The Team Reference Document of UWr1

Spis treści

Generowanie sumy kontrolnej pliku:

```
cat plik | tr -d "[:space:]" | md5sum
```

1. Makefile

b5b7c66cc0f4193b9ec2c3f33ca878f3

```
CXXFLAGS = -Wall -Wextra -Wshadow -Wunused -std=c++17 -O3 -fsanitize=address
↪  -fsanitize=undefined
```

2. Pragma

dd7c731ec381c16234b3b75e11daf308

```
// wpisujemy na samym poczatku kodu
#pragma comment(linker, "/stack:200000000")
#pragma GCC optimize("O3")
#pragma GCC optimize("Ofast")
#pragma GCC optimize("unroll-loops")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,avx,avx2")
#pragma GCC target("popcnt,abm,mmx,avx,tune=native")
```

3. Vimrc

f2c727b8f588b09620aac2f114d1770f

```
syn on
set nu cin sw=4 ts=4
ino jk <ESC>
no <C-e> :w <bar> :!make %:r && ./%:r <cr>
no  <F4> :w <bar> :!make %:r && ./%:r <cr>
```

4. Policy-Based Data Structures

a6926ec516e543f30cf4af27198c7835

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace __gnu_pbds;

using ordered_set = tree< int, null_type, less<int>, rb_tree_tag,
↪  tree_order_statistics_node_update >;
using ordered_multiset = tree<int, null_type, less_equal<int>, rb_tree_tag,
↪  tree_order_statistics_node_update>;
using ordered_map = tree<int, int, less<int>, rb_tree_tag,
↪  tree_order_statistics_node_update>;

int main() {
    ordered_set X; X.insert(12);
    cout << *X.find_by_order(37) << X.order_of_key(5);

    ordered_map M;
    M[10] = 5;
    cout << M.find_by_order(0) -> second << " " << M.order_of_key(3);
}
```

5. RMQ

f1c82285d5563e4325f8b48d558ade25

```
// uzycie - RMQ < int, max > R(n, t);
template < typename TP, const TP& (*F)(const TP&, const TP&) >
struct RMQ {
        int n; vector < TP > t;
        vector < vector < TP > > res;
        RMQ(vector < TP > T) : n(sz(T)), t(T) {
                res.push_back(t);
                for(int p = 2; p <= n; p <<= 1) {
                        vector < TP > tmp(n - p + 1);
                        for(int j = 0; j <= n - p; j++)
                                tmp[j] = F(res.back()[j], res.back()[j + p / 2]);
                        res.push_back(tmp);
                }
        }
        TP query(int a, int b) {
                int d = b - a + 1, lg = 31 - __builtin_clz(d);
                return F(res[lg][a], res[lg][b - (1 << lg) + 1]);
        }
};
```

6. KD Tree

4176f5e9557c3f86af88fb8e33f32399

```
const int d = 2; ///dimension

struct point {
  int p[d];
  bool operator !=(const point &a) const {
    bool ok = 1;
    for(int i = 0; i < d; i++) ok &= (p[i] == a.p[i]);
    return !ok;
  }
};

struct kd_node {
  int axis, value;
  point p;
  kd_node *left, *right;
};

struct cmp_points {
  int axis;
  cmp_points() {}
  cmp_points(int x): axis(x) {}
  bool operator () (const point &a, const point &b) const {
    return a.p[axis] < b.p[axis];
  }
};

typedef kd_node* node_ptr;

int tests, n;
point arr[N], pts[N];
node_ptr root;
long long ans;

long long squared_distance(point a, point b) {
  long long ans = 0;
  for(int i = 0; i < d; i++) ans += (a.p[i] - b.p[i]) * 1ll * (a.p[i] - b.p[i]);
  return ans;
}

void build_tree(node_ptr &node, int from, int to, int axis) {
  if(from > to) {
    node = NULL;
    return;
  }

  node = new kd_node();

  if(from == to) {
    node->p = arr[from];
    node->left = NULL;
    node->right = NULL;
    return;
  }

  int mid = (from + to) / 2;

  nth_element(arr + from, arr + mid, arr + to + 1, cmp_points(axis));
  node->value = arr[mid].p[axis];
  node->axis = axis;
  build_tree(node->left, from, mid, (axis + 1) % d);
  build_tree(node->right, mid + 1, to, (axis + 1) % d);
}

void nearest_neighbor(node_ptr node, point q, long long &ans) {
  if(node == NULL) return;

  if(node->left == NULL && node->right == NULL) {
    if(q != node->p) ans = min(ans, squared_distance(node->p, q)); ///Beware!!! take
    ↪  care here
    return;
  }
```

```
  if(q.p[node->axis] <= node->value) {
    nearest_neighbor(node->left, q, ans);
    if(q.p[node->axis] + sqrt(ans) >= node->value) nearest_neighbor(node->right, q,
    ↪  ans);
  }

  else {
    nearest_neighbor(node->right, q, ans);
    if(q.p[node->axis] - sqrt(ans) <= node->value) nearest_neighbor(node->left, q, ans);
  }
}
```

7. Li Chao Tree

2c59c8cbf4632218545950f1afa7ef8d

```
/*
    minimum value of linear functions
    1 a b - add linear function f(x) = a * x + b
    2 x - find minimum value of f(x), 0 <= x <= 1e6
*/
namespace discreteLinear {
    typedef double T;
    const T INF = 1e15;
    struct line {
        T a, b;
        line(T A = 0, T B = INF) : a(A), b(B) {}
        T val(T x) { return a * x + b; }
    };
    const int MAX = 1 << 20;
    line tree[2 * MAX];
    void update(line f, int node = 1, int l = 0, int r = MAX - 1) {
        if(tree[node].val(l) > f.val(l)) swap(tree[node], f);
        if(tree[node].val(r) < f.val(r)) return;
        int m = (l + r) / 2;
        if(tree[node].val(m) < f.val(m))
            update(f, 2 * node + 1, m + 1, r);
        else
            swap(tree[node], f), update(f, 2 * node, l, m);
    }
    T query(int x, int node = 1, int l = 0, int r = MAX - 1) {
        if(l == r) return tree[node].val(x);
        int m = (l + r) / 2;
        if(x <= m) return min(tree[node].val(x), query(x, 2 * node, l, m));
        return min(tree[node].val(x), query(x, 2 * node + 1, m + 1, r));
    }
}
```

8. LCT Jarek

1d8a7c903746f7e7f981f5718cfa5468

```
struct node {
    typedef int T;
    static const T def = 0;
    node *left = 0, *right = 0, *up = 0, *path_parent = 0;
    T val, agg = def;
    bool rev = false;
    node(T v = def) : val(v) { update(); }
    static int f(int a, int b) {
        return max(a, b);
    }
    void update() {
        agg = f(left ? left->agg : def, f(val, right ? right->agg : def));
    }
    bool left_son() {
        return up->left == this;
    }
    void swap_ups(node *n) {
        n->up = up;
        n->path_parent = path_parent;
        up = n;
        path_parent = nullptr;
        n->update();
        if(n->up) {
            if(n->up->left == this) n->up->left = n;
            else n->up->right = n;
            n->up->update();
        }
    }
    void rot_left() {
        node *n = left;
        left = n->right;
        if(left) left->up = this;
        n->right = this;
        update();
        swap_ups(n);
    }
    void rot_right() {
        node *n = right;
        right = n->left;
        if(right) right->up = this;
        n->left = this;
        update();
        swap_ups(n);
    }
    void rot_up() {
        if(left_son()) up->rot_left();
        else up->rot_right();
    }
    void update_rev() {
        if(up) up->update_rev();
        if(rev) {
            rev = false;
            swap(left, right);
            if(left) left->rev = !left->rev;
            if(right) right->rev = !right->rev;
        }
    }
    node* go_up() {
        update_rev();
        while(up) {
            if(up->up)
                if(up->left_son() == left_son()) {
                    up->rot_up();
                    rot_up();
                }
                else {
                    rot_up();
                    rot_up();
                }
            else rot_up();
        }
        return this;
    }
    node* access() {
        node *last = nullptr;
        node *n = this;
        while(n) {
            n->go_up();
            if(n->left)
                swap(n->left->up, n->left->path_parent);
            n->left = last;
            if(last) swap(last->up, last->path_parent);
            n->update();
            last = n;
            n = n->path_parent;
        }
        go_up();
        return last;
    }
    void link(node *n) {
        access();
        rev = !rev;
        path_parent = n;
    }
    void cut() {
        access();
        if(right) {
            right->up = 0;
            right->path_parent = 0;
            right = 0;
        }
    }
    node* root() {
        access();
        node *n = this;
        while(n->right)
            n = n->right;
        n->go_up();
        return n;
    }
};
node* lca(node *a, node *b) {
    a->access();
    return b->access();
}
node::T on_path(node *a, node *b) {
    node *c = lca(a, b);
    a->go_up();
    c->go_up();
    if(a == c) return c->left->agg;
    if(b == c) return a->agg;
    return node::f(a->agg, c->left->agg);
}
node* insert(node *n, T k) {
    if(n == 0) return new node(k);
    n = n->splay(k);
    node *r = new node(k);
    if(n->val < k) {
        r->right = n->right; n->right = 0;
        if(r->right) r->right->up = r;
        r->left = n; n->up = r;
        n->update(); r->update();
    } else {
        r->left = n->left; n->left = 0;
        if(r->left) r->left->up = r;
        r->right = n; n->up = r;
        n->update(); r->update();
    }
    return r;
}
node* erase(node *n, T k) {
    if(n == 0) return 0;
    n = n->splay(k);
    if(n->val != k) return n;
    if(n->left == 0) {
        if(n->right == 0) return 0;
        n->right->up = 0;
        return n->right;
    }
    n->left->up = 0;
    node *r = n->left->splay(k);
    r->right = n->right;
    if(r->right) r->right->up = r;
    r->update(); delete n;
    return r;
}
```

9. LCT Bohun

3de26330f43cb16a3249de56e7802dbf

```
struct SplayTree {
  struct Node {
    int ch[2] = {0, 0}, p = 0;
    long long self = 0, path = 0;        // Path aggregates
    long long sub = 0, vir = 0;          // Subtree aggregates
    bool flip = 0;                       // Lazy tags
  };
  vector<Node> T;

  SplayTree(int n) : T(n + 1) {}

  void push(int x) {
    if (!x || !T[x].flip) return;
    int l = T[x].ch[0], r = T[x].ch[1];

    T[l].flip ^= 1, T[r].flip ^= 1;
    swap(T[x].ch[0], T[x].ch[1]);
    T[x].flip = 0;
  }

  void pull(int x) {
    int l = T[x].ch[0], r = T[x].ch[1]; push(l); push(r);

    T[x].path = T[l].path + T[x].self + T[r].path;
    T[x].sub = T[x].vir + T[l].sub + T[r].sub + T[x].self;
  }

  void set(int x, int d, int y) {
    T[x].ch[d] = y; T[y].p = x; pull(x);
  }

  void splay(int x) {
    auto dir = [&](int x) {
      int p = T[x].p; if (!p) return -1;
      return T[p].ch[0] == x ? 0 : T[p].ch[1] == x ? 1 : -1;
    };
    auto rotate = [&](int x) {
      int y = T[x].p, z = T[y].p, dx = dir(x), dy = dir(y);
      set(y, dx, T[x].ch[!dx]);
      set(x, !dx, y);
      if (~dy) set(z, dy, x);
      T[x].p = z;
    };
    for (push(x); ~dir(x); ) {
      int y = T[x].p, z = T[y].p;
      push(z); push(y); push(x);
      int dx = dir(x), dy = dir(y);
      if (~dy) rotate(dx != dy ? x : y);
      rotate(x);
    }
  }
};

struct LinkCut : SplayTree {
  LinkCut(int n) : SplayTree(n) {}

  int access(int x) {
    int u = x, v = 0;
    for (; u; v = u, u = T[u].p) {
      splay(u);
      int& ov = T[u].ch[1];
      T[u].vir += T[ov].sub;
      T[u].vir -= T[v].sub;
      ov = v; pull(u);
    }
    return splay(x), v;
  }

  void reroot(int x) {
    access(x); T[x].flip ^= 1; push(x);
  }

  // Podczep u jako dziecko v
  void Link(int u, int v) {
    reroot(u); access(v);
    T[v].vir += T[u].sub;
    T[u].p = v; pull(v);
  }

  // Usuń krawędz (u, v)
  void Cut(int u, int v) {
    reroot(u); access(v);
    T[v].ch[0] = T[u].p = 0; pull(v);
  }

  // Rooted tree LCA. Returns 0 if u and v arent connected.
  int LCA(int u, int v) {
    if (u == v) return u;
    access(u); int ret = access(v);
    return T[u].p ? ret : 0;
  }

  // Query subtree of u where v is outside the subtree.
  long long Subtree(int u, int v) {
    reroot(v); access(u); return T[u].vir + T[u].self;
  }

  // Query path [u..v]
  long long Path(int u, int v) {
    reroot(u); access(v); return T[v].path;
  }

  // Update vertex u with value v
  void Update(int u, long long v) {
    access(u); T[u].self = v; pull(u);
  }
};
```

10. Treap Adam

5eb594d6af64d9ef8446db75b51d7500

```
struct treap // treap z zadania odwracane sortowanie
{
        struct node
        {
                node* l;
                node* r;
                int val;
                int min;
                int rev;
                int size;
                int rank;
                node (int val = 0) : val(val), min(val), rev(false), l(NULL), r(NULL),
                ↪  size(1), rank(rand()) {}
        };

        node pool[1000005];
        int head = 0;

        node* mynew (int val){
                pool[head] = node(val);
                return pool + head ++;
        }

        node* root = NULL;

        int size(node* u) { return u == NULL ? 0 : u->size; }
        int min(node* u) { return u == NULL ? 1e9 : u->min; }

        void pull(node* u){
                u->size = 1 + size(u->l) + size(u->r);
                u->min = std::min(u->val, std::min(min(u->l), min(u->r)));
        }
        void push(node* u){
                if (u->rev){
                        swap(u->l, u->r);
                        if (u->l != NULL) u->l->rev ^= 1;
                        if (u->r != NULL) u->r->rev ^= 1;
                        u->rev = false;
                }
        }

        pair <node*, node*> split(node* u, int k){
                if (u == NULL) return {NULL, NULL};
                push(u);
                if (k <= size(u->l)){
                        auto p = split(u->l, k);
                        u->l = p.s;
                        pull(u);
                        return {p.f, u};
                } else {
                        auto p = split(u->r, k - size(u->l) - 1);
                        u->r = p.f;
                        pull(u);
                        return {u, p.s};
                }
        }

        node* merge(node* a, node* b){
                if (a == NULL) return b;
                if (b == NULL) return a;
                push(a); push(b);
                if (a->rank > b->rank){
                        a->r = merge(a->r, b);
                        pull(a);
                        return a;
                } else {
                        b->l = merge(a, b->l);
                        pull(b);
                        return b;
                }
        }

        void push_back(int val){
                root = merge(root, mynew(val));
        }

        void pop_front(){
                root = split(root, 1).s;
        }

        int find_min(){
                int res = 0;
                node* u = root;
                push(u);
                while (u->val != u->min){
                        push(u);
                        if (min(u->l) < min(u->r))
                                u = u->l;
                        else {
                                res += size(u->l) + 1;
                                u = u->r;
                        }
                }
                push(u);
                res += size(u->l);
                return res;
        }

        void reverse_pref(int k){
                auto p = split(root, k);
                p.f->rev ^= 1;
                root = merge(p.f, p.s);
        }
};
```

11. Wavelet

b1df6bfeb814126836a6397c6686a611

```
//array values can be negative too, use appropriate minimum and maximum value
struct wavelet_tree {
  int lo, hi;
  wavelet_tree *l, *r;
  int *b, *c, bsz, csz; // c holds the prefix sum of elements

  wavelet_tree() {
    lo = 1;
    hi = 0;
    bsz = 0;
    csz = 0, l = NULL;
    r = NULL;
  }

  void init(int *from, int *to, int x, int y) {
    lo = x, hi = y;
    if(from >= to) return;
    int mid = (lo + hi) >> 1;
    auto f = [mid](int x) {
      return x <= mid;
    };
    b = (int*)malloc((to - from + 2) * sizeof(int));
    bsz = 0;
    b[bsz++] = 0;
    c = (int*)malloc((to - from + 2) * sizeof(int));
    csz = 0;
    c[csz++] = 0;
    for(auto it = from; it != to; it++) {
      b[bsz] = (b[bsz - 1] + f(*it));
      c[csz] = (c[csz - 1] + (*it));
      bsz++;
      csz++;
    }
    if(hi == lo) return;
    auto pivot = stable_partition(from, to, f);
    l = new wavelet_tree();
    l->init(from, pivot, lo, mid);
    r = new wavelet_tree();
    r->init(pivot, to, mid + 1, hi);
  }
  //kth smallest element in [l, r]
  //for array [1,2,1,3,5] 2nd smallest is 1 and 3rd smallest is 2
  int kth(int l, int r, int k) {
    if(l > r) return 0;
    if(lo == hi) return lo;
    int inLeft = b[r] - b[l - 1], lb = b[l - 1], rb = b[r];
    if(k <= inLeft) return this->l->kth(lb + 1, rb, k);
    return this->r->kth(l - lb, r - rb, k - inLeft);
  }
  //count of numbers in [l, r] Less than or equal to k
  int LTE(int l, int r, int k) {
    if(l > r || k < lo) return 0;
    if(hi <= k) return r - l + 1;
    int lb = b[l - 1], rb = b[r];
    return this->l->LTE(lb + 1, rb, k) + this->r->LTE(l - lb, r - rb, k);
  }
  //count of numbers in [l, r] equal to k
  int count(int l, int r, int k) {
    if(l > r || k < lo || k > hi) return 0;
    if(lo == hi) return r - l + 1;
    int lb = b[l - 1], rb = b[r];
    int mid = (lo + hi) >> 1;
    if(k <= mid) return this->l->count(lb + 1, rb, k);
    return this->r->count(l - lb, r - rb, k);
  }
  //sum of numbers in [l ,r] less than or equal to k
  int sum(int l, int r, int k) {
    if(l > r or k < lo) return 0;
    if(hi <= k) return c[r] - c[l - 1];
    int lb = b[l - 1], rb = b[r];
    return this->l->sum(lb + 1, rb, k) + this->r->sum(l - lb, r - rb, k);
  }
  ~wavelet_tree() {
    delete l;
    delete r;
  }
};
```

12. Subset Sum DP

```
    9852cf0f880d1f38b9a53e7e6bcf0328
```

```
vector <int> solve(int W, vector <int> coins)
{
    int n = coins.size();
    if (n == 0) {
        return 0;
    }

    vector <int> dp[2];
    for (int t = 0; t < 2; ++t) {
        dp[t].resize(W + W + 1);
    }

    int all_sum = 0;
    for (auto v: coins) {
        all_sum += v;
    }

    int goal = all_sum / 2;
    int start_idx = 0;
    int cur_sum = 0;

    while (cur_sum + coins[start_idx] <= goal) {
        cur_sum += coins[start_idx++];
    }

    start_idx--;
    for (int i = 0; i < 2; ++i) {
        for (int j = 0; j <= W + W; ++j) {
            dp[i][j] = -1;
        }
    }

    auto update = [&](const int id) {
        for (int i = W + W; i >= 0; --i) {
            for (int j = max(0, dp[id ^ 1][i]); j < dp[id][i]; ++j) {
                if (i - coins[j] >= 0) {
                    dp[id][i - coins[j]] = max(dp[id][i - coins[j]], j);
                }
            }
        }
    };

    dp[start_idx & 1][cur_sum + W - goal] = start_idx + 1;
    update(start_idx & 1);

    for (int i = start_idx + 1; i < n; ++i) {
        const int id = i & 1;
        dp[id].assign(W + W + 1, -1);

        for (int j = 0; j + coins[i] <= W + W; ++j) {
            dp[id][j] = max(dp[id][j], dp[id ^ 1][j]);
            dp[id][j + coins[i]] = max(dp[id][j + coins[i]], dp[id ^ 1][j]);
        }

        update(id);
    }

    const int id = (n - 1) & 1;
    return dp[id];
}
```

13. Otoczka wypukła

47743f9c1902e90fbb2b5810c8e8ec05

```
#define f first
#define s second
#define x first
#define y second
#define all(x) x.begin(),x.end()
#define ss(x) ((int)((x).size()))
#define rep(i,a,b) for(int i=(a);i<=(b);i++)
template <class p, class q> pair<p,q> operator-(pair<p,q> a, pair<p,q> b) { return
↪  mp(a.f - b.f, a.s - b.s); }
template <class p, class q> pair<p,q> operator+(pair<p,q> a, pair<p,q> b) { return
↪  mp(a.f + b.f, a.s + b.s); }
template <class p, class q> void umin(p &a, const q &b) { if (a > b) a = b; }
template <class p, class q> void umax(p &a, const q &b) { if (a < b) a = b; }
using ll = long long;
using cll = const ll;
using point = pair <ll,ll>;
using cpoint = const point;

ll IS(cpoint& a, cpoint& b) { return a.x * b.x + a.y * b.y; }
ll IW(cpoint& a, cpoint& b) { return a.x * b.y - a.y * b.x; }
ll IW(cpoint& a, cpoint& b, cpoint& c) { return IW(a - c, b - c); }
point rot90(point a) { return point(-a.y, a.x); }

vector <point> convex_hull(vector <point> points, bool strict){
        sort(all(points));
        vector <point> hull;
        rep(phase, 0, 1){
                int start = ss(hull);
                for (cpoint& p : points){
                        while (hull.size() >= start + 2){
                                cll iw = IW(p, hull.back(), hull[ss(hull) - 2]);
                                if (iw < 0 || iw == 0 && strict == false) break;
                                hull.pop_back();
                        }
                        hull.pb(p);
                }
                hull.pop_back();
                reverse(all(points));
        }
        if (ss(hull) == 2 && hull[0] == hull[1])
                hull.pop_back();
        return hull;
}

struct cht
{
        vector <point> hull;
        vector <point> vecs;
        vector <int> ids;

        void insert(point p, int id) // utrzymuje dolną otoczkę; p.x powinny rosnąć
        {
                while (!vecs.empty() && IS(vecs.back(), p - hull.back()) <= 0)
                {
                        hull.pop_back();
                        vecs.pop_back();
                        ids.pop_back();
                }
                if (!hull.empty())
                        vecs.pb(rot90(p - hull.back()));
                hull.pb(p);
                ids.pb(id);
        }

        pair <ll,int> query(ll x) // zwraca minimum w punkcie x / minimalny iloczyn
        ↪  skalarny + do kogo należy
        {
                point p(x, 1);
                auto it = lower_bound(all(vecs), p, [] (point a, point b) {
                        return IW(a, b) > 0;
                });
                int i = it - vecs.begin();
                return { IS(p, hull[i]), ids[i] };
        }
};
```

14. Punkty i proste

cd161c446c970b07550e805e8b129c1a

```
typedef long double T;
int sgn(T x) {
    const T eps = 1e-9;
    if(abs(x) <= eps) return 0;
    return x > 0 ? 1 : -1;
}
struct Vec {
    T x, y;
};
Vec operator+(const Vec &a, const Vec &b) {
    return { a.x + b.x, a.y + b.y };
}
Vec operator-(const Vec &a, const Vec &b) {
    return { a.x - b.x, a.y - b.y };
}
Vec operator*(const Vec &a, const T &b) {
    return  { a.x * b, a.y * b };
}
Vec operator/(const Vec &a, const T &b) {
    return  { a.x / b, a.y / b };
}
T dot(const Vec &a, const Vec &b) {
    return a.x * b.x + a.y * b.y;
}
T prod(const Vec &a, const Vec &b) {
    return a.x * b.y - a.y * b.x;
}
T len(const Vec &v) {
    return sqrt(dot(v, v));
}
T len2(const Vec &v) {
    return dot(v, v);
}
T alpha(const Vec &v) {
    return atan2(v.y, v.x);
}
struct ParamLine {
    Vec p, v;
    ParamLine() = default;
    ParamLine(Vec a, Vec b, bool norm=false) {
        p = a; v = b - a;
        if(norm) v = v / len(v);
    }
};
Vec project(const ParamLine &l, const Vec &u) {
    return l.p + l.v * dot((u - l.p), l.v) / len2(l.v);
}
T dist(const ParamLine &l, const Vec &u) {
    return abs(prod(l.v, u - l.p) / len(l.v));
}
Vec intersection(ParamLine a, ParamLine b) {
        T norm = prod(a.v, b.v);
        T t = prod((b.p - a.p), b.v) / norm;
        return a.p + a.v * t;
}
```

15. Kółka

71ef43e8cdc940998eee383d3e1ed76e

```
struct Circle {
    Vec o;
    T r;
};
pair<Vec, Vec> intersect(const Circle &a, const Circle &b) {
    Vec v = b.o - a.o; T d = len(v);
    T x = (d * d + a.r * a.r - b.r * b.r) / (2 * d);
    T y = sqrt(a.r * a.r - x * x);
    v = v / d; Vec u { -v.y, v.x };
    return { a.o + v * x + u * y, a.o + v * x - u * y };
}
pair<Vec, Vec> intersect(const Circle &c, const ParamLine &l) {
    Vec pr = project(l, c.o);
    T t = sqrt(c.r * c.r - len2(pr - c.o));
    Vec v = l.v / len(l.v);
    return { pr + v * t, pr - v * t };
}
pair<Vec, Vec> tangent(const Circle &c, const Vec &v) {
    Vec d = v - c.o;
    T r = sqrt(len2(d) - c.r * c.r);
    return intersect(c, Circle { v, r } );
}
void tangents(Vec c, double r1, double r2, vector<ParamLine> & ans) {
    double r = r2 - r1;
    double z = c.x * c.x + c.y * c.y;
    double d = z - r * r;
    if(sgn(d) == -1)  return;
    d = sqrt (abs (d));
    Vec v { c.x * d - c.y * r, c.x * r + c.y * d };
    Vec p = Vec { -v.y, v.x } * r1 / z;
    ans.emplace_back(p, p + v);
}
vector<ParamLine> tangents (Circle a, Circle b) {
    vector<ParamLine> ans;
    for (int i=-1; i<=1; i+=2)
        for (int j=-1; j<=1; j+=2)
            tangents (b.o-a.o, a.r*i, b.r*j, ans);
    for (size_t i=0; i<ans.size(); ++i)
        ans[i].p = ans[i].p + a.o;
    return ans;
}
```

16. Przecięcie półpłaszczyzn

b51a3286beabb7a52727669ccd1b534c

```
#define ALL(X) X.begin(), X.end()
#define FORW(I, A, B) for (int(I) = (A); (I) < (B); (I)++)
#define SIZE(X) int(X.size())
#define PB push_back
inline int type(const Vec &p) {
    return sgn(p.y) == 1 or (sgn(p.y) == 0 && sgn(p.x) == 1);
}
bool compare_angle(const Vec &a, const Vec &b) {
    int at = type(a), bt = type(b);
    if (at != bt)
        return at < bt;
    int p = sgn(prod(a, b));
    if(p) return p > 0;
    return sgn(len2(a) - len2(b)) == -1;
}
bool operator<(const ParamLine &a, const ParamLine &l) {
    if (sgn(a.v.x - l.v.x) or sgn(a.v.y - l.v.y))
        return compare_angle(
            a.v, l.v);
    return sgn(prod(a.v, l.p - a.p)) < 0;
}
int sgn_dist(const ParamLine &l, const Vec &u) {
    return sgn(prod(l.v, u - l.p));
}
vector<Vec> halfcoat(vector<ParamLine> h) {
    const static int MAXN = 40020;
    static const T Z = 1e6;
    static Vec p[MAXN];
    static Vec box[4] = {
        { Z, -Z },
        { Z, Z },
        { -Z, Z },
        { -Z, -Z }};
    FORW(i, 0, 4)
    h.PB(ParamLine(box[i], box[(i + 1) % 4], true));
    int n = SIZE(h), z = 0;
    sort(ALL(h));
    FORW(i, 0, n)
    if (i == 0 or (sgn(h[i].v.x - h[i - 1].v.x) or sgn(h[i].v.y - h[i - 1].v.y)))
        h[z++] = h[i];
    n = z;
    int m = 0, del = 0;
    FORW(i, 1, n)
    {
        while (m > del and sgn_dist(h[i], p[m - 1]) <= 0)
            --m;
        while (m > del and sgn_dist(h[i], p[del]) <= 0)
            ++del;
        if (del == m and sgn(prod(h[m].v, h[i].v)) <= 0)
            return {};
        Vec q = intersection(h[i], h[m]);
        if (sgn_dist(h[del], q) >= 0)
                p[m++] = q, h[m] = h[i];
    }
    rotate(p, p + del, p + m);
    rotate(h.begin(), h.begin() + del, h.end());
    m -= del;
    if (m == 0)
        return {};
    Vec q = intersection(h[0], h[m]);
    p[m++] = q;
    return vector<Vec>(p, p + m);
}
```

17. Trapezoidacja

56abea681ae28d4d928da0a8b0bc23cb

```
bool operator<(const Vec &a, const Vec &b) {
    return make_pair(a.x, a.y) < make_pair(b.x, b.y);
}
struct Trapezoid {
    Vec bl, br, tl, tr;
};
T get_y(const ParamLine &l, T x) {
    return l.p.y + l.v.y * (x - l.p.x) / l.v.x;
}
struct CmpY {
    T *x;
    T y(const ParamLine &l) const {
        return get_y(l, *x);
    }
    bool operator()(const ParamLine &a, const ParamLine &b) const {
        return y(a) < y(b);
    }
};
vector<Trapezoid> trapezoidation(const vector<Vec> &poly) {
    const T leps = 1e-9;
    int n = poly.size();
    auto previ = [&](int k) { return (k + n - 1) % n; };
    auto nexti = [&](int k) { return (k + 1) % n; };
    vector<int> pts(n);
    for(int i = 0; i < n; i++) pts[i] = i;
    sort(pts.begin(), pts.end(), [&](int a, int b) {
        return poly[a] < poly[b];
    });
    vector<bool> added(n);
    vector<Trapezoid> ans;
    T x; CmpY compare { &x };
    multimap<ParamLine, bool, CmpY> st(compare);
    for(int i: pts) {
        if(added[i]) continue;
        added[i] = true;
        x = poly[i].x;
        int i1 = i, i2 = i;
        int pi = previ(i1), ni = nexti(i2);
        while(sgn(poly[i1].x - poly[pi].x) == 0) {
            added[pi] = true;
            i1 = pi; pi = previ(pi);
        }
        while(sgn(poly[i2].x - poly[ni].x) == 0) {
            added[ni] = true;
            i2 = ni; ni = nexti(ni);
        }
        if(!added[pi] && !added[ni]) {
            if((i1 == i2 && prod(poly[pi] - poly[i], poly[ni] - poly[i]) < 0)
             || sgn(poly[i1].y - poly[i2].y) > 0) {
                swap(pi, ni); swap(i1, i2);
            }
            auto it1 = st.insert({ ParamLine(poly[i1], poly[pi]), true });
            auto it2 = st.insert({ ParamLine(poly[i2], poly[ni]), false });
            if(it1 != st.begin() && prev(it1)->second == true) {
                swap(it1->second, it2->second);
                auto &a = prev(it1)->first, &b = next(it2)->first;
                Vec na { x, compare.y(a) }, nb { x, compare.y(b) };
                ans.push_back({ a.p, na, b.p, nb });
                a.p = na; b.p = nb;
            }
        } else if(added[pi] && added[ni]) {
            if((i1 == i2 && prod(poly[pi] - poly[i], poly[ni] - poly[i]) > 0)
             || sgn(poly[i1].y - poly[i2].y) > 0) {
                swap(pi, ni); swap(i1, i2);
            }
            auto it1 = st.lower_bound(ParamLine(poly[pi], poly[i1] - Vec { 0, leps }));
            auto it2 = prev(st.upper_bound(ParamLine(poly[ni], poly[i2] + Vec { 0, leps
            ↪  })));
            if(it1->second == true) {
                ans.push_back({ it1->first.p, poly[i1], it2->first.p, poly[i2] });
            } else {
                auto &a = prev(it1)->first, &b = next(it2)->first;
                Vec na { x, compare.y(a) }, nb { x, compare.y(b) };
                ans.push_back({ a.p, na, it1->first.p, poly[i1] });
                ans.push_back({ it2->first.p, poly[i2], b.p, nb });
                a.p = na; b.p = nb;
            }
            st.erase(it1); st.erase(it2);
        } else {
            if(!added[pi]) {
                swap(pi, ni); swap(i1, i2);
            }
            auto it = st.lower_bound(ParamLine(poly[pi], poly[i1] - Vec { 0, leps }));
            assert(it != st.end());
            if(it->second) {
                auto &a = next(it)->first;
                Vec na { x, compare.y(a) };
                ans.push_back({ it->first.p, poly[i1], a.p, na });
                a.p = na;
            } else {
                auto &a = prev(it)->first;
                Vec na { x, compare.y(a) };
                ans.push_back({ a.p, na, it->first.p, poly[i1] });
                a.p = na;
            }
            it->first.p = poly[i2];
            it->first.v = poly[ni] - poly[i2];
        }
    }
    ans.erase(remove_if(ans.begin(), ans.end(), [](Trapezoid t) {
        return sgn(t.bl.x - t.br.x) == 0;
    }), ans.end());
    return ans;
}
```

18. Voronoi

b7616e903d792310c7c1605db6d8f1f5

```
#define Oi(e) ((e)->oi)
#define Dt(e) ((e)->dt)
#define On(e) ((e)->on)
#define Op(e) ((e)->op)
#define Dn(e) ((e)->dn)
#define Dp(e) ((e)->dp)
#define Other(e, p) ((e)->oi == p ? (e)->dt : (e)->oi)
#define Next(e, p) ((e)->oi == p ? (e)->on : (e)->dn)
#define Prev(e, p) ((e)->oi == p ? (e)->op : (e)->dp)
#define V(p1, p2, u, v) (u = p2->x - p1->x, v = p2->y - p1->y)
#define C2(u1, v1, u2, v2) (u1 * v2 - v1 * u2)
#define C3(p1, p2, p3) ((p2->x - p1->x) * (p3->y - p1->y) - (p2->y - p1->y) * (p3->x -
↪  p1->x))
#define Dot(u1, v1, u2, v2) (u1 * u2 + v1 * v2)
#define dis(a,b) (sqrt( (a->x - b->x) * (a->x - b->x) + (a->y - b->y) * (a->y - b->y) ))
const int maxn = 110024;
const int aix = 4;
const double eps = 1e-7;
int n, M, k;
struct gEdge {
        int u, v; double w;
        bool operator <(const gEdge &e1) const { return w < e1.w - eps; }
} E[aix * maxn], MST[maxn];
struct point {
        double x, y; int index; edge *in;
        bool operator <(const point &p1) const { return x < p1.x - eps || (abs(x - p1.x)
        ↪  <= eps && y < p1.y - eps); }
};
```

```
struct edge { point *oi, *dt; edge *on, *op, *dn, *dp; };
point p[maxn], *Q[maxn];
edge mem[aix * maxn], *elist[aix * maxn];
int nfree;
void Alloc_memory() { nfree = aix * n; edge *e = mem; for (int i = 0; i < nfree; i++)
↪  elist[i] = e++; }
void Splice(edge *a, edge *b, point *v) {
        edge *next;
        if (Oi(a) == v) next = On(a), On(a) = b; else next = Dn(a), Dn(a) = b;
        if (Oi(next) == v) Op(next) = b; else Dp(next) = b;
        if (Oi(b) == v) On(b) = next, Op(b) = a; else Dn(b) = next, Dp(b) = a;
}
edge *Make_edge(point *u, point *v) {
        edge *e = elist[--nfree];
        e->on = e->op = e->dn = e->dp = e; e->oi = u; e->dt = v;
        if (!u->in) u->in = e;
        if (!v->in) v->in = e;
        return e;
}
edge *Join(edge *a, point *u, edge *b, point *v, int side) {
        edge *e = Make_edge(u, v);
        if (side == 1) {
                if (Oi(a) == u) Splice(Op(a), e, u);
                else Splice(Dp(a), e, u);
                Splice(b, e, v);
        } else {
                Splice(a, e, u);
                if (Oi(b) == v) Splice(Op(b), e, v);
                else Splice(Dp(b), e, v);
        } return e;
}
void Remove(edge *e) {
        point *u = Oi(e), *v = Dt(e);
        if (u->in == e) u->in = e->on;
        if (v->in == e) v->in = e->dn;
        if (Oi(e->on) == u) e->on->op = e->op; else e->on->dp = e->op;
        if (Oi(e->op) == u) e->op->on = e->on; else e->op->dn = e->on;
        if (Oi(e->dn) == v) e->dn->op = e->dp; else e->dn->dp = e->dp;
        if (Oi(e->dp) == v) e->dp->on = e->dn; else e->dp->dn = e->dn;
        elist[nfree++] = e;
}
void Low_tangent(edge *e_l, point *o_l, edge *e_r, point *o_r, edge **l_low, point **OL,
↪  edge **r_low, point **OR) {
        for (point *d_l = Other(e_l, o_l), *d_r = Other(e_r, o_r); ; )
                if (C3(o_l, o_r, d_l) < -eps)      e_l = Prev(e_l, d_l), o_l = d_l, d_l
                ↪  = Other(e_l, o_l);
                else if (C3(o_l, o_r, d_r) < -eps) e_r = Next(e_r, d_r), o_r = d_r, d_r
                ↪  = Other(e_r, o_r);
                else break;
        *OL = o_l, *OR = o_r; *l_low = e_l, *r_low = e_r;
}
```

```
void Merge(edge *lr, point *s, edge *rl, point *u, edge **tangent) {
        double l1, l2, l3, l4, r1, r2, r3, r4, cot_L, cot_R, u1, v1, u2, v2, n1, cot_n,
        ↪  P1, cot_P;
        point *O, *D, *OR, *OL; edge *B, *L, *R;
        Low_tangent(lr, s, rl, u, &L, &OL, &R, &OR);
        for (*tangent = B = Join(L, OL, R, OR, 0), O = OL, D = OR; ; ) {
                edge *El = Next(B, O), *Er = Prev(B, D), *next, *prev;
                point *l = Other(El, O), *r = Other(Er, D);
                V(l, O, l1, l2); V(l, D, l3, l4); V(r, O, r1, r2); V(r, D, r3, r4);
                double cl = C2(l1, l2, l3, l4), cr = C2(r1, r2, r3, r4);
                bool BL = cl > eps, BR = cr > eps;
                if (!BL && !BR) break;
                if (BL) {
                        double dl = Dot(l1, l2, l3, l4);
                        for (cot_L = dl / cl; ; Remove(El), El = next, cot_L = cot_n) {
                                next = Next(El, O); V(Other(next, O), O, u1, v1);
                                ↪  V(Other(next, O), D, u2, v2);
                                n1 = C2(u1, v1, u2, v2); if (!(n1 > eps)) break;
                                cot_n = Dot(u1, v1, u2, v2) / n1;
                                if (cot_n > cot_L) break;
                        }
                } if (BR) {
                        double dr = Dot(r1, r2, r3, r4);
                        for (cot_R = dr / cr; ; Remove(Er), Er = prev, cot_R = cot_P) {
                                prev = Prev(Er, D); V(Other(prev, D), O, u1, v1);
                                ↪  V(Other(prev, D), D, u2, v2);
                                P1 = C2(u1, v1, u2, v2); if (!(P1 > eps)) break;
                                cot_P = Dot(u1, v1, u2, v2) / P1;
                                if (cot_P > cot_R) break;
                        }
                } l = Other(El, O); r = Other(Er, D);
                if (!BL || (BL && BR && cot_R < cot_L)) B = Join(B, O, Er, r, 0), D = r;
                else B = Join(El, l, B, D, 0), O = l;
        }
}
void Divide(int s, int t, edge **L, edge **R) {
        edge *a, *b, *c, *ll, *lr, *rl, *rr, *tangent;
        int n = t - s + 1;
        if (n == 2) *L = *R = Make_edge(Q[s], Q[t]);
        else if (n == 3) {
                a = Make_edge(Q[s], Q[s + 1]), b = Make_edge(Q[s + 1], Q[t]);
                Splice(a, b, Q[s + 1]);
                double v = C3(Q[s], Q[s + 1], Q[t]);
                if (v > eps)       c = Join(a, Q[s], b, Q[t], 0), *L = a, *R = b;
                else if (v < -eps) c = Join(a, Q[s], b, Q[t], 1), *L = c, *R = c;
                else *L = a, *R = b;
        } else if (n > 3) {
                int split = (s + t) / 2;
                Divide(s, split, &ll, &lr); Divide(split + 1, t, &rl, &rr);
                Merge(lr, Q[split], rl, Q[split + 1], &tangent);
                if (Oi(tangent) == Q[s]) ll = tangent;
                if (Dt(tangent) == Q[t]) rr = tangent;
                *L = ll; *R = rr;
        }
}
void Make_Graph() {
        edge *start, *e; point *u, *v;
        for (int i = 0; i < n; i++) {
                start = e = (u = &p[i])->in;
                do{ v = Other(e, u);
                        if (u < v) E[M++].u = (u - p, v - p, dis(u, v)); // M < aix *
                        ↪  maxn
                } while ((e = Next(e, u)) != start);
        }
}
int b[maxn];
int Find(int x) { while (x != b[x]) { b[x] = b[b[x]]; x = b[x]; } return x; }
void Kruskal() {
        memset(b, 0, sizeof(b)); sort(E, E + M);
        for (int i = 0; i < n; i++) b[i] = i;
        for (int i = 0, kk = 0; i < M && kk < n - 1; i++) {
                int m1 = Find(E[i].u), m2 = Find(E[i].v);
                if (m1 != m2) b[m1] = m2, MST[kk++] = E[i];
        }
}
void solve() {
        scanf("%d", &n);
        for (int i = 0; i < n; i++) scanf("%lf%lf", &p[i].x, &p[i].y), p[i].index = i,
        ↪  p[i].in = NULL;
        Alloc_memory(); sort(p, p + n);
        for (int i = 0; i < n; i++) Q[i] = p + i;
        edge *L, *R; Divide(0, n - 1, &L, &R);
        M = 0; Make_Graph(); Kruskal();
}
```

19. Dinic

ee279b38c7ec82ffad2853c69d09fc8a

```
#define sz(x) (int)(x).size()
// indeksowany od 1
// uzupełniamy [n, s, t] i dodajemy krawedzie
typedef int T;
const int N = 2005;
const T INF = 1e9;
struct edge {
    int a, b;
    T cap, flow;
};
int n, s, t, d[N], ptr[N], q[N];
vector < edge > e;
vector < int > g[N];
void clear() {
        e.clear();
        for(int i = 1; i <= n; ++i)
                g[i].clear();
}
/* Edges can be added only using add_edge to use this function! */
void clear_flow() {
        for(int i = 0; i < (int)e.size(); i += 2)
                e[i].flow = 0, e[i + 1].flow = e[i + 1].cap;
}
int add_edge(int a, int b, T cap) {
    edge e1 = { a, b, cap, 0 };
    edge e2 = { b, a, cap, cap };
    g[a].push_back(sz(e));
    e.push_back(e1);
    g[b].push_back(sz(e));
    e.push_back(e2);
    return sz(e) - 2;
}
bool bfs() {
    int qh = 0, qt = 0;
    q[qt++] = s;
    memset(d + 1, -1, n * sizeof d[0]);
    d[s] = 0;
    while(qh < qt && d[t] == -1) {
        int v = q[qh++];
        for(int i = 0; i < sz(g[v]); ++i) {
            int id = g[v][i], to = e[id].b;
            if(d[to] == -1 && e[id].flow < e[id].cap) {
                q[qt++] = to;
                d[to] = d[v] + 1;
            }
        }
    }
    return d[t] != -1;
}
T dfs(int v, T flow) {
    if(flow <= 0)  return 0;
    if(v == t)  return flow;
    T res = 0;
    for(; ptr[v]<sz(g[v]); ++ptr[v]) {
        int id = g[v][ptr[v]], to = e[id].b;
        if(d[to] != d[v] + 1) continue;
        T pushed = dfs(to, min(flow, e[id].cap - e[id].flow));
        e[id].flow += pushed;
        e[id^1].flow -= pushed;
        res += pushed;
        flow -= pushed;
        if(flow == 0) break;
    }
    return res;
}
T dinic(int _n, int _s, int _t) {
    n = _n; s = _s; t = _t;
    T flow = 0;
    for (;;) {
        if(!bfs())  break;
        memset(ptr, 0, (n + 1) * sizeof ptr[0]);
        flow += dfs(s, INF);
    }
    return flow;
}
```

20. Gomory-Hu Tree

5516aac48c56368bc1a189c391d931e2

```
struct edge {
        int u, v;
        long long w;
};
int n;
vector <int> p, w, c;
vector <edge> tree;
void dfs(int u) {
        c[u] = 1;
        for(const int &id: Dinic::g[u]) {
                int v = Dinic::e[id].b;
                if(!c[v] and Dinic::e[id].flow < Dinic::e[id].cap)
                        dfs(v);
        }
}
/* Clears and runs */
vector <edge> run(int _n, const vector <edge> &ed) {
        n = _n;
        tree.clear();
        p.resize(n + 1), w.resize(n + 1), c.resize(n + 1);
        for(const auto &e: ed) {
                Dinic::add_edge(e.u, e.v, e.w);
                Dinic::add_edge(e.v, e.u, e.w);
        }
        p[1] = 0, fill(p.begin() + 2, p.end(), 1);
        for(int i = 2; i <= n; ++i) {
                w[i] = Dinic::dinic(n, i, p[i]);
                fill(c.begin(), c.end(), 0);
                dfs(i);
                for(int j = i + 1; j <= n; ++j)
                        if(c[j] && p[j] == p[i])
                                p[j] = i;
                if(p[p[i]] && c[p[p[i]]]) {
                        int pi = p[i];
                        swap(w[i], w[pi]);
                        p[i] = p[pi];
                        p[pi] = i;
                }
                Dinic::clear_flow();
        }
        tree.clear();
        for(int i = 1; i <= n; ++i) {
                if(p[i])
                        tree.push_back(edge{i, p[i], w[i]});
        }
        return tree;
}
```

21. Min-cost max-flow

9f227bfb2cfcd5fb30c148c86ab2dd1b

```
typedef int flow_t;
typedef int cost_t;
struct edge {
    int u, v;
    flow_t flow, capa;
    cost_t cost;
};
const int N = 10000;
const cost_t cinf = 1e9;
const flow_t finf = 1e9;
vector<int> g[N];
cost_t d[N];
cost_t p[N];
int pre[N];
vector<edge> e;
int n;
inline bool remin(cost_t &a, cost_t b) {
    return a > b ? a = b, true : false;
}
void init(int _n) { //wierzcholki numerowane od 0 do n - 1
    n = _n;
    for(int i = 0; i < n; i++)
        g[i].clear();
    e.clear();
}
void add_edge(int u, int v, flow_t capa, cost_t cost) {
    g[u].push_back(e.size());
    e.push_back({ u, v, 0, capa, cost });
    g[v].push_back(e.size());
    e.push_back({ v, u, 0, 0, -cost });
}
pair<flow_t, cost_t> flow(int s, int t) {
    fill(p, p + n, 0);
    bool improved = true;
    while(improved) {
        improved = false;
        for(auto &ed: e)
            if(ed.flow < ed.capa && remin(p[ed.v], p[ed.u] + ed.cost))
                improved = true;
    }
    flow_t fans = 0;
    cost_t cans = 0;
    while(true) {
        fill(d, d + n, cinf);
        priority_queue<pair<cost_t, int>> q;
        d[s] = 0; q.push({ 0, s });
        while(!q.empty()) {
            auto u = q.top().second, c = -q.top().first;
            q.pop();
            if(c != d[u]) continue;
            for(int ed: g[u]) {
                if(e[ed].flow == e[ed].capa) continue;
                auto v = e[ed].v;
                if(remin(d[v], c + p[u] - p[v] + e[ed].cost)) {
                    pre[v] = ed;
                    q.push({ -d[v], v });
                }
            }
        }
        if(d[t] == cinf) break;
        vector<int> path;
        int v = t, ed = pre[v];
        flow_t flow = finf;
        while(v != s) {
            path.push_back(ed);
            flow = min(flow, e[ed].capa - e[ed].flow);
            v = e[ed].u;
            ed = pre[v];
        }
        for(auto ed: path) {
            e[ed].flow += flow;
            e[ed^1].flow -= flow;
        }
        fans += flow;
        cans += flow * (d[t] + p[t] - p[s]);
        for(int i = 0; i < n; i++)
            p[i] += d[i];
    }
    return { fans, cans };
}
```

22. Hungarian

bf99fa2858cd62d5c725aa617899112d

```
typedef int T;
const int N = 507;
const T INF = 1e9 + 7;
int n, max_match;
T cost[N][N], lx[N], ly[N], slack[N], slackx[N];
int xy[N], yx[N], prev[N];
bool S[N], U[N];
void update_labels() {
    T delta = INF;
    FOR(y, n) if(!U[y]) delta = min(delta, slack[y]);
    FOR(x, n) if(S[x]) lx[x] -= delta;
    FOR(y, n) if(U[y]) ly[y] += delta;
    FOR(y, n) if(!U[y]) slack[y] -= delta;
}
void add_to_tree(int x, int f) {
    S[x] = true; prev[x] = f;
    FOR(y, n) if(lx[x] + ly[y] - cost[x][y] < slack[y]) {
        slack[y] = lx[x] + ly[y] - cost[x][y];
        slackx[y] = x;
    }
}
void augment() {
    if (max_match == n) return;
    int root, q[N], wr = 0, rd = 0;
    memset(S, false, sizeof(S));
    memset(U, false, sizeof(U));
    memset(prev, -1, sizeof(prev));
    FOR(x, n) if(xy[x] == -1) {
        q[wr++] = root = x;
        prev[x] = -2;
        S[x] = true;
        break;
    }
    FOR(y, n) {
        slack[y] = lx[root] + ly[y] - cost[root][y];
        slackx[y] = root;
    }
    int x, y;
    while(true) {
        while(rd < wr) {
            x = q[rd++];
            for(y = 0; y < n; y++) {
                if(cost[x][y] == lx[x] + ly[y] && !U[y]) {
                    if (yx[y] == -1) break;
                    U[y] = true;
                    q[wr++] = yx[y];
                    add_to_tree(yx[y], x);
                }
            }
            if(y < n) break;
        }
        if (y < n) break;
        update_labels();
        wr = rd = 0;
        for(y = 0; y < n; y++) {
            if(!U[y] && slack[y] == 0) {
                if(yx[y] == -1) {
                    x = slackx[y];
                    break;
                } else {
                    U[y] = true;
                    if(!S[yx[y]]) {
                        q[wr++] = yx[y];
                        add_to_tree(yx[y], slackx[y]);
                    }
                }
            }
        }
        if(y < n) break;
    }
    if(y < n) {
        max_match++;
        for(int cx = x, cy = y, ty; cx != -2; cx = prev[cx], cy = ty) {
            ty = xy[cx];
            yx[cy] = cx;
            xy[cx] = cy;
        }
        augment();
    }
}
T maxCostMatching()
{
    T res = 0; max_match = 0;
    FOR(i, n) xy[i] = yx[i] = -1, lx[i] = ly[i] = 0;
    FOR(x, n) FOR(y, n) lx[x] = max(lx[x], cost[x][y]);
    augment();
    FOR(x, n) res += cost[x][xy[x]];
    return res;
}
```

23. Dominatory

b0236cf0296506f82cff4f349582e533

```
#define NODES 100007
int parent[NODES+1], ancestor[NODES+1], vertex[NODES+1];
int label[NODES+1], semi[NODES+1];
int dom[NODES+1]; // drzewo dominacji, dom[v] - ojciec v w drzewie, dom[root] = 0
vi succ[NODES+1], pred[NODES+1], bucket[NODES+1];
int czas, N, root;
void czysc (int _N, int _root) {
    N = _N;
    root = _root;
    czas = 0;
    FOR(i,N+1) semi[i] = 0;
    FOR(i,N+1) pred[i].clear();
    FOR(i,N+1) succ[i].clear();
    FOR(i,N+1) bucket[i].clear();
}
void dfs (int v) {
    semi[v] = ++czas;
    label[v] = vertex[czas] = v;
    ancestor[v] = 0;
    for(auto w: succ[v]){
        if (semi[w] == 0) { parent[w] = v; dfs(w); }
        pred[w].push_back(v);
    }
}
void compress (int v) {
    if (ancestor[ancestor[v]] == 0) return;
    compress(ancestor[v]);
    if (semi[label[ancestor[v]]] < semi[label[v]]) label[v] = label[ancestor[v]];
    ancestor[v] = ancestor[ancestor[v]];
}
int eval(int v) {
    if (ancestor[v] == 0) return v;
    compress(v);
    return label[v];
}
void dominacja () {
    dfs(root);
    for(int i = czas; i >= 2; i--) {
        int w = vertex[i];
        for(auto v: pred[w]) {
            int u = eval(v);
            if (semi[u] < semi[w]) semi[w] = semi[u];
        }
        bucket[vertex[semi[w]]].push_back(w);
        ancestor[w] = parent[w];
        for(auto v: bucket[parent[w]]) {
            int u = eval(v);
            dom[v] = (semi[u] < semi[v] ? u : parent[w]);
        }
        bucket[parent[w]].clear();
    }
    for(int i = 2; i <= czas; i++) {
        int w = vertex[i];
        if (dom[w] != vertex[semi[w]]) dom[w] = dom[dom[w]];
    }
    dom[root] = 0;
}
int main() { int n = 10; czysc(n,1); /*dod krawedzie do succ*/ dominacja(); }
```

24. Skierowane MST

ebb02b3ab6583bb66d9998e92f93c675

```
// directed mst z wierzcholka 0 w grafie 0..n-1
// preconditions: 1. nie ma krawedzi wchodzacych do wierzcholka 0
//                2. wszystkie wierzcholki sa osiagalne z wierzcholka 0
// przed uzyciem ustawic n, m, MAXN, MAXM i wypelnic
//edge[0..m-1] (wystarczy pola: u, v, key)
const int MAXN = 100007, MAXM = 100007;
struct edge { // krawedz/element kolejki zlaczalnej
    int u, v; // IN: poczatek i koniec krawedzi
    int key;  // IN: waga krawedzi (zmienia sie!)
    edge *left, *right; // poczatkowo: 0, 0
    int len, add;       // poczatkowo: 1, 0
};
struct node1 { // element zbioru
    node1 *parent; int size, scc;
};
struct node2 { // j.w.
    node2 *parent; // poczatkowa wartosc: this
    int size;      // poczatkowa wartosc: 1
};
// Operacje na zbiorach rozlacznych
template<class T> T *set_find(T *p) { // znajduje reprezentanta
    if (p->parent != p) p->parent = set_find(p->parent);
    return p->parent;
}
template<class T> T *set_union(T *p1, T *p2) { // laczy zbiory
    if (p1->size < p2->size) swap(p1, p2);
    p2->parent = p1;
    p1->size += p2->size;
    return p1;
}
// Operacje na kolejkach zlaczalnych
void tree_push(edge *p) {
    p->key += p->add;
    if (p->left) p->left->add += p->add;
    if (p->right) p->right->add += p->add;
    p->add = 0;
}
edge *tree_union(edge *p1, edge *p2) { // laczy kolejki
    if (!p1) return p2; if (!p2) return p1;
    if (p2->key+p2->add < p1->key+p1->add) swap(p1, p2);
    tree_push(p1);
    p1->right = tree_union(p1->right, p2);
    if (!p1->left || p1->left->len < p1->right->len) swap(p1->left, p1->right);
    p1->len = p1->right ? p1->right->len+1 : 1;
    return p1;
}
edge *tree_extract(edge *p) { // usuwa z kolejki element najmniejszy
    tree_push(p); return tree_union(p->left, p->right);
}
void tree_add(edge *p, int x) { // dodaje x do wszystkich wartosci w kolejce
    if (p) p->add += x;
}
int n, m;          // IN: liczba wierzcholkow, liczba krawedzi
edge edges[MAXM]; // IN: tablica wszystkich krawedzi
node1 scc_set[MAXN];
node2 wcc_set[MAXN];
int upper[2*MAXN], lower[2*MAXN];
edge *adj[2*MAXN];
edge *res[2*MAXN]; // OUT: krawedz do rodzica w drzewie (korzen ma NULL)
int compute_branching() { // zwraca wage drzewa
    FOR(i,n) {
        scc_set[i].parent = scc_set+i;
        scc_set[i].size = 1;
        scc_set[i].scc = i;
        wcc_set[i].parent = wcc_set+i;
        wcc_set[i].size = 1;
        upper[i] = lower[i] = -1;
        adj[i] = res[i] = 0;
    }
    FOR(j,m) {
        edges[j].left = edges[j].right = 0;
        edges[j].len = 1;
        edges[j].add = 0;
        adj[edges[j].v] = tree_union(adj[edges[j].v], edges+j);
    }
    int scc_c=n, value=0;
    FOR(i,n) {
        int c = set_find(scc_set+i)->scc;
        while (adj[c] && !res[c]) {
            edge *e = adj[c];
            adj[c] = tree_extract(adj[c]);
            node1 *s1 = set_find(scc_set+e->v), *s2 = set_find(scc_set+e->u);
            if (s1==s2) continue;
            res[c] = e;
            value += e->key;
            tree_add(adj[c], -e->key);
            node2 *w1 = set_find(wcc_set+e->v), *w2 = set_find(wcc_set+e->u);
            if (w1!=w2) { set_union(w1, w2); continue; }
            upper[c] = scc_c;
            do {
                e = res[s2->scc];
                upper[s2->scc] = scc_c;
                adj[c] = tree_union(adj[c], adj[s2->scc]);
                s1 = set_union(s1, s2);
                s2 = set_find(scc_set+e->u);
            } while (s1!=s2);
            s1->scc = scc_c;
            upper[scc_c] = lower[scc_c] = -1;
            adj[scc_c] = adj[c];
            res[scc_c] = 0;
            c = scc_c++;
        }
    }
    REPD(c,scc_c - 1,n) {
        if (lower[c]==-1)
          for (int i=res[c]->v; i!=c; i=upper[i]) lower[upper[i]] = i;
        res[lower[c]] = res[c];
    }
    return value;
}
```

25. Matching dowolny (random)

97192338fddf7d1f2771187687264d4c

```
const int N = 555;
mt19937 rnd(time(0));

struct Matching {        // [1, n]
        vector<int> e[N];
        int mate[N], vis[N];
        void add(int a, int b) {
                if (a == b) return;
                e[a].push_back(b);
                e[b].push_back(a);
        }
        bool dfs(int a) {
                shuffle(e[a].begin(), e[a].end(), rnd);
                vis[a] = 1;
                for (auto b : e[a]) {
                        int c = mate[b];
                        if (vis[c]) continue;
                        mate[a] = b; mate[b] = a; mate[c] = 0;
                        if (!c || dfs(c)) return 1;
                        mate[a] = 0; mate[b] = c; mate[c] = b;
                }
                return 0;
        }
        vector<pair<int, int>> matching(int n) {
                vector<pair<int, int>> res;
                rep(_, 1, 20) {
                        memset(mate, 0, sizeof mate);
                        rep(i, 1, n) {
                                if (!mate[i]) {
                                        memset(vis, 0, sizeof vis);
                                        dfs(i);
                                }
                        }
                        vector<pair<int, int>> cur;
                        rep(i, 1, n) if (mate[i] > i) cur.push_back({i, mate[i]});
                        if (cur.size() > res.size()) res = cur;
                }
                return res;
        }
};
```

26. Matching dowolny

fa9b662370e56f0b1a48ac4fe9352938

```
#define N 307
int n;                 // IN: liczba wierzcholkow
bool edge[N][N];       // IN: macierz sasiedztwa (mozna zmienic na liste)
int mate[N];           // OUT: wierzcholek skojarzony (-1 oznacza brak)
int label[N], base[N], prev1[N], prev2[N];
bool mark[N];
bool prepare (int v) {
    while(1) {
        mark[v] = !mark[v];
        if (mate[v] == -1) return mark[v];
        v = base[prev2[mate[v]]];
    }
}
int shrink (int v, int b1, int b2, queue<int> &Q) {
    while (mark[v]) {
        prev1[v] = b1; prev2[v] = b2;
        mark[mate[v]] = 1;
        Q.push(mate[v]);
        v = base[prev2[mate[v]]];
    }
    return v;
}
bool make_blos (int i, int j, int bi, int bj, queue<int> &Q) {
    if (label[i]!=1 || i==j) return 0;
    if (prepare(i), prepare(j)) return 1;
    int b = (shrink(i, bi, bj, Q), shrink(j, bj, bi, Q));
    FOR(v,n) if (mark[base[v]]) base[v] = b;
    return 0;
}
void rematch(int i, int j) {
    int nxt = mate[i];
    mate[i] = j;
    if (nxt==-1) return;
    mate[nxt] = -1;
    rematch(prev2[nxt], prev1[nxt]);
    rematch(prev1[nxt], prev2[nxt]);
}
bool augment() {
    queue<int> Q;
    FOR(i,n) {
        label[i] = mate[i]==-1;
        if (mate[i]==-1) Q.push(i);
        mark[i] = 0;
        base[i] = i;
    }
    while (!Q.empty()) {
        int cur = Q.front(); Q.pop();
        FOR(i,n) /*tu zmienic*/ if (edge[cur][i] && i!=mate[cur]) {
            if (!label[i]) {             // (nie wiem co "zmienic", dziala jak jest)
                label[i] = -1;
                label[mate[i]] = 1;
                Q.push(mate[i]);
                prev1[i] = i; prev2[i] = cur;
            } else if (make_blos(base[i], base[cur], i, cur, Q)) {
                rematch(i, cur); rematch(cur, i);
                return 1;
            }
        }
    }
    return 0;
}
int compute_gcm() { // zwraca licznosc maksymalnego skojarzenia
    fill_n(mate, n, -1);
    int res = 0;
    while (augment()) ++res;
    return res;
}
```

27. Matching dowolny ważony (random)

bf3d0e46ae9fb4933a125fa1c8ccdcc9

```
struct RandomizedMatching {
        long long G[N][N], dis[N];
        int match[N];
        int mat[N], stk[N], id[N], vis[N];
        int n, top;
        const long long inf = 1e18;
        RandomizedMatching() {}
        RandomizedMatching(int _n) {
                n = _n; top = 0;
                memset(match, 0, sizeof match);
                for (int i = 1; i <= n + 1; i++) {
                        for (int j = 1; j <= n + 1; j++) {
                                G[i][j] = 0;
                        }
                }
        }
        void add_edge(int u, int v, long long w) {
                G[u][v] = max(G[u][v], w);
                G[v][u] = max(G[v][u], w);
        }
        bool spfa(int u) {
                stk[top ++] = u;
                if (vis[u]) return true;
                vis[u] = true;
                for (int i = 1; i <= n; ++ i) {
                        if (i != u && i != mat[u] && !vis[i]) {
                                int v = mat[i];
                                if (dis[v] < dis[u] + G[u][i] - G[i][v]) {
                                        dis[v] = dis[u] + G[u][i] - G[i][v];
                                        if (spfa(v)) return true;
                                }
                        }
                }
                top --; vis[u] = false;
                return false;
        }
        long long maximum_matching() {
                for (int i = 1; i <= n; ++ i) id[i] = i;
                for (int i = 1; i <= n; i += 2) mat[i] = i + 1, mat[i + 1] = i;
                for (int times = 0, flag; times < 3; ) { //increase the iteration value
                ↪   for higher probability
                        memset(dis, 0, sizeof(dis));
                        memset(vis, 0, sizeof(vis));
                        top = 0; flag = 0;
                        for (int i = 1; i <= n; ++ i) {
                                if (spfa(id[i])) {
                                        flag = 1;
                                        int t = mat[stk[top - 1]], j = top - 2;
                                        while (stk[j] != stk[top - 1]) {
                                                mat[t] = stk[j];
                                                swap(t, mat[stk[j]]);
                                                -- j;
                                        }
                                        mat[t] = stk[j]; mat[stk[j]] = t;
                                        break;
                                }
                        }
                        if (!flag) times ++;
                        if (!flag) random_shuffle(id + 1, id + n + 1);
                }
                long long ans = 0;
                for (int i = 1; i <= n; ++ i) {
                        if (mat[i] <= n && i < mat[i]) {
                                if (G[i][mat[i]] != 0) ans += G[i][mat[i]], match[i] =
                                ↪   mat[i], match[mat[i]] = i;
                        }
                }
                return ans;
        }
};
```

28. Turbo Matching

565529bc2bd2546c12d5ac514d100004

```
struct Matching {
    int n; vector < int > *G, match, vis;
    bool dfs(int v) {
        vis[v] = 1;
        for(auto u: G[v]) if(!match[u] || (!vis[match[u]] && dfs(match[u]))) {
                match[v] = u; match[u] = v; return true;
        }
        return false;
    }
    Matching(int N, vector < int > *g) : n(N), G(g) {
        match.resize(n + 1, 0), vis.resize(n + 1, 0); bool ok = 1;
        while(ok) {
            ok = 0;
            for(int i = 1; i <= n; i++) if(!match[i] && dfs(i)) ok = 1;
            for(int i = 1; i <= n; i++) vis[i] = 0;
        }
    }
};
```

29. 2 SAT

b4b2a54c9e18151894b58de59501f2ca

```
struct twosat {
    int n;
    vector <vector <int> > G, R;
    vector <int> order, comp, ans;
    vector <bool> vis;

    twosat() {}
    twosat(int _n) : n(_n) {
        G.resize(n + n);
        R.resize(n + n);
        comp.resize(n + n);
        vis.resize(n + n);
        ans.resize(n);
        order.reserve(n + n);
    }

    void add_edge(int u, int v) {
        G[u].push_back(v);
        R[v].push_back(u);
    }

    /* 0-indexed, subtract if necessary */
    void add_clause(int u, bool fu, int v, bool fv) {
        add_edge(u << 1 | !fu, v << 1 | fv);
        add_edge(v << 1 | !fv, u << 1 | fu);
    }

    void dfs(int u) {
        vis[u] = true;
        for (const auto &v: G[u])
            if (!vis[v])    dfs(v);
        order.push_back(u);
    }

    void scc(int u, int id) {
        vis[u] = true, comp[u] = id;
        for (const auto &v: R[u])
            if (!vis[v])    scc(v, id);
    }

    bool run() {
        for (int i = 0; i < n + n; ++i)
            if (!vis[i])    dfs(i);

        fill(vis.begin(), vis.end(), false);
        reverse(order.begin(), order.end());

        int cnt = 0;
        for (const auto &v: order)
            if (!vis[v])    scc(v, ++cnt);

        for (int i = 0; i < n; ++i) {
            if (comp[i << 1] == comp[i << 1 | 1])    return false;
            ans[i] = comp[i << 1] < comp[i << 1 | 1];
        }

        return true;
    }
};
```

30. Ogólna transformata Adama

7d9acfc445ea90039f75c68e876dde44

```
const int mod = 998244353;
const int gen = 3;
const int inv2 = (mod + 1) / 2;

struct tran // przodek
{
        virtual int size () {}
        virtual void apply (int* A, bool inverse, cint& jump = 1) {}
};

struct xor_tran : tran // binarny XOR (Walsh-Hadamard)
{
        int size () { return 2; }
        void apply (int* A, bool inverse, cint& jump = 1)
        {
                int& na = A[0 * jump];
                int& nb = A[1 * jump];
                cint pa = na;
                cint pb = nb;
                na = pa + pb;
                nb = pa - pb;

                if (na >= mod)
                        na -= mod;
                if (nb < 0)
                        nb += mod;

                if (inverse)
                {
                        na = 1ll * na * inv2 % mod;
                        nb = 1ll * nb * inv2 % mod;
                }
        }
};

/* uwaga na modulo
struct and_tran : tran // binarny AND
{
        int size () { return 2; }
        void apply (int* A, bool inverse, cint& jump = 1)
        {
                int& na = A[0 * jump];
                int& nb = A[1 * jump];
                cint pa = na;
                cint pb = nb;
                if (inverse == false)
                {
                        na = pb;
                        nb = pa + pb;
                }
                if (inverse == true)
                {
                        na = -pa + pb;
                        nb = pa;
                }
        }
};

struct or_tran : tran // binarny OR (Mobius)
{
        int size () { return 2; }
        void apply (int* A, bool inverse, cint& jump = 1)
        {
                int& na = A[0 * jump];
```

```
                int& nb = A[1 * jump];
                cint pa = na;
                cint pb = nb;
                if (inverse == false)
                {
                        na = pa + pb;
                        nb = pa;
                }
                if (inverse == true)
                {
                        na = pb;
                        nb = pa - pb;
                }
        }
};
*/

struct sum_tran : tran // suma (Fourier)
{
        int mysize;
        int size () { return mysize; }
        sum_tran (int s) : mysize(s) {}

        void apply (int* A, bool inverse, cint& jump = 1)
        {
                int n, i, j, k, w, a, b, root;
                n = mysize;

//                dla jumpa > 1 odkomentować, zmienić argument A na B
                assert(jump == 1);
//                vector <int> A(n);
//                for (i=0; i<n; i++)
//                        A[i] = B[i * jump];

                for (i=1, j=0; i<n; i++){
                        for (k=n/2; j>=k; k/=2)
                                j ^= k;
                        j ^= k;
                        if (i < j)
                                swap(A[i], A[j]);
                }

                for (k=1; k<n; k*=2){
                        root = pw(gen, (mod-1)/(k+k));
                        if (inverse == true)
                                root = pw(root, mod-2);
                        for (i=0; i<n; i+=k+k){
                                w = 1;
                                for (j=i; j<i+k; j++){
                                        a = A[j];
                                        b = 1ll * A[j+k] * w % mod;
                                        A[j] = a + b;
                                        A[j+k] = a - b;
                                        if (A[j] >= mod)
                                                A[j] -= mod;
                                        if (A[j+k] < 0)
                                                A[j+k] += mod;
                                        w = 1ll * w * root % mod;
                                }
                        }
                }

                if (inverse == true){
                        w = pw(n, mod-2);
                        for (i=0; i<n; i++)
                                A[i] = 1ll * A[i] * w % mod;
                }

//                for (i=0; i<n; i++)
//                        B[i * jump] = A[i];
        }
};

struct russian : tran // niezależne podkonwolucje (kolejność od najmłodszej do
↪  najstarszej)
{
        int mysize;
        vector <tran*> T;
        int size () { return mysize; }

        russian (vector <tran*> vec = {})
        {
                T = vec;
                mysize = 1;
                for (tran* t : T)
                        mysize *= t->size();
        }

        void apply (int* A, bool inverse, cint& jump = 1)
        {
                int i, j, s = 1;
                for (tran* t : T){
                        for (i = 0; i < this->size(); i += s * t->size())
                                for (j = 0; j < s; j++)
                                        t->apply(A + (i + j) * jump, inverse, s * jump);
                        s *= t->size();
                }
        }
};

vector <int> convolute (tran& t, vector <int> A, vector <int> B) // konwolucja w
↪  ogólności
{
        int n = t.size();

        assert(A.size() <= n); A.resize(n, 0);
        assert(B.size() <= n); B.resize(n, 0);

        t.apply(&A[0], false);
        t.apply(&B[0], false);

        for (int i=0; i<n; i++)
                A[i] = 1ll * A[i] * B[i] % mod;

        t.apply(&A[0], true);

        return A;
}
```

31. Polynomial division

ea39ca4b66d63e4fcd3faa4f25f03fd6

```
vector <int> polydiv(vector <int> a, vector <int> b)
{
        int n, m, i, k, s;
        n = a.size();
        m = b.size();
        if (m > n) return {0};
        reverse(a.begin(), a.end());
        reverse(b.begin(), b.end());
        s = moddiv(1, b[0], MOD);
        for (int& x : b)
                x = modmul(x, s, MOD);
        k = n - m + 1;
        vector <int> r, v, g = {1};
        for (int w = 1; (1 << (w - 1)) < k; w++){
                r = polymul(g, g);
                r.resize(1 << w);
                v = b;
                v.resize(1 << w);
                r = polymul(r, v);
                r.resize(1 << w);
                g.resize(1 << w);
                polyadd(g, g);
                polysub(g, r);
        }
        g = polymul(g, a);
        g.resize(k);
        reverse(g.begin(), g.end());
        for (int& x : g)
                x = modmul(x, s, MOD);
        return g;
}
```

32. Karatsuba

ee27a5660b46a795436be1a725683b98

```
#pragma GCC optimize("Ofast,unroll-loops")
#pragma GCC target("avx,avx2,fma")
namespace {
    template<int n, typename T>
    void mult(const T *__restrict a, const T *__restrict b, T *__restrict res) {
        if (n <= 64) { // if length is small then naive multiplication if faster
            for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                    res[i + j] += a[i] * b[j];
                }
            }
        } else {
            const int mid = n / 2;
            alignas(64) T btmp[n], E[n] = {};
            auto atmp = btmp + mid;
            for (int i = 0; i < mid; i++) {
                atmp[i] = a[i] + a[i + mid]; // atmp(x) - sum of two halfs a(x)
                btmp[i] = b[i] + b[i + mid]; // btmp(x) - sum of two halfs b(x)
            }
            mult<mid>(atmp, btmp, E); // Calculate E(x) = (alow(x) + ahigh(x)) *
            ↪   (blow(x) + bhigh(x))
            mult<mid>(a + 0, b + 0, res); // Calculate rlow(x) = alow(x) * blow(x)
            mult<mid>(a + mid, b + mid, res + n); // Calculate rhigh(x) = ahigh(x) *
            ↪   bhigh(x)
            for (int i = 0; i < mid; i++) { // Then, calculate rmid(x) = E(x) - rlow(x)
            ↪   - rhigh(x) and write in memory
                const auto tmp = res[i + mid];
                res[i + mid] += E[i] - res[i] - res[i + 2 * mid];
                res[i + 2 * mid] += E[i + mid] - tmp - res[i + 3 * mid];
            }
        }
    }
}
```

33. NTT

06b16ee6c824db06b0859775d87d3992

```
typedef unsigned long long ull;
typedef unsigned T;
//ma byc unsigned int albo ll albo ull
//jak sie przekrecimy przez inta to zly wynik!
const T P = 998244353;
const T ROOT = 570984967;
const int MN=21;
T omega[1<<MN];
T pw (T x, T n) {
        T res = 1;
        while(n) {
                        if (n&1) res = (ull)res*x%P;
                        x = (ull)x*x%P;
                        n>>=1;
        }
        return res;
}
void fft(vector<T> &a, int n, bool inverse=false) {
        int N = 1<<n;
        a.insert(a.end(), N-SZ(a), 0);
        T root = pw(ROOT, (1<<23)/N*(inverse?(N-1):1));
        omega[0] = 1;
        rep(i, 1, N) omega[i] = (ull)omega[i-1]*root%P;
        rep(i, 0, n) {
                        rep(j, 0, 1<<i) {
                                        rep(k, 0, 1<<(n-i-1)) {
                                                        int s = (j<<(n-i))+k;
                                                        int t = s + (1<<(n-i-1));
                                                        T w = omega[k<<i];
                                                        T temp = a[s] + a[t];
                                                        if (temp >= P) temp -= P;
                                                        T t2 = a[s] - a[t] + P;
                                                        a[t] = (ull) w * t2 % P;
                                                        a[s] = temp;
                                        }
                        }
        }
        rep(i, 0, N) {
                        int x=i,y=0;
                        rep(j, 0, n)  y=(y<<1)+(x&1), x>>=1;
                        if (i<y) swap(a[i],a[y]);
        }
        if (inverse) {
                        T inv = pw(N, P-2);
                        rep(i, 0, N) a[i] = (ull)a[i]*inv%P;
        }
}
vector<T> conv(vector<T> A, vector<T> B) {
        int n = 31-__builtin_clz(2*(SZ(A)+SZ(B))-1);
        fft(A, n);
        fft(B, n);
        rep(i, 0, (1<<n)) A[i] = (ull) A[i] * B[i] % P;
        fft(A, n, true);
        return A;
}
vector<T> square(vector<T> A) {
        int n = 32-__builtin_clz(2*SZ(A)-1);
        fft(A, n);
        rep(i, 0, (1<<n)) A[i] = (ull) A[i] * A[i] % P;
        fft(A, n, true);
        return A;
}
```

34. Josephus

421dd32d75d61954519d94f3fc96fc4a

```
ll josephus(ll n, ll k, ll m) {
  m = n - m;
  if (k <= 1)return n - m;
  ll i = m;
  while (i < n) {
    ll r = (i - m + k - 2) / (k - 1);
    if ((i + r) > n) r = n - i;
    else if (!r) r = 1;
    i += r;
    m = (m + (r * k)) % i;
  } return m + 1;
}
```

35. Knights Moves in Infinity Grid

964cf59287e3516922e3704eeba9201e

```
#define ll long long int
// Minimum number of knight moves from (x,y) to
// (0,0) in non-negative infinite chessboard
ll knight_move(ll x, ll y) {
  ll cnt = max({(x + 1) / 2, (y + 1) / 2, (x + y + 2) / 3});
  while((cnt % 2) != (x + y) % 2) cnt++;
  if(x == 1 && !y) return 3;
  if(y == 1 && !x) return 3;
  if(x == y && x == 2) return 4;
  return cnt;
}
```

36. Mnożenie nimberów

26a891adbcc465fbff392ab63dbe4c0f

```
typedef unsigned long long T;
const int POWERS = 5;
const T vals[] = {1, 4, 16, 256, 65536, 4294967296ull};
/* Działa w log^2, jeśli trzeba szybciej to można skorzystać z rodzielności względem
↪  ksora (preprocessing) */
T multiply(T a, T b) {
        if(!a or !b)
                return 0;
        T pa = 0, pb = 0;
        for(int i = POWERS; i >= 0; --i) {
                if(!pa and a >= vals[i])        pa = vals[i];
                if(!pb and b >= vals[i])        pb = vals[i];
        }
        if(pa != pb) {
                if(pa < pb)        swap(pa, pb), swap(a, b);
                return pa * multiply(a / pa, b) ^ multiply(a % pa, b);
        }
        if(pa == 1) {
                if(a == 1 or b == 1)        return a * b;
                if(a != b)        return 1;
                return a ^ 1;
        }
        T a1 = a / pa, a2 = a % pa;
        T b1 = b / pb, b2 = b % pb;
        T p1 = multiply(a1, b1);
        T p2 = multiply(a2, b2);
        T p3 = multiply(a1 ^ a2, b1 ^ b2);
        T p4 = multiply(p1, pa >> 1);
        T p5 = p2 ^ p3;
        return p2 ^ p4 ^ p5 * pa;
}
```

37. FFT

4d6c99e53e9c2eab4ea8fffea35c5218

```
typedef double T;
const T PI = acos(-1.0);
struct C {
    T re, im;
    C () {}
    C (T r) : re(r), im(0) {}
    C (T r, T i) : re(r), im(i) {}
    C operator * (const C &c) const {
        return C(re * c.re - im * c.im, im * c.re + re * c.im);
    }
    C operator + (const C &c) const {
        return C(re + c.re, im + c.im);
    }
    C operator - (const C &c) const {
        return C(re - c.re, im - c.im);
    }
    void operator += (const C &c) {
        re += c.re, im += c.im;
    }
    C conj() const {
        return C(re, -im);
    }
};
```

```
typedef vector < C > VC;
typedef vector < LL > VLL;
inline void FFT(C *a, int n, int dir) {
    for(int i = 0, j = 0; i < n; i++) {
        if(i > j) swap(a[i], a[j]);
        for(int k = n >> 1; (j ^= k) < k; k >>= 1);
    }
    for(int p = 2; p <= n; p <<= 1) {
        C wn(cos(2.0 * dir * PI / p), sin(2.0 * dir * PI / p));
        for(int k = 0; k < n; k += p) {
            C w = 1;
            for(int j = 0; j < (p >> 1); j++) {
                C xx = a[k + j];
                C yy = w * a[k + j + (p >> 1)];
                a[k + j] = xx + yy;
                a[k + j + (p >> 1)] = xx - yy;
                w = w * wn;
            }
        }
    }
}
void multiply(VLL &a, VLL &b, VLL &res)  {
    int n = max(a.size(), b.size()), p = 2;
    while((p >> 1) < n) p <<= 1;
    C *fa = new C[p + 4];
    for(int i = 0; i < p; i++) fa[i] = 0;
    for(int i = 0; i < sz(a); i++) fa[i] += C(a[i], 0);
    for(int i = 0; i < sz(b); i++) fa[i] += C(0, b[i]);
    FFT(fa, p, 1);
    for(int i = 0; i <= p / 2; i++) {
        C bp = fa[i] + fa[p - i == p ? 0 : p - i].conj();
        C _q = fa[p - i == p ? 0 : p - i] - fa[i].conj();
        C q(_q.im, _q.re);
        fa[i] = (bp * q) * C(0.25);
        if(i > 0) fa[p - i] = fa[i].conj();
    }
    FFT(fa, p, -1);
    res.resize(sz(a) + sz(b) - 1);
    for(int i = 0; i < sz(res); i++) {
        res[i] = round(fa[i].re / p);
    }
    delete [] fa;
}
```

38. FFT Radecki

e7aa770b0e04b5fb8e2ee1dacac4a4cf

```
/* Prec. error max_ans/1e15 (2.5e18) for (long) doubles, so int rounding works
for doubles with answers 0.5e15, e.g. for sizes 2^20 and RANDOM ints in [0,45k],
assuming DBL_MANT_DIG=53 and LDBL_MANT_DIG=64. Consider normalizing and brute.*/
#define REP(i,n) for(int i = 0; i < int(n); ++i)
typedef double ld; // 'long double' is 2.2 times slower
struct C { ld re, im;
        C operator * (const C & he) const {
                return C{re * he.re - im * he.im,
                                re * he.im + im * he.re};
        }
        void operator += (const C & he) { re += he.re; im += he.im; }
};
void dft(vector<C> & a, bool rev) {
        const int n = a.size();
        for(int i = 1, k = 0; i < n; ++i) {
                for(int bit = n / 2; (k ^= bit) < bit; bit /= 2);;;
                if(i < k) swap(a[i], a[k]);
        }
        for(int len = 1, who = 0; len < n; len *= 2, ++who) {
                static vector<C> t[30];
                vector<C> & om = t[who];
                if(om.empty()) {
                        om.resize(len);
                        const ld ang = 2 * acosl(0) / len;
                        REP(i, len) om[i] = i%2 || !who ?
                                        C{cos(i*ang), sin(i*ang)} : t[who-1][i/2];
                }
                for(int i = 0; i < n; i += 2 * len)
                        REP(k, len) {
                                 const C x = a[i+k], y = a[i+k+len]
                                                * C{om[k].re, om[k].im * (rev ? -1 :
                                                ↪  1)};
                                a[i+k] += y;
                                a[i+k+len] = C{x.re - y.re, x.im - y.im};
                        }
        }
        if(rev) REP(i, n) a[i].re /= n;
}
template<typename T>vector<T> multiply(const vector<T> & a, const vector<T> & b,
                bool split = true, bool normalize = false) {
        if(a.empty() || b.empty()) return {};
        T big = 0; if(normalize) { // [0,B] into [-B/2, B/2]
                assert(a.size() == b.size()); // equal size!!!
                for(T x : a) big = max(big, x);
                for(T x : b) big = max(big, x);
                big /= 2;
        }
        int n = a.size() + b.size();
        vector<T> ans(n - 1);
        /* if(min(a.size(),b.size()) < 190) { // BRUTE FORCE
                REP(i, a.size()) REP(j, b.size()) ans[i+j] += a[i]*b[j];
                return ans; } */
        while(n&(n-1)) ++n;
        auto foo = [&](const vector<C> & w, int i, int k) {
                int j = i ? n - i : 0, r = k ? -1 : 1;
                return C{w[i].re + w[j].re * r, w[i].im
                                - w[j].im * r} * (k ? C{0, -0.5} : C{0.5, 0});
        };
        if(!split) { // standard fast version
                vector<C> in(n), done(n);
                REP(i, a.size()) in[i].re = a[i] - big;
                REP(i, b.size()) in[i].im = b[i] - big;
                dft(in, false);
                REP(i, n) done[i] = foo(in, i, 0) * foo(in, i, 1);
                dft(done, true);
                REP(i, ans.size()) ans[i] = is_integral<T>::value ?
                                llround(done[i].re) : done[i].re;
        //REP(i,ans.size())err=max(err,abs(done[i].re-ans[i]));
        }
        else { // Split big INTEGERS into pairs a1*M+a2,
                const T M = 1<<15; // where M = sqrt(max_absvalue).
                vector<C> t[2]; // This version is 2.2-2.5 times slower.
                REP(x, 2) {
                        t[x].resize(n);
                        auto & in = x ? b : a; // below use (in[i]-big) if normalized
                        REP(i, in.size()) t[x][i]=C{ld(in[i]%M), ld(in[i]/M)};
                        dft(t[x], false);
                }
                T mul = 1;
                for(int s = 0; s < 3; ++s, mul = (mul*M)%mod) {
                        vector<C> prod(n);
                        REP(x, 2) REP(y, 2) if(x + y == s) REP(i, n)
                                prod[i] += foo(t[0], i, x) * foo(t[1], i, y);
                        dft(prod, true); // remember: llround(prod[i].re)%MOD*mul !!!
                        REP(i, ans.size()) ans[i]=
                        ↪  (ans[i]+llround(prod[i].re)%mod*mul)%mod;
                }
        }
        if(normalize) {
                T so_far = 0;
                REP(i, ans.size()) {
                        if(i < (int) a.size()) so_far += a[i] + b[i];
                        else so_far -= a[i-a.size()] + b[i-a.size()];
                        ans[i] += big * so_far - big * big * min(i + 1, (int) ans.size()
                        ↪  - i);
                }
        }
        return ans;
}
```

39. Faktoryzacja Rho-Pollarda

2221309c189efb5908813dc33d7351b7

```
typedef long long int LL;
const int maxv = 50;
const int maxc = 5007;
const int maxp = 1e6 + 7;
int cnt;
int ptot;
int d[maxp];
int pr[maxp];
LL ans[maxc];
inline LL mod(LL a, LL n){
        if(a >= n)         a -= n;
        return a;
}
inline LL add(LL a, LL b, LL n){
        a += b; mod(a, n);
        return a;
}
inline LL mul(LL x, LL y, LL p) {
        LL ret = x * y - (LL)((long double)x * y / p + 0.5) * p;
        return ret < 0 ? ret + p : ret;
}
LL fast(LL x, LL k, LL p){
        LL ret = 1%p;
        for(; k > 0; k >>= 1, x = mul(x, x, p))
                (k & 1) && (ret = mul(ret, x, p));
        return ret;
}
bool rabin(LL n){
        if(n == 2) return 1;
        if(n < 2 || !(n & 1))
                return 0;
        LL s = 0, r = n - 1;
        for(; !(r & 1); r >>= 1, ++s);
        for(int i = 0; pr[i] < n && pr[i] < maxv; ++i) {
                LL cur = fast(pr[i], r, n), nxt;
                for(int j = 0; j < s; ++j) {
                        nxt = mul(cur, cur, n);
                        if(nxt == 1 && cur != 1 && cur != n - 1) return false;
                        cur = nxt;
                }
                if(cur != 1) return false;
        }
        return true;
}
LL gcd(LL a, LL b){
        LL tmp;
        while(b){
                tmp = b;
                b = a%b;
                a = tmp;
        }
        return a;
}
LL factor(LL n) {
        static LL seq[maxp];
        while(true){
                LL x = rand()%n, y = x, c = rand()%n;
                LL *px = seq, *py = seq, tim = 0, prd = 1;
                while(true){
                        *py++ = y = add(mul(y, y, n), c, n);
                        *py++ = y = add(mul(y, y, n), c, n);
                        if((x = *px++) == y) break;
                        LL tmp = prd;
                        prd = mul(prd, abs(y - x), n);
                        if(!prd) return gcd(tmp, n);
                        if((++tim) == maxv){
                                if((prd = gcd(prd, n)) > 1 && prd < n) return prd;
                                tim = 0;
                        }
                }
                if(tim && (prd = gcd(prd, n)) > 1 && prd < n) return prd;
        }
}
void decompose(LL n) {
        if(n < maxp){
                while(n > 1)
                        ans[cnt++] = d[n], n /= d[n];
        }
        else if(rabin(n))
                ans[cnt++] = n;
        else{
                LL fact = factor(n);
                decompose(fact), decompose(n / fact);
        }
}
void init(){
        d[1] = 1;
        for(int i = 2; i * i < maxp; ++i)
                if(!d[i])
                        for(int j = i * i; j < maxp; j += i)
                                d[j] = i;
        for(int i = 2; i < maxp; ++i)
                if(!d[i]){
                        d[i] = i;
                        pr[ptot++] = i;
                }
}
void clear(){
        cnt = 0;
}
```

40. Drzewo Sterna-Brocota

c508131f936fb1d35fde4f83ec43c823

```
typedef double D;
typedef long long int LL;
#define st first
#define nd second
#define PII pair <int, int>
const int INF = 1e9;
PII operator*(int t, PII a){
        return {a.st * t, a.nd * t};
}
PII operator+(PII a, PII b){
        return {a.st + b.st, a.nd + b.nd};
}
bool operator<=(PII a, PII b){
        return 1LL * a.st * b.nd <= 1LL * b.st * a.nd;
}
PII getLeft(PII &Left, PII &Right, PII &a, int &d){
        int p = 1, k = (a.nd - Left.nd) / Right.nd;
        if(k < 1)        assert(false);
        while(p < k){
                int m = (p + k + 1) / 2;
                PII maybe = m * Right + Left;
                if(maybe <= a)
                        p = m;
                else
                        k = m - 1;
        }
        return p * Right + Left;
}
PII getRight(PII &Left, PII &Right, PII &a, int &d){
        int p = 1, k = (d - Right.nd - 1) / Left.nd;
        if(k < 1)        assert(false);
        while(p < k){
                int m = (p + k + 1) / 2;
                PII maybe = m * Left + Right;
                if(maybe <= a)
                        k = m - 1;
                else
                        p = m;
        }
        return p * Left + Right;
}
PII getFraction(PII a, int d){        //Zwraca najmniejszy ulamek wiekszy niz a, o
↪ mianowniku < d, d >= 2, a < 1
        PII Left = {0, 1}, Right = {1, 1};
        while(true){
                PII Mid = Left + Right;
                if(Mid <= a)
                        Left = getLeft(Left, Right, a, d);
                else if(Mid.nd >= d)
                        break;
                else
                        Right = getRight(Left, Right, a, d);
        }
        assert(Right.nd < d);
        return Right;
}
```

41. Pierwiastek Dyskretny

3181eabf11c32bfb2d74eed98ebf27b5

```
int fast(int a, int b, int mod){
        int ret = 1;
        while(b){
                if(b & 1)
                        ret = (1LL * ret * a)%mod;
                b >>= 1;
                a = (1LL * a * a)%mod;
        }
        return ret;
}
int get(int p){
        int t = 2;
        while(fast(t, (p - 1) / 2, p) == 1)
                ++t;
        return t;
}
int dsr(int v, int p){          //Jeśli -1 to nie ma rozwiązania, wpp. rozwiązanie to r i
↪   -r, p jest pierwsze
        if(v == 0)
                return 0;
        if(p == 2)
                return 1;
        if(fast(v, (p - 1) / 2, p) == p - 1)
                return -1;
        int q = p - 1, s = 0;
        while(!(q & 1))
                q /= 2, ++s;
        if(s == 1)
                return fast(v, (p + 1) / 4, p);
        int z = get(p);
        int c = fast(z, q, p);
        int r = fast(v, (q + 1) / 2, p);
        int t = fast(v, q, p);
        int m = s;
        while(t != 1){
                int tt = t;
                int i = 0;
                while(tt != 1){
                        tt = (1LL * tt * tt)%p;
                        ++i;
                }
                int b = fast(c, fast(2, m - i - 1, p - 1), p);
                int b2 = (1LL * b * b)%p;
                r = (1LL * r * b)%p;
                t = (1LL * t * b2)%p;
                c = b2;
                m = i;
        }
        return r;
}
```

42. Berlekamp-Massey

0d0ab87e89132a8fab9e9a9b81638992

```
#define FOR(i, n) for(int i = 0; i < (n); i++)
#define sz(x) (int)(x).size()
const int N = 5005;
const int M = 1e9 + 7;
LL fast(LL a, LL n) {
    LL x = 1; a %= M;
    while(n) {
        if(n & 1) x = x * a % M;
        a = a * a % M; n >>= 1;
    }
    return x;
}
vector < LL > BM(vector < LL > x) {
    vector < LL > ls, cur;
    LL lf = 0, ld = 0;
    FOR(i, sz(x)) {
        LL t = 0;
        FOR(j, sz(cur)) t = (t + (LL)x[i - j - 1] * cur[j]) % M;
        if((t - x[i]) % M == 0) continue;
        if(cur.empty()) {
            cur.resize(i + 1);
            lf = i; ld = (t - x[i]) % M;
            continue;
        }
        LL k = -(x[i] - t) * fast(ld, M - 2) % M;
        vector < LL > c(i - lf - 1); c.push_back(k);
        for(auto y: ls) c.push_back(-y * k % M);
        if(sz(c) < sz(cur)) c.resize(sz(cur));
        FOR(j, sz(cur)) c[j] = (c[j] + cur[j]) % M;
        if(i - lf + sz(ls) >= sz(cur))
            ls = cur, lf = i, ld = (t - x[i]) % M;
        cur = c;
    }
    for(auto &y: cur) y = (y % M + M) % M;
    return cur;
}
int m;
LL a[N], h[N], t_[N], s[N], t[N];
void mull(LL *p, LL *q) {
    FOR(i, 2 * m) t_[i] = 0;
    FOR(i, m) if(p[i]) FOR(j, m)
        t_[i + j] = (t_[i + j] + p[i] * q[j]) % M;
    for(int i = 2 * m - 1; i >= m; i--) if(t_[i])
        for(int j = m - 1; ~j; --j)
            t_[i - j - 1] = (t_[i - j - 1] + t_[i] * h[j]) % M;
    FOR(i, m) p[i] = t_[i];
}
LL calc(LL k) {
    for(int i = m; ~i; i--) s[i] = t[i] = 0;
    s[0] = 1; (m != 1 ? t[1] = 1 : t[0] = h[0]);
    while(k) {
        if(k & 1) mull(s, t);
        mull(t, t); k >>= 1;
    }
    LL su = 0;
    FOR(i, m) su = (su + s[i] * a[i]) % M;
    return (su % M + M) % M;
}
LL nth_element(vector < LL > x, LL n) {
    if(n < (int)sz(x)) return x[n];
    vector < LL > v = BM(x); m = v.size(); if(!m) return 0;
    FOR(i, m) h[i] = v[i], a[i] = x[i];
    return calc(n);
}
```

43. Chińskie twierdzenie o resztach

5101903203c6e2f4fd3b35236a2dfc78

```
typedef long long T;
pair<T, T> gcd_ext(T a, T b)
{
    if(b == 0) return { 1, 0 };
    auto p = gcd_ext(b, a % b);
    return { p.second, p.first - a / b * p.second };
}
pair<T, T> CRT(vector<pair<T, T>> con)
{
    T k = 0, m = 1;
    for(auto c: con)
    {
        T k1 = k, m1 = m, k2 = c.first, m2 = c.second;
        auto q = gcd_ext(m1, m2);
        T gcd = m1 * q.first + m2 * q.second;
        m = m1 / gcd * m2;
        if(k1 % gcd != k2 % gcd)
            return { -1, -1 };
        k = (k1 / gcd) * m2 * q.second + (k2 / gcd) * m1 * q.first + k1 % gcd;
        k %= m;
        if(k < 0) k += m;
    }
    return { k, m };
}
```

44. Liczba punktów w trójkącie

ed44d297144492d6a9fb968a8a3fbfbe

```
LL go(LL n, LL p, LL q) {
    if(n * p < q) return n + 1;
    if(p < q) {
        LL rect = (n + 1) * (n * p / q + 1);
        LL diag = n / q + 1;
        return diag + rect - go(n * p / q, q, p);
    }
    LL k = p / q;
    return k * n * (n + 1) / 2 + go(n, p - k * q, q);
}
LL in_triangle(LL n, LL p, LL q) {
    LL d = __gcd(p, q);
    return go(n, p / d, q / d);
}
```

45. Sumy ciagów arytmetycznych

4e1e30dd460407d3ca69674351ea4e1f

```
ll sumsq(ll n) {
  return n / 2 * ((n - 1) | 1);
}
// \sum_{i = 0}^{n - 1}{(a + d * i) / m}, O(log m)
ll floor_sum(ll a, ll d, ll m, ll n) {
  ll res = d / m * sumsq(n) + a / m * n;
  d %= m; a %= m;
  if (!d) return res;
  ll to = (n * d + a) / m;
  return res + (n - 1) * to - floor_sum(m - 1 - a, m, d, to);
}
// \sum_{i = 0}^{n - 1}{(a + d * i) % m}
ll mod_sum(ll a, ll d, ll m, ll n) {
  a = ((a % m) + m) % m;
  d = ((d % m) + m) % m;
  return n * a + d * sumsq(n) - m * floor_sum(a, d, m, n);
}
```

46. Suma prefiksowa funkcji multiplikatywnej

ca2c8162f93b86c71e8efba7f759ce9a

```
/*      Prefix sum of (not necessarily?) multiplicative functions:
                p_f : the prefix sum of f(x) (1 <= x <= th).
                p_g : the prefix sum of g(x) (0 <= x <= N).
                p_c : the prefix sum of (f * g)(x) (0 <= x <= N).
                th : the thereshold, generally should be n ^ (2 / 3).
*/

struct prefix_mul
{
        typedef ll (*func) (ll);

        func p_f, p_g, p_c;
        ll n, th, inv;
        unordered_map <ll,ll> mem;

        prefix_mul(func p_f, func p_g, func p_c) : p_f(p_f), p_g(p_g), p_c(p_c) {}

        ll calc(ll x)
        {
                if (x <= th) return p_f(x);
                auto d = mem.find(x);
                if (d != mem.end()) return d->second;
                ll ans = 0;
                for (ll i = 2, la; i <= x; i = la + 1){
                        la = x / (x / i);
                        ans = ans + (p_g(la) - p_g(i - 1)) * calc(x / i);
                }
                ans = (p_c(x) - ans) / inv;
                return mem[x] = ans;
        }

        ll solve(ll n, ll th)
        {
                if (n <= 0) return 0;
                prefix_mul::n = n;
                prefix_mul::th = th;
                inv = p_g(1);
                return calc(n);
        }
};
```

47. Xor- And- Convolution

aa28e1d6bd7e255bafcc3ae14d55a779

```
typedef long long T;
int n, invn;
T mod;
void convolve(T *tab, vector<vector<int>> M) {
    for(int i = 1; i < n; i *= 2)
        for(int j = 0; j < n; j += 2 * i)
            for(int k = 0; k < i; k++)
            {
                T &a = tab[j + k], &b = tab[j + i + k];
                T na = (M[0][0] * a + M[0][1] * b) % mod, nb = (M[1][0] * a + M[1][1] *
                ↪ b) % mod;
                a = na; b = nb;
            }
}
void mult(T *a, T *b, T *c) {
    for(int i = 0; i < n; i++)
        c[i] = a[i] * b[i] % mod;
}
//tych funkcji uzywac
//N MUSI byc potega dwojki
void init(int N, T Mod) {
    mod = Mod;
    n = N;
    invn = inv(n); //Trzeba sobie napisać funkcję inv, która liczy odwrotność modulo
}
//UWAGA modyfikuje tablice a i b, robic kopie jesli potrzebne pozniej
void xor_conv(T *a, T *b, T *res) {
    convolve(a, { { 1, 1 }, { 1, -1 } });
    if(a != b) convolve(b, { { 1, 1 }, { 1, -1 } });
    mult(a, b, res);
    convolve(res, { { 1, 1 }, { 1, -1 } });
    for(int i = 0; i < n; i++)
        res[i] = res[i] * invn % mod;
}
void and_conv(T *a, T *b, T *res) {
    convolve(a, { { 0, 1 }, { 1, 1 } });
    if(a != b) convolve(b, { { 0, 1 }, { 1, 1 } });
    mult(a, b, res);
    convolve(res, { { -1, 1 }, { 1, 0 } });
}
void or_conv(T *a, T *b, T *res) {
    convolve(a, { { 1, 0 }, { 1, 1 } });
    if(a != b) convolve(b, { { 1, 0 }, { 1, 1 } });
    mult(a, b, res);
    convolve(res, { { 1, -1 }, { 0, 1 } });
}
```

48. Simplex

2c86cc90476d4687f17c0061f9956edf

```
const double EPS = 1e-7;
typedef long double T;
typedef vector <T> VT;
typedef vector < int > vi;
#define FOR(i,n) for(int i = 0; i < (n); i++)
vector < VT > A; VT b, c, res; vi kt, N; int m;
inline void pivot(int k, int l, int e) {
        int x = kt[l]; T p = A[l][e];
        FOR(i, k) A[l][i] /= p; b[l] /= p; N[e] = 0;
        FOR(i, m) if(i != l) b[i] -= A[i][e] * b[l], A[i][x] = A[i][e] * -A[l][x];
        FOR(j, k) if(N[j]) {
                c[j] -= c[e] * A[l][j];
                FOR(i, m) if(i != l) A[i][j] -= A[i][e] * A[l][j];
        }
        kt[l] = e; N[x] = 1; c[x] = c[e] * -A[l][x];
}
VT doit(int k) {
        VT res; T best;
        while(1) {
                int e = -1, l = -1; FOR(i, k) if(N[i] && c[i] > EPS) {e = i; break;}
                if(e == -1) break;
                FOR(i, m) if(A[i][e] > EPS && (l == -1 || best > b[i] / A[i][e]))
                        best = b[ l = i ]  / A[i][e];
                if(l == -1)
                        return VT();
                pivot(k, l, e);
        }
        res.resize(k, 0); FOR(i, m) res[kt[i]] = b[i];
        return res;
}
VT simplex(const vector < VT > &AA, const VT &bb, const VT &cc) {
        int n = AA[0].size(), k;
        m = AA.size(); k = n + m + 1; kt.resize(m); b = bb; c = cc; c.resize(n + m);
        A = AA; FOR(i, m) { A[i].resize(k); A[i][n + i] = 1; A[i][k - 1] = -1; kt[i] = n
        ↪  + i; }
        N = vi(k, 1); FOR(i, m) N[kt[i]] = 0;
        int pos = min_element(b.begin(), b.end()) - b.begin();
        if(b[pos] < -EPS) {
                c = VT(k, 0); c[k - 1] = -1; pivot(k, pos, k - 1); res = doit(k);
                if(res[k - 1] > EPS) return VT();
                FOR(i, m) if (kt[i] == k - 1) {
                        FOR(j, k - 1) if(N[j] && (A[i][j] < -EPS || EPS < A[i][j])) {
                                pivot(k, i, j); break;
                        }
                }
                c = cc; c.resize(k, 0); FOR(i, m) FOR(j, k) if(N[j]) c[j] -= c[kt[i]] *
                ↪  A[i][j];
        }
        res = doit(k - 1); if(!res.empty()) res.resize(n);
        return res;
}
```

49. Subset Convolution

    5ca2aa1714889771ee5c7e75a28c9a1d

```
typedef long long T;
const int B = 17, N = 1 << B;
const T mod = 1000000007;
T tA[B+1][N];
T tB[B+1][N];
T conv[B+1][N];
int cnt[N];
void sum_over_subset(T *a, T *b, int k = -1)
{
    for(int i = 0; i < N; i++)
        b[i] = (k == -1 || k == cnt[i]) * a[i];
    for(int i = 0; i < B; i++)
        for(int j = 0; j < N; j++)
            if(j & (1 << i))
                b[j] += b[j ^ (1 << i)];
    for(int i = 0; i < N; i++)
        b[i] %= mod;
}
void mobius_transform(T *a, T b[B+1][N])
{
    for(int i = 0; i <= B; i++)
        sum_over_subset(a, b[i], i);
}
void subset_conv(T *a, T *b, T *res)
{
    for(int i = 0; i < N; i++)
        cnt[i] = __builtin_popcount(i);
    mobius_transform(a, tA);
    if(a != b)
        mobius_transform(b, tB);
    else copy(tA[0], tA[0] + (B+1)*N, tB[0]);
    for(int i = 0; i <= B; i++)
        for(int j = 0; j < N; j++)
            conv[i][j] = 0;
    for(int i = 0; i <= B; i++)
        for(int j = 0; i + j <= B; j++)
            for(int k = 0; k < N; k++)
            {
                conv[i+j][k] += tA[i][k] * tB[j][k];
                if(conv[i+j][k] >= (1LL << 62))
                    conv[i+j][k] %= mod;
            }
    for(int i = 0; i <= B; i++)
        for(int j = 0; j < N; j++)
        {
            conv[i][j] %= mod;
            if(cnt[j] & 1)
                conv[i][j] *= -1;
        }
    for(int i = 0; i <= B; i++)
        sum_over_subset(conv[i], conv[i]);
    for(int i = 0; i < N; i++)
    {
        res[i] = conv[cnt[i]][i];
        if(cnt[i] & 1)
            res[i] *= -1;
    }
}
```

50. Fast Rand

db8d516d7e531668d3c8c57d1703abe6

```
int getInt(int a = INT_MIN, int b = INT_MAX){
        static mt19937
        ↪  rng(chrono::steady_clock::now().time_since_epoch().count());
        return uniform_int_distribution <int> (a, b)(rng);
}

long long getLL(long long a = LLONG_MIN, long long b = LLONG_MAX){
        static mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());
        return uniform_int_distribution <long long int> (a, b)(rng);
}

double getReal(double a = 0.0, double b = 1.0){
        static mt19937
        ↪  rng(chrono::steady_clock::now().time_since_epoch().count());
        return uniform_real_distribution <double> (a, b)(rng);
}
```

51. Yarin Sieve

dbd7f9d81b80e583a4cc973cf5149222

```
// This is the famous "Yarin sieve", for use when memory is tight.
#define MAXSIEVE 100000000 // All prime numbers up to this
#define MAXSIEVEHALF (MAXSIEVE/2)
#define MAXSQRT 5000 // sqrt(MAXSIEVE)/2
char a[MAXSIEVE/16+2];
#define isprime(n) (a[(n)>>4]&(1<<(((n)>>1)&7))) // Works when n is odd

int main()
{
  int i,j;
  memset(a,255,sizeof(a));
  a[0]=0xFE;
  for(i=1;i<MAXSQRT;i++)
  if (a[i>>3]&(1<<(i&7)))
  for(j=i+i+i+1;j<MAXSIEVEHALF;j+=i+i+1)
  a[j>>3]&=~(1<<(j&7));

  //isprime(n) - mowi czy n pierwsze
}
```

52. Green Hackenbush

cc6ba075e6892c806d17df34ace68424

```
struct GreenHack {
    /* Set ground should be called before adding edges! */
    int n;
    vector <int> id;
    vector <vector <int> > G;

    int T;
    vector <int> low, pre;

    GreenHack(int _n) : n(_n) {
        id.resize(n + 1);
        iota(id.begin(), id.end(), 0);

        G.resize(n + 1);
        low.resize(n + 1);
        pre.resize(n + 1);
    }

    void set_ground(int u) {
        id[u] = 0;
    }

    void add_edge(int u, int v) {
        G[id[u]].push_back(id[v]);
        if (id[u] != id[v]) {
            G[id[v]].push_back(id[u]);
        }
    }

    int dfs(int u, int p) {
        int ans = 0;
        pre[u] = low[u] = ++T;
        for (const int &v: G[u]) {
            if (v == p) {
                p = -1;
                continue;
            }

            if (pre[v] == 0) {
                int res = dfs(v, u);
                low[u] = min(low[u], low[v]);
                if (low[v] > pre[u]) {
                    ans ^= res + 1;
                } else {
                    ans ^= res ^ 1;
                }
            } else if (pre[v] < pre[u]) {
                low[u] = min(low[u], pre[v]);
            } else {
                ans ^= 1;
            }
        }

        return ans;
    }

    int run() {
        return dfs(0, -1);
    }
};
```

53. Red Blue Hackenbush

aa85012a35331797c1b1f69c95f26b77

```
struct Surreal {
    int value = 0, offset = 0;
    set <int> powers;

    void clear() {
        value = offset = 0;
        powers.clear();
    }

    int size() {
        return powers.size();
    }

    int sign() {
        const int tmp = 2 * value + !powers.empty();
        return tmp < 0 ? -1 : (tmp > 0);
    }

    int add_power(int power) {
        while (power) {
            if (!powers.count(power - offset)) {
                powers.insert(power - offset);
                break;
            }

            powers.erase(power - offset);
            --power;
        }

        return !power;
    }

    void operator += (const Surreal &v) {
        value += v.value;
        for (const int &power: v.powers) {
            value += add_power(power + v.offset);
        }
    }

    void divide(int power) {
        offset += power;
        int to_add = 0;

        for (int i = 0; i < power; ++i) {
            if (value & 1) {
                to_add += add_power(power - i);
            }

            value >>= 1;
        }

        value += to_add;
    }

    void get_next(int t) {
        int power = max(0, -t * value);
        value += t * (power + 1);

        if (value == -1 || (value == 1 && powers.empty())) {
            power++;
            value += t;
        }

        divide(power);
    }
};

struct RedBlueHack {
    /* Weights on edges should be -1 or 1 */
    int n;
    vector <int> id;
    vector <Surreal> ans;
    vector <vector <pair <int, int> > > G;

    RedBlueHack(int _n) : n(_n) {
        id.resize(n + 1);
        iota(id.begin(), id.end(), 0);

        ans.resize(n + 1);
        G.resize(n + 1);
    }

    void add_edge(int u, int v, int t) {
        G[u].push_back({v, t});
        G[v].push_back({u, t});
    }

    void dfs(int u, int p)
    {
        ans[u].clear();
        for (auto &[v, w]: G[u])
        {
            if (v == p) {
                continue;
            }

            dfs(v, u);
            ans[id[v]].get_next(w);

            if (ans[id[u]].size() < ans[id[v]].size()) {
                swap(id[u], id[v]);
            }

            ans[id[u]] += ans[id[v]];
        }
    }

    int run() {
        dfs(1, 1);
        return ans[id[1]].sign();
    }
};
```

54. Distinct K-th powers

192bcedab9c2be0ca62e73bc72c47976

```
// returns the number of distinct values of (a^k % p^cnt) over all integers a (p is
↪  prime)
// can be optimized by precalculating powers
// current complexity: O(cnt * cnt)
ll f(ll p, ll cnt, ll k) {
  if (cnt <= 0 or k == 0) return 1;
  if (p == 2) {
    if (cnt == 1) return 2;
    ll u = power(2, cnt - 2) / __gcd(k, power(2, cnt - 2));
    if (k % 2) u *= 2;
    return u + f(2, cnt - k, k);
  }
  ll phi = power(p, cnt) - power(p, cnt - 1);
  ll u = phi / __gcd(k, phi);
  return u + f(p, cnt - k, k);
}
// returns the number of distinct values of (a^k % n) over all integers a
ll yo(ll k, ll n) {
  ll ans = 1;
  for (ll i = 2; i * i <= n; i++) {
    if (n % i == 0) {
      int cnt = 0;
      while (n % i == 0) {
        cnt++;
        n /= i;
      }
      ans *= f(i, cnt, k);
    }
  }
  if (n > 1) ans *= f(n, 1, k);
  return ans;
}
```

55. Max Convolution

7c138e61281f52dde1cb2939c0dbd2ba

```
// a[i + 1] - a[i] >= a[i] - a[i - 1] -> convex
// b[i + 1] - b[i] >= b[i] - b[i - 1] -> convex
// compute ans(i + j) = max(a(i) + b(j))
vector<int> multiply(vector<int> a, vector<int> b) {
  int n = a.size() - 1, m = b.size() - 1;
  vector<int> ans(n + m + 1);
  int sum = a[0] + b[0]; ans[0] = sum;
  int l = 0, r = 0;
  while (l < n && r < m) {
    if (a[l + 1] - a[l] > b[r + 1] - b[r]) {
      sum += a[l + 1] - a[l];
      l++;
    } else {
      sum += b[r + 1] - b[r];
      r++;
    }
    ans[l + r] = sum;
  }
  while (l < n) sum += a[l + 1] - a[l], l++, ans[l + r] = sum;
  while (r < m) sum += b[r + 1] - b[r], r++, ans[l + r] = sum;
  return ans;
}
```

56. Liczba partycji

b9fc1294ad577760b8e822e37ab9fa3b

```
vector<mint> solve(int n) {
  vector<mint> ans(n + 1);
  vector<pair<int, int>> gp; // (sign, generalized pentagonal numbers)
  gp.emplace_back(0, 0);
  for (int i = 1; gp.back().second <= n; i++) {
    gp.emplace_back(i % 2 ? 1 : -1, i * (3 * i - 1) / 2);
    gp.emplace_back(i % 2 ? 1 : -1, i * (3 * i + 1) / 2);
  }
  ans[0] = 1;
  for (int i = 1; i <= n; i++) {
    for (auto it : gp) {
      if (i >= it.second) ans[i] += ans[i - it.second] * it.first;
      else break;
    }
  }
  return ans;
}
```

57. Aho-Corasick

396bcb66ecd505afb85059b4568b6ef7

```
struct aho {
        static const int A = 26;
        static const int off = 'a';

        int n;
        vector <int> par, fail;
        vector <char> last;
        vector <vector <int> > nxt;        //mozna mape przy duzym alfabecie

        aho() {
                n = 0;
                par = { 0 };
                nxt = { vector <int> (A, -1) };
                last = { 0 };
                fail = { };
        }

        int create(int parent, char letter) {
                ++v;
                nxt.push_back(vector <int> (A, -1));
                par.push_back(parent);
                last.push_back(letter);
                return v;
        }

        void add(string &s) {
                int cur = 0;
                for(auto c: s) {
                        if(nxt[cur][c - off] == -1)
                                nxt[cur][c - off] = create(cur, c);
                        cur = nxt[cur][c - off];
                }
        }

        void make() {
                queue <int> Q;
                Q.push(0);

                fail.resize(n + 1);
                while(!Q.empty()) {
                        int u = Q.front();
                        Q.pop();

                        fail[u] = u ? nxt[fail[par[u]]][last[u]] : 0;
                        for(int i = 0; i < A; ++i)
                                if(nxt[u][i] == -1)
                                        nxt[u][i] = u ? nxt[fail[u]][i] : 0;
                                else
                                        Q.push(nxt[u][i]);
                }
        }
};
```

58. Drzewo palindromiczne

44dcb230d61198956a6f35d6950e6316

```
struct node {
    int len;
    unordered_map<char, node*> ch;
    node *suf;
    char *pos = 0;
    node(int l = 0) : len(l), suf(this) { }
};
struct eertree {
    node *r0, *r1;
    vector<node*> nodes;
    eertree() {
        r0 = new node(0);
        r1 = new node(-1);
        r0->suf = r1;
    }
    vector<node*> update(char *c) {
        vector<node*> ans;
        node *cur = r1;
        while(*c) {
            char x = *c;
            while(x != c[-1 - cur->len])
                cur = cur->suf;
            if(!cur->ch.count(x)) {
                node *n = new node(cur->len + 2);
                if(n->len == 1)
                    n->suf = r0;
                else {
                    n->suf = cur->suf;
                    while(x != c[-1 - n->suf->len])
                        n->suf = n->suf->suf;
                    n->suf = n->suf->ch[x];
                }
                n->pos = c - n->len + 1;
                cur->ch[x] = n;
                nodes.push_back(n);
            }
            cur = cur->ch[x];
            ans.push_back(cur);
            c++;
        }
        return ans;
    }
};
```

59. All Substring LCS

c5282ef0a4b5607d9f10834d57249cee

```
const int N = 2002;
int f[N][N], g[N][N];
void solve(string s, string t)
{
    int  n = s.size(), m = t.size();
    s = "#" + s;
    t = "#" + t;
    for (int i = 1; i <= m; ++i) f[0][i] = i;
    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= m; ++j) {
            if (s[i] == t[j]) {
                f[i][j] = g[i][j - 1];
                g[i][j] = f[i - 1][j];
            }
            else {
                f[i][j] = max(f[i - 1][j], g[i][j - 1]);
                g[i][j] = min(g[i][j - 1], f[i - 1][j]);
            }
        }
    }
    for (int i = 1; i <= m; ++i) {
        for (int j = i, ans = 0; j <= m; ++j) {
            if (i > f[n][j]) ++ans;
            //ans is lcs of s and t[i, j]
        }
    }
}
```

60. Cyclic LCS

95d7ef1685641b9253ddd64e1bbe322c

```
const int N = 2010;
/*Cyclic Longest Common Subsequence
maximum of lcs(any cyclic shift of s, any cyclic shift of t)
O(nm)*/
int dp[N * 2][N], from[N * 2][N];
int solve(string s, string t) {
    int n = s.size(), m = t.size();
    auto eq = [&](int a, int b) {
        return s[(a - 1) % n] == t[(b - 1) % m];
    };
    dp[0][0] = from[0][0] = 0;
    for (int i = 0; i <= n * 2; ++i) {
        for (int j = 0; j <= m; ++j) {
            dp[i][j] = 0;
            if (j && dp[i][j - 1] > dp[i][j]) {
                dp[i][j] = dp[i][j - 1];
                from[i][j] = 0;
            }
            if (i && j && eq(i, j) && dp[i - 1][j - 1] + 1 > dp[i][j]) {
                dp[i][j] = dp[i - 1][j - 1] + 1;
                from[i][j] = 1;
            }
            if (i && dp[i - 1][j] > dp[i][j]) {
                dp[i][j] = dp[i - 1][j];
                from[i][j] = 2;
            }
        }
    }
    int ret = 0;
    for (int i = 0; i < n; ++i) {
        ret = max(ret, dp[i + n][m]);
        // re-root
        int x = i + 1, y = 0;
        while (y <= m && from[x][y] == 0) ++y;
        for (; y <= m && x <= n * 2; ++x) {
            from[x][y] = 0, --dp[x][m];
            if (x == n * 2) break;
            for (; y <= m; ++y) {
                if (from[x + 1][y] == 2) break;
                if (y + 1 <= m && from[x + 1][y + 1] == 1) {
                    ++y;
                    break;
                }
            }
        }
    }
    return ret;
}
```

61. KMP

d89d819c3e88f1fbed457befb8d21f89

```
vector<int> KMP(char *s) {
    int n = strlen(s + 1);
    vector<int> T(n + 1, 0);
    for (int i = 2; i <= n; i++) {
        int t = T[i - 1];
        while (t > 0 && s[i] != s[t + 1]) {
            t = T[t];
        }
        if (s[i] == s[t + 1])
            t++;
        T[i] = t;
    }
    return T;
}
```

62. Minimal Cyclic Shift

2b037c0badead7f860b57ca5d7b01414

```
string min_cyclic_string(string s) {
    s += s;
    int n = s.size();
    int i = 0, ans = 0;
    while (i < n / 2) {
        ans = i;
        int j = i + 1, k = i;
        while (j < n && s[k] <= s[j]) {
            if (s[k] < s[j])
                k = i;
            else
                k++;
            j++;
        }
        while (i <= k)
            i += j - k;
    }
    return s.substr(ans, n / 2);
}
```

63. Manacher

b9d984af25d554d8831d1e2746d0db3b

```
//wynik dla palindromu parzystego o środku między pozycjami i, i + 1 znajduje się 2 * i
↪  + 1
//wynik dla palindromu nieparzystego o środku na pozycji i znajduje się w 2 * i
struct manacher{
        int n;
        string s;
        vector <int> rad;
        void init(int N, string &in){
                n = N;
                s.resize(n + n);
                rad.resize(n + n);
                for(int i = 0; i < n + n; ++i)
                        s[i] = '#';                //znak niewystepujacy w tekscie
                for(int i = 0; i < n; ++i)
                        s[i + i] = in[i];
        }
        void go(){
                int m = 2 * n - 1;
                int x = 0;
                for(int i = 1; i < m; ++i){
                        int &r = rad[i] = 0;
                        if(i <= x + rad[x])
                                r = min(rad[x + x - i], x + rad[x] - i);
                        while(i - r - 1 >= 0 && i + r + 1 < m && s[i - r - 1] == s[i + r
                        ↪  + 1])
                                ++r;
                        if(i + r >= x + rad[x])
                                x = i;
                }
                for(int i = 0; i < m; ++i){
                        if(i - rad[i] == 0 || i + rad[i] == m - 1)
                                ++rad[i];
                        rad[i] /= 2;
                }
        }
};
```

64. Manacher Radecki

bec3c3aa22e11af07ca65aba34f8f78e

```
// @s[0..n-1]  - napis długości @n.
// @r[0..2n-2] - tablica promieni palindromów.
// s: a   b   a   a   b   a   a   c   a   a   b   b   b   b   a   a   c   a   c
// r: 0 0 1 0 0 3 0 0 2 0 0 1 0 0 3 0 0 1 0 0 0 1 1 6 1 1 0 0 0 1 0 0 1 0 1 0 0
void Manacher(const char* s, int n) {
  for (int i = 0, m = 0, k = 0, p = 0; i < 2 * n - 1; m = i++ - 1) {
    while (p < k and i / 2 + r[m] != k)
      r[i++] = min(r[m--], (k + 1 - p++) / 2);
    while (k + 1 < n and p > 0 and s[k + 1] == s[p - 1])
```

```
   k++, p--;
  r[i] = (k + 1 - p++) / 2;
 }
}
```

65. Tablica sufiksowa

4e725887a0b7d6b22f995808e00ed07c

```
struct SA {
    char s[N];
    int n, sa[N], x[N], y[N], lcp[N], cnt[N];

    bool dif(int a, int b, int k) {
        return y[a] != y[b] || (a + k <= n ? y[a + k] : -1) != (b + k <= n ? y[b
        ↪  + k] : -1);
    }

    void build() {
        n = strlen(s + 1);
        int A = 26;
        rep(i, 1, n) cnt[x[i] = s[i] - 'a' + 1]++;
        rep(i, 1, A) cnt[i] += cnt[i - 1];
        per(i, 1, n) sa[cnt[x[i]]--] = i;

        for (int k = 1; k < n; k *= 2) {
            int p = 0;
            rep(i, n - k + 1, n) y[++p] = i;
            rep(i, 1, n) if (sa[i] > k) y[++p] = sa[i] - k;
            rep(i, 1, A) cnt[i] = 0;
            rep(i, 1, n) cnt[x[i]]++;
            rep(i, 1, A) cnt[i] += cnt[i - 1];
            per(i, 1, n) sa[cnt[x[y[i]]]--] = y[i];
            swap(x, y);
            A = x[sa[1]] = 1;
            rep(i, 2, n) x[sa[i]] = (A += dif(sa[i - 1], sa[i], k));
            if (n == A) break;
        }

        int j = 0;
        rep(i, 1, n) {
            if (x[i] == n) {
                lcp[x[i]] = 0;
                continue;
            }
            int nxt = sa[x[i] + 1];
            while (max(i, nxt) + j <= n && s[i + j] == s[nxt + j]) j++;
            lcp[x[i]] = j;
            j = max(j - 1, 0);
        }
    }
};
```

66. Tablica sufiksowa (KMR)

5d82fcf670689a48eedc3c4ee85342d5

```
int n, logn;
vector<int> suf;
vector<vector<int>> kmr_tab;
void init(string input) {
  n = input.size();
  logn = 0;
  kmr_tab.clear();
  vector<int> val(input.begin(), input.end());
  vector<long long> pairs(n);
  suf.resize(n);
  for(int i = 0; i < n; i++)
    suf[i] = i;
  for(int t = 1;; logn++, t *= 2) {
    //zakomentowac jesli niepotrzebne, dodaje n log n do pamieci
    kmr_tab.push_back(val);
    if(t >= n) break;
    for(int i = 0; i < n; i++) {
      pairs[i] = (i + t < n ? val[i+t] : 0) | ((long long)val[i] << 30LL);
    }
    sort(suf.begin(), suf.end(),
      [&](int a, int b) { return pairs[a] < pairs[b]; });
    int k = 0;
    for(int i = 0; i < n; i++) {
      if(i == 0 || pairs[suf[i]] != pairs[suf[i-1]])
        k++;
      val[suf[i]] = k;
    }
  }
}
int common_prefix(int a, int b) {
  int v = 0;
  for(int i = logn; i >= 0; i--) {
    if(a == n || b == n) break;
    if(kmr_tab[i][a] == kmr_tab[i][b]) {
      int k = 1 << i;
      a += k;
      b += k;
      v += k;
    }
  }
  return v;
}
```

67. Tablica sufiksowa

```
5fa11091fb1d8efef4a3cb97123672ae
```

```
inline bool leq(int a1, int a2, int b1, int b2){
        return(a1 < b1 || a1 == b1 && a2 <= b2);
}
inline bool leq(int a1, int a2, int a3, int b1, int b2, int b3){
        return(a1 < b1 || a1 == b1 && leq(a2,a3, b2,b3));
}
static void radixPass(int* a, int* b, int* r, int n, int K){
        int* c = new int[K + 1];
        for (int i = 0; i <= K; i++) c[i] = 0;
        for (int i = 0; i < n; i++) c[r[a[i]]]++;
        for (int i = 0, sum = 0; i <= K; i++){
                int t = c[i]; c[i] = sum; sum += t;
        }
        for (int i = 0; i < n; i++)
                b[c[r[a[i]]]++] = a[i];
        delete [] c;
}
// require s[n]=s[n+1]=s[n+2]=0, n>=2
void suffixArray(int* s, int* SA, int n, int K){         //ma problemy dla n = 1
        int n0=(n+2)/3, n1=(n+1)/3, n2=n/3, n02=n0+n2;
        int* s12 = new int[n02 + 3]; s12[n02]= s12[n02+1]= s12[n02+2]=0;
        int* SA12 = new int[n02 + 3]; SA12[n02]=SA12[n02+1]=SA12[n02+2]=0;
        int* s0 = new int[n0];
        int* SA0 = new int[n0];
        for (int i=0, j=0; i < n+(n0-n1); i++) if (i%3 != 0) s12[j++] = i;
        radixPass(s12 , SA12, s+2, n02, K);
        radixPass(SA12, s12 , s+1, n02, K);
        radixPass(s12 , SA12, s , n02, K);
        int name = 0, c0 = -1, c1 = -1, c2 = -1;
        for (int i = 0; i < n02; i++){
                if (s[SA12[i]] != c0 || s[SA12[i]+1] != c1 || s[SA12[i]+2] != c2){
                        name++; c0 = s[SA12[i]]; c1 = s[SA12[i]+1]; c2 = s[SA12[i]+2];
                }
                if (SA12[i] % 3 == 1)
                        s12[SA12[i]/3] = name;
                else
                        s12[SA12[i]/3 + n0] = name;
        }
        if (name < n02) {
                suffixArray(s12, SA12, n02, name);
                for (int i = 0; i < n02; i++) s12[SA12[i]] = i + 1;
        }
        else
                for (int i = 0; i < n02; i++) SA12[s12[i] - 1] = i;
        for (int i=0, j=0; i < n02; i++) if (SA12[i] < n0) s0[j++] = 3*SA12[i];
        radixPass(s0, SA0, s, n0, K);
        for (int p=0, t=n0-n1, k=0; k < n; k++){
                #define GetI() (SA12[t] < n0 ? SA12[t]*3+1: (SA12[t] - n0) * 3 + 2)
                int i = GetI();
                int j = SA0[p];
                if(SA12[t] < n0 ? leq(s[i], s12[SA12[t] + n0], s[j], s12[j/3]) :
                ↪  leq(s[i],s[i+1],s12[SA12[t]-n0+1], s[j],s[j+1],s12[j/3+n0]))){
                        SA[k] = i; t++;
                        if (t == n02)
                                for (k++; p < n0; p++, k++) SA[k] = SA0[p];
                }
                else{
                        SA[k] = j; p++;
                        if (p == n0)            for (k++; t < n02; t++, k++) SA[k] = GetI();
                }
        }
        delete [] s12; delete [] SA12; delete [] SA0; delete [] s0;
}
void lcp(int *s, int *SA, int *ans, int n){
        int* tmp = new int[n];
        int last = 0;
        for(int i = 0; i < n; ++i)
                tmp[SA[i]] = i;
        for(int i = 0; i < n; ++i){
                if(last)        --last;
                if(tmp[i] == 0)         continue;
                int who = SA[tmp[i] - 1];
                while(s[i + last] == s[who + last])
                        ++last;
                ans[tmp[who]] = last;
        }
        delete [] tmp;
}
```

68. Automat sufiksowy

```
9d23628122d8f32efb9e4496ad25441d
```

```
// len -> largest string length of the corresponding endpos-equivalent class
// link -> longest suffix that is another endpos-equivalent class.
// firstpos -> 1 indexed end position of the first occurrence of the largest string of
↪  that node
// minlen(v) -> smallest string of node v = len(link(v)) + 1
// terminal nodes -> store the suffixes
struct SuffixAutomaton {
    struct node {
        int len, link, firstpos;
        map<char, int> nxt;
    };
    int sz, last;
    vector<node> t;
    vector<int> terminal;
    vector<long long> dp;
    vector<vector<int>> g;
    SuffixAutomaton() {}
    SuffixAutomaton(int n) {
        t.resize(2 * n); terminal.resize(2 * n, 0);
        dp.resize(2 * n, -1); sz = 1; last = 0;
        g.resize(2 * n);
        t[0].len = 0; t[0].link = -1; t[0].firstpos = 0;
    }
    void extend(char c) {
        int p = last;
        if (t[p].nxt.count(c)) {
            int q = t[p].nxt[c];
            if (t[q].len == t[p].len + 1) {
                last = q;
                return;
            }
            int clone = sz++;
            t[clone] = t[q];
            t[clone].len = t[p].len + 1;
            t[q].link = clone;
            last = clone;
            while (p != -1 && t[p].nxt[c] == q) {
                t[p].nxt[c] = clone;
                p = t[p].link;
            }
            return;
        }
        int cur = sz++;
        t[cur].len = t[last].len + 1;
        t[cur].firstpos = t[cur].len;
        p = last;
        while (p != -1 && !t[p].nxt.count(c)) {
            t[p].nxt[c] = cur;
            p = t[p].link;
        }
        if (p == -1) t[cur].link = 0;
        else {
            int q = t[p].nxt[c];
            if (t[p].len + 1 == t[q].len) t[cur].link = q;
            else {
                int clone = sz++;
                t[clone] = t[q];
                t[clone].len = t[p].len + 1;
                while (p != -1 && t[p].nxt[c] == q) {
                    t[p].nxt[c] = clone;
                    p = t[p].link;
                }
                t[q].link = t[cur].link = clone;
            }
        }
        last = cur;
    }
    void build_tree() {
        for (int i = 1; i < sz; i++) g[t[i].link].push_back(i);
    }
    void build(string &s) {
        for (auto x: s) {
            extend(x);
            terminal[last] = 1;
        }
        build_tree();
    }
    long long cnt(int i) { //number of times i-th node occurs in the string
        if (dp[i] != -1) return dp[i];
        long long ret = terminal[i];
        for (auto &x: g[i]) ret += cnt(x);
        return dp[i] = ret;
    }
};
```

69. Tablice PREF

8319c71e02aa9ee6d697aa03dc9db4f1

```
//numeracja od zera
struct pref{
        int n;
        string s;
        vector <int> rad;
        void init(int N, string &in){
                n = N;
                s = in;
                rad.resize(n);
        }
        void go(){
                int x = 0; rad[0] = -1;
                for(int i = 1; i < n; ++i){
                        int &r = rad[i] = -1;
                        if(i <= x + rad[x])
                                r = min(rad[i - x], x + rad[x] - i);
                        while(i + r + 1 < m && s[r + 1] == s[i + r + 1])
                                ++r;
                        if(i + r >= x + rad[x])
                                x = i;
                }
        }
};
```

70. Wzory

Liczby pierwsze: $10^9 + 123, 10^9 + 321, 999999929, 999999937, 10^{18} + 9$.

Sumy: $\sum_{k=0}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$, $\sum_{k=0}^{n} k^3 = \frac{n^2(n+1)^2}{4}$, $\sum_{i=n}^{m} \binom{i}{n} = \binom{m+1}{n+1}$, $\sum_{i=0}^{k} \binom{n}{i}\binom{m}{k-i} = \binom{n+m}{k}$.

Całki: $\int \frac{1}{ax+b}dx = \frac{1}{a}\ln|ax+b|$, $\int \tan x dx = -\ln|\cos x|$, $\int \frac{1}{x^2+a^2}dx = \frac{1}{a}\arctan\frac{x}{a}$, $\int \frac{1}{x^2-a^2}dx = \frac{1}{2a}\ln|\frac{x-a}{x+a}|$, $\int \frac{1}{\sqrt{a^2-x^2}}dx = \arcsin\frac{x}{a}$, $\int \frac{1}{\sqrt{x^2+q}}dx = \ln|x+\sqrt{x^2+q}|$, $\int a^x dx = \frac{a^x}{\ln a}$.

Liczby Catalana: $C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i}$, $C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1}$, $C_{n+1} = C_n \frac{4n+2}{n+2}$, $1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 74290, \ldots$

Suma dzielników: $\sigma(n) = \sigma(p_1^{\alpha_1}, \ldots, p_k^{\alpha_k}) = \prod_{i=1}^{k} \frac{p_i^{\alpha_i+1}-1}{p_i-1}$.

Funkcja Eulera: $\phi(p^k) = p^k - p^{k-1}, \phi(ab) = \phi(a)\phi(b)$ dla $a \perp b, \sum_{d|n} \phi(d) = n$.

Funckja Mobiusa: 1 dla liczb bezkwadratowych z parzystą liczbą czynników, -1 – nieparzystą, 0 dla liczb nie bezkwadratowych. Inaczej $\mu(n)$ to suma pierwotnych pierwiastków z jedności stopnia n, $\sum_{d|n} \mu(d) = 0$ dla $n > 1$.

Związek między ϕ a μ: $\phi(n) = \sum_{d|n} \mu(d)\frac{n}{d}$.

Problem znaczków pocztowych: Niech a, b względnie pierwsze. Jest dokładnie $\frac{1}{2}(a-1)(b-1)$ liczb, których nie da się zapisać w postaci $ax + by(x, y \leq 0)$. Największa z nich to $(a-1)(b-1) - 1$.

Lemat Burnside'a: Liczba orbit grupy G na zbiorze X: $|X/G| = \frac{1}{|G|}\sum_{g \in G} |X_g|$, gdzie $X_g = \{x \in X : g(x) = x\}$. ("Średnia liczba punktów stałych")

Metoda Simpsona: $\int_a^{b=a+2h} f(x)dx = \frac{b-a}{6}(f(a) + 4f(a+h) + f(b)) + O(h^5 f^{(4)}(\xi))$.

Liczby Stirlinga pierwszego rodzaju: Opisują liczbę sposobów na rozmieszczenie n liczb w k cyklach, $\left[{n \atop k}\right] = (n-1)\left[{n-1 \atop k}\right] + \left[{n-1 \atop k-1}\right]$.

Liczby Stirlinga drugiego rodzaju: Opisują liczbę sposobów podziału zbioru n elementowego na k niepustych podzbiorów, $\left\{{n \atop k}\right\} = k\left\{{n-1 \atop k}\right\} + \left\{{n-1 \atop k-1}\right\}$, $\left\{{n \atop k}\right\} = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j}j^n$.

Liczby Bella: Liczba podziałów zbioru n elementowego, $\mathcal{B}_{n+1} = \sum_{k=0}^{n}\binom{n}{k}\mathcal{B}_k$.

Nieuporządkowania: Permutacje bez elementu stałego, $!n = \lfloor \frac{n!}{e} + \frac{1}{2} \rfloor$.

Liczby harmoniczne: $H_n = \sum_{k=1}^{n}\frac{1}{k}$, $\frac{1}{2n+1} < H_n - \ln n - \gamma < \frac{1}{2n}$, $\gamma = 0.57721\,56649\,01532\,86060\,65120\ldots$

Wzór Picka: $P = W + \frac{B}{2} - 1$ gdzie P – pole, W — wewnętrzne, B – brzegowe.

Trygonometria: $\sin(\alpha + \beta) = \sin(\alpha)\cos(\beta) + \cos(\alpha)\sin(\beta)$, $\cos(\alpha + \beta) = \cos(\alpha)\cos(\beta) - \sin(\alpha)\sin(\beta)$, tw. sinusów: $\frac{a}{\sin(\alpha)} = \frac{b}{\sin(\beta)} = \frac{c}{\sin(\gamma)} = 2R$, tw. cosinusów: $c^2 = a^2 + b^2 - 2ab\cos(\gamma)$, $R = \frac{abc}{4S}$, $r = \frac{2S}{a+b+c}$, $S = \sqrt{s(s-a)(s-b)(s-c)}$, gdzie S – pole trójkąta, $s = \frac{a+b+c}{2}$, r, R – promień okręgu wpisanego/opisanego.

Reguła Warnsdorffa obchodzenia skoczkiem szachownicy: W każdym kroku idź na pole, z którego można zrobić najmniejszą liczbę ruchów do nieodwiedzonych pól.

Optymalizacja Knutha: $dp[i][j] = \min_{i<k<j}\{dp[i][k] + dp[k][j]\} + C[i][j]$, potrzebujemy $opt[i][j-1] \leq opt[i][j] \leq opt[i+1][j]$, gdzie $opt[i][j]$ daje najmniejsze optymalne k dla $dp[i][j]$, wystarcza też $C[a][c] + C[b][d] \leq C[a][d] + C[b][c]$ i $C[b][c] \leq C[a][d]$, dla wszytskich $a \leq b \leq c \leq d$.

Pokrycie wierzchołkowe i zbiór niezależny: Niech M, C, I – maksymalne skojarzenie, minimalne pokrycie wierzchołkowe i maksymalny zbiór niezależny, wtedy $|M| \leq |C| = N - |I|$, równość zachodzi dla grafów dwudzielnych. Dodatkowo $C^c = I$ (zawsze). Znajdowanie C, I (dla grafu dwudzielnego (A, B)): łączymy źródło z wierzchołkami z A, wierzchołki z B z ujściem (przepustowość taka jak wagi wierzchołków lub 1 dla nieważonego), krawędzie między A i B mają przepustowość ∞. Znajdujemy minimalne cięcie (S, T). Wtedy $C = (A \cap T) \cup (B \cap S)$ i $I = (A \cap S) \cup (B \cap T)$.

Macierz sąsiedztwa a liczba drzew spinających: Niech macierz $T = [t_{ij}]$, gdzie t_{ij} to liczba krawędzi z wierzchołka i do j dla $i \neq j$, $t_{ii} = -\deg(i)$. Liczba drzew spinających jest równa wyznacznikowi macierzy T po usunięciu k-tego wiersza i k-tej kolumny (k dowolne). Uwaga: dla niespójnych odpalać osobno dla każdej spójnej składowej.

Macierz a perfect matching: Tutte matrix: $A_{ij} = x_{ij}$ if $(i, j) \in E$ and $i < j$, $-x_{ij}$ if $i > j$, 0 if there is no edge.

Tw. Erdősa-Gallai: Ciąg $d_1, d_2, ..., d_n$ $(n - 1 \geq d_1 \geq \cdots \geq d_n \geq 0)$ jest ciągiem stopni wierzchołków pewnego nieskierowanego grafu prostego $\iff 2|\sum d_i$ i $(\forall k \in \{1, \ldots, n-1\})\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n}\min(k, d_i)$.

Liczby Bernoulliego: $B_0 = 1$; $\sum_{k=0}^{m}\binom{m+1}{k}B_k = 0$; $1, \frac{-1}{2}, \frac{1}{6}, 0, \frac{-1}{30}, ...$; $\sum_{v=1}^{n} v^k = \frac{1}{k+1}\sum_{j=0}^{k}\binom{k+1}{j}B_j n^{k+1-j}$

Dni tygodnia: 01.01.1600 – sobota, 01.01.1900 – poniedziałek, 13.06.2042 – piątek, 01.04.2008 – wtorekm 31.12.1999 - piątek, 01.01.3000 – środa, 04.04.2019 – czwartek (dzień finałów).