# Table des matières



# 1 Paramètres

## 1.1 Compilation

```
g++ -std=c++17 -DLOCAL -D_GLIBCXX_DEBUG -D_GLIBCXX_DEBUG_PEDANTIC
↪  -D_FORTIFY_SOURCE=2 -fsanitize=address -fsanitize=undefined
↪  -fno-sanitize-recover -fsanitize=undefined -fstack-protector -g
↪  -Wall -Wextra -Wshadow -Wformat=2 -Wfloat-equal -Wconversion
↪  -Wlogical-op main.cpp -o prog
```

## 1.2 Squelette de code

```cpp
#include <bits/stdc++.h>
using namespace std;

struct Struct {};
const int constant = 0;
int variable;

// pour eviter les conflits de notation entre codes du notebook
namespace notebook
{

struct Struct {};
const int constant = 0;
```

```
int variable;
void function() {}

};

void function() {}

int main() {
    ios_base::sync_with_stdio(false);
    cin >> variable;
    printf("%d\n", notebook::variable);
}
```

# 2 Combinatoire

## 2.1 Lemme de Burnside et équation aux classes

Si $\cdot$ est une action du groupe $G$ sur l'ensemble $E$ alors on définit

$$\begin{aligned} G^x &= \{g \in G, g \cdot x = x\} \text{ le stabilisateur de } x \\ G \cdot x &= \{g \cdot x, g \in G\} \text{ l'orbite de } x \\ \mathrm{Fix}(g) &= \{x \in E, g \cdot x = x\} \text{ les points fixes de } g \\ \Omega &= \{G \cdot x, x \in E\} \text{ l'ensemble des orbites} \end{aligned}$$

Nous déduisons de la relation $|G \cdot x| = |G|/|G^x|$ l'équation aux classes

$$|\Omega| = \sum_{x \in E} \frac{1}{|G \cdot x|} = \frac{1}{|G|} \sum_{x \in E} |G^x| = \frac{1}{|G|} \sum_{x \in E} \sum_{g \in G} \mathbf{1}_{g \cdot x = x} = \frac{1}{|G|} \sum_{g \in G} |\mathrm{Fix}(g)|$$

## 2.2 Formule de Legendre

La valuation $p$-adique de $n!$ est

$$\nu_p(n!) = \sum_{k=1}^{\infty} \left\lfloor \frac{n}{p^k} \right\rfloor$$

## 2.3 Coefficients binomiaux

$$\binom{n}{k} = \#\{I \subset \{1, \ldots, n\}, |I| = k\} = \frac{k!(n-k)!}{n!}$$

— Symétrie : $\binom{n}{k} = \binom{n}{n-k}$

— Formule de Pascal $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$

— Formule du chef : $n\binom{n-1}{k-1} = k\binom{n}{k}$

— Binôme de Newton : $(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^k b^{n-k}$

— Somme sur $n$ : $\sum_{n=p}^{q} \binom{n}{k} = \binom{q+1}{k+1} - \binom{p}{k+1}$

## 2.4 Nombre de Fibonacci

$$F_0 = 0 \qquad F_1 = 1 \qquad F_{n+2} = F_{n-1} + F_n$$

| $n$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| $F_n$ | 0 | 1 | 1 | 2 | 3 | 5 | 8 | 13 | 21 | 34 |

$$F_{n+1} = \sum_{k=0}^{n} \binom{n-k}{k}$$

## 2.5 Principe d'inclusion-exclusion

$$\left| \bigcup_{i \in I} A_i \right| = \sum_{\substack{J \subset I \\ J \neq \emptyset}} (-1)^{|J|-1} \left| \bigcap_{j \in J} A_j \right|$$

## 2.6 Nombres de Catalan

Le nombre d'arbres binaires à $n+1$ feuilles est

$$C_n = \frac{1}{n+1}\binom{2n}{n}$$

| $n$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| $C_n$ | 1 | 1 | 2 | 5 | 14 | 42 | 132 | 429 | 1430 | 4862 |

Formule de récurrence $C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i}$

### 2.7 Formule de Cayley et théorème de Kirchhoff

Il y a $n^{n-2}$ arbres sur les sommets $\{1,\dots,n\}$ : écrire le voisin de la feuille minimale, la supprimer et recommencer jusqu'à n'avoir que deux sommets :

Le nombre d'arbres couvrants de $G=(V,E)$ est la valeur des mineurs de rang $n-1$ du laplacien $L$ de $G$

$$L_{ij} = \begin{cases} \deg(i) \text{ si } i=j \\ -1 \text{ si } \{i,j\}\in E \\ 0 \text{sinon} \end{cases}$$

# 3 Graphes

## 3.1 DFS : parcours en profondeur

```
vector<int> vs[N];
int visited[N];

void dfs(int u) {
    if (visited[u] == 1)
        /*...*/;
    else if (visited[u] == 2)
        /*...*/;
    else {
        visited[u] = 1;
        for (int v : vs[u])
            dfs(v);
        visited[u] = 2;
    }
}
```

## 3.2 BFS : parcours en largeur

```
int n;
vector<int> vs[N];

vector<int> bfs(int u0) {
    vector<int> dist(n, n + 1);
    queue<int> q;
    dist[u0] = 0;
    q.push(u0);
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        for (int v : vs[u]) if (dist[v] == n + 1) {
            dist[v] = dist[u] + 1;
            q.push(v);
        }
    }
    return dist;
}
```

## 3.3 Plus court chemin

**Poids positifs : Dijkstra**

```
const int oo = 1e9;
int n, m;
vector<pair<int, int>> vs[N];
int dist[N];

void dijkstra(int u0) {
    priority_queue<pair<int, int>> q;
    q.emplace(-0, u0);
    while (!q.empty()) {
        auto [d, u] = q.top();
        q.pop();
        if (dist[u] == +oo) {
            dist[u] = -d;
            for (auto [v, c] : vs[u])
                q.emplace(d - c, v);
        }
    }
}
```

**Bellman–Ford**

```
const int oo = 1e9;
int n, m;
int dist[N];
vector<tuple<int, int, int>> edge;

bool bellmanford (int u0) {
    fill_n (dist, n, +oo);
    dist[u0] = 0;
    bool stable = false;
    for (int t = 0; t < n && !stable; t++) {
        stable = true;
```

```
        for (auto[u, v, c] : edge) if (dist[u] < +oo && dist[u] + c < dist[v]) {
            dist[v] = dist[u] + c;
            stable = false;
        }
    }
    return stable;
}
```

**Entre toutes paires : Floyd–Warshall**

```
void floydwarshall(vector<vector<int>>& d) {
    // d[u][v] = c(u, v) si (u, v) arc et +oo sinon
    int n = d.size();
    for (int w = 0; w < n; w++)
        for (int u = 0; u < n; u++)
            for (int v = 0; v < n; v++)
                d[u][v] = min(d[u][v], d[u][w] + d[w][v]);
}
```

## 3.4 Cycles et chemins eulériens

```
int n, m;
int deg[N];
// non orienté : deg[u]++ ; deg[v]++
// orienté : deg[u]-- ; deg[v]++
vector<pair<int, int>> vs[N];
// non orienté : vs[u].push_back(v, e) ; vs[v].push_back(u, e)
// orienté : vs[u].push_back(v, a)
vector<int> path;
bool visited[M];

void dfs(int u) {
    for (auto e : vs[u]) if (!visited[e.second]) {
        visited[e.second] = true;
        dfs(e.first);
        path.push_back(e.second);
    }
}

bool eulercycle(bool oriented) {
    for (int u = 0; u < n; u++) {
        if (oriented && deg[u] != 0) return false;
        else if (!oriented && deg[u] % 2 != 0) return false;
    }
    dfs(0);
    reverse(path.begin(), path.end());
    return true;
}

bool eulerpath(bool oriented) {
    int s = 0;
    for (int u = 0; u < n; u++) {
        if (oriented && deg[u] > deg[s]) s = u;
        else if (!oriented && deg[u] % 2 == 1) s = u;
    }
    dfs(s);
    reverse(path.begin(), path.end());
    return path.size() == 0;
}
```

## 3.5 Composantes fortement connexes : Kosaraju

```
int n, m;
vector<int> vs[N];
vector<int> rvs[N];
vector<int> topo;
int scc[N];

void toposort(int u) {
    if (!scc[u]) {
        scc[u] = true;
        for (int v : vs[u])
            toposort(v);
        topo.push_back(u);
    }
}

void markscc(int u, int c) {
    if (scc[u] == -1) {
        scc[u] = c;
        for (int v : rvs[u])
            markscc(v, c);
    }
}

void computescc() {
    fill_n(scc, n, false);
```

```
    for (int u = 0; u < n; u++)
        toposort(u);
    reverse(topo.begin(), topo.end());
    fill_n(scc, n, -1);
    int c = 0;
    for (int u : topo) if (scc[u] == -1)
        markscc(u, c++);
}
```

## 3.6 Ponts

```
int n, m;
vector<pair<int, int>> vs[N];
int depth[N];
bool bridge[M];

int dfs(int u, int p = -1, int d = 0) {
    if (depth[u] != -1)
        return depth[u];
    depth[u] = d;
    for (auto [v, e] : vs[u]) if (v != p) {
        int r = dfs(v, u, d + 1);
        if (r <= d)
            bridge[e] = false;
        depth[u] = min(depth[u], r);
    }
    return depth[u];
}
```

## 3.7 Sommets d'articulation

```
int n, m;
vector<int> vs[N];
int depth[N];
bool articulation[N];

int dfs(int u, int p = -1, int d = 0) {
    if (depth[u] != -1)
        return depth[u];
    int low = depth[u] = d;
    int deg = 0;
    for (int v : vs[u]) if (v != p) {
        deg += depth[v] == -1;
        if (depth[v] != -1) {
            int r = dfs(v, u, d + 1);
            low = min(low, r);
        } else {
            int r = dfs(v, u, d + 1);
            low = min(low, r);
            if (r >= depth[u] && p != -1)
                articulation[u] = true;
        }
    }
    if (p == -1)
        articulation[u] = deg > 1;
    return low;
}
```

## 3.8 Arbre couvrant minimal

### Kruskal

```
int n;
int par[N];

int rep(int u) {
    return par[u] = par[u] == u? u : rep(par[u]);
}

vector<int> kruskal(vector<tuple<int, int, int, int>> edge) {
    sort(edge.begin(), edge.end());
    iota(par, par + n, 0);
    vector<int> mst;
    for (auto [c, u, v, e] : edge) if (rep(u) != rep(v)) {
        par[rep(u)] = rep(v);
        mst.emplace_back(e);
    }
    return mst;
}
```

### Prim

```
int n, m;
vector<tuple<int, int, int>> vs[N];
bool visited[N];
vector<int> mst;

void prim() {
    priority_queue<tuple<int, int, int>> q;
```

```
    q.emplace(-0, 0, -1);
    while (!q.empty()) {
        auto [c, u, e] = q.top();
        q.pop();
        if (!visited[u]) {
            visited[u] = true;
            if (e != -1)
                mst.push_back(e);
            for (auto [v, d, f] : vs[u])
                q.emplace(-d, v, f);
        }
    }
}
```

## 3.9 Couplage maximal

```
int nl, nr;
vector<int> vs[NL]; // vs[ul].push_back(vr);
int lmatch[NL];
int rmatch[NR];
int visited[NR], iter;

bool dfs(int u) {
    if (visited[u] < iter) {
        visited[u] = iter;
        for (int v : vs[u]) {
            if (rmatch[v] == -1 || dfs(rmatch[v])) {
                lmatch[u] = v;
                rmatch[v] = u;
                return true;
            }
        }
    }
    return false;
}

int maxmatching() {
    fill(lmatch, lmatch + nl, -1);
    fill(rmatch, rmatch + nr, -1);
    int m = 0, dm = 1;
    while (dm > 0) {
        dm = 0;
        iter++;
        for (int u = 0; u < nl; u++) if (lmatch[u] == -1)
            dm += dfs(u);
        m += dm;
    }
    return m;
}
```

## 3.10 Flot maximal : Ford Fulkerson

```
int n, m, source, sink;
vector<tuple<int, int, int, int>> vs[N];
// vs[u].emplace_back(v, arc_id, c, +1);
// vs[v].emplace_back(u, arc_id, 0, -1);
int flow[M];
int visited[N], iter;

bool dfs(int u, int f) {
    if (u == sink)
        return true;
    else if (visited[u] < iter) {
        visited[u] = iter;
        for (auto [v, a, c, k] : vs[u]) if (c - k * flow[a] >= f &&
↪  dfs(v, f)) {
            flow[a] += k * f;
            return true;
        }
    }
    return false;
}

int maxflow() {
    int f = 0;
    int df = 1 << 20; // >= cmax
    while (df > 0) {
        iter++;
        if (dfs(source, df))
            f += df;
        else
            df /= 2;
    }
    return f;
}
```

Scaling capacity : $O(n^2 \log c_{\max})$

# 4 Structures de données

## 4.1 Dichotomie

```
// dernier 0 de predicat[l..r[ en supposant croissance
if (predicat(l))
    return l - 1;
while (r - l > 1) {
    int m = (l + r) / 2;
    if (predicat(m))
        r = m;
    else
        l = m;
}
return l;
```

## 4.2 Tri et statistiques

### STL

```
sort(begin, end, [cmp]);
min_element(begin, end, [cmp]);
max_element(begin, end, [cmp]);
nth_elemnt(begin, begin + nth, end, [cmp]); // put nth in place
random_shuffle(begin, end);
unique(begin, end); // place les doublons à la fin si trié
lower_bound(begin, end, val); // premier >= val si trié
upper_bound(begin, end, val); // premier > val si trié
```

### Tri fusion

```
void mergesort(int* a, int l, int r) {
    if (r - l > 1) {
        int m = (l + r) / 2;
        mergesort(a, l, m);
        mergesort(a, m, r);
        int p[m - l]; copy(a + l, a + m, p);
        int q[r - m]; copy(a + m, a + r, q);
        for (int i = l, j = 0, k = 0; i < r; i++) {
            if (j == m - l) a[i] = q[k++];
            else if (k == r - m) a[i] = p[j++];
            else if (p[j] <= q[k]) a[i] = p[j++];
            else a[i] = q[k++]; // + (m - l - j) inversions
        }
    }
}
```

## 4.3 Disjoint Set Union : Union Find

```
int par[N]; // init : iota(par, par + n, 0);
int siz[N]; // init : fill(siz, siz + n, 1);

int rep(int u) {
    if (par[u] == u) return u;
    return par[u] = rep(par[u]);
}

bool merge(int u, int v) {
    u = rep(u), v = rep(v);
    if (u == v) return false;
    if (siz[u] < siz[v]) swap(u, v);
    par[v] = u;
    siz[u] += siz[v];
    return true;
}
```

## 4.4 Arbre binaire

```
const int N = 1 << 17;
int t[2 * N];

void build() {
    for (int u = N - 1; u >= 1; u--)
        t[u] = t[2 * u] + t[2 * u + 1];
}

void setval(int u, int x)
{
    u += N;
    t[u] = x;
    while (u > 1)
    {
        u /= 2;
        t[u] = t[2 * u] + t[2 * u + 1];
    }
}

int sum(int u, int v) { // sum on [u:v[
    int s = 0;
    u += N, v += N;
```

```
    while (u < v) {
        if (u & 1) s += t[u++];
        if (v & 1) s += t[-v];
        u /= 2, v /= 2;
    }
    return s;
}
```

## 4.5 Arbre binaire avec propagation paresseuse

```
const int N = 1 << 17;
struct {int s; bool lazy; int x;} t[2 * N];

void update(int u, int n) {
    if (t[u].lazy && u < N)
        t[2 * u] = t[2 * u + 1] = {t[u].x * n / 2, true, t[u].x};
    t[u].lazy = false;
}

void setval(int x, int L, int R, int u = 1, int l = 0, int r = N) {
    update(u, r - l);
    if (L <= l && r <= R)
        t[u] = {(r - l) * x, true, x};
    else if (!(r <= L || R <= l)) {
        int m = (l + r) / 2;
        setval(x, L, R, 2 * u, l, m);
        setval(x, L, R, 2 * u + 1, m, r);
        t[u].s = t[2 * u].s + t[2 * u + 1].s;
    }
}

int sum(int L, int R, int u = 1, int l = 0, int r = N) {
    update(u, r - l);
    if (L <= l && r <= R)
        return t[u].s;
    else if (r <= L || R <= l)
        return 0;
    int m = (l + r) / 2;
    return sum(L, R, 2 * u, l, m) + sum(L, R, 2 * u + 1, m, r);
}
```

## 4.6 Arbre binaire persistant

```
const int N = 1 << 17; // power of 2

struct Node {int s; Node *v, *w;};

Node* setval(Node *u, int i, int x, int l = 0, int r = N) {
    if (!u) u = new Node {0, nullptr, nullptr};
    else u = new Node {0, u->v, u->w};
    if (r - l == 1)
        u->s = x;
    else {
        int m = (l + r) / 2;
        if (i < m) u->v = setval(u->v, i, x, l, m);
        else u->w = setval(u->w, i, x, m, r);
        u->s = (u->v? u->v->s : 0) + (u->w? u->w->s : 0);
    }
    return u;
}

int sum(Node *u, int L, int R, int l = 0, int r = N) {
    if (!u || r <= L || l >= R) return 0;
    else if (L <= l && r <= R) return u->s;
    int m = (l + r) / 2;
    return sum(u->v, L, R, l, m) + sum(u->w, L, R, m, r);
}
```

## 4.7 Arbre cartésien

```
struct Treap {
    Treap *l, *r;
    int x;
    int y;
};

Treap* merge(Treap *u, Treap *v) {
    if (!u) return v;
    else if (!v) return u;
    else if (u->y > v->y) {
        u->r = merge(u->r, v);
        return u;
    }
    else {
        v->l = merge(u, v->l);
```

```
        return v;
    }
}

pair<Treap*, Treap*> split(Treap *u, int x) {
    if (!u)
        return {nullptr, nullptr};
    else if (u->x <= x) {
        auto [v, w] = split(u->r, x);
        u->r = v;
        return {u, w};
    }
    else {
        auto [v, w] = split(u->l, x);
        u->l = w;
        return {v, u};
    }
}

Treap* insert(Treap *u, int x) {
    int y = rand();
    auto [v, w] = split(u, x);
    Treap *t = new Treap {nullptr, nullptr, x, y};
    return merge(v, merge(t, w));
}

Treap* erase(Treap *u, int x) {
    if (u->x == x)
        return merge(u->l, u->r);
    else if (x < u->x)
        return erase(u->l, x);
    else
        return erase(u->r, x);
}
```

## 4.8 Décomposition heavy light

```
vector<int> vs[N];
int par[N];
int heavy[N];
int head[N];
int pos[N];
int curpos;
```

```
int dfs(int u, int p = -1) {
    par[u] = p;
    heavy[u] = -1;
    int k = 1, mk = 0;
    for (int v : vs[u]) if (v != p) {
        int vk = dfs(v, u);
        k += vk;
        if (vk > mk)
            mk = vk, heavy[u] = v;
    }
    return k;
}

void heavylight(int u) {
    pos[u] = curpos++;
    if (heavy[u] != -1) {
        head[heavy[u]] = head[u];
        heavylight(heavy[u]);
    }
    for (int v : vs[u]) if (v != par[u] && v != heavy[u]) {
        head[v] = v;
        heavylight(v);
    }
}

void query(int u, int v) {
    while (head[u] != head[v]) {
        if (pos[u] > pos[v])
            swap(u, v);
        /* query on [ pos[head[v]] ... pos[v] + 1 [ */
        v = par[head[v]];
    }
    if (pos[u] > pos[v])
        swap(u, v);
    /* query on [ pos[u] + 1 ... pos[v] + 1 [ */
}

// curpos = 0;
// dfs(root);
// head[root] = root;
// heavylight(root);
```

## 4.9 Range minimum query

```
const int LOG_N = 17;
const int N = 1 << LOG_N;
const int oo = INT_MAX;

int range[LOG_N + 1][N];

void build(vector<int>& a) {
    int n = a.size();
    fill_n((int*)range, (LOG_N + 1) * N, +oo);
    for (int i = 0; i < n; i++)
        range[0][i] = a[i];
    for (int k = 0; k < LOG_N; k++) {
        for (int i = 0; i < n; i++) {
            int j = i + (1 << k);
            range[k + 1][i] = min(
                    range[k][i],
                    j < n? range[k][j] : +oo
                );
        }
    }
}

int rmq(int l, int r) {
    int k = 0;
    int n = r - l;
    while (n > 1)
        k++, n /= 2;
    return min(range[k][l], range[k][r - (1 << k)]);
}
```

## 4.10 Plus petit ancêtre commun

### Binary lifting

```
const int LOG_N = 17;
const int N = 1 << LOG_N;

int n;
vector<int> childs[N];
int depth[N];
int par[LOG_N + 1][N];

void dfs(int u) {
    for (int v : childs[u]) {
        par[0][v] = u;
        depth[v] = depth[u] + 1;
        dfs(v);
    }
}

void build(int root) {
    fill_n((int*)par, (LOG_N + 1) * N, root);
    par[0][root] = root;
    depth[root] = 0;
    dfs(root);
    for (int k = 0; k < LOG_N; k++)
        for (int u = 0; u < n; u++)
            par[k + 1][u] = par[k][par[k][u]];
}

int lca(int u, int v) {
    if (depth[u] > depth[v])
        swap(u, v);
    for (int k = LOG_N; k >= 0; k--)
        if ((depth[v] - depth[u]) & (1 << k))
            v = par[k][v];
    if (u == v)
        return u;
    for (int k = LOG_N; k >= 0; k--)
        if (par[k][u] != par[k][v])
            u = par[k][u], v = par[k][v];
    return par[0][u];
}
```

### RMQ

```
vector<int> childs[N];
int depth[N], pos[N];
vector<pair<int, int>> euler;

void dfs(int u) {
    pos[u] = euler.size();
    euler.emplace_back(depth[u], u);
    for (int v : childs[u]) {
        depth[v] = depth[u] + 1;
        dfs(v);
        euler.emplace_back(depth[u], u);
```

```
    }
}

// dfs(root);
// [d, lca] = rmq(min(pos[u], pos[v]), max(pos[u], pos[v]) + 1);
```

# 5 Chaînes de caractères

## 5.1 Knuth–Morris–Pratt

$$\pi(i) = \max\{j < i, s[0 \dots j[= s[i - j + 1 \dots i + 1[\}$$

```
vector<int> prefix_function(string s) {
    int n = s.size();
    vector<int> pi(n);
    for (int i = 1; i < n; i++) {
        int j = pi[i - 1];
        while (j > 0 && s[i] != s[j])
            j = pi[j - 1];
        if (s[i] == s[j])
            j++;
        pi[i] = j;
    }
    return pi;
}
```

## 5.2 Fonction Z

$$z(i) = \max\{j \geqslant 1, s[i \dots i + j[= s[0 \dots j[\}$$

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

## 5.3 Suffix array

```
vector<int> suffixarray(string s) {
    int n = s.size();
    vector<int> suf(n), rank(n, 0);
    iota(suf.begin(), suf.end(), 0);
    for (int i = 0; i < n; i++)
        rank[i] = s[i];
    for (int k = 1; k < n; k *= 2) {
        auto lt = [&] (int i, int j) -> bool {
            int ri = i + k < n? rank[i + k] : 0;
            int rj = j + k < n? rank[j + k] : 0;
            return rank[i] < rank[j] || (rank[i] == rank[j] && ri <
↪   rj);
        };
        sort(suf.begin(), suf.end(), lt);
        int r = 0, p = suf[0];
        for (int i : suf) if (lt(p, i)) {
            p = i;
            rank[i] = r++;
        }
    }
    return suf;
}
```

## 5.4 Aho-Corasick

```
const int K = 26;

struct Vertex {
    int next[K];
    bool leaf = false;
    int p = -1;
    char pch;
    int link = -1;
    int go[K];

    Vertex(int p=-1, char ch='$') : p(p), pch(ch) {
        fill(begin(next), end(next), -1);
        fill(begin(go), end(go), -1);
    }
};

vector<Vertex> t(1);
```

```
void add_string(string const& s) {
    int v = 0;
    for (char ch : s) {
        int c = ch - 'a';
        if (t[v].next[c] == -1) {
            t[v].next[c] = t.size();
            t.emplace_back(v, ch);
        }
        v = t[v].next[c];
    }
    t[v].leaf = true;
}

int go(int v, char ch);

int get_link(int v) {
    if (t[v].link == -1) {
        if (v == 0 || t[v].p == 0)
            t[v].link = 0;
        else
            t[v].link = go(get_link(t[v].p), t[v].pch);
    }
    return t[v].link;
}

int go(int v, char ch) {
    int c = ch - 'a';
    if (t[v].go[c] == -1) {
        if (t[v].next[c] != -1)
            t[v].go[c] = t[v].next[c];
        else
            t[v].go[c] = v == 0 ? 0 : go(get_link(v), ch);
    }
    return t[v].go[c];
}
```

## 5.5 Automate des suffixes

$\text{endpos}(u) = \{i, s[i - |u| + 1 \ldots i] = u\}$ la classe de $u$ i.e. l'état de l'automate

$\text{link}(u) = \text{endpos}(v)$ avec $v$ plus grand suffixe de $u$ tq $\text{endpos}(u) \subsetneq \text{endpos}(v)$

```
tuple<vector<map<char, int>>, vector<int>, vector<bool>>
↪ automaton(string s) {
    int n = s.size();
    vector<int> len(2 * n);
    vector<map<char, int>> next(2 * n);
    vector<int> link(2 * n);
    int sz = 1, last = 0;
    link[0] = -1;
    for (char c : s) {
        int cur = sz++;
        len[cur] = len[last] + 1;
        int u = last;
        while (u != -1 && next[u].count(c) == 0) {
            next[u][c] = cur;
            u = link[u];
        }
        if (u == -1)
            link[cur] = 0;
        else if (int v = next[u][c]; len[v] == len[u] + 1)
            link[cur] = v;
        else {
            int w = sz++;
            len[w] = len[u] + 1;
            next[w] = next[v];
            link[w] = link[v];
            link[v] = link[cur] = w;
            while (u != -1 && next[u][c] == v)
            {
                next[u][c] = w;
                u = link[u];
            }
        }
        last = cur;
    }
    next.resize(sz);
    len.resize(sz);
    vector<bool> suf(sz);
    while (last != -1) suf[last] = true, last = link[last];
    return {next, len, suf};
}
```

# 6 Géométrie

## 6.1 Formules

1. $u^\perp = iu$
2. $u \cdot v = \Re u \Re v + \Im u \Im v = \Re(\bar{u}v) = |u|\,|v| \cos(u, v)$
3. $\det(u, v) = \Re u \Im v - \Im u \Re v = \Im(\bar{u}v) = |u|\,|v| \sin(u, v)$
4. Formule de Cramer $2 \times 2$ : $tu + sv = a \iff t = \dfrac{\det(a, v)}{\det(u, v)}$ et $s = \dfrac{\det(u, a)}{\det(u, v)}$
5. Projeté de $c$ sur $(ab)$ :
$$h = a + \frac{(c-a)\cdot(b-a)}{|b-a|^2}(b-a)$$
6. $|h - a| = \dfrac{|(c-a)\cdot(b-a)|}{|b-a|}$
7. $|h - c| = \dfrac{|\det(c-a, b-a)|}{|b-a|}$
8. $a$, $b$ et $c$ sont alignés si $\det(b-a, c-a) = 0$
9. La droite $c$ et $d$ sont de part et d'autre de $(ab)$ si :
$$\det(b-a, c-a)\det(b-a, d-a) \leqslant 0$$
10. Les segments $[ab]$ et $[cd]$ se coupent si :
$$\det(b-a, c-a)\det(b-a, d-a) \leqslant 0 \text{ et } \det(d-c, a-c)\det(d-c, b-c) \leqslant 0$$
11. Si $a + \mathbb{R}u$ et $b + \mathbb{R}v$ sont deux droites
    — Elles sont confondues si $u = v$ et $\det(b-a, u) = 0$
    — Elles sont parallèles si $u = v$ et $\det(b-a, u) \neq 0$
    — Elles se coupent en $z = a + \dfrac{\det(a-b, v)}{\det(u, v)}u$ sinon
12. Le milieu de $[ab]$ est $\dfrac{a+b}{2}$ et la médiatrice est $\dfrac{a+b}{2} + \mathbb{R}i(b-a)$
13. Si $a$, $b$ et $c$ sont trois points non alignés alors le centre du cercle passant par $a$, $b$ et $c$ est
$$\frac{1}{2}\left(a + b + \frac{(c-b)(c-a)}{\det(b-a, c-a)}i(b-a)\right)$$
14. L'aire du triangle $(abc)$ est $\dfrac{1}{2}|\det(b-a, c-a)|$
15. L'aire du polygone $(a_1 \dots a_n)$ est avec $a_{n+1} = a_1$
$$\left|\sum_{i=1}^{n} \frac{1}{2}(\Re a_{i+1} - \Re a_i)(\Im a_i + \Im a_{i+1})\right|$$
16. Théorème de Pick : soit un polygone $A$. Alors
$$|A| = \left|A^\circ \cap \mathbb{Z}^2\right| + \frac{1}{2}\left|\partial A \cap \mathbb{Z}^2\right| - 1$$

## 6.2 Bases

```
typedef complex<double> Z;
double det(Z u, Z v) {return imag(conj(u) * v);}
double dot(Z u, Z v) {return real(conj(u) * v);}
const Z I(0, 1);

const double EPSILON = 1e-6;
bool fzero(double x) {return abs(x) < EPSILON;}
bool feq(double x, double y) {return abs(x - y) < EPSILON;}
bool flt(double x, double y) {return x < y - EPSILON;}
bool fleq(double x, double y) {return x <= y + EPSILON;}
bool fgt(double x, double y) {return x > y + EPSILON;}
bool fgeq(double x, double y) {return x >= y - EPSILON;}
// norm(u); |u|^2
// abs(u); |u|^2

bool ltarg(Z u, Z v) {
    if (!(abs(u) > 0))
        return abs(v) > 0;
    bool pu = imag(u) > 0 || !(imag(u) < 0) && real(u) > 0;
    bool pv = imag(v) > 0 || !(imag(v) < 0) && real(v) > 0;
    if (pu != pv)
        return pu < pv;
    return det(u, v) > 0;
}
```

`ltarg` compare les arguments principaux dans $[-\pi, \pi[$ avec $\arg 0 = -\pi$.

## 6.3 Convex hull trick

Soient trois droites $f_i(x) = a_i x + b_i$ avec $a_1 < a_2 < a_3$. Alors

$$\forall x, f_2(x) < \max(f_1(x), f_3(x)) \iff \Re(f_1 \cap f_2) < \Re(f_1 \cap f_3)$$

avec

$$\Re(f_1 \cap f_2) = -\frac{b_2 - b_1}{a_2 - a_1}$$

### 6.4 Enveloppe convexe : scan de Graham

```
vector<Z> grahamscan(vector<Z> p) {
    Z o(1e18, 1e18);
    for (auto& z : p)
        if (real(z) < real(o) || !(real(z) > real(o)) && imag(z) <
↪   imag(o))
            o = z;
    sort(p.begin(), p.end(), [&] (Z u, Z v) -> bool {return ltarg(u -
↪   o, v - o);});
    vector<Z> c;
    for (auto& z : p) {
        while (c.size() > 2 && det(c[c.size() - 2] - c.back(), z -
↪   c.back()) >= 0)
            c.pop_back();
        c.push_back(z);
    }
    return c;
}
```

### 6.5 Tester si un point est dans un polygone

```
bool inside(vector<Z> &a, Z z) {
    int n = a.size();
    for (int i = 0; i < n; i++) {
        Z p = a[i], q = a[(i + 1) % a.size()];
        double t = dot(q - p, z - p);
        double s = norm(q - p);
        if (fzero(det(q - p, z - p)) && fgeq(t, 0) && fleq(t, s))
            return true; // sur un cote
    }
    Z u; // tirer une demi droite qui ne coupe pas un sommets de a
    bool dir = true;
    while (dir) {
        double theta = 2 * acos(-1) * (double)rand() / RAND_MAX;
        u = Z(cos(theta), sin(theta));
        dir = false;
        for (Z p : a) if (fzero(det(p - z, u)))
            dir = true;
    }
    int k = 0;
    for (int i = 0; i < n; i++) {
        Z p = a[i], q = a[(i + 1) % n];
        if (!fzero(det(p - q, u))) {
            double t = det(p - z, u) / det(p - q, u);
            double s = det(p - q, p - z) / det(p - q, u);
            k += fgeq(t, 0) && fleq(t, 1) && fgeq(s, 0);
        }
    }
    return k % 2;
}
```

# 7 Algèbre

### 7.1 Théorème des restes chinois

Si $n_1, \dots n_r$ sont des entiers deux à deux premiers entre eux alors

$$\Phi : x[n_1 \dots n_r] \in \mathbb{Z}/(n_1 \dots n_r)\mathbb{Z} \mapsto (x[n_1], \dots, x[n_r]) \in \mathbb{Z}/n_1\mathbb{Z} \times \dots \times \mathbb{Z}/n_r\mathbb{Z}$$

est un isomorphisme d'anneaux d'inverse avec $n_j u_{ij} \equiv 1[n_i]$

$$\Phi^{-1}(a_1[n_1], \dots, a_r[n_r]) = \sum_{i=1}^{r} \left( \prod_{j \neq i} n_j u_{ij} \right) a_i[n_1 \dots n_r]$$

### 7.2 Indicatrice d'Euler

$$\phi(n) = |\mathbb{Z}/n\mathbb{Z}^*| = \#\left\{k \in \{1, \dots, n\}, k \wedge n = 1\right\}$$

1. $\varphi$ est multiplicative : si $n \wedge m = 1$ alors $\varphi(nm) = \varphi(n)\varphi(m)$
2. $\varphi(p^k) = p^k - p^{k-1}$ si $p$ est premier

```
void eulertotient(int n) {
    vector<int> phi(n + 1);
    for (int i = 0; i <= n; i++)
        phi[i] = i;
    for (int i = 2; i <= n; i++) {
        if (phi[i] == i) {
            for (int j = i; j <= n; j += i)
                phi[j] -= phi[j] / i;
        }
    }
    return phi;
}
```

## 7.3 Inversion de Möbius

Soit $\mathcal{A} = \{f, f : \mathbb{N}^* \to \mathbb{C}\}$ qu'on munit de + et $*$ la convolution de Dirichlet

$$(f * g)(n) = \sum_{ij=n} f(i)g(j) = \sum_{d|n} f(d)g(n/d)$$

$\mathcal{A}$ est un anneau commutatif de neutre multiplicatif $\delta_1$. On pose la fonction de Möbius

$$\mu(p_1^{k_1} \dots p_r^{k_r}) = \begin{cases} (-1)^r \text{ si } \forall i, k_i = 1 \\ \qquad 0 \text{ sinon} \end{cases}$$

On a les propriétés

1. $1 * \mu = \delta_1$
2. $1 * \varphi = \mathrm{id}$
3. $\varphi$ est multiplicative : si $n \wedge m = 1$ alors $\varphi(nm) = \varphi(n)\varphi(m)$
4. $\mu$ est multiplicative
5. une convolée de deux fonctions multiplicatives l'est

## 7.4 Euclide et inverse modulaire

```
// g++ function __gcd(a, b)
// gcd(a0, a1) = a0 u0 + a1 v0
int euclid(int a0, int a1, int& u0, int& v0) {
    u0 = 1, v0 = 0;
    int u1 = 0, v1 = 1;
    while (a1 != 0)
    {
        int q = a0 / a1;
        a0 -= q * a1; swap(a0, a1);
        u0 -= q * u1; swap(u0, u1);
        v0 -= q * v1; swap(v0, v1);
    }
    return a0;
}
```

## 7.5 Crible d'Ératosthène

```
// find prime numbers in O(n log log n)
vector<bool> sieve(int n) {
    vector<bool> prime(n + 1, true);
    prime[0] = prime[1] = false;
    for (int i = 2; i * i <= n; i++) if (prime[i])
        for (int j = i * i; j <= n; j += i)
            prime[j] = false;
    return prime;
}

// find lowest prime factor and prime numbers in O(n)
vector<int> lpfactor(int n) {
    vector<int> lp(n + 1, 0), prime;
    for (int i = 2; i <= n; i++) if (lp[i] == 0) {
        lp[i] = i;
        pr.push_back(i);
    }
    for (int p : prime) if(p <= lp[i] && i * p <= n)
        lp[i * p] = p;
    return lp;
}
```

## 7.6 Exponentiation rapide

```
int fastexp(int x, int n) {
    if (n == 0)
        return 1;
    int y = fastexp(x, n / 2);
    return (n & 1? x : 1) * y * y;
}
```

## 7.7 FFT

```
void fft(vector<complex<double>>& a, bool invert) {
    int n = a.size();
    if (n == 1) return;
    vector<complex<double>> a0(n / 2), a1(n / 2);
    for (int i = 0; 2 * i < n; i++) {
        a0[i] = a[2 * i];
        a1[i] = a[2 * i + 1];
    }
    fft(a0, invert);
    fft(a1, invert);
    double ang = 2 * acos(-1) / n * (invert ? -1 : 1);
    complex<double> w(1), wn(cos(ang), sin(ang));
    for (int i = 0; 2 * i < n; i++) {
        a[i] = a0[i] + w * a1[i];
        a[i + n/2] = a0[i] - w * a1[i];
        w *= wn;
    }
```

```
}
```

Ne pas oublier de diviser par $N$ pour la FFT inverse. Ne marche que pour les puissances de 2 (taille du polynôme). Pour multiplier des polynômes, multiplier terme à terme les coefficients de Fourier. Ne pas oublier qu'il faut garder une marge d'un facteur 2 pour que le polynôme produit puisse aussi passer en sens inverse.

## 7.8 Pivot de Gauss

```
const double EPSILON = 1e-6;

bool solvegauss(vector<vector<double>> A, vector<double> b,
↪   vector<double> &x) {
    int m = A.size();
    int n = A[0].size();
    for (int p = 0; p < min(n, m); p++) {
        int i = p, k = p;
        for (int i = p; i < m; i++)
            if (abs(A[i][p]) > abs(A[k][p]))
                k = p;
        swap(A[k], A[p]);
        swap(b[k], b[p]);
        if (abs(A[p][p]) > EPSILON) {
            double k = A[p][p];
            for (int j = 0; j < n; j++)
                A[p][j] /= k;
            b[p] /= k;
            for (int i = 0; i < m; i++) if (i != p) {
                double k = A[i][p];
                for (int j = 0; j < n; j++)
                    A[i][j] -= k * A[p][j];
                b[i] -= k * b[p];
            }
        }
    }
    x.resize(n);
    for (int p = 0; p < min(n, m); p++) {
        if (A[p][p] > EPSILON) x[p] = b[p];
        else if (abs(b[p]) > EPSILON) return false;
    }
    for (int p = n; p < m; p++) if (abs(b[p]) > EPSILON)
        return false;
    return true;
}
```

## 7.9 Simplexe

$$\min c^\top x \text{ sous les contraintes } Ax \leqslant b \text{ et } x \geqslant 0$$

```
typedef vector<double> Vec;
typedef vector<Vec> Mat;
enum Status {OPTIMAL, INFEASIBLE, UNBOUNDED};
const double EPSILON = 1e-5;

void pivot(Mat& A, Vec& b, Vec& c, Vec& d, int* N, int* B, int k, int
↪   l) {
    int m = b.size();
    int n = c.size();
    swap(B[k], N[l]);
    double a = A[k][l];
    for (int i = 0; i < m; i++) if (i != k) {
        for (int j = 0; j < n; j++) if (j != l)
            A[i][j] -= A[i][l] * A[k][j] / a;
        b[i] -= A[i][l] * b[k] / a;
        A[i][l] /= -a;
    }
    for (int j = 0; j < n; j++) if (j != l) {
        c[j] -= c[l] * A[k][j] / a;
        d[j] -= d[l] * A[k][j] / a;
        A[k][j] /= a;
    }
    A[k][l] = 1 / a;
    b[k] /= a;
    c[l] /= -a;
    d[l] /= -a;
}

Status simplexiters(Mat& A, Vec& b, Vec& c, Vec& d, int* N, int* B,
↪   Vec& x, int xfea) {
    int m = b.size();
    int n = c.size();
    while (true) {
        int l = -1;
        for (int j = 0; j < n; j++)
            if (N[j] != xfea && c[j] < -EPSILON && (l == -1 || N[j] <
↪   N[l]))
                l = j;
        if (l == -1)
            break;
```

```
        double t = 1e9;
        int k = -1;
        for (int i = 0; i < m; i++)
            if (A[i][l] > 0 && b[i] / A[i][l] < t)
                t = b[i] / A[i][l], k = i;
        if (k == -1)
            return UNBOUNDED;
        pivot(A, b, c, d, N, B, k, l);
    }
    x.resize(n + m);
    fill(x.begin(), x.end(), 0);
    for (int i = 0; i < m; i++)
        x[B[i]] = b[i];
    return OPTIMAL;
}

Status simplex(Mat A, Vec b, Vec c, Vec& x) {
    int m = b.size();
    int n = c.size();
    for (int i = 0; i < m; i++)
        A[i].push_back(-1);
    c.push_back(0);
    Vec d(n + 1);
    d[n] = 1;
    int N[n + 1]; iota(N, N + n + 1, 0);
    int B[m]; iota(B, B + m, n + 1);
    int k = 0;
    for (int i = 0; i < m; i++) if (b[i] < b[k])
        k = i;
    if (b[k] < 0) { // feasability
        pivot(A, b, c, d, N, B, k, n);
        assert(simplexiters(A, b, d, c, N, B, x, -1) == OPTIMAL);
        if (x[n] > EPSILON)
            return INFEASIBLE;
        int k = -1;
        for (int i = 0; i < m; i++) if (B[i] == n)
            k = i;
        if (k != -1) {
            int l = 0;
            for (int j = 0; j < n + 1; j++)
                if (abs(A[k][j]) > abs(A[k][l]))
                    l = j;
            assert(abs(A[k][l]) > EPSILON);
            pivot(A, b, c, d, N, B, k, l);
        }
    }
    if (simplexiters(A, b, c, d, N, B, x, n) == UNBOUNDED)
        return UNBOUNDED;
    x.resize(n);
    return OPTIMAL;
}
```

## 7.10 Big Int

```
// github williamchanrico biginteger-cpp

class BigInt {
public:
    int sign;
    string s;

    BigInt() : s("") {}

    BigInt(string x) {*this = x;}

    BigInt(int x) {*this = to_string(x);}

    BigInt negative() {
        BigInt x = *this;
        x.sign *= -1;
        return x;
    }

    BigInt normalize(int newSign) {
        for (int a = s.size() - 1; a > 0 && s[a] == '0'; a-)
            s.erase(s.begin() + a);
        sign = (s.size() == 1 && s[0] == '0' ? 1 : newSign);
        return *this;
    }

    void operator = (string x) {
        int newSign = (x[0] == '-' ? -1 : 1);
        s = (newSign == -1 ? x.substr(1) : x);
        reverse(s.begin(), s.end());
        this->normalize(newSign);
    }
```

```cpp
    bool operator == (const BigInt& x) const {
        return (s == x.s && sign == x.sign);
    }

    bool operator < (const BigInt& x) const {
        if (sign != x.sign)
            return sign < x.sign;
        if (s.size() != x.s.size())
            return (sign == 1 ? s.size() < x.s.size() : s.size() > x.s.size());
        for (int a = s.size() - 1; a >= 0; a--)
            if (s[a] != x.s[a])
                return (sign == 1 ? s[a] < x.s[a] : s[a] > x.s[a]);
        return false;
    }

    bool operator <= (const BigInt& x) const {
        return (*this < x || *this == x);
    }

    bool operator > (const BigInt& x) const {
        return (!(*this < x) && !(*this == x));
    }

    bool operator >= (const BigInt& x) const {
        return (*this > x || *this == x);
    }

    BigInt operator + (BigInt x) {
        BigInt curr = *this;
        if (curr.sign != x.sign)
            return curr - x.negative();
        BigInt res;
        for (int a = 0, carry = 0; a < s.size() || a < x.s.size() || carry; a++) {
            carry += (a < curr.s.size() ? curr.s[a] - '0' : 0) + (a < x.s.size() ? x.s[a] - '0' : 0);
            res.s += (carry % 10 + '0');
            carry /= 10;
        }
        return res.normalize(sign);
    }

    BigInt operator - (BigInt x) {
        BigInt curr = *this;
        if (curr.sign != x.sign)
            return curr + x.negative();
        int realSign = curr.sign;
        curr.sign = x.sign = 1;
        if (curr < x)
            return ((x - curr).negative()).normalize(-realSign);
        BigInt res;
        for (int a = 0, borrow = 0; a < s.size(); a++) {
            borrow = (curr.s[a] - borrow - (a < x.s.size() ? x.s[a] : '0'));
            res.s += (borrow >= 0 ? borrow + '0' : borrow + '0' + 10);
            borrow = (borrow >= 0 ? 0 : 1);
        }
        return res.normalize(realSign);
    }

    BigInt operator * (BigInt x) {
        BigInt res("0");
        for (int a = 0, b = s[a] - '0'; a < s.size(); a++, b = s[a] - '0') {
            while (b--)
                res = (res + x);
            x.s.insert(x.s.begin(), '0');
        }
        return res.normalize(sign * x.sign);
    }

    BigInt operator / (BigInt x) {
        if (x.s.size() == 1 && x.s[0] == '0')
            x.s[0] /= (x.s[0] - '0');
        BigInt temp("0"), res;
        for (int a = 0; a < s.size(); a++)
            res.s += "0";
        int newSign = sign * x.sign;
        x.sign = 1;
        for (int a = s.size() - 1; a >= 0; a--) {
            temp.s.insert(temp.s.begin(), '0');
            temp = temp + s.substr(a, 1);
            while (!(temp < x)) {
                temp = temp - x;
                res.s[a]++;
            }
```

```
        }
        return res.normalize(newSign);
    }

    BigInt operator % (BigInt x) {
        if (x.s.size() == 1 && x.s[0] == '0')
            x.s[0] /= (x.s[0] - '0');
        BigInt res("0");
        x.sign = 1;
        for (int a = s.size() - 1; a >= 0; a–) {
            res.s.insert(res.s.begin(), '0');
            res = res + s.substr(a, 1);
            while (!(res < x))
                res = res - x;
        }
        return res.normalize(sign);
    }

    string toString() const {
        string ret = s;
        reverse(ret.begin(), ret.end());
        return (sign == -1 ? "-" : "") + ret;
    }

    BigInt toBase10(int base) {
        BigInt exp(1), res("0"), BASE(base);
        for (int a = 0; a < s.size(); a++) {
            int curr = (s[a] < '0' || s[a] > '9' ? (toupper(s[a]) - 'A' + 10) : (s[a] - '0'));
            res = res + (exp * BigInt(curr));
            exp = exp * BASE;
        }
        return res.normalize(sign);
    }

    BigInt toBase10(int base, BigInt mod) {
        BigInt exp(1), res("0"), BASE(base);
        for (int a = 0; a < s.size(); a++) {
            int curr = (s[a] < '0' || s[a] > '9' ? (toupper(s[a]) - 'A' + 10) : (s[a] - '0'));
            res = (res + ((exp * BigInt(curr) % mod)) % mod);
            exp = ((exp * BASE) % mod);
        }
        return res.normalize(sign);
    }

    string convertToBase(int base) {
        BigInt ZERO(0), BASE(base), x = *this;
        string modes = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
        if (x == ZERO)
            return "0";
        string res = "";
        while (x > ZERO) {
            BigInt mod = x % BASE;
            x = x - mod;
            if (x > ZERO)
                x = x / BASE;
            res = modes[stoi(mod.toString())] + res;
        }
        return res;
    }

    BigInt toBase(int base) {
        return BigInt(this->convertToBase(base));
    }

    friend ostream& operator«(ostream& os, const BigInt& x) {
        os « x.toString();
        return os;
    }
};
```