ICPC Templates

我们需要更深入浅出一些

Mine_King, Tx_Lcy, 369pai

洛谷科技

October 20, 2024

目录

Chapter 1

字符串

1.1 最小表示法

```
// == 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，$\mathrm{pre}(s, i) \in \mathcal{B}(s) \iff |s| - i \in \mathcal{P}(s)$。

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

```
// == 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。

```
// == 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) {
```

```
                int id = c - 'a';
                if (!trie[now][id]) trie[now][id] = ++tot;
                now = trie[now][id];
            }
            return;
        }
    } tr;

    GSAM() {link[0] = -1;}
    int insert(int c, int lst) {
        int now = ++tot;
        len[now] = len[lst] + 1;
        for (int p = lst; p != -1; p = link[p])
            if (!delta[p][c]) delta[p][c] = now;
            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; i != -1 && delta[i][c] == q; i = link[i]) delta[i][c] = v;
                break;
            }
        return now;
    }
    void build() {
        queue<int> q;
        tr.st[0] = 0;
        q.push(0);
        while (!q.empty()) {
            int now = q.front();
            q.pop();
            for (int i = 0; i < 26; i++)
                if (tr.trie[now][i])
                    tr.st[tr.trie[now][i]] = insert(i, tr.st[now]), q.push(tr.trie[now][i]);
        }
        return;
    }
} 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

数论

2.1 Miller Rabin 和 Pollard Rho

2.1.1 Miller Rabin

```
// == 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;
}
```

2.1.2 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 ;
}
```

2.2 常用数论算法

2.2.1 exgcd

求出来的解满足 $|x| \le b, |y| \le a$。

```
// == 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;
}
```

2.2.2 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;
}
```

2.2.3 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;
}
```

2.2.4 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);
}
```

2.3 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);
}
```

Chapter 3

数据结构

3.1 K-D Tree

```cpp
// == Main ==
template<const int Dim = 2>
struct KDTree {
    using point = array<int, Dim>;
    struct node {
        point p, l, r;
        int val, siz, sum;
        node *ls, *rs;

        node() = default;
        node(point _p, int _val = 0):
            p(_p), l(_p), r(_p), val(_val), siz(1), sum(_val), ls(nullptr), rs(nullptr) {}
        void pushup() {
            l = r = p, siz = 1, sum = val;
            for (int i = 0; i < Dim; i++) {
                if (ls) l[i] = min(l[i], ls->l[i]), r[i] = max(r[i], ls->r[i]);
                if (rs) l[i] = min(l[i], rs->l[i]), r[i] = max(r[i], rs->r[i]);
            }
            if (ls) siz += ls->siz, sum += ls->sum;
            if (rs) siz += rs->siz, sum += rs->sum;
            return ;
        }
    };
    vector<node *> root;
    using itor = typename vector<node>::iterator;

    node *build(itor l, itor r, int dim = 0) {
        if (l == r) return nullptr;
        int mid = (r - l) / 2;
        nth_element(l, l + mid, r, [&dim](const node &x, const node &y) {return x.p[dim] < y.p[dim];});
```

```
        node *now = new node(*(l + mid));
        now->ls = build(l, l + mid, (dim + 1) % Dim);
        now->rs = build(l + mid + 1, r, (dim + 1) % Dim);
        now->pushup();
        return now;
    }
    void getnode(node *now, vector<node> &vec) {
        if (!now) return ;
        vec.push_back(*now);
        getnode(now->ls, vec), getnode(now->rs, vec);
        delete now;
        return ;
    }
    void insert(point p, int val) {
        vector<node> tmp({node(p, val)});
        while (!root.empty() && root.back()->siz == (int)tmp.size())
            getnode(root.back(), tmp), root.pop_back();
        sort(tmp.begin(), tmp.end(), [](const node &x, const node &y) {return x.p <
        ↪ y.p;});
        vector<node> vec;
        for (node i : tmp)
            if (!vec.empty() && vec.back().p == i.p) vec.back().val += i.val;
            else vec.push_back(i);
        root.push_back(build(vec.begin(), vec.end()));
        return ;
    }
    int query(point ll, point rr, node *now) {
        if (!now) return 0;
        int flag = 1;
        for (int i = 0; i < Dim; i++)
            if (now->r[i] < ll[i] || now->l[i] > rr[i]) return 0;
            else flag &= ll[i] <= now->l[i] && now->r[i] <= rr[i];
        if (flag) return now->sum;
        flag = 1;
        for (int i = 0; i < Dim; i++) flag &= ll[i] <= now->p[i] && now->p[i] <=
        ↪ rr[i];
        return flag * now->val + query(ll, rr, now->ls) + query(ll, rr, now->rs);
    }
    int query(point ll, point rr) {
        int ans = 0;
        for (node *rt : root) ans += query(ll, rr, rt);
        return ans;
    }
    ~KDTree() {
        vector<node> tmp;
        for (node *rt : root) getnode(rt, tmp);
    }
};
```

3.2 Link Cut Tree

代码维护的是点权异或和。

```
// == Main ==
struct LinkCutTree {
    int fa[100005], son[100005][2], siz[100005], swp[100005];
    int val[100005], Xor[100005];

    void pushup(int now) {
        siz[now] = siz[son[now][0]] + siz[son[now][1]] + 1;
        Xor[now] = Xor[son[now][0]] ^ val[now] ^ Xor[son[now][1]]; // 此处更新信息。
        return;
    }
    void pushdown(int now) {
        if(!swp[now]) return ;
        swap(son[now][0], son[now][1]);
        swp[son[now][0]] ^= 1, swp[son[now][1]] ^= 1;
        swp[now] = 0;
        // 此处将信息 pushdown
        return;
    }
    int isRoot(int now) {return now != son[fa[now]][0] && now != son[fa[now]][1];}
    int get(int now) {return now == son[fa[now]][1];}
    void rotate(int x) {
        int y = fa[x], z = fa[fa[x]], chk = get(x);
        if (!isRoot(y)) son[z][get(y)] = x;
        son[y][chk] = son[x][chk ^ 1], fa[son[x][chk ^ 1]] = y;
        son[x][chk ^ 1] = y, fa[y] = x;
        fa[x] = z;
        pushup(y), pushup(x);
        return;
    }
    void splay(int now) {
        vector<int> stk;
        stk.push_back(now);
        for (int i = now; !isRoot(i); i = fa[i]) stk.push_back(fa[i]);
        while (!stk.empty()) pushdown(stk.back()), stk.pop_back();
        for (int f; f = fa[now], !isRoot(now); rotate(now))
            if (!isRoot(f)) rotate(get(f) == get(now) ? f : now);
        return;
    }
    void access(int now) { // 打通到根的链
        for (int lst = 0; now; lst = now, now = fa[now]) splay(now), son[now][1] =
          ↪ lst, pushup(now);
        return;
    }
```

```
    void makeRoot(int now) {access(now); splay(now); swp[now] ^= 1; return;} // 设置根
    void link(int u, int v) {makeRoot(u); fa[u] = v; return;} // 连接
    void cut(int u, int v) {makeRoot(u); access(v); splay(v); son[v][0] = fa[u] = 0; return;} // 切割
    int find(int now) { // 找根
        access(now), splay(now);
        pushdown(now);
        while (son[now][0]) now = son[now][0], pushdown(now);
        splay(now);
        return now;
    }
    void split(int u, int v) {makeRoot(u); access(v); splay(u); return;} // 剖出 u ~ v 的链
    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。
    int query(int u, int v) {split(u, v); return Xor[u];} // 查询操作，split 后做就行了。
    int isConnected(int u, int v) {return find(u) == find(v);} // 查询两个点是否连通。
};
```

Chapter 4

图论

4.1 Tarjan

4.1.1 强连通分量

```cpp
// == 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 ;
}
```

4.1.2 割边与边双

割边：

```cpp
// == 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 ;
}
```

4.1.3 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 ;
}
```

4.2 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 ;
}
```

4.3 二分图匹配

4.3.1 最大匹配

```
// == 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);
}
```

4.3.2 最大权匹配

```
// == 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("");
```

4.4 网络流

4.4.1 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;
```

4.4.2 费用流

原始对偶：

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

```cpp
struct graph {
    int tot, hd[805];
    int nxt[30005], to[30005], flw[30005], cst[30005];

    graph() {tot = 1;}
    void add(int u, int v, int f, int c) {
        nxt[++tot] = hd[u];
        hd[u] = tot;
        to[tot] = v;
        flw[tot] = f;
        cst[tot] = c;
        return ;
    }
} g;
int h[805], dis[805], f[805], pre[805];
struct node {
    int id, val;

    node() = default;
    node(int _id, int _val): id(_id), val(_val) {}
    bool operator<(const node &x) const {return val > x.val;}
};

void add_edge(int u, int v, int f, int c) {g.add(u, v, f, c), g.add(v, u, 0, -c);
↪   return;}
void spfa() {
    queue<int> q;
    memset(h, 0x3f, sizeof(h));
    h[s] = 0;
    q.push(s);
    while (!q.empty()) {
        int now = q.front();
        q.pop();
        f[now] = 0;
        for (int i = g.hd[now]; i; i = g.nxt[i])
            if (g.flw[i] && h[g.to[i]] > h[now] + g.cst[i]) {
                h[g.to[i]] = h[now] + g.cst[i];
                if (!f[g.to[i]]) q.push(g.to[i]), f[g.to[i]] = 1;
            }
    }
    return ;
}
int dijkstra() {
    priority_queue<node> q;
    memset(dis, 0x3f, sizeof(dis));
    memset(pre, 0, sizeof(pre));
    q.emplace(s, dis[s] = 0);
    while (!q.empty()) {
        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;
```

4.4.3 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。

4.4.4 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。

4.5 k 短路

复杂度为 $O((n+m)\log n+k\log k)$。

```cpp
// == 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 ;
    }
```

```cpp
} 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);
}
```

4.6 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;
}
```

4.7 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;
```

```cpp
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 ;
```

```
}
```

4.8 弦图

4.8.1 MCS 最大势算法。

```
// == 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);
```

4.8.2 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。

4.8.3 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。

4.8.4 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。

4.8.5 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。

4.9 图计数相关

4.9.1 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）

4.9.2 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。

```cpp
// == 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。

```
// == 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 5

多项式

5.1 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 $F \pmod{x^n}$。

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。

5.2 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;
}
```

5.3 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	$17 \times 2^{27} + 1$

5.4 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;
                }
```

```
            if (mid == 1 << 10)
                for (int i = 0; i < n; i++) f[i] %= mod;
        }
        int inv = flag ? 1 : power(n, mod - 2);
        for (int i = 0; i < n; i++) g[i] = f[i] % mod * inv % mod;
        return;
    }

    // 下面是基于转置原理的 NTT，相对朴素版本效率更高。
    void NTT(poly &g, int flag) const {
        int n = g.size();
        vector<int> f(g.begin(), g.end());
        if (flag) {
            for (int mid = n >> 1; mid >= 1; mid >>= 1) {
                int w1 = power(G, (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)(f[i + j] - f[i + mid + j] + mod) * w[j] % mod;
                        f[i + j] = f[i + j] + f[i + mid + j] >= mod ?
                            f[i + j] + f[i + mid + j] - mod : f[i + j] + f[i + mid + j];
                        f[i + mid + j] = t;
                    }
            }
            for (int i = 0; i < n; i++) g[i] = f[i];
        } else {
            for (int mid = 1; mid < n; mid <<= 1) {
                int w1 = power(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 < 0 ? f[i + j] - t + mod : f[i + j] - t;
                        f[i + j] = f[i + j] + t >= mod ? f[i + j] + t - mod : f[i + j] + t;
                    }
            }
            int inv = power(n, mod - 2);
            for (int i = 0; i < n; i++) g[i] = (long long)f[i] * inv % mod;
        }
```

```
            return;
        }

        poly operator*(poly b) const {
            poly a(*this);
            int n = 1, len = (int)(a.size() + b.size()) - 1;
            while (n < len) n <<= 1;
            a.resize(n), b.resize(n);
            NTT(a, 1), NTT(b, 1);
            poly c(n);
            for (int i = 0; i < n; i++) c[i] = (long long)a[i] * b[i] % mod;
            NTT(c, 0);
            c.resize(len);
            return c;
        }
        poly operator*=(const poly &b) {return *this = *this * b;}
        poly inv() const {
            poly f = *this, g;
            g.push_back(power(f[0], mod - 2));
            int n = 1;
            while (n < (int)f.size()) n <<= 1;
            f.resize(n << 1);
            for (int len = 2; len <= n; len <<= 1) {
                poly tmp(len), ff(len << 1);
                for (int i = 0; i < len >> 1; i++) tmp[i] = g[i] * 2 % mod;
                for (int i = 0; i < len; i++) ff[i] = f[i];
                g.resize(len << 1);
                NTT(g, 1), NTT(ff, 1);
                for (int i = 0; i < len << 1; i++) g[i] = (long long)g[i] * g[i] % mod * ff[i] % mod;
                NTT(g, 0);
                g.resize(len);
                for (int i = 0; i < len; i++) g[i] = (tmp[i] - g[i] + mod) % mod;
            }
            g.resize(size());
            return g;
        }
        poly sqrt() const { // need F[0] = 1.
            poly f = *this, g;
            g.push_back(1);
            int n = 1;
            while (n < (int)f.size()) n <<= 1;
            f.resize(n << 1);
            for (int len = 2; len <= n; len <<= 1) {
                poly tmp(len), ff(len << 1);
                for (int i = 0; i < len >> 1; i++) tmp[i] = g[i] * 2 % mod;
                for (int i = 0; i < len; i++) ff[i] = f[i];
                g.resize(len << 1);
                NTT(g, 1);
```

```
            for (int i = 0; i < len << 1; i++) g[i] = (long long)g[i] * g[i] %
             ↪  mod;
            NTT(g, 0);
            g += ff;
            g *= tmp.inv();
            g.resize(len);
        }
        g.resize(size());
        return g;
    }
    poly derivative() const {
        poly f(*this);
        for (int i = 1; i < (int)f.size(); i++) f[i - 1] = (long long)f[i] * i %
         ↪  mod;
        f.pop_back();
        return f;
    }
    poly integral() const {
        poly f(*this);
        f.push_back(0);
        for (int i = f.size() - 1; i >= 1; i--) f[i] = (long long)f[i - 1] *
         ↪  power(i, mod - 2) % mod;
        f[0] = 0;
        return f;
    }
    poly ln() const {
        poly f((derivative() * inv()).integral());
        f.resize(size());
        return f;
    }
    poly exp() const { // 需要满足 F[0] = 0
        poly f(*this), g;
        g.push_back(1);
        int n = 1;
        while (n < (int)size()) n <<= 1;
        f.resize(n);
        for (int len = 2; len <= n; len <<= 1) {
            poly tmp(g);
            g.resize(len);
            g = g.ln();
            for (int i = 0; i < len; i++) g[i] = (f[i] - g[i] + mod) % mod;
            g[0] = (g[0] + 1) % mod;
            g *= tmp;
            g.resize(len);
        }
        g.resize(size());
        return g;
    }
};
```

```cpp

    inline poly power(poly f, int b) { // 需要满足 F[0] = 1
        f = f.ln();
        for (int i = 0; i < (int)f.size(); i++) f[i] = (long long)f[i] * b % mod;
        f = f.exp();
        return f;
    }
    // 不要求 F[0] = 1 的多项式快速幂，但是我忘记怎么用了，记得去回顾一下！
    poly power(poly f, int b1, int b2 = -1) {
        if (b2 == -1) b2 = b1;
        int n = f.size(), p = 0;
        reverse(f.begin(), f.end());
        while (!f.empty() && !f.back()) f.pop_back(), p++;
        if (f.empty() || (long long)p * b1 >= n) return poly(n);
        int v = f.back();
        int inv = power(v, mod - 2);
        for (int &i : f) i = (long long)i * inv % mod;
        reverse(f.begin(), f.end());
        f = f.ln();
        for (int &i : f) i = (long long)i * b1 % mod;
        f = f.exp();
        reverse(f.begin(), f.end());
        for (int i = 1; i <= p * b1; i++) f.push_back(0);
        reverse(f.begin(), f.end());
        f.resize(n);
        v = power(v, b2);
        for (int &i : f) i = (long long)i * v % mod;
        return f;
    }
}
```

Chapter 6

杂项

6.1 取模类

```
// == 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);}
};
```

6.1.1 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;
```

6.2 对拍脚本

```bash
#!/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
```

6.3 VS Code 配置

6.3.1 User Tasks

```json
{
    // 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
            }
        }
    ]
}
```

6.3.2 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
}
```

6.3.3 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