Notebook - Maratona de Programação

[UnB] HatsuneMiku não manda WA

Contents

1 Misc **2**
1.1 Submask . . . 2
1.2 Safe Map . . . 2
1.3 Ordered Set . . . 2
1.4 Bitwise . . . 2

2 DP **2**
2.1 Knapsack . . . 2
2.2 Lis . . . 2
2.3 Dp Digitos . . . 3

3 ED **3**
3.1 Prefixsum2d . . . 3
3.2 Sparse Table . . . 3
3.3 Dsu . . . 4
3.4 Minqueue . . . 4
3.5 Segtree Implicita Lazy . . . 4
3.6 Delta Encoding . . . 5

4 Strings **5**
4.1 Z Func . . . 5
4.2 Edit Distance . . . 5
4.3 Lcsubseq . . . 5
4.4 Kmp . . . 5
4.5 Hash . . . 5
4.6 Aho Corasick . . . 6
4.7 Lcs . . . 6

5 Geometria **6**
5.1 Mindistpair . . . 6
5.2 Inside Polygon . . . 6
5.3 3d . . . 7
5.4 Convex Hull . . . 8
5.5 Linear Transformation . . . 8
5.6 Voronoi . . . 8
5.7 Polygon Area . . . 8
5.8 Intersect Polygon . . . 9
5.9 Sort By Angle . . . 9
5.10 2d . . . 9

6 Grafos **11**
6.1 Dfs Tree . . . 11
6.2 Kosaraju . . . 11
6.3 Topological Sort . . . 11
6.4 Dijkstra . . . 12
6.5 Dinic . . . 12
6.6 Hungarian . . . 12
6.7 Floyd Warshall . . . 13
6.8 Lca . . . 13
6.9 Kruskal . . . 14
6.10 Ford . . . 14

7 Algoritmos **14**
7.1 Ternary Search . . . 14

8 Math **15**
8.1 Totient . . . 15
8.2 Inverso Mult . . . 15

8.3 Miller Habin 15
8.4 Matrix Exponentiation 15
8.5 Division Trick 16
8.6 Crivo 16
8.7 Bigmod 16
8.8 Linear Diophantine Equation 16

1 Misc

1.1 Submask

```
// O(3^n)
for (int m = 0; m < (1<<n); m++) {
    for (int s = m; s; s = (s-1) & m) {
        // s is every submask of m
    }
}

// O(2^n * n) SOS dp like
for (int b = n-1; b >= 0; b--) {
    for (int m = 0; m < (1 << n); m++) {
        if (j & (1 << b)) {
            // propagate info through submasks
            amount[j ^ (1 << b)] += amount[j];
        }
    }
}
```

1.2 Safe Map

```
struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        // http://xorshift.di.unimi.it/splitmix64.c
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }

    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM = chrono::steady_clock::now().time_since_epoch().count();
        return splitmix64(x + FIXED_RANDOM);
    }
};

unordered_map<long long, int, custom_hash> safe_map;

// when using pairs
struct custom_hash {
    inline size_t operator ()(const pii & a) const {
        return (a.first << 6) ^ (a.first >> 2) ^ 2038074743 ^ a.second;
    }
};
```

1.3 Ordered Set

```
#include <bits/extc++.h>
// or
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace __gnu_pbds; // or pb_ds;
template<typename T, typename B = null_type>
using ordered_set = tree<T, B, less<T>, rb_tree_tag, tree_order_statistics_node_update>;

// order_of_key(k)  : Number of items strictly smaller than k
// find_by_order(k) : K-th element in a set (counting from zero)

// to swap two sets, use a.swap(b);
```

1.4 Bitwise

```
// Least significant bit (lsb)
    int lsb(int x) { return x&-x; }
    int lsb(int x) { return __builtin_ctz(x); } // bit position
// Most significant bit (msb)
    int msb(int x) { return 32-1-__builtin_clz(x); } // bit position

// Power of two
    bool isPowerOfTwo(int x){ return x && (!(x&(x-1))); }

// floor(log2(x))
int flog2(int x)  { return 32-1-__builtin_clz(x); }
int flog2ll(ll x) { return 64-1-__builtin_clzll(x); }

// Built-in functions
// Number of bits 1
__builtin_popcount()
__builtin_popcountll()

// Number of leading zeros
__builtin_clz()
__builtin_clzll()

// Number of trailing zeros
__builtin_ctz()
__builtin_ctzll()
```

2 DP

2.1 Knapsack

```
// Caso base, como i == n
dp[0][0] = 0;

// Itera por todos os estados
for(int i = 1; i <= n; ++i)
    for(int P = 0; P <= w; ++P){
        int &temp = dp[i][P];
        // Primeira possibilidade, ãno pega i
        temp = dp[i - 1][P];

        // Segunda possibilidade, se puder, pega o item
        if(P - p[i] >= 0)
            temp = max(temp, dp[i - 1][P - p[i]] + v[i]);

        ans = max(ans, temp);
    }
```

2.2 Lis

```
multiset<int> S;
for(int i=0;i<n;i++){
    auto it = S.upper_bound(vet[i]); // low for inc
    if(it != S.end())
        S.erase(it);
    S.insert(vet[i]);
}
// size of the lis
int ans = S.size();

vi LIS(const vi &elements){
    auto compare = [&](int x, int y) {
        return elements[x] < elements[y];
    };
    set< int, decltype(compare) > S(compare);

    vi previous( elements.size(), -1 );
    for(int i=0; i<int( elements.size() ); ++i){
        auto it = S.insert(i).first;
        if(it != S.begin())
            previous[i] = *prev(it);
        if(*it == i and next(it) != S.end())
            S.erase(next(it));
    }

    vi answer;
    answer.push_back( *S.rbegin() );
    while ( previous[answer.back()] != -1 )
        answer.push_back( previous[answer.back()] );
    reverse( answer.begin(), answer.end() );
    return answer;
}
```

2.3 Dp Digitos

```
// dp de quantidade de numeros <= r com ate qt digitos diferentes de 0
ll dp(int idx, string& r, bool menor, int qt, vector<vector<vi>>& tab) {
    if(qt > 3) return 0;
    if(idx >= r.size()) {
        return 1;
    }
    if(tab[idx][menor][qt] != -1)
        return tab[idx][menor][qt];

    ll res = 0;
    for(int i = 0; i <= 9; i++) {
        if(menor or i <= r[idx]-'0') {
            res += dp(idx+1, r, menor or i < (r[idx]-'0') , qt+(i>0), tab);
        }
    }

    return tab[idx][menor][qt] = res;
}
```

3 ED

3.1 Prefixsum2d

```
ll find_sum(vector<vi> &mat, int x1, int y1, int x2, int y2){
    // superior-esq(x1,y1) (x2,y2)inferior-dir
    return mat[x2][y2]-mat[x2][y1-1]-mat[x1-1][y2]+mat[x1-1][y1-1];
}

int main(){

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

}
```

3.2 Sparse Table

```
int logv[N+1];
void make_log() {
    logv[1] = 0; // pre-computar tabela de log
    for (int i = 2; i <= N; i++)
        logv[i] = logv[i/2] + 1;
}
struct Sparse {
    int n;
    vector<vector<int>> st;

    Sparse(vector<int>& v) {
        n = v.size();
        int k = logv[n];
        st.assign(n+1, vector<int>(k+1, 0));

        for (int i=0;i<n;i++) {
            st[i][0] = v[i];
        }

        for(int j = 1; j <= k; j++) {
            for(int i = 0; i + (1 << j) <= n; i++) {
                st[i][j] = f(st[i][j-1], st[i + (1 << (j-1))][j-1]);
            }
        }
    }

    int f(int a, int b) {
        return min(a, b);
    }

    int query(int l, int r) {
        int k = logv[r-l+1];
        return f(st[l][k], st[r - (1 << k) + 1][k]);
    }
};


struct Sparse2d {
    int n, m;
    vector<vector<vector<int>>> st;

    Sparse2d(vector<vector<int>> mat) {
        n = mat.size();
        m = mat[0].size();
        int k = logv[min(n, m)];

        st.assign(n+1, vector<vector<int>>(m+1, vector<int>(k+1)));
        for(int i = 0; i < n; i++)
            for(int j = 0; j < m; j++)
                st[i][j][0] = mat[i][j];

        for(int j = 1; j <= k; j++) {
            for(int x1 = 0; x1 < n; x1++) {
                for(int y1 = 0; y1 < m; y1++) {
                    int delta = (1 << (j-1));
                    if(x1+delta >= n or y1+delta >= m) continue;

                    st[x1][y1][j] = st[x1][y1][j-1];
                    st[x1][y1][j] = f(st[x1][y1][j], st[x1+delta][y1][j-1]);
                    st[x1][y1][j] = f(st[x1][y1][j], st[x1][y1+delta][j-1]);
                    st[x1][y1][j] = f(st[x1][y1][j], st[x1+delta][y1+delta][j-1]);
                }
            }
        }
    }

    // so funciona para quadrados
    int query(int x1, int y1, int x2, int y2) {
        assert(x2-x1+1 == y2-y1+1);
        int k = logv[x2-x1+1];
        int delta = (1 << k);

        int res = st[x1][y1][k];
        res = f(res, st[x2 - delta+1][y1][k]);
        res = f(res, st[x1][y2 - delta+1][k]);
        res = f(res, st[x2 - delta+1][y2 - delta+1][k]);
        return res;
    }

    int f(int a, int b) {
        return a | b;
    }

};
```

3.3 Dsu

```
struct DSU {
    int n;
    vector<int> parent, size;

    DSU(int n): n(n) {
        parent.resize(n, 0);
        size.assign(n, 1);

        for(int i=0;i<n;i++)
            parent[i] = i;
    }

    int find(int a) {
        if(a == parent[a]) return a;
        return parent[a] = find(parent[a]);
    }

    void join(int a, int b) {
        a = find(a); b = find(b);
        if(a != b) {
            if(size[a] < size[b]) swap(a, b);
            parent[b] = a;
            size[a] += size[b];
        }
    }
};
```

3.4 Minqueue

```
struct MinQ {
    stack<pair<ll,ll>> in;
    stack<pair<ll,ll>> out;

    void add(ll val) {
        ll minimum = in.empty() ? val : min(val, in.top().ss);
        in.push({val, minimum});
    }

    ll pop() {
        if(out.empty()) {
            while(!in.empty()) {
                ll val = in.top().ff;
                in.pop();
                ll minimum = out.empty() ? val : min(val, out.top().ss);
                out.push({val, minimum});
            }
        }
        ll res = out.top().ff;
        out.pop();
        return res;
    }

    ll minn() {
        ll minimum = LLINF;
        if(in.empty() || out.empty())
            minimum = in.empty() ? (ll)out.top().ss : (ll)in.top().ss;
        else
            minimum = min((ll)in.top().ss, (ll)out.top().ss);

        return minimum;
    }

    ll size() {
        return in.size() + out.size();
    }
};
```

3.5 Segtree Implicita Lazy

```
struct node{
    pll val;
    ll lazy;
    ll l, r;
    node(){
        l=-1;r=-1;val={0,0};lazy=0;
    }
};

node tree[40*MAX];
int id = 2;
ll N=1e9+10;

pll merge(pll A, pll B){
    if(A.ff==B.ff) return {A.ff, A.ss+B.ss};
    return (A.ff<B.ff ? A:B);
}

void prop(ll l, ll r, int no){
    ll mid = (l+r)/2;
    if(l!=r){
        if(tree[no].l==-1){
            tree[no].l = id++;
            tree[tree[no].l].val = {0, mid-l+1};
        }
        if(tree[no].r==-1){
            tree[no].r = id++;
            tree[tree[no].r].val = {0, r-(mid+1)+1};
        }
        tree[tree[no].l].lazy += tree[no].lazy;
        tree[tree[no].r].lazy += tree[no].lazy;
    }
    tree[no].val.ff += tree[no].lazy;
    tree[no].lazy=0;
}

void update(int a, int b, int x, ll l=0, ll r=2*N, ll no=1){
    prop(l, r, no);
    if(a<=l and r<=b){
        tree[no].lazy += x;
        prop(l, r, no);
        return;
    }
    if(r<a or b<l) return;
    int m = (l+r)/2;
    update(a, b, x, l, m, tree[no].l);
    update(a, b, x, m+1, r, tree[no].r);

    tree[no].val = merge(tree[tree[no].l].val, tree[tree[no].r].val);
}

pll query(int a, int b, int l=0, int r=2*N, int no=1){
    prop(l, r, no);
    if(a<=l and r<=b) return tree[no].val;
    if(r<a or b<l) return {INF, 0};
    int m = (l+r)/2;
    int left = tree[no].l, right = tree[no].r;

    return tree[no].val = merge(query(a, b, l, m, left),
                                query(a, b, m+1, r, right));
}
```

3.6 Delta Encoding

```
// Delta encoding

for(int i=0;i<q;i++){
    int l,r,x;
    cin >> l >> r >> x;
    delta[l] += x;
    delta[r+1] -= x;
}

int atual = 0;

for(int i=0;i<n;i++){
    atual += delta[i];
    v[i] += atual;
}
```

4 Strings

4.1 Z Func

```
vector<int> Z(string s) {
    int n = s.size();
    vector<int> z(n);
    int l = 0, r = 0;
    for (int i = 1; i < n; i++) {
        z[i] = max(0, min(z[i - l], r - i + 1));
        while (i + z[i] < n and s[z[i]] == s[i + z[i]]) {
            l = i; r = i + z[i]; z[i]++;
        }
    }
    return z;
}
```

4.2 Edit Distance

```
int edit_distance(int a, int b, string& s, string& t) {
    // indexado em 0, transforma s em t
    if(a == -1) return b+1;
    if(b == -1) return a+1;
    if(tab[a][b] != -1) return tab[a][b];

    int ins = INF, del = INF, mod = INF;
    ins = edit_distance(a-1, b, s, t) + 1;
    del = edit_distance(a, b-1, s, t) + 1;
    mod = edit_distance(a-1, b-1, s, t) + (s[a] != t[b]);

    return tab[a][b] = min(ins, min(del, mod));
}
```

4.3 Lcsubseq

```
// Longest Common Subsequence
string lcs(string x, string y){
    int n = x.size(), m = y.size();
    vector<vi> dp(n+1, vi(m+1, 0));

    for(int i=0;i<=n;i++){
        for(int j=0;j<=m;j++){
            if(!i or !j)
                dp[i][j]=0;
            else if(x[i-1] == y[j-1])
                dp[i][j]=dp[i-1][j-1]+1;
            else
                dp[i][j]=max(dp[i-1][j], dp[i][j-1]);
        }
    }

    // int len = dp[n][m];
    string ans="";

    // recover string
    int i = n-1, j = m-1;
    while(i>=0 and j>=0){
        if(x[i] == y[j]){
            ans.pb(x[i]);
            i--; j--;
        }else if(dp[i][j+1]>dp[i+1][j])
            i--;
        else
            j--;
    }

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

    return ans;
}
```

4.4 Kmp

```
string p;
int neighbor[N];
int walk(int u, char c) { // leader after inputting `c`
    while (u != -1 && (u+1 >= (int)p.size() || p[u + 1] != c)) // leader doesn't match
        u = neighbor[u];
    return p[u + 1] == c ? u+1 : u;
}
void build() {
    neighbor[0] = -1; // -1 is the leftmost state
    for (int i = 1; i < (int)p.size(); i++)
        neighbor[i] = walk(neighbor[i-1], p[i]);
}
```

4.5 Hash

```
// String Hash template
// constructor(s) - O(|s|)
// query(l, r) - returns the hash of the range [l,r] from left to right - O(1)
// query_inv(l, r) from right to left - O(1)

struct Hash {
    const ll P = 31;
    int n; string s;
    vector<ll> h, hi, p;
    Hash() {}
    Hash(string s): s(s), n(s.size()), h(n), hi(n), p(n) {
        for (int i=0;i<n;i++) p[i] = (i ? P*p[i-1]:1) % MOD;
        for (int i=0;i<n;i++)
            h[i] = (s[i] + (i ? h[i-1]:0) * P) % MOD;
        for (int i=n-1;i>=0;i--)
            hi[i] = (s[i] + (i+1<n ? hi[i+1]:0) * P) % MOD;
    }
    int query(int l, int r) {
        ll hash = (h[r] - (l ? h[l-1]*p[r-l+1]%MOD : 0));
        return hash < 0 ? hash + MOD : hash;
    }
    int query_inv(int l, int r) {
        ll hash = (hi[l] - (r+1 < n ? hi[r+1]*p[r-l+1] % MOD : 0));
        return hash < 0 ? hash + MOD : hash;
    }
};
```

4.6 Aho Corasick

```
// https://github.com/joseleite19/icpc-notebook/blob/master/code/string/aho_corasick.cpp
const int A = 26;
int to[N][A];
int ne = 2, fail[N], term[N];
void add_string(string str, int id){
    int p = 1;
    for(auto c: str){
        int ch = c - 'a'; // !
        if(!to[p][ch]) to[p][ch] = ne++;
        p = to[p][ch];
    }
    term[p]++;
}
void init(){
    for(int i = 0; i < ne; i++) fail[i] = 1;
    queue<int> q; q.push(1);
    int u, v;
    while(!q.empty()){
        u = q.front(); q.pop();
        for(int i = 0; i < A; i++){
            if(to[u][i]){
                v = to[u][i]; q.push(v);
                if(u != 1){
                    fail[v] = to[ fail[u] ][i];
                    term[v] += term[ fail[v] ];
                }
            }
            else if(u != 1) to[u][i] = to[ fail[u] ][i];
            else to[u][i] = 1;
        }
    }
}
```

4.7 Lcs

```
string LCSubStr(string X, string Y)
{
    int m = X.size();
    int n = Y.size();

    int result = 0, end;
    int len[2][n];
    int currRow = 0;

    for(int i=0;i<=m;i++){
        for(int j=0;j<=n;j++){
            if(i==0 || j==0)
                len[currRow][j] = 0;
            else if(X[i-1] == Y[j-1]){
                len[currRow][j] = len[1-currRow][j-1] + 1;
                if(len[currRow][j] > result){
                    result = len[currRow][j];
                    end = i - 1;
                }
            }
            else
                len[currRow][j] = 0;
        }

        currRow = 1 - currRow;
    }

    if(result==0)
        return string();

    return X.substr(end - result + 1, result);
}
```

5 Geometria

5.1 Mindistpair

```
ll MinDistPair(vp &vet){
    int n = vet.size();
    sort(vet.begin(), vet.end());
    set<point> s;

    ll best_dist = LLINF;
    int j=0;
    for(int i=0;i<n;i++){
        ll d = ceil(sqrt(best_dist));
        while(j<n and vet[i].x-vet[j].x >= d){
            s.erase(point(vet[j].y, vet[j].x));
            j++;
        }

        auto it1 = s.lower_bound({vet[i].y - d, vet[i].x});
        auto it2 = s.upper_bound({vet[i].y + d, vet[i].x});

        for(auto it=it1; it!=it2; it++){
            ll dx = vet[i].x - it->y;
            ll dy = vet[i].y - it->x;
            if(best_dist > dx*dx + dy*dy){
                best_dist = dx*dx + dy*dy;
                // vet[i] e inv(it)
            }
        }

        s.insert(point(vet[i].y, vet[i].x));
    }
    return best_dist;
}
```

5.2 Inside Polygon

```
// Convex O(logn)

bool insideT(point a, point b, point c, point e){
    int x = ccw(a, b, e);
    int y = ccw(b, c, e);
    int z = ccw(c, a, e);
    return !((x==1 or y==1 or z==1) and (x==-1 or y==-1 or z==-1));
}

bool inside(vp &p, point e){ // ccw
    int l=2, r=(int)p.size()-1;
    while(l<r){
        int mid = (l+r)/2;
        if(ccw(p[0], p[mid], e) == 1)
            l=mid+1;
        else{
            r=mid;
        }
    }
    // bordo
    // if(r==(int)p.size()-1 and ccw(p[0], p[r], e)==0) return false;
    // if(r==2 and ccw(p[0], p[1], e)==0) return false;
    // if(ccw(p[r], p[r-1], e)==0) return false;
    return insideT(p[0], p[r-1], p[r], e);
}


// Any O(n)

int inside(vp &p, point pp){
    // 1 - inside / 0 - boundary / -1 - outside
    int n = p.size();
    for(int i=0;i<n;i++){
        int j = (i+1)%n;
        if(line({p[i], p[j]}).inside_seg(pp))
            return 0;
    }
    int inter = 0;
    for(int i=0;i<n;i++){
        int j = (i+1)%n;
        if(p[i].x <= pp.x and pp.x < p[j].x and ccw(p[i], p[j], pp)==1)
            inter++; // up
        else if(p[j].x <= pp.x and pp.x < p[i].x and ccw(p[i], p[j], pp)==-1)
            inter++; // down
    }

    if(inter%2==0) return -1; // outside
    else return 1; // inside
}
```

5.3 3d

```
// typedef ll cod;
// bool eq(cod a, cod b){ return (a==b); }

const ld EPS = 1e-6;
#define vp vector<point>
typedef ld cod;
bool eq(cod a, cod b){ return fabs(a - b) <= EPS; }

struct point
{
    cod x, y, z;
    point(cod x=0, cod y=0, cod z=0): x(x), y(y), z(z) {}

    point operator+(const point &o) const {
        return {x+o.x, y+o.y, z+o.z};
    }
    point operator-(const point &o) const {
        return {x-o.x, y-o.y, z-o.z};
    }
    point operator*(cod t) const {
        return {x*t, y*t, z*t};
    }
    point operator/(cod t) const {
        return {x/t, y/t, z/t};
    }
    bool operator==(const point &o) const {
        return eq(x, o.x) and eq(y, o.y) and eq(z, o.z);
    }
    cod operator*(const point &o) const { // dot
        return x*o.x + y*o.y + z*o.z;
    }
    point operator^(const point &o) const { // cross
        return point(y*o.z - z*o.y,
                     z*o.x - x*o.z,
                     x*o.y - y*o.x);
    }
};

ld norm(point a) { // Modulo
    return sqrt(a * a);
}
cod norm2(point a) {
    return a * a;
}
bool nulo(point a) {
    return (eq(a.x, 0) and eq(a.y, 0) and eq(a.z, 0));
}
ld proj(point a, point b) { // a sobre b
    return (a*b)/norm(b);
}
ld angle(point a, point b) { // em radianos
    return acos((a*b) / norm(a) / norm(b));
}

cod triple(point a, point b, point c) {
    return (a * (b^c)); // Area do paralelepipedo
}

point normilize(point a) {
    return a/norm(a);
}

struct plane {
    cod a, b, c, d;
    point p1, p2, p3;
    plane(point p1=0, point p2=0, point p3=0): p1(p1), p2(p2), p3(p3) {
        point aux = (p1-p3)^(p2-p3);
        a = aux.x; b = aux.y; c = aux.z;
        d = -a*p1.x - b*p1.y - c*p1.z;
    }
    plane(point p, point normal) {
        normal = normilize(normal);
        a = normal.x; b = normal.y; c = normal.z;
        d = -(p*normal);
    }

    // ax+by+cz+d = 0;
    cod eval(point &p) {
        return a*p.x + b*p.y + c*p.z + d;
    }
};

cod dist(plane pl, point p) {
    return fabs(pl.a*p.x + pl.b*p.y + pl.c*p.z + pl.d) / sqrt(pl.a*pl.a + pl.b*pl.b + pl.c*pl.c);
}

point rotate(point v, point k, ld theta) {
    // Rotaciona o vetor v theta graus em torno do eixo k
    // theta *= PI/180; // graus
    return (
        v*cos(theta)) +
        ((k^v)*sin(theta)) +
        (k*(k*v))*(1-cos(theta)
    );
}

// 3d line inter / mindistance
cod d(point p1, point p2, point p3, point p4) {
    return (p2-p1) * (p4-p3);
}
vector<point> inter3d(point p1, point p2, point p3, point p4) {
    cod mua = ( d(p1, p3, p4, p3) * d(p4, p3, p2, p1) - d(p1, p3, p2, p1) * d(p4, p3, p4, p3) )
            / ( d(p2, p1, p2, p1) * d(p4, p3, p4, p3) - d(p4, p3, p2, p1) * d(p4, p3, p2, p1) );
    cod mub = ( d(p1, p3, p4, p3) + mua * d(p4, p3, p2, p1) ) / d(p4, p3, p4, p3);
    point pa = p1 + (p2-p1) * mua;
    point pb = p3 + (p4-p3) * mub;
    if (pa == pb) return {pa};
    return {};
}
```

5.4 Convex Hull

```
vp convex_hull(vp P)
{
    sort(P.begin(), P.end());
    vp L, U;
    for(auto p: P){
        while(L.size()>=2 and ccw(L.end()[-2], L.back(), p)!=1)
            L.pop_back();
        L.push_back(p);
    }
    reverse(P.begin(), P.end());
    for(auto p: P){
        while(U.size()>=2 and ccw(U.end()[-2], U.back(), p)!=1)
            U.pop_back();
        U.push_back(p);
    }
    L.pop_back();
    L.insert(L.end(), U.begin(), U.end()-1);
    return L;
}
```

5.5 Linear Transformation

```
// Apply linear transformation (p -> q) to r.
point linear_transformation(point p0, point p1, point q0, point q1, point r) {
    point dp = p1-p0, dq = q1-q0, num((dp^dq), (dp^dq));
    return q0 + point((r-p0)^(num), (r-p0)*(num))/(dp*dp);
}
```

5.6 Voronoi

```
bool polygonIntersection(line &seg, vp &p) {
    long double l = -1e18, r = 1e18;
    for(auto ps : p) {
        long double z = seg.eval(ps);
        l = max(l, z);
        r = min(r, z);
    }
    return l - r > EPS;
}

int w, h;

line getBisector(point a, point b) {
    line ans(a, b);
    swap(ans.a, ans.b);
    ans.b *= -1;
    ans.c = ans.a * (a.x + b.x) * 0.5 + ans.b * (a.y + b.y) * 0.5;
    return ans;
}

vp cutPolygon(vp poly, line seg) {
    int n = (int) poly.size();
    vp ans;
    for(int i = 0; i < n; i++) {
        double z = seg.eval(poly[i]);
        if(z > -EPS) {
            ans.push_back(poly[i]);
        }
        double z2 = seg.eval(poly[(i + 1) % n]);
        if((z > EPS && z2 < -EPS) || (z < -EPS && z2 > EPS)) {
            ans.push_back(inter_line(seg, line(poly[i], poly[(i + 1) % n]))[0]);
        }
    }
    return ans;
}

// BE CAREFUL!
// the first point may be any point
// O(N^3)
vp getCell(vp pts, int i) {
    vp ans;
    ans.emplace_back(0, 0);
    ans.emplace_back(1e6, 0);
    ans.emplace_back(1e6, 1e6);
    ans.emplace_back(0, 1e6);
    for(int j = 0; j < (int) pts.size(); j++) {
        if(j != i) {
            ans = cutPolygon(ans, getBisector(pts[i], pts[j]));
        }
    }
    return ans;
}

// O(N^2) expected time
vector<vp> getVoronoi(vp pts) {
    // assert(pts.size() > 0);
    int n = (int) pts.size();
    vector<int> p(n, 0);
    for(int i = 0; i < n; i++) {
        p[i] = i;
    }
    shuffle(p.begin(), p.end(), rng);
    vector<vp> ans(n);
    ans[0].emplace_back(0, 0);
    ans[0].emplace_back(w, 0);
    ans[0].emplace_back(w, h);
    ans[0].emplace_back(0, h);
    for(int i = 1; i < n; i++) {
        ans[i] = ans[0];
    }
    for(auto i : p) {
        for(auto j : p) {
            if(j == i) break;
            auto bi = getBisector(pts[j], pts[i]);
            if(!polygonIntersection(bi, ans[j])) continue;
            ans[j] = cutPolygon(ans[j], getBisector(pts[j], pts[i]));
            ans[i] = cutPolygon(ans[i], getBisector(pts[i], pts[j]));
        }
    }
    return ans;
}
```

5.7 Polygon Area

```
ll area = 0;

for(int i = 0; i < n - 1; ++i){
    area += pontos[i].x*pontos[i+1].y - pontos[i+1].x*pontos[i].y;
}
area += pontos[n-1].x*pontos[0].y - pontos[0].x*pontos[n-1].y;

area = abs(area);
```

5.8 Intersect Polygon

```
bool intersect(vector<point> A, vector<point> B) // Ordered ccw
{
    for(auto a: A)
        if(inside(B, a))
            return true;
    for(auto b: B)
        if(inside(A, b))
            return true;

    if(inside(B, center(A)))
        return true;

    return false;
}
```

5.9 Sort By Angle

```
// Comparator funcion for sorting points by angle

int ret[2][2] = {{3, 2},{4, 1}};
inline int quad(point p) {
    return ret[p.x >= 0][p.y >= 0];
}

bool comp(point a, point b) { // ccw
    int qa = quad(a), qb = quad(b);
    return (qa == qb ? (a ^ b) > 0 : qa < qb);
}

// only vectors in range [x+0, x+180)
bool comp(point a, point b){
    return (a ^ b) > 0; // ccw
    // return (a ^ b) < 0; // cw
}
```

5.10 2d

```
#define vp vector<point>
#define ld long double
const ld EPS = 1e-6;
const ld PI = acos(-1);

typedef ld T;
bool eq(T a, T b){ return abs(a - b) <= EPS; }

struct point{
    T x, y;
    int id;
    point(T x=0, T y=0): x(x), y(y){}

    point operator+(const point &o) const{ return {x + o.x, y + o.y}; }
    point operator-(const point &o) const{ return {x - o.x, y - o.y}; }
    point operator*(T t) const{ return {x * t, y * t }; }
    point operator/(T t) const{ return {x / t, y / t }; }
    T operator*(const point &o) const{ return x * o.x + y * o.y; }
    T operator^(const point &o) const{ return x * o.y - y * o.x; }
    bool operator<(const point &o) const{
        return (eq(x, o.x) ? y < o.y : x < o.x);
    }
    bool operator==(const point &o) const{
        return eq(x, o.x) and eq(y, o.y);
    }
    friend ostream& operator<<(ostream& os, point p){
        return os << "(" << p.x << "," << p.y << ")"; }
};

int ccw(point a, point b, point e){ // -1=dir; 0=collinear; 1=esq;
    T tmp = (b-a) ^ (e-a); // vector from a to b
    return (tmp > EPS) - (tmp < -EPS);
}

ld norm(point a){ // Modulo
    return sqrt(a * a);
}
T norm2(point a){
    return a * a;
}
bool nulo(point a){
    return (eq(a.x, 0) and eq(a.y, 0));
}
point rotccw(point p, ld a){
    // a = PI*a/180; // graus
    return point((p.x*cos(a)-p.y*sin(a)), (p.y*cos(a)+p.x*sin(a)));
}
point rot90cw(point a) { return point(a.y, -a.x); };
point rot90ccw(point a) { return point(-a.y, a.x); };

ld proj(point a, point b){ // a sobre b
    return a*b/norm(b);
}
ld angle(point a, point b){ // em radianos
    ld ang = a*b / norm(a) / norm(b);
    return acos(max(min(ang, (ld)1), (ld)-1));
}
ld angle_vec(point v){
    // return 180/PI*atan2(v.x, v.y); // graus
    return atan2(v.x, v.y);
}
ld order_angle(point a, point b){ // from a to b ccw (a in front of b)
    ld aux = angle(a,b)*180/PI;
    return ((a^b)<=0 ? aux:360-aux);
}
bool angle_less(point a1, point b1, point a2, point b2){ // ang(a1,b1) <= ang(a2,b2)
    point p1((a1*b1), abs((a1^b1)));
    point p2((a2*b2), abs((a2^b2)));
    return (p1^p2) <= 0;
}

ld area(vp &p){ // (points sorted)
    ld ret = 0;
    for(int i=2;i<(int)p.size();i++)
        ret += (p[i]-p[0])^(p[i-1]-p[0]);
    return abs(ret/2);
}
ld areaT(point &a, point &b, point &c){
    return abs((b-a)^(c-a))/2.0;
}

point center(vp &A){
    point c = point();
    int len = A.size();
    for(int i=0;i<len;i++)
        c=c+A[i];
    return c/len;
}

point forca_mod(point p, ld m){
    ld cm = norm(p);
    if(cm<EPS) return point();
    return point(p.x*m/cm,p.y*m/cm);
}

ld param(point a, point b, point v){
    // v = t*(b-a) + a // return t;
    // assert(line(a, b).inside_seg(v));
    return ((v-a) * (b-a)) / ((b-a) * (b-a));
}

bool simetric(vp &a){ //ordered
    int n = a.size();
    point c = center(a);
    if(n&1) return false;
    for(int i=0;i<n/2;i++)
        if(ccw(a[i], a[i+n/2], c) != 0)
            return false;
    return true;
}

point mirror(point m1, point m2, point p){
    // mirror point p around segment m1m2
    point seg = m2-m1;
    ld t0 = ((p-m1)*seg) / (seg*seg);
    point ort = m1 + seg*t0;
    point pm = ort-(p-ort);
    return pm;
}


///////////
//  Line  //
///////////

struct line{
    point p1, p2;
    T a, b, c; // ax+by+c = 0;
    // y-y1 = ((y2-y1)/(x2-x1))(x-x1)
    line(point p1=0, point p2=0): p1(p1), p2(p2){
        a = p1.y - p2.y;
        b = p2.x - p1.x;
        c = p1 ^ p2;
    }
    line(T a=0, T b=0, T c=0): a(a), b(b), c(c){
        // Gera os pontos p1 p2 dados os coeficientes
        // isso aqui eh um lixo mas quebra um galho kkkkkk
        if(b==0){
            p1 = point(1, -c/a);
            p2 = point(0, -c/a);
        }else{
            p1 = point(1, (-c-a*1)/b);
            p2 = point(0, -c/b);
        }
    }

    T eval(point p){
        return a*p.x+b*p.y+c;
    }
    bool inside(point p){
        return eq(eval(p), 0);
    }
    point normal(){
        return point(a, b);
    }

    bool inside_seg(point p){
        return (
            ((p1-p) ^ (p2-p)) == 0 and
            ((p1-p) * (p2-p)) <= 0
        );
    }

};

// be careful with precision error
vp inter_line(line l1, line l2){
    ld det = l1.a*l2.b - l1.b*l2.a;
    if(det==0) return {};
    ld x = (l1.b*l2.c - l1.c*l2.b)/det;
    ld y = (l1.c*l2.a - l1.a*l2.c)/det;
    return {point(x, y)};
}

// segments not collinear
vp inter_seg(line l1, line l2){
    vp ans = inter_line(l1, l2);
    if(ans.empty() or !l1.inside_seg(ans[0]) or !l2.inside_seg(ans[0]))
        return {};
    return ans;
}
bool seg_has_inter(line l1, line l2){
    return ccw(l1.p1, l1.p2, l2.p1) * ccw(l1.p1, l1.p2, l2.p2) < 0 and
            ccw(l2.p1, l2.p2, l1.p1) * ccw(l2.p1, l2.p2, l1.p2) < 0;
}

ld dist_seg(point p, point a, point b){ // point - seg
    if((p-a)*(b-a) < EPS) return norm(p-a);
    if((p-b)*(a-b) < EPS) return norm(p-b);
    return abs((p-a)^(b-a)) / norm(b-a);
}

ld dist_line(point p, line l){ // point - line
    return abs(l.eval(p))/sqrt(l.a*l.a + l.b*l.b);
}

line bisector(point a, point b){
    point d = (b-a)*2;
    return line(d.x, d.y, a*a - b*b);
}

line perpendicular(line l, point p){ // passes through p
    return line(l.b, -l.a, -l.b*p.x + l.a*p.y);
}


///////////
// Circle //
///////////

struct circle{
    point c; T r;
    circle() : c(0, 0), r(0){}
    circle(const point o) : c(o), r(0){}
    circle(const point a, const point b){
        c = (a+b)/2;
        r = norm(a-c);
    }
    circle(const point a, const point b, const point cc){
        assert(ccw(a, b, cc) != 0);
        c = inter_line(bisector(a, b), bisector(b, cc))[0];
        r = norm(a-c);
    }
    bool inside(const point &a) const{
        return norm(a - c) <= r + EPS;
    }
};

pair<point, point> tangent_points(circle cr, point p)
```

```
     {
    ld d1 = norm(p-cr.c), theta = asin(cr.r/d1);
    point p1 = rotccw(cr.c-p, -theta);
    point p2 = rotccw(cr.c-p, theta);
    assert(d1 >= cr.r);
    p1 = p1 * (sqrt(d1*d1-cr.r*cr.r) / d1) + p;
    p2 = p2 * (sqrt(d1*d1-cr.r*cr.r) / d1) + p;
    return {p1, p2};
}


circle incircle(point p1, point p2, point p3){
    ld m1 = norm(p2-p3);
    ld m2 = norm(p1-p3);
    ld m3 = norm(p1-p2);
    point c = (p1*m1 + p2*m2 + p3*m3)*(1/(m1+m2+m3));
    ld s = 0.5*(m1+m2+m3);
    ld r = sqrt(s*(s-m1)*(s-m2)*(s-m3)) / s;
    return circle(c, r);
}

circle circumcircle(point a, point b, point c) {
    circle ans;
    point u = point((b-a).y, -(b-a).x);
    point v = point((c-a).y, -(c-a).x);
    point n = (c-b)*0.5;
    ld t = (u^n)/(v^u);
    ans.c = ((a+c)*0.5) + (v*t);
    ans.r = norm(ans.c-a);
    return ans;
}

vp inter_circle_line(circle C, line L){
    point ab = L.p2 - L.p1, p = L.p1 + ab * ((C.c-L.p1)*(ab) / (ab*ab));
    ld s = (L.p2-L.p1)^(C.c-L.p1), h2 = C.r*C.r - s*s / (ab*ab);
    if (h2 < -EPS) return {};
    if (eq(h2, 0)) return {p};
    point h = (ab/norm(ab)) * sqrt(h2);
    return {p - h, p + h};
}

vp inter_circle(circle c1, circle c2){
    if (c1.c == c2.c) { assert(c1.r != c2.r); return {}; }
    point vec = c2.c - c1.c;
    ld d2 = vec * vec, sum = c1.r + c2.r, dif = c1.r - c2.r;
    ld p = (d2 + c1.r * c1.r - c2.r * c2.r) / (2 * d2);
    ld h2 = c1.r * c1.r - p * p * d2;
    if (sum * sum < d2 or dif * dif > d2) return {};
    point mid = c1.c + vec * p, per = point(-vec.y, vec.x) * sqrt(fmax(0, h2) / d2);
    if (eq(per.x, 0) and eq(per.y, 0)) return {mid};
    return {mid + per, mid - per};
}

// minimum circle cover O(n) amortizado
circle min_circle_cover(vp v){
    random_shuffle(v.begin(), v.end());
    circle ans;
    int n = v.size();
    for(int i=0;i<n;i++) if(!ans.inside(v[i])){
        ans = circle(v[i]);
        for(int j=0;j<i;j++) if(!ans.inside(v[j])){
            ans = circle(v[i], v[j]);
            for(int k=0;k<j;k++) if(!ans.inside(v[k])){
                ans = circle(v[i], v[j], v[k]);
            }
        }
    }
    return ans;
}
```

6 Grafos

6.1 Dfs Tree

```
int desce[N], sobe[N], vis[N], h[N];
int backedges[N], pai[N];

// backedges[u] = backedges que comecam embaixo de (ou =) u e sobem pra cima de u; backedges[u] == 0 => u eh ponte
void dfs(int u, int p) {
    if(vis[u]) return;
    pai[u] = p;
    h[u] = h[p]+1;
    vis[u] = 1;

    for(auto v : g[u]) {
        if(p == v or vis[v]) continue;
        dfs(v, u);
        backedges[u] += backedges[v];
    }
    for(auto v : g[u]) {
        if(h[v] > h[u]+1)
            desce[u]++;
        else if(h[v] < h[u]-1)
            sobe[u]++;
    }
    backedges[u] += sobe[u] - desce[u];
}
```

6.2 Kosaraju

```
vector<int> g[N], gi[N]; // grafo invertido
int vis[N], comp[N]; // componente conexo de cada vertice
stack<int> S;

void dfs(int u){
    vis[u] = 1;
    for(auto v: g[u]) if(!vis[v]) dfs(v);
    S.push(u);
}

void scc(int u, int c){
    vis[u] = 1; comp[u] = c;
    for(auto v: gi[u]) if(!vis[v]) scc(v, c);
}

void kosaraju(int n){
    for(int i=0;i<n;i++) vis[i] = 0;
    for(int i=0;i<n;i++) if(!vis[i]) dfs(i);
    for(int i=0;i<n;i++) vis[i] = 0;
    while(S.size()){
        int u = S.top();
        S.pop();
        if(!vis[u]) scc(u, u);
    }
}
```

6.3 Topological Sort

```
int n; // number of vertices
vector<vector<int>> adj; // adjacency list of graph
vector<bool> visited;
vector<int> ans;

void dfs(int v) {
    visited[v] = true;
    for (int u : adj[v]) {
        if (!visited[u])
            dfs(u);
    }
    ans.push_back(v);
}

void topological_sort() {
    visited.assign(n, false);
    ans.clear();
    for (int i = 0; i < n; ++i) {
        if (!visited[i]) {
            dfs(i);
        }
    }
    reverse(ans.begin(), ans.end());
}
```

6.4 Dijkstra

```
#define pii pair<int, int>
vector<vector<pii>> g(N);
vector<bool> used(N);
vector<ll> d(N, LLINF);
priority_queue< pii, vector<pii>, greater<pii> > fila;

void dijkstra(int k) {
    d[k] = 0;
    fila.push({0, k});

    while (!fila.empty()) {
        auto [w, u] = fila.top();
        fila.pop();
        if (used[u]) continue;
        used[u] = true;

        for (auto [v, w]: g[u]) {
            if (d[v] > d[u] + w) {
                d[v] = d[u] + w;
                fila.push({d[v], v});
            }
        }
    }
}
```

6.5 Dinic

```
const int N = 300;

struct Dinic {
    struct Edge{
        int from, to; ll flow, cap;
    };
    vector<Edge> edge;

    vector<int> g[N];
    int ne = 0;
    int lvl[N], vis[N], pass;
    int qu[N], px[N], qt;

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

        ll ans = 0;

        for(; px[s] < (int)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.flow >= v.cap)
                continue;           // v.cap - v.flow < lim
            ll tmp = run(v.to, sink,min(minE, v.cap-v.flow));
            v.flow += tmp, rev.flow -= tmp;
            ans += tmp, minE -= tmp;
            if(minE == 0) break;
        }
        return ans;
    }
    bool bfs(int source, int sink) {
        qt = 0;
        qu[qt++] = source;
        lvl[source] = 1;
        vis[source] = ++pass;
        for(int i = 0; i < qt; i++) {
            int u = qu[i];
            px[u] = 0;
            if(u == sink) return true;
            for(auto& ed : g[u]) {
                auto v = edge[ed];
                if(v.flow >= v.cap || vis[v.to] == pass)
                    continue; // v.cap - v.flow < lim
                vis[v.to] = pass;
                lvl[v.to] = lvl[u]+1;
                qu[qt++] = v.to;
            }
        }
        return false;
    }
    ll flow(int source, int sink) {
        reset_flow();
        ll ans = 0;
        //for(lim = (1LL << 62); lim >= 1; lim /= 2)
        while(bfs(source, sink))
            ans += run(source, sink, LLINF);
        return ans;
    }
    void addEdge(int u, int v, ll c, ll rc) {
        Edge e = {u, v, 0, c};
        edge.pb(e);
        g[u].push_back(ne++);

        e = {v, u, 0, rc};
        edge.pb(e);
        g[v].push_back(ne++);
    }
    void reset_flow() {
        for(int i = 0; i < ne; i++)
            edge[i].flow = 0;
        memset(lvl, 0, sizeof(lvl));
        memset(vis, 0, sizeof(vis));
        memset(qu, 0, sizeof(qu));
        memset(px, 0, sizeof(px));
        qt = 0; pass = 0;
    }
    vector<pair<int, int>> cut() {
        vector<pair<int, int>> cuts;
        for (auto [from, to, flow, cap]: edge) {
            if (flow == cap and vis[from] == pass and vis[to] < pass and cap>0) {
                cuts.pb({from, to});
            }
        }
        return cuts;
    }
};
```

6.6 Hungarian

```
// Hungarian Algorithm
//
```

```
// Assignment problem
// Put the edges in the ’a’ matrix (negative or positive)
// assignment() returns a pair with the min assignment,
// and the column choosen by each row
// assignment() - O(n^3)

template<typename T>
struct hungarian {
    int n, m;
    vector<vector<T>> a;
    vector<T> u, v;
    vector<int> p, way;
    T inf;

    hungarian(int n_, int m_) : n(n_), m(m_), u(m+1), v(m+1), p(m+1), way(m+1) {
        a = vector<vector<T>>(n, vector<T>(m));
        inf = numeric_limits<T>::max();
    }
    pair<T, vector<int>> assignment() {
        for (int i = 1; i <= n; i++) {
            p[0] = i;
            int j0 = 0;
            vector<T> minv(m+1, inf);
            vector<int> used(m+1, 0);
            do {
                used[j0] = true;
                int i0 = p[j0], j1 = -1;
                T delta = inf;
                for (int j = 1; j <= m; j++) if (!used[j]) {
                    T cur = a[i0-1][j-1] - u[i0] - v[j];
                    if (cur < minv[j]) minv[j] = cur, way[j] = j0;
                    if (minv[j] < delta) delta = minv[j], j1 = j;
                }
                for (int j = 0; j <= m; j++)
                    if (used[j]) u[p[j]] += delta, v[j] -= delta;
                    else minv[j] -= delta;
                j0 = j1;
            } while (p[j0] != 0);
            do {
                int j1 = way[j0];
                p[j0] = p[j1];
                j0 = j1;
            } while (j0);
        }
        vector<int> ans(m);
        for (int j = 1; j <= n; j++) ans[p[j]-1] = j-1;
        return make_pair(-v[0], ans);
    }
};
```

6.7 Floyd Warshall

```
// Floyd Warshall

int dist[N][N];

for(int k = 1; k <= n; k++)
    for(int i = 1; i <= n; i++)
        for(int j = 1; j <= n; j++)
            dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
```

6.8 Lca

```
const int LOG = 22;
vector<vector<int>> g(N);
int t, n;
vector<int> in(N), height(N);
vector<vector<int>> up(LOG, vector<int>(N));
void dfs(int u, int h=0, int p=-1) {
    up[0][u] = p;
    in[u] = t++;
    height[u] = h;
    for (auto v: g[u]) if (v != p) dfs(v, h+1, u);
}

void blift() {
    up[0][0] = 0;
    for (int j=1;j<LOG;j++) {
        for (int i=0;i<n;i++) {
            up[j][i] = up[j-1][up[j-1][i]];
        }
    }
}

int lca(int u, int v) {
    if (u == v) return u;
    if (in[u] < in[v]) swap(u, v);
    for (int i=LOG-1;i>=0;i--) {
        int u2 = up[i][u];
        if (in[u2] > in[v])
            u = u2;
    }
    return up[0][u];
}

t = 0;
dfs(0);
blift();

// lca O(1)

template<typename T> struct rmq {
    vector<T> v;
    int n; static const int b = 30;
    vector<int> mask, t;

    int op(int x, int y) { return v[x] < v[y] ? x : y; }
    int msb(int x) { return __builtin_clz(1)-__builtin_clz(x); }
    rmq() {}
    rmq(const vector<T>& v_) : v(v_), n(v.size()), mask(n), t(n) {
        for (int i = 0, at = 0; i < n; mask[i++] = at |= 1) {
            at = (at<<1)&((1<<b)-1);
            while (at and op(i, i-msb(at&-at)) == i) at ^= at&-at;
        }
        for (int i = 0; i < n/b; i++) t[i] = b*i+b-1-msb(mask[b*i+b-1]);
        for (int j = 1; (1<<j) <= n/b; j++) for (int i = 0; i+(1<<j) <= n/b; i++)
            t[n/b*j+i] = op(t[n/b*(j-1)+i], t[n/b*(j-1)+i+(1<<(j-1))]);
    }
    int small(int r, int sz = b) { return r-msb(mask[r]&((1<<sz)-1)); }
    T query(int l, int r) {
        if (r-l+1 <= b) return small(r, r-l+1);
        int ans = op(small(l+b-1), small(r));
        int x = l/b+1, y = r/b-1;
        if (x <= y) {
            int j = msb(y-x+1);
            ans = op(ans, op(t[n/b*j+x], t[n/b*j+y-(1<<j)+1]));
```

```cpp
        }
        return ans;
    }
};

namespace lca {
    vector<int> g[N];
    int v[2*N], pos[N], dep[2*N];
    int t;
    rmq<int> RMQ;

    void dfs(int i, int d = 0, int p = -1) {
        v[t] = i, pos[i] = t, dep[t++] = d;
        for (int j : g[i]) if (j != p) {
            dfs(j, d+1, i);
            v[t] = i, dep[t++] = d;
        }
    }
    void build(int n, int root) {
        t = 0;
        dfs(root);
        RMQ = rmq<int>(vector<int>(dep, dep+2*n-1));
    }
    int lca(int a, int b) {
        a = pos[a], b = pos[b];
        return v[RMQ.query(min(a, b), max(a, b))];
    }
    int dist(int a, int b) {
        return dep[pos[a]] + dep[pos[b]] - 2*dep[pos[lca(a, b)]];
    }
}
```

6.9 Kruskal

```cpp
struct DSU {
    int n;
    vector<int> parent, size;

    DSU(int n): n(n) {
        parent.resize(n, 0);
        size.assign(n, 1);

        for(int i=0;i<n;i++)
            parent[i] = i;
    }

    int find(int a) {
        if(a == parent[a]) return a;
        return parent[a] = find(parent[a]);
    }

    void join(int a, int b) {
        a = find(a); b = find(b);
        if(a != b) {
            if(size[a] < size[b]) swap(a, b);
            parent[b] = a;
            size[a] += size[b];
        }
    }
};

struct Edge {
    int u, v, weight;
    bool operator<(Edge const& other) {
        return weight < other.weight;
    }
};

vector<Edge> kruskal(int n, vector<Edge> edges) {
    vector<Edge> mst;
    DSU dsu = DSU(n+1);

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

    for(Edge e : edges) {
        if(dsu.find(e.u) != dsu.find(e.v)) {
            mst.push_back(e);
            dsu.join(e.u,e.v);
        }
    }

    return mst;
}
```

6.10 Ford

```cpp
const int N = 2000010;

struct Ford {
    struct Edge {
        int to, f, c;
    };

    int vis[N];
    vector<int> adj[N];
    vector<Edge> edges;
    int cur = 0;

    void addEdge(int a, int b, int cap, int rcap) {
        Edge e;
        e.to = b; e.c = cap; e.f = 0;
        edges.pb(e);
        adj[a].pb(cur++);

        e = Edge();
        e.to = a; e.c = rcap; e.f = 0;
        edges.pb(e);
        adj[b].pb(cur++);
    }

    int dfs(int s, int t, int f, int tempo) {
        if(s == t)
            return f;
        vis[s] = tempo;

        for(int e : adj[s]) {
            if(vis[edges[e].to] < tempo and (edges[e].c - edges[e].f) > 0) {
                if(int a = dfs(edges[e].to, t, min(f, edges[e].c-edges[e].f) , tempo)) {
                    edges[e].f += a;
                    edges[e^1].f -= a;
                    return a;
                }
            }
        }
        return 0;
    }

    int flow(int s, int t) {
        int mflow = 0, tempo = 1;
        while(int a = dfs(s, t, INF, tempo)) {
            mflow += a;
            tempo++;
        }
        return mflow;
    }
};
```

7 Algoritmos

7.1 Ternary Search

```
// Ternary
ld l = -1e4, r = 1e4;
int iter = 100;
while(iter--){
    ld m1 = (2*l + r) / 3;
    ld m2 = (l + 2*r) / 3;
    if(check(m1) > check(m2))
        l = m1;
    else
        r = m2;
}
```

8 Math

8.1 Totient

```
// phi(p^k) = (p^(k-1))*(p-1) com p primo
// O(sqrt(m))
ll phi(ll m){
    ll res = m;
    for(ll d=2;d*d<=m;d++){
        if(m % d == 0){
            res = (res/d)*(d-1);
            while(m%d == 0)
                m /= d;
        }
    }
    if(m > 1) {
        res /= m;
        res *= (m-1);
    }
    return res;
}

// modificacao do crivo, O(n*log(log(n)))
vector<ll> phi_to_n(ll n){
    vector<bool> isprime(n+1, true);
    vector<ll> tot(n+1);
    tot[0] = 0; tot[1] = 1;
    for(ll i=1;i<=n; i++){
        tot[i] = i;
    }

    for(ll p=2;p<=n;p++){
        if(isprime[p]){
            tot[p] = p-1;
            for(ll i=p+p;i<=n;i+=p){
                isprime[i] = false;
                tot[i] = (tot[i]/p)*(p-1);
            }
        }
    }
    return tot;
}
```

8.2 Inverso Mult

```
// gcd(a, m) = 1 para existir solucao
// ax + my = 1, ou a*x = 1 (mod m)
ll inv(ll a, ll m) { // com gcd
    ll x, y;
    gcd(a, m, x, y);
    return (((x % m) +m) %m);
}

ll inv(ll a, ll phim) { // com phi(m), se m for primo entao phi(m) = p-1
    ll e = phim-1;
    return fexp(a, e);
}
```

8.3 Miller Habin

```
ll mul(ll a, ll b, ll m) {
    return (a*b-ll(a*(long double)b/m+0.5)*m+m)%m;
}

ll expo(ll a, ll b, ll m) {
    if (!b) return 1;
    ll ans = expo(mul(a, a, m), b/2, m);
    return b%2 ? mul(a, ans, m) : ans;
}

bool prime(ll n) {
    if (n < 2) return 0;
    if (n <= 3) return 1;
    if (n % 2 == 0) return 0;

    ll d = n - 1;
    int r = 0;
    while (d % 2 == 0) {
        r++;
        d /= 2;
    }

    // com esses primos, o teste funciona garantido para n <= 2^64
    // funciona para n <= 3*10^24 com os primos ate 41
    for (int i : {2, 325, 9375, 28178, 450775, 9780504, 795265022}) {
        if (i >= n) break;
        ll x = expo(i, d, n);
        if (x == 1 or x == n - 1) continue;

        bool composto = 1;
        for (int j = 0; j < r - 1; j++) {
            x = mul(x, x, n);
            if (x == n - 1) {
                composto = 0;
                break;
            }
        }
        if (composto) return 0;
    }
    return 1;
}
```

8.4 Matrix Exponentiation

```
struct Matrix {
    vector<vl> m;
    int r, c;

    Matrix(vector<vl> mat) {
        m = mat;
        r = mat.size();
        c = mat[0].size();
    }

    Matrix(int row, int col, bool ident=false) {
        r = row; c = col;
        m = vector<vl>(r, vl(c, 0));
        if(ident) {
            for(int i = 0; i < min(r, c); i++) {
                m[i][i] = 1;
            }
        }
    }

    Matrix operator*(const Matrix &o) const {
        assert(c == o.r); // garantir que da pra multiplicar
        vector<vl> res(r, vl(o.c, 0));

        for(int i = 0; i < r; i++) {
            for(int k = 0; k < c; k++) {
                for(int j = 0; j < o.c; j++) {
                    res[i][j] = (res[i][j] + m[i][k]*o.m[k][j]) % MOD;
                }
            }
        }

        return Matrix(res);
    }
};

Matrix fexp(Matrix b, int e, int n) {
    if(e == 0) return Matrix(n, n, true); // identidade
    Matrix res = fexp(b, e/2, n);
    res = (res * res);
    if(e%2) res = (res * b);

    return res;
}
```

8.5 Division Trick

```
for(int l = 1, r; l <= n; l = r + 1) {
    r = n / (n / l);
    // n / i has the same value for l <= i <= r
}
```

8.6 Crivo

```
vi p(N, 0);
p[0] = p[1] = 1;
for(ll i=4; i<N; i+=2) p[i] = 2;
for(ll i=3; i<N; i+=2)
    if(!p[i])
        for(ll j=i*i; j<N; j+=2*i)
            p[j] = i;
```

8.7 Bigmod

```
ll mod(string a, ll p) {
    ll res = 0, b = 1;
    reverse(all(a));

    for(auto c : a) {
        ll tmp = (((ll)c-'0')*b) % p;
        res = (res + tmp) % p;

        b = (b * 10) % p;
    }

    return res;
}
```

8.8 Linear Diophantine Equation

```
// Linear Diophantine Equation
array<ll, 3> exgcd(int a, int b) {
    if (a == 0) return {0, 1, b};
    auto [x, y, g] = exgcd(b % a, a);
    return {y - b / a * x , x, g};
}

array<ll, 4> find_any_solution(ll a, ll b, ll c) {
    auto[x, y, g] = exgcd(a, b);
    if (c % g) return {false, 0, 0, 0};
    x *= c / g;
    y *= c / g;
    return {true, x, y, g};
}

//  All solutions
//  x' = x + k*b/g
//  y' = y - k*a/g
```