算法竞赛小记（代码版）

Competitive Programming Library

Calculatelove



github.com/Calculatelove/cp-library-v4-LaTeX

LAST COMMIT

2026-11-19 b71bfa3

feat: 2026 CCPC 厦门站前更新

目录

0x11 基础 Misc 5

常用公式 5
日期公式 5
xorshift 5
xorsum 5
常用配置 5
makefile 5
pragma 5
ulimit 5
Linux 对拍脚本 5
Windows 对拍程序 6
随机数据生成 6
生成区间 $[l,r]$ 中的随机整数 6
生成区间 $[l,r)$ 中的随机浮点数 6
生成互不相同的随机整数 6
生成随机树 6
常用库函数重载 7
i128 7
整数除法精确取整 7
取最值 7
取模 8
精确开根号 8
精确取对数 8
快读 8
超级快读 9
高精度 9
分数类 11

0x12 基础 算法 11

快速乘 11
正道：i128 11
魔道：long double 11
光速幂 11

0x21 字符串 I 12

kmp 12
Z Algorithm 12
manacher 13
Lyndon 分解 13
最小表示法 14

0x22 字符串 II 14

Trie 14
AC 自动机 15
SA 17
SAM 18
PAM 21
Palindrome Series 22

0x31 数学 多项式 23

拉格朗日插值 23
FFT 24
NTT 25
MTT 25
三模 NTT 25
拆系数 FFT 27
多项式多点求值 28

多项式全家桶 28
Bostan-Mori 33
分式远处系数 33
常系数齐次线性递推 33
Fiduccia 算法 33
递推式转分式 34
整式递推 34
应用：快速阶乘算法 34
Power Projection 35

0x32 数学 集合幂级数 36

FWT 36
位运算卷积 36
子集 dp 36
子集卷积 37

0x33 数学 组合计数 37

组合数常见求法 37
求法 1：递推 37
求法 2：预处理阶乘与阶乘逆元 38
图论计数 38
有标号无向连通图计数 38
有标号无向欧拉图计数 38
有标号无向二分图计数 39
有标号 DAG 计数 39
有标号弱连通 DAG 计数 39
有标号强连通图计数 40

0x34 数学 初等数论–质数 40

质数 40
埃氏筛 40
线性筛 40
狄利克雷前缀和 41
Pollard-Rho 质因数分解 41

0x34 数学 初等数论–约数 42

最大公约数常见求法 42
求法 1：辗转相除法 42
求法 2：更相减损术 42
求法 3：递推 42
求法 4：基于值域预处理的快速 gcd 43
类欧几里得算法 43

0x34 数学 初等数论–取整 44

数论分块 44
下取整数论分块 44
上取整数论分块 44
多维数论分块 44
积性函数与狄利克雷卷积 44
线性筛筛积性函数 44
杜教筛 45

0x34 数学 初等数论–同余 45

模意义下逆元 45
单个逆元的求解 45
预处理线性逆元 45
线性同余方程 46
线性同余方程组 46
阶与原根 46
高次同余方程 48

0x41 数据结构 序列结构 48

ST 表 48
并查集 49

线段树 49
势能线段树 49

0x65 图论 二分图与网络流 51

最大流 51
Dinic 51

0x71 计算几何 I 52

计算几何基础 52

0x72 计算几何 II 52

凸壳 52
凸包 52
闵可夫斯基和 53

0x11 基础 Misc

常用公式

日期公式

0x11 日期公式.cpp

```cpp
// 返回从 1/1/1 到 y/m/d 的天数，由于 1/1/1 是周一，所以 mod 7 可以算出是周几
int getday(int y, int m, int d) {
    if (m < 3) y --, m += 12;
    return (365 * y + y / 4 - y / 100 + y / 400 + (153 * (m - 3) + 2) / 5 + d
     ↪  - 307);
}

// 返回从 1/1/1 开始经过 n 天的日期（getday 的逆函数）
std::array<int, 3> date(int n) {
    int y, m, d;
    n += 429 + ((4 * n + 1227) / 146097 + 1) * 3 / 4;
    y = (4 * n - 489) / 1461;
    n -= y * 1461 / 4;
    m = (5 * n - 1) / 153;
    d = n - m * 153 / 5;
    if (-- m > 12) m -= 12, y ++;
    return {y, m, d};
}
```

xorshift

0x11 xorshift.cpp

```cpp
u64 mask = std::mt19937_64(std::random_device{}())();
u64 xorshift(u64 x) {
    x ^= mask;
    x ^= x << 13, x ^= x >> 7, x ^= x << 17;
    x ^= mask;
    return x;
}
```

xorsum

0x11 xorsum.cpp

```cpp
i64 xorsum(i64 n) {
    int r = n % 4;
    if (r == 0) {
        return n;
    } else if (r == 1) {
        return 1;
    } else if (r == 2) {
        return n + 1;
    } else {
        return 0;
    }
}
```

常用配置

makefile

```
CXXFLAGS = -std=c++23 -O2 -Wall -fsanitize=undefined

# -D_GLIBCXX_DEBUG : STL debug mode
# -fsanitize=address : 内存错误检查
# -fsanitize=undefined : UB 检查
```

pragma

```cpp
#pragma GCC optimize("O3,unroll-loops")
#pragma GCC target("avx2")
```

ulimit

```
ulimit -s 1048576 # 单位 KB
```

Linux 对拍脚本

```
while true; do
    ./gen >0.in
    ./obj <0.in >1.out
    ./std <0.in >2.out
    let cnt=cnt+1; echo "Test $cnt:"
```

```
    if diff 1.out 2.out; then
        echo ac
    else
        echo wa
        break;
    fi
done
```

Windows 对拍程序

```
#include <bits/stdc++.h>
int main() {
    int T = 100;
    for (int i = 1; i <= T; i ++) {
        system("gen.exe >0.in");
        system("obj.exe <0.in >1.out");
        system("std.exe <0.in >2.out");
        std::cout << "Test " << i << ":\n";
        if (system("fc 1.out 2.out")) {
            std::cout << "wa\n";
            break;
        } else {
            std::cout << "ac\n";
        }
    }
}
```

随机数据生成

生成区间 $[l,r]$ 中的随机整数

0x11 生成区间 [l,r] 中的随机整数.cpp

```
std::mt19937_64 mtrand{std::random_device{}()};

// 生成区间 [l, r] 中的随机整数
int rand(int l, int r) {
    std::uniform_int_distribution<int> range(l, r);
    return range(mtrand);
}
```

生成区间 $[l,r)$ 中的随机浮点数

0x11 生成区间 [l,r) 中的随机浮点数.cpp

```
std::mt19937_64 mtrand{std::random_device{}()};

// 生成区间 [l, r) 中的随机浮点数
double rand(double l, double r) {
    std::uniform_real_distribution<double> range(l, r);
    return range(mtrand);
}
```

生成互不相同的随机整数

0x11 生成互不相同的随机整数.cpp

```
// 生成 n 个在区间 [l, r] 中互不相同的随机整数
auto GenSeq(int n, int l, int r) {
    assert(n <= r - l + 1);
    std::map<int, int> buc;
    std::vector<int> res(n);
    for (int i = 0; i < n; i ++) {
        do {
            res[i] = rand(l, r);
        } while (buc.contains(res[i]));
        buc[res[i]] = 1;
    }
    return res;
}
```

生成随机树

0x11 生成随机树.cpp

```
// 生成 n 个点的随机树
auto GenTree(int n) {
    std::vector<std::pair<int, int>> res;
    std::vector<int> prufer(n - 1), deg(n + 1, 1);
    for (int i = 1; i <= n - 2; i ++) {
        prufer[i] = rand(1, n);
        deg[prufer[i]] ++;
    }
    int leaf = 0, p = 0;
    for (int i = 1; i <= n; i ++) {
```

```
        if (deg[i] == 1) {
            leaf = p = i;
            break;
        }
    }
    for (int i = 1; i <= n - 2; i ++) {
        int x = prufer[i];
        res.push_back({leaf, x});
        if (-- deg[x] == 1 && x < p) {
            leaf = x;
        } else {
            p ++;
            while (deg[p] != 1) p ++;
            leaf = p;
        }
    }
    if (n > 1) {
        res.push_back({leaf, n});
    }
    return res;
}
```

常用库函数重载

i128

0x11 i128.cpp

```
using i128 = __int128;

// 重载 i128 标准输入流
std::istream &operator >> (std::istream &is, i128 &x) {
    std::string s;
    is >> s;
    int n = s.length(), f = s[0] == '-';
    x = 0;
    for (int i = f; i < n; i ++) {
        x = x * 10 + s[i] - '0';
    }
    if (f) x = -x;
    return is;
}
// 重载 i128 标准输出流
std::ostream &operator << (std::ostream &os, i128 x) {
    if (x < 0) x = -x, os << '-';
    std::string s;
    do {
        s += char(x % 10 + '0'), x /= 10;
    } while (x);
    std::reverse(s.begin(), s.end());
    return os << s;
}
```

整数除法精确取整

0x11 整数除法精确取整.cpp

```
// 整数除法 精确上取整
i64 ceilDiv(i64 n, i64 m) {
    // assert(m);
    if (m < 0) n = -n, m = -m;
    return n >= 0 ? (n + m - 1) / m : n / m;
}

// 整数除法 精确下取整
i64 floorDiv(i64 n, i64 m) {
    // assert(m);
    if (m < 0) n = -n, m = -m;
    return n >= 0 ? n / m : (n - m + 1) / m;
}
```

取最值

0x11 取最值.cpp

```
template <class T>
inline void chmin(T &x, const T &y) {
    if (x > y) {
        x = y;
    }
}
template <class T>
inline void chmax(T &x, const T &y) {
    if (x < y) {
        x = y;
    }
}
```

取模

0x11 取模.cpp

```cpp
const int mod = 998244353; // 模数需要根据实际问题调整

template <class T>
inline int norm(T x) {
    x %= mod;
    return x < 0 ? x + mod : x;
}

inline void add(int &x, const int &y) {
    x += y; if (x >= mod) x -= mod;
}
inline void dec(int &x, const int &y) {
    x -= y; if (x < 0) x += mod;
}
inline void mul(int &x, const int &y) {
    x = 1ll * x * y % mod;
}
inline void neg(int &x) {
    if (x) x = mod - x;
}

template <class T>
constexpr int qpow(int a, T b, int p) {
    int ans = 1;
    for (; b; b >>= 1) {
        if (b & 1) ans = 1ll * ans * a % p;
        a = 1ll * a * a % p;
    }
    return ans;
}
```

精确开根号

0x11 精确开根号.cpp

```cpp
// 精确开根号：找到最大整数 d，使得 d * d <= n
i64 isqrt(i64 n) {
    i64 d = sqrt(n);
    while (d * d > n) {
        d --;
    }
    while ((d + 1) * (d + 1) <= n) {
        d ++;
    }
    return d;
}
```

精确取对数

0x11 精确取对数.cpp

```cpp
// 精确取对数（下取整）：找到最大整数 t，使得 a^t <= b
int ilog(i64 a, i64 b) {
    int t = 0;
    i64 v = 1;
    while (v <= b / a) {
        t ++;
        v *= a;
    }
    return t;
}

// 精确取对数（上取整）：找到最小整数 t，使得 a^t >= b
int iLog(i64 a, i64 b) {
    int t = 0;
    i64 v = 1;
    while (v < b) {
        t ++; if (v > b / a) break;
        v *= a;
    }
    return t;
}
```

快读

0x11 快读.cpp

```cpp
// 读入一个整数（无需在意类型）
template <class T>
inline void read(T &x) {
    char s;
    bool opt;
    while (s = getchar(), (s < '0' || s > '9') && s != '-');
    x = (opt = s == '-') ? 0 : s - '0';
    while (s = getchar(), s >= '0' && s <= '9') x = x * 10 + s - '0';
```

```
    if (opt) x = -x;
}
```

超级快读

0x11 超级快读.cpp

```cpp
/**
 * 注意，此代码的所有输出类函数，均不带“空格”与“换行”
 * 注意，虽然此代码使用了缓冲区，但不需要你手动清空，fastio 的析构函数会自动清空
 * 注意，此快读快写不能与 C/C++ 风格的 IO 混用，否则会导致 IO 的顺序混乱
 * 注意，如果你需要使用控制台调试，请在输入的最后使用 Ctrl+Z (Windows) 或 Ctrl+D (Linux/macOS) 手动输入 EOF 来结束输入
 */
struct fastio {
    static const int N = 1 << 20;

    char buf[N], *p1 = buf, *p2 = buf;
    #define gc() (p1 == p2 && (p2 = (p1 = buf) + fread(buf, 1, N, stdin), p1 == p2) ? EOF : *p1 ++)

    char pbuf[N], *pp = pbuf;
    void pc(const char &c) {
        if (pp - pbuf == N) fwrite(pbuf, 1, N, stdout), pp = pbuf;
        *pp ++ = c;
    }

    // 读入一个整数（无需在意类型）
    template <class T>
    void read(T &x) {
        char s;
        bool opt;
        while (s = gc(), (s < '0' || s > '9') && s != '-');
        x = (opt = s == '-') ? 0 : s - '0';
        while (s = gc(), s >= '0' && s <= '9') x = x * 10 + s - '0';
        if (opt) x = -x;
    }
    // 读入一个字符串
    void readString(std::string &x) {
        char s;
        while (s = gc(), std::isspace(s));
        x = s;
        while (s = gc(), !std::isspace(s)) x += s;
    }

    // 输出一个整数（无需在意类型）
    template <class T>
    void print(T x) {
        int top = 0; static int stk[40];
        if (x < 0) x = -x, pc('-');
        do {
            stk[++ top] = x % 10, x /= 10;
        } while (x);
        while (top) pc(stk[top --] + '0');
    }
    // 输出一个字符串（char 数组类型）
    void puts(const char *s) {
        while (*s) pc(*s), s ++;
    }
    // 输出一个字符串（std::string 类型）
    void puts(const std::string &s) {
        for (char ch : s) pc(ch);
    }

    ~fastio() {
        fwrite(pbuf, pp - pbuf, 1, stdout);
    }
} io;
```

高精度

0x11 高精度.cpp

```cpp
// 高精度
struct bint {
    static const int SIZE = 4000; // 位数上限需要根据实际问题调整
    int len, num[SIZE];

    explicit bint(int len = 1) : len(len) {
        memset(num, 0, sizeof(num));
    };

    auto operator <=> (const bint &b) const {
        if (len ^ b.len) return len <=> b.len;
        for (int i = len; i >= 1; i --)
            if (num[i] ^ b.num[i]) return num[i] <=> b.num[i];
        return std::strong_ordering::equal;
    }
    auto operator == (const bint &b) const {
```

```
        if (len ^ b.len) return false;
        for (int i = len; i >= 1; i --)
            if (num[i] ^ b.num[i]) return false;
        return true;
    }

    // 高精度加法
    bint operator + (const bint &b) const {
        bint c(std::max(len, b.len));
        for (int i = 1; i <= c.len; i ++) c.num[i] = num[i] + b.num[i];
        for (int i = 1; i <= c.len; i ++)
            if (c.num[i] >= 10) c.num[i + 1] ++, c.num[i] -= 10;
        while (c.num[c.len + 1]) {
            c.len ++;
            if (c.num[c.len] >= 10) c.num[c.len + 1] ++, c.num[c.len] -= 10;
        }
        return c;
    }
    // 高精度减法
    bint operator - (const bint &b) const {
        bint c(len);
        for (int i = 1; i <= c.len; i ++) c.num[i] = num[i] - b.num[i];
        for (int i = 1; i <= c.len; i ++)
            if (c.num[i] < 0) c.num[i + 1] --, c.num[i] += 10;
        while (c.len > 1 && !c.num[c.len]) c.len --;
        return c;
    }

    // 高精度乘以低精度
    bint operator * (const int &b) const {
        bint c(len);
        i64 x = 0;
        for (int i = 1; i <= len; i ++) {
            x += 1ll * num[i] * b;
            c.num[i] = x % 10, x /= 10;
        }
        while (x) {
            c.len ++;
            c.num[c.len] = x % 10, x /= 10;
        }
        while (c.len > 1 && !c.num[c.len]) c.len --;
        return c;
    }
    // 高精度乘以高精度
    bint operator * (const bint &b) const {
        bint c(len + b.len);
        for (int i = 1; i <= len; i ++) {
            for (int j = 1; j <= b.len; j ++) {
                c.num[i + j - 1] += num[i] * b.num[j];
            }
        }
        for (int i = 1; i <= c.len; i ++) {
            c.num[i + 1] += c.num[i] / 10;
            c.num[i] %= 10;
        }
        while (c.len > 1 && !c.num[c.len]) c.len --;
        return c;
    }

    // 高精度除以低精度（带余除法）
    std::pair<bint, int> divmod(const int &b) const {
        bint c(len);
        i64 r = 0;
        for (int i = len; i >= 1; i --) {
            c.num[i] = (r * 10 + num[i]) / b;
            r = (r * 10 + num[i]) % b;
        }
        while (c.len > 1 && !c.num[c.len]) c.len --;
        return {c, r};
    }
    // 高精度除以高精度（带余除法）
    std::pair<bint, bint> divmod(const bint &b) const {
        if (*this < b) return {bint(), *this};
        bint c(len - b.len + 1);
        bint r = *this;
        for (int i = c.len; i >= 1; i --) {
            bint tmp(b.len + i - 1);
            for (int j = 1; j <= b.len; j ++) {
                tmp.num[j + i - 1] = b.num[j];
            }
            while (tmp <= r) {
                r = r - tmp, c.num[i] ++;
            }
        }
        while (c.len > 1 && !c.num[c.len]) c.len --;
        return {c, r};
    }

    friend std::istream &operator >> (std::istream &is, bint &b) { // 默认没有前导 0
        std::string s;
        is >> s;
```

```
        b.len = s.length();
        memset(b.num, 0, sizeof(b.num));
        for (int i = 0; i < b.len; i ++) {
            b.num[b.len - i] = s[i] - '0';
        }
        return is;
    }
    friend std::ostream &operator << (std::ostream &os, const bint &b) {
        for (int i = b.len; i >= 1; i --) {
            os << b.num[i];
        }
        return os;
    }
};
```

分数类

0x11 分数类.cpp

```
// 分数类
struct frac {
    i64 p, q;
    frac() : p(0), q(1) {}
    frac(i64 p, i64 q) : p(p), q(q) {} // 必要时，在此处规范化分数（保证 q>0 以及 p,q 最简）

    frac operator + (const frac &rhs) const {
        return frac(p * rhs.q + rhs.p * q, q * rhs.q);
    }
    frac operator - (const frac &rhs) const {
        return frac(p * rhs.q - rhs.p * q, q * rhs.q);
    }
    frac operator * (const frac &rhs) const {
        return frac(p * rhs.p, q * rhs.q);
    }
    frac operator / (const frac &rhs) const {
        return frac(p * rhs.q, q * rhs.p);
    }

    auto operator <=> (const frac &rhs) const { // 比较大小时，必须保证 q>0
        return i128(p) * rhs.q <=> i128(rhs.p) * q;
    }
    auto operator == (const frac &rhs) const {
        return i128(p) * rhs.q == i128(rhs.p) * q;
    }

    friend std::ostream &operator << (std::ostream &os, frac x) {
        i64 g = std::gcd(x.p, x.q);
        return os << x.p / g << '/' << x.q / g;
    }
};
```

0x12 基础 算法

快速乘

正道：i128

0x12 快速乘（i128）.cpp

```
constexpr i64 qmul(i64 a, i64 b, i64 p) {
    return i128(a) * b % p;
}
```

魔道：long double

0x12 快速乘（long double）.cpp

```
constexpr i64 qmul(i64 a, i64 b, i64 p) {
    i64 c = static_cast<long double>(a) * b / p + 1e-8;
    i64 ans = a * b - c * p;
    if (ans < 0) ans += p;
    if (ans >= p) ans -= p;
    return ans;
}
```

光速幂

0x12 光速幂.cpp

```
// 光速幂
struct flashPower {
    int b;
    std::vector<int> w1, w2;

    flashPower() {}
```

```
    flashPower(int a, int n) {
        init(a, n);
    }

    void init(int a, int n) {
        b = sqrt(n) + 1;
        w1.resize(b + 1), w2.resize(b + 1);

        w1[0] = 1;
        for (int i = 1; i <= b; i ++) {
            w1[i] = 1ll * w1[i - 1] * a % mod;
        }

        w2[0] = 1;
        for (int i = 1; i <= b; i ++) {
            w2[i] = 1ll * w2[i - 1] * w1[b] % mod;
        }
    }

    int pow(int n) {
        return 1ll * w2[n / b] * w1[n % b] % mod;
    }
} fp;
```

0x21 字符串 I

kmp

0x21 kmp.cpp

```
// net[i]: s[1:i] 真前缀与真后缀匹配的最大长度（s[1:i] 的 border）
auto kmp(const auto &s) {
    int n = s.size() - 1; // s 下标从 1 开始
    std::vector<int> net(n + 1);
    net[1] = 0;
    for (int i = 2, j = 0; i <= n; i ++) {
        while (j > 0 && s[j + 1] != s[i]) j = net[j];
        if (s[j + 1] == s[i]) j ++;
        net[i] = j;
    }
    return net;
}

// f[i]: t[1:i] 后缀与 s 前缀匹配的最大长度
auto kmp_match(const auto &s, const auto &t, const auto &net) {
    int n = s.size() - 1, m = t.size() - 1; // s, t 下标从 1 开始
    std::vector<int> f(m + 1);
    for (int i = 1, j = 0; i <= m; i ++) {
        while (j > 0 && (j == n || s[j + 1] != t[i])) j = net[j];
        if (s[j + 1] == t[i]) j ++;
        f[i] = j;
    }
    return f;
}
```

Z Algorithm

0x21 Z-Algorithm.cpp

```
// Z[i]: lcp(s, s[i:n])
auto Zalgo(const auto &s) {
    int n = s.size() - 1; // s 下标从 1 开始
    std::vector<int> Z(n + 1);
    Z[1] = n;
    for (int i = 2, l = 0, r = 0; i <= n; i ++) {
        int &x = Z[i] = i <= r ? std::min(Z[i - l + 1], r - i + 1) : 0;
        while (i + x <= n && s[1 + x] == s[i + x]) {
            x ++;
        }
        if (i + x - 1 > r) {
            l = i, r = i + x - 1;
        }
    }
    return Z;
}

// p[i]: lcp(s, t[i:m])
auto Zalgo_match(const auto &s, const auto &t, const auto &Z) {
    int n = s.size() - 1, m = t.size() - 1; // s, t 下标从 1 开始
    std::vector<int> p(m + 1);
    for (int i = 1, l = 0, r = 0; i <= m; i ++) {
        int &x = p[i] = i <= r ? std::min(Z[i - l + 1], r - i + 1) : 0;
        while (1 + x <= n && i + x <= m && s[1 + x] == t[i + x]) {
            x ++;
        }
        if (i + x - 1 > r) {
            l = i, r = i + x - 1;
        }
```

```
    }
    return p;
}
```

manacher

0x21 manacher.cpp

```cpp
// 转化为形如 #a#b#c# 的字符串
auto manacher_init(const std::string &s) {
    std::string t = "#";
    for (char ch : s) {
        t += ch, t += "#";
    }
    return t;
}

// p[i]: 区间 [i - p[i] + 1, i + p[i] - 1] 回文
auto manacher(const auto &s) {
    int n = s.size() - 1; // s 下标从 1 开始
    std::vector<int> p(n + 1);
    for (int i = 1, mid = 0, r = 1; i <= n; i ++) {
        int &x = p[i] = i < r ? std::min(r - i, p[mid - i]) : 0;
        while (i - x > 0 && i + x <= n && s[i - x] == s[i + x]) {
            x ++;
        }
        if (i + x > r) {
            mid = i * 2, r = i + x;
        }
    }
    return p;
}

// 以位置 i 为回文中心的最长回文子串，在原串中的长度为 p[i] - 1（转换成 #a#b#c# 形式后）
// 判断原串区间 [l, r] 是否回文：p[l + r] - 1 >= r - l + 1（转换成 #a#b#c# 形式后）
```

0x21 manacher（奇偶分开考虑）.cpp

```cpp
// p[i]: 区间 [i - p[i] + 1, i + p[i] - !is_even] 回文，奇偶分开考虑
auto manacher(const auto &s, int is_even) {
    int n = s.size() - 1; // s 下标从 1 开始
    std::vector<int> p(n + 1);
    for (int i = 1, mid = 0, r = 1; i <= n - is_even; i ++) {
        int &x = p[i] = i < r ? std::min(r - i, p[mid - i]) : 0;
        while (i - x > 0 && i + x + is_even <= n && s[i - x] == s[i + x + is_even]) {
            x ++;
        }
        if (i + x > r) {
            mid = i * 2, r = i + x;
        }
    }
    return p;
}
```

Lyndon 分解

0x21 duval.cpp

```cpp
// Lyndon 分解
auto duval(const auto &s) {
    int n = s.size() - 1; // s 下标从 1 开始
    std::vector<int> res;
    for (int i = 1; i <= n;) {
        int j = i + 1, k = i;
        for (; j <= n && s[k] <= s[j]; j ++) {
            if (s[k] < s[j]) {
                k = i;
            } else {
                k ++;
            }
        }
        for (; i <= k; i += j - k) {
            res.push_back(i);
        }
    }
    return res;
}
```

最小表示法

0x21 最小表示法.cpp

```cpp
// 最小表示法
// std::rotate(s.begin(), s.begin() + minrep(s), s.end());
int minrep(const auto &s) {
    int n = s.size(), k = 0, i = 0, j = 1; // s 下标从 0 开始
    while (k < n && i < n && j < n) {
        int d = s[(i + k) % n] - s[(j + k) % n];
        if (d == 0) {
            k ++;
        } else {
            (d > 0 ? i : j) += k + 1;
            if (i == j) i ++;
            k = 0;
        }
    }
    return std::min(i, j);
}
```

0x22 字符串 II

Trie

0x22 01 trie.cpp

```cpp
// 01 trie
namespace trie {
    const int pond = 10001000; // 内存池大小，需要根据实际问题调整

    int nodeCount;
    struct node {
        int trans[2];
        int cnt;
        #define lc trans[0]
        #define rc trans[1]
    } t[pond];

    int newnode() {
        int p = ++ nodeCount;
        t[p].lc = t[p].rc = t[p].cnt = 0;
        return p;
    }

    void insert(int &p, int d, int x) {
        if (!p) p = newnode();
        t[p].cnt ++;
        if (d < 0) return;
        int v = x >> d & 1;
        insert(t[p].trans[v], d - 1, x);
    }

    int merge(int p, int q) {
        if (!p || !q) return p ^ q;
        t[p].cnt += t[q].cnt;
        t[p].lc = merge(t[p].lc, t[q].lc);
        t[p].rc = merge(t[p].rc, t[q].rc);
        return p;
    }

    // 可持久化 01 trie 合并
    // int merge(int p, int q) {
    //     if (!p || !q) return p ^ q;
    //     int u = newnode();
    //     t[u].cnt = t[p].cnt + t[q].cnt;
    //     t[u].lc = merge(t[p].lc, t[q].lc);
    //     t[u].rc = merge(t[p].rc, t[q].rc);
    //     return u;
    // }
}
```

0x22 可持久化 01 trie.cpp

```cpp
// 可持久化 01 trie
namespace trie {
    const int pond = 10001000; // 内存池大小，需要根据实际问题调整

    int nodeCount;
    struct node {
        int trans[2];
        int cnt;
    } t[pond];

    void insert(int &p, int q, int d, int x) {
        p = ++ nodeCount, t[p] = t[q], t[p].cnt ++;
        if (d < 0) return;
        int v = x >> d & 1;
```

```
        insert(t[p].trans[v], t[q].trans[v], d - 1, x);
    }
}
```

0x22 01 trie 维护集合异或和.cpp

```cpp
// 01 trie 维护集合异或和
namespace trie {
    const int pond = 10001000; // 内存池大小，需要根据实际问题调整

    int nodeCount;
    struct node {
        int trans[2];
        int cnt;
        int xr;
        #define lc trans[0]
        #define rc trans[1]
    } t[pond];

    int newnode() {
        int p = ++ nodeCount;
        t[p].lc = t[p].rc = t[p].cnt = t[p].xr = 0;
        return p;
    }

    void upd(int p) {
        t[p].cnt = t[t[p].lc].cnt + t[t[p].rc].cnt;
        t[p].xr = (t[t[p].lc].xr << 1) ^ (t[t[p].rc].xr << 1) ^
        ↪ (t[t[p].rc].cnt & 1);
    }

    void insert(int &p, int d, int x, int y) {
        if (!p) p = newnode();
        if (d == 31) {
            t[p].cnt += y;
            return;
        }
        int v = x >> d & 1;
        insert(t[p].trans[v], d + 1, x, y);
        upd(p);
    }

    void addone(int p) {
        std::swap(t[p].lc, t[p].rc);
        if (t[p].lc) addone(t[p].lc);
        upd(p);
    }
}
```

AC 自动机

0x22 AC 自动机.cpp

```cpp
// AC 自动机
namespace AC {
    const int SIZE = 1000100; // 多模式串长度总和，需要根据实际问题调整

    int nodeCount;
    struct node {
        int trans[26];
        int fail;
    } t[SIZE];

    int newnode() {
        int p = ++ nodeCount;
        for (int c = 0; c < 26; c ++) t[p].trans[c] = 0;
        t[p].fail = 0;
        return p;
    }
    void init() {
        nodeCount = 0, newnode(); // 根节点为 1
    }

    void insert(const std::string &s) {
        int p = 1;
        for (char ch : s) {
            int v = ch - 'a';
            if (!t[p].trans[v]) t[p].trans[v] = newnode();
            p = t[p].trans[v];
        }
    }

    void build_fail() {
        for (int i = 0; i < 26; i ++) t[0].trans[i] = 1;
        t[1].fail = 0;

        std::queue<int> q;
        q.push(1);

        while (q.size()) {
```

```
            int u = q.front(); q.pop();
            for (int i = 0; i < 26; i ++) {
                if (t[u].trans[i]) {
                    t[t[u].trans[i]].fail = t[t[u].fail].trans[i];
                    q.push(t[u].trans[i]);
                } else {
                    t[u].trans[i] = t[t[u].fail].trans[i];
                }
            }
        }
    }
}
```

0x22 AC 自动机（大字符集）.cpp

```
// 可持久化数组
namespace SGT {
    const int pond = 10001000; // 内存池大小，需要根据实际问题调整

    int nodeCount;
    struct node {
        int lc, rc;
        int val;
    } t[pond];

    void init() {
        nodeCount = 0, t[0].val = 1;
    }

    void insert(int &p, int q, int l, int r, int x, int y) {
        p = ++ nodeCount, t[p] = t[q];
        if (l == r) { t[p].val = y; return; }
        int mid = (l + r) >> 1;
        if (x <= mid) {
            insert(t[p].lc, t[q].lc, l, mid, x, y);
        } else {
            insert(t[p].rc, t[q].rc, mid + 1, r, x, y);
        }
    }

    int ask(int p, int l, int r, int x) {
        if (l == r) return t[p].val;
        int mid = (l + r) >> 1;
        if (x <= mid) {
            return ask(t[p].lc, l, mid, x);
        } else {
            return ask(t[p].rc, mid + 1, r, x);
        }
    }
}

// AC 自动机（大字符集）
namespace AC {
    const int SIZE = 1000100; // 多模式串长度总和，需要根据实际问题调整

    int nodeCount;
    struct node {
        std::map<int, int> trans;
        int fail;
    } t[SIZE];

    int root[SIZE];

    int newnode() {
        int p = ++ nodeCount;
        t[p].trans.clear();
        t[p].fail = 0;
        return p;
    }
    void init() {
        nodeCount = 0, newnode(); // 根节点为 1
    }

    void insert(const std::vector<int> &a) {
        int p = 1;
        for (int v : a) {
            if (!t[p].trans.contains(v)) t[p].trans[v] = newnode();
            p = t[p].trans[v];
        }
    }

    void build_fail() {
        SGT::init(), root[0] = 0;
        t[1].fail = 0;

        std::queue<int> q;
        q.push(1);

        while (q.size()) {
            int u = q.front(); q.pop();
            root[u] = root[t[u].fail];
```

```
            for (auto [c, v] : t[u].trans) {
                SGT::insert(root[u], root[u], 1, V, c, v);
                t[v].fail = SGT::ask(root[t[u].fail], 1, V, c);
                q.push(v);
            }
        }
    }
}
```

SA

0x22 SA.cpp

```
// 后缀数组
struct SuffixArray {
    int n, t;
    std::vector<int> sa, rk, height;
    std::vector<std::vector<int>> f;

    SuffixArray() {}
    SuffixArray(const auto &s) {
        build(s);
    }

    void build(const auto &s) {
        n = s.size() - 1; // s 下标从 1 开始
        int m = 256;
        sa.resize(n + 1), rk.resize(n + 1), height.resize(n + 1);
        std::vector<int> cnt(std::max(n, m) + 1), id(n + 1), px(n + 1),
         ↪  tmp(2 * n + 1);

        auto same = [&] (int x, int y, int k) {
            return tmp[x] == tmp[y] && tmp[x + k] == tmp[y + k];
        };

        for (int i = 1; i <= n; i ++) cnt[rk[i] = s[i]] ++;
        for (int i = 1; i <= m; i ++) cnt[i] += cnt[i - 1];
        for (int i = n; i >= 1; i --) sa[cnt[rk[i]] --] = i;

        for (int k = 1, p = 0; k < n; k <<= 1, m = p) {
            p = 0;
            for (int i = n - k + 1; i <= n; i ++) id[++ p] = i;
            for (int i = 1; i <= n; i ++)
                if (sa[i] > k) id[++ p] = sa[i] - k;

            for (int i = 1; i <= m; i ++) cnt[i] = 0;
            for (int i = 1; i <= n; i ++) cnt[px[i] = rk[id[i]]] ++;
            for (int i = 1; i <= m; i ++) cnt[i] += cnt[i - 1];
            for (int i = n; i >= 1; i --) sa[cnt[px[i]] --] = id[i];

            p = 0;
            for (int i = 1; i <= n; i ++) tmp[i] = rk[i];
            for (int i = 1; i <= n; i ++) rk[sa[i]] = same(sa[i - 1], sa[i],
             ↪  k) ? p : ++ p;
            if (p == n) break;
        }
        if (n == 1) rk[1] = 1;

        for (int i = 1, h = 0; i <= n; i ++) {
            int j = sa[rk[i] - 1];
            if (h) h --;
            while (i + h <= n && j + h <= n && s[i + h] == s[j + h]) h ++;
            height[rk[i]] = h;
        }

        t = std::__lg(n);
        f.assign(t + 1, std::vector<int>(n + 1));
        for (int i = 1; i <= n; i ++) {
            f[0][i] = height[i];
        }
        for (int j = 1; j <= t; j ++) {
            for (int i = 1; i <= n - (1 << j) + 1; i ++) {
                f[j][i] = std::min(f[j - 1][i], f[j - 1][i + (1 << (j - 1))]);
            }
        }
    }

    int ask(int l, int r) {
        int k = std::__lg(r - l + 1);
        return std::min(f[k][l], f[k][r - (1 << k) + 1]);
    }
    int lcp(int i, int j) {
        if (i == j) return n - i + 1;
        auto [l, r] = std::minmax(rk[i], rk[j]);
        return ask(l + 1, r);
    }
};
```

SAM

0x22 SAM.cpp

```cpp
// 后缀自动机
namespace SAM {
    const int SIZE = N * 2;

    int nodeCount, Last;
    struct node {
        int trans[26];
        int link, maxl;
    } t[SIZE];

    int newnode() {
        int p = ++ nodeCount;
        for (int c = 0; c < 26; c ++) t[p].trans[c] = 0;
        t[p].link = t[p].maxl = 0;
        return p;
    }
    void init() {
        nodeCount = 0, Last = 1, newnode(); // 根节点为 1
    }

    void extend(int c) {
        int p = Last, np = Last = newnode();

        t[np].maxl = t[p].maxl + 1;
        for (; p && t[p].trans[c] == 0; p = t[p].link) {
            t[p].trans[c] = np;
        }

        if (!p) {
            t[np].link = 1;
        } else {
            int q = t[p].trans[c];
            if (t[q].maxl == t[p].maxl + 1) {
                t[np].link = q;
            } else {
                int nq = newnode();
                t[nq] = t[q], t[nq].maxl = t[p].maxl + 1;
                t[np].link = t[q].link = nq;
                for (; p && t[p].trans[c] == q; p = t[p].link) {
                    t[p].trans[c] = nq;
                }
            }
        }
    }
}
```

```cpp
int p = 1, len = 0; // 当前匹配到的状态与长度
for (int i = 1; i <= m; i ++) {
    int c = T[i] - 'a';
    while (p > 1 && !SAM::t[p].trans[c]) {
        p = SAM::t[p].link, len = SAM::t[p].maxl;
    }
    if (SAM::t[p].trans[c]) {
        p = SAM::t[p].trans[c], len ++;
    }
    // 此时的 p 与 len，就是以 i 为结尾时匹配到的最长状态与长度
}
```

0x22 SAM 前端操作.cpp

```cpp
// 后缀自动机（前端操作）
namespace SAM {
    const int SIZE = N * 2;

    int strL, str[N];
    int nodeCount, Last;
    struct node {
        int trans[26];
        int link, maxl, pos;
    } t[SIZE];

    int net[SIZE][26];
    std::vector<int> son[SIZE];

    int newnode() {
        int p = ++ nodeCount;
        for (int c = 0; c < 26; c ++) t[p].trans[c] = net[p][c] = 0;
        t[p].link = t[p].maxl = t[p].pos = 0;
        son[p].clear();
        return p;
    }
    void init() {
        nodeCount = 0, Last = 1, newnode(); // 根节点为 1
        str[strL = 0] = -1;
    }

    void extend(int c) {
```

```
        int p = Last, np = Last = newnode();

        str[++ strL] = c, t[np].pos = strL;
        t[np].maxl = t[p].maxl + 1;
        for (; p && t[p].trans[c] == 0; p = t[p].link) {
            t[p].trans[c] = np;
        }

        if (!p) {
            t[np].link = 1;
        } else {
            int q = t[p].trans[c];
            if (t[q].maxl == t[p].maxl + 1) {
                t[np].link = q;
            } else {
                int nq = newnode();
                t[nq] = t[q], t[nq].maxl = t[p].maxl + 1;
                t[np].link = t[q].link = nq;
                for (; p && t[p].trans[c] == q; p = t[p].link) {
                    t[p].trans[c] = nq;
                }
            }
        }
    }

    void dfs_init(int u) {
        for (int v : son[u]) {
            net[u][str[t[v].pos - t[u].maxl]] = v;
            dfs_init(v);
        }
    }
    void build_tree() {
        for (int i = 2; i <= nodeCount; i ++) {
            son[t[i].link].push_back(i);
        }
        dfs_init(1);
    }

    void push_front(int &p, int &len, int c) {
        if (len == t[p].maxl) {
            p = net[p][c];
        } else if (str[t[p].pos - len] != c) {
            p = 0;
        }
        len = p ? len + 1 : 0;
    }
}
```

0x22 广义 SAM（离线）.cpp

```
// 广义 SAM（离线）
namespace GSAM {
    const int SIZE = N * 2;

    int trieCount, trie[N][26], site[N];
    int nodeCount;
    struct node {
        int trans[26];
        int link, maxl;
    } t[SIZE];

    int newtrie() {
        int p = ++ trieCount;
        for (int c = 0; c < 26; c ++) trie[p][c] = 0;
        site[p] = 0;
        return p;
    }
    int newnode() {
        int p = ++ nodeCount;
        for (int c = 0; c < 26; c ++) t[p].trans[c] = 0;
        t[p].link = t[p].maxl = 0;
        return p;
    }
    void init() {
        trieCount = 0, newtrie(); // trie 根节点为 1
        nodeCount = 0, newnode(); // GSAM 根节点为 1
    }

    void insert(const std::string &s) {
        int p = 1;
        for (char ch : s) {
            int v = ch - 'a';
            if (!trie[p][v]) trie[p][v] = newtrie();
            p = trie[p][v];
        }
    }

    int extend(int Last, int c) {
        int p = Last, np = newnode();

        t[np].maxl = t[p].maxl + 1;
```

```
        for (; p && t[p].trans[c] == 0; p = t[p].link) {
            t[p].trans[c] = np;
        }

        if (!p) {
            t[np].link = 1;
        } else {
            int q = t[p].trans[c];
            if (t[q].maxl == t[p].maxl + 1) {
                t[np].link = q;
            } else {
                int nq = newnode();
                t[nq] = t[q], t[nq].maxl = t[p].maxl + 1;
                t[np].link = t[q].link = nq;
                for (; p && t[p].trans[c] == q; p = t[p].link) {
                    t[p].trans[c] = nq;
                }
            }
        }
        return np;
    }

    void build() {
        std::queue<int> q;
        q.push(1), site[1] = 1;

        while (q.size()) {
            int u = q.front(); q.pop();
            for (int i = 0; i < 26; i ++) {
                if (trie[u][i]) {
                    site[trie[u][i]] = extend(site[u], i);
                    q.push(trie[u][i]);
                }
            }
        }
    }
}
```

0x22 广义 SAM（在线）.cpp

```
// 广义 SAM（在线）
// 在插入一个串之前，需要令 GSAM::Last = 1
namespace GSAM {
    const int SIZE = N * 2;

    int nodeCount, Last;
    struct node {
        int trans[26];
        int link, maxl;
    } t[SIZE];

    int newnode() {
        int p = ++ nodeCount;
        for (int c = 0; c < 26; c ++) t[p].trans[c] = 0;
        t[p].link = t[p].maxl = 0;
        return p;
    }
    void init() {
        nodeCount = 0, Last = 1, newnode(); // 根节点为 1
    }

    void extend(int c) {
        int p = Last;
        if (t[p].trans[c]) {
            int q = t[p].trans[c];
            if (t[q].maxl == t[p].maxl + 1) {
                Last = q;
            } else {
                int nq = newnode();
                t[nq] = t[q], t[nq].maxl = t[p].maxl + 1;
                t[q].link = nq;
                for (; p && t[p].trans[c] == q; p = t[p].link) {
                    t[p].trans[c] = nq;
                }
                Last = nq;
            }
        } else {
            int np = Last = newnode();

            t[np].maxl = t[p].maxl + 1;
            for (; p && t[p].trans[c] == 0; p = t[p].link) {
                t[p].trans[c] = np;
            }

            if (!p) {
                t[np].link = 1;
            } else {
                int q = t[p].trans[c];
                if (t[q].maxl == t[p].maxl + 1) {
                    t[np].link = q;
                } else {
```

```
                    int nq = newnode();
                    t[nq] = t[q], t[nq].maxl = t[p].maxl + 1;
                    t[np].link = t[q].link = nq;
                    for (; p && t[p].trans[c] == q; p = t[p].link) {
                        t[p].trans[c] = nq;
                    }
                }
            }
        }
    }
}
```

PAM

0x22 PAM.cpp

```
// 回文自动机
namespace PAM {
    int strL, str[N];
    int nodeCount, Last;
    struct node {
        int trans[26];
        int link, len;
    } t[N];

    int newnode() {
        int p = ++ nodeCount;
        for (int c = 0; c < 26; c ++) t[p].trans[c] = 0;
        t[p].link = t[p].len = 0;
        return p;
    }
    void init() {
        nodeCount = -1, Last = 1, newnode(), newnode(); // 偶根为 0，奇根为 1
        str[strL = 0] = -1;
        t[0].len = 0, t[0].link = 1;
        t[1].len = -1;
    }

    int find(int p) {
        while (str[strL - t[p].len - 1] != str[strL]) {
            p = t[p].link;
        }
        return p;
    }
    void extend(int c) {
        str[++ strL] = c;
        int p = find(Last);
        if (!t[p].trans[c]) {
            int np = newnode();
            t[np].len = t[p].len + 2;
            t[np].link = t[find(t[p].link)].trans[c];
            t[p].trans[c] = np;
        }
        Last = t[p].trans[c];
    }
}
```

0x22 PAM half.cpp

```
// 回文自动机 half
namespace PAM {
    int strL, str[N];
    int nodeCount, Last;
    struct node {
        int trans[26];
        int link, half, len;
    } t[N];

    int newnode() {
        int p = ++ nodeCount;
        for (int c = 0; c < 26; c ++) t[p].trans[c] = 0;
        t[p].link = t[p].half = t[p].len = 0;
        return p;
    }
    void init() {
        nodeCount = -1, Last = 1, newnode(), newnode(); // 偶根为 0，奇根为 1
        str[strL = 0] = -1;
        t[0].len = 0, t[0].link = 1;
        t[1].len = -1;
    }

    int find(int p) {
        while (str[strL - t[p].len - 1] != str[strL]) {
            p = t[p].link;
        }
        return p;
    }
    int find_half(int p, int L) {
        while (t[p].len + 2 > L || str[strL - t[p].len - 1] != str[strL]) {
            p = t[p].link;
```

```
        }
        return p;
    }
    void extend(int c) {
        str[++ strL] = c;
        int p = find(Last);
        if (!t[p].trans[c]) {
            int np = newnode();
            t[np].len = t[p].len + 2;
            t[np].link = t[find(t[p].link)].trans[c];
            t[np].half = t[np].len > 2 ? t[find_half(t[p].half, t[np].len / 2)].trans[c] : t[np].link;
            t[p].trans[c] = np;
        }
        Last = t[p].trans[c];
    }
}
```

Palindrome Series

0x22 PAM series-link.cpp

```
// 回文自动机 series-link
namespace PAM {
    int strL, str[N];
    int nodeCount, Last;
    struct node {
        int trans[26];
        int link, len;
        int diff, slink;
    } t[N];
    int g[N];

    int newnode() {
        int p = ++ nodeCount;
        for (int c = 0; c < 26; c ++) t[p].trans[c] = 0;
        t[p].link = t[p].len = 0;
        t[p].diff = t[p].slink = 0;
        g[p] = 0x3f3f3f3f;
        return p;
    }
    void init() {
        nodeCount = -1, Last = 1, newnode(), newnode(); // 偶根为 0，奇根为 1
        str[strL = 0] = -1;
        t[0].len = 0, t[0].link = 1;
        t[1].len = -1;
    }

    int find(int p) {
        while (str[strL - t[p].len - 1] != str[strL]) {
            p = t[p].link;
        }
        return p;
    }
    void extend(int c) {
        str[++ strL] = c;
        int p = find(Last);
        if (!t[p].trans[c]) {
            int np = newnode();
            t[np].len = t[p].len + 2;
            t[np].link = t[find(t[p].link)].trans[c];
            t[np].diff = t[np].len - t[t[np].link].len;
            t[np].slink = t[np].diff == t[t[np].link].diff ? t[t[np].link].slink : t[np].link;
            t[p].trans[c] = np;
        }
        Last = t[p].trans[c];
    }

    int ask(int x, int i) {
        int ans = 0x3f3f3f3f;
        for (int u = x; u > 1; u = t[u].slink) {
            g[u] = f[i - t[u].diff - t[t[u].slink].len];
            if (t[u].diff == t[t[u].link].diff) {
                chmin(g[u], g[t[u].link]);
            }
            chmin(ans, g[u]);
        }
        return ans;
    }
}
```

0x31 数学 多项式

拉格朗日插值

0x31 拉格朗日插值.cpp

```cpp
// 拉格朗日插值
int lagrange(std::vector<std::pair<int, int>> seq, int k) {
    int n = seq.size(); // seq 下标从 0 开始，至多 n-1 次多项式需要 n 个点值
    int ans = 0;
    for (int i = 0; i < n; i ++) {
        int p = 1, q = 1;
        for (int j = 0; j < n; j ++) {
            if (i == j) continue;
            mul(p, k - seq[j].first);
            mul(q, seq[i].first - seq[j].first);
        }
        ans = (ans + 1ll * p * qpow(q, mod - 2, mod) % mod * seq[i].second)
         ↪ % mod;
    }
    return norm(ans);
}
```

0x31 拉格朗日插值求系数.cpp

```cpp
// 拉格朗日插值求系数
std::vector<int> lagrange(std::vector<std::pair<int, int>> seq) {
    int n = seq.size(); // seq 下标从 0 开始，至多 n-1 次多项式需要 n 个点值

    std::vector<int> a(n); // 系数部分 a[]
    for (int i = 0; i < n; i ++) {
        int q = 1;
        for (int j = 0; j < n; j ++) {
            if (i == j) continue;
            mul(q, seq[i].first - seq[j].first);
        }
        a[i] = 1ll * seq[i].second * qpow(q, mod - 2, mod) % mod;
    }

    std::vector<int> g(n + 1); // 总多项式 g
    g[0] = 1;
    for (int i = 0; i < n; i ++) {
        for (int j = i + 1; j >= 1; j --) {
            g[j] = (1ll * g[j] * -seq[i].first + g[j - 1]) % mod;
        }
        g[0] = 1ll * g[0] * -seq[i].first % mod;
    }

    std::vector<int> f(n); // 答案多项式 f
    for (int i = 0; i < n; i ++) {
        std::vector<int> h(n); // 分多项式 h_i
        h[n - 1] = g[n];
        for (int j = n - 2; j >= 0; j --) {
            h[j] = (g[j + 1] + 1ll * h[j + 1] * seq[i].first) % mod;
        }

        for (int j = 0; j < n; j ++) {
            f[j] = (f[j] + 1ll * a[i] * h[j]) % mod;
        }
    }

    for (int &x : f) x = norm(x);
    return f;
}

int func(const std::vector<int> &f, int k) {
    int n = f.size();
    int y = 0;
    for (int i = n - 1; i >= 0; i --) {
        y = (1ll * y * k + f[i]) % mod;
    }
    return norm(y);
}
```

0x31 连续点值的拉格朗日插值.cpp

```cpp
// 连续点值的拉格朗日插值
int lagrange(std::vector<int> y, int k) {
    int n = y.size() - 1; // y 下标从 1 开始，连续点值为 f(1), ..., f(n)
    if (k >= 1 && k <= n) {
        return y[k];
    }

    std::vector<int> pre(n + 2), suf(n + 2);
    pre[0] = 1;
    for (int i = 1; i <= n; i ++) {
        pre[i] = 1ll * pre[i - 1] * (k - i) % mod;
    }
    suf[n + 1] = 1;
```

```
    for (int i = n; i >= 1; i --) {
        suf[i] = 1ll * suf[i + 1] * (k - i) % mod;
    }

    std::vector<int> fact(n + 1), facv(n + 1);
    fact[0] = 1;
    for (int i = 1; i <= n; i ++) {
        fact[i] = 1ll * fact[i - 1] * i % mod;
    }
    facv[n] = qpow(fact[n], mod - 2, mod);
    for (int i = n - 1; i >= 0; i --) {
        facv[i] = 1ll * facv[i + 1] * (i + 1) % mod;
    }

    int ans = 0;
    for (int i = 1; i <= n; i ++) {
        int cur = y[i];
        cur = 1ll * cur * pre[i - 1] % mod * suf[i + 1] % mod;
        cur = 1ll * cur * facv[i - 1] % mod * facv[n - i] % mod;
        if ((n - i) & 1) cur = -cur;
        add(ans, norm(cur));
    }

    return ans;
}
```

FFT

0x31 FFT.cpp

```
const double pi = acos(-1);
using comp = std::complex<double>;

std::vector<comp> roots{0, 1};
void dft(std::vector<comp> &a) {
    int n = a.size();
    for (int i = 1, j = 0; i < n; i ++) {
        for (int k = n >> 1; (j ^= k) < k; k >>= 1);
        if (i < j) std::swap(a[i], a[j]);
    }
    for (int k = roots.size(); k < n; k <<= 1) {
        comp w(cos(pi / k), sin(pi / k));
        roots.resize(k << 1);
        for (int i = k >> 1; i < k; i ++) {
            roots[i * 2] = roots[i];
            roots[i * 2 + 1] = roots[i] * w;
        }
    }
    for (int k = 1; k < n; k <<= 1) {
        for (int i = 0; i < n; i += (k << 1)) {
            for (int j = 0; j < k; j ++) {
                comp u = a[i + j], v = a[i + j + k] * roots[k + j];
                a[i + j] = u + v, a[i + j + k] = u - v;
            }
        }
    }
}
void idft(std::vector<comp> &a) {
    int n = a.size();
    std::reverse(a.begin() + 1, a.end());
    dft(a);
    for (int i = 0; i < n; i ++) {
        a[i] /= n;
    }
}

struct poly : public std::vector<comp> {
    using std::vector<comp>::vector;
    explicit poly(const std::vector<comp> &a) : std::vector<comp>(a) {}

    friend poly operator * (poly a, poly b) {
        int tot = a.size() + b.size() - 1;
        if (tot < 128) {
            poly c(tot);
            for (int i = 0; i < a.size(); i ++) {
                for (int j = 0; j < b.size(); j ++) {
                    c[i + j] += a[i] * b[j];
                }
            }
            return c;
        }
        int L = 1;
        while (L < tot) L <<= 1;
        a.resize(L), b.resize(L);
        dft(a), dft(b);
        for (int i = 0; i < L; i ++) {
            a[i] *= b[i];
        }
        idft(a);
        a.resize(tot);
        return a;
```

```
    }
};
```

NTT

0x31 NTT.cpp

```cpp
std::vector<int> roots{0, 1};
void dft(std::vector<int> &a) {
    int n = a.size();
    for (int i = 1, j = 0; i < n; i ++) {
        for (int k = n >> 1; (j ^= k) < k; k >>= 1);
        if (i < j) std::swap(a[i], a[j]);
    }
    for (int k = roots.size(); k < n; k <<= 1) {
        int w = qpow(3, (mod - 1) / (k << 1), mod); // 原根根据实际模数调整
        roots.resize(k << 1);
        for (int i = k >> 1; i < k; i ++) {
            roots[i * 2] = roots[i];
            roots[i * 2 + 1] = 1ll * roots[i] * w % mod;
        }
    }
    for (int k = 1; k < n; k <<= 1) {
        for (int i = 0; i < n; i += (k << 1)) {
            for (int j = 0; j < k; j ++) {
               int u = a[i + j], v = 1ll * a[i + j + k] * roots[k + j] % mod;
                add(a[i + j] = u, v), dec(a[i + j + k] = u, v);
            }
        }
    }
}
void idft(std::vector<int> &a) {
    int n = a.size(), inv = (1 - mod) / n + mod;
    std::reverse(a.begin() + 1, a.end());
    dft(a);
    for (int i = 0; i < n; i ++) {
        mul(a[i], inv);
    }
}

struct poly : public std::vector<int> {
    using std::vector<int>::vector;
    explicit poly(const std::vector<int> &a) : std::vector<int>(a) {}

    friend poly operator * (poly a, poly b) {
        int tot = a.size() + b.size() - 1;
        if (tot < 128) {
            poly c(tot);
            for (int i = 0; i < a.size(); i ++) {
                for (int j = 0; j < b.size(); j ++) {
                    c[i + j] = (c[i + j] + 1ll * a[i] * b[j]) % mod;
                }
            }
            return c;
        }
        int L = 1;
        while (L < tot) L <<= 1;
        a.resize(L), b.resize(L);
        dft(a), dft(b);
        for (int i = 0; i < L; i ++) {
            mul(a[i], b[i]);
        }
        idft(a);
        a.resize(tot);
        return a;
    }
};
```

MTT

三模 NTT

0x31 MTT（三模 NTT）.cpp

```cpp
template <const int mod>
struct ntt {
    inline void add(int &x, const int &y) { x += y; if (x >= mod) x -= mod; }
    inline void dec(int &x, const int &y) { x -= y; if (x < 0) x += mod; }
    inline void mul(int &x, const int &y) { x = 1ll * x * y % mod; }

    std::vector<int> roots{0, 1};
    void dft(std::vector<int> &a) {
        int n = a.size();
        for (int i = 1, j = 0; i < n; i ++) {
            for (int k = n >> 1; (j ^= k) < k; k >>= 1);
            if (i < j) std::swap(a[i], a[j]);
        }
        for (int k = roots.size(); k < n; k <<= 1) {
            int w = qpow(3, (mod - 1) / (k << 1), mod);
            roots.resize(k << 1);
```

```
            for (int i = k >> 1; i < k; i ++) {
                roots[i * 2] = roots[i];
                roots[i * 2 + 1] = 1ll * roots[i] * w % mod;
            }
        }
        for (int k = 1; k < n; k <<= 1) {
            for (int i = 0; i < n; i += (k << 1)) {
                for (int j = 0; j < k; j ++) {
                    int u = a[i + j], v = 1ll * a[i + j + k] * roots[k + j] %
                        ↪ mod;
                    add(a[i + j] = u, v), dec(a[i + j + k] = u, v);
                }
            }
        }
    }
    void idft(std::vector<int> &a) {
        int n = a.size(), inv = (1 - mod) / n + mod;
        std::reverse(a.begin() + 1, a.end());
        dft(a);
        for (int i = 0; i < n; i ++) {
            mul(a[i], inv);
        }
    }

    auto mul(std::vector<int> a, std::vector<int> b) {
        for (int &x : a) x %= mod;
        for (int &x : b) x %= mod;
        dft(a), dft(b);
        for (int i = 0; i < a.size(); i ++) {
            mul(a[i], b[i]);
        }
        idft(a);
        return a;
    }
};

const int mod1 = 998244353, mod2 = 1004535809, mod3 = 469762049;
ntt<mod1> ntt1;
ntt<mod2> ntt2;
ntt<mod3> ntt3;

int CRT(int x1, int x2, int x3) {
    static const i64 M12 = 1ll * mod1 * mod2;
    static const int
        inv1 = qpow(mod1, mod2 - 2, mod2),
        inv12 = qpow(M12 % mod3, mod3 - 2, mod3);
    int t1 = 1ll * (x2 - x1 + mod2) * inv1 % mod2;
    i64 x12 = x1 + 1ll * mod1 * t1;
    int t2 = (x3 - x12 % mod3 + mod3) * inv12 % mod3;
    return (x12 % mod + (M12 % mod) * t2) % mod;
}

// int CRT(int x1, int x2, int x3) {
//     static const i128 M = i128(mod1) * mod2 * mod3;
//     static const i64 M1 = M / mod1, M2 = M / mod2, M3 = M / mod3;
//     static const int
//         V1 = qpow(M1 % mod1, mod1 - 2, mod1),
//         V2 = qpow(M2 % mod2, mod2 - 2, mod2),
//         V3 = qpow(M3 % mod3, mod3 - 2, mod3);
//     i128 x = 0;
//     x += i128(x1) * M1 * V1;
//     x += i128(x2) * M2 * V2;
//     x += i128(x3) * M3 * V3;
//     x %= M; // 此时的 x 即为 CRT 的解
//     return x % mod;
// }

struct poly : public std::vector<int> {
    using std::vector<int>::vector;
    explicit poly(const std::vector<int> &a) : std::vector<int>(a) {}

    friend poly operator * (poly a, poly b) {
        int tot = a.size() + b.size() - 1;
        if (tot < 128) {
            poly c(tot);
            for (int i = 0; i < a.size(); i ++) {
                for (int j = 0; j < b.size(); j ++) {
                    c[i + j] = (c[i + j] + 1ll * a[i] * b[j]) % mod;
                }
            }
            return c;
        }
        int L = 1;
        while (L < tot) L <<= 1;
        a.resize(L), b.resize(L);
        auto c1 = ntt1.mul(a, b), c2 = ntt2.mul(a, b), c3 = ntt3.mul(a, b);
        poly c(tot);
        for (int i = 0; i < tot; i ++) {
            c[i] = CRT(c1[i], c2[i], c3[i]);
        }
        return c;
    }
```

```
};
```

拆系数 FFT

0x31 MTT（拆系数 FFT）.cpp

```cpp
const double pi = acos(-1);
using comp = std::complex<double>;

std::vector<comp> roots{0, 1};
void dft(std::vector<comp> &a) {
    int n = a.size();
    for (int i = 1, j = 0; i < n; i ++) {
        for (int k = n >> 1; (j ^= k) < k; k >>= 1);
        if (i < j) std::swap(a[i], a[j]);
    }
    for (int k = roots.size(); k < n; k <<= 1) {
        comp w(cos(pi / k), sin(pi / k));
        roots.resize(k << 1);
        for (int i = k >> 1; i < k; i ++) {
            roots[i * 2] = roots[i];
            roots[i * 2 + 1] = roots[i] * w;
        }
    }
    for (int k = 1; k < n; k <<= 1) {
        for (int i = 0; i < n; i += (k << 1)) {
            for (int j = 0; j < k; j ++) {
                comp u = a[i + j], v = a[i + j + k] * roots[k + j];
                a[i + j] = u + v, a[i + j + k] = u - v;
            }
        }
    }
}
void idft(std::vector<comp> &a) {
    int n = a.size();
    std::reverse(a.begin() + 1, a.end());
    dft(a);
    for (int i = 0; i < n; i ++) {
        a[i] /= n;
    }
}

struct poly : public std::vector<int> {
    using std::vector<int>::vector;
    explicit poly(const std::vector<int> &a) : std::vector<int>(a) {}

    friend poly operator * (poly a, poly b) {
        int tot = a.size() + b.size() - 1;
        if (tot < 128) {
            poly c(tot);
            for (int i = 0; i < a.size(); i ++) {
                for (int j = 0; j < b.size(); j ++) {
                    c[i + j] = (c[i + j] + 1ll * a[i] * b[j]) % mod;
                }
            }
            return c;
        }
        int L = 1;
        while (L < tot) L <<= 1;
        std::vector<comp> P(L), Q(L), R(L), S(L);
        for (int i = 0; i < a.size(); i ++) {
            P[i] = comp(a[i] & 32767, a[i] >> 15);
        }
        for (int i = 0; i < b.size(); i ++) {
            Q[i] = comp(b[i] & 32767, b[i] >> 15);
        }
        dft(P), dft(Q);
        for (int i = 0; i < L; i ++) {
            int j = (L - i) & (L - 1);
            comp a0 = (P[i] + std::conj(P[j])) * 0.5;
            comp a1 = (P[i] - std::conj(P[j])) * comp(0, -0.5);
            R[i] = a0 * Q[i], S[i] = a1 * Q[i];
        }
        idft(R), idft(S);
        poly c(tot);
        for (int i = 0; i < tot; i ++) {
            i64 c0 = llround(R[i].real()) % mod;
            i64 c1 = (llround(R[i].imag()) + llround(S[i].real())) % mod;
            i64 c2 = llround(S[i].imag()) % mod;
            c[i] = ((c2 << 30) + (c1 << 15) + c0) % mod;
        }
        return c;
    }
};
```

多项式多点求值

0x31 多项式多点求值（多项式取模）.cpp

```cpp
// 多项式多点求值（多项式取模做法），作为 poly 的成员函数
std::vector<int> eval(std::vector<int> x) const {
    if (empty()) return std::vector<int>(x.size());
    if (x.empty()) return {};

    int n = x.size();
    std::vector<poly> t(n * 4);
    std::vector<int> ans(n);

    std::function<void(int, int, int)> build = [&] (int p, int l, int r) {
        if (l == r) {
            t[p] = poly{norm(-x[l]), 1};
            return;
        }
        int mid = (l + r) >> 1;
        build(p * 2, l, mid), build(p * 2 + 1, mid + 1, r);
        t[p] = t[p * 2] * t[p * 2 + 1];
    };
    build(1, 0, n - 1);

    std::function<void(int, int, int, const poly &)> solve = [&] (int p, int
     ↪ l, int r, const poly &f) {
        if (l == r) {
            ans[l] = f[0];
            return;
        }
        int mid = (l + r) >> 1;
        solve(p * 2, l, mid, divmod(f, t[p * 2]).second);
        solve(p * 2 + 1, mid + 1, r, divmod(f, t[p * 2 + 1]).second);
    };
    solve(1, 0, n - 1, divmod(*this, t[1]).second);

    return ans;
}
```

多项式全家桶

0x31 多项式全家桶.cpp

```cpp
std::vector<int> roots{0, 1};
void dft(std::vector<int> &a) {
    int n = a.size();
    for (int i = 1, j = 0; i < n; i ++) {
        for (int k = n >> 1; (j ^= k) < k; k >>= 1);
        if (i < j) std::swap(a[i], a[j]);
    }
    for (int k = roots.size(); k < n; k <<= 1) {
        int w = qpow(3, (mod - 1) / (k << 1), mod); // 原根根据实际模数调整
        roots.resize(k << 1);
        for (int i = k >> 1; i < k; i ++) {
            roots[i * 2] = roots[i];
            roots[i * 2 + 1] = 1ll * roots[i] * w % mod;
        }
    }
    for (int k = 1; k < n; k <<= 1) {
        for (int i = 0; i < n; i += (k << 1)) {
            for (int j = 0; j < k; j ++) {
               int u = a[i + j], v = 1ll * a[i + j + k] * roots[k + j] % mod;
                add(a[i + j] = u, v), dec(a[i + j + k] = u, v);
            }
        }
    }
}
void idft(std::vector<int> &a) {
    int n = a.size(), inv = (1 - mod) / n + mod;
    std::reverse(a.begin() + 1, a.end());
    dft(a);
    for (int i = 0; i < n; i ++) {
        mul(a[i], inv);
    }
}

// 多项式全家桶（2026-11-05）
struct poly : public std::vector<int> {
    using std::vector<int>::vector;
    explicit poly(const std::vector<int> &a) : std::vector<int>(a) {}

    // 多项式乘法
    friend poly operator * (poly a, poly b) {
        int tot = a.size() + b.size() - 1;
        if (tot < 128) {
```

```
            poly c(tot);
            for (int i = 0; i < a.size(); i ++) {
                for (int j = 0; j < b.size(); j ++) {
                    c[i + j] = (c[i + j] + 1ll * a[i] * b[j]) % mod;
                }
            }
            return c;
        }
        int L = 1;
        while (L < tot) L <<= 1;
        a.resize(L), b.resize(L);
        dft(a), dft(b);
        for (int i = 0; i < L; i ++) {
            mul(a[i], b[i]);
        }
        idft(a);
        a.resize(tot);
        return a;
    }

    // 多项式乘法（暴力）
    friend poly mul_bf(poly a, poly b) {
        poly c(a.size() + b.size() - 1);
        for (int i = 0; i < a.size(); i ++) {
            for (int j = 0; j < b.size(); j ++) {
                c[i + j] = (c[i + j] + 1ll * a[i] * b[j]) % mod;
            }
        }
        return c;
    }
    // 多项式除法（暴力）
    friend poly div_bf(poly a, poly b) {
        int t = 0;
        while (t < b.size() && b[t] == 0) t ++;
        if (t == b.size()) return poly(a.size() - b.size() + 1); // 除以全 0
         ↪ 多项式无意义
        a = poly(a.begin() + t, a.end());
        b = poly(b.begin() + t, b.end());
        int m = b.size() - 1, inv = qpow(b[0], mod - 2, mod);
        poly c(a.size() - b.size() + 1);
        for (int i = 0; i < c.size(); i ++) {
            c[i] = a[i];
            for (int j = std::max(0, i - m); j < i; j ++) {
                dec(c[i], 1ll * c[j] * b[i - j] % mod);
            }
            mul(c[i], inv);
        }
        return c;
    }

    friend poly operator + (poly a, poly b) {
        poly c(std::max(a.size(), b.size()));
        for (int i = 0; i < a.size(); i ++) add(c[i], a[i]);
        for (int i = 0; i < b.size(); i ++) add(c[i], b[i]);
        return c;
    }
    friend poly operator - (poly a, poly b) {
        poly c(std::max(a.size(), b.size()));
        for (int i = 0; i < a.size(); i ++) add(c[i], a[i]);
        for (int i = 0; i < b.size(); i ++) dec(c[i], b[i]);
        return c;
    }
    friend poly operator - (poly a) {
        for (int &x : a) neg(x);
        return a;
    }

    friend poly operator * (poly a, int b) {
        for (int &x : a) mul(x, b);
        return a;
    }
    friend poly operator / (poly a, int b) {
        int inv = qpow(b, mod - 2, mod);
        for (int &x : a) mul(x, inv);
        return a;
    }

    poly &operator += (poly b) {
        return *this = *this + b;
    }
    poly &operator -= (poly b) {
        return *this = *this - b;
    }
    poly &operator *= (poly b) {
        return *this = *this * b;
    }
    poly &operator *= (int b) {
        return *this = *this * b;
    }
    poly &operator /= (int b) {
        return *this = *this / b;
    }
```

```
    poly mulxk(int k) const {
        auto res = *this;
        res.insert(res.begin(), k, 0);
        return res;
    }
    poly divxk(int k) const {
        if (k >= size()) return poly();
        return poly(begin() + k, end());
    }
    poly modxk(int k) const {
        if (k >= size()) return *this;
        return poly(begin(), begin() + k);
    }
    poly cirxk(int k) const {
        auto res = *this;
        for (int i = k; i < size(); i ++) {
            add(res[i % k], res[i]); // 若多项式次数 <2k，可改成 res[i - k]
        }
        return res.modxk(k);
    }

    poly flip() const {
        auto res = *this;
        std::reverse(res.begin(), res.end());
        return res;
    }

    poly mulT(poly b) const {
        if (b.empty()) return poly();
        std::reverse(b.begin(), b.end());
        return ((*this) * b).divxk(b.size() - 1);
        // 等价于“翻转卷积”，a.mulT(b) 相当于是求 a[i] * b[j] -> c[i - j]
    }

    // 多项式导数
    poly deriv() const {
        if (empty()) return poly();
        poly res(size() - 1);
        for (int i = 1; i < size(); i ++) {
            res[i - 1] = 1ll * (*this)[i] * i % mod;
        }
        return res;
    }

    // 多项式积分
    poly integ() const {
        poly res(size() + 1);
        for (int i = 0; i < size(); i ++) {
            res[i + 1] = 1ll * (*this)[i] * qpow(i + 1, mod - 2, mod) % mod;
        }
        return res;
    }

    // 多项式求逆
    poly inv(int m) const { // 需要保证常数项不为 0
        poly x{qpow(front(), mod - 2, mod)};
        int k = 1;
        while (k < m) {
            k <<= 1;
            x = (x * (poly{2} - modxk(k) * x)).modxk(k);
        }
        return x.modxk(m);
    }

    // 多项式开方
    poly sqrt(int m) const {
        poly x{1}; // 默认常数项为 1
        int k = 1;
        while (k < m) {
            k <<= 1;
            x = (x + (modxk(k) * x.inv(k)).modxk(k)) / 2;
        }
        return x.modxk(m);
    }

    // 多项式带余除法
    friend std::pair<poly, poly> divmod(poly a, poly b) {
        int n = a.size(), m = b.size(), L = n - m + 1;
        if (n < m) return {poly{0}, a};
        poly q = (a.flip().modxk(L) * b.flip().inv(L)).modxk(L).flip();
        poly r = a - b * q;
        while (r.size() > 1 && !r.back()) {
            r.pop_back();
        }
        return {q, r};
    }

    // 多项式 ln
    poly ln(int m) const { // 需要保证常数项为 1
        return (deriv() * inv(m)).integ().modxk(m);
    }
```

```
    // 多项式 exp
    poly exp(int m) const { // 需要保证常数项为 0
        poly x{1};
        int k = 1;
        while (k < m) {
            k <<= 1;
            x = (x * (poly{1} - x.ln(k) + modxk(k))).modxk(k);
        }
        return x.modxk(m);
    }

    // 多项式快速幂
    poly pow(int k, int m) const {
        int i = 0;
        while (i < size() && (*this)[i] == 0) i ++;
        if (i == size() || 1ll * i * k >= m) return poly(m);
        int v = (*this)[i];
        poly f = divxk(i) / v;
        return (f.ln(m - i * k) * k).exp(m - i * k).mulxk(i * k) * qpow(v,
         ↪ k, mod);
        /*
            当实际 k < mod 时，可以放心使用
            当实际 k >= mod 时，请注意：
            - 当 i > 0 时，直接返回 poly(m)
            - 当 i = 0 时
              - 在多项式 ln 与多项式 exp 部分，k 需要对 mod 取模
              - 在 qpow(v, k, mod) 部分，k 需要对 mod - 1 取模
        */
    }
    // 少项式快速幂
    poly pow_bf(int k, int m) const { // 需要保证常数项不为 0
        std::vector<int> v(m); // 需要保证 m > 1
        v[1] = 1;
        for (int i = 2; i < m; i ++) {
            v[i] = 1ll * v[mod % i] * (mod - mod / i) % mod;
        }

        const poly a = *this;
        int n = a.size() - 1, inv = qpow(a[0], mod - 2, mod);
        poly b(m);
        b[0] = qpow(a[0], k, mod);
        for (int i = 0; i + 1 < m; i ++) {
            for (int j = std::max(0, i - n + 1); j <= i; j ++) {
                add(b[i + 1], 1ll * (i - j + 1) * a[i - j + 1] % mod * b[j] %
                 ↪ mod);
            }
            mul(b[i + 1], k);
            for (int j = std::max(0, i - n); j < i; j ++) {
              dec(b[i + 1], 1ll * (j + 1) * b[j + 1] % mod * a[i - j] % mod);
            }
            mul(b[i + 1], v[i + 1]), mul(b[i + 1], inv);
        }
        return b;
    }

    // 多项式多点求值
    std::vector<int> eval(std::vector<int> x) const {
        if (empty()) return std::vector<int>(x.size());

        int n = std::max(x.size(), size());
        std::vector<poly> t(n * 4);
        std::vector<int> ans(x.size());
        x.resize(n);

        std::function<void(int, int, int)> build = [&] (int p, int l, int r)
         ↪ {
            if (l == r) {
                t[p] = poly{1, norm(-x[l])};
                return;
            }
            int mid = (l + r) >> 1;
            build(p * 2, l, mid), build(p * 2 + 1, mid + 1, r);
            t[p] = t[p * 2] * t[p * 2 + 1];
        };
        build(1, 0, n - 1);

        std::function<void(int, int, int, const poly &)> solve = [&] (int p,
         ↪ int l, int r, const poly &num) {
            if (l >= ans.size()) return;
            if (l == r) {
                ans[l] = num[0];
                return;
            }
            int mid = (l + r) >> 1;
            solve(p * 2, l, mid, num.mulT(t[p * 2 + 1]).modxk(mid - l + 1));
            solve(p * 2 + 1, mid + 1, r, num.mulT(t[p * 2]).modxk(r - mid));
        };
        solve(1, 0, n - 1, mulT(t[1].inv(n)));

        return ans;
    }
```

```
    // 多项式平移
    poly shift(int c) const {
        int n = size() - 1;
        std::vector<int> fact(n + 1), facv(n + 1);
        fact[0] = 1;
        for (int i = 1; i <= n; i ++) {
            fact[i] = 1ll * fact[i - 1] * i % mod;
        }
        facv[n] = qpow(fact[n], mod - 2, mod);
        for (int i = n - 1; i >= 0; i --) {
            facv[i] = 1ll * facv[i + 1] * (i + 1) % mod;
        }

        c = norm(c);
        poly a(n + 1), b(n + 1);
        for (int i = 0; i <= n; i ++) {
            a[i] = 1ll * (*this)[i] * fact[i] % mod;
        }
        for (int i = 0, w = 1; i <= n; i ++, mul(w, c)) {
            b[i] = 1ll * w * facv[i] % mod;
        }

        poly res = a.mulT(b);
        for (int i = 0; i <= n; i ++) {
            mul(res[i], facv[i]);
        }
        return res;
    }

    // Chirp Z-Transform
    std::vector<int> czt(int c, int m) const {
        c = norm(c);
        if (c == 0) {
            std::vector<int> res(m, front());
            res[0] = 0;
            for (int x : *this) add(res[0], x);
            return res;
        }

        poly a = *this;
        int n = a.size(), ic = qpow(c, mod - 2, mod);
        poly b(n + m - 1), v(std::max(n, m));
        b[0] = 1;
        for (int i = 1, w = 1; i < b.size(); i ++, mul(w, c)) {
            b[i] = 1ll * b[i - 1] * w % mod;
        }
        v[0] = 1;
        for (int i = 1, w = 1; i < v.size(); i ++, mul(w, ic)) {
            v[i] = 1ll * v[i - 1] * w % mod;
        }

        for (int i = 0; i < n; i ++) {
            mul(a[i], v[i]);
        }
        poly res = b.mulT(a).modxk(m);
        for (int i = 0; i < m; i ++) {
            mul(res[i], v[i]);
        }
        return res;
    }
};

// 多项式快速插值
poly lagrange(std::vector<std::pair<int, int>> seq) {
    int n = seq.size(); // seq 下标从 0 开始，至多 n-1 次多项式需要 n 个点值
    std::vector<poly> t(n * 4);
    std::vector<int> x(n);
    for (int i = 0; i < n; i ++) {
        x[i] = seq[i].first;
    }

    std::function<void(int, int, int)> build = [&] (int p, int l, int r) {
        if (l == r) {
            t[p] = poly{norm(-x[l]), 1};
            return;
        }
        int mid = (l + r) >> 1;
        build(p * 2, l, mid), build(p * 2 + 1, mid + 1, r);
        t[p] = t[p * 2] * t[p * 2 + 1];
    };
    build(1, 0, n - 1);

    auto res = (t[1].deriv()).eval(x);
    std::function<poly(int, int, int)> solve = [&] (int p, int l, int r) {
        if (l == r) {
            return poly{norm(1ll * seq[l].second * qpow(res[l], mod - 2,
             ↪  mod))};
        }
        int mid = (l + r) >> 1;
        return solve(p * 2, l, mid) * t[p * 2 + 1] + solve(p * 2 + 1, mid +
         ↪  1, r) * t[p * 2];
```

```
    };
    return solve(1, 0, n - 1);
}

// 连续点值平移
std::vector<int> shift_conti(std::vector<int> y, int c) {
    int n = y.size() - 1; // y 下标从 0 开始，连续点值为 f(0), ..., f(n)
    std::vector<int> fact(n + 1), facv(n + 1);
    fact[0] = 1;
    for (int i = 1; i <= n; i ++) {
        fact[i] = 1ll * fact[i - 1] * i % mod;
    }
    facv[n] = qpow(fact[n], mod - 2, mod);
    for (int i = n - 1; i >= 0; i --) {
        facv[i] = 1ll * facv[i + 1] * (i + 1) % mod;
    }

    poly a(2 * n + 1), b(n + 1);
    for (int i = 0; i <= 2 * n; i ++) {
        a[i] = qpow(norm(c - n + i), mod - 2, mod);
    }
    for (int i = 0; i <= n; i ++) {
        b[i] = 1ll * facv[i] * facv[n - i] % mod * norm(y[i]) % mod;
        if ((n - i) & 1) neg(b[i]);
    }

    poly res = a * b;
    int num = 1, z = 0;
    for (int i = c - n; i <= c; i ++) {
        int v = norm(i);
        if (v) mul(num, v); else z ++;
    }
    std::vector<int> ans(n + 1);
    for (int i = 0; i <= n; i ++) {
        ans[i] = z ? y[norm(c + i)] : 1ll * num * res[i + n] % mod;
        int v1 = norm(c - n + i), v2 = norm(c + i + 1);
        if (v1) mul(num, a[i]); else z --;
        if (v2) mul(num, v2); else z ++;
    }
    return ans;
}
```

Bostan–Mori

分式远处系数

0x31 分式远处系数.cpp

```
// 分式远处系数
int BostanMori(poly p, poly q, int n) {
    for (; n; n >>= 1) {
        poly nq = q;
        for (int i = 1; i < nq.size(); i += 2) {
            neg(nq[i]);
        }
        p *= nq, q *= nq;

        int i = n & 1;
        while (i < p.size()) {
            p[i / 2] = p[i], i += 2;
        }
        p.resize(i / 2);

        int j = 0;
        while (j < q.size()) {
            q[j / 2] = q[j], j += 2;
        }
        q.resize(j / 2);
    }
    return 1ll * p[0] * qpow(q[0], mod - 2, mod) % mod;
}
```

常系数齐次线性递推

Fiduccia 算法

0x31 常系数齐次线性递推（Fiduccia）.cpp

```
// 常系数齐次线性递推（Fiduccia）
int linearRecurrence(std::vector<int> c, std::vector<int> y, int n) {
    int k = y.size(); // k 阶齐次线性递推数列
    if (n < k) return norm(y[n]);

    poly g(k + 1);
    g[k] = 1;
    for (int i = 1; i <= k; i ++) {
        g[k - i] = norm(-c[i]);
```

```
    }

    poly res{1}, a{0, 1};
    for (int b = n; b; b >>= 1) {
        if (b & 1) res = divmod(res * a, g).second;
        a = divmod(a * a, g).second;
    }

    int ans = 0;
    for (int i = 0; i < res.size(); i ++) {
        ans = (ans + 1ll * y[i] * res[i]) % mod;
    }
    return norm(ans);
}
```

递推式转分式

0x31 常系数齐次线性递推（递推式转分式）.cpp

```
// 常系数齐次线性递推（递推式转分式）
int linearRecurrence(std::vector<int> c, std::vector<int> y, int n) {
    int k = y.size(); // k 阶齐次线性递推数列
    if (n < k) return norm(y[n]);

    for (int &x : c) x = norm(x);
    for (int &x : y) x = norm(x);

    poly q = poly{1} - poly(c);
    poly p = (poly(y) * q).modxk(k);
    return BostanMori(p, q, n);
}
```

整式递推

应用：快速阶乘算法

0x31 快速阶乘算法.cpp

```
// 快速阶乘算法
int fastFactorial(int n) {
    if (n == 0) return 1;
    if (n >= mod) return 0;

    int v = sqrt(n), iv = qpow(v, mod - 2, mod);

    std::function<std::vector<int>(int)> solve = [&] (int d) {
        if (d == 1) {
            return std::vector<int>{1, v + 1};
        }
        if (d & 1) {
            auto y = solve(d - 1);
            for (int i = 0; i < y.size(); i ++) {
                mul(y[i], i * v + d);
            }
            int num = 1;
            for (int i = 1; i <= d; i ++) {
                mul(num, d * v + i);
            }
            y.push_back(num);
            return y;
        } else {
            auto y = solve(d / 2);
            auto y1 = shift_conti(y, d / 2);
            y.insert(y.end(), y1.begin() + 1, y1.end());
            auto y2 = shift_conti(y, 1ll * (d / 2) * iv % mod);
            for (int i = 0; i < y.size(); i ++) {
                mul(y[i], y2[i]);
            }
            return y;
        }
    };

    auto res = solve(v);
    int ans = 1;
    for (int i = 0; i < v; i ++) {
        mul(ans, res[i]);
    }
    for (int i = v * v + 1; i <= n; i ++) {
        mul(ans, i);
    }
    return ans;
}
```

Power Projection

0x31 Power Projection.cpp

```cpp
// Power Projection
poly PowerProjection(poly a, poly b, int n) {
    // assert(n >= 1);
    // assert(!b.size() || !b[0]);

    a.resize(n), b.resize(n);
    for (int &x : b) {
        neg(x);
    }

    int n0 = n --;
    for (int m = 1; n; m <<= 1, n >>= 1) {
        int L = n * 2 + 2, len = 1;
        while (len < 2 * m * L) len <<= 1;

        poly d(len);
        for (int i = 0; i < m; i ++) {
            for (int j = 0; j <= n; j ++) {
                d[i * L + j] = b[i * (n + 1) + j];
            }
        }
        dft(d);

        int h = len >> 1;
        poly work(h);
        for (int i = 0; i < h; i ++) {
            work[i] = 1ll * d[i] * d[i + h] % mod;
            std::swap(d[i], d[i + h]);
        }
        idft(work);

        int t = n / 2 + 1;
        poly p(2 * m * t), q(2 * m * t);
        for (int i = 0; i < 2 * m - 1; i ++) {
            for (int j = 0; j < t; j ++) {
                q[i * t + j] = work[i * (n + 1) + j];
            }
        }
        for (int i = 0; i < m; i ++) {
            for (int j = 0; j < t; j ++) {
                add(q[(m + i) * t + j], b[i * (n + 1) + 2 * j]);
                add(q[(m + i) * t + j], b[i * (n + 1) + 2 * j]);
            }
        }

        poly c(len);
        for (int i = 0; i < m; i ++) {
            for (int j = 0; j <= n; j ++) {
                c[i * L + j] = a[i * (n + 1) + j];
            }
        }
        dft(c);
        for (int i = 0; i < len; i ++) {
            mul(c[i], d[i]);
        }
        idft(c);

        for (int i = 0; i < 2 * m - 1; i ++) {
            for (int j = 0; j < t; j ++) {
                p[i * t + j] = c[i * L + 2 * j + (n & 1)];
            }
        }
        for (int i = 0; i < m; i ++) {
            for (int j = 0; j < t; j ++) {
                add(p[(m + i) * t + j], a[i * (n + 1) + 2 * j + (n & 1)]);
            }
        }
        a.swap(p), b.swap(q);
    }

    std::reverse(a.begin(), a.end());
    a.resize(n0);
    return a;
}
```

0x32 数学 集合幂级数

FWT

位运算卷积

0x32 FWT.cpp

```cpp
void dwtOr(std::vector<int> &a, int opt) {
    int n = a.size();
    for (int k = 1; k < n; k <<= 1) {
        for (int i = 0; i < n; i += (k << 1)) {
            for (int j = 0; j < k; j ++) {
                opt > 0 ? add(a[i + j + k], a[i + j]) : dec(a[i + j + k], a[i + j]);
            }
        }
    }
}
void dwtAnd(std::vector<int> &a, int opt) {
    int n = a.size();
    for (int k = 1; k < n; k <<= 1) {
        for (int i = 0; i < n; i += (k << 1)) {
            for (int j = 0; j < k; j ++) {
                opt > 0 ? add(a[i + j], a[i + j + k]) : dec(a[i + j], a[i + j + k]);
            }
        }
    }
}
void dwtXor(std::vector<int> &a, int opt) {
    int n = a.size();
    for (int k = 1; k < n; k <<= 1) {
        for (int i = 0; i < n; i += (k << 1)) {
            for (int j = 0; j < k; j ++) {
                int u = a[i + j], v = a[i + j + k];
                add(a[i + j] = u, v), dec(a[i + j + k] = u, v);
            }
        }
    }
    if (opt < 0) {
        int inv = qpow(n, mod - 2, mod);
        for (int i = 0; i < n; i ++) {
            mul(a[i], inv);
        }
    }
}

// 位运算卷积
using dwtFunc = void(*)(std::vector<int>&, int);
std::vector<int> conv(std::vector<int> a, std::vector<int> b, dwtFunc dwt) {
    dwt(a, +1), dwt(b, +1);
    for (int i = 0; i < a.size(); i ++) {
        mul(a[i], b[i]);
    }
    dwt(a, -1);
    return a;
}
```

子集 dp

```cpp
// 注意：要先枚举维度 i，再枚举状态 S
for (int i = 0; i < m; i ++) {
    for (int S = 0; S < (1 << m); S ++) {
        if (S >> i & 1) {
            f[S] += f[S ^ (1 << i)];
        }
    }
}


for (int s = 1; s <= m; s ++) {
    for (int S = 0; S < (1 << m); S ++) {
        tmp[S] = f[S];
    }
    for (int i = 0; i < m; i ++) {
        for (int S = 0; S < (1 << m); S ++) {
            if (S >> i & 1) {
                tmp[S] += tmp[S ^ (1 << i)];
            }
        }
    }

    for (int S = 0; S < (1 << m); S ++) {
        if (__builtin_popcount(S) == s) {
            // 使用 tmp[S] 更新 f[S]
        }
    }
}
```

子集卷积

0x32 子集卷积.cpp

```cpp
// 子集卷积
std::vector<int> subset_conv(std::vector<int> a, std::vector<int> b) {
    int n = a.size(), m = std::__lg(n);
    std::vector<std::vector<int>> A(m + 1, std::vector<int>(n, 0));
    std::vector<std::vector<int>> B(m + 1, std::vector<int>(n, 0));
    std::vector<std::vector<int>> C(m + 1, std::vector<int>(n, 0));

    for (int i = 0; i < n; i ++) {
        int s = __builtin_popcount(i);
        A[s][i] = a[i], B[s][i] = b[i];
    }
    for (int i = 0; i <= m; i ++) {
        dwtOr(A[i], +1), dwtOr(B[i], +1);
    }
    for (int i = 0; i <= m; i ++) {
        for (int j = 0; j <= i; j ++) {
            for (int k = 0; k < n; k ++) {
                add(C[i][k], 1ll * A[j][k] * B[i - j][k] % mod);
            }
        }
    }
    for (int i = 0; i <= m; i ++) {
        dwtOr(C[i], -1);
    }

    std::vector<int> c(n);
    for (int i = 0; i < n; i ++) {
        c[i] = C[__builtin_popcount(i)][i];
    }
    return c;
}
```

0x32 半在线子集卷积.cpp

```cpp
// 半在线子集卷积
std::vector<int> subset_conv(std::vector<int> g) {
    int n = g.size(), m = std::__lg(n);
    std::vector<std::vector<int>> F(m + 1, std::vector<int>(n, 0));
    std::vector<std::vector<int>> G(m + 1, std::vector<int>(n, 0));

    for (int i = 1; i < n; i ++) { // 一般来说 g_0 = 0，所以不考虑 G[0]
        G[__builtin_popcount(i)][i] = g[i];
    }
    for (int i = 1; i <= m; i ++) {
        dwtOr(G[i], +1);
    }

    int f0 = 1;
    std::fill(F[0].begin(), F[0].end(), f0); // 一般来说 f_0 = 1，这一步模拟了 dwtOr(F[0], +1)

    std::vector<int> f(n);
    f[0] = f0;
    for (int i = 1; i <= m; i ++) {
        for (int j = 1; j <= i; j ++) {
            for (int k = 0; k < n; k ++) {
                add(F[i][k], 1ll * G[j][k] * F[i - j][k] % mod);
            }
        }
        dwtOr(F[i], -1);

        for (int S = 0; S < n; S ++) {
            if (__builtin_popcount(S) == i) {
                // 在此更新 F[i][S]，即为真正的 f[S]
                f[S] = F[i][S];
            } else {
                F[i][S] = 0;
            }
        }
        dwtOr(F[i], +1);
    }
    return f;
}
```

0x33 数学 组合计数

组合数常见求法

求法 1：递推

```cpp
for (int i = 0; i <= n; i ++) {
    C[i][0] = C[i][i] = 1;
    for (int j = 1; j < i; j ++) {
        C[i][j] = (C[i - 1][j] + C[i - 1][j - 1]) % mod;
    }
}
```

求法 2：预处理阶乘与阶乘逆元

0x33 组合数（预处理阶乘与阶乘逆元）.cpp

```cpp
// 组合数（预处理阶乘与阶乘逆元）
struct BinomCoef {
    std::vector<int> fact, facv;

    BinomCoef() {}
    BinomCoef(int n) {
        init(n);
    }

    void init(int n) {
        fact.resize(n + 1), facv.resize(n + 1);
        fact[0] = 1;
        for (int i = 1; i <= n; i ++) {
            fact[i] = 1ll * fact[i - 1] * i % mod;
        }
        facv[n] = qpow(fact[n], mod - 2, mod);
        for (int i = n - 1; i >= 0; i --) {
            facv[i] = 1ll * facv[i + 1] * (i + 1) % mod;
        }
    }

    int binom(int n, int m) {
        if (n < m || m < 0) {
            return 0;
        }
        return 1ll * facv[m] * facv[n - m] % mod * fact[n] % mod;
    }
    int perm(int n, int m) {
        if (n < m || m < 0) {
            return 0;
        }
        return 1ll * fact[n] * facv[n - m] % mod;
    }
} bc;
```

图论计数

有标号无向连通图计数

算法二：多项式 ln

0x33 有标号无向连通图计数.cpp

```cpp
// 有标号无向连通图计数
poly GraphCount(int n) {
    std::vector<int> fact(n + 1), facv(n + 1);
    fact[0] = 1;
    for (int i = 1; i <= n; i ++) {
        fact[i] = 1ll * fact[i - 1] * i % mod;
    }
    facv[n] = qpow(fact[n], mod - 2, mod);
    for (int i = n - 1; i >= 0; i --) {
        facv[i] = 1ll * facv[i + 1] * (i + 1) % mod;
    }

    poly g(n + 1);
    for (int i = 0; i <= n; i ++) {
        g[i] = 1ll * qpow(2, 1ll * i * (i - 1) / 2, mod) * facv[i] % mod;
    }
    poly f = g.ln(n + 1);
    for (int i = 0; i <= n; i ++) {
        mul(f[i], fact[i]);
    }
    return f;
}
```

有标号无向欧拉图计数

0x33 有标号无向欧拉图计数.cpp

```cpp
// 有标号无向欧拉图计数
poly GraphCount(int n) {
    std::vector<int> fact(n + 1), facv(n + 1);
    fact[0] = 1;
    for (int i = 1; i <= n; i ++) {
        fact[i] = 1ll * fact[i - 1] * i % mod;
    }
    facv[n] = qpow(fact[n], mod - 2, mod);
    for (int i = n - 1; i >= 0; i --) {
        facv[i] = 1ll * facv[i + 1] * (i + 1) % mod;
```

```
    }

    poly g(n + 1);
    g[0] = 1;
    for (int i = 1; i <= n; i ++) {
        g[i] = 1ll * qpow(2, 1ll * (i - 1) * (i - 2) / 2, mod) * facv[i] %
        ↪  mod;
    }
    poly f = g.ln(n + 1);
    for (int i = 0; i <= n; i ++) {
        mul(f[i], fact[i]);
    }
    return f;
}
```

有标号无向二分图计数

0x33 有标号无向二分图计数.cpp

```
// 有标号无向二分图计数
poly GraphCount(int n) {
    std::vector<int> fact(n + 1), facv(n + 1);
    fact[0] = 1;
    for (int i = 1; i <= n; i ++) {
        fact[i] = 1ll * fact[i - 1] * i % mod;
    }
    facv[n] = qpow(fact[n], mod - 2, mod);
    for (int i = n - 1; i >= 0; i --) {
        facv[i] = 1ll * facv[i + 1] * (i + 1) % mod;
    }

    poly a(n + 1);
    for (int i = 0, i2 = qpow(2, mod - 2, mod); i <= n; i ++) {
        a[i] = 1ll * facv[i] * qpow(i2, 1ll * i * (i - 1) / 2, mod) % mod;
    }
    poly g = (a * a).modxk(n + 1);
    for (int i = 0; i <= n; i ++) {
        mul(g[i], qpow(2, 1ll * i * (i - 1) / 2, mod));
    }
    poly f = g.sqrt(n + 1);
    for (int i = 0; i <= n; i ++) {
        mul(f[i], fact[i]);
    }
    return f;
}
```

有标号 DAG 计数

0x33 有标号 DAG 计数.cpp

```
// 有标号 DAG 计数
poly GraphCount(int n) {
    std::vector<int> fact(n + 1), facv(n + 1);
    fact[0] = 1;
    for (int i = 1; i <= n; i ++) {
        fact[i] = 1ll * fact[i - 1] * i % mod;
    }
    facv[n] = qpow(fact[n], mod - 2, mod);
    for (int i = n - 1; i >= 0; i --) {
        facv[i] = 1ll * facv[i + 1] * (i + 1) % mod;
    }

    poly g(n + 1);
    g[0] = 0;
    for (int i = 1, i2 = qpow(2, mod - 2, mod); i <= n; i ++) {
        g[i] = 1ll * facv[i] * qpow(i2, 1ll * i * (i - 1) / 2, mod) % mod;
        if ((i + 1) & 1) neg(g[i]);
    }
    poly f = (poly{1} - g).inv(n + 1);
    for (int i = 0; i <= n; i ++) {
        mul(f[i], fact[i]), mul(f[i], qpow(2, 1ll * i * (i - 1) / 2, mod));
    }
    return f;
}
```

有标号弱连通 DAG 计数

0x33 有标号弱连通 DAG 计数.cpp

```
// 有标号弱连通 DAG 计数
poly GraphCount(int n) {
    std::vector<int> fact(n + 1), facv(n + 1);
    fact[0] = 1;
    for (int i = 1; i <= n; i ++) {
        fact[i] = 1ll * fact[i - 1] * i % mod;
    }
    facv[n] = qpow(fact[n], mod - 2, mod);
    for (int i = n - 1; i >= 0; i --) {
        facv[i] = 1ll * facv[i + 1] * (i + 1) % mod;
    }

```

```
    poly g(n + 1);
    g[0] = 0;
    for (int i = 1, i2 = qpow(2, mod - 2, mod); i <= n; i ++) {
        g[i] = 1ll * facv[i] * qpow(i2, 1ll * i * (i - 1) / 2, mod) % mod;
        if ((i + 1) & 1) neg(g[i]);
    }
    poly f = (poly{1} - g).inv(n + 1);
    for (int i = 0; i <= n; i ++) {
        mul(f[i], qpow(2, 1ll * i * (i - 1) / 2, mod));
    }
    f = f.ln(n + 1);
    for (int i = 0; i <= n; i ++) {
        mul(f[i], fact[i]);
    }
    return f;
}
```

有标号强连通图计数

0x33 有标号强连通图计数.cpp

```
// 有标号强连通图计数
poly GraphCount(int n) {
    std::vector<int> fact(n + 1), facv(n + 1);
    fact[0] = 1;
    for (int i = 1; i <= n; i ++) {
        fact[i] = 1ll * fact[i - 1] * i % mod;
    }
    facv[n] = qpow(fact[n], mod - 2, mod);
    for (int i = n - 1; i >= 0; i --) {
        facv[i] = 1ll * facv[i + 1] * (i + 1) % mod;
    }

    poly h(n + 1);
    h[0] = 1;
    for (int i = 1; i <= n; i ++) {
        h[i] = 1ll * qpow(2, 1ll * i * (i - 1) / 2, mod) * facv[i] % mod;
    }
    poly g2 = poly{1} - h.inv(n + 1);
    for (int i = 1; i <= n; i ++) {
        mul(g2[i], qpow(2, 1ll * i * (i - 1) / 2, mod));
    }
    poly g1 = g2;
    for (int i = 1; i <= n; i ++) {
        mul(g1[i], i);
    }
    }
    poly f = (g1 * (poly{1} - g2).inv(n + 1)).modxk(n + 1);
    for (int i = 1; i <= n; i ++) {
        mul(f[i], fact[i - 1]);
    }
    return f;
}
```

0x34 数学 初等数论–质数

质数

埃氏筛

0x34 埃氏筛.cpp

```
// 埃氏筛
std::bitset<MaxV> vis;
void sieve(int n) {
    for (int i = 2; i <= n; i ++) {
        if (vis[i]) {
            continue;
        }
        for (int j = 2; j <= n / i; j ++) {
            vis[i * j] = 1;
        }
    }
}
```

线性筛

0x34 线性筛.cpp

```
// 线性筛
int primeCount, prime[MaxV], fac[MaxV];
void sieve(int n) {
    for (int i = 2; i <= n; i ++) {
        if (!fac[i]) {
            prime[++ primeCount] = i, fac[i] = i;
        }
        for (int j = 1; j <= primeCount; j ++) {
            if (prime[j] > fac[i] || prime[j] > n / i) break;
```

```
            fac[i * prime[j]] = prime[j];
        }
    }
}
```

狄利克雷前缀和

```
// 注意：要先枚举质数 prime[j]
for (int j = 1; j <= primeCount; j ++) {
    for (int i = 1; i <= n / prime[j]; i ++) {
        f[i * prime[j]] += f[i];
    }
}


// 注意：要先枚举质数 prime[j]
for (int j = 1; j <= primeCount; j ++) {
    for (int i = n / prime[j]; i >= 1; i --) {
        f[i] += f[i * prime[j]];
    }
}


// 注意：要先枚举质数 prime[j]
for (int j = 1; j <= primeCount; j ++) {
    for (int i = n / prime[j]; i >= 1; i --) { // 顺序与正变换相反
        f[i * prime[j]] -= f[i];
    }
}


// 注意：要先枚举质数 prime[j]
for (int j = 1; j <= primeCount; j ++) {
    for (int i = 1; i <= n / prime[j]; i ++) { // 顺序与正变换相反
        f[i] -= f[i * prime[j]];
    }
}
```

Pollard-Rho 质因数分解

0x34 素性测试与质因数分解（Miller-Rabin + Pollard-Rho）.cpp

```cpp
// 素性测试与质因数分解
template <class T, auto qmul>
struct primeTools {
    constexpr T qpow(T a, T b, T p) {
        T ans = 1;
        for (; b; b >>= 1) {
            if (b & 1) ans = qmul(ans, a, p);
            a = qmul(a, a, p);
        }
        return ans;
    }

    std::mt19937_64 mtrand{std::random_device{}()};
    T rand(T l, T r) {
        std::uniform_int_distribution<T> range(l, r);
        return range(mtrand);
    }

    // 素性测试 Miller-Rabin
    bool isPrime(T n) {
        assert(n > 1);
        if (n % 2 == 0) return n == 2;
        if (n % 3 == 0) return n == 3;

        T t = __builtin_ctzll(n - 1);
        T u = (n - 1) >> t;

        int test = 6;
        while (test --) {
            T a = rand(2, n - 1);
            T v = qpow(a, u, n);
            if (v == 1 || v == n - 1) {
                continue;
            }
            for (int i = 1; i < t; i ++) {
                v = qmul(v, v, n);
                if (v == n - 1) {
                    break;
                }
            }
            if (v != n - 1) {
                return 0;
            }
        }
```

```
        return 1;
    }

    T walk(T n) {
        T c = rand(1, n - 1);
        T x = rand(1, n - 1), y, v, d;
        for (int k = 1;; k <<= 1) {
            y = x, v = 1;
            for (int i = 1; i <= k; i ++) {
                x = (qmul(x, x, n) + c) % n;
                v = qmul(v, x - y + n, n);
                if (i % 127 == 0 || i == k) {
                    d = std::gcd(v, n);
                    if (d > 1) return d;
                }
            }
        }
    }
    // 质因数分解 Pollard-Rho
    auto primeFactor(T n) {
        std::map<T, int> buc;
        std::function<void(T)> solve = [&] (T n) {
            if (n == 1) {
                return;
            }
            if (isPrime(n)) {
                buc[n] ++;
                return;
            }
            T d = 0;
            while (d = walk(n), d == n);
            solve(d), solve(n / d);
        };
        solve(n);
        return std::vector<std::pair<T, int>>(buc.begin(), buc.end());
    }
};

// primeTools<int, [] (int a, int b, int p) { return 1ll * a * b % p; }>
 ↪  mrpr;
// primeTools<i64, [] (i64 a, i64 b, i64 p) { return i128(a) * b % p; }>
 ↪  mrpr;
// primeTools<i64, [] (i64 a, i64 b, i64 p) {
//     i64 c = static_cast<long double>(a) * b / p + 1e-8;
//     i64 ans = a * b - c * p;
//     if (ans < 0) ans += p;
//     if (ans >= p) ans -= p;
//     return ans;
// }> mrpr;
```

0x34 数学 初等数论–约数

最大公约数常见求法

求法 1：辗转相除法

```
// 辗转相除法
int gcd(int a, int b) {
    return b == 0 ? a : gcd(b, a % b);
}
```

求法 2：更相减损术

0x34 gcd（更相减损术）.cpp

```
// 更相减损术
int gcd(int a, int b) {
    if (a == b) return a;
    if (a == 0 || b == 0) return a ^ b;
    int az = __builtin_ctz(a), bz = __builtin_ctz(b), z = az < bz ? az : bz,
     ↪  v = 0;
    a >>= az, b >>= bz;
    while (b) {
        v = b - a;
        a = a < b ? a : b;
        b = v < 0 ? -v : v;
        if (b) b >>= __builtin_ctz(b);
    }
    return a << z;
}
```

求法 3：递推

```
for (int i = 0; i <= n; i ++) {
    g[i][0] = g[0][i] = i;
    for (int j = 1; j <= i; j ++) {
        g[i][j] = g[j][i] = g[i - j][j];
    }
}
```

求法 4：基于值域预处理的快速 gcd

0x34 gcd（基于值域预处理的快速 gcd）.cpp

```cpp
// O(n)-O(1) gcd
struct flashGCD {
    int b;
    std::vector<int> prime, fac;
    std::vector<std::array<int, 3>> s;
    std::vector<std::vector<int>> g;

    flashGCD() {}
    flashGCD(int n) {
        init(n);
    }

    void init(int n) {
        b = sqrt(n);
        prime.clear(), fac.assign(n + 1, 0);
        s.assign(n + 1, {});
        g.assign(b + 1, std::vector<int>(b + 1, 0));

        for (int i = 2; i <= n; i ++) {
            if (!fac[i]) {
                prime.push_back(i), fac[i] = i;
            }
            for (int p : prime) {
                if (p > fac[i] || p > n / i) break;
                fac[i * p] = p;
            }
        }

        s[1] = {1, 1, 1};
        for (int i = 2; i <= n; i ++) {
            s[i] = s[i / fac[i]], s[i][0] *= fac[i];
            if (s[i][0] > s[i][1]) {
                std::swap(s[i][0], s[i][1]);
            }
            if (s[i][1] > s[i][2]) {
                std::swap(s[i][1], s[i][2]);
            }
        }

        for (int i = 0; i <= b; i ++) {
            g[i][0] = g[0][i] = i;
            for (int j = 1; j <= i; j ++) {
                g[i][j] = g[j][i] = g[i - j][j];
            }
        }
    }

    int gcd(int p, int q) {
        if (p == 0 || q == 0) {
            return p ^ q;
        }
        int ans = 1;
        for (int x : s[q]) {
            int t = x <= b ? g[x][p % x] : (p % x ? 1 : x);
            p /= t, ans *= t;
        }
        return ans;
    }
} fg;
```

类欧几里得算法

0x34 类欧几里得算法.cpp

```cpp
// 类欧几里得算法（要求 a, b >= 0 以及 c > 0）
i64 euclid(i64 a, i64 b, i64 c, i64 n) {
    if (a == 0) {
        return (n + 1) * (b / c);
    }
    if (a >= c || b >= c) {
        return euclid(a % c, b % c, c, n) + (n * (n + 1) / 2) * (a / c) + (n
        ↪  + 1) * (b / c);
    }
    i64 m = (a * n + b) / c;
    if (m == 0) {
        return 0;
    }
    return n * m - euclid(c, c - b - 1, a, m - 1);
}
// 类欧几里得算法
i64 euclid_norm(i64 a, i64 b, i64 c, i64 n) {
    if (c < 0) {
        a = -a, b = -b, c = -c;
    }
    i64 sum = 0;
    if (a < 0) {
        i64 t = (a - c + 1) / c;
```

```
        a -= t * c, sum += (n * (n + 1) / 2) * t;
    }
    if (b < 0) {
        i64 t = (b - c + 1) / c;
        b -= t * c, sum += (n + 1) * t;
    }
    return euclid(a, b, c, n) + sum;
}
```

0x34 数学 初等数论–取整

数论分块

下取整数论分块

```
// h 表示 x 的上界
for (int x = 1, nx; x <= h; x = nx + 1) {
    nx = h;
    if (x <= n) {
        chmin(nx, n / (n / x));
    }
    // 区间 [x, nx] 内：下取整的值均为 n/x
}
```

上取整数论分块

```
// h 表示 x 的上界
for (int x = 1, nx; x <= h; x = nx + 1) {
    nx = h;
    if (x < n) {
        chmin(nx, (n - 1) / ((n - 1) / x));
    }
    // 区间 [x, nx] 内：上取整的值均为 (n-1)/x+1
}
```

多维数论分块

```
// h 表示 x 的上界
for (int x = 1, nx; x <= h; x = nx + 1) {
    nx = h;
    if (x <= n) {
        chmin(nx, n / (n / x));
    }
    if (x <= m) {
        chmin(nx, m / (m / x));
    }
    // 区间 [x, nx] 内：n, m 对 x 下取整的值分别为 n/x, m/x
}
```

积性函数与狄利克雷卷积

线性筛筛积性函数

0x34 线性筛筛积性函数.cpp

```
// 线性筛筛积性函数
int primeCount, prime[MaxV], low[MaxV];
int f[MaxV];
void sieve(int n) {
    f[1] = 1;
    for (int i = 2; i <= n; i ++) {
        if (!low[i]) {
            prime[++ primeCount] = i, low[i] = i;
            for (i64 v = i, c = 1; v <= n; v *= i, c ++) {
                // 计算 f[v]
            }
        }
        for (int j = 1; j <= primeCount; j ++) {
            if (prime[j] > n / i) break;
            low[i * prime[j]] = i % prime[j] ? prime[j] : low[i] * prime[j];

            if (i % prime[j] == 0) break;
        }
        if (low[i] < i) {
            f[i] = f[low[i]] * f[i / low[i]];
        }
    }
}
```

杜教筛

0x34 杜教筛.cpp

```cpp
namespace du {
    const int V = 1e6; // 预处理的范围（需根据实际问题调整）
    const int MaxV = V + 10;

    int primeCount, prime[MaxV], low[MaxV];
    int f[MaxV];

    i64 f1[MaxV];
    std::map<int, i64> f2;

    void init(int n = V) {
        f[1] = 1;
        for (int i = 2; i <= n; i ++) {
            if (!low[i]) {
                prime[++ primeCount] = i, low[i] = i;
                for (i64 v = i, c = 1; v <= n; v *= i, c ++) {
                    // 计算 f[v]
                }
            }
            for (int j = 1; j <= primeCount; j ++) {
                if (prime[j] > n / i) break;
                low[i * prime[j]] = i % prime[j] ? prime[j] : low[i] *
                 ↪  prime[j];

                if (i % prime[j] == 0) break;
            }
            if (low[i] < i) {
                f[i] = f[low[i]] * f[i / low[i]];
            }
        }

        for (int i = 1; i <= n; i ++) {
            f1[i] = f1[i - 1] + f[i];
        }
    }

    i64 fSum(int n) {
        if (n <= V) return f1[n];
        if (f2.contains(n)) return f2[n];

        i64 ans = 0; // 初始计算函数 f*g 第 1~n 项的和
        for (int x = 2, nx; x <= n; x = nx + 1) {
            nx = n / (n / x);
            i64 sum = 0; // 计算函数 g 第 x~nx 项的和
            ans -= sum * fSum(n / x);
        }
        return f2[n] = ans;
    }
}
```

0x34 数学 初等数论–同余

模意义下逆元

单个逆元的求解

0x34 单个逆元的求解.cpp

```cpp
// 单个逆元的求解
i64 inverse(i64 a, i64 m) {
    auto [d, x, y] = exgcd(a, m);
    // assert(d == 1);
    return (x % m + m) % m;
}
```

预处理线性逆元

0x34 预处理线性逆元.cpp

```cpp
// 预处理线性逆元
std::vector<int> v;
void linearInverse(int n) {
    v.resize(n + 1);
    v[1] = 1;
    for (int i = 2; i <= n; i ++) {
        v[i] = 1ll * v[mod % i] * (mod - mod / i) % mod;
    }
}
```

线性同余方程

0x34 exgcd（扩展欧几里得算法）.cpp

```cpp
// 扩展欧几里得算法
std::array<i64, 3> exgcd(i64 a, i64 b) {
    if (b == 0) return {a, 1, 0};
    auto [d, x, y] = exgcd(b, a % b);
    return {d, y, x - (a / b) * y};
}
```

线性同余方程组

0x34 CRT（中国剩余定理）.cpp

```cpp
i64 inverse(i64 a, i64 m) {
    auto [d, x, y] = exgcd(a, m);
    return (x % m + m) % m;
}

using equ = std::pair<i64, i64>; // {a, m} 表示方程 x = a (mod m)

// 中国剩余定理
equ CRT(std::vector<equ> seq) {
    i64 M = 1;
    for (auto [a, m] : seq) {
        M *= m;
    }
    i64 x = 0;
    for (auto [a, m] : seq) {
        i64 Mi = M / m;
        i64 Vi = inverse(Mi, m);
        x = (x + a * Mi * Vi) % M; // 必要时，使用“快速乘”防止爆 long long
    }
    x = (x % M + M) % M;
    return {x, M};
}
```

0x34 exCRT（扩展中国剩余定理）.cpp

```cpp
// 扩展中国剩余定理
struct equ {
    i64 a, m; // {a, m} 表示方程 x = a (mod m)
    equ() {}
    equ(i64 a, i64 m) : a(a), m(m) {}

    friend equ operator + (equ l, equ r) {
        auto [a1, m1] = l;
        auto [a2, m2] = r;
        if (a1 == -1 || a2 == -1) {
            return {-1, -1};
        }
        auto [d, x, y] = exgcd(m1, m2);
        if ((a2 - a1) % d) {
            return {-1, -1};
        }
        x = x * ((a2 - a1) / d) % (m2 / d); // 必要时，使用“快速乘”防止爆 long long
        if (x < 0) x += (m2 / d);
        i64 m = m1 * (m2 / d);
        i64 a = (x * m1 + a1) % m; // 必要时，使用“快速乘”防止爆 long long
        return {a, m};
    }
};
```

阶与原根

0x34 最小原根.cpp

```cpp
// 最小原根
int findPrimitiveRoot(int m) {
    if (m == 1) return 0;
    if (m == 2) return 1;

    int phi = m, x = m; // 若已知 m 为质数，则直接令 phi = m - 1
    for (int i = 2; i * i <= x; i ++) {
        if (x % i == 0) {
            phi = phi / i * (i - 1);
            while (x % i == 0) {
                x /= i;
            }
        }
    }
    if (x > 1) {
        phi = phi / x * (x - 1);
    }

    std::vector<int> pf;
    x = phi;
```

```
    for (int i = 2; i * i <= x; i ++) {
        if (x % i == 0) {
            pf.push_back(i);
            while (x % i == 0) {
                x /= i;
            }
        }
    }
    if (x > 1) {
        pf.push_back(x);
    }

    for (int g = 2; g < m; g ++) {
        if (std::gcd(g, m) != 1) { // 若已知 m 为质数，则可以省略该判断
            continue;
        }
        int flag = 1;
        for (int p : pf) {
            if (qpow(g, phi / p, m) == 1) {
                flag = 0;
                break;
            }
        }
        if (flag) {
            return g;
        }
    }
    return -1; // 若 m 较大，建议先判断 m 是否存在原根
}
```

0x34 所有原根.cpp

```
// 所有原根
std::vector<int> allPrimitiveRoot(int m) {
    if (m == 1) return {0};
    if (m == 2) return {1};

    int phi = m, x = m; // 若已知 m 为质数，则直接令 phi = m - 1
    for (int i = 2; i * i <= x; i ++) {
        if (x % i == 0) {
            phi = phi / i * (i - 1);
            while (x % i == 0) {
                x /= i;
            }
        }
    }
    if (x > 1) {
        phi = phi / x * (x - 1);
    }

    std::vector<int> pf;
    x = phi;
    for (int i = 2; i * i <= x; i ++) {
        if (x % i == 0) {
            pf.push_back(i);
            while (x % i == 0) {
                x /= i;
            }
        }
    }
    if (x > 1) {
        pf.push_back(x);
    }

    int g = 2;
    for (; g < m; g ++) {
        if (std::gcd(g, m) != 1) { // 若已知 m 为质数，则可以省略该判断
            continue;
        }
        int flag = 1;
        for (int p : pf) {
            if (qpow(g, phi / p, m) == 1) {
                flag = 0;
                break;
            }
        }
        if (flag) {
            break;
        }
    }
    if (g == m) {
        return {};
    }

    std::vector<int> res;
    x = 1;
    for (int i = 1; i <= phi; i ++) {
        x = 1ll * x * g % m;
        if (std::gcd(i, phi) == 1) {
            res.push_back(x);
        }
```

```
    }
    std::sort(res.begin(), res.end());
    return res;
}
```

高次同余方程

0x34 BSGS（大步小步算法）.cpp

```cpp
// 大步小步算法
int BSGS(int a, int b, int p) { // a^x = b (mod p)
    if (b == 1 || p == 1) return 0;
    if (a == 0) return b == 0 ? 1 : -1;

    int S = sqrt(p) + 1;
    std::unordered_map<int, int> buc;

    int w = 1;
    for (int j = 0; j < S; j ++, w = 1ll * w * a % p) {
        int v = 1ll * w * b % p;
        buc[v] = j;
    }
    int x = w;
    for (int i = 1; i <= S; i ++, x = 1ll * x * w % p) {
        if (buc.contains(x)) {
            return i * S - buc[x];
        }
    }
    return -1;
}
```

0x34 exBSGS（扩展大步小步算法）.cpp

```cpp
// 扩展大步小步算法
int exBSGS(int a, int b, int p) { // a^x = b (mod p)
    if (b == 1 || p == 1) return 0;
    if (a == 0) return b == 0 ? 1 : -1;

    int d, r = 1, step = 0;
    while (d = std::gcd(a, p), d > 1) {
        if (b % d) {
            return -1;
        }
        b /= d, p /= d, r = 1ll * r * (a / d) % p;
        step ++;
        if (r == b) {
            return step;
        }
    }
    if (p == 1) {
        return step;
    }

    int S = sqrt(p) + 1;
    std::unordered_map<int, int> buc;

    int w = 1;
    for (int j = 0; j < S; j ++, w = 1ll * w * a % p) {
        int v = 1ll * w * b % p;
        buc[v] = j;
    }
    int x = 1ll * r * w % p;
    for (int i = 1; i <= S; i ++, x = 1ll * x * w % p) {
        if (buc.contains(x)) {
            return step + i * S - buc[x];
        }
    }
    return -1;
}
```

0x41 数据结构 序列结构

ST 表

0x41 ST 表.cpp

```cpp
// ST 表
template <class T, auto op>
struct SparseTable {
    int t;
    std::vector<std::vector<T>> f;

    SparseTable() {}
    SparseTable(int n, const std::vector<T> &a) {
        init(n, a);
    }

```

```
    void init(int n, const std::vector<T> &a) {
        t = std::__lg(n);
        f.assign(t + 1, std::vector<T>(n + 1));
        for (int i = 1; i <= n; i ++) {
            f[0][i] = a[i];
        }
        for (int j = 1; j <= t; j ++) {
            for (int i = 1; i <= n - (1 << j) + 1; i ++) {
                f[j][i] = op(f[j - 1][i], f[j - 1][i + (1 << (j - 1))]);
            }
        }
    }

    T ask(int l, int r) {
        int k = std::__lg(r - l + 1);
        return op(f[k][l], f[k][r - (1 << k) + 1]);
    }
};
```

并查集

0x41 可撤销并查集.cpp

```
// 可撤销并查集
struct DSU {
    std::vector<int> sze;
    std::vector<int> fa;
    std::vector<std::pair<int, int>> his;

    DSU() {}
    DSU(int n) {
        init(n);
    }

    void init(int n) {
        sze.assign(n + 1, 1);
        fa.resize(n + 1);
        std::iota(fa.begin(), fa.end(), 0);
        his.clear();
    }

    int get(int x) {
        return fa[x] == x ? x : get(fa[x]); // 按秩合并不能路径压缩
    }

    bool merge(int x, int y) {
        int p = get(x), q = get(y);
        if (p == q) {
            return 0;
        }
        if (sze[p] > sze[q]) {
            std::swap(p, q);
        }
        his.push_back({p, q});
        sze[q] += sze[p], fa[p] = q;
        return 1;
    }

    int time() {
        return his.size();
    }
    void revert(int tim) {
        while (his.size() > tim) {
            auto [p, q] = his.back(); his.pop_back();
            sze[q] -= sze[p], fa[p] = p;
        }
    }
};
```

线段树

势能线段树

0x41 吉老师线段树.cpp

```
struct Info {
    int mx, md;
    int cnt;
    Info() {}
    Info(int mx, int md, int cnt) : mx(mx), md(md), cnt(cnt) {}

    friend Info operator + (Info l, Info r) {
        if (l.mx == r.mx) {
            return Info(l.mx, std::max(l.md, r.md), l.cnt + r.cnt);
        } else if (l.mx > r.mx) {
            return Info(l.mx, std::max(l.md, r.mx), l.cnt);
        } else {
            return Info(r.mx, std::max(l.mx, r.md), r.cnt);
        }
    }
```

```
};

// 吉老师线段树
namespace SGT {
    struct node {
        int len;
        i64 sum;
        Info info;

        int add; // 加法标记（初始值等于 0）
        int tag; // 区间取 min 标记（初始值等于 +inf，需要根据数据范围修改）
        void mk_add(int x) {
            sum += 1ll * x * len;
            info.mx += x;
            if (info.md != -inf) info.md += x;
            add += x;
            if (tag < inf) tag += x;
        }
        void mk_tag(int x) {
            if (info.mx <= x) return;
            sum -= 1ll * (info.mx - x) * info.cnt;
            info.mx = x;
            tag = x;
        }
    } t[N * 4];

    void upd(int p) {
        t[p].sum = t[p * 2].sum + t[p * 2 + 1].sum;
        t[p].info = t[p * 2].info + t[p * 2 + 1].info;
    }

    void spread(int p) {
        if (t[p].add) {
            t[p * 2].mk_add(t[p].add), t[p * 2 + 1].mk_add(t[p].add);
            t[p].add = 0;
        }
        if (t[p].tag < inf) {
            t[p * 2].mk_tag(t[p].tag), t[p * 2 + 1].mk_tag(t[p].tag);
            t[p].tag = inf;
        }
    }

    void build(int p, int l, int r) {
        t[p].add = 0, t[p].tag = inf, t[p].len = r - l + 1;
        if (l == r) {
            t[p].sum = a[l], t[p].info = Info(a[l], -inf, 1);
            return;
        }
        int mid = (l + r) >> 1;
        build(p * 2, l, mid), build(p * 2 + 1, mid + 1, r);
        upd(p);
    }

    void change_add(int p, int l, int r, int s, int e, int x) {
        if (s <= l && r <= e) {
            t[p].mk_add(x);
            return;
        }
        spread(p);
        int mid = (l + r) >> 1;
        if (s <= mid) {
            change_add(p * 2, l, mid, s, e, x);
        }
        if (mid < e) {
            change_add(p * 2 + 1, mid + 1, r, s, e, x);
        }
        upd(p);
    }

    void change_min(int p, int l, int r, int s, int e, int x) {
        if (s <= l && r <= e && t[p].info.md < x) {
            t[p].mk_tag(x);
            return;
        }
        spread(p);
        int mid = (l + r) >> 1;
        if (s <= mid) {
            change_min(p * 2, l, mid, s, e, x);
        }
        if (mid < e) {
            change_min(p * 2 + 1, mid + 1, r, s, e, x);
        }
        upd(p);
    }
}
```

0x65 图论 二分图与网络流

最大流

Dinic

0x65 最大流（Dinic）.cpp

```cpp
template <class T>
struct MaxFlow {
    struct Edge {
        int to;
        T cap;
    };

    int n;
    std::vector<Edge> e;
    std::vector<std::vector<int>> g;
    std::vector<int> d, cur;

    MaxFlow() {}
    MaxFlow(int n) {
        init(n);
    }

    void init(int n) {
        this->n = n;
        e.clear();
        g.assign(n + 1, {});
        d.resize(n + 1), cur.resize(n + 1);
    }

    void addEdge(int u, int v, T w) {
        g[u].push_back(e.size());
        e.push_back({v, w});
        g[v].push_back(e.size());
        e.push_back({u, 0});
    }

    bool bfs(int s, int t) {
        d.assign(n + 1, 0);
        std::queue<int> q;
        q.push(s), d[s] = 1;
        while (q.size()) {
            int u = q.front(); q.pop();
            for (int i : g[u]) {
                auto [v, w] = e[i];
                if (w > 0 && !d[v]) {
                    d[v] = d[u] + 1;
                    q.push(v);
                    if (v == t) {
                        return 1;
                    }
                }
            }
        }
        return 0;
    }
    T dfs(int u, int t, T f) {
        if (u == t) {
            return f;
        }
        auto r = f;
        for (int &i = cur[u]; i < g[u].size(); i ++) {
            int j = g[u][i];
            auto [v, w] = e[j];
            if (w > 0 && d[v] == d[u] + 1) {
                auto a = dfs(v, t, std::min(r, w));
                e[j].cap -= a;
                e[j ^ 1].cap += a;
                r -= a;
                if (r == 0) {
                    return f;
                }
            }
        }
        return f - r;
    }

    T dinic(int s, int t) {
        T ans = 0;
        while (bfs(s, t)) {
            cur.assign(n + 1, 0);
            ans += dfs(s, t, std::numeric_limits<T>::max());
        }
        return ans;
    }
};
```

0x71 计算几何 I

计算几何基础

0x71 计算几何全家桶（i64）.cpp

```cpp
// 二维点
struct point {
    i64 x, y;
    point() {}
    point(i64 x, i64 y) : x(x), y(y) {}
};

point operator + (point a, point b) {
    return point(a.x + b.x, a.y + b.y);
}
point operator - (point a, point b) {
    return point(a.x - b.x, a.y - b.y);
}
point operator * (point a, i64 b) {
    return point(a.x * b, a.y * b);
}

// 叉积
i128 cross(point a, point b) {
    return i128(a.x) * b.y - i128(a.y) * b.x;
}
// ab 与 ac 的叉积
// 检测连续三个点的凸性，按 x 递增时 >0 下凸 <0 上凸，按 x 递减时反之
i128 cross(point a, point b, point c) {
    return i128(b.x - a.x) * (c.y - a.y) - i128(c.x - a.x) * (b.y - a.y);
}
```

0x72 计算几何 II

凸壳

0x72 凸壳.cpp

```cpp
// 凸壳（默认：按 x 递增的上凸壳）
auto gethull(std::vector<point> s) {
    std::sort(s.begin(), s.end(), [] (point l, point r) -> bool {
        return l.x != r.x ? l.x < r.x : l.y > r.y; // 若要求下凸壳，改成 l.y < r.y
    });

    std::vector<point> h;
    for (point p : s) {
        if (h.size() && h.back().x == p.x) {
            continue;
        }
        while (h.size() >= 2 && cross(h.rbegin()[1], h.back(), p) >= 0) { // 若要求下凸壳，改成 <=
            h.pop_back();
        }
        h.push_back(p);
    }
    return h;
}
```

凸包

0x72 凸包.cpp

```cpp
// 凸包（默认：逆时针）
auto gethull(std::vector<point> s) {
    std::sort(s.begin(), s.end(), [] (point l, point r) -> bool {
        return l.x != r.x ? l.x < r.x : l.y < r.y;
    });
    s.erase(std::unique(s.begin(), s.end(), [] (point l, point r) -> bool {
        return l.x == r.x && l.y == r.y;
    }), s.end());

    std::vector<point> h;
    for (int i = 0; i < s.size(); i ++) {
        while (h.size() >= 2 && cross(h.rbegin()[1], h.back(), s[i]) <= 0) {
            h.pop_back();
```

```
        }
        h.push_back(s[i]);
    }
    for (int i = (int)s.size() - 2, t = h.size(); i >= 0; i --) {
        while (h.size() > t && cross(h.rbegin()[1], h.back(), s[i]) <= 0) {
            h.pop_back();
        }
        h.push_back(s[i]);
    }
    if (h.size() > 1) {
        h.pop_back();
    }
    return h;
}
```

闵可夫斯基和

0x72 闵可夫斯基和.cpp

```
// 闵可夫斯基和（默认：逆时针）
auto minkowski(std::vector<point> a, std::vector<point> b) {
    if (a.size() <= 2 || b.size() <= 2) { // 若 a, b 均为凸壳，可以省去这一段特判
        std::vector<point> c;
        for (point p : a) for (point q : b) c.push_back(p + q);
        return gethull(c);
    }

    std::vector<point> c{a[0] + b[0]};
    a.push_back(a[0]), b.push_back(b[0]); // 若 a, b 均为凸壳，省去此行
    for (int i = 0; i + 1 < a.size(); i ++) {
        a[i] = a[i + 1] - a[i];
    }
    for (int i = 0; i + 1 < b.size(); i ++) {
        b[i] = b[i + 1] - b[i];
    }
    a.pop_back(), b.pop_back();
    c.resize(a.size() + b.size() + 1);
    std::merge(a.begin(), a.end(), b.begin(), b.end(), c.begin() + 1, [](point l, point r) {
        return cross(l, r) > 0; // 若 a, b 均为顺时针凸包（或凸壳），改成 <0
    });
    for (int i = 1; i < c.size(); i ++) {
        c[i] = c[i] + c[i - 1];
    }
    c.pop_back(); // 若 a, b 均为凸壳，省去此行
    return c; // c 可能会共线，但不会重点
}
```