[f(W[i])];

}

tree[nod].l = l;

}

void links( )

{

Q.push(par(0,0));

int nodo, car;

while( !Q.empty() )

{

nodo = Q.front().first;

car = Q.front().second;

Q.pop();

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

if( tree[nodo].dic[i] != -1 )

Q.push(par(tree[nodo].dic[i],i));

if( nodo == 0 || tree[0].dic[car] == nodo )continue;

int &link = tree[nodo].link;

link = tree[tree[nodo].parent].link;

while( link && tree[link].dic[car] == -1 )

link = tree[link].link;

link = tree[link].dic[car];

if( link == -1 )link++;

tree[nodo].l = max(tree[nodo].l,tree[link].l);

}

}

int autom[MAXL\*5][28];

bool OK[MAXL\*5][28];

int go( int nodo, int car )

{

if( OK[nodo][car] )

return autom[nodo][car];

OK[nodo][car] = 1;

if( tree[nodo].dic[car] != -1 )

return autom[nodo][car] = tree[nodo].dic[car];

int link = tree[nodo].link;

while( link && tree[link].dic[car] == -1 && !OK[link][car] )

link = tree[link].link;

if( OK[link][car] )

return autom[nodo][car] = autom[link][car];

link = tree[link].dic[car];

if( link == -1 )link++;

return autom[nodo][car] = link;

}

int main()

{

tree.push\_back(node(0));

scanf("%s",S+1);

int N;

scanf("%d",&N);

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

{

scanf("%s",W);

add\_word();

}

links();

int nod = 0, pos = 0;

N = strlen(S+1);

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

{

nod = go(nod,f(S[i]));

pos++;

NODO[pos] = nod;

SOL[pos] = S[i];

if( tree[nod].l )

{

pos -= tree[nod].l;

nod = NODO[pos];

}

}

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

printf("%c",SOL[i]);

return 0;

}

### 2D PATTERN MATCHING

const int MAXN=2e3+10;

char P[MAXN][MAXN];

char T[MAXN][MAXN];

typedef unsigned long long ull;

ull hp[2][MAXN][MAXN];

ull ht[2][MAXN][MAXN];

ull bas[2] = {1e9 + 7, 1e9 + 11};

ull po[2][MAXN];

int np,mp,nt,mt;

int F[MAXN];

ull hash\_P(int k,int u)

{

return hp[u][k][np] - hp[u][k][0]\*po[u][np];

}

ull hash\_T(int k,int i,int f,int u)

{

return ht[u][k][f-1] - ht[u][k][i-1]\*po[u][f-i];

}

int main()

{

scanf("%d%d%d%d",&np,&mp,&nt,&mt);

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

scanf("%s",P[i]+1);

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

scanf("%s",T[i]+1);

po[0][0] = po[1][0] = 1;

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

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

po[j][i] = po[j][i-1]\*bas[j];

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

hp[1][i][0] = hp[0][i][0] = 1;

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

for(int k=1; k<=mp; k++)

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

hp[j][k][i] = (hp[j][k][i-1]\*bas[j]) + P[i][k];

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

ht[1][i][0] = ht[0][i][0] = T[i][0];

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

for(int k=1; k<=mt; k++)

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

ht[j][k][i] = (ht[j][k][i-1]\*bas[j]) + T[i][k];

F[1] = 0;

int k=0;

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

{

while(k>0 && (hash\_P(k+1,0)!=hash\_P(i,0) || hash\_P(k+1,1)!=hash\_P(i,1)))

k=F[k];

if(hash\_P(k+1,0)==hash\_P(i,0) &&hash\_P(k+1,1)==hash\_P(i,1)) k++;

F[i]=k;

}

int cont=0;

for(int f=1; f<=nt-np+1; f++)

{

int k=0;

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

{

while((k>0) && (hash\_P(k+1,0)!=hash\_T(i,f,f+np,0) || hash\_P(k+1,1)!=hash\_T(i,f,f+np,1)))

k = F[k];

if(hash\_P(k+1,0)==hash\_T(i,f,f+np,0) && hash\_P(k+1,1)==hash\_T(i,f,f+np,1)) k++;

if(k==mp)

cont++,k = F[k];//found

}

}

cout << cont << '\n';

return 0;

}

### HASHING

const int MAXN = 2e4 + 10;

int N,K ;

int A[MAXN];

typedef unsigned long long ull;

ull h[2][MAXN];

ull bas[2] = {1e9 + 7, 1e9 + 11};

ull po[2][MAXN];

//hash desde i a f sin incluir f, con el primo u

ull hash\_to(int i, int f, int u)

{

return h[u][f-1] - h[u][i-1]\*po[u][f-i];

}

int main()

{

scanf("%d %d",&N,&K);

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

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

po[0][0] = po[1][0] = 1;

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

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

po[j][i] = po[j][i-1]\*bas[j];

h[1][0] = h[0][0] = 1;

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

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

h[j][i] = (h[j][i-1]\*bas[j]) + A[i];

}

### Knuth Morris Pratt

char TEXT[500005],PATT[500005];

int F[500005];

int main()

{

int i = 0, j = -1;

b[0] = -1; // starting values

while (i < m) // pre-process the pattern string P

{

while (j >= 0 && P[i] != P[j]) j = b[j]; // if different, reset j using b

i++;

j++; // if same, advance both pointers

b[i] = j; // observe i = 8, 9, 10, 11, 12 with j = 0, 1, 2, 3, 4

}

int i = 0, j = 0; // starting values

while (i < n) // search through string T

{

while (j >= 0 && T[i] != P[j]) j = b[j]; // if different, reset j using b

i++;

j++; // if same, advance both pointers

if (j == m) // a match found when j == m

{

printf("P is found at index %d in T\n", i - j);

j = b[j]; // prepare j for the next possible match

}

}

return 0;

}

### Longest Square (TANDEMS)

voidbfail(char \*l,int n,char \*r,int m) //fail[i] guarda el mayor sufijo de l, que es sufijo para la posición i en r

{

int it=0;

for(int i=n-1; i>=0; i--) temp[it++]=l[i]; //invierte las dos cadenas y las concatena

for(int i=m-1; i>=0; i--) temp[it++]=r[i];

Zfunction(temp,it);

for(int i=0; i<m; i++) fail[i]=min(z[m+n-i-1],n);

}

voidsqfind(char \*s1,int l1,char \*s2,int l2) //encuentra los cuadrados de la concatenación de l2 y

{

bfail(s1,l1,s2,l2); // l1 centrados en l2 o entre las dos cadenas, que abarcan a l1

Zfunction(s2,l2);

for(int i=l2-1; i>resz; i--)

if(z[i]+fail[i-1]>=i) //implica que hay un cuadrado centrado entre i e i-1

resz=max(resz,i);

for(int i=l2-1; i>=resz; i--) //implica que hay un cuadrado entre

if(fail[i]>=i+1) //implica que hay un cuadrado entre l1 y l2;

resz=max(resz,i+1);

}

voidlsquare(char \*txt,int len)

{

if(len==1) return;

if(len==2)

{

resz=max(resz,int(txt[0]==txt[1]));

return;

}

int n=len/2,m=len-len/2;

char \*s1=txt,\*s2=txt+n;

lsquare(s1,n);

lsquare(s2,m);

sqfind(s1,n,s2,m);

reverse(s1,s1+n);

reverse(s2,s2+m);

sqfind(s2,m,s1,n);

reverse(s1,s1+n);

reverse(s2,s2+m);

}

### Lyndon Descomposition

Dada una string s devuelve la Lyndon decomposition en tiempo lineal usando el algo-

ritmo de Duval. Factoriza s como s 1 s 2 ...s k con s 1 = s 2 = иии = s k y tal que s i es

Lyndon, esto es, es su menor rotacion.

Obtener la mnima rotacin de s:

en la descomposicin de Lyndon de s 2 es el ltimo i < |s|

con el que empieza una Lyndon.

string s;

/ / input string

int n = (int) s. length ();

int i = 0;

while (i <n)

{

int j = i + 1, k = i;

while (j <n & & s [k] <= s [j])

{

if (s [k] <s [j])

k = i;

else

+ + K;

+ + J;

}

while (i <= k)

{

cout <<s. substr (i, j - k) <<'';

i + = j - k;

}

}

### MANACHER

char s[100005];

int r[100005];

int main()

{

scanf("%s",s);

int n=strlen(s);

int i,j,k=0;

for(i=0,j=0; i<2\*n; i+=k,j=max(j-k,0))

{

while(i-j>=0 && i+j+1<2\*n && s[(i-j)/2]==s[(i+j+1)/2])

++j;

r[i]=j;

for(k=1; i>=k && r[i]>=k && r[i-k]!=r[i]-k; ++k)

r[i+k] = min(r[i-k],r[i]-k);

}

for(int i=0; i<2\*n; i++)

printf("%d ",r[i]);

return 0;

}

### MAXIMUN-MINIMUN-SUFFIX

//Dada una string s devuelve un array m[0:n] tal que m[i] contine el mximo sufijo de

//s[0:i+1].

void maximumSuffixArray (char\* s, int\* res)

{

int i = 0, n = strlen(s), j, k;

forn(l, n) res[l] = -1;

while (i < n)

{

j = i + 1;

k = i;

if (res[i] == -1) res[i] = i;

while (j < n and s[k] >= s[j])

{

if (s[k] > s[j]) k = i;

else k++;

if (res[j] == -1) res[j] = i;

j++;

}

while (i <= k) i += j - k;

}

}

Dada una string s devuelve un array m[0:n] tal que m[i] contine el mnimo sufijo de

s[0:i+1].

void minimumSuffixArray (char\* s, int\* res)

{

int i = 0, n = strlen(s), j, k;

while (i < n)

{

j = i + 1;

k = i;

res[i] = i;

while (j < n and s[k] <= s[j])

{

if (s[k] < s[j]) res[j] = k = i;

else res[j] = j - k + res[k], k++;

j++;

}

while (i <= k) i += j - k;

}

}

### Period

char S[1000005];

bool B;

int main()

{

int N;

scanf("%d",&N);

scanf("%s",S+1);

int l=1;

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

{

if(S[i]==S[i-l])

{

if(i%l==0)

printf("%d %d\n",i,i/l),B=1;

}

else

{

if(S[i]==S[1])

l=i-1;

else

l=i;

}

}

if(!B)

printf("0");

return 0;

}

### SMALLEST CYCLIC SHIFT

string min\_cyclic\_shift (string s)

{

s + = s;

int n = (int) s. length ();

int i = 0, ans = 0;

while (i <n / 2)

{

ans = i;

int j = i + 1, k = i;

while (j <n & & s [k] <= s [j])

{

if (s [k] <s [j]) k = i;

else + + k;

+ + j;

}

while (i <= k) i + = j - k;

}

return s. substr (ans, n / 2);

}

### SUFFIX ARRAY (N log^2 N)

struct T

{

int nr[2],p;

} L[200005];

bool com(const T &s,const T &p)

{

if(s.nr[0]!=p.nr[0])

return s.nr[0]<p.nr[0];

return s.nr[1]<p.nr[1];

}

int N,K,stp,delta;

char st[200005];

int P[20][200005];

int pos[200005];

int LCP(int x,int y)

{

int ret=0;

for(int k=stp-1; k>=0 && x<N && y<N; k--)

if (P[k][x]==P[k][y])

{

x+=(1<<k);

y+=(1<<k);

ret+=(1<<k);

}

return ret;

}

int main ()

{

gets( st );

N = strlen( st );

/\*copy( st, st + N , st + N );

reverse( st + N, st + 2 \* N );

N \*= 2;\*/

/\* Suffix Array Computation \*/

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

P[0][i]=st[i]-'A';

/\* build suffix array \*/

for(stp=1,delta=1; (delta>>1) < N; stp++,delta<<=1)

{

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

{

L[i].nr[0]=P[stp - 1][i];

L[i].p = i;

if(i+delta<N)

L[i].nr[1]=P[stp-1][i+delta];

else

L[i].nr[1]=-1;

}

sort(L,L+N,com);

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

if(i>0 && L[i].nr[0] == L[i - 1].nr[0] && L[i].nr[1] == L[i - 1].nr[1] )

P[stp][L[i].p]=P[stp][L[i - 1].p];

else

P[stp][L[i].p]=i;

}

/\* pos gives me the position of suffix with order at P[stp - 1][i] \*/

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

pos[P[stp - 1][i]]=i;

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

printf("%d %s\n",pos[i],st+pos[i]);

/\*Computing the LCP ( Longest Comon Prefix ) between 2 suffixes, one starting at

a, and the other starting at b ( a & b are provided by queries ) \*/

/\*int solution = 1;

for (int i = 0 ; i < ( N / 2 ) - 1 ; i++ ) {

// odd & even length

if ( i ) // n - i < n

solution =max( 2 \* LCP( i + 1, N - i ) + 1, solution);

solution = max( 2 \* LCP( i + 1, N - i - 1), solution );

}\*/

//printf("%d",solution);

// $ < # < @

// LCP 3 suffixes

return 0;

}

### SUFFIX ARRAY (N log N)

#define ll long long

#define MAX 500005

char s[MAX];

int SA[MAX],wa[MAX], wb[MAX], we[MAX], wv[MAX],S[MAX],A[MAX];

void Sufix\_Array(char \*cad,int \*SA,int N)

{

N++;

int i, j, p, \*x = wa, \*y = wb, range = 256;

memset(we, 0, range \* sizeof(int));

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

we[ x[i] = cad[i] ]++;

for (i = 1; i < range; i++) we[i] += we[ i-1 ];

for (i = N - 1; i >= 0; i--)

SA[ --we[ x[i] ] ] = i;

for (j = p = 1; p < N; j <<= 1, range = p)

{

for (p = 0, i = N - j; i < N; y[p++] = i, i++) ;

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

if (SA[i] >= j) y[p++] = SA[i] - j;

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

wv[i] = x[ y[i] ];

memset(we, 0, range \* sizeof(int));

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

we[ wv[i] ]++;

for (i = 1; i < range; i++) we[i] += we[i-1];

for (i = N-1; i >= 0; i--) SA[--we[wv[i]]] = y[i];

swap(x, y);

x[SA[0]] = 0;

for (p = i = 1; i < N; i++)

if(y[SA[i]] == y[SA[i-1]] && y[SA[i]+j] == y[SA[i-1]+j])

x[SA[i]] = p - 1;

else x[SA[i]] = p++;

}

N--;

}

int rank[MAX], LCP [MAX];

void FindLCP(char \*cad, int \*SA, int N)

{

int i, j, k;

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

rank[ SA[i] ] = i;

for (k = i = 0; i < N; LCP [rank[i++]] = k)

for (k ? k-- : 0,j = SA[rank[i]-1]; cad[i + k] == cad[j + k];

k++);

}

char cad[MAX];

int n;

int main()

{

scanf("%s", cad);

n = strlen(cad);

Sufix\_Array(cad, SA, n);

FindLCP(cad, SA, n);

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

printf("%d %s\n",SA[i],cad+SA[i]);

return 0;

}

### Z and Prefix function

string s;

int z[100005];

int main()

{

cin >> s;

int n = ( int ) s.length ( ) ;

vector < int > pi ( n ) ;

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

{

int j = pi [ I - 1 ] ;

while ( j > 0 && s [ I ] != s [ j ] )

j = pi [ j - 1 ] ;

if ( s [ I ] == s [ j ] ) ++ j ;

pi [ I ] = j ;

}

//cantidad de veces que aparece el prefijo de tamaño

//i en la cadena

vector < int > ans ( n + 1 ) ;

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

++ ans [ pi [ I ] ] ;

for ( int I = n - 1 ; I > 0 ; -- I )

ans [ pi [ I - 1 ] ] += ans [ I ] ;

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

printf("%d ",ans[i]+1);

//Given a string S of length n,

//the Z Algorithm produces an array Z

//where Z[i] is the length of the longest

//substring starting from S[i] which is also a prefix of S

int L = 0, R = 0;

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

{

if (i > R)

{

L = R = i;

while (R < n && s[R-L] == s[R]) R++;

z[i] = R-L;

R--;

}

else

{

int k = i-L;

if (z[k] < R-i+1) z[i] = z[k];

else

{

L = i;

while (R < n && s[R-L] == s[R]) R++;

z[i] = R-L;

R--;

}

}

}

cout << '\n';

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

printf("%d ",z[i]);

return 0;

}

### Palindromic Tree

#define maxn 1000

struct Node

{

int start, end, length, insertEdg[26], suffixEdg;

};

Node tree[maxn], root1, root2;

int currNode, ptr;

string s;

void insert(int idx)

{

int tmp = currNode;

while (true){

int curLength = tree[tmp].length;

if (idx - curLength >= 1 and s[idx] == s[idx-curLength-1])

break;

tmp = tree[tmp].suffixEdg;

}

if(tree[tmp].insertEdg[s[idx]-'a'] != 0){

currNode = tree[tmp].insertEdg[s[idx]-'a'];

return;

}

ptr++;

tree[tmp].insertEdg[s[idx]-'a'] = ptr;

tree[ptr].length = tree[tmp].length + 2;

tree[ptr].end = idx;

tree[ptr].start = idx - tree[ptr].length + 1;

tmp = tree[tmp].suffixEdg;

currNode = ptr;

if (tree[currNode].length == 1)

{

tree[currNode].suffixEdg = 2;

return;

}

while (true)

{

int curLength = tree[tmp].length;

if (idx-curLength >= 1 and s[idx] == s[idx-curLength-1])

break;

tmp = tree[tmp].suffixEdg;

}

tree[currNode].suffixEdg = tree[tmp].insertEdg[s[idx]-'a'];

//do things with the tree...

}

void init\_tree(){

root1.length = -1;

root1.suffixEdg = 1;

root2.length = 0;

root2.suffixEdg = 1;

tree[1] = root1;

tree[2] = root2;

ptr = 2;

currNode = 1;

}

int main()

{

init\_tree();

s = "abcbab";

int l = s.length();

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

insert(i);

cout << "All distinct palindromic substring for "

<< s << " : \n";

for (int i=3; i<=ptr; i++)

{

cout << i-2 << ") ";

for (int j=tree[i].start; j<=tree[i].end; j++)

cout << s[j];

cout << endl;

}

return 0;

}

# Grafos

### Componentes Biconexas

vector<int>ID[1005];//id de las aristas en que esta presente cada nodo

int TD[1005],LOW[1005];

int dc\_time;

int num;

int CB[10005];

bool mark[10005];

stack<int>P;

struct T

{

int nod,nn;

bool band;

T(int x=0,int y=0,bool b=0)

{

nod=x;

nn=y;

band=b;

}

int nextn(int x)

{

if(x==nod)

return nn;

else

return nod;

}

} edge[10005];

void BCC( int nod )

{

TD[nod] = LOW[nod] = ++dc\_time;

int newn, id;

vector<int>::iterator it;

for( it = ID[nod].begin(); it != ID[nod].end(); it ++ )

{

id = \*it;

newn = edge[id].nextn( nod );

if( !mark[id] )

{

P.push( id );

mark[id] = true;

}

if( TD[newn] )

{

LOW[nod] = min( LOW[nod], TD[newn] );

continue;

}

BCC( newn );

LOW[nod] = min( LOW[newn], LOW[nod] );

if( TD[nod] <= LOW[newn] )

{

num ++;

while( !CB[id] )

{

CB[P.top()] = num;

P.pop();

}

}

}

}

int main()

{

int n,m;

scanf("%d%d",&n,&m);

int a,b;

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

{

scanf("%d%d",&a,&b);

ID[a].push\_back(i);

ID[b].push\_back(i);

edge[i]=T(a,b);

}

BCC(1);

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

cout << CB[i] << ' ';

return 0;

}

### Bridges and Articulation Points

typedef pair<int,int>par;

vector<int>ID[1005];//id de las aristas en que esta presente cada nodo

int TD[1005],LOW[1005];

int dc\_time;

bool AP[10005];

stack<par>puente;

int a,b;

struct T

{

int nod,nn;

bool band;

T(int x=0,int y=0,bool b=0)

{

nod=x;

nn=y;

band=b;

}

int nextn(int x)

{

if(x==nod)

return nn;

else

return nod;

}

} edge[10005];

void bridges\_PtoArt ( int nod )

{

int newn, num;

vector<int>::iterator it;

TD[nod] = LOW[nod] = ++ dc\_time;

for(it = ID[nod].begin(); it != ID[nod].end(); it ++)

{

num = \*it;

newn = edge[num].nextn( nod );

if( edge[num].band ) continue;

edge[num].band = true;

if( TD[newn] )

{

LOW[nod] = min( LOW[nod], TD[newn] );

continue;

}

bridges\_PtoArt( newn );

LOW[nod] = min( LOW[nod], LOW[newn] );

if(TD[nod] < LOW[newn])

puente.push(par( nod, newn ));

if( (TD[nod] == 1 && TD[newn] > 2 ) ||

( TD[nod] != 1 && TD[nod] <= LOW[newn] ) )

AP[nod] = true;

}

}

int main()

{

int n,m;

scanf("%d%d",&n,&m);

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

{

scanf("%d%d",&a,&b);

ID[a].push\_back(i);

ID[b].push\_back(i);

edge[i]=T(a,b);

}

bridges\_PtoArt(1);

return 0;

}

### Centroid descomposition

const int maxn = 200003, maxk = 1000006;

int N, K, dp[maxk], cost[maxk];

struct par{

int b, c;

};

vector<par>G[maxn];

int mark[maxn], T[maxn];

void dfs( int nodo, int pad ){

T[nodo] = 0;

for( auto i : G[nodo] ){

if( i.b == pad || mark[i.b] )continue;

dfs(i.b,nodo);

T[nodo] += T[i.b];

}

T[nodo]++;

}

int dfs\_c( int nodo, int pad, int c ){

for( auto i : G[nodo] ){

if( i.b != pad && !mark[i.b] && 2\*T[i.b] >= c )

return dfs\_c(i.b,nodo,c);

}

return nodo;

}

set<int>S;

int sol = 1 << 30;

void query( int nodo, int pad, int c, int ca ){

if( c <= K && dp[K-c] )

sol = min(sol,ca+cost[K-c]);

if( c > K )return;

for( auto i : G[nodo] ){

if( i.b == pad || mark[i.b] )continue;

query(i.b,nodo,c+i.c,ca+1);

}

}

void update( int nodo, int pad, int c, int ca ){

if( c <= K )

S.insert(c), dp[c] = 1, cost[c] = min(cost[c],ca);

if( c > K )return;

for( auto i : G[nodo] ){

if( i.b == pad || mark[i.b] )continue;

update(i.b,nodo,c+i.c,ca+1);

}

}

void centroid( int nodo, int pad ){

dfs(nodo,pad);

int centro = dfs\_c( nodo, pad, T[nodo] );

mark[centro] = 1;

S.clear();

S.insert(0);

dp[0] = 1;

cost[0] = 0;

for( auto i : G[centro] ){

if( i.b == pad || mark[i.b] )continue;

query(i.b,nodo,i.c,1);

update(i.b,nodo,i.c,1);

}

for( auto i : S )

dp[i] = 0, cost[i] = 1 << 30;

for( auto i : G[centro] ){

if( i.b != pad && !mark[i.b] )

centroid(i.b,centro);

}

}

int main()

{

cin.tie(0);

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

#ifdef acm

freopen("a.in","r",stdin);

#endif // acm

int a, b, c;

cin >> N >> K;

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

cost[i] = 1 << 30, dp[i] = 0;

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

cin >> a >> b >> c;

G[a].push\_back(par{b,c});

G[b].push\_back(par{a,c});

}

centroid(1,-1);

if( sol == 1 << 30 )cout << "-1";

else cout << sol;

return 0;

}

### Centroid Descomposition (centroid tree)

const int MAXN=250005;

int P[MAXN];

int HD[MAXN],HI[MAXN];

int subsize[MAXN],Ant[MAXN];

vector<int>ady[MAXN];

bool mk[MAXN];

int ref\_pos;

long long sum[MAXN],contribution[MAXN],cont[MAXN];

void DFS1(int np, int prev){

subsize[np] = 1;

int l = ady[np].size();

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

int nh = ady[np][i];

if(nh != prev && !mk[nh]){

DFS1(nh, np);

subsize[np] += subsize[nh];

}

}

}

int DFS2(int np, int prev){

int l = ady[np].size();

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

int nh = ady[np][i];

if(nh != prev && !mk[nh] && subsize[nh] > subsize[ref\_pos]/2)

return DFS2(nh, np);

}

return np;

}

void Descomposition(int root, int prev){

ref\_pos = root;

DFS1(root, root);

int centroid = DFS2(root, root);

Ant[centroid] = prev;

mk[centroid] = true;

int l = ady[centroid].size();

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

int nh = ady[centroid][i];

if(!mk[nh])

Descomposition(nh, centroid);

}

}

struct LCA {

int T[100005][20], L[100005];

void DFS0(int np, int prev){

T[np][0] = prev;

L[np] = L[prev]+1;

int l = ady[np].size();

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

int nh = ady[np][i];

if(nh != prev)

DFS0(nh, np);

}

}

void Build(int n){

DFS0(1, 0);

int lg = log2(n);

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

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

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

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

}

int Query(int x, int y){

int res = 0;

if(L[x] < L[y])swap(x, y);

int lg = (int)log2(L[x]);

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

if(L[x] - (1 << i) >= L[y] && T[x][i])

x = T[x][i], res += (1 << i);

if(x == y)return res;

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

if(T[x][i] != T[y][i] && T[x][i])

x = T[x][i], y = T[y][i], res += 2\*(1 << i);

return res+2;

return T[x][0];

}

}Lc;

void update(int nod)

{

int pos = nod;

cont[pos]++;

while(Ant[pos]!=-1)

{

sum[Ant[pos]]+=Lc.Query(nod,Ant[pos]);

contribution[pos]+=Lc.Query(nod,Ant[pos]);

cont[Ant[pos]]++;

pos=Ant[pos];

}

}

long long query(int nod)

{

long long ret=0;

int pos=nod;

ret=sum[pos];

while(Ant[pos]!=-1)

{

ret+=sum[Ant[pos]]-contribution[pos]+(cont[Ant[pos]]-cont[pos])\*Lc.Query(nod,Ant[pos]);

pos=Ant[pos];

}

return ret;

}

int main()

{

int N;

cin >> N;

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

cin >> P[i];

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

HI[i]=HD[i]=-1;

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

{

int pos=P[1];

while(1)

{

if(P[i]>pos)

{

if(HD[pos]==-1)

{

HD[pos]=P[i];

break;

}

else

pos=HD[pos];

}

else

{

if(HI[pos]==-1)

{

HI[pos]=P[i];

break;

}

else

pos=HI[pos];

}

}

}

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

{

if(HI[i]!=-1)ady[i].push\_back(HI[i]),ady[HI[i]].push\_back(i);

if(HD[i]!=-1)ady[i].push\_back(HD[i]),ady[HD[i]].push\_back(i);

Ant[i]=-1;

}

Lc.Build(N);

Descomposition(P[1], -1);

long long sol=0;

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

{

sol+=query(P[i]);

cout << sol << '\n';

update(P[i]);

}

return 0;

}

### Dinic

const int MAXN = 2005;

const int INF = 1000000000;

struct edge

{

int a, b, cap, flow;

};

int n, s, t, d[MAXN], ptr[MAXN], q[MAXN];

vector<edge> e;

vector<int> g[MAXN];

void add\_edge (int a, int b, int cap)

{

edge e1 = { a, b, cap, 0 };

edge e2 = { b, a, 0, 0 };

g[a].push\_back ((int) e.size());

e.push\_back (e1);

g[b].push\_back ((int) e.size());

e.push\_back (e2);

}

bool bfs()

{

int qh=0, qt=0;

q[qt++] = s;

memset (d, -1, sizeof d);

d[s] = 0;

while (qh < qt && d[t] == -1)

{

int v = q[qh++];

for (size\_t i=0; i<g[v].size(); ++i)

{

int id = g[v][i],

to = e[id].b;

if (d[to] == -1 && e[id].flow < e[id].cap)

{

q[qt++] = to;

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

}

}

}

return d[t] != -1;

}

int dfs (int v, int flow)

{

if (!flow) return 0;

if (v == t) return flow;

for (; ptr[v]<(int)g[v].size(); ++ptr[v])

{

int id = g[v][ptr[v]],

to = e[id].b;

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

int pushed = dfs (to, min (flow, e[id].cap - e[id].flow));

if (pushed)

{

e[id].flow += pushed;

e[id^1].flow -= pushed;

return pushed;

}

}

return 0;

}

int dinic( )

{

int flow = 0;

for (;;)

{

if (!bfs()) break;

memset (ptr, 0, sizeof ptr);

while (int pushed = dfs (s, INF))

flow += pushed;

}

return flow;

}

///para multicasos limpiar el vector g desde 1->cant nodos y el vector e

### Edmonds

//Encuentra el Maximo Matching en un Grafo general donde pueden haber ciclos de tamaño impar

#define maxN 1005

int n;

vector<int>G[maxN];

int match[maxN],Head, Tail,Queue[maxN],Start, Finish,NewBase,Father[maxN],Base[maxN];

bool InQueue[maxN], InPath[maxN], InBlossom[maxN];

void Push(int u)

{

Queue[Tail++]= u;

InQueue[u]= true;

}

int Pop()

{

return Queue[Head++];

}

int FindCommonAncestor(int u, int v)

{

memset(InPath, 0, sizeof(InPath));

while(true)

{

u=Base[u];

InPath[u]= true;

if (u == Start)break;

u= Father[match[u]];

}

while(true)

{

v= Base[v];

if (InPath[v]) break;

v = Father[match[v]];

}

return v;

}

void ResetTrace(int u)

{

int v;

while (Base[u] != NewBase)

{

v= match[u];

InBlossom[Base[u]]= 1;

InBlossom[Base[v]]= 1;

u= Father[v];

if (Base[u] != NewBase)Father[u]=v;

}

}

void BlossomContract(int u,int v)

{

NewBase= FindCommonAncestor(u, v);

memset(InBlossom,0,sizeof(InBlossom));

ResetTrace(u);

ResetTrace(v);

if (Base[u] != NewBase)Father[u]= v;

if (Base[v] != NewBase)Father[v]= u;

for(u=1; u<=n; u++)

if (InBlossom[Base[u]])

{

Base[u]= NewBase;

if (!InQueue[u]) Push(u);

}

}

void FindAugmentingPath()

{

int u,v;

memset(InQueue,false, sizeof(InQueue));

memset(Father,0,sizeof(Father));

for(u=1; u<=n; u++) Base[u]=u;

Head= 1;

Tail= 1;

Push(Start);

Finish = 0;

while (Head < Tail)

{

u= Pop();

//for (v=1;v<=n;v++)

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

{

v = G[u][i];

if ((Base[u]!=Base[v])&&(match[u]!= v))

{

if ((v == Start)||((match[v] > 0)&&(Father[match[v]] > 0)))

BlossomContract(u, v);

else if (Father[v] == 0)

{

Father[v]=u;

if (match[v] > 0)

Push(match[v]);

else

{

Finish=v;

return;

}

}

}

}

}

}

void AugmentPath()

{

int u,v,w;

u=Finish;

while(u > 0)

{

v=Father[u];

w=match[v];

match[v]= u;

match[u]= v;

u= w;

}

}

void PrintMatch()

{

int u,sol;

sol = 0;

for(u=1; u<=n; u++)

if (u < match[u])sol++;

printf("%d\n",sol\*2);

}

void Edmonds()

{

int u;

memset(match,0,sizeof(match));

for(u=1; u<=n; u++)

if (match[u]==0)

{

Start=u;

FindAugmentingPath();

if (Finish > 0) AugmentPath();

}

}

void BuildPreMatch()

{

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

{

if( match[i] )

continue;

for( int j = 0; j < (int)G[i].size(); j++ )

{

if( match[G[i][j]] )

continue;

match[i] = G[i][j];

match[G[i][j]] = i;

break;

}

}

}

int main()

{

scanf("%d",&n);

int a, b;

while( cin >> a >> b )

{

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

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

}

BuildPreMatch();

Edmonds();

PrintMatch();

return 0;

}

### Edmond-Karp

struct edge

{

int nodo, ady, cap, ant;

edge( int a = 0, int b = 0, int c = 0, int d = 0 )

{

nodo = a, ady = b, cap = c, ant = d;

}

} E[600\*600];

int sr, sk, flow[600], parent[600], T[600], pos;

void insertar( int a, int b, int c )

{

E[pos] = edge(a,b,c,T[a]);

T[a] = pos++;

E[pos] = edge(b,a,0,T[b]);

T[b] = pos++;

}

queue<int>Q;

int edmond( )

{

int maxflow = 0;

while( 1 )

{

while( !Q.empty() )

Q.pop();

int nodo, ady, band = 0;

for( int i = sr; i <= sk; i++ )

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

flow[sr] = parent[sr] = 1 << 30;

Q.push(sr);

while( !Q.empty() && !band )

{

nodo = Q.front();

Q.pop();

for( int i = T[nodo]; i != -1; i = E[i].ant )

{

ady = E[i].ady;

if( E[i].cap && parent[ady] == -1 )

{

parent[ady] = i;

flow[ady] = min(flow[nodo],E[i].cap);

Q.push(ady);

if( ady == sk )

{

band = 1;

break;

}

}

}

}

if( parent[sk] == -1 )

break;

maxflow+=flow[sk];

for( int i = sk; i != sr; i = E[parent[i]].nodo )

{

E[parent[i]].cap -= flow[sk];

E[parent[i]^1].cap += flow[sk];

}

}

return maxflow;

}

int main()

{

fill(T,T+sk+2,-1);

return 0;

}

### Hungarian

const int INF = 1000 \* 1000 \* 1000;

int n;

vector <int> a[1003];

vector <int> xy, yx;

vector <char> vx, vy;

vector <int> minrow, mincol;

bool dotry (int i)

{

if (vx [i]) return false;

vx [i] = true;

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

if (a [i][j]-minrow [i]-mincol [j] == 0)

vy [j] = true;

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

if (a [i][j]-minrow[i]-mincol[j] == 0 && yx[j] == -1)

{

xy [i] = j;

yx [j] = i;

return true;

}

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

if (a[i][j]-minrow[i]-mincol[j] == 0 && dotry (yx[j]))

{

xy[i] = j;

yx[j] = i;

return true;

}

return false;

}

void limpiar( )

{

mincol.clear();

minrow.clear();

xy.clear();

yx.clear();

vx.clear();

vy.clear();

}

int hungarian( )

{

limpiar();

mincol.assign (n, INF);

minrow.assign (n, INF);

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

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

minrow [i] = min (minrow[i], a [i][j]);

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

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

mincol [j] = min (mincol[j], a [i][j] - minrow[i]);

xy.assign (n, -1);

yx.assign (n, -1);

for (int c = 0; c <n;)

{

vx.assign (n, 0);

vy.assign (n, 0);

int k = 0;

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

if (xy[i] == -1 && dotry (i))

++k;

c += k;

if (k == 0)

{

int z = INF;

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

if (vx [i])

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

if (!vy[j])

z = min (z, a[i][j]-minrow[i]-mincol[j]);

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

{

if (vx[i]) minrow[i] += z;

if (vy[i]) mincol[i] -= z;

}

}

}

int ans = 0;

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

ans += a[i][xy[i]];

return ans;

}

int main ()

{

scanf("%d",&n);

int x;

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

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

scanf("%d",&x), x \*=-1, a[i].push\_back(x);

printf("%d ",-hungarian());

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

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

a[i][j] \*= -1;

printf("%d",hungarian());

}

### Khun

vector<int>V[300];

int match[300],mark[300];

int N, M, W;

bool used[1003];

bool khun( int nodo )

{

if( mark[nodo] )

return false;

mark[nodo] = 1;

int tam = V[nodo].size();

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

{

int ady = V[nodo][i];

if( ( match[ady] == -1 || khun(match[ady])) )

{

match[ady] = nodo;

return true;

}

}

return false;

}

void PreMatching()

{

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

{

for( int j = 0; j < (int)V[i].size(); j++ )

{

int ady = V[i][j];

if( match[ady] != -1 )

continue;

match[ady] = i;

used[i] = 1;

break;

}

}

}

int main()

{

scanf("%d%d%d",&N,&W,&M);

int a, b;

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

{

scanf("%d%d",&a,&b);

V[a].push\_back(N+b);

}

memset(match,-1,sizeof(int)\*(N+W+1));

PreMatching();

int sol = 0;

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

{

fill(mark,mark+N+1,false);

if( used[i] )

{

sol++;

continue;

}

if( khun(i) )

sol++;

}

printf("%d",sol);

return 0;

}

### Max Flow Min Cost

const int INF = 1e9;

typedef pair <int, int> par;

namespace MCF

{

struct edge

{

int dest, cap, flow, cost, next;

edge() {}

edge(int dest, int cap, int cost, int next) :

dest(dest), cap(cap), cost(cost), next(next)

{

flow = 0;

}

};

const int

MAXV = 10e3 + 5,

MAXE = 10e4 + 5;

int nodes, source, sink, size;

edge g[MAXE];

int first[MAXV], nLast[MAXV], eLast[MAXV], cst[MAXV];

void init(int \_nodes, int \_source, int \_sink)

{

size = 0, nodes = \_nodes, source = \_source;

sink = \_sink;

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

first[i] = -1;

}

void addEdge(int u, int v, int c, int cost)

{

g[size] = edge(v, c, +cost, first[u]);

first[u] = size++;

g[size] = edge(u, 0, -cost, first[v]); //bidirectional, put zero if only directed edges!

first[v] = size++;

}

par maxFlowMinCost()

{

int flow = 0, flowCost = 0;

while(true)

{

priority\_queue <par> q;

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

cst[i] = INF;

cst[source] = 0;

q.push(make\_pair(0, source));

while(!q.empty())

{

int u = q.top().second;

int c = -q.top().first;

q.pop();

if(u == sink)

break;

for(int k = first[u]; k != -1; k = g[k].next)

{

int v = g[k].dest;

int w = g[k].cost + c;

if(g[k].cap > g[k].flow && w < cst[v])

{

cst[v] = w;

nLast[v] = u;

eLast[v] = k;

q.push(make\_pair(-w, v));

}

}

}

if(cst[sink] == INF)

break;

int push = INF;

for(int u = sink; u != source; u = nLast[u])

push = min(push, g[eLast[u]].cap - g[eLast[u]].flow);

flow += push;

flowCost += cst[sink] \* push;

for(int u = sink; u != source; u = nLast[u])

{

g[eLast[u]].flow += push;

g[eLast[u] ^ 1].flow -= push;

}

}

return make\_pair(flow, flowCost);

}

}

### Stable marriage

#define MAX\_N 500

int N,pref\_men[MAX\_N][MAX\_N],pref\_women[MAX\_N][MAX\_N];

int inv[MAX\_N][MAX\_N],cont[MAX\_N],wife[MAX\_N],husband[MAX\_N];

void stable\_marriage(){

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

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

inv[i][pref\_women[i][j]] = j;

fill(cont,cont+N,0);

fill(husband,husband+N,-1);

int m,w,dumped;

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

m = i;

while(m>=0){

while(true){

w = pref\_men[m][cont[m]];

++cont[m];

if(husband[w]<0 || inv[w][m]<inv[w][husband[w]]) break;

}

dumped = husband[w];

husband[w] = m;

wife[m] = w;

m = dumped;

}

}

}

### Stoer Wagner

#define ll long long

#define llu unsigned long long

#define MAXN 55

int G[MAXN][MAXN], w[MAXN], N;

bool A[MAXN], merged[MAXN];

int StoerWagner(int n)

{

int best = 1e8;

for(int i=1; i<n; ++i) merged[i] = 0;

merged[0] = 1;

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

{

A[0] = 1;

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

{

if(merged[i]) continue;

A[i] = 0;

w[i] = G[0][i];

}

int prev = 0,next;

for(int i=n-1-phase; i>=0; --i)

{

// hallar siguiente vertice que no esta en A

next = -1;

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

if(!A[j] && (next==-1 || w[j]>w[next]))

next = j;

A[next] = true;

if(i>0)

{

prev = next;

// actualiza los pesos

for(int j=1; j<n; ++j) if(!A[j])

w[j] += G[next][j];

}

}

if(best>w[next]) best = w[next];

// mezcla s y t

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

{

G[i][prev] += G[next][i];

G[prev][i] += G[next][i];

}

merged[next] = true;

}

return best;

}

int main()

{

char cad[2323];

while (scanf("%s",cad) != EOF)

{

N = strlen(cad);

for(int i=0; i<N; i++) G[0][i] = cad[i]-'0';

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

{

scanf("%s",cad);

for(int j=0; j<N; j++) G[i][j] = cad[j]-'0';

}

//int src, snk;

printf("%d\n", StoerWagner(N));

}

}

### Tarjan SCC

int N, M, low[10005], Td[10005], k, sol;

stack<int> P;

vector<int> V[10005];

bool mark[10005];

void Tarjan\_SCC( int nod )

{

int newn;

vector<int>::iterator it;

Td[nod] = low[nod] = ++ k;

P.push( nod );

for(it = V[nod].begin(); it != V[nod].end(); it ++)

{

newn = \*it;

if( Td[newn] )

{

if( !mark[newn] )

low[nod] = min( low[nod], Td[newn] );

continue;

}

Tarjan\_SCC( newn );

low[nod] = min( low[nod], low[newn] );

}

if( low[nod] == Td[nod] )

{

sol ++;

printf("SCC %d: ", sol);

while( !mark[nod] )

{

printf("%d ", (int)P.top());

mark[(int)P.top()] = true;

P.pop();

}

printf("\n");

}

}

int main()

{

scanf("%d%d", &N, &M);

int a, b;

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

{

scanf("%d%d", &a, &b);

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

}

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

if( !Td[i] )

Tarjan\_SCC ( i );

return 0;

}

### Minimun Vertex Cover Bipartite

// Task : Given a bipartite graph, find its minimum vertex cover

// Running time: O(VE)

#define MAXV 5000

int X, Y, E;

int matched[MAXV];

bool mark[MAXV];

bool T[MAXV];

vector<int> ady[MAXV];

typedef pair<int, bool> par;

queue<par> Q;

bool augment( int nod )

{

if ( nod == -1 ) return true;

int size = ady[nod].size();

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

{

int newn = ady[nod][i];

if ( mark[newn] ) continue ;

mark[newn] = true;

if ( augment( matched[newn] ) )

{

matched[nod] = newn;

matched[newn] = nod;

return true;

}

}

return false;

}

/// X->Y

void Vertex\_Cover\_Bipartite( )

{

/\* Find maximum matching \*/

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

memset( T, false, sizeof( T ) );

int cardinality = 0;

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

{

memset( mark, 0, sizeof( mark ) );

if ( augment( i ) ) cardinality++;

}

/\* Find minimum vertex cover \*/

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

if ( matched[i] == -1 )

{

T[i] = true;

Q.push( par( i, true ) );

}

int nod, newn;

bool band;

while ( !Q.empty() )

{

nod = Q.front().first;

band = Q.front().second;

Q.pop();

int size = ady[nod].size();

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

{

newn = ady[nod][i];

if ( T[newn] ) continue ;

if ( ( band && newn != matched[nod] ) || ( !band && newn == matched[nod] ))

{

T[newn] = true;

Q.push( par( newn, !band ) );

}

}

}

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

//printf( "Minimum Vertex Cover:\n" );

for ( int i = X; i < X + Y; i++ )

if ( T[i] ) printf("vline %d %d %d\n", V[i-X+1].x, V[i-X+1].a, V[i-X+1].b );

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

if ( !T[i] ) printf("hline %d %d %d\n", H[i+1].x, H[i+1].a, H[i+1].b );

}

### Bellman Ford

//los nodos en true en el arreglo fals pertenecen a un ciclo neg

#define MN 1000

struct edge{

int u,v;

long long w;

edge(int uu,int vv,long long ww){

u = uu;v = vv;w = ww;

}

};

long long B[MN],D[MN];

vector<edge> E;

int n,m,u,v,w;

bool fals[MN];

int main() {

E.clear();

scanf("%d",&n);

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

D[i] = 1e9;

scanf("%d",&m);

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

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

E.push\_back(edge(u,v,w));

}

D[1] = 0;

edge e(0,0,0);

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

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

e = E[j];

if(D[e.u] + e.w < D[e.v] && D[e.u]<1e9)

D[e.v] = D[e.u] + e.w;

}

}

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

edge e = E[j];

if(D[e.u] + e.w < D[e.v] && D[e.u]<1e9)

fals[e.v] = true;

}

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

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

edge e = E[j];

if(fals[e.u] ) fals[e.v] = true;

}

}

return 0;

}

### DSU on tree

int cnt[maxn];

bool big[maxn];

void add(int v, int p, int x){

cnt[ col[v] ] += x;

for(auto u: g[v])

if(u != p && !big[u])

add(u, v, x)

}

void dfs(int v, int p, bool keep){

int mx = -1, bigChild = -1;

for(auto u : g[v])

if(u != p && sz[u] > mx)

mx = sz[u], bigChild = u;

for(auto u : g[v])

if(u != p && u != bigChild)

dfs(u, v, 0); // run a dfs on small childs and clear them from cnt

if(bigChild != -1)

dfs(bigChild, v, 1), big[bigChild] = 1; // bigChild marked as big and not cleared from cnt

add(v, p, 1);

//now cnt[c] is the number of vertices in subtree of vertex v that has color c. You can answer the queries easily.

if(bigChild != -1)

big[bigChild] = 0;

if(keep == 0)

add(v, p, -1);

}

### Camino Euleriano

int n,m;

int a,b,c;

vector<int>ID[1001];

int start;

int G[1001];

stack<int>pila;

struct edge

{

int nod,nn;

bool mark;

edge(int a=0,int b=0,bool c=0)

{

nod=a;

nn=b;

mark=c;

}

int next(int x)

{

if(x==nod)

return nn;

return nod;

}

}A[1001];

void euler(int nod)

{

int t=ID[nod].size();

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

{

int id=ID[nod][i];

if(A[id].mark==0)

{

A[id].mark=1;

euler(A[id].next(nod));

}

}

pila.push(nod);

}

int main()

{

freopen("Euler\_P\_C.in","r",stdin);

freopen("Euler\_P\_C.out","w",stdout);

scanf("%d%d",&n,&m);

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

{

scanf("%d%d",&a,&b);

ID[a].push\_back(i);

ID[b].push\_back(i);

G[a]++;

G[b]++;

A[i]=edge(a,b,0);

}

int I=0;

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

{

if(G[i]%2==1)

I++;

if(I>2)

{

printf("NO HAY CAMINO EULERIANO\n");

return 0;

}

}

scanf("%d",&start);

euler(start);

if(I)

printf("EXISTE UN CAMINO EULERIANO\n");

else

printf("EXISTE UN CIRCUITO EULERIANO\n");

for(;!pila.empty();)

{

printf("%d\n",pila.top());

pila.pop();

}

return 0;

}

### Steiner Tree

//el steiner tree es un caso especiel de mst donde en este no se abarca necesariamente todo el conjunto de nodos

//sino un subconjunto estpecifico de estos permitiendo el uso de algunos nodos intermedios

//Complejidad (2^cTerminales)\*N^2 cTerminales->numero de nodos terminales N->numero de nodos totales

//incluidos los terminales.

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

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

dp[i][j] = oo;

//importante caso base cuando el nodo terminal es la raiz de su arbol

//mask = 1 << i i->id del nodo con respecto a los terminales,

//id\_terminal = id del nodo con respecto a todos los nodos

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

dp[1 << i][id\_terminal] = 0;

//dp para computar el minimo steiner tree

//los estados son teniendo a root como raiz del arbol y a mask como los nodos terminales que

//estan en el arbol

for( int mask = 1; mask < (1 << cTerminal); mask++ ){

for( int root = 0; root < N; root ++ ){

for( int smask = mask; smask; smask = (smask-1)&mask ){

dp[mask][root] = min(dp[mask][root], dp[smask][root]+dp[mask^smask][root]);

}

//ciclo para mover la raiz del arbol

if( dp[mask][root] < oo ){

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

dp[mask][j] = min(dp[mask][j],dp[mask][root] + C[root][j]);

}

}

}

//sol = min(dp[1 << Cterminal][i])

### Tarjan (Vertex) BCC

//Tarjan BCC

void BCC( int nod ){

Td[nod] = Low[nod] = ++ k;

int newn, id;

vector<int>::iterator it;

for( it = V[nod].begin(); it != V[nod].end(); it ++ ){

id = \*it;

newn = G[id].nextn( nod );

if( !mark[id] ){

P.push( id );

mark[id] = true;

}

if( Td[newn] ){

Low[nod] = min( Low[nod], Td[newn] );

continue;

}

BCC( newn );

Low[nod] = min( Low[newn], Low[nod] );

if( Td[nod] <= Low[newn] ){

num ++;

while( !CB[id] ){

CB[P.top()] = num;

P.pop();

}

}

}

}

# Data Structure

### AVL

template <class T>

struct avl\_tree

{

struct node

{

T key;

int size, height;

node \*child[2];

node(const T &key) : key(key), size(1), height(1)

{

child[0] = child[1] = 0;

}

} \*root;

typedef node \*pointer;

avl\_tree()

{

root = NULL;

}

pointer find(const T &key)

{

return find(root, key);

}

node \*find(node \*t, const T &key)

{

if (t == NULL) return NULL;

if (key == t->key) return t;

else if (key < t->key) return find(t->child[0], key);

else return find(t->child[1], key);

}

void insert(const T &key)

{

root = insert(root, new node(key));

}

node \*insert(node \*t, node \*x)

{

if (t == NULL) return x;

if (x->key < t->key) t->child[0] = insert(t->child[0], x);

else t->child[1] = insert(t->child[1], x);

t->size += 1;

return balance(t);

}

void erase(const T &key)

{

root = erase(root, key);

}

node \*erase(node \*t, const T &x)

{

if (t == NULL) return NULL;

if (x == t->key)

{

return move\_down(t->child[0], t->child[1]);

}

else

{

if (x < t->key) t->child[0] = erase(t->child[0], x);

else t->child[1] = erase(t->child[1], x);

t->size -= 1;

return balance(t);

}

}

node \*move\_down(node \*t, node \*rhs)

{

if (t == NULL) return rhs;

t->child[1] = move\_down(t->child[1], rhs);

return balance(t);

}

#define sz(t) (t ? t->size : 0)

#define ht(t) (t ? t->height : 0)

node \*rotate(node \*t, int l, int r)

{

node \*s = t->child[r];

t->child[r] = s->child[l];

s->child[l] = balance(t);

if (t) t->size = sz(t->child[0]) + sz(t->child[1]) + 1;

if (s) s->size = sz(s->child[0]) + sz(s->child[1]) + 1;

return balance(s);

}

node \*balance(node \*t)

{

for (int i = 0; i < 2; ++i)

{

if (ht(t->child[!i]) - ht(t->child[i]) < -1)

{

if (ht(t->child[i]->child[!i]) - ht(t->child[i]->child[i]) > 0)

t->child[i] = rotate(t->child[i], i, !i);

return rotate(t, !i, i);

}

}

if (t) t->height = max(ht(t->child[0]), ht(t->child[1])) + 1;

if (t) t->size = sz(t->child[0]) + sz(t->child[1]) + 1;

return t;

}

pointer rank(int k) const

{

return rank(root, k);

}

pointer rank(node \*t, int k) const

{

if (!t) return NULL;

int m = sz(t->child[0]);

if (k < m) return rank(t->child[0], k);

if (k == m) return t;

if (k > m) return rank(t->child[1], k - m - 1);

}

void clear( node \*x )

{

if( !x ) return;

if( x->child[0] )

clear( x->child[0] );

if( x->child[1] )

clear( x->child[1] );

delete x;

}

int solve( const T v )

{

node \*p = root;

int sol = 0;

while( p )

{

if (v < p->key)

p = p->child[0];

else

sol += (( !p->child[0] )?0:p->child[0]->size)+1, p = p->child[1];

}

return sol;

}

};

### Heavy-Light Descomposition

const int maxn = 100005;

vector<int>G[maxn];

int next[maxn], LCA[maxn][30], L[maxn];

typedef pair<int,int>par;

par P[maxn];

int I[maxn], top, S[maxn];

void dfs\_sin\_rec( int nodo, int pad )

{

P[++top] = par(nodo,pad);//meto en la pila el nodo

I[top] = 0;//inicializo el inicio del for de los adyacentes

while( top )

{

nodo = P[top].first, pad = P[top].second;//saco de la pila

int tam = G[nodo].size();

for( int i = I[top]; i < tam; i++ )//recorro los adyacentes de donde me quede

{

int ady = G[nodo][i];

I[top] = i+1;//actualizo donde me quede en ese nodo

if( ady == pad )

continue;

P[++top] = par(ady,nodo); //meto en la pila(como llamar recursivamente)

break;

}

if( I[top] == tam )//verifico si ya revise todos los adyacentes para bajar la recursividad

{

nodo = P[top].first, pad = P[top].second;

I[top] = 0;

top--;//bajo la recursividad

int tam = G[nodo].size();

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

{

int ady = G[nodo][i];

if( ady == pad )

continue;

if( S[ady] + 1 > S[nodo] )//actualizo el siguente nodo de la cadena

S[nodo] = S[ady]+1, next[nodo] = ady;

}

}

}

}

int dfs( int nodo, int pad )

{

int tam = G[nodo].size();

int msize = 0;

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

{

int ady = G[nodo][i];

if( ady == pad )

continue;

int sz = dfs(ady,nodo);

if( sz > msize )

msize = sz, next[nodo] = ady;

}

return msize+1;

}

int cad[maxn], ncad, len[maxn], pos[maxn], link[maxn];

struct st

{

int on, off, b;

st( int a = 0, int d = 0, int c = 0 )

{

on = a, off = d, b = c;

}

};

vector< vector<st> >T;

int mark[maxn];

void hld\_sin\_rec( int nodo, int pad )

{

top = 0;

P[++top] = par(nodo,pad);

I[top] = 0;

while( top )

{

nodo = P[top].first, pad = P[top].second;

if( !mark[nodo] )//para no analizarlo 2 veces

{

cad[nodo] = ncad;

pos[nodo] = ++len[ncad];

}

mark[nodo] ++;

if( next[nodo] && mark[nodo] == 1 )//si tiene un siguiente nodo y es la primera vez que lo visito

{

P[++top] = par(next[nodo],nodo);//me muevo al siguiente nodo

continue;

}

else if( !next[nodo] ) //si es el ultimo de la cadeno creo el vector del segment tree

T.push\_back( vector<st>(4\*(len[ncad]+1)) );

int tam = G[nodo].size();

for( int i = I[top]; i < tam; i++ )//recorro los adyacentes

{

int ady = G[nodo][i];

I[top] = i+1;

if( ady == pad || ady == next[nodo] )

continue;

ncad++;

link[ncad] = nodo;

P[++top] = par(ady,nodo);

break;

}

if( I[top] == tam )//bajo la recursividad

{

I[top] = 0;

top--;

}

}

}

void hld( int nodo, int pad )

{

int tam = G[nodo].size();

cad[nodo] = ncad;

pos[nodo] = ++len[ncad];

if( next[nodo] )

{

hld(next[nodo],nodo);

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

{

int ady = G[nodo][i];

if( ady == pad || ady == next[nodo] )

continue;

ncad++;

link[ncad] = nodo;

hld(ady,nodo);

}

}

else

T.push\_back( vector<st>( 4\*len[ncad],0 ));

}

queue<int>Q;

int lg;

void processLCA( int nodo )

{

Q.push(nodo);

L[nodo] = 1;

while( !Q.empty() )

{

nodo = Q.front();

Q.pop();

int tam = G[nodo].size();

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

{

int ady = G[nodo][i];

if( L[ady] )

continue;

L[ady] = L[nodo]+1;

Q.push(ady);

LCA[ady][0] = nodo;

lg = log2(L[ady]);

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

LCA[ady][j] = LCA[LCA[ady][j-1]][j-1];

}

}

}

int QLCA( int a, int b )

{

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

swap(a,b);

lg = log2(L[a]);

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

if( L[a] - (1<<i) >= L[b] )

a = LCA[a][i];

if( a == b )

return a;

lg = log2(L[a]);

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

if( LCA[a][i] != LCA[b][i] && LCA[a][i] )

a = LCA[a][i], b = LCA[b][i];

return LCA[a][0];

}

int ids;

void build( int ini, int fin, int lv )

{

T[ids][lv].on = fin-ini+1;

T[ids][lv].off = T[ids][lv].b = 0;

if( ini == fin )

return;

int mit = (ini+fin)/2;

build(ini,mit,lv\*2);

build(mit+1,fin,lv\*2+1);

}

void lazy( int ini, int fin, int lv )

{

T[ids][lv].b = 0;

if( ini == fin )

return;

T[ids][lv\*2].b ^= 1;

T[ids][lv\*2+1].b ^= 1;

swap(T[ids][lv\*2].on,T[ids][lv\*2].off);

swap(T[ids][lv\*2+1].on,T[ids][lv\*2+1].off);

}

void update( int ini, int fin, int lv, int a, int b )

{

if( a > b )

return;

if( ini > b || fin < a )

return;

if( T[ids][lv].b )

lazy(ini,fin,lv);

if( ini >= a && fin <= b )

{

T[ids][lv].b = 1;

swap(T[ids][lv].on,T[ids][lv].off);

return;

}

int mit = (ini+fin)/2;

update(ini,mit,lv\*2,a,b);

update(mit+1,fin,lv\*2+1,a,b);

T[ids][lv].on = T[ids][lv\*2].on+T[ids][lv\*2+1].on;

T[ids][lv].off = T[ids][lv\*2].off+T[ids][lv\*2+1].off;

}

void UP( int nodo, int lca, int b )

{

while( cad[nodo] != cad[lca] )

{

ids = cad[nodo];

update(1,len[ids],1,1,pos[nodo]);

nodo = link[ids];

}

ids = cad[nodo];

update(1,len[ids],1,pos[lca]+b,pos[nodo]);

}

int query( int ini, int fin, int lv, int a, int b )

{

if( a > b )

return 0;

if( ini > b || fin < a )

return 0;

if( T[ids][lv].b )

lazy(ini,fin,lv);

if( ini >= a && fin <= b )

return T[ids][lv].on;

int mit = (ini+fin)/2;

return query(ini,mit,lv\*2,a,b)+query(mit+1,fin,lv\*2+1,a,b);

}

int QU( int nodo, int lca, int b )

{

int ret = 0;

while( cad[nodo] != cad[lca] )

{

ids = cad[nodo];

ret += query(1,len[ids],1,1,pos[nodo]);

nodo = link[ids];

}

ids = cad[nodo];

ret += query(1,len[ids],1,pos[lca]+b,pos[nodo]);

return ret;

}

int main()

{

int N;

cin >> N;

int a, b;

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

{

cin >> a >> b;

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

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

}

//dfs(1,-1);

dfs\_sin\_rec(1,-1);

hld\_sin\_rec(1,-1);

//hld(1,-1);

processLCA(1);

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

{

ids = i;

build(1,len[ids],1);

}

int c, Q;

cin >> Q;

while( Q-- )

{

cin >> c >> a >> b;

int lca = QLCA(a,b);

if( c == 1 )

{

UP(a,lca,0);

UP(b,lca,1);

}

else

cout << QU(a,lca,0)+QU(b,lca,1) << '\n';

}

return 0;

}

### KD-Tree

typedef long long ll;

int N, Q;

struct point

{

int x, y;

} P[maxn];

bool cmpx ( const point &a, const point &b )

{

return a.x < b.x;

}

bool cmpy ( const point &a, const point &b )

{

return a.y < b.y;

}

inline ll dist ( point a, point b )

{

return 1ll\*(a.x-b.x)\*(a.x-b.x) + 1ll\*(a.y-b.y)\*(a.y-b.y);

}

struct kd

{

kd \*h1, \*h2;

point p;

}\*KD;

void init ( int ini, int fin, kd \*nod, int split )

{

if ( !split )

sort ( P+ini, P+1+fin, cmpx );

else

sort ( P+ini, P+1+fin, cmpy );

int piv = ( ini+fin )>> 1;

nod->p = P[piv];

if ( ini < piv )

{

nod->h1 = new kd();

init ( ini, piv-1, nod->h1, split^1 );

}

if ( piv+1 <= fin )

{

nod->h2 = new kd();

init ( piv+1, fin, nod->h2, split^1 );

}

}

ll best;

void query ( kd \*nod, point p, int split )

{

best = min ( best, dist ( p, nod->p ) );

ll tmp = ( !split )? p.x - nod->p.x : p.y - nod->p.y;

if ( tmp < 0 )

{

if ( nod->h1 )

query ( nod->h1, p, split^1 );

if ( nod->h2 && tmp\*tmp < best )

query ( nod->h2, p, split^1 );

}

else

{

if ( nod->h2 )

query ( nod->h2, p, split^1 );

if ( nod->h1 && tmp\*tmp < best )

query ( nod->h1, p, split^1 );

}

}

int main()

{

scanf ("%d%d", &N, &Q );

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

scanf ("%d%d", &P[i].x, &P[i].y );

KD = new kd();

init ( 1, N, KD, 0 );

int x, y;

while ( Q-- )

{

scanf ("%d%d", &x, &y );

best = 1ll << 50;

query ( KD, point{x,y}, 0 );

printf ("%.2f\n", 1.0\*sqrt(best) );

}

return 0;

}

### SQRT Descomposition

int sq;

int N, Q;

struct dos

{

long long val;

int id;

dos( long long a = 0, int b = 0)

{

val = a, id = b;

}

bool operator<( const dos &x )const

{

return val < x.val;

}

} P[1003][1003];

long long A[1000006];

int tam[1003];

void update( int a, long long x )

{

int ba = a/sq;

for( int i = 0; i < tam[ba]; i++ )

if( P[ba][i].id == a )

{

P[ba][i].val = x;

break;

}

sort(P[ba],P[ba]+tam[ba]);

}

int query( int a, int b, int x )

{

int ba = a/sq;

int bb = b/sq;

int sol = 0;

if( ba == bb )

{

for( int i = 0; i < tam[ba]; i++ )

if( P[ba][i].id >= a && P[ba][i].id <= b && P[ba][i].val <= x )

sol ++;

return sol;

}

for( int i = 0; i < tam[ba]; i++ )

if( P[ba][i].id >= a && P[ba][i].val <= x )

sol++;

for( int i = 0; i < tam[bb]; i++ )

if( P[bb][i].id <= b && P[bb][i].val <= x )

sol ++;

for( int i = ba+1; i < bb; i++ )

sol += upper\_bound(P[i],P[i]+tam[i],dos(x,1e7))-P[i];

return sol;

}

int main()

{

cin.tie(0);

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

cin >> N >> Q;

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

cin >> A[i];

sq = sqrt(N);

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

P[i/sq][i%sq] = dos(A[i],i), tam[i/sq]++;

for( int i = 0; i <= (N-1)/sq; i++ )

sort(P[i],P[i]+tam[i]);

char c;

int a, b;

long long x;

while( Q-- )

{

cin >> c;

if( c == 'C' )

{

cin >> a >> x;

update(a-1,x);

}

else

{

cin >> a >> b >> x;

cout << query(a-1,b-1,x) << '\n';

}

}

return 0;

}

### Order Statistic Set

#include <bits/stdc++.h>

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

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

#define maxn 100005

using namespace std;

using namespace \_\_gnu\_pbds;

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

Tr T;

/\*\*

- Funciona Como un set no como multiset, para repetir elementos, llevar un par en vez de un int donde el .second sea el

id y para usar las funciones utilizar ej order\_of\_key(par(x,1<<30));

- Las funciones:

insert(key), find(key), erase(key), erase(\*it)

find\_by\_order(k)->deveulve un puntero con el k-esimo menor elemento comenzando desde 0

order\_of\_key(key)->devulve la cantidad de elementos que son estrictamente menores que key

\*\*/

### Persistent Segment Tree

#define MAXN 500005

int sum[3000005], L[3000005], R[3000005];

int root[MAXN];

int A[MAXN],aux[MAXN];

int sz = 1;

int newnode(int s = 0)

{

sum[sz] = s;

return sz++;

}

int build(int I, int F)

{

if(I == F)

return newnode();

int piv=(I+F)/2;

int nod = newnode();

L[nod] = build(I, piv);

R[nod] = build(piv+1, F);

return nod;

}

int update(int nod, int I, int F, int pos)

{

if(I==F)

return newnode(sum[nod]+1);

int piv=(I+F)/2;

int nnod = newnode();

if(pos<=piv)

{

L[nnod] = update(L[nod],I,piv,pos);

R[nnod] = R[nod];

}

else

{

R[nnod] = update(R[nod],piv+1,F, pos);

L[nnod] = L[nod];

}

sum[nnod] = sum[L[nnod]] + sum[R[nnod]];

return nnod;

}

int query(int nod1,int nod2,int I,int F,int k)

{

if(I==F)

return I;

int suma = sum[L[nod2]] - sum[L[nod1]];

int piv=(I+F)/2;

if(suma >= k)

return query(L[nod1], L[nod2],I,piv,k);

else

return query(R[nod1], R[nod2],piv+1,F, k-suma);

}

int main()

{

int N, M;

cin >> N >> M;

root[0]=build(1, N);

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

{

cin >> A[i];

aux[i]=A[i];

}

sort(aux, aux+N);

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

A[i]=lower\_bound(aux,aux+N,A[i])-aux;

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

root[i+1] = update(root[i],1,N, A[i]+1);

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

{

int a,b,k;

cin >> a >> b >> k;

cout << aux[query(root[a-1], root[b], 1, N, k)-1] << '\n';

}

return 0;

}

### RMQ

int A[10005],M[10005][20];

int main()

{

int N;

scanf("%d",&N);

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

scanf("%d",&A[i]),M[i][0]=i;

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

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

if(A[M[j][i-1]]<A[M[j+(1<<(i-1))][i-1]])

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

else

M[j][i]=M[j+(1<<(i-1))][i-1];

int Q;

scanf("%d",&Q);

int a,b;

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

{

scanf("%d%d",&a,&b);a--;b--;

if(a>b)swap(a,b);

int lg=(int)log2(b-a+1);

int sol=min(A[M[a][lg]],A[M[b-(1<<lg)+1][lg]]);

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

}

return 0;

}

### Splay Tree

#define maxn 100005

using namespace std;

char cad[maxn];

int l, Q;

struct splay\_tree{

const int inf = 1e9;

struct nodo {

int size;

nodo \*l, \*r, \*p;

bool inv;

int laz, let, cant[30];

nodo(nodo \*f=0, nodo \*i = 0, nodo \*d = 0){

l = i, p = f, r = d, size = 1, let = 0, laz = 0, inv = false;

for( int i = 0; i <= 29; i++ )

cant[i] = 0;

}

} \*root;

splay\_tree () { root = NULL; }

inline void zig ( nodo \*x ) {

nodo \*y = x->p, \*z = y->p;

y->l = x->r;

if ( x->r )

x->r->p = y;

x->p = z;

if ( z )

if ( z->l == y ) z->l = x;

else z->r = x;

y->p = x;

x->r = y;

updata (y);

}

inline void zag ( nodo \*x ) {

nodo \*y = x->p, \*z = y->p;

y->r = x->l;

if ( x->l )

x->l->p = y;

x->p = z;

if ( z )

if ( z->l == y ) z->l = x;

else z->r = x;

y->p = x;

x->l = y;

updata (y);

}

inline void splay ( nodo \*x ) {

nodo \*y, \*z;

while ( x->p ) {

y = x->p, z = y->p;

if ( !z ) {

if ( y->l == x ) zig(x);

else zag(x);

}

else if ( z->l == y && y->l == x ) zig(y), zig(x);

else if ( z->r == y && y->r == x ) zag(y), zag(x);

else if ( z->r == y && y->l == x ) zig(x), zag(x);

else zag(x), zig(x);

}

root = x;

updata (root);

}

inline void updata ( nodo \*x ) {

x->size = ( (x->l)? x->l->size : 0 ) + ( (x->r)? x->r->size : 0 ) + 1;

for( int i = 0; i <= 27; i++ )

x->cant[i] = ((x->l)?x->l->cant[i]:0) + ((x->r)?x->r->cant[i]:0) + (i == x->let);

}

inline void find ( int x ) {

if ( !root )

return;

nodo \*p = root;

int izq;

while ( true ) {

lazy (p);

izq = ( p->l )? p->l->size : 0;

if ( x == izq+1 ) break;

if ( x > izq+1 ) {

x -= izq+1;

if ( p->r ) p = p->r; else break;

}

else

if ( p->l ) p = p->l; else break;

}

splay(p);

}

inline void insert ( int a ) {

nodo \*p = root, \*f = 0;

while ( p ) { f = p, p = p->r; };

p = new nodo ( f, 0, 0 );

p->let = a;

if ( f ) f->r = p;

splay (p);

}

inline splay\_tree split ( int x ) {

if ( !root ) return splay\_tree ();

splay\_tree L = splay\_tree ();

find(x);

if ( root->l )

root->l->p = 0;

L.root = root->l,

root->l = 0;

updata ( root );

return L;

}

inline void join ( splay\_tree L ) {

if ( !L.root ) return;

if ( !root ) root = L.root;

else {

find(-inf);

root->l = L.root;

root->l->p = root;

updata(root);

}

L.root = NULL;

}

inline void erase ( int x ) {

find(x);

if ( !root ) return;

if ( !root->l ) {

nodo \*tmp = root;

root = root->r;

if ( root ) root->p = 0;

delete tmp;

}

else {

nodo \*t = root->r, \*tmp = root;

root = root->l;

root->p = 0;

find ( x );

root->r = t;

if ( t ) t->p = root;

updata ( root );

delete ( tmp );

}

}

void lazy ( nodo \*x ) {

if ( !x )

return;

if( x->inv ){

if( x->l ){

x->l->inv ^= 1;

swap(x->l->l,x->l->r);

}

if( x->r ){

x->r->inv ^= 1;

swap(x->r->l,x->r->r);

}

x->inv = 0;

}

if( x->laz ){

if( x->l ){

x->l->laz = 1;

x->l->let = x->let;

for( int i = 0; i <= 27; i++ )

x->l->cant[i] = 0;

x->l->cant[x->let] = x->l->size;

}

if( x->r ){

x->r->laz = 1;

x->r->let = x->let;

for( int i = 0; i <= 27; i++ )

x->r->cant[i] = 0;

x->r->cant[x->let] = x->r->size;

}

x->laz = 0;

}

}

int count( int a, int b, int c ){

splay\_tree t = split(a);

splay\_tree t1 = split(b-a+2);

int ret = t1.root->cant[c];

join(t1);

join(t);

return ret;

}

void change( int a, int b, int c ){

splay\_tree t = split(a);

splay\_tree t1 = split(b-a+2);

for( int i = 0; i <= 27; i++ )

t1.root->cant[i] = 0;

t1.root->cant[c] = b-a+1;

t1.root->laz = 1;

t1.root->let = c;

join(t1);

join(t);

}

void reverse( int a, int b ){

splay\_tree t = split(a);

splay\_tree t1 = split(b-a+2);

t1.root->inv = 1;

swap(t1.root->l,t1.root->r);

join(t1);

join(t);

}

}ST;

int main() {

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

cin.tie(0);

#ifdef acm

freopen("a.in","r",stdin);

#endif // acm

int N;

cin >> (cad+1) >> N;

l = strlen ( cad+1 );

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

ST.insert ( cad[i]-'a' );

ST.insert ( 26 );

int a, b, d;

char q, c;

while( N-- ){

cin >> q;

if( q == 'C' ){

cin >> a >> b >> c;

cout << ST.count(a,b,c-'a') << '\n';

}

if( q == 'S' ){

cin >> a >> b >> c;

ST.change(a,b,c-'a');

}

if( q == 'R' ){

cin >> a >> b;

ST.reverse(a,b);

}

}

return 0;

}

### Treap

template <class T>

struct treap{

struct nodo{

nodo \*l, \*r;

T key;

int size, prio, inv, lazy;

nodo(){

l = r = nullptr;

size = 1;

inv = lazy = 0;

prio = rand();

}

}\*root;

treap(){

root = nullptr;

}

typedef pair<nodo\*,nodo\*>nn;

//necesario para simular AVL y similar queda el upper bound y lower bound

int find( nodo \*n, T p ){

if( !n )return 1;

if( n->key < p )

return sz(n->l)+1+find(n->r,p);

return find(n->l,p);

}

//para simular un AVL e insertar en la posicion que le corresponde y no sea necesariamente al final

void insert\_1( T p ){

int pos = find(root,p);

treap a = split(pos-1);

nodo \*newn = new nodo(p);

root = merge(a.root,merge(newn,root));

}

int sz( nodo \*n ){

if( !n )return 0;

return n->size;

}

void update( nodo \*n ){

if( !n )return;

n->size = sz(n->l) + sz(n->r) + 1;

//do the merge between the subtrees like sum, min, max, hash etc

}

void insert( T x ){

nodo \*n = new nodo();

n->key = x;

root = merge(root,n);

}

void merge( treap a ){

root = merge(root,a.root);

}

nodo \*merge( nodo \*l, nodo \*r ){

if( !l )return r;

if( !r )return l;

if( l->prio > r->prio ){

updLazy(l);

l->r = merge(l->r,r);

update(l);

return l;

}

updLazy(r);

r->l = merge(l,r->l);

update(r);

return r;

}

treap split( int k ){

treap ret;

nn x = split(root,k);

root = x.second;

ret.root = x.first;

return ret;

}

nn split( nodo \*n, int k ){

if( !n ) return nn(n,n);

updLazy(n);

if( sz(n->l) >= k ){

nn x = split(n->l,k);

n->l = x.second;

update(n);

return nn(x.first,n);

}else{

nn x = split(n->r,k-sz(n->l)-1);

n->r = x.first;

update(n);

return nn(n,x.second);

}

}

void print( ){

print(root);

}

void updateInv( nodo \*n ){//update the reverse lazy

if( !n )return;

n->inv ^= 1;

swap( n->l,n->r );

}

void updLazy( nodo \*n ){//update all the lazy propagation

if( !n )return;

if( n->lazy ){

//do a lazy propagation like sum, min, max etc

n->lazy = 0;

}

if( n->inv ){//do lazy of reverse

updateInv(n->l);

updateInv(n->r);

n->inv = 0;

}

}

void print( nodo \*n ){

if( !n )return;

updLazy(n);

print(n->l);

cout << n->key << " ";

print(n->r);

}

void reverse( int l, int r ){

treap a = split(l-1);

treap b = split(r-l+1);

b.root->inv = 1;

swap( b.root->l, b.root->r );

root = merge(b.root,root);

root = merge(a.root,root);

}

};

//srand(time(0));

### Treap Persistente

#define le first

#define ri second

const int maxn = 100005;

struct nodo{

int val, sz;

nodo \*l, \*r;

nodo(){

val = 0;

sz = 1;

l = r = nullptr;

}

}\*root;//lista de versiones del treap; Importante recalcar que en una implementacion de una estructura persistente

//nunca se modifica una version, sino se crea otra con la modificacion

int sz( nodo \*x ){

if( !x )return 0;

return x->sz;

}

void updsz( nodo \*x ){

if( !x )return;

x->sz = sz(x->l) + sz(x->r) + 1;

}

int A[maxn], T, I1[maxn], F1[maxn], I2[maxn], F2[maxn];

nodo \*trash[3000006];

typedef pair<nodo\*,nodo\*>par;

nodo \*merge( nodo \*l, nodo \*r ){//mezcla dos treaps en uno

if( !l )return r;

if( !r )return l;

int tl = sz(l), tr = sz(r);

nodo \*ret = new nodo();//nodo copia que va a contener el treap resultante del merge

trash[++T] = ret;

if( rand()%(tl+tr) < tl ){//(talla de probabilidades que no entiendo),

ret->val = l->val;//si el nodo de la izquierda tiene mas prioridad, entonces es la raiz y su hijo izquierdo

ret->l = l->l; //se convierte en el hijo izquierdo de la nueva version

ret->r = merge(l->r,r);//en cambio el derecho es la mezcla del drecho del root y el arbol r

}else{

ret->val = r->val;//aqui lo mismo pero al reves

ret->r = r->r;

ret->l = merge(l,r->l);

}

updsz(ret);

return ret;

}

par split( nodo \*root, int k ){//retorna a 2 treaps, el primero contiene los k primeros elementos y el segundo los demas

if( !root ) return par(root,root);

nodo \*ret = new nodo();

trash[++T] = ret;

ret->val = root->val;//copio el valor de root

if( sz(root->l)+1 <= k ){//si la cant de elementos en left+1 <= k es decir que este pedazo pertenece a la solucion

par x = split(root->r,k-sz(root->l)-1);//calculamos el otro pedazo

ret->l = root->l;//entonces el nodo actual va a tener left = al left de root y rigth = a left del split

ret->r = x.le;

updsz(ret);

return par(ret,x.ri);//retorno los elementos que sirven y los que no

}else{//en caso que este nodo no sirva su copia tampoco

par x = split(root->l,k);//calculo el pedazo que me sirve de la izquierda

ret->r = root->r;//el nodo actual no me sirve y lo igualo al derecho del root que no me sirve y

ret->l = x.ri;//al que no me sirve del que me retorne.

updsz(ret);

return par(x.le,ret);//retono lo que me sirve y no me sirve

}

}

int N, M;

void crear( ){

root = nullptr;

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

nodo \*n = new nodo();

trash[++T] = n;

n->val = A[i];

n->sz = 1;

root = merge(root,n);

}

}

int C;

void recover( nodo \*cur ){

if( !cur )return;

recover(cur->l);

A[++C] = cur->val;

recover(cur->r);

}

void print(){

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

cout << A[i];

if( i != N )

cout << ' ';

}

}

int main()

{

cin.tie(0);

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

#ifdef home

freopen("a.in","r",stdin);

#endif // acm

srand(0);

cin >> N >> M;

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

A[i] = i;

}

crear();

int k, a, b;

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

cin >> k >> a >> b;

I2[i] = a, F2[i] = k;

I1[i] = b, F1[i] = k;

}

nodo \*p1, \*p2, \*p3;

for( int i = M; i >= 1; i-- ){

par x = split(root,I1[i]-1);

p1 = x.le, p2 = x.ri;

x = split(p2,F1[i]);

p3 = x.le;//1 pedazo valido;

x = split(root,I2[i]-1);

p1 = x.le, p2 = x.ri;

x = split(p2,F2[i]);

p2 = x.ri;

root = merge(p1,p3);

root = merge(root,p2);

if( T > 2500000 ){

C = 0;

recover(root);

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

trash[j]->l = trash[j]->r = nullptr;

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

delete trash[j];

T = 0;

crear();

}

}

C = 0;

recover(root);

print();

return 0;

}

# Dynamic Programming

### Convex Hull Trick

typedef long long ll;

struct line{

ll m, n;

inline ll y( ll x ){

return m\*x+n;

}

};

struct cht{

vector<line>hull;

int pt;

cht(){

pt = 0;

}

inline bool is\_bad( line a, line b, line c ){

return (b.n-a.n)\*(a.m-c.m) >= (c.n-a.n)\*(a.m-b.m);

}

inline void update( line a ){

int sz = hull.size();

while( sz >= 2 && is\_bad(hull[sz-2],hull[sz-1],a) )

hull.pop\_back(), sz--;

hull.push\_back(a);

}

inline ll query( ll x ){

if( pt > (int)hull.size()-1 )

pt = hull.size()-1;

while( pt < (int)hull.size()-1 && hull[pt].y(x) >= hull[pt+1].y(x) )

pt++;

return hull[pt].y(x);

}

};

### Divide and Conquer

dp[i][k] = min(dp[j][k-1] + C[j+1][i])

j<i

// si se cumple C[i][j] <= C[i][j+1]

void get(int k, int l, int r, int optL, int optR) {

if(l > r)

return;

int i = (l + r) >> 1;

int opt = -1;

for(int j = max(optL, 0); j <= optR && j < i; j++) {

int ct = dp[j][k-1] + C[j + 1][i];

if(ct < dp[i][k]) {

dp[i][k] = ct;

opt = j;

}

}

get(k, l, i - 1, optL, opt);

get(k, i + 1, r, opt, optR);

}

for( int k = 1; k <= K; k++ )

get(k,1,N,0,N);

sol = dp[N][K];

### From Divide and Conquer to Knuth

/\*\*

dp[i][k] = min( dp[j][k-1] + C[j+1][i] ) divide and conquer

j < i

dp[k][i] = min( dp[k-1][j-1] + C[j][i] )

k <= j <= i

\*/

typedef long long ll;

const int maxn = 1003;

int N, K;

ll L[maxn], X[maxn], W[maxn], dp[maxn][maxn], O[maxn][maxn];

inline ll costo( int l, int r ){

ll ret = L[r] - (L[l] + (X[r]-X[l])\*W[l-1]);

return ret;

}

int main()

{

cin.tie(0);

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

#ifdef acm

freopen("a.in","r",stdin);

#endif // acm

while ( cin >> N >> K ){

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

cin >> X[i] >> W[i], W[i] += W[i-1];

L[1] = 0;

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

L[i] = L[i-1] + (X[i]-X[i-1])\*W[i-1];

}

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

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

dp[i][j] = 1LL << 60, O[i][j] = 0;

dp[0][0] = 0;

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

dp[1][i] = costo(1,i), O[1][i] = 1;

int opt = 0;

for( int k = 2; k <= K; k++ ){

for( int i = N; i >= k; i-- ){

opt = (i == N)?N:O[k][i+1];

for( int j = max((int)O[k-1][i],k); j <= opt; j++ )

if( dp[k][i] > dp[k-1][j-1] + costo(j,i) )

dp[k][i] = dp[k-1][j-1] + costo(j,i), O[k][i] = j;

}

}

cout << dp[K][N] << '\n';

}

return 0;

}

///vers menos memoria

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

const int maxn = 1003;

int N, K;

ll L[maxn], X[maxn], W[maxn], dp[2][maxn], O[2][maxn];

int now;

inline ll costo( int l, int r ){

ll ret = L[r] - (L[l] + (X[r]-X[l])\*W[l-1]);

return ret;

}

int main()

{

cin.tie(0);

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

#ifdef acm

freopen("a.in","r",stdin);

#endif // acm

while ( cin >> N >> K ){

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

cin >> X[i] >> W[i], W[i] += W[i-1];

L[1] = 0;

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

L[i] = L[i-1] + (X[i]-X[i-1])\*W[i-1];

}

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

dp[1][i] = costo(1,i), O[1][i] = 1;

int opt = 0;

now = 1;

for( int k = 2; k <= K; k++ ){

now ^= 1;

for( int i = N; i >= k; i-- ){

dp[now][i] = 1LL << 60;

opt = (i == N)?N:O[now][i+1];

for( int j = max((int)O[now^1][i],k); j <= opt; j++ )

if( dp[now][i] > dp[now^1][j-1] + costo(j,i) )

dp[now][i] = dp[now^1][j-1] + costo(j,i), O[now][i] = j;

}

}

cout << dp[now][N] << '\n';

}

return 0;

}

### Knuth Optimization

It is applicable in the case where recurrence is in the form :

dp[i][j]=min i <k<j {dp[i][k]+dp[k][j]}+C[i][j]

The sufficient condition for applicability is:

A[i,j-1] = A[i, j] = A[i + 1, j]

Where,

A[i][j] — the smallest k that gives optimal answer, for example in:

dp[i][j] = dp[i - 1][k] + C[k][j]

C[i][j] — given cost function

for (int s = 0; s<=k; s++) //s - length(size) of substring

for (int l = 0; l+s<=k; l++) { //l - left point

int r = l + s; //r - right point

if (s < 2) {

res[l][r] = 0; //DP base - nothing to break

mid[l][r] = l; //mid is equal to left border

continue;

}

int mleft = mid[l][r-1]; //Knuth's trick: getting bounds on m

int mright = mid[l+1][r];

res[l][r] = 1000000000000000000LL;

for (int m = mleft; m<=mright; m++) { //iterating for m in the bounds only

int64 tres = res[l][m] + res[m][r] + (x[r]-x[l]);

if (res[l][r] > tres) { //relax current solution

res[l][r] = tres;

mid[l][r] = m;

}

}

}

int64 answer = res[0][k];

# SQRT

### Mo’ s Algorithm

#define N 311111

#define A 1111111

#define BLOCK 555 // ~sqrt(N)

int cnt[A], a[N], ans[N], answer = 0;

struct node

{

int L, R, i;

} q[N];

bool cmp(node x, node y)

{

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

{

// different blocks, so sort by block.

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

}

// same block, so sort by R value

return x.R < y.R;

}

void add(int position)

{

cnt[a[position]]++;

if(cnt[a[position]] == 1)

{

answer++;

}

}

void remove(int position)

{

cnt[a[position]]--;

if(cnt[a[position]] == 0)

{

answer--;

}

}

int main()

{

int n;

scanf("%d", &n);

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

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

int m;

scanf("%d", &m);

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

{

scanf("%d%d", &q[i].L, &q[i].R);

q[i].L--;

q[i].R--;

q[i].i = i;

}

sort(q, q + m, cmp);

int currentL = 0, currentR = 0;

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

{

int L = q[i].L, R = q[i].R;

while(currentL < L)

{

remove(currentL);

currentL++;

}

while(currentL > L)

{

add(currentL-1);

currentL--;

}

while(currentR <= R)

{

add(currentR);

currentR++;

}

while(currentR > R+1)

{

remove(currentR-1);

currentR--;

}

ans[q[i].i] = answer;

}

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

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

}

### Mo’s Algorithm on tree

using namespace std;

const int maxn=40011,maxq=100011;

int BLOCK;

struct node{

int l, r, i, sp, ans;

}Q[maxq];

int lvl[maxn], p[maxn][16], st[maxn], en[maxn], id[2\*maxn], occ[maxn], ans, w[maxn];

map<int, int> HASH;

int cnt[maxn];

vector<int> g[maxn];

int ti;

bool f(node a, node b){

if (a.l/BLOCK != b.l/BLOCK)

return a.l < b.l;

return a.r < b.r;

}

bool gg(node a, node b){

return a.i < b.i;

}

void dfs(int u, int p){

st[u] = ++ti;

id[ti] = u;

int v;

for(auto v: g[u]){

if (v == p)

continue;

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

::p[v][0] = u;

dfs(v, u);

}

en[u] = ++ti;

id[ti] = u;

}

int lca(int u, int v){

int lg, i;

for (lg = 0; (1<<lg) <= lvl[u]; lg++);

lg--;

for(i=lg; i>=0; i--)

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

u = p[u][i];

if (u == v)

return u;

for(i = lg; i >= 0; i--){

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

u = p[u][i], v = p[v][i];

}

return p[u][0];

}

void add(int node){

occ[node]++;

cnt[w[node]]++;

if (occ[node] == 2){

cnt[w[node]] -= 2;

if (cnt[w[node]] == 0)

ans--;

}

else if (cnt[w[node]] == 1) ans++;

}

void del(int node){

int wt = w[node];

occ[node]--;

if (occ[node] == 1){

cnt[wt]++;

if (cnt[wt] == 1)

ans++;

return;

}

cnt[wt]--;

if (cnt[wt] == 0) ans--;

}

int main() {

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

int n, m, i, j, u, v;

ans = ti = 0;

cin>>n>>m;

BLOCK = sqrt(n);

int no = 0;

HASH.clear();

//compress coordinates

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

cin>>w[i];

if (HASH.find(w[i]) == HASH.end())

HASH[w[i]] = ++no;

w[i] = HASH[w[i]];

}

//read de graph

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

cin>>u>>v;

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

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

}

lvl[1] = 0;

memset(cnt, 0, sizeof(cnt));

memset(occ, 0, sizeof(occ));

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

//compute the necesary values

dfs(1, 0);

//compute lca

for(i=1; i<16; i++)

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

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

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

//read th queries

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

Q[i].i = i;

Q[i].sp = -1;

cin>>u>>v;

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

swap(u, v);

int w = lca(u, v);

//case 1

if (w == v){

Q[i].l = st[v];

Q[i].r = st[u]+1;

}//case 2

else{

if (st[v] > en[u]){

Q[i].l = en[u];

Q[i].r = st[v]+1;

}

else{

Q[i].l = en[v];

Q[i].r = st[u]+1;

}

// Special case: We have to consider 'w' separately.

Q[i].sp = w;

}

}

sort(Q, Q+m, f);

int currL = 0, currR = 0, L, R;

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

L = Q[i].l, R = Q[i].r;

while (currL < L){

del(id[currL]);

currL++;

}

while (currL > L){

add(id[currL-1]);

currL--;

}

while (currR < R){

add(id[currR]);

currR++;

}

while (currR > R){

del(id[currR-1]);

currR--;

}

Q[i].ans = ans;

if (Q[i].sp != -1){

if (cnt[w[Q[i].sp]] == 0)

Q[i].ans = ans+1;

}

}

sort(Q, Q+m, gg);

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

cout<<Q[i].ans<<endl;

return 0;

}

### SQRT Descomposition

int sq;

int N, Q;

struct dos{

long long val;

int id;

dos( long long a = 0, int b = 0){

val = a, id = b;

}

bool operator<( const dos &x )const

{

return val < x.val;

}

}P[1003][1003];

long long A[1000006];

int tam[1003];

void update( int a, long long x )

{

int ba = a/sq;

for( int i = 0; i < tam[ba]; i++ )

if( P[ba][i].id == a )

{

P[ba][i].val = x;

break;

}

sort(P[ba],P[ba]+tam[ba]);

}

int query( int a, int b, int x )

{

int ba = a/sq;

int bb = b/sq;

int sol = 0;

if( ba == bb )

{

for( int i = 0; i < tam[ba]; i++ )

if( P[ba][i].id >= a && P[ba][i].id <= b && P[ba][i].val <= x )

sol ++;

return sol;

}

for( int i = 0; i < tam[ba]; i++ )

if( P[ba][i].id >= a && P[ba][i].val <= x )

sol++;

for( int i = 0; i < tam[bb]; i++ )

if( P[bb][i].id <= b && P[bb][i].val <= x )

sol ++;

for( int i = ba+1; i < bb; i++ )

sol += upper\_bound(P[i],P[i]+tam[i],dos(x,1e7))-P[i];

return sol;

}

int main()

{

cin.tie(0);

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

cin >> N >> Q;

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

cin >> A[i];

sq = sqrt(N);

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

P[i/sq][i%sq] = dos(A[i],i), tam[i/sq]++;

for( int i = 0; i <= (N-1)/sq; i++ )

sort(P[i],P[i]+tam[i]);

char c;

int a, b;

long long x;

while( Q-- )

{

cin >> c;

if( c == 'C' )

{

cin >> a >> x;

update(a-1,x);

}

else

{

cin >> a >> b >> x;

cout << query(a-1,b-1,x) << '\n';

}

}

return 0;

}