ACM 模板

数据结构

并查集

```
struct dsu{
    int n;
    vector<int> fa;
    dsu(int _n) :n(_n){
        fa.resize(n + 1);
        iota(fa.begin(), fa.end(), 0);
    }
    int find(int x){
        return x == fa[x] ? x : fa[x] = find(fa[x]);
    }
    int merge(int x, int y){
        int fax = find(x), fay = find(y);
        if(fax == fay)return 0; // 一个集合
        return fa[find(x)] = find(y); // 合并到哪个集合了
    }
};
```

树状数组

一维

```
template<class T>
struct Fenwick_tree {
    Fenwick_tree(int n) : n(n), tree(n + 1, 0) {}
    T query(int l, int r) {
        auto query = [&](int pos) {
            T res = 0;
            while (pos) { res += tree[pos]; pos -= lowbit(pos); }
            return res;
        };
        return query(r) - query(l - 1);
    }
    void update(int pos, T num) {
        while (pos <= n) { tree[pos] += num; pos += lowbit(pos); }
    }
    private:
    int n;
    vector<T> tree;
};
```

二维

```
template<class T>
struct Fenwick_tree_2 {
    Fenwick_tree_2(int n, int m) :n(n), m(m), tree(n + 1, vector<T>(m + 1)) {}
    T query(int l1, int r1, int l2, int r2) {
        auto query = [&](int l, int r) {
            T res = 0;
            for (int i = l; i; i -= lowbit(i))
                for (int j = r; j; j -= lowbit(j)) res += tree[i][j];
            return res;
        };
        return query(l2, r2) - query(l2, r1 - 1) - query(l1 - 1, r2) + query(l1 -
1, r1 - 1);
    }
    void update(int x, int y, T num) {
        for (int i = x; i <= n; i += lowbit(i))
            for (int j = y; j <= m; j += lowbit(j)) tree[i][j] += num;
    }
    private:
    int n, m;
    vector<vector<T>> tree;
};
```

三维

```
template<class T>
struct Fenwick_tree_3 {
    Fenwick_tree_3(int n, int m, int k) :n(n), m(m), k(k), tree(n + 1,
vector<vector<T>>(m + 1, vector<T>(k + 1))) {}
    T query(int a, int b, int c, int d, int e, int f) {
        auto query = [&](int x, int y, int z) {
            T res = 0;
            for (int i = x; i; i -= lowbit(i))
                for (int j = y; j; j -= lowbit(j))
                    for (int p = z; p; p -= lowbit(p)) res += tree[i][j][p];
            return res;
        };
        T res = query(d, e, f);
        res -= query(a - 1, e, f) + query(d, b - 1, f) + query(d, e, c - 1);
        res += query(a - 1, b - 1, f) + query(a - 1, e, c - 1) + query(d, b - 1, c
- 1);
        res -= query(a - 1, b - 1, c - 1);
        return res;
    }
    void update(int x, int y, int z, T num) {
        for (int i = x; i <= n; i += lowbit(i))
            for (int j = y; j <= m; j += lowbit(j))
                for (int p = z; p <= k; p += lowbit(p)) tree[i][j][p] += num;
    }
    private:
    int n, m, k;
```

```
    vector<vector<vector<T>>> tree;
};
```

线段树

```
template <class Data, class Num>
struct Segment_Tree{
    inline void update(int l, int r, Num x){ update(1, l, r, x); }
    inline Data query(int l, int r){ return query(1, l, r); }
    Segment_Tree(vector<Data>& a){
        n = a.size();
        tree.assign(n * 4 + 1, {});
        build(a, 1, 1, n);
    }
private:
    int n;
    struct Tree{ int l, r; Data data; };
    vector<Tree> tree;
    inline void pushup(int pos){
        tree[pos].data = tree[pos << 1].data + tree[pos << 1 | 1].data;
    }
    inline void pushdown(int pos){
        tree[pos << 1].data = tree[pos << 1].data + tree[pos].data.lazytag;
        tree[pos << 1 | 1].data = tree[pos << 1 | 1].data +
tree[pos].data.lazytag;
        tree[pos].data.lazytag = Num::zero();
    }
    void build(vector<Data>& a, int pos, int l, int r){
        tree[pos].l = l; tree[pos].r = r;
        if(l == r){ tree[pos].data = a[l - 1]; return; }
        int mid = (tree[pos].l + tree[pos].r) >> 1;
        build(a, pos << 1, l, mid);
        build(a, pos << 1 | 1, mid + 1, r);
        pushup(pos);
    }
    void update(int pos, int& l, int& r, Num& x){
        if(l > tree[pos].r || r < tree[pos].l)return;
        if(l <= tree[pos].l && tree[pos].r <= r){ tree[pos].data = tree[pos].data
+ x; return; }
        pushdown(pos);
        update(pos << 1, l, r, x); update(pos << 1 | 1, l, r, x);
        pushup(pos);
    }
    Data query(int pos, int& l, int& r){
        if(l > tree[pos].r || r < tree[pos].l)return Data::zero();
        if(l <= tree[pos].l && tree[pos].r <= r)return tree[pos].data;
        pushdown(pos);
        return query(pos << 1, l, r) + query(pos << 1 | 1, l, r);
    }
};
struct Num{
```

```
    ll add;
    inline static Num zero(){ return { 0 }; }
    inline Num operator+(Num b){ return { add + b.add }; }
};
struct Data{
    ll sum, len;
    Num lazytag;
    inline static Data zero(){ return { 0,0,Num::zero() }; }
    inline Data operator+(Num b){ return { sum + len * b.add,len,lazytag + b }; }
    inline Data operator+(Data b){ return { sum + b.sum,len + b.len,Num::zero() };
}
};
```

可持久化线段树

```
constexpr int MAXN = 200000;
vector<int> root(MAXN << 5);
struct Persistent_seg {
    int n;
    struct Data {
        int ls, rs;
        int val;
    };
    vector<Data> tree;
    Persistent_seg(int n, vector<int>& a):n(n) { root[0] = build(1, n, a); }
    int build(int l, int r, vector<int>& a) {
        if (l == r) {
            tree.push_back({ 0,0,a[l] });
            return tree.size() - 1;
        }
        int mid = l + r >> 1;
        int ls = build(l, mid, a), rs = build(mid + 1, r, a);
        tree.push_back({ ls,rs,tree[ls].val + tree[rs].val });
        return tree.size() - 1;
    }
    int update(int rt, const int& idx, const int& val, int l, int r) {
        if (l == r) {
            tree.push_back({ 0,0,tree[rt].val + val });
            return tree.size() - 1;
        }
        int mid = l + r >> 1, ls = tree[rt].ls, rs = tree[rt].rs;
        if (idx <= mid)ls = update(ls, idx, val, l, mid);
        else rs = update(rs, idx, val, mid + 1, r);
        tree.push_back({ ls,rs,tree[ls].val + tree[rs].val });
        return tree.size() - 1;
    }
    int query(int rt1, int rt2, int k, int l, int r) {
        if (l == r)return l;
        int mid = l + r >> 1;
        int lcnt = tree[tree[rt2].ls].val - tree[tree[rt1].ls].val;
        if (k <= lcnt)return query(tree[rt1].ls, tree[rt2].ls, k, l, mid);
```

```
        else return query(tree[rt1].rs, tree[rt2].rs, k - lcnt, mid + 1, r);
    }
};
```

图论

存图

```
struct Graph{
    int n;
    struct Edge{ int to, w; };
    vector<vector<Edge>> graph;
    Graph(int _n){ n = _n; graph.assign(n + 1, vector<Edge>()); };
    void add(int u, int v, int w){ graph[u].push_back({ v,w }); }
};
```

最短路

dijkstra

```
void dij(Graph& graph, vector<int>& dis, int t){
    vector<int> visit(graph.n + 1, 0);
    priority_queue<pair<int, int>> que;
    dis[t] = 0;
    que.emplace(0, t);
    while(!que.empty()){
        int u = que.top().second; que.pop();
        if(visit[u])continue;
        visit[u] = 1;
        for(auto& [to, w] : graph.graph[u]){
            if(dis[to] > dis[u] + w){
                dis[to] = dis[u] + w;
                que.emplace(-dis[to], to);
            }
        }
    }
}
```

树上问题

最近公公祖先

倍增法

```
vector<int> dep;
vector<array<int, 21>> fa;
```

```
dep.assign(n + 1, 0);
fa.assign(n + 1, array<int, 21>{});
void binary_jump(int root){
    function<void(int)> dfs = [&](int t){
        dep[t] = dep[fa[t][0]] + 1;
        for(auto& [to] : graph[t]){
            if(to == fa[t][0])continue;
            fa[to][0] = t;
            dfs(to);
        }
    };
    dfs(root);
    for(int j = 1; j <= 20; j++)
        for(int i = 1; i <= n; i++)
            fa[i][j] = fa[fa[i][j - 1]][j - 1];
}
int lca(int x, int y){
    if(dep[x] < dep[y])swap(x, y);
    for(int i = 20; i >= 0; i--){
        if(dep[fa[x][i]] >= dep[y])x = fa[x][i];
    }
    if(x == y)return x;
    for(int i = 20; i >= 0; i--){
        if(fa[x][i] != fa[y][i]){
            x = fa[x][i];
            y = fa[y][i];
        }
    }
    return fa[x][0];
}
```

树剖

```
int lca(int x, int y){
    while(top[x] != top[y]){
        if(dep[top[x]] < dep[top[y]])swap(x, y);
        x = fa[top[x]];
    }
    if(dep[x] < dep[y])swap(x, y);
    return y;
}
```

树链剖分

```
vector<int> fa, siz, dep, son, dfn, rnk, top;
fa.assign(n + 1, 0);
siz.assign(n + 1, 0);
dep.assign(n + 1, 0);
```

```
son.assign(n + 1, 0);
dfn.assign(n + 1, 0);
rnk.assign(n + 1, 0);
top.assign(n + 1, 0);
void hld(int root){
    function<void(int)> dfs1 = [&](int t){
        dep[t] = dep[fa[t]] + 1;
        siz[t] = 1;
        for(auto& [to, w] : graph[t]){
            if(to == fa[t])continue;
            fa[to] = t;
            dfs1(to);
            if(siz[son[t]] < siz[to])son[t] = to;
            siz[t] += siz[to];
        }
    }; dfs1(root);
    int dfn_tail = 0;
    for(int i = 1; i <= n; i++)top[i] = i;
    function<void(int)> dfs2 = [&](int t){
        dfn[t] = ++dfn_tail;
        rnk[dfn_tail] = t;
        if(!son[t])return;
        top[son[t]] = top[t];
        dfs2(son[t]);
        for(auto& [to, w] : graph[t]){
            if(to == fa[t] || to == son[t])continue;
            dfs2(to);
        }
    }; dfs2(root);
}
```

强连通分量

```
void tarjan(Graph& g1, Graph& g2){
    int dfn_tail = 0, cnt = 0;
    vector<int> dfn(g1.n + 1, 0), low(g1.n + 1, 0), exist(g1.n + 1, 0),
belong(g1.n + 1, 0);
    stack<int> sta;
    function<void(int)> dfs = [&](int t){
        dfn[t] = low[t] = ++dfn_tail;
        sta.push(t); exist[t] = 1;
        for(auto& [to] : g1.graph[t]){
            if(!dfn[to]){
                dfs(to);
                low[t] = min(low[t], low[to]);
            }
            else if(exist[to])low[t] = min(low[t], dfn[to]);
        }
        if(dfn[t] == low[t]){
            cnt++;
            while(int temp = sta.top()){
```

```
                belong[temp] = cnt;
                exist[temp] = 0;
                sta.pop();
                if(temp == t)break;
            }
        }
    };
    for(int i = 1; i <= g1.n; i++)if(!dfn[i])dfs(i);
    g2 = Graph(cnt);
    for(int i = 1; i <= g1.n; i++)g2.w[belong[i]] += g1.w[i];
    for(int i = 1; i <= g1.n; i++)
        for(auto& [to] : g1.graph[i])
            if(belong[i] != belong[to])g2.add(belong[i], belong[to]);
}
```

拓扑排序

```
void toposort(Graph& g, vector<int>& dis){
    vector<int> in(g.n + 1, 0);
    for(int i = 1; i <= g.n; i++)
        for(auto& [to] : g.graph[i])in[to]++;
    queue<int> que;
    for(int i = 1; i <= g.n; i++)
        if(!in[i]){
            que.push(i);
            dis[i] = g.w[i]; // dp
        }
    while(!que.empty()){
        int u = que.front(); que.pop();
        for(auto& [to] : g.graph[u]){
            in[to]--;
            dis[to] = max(dis[to], dis[u] + g.w[to]); // dp
            if(!in[to])que.push(to);
        }
    }
}
```

字符串

kmp

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

```
    return next;
}
```

哈希

```
constexpr int N = 2e6;
constexpr ll mod[2] = { 2000000011, 2000000033 }, base[2] = { 20011,20033 };
vector<array<ll, 2>> pow_base(N);

pow_base[0][0] = pow_base[0][1] = 1;
for(int i = 1; i < N; i++){
    pow_base[i][0] = pow_base[i - 1][0] * base[0] % mod[0];
    pow_base[i][1] = pow_base[i - 1][1] * base[1] % mod[1];
}

struct Hash{
    int size;
    vector<array<ll, 2>> hash;
    Hash(){}
    Hash(const string& s){
        size = s.size();
        hash.resize(size);
        hash[0][0] = hash[0][1] = s[0];
        for(int i = 1; i < size; i++){
            hash[i][0] = (hash[i - 1][0] * base[0] + s[i]) % mod[0];
            hash[i][1] = (hash[i - 1][1] * base[1] + s[i]) % mod[1];
        }
    }
    array<ll, 2> operator[](const array<int, 2>& range)const{
        int l = range[0], r = range[1];
        if(l == 0)return hash[r];
        auto single_hash = [&](bool flag){
            return (hash[r][flag] - hash[l - 1][flag] * pow_base[r - l + 1][flag]
% mod[flag] + mod[flag]) % mod[flag];
        };
        return { single_hash(0),single_hash(1) };
    }
};
```

manacher

```
void manacher(const string& _s, vector<int>& r){
    string s(_s.size() * 2 + 1, '$');
    for(int i = 0; i < _s.size(); i++)s[2 * i + 1] = _s[i];
    r.resize(_s.size() * 2 + 1);
    for(int i = 0, maxr = 0, mid = 0; i < s.size(); i++){
        if(i < maxr)r[i] = min(r[mid * 2 - i], maxr - i);
        while(i - r[i] - 1 >= 0 && i + r[i] + 1 <s.size() && s[i - r[i] - 1] ==
s[i + r[i] + 1]) ++r[i];
```

```
            if(i + r[i] > maxr) maxr = i + r[i], mid = i;
        }
    }
```

数学

线性筛法

```
auto [min_prime,primes] = [](){
    constexpr int N = 10000000;
    vector<int> min_prime(N + 1, 0), primes;
    for (int i = 2; i <= N; i++) {
        if (min_prime[i] == 0) {
            min_prime[i] = i;
            primes.push_back(i);
        }
        for (auto& prime : primes) {
            if (prime > min_prime[i] || prime > N / i)break;
            min_prime[prime * i] = prime;
        }
    }
    return tuple{ min_prime, primes };
}();
```

分解质因数

```
void num_primes(int num, vector<int>& ans) {
    for (auto& prime : primes) {
        if (prime > num / prime)break;
        if (num % prime == 0) {
            while (num % prime == 0)num /= prime;
            ans.push_back(prime);
        }
    }
    if (num > 1)ans.push_back(num);
}
```

组合数

```
modint C(int n, int m){
    if(m == 0)return 1;
    if(n <= mod)return factorial[n] * factorial[m].inv() * factorial[n - m].inv();
    else return C(n % mod, m % mod) * C(n / mod, m / mod); // n >= mod 时需要这个
}
```

盒子与球

$n个球,m个盒$

球同	盒同	可空	公式
✓	✓	✓	$f_{n,m}=f_{n-1,m-1}+f_{n-m,m}$
✓	✓	×	$f_{n-m,m}$
×	✓	✓	$\Sigma_{i=1}^{m}g_{n,i}$
×	✓	×	$g_{n,m}=g_{n-1,m-1}+m*g_{n-1,m}$
✓	×	✓	C_{n+m-1}^{m-1}
✓	×	×	C_{n-1}^{m-1}
×	×	✓	m^n
×	×	×	$m!*g_{n,m}$

线性基

```
// 线性基
struct basis{
    array<unsigned ll, 64> p{};

    // 将x插入此线性基中
    void insert(unsigned ll x){
        for(int i = 63; i >= 0; i--){
            if((x >> i) & 1){
                if(p[i])x ^= p[i];
                else{
                    p[i] = x; break;
                }
            }
        }
    }

    // 将另一个线性基插入此线性基中
    void insert(basis other){
        for(int i = 0; i <= 63; i++){
            if(!other.p[i])continue;
            insert(other.p[i]);
        }
    }

    // 最大异或值
    unsigned ll max_basis(){
        unsigned ll res = 0;
        for(int i = 63; i >= 0; i--){
            if((res ^ p[i]) > res)res ^= p[i];
        }
```

```
        return res;
    }
};
```

矩阵快速幂

```
constexpr ll mod = 2147493647;
struct Mat{
    int n, m;
    vector<vector<ll>> mat;
    Mat(int n, int m) :n(n), m(n), mat(n, vector<ll>(m, 0)){}
    Mat(vector<vector<ll>> mat) :n(mat.size()), m(mat[0].size()), mat(mat){}
    Mat operator*(const Mat& other){
        assert(m == other.n);
        Mat res(n, other.m);
        for(int i = 0; i < res.n; i++)
            for(int j = 0; j < res.m; j++)
                for(int k = 0; k < m; k++)
                    res.mat[i][j] = (res.mat[i][j] + mat[i][k] * other.mat[k][j] %
mod) % mod;
        return res;
    }
};
Mat ksm(Mat a, ll b){
    assert(a.n == a.m);
    Mat res(a.n, a.m);
    for(int i = 0; i < res.n; i++)res.mat[i][i] = 1;
    while(b){
        if(b & 1)res = res * a;
        b >>= 1;
        a = a * a;
    }
    return res;
}
```

计算几何

```
#define PI M_PI
constexpr double eps = 1e-8;
using T = int;

template<typename T>
bool equal(T a, T b) {
    return a == b;
}
// 两浮点数是否相等
bool equal(double a, double b) { return abs(a - b) < eps; }

// 向量
```

```
struct vec {
    T x, y;
    vec(T _x = 0, T _y = 0) : x(_x), y(_y) {}

    // 模
    double length() const { return sqrt(x * x + y * y); }

    // 与x轴正方向的夹角
    double angle() const {
        double angle = atan2(y, x);
        if (angle < 0) angle += 2 * PI;
        return angle;
    }

    // 逆时针旋转
    void rotate(const double &theta) {
        double temp = x;
        x = x * cos(theta) - y * sin(theta);
        y = y * cos(theta) + temp * sin(theta);
    }

    bool operator==(const vec &other) const { return equal(x, other.x) && equal(y,
other.y); }
    bool operator<(const vec &other) const {
        return equal(angle(), other.angle()) ? x < other.x : angle() <
other.angle();
    }

    vec operator+(const vec &other) const { return {x + other.x, y + other.y}; }
    vec operator-() const { return {-x, -y}; }
    vec operator-(const vec &other) const { return -other + (*this); }
    vec operator*(const T &other) const { return {x * other, y * other}; }
    vec operator/(const T &other) const { return {x / other, y / other}; }
    T operator*(const vec &other) const { return x * other.x + y * other.y; }

    // 叉积 结果大于0，a在b的顺时针，小于0，a在b的逆时针，
    // 等于0共线，可能同向或反向，结果绝对值表示 a b形成的平行四边行的面积
    T operator^(const vec &other) const { return x * other.y - y * other.x; }

    friend istream &operator>>(istream &input, vec &data) {
        input >> data.x >> data.y;
        return input;
    }
    friend ostream &operator<<(ostream &output, const vec &data) {
        output << fixed << setprecision(6);
        output << data.x << " " << data.y;
        return output;
    }
};

// 两点间的距离
T distance(const vec &a, const vec &b) { return sqrt((a.x - b.x) * (a.x - b.x) +
(a.y - b.y) * (a.y - b.y)); }
```

```cpp
// 两向量夹角
double angle(const vec &a, const vec &b) {
    double theta = abs(a.angle() - b.angle());
    if (theta > PI) theta = 2 * PI - theta;
    return theta;
}

// 多边形的面积
double polygon_area(vector<vec> &p) {
    T area = 0;
    for (int i = 1; i < p.size(); i++) area += p[i - 1] ^ p[i];
    area += p.back() ^ p[0];
    return abs(area / 2.0);
}

// 多边形的周长
double polygon_length(vector<vec> &p) {
    double length = 0;
    for (int i = 1; i < p.size(); i++) length += (p[i - 1] - p[i]).length();
    length += (p.back() - p[0]).length();
    return length;
}

// 多边形直径的两个端点
auto polygon_dia(vector<vec> &p) {
    int n = p.size();
    array<vec, 2> res{};
    if (n <= 1) return res;
    if (n == 2) return res = {p[0], p[1]};
    T mx = 0;
    for (int i = 0, j = 3; i < n; i++) {
        while (abs((p[i] - p[j]) ^ (p[(i + 1) % n] - p[j])) <=
               abs((p[i] - p[(j + 1) % n]) ^ (p[(i + 1) % n] - p[(j + 1) % n])))
            j = (j + 1) % n;
        if (auto tmp = distance(p[i], p[j]); tmp > mx) {
            mx = tmp;
            res = {p[i], p[j]};
        }
        if (auto tmp = distance(p[(i + 1) % n], p[j]); tmp > mx) {
            mx = tmp;
            res = {p[(i + 1) % n], p[j]};
        }
    }
    return res;
}

// 凸包
auto convex_hull(vector<vec> &p) {
    sort(p.begin(), p.end(), [](vec &a, vec &b) { return equal(a.x, b.x) ? a.y <
b.y : a.x < b.x; });

    vector<int> sta(p.size() + 1, 0);
    vector<bool> v(p.size(), false);
    int tp = -1;
```

```
    sta[++tp] = 0;

    auto update_convex_hull = [&](int lim, int i) {
        while (tp > lim && ((p[sta[tp]] - p[sta[tp - 1]]) ^ (p[i] - p[sta[tp]]))
<= 0) sta[++tp] = i;
        v[i] = true;
    };

    for (int i = 1; i < p.size(); i++) update_convex_hull(0, i);

    int cnt = tp;
    for (int i = p.size() - 1; i >= 0; i--) {
        if (v[i]) continue;
        update_convex_hull(cnt, i);
    }

    vector<vec> res(tp);
    for (int i = 0; i < tp; i++) res[i] = p[sta[i]];
    return res;
}

// 以整点为顶点的线段上的整点个数
T count(const vec &a, const vec &b) {
    vec c = a - b;
    return gcd(abs(c.x), abs(c.y)) + 1;
}

// 以整点为顶点的多边形边上整点个数
T count(vector<vec> &p) {
    T cnt = 0;
    for (int i = 1; i < p.size(); i++) cnt += count(p[i - 1], p[i]);
    cnt += count(p.back(), p[0]);
    return cnt - p.size();
}

// 直线
struct line {
    vec point, direction;
    line(const vec &p, const vec &d) : point(p), direction(d) {}
};

// 点到直线距离
double distance(const vec &a, const line &b) {
    return abs((b.point - a) ^ (b.point + b.direction - a)) /
b.direction.length();
}

// 两直线是否垂直
bool perpendicular(const line &a, const line &b) { return equal(a.direction *
b.direction, 0); }

// 两直线是否平行
bool parallel(const line &a, const line &b) { return equal(a.direction ^
b.direction, 0); }
```

```
// 两直线交点
vec intersection(T A, T B, T C, T D, T E, T F) {
    return {(B * F - C * E) / (A * E - B * D), (C * D - A * F) / (A * E - B * D)};
}

// 两直线交点
vec intersection(const line &a, const line &b) {
    return intersection(a.direction.y, -a.direction.x, a.direction.x * a.point.y -
a.direction.y * a.point.x,
                        b.direction.y, -b.direction.x, b.direction.x * b.point.y -
b.direction.y * b.point.x);
}
```

杂项

高精度

```
struct bignum{
    string num;

    bignum() :num("0"){}
    bignum(const string& num) :num(num){ reverse(this->num.begin(), this-
>num.end()); }
    bignum(ll num) :num(to_string(num)){ reverse(this->num.begin(), this-
>num.end()); }

    bignum operator+(const bignum& other){
        bignum res;
        res.num.pop_back();
        res.num.reserve(max(num.size(), other.num.size()) + 1);
        for(int i = 0, j = 0, x; i < num.size() || i < other.num.size() || j; i++)
{
            x = j; j = 0;
            if(i < num.size())x += num[i] - '0';
            if(i < other.num.size())x += other.num[i] - '0';
            if(x >= 10)j = 1, x -= 10;
            res.num.push_back(x + '0');
        }
        res.num.capacity();
        return res;
    }

    bignum operator*(const bignum& other){
        vector<int> res(num.size() + other.num.size() - 1, 0);
        for(int i = 0; i < num.size(); i++){
            for(int j = 0; j < other.num.size(); j++){
                res[i + j] += (num[i] - '0') * (other.num[j] - '0');
            }
        }
        int g = 0;
```

```
        for(int i = 0; i < res.size(); i++){
            res[i] += g;
            g = res[i] / 10;
            res[i] %= 10;
        }
        while(g){
            res.push_back(g % 10);
            g /= 10;
        }
        int lim = res.size();
        while(lim > 1 && res[lim - 1] == 0)lim--;
        bignum res2;
        res2.num.resize(lim);
        for(int i = 0; i < lim; i++)res2.num[i] = res[i] + '0';
        return res2;
    }

    bool operator<(const bignum& other){
        if(num.size() == other.num.size())
            for(int i = num.size() - 1; i >= 0; i--)
                if(num[i] == other.num[i])continue;
                else return num[i] < other.num[i];
        return num.size() < other.num.size();
    }

    friend istream& operator>>(istream& in, bignum& a){
        in >> a.num;
        reverse(a.num.begin(), a.num.end());
        return in;
    }
    friend ostream& operator<<(ostream& out, bignum a){
        reverse(a.num.begin(), a.num.end());
        return out << a.num;
    }
};
```

扫描线

```
#define ls (pos << 1)
#define rs (ls | 1)
#define mid ((tree[pos].l + tree[pos].r) >> 1)
struct Rectangle{
    ll x_l, y_l, x_r, y_r;
};
ll area(vector<Rectangle>& rec){
    struct Line{
        ll x, y_up, y_down;
        int pd;
    };
    vector<Line> line(rec.size() * 2);
    vector<ll> y_set(rec.size() * 2);
```

```
    for(int i = 0; i < rec.size(); i++){
        y_set[i * 2] = rec[i].y_l;
        y_set[i * 2 + 1] = rec[i].y_r;
        line[i * 2] = { rec[i].x_l,rec[i].y_r,rec[i].y_l,1 };
        line[i * 2 + 1] = { rec[i].x_r,rec[i].y_r,rec[i].y_l,-1 };
    }
    sort(y_set.begin(), y_set.end());
    y_set.erase(unique(y_set.begin(), y_set.end()), y_set.end());
    sort(line.begin(), line.end(), [](Line a, Line b){return a.x < b.x; });
    struct Data{
        int l, r;
        ll len, cnt, raw_len;
    };
    vector<Data> tree(4 * y_set.size());
    function<void(int, int, int)> build = [&](int pos, int l, int r){
        tree[pos].l = l;
        tree[pos].r = r;
        if(l == r){
            tree[pos].raw_len = y_set[r + 1] - y_set[l];
            tree[pos].cnt = tree[pos].len = 0;
            return;
        }
        build(ls, l, mid);
        build(rs, mid + 1, r);
        tree[pos].raw_len = tree[ls].raw_len + tree[rs].raw_len;
    };
    function<void(int, int, int, int)> update = [&](int pos, int l, int r, int
num){
        if(l <= tree[pos].l && tree[pos].r <= r){
            tree[pos].cnt += num;
            tree[pos].len = tree[pos].cnt ? tree[pos].raw_len : tree[pos].l ==
tree[pos].r ? 0 : tree[ls].len + tree[rs].len;
            return;
        }
        if(l <= mid)update(ls, l, r, num);
        if(r > mid)update(rs, l, r, num);
        tree[pos].len = tree[pos].cnt ? tree[pos].raw_len : tree[ls].len +
tree[rs].len;
    };
    build(1, 0, y_set.size() - 2);
    auto find_pos = [&](ll num){
        return lower_bound(y_set.begin(), y_set.end(), num) - y_set.begin();
    };
    ll res = 0;
    for(int i = 0; i < line.size() - 1; i++){
        update(1, find_pos(line[i].y_down), find_pos(line[i].y_up) - 1,
line[i].pd);
        res += (line[i + 1].x - line[i].x) * tree[1].len;
    }
    return res;
}
```

模运算

```
class modint{
    ll num;
public:
    modint(ll num = 0) :num(num % mod){}
    modint pow(modint other){
        modint res(1), temp = *this;
        while(other.num){
            if(other.num & 1)res = res * temp;
            temp = temp * temp;
            other.num >>= 1;
        }
        return res;
    }
    modint inv(){ return this->pow(mod - 2); }
    modint operator+(modint other){ return modint(this->num + other.num); }
    modint operator-(){ return { -this->num }; }
    modint operator-(modint other){ return modint(-other + *this); }
    modint operator*(modint other){ return modint(this->num * other.num); }
    modint operator/(modint other){ return *this * other.inv(); }
    friend istream& operator>>(istream& is, modint& other){ is >> other.num;
other.num %= mod; return is; }
    friend ostream& operator<<(ostream& os, modint other){ other.num = (other.num
+ mod) % mod; return os << other.num; }
};
```

分数

```
struct frac{
    ll a, b;
    frac() :a(0), b(1){}
    frac(ll a, ll b = 1){
        if(a){
            ll temp = gcd(a, b);
            this->a = a / temp; this->b = b / temp;
        }
        else{
            this->a = 0; this->b = 1;
        }
    }
    frac operator+(const frac& other){
        return frac(a * other.b + other.a * b, b * other.b);
    }
    frac operator-()const{
        frac res = *this;
        res.a = -res.a;
        return res;
    }
    frac operator-(const frac& other){
```

```
        return -other + *this;
    }
    frac operator*(const frac& other){
        return frac(a * other.a, b * other.b);
    }
    frac operator/(const frac& other){
        assert(other.a);
        return *this * frac(other.b, other.a);
    }
    bool operator<(const frac& other){
        return (*this - other).a < 0;
    }
    bool operator<=(const frac& other){
        return (*this - other).a <= 0;
    }
    bool operator>(const frac& other){
        return (*this - other).a > 0;
    }
    bool operator>=(const frac& other){
        return (*this - other).a >= 0;
    }
    bool operator==(const frac& other){
        return a == other.a && b == other.b;
    }
    bool operator!=(const frac& other){
        return !(*this == other);
    }
};
```

表达式求值

```
// 格式化表达式
string format(const string& s1){
    stringstream ss(s1);
    string s2;
    char ch;
    while((ch = ss.get()) != EOF){
        if(ch == ' ')continue;
        if(isdigit(ch))s2 += ch;
        else{
            if(s2.back() != ' ')s2 += ' ';
            s2 += ch; s2 += ' ';
        }
    }
    return s2;
}

// 中缀表达式转后缀表达式
string convert(const string& s1){

    unordered_map<char, int> rank{ {'+',2},{'-',2},{'*',1},{'/',1},{'^',0} };
```

```
    stringstream ss(s1);
    string s2, temp;
    stack<char> op;
    while(ss >> temp){
        if(isdigit(temp[0]))s2 += temp + ' ';
        else if(temp[0] == '(')op.push('(');
        else if(temp[0] == ')'){
            while(op.top() != '('){
                s2 += op.top(); s2 += ' '; op.pop();
            }
            op.pop();
        }
        else{
            while(!op.empty() && op.top() != '(' && (temp[0] != '^' &&
rank[op.top()] <= rank[temp[0]] || rank[op.top()] < rank[temp[0]])){
                s2 += op.top(); s2 += ' '; op.pop();
            }
            op.push(temp[0]);
        }
    }
    while(!op.empty()){
        s2 += op.top(); s2 += ' '; op.pop();
    }
    return s2;
}

// 计算后缀表达式
int calc(const string& s){
    stack<int> num;
    stringstream ss(s);
    string temp;
    while(ss >> temp){
        if(isdigit(temp[0]))num.push(stoi(temp));
        else{
            int b = num.top(); num.pop();
            int a = num.top(); num.pop();
            if(temp[0] == '+')a += b;
            else if(temp[0] == '-')a -= b;
            else if(temp[0] == '*')a *= b;
            else if(temp[0] == '/')a /= b;
            else if(temp[0] == '^')a = ksm(a, b);
            num.push(a);
        }
    }
    return num.top();
}
```

对拍

linux/Mac

```
g++ a.cpp -o program/a -O2 -std=c++17
g++ b.cpp -o program/b -O2 -std=c++17
g++ suiji.cpp -o program/suiji -O2 -std=c++17

cnt=0

while true; do
    let cnt++
    echo TEST:$cnt

    ./program/suiji > in
    ./program/a < in > out.a
    ./program/b < in > out.b

    diff out.a out.b
    if [ $? -ne 0 ];then break;fi
done
```

windows

```
@echo off

g++ a.cpp -o program/a -O2 -std=c++17
g++ b.cpp -o program/b -O2 -std=c++17
g++ suiji.cpp -o program/suiji -O2 -std=c++17

set cnt=0

:again
    set /a cnt=cnt+1
    echo TEST:%cnt%
    .\program\suiji > in
    .\program\a < in > out.a
    .\program\b < in > out.b

    fc output.a output.b
if not errorlevel 1 goto again
```

开栈

任选一种

```
-Wl,--stack=0x10000000
-Wl,-stack_size -Wl,0x10000000
-Wl,-z,stack-size=0x10000000
```

日期

```
int month[] = { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
int pre[13];
vector<int> leap;
struct Date {
    int y, m, d;
    bool operator < (const Date& other) const {
        return array<int, 3>{y, m, d} < array<int, 3>{other.y, other.m, other.d};
    }
    Date(const string& s) {
        stringstream ss(s);
        char ch;
        ss >> y >> ch >> m >> ch >> d;
    }
    int dis()const {
        int yd = (y - 1) * 365 + (upper_bound(leap.begin(), leap.end(), y - 1) -
leap.begin());
        int md = pre[m - 1] + (m > 2 && (y % 4 == 0 && y % 100 || y % 400 == 0));
        return yd + md + d;
    }
    int dis(const Date& other)const {
        return other.dis() - dis();
    }
};
for (int i = 1; i <= 12; i++)pre[i] = pre[i - 1] + month[2];
for (int i = 1; i <= 1000000; i++)
    if (i % 4 == 0 && i % 100 || i % 400 == 0)leap.push_back(i);
```