

Contents

1 Graph Algorithms

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

2 Strings

2.1	Aho-Corasick	3
2.2	KMP	4
2.3	Suffix Array	4
2.4	Rabin Karp	4

3 Numerical Algorithms

3.1	Fast Fourier Transform	5
3.2	Fast Fourier Transform 2	5
3.3	Simpson's Algorithm	5

4 Mathematics

4.1	Big Number	6
4.2	Chinese Remainder	6
4.3	Matrix Exponentiation	6
4.4	Pascal Triangle	6
4.5	Euler's Totient Function	7
4.6	Pollard Rho	7
4.7	Sieve of Eratosthenes	7
4.8	Extended Euclidean Algorithm	7
4.9	Multiplicative Inverse	8

5 Combinatorial Optimization

5.1	Dinic	8
5.2	Hopcroft-Karp	8
5.3	Min Cost Max Flow	8
5.4	Edmonds Karp	9

1 Graph Algorithms

1.1 2-SAT

```

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

```

class kosaraju {
private:
    vi usados;
    vvi graph;
    vvi 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;
    }
    vvi compression(int n)
    {
        int tam = SSC(n);
        vvi resp;
        vvi Trans;
        resp.resize(tam);
        Trans.resize(tam);
        for (int u = 0; u < graph.size(); u++) {

```

```

for (int j = 0; j < graph[u].size(); j++) {
    int v = graph[u][j];
    if (usados[u] != usados[v]) {
        resp[usados[u] - 1].pb(usados[v] - 1);
        Trans[usados[v] - 1].pb(usados[u] - 1);
    }
}
return Trans;
}
}
;

```

1.3 Tree Isomorphism

```

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

```

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<i> 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

```

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

```

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[1000000];
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(L, 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 Rabin Karp

```

const ll M = 1000004099;
const ll B = 31;
ll int_mod(ll a, ll b) { return (a % b + b) % b; }
ll eleva(ll a, ll b, ll mod)
{
    if (b == 0)
        return 1;
    else if (b == 1)
        return a;
    ll x = eleva(a, b / 2, mod);
    if (b % 2 == 0)
        return (x * x) % mod;
    else
        return (a * ((x * x) % mod)) % mod;
}

bool Rabin_karp(string text, string pattern)
{
    int n = text.size();
    int m = pattern.size();
    if (n < m) return false;
    ll hp = 0;
    for (int i = 0; i < m; i++) hp = int_mod(hp * B + pattern[i], M);
    ll ht = 0;
    for (int i = 0; i < m; i++) ht = int_mod(ht * B + text[i], M);
    if (ht == hp) return true;
    ll E = eleva(B, m - 1, M);
    for (int i = m; i < n; i++) {
        ht = int_mod(ht - int_mod(text[i - m] * E, M), M);
        ht = int_mod(ht * B + text[i], M);
        ht = int_mod(ht + text[i], M);
        if (ht == hp) return true;
    }
    return false;
}

```

3 Numerical Algorithms

3.1 Fast Fourier Transform

```

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 tam,coco,maiorpot2[MAXN];

/*how to use:
    fft(a,1)
    fft(b,1)
    mul(a,b)
    fft(a,-1)
*/

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<tam; i++)
    {
        a[i]=a[i]*b[i];
    }
}

```

3.2 Fast Fourier Transform 2

```

dificil tmp[MAXN*2];
int tam,coco,maiorpot2[MAXN];
/* how to use:
    fft(&a[0],tam,0)
    fft(&b[0],tam,0)
    mul(a,b)
    fft(&a[0],tam,1)
    ->tam tem que ser potencia de 2<-
*/

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<tam; 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;
double integrall(double altura)
{
    double h = W / (NPASSOS);
    double a = 0;
    double b = W;
    double s = f(a, -altura) + f(b, -altura);
}

```

```

for (double i = 1; i <= n; i += 2) s += f(a + i * h, -altura) * 4.0;
for (double i = 2; i <= (n - 1); i += 2) s += f(a + i * h, -altura) * 2.0;
return s * h / 3.0;
}

```

4 Mathematics

4.1 Big Number

```

void zero_esq(string &resp)
{
    string retorno = resp;
    reverse(retorno.begin(), retorno.end());
    int i = resp.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');
        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 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;
}

```

4.3 Matrix Exponentiation

```

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;
}

```

4.4 Pascal Triangle

```

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];

```

4.5 Eulers Totient Function

```

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;
}

```

4.6 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 " <<v<<" " <<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;
}

```

4.7 Sieve of Eratosthenes

```

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;
}

```

4.8 Extended Euclidean Algorithm

```

struct ext {
    ll x;
    ll y;
    ll mdc;
};
ext tmp;
// ax + by=c, se mdc(a,b) nao divide c, nao tem solucao, caso contrario, x = x0
// // + (b/mdc) *n, y=y0-(a/mdc)*n

```

```

ll ee(ll a, ll b, ll &x, ll &y)
{
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    ll x1, y1;
    ll tmp = ee(b, a % b, x1, y1);
    x = y1;
    y = x1 - (a / b) * y1;
    return tmp;
}
ext extended_euclid(ll a, ll b)
{
    ll tmp, tmp1;
    ext ret;
    ret.mdc = ee(a, b, tmp, tmp1);
    ret.x = tmp;
    ret.y = tmp1;
    return ret;
}

```

4.9 Multiplicative Inverse

```

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;
}

```

5 Combinatorial Optimization

5.1 Dinic

```

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

```

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 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();
        }
    }
}

```

5.4 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);
    }

    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;
    }
};

```