

Contents

1 Graph Algorithms

1.1	2-SAT	1
1.2	Kosaraju	1
1.3	Tree Isomorphism	2
1.4	LCA	3
1.5	Bridges and Articulation Points	3
1.6	Eulerian Tour	3
1.7	Floyd Warshall	4

2 Strings

2.1	Aho-Corasick	4
2.2	KMP	4
2.3	Suffix Array	4
2.4	Suffix Array 2	4
2.5	Suffix Array Disulguinha	5
2.6	Manacher's Algorithm	6
2.7	Splitting String	6

3 Numerical Algorithms

3.1	Fast Fourier Transform	6
3.2	Fast Fourier Transform 2	6
3.3	Simpson's Algorithm	7
3.4	Matrix Exponentiation	8

4 Mathematics

4.1	Big Number	8
4.2	Big Number 2	8
4.3	Chinese Remainder	11
4.4	Chinese Remainder 2	11
4.5	Matrix Exponentiation	11
4.6	Pascal Triangle	12
4.7	Euler's Totient Function	12
4.8	Pollard Rho	12
4.9	Sieve of Eratosthenes	13
4.10	Extended Euclidean Algorithm	13
4.11	Multiplicative Inverse	13
4.12	Multiplicative Inverse 2	13
4.13	Linear Diophantine Equation	13

5 Combinatorial Optimization

5.1	Dinic	13
5.2	Hopcroft-Karp - Bipartite Matching	13
5.3	Max Bipartite Matching 2	14
5.4	Min Cost Matching	14
5.5	Min Cost Max Flow	15
5.6	Min Cost Max Flow 2	15
5.7	Edmonds Karp	16

6 Dynamic Programming

6.1	Convex Hull Trick	16
6.2	Convex Hull Trick 2	17
6.3	Divide-and-Conquer	17
6.4	LIS - Longest Increasing Subsequence	17

7 Geometry

7.1	Convex Hull - Monotone Chain	18
7.2	Minimum Enclosing Circle	18
7.3	Minimum Enclosing Circle 2	19
7.4	Fast Geometry in Cpp	19

8 Data Structures

8.1	Disjoint Set Union	21
-----	--------------------	----

8.2	Persistent Segment Tree	21
8.3	RMQ of Indices	22
8.4	RSQ with Lazy-Propagation	22
8.5	Segment Tree	22
8.6	Sparse Table	23

1 Graph Algorithms

1.1 2-SAT

```
//RODAR O COMPONENTE FORTEMENTE CONECTADO
//RECUPERAR NA ORDEM DE descarga DA PILHA
//CONFORME O EXEMPLO A SEGUIR
const int N = 510;
vi graph[N], rev[N];
int us[N];
stack<int> pilha;
int resposta[N];
void dfs1(int u)
{
    us[u] = 1;
    for (int v : graph[u])
        if (!us[v]) dfs1(v);
    pilha.push(u);
}
void dfs2(int u, int color)
{
    us[u] = color;
    for (int v : rev[u])
        if (!us[v]) dfs2(v, color);
}
int Sat(int n)
{
    for (int i = 0; i < n; i++)
        if (!us[i]) dfs1(i);
    int color = 1;
    vi r;
    memset(us, 0, sizeof(us));
    while (!pilha.empty()) {
        int topo = pilha.top();
        r.pb(topo);
        pilha.pop();
        if (!us[topo]) dfs2(topo, color++);
    }
    for (int i = 0; i < n; i += 2) {
        if (us[i] == us[i + 1]) return 0;
    }
    memset(resposta, -1, sizeof(resposta));
    for (int i = r.size() - 1; i >= 0; i--) {
        int vert = r[i] / 2;
        int ok = r[i] % 2;
        if (resposta[vert] == -1) resposta[vert] = !ok;
    }
    return 1;
}
inline void add(int u, int v)
{
    graph[u].pb(v);
    rev[v].pb(u);
}
inline int pos(int u) { return 2 * u; }
inline int neg(int u) { return 2 * u + 1; }
```

1.2 Kosaraju

```
//Retorna os componentes fortemente conectados
//Se o usados[i]=usados[j], temos que i e j
//pertencem ao mesmo componente, col-i= numero
//de componentes fortemente conectados do grafo
class kosaraju {
private:
    vi usados;
    vi graph;
    vi trans;
    vi pilha;

public:
    kosaraju(int N)
    {
        graph.resize(N);
```

```

    trans.resize(N);
}
void AddEdge(int u, int v)
{
    graph[u].pb(v);
    trans[v].pb(u);
}
void dfs(int u, int pass, int color)
{
    usados[u] = color;
    vi vizinhos;
    if (pass == 1)
        vizinhos = graph[u];
    else
        vizinhos = trans[u];
    for (int j = 0; j < vizinhos.size(); j++) {
        int v = vizinhos[j];
        if (usados[v] == 0) {
            dfs(v, pass, color);
        }
    }
    pilha.pb(u);
}
int SSC(int n)
{
    pilha.clear();
    usados.assign(n, 0);
    for (int i = 0; i < n; i++) {
        if (!usados[i]) dfs(i, 1, 1);
    }
    usados.assign(n, 0);
    int color = 1;
    for (int i = n - 1; i >= 0; i--) {
        if (usados[pilha[i]] == 0) {
            dfs(pilha[i], 2, color);
            color++;
        }
    }
    return color - 1;
}
};

```

1.3 Tree Isomorphism

```

//Segue no main um exemplo de utilizacao do isomorfismo de arvore!
vvi children, subtreeLabels, tree, L;
vipred, map;
int n;

bool compare(int a, int b) {
    return subtreeLabels[a] < subtreeLabels[b];
}
boolequals(int a, int b) {
    return subtreeLabels[a] == subtreeLabels[b];
}
void generateMapping(int r1, int r2) {
    map.resize(n);
    map[r1] = r2 - n;
    sort(children[r1].begin(), children[r1].end(), compare);
    sort(children[r2].begin(), children[r2].end(), compare);
    for (int i = 0; i < (int) children[r1].size(); i++) {
        int u = children[r1][i];
        int v = children[r2][i];
        generateMapping(u, v);
    }
}
vifindCenter(int offset = 0) {
    int cnt = n;
    vi a;
    vi deg(n);
    for (int i = 0; i < n; i++) {
        deg[i] = tree[i + offset].size();
        if (deg[i] <= 1) {
            a.push_back(i + offset);
            --cnt;
        }
    }
    while (cnt > 0) {
        vi na;
        for (int i = 0; i < (int) a.size(); i++) {
            int u = a[i];
            for (int j = 0; j < (int) tree[u].size(); j++) {
                int v = tree[u][j];
                if (--deg[v - offset] == 1) {
                    na.push_back(v);
                    --cnt;
                }
            }
        }
    }
}

```

```

    }
    a = na;
    return a;
}
int dfs(int u, int p = -1, int depth = 0) {
    L[depth].push_back(u);
    int h = 0;
    for (int i = 0; i < (int) tree[u].size(); i++) {
        int v = tree[u][i];
        if (v == p)
            continue;
        pred[v] = u;
        children[u].push_back(v);
        h = max(h, dfs(v, u, depth + 1));
    }
    return h + 1;
}
bool rootedTreeIsomorphism(int r1, int r2) {
    L.assign(n, vi());
    pred.assign(2 * n, -1);
    children.assign(2 * n, vi());
    int h1 = dfs(r1);
    int h2 = dfs(r2);
    if (h1 != h2)
        return false;
    int h = h1 - 1;
    vi label(2 * n);
    subtreeLabels.assign(2 * n, vi());
    for (int i = h - 1; i >= 0; i--) {
        for (int j = 0; j < (int) L[i + 1].size(); j++) {
            int v = L[i + 1][j];
            subtreeLabels[pred[v]].push_back(label[v]);
        }
        for (int j = 0; j < (int) L[i].size(); j++) {
            int v = L[i][j];
            sort(subtreeLabels[v].begin(), subtreeLabels[v].end());
        }
        sort(L[i].begin(), L[i].end(), compare);
        for (int j = 0, cnt = 0; j < (int) L[i].size(); j++) {
            if (j % % !equals(L[i][j], L[i][j - 1]))
                ++cnt;
            label[L[i][j]] = cnt;
        }
    }
    if (!equals(r1, r2))
        return false;
    generateMapping(r1, r2);
    return true;
}
bool treeIsomorphism() {
    vi c1 = findCenter();
    vi c2 = findCenter(n);
    if (c1.size() == c2.size()) {
        if (rootedTreeIsomorphism(c1[0], c2[0]))
            return true;
        else if (c1.size() > 1)
            return rootedTreeIsomorphism(c1[1], c2[0]);
    }
    return false;
}
int main() {
    n = 5;
    vvi t1(n);
    t1[0].push_back(1);
    t1[1].push_back(0);
    t1[1].push_back(2);
    t1[2].push_back(1);
    t1[1].push_back(3);
    t1[3].push_back(1);
    t1[0].push_back(4);
    t1[4].push_back(0);
    vvi t2(n);
    t2[0].push_back(1);
    t2[1].push_back(0);
    t2[0].push_back(4);
    t2[4].push_back(0);
    t2[4].push_back(3);
    t2[3].push_back(4);
    t2[4].push_back(2);
    t2[2].push_back(4);
    tree.assign(2 * n, vi());
    for (int u = 0; u < n; u++) {
        for (int i = 0; i < t1[u].size(); i++) {
            int v = t1[u][i];
            tree[u].push_back(v);
        }
        for (int i = 0; i < t2[u].size(); i++) {
            int v = t2[u][i];

```

```

        tree[u + n].push_back(v + n);
    }
}
bool res = treeIsomorphism();
cout << res << endl;
if (res)
    for (int i = 0; i < n; i++)
        cout << map[i] << endl;
}

```

1.4 LCA

```

//antes de usar as queries de lca, e etc..
//certifique-se de chamar a dfs da arvore e
//process()
const int N = 100000;
const int M = 22;
int P[N][M];
int big[N][M], low[N][M], level[N];
vii graph[N];
int n;
void dfs(int u, int last, int l)
{
    level[u] = l;
    P[u][0] = last;
    for (ii v : graph[u])
        if (v.first != last) {
            big[v.first][0] = low[v.first][0] = v.second;
            dfs(v.first, u, l + 1);
        }
}
void process()
{
    for (int j = 1; j < M; j++)
        for (int i = 1; i <= n; i++) {
            P[i][j] = P[P[i][j - 1]][j - 1];
            big[i][j] = max(big[i][j - 1], big[P[i][j - 1]][j - 1]);
            low[i][j] = min(low[i][j - 1], low[P[i][j - 1]][j - 1]);
        }
}
int lca(int u, int v)
{
    if (level[u] < level[v]) swap(u, v);
    for (int i = M - 1; i >= 0; i--)
        if (level[u] - (1 << i) >= level[v]) u = P[u][i];
    if (u == v) return u;
    for (int i = M - 1; i >= 0; i--) {
        if (P[u][i] != P[v][i]) u = P[u][i], v = P[v][i];
    }
    return P[u][0];
}
int maximum(int u, int v, int x)
{
    int resp = 0;
    for (int i = M - 1; i >= 0; i--)
        if (level[u] - (1 << i) >= level[x]) {
            resp = max(resp, big[u][i]);
            u = P[u][i];
        }
    for (int i = M - 1; i >= 0; i--)
        if (level[v] - (1 << i) >= level[x]) {
            resp = max(resp, big[v][i]);
            v = P[v][i];
        }
    return resp;
}
int minimum(int u, int v, int x)
{
    int resp = INF;
    for (int i = M - 1; i >= 0; i--)
        if (level[u] - (1 << i) >= level[x]) {
            resp = min(resp, low[u][i]);
            u = P[u][i];
        }
    for (int i = M - 1; i >= 0; i--)
        if (level[v] - (1 << i) >= level[x]) {
            resp = min(resp, low[v][i]);
            v = P[v][i];
        }
    return resp;
}

```

1.5 Bridges and Articulation Points

```

class ponte {
private:
    vvi graph;
    vi usados;
    vi e_articulacao;
    vi dfs_low;
    vi dfs_prof;
    vector<ii> pontes;
    int tempo;

public:
    ponte(int N)
    {
        graph.clear();
        graph.resize(N);
        usados.assign(N, 0);
        dfs_low.assign(N, 0);
        dfs_prof.assign(N, 0);
        e_articulacao.assign(N, 0);
        tempo = 0;
    }
    void AddEdge(int u, int v)
    {
        graph[u].pb(v);
        graph[v].pb(u);
    }
    void dfs(int u, int pai)
    {
        usados[u] = 1;
        int nf = 0;
        dfs_low[u] = dfs_prof[u] = tempo++;
        for (int v : graph[u]) {
            if (!usados[v]) {
                dfs(v, u);
                nf++;
                if (dfs_low[v] >= dfs_prof[u] and pai != -1) e_articulacao[u] = true;
                if (pai == -1 and nf > 1) e_articulacao[u] = true;
                if (dfs_low[v] > dfs_prof[u]) pontes.pb(mp(u, v));
                dfs_low[u] = min(dfs_low[u], dfs_low[v]);
            }
            else if (v != pai)
                dfs_low[u] = min(dfs_low[u], dfs_prof[v]);
        }
    }
    void olha_as_pontes()
    {
        for (int i = 0; i < graph.size(); i++)
            if (!usados[i]) dfs(i, -1);
        if (pontes.size() == 0)
            cout << " Que merda! nao tem ponte!" << endl;
        else {
            for (ii i : pontes) cout << i.first << " " << i.second << endl;
        }
    }
    void olha_as_art()
    {
        for (int i = 0; i < graph.size(); i++)
            if (!usados[i]) dfs(i, -1);
        for (int i = 0; i < e_articulacao.size(); i++)
            if (e_articulacao[i]) cout << " OIAAA A PONTE " << i << endl;
    }
}

```

1.6 Eulerian Tour

```

multiset<int> graph[N];
stack<int> path;
// -> It suffices to call dfs1 just
// one time leaving from node 0.
// -> To calculate the path,
// call the dfs from the odd degree node.
// -> O(n + log(n))
void dfs1(int u)
{
    while(graph[u].size())
    {
        int v = *graph[u].begin();
        graph[u].erase(graph[u].begin());
        graph[v].erase(graph[v].find(u));
        dfs1(v);
    }
    path.push(u);
}

```

1.7 Floyd Warshall

```
//menor caminho para todos os vertices
for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        if (graph[i][j] != INF) pai[i][j] = i;

for (int k = 0; k < n; k++) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (graph[i][j] > graph[i][k] + graph[k][j]) {
                graph[i][j] = graph[i][k] + graph[k][j];
                pai[i][j] = pai[k][j];
            }
        }
    }
}
```

2 Strings

2.1 Aho-Corasick

```
//N= tamanho da trie, M tamanho do alfabeto
int to[N][M], Link[N], fim[N];
int idx = 1;
void add_str(string &s)
{
    int v = 0;
    for (int i = 0; i < s.size(); i++) {
        if (!to[v][s[i]]) to[v][s[i]] = idx++;
        v = to[v][s[i]];
    }
    fim[v] = 1;
}
void process()
{
    queue<int> fila;
    fila.push(0);
    while (!fila.empty()) {
        int cur = fila.front();
        fila.pop();
        int l = Link[cur];
        fim[cur] |= fim[l];
        for (int i = 0; i < 200; i++) {
            if (to[cur][i]) {
                if (cur != 0) {
                    Link[to[cur][i]] = to[l][i];
                }
                else
                    Link[to[cur][i]] = 0;
                fila.push(to[cur][i]);
            }
            else {
                to[cur][i] = to[l][i];
            }
        }
    }
}
int resolve(string &s)
{
    int v = 0, r = 0;
    for (int i = 0; i < s.size(); i++) {
        v = to[v][s[i]];
        if (fim[v]) r++, v = 0;
    }
    return r;
}
```

2.2 KMP

```
int b[100000];
int szet, sizep;
void kmpPreprocess(string &text, string &pattern)
{
    int i = 0, j = -1;
    b[0] = -1;
    while (i < sizep) {
```

```
        while (j >= 0 and pattern[i] != pattern[j]) j = b[j];
        i++, j++;
        b[i] = j;
    }
}

void kmpSearch(string &text, string &pattern)
{
    kmpPreprocess(text, pattern);
    int i = 0, j = 0;
    while (i < szet) {
        while (j >= 0 and text[i] != pattern[j]) j = b[j];
        i++, j++;
        if (j == sizep) {
            cout << "Olha a substring do texto " << i - j << endl;
            j = b[j];
        }
    }
}
```

2.3 Suffix Array

```
/*
 * O(nlog^2(n)) para o sufix array
 * O(logn) para o LCP(i,j)
 * LCP de i para j;
 */
struct SA {
    const int L;
    string s;
    vvi P;
    vector<pair< ii,int> > M;

    SA(const string &s) : L(s.size()), s(s), P(1, vi(L, 0)), M(L) {
        for (int i = 0; i < L; i++) P[0][i] = s[i] - 'a';
        for (int skip = 1, level = 1; skip < L; skip *= 2, level++) {
            P.pb(vi(L, 0));
            for (int i = 0; i < L; i++)
                M[i] = mp(mp(P[level-1][i], i + skip < L ? P[level-1][i + skip] : -1000), i);
            sort(M.begin(), M.end());
            for (int i = 0; i < L; i++)
                P[level][M[i].second] = (i > 0 && M[i].first == M[i-1].first) ? P[level][M[i-1].second] : i;
        }
    }

    vi GetSA() {
        vi v=P.back();
        vi ret(v.size());
        for (int i=0;i<v.size();i++){
            ret[v[i]]=i;
        }
        return ret;
    }

    int LCP(int i, int j) {
        int len = 0;
        if (i == j) return L - i;
        for (int k = P.size() - 1; k >= 0 && i < L && j < L; k--) {
            if (P[k][i] == P[k][j]) {
                i += 1 << k;
                j += 1 << k;
                len += 1 << k;
            }
        }
        return len;
    }

    vi GetLCP(vi &sa)
    {
        vi lcp(sa.size()-1);
        for (int i=0;i<sa.size()-1;i++){
            lcp[i]=LCP(sa[i],sa[i+1]);
        }
        return lcp;
    }
};
```

2.4 Suffix Array 2

```
/******
 * Suffix Array. Building works in O(NlogN).
 * Also LCP array is calculated in O(NlogN).
 * This code counts number of different substrings in the string.
 * Based on problem I from here: http://codeforces.ru/gym/100133
 *****/
```

```

#include <iostream>
#include <fstream>
#include <cmath>
#include <algorithm>
#include <vector>
#include <set>
#include <map>
#include <stack>
#include <queue>
#include <cstdlib>
#include <stdio.h>
#include <string>
#include <cstring>
#include <cassert>
#include <utility>
#include <iomanip>

using namespace std;

const int MAXN = 205000;
const int ALPH = 256;
const int MAXLOG = 20;

int n;
char s[MAXN];
int p[MAXN]; // suffix array itself
int pcur[MAXN];
int c[MAXN][MAXLOG];
int num[MAXN];
int classesNum;
int lcp[MAXN];

void buildSuffixArray() {
    n++;

    for (int i = 0; i < n; i++)
        num[s[i]]++;

    for (int i = 1; i < ALPH; i++)
        num[i] += num[i - 1];

    for (int i = 0; i < n; i++) {
        p[num[s[i]] - 1] = i;
        num[s[i]]--;
    }

    c[p[0]][0] = 1;
    classesNum = 1;
    for (int i = 1; i < n; i++) {
        if (s[p[i]] != s[p[i - 1]])
            classesNum++;
        c[p[i]][0] = classesNum;
    }

    for (int i = 1; ; i++) {
        int half = (1 << (i - 1));

        for (int j = 0; j < n; j++) {
            pcur[j] = p[j] - half;
            if (pcur[j] < 0)
                pcur[j] += n;
        }

        for (int j = 1; j <= classesNum; j++)
            num[j] = 0;

        for (int j = 0; j < n; j++)
            num[c[pcur[j]][i - 1]]++;
        for (int j = 2; j <= classesNum; j++)
            num[j] += num[j - 1];

        for (int j = n - 1; j >= 0; j--) {
            p[num[c[pcur[j]][i - 1]] - 1] = pcur[j];
            num[c[pcur[j]][i - 1]]--;
        }

        c[p[0]][i] = 1;
        classesNum = 1;

        for (int j = 1; j < n; j++) {
            int p1 = (p[j] + half) % n, p2 = (p[j - 1] + half) % n;
            if (c[p[j]][i - 1] != c[p[j - 1]][i - 1] || c[p1][i - 1] != c[p2][i - 1])
                classesNum++;
            c[p[j]][i] = classesNum;
        }

        if ((1 << i) >= n)
            break;
    }
}

```

```

    for (int i = 0; i < n; i++)
        p[i] = p[i + 1];
    n--;
}

int getLcp(int a, int b) {
    int res = 0;
    for (int i = MAXLOG - 1; i >= 0; i--) {
        int curlen = (1 << i);
        if (curlen > n)
            continue;
        if (c[a][i] == c[b][i]) {
            res += curlen;
            a += curlen;
            b += curlen;
        }
    }
    return res;
}

void calcLcpArray() {
    for (int i = 0; i < n - 1; i++)
        lcp[i] = getLcp(p[i], p[i + 1]);
}

int main() {
    assert(freopen("substr.in", "r", stdin));
    assert(freopen("substr.out", "w", stdout));

    gets(s);
    n = strlen(s);

    buildSuffixArray();

    // Now p from 0 to n - 1 contains suffix array of original string

    /*for (int i = 0; i < n; i++) {
        printf("%d ", p[i] + 1);
    }*/

    calcLcpArray();

    long long ans = 0;
    for (int i = 0; i < n; i++)
        ans += n - p[i];
    for (int i = 1; i < n; i++)
        ans -= lcp[i - 1];

    cout << ans << endl;

    return 0;
}

```

2.5 Suffix Array Disulguinha

```

#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include <cstring>

using namespace std;

struct SuffixArray{
    const string& s;
    int n;
    vector<int> order, rank, lcp;
    vector<int> count, x, y;

    SuffixArray(const string& s) : s(s), n(s.size()), order(n), rank(n),
        count(n + 1), x(n), y(n), lcp(n) {

        build();
        buildLCP();
    }

    void build() {
        //sort suffixes by the first character
        for(int i = 0; i < n; i++) order[i] = i;
        sort(order.begin(), order.end(), [&](int a, int b){return s[a] < s[b];});
        rank[order[0]] = 0;
        for(int i = 1; i < n; i++){
            rank[order[i]] = rank[order[i - 1]];
            if(s[order[i]] != s[order[i - 1]]) rank[order[i]]++;
        }
    }
}

```

```
//sort suffixes by the the first 2*p characters, for p in 1, 2, 4, 8,...
for(int p = 1; p < n; rank[order[n - 1]] < n - 1; p <= 1){
    for(int i = 0; i < n; i++){
        x[i] = rank[i];
        y[i] = i + p < n ? rank[i + p] + 1 : 0;
    }

    radixPass(y);
    radixPass(x);

    rank[order[0]] = 0;
    for(int i = 1; i < n; i++){
        rank[order[i]] = rank[order[i - 1]];
        if(x[order[i]] != x[order[i - 1]] or y[order[i]] != y[order[i - 1]]) rank[order[i]]++;
    }
}

//Stable counting sort
void radixPass(vector<int>& key){
    fill(count.begin(), count.end(), 0);
    for(auto index : order) count[key[index]]++;
    for(int i = 1; i <= n; i++) count[i] += count[i - 1];
    for(int i = n - 1; i >= 0; i--) lcp[--count[key[order[i]]]] = order[i];
    order.swap(lcp);
}

//Kasai's algorithm to build the LCP array from order, rank and s
//For i >= 1, lcp[i] refers to the suffixes starting at order[i] and order[i - 1]
void buildLCP(){
    lcp[0] = 0;
    int k = 0;
    for(int i = 0; i < n; i++){
        if(rank[i] == n - 1){
            k = 0;
        }else{
            int j = order[rank[i] + 1];
            while(i + k < n and j + k < n and s[i + k] == s[j + k]) k++;
            lcp[rank[j]] = k;
            if(k) k--;
        }
    }
}

int main(){
    ios::sync_with_stdio(false);
    string s;
    cin >> s;
    SuffixArray sa(s);
    for(int i = 0; i < s.size(); i++) cout << sa.order[i] << '\n';
}
```

2.6 Manacher's Algorithm

```
/******
Manacher's algorithm for finding all subpalindromes in the string.
Based on problem L from here: http://codeforces.ru/gym/100133
******/

const int MAXN = 105000;

string s;
int n;
int odd[MAXN], even[MAXN];
int l, r;
long long ans;

int main() {
    assert(freopen("palindrome.in", "r", stdin));
    assert(freopen("palindrome.out", "w", stdout));

    getline(cin, s);
    n = (int) s.length();

    // Odd case
    l = r = -1;
    for (int i = 0; i < n; i++) {
        int cur = 1;
        if (i < r)
            cur = min(r - i + 1, odd[l + r - i]);
        while (i + cur < n && i - cur >= 0 && s[i - cur] == s[i + cur])
            cur++;
        odd[i] = cur;
        if (i + cur - 1 > r) {
            l = i - cur + 1;

```

```

            r = i + cur - 1;
        }
    }

    // Even case
    l = r = -1;
    for (int i = 0; i < n; i++) {
        int cur = 0;
        if (i < r)
            cur = min(r - i + 1, even[l + r - i + 1]);
        while (i + cur < n && i - 1 - cur >= 0 && s[i - 1 - cur] == s[i + cur])
            cur++;
        even[i] = cur;
        if (i + cur - 1 > r) {
            l = i - cur;
            r = i + cur - 1;
        }
    }

    for (int i = 0; i < n; i++) {
        if (odd[i] > 1) {
            ans += odd[i] - 1;
        }
        if (even[i])
            ans += even[i];
    }

    cout << ans << endl;

    return 0;
}
```

2.7 Splitting String

```
/* String s to be splitted and the delimiter used to split it.*/
vector<string> splitstr(string s, string delimiter)
{
    vector<string> result;
    string str = s, token;
    size_t pos=0;
    while((pos=str.find(delimiter)) != std::string::npos)
    {
        token = str.substr(0, pos);
        result.push_back(token);
        str.erase(0, pos+delimiter.length());
    }
    result.push_back(str);
    return result;
}
```

3 Numerical Algorithms

3.1 Fast Fourier Transform

```
// FFT - The Iterative Version
//
// Running Time:
// O(n*log n)
//
// How To Use:
// fft(a,1)
// fft(b,1)
// mul(a,b)
// fft(a,-1)
//
// INPUT:
// - fft method:
//   * The vector representing the polynomial
//   * 1 to normal transform
//   * -1 to inverse transform
// - mul method:
//   * The two polynomials to be multiplied
//
// OUTPUT:
// - fft method: Transforms the vector sent.
// - mul method: The result is kept in the first vector.
//
// NOTES:
// - You can either use the struct defined or define difcil as complex<double>
//
```

3.2 Fast Fourier Transform 2

```
// SOLVED:
// * Codeforces Round #296 (Div. 1) D. Fuzzy Search
//

struct dificil {
    double real;
    double im;
    dificil() {
        real=0.0;
        im=0.0;
    }
    dificil(double real, double im):real(real),im(im){}

    dificil operator+(const dificil &o) const {
        return dificil(o.real+real, im+o.im);
    }

    dificil operator/(double v) const {
        return dificil(real/v, im/v);
    }

    dificil operator*(const dificil &o) const {
        return dificil(real*o.real-im*o.im, real*o.im+o.real);
    }

    dificil operator-(const dificil &o) const {
        return dificil(real-o.real, im-o.im);
    }
};

dificil tmp[MAXN*2];
int coco,maiorpot2[MAXN];

void fft(vector<dificil> &A, int s)
{
    int n = A.size(), p = 0;
    while(n>1){
        p++;
        n >>= 1;
    }
    n = (1<<p);
    vector<dificil> a=A;

    for(int i = 0; i < n; ++i){
        int rev = 0;
        for(int j = 0; j < p; ++j){
            rev <<= 1;
            rev |= ( (i >> j) & 1 );
        }
        A[i] = a[rev];
    }

    dificil w, wn;

    for(int i = 1; i <= p; ++i){
        int M = 1 << i;
        int K = M >> 1;
        wn = dificil(cos(s*2.0*pi/(double)M), sin(s*2.0*pi/(double)M) );
        for(int j = 0; j < n; j += M){
            w = dificil(1.0, 0.0);
            for(int l = j; l < K + j; ++l){
                dificil t = w;
                t = t*A[l + K];
                dificil u = A[l];
                A[l] = A[l] + t;
                u = u-t;
                A[l + K] = u;
                w = wn*w;
            }
        }

        if(s== -1){
            for(int i = 0; i<n; ++i)
                A[i] = A[i]/(double)n;
        }
    }
}

void mul(vector<dificil> &a, vector<dificil> &b)
{
    for(int i=0;i<a.size();i++)
    {
        a[i]=a[i]*b[i];
    }
}
```

```
// FFT - The Recursive Version
//
// Running Time:
// O(n*log n)
//
// How To Use:
// fft(&a[0], tam, 0)
// fft(&b[0], tam, 0)
// mul(a,b)
// fft(&a[0], tam, 1)
//
// INPUT:
// - fft method:
//   * The vector representing the polynomial
//   * 0 to normal transform
//   * 1 to inverse transform
// - mul method:
//   * The two polynomials to be multiplied
//
// OUTPUT:
// - fft method: Transforms the vector sent.
// - mul method: The result is kept in the first vector.
//
// NOTES:
// - Tam has to be a power of 2.
// - You can either use the struct defined or define dificil as complex<double>
//
// SOLVED:
// * Codeforces Round #296 (Div. 1) D. Fuzzy Search
//

dificil tmp[MAXN*2];
int coco, maiorpot2[MAXN];

void fft(dificil *v, int N, bool inv)
{
    if(N<=1) return;
    dificil *vodd = v;
    dificil *veven = v+N/2;
    for(int i=0; i<N; i++) tmp[i] = v[i];
    coco = 0;
    for(int i=0; i<N; i+=2)
    {
        veven[coco] = tmp[i];
        vodd[coco] = tmp[i+1];
        coco++;
    }
    fft(&vodd[0], N/2, inv);
    fft(&veven[0], N/2, inv);

    dificil w(1);
    double angucomleite = 2.0*PI/(double)N;
    if(inv) angucomleite = -angucomleite;

    dificil wn(cos(angucomleite),sin(angucomleite));
    for(int i=0;i<N/2;i++)
    {
        tmp[i] = veven[i]+w*vodd[i];
        tmp[i+N/2] = veven[i]-w*vodd[i];
        w *= wn;
        if(inv)
        {
            tmp[i] /= 2;
            tmp[i+N/2] /= 2;
        }
    }
    for(int i=0; i<N; i++) v[i] = tmp[i];
}

void mul(vector<dificil> &a, vector<dificil> &b)
{
    for(int i=0; i<a.size(); i++)
    {
        a[i] = a[i]*b[i];
    }
}

void precomp()
{
    int pot=0;
    for(int i=1;i<MAXN;i++)
    {
        if((1<<pot)<i) pot++;
        maiorpot2[i] = (1<<pot);
    }
}
```

3.3 Simpson's Algorithm

```
const int NPASSOS = 100000;
const int W=1000000;
//W= tamanho do intervalo que eu estou integrando
double integrall()
{
    double h = W / (NPASSOS);
    double a = 0;
    double b = W;
    double s = f(a) + f(b);
    for (double i = 1; i <= NPASSOS; i += 2) s += f(a + i * h) * 4.0;
    for (double i = 2; i <= (NPASSOS - 1); i += 2) s += f(a + i * h) * 2.0;
    return s * h / 3.0;
}
```

3.4 Matrix Exponentiation

```
//matmul multiplica m1 por m2
//matpow exponencia a matrix m1 por p
//mul vet multiplica a matrix m1 pelo vetor vet
vvi matmul(vvi &m1, vvi &m2)
{
    vvi ans;
    ans.resize(m1.size(), vi(m2.size(), 0));
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            for (int k = 0; k < n; k++) {
                ans[i][j] += m1[i][k] * m2[k][j];
                ans[i][j] %= MOD;
            }
    return ans;
}

vvi matpow(vvi &m1, ll p)
{
    vvi ans;
    ans.resize(m1.size(), vi(m1.size(), 0));
    for (int i = 0; i < n; i++) ans[i][i] = 1;
    while (p) {
        if (p & 1) ans = matmul(ans, m1);
        m1 = matmul(m1, m1);
        p >>= 1;
    }
    return ans;
}

// VETOR TEM N LINHAS E A MATRIZ E QUADRADA
vi mulvet(vvi &m1, vi &vet)
{
    vi ans;
    ans.resize(vet.size(), 0);
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++) {
            ans[i] += (m1[i][j] * vet[j]);
            ans[i] %= MOD;
        }
    return ans;
}
```

4 Mathematics

4.1 Big Number

```
void zero_esq(string &resp)
{
    string retorno = resp;
    reverse(retorno.begin(), retorno.end());
    int i = retorno.size() - 1;
    while (retorno[i] == '0' and i > 0) {
        retorno.erase(i);
        i--;
    }
    reverse(retorno.begin(), retorno.end());
    resp = retorno;
}

string sum_big(string a, string b)
```

```
{
    string resp;
    reverse(a.begin(), a.end());
    reverse(b.begin(), b.end());
    if (a.size() <= b.size()) {
        int carry = 0;
        for (int i = 0; i < a.size(); i++) {
            int x = b[i] - '0' + a[i] - '0' + carry;
            resp.push_back((char)(x % 10 + '0'));
            carry = x / 10;
        }
        for (int i = a.size(); i < b.size(); i++) {
            int x = b[i] - '0' + carry;
            resp.push_back((char)(x % 10 + '0'));
            carry = x / 10;
        }
        if (carry > 0) resp.push_back((char)(carry + '0'));
    }
    else {
        int carry = 0;
        for (int i = 0; i < b.size(); i++) {
            int x = a[i] - '0' + b[i] - '0' + carry;
            resp.push_back((char)(x % 10 + '0'));
            carry = x / 10;
        }
        for (int i = b.size(); i < a.size(); i++) {
            int x = a[i] - '0' + carry;
            resp.push_back((char)(x % 10 + '0'));
            carry = x / 10;
        }
        if (carry > 0) resp.push_back((char)(carry + '0'));
    }
    reverse(resp.begin(), resp.end());
    zero_esq(resp);
    return resp;
}

string mul_big(string a, string b)
{
    string resp;
    resp.push_back('0');
    string temp;
    int carry = 0;
    reverse(a.begin(), a.end());
    reverse(b.begin(), b.end());
    for (int i = 0; i < a.size(); i++) {
        temp.clear();
        for (int k = 0; k < i; k++) temp.push_back('0');
        int x = a[i] - '0';
        for (int j = 0; j < b.size(); j++) {
            int y = b[j] - '0';
            int novo = (x * y + carry);
            temp.push_back((novo % 10) + '0');
            carry = novo / 10;
        }
        if (carry > 0) temp.push_back(carry + '0');
        reverse(temp.begin(), temp.end());
        carry = 0;
        resp = sum_big(temp, resp);
    }
    zero_esq(resp);
    return resp;
}
```

4.2 Big Number 2

```
/*
*****
Structure implementing long arithmetic in C++
Analogue to BigInteger in Java.
Tested on many problems.
TODO: list some problems
*****
*/

struct BigInt {
    vector<int> num;

    static const int base = 1000 * 1000 * 1000;
    static const int baseDigits = 9;
    string leadingZerosModifier;

    /*
    *****
    * CONSTRUCTORS & SETTERS
    *****
    */

    void setLeadingZerosModifier() {
        leadingZerosModifier = "%0xd";
        leadingZerosModifier[2] = '0' + baseDigits;
    }
}
```



```

}

void set(int value) {
    num.clear();
    if (value == 0)
        num.push_back(0);
    while (value) {
        num.push_back(value % base);
        value /= base;
    }
}

void set(long long value) {
    num.clear();
    if (value == 0)
        num.push_back(0);
    while (value) {
        num.push_back(value % base);
        value /= base;
    }
}

void set(string &value) {
    num.clear();
    for (int i = (int)value.length() - 1; i >= 0; i -= baseDigits) {
        int add = 0;
        for (int j = max(0, i - baseDigits + 1); j <= i; j++)
            add = add * 10 + value[j] - '0';
        num.push_back(add);
    }
}

void operator = (int value) {
    set(value);
}

void operator = (long long value) {
    set(value);
}

void operator = (string &value) {
    set(value);
}

BigInt() {
    setLeadingZerosModifier();
    set(0);
}

BigInt(int value) {
    setLeadingZerosModifier();
    set(value);
}

BigInt(string value) {
    setLeadingZerosModifier();
    set(value);
}

/*=====
 * SIZE METHODS
=====*/

//returns size of vector
int size() {
    return (int)num.size();
}

//returns length of the number
int digitNum() {
    int result = 0;
    for (int i = 0; i < (int)num.size() - 1; i++)
        result += baseDigits;
    int lastNum = num.back();
    while (lastNum) {
        result++;
        lastNum /= 10;
    }
    return result;
}

/*=====
 * I/O
=====*/

void read() {
    string s;
    cin >> s;
    num.clear();
    for (int i = (int)s.length() - 1; i >= 0; i -= baseDigits) {
        int add = 0;
        for (int j = max(0, i - baseDigits + 1); j <= i; j++)

```

```

        add = add * 10 + s[j] - '0';
        num.push_back(add);
    }
}

void print() {
    printf("%d", num.back());
    for (int i = (int)num.size() - 2; i >= 0; i--)
        printf(leadingZerosModifier.c_str(), num[i]);
}

void println() {
    print();
    printf("\n");
}

string toString() {
    string result = "";
    for (int i = 0; i < (int)num.size(); i++) {
        int cur = num[i];
        for (int j = 1; j <= baseDigits; j++) {
            if (cur == 0 && i == (int) num.size() - 1)
                break;
            result.append(1, (char) '0' + cur % 10);
            cur /= 10;
        }
        reverse(result.begin(), result.end());
        return result;
    }
}

/*=====
 * ADDITION
=====*/

void sumThis(BigInt number) {
    int carry = 0;
    for (int i = 0; i < max((int)num.size(), number.size()) || carry; i++) {
        if (i == num.size())
            num.push_back(0);
        if (i >= number.size())
            carry = num[i] + carry;
        else
            carry = num[i] + carry + number.num[i];
        num[i] = carry % base;
        carry /= base;
    }
}

void sumThis(int number) {
    int carry = number;
    for (int i = 0; i < (int)num.size() || carry; i++) {
        if (i == num.size())
            num.push_back(0);
        carry = num[i] + carry;
        num[i] = carry % base;
        carry /= base;
    }
}

BigInt sum(BigInt number) {
    BigInt result = *this;
    result.sumThis(number);
    return result;
}

BigInt sum(int number) {
    BigInt result = *this;
    result.sumThis(number);
    return result;
}

void operator += (BigInt number) {
    sumThis(number);
}

void operator += (int number) {
    sumThis(number);
}

BigInt operator + (BigInt number) {
    return sum(number);
}

BigInt operator + (int number) {
    return sum(number);
}

/*=====
 * SUBTRACTION
=====*/

```

```

void subThis(BigInt number) {
    int carry = 0;
    for (int i = 0; i < (int)number.size() || carry; i++) {
        if (i < (int)number.size())
            num[i] -= carry + number.num[i];
        else
            num[i] -= carry;
        if (num[i] < 0) {
            carry = 1;
            num[i] += base;
        }
        else
            carry = 0;
    }
    while (num.size() > 1 && num.back() == 0)
        num.pop_back();
}

void subThis(int number) {
    int carry = -number;
    for (int i = 0; carry > 0; i++) {
        num[i] -= carry;
        if (num[i] < 0) {
            carry = 1;
            num[i] += base;
        }
        else
            carry = 0;
    }
    while (num.size() > 1 && num.back() == 0)
        num.pop_back();
}

BigInt sub(BigInt number) {
    BigInt result = *this;
    result.subThis(number);
    return result;
}

BigInt sub(int number) {
    BigInt result = *this;
    result.subThis(number);
    return result;
}

void operator -= (BigInt number) {
    subThis(number);
}

void operator -= (int number) {
    subThis(number);
}

BigInt operator - (BigInt number) {
    return sub(number);
}

BigInt operator - (int number) {
    return sub(number);
}

/*=====
 * MULTIPLICATION
=====*/

BigInt mult(BigInt number) {
    BigInt product;
    product.num.resize(num.size() + number.size());
    for (int i = 0; i < (int)num.size(); i++) {
        for (int j = 0, carry = 0; j < (int)number.size() || carry; j++) {
            long long cur = product.num[i + j] + num[i] * 11l * (j < (int)number.size() ? number.num[j] : 0) + carry;
            product.num[i + j] = int (cur % base);
            carry = int (cur / base);
        }
    }
    while (product.size() > 1 && product.num.back() == 0)
        product.num.pop_back();
    return product;
}

void multThis(BigInt number) {
    *this = mult(number);
}

void multThis(int number) {
    int carry = 0;
    for (int i = 0; i < (int)num.size() || carry; ++i) {
        if (i == num.size())
            num.push_back(0);
        long long cur = carry + num[i] * 11l * number;
        num[i] = int (cur % base);
        carry = int (cur / base);
    }
}

void multThisByPowerOfTen(int power) {
    int baseNums = power / baseDigits;
    int curLen = (int)num.size();
    num.resize(curLen + baseNums);
    for (int i = num.size() - 1; i >= baseNums; i--) {
        num[i] = num[i - baseNums];
    }
    for (int i = baseNums - 1; i >= 0; i--)
        num[i] = 0;
    power %= baseDigits;
    int multBy = (int)pow(10.0, power);
    multThis(multBy);
}

/*=====
 * DIVISION
=====*/

void divThis(int number) {
    int carry = 0;
    for (int i = (int)num.size() - 1; i >= 0; i--) {
        long long cur = num[i] + carry * 11l * base;
        num[i] = int (cur / number);
        carry = int (cur % number);
    }
    while (num.size() > 1 && num.back() == 0)
        num.pop_back();
}

BigInt div(int number) {
    BigInt result = *this;
    result.divThis(number);
    return result;
}

void operator /= (int number) {
    divThis(number);
}

BigInt operator / (int number) {
    return div(number);
}

void divThisByPowerOfTen(int power) {
    int baseNums = power / baseDigits;
    int curLen = (int)num.size();
    for (int i = 0; i < (int)num.size() - baseNums; i++) {
        num[i] = num[i + baseNums];
    }
    for (int i = 1; i <= baseNums; i++)
        num.pop_back();
    power %= baseDigits;
    int divBy = (int)pow(10.0, power);
    divThis(divBy);
}

/*=====
 * MODULO
=====*/

void modThis(int number) {
    int carry = 0;
    for (int i = (int)num.size() - 1; i >= 0; i--) {
        long long cur = num[i] + carry * 11l * base;
        carry = int (cur / number);
        num[i] = int (cur % number);
    }
}

```

```

    num[i] = int (cur / number);
    carry = int (cur % number);
}
set(carry);
}

BigInt mod(int number) {
    BigInt result = *this;
    result.modThis(number);
    return result;
}

void operator %= (int number) {
    modThis(number);
}

BigInt operator % (int number) {
    return mod(number);
}

/*=====
* COMPARISON
=====*/

//Returns: -1 - this number is less than argument, 0 - equal, 1 - this number is greater
int compareTo(BigInt number) {
    if ((int)num.size() < number.size())
        return -1;
    if ((int)num.size() > number.size())
        return 1;
    for (int i = (int)num.size() - 1; i >= 0; i--) {
        if (num[i] > number.num[i])
            return 1;
        if (num[i] < number.num[i])
            return -1;
    }
    return 0;
}

//Returns: -1 - this number is less than argument, 0 - equal, 1 - this number is greater
int compareTo(int number) {
    if (num.size() > 1 || num[0] > number)
        return 1;
    if (num[0] < number)
        return -1;
    return 0;
}

bool operator < (BigInt number) {
    return compareTo(number) == -1;
}

bool operator < (int number) {
    return compareTo(number) == -1;
}

bool operator <= (BigInt number) {
    return compareTo(number) != 1;
}

bool operator <= (int number) {
    return compareTo(number) != 1;
}

bool operator == (BigInt number) {
    return compareTo(number) == 0;
}

bool operator == (int number) {
    return compareTo(number) == 0;
}

bool operator > (BigInt number) {
    return compareTo(number) == 1;
}

bool operator > (int number) {
    return compareTo(number) == 1;
}

bool operator >= (BigInt number) {
    return compareTo(number) != -1;
}

bool operator >= (int number) {
    return compareTo(number) != -1;
}

bool operator != (BigInt number) {
    return compareTo(number) != 0;
}

```

```

bool operator != (int number) {
    return compareTo(number) != 0;
}
};

```

4.3 Chinese Remainder

```

ll mulmod(ll a, ll b, ll m)
{
    ll ret = 0;
    while (b > 0) {
        if (b % 2 != 0) ret = (ret + a) % m;
        a = (a + a) % m;
        b >>= 1;
    }
    return ret;
}

ll expmod(ll a, ll e, ll m)
{
    ll ret = 1;
    while (e > 0) {
        if (e % 2 != 0) ret = mulmod(ret, a, m);
        a = mulmod(a, a, m);
        e >>= 1;
    }
    return ret;
}

ll invmul(ll a, ll m) { return expmod(a, m - 2, m); }
ll chinese(vector<ll> r, vector<ll> m)
{
    int sz = m.size();
    ll M = 1;
    for (int i = 0; i < sz; i++) {
        M *= m[i];
    }
    ll ret = 0;
    for (int i = 0; i < sz; i++) {
        ret += mulmod(mulmod(M / m[i], r[i], M), invmul(M / m[i], M), M);
        ret = ret % M;
    }
    return ret;
}

```

4.4 Chinese Remainder 2

```

// Chinese remainder theorem (special case): find z such that // z % m1 = r1, z
// % m2 = r2. Here, z is unique modulo M = lcm(m1, m2). // Return (z, M). On
// failure, M = -1;
ii chinese_remainder_theorem(int m1, int r1, int m2, int r2)
{
    int s, t;
    int g = extended_euclid(m1, m2, s, t);
    if (r1 % g != r2 % g) return mp(0, -1);
    return mp(mod(s * r2 * m1 + t * r1 * m2, m1 * m2) / g, m1 * m2 / g);
}

// Chinese remainder theorem: find z such that // z % m[i] =
// r[i] for all i
// .Note that the solution is unique modulo M = lcm_i (m[i]).
// Return(z, M)
// .On // failure, M = -1. Note that we do not require the a[i] s
// to be relatively prime.
ii chinese_remainder_theorem(const vi &m, const vi &r)
{
    ii ret = make_pair(r[0], m[0]);
    for (int i = 1; i < m.size(); i++) {
        ret = chinese_remainder_theorem(ret.second, ret.first, m[i], r[i]);
        if (ret.second == -1) break;
    }
    return ret;
}

```

4.5 Matrix Exponentiation

```

//matmul multiplica m1 por m2
//matpow exponencia a matrix m1 por p
//mul vet multiplica a matrix m1 pelo vetor vet
vvi matmul(vvi &m1, vvi &m2)

```

```

{
    vvi ans;
    ans.resize(m1.size(), vi(m2.size(), 0));
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            for (int k = 0; k < n; k++) {
                ans[i][j] += m1[i][k] * m2[k][j];
                ans[i][j] %= MOD;
            }
    return ans;
}

vvi matpow(vvi &m1, ll p)
{
    vvi ans;
    ans.resize(m1.size(), vi(m1.size(), 0));
    for (int i = 0; i < n; i++) ans[i][i] = 1;
    while (p) {
        if (p & 1) ans = matmul(ans, m1);
        m1 = matmul(m1, m1);
        p >>= 1;
    }
    return ans;
}

// VETOR TEM N LINHAS E A MATRIZ E QUADRADA
vi mulvet(vvi &m1, vi &vet)
{
    vi ans;
    ans.resize(vet.size(), 0);
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++) {
            ans[i] += (m1[i][j] * vet[j]);
            ans[i] %= MOD;
        }
    return ans;
}

```

4.6 Pascal Triangle

```

//Fazer combinacao de N escolhe M
//por meio do triangulo de pascal
//Complexidade: O(m*n)
unsigned long long comb[61][61];
for (int i = 0; i < 61; i++) {
    comb[i][i] = 1;
    comb[i][0] = 1;
}
for (int i = 2; i < 61; i++)
    for (int j = 1; j < i; j++)
        comb[i][j] = comb[i - 1][j] + comb[i - 1][j - 1];

```

4.7 Eulers Totient Function

```

//retorna quantos elementos coprimos
//a N e menores que n existem
int phi(int n)
{
    int result = n;
    for (int i = 2; i * i <= n; ++i)
        if (n % i == 0) {
            while (n % i == 0) n /= i;
            result -= result / i;
        }
    if (n > 1) result -= result / n;
    return result;
}

```

4.8 Pollard Rho

```

ll u;
ll t;
const int tamteste=5;
ll abss(ll v){ return v>=0 ? v : -v;}
ll randerson()
{
    ld pseudo=(ld)rand()/ (ld)RAND_MAX;
    return (ll) (round((ld)range*pseudo))+1LL;
}

ll mulmod(ll a, ll b, ll mod)

```

```

{
    ll ret=0;
    while (b>0)
    {
        if (b%2!=0) ret=(ret+a)%mod;
        a=(a+a)%mod;
        b=b/2LL;
    }
    return ret;
}

ll expmod(ll a, ll e, ll mod)
{
    ll ret=1;
    while (e>0)
    {
        if (e%2!=0) ret=mulmod(ret,a,mod);
        a=mulmod(a,a,mod);
        e=e/2LL;
    }
    return ret;
}

bool jeova(ll a, ll n)
{
    ll x = expmod(a,u,n);
    ll last=x;
    for (int i=0;i<t;i++)
    {
        x=mulmod(x,x,n);
        if (x==1 and last!=1 and last!=(n-1)) return true;
        last=x;
    }
    if (x==1) return false;
    return true;
}

bool isprime(ll n)
{
    u=n-1;
    t=0;
    while (u%2==0)
    {
        t++;
        u/=2LL;
    }
    if (n==2) return true;
    if (n==3) return true;
    if (n%2==0) return false;
    if (n<2) return false;
    for (int i=0;i<tamteste;i++)
    {
        ll v = randerson()% (n-2)+1;
        //cout<<"jeova "<<y<<" "<<n<<endl;
        if (jeova(v,n)) return false;
    }
    return true;
}

ll gcd(ll a, ll b){ return !b ? a : gcd(b,a%b);}

ll calc(ll x, ll n, ll c)
{
    return (mulmod(x,x,n)+c)%n;
}

ll pollard(ll n)
{
    ll d=1;
    ll i=1;
    ll k=1;
    ll x=2;
    ll y=x;
    ll c;
    do
    {
        c=randerson()% n;
    }while (c==0 or (c+2)%n==0);
    while (d!=n)
    {
        if (i==k)
        {
            k+=2LL;
            y=x;
            i=0;
        }
        x=calc(x,n,c);
        i++;
        d=gcd(abss(y-x),n);
        if (d!=1) return d;
    }
}

```

```
vector<ll> getdiv(ll n)
{
    vector<ll> ret;
    if(n==1) return ret;
    if(isprime(n))
    {
        ret.pb(n);
        return ret;
    }
    ll d = pollard(n);
    ret=getdiv(d);
    vector<ll> ret2=getdiv(n/d);
    for(int i=0; i<ret2.size(); i++) ret.pb(ret2[i]);
    return ret;
}
```

4.9 Sieve of Eratosthenes

```
//esse crivo gera MAXN primos
const int MAX = 1e6;
int primes[MAX];
void gen_primes()
{
    int i, j;
    for (i = 2; i*i <= MAX; i++)
        if (primes[i])
            for (j = i; j * i < MAX; j++) primes[i * j] = 0;
}
```

4.10 Extended Euclidean Algorithm

```
//returns g = gcd(a, b);
//finds x,y such that d= ax+by;
int extended_euclid(int a, int b, int &x, int &y)
{
    int xx = y = 0;
    int yy = x = 1;
    while (b) {
        int q = a / b;
        int t = b;
        b = a % b;
        a = t;
        t = xx;
        xx = x - q * xx;
        x = t;
        t = yy;
        yy = y - q * yy;
        y = t;
    }
    return a;
}
```

4.11 Multiplicative Inverse

```
//computes b such that ab = 1(mod n), returns -1 on failure
int mod_inverse(int a, int n)
{
    int x, y;
    int g = extended_euclid(a, n, x, y);
    if (g > 1) return -1;
    return (x+n)%n;
}
```

4.12 Multiplicative Inverse 2

```
//inverso multiplicativo de A % MOD
//certifique de MOD estar definido antes bonito!
//complexidade: O(log(a))
ll mul_inv(ll a)
{
    ll pin0 = MOD, pin = MOD, t, q;
    ll x0 = 0, x1 = 1;
    if (pin == 1) return 1;
    while (a > 1) {
```

```
q = a / pin;
t = pin, pin = a % pin, a = t;
t = x0, x0 = x1 - q * x0, x1 = t;
}
if (x1 < 0) x1 += pin0;
return x1;
}
```

4.13 Linear Diophantine Equation

5 Combinatorial Optimization

5.1 Dinic

```
//grafo bipartido O(Esqrt(v))
//Para recuperar a resposta, e so colocar um bool
//de false na aresta de retorno e fazer uma bfs/dfs
//andando pelos vertices de capacidade =0 e arestas
//que nao sao de retorno
struct Edge {
    int v, rev;
    int cap;
    Edge(int v_, int cap_, int rev_) : v(v_), rev(rev_), cap(cap_) {}
};

struct MaxFlow {
    vector<vector<Edge>> > g;
    vector<int> level;
    queue<int> q;
    int flow, n;

    MaxFlow(int n_) : g(n_), level(n_), n(n_) {}
    void addEdge(int u, int v, int cap)
    {
        if (u == v) return;
        Edge e(v, cap, int(g[v].size()));
        Edge r(u, 0, int(g[u].size()));
        g[u].push_back(e);
        g[v].push_back(r);
    }

    bool buildLevelGraph(int src, int sink)
    {
        fill(level.begin(), level.end(), -1);
        while (not q.empty()) q.pop();
        level[src] = 0;
        q.push(src);
        while (not q.empty()) {
            int u = q.front();
            q.pop();
            for (auto e = g[u].begin(); e != g[u].end(); ++e) {
                if (not e->cap or level[e->v] != -1) continue;
                level[e->v] = level[u] + 1;
                if (e->v == sink) return true;
                q.push(e->v);
            }
        }
        return false;
    }

    int blockingFlow(int u, int sink, int f)
    {
        if (u == sink or not f) return f;
        int fu = f;
        for (auto e = g[u].begin(); e != g[u].end(); ++e) {
            if (not e->cap or level[e->v] != level[u] + 1) continue;
            int mincap = blockingFlow(e->v, sink, min(fu, e->cap));
            if (mincap) {
                g[e->v][e->rev].cap += mincap;
                e->cap -= mincap;
                fu -= mincap;
            }
        }
        if (f == fu) level[u] = -1;
        return f - fu;
    }

    int maxFlow(int src, int sink)
    {

```

```

    flow = 0;
    while (buildLevelGraph(src, sink))
        flow += blockingFlow(src, sink, numeric_limits<int>::max());
    return flow;
}
};

```

5.2 Hopcroft-Karp - Bipartite Matching

```

/* O(V^3)
 * Matching maximo de grafo bipartido de peso 1 nas arestas
 * supondo que o grafo bipartido seja enumerado de 0-n-1
 * chamamos maxMatch(n)
 */
class MaxMatch {
    vi graph[N];
    int match[N], us[N];

public:
    MaxFlow(){};
    void addEdge(int u, int v) { graph[u].pb(v); }
    int dfs(int u)
    {
        if (us[u]) return 0;
        us[u] = 1;
        for (int v : graph[u]) {
            if (match[v] == -1 or (dfs(match[v]))) {
                match[v] = u;
                return 1;
            }
        }
        return 0;
    }
    int maxMatch(int n)
    {
        memset(match, -1, sizeof(match));
        int ret = 0;
        for (int i = 0; i < n; i++) {
            memset(us, 0, sizeof(us));
            ret += dfs(i);
        }
        return ret;
    }
};

```

5.3 Max Bipartite Matching 2

```

// This code performs maximum bipartite matching.
//
// Running time: O(|E| |V|) -- often much faster in practice
//
// INPUT: w[i][j] = edge between row node i and column node j
// OUTPUT: mr[i] = assignment for row node i, -1 if unassigned
//         mc[j] = assignment for column node j, -1 if unassigned
//         function returns number of matches made
#include <vector>
using namespace std;
typedef vector<int> VI;
typedef vector<VI> VVI;
typedef vector<VI> VVI;
bool FindMatch(int i, const VVI &w, VI &mr, VI &mc, VI &seen) {
    for (int j = 0; j < w[i].size(); j++) {
        if (w[i][j] && !seen[j]) {
            seen[j] = true;
            if (mc[j] < 0 || FindMatch(mc[j], w, mr, mc, seen)) {
                mr[i] = j;
                mc[j] = i;
                return true;
            }
        }
    }
    return false;
}
int BipartiteMatching(const VVI &w, VI &mr, VI &mc) {
    mr = VI(w.size(), -1);
    mc = VI(w[0].size(), -1);
    int ct = 0;
    for (int i = 0; i < w.size(); i++) {
        VI seen(w[0].size());
        if (FindMatch(i, w, mr, mc, seen)) ct++;
    }
    return ct;
}

```

5.4 Min Cost Matching

```

////////////////////////////////////
// Min cost bipartite matching via shortest augmenting paths
//
// This is an O(n^3) implementation of a shortest augmenting path
// algorithm for finding min cost perfect matchings in dense
// graphs. In practice, it solves 1000x1000 problems in around 1
// second.
//
// cost[i][j] = cost for pairing left node i with right node j
// Lmate[i] = index of right node that left node i pairs with
// Rmate[j] = index of left node that right node j pairs with
//
// The values in cost[i][j] may be positive or negative. To perform
// maximization, simply negate the cost[][] matrix.
////////////////////////////////////

#include <algorithm>
#include <cmath>
#include <cstdio>
#include <vector>

using namespace std;

typedef vector<double> VVD;
typedef vector<VD> VVD;
typedef vector<int> VI;

double MinCostMatching(const VVD &cost, VI &Lmate, VI &Rmate)
{
    int n = int(cost.size());

    // construct dual feasible solution
    VD u(n);
    VD v(n);
    for (int i = 0; i < n; i++) {
        u[i] = cost[i][0];
        for (int j = 1; j < n; j++) u[i] = min(u[i], cost[i][j]);
    }
    for (int j = 0; j < n; j++) {
        v[j] = cost[0][j] - u[0];
        for (int i = 1; i < n; i++) v[j] = min(v[j], cost[i][j] - u[i]);
    }

    // construct primal solution satisfying complementary slackness
    Lmate = VI(n, -1);
    Rmate = VI(n, -1);
    int mated = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (Rmate[j] != -1) continue;
            if (fabs(cost[i][j] - u[i] - v[j]) < 1e-10) {
                Lmate[i] = j;
                Rmate[j] = i;
                mated++;
                break;
            }
        }
    }

    VD dist(n);
    VI dad(n);
    VI seen(n);

    // repeat until primal solution is feasible
    while (mated < n) {
        // find an unmatched left node
        int s = 0;
        while (Lmate[s] != -1) s++;

        // initialize Dijkstra
        fill(dad.begin(), dad.end(), -1);
        fill(seen.begin(), seen.end(), 0);
        for (int k = 0; k < n; k++) dist[k] = cost[s][k] - u[s] - v[k];

        int j = 0;
        while (true) {
            // find closest
            j = -1;
            for (int k = 0; k < n; k++) {
                if (seen[k]) continue;
                if (j == -1 || dist[k] < dist[j]) j = k;
            }
            seen[j] = 1;

            // termination condition
            if (Rmate[j] == -1) break;

```

```

// relax neighbors
const int i = Rmate[j];
for (int k = 0; k < n; k++) {
    if (seen[k]) continue;
    const double new_dist = dist[j] + cost[i][k] - u[i] - v[k];
    if (dist[k] > new_dist) {
        dist[k] = new_dist;
        dad[k] = j;
    }
}

// update dual variables
for (int k = 0; k < n; k++) {
    if (k == j || !seen[k]) continue;
    const int i = Rmate[k];
    v[k] += dist[k] - dist[j];
    u[i] -= dist[k] - dist[j];
}
u[s] += dist[j];

// augment along path
while (dad[j] >= 0) {
    const int d = dad[j];
    Rmate[j] = Rmate[d];
    Lmate[Rmate[j]] = j;
    j = d;
}
Rmate[j] = s;
Lmate[s] = j;

mated++;
}

double value = 0;
for (int i = 0; i < n; i++) value += cost[i][Lmate[i]];

return value;
}

```

5.5 Min Cost Max Flow

```

int flow[N][N];
vector<pair<int, int>> g[N];

int n, m, k;

inline int ent(int a) { return a * 2; }
inline int out(int a) { return a * 2 + 1; }
inline void addEdge(int a, int b, int custo, int fluxo)
{
    flow[a][b] += fluxo;
    g[a].push_back(make_pair(b, custo));
    g[b].push_back(make_pair(a, -custo));
}

int src = N - 1, tgt = N - 2;
int dis[N], pai[N];

inline int dij()
{
    memset(dis, INF, sizeof dis);
    memset(pai, -1, sizeof pai);
    priority_queue<pair<int, int>> q;
    dis[src] = 0;
    q.push(make_pair(0, src));
    while (!q.empty()) {
        pair<int, int> foo = q.top();
        q.pop();
        int x = foo.second, cost = -foo.first;
        if (dis[x] != cost) continue;
        for (int i = 0; i < g[x].size(); ++i) {
            int y = g[x][i].first, w = g[x][i].second;
            if (flow[x][y] <= 0) continue;
            if (dis[y] > dis[x] + w) {
                dis[y] = dis[x] + w;
                pai[y] = x;
                q.push(make_pair(-dis[y], y));
            }
        }
    }
    return dis[tgt] != INF;
}

int minCost()
{

```

```

int maxFlow = 0;
int minC = 0;
while (dij()) {
    int u = tgt;
    int minFlow = INF;
    while (pai[u] != -1) {
        minFlow = min(minFlow, flow[pai[u]][u]);
        u = pai[u];
    }
    maxFlow += minFlow;
    minC += minFlow * dis[tgt];
    u = tgt;
    while (pai[u] != -1) {
        flow[pai[u]][u] -= minFlow;
        flow[u][pai[u]] += minFlow;
        u = pai[u];
    }
}
if (maxFlow != n * k) minC = -1;
return minC;
}

inline void init()
{
    memset(flow, 0, sizeof flow);
    for (int i = 0; i < N; ++i) {
        g[i].clear();
    }
}

```

5.6 Min Cost Max Flow 2

```

// Implementation of min cost max flow algorithm using adjacency
// matrix (Edmonds and Karp 1972). This implementation keeps track of
// forward and reverse edges separately (so you can set cap[i][j] !=
// cap[j][i]). For a regular max flow, set all edge costs to 0.
//
// Running time: O(|V|^2) cost per augmentation
// max flow: O(|V|^3) augmentations
// min cost max flow: O(|V|^4 * MAX_EDGE_COST) augmentations
//
// INPUT:
// - graph, constructed using AddEdge()
// - source
// - sink
//
// OUTPUT:
// - (maximum flow value, minimum cost value)
// - To obtain the actual flow, look at positive values only.

#include <cmath>
#include <iostream>
#include <vector>

using namespace std;

typedef vector<VI> VVI;
typedef long long LL;
typedef vector<LL> VL;
typedef vector<VL> VVL;
typedef pair<int, int> PII;
typedef vector<PII> VPPI;

const LL INF = numeric_limits<LL>::max() / 4;

struct MinCostMaxFlow {
    int N;
    VVL cap, flow, cost;
    VI found;
    VL dist, pi, width;
    VPPI dad;

    MinCostMaxFlow(int N): N(N), cap(N, VL(N)), flow(N, VL(N)), cost(N, VL(N)),
        found(N), dist(N), pi(N), width(N), dad(N) {}

    void AddEdge(int from, int to, LL cap, LL cost)
    {
        this->cap[from][to] = cap;
        this->cost[from][to] = cost;
    }

    void Relax(int s, int k, LL cap, LL cost, int dir)
    {
        LL val = dist[s] + pi[s] - pi[k] + cost;
        if (cap && val < dist[k]) {
            dist[k] = val;
            dad[k] = make_pair(s, dir);
            width[k] = min(cap, width[s]);

```

```

    }
}

LL Dijkstra(int s, int t)
{
    fill(found.begin(), found.end(), false);
    fill(dist.begin(), dist.end(), INF);
    fill(width.begin(), width.end(), 0);
    dist[s] = 0;
    width[s] = INF;

    while (s != -1) {
        int best = -1;
        found[s] = true;
        for (int k = 0; k < N; k++) {
            if (found[k]) continue;
            Relax(s, k, cap[s][k] - flow[s][k], cost[s][k], 1);
            Relax(s, k, flow[k][s], -cost[k][s], -1);
            if (best == -1 || dist[k] < dist[best]) best = k;
        }
        s = best;

        for (int k = 0; k < N; k++) pi[k] = min(pi[k] + dist[k], INF);
        return width[t];
    }

    pair<LL, LL> GetMaxFlow(int s, int t)
    {
        LL totflow = 0, totcost = 0;
        while (LL amt = Dijkstra(s, t)) {
            totflow += amt;
            for (int x = t; x != s; x = dad[x].first) {
                if (dad[x].second == 1) {
                    flow[dad[x].first][x] += amt;
                    totcost += amt * cost[dad[x].first][x];
                }
                else {
                    flow[x][dad[x].first] -= amt;
                    totcost -= amt * cost[x][dad[x].first];
                }
            }
        }
        return make_pair(totflow, totcost);
    }
};

```

```

}

void addEdge(int u, int v, ll cap)
{
    Edge A;
    A.init(v, cap, g[v].size());
    Edge B;
    B.init(u, 0, g[u].size());
    g[u].pb(A);
    g[v].pb(B);
}

int maxFlow(int source, int sink)
{
    s = source;
    t = sink;
    mf = 0;
    while (true) {
        f = 0;
        vector<int> dist(g.size(), INF);
        dist[s] = 0;
        queue<int> q;
        q.push(s);
        p.clear();
        p.resize(g.size());
        while (!q.empty()) {
            int u = q.front();
            q.pop();
            if (u == t) break;
            for (int i = 0; i < g[u].size(); i++) {
                Edge prox = g[u][i];
                if (dist[prox.at] == INF and prox.cap > 0) {
                    dist[prox.at] = dist[u] + 1;
                    q.push(prox.at);
                    dad[paizao(u, i, prox.where)];
                    p[prox.at] = paizao;
                }
            }
            augment(t, INF);
            if (f == 0) break;
            mf += f;
        }
        return mf;
    }
};

```

5.7 Edmonds Karp

```

struct Edge {
    int at, where;
    ll cap;
    void init(int _at, ll _cap, int _where)
    {
        at = _at, cap = _cap, where = _where;
    }
};

struct dad {
    int at, up, down;
    dad() { at = -1; }
    dad(int _at, int _up, int _down) { at = _at, up = _up, down = _down; }
};

class MaxFlow {
private:
    vector<vector<Edge>> > g;
    ll mf, f;
    int s, t;
    vector<dad> p;

public:
    void augment(int v, ll minEdge)
    {
        if (v == s) {
            f = minEdge;
            return;
        }
        else if (p[v].at != -1) {
            augment(p[v].at, min(minEdge, g[p[v].at][p[v].up].cap));
            g[p[v].at][p[v].up].cap -= f;
            g[v][p[v].down].cap += f;
        }
    }

    void init(int N)
    {
        for (int i = 0; i < g.size(); i++) g[i].clear();
        mf = 0, f = 0;
        g.resize(N);
    }
};

```

6 Dynamic Programming

6.1 Convex Hull Trick

```

/* Esse convex hull trick e para achar a reta minima!
 * Para maximizar a reta dada , basta trocar o '>' para
 * para '<' na funcao query;
 * Nao chamar query com B ou A vazios! Atualizar dp para
 * depois fazer a query =)
 * ATENCAO COM O DOUBLE!! ESTA EM LONG LONG :)
 */
vi A[N], B[N];
int pont[N];
bool odomeioehlixo(int r1, int r2, int r3, int j)
{
    return (B[j][r1] - B[j][r3]) * (A[j][r2] - A[j][r1]) <
           (B[j][r1] - B[j][r2]) * (A[j][r3] - A[j][r1]);
}

void add(ll a, ll b, int j)
{
    B[j].pb(b);
    A[j].pb(a);
    while (B[j].size() >= 3 and
           odomeioehlixo(B[j].size() - 3, B[j].size() - 2, B[j].size() - 1, j)) {
        B[j].erase(B[j].end() - 2);
        A[j].erase(A[j].end() - 2);
    }
}

ll query(ll x, int j)
{
    if (pont[j] >= B[j].size()) pont[j] = B[j].size() - 1;
    while (pont[j] < B[j].size() - 1 and
           (A[j][pont[j] + 1] * x + B[j][pont[j] + 1] >
            A[j][pont[j]] * x + B[j][pont[j]]))
        pont[j]++;
    return A[j][pont[j]] * x + B[j][pont[j]];
}

/* Testado em :

```



```
* http://www.spoj.com/problems/APIO10A/
* http://www.spoj.com/problems/ACQUIRE/
*/
```

6.2 Convex Hull Trick 2

```
/*
 * Given a set of pairs (m, b) specifying lines of the form y = m*x + b, process
 * a
 * set of x-coordinate queries each asking to find the minimum y-value when any
 * of
 * the given lines are evaluated at the specified x. To instead have the queries
 * optimize for maximum y-value, set the QUERY_MAX flag to true.
 * The following implementation is a fully dynamic variant of the convex hull
 * optimization technique, using a self-balancing binary search tree (std::set)
 * to
 * support the ability to call add_line() and get_best() in any desired order.
 * Explanation: http://wcipeg.com/wiki/Convex_hull_trick#Fully_dynamic_variant
 * Time Complexity: O(n log n) on the total number of calls made to add_line(),
 * for
 * any length n sequence of arbitrarily interlaced add_line() and get_min()
 * calls.
 * Each individual call to add_line() is O(log n) amortized and each individual
 * call to get_best() is O(log n), where n is the number of lines added so far.
 * Space Complexity: O(n) auxiliary on the number of calls made to add_line().
 */

#include <limits> // std::numeric_limits
#include <set>

class hull_optimizer {
    struct line {
        long long m, b, val;
        double xlo;
        bool is_query;
        bool query_max;

        line(long long m, long long b, long long val, bool is_query, bool query_max)
        {
            this->m = m;
            this->b = b;
            this->val = val;
            this->xlo = -std::numeric_limits<double>::max();
            this->is_query = is_query;
            this->query_max = query_max;
        }

        bool parallel(const line &l) const { return m == l.m; }
        double intersect(const line &l) const
        {
            if (parallel(l)) return std::numeric_limits<double>::max();
            return (double) (l.b - b) / (m - l.m);
        }

        bool operator<(const line &l) const
        {
            if (l.is_query) return query_max ? (xlo < l.val) : (l.val < xlo);
            return m < l.m;
        }
    };

    std::set<line> hull;
    bool _query_max;

    typedef std::set<line>::iterator hulliter;

    bool has_prev(hulliter it) const { return it != hull.begin(); }
    bool has_next(hulliter it) const
    {
        return (it != hull.end()) && (++it != hull.end());
    }

    bool irrelevant(hulliter it) const
    {
        if (!has_prev(it) || !has_next(it)) return false;
        hulliter prev = it, next = it;
        --prev;
        ++next;
        return _query_max ? prev->intersect(*next) <= prev->intersect(*it)
            : next->intersect(*prev) <= next->intersect(*it);
    }

    hulliter update_left_border(hulliter it)
    {
        if ((_query_max && !has_prev(it)) || (!_query_max && !has_next(it)))
            return it;
        hulliter it2 = it;
        double val = it->intersect(_query_max ? --it2 : ++it2);
    }
};
```

```
    line l(*it);
    l.xlo = val;
    hull.erase(it++);
    return hull.insert(it, l);
}

public:
hull_optimizer(bool query_max = false) { this->_query_max = query_max; }
void add_line(long long m, long long b)
{
    line l(m, b, 0, false, _query_max);
    hulliter it = hull.lower_bound(l);
    if (it != hull.end() && it->parallel(l)) {
        if ((_query_max && it->b < b) || (!_query_max && b < it->b))
            hull.erase(it++);
        else
            return;
    }
    it = hull.insert(it, l);
    if (irrelevant(it)) {
        hull.erase(it);
        return;
    }
    while (has_prev(it) && irrelevant(--it)) hull.erase(it++);
    while (has_next(it) && irrelevant(++it)) hull.erase(it--);
    it = update_left_border(it);
    if (has_prev(it)) update_left_border(--it);
    if (has_next(++it)) update_left_border(++it);
}

long long get_best(long long x) const
{
    line q(0, 0, x, true, _query_max);
    hulliter it = hull.lower_bound(q);
    if (_query_max) --it;
    return it->m * x + it->b;
}
};

/** Example Usage **/

#include <cassert>

int main()
{
    hull_optimizer h;
    h.add_line(3, 0);
    h.add_line(0, 6);
    h.add_line(1, 2);
    h.add_line(2, 1);
    assert(h.get_best(0) == 0);
    assert(h.get_best(2) == 4);
    assert(h.get_best(1) == 3);
    assert(h.get_best(3) == 5);
    return 0;
}
```

6.3 Divide-and-Conquer

```
//Um exemplo de Divide and conquer:
int MOD = 1e9 + 7;
const int N = 1010;
int dp[N][N], cost[N][N], v[N], pref[N], n, m;
void compDP(int j, int L, int R, int b, int e)
{
    if (L > R) return;
    int mid = (L + R) / 2;
    int idx = -1;
    for (int i = b; i <= min(mid, e); i++)
        if (dp[mid][j] > dp[i][j - 1] + cost[i + 1][mid]) {
            idx = i;
            dp[mid][j] = dp[i][j - 1] + cost[i + 1][mid];
        }
    compDP(j, L, mid - 1, b, idx);
    compDP(j, mid + 1, R, idx, e);
}

//chamada!
for(int i=1;i<=n;i++) dp[i][0]=cost[1][i];
for(int i=1;i<=m;i++) compDP(i,1,n,1,n);
```

6.4 LIS - Longest Increasing Subsequence

```
//asw -> vetor com resposta!!
```

```
//asw.size() o tamanho da maior lis
void lis( const vector< int > & v, vector< int > & asw )
{
    vector<int> pd(v.size(),0), pd_index(v.size()), pred(v.size());
    int maxi = 0, x=0, j=0, ind=0;
    for(int i=0; i<v.size(); i++)
    {
        x = v[i];
        j=lower_bound(pd.begin(),pd.begin()+maxi,x) -pd.begin();
        pd[j] = x;
        pd_index[j] = i;
        if(j==maxi)
        {
            maxi++;
            ind = i;
        }
        if(pred[i] == j) pd_index[j-1] = -1;
    }
    int pos=maxi-1,k=v[ind];
    asw.resize( maxi );
    while ( pos >= 0 )
    {
        asw[pos--] = k;
        ind = pred[ind];
        k = v[ind];
    }
}
```

7 Geometry

7.1 Convex Hull - Monotone Chain

```
typedef struct sPoint {
    int x, y;
    sPoint(int _x, int _y)
    {
        x = _x;
        y = _y;
    }
} point;
bool comp(point a, point b)
{
    if (a.x == b.x) return a.y < b.y;
    return a.x < b.x;
}
int cross(point a, point b, point c) // AB x BC
{
    a.x -= b.x;
    a.y -= b.y;
    b.x -= c.x;
    b.y -= c.y;
    return a.x * b.y - a.y * b.x;
}
bool isCw(point a, point b, point c) // Clockwise
{
    return cross(a, b, c) < 0;
}
// >= if you want to put collinear points on the convex hull
bool isCcw(point a, point b, point c) // Counter Clockwise
{
    return cross(a, b, c) > 0;
}
vector<point> convexHull(vector<point> p)
{
    vector<point> u, l; // Upper and Lower hulls
    sort(p.begin(), p.end(), comp);
    for (unsigned int i = 0; i < p.size(); i++) {
        while (l.size() > 1 && !isCcw(l[l.size() - 1], l[l.size() - 2], p[i]))
            l.erase(l.begin() + (l.size() - 1));
        l.push_back(p[i]);
    }
    for (int i = p.size() - 1; i >= 0; i--) {
        while (u.size() > 1 && !isCcw(u[u.size() - 1], u[u.size() - 2], p[i]))
            u.erase(u.begin() + (u.size() - 1));
        u.push_back(p[i]);
    }
    u.erase(u.begin() + (u.size() - 1));
    l.erase(l.begin() + (l.size() - 1));
    l.insert(l.end(), u.begin(), u.end());
    return l;
}
```

7.2 Minimum Enclosing Circle

```
//6.5- Minimum Enclosing Circle
const double eps = 1e-6;
#define CIRCLE circ
#define PT Ponto
#define MP 101
#define eps 1e-9
#define x first
#define y second
typedef double cood;
typedef int num;
typedef int point;
double resp;
cood x[MP], y[MP], ar, ax, ay;
int p[MP];
typedef pair<double, double> ponto;
typedef pair<double, double> Ponto;
double dista(ponto a, ponto b)
{
    return sqrt((a.first - b.first) * (a.first - b.first) +
                (a.second - b.second) * (a.second - b.second));
}
bool in(ponto a, pair<double, ponto> c)
{
    if (dista(a, c.second) - eps < c.first) return true;
    return false;
}
bool same(point a, point b)
{
    return (fabs(x[a] - x[b]) < eps && fabs(y[a] - y[b]) < eps);
}
bool lexLess(point a, point b)
{
    if (fabs(x[a] - x[b]) < eps) return y[a] < y[b];
    return x[a] < x[b];
}
inline cood dist(cood xx, cood yy, point a)
{
    return sqrt((xx - x[a]) * (xx - x[a]) + (yy - y[a]) * (yy - y[a]));
}
inline cood cP(point a, point b, point c)
{
    return (x[a] - x[b]) * (y[c] - y[b]) - (x[c] - x[b]) * (y[a] - y[b]);
}
void findCircle(point a, point b, point c, cood& cx, cood& cy)
{
    cx = 0.5 * (x[a] * x[a] + x[a] * y[a] * y[a] - x[b] * x[b] - y[b] * y[b]) *
        (y[b] - y[c]) -
        0.5 * (x[b] * x[b] + y[b] * y[b] - x[c] * x[c] - y[c] * y[c]) *
        (y[a] - y[b]),
    cy = 0.5 * (x[b] * x[b] + y[b] * y[b] - x[c] * x[c] - y[c] * y[c]) *
        (x[a] - x[b]) -
        0.5 * (x[a] * x[a] + y[a] * y[a] - x[b] * x[b] - y[b] * y[b]) *
        (x[b] - x[c]);
    cx /= (x[a] - x[b]) * (y[b] - y[c]) - (x[b] - x[c]) * (y[a] - y[b]);
    cy /= (x[a] - x[b]) * (y[b] - y[c]) - (x[b] - x[c]) * (y[a] - y[b]);
}
void spanCircle2(int k, point p0, point p1, cood& cx, cood& cy, cood& r)
{
    cx = 0.5 * (x[p0] + x[p1]);
    cy = 0.5 * (y[p0] + y[p1]);
    r = dist(cx, cy, p0);
    for (int i = 0; i < k; i++)
        if (dist(cx, cy, p[i]) > r) {
            findCircle(p0, p1, p[i], cx, cy);
            r = dist(cx, cy, p[i]);
        }
}
void spanCircle1(int k, point p0, cood& cx, cood& cy, cood& r)
{
    cx = 0.5 * (x[p0] + x[p[0]]);
    cy = 0.5 * (y[p0] + y[p[0]]);
    r = dist(cx, cy, p0);
    for (int i = 0; i < k; i++)
        if (dist(cx, cy, p[i]) > r) spanCircle2(i, p0, p[i], cx, cy, r);
}
void spanCircle(int n, cood& cx, cood& cy, cood& r)
```

```

{
    // Bem importante, retirar repetidos
    sort(p, p + 1, lexLess);
    n = unique(p, p + n) - p;
    random_shuffle(p, p + n);
    if (n > 1) {
        cx = 0.5 * (x[p[0]] + x[p[1]]);
        cy = 0.5 * (y[p[0]] + y[p[1]]);
        r = dist(cx, cy, p[1]);
        for (int i = 2; i < n; i++)
            if (dist(cx, cy, p[i]) > r) spanCircle1(i, p[i], cx, cy, r);
    }
    else {
        cx = x[0];
        cy = y[0];
        r = 0.0;
    }
}

void solve(vector<pair<double, double> >& v)
{
    int N = v.size();
    for (int i = 0; i < N; i++) {
        x[i] = v[i].first;
        y[i] = v[i].second;
        p[i] = i;
    }
    spanCircle(N, ax, ay, ar);
}

```

7.3 Minimum Enclosing Circle 2

```

const double eps = 1e-6;
#define CIRCLE circ
#define PT Ponto
#define MP 101
#define eps 1e-9
#define x first
#define y second
typedef double cood;
typedef int num;
typedef int point;
double resp;
cood x[MP], y[MP], ar, ax, ay;
int p[MP];
typedef pair<double, double> ponto;
typedef pair<double, double> Ponto;
double dista(ponto a, ponto b)
{
    return sqrt((a.first - b.first) * (a.first - b.first) +
                (a.second - b.second) * (a.second - b.second));
}

bool in(ponto a, pair<double, ponto> c)
{
    if (dista(a, c.second) - eps < c.first) return true;
    return false;
}

bool same(point a, point b)
{
    return (fabs(x[a] - x[b]) < eps && fabs(y[a] - y[b]) < eps);
}

bool lexLess(point a, point b)
{
    if (fabs(x[a] - x[b]) < eps) return y[a] < y[b];
    return x[a] < x[b];
}

inline cood dist(cood xx, cood yy, point a)
{
    return sqrt((xx - x[a]) * (xx - x[a]) + (yy - y[a]) * (yy - y[a]));
}

inline cood cP(point a, point b, point c)
{
    return (x[a] - x[b]) * (y[c] - y[b]) - (x[c] - x[b]) * (y[a] - y[b]);
}

void findCircle(point a, point b, point c, cood& cx, cood& cy)
{
    cx = 0.5 * (x[a] + x[a] + y[a] * y[a] - x[b] * x[b] - y[b] * y[b]) *
        (y[b] - y[c]) -
        0.5 * (x[b] * x[b] + y[b] * y[b] - x[c] * x[c] - y[c] * y[c]) *
        (y[a] - y[b]),
    cy = 0.5 * (x[b] * x[b] + y[b] * y[b] - x[c] * x[c] - y[c] * y[c]) *
        (x[a] - x[b]) -
        0.5 * (x[a] * x[a] + y[a] * y[a] - x[b] * x[b] - y[b] * y[b]) *
        (x[b] - x[c]);
}

```

```

cx /= (x[a] - x[b]) * (y[b] - y[c]) - (x[b] - x[c]) * (y[a] - y[b]);
cy /= (x[a] - x[b]) * (y[b] - y[c]) - (x[b] - x[c]) * (y[a] - y[b]);
}

void spanCircle2(int k, point p0, point p1, cood& cx, cood& cy, cood& r)
{
    cx = 0.5 * (x[p0] + x[p1]);
    cy = 0.5 * (y[p0] + y[p1]);
    r = dist(cx, cy, p0);
    for (int i = 0; i < k; i++)
        if (dist(cx, cy, p[i]) > r) {
            findCircle(p0, p1, p[i], cx, cy);
            r = dist(cx, cy, p[i]);
        }
}

void spanCircle1(int k, point p0, cood& cx, cood& cy, cood& r)
{
    cx = 0.5 * (x[p0] + x[p[0]]);
    cy = 0.5 * (y[p0] + y[p[0]]);
    r = dist(cx, cy, p0);
    for (int i = 0; i < k; i++)
        if (dist(cx, cy, p[i]) > r) spanCircle2(i, p0, p[i], cx, cy, r);
}

void spanCircle(int n, cood& cx, cood& cy, cood& r)
{
    // Bem importante, retirar repetidos
    sort(p, p + 1, lexLess);
    n = unique(p, p + n) - p;
    random_shuffle(p, p + n);
    if (n > 1) {
        cx = 0.5 * (x[p[0]] + x[p[1]]);
        cy = 0.5 * (y[p[0]] + y[p[1]]);
        r = dist(cx, cy, p[1]);
        for (int i = 2; i < n; i++)
            if (dist(cx, cy, p[i]) > r) spanCircle1(i, p[i], cx, cy, r);
    }
    else {
        cx = x[0];
        cy = y[0];
        r = 0.0;
    }
}

void solve(vector<pair<double, double> >& v)
{
    int N = v.size();
    for (int i = 0; i < N; i++) {
        x[i] = v[i].first;
        y[i] = v[i].second;
        p[i] = i;
    }
    spanCircle(N, ax, ay, ar);
}

```

7.4 Fast Geometry in Cpp

```

// C++ routines for computational geometry.

#include <iostream>
#include <vector>
#include <cmath>
#include <cassert>

using namespace std;

double INF = 1e100;
double EPS = 1e-12;

struct PT {
    double x, y;
    PT() {}
    PT(double x, double y) : x(x), y(y) {}
    PT(const PT &p) : x(p.x), y(p.y) {}
    PT operator + (const PT &p) const { return PT(x+p.x, y+p.y); }
    PT operator - (const PT &p) const { return PT(x-p.x, y-p.y); }
    PT operator * (double c) const { return PT(x*c, y*c); }
    PT operator / (double c) const { return PT(x/c, y/c); }
};

double dot(PT p, PT q) { return p.x*q.x+p.y*q.y; }
double dist2(PT p, PT q) { return dot(p-q,p-q); }
double cross(PT p, PT q) { return p.x*q.y-p.y*q.x; }
ostream &operator<<(ostream &os, const PT &p) {
    os << "(" << p.x << ", " << p.y << ")";
}

```

```

// rotate a point CCW or CW around the origin
PT RotateCCW90(PT p) { return PT(-p.y,p.x); }
PT RotateCW90(PT p) { return PT(p.y,-p.x); }
PT RotateCCW(PT p, double t) {
    return PT(p.x*cos(t)-p.y*sin(t), p.x*sin(t)+p.y*cos(t));
}

// project point c onto line through a and b
// assuming a != b
PT ProjectPointLine(PT a, PT b, PT c) {
    return a + (b-a)*dot(c-a, b-a)/dot(b-a, b-a);
}

// project point c onto line segment through a and b
PT ProjectPointSegment(PT a, PT b, PT c) {
    double r = dot(b-a,b-a);
    if (fabs(r) < EPS) return a;
    r = dot(c-a, b-a)/r;
    if (r < 0) return a;
    if (r > 1) return b;
    return a + (b-a)*r;
}

// compute distance from c to segment between a and b
double DistancePointSegment(PT a, PT b, PT c) {
    return sqrt(dist2(c, ProjectPointSegment(a, b, c)));
}

// compute distance between point (x,y,z) and plane ax+by+cz=d
double DistancePointPlane(double x, double y, double z,
    double a, double b, double c, double d)
{
    return fabs(a*x+b*y+c*z-d)/sqrt(a*a+b*b+c*c);
}

// determine if lines from a to b and c to d are parallel or collinear
bool LinesParallel(PT a, PT b, PT c, PT d) {
    return fabs(cross(b-a, c-d)) < EPS;
}

bool LinesCollinear(PT a, PT b, PT c, PT d) {
    return LinesParallel(a, b, c, d)
        && fabs(cross(a-b, a-c)) < EPS
        && fabs(cross(c-d, c-a)) < EPS;
}

// determine if line segment from a to b intersects with
// line segment from c to d
bool SegmentsIntersect(PT a, PT b, PT c, PT d) {
    if (LinesCollinear(a, b, c, d)) {
        if (dist2(a, c) < EPS || dist2(a, d) < EPS ||
            dist2(b, c) < EPS || dist2(b, d) < EPS) return true;
        if (dot(c-a, c-b) > 0 && dot(d-a, d-b) > 0 && dot(c-b, d-b) > 0)
            return false;
        return true;
    }
    if (cross(d-a, b-a) * cross(c-a, b-a) > 0) return false;
    if (cross(a-c, d-c) * cross(b-c, d-c) > 0) return false;
    return true;
}

// compute intersection of line passing through a and b
// with line passing through c and d, assuming that unique
// intersection exists; for segment intersection, check if
// segments intersect first
PT ComputeLineIntersection(PT a, PT b, PT c, PT d) {
    b=b-a; d=d-c; c=c-a;
    assert(dot(b, b) > EPS && dot(d, d) > EPS);
    return a + b*cross(c, d)/cross(b, d);
}

// compute center of circle given three points
PT ComputeCircleCenter(PT a, PT b, PT c) {
    b=(a+b)/2;
    c=(a+c)/2;
    return ComputeLineIntersection(b, b+RotateCW90(a-b), c, c+RotateCW90(a-c));
}

// determine if point is in a possibly non-convex polygon (by William
// Randolph Franklin); returns 1 for strictly interior points, 0 for
// strictly exterior points, and 0 or 1 for the remaining points.
// Note that it is possible to convert this into an *exact* test using
// integer arithmetic by taking care of the division appropriately
// (making sure to deal with signs properly) and then by writing exact
// tests for checking point on polygon boundary
bool PointInPolygon(const vector<PT> &p, PT q) {
    bool c = 0;
    for (int i = 0; i < p.size(); i++){
        int j = (i+1)%p.size();
        if ((p[i].y <= q.y && q.y < p[j].y ||
            p[j].y <= q.y && q.y < p[i].y) &&
            q.x < p[i].x + (p[j].x - p[i].x) * (q.y - p[i].y) / (p[j].y - p[i].y))

```

```

        c = !c;
    }
    return c;
}

// determine if point is on the boundary of a polygon
bool PointOnPolygon(const vector<PT> &p, PT q) {
    for (int i = 0; i < p.size(); i++)
        if (dist2(ProjectPointSegment(p[i], p[(i+1)%p.size()], q), q) < EPS)
            return true;
    return false;
}

// compute intersection of line through points a and b with
// circle centered at c with radius r > 0
vector<PT> CircleLineIntersection(PT a, PT b, PT c, double r) {
    vector<PT> ret;
    b = b-a;
    a = a-c;
    double A = dot(b, b);
    double B = dot(a, b);
    double C = dot(a, a) - r*r;
    double D = B*B - A*C;
    if (D < -EPS) return ret;
    ret.push_back(c+a+b*(-B+sqrt(D+EPS))/A);
    if (D > EPS)
        ret.push_back(c+a+b*(-B-sqrt(D))/A);
    return ret;
}

// compute intersection of circle centered at a with radius r
// with circle centered at b with radius R
vector<PT> CircleCircleIntersection(PT a, PT b, double r, double R) {
    vector<PT> ret;
    double d = sqrt(dist2(a, b));
    if (d > r+R || d+min(r, R) < max(r, R)) return ret;
    double x = (d*d-R*R+r*r)/(2*d);
    double y = sqrt(r*r-x*x);
    PT v = (b-a)/d;
    ret.push_back(a+v*x + RotateCCW90(v)*y);
    if (y > 0)
        ret.push_back(a+v*x - RotateCCW90(v)*y);
    return ret;
}

// This code computes the area or centroid of a (possibly nonconvex)
// polygon, assuming that the coordinates are listed in a clockwise or
// counterclockwise fashion. Note that the centroid is often known as
// the "center of gravity" or "center of mass".
double ComputeSignedArea(const vector<PT> &p) {
    double area = 0;
    for(int i = 0; i < p.size(); i++) {
        int j = (i+1) % p.size();
        area += p[i].x*p[j].y - p[j].x*p[i].y;
    }
    return area / 2.0;
}

double ComputeArea(const vector<PT> &p) {
    return fabs(ComputeSignedArea(p));
}

PT ComputeCentroid(const vector<PT> &p) {
    PT c(0,0);
    double scale = 6.0 * ComputeSignedArea(p);
    for (int i = 0; i < p.size(); i++){
        int j = (i+1) % p.size();
        c = c + (p[i]+p[j])*p[i].x*p[j].y - p[j].x*p[i].y;
    }
    return c / scale;
}

// tests whether or not a given polygon (in CW or CCW order) is simple
bool IsSimple(const vector<PT> &p) {
    for (int i = 0; i < p.size(); i++) {
        for (int k = i+1; k < p.size(); k++) {
            int j = (i+1) % p.size();
            int l = (k+1) % p.size();
            if (i == l || j == k) continue;
            if (SegmentsIntersect(p[i], p[j], p[k], p[l]))
                return false;
        }
    }
    return true;
}

int main() {
    // expected: (-5,2)
    cerr << RotateCCW90(PT(2,5)) << endl;

    // expected: (5,-2)

```

```

cerr << RotateCW90(PT(2,5)) << endl;

// expected: (-5,2)
cerr << RotateCCW(PT(2,5),M_PI/2) << endl;

// expected: (5,2)
cerr << ProjectPointLine(PT(-5,-2), PT(10,4), PT(3,7)) << endl;

// expected: (5,2) (7.5,3) (2.5,1)
cerr << ProjectPointSegment(PT(-5,-2), PT(10,4), PT(3,7)) << " "
    << ProjectPointSegment(PT(7.5,3), PT(10,4), PT(3,7)) << " "
    << ProjectPointSegment(PT(-5,-2), PT(2.5,1), PT(3,7)) << endl;

// expected: 6.78903
cerr << DistancePointPlane(4,-4,3,2,-2,5,-8) << endl;

// expected: 1 0 1
cerr << LinesParallel(PT(1,1), PT(3,5), PT(2,1), PT(4,5)) << " "
    << LinesParallel(PT(1,1), PT(3,5), PT(2,0), PT(4,5)) << " "
    << LinesParallel(PT(1,1), PT(3,5), PT(5,9), PT(7,13)) << endl;

// expected: 0 0 1
cerr << LinesCollinear(PT(1,1), PT(3,5), PT(2,1), PT(4,5)) << " "
    << LinesCollinear(PT(1,1), PT(3,5), PT(2,0), PT(4,5)) << " "
    << LinesCollinear(PT(1,1), PT(3,5), PT(5,9), PT(7,13)) << endl;

// expected: 1 1 1 0
cerr << SegmentsIntersect(PT(0,0), PT(2,4), PT(3,1), PT(-1,3)) << " "
    << SegmentsIntersect(PT(0,0), PT(2,4), PT(4,3), PT(0,5)) << " "
    << SegmentsIntersect(PT(0,0), PT(2,4), PT(2,-1), PT(-2,1)) << " "
    << SegmentsIntersect(PT(0,0), PT(2,4), PT(5,5), PT(1,7)) << endl;

// expected: (1,2)
cerr << ComputeLineIntersection(PT(0,0), PT(2,4), PT(3,1), PT(-1,3)) << endl;

// expected: (1,1)
cerr << ComputeCircleCenter(PT(-3,4), PT(6,1), PT(4,5)) << endl;

vector<PT> v;
v.push_back(PT(0,0));
v.push_back(PT(5,0));
v.push_back(PT(5,5));
v.push_back(PT(0,5));

// expected: 1 1 1 0 0
cerr << PointInPolygon(v, PT(2,2)) << " "
    << PointInPolygon(v, PT(2,0)) << " "
    << PointInPolygon(v, PT(0,2)) << " "
    << PointInPolygon(v, PT(5,2)) << " "
    << PointInPolygon(v, PT(2,5)) << endl;

// expected: 0 1 1 1 1
cerr << PointOnPolygon(v, PT(2,2)) << " "
    << PointOnPolygon(v, PT(2,0)) << " "
    << PointOnPolygon(v, PT(0,2)) << " "
    << PointOnPolygon(v, PT(5,2)) << " "
    << PointOnPolygon(v, PT(2,5)) << endl;

// expected: (1,6)
// (5,4) (4,5)
// blank line
// (4,5) (5,4)
// blank line
// (4,5) (5,4)
vector<PT> u = CircleLineIntersection(PT(0,6), PT(2,6), PT(1,1), 5);
for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
u = CircleLineIntersection(PT(0,9), PT(9,0), PT(1,1), 5);
for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
u = CircleCircleIntersection(PT(1,1), PT(10,10), 5, 5);
for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
u = CircleCircleIntersection(PT(1,1), PT(8,8), 5, 5);
for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
u = CircleCircleIntersection(PT(1,1), PT(4.5,4.5), 10, sqrt(2.0)/2.0);
for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;
u = CircleCircleIntersection(PT(1,1), PT(4.5,4.5), 5, sqrt(2.0)/2.0);
for (int i = 0; i < u.size(); i++) cerr << u[i] << " "; cerr << endl;

// area should be 5.0
// centroid should be (1.1666666, 1.1666666)
PT pa[] = { PT(0,0), PT(5,0), PT(1,1), PT(0,5) };
vector<PT> p(pa, pa+4);
PT c = ComputeCentroid(p);
cerr << "Area: " << ComputeArea(p) << endl;
cerr << "Centroid: " << c << endl;

return 0;
}

```

8 Data Structures

8.1 Disjoint Set Union

```

const int N=500010;
int p[N],rank[N];
void init()
{
    memset(rank,0,sizeof(rank));
    for(int i=0;i<N;i++) p[i]=i;
}
int findset(int i)
{
    if(p[i]==i) return i;
    return p[i]=findset(p[i]);
}
bool same(int i, int j)
{
    return (findset(i) == findset(j));
}
void unionSet(int i, int j)
{
    if (!same(i, j)) {
        int x = findset(i), y=findset(j);
        if (rank[x] > rank[y])
            p[y] = x;
        else {
            p[x] = y;
            if (rank[x] == rank[y]) rank[y]++;
        }
    }
}

```

8.2 Persistent Segment Tree

```

//PRINTAR O NUMERO DE ELEMENTOS DISTINTOS
//EM UM INTERVALO DO ARRAY
const int N = 30010;
int tr[100 * N], L[100 * N], R[100 * N], root[100 * N];
int v[N], mapa[100 * N];
int cont = 0;
void build(int node, int b, int e)
{
    if (b == e) {
        tr[node] = 0;
    }
    else {
        L[node] = cont++;
        R[node] = cont++;
        build(L[node], b, (b + e) / 2);
        build(R[node], (b + e) / 2 + 1, e);
        tr[node] = tr[L[node]] + tr[R[node]];
    }
}
int update(int node, int b, int e, int i, int val)
{
    int idx = cont++;
    tr[idx] = tr[node] + val;
    L[idx] = L[node];
    R[idx] = R[node];
    if (b == e) return idx;
    int mid = (b + e) / 2;
    if (i <= mid)
        L[idx] = update(L[node], b, mid, i, val);
    else
        R[idx] = update(R[node], mid + 1, e, i, val);
    return idx;
}
int query(int nodeL, int nodeR, int b, int e, int i, int j)
{
    if (b > j or i > e) return 0;
    if (i <= b and j >= e) {
        int p1 = tr[nodeR];
        int p2 = tr[nodeL];
        return p1 - p2;
    }
    int mid = (b + e) / 2;
    return query(L[nodeL], L[nodeR], b, mid, i, j) +
        query(R[nodeL], R[nodeR], mid + 1, e, i, j);
}
int main()

```

```

{
    int n;
    sc(n);
    memset(mapa, -1, sizeof(mapa));
    for (int i = 0; i < n; i++) sc(v[i]);
    build(1, 0, n - 1);
    for (int i = 0; i < n; i++) {
        if (mapa[v[i]] == -1) {
            root[i + 1] = update(root[i], 0, n - 1, i, 1);
            mapa[v[i]] = i;
        }
        else {
            root[i + 1] = update(root[i], 0, n - 1, mapa[v[i]], -1);
            mapa[v[i]] = i;
            root[i + 1] = update(root[i + 1], 0, n - 1, i, 1);
        }
    }
    int q;
    sc(q);
    for (int i = 0; i < q; i++) {
        int l, r;
        sc2(l, r);
        int resp = query(root[l - 1], root[r], 0, n - 1, l - 1, r - 1);
        pri(resp);
    }
    return 0;
}

```

8.3 RMQ of Indices

```

//RMQ DE INDICE
class RMQ {
private:
    vi A;
    vi M;
public:
    RMQ(vi &v)
    {
        A = v;
        M.resize(4 * v.size());
        build(1, 0, v.size() - 1);
    }
    void build(int node, int b, int e)
    {
        if (b == e)
            M[node] = b;
        else {
            build(2 * node, b, (b + e) / 2);
            build(2 * node + 1, (b + e) / 2 + 1, e);
            if (A[M[2 * node]] <= A[M[2 * node + 1]])
                M[node] = M[2 * node];
            else
                M[node] = M[2 * node + 1];
        }
    }
    int query(int node, int b, int e, int i, int j)
    {
        int p1, p2;
        if (i > e || j < b) return -1;
        if (b >= i and e <= j) return M[node];
        p1 = query(2 * node, b, (b + e) / 2, i, j);
        p2 = query(2 * node + 1, (b + e) / 2 + 1, e, i, j);
        if (p1 == -1) return p2;
        if (p2 == -1) return p1;
        if (A[p1] <= A[p2]) return p1;
        return p2;
    }
    void atualiza(int node, int b, int e, int i, int val)
    {
        if (i > e || i < b) return;
        if (e == b) {
            A[i] = val;
        }
        else {
            atualiza(2 * node, b, (b + e) / 2, i, val);
            atualiza(2 * node + 1, (b + e) / 2 + 1, e, i, val);
            if (A[M[2 * node]] <= A[M[2 * node + 1]])
                M[node] = M[2 * node];
            else
                M[node] = M[2 * node + 1];
        }
    }
};

```

8.4 RSQ with Lazy-Propagation

```

//RSQ COM LAZY PROPAGATION!
class RSQ {
private:
    vll A;
    vll M;
    vll lazy;
public:
    RSQ(vll &v)
    {
        A = v;
        M.resize(v.size() * 4);
        lazy.assign(v.size() * 4, 0);
        build(1, 0, v.size() - 1);
    }
    void build(int node, int b, int e)
    {
        if (b == e) {
            M[node] = A[b];
            return;
        }
        build(2 * node, b, (b + e) / 2);
        build(2 * node + 1, (b + e) / 2 + 1, e);
        M[node] = M[2 * node] + M[2 * node + 1];
    }
    void atualiza(int node, int b, int e, int i, int j, ll val)
    {
        if (lazy[node] != 0) {
            M[node] += lazy[node];
            if (b != e) {
                ll inter = (e - b + 1);
                ll i1 = (b + e) / 2 - b + 1;
                ll i2 = e - (b + e) / 2;
                ll un = lazy[node] / inter;
                lazy[2 * node] += un * i1;
                lazy[2 * node + 1] += un * i2;
            }
            lazy[node] = 0;
        }
        if (i > e or j < b) return;
        if (i <= b and j >= e) {
            ll inter = (e - b + 1);
            M[node] += val * inter;
            if (b != e) {
                ll i1 = (b + e) / 2 - b + 1;
                ll i2 = e - (b + e) / 2;
                lazy[2 * node] += i1 * (ll)val;
                lazy[2 * node + 1] += i2 * (ll)val;
            }
            return;
        }
        atualiza(2 * node, b, (b + e) / 2, i, j, val);
        atualiza(2 * node + 1, (b + e) / 2 + 1, e, i, j, val);
        M[node] = M[2 * node] + M[2 * node + 1];
    }
    ll query(int node, int b, int e, int i, int j)
    {
        if (i > e or j < b) return 0;
        ll p1, p2;
        if (lazy[node] != 0) {
            M[node] += lazy[node];
            if (b != e) {
                ll inter = (e - b + 1);
                ll i1 = (b + e) / 2 - b + 1;
                ll i2 = e - (b + e) / 2;
                ll un = lazy[node] / inter;
                lazy[2 * node] += un * i1;
                lazy[2 * node + 1] += un * i2;
            }
            lazy[node] = 0;
        }
        if (i <= b and j >= e) return M[node];
        p1 = query(2 * node, b, (b + e) / 2, i, j);
        p2 = query(2 * node + 1, (b + e) / 2 + 1, e, i, j);
        return p1 + p2;
    }
};

```

8.5 Segment Tree

```

//compilar em C++11, essa segment tree
//computa qual e o k's elemento compreendido

```

8.6 Sparse Table

```
//no intervalo entre i,j
//presentes no array
vi tr[5 * N];
void build(int node, int b, int e)
{
    if (b == e)
        tr[node].pb(v[b]);
    else {
        build(2 * node, b, (b + e) / 2);
        build(2 * node + 1, (b + e) / 2 + 1, e);
        merget(tr[2 * node], tr[2 * node + 1], tr[node]);
        merge(tr[2 * node].begin(), tr[2 * node].end(), tr[2 * node + 1].begin(),
              tr[2 * node + 1].end(), back_inserter(tr[node]));
    }
}

int query(int node, int b, int e, int i, int j, int k)
{
    if (i > e or b > j) return 0;
    if (i <= b and j >= e) {
        int resp =
            upper_bound(tr[node].begin(), tr[node].end(), k) - tr[node].begin();
        return tr[node].size() - resp;
    }
    return query(2 * node, b, (b + e) / 2, i, j, k) +
           query(2 * node + 1, (b + e) / 2 + 1, e, i, j, k);
}
```

```
//computar RMQ , favor inicializar: dp[i][0]=v[0]
//sendo v[0] o vetor do rmq
//chamar o build!
int dp[200100][22];
int n;
int d[200100];
void build()
{
    d[0] = d[1] = 0;
    for (int i = 2; i < n; i++) d[i] = d[i >> 1] + 1;
    for (int j = 1; j < 22; j++) {
        for (int i = 0; i + (1 << (j - 1)) < n; i++) {
            dp[i][j] = min(dp[i][j - 1], dp[i + (1 << (j - 1))][j - 1]);
        }
    }
}

int query(int i, int j)
{
    int k = d[j - i];
    int x = min(dp[i][k], dp[j - (1 << k) + 1][k]);
    return x;
}
```