



# Standard Code Library

*JOWW*

*Changsha University of Science & Technology*

**2023 年 10 月 5 日**

# Contents 目录

<b>1 Math 数学</b> .....	<b>1</b>
<b>1.1 Prime 素性</b> .....	<b>1</b>
1.1.1 大数的判素与分解.....	1
1.1.2 筛法 .....	4
1.1.3 莫比乌斯反演 <b>mobius inversion</b> .....	5
<b>1.2 同余</b> .....	<b>5</b>
1.2.1 <b>exgcd</b> .....	5
1.2.2 <b>exCRT</b> .....	6
1.2.3 离散对数 <b>(ex)bsgs</b> .....	7
<b>1.3 多项式 Polynomial</b> .....	<b>9</b>
1.3.1 <b>FFT/NTT</b> .....	9
<b>1.4 组合数学 Conbinatorics</b> .....	<b>10</b>
1.4.1 <b>Polynomial (MOD 998244353)</b> .....	10
<b>1.5 MAGIC PRIMES</b> .....	<b>15</b>
<b>1.6 Notes</b> .....	<b>15</b>
<b>2 String 字符串</b> .....	<b>17</b>
<b>2.1 String-Match 字符串匹配</b> .....	<b>17</b>
2.1.1 <b>KMP</b> .....	17
2.1.2 <b>Z function</b> .....	17
2.1.3 <b>AC 自动机</b> .....	17
2.1.4 <b>BM</b> .....	18
2.1.5 后缀数组 .....	19
2.1.6 后缀自动机 .....	20
<b>2.2 misc</b> .....	<b>22</b>
2.2.1 回文串 .....	22
2.2.2 <b>Lyndon 分解 - Duval 算法</b> .....	24
2.2.3 最小表示法 .....	24
<b>3 Data Structure 数据结构</b> .....	<b>25</b>
<b>3.1 BST (二叉) 平衡树</b> .....	<b>25</b>
3.1.1 伸展树 <b>Splay</b> .....	25
3.1.2 可持续化 <b>fhq treap</b> .....	29
3.1.3 动态树 <b>Link Cut Tree</b> .....	31
<b>3.2 STL / pbds</b> .....	<b>35</b>
3.2.1 优先队列 & 树哈希 .....	35
3.2.2 <b>bits/extc++.h</b> .....	35
<b>3.3 misc</b> .....	<b>36</b>

3.3.1 (左偏树) 可并堆 <b>Left Heap</b> .....	36
<b>4 Graph 图论</b> .....	<b>38</b>
4.1 特殊图性质.....	38
4.2 SP 最短路 <b>Shortest Path</b> .....	38
4.3 MST 最小生成树 <b>Minimal Spanning Tree</b> .....	40
4.3.1 矩阵树定理 <b>Kirchhoff's matrix tree theorem</b> .....	40
4.3.2 <b>Kruskal</b> (可判定唯一性) .....	40
4.3.3 <b>Prim</b> .....	41
4.3.4 <b>Boruvka</b> .....	41
4.3.5 <b>XOR-MST</b> 最小异或生成树 .....	41
4.4 网络流 <b>Net Flow</b> .....	42
4.5 <b>XX</b> 连通分量 <b>XX-Connected Component</b> .....	43
4.6 树の剖分 <b>Decomposition</b> .....	46
4.7 支配树 <b>Dominator Tree</b> .....	49
<b>5 Computational Geometry 计算几何</b> .....	<b>51</b>
5.1 <b>utils / tools</b> 实用工具.....	51
5.1.1 二维向量 <b>vector 2d</b> .....	51
5.1.2 三维向量 <b>vector 3d</b> .....	51
5.2 <b>Algorithms</b> 算法 .....	51
5.2.1 凸包构建 <b>Andrew's Algorithm for Convex Hull</b> .....	51
5.2.2 旋转卡壳 <b>Rotating Calipers</b> .....	52
5.2.3 <b>Delaunay</b> 三角剖分 <b>Triangulation</b> .....	53
5.3 <b>Formula/Notes</b> 公式/注记 .....	56
5.3.1 <b>Pick</b> 定理 .....	56
5.3.2 三角形外接圆 .....	56
<b>6 Misc 杂项</b> .....	<b>57</b>
6.1 <b>C/C++ IO Cheat Sheet</b> .....	57
6.1.1 <b>read()</b> .....	57
6.1.2 <b>std::cin</b> .....	57
6.2 <b>Python Helper</b> .....	57
6.3 <b>Bit Tricks</b> .....	57
6.4 <b>.vimrc</b> .....	58
6.5 <b>Check list</b> .....	58
<b>7 TEST</b> .....	<b>59</b>
7.1 <b>ALL todo lists</b> .....	59
7.1.1 <b>Math</b> .....	59
7.1.2 <b>String</b> .....	59
7.1.3 <b>Data-Structure</b> .....	59

<b>7.1.4 Graph.....</b>	<b>59</b>
<b>7.1.5 CG .....</b>	<b>59</b>

# 1 Math 数学

## 1.1 Prime 素性

### 1.1.1 大数的判素与分解

#### ★ Miller-Rabin.cpp

```
1 template <typename T, typename Y>
2 bool miller_rabin(T n) {
3     if (n < 3 not n % 2 == 0) return n == 2;
4     T u = n - 1, t = 0;
5     while (u % 2 == 0) u /= 2, t++;
6     constexpr T al[] =
7         // /* int32 */ { 2, 7, 61 };
8         // /* int64 */ { 2, 325, 9375, 28178, 450775, 9780504, 1795265022 };
9         // /* int64 */ { 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37 };
10    for (T a : al) {
11        T v = mod_pow(a, u, n), s;
12        if (v == 1) continue;
13        for (s = 0; s < t; s++) {
14            if (v == n - 1) break;
15            v = (Y) v * v % n;
16        }
17        if (s == t) return false;
18    }
19    return true;
20 }
```

#### ★ Pollard-Rho.cpp

```
1 #include <bits/stdc++.h>
2
3 using int64 = long long;
4 using int128 = __int128;
5
6 int64 add(int64 a, int64 b, int64 p) {
7     a += b;
8     return a >= p ? a - p : a;
9 }
10 int64 sub(int64 a, int64 b, int64 p) {
11     a -= b;
12     return a < 0 ? a + p : a;
13 }
14 int64 mul(int64 a, int64 b, int64 p) {
15     int64 r = (int128) a * b % p;
16     return r - p * int(r >= p) + p * int(r < 0);
17 }
18 int64 mod_pow(int64 a, int64 b, int64 p) {
19     int64 res(1);
20     for (; b; b /= 2, a = mul(a, a, p)) {
21         if (b & 1) {
22             res = mul(res, a, p);
23         }
24     }
25     return res;
26 }
27
28 constexpr int64 base[] =
29     {2, 3, 5, 7, 11, 13, 17, 19, 23}; // < 3e18 (3825123056546413051)
30     // {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}; // < 2^64
31     // {2, 325, 9375, 28178, 450775, 9780504, 1795265022}; // < 2^64
32
33 // Miller Rabin
34 bool is_prime(int64 n) {
35     if (n <= 1) return false;
36     for (int64 p : base) {
37         if (n == p) return true;
38         if (n % p == 0) return false;
39     }
40 }
```

```

40     int64 m = (n - 1) >> __builtin_ctz(n - 1);
41     for (int64 p : base) {
42         int64 t = m, a = mod_pow(p, m, n);
43         while (t != n - 1 && a != 1 && a != n - 1) {
44             a = mul(a, a, n);
45             t *= 2;
46         }
47         if (a != n - 1 && t % 2 == 0) return false;
48     }
49     return true;
50 }
51
52 int64 get_factor(int64 n) {
53     for (int64 p : base) {
54         if (n % p == 0) return p;
55     }
56     auto f = [&](int64 x) { return add(mul(x, x, n), 1, n); };
57     int64 x = 0, y = 0, tot = 0, p = 1, q, g;
58     for (int64 i = 0; (i & 0xfff) || (g = std::gcd(p, n)) == 1; i++, x = f(x), y = f(f(y))) {
59         if (x == y) x = tot++, y = f(x);
60         q = mul(p, sub(x, y, n), n);
61         if (q) p = q;
62     }
63     return g;
64 }
65
66 std::vector<int64> factorization(int64 n) {
67     if (n == 1) return {};
68     if (is_prime(n)) return {n};
69     int64 d = get_factor(n);
70     auto v1 = factorization(d), v2 = factorization(n / d);
71     auto i1 = v1.begin(), i2 = v2.begin();
72     std::vector<int64> ans;
73     while (i1 != v1.end() || i2 != v2.end()) {
74         if (i1 == v1.end()) {
75             ans.push_back(*i2++);
76         } else if (i2 == v2.end()) {
77             ans.push_back(*i1++);
78         } else {
79             if (*i1 < *i2) {
80                 ans.push_back(*i1++);
81             } else {
82                 ans.push_back(*i2++);
83             }
84         }
85     }
86     return ans;
87 }

```

---

## ★ prime.py

---

```

1  # reference:
2  # https://blog.csdn.net/apple_51931783/article/details/123937695
3  from random import randint
4  from math import gcd, isqrt
5
6
7  def miller_rabin(p):
8      ''' 素性测试 '''
9      # 特判 4
10     if p <= 4: return p in (2, 3)
11     # 对 p-1 进行分解
12     pow_2, tmp = 0, p - 1
13     while tmp % 2 == 0:
14         tmp //= 2
15         pow_2 += 1
16     # 进行多次素性测试
17     for a in (2, 3, 5, 7, 11, 13, 17, 19, 23):
18         basic = pow(a, tmp, p)
19         # a^m 是 p 的倍数或者满足条件
20         if basic in (0, 1, p - 1): continue
21         # 进行 r-1 次平方

```

```

22         for _ in range(1, pow_2):
23             basic = basic ** 2 % p
24             # 怎样平方都是 1
25             if basic == 1: return False
26             # 通过 a 的素性测试
27             if basic == p - 1: break
28             # 未通过 a 的素性测试
29             if basic != p - 1: return False
30         # 通过所有 a 的素性测试
31         return True
32
33
34 def pollard_rho(n):
35     ''' 求因数: 7e5 以上 '''
36     # 更新函数
37     bias = randint(3, n - 1)
38     update = lambda i: (i ** 2 + bias) % n
39     # 初始值
40     x = randint(0, n - 1)
41     y = update(x)
42     # 查找序列环
43     while x != y:
44         factor = gcd(abs(x - y), n)
45         # gcd(|x - y|, n) 不为 1 时, 即为答案
46         if factor != 1: return factor
47         x = update(x)
48         y = update(update(y))
49     return n
50
51
52 class prime_factor(dict):
53     ''' 质因数分解
54         require: miller_rabin, pollard_rho'''
55
56     def __init__(self, n):
57         super(prime_factor, self).__init__()
58         self.main(n, gain=1)
59
60     def add(self, n, cnt):
61         # 更新因数表
62         self[n] = self.get(n, 0) + cnt
63
64     def count(self, n, fac):
65         # 试除并记录幂次
66         cnt = 1
67         n //= fac
68         while n % fac == 0:
69             cnt += 1
70             n //= fac
71         return n, cnt
72
73     def main(self, n, gain):
74         if n > 7e5:
75             # 米勒罗宾判素
76             if miller_rabin(n):
77                 self.add(n, gain)
78             else:
79                 # pollard rho 求解因数
80                 fac = pollard_rho(n)
81                 # 求解幂次
82                 n, cnt = self.count(n, fac)
83                 # 递归求解因数的因数
84                 self.main(fac, gain=cnt * gain)
85                 # 递归求解剩余部分
86                 if n > 1: self.main(n, gain=gain)
87         # 试除法求解
88         else:
89             self.try_divide(n, gain=gain)
90
91     def try_divide(self, n, gain=1):
92         ''' 试除法分解 '''
93         i, bound = 2, isqrt(n)

```

```

94         while i <= bound:
95             if n % i == 0:
96                 # 计数 + 整除
97                 n, cnt = self.count(n, i)
98                 # 记录幂次, 更新边界
99                 self.add(i, cnt * gain)
100                 bound = isqrt(n)
101                 i += 1
102             if n > 1: self.add(n, gain)
103
104
105 def is_prime(m:int) -> bool:
106     return miller_rabin(m)
107
108 n = int(input())
109 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

---

```

1  #include <iostream>
2  #include <vector>
3  #include <map>
4
5  using int64 = long long;
6  constexpr int lim = 20'000'001;
7  constexpr int mod = 1e91 + 7;
8
9  int phi[lim + 10];
10 std::vector<int> prime;
11 bool not_prime[lim + 10];
12 int64 sumf[lim + 10], summ[lim + 10];
13
14 void init() {
15     phi[1] = 1;
16     for (int i = 2; i <= lim; i++) {
17         if (not not_prime[i]) {
18             prime.push_back(i);
19             phi[i] = i - 1;
20         }
21         for (int p : prime) {
22             if (i * p > lim) break;
23             not_prime[i * p] = true;
24             if (i % p == 0) {
25                 phi[i * p] = phi[i] * p;
26                 break;
27             } else {
28                 phi[i * p] = phi[i] * (p - 1);
29             }
30         }
31     }
32     for (int i = 1; i <= lim; i++) {
33         sumf[i] = sumf[i - 1] + phi[i];
34     }
35 }
36
37 std::map<int64, int64> Sf, Sm;
38 int64 calcf(int64 l) {
39     if (l <= lim) return sumf[l];
40     if (Sf.find(l) != Sf.end()) return Sf[l];
41     int64 SS = 111 * l * (l + 1) / 2;

```



```

42     if (l & 1) SS = ((l + 1) / 2) % mod * 1 % mod;
43     else SS = (1 / 2) % mod * (l + 1) % mod;
44     for (int64 i = 2, r; i <= l; i = r + 1) {
45         r = 1 / (1 / i);
46         SS -= (r - i + 1) % mod * calcf(1 / i) % mod;
47         SS = (SS + mod) % mod;
48     }
49     return Sf[l] = SS;
50 }
51
52 int main() {
53     init();
54     int64 n;
55     std::cin >> n;
56
57     int ans = 0;
58     for (int64 l = 1, r; l <= n; l = r + 1) {
59         r = n / (n / l);
60         int64 f = n / l;
61         int len = (r - l + 1) % mod;
62         int pref = calcf(f) % mod;
63         int res = 1ll * f % mod * len % mod * (2 * pref - 1) % mod;
64         ans = (ans + res) % mod;
65     }
66
67     std::cout << (mod + ans) % mod << '\n';
68 }

```

---

### 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

```

1 // n = \sum (d : n % d == 0) phi(d)
2 int solve(int n, int m) {
3     int res = 0; // muSum is the presum of mu
4     for (int i = 1, j; i <= std::min(n, m); i = j + 1) {
5         j = std::min(n / (n / i), m / (m / i));
6         res += (muSum[j] - muSum[i - 1]) * (n / i) * (m / i);
7     }
8     return res;
9 }
10 int query(int a, int b, int c, int d, int k) {
11     --a, --c;
12     return + solve(b / k, d / k) - solve(b / k, c / k)
13           - solve(a / k, d / k) + solve(a / k, c / k);
14 }

```

---

## 1.2 同余

### 1.2.1 exgcd

#### ★ exgcd.cpp

```

1 int exgcd(int a, int b, int &x, int &y) {
2     int x1 = 1, x2 = 0, x3 = 0, x4 = 1;
3     while (b != 0) {
4         int c = a / b;
5         std::tie(x1, x2, x3, x4, a, b) =
6             std::make_tuple(x3, x4, x1 - x3 * c, x2 - x4 * c, b, a - b * c);
7     }
8     x = x1, y = x2;
9     return a;
10 }

```

---

## ★ another-exgcd.cpp

---

```
1  #include <iostream>
2
3  template <typename T>
4  T exgcd(T a, T b, T &x, T &y) {
5      // a * x + b * y == return_value
6      if (b == 0) {
7          x = 1; y = 0;
8          return a;
9      } else {
10         T t = exgcd(b, a % b, y, x);
11         // b * y + (a - (a / b) * b) * x == t
12         // a * x + b * (y - (a / b) * x) == t
13         y -= (a / b) * x;
14         return t;
15     }
16 }
17
18 template <typename T>
19 void get_min_pos(T a, T b, T &x, T &y) {
20     T target = x % b;
21     if (target <= 0) target += b; // making target POSITIVE
22     y -= (target - x) / b * a;
23     x = target;
24 }
25
26 int main() {
27     long long a, b, c;
28     std::cin >> a >> b >> c;
29     long long x, y;
30     long long g = exgcd(a, b, x, y);
31     if (c % g != 0) {
32         puts("-1\n");
33     } else {
34         a /= g; b /= g; c /= g;
35         x *= c; y *= c;
36
37         get_min_pos(a, b, x, y);
38         long long o1 = x, o4 = y;
39         get_min_pos(b, a, y, x);
40         long long o2 = y, o3 = x;
41
42         if (x <= 0) {
43             // no pos sol: min pos x, min pos y
44             printf("%lld %lld\n", o1, o2);
45         } else {
46             // all pos: pos sol count, min x, min y, max x, max y
47             long long o0 = (o3 - o1) / b + 1;
48             printf("%lld %lld %lld, %lld, %lld\n", o0, o1, o2, o3, o4);
49         }
50     }
51 }
```

---

## 1.2.2 exCRT

### ★ exCRT.cpp

---

```
1  template <typename T>
2  T exgcd(T a, T b, T& x, T& y) {
3      // ax + by == g
4      if (b == 0) {
5          x = 1;
6          y = 0;
7          return a;
8      } else {
9          // (a - (a / b) * b) x + by == g
10         // ax + b(y + (a / b) x) == g
11         T t = exgcd(b, a % b, y, x);
12         y -= (a / b) * x;
13         return t;
14     }
15 }
```

```

15 }
16
17 // [modulo, remainder] in num, T: int{B}, Y: int{2B}
18 template <typename T, typename Y>
19 T exCRT(const std::vector<std::pair<T, T>> &num) {
20     Y A = 1;
21     T B = 0;
22     for (auto [b, a] : num) {
23         // t = Ax + B = ay + b
24         // Ax - ay = b - B
25         T c = b - B, x, y;
26         T g = exgcd<T>(A, a, x, y);
27         if (c % g != 0) return -1; // no solution.
28         x *= (c / g), y *= (c / g);
29         B = (B + A * x) % (A * a / g);
30         A *= a / g;
31         if (B < 0) B += A;
32     }
33     return B;
34 }

```

---

### 1.2.3 离散对数 (ex)bsgs

#### ★ bsgs-basic.cpp

---

```

1  #include <unordered_map>
2  #include <iostream>
3  #include <cassert>
4  #include <vector>
5  #include <cmath>
6
7  using int64 = long long;
8
9  int64 mod(int64 x, int64 p) {
10     if (x >= p) x -= p;
11     if (x < 0) x += p;
12     return x;
13 }
14
15 int64 mod_add(int64 a, int64 b, int64 p) {
16     return mod(a + b, p);
17 }
18
19 int64 mod_mul(int64 a, int64 b, int64 p) {
20     return mod((__int128_t) a * b % p, p); // 64-bit only
21     int64 res = 0;
22     for (a %= p, b %= p; b; b /= 2, a = mod_add(a, a, p)) {
23         if (b & 1) {
24             res = mod_add(res, a, p);
25         }
26     }
27     return mod(res, p);
28 }
29
30 int64 mod_pow(int64 a, int64 b, int64 p) {
31     int64 res = 1;
32     for (a %= p; b; b /= 2, a = mod_mul(a, a, p)) {
33         if (b & 1) {
34             res = mod_mul(res, a, p);
35         }
36     }
37     return res;
38 }
39
40 std::vector<int64> get_prime_factor(int64 p) {
41     std::vector<int64> res;
42     for (int64 k = 2; k * k <= p; k++) {
43         if (p % k == 0) {
44             res.push_back(k);
45             while (p % k == 0) {
46                 p /= k;
47             }
48         }
49     }
50     if (p > 1) res.push_back(p);
51     return res;
52 }

```

```

48     }
49 }
50 if (p > 1) res.push_back(p);
51 return res;
52 }
53
54 int64 find_primitive_root(int64 p) {
55     auto pf = get_prime_factor(p - 1);
56     for (int i = 1; i < p; i++) {
57         bool valid = true;
58         for (int64 f : pf) {
59             if (mod_pow(i, (p - 1) / f, p) == 1) {
60                 valid = false;
61                 break;
62             }
63         }
64         if (valid) return i;
65     }
66     assert(false);
67 }
68
69 template <typename T>
70 T exgcd(T a, T b, T &x, T &y) {
71     if (b == 0) {
72         x = 1; y = 0;
73         return a;
74     } else {
75         T d = exgcd(b, a % b, y, x);
76         // (a - (a / b) * b)x + by = d
77         // ax + b(y - (a / b)x) = d
78         y -= a / b * x;
79         return d;
80     }
81 }
82
83 //: accepted when T <= 200, p < 1e12, p is prime
84 int main() {
85     int T;
86     int64 p;
87     std::cin >> T >> p;
88     // w * T = p / w ~> w*w = p/T
89     int64 w = std::max((int64) std::sqrt(p / 25), 111);
90     int64 g = find_primitive_root(p);
91     int64 invg = mod_pow(g, p - 2, p);
92     std::unordered_map<int64, int64> umap;
93     umap.reserve(50'000'000);
94     for (int64 i = 0, val = 1, mul = mod_pow(g, w, p); i < p - 1;
95         i += w, val = mod_mul(val, mul, p)) {
96         umap[val] = i;
97     }
98     auto dlog = [&] (int64 num) -> int64 {
99         for (int64 j = 0; j < w; j++, num = mod_mul(num, invg, p)) {
100             // x * pow(g, j) == num -> x = num / pow(g, j)
101             if (umap.count(num)) {
102                 return umap[num] + j;
103             }
104         }
105         assert(false);
106     };
107     for (int cs = 1; cs <= T; cs++) {
108         // find minimal non-negative `x` s.t. `a^x \equiv b \pmod p`
109         int64 a, b;
110         std::cin >> a >> b;
111         int64 ap = dlog(a), bp = dlog(b);
112         // ap * x == bp (mod p - 1)
113         int64 mp = p - 1, x, y;
114         int64 d = exgcd(ap, mp, x, y);
115         if (bp % d != 0) {
116             std::cout << "-1" << std::endl;
117         } else {
118             ap /= d;
119             mp /= d;
120             bp /= d;

```

```

121         x = mod_mul(x, bp, mp);
122         std::cout << x << std::endl;
123     }
124 }
125 }

```

---

### ★ exbsgs.cpp

---

```

1 // a^x = b (mod p) => min({x}); min(\emptyset) := -inf( < 0 )
2
3 int bsgs(int a, int p, int b) {
4     // if (b == 1 % p) return 0;
5
6     int T = __builtin_sqrt(p) + 1;
7     int N = p / T + 1;
8     int cur = b, c = 1;
9     unordered_map<int, int> mp;
10    for (int i = 0; i < T; ++i) {
11        mp[cur] = i;
12        cur = 1LL * cur * a % p;
13        c = 1LL * c * a % p;
14    }
15
16    for (int i = 1, k = c; i <= N; ++i, k = 1LL * k * c % p)
17        if (mp.count(k))
18            return i * T - mp[k];
19
20    return -100;
21 }
22
23 int exbsgs(int a, int p, int b) {
24     int d = __gcd(a, p);
25
26     if (b == 1 % p)
27         return 0;
28     if (d == 1)
29         return bsgs(a, p, b);
30     if (b % d != 0)
31         return -100;
32
33     p /= d, b /= d;
34     int invc, y;
35     exgcd(a / d, p, invc, y);
36     b = (1LL * b * invc % p + p) % p;
37
38     return exbsgs(a, p, b) + 1;
39 }

```

---

## 1.3 多项式 Polynomial

### 1.3.1 FFT/NTT

#### ★ FFT.cpp

---

```

1 #include <iostream>
2 #include <complex>
3 #include <cmath>
4
5 using Complex = complex<double>;
6
7 const double pi = acos(-1.0);
8
9 inline void dft(Complex a[], int len, int f) {
10    for (int i = 0, k = 0; i < len; ++i) {
11        if (i < k) std::swap(a[i], a[k]);
12        for (int j = len >> 1; (k ^= j) < j; j >>= 1);
13    }
14    for (int h = 1; h < len; h *= 2) {
15        Complex wn(cos(pi / h), f * sin(pi / h));
16        for (int L = 0; L < len; L += h * 2) {

```

```

17         Complex t(1.0, 0.0);
18         for (int k = L; k < L + h; k++, t *= wn) {
19             Complex t1 = a[k], t2 = t * a[k + h];
20             a[k] = t1 + t2, a[k + h] = t1 - t2;
21         }
22     }
23 }
24 if (f == -1) {
25     for (int i = 0; i < len; ++i) {
26         a[i] /= len;
27     }
28 }
29 }

```

---

## ★ NTT.cpp

---

```

1  const int P = 998244353;
2
3  void ntt(int a[], int len, int f) {
4      for (int i = 0, k = 0; i < len; i++) {
5          if (i < k) std::swap(a[i], a[k]);
6          for (int j = len >> 1; (k ^= j) < j; j >>= 1) {}
7      }
8      for (int h = 1; h < len; h *= 2) {
9          const int g = 3;
10         int wn = mod_pow(g, (P - 1) / (2 * h));
11         for (int L = 0; L < len; L += 2 * h) {
12             for (int w = 1, k = L; k < L + h; k++, w = 1ll * w * wn % P) {
13                 int x = a[k], y = 1ll * a[k + h] * w % P;
14                 a[k] = x + y;
15                 a[k + h] = x - y;
16                 if (a[k] >= P) a[k] -= P;
17                 if (a[k + h] < 0) a[k + h] += P;
18             }
19         }
20     }
21     if (f == -1) {
22         std::reverse(a + 1, a + len);
23         for (int i = 0, val = inv_of(len); i < len; i++) {
24             a[i] = 1ll * a[i] * val % P;
25         }
26     }
27 }

```

---

## 1.4 组合数学 Combinatorics

### ★ fibonacci.cpp

---

```

1  // fast doubling method
2  template <typename T>
3  std::pair<T, T> fib(int n) {
4      if (n == 0) return { 0, 1 };
5      auto [c0, d0] = fib<T>(n >> 1);
6      T c = c0 * (2 * d0 - c0);
7      T d = c0 * c0 + d0 * d0;
8      if (n & 1) return { d, c + d };
9      else return { c, d };
10 }

```

---

### 1.4.1 Polynomial (MOD 998244353)

#### ★ Poly.cpp

---

```

1  #include <functional>
2  #include <algorithm>
3  #include <iostream>
4  #include <vector>
5
6  const int P = 998'244'353;

```

```

7  const int N = 1e6 + 10;
8
9  int inv[N];
10
11 __attribute__((constructor))
12 void init_inv() {
13     inv[1] = 1;
14     for (int i = 2; i < N; i++) {
15         inv[i] = P - 111 * P / i * inv[P % i] % P;
16     }
17 }
18
19 int add(int x, int y, int p = P) {
20     x += y;
21     if (x >= p) x -= p;
22     if (x < 0) x += p;
23     return x;
24 }
25
26 template <typename T>
27 int mod_pow(int a, T b, int p = P) {
28     int res = 1;
29     for (; b; b /= 2, a = 111 * a * a % p) {
30         if (b & 1) {
31             res = 111 * res * a % p;
32         }
33     }
34     return res;
35 }
36
37 int inv_of(int a, int p = P) {
38     if (a < N) {
39         return inv[a];
40     } else {
41         return mod_pow(a, p - 2, p);
42     }
43 }
44
45 const int primitiveRoot = 3;
46 std::vector<int> Roots { 0, 1 };
47
48 struct poly : public std::vector<int> {
49
50     poly() : std::vector<int>() {}
51
52     poly(int n) : std::vector<int>(n) {}
53
54     poly(const std::vector<int> &a) : std::vector<int>(a) {}
55
56     poly(const std::initializer_list<int> &a) : std::vector<int>(a) {}
57
58     template<class InputIt, class = std::RequireInputIter<InputIt>>
59     poly(InputIt first, InputIt last) : std::vector<int>(first, last) {}
60
61     poly shift(int k) const {
62         if (k >= 0) {
63             poly f(*this);
64             f.insert(f.begin(), k, 0);
65             return f;
66         } else if (this->size() <= -k) {
67             return poly{};
68         } else {
69             return poly(this->begin() + (-k), this->end());
70         }
71     }
72
73     poly trunc(int k) const {
74         poly f(*this);
75         f.resize(k);
76         return f;
77     }
78
79     poly dft(int len) const {

```

```

80     while (len > Roots.size()) {
81         int h = Roots.size();
82         int wn = mod_pow(primitiveRoot, (P - 1) / (2 * h));
83         for (int i = 0, v = 1; i < h; i++, v = 111 * v * wn % P) {
84             Roots.push_back(v);
85         }
86     }
87     poly a = this->trunc(len);
88     for (int i = 0, k = 0; i < len; i++) {
89         if (i < k) std::swap(a[i], a[k]);
90         for (int j = len >> 1; (k ^= j) < j; j >>= 1) {}
91     }
92     for (int h = 1; h < len; h *= 2) {
93         for (int L = 0; L < len; L += 2 * h) {
94             for (int t = L; t < L + h; t++) {
95                 int x = a[t], y = 111 * a[t + h] * Roots[h + t - L] % P;
96                 a[t] = add(x, y);
97                 a[t + h] = add(x, -y);
98             }
99         }
100     }
101     return a;
102 }
103
104 poly idft(int len) const {
105     poly a = this->dft(len);
106     std::reverse(a.begin() + 1, a.end());
107     // int invlen = inv_of(len);
108     int invlen = P - (P - 1) / len;
109     for (int i = 0; i < len; i++) {
110         a[i] = 111 * a[i] * invlen % P;
111     }
112     return a;
113 }
114
115 poly mul(const poly &g0) const {
116     int len = 1, n = this->size() + g0.size() - 2;
117     while (len <= n) {
118         len *= 2;
119     }
120     poly f = this->dft(len);
121     poly g = g0.dft(len);
122     for (int i = 0; i < len; i++) {
123         f[i] = 111 * f[i] * g[i] % P;
124     }
125     return f.idft(len).trunc(n + 1);
126 }
127
128 friend poly operator* (poly f, int x) {
129     for (int &val : f) {
130         val = 111 * val * x % P;
131     }
132     return f;
133 }
134
135 friend poly operator* (int x, const poly &f) {
136     return f * x;
137 }
138
139 friend poly operator* (const poly &f, const poly &g) {
140     return f.mul(g);
141 }
142
143 friend poly operator+ (const poly &f, const poly& g) {
144     poly h(std::max(f.size(), g.size()));
145     for (int i = 0; i < f.size(); i++) {
146         h[i] = add(h[i], +f[i]);
147     }
148     for (int i = 0; i < g.size(); i++) {
149         h[i] = add(h[i], +g[i]);
150     }
151     return h;
152 }

```



```

153
154 friend poly operator- (const poly &f, const poly& g) {
155     poly h(std::max(f.size(), g.size()));
156     for (int i = 0; i < f.size(); i++) {
157         h[i] = add(h[i], +f[i]);
158     }
159     for (int i = 0; i < g.size(); i++) {
160         h[i] = add(h[i], -g[i]);
161     }
162     return h;
163 }
164
165 poly derive() const {
166     poly f(this->size() - 1);
167     for (int i = 1; i < this->size(); i++) {
168         f[i - 1] = 11l * i * (*this)[i] % P;
169     }
170     return f;
171 }
172
173 poly integral() {
174     poly f(this->size() + 1);
175     for (int i = 1; i < f.size(); i++) {
176         f[i] = 11l * (*this)[i - 1] * inv_of(i) % P;
177     }
178     return f;
179 }
180
181 poly inv(int m) const {
182     poly f {inv_of((*this)[0])};
183     int k = 1;
184     while (k < m) {
185         k *= 2;
186         f = (f * (poly{2} - f * this->trunc(k))).trunc(k);
187     }
188     return f;
189 }
190
191 // [q, r] = f / g <==> f = q * g + r
192 std::pair<poly, poly> operator/ (const poly &g) const {
193     int n = this->size() - 1, m = g.size() - 1;
194     poly fr(this->rbegin(), this->rend());
195     poly gr(g.rbegin(), g.rend());
196     poly qr = (fr * gr.inv(n - m + 1)).trunc(n - m + 1);
197     poly q(qr.rbegin(), qr.rend());
198     poly r = ((*this) - g * q).trunc(m);
199     return std::make_pair(q, r);
200 }
201
202 poly log(int m) const {
203     return (this->derive() * this->inv(m)).integral().trunc(m);
204 }
205
206 poly exp(int m) const {
207     poly f{1};
208     int k = 1;
209     while (k < m) {
210         k *= 2;
211         f = (f * (poly{1} - f.log(k) + this->trunc(k))).trunc(k);
212     }
213     return f.trunc(m);
214 }
215
216 poly pow(int k, int m) const {
217     int i = 0;
218     while (i < this->size() and (*this)[i] == 0) {
219         i++;
220     }
221     if (i == this->size() or 11l * i * k >= m) {
222         return poly(m);
223     }
224     int val = (*this)[i];
225     auto f = this->shift(-i) * inv_of(val);

```

```

226     return (f.log(m - i * k) * k).exp(m - i * k).shift(i * k) * mod_pow(val, k);
227 }
228
229 poly pow_bigint(const std::string &k, int m) const {
230     int i = 0;
231     while (i < this->size() and (*this)[i] == 0) {
232         i++;
233     }
234     if (i == this->size() or (i > 0 and (k.length() >= 8 or 111 * i * std::stoi(k) >= m))) {
235         return poly(m);
236     }
237     int k1 = 0, k2 = 0;
238     for (char c : k) {
239         k1 = (1011 * k1 + c - '0') % P;
240         k2 = (1011 * k2 + c - '0') % (P - 1);
241     }
242     int val = (*this)[i];
243     return ((this->shift(-i) * inv_of(val)).log(m - i * k1) * k1
244         .exp(m - i * k1).shift(i * k1) * mod_pow(val, k2);
245 }
246
247 poly sqrt(int m) const {
248     poly f{1};
249     int k = 1;
250     while (k < m) {
251         k *= 2;
252         f = (f + (trunc(k) * f.inv(k)).trunc(k)) * inv_of(2);
253     }
254     return f.trunc(m);
255 }
256
257 poly mult(const poly &g) const {
258     if (g.size() == 0) {
259         return poly{};
260     }
261     int n = g.size();
262     return ((*this) * poly(g.rbegin(), g.rend())).shift(-(n - 1));
263 }
264
265 std::vector<int> eval(std::vector<int> x) const {
266     if (this->size() == 0) {
267         return std::vector<int>(x.size(), 0);
268     }
269     const int n = std::max(x.size(), this->size());
270     std::vector<poly> q(4 * n);
271     std::vector<int> ans(x.size());
272     x.resize(n);
273     std::function<void(int, int, int)> build = [&](int p, int l, int r) {
274         if (r - l == 1) {
275             q[p] = poly{1, -x[l]};
276         } else {
277             int m = (l + r) / 2;
278             build(2 * p, l, m);
279             build(2 * p + 1, m, r);
280             q[p] = q[2 * p] * q[2 * p + 1];
281         }
282     };
283     build(1, 0, n);
284     std::function<void(int, int, int, const poly &)>
285     work = [&](int p, int l, int r, const poly &num) -> void {
286         if (r - l == 1) {
287             if (l < int(ans.size())) {
288                 ans[l] = num[0];
289             }
290         } else {
291             int m = (l + r) / 2;
292             work(2 * p, l, m, num.mult(q[2 * p + 1]).trunc(m - 1));
293             work(2 * p + 1, m, r, num.mult(q[2 * p]).trunc(r - m));
294         }
295     };
296     work(1, 0, n, mult(q[1].inv(n)));
297     return ans;
298 }

```

## 1.5 MAGIC PRIMES

$r \cdot 2^k + 1$	$r$	$k$	$g$
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**):  $g(f(x)) \equiv 0 \pmod{x^n} \Rightarrow f(x) \equiv f_0(x) - \frac{g(f_0(x))}{g'(f_0(x))} \pmod{x^n}$
- 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  $H_n = \begin{cases} \sum_{i=1}^n H_{i-1}H_{n-i} & n \geq 2 \\ 1 & n \in \{0, 1\} \end{cases}$
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  $y=x-m$  ( $m \geq 1$ ) 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

---

```
1 // optimize = false -> next = \pi
2 std::vector<int> get_next(const std::string &t, bool optimize) {
3     std::vector<int> next(t.size());
4     int i = 0, j = -1;
5     next[i] = j;
6     while (t[i] != '\0') {
7         if (j == -1 || t[i] == t[j]) {
8             i++, j++;
9             if (optimize && t[i] == t[j]) next[i] = next[j];
10            else next[i] = j;
11        }
12        else j = next[j];
13    }
14    return next;
15 }
16 std::vector<int> kmp(const std::vector<int> &next, const std::string &s, const std::string &t) {
17     std::vector<int> pos;
18     int i = 0, j = 0;
19     while (s[i] != '\0') {
20         if (j == -1 || s[i] == t[j]) i++, j++;
21         else j = next[j];
22         if (j > -1 && t[j] == '\0') pos.push_back(i - j + 1), j = next[j];
23     }
24     return pos;
25 }
```

---

#### 2.1.2 Z function

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

##### ★ Z-Function.cpp

---

```
1 std::vector<int> z_func(const std::string &s) {
2     int n = s.size();
3     std::vector<int> z(n);
4     for (int i = 1, l = 0, r = 1; i < n; i++) {
5         if (i < r && z[i - l] < r - i) {
6             z[i] = z[i - l];
7         } else {
8             z[i] = std::max(0, r - i);
9             while (z[i] + i < n && s[z[i]] == s[i + z[i]]) {
10                z[i]++;
11            }
12        }
13        if (r < i + z[i]) {
14            l = i; r = i + z[i];
15        }
16    }
17    return z;
18 }
```

---

#### 2.1.3 AC 自动机

##### ★ AC-Automaton.cpp

---

```
1 #include <iostream>
2 #include <queue>
3
4 const int maxnode = 1e6 + 7;
```

```

5  const int charset = 26;
6  const char base = 'a';
7  int ac[maxnode][charset], tot, acc[maxnode], fail[maxnode];
8  inline int getid(int ch) { return int(ch - base); }
9
10 void insert(const std::string &p) {
11     int u = 0;
12     for (const auto &ch : p) {
13         int id = getid(ch);
14         if (!ac[u][id]) ac[u][id] = ++tot;
15         u = ac[u][id];
16     }
17     ++acc[u];
18 }
19
20 // call getfail() after all `insert()` calls.
21 void getfail() {
22     std::queue<int> que;
23     for (int i = 0; i < charset; ++i) {
24         if (ac[0][i]) que.emplace(ac[0][i]);
25     }
26     while (!que.empty()) {
27         int u = que.front(); que.pop();
28         for (int i = 0; i < charset; ++i) {
29             if (ac[u][i]) {
30                 fail[ac[u][i]] = ac[fail[u]][i];
31                 que.emplace(ac[u][i]);
32             } else {
33                 ac[u][i] = ac[fail[u]][i];
34             }
35         }
36     }
37 }
38
39 int match(const std::string &t) {
40     int u = 0, ans = 0;
41     for (const auto &ch : t) {
42         int id = getid(ch);
43         u = ac[u][id];
44         for (int v = u; v && -1 != acc[v]; v = fail[v]) {
45             ans += acc[v];
46             acc[v] = -1;
47         }
48     }
49     return ans;
50 }
51
52 int main() {
53     std::cin.tie(0) -> sync_with_stdio(false);
54     int n;
55     std::cin >> n;
56     while (n--) {
57         std::string t;
58         std::cin >> t;
59         insert(t);
60     }
61     getfail();
62     std::string s;
63     std::cin >> s;
64     std::cout << match(s) << "\n";
65     return 0;
66 }

```

---

## 2.1.4 BM

### ★ Boyer-Moore.cpp

```

1  const int charset = 26, base = 'a', maxlen = 1e6 + 2;
2  int bc0[charset], *bc = bc0 - base, ss[maxlen], gs[maxlen];
3
4  void buildBC(int bc[], const std::string &p) {
5      for (int j = base; j < base + charset; ++j) bc[j] = -1;

```

```

6     for (int m = p.size(), j = 0; j < m; ++j) bc[int(p[j])] = j;
7 } // buildBC - O( s + m )
8
9 void buildSS(int ss[], const std::string &p) {
10     int m = p.size();
11     ss[m - 1] = m;
12     for (int l = m - 1, r = m - 1, j = l - 1; j >= 0; --j)
13         if (l < j && ss[m - r + j - 1] <= j - 1)
14             ss[j] = ss[m - r + j - 1];
15         else {
16             r = j, l = min(l, r);
17             while (0 <= l && p[l] == p[m - r + l - 1]) --l;
18             ss[j] = r - 1;
19         }
20 } // buildSS
21
22 void buildGS(int gs[], const std::string &p) {
23     buildSS(ss, p);
24     int m = p.size();
25     for (int j = 0; j < m; ++j) gs[j] = m;
26     for (int i = 0, j = m - 1; j >= 0; --j)
27         if (j + 1 == ss[j])
28             while (i < m - j - 1)
29                 gs[i++] = m - j - 1;
30     for (int j = 0; j < m - 1; ++j)
31         gs[m - ss[j] - 1] = m - j - 1;
32 } // buildGS
33
34 int match(const std::string &p, const std::string &t) {
35     buildBC(bc, p), buildGS(gs, p);
36     int i = 0;
37     while (t.size() >= i + p.size()) {
38         int j = p.size() - 1;
39         while (p[j] == t[i + j]) if (0 > --j) break;
40         if (0 > j) break;
41         else i += max(gs[j], j - bc[int(t[i + j])]);
42     }
43     return i;
44 } // match - BM version

```

---

## 2.1.5 后缀数组

### ★ SA.cpp

```

1 int count[N], sa[N];
2
3 template <typename T>
4 void radix_sort(const T &str, int old_rank[], int *rank, int n, int m) {
5     std::fill(count, count + m + 1, 0);
6     for (int i = 0; i < n; i++) {
7         count[str[old_rank[i]]] ++;
8     }
9     for (int i = 1; i <= m; i++) {
10         count[i] += count[i - 1];
11     }
12     for (int i = n - 1; i >= 0; i--) {
13         rank[--count[str[old_rank[i]]]] = old_rank[i];
14     }
15 }
16
17 void suffix_array(const std::string &str, int *sa, int n, int m) {
18     static int rank[N], a[N], b[N];
19     std::iota(rank, rank + n, 0);
20     radix_sort(str, rank, sa, n, m);
21
22     rank[sa[0]] = 0;
23     for (int i = 1; i < n; i++) {
24         rank[sa[i]] = rank[sa[i - 1]] + (str[sa[i - 1]] != str[sa[i]]);
25     }
26
27     for (int h = 1; h < n; h *= 2) {
28         std::iota(sa, sa + n, 0);

```

```

29     for (int i = 0; i < n; i++) {
30         a[i] = rank[i] + 1;
31         b[i] = i + h >= n ? 0 : rank[i + h] + 1;
32     }
33     radix_sort(b, sa, rank, n, n);
34     radix_sort(a, rank, sa, n, n);
35
36     rank[sa[0]] = 0;
37     for (int i = 1; i < n; i++) {
38         rank[sa[i]] = rank[sa[i - 1]] +
39             (a[sa[i - 1]] != a[sa[i]] || b[sa[i - 1]] != b[sa[i]]);
40     }
41 }
42 }
43
44 int main() {
45     std::string str;
46     suffix_array(str, sa, str.size(), 128);
47     for (int i = 0; i < str.size(); i++) {
48         std::cout << 1 + sa[i] << " \n"[i + 1 == str.size()];
49     }
50 }

```

---

## 2.1.6 后缀自动机

### ★ SAM.cpp

```

1  #include <iostream>
2
3  constexpr int N = 1'000'000 + 10;
4  constexpr int CHARSET = 26;
5
6  struct sam_node {
7      int len, pa, size;
8      int next[CHARSET];
9      sam_node(int l = 0, int p = 0, int s = 1) : len(l), pa(p), size(s), next{} {}
10 } sam[N * 2];
11
12 int tot, last;
13 void sam_init() {
14     tot = 0;
15     sam[last = ++tot] = sam_node(0, 0, 0);
16 }
17
18 void extend(int c) {
19     int cur = ++tot;
20     sam[cur] = sam_node(sam[last].len + 1);
21
22     int p = last;
23     while (p and not sam[p].next[c]) {
24         sam[p].next[c] = cur;
25         p = sam[p].pa;
26     }
27
28     if (p == 0) {
29         sam[cur].pa = 1;
30     } else {
31         int q = sam[p].next[c];
32         if (sam[p].len + 1 == sam[q].len) {
33             sam[cur].pa = q;
34         } else {
35             int clone = ++tot;
36             sam[clone] = sam[q];
37             sam[clone].len = sam[p].len + 1;
38             sam[clone].size = 0;
39             while (p and sam[p].next[c] == q) {
40                 sam[p].next[c] = clone;
41                 p = sam[p].pa;
42             }
43             sam[q].pa = sam[cur].pa = clone;
44         }
45     }

```



```

46     last = cur;
47 }
48
49 int count[N], rank[N * 2];
50 int main() {
51     std::string s;
52     std::cin >> s;
53     int n = s.size();
54     sam_init();
55     for (char c : s) {
56         extend(c - 'a');
57     }
58
59     for (int i = 1; i <= tot; i++) {
60         count[sam[i].len] ++;
61     }
62     for (int i = 1; i <= n; i++) {
63         count[i] += count[i - 1];
64     }
65     for (int i = 1; i <= tot; i++) {
66         rank[count[sam[i].len] --] = i;
67     }
68
69     long long ans = 0;
70     for (int i = tot; i >= 1; i--) {
71         int x = rank[i], p = sam[x].pa;
72         if (p) sam[p].size += sam[x].size;
73         if (sam[x].size > 1) {
74             ans = std::max(ans, 1LL * sam[x].size * sam[x].len);
75         }
76     }
77
78     // 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
79     std::cout << ans << '\n';
80
81     return 0;
82 }

```

还有广义后缀自动机（多模串）

#### ★ extsam.cpp

```

1  #include <algorithm>
2  #include <iostream>
3
4  constexpr int N = 2'000'000 + 5;
5  constexpr int CHARSET = 26;
6
7  struct suffix_automaton {
8      int tot, link[N], maxlen[N], trans[N][CHARSET];
9      suffix_automaton() { tot = 1; }
10     int insert(int ch, int last) {
11         if (trans[last][ch]) {
12             int p = last, x = trans[p][ch];
13             if (maxlen[p] + 1 == maxlen[x]) {
14                 return x;
15             } else {
16                 int y = ++ tot;
17                 maxlen[y] = maxlen[p] + 1;
18                 for (int i = 0; i < CHARSET; i++) {
19                     trans[y][i] = trans[x][i];
20                 }
21                 while (p and trans[p][ch] == x) {
22                     trans[p][ch] = y;
23                     p = link[p];
24                 }
25                 link[y] = link[x];
26                 link[x] = y;
27                 return y;
28             }
29         }
30         int z = ++ tot, p = last;

```

```

31     maxlen[z] = maxlen[last] + 1;
32     while (p and not trans[p][ch]) {
33         trans[p][ch] = z;
34         p = link[p];
35     }
36     if (!p) {
37         link[z] = 1;
38     } else {
39         int x = trans[p][ch];
40         if (maxlen[p] + 1 == maxlen[x]) {
41             link[z] = x;
42         } else {
43             int y = ++ tot;
44             maxlen[y] = maxlen[p] + 1;
45             for (int i = 0; i < CHARSET; i++) {
46                 trans[y][i] = trans[x][i];
47             }
48             while (p and trans[p][ch] == x) {
49                 trans[p][ch] = y;
50                 p = link[p];
51             }
52             link[y] = link[x];
53             link[z] = link[x] = y;
54         }
55     }
56     return z;
57 }
58 long long sakura() {
59     // 本质不同的子串个数.
60     long long ans = 0;
61     for (int i = 2; i <= tot; i++) {
62         ans += maxlen[i] - maxlen[link[i]];
63     }
64     return ans;
65 }
66 } sam;
67
68 int main() {
69     std::cin.tie(nullptr)->sync_with_stdio(false);
70     int n;
71     std::cin >> n;
72     for (int i = 0; i < n; i++) {
73         std::string s;
74         std::cin >> s;
75         int last = 1;
76         for (char c : s) {
77             last = sam.insert(c - 'a', last);
78         }
79     }
80     std::cout << sam.sakura() << '\n';
81     std::cout << sam.tot << '\n';
82 }

```

---

## 2.2 misc

### 2.2.1 回文串

#### ★ Manacher.cpp

---

```

1  #include <iostream>
2  #include <cstring>
3
4  const int N = 2 * 1.1e7 + 233;
5  int p[N];
6
7  int main() {
8      std::cin.tie(nullptr) -> sync_with_stdio(false);
9
10     std::string s0;
11     std::cin >> s0;
12

```

```

13     std::string s = "$#";
14     for (char c : s0) {
15         s += c;
16         s += '#';
17     }
18     s += '%';
19
20     int n = s.size() - 1;
21
22     int mid = 0, mr = 0, ans = 0;
23     for (int i = 1; i < n; i++) {
24         if (i <= mr) p[i] = std::min(p[2 * mid - i], mr - i + 1);
25         else p[i] = 1;
26         while (s[i - p[i]] == s[i + p[i]]) p[i] ++;
27         if (i + p[i] > mr) mr = i + p[i] - 1, mid = i;
28         ans = std::max(ans, p[i]);
29     }
30     std::cout << ans - 1 << '\n';
31
32     return 0;
33 }

```

## Palindromic Tree 回文树（回文自动机）

### ★ PAM.cpp

```

1  #include <iostream>
2
3  constexpr int N = 2'000'000 + 10;
4  constexpr int CHARSET = 26;
5
6  struct pam_node {
7      int len, fail, dep;
8      int next[CHARSET];
9  };
10
11  struct pam {
12      pam_node tr[N];
13      int root[2], n, tot, last;
14      char s[N] = "$";
15
16      pam() : root{0, 1}, n(0), tot(1), last(0), s("$") {
17          tr[root[0]].len = 0;
18          tr[root[1]].len = -1;
19          tr[root[0]].fail = root[1];
20      }
21
22      int get_fail(int x) {
23          while (s[n - tr[x].len - 1] != s[n]) {
24              x = tr[x].fail;
25          }
26          return x;
27      }
28
29      void insert(int c) {
30          s[++n] = 'a' + c;
31          int p = get_fail(last);
32          if (not tr[p].next[c]) {
33              int x = ++tot;
34              tr[x].len = tr[p].len + 2;
35              tr[x].fail = tr[get_fail(tr[p].fail)].next[c];
36              tr[x].dep = tr[tr[x].fail].dep + 1;
37              tr[p].next[c] = x;
38          }
39          last = tr[p].next[c];
40      }
41  } p;
42
43  int main() {
44      std::string s;
45      std::cin >> s;
46      for (char c : s) {
47          p.insert(c - 'a');

```

```

48     }
49     return 0;
50 }

```

---

## 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**： $s = w_1 + \dots + w_k$ ，其中  $w_i$  是 **Lyndon 串** 且  $w_1 \geq \dots \geq w_k$ 。

### ★ Lyndon.cpp

```

1  #include <cstdio>
2  const int N = 5e6 + 7;
3  char s[N];
4
5  int main() {
6      scanf("%s", s + 1);
7      int i = 1, ans = 0;
8      while (s[i]) {
9          int j = i, k = i + 1;
10         while (s[k] and s[j] <= s[k]) j = s[j] == s[k++] ? j + 1 : i;
11         while (i <= j) i += k - j, ans ^= i - 1; // 所有右端点异或和
12     }
13     printf("%d\n", ans);
14     return 0;
15 }

```

---

## 2.2.3 最小表示法

### ★ minimal-cyclic-shift.cpp

```

1  #include <cstdio>
2  const int N = 6e5 + 7;
3  int n, ans, s[N];
4
5  int main() {
6      scanf("%d", &n);
7      for (int i = 1; i <= n; i++) {
8          scanf("%d", s + i);
9          s[i + n] = s[i];
10     }
11     int i = 1;
12     while (i <= n) {
13         int j = i, k = i + 1;
14         while (k <= n * 2 and s[j] <= s[k]) j = s[j] == s[k++] ? j + 1 : i;
15         while (i <= j) i += k - j, ans = i <= n ? i : ans;
16     }
17     for (int i = 1; i <= n; i++) {
18         printf("%d%c", s[ans - 1 + i], " \n"[i == n]);
19     }
20     return 0;
21 }

```

---

## 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

```
1  #include <iostream>
2  #include <algorithm>
3
4  const static int maxn = 5e5 + 10, inf = 1e9;
5
6  struct Node {
7      int val, lsum, rsum, mxsum, sum, ch[2], p, sz;
8      bool rev, same;
9      Node () {}
10     Node (int v, int p) : val(v), mxsum(v), sum(v), p(p), sz(1), rev(false), same(false) {
11         ch[0] = ch[1] = 0;
12         lsum = rsum = std::max(0, v);
13     }
14 } tr[maxn];
15
16 int nodes[maxn], top, root;
17 void initNodes() {
18     tr[0].mxsum = -inf;
19     for (int i = maxn - 1; i; --i) nodes[++top] = i;
20 }
21 void delNode(int u) { nodes[++top] = u; }
22 int newNode(int v, int p) {
23     int u = nodes[top--];
24     tr[u] = Node(v, p);
25     return u;
26 }
27
28 void pushup(int x) {
29     Node & u = tr[x], & lc = tr[tr[x].ch[0]], & rc = tr[tr[x].ch[1]];
30     u.sz = lc.sz + rc.sz + 1;
31     u.sum = lc.sum + rc.sum + u.val;
32     u.lsum = std::max(lc.lsum, lc.sum + u.val + rc.lsum);
33     u.rsum = std::max(rc.rsum, rc.sum + u.val + lc.rsum);
34     u.mxsum = std::max( { lc.mxsum, rc.mxsum, lc.rsum + u.val + rc.lsum } );
35 }
36
37 void downlz(int x, int v) {
38     if (!x) return;
39     Node & u = tr[x];
40     u.same = true;
41     u.val = v;
42     u.sum = v * u.sz;
43     u.lsum = u.rsum = std::max(0, u.sum);
44     u.mxsum = std::max(v, u.sum);
45 }
46
47 void downrev(int x) {
48     if (!x) return;
49     tr[x].rev ^= true;
50     std::swap(tr[x].ch[0], tr[x].ch[1]);
51     std::swap(tr[x].lsum, tr[x].rsum);
52 }
53
54 void pushdown(int x) {
55     if (tr[x].same) {
56         downlz(tr[x].ch[0], tr[x].val);
57         downlz(tr[x].ch[1], tr[x].val);
58         tr[x].same = tr[x].rev = false;
59     } else if (tr[x].rev) {
60         downrev(tr[x].ch[0]);
61         downrev(tr[x].ch[1]);
62         tr[x].rev = false;
63     }
```

```

64 }
65
66 int get(int x) {
67     return tr[tr[x].p].ch[1] == x;
68 }
69
70 void rotate(int x) {
71     int y = tr[x].p, z = tr[y].p;
72     int kx = get(x), ky = get(y);
73     tr[y].ch[kx] = tr[x].ch[kx ^ 1];
74     if (tr[x].ch[kx ^ 1]) tr[tr[x].ch[kx ^ 1]].p = y;
75     tr[x].ch[kx ^ 1] = y;
76     tr[y].p = x;
77     tr[x].p = z;
78     if (z) tr[z].ch[ky] = x;
79     pushup(y);
80     pushup(x);
81 }
82
83 void splay(int x, int goal) {
84     for (int p; p = tr[x].p, p != goal; rotate(x)) {
85         if (tr[p].p != goal)
86             rotate( get(x) ^ get(p) ? x : p );
87     }
88     if (goal == 0) root = x;
89 }
90
91 int k_th(int k) {
92     int u = root;
93     while (u) {
94         pushdown(u);
95         if (k <= tr[tr[u].ch[0]].sz) u = tr[u].ch[0];
96         else if (k == tr[tr[u].ch[0]].sz + 1) break;
97         else k -= tr[tr[u].ch[0]].sz + 1, u = tr[u].ch[1];
98     }
99     return u;
100 }
101
102 void recycle(int x) {
103     if (x == 0) return;
104     recycle(tr[x].ch[0]);
105     recycle(tr[x].ch[1]);
106     delNode(x);
107 }
108
109 int a[maxn];
110 int build(int l, int r, int p) {
111     int m = (l + r) / 2;
112     int u = newNode(a[m], p);
113     if (l < m) tr[u].ch[0] = build(l, m, u);
114     if (m + 1 < r) tr[u].ch[1] = build(m + 1, r, u);
115     pushup(u);
116     return u;
117 }
118
119 int main() {
120     std::cin.tie(0) -> sync_with_stdio(false);
121     initNodes();
122     int n, m;
123     std::cin >> n >> m;
124     a[0] = a[n + 1] = -inf;
125     for (int i = 1; i <= n; ++i) {
126         std::cin >> a[i];
127     }
128     root = build(0, n + 2, 0);
129     while (m--) {
130         std::string op;
131         std::cin >> op;
132         int posi, tot, c;
133         if (op == "INSERT") {
134             std::cin >> posi >> tot;
135             for (int i = 0; i < tot; ++i) {
136                 std::cin >> a[i];

```

```

137     }
138     int L = k_th(posi + 1), R = k_th(posi + 2);
139     splay(L, 0), splay(R, L);
140     tr[R].ch[0] = build(0, tot, R);
141     pushup(R), pushup(L);
142 } else if (op == "DELETE") {
143     std::cin >> posi >> tot;
144     int L = k_th(posi), R = k_th(posi + tot + 1);
145     splay(L, 0), splay(R, L);
146     recycle(tr[R].ch[0]);
147     tr[R].ch[0] = 0;
148     pushup(R), pushup(L);
149 } else if (op == "MAKE-SAME") {
150     std::cin >> posi >> tot >> c;
151     int L = k_th(posi), R = k_th(posi + tot + 1);
152     splay(L, 0), splay(R, L);
153     downlz(tr[R].ch[0], c);
154     pushup(R), pushup(L);
155 } else if (op == "REVERSE") {
156     std::cin >> posi >> tot;
157     int L = k_th(posi), R = k_th(posi + tot + 1);
158     splay(L, 0), splay(R, L);
159     downrev(tr[R].ch[0]);
160     pushup(R), pushup(L);
161 } else if (op == "GET-SUM") {
162     std::cin >> posi >> tot;
163     int L = k_th(posi), R = k_th(posi + tot + 1);
164     splay(L, 0), splay(R, L);
165     std::cout << tr[tr[R].ch[0]].sum << '\n';
166 } else /* op == "MAX-SUM" */ {
167     std::cout << tr[root].mxsum << '\n';
168 }
169 }
170 return 0;
171 }

```

---

### ★ BST/YetAnotherSplay.cpp

---

```

1  #include <cassert>
2  #include <iostream>
3
4  namespace Solution {
5      const int N = 1010, MAX_NODE = N * N;
6      int nodeCnt, root[N], idx, ch[MAX_NODE][2];
7      typedef int array_type[MAX_NODE];
8      array_type type, pa, sz, sum, stack, lazy, value;
9
10     int newNode() {
11         int x = nodeCnt++;
12         pa[x] = 0;
13         ch[x][0] = ch[x][1] = 0;
14         type[x] = 2;
15         sz[x] = 1;
16         sum[x] = value[x] = lazy[x] = 0;
17         return x;
18     }
19
20     void build(int n) {
21         nodeCnt = 1;
22         for (int i = 1; i <= n; i++) {
23             int last = root[i] = newNode();
24             sz[last] = n + 2;
25             for (int j = 1, now; j <= n + 1; last = now, j++) {
26                 now = newNode();
27                 pa[now] = last;
28                 type[now] = 1;
29                 ch[last][1] = now;
30                 sz[now] = sz[last] - 1;
31             }
32         }
33     }
34
35     void push_up(int x) {

```

```

36     sz[x] = sz[ch[x][0]] + 1 + sz[ch[x][1]];
37     sum[x] = sum[ch[x][0]] + sum[ch[x][1]] + value[x] + (sz[x] % 2 == 1 ? lazy[x] : 0);
38 }
39
40 void push_down(int x) {
41     if (0 == lazy[x]) return;
42     lazy[ch[x][0]] += lazy[x];
43     lazy[ch[x][1]] += (sz[ch[x][0]] % 2 == 1 ? +1 : -1) * lazy[x];
44     value[x] += (sz[ch[x][0]] % 2 == 0 ? +1 : -1) * lazy[x];
45     if (ch[x][0]) push_up(ch[x][0]);
46     if (ch[x][1]) push_up(ch[x][1]);
47     lazy[x] = 0;
48 }
49
50 void rotate(int x) {
51     int t = type[x], y = pa[x], z = ch[x][1 - t];
52     type[x] = type[y];
53     pa[x] = pa[y];
54     if (type[x] != 2) ch[pa[x]][type[x]] = x;
55     type[y] = 1 - t;
56     pa[y] = x;
57     ch[x][1 - t] = y;
58     if (z) {
59         type[z] = t;
60         pa[z] = y;
61     }
62     ch[y][t] = z;
63     push_up(y);
64 }
65
66 void splay(int x) {
67     int top = 0;
68     stack[top++] = x;
69     for (int i = x; type[i] != 2; i = pa[i]) {
70         stack[top++] = pa[i];
71     }
72     do {
73         push_down(stack[--top]);
74     } while (top);
75
76     while (type[x] != 2) {
77         int y = pa[x];
78         if (type[x] == type[y]) rotate(y);
79         else rotate(x);
80         if (type[x] == 2) break;
81         rotate(x);
82     }
83     push_up(x);
84 }
85
86 int find(int x, int rank) {
87     while (true) {
88         push_down(x);
89         if (sz[ch[x][0]] + 1 == rank) break;
90         if (rank <= sz[ch[x][0]]) x = ch[x][0];
91         else rank -= sz[ch[x][0]] + 1, x = ch[x][1];
92     }
93     return x;
94 }
95
96 void split(int &x, int &y, int a) {
97     y = find(x, a + 1);
98     splay(y);
99     x = ch[y][0];
100    type[x] = 2;
101    ch[y][0] = 0;
102    push_up(y);
103 }
104
105 void split3(int &x, int &y, int &z, int a, int b) {
106     split(x, z, b);
107     split(x, y, a - 1);
108 }

```



```

109
110 void join(int &x, int y) {
111     x = find(x, sz[x]);
112     splay(x);
113     ch[x][1] = y;
114     type[y] = 1;
115     pa[y] = x;
116     push_up(x);
117 }
118
119 void join3(int &x, int y, int z) {
120     join(y, z);
121     join(x, y);
122 }
123
124 void main(int n, int m) {
125     build(n);
126     while (m--) {
127         int op, a, s, e;
128         std::cin >> op >> a >> s >> e;
129         if (op == 1) {
130             int y, z;
131             split3(root[a], y, z, s + 1, e + 1);
132             lazy[y] ++;
133             push_up(y);
134             join3(root[a], y, z);
135             std::cout << sum[root[a]] << '\n';
136         } else if (op == 2) {
137             int b, y, z, t;
138             std::cin >> b;
139             split3(root[a], y, z, s + 1, e + 1);
140             split(root[b], t, sz[root[b]] - 1);
141             join(root[a], z);
142             join3(root[b], y, t);
143             std::cout << sum[root[a]] << ' ' << sum[root[b]] << '\n';
144         } else {
145             assert(false);
146         }
147     }
148 }
149 };
150
151 int main() {
152     std::cin.tie(nullptr) -> sync_with_stdio(false);
153     int n, m;
154     while (std::cin >> n >> m) {
155         Solution::main(n, m);
156     }
157     return 0;
158 }

```

### 3.1.2 可持续化 fhq treap

#### ★ BST/PersistentTreap.cpp

```

1 #include <iostream>
2 #include <cassert>
3 #include <climits>
4
5 const int N = 5e5 + 10;
6
7 struct Node {
8     int l, r, key, val, sz;
9     Node () {}
10     Node (int val) : l(0), r(0), key(std::rand()), val(val), sz(1) {}
11     Node (const Node &b) : l(b.l), r(b.r), key(b.key), val(b.val), sz(b.sz) {}
12 } tr[N * 100];
13
14 int tot, dl, dr, tmp, root[N];
15
16 int newNode(int val) {
17     tr[++tot] = Node(val);

```

```

18     return tot;
19 }
20 int clone(int u) {
21     tr[++tot] = Node(tr[u]);
22     return tot;
23 }
24
25 void pushup(int u) {
26     tr[u].sz = tr[tr[u].l].sz + tr[tr[u].r].sz + 1;
27 }
28
29 void split(int u, int x, int &l, int &r) {
30     if (!u) return l = r = 0, void();
31     if (tr[u].val <= x)
32         l = clone(u), split(tr[l].r, x, tr[l].r, r), pushup(l);
33     else
34         r = clone(u), split(tr[r].l, x, l, tr[r].l), pushup(r);
35 }
36
37 int merge(int l, int r) {
38     if (!l || !r) return l | r;
39     int p;
40     if (tr[l].key < tr[r].key)
41         p = clone(l), tr[p].r = merge(tr[p].r, r);
42     else
43         p = clone(r), tr[p].l = merge(l, tr[p].l);
44     pushup(p);
45     return p;
46 }
47
48 void insert(int &rt, int x) {
49     split(rt, x, dl, dr);
50     rt = merge(merge(dl, newNode(x)), dr);
51 }
52
53 void erase(int &rt, int x) {
54     split(rt, x, dl, dr);
55     split(dl, x - 1, dl, tmp);
56     tmp = merge(tr[tmp].l, tr[tmp].r);
57     rt = merge(merge(dl, tmp), dr);
58 }
59
60 int getrk(int &rt, int x) {
61     split(rt, x - 1, dl, dr);
62     int rnk = tr[dl].sz + 1;
63     rt = merge(dl, dr);
64     return rnk;
65 }
66
67 int k_th(int u, int k) {
68     while (u) {
69         if (k <= tr[tr[u].l].sz) u = tr[u].l;
70         else if (tr[tr[u].l].sz + 1 == k) break;
71         else k -= tr[tr[u].l].sz + 1, u = tr[u].r;
72     }
73     return tr[u].val;
74 }
75
76 int pre(int &rt, int x) {
77     split(rt, x - 1, dl, dr);
78     if (!dl) return -INT_MAX;
79     int res = k_th(dl, tr[dl].sz);
80     rt = merge(dl, dr);
81     return res;
82 }
83
84 int nxt(int &rt, int x) {
85     split(rt, x, dl, dr);
86     if (!dr) return +INT_MAX;
87     int res = k_th(dr, 1);
88     rt = merge(dl, dr);
89     return res;
90 }

```

```

91
92 int main() {
93     std::cin.tie(0) -> sync_with_stdio(false);
94     int n;
95     std::cin >> n;
96     for (int cur = 1; cur <= n; ++cur) {
97         int ver, op, x, &rt = root[cur];
98         std::cin >> ver >> op >> x;
99         rt = root[ver];
100         if (op == 1) insert(rt, x);
101         else if (op == 2) erase(rt, x);
102         else if (op == 3) std::cout << getrk(rt, x) << '\n';
103         else if (op == 4) std::cout << k_th(rt, x) << '\n';
104         else if (op == 5) std::cout << pre(rt, x) << '\n';
105         else if (op == 6) std::cout << nxt(rt, x) << '\n';
106         else assert(false);
107     }
108     return 0;
109 }

```

---

### 3.1.3 动态树 Link Cut Tree

#### ★ BST/lct-chain.cpp

---

```

1  #include <algorithm>
2  #include <cstdio>
3  #include <cstring>
4  #include <cassert>
5
6  const int maxn = 300010;
7  int op, n, m, u, v, c;
8
9  struct lct_chain {
10     int ch[maxn][2], pa[maxn], val[maxn], sum[maxn], rev[maxn];
11
12     void clear(int x) {
13         ch[x][0] = ch[x][1] = pa[x] = val[x] = sum[x] = rev[x] = 0;
14     }
15
16     int get(int x) { return (ch[pa[x]][1] == x); }
17
18     int isroot(int x) {
19         clear(0);
20         return ch[pa[x]][0] != x && ch[pa[x]][1] != x;
21     }
22
23     void maintain(int x) {
24         clear(0);
25         sum[x] = sum[ch[x][0]] ^ val[x] ^ sum[ch[x][1]];
26     }
27
28     void down_rev(int x) {
29         if (x == 0) return;
30         rev[x] ^= 1;
31         std::swap(ch[x][0], ch[x][1]);
32     }
33
34     void pushdown(int x) {
35         clear(0);
36         if (rev[x] == 1) {
37             down_rev(ch[x][0]);
38             down_rev(ch[x][1]);
39             rev[x] = 0;
40         }
41     }
42
43     void update(int x) {
44         if (!isroot(x)) update(pa[x]);
45         pushdown(x);
46     }
47
48     void rotate(int x) {

```

```

49     int y = pa[x], z = pa[y], chx = get(x), chy = get(y);
50     pa[x] = z;
51     if (!isroot(y)) ch[z][chy] = x;
52     ch[y][chx] = ch[x][chx ^ 1];
53     pa[ch[x][chx ^ 1]] = y;
54     ch[x][chx ^ 1] = y;
55     pa[y] = x;
56     maintain(y);
57     maintain(x);
58     maintain(z);
59 }
60
61 void splay(int x) {
62     update(x);
63     for (int f = pa[x]; f = pa[x], !isroot(x); rotate(x)) {
64         if (!isroot(f)) {
65             rotate(get(x) == get(f) ? f : x);
66         }
67     }
68 }
69
70 void access(int x) {
71     for (int f = 0; x; f = x, x = pa[x]) {
72         splay(x);
73         ch[x][1] = f;
74         maintain(x);
75     }
76 }
77
78 void makeroot(int x) {
79     access(x);
80     splay(x);
81     down_rev(x);
82 }
83
84 int find(int x) {
85     access(x);
86     splay(x);
87     while (ch[x][0]) x = ch[x][0];
88     splay(x);
89     return x;
90 }
91
92 void split(int u, int v) {
93     makeroot(u);
94     access(v);
95     splay(v);
96 }
97
98 void link(int u, int v) {
99     if (find(u) != find(v)) {
100         makeroot(u);
101         pa[u] = v;
102     }
103 }
104
105 void cut(int u, int v) {
106     split(u, v);
107     if (ch[v][0] == u && !ch[u][1]) {
108         ch[v][0] = pa[u] = 0;
109     }
110 }
111 } st;
112
113 int main() {
114     std::scanf("%d%d", &n, &m);
115     for (int i = 1; i <= n; i++) {
116         std::scanf("%d", &st.val[i]);
117         st.maintain(i);
118     }
119     while (m--) {
120         std::scanf("%d%d%d", &op, &u, &v);
121         if (op == 0) {

```

```

122         st.split(u, v);
123         std::printf("%d\n", st.sum[v]);
124     } else if (op == 1) {
125         st.link(u, v);
126     } else if (op == 2) {
127         st.cut(u, v);
128     } else if (op == 3) {
129         st.splay(u);
130         st.val[u] = v;
131         st.maintain(u);
132     } else {
133         assert(false);
134     }
135 }
136 return 0;
137 }

```

---

### ★ BST/lct-subtree.cpp

---

```

1  #include <algorithm>
2  #include <iostream>
3  #include <cstring>
4  #include <cassert>
5
6  const int N = 300010;
7  int n, m, u, v, c;
8  std::string op;
9
10 struct lct_subtree {
11     int ch[N][2], pa[N], rev[N], siz[N], siz2[N];
12
13     void clear(int x) {
14         ch[x][0] = ch[x][1] = pa[x] = rev[x] = siz[x] = siz2[x] = 0;
15     }
16
17     int get(int x) { return (ch[pa[x]][1] == x); }
18
19     int isroot(int x) {
20         clear(0);
21         return ch[pa[x]][0] != x && ch[pa[x]][1] != x;
22     }
23
24     void maintain(int x) {
25         clear(0);
26         if (x == 0) return;
27         siz[x] = siz[ch[x][0]] + 1 + siz[ch[x][1]] + siz2[x];
28     }
29
30     void down_rev(int x) {
31         if (x == 0) return;
32         rev[x] ^= 1;
33         std::swap(ch[x][0], ch[x][1]);
34     }
35
36     void pushdown(int x) {
37         clear(0);
38         if (rev[x] == 1) {
39             down_rev(ch[x][0]);
40             down_rev(ch[x][1]);
41             rev[x] = 0;
42         }
43     }
44
45     void update(int x) {
46         if (!isroot(x)) update(pa[x]);
47         pushdown(x);
48     }
49
50     void rotate(int x) {
51         int y = pa[x], z = pa[y], chx = get(x), chy = get(y);
52         pa[x] = z;
53         if (!isroot(y)) ch[z][chy] = x;
54         ch[y][chx] = ch[x][chx ^ 1];

```

```

55     pa[ch[x][chx ^ 1]] = y;
56     ch[x][chx ^ 1] = y;
57     pa[y] = x;
58     maintain(y);
59     maintain(x);
60     maintain(z);
61 }
62
63 void splay(int x) {
64     update(x);
65     for (int f = pa[x]; f = pa[x], !isroot(x); rotate(x)) {
66         if (!isroot(f)) {
67             rotate(get(x) == get(f) ? f : x);
68         }
69     }
70 }
71
72 void access(int x) {
73     for (int f = 0; x; f = x, x = pa[x]) {
74         splay(x);
75         siz2[x] += siz[ch[x][1]] - siz[f];
76         ch[x][1] = f;
77         maintain(x);
78     }
79 }
80
81 void makeroot(int x) {
82     access(x);
83     splay(x);
84     down_rev(x);
85 }
86
87 int find(int x) {
88     access(x);
89     splay(x);
90     while (ch[x][0]) x = ch[x][0];
91     splay(x);
92     return x;
93 }
94
95 void split(int u, int v) {
96     makeroot(u);
97     access(v);
98     splay(v);
99 }
100
101 void link(int u, int v) {
102     if (find(u) != find(v)) {
103         makeroot(u);
104         makeroot(v);
105         pa[u] = v;
106         siz2[v] += siz[u];
107     }
108 }
109
110 void cut(int u, int v) {
111     split(u, v);
112     if (ch[v][0] == u && !ch[u][1]) {
113         ch[v][0] = pa[u] = 0;
114     }
115 }
116 } st;
117
118 int main() {
119     std::cin.tie(nullptr)->sync_with_stdio(false);
120     std::cin >> n >> m;
121     for (int i = 1; i <= n; i++) {
122         st.maintain(i);
123     }
124     while (m--) {
125         std::cin >> op >> u >> v;
126         if (op == "A") {
127             st.link(u, v);

```

```

128         } else if (op == "Q") {
129             st.cut(u, v);
130             st.maintain(u);
131             st.makeroot(u);
132             st.makeroot(v);
133             std::cout << 1ll * st.siz[u] * st.siz[v] << '\n';
134             st.link(u, v);
135         } else {
136             assert(false);
137         }
138     }
139     return 0;
140 }

```

---

## 3.2 STL / pbds

### 3.2.1 优先队列 & 树哈希

#### ★ bits-ext/pque.cpp

```

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

```

---

#### ★ bits-ext/tree-hash.cpp

```

1 // #include "hashmap-pbds.cpp"
2 void dfs(int u, int p) { // 有根
3     for (int v : edge[u]) if (v != p) dfs(v, u), H[u] += splitmix64(H[v] ^ SEED);
4 }
5 void sol(int u, int p) { // 无根
6     if (p != 0) H[u] = H[u] + splitmix64((G[p] - splitmix64(H[u] ^ SEED)) ^ SEED);
7     for (int v : edge[u]) if (v != p) sol(v, u);
8 }

```

---

### 3.2.2 bits/extc++.h

#### ★ bits-ext/hashmap-pbds.cpp

```

1 using LL = long long;
2 using ULL = unsigned long long;
3 #include <bits/extc++.h>
4 // or : mt19937_64(chrono::steady_clock::now().time_since_epoch().count())
5 const int SEED = std::chrono::steady_clock::now().time_since_epoch().count();
6 struct chash { // To use most bits rather than just the lowest ones:
7     const ULL C = LL(4e18 * acos(0)) | 71; // large odd number
8     LL operator()(LL x) const { return __builtin_bswap64((x ^ SEED) * C); }
9 };
10 using HashMap = __gnu_pbds::gp_hash_table<LL, int, chash>;
11
12 ULL splitmix64(ULL x) { // http://xorshift.di.unimi.it/splitmix64.c
13     x += 0x9e3779b97f4a7c15;
14     x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
15     x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
16     return x ^ (x >> 31);
17 }
18 int SPLITMIX32(int z) {
19     z += 0x9e3779b9;
20     z = (z ^ (z >> 16)) * 0x85ebca6b;
21     z = (z ^ (z >> 13)) * 0xc2b2ae35;
22     return z ^ (z >> 16);
23 }

```

---

#### ★ bits-ext/rbtree-pbds.cpp

---

```

1 // 插入 xx 数
2 // 删除 xx 数(若有多个相同的数, 因只删除一个)
3 // 查询 xx 数的排名(排名定义为比当前数小的数的个数 +1+1 )
4 // 查询排名为 xx 的数
5 // 求 xx 的前驱(前驱定义为小于 xx, 且最大的数)
6 // 求 xx 的后继(后继定义为大于 xx, 且最小的数)
7 #include <iostream>
8 #include <map>
9 #include <ext/pb_ds/assoc_container.hpp>
10 using namespace std;
11 using namespace __gnu_pbds;
12 using pii = pair<int, int>;
13 tree<pii, null_type, less<pii>, rb_tree_tag, tree_order_statistics_node_update> T;
14 map<int, int> ins, era; // pbds的bbt不含重复元素, 因此需要"手动支持"
15
16 int main() {
17     cin.tie(0) -> sync_with_stdio(false);
18     int n, opt, x;
19     cin >> n;
20     while (n--) {
21         cin >> opt >> x;
22         switch (opt) {
23             case 1: T.insert(pii(x, ins[x]++)); break;
24             case 2: T.erase(T.lower_bound(pii(x, era[x]++))); break;
25             case 3: cout << T.order_of_key(pii(x, era[x])) + 1 << "\n"; break;
26             case 4: cout << T.find_by_order(x - 1) -> first << "\n"; break;
27             case 5: cout << (--T.lower_bound(pii(x, 0))) -> first << "\n"; break;
28             case 6: cout << T.upper_bound(pii(x, ins[x])) -> first << "\n"; break;
29         }
30     }
31     return 0;
32 }

```

---

## 3.3 misc

### 3.3.1 (左偏树) 可并堆 Left Heap

#### ★ misc/Left-Heap.cpp

---

```

1 struct left_heap {
2     left_heap *lc, *rc;
3     int val, npl;
4     left_heap() {}
5     left_heap(left_heap *_l, left_heap *_r, int _v) : lc(_l), rc(_r), val(_v) {
6         if (lc->npl < rc->npl) std::swap(lc, rc);
7         npl = rc->npl + 1;
8     }
9 } pool[N], *tail = pool, *nil, *hp[N];
10
11 left_heap *merge(left_heap *a, left_heap *b) {
12     if (a == nil) return b;
13     if (b == nil) return a;
14     if (a->val > b->val) std::swap(a, b); // 小顶堆
15     a->rc = merge(a->rc, b);
16     if (a->lc != nil || (a->lc->npl < a->rc->npl)) std::swap(a->lc, a->rc);
17     a->npl = a->rc->npl;
18     return a;
19 }
20
21 int del_max(left_heap *&h) {
22     if (h == nil) return -1;
23     int ret = h->val;
24     h = merge(h->lc, h->rc);
25     return ret;
26 }
27
28 void left_heap_init() {
29     nil = new left_heap();
30     nil->lc = nil->rc = nil;

```



```
31     nil->val = inf;  
32     nil->npl = 0;  
33 }
```

---

## 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 Theorem**): 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  $\forall 1 \leq k \leq n, \sum_{i=1}^k s_i \leq \binom{k}{2}$ , 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**

```
1  #include <iostream>
2  #include <cassert>
3  #include <vector>
4  #include <queue>
5
6  constexpr int inf = 1e9;
7
8  template <typename T, typename DT>
9  std::vector<DT>
10 spfa(const std::vector<std::vector<std::pair<int, T>>> &edge, int n, int s) {
11     std::vector<DT> dis(n, inf);
12     std::vector<bool> vis(n, false);
13     std::vector<int> cnt(n, 0);
14
15     std::queue<int> que;
16     que.push(s);
17     vis[s] = true;
18     dis[s] = 0;
19     cnt[s] = 0;
20
21     while (not que.empty()) {
22         int u = que.front();
23         que.pop();
24         vis[u] = false; // * note: reset vis!
25         for (auto [v, w] : edge[u]) {
26             if (dis[v] > dis[u] + w) {
27                 dis[v] = dis[u] + w;
28                 if (not vis[v]) {
29                     cnt[v]++;
30                     que.push(v);
31                     vis[v] = true;
32                     if (cnt[v] > n) {
33                         return {};
34                     }
35                 }
36             }
37         }
38     }
39     return dis;
40 }
41
42 template <typename T, typename DT>
43 std::vector<DT>
44 dijkstra(const std::vector<std::vector<std::pair<int, T>>> &edge, int n, int s) {
45     std::vector<DT> dis(n, inf);
46     std::vector<bool> vis(n, false);
47     using PLI = std::pair<DT, int>;
```

```

48     std::priority_queue< PLI, std::vector<PLI>, std::greater<PLI> > pque;
49     pque.emplace(0, s);
50     dis[s] = 0;
51     while (!pque.empty()) {
52         int u = pque.top().second;
53         pque.pop();
54         if (vis[u]) continue;
55         vis[u] = true;
56         for (auto [v, w] : edge[u]) {
57             if (dis[v] > dis[u] + w) {
58                 dis[v] = dis[u] + w;
59                 pque.emplace(dis[v], v);
60             }
61         }
62     }
63     return dis;
64 }
65
66 template <typename T, typename DT>
67 std::vector<std::vector<DT>>
68 johnson(std::vector<std::vector<std::pair<int, T> > > edge, int n) {
69     assert((int) edge.size() == n);
70
71     edge.push_back({}); // * note 1: edge changed (1)
72     for (int i = 0; i < n; i++) {
73         edge[n].emplace_back(i, 0ll);
74     }
75     auto dis0 = spfa<T, DT>(edge, n + 1, n);
76
77     bool has_negative_cycle = dis0.empty();
78     if (has_negative_cycle) {
79         return {};
80     }
81
82     for (int u = 0; u < n; u++) {
83         for (auto &[v, w] : edge[u]) {
84             w += dis0[u] - dis0[v]; // * note 2: edge changed (2)
85         }
86     }
87
88     std::vector<std::vector<DT>> dis;
89     for (int u = 0; u < n; u++) {
90         dis.emplace_back(dijkstra<T, DT>(edge, n, u));
91         for (int v = 0; v < n; v++) {
92             if (dis[u][v] != inf) {
93                 dis[u][v] -= dis0[u] - dis0[v];
94             }
95         }
96     }
97
98     return dis; // bool has_neg_cyc = dis.empty();
99 }
100
101 int main() {
102     std::cin.tie(nullptr)->sync_with_stdio(false);
103
104     int n, m;
105     std::cin >> n >> m;
106
107     std::vector<std::vector<std::pair<int, int>>> edge(n);
108     while (m--) {
109         int u, v, w;
110         std::cin >> u >> v >> w;
111         u--; v--;
112         edge[u].emplace_back(v, w);
113     }
114
115     auto dis = johnson<int, int>(edge, n);
116     bool has_neg_cyc = dis.empty();
117     if (has_neg_cyc) {
118         std::cout << "-1\n";
119         return 0;
120     }

```

```

121
122     for (int u = 0; u < n; u++) {
123         long long ans = 0ll;
124         for (int v = 0; v < n; v++) {
125             ans += 1ll * (v + 1) * dis[u][v];
126         }
127         std::cout << ans << '\n';
128     }
129
130     return 0;
131 }

```

---

## 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

---

```

1  #include <iostream>
2  #include <algorithm>
3  #include <vector>
4
5  const static int maxn = 110, inf = 0x3f3f3f3f;
6  int pa[maxn];
7
8  int find(int x) { return pa[x] == x ? x : pa[x] = find(pa[x]); }
9
10 struct Edge {
11     int u, v, w;
12     Edge(int u, int v, int w) : u(u), v(v), w(w) {}
13 };
14
15 bool cmp(Edge a, Edge b) { return a.w < b.w; }
16
17 int solve(int n, std::vector<Edge> edges) {
18     int m = (int) edges.size();
19     for (int i = 1; i <= n; ++i) pa[i] = i;
20     sort(edges.begin(), edges.end(), cmp);
21     edges.emplace_back(0, 0, inf);
22     bool unic = true;
23     int sum = 0, tail = -1, avail = 0, used = 0, cnt = n - 1;
24     for (int i = 0; i < m; ++i) {
25         if (i > tail) {
26             if (avail != used) unic = false;
27             avail = used = 0;
28             do { ++tail; } while (edges[tail].w == edges[tail + 1].w);
29             for (int j = i; j <= tail; ++j) {
30                 if (find(edges[j].u) != find(edges[j].v)) ++avail;
31             }
32         }
33         if (find(edges[i].u) == find(edges[i].v)) continue;
34         sum += edges[i].w;
35         ++used, --cnt;
36         pa[pa[edges[i].u]] = pa[edges[i].v];
37     }
38     if (avail != used) unic = false;
39     if (cnt > 0) return -1; // no MST exists
40     else if (!unic) return -2; // multiple MSTs exist
41     else return sum; // unique MST exists
42 }

```

---

### 4.3.3 Prim

#### ★ MST/Prim.cpp

```
1 #include <iostream>
2 #include <vector>
3 #include <queue>
4 using PII = std::pair<int, int>;
5
6 const static int N = 100010;
7 bool in_S[N];
8 std::vector<PII> edge[N];
9
10 int prim(int n) {
11     std::fill(in_S + 1, in_S + 1 + n, false);
12     std::priority_queue<PII, std::vector<PII>, std::greater<PII> > pque;
13     pque.emplace(0, 1);
14     int cnt = n, ans = 0;
15     while (!pque.empty()) {
16         int w = pque.top().first;
17         int u = pque.top().second;
18         pque.pop();
19         if (in_S[u]) continue;
20         in_S[u] = true;
21         ans += w;
22         if (0 == (-- cnt)) return ans;
23         for (auto p : edge[u]) {
24             int v = p.first, c = p.second;
25             if (!in_S[v]) pque.emplace(c, v);
26         }
27     }
28     return -1;
29 }
```

### 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

```
1 #include <algorithm>
2 #include <iostream>
3 #include <vector>
4
5 constexpr int N = 2'000'000 + 10;
6 constexpr int T = 30;
7 int trie[N * T][2], now = 1;
8
9 void insert(int x, int u = 0, int t = T - 1) {
10     for (int j = t; j >= 0; j--) {
11         int c = ((x >> j) & 1);
12         if (not trie[u][c]) {
13             trie[u][c] = now++;
14         }
15         u = trie[u][c];
16     }
17 }
18
19 int minxor(int x, int u = 0, int t = T - 1) {
20     int res = 0;
21     for (int j = t; j >= 0; j--) {
22         int c = ((x >> j) & 1);
23         if (not trie[u][c]) {
```

```

24         res |= (1 << j);
25         c ^= 1;
26     }
27     u = trie[u][c];
28 }
29 return res;
30 }
31
32 long long dfs(const std::vector<int> &a, int l, int r, int u, int t) {
33     if (r - l == 1 or t == -1) {
34         return 0ll;
35     }
36     if (trie[u][0] == 0 or trie[u][1] == 0) {
37         return dfs(a, l, r, trie[u][0] | trie[u][1], t - 1);
38     }
39
40     int val = a[r - 1] & (~((1 << t) - 1));
41     int m = std::lower_bound(a.begin() + l, a.begin() + r, val) - a.begin();
42
43     int res = 1 << t;
44     if (m - l <= r - m) {
45         for (int i = l; i < m; i++) {
46             res = std::min(res, minxor(a[i], trie[u][1], t - 1));
47         }
48     } else {
49         for (int i = m; i < r; i++) {
50             res = std::min(res, minxor(a[i], trie[u][0], t - 1));
51         }
52     }
53
54     return (1 << t) + res + dfs(a, l, m, trie[u][0], t - 1) + dfs(a, m, r, trie[u][1], t - 1);
55 }
56
57 int main() {
58     std::cin.tie(nullptr)->sync_with_stdio(false);
59
60     int n;
61     std::cin >> n;
62
63     std::vector<int> a(n);
64     for (int i = 0; i < n; i++) {
65         std::cin >> a[i];
66         insert(a[i]);
67     }
68     std::sort(a.begin(), a.end());
69
70     long long ans = dfs(a, 0, n, 0, T - 1);
71     std::cout << ans << std::endl;
72
73     return 0;
74 }

```

---

## 4.4 网络流 Net Flow

### ★ NetFlow/Dinic.cpp

```

1  #include <iostream>
2  #include <vector>
3  #include <queue>
4
5  const int INF = 1e9;
6
7  template <typename T>
8  struct Dinic {
9      struct Edge {
10         int from, to;
11         T cap, flow;
12         Edge(int u, int v, T c, T f) : from(u), to(v), cap(c), flow(f) {}
13     };
14
15     int n, m, s, t;

```

```

16     std::vector<Edge> edges;
17     std::vector<std::vector<int>>> G;
18     std::vector<int> dep, cur;
19
20     Dinic(int _n) : n(_n), m(0), G(n), dep(n), cur(n) {}
21
22     int add_edge(int u, int v, T c) {
23         edges.emplace_back(u, v, c, 0);
24         edges.emplace_back(v, u, 0, 0);
25         m = edges.size();
26         G[u].push_back(m - 2);
27         G[v].push_back(m - 1);
28         return m - 2;
29     }
30
31     bool bfs() {
32         dep.assign(n, 0);
33         std::queue<int> que;
34         que.push(s);
35         dep[s] = 1;
36         while (not que.empty()) {
37             int x = que.front();
38             que.pop();
39             for (int i = 0; i < (int) G[x].size(); i++) {
40                 Edge &e = edges[G[x][i]];
41                 if (not dep[e.to] and e.cap > e.flow) {
42                     dep[e.to] = dep[x] + 1;
43                     que.push(e.to);
44                 }
45             }
46         }
47         return dep[t] > 0;
48     }
49
50     T dfs(int x, T a) {
51         if (x == t or a == 0) return a;
52         T res = 0, f;
53         for (int &i = cur[x]; i < (int) G[x].size(); i++) {
54             Edge &e = edges[G[x][i]];
55             if (dep[x] + 1 == dep[e.to]
56                 and (f = dfs(e.to, std::min(a, e.cap - e.flow))) > 0) {
57                 e.flow += f;
58                 edges[G[x][i] ^ 1].flow -= f;
59                 res += f;
60                 a -= f;
61                 if (a == 0) break;
62             }
63         }
64         return res;
65     }
66
67     T max_flow(int s, int t, T lim = INF) {
68         this->s = s, this->t = t;
69         T flow = 0;
70         while (bfs() and flow < lim) {
71             cur.assign(n, 0);
72             flow += dfs(s, INF);
73         }
74         return flow;
75     }
76 };

```

## 4.5 XX 连通分量 XX-Connected Component

### ★ Tarjan-E-BCC.cpp

```

1  int e[M], e_cnt;
2  std::vector<int> edge[N];
3  void add_edge(int u, int v) {
4      int i = e_cnt++;
5      edge[u].push_back(i);

```

```

6     e[i] = v;
7 }
8 int dfn[N], low[N], now, ebcc_cnt;
9 std::vector<int> ebcc[N], sta;
10 // start point -> from == -1
11 void tarjan(int u, int from) {
12     dfn[u] = low[u] = ++now;
13     sta.push_back(u);
14     int child = 0;
15     for (int j : edge[u]) {
16         if ((j ^ from) == 1) continue;
17         child++;
18         int v = e[j];
19         if (dfn[v] > 0) {
20             low[u] = std::min(low[u], dfn[v]);
21         } else {
22             tarjan(v, j);
23             low[u] = std::min(low[u], low[v]);
24             if (low[v] >= dfn[u]) {
25                 int idx = ebcc_cnt ++, x;
26                 do {
27                     x = sta.back();
28                     sta.pop_back();
29                     ebcc[idx].push_back(x);
30                 } while (x != v);
31                 ebcc[idx].push_back(u);
32             }
33         }
34     }
35     if (from == -1 && child == 0) {
36         ebcc[ebcc_cnt ++].push_back(u);
37     }
38 }

```

---

### ★ Tarjan-V-BCC.cpp

---

```

1 #include <iostream>
2 #include <vector>
3
4 struct vertex_strongly_connected_components {
5     int n, now, cnt;
6     std::vector<int> dfn, low, sta;
7     std::vector<std::vector<int>> &edge, vbcc;
8     vertex_strongly_connected_components(std::vector<std::vector<int>> &edges)
9         : n((int) edges.size()), now(0), cnt(0), dfn(n), low(n), edge(edges) {}
10
11 void dfs(int u) {
12     dfn[u] = low[u] = ++now;
13     sta.push_back(u);
14
15     for (int v : edge[u]) {
16         if (dfn[v] > 0) {
17             low[u] = std::min(low[u], dfn[v]);
18         } else {
19             dfs(v);
20             low[u] = std::min(low[u], low[v]);
21             if (low[v] >= dfn[u]) {
22                 int idx = vbcc.size(), x;
23                 vbcc.push_back({});
24                 do {
25                     x = sta.back();
26                     sta.pop_back();
27                     vbcc[idx].push_back(x);
28                 } while (x != v);
29                 vbcc[idx].push_back(u);
30             }
31         }
32     }
33 }
34
35 std::vector<std::vector<int>> operator() () {
36     for (int i = 0; i < n; i++) {
37         if (edge[i].size() == 0) {

```



```

38         vbcc.push_back({i});
39     } else if (dfn[i] == 0) {
40         dfs(i);
41     }
42 }
43 return vbcc;
44 }
45 };
46
47 int main() {
48     std::cin.tie(nullptr)->sync_with_stdio(false);
49     int n, m;
50     std::cin >> n >> m;
51
52     std::vector<std::vector<int>> edge(n);
53     while (m--) {
54         int u, v;
55         std::cin >> u >> v;
56         if (u == v) continue;
57         u--, v--;
58         edge[u].push_back(v);
59         edge[v].push_back(u);
60     }
61
62     auto vbcc = vertex_strongly_connected_components(edge());
63
64     std::cout << vbcc.size() << '\n';
65     for (auto &v : vbcc) {
66         std::cout << v.size();
67         for (int x : v) {
68             std::cout << ' ' << x + 1;
69         }
70         std::cout << '\n';
71     }
72
73     return 0;
74 }

```

---

### ★ kosaraju.cpp

---

```

1  #include <iostream>
2  #include <vector>
3
4  std::vector<int> edge[N], rEdge[N];
5  std::vector<int> scc_nodes[N];
6  int scc[N], scc_count;
7
8  bool visited[N];
9  int stack[N], top;
10
11 void dfs(int node) {
12     visited[node] = true;
13     for (int next : edge[node]) {
14         if (visited[next]) continue;
15         dfs(next);
16     }
17     stack[++top] = node;
18 }
19
20 void rDfs(int node, int scc_num) {
21     visited[node] = true;
22     for (int next : rEdge[node]) {
23         if (visited[next]) continue;
24         rDfs(next, scc_num);
25     }
26     scc_nodes[scc_num].push_back(node);
27     scc[node] = scc_num;
28 }
29
30 void kosaraju(int n) {
31     ::top = 0;
32     ::scc_count = 0;
33     std::fill(visited, visited + n + 1, false);

```

```

34     for (int i = 1; i <= n; i++) {
35         if (!visited[i]) dfs(i);
36     }
37
38     std::fill(visited, visited + n + 1, false);
39     while (top) {
40         int node = stack[top--];
41         if (!visited[node]) rDfs(node, ++scc_count);
42     }
43 }
44
45 void clear(int n) {
46     for (int i = 1; i <= n; i++) {
47         edge[i].clear();
48         rEdge[i].clear();
49     }
50     for (int i = 1; i <= scc_count; i++) {
51         scc_nodes[i].clear();
52     }
53 }

```

---

## 4.6 木の剖分 Decomposition

### ★ Heavy-Light-Decomposition.cpp

---

```

1  #include <iostream>
2  #include <vector>
3
4  const int N = 1'000'010;
5  const int T = 18;
6  std::vector<int> edge[N];
7
8  int size[N], heavy[N], pa[N][T], depth[N];
9  void pre_dfs(int u, int p, int dep) {
10     size[u] = 1;
11     pa[u][0] = p;
12     depth[u] = dep;
13     for (int t = 0; t + 1 < T; t++) {
14         pa[u][t + 1] = pa[pa[u][t]][t];
15     }
16     for (int v : edge[u]) {
17         pre_dfs(v, u, dep + 1);
18         size[u] += size[v];
19         if (size[v] > size[heavy[u]]) {
20             heavy[u] = v;
21         }
22     }
23 }
24
25 int tin[N], tout[N], top[N], now;
26 void dfs(int u, int tp) {
27     tin[u] = now++;
28     top[u] = tp;
29     if (heavy[u]) dfs(heavy[u], tp);
30     for (int v : edge[u]) {
31         if (v == heavy[u]) continue;
32         dfs(v, v);
33     }
34     tout[u] = now;
35 }
36
37 int a[N], val[N];
38 using int64 = long long;
39 struct node {
40     int64 sum, lazy;
41 } tr[N * 4];
42
43 #define lc (u * 2 + 1)
44 #define rc (u * 2 + 2)
45 #define mid (l + (r - 1) / 2)
46

```

```

47 void push_up(int u, int l, int r) {
48     tr[u].sum = tr[lc].sum + tr[rc].sum;
49     tr[u].lazy = 0ll;
50 }
51
52 void down(int u, int l, int r, int64 val) {
53     tr[u].sum += (r - l) * val;
54     tr[u].lazy += val;
55 }
56
57 void push_down(int u, int l, int r) {
58     int64 lazy = tr[u].lazy;
59     if (lazy != 0ll) {
60         tr[u].lazy = 0ll;
61         down(lc, l, mid, lazy);
62         down(rc, mid, r, lazy);
63     }
64 }
65
66 void build(int u, int l, int r) {
67     if (l + 1 == r) {
68         tr[u].sum = val[l];
69         tr[u].lazy = 0ll;
70     } else {
71         build(lc, l, mid);
72         build(rc, mid, r);
73         push_up(u, l, r);
74     }
75 }
76
77 void range_add(int u, int l, int r, int lo, int hi, int val) {
78     if (lo <= l and r <= hi) {
79         down(u, l, r, val);
80     } else if (hi <= l or r <= lo or l + 1 == r) {
81         // pass
82     } else {
83         push_down(u, l, r);
84         range_add(lc, l, mid, lo, hi, val);
85         range_add(rc, mid, r, lo, hi, val);
86         push_up(u, l, r);
87     }
88 }
89
90 int64 range_sum(int u, int l, int r, int lo, int hi) {
91     if (lo <= l and r <= hi) {
92         return tr[u].sum;
93     } else if (hi <= l or r <= lo or l + 1 == r) {
94         return 0ll;
95     } else {
96         push_down(u, l, r);
97         int64 res = range_sum(lc, l, mid, lo, hi) + range_sum(rc, mid, r, lo, hi);
98         push_up(u, l, r);
99         return res;
100     }
101 }
102
103 int main() {
104     std::cin.tie(nullptr)->sync_with_stdio(false);
105
106     int n;
107     std::cin >> n;
108     for (int i = 1; i <= n; i++) {
109         std::cin >> a[i];
110     }
111
112     for (int i = 2, p; i <= n; i++) {
113         std::cin >> p;
114         edge[p].push_back(i);
115     }
116
117     pre_dfs(1, 1, 0);
118     dfs(1, 1);
119

```

```

120     for (int i = 1; i <= n; i++) {
121         val[tin[i]] = a[i];
122     }
123     build(0, 0, n);
124
125     int root = 1;
126
127     auto anc_of = [&] (int u, int d) {
128         for (int j = 0; j < T; j++) {
129             if ((d >> j) & 1) {
130                 u = pa[u][j];
131             }
132         }
133         return u;
134     };
135
136     auto is_child_of = [&] (int u, int v) -> bool {
137         return (tin[v] <= tin[u] and tout[u] <= tout[v]);
138     };
139
140     auto add_path = [&] (int u, int v, int k) {
141         while (top[u] != top[v]) {
142             if (depth[top[u]] < depth[top[v]]) std::swap(u, v);
143             range_add(0, 0, n, tin[top[u]], tin[u] + 1, k);
144             u = pa[top[u]][0];
145         }
146         if (depth[u] < depth[v]) std::swap(u, v);
147         range_add(0, 0, n, tin[v], tin[u] + 1, k);
148     };
149
150     auto add_subtree = [&] (int u, int k) {
151         if (root == u) {
152             range_add(0, 0, n, tin[1], tout[1], +k);
153         } else if (is_child_of(root, u)) {
154             int v = anc_of(root, depth[root] - depth[u] - 1);
155             range_add(0, 0, n, tin[1], tout[1], +k);
156             range_add(0, 0, n, tin[v], tout[v], -k);
157         } else {
158             range_add(0, 0, n, tin[u], tout[u], +k);
159         }
160     };
161
162     auto query_path = [&] (int u, int v) {
163         int64 res = 0ll;
164         while (top[u] != top[v]) {
165             if (depth[top[u]] < depth[top[v]]) std::swap(u, v);
166             res += range_sum(0, 0, n, tin[top[u]], tin[u] + 1);
167             u = pa[top[u]][0];
168         }
169         if (depth[u] < depth[v]) std::swap(u, v);
170         res += range_sum(0, 0, n, tin[v], tin[u] + 1);
171         return res;
172     };
173
174     auto query_subtree = [&] (int u) {
175         if (root == u) {
176             return range_sum(0, 0, n, tin[1], tout[1]);
177         } else if (is_child_of(root, u)) {
178             int v = anc_of(root, depth[root] - depth[u] - 1);
179             return range_sum(0, 0, n, tin[1], tout[1])
180                 - range_sum(0, 0, n, tin[v], tout[v]);
181         } else {
182             return range_sum(0, 0, n, tin[u], tout[u]);
183         }
184     };
185
186     int m;
187     std::cin >> m;
188     while (m--) {
189         int op, u, v, k;
190         std::cin >> op;
191         switch (op) {
192             case 1:

```

```

193         std::cin >> u;
194         root = u;
195         break;
196     case 2:
197         std::cin >> u >> v >> k;
198         add_path(u, v, k);
199         break;
200     case 3:
201         std::cin >> u >> k;
202         add_subtree(u, k);
203         break;
204     case 4:
205         std::cin >> u >> v;
206         std::cout << query_path(u, v) << '\n';
207         break;
208     case 5:
209         std::cin >> u;
210         std::cout << query_subtree(u) << '\n';
211         break;
212     }
213 }
214
215 return 0;
216 }

```

---

## 4.7 支配树 Dominator Tree

### ★ Dominator.cpp

---

```

1  #include <vector>
2  #include <numeric>
3  #include <iostream>
4  #include <functional>
5
6  std::vector<int> dominator(const std::vector<std::vector<int>> &g, int s) {
7      int n = (int) g.size();
8      std::vector<int> pos(n, -1), p, label(n), dom(n), sdom(n), dsu(n), par(n);
9      std::vector<std::vector<int>> rg(n), bucket(n);
10     std::function<void(int)> dfs = [&](int u) {
11         int t = (int) p.size();
12         p.push_back(u);
13         label[t] = sdom[t] = dsu[t] = pos[u] = t;
14         for (int v : g[u]) {
15             if (pos[v] == -1) {
16                 dfs(v);
17                 par[pos[v]] = t;
18             }
19             rg[pos[v]].push_back(t);
20         }
21     };
22     std::function<int(int, int)> find = [&](int u, int x) {
23         if (u == dsu[u]) {
24             return x ? -1 : u;
25         }
26         int v = find(dsu[u], x + 1);
27         if (v < 0) {
28             return u;
29         }
30         if (sdom[label[dsu[u]]] < sdom[label[u]]) {
31             label[u] = label[dsu[u]];
32         }
33         dsu[u] = v;
34         return x ? v : label[u];
35     };
36     dfs(s);
37     std::iota(dom.begin(), dom.end(), 0);
38     for (int i = (int) p.size() - 1; i >= 0; i -= 1) {
39         for (int j : rg[i]) {
40             sdom[i] = std::min(sdom[i], sdom[find(j, 0)]);
41         }
42         if (i) {

```

```

43         bucket[sdom[i]].push_back(i);
44     }
45     for (int k : bucket[i]) {
46         int j = find(k, 0);
47         dom[k] = sdom[j] == sdom[k] ? sdom[j] : j;
48     }
49     if (i > 1) {
50         dsu[i] = par[i];
51     }
52 }
53 for (int i = 1; i < (int) p.size(); i += 1) {
54     if (dom[i] != sdom[i]) {
55         dom[i] = dom[dom[i]];
56     }
57 }
58 std::vector<int> res(n, -1);
59 res[s] = s;
60 for (int i = 1; i < (int) p.size(); i += 1) {
61     res[p[i]] = p[dom[i]];
62 }
63 return res;
64 }
65
66 int main() {
67     std::cin.tie(nullptr)->sync_with_stdio(false);
68     int n, m;
69     std::cin >> n >> m;
70     std::vector<std::vector<int>>> g(n);
71     for (int i = 0, u, v; i < m; i += 1) {
72         std::cin >> u >> v;
73         g[u - 1].push_back(v - 1);
74     }
75     auto p = dominator(g, 0);
76     std::vector<std::vector<int>>> t(n);
77     for (int i = 1; i < n; i += 1) {
78         t[p[i]].push_back(i);
79     }
80     std::vector<int> ans(n, 1);
81     std::function<void(int)> dfs = [&](int u) {
82         for (int v : t[u]) {
83             dfs(v);
84             ans[u] += ans[v];
85         }
86     };
87     dfs(0);
88     for (int x : ans) {
89         std::cout << x << " ";
90     }
91 }

```

---

## 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

```
1 point rotate(point p, double a) {  
2     double cosa = std::cos(a), sina = std::sin(a);  
3     return point(p.x * cost - p.y * sint, p.x * sint + p.y * cost);  
4 }
```

#### 5.1.2 三维向量 vector 3d

##### ★ Vec3.cpp

```
1 vec3 cross(vec3 a, vec3 b) {  
2     return vec3(  
3         a.y * b.z - a.z * b.y,  
4         a.z * b.x - a.x * b.z,  
5         a.x * b.y - a.y * b.x  
6     );  
7 }  
8  
9 // rotate vec `v` around vec `k` by an angle `a`  
10 vec3 rotate3(vec3 v, vec3 k, double a) {  
11     k = unit(k);  
12     vec3 kv = cross(k, v);  
13     return v + std::sin(a) * kv + (1.0 - std::cos(a)) * cross(k, kv);  
14 }
```

## 5.2 Algorithms 算法

### 5.2.1 凸包构建 Andrew's Algorithm for Convex Hull

##### ★ ConvexHull.cpp

```
1 convex get_convex(convex p) {  
2     std::sort(p.begin(), p.end(), [&] (point a, point b) -> bool {  
3         return a.y != b.y ? a.y < b.y : a.x < b.x;  
4     });  
5     int n = (int) p.size(), tp = 0, lim = 1;  
6     std::vector<int> used(n, 0), t(2 * n);  
7     for (int i = 0; i < n; i++) {  
8         while (tp > lim and cross(p[t[tp - 1]] - p[t[tp - 2]], p[i] - p[t[tp - 2]]) <= 0) {  
9             used[t[-- tp]] = 0;  
10        }  
11        used[t[tp++] = i] = 1;  
12    }  
13    lim = std::max(lim, tp);  
14    for (int i = n - 2; i >= 0; i--) {  
15        if (used[i]) continue;  
16        while (tp > lim and cross(p[t[tp - 1]] - p[t[tp - 2]], p[i] - p[t[tp - 2]]) <= 0) {  
17            used[t[-- tp]] = 0;  
18        }  
19        used[t[tp++] = i] = 1;  
20    }  
21    while (cross(p[t[tp - 1]] - p[t[tp - 2]], p[t[0]] - p[t[tp - 2]]) <= 0) {  
22        used[t[-- tp]] = 0;  
23    }  
24    convex c(tp);  
25    for (int i = 0; i < tp; i++) {  
26        c[i] = p[t[i]];  
27    }  
28    return c;  
29 }
```

```

30
31
32 void reorder_convex(convex &c) {
33     int p = 0;
34     for (int i = 1; i < (int) c.size(); i++) {
35         if (c[i].y < c[p].y or (c[i].y == c[p].y and c[i].x < c[p].x)) {
36             p = i;
37         }
38     }
39     std::rotate(c.begin(), c.begin() + p, c.end());
40 }
41
42 convex minkowski_sum(convex c1, convex c2) {
43     auto prepare = [&] (convex &c) {
44         reorder_convex(c);
45         c.push_back(c[0]);
46         c.push_back(c[1]);
47     };
48     int n1 = (int) c1.size(), n2 = (int) c2.size();
49     prepare(c1);
50     prepare(c2);
51     convex c;
52     for (int i = 0, j = 0; i < n1 or j < n2; ) {
53         c.push_back(c1[i] + c2[j]);
54         auto value = cross(c1[i + 1] - c1[i], c2[j + 1] - c2[j]);
55         if (value >= 0 and i < n1) i++;
56         if (value <= 0 and j < n2) j++;
57     }
58     return c;
59 }
60
61 bool in_convex(point p, const convex &c) {
62     int lo = 0, hi = (int) c.size() - 1;
63     while (lo < hi) {
64         int mi = hi - (hi - lo) / 2;
65         if (cross(c[mi] - c[0], p - c[0]) >= 0) {
66             lo = mi;
67         } else {
68             hi = mi - 1;
69         }
70     }
71     if (hi == 0) {
72         return false;
73     } else if (hi == (int) c.size() - 1) {
74         return on_segment(p, c[0], c[hi]);
75     } else {
76         return in_triangle(p, c[0], c[hi], c[hi + 1]);
77     }
78 }

```

---

## 5.2.2 旋转卡壳 Rotating Calipers

### ★ RotatingCalipers.cpp

```

1 // required: P <- getConvexHull(P);
2 double findConvexHullWidth(const std::vector<point> &P) {
3     double res = inf;
4     int sz = P.size();
5     for (int i = 0, q = 1; i < sz; ++i) {
6         int j = (i + 1) % sz;
7         while (cross(P[j] - P[i], P[q] - P[i]) < cross(P[j] - P[i], P[(q + 1) % sz] - P[i])) {
8             q = (q + 1) % sz;
9         }
10        res = std::min(res, DistLinePoint(P[i], P[j], P[q]));
11    }
12    return res;
13 }

```

---



### 5.2.3 Delaunay 三角剖分 Triangulation

#### ★ Delaunay.cpp

---

```
1  typedef long long ll;
2
3  bool ge(const ll& a, const ll& b) { return a >= b; }
4  bool le(const ll& a, const ll& b) { return a <= b; }
5  bool eq(const ll& a, const ll& b) { return a == b; }
6  bool gt(const ll& a, const ll& b) { return a > b; }
7  bool lt(const ll& a, const ll& b) { return a < b; }
8  int sgn(const ll& a) { return a >= 0 ? a ? 1 : 0 : -1; }
9
10 struct pt {
11     ll x, y;
12     pt() { }
13     pt(ll _x, ll _y) : x(_x), y(_y) { }
14     pt operator-(const pt& p) const {
15         return pt(x - p.x, y - p.y);
16     }
17     ll cross(const pt& p) const {
18         return x * p.y - y * p.x;
19     }
20     ll cross(const pt& a, const pt& b) const {
21         return (a - *this).cross(b - *this);
22     }
23     ll dot(const pt& p) const {
24         return x * p.x + y * p.y;
25     }
26     ll dot(const pt& a, const pt& b) const {
27         return (a - *this).dot(b - *this);
28     }
29     ll sqrLength() const {
30         return this->dot(*this);
31     }
32     bool operator==(const pt& p) const {
33         return eq(x, p.x) && eq(y, p.y);
34     }
35 };
36
37 const pt inf_pt = pt(1e18, 1e18);
38
39 struct QuadEdge {
40     pt origin;
41     QuadEdge* rot = nullptr;
42     QuadEdge* onext = nullptr;
43     bool used = false;
44     QuadEdge* rev() const {
45         return rot->rot;
46     }
47     QuadEdge* lnext() const {
48         return rot->rev()->onext->rot;
49     }
50     QuadEdge* oprev() const {
51         return rot->onext->rot;
52     }
53     pt dest() const {
54         return rev()->origin;
55     }
56 };
57
58 QuadEdge* make_edge(pt from, pt to) {
59     QuadEdge* e1 = new QuadEdge;
60     QuadEdge* e2 = new QuadEdge;
61     QuadEdge* e3 = new QuadEdge;
62     QuadEdge* e4 = new QuadEdge;
63     e1->origin = from;
64     e2->origin = to;
65     e3->origin = e4->origin = inf_pt;
66     e1->rot = e3;
67     e2->rot = e4;
68     e3->rot = e2;
69     e4->rot = e1;
```

```

70     e1->onext = e1;
71     e2->onext = e2;
72     e3->onext = e4;
73     e4->onext = e3;
74     return e1;
75 }
76
77 void splice(QuadEdge* a, QuadEdge* b) {
78     swap(a->onext->rot->onext, b->onext->rot->onext);
79     swap(a->onext, b->onext);
80 }
81
82 void delete_edge(QuadEdge* e) {
83     splice(e, e->oprev());
84     splice(e->rev(), e->rev()->oprev());
85     delete e->rev()->rot;
86     delete e->rev();
87     delete e->rot;
88     delete e;
89 }
90
91 QuadEdge* connect(QuadEdge* a, QuadEdge* b) {
92     QuadEdge* e = make_edge(a->dest(), b->origin);
93     splice(e, a->lnext());
94     splice(e->rev(), b);
95     return e;
96 }
97
98 bool left_of(pt p, QuadEdge* e) {
99     return gt(p.cross(e->origin, e->dest()), 0);
100 }
101
102 bool right_of(pt p, QuadEdge* e) {
103     return lt(p.cross(e->origin, e->dest()), 0);
104 }
105
106 template <class T>
107 T det3(T a1, T a2, T a3, T b1, T b2, T b3, T c1, T c2, T c3) {
108     return a1 * (b2 * c3 - c2 * b3) - a2 * (b1 * c3 - c1 * b3) +
109         a3 * (b1 * c2 - c1 * b2);
110 }
111
112 bool in_circle(pt a, pt b, pt c, pt d) {
113     // If there is __int128, calculate directly.
114     // Otherwise, calculate angles.
115     #if defined(__LP64__) || defined(_WIN64)
116         __int128 det = 0;
117         det -= det3<__int128>(b.x, b.y, b.sqrLength(),
118                             c.x, c.y, c.sqrLength(), d.x, d.y, d.sqrLength());
119         det += det3<__int128>(a.x, a.y, a.sqrLength(),
120                             c.x, c.y, c.sqrLength(), d.x, d.y, d.sqrLength());
121         det -= det3<__int128>(a.x, a.y, a.sqrLength(),
122                             b.x, b.y, b.sqrLength(), d.x, d.y, d.sqrLength());
123         det += det3<__int128>(a.x, a.y, a.sqrLength(),
124                             b.x, b.y, b.sqrLength(), c.x, c.y, c.sqrLength());
125         return det > 0;
126     #else
127         auto ang = [](pt l, pt mid, pt r) {
128             ll x = mid.dot(l, r);
129             ll y = mid.cross(l, r);
130             long double res = atan2((long double)x, (long double)y);
131             return res;
132         };
133         long double kek = ang(a, b, c) + ang(c, d, a) - ang(b, c, d) - ang(d, a, b);
134         if (kek > 1e-8)
135             return true;
136         else
137             return false;
138     #endif
139 }
140
141 pair<QuadEdge*, QuadEdge*> build_tr(int l, int r, vector<pt>& p) {
142     if (r - l + 1 == 2) {

```

```

143     QuadEdge* res = make_edge(p[l], p[r]);
144     return make_pair(res, res->rev());
145 }
146 if (r - l + 1 == 3) {
147     QuadEdge *a = make_edge(p[l], p[l + 1]), *b = make_edge(p[l + 1], p[r]);
148     splice(a->rev(), b);
149     int sg = sgn(p[l].cross(p[l + 1], p[r]));
150     if (sg == 0) return make_pair(a, b->rev());
151     QuadEdge* c = connect(b, a);
152     if (sg == 1) return make_pair(a, b->rev());
153     else return make_pair(c->rev(), c);
154 }
155 int mid = (l + r) / 2;
156 QuadEdge *ldo, *ldi, *rdo, *rdi;
157 tie(ldo, ldi) = build_tr(l, mid, p);
158 tie(rdi, rdo) = build_tr(mid + 1, r, p);
159 while (true) {
160     if (left_of(rdi->origin, ldi)) {
161         ldi = ldi->lnext();
162         continue;
163     }
164     if (right_of(ldi->origin, rdi)) {
165         rdi = rdi->rev()->onext;
166         continue;
167     }
168     break;
169 }
170 QuadEdge* basel = connect(rdi->rev(), ldi);
171 auto valid = [&basel](QuadEdge* e) { return right_of(e->dest(), basel); };
172 if (ldi->origin == ldo->origin) ldo = basel->rev();
173 if (rdi->origin == rdo->origin) rdo = basel;
174 while (true) {
175     QuadEdge* lcand = basel->rev()->onext;
176     if (valid(lcand)) {
177         while (in_circle(basel->dest(), basel->origin, lcand->dest(),
178             lcand->onext->dest())) {
179             QuadEdge* t = lcand->onext;
180             delete_edge(lcand);
181             lcand = t;
182         }
183     }
184     QuadEdge* rcand = basel->oprev();
185     if (valid(rcand)) {
186         while (in_circle(basel->dest(), basel->origin, rcand->dest(),
187             rcand->oprev()->dest())) {
188             QuadEdge* t = rcand->oprev();
189             delete_edge(rcand);
190             rcand = t;
191         }
192     }
193     if (!valid(lcand) && !valid(rcand))
194         break;
195     if (!valid(lcand) ||
196         (valid(rcand) && in_circle(lcand->dest(), lcand->origin,
197             rcand->origin, rcand->dest())))
198         basel = connect(rcand, basel->rev());
199     else
200         basel = connect(basel->rev(), lcand->rev());
201 }
202 return make_pair(ldo, rdo);
203 }
204
205 vector<tuple<pt, pt, pt>> delaunay(vector<pt> p) {
206     sort(p.begin(), p.end(), [](const pt& a, const pt& b) {
207         return lt(a.x, b.x) || (eq(a.x, b.x) && lt(a.y, b.y));
208     });
209     auto res = build_tr(0, (int)p.size() - 1, p);
210     QuadEdge* e = res.first;
211     vector<QuadEdge*> edges = {e};
212     while (lt(e->onext->dest().cross(e->dest(), e->origin), 0)) e = e->onext;
213     auto add = [&p, &e, &edges]() {
214         QuadEdge* curr = e;
215         do {

```

```

216         curr->used = true;
217         p.push_back(curr->origin);
218         edges.push_back(curr->rev());
219         curr = curr->lnext();
220     } while (curr != e);
221 };
222 add();
223 p.clear();
224 int kek = 0;
225 while (kek < (int)edges.size()) {
226     if (!(e = edges[kek++])>used) add();
227 }
228 vector<tuple<pt, pt, pt>> ans;
229 for (int i = 0; i < (int)p.size(); i += 3) {
230     ans.push_back(make_tuple(p[i], p[i + 1], p[i + 2]));
231 }
232 return ans;
233 }

```

---

## 5.3 Formula/Notes 公式/笔记

### 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

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

#### ★ Circumcircle.cpp

```

1 // https://fanfansann.blog.csdn.net/article/details/108834399
2 Circle GetCircumcircle(Point p1, Point p2, Point p3) {
3     double Bx = p2.x-p1.x, By = p2.y-p1.y;
4     double Cx = p3.x-p1.x, Cy = p3.y-p1.y;
5     double D = 2*(Bx*Cy-By*Cx);
6     double ansx = (Cy*(Bx*Bx+By*By)-By*(Cx*Cx+Cy*Cy))/D + p1.x;
7     double ansy = (Bx*(Cx*Cx+Cy*Cy)-Cx*(Bx*Bx+By*By))/D + p1.y;
8     Point p(ansx, ansy);
9     return Circle(p, Length(p1-p));
10 }

```

---

## 6 Misc 杂项

### 6.1 C/C++ IO Cheat Sheet

#### 6.1.1 read()

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

#### 6.1.2 std::cin

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

### 6.2 Python Helper

记忆化装饰器 @lru\_cache(maxsize=128, typed=False)

```
1 @lru_cache(maxsize = None) # None 表示无限缓存
2 def fib(n):
3     if n < 2:
4         return n
5     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

```
1 // Detect if two integers have opposite signs:
2 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

```
1 for (int s = t; s; s = (s - 1) & m) {} // subset
2 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

---

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

---

## 6.4 .vimrc

---

```
1 set nu          "number
2 set ci          "cindent
3 set hls         "hlsearch
4 set ar          "autoread
5 set aw          "autowrite
6 set is          "incsearch
7 set et          "expandtab
8 set sm          "showmatch
9 set ai          "autoindent
10 set ic          "ignorecase
11 set cul         "cursorline
12 set cuc         "cursorcolumn
13 set nocp        "nocompatible
14 set noeb        "noerrorbells
15 set smarttab
16
17 set ts=4        "tabstop
18 set hi=1000     "history
19 set ch=2        "cmdheight
20 set so=3        "scrolloff
21 set bs=2        "backspace
22 set ls=2        "laststatus
23 set sw=4        "shiftwidth
24 set sts=4       "sosttabstop
25 set mouse=a
26 set completeopt=longest,menu
27 set statusline=%F%m%r%h%w\ [TYPE=%Y]\ [POS=%l,%v]\ %{strftime(\"%H:%M\")}
28
29 color ron
30 " color torte
31 nmap tt :%s/\t/ /g<CR>
32
33 map <F8> :call Rungdb()<CR>
34 func! Rungdb()
35 exec "w"
36 exec "!g++ % -g -o gdb_%< && gdb ./gdb_%<"
37 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