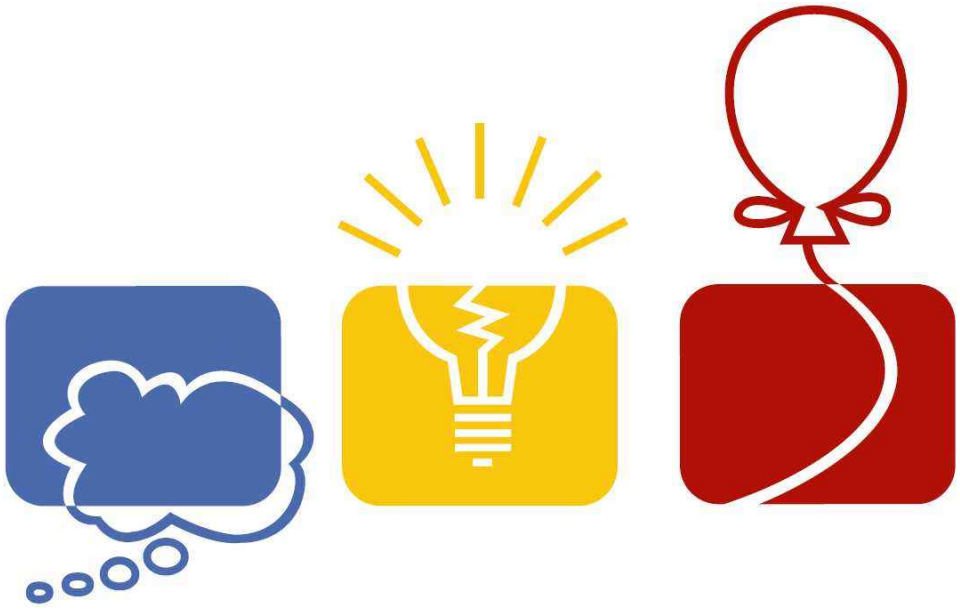

ICPC Templates

我们需要更深入浅出一些

Mine_King, Tx_Lcy, 369pai

洛谷科技

October 28, 2024

目录

Chapter 1

字符串

1.1 最小表示法

```cpp
// == Main ==
int i = 0, j = 1, k = 0;
while (k < n && i < n && j < n)
    if (a[(i + k) % n] == a[(j + k) % n]) k++;
    else {
        if (a[(i + k) % n] > a[(j + k) % n]) i = i + k + 1;
        else j = j + k + 1;
        if (i == j) i++;
        k = 0;
    }
ans = min(i, j);
```

1.2 Border 理论

1.2.1 关键结论

定理 1：对于一个字符串 s，若用 t 表示其最长的 Border，则有 $\mathcal{B}(s) = \mathcal{B}(t) \cup \{t\}$。

定理 2：一个字符串的 Border 与 Period 一一对应。具体地，$\text{pre}(s, i) \in \mathcal{B}(s) \iff |s| - i \in \mathcal{P}(s)$。

弱周期引理：

$$\forall p, q \in \mathcal{P}(s), p + q \leq |s| \implies \gcd(p, q) \in \mathcal{P}(s)$$

定理 3： 若字符串 t 是字符串 s 的前缀，且 $a \in \mathcal{P}(s), b \in \mathcal{P}(t), b \mid a, |t| \geq a$，且 b 是 t 的整周期，则有 $b \in \mathcal{P}(s)$。

周期引理：

$$\forall p, q \in \mathcal{P}(s), p + q - \gcd(p, q) \leq |s| \implies \gcd(p, q) \in \mathcal{P}(s)$$

定理 4： 对于文本串 s 和模式串 t，若 $|t| \geq \frac{|s|}{2}$，且 t 在 s 中至少成功匹配了 3 次，则每次匹配的位置形成一个等差数列，且公差为 t 的最小周期。

定理 5： 一个字符串 s 的所有长度不小于 $\frac{|s|}{2}$ 的 Border 的长度构成一个等差数列。

定理 6： 一个字符串的所有 Border 的长度排序后可以划分成 $\lceil \log_2 |s| \rceil$ 个连续段，使得每段都是一个等差数列。

定理 7： 回文串的回文前/后缀即为该串的 Border。

定理 8： 若回文串 s 有周期 p，则可以把 $\text{pre}(s, p)$ 划分成长度为 $|s| \bmod p$ 的前缀和长度为 $p - |s| \bmod p$ 的后缀，使得它们都是回文串。

定理 9： 若 t 是回文串 s 的最长 Border 且 $|t| \geq \frac{|s|}{2}$，则 t 在 s 中只能匹配 2 次。

定理 10： 对于任意一个字符串以及 $u, v \in \text{Ssuf}(s), |u| < |v|$，一定有 u 是 v 的 Border。

定理 11： 对于任意一个字符串 s 以及 $u, v \in \text{Ssuf}(s), |u| < |v|$，一定有 $2|u| \leq |v|$。

定理 12： $|\text{Ssuf}(s)| \leq \log_2 |s|$。

1.2.2 KMP

```
// == Main ==
// n is |s|, m is |t|.
for (int i = 1, j = 0; i <= n; i++) {
    while (j && t[j + 1] != s[i]) j = pi[j];
    if (t[j + 1] == s[i])
        if (++j == m) {
```

```
            // ...
            j = nxt[j];
        }
}
```

1.3 Z 函数

```
// == Main ==
// n is |s|.
for (int i = 2, j = 0; i <= n; i++) {
    if (i < j + z[j]) z[i] = min(z[i - j + 1], j + z[j] - i);
    while (i + z[i] <= n && s[i + z[i]] == s[1 + z[i]]) z[i]++;
    if (i + z[i] > j + z[j]) j = i;
}
```

1.4 Manacher

```
// == Main ==
// t is the original string, n is |t|.
string s = "^#";
for (char i : t) s.push_back(i), s.push_back('#');
s.push_back ('@');
for (int i = 1, j = 0; i <= 2 * n + 1; i++) {
    if (i <= j + p[j]) p[i] = min (p[2 * j - i], j + p[j] - i);
    while (s[i - p[i] - 1] == s[i + p[i] + 1]) p[i]++;
    if (i + p[i] > j + p[j]) j = i;
}
```

1.5 AC 自动机

```
// == Preparations ==
#include <queue>
// == Main ==
struct ACAM {
    int tot, fail[200005], delta[200005][26];

    void insert(string s, int id) {
        int now = 0;
        for (char c : s) {
            int v = c - 'a';
            if (!delta[now][v]) delta[now][v] = ++tot;
```

```
            now = delta[now][v];
        }
        return;
    }
    void build() {
        queue<int> q;
        for (int c = 0; c < 26; c++)
            if (delta[0][c]) q.push(delta[0][c]);
        while (!q.empty()) {
            int now = q.front();
            q.pop();
            for (int c = 0; c < 26; c++)
                if (delta[now][c]) fail[delta[now][c]] = delta[fail[now]][c],
                ↪  q.push(delta[now][c]);
                else delta[now][c] = delta[fail[now]][c];
        }
        return;
    }
} ac;
```

1.6 回文自动机 PAM

```
// == Main ==
struct PAM {
    int tot, delta[500005][26], len[500005], fail[500005], ans[500005];
    string s;
    int lst;

    PAM() {tot = 1; len[0] = 0; len[1] = -1; fail[0] = fail[1] = 1;}
    int getfail(int now, int i) {
        while (s[i - len[now] - 1] != s[i]) now = fail[now];
        return now;
    }
    void insert(int i) {
        int now = getfail(lst, i);
        if (!delta[now][s[i] - 'a']) {
            len[++tot] = len[now] + 2;
            fail[tot] = delta[getfail(fail[now], i)][s[i] - 'a'];
            delta[now][s[i] - 'a'] = tot;
            ans[tot] = ans[fail[tot]] + 1;
        }
        lst = delta[now][s[i] - 'a'];
        return;
    }
} p;
```

1.7 后缀自动机

1.7.1 普通 SAM

```cpp
// == Main ==
struct SAM {
    int tot, lst;
    int len[2000005], siz[2000005], link[2000005];
    int delta[2000005][26];

    SAM() {link[0] = -1;}
    void insert(char ch) {
        int c = ch - 'a', now = ++tot;
        len[now] = len[lst] + 1;
        siz[now] = 1;
        for (int p = lst; p != -1; p = link[p])
            if (!delta[p][c]) delta[p][c] = tot;
            else if (len[delta[p][c]] == len[p] + 1) {link[now] = delta[p][c];
            ↪ break;}
            else {
                int q = delta[p][c], v = ++tot;
                len[v] = len[p] + 1;
                memcpy(delta[v], delta[q], sizeof(delta[v]));
                link[v] = link[q], link[q] = v, link[now] = v;
                for (int i = p; delta[i][c] == q; i = link[i]) delta[i][c] = v;
                break;
            }
        lst = now;
        return ;
    }
} sam;
```

1.7.2 广义 SAM

注意自动机空间要开 Trie 的两倍。

```cpp
// == Main ==
struct GSAM {
    int tot;
    int delta[2000005][26], link[2000005], len[2000005];
    struct Trie {
        int tot, trie[1000005][26], st[1000005];

        void insert(string s) {
            int now = 0;
            for (char c : s) {
```

```
11                int id = c - 'a';
12                if (!trie[now][id]) trie[now][id] = ++tot;
13                now = trie[now][id];
14            }
15            return;
16        }
17    } tr;
18
19    GSAM() {link[0] = -1;}
20    int insert(int c, int lst) {
21        int now = ++tot;
22        len[now] = len[lst] + 1;
23        for (int p = lst; p != -1; p = link[p])
24            if (!delta[p][c]) delta[p][c] = now;
25            else if (len[delta[p][c]] == len[p] + 1) {link[now] = delta[p][c];
              ↪ break;}
26            else {
27                int q = delta[p][c], v = ++tot;
28                len[v] = len[p] + 1;
29                memcpy(delta[v], delta[q], sizeof(delta[v]));
30                link[v] = link[q], link[q] = v, link[now] = v;
31                for (int i = p; i != -1 && delta[i][c] == q; i = link[i]) delta[i][c]
                  ↪ = v;
32                break;
33            }
34        return now;
35    }
36    void build() {
37        queue<int> q;
38        tr.st[0] = 0;
39        q.push(0);
40        while (!q.empty()) {
41            int now = q.front();
42            q.pop();
43            for (int i = 0; i < 26; i++)
44                if (tr.trie[now][i])
45                    tr.st[tr.trie[now][i]] = insert(i, tr.st[now]),
                      ↪ q.push(tr.trie[now][i]);
46        }
47        return;
48    }
49 } gsam;
```

1.8 后缀排序

```
// == Preparations ==
int sa[2000005], rk[2000005], b[1000005], cp[2000005];
// == Main ==
for (int i = 1; i <= n; i++) b[rk[i] = s[i]]++;
for (int i = 1; i < 128; i++) b[i] += b[i - 1];
for (int i = n; i >= 1; i--) sa[b[rk[i]]--] = i;
memcpy(cp, rk, sizeof(cp));
for (int i = 1, j = 0; i <= n; i++)
    if (cp[sa[i]] == cp[sa[i - 1]]) rk[sa[i]] = j;
    else rk[sa[i]] = ++j;
for (int w = 1; w < n; w <<= 1) {
    memcpy(cp, sa, sizeof(cp));
    memset(b, 0, sizeof(b));
    for (int i = 1; i <= n; i++) b[rk[cp[i] + w]]++;
    for (int i = 1; i <= n; i++) b[i] += b[i - 1];
    for (int i = n; i >= 1; i--) sa[b[rk[cp[i] + w]]--] = cp[i];
    memcpy(cp, sa, sizeof(cp));
    memset(b, 0, sizeof(b));
    for (int i = 1; i <= n; i++) b[rk[cp[i]]]++;
    for (int i = 1; i <= n; i++) b[i] += b[i - 1];
    for (int i = n; i >= 1; i--) sa[b[rk[cp[i]]]--] = cp[i];
    memcpy(cp, rk, sizeof(cp));
    for (int i = 1, j = 0; i <= n; i++)
        if (cp[sa[i]] == cp[sa[i - 1]] && cp[sa[i] + w] == cp[sa[i - 1] + w])
        ↪ rk[sa[i]] = j;
        else rk[sa[i]] = ++j;
}
```

Chapter 2

数据结构

2.1 FHQ Treap

```
// == Preparations ==
#include <chrono>
#include <random>

mt19937 gen(chrono::system_clock::now().time_since_epoch().count());
// == Main ==
struct FHQ_Treap {
    struct node {
        int key, val, siz;
        node *ls, *rs;

        node() = default;
        node(int _key): key(_key), val(gen()), siz(1), ls(nullptr), rs(nullptr) {}
        void pushup() {
            siz = 1;
            if (ls) siz += ls->siz;
            if (rs) siz += rs->siz;
            return ;
        }
    };
    node *root;

    node *merge(node *u, node *v) {
        if (!u) return v;
        if (!v) return u;
        if (u->val < v->val) {u->rs = merge(u->rs, v); u->pushup(); return u;}
        else {v->ls = merge(u, v->ls); v->pushup(); return v;}
    }
    void split(node *now, int k, node *&u, node*&v) {
        if (!now) {u = v = nullptr; return ;}
        if (now->key < k) u = now, split(now->rs, k, u->rs, v);
```

```
32          else v = now, split(now->ls, k, u, v->ls);
33          now->pushup();
34          return ;
35      }
36      void insert(int k) {
37          node *u, *v;
38          split(root, k, u, v);
39          root = merge(u, merge(new node(k), v));
40          return ;
41      }
42      void erase(int k) {
43          node *u, *v, *w;
44          split(root, k, u, v), split(v, k + 1, v, w);
45          node *tmp = v;
46          v = merge(v->ls, v->rs);
47          delete tmp;
48          root = merge(u, merge(v, w));
49          return ;
50      }
51      int order_of_key(int k) {
52          node *u, *v;
53          split(root, k, u, v);
54          int ans = u ? u->siz + 1 : 1;
55          root = merge(u, v);
56          return ans;
57      }
58      int find_by_order(int k) {
59          node *now = root;
60          while (1) {
61              int lsiz = now->ls ? now->ls->siz : 0;
62              if (lsiz + 1 == k) return now->key;
63              else if (k <= lsiz) now = now->ls;
64              else k -= lsiz + 1, now = now->rs;
65          }
66      }
67      int find_prev(int k) {return find_by_order(order_of_key(k) - 1);}
68      int find_next(int k) {return find_by_order(order_of_key(k + 1));}
69  } tr;
```

2.2 K-D Tree

```
1  // == Main ==
2  template<const int Dim = 2>
3  struct KDTree {
4      using point = array<int, Dim>;
5      struct node {
```

```
6         point p, l, r;
7         int val, siz, sum;
8         node *ls, *rs;
9
10        node() = default;
11        node(point _p, int _val = 0):
12            p(_p), l(_p), r(_p), val(_val), siz(1), sum(_val), ls(nullptr),
              ↪ rs(nullptr) {}
13        void pushup() {
14            l = r = p, siz = 1, sum = val;
15            for (int i = 0; i < Dim; i++) {
16                if (ls) l[i] = min(l[i], ls->l[i]), r[i] = max(r[i], ls->r[i]);
17                if (rs) l[i] = min(l[i], rs->l[i]), r[i] = max(r[i], rs->r[i]);
18            }
19            if (ls) siz += ls->siz, sum += ls->sum;
20            if (rs) siz += rs->siz, sum += rs->sum;
21            return ;
22        }
23    };
24    vector<node *> root;
25    using itor = typename vector<node>::iterator;
26
27    node *build(itor l, itor r, int dim = 0) {
28        if (l == r) return nullptr;
29        int mid = (r - l) / 2;
30        nth_element(l, l + mid, r, [&dim](const node &x, const node &y) {return
              ↪ x.p[dim] < y.p[dim];});
31        node *now = new node(*(l + mid));
32        now->ls = build(l, l + mid, (dim + 1) % Dim);
33        now->rs = build(l + mid + 1, r, (dim + 1) % Dim);
34        now->pushup();
35        return now;
36    }
37    void getnode(node *now, vector<node> &vec) {
38        if (!now) return ;
39        vec.push_back(*now);
40        getnode(now->ls, vec), getnode(now->rs, vec);
41        delete now;
42        return ;
43    }
44    void insert(point p, int val) {
45        vector<node> tmp({node(p, val)});
46        while (!root.empty() && root.back()->siz == (int)tmp.size())
47            getnode(root.back(), tmp), root.pop_back();
48        sort(tmp.begin(), tmp.end(), [](const node &x, const node &y) {return x.p <
              ↪ y.p;});
49        vector<node> vec;
50        for (node i : tmp)
51            if (!vec.empty() && vec.back().p == i.p) vec.back().val += i.val;
```

```
52              else vec.push_back(i);
53          root.push_back(build(vec.begin(), vec.end()));
54          return ;
55      }
56      int query(point ll, point rr, node *now) {
57          if (!now) return 0;
58          int flag = 1;
59          for (int i = 0; i < Dim; i++)
60              if (now->r[i] < ll[i] || now->l[i] > rr[i]) return 0;
61              else flag &= ll[i] <= now->l[i] && now->r[i] <= rr[i];
62          if (flag) return now->sum;
63          flag = 1;
64          for (int i = 0; i < Dim; i++) flag &= ll[i] <= now->p[i] && now->p[i] <=
            ↪ rr[i];
65          return flag * now->val + query(ll, rr, now->ls) + query(ll, rr, now->rs);
66      }
67      int query(point ll, point rr) {
68          int ans = 0;
69          for (node *rt : root) ans += query(ll, rr, rt);
70          return ans;
71      }
72      ~KDTree() {
73          vector<node> tmp;
74          for (node *rt : root) getnode(rt, tmp);
75      }
76  };
```

2.3 Link Cut Tree

代码维护的是点权异或和。

```
1   // == Main ==
2   struct LinkCutTree {
3       int fa[100005], son[100005][2], siz[100005], swp[100005];
4       int val[100005], Xor[100005];
5
6       void pushup(int now) {
7           siz[now] = siz[son[now][0]] + siz[son[now][1]] + 1;
8           Xor[now] = Xor[son[now][0]] ^ val[now] ^ Xor[son[now][1]]; // 此处更新信息。
9           return;
10      }
11      void pushdown(int now) {
12          if(!swp[now]) return ;
13          swap(son[now][0], son[now][1]);
14          swp[son[now][0]] ^= 1, swp[son[now][1]] ^= 1;
15          swp[now] = 0;
```

```
16          // 此处将信息 pushdown
17          return;
18      }
19      int isRoot(int now) {return now != son[fa[now]][0] && now != son[fa[now]][1];}
20      int get(int now) {return now == son[fa[now]][1];}
21      void rotate(int x) {
22          int y = fa[x], z = fa[fa[x]], chk = get(x);
23          if (!isRoot(y)) son[z][get(y)] = x;
24          son[y][chk] = son[x][chk ^ 1], fa[son[x][chk ^ 1]] = y;
25          son[x][chk ^ 1] = y, fa[y] = x;
26          fa[x] = z;
27          pushup(y), pushup(x);
28          return;
29      }
30      void splay(int now) {
31          vector<int> stk;
32          stk.push_back(now);
33          for (int i = now; !isRoot(i); i = fa[i]) stk.push_back(fa[i]);
34          while (!stk.empty()) pushdown(stk.back()), stk.pop_back();
35          for (int f; f = fa[now], !isRoot(now); rotate(now))
36              if (!isRoot(f)) rotate(get(f) == get(now) ? f : now);
37          return;
38      }
39      void access(int now) { // 打通到根的链
40          for (int lst = 0; now; lst = now, now = fa[now]) splay(now), son[now][1] =
            ↪ lst, pushup(now);
41          return;
42      }
43      void makeRoot(int now) {access(now); splay(now); swp[now] ^= 1; return;} // 设置
        ↪ 根
44      void link(int u, int v) {makeRoot(u); fa[u] = v; return;} // 连接
45      void cut(int u, int v) {makeRoot(u); access(v); splay(v); son[v][0] = fa[u] = 0;
        ↪ return;} // 切割
46      int find(int now) { // 找根
47          access(now), splay(now);
48          pushdown(now);
49          while (son[now][0]) now = son[now][0], pushdown(now);
50          splay(now);
51          return now;
52      }
53      void split(int u, int v) {makeRoot(u); access(v); splay(u); return;} // 剖出 u ~
        ↪ v 的链
54      void update(int u, int _val) {split(u, u); val[u] = Xor[u] = _val; return ;} //
        ↪ 修改操作，split 后做就行了，此处为单点修改。
55      int query(int u, int v) {split(u, v); return Xor[u];} // 查询操作，split 后做就行
        ↪ 了。
56      int isConnected(int u, int v) {return find(u) == find(v);} // 查询两个点是否连通。
57  };
```

Chapter 3

数论

3.1 Miller Rabin 和 Pollard Rho

3.1.1 Miller Rabin

```cpp
// == Preparations ==
const int prime[] = {2, 3, 5, 7, 9, 11, 13, 17, 19, 23, 29, 31, 37};

long long power(long long a, long long b, long long mod) {
    long long ans = 1;
    while (b) {
        if (b & 1) ans = (__int128)ans * a % mod;
        a = (__int128)a * a % mod;
        b >>= 1;
    }
    return ans % mod;
}
// == Main ==
inline int Miller_Rabin(long long n) {
    if (n == 1) return 0;
    if (n == 2) return 1;
    if (n % 2 == 0) return 0;
    long long u = n - 1, t = 0;
    while (u % 2 == 0) u /= 2, t++;
    for (int i = 0; i < 12; i++) {
        if (prime[i] % n == 0) continue;
        long long x = power(prime[i] % n, u, n);
        if (x == 1) continue;
        int flag = 0;
        for (int j = 1; j <= t; j++) {
            if (x == n - 1) {flag = 1; break;}
            x = (__int128)x * x % n;
        }
        if (!flag) return 0;
```

```
    }
    return 1;
}
```

3.1.2 Pollard Rho

```
// == Preparations ==
#include <chrono>
#include <random>

mt19937_64 gen(chrono::system_clock::now().time_since_epoch().count());
/*
Miller Rabin
*/
// == Main ==
long long Pollard_Rho(long long n) {
    long long s = 0, t = 0, c = gen() % (n - 1) + 1;
    for (int goal = 1; ; goal <<= 1, s = t) {
        long long val = 1;
        for (int step = 1; step <= goal; step++) {
            t = ((__int128)t * t + c) % n;
            val = (__int128)val * abs(t - s) % n;
            if (!val) return n;
            if (step % 127 == 0) {
                long long d = __gcd(val, n);
                if (d > 1) return d;
            }
        }
        long long d = __gcd(val, n);
        if (d > 1) return d;
    }
}
void factor(long long n) {
    if (n < ans) return ;
    if (n == 1 || Miller_Rabin(n)) {
        // n = 1 或 n 是一个质因子。
        // ...
    }
    long long p;
    do p = Pollard_Rho(n);
    while (p == n);
    while (n % p == 0) n /= p;
    factor(n), factor(p);
    return ;
}
```

3.2 常用数论算法

3.2.1 exgcd

求出来的解满足 $|x| \leq b, |y| \leq a$。

```
// == Main ==
int exgcd(int a, int b, int &x, int &y) {
    if (b == 0) {x = 1; y = 0; return a;}
    int m = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return m;
}
```

3.2.2 CRT

没用。

```
// == Preparations ==
#include <vector>
// == Main ==
int CRT(vector<pair<int, int>> &a) {
    int M = 1;
    for (auto i : a) M *= i.second;
    int res = 0;
    for (auto i : a) {
        int t = inv(M / i.second, i.second);
        res = (res + (long long)i.first * (M / i.second) % M * t % M) % M;
    }
    return res;
}
```

3.2.3 exCRT

需保证 lcm 在 `long long` 范围内。

```
// == Preparations ==
long long exgcd(long long a, long long b, long long &x, long long &y);
// == Main ==
long long exCRT(vector<pair<long long, long long>> vec) {
    long long ans = vec[0].first, mod = vec[0].second;
    for (int i = 1; i < (int)vec.size(); i++) {
        long long a = mod, b = vec[i].second, c = vec[i].first - ans % b;
        long long x, y;
```

```
        long long g = exgcd(a, b, x, y);
        if (c % g != 0) return -1;
        b /= g;
        x = (__int128)x * (c / g) % b;
        ans += x * mod;
        mod *= b;
        ans = (ans % mod + mod) % mod;
    }
    return ans;
}
```

3.2.4 exLucas

```
// == Preparations ==
int power(int a, long long b, int mod);
int exgcd(int a, int b, int &x, int &y);
int CRT(vector<pair<int, int>> &a);
// == Main ==
int inv(int n, int p) {
    int x, y;
    exgcd(n, p, x, y);
    return (x % p + p) % p;
}
int fac(long long n, int p, int pk) {
    if (n == 0) return 1;
    int res = 1;
    for (int i = 1; i < pk; i++)
        if (i % p != 0) res = (long long)res * i % pk;
    res = power(res, n / pk, pk);
    for (int i = 1; i <= n % pk; i++)
        if (i % p != 0) res = (long long)res * i % pk;
    return (long long)res * fac(n / p, p, pk) % pk;
}
int C(long long n, long long m, int p, int pk) {
    long long x = n, y = m, z = n - m;
    int res = (long long)fac(x, p, pk) * inv(fac(y, p, pk), pk) % pk * inv(fac(z, p,
    ↪ pk), pk) % pk;
    long long e = 0;
    while (x) e += x / p, x /= p;
    while (y) e -= y / p, y /= p;
    while (z) e -= z / p, z /= p;
    return (long long)res * power(p, e, pk) % pk;
}
int exLucas(long long n, long long m, int p) {
    vector<pair<int, int>> a;
    for (int i = 2; i * i <= p; i++)
        if (p % i == 0) {
```

```
            int pk = 1;
            while (p % i == 0) pk *= i, p /= i;
            a.emplace_back(C(n, m, i, pk), pk);
        }
    if (p != 1) a.emplace_back(C(n, m, p, p), p);
    return CRT(a);
}
```

3.3 万能欧几里得算法

问题描述：
给出一个幺半群 $(S,\cdot)$ 和元素 $u,r\in S$，以及一条直线 $y=\frac{ax+b}{c}$。
画出平面中所有坐标为正整数的横线和竖线，维护一个 f，初值为单位元 e。
从原点出发，先向 y 轴正方向走直到到达直线与 y 的交点，然后沿直线走一直走到与 $x=n$ 的交点为止。
每当经过一条横线时，执行 $f\leftarrow fu$，经过一条竖线时执行 $f\leftarrow fr$。特别地，在 y 轴上行走时不考虑竖线，同时经过横线和竖线时先执行前者。
求最终的 f。记为 $\text{euclid}(a,b,c,n,u,r)$。
其中 $a,b\geq 0,n,c>0$。

做法：

$$\text{euclid}(a,b,c,n,u,r)=\begin{cases}r^n & m=0\\ u^{\lfloor\frac{b}{c}\rfloor}\cdot\text{euclid}(a\bmod c,b\bmod c,c,n,u,u^{\lfloor\frac{a}{c}\rfloor}r) & a\geq c\vee b\geq c\\ r^{\lfloor\frac{c-b-1}{a}\rfloor}u\cdot\text{euclid}(c,(c-b-1)\bmod a,a,m-1,r,u)\cdot r^{n-\lfloor\frac{cm-b-1}{a}\rfloor} & \text{otherwise}\end{cases}$$

设一次乘法的复杂度为 $O(T)$，则复杂度为 $O(T\log(a+c)\log(a+n+c))$。

```
// == Preparations ==
struct Node {
    // ...

    Node operator*(const Node &x) const {
        // ...
    }
};
// == Main ==
Node power(Node a, long long b) {
    Node ans = Node(/* 幺元 */);
    while (b) {
        if (b & 1) ans = ans * a;
        a = a * a;
```

```
        b >>= 1;
    }
    return ans;
}
Node Euclid(int a, int b, int c, long long n, Node r, Node u) {
    long long m = (a * n + b) / c;
    if (!m) return power(r, n);
    if (a >= c || b >= c)
        return power(u, b / c) * Euclid(a % c, b % c, c, n, power(u, a / c) * r, u);
    return power(r, (c - b - 1) / a) * u *
        Euclid(c, (c - b - 1) % a, a, m - 1, u, r) * power(r, n - (c * m - b - 1) /
        ↪  a);
}
```

3.4 离散对数

定义：
取有原根的正整数模数 m，设其一个原根为 g . 对满足 $(a,m)=1$ 的整数 a，我们知道必存在唯一的整数 $0 \le k < \varphi(m)$ 使得 $g^k \equiv a \pmod m$。
我们称这个 k 为以 g 为底，模 m 的离散对数，记作 $k = \operatorname{ind}_g a$。

离散对数问题即在模 p 意义下求解 $\log_a b$。这等价于求解离散对数方程

$$a^x \equiv b \pmod p$$

3.4.1 BSGS

运用了根号分治的思想：设块长为 M 且 $x = AM - B$，则有 $a^{AM} \equiv ba^B \pmod p$。
固定模数 p 和底数 a 时，预处理时间为 $\mathcal{O}(\frac{p}{M})$，单次询问 $\mathcal{O}(M)$
如果数据组数为 T，当 M 取 $\lceil \sqrt{\frac{p}{T}} \rceil$ 时取到最优复杂度。
以下代码输入要求 $a \perp p, 2 \le b, n < p < 2^{31}$，求得的是模意义下 $\log_a b$ 的最小的非负整数解。

```
// == Preparations ==
int a , p , m , t , sz , phi;
__gnu_pbds::gp_hash_table<int , int>mp;
int Qpow(int x , int p , int mod);
int Phi(int n);
int Inv(int a , int p){return Qpow(a , phi - 1 , p);}
// == Main ==
void Init()
{
    mp.clear();
    phi = Phi(p);
```

```
    sz = sqrt(phi / m) + 1;
    t = ((ll)phi + sz - 1) / sz;
    int as = Qpow(a , sz , p) , aa = 1;
    for(int i = 1 ; i <= t ; i++)
    {
        aa = (ll)aa * as % p;
        if(!mp[aa])mp[aa] = i;
    }
}
int BSGS(int b)
{
    int r = b , ans = p;
    for(int k = 0 ; k <= sz ; k++)
    {
        if(mp.find(r) != mp.end())
            ans = min((ll)ans , (ll)mp[r] * sz - k);
        r = (ll)r * a % p;
    }
    return ans >= p ? -1 : ans;
}
```

3.4.2 exBSGS

将方程化为如下形式，其中 $\frac{p}{D} \perp a$。

$$\frac{a^k}{D} \cdot a^{x-k} \equiv \frac{b}{D} \pmod{\frac{p}{D}}$$

然后使用 BSGS 求解，注意特判 $x \leq k$ 的情况。

以下代码输入要求 $1 \leq a, p, b \leq 10^9$，求得的是模意义下 $\log_a b$ 的最小的非负整数解。

```
// == Preparations ==
int a , p , mod , m , t , sz , phi , k , d , inv;
__gnu_pbds::gp_hash_table<int , int>mp , low;
int Qpow(int x , int p , int mod);
int Phi(int n);
int Inv(int a , int p);
int BSGS(int b);
// == Main ==
void Init()
{
    mp.clear() , low.clear();
    a %= p , mod = p , d = 1 , k = 0;
    int ad = 1 , ak = 1;
    for(int g = __gcd(a , p) ; g != 1 ; g = __gcd(a , p))
    {
        ak = (ll)ak * a % mod , k++;
```

```
        if(!low[ak])low[ak] = k;
        d *= g , p /= g , ad = ad * ll(a / g) % p;
    }
    phi = Phi(p) , inv = Inv(ad , p);
    sz = sqrt(phi / m) + 1;
    t = ((ll)phi + sz - 1) / sz;
    int as = Qpow(a , sz , p) , aa = 1;
    for(int i = 1 ; i <= t ; i++)
    {
        aa = (ll)aa * as % p;
        if(!mp[aa])mp[aa] = i;
    }
}
int exBSGS(int b)
{
    int bm = b % mod;
    if(mod == 1 || bm == 1)return 0;
    if(low.find(bm) != low.end())return low[bm];
    if(b % d)return -1;
    b = (ll)(b / d) * inv % p;
    int ans = BSGS(b);
    return ans == -1 ? -1 : ans + k;
}
```

3.5 原根

3.5.1 阶

定义：满足同余式 $a^n \equiv 1 \pmod{m}$ 的最小正整数 n 存在，这个 n 称作 a 模 m 的阶，记作 $\delta_m(a)$ 或 $\mathrm{ord}_m(a)$。

性质 1：$a, a^2, \cdots, a^{\delta_m(a)}$ 模 m 两两不同余。

性质 2：若 $a^n \equiv 1 \pmod{m}$，则 $\delta_m(a) \mid n$。

性质 2 推论：若 $a^p \equiv a^q \pmod{m}$，则有 $p \equiv q \pmod{\delta_m(a)}$。

性质 3:
设 $m \in \mathbf{N}^*$，$a, b \in \mathbf{Z}$，$(a, m) = (b, m) = 1$，则

$$\delta_m(ab) = \delta_m(a)\delta_m(b)$$

的充分必要条件是

$$(\delta_m(a), \delta_m(b)) = 1$$

性质 4:
设 $k \in \mathbf{N}$，m 为正整数，$a \in \mathbf{Z}$，$(a, m) = 1$，则

$$\delta_m(a^k) = \frac{\delta_m(a)}{(\delta_m(a), k)}$$

3.5.2 原根

定义:
设 m 为正整数，g 为整数。若 $(g, m) = 1$，且 $\delta_m(g) = \varphi(m)$，则称 g 为模 m 的原根。
即 g 满足 $\delta_m(g) = \varphi(m)$。当 m 是质数时，我们有 $g^i \bmod m, 0 < i < m$ 的结果互不相同。

原根判定定理:
设 $m \geq 3, (g, m) = 1$，则 g 是模 m 的原根的充要条件是，对于 $\varphi(m)$ 的每个素因数 p，都有 $g^{\frac{\varphi(m)}{p}} \not\equiv 1 \pmod{m}$。

原根个数:
若一个数 m 有原根，则它原根的个数为 $\varphi(\varphi(m))$。

原根存在定理:
一个数 m 存在原根当且仅当 $m = 2, 4, p^\alpha, 2p^\alpha$，其中 p 为奇素数，$\alpha \in \mathbf{N}^*$。

```
// == Preparations ==
const int N = 1e6 + 5;
int m , pr[N] , phi[N] , vis[N] , mod; bitset<N>b;
void Sieve(int n = N - 5);
int Qpow(int x , int p , int mod);
// == Main ==
vector<int> PrimitiveRoot(int p , int d)
{
    vector<int>pfac , g; int mod = p;
```

```
    if(!vis[p]){return g;} // whether exist primitive root
    for(int i = 1 ; i <= m && pr[i] <= phi[p] ; i++)
        if(phi[p] % pr[i] == 0)pfac.push_back(pr[i]);
    for(int a = 1 ; a < p ; a++) // minimal primitive root is O(n^{0.25+eps})
    {
        if(__gcd(a , p) != 1)continue ;
        bool ok = 1;
        for(int x : pfac)if(Qpow(a , phi[p] / x , mod) == 1){ok = 0; break ;}
        if(!ok)continue ;
        g.push_back(a); break ;
    }
    for(int i = 2 ; i < phi[p] ; i++)
        if(__gcd(i , phi[p]) == 1)g.push_back(Qpow(g[0] , i , mod));
    sort(g.begin() , g.end());
    return g;
}
```

3.6 剩余

定义：
令整数 $k \geq 2$，整数 a，m 满足 $(a, m) = 1$，若存在整数 x 使得

$$x^k \equiv a \pmod{m} \tag{1}$$

则称 a 为模 m 的 k 次剩余，否则称 a 为模 m 的 k 次非剩余。

当整数 $k \geq 2$，整数 a，m 满足 $(a, m) = 1$，模 m 有原根 g 时，令 $d = (k, \varphi(m))$，则：

性质 1： a 为模 m 的 k 次剩余当且仅当 $d \mid \operatorname{ind}_g a$，即：$a^{\frac{\varphi(m)}{d}} \equiv 1 \pmod{m}$

证明 1： 令 $x = g^y$，则方程 (1) 等价于 $g^{ky} \equiv g^{\operatorname{ind}_g a} \pmod{m}$
其等价于 $ky \equiv \operatorname{ind}_g a \pmod{\varphi(m)}$
由同余的性质，我们知道 y 有整数解当且仅当 $d = (k, \varphi(m)) \mid \operatorname{ind}_g a$，进而

$$\begin{aligned}
a^{\frac{\varphi(m)}{d}} \equiv 1 \pmod{m} &\iff g^{\frac{\varphi(m)}{d} \operatorname{ind}_g a} \equiv 1 \pmod{m} \\
&\iff \varphi(m) \mid \frac{\varphi(m)}{d} \operatorname{ind}_g a \\
&\iff \varphi(m) d \mid \varphi(m) \operatorname{ind}_g a \\
&\iff d \mid \operatorname{ind}_g a
\end{aligned}$$

性质 2： 方程 (1) 若有解，则模 m 下恰有 d 个解

性质 3：模 m 的 k 次剩余类的个数为 $\frac{\varphi(m)}{d}$，其有形式

$$a \equiv g^{di} \pmod{m}, \qquad \left(0 \le i < \frac{\varphi(m)}{d}\right)$$

3.6.1 二次剩余

下面讨论模数是奇素数时的二次剩余问题：

$$x^2 \equiv n \pmod{p}$$

参考以上结论不难得出，也就说二次剩余的数量恰为 $\frac{p-1}{2}$，其他的非 0 数都是非二次剩余，数量也是 $\frac{p-1}{2}$；判别是否为二次剩余只需检查是否 $n^{\frac{p-1}{2}} \equiv 1 \pmod{p}$ 即可。

3.6.2 Cipolla

通过随机找到一个 a 满足 $a^2 - n$ 是非二次剩余。
定义 $i^2 \equiv a^2 - n$，将所有数表示为 $A + Bi$ 的形式，有 $(a+i)^{p+1} \equiv n$。
那么 $(a+i)^{\frac{p-1}{2}}$ 即是一个解，其相反数是另一个解。

```
// == Preparations ==
typedef long long ll;
int mod , w2;
mt19937 rnd(114514);
struct Complex
{
    int a , b;
    Complex(ll aa = 0 , ll bb = 0):a(aa % mod) , b(bb % mod){}
};
typedef Complex cp;
cp operator + (cp x , cp y){return cp(x.a + y.a , x.b + y.b);}
cp operator - (cp x , cp y){return cp(x.a - y.a , x.b - y.b);}
cp operator * (cp x , cp y)
{
    return cp((ll)x.a * y.a + (ll)w2 * x.b % mod * y.b ,
              (ll)x.b * y.a + (ll)y.b * x.a);
}
cp Qpow(cp x , int p)
{
    cp res = 1;
    while(p)
    {
        if(p & 1)res = res * x;
        x = x * x;
```

```
        p >>= 1;
    }
    return res;
}
// == Main ==
int Legendre(int a){return Qpow(a , (mod - 1) / 2).a;}
pair<int , int> Cipolla(int a)
{
    if(a % mod == 0)return {0 , -1};
    if(Legendre(a) == mod - 1)return {-1 , -1};
    int res = 0;
    while(1)
    {
        int r = rnd() % mod;
        w2 = ((ll)r * r - a) % mod;
        if(w2 < 0)w2 += mod;
        if(w2 == 0)
        {
            res = r;
            break ;
        }
        if(Legendre(w2) == mod - 1)
        {
            res = Qpow(cp(r , 1) , (mod + 1) / 2).a;
            break ;
        }
    }
    if(res * 2 >= mod)res = mod - res;
    return {res , mod - res};
}
```

3.6.3 模素数 N 次剩余

我们令 $x = g^c$，g 是 p 的原根（我们一定可以找到这个 g 和 c），问题转化为求解 $(g^c)^a \equiv b \pmod p$. 稍加变换，得到

$$(g^a)^c \equiv b \pmod p$$

于是就转换成了 BSGS 的基本模型了，可以在 $O(\sqrt{p})$ 解出 c，这样可以得到原方程的一个特解 $x_0 \equiv g^c \pmod p$.

3.7 min25 筛

3.7.1 简述

应用范围：求 $\sum_{i=1}^{n} f(i)$，其中 $f(i)$ 是积性函数。
需要满足 $f(i)$ 在 i 是质数时的取值是多项式。

时间复杂度：$\Theta(n^{1-\epsilon})/\Theta(\frac{n^{\frac{3}{4}}}{\log n})$。

主要想法是将 $f(i)$ 分成三个部分后求和：i 是质数，i 是合数，$i=1$。

先求出 i 是质数部分的和。

为了方便起见，我们将 $f(i)$ 分解成若干 x^k 之和，然后分别计算，这样我们要计算的函数就是**完全积性函数**了。

考虑埃氏筛的过程，设 $g_{n,k}$ 表示**筛了 $2\sim n$ 中的数，用前 k 个质数来筛，没有被筛掉的数的 f' 值之和**。

设 p_i 表示第 i 个质数，考虑求出 g 数组，首先容易发现如果 $p_k^2>n$，$g_{n,k}=g_{n,k-1}$。

否则我们会筛掉一些数，由于是完全积性函数，容易得出转移：

$$g_{n,k}=g_{n,k-1}-f'(p_k)\times(g_{\lfloor\frac{n}{p_k}\rfloor,k-1}-\sum_{i=1}^{k-1}f'(p_i))$$

注意 g 数组计算的答案是**“将没被筛掉的数也当成质数”时的答案**。

我们考虑计算答案，设 $s_{n,k}$ 表示**筛了 $2\sim n$ 中的数，用前 k 个质数来筛，没有被筛掉的数的 f 值之和。**

现在这个 $s_{n,k}$ 求的就是真正的答案了，最终答案就是 $s_{n,0}+f(1)$。

设 $sp_i=\sum_{j=1}^{i}p_j$，D 为最大的 i 满足 $p_i^2\le n$，容易得出转移：

$$s_{n,k}=g_{n,D}-sp_k+\sum_{p_i^e\le n\&i>k}f(p_i^e)(s_{\frac{n}{p_i^e},e}+[e>1])$$

然后暴力递归求出 s 就行了，不需要记忆化。

3.7.2 求素数个数

```
// == Preparations ==
#define int long long
#define id(x) ((x<=lim)?(id1[x]):(id2[n/(x)]))
#define rep(i,j,k) for(int i=j;i<=k;++i)
#define per(i,j,k) for(int i=j;i>=k;--i)
int const N=1e6+10;
int lim,prime[N],w[N],g[N],id1[N],id2[N],m;bool vis[N];
// == Main ==
inline void init(){
    //预处理根号以内质数
    rep(i,2,lim){
        if (!vis[i]) prime[++prime[0]]=i;
        for (int j=1;j<=prime[0] && i*prime[j]<=lim;++j){
            vis[i*prime[j]]=1;
            if (i%prime[j]==0) break;
        }
    }
}
inline void solve(){
```

```
    int n;cin>>n,lim=sqrtl(n),init();
    for (int l=1,r;l<=n;l=r+1){
        r=n/(n/l),w[++m]=n/l,g[m]=w[m]-1;
        if (w[m]<=lim) id1[w[m]]=m;
        else id2[n/w[m]]=m;
        //预处理 g 数组第一维有可能的取值
    }
    rep(i,1,prime[0])
        for (int j=1;j<=m && prime[i]*prime[i]<=w[j];++j)
            g[j]-=g[id(w[j]/prime[i])]-(i-1);
    cout<<g[id(n)]<<'\n';
}
```

3.7.3 完整模板

```
// == Preparations ==
#define int long long
#define rep(i,j,k) for(int i=j;i<=k;++i)
#define per(i,j,k) for(int i=j;i>=k;--i)
#define id(x) ((x<=lim)?(id1[x]):(id2[n/(x)]))
#define add(x,y) (x=((x+y>=mod)?(x+y-mod):(x+y)))
// == Main ==
//例: 求 f(p^k)=p^k(p^k-1) 的前缀和
//发现 f(p^k)=p^{2k}-p^k
int const mod=1e9+7;
int n;
inline int f(int op,int x){
    x%=mod;
    if (!op) return x;
    return x*x%mod;
}
inline int F(int x){
    x%=mod;
    return x*(x-1)%mod;
}
inline int smf(int op,int x){
    x%=mod;
    if (!op) return (x*(x+1)/2)%mod;
    return x*(x+1)%mod*(2*x%mod+1)%mod*166666668%mod;
}
//smf -> f 前缀和
int const N=1e6+10;
int lim,prime[N],sp1[N],sp2[N],w[N],g1[N],g2[N],id1[N],id2[N],m;bool vis[N];
inline void init(){
    //预处理根号以内质数
    rep(i,2,lim){
        if (!vis[i]) prime[++prime[0]]=i;
```

```
33          for (int j=1;j<=prime[0] && i*prime[j]<=lim;++j){
34              vis[i*prime[j]]=1;
35              if (i%prime[j]==0) break;
36          }
37      }
38      rep(i,1,prime[0]){
39          sp1[i]=sp1[i-1],add(sp1[i],f(0,prime[i]));
40          sp2[i]=sp2[i-1],add(sp2[i],f(1,prime[i]));
41      }
42  }
43  inline int S(int x,int y){
44      if (prime[y]>=x) return 0;
45      int ans=((g2[id(x)]+mod-g1[id(x)])%mod+mod-(sp2[y]-sp1[y]+mod)%mod)%mod;
46      for (int i=y+1;i<=prime[0] && prime[i]*prime[i]<=x;++i)
47          for (int e=1,gg=prime[i];gg<=x;++e,gg*=prime[i])
48              add(ans,F(gg)*(S(x/gg,i)+(e>1))%mod);
49      return ans;
50  }
51  inline void solve(){
52      cin>>n,lim=sqrtl(n),init();
53      for (int l=1,r;l<=n;l=r+1){
54          r=n/(n/l),w[++m]=n/l;
55          g1[m]=(smf(0,w[m])+mod-1)%mod;
56          g2[m]=(smf(1,w[m])+mod-1)%mod;
57          if (w[m]<=lim) id1[w[m]]=m;
58          else id2[n/w[m]]=m;
59          //预处理 g 数组第一维有可能的取值
60      }
61      rep(i,1,prime[0])
62          for (int j=1;j<=m && prime[i]*prime[i]<=w[j];++j)
63              add(g1[j],mod-f(0,prime[i])*(g1[id(w[j]/prime[i])]+mod-sp1[i-1])%mod),
64              add(g2[j],mod-f(1,prime[i])*(g2[id(w[j]/prime[i])]+mod-sp2[i-1])%mod);
65      cout<<(S(n,0)+1)%mod<<'\n';
66  }
```

Chapter 4

线性代数

4.1 行列式求值

```cpp
// == Preparations ==
const int N = 605;
// == Main ==
void Swap(int i , int j)
{
    if(i == j)return ;
    for(int k = 1 ; k <= n ; k++)
        swap(a[i][k] , a[j][k]);
    f = -f;
}
int Det(int n , int mod , ll a[N][N])
{
    int m = 0;
    for(int i = 1 ; i <= n ; i++)
    {
        m++;
        for(int j = m ; j <= n ; j++)
            if(a[j][i]){Swap(m , j); break ;};
        if(!a[m][i])return 0;
        for(int j = m + 1 ; j <= n ; j++)
        {
            while(a[j][i] && a[m][i])
            {
                ll r = a[j][i] / a[m][i] % mod;
                for(int k = 1 ; k <= n ; k++)
                    a[j][k] = (a[j][k] - (ll)r * a[m][k]) % mod;
                Swap(m , j);
            }
            if(!a[m][i])Swap(m , j);
        }
    }
```

```
32      ll ans = 1;
33      for(int i = 1 ; i <= n ; i++)
34          ans *= a[i][i];
35      return (ans * f % mod + mod) % mod;
36  }
```

Chapter 5

图论

5.1 Tarjan

5.1.1 强连通分量

```
// == Main ==
void Tarjan(int now) {
    dfn[now] = low[now] = ++Index;
    s.push(now);
    for (int i = g.hd[now]; i; i = g.nxt[i])
        if (!dfn[g.to[i]]) {
            Tarjan(g.to[i]);
            low[now] = min(low[now], low[g.to[i]]);
        } else if (!scc[g.to[i]]) low[now] = min(low[now], dfn[g.to[i]]);
    if (low[now] == dfn[now]) {
        scc_cnt++;
        for (int x = 0; x != now; s.pop()) {
            x = s.top();
            scc[x] = scc_cnt;
        }
    }
    return ;
}
```

5.1.2 割边与边双

割边：

```
// == Main ==
void Tarjan(int now, int fa) {
    dfn[now] = low[now] = ++Index;
    for (int i = g.hd[now]; i; i = g.nxt[i])
        if (!dfn[g.to[i]]) {
```

```
            Tarjan(g.to[i], now);
            low[now] = min(low[now], low[g.to[i]]);
            if (low[g.to[i]] > dfn[now])
                printf("A Bridge of the Input Garph is (%d, %d)\n", now, g.to[i]);
        } else if (g.to[i] != fa) low[now] = min(low[now], dfn[g.to[i]]);
    return ;
}
```

边双：

```
// == Main ==
void Tarjan(int now, int fa) {
    dfn[now] = low[now] = ++Index;
    s.push(now);
    for (int i = g.hd[now]; i; i = g.nxt[i])
        if (!dfn[g.to[i]]) {
            Tarjan(g.to[i], now);
            low[now] = min(low[now], low[g.to[i]]);
        } else if (g.to[i] != fa) low[now] = min(low[now], dfn[g.to[i]]);
    if (low[now] == dfn[now]) {
        bcc_cnt++;
        for (int x = 0; x != now; s.pop()) {
            x = s.top();
            bcc[x] = bcc_cnt;
        }
    }
    return ;
}
```

5.1.3 割点与点双

割点：

```
// == Main ==
void Tarjan(int now, int root) {
    dfn[now] = low[now] = ++Index;
    int sons=0, flag=0;
    for (int i=g.hd[now]; i; i = g.nxt[i], sons++)
        if (!dfn[g.to[i]]) {
            Tarjan(g.to[i], now);
            low[now] = min(low[now], low[g.to[i]]);
            if (now!=root && low[g.to[i]] == dfn[now] && !flag)
                printf("A Cut Vertex of the Input Graph is %d.", now), flag=1;
        } else low[now] = min(low[now], dfn[g.to[i]]);
    if (now == root && sons >= 2)
        printf("A Cut Vertex of the Input Graph is %d.", now);
```

```
    return ;
}
```

点双：

```
// == Main ==
void Tarjan(int now) {
    dfn[now] = low[now] = ++Index;
    s.push(now);
    for (int i = g.hd[now]; i; i = g.nxt[i], sons++)
        if (!dfn[g.to[i]]) {
            Tarjan(g.to[i]);
            low[now] = min(low[now], low[g.to[i]]);
            if (low[g.to[i]] == dfn[now]) {
                printf("BCC #%d:\n", ++bcc_cnt);
                for (int x = 0; x != g.to[i]; s.pop())
                    printf("%d ", x = s.top());
                printf("%d\n", now);
            }
        } else low[now] = min(low[now], dfn[g.to[i]]);
    return ;
}
```

5.2 欧拉路径

```
// == Preparations ==
#include <vector>

vector<int> g[100005], ans;
// == Main ==
void dfs(int now) {
    while (!g[now].empty()) {
        int to = g[now].back();
        g[now].pop_back();
        dfs(to);
    }
    ans.push_back(now);
    return ;
}
```

5.3 二分图匹配

5.3.1 最大匹配

```cpp
// == Preparations ==
int chos[100005], vis[100005];
struct graph {/* ... */} g;
// == Main ==
int dfs(int now) {
    for (int i = g.hd[now]; i; i = g.nxt[i]) {
        if (vis[g.to[i]]) continue;
        vis[g.to[i]] = true;
        if (!chos[g.to[i]] || dfs(chos[g.to[i]])) {
            chos[g.to[i]] = now;
            return 1;
        }
    }
    return 0;
}

for (int i = 1; i <= n; i++) {
    memset(vis, 0, sizeof(vis));
    ans += dfs(i);
}
```

5.3.2 最大权匹配

```cpp
// == Preparations ==
int vis[1005], mat[1005], pre[1005];
long long g[505][1005];
long long w[1005], slack[1005];
// edge: g[u][n + v] = w;
// == Main ==
for (int i = 1; i <= n; i++) {
    w[i] = ~0x3f3f3f3f3f3f3f3f;
    for (int j = n + 1; j <= n + n; j++) w[i] = max(w[i], (long long)g[i][j]);
}
for (int i = 1; i <= n; i++) {
    memset(vis, 0, sizeof(vis));
    memset(slack, 0x3f, sizeof(slack));
    memset(pre, 0, sizeof(pre));
    int now = i, ri = 0;
    while (1) {
        int id = 0;
        long long delta = 0x3f3f3f3f3f3f3f3f;
        for (int j = n + 1; j <= n + n; j++)
```

```
            if (!vis[j]) {
                long long val = w[now] + w[j] - g[now][j];
                if (val < slack[j]) slack[j] = val, pre[j] = ri;
                if (slack[j] < delta) delta = slack[j], id = j;
            }
        w[i] -= delta;
        for (int j = n + 1; j <= n + n; j++)
            if (vis[j]) w[j] += delta, w[mat[j]] -= delta;
            else slack[j] -= delta;
        vis[ri = id] = 1;
        if (mat[ri]) now = mat[ri];
        else break;
    }
    while (ri) {
        mat[ri] = mat[pre[ri]];
        if (!pre[ri]) {mat[ri] = i; break;}
        ri = pre[ri];
    }
}
long long ans = 0;
for (int i = 1; i <= n + n; i++) ans += w[i];
printf("%lld\n", ans);
for (int i = n + 1; i <= n + n; i++) printf("%d ", mat[i]);
puts("");
```

5.4 网络流

5.4.1 最大流

```
// == Preparations ==
#include <queue>
// == Main ==
struct Dinic {
    int s, t;
    struct graph {
        int tot, hd[205];
        int nxt[10005], to[10005], dt[10005];
        graph() {tot = 1;}
        void add(int u, int v, int w) {
            nxt[++tot] = hd[u];
            hd[u] = tot;
            to[tot] = v;
            dt[tot] = w;
            return ;
        }
    } g;
```

```
    int cur[205], dis[205];

    void add_edge(int u, int v, int f) {g.add(u, v, f), g.add(v, u, 0); return;}
    int bfs() {
        memset(dis, 0, sizeof(dis));
        queue<int>q;
        q.push(s);
        dis[s] = 1;
        while (!q.empty()) {
            int now = q.front();
            q.pop();
            cur[now] = g.hd[now];
            for (int i = g.hd[now]; i; i = g.nxt[i])
                if (g.dt[i] && !dis[g.to[i]]) dis[g.to[i]] = dis[now] + 1,
                ↪  q.push(g.to[i]);
        }
        return dis[t];
    }
    long long dinic(int now, long long flow) {
        if (now == t) return flow;
        long long used = 0;
        for (int i = cur[now]; i && used < flow; i = g.nxt[i])
            if (g.dt[i] && dis[g.to[i]] == dis[now] + 1) {
                long long k = dinic(g.to[i], min(flow - used, (long long)g.dt[i]));
                g.dt[i] -= k, g.dt[i ^ 1] += k;
                used += k;
                cur[now] = i;
            }
        if (used == 0) dis[now] = 0;
        return used;
    }
    long long solve() {
        long long ans = 0;
        while (bfs()) ans += dinic(s, 0x3f3f3f3f3f3f3f3f);
        return ans;
    }
} F;
```

5.4.2 费用流

原始对偶：

```
// == Preparations ==
#include <queue>
// == Main ==
struct PrimalDual {
    int n, s, t;
```

```
6     struct graph {
7         int tot, hd[805];
8         int nxt[30005], to[30005], flw[30005], cst[30005];
9
10        graph() {tot = 1;}
11        void add(int u, int v, int f, int c) {
12            nxt[++tot] = hd[u];
13            hd[u] = tot;
14            to[tot] = v;
15            flw[tot] = f;
16            cst[tot] = c;
17            return ;
18        }
19    } g;
20    int h[805], dis[805], f[805], pre[805];
21    struct node {
22        int id, val;
23
24        node() = default;
25        node(int _id, int _val): id(_id), val(_val) {}
26        bool operator<(const node &x) const {return val > x.val;}
27    };
28
29    void add_edge(int u, int v, int f, int c) {g.add(u, v, f, c), g.add(v, u, 0, -c);
      ↪ return;}
30    void spfa() {
31        queue<int> q;
32        memset(h, 0x3f, sizeof(h));
33        h[s] = 0;
34        q.push(s);
35        while (!q.empty()) {
36            int now = q.front();
37            q.pop();
38            f[now] = 0;
39            for (int i = g.hd[now]; i; i = g.nxt[i])
40                if (g.flw[i] && h[g.to[i]] > h[now] + g.cst[i]) {
41                    h[g.to[i]] = h[now] + g.cst[i];
42                    if (!f[g.to[i]]) q.push(g.to[i]), f[g.to[i]] = 1;
43                }
44        }
45        return ;
46    }
47    int dijkstra() {
48        priority_queue<node> q;
49        memset(dis, 0x3f, sizeof(dis));
50        memset(pre, 0, sizeof(pre));
51        q.emplace(s, dis[s] = 0);
52        while (!q.empty()) {
53            int now = q.top().id, tmp = q.top().val;
```

```
            q.pop();
            if (dis[now] != tmp) continue;
            for (int i = g.hd[now]; i; i = g.nxt[i])
                if (g.flw[i] && dis[g.to[i]] > dis[now] + g.cst[i] + h[now] -
                ↪ h[g.to[i]]) {
                    q.emplace(g.to[i], dis[g.to[i]] = dis[now] + g.cst[i] + h[now] -
                    ↪ h[g.to[i]]);
                    pre[g.to[i]] = i ^ 1;
                }
        }
        return pre[t];
    }
    pair<int, int> solve() {
        int flow = 0, cost = 0;
        spfa();
        while (dijkstra()) {
            for (int i = 1; i <= n; i++)
                if (dis[i] < 0x3f3f3f3f) h[i] += dis[i];
            int mnflow = 0x3f3f3f3f;
            for (int i = t; i != s; i = g.to[pre[i]]) mnflow = min(mnflow,
            ↪ g.flw[pre[i] ^ 1]);
            for (int i = t; i != s; i = g.to[pre[i]]) g.flw[pre[i] ^ 1] -= mnflow,
            ↪ g.flw[pre[i]] += mnflow;
            flow += mnflow;
            cost += mnflow * h[t];
        }
        return {flow, cost};
    }
} F;
```

5.4.3 上下界

$f(u,v)$ 表示边 (u,v) 的流量，$f(u)$ 表示 u 的出流减入流，$c(u,v)$ 表示边 (u,v) 的容量。
对于每条边给定一个流量下界 $b(u,v)$，需要额外满足 $\forall(u,v), b(u,v) \le f(u,v) \le c(u,v)$。

无源汇上下界可行流

没有源点和汇点，对于所有点满足 $f(u)=0$，求一个可行的流。
先强制每条边流到流量下界，建立虚拟源汇点 s,t，对于每个点 u 考虑此时的净流量：

- $f(u)=0$：满足条件，不用管。
- $f(u)>0$：出流大于入流，从 u 向 t 连容量为 $f(u)$ 的边。
- $f(u)<0$：入流大于出流，从 s 向 u 连容量为 $-f(u)$ 的边。

将原图中每条边的容量设为 $c(u,v)-b(u,v)$，则从 s 到 t 的流相当于增加调整流量的过程。

若 s 的出边流满（等同于 t 的入边流满），则找到了一条可行流。

有源汇上下界可行流

连一条 t 到 s 容量正无穷下界为 0 的边，然后跑无源汇上下界可行流即可，流量为新增边的流量。

有源汇上下界最大流

求出可行流后删掉 t 到 s 的边，在残量网络上跑 s 到 t 的最大流，该最大流加上原本的可行流即为答案。

有源汇上下界最小流

同理，改成求 t 到 s 的最大流，原可行流减去该最大流即为答案。

有源汇上下界最小费用流

做法是一样的，所有新增边费用为 0。

需要注意求最小流时需要改成费用最大。

5.4.4 有负圈的最小费用最大流

先钦定所有负圈边流满，即上下界均为流量。然后对于负边建反向、容量相同、费用为相反数的边用于退流原边。

这样就转化成了有源汇上下界最小费用最大流。

5.5 k 短路

复杂度为 $O((n+m)\log n+k\log k)$。

```
// == Preparations ==
int ontree[200005];
struct graph {
    int tot, hd[5005];
    int nxt[200005], to[200005];
    long long dt[200005];

    void add(int u, int v, long long w) {
        nxt[++tot] = hd[u];
        hd[u] = tot;
        to[tot] = v;
        dt[tot] = w;
        return ;
    }
```

```
} g;
long long dis[5005];
struct node {
    int id;
    long long val;

    node() = default;
    node(int _id, long long _val): id(_id), val(_val) {}
    bool operator<(const node &x) const {return val > x.val;}
};
priority_queue<node> q;
int vis[5005];

// 以下左偏树
struct HeapNode {
    long long val;
    int to, dist;
    HeapNode *ls, *rs;

    HeapNode() = default;
    HeapNode(long long _val, int _to): val(_val), to(_to), dist(1), ls(nullptr),
    ↪  rs(nullptr) {}
};
struct Heap {
    HeapNode *root[5005];

    HeapNode *merge(HeapNode *u, HeapNode *v) {
        if (!u) return v;
        if (!v) return u;
        if (u->val > v->val) swap(u, v);
        HeapNode *p = new HeapNode(*u);
        p->rs = merge(u->rs, v);
        if (!p->ls || p->ls->dist < p->rs->dist) swap(p->ls, p->rs);
        if (p->rs) p->dist = p->rs->dist + 1;
        else p->dist = 1;
        return p;
    }
} h;

struct Node {
    HeapNode *id;
    long long val;

    Node() = default;
    Node(HeapNode *_id, long long _val): id(_id), val(_val) {}
    bool operator<(const Node &x) const {return val > x.val;}
};
priority_queue<Node> Q;
// == Main ==
```

```
void dfs(int now) {
    vis[now] = 1;
    for (int i = g.hd[now]; i; i = g.nxt[i])
        if (!vis[g.to[i]] && dis[g.to[i]] == dis[now] + g.dt[i]) ontree[i] = 1,
        ↪ dfs(g.to[i]);
    return ;
}
void dfs2(int now) {
    for (int i = g.hd[now]; i; i = g.nxt[i])
        if (ontree[i]) h.root[g.to[i]] = h.merge(h.root[g.to[i]], h.root[now]),
        ↪ dfs2(g.to[i]);
    return;
}

memset(dis, 0x3f, sizeof(dis));
q.emplace(n, dis[n] = 0);
while (!q.empty()) {
    int now = q.top().id;
    long long tmp = q.top().val;
    q.pop();
    if (tmp != dis[now]) continue;
    for (int i = g.hd[now]; i; i = g.nxt[i])
        if (dis[g.to[i]] > dis[now] + g.dt[i]) q.emplace(g.to[i], dis[g.to[i]] =
        ↪ dis[now] + g.dt[i]);
}
dfs(n);
for (int i = 1; i <= n; i++)
    for (int j = g.hd[i]; j; j = g.nxt[j])
        if (!ontree[j] && g.to[j] != n)
            h.root[g.to[j]] = h.merge(h.root[g.to[j]], new HeapNode(dis[i] + g.dt[j]
            ↪ - dis[g.to[j]], i));
dfs2(n);
if (h.root[1]) Q.emplace(h.root[1], dis[1] + h.root[1]->val);
while (!Q.empty()) { // 每次取出来一条路径
    HeapNode *now = Q.top().id;
    long long d = Q.top().val;
    Q.pop();
    if (now->ls) Q.emplace(now->ls, d - now->val + now->ls->val);
    if (now->rs) Q.emplace(now->rs, d - now->val + now->rs->val);
    HeapNode *tmp = h.root[now->to];
    if (tmp) Q.emplace(tmp, d + tmp->val);
}
```

5.6 全局最小割

时间复杂度为 $O(|V|^3)$。

```cpp
// == Preparations ==
int g[605][605], vis1[605], vis2[605];
long long w[605];
// == Main ==
long long Stoer_Wagner() {
    long long ans = 0x3f3f3f3f3f3f3f3f;
    for (int i = 1; i < n; i++) {
        int s = 0, t = 0;
        memset(vis2, 0, sizeof(vis2));
        memset(w, 0, sizeof(w));
        for (int j = 1; j <= n - i + 1; j++) {
            int now = 0;
            for (int k = 1; k <= n; k++)
                if (!vis1[k] && !vis2[k] && w[k] >= w[now]) now = k;
            s = t, t = now;
            vis2[now] = 1;
            for (int k = 1; k <= n; k++) w[k] += g[k][now];
        }
        ans = min(ans, w[t]);
        vis1[t] = 1;
        for (int j = 1; j <= n; j++)
            if (j != s) g[s][j] += g[t][j], g[j][s] += g[j][t];
    }
    return ans;
}
```

5.7 支配树

$idom_u$ 为 u 在支配树上的父亲。

最后 id 形成 dfs 序。

```cpp
// == Preparations ==
#include <vector>

struct graph {
    int tot, hd[200005];
    int nxt[300005], to[300005];

    void add(int u, int v) {
        nxt[++tot] = hd[u];
        hd[u] = tot;
        to[tot] = v;
        return ;
    }
} g, fg;
```

```
int timer, fa[200005], dfn[200005], id[200005];
int sdom[200005], idom[200005];
struct dsu {
    int fa[200005], mn[200005];

    dsu() {for (int i = 1; i < 200005; i++) fa[i] = mn[i] = i;}
    int find(int x) {
        if (x == fa[x]) return x;
        int tmp = find(fa[x]);
        if (dfn[sdom[mn[fa[x]]]] < dfn[sdom[mn[x]]]) mn[x] = mn[fa[x]];
        return fa[x] = tmp;
    }
} d;
vector<int> vec[200005];
int siz[200005];
// == Main ==
void dfs(int now) {
    id[dfn[now] = ++timer] = now;
    for (int i = g.hd[now]; i; i = g.nxt[i])
        if (!dfn[g.to[i]]) fa[g.to[i]] = now, dfs(g.to[i]);
    return ;
}
void solve() {
    dfs(1);
    for (int i = 1; i <= n; i++) sdom[i] = i;
    for (int i = timer; i >= 1; i--) {
        int u = id[i];
        for (int v : vec[u]) {
            d.find(v);
            if (sdom[d.mn[v]] == u) idom[v] = u;
            else idom[v] = d.mn[v];
        }
        if (i == 1) continue;
        for (int j = fg.hd[u]; j; j = fg.nxt[j]) {
            if (!dfn[fg.to[j]]) continue;
            if (dfn[fg.to[j]] < dfn[sdom[u]]) sdom[u] = fg.to[j];
            else if (dfn[fg.to[j]] > dfn[u]) {
                d.find(fg.to[j]);
                if (dfn[sdom[d.mn[fg.to[j]]]] < dfn[sdom[u]]) sdom[u] =
                ↪  sdom[d.mn[fg.to[j]]];
            }
        }
        vec[sdom[u]].push_back(u);
        d.fa[u] = fa[u];
    }
    for (int i = 2; i <= timer; i++)
        if (idom[id[i]] != sdom[id[i]]) idom[id[i]] = idom[idom[id[i]]];
    return ;
```

```
}
```

5.8 最大团搜索

给定一张无向图，Bron-Kerbosch 算法可以搜索出所有极大团。其时间复杂度与极大团数量相同，为 $\mathcal{O}(3^{\frac{n}{3}})$。

我们定义一张图的 degeneracy 为最小的 d 满足其任意导出子图都存在至少一个点度数 $\leq d$，则复杂度为 $\mathcal{O}(dn3^{\frac{d}{3}})$。

```
// == Preparations ==
#define popc __builtin_popcountll
typedef long long ll;
const int N = 55;
int n , ans , a[N]; ll g[N] , d[N]; //g is graph matrix , b[0...n-1][0...n-1]
// == Main ==
void Dfs(ll R , ll P , ll X)
{
    if(!P)
    {
        if(!X)ans = max(ans , popc(R));//now R is the maximal clique
        return ;
    }
    ll v = __lg(P | X);
    for(ll s = P ^ (P & g[v]) ; s ; s -= s & -s)
    {
        ll u = __lg(s & -s);
        Dfs(R | (1ll << u) , P & g[u] , X & g[u]);
        P ^= 1ll << u , X |= 1ll << u;
    }
}
void BronKerbosch()
{
    for(int i = 0 ; i < n ; i++)d[i] = g[i];
    for(int i = 0 ; i < n ; i++)
    {
        int &k = a[i]; k = -1;
        for(int j = 0 ; j < n ; j++)
            if(d[j] != -1 && (k == -1 || popc(d[j]) <= popc(d[k])))
                k = j;
        d[k] = -1;
        for(int j = 0 ; j < n ; j++)
            if(d[j] >= 0 && (d[j] >> k) & 1)d[j] ^= 1ll << k;
    }
    ll P = (1ll << n) - 1 , X = 0;
    for(int i = 0 ; i < n ; i++)
    {
```

```
        int u = a[i];
        Dfs(1ll << u , P & g[u] , X & g[u]);
        P ^= 1ll << u , X |= 1ll << u;
    }
}
```

5.9 最小树形图

5.9.1 朱刘算法

基本想法：对除根节点外的所有点，选一条边权最小的入边；对于入边形成的简单环，最多只有一条边会被删除，将环缩成一个点，并改变与其相连边的边权。

如果有环，每轮至少减少一个点。时间复杂度 $\mathcal{O}(nm)$。

```
bool ZhuLiu(int n , int m , int rt)
{
    while(1)
    {
        fill(mn + 1 , mn + n + 1 , 1e9);
        fill(id + 1 , id + n + 1 , 0);
        fill(top + 1 , top + n + 1 , 0);
        for(int i = 1 ; i <= m ; i++)
        {
            auto [u , v , w] = e[i];
            if(w < mn[v])mn[v] = w , fa[v] = u;
        }
        for(int i = 1 ; i <= n ; i++)
        {
            if(i == rt)continue ;
            if(mn[i] != 1e9)ans += mn[i];
            else return 0;
        }
        int cnt = 0 , tot = 0;
        for(int i = 1 ; i <= n ; i++)
        {
            if(id[i] || i == rt)continue ;
            int u = i;
            for(; u != rt && !top[u] ; u = fa[u])top[u] = i;
            if(top[u] == i)
            {
                id[u] = ++cnt;
                for(int v = fa[u] ; v != u ; v = fa[v])
                    id[v] = cnt;
            }
        }
        if(!cnt)return 1;
```

```
        for(int i = 1 ; i <= n ; i++)
            if(!id[i])id[i] = ++cnt;
        for(int i = 1 ; i <= m ; i++)
        {
            auto [u , v , w] = e[i];
            if(id[u] == id[v])continue ;
            e[++tot] = {id[u] , id[v] , w - mn[v]};
        }
        n = cnt , m = tot , rt = id[rt];
    }
}
```

5.9.2 可并堆优化

上述算法的瓶颈在于对每个点（包括缩环得到的点）找边权最小的入边。

考虑用可并堆存储每个点 u 的入边，为对该点入边边权同时减去 $w(\text{in}(u))$，堆需要支持全局减。

为了避免每次对没有缩环的点的遍历，将过程改为迭代式的。

同时维护两个并查集，分别记录连通块信息、当前所在环的编号。

为保证图能被缩成一个点，需要先加入边 $(i, i \bmod n + 1, +\infty)$。

> **contraction 树**：为了计算方案，引入一种树结构，树上的叶子是图的原始节点，每个非叶子节点都表示一个环，每个点的父亲是它所在的环。

收缩过程中，对每个环内的点记录其所在环的编号 f、环上的前驱（即树上的兄弟）pre、以及入边 in，即可刻画出该结构。

由于优化后的缩环过程与 r 无关，对于任意合法的根 r，都可以通过展开该结构得到以其为根的最小树形图。展开过程：

1. 对于任意合法的根节点 r，从 r 开始跳父亲直到 contraction 树的树根。
2. 对于每层的环，r 的祖先 u 就是从 r 开始沿树边走，第一个访问到的点，该环不要的边就是 $\text{in}(u)$。
3. 从 $v = pre_u$ 开始依次访问环上的点，跳 $v = pre_v$，所有 $\text{in}(v)$ 就是环上的边，答案加上 $w(\text{in}(v))$。根据 v 的入边可以得到进入环 v 的点 p，递归处理 p（即展开环 v），注意 p 只要跳到 v 即可。

展开过程 contraction 树上每条边只会访问一次，是 $\mathcal{O}(n)$ 的。

对于缩环过程，可以构造数据，使得 $\mathcal{O}(m)$ 条边需要从堆中 pop，时间复杂度 $\mathcal{O}(m \log m)$。

```
// == Preparations ==
#define pre(x) (id[e[in[x]].u])
typedef pair<int , int>pr;
int n , m , rt , num , h[N] , in[N] , f[N] , pre[N]; ll ans;
struct Edge{int u , v , w;}e[M + N];
struct Heap<pr>{}pq[N];//mergeable heap
struct UnionSet
```

```
{
    int fa[N];
    void init(int n){iota(fa + 1 , fa + n + 1 , 1);}
    int find(int x){return x == fa[x] ? x : fa[x] = find(fa[x]);}
    int& operator [](int x){return fa[find(x)];}
}col , id;
// == Main ==
bool Work(int u)
{
    while(pq[u] && (in[u] = pq[u].top().se , pre(u) == u))
        pq[u].pop();
    if(!pq[u])return 0;
    auto [w , i] = pq[u].top(); int v = pre(u);
    in[u] = i , pq[u].tag -= w , pq[u].pop();
    if(col[u] != col[v])
    {
        col[u] = col[v];
        return 1;
    }
    pq[++num].merge(pq[u]) , col[u] = col[num];
    for(int x = v ; x != u ; pre[x] = pre(x) , x = pre[x])
        f[x] = id[x] = num , pq[num].merge(pq[x]);
    f[u] = id[u] = num , pre[u] = v;
    return Work(num);
}
ll Expand(ll u , ll rt)
{
    ll ans = 0;
    for(; u != rt ; u = f[u])
    {
        for(int v = pre[u] ; v != u ; v = pre[v])
        {
            int w = e[in[v]].w;
            if(w >= INF)return -1;
            ans += Expand(e[in[v]].v , v) + w;
        }
    }
    return ans;
}
ll Solve(int n , int m , int rt , Edge e[N + M])//filled e[1...m]
{
    for(int i = 1 ; i <= n ; i++)e[++m] = {i % n + 1 , i , INF};
    for(int i = 1 ; i <= m ; i++)pq[e[i].v].push(pr(e[i].w , i));
    col.init(n * 2) , id.init(n * 2) , num = n;
    for(int i = 1 ; i <= n ; i++)if(id[i] == i)Work(i);
    return Expand(rt , num);
}
```

5.10 弦图

5.10.1 MCS 最大势算法。

```cpp
// == Preparations ==
#include <vector>

int pos[/* ... */], p[/* ... */];
vector<int> vec[/* ... */];
// == Main ==
for (int i = 1; i <= n; i++) pos[i] = vec[0].size(), vec[0].push_back(i);
for (int i = 1, j = 0; i <= n; i++, j++) {
    while (vec[j].empty()) j--;
    int u = p[i] = vec[j].back();
    vec[j].pop_back();
    pos[u] = -1;
    for (int k = g.hd[u]; k; k = g.nxt[k])
        if (pos[g.to[k]] != -1) {
            int v = g.to[k];
            pos[vec[l[v]].back()] = pos[v];
            swap(vec[l[v]][pos[v]], vec[l[v]].back());
            vec[l[v]].pop_back();
            pos[v] = vec[++l[v]].size();
            vec[l[v]].push_back(v);
        }
}
reverse(p + 1, p + n + 1);
```

5.10.2 弦图判定

跑 MCS，然后判断是否为完美消除序列。

具体地，对于每个 p_i，找到与之相连且在它之后出现的点，按出现顺序记为 $c_1, c_2, \ldots, c_k$，我们只需要判断 c_1 与 c_j 之间是否有边即可。因为这个团中其他边会在 $p_{c_2}, p_{c_3}, \ldots, p_{c_k}$ 中被判断。

5.10.3 求弦图的团数与色数

求团数：

设 $N(x)$ 为完美消除序列中在 x 之后且与 x 相连的点的集合，则弦图的最大团一定可以被表示为 $\{x\} + N(x)$，则 $|\{x\} + N(x)|$ 的最大值就是弦图的团数。

求色数：

考虑按完美消除序列从后往前考虑，贪心染 mex，这样需要的颜色数量等于团数。由于团数小于等于色数，这样取到等号，一定最小。

5.10.4 求弦图的最大独立集和最小团覆盖

最大独立集：

按完美消除序列从前往后贪心。正确性证明：每次考虑最靠前的极大团，选最前面的点不劣于选其他点，且优于不选点。

最小团覆盖：

取最大独立集中的每个点 x 对应的团 $\{x\}+N(x)$，这样需要的团的数量等于最大独立集的大小。由于最大独立集小于等于最小团覆盖，这样取到等号，一定最小。

5.10.5 tricks

区间图是弦图，完美消除序列为按区间右端点从小到大排序。

树上距离不超过 k 的点连边是弦图，完美消除序列为 bfs 序的逆序。

5.11 图计数相关

5.11.1 环计数

任意环计数：状压 DP，设 $dp_{S,i}$ 为从 S 中最小的点出发，走过的点集为 S，现在在 i 的方案数，在能走到起点时统计答案，最终答案为减去边数再除 2。

三元环计数：将点按度数从小到大排序，然后边从前往后定向，这样每个点出度只有 $O(\sqrt{m})$ 条，枚举一个点 u，再枚举 u 指向的点 v，再枚举 v 指向的点 w，check 是否存在边 (u,w) 即可。复杂度分析：当 v 入度 $\le\sqrt{m}$ 时，u 只有 $O(\sqrt{m})$ 个；当 v 入度 $>\sqrt{m}$ 时，w 至多 $O(\sqrt{m})$ 个。

竞赛图三元环计数：$\binom{n}{3}-\sum_{i=1}^{n}\binom{d(i)}{2}$，其中 $d(i)$ 为 i 的入度或出度。

四元环计数：同样的方法排序定向，然后枚举一个点 a，对所有 c 记录 $a\to b\to c$ 的数量，然后对于每个 c 任取两个 b 即可组成一个四元环。

团计数：同样的方法排序定向，然后枚举一个点 u，对 u 的出边 meet-in-middle。具体来说，对一个集合搜出每个子集是否为团，然后做高维前缀和，枚举另一个集合的子集，维护其是否为团及这个集合中的点与前一个集合中的点的连边的交即可。时间复杂度为 $O(\sqrt{m}\times 2^{\frac{\sqrt{2m}}{2}})$。（QOJ7514）

5.11.2 Prufer 序列

树到 Prufer 序列的映射：每次选择编号最小的叶子删掉并记录其父亲直到只剩两个点。

```
// == Main ==
// d[i] 为 i 的度数 -1
for (int i = 1, j = 1; i <= n - 2; i++, j++) {
    while (d[j]) j++;
    p[i] = fa[j];
    while (i <= n - 2 && !--d[p[i]] && p[i] < j) p[i + 1] = fa[p[i]], i++;
}
```

Prufer 序列到树的映射：先算出度数，每次选择一个编号最小的度数为 1 的点与当前 Prufer 序列的点连接，然后给两个点的度数都 -1，最后剩两个度数为 1 的点连起来。

```cpp
// == Main ==
// d[i] 为 i 的度数 -1
p[n - 1] = n;
for (int i = 1, j = 1; i < n; i++, j++) {
    while (d[j]) j++;
    fa[j] = p[i];
    while (i < n && !--d[p[i]] && p[i] < j) fa[p[i]] = p[i + 1], i++;
}
```

给一张 k 个连通块的图，每个连通块的点数为 s_i，求连 $k-1$ 条边使其连通的方案数：

$$
\begin{aligned}
&\sum_{d_i\geq 1,\sum_{i=1}^{k}(d_i-1)=k-2}\binom{k-2}{d_1-1,d_2-1,\cdots,d_k-1}\cdot\prod_{i=1}^{k}s_i^{d_i}\\
=&\prod_{i=1}^{k}s_i\sum_{d_i\geq 1,\sum_{i=1}^{k}(d_i-1)=k-2}\binom{k-2}{d_1-1,d_2-1,\cdots,d_k-1}\cdot\prod_{i=1}^{k}s_i^{d_i-1}\\
=&(\sum_{i=1}^{k}s_i)^{k-2}\prod_{i=1}^{k}s_i\\
=&n^{k-2}\prod_{i=1}^{k}s_i
\end{aligned}
$$

第二个等号为多元二项式定理。

Chapter 6

多项式

6.1 拉格朗日插值法

$$f(k)=\sum_{i=0}^{n}y_i\prod_{j\neq i}\frac{k-x_j}{x_i-x_j}$$

取值连续时的优化：
当 $x_i=i$ 时可以优化成 $O(n)$。

$$\begin{aligned}f(k)&=\sum_{i=0}^{n}y_i\prod_{j\neq i}\frac{k-x_j}{x_i-x_j}\\&=\sum_{i=0}^{n}y_i\prod_{j\neq i}\frac{k-j}{i-j}\end{aligned}$$

令 $pre_i=\prod_{j=0}^{i}(k-j), suf_i=\prod_{j=i}^{n}(k-j)$：

$$f(k)=\sum_{i=0}^{n}y_i\frac{pre_{i-1}\times suf_{i+1}}{i!\times(n-i)!}\times(-1)^{n-i}$$

重心拉格朗日插值：

$$\begin{aligned}f(k)&=\sum_{i=0}^{n}y_i\prod_{i\neq j}\frac{k-x_j}{x_i-x_j}\\&=\sum_{i=0}^{n}y_i\frac{\prod\limits_{j\neq i}k-x_j}{\prod\limits_{j\neq i}x_i-x_j}\\&=\sum_{i=0}^{n}\frac{y_i}{k-x_i}\frac{\prod\limits_{j=0}^{n}k-x_j}{\prod\limits_{j\neq i}x_i-x_j}\end{aligned}$$

设 $g(k)=\prod_{i=0}^{n}k-x_i, t_i=\dfrac{y_i}{\prod_{j\neq i}x_i-x_j}$。

则

$$\begin{aligned}f(k)&=\sum_{i=0}^{n}\frac{g(k)t_i}{k-x_i}\\&=g(k)\sum_{i=0}^{n}\frac{t_i}{k-x_i}\end{aligned}$$

于是每次插入只需要更新之前的所有 t_i 并计算当前点的 t_i 即可，时间复杂度为 $O(n)$。

由于 $g(k)$ 和 $\sum_{i=0}^{k}\dfrac{t_i}{k-x_i}$ 都可以 $O(n)$ 求得，故询问复杂度也是 $O(n)$。

6.2 牛顿迭代

用于解决下列问题：

> 已知函数 G 且 $G(F(x))=0$，求多项式 F（ $\bmod x^n$）。

结论：

$$F(x)=F_*(x)-\frac{G(F_*(x))}{G'(F_*(x))}\pmod{x^n}$$

其中 $F_*(x)$ 为做到 $x^{n/2}$ 时的答案。

6.3 FFT

```
// == Preparations ==
struct complex {
    double a, b;

    complex() = default;
    complex(double _a, double _b): a(_a), b(_b) {}
    complex operator+(const complex &x) const {return complex(a + x.a, b + x.b);}
    complex operator-(const complex &x) const {return complex(a - x.a, b - x.b);}
    complex operator*(const complex &x) const {return complex(a * x.a - b * x.b, a *
     ↪  x.b + b * x.a);}
    complex operator/(const complex &x) const {
        double t = b * b + x.b * x.b;
        return complex((a * x.a + b * x.b) / t, (b * x.a - a * x.b) / t);
    }
    complex &operator+=(const complex &x) {return *this = *this + x;}
    complex &operator-=(const complex &x) {return *this = *this - x;}
    complex &operator*=(const complex &x) {return *this = *this * x;}
```

```
    complex &operator/=(const complex &x) {return *this = *this / x;}
};
// == Main ==
void FFT(vector<complex> &f, int flag) const {
    int n = f.size();
    vector<int> swp(n);
    for (int i = 0; i < n; i++) {
        swp[i] = swp[i >> 1] >> 1 | ((i & 1) * (n >> 1));
        if (i < swp[i]) std::swap(f[i], f[swp[i]]);
    }
    for (int mid = 1; mid < n; mid <<= 1) {
        complex w1(cos(pi / mid), flag * sin(pi / mid));
        for (int i = 0; i < n; i += mid << 1) {
            complex w(1, 0);
            for (int j = 0; j < mid; j++, w *= w1) {
                complex x = f[i + j], y = w * f[i + mid + j];
                f[i + j] = x + y, f[i + mid + j] = x - y;
            }
        }
    }
    return;
}
```

6.4 常用 NTT 模数及其原根

模数	原根	分解
167772161	3	$5 \times 2^{25} + 1$
469762049	3	$7 \times 2^{26} + 1$
998244353	3	$119 \times 2^{23} + 1$
1004535809	3	$479 \times 2^{21} + 1$
2013265921	31	$15 \times 2^{27} + 1$
2281701377	3	$17 \times 2^{27} + 1$

6.5 多项式模板

```
// == Preparations ==
#include <vector>
// == Main ==
namespace Poly {
    const int mod = 998244353, G = 3, invG = 332748118;

    inline int power(int a, int b) {
        int ans = 1;
        while (b) {
```

```
        if (b & 1) ans = (long long)ans * a % mod;
        a = (long long)a * a % mod;
        b >>= 1;
    }
    return ans % mod;
}

struct poly: vector<int> {
    poly(initializer_list<int> &&arg): vector<int>(arg) {}
    template<typename... argT>
    poly(argT &&...args): vector<int>(forward<argT>(args)...) {}

    poly operator+(const poly &b) const {
        const poly &a = *this;
        poly ans(max(a.size(), b.size()));
        for (int i = 0; i < (int)ans.size(); i++)
            ans[i] = ((i < (int)a.size() ? a[i] : 0) + (i < (int)b.size() ? b[i]
            ↪  : 0)) % mod;
        return ans;
    }
    poly operator+=(const poly &b) {return *this = *this + b;}
    poly operator-(const poly &b) const {
        const poly &a = *this;
        poly ans(max(a.size(), b.size()));
        for (int i = 0; i < (int)ans.size(); i++)
            ans[i] = ((i < (int)a.size() ? a[i] : 0) - (i < (int)b.size() ? b[i]
            ↪  : 0) + mod) % mod;
        return ans;
    }
    poly operator-=(const poly &b) {return *this = *this - b;}
    void NTT(poly &g, int flag) const {
        int n = g.size();
        vector<unsigned long long> f(g.begin(), g.end());
        vector<int> swp(n);
        for (int i = 0; i < n; i++) {
            swp[i] = swp[i >> 1] >> 1 | ((i & 1) * (n >> 1));
            if (i < swp[i]) std::swap(f[i], f[swp[i]]);
        }
        for (int mid = 1; mid < n; mid <<= 1) {
            int w1 = power(flag ? G : invG, (mod - 1) / mid / 2);
            vector<int> w(mid);
            w[0] = 1;
            for (int i = 1; i < mid; i++) w[i] = (long long)w[i - 1] * w1 % mod;
            for (int i = 0; i < n; i += mid << 1)
                for (int j = 0; j < mid; j++) {
                    int t = (long long)w[j] * f[i + mid + j] % mod;
                    f[i + mid + j] = f[i + j] - t + mod;
                    f[i + j] += t;
                }
```

```
57              if (mid == 1 << 10)
58                  for (int i = 0; i < n; i++) f[i] %= mod;
59          }
60          int inv = flag ? 1 : power(n, mod - 2);
61          for (int i = 0; i < n; i++) g[i] = f[i] % mod * inv % mod;
62          return;
63      }
64
65      // 下面是基于转置原理的 NTT，相对朴素版本效率更高。
66      void NTT(poly &g, int flag) const {
67          int n = g.size();
68          vector<int> f(g.begin(), g.end());
69          if (flag) {
70              for (int mid = n >> 1; mid >= 1; mid >>= 1) {
71                  int w1 = power(G, (mod - 1) / mid / 2);
72                  vector<int> w(mid);
73                  w[0] = 1;
74                  for (int i = 1; i < mid; i++) w[i] = (long long)w[i - 1] * w1 %
                        ↪ mod;
75                  for (int i = 0; i < n; i += mid << 1)
76                      for (int j = 0; j < mid; j++) {
77                          int t = (long long)(f[i + j] - f[i + mid + j] + mod) *
                                ↪ w[j] % mod;
78                          f[i + j] = f[i + j] + f[i + mid + j] >= mod ?
79                              f[i + j] + f[i + mid + j] - mod : f[i + j] + f[i +
                                ↪ mid + j];
80                          f[i + mid + j] = t;
81                      }
82              }
83              for (int i = 0; i < n; i++) g[i] = f[i];
84          } else {
85              for (int mid = 1; mid < n; mid <<= 1) {
86                  int w1 = power(invG, (mod - 1) / mid / 2);
87                  vector<int> w(mid);
88                  w[0] = 1;
89                  for (int i = 1; i < mid; i++) w[i] = (long long)w[i - 1] * w1 %
                        ↪ mod;
90                  for (int i = 0; i < n; i += mid << 1)
91                      for (int j = 0; j < mid; j++) {
92                          int t = (long long)w[j] * f[i + mid + j] % mod;
93                          f[i + mid + j] = f[i + j] - t < 0 ? f[i + j] - t + mod :
                                ↪ f[i + j] - t;
94                          f[i + j] = f[i + j] + t >= mod ? f[i + j] + t - mod : f[i
                                ↪ + j] + t;
95                      }
96              }
97              int inv = power(n, mod - 2);
98              for (int i = 0; i < n; i++) g[i] = (long long)f[i] * inv % mod;
99          }
```

```
100             return;
101         }
102
103         poly operator*(poly b) const {
104             poly a(*this);
105             int n = 1, len = (int)(a.size() + b.size()) - 1;
106             while (n < len) n <<= 1;
107             a.resize(n), b.resize(n);
108             NTT(a, 1), NTT(b, 1);
109             poly c(n);
110             for (int i = 0; i < n; i++) c[i] = (long long)a[i] * b[i] % mod;
111             NTT(c, 0);
112             c.resize(len);
113             return c;
114         }
115         poly operator*=(const poly &b) {return *this = *this * b;}
116         poly inv() const {
117             poly f = *this, g;
118             g.push_back(power(f[0], mod - 2));
119             int n = 1;
120             while (n < (int)f.size()) n <<= 1;
121             f.resize(n << 1);
122             for (int len = 2; len <= n; len <<= 1) {
123                 poly tmp(len), ff(len << 1);
124                 for (int i = 0; i < len >> 1; i++) tmp[i] = g[i] * 2 % mod;
125                 for (int i = 0; i < len; i++) ff[i] = f[i];
126                 g.resize(len << 1);
127                 NTT(g, 1), NTT(ff, 1);
128                 for (int i = 0; i < len << 1; i++) g[i] = (long long)g[i] * g[i] %
                    ↪ mod * ff[i] % mod;
129                 NTT(g, 0);
130                 g.resize(len);
131                 for (int i = 0; i < len; i++) g[i] = (tmp[i] - g[i] + mod) % mod;
132             }
133             g.resize(size());
134             return g;
135         }
136         poly sqrt() const { // need F[0] = 1.
137             poly f = *this, g;
138             g.push_back(1);
139             int n = 1;
140             while (n < (int)f.size()) n <<= 1;
141             f.resize(n << 1);
142             for (int len = 2; len <= n; len <<= 1) {
143                 poly tmp(len), ff(len << 1);
144                 for (int i = 0; i < len >> 1; i++) tmp[i] = g[i] * 2 % mod;
145                 for (int i = 0; i < len; i++) ff[i] = f[i];
146                 g.resize(len << 1);
147                 NTT(g, 1);
```

```
148                 for (int i = 0; i < len << 1; i++) g[i] = (long long)g[i] * g[i] %
                    ↪ mod;
149                 NTT(g, 0);
150                 g += ff;
151                 g *= tmp.inv();
152                 g.resize(len);
153             }
154             g.resize(size());
155             return g;
156         }
157         poly derivative() const {
158             poly f(*this);
159             for (int i = 1; i < (int)f.size(); i++) f[i - 1] = (long long)f[i] * i %
                ↪ mod;
160             f.pop_back();
161             return f;
162         }
163         poly integral() const {
164             poly f(*this);
165             f.push_back(0);
166             for (int i = f.size() - 1; i >= 1; i--) f[i] = (long long)f[i - 1] *
                ↪ power(i, mod - 2) % mod;
167             f[0] = 0;
168             return f;
169         }
170         poly ln() const {
171             poly f((derivative() * inv()).integral());
172             f.resize(size());
173             return f;
174         }
175         poly exp() const { // 需要满足 F[0] = 0
176             poly f(*this), g;
177             g.push_back(1);
178             int n = 1;
179             while (n < (int)size()) n <<= 1;
180             f.resize(n);
181             for (int len = 2; len <= n; len <<= 1) {
182                 poly tmp(g);
183                 g.resize(len);
184                 g = g.ln();
185                 for (int i = 0; i < len; i++) g[i] = (f[i] - g[i] + mod) % mod;
186                 g[0] = (g[0] + 1) % mod;
187                 g *= tmp;
188                 g.resize(len);
189             }
190             g.resize(size());
191             return g;
192         }
193     };
```

```
194
195     inline poly power(poly f, int b) { // 需要满足 F[0] = 1
196         f = f.ln();
197         for (int i = 0; i < (int)f.size(); i++) f[i] = (long long)f[i] * b % mod;
198         f = f.exp();
199         return f;
200     }
201     // 不要求 F[0] = 1 的多项式快速幂
202     /*
203     b1 为指数 mod m，b2 为指数 mod (phi(m) = mod - 1)
204     如果 b1 传入之前被取模了记得特判 a[0] = 0 且指数 > n 的情况，此时多项式每一位系数都
        ↪ 是 0
205     */
206     poly power(poly f, int b1, int b2 = -1) {
207         if (b2 == -1) b2 = b1;
208         int n = f.size(), p = 0;
209         reverse(f.begin(), f.end());
210         while (!f.empty() && !f.back()) f.pop_back(), p++;
211         if (f.empty() || (long long)p * b1 >= n) return poly(n);
212         int v = f.back();
213         int inv = power(v, mod - 2);
214         for (int &i : f) i = (long long)i * inv % mod;
215         reverse(f.begin(), f.end());
216         f = f.ln();
217         for (int &i : f) i = (long long)i * b1 % mod;
218         f = f.exp();
219         reverse(f.begin(), f.end());
220         for (int i = 1; i <= p * b1; i++) f.push_back(0);
221         reverse(f.begin(), f.end());
222         f.resize(n);
223         v = power(v, b2);
224         for (int &i : f) i = (long long)i * v % mod;
225         return f;
226     }
227 }
```

6.6 FWT

```
1 void FWTor(int a[], int op) {
2     for (int l = 2; l <= n; l <<= 1)
3         for (int i = 0; i < n; i += l)
4             for (int j = 0; j < l / 2; j++)
5                 a[i + j + l / 2] = (a[i + j + l / 2] + (long long)a[i + j] * op % mod
                      ↪ + mod) % mod;
6     return;
7 }
```

```
void FWTand(int a[], int op) {
    for (int l = 2; l <= n; l <<= 1)
        for (int i = 0; i < n; i += l)
            for (int j = 0; j < l / 2; j++)
                a[i + j] = (a[i + j] + (long long)a[i + j + l / 2] * op % mod + mod)
                ↪  % mod;
    return;
}
void FWTxor(int a[], int op) {
    for (int l = 2; l <= n; l <<= 1)
        for (int i = 0; i < n; i += l)
            for (int j = 0; j < l / 2; j++) {
                int p = (a[i + j] + a[i + j + l / 2]) % mod, d = (a[i + j] - a[i + j
                ↪  + l / 2] + mod) % mod;
                a[i + j] = (long long)p * op % mod;
                a[i + j + l / 2] = (long long)d * op % mod;
            }
    return;
}
```

Chapter 7

计算几何

7.1 计算几何模板

```
// == Preparations ==
#include <cmath>
#include <chrono>
#include <random>
#include <vector>
#include <algorithm>
// == Main ==
const double eps = 1e-9;
const double pi = acos(-1);

// 判断 x < 0 或 x = 0 或 x > 0
inline int sign(double x) {return fabs(x) <= eps ? 0 : (x > 0 ? 1 : -1);}

// 向量 & 点
struct Vect {
    double x, y;

    Vect() = default;
    Vect(double _x, double _y): x(_x), y(_y) {}
    Vect operator+() const {return *this;}
    Vect operator-() const {return Vect(-x, -y);}
    Vect operator+(const Vect &b) const {return Vect(x + b.x, y + b.y);}
    Vect operator-(const Vect &b) const {return Vect(x - b.x, y - b.y);}
    Vect operator*(const double &t) const {return Vect(x * t, y * t);}
    Vect operator/(const double &t) const {return Vect(x / t, y / t);}
    double operator*(const Vect &b) const {return x * b.x + y * b.y;}
    double operator^(const Vect &b) const {return x * b.y - y * b.x;}
    Vect &operator+=(const Vect &b) {return *this = *this + b;}
    Vect &operator-=(const Vect &b) {return *this = *this - b;}
    Vect &operator*=(const double &t) {return *this = *this * t;}
    Vect &operator/=(const double &t) {return *this = *this / t;}
```

```
32      bool operator==(const Vect &b) {return !sign(x - b.x) && !sign(y - b.y);}
33      double length() const {return sqrt(x * x + y * y);}
34      Vect normal() const {return *this / length();}
35  };
36  using Point = Vect;
37  // 多边形
38  struct Polygon: vector<Point> {
39      Polygon(initializer_list<Point> &&arg): vector<Point>(arg) {}
40      template<typename... argT>
41      explicit Polygon(argT &&...args): vector<Point>(forward<argT>(args)...) {}
42
43      // 多边形面积
44      double area() const {
45          double ans = 0;
46          for (int i = 0; i < (int)size(); i++) ans += (*this)[i] ^ (*this)[(i + 1) %
            ↪  size()];
47          return ans / 2;
48      }
49  };
50  // 直线
51  struct Line {
52      // 表示方式为起点 + 向量
53      Point s;
54      Vect dir;
55
56      Line() = default;
57      Line(Point _s, Vect _dir): s(_s), dir(_dir) {}
58  };
59  // 线段
60  struct Segment: Line {
61      Segment() = default;
62      Segment(Point _s, Vect _dir): Line(_s, _dir) {}
63  };
64  // 圆
65  struct Circle {
66      // 表示方式为圆心 + 半径
67      Point O;
68      double r;
69
70      Circle() = default;
71      Circle(Point _O, double _r): O(_O), r(_r) {}
72      double area() {return r * r * pi;}
73  };
74
75  // 两点的中点
76  inline Point middle(Point a, Point b) {return Point((a.x + b.x) / 2, (a.y + b.y) /
    ↪  2);}
77  // 向量 u 逆时针旋转 rad 后得到的向量
78  inline Vect rotate(Vect u, double rad) {
```

```
    return Vect(u.x * cos(rad) - u.y * sin(rad), u.x * sin(rad) + u.y * cos(rad));
}
// 两点距离
inline double dist(Point u, Point v) {return sqrt((u.x - v.x) * (u.x - v.x) + (u.y -
    v.y) * (u.y - v.y));}
// 两个向量的夹角
inline double angle(Vect a, Vect b) {return atan2(a ^ b, a * b);}
// 向量 a 到向量 b 的投影
inline Vect projection(Vect a, Vect b) {return b * (a * b / b.length() /
    b.length());}
// 点 a 到直线 b 的投影
inline Point projection(Point a, Line b) {return b.s + projection(a - b.s, b.dir);}
// 点 a 关于直线 b 的轴对称点
inline Point reflection(Point a, Line b) {return b.s + projection(a - b.s, b.dir) * 2
    - a;}
// 判断两条直线是否平行
inline int isParallel(Line a, Line b) {return !sign(a.dir ^ b.dir);}
// 判断两条直线是否相交
inline int isIntersecting(Line a, Line b) {return !isParallel(a, b);}
// 判断两条直线是否垂直
inline int isOrthogonal(Line a, Line b) {return !sign(a.dir * b.dir);}
// 判断点 a 是否在直线 b 上
inline int isOnLine(Point a, Line b) {return !sign(b.dir ^ (a - b.s));}
// 判断点 a 是否在线段 b 上
inline int isOnSegment(Point a, Segment b) {
    return isOnLine(a, b) && sign((a - b.s) * (a - b.s - b.dir)) <= 0;
}
// 判断直线 a 与线段 b 是否相交
inline int isIntersecting(Line a, Segment b) {
    int va = sign((b.s - a.s) ^ a.dir), vb = sign((b.s + b.dir - a.s) ^ a.dir);
    return !va || !vb || va != vb;
}
// 判断两条线段是否相交
inline int isIntersecting(Segment a, Segment b) {
    return isOnSegment(b.s, a) || isOnSegment(b.s + b.dir, a) ||
        isOnSegment(a.s, b) || isOnSegment(a.s + a.dir, b) ||
        (sign((b.s - a.s) ^ a.dir) != sign((b.s + b.dir - a.s) ^ a.dir) &&
            sign((a.s - b.s) ^ b.dir) != sign((a.s + a.dir - b.s) ^ b.dir));
}
// 直线交点
inline Point intersection(Line a, Line b) {return b.s + b.dir * (a.dir ^ (b.s - a.s))
    / (b.dir ^ a.dir);}
// 点 a 到直线 b 的距离
inline double dist(Point a, Line b) {return dist(a, projection(a, b));}
// 点 a 到线段 b 的距离
inline double dist(Point a, Segment b) {
    double ans = min(dist(a, b.s), dist(a, b.s + b.dir));
    Point tmp = projection(a, b);
    if (isOnSegment(tmp, b)) ans = min(ans, dist(a, tmp));
```

```
124     return ans;
125 }
126 // 线段间的距离
127 inline double dist(Segment a, Segment b) {
128     if (isIntersecting(a, b)) return 0;
129     return min({dist(a.s, b), dist(a.s + a.dir, b), dist(b.s, a), dist(b.s + b.dir,
        ↪ a)});
130 }
131 // 判断多边形是否是凸包
132 inline int isConvex(const Polygon &a) {
133     int n = a.size();
134     for (int i = 0; i < n; i++)
135         if (sign((a[(i + 1) % n] - a[i]) ^ (a[(i + 2) % n] - a[i])) < 0) return 0;
136     return 1;
137 }
138 // 判断点 a 与多边形 b 的位置关系，返回值：0 - 在多边形外，1 - 在多边形边上，2 - 在多边
    ↪ 形内部
139 inline int Containment(Point a, const Polygon &b) {
140     int n = b.size(), val = 0;
141     for (int i = 0; i < n; i++) {
142         Segment s = Segment(b[i], b[(i + 1) % n] - b[i]);
143         if (isOnSegment(a, s)) return 1;
144         int t = sign((b[(i + 1) % n] - b[i]) ^ (a - b[i]));
145         int va = sign(b[i].y - a.y), vb = sign(b[(i + 1) % n].y - a.y);
146         if (t > 0 && va <= 0 && vb > 0) val++;
147         if (t < 0 && vb <= 0 && va > 0) val--;
148     }
149     return val ? 2 : 0;
150 }
151 // 点集的凸包
152 inline Polygon Andrew(Polygon a) {
153     Polygon ans;
154     sort(a.begin(), a.end(), [](const Point &u, const Point &v) {
155         return u.x != v.x ? u.x < v.x : u.y < v.y;
156     });
157     ans.push_back(a[0]);
158     for (int i = 1; i < (int)a.size(); i++) {
159         while (ans.size() > 1) {
160             Point tmp = ans.back();
161             ans.pop_back();
162             if (sign((tmp - ans.back()) ^ (a[i] - tmp)) > 0) {ans.push_back(tmp);
                ↪ break;}
163         }
164         ans.push_back(a[i]);
165     }
166     int top = ans.size();
167     for (int i = (int)a.size() - 2; i >= 0; i--) {
168         while ((int)ans.size() > top) {
169             Point tmp = ans.back();
```

```
170             ans.pop_back();
171             if (sign((tmp - ans.back()) ^ (a[i] - tmp)) > 0) {ans.push_back(tmp);
                ↪  break;}
172         }
173         ans.push_back(a[i]);
174     }
175     ans.pop_back();
176     return ans;
177 }
178 // 两个凸包的闵可夫斯基和
179 inline Polygon Minkowski(Polygon l, Polygon r) {
180     Polygon diffl, diffr;
181     int sizl = l.size(), sizr = r.size();
182     for (int i = 1; i < sizl; i++) diffl.push_back(l[i] - l[i - 1]);
183     for (int i = 1; i < sizr; i++) diffr.push_back(r[i] - r[i - 1]);
184     Polygon ans;
185     ans.push_back(l[0] + r[0]);
186     int pl = 0, pr = 0;
187     while (pl < sizl - 1 && pr < sizr - 1)
188         if (sign(diffl[pl] ^ diffr[pr]) < 0) ans.push_back(ans.back() + diffl[pl++]);
189         else ans.push_back(ans.back() + diffr[pr++]);
190     while (pl < sizl - 1) ans.push_back(ans.back() + diffl[pl++]);
191     while (pr < sizr - 1) ans.push_back(ans.back() + diffr[pr++]);
192     return ans;
193 }
194 // 旋转卡壳求凸包直径
195 inline double Diameter(Polygon a) {
196     const int n = a.size();
197     if (n == 1) return 0;
198     if (n == 2) return dist(a[0], a[1]);
199     double ans = 0;
200     a.push_back(a[0]);
201     for (int i = 0, pos = 2; i < n; i++) {
202         Point u = a[i], v = a[i + 1];
203         ans = max(ans, dist(u, v));
204         while (sign(((u - a[pos + 1]) ^ (v - a[pos + 1])) - ((u - a[pos]) ^ (v -
            ↪  a[pos]))) >= 0)
205             pos = (pos + 1) % n;
206         ans = max({ans, dist(u, a[pos]), dist(v, a[pos])});
207     }
208     return ans;
209 }
210 // 扇形面积
211 inline double sector(double r, double rad) {return r * r * rad / 2;}
212 // 弧度为 rad 的弓形的面积
213 inline double arch(double r, double rad) {return sector(r, rad) - r * r * sin(rad) /
    ↪  2;}
214 // 三角形内切圆
215 inline Circle incircle(Point a, Point b, Point c) {
```

```
216     Line u(a, rotate(b - a, angle(b - a, c - a) / 2)), v(b, rotate(a - b, angle(a -
        ↪  b, c - b) / 2));
217     Circle ans;
218     ans.O = intersection(u, v);
219     ans.r = dist(ans.O, Line(a, b - a));
220     return ans;
221 }
222 // 三角形外接圆
223 inline Circle circumscribedCircle(Point a, Point b, Point c) {
224     Line u(middle(a, b), rotate(b - a, pi / 2)), v(middle(a, c), rotate(c - a, pi /
        ↪  2));
225     Circle ans;
226     ans.O = intersection(u, v);
227     ans.r = dist(ans.O, a);
228     return ans;
229 }
230 // 最小圆覆盖
231 inline Circle minimumCircle(Polygon a) {
232     mt19937 gen(chrono::system_clock::now().time_since_epoch().count());
233     int n = a.size();
234     shuffle(a.begin(), a.end(), gen);
235     Circle ans(Point(0, 0), 0);
236     for (int i = 0; i < n; i++)
237         if (sign(dist(ans.O, a[i]) - ans.r) > 0) {
238             ans = Circle(a[i], 0);
239             for (int j = 0; j < i; j++)
240                 if (sign(dist(ans.O, a[j]) - ans.r) > 0) {
241                     ans = Circle(middle(a[i], a[j]), dist(a[i], a[j]) / 2);
242                     for (int k = 1; k < j; k++)
243                         if (sign(dist(ans.O, a[k]) - ans.r) > 0)
244                             ans = circumscribedCircle(a[i], a[j], a[k]);
245                 }
246         }
247     return ans;
248 }
249 // 圆与直线的交点
250 inline pair<Point, Point> intersection(Circle a, Line b) {
251     Point p = projection(a.O, b);
252     double dis = dist(a.O, p);
253     dis = sqrt(a.r * a.r - dis * dis);
254     return {p - b.dir.normal() * dis, p + b.dir.normal() * dis};
255 }
256 // 两个圆的交点
257 inline pair<Point, Point> intersection(Circle a, Circle b) {
258     Vect u = b.O - a.O;
259     double rad = acos((a.r * a.r + u.length() * u.length() - b.r * b.r) / a.r /
        ↪  u.length() / 2);
260     u = u.normal() * a.r;
261     return {a.O + rotate(u, rad), a.O + rotate(u, -rad)};
```

```
}
// 点到圆的切点
inline pair<Point, Point> tangent(Point a, Circle b) {
    double rad = asin(b.r / dist(b.O, a));
    double dis = dist(b.O, a);
    dis = sqrt(dis * dis - b.r * b.r);
    Vect u = (b.O - a).normal() * dis;
    return {a + rotate(u, -rad), a + rotate(u, rad)};
}
// 两个圆的公切线，返回切点
inline vector<pair<Point, Point>> commonTangent(Circle a, Circle b) {
    vector<pair<Point, Point>> ans;
    double dis = dist(a.O, b.O);
    if (sign(dis - fabs(a.r - b.r)) < 0) return ans;
    else if (sign(dis - fabs(a.r - b.r)) == 0) {
        ans.push_back(intersection(a, b));
        return ans;
    }
    int flag = 0;
    if (a.r < b.r) swap(a, b), flag = 1;
    Vect u = (b.O - a.O).normal();
    double rad = acos((a.r - b.r) / dis);
    ans.emplace_back(a.O + rotate(u, rad) * a.r, b.O + rotate(-u, rad - pi) * b.r);
    ans.emplace_back(a.O + rotate(u, -rad) * a.r, b.O + rotate(-u, pi - rad) * b.r);
    if (sign(dis - a.r - b.r) >= 0) {
        rad = acos((a.r + b.r) / dis);
        if (sign(rad)) {
            ans.emplace_back(a.O + rotate(u, rad) * a.r, b.O + rotate(-u, rad) *
            ↪ b.r);
            ans.emplace_back(a.O + rotate(u, -rad) * a.r, b.O + rotate(-u, -rad) *
            ↪ b.r);
        } else ans.emplace_back(a.O + u * a.r, b.O - u * b.r);
    }
    if (flag)
        for (auto &i : ans) swap(i.first, i.second);
    return ans;
}
// 两个圆的交的面积
inline double intersectionArea(Circle a, Circle b) {
    int state = commonTangent(a, b).size();
    if (state <= 1) return min(a.area(), b.area());
    if (state >= 3) return 0;
    pair<Point, Point> tmp = intersection(a, b);
    double rada = fmod(angle(tmp.second - a.O, tmp.first - a.O) + 2 * pi, 2 * pi);
    double radb = fmod(angle(tmp.first - b.O, tmp.second - b.O) + 2 * pi, 2 * pi);
    return arch(a.r, rada) + arch(b.r, radb);
}
```

Chapter 8

杂项

8.1 自定义哈希

8.1.1 splitmix 和仿函数

```
// == Preparations ==
#include <chrono>
#include <random>

mt19937 gen(chrono::system_clock::now().time_since_epoch().count());
mt19937_64 genll(chrono::system_clock::now().time_since_epoch().count());
// == Main ==
unsigned splitmix32(unsigned x) {
    unsigned z = (x += 0x9E3779B9);
    z = (z ^ (z >> 16)) * 0x85ebca6b;
    z = (z ^ (z >> 13)) * 0xc2b2ae35;
    return z ^ (z >> 16);
}

unsigned long long splitmix64(unsigned long long x) {
    unsigned long long z = (x += 0x9e3779b97f4a7c15);
    z = (z ^ (z >> 30)) * 0xbf58476d1ce4e5b9;
    z = (z ^ (z >> 27)) * 0x94d049bb133111eb;
    return z ^ (z >> 31);
}

struct custom_hash { // custom hash for unsigned long long
    size_t operator()(unsigned long long x) const {
        return splitmix64(x + genll());
    }
};

struct custom_hash { // custom hash for pair<unsigned long long, unsigned long long>
    size_t operator()(pair<unsigned long long, unsigned long long> x) const {
```

```
        return (splitmix64(x.first + genll()) << 1) ^ splitmix64(x.second + genll());
    }
};
```

8.1.2 xorshift

```
// == Main ==
unsigned long long xorshift(unsigned long long x) { x ^= x << 13; x ^= x >> 7; x ^= x << 17; return x; }
unsigned xorshift(unsigned x) { x ^= x << 13; x ^= x >> 17; x ^= x << 5; return x; }
```

8.1.3 手写哈希表

```
// == Main ==
template<typename ValType , typename KeyType>
struct HashTable
{
    static const int M = 13075 , P = 13003; // P < M and P is between N^1.1 to N^1.7
    struct Node{KeyType key; ValType v; int nxt;}e[M];
    int tot , head[P];
    inline void Add(int u , const KeyType& k , const ValType &v){e[++tot] = {k , v , head[u]} , head[u] = tot;}
    inline int hash(const KeyType& x){return x;}
    ValType& operator[](const KeyType& x)
    {
        for(int i = head[hash(x) % P] ; i ; i = e[i].nxt)
            if(e[i].key == x)return e[i].v;
        Add(hash(x) % P , x , ValType());
        return e[tot].v;
    }
    ValType at(const KeyType& x)
    {
        for(int i = head[hash(x) % P] ; i ; i = e[i].nxt)
            if(e[i].key == x)return e[i].v;
        return ValType();
    }
    void clear()
    {
        for(int i = 1 ; i <= tot ; i++)
            head[hash(e[i].key) % P] = 0 , e[i] = {};
        tot = 0;
    }
};
```

8.2 取模类

```cpp
// == Main ==
struct mint {
    static const int mod = 998244353;
    int v;

    mint() = default;
    mint(int _v): v((_v % mod + mod) % mod) {}
    explicit operator int() const {return v;}
    mint operator+(const mint &x) const {return v + x.v - (v + x.v < mod ? 0 : mod);}
    mint &operator+=(const mint &x) {return *this = *this + x;}
    mint operator-(const mint &x) const {return v - x.v + (v - x.v >= 0 ? 0 : mod);}
    mint &operator-=(const mint &x) {return *this = *this - x;}
    mint operator*(const mint &x) const {return (long long)v * x.v % mod;}
    mint &operator*=(const mint &x) {return *this = *this * x;}
    mint inv() const {
        mint a(*this), ans(1);
        int b(mod - 2);
        while (b) {
            if (b & 1) ans *= a;
            a *= a;
            b >>= 1;
        }
        return ans;
    }
    mint operator/(const mint &x) const {return *this * x.inv();}
    mint &operator/=(const mint &x) {return *this = *this / x;}
    mint operator-() {return mint(-v);}
};
```

8.2.1 Barrett 约减

当模数不固定时可以加速。

用法：在构造函数中传模数，使用方法为 `F.reduce(x)`，其中 x 是需要取模的数。

```cpp
// == Main ==
struct Barrett {
    unsigned long long b, m;
    Barrett(unsigned long long b = 2): b(b), m((__uint128_t(1) << 64) / b) {}
    unsigned long long reduce(long long x) {
        unsigned long long r = (__uint128_t(x + b) * m) >> 64;
        unsigned long long q = (x + b) - b * r;
        return q >= b ? q - b : q;
    }
```

```
} F;
```

8.3 对拍脚本

```
#!/usr/bin/bash

declare -i num=0

while [ true ]; do
    ./mkdata > in.txt
    time ./mine < in.txt > out.txt
    ./correct < in.txt > ans.txt
    diff out.txt ans.txt
    if [ $? -ne 0 ]; then
        echo "WA"
        break
    fi
    num=num+1
    echo "Passed $num tests."
done
```

8.4 VS Code 配置

8.4.1 User Tasks

```
{
    // See https://go.microsoft.com/fwlink/?LinkId=733558
    // for the documentation about the tasks.json format
    "version": "2.0.0",
    "tasks": [
        {
            "type": "shell",
            "label": "My C++ Runner",
            "detail": "Build and Run Current C++ Program",
            "command": [ // 三个编译方式保留一个即可。
                "clear",
                "&&",
                "g++ ${file} -o ${fileDirname}/${fileBasenameNoExtension}
                ↪  -std=c++14 -Wall -Wextra && echo '== Normal =='",
                "g++ ${file} -o ${fileDirname}/${fileBasenameNoExtension}
                ↪  -std=c++14 -Wall -Wextra -O2 && echo '== O2 =='",
                "g++ ${file} -o ${fileDirname}/${fileBasenameNoExtension}
                ↪  -std=c++14 -Wall -Wextra -fsanitize=undefined,address && echo
                ↪  '== UB Check =='",
```

```
                "&&",
                "gnome-terminal -- bash -c \"ulimit -s 524288; time
                ↪  ${fileDirname}/${fileBasenameNoExtension}; read -p 'Press ENTER
                ↪  to continue...'; exit\""
            ],
            "problemMatcher": [ // 非必要
                "$gcc"
            ],
            "group": { // 非必要
                "kind": "build",
                "isDefault": true
            },
            "presentation": { // 非必要
                "showReuseMessage": false
            }
        }
    ]
}
```

8.4.2 设置

- 字体大小：16。（`"editor.fontSize": 16`）
- 添加多个光标的方式：ctrl。（`"editor.multiCursorModifier": "ctrlCmd"`）
- 不适用空格代替 Tab。（`"editor.insertSpaces": false`）
- 不允许 Enter 进行代码补全。（`"editor.acceptSuggestionOnEnter": "off"`）
- 标尺：110。（`"editor.rulers": [110]`）
- 平滑。（`"editor.cursorSmoothCaretAnimation": "on"`）
- 标题栏外观。（`"window.titleBarStyle": "custom"`）

totally：

```
{
    "editor.fontSize": 16,
    "editor.multiCursorModifier": "ctrlCmd",
    "editor.insertSpaces": false,
    "editor.acceptSuggestionOnEnter": "off",
    "editor.rulers": [110],
    "editor.cursorSmoothCaretAnimation": "on",
    "window.commandCenter": false
}
```

8.4.3 快捷键

- 切换块注释：Ctrl+Shift+A -> Ctrl+Shift+/
- 运行任务：Ctrl+Shift+B -> F11
- 向上移动行：Alt+up -> Ctrl+Shift+up
- 向下移动行：Alt+down -> Ctrl+Shift+down