ACM 常用算法模板

therehello

2023 年 9 月 29 日

目录

1 数据结构

1.1 并查集

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

1.2 树状数组

1.2.1 一维

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

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, 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   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

```cpp
auto lg = []() {
    array<int, 10000001> lg;
    lg[1] = 0;
    for (int i = 2; i <= 10000000; i++) lg[i] = lg[i >> 1] + 1;
    return lg;
}();
template <typename T>
struct st {
    int n;
    vector<vector<T>> a;
    st(vector<T>& _a) : n(_a.size()) {
        a.assign(lg[n] + 1, vector<int>(n));
        for (int i = 0; i < n; i++) a[0][i] = _a[i];
        for (int j = 1; j <= lg[n]; j++)
            for (int i = 0; i + (1 << j) - 1 < n; i++)
                a[j][i] = max(a[j - 1][i], a[j - 1][i + (1 << (j - 1))]);
    }
    T query(int l, int r) {
        int k = lg[r - l + 1];
        return max(a[k][l], a[k][r - (1 << k) + 1]);
    }
};
```

2 图论

存图

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

2.1 最短路

2.1.1 dijkstra

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

2.2 树上问题

2.2.1 最近公公祖先

倍增法

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

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

树剖

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

2.2.2 树链剖分

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

```
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

```cpp
auto kmp(string& s) {
    vector next(s.size(), -1);
    for (int i = 1, j = -1; i < s.size(); i++) {
        while (j >= 0 && s[i] != s[j + 1]) j = next[j];
        if (s[i] == s[j + 1]) j++;
        next[i] = j;
    }
    // next 意为长度
    for (auto& i : next) i++;
    return next;
}
```

3.2 哈希

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

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

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

```
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 $a, b >= 0$

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

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

```cpp
int __exgcd(int a, int b, int& x, int& y) {
    if (!b) {
        x = 1;
        y = 0;
        return a;
    }
    int g = __exgcd(b, a % b, y, x);
    y -= a / b * x;
    return g;
}

array<int, 2> exgcd(int a, int b, int c) {
    int x, y;
    int g = __exgcd(a, b, x, y);
    if (c % g) return {INT_MAX, INT_MAX};
    int dx = b / g;
    int dy = a / g;
    x = c / g % dx * x % dx;
    if (x < 0) x += dx;
    y = (c - a * x) / b;
    return {x, y};
}
```

4.2 线性筛法

```cpp
constexpr int N = 10000000;
array<int, N + 1> min_prime;
vector<int> primes;
bool ok = []() {
    for (int i = 2; i <= N; i++) {
        if (min_prime[i] == 0) {
            min_prime[i] = i;
            primes.push_back(i);
        }
        for (auto& j : primes) {
            if (j > min_prime[i] || j > N / i) break;
            min_prime[j * i] = j;
        }
    }
    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 = 1e6;
2 array<modint, N + 1> fac, ifac;
3
4 modint C(int n, int m) {
5    if (n < m) return 0;
6    if (n <= mod) return fac[n] * ifac[m] * ifac[n - m];
```

```
7       // n >= mod 时需要这个
8       return C(n % mod, m % mod) * C(n / mod, m / mod);
9   }
10
11  auto _ = []() {
12      fac[0] = 1;
13      for (int i = 1; i <= N; i++) fac[i] = fac[i - 1] * i;
14      ifac[N] = fac[N].inv();
15      for (int i = N - 1; i >= 0; i--) ifac[i] = ifac[i + 1] * (i + 1);
16      return true;
17  }();
```

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：$\mathrm{id}_k(n) = n^k$。（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：$\mu(n) = \begin{cases} 1 & n = 1 \\ 0 & \exists d > 1, d^2 \mid n \\ (-1)^{\omega(n)} & \text{otherwise} \end{cases}$，其中 $\omega(n)$ 表示 n 的本质不同质因子个数，它是一个加性函数。

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

- $\varepsilon = \mu * 1 \iff \varepsilon(n) = \sum_{d|n} \mu(d)$

- $id = \varphi * 1 \iff id(n) = \sum_{d|n} \varphi(d)$

- $d = 1 * 1 \iff d(n) = \sum_{d|n} 1$

- $\sigma = \mathrm{id} * 1 \iff \sigma(n) = \sum_{d|n} d$

- $\varphi = \mu * \mathrm{id} \iff \varphi(n) = \sum_{d|n} d \cdot \mu(\frac{n}{d})$

4.9 欧拉函数

```cpp
constexpr int N = 1e6;
array<int, N + 1> phi;
auto _ = []() {
    iota(phi.begin() + 1, phi.end(), 1);
    for (int i = 2; i <= N; i++) {
        if (phi[i] == i)
            for (int j = i; j <= N; j += i) phi[j] = phi[j] / i * (i - 1);
    }
    return true;
}();
```

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 $\sum_{d|n} \mu(d) = \varepsilon(n)$, $\mu * 1 = \varepsilon$

- $[\gcd(i,j) = 1] = \sum_{d|\gcd(i,j)} \mu(d)$

```cpp
constexpr int N = 1e6;
array<int, N + 1> miu;
array<bool, N + 1> ispr;

```

```
5  auto _ = []() {
6      miu.fill(1);
7      ispr.fill(1);
8      for (int i = 2; i <= N; i++) {
9          if (!ispr[i]) continue;
10         miu[i] = -1;
11         for (int j = 2 * i; j <= N; j += i) {
12             ispr[j] = 0;
13             if ((j / i) % i == 0) miu[j] = 0;
14             else miu[j] *= -1;
15         }
16     }
17     return true;
18 }();
```

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 $g = f * \mu$。

$f \to g$ 称为莫比乌斯反演，$g \to f$ 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 $S(n) = \sum_{i=1}^{n} f(i)$。

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

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 $\sum_{i=2}^{n} g(i)S\left(\left\lfloor \frac{n}{i} \right\rfloor\right)$。

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
13 ll sum_miu(ll n) {
14     if (n <= N) return sm[n];
15     if (sm2.count(n)) return sm2[n];
```

```
16      ll res = 0, l = 2;
17      while (l <= n) {
18          ll r = n / (n / l);
19          res = res + (r - l + 1) * sum_miu(n / l);
20          l = r + 1;
21      }
22      return sm2[n] = 1 - res;
23  }
```

4.12 多项式

```
1  #define countr_zero(n) __builtin_ctz(n)
2  constexpr int N = 1e6;
3  array<int, N + 1> inv;
4
5  int power(int a, int b) {
6      int res = 1;
7      while (b) {
8          if (b & 1) res = 1ll * res * a % mod;
9          a = 1ll * a * a % mod;
10         b >>= 1;
11     }
12     return res;
13 }
14
15 namespace NFTS {
16 int g = 3;
17 vector<int> rev, roots{0, 1};
18 void dft(vector<int> &a) {
19     int n = a.size();
20     if (rev.size() != n) {
21         int k = countr_zero(n) - 1;
22         rev.resize(n);
23         for (int i = 0; i < n; ++i) rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
24     }
25     if (roots.size() < n) {
26         int k = countr_zero(roots.size());
27         roots.resize(n);
28         while ((1 << k) < n) {
29             int e = power(g, (mod - 1) >> (k + 1));
30             for (int i = 1 << (k - 1); i < (1 << k); ++i) {
31                 roots[2 * i] = roots[i];
32                 roots[2 * i + 1] = 1ll * roots[i] * e % mod;
33             }
34             ++k;
35         }
36     }
37     for (int i = 0; i < n; ++i)
38         if (rev[i] < i) swap(a[i], a[rev[i]]);
39     for (int k = 1; k < n; k *= 2) {
40         for (int i = 0; i < n; i += 2 * k) {
```

```
41              for (int j = 0; j < k; ++j) {
42                  int u = a[i + j];
43                  int v = 1ll * a[i + j + k] * roots[k + j] % mod;
44                  int x = u + v, y = u - v;
45                  if (x >= mod) x -= mod;
46                  if (y < 0) y += mod;
47                  a[i + j] = x;
48                  a[i + j + k] = y;
49              }
50          }
51      }
52  }
53  void idft(vector<int> &a) {
54      int n = a.size();
55      reverse(a.begin() + 1, a.end());
56      dft(a);
57      int inv_n = power(n, mod - 2);
58      for (int i = 0; i < n; ++i) a[i] = 1ll * a[i] * inv_n % mod;
59  }
60  }  // namespace NFTS
61
62  struct poly {
63      poly &format() {
64          while (!a.empty() && a.back() == 0) a.pop_back();
65          return *this;
66      }
67      poly &reverse() {
68          ::reverse(a.begin(), a.end());
69          return *this;
70      }
71      vector<int> a;
72      poly() {}
73      poly(int x) {
74          if (x) a = {x};
75      }
76      poly(const vector<int> &_a) : a(_a) {}
77      int size() const { return a.size(); }
78      int &operator[](int id) { return a[id]; }
79      int at(int id) const {
80          if (id < 0 || id >= (int)a.size()) return 0;
81          return a[id];
82      }
83      poly operator-() const {
84          auto A = *this;
85          for (auto &x : A.a) x = (x == 0 ? 0 : mod - x);
86          return A;
87      }
88      poly mulXn(int n) const {
89          auto b = a;
90          b.insert(b.begin(), n, 0);
91          return poly(b);
92      }
```

```
93      poly modXn(int n) const {
94          if (n > size()) return *this;
95          return poly({a.begin(), a.begin() + n});
96      }
97      poly divXn(int n) const {
98          if (size() <= n) return poly();
99          return poly({a.begin() + n, a.end()});
100     }
101     poly &operator+=(const poly &rhs) {
102         if (size() < rhs.size()) a.resize(rhs.size());
103         for (int i = 0; i < rhs.size(); ++i)
104             if ((a[i] += rhs.a[i]) >= mod) a[i] -= mod;
105         return *this;
106     }
107     poly &operator-=(const poly &rhs) {
108         if (size() < rhs.size()) a.resize(rhs.size());
109         for (int i = 0; i < rhs.size(); ++i)
110             if ((a[i] -= rhs.a[i]) < 0) a[i] += mod;
111         return *this;
112     }
113     poly &operator*=(poly rhs) {
114         int n = size(), m = rhs.size(), tot = max(1, n + m - 1);
115         int sz = 1 << __lg(tot * 2 - 1);
116         a.resize(sz);
117         rhs.a.resize(sz);
118         NFTS::dft(a);
119         NFTS::dft(rhs.a);
120         for (int i = 0; i < sz; ++i) a[i] = 1ll * a[i] * rhs.a[i] % mod;
121         NFTS::idft(a);
122         return *this;
123     }
124     poly &operator/=(poly rhs) {
125         int n = size(), m = rhs.size();
126         if (n < m) return (*this) = poly();
127         reverse();
128         rhs.reverse();
129         (*this) *= rhs.inv(n - m + 1);
130         a.resize(n - m + 1);
131         reverse();
132         return *this;
133     }
134     poly &operator%=(poly rhs) { return (*this) -= (*this) / rhs * rhs; }
135     poly operator+(const poly &rhs) const { return poly(*this) += rhs; }
136     poly operator-(const poly &rhs) const { return poly(*this) -= rhs; }
137     poly operator*(poly rhs) const { return poly(*this) *= rhs; }
138     poly operator/(poly rhs) const { return poly(*this) /= rhs; }
139     poly operator%(poly rhs) const { return poly(*this) %= rhs; }
140     poly powModPoly(int n, poly p) {
141         poly r(1), x(*this);
142         while (n) {
143             if (n & 1) (r *= x) %= p;
144             (x *= x) %= p;
```

```
145            n >>= 1;
146        }
147        return r;
148    }
149    int inner(const poly &rhs) {
150        int r = 0, n = min(size(), rhs.size());
151        for (int i = 0; i < n; ++i) r = (r + 1ll * a[i] * rhs.a[i]) % mod;
152        return r;
153    }
154    poly derivation() const {
155        if (a.empty()) return poly();
156        int n = size();
157        vector<int> r(n - 1);
158        for (int i = 1; i < n; ++i) r[i - 1] = 1ll * a[i] * i % mod;
159        return poly(r);
160    }
161    poly integral() const {
162        if (a.empty()) return poly();
163        int n = size();
164        vector<int> r(n + 1);
165        for (int i = 0; i < n; ++i) r[i + 1] = 1ll * a[i] * ::inv[i + 1] % mod;
166        return poly(r);
167    }
168    poly inv(int n) const {
169        assert(a[0] != 0);
170        poly x(power(a[0], mod - 2));
171        int k = 1;
172        while (k < n) {
173            k *= 2;
174            x *= (poly(2) - modXn(k) * x).modXn(k);
175        }
176        return x.modXn(n);
177    }
178    // 需要保证首项为 1
179    poly log(int n) const {
180        return (derivation() * inv(n)).integral().modXn(n);
181    }
182    // 需要保证首项为 0
183    poly exp(int n) const {
184        poly x(1);
185        int k = 1;
186        while (k < n) {
187            k *= 2;
188            x = (x * (poly(1) - x.log(k) + modXn(k))).modXn(k);
189        }
190        return x.modXn(n);
191    }
192    // 需要保证首项为 1，开任意次方可以先 ln 再 exp 实现。
193    poly sqrt(int n) const {
194        poly x(1);
195        int k = 1;
196        while (k < n) {
```

```cpp
            k *= 2;
            x += modXn(k) * x.inv(k);
            x = x.modXn(k) * inv2;
        }
        return x.modXn(n);
    }
    // 减法卷积，也称转置卷积 {\rm MULT}(F(x),G(x))=\sum_{i\ge0}(\sum_{j\ge
    // 0}f_{i+j}g_j)x^i
    poly mulT(poly rhs) const {
        if (rhs.size() == 0) return poly();
        int n = rhs.size();
        ::reverse(rhs.a.begin(), rhs.a.end());
        return ((*this) * rhs).divXn(n - 1);
    }
    int eval(int x) {
        int r = 0, t = 1;
        for (int i = 0, n = size(); i < n; ++i) {
            r = (r + 1ll * a[i] * t) % mod;
            t = 1ll * t * x % mod;
        }
        return r;
    }
    // 多点求值新科技：https://jkloverdcoi.github.io/2020/08/04/转置原理及其应用/
    // 模板例题：https://www.luogu.com.cn/problem/P5050
    auto evals(vector<int> &x) const {
        if (size() == 0) return vector(x.size(), 0);
        int n = x.size();
        vector ans(n, 0);
        vector<poly> g(4 * n);
        auto build = [&](auto self, int l, int r, int p) -> void {
            if (r - l == 1) {
                g[p] = poly({1, x[l] ? mod - x[l] : 0});
            } else {
                int m = (l + r) / 2;
                self(self, l, m, 2 * p);
                self(self, m, r, 2 * p + 1);
                g[p] = g[2 * p] * g[2 * p + 1];
            }
        };
        build(build, 0, n, 1);
        auto solve = [&](auto self, int l, int r, int p, poly f) -> void {
            if (r - l == 1) {
                ans[l] = f[0];
            } else {
                int m = (l + r) / 2;
                self(self, l, m, 2 * p, f.mulT(g[2 * p + 1]).modXn(m - l));
                self(self, m, r, 2 * p + 1, f.mulT(g[2 * p]).modXn(r - m));
            }
        };
        solve(solve, 0, n, 1, mulT(g[1].inv(size())).modXn(n));
        return ans;
    }
```

```
249 };  // 全家桶测试: https://www.luogu.com.cn/training/3015#information
250
251 auto _ = []() {
252     inv[0] = inv[1] = 1;
253     for (int i = 2; i < inv.size(); i++)
254         inv[i] = 1ll * (mod - mod / i) * inv[mod % i] % mod;
255     return true;
256 }();
```

4.13 盒子与球

n 个球, m 个盒

球同	盒同	可空	公式
✓	✓	✓	$f_{n,m} = f_{n,m-1} + f_{n-m,m}$ 或 $[x^n]e^{\sum\limits_{i=1}^{m}\sum\limits_{j=1}^{\infty}\frac{x^{ij}}{j}}$
✓	✓	✗	$f_{n-m,m}$
✗	✓	✓	$\Sigma_{i=1}^{m}g_{n,i}$ 或 $\sum\limits_{i=1}^{m}\sum\limits_{j=0}^{i}\frac{j^n}{j!}\frac{(-1)^{i-j}}{(i-j)!}$
✗	✓	✗	$g_{n,m} = g_{n-1,m-1} + m * g_{n-1,m}$ 或 $\frac{1}{m!}\sum\limits_{i=0}^{m}(-1)^i\binom{m}{i}(m-i)^n$
✓	✗	✓	C_{n+m-1}^{m-1}
✓	✗	✗	C_{n-1}^{m-1}
✗	✗	✓	m^n
✗	✗	✗	$m! * g_{n,m}$ 或 $\sum\limits_{i=0}^{m}(-1)^i\binom{m}{i}(m-i)^n$

4.13.1 球同, 盒同, 可空

```
1 int solve(int n, int m) {
2     vector a(n + 1, 0);
3     for (int i = 1; i <= m; i++)
4         for (int j = i, k = 1; j <= n; j += i, k++)
5             a[j] = (a[j] + inv[k]) % mod;
6     auto p = poly(a).exp(n + 1);
7     return (p.a[n] + mod) % mod;
8 }
```

若要求不超过 k 个, 答案为 $[x^n y^m]\prod\limits_{i=0}^{k}\left(\sum\limits_{j=0}^{m}x^{ij}y^j\right)$。

4.13.2 球不同，盒同，可空

```
int solve(int n, int m) {
    vector a(n + 1, 0);
    vector b(n + 1, 0);
    for (int i = 0; i <= n; i++) {
        a[i] = ifac[i];
        if (i & 1) a[i] = -a[i];
        b[i] = 1ll * power(i, n) * ifac[i] % mod;
    }
    auto p = poly(a) * poly(b);
    int ans = 0;
    for (int i = 1; i <= min(n, m); i++) ans = (ans + p.a[i]) % mod;
    return (ans + mod) % mod;
}
```

若要求不超过 k 个，答案为 $m! \cdot [x^n y^m] \prod\limits_{i=0}^{k} \left(\sum\limits_{j=0}^{n} \frac{1}{i!^j} x^{ij} y^j \right)$。

4.13.3 球同，盒不同，可空

若要求不超过 k 个，答案为 $[x^n] \left(\sum\limits_{i=0}^{k} x^i \right)^m = [x^n] \frac{(x^{k+1}-1)^m}{(x-1)^m}$。

也可以考虑容斥，令 $f(i)$ 表示至少有 i 个盒子装了 $> k$ 个球方案数，$f(i) = \binom{m}{i}\binom{n-(k+1)i+m-1}{m-1}$。

总方案数则为 $\sum\limits_{i=0}^{m} (-1)^i f(i)$。

4.13.4 球同，盒不同，不可空

若要求不超过 k 个，答案为 $[x^n] \left(\sum\limits_{i=1}^{k} x^i \right)^m = [x^n] \frac{(x^k-1)^m x^m}{(x-1)^m}$。

也可以考虑容斥，令 $f(i)$ 表示至少有 i 个盒子装了 $> k$ 个球方案数，$f(i) = \binom{m}{i}\binom{n-ki-1}{m-1}$。

总方案数则为 $\sum\limits_{i=0}^{m} (-1)^i f(i)$。

4.13.5 球不同，盒不同，可空

若要求不超过 k 个，答案为 $m! \cdot [x^n] \left(\sum\limits_{i=0}^{k} \frac{1}{i!} x^i \right)^m$。

4.13.6 球不同，盒不同，不可空

若要求不超过 k 个，答案为 $m! \cdot [x^n] \left(\sum\limits_{i=1}^{k} \frac{1}{i!} x^i \right)^m$。

4.14 线性基

```
// 线性基
struct basis {
    int rnk = 0;
    array<ull, 64> p{};

    // 将x插入此线性基中
    void insert(ull x) {
        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 };
```

4.15　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(n), 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

```cpp
constexpr double inf = 1e100;

// 向量
struct vec {
    static bool cmp(const vec &a, const vec &b) {
        return tie(a.x, a.y) < tie(b.x, b.y);
    }

    ll x, y;
    vec() : x(0), y(0) {}
    vec(ll _x, ll _y) : x(_x), y(_y) {}

    // 模
    ll len2() const { return x * x + y * y; }
    double len() const { return sqrt(x * x + y * y); }

    // 是否在上半轴
    bool up() const { return y > 0 || y == 0 && x >= 0; }

    bool operator==(const vec &b) const { return tie(x, y) == tie(b.x, b.y); }
    // 极角排序
    bool operator<(const vec &b) const {
        if (up() != b.up()) return up() > b.up();
        ll tmp = (*this) ^ b;
        return tmp ? tmp > 0 : cmp(*this, b);
    }

    vec operator+(const vec &b) const { return {x + b.x, y + b.y}; }
    vec operator-() const { return {-x, -y}; }
    vec operator-(const vec &b) const { return -b + (*this); }
    vec operator*(ll b) const { return {x * b, y * b}; }
    ll operator*(const vec &b) const { return x * b.x + y * b.y; }

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

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

```cpp
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(lf _x, lf _y) : x(_x), y(_y) {}
19
20        // 模
21        lf len2() const { return x * x + y * y; }
22        lf len() const { return sqrt(x * x + y * y); }
23
24        // 与x轴正方向的夹角
25        lf angle() const {
26            lf 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 lf &theta) const {
33            return {x * cos(theta) - y * sin(theta),
34                    y * cos(theta) + x * sin(theta)};
35        }
36
37        vec e() const {
38            lf 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            lf 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*(lf b) const { return {x * b, y * b}; }
61        vec operator/(lf b) const { return {x / b, y / b}; }
62        lf 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 形成的平行四边行的面积
66        lf 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

```cpp
struct bignum {
    string num;

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

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

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

```
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;
```

```cpp
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
```