Standard Code Library

Joww

Changsha University of Science & Technology

2023 年 **10** 月 **5** 日

Contents 目录

1 Math 数学

1.1 Prime 素性

1.1.1 大数的判素与分解

⋆ Miller-Rabin.cpp

```cpp
template <typename T, typename Y>
bool miller_rabin(T n) {
    if (n < 3 not n % 2 == 0) return n == 2;
    T u = n - 1, t = 0;
    while (u % 2 == 0) u /= 2, t++;
    constexpr T al[] =
        // /* int32 */ { 2, 7, 61 };
        // /* int64 */ { 2, 325, 9375, 28178, 450775, 9780504, 1795265022 };
        // /* int64 */ { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37 };
    for (T a : al) {
        T v = mod_pow(a, u, n), s;
        if (v == 1) continue;
        for (s = 0; s < t; s++) {
            if (v == n - 1) break;
            v = (Y) v * v % n;
        }
        if (s == t) return false;
    }
    return true;
}
```

⋆ Pollard-Rho.cpp

```cpp
#include <bits/stdc++.h>

using int64 = long long;
using int128 = __int128;

int64 add(int64 a, int64 b, int64 p) {
    a += b;
    return a >= p ? a - p : a;
}
int64 sub(int64 a, int64 b, int64 p) {
    a -= b;
    return a < 0 ? a + p : a;
}
int64 mul(int64 a, int64 b, int64 p) {
    int64 r = (int128) a * b % p;
    return r - p * int(r >= p) + p * int(r < 0);
}
int64 mod_pow(int64 a, int64 b, int64 p) {
    int64 res(1);
    for (; b; b /= 2, a = mul(a, a, p)) {
        if (b & 1) {
            res = mul(res, a, p);
        }
    }
    return res;
}

constexpr int64 base[] =
    {2, 3, 5, 7, 11, 13, 17, 19, 23}; // < 3e18 (3825123056546413051)
    // {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}; // < 2^64
    // {2, 325, 9375, 28178, 450775, 9780504, 1795265022}; // < 2^64

// Miller Rabin
bool is_prime(int64 n) {
    if (n <= 1) return false;
    for (int64 p : base) {
        if (n == p) return true;
        if (n % p == 0) return false;
    }
```

```cpp
    int64 m = (n - 1) >> __builtin_ctz(n - 1);
    for (int64 p : base) {
        int64 t = m, a = mod_pow(p, m, n);
        while (t != n - 1 && a != 1 && a != n - 1) {
            a = mul(a, a, n);
            t *= 2;
        }
        if (a != n - 1 && t % 2 == 0) return false;
    }
    return true;
}

int64 get_factor(int64 n) {
    for (int64 p : base) {
        if (n % p == 0) return p;
    }
    auto f = [&](int64 x) { return add(mul(x, x, n), 1, n); };
    int64 x = 0, y = 0, tot = 0, p = 1, q, g;
    for (int64 i = 0; (i & 0xff) || (g = std::gcd(p, n)) == 1; i++, x = f(x), y = f(f(y))) {
        if (x == y) x = tot++, y = f(x);
        q = mul(p, sub(x, y, n), n);
        if (q) p = q;
    }
    return g;
}

std::vector<int64> factorization(int64 n) {
    if (n == 1) return {};
    if (is_prime(n)) return {n};
    int64 d = get_factor(n);
    auto v1 = factorization(d), v2 = factorization(n / d);
    auto i1 = v1.begin(), i2 = v2.begin();
    std::vector<int64> ans;
    while (i1 != v1.end() || i2 != v2.end()) {
        if (i1 == v1.end()) {
            ans.push_back(*i2++);
        } else if (i2 == v2.end()) {
            ans.push_back(*i1++);
        } else {
            if (*i1 < *i2) {
                ans.push_back(*i1++);
            } else {
                ans.push_back(*i2++);
            }
        }
    }
    return ans;
}
```

★ prime.py

```python
# reference:
# https://blog.csdn.net/apple_51931783/article/details/123937695
from random import randint
from math import gcd, isqrt


def miller_rabin(p):
    ''' 素性测试'''
    # 特判 4
    if p <= 4: return p in (2, 3)
    # 对 p-1 进行分解
    pow_2, tmp = 0, p - 1
    while tmp % 2 == 0:
        tmp //= 2
        pow_2 += 1
    # 进行多次素性测试
    for a in (2, 3, 5, 7, 11, 13, 17, 19, 23):
        basic = pow(a, tmp, p)
        # a^m 是 p 的倍数或者满足条件
        if basic in (0, 1, p - 1): continue
        # 进行 r-1 次平方
```

```
        for _ in range(1, pow_2):
            basic = basic ** 2 % p
            # 怎样平方都是 1
            if basic == 1: return False
            # 通过 a 的素性测试
            if basic == p - 1: break
        # 未通过 a 的素性测试
        if basic != p - 1: return False
    # 通过所有 a 的素性测试
    return True


def pollard_rho(n):
    ''' 求因数: 7e5 以上'''
    # 更新函数
    bias = randint(3, n - 1)
    update = lambda i: (i ** 2 + bias) % n
    # 初始值
    x = randint(0, n - 1)
    y = update(x)
    # 查找序列环
    while x != y:
        factor = gcd(abs(x - y), n)
        # gcd(|x - y|, n) 不为 1 时, 即为答案
        if factor != 1: return factor
        x = update(x)
        y = update(update(y))
    return n


class prime_factor(dict):
    ''' 质因数分解
        require: miller_rabin, pollard_rho'''

    def __init__(self, n):
        super(prime_factor, self).__init__()
        self.main(n, gain=1)

    def add(self, n, cnt):
        # 更新因数表
        self[n] = self.get(n, 0) + cnt

    def count(self, n, fac):
        # 试除并记录幂次
        cnt = 1
        n //= fac
        while n % fac == 0:
            cnt += 1
            n //= fac
        return n, cnt

    def main(self, n, gain):
        if n > 7e5:
            # 米勒罗宾判素
            if miller_rabin(n):
                self.add(n, gain)
            else:
                # pollard rho 求解因数
                fac = pollard_rho(n)
                # 求解幂次
                n, cnt = self.count(n, fac)
                # 递归求解因数的因数
                self.main(fac, gain=cnt * gain)
                # 递归求解剩余部分
                if n > 1: self.main(n, gain=gain)
        # 试除法求解
        else:
            self.try_divide(n, gain=gain)

    def try_divide(self, n, gain=1):
        ''' 试除法分解'''
        i, bound = 2, isqrt(n)
```

```python
        while i <= bound:
            if n % i == 0:
                # 计数 + 整除
                n, cnt = self.count(n, i)
                # 记录幂次, 更新边界
                self.add(i, cnt * gain)
                bound = isqrt(n)
            i += 1
        if n > 1: self.add(n, gain)


def is_prime(m:int) -> bool:
    return miller_rabin(m)

n = int(input())
print(f"{n} -> {prime_factor(n)}")
```

1.1.2 筛法

能够 $O(1)$ 计算素数 p 处值的积性函数均可使用**欧拉筛**在 $O(n)$ 内预处理。

若 $f(x)$ 和 $g(x)$ 均为积性函数，则以下函数也是积性函数：

$$f(x^p),\quad f^p(x),\quad f(x)g(x),\quad \sum_{d|x} f(d)g\left(\frac{x}{d}\right)$$

欧拉函数前缀和（大数）

★ phi-presum-1e10.cpp

```cpp
#include <iostream>
#include <vector>
#include <map>

using int64 = long long;
constexpr int lim = 20'000'001;
constexpr int mod = 1e9l + 7;

int phi[lim + 10];
std::vector<int> prime;
bool not_prime[lim + 10];
int64 sumf[lim + 10], summ[lim + 10];

void init() {
    phi[1] = 1;
    for (int i = 2; i <= lim; i++) {
        if (not not_prime[i]) {
            prime.push_back(i);
            phi[i] = i - 1;
        }
        for (int p : prime) {
            if (i * p > lim) break;
            not_prime[i * p] = true;
            if (i % p == 0) {
                phi[i * p] = phi[i] * p;
                break;
            } else {
                phi[i * p] = phi[i] * (p - 1);
            }
        }
    }
    for (int i = 1; i <= lim; i++) {
        sumf[i] = sumf[i - 1] + phi[i];
    }
}

std::map<int64, int64> Sf, Sm;
int64 calcf(int64 l) {
    if (l <= lim) return sumf[l];
    if (Sf.find(l) != Sf.end()) return Sf[l];
    int64 SS = 1ll * l * (l + 1) / 2;
```

```cpp
    if (l & 1) SS = ((l + 1) / 2) % mod * l % mod;
    else SS = (l / 2) % mod * (l + 1) % mod;
    for (int64 i = 2, r; i <= l; i = r + 1) {
        r = l / (l / i);
        SS -= (r - i + 1) % mod * calcf(l / i) % mod;
        SS = (SS + mod) % mod;
    }
    return Sf[l] = SS;
}

int main() {
    init();
    int64 n;
    std::cin >> n;

    int ans = 0;
    for (int64 l = 1, r; l <= n; l = r + 1) {
        r = n / (n / l);
        int64 f = n / l;
        int len = (r - l + 1) % mod;
        int pref = calcf(f) % mod;
        int res = 1ll * f % mod * len % mod * (2 * pref - 1) % mod;
        ans = (ans + res) % mod;
    }

    std::cout << (mod + ans) % mod << '\n';
}
```

1.1.3 莫比乌斯反演 mobius inversion

useful conclusion: $n = \sum_{d|n} \varphi(d)$

example: $\sum_{i=a}^{b} \sum_{j=c}^{d} [\gcd(i,j) = k]$

⋆ Mobius.cpp

```cpp
// n = \sum (d : n % d == 0) phi(d)
int solve(int n, int m) {
    int res = 0; // muSum is the presum of mu
    for (int i = 1, j; i <= std::min(n, m); i = j + 1) {
        j = std::min(n / (n / i), m / (m / i));
        res += (muSum[j] - muSum[i - 1]) * (n / i) * (m / i);
    }
    return res;
}
int query(int a, int b, int c, int d, int k) {
    --a, --c;
    return + solve(b / k, d / k) - solve(b / k, c / k)
           - solve(a / k, d / k) + solve(a / k, c / k);
}
```

1.2 同余

1.2.1 exgcd

⋆ exgcd.cpp

```cpp
int exgcd(int a, int b, int &x, int &y) {
    int x1 = 1, x2 = 0, x3 = 0, x4 = 1;
    while (b != 0) {
        int c = a / b;
        std::tie(x1, x2, x3, x4, a, b) =
            std::make_tuple(x3, x4, x1 - x3 * c, x2 - x4 * c, b, a - b * c);
    }
    x = x1, y = x2;
    return a;
}
```

⋆ another-exgcd.cpp

```cpp
#include <iostream>

template <typename T>
T exgcd(T a, T b, T &x, T &y) {
    // a * x + b * y == return_value
    if (b == 0) {
        x = 1; y = 0;
        return a;
    } else {
        T t = exgcd(b, a % b, y, x);
        // b * y + (a - (a / b) * b) * x == t
        // a * x + b * (y - (a / b) * x) == t
        y -= (a / b) * x;
        return t;
    }
}

template <typename T>
void get_min_pos(T a, T b, T &x, T &y) {
    T target = x % b;
    if (target <= 0) target += b; // making target POSITIVE
    y -= (target - x) / b * a;
    x = target;
}

int main() {
    long long a, b, c;
    std::cin >> a >> b >> c;
    long long x, y;
    long long g = exgcd(a, b, x, y);
    if (c % g != 0) {
        puts("-1\n");
    } else {
        a /= g; b /= g; c /= g;
        x *= c; y *= c;

        get_min_pos(a, b, x, y);
        long long o1 = x, o4 = y;
        get_min_pos(b, a, y, x);
        long long o2 = y, o3 = x;

        if (x <= 0) {
            // no pos sol: min pos x, min pos y
            printf("%lld %lld\n", o1, o2);
        } else {
            // all pos: pos sol count, min x, min y, max x, max y
            long long o0 = (o3 - o1) / b + 1;
            printf("%lld %lld %lld, %lld, %lld\n", o0, o1, o2, o3, o4);
        }
    }
}
```

1.2.2 exCRT

⋆ exCRT.cpp

```cpp
template <typename T>
T exgcd(T a, T b, T& x, T& y) {
    // ax + by == g
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    } else {
        // (a - (a / b) * b) x + by == g
        // ax + b(y + (a / b) x) == g
        T t = exgcd(b, a % b, y, x);
        y -= (a / b) * x;
        return t;
    }
```

```cpp
}

// [modulo, remainder] in num, T: int{B}, Y: int{2B}
template <typename T, typename Y>
T excrt(const std::vector<std::pair<T, T>> &num) {
    Y A = 1;
    T B = 0;
    for (auto [b, a] : num) {
        // t = Ax + B = ay + b
        // Ax - ay = b - B
        T c = b - B, x, y;
        T g = exgcd<T>(A, a, x, y);
        if (c % g != 0) return -1; // no solution.
        x *= (c / g), y *= (c / g);
        B = (B + A * x) % (A * a / g);
        A *= a / g;
        if (B < 0) B += A;
    }
    return B;
}
```

1.2.3 离散对数 (ex)bsgs

⋆ bsgs-basic.cpp

```cpp
#include <unordered_map>
#include <iostream>
#include <cassert>
#include <vector>
#include <cmath>

using int64 = long long;

int64 mod(int64 x, int64 p) {
    if (x >= p) x -= p;
    if (x <  0) x += p;
    return x;
}

int64 mod_add(int64 a, int64 b, int64 p) {
    return mod(a + b, p);
}

int64 mod_mul(int64 a, int64 b, int64 p) {
    return mod((__int128_t) a * b % p, p); // 64-bit only
    int64 res = 0;
    for (a %= p, b %= p; b; b /= 2, a = mod_add(a, a, p)) {
        if (b & 1) {
            res = mod_add(res, a, p);
        }
    }
    return mod(res, p);
}

int64 mod_pow(int64 a, int64 b, int64 p) {
    int64 res = 1;
    for (a %= p; b; b /= 2, a = mod_mul(a, a, p)) {
        if (b & 1) {
            res = mod_mul(res, a, p);
        }
    }
    return res;
}

std::vector<int64> get_prime_factor(int64 p) {
    std::vector<int64> res;
    for (int64 k = 2; k * k <= p; k++) {
        if (p % k == 0) {
            res.push_back(k);
            while (p % k == 0) {
                p /= k;
            }
```

```
        }
    }
    if (p > 1) res.push_back(p);
    return res;
}

int64 find_primitive_root(int64 p) {
    auto pf = get_prime_factor(p - 1);
    for (int i = 1; i < p; i++) {
        bool valid = true;
        for (int64 f : pf) {
            if (mod_pow(i, (p - 1) / f, p) == 1) {
                valid = false;
                break;
            }
        }
        if (valid) return i;
    }
    assert(false);
}

template <typename T>
T exgcd(T a, T b, T &x, T &y) {
    if (b == 0) {
        x = 1; y = 0;
        return a;
    } else {
        T d = exgcd(b, a % b, y, x);
        // (a - (a / b) * b)x + by = d
        // ax + b(y - (a / b)x) = d
        y -= a / b * x;
        return d;
    }
}

//: accepted when T <= 200, p < 1e12, p is prime
int main() {
    int T;
    int64 p;
    std::cin >> T >> p;
    // w * T = p / w ~> w*w = p/T
    int64 w = std::max((int64) std::sqrt(p / 25), 1ll);
    int64 g = find_primitive_root(p);
    int64 invg = mod_pow(g, p - 2, p);
    std::unordered_map<int64, int64> umap;
    umap.reserve(50'000'000);
    for (int64 i = 0, val = 1, mul = mod_pow(g, w, p); i < p - 1;
            i += w, val = mod_mul(val, mul, p)) {
        umap[val] = i;
    }
    auto dlog = [&] (int64 num) -> int64 {
        for (int64 j = 0; j < w; j++, num = mod_mul(num, invg, p)) {
            // x * pow(g, j) == num -> x = num / pow(g, j)
            if (umap.count(num)) {
                return umap[num] + j;
            }
        }
        assert(false);
    };
    for (int cs = 1; cs <= T; cs ++) {
        // find minimal non-negative `x` s.t. `a^x \equiv b \pmod p`
        int64 a, b;
        std::cin >> a >> b;
        int64 ap = dlog(a), bp = dlog(b);
        // ap * x == bp (mod p - 1)
        int64 mp = p - 1, x, y;
        int64 d = exgcd(ap, mp, x, y);
        if (bp % d != 0) {
            std::cout << "-1" << std::endl;
        } else {
            ap /= d;
            mp /= d;
            bp /= d;
```

```
            x = mod_mul(x, bp, mp);
            std::cout << x << std::endl;
        }
    }
}
```

⋆ exbsgs.cpp

```
// a^x = b (mod p) => min({x}); min(\emptyset) := -inf( < 0 )

int bsgs(int a, int p, int b) {
    // if (b == 1 % p) return 0;

    int T = __builtin_sqrt(p) + 1;
    int N = p / T + 1;
    int cur = b, c = 1;
    unordered_map<int, int> mp;
    for (int i = 0; i < T; ++i) {
        mp[cur] = i;
        cur = 1LL * cur * a % p;
        c = 1LL * c * a % p;
    }

    for (int i = 1, k = c; i <= N; ++i, k = 1LL * k * c % p)
        if (mp.count(k))
            return i * T - mp[k];

    return -100;
}

int exbsgs(int a, int p, int b) {
    int d = __gcd(a, p);

    if (b == 1 % p)
        return 0;
    if (d == 1)
        return bsgs(a, p, b);
    if (b % d != 0)
        return -100;

    p /= d, b /= d;
    int invc, y;
    exgcd(a / d, p, invc, y);
    b = (1LL * b * invc % p + p) % p;

    return exbsgs(a, p, b) + 1;
}
```

1.3 多项式 Polynomial

1.3.1 FFT/NTT

⋆ FFT.cpp

```
#include <iostream>
#include <complex>
#include <cmath>

using Complex = complex<double>;

const double pi = acos(-1.0);

inline void dft(Complex a[], int len, int f) {
    for (int i = 0, k = 0; i < len; ++i) {
        if (i < k) std::swap(a[i], a[k]);
        for (int j = len >> 1; (k ^= j) < j; j >>= 1);
    }
    for (int h = 1; h < len; h *= 2) {
        Complex wn(cos(pi / h), f * sin(pi / h));
        for (int L = 0; L < len; L += h * 2) {
```

```
            Complex t(1.0, 0.0);
            for (int k = L; k < L + h; k++, t *= wn) {
                Complex t1 = a[k], t2 = t * a[k + h];
                a[k] = t1 + t2, a[k + h] = t1 - t2;
            }
        }
    }
    if (f == -1) {
        for (int i = 0; i < len; ++i) {
            a[i] /= len;
        }
    }
}
```

⋆ NTT.cpp

```
const int P = 998244353;

void ntt(int a[], int len, int f) {
    for (int i = 0, k = 0; i < len; i++) {
        if (i < k) std::swap(a[i], a[k]);
        for (int j = len >> 1; (k ^= j) < j; j >>= 1) {}
    }
    for (int h = 1; h < len; h *= 2) {
        const int g = 3;
        int wn = mod_pow(g, (P - 1) / (2 * h));
        for (int L = 0; L < len; L += 2 * h) {
            for (int w = 1, k = L; k < L + h; k++, w = 1ll * w * wn % P) {
                int x = a[k], y = 1ll * a[k + h] * w % P;
                a[k] = x + y;
                a[k + h] = x - y;
                if (a[k] >= P) a[k] -= P;
                if (a[k + h] < 0) a[k + h] += P;
            }
        }
    }
    if (f == -1) {
        std::reverse(a + 1, a + len);
        for (int i = 0, val = inv_of(len); i < len; i++) {
            a[i] = 1ll * a[i] * val % P;
        }
    }
}
```

1.4 组合数学 Conbinatorics

⋆ fibonacci.cpp

```
// fast doubling method
template <typename T>
std::pair<T, T> fib(int n) {
    if (n == 0) return { 0, 1 };
    auto [c0, d0] = fib<T>(n >> 1);
    T c = c0 * (2 * d0 - c0);
    T d = c0 * c0 + d0 * d0;
    if (n & 1) return { d, c + d };
    else return { c, d };
}
```

1.4.1 Polynomial (MOD 998244353)

⋆ Poly.cpp

```
#include <functional>
#include <algorithm>
#include <iostream>
#include <vector>

const int P = 998'244'353;
```

```cpp
const int N = 1e6 + 10;

int inv[N];

__attribute__((constructor))
void init_inv() {
    inv[1] = 1;
    for (int i = 2; i < N; i++) {
        inv[i] = P - 1ll * P / i * inv[P % i] % P;
    }
}

int add(int x, int y, int p = P) {
    x += y;
    if (x >= p) x -= p;
    if (x <  0) x += p;
    return x;
}

template <typename T>
int mod_pow(int a, T b, int p = P) {
    int res = 1;
    for (; b; b /= 2, a = 1ll * a * a % p) {
        if (b & 1) {
            res = 1ll * res * a % p;
        }
    }
    return res;
}

int inv_of(int a, int p = P) {
    if (a < N) {
        return inv[a];
    } else {
        return mod_pow(a, p - 2, p);
    }
}

const int primitiveRoot = 3;
std::vector<int> Roots { 0, 1 };

struct poly : public std::vector<int> {

    poly() : std::vector<int>() {}

    poly(int n) : std::vector<int>(n) {}

    poly(const std::vector<int> &a) : std::vector<int>(a) {}

    poly(const std::initializer_list<int> &a) : std::vector<int>(a) {}

    template<class InputIt, class = std::_RequireInputIter<InputIt>>
    poly(InputIt first, InputIt last) : std::vector<int>(first, last) {}

    poly shift(int k) const {
        if (k >= 0) {
            poly f(*this);
            f.insert(f.begin(), k, 0);
            return f;
        } else if (this->size() <= -k) {
            return poly{};
        } else {
            return poly(this->begin() + (-k), this->end());
        }
    }

    poly trunc(int k) const {
        poly f(*this);
        f.resize(k);
        return f;
    }

    poly dft(int len) const {
```

```
        while (len > Roots.size()) {
            int h = Roots.size();
            int wn = mod_pow(primitiveRoot, (P - 1) / (2 * h));
            for (int i = 0, v = 1; i < h; i++, v = 1ll * v * wn % P) {
                Roots.push_back(v);
            }
        }
        poly a = this->trunc(len);
        for (int i = 0, k = 0; i < len; i++) {
            if (i < k) std::swap(a[i], a[k]);
            for (int j = len >> 1; (k ^= j) < j; j >>= 1) {}
        }
        for (int h = 1; h < len; h *= 2) {
            for (int L = 0; L < len; L += 2 * h) {
                for (int t = L; t < L + h; t++) {
                    int x = a[t], y = 1ll * a[t + h] * Roots[h + t - L] % P;
                    a[t] = add(x, y);
                    a[t + h] = add(x, -y);
                }
            }
        }
        return a;
    }

    poly idft(int len) const {
        poly a = this->dft(len);
        std::reverse(a.begin() + 1, a.end());
        // int invlen = inv_of(len);
        int invlen = P - (P - 1) / len;
        for (int i = 0; i < len; i++) {
            a[i] = 1ll * a[i] * invlen % P;
        }
        return a;
    }

    poly mul(const poly &g0) const {
        int len = 1, n = this->size() + g0.size() - 2;
        while (len <= n) {
            len *= 2;
        }
        poly f = this->dft(len);
        poly g = g0.dft(len);
        for (int i = 0; i < len; i++) {
            f[i] = 1ll * f[i] * g[i] % P;
        }
        return f.idft(len).trunc(n + 1);
    }

    friend poly operator* (poly f, int x) {
        for (int &val : f) {
            val = 1ll * val * x % P;
        }
        return f;
    }

    friend poly operator* (int x, const poly &f) {
        return f * x;
    }

    friend poly operator* (const poly &f, const poly &g) {
        return f.mul(g);
    }

    friend poly operator+ (const poly &f, const poly& g) {
        poly h(std::max(f.size(), g.size()));
        for (int i = 0; i < f.size(); i++) {
            h[i] = add(h[i], +f[i]);
        }
        for (int i = 0; i < g.size(); i++) {
            h[i] = add(h[i], +g[i]);
        }
        return h;
    }
```

```cpp

    friend poly operator- (const poly &f, const poly& g) {
        poly h(std::max(f.size(), g.size()));
        for (int i = 0; i < f.size(); i++) {
            h[i] = add(h[i], +f[i]);
        }
        for (int i = 0; i < g.size(); i++) {
            h[i] = add(h[i], -g[i]);
        }
        return h;
    }

    poly derive() const {
        poly f(this->size() - 1);
        for (int i = 1; i < this->size(); i++) {
            f[i - 1] = 1ll * i * (*this)[i] % P;
        }
        return f;
    }

    poly integral() {
        poly f(this->size() + 1);
        for (int i = 1; i < f.size(); i++) {
            f[i] = 1ll * (*this)[i - 1] * inv_of(i) % P;
        }
        return f;
    }

    poly inv(int m) const {
        poly f {inv_of((*this)[0])};
        int k = 1;
        while (k < m) {
            k *= 2;
            f = (f * (poly{2} - f * this->trunc(k))).trunc(k);
        }
        return f;
    }

    // [q, r] = f / g  <==>  f = q · g + r
    std::pair<poly, poly> operator/ (const poly &g) const {
        int n = this->size() - 1, m = g.size() - 1;
        poly fr(this->rbegin(), this->rend());
        poly gr(g.rbegin(), g.rend());
        poly qr = (fr * gr.inv(n - m + 1)).trunc(n - m + 1);
        poly q(qr.rbegin(), qr.rend());
        poly r = ((*this) - g * q).trunc(m);
        return std::make_pair(q, r);
    }

    poly log(int m) const {
        return (this->derive() * this->inv(m)).integral().trunc(m);
    }

    poly exp(int m) const {
        poly f{1};
        int k = 1;
        while (k < m) {
            k *= 2;
            f = (f * (poly{1} - f.log(k) + this->trunc(k))).trunc(k);
        }
        return f.trunc(m);
    }

    poly pow(int k, int m) const {
        int i = 0;
        while (i < this->size() and (*this)[i] == 0) {
            i++;
        }
        if (i == this->size() or 1ll * i * k >= m) {
            return poly(m);
        }
        int val = (*this)[i];
        auto f = this->shift(-i) * inv_of(val);
```

```cpp
        return (f.log(m - i * k) * k).exp(m - i * k).shift(i * k) * mod_pow(val, k);
    }

    poly pow_bigint(const std::string &k, int m) const {
        int i = 0;
        while (i < this->size() and (*this)[i] == 0) {
            i++;
        }
        if (i == this->size() or (i > 0 and (k.length() >= 8 or 1ll * i * std::stoi(k) >= m))) {
            return poly(m);
        }
        int k1 = 0, k2 = 0;
        for (char c : k) {
            k1 = (10ll * k1 + c - '0') % P;
            k2 = (10ll * k2 + c - '0') % (P - 1);
        }
        int val = (*this)[i];
        return ((this->shift(-i) * inv_of(val)).log(m - i * k1) * k1)
            .exp(m - i * k1).shift(i * k1) * mod_pow(val, k2);
    }

    poly sqrt(int m) const {
        poly f{1};
        int k = 1;
        while (k < m) {
            k *= 2;
            f = (f + (trunc(k) * f.inv(k)).trunc(k)) * inv_of(2);
        }
        return f.trunc(m);
    }

    poly mulT(const poly &g) const {
        if (g.size() == 0) {
            return poly{};
        }
        int n = g.size();
        return ((*this) * poly(g.rbegin(), g.rend())).shift(-(n - 1));
    }

    std::vector<int> eval(std::vector<int> x) const {
        if (this->size() == 0) {
            return std::vector<int>(x.size(), 0);
        }
        const int n = std::max(x.size(), this->size());
        std::vector<poly> q(4 * n);
        std::vector<int> ans(x.size());
        x.resize(n);
        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                q[p] = poly{1, -x[l]};
            } else {
                int m = (l + r) / 2;
                build(2 * p, l, m);
                build(2 * p + 1, m, r);
                q[p] = q[2 * p] * q[2 * p + 1];
            }
        };
        build(1, 0, n);
        std::function<void(int, int, int, const poly &)>
        work = [&](int p, int l, int r, const poly &num) -> void {
            if (r - l == 1) {
                if (l < int(ans.size())) {
                    ans[l] = num[0];
                }
            } else {
                int m = (l + r) / 2;
                work(2 * p, l, m, num.mulT(q[2 * p + 1]).trunc(m - l));
                work(2 * p + 1, m, r, num.mulT(q[2 * p]).trunc(r - m));
            }
        };
        work(1, 0, n, mulT(q[1].inv(n)));
        return ans;
    }
```

```
};
```

1.5 MAGIC PRIMES

$r\ 2^k+1$	r	k	g
3	1	1	2
5	1	2	2
17	1	4	3
97	3	5	5
193	3	6	5
257	1	8	3
7681	15	9	17
12289	3	12	11
40961	5	13	3
65537	1	16	3
786433	3	18	10
5767169	11	19	3
7340033	7	20	3
23068673	11	21	3
104857601	25	22	3
167772161	5	25	3
469762049	7	26	3
998244353	119	23	3
1004535809	479	21	3
2013265921	15	27	31
2281701377	17	27	3
3221225473	3	30	5
75161927681	35	31	3
77309411329	9	33	7
206158430209	3	36	22
2061584302081	15	37	7
2748779069441	5	39	3
6597069766657	3	41	5
39582418599937	9	42	5
79164837199873	9	43	5
263882790666241	15	44	7
1231453023109121	35	45	3
1337006139375617	19	46	3
3799912185593857	27	47	5
4222124650659841	15	48	19
7881299347898369	7	50	6
31525197391593473	7	52	3
180143985094819841	5	55	6
1945555039024054273	27	56	5
4179340454199820289	29	57	3

1.6 Notes

I. 错排公式：$D(n) = (n-1)(D(n-1) + D(n-2))$

II. 牛顿迭代 **(poly)**：$g(f(x)) \equiv 0 \pmod{x^n} \Rightarrow f(x) \equiv f_0(x) - \dfrac{g(f_0(x))}{g'(f_0(x))} \pmod{x^n}$

III. 卡特兰数与路径计数

1. 第 n 个卡特兰数 $H_n = \frac{1}{n+1}\binom{2n}{n} = \frac{H_{n-1}(4n-2)}{n+1}$.

2. 经典递推式 $H_n = \begin{cases} \sum_{i=1}^{n} H_{i-1}H_{n-i} & n \geqslant 2 \\ 1 & n \in \{0,1\} \end{cases}$

3. 圆上 $2n$ 个点成对连接（n 个匹配）不相交的方案数为 H_n.

4. 从 $(0,0)$ 到 (n,n) 的不穿过直线 $y=x$ 的非降路径数为 $\frac{2}{n+1}\binom{2n}{n}$.

5. 从 $(0,0)$ 到 (n,n) 的除起点与终点外均不接触直线 $y=x$ 的非降路径数为 $2\binom{2n-2}{n-1} - 2\binom{2n-2}{n}$.

6. 从 $(0,0)$ 到 (n,n) 的不穿过直线 $y=x-m$ $(m \geqslant 1)$ 的非降路径数为 $\binom{2n}{n} - \binom{2n}{n-m-1}$.

7. 从 $(0,0)$ 到 (n,m) $(n>m)$ 的不穿过直线 $y=x$ 的非降路径数为 $\binom{n+m}{n} - \binom{n+m}{n+1}$.

8. 生成函数 $H(x) = \sum_{k=1}^{\infty} A_k x^k = \frac{1-\sqrt{1-4x}}{2x}$.

2 String 字符串

2.1 String-Match 字符串匹配

2.1.1 KMP

⋆ KMP.cpp

```cpp
// optimize = false -> next = \pi
std::vector<int> get_next(const std::string &t, bool optimize) {
    std::vector<int> next(t.size());
    int i = 0, j = -1;
    next[i] = j;
    while (t[i] != '\0') {
        if (j == -1 || t[i] == t[j]) {
            i++, j++;
            if (optimize && t[i] == t[j]) next[i] = next[j];
            else next[i] = j;
        }
        else j = next[j];
    }
    return next;
}
std::vector<int> kmp(const std::vector<int> &next, const std::string &s, const std::string &t) {
    std::vector<int> pos;
    int i = 0, j = 0;
    while (s[i] != '\0') {
        if (j == -1 || s[i] == t[j]) i++, j++;
        else j = next[j];
        if (j > -1 && t[j] == '\0') pos.push_back(i - j + 1), j = next[j];
    }
    return pos;
}
```

2.1.2 Z function

$$z_i = \max_j s.\mathbf{substr}(0,j) = s.\mathbf{substr}(i,j)$$

⋆ Z-Function.cpp

```cpp
std::vector<int> z_func(const std::string &s) {
    int n = s.size();
    std::vector<int> z(n);
    for (int i = 1, l = 0, r = 1; i < n; i++) {
        if (i < r and z[i - l] < r - i) {
            z[i] = z[i - l];
        } else {
            z[i] = std::max(0, r - i);
            while (z[i] + i < n and s[z[i]] == s[i + z[i]]) {
                z[i] ++;
            }
        }
        if (r < i + z[i]) {
            l = i; r = i + z[i];
        }
    }
    return z;
}
```

2.1.3 AC 自动机

⋆ AC-Automaton.cpp

```cpp
#include <iostream>
#include <queue>

const int maxnode = 1e6 + 7;
```

```cpp
const int charset = 26;
const char base = 'a';
int ac[maxnode][charset], tot, acc[maxnode], fail[maxnode];
inline int getid(int ch) { return int(ch - base); }

void insert(const std::string &p) {
    int u = 0;
    for (const auto &ch : p) {
        int id = getid(ch);
        if (!ac[u][id]) ac[u][id] = ++tot;
        u = ac[u][id];
    }
    ++acc[u];
}

// call getfail() after all `insert()` calls.
void getfail() {
    std::queue<int> que;
    for (int i = 0; i < charset; ++i) {
        if (ac[0][i]) que.emplace(ac[0][i]);
    }
    while (!que.empty()) {
        int u = que.front(); que.pop();
        for (int i = 0; i < charset; ++i) {
            if (ac[u][i]) {
                fail[ac[u][i]] = ac[fail[u]][i];
                que.emplace(ac[u][i]);
            } else {
                ac[u][i] = ac[fail[u]][i];
            }
        }
    }
}

int match(const std::string &t) {
    int u = 0, ans = 0;
    for (const auto &ch : t) {
        int id = getid(ch);
        u = ac[u][id];
        for (int v = u; v && -1 != acc[v]; v = fail[v]) {
            ans += acc[v];
            acc[v] = -1;
        }
    }
    return ans;
}

int main() {
    std::cin.tie(0) -> sync_with_stdio(false);
    int n;
    std::cin >> n;
    while (n--) {
        std::string t;
        std::cin >> t;
        insert(t);
    }
    getfail();
    std::string s;
    std::cin >> s;
    std::cout << match(s) << "\n";
    return 0;
}
```

2.1.4 BM

⋆ Boyer–Mooer.cpp

```cpp
const int charset = 26, base = 'a', maxlen = 1e6 + 2;
int bc0[charset], *bc = bc0 - base, ss[maxlen], gs[maxlen];

void buildBC(int bc[], const std::string &p) {
    for (int j = base; j < base + charset; ++j) bc[j] = -1;
```

```
    for (int m = p.size(), j = 0; j < m; ++j) bc[int(p[j])] = j;
} // buildBC - O( s + m )

void buildSS(int ss[], const std::string &p) {
    int m = p.size();
    ss[m - 1] = m;
    for (int l = m - 1, r = m - 1,j = l - 1; j >= 0; --j)
        if (l < j && ss[m - r + j - 1] <= j - l)
            ss[j] = ss[m - r + j - 1];
        else {
            r = j, l = min(l, r);
            while (0 <= l && p[l] == p[m - r + l - 1]) --l;
            ss[j] = r - l;
        }
} // buildSS

void buildGS(int gs[], const std::string &p) {
    buildSS(ss, p);
    int m = p.size();
    for (int j = 0; j < m; ++j) gs[j] = m;
    for (int i = 0, j = m - 1; j >= 0; --j)
        if (j + 1 == ss[j])
            while (i < m - j - 1)
                gs[i++] = m - j - 1;
    for (int j = 0; j < m - 1; ++j)
        gs[m - ss[j] - 1] = m - j - 1;
} // buildGS

int match(const std::string &p, const std::string &t) {
    buildBC(bc, p), buildGS(gs, p);
    int i = 0;
    while (t.size() >= i + p.size()) {
        int j = p.size() - 1;
        while (p[j] == t[i + j]) if (0 > --j) break;
        if (0 > j) break;
        else i += max(gs[j], j - bc[int(t[i + j])]);
    }
    return i;
} // match - BM version
```

2.1.5 后缀数组

⋆ SA.cpp

```
int count[N], sa[N];

template <typename T>
void radix_sort(const T &str, int old_rank[], int *rank, int n, int m) {
    std::fill(count, count + m + 1, 0);
    for (int i = 0; i < n; i++) {
        count[str[old_rank[i]]] ++;
    }
    for (int i = 1; i <= m; i++) {
        count[i] += count[i - 1];
    }
    for (int i = n - 1; i >= 0; i--) {
        rank[-- count[str[old_rank[i]]]] = old_rank[i];
    }
}

void suffix_array(const std::string &str, int *sa, int n, int m) {
    static int rank[N], a[N], b[N];
    std::iota(rank, rank + n, 0);
    radix_sort(str, rank, sa, n, m);

    rank[sa[0]] = 0;
    for (int i = 1; i < n; i++) {
        rank[sa[i]] = rank[sa[i - 1]] + (str[sa[i - 1]] != str[sa[i]]);
    }

    for (int h = 1; h < n; h *= 2) {
        std::iota(sa, sa + n, 0);
```

```cpp
        for (int i = 0; i < n; i++) {
            a[i] = rank[i] + 1;
            b[i] = i + h >= n ? 0 : rank[i + h] + 1;
        }
        radix_sort(b, sa, rank, n, n);
        radix_sort(a, rank, sa, n, n);

        rank[sa[0]] = 0;
        for (int i = 1; i < n; i++) {
            rank[sa[i]] = rank[sa[i - 1]] +
                (a[sa[i - 1]] != a[sa[i]] || b[sa[i - 1]] != b[sa[i]]);
        }
    }
}

int main() {
    std::string str;
    suffix_array(str, sa, str.size(), 128);
    for (int i = 0; i < str.size(); i++) {
        std::cout << 1 + sa[i] << " \n"[i + 1 == str.size()];
    }
}
```

2.1.6 后缀自动机

⋆ SAM.cpp

```cpp
#include <iostream>

constexpr int N = 1'000'000 + 10;
constexpr int CHARSET = 26;

struct sam_node {
    int len, pa, size;
    int next[CHARSET];
    sam_node(int l = 0, int p = 0, int s = 1) : len(l), pa(p), size(s), next{} {}
} sam[N * 2];

int tot, last;
void sam_init() {
    tot = 0;
    sam[last = ++ tot] = sam_node(0, 0, 0);
}

void extend(int c) {
    int cur = ++ tot;
    sam[cur] = sam_node(sam[last].len + 1);

    int p = last;
    while (p and not sam[p].next[c]) {
        sam[p].next[c] = cur;
        p = sam[p].pa;
    }

    if (p == 0) {
        sam[cur].pa = 1;
    } else {
        int q = sam[p].next[c];
        if (sam[p].len + 1 == sam[q].len) {
            sam[cur].pa = q;
        } else {
            int clone = ++ tot;
            sam[clone] = sam[q];
            sam[clone].len = sam[p].len + 1;
            sam[clone].size = 0;
            while (p and sam[p].next[c] == q) {
                sam[p].next[c] = clone;
                p = sam[p].pa;
            }
            sam[q].pa = sam[cur].pa = clone;
        }
    }
```

```cpp
    last = cur;
}

int count[N], rank[N * 2];
int main() {
    std::string s;
    std::cin >> s;
    int n = s.size();
    sam_init();
    for (char c : s) {
        extend(c - 'a');
    }

    for (int i = 1; i <= tot; i++) {
        count[sam[i].len] ++;
    }
    for (int i = 1; i <= n; i++) {
        count[i] += count[i - 1];
    }
    for (int i = 1; i <= tot; i++) {
        rank[count[sam[i].len] --] = i;
    }

    long long ans = 0;
    for (int i = tot; i >= 1; i--) {
        int x = rank[i], p = sam[x].pa;
        if (p) sam[p].size += sam[x].size;
        if (sam[x].size > 1) {
            ans = std::max(ans, 1LL * sam[x].size * sam[x].len);
        }
    }

    // S 的所有出现次数不为 1 的子串的出现次数乘上该子串长度的最大值
    std::cout << ans << '\n';

    return 0;
}
```

还有广义后缀自动机（多模串）

⋆ extsam.cpp

```cpp
#include <algorithm>
#include <iostream>

constexpr int N = 2'000'000 + 5;
constexpr int CHARSET = 26;

struct suffix_automaton {
    int tot, link[N], maxlen[N], trans[N][CHARSET];
    suffix_automaton() { tot = 1; }
    int insert(int ch, int last) {
        if (trans[last][ch]) {
            int p = last, x = trans[p][ch];
            if (maxlen[p] + 1 == maxlen[x]){
                return x;
            } else {
                int y = ++ tot;
                maxlen[y] = maxlen[p] + 1;
                for (int i = 0; i < CHARSET; i++) {
                    trans[y][i] = trans[x][i];
                }
                while (p and trans[p][ch] == x) {
                    trans[p][ch] = y;
                    p = link[p];
                }
                link[y] = link[x];
                link[x] = y;
                return y;
            }
        }
        int z = ++ tot, p = last;
```

```cpp
        maxlen[z] = maxlen[last] + 1;
        while (p and not trans[p][ch]) {
            trans[p][ch] = z;
            p = link[p];
        }
        if (!p) {
            link[z] = 1;
        } else {
            int x = trans[p][ch];
            if (maxlen[p] + 1 == maxlen[x]) {
                link[z] = x;
            } else {
                int y = ++ tot;
                maxlen[y] = maxlen[p] + 1;
                for (int i = 0; i < CHARSET; i++) {
                    trans[y][i] = trans[x][i];
                }
                while (p and trans[p][ch] == x) {
                    trans[p][ch] = y;
                    p = link[p];
                }
                link[y] = link[x];
                link[z] = link[x] = y;
            }
        }
        return z;
    }
    long long sakura() {
        // 本质不同的子串个数.
        long long ans = 0;
        for (int i = 2; i <= tot; i++) {
            ans += maxlen[i] - maxlen[link[i]];
        }
        return ans;
    }
} sam;

int main() {
    std::cin.tie(nullptr)->sync_with_stdio(false);
    int n;
    std::cin >> n;
    for (int i = 0; i < n; i++) {
        std::string s;
        std::cin >> s;
        int last = 1;
        for (char c : s) {
            last = sam.insert(c - 'a', last);
        }
    }
    std::cout << sam.sakura() << '\n';
    std::cout << sam.tot << '\n';
}
```

2.2 misc

2.2.1 回文串

⋆ Manacher.cpp

```cpp
#include <iostream>
#include <cstring>

const int N = 2 * 1.1e7 + 233;
int p[N];

int main() {
    std::cin.tie(nullptr) -> sync_with_stdio(false);

    std::string s0;
    std::cin >> s0;

```

```cpp
    std::string s = "$#";
    for (char c : s0) {
        s += c;
        s += '#';
    }
    s += '%';

    int n = s.size() - 1;

    int mid = 0, mr = 0, ans = 0;
    for (int i = 1; i < n; i++) {
        if (i <= mr) p[i] = std::min(p[2 * mid - i], mr - i + 1);
        else p[i] = 1;
        while (s[i - p[i]] == s[i + p[i]]) p[i] ++;
        if (i + p[i] > mr) mr = i + p[i] - 1, mid = i;
        ans = std::max(ans, p[i]);
    }
    std::cout << ans - 1 << '\n';

    return 0;
}
```

Palindromic Tree 回文树（回文自动机）

⋆ PAM.cpp

```cpp
#include <iostream>

constexpr int N = 2'000'000 + 10;
constexpr int CHARSET = 26;

struct pam_node {
    int len, fail, dep;
    int next[CHARSET];
};

struct pam {
    pam_node tr[N];
    int root[2], n, tot, last;
    char s[N] = "$";

    pam() : root{0, 1}, n(0), tot(1), last(0), s("$") {
        tr[root[0]].len = 0;
        tr[root[1]].len = -1;
        tr[root[0]].fail = root[1];
    }

    int get_fail(int x) {
        while (s[n - tr[x].len - 1] != s[n]) {
            x = tr[x].fail;
        }
        return x;
    }

    void insert(int c) {
        s[++n] = 'a' + c;
        int p = get_fail(last);
        if (not tr[p].next[c]) {
            int x = ++ tot;
            tr[x].len = tr[p].len + 2;
            tr[x].fail = tr[get_fail(tr[p].fail)].next[c];
            tr[x].dep = tr[tr[x].fail].dep + 1;
            tr[p].next[c] = x;
        }
        last = tr[p].next[c];
    }
} p;

int main() {
    std::string s;
    std::cin >> s;
    for (char c : s) {
        p.insert(c - 'a');
```

```
    }
    return 0;
}
```

2.2.2 Lyndon 分解 – Duval 算法

Lyndon 串：字典序严格小于自身所有非平凡后缀的字符串。

Lyndon 分解：$s = w_1 + \cdots + w_k$，其中 w_i 是 **Lyndon** 串且 $w_1 \geqslant \cdots \geqslant w_k$.

⋆ Lyndon.cpp

```cpp
#include <cstdio>
const int N = 5e6 + 7;
char s[N];

int main() {
    scanf("%s", s + 1);
    int i = 1, ans = 0;
    while (s[i]) {
        int j = i, k = i + 1;
        while (s[k] and s[j] <= s[k]) j = s[j] == s[k++] ? j + 1 : i;
        while (i <= j) i += k - j, ans ^= i - 1; // 所有右端点异或和
    }
    printf("%d\n", ans);
    return 0;
}
```

2.2.3 最小表示法

⋆ minimal-cyclic-shift.cpp

```cpp
#include <cstdio>
const int N = 6e5 + 7;
int n, ans, s[N];

int main() {
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) {
        scanf("%d", s + i);
        s[i + n] = s[i];
    }
    int i = 1;
    while (i <= n) {
        int j = i, k = i + 1;
        while (k <= n * 2 && s[j] <= s[k]) j = s[j] == s[k++] ? j + 1 : i;
        while (i <= j) i += k - j, ans = i <= n ? i : ans;
    }
    for (int i = 1; i <= n; i++) {
        printf("%d%c", s[ans - 1 + i], " \n"[i == n]);
    }
    return 0;
}
```

3 Data Structure 数据结构

3.1 BST （二叉）平衡树

3.1.1 伸展树 Splay

⋆ BST/Splay.cpp

```cpp
#include <iostream>
#include <algorithm>

const static int maxn = 5e5 + 10, inf = 1e9;

struct Node {
    int val, lsum, rsum, mxsum, sum, ch[2], p, sz;
    bool rev, same;
    Node () {}
    Node (int v, int p) : val(v), mxsum(v), sum(v), p(p), sz(1), rev(false), same(false) {
        ch[0] = ch[1] = 0;
        lsum = rsum = std::max(0, v);
    }
} tr[maxn];

int nodes[maxn], top, root;
void initNodes() {
    tr[0].mxsum = -inf;
    for (int i = maxn - 1; i; --i) nodes[++top] = i;
}
void delNode(int u) { nodes[++top] = u; }
int newNode(int v, int p) {
    int u = nodes[top--];
    tr[u] = Node(v, p);
    return u;
}

void pushup(int x) {
    Node & u = tr[x], & lc = tr[tr[x].ch[0]], & rc = tr[tr[x].ch[1]];
    u.sz = lc.sz + rc.sz + 1;
    u.sum = lc.sum + rc.sum + u.val;
    u.lsum = std::max(lc.lsum, lc.sum + u.val + rc.lsum);
    u.rsum = std::max(rc.rsum, rc.sum + u.val + lc.rsum);
    u.mxsum = std::max( { lc.mxsum, rc.mxsum, lc.rsum + u.val + rc.lsum } );
}

void downlz(int x, int v) {
    if (!x) return;
    Node & u = tr[x];
    u.same = true;
    u.val = v;
    u.sum = v * u.sz;
    u.lsum = u.rsum = std::max(0, u.sum);
    u.mxsum = std::max(v, u.sum);
}

void downrev(int x) {
    if (!x) return;
    tr[x].rev ^= true;
    std::swap(tr[x].ch[0], tr[x].ch[1]);
    std::swap(tr[x].lsum, tr[x].rsum);
}

void pushdown(int x) {
    if (tr[x].same) {
        downlz(tr[x].ch[0], tr[x].val);
        downlz(tr[x].ch[1], tr[x].val);
        tr[x].same = tr[x].rev = false;
    } else if (tr[x].rev) {
        downrev(tr[x].ch[0]);
        downrev(tr[x].ch[1]);
        tr[x].rev = false;
    }
```

```
}

int get(int x) {
    return tr[tr[x].p].ch[1] == x;
}

void rotate(int x) {
    int y = tr[x].p, z = tr[y].p;
    int kx = get(x), ky = get(y);
    tr[y].ch[kx] = tr[x].ch[kx ^ 1];
    if (tr[x].ch[kx ^ 1]) tr[tr[x].ch[kx ^ 1]].p = y;
    tr[x].ch[kx ^ 1] = y;
    tr[y].p = x;
    tr[x].p = z;
    if (z) tr[z].ch[ky] = x;
    pushup(y);
    pushup(x);
}

void splay(int x, int goal) {
    for (int p; p = tr[x].p, p != goal; rotate(x)) {
        if (tr[p].p != goal)
            rotate( get(x) ^ get(p) ? x : p );
    }
    if (goal == 0) root = x;
}

int k_th(int k) {
    int u = root;
    while (u) {
        pushdown(u);
        if (k <= tr[tr[u].ch[0]].sz) u = tr[u].ch[0];
        else if (k == tr[tr[u].ch[0]].sz + 1) break;
        else k -= tr[tr[u].ch[0]].sz + 1, u = tr[u].ch[1];
    }
    return u;
}

void recycle(int x) {
    if (x == 0) return;
    recycle(tr[x].ch[0]);
    recycle(tr[x].ch[1]);
    delNode(x);
}

int a[maxn];
int build(int l, int r, int p) {
    int m = (l + r) / 2;
    int u = newNode(a[m], p);
    if (l < m) tr[u].ch[0] = build(l, m, u);
    if (m + 1 < r) tr[u].ch[1] = build(m + 1, r, u);
    pushup(u);
    return u;
}

int main() {
    std::cin.tie(0) -> sync_with_stdio(false);
    initNodes();
    int n, m;
    std::cin >> n >> m;
    a[0] = a[n + 1] = -inf;
    for (int i = 1; i <= n; ++i) {
        std::cin >> a[i];
    }
    root = build(0, n + 2, 0);
    while (m--) {
        std::string op;
        std::cin >> op;
        int posi, tot, c;
        if (op == "INSERT") {
            std::cin >> posi >> tot;
            for (int i = 0; i < tot; ++i) {
                std::cin >> a[i];
```

```
            }
            int L = k_th(posi + 1), R = k_th(posi + 2);
            splay(L, 0), splay(R, L);
            tr[R].ch[0] = build(0, tot, R);
            pushup(R), pushup(L);
        } else if (op == "DELETE") {
            std::cin >> posi >> tot;
            int L = k_th(posi), R = k_th(posi + tot + 1);
            splay(L, 0), splay(R, L);
            recycle(tr[R].ch[0]);
            tr[R].ch[0] = 0;
            pushup(R), pushup(L);
        } else if (op == "MAKE-SAME") {
            std::cin >> posi >> tot >> c;
            int L = k_th(posi), R = k_th(posi + tot + 1);
            splay(L, 0), splay(R, L);
            downlz(tr[R].ch[0], c);
            pushup(R), pushup(L);
        } else if (op == "REVERSE") {
            std::cin >> posi >> tot;
            int L = k_th(posi), R = k_th(posi + tot + 1);
            splay(L, 0), splay(R, L);
            downrev(tr[R].ch[0]);
            pushup(R), pushup(L);
        } else if (op == "GET-SUM") {
            std::cin >> posi >> tot;
            int L = k_th(posi), R = k_th(posi + tot + 1);
            splay(L, 0), splay(R, L);
            std::cout << tr[tr[R].ch[0]].sum << '\n';
        } else /* op == "MAX-SUM" */ {
            std::cout << tr[root].mxsum << '\n';
        }
    }
    return 0;
}
```

⋆ BST/YetAnotherSplay.cpp

```
#include <cassert>
#include <iostream>

namespace Solution {
    const int N = 1010, MAX_NODE = N * N;
    int nodeCnt, root[N], idx, ch[MAX_NODE][2];
    typedef int array_type[MAX_NODE];
    array_type type, pa, sz, sum, stack, lazy, value;

    int newNode() {
        int x = nodeCnt ++;
        pa[x] = 0;
        ch[x][0] = ch[x][1] = 0;
        type[x] = 2;
        sz[x] = 1;
        sum[x] = value[x] = lazy[x] = 0;
        return x;
    }

    void build(int n) {
        nodeCnt = 1;
        for (int i = 1; i <= n; i++) {
            int last = root[i] = newNode();
            sz[last] = n + 2;
            for (int j = 1, now; j <= n + 1; last = now, j++) {
                now = newNode();
                pa[now] = last;
                type[now] = 1;
                ch[last][1] = now;
                sz[now] = sz[last] - 1;
            }
        }
    }

    void push_up(int x) {
```

```
        sz[x] = sz[ch[x][0]] + 1 + sz[ch[x][1]];
        sum[x] = sum[ch[x][0]] + sum[ch[x][1]] + value[x] + (sz[x] % 2 == 1 ? lazy[x] : 0);
    }

    void push_down(int x) {
        if (0 == lazy[x]) return;
        lazy[ch[x][0]] += lazy[x];
        lazy[ch[x][1]] += (sz[ch[x][0]] % 2 == 1 ? +1 : -1) * lazy[x];
        value[x] += (sz[ch[x][0]] % 2 == 0 ? +1 : -1) * lazy[x];
        if (ch[x][0]) push_up(ch[x][0]);
        if (ch[x][1]) push_up(ch[x][1]);
        lazy[x] = 0;
    }

    void rotate(int x) {
        int t = type[x], y = pa[x], z = ch[x][1 - t];
        type[x] = type[y];
        pa[x] = pa[y];
        if (type[x] != 2) ch[pa[x]][type[x]] = x;
        type[y] = 1 - t;
        pa[y] = x;
        ch[x][1 - t] = y;
        if (z) {
            type[z] = t;
            pa[z] = y;
        }
        ch[y][t] = z;
        push_up(y);
    }

    void splay(int x) {
        int top = 0;
        stack[top ++] = x;
        for (int i = x; type[i] != 2; i = pa[i]) {
            stack[top ++] = pa[i];
        }
        do {
            push_down(stack[-- top]);
        } while (top);

        while (type[x] != 2) {
            int y = pa[x];
            if (type[x] == type[y]) rotate(y);
            else rotate(x);
            if (type[x] == 2) break;
            rotate(x);
        }
        push_up(x);
    }

    int find(int x, int rank) {
        while (true) {
            push_down(x);
            if (sz[ch[x][0]] + 1 == rank) break;
            if (rank <= sz[ch[x][0]]) x = ch[x][0];
            else rank -= sz[ch[x][0]] + 1, x = ch[x][1];
        }
        return x;
    }

    void split(int &x, int &y, int a) {
        y = find(x, a + 1);
        splay(y);
        x = ch[y][0];
        type[x] = 2;
        ch[y][0] = 0;
        push_up(y);
    }

    void split3(int &x, int &y, int &z, int a, int b) {
        split(x, z, b);
        split(x, y, a - 1);
    }
```

```

    void join(int &x, int y) {
        x = find(x, sz[x]);
        splay(x);
        ch[x][1] = y;
        type[y] = 1;
        pa[y] = x;
        push_up(x);
    }

    void join3(int &x, int y, int z) {
        join(y, z);
        join(x, y);
    }

    void main(int n, int m) {
        build(n);
        while (m--) {
            int op, a, s, e;
            std::cin >> op >> a >> s >> e;
            if (op == 1) {
                int y, z;
                split3(root[a], y, z, s + 1, e + 1);
                lazy[y] ++;
                push_up(y);
                join3(root[a], y, z);
                std::cout << sum[root[a]] << '\n';
            } else if (op == 2) {
                int b, y, z, t;
                std::cin >> b;
                split3(root[a], y, z, s + 1, e + 1);
                split(root[b], t, sz[root[b]] - 1);
                join(root[a], z);
                join3(root[b], y, t);
                std::cout << sum[root[a]] << ' ' << sum[root[b]] << '\n';
            } else {
                assert(false);
            }
        }
    }
};

int main() {
    std::cin.tie(nullptr) -> sync_with_stdio(false);
    int n, m;
    while (std::cin >> n >> m) {
        Solution::main(n, m);
    }
    return 0;
}
```

3.1.2 可持续化 fhq treap

⋆ BST/PersistentTreap.cpp

```
#include <iostream>
#include <cassert>
#include <climits>

const int N = 5e5 + 10;

struct Node {
    int l, r, key, val, sz;
    Node () {}
    Node (int val) : l(0), r(0), key(std::rand()), val(val), sz(1) {}
    Node (const Node &b) : l(b.l), r(b.r), key(b.key), val(b.val), sz(b.sz) {}
} tr[N * 100];

int tot, dl, dr, tmp, root[N];

int newNode(int val) {
    tr[++tot] = Node(val);
```

```cpp
    return tot;
}
int clone(int u) {
    tr[++tot] = Node(tr[u]);
    return tot;
}

void pushup(int u) {
    tr[u].sz = tr[tr[u].l].sz + tr[tr[u].r].sz + 1;
}

void split(int u, int x, int &l, int &r) {
    if (!u) return l = r = 0, void();
    if (tr[u].val <= x)
        l = clone(u), split(tr[l].r, x, tr[l].r, r), pushup(l);
    else
        r = clone(u), split(tr[r].l, x, l, tr[r].l), pushup(r);
}

int merge(int l, int r) {
    if (!l || !r) return l | r;
    int p;
    if (tr[l].key < tr[r].key)
        p = clone(l), tr[p].r = merge(tr[p].r, r);
    else
        p = clone(r), tr[p].l = merge(l, tr[p].l);
    pushup(p);
    return p;
}

void insert(int &rt, int x) {
    split(rt, x, dl, dr);
    rt = merge(merge(dl, newNode(x)), dr);
}

void erase(int &rt, int x) {
    split(rt, x, dl, dr);
    split(dl, x - 1, dl, tmp);
    tmp = merge(tr[tmp].l, tr[tmp].r);
    rt = merge(merge(dl, tmp), dr);
}

int getrk(int &rt, int x) {
    split(rt, x - 1, dl, dr);
    int rnk = tr[dl].sz + 1;
    rt = merge(dl, dr);
    return rnk;
}

int k_th(int u, int k) {
    while (u) {
        if (k <= tr[tr[u].l].sz) u = tr[u].l;
        else if (tr[tr[u].l].sz + 1 == k) break;
        else k -= tr[tr[u].l].sz + 1, u = tr[u].r;
    }
    return tr[u].val;
}

int pre(int &rt, int x) {
    split(rt, x - 1, dl, dr);
    if (!dl) return -INT_MAX;
    int res = k_th(dl, tr[dl].sz);
    rt = merge(dl, dr);
    return res;
}

int nxt(int &rt, int x) {
    split(rt, x, dl, dr);
    if (!dr) return +INT_MAX;
    int res = k_th(dr, 1);
    rt = merge(dl, dr);
    return res;
}
```

```cpp

int main() {
    std::cin.tie(0) -> sync_with_stdio(false);
    int n;
    std::cin >> n;
    for (int cur = 1; cur <= n; ++cur) {
        int ver, op, x, &rt = root[cur];
        std::cin >> ver >> op >> x;
        rt = root[ver];
        if (op == 1) insert(rt, x);
        else if (op == 2) erase(rt, x);
        else if (op == 3) std::cout << getrk(rt, x) << '\n';
        else if (op == 4) std::cout << k_th(rt, x) << '\n';
        else if (op == 5) std::cout << pre(rt, x) << '\n';
        else if (op == 6) std::cout << nxt(rt, x) << '\n';
        else assert(false);
    }
    return 0;
}
```

3.1.3 动态树 Link Cut Tree

⋆ BST/lct-chain.cpp

```cpp
#include <algorithm>
#include <cstdio>
#include <cstring>
#include <cassert>

const int maxn = 300010;
int op, n, m, u, v, c;

struct lct_chain {
    int ch[maxn][2], pa[maxn], val[maxn], sum[maxn], rev[maxn];

    void clear(int x) {
        ch[x][0] = ch[x][1] = pa[x] = val[x] = sum[x] = rev[x] = 0;
    }

    int get(int x) { return (ch[pa[x]][1] == x); }

    int isroot(int x) {
        clear(0);
        return ch[pa[x]][0] != x && ch[pa[x]][1] != x;
    }

    void maintain(int x) {
        clear(0);
        sum[x] = sum[ch[x][0]] ^ val[x] ^ sum[ch[x][1]];
    }

    void down_rev(int x) {
        if (x == 0) return;
        rev[x] ^= 1;
        std::swap(ch[x][0], ch[x][1]);
    }

    void pushdown(int x) {
        clear(0);
        if (rev[x] == 1) {
            down_rev(ch[x][0]);
            down_rev(ch[x][1]);
            rev[x] = 0;
        }
    }

    void update(int x) {
        if (!isroot(x)) update(pa[x]);
        pushdown(x);
    }

    void rotate(int x) {
```

```
        int y = pa[x], z = pa[y], chx = get(x), chy = get(y);
        pa[x] = z;
        if (!isroot(y)) ch[z][chy] = x;
        ch[y][chx] = ch[x][chx ^ 1];
        pa[ch[x][chx ^ 1]] = y;
        ch[x][chx ^ 1] = y;
        pa[y] = x;
        maintain(y);
        maintain(x);
        maintain(z);
    }

    void splay(int x) {
        update(x);
        for (int f = pa[x]; f = pa[x], !isroot(x); rotate(x)) {
            if (!isroot(f)) {
                rotate(get(x) == get(f) ? f : x);
            }
        }
    }

    void access(int x) {
        for (int f = 0; x; f = x, x = pa[x]) {
            splay(x);
            ch[x][1] = f;
            maintain(x);
        }
    }

    void makeroot(int x) {
        access(x);
        splay(x);
        down_rev(x);
    }

    int find(int x) {
        access(x);
        splay(x);
        while (ch[x][0]) x = ch[x][0];
        splay(x);
        return x;
    }

    void split(int u, int v) {
        makeroot(u);
        access(v);
        splay(v);
    }

    void link(int u, int v) {
        if (find(u) != find(v)) {
            makeroot(u);
            pa[u] = v;
        }
    }

    void cut(int u, int v) {
        split(u, v);
        if (ch[v][0] == u && !ch[u][1]) {
            ch[v][0] = pa[u] = 0;
        }
    }
} st;

int main() {
    std::scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) {
        std::scanf("%d", &st.val[i]);
        st.maintain(i);
    }
    while (m--) {
        std::scanf("%d%d%d", &op, &u, &v);
        if (op == 0) {
```

```
            st.split(u, v);
            std::printf("%d\n", st.sum[v]);
        } else if (op == 1) {
            st.link(u, v);
        } else if (op == 2) {
            st.cut(u, v);
        } else if (op == 3) {
            st.splay(u);
            st.val[u] = v;
            st.maintain(u);
        } else {
            assert(false);
        }
    }
    return 0;
}
```

⋆ BST/lct-subtree.cpp

```
#include <algorithm>
#include <iostream>
#include <cstring>
#include <cassert>

const int N = 300010;
int n, m, u, v, c;
std::string op;

struct lct_subtree {
    int ch[N][2], pa[N], rev[N], siz[N], siz2[N];

    void clear(int x) {
        ch[x][0] = ch[x][1] = pa[x] = rev[x] = siz[x] = siz2[x] = 0;
    }

    int get(int x) { return (ch[pa[x]][1] == x); }

    int isroot(int x) {
        clear(0);
        return ch[pa[x]][0] != x && ch[pa[x]][1] != x;
    }

    void maintain(int x) {
        clear(0);
        if (x == 0) return;
        siz[x] = siz[ch[x][0]] + 1 + siz[ch[x][1]] + siz2[x];
    }

    void down_rev(int x) {
        if (x == 0) return;
        rev[x] ^= 1;
        std::swap(ch[x][0], ch[x][1]);
    }

    void pushdown(int x) {
        clear(0);
        if (rev[x] == 1) {
            down_rev(ch[x][0]);
            down_rev(ch[x][1]);
            rev[x] = 0;
        }
    }

    void update(int x) {
        if (!isroot(x)) update(pa[x]);
        pushdown(x);
    }

    void rotate(int x) {
        int y = pa[x], z = pa[y], chx = get(x), chy = get(y);
        pa[x] = z;
        if (!isroot(y)) ch[z][chy] = x;
        ch[y][chx] = ch[x][chx ^ 1];
```

```
        pa[ch[x][chx ^ 1]] = y;
        ch[x][chx ^ 1] = y;
        pa[y] = x;
        maintain(y);
        maintain(x);
        maintain(z);
    }

    void splay(int x) {
        update(x);
        for (int f = pa[x]; f = pa[x], !isroot(x); rotate(x)) {
            if (!isroot(f)) {
                rotate(get(x) == get(f) ? f : x);
            }
        }
    }

    void access(int x) {
        for (int f = 0; x; f = x, x = pa[x]) {
            splay(x);
            siz2[x] += siz[ch[x][1]] - siz[f];
            ch[x][1] = f;
            maintain(x);
        }
    }

    void makeroot(int x) {
        access(x);
        splay(x);
        down_rev(x);
    }

    int find(int x) {
        access(x);
        splay(x);
        while (ch[x][0]) x = ch[x][0];
        splay(x);
        return x;
    }

    void split(int u, int v) {
        makeroot(u);
        access(v);
        splay(v);
    }

    void link(int u, int v) {
        if (find(u) != find(v)) {
            makeroot(u);
            makeroot(v);
            pa[u] = v;
            siz2[v] += siz[u];
        }
    }

    void cut(int u, int v) {
        split(u, v);
        if (ch[v][0] == u && !ch[u][1]) {
            ch[v][0] = pa[u] = 0;
        }
    }
} st;

int main() {
    std::cin.tie(nullptr)->sync_with_stdio(false);
    std::cin >> n >> m;
    for (int i = 1; i <= n; i++) {
        st.maintain(i);
    }
    while (m--) {
        std::cin >> op >> u >> v;
        if (op == "A") {
            st.link(u, v);
```

```cpp
        } else if (op == "Q") {
            st.cut(u, v);
            st.maintain(u);
            st.makeroot(u);
            st.makeroot(v);
            std::cout << 1ll * st.siz[u] * st.siz[v] << '\n';
            st.link(u, v);
        } else {
            assert(false);
        }
    }
    return 0;
}
```

3.2 STL / pbds

3.2.1 优先队列 & 树哈希

⋆ bits-ext/pque.cpp

```cpp
struct cmp { bool operator() (int a, int b) { return a > b; } };
priority_queue<int, vector<int>, cmp> pque;
auto cmp = [](int x, int y) { return x > y; };
priority_queue<int, vector<int>, decltype(cmp) > pque1(cmp);
```

⋆ bits-ext/tree-hash.cpp

```cpp
// #include "hashmap-pbds.cpp"
void dfs(int u, int p) { // 有根
    for (int v : edge[u]) if (v != p) dfs(v, u), H[u] += splitmix64(H[v] ^ SEED);
}
void sol(int u, int p) { // 无根
    if (p != 0) H[u] = H[u] + splitmix64((G[p] - splitmix64(H[u] ^ SEED)) ^ SEED);
    for (int v : edge[u]) if (v != p) sol(v, u);
}
```

3.2.2 bits/extc++.h

⋆ bits-ext/hashmap-pbds.cpp

```cpp
using LL = long long;
using ULL = unsigned long long;
#include <bits/extc++.h>
// or : mt19937_64(chrono::steady_clock::now().time_since_epoch().count())
const int SEED = std::chrono::steady_clock::now().time_since_epoch().count();
struct chash {  // To use most bits rather than just the lowest ones:
    const ULL C = LL(4e18 * acos(0)) | 71;  // large odd number
    LL operator()(LL x) const { return __builtin_bswap64((x ^ SEED) * C); }
};
using HashMap = __gnu_pbds::gp_hash_table<LL, int, chash>;

ULL splitmix64(ULL x) { // http://xorshift.di.unimi.it/splitmix64.c
    x += 0x9e3779b97f4a7c15;
    x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
    x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
    return x ^ (x >> 31);
}
int SPLITMIX32(int z) {
    z += 0x9e3779b9;
    z = (z ^ (z >> 16)) * 0x85ebca6b;
    z = (z ^ (z >> 13)) * 0xc2b2ae35;
    return z ^ (z >> 16);
}
```

⋆ bits-ext/rbtree-pbds.cpp

```cpp
// 插入 xx 数
// 删除 xx 数(若有多个相同的数，因只删除一个)
// 查询 xx 数的排名(排名定义为比当前数小的数的个数 +1+1 )
// 查询排名为 xx 的数
// 求 xx 的前驱(前驱定义为小于 xx，且最大的数)
// 求 xx 的后继(后继定义为大于 xx，且最小的数)
#include <iostream>
#include <map>
#include <ext/pb_ds/assoc_container.hpp>
using namespace std;
using namespace __gnu_pbds;
using pii = pair<int, int>;
tree<pii, null_type, less<pii>, rb_tree_tag, tree_order_statistics_node_update> T;
map<int, int> ins, era; // pbds的bbt不含重复元素，因此需要"手动支持"

int main() {
    cin.tie(0) -> sync_with_stdio(false);
    int n, opt, x;
    cin >> n;
    while (n--) {
        cin >> opt >> x;
        switch (opt) {
            case 1: T.insert(pii(x, ins[x]++)); break;
            case 2: T.erase(T.lower_bound(pii(x, era[x]++))); break;
            case 3: cout << T.order_of_key(pii(x, era[x])) + 1 << "\n"; break;
            case 4: cout << T.find_by_order(x - 1) -> first << "\n"; break;
            case 5: cout << (--T.lower_bound(pii(x, 0))) -> first << "\n"; break;
            case 6: cout << T.upper_bound(pii(x,ins[x])) -> first << "\n"; break;
        }
    }
    return 0;
}
```

3.3 misc

3.3.1 (左偏树) 可并堆 Left Heap

⋆ misc/Left-Heap.cpp

```cpp
struct left_heap {
    left_heap *lc, *rc;
    int val, npl;
    left_heap() {}
    left_heap(left_heap *_l, left_heap *_r, int _v) : lc(_l), rc(_r), val(_v) {
        if (lc->npl < rc->npl) std::swap(lc, rc);
        npl = rc->npl + 1;
    }
} pool[N], *tail = pool, *nil, *hp[N];

left_heap *merge(left_heap *a, left_heap *b) {
    if (a == nil) return b;
    if (b == nil) return a;
    if (a->val > b->val) std::swap(a, b); // 小顶堆
    a->rc = merge(a->rc, b);
    if (a->lc != nil || (a->lc->npl < a->rc->npl)) std::swap(a->lc, a->rc);
    a->npl = a->rc->npl;
    return a;
}

int del_max(left_heap *&h) {
    if (h == nil) return -1;
    int ret = h->val;
    h = merge(h->lc, h->rc);
    return ret;
}

void left_heap_init() {
    nil = new left_heap();
    nil->lc = nil->rc = nil;
```

```
    nil->val = inf;
    nil->npl = 0;
}
```

4 Graph 图论

4.1 特殊图性质

a.竞赛图: 基图为无向完全图的有向简单图。

1. 竞赛图强连通缩点后的 **DAG** 呈链状, 前面的所有点向后面的所有点连边。
2. 任意竞赛图都有哈密顿路径；存在哈密顿回路当且仅当强联通。
3. 竞赛图中大小为 n 的强联通子图中存在大小为 $[3,n]$ 的环。
4. 兰道定理 **(Landau's Theoerm):** 不降序列 $\{s_n\}$ 是合法的比分序列（即竞赛图的出度序列）当且仅当 $\forall 1 \leqslant k \leqslant n, \sum_{i=1}^{k} s_i \leqslant \binom{k}{2}$，且 $\sum_{i=1}^{n} s_i = \binom{n}{2}$。

4.2 SP 最短路 Shortest Path

Johnson 全源最短路——任意图，复杂度 $\mathcal{O}(N^2 + NM\log M)$.

⋆ Johnson.cpp

```cpp
#include <iostream>
#include <cassert>
#include <vector>
#include <queue>

constexpr int inf = 1e9;

template <typename T, typename DT>
std::vector<DT>
spfa(const std::vector<std::vector<std::pair<int, T> > > &edge, int n, int s) {
    std::vector<DT> dis(n, inf);
    std::vector<bool> vis(n, false);
    std::vector<int> cnt(n, 0);

    std::queue<int> que;
    que.push(s);
    vis[s] = true;
    dis[s] = 0;
    cnt[s] = 0;

    while (not que.empty()) {
        int u = que.front();
        que.pop();
        vis[u] = false; // * note: reset vis!
        for (auto [v, w] : edge[u]) {
            if (dis[v] > dis[u] + w) {
                dis[v] = dis[u] + w;
                if (not vis[v]) {
                    cnt[v] ++;
                    que.push(v);
                    vis[v] = true;
                    if (cnt[v] > n) {
                        return {};
                    }
                }
            }
        }
    }
    return dis;
}

template <typename T, typename DT>
std::vector<DT>
dijkstra(const std::vector<std::vector<std::pair<int, T> > > &edge, int n, int s) {
    std::vector<DT> dis(n, inf);
    std::vector<bool> vis(n, false);
    using PLI = std::pair<DT, int>;
```

```cpp
    std::priority_queue< PLI, std::vector<PLI>, std::greater<PLI> > pque;
    pque.emplace(0, s);
    dis[s] = 0;
    while (!pque.empty()) {
        int u = pque.top().second;
        pque.pop();
        if (vis[u]) continue;
        vis[u] = true;
        for (auto [v, w] : edge[u]) {
            if (dis[v] > dis[u] + w) {
                dis[v] = dis[u] + w;
                pque.emplace(dis[v], v);
            }
        }
    }
    return dis;
}

template <typename T, typename DT>
std::vector<std::vector<DT>>
johnson(std::vector<std::vector<std::pair<int, T> > > edge, int n) {
    assert((int) edge.size() == n);

    edge.push_back({}); // * note 1: edge changed (1)
    for (int i = 0; i < n; i++) {
        edge[n].emplace_back(i, 0ll);
    }
    auto dis0 = spfa<T, DT>(edge, n + 1, n);

    bool has_negative_cycle = dis0.empty();
    if (has_negative_cycle) {
        return {};
    }

    for (int u = 0; u < n; u++) {
        for (auto &[v, w] : edge[u]) {
            w += dis0[u] - dis0[v]; // * note 2: edge changed (2)
        }
    }

    std::vector<std::vector<DT>> dis;
    for (int u = 0; u < n; u++) {
        dis.emplace_back(dijkstra<T, DT>(edge, n, u));
        for (int v = 0; v < n; v++) {
            if (dis[u][v] != inf) {
                dis[u][v] -= dis0[u] - dis0[v];
            }
        }
    }

    return dis; // bool has_neg_cyc = dis.empty();
}

int main() {
    std::cin.tie(nullptr)->sync_with_stdio(false);

    int n, m;
    std::cin >> n >> m;

    std::vector<std::vector<std::pair<int, int>>> edge(n);
    while (m--) {
        int u, v, w;
        std::cin >> u >> v >> w;
        u--; v--;
        edge[u].emplace_back(v, w);
    }

    auto dis = johnson<int, int>(edge, n);
    bool has_neg_cyc = dis.empty();
    if (has_neg_cyc) {
        std::cout << "-1\n";
        return 0;
    }
```

```cpp

    for (int u = 0; u < n; u++) {
        long long ans = 0ll;
        for (int v = 0; v < n; v++) {
            ans += 1ll * (v + 1) * dis[u][v];
        }
        std::cout << ans << '\n';
    }

    return 0;
}
```

4.3 MST 最小生成树 Minimal Spanning Tree

4.3.1 矩阵树定理 Kirchhoff's matrix tree theorem

the Laplacian matrix L = the degree matrix D − the adjacency matrix A.

the number of spanning trees = the absolute value of any cofactor of L.

4.3.2 Kruskal（可判定唯一性）

⋆ MST/Kruskal.cpp

```cpp
#include <iostream>
#include <algorithm>
#include <vector>

const static int maxn = 110, inf = 0x3f3f3f3f;
int pa[maxn];

int find(int x) { return pa[x] == x ? x : pa[x] = find(pa[x]); }

struct Edge {
    int u, v, w;
    Edge(int u, int v, int w) : u(u), v(v), w(w) {}
};

bool cmp(Edge a, Edge b) { return a.w < b.w; }

int solve(int n, std::vector<Edge> edges) {
    int m = (int) edges.size();
    for (int i = 1; i <= n; ++i) pa[i] = i;
    sort(edges.begin(), edges.end(), cmp);
    edges.emplace_back(0, 0, inf);
    bool unic = true;
    int sum = 0, tail = -1, avail = 0, used = 0, cnt = n - 1;
    for (int i = 0; i < m; ++i) {
        if (i > tail) {
            if (avail != used) unic = false;
            avail = used = 0;
            do { ++tail; } while (edges[tail].w == edges[tail + 1].w);
            for (int j = i; j <= tail; ++j) {
                if (find(edges[j].u) != find(edges[j].v)) ++avail;
            }
        }
        if (find(edges[i].u) == find(edges[i].v)) continue;
        sum += edges[i].w;
        ++used, --cnt;
        pa[pa[edges[i].u]] = pa[edges[i].v];
    }
    if (avail != used) unic = false;
    if (cnt > 0) return -1; // no MST exists
    else if (!unic) return -2; // multiple MSTs exist
    else return sum; // unique MST exists
}
```

4.3.3 Prim

⋆ MST/Prim.cpp

```
#include <iostream>
#include <vector>
#include <queue>
using PII = std::pair<int, int>;

const static int N = 100010;
bool in_S[N];
std::vector<PII> edge[N];

int prim(int n) {
    std::fill(in_S + 1, in_S + 1 + n, false);
    std::priority_queue<PII, std::vector<PII>, std::greater<PII> > pque;
    pque.emplace(0, 1);
    int cnt = n, ans = 0;
    while (!pque.empty()) {
        int w = pque.top().first;
        int u = pque.top().second;
        pque.pop();
        if (in_S[u]) continue;
        in_S[u] = true;
        ans += w;
        if (0 == (-- cnt)) return ans;
        for (auto p : edge[u]) {
            int v = p.first, c = p.second;
            if (!in_S[v]) pque.emplace(c, v);
        }
    }
    return -1;
}
```

4.3.4 Boruvka

须保证：对于每个连通块，都能够找到与之距离最小的另一联通块。记对于一轮这一过程复杂度为 $O(p)$，那么最终的复杂度为 $O(p\log n)$.

4.3.5 XOR-MST 最小异或生成树

可借助字典树用 **Boruvka** 算法求解，复杂度 $\mathcal{O}(n\log n\log a_i)$.

也可以对字典树 **dfs** 求解，复杂度 $\mathcal{O}(n\log\max(n,a_i))$.

⋆ MST/xormst.cpp

```
#include <algorithm>
#include <iostream>
#include <vector>

constexpr int N = 2'000'000 + 10;
constexpr int T = 30;
int trie[N * T][2], now = 1;

void insert(int x, int u = 0, int t = T - 1) {
    for (int j = t; j >= 0; j--) {
        int c = ((x >> j) & 1);
        if (not trie[u][c]) {
            trie[u][c] = now ++;
        }
        u = trie[u][c];
    }
}

int minxor(int x, int u = 0, int t = T - 1) {
    int res = 0;
    for (int j = t; j >= 0; j--) {
        int c = ((x >> j) & 1);
        if (not trie[u][c]) {
```

```cpp
                res |= (1 << j);
                c ^= 1;
            }
            u = trie[u][c];
        }
        return res;
}

long long dfs(const std::vector<int> &a, int l, int r, int u, int t) {
    if (r - l == 1 or t == -1) {
        return 0ll;
    }
    if (trie[u][0] == 0 or trie[u][1] == 0) {
        return dfs(a, l, r, trie[u][0] | trie[u][1], t - 1);
    }

    int val = a[r - 1] & (~((1 << t) - 1));
    int m = std::lower_bound(a.begin() + l, a.begin() + r, val) - a.begin();

    int res = 1 << t;
    if (m - l <= r - m) {
        for (int i = l; i < m; i++) {
            res = std::min(res, minxor(a[i], trie[u][1], t - 1));
        }
    } else {
        for (int i = m; i < r; i++) {
            res = std::min(res, minxor(a[i], trie[u][0], t - 1));
        }
    }

    return (1 << t) + res + dfs(a, l, m, trie[u][0], t - 1) + dfs(a, m, r, trie[u][1], t - 1);
}

int main() {
    std::cin.tie(nullptr)->sync_with_stdio(false);

    int n;
    std::cin >> n;

    std::vector<int> a(n);
    for (int i = 0; i < n; i++) {
        std::cin >> a[i];
        insert(a[i]);
    }
    std::sort(a.begin(), a.end());

    long long ans = dfs(a, 0, n, 0, T - 1);
    std::cout << ans << std::endl;

    return 0;
}
```

4.4 网络流 Net Flow

⋆ NetFlow/Dinic.cpp

```cpp
#include <iostream>
#include <vector>
#include <queue>

const int INF = 1e9;

template <typename T>
struct Dinic {
    struct Edge {
        int from, to;
        T cap, flow;
        Edge(int u, int v, T c, T f) : from(u), to(v), cap(c), flow(f) {}
    };

    int n, m, s, t;
```

```cpp
    std::vector<Edge> edges;
    std::vector<std::vector<int>> G;
    std::vector<int> dep, cur;

    Dinic(int _n) : n(_n), m(0), G(n), dep(n), cur(n) {}

    int add_edge(int u, int v, T c) {
        edges.emplace_back(u, v, c, 0);
        edges.emplace_back(v, u, 0, 0);
        m = edges.size();
        G[u].push_back(m - 2);
        G[v].push_back(m - 1);
        return m - 2;
    }

    bool bfs() {
        dep.assign(n, 0);
        std::queue<int> que;
        que.push(s);
        dep[s] = 1;
        while (not que.empty()) {
            int x = que.front();
            que.pop();
            for (int i = 0; i < (int) G[x].size(); i++) {
                Edge &e = edges[G[x][i]];
                if (not dep[e.to] and e.cap > e.flow) {
                    dep[e.to] = dep[x] + 1;
                    que.push(e.to);
                }
            }
        }
        return dep[t] > 0;
    }

    T dfs(int x, T a) {
        if (x == t or a == 0) return a;
        T res = 0, f;
        for (int &i = cur[x]; i < (int) G[x].size(); i++) {
            Edge &e = edges[G[x][i]];
            if (dep[x] + 1 == dep[e.to]
            and (f = dfs(e.to, std::min(a, e.cap - e.flow))) > 0) {
                e.flow += f;
                edges[G[x][i] ^ 1].flow -= f;
                res += f;
                a -= f;
                if (a == 0) break;
            }
        }
        return res;
    }

    T max_flow(int s, int t, T lim = INF) {
        this->s = s, this->t = t;
        T flow = 0;
        while (bfs() and flow < lim) {
            cur.assign(n, 0);
            flow += dfs(s, INF);
        }
        return flow;
    }
};
```

4.5 XX 连通分量 XX–Connected Component

⋆ Tarjan–E–BCC.cpp

```cpp
int e[M], e_cnt;
std::vector<int> edge[N];
void add_edge(int u, int v) {
    int i = e_cnt ++;
    edge[u].push_back(i);
```

```cpp
    e[i] = v;
}
int dfn[N], low[N], now, ebcc_cnt;
std::vector<int> ebcc[N], sta;
// start point -> from == -1
void tarjan(int u, int from) {
    dfn[u] = low[u] = ++now;
    sta.push_back(u);
    int child = 0;
    for (int j : edge[u]) {
        if ((j ^ from) == 1) continue;
        child ++;
        int v = e[j];
        if (dfn[v] > 0) {
            low[u] = std::min(low[u], dfn[v]);
        } else {
            tarjan(v, j);
            low[u] = std::min(low[u], low[v]);
            if (low[v] >= dfn[u]) {
                int idx = ebcc_cnt ++, x;
                do {
                    x = sta.back();
                    sta.pop_back();
                    ebcc[idx].push_back(x);
                } while (x != v);
                ebcc[idx].push_back(u);
            }
        }
    }
    if (from == -1 && child == 0) {
        ebcc[ebcc_cnt ++].push_back(u);
    }
}
```

⋆ Tarjan–V–BCC.cpp

```cpp
#include <iostream>
#include <vector>

struct vertex_strongly_connected_components {
    int n, now, cnt;
    std::vector<int> dfn, low, sta;
    std::vector<std::vector<int>> &edge, vbcc;
    vertex_strongly_connected_components(std::vector<std::vector<int>> &edges)
        : n((int) edges.size()), now(0), cnt(0), dfn(n), low(n), edge(edges) {}

    void dfs(int u) {
        dfn[u] = low[u] = ++now;
        sta.push_back(u);

        for (int v : edge[u]) {
            if (dfn[v] > 0) {
                low[u] = std::min(low[u], dfn[v]);
            } else {
                dfs(v);
                low[u] = std::min(low[u], low[v]);
                if (low[v] >= dfn[u]) {
                    int idx = vbcc.size(), x;
                    vbcc.push_back({});
                    do {
                        x = sta.back();
                        sta.pop_back();
                        vbcc[idx].push_back(x);
                    } while (x != v);
                    vbcc[idx].push_back(u);
                }
            }
        }
    }

    std::vector<std::vector<int>> operator() () {
        for (int i = 0; i < n; i++) {
            if (edge[i].size() == 0) {
```

```cpp
                vbcc.push_back({i});
            } else if (dfn[i] == 0) {
                dfs(i);
            }
        }
        return vbcc;
    }
};

int main() {
    std::cin.tie(nullptr)->sync_with_stdio(false);
    int n, m;
    std::cin >> n >> m;

    std::vector<std::vector<int>> edge(n);
    while (m--) {
        int u, v;
        std::cin >> u >> v;
        if (u == v) continue;
        u--, v--;
        edge[u].push_back(v);
        edge[v].push_back(u);
    }

    auto vbcc = vertex_strongly_connected_components(edge)();

    std::cout << vbcc.size() << '\n';
    for (auto &v : vbcc) {
        std::cout << v.size();
        for (int x : v) {
            std::cout << ' ' << x + 1;
        }
        std::cout << '\n';
    }

    return 0;
}
```

⋆ kosaraju.cpp

```cpp
#include <iostream>
#include <vector>

std::vector<int> edge[N], rEdge[N];
std::vector<int> scc_nodes[N];
int scc[N], scc_count;

bool visited[N];
int stack[N], top;

void dfs(int node) {
    visited[node] = true;
    for (int next : edge[node]) {
        if (visited[next]) continue;
        dfs(next);
    }
    stack[++top] = node;
}

void rDfs(int node, int scc_num) {
    visited[node] = true;
    for (int next : rEdge[node]) {
        if (visited[next]) continue;
        rDfs(next, scc_num);
    }
    scc_nodes[scc_num].push_back(node);
    scc[node] = scc_num;
}

void kosaraju(int n) {
    ::top = 0;
    ::scc_count = 0;
    std::fill(visited, visited + n + 1, false);
```

```cpp
    for (int i = 1; i <= n; i++) {
        if (!visited[i]) dfs(i);
    }

    std::fill(visited, visited + n + 1, false);
    while (top) {
        int node = stack[top --];
        if (!visited[node]) rDfs(node, ++scc_count);
    }
}

void clear(int n) {
    for (int i = 1; i <= n; i++) {
        edge[i].clear();
        rEdge[i].clear();
    }
    for (int i = 1; i <= scc_count; i++) {
        scc_nodes[i].clear();
    }
}
```

4.6 树の剖分 Decomposition

⋆ Heavy-Light-Decomposition.cpp

```cpp
#include <iostream>
#include <vector>

const int N = 1'000'010;
const int T = 18;
std::vector<int> edge[N];

int size[N], heavy[N], pa[N][T], depth[N];
void pre_dfs(int u, int p, int dep) {
    size[u] = 1;
    pa[u][0] = p;
    depth[u] = dep;
    for (int t = 0; t + 1 < T; t++) {
        pa[u][t + 1] = pa[pa[u][t]][t];
    }
    for (int v : edge[u]) {
        pre_dfs(v, u, dep + 1);
        size[u] += size[v];
        if (size[v] > size[heavy[u]]) {
            heavy[u] = v;
        }
    }
}

int tin[N], tout[N], top[N], now;
void dfs(int u, int tp) {
    tin[u] = now ++;
    top[u] = tp;
    if (heavy[u]) dfs(heavy[u], tp);
    for (int v : edge[u]) {
        if (v == heavy[u]) continue;
        dfs(v, v);
    }
    tout[u] = now;
}

int a[N], val[N];
using int64 = long long;
struct node {
    int64 sum, lazy;
} tr[N * 4];

#define lc (u * 2 + 1)
#define rc (u * 2 + 2)
#define mid (l + (r - l) / 2)

```

```cpp
void push_up(int u, int l, int r) {
    tr[u].sum = tr[lc].sum + tr[rc].sum;
    tr[u].lazy = 0ll;
}

void down(int u, int l, int r, int64 val) {
    tr[u].sum += (r - l) * val;
    tr[u].lazy += val;
}

void push_down(int u, int l, int r) {
    int64 lazy = tr[u].lazy;
    if (lazy != 0ll) {
        tr[u].lazy = 0ll;
        down(lc, l, mid, lazy);
        down(rc, mid, r, lazy);
    }
}

void build(int u, int l, int r) {
    if (l + 1 == r) {
        tr[u].sum = val[l];
        tr[u].lazy = 0ll;
    } else {
        build(lc, l, mid);
        build(rc, mid, r);
        push_up(u, l, r);
    }
}

void range_add(int u, int l, int r, int lo, int hi, int val) {
    if (lo <= l and r <= hi) {
        down(u, l, r, val);
    } else if (hi <= l or r <= lo or l + 1 == r) {
        // pass
    } else {
        push_down(u, l, r);
        range_add(lc, l, mid, lo, hi, val);
        range_add(rc, mid, r, lo, hi, val);
        push_up(u, l, r);
    }
}

int64 range_sum(int u, int l, int r, int lo, int hi) {
    if (lo <= l and r <= hi) {
        return tr[u].sum;
    } else if (hi <= l or r <= lo or l + 1 == r) {
        return 0ll;
    } else {
        push_down(u, l, r);
        int64 res = range_sum(lc, l, mid, lo, hi) + range_sum(rc, mid, r, lo, hi);
        push_up(u, l, r);
        return res;
    }
}

int main() {
    std::cin.tie(nullptr)->sync_with_stdio(false);

    int n;
    std::cin >> n;
    for (int i = 1; i <= n; i++) {
        std::cin >> a[i];
    }

    for (int i = 2, p; i <= n; i++) {
        std::cin >> p;
        edge[p].push_back(i);
    }

    pre_dfs(1, 1, 0);
    dfs(1, 1);

```

```
    for (int i = 1; i <= n; i++) {
        val[tin[i]] = a[i];
    }
    build(0, 0, n);

    int root = 1;

    auto anc_of = [&] (int u, int d) {
        for (int j = 0; j < T; j++) {
            if ((d >> j) & 1) {
                u = pa[u][j];
            }
        }
        return u;
    };

    auto is_child_of = [&] (int u, int v) -> bool {
        return (tin[v] <= tin[u] and tout[u] <= tout[v]);
    };

    auto add_path = [&] (int u, int v, int k) {
        while (top[u] != top[v]) {
            if (depth[top[u]] < depth[top[v]]) std::swap(u, v);
            range_add(0, 0, n, tin[top[u]], tin[u] + 1, k);
            u = pa[top[u]][0];
        }
        if (depth[u] < depth[v]) std::swap(u, v);
        range_add(0, 0, n, tin[v], tin[u] + 1, k);
    };

    auto add_subtree = [&] (int u, int k) {
        if (root == u) {
            range_add(0, 0, n, tin[1], tout[1], +k);
        } else if (is_child_of(root, u)) {
            int v = anc_of(root, depth[root] - depth[u] - 1);
            range_add(0, 0, n, tin[1], tout[1], +k);
            range_add(0, 0, n, tin[v], tout[v], -k);
        } else {
            range_add(0, 0, n, tin[u], tout[u], +k);
        }
    };

    auto query_path = [&] (int u, int v) {
        int64 res = 0ll;
        while (top[u] != top[v]) {
            if (depth[top[u]] < depth[top[v]]) std::swap(u, v);
            res += range_sum(0, 0, n, tin[top[u]], tin[u] + 1);
            u = pa[top[u]][0];
        }
        if (depth[u] < depth[v]) std::swap(u, v);
        res += range_sum(0, 0, n, tin[v], tin[u] + 1);
        return res;
    };

    auto query_subtree = [&] (int u) {
        if (root == u) {
            return range_sum(0, 0, n, tin[1], tout[1]);
        } else if (is_child_of(root, u)) {
            int v = anc_of(root, depth[root] - depth[u] - 1);
            return range_sum(0, 0, n, tin[1], tout[1])
                 - range_sum(0, 0, n, tin[v], tout[v]);
        } else {
            return range_sum(0, 0, n, tin[u], tout[u]);
        }
    };

    int m;
    std::cin >> m;
    while (m--) {
        int op, u, v, k;
        std::cin >> op;
        switch (op) {
        case 1:
```

```cpp
            std::cin >> u;
            root = u;
            break;
        case 2:
            std::cin >> u >> v >> k;
            add_path(u, v, k);
            break;
        case 3:
            std::cin >> u >> k;
            add_subtree(u, k);
            break;
        case 4:
            std::cin >> u >> v;
            std::cout << query_path(u, v) << '\n';
            break;
        case 5:
            std::cin >> u;
            std::cout << query_subtree(u) << '\n';
            break;
        }
    }

    return 0;
}
```

4.7 支配树 Dominator Tree

⋆ Dominator.cpp

```cpp
#include <vector>
#include <numeric>
#include <iostream>
#include <functional>

std::vector<int> dominator(const std::vector<std::vector<int>> &g, int s) {
    int n = (int) g.size();
    std::vector<int> pos(n, -1), p, label(n), dom(n), sdom(n), dsu(n), par(n);
    std::vector<std::vector<int>> rg(n), bucket(n);
    std::function<void(int)> dfs = [&](int u) {
        int t = (int) p.size();
        p.push_back(u);
        label[t] = sdom[t] = dsu[t] = pos[u] = t;
        for (int v : g[u]) {
            if (pos[v] == -1) {
                dfs(v);
                par[pos[v]] = t;
            }
            rg[pos[v]].push_back(t);
        }
    };
    std::function<int(int, int)> find = [&](int u, int x) {
        if (u == dsu[u]) {
            return x ? -1 : u;
        }
        int v = find(dsu[u], x + 1);
        if (v < 0) {
            return u;
        }
        if (sdom[label[dsu[u]]] < sdom[label[u]]) {
            label[u] = label[dsu[u]];
        }
        dsu[u] = v;
        return x ? v : label[u];
    };
    dfs(s);
    std::iota(dom.begin(), dom.end(), 0);
    for (int i = (int) p.size() - 1; i >= 0; i -= 1) {
        for (int j : rg[i]) {
            sdom[i] = std::min(sdom[i], sdom[find(j, 0)]);
        }
        if (i) {
```

```
                bucket[sdom[i]].push_back(i);
            }
            for (int k : bucket[i]) {
                int j = find(k, 0);
                dom[k] = sdom[j] == sdom[k] ? sdom[j] : j;
            }
            if (i > 1) {
                dsu[i] = par[i];
            }
        }
        for (int i = 1; i < (int) p.size(); i += 1) {
            if (dom[i] != sdom[i]) {
                dom[i] = dom[dom[i]];
            }
        }
        std::vector<int> res(n, -1);
        res[s] = s;
        for (int i = 1; i < (int) p.size(); i += 1) {
            res[p[i]] = p[dom[i]];
        }
        return res;
    }

    int main() {
        std::cin.tie(nullptr)->sync_with_stdio(false);
        int n, m;
        std::cin >> n >> m;
        std::vector<std::vector<int>> g(n);
        for (int i = 0, u, v; i < m; i += 1) {
            std::cin >> u >> v;
            g[u - 1].push_back(v - 1);
        }
        auto p = dominator(g, 0);
        std::vector<std::vector<int>> t(n);
        for (int i = 1; i < n; i += 1) {
            t[p[i]].push_back(i);
        }
        std::vector<int> ans(n, 1);
        std::function<void(int)> dfs = [&](int u) {
            for (int v : t[u]) {
                dfs(v);
                ans[u] += ans[v];
            }
        };
        dfs(0);
        for (int x : ans) {
            std::cout << x << " ";
        }
    }
```

5 Computational Geometry 计算几何

5.1 utils / tools 实用工具

5.1.1 二维向量 vector 2d

⋆ Vec2.cpp

```
point rotate(point p, double a) {
    double cosa = std::cos(a), sina = std::sin(a);
    return point(p.x * cost - p.y * sint, p.x * sint + p.y * cost);
}
```

5.1.2 三维向量 vector 3d

⋆ Vec3.cpp

```
vec3 cross(vec3 a, vec3 b) {
    return vec3(
        a.y * b.z - a.z * b.y,
        a.z * b.x - a.x * b.z,
        a.x * b.y - a.y * b.x
    );
}

// rotate vec `v` around vec `k` by an angle `a`
vec3 rotate3(vec3 v, vec3 k, double a) {
    k = unit(k);
    vec3 kv = cross(k, v);
    return v + std::sin(a) * kv + (1.0 - std::cos(a)) * cross(k, kv);
}
```

5.2 Algorithms 算法

5.2.1 凸包构建 Andrew's Algorithm for Convex Hull

⋆ ConvexHull.cpp

```
convex get_convex(convex p) {
    std::sort(p.begin(), p.end(), [&] (point a, point b) -> bool {
        return a.y != b.y ? a.y < b.y : a.x < b.x;
    });
    int n = (int) p.size(), tp = 0, lim = 1;
    std::vector<int> used(n, 0), t(2 * n);
    for (int i = 0; i < n; i++) {
        while (tp > lim and cross(p[t[tp - 1]] - p[t[tp - 2]], p[i] - p[t[tp - 2]]) <= 0) {
            used[t[-- tp]] = 0;
        }
        used[t[tp ++] = i] = 1;
    }
    lim = std::max(lim, tp);
    for (int i = n - 2; i >= 0; i--) {
        if (used[i]) continue;
        while (tp > lim and cross(p[t[tp - 1]] - p[t[tp - 2]], p[i] - p[t[tp - 2]]) <= 0) {
            used[t[-- tp]] = 0;
        }
        used[t[tp ++] = i] = 1;
    }
    while (cross(p[t[tp - 1]] - p[t[tp - 2]], p[t[0]] - p[t[tp - 2]]) <= 0) {
        used[t[-- tp]] = 0;
    }
    convex c(tp);
    for (int i = 0; i < tp; i++) {
        c[i] = p[t[i]];
    }
    return c;
}
```

```cpp


void reorder_convex(convex &c) {
    int p = 0;
    for (int i = 1; i < (int) c.size(); i++) {
        if (c[i].y < c[p].y or (c[i].y == c[p].y and c[i].x < c[p].x)) {
            p = i;
        }
    }
    std::rotate(c.begin(), c.begin() + p, c.end());
}

convex minkowski_sum(convex c1, convex c2) {
    auto prepare = [&] (convex &c) {
        reorder_convex(c);
        c.push_back(c[0]);
        c.push_back(c[1]);
    };
    int n1 = (int) c1.size(), n2 = (int) c2.size();
    prepare(c1);
    prepare(c2);
    convex c;
    for (int i = 0, j = 0; i < n1 or j < n2; ) {
        c.push_back(c1[i] + c2[j]);
        auto value = cross(c1[i + 1] - c1[i], c2[j + 1] - c2[j]);
        if (value >= 0 and i < n1) i++;
        if (value <= 0 and j < n2) j++;
    }
    return c;
}

bool in_convex(point p, const convex &c) {
    int lo = 0, hi = (int) c.size() - 1;
    while (lo < hi) {
        int mi = hi - (hi - lo) / 2;
        if (cross(c[mi] - c[0], p - c[0]) >= 0) {
            lo = mi;
        } else {
            hi = mi - 1;
        }
    }
    if (hi == 0) {
        return false;
    } else if (hi == (int) c.size() - 1) {
        return on_segment(p, c[0], c[hi]);
    } else {
        return in_triangle(p, c[0], c[hi], c[hi + 1]);
    }
}
```

5.2.2 旋转卡壳 Rotating Calipers

⋆ RotatingCalipers.cpp

```cpp
// required: P <- getConvexHull(P);
double findConvexHullWidth(const std::vector<point> &P) {
    double res = inf;
    int sz = P.size();
    for (int i = 0, q = 1; i < sz; ++i) {
        int j = (i + 1) % sz;
        while (cross(P[j] - P[i], P[q] - P[i]) < cross(P[j] - P[i], P[(q + 1) % sz] - P[i])) {
            q = (q + 1) % sz;
        }
        res = std::min(res, DistLinePoint(P[i], P[j], P[q]));
    }
    return res;
}
```

5.2.3 Delaunay 三角剖分 Triangulation

★ **Delaunay.cpp**

```
typedef long long ll;

bool ge(const ll& a, const ll& b) { return a >= b; }
bool le(const ll& a, const ll& b) { return a <= b; }
bool eq(const ll& a, const ll& b) { return a == b; }
bool gt(const ll& a, const ll& b) { return a > b; }
bool lt(const ll& a, const ll& b) { return a < b; }
int sgn(const ll& a) { return a >= 0 ? a ? 1 : 0 : -1; }

struct pt {
    ll x, y;
    pt() { }
    pt(ll _x, ll _y) : x(_x), y(_y) { }
    pt operator-(const pt& p) const {
        return pt(x - p.x, y - p.y);
    }
    ll cross(const pt& p) const {
        return x * p.y - y * p.x;
    }
    ll cross(const pt& a, const pt& b) const {
        return (a - *this).cross(b - *this);
    }
    ll dot(const pt& p) const {
        return x * p.x + y * p.y;
    }
    ll dot(const pt& a, const pt& b) const {
        return (a - *this).dot(b - *this);
    }
    ll sqrLength() const {
        return this->dot(*this);
    }
    bool operator==(const pt& p) const {
        return eq(x, p.x) && eq(y, p.y);
    }
};

const pt inf_pt = pt(1e18, 1e18);

struct QuadEdge {
    pt origin;
    QuadEdge* rot = nullptr;
    QuadEdge* onext = nullptr;
    bool used = false;
    QuadEdge* rev() const {
        return rot->rot;
    }
    QuadEdge* lnext() const {
        return rot->rev()->onext->rot;
    }
    QuadEdge* oprev() const {
        return rot->onext->rot;
    }
    pt dest() const {
        return rev()->origin;
    }
};

QuadEdge* make_edge(pt from, pt to) {
    QuadEdge* e1 = new QuadEdge;
    QuadEdge* e2 = new QuadEdge;
    QuadEdge* e3 = new QuadEdge;
    QuadEdge* e4 = new QuadEdge;
    e1->origin = from;
    e2->origin = to;
    e3->origin = e4->origin = inf_pt;
    e1->rot = e3;
    e2->rot = e4;
    e3->rot = e2;
    e4->rot = e1;
```

```
    e1->onext = e1;
    e2->onext = e2;
    e3->onext = e4;
    e4->onext = e3;
    return e1;
}

void splice(QuadEdge* a, QuadEdge* b) {
    swap(a->onext->rot->onext, b->onext->rot->onext);
    swap(a->onext, b->onext);
}

void delete_edge(QuadEdge* e) {
    splice(e, e->oprev());
    splice(e->rev(), e->rev()->oprev());
    delete e->rev()->rot;
    delete e->rev();
    delete e->rot;
    delete e;
}

QuadEdge* connect(QuadEdge* a, QuadEdge* b) {
    QuadEdge* e = make_edge(a->dest(), b->origin);
    splice(e, a->lnext());
    splice(e->rev(), b);
    return e;
}

bool left_of(pt p, QuadEdge* e) {
    return gt(p.cross(e->origin, e->dest()), 0);
}

bool right_of(pt p, QuadEdge* e) {
    return lt(p.cross(e->origin, e->dest()), 0);
}

template <class T>
T det3(T a1, T a2, T a3, T b1, T b2, T b3, T c1, T c2, T c3) {
    return a1 * (b2 * c3 - c2 * b3) - a2 * (b1 * c3 - c1 * b3) +
           a3 * (b1 * c2 - c1 * b2);
}

bool in_circle(pt a, pt b, pt c, pt d) {
// If there is __int128, calculate directly.
// Otherwise, calculate angles.
#if defined(__LP64__) || defined(_WIN64)
    __int128 det = 0;
    det -= det3<__int128>(b.x, b.y, b.sqrLength(),
                          c.x, c.y, c.sqrLength(), d.x, d.y, d.sqrLength());
    det += det3<__int128>(a.x, a.y, a.sqrLength(),
                          c.x, c.y, c.sqrLength(), d.x, d.y, d.sqrLength());
    det -= det3<__int128>(a.x, a.y, a.sqrLength(),
                          b.x, b.y, b.sqrLength(), d.x, d.y, d.sqrLength());
    det += det3<__int128>(a.x, a.y, a.sqrLength(),
                          b.x, b.y, b.sqrLength(), c.x, c.y, c.sqrLength());
    return det > 0;
#else
    auto ang = [](pt l, pt mid, pt r) {
        ll x = mid.dot(l, r);
        ll y = mid.cross(l, r);
        long double res = atan2((long double)x, (long double)y);
        return res;
    };
    long double kek = ang(a, b, c) + ang(c, d, a) - ang(b, c, d) - ang(d, a, b);
    if (kek > 1e-8)
        return true;
    else
        return false;
#endif
}

pair<QuadEdge*, QuadEdge*> build_tr(int l, int r, vector<pt>& p) {
    if (r - l + 1 == 2) {
```

```
        QuadEdge* res = make_edge(p[l], p[r]);
        return make_pair(res, res->rev());
    }
    if (r - l + 1 == 3) {
        QuadEdge *a = make_edge(p[l], p[l + 1]), *b = make_edge(p[l + 1], p[r]);
        splice(a->rev(), b);
        int sg = sgn(p[l].cross(p[l + 1], p[r]));
        if (sg == 0) return make_pair(a, b->rev());
        QuadEdge* c = connect(b, a);
        if (sg == 1) return make_pair(a, b->rev());
        else return make_pair(c->rev(), c);
    }
    int mid = (l + r) / 2;
    QuadEdge *ldo, *ldi, *rdo, *rdi;
    tie(ldo, ldi) = build_tr(l, mid, p);
    tie(rdi, rdo) = build_tr(mid + 1, r, p);
    while (true) {
        if (left_of(rdi->origin, ldi)) {
            ldi = ldi->lnext();
            continue;
        }
        if (right_of(ldi->origin, rdi)) {
            rdi = rdi->rev()->onext;
            continue;
        }
        break;
    }
    QuadEdge* basel = connect(rdi->rev(), ldi);
    auto valid = [&basel](QuadEdge* e) { return right_of(e->dest(), basel); };
    if (ldi->origin == ldo->origin) ldo = basel->rev();
    if (rdi->origin == rdo->origin) rdo = basel;
    while (true) {
        QuadEdge* lcand = basel->rev()->onext;
        if (valid(lcand)) {
            while (in_circle(basel->dest(), basel->origin, lcand->dest(),
                             lcand->onext->dest())) {
                QuadEdge* t = lcand->onext;
                delete_edge(lcand);
                lcand = t;
            }
        }
        QuadEdge* rcand = basel->oprev();
        if (valid(rcand)) {
            while (in_circle(basel->dest(), basel->origin, rcand->dest(),
                             rcand->oprev()->dest())) {
                QuadEdge* t = rcand->oprev();
                delete_edge(rcand);
                rcand = t;
            }
        }
        if (!valid(lcand) && !valid(rcand))
            break;
        if (!valid(lcand) ||
            (valid(rcand) && in_circle(lcand->dest(), lcand->origin,
                                       rcand->origin, rcand->dest())))
            basel = connect(rcand, basel->rev());
        else
            basel = connect(basel->rev(), lcand->rev());
    }
    return make_pair(ldo, rdo);
}

vector<tuple<pt, pt, pt>> delaunay(vector<pt> p) {
    sort(p.begin(), p.end(), [](const pt& a, const pt& b) {
        return lt(a.x, b.x) || (eq(a.x, b.x) && lt(a.y, b.y));
    });
    auto res = build_tr(0, (int)p.size() - 1, p);
    QuadEdge* e = res.first;
    vector<QuadEdge*> edges = {e};
    while (lt(e->onext->dest().cross(e->dest(), e->origin), 0)) e = e->onext;
    auto add = [&p, &e, &edges]() {
        QuadEdge* curr = e;
        do {
```

```
            curr->used = true;
            p.push_back(curr->origin);
            edges.push_back(curr->rev());
            curr = curr->lnext();
        } while (curr != e);
    };
    add();
    p.clear();
    int kek = 0;
    while (kek < (int)edges.size()) {
        if (!(e = edges[kek++])->used) add();
    }
    vector<tuple<pt, pt, pt>> ans;
    for (int i = 0; i < (int)p.size(); i += 3) {
        ans.push_back(make_tuple(p[i], p[i + 1], p[i + 2]));
    }
    return ans;
}
```

5.3 Formula/Notes 公式/注记

5.3.1 Pick 定理

对于平行四边形格点中的简单多边形，面积 A、内部格点数 i、边上格点数 b 满足 $A = i + \frac{b}{2} - 1$；对于三角形格点则为 $A = 2i + b - 2$.

5.3.2 三角形外接圆

$$\begin{cases} D = \dfrac{(x_2^2 + y_2^2 - x_3^2 + y_3^2)(y_1 - y_2)}{(x_1 - x_2)(y_2 - y_3) - (x_2 - x_3)(y_1 - y_2)} \\ E = \dfrac{x_1^2 + y_1^2 - x_2^2 - y_2^2 + D(x_1 - x_2)}{y_2 - y_1} \\ F = -(x_1^2 + y_1^2 + Dx_1 + Ey_1) \\ O = (-\dfrac{D}{2}, -\dfrac{E}{2}) \\ r = \dfrac{D^2 + E^2 - 4F}{4} \end{cases}$$

⋆ Circumcircle.cpp

```
// https://fanfansann.blog.csdn.net/article/details/108834399
Circle GetCircumcircle(Point p1, Point p2, Point p3) {
    double Bx = p2.x-p1.x, By = p2.y-p1.y;
    double Cx = p3.x-p1.x, Cy = p3.y-p1.y;
    double D = 2*(Bx*Cy-By*Cx);
    double ansx = (Cy*(Bx*Bx+By*By)-By*(Cx*Cx+Cy*Cy))/D + p1.x;
    double ansy = (Bx*(Cx*Cx+Cy*Cy)-Cx*(Bx*Bx+By*By))/D + p1.y;
    Point p(ansx, ansy);
    return Circle(p, Length(p1-p));
}
```

6 Misc 杂项

6.1 C/C++ IO Cheat Sheet

6.1.1 read()

```cpp
#define getchar() \
    (tt==ss&&(tt=(ss=In)+fread(In,1,1<<20,stdin),ss==tt)?EOF:*ss++)
char In[1 << 20], *ss=In, *tt=In;
int read() {
    int x=0,f=1;
    char c=getchar();
    while(c<'0'||c>'9'){if(c=='-') f=-1;c=getchar();}
    while(c>='0'&&c<='9') x=x*10+c-'0',c=getchar();
    return x*f;
}
```

6.1.2 std::cin

```cpp
    // ref: https://blog.csdn.net/lingfeng2019/article/details/78463012
    // set base = 16, 8, 10 (reading integer)
    std::cin >> (std::hex, std::oct, std::dec) >> number;
    // ignore next `count` chars (counting trailing '\0')
    istream & std::istream::ignore(int count = 1, int delim = EOF);
    // read next `count` chars to `buf` (counting trailing '\0')
    istream & std::istream::get(char * buf, int count, char delim = '\n');
    // read next `count` chars and don't add '\0' at the end of `buff`
    std::cin.read(buf, 5).read(buf + 5, 5);
    // peek (and don't read in) the next char
    char ch = std::cin.peek();
    // then we have: $peek() = get(ch) + putback(ch)$.
```

6.2 Python Helper

记忆化装饰器 **@lru_cache(maxsize=128, typed=False)**

```python
@lru_cache(maxsize = None)  # None表示无限缓存
def fib(n):
    if n < 2:
        return n
    return fib(n - 1) + fib(n - 2)
```

6.3 Bit Tricks

References:

(i) https://codeforces.com/blog/entry/98332

(ii) https://graphics.stanford.edu/~seander/bithacks.html

1. 异号判定

```cpp
// Detect if two integers have opposite signs:
bool f = ((x ^ y) < 0);
```

2. 从高位加到低位的加法 (见 **DFT/NTT**)

3. 枚举子集、枚举超集

```cpp
for (int s = t; s; s = (s - 1) & m) {} // subset
for (int s = t; s < max_state; s = (s + 1) | t) {} // superset
```

4. 枚举所有掩码的子掩码复杂度 $\mathcal{O}(\sum_{m=1}^{2^n} 2^m) = \mathcal{O}(3^n)$

5. next permutation

```
int t = x | (x - 1);
x = (t + 1) | (((~t & -~t) - 1) >> (__builtin_ctz(x) + 1));
```

6.4 .vimrc

```
set nu        "number
set ci        "cindent
set hls       "hlsearch
set ar        "autoread
set aw        "autowrite
set is        "incsearch
set et        "expandtab
set sm        "showmatch
set ai        "autoindent
set ic        "ignorecase
set cul       "cursorline
set cuc       "cursorcolumn
set nocp      "nocompatible
set noeb      "noerrorbells
set smarttab

set ts=4      "tabstop
set hi=1000   "history
set ch=2      "cmdheight
set so=3      "scrolloff
set bs=2      "backspace
set ls=2      "laststatus
set sw=4      "shiftwidth
set sts=4     "sosttabstop
set mouse=a
set completeopt=longest,menu
set statusline=%F%m%r%h%w\ [TYPE=%Y]\ [POS=%l,%v]\ %{strftime(\"%H:%M\")}

color ron
" color torte
nmap tt :%s/\t/    /g<CR>

map <F8> :call Rungdb()<CR>
func! Rungdb()
exec "w"
exec "!g++ % -g -o gdb_%< && gdb ./gdb_%<"
endfunc
```

6.5 Check list

1. 数组是否需要排序？
2. 数据范围是否符合预期？
3. 模数是否正确？
4. 是否混用 **c/c++ IO**？
5. 多组或多轮情形下：初始化好了吗？
6. 下标起始是 **0** 还是 **1**？是否与输入同步（加减 **1**）？
7. **debug** 时修改的细节是否恢复？
8. 被左移的数是否须为 **long long**？
9. 输出格式是否匹配精度要求？精度过高是否会导致死循环？

7 TEST

7.1 ALL todo lists

7.1.1 Math

- ☐ 素性：杜教筛，**Min25-**筛
- ☐ 数论函数：狄利克雷卷积，莫比乌斯反演
- ☐ 线性代数：线性基，常系数线性递推
- ☐ 多项式：拉格朗日插值，集合幂级数（**FWT/FMT**）
- ☐ 组合数学：错排，卡特兰数，斯特林数，伯努利数，**BM**（最短线性递推），**min-max** 容斥，二项式反演，**prufer** 序列
- ☐ 群论：置换，**Burnside** 定理，**Polya** 定理
- ☐ 数值积分：辛普森，自适应辛普森

7.1.2 String

- ☐ (?)：后缀自动机，回文自动机，最小表示法，**Lyndon** 分解

7.1.3 Data-Structure

- ☐ 堆：对顶堆
- ☐ 区间操作：树状数组求第 **k** 大，二维树状数组，李超线段树，线段树合并；
- ☐ 树相关：替罪羊树；笛卡尔树，虚树，**kd-tree**，析合树；长链剖分
- ☐ 并查集：带权并查集，可持续化并查集
- ☐ 分块：莫队，链上分块，树上分块
- ☐ **misc**：**DLX(Dancing Links)**

7.1.4 Graph

- ☐ 最短路：差分约束，**k** 短路
- ☐ 连通分量：圆方树
- ☐ 二分图：匈牙利，**KM**，**Hopcraft-Karp**
- ☐ 网络流：**SAP**；最大流，可行流，**zkw** 费用流 **(?)**；上下界网络流
- ☐ **misc**：欧拉回路，**2-SAT**，斯坦纳树，**3/4** 元环，最小树形图，一般图匹配，最小瓶颈路，全局最小割

7.1.5 CG

- ☐ 工具：交点，**Voronoi**，最小圆覆盖