Fibonacci

$$F_n = \sum_{k=0}^{\lfloor \frac{n-1}{2} \rfloor} \binom{n-k-1}{k}$$

$$\sum_{j=0}^{n} F_j = F_{n+2} - 1$$

$$\sum_{j=0}^{n} F_j^2 = F_n F_{n+1}$$

$$\sum_{j=1}^{n} F_{2j} = F_{2n+1} - 1$$

$$\sum_{j=1}^{n} F_{2j-1} = F_{2n}$$

$$\sum_{j=1}^{2n-1} F_j F_{j+1} = F_{2n}^2$$

$$\sum_{j=1}^{2n} F_j F_{j+1} = F_{2n+1}^2 - 1$$

$$F_{n+1} F_{n-1} - F_n^2 = (-1)^n$$

$$F_n^2 - F_{n-r} F_{n+r} = (-1)^{n-r} F_r^2$$

$$F_{n+k} F_{m-k} - F_n F_m = (-1)^n F_{m-n-k} F_k$$

$$(-1)^n F_{m-n} = F_m F_{n+1} - F_n F_{n-1}$$

$$(-1)^n F_{m+n} = F_{m-1} F_n - F_m F_{n+1}$$

$$m \mid n \Leftrightarrow F_m \mid F_n$$

$$\gcd(F_m, F_n) = F_{\gcd(m,n)}$$

Combinatorics

- Binomial

$$\binom{r}{k} = (-1)^k \binom{k-r-1}{k}$$

$$\binom{n}{m} = (-1)^{n-m} \binom{-m-1}{n-m}$$

$$\binom{n}{k-1} + \binom{n}{k} = \binom{n+1}{k}$$

$$\sum_{i=0}^{n} \binom{n}{i} = 2^n$$

$$\sum_{i \geq 0} \binom{n}{2i} = 2^{n-1}$$

$$\sum_{k \leq n} (-1)^k \binom{r}{k} = (-1)^n \binom{r-1}{n}$$

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

$$\sum_{j=0}^{m} \binom{n+j}{n} = \binom{n+m+1}{m}$$

$$\sum_{j=0}^{n} \binom{j}{m} = \binom{n+1}{m+1}$$

$$\sum_{i=0}^{n} i \binom{n}{i} = n 2^{n-1}$$

$$\sum_{k=0}^{n} \binom{r}{k} \binom{s}{n-k} = \binom{r+s}{n}$$

- Catalan $C_n = \frac{1}{n+1} \binom{2n}{n} = \sum_{k=0}^{n-1} C_k C_{n-1-k}$
- Next combination

```
bool next_combination(vector<int>& a, int n) {
    int k = (int)a.size();
    for (int i = k - 1; i >= 0; i--) {
```

```
        if (a[i] < n - k + i + 1) {
            a[i]++;
            for (int j = i + 1; j < k; j++)
                a[j] = a[j - 1] + 1;
            return true;
        }
    }
    return false;
}
```

- Number of elements in exactly r set

$$\sum_{m=r}^{n}(-1)^{m-r}\binom{m}{r}\sum_{\mid X\mid=m}\mid\cup_{i\in X}A_i\mid$$

- Burnside's lemma

$$\mid \text{Classes} \mid = \frac{1}{\mid G\mid}\sum_{\pi\in G}\text{Number of fixed points of transformation }\pi$$

- Bishop placement

```
int bishop_placements(int N, int K)
{
    if (K > 2 * N - 1)
        return 0;
    vector<vector<int>> D(N * 2, vector<int>(K + 1));
    for (int i = 0; i < N * 2; ++i)
        D[i][0] = 1;
    D[1][1] = 1;
    for (int i = 2; i < N * 2; ++i)
        for (int j = 1; j <= K; ++j)
            D[i][j] = D[i-2][j] + D[i-2][j-1] * (squares(i) - j + 1);
    int ans = 0;
    for (int i = 0; i <= K; ++i)
        ans += D[N*2-1][i] * D[N*2-2][K-i];
    return ans;
}
```

- Number of labeled graph $G_n = \text{pow}\left(2, \frac{n(n+1)}{2}\right)$
- Number of connected labeled graphs

$$C_n = G_n - \frac{1}{n}\sum_{k=1}^{n-1}k\binom{n}{k}C_kG_{n-k}$$

- Number of graphs with n nodes and k connected components

$$D[n][k] = \sum_{s=1}^{n}\binom{n-1}{s-1}C_sD[n-s][k-1]$$

Number theory

- Generalized lucas

```cpp
const int MOD = 27, prime = 3;
long long fact[MOD], ifact[MOD];
void init(){
    fact[0] = 1;
    for (int i = 1; i < MOD; i++) {
        if (i % prime != 0)
            fact[i] = (fact[i - 1] * i) % MOD;
        else
            fact[i] = fact[i - 1];
    }
    int phi = MOD / prime * (prime - 1) - 1;
    ifact[MOD - 1] = binpow(fact[MOD - 1], phi, MOD);
    for (int i = MOD - 1; i > 0; i--) {
        if (i % prime != 0)
            ifact[i - 1] = (ifact[i] * i) % MOD;
        else
            ifact[i - 1] = ifact[i];
    }
}
long long C(long long N, long long K, long long R){
    return (fact[N] * ifact[R] % MOD) * ifact[K] % MOD;
}
int count_carry(long long n, long long k, long long r, int p, long long t){
    long long res = 0;
    while (n >= t) {
        res += ((n / t) - (k / t) - (r / t));
        t *= p;
    }
    return res;
}
long long calc(long long N, long long K, long long R) {
    if (K > N)
        return 0;
    long long res = 1;
    int vp = count_carry(N, K, R, prime, prime);
    int vp2 = count_carry(N, K, R, prime, MOD);
    while (N > 0) {
        res = (res * C(N % MOD, K % MOD, R % MOD)) % MOD;
        N /= prime; K /= prime; R /= prime;
    }
    res = res * binpow(prime, vp, MOD) % MOD;
    if ((vp2 % 2 == 1) && (prime != 2 || MOD <= 4))
        res = (MOD - res) % MOD;
    return res;
}
```

- Gray code

$$g = n \oplus (n \gg 1)$$

```cpp
int rev_g (int g)
  int n = 0;
  for (; g; g >>= 1) n ^= g;
```

- Discrete log

$$a^x = b(\operatorname{mod} m) \Rightarrow a^{np-q} = b(\operatorname{mod} m)$$
$$n = \lfloor\sqrt{m}\rfloor + 1, p \in [1, n], q \in [0, n]$$

- Discrete root

$$x^k = a(\operatorname{mod} n) \Rightarrow \left(g^k\right)^y = a(\operatorname{mod} n)$$
$$g \text{ is primitive root of } n$$

- Chinese remainder theorem $x = \sum_{i=1}^{n} \left(r_i M_i M_i^{-1}\right)(\operatorname{mod} M)$

Linear algebra

- Gauss - Jordan

```
template <class T> int gauss(vector<vector <T>> eqs, vector<T>& res, const T
eps=1e-12){
    int n = eqs.size(), m = eqs[0].size() - 1;
    int i, j, k, l, p, f_var = 0;

    res.assign(m, 0);
    vector <int> pos(m, -1);

    for (j = 0, i = 0; j < m && i < n; j++){
        for (k = i, p = i; k < n; k++){
            if (abs(eqs[k][j]) > abs(eqs[p][j])) p = k;
        }

        if (abs(eqs[p][j]) > eps){
            pos[j] = i;
            for (l = j; l <= m; l++) swap(eqs[p][l], eqs[i][l]);

            for (k = 0; k < n; k++){
                if (k != i){
                    T x = eqs[k][j] / eqs[i][j];
                    for (l = j; l <= m; l++) eqs[k][l] -= eqs[i][l] * x;
                }
            }
            i++;
        }
    }

    for (i = 0; i < m; i++){
        if (pos[i] == -1) f_var++;
        else res[i] = eqs[pos[i]][m] / eqs[pos[i]][i];
    }

    for (i = 0; i < n; i++) {
        T val = 0;
        for (j = 0; j < m; j++) val += res[j] * eqs[i][j];
        if (abs(val - eqs[i][m]) > eps) return -1;
    }
```

```
    return f_var;
}
```

Geometry

- Common tangents of (O, r_1) and (I, r_2) $(ax + by + c = 0)$

$$d_2 = \pm r_1, d_1 = \pm r_2, c = d_1$$

$$a = \frac{(d_2 - d_1)I_x + I_y\sqrt{I_x^2 + I_y^2 - (d_2 - d_1)^2}}{I_x^2 + I_y^2} \quad b = \frac{(d_2 - d_1)I_y + I_x\sqrt{I_x^2 + I_y^2 - (d_2 - d_1)^2}}{I_x^2 + I_y^2}$$

- Convex hull

```
struct Point {
    int64_t x, y; /// x*x or y*y should not overflow
    Point(){}
    Point(int64_t x, int64_t y) : x(x), y(y) {}
    inline bool operator < (const Point &p) const {
        return ((x < p.x) || (x == p.x && y < p.y));
    }
};
int64_t cross(const Point &O, const Point &A, const Point &B){
    return ((A.x - O.x) * (B.y - O.y)) - ((A.y - O.y) * (B.x - O.x));
}
vector<Point> get_convex_hull(vector<Point> P){
    int i, t, k = 0, n = P.size();
    vector<Point> H(n << 1);
    sort(P.begin(), P.end());
    for (i = 0; i < n; i++) {
        while (k >= 2 && cross(H[k - 2], H[k - 1], P[i]) < 0) k--;
        H[k++] = P[i];
    }
    for (i = n - 2, t = k + 1; i >= 0; i--) {
        while (k >= t && cross(H[k - 2], H[k - 1], P[i]) < 0) k--;
        H[k++] = P[i];
    }
    H.resize(k - 1);
    return H;
}
bool is_convex(vector <Point> P){
    int n = P.size();
    if (n <= 2) return false; /// Line or point is not convex
    n++, P.push_back(P[0]);  /// Last point = first point
    bool flag1 = (cross(P[0], P[1], P[2]) > 0);
    for (int i = 1; (i + 1) < n; i++){
        bool flag2 = (cross(P[i], P[i + 1], (i + 2) == n ? P[1] : P[i + 2]) >
0);
        if (flag1 ^ flag2) return false;
    }
```

```
    return true;
}
```

- Check point in convex polygon

```
struct pt {
    long long x, y;
    pt() {}
    pt(long long _x, long long _y) : x(_x), y(_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); }
    long long cross(const pt &p) const { return x * p.y - y * p.x; }
    long long dot(const pt &p) const { return x * p.x + y * p.y; }
    long long cross(const pt &a, const pt &b) const { return (a -
*this).cross(b - *this); }
    long long dot(const pt &a, const pt &b) const { return (a - *this).dot(b
- *this); }
    long long sqrLen() const { return this->dot(*this); }
};
bool lexComp(const pt &l, const pt &r) {
    return l.x < r.x || (l.x == r.x && l.y < r.y);
}
int sgn(long long val) { return val > 0 ? 1 : (val == 0 ? 0 : -1); }
vector<pt> seq;
pt translation;
int n;
bool pointInTriangle(pt a, pt b, pt c, pt point) {
    long long s1 = abs(a.cross(b, c));
    long long s2 = abs(point.cross(a, b)) + abs(point.cross(b, c)) +
abs(point.cross(c, a));
    return s1 == s2;
}
void prepare(vector<pt> &points) {
    n = points.size();
    int pos = 0;
    for (int i = 1; i < n; i++) {
        if (lexComp(points[i], points[pos]))
            pos = i;
    }
    rotate(points.begin(), points.begin() + pos, points.end());

    n--;
    seq.resize(n);
    for (int i = 0; i < n; i++)
        seq[i] = points[i + 1] - points[0];
    translation = points[0];
}
bool pointInConvexPolygon(pt point) {
    point = point - translation;
    if (seq[0].cross(point) != 0 &&
            sgn(seq[0].cross(point)) != sgn(seq[0].cross(seq[n - 1])))
```

```
        return false;
    if (seq[n - 1].cross(point) != 0 &&
            sgn(seq[n - 1].cross(point)) != sgn(seq[n - 1].cross(seq[0])))
        return false;

    if (seq[0].cross(point) == 0)
        return seq[0].sqrLen() >= point.sqrLen();

    int l = 0, r = n - 1;
    while (r - l > 1) {
        int mid = (l + r) / 2;
        int pos = mid;
        if (seq[pos].cross(point) >= 0)
            l = mid;
        else
            r = mid;
    }
    int pos = l;
    return pointInTriangle(seq[pos], seq[pos + 1], pt(0, 0), point);
}
```

- Planar graphs: $|\text{ vertices }| - |\text{ edges }| + |\text{ faces (or regions) }| = 2$
- Pick's theorem

$$S = I + \frac{B}{2} - 1$$

I: the number of points with integer coordinates lying strictly inside
B: the number of points lying on polygon sides

- Shoelace formula

$$S = \frac{1}{2} \sum_{i=1}^{n} (y_i + y_{i+1})(x_i - x_{i+1}) = \sum_{p,q \in \text{ edges}} \frac{(p_x - q_x)(p_y + q_y)}{2}$$

- Manhattan distance

$$d((x_1, y_1), (x_2, y_2)) = \max(|\ (x_1 + y_1) - (x_2 + y_2)\ |, |\ (x_1 - y_1) - (x_2 - y_2)\ |)$$

- Segment intersection

```
const double EPS = 1E-9;
struct seg {
    pt p, q;
    int id;
    double get_y(double x) const {
        if (abs(p.x - q.x) < EPS)
            return p.y;
        return p.y + (q.y - p.y) * (x - p.x) / (q.x - p.x);
    }
};
bool intersect1d(double l1, double r1, double l2, double r2) {
    if (l1 > r1)
```

```
        swap(l1, r1);
    if (l2 > r2)
        swap(l2, r2);
    return max(l1, l2) <= min(r1, r2) + EPS;
}
int vec(const pt& a, const pt& b, const pt& c) {
    double s = (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x - a.x);
    return abs(s) < EPS ? 0 : s > 0 ? +1 : -1;
}
bool intersect(const seg& a, const seg& b)
{
    return intersect1d(a.p.x, a.q.x, b.p.x, b.q.x) &&
           intersect1d(a.p.y, a.q.y, b.p.y, b.q.y) &&
           vec(a.p, a.q, b.p) * vec(a.p, a.q, b.q) <= 0 &&
           vec(b.p, b.q, a.p) * vec(b.p, b.q, a.q) <= 0;
}
bool operator<(const seg& a, const seg& b)
{
    double x = max(min(a.p.x, a.q.x), min(b.p.x, b.q.x));
    return a.get_y(x) < b.get_y(x) - EPS;
}
struct event {
    double x;
    int tp, id;

    event() {}
    event(double x, int tp, int id) : x(x), tp(tp), id(id) {}

    bool operator<(const event& e) const {
        if (abs(x - e.x) > EPS)
            return x < e.x;
        return tp > e.tp;
    }
};
set<seg> s;
vector<set<seg>::iterator> where;
set<seg>::iterator prev(set<seg>::iterator it) {
    return it == s.begin() ? s.end() : --it;
}
set<seg>::iterator next(set<seg>::iterator it) {
    return ++it;
}
pair<int, int> solve(const vector<seg>& a) {
    int n = (int)a.size();
    vector<event> e;
    for (int i = 0; i < n; ++i) {
        e.push_back(event(min(a[i].p.x, a[i].q.x), +1, i));
        e.push_back(event(max(a[i].p.x, a[i].q.x), -1, i));
    }
    sort(e.begin(), e.end());
```

```
    s.clear();
    where.resize(a.size());
    for (size_t i = 0; i < e.size(); ++i) {
        int id = e[i].id;
        if (e[i].tp == +1) {
            set<seg>::iterator nxt = s.lower_bound(a[id]), prv = prev(nxt);
            if (nxt != s.end() && intersect(*nxt, a[id]))
                return make_pair(nxt->id, id);
            if (prv != s.end() && intersect(*prv, a[id]))
                return make_pair(prv->id, id);
            where[id] = s.insert(nxt, a[id]);
        } else {
            set<seg>::iterator nxt = next(where[id]), prv = prev(where[id]);
            if (nxt != s.end() && prv != s.end() && intersect(*nxt, *prv))
                return make_pair(prv->id, nxt->id);
            s.erase(where[id]);
        }
    }
    return make_pair(-1, -1);
}
```

RMQ

```
// RMQ
void preprocess() {
    for (int i = 1; i <= n; ++i) RMQ[0][i] = a[i];
    for (int j = 1; j <= LG; ++j)
        for (int i = 1; i + (1 << j) - 1 <= n; ++i)
            RMQ[j][i] = min(RMQ[j - 1][i], RMQ[j - 1][i + (1 << (j - 1))]);
}
int getMinRange(int l, int r) {
    int lg = __lg(r - l + 1);
    return min(RMQ[lg][l], RMQ[lg][r - (1 << lg) + 1]);
}
// RSQ
void preprocess() {
    for (int i = 1; i <= n; ++i) RSQ[0][i] = a[i];
    for (int j = 1; j <= LG; ++j)
        for (int i = 1; i + (1 << j) - 1 <= n; ++i)
            RSQ[j][i] = RSQ[j - 1][i] + RSQ[j - 1][i + (1 << (j - 1))];
}
int getSumRange(int l, int r) {
    int len = r - l + 1;
    int lg = __lg(len) + 1;
    int sum = 0;
    for (int i = 0; (1 << i) <= len; ++i) {
        if ((len >> i) & 1) {
            sum += RSQ[i][l];
            l += (1 << i);
        }
```

```
    }
    return sum;
}
```

LCA

```
void dfs(int u) {
    for (auto i : g[u]) {
        int v = i.second;
        int uv = i.first;
        if (v == up[u][0]) continue;
        height[v] = height[u] + 1;
        up[v][0] = u;
        min_d[v][0] = uv;
        max_d[v][0] = uv;
        for (int j = 1; j < LOG; ++j) {
            min_d[v][j] = min(min_d[v][j - 1], min_d[up[v][j - 1]][j - 1]);
            max_d[v][j] = max(max_d[v][j - 1], max_d[up[v][j - 1]][j - 1]);
            up[v][j] = up[up[v][j - 1]][j - 1];
        }
        dfs(v);
    }
}
pair<int, int> lca(int u, int v) {
    pair<int, int> res = {1e9, -1e9};
    if (height[u] < height[v]) swap(u, v);
    int diff = height[u] - height[v];
    for (int i = 0; (1 << i) <= diff; ++i)
        if (diff & (1 << i)) {
            res.first = min(res.first, min_d[u][i]);
            res.second = max(res.second, max_d[u][i]);
            u = up[u][i];
        }
    if (u == v) return res;
    int k = __lg(height[u]);
    for (int i = k; i >= 0; --i) {
        if (up[u][i] != up[v][i]) {
            res.first = min({res.first, min_d[u][i], min_d[v][i]});
            res.second = max({res.second, max_d[u][i], max_d[v][i]});

            u = up[u][i];
            v = up[v][i];
        }
    }
    res.first = min({res.first, min_d[u][0], min_d[v][0]});
    res.second = max({res.second, max_d[u][0], max_d[v][0]});
    return res;
}
```

Trie

```
class Trie {
    struct Node {
        Node* child[2];
        int cnt;
        int exist;
        Node() : cnt(0), exist(0) {
            for (int ch = 0; ch < 2; ++ch)
                child[ch] = nullptr;
        }
        ~Node() {
            for (int ch = 0; ch < 2; ++ch)
                delete child[ch];
        }
    };
    Node* root;
    Trie() : root(new Node()) {}
    ~Trie() {
        delete root;
    }

    void insert(const int& num) {
        Node* cur = root;
        for (int i = 31; i >= 0; --i) {
            int nxt = (num >> i) & 1;
            if (cur->child[nxt] == nullptr) {
                cur->child[nxt] = new Node();
            }
            cur = cur->child[nxt];
            cur->cnt++;
        }
        cur->exist++;
    }
    bool find(const int& num) {
        Node* cur = root;
        for (int i = 31; i >= 0; --i) {
            int nxt = (num >> i) & 1;
            if (cur->child[nxt] == nullptr) return false;
            cur = cur->child[nxt];
        }
        return cur->exist > 0;
    }
    bool erase_recursive(Node* current, const int& num, int idx) {
        if (idx != 0) {
            int nxt = (num >> idx) & 1;
            bool is_child_deleted = erase_recursive(current->child[nxt], num,
idx - 1);
            if (is_child_deleted) current->child[nxt] = nullptr;
        } else {
            current->exist--;
        }
```

```cpp
        if (current != root) {
            current->cnt--;
            if (current->cnt == 0) {
                delete current;
                return true; // deleted
            }
        }
        return false;
    }
    bool erase(const int& num) {
        if (!find(num)) return false;
        return !erase_recursive(root, num, 31);
    }
    int find_max_xor(const int& num) {
        Node* cur = root;
        int res = 0;
        for (int i = 31; i >= 0; --i) {
            int nxt = (num >> i) & 1;
            if (cur->child[nxt ^ 1] != nullptr) {
                cur = cur->child[nxt ^ 1];
                res |= (1 << i);
            } else {
                cur = cur->child[nxt];
            }
        }
        return res;
    }
};


class Trie {
    struct Node {
        Node* child[26];
        int cnt;
        int exist;
        Node() : cnt(0), exist(0) {
            for (int ch = 0; ch < 26; ++ch)
                child[ch] = nullptr;
        }
        ~Node() {
            for (int ch = 0; ch < 26; ++ch)
                delete child[ch];
        }
    };
    Node* root;
    Trie() : root(new Node()) {}
    ~Trie() {
        delete root;
    }
    void insert(const string& s) {
```

```
        Node* cur = root;
        for (auto c : s) {
            int nxt = c - 'a';
            if (cur->child[nxt] == nullptr) {
                cur->child[nxt] = new Node();
            }
            cur = cur->child[nxt];
            cur->cnt++;
        }
        cur->exist++;
    }
    bool find(const string& s) {
        Node* cur = root;
        for (auto c : s) {
            int nxt = c - 'a';
            if (cur->child[nxt] == nullptr) return false;
            cur = cur->child[nxt];
        }
        return cur->exist > 0;
    }
    bool erase_recursive(Node* current, const string& s, int idx) {
        if (idx != s.size()) {
            int nxt = s[idx] - 'a';
            bool is_child_deleted = erase_recursive(current->child[nxt], s,
idx + 1);
            if (is_child_deleted) current->child[nxt] = nullptr;
        } else {
            current->exist--;
        }

        if (current != root) {
            current->cnt--;
            if (current->cnt == 0) {
                delete current;
                return true; // deleted
            }
        }
        return false;
    }
    bool erase(const string& s) {
        if (!find(s)) return false;
        return !erase_recursive(root, s, 0);
    }
};

struct aho_corasick{
    struct node{
        int suffix_link = -1, cnt = 0, nxt[26], go[26];
        node() {fill(nxt, nxt+26, -1);}
    };
    vector<node> g = {node()};
```

```
    void build_automaton(){
        for (deque<int> q = {0}; q.size(); q.pop_front()){
            int v = q.front(), suffix_link = g[v].suffix_link;
            for (int i=0; i<26; i++){
                int nxt = g[v].nxt[i], go_sf = v ? g[suffix_link].go[i] : 0;
                if (nxt == -1) g[v].go[i] = go_sf;
                else{
                    g[v].go[i] = nxt;
                    g[nxt].suffix_link = go_sf;
                    q.push_back(nxt);
                }
            }
        }
    }
};
```

KMP

```
int k = kmp[1] = 0;
for (int i = 2; i <= n; ++i) {
    while (k > 0 && S[i] != S[k + 1]) k = kmp[k];
    if (S[i] == S[k + 1]) kmp[i] = ++k;
    else kmp[i] = 0;
}
k = 0;
for (int i = 1; i <= m; ++i) {
    while (k > 0 && T[i] != S[k + 1]) k = kmp[k];
    if (T[i] == S[k + 1]) match[i] = ++k;
    else match[i] = 0;
    // Found S in T[i - length(S) + 1..i]
    if (match[i] == n) {
        cout << i - n + 1 << ' ';
    }
}
```

Rabin - Karp

```
vector<int> rabin_karp(string const& s, string const& t) {
    const int p = 31;
    const int m = 1e9 + 9;
    int S = s.size(), T = t.size();
    vector<long long> p_pow(max(S, T));
    p_pow[0] = 1;
    for (int i = 1; i < (int)p_pow.size(); i++)
        p_pow[i] = (p_pow[i-1] * p) % m;
    vector<long long> h(T + 1, 0);
    for (int i = 0; i < T; i++)
        h[i+1] = (h[i] + (t[i] - 'a' + 1) * p_pow[i]) % m;
    long long h_s = 0;
    for (int i = 0; i < S; i++)
        h_s = (h_s + (s[i] - 'a' + 1) * p_pow[i]) % m;
```

```
    vector<int> occurrences;
    for (int i = 0; i + S - 1 < T; i++) {
        long long cur_h = (h[i+S] + m - h[i]) % m;
        if (cur_h == h_s * p_pow[i] % m)
            occurrences.push_back(i);
    }
    return occurrences;
}
```

Z function

```
vector<int> z_function(string s) {
    int n = s.length();
    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;
}
```

Miller Rabin

```
bool MillerTest(long long d, long long n) {
    long long a = rand() % (n - 4) + 2;
    long long x = powmod(a, d, n);
    if (x == 1 || x == n - 1) return true;
    while (d != n - 1) {
        x = (x * x) % n;
        d <<= 1;
        if (x == 1) return false;
        if (x == n - 1) return true;
    }
    return false;
}
bool isPrime(long long n, int k = 100) {
    if (n <= 1 || n == 4)  return false;
    if (n <= 3) return true;
    long long d = n - 1;
    while (d%2 == 0) d /= 2;
    while (k--) {
        if (!MillerTest(d, n)) return false;
    }
    return true;
}
```

Minkowski sum

```
void reorder_polygon(vector<pt> & P){
    size_t pos = 0;
    for(size_t i = 1; i < P.size(); i++){
        if(P[i].y < P[pos].y || (P[i].y == P[pos].y && P[i].x < P[pos].x))
            pos = i;
    }
    rotate(P.begin(), P.begin() + pos, P.end());
}

vector<pt> minkowski(vector<pt> P, vector<pt> Q){
    // the first vertex must be the lowest
    reorder_polygon(P);
    reorder_polygon(Q);
    // we must ensure cyclic indexing
    P.push_back(P[0]);
    P.push_back(P[1]);
    Q.push_back(Q[0]);
    Q.push_back(Q[1]);
    // main part
    vector<pt> result;
    size_t i = 0, j = 0;
    while(i < P.size() - 2 || j < Q.size() - 2){
        result.push_back(P[i] + Q[j]);
        auto cross = (P[i + 1] - P[i]).cross(Q[j + 1] - Q[j]);
        if(cross >= 0 && i < P.size() - 2)
            ++i;
        if(cross <= 0 && j < Q.size() - 2)
            ++j;
    }
    return result;
}
```

Fenwick tree

```
void range_add(int l, int r, ll x) {
    ft1.update(l, x);
    ft1.update(r + 1, -x);
    ft2.update(l, x * (l - 1));
    ft2.update(r + 1, -x * r);
}

ll prefix_sum(int idx) {
    return ft1.get_sum(idx) * idx - ft2.get_sum(idx);
}
```

Max xor subset

```
long long max_xor_subset(const vector<long long>& ar){
    long long m, x, res = 0;
```

```cpp
    int i, j, l, n = ar.size();

    vector <long long> v[64];
    for (i = 0; i < n; i++) v[bitlen(ar[i])].push_back(ar[i]);

    for (i = 63; i > 0; i--){
        l = v[i].size();
        if (l){
            m = v[i][0];
            res = max(res, res ^ m);

            for (j = 1; j < l; j++){
                x = m ^ v[i][j];
                if (x) v[bitlen(x)].push_back(x);
            }
            v[i].clear();
        }
    }

    return res;
}
```

Manacher

```cpp
vector <int> manacher(const string& str){
    int i, j, k, l = str.size(), n = l << 1;
    vector <int> pal(n);

    for (i = 0, j = 0, k = 0; i < n; j = max(0, j - k), i += k){
        while (j <= i && (i + j + 1) < n && str[(i - j) >> 1] == str[(i + j +
1) >> 1]) j++;
        for (k = 1, pal[i] = j; k <= i && k <= pal[i] && (pal[i] - k) !=
pal[i - k]; k++){
            pal[i + k] = min(pal[i - k], pal[i] - k);
        }
    }
    pal.pop_back();
    return pal;
}
```

MEX

```cpp
class Mex {
    map<int, int> frequency;
    set<int> missing_numbers;
    vector<int> A;
    Mex(const vector<int>& A) : A(A) {
        for (int i = 0; i <= A.size(); i++)
            missing_numbers.insert(i);

        for (int x : A) {
```

```
            ++frequency[x];
            missing_numbers.erase(x);
        }
    }
    int mex() {
        return *missing_numbers.begin();
    }
    void update(int idx, int value) {
        if (--frequency[A[idx]] == 0)
            missing_numbers.insert(A[idx]);
        A[idx] = value;
        ++frequency[value];
        missing_numbers.erase(value);
    }
};
```

FFT

```
using cd = complex<double>; const double PI = acos(-1);
void fft(vector<cd> & a, bool invert) {
    int n = a.size();
    if (n == 1)
        return;
    vector<cd> 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 * PI / n * (invert ? -1 : 1);
    cd 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];
        if (invert) {
            a[i] /= 2;
            a[i + n/2] /= 2;
        }
        w *= wn;
    }
}
vector<int> multiply_polynomial(vector<int> const& a, vector<int> const& b) {
    vector<cd> fa(a.begin(), a.end()), fb(b.begin(), b.end());
    int n = 1;
    while (n < a.size() + b.size())
        n <<= 1;
    fa.resize(n);
    fb.resize(n);
```

```
    fft(fa, false);
    fft(fb, false);
    for (int i = 0; i < n; i++)
        fa[i] *= fb[i];
    fft(fa, true);

    vector<int> result(n);
    for (int i = 0; i < n; i++)
        result[i] = round(fa[i].real());
    return result;
}
```

Graph

- Max flow

```
struct Edge{
    int u, v;
    long long cap, flow;

    Edge(){}
    Edge(int u, int v, long long cap, long long flow) : u(u), v(v), cap(cap),
flow(flow) {}
};

struct FlowGraph{
    vector <int> adj[MAXN];
    vector <struct Edge> E;
    int n, src, sink, Q[MAXN], ptr[MAXN], dis[MAXN];

    FlowGraph(){}
    FlowGraph(int n, int src, int sink): n(n), src(src), sink(sink) {}

    void add_directed_edge(int u, int v, long long cap){
        adj[u].push_back(E.size());
        E.push_back(Edge(u, v, cap, 0));

        adj[v].push_back(E.size());
        E.push_back(Edge(v, u, 0, 0));
    }

    void add_edge(int u, int v, int cap){
        add_directed_edge(u, v, cap);
        add_directed_edge(v, u, cap);
    }

    bool bfs(){
        int u, f = 0, l = 0;
        memset(dis, -1, sizeof(dis[0]) * n);

        dis[src] = 0, Q[l++] = src;
```

```
        while (f < l && dis[sink] == -1){
            u = Q[f++];
            for (auto id: adj[u]){
                if (dis[E[id].v] == -1 && E[id].flow < E[id].cap){
                    Q[l++] = E[id].v;
                    dis[E[id].v] = dis[u] + 1;
                }
            }
        }
        return dis[sink] != -1;
    }

    long long dfs(int u, long long flow){
        if (u == sink || !flow) return flow;

        int len = adj[u].size();
        while (ptr[u] < len){
            int id = adj[u][ptr[u]];
            if (dis[E[id].v] == dis[u] + 1){
                long long f = dfs(E[id].v, min(flow, E[id].cap -
E[id].flow));
                if (f){
                    E[id].flow += f, E[id ^ 1].flow -= f;
                    return f;
                }
            }
            ptr[u]++;
        }

        return 0;
    }

    long long maxflow(){
        long long flow = 0;

        while (bfs()){
            memset(ptr, 0, n * sizeof(ptr[0]));
            while (long long f = dfs(src, LLONG_MAX)){
                flow += f;
            }
        }

        return flow;
    }
};

struct FlowGraphWithNodeCap{
    FlowGraph flowgraph;

    FlowGraphWithNodeCap(int n, int src, int sink, vector <long long>
```

```
node_capacity){
        flowgraph = FlowGraph(2 * n, 2 * src, 2 * sink + 1);

        for (int i = 0; i < n; i++){
            flowgraph.add_directed_edge(2 * i, 2 * i + 1, node_capacity[i]);
        }
    }

    void add_directed_edge(int u, int v, long long cap){
        flowgraph.add_directed_edge(2 * u + 1, 2 * v, cap);
    }

    void add_edge(int u, int v, long long cap){
        add_directed_edge(u, v, cap);
        add_directed_edge(v, u, cap);
    }

    long long maxflow(){
        return flowgraph.maxflow();
    }
};
```

- Topo sort

```
int n; // number of vertices
vector<vector<int>> adj; // adjacency list of graph
vector<bool> visited;
vector<int> ans;
void dfs(int v) {
    visited[v] = true;
    for (int u : adj[v]) {
        if (!visited[u])
            dfs(u);
    }
    ans.push_back(v);
}

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

- Euler cycle

```
struct Edge {
    int target, id;
```

```
    Edge(int _target, int _id): target(_target), id(_id) {}
};
vector<Edge> adj[N];
bool used_edge[M];
list<int> euler_walk(int u) {
    list<int> ans;
    ans.push_back(u);
    while (!adj[u].empty()) {
        int v = adj[u].back().target;
        int eid = adj[u].back().id;
        adj[u].pop_back();
        if (used_edge[eid]) continue;
        used_edge[eid] = true;
        u = v;
        ans.push_back(u);
    }
    for (auto it = ++ans.begin(); it != ans.end(); ++it) {
        auto t = euler_walk(*it);
        t.pop_back();
        ans.splice(it, t);
    }
    return ans;
}
```

- Shortest path

```
vector<vector<ii>> AdjList;
int Dist[MaxN], Cnt[MaxN], S, N;
bool inqueue[MaxN];
queue<int> q;
bool spfa() {
    for(int i = 1 ; i <= N ; i++) {
        Dist[i] = Inf;
        Cnt[i] = 0;
        inqueue[i] = false;
    }
    Dist[S] = 0;
    q.push(S);
    inqueue[S] = true;
    while(!q.empty()) {
        int u = q.front();
        q.pop();
        inqueue[u] = false;
        for (ii tmp: AdjList[u]) {
            int v = tmp.first;
            int w = tmp.second;
            if (Dist[u] + w < Dist[v]) {
                Dist[v] = Dist[u] + w;
                if (!inqueue[v]) {
                    q.push(v);
                    inqueue[v] = true;
```

```
                    Cnt[v]++;
                    if (Cnt[v] > N)
                        return false;
                }
            }
        }
    }
    return true;
}
```

- Euler tour on tree

```
void add(int u) {
    tour[++m] = u;
    en[u] = m;
}
void dfs(int u, int parent_of_u) {
    h[u] = h[parent_of_u] + 1;
    add(u);
    st[u] = m;
    for (int v : adj[u]) {
        if (v != parent_of_u) {
            dfs(v, u);
        }
    }
    if (u != root) add(parent_of_u);
}
```

- Tarjan

```
int n, m, Num[N], Low[N], cnt = 0;
vector<int> a[N];
stack<int> st;
int Count = 0;
void visit(int u) {
    Low[u] = Num[u] = ++cnt;
    st.push(u);
    for (int v : a[u])
        if (Num[v])
            Low[u] = min(Low[u], Num[v]);
        else {
            visit(v);
            Low[u] = min(Low[u], Low[v]);
        }
    if (Num[u] == Low[u]) {  // found one
        Count++;
        int v;
        do {
            v = st.top();
            st.pop();
            Num[v] = Low[v] = oo;  // remove v from graph
        } while (v != u);
```

```
    }
}
int main() {
    for (int i = 1; i <= n; i++)
        if (!Num[i]) visit(i);
    cout << Count << endl;
}
```

Fenwick 2D

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
typedef pair<int, int> pii;
typedef tree<pii, null_type, less<pii>, rb_tree_tag,
tree_order_statistics_node_update> OST;
const int N = 200001;
OST bit[N];
void insert(int x, int y)
{
    for(int i = x; i < N; i += i & -i)
  bit[i].insert(make_pair(y, x));
}
void remove(int x, int y)
{
    for(int i = x; i < N; i += i & -i)
  bit[i].erase(make_pair(y, x));
}
int query(int x, int y)
{
    int ans = 0;
    for(int i = x; i > 0; i -= i & -i)
  ans += bit[i].order_of_key(make_pair(y+1, 0));
    return ans;
}
```

Misc

- Hash table

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
const int RANDOM =
chrono::high_resolution_clock::now().time_since_epoch().count();
struct chash {
    int operator()(int x) const { return x ^ RANDOM; }
};
gp_hash_table<int, int, chash> table;
```

- Josephus problem $J_{n,k} = \left(J_{n-1,k} + k\right) \bmod n$
- 15 puzzle game have solution if Number of inversion $+ \left\lfloor \frac{\text{Position of empty cell}}{4} \right\rfloor \bmod 2 = 0$

- k^{th} min element `nth_element(a.begin(), a.begin() + k, a.end(), [](int a, int b) return a < b);`
- Fast IO

```
nline char gc() {
   static char buf[1 << 16];
   static size_t bc, be;
   if (bc >= be) {
       buf[0] = 0, bc = 0;
       be = fread(buf, 1, sizeof(buf), stdin);
   }
   return buf[bc++];
}
inline void readInt(int& x) {
   int c;
   x = 0;
   while ((c = gc()) < '0' || c > '9');
   do {
       x = x * 10 + (c - '0');
   } while ((c = gc()) >= '0' && c <= '9');
}
char outbuf[20];
inline void writeInt(int x) {
   if (x == 0) {
       putchar('0');
       return;
   }
   int i = 0;
   while (x) {
       outbuf[i++] = static_cast<char>(x % 10) + '0';
       x /= 10;
   }
   while (i--) putchar(outbuf[i]);
}
```