Combinatorics

- Binomial

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

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

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

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

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

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

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

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

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

$$\sum_{i=0}^{n} i\binom{n}{i} = n2^{n-1}$$

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

- 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

$$\mid \text{Classes} \mid = \frac{1}{\mid G \mid} \sum_{\pi\in G} I(\pi)$$

$|G|$: count all permutation
π: a transformation with retain the class of a element
$I(\pi)$: number of fixed points of π

- Polýa enumeration theorem

$$| \text{ Classes } | = \frac{1}{| G |} \sum_{\pi \in G} k^{C(\pi)}$$

$C(\pi)$: number of cycles in π
k: number of values that each representation element can take

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

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

- Number of graphs with n nodes and k connected components

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

Fibonacci

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Number theory

- 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

$$x^k = a (\operatorname{mod} n) \Rightarrow \left(g^k\right)^y = a (\operatorname{mod} n)$$

$$g \text{ is primitive root of } n$$

- Gray code

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

```
int rev_g (int g) {
  int n = 0;
  for (; g; g >>= 1)
    n ^= g;
  return n;
}
```

Linear algebra

- Gauss - Jordan

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

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

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

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

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

Geometry

- Common tangents of (O, r_1) and (I, r_2)

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

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

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

$$c = d_1$$

- 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;
}
```

- 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

$$\text{manhattan}((x_1, y_1), (x_2, y_2)) = \max(|\ (x_1 + y_1) - (x_2 + y_2)\ |, |\ (x_1 - y_1) - (x_2 - y_2)\ |)$$

- 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_dist[v][0] = uv;
        max_dist[v][0] = uv;
        for (int j = 1; j < LOG; ++j) {
            min_dist[v][j] = min(min_dist[v][j - 1], min_dist[up[v][j - 1]][j
- 1]);
            max_dist[v][j] = max(max_dist[v][j - 1], max_dist[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_dist[u][i]);
            res.second = max(res.second, max_dist[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_dist[u][i], min_dist[v][i]});
            res.second = max({res.second, max_dist[u][i], max_dist[v][i]});

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

```cpp
    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--;
        }
        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);
    }
};
```

Aho - Corasick

```
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

```cpp
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

```cpp
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

```cpp
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]])
```

```cpp
            ++z[i];
        // update [l, r]
        if (i + z[i] - 1 > r) {
            l = i;
            r = i + z[i] - 1;
        }
    }
    return z;
}
```

Miller Rabin

```cpp
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

```cpp
struct pt{
    long long x, 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;
    }
};

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;
}
```

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 $+ \left\lfloor \frac{\text{Position of empty cell}}{4} \right\rfloor \bmod 2 = 0$