

# ACM 常用算法模板

therehello

2023 年 8 月 2 日



# 目录

<b>1</b>	<b>数据结构</b>	<b>2</b>
1.1	并查集 . . . . .	2
1.2	树状数组 . . . . .	2
1.3	线段树 . . . . .	4
1.4	可持久化线段树 . . . . .	5
1.5	st 表 . . . . .	6
<b>2</b>	<b>图论</b>	<b>7</b>
2.1	最短路 . . . . .	7
2.2	树上问题 . . . . .	7
2.2.1	最近公公祖先 . . . . .	7
2.2.2	树链剖分 . . . . .	8
2.3	强连通分量 . . . . .	9
2.4	拓扑排序 . . . . .	10
<b>3</b>	<b>字符串</b>	<b>11</b>
3.1	kmp . . . . .	11
3.2	哈希 . . . . .	11
3.3	manacher . . . . .	12
<b>4</b>	<b>数学</b>	<b>13</b>
4.1	扩展欧几里得 . . . . .	13
4.2	线性筛法 . . . . .	13
4.3	分解质因数 . . . . .	14
4.4	组合数 . . . . .	14
4.5	盒子与球 . . . . .	14
4.6	线性基 . . . . .	15
4.7	矩阵快速幂 . . . . .	16
<b>5</b>	<b>计算几何</b>	<b>17</b>
5.1	扫描线 . . . . .	23
<b>6</b>	<b>杂项</b>	<b>25</b>
6.1	高精度 . . . . .	25
6.2	模运算 . . . . .	26
6.3	分数 . . . . .	27
6.4	表达式求值 . . . . .	27
6.5	对拍 . . . . .	29
6.6	开栈 . . . . .	30
6.7	日期 . . . . .	30

# 1 数据结构

## 1.1 并查集

```

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

```

## 1.2 树状数组

一维

```

1 template <class T>
2 struct Fenwick_tree {
3     Fenwick_tree(int n) : n(n), tree(n + 1, 0) {}
4     T query(int l, int r) {
5         auto query = [&](int pos) {
6             T res = 0;
7             while (pos) {
8                 res += tree[pos];
9                 pos -= lowbit(pos);
10            }
11            return res;
12        };
13        return query(r) - query(l - 1);
14    }
15    void update(int pos, T num) {
16        while (pos <= n) {
17            tree[pos] += num;
18            pos += lowbit(pos);
19        }
20    }
21 private:
22     int n;
23     vector<T> tree;
24 };

```

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 };

```

### 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, l, r, x);
46         update(pos << 1 | 1, l, 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 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 };

```

### 1.5 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

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

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

$$\text{若要求 } x > 0, k > -\frac{x}{dx} \Rightarrow k \geq \lceil \frac{-x+1}{dx} \rceil$$

$$\text{若要求 } x \geq 0, k > -\frac{x}{dx} \Rightarrow k \geq \lceil -\frac{x}{dx} \rceil$$

$$\text{若要求 } y > 0, k < \frac{y}{dy} \Rightarrow k \leq \lfloor \frac{y-1}{dy} \rfloor$$

$$\text{若要求 } y \geq 0, k < \frac{y}{dy} \Rightarrow k \leq \lceil \frac{y}{dy} \rceil$$

```

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 d = __exgcd(b, a % b, x, y);
8     int t = x;
9     x = y;
10    y = t - (a / b) * y;
11    return d;
12 }
13 array<int, 2> exgcd(int a, int b, int c) {
14     int x, y;
15     int gcd_a_b = __exgcd(a, b, x, y);
16     if (c % gcd_a_b) return {INT_MAX, INT_MAX};
17     x *= c / gcd_a_b;
18     y *= c / gcd_a_b;
19     int dx = b / gcd_a_b;
20     int dy = a / gcd_a_b;
21     // x = x + k* dx y = y - k* dy
22     // 调整为 x >= 0 的最小解
23     int k = ceil(-1.0 * x / dx);
24     x += k * dx;
25     y -= k * dy;
26     return {x, y};
27 }

```

### 4.2 线性筛法

```

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

```

```

12     }
13 }
14 return tuple{min_prime, prime};
15 }();

```

### 4.3 分解质因数

```

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

```

### 4.4 组合数

```

1 array<modint, N + 1> fac, ifac;
2 fac[0] = ifac[0] = 1;
3 for (int i = 1; i <= N; i++) {
4     fac[i] = fac[i - 1] * i;
5     ifac[i] = fac[i].inv();
6 }
7 modint C(int n, int m) {
8     if (n < m) return 0;
9     if (m == 0) return 1;
10    if (n <= mod) return fac[n] * ifac[m] * ifac[n - m];
11    // n >= mod 时需要这个
12    return C(n % mod, m % mod) * C(n / mod, m / mod);
13 }

```

### 4.5 盒子与球

$n$  个球,  $m$  个盒



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

4.6 线性基

```
1 // 线性基
2 struct basis {
3     array<unsigned ll, 64> p{};
4
5     // 将x插入此线性基中
6     void insert(unsigned ll x) {
7         for (int i = 63; i >= 0; i--) {
8             if ((x >> i) & 1) {
9                 if (p[i] x ^= p[i];
10                 else {
11                     p[i] = x;
12                     break;
13                 }
14             }
15         }
16     }
17
18     // 将另一个线性基插入此线性基中
19     void insert(basis other) {
20         for (int i = 0; i <= 63; i++) {
21             if (!other.p[i]) continue;
22             insert(other.p[i]);
23         }
24     }
25
26     // 最大异或值
27     unsigned ll max_basis() {
28         unsigned ll res = 0;
29         for (int i = 63; i >= 0; i--)
30             if ((res ^ p[i]) > res) res ^= p[i];
31         return res;
32     }
33 };
```

## 4.7 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

```

1 double eps = 1e-8;
2 const double PI = acos(-1);
3 using T = ll;
4
5 template <typename T>
6 int cmp(T a, T b) {
7     return a != b ? a < b ? -1 : 1 : 0;
8 }
9
10 int cmp(double a, double b) {
11     double c = a - b;
12     if (abs(c) < eps) return 0;
13     return c < 0 ? -1 : 1;
14 }
15
16 // 向量
17 struct vec {
18     T x, y;
19     vec(T _x = 0, T _y = 0) : x(_x), y(_y) {}
20
21     // 模
22     double length() const { return sqrt(x * x + y * y); }
23
24     // 与x轴正方向的夹角
25     double angle() const {
26         double 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 double &theta) {
33         double tmp = x;
34         x = x * cos(theta) - y * sin(theta);
35         y = y * cos(theta) + tmp * sin(theta);
36         return *this;
37     }
38
39     bool operator==(const vec &other) const {
40         return !cmp(x, other.x) && !cmp(y, other.y);
41     }
42     bool operator<(const vec &other) const {
43         int tmp = cmp(angle(), other.angle());
44         if (tmp) return tmp == -1 ? 0 : 1;
45         tmp = cmp(x, other.x);
46         return tmp == -1 ? 0 : 1;
47     }
48
49     vec operator+(const vec &other) const { return {x + other.x, y + other.y}; }
50     vec operator-() const { return {-x, -y}; }

```

```

51     vec operator-(const vec &other) const { return -other + (*this); }
52     vec operator*(const T &other) const { return {x * other, y * other}; }
53     vec operator/(const T &other) const { return {x / other, y / other}; }
54     T operator*(const vec &other) const { return x * other.x + y * other.y; }
55
56     // 叉积 结果大于0, a到b为逆时针, 小于0, a到b顺时针,
57     // 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
58     T operator^(const vec &other) const { return x * other.y - y * other.x; }
59
60     friend istream &operator>>(istream &input, vec &data) {
61         input >> data.x >> data.y;
62         return input;
63     }
64     friend ostream &operator<<(ostream &output, const vec &data) {
65         output << fixed << setprecision(6);
66         output << data.x << " " << data.y;
67         return output;
68     }
69 };
70
71 T cross(const vec &a, const vec &b, const vec &c) { return (a - c) ^ (b - c); }
72
73 // 两点间的距离
74 T distance(const vec &a, const vec &b) {
75     return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
76 }
77
78 // 两向量夹角
79 double angle(const vec &a, const vec &b) {
80     double theta = abs(a.angle() - b.angle());
81     if (theta > PI) theta = 2 * PI - theta;
82     return theta;
83 }
84
85 // 判断点是否在凸包内
86 bool in_polygon(const vec &a, vector<vec> &p) {
87     int n = p.size();
88     if (n == 1) return a == p[0];
89     if (cross(a, p[1], p[0]) > 0 || cross(p.back(), a, p[0]) > 0) return 0;
90     auto cmp = [&p](vec &x, const vec &y) { return ((x - p[0]) ^ y) >= 0; };
91     int i = lower_bound(p.begin() + 2, p.end(), a - p[0], cmp) - p.begin() - 1;
92     return cross(p[(i + 1) % n], a, p[i]) >= 0;
93 }
94
95 // 多边形的面积
96 double polygon_area(vector<vec> &p) {
97     T area = 0;
98     for (int i = 1; i < p.size(); i++) area += p[i - 1] ^ p[i];
99     area += p.back() ^ p[0];
100     return abs(area / 2.0);
101 }
102

```

```

103 // 多边形的周长
104 double polygon_length(vector<vec> &p) {
105     double length = 0;
106     for (int i = 1; i < p.size(); i++) length += (p[i] - p[i-1]).length();
107     length += (p.back() - p[0]).length();
108     return length;
109 }
110
111 // 以整点为顶点的线段上的整点个数
112 T count(const vec &a, const vec &b) {
113     vec c = a - b;
114     return gcd(abs(c.x), abs(c.y)) + 1;
115 }
116
117 // 以整点为顶点的多边形边上整点个数
118 T count(vector<vec> &p) {
119     T cnt = 0;
120     for (int i = 1; i < p.size(); i++) cnt += count(p[i] - p[i-1], p[i-1]);
121     cnt += count(p.back() - p[0], p[0]);
122     return cnt - p.size();
123 }
124
125 // 凸包直径的两个端点
126 auto polygon_dia(vector<vec> &p) {
127     int n = p.size();
128     array<vec, 2> res{};
129     if (n == 1) return res;
130     if (n == 2) return res = {p[0], p[1]};
131     T mx = 0;
132     for (int i = 0, j = 2; i < n; i++) {
133         while (abs(cross(p[i], p[(i + 1) % n], p[j])) <=
134             abs(cross(p[i], p[(i + 1) % n], p[(j + 1) % n])))
135             j = (j + 1) % n;
136         if (T tmp = distance(p[i], p[j]); tmp > mx) {
137             mx = tmp;
138             res = {p[i], p[j]};
139         }
140         if (T tmp = distance(p[(i + 1) % n], p[j]); tmp > mx) {
141             mx = tmp;
142             res = {p[(i + 1) % n], p[j]};
143         }
144     }
145     return res;
146 }
147
148 // 凸包
149 auto convex_hull(vector<vec> &p) {
150     sort(p.begin(), p.end(), [](vec &a, vec &b) {
151         int tmp = cmp(a.x, b.x);
152         if (tmp) return tmp == -1 ? 0 : 1;
153         tmp = cmp(a.y, b.y);
154         return tmp == -1 ? 0 : 1;

```

```

155     });
156     int n = p.size();
157     vector sta(n + 1, 0);
158     vector v(n, false);
159     int tp = -1;
160     sta[++tp] = 0;
161     auto update = [&](int lim, int i) {
162         while (tp > lim &&
163             ((p[sta[tp]] - p[sta[tp - 1]]) ^ (p[i] - p[sta[tp]])) <= 0)
164             v[sta[tp--]] = 0;
165         sta[++tp] = i;
166         v[i] = 1;
167     };
168     for (int i = 1; i < n; i++) update(0, i);
169     int cnt = tp;
170     for (int i = n - 1; i >= 0; i--) {
171         if (v[i]) continue;
172         update(cnt, i);
173     }
174     vector<vec> res(tp);
175     for (int i = 0; i < tp; i++) res[i] = p[sta[i]];
176     return res;
177 }
178
179 // 闵可夫斯基和 两个点集的和构成一个凸包
180 auto minkowski(vector<vec> &a, vector<vec> &b) {
181     int n = a.size(), m = b.size();
182     vector<vec> c{a[0] + b[0]};
183     c.reserve(n + m);
184     int i = 0, j = 0;
185     while (i < n && j < m) {
186         vec x = a[(i + 1) % n] - a[i];
187         vec y = b[(j + 1) % m] - b[j];
188         c.push_back(c.back() + ((x ^ y) >= 0 ? (i++, x) : (j++, y)));
189     }
190     while (i + 1 < n) {
191         c.push_back(c.back() + a[(i + 1) % n] - a[i]);
192         i++;
193     }
194     while (j + 1 < m) {
195         c.push_back(c.back() + b[(j + 1) % m] - b[j]);
196         j++;
197     }
198     return c;
199 }
200
201 // 直线
202 struct line {
203     vec point, direction;
204     line(const vec &p = vec(), const vec &d = vec()) : point(p), direction(d) {}
205 };
206

```

```

207 // 点到直线距离
208 double distance(const vec &a, const line &b) {
209     return abs((b.point - a) ^ (b.point + b.direction - a)) /
210         b.direction.length();
211 }
212
213 // 判断点在直线哪边, 大于0在左边, 等于0在线上, 小于0在右边
214 T side_line(const vec &a, const line &b) { return b.direction ^ (a - b.point); }
215
216 // 两直线是否垂直
217 bool perpendicular(const line &a, const line &b) {
218     return !cmp(a.direction * b.direction, 0);
219 }
220
221 // 点的垂线是否与线段有交点
222 bool perpendicular(const vec &a, const line &b) {
223     vec perpen(-b.direction.y, b.direction.x);
224     bool cross1 = (perpen ^ (b.point - a)) > 0;
225     bool cross2 = (perpen ^ (b.point + b.direction - a)) > 0;
226     return cross1 != cross2;
227 }
228
229 // 两直线是否平行
230 bool parallel(const line &a, const line &b) {
231     return !cmp(a.direction ^ b.direction, 0);
232 }
233
234 // 两直线交点
235 vec intersection(T A, T B, T C, T D, T E, T F) {
236     return {(B * F - C * E) / (A * E - B * D),
237         (C * D - A * F) / (A * E - B * D)};
238 }
239
240 // 两直线交点
241 vec intersection(const line &a, const line &b) {
242     return intersection(a.direction.y, -a.direction.x,
243         a.direction.x * a.point.y - a.direction.y * a.point.x,
244         b.direction.y, -b.direction.x,
245         b.direction.x * b.point.y - b.direction.y * b.point.x);
246 }
247
248 struct circle {
249     vec o;
250     double r;
251     circle(const vec &o, T _r) : o(o), r(_r){};
252     // 点与圆的关系 -1在圆内, 0在圆上, 1在圆外
253     int relation(const vec &other) const {
254         double len = (other - o).length();
255         return cmp(len, r);
256     }
257     double area() { return PI * r * r; }
258 };

```

```

259
260 // 圆与直线交点
261 auto intersection(const circle &c, const line &l) {
262     double d = distance(c.o, l);
263     vector<vec> res;
264     double len = l.direction.length();
265     vec mid = l.point + l.direction * ((c.o - l.point) * l.direction / len);
266     if (!cmp(d, c.r)) res.push_back(mid);
267     else if (d < c.r) {
268         d = sqrt(c.r * c.r - d * d) / len;
269         res.push_back(mid + l.direction * d);
270         res.push_back(mid - l.direction * d);
271     }
272     return res;
273 }
274
275 // oab三角形与圆相交的面积
276 double area(const circle &c, const vec &a, const vec &b) {
277     vec oa = a - c.o, ob = b - c.o;
278     T cab = oa ^ ob;
279     if (!cmp(cab, 0)) return 0;
280     if (c.relation(a) != 1 && c.relation(b) != 1) return cab / 2.0;
281     vec ba = a - b, bo = -ob;
282     vec ab = -ba, ao = -oa;
283     auto r = c.r;
284     double ang;
285     double loa = oa.length(), lob = ob.length(), lab = ab.length();
286     double x =
287         (ba * bo + sqrt(r * r * lab * lab - (ba ^ bo) * (ba ^ bo))) / lab;
288     double y =
289         (ab * ao + sqrt(r * r * lab * lab - (ab ^ ao) * (ab ^ ao))) / lab;
290     if (cmp(lob, r) == -1 && cmp(loa, r) != -1) {
291         ang = cab * (1 - x / lab) / (r * loa);
292         ang = min(max((double)-1, ang), (double)1);
293         return (asin(ang) * r * r + cab * x / lab) / 2;
294     }
295     if (cmp(lob, r) != -1 && cmp(loa, r) == -1) {
296         ang = cab * (1 - y / lab) / (r * lob);
297         ang = min(max((double)-1, ang), (double)1);
298         return (asin(ang) * r * r + cab * y / lab) / 2;
299     }
300     if (cmp(abs(cab), r * lab) != -1 || cmp(ab * ao, 0) != 1 ||
301         cmp(ba * bo, 0) != 1) {
302         ang = cab / (loa * lob);
303         ang = min(max((double)-1, ang), (double)1);
304         double tmp = -asin(ang);
305         if (cmp(oa * ob, 0) == -1)
306             if (cmp(cab, 0) == -1) tmp -= PI;
307             else tmp += PI;
308         else tmp = -tmp;
309         return tmp * r * r / 2;
310     }

```



```

311     ang = cab * (1 - x / lab) / (r * loa);
312     ang = min(max((double)-1, ang), (double)1);
313     double ang2 = cab * (1 - y / lab) / (r * lob);
314     ang2 = min(max((double)-1, ang2), (double)1);
315     return ((asin(ang) + asin(ang2)) * r * r + cab * ((x + y) / lab - 1)) / 2;
316 }
317
318 // 多边形与圆相交的面积
319 double area(vector<vec> &p, circle c) {
320     double res = 0;
321     for (int i = 1; i < p.size(); i++) res += area(c, p[i - 1], p[i]);
322     res += area(c, p.back(), p[0]);
323     return abs(res);
324 }

```

## 5.1 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 constexpr int N = 1e5;
2 constexpr int mod = 1e9 + 7;
3 struct modint {
4     int x;
5     modint(ll _x = 0) : x(_x % mod) {}
6     modint pow(ll b) const {
7         modint res(1), a = *this;
8         while (b) {
9             if (b & 1) res = res * a;
10            a = a * a;
11            b >>= 1;
12        }
13        return res;
14    }
15    modint inv() const { return pow(mod - 2); }
16    modint operator+(const modint& other) { return modint(x + other.x); }
17    modint operator-(const modint& other) { return modint(-other.x); }
18    modint operator-(const modint& other) { return modint(-other + *this); }
19    modint operator*(const modint& other) { return modint((ll)x * other.x); }
20    modint operator/(const modint& other) { return *this * other.inv(); }
21    friend istream& operator>>(istream& is, modint& other) {
22        ll _x;
23        is >> _x;
24        other = modint(_x);
25        return is;
26    }
27    friend ostream& operator<<(ostream& os, modint other) {

```

```

28     other.x = (other.x + mod) % mod;
29     return os << other.x;
30 }
31 };

```

### 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 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++ suiiji.cpp -o program/suiiji -O2 -std=c++17
4
5 cnt=0
6
7 while true; do
8     let cnt++
9     echo TEST:$cnt
10
11     ./program/suiiji > 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++ suiiji.cpp -o program/suiiji -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

```

## 6.6 编译常用选项

```

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

```

## 6.7 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

```

## 6.8 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[i];
25 for (int i = 1; i <= 1000000; i++)
26     if (i % 4 == 0 && i % 100 || i % 400 == 0) leap.push_back(i);

```