Contents

Template

```cpp
#include <bits/stdc++.h>

#define fore(i, a, b) for(int i = a; i < b; i++)
#define forn(i, n) fore(i, 0, n)
#define ff first
#define ss second
#define mp make_pair
#define pb push_back
#define eb emplace_back
#define ford(i, n) for(int i = n - 1; i >= 0; i--)

using namespace std;

typedef long long ll;

const int N = 100005;

int main(){


    return 0;
}
```

.vimrc

```
set statusline+=%F nu showcmd cindent autoindent smartindent tabstop=4 shiftwidth=4
syntax on
```

Ad hoc

Go up for ultras

```cpp
struct Segtree{
    int n;
    vector<int> st;

    void build(int p, int L, int R, const int v[]){
        if(L == R){ st[p] = v[L]; return; }
        int mid = (L + R) / 2;
        build(2*p, L, mid, v);
        build(2*p+1, mid+1, R, v);
        st[p] = max(st[2*p], st[2*p+1]);
    }
    int findL(int p, int i, int L, int R, int v){
        if(i <= L || v >= st[p]) return -1;
        if(L == R) return L;

        int mid = (L + R) / 2;
        int ret = -1;
        if(i > mid+1 && st[2*p+1] > v)
            ret = findL(2*p+1, i, mid+1, R, v);
        if(ret != -1) return ret;
        return findL(2*p, i, L, mid, v);
    }

    int findR(int p, int i, int L, int R, int v){
        if(i > R || v >= st[p]) return -1;
        if(L == R) return L;

        int mid = (L + R) / 2;
        int ret = -1;
        if(i < mid && st[2*p] > v)
            ret = findR(2*p, i, L, mid, v);
        if(ret != -1) return ret;
        return findR(2*p+1, i, mid+1, R, v);
    }

public:
    Segtree(int sz, const int v[]) : n(sz), st(4*sz, 0){
        build(1, 0, n-1, v);
    }
    int findR(int i, int v){
        return findR(1, i, 0, n-1, v);
    }
    int findL(int i, int v){
        return findL(1, i, 0, n-1, v);
    }
};
```

Small to Large

```cpp
void cnt_sz(int u, int p = -1){
    sz[u] = 1;

    for(int v : g[u]) if(v != p)
        cnt_sz(v, u), sz[u] += sz[v];
}


void add(int u, int p, int big = -1){
    // Update info about this vx in global answer

    for(int v : g[u]) if(v != p && v != big)
        add(v, u);
}


void dfs(int u, int p, int keep){
```

```
    int big = -1, mmx = -1;

    for(int v : g[u]) if(v != p && sz[v] > mmx)
        mmx = sz[v], big = v;

    for(int v : g[u]) if(v != p && v != big)
        dfs(v, u, 0);

    if(big != -1) dfs(big, u, 1);

    add(u, p, big);

    for(auto x : q[u]){
        // answer all queries for this vx
    }

    if(!keep){
        // Remove data from this subtree
    }
}
```

Estrutura de Dados

Segtree

```
struct Segtree{
    int n;
    vector<ll> st, lazy;

    void prop(int p, int L, int R){
        if(lazy[p]){
            st[p] += lazy[p] * (R - L + 1);
            if(L != R) lazy[2*p] += lazy[p], lazy[2*p+1] += lazy[p];
            lazy[p] = 0;
        }
    }

    void update(int p, int L, int R, int i, int j, ll v){
        prop(p, L, R);
        if(L > j || R < i) return;
        if(L >= i && R <= j){
            lazy[p] = v;
            prop(p, L, R);
            return;
        }
        int mid = (L+R)/2;
        update(2*p, L, mid, i, j, v);
        update(2*p+1, mid+1, R, i, j, v);
```

```
        st[p] = st[2*p] + st[2*p+1];
    }
    ll query(int p, int L, int R, int i, int j){
        prop(p, L, R);
        if(L > j || R < i) return 0;
        if(L >= i && R <= j) return st[p];
        int mid = (L+R)/2;
        return query(2*p, L, mid, i, j) + query(2*p+1, mid+1, R, i, j);

    }
public:
    Segtree(int sz = 0) : n(sz), st(4*sz, 0), lazy(4*sz, 0){}

    // sum v to every element in range [i, j]
    void update(int i, int j, ll v){
        update(1, 1, n, i, j, v);
    }
    ll query(int i, int j){
        return query(1, 1, n, i, j);
    }
};
```

Contra Ataque Ramsay

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <list>
#include <vector>

#define LADO 800
#define SEG_LADO 1400000

using namespace std;

class Sgtree{
    private:
        int arv[SEG_LADO];
    public:
        int query(int L, int R, int D, int U, int l, int r, int d, int u, int p);
        void update(int L, int R, int D, int U, int pos_x, int pos_y, int num, int p);
        Sgtree(){memset(arv, 0, SEG_LADO<<2);};
}tipo_segtree;


typedef struct _sold{
    int x,y;
    bool pai;
}tipo_soldado;


int Sgtree::query(int L, int R, int D, int U, int l, int r, int d, int u, int p) {
```

```cpp
    if(r < L || l > R || d > U || u < D)
        return 0;
    if(L >= l && R <= r && U <= u && D >= d){
        return arv[p];
    }
    int tmp1, tmp2, tmp3, tmp4;
    tmp1 = query(L, (L+R)>>1, D, (U+D)>>1, l, r, d, u, (p<<2)+1);
    tmp2 = query(L, (L+R)>>1, ((U+D)>>1) + 1, U, l, r, d, u, (p<<2)+2);
    tmp3 = query(((L+R)>>1) + 1, R, D, (U+D)>>1, l, r, d, u, (p<<2)+3);
    tmp4 = query(((L+R)>>1) + 1, R, ((U+D)>>1) + 1, U, l, r, d, u, (p<<2)+4);

    tmp1 = max(tmp1, tmp2);
    tmp1 = max(tmp1, tmp3);
    return max(tmp1, tmp4);
}

void Sgtree::update(int L, int R, int D, int U, int pos_x, int pos_y, int num, int p){{}
    if(L == R && U == D){
        arv[p] = num;
    }
    else{
        if(pos_x <= (L+R)>>1){
            if(pos_y <= (U+D)>>1)
                update(L, (L+R)>>1, D, (U+D)>>1, pos_x, pos_y, num, (p<<2)+1);
            else
                update(L, (L+R)>>1, ((U+D)>>1) + 1, U, pos_x, pos_y, num, (p<<2)+2);
        }
        else{
            if(pos_y <= (U+D)>>1)
                update(((L+R)>>1) + 1, R, D, (U+D)>>1, pos_x, pos_y, num, (p<<2)+3);
            else
                update(((L+R)>>1) + 1, R, ((U+D)>>1) + 1, U, pos_x, pos_y, num, (p<<2)+4);
        }
        int tmp1, tmp2;
        tmp1 = max(arv[(p<<2)+1], arv[(p<<2)+2]);
        tmp2 = max(arv[(p<<2)+3], arv[(p<<2)+4]);
        arv[p] = max(tmp1, tmp2);
    }

}

vector<int> G[50001];
Sgtree *tree;
tipo_soldado soldado[50001];
bool is_root[50001];
int ans = 1;
void dfs(int no){
    int tmp = tree->query(0, LADO, 0, LADO, 0 ,soldado[no].x-1, 0, soldado[no].y-1, 0);
    tmp++;
    tree->update(0, LADO, 0, LADO, soldado[no].x, soldado[no].y, tmp, 0);

    ans = max(ans, tmp);
```

```cpp
    for(int j : G[no])
        dfs(j);
    tree->update(0, LADO, 0, LADO, soldado[no].x, soldado[no].y, 0, 0);
}

int main() {
    int N, M, i, j, no_pai, no_filho;
    tree = new Sgtree();

    scanf("%d %d", &N, &M);
    for(i = 1; i <= N; i++){
        scanf("%d", &soldado[i].x);
        scanf("%d", &soldado[i].y);
        soldado[i].x += 400;
        soldado[i].y += 400;
    }

    for(i = 1; i <= N; i++) is_root[i] = 1;
    for(i = 1; i <= M; i++){
        scanf("%d %d", &no_filho, &no_pai);
        G[no_pai].push_back(no_filho);
        is_root[no_filho] = 0;
    }

    for(i = 1; i <= N; i++){
        if(!is_root[i]) continue;
        if(!G[i].empty())
            dfs(i);
    }
    printf("%d\n", ans);
    return 0;
}
```

Sparse Table

```cpp
#include <bits/stdc++.h>

#define mp make_pair
#define ff first
#define ss second

using namespace std;

typedef long long ll;

int st1[200005][20], st2[200005][20];

void build(int n){
    int i,j;
```

```
    for(j = 1; 1<<j <= n; j++){
        for(i = 0; i +(1<<j) <= n; i++){
            st1[i][j] = max(st1[i][j-1],st1[i+(1<<(j-1))][j-1]);
            st2[i][j] = min(st2[i][j-1],st2[i+(1<<(j-1))][j-1]);
        }
    }
}


int getA(int l, int r){
    int num = r-l+1,k=0;
    while(num>=2) num/=2, k++;
    return max(st1[l][k], st1[r-(1<<k)+1][k]);
}


int getB(int l, int r){
    int num = r-l+1,k=0;
    while(num>=2) num/=2, k++;
    return min(st2[l][k], st2[r-(1<<k)+1][k]);
}


int bsearch1(int i, int n){
    int mid,l=i-1,r=n;
    while(l+1 < r){
        mid = (l+r)>>1;
        int a = getA(i, mid);
        int b = getB(i, mid);
        if(a < b) l = mid;
        else r = mid;
    }
    return r;
}


int bsearch2(int i, int n){
    int mid,l=i-1,r=n;
    while(l+1 < r){
        mid = (l+r)>>1;
        int a = getA(i, mid);
        int b = getB(i, mid);
        if(a <= b) l = mid;
        else r = mid;
    }
    return r;
}


int main(){
    int n,i,j;

    scanf("%d", &n);
    for(i = 0; i < n; i++) scanf("%d", &st1[i][0]);
    for(i = 0; i < n; i++) scanf("%d", &st2[i][0]);
    build(n);
    ll ans = 0;
```

```
    for(i = 0; i < n; i++){
        int x = bsearch1(i,n);
        int y = bsearch2(i,n);
        if(y > x)
            ans += (y-x);
    }
    printf("%lld\n", ans);

    return 0;
}
```

Merge Sort Tree

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

using namespace std;

struct MergeTree{
    int n;
    vector<vector<int> > st;

    void build(int p, int L, int R, const int v[]){
        if(L == R){
            st[p].push_back(v[L]);
            return;
        }
        int mid = (L+R)/2;
        build(2*p, L, mid, v);
        build(2*p+1, mid+1, R, v);
        st[p].resize(R-L+1);
        merge(st[2*p].begin(), st[2*p].end(),
            st[2*p+1].begin(), st[2*p+1].end(),
            st[p].begin());
    }

    int query(int p, int L, int R, int i, int j, int x) const{
        if(L > j || R < i) return 0;
        if(L >= i && R <= j){
            int id = lower_bound(st[p].begin(), st[p].end(), x) - st[p].begin();
            return int(st[p].size()) - id;
        }
        int mid = (L+R)/2;
        return query(2*p, L, mid, i, j, x) +
            query(2*p+1, mid+1, R, i, j, x);
    }

public:
MergeTree(int sz, const int v[]): n(sz), st(4*sz){
    build(1, 1, n, v);
}
```

```
        //number of elements >= x on segment [i, j]
    int query(int i, int j, int x) const{
        if(i > j) swap(i, j);
        return query(1, 1, n, i, j, x);
    }
};


int n, v[100005];

int main(){

    scanf("%d", &n);

    for(int i = 1; i <= n; i++)
        scanf("%d", v+i);

    MergeTree mst(n, v);

    return 0;
}
```

Bit bolada

```
/* bit normal -> cumulative frequency */
class ft_normal {
public:

    vector<int> ft;
    int n;

    ft_normal (int size) : n(size), ft(n, 0){}
    int query_cf(int a) const{ int resp = 0;
        for (int i = a; i; i -= (i & -i)) resp += ft[i];
        return resp; }
    int update_cf (int a, int v) {
        for (int i = a; i < n; i += (i & -i)) ft[i] += v;
        return 0; }

};


class ft_invert {
public:

    vector<int> ft_inv;
    int n;

    ft_invert (int size) : n(size), ft_inv(n, 0){}
    int query_cf_inv(int a) const{ int resp = 0;
        for (int i = a; i < n; i += (i & -i)) resp += ft_inv[i];
        return resp; }
```

```
    int update_cf_inv (int a, int v) {
        for (int i = a; i; i -= (i & -i)) ft_inv[i] += v;
        return 0; }
};

/* bit diferente -> range minimum query*/

class ft_diff {
public:
    int n;
    vector<int> ft;
    vector<int> ft_inv;
    vector<int> vet;

    ft_diff(int size) : n(size), ft(n, 0), ft_inv(n, 0), vet(n, 0){
        for (int i = 1; i < n; i++) ft[i] = ft_inv[i] = i; }
    int query_rmq (int a, int b) const{
        int i, mini = a;
        //bit normal
        for (i = a; i <= b; i += (i & -i))
            if (i + (i & -i) <= b)
                mini = vet[mini] < vet[ft_inv[i]] ? mini : ft_inv[i];
            else mini = vet[mini] < vet[i] ? mini : i;
        //bit invertida
        for (i = b; i >= a; i -= (i & -i))
            if (i - (i & -i) >= a)
                mini = vet[mini] < vet[ft[i]] ? mini : ft[i];
        return mini;
    }

    int upd (int a, int v) {
        put_value(a, v);
        update_rmq(a, v);
        return 0;
    }

    int put_value (int a, int v) {
        vet[a] = v;
        return 0;
    }
    int update_rmq (int a, int v) const{
        int mini_l = a, mini_r = a;
        for (int i = a; i < n; i += (i & -i)) {
            if (ft[i] != a) ft[i] = v < vet[ft[i]] ? a : ft[i];
            else {
                ft[i] = vet[mini_l] < vet[mini_r] ?
                    (vet[mini_l] < v ? mini_l : a) : (vet[mini_r] < v ? mini_r : a);
                if (i >= a + 1)
                    mini_r = vet[mini_r] < vet[ft_inv[i]] ? mini_r : i;
                if (i - (i & -i) + 1 <= a - 1)
                    mini_l = vet[mini_l] < vet[ft[i - (i & -i) + 1]] ?
                        mini_l : i - (i & -i) + 1;
```

```
            }
        }
        mini_l = a, mini_r = a;
        //bit invertida
        for (int i = a; i; i -= (i & -i)) {
            if (ft_inv[i] != a) ft_inv[i] = v < vet[ft_inv[i]] ? a : ft_inv[i];
            else {
                ft_inv[i] = vet[mini_l] < vet[mini_r] ?
                    (vet[mini_l] < v ? mini_l : a) : (vet[mini_r] < v ? mini_r : a);
                if (i + (i & -i) - 1 >= a + 1)
                    mini_r = vet[mini_r] < vet[ft_inv[i + (i & -i) - 1]] ?
                        mini_r : (i + (i & -i) - 1);
                if (i <= a - 1)
                    mini_l = vet[mini_l] < vet[ft[i]] ? mini_l : i;
            }
        }
        return 0;
    }

};
```

Palindromic Tree

```
/***************************************************************************

  This code counts number of palindrome substrings of the string.
  Based on problem 1750 from informatics.mccme.ru:
http://informatics.mccme.ru/moodle/mod/statements/view.php?chapterid=1750

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

#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <algorithm>
#include <vector>
#include <set>
#include <map>
#include <string>
#include <utility>
#include <cstring>
#include <cassert>
#include <cmath>
#include <stack>
#include <queue>

using namespace std;

const int MAXN = 105000;

struct node {
    int next[26];
    int len;
    int sufflink;
    int num;
};

int len;
char s[MAXN];
node tree[MAXN];
int num; // node 1 - root with len -1, node 2 - root with len 0
int suff; // max suffix palindrome
long long ans;

bool addLetter(int pos) {
    int cur = suff, curlen = 0;
    int let = s[pos] - 'a';

    while (true) {
        curlen = tree[cur].len;
        if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos])
            break;
        cur = tree[cur].sufflink;
    }
    if (tree[cur].next[let]) {
        suff = tree[cur].next[let];
        return false;
    }

    num++;
    suff = num;
    tree[num].len = tree[cur].len + 2;
    tree[cur].next[let] = num;

    if (tree[num].len == 1) {
        tree[num].sufflink = 2;
        tree[num].num = 1;
        return true;
    }

    while (true) {
        cur = tree[cur].sufflink;
        curlen = tree[cur].len;
        if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos]) {
            tree[num].sufflink = tree[cur].next[let];
            break;
        }
    }

    tree[num].num = 1 + tree[tree[num].sufflink].num;

    return true;
}
```

```
void initTree() {
    num = 2; suff = 2;
    tree[1].len = -1; tree[1].sufflink = 1;
    tree[2].len = 0; tree[2].sufflink = 1;
}

int main() {
    //assert(freopen("input.txt", "r", stdin));
    //assert(freopen("output.txt", "w", stdout));

    gets(s);
    len = strlen(s);

    initTree();

    for (int i = 0; i < len; i++) {
        addLetter(i);
        ans += tree[suff].num;
    }

    cout << ans << endl;

    return 0;
}
```

Paradigmas

FFT

```
// typedef complex<double> base;
struct base{
    double r, i;
    base(double r = 0, double i = 0) : r(r), i(i) {}
    base operator*(const base &o){
        return base(r*o.r - i*o.i, r*o.i + o.r*i);
    }

    base& operator*=(const base &o){
        double newr = r*o.r - i*o.i, newi = r*o.i + o.r*i;
        r = newr, i = newi;
        return *this;
    }

    base& operator+=(const base &o){
        r += o.r, i += o.i;
        return *this;
    }
```

```
    base& operator/=(const double &o){
        r /= o, i /= o;
        return *this;
    }

    base& operator-=(const base &o){
        r -= o.r, i -= o.i;
        return *this;
    }

    base operator+(const base &o){
        return base(r + o.r, i + o.i);
    }

    base operator-(const base &o){
        return base(r - o.r, i - o.i);
    }
};

double PI = acos(-1);

void fft(vector<base> &a, bool inv){
    int n = (int)a.size();

    for(int i = 1, j = 0; i < n; i++){
        int 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 * (inv ? -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(inv)
        for(int i = 0; i < n; i++)
            a[i] /= n;
}
void multiply(const vector<int> &a, const vector<int> &b, vector<int> &res){
    vector<base> fa(a.begin(), a.end()), fb(b.begin(), b.end());
```

```
    size_t n = 1;
    while(n < a.size()) n <<= 1;
    while(n < 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] = int(fa[i].real() + 0.5);
}
```

NTT

```
const int mod = 7340033;
const int root = 5;
const int root_1 = 4404020;
const int root_pw = 1<<20;

void fft (vector<int> & a, bool invert) {
    int n = (int) a.size();

    for (int i=1, j=0; i<n; ++i) {
        int 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) {
        int wlen = invert ? root_1 : root;
        for (int i=len; i<root_pw; i<<=1)
            wlen = int (wlen * 1ll * wlen % mod);
        for (int i=0; i<n; i+=len) {
            int w = 1;
            for (int j=0; j<len/2; ++j) {
                int u = a[i+j], v = int (a[i+j+len/2] * 1ll * w % mod);
                a[i+j] = u+v < mod ? u+v : u+v-mod;
                a[i+j+len/2] = u-v >= 0 ? u-v : u-v+mod;
                w = int (w * 1ll * wlen % mod);
            }
        }
    }
    if (invert) {
        int nrev = reverse (n, mod);
```

```
        for (int i=0; i<n; ++i)
            a[i] = int (a[i] * 1ll * nrev % mod);
    }
}
```

Matemática

Euclides Extendido

```
// a*x + b*y = gcd(a, b), <gcd, <x, y>>
typedef tuple<int, int, int> tiii;
tiii euclidesExt(int a, int b) {
    if(b == 0) return tiii(a, 1, 0);
    tiii ret = euclidesExt(b, a % b);

    int q, w, e;
    tie(q, w, e) = ret;

    get<1>(ret) = e;
    get<2>(ret) = w - e*(a / b);

    return ret;
}

// a*b = 1 (mod c) <-> a*b + c*k = 1
int invmult(int a, int b){
    return (get<1>(euclidesExt(a, b)) + b) % b;
}
```

Função totiente

```
ll totiente(ll n){
    ll ans = n;
    for(ll i = 2; i*i <= n; i++){
        if(n % i == 0){
            ans = (ans / i) * (i-1);
            while(n % i == 0) n /= i;
        }
    }

    if(n > 1) ans = (ans / n) * (n - 1);

    return ans;
}
```

Eliminação Gaussiana

```cpp
// Gaussian elimination
//
// Solves systems of linear equations.
//
// To use, build a matrix of coefficients and call run(mat, R, C).
// If the i-th variable is free, row[i] will be -1, otherwise it's value will
// be ans[i].
//
// Time complexity: O(R * C^2)
//
// Constants to configure:
// - MAXC is the number of columns
// - eps is the epsilon value

namespace Gauss {
  const int MAXC = 1001;

  int row[MAXC];
  double ans[MAXC];

  void run(double mat[][MAXC], int R, int C) {
    REP(i, C) row[i] = -1;

    int r = 0;
    REP(c, C) {
      int k = r;
      FOR(i, r, R) if (fabs(mat[i][c]) > fabs(mat[k][c])) k = i;
      if (fabs(mat[k][c]) < eps) continue;

      REP(j, C+1) swap(mat[r][j], mat[k][j]);
      REP(i, R) if (i != r) {
        double w = mat[i][c] / mat[r][c];
        REP(j, C+1) mat[i][j] -= mat[r][j] * w;
      }
      row[c] = r++;
    }

    REP(i, C) {
      int r = row[i];
      ans[i] = r == -1 ? 0 : mat[r][C] / mat[r][i];
    }
  }
}
```

Grafos

LCA

```cpp
int p[N], L[N], vis[N], dp[N][MAXL], pw[N], dp2[N][MAXL];
```

```cpp
vector<vector<ii> > g;

void dfs(int u){
    vis[u] = 1;

    for(ii &v : g[u]){
        if(!vis[v.ff]){
            p[v.ff] = u;
            L[v.ff] = L[u]+1;
            pw[v.ff] = v.ss;
            dfs(v.ff);
        }
    }
}

int f(int u, int j){
    if(j == 0) return p[u];
    if(dp[u][j] != -1) return dp[u][j];
    return dp[u][j] = f(f(u, j-1), j-1);
}

ll h(int u, int j){
    if(j == 0) return pw[u];
    if(dp2[u][j] != -1) return dp2[u][j];
    return dp2[u][j] = h(u, j-1) + h(f(u, j-1), j-1);
}

int LCA(int u, int v){
    if(L[u] < L[v]) swap(u, v);

    for(int i = MAXL-1; i >= 0; i--)
        if(L[u] - (1 << i) >= L[v])
            u = f(u, i);

    if(u == v) return v;

    for(int i = MAXL-1; i >= 0; i--)
        if(L[u] - (1 << i) && f(u, i) != f(v, i))
            u = f(u, i), v = f(v, i);

    return p[u];
}

long long DIST(int u, int v){
    if(L[u] < L[v]) swap(u, v);

    ll ans = 0;

    for(int i = MAXL-1; i >= 0; i--)
        if(L[u] - (1 << i) >= L[v])
            ans += h(u, i), u = f(u, i);
```

```cpp
    if(u == v) return ans;

    for(int i = MAXL-1; i >= 0; i--)
        if(L[u] > (1 << i) && f(u, i) != f(v, i))
            ans += h(u, i) + h(v, i), u = f(u, i), v = f(v, i);

    ans += pw[u] + pw[v];
    return ans;
}


int KTH(int a, int b, int c){
    int lca = LCA(a, b);
    int u = a, v = b;
    if(c > L[a] - L[lca] + 1){
        c -= L[a] - L[lca] + 1;
        for(int i = MAXL-1; i >= 0; i--)
            if(L[v] - (1 << i) - L[lca] >= c)
                v = f(v, i);
        return v;
    }
    else{
        for(int i = MAXL-1; i >= 0; i--)
            if(L[a] - (L[u] - (1 << i)) + 1 <= c)
                u = f(u, i);
        return u;

    }
    return 0;
}
```

Fluxo Máximo

Batalha Naval

```cpp
#include <bits/stdc++.h>

using namespace std;

int pX[200200], pY[200200], distX[200200], distY[200200];
int n, ptrx = 1, ptry = 100100;

vector<vector<int> > g(200200);

bool bfs(){
    bool found = false;
    queue<int> Q;

    for(int i = 1; i < ptrx; i++)
        if(pX[i] == -1)
            Q.push(i);
```

```cpp
    memset(distX, 0, sizeof distX);
    memset(distY, 0, sizeof distY);

    while(!Q.empty()){
        int u = Q.front(); Q.pop();
        for(int i = 0; i < (int)g[u].size(); i++){
            int v = g[u][i];
            if(distY[v] == 0){
                distY[v] = distX[u]+1;
                if(pY[v] == -1) found = true;
                else{
                    distX[ pY[v] ] = distY[v]+1;
                    Q.push(pY[v]);
                }
            }
        }
    }
    return found;
}

bool dfs(int u){
    for(int i = 0; i < (int)g[u].size(); i++){
        int v = g[u][i];
        if(distX[u]+1 == distY[v]){
            distY[v] = 0; // "apaga" o vertice para a dfs
            if(pY[v] == -1 || dfs(pY[v])){
                pX[u] = v, pY[v] = u;
                return true;
            }
        }
    }
    return false;
}

int f(){
    int ans = 0;
    memset(pX, -1, sizeof pX);
    memset(pY, -1, sizeof pY);

    while(bfs())
        for(int i = 1; i < ptrx; i++)
            if(pX[i] == -1 && dfs(i)) // eh possivel escolher um par para i
                ans++;

    return ans;
}

int main(){
    int x, y;

    map<int, int> mapx;
    map<int, int> mapy;
```

```
    while(scanf("%d", &n) == 1){
        g.assign(200200, vector<int>());

        for(int i = 0; i < n; i++){
            scanf("%d_%d", &x, &y);

            if(mapx.find(x) == mapx.end()) mapx[x] = ptrx++;
            if(mapy.find(y) == mapy.end()) mapy[y] = ptry++;

            x = mapx[x];
            y = mapy[y];

            g[x].push_back(y);
            g[y].push_back(x);
        }

        printf("%d\n", f());
    }
    return 0;
}
```

Circulation

```
// Circulation
//
// Given a directed weighted graph, computes the minimum cost to run the maximum
// amount of circulation flow through the graph.
//
// Configure: MAXV
// Configure: MAXE (at least 2 * calls_to_edge)
//
// Functions:
// - init(n) initializes the algorithm with the given number of nodes
// - edge(x, y, c, w) adds an edge x->y with capacity c and weight w
// - run() runs the algorithm and returns total cost
//
// Time complexity: No idea, but it should be fast enough to solve any problem
// where V and E are up to around 1000.
//
// Constants to configure:
// - MAXV is the maximum number of vertices
// - MAXE is the maximum number of edges (i.e. twice the calls to function edge)

namespace Circu {
  const int MAXV = 1000100;
  const int MAXE = 1000100;

  int V, E;
  int how[MAXV], good[MAXV], bio[MAXV], cookie = 1; llint dist[MAXV];
  int from[MAXE], to[MAXE]; llint cap[MAXE], cost[MAXE];
```

```
void init(int n) { V = n; E = 0; }

void edge(int x, int y, llint c, llint w) {
  from[E] = x; to[E] = y; cap[E] = c; cost[E] = +w; ++E;
  from[E] = y; to[E] = x; cap[E] = 0; cost[E] = -w; ++E;
}

void reset() {
  REP(i, V) dist[i] = 0;
  REP(i, V) how[i] = -1;
}

bool relax() {
  bool ret = false;
  REP(e, E) if (cap[e]) {
    int x = from[e];
    int y = to[e];

    if (dist[x] + cost[e] < dist[y]) {
      dist[y] = dist[x] + cost[e];
      how[y] = e;
      ret = true;
    }
  }
  return ret;
}

llint cycle(int s, bool flip = false) {
  int x = s;
  llint c = cap[how[x]];
  do {
    int e = how[x];
    c = min(c, cap[e]);
    x = from[e];
  } while (x != s);

  llint sum = 0;
  do {
    int e = how[x];
    if (flip) {
      cap[e] -= c;
      cap[e^1] += c;
    }
    sum += cost[e] * c;
    x = from[e];
  } while (x != s);
  return sum;
}

llint push(int x) {
  for (++cookie; bio[x] != cookie; x = from[how[x]]) {
```

```
      if (!good[x] || how[x] == -1 || cap[how[x]] == 0) return 0;
      bio[x] = cookie;
      good[x] = false;
    }
    return cycle(x) >= 0 ? 0 : cycle(x, true);
  }

  llint run() {
    reset();
    llint ret = 0;
    REP(step, 2*V) {
      if (step == V) reset();
      if (!relax()) continue;

      REP(i, V) good[i] = true;
      REP(i, V) if (llint w = push(i)) ret += w, step = 0;
    }
    return ret;
  }
}
```

Dinic

```
const int N = 205, int E = 30006;
vector<int> g[N];

int ne;
struct Edge{
    int from, to;
    ll cap;
} edge[E];

int lvl[N], vis[N], pass, source, target, px[N];

ll run(int s, ll minE){
    if(s == target) return minE;

    ll ans = 0;

    for(; px[s] < g[s].size(); px[s]++){
        int e = g[s][ px[s] ];
        auto &v = edge[e], &rev = edge[e^1];
        if(lvl[v.to] != lvl[s]+1 || !v.cap) continue;
        ll tmp = run(v.to, min(minE, v.cap));
        v.cap -= tmp, rev.cap += tmp;
        ans += tmp, minE -= tmp;
        if(minE == 0) break;
    }
    return ans;
}
```

```
int bfs(){

    queue<int> q;
    q.push(source);
    lvl[source] = 1;
    vis[source] = ++pass;

    while(!q.empty()){
        int u = q.front(); q.pop();
        px[u] = 0;

        for(int e : g[u]){
            auto v = edge[e];
            if(v.cap <= 0 || vis[v.to] == pass) continue;
            vis[v.to] = pass;
            lvl[v.to] = lvl[u]+1;
            q.push(v.to);
        }
    }
    return vis[target] == pass;
}

ll flow(){
    ll ans = 0;
    while(bfs()) ans += run(source, oo);
    return ans;
}

void addEdge(int u, int v, ll c){
    edge[ne] = {u, v, c};
    g[u].pb(ne++);
}
```

Min Cost Max Flow

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

#define forn(i, n) for(int i = 0; i < int(n); i++)
#define fore(i, a, b) for(int i = int(a); i <= int(b); i++)
#define ff first
#define ss second
#define mp make_pair
#define mt make_tuple
#define pb push_back
#define eb emplace_back

using namespace std;

typedef long long ll;
const ll oo = 1e18;
```

14

```
const int N = 505;
const int E = 30006;

vector<int> g[N];

int ne;

struct Edge{
    int from, to;
    ll cap, cost;
} edge[E];

int lvl[N], vis[N], pass, source, target, p[N], px[N];

ll d[N];

ll back(int s, ll minE){
    if(s == source) return minE;

    int e = p[s];

    ll f = back(edge[e].from, min(minE, edge[e].cap));
    edge[e].cap -= f;
    edge[e^1].cap += f;
    return f;
}

int dijkstra(){
    forn(i, N) d[i] = oo;

    priority_queue<pair<ll, int> > q;

    d[source] = 0;

    q.emplace(0, source);

    while(!q.empty()){
        ll dis = -q.top().ff;
        int u = q.top().ss; q.pop();

        if(dis > d[u]) continue;

        for(int e : g[u]){
            auto v = edge[e];
            if(v.cap <= 0) continue;
            if(d[u] + v.cost < d[v.to]){
                d[v.to] = d[u] + v.cost;
                p[v.to] = e;
                q.emplace(-d[v.to], v.to);
            }
        }
    }
```

```
    return d[target] != oo;
}

pair<ll, ll> mincost(){
    ll ans = 0, mf = 0;
    while(dijkstra()){
        ll f = back(target, oo);
        mf += f;
        ans += f * d[target];
    }
    return {mf, ans};
}

void addEdge(int u, int v, ll c, ll cost){
    edge[ne] = {u, v, c, cost};
    g[u].pb(ne++);
}

char s[55][55];

int main(){

    int n, m;
    for(int tc = 1; scanf("%d %d", &n, &m) == 2; tc++){
        ne = 0;

        int a, b, c;
        forn(i, m){
            scanf("%d %d %d", &a, &b, &c);
            addEdge(a, b, 0, c);
            addEdge(b, a, 0, -c);
            addEdge(b, a, 0, c);
            addEdge(a, b, 0, -c);
        }
        int d, k;

        scanf("%d %d", &d, &k);

        forn(i, ne) if(i % 2 == 0) edge[i].cap = k;

        source = 0, target = n;

        addEdge(0, 1, d, 0);
        addEdge(1, 0, 0, 0);

        pair<ll, ll> ans = mincost();
        printf("Instancia %d\n", tc);
        if(ans.ff < d) printf("impossivel\n");
        else printf("%lld\n", ans.ss);

        printf("\n");
        forn(i, N) g[i].clear();
```

```
    }
}
```

Junior e Falta de Ideias

```cpp
#include <bits/stdc++.h>

#define ff first
#define ss second
#define mp make_pair

using namespace std;

typedef long long ll;

vector<pair<int,int>> G[500005];
int subtree[500005], treesize, k;
bool vis[500005];
ll dist[500005], ans;

int dfs(int v, int p){
    subtree[v] = 1;
    for(pair<int,int> x : G[v])
        if(x.ff != p && !vis[x.ff]) subtree[v] += dfs(x.ff,v);
    return subtree[v];
}

int centroid(int v, int p){
    for(pair<int,int> x : G[v]){
        if(x.ff == p || vis[x.ff]) continue;
        if(subtree[x.ff]*2 > treesize) return centroid(x.ff,v);
    }
    return v;
}

void procurar_ans(int v, int p, int d_atual, ll custo){
    ans = min(ans, dist[k-d_atual] + custo);
    if(d_atual == k) return;
    for(pair<int,int> x : G[v]){
        if(!vis[x.ff] && x.ff != p)
            procurar_ans(x.ff,v,d_atual+1,custo+x.ss);
    }
}

void atualiza_distancia(int v, int p, int d_atual, ll custo){
    dist[d_atual] = min(dist[d_atual], custo);
    if(d_atual == k) return;
    for(pair<int,int> x : G[v]){
        if(!vis[x.ff] && x.ff != p)
            atualiza_distancia(x.ff,v,d_atual+1,custo+x.ss);
```

```cpp
    }
}

void decomp(int v, int p){
    treesize = dfs(v,v);
    // if(treesize < k) return;
    int cent = centroid(v,v);
    vis[cent] = 1;

    for(int i = 1; i <= treesize; i++)
        dist[i] = 1e18;

    for(pair<int,int> x : G[cent]){
        if(!vis[x.ff]){
            procurar_ans(x.ff,cent,1,x.ss);
            atualiza_distancia(x.ff,cent,1,x.ss);
        }
    }

    for(pair<int,int> x : G[cent]){
        if(!vis[x.ff])
            decomp(x.ff, cent);
    }
}

int main(){
    int n,i,a,b;

    scanf("%d%d", &n,&k);
    for(i = 2; i <= n; i++){
        scanf("%d%d", &a,&b);
        G[i].push_back(mp(a,b));
        G[a].push_back(mp(i,b));
    }
    ans = 1e18;
    decomp(1,-1);

    printf("%lld\n", ans == 1e18 ? -1 : ans);

    return 0;
}
```

Componentes Fortemente Conexas

Kosaraju

```cpp
vector<int> g[N], gt[N], S;

int vis[N], cor[N], tempo = 1;
```

```
void dfs(int u){
    vis[u] = 1;
    for(int v : g[u]) if(!vis[v])
        dfs(v);
    S.push_back(u);
}
int e;
void dfst(int u){
    cor[u] = e;
    for(int v : gt[u]) if(!cor[v])
        dfst(v);
}

int main(){

    memset(vis, 0, sizeof vis);
    memset(cor, 0, sizeof color);

    for(int i = 1; i <= n; i++) if(!vis[i])
        dfs(i);

    e = 0;
    for(int u : S) if(!cor[u])
        e++, dfst(u);

    return 0;
}
```

Tarjan

```
int num[N], low[N], vis[N], cor[N];
vector<int> g[N], S;

int t, e;
void dfs(int u){
    low[u] = num[u] = t++;
    S.push_back(u);
    vis[u] = 1;
    for(int v : g[u]){
        if(num[v] == -1) dfs(v);
        if(vis[v]) low[u] = min(low[u], low[v]);
    }
    if(low[u] == num[u]){
        e++;
        int v;
        do{
            v = s.back(), s.pop_back();
            vis[v] = 0, cor[v] = e;
        }while(v != u);
```

```
    }
}

int main(){
    memset(num, -1, sizeof num);
    memset(vis, 0, sizeof vis);
    memset(low, 0, sizeof low);
    for(int i = 1; i <= n; i++) if(num[i] == -1)
        dfs(i);
}
```

Clique

Clique máximo por meet in the meddle

```
long long adj[N], dp[N];

for(int i = 0; i < n; i++){
    for(int j = 0; j < n; j++){
        int x;
        scanf("%d",&x);
        if(x || i == j)
            adj[i] |= 1LL << j;
    }
}

int resto = n - n/2;
int C = n/2;
for(int i = 1; i < (1 << resto); i++){
    int x = i;
    for(int j = 0; j < resto; j++)
        if(i & (1 << j))
            x &= adj[j + C] >> C;
    if(x == i){
        dp[i] = __builtin_popcount(i);
    }
}

for(int i = 1; i < (1 << resto); i++)
    for(int j = 0; j < resto; j++)
        if(i & (1 << j))
            dp[i] = max(dp[i], dp[i ^ (1 << j)]);

int maxCliq = 0;
for(int i = 0; i < (1 << C); i++){
    int x = i, y = (1 << resto) - 1;
    for(int j = 0; j < C; j++)
        if(i & (1 << j))
            x &= adj[j] & ((1 << C) - 1), y &= adj[j] >> C;
```

```
    if(x != i) continue;
    maxCliq = max(maxCliq, __builtin_popcount(i) + dp[y]);
}
```

Strings

YATG

```cpp
#include <bits/stdc++.h>

#define ff first
#define ss second
#define mp make_pair
#define oo 1000000000

using namespace std;

int n, k, mat[300005][26], ans;

int dfs(int u){
    int ret = oo;
    ans++;

    for(int i = 0; i < 26; i++){
        if(mat[u][i]){
            ret = min(ret, dfs(mat[u][i]));
        }
    }
    if(ret == oo){
        ret = 0;
        ans++;
    }
    ret++;
    if(ret > k){
        ans++;
        ret = 1;
    }
    return ret;
}

int main(){

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

    char s[100005];

    int ptr = 1;

    for(int i = 0; i < n; i++){
        scanf(" %s", s);
```

```cpp
        int node = 0;
        for(int j = 0; s[j]; j++){
            int letra = s[j]-'a';
            if(!mat[node][letra])
                mat[node][letra] = ptr++;
            node = mat[node][letra];
        }
    }

    for(int i = 0; i < 26; i++)
        if(mat[0][i])
            dfs(mat[0][i]);

    ans += n;

    printf("%d\n", ans);

    return 0;
}
```

Trie ponteiro

```cpp
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

struct node{
    bool is_end;
    int prefixes, words, maxsize;
    struct node* edge[26];
    //initialize()
    //addword(vertex, word)
    //countPrefixes(vertex, prefix)
    //countWords(vertex, word)

    node(){
        maxsize = 0;
        prefixes = 0;
        words = 0;
        is_end = false;
        for(int i = 0; i < 26; i++) edge[i] = NULL;
    }

    void addWord(string word, int tam){
        maxsize = max(tam, maxsize);
        if(word.empty()){
            prefixes++;
            words++;
        }
```

```cpp
        else{
            prefixes++;
            int k = word[0] - 'a';
            if(edge[k] == NULL){
                node *p1 = new node();
                p1->addWord(word.substr(1,word.size()-1), tam );
                edge[k] = p1;
            }
            else
                edge[k]->addWord(word.substr(1,word.size()-1), tam );
        }
    }

    int countWords(string word){
        if(word.empty())
            return words;
        int k = word[0] - 'a';
        if(edge[k] == NULL)
            return 0;
        return edge[k]->countWords(word.substr(1,word.size()-1) );
    }

    int countPrefixes(string word){
        if(word.empty())
            return prefixes;
        int k = word[0] - 'a';
        if(edge[k] == NULL)
            return 0;
        return edge[k]->countPrefixes(word.substr(1,word.size()-1) );
    }

    int countSize(string word){
        if(word.empty())
            return maxsize;
        int k = word[0] - 'a';
        if(edge[k] == NULL)
            return 0;
        return edge[k]->countSize(word.substr(1,word.size()-1) );
    }
};

int main() {
    int n,m;
    string s;

    while(scanf("%d", &n) != EOF){
        node *trie = new node();
        while(n--){
            cin >> s;
            trie->addWord(s, s.size());
        }
        scanf("%d", &m);
```

```cpp
        while(m--){
            cin >> s;
            int x = trie->countPrefixes(s);
            if(x == 0)
                cout << "-1\n";
            else
                cout << x << "_" << trie->countSize(s) << endl;
        }
    }


    return 0;
}
```

Trie

```cpp
#include <bits/stdc++.h>

#define oo 1000000000000000000

using namespace std;

int mat[6400640][2];
int cnt[6400640][2];

int main(){
    int n, x, ptr, next = 1;
    char c;

    scanf("%d", &n);

    ptr = 0;
    for(int i = 30; i >= 0; i--){
        if(!mat[ptr][0]) mat[ptr][0] = next++;
        cnt[ptr][0]++;
        ptr = mat[ptr][0];
    }

    for(int i = 0; i < n; i++){
        scanf("_%c_%d", &c, &x);

        if(c == '+'){
            ptr = 0;
            for(int i = 30; i >= 0; i--){
                if((1 << i) & x){
                    if(!mat[ptr][1]) mat[ptr][1] = next++;
                    cnt[ptr][1]++;
                    ptr = mat[ptr][1];
                }
                else{
                    if(!mat[ptr][0]) mat[ptr][0] = next++;
```

```
            cnt[ptr][0]++;
            ptr = mat[ptr][0];
        }
    }
}
else if(c == '-'){
    ptr = 0;
    for(int i = 30; i >= 0; i--){
        if((1 << i) & x){
            cnt[ptr][1]--;
            ptr = mat[ptr][1];
        }
        else{
            cnt[ptr][0]--;
            ptr = mat[ptr][0];
        }
    }
}
else{
    int ans = 0;
    ptr = 0;
    for(int i = 30; i >= 0; i--){
        if((1 << i) & x){
            if(cnt[ptr][0] > 0){
                ans += (1 << i);
                ptr = mat[ptr][0];
            }
            else ptr = mat[ptr][1];
        }
        else{
            if(cnt[ptr][1] > 0){
                ans += (1 << i);
                ptr = mat[ptr][1];
            }
            else ptr = mat[ptr][0];
        }
    }
    printf("%d\n", ans);
}
    }

    return 0;
}
```

Aho Corasick

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

#define ff first
#define ss second
#define mp make_pair
```

```
using namespace std;

typedef long long ll;

int trie[1000005][52], fn[1000005];
int ptr;
bool passou[1000005];
vector<int> final_vec;

int insert(char *str){
    int v = 0;

    for(int i = 0; str[i]; i++){
        int to = str[i] >= 'a' ? str[i]-'a'+26 : str[i]-'A';
        if(trie[v][to])
            v = trie[v][to];
        else
            v = trie[v][to] = ptr++;
    }

    return v;
}

void init_aho(){
    queue<int> Q;

    Q.push(0);

    while(!Q.empty()){
        int t = Q.front(); Q.pop();

        for(int i = 0; i < 52; i++){
            if(trie[t][i]){
                int x = trie[t][i];
                Q.push(x);

                if(t){
                    fn[x] = fn[t];

                    while(fn[x] && trie[fn[x]][i] == 0) fn[x] = fn[fn[x]];
                    if(trie[fn[x]][i]) fn[x] = trie[fn[x]][i];
                    trie[x][i] = fn[x];
                }
            }
        }
    }
}

int main(){
    int t,q;
    char s[100005], s2[1005];
```

```
    scanf("%d", &t);
    while(t--){
        scanf("_%s", s);

        memset(trie, 0, sizeof trie);
        memset(passou, 0, sizeof passou);
        memset(fn, 0, sizeof fn);
        final_vec.clear();

        ptr = 1;

        scanf("%d", &q);

        // build trie
        while(q--){
            scanf("_%s", s2);
            final_vec.push_back(insert(s2));
        }

        // build failures
        init_aho();

        // simulate
        int v = 0;
        for(int i = 0; s[i]; i++){
            int to = s[i] >= 'a' ? s[i]-'a'+26 : s[i]-'A';
            v = trie[v][to];
            else{
                while(v){
                    v = fn[v];
                    passou[v] = 1;
                    if(trie[v][to]){
                        v = trie[v][to];
                        break;
                    }
                }
            }
            passou[v] = 1;
        }

        for(int i = 0; i < final_vec.size(); i++){
            if(passou[final_vec[i]]) printf("y\n");
            else printf("n\n");
        }
    }

    return 0;
}
```

Suffix Array

```
char s[N];
int n, sa[N], tsa[N], lcp[N], r[N], nr[N], c[N];

void sort(int k, int mx){
    mx++;
    memset(c, 0, sizeof(int) * mx);
    forn(i, n) c[i + k < n ? r[i+k]+1 : 1]++;
    partial_sum(c, c+mx, c);
    int t;
    forn(i, n) t = sa[i]+k < n ? r[ sa[i]+k ] : 0,
            tsa[ c[t]++ ] = sa[i];
    memcpy(sa, tsa, sizeof(int) * n);
}


void build_sa(){

    forn(i, n) sa[i] = i, r[i] = s[i];

    int t = 300, a, b;
    for(int sz = 1; sz < n; sz *= 2){
        sort(sz, t), sort(0, t);
        t = nr[ sa[0] ] = 0;
        for(int i = 1; i < n; i++){
            a = sa[i]+sz < n ? r[ sa[i]+sz ] : -1;
            b = sa[i-1]+sz < n ? r[ sa[i-1]+sz ] : -1;
            nr[ sa[i] ] = r[ sa[i] ] == r[ sa[i-1] ] && a == b ? t : ++t;
        }
        if(t == n-1) break;
        memcpy(r, nr, sizeof(int) * n);
    }
}


void build_lcp(){ // lcp[i] = lcp(s[:i], s[:i+1])
    int k = 0;
    forn(i, n) r[ sa[i] ] = i;

    forn(i, n){
        if(r[i] == n-1) k = 0;
        else{
            int j = sa[r[i]+1];
            while(i+k < n && j+k < n && s[i+k] == s[j+k]) k++;
        }
        lcp[r[i]] = k;
        if(k) k--;
    }
}
```

Z Algorithm

```
vector<int> z_algo(const string &s) {
```

```cpp
    int n = s.size(), L = 0, R = 0;
    vector<int> z(n, 0);
    for(int i = 1; i < n; i++){
        if(i <= R) z[i] = min(z[i-L], R - i + 1);
        while(z[i]+i < n && s[ z[i]+i ] == s[ z[i] ]) z[i]++;
        if(i+z[i]-1 > R) L = i, R = i + z[i] - 1;
    }
    return z;
}
```

Prefix function/KMP

```cpp
vector<int> pf(const string &s){
    int n = s.size();
    vector<int> b(n+1);
    b[0] = -1;
    int i = 0, j = -1;
    while(i < n){
        while(j >= 0 && s[i] != s[j]) j = b[j];
        b[++i] = ++j;
    }
    return b;
}

void kmp(const string &t, const string &p){
    vector<int> b = pf(p);
    int n = t.size();
    int j = 0;
    for(int i = 0; i < n; i++){
        while(j >= 0 && t[i] != p[j]) j = b[j];
        j++;
        if(j == m){
            //patern found
            j = b[j];
        }
    }
}
```

Min rotation

```cpp
// Lexicographically minimum rotation of a sequence
//
// Given a sequence s of length N, min_rotation(s, N) returns the start index
// of the lexicographically minimum rotation.
//
// Note: array s must be of length of at least 2 * N.
//
// Time complexity: O(N)

int min_rotation(int *s, int N) {
  REP(i, N) s[N+i] = s[i];
```

```cpp
  int a = 0;
  REP(b, N) REP(i, N) {
    if (a+i == b || s[a+i] < s[b+i]) { b += max(0, i-1); break; }
    if (s[a+i] > s[b+i]) { a = b; break; }
  }
  return a;
}
```

All palindrome

```cpp
// Finds all palindromes in a string
//
// Given a string s of length N, finds all palindromes as its substrings.
//
// After calling manacher(s, N, rad), rad[x] will be the radius of the largest
// palindrome centered at index x / 2.
// Example:
// s = b a n a n a a
// rad = 0000102010010
//
// Note: Array rad must be of length at least twice the length of the string.
// Also, "invalid" characters are denoted by -1, therefore the string must not
// contain such characters.
//
// Time complexity: O(N)
//
// Constants to configure:
// - MAX is the maximum length of the string

void manacher(char *s, int N, int *rad) {
  static char t[2*MAX];
  int m = 2*N - 1;

  REP(i, m) t[i] = -1;
  REP(i, N) t[2*i] = s[i];

  int x = 0;
  FOR(i, 1, m) {
    int &r = rad[i] = 0;
    if (i <= x+rad[x]) r = min(rad[x+x-i], x+rad[x]-i);
    while (i-r-1 >= 0 && i+r+1 < m && t[i-r-1] == t[i+r+1]) ++r;
    if (i+r >= x+rad[x]) x = i;
  }

  REP(i, m) if (i-rad[i] == 0 || i+rad[i] == m-1) ++rad[i];
  REP(i, m) rad[i] /= 2;
}
```

Geometria

Convex Hull

```
ll D(const Point& P, const Point& Q, const Point& R){
    return (P.x * Q.y + P.y * R.x + Q.x * R.y) -
        (R.x * Q.y + R.y * P.x + Q.x * P.y);
}

vector<Point> monotone_chain_ch(vector<Point> P){
    sort(P.begin(), P.end());

    vector<Point> L, U;

    for(auto p : P){
        while (L.size() >= 2 and D(L[L.size() - 2], L[L.size() -1], p) < 0)
            L.pop_back();

        L.push_back(p);
    }

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

    for(auto p : P){
        while (U.size() >= 2 and D(U[U.size() - 2], U[U.size() -1], p) < 0)
            U.pop_back();

        U.push_back(p);
    }

    L.pop_back();
    U.pop_back();

    L.reserve(L.size() + U.size());
    L.insert(L.end(), U.begin(), U.end());

    return L;
}
```

Nearest Points

```
struct pt {
    int x, y, id;
};

inline bool cmp_x (const pt & a, const pt & b) {
    return a.x < b.x || a.x == b.x && a.y < b.y;
}
```

```
inline bool cmp_y (const pt & a, const pt & b) {
    return a.y < b.y;
}

pt a[MAXN];

double mindist;
int ansa, ansb;

inline void upd_ans (const pt & a, const pt & b) {
    double dist = sqrt ((a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y) + .0);
    if (dist < mindist)
        mindist = dist, ansa = a.id, ansb = b.id;
}

void rec (int l, int r) {
    if (r - l <= 3) {
        for (int i=l; i<=r; ++i)
            for (int j=i+1; j<=r; ++j)
                upd_ans (a[i], a[j]);
        sort (a+l, a+r+1, &cmp_y);
        return;
    }

    int m = (l + r) >> 1;
    int midx = a[m].x;
    rec (l, m), rec (m+1, r);
    static pt t[MAXN];
    merge (a+l, a+m+1, a+m+1, a+r+1, t, &cmp_y);
    copy (t, t+r-l+1, a+l);

    int tsz = 0;
    for (int i=l; i<=r; ++i)
        if (abs (a[i].x - midx) < mindist) {
            for (int j=tsz-1; j>=0 && a[i].y - t[j].y < mindist; --j)
                upd_ans (a[i], t[j]);
            t[tsz++] = a[i];
        }
}

sort (a, a+n, &cmp_x);
mindist = 1E20;
rec (0, n-1);
```

Determinante

```
const double EPS = 1E-9;
int n;
vector < vector<double> > a (n, vector<double> (n));

double det = 1;
```

```
for (int i=0; i<n; ++i) {
   int k = i;
   for (int j=i+1; j<n; ++j)
      if (abs (a[j][i]) > abs (a[k][i]))
         k = j;
   if (abs (a[k][i]) < EPS) {
      det = 0;
      break;
   }
   swap (a[i], a[k]);
   if (i != k)
      det = -det;
   det *= a[i][i];
   for (int j=i+1; j<n; ++j)
      a[i][j] /= a[i][i];
   for (int j=0; j<n; ++j)
      if (j != i && abs (a[j][i]) > EPS)
         for (int k=i+1; k<n; ++k)
            a[j][k] -= a[i][k] * a[j][i];
}

cout << det;
```

Convex Hull Trick

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

#define ff first
#define ss second
#define pb push_back
#define eb emplace_back
#define mp make_pair

using namespace std;

const int N = 100005;

const double EPS = 1e-9;

int n;
int h[N], ht[N], c[N], ct[N], erased[N], id[N];

pair<double, double> H[N], C[N];

double getx(int *h, int *hc, int i, int j){
   return 1.0*(h[i] - h[j])/(hc[j] - hc[i]);
}

void cht(int *h, int *hc, pair<double, double> *H){
   for(int i = 0; i < n; i++) id[i] = i;
```

```
   sort(id, id+n, [=](int a, int b){
      if(hc[a] == hc[b]) return h[a] < h[b];
         return hc[a] < hc[b];
   });

   vector<int> v;

   for(int i = 0; i < n; i++){
      while(v.size() >= 1){

         bool taken = 0;
         double x3 = getx(h, hc, v[ v.size()-1], id[i]);
         if(x3 < EPS) taken = 1, v.pop_back();

         if(v.size() >= 2){
            double x1 = getx(h, hc, v[ v.size()-1], v[v.size()-2]);
            double x2 = getx(h, hc, v[ v.size()-2], id[i]);
            if(x2-EPS <= x1) v.pop_back();
            else break;
         }
         else if(!taken) break;
      }
      v.push_back(id[i]);
   }
   double last = 0;
   for(int i = 0; i < v.size(); i++){
      H[ v[i] ].ff = last;
      if(i+1 < v.size()) last = H[ v[i] ].ss = getx(h, hc, v[i], v[i+1]);
      else H[ v[i] ].ss = 1e50;
   }
}

int main(){

   scanf("%d", &n);

   for(int i = 0; i < n; i++)
      scanf("%d %d %d %d", h+i, ht+i, c+i, ct+i), c[i] *= -1, ct[i] *= -1;

   {
      multiset<pair<int, int> > hh, cc;
      for(int i = 0; i < n; i++)
         hh.insert(mp(h[i], ht[i])),
         cc.count(mp(c[i], ct[i]));
      for(int i = 0; i < n; i++)
         if(hh.count(mp(h[i], ht[i])) > 1 || cc.count(mp(c[i], ct[i])) > 1)
            erased[i] = 1;
   }

   for(int i = 0; i < n; i++)
      H[i].ff = H[i].ss = C[i].ff = C[i].ss = -1;
   cht(h, ht, H);
```

```
    cht(c, ct, C);

    int ans = 0;

    for(int i = 0; i < n; i++) if(!erased[i]){
        if(H[i].ff == -1 || C[i].ff == -1) continue;
        double a = max(H[i].ff, C[i].ff);
        double b = min(H[i].ss, C[i].ss);
        if(a+EPS <= b){
                ans++;
            }
        }
    }

    printf("%d\n", ans);

    return 0;
}
```