

ACM 常用算法模板

therehello

2023 年 9 月 19 日

目录

1 数据结构	3
1.1 并查集	3
1.2 树状数组	3
1.2.1 一维	3
1.2.2 二维	3
1.2.3 三维	4
1.3 线段树	5
1.4 普通平衡树	6
1.4.1 树状数组实现	6
1.5 可持久化线段树	8
1.6 st 表	8
2 图论	10
2.1 最短路	10
2.1.1 dijkstra	10
2.2 树上问题	10
2.2.1 最近公公祖先	10
2.2.2 树链剖分	11
2.3 强连通分量	12
2.4 拓扑排序	13
3 字符串	14
3.1 kmp	14
3.2 哈希	14
3.3 manacher	15
4 数学	16
4.1 扩展欧几里得	16
4.2 线性筛法	16
4.3 分解质因数	17
4.4 pollard rho	17
4.5 组合数	18
4.6 数论分块	19
4.7 积性函数	19
4.7.1 定义	19
4.7.2 例子	19
4.8 狄利克雷卷积	20
4.8.1 性质	20
4.8.2 例子	20
4.9 欧拉函数	20
4.10 莫比乌斯反演	20
4.10.1 莫比乌斯函数性质	20
4.10.2 莫比乌斯变换/反演	21
4.11 杜教筛	21

4.11.1 示例	21
4.12 盒子与球	22
4.13 线性基	22
4.14 矩阵快速幂	23
5 计算几何	24
5.1 整数	24
5.2 浮点数	28
5.3 扫描线	33
6 杂项	35
6.1 高精度	35
6.2 模运算	36
6.3 分数	36
6.4 表达式求值	37
6.5 日期	39
6.6 对拍	39
6.7 编译常用选项	40
6.8 开栈	40
6.9 clang-format	40

1 数据结构

1.1 并查集

```
1 struct dsu {
2     int n;
3     vector<int> fa, sz;
4     dsu(int _n) : n(_n), fa(n + 1), sz(n + 1, 1) {
5         iota(fa.begin(), fa.end(), 0);
6     }
7     int find(int x) { return x == fa[x] ? x : fa[x] = find(fa[x]); }
8     int merge(int x, int y) {
9         int fax = find(x), fay = find(y);
10        if (fax == fay) return 0; // 一个集合
11        sz[fay] += sz[fax];
12        return fa[fax] = fay; // 合并到哪个集合了
13    }
14    int size(int x) { return sz[find(x)]; }
15};
```

1.2 树状数组

1.2.1 一维

```
1 template <class T>
2 struct fenwick {
3     int n;
4     vector<T> t;
5     fenwick(int _n) : n(_n), t(n + 1) {}
6     T query(int l, int r) {
7         auto query = [&](int pos) {
8             T res = 0;
9             while (pos) {
10                res += t[pos];
11                pos -= lowbit(pos);
12            }
13            return res;
14        };
15        return query(r) - query(l - 1);
16    }
17    void add(int pos, T num) {
18        while (pos <= n) {
19            t[pos] += num;
20            pos += lowbit(pos);
21        }
22    }
23};
```

1.2.2 二维

```

1 template <class T>
2 struct Fenwick_tree_2 {
3     Fenwick_tree_2(int n, int m) : n(n), m(m), tree(n + 1, vector<T>(m + 1)) {}
4     T query(int l1, int r1, int l2, int r2) {
5         auto query = [&](int l, int r) {
6             T res = 0;
7             for (int i = l; i; i -= lowbit(i))
8                 for (int j = r; j; j -= lowbit(j)) res += tree[i][j];
9             return res;
10        };
11        return query(l2, r2) - query(l2, r1 - 1) - query(l1 - 1, r2) +
12               query(l1 - 1, r1 - 1);
13    }
14    void update(int x, int y, T num) {
15        for (int i = x; i <= n; i += lowbit(i))
16            for (int j = y; j <= m; j += lowbit(j)) tree[i][j] += num;
17    }
18 private:
19     int n, m;
20     vector<vector<T>> tree;
21 };

```

1.2.3 三维

```

1 template <class T>
2 struct Fenwick_tree_3 {
3     Fenwick_tree_3(int n, int m, int k)
4         : n(n),
5           m(m),
6           k(k),
7           tree(n + 1, vector<vector<T>>(m + 1, vector<T>(k + 1))) {}
8     T query(int a, int b, int c, int d, int e, int f) {
9         auto query = [&](int x, int y, int z) {
10             T res = 0;
11             for (int i = x; i; i -= lowbit(i))
12                 for (int j = y; j; j -= lowbit(j))
13                     for (int p = z; p; p -= lowbit(p)) res += tree[i][j][p];
14             return res;
15        };
16        T res = query(d, e, f);
17        res -= query(a - 1, e, f) + query(d, b - 1, f) + query(d, e, c - 1);
18        res += query(a - 1, b - 1, f) + query(a - 1, e, c - 1) +
19               query(d, b - 1, c - 1);
20        res -= query(a - 1, b - 1, c - 1);
21        return res;
22    }
23    void update(int x, int y, int z, T num) {
24        for (int i = x; i <= n; i += lowbit(i))
25            for (int j = y; j <= m; j += lowbit(j))
26                for (int p = z; p <= k; p += lowbit(p)) tree[i][j][p] += num;

```

```

27     }
28 private:
29     int n, m, k;
30     vector<vector<vector<T>>> tree;
31 };

```

1.3 线段树

```

1 template <class Data, class Num>
2 struct Segment_Tree {
3     inline void update(int l, int r, Num x) { update(1, l, r, x); }
4     inline Data query(int l, int r) { return query(1, l, r); }
5     Segment_Tree(vector<Data>& a) {
6         n = a.size();
7         tree.assign(n * 4 + 1, {});
8         build(a, 1, 1, n);
9     }
10 private:
11     int n;
12     struct Tree {
13         int l, r;
14         Data data;
15     };
16     vector<Tree> tree;
17     inline void pushup(int pos) {
18         tree[pos].data = tree[pos << 1].data + tree[pos << 1 | 1].data;
19     }
20     inline void pushdown(int pos) {
21         tree[pos << 1].data = tree[pos << 1].data + tree[pos].data.lazytag;
22         tree[pos << 1 | 1].data =
23             tree[pos << 1 | 1].data + tree[pos].data.lazytag;
24         tree[pos].data.lazytag = Num::zero();
25     }
26     void build(vector<Data>& a, int pos, int l, int r) {
27         tree[pos].l = l;
28         tree[pos].r = r;
29         if (l == r) {
30             tree[pos].data = a[l - 1];
31             return;
32         }
33         int mid = (tree[pos].l + tree[pos].r) >> 1;
34         build(a, pos << 1, l, mid);
35         build(a, pos << 1 | 1, mid + 1, r);
36         pushup(pos);
37     }
38     void update(int pos, int& l, int& r, Num& x) {
39         if (l > tree[pos].r || r < tree[pos].l) return;
40         if (l <= tree[pos].l && tree[pos].r <= r) {
41             tree[pos].data = tree[pos].data + x;
42             return;
43         }

```

```

44     pushdown(pos);
45     update(pos << 1, 1, r, x);
46     update(pos << 1 | 1, 1, r, x);
47     pushup(pos);
48 }
49 Data query(int pos, int& l, int& r) {
50     if (l > tree[pos].r || r < tree[pos].l) return Data::zero();
51     if (l <= tree[pos].l && tree[pos].r <= r) return tree[pos].data;
52     pushdown(pos);
53     return query(pos << 1, l, r) + query(pos << 1 | 1, l, r);
54 }
55 };
56 struct Num {
57     ll add;
58     inline static Num zero() { return {0}; }
59     inline Num operator+(Num b) { return {add + b.add}; }
60 };
61 struct Data {
62     ll sum, len;
63     Num lazytag;
64     inline static Data zero() { return {0, 0, Num::zero()}; }
65     inline Data operator+(Num b) {
66         return {sum + len * b.add, len, lazytag + b};
67     }
68     inline Data operator+(Data b) {
69         return {sum + b.sum, len + b.len, Num::zero()};
70     }
71 };

```

1.4 普通平衡树

1.4.1 树状数组实现

需要预先处理出来所有可能的数。

```

1  template <typename T>
2  struct treap {
3      int n, size;
4      vector<int> t;
5      vector<T> t2, S;
6      treap(const vector<T>& b) {
7          S = b;
8          sort(S.begin(), S.end());
9          S.erase(unique(S.begin(), S.end()), S.end());
10         n = S.size();
11         size = 0;
12         t = vector<int>(n + 1);
13         t2 = vector<T>(n + 1);
14     }
15     int pos(T x) { return lower_bound(S.begin(), S.end(), x) - S.begin() + 1; }
16     int sum(int pos) {
17         int res = 0;

```



```
18     while (pos) {
19         res += t[pos];
20         pos -= lowbit(pos);
21     }
22     return res;
23 }
24
25 // 插入cnt个x
26 void insert(T x, int cnt) {
27     size += cnt;
28     for (int i = pos(x); i <= n; i += lowbit(i)) {
29         t[i] += cnt;
30         t2[i] += cnt * x;
31     }
32 }
33
34 // 删除cnt个x
35 void erase(T x, int cnt) { insert(x, -cnt); }
36
37 // x的排名
38 int rank(T x) { return sum(pos(x) - 1) + 1; }
39
40 // 统计出现次数
41 int count(T x) { return sum(pos(x)) - sum(pos(x) - 1); }
42
43 // 第k小
44 T kth(int k) {
45     int cnt = 0, x = 0;
46     for (int i = log2(n); i >= 0; i--) {
47         x += 1 << i;
48         if (x >= n || cnt + t[x] >= k) x -= 1 << i;
49         else cnt += t[x];
50     }
51     return S[x];
52 }
53
54 // 前k小的数之和
55 T pre_sum(int k) {
56     int cnt = 0, x = 0;
57     T res = 0;
58     for (int i = log2(n); i >= 0; i--) {
59         x += 1 << i;
60         if (x >= n || cnt + t[x] >= k) x -= 1 << i;
61         else {
62             cnt += t[x];
63             res += t2[x];
64         }
65     }
66     return res + (k - cnt) * S[x];
67 }
68
69 // 小于x, 最大的数
```

```

70 T prev(int x) { return kth(sum(pos(x) - 1)); }
71
72 // 大于x, 最小的数
73 T next(int x) { return kth(sum(pos(x)) + 1); }
74 };

```

1.5 可持久化线段树

```

1 constexpr int MAXN = 200000;
2 vector<int> root(MAXN << 5);
3 struct Persistent_seg {
4     int n;
5     struct Data {
6         int ls, rs;
7         int val;
8     };
9     vector<Data> tree;
10    Persistent_seg(int n, vector<int>& a) : n(n) { root[0] = build(1, n, a); }
11    int build(int l, int r, vector<int>& a) {
12        if (l == r) {
13            tree.push_back({0, 0, a[l]});
14            return tree.size() - 1;
15        }
16        int mid = l + r >> 1;
17        int ls = build(l, mid, a), rs = build(mid + 1, r, a);
18        tree.push_back({ls, rs, tree[ls].val + tree[rs].val});
19        return tree.size() - 1;
20    }
21    int update(int rt, const int& idx, const int& val, int l, int r) {
22        if (l == r) {
23            tree.push_back({0, 0, tree[rt].val + val});
24            return tree.size() - 1;
25        }
26        int mid = l + r >> 1, ls = tree[rt].ls, rs = tree[rt].rs;
27        if (idx <= mid) ls = update(ls, idx, val, l, mid);
28        else rs = update(rs, idx, val, mid + 1, r);
29        tree.push_back({ls, rs, tree[ls].val + tree[rs].val});
30        return tree.size() - 1;
31    }
32    int query(int rt1, int rt2, int k, int l, int r) {
33        if (l == r) return l;
34        int mid = l + r >> 1;
35        int lcnt = tree[tree[rt2].ls].val - tree[tree[rt1].ls].val;
36        if (k <= lcnt) return query(tree[rt1].ls, tree[rt2].ls, k, l, mid);
37        else return query(tree[rt1].rs, tree[rt2].rs, k - lcnt, mid + 1, r);
38    }
39 };

```

1.6 st 表

```
1 auto lg = []() {
2     array<int, 10000001> lg;
3     lg[1] = 0;
4     for (int i = 2; i <= 10000000; i++) lg[i] = lg[i >> 1] + 1;
5     return lg;
6 }();
7 template <typename T>
8 struct st {
9     int n;
10    vector<vector<T>>> a;
11    st(vector<T>& _a) : n(_a.size()) {
12        a.assign(lg[n] + 1, vector<int>(n));
13        for (int i = 0; i < n; i++) a[0][i] = _a[i];
14        for (int j = 1; j <= lg[n]; j++)
15            for (int i = 0; i + (1 << j) - 1 < n; i++)
16                a[j][i] = max(a[j - 1][i], a[j - 1][i + (1 << (j - 1))]);
17    }
18    T query(int l, int r) {
19        int k = lg[r - l + 1];
20        return max(a[k][l], a[k][r - (1 << k) + 1]);
21    }
22 };
```

2 图论

存图

```

1 struct Graph {
2     int n;
3     struct Edge {
4         int to, w;
5     };
6     vector<vector<Edge>> graph;
7     Graph(int _n) {
8         n = _n;
9         graph.assign(n + 1, vector<Edge>());
10    };
11    void add(int u, int v, int w) { graph[u].push_back({v, w}); }
12 };

```

2.1 最短路

2.1.1 dijkstra

```

1 void dij(Graph& graph, vector<int>& dis, int t) {
2     vector<int> visit(graph.n + 1, 0);
3     priority_queue<pair<int, int>> que;
4     dis[t] = 0;
5     que.emplace(0, t);
6     while (!que.empty()) {
7         int u = que.top().second;
8         que.pop();
9         if (visit[u]) continue;
10        visit[u] = 1;
11        for (auto& [to, w] : graph.graph[u]) {
12            if (dis[to] > dis[u] + w) {
13                dis[to] = dis[u] + w;
14                que.emplace(-dis[to], to);
15            }
16        }
17    }
18 }

```

2.2 树上问题

2.2.1 最近公公祖先

倍增法

```

1 vector<int> dep;
2 vector<array<int, 21>> fa;
3 dep.assign(n + 1, 0);
4 fa.assign(n + 1, array<int, 21>{});
5 void binary_jump(int root) {
6     function<void(int)> dfs = [&](int t) {

```

```

7     dep[t] = dep[fa[t][0]] + 1;
8     for (auto& [to] : graph[t]) {
9         if (to == fa[t][0]) continue;
10        fa[to][0] = t;
11        dfs(to);
12    }
13 };
14 dfs(root);
15 for (int j = 1; j <= 20; j++)
16     for (int i = 1; i <= n; i++) fa[i][j] = fa[fa[i][j - 1]][j - 1];
17 }
18 int lca(int x, int y) {
19     if (dep[x] < dep[y]) swap(x, y);
20     for (int i = 20; i >= 0; i--)
21         if (dep[fa[x][i]] >= dep[y]) x = fa[x][i];
22     if (x == y) return x;
23     for (int i = 20; i >= 0; i--) {
24         if (fa[x][i] != fa[y][i]) {
25             x = fa[x][i];
26             y = fa[y][i];
27         }
28     }
29     return fa[x][0];
30 }

```

树剖

```

1 int lca(int x, int y) {
2     while (top[x] != top[y]) {
3         if (dep[top[x]] < dep[top[y]]) swap(x, y);
4         x = fa[top[x]];
5     }
6     if (dep[x] < dep[y]) swap(x, y);
7     return y;
8 }

```

2.2.2 树链剖分

```

1 vector<int> fa, siz, dep, son, dfn, rnk, top;
2 fa.assign(n + 1, 0);
3 siz.assign(n + 1, 0);
4 dep.assign(n + 1, 0);
5 son.assign(n + 1, 0);
6 dfn.assign(n + 1, 0);
7 rnk.assign(n + 1, 0);
8 top.assign(n + 1, 0);
9 void hld(int root) {
10     function<void(int)> dfs1 = [&](int t) {
11         dep[t] = dep[fa[t]] + 1;
12         siz[t] = 1;
13         for (auto& [to, w] : graph[t]) {
14             if (to == fa[t]) continue;

```

```

15         fa[to] = t;
16         dfs1(to);
17         if (siz[son[t]] < siz[to]) son[t] = to;
18         siz[t] += siz[to];
19     }
20 };
21 dfs1(root);
22 int dfn_tail = 0;
23 for (int i = 1; i <= n; i++) top[i] = i;
24 function<void(int)> dfs2 = [&](int t) {
25     dfn[t] = ++dfn_tail;
26     rnk[dfn_tail] = t;
27     if (!son[t]) return;
28     top[son[t]] = top[t];
29     dfs2(son[t]);
30     for (auto& [to, w] : graph[t]) {
31         if (to == fa[t] || to == son[t]) continue;
32         dfs2(to);
33     }
34 };
35 dfs2(root);
36 }

```

2.3 强连通分量

```

1 void tarjan(Graph& g1, Graph& g2) {
2     int dfn_tail = 0, cnt = 0;
3     vector<int> dfn(g1.n + 1, 0), low(g1.n + 1, 0), exist(g1.n + 1, 0),
4         belong(g1.n + 1, 0);
5     stack<int> sta;
6     function<void(int)> dfs = [&](int t) {
7         dfn[t] = low[t] = ++dfn_tail;
8         sta.push(t);
9         exist[t] = 1;
10        for (auto& [to] : g1.graph[t])
11            if (!dfn[to]) {
12                dfs(to);
13                low[t] = min(low[t], low[to]);
14            } else if (exist[to]) low[t] = min(low[t], dfn[to]);
15        if (dfn[t] == low[t]) {
16            cnt++;
17            while (int temp = sta.top()) {
18                belong[temp] = cnt;
19                exist[temp] = 0;
20                sta.pop();
21                if (temp == t) break;
22            }
23        }
24    };
25    for (int i = 1; i <= g1.n; i++)
26        if (!dfn[i]) dfs(i);

```

```
27 g2 = Graph(cnt);
28 for (int i = 1; i <= g1.n; i++) g2.w[belong[i]] += g1.w[i];
29 for (int i = 1; i <= g1.n; i++)
30     for (auto& [to] : g1.graph[i])
31         if (belong[i] != belong[to]) g2.add(belong[i], belong[to]);
32 }
```

2.4 拓扑排序

```
1 void toposort(Graph& g, vector<int>& dis) {
2     vector<int> in(g.n + 1, 0);
3     for (int i = 1; i <= g.n; i++)
4         for (auto& [to] : g.graph[i]) in[to]++;
5     queue<int> que;
6     for (int i = 1; i <= g.n; i++)
7         if (!in[i]) {
8             que.push(i);
9             dis[i] = g.w[i]; // dp
10        }
11    while (!que.empty()) {
12        int u = que.front();
13        que.pop();
14        for (auto& [to] : g.graph[u]) {
15            in[to]--;
16            dis[to] = max(dis[to], dis[u] + g.w[to]); // dp
17            if (!in[to]) que.push(to);
18        }
19    }
20 }
```

3 字符串

3.1 kmp

```

1 auto kmp(string& s) {
2     vector next(s.size(), -1);
3     for (int i = 1, j = -1; i < s.size(); i++) {
4         while (j >= 0 && s[i] != s[j + 1]) j = next[j];
5         if (s[i] == s[j + 1]) j++;
6         next[i] = j;
7     }
8     // next 意为长度
9     for (auto& i : next) i++;
10    return next;
11 }

```

3.2 哈希

```

1 constexpr int N = 2e6;
2 constexpr ll mod[2] = {2000000011, 2000000033}, base[2] = {20011, 20033};
3 vector<array<ll, 2>> pow_base(N);
4
5 pow_base[0][0] = pow_base[0][1] = 1;
6 for (int i = 1; i < N; i++) {
7     pow_base[i][0] = pow_base[i - 1][0] * base[0] % mod[0];
8     pow_base[i][1] = pow_base[i - 1][1] * base[1] % mod[1];
9 }
10
11 struct Hash {
12     int size;
13     vector<array<ll, 2>> hash;
14     Hash() {}
15     Hash(const string& s) {
16         size = s.size();
17         hash.resize(size);
18         hash[0][0] = hash[0][1] = s[0];
19         for (int i = 1; i < size; i++) {
20             hash[i][0] = (hash[i - 1][0] * base[0] + s[i]) % mod[0];
21             hash[i][1] = (hash[i - 1][1] * base[1] + s[i]) % mod[1];
22         }
23     }
24     array<ll, 2> operator[] (const array<int, 2>& range) const {
25         int l = range[0], r = range[1];
26         if (l == 0) return hash[r];
27         auto single_hash = [&](bool flag) {
28             return (hash[r][flag] -
29                     hash[l - 1][flag] * pow_base[r - l + 1][flag] % mod[flag] +
30                     mod[flag]) %
31                     mod[flag];
32         };
33         return {single_hash(0), single_hash(1)};

```



```
34     }  
35 };
```

3.3 manacher

```
1 void manacher(const string& _s, vector<int>& r) {  
2     string s(_s.size() * 2 + 1, '$');  
3     for (int i = 0; i < _s.size(); i++) s[2 * i + 1] = _s[i];  
4     r.resize(_s.size() * 2 + 1);  
5     for (int i = 0, maxr = 0, mid = 0; i < s.size(); i++) {  
6         if (i < maxr) r[i] = min(r[mid * 2 - i], maxr - i);  
7         while (i - r[i] - 1 >= 0 && i + r[i] + 1 < s.size() &&  
8             s[i - r[i] - 1] == s[i + r[i] + 1])  
9             ++r[i];  
10        if (i + r[i] > maxr) maxr = i + r[i], mid = i;  
11    }  
12 }
```

4 数学

4.1 扩展欧几里得

需保证 $a, b \geq 0$

$$x = x + k * dx, y = y - k * dy$$

若要求 $x \geq p$, $k \geq \lceil \frac{p-x}{dx} \rceil$

若要求 $x \leq q$, $k \leq \lfloor \frac{q-x}{dx} \rfloor$

若要求 $y \geq p$, $k \leq \lfloor \frac{y-p}{dy} \rfloor$

若要求 $y \leq q$, $k \geq \lceil \frac{y-q}{dy} \rceil$

```

1 int __exgcd(int a, int b, int& x, int& y) {
2     if (!b) {
3         x = 1;
4         y = 0;
5         return a;
6     }
7     int g = __exgcd(b, a % b, y, x);
8     y -= a / b * x;
9     return g;
10 }
11
12 array<int, 2> exgcd(int a, int b, int c) {
13     int x, y;
14     int g = __exgcd(a, b, x, y);
15     if (c % g) return {INT_MAX, INT_MAX};
16     int dx = b / g;
17     int dy = a / g;
18     x = c / g % dx * x % dx;
19     if (x < 0) x += dx;
20     y = (c - a * x) / b;
21     return {x, y};
22 }

```

4.2 线性筛法

```

1 constexpr int N = 10000000;
2 array<int, N + 1> min_prime;
3 vector<int> primes;
4 bool ok = []() {
5     for (int i = 2; i <= N; i++) {
6         if (min_prime[i] == 0) {
7             min_prime[i] = i;
8             primes.push_back(i);
9         }
10        for (auto& j : primes) {
11            if (j > min_prime[i] || j > N / i) break;
12            min_prime[j * i] = j;
13        }
14    }
15    return 1;

```

```
16 }();
```

4.3 分解质因数

```
1 auto getprimes(int n) {
2     vector<array<int, 2>> res;
3     for (auto& i : primes) {
4         if (i > n / i) break;
5         if (n % i == 0) {
6             res.push_back({i, 0});
7             while (n % i == 0) {
8                 n /= i;
9                 res.back()[1]++;
10            }
11        }
12    }
13    if (n > 1) res.push_back({n, 1});
14    return res;
15 }
```

4.4 pollard rho

```
1 using LL = __int128_t;
2
3 random_device rd;
4 mt19937 seed(rd());
5
6 ll power(ll a, ll b, ll mod) {
7     ll res = 1;
8     while (b) {
9         if (b & 1) res = (LL)res * a % mod;
10        a = (LL)a * a % mod;
11        b >>= 1;
12    }
13    return res;
14 }
15
16 bool isprime(ll n) {
17     static array primes{2, 3, 5, 7, 11, 13, 17, 19, 23};
18     static unordered_map<ll, bool> S;
19     if (n < 2) return 0;
20     if (S.count(n)) return S[n];
21     ll d = n - 1, r = 0;
22     while (!(d & 1)) {
23         r++;
24         d >>= 1;
25     }
26     for (auto& a : primes) {
27         if (a == n) return S[n] = 1;
28         ll x = power(a, d, n);
```

```

29     if (x == 1 || x == n - 1) continue;
30     for (int i = 0; i < r - 1; i++) {
31         x = (LL)x * x % n;
32         if (x == n - 1) break;
33     }
34     if (x != n - 1) return S[n] = 0;
35 }
36 return S[n] = 1;
37 }
38
39 ll pollard_rho(ll n) {
40     ll s = 0, t = 0;
41     ll c = seed() % (n - 1) + 1;
42     ll val = 1;
43     for (int goal = 1;; goal *= 2, s = t, val = 1) {
44         for (int step = 1; step <= goal; step++) {
45             t = ((LL)t * t + c) % n;
46             val = (LL)val * abs(t - s) % n;
47             if (step % 127 == 0) {
48                 ll g = gcd(val, n);
49                 if (g > 1) return g;
50             }
51         }
52         ll g = gcd(val, n);
53         if (g > 1) return g;
54     }
55 }
56 auto getprimes(ll n) {
57     unordered_set<ll> S;
58     auto get = [&](auto self, ll n) {
59         if (n < 2) return;
60         if (isprime(n)) {
61             S.insert(n);
62             return;
63         }
64         ll mx = pollard_rho(n);
65         self(self, n / mx);
66         self(self, mx);
67     };
68     get(get, n);
69     return S;
70 }

```

4.5 组合数

```

1 constexpr int N = 1e7;
2 array<modint, N + 1> fac, ifac;
3 auto _ = []() {
4     fac[0] = 1;
5     for (int i = 1; i <= N; i++) fac[i] = fac[i - 1] * i;
6     ifac[N] = fac[N].inv();

```

```

7   for (int i = N - 1; i >= 0; i--) ifac[i] = ifac[i + 1] * (i + 1);
8   return true;
9 }();
10
11 modint C(int n, int m) {
12     if (n < m) return 0;
13     if (n <= mod) return fac[n] * ifac[m] * ifac[n - m];
14     // n >= mod 时需要这个
15     return C(n % mod, m % mod) * C(n / mod, m / mod);
16 }

```

4.6 数论分块

求解形如 $\sum_{i=1}^n f(i)g(\lfloor \frac{n}{i} \rfloor)$ 的合式

$$s(n) = \sum_{i=1}^n f(i)$$

```

1 modint sqrt_decomposition(int n) {
2     auto s = [&](int x) { return x; };
3     auto g = [&](int x) { return x; };
4     modint res = 0;
5     while (l <= R) {
6         int r = n / (n / l);
7         res = res + (s(r) - s(l - 1)) * g(n / l);
8         l = r + 1;
9     }
10    return res;
11 }

```

4.7 积性函数

4.7.1 定义

函数 $f(n)$ 满足 $f(1) = 1$ 且 $\forall x, y \in \mathbf{N}^*, \gcd(x, y) = 1$ 都有 $f(xy) = f(x)f(y)$, 则 $f(n)$ 为积性函数。

函数 $f(n)$ 满足 $f(1) = 1$ 且 $\forall x, y \in \mathbf{N}^*$ 都有 $f(xy) = f(x)f(y)$, 则 $f(n)$ 为完全积性函数。

4.7.2 例子

- 单位函数: $\varepsilon(n) = [n = 1]$ 。(完全积性)
- 恒等函数: $\text{id}_k(n) = n^k$ 。(完全积性)
- 常数函数: $1(n) = 1$ 。(完全积性)
- 除数函数: $\sigma_k(n) = \sum_{d|n} d^k$ 。 $\sigma_0(n)$ 通常简记作 $d(n)$ 或 $\tau(n)$, $\sigma_1(n)$ 通常简记作 $\sigma(n)$ 。
- 欧拉函数: $\varphi(n) = \sum_{i=1}^n [\gcd(i, n) = 1]$ 。
- 莫比乌斯函数: $\mu(n) = \begin{cases} 1 & n = 1 \\ 0 & \exists d > 1, d^2 | n, \text{ 其中 } \omega(n) \text{ 表示 } n \text{ 的本质不同质因子个数, 它是} \\ (-1)^{\omega(n)} & \text{otherwise} \end{cases}$
一个加性函数。

4.8 狄利克雷卷积

对于两个数论函数 $f(x)$ 和 $g(x)$ ，则它们的狄利克雷卷积得到的结果 $h(x)$ 定义为：

$$h(x) = \sum_{d|x} f(d)g\left(\frac{x}{d}\right) = \sum_{ab=x} f(a)g(b)$$

可以简记为： $h = f * g$ 。

4.8.1 性质

交换律： $f * g = g * f$ 。

结合律： $(f * g) * h = f * (g * h)$ 。

分配律： $(f + g) * h = f * h + g * h$ 。

等式的性质： $f = g$ 的充要条件是 $f * h = g * h$ ，其中数论函数 $h(x)$ 要满足 $h(1) \neq 0$ 。

4.8.2 例子

- $\varepsilon = \mu * 1 \iff \varepsilon(n) = \sum_{d|n} \mu(d)$
- $id = \varphi * 1 \iff id(n) = \sum_{d|n} \varphi(d)$
- $d = 1 * 1 \iff d(n) = \sum_{d|n} 1$
- $\sigma = id * 1 \iff \sigma(n) = \sum_{d|n} d$
- $\varphi = \mu * id \iff \varphi(n) = \sum_{d|n} d \cdot \mu\left(\frac{n}{d}\right)$

4.9 欧拉函数

```
1 array<int, N + 1> phi;
2 auto _ = []() {
3     iota(phi.begin() + 1, phi.end(), 1);
4     for (int i = 2; i <= N; i++) {
5         if (phi[i] == i)
6             for (int j = i; j <= N; j += i) phi[j] = phi[j] / i * (i - 1);
7     }
8     return true;
9 }();
```

4.10 莫比乌斯反演

4.10.1 莫比乌斯函数性质

- $\sum_{d|n} \mu(d) = \begin{cases} 1 & n = 1 \\ 0 & n \neq 1 \end{cases}$ ，即 $\sum_{d|n} \mu(d) = \varepsilon(n)$ ， $\mu * 1 = \varepsilon$
- $[\gcd(i, j) = 1] = \sum_{d|\gcd(i, j)} \mu(d)$

```
1 array<int, N + 1> miu;
2 array<bool, N + 1> ispr;
3 auto _ = []() {
4     miu.fill(1);
5     ispr.fill(1);
```

```

6   for (int i = 2; i <= N; i++) {
7       if (!ispr[i]) continue;
8       miu[i] = -1;
9       for (int j = 2 * i; j <= N; j += i) {
10          ispr[j] = 0;
11          if ((j / i) % i == 0) miu[j] = 0;
12          else miu[j] *= -1;
13      }
14  }
15  return true;
16 }();

```

4.10.2 莫比乌斯变换/反演

$f(n) = \sum_{d|n} g(d)$, 那么有 $g(n) = \sum_{d|n} \mu(d) f(\frac{n}{d}) = \sum_{n|d} \mu(\frac{d}{n}) f(d)$ 。

用狄利克雷卷积表示则为 $f = g * 1$, 有 $g = f * \mu$ 。

$f \rightarrow g$ 称为莫比乌斯反演, $g \rightarrow f$ 称为莫比乌斯反演。

4.11 杜教筛

杜教筛被用于处理一类数论函数的前缀和问题。对于数论函数 f , 杜教筛可以在低于线性时间的复杂度内计算 $S(n) = \sum_{i=1}^n f(i)$ 。

$$S(n) = \frac{\sum_{i=1}^n (f * g)(i) - \sum_{i=2}^n g(i) S(\lfloor \frac{n}{i} \rfloor)}{g(1)}$$

可以构造恰当的数论函数 g 使得:

- 可以快速计算 $\sum_{i=1}^n (f * g)(i)$ 。
- 可以快速计算 g 的单点值, 用数论分块求解 $\sum_{i=2}^n g(i) S(\lfloor \frac{n}{i} \rfloor)$ 。

4.11.1 示例

```

1 ll sum_phi(ll n) {
2     if (n <= N) return sp[n];
3     if (sp2.count(n)) return sp2[n];
4     ll res = 0, l = 2;
5     while (l <= n) {
6         ll r = n / (n / l);
7         res = res + (r - l + 1) * sum_phi(n / l);
8         l = r + 1;
9     }
10    return sp2[n] = (ll)n * (n + 1) / 2 - res;
11 }
12 ll sum_miu(ll n) {
13     if (n <= N) return sm[n];
14     if (sm2.count(n)) return sm2[n];
15     ll res = 0, l = 2;
16     while (l <= n) {
17         ll r = n / (n / l);
18         res = res + (r - l + 1) * sum_miu(n / l);

```

```
19     l = r + 1;
20 }
21 return sm2[n] = 1 - res;
22 }
```

4.12 盒子与球

n 个球, m 个盒

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

扩展:

n 个相同的球, m 个不同的盒, 每个盒子超过 k 个球, 问方案数?

可以考虑容斥, $f(d)$ 表示至少有 d 个盒子装了 $> k$ 个球方案数, 总方案数则为 $f(0) - f(1) + f(2) - \dots$

4.13 线性基

```
1 // 线性基
2 struct basis {
3     int rnk = 0;
4     array<ull, 64> p{};
5
6     // 将x插入此线性基中
7     void insert(ull x) {
8         for (int i = 63; i >= 0; i--) {
9             if ((x >> i) & 1) {
10                 if (p[i] x ^= p[i];
11                 else {
12                     for (int j = 0; j < i; j++)
13                         if (x >> j & 1) x ^= p[j];
14                     for (int j = i + 1; j <= 63; j++)
15                         if (p[j] >> i & 1) p[j] ^= x;
16                     p[i] = x;
17                     rnk++;
18                     break;
19                 }
20             }
```



```

21     }
22 }
23
24 // 将另一个线性基插入此线性基中
25 void insert(basis other) {
26     for (int i = 0; i <= 63; i++) {
27         if (!other.p[i]) continue;
28         insert(other.p[i]);
29     }
30 }
31
32 // 最大异或值
33 ull max_basis() {
34     ull res = 0;
35     for (int i = 63; i >= 0; i--)
36         if ((res ^ p[i]) > res) res ^= p[i];
37     return res;
38 }
39 };

```

4.14 矩阵快速幂

```

1 constexpr ll mod = 2147493647;
2 struct Mat {
3     int n, m;
4     vector<vector<ll>> mat;
5     Mat(int n, int m) : n(n), m(m), mat(n, vector<ll>(m, 0)) {}
6     Mat(vector<vector<ll>> mat) : n(mat.size()), m(mat[0].size()), mat(mat) {}
7     Mat operator*(const Mat& other) {
8         assert(m == other.n);
9         Mat res(n, other.m);
10        for (int i = 0; i < res.n; i++)
11            for (int j = 0; j < res.m; j++)
12                for (int k = 0; k < m; k++)
13                    res.mat[i][j] =
14                        (res.mat[i][j] + mat[i][k] * other.mat[k][j] % mod) %
15                        mod;
16        return res;
17    }
18 };
19 Mat ksm(Mat a, ll b) {
20     assert(a.n == a.m);
21     Mat res(a.n, a.m);
22     for (int i = 0; i < res.n; i++) res.mat[i][i] = 1;
23     while (b) {
24         if (b & 1) res = res * a;
25         b >>= 1;
26         a = a * a;
27     }
28     return res;
29 }

```

5 计算几何

5.1 整数

```

1 constexpr double inf = 1e100;
2
3 // 向量
4 struct vec {
5     static bool cmp(const vec &a, const vec &b) {
6         return tie(a.x, a.y) < tie(b.x, b.y);
7     }
8
9     ll x, y;
10    vec() : x(0), y(0) {}
11    vec(ll _x, ll _y) : x(_x), y(_y) {}
12
13    // 模
14    ll len2() const { return x * x + y * y; }
15    double len() const { return sqrt(x * x + y * y); }
16
17    // 是否在上半轴
18    bool up() const { return y > 0 || y == 0 && x >= 0; }
19
20    bool operator==(const vec &b) const { return tie(x, y) == tie(b.x, b.y); }
21    // 极角排序
22    bool operator<(const vec &b) const {
23        if (up() != b.up()) return up() > b.up();
24        ll tmp = (*this) ^ b;
25        return tmp ? tmp > 0 : cmp(*this, b);
26    }
27
28    vec operator+(const vec &b) const { return {x + b.x, y + b.y}; }
29    vec operator-() const { return {-x, -y}; }
30    vec operator-(const vec &b) const { return -b + (*this); }
31    vec operator*(ll b) const { return {x * b, y * b}; }
32    ll operator*(const vec &b) const { return x * b.x + y * b.y; }
33
34    // 叉积 结果大于0, a到b为逆时针, 小于0, a到b顺时针,
35    // 等于0共线, 可能同向或反向, 结果绝对值表示 a b 形成的平行四边形的面积
36    ll operator^(const vec &b) const { return x * b.y - y * b.x; }
37
38    friend istream &operator>>(istream &in, vec &data) {
39        in >> data.x >> data.y;
40        return in;
41    }
42    friend ostream &operator<<(ostream &out, const vec &data) {
43        out << fixed << setprecision(6);
44        out << data.x << " " << data.y;
45        return out;
46    }
47 };
48

```

```

49 ll cross(const vec &a, const vec &b, const vec &c) { return (a - c) ^ (b - c); }
50
51 // 多边形的面积a
52 double polygon_area(vector<vec> &p) {
53     ll area = 0;
54     for (int i = 1; i < p.size(); i++) area += p[i - 1] ^ p[i];
55     area += p.back() ^ p[0];
56     return abs(area / 2.0);
57 }
58
59 // 多边形的周长
60 double polygon_len(vector<vec> &p) {
61     double len = 0;
62     for (int i = 1; i < p.size(); i++) len += (p[i - 1] - p[i]).len();
63     len += (p.back() - p[0]).len();
64     return len;
65 }
66
67 // 以整点为顶点的线段上的整点个数
68 ll count(const vec &a, const vec &b) {
69     vec c = a - b;
70     return gcd(abs(c.x), abs(c.y)) + 1;
71 }
72
73 // 以整点为顶点的多边形边上整点个数
74 ll count(vector<vec> &p) {
75     ll cnt = 0;
76     for (int i = 1; i < p.size(); i++) cnt += count(p[i - 1], p[i]);
77     cnt += count(p.back(), p[0]);
78     return cnt - p.size();
79 }
80
81 // 判断点是否在凸包内，凸包必须为逆时针顺序
82 bool in_polygon(const vec &a, vector<vec> &p) {
83     int n = p.size();
84     if (n == 0) return 0;
85     if (n == 1) return a == p[0];
86     if (n == 2)
87         return cross(a, p[1], p[0]) == 0 && (p[0] - a) * (p[1] - a) <= 0;
88     if (cross(a, p[1], p[0]) > 0 || cross(p.back(), a, p[0]) > 0) return 0;
89     auto cmp = [&](vec &x, const vec &y) { return ((x - p[0]) ^ y) >= 0; };
90     int i =
91         lower_bound(p.begin() + 2, p.end() - 1, a - p[0], cmp) - p.begin() - 1;
92     return cross(p[(i + 1) % n], a, p[i]) >= 0;
93 }
94
95 // 凸包直径的两个端点
96 auto polygon_dia(vector<vec> &p) {
97     int n = p.size();
98     array<vec, 2> res{};
99     if (n == 1) return res;
100     if (n == 2) return res = {p[0], p[1]};

```

```

101     ll mx = 0;
102     for (int i = 0, j = 2; i < n; i++) {
103         while (abs(cross(p[i], p[(i + 1) % n], p[j])) <=
104             abs(cross(p[i], p[(i + 1) % n], p[(j + 1) % n])))
105             j = (j + 1) % n;
106         ll tmp = (p[i] - p[j]).len2();
107         if (tmp > mx) {
108             mx = tmp;
109             res = {p[i], p[j]};
110         }
111         tmp = (p[(i + 1) % n] - p[j]).len2();
112         if (tmp > mx) {
113             mx = tmp;
114             res = {p[(i + 1) % n], p[j]};
115         }
116     }
117     return res;
118 }
119
120 // 凸包
121 auto convex_hull(vector<vec> &p) {
122     sort(p.begin(), p.end(), vec::cmp);
123     int n = p.size();
124     vector sta(n + 1, 0);
125     vector v(n, false);
126     int tp = -1;
127     sta[++tp] = 0;
128     auto update = [&](int lim, int i) {
129         while (tp > lim && cross(p[i], p[sta[tp]], p[sta[tp - 1]]) >= 0)
130             v[sta[tp--]] = 0;
131         sta[++tp] = i;
132         v[i] = 1;
133     };
134     for (int i = 1; i < n; i++) update(0, i);
135     int cnt = tp;
136     for (int i = n - 1; i >= 0; i--) {
137         if (v[i]) continue;
138         update(cnt, i);
139     }
140     vector<vec> res(tp);
141     for (int i = 0; i < tp; i++) res[i] = p[sta[i]];
142     return res;
143 }
144
145 // 闵可夫斯基和，两个点集的和构成一个凸包
146 auto minkowski(vector<vec> &a, vector<vec> &b) {
147     rotate(a.begin(), min_element(a.begin(), a.end(), vec::cmp), a.end());
148     rotate(b.begin(), min_element(b.begin(), b.end(), vec::cmp), b.end());
149     int n = a.size(), m = b.size();
150     vector<vec> c{a[0] + b[0]};
151     c.reserve(n + m);
152     int i = 0, j = 0;

```

```

153     while (i < n && j < m) {
154         vec x = a[(i + 1) % n] - a[i];
155         vec y = b[(j + 1) % m] - b[j];
156         c.push_back(c.back() + ((x ^ y) >= 0 ? (i++, x) : (j++, y)));
157     }
158     while (i + 1 < n) {
159         c.push_back(c.back() + a[(i + 1) % n] - a[i]);
160         i++;
161     }
162     while (j + 1 < m) {
163         c.push_back(c.back() + b[(j + 1) % m] - b[j]);
164         j++;
165     }
166     return c;
167 }
168
169 // 过凸多边形外一点求凸多边形的切线，返回切点下标
170 auto tangent(const vec &a, vector<vec> &p) {
171     int n = p.size();
172     int l = -1, r = -1;
173     for (int i = 0; i < n; i++) {
174         ll tmp1 = cross(p[i], p[(i - 1 + n) % n], a);
175         ll tmp2 = cross(p[i], p[(i + 1) % n], a);
176         if (l == -1 && tmp1 <= 0 && tmp2 <= 0) l = i;
177         else if (r == -1 && tmp1 >= 0 && tmp2 >= 0) r = i;
178     }
179     return array{l, r};
180 }
181
182 // 直线
183 struct line {
184     vec p, d;
185     line() : p(vec()), d(vec()) {}
186     line(const vec &p, const vec &d) : p(p), d(d) {}
187 };
188
189 // 点到直线距离
190 double dis(const vec &a, const line &b) {
191     return abs((b.p - a) ^ (b.p + b.d - a)) / b.d.len();
192 }
193
194 // 点在直线哪边，大于0在左边，等于0在线上，小于0在右边
195 ll side_line(const vec &a, const line &b) { return b.d ^ (a - b.p); }
196
197 // 两直线是否垂直
198 bool perpen(const line &a, const line &b) { return a.d * b.d == 0; }
199
200 // 两直线是否平行
201 bool parallel(const line &a, const line &b) { return (a.d ^ b.d) == 0; }
202
203 // 点的垂线是否与线段有交点
204 bool perpen(const vec &a, const line &b) {

```

```

205     vec p(-b.d.y, b.d.x);
206     bool cross1 = (p ^ (b.p - a)) > 0;
207     bool cross2 = (p ^ (b.p + b.d - a)) > 0;
208     return cross1 != cross2;
209 }
210
211 // 点到线段距离
212 double dis_seg(const vec &a, const line &b) {
213     if (perpen(a, b)) return dis(a, b);
214     return min((b.p - a).len(), (b.p + b.d - a).len());
215 }
216
217 // 点到凸包距离
218 double dis(const vec &a, vector<vec> &p) {
219     double res = inf;
220     for (int i = 1; i < p.size(); i++)
221         res = min(dis_seg(a, line(p[i - 1], p[i] - p[i - 1])), res);
222     res = min(dis_seg(a, line(p.back(), p[0] - p.back())), res);
223     return res;
224 }
225
226 // 两直线交点
227 vec intersection(ll A, ll B, ll C, ll D, ll E, ll F) {
228     return {(B * F - C * E) / (A * E - B * D),
229             (C * D - A * F) / (A * E - B * D)};
230 }
231
232 // 两直线交点
233 vec intersection(const line &a, const line &b) {
234     return intersection(a.d.y, -a.d.x, a.d.x * a.p.y - a.d.y * a.p.x, b.d.y,
235                        -b.d.x, b.d.x * b.p.y - b.d.y * b.p.x);
236 }

```

5.2 浮点数

```

1 constexpr lf eps = 1e-6;
2 constexpr lf inf = 1e100;
3 const lf PI = acos(-1);
4
5 int sgn(lf a, lf b) {
6     lf c = a - b;
7     return c < -eps ? -1 : c < eps ? 0 : 1;
8 }
9
10 // 向量
11 struct vec {
12     static bool cmp(const vec &a, const vec &b) {
13         return sgn(a.x, b.x) ? a.x < b.x : sgn(a.y, b.y) < 0;
14     }
15
16     lf x, y;

```

```

17  vec() : x(0), y(0) {}
18  vec(1f _x, 1f _y) : x(_x), y(_y) {}
19
20  // 模
21  1f len2() const { return x * x + y * y; }
22  1f len() const { return sqrt(x * x + y * y); }
23
24  // 与x轴正方向的夹角
25  1f angle() const {
26      1f angle = atan2(y, x);
27      if (angle < 0) angle += 2 * PI;
28      return angle;
29  }
30
31  // 逆时针旋转
32  vec rotate(const 1f &theta) const {
33      return {x * cos(theta) - y * sin(theta),
34              y * cos(theta) + x * sin(theta)};
35  }
36
37  vec e() const {
38      1f tmp = len();
39      return {x / tmp, y / tmp};
40  }
41
42  // 是否在上半轴
43  bool up() const {
44      return sgn(y, 0) > 0 || sgn(y, 0) == 0 && sgn(x, 0) >= 0;
45  }
46
47  bool operator==(const vec &other) const {
48      return sgn(x, other.x) == 0 && sgn(y, other.y) == 0;
49  }
50  // 极角排序
51  bool operator<(const vec &b) const {
52      if (up() != b.up()) return up() > b.up();
53      1f tmp = (*this) ^ b;
54      return sgn(tmp, 0) ? tmp > 0 : cmp(*this, b);
55  }
56
57  vec operator+(const vec &b) const { return {x + b.x, y + b.y}; }
58  vec operator-() const { return {-x, -y}; }
59  vec operator-(const vec &b) const { return -b + (*this); }
60  vec operator*(1f b) const { return {x * b, y * b}; }
61  vec operator/(1f b) const { return {x / b, y / b}; }
62  1f operator*(const vec &b) const { return x * b.x + y * b.y; }
63
64  // 叉积 结果大于0, a到b为逆时针, 小于0, a到b顺时针,
65  // 等于0共线, 可能同向或反向, 结果绝对值表示 a b 形成的平行四边形的面积
66  1f operator^(const vec &b) const { return x * b.y - y * b.x; }
67
68  friend istream &operator>>(istream &in, vec &data) {

```

```

69     in >> data.x >> data.y;
70     return in;
71 }
72 friend ostream &operator<<(ostream &out, const vec &data) {
73     out << fixed << setprecision(6);
74     out << data.x << " " << data.y;
75     return out;
76 }
77 };
78
79 lf cross(const vec &a, const vec &b, const vec &c) { return (a - c) ^ (b - c); }
80
81 lf angle(const vec &a, const vec &b) { return atan2(abs(a ^ b), a * b); }
82
83 // 多边形的面积
84 lf polygon_area(vector<vec> &p) {
85     lf area = 0;
86     for (int i = 1; i < p.size(); i++) area += p[i - 1] ^ p[i];
87     area += p.back() ^ p[0];
88     return abs(area / 2.0);
89 }
90
91 // 多边形的周长
92 lf polygon_len(vector<vec> &p) {
93     lf len = 0;
94     for (int i = 1; i < p.size(); i++) len += (p[i - 1] - p[i]).len();
95     len += (p.back() - p[0]).len();
96     return len;
97 }
98
99 // 判断点是否在凸包内，凸包必须为逆时针顺序
100 bool in_polygon(const vec &a, vector<vec> &p) {
101     int n = p.size();
102     if (n == 0) return 0;
103     if (n == 1) return a == p[0];
104     if (n == 2)
105         return sgn(cross(a, p[1], p[0]), 0) == 0 &&
106             sgn((p[0] - a) * (p[1] - a), 0) <= 0;
107     if (sgn(cross(a, p[1], p[0]), 0) > 0 ||
108         sgn(cross(p.back(), a, p[0]), 0) > 0)
109         return 0;
110     auto cmp = [&](vec &x, const vec &y) {
111         return sgn((x - p[0]) ^ y, 0) >= 0;
112     };
113     int i =
114         lower_bound(p.begin() + 2, p.end() - 1, a - p[0], cmp) - p.begin() - 1;
115     return sgn(cross(p[(i + 1) % n], a, p[i]), 0) >= 0;
116 }
117
118 // 凸包直径的两个端点
119 auto polygon_dia(vector<vec> &p) {
120     int n = p.size();

```



```

121     array<vec, 2> res{};
122     if (n == 1) return res;
123     if (n == 2) return res = {p[0], p[1]};
124     lf mx = 0;
125     for (int i = 0, j = 2; i < n; i++) {
126         while (sgn(abs(cross(p[i], p[(i + 1) % n], p[j])),
127                     abs(cross(p[i], p[(i + 1) % n], p[(j + 1) % n]))) <= 0)
128             j = (j + 1) % n;
129         lf tmp = (p[i] - p[j]).len();
130         if (tmp > mx) {
131             mx = tmp;
132             res = {p[i], p[j]};
133         }
134         tmp = (p[(i + 1) % n] - p[j]).len();
135         if (tmp > mx) {
136             mx = tmp;
137             res = {p[(i + 1) % n], p[j]};
138         }
139     }
140     return res;
141 }
142
143 // 凸包
144 auto convex_hull(vector<vec> &p) {
145     sort(p.begin(), p.end(), vec::cmp);
146     int n = p.size();
147     vector sta(n + 1, 0);
148     vector v(n, false);
149     int tp = -1;
150     sta[++tp] = 0;
151     auto update = [&](int lim, int i) {
152         while (tp > lim && sgn(cross(p[i], p[sta[tp]], p[sta[tp - 1]]), 0) >= 0)
153             v[sta[tp--]] = 0;
154         sta[++tp] = i;
155         v[i] = 1;
156     };
157     for (int i = 1; i < n; i++) update(0, i);
158     int cnt = tp;
159     for (int i = n - 1; i >= 0; i--) {
160         if (v[i]) continue;
161         update(cnt, i);
162     }
163     vector<vec> res(tp);
164     for (int i = 0; i < tp; i++) res[i] = p[sta[i]];
165     return res;
166 }
167
168 // 闵可夫斯基和，两个点集的和构成一个凸包
169 auto minkowski(vector<vec> &a, vector<vec> &b) {
170     rotate(a.begin(), min_element(a.begin(), a.end(), vec::cmp), a.end());
171     rotate(b.begin(), min_element(b.begin(), b.end(), vec::cmp), b.end());
172     int n = a.size(), m = b.size();

```

```

173     vector<vec> c{a[0] + b[0]};
174     c.reserve(n + m);
175     int i = 0, j = 0;
176     while (i < n && j < m) {
177         vec x = a[(i + 1) % n] - a[i];
178         vec y = b[(j + 1) % m] - b[j];
179         c.push_back(c.back() + (sgn(x ^ y, 0) >= 0 ? (i++, x) : (j++, y)));
180     }
181     while (i + 1 < n) {
182         c.push_back(c.back() + a[(i + 1) % n] - a[i]);
183         i++;
184     }
185     while (j + 1 < m) {
186         c.push_back(c.back() + b[(j + 1) % m] - b[j]);
187         j++;
188     }
189     return c;
190 }
191
192 // 过凸多边形外一点求凸多边形的切线, 返回切点下标
193 auto tangent(const vec &a, vector<vec> &p) {
194     int n = p.size();
195     int l = -1, r = -1;
196     for (int i = 0; i < n; i++) {
197         lf tmp1 = cross(p[i], p[(i - 1 + n) % n], a);
198         lf tmp2 = cross(p[i], p[(i + 1) % n], a);
199         if (l == -1 && sgn(tmp1, 0) <= 0 && sgn(tmp2, 0) <= 0) l = i;
200         else if (r == -1 && sgn(tmp1, 0) >= 0 && sgn(tmp2, 0) >= 0) r = i;
201     }
202     return array{l, r};
203 }
204
205 // 直线
206 struct line {
207     vec p, d;
208     line() : p(vec()), d(vec()) {}
209     line(const vec &_p, const vec &_d) : p(_p), d(_d) {}
210 };
211
212 // 点到直线距离
213 lf dis(const vec &a, const line &b) {
214     return abs((b.p - a) ^ (b.p + b.d - a)) / b.d.len();
215 }
216
217 // 点在直线哪边, 大于0在左边, 等于0在线上, 小于0在右边
218 int side_line(const vec &a, const line &b) { return sgn(b.d ^ (a - b.p), 0); }
219
220 // 两直线是否垂直
221 bool perpen(const line &a, const line &b) { return sgn(a.d * b.d, 0) == 0; }
222
223 // 两直线是否平行
224 bool parallel(const line &a, const line &b) { return sgn(a.d ^ b.d, 0) == 0; }

```

```

225
226 // 点的垂线是否与线段有交点
227 bool perpen(const vec &a, const line &b) {
228     vec p(-b.d.y, b.d.x);
229     bool cross1 = sgn(p ^ (b.p - a), 0) > 0;
230     bool cross2 = sgn(p ^ (b.p + b.d - a), 0) > 0;
231     return cross1 != cross2;
232 }
233
234 // 点到线段距离
235 lf dis_seg(const vec &a, const line &b) {
236     if (perpen(a, b)) return dis(a, b);
237     return min((b.p - a).len(), (b.p + b.d - a).len());
238 }
239
240 // 点到凸包距离
241 lf dis(const vec &a, vector<vec> &p) {
242     lf res = inf;
243     for (int i = 1; i < p.size(); i++)
244         res = min(dis_seg(a, line(p[i - 1], p[i] - p[i - 1])), res);
245     res = min(dis_seg(a, line(p.back(), p[0] - p.back())), res);
246     return res;
247 }
248
249 // 两直线交点
250 vec intersection(lf A, lf B, lf C, lf D, lf E, lf F) {
251     return {(B * F - C * E) / (A * E - B * D),
252             (C * D - A * F) / (A * E - B * D)};
253 }
254
255 // 两直线交点
256 vec intersection(const line &a, const line &b) {
257     return intersection(a.d.y, -a.d.x, a.d.x * a.p.y - a.d.y * a.p.x, b.d.y,
258                         -b.d.x, b.d.x * b.p.y - b.d.y * b.p.x);
259 }
260
261 struct circle {
262     vec o;
263     lf r;
264     circle(const vec &o, lf _r) : o(o), r(_r){};
265     // 点与圆的关系 -1在圆内, 0在圆上, 1在圆外
266     int relation(const vec &a) const {
267         lf len = (a - o).len();
268         return sgn(len, r);
269     }
270     lf area() { return PI * r * r; }
271 };
272
273 // 圆与直线交点
274 auto intersection(const circle &c, const line &l) {
275     lf d = dis(c.o, l);
276     vector<vec> res;

```

```

277     vec mid = l.p + l.d.e() * ((c.o - l.p) * l.d / l.d.len());
278     if (sgn(d, c.r) == 0) res.push_back(mid);
279     else if (sgn(d, c.r) < 0) {
280         d = sqrt(c.r * c.r - d * d);
281         res.push_back(mid + l.d.e() * d);
282         res.push_back(mid - l.d.e() * d);
283     }
284     return res;
285 }
286
287 // oab三角形与圆相交的面积
288 lf area(const circle &c, const vec &a, const vec &b) {
289     if (sgn(cross(a, b, c.o), 0) == 0) return 0;
290     vector<vec> p;
291     p.push_back(a);
292     line l(a, b - a);
293     auto tmp = intersection(c, l);
294     if (tmp.size() == 2) {
295         for (auto &i : tmp)
296             if (sgn((a - i) * (b - i), 0) < 0) p.push_back(i);
297     }
298     p.push_back(b);
299     if (p.size() == 4 && sgn((p[0] - p[1]) * (p[2] - p[1]), 0) > 0)
300         swap(p[1], p[2]);
301     lf res = 0;
302     for (int i = 1; i < p.size(); i++)
303         if (c.relation(p[i - 1]) == 1 || c.relation(p[i]) == 1) {
304             lf ang = angle(p[i - 1] - c.o, p[i] - c.o);
305             res += c.r * c.r * ang / 2;
306         } else res += abs(cross(p[i - 1], p[i], c.o)) / 2.0;
307     return res;
308 }
309
310 // 多边形与圆相交的面积
311 lf area(vector<vec> &p, circle c) {
312     lf res = 0;
313     for (int i = 0; i < p.size(); i++) {
314         int j = i + 1 == p.size() ? 0 : i + 1;
315         if (sgn(cross(p[i], p[j], c.o), 0) <= 0) res += area(c, p[i], p[j]);
316         else res -= area(c, p[i], p[j]);
317     }
318     return abs(res);
319 }

```

5.3 扫描线

```

1 #define ls (pos << 1)
2 #define rs (ls | 1)
3 #define mid ((tree[pos].l + tree[pos].r) >> 1)
4 struct Rectangle {
5     ll x_l, y_l, x_r, y_r;

```

```

6 };
7 ll area(vector<Rectangle>& rec) {
8     struct Line {
9         ll x, y_up, y_down;
10        int pd;
11    };
12    vector<Line> line(rec.size() * 2);
13    vector<ll> y_set(rec.size() * 2);
14    for (int i = 0; i < rec.size(); i++) {
15        y_set[i * 2] = rec[i].y_l;
16        y_set[i * 2 + 1] = rec[i].y_r;
17        line[i * 2] = {rec[i].x_l, rec[i].y_r, rec[i].y_l, 1};
18        line[i * 2 + 1] = {rec[i].x_r, rec[i].y_r, rec[i].y_l, -1};
19    }
20    sort(y_set.begin(), y_set.end());
21    y_set.erase(unique(y_set.begin(), y_set.end()), y_set.end());
22    sort(line.begin(), line.end(), [](Line a, Line b) { return a.x < b.x; });
23    struct Data {
24        int l, r;
25        ll len, cnt, raw_len;
26    };
27    vector<Data> tree(4 * y_set.size());
28    function<void(int, int, int)> build = [&](int pos, int l, int r) {
29        tree[pos].l = l;
30        tree[pos].r = r;
31        if (l == r) {
32            tree[pos].raw_len = y_set[r + 1] - y_set[l];
33            tree[pos].cnt = tree[pos].len = 0;
34            return;
35        }
36        build(ls, l, mid);
37        build(rs, mid + 1, r);
38        tree[pos].raw_len = tree[ls].raw_len + tree[rs].raw_len;
39    };
40    function<void(int, int, int, int)> update = [&](int pos, int l, int r,
41                                                int num) {
42        if (l <= tree[pos].l && tree[pos].r <= r) {
43            tree[pos].cnt += num;
44            tree[pos].len = tree[pos].cnt ? tree[pos].raw_len
45                            : tree[pos].l == tree[pos].r
46                                ? 0
47                                : tree[ls].len + tree[rs].len;
48            return;
49        }
50        if (l <= mid) update(ls, l, r, num);
51        if (r > mid) update(rs, l, r, num);
52        tree[pos].len =
53            tree[pos].cnt ? tree[pos].raw_len : tree[ls].len + tree[rs].len;
54    };
55    build(1, 0, y_set.size() - 2);
56    auto find_pos = [&](ll num) {
57        return lower_bound(y_set.begin(), y_set.end(), num) - y_set.begin();

```

```
58     };
59     ll res = 0;
60     for (int i = 0; i < line.size() - 1; i++) {
61         update(1, find_pos(line[i].y_down), find_pos(line[i].y_up) - 1,
62             line[i].pd);
63         res += (line[i + 1].x - line[i].x) * tree[1].len;
64     }
65     return res;
66 }
```

6 杂项

6.1 高精度

```

1 struct bignum {
2     string num;
3
4     bignum() : num("0") {}
5     bignum(const string& num) : num(num) {
6         reverse(this->num.begin(), this->num.end());
7     }
8     bignum(ll num) : num(to_string(num)) {
9         reverse(this->num.begin(), this->num.end());
10    }
11
12    bignum operator+(const bignum& other) {
13        bignum res;
14        res.num.pop_back();
15        res.num.reserve(max(num.size(), other.num.size()) + 1);
16        for (int i = 0, j = 0, x; i < num.size() || i < other.num.size() || j;
17             i++) {
18            x = j;
19            j = 0;
20            if (i < num.size()) x += num[i] - '0';
21            if (i < other.num.size()) x += other.num[i] - '0';
22            if (x >= 10) j = 1, x -= 10;
23            res.num.push_back(x + '0');
24        }
25        res.num.capacity();
26        return res;
27    }
28
29    bignum operator*(const bignum& other) {
30        vector<int> res(num.size() + other.num.size() - 1, 0);
31        for (int i = 0; i < num.size(); i++)
32            for (int j = 0; j < other.num.size(); j++)
33                res[i + j] += (num[i] - '0') * (other.num[j] - '0');
34        int g = 0;
35        for (int i = 0; i < res.size(); i++) {
36            res[i] += g;
37            g = res[i] / 10;
38            res[i] %= 10;
39        }
40        while (g) {
41            res.push_back(g % 10);
42            g /= 10;
43        }
44        int lim = res.size();
45        while (lim > 1 && res[lim - 1] == 0) lim--;
46        bignum res2;
47        res2.num.resize(lim);
48        for (int i = 0; i < lim; i++) res2.num[i] = res[i] + '0';

```

```

49     return res2;
50 }
51
52 bool operator<(const bignum& other) {
53     if (num.size() == other.num.size())
54         for (int i = num.size() - 1; i >= 0; i--)
55             if (num[i] == other.num[i]) continue;
56             else return num[i] < other.num[i];
57     return num.size() < other.num.size();
58 }
59
60 friend istream& operator>>(istream& in, bignum& a) {
61     in >> a.num;
62     reverse(a.num.begin(), a.num.end());
63     return in;
64 }
65 friend ostream& operator<<(ostream& out, bignum a) {
66     reverse(a.num.begin(), a.num.end());
67     return out << a.num;
68 }
69 };

```

6.2 模运算

```

1 struct modint {
2     int x;
3     modint(ll _x = 0) : x(_x % mod) {}
4     modint inv() const { return power(*this, mod - 2); }
5     modint operator+(const modint& b) { return {x + b.x}; }
6     modint operator-() const { return {-x}; }
7     modint operator-(const modint& b) { return {-b + *this}; }
8     modint operator*(const modint& b) { return {(ll)x * b.x}; }
9     modint operator/(const modint& b) { return *this * b.inv(); }
10    friend istream& operator>>(istream& is, modint& other) {
11        ll _x;
12        is >> _x;
13        other = modint(_x);
14        return is;
15    }
16    friend ostream& operator<<(ostream& os, modint other) {
17        other.x = (other.x + mod) % mod;
18        return os << other.x;
19    }
20 };

```

6.3 分数

```

1 struct frac {
2     ll a, b;
3     frac() : a(0), b(1) {}

```



```

4   frac(ll _a, ll _b) : a(_a), b(_b) {
5       assert(b);
6       if (a) {
7           int tmp = gcd(a, b);
8           a /= tmp;
9           b /= tmp;
10      } else *this = frac();
11  }
12  frac operator+(const frac& other) {
13      return frac(a * other.b + other.a * b, b * other.b);
14  }
15  frac operator-() const {
16      frac res = *this;
17      res.a = -res.a;
18      return res;
19  }
20  frac operator-(const frac& other) const { return -other + *this; }
21  frac operator*(const frac& other) const {
22      return frac(a * other.a, b * other.b);
23  }
24  frac operator/(const frac& other) const {
25      assert(other.a);
26      return *this * frac(other.b, other.a);
27  }
28  bool operator<(const frac& other) const { return (*this - other).a < 0; }
29  bool operator<=(const frac& other) const { return (*this - other).a <= 0; }
30  bool operator>=(const frac& other) const { return (*this - other).a >= 0; }
31  bool operator>(const frac& other) const { return (*this - other).a > 0; }
32  bool operator==(const frac& other) const {
33      return a == other.a && b == other.b;
34  }
35  bool operator!=(const frac& other) const { return !(*this == other); }
36 };

```

6.4 表达式求值

```

1  // 格式化表达式
2  string format(const string& s1) {
3      stringstream ss(s1);
4      string s2;
5      char ch;
6      while ((ch = ss.get()) != EOF) {
7          if (ch == ' ') continue;
8          if (isdigit(ch)) s2 += ch;
9          else {
10             if (s2.back() != ' ') s2 += ' ';
11             s2 += ch;
12             s2 += ' ';
13         }
14     }
15     return s2;

```

```
16 }
17
18 // 中缀表达式转后缀表达式
19 string convert(const string& s1) {
20     unordered_map<char, int> rank{
21         {'+', 2}, {'-', 2}, {'*', 1}, {'/', 1}, {'^', 0}};
22     stringstream ss(s1);
23     string s2, temp;
24     stack<char> op;
25     while (ss >> temp) {
26         if (isdigit(temp[0])) s2 += temp + ' ';
27         else if (temp[0] == '(') op.push('(');
28         else if (temp[0] == ')') {
29             while (op.top() != '(') {
30                 s2 += op.top();
31                 s2 += ' ';
32                 op.pop();
33             }
34             op.pop();
35         } else {
36             while (!op.empty() && op.top() != '(' &&
37                 (temp[0] != '^' && rank[op.top()] <= rank[temp[0]] ||
38                 rank[op.top()] < rank[temp[0]])) {
39                 s2 += op.top();
40                 s2 += ' ';
41                 op.pop();
42             }
43             op.push(temp[0]);
44         }
45     }
46     while (!op.empty()) {
47         s2 += op.top();
48         s2 += ' ';
49         op.pop();
50     }
51     return s2;
52 }
53
54 // 计算后缀表达式
55 int calc(const string& s) {
56     stack<int> num;
57     stringstream ss(s);
58     string temp;
59     while (ss >> temp) {
60         if (isdigit(temp[0])) num.push(stoi(temp));
61         else {
62             int b = num.top();
63             num.pop();
64             int a = num.top();
65             num.pop();
66             if (temp[0] == '+') a += b;
67             else if (temp[0] == '-') a -= b;
```

```

68         else if (temp[0] == '*') a *= b;
69         else if (temp[0] == '/') a /= b;
70         else if (temp[0] == '^') a = ksm(a, b);
71         num.push(a);
72     }
73 }
74 return num.top();
75 }

```

6.5 日期

```

1  int month[] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
2  int pre[13];
3  vector<int> leap;
4  struct Date {
5      int y, m, d;
6      bool operator<(const Date& other) const {
7          return array<int, 3>{y, m, d} <
8              array<int, 3>{other.y, other.m, other.d};
9      }
10     Date(const string& s) {
11         stringstream ss(s);
12         char ch;
13         ss >> y >> ch >> m >> ch >> d;
14     }
15     int dis() const {
16         int yd = (y - 1) * 365 +
17             (upper_bound(leap.begin(), leap.end(), y - 1) - leap.begin());
18         int md =
19             pre[m - 1] + (m > 2 && (y % 4 == 0 && y % 100 || y % 400 == 0));
20         return yd + md + d;
21     }
22     int dis(const Date& other) const { return other.dis() - dis(); }
23 };
24 for (int i = 1; i <= 12; i++) pre[i] = pre[i - 1] + month[2];
25 for (int i = 1; i <= 1000000; i++)
26     if (i % 4 == 0 && i % 100 || i % 400 == 0) leap.push_back(i);

```

6.6 对拍

linux/Mac

```

1  g++ a.cpp -o program/a -O2 -std=c++17
2  g++ b.cpp -o program/b -O2 -std=c++17
3  g++ suiji.cpp -o program/suiji -O2 -std=c++17
4
5  cnt=0
6
7  while true; do
8      let cnt++
9      echo TEST:$cnt

```

```

10
11     ./program/suiji > in
12     ./program/a < in > out.a
13     ./program/b < in > out.b
14
15     diff out.a out.b
16     if [ $? -ne 0 ];then break;fi
17 done

```

windows

```

1 @echo off
2
3 g++ a.cpp -o program/a -O2 -std=c++17
4 g++ b.cpp -o program/b -O2 -std=c++17
5 g++ suiji.cpp -o program/suiji -O2 -std=c++17
6
7 set cnt=0
8
9 :again
10     set /a cnt=cnt+1
11     echo TEST:%cnt%
12     .\program\suiji > in
13     .\program\a < in > out.a
14     .\program\b < in > out.b
15
16     fc output.a output.b
17 if not errorlevel 1 goto again

```

6.7 编译常用选项

```
1 -Wall -Woverflow -Wextra -Wpedantic -Wfloat-equal -Wshadow -fsanitize=address,undefined
```

6.8 开栈

不同的编译器可能命令不一样

```

1 -Wl,--stack=0x10000000
2 -Wl,-stack_size -Wl,0x10000000
3 -Wl,-z,stack-size=0x10000000

```

6.9 clang-format

```

1 BasedOnStyle: Google
2 IndentWidth: 4
3 ColumnLimit: 80
4 AllowShortIfStatementsOnASingleLine: AllIfsAndElse
5 AccessModifierOffset: -4
6 EmptyLineBeforeAccessModifier: Leave
7 RemoveBracesLLVM: true

```