

ACM 常用算法模板

therehello

2023 年 10 月 23 日

目录

1 数据结构	3
1.1 并查集	3
1.2 树状数组	3
1.2.1 一维	3
1.2.2 二维	3
1.2.3 三维	4
1.3 线段树	4
1.4 普通平衡树	6
1.4.1 树状数组实现	6
1.5 可持久化线段树	7
1.6 st 表	8
2 图论	9
2.1 最短路	9
2.1.1 dijkstra	9
2.2 树上问题	9
2.2.1 最近公公祖先	9
2.2.2 树链剖分	10
2.3 强连通分量	11
2.4 拓扑排序	11
3 字符串	13
3.1 kmp	13
3.2 哈希	13
3.3 manacher	14
4 数学	15
4.1 扩展欧几里得	15
4.2 线性代数	15
4.2.1 向量公约数	15
4.3 筛法	16
4.4 分解质因数	18
4.5 pollard rho	18
4.6 组合数	20
4.6.1 常用式子	20
4.7 数论分块	21
4.8 积性函数	21
4.8.1 定义	21
4.8.2 例子	21
4.9 狄利克雷卷积	21
4.9.1 性质	21
4.9.2 例子	22
4.10 欧拉函数	22
4.11 莫比乌斯反演	22

4.11.1 莫比乌斯函数性质	22
4.11.2 莫比乌斯变换/反演	23
4.12 杜教筛	23
4.12.1 示例	23
4.13 多项式	23
4.14 盒子与球	28
4.14.1 球同, 盒同, 可空	28
4.14.2 球不同, 盒同, 可空	29
4.14.3 球同, 盒不同, 可空	29
4.14.4 球同, 盒不同, 不可空	29
4.14.5 球不同, 盒不同, 可空	29
4.14.6 球不同, 盒不同, 不可空	29
4.15 线性基	29
4.16 矩阵快速幂	30
5 计算几何	31
5.1 整数	31
5.2 浮点数	35
5.3 扫描线	42
6 杂项	44
6.1 快读	44
6.2 高精度	44
6.3 离散化	45
6.4 模运算	46
6.5 分数	46
6.6 表达式求值	47
6.7 日期	48
6.8 builtin 函数	49
6.9 对拍	49
6.10 编译常用选项	50
6.11 开栈	50
6.12 clang-format	50

1 数据结构

1.1 并查集

```

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

```

1.2 树状数组

1.2.1 一维

```

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

```

1.2.2 二维

```

1 template <class T>
2 struct Fenwick_tree_2 {
3     Fenwick_tree_2(int n, int m) : n(n), m(m), tree(n + 1, vector<T>(m + 1)) {}
4     T query(int l1, int r1, int l2, int r2) {
5         auto query = [&](int l, int r) {
6             T res = 0;
7             for (int i = l; i; i -= lowbit(i))

```

```

8         for (int j = r; j; j -= lowbit(j)) res += tree[i][j];
9         return res;
10    };
11    return query(l2, r2) - query(l2, r1 - 1) - query(l1 - 1, r2) + query(l1 - 1, r1 - 1);
12 }
13 void update(int x, int y, T num) {
14     for (int i = x; i <= n; i += lowbit(i))
15         for (int j = y; j <= m; j += lowbit(j)) tree[i][j] += num;
16 }
17 private:
18     int n, m;
19     vector<vector<T>> tree;
20 };

```

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), m(m), k(k), tree(n + 1, vector<vector<T>>(m + 1, vector<T>(k + 1))) {}
5     T query(int a, int b, int c, int d, int e, int f) {
6         auto query = [&](int x, int y, int z) {
7             T res = 0;
8             for (int i = x; i; i -= lowbit(i))
9                 for (int j = y; j; j -= lowbit(j))
10                     for (int p = z; p; p -= lowbit(p)) res += tree[i][j][p];
11             return res;
12         };
13         T res = query(d, e, f);
14         res -= query(a - 1, e, f) + query(d, b - 1, f) + query(d, e, c - 1);
15         res += query(a - 1, b - 1, f) + query(a - 1, e, c - 1) + query(d, b - 1, c - 1);
16         res -= query(a - 1, b - 1, c - 1);
17         return res;
18     }
19     void update(int x, int y, int z, T num) {
20         for (int i = x; i <= n; i += lowbit(i))
21             for (int j = y; j <= m; j += lowbit(j))
22                 for (int p = z; p <= k; p += lowbit(p)) tree[i][j][p] += num;
23     }
24 private:
25     int n, m, k;
26     vector<vector<vector<T>>> tree;
27 };

```

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) { tree[pos].data = tree[pos << 1].data + tree[pos << 1 | 1].data; }
18  inline void pushdown(int pos) {
19      tree[pos << 1].data = tree[pos << 1].data + tree[pos].data.lazytag;
20      tree[pos << 1 | 1].data = tree[pos << 1 | 1].data + tree[pos].data.lazytag;
21      tree[pos].data.lazytag = Num::zero();
22  }
23  void build(vector<Data>& a, int pos, int l, int r) {
24      tree[pos].l = l;
25      tree[pos].r = r;
26      if (l == r) {
27          tree[pos].data = a[l - 1];
28          return;
29      }
30      int mid = (tree[pos].l + tree[pos].r) >> 1;
31      build(a, pos << 1, l, mid);
32      build(a, pos << 1 | 1, mid + 1, r);
33      pushup(pos);
34  }
35  void update(int pos, int& l, int& r, Num& x) {
36      if (l > tree[pos].r || r < tree[pos].l) return;
37      if (l <= tree[pos].l && tree[pos].r <= r) {
38          tree[pos].data = tree[pos].data + x;
39          return;
40      }
41      pushdown(pos);
42      update(pos << 1, l, r, x);
43      update(pos << 1 | 1, l, r, x);
44      pushup(pos);
45  }
46  Data query(int pos, int& l, int& r) {
47      if (l > tree[pos].r || r < tree[pos].l) return Data::zero();
48      if (l <= tree[pos].l && tree[pos].r <= r) return tree[pos].data;
49      pushdown(pos);
50      return query(pos << 1, l, r) + query(pos << 1 | 1, l, r);
51  }
52 };
53 struct Num {
54     ll add;
55     inline static Num zero() { return {0}; }
56     inline Num operator+(Num b) { return {add + b.add}; }
57 };
58 struct Data {
59     ll sum, len;
60     Num lazytag;
61     inline static Data zero() { return {0, 0, Num::zero()}; }
62     inline Data operator+(Num b) { return {sum + len * b.add, len, lazytag + b}; }
63     inline Data operator+(Data b) { return {sum + b.sum, len + b.len, Num::zero()}; }
64 };

```

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 int lowbit(int x) { return x & -x; }
2
3 template <typename T>
4 struct treap {
5     int n, size;
6     vector<int> t;
7     vector<T> t2, S;
8     treap(const vector<T>& a) : S(a) {
9         sort(S.begin(), S.end());
10        S.erase(unique(S.begin(), S.end()), S.end());
11        n = S.size();
12        size = 0;
13        t = vector<int>(n + 1);
14        t2 = vector<T>(n + 1);
15    }
16    int pos(T x) { return lower_bound(S.begin(), S.end(), x) - S.begin() + 1; }
17    int sum(int pos) {
18        int res = 0;
19        while (pos) {
20            res += t[pos];
21            pos -= lowbit(pos);
22        }
23        return res;
24    }
25
26    // 插入cnt个x
27    void insert(T x, int cnt) {
28        size += cnt;
29        int i = pos(x);
30        assert(i <= n && S[i - 1] == x);
31        for (; i <= n; i += lowbit(i)) {
32            t[i] += cnt;
33            t2[i] += cnt * x;
34        }
35    }
36
37    // 删除cnt个x
38    void erase(T x, int cnt) {
39        assert(cnt <= count(x));
40        insert(x, -cnt);
41    }
42
43    // x的排名
44    int rank(T x) {
45        assert(count(x));
46        return sum(pos(x) - 1) + 1;
47    }
48
49    // 统计出现次数
50    int count(T x) { return sum(pos(x)) - sum(pos(x) - 1); }
51
52    // 第k小
53    T kth(int k) {

```



```

54     assert(0 < k && k <= size);
55     int cnt = 0, x = 0;
56     for (int i = __lg(n); i >= 0; i--) {
57         x += 1 << i;
58         if (x >= n || cnt + t[x] >= k) x -= 1 << i;
59         else cnt += t[x];
60     }
61     return S[x];
62 }
63
64 // 前k小的数之和
65 T pre_sum(int k) {
66     assert(0 < k && k <= size);
67     int cnt = 0, x = 0;
68     T res = 0;
69     for (int i = __lg(n); i >= 0; i--) {
70         x += 1 << i;
71         if (x >= n || cnt + t[x] >= k) x -= 1 << i;
72         else {
73             cnt += t[x];
74             res += t2[x];
75         }
76     }
77     return res + (k - cnt) * S[x];
78 }
79
80 // 小于x, 最大的数
81 T prev(T x) { return kth(sum(pos(x) - 1)); }
82
83 // 大于x, 最小的数
84 T next(T x) { return kth(sum(pos(x)) + 1); }
85 };

```

1.5 可持久化线段树

```

1  constexpr int MAXN = 200000;
2  vector<int> root(MAXN << 5);
3  struct Persistent_seg {
4      int n;
5      struct Data {
6          int ls, rs;
7          int val;
8      };
9      vector<Data> tree;
10     Persistent_seg(int n, vector<int>& a) : n(n) { root[0] = build(1, n, a); }
11     int build(int l, int r, vector<int>& a) {
12         if (l == r) {
13             tree.push_back({0, 0, a[l]});
14             return tree.size() - 1;
15         }
16         int mid = l + r >> 1;
17         int ls = build(l, mid, a), rs = build(mid + 1, r, a);
18         tree.push_back({ls, rs, tree[ls].val + tree[rs].val});
19         return tree.size() - 1;
20     }
21     int update(int rt, const int& idx, const int& val, int l, int r) {
22         if (l == r) {

```

```

23     tree.push_back({0, 0, tree[rt].val + val});
24     return tree.size() - 1;
25 }
26 int mid = l + r >> 1, ls = tree[rt].ls, rs = tree[rt].rs;
27 if (idx <= mid) ls = update(ls, idx, val, l, mid);
28 else rs = update(rs, idx, val, mid + 1, r);
29 tree.push_back({ls, rs, tree[ls].val + tree[rs].val});
30 return tree.size() - 1;
31 }
32 int query(int rt1, int rt2, int k, int l, int r) {
33     if (l == r) return l;
34     int mid = l + r >> 1;
35     int lcnt = tree[tree[rt2].ls].val - tree[tree[rt1].ls].val;
36     if (k <= lcnt) return query(tree[rt1].ls, tree[rt2].ls, k, l, mid);
37     else return query(tree[rt1].rs, tree[rt2].rs, k - lcnt, mid + 1, r);
38 }
39 };

```

1.6 st 表

```

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

```

2 图论

存图

```

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

```

2.1 最短路

2.1.1 dijkstra

```

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

```

2.2 树上问题

2.2.1 最近公公祖先

倍增法

```

1 vector<int> dep;
2 vector<array<int, 21>> fa;
3 dep.assign(n + 1, 0);
4 fa.assign(n + 1, array<int, 21>{{}});
5 void binary_jump(int root) {
6     function<void(int)> dfs = [&](int t) {
7         dep[t] = dep[fa[t][0]] + 1;
8         for (auto& [to] : graph[t]) {
9             if (to == fa[t][0]) continue;
10            fa[to][0] = t;
11            dfs(to);

```

```

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

```

树剖

```

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

```

2.2.2 树链剖分

```

1 vector<int> fa, siz, dep, son, dfn, rnk, top;
2 fa.assign(n + 1, 0);
3 siz.assign(n + 1, 0);
4 dep.assign(n + 1, 0);
5 son.assign(n + 1, 0);
6 dfn.assign(n + 1, 0);
7 rnk.assign(n + 1, 0);
8 top.assign(n + 1, 0);
9 void hld(int root) {
10     function<void(int)> dfs1 = [&](int t) {
11         dep[t] = dep[fa[t]] + 1;
12         siz[t] = 1;
13         for (auto& [to, w] : graph[t]) {
14             if (to == fa[t]) continue;
15             fa[to] = t;
16             dfs1(to);
17             if (siz[son[t]] < siz[to]) son[t] = to;
18             siz[t] += siz[to];
19         }
20     };
21     dfs1(root);
22     int dfn_tail = 0;
23     for (int i = 1; i <= n; i++) top[i] = i;
24     function<void(int)> dfs2 = [&](int t) {
25         dfn[t] = ++dfn_tail;

```

```

26     rnk[dfn_tail] = t;
27     if (!son[t]) return;
28     top[son[t]] = top[t];
29     dfs2(son[t]);
30     for (auto& [to, w] : graph[t]) {
31         if (to == fa[t] || to == son[t]) continue;
32         dfs2(to);
33     }
34 };
35 dfs2(root);
36 }

```

2.3 强连通分量

```

1 void tarjan(Graph& g1, Graph& g2) {
2     int dfn_tail = 0, cnt = 0;
3     vector<int> dfn(g1.n + 1, 0), low(g1.n + 1, 0), exist(g1.n + 1, 0), belong(g1.n + 1, 0);
4     stack<int> sta;
5     function<void(int)> dfs = [&](int t) {
6         dfn[t] = low[t] = ++dfn_tail;
7         sta.push(t);
8         exist[t] = 1;
9         for (auto& [to] : g1.graph[t])
10             if (!dfn[to]) {
11                 dfs(to);
12                 low[t] = min(low[t], low[to]);
13             } else if (exist[to]) low[t] = min(low[t], dfn[to]);
14         if (dfn[t] == low[t]) {
15             cnt++;
16             while (int temp = sta.top()) {
17                 belong[temp] = cnt;
18                 exist[temp] = 0;
19                 sta.pop();
20                 if (temp == t) break;
21             }
22         }
23     };
24     for (int i = 1; i <= g1.n; i++)
25         if (!dfn[i]) dfs(i);
26     g2 = Graph(cnt);
27     for (int i = 1; i <= g1.n; i++) g2.w[belong[i]] += g1.w[i];
28     for (int i = 1; i <= g1.n; i++)
29         for (auto& [to] : g1.graph[i])
30             if (belong[i] != belong[to]) g2.add(belong[i], belong[to]);
31 }

```

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 = 1e6;
2 int pow_base[N + 1][2];
3 constexpr ll mod[2] = {(int)2e9 + 11, (int)2e9 + 33}, base[2] = {(int)2e5 + 11, (int)2e5 + 33};
4
5 struct Hash {
6     int size;
7     vector<array<int, 2>> a;
8     Hash() {}
9     Hash(const string& s) {
10         size = s.size();
11         a.resize(size);
12         a[0][0] = a[0][1] = s[0];
13         for (int i = 1; i < size; i++) {
14             a[i][0] = (a[i - 1][0] * base[0] + s[i]) % mod[0];
15             a[i][1] = (a[i - 1][1] * base[1] + s[i]) % mod[1];
16         }
17     }
18     array<int, 2> get(int l, int r) const {
19         if (l == 0) return a[r];
20         auto getone = [&](bool f) {
21             int x = (a[r][f] - 1ll * a[l - 1][f] * pow_base[r - l + 1][f]) % mod[f];
22             if (x < 0) x += mod[f];
23             return x;
24         };
25         return {getone(0), getone(1)};
26     }
27 };
28
29 auto _ = []() {
30     pow_base[0][0] = pow_base[0][1] = 1;
31     for (int i = 1; i <= N; i++) {
32         pow_base[i][0] = pow_base[i - 1][0] * base[0] % mod[0];
33         pow_base[i][1] = pow_base[i - 1][1] * base[1] % mod[1];
34     }
35     return true;
36 }();

```

3.3 manacher

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


4 数学

4.1 扩展欧几里得

需保证 $a, b \geq 0$

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

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

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

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

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

```

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

```

4.2 线性代数

4.2.1 向量公约数

```

1 // 将这两个向量组转化为b.y=0的形式
2 array<vec, 2> gcd(vec a, vec b) {
3     while (b.y != 0) {
4         int t = a.y / b.y;
5         a = a - b * t;
6         swap(a, b);
7     }
8     return {a, b};
9 }
10
11 array<vec, 2> gcd(array<vec, 2> g, vec a) {
12     auto [b, c] = gcd(g[0], a);
13     g[0] = b;
14     g[1] = vec(gcd(g[1].x, c.x), 0);
15     if (g[1].x != 0) g[0].x %= g[1].x;
16     return g;
17 }

```

4.3 筛法

primes

```

1 constexpr int N = 1e7;
2 bitset<N + 1> ispr;
3 vector<int> primes;
4 bool _ = []() {
5     ispr.set();
6     ispr[0] = ispr[1] = 0;
7     for (int i = 2; i <= N; i++) {
8         if (!ispr[i]) continue;
9         primes.push_back(i);
10        for (int j = 2 * i; j <= N; j += i) ispr[j] = 0;
11    }
12    return 1;
13 }();

```

 φ

```

1 constexpr int N = 1e7;
2 array<int, N + 1> phi;
3 auto _ = []() {
4     iota(phi.begin() + 1, phi.end(), 1);
5     for (int i = 2; i <= N; i++) {
6         if (phi[i] == i)
7             for (int j = i; j <= N; j += i) phi[j] = phi[j] / i * (i - 1);
8     }
9     return true;
10 }();

```

 μ

```

1 constexpr int N = 1e7;
2 bitset<N + 1> ispr;
3 array<int, N + 1> mu;
4 auto _ = []() {
5     mu.fill(1);
6     ispr.set();
7     mu[0] = ispr[0] = ispr[1] = 0;
8     for (int i = 2; i <= N; i++) {
9         if (!ispr[i]) continue;
10        mu[i] = -1;
11        for (int j = 2 * i; j <= N; j += i) {
12            ispr[j] = 0;
13            if (j / i % i == 0) mu[j] = 0;
14            else mu[j] *= -1;
15        }
16    }
17    return true;
18 }();

```

prime φ

```

1 constexpr int N = 1e7;
2 bitset<N + 1> ispr;
3 array<int, N + 1> phi;
4 vector<int> primes;
5 bool _ = []() {
6     ispr.set();

```

```

7   ispr[0] = ispr[1] = 0;
8   iota(phi.begin() + 1, phi.end(), 1);
9   for (int i = 2; i <= N; i++) {
10      if (!ispr[i]) continue;
11      phi[i] = i - 1;
12      primes.push_back(i);
13      for (int j = 2 * i; j <= N; j += i) {
14         ispr[j] = 0;
15         phi[j] = phi[j] / i * (i - 1);
16      }
17   }
18   return 1;
19 }();

```

prime μ

```

1  constexpr int N = 1e7;
2  bitset<N + 1> ispr;
3  array<int, N + 1> mu;
4  vector<int> primes;
5  bool _ = []() {
6     mu.fill(1);
7     ispr.set();
8     mu[0] = ispr[0] = ispr[1] = 0;
9     for (int i = 2; i <= N; i++) {
10        if (!ispr[i]) continue;
11        mu[i] = -1;
12        primes.push_back(i);
13        for (int j = 2 * i; j <= N; j += i) {
14           ispr[j] = 0;
15           if (j / i % i == 0) mu[j] = 0;
16           else mu[j] *= -1;
17        }
18    }
19    return 1;
20 }();

```

prime $\mu \varphi$

```

1  constexpr int N = 1e7;
2  bitset<N + 1> ispr;
3  array<int, N + 1> mu, phi;
4  vector<int> primes;
5  bool _ = []() {
6     mu.fill(1);
7     ispr.set();
8     mu[0] = ispr[0] = ispr[1] = 0;
9     iota(phi.begin() + 1, phi.end(), 1);
10    for (int i = 2; i <= N; i++) {
11       if (!ispr[i]) continue;
12       mu[i] = -1;
13       phi[i] = i - 1;
14       primes.push_back(i);
15       for (int j = 2 * i; j <= N; j += i) {
16          ispr[j] = 0;
17          if (j / i % i == 0) mu[j] = 0;
18          else mu[j] *= -1;
19          phi[j] = phi[j] / i * (i - 1);
20       }

```

```

21     }
22     return 1;
23 }();

1 constexpr int N = 1e7;
2 array<int, N + 1> minpr, mu, phi;
3 vector<int> primes;
4 bool _ = []() {
5     phi[1] = mu[1] = 1;
6     for (int i = 2; i <= N; i++) {
7         if (minpr[i] == 0) {
8             minpr[i] = i;
9             mu[i] = -1;
10            phi[i] = i - 1;
11            primes.push_back(i);
12        }
13        for (auto& j : primes) {
14            if (i * j > N) break;
15            minpr[i * j] = j;
16            if (j < minpr[i]) {
17                phi[i * j] = phi[i] * phi[j];
18                mu[i * j] = -mu[i];
19            } else {
20                mu[i * j] = 0;
21                phi[i * j] = phi[i] * j;
22                break;
23            }
24        }
25    }
26    return 1;
27 }();

```

4.4 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 }

```

4.5 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 }

```

4.6 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 (m < 0 || m > n) 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 }();

```

4.6.1 常用式子

- $\binom{n}{k} = \frac{n}{k} \binom{n-1}{k-1}$
- $\binom{n}{k} = \frac{n-k}{k} \binom{n}{k-1}$
- $\sum_{i=0}^n (-1)^i \binom{n}{i} = [n = 0]$
- $\sum_{i=0}^m \binom{n}{i} \binom{m}{i} = \binom{m+n}{m}$
- $\sum_{i=0}^n \binom{n}{i}^2 = \binom{2n}{n}$
- $\sum_{i=0}^n i \binom{n}{i} = n2^{n-1}$
- $\sum_{i=0}^n i^2 \binom{n}{i} = n(n+1)2^{n-2}$
- $\sum_{l=0}^n \binom{l}{k} = \binom{n+1}{k+1}$
- $\binom{n}{r} \binom{r}{k} = \binom{n}{k} \binom{n-k}{r-k}$
- $\sum_{i=0}^n \binom{n-i}{i} = F_{n+1}$, 其中 F 是斐波那契数列。
- $\sum_{i=0}^k \binom{n}{i} \binom{m}{k-i} = \binom{n+m}{k}$
- $\sum_{i=1}^n \binom{n}{i} \binom{n}{i-1} = \binom{2n}{n+1}$
- $m^n = \sum_{i=0}^m \{n\}_i \binom{m}{i} i!$

4.7 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 }

```

4.8 积性函数

4.8.1 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。

4.8.2 例子

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

一个加性函数。

4.9 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: $h = f * g$ 。

4.9.1 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 $h(1) \neq 0$ 。

4.9.2 例子

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

4.10 欧拉函数

```

1 constexpr int N = 1e6;
2 array<int, N + 1> phi;
3 auto _ = []() {
4     iota(phi.begin() + 1, phi.end(), 1);
5     for (int i = 2; i <= N; i++) {
6         if (phi[i] == i)
7             for (int j = i; j <= N; j += i) phi[j] = phi[j] / i * (i - 1);
8     }
9     return true;
10 }();

```

4.11 莫比乌斯反演

4.11.1 莫比乌斯函数性质

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

```

1 constexpr int N = 1e6;
2 array<int, N + 1> miu;
3 array<bool, N + 1> ispr;
4
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 }();

```


4.11.2 莫比乌斯变换/反演

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

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

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

4.12 杜教筛

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

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

可以构造恰当的数论函数 g 使得:

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

4.12.1 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 }

```

4.13 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 = 111 * res * a % mod;
9     a = 111 * 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] = 111 * 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 = 111 * 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] = 111 * 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 + 111 * 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] = 111 * 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] = 111 * 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 { return (derivation() * inv(n)).integral().modXn(n); }
180 // 需要保证首项为 0
181 poly exp(int n) const {

```

```

182     poly x(1);
183     int k = 1;
184     while (k < n) {
185         k *= 2;
186         x = (x * (poly(1) - x.log(k) + modXn(k))).modXn(k);
187     }
188     return x.modXn(n);
189 }
190 // 需要保证首项为 1, 开任意次方可以先 ln 再 exp 实现。
191 poly sqrt(int n) const {
192     poly x(1);
193     int k = 1;
194     while (k < n) {
195         k *= 2;
196         x += modXn(k) * x.inv(k);
197         x = x.modXn(k) * inv2;
198     }
199     return x.modXn(n);
200 }
201 // 减法卷积, 也称转置卷积  $\{\rm MULT\}(F(x), G(x)) = \sum_{i \geq 0} (\sum_{j \geq 0} f_{i+j} g_j) x^i$ 
202 //  $0 \leq i+j \leq n$ 
203 poly mult(poly rhs) const {
204     if (rhs.size() == 0) return poly();
205     int n = rhs.size();
206     ::reverse(rhs.a.begin(), rhs.a.end());
207     return ((*this) * rhs).divXn(n - 1);
208 }
209 int eval(int x) {
210     int r = 0, t = 1;
211     for (int i = 0, n = size(); i < n; ++i) {
212         r = (r + 111 * a[i] * t) % mod;
213         t = 111 * t * x % mod;
214     }
215     return r;
216 }
217 // 多点求值新科技: https://jkloverdcoi.github.io/2020/08/04/转置原理及其应用/
218 // 模板例题: https://www.luogu.com.cn/problem/P5050
219 auto evals(vector<int> &x) const {
220     if (size() == 0) return vector(x.size(), 0);
221     int n = x.size();
222     vector ans(n, 0);
223     vector<poly> g(4 * n);
224     auto build = [&](auto self, int l, int r, int p) -> void {
225         if (r - l == 1) {
226             g[p] = poly({1, x[l] ? mod - x[l] : 0});
227         } else {
228             int m = (l + r) / 2;
229             self(self, l, m, 2 * p);
230             self(self, m, r, 2 * p + 1);
231             g[p] = g[2 * p] * g[2 * p + 1];
232         }
233     };
234     build(build, 0, n, 1);
235     auto solve = [&](auto self, int l, int r, int p, poly f) -> void {
236         if (r - l == 1) {
237             ans[l] = f[0];
238         } else {
239             int m = (l + r) / 2;

```

```

240         self(self, 1, m, 2 * p, f.mulT(g[2 * p + 1]).modXn(m - 1));
241         self(self, m, r, 2 * p + 1, f.mulT(g[2 * p]).modXn(r - m));
242     }
243 };
244 solve(solve, 0, n, 1, mulT(g[1].inv(size())).modXn(n));
245 return ans;
246 }
247 }; // 全家桶测试: https://www.luogu.com.cn/training/3015#information
248
249 auto _ = []() {
250     inv[0] = inv[1] = 1;
251     for (int i = 2; i < inv.size(); i++) inv[i] = 1ll * (mod - mod / i) * inv[mod % i] % mod;
252     return true;
253 }();

```

4.14 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 $[x^n]e^{\sum_{i=1}^m \sum_{j=1}^{\infty} \frac{x^i j}{j}}$
✓	✓	✗	$f_{n-m,m}$
✗	✓	✓	$\sum_{i=1}^m g_{n,i}$ 或 $\sum_{i=1}^m \sum_{j=0}^i \frac{j^n}{j!} \frac{(-1)^{i-j}}{(i-j)!}$
✗	✓	✗	$g_{n,m} = g_{n-1,m-1} + m * g_{n-1,m}$ 或 $\frac{1}{m!} \sum_{i=0}^m (-1)^i \binom{m}{i} (m-i)^n$
✓	✗	✓	C_{n+m-1}^{m-1}
✓	✗	✗	C_{n-1}^{m-1}
✗	✗	✓	m^n
✗	✗	✗	$m! * g_{n,m}$ 或 $\sum_{i=0}^m (-1)^i \binom{m}{i} (m-i)^n$

4.14.1 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++) a[j] = (a[j] + inv[k]) % mod;
5     auto p = poly(a).exp(n + 1);
6     return (p.a[n] + mod) % mod;
7 }

```

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

4.14.2 球不同, 盒同, 可空

```

1 int solve(int n, int m) {
2     vector a(n + 1, 0);
3     vector b(n + 1, 0);
4     for (int i = 0; i <= n; i++) {
5         a[i] = ifac[i];
6         if (i & 1) a[i] = -a[i];
7         b[i] = 1ll * power(i, n) * ifac[i] % mod;
8     }
9     auto p = poly(a) * poly(b);
10    int ans = 0;
11    for (int i = 1; i <= min(n, m); i++) ans = (ans + p.a[i]) % mod;
12    return (ans + mod) % mod;
13 }

```

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

4.14.3 球同, 盒不同, 可空

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

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 $\sum_{i=0}^m (-1)^i f(i)$ 。

4.14.4 球同, 盒不同, 不可空

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

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 $\sum_{i=0}^m (-1)^i f(i)$ 。

4.14.5 球不同, 盒不同, 可空

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

4.14.6 球不同, 盒不同, 不可空

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

4.15 线性基

```

1 // 线性基
2 struct basis {
3     int rnk = 0;
4     array<ull, 64> p{};
5
6     // 将x插入此线性基中
7     void insert(ull x) {

```

```

8     for (int i = 63; i >= 0; i--) {
9         if (!(x >> i & 1)) continue;
10        if (p[i] x ^= p[i];
11        else {
12            p[i] = x;
13            rnk++;
14            break;
15        }
16    }
17 }
18
19 // 将另一个线性基插入此线性基中
20 void insert(basis other) {
21     for (int i = 0; i <= 63; i++) {
22         if (!other.p[i]) continue;
23         insert(other.p[i]);
24     }
25 }
26
27 // 最大异或值
28 ull max_basis() {
29     ull res = 0;
30     for (int i = 63; i >= 0; i--)
31         if ((res ^ p[i]) > res) res ^= p[i];
32     return res;
33 }
34 };

```

4.16 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] = (res.mat[i][j] + mat[i][k] * other.mat[k][j] % mod) % mod;
14        return res;
15    }
16 };
17 Mat ksm(Mat a, ll b) {
18     assert(a.n == a.m);
19     Mat res(a.n, a.m);
20     for (int i = 0; i < res.n; i++) res.mat[i][i] = 1;
21     while (b) {
22         if (b & 1) res = res * a;
23         b >>= 1;
24         a = a * a;
25     }
26     return res;
27 }

```


5 计算几何

5.1 整数

```

1 constexpr double inf = 1e100;
2
3 // 向量
4 struct vec {
5     static bool cmp(const vec &a, const vec &b) { return tie(a.x, a.y) < tie(b.x, b.y); }
6
7     ll x, y;
8     vec() : x(0), y(0) {}
9     vec(ll _x, ll _y) : x(_x), y(_y) {}
10
11     vec rotleft() const { return {-y, x}; }
12     vec rotright() const { return {y, -x}; }
13
14     // 模
15     ll len2() const { return x * x + y * y; }
16     double len() const { return sqrt(x * x + y * y); }
17
18     // 是否在上半轴
19     bool up() const { return y > 0 || y == 0 && x >= 0; }
20
21     bool operator==(const vec &b) const { return tie(x, y) == tie(b.x, b.y); }
22     // 极角排序
23     bool operator<(const vec &b) const {
24         if (up() != b.up()) return up() > b.up();
25         ll tmp = (*this) ^ b;
26         return tmp ? tmp > 0 : cmp(*this, b);
27     }
28
29     vec operator+(const vec &b) const { return {x + b.x, y + b.y}; }
30     vec operator-() const { return {-x, -y}; }
31     vec operator-(const vec &b) const { return -b + (*this); }
32     vec operator*(ll b) const { return {x * b, y * b}; }
33     ll operator*(const vec &b) const { return x * b.x + y * b.y; }
34
35     // 叉积 结果大于0, a到b为逆时针, 小于0, a到b顺时针,
36     // 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
37     ll operator^(const vec &b) const { return x * b.y - y * b.x; }
38
39     friend istream &operator>>(istream &in, vec &data) {
40         in >> data.x >> data.y;
41         return in;
42     }
43     friend ostream &operator<<(ostream &out, const vec &data) {
44         out << fixed << setprecision(6);
45         out << data.x << " " << data.y;
46         return out;
47     }
48 };
49
50 ll cross(const vec &a, const vec &b, const vec &c) { return (a - c) ^ (b - c); }
51
52 // 多边形的面积a
53 double polygon_area(vector<vec> &p) {
54     ll area = 0;

```

```

55     for (int i = 1; i < p.size(); i++) area += p[i - 1] ^ p[i];
56     area += p.back() ^ p[0];
57     return abs(area / 2.0);
58 }
59
60 // 多边形的周长
61 double polygon_len(vector<vec> &p) {
62     double len = 0;
63     for (int i = 1; i < p.size(); i++) len += (p[i - 1] - p[i]).len();
64     len += (p.back() - p[0]).len();
65     return len;
66 }
67
68 // 以整点为顶点的线段上的整点个数
69 ll count(const vec &a, const vec &b) {
70     vec c = a - b;
71     return gcd(abs(c.x), abs(c.y)) + 1;
72 }
73
74 // 以整点为顶点的多边形边上整点个数
75 ll count(vector<vec> &p) {
76     ll cnt = 0;
77     for (int i = 1; i < p.size(); i++) cnt += count(p[i - 1], p[i]);
78     cnt += count(p.back(), p[0]);
79     return cnt - p.size();
80 }
81
82 // 判断点是否在凸包内，凸包必须为逆时针顺序
83 bool in_polygon(const vec &a, vector<vec> &p) {
84     int n = p.size();
85     if (n == 0) return 0;
86     if (n == 1) return a == p[0];
87     if (n == 2) 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 = lower_bound(p.begin() + 2, p.end() - 1, a - p[0], cmp) - p.begin() - 1;
91     return cross(p[(i + 1) % n], a, p[i]) >= 0;
92 }
93
94 // 凸包直径的两个端点
95 auto polygon_dia(vector<vec> &p) {
96     int n = p.size();
97     array<vec, 2> res{};
98     if (n == 1) return res;
99     if (n == 2) return res = {p[0], p[1]};
100     ll mx = 0;
101     for (int i = 0, j = 2; i < n; i++) {
102         while (abs(cross(p[i], p[(i + 1) % n], p[j])) <=
103             abs(cross(p[i], p[(i + 1) % n], p[(j + 1) % n])))
104             j = (j + 1) % n;
105         ll tmp = (p[i] - p[j]).len2();
106         if (tmp > mx) {
107             mx = tmp;
108             res = {p[i], p[j]};
109         }
110         tmp = (p[(i + 1) % n] - p[j]).len2();
111         if (tmp > mx) {
112             mx = tmp;

```

```

113         res = {p[(i + 1) % n], p[j]};
114     }
115 }
116 return res;
117 }
118
119 // 凸包
120 auto convex_hull(vector<vec> &p) {
121     sort(p.begin(), p.end(), vec::cmp);
122     int n = p.size();
123     vector sta(n + 1, 0);
124     vector v(n, false);
125     int tp = -1;
126     sta[++tp] = 0;
127     auto update = [&](int lim, int i) {
128         while (tp > lim && cross(p[i], p[sta[tp]], p[sta[tp] - 1]) >= 0) v[sta[tp--]] = 0;
129         sta[++tp] = i;
130         v[i] = 1;
131     };
132     for (int i = 1; i < n; i++) update(0, i);
133     int cnt = tp;
134     for (int i = n - 1; i >= 0; i--) {
135         if (v[i]) continue;
136         update(cnt, i);
137     }
138     vector<vec> res(tp);
139     for (int i = 0; i < tp; i++) res[i] = p[sta[i]];
140     return res;
141 }
142
143 // 闵可夫斯基和，两个点集的和构成一个凸包
144 auto minkowski(vector<vec> &a, vector<vec> &b) {
145     rotate(a.begin(), min_element(a.begin(), a.end(), vec::cmp), a.end());
146     rotate(b.begin(), min_element(b.begin(), b.end(), vec::cmp), b.end());
147     int n = a.size(), m = b.size();
148     vector<vec> c{a[0] + b[0]};
149     c.reserve(n + m);
150     int i = 0, j = 0;
151     while (i < n && j < m) {
152         vec x = a[(i + 1) % n] - a[i];
153         vec y = b[(j + 1) % m] - b[j];
154         c.push_back(c.back() + ((x ^ y) >= 0 ? (i++, x) : (j++, y)));
155     }
156     while (i + 1 < n) {
157         c.push_back(c.back() + a[(i + 1) % n] - a[i]);
158         i++;
159     }
160     while (j + 1 < m) {
161         c.push_back(c.back() + b[(j + 1) % m] - b[j]);
162         j++;
163     }
164     return c;
165 }
166
167 // 过凸多边形外一点求凸多边形的切线，返回切点下标
168 auto tangent(const vec &a, vector<vec> &p) {
169     int n = p.size();
170     int l = -1, r = -1;

```

```

171     for (int i = 0; i < n; i++) {
172         ll tmp1 = cross(p[i], p[(i - 1 + n) % n], a);
173         ll tmp2 = cross(p[i], p[(i + 1) % n], a);
174         if (l == -1 && tmp1 <= 0 && tmp2 <= 0) l = i;
175         else if (r == -1 && tmp1 >= 0 && tmp2 >= 0) r = i;
176     }
177     return array{l, r};
178 }
179
180 // 直线
181 struct line {
182     vec p, d;
183     line() {}
184     line(const vec &a, const vec &b) : p(a), d(b - a) {}
185 };
186
187 // 点到直线距离
188 double dis(const vec &a, const line &b) { return abs((b.p - a) ^ (b.p + b.d - a)) / b.d.len(); }
189
190 // 点在直线哪边, 大于0在左边, 等于0在线上, 小于0在右边
191 ll side_line(const vec &a, const line &b) { return b.d ^ (a - b.p); }
192
193 // 两直线是否垂直
194 bool perpen(const line &a, const line &b) { return a.d * b.d == 0; }
195
196 // 两直线是否平行
197 bool parallel(const line &a, const line &b) { return (a.d ^ b.d) == 0; }
198
199 // 点的垂线是否与线段有交点
200 bool perpen(const vec &a, const line &b) {
201     vec p(-b.d.y, b.d.x);
202     bool cross1 = (p ^ (b.p - a)) > 0;
203     bool cross2 = (p ^ (b.p + b.d - a)) > 0;
204     return cross1 != cross2;
205 }
206
207 // 点到线段距离
208 double dis_seg(const vec &a, const line &b) {
209     if (perpen(a, b)) return dis(a, b);
210     return min((b.p - a).len(), (b.p + b.d - a).len());
211 }
212
213 // 点到凸包距离
214 double dis(const vec &a, vector<vec> &p) {
215     double res = inf;
216     for (int i = 1; i < p.size(); i++) res = min(dis_seg(a, line(p[i - 1], p[i])), res);
217     res = min(dis_seg(a, line(p.back(), p[0] - p.back())), res);
218     return res;
219 }
220
221 // 两直线交点
222 vec intersection(ll A, ll B, ll C, ll D, ll E, ll F) {
223     return {(B * F - C * E) / (A * E - B * D), (C * D - A * F) / (A * E - B * D)};
224 }
225
226 // 两直线交点
227 vec intersection(const line &a, const line &b) {
228     return intersection(a.d.y, -a.d.x, a.d.x * a.p.y - a.d.y * a.p.x, b.d.y, -b.d.x,

```

```

229         b.d.x * b.p.y - b.d.y * b.p.x);
230     }

```

5.2 浮点数

```

1  using lf = double;
2
3  constexpr lf eps = 1e-8;
4  constexpr lf inf = 1e100;
5  const lf PI = acos(-1);
6
7  int sgn(lf a, lf b) {
8      lf c = a - b;
9      return c < -eps ? -1 : c > eps ? 1 : 0;
10 }
11
12 // 向量
13 struct vec {
14     static bool cmp(const vec &a, const vec &b) {
15         return sgn(a.x, b.x) ? a.x < b.x : sgn(a.y, b.y) < 0;
16     }
17
18     lf x, y;
19     vec() : x(0), y(0) {}
20     vec(lf _x, lf _y) : x(_x), y(_y) {
21         if (sgn(y, 0) == 0) y = 0;
22     }
23
24     // 模
25     lf len2() const { return x * x + y * y; }
26     lf len() const { return sqrt(x * x + y * y); }
27
28     // 与x轴正方向的夹角
29     lf angle() const {
30         lf angle = atan2(y, x);
31         if (angle < 0) angle += 2 * PI;
32         return angle;
33     }
34
35     // 逆时针旋转
36     vec rotate(const lf &theta) const {
37         lf sint = sin(theta);
38         lf cost = cos(theta);
39         return {x * cost - y * sint, x * sint + y * cost};
40     }
41
42     vec rotleft() const { return {-y, x}; }
43
44     vec rotright() const { return {y, -x}; }
45
46     vec e() const {
47         lf tmp = len();
48         return {x / tmp, y / tmp};
49     }
50
51     // 是否在上半轴
52     bool up() const { return sgn(y, 0) > 0 || sgn(y, 0) == 0 && sgn(x, 0) >= 0; }

```

```

53
54     bool operator==(const vec &other) const { return sgn(x, other.x) == 0 && sgn(y, other.y) == 0; }
55
56     // 极角排序
57     bool operator<(const vec &b) const {
58         if (up() != b.up()) return up() > b.up();
59         if tmp = (*this) ^ b;
60         return sgn(tmp, 0) ? tmp > 0 : cmp(*this, b);
61     }
62
63     vec operator+(const vec &b) const { return {x + b.x, y + b.y}; }
64     vec operator-() const { return {-x, -y}; }
65     vec operator-(const vec &b) const { return -b + (*this); }
66     vec operator*(lf b) const { return {x * b, y * b}; }
67     vec operator/(lf b) const { return {x / b, y / b}; }
68     lf operator*(const vec &b) const { return x * b.x + y * b.y; }
69
70     // 叉积 结果大于0, a到b为逆时针, 小于0, a到b顺时针,
71     // 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
72     lf operator^(const vec &b) const { return x * b.y - y * b.x; }
73
74     friend istream &operator>>(istream &in, vec &data) {
75         in >> data.x >> data.y;
76         return in;
77     }
78     friend ostream &operator<<(ostream &out, const vec &data) {
79         out << fixed << setprecision(6);
80         out << data.x << " " << data.y;
81         return out;
82     }
83 };
84
85 lf cross(const vec &a, const vec &b, const vec &c) { return (a - c) ^ (b - c); }
86
87 lf angle(const vec &a, const vec &b) { return atan2(abs(a ^ b), a * b); }
88
89 // 多边形的面积
90 lf polygon_area(vector<vec> &p) {
91     lf area = 0;
92     for (int i = 1; i < p.size(); i++) area += p[i - 1] ^ p[i];
93     area += p.back() ^ p[0];
94     return abs(area / 2.0);
95 }
96
97 // 多边形的周长
98 lf polygon_len(vector<vec> &p) {
99     lf len = 0;
100     for (int i = 1; i < p.size(); i++) len += (p[i - 1] - p[i]).len();
101     len += (p.back() - p[0]).len();
102     return len;
103 }
104
105 // 判断点是否在凸包内, 凸包必须为逆时针顺序
106 bool in_polygon(const vec &a, vector<vec> &p) {
107     int n = p.size();
108     if (n == 0) return 0;
109     if (n == 1) return a == p[0];
110     if (n == 2) return sgn(cross(a, p[1], p[0]), 0) == 0 && sgn((p[0] - a) * (p[1] - a), 0) <= 0;

```

```

111     if (sgn(cross(a, p[1], p[0]), 0) > 0 || sgn(cross(p.back(), a, p[0]), 0) > 0) return 0;
112     auto cmp = [&](vec &x, const vec &y) { return sgn((x - p[0]) ^ y, 0) >= 0; };
113     int i = lower_bound(p.begin() + 2, p.end() - 1, a - p[0], cmp) - p.begin() - 1;
114     return sgn(cross(p[(i + 1) % n], a, p[i]), 0) >= 0;
115 }
116
117 // 凸包直径的两个端点
118 auto polygon_dia(vector<vec> &p) {
119     int n = p.size();
120     array<vec, 2> res{};
121     if (n == 1) return res;
122     if (n == 2) return res = {p[0], p[1]};
123     lf mx = 0;
124     for (int i = 0, j = 2; i < n; i++) {
125         while (sgn(abs(cross(p[i], p[(i + 1) % n], p[j])),
126             abs(cross(p[i], p[(i + 1) % n], p[(j + 1) % n])) <= 0)
127             j = (j + 1) % n;
128         lf tmp = (p[i] - p[j]).len();
129         if (tmp > mx) {
130             mx = tmp;
131             res = {p[i], p[j]};
132         }
133         tmp = (p[(i + 1) % n] - p[j]).len();
134         if (tmp > mx) {
135             mx = tmp;
136             res = {p[(i + 1) % n], p[j]};
137         }
138     }
139     return res;
140 }
141
142 // 凸包
143 auto convex_hull(vector<vec> &p) {
144     sort(p.begin(), p.end(), vec::cmp);
145     int n = p.size();
146     vector sta(n + 1, 0);
147     vector v(n, false);
148     int tp = -1;
149     sta[++tp] = 0;
150     auto update = [&](int lim, int i) {
151         while (tp > lim && sgn(cross(p[i], p[sta[tp]], p[sta[tp - 1]]), 0) >= 0) v[sta[tp--]] = 0;
152         sta[++tp] = i;
153         v[i] = 1;
154     };
155     for (int i = 1; i < n; i++) update(0, i);
156     int cnt = tp;
157     for (int i = n - 1; i >= 0; i--) {
158         if (v[i]) continue;
159         update(cnt, i);
160     }
161     vector<vec> res(tp);
162     for (int i = 0; i < tp; i++) res[i] = p[sta[i]];
163     return res;
164 }
165
166 // 闵可夫斯基和，两个点集的和构成一个凸包
167 auto minkowski(vector<vec> &a, vector<vec> &b) {
168     rotate(a.begin(), min_element(a.begin(), a.end(), vec::cmp), a.end());

```

```

169 rotate(b.begin(), min_element(b.begin(), b.end(), vec::cmp), b.end());
170 int n = a.size(), m = b.size();
171 vector<vec> c{a[0] + b[0]};
172 c.reserve(n + m);
173 int i = 0, j = 0;
174 while (i < n && j < m) {
175     vec x = a[(i + 1) % n] - a[i];
176     vec y = b[(j + 1) % m] - b[j];
177     c.push_back(c.back() + (sgn(x ^ y, 0) >= 0 ? (i++, x) : (j++, y)));
178 }
179 while (i + 1 < n) {
180     c.push_back(c.back() + a[(i + 1) % n] - a[i]);
181     i++;
182 }
183 while (j + 1 < m) {
184     c.push_back(c.back() + b[(j + 1) % m] - b[j]);
185     j++;
186 }
187 return c;
188 }
189
190 // 过凸多边形外一点求凸多边形的切线, 返回切点下标
191 auto tangent(const vec &a, vector<vec> &p) {
192     int n = p.size();
193     int l = -1, r = -1;
194     for (int i = 0; i < n; i++) {
195         lf tmp1 = cross(p[i], p[(i - 1 + n) % n], a);
196         lf tmp2 = cross(p[i], p[(i + 1) % n], a);
197         if (l == -1 && sgn(tmp1, 0) <= 0 && sgn(tmp2, 0) <= 0) l = i;
198         else if (r == -1 && sgn(tmp1, 0) >= 0 && sgn(tmp2, 0) >= 0) r = i;
199     }
200     return array{l, r};
201 }
202
203 // 直线
204 struct line {
205     vec p, d;
206     line() {}
207     line(const vec &a, const vec &b) : p(a), d(b - a) {}
208 };
209
210 // 点到直线距离
211 lf dis(const vec &a, const line &b) { return abs((b.p - a) ^ (b.p + b.d - a)) / b.d.len(); }
212
213 // 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 int side_line(const vec &a, const line &b) { return sgn(b.d ^ (a - b.p), 0); }
215
216 // 两直线是否垂直
217 bool perpen(const line &a, const line &b) { return sgn(a.d * b.d, 0) == 0; }
218
219 // 两直线是否平行
220 bool parallel(const line &a, const line &b) { return sgn(a.d ^ b.d, 0) == 0; }
221
222 // 点的垂线是否与线段有交点
223 bool perpen(const vec &a, const line &b) {
224     vec p(-b.d.y, b.d.x);
225     bool cross1 = sgn(p ^ (b.p - a), 0) > 0;
226     bool cross2 = sgn(p ^ (b.p + b.d - a), 0) > 0;

```



```

227     return cross1 != cross2;
228 }
229
230 // 点到线段距离
231 lf dis_seg(const vec &a, const line &b) {
232     if (perpen(a, b)) return dis(a, b);
233     return min((b.p - a).len(), (b.p + b.d - a).len());
234 }
235
236 // 点到凸包距离
237 lf dis(const vec &a, vector<vec> &p) {
238     lf res = inf;
239     for (int i = 1; i < p.size(); i++) res = min(dis_seg(a, line(p[i - 1], p[i] - p[i - 1])), res);
240     res = min(dis_seg(a, line(p.back(), p[0] - p.back())), res);
241     return res;
242 }
243
244 // 两直线交点
245 vec intersection(lf A, lf B, lf C, lf D, lf E, lf F) {
246     return {(B * F - C * E) / (A * E - B * D), (C * D - A * F) / (A * E - B * D)};
247 }
248
249 // 两直线交点
250 vec intersection(const line &a, const line &b) {
251     return intersection(a.d.y, -a.d.x, a.d.x * a.p.y - a.d.y * a.p.x, b.d.y, -b.d.x,
252         b.d.x * b.p.y - b.d.y * b.p.x);
253 }
254
255 struct circle {
256     vec o;
257     lf r;
258     circle(const vec &o, lf _r) : o(o), r(_r){};
259     circle(const vec &a, const vec &b, const vec &c) {
260         line u((a + b) / 2, (a + b) / 2 + (b - a).rotleft());
261         line v((b + c) / 2, (b + c) / 2 + (c - b).rotleft());
262         o = intersection(u, v);
263         r = (o - a).len();
264     }
265     // 内切圆
266     circle(const vec &a, const vec &b, const vec &c, bool t) {
267         line u, v;
268         double m = atan2(b.y - a.y, b.x - a.x), n = atan2(c.y - a.y, c.x - a.x);
269         u.p = a;
270         u.d = vec(cos((n + m) / 2), sin((n + m) / 2));
271         v.p = b;
272         m = atan2(a.y - b.y, a.x - b.x), n = atan2(c.y - b.y, c.x - b.x);
273         v.d = vec(cos((n + m) / 2), sin((n + m) / 2));
274         o = intersection(u, v);
275         r = dis_seg(o, line(a, b));
276     }
277
278     // 点与圆的关系 -1在圆内, 0在圆上, 1在圆外
279     int relation(const vec &a) const { return sgn((a - o).len(), r); }
280
281     // 圆与圆的关系 -3包含, -2内切, -1相交, 0外切, 1相离
282     int relation(const circle &a) const {
283         lf l = (a.o - o).len();
284         if (sgn(l, abs(r - a.r)) < 0) return -3;

```

```

285     if (sgn(1, abs(r - a.r)) == 0) return -2;
286     if (sgn(1, abs(r + a.r)) < 0) return -1;
287     if (sgn(1, abs(r + a.r)) == 0) return 0;
288     return 1;
289 }
290
291 lf area() { return PI * r * r; }
292 };
293
294 // 圆与圆交点
295 auto intersection(const circle &a, const circle &b) {
296     int rel = a.relation(b);
297     vector<vec> res;
298     if (rel == -3 || rel == 1) return res;
299     vec o = b.o - a.o;
300     lf l = (o.len2() + a.r * a.r - b.r * b.r) / (2 * o.len());
301     lf h = sqrt(a.r * a.r - l * l);
302     o = o.e();
303     vec tmp = a.o + o * l;
304     if (rel == -2 || rel == 0) res.push_back(tmp);
305     else {
306         res.push_back(tmp + o.rotleft() * h);
307         res.push_back(tmp + o.rotright() * h);
308     }
309     return res;
310 }
311
312 // 圆与直线交点
313 auto intersection(const circle &c, const line &l) {
314     lf d = dis(c.o, l);
315     vector<vec> res;
316     vec mid = l.p + l.d.e() * ((c.o - l.p) * l.d / l.d.len());
317     if (sgn(d, c.r) == 0) res.push_back(mid);
318     else if (sgn(d, c.r) < 0) {
319         d = sqrt(c.r * c.r - d * d);
320         res.push_back(mid + l.d.e() * d);
321         res.push_back(mid - l.d.e() * d);
322     }
323     return res;
324 }
325
326 // oab三角形与圆相交的面积
327 lf area(const circle &c, const vec &a, const vec &b) {
328     if (sgn(cross(a, b, c.o), 0) == 0) return 0;
329     vector<vec> p;
330     p.push_back(a);
331     line l(a, b);
332     auto tmp = intersection(c, l);
333     if (tmp.size() == 2) {
334         for (auto &i : tmp)
335             if (sgn((a - i) * (b - i), 0) < 0) p.push_back(i);
336     }
337     p.push_back(b);
338     if (p.size() == 4 && sgn((p[0] - p[1]) * (p[2] - p[1]), 0) > 0) swap(p[1], p[2]);
339     lf res = 0;
340     for (int i = 1; i < p.size(); i++)
341         if (c.relation(p[i - 1]) == 1 || c.relation(p[i]) == 1) {
342             lf ang = angle(p[i - 1] - c.o, p[i] - c.o);

```

```

343         res += c.r * c.r * ang / 2;
344     } else res += abs(cross(p[i - 1], p[i], c.o)) / 2.0;
345     return res;
346 }
347
348 // 多边形与圆相交的面积
349 lf area(vector<vec> &p, circle c) {
350     lf res = 0;
351     for (int i = 0; i < p.size(); i++) {
352         int j = i + 1 == p.size() ? 0 : i + 1;
353         if (sgn(cross(p[i], p[j], c.o), 0) <= 0) res += area(c, p[i], p[j]);
354         else res -= area(c, p[i], p[j]);
355     }
356     return abs(res);
357 }

```

三维

```

1  constexpr lf eps = 1e-8;
2
3  int sgn(lf a, lf b) {
4      lf c = a - b;
5      return c < -eps ? -1 : c < eps ? 0 : 1;
6  }
7
8  // 向量
9  struct vec3 {
10     lf x, y, z;
11     vec3() : x(0), y(0), z(0) {}
12     vec3(lf _x, lf _y, lf _z) : x(_x), y(_y), z(_z) {}
13
14     // 模
15     lf len2() const { return x * x + y * y + z * z; }
16     lf len() const { return hypot(x, y, z); }
17
18     bool operator==(const vec3 &b) const {
19         return sgn(x, b.x) == 0 && sgn(y, b.y) == 0 && sgn(z, b.z) == 0;
20     }
21     bool operator!=(const vec3 &b) const { return !(*this == b); }
22
23     vec3 operator+(const vec3 &b) const { return {x + b.x, y + b.y, z + b.z}; }
24     vec3 operator-(const vec3 &b) const { return {x - b.x, y - b.y, z - b.z}; }
25     vec3 operator*(lf b) const { return {b * x, b * y, b * z}; }
26     lf operator*(const vec3 &b) const { return x * b.x + y * b.y + z * b.z; }
27
28     // 叉积 结果大于0, a到b为逆时针, 小于0, a到b顺时针,
29     // 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
30     vec3 operator^(const vec3 &b) const {
31         return {y * b.z - z * b.y, z * b.x - x * b.z, x * b.y - y * b.x};
32     }
33
34     friend istream &operator>>(istream &in, vec3 &a) {
35         in >> a.x >> a.y >> a.z;
36         return in;
37     }
38
39     friend ostream &operator<<(ostream &out, const vec3 &a) {
40         out << fixed << setprecision(6);
41         out << a.x << " " << a.y << " " << a.z;

```

```

42     return out;
43 }
44 };
45
46 struct line3 {
47     vec3 p, d;
48     line3() {}
49     line3(const vec3 &a, const vec3 &b) : p(a), d(b - a) {}
50 };
51
52 struct plane {
53     vec3 p, d;
54     plane() {}
55     plane(const vec3 &a, const vec3 &b, const vec3 &c) : p(a) {
56         d = (b - a) ^ (c - a);
57         assert(d != vec3());
58     }
59 };
60
61 // 线面是否垂直
62 bool perpen(const line3 &a, const plane &b) { return (a.d ^ b.d) == vec3(); }
63
64 // 线面是否平行
65 bool parallel(const line3 &a, const plane &b) { return sgn(a.d * b.d, 0) == 0; }
66
67 // 线面交点
68 vec3 intersection(const line3 &a, const plane &b) {
69     assert(!parallel(a, b));
70     double t = (b.p - a.p) * b.d / (a.d * b.d);
71     return a.p + a.d * t;
72 }

```

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, int num) {
41     if (l <= tree[pos].l && tree[pos].r <= r) {
42         tree[pos].cnt += num;
43         tree[pos].len = tree[pos].cnt ? tree[pos].raw_len
44             : tree[pos].l == tree[pos].r ? 0
45             : tree[ls].len + tree[rs].len;
46         return;
47     }
48     if (l <= mid) update(ls, l, r, num);
49     if (r > mid) update(rs, l, r, num);
50     tree[pos].len = tree[pos].cnt ? tree[pos].raw_len : tree[ls].len + tree[rs].len;
51 };
52 build(1, 0, y_set.size() - 2);
53 auto find_pos = [&](ll num) {
54     return lower_bound(y_set.begin(), y_set.end(), num) - y_set.begin();
55 };
56 ll res = 0;
57 for (int i = 0; i < line.size() - 1; i++) {
58     update(1, find_pos(line[i].y_down), find_pos(line[i].y_up) - 1, line[i].pd);
59     res += (line[i + 1].x - line[i].x) * tree[1].len;
60 }
61 return res;
62 }

```

6 杂项

6.1 快读

```

1 namespace IO {
2 constexpr int N = (1 << 20) + 1;
3 char Buffer[N];
4 int p = N;
5
6 char& get() {
7     if (p == N) {
8         fread(Buffer, 1, N, stdin);
9         p = 0;
10    }
11    return Buffer[p++];
12 }
13
14 template <typename T = int>
15 T read() {
16     T x = 0;
17     bool f = 1;
18     char c = get();
19     while (!isdigit(c)) {
20         f = c != '-';
21         c = get();
22     }
23     while (isdigit(c)) {
24         x = x * 10 + c - '0';
25         c = get();
26     }
27     return f ? x : -x;
28 }
29 } // namespace IO
30 using IO::read;

```

6.2 高精度

```

1 struct bignum {
2     string num;
3
4     bignum() : num("0") {}
5     bignum(const string& num) : num(num) { reverse(this->num.begin(), this->num.end()); }
6     bignum(ll num) : num(to_string(num)) { reverse(this->num.begin(), this->num.end()); }
7
8     bignum operator+(const bignum& other) {
9         bignum res;
10        res.num.pop_back();
11        res.num.reserve(max(num.size(), other.num.size()) + 1);
12        for (int i = 0, j = 0, x; i < num.size() || i < other.num.size() || j; i++) {
13            x = j;
14            j = 0;
15            if (i < num.size()) x += num[i] - '0';
16            if (i < other.num.size()) x += other.num[i] - '0';
17            if (x >= 10) j = 1, x -= 10;
18            res.num.push_back(x + '0');
19        }
20        res.num.capacity();

```

```

21     return res;
22 }
23
24 bignum operator*(const bignum& other) {
25     vector<int> res(num.size() + other.num.size() - 1, 0);
26     for (int i = 0; i < num.size(); i++)
27         for (int j = 0; j < other.num.size(); j++)
28             res[i + j] += (num[i] - '0') * (other.num[j] - '0');
29     int g = 0;
30     for (int i = 0; i < res.size(); i++) {
31         res[i] += g;
32         g = res[i] / 10;
33         res[i] %= 10;
34     }
35     while (g) {
36         res.push_back(g % 10);
37         g /= 10;
38     }
39     int lim = res.size();
40     while (lim > 1 && res[lim - 1] == 0) lim--;
41     bignum res2;
42     res2.num.resize(lim);
43     for (int i = 0; i < lim; i++) res2.num[i] = res[i] + '0';
44     return res2;
45 }
46
47 bool operator<(const bignum& other) {
48     if (num.size() == other.num.size())
49         for (int i = num.size() - 1; i >= 0; i--)
50             if (num[i] == other.num[i]) continue;
51             else return num[i] < other.num[i];
52     return num.size() < other.num.size();
53 }
54
55 friend istream& operator>>(istream& in, bignum& a) {
56     in >> a.num;
57     reverse(a.num.begin(), a.num.end());
58     return in;
59 }
60 friend ostream& operator<<(ostream& out, bignum a) {
61     reverse(a.num.begin(), a.num.end());
62     return out << a.num;
63 }
64 };

```

6.3 离散化

```

1  template <typename T>
2  struct Hash {
3      vector<int> S;
4      vector<T> a;
5      Hash(const vector<int>& b) : S(b) {
6          sort(S.begin(), S.end());
7          S.erase(unique(S.begin(), S.end()), S.end());
8          a = vector<T>(S.size());
9      }
10     T& operator[](int i) const {

```

```

11     auto pos = lower_bound(S.begin(), S.end(), i) - S.begin();
12     assert(pos != S.size() && S[pos] == i);
13     return a[pos];
14 }
15 };

```

6.4 模运算

```

1 constexpr int mod = 998244353;
2
3 template <typename T>
4 T power(T a, int b) {
5     T res = 1;
6     while (b) {
7         if (b & 1) res = res * a;
8         a = a * a;
9         b >>= 1;
10    }
11    return res;
12 }
13
14 struct modint {
15     int x;
16     modint(int _x = 0) : x(_x) {
17         if (x < 0) x += mod;
18         else if (x >= mod) x -= mod;
19     }
20     modint inv() const { return power(*this, mod - 2); }
21     modint operator+(const modint& b) { return x + b.x; }
22     modint operator-() const { return -x; }
23     modint operator-(const modint& b) { return -b + *this; }
24     modint operator*(const modint& b) { return (ll)x * b.x % mod; }
25     modint operator/(const modint& b) { return *this * b.inv(); }
26     friend istream& operator>>(istream& is, modint& other) {
27         ll _x;
28         is >> _x;
29         other = modint(_x);
30         return is;
31     }
32     friend ostream& operator<<(ostream& os, modint other) { return os << other.x; }
33 };

```

6.5 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) { return frac(a * other.b + other.a * b, b * other.b); }

```



```

13     frac operator-() const {
14         frac res = *this;
15         res.a = -res.a;
16         return res;
17     }
18     frac operator-(const frac& other) const { return -other + *this; }
19     frac operator*(const frac& other) const { return frac(a * other.a, b * other.b); }
20     frac operator/(const frac& other) const {
21         assert(other.a);
22         return *this * frac(other.b, other.a);
23     }
24     bool operator<(const frac& other) const { return (*this - other).a < 0; }
25     bool operator<=(const frac& other) const { return (*this - other).a <= 0; }
26     bool operator>=(const frac& other) const { return (*this - other).a >= 0; }
27     bool operator>(const frac& other) const { return (*this - other).a > 0; }
28     bool operator==(const frac& other) const { return a == other.a && b == other.b; }
29     bool operator!=(const frac& other) const { return !(*this == other); }
30 };

```

6.6 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{{'+', 2}, {'-', 2}, {'*', 1}, {'/', 1}, {'^', 0}};
21     stringstream ss(s1);
22     string s2, temp;
23     stack<char> op;
24     while (ss >> temp) {
25         if (isdigit(temp[0])) s2 += temp + ' ';
26         else if (temp[0] == '(') op.push('(');
27         else if (temp[0] == ')') {
28             while (op.top() != '(') {
29                 s2 += op.top();
30                 s2 += ' ';
31                 op.pop();
32             }
33             op.pop();
34         } else {
35             while (!op.empty() && op.top() != '(' &&
36                 (temp[0] != '^' && rank[op.top()] <= rank[temp[0]])) {

```

```

37         rank[op.top()] < rank[temp[0]])) {
38             s2 += op.top();
39             s2 += ' ';
40             op.pop();
41         }
42         op.push(temp[0]);
43     }
44 }
45 while (!op.empty()) {
46     s2 += op.top();
47     s2 += ' ';
48     op.pop();
49 }
50 return s2;
51 }
52
53 // 计算后缀表达式
54 int calc(const string& s) {
55     stack<int> num;
56     stringstream ss(s);
57     string temp;
58     while (ss >> temp) {
59         if (isdigit(temp[0])) num.push(stoi(temp));
60         else {
61             int b = num.top();
62             num.pop();
63             int a = num.top();
64             num.pop();
65             if (temp[0] == '+') a += b;
66             else 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 = ksm(a, b);
70             num.push(a);
71         }
72     }
73     return num.top();
74 }

```

6.7 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} < array<int, 3>{other.y, other.m, other.d};
8      }
9      Date(const string& s) {
10         stringstream ss(s);
11         char ch;
12         ss >> y >> ch >> m >> ch >> d;
13     }
14     int dis() const {
15         int yd = (y - 1) * 365 + (upper_bound(leap.begin(), leap.end(), y - 1) - leap.begin());
16         int md = pre[m - 1] + (m > 2 && (y % 4 == 0 && y % 100 != 0 || y % 400 == 0));

```

```

17     return yd + md + d;
18 }
19 int dis(const Date& other) const { return other.dis() - dis(); }
20 };
21 for (int i = 1; i <= 12; i++) pre[i] = pre[i - 1] + month[2];
22 for (int i = 1; i <= 1000000; i++)
23     if (i % 4 == 0 && i % 100 != 0 || i % 400 == 0) leap.push_back(i);

```

6.8 builtin 函数

如果是 long long 型，记得函数后多加个 ll。

- ctz, 从最低位连续的 0 的个数，如果传入 0 则行为未定义。
- clz, 从最高位连续的 0 的个数，如果传入 0 则行为未定义。
- popcount, 二进制 1 的个数。
- parity, 二进制 1 的个数奇偶性。

6.9 对拍

linux/Mac

```

1 #!/bin/bash
2
3 g++ $1 -o a -O2
4 g++ $2 -o b -O2
5 g++ random.cpp -o random -O2
6
7 cnt=0
8 while true; do
9     let cnt++
10    echo TEST:$cnt
11    ./random > in
12    ./a < in > out.a
13    ./b < in > out.b
14    if ! diff out.a out.b; then break; fi
15 done

```

windows

```

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

```

6.10 编译常用选项

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

6.11 开栈

不同的系统/编译器可能命令不一样

```
1 ulimit -s
2 -Wl,--stack=0x10000000
3 -Wl,-stack_size -Wl,0x10000000
4 -Wl,-z,stack-size=0x10000000
```

6.12 clang-format

转储配置

```
1 clang-format -style=Google -dump-config > ./clang-format
```

.clang-format

```
1 BasedOnStyle: Google
2 IndentWidth: 4
3 AllowShortIfStatementsOnASingleLine: AllIfsAndElse
4 ColumnLimit: 100
```