

# icpc 算法模板

Catch-22

2023 年 3 月 17 日

## 目录

<b>1</b>	<b>数学</b>	<b>4</b>
1.1	求逆元 . . . . .	4
1.2	扩展欧几里德算法 . . . . .	4
1.3	BSGS . . . . .	5
1.4	筛法 . . . . .	6
1.5	分解质因数 (预处理因子) . . . . .	8
1.6	组合数 . . . . .	10
1.7	卡特兰数 (catalan) . . . . .	11
1.8	容斥原理 . . . . .	12
1.9	数论分块 . . . . .	13
1.10	Möbius 反演 . . . . .	13
1.11	高斯消元 . . . . .	14
1.12	Miller Rabin 素数测试 . . . . .	15
1.13	FFT . . . . .	16
<b>2</b>	<b>数据结构</b>	<b>18</b>
2.1	(带权) 并查集 . . . . .	18
2.2	Sparse Table . . . . .	18
2.3	01Trie . . . . .	19
2.4	树状数组 . . . . .	20
2.5	线段树 . . . . .	21
2.6	可持久化线段树 . . . . .	25
2.7	线段树合并 . . . . .	29
2.8	DSU on tree . . . . .	32
2.9	树链剖分 . . . . .	34
2.10	珂朵莉 . . . . .	37
2.11	CDQ 分治 . . . . .	38
2.12	莫队 . . . . .	40
<b>3</b>	<b>图论</b>	<b>43</b>
3.1	spfa . . . . .	43
3.2	dijkstra . . . . .	44
3.3	最小生成树 . . . . .	46
3.4	kruskal 重构树 . . . . .	47
3.5	二分图匹配 . . . . .	49
3.6	强连通分量缩点 . . . . .	50
3.7	无向图的双连通分量 . . . . .	52
3.8	lca . . . . .	55
3.9	2-SAT . . . . .	56
3.10	基环树 . . . . .	57
3.11	dinic . . . . .	59

<b>4</b>	<b>动态规划</b>	<b>60</b>
4.1	背包 . . . . .	60
4.2	数位 dp . . . . .	62
4.3	换根 dp . . . . .	62
4.4	数据结构优化 dp . . . . .	64
4.5	斜率优化 dp . . . . .	64
<b>5</b>	<b>字符串</b>	<b>66</b>
5.1	字符串 Hash . . . . .	66
5.2	Trie . . . . .	68
5.3	KMP . . . . .	69
5.4	Z-algorithm . . . . .	69
5.5	AC 自动机 . . . . .	70
5.6	SA . . . . .	73
5.7	SAM . . . . .	75
5.8	Manacher . . . . .	77
<b>6</b>	<b>其他</b>	<b>78</b>
6.1	glibc 内置函数 . . . . .	78
6.2	__int128 读写 . . . . .	79
6.3	大整数运算 . . . . .	79
6.4	整数二分 . . . . .	82
6.5	单调栈 . . . . .	82
6.6	单调队列 . . . . .	83
6.7	矩阵快速幂 . . . . .	83
6.8	GospersHack . . . . .	84
6.9	C++17-STL . . . . .	84
6.10	pbds . . . . .	88
6.11	一些结论 . . . . .	89

# 1 数学

## 1.1 求逆元

注意考虑  $x$  是  $mod$  倍数的情况

```

1  ll qpow(ll a, ll b) {
2      ll res = 1;
3      while(b) {
4          if(b & 1) res = res * a % mod;
5          a = a * a % mod;
6          b >>= 1;
7      }
8      return res;
9  }
10
11 ll inv(ll x) { return qpow(x, mod - 2); }
12
13 const int N = 1e6 + 10;
14 // 线性递推求逆元 [1, n] 的所有数关于 p 的逆元
15 int inv[N];
16 void init_inv () {
17     int n, p;
18     cin >> n >> p;
19     inv[0] = 0, inv[1] = 1;
20     for (int i = 2; i <= n; i++)
21         inv[i] = (ll)(p - p / i) * inv[p % i] % p; // 为了保证大于零加了个 p
22
23     return 0;
24 }
```

## 1.2 扩展欧几里德算法

**bezout 定理**: 设  $a, b$  为正整数, 则关于  $x, y$  的方程  $ax + by = c$  有整数解当且仅当  $c$  是  $\gcd(a, b)$  的倍数。

返回结果:  $ax + by = \gcd(a, b)$  的一组解  $(x, y)$

时间复杂度:  $\mathcal{O}(n \log n)$

```

1  // 拓欧解线性同余方程  $a \cdot x \equiv b \pmod m$ 
2  #include <bits/stdc++.h>
3  using namespace std;
4  using ll = long long;
5  int a, b, m, n;
6
7  int exgcd(int a, int b, int &x, int &y) {
8      if(b == 0) {
9          x = 1, y = 0;
10         return a;
11     }
12     int d = exgcd(b, a % b, y, x);
```

```

13     y -= a/b * x;
14     return d;
15 }
16
17 int main() {
18     int x, y;
19     cin >> n;
20     while(n -- ) {
21         cin >> a >> b >> m;
22         int d = exgcd(a, m, x, y); // d = gcd(a, m)
23         if(b % d != 0) puts("impossible"); //bezout 定理: 有解的条件, gcd(a, m) | b
24         else printf("%lld\n", (1ll)x * (b/d) % m);
25     }
26     return 0;
27 }

```

### 1.3 BSGS

find smallest non-negative  $x$  s.t.  $a^x = b \bmod p$ , or  $-1$  (assume  $0^0 = 1$ )

```

1 // find smallest non-negative x s.t. a^x = b mod p, or -1 (assume 0^0 = 1)
2 int babyStepGiantStep(int a, int b, int p) {
3     a %= p; b %= p;
4     if (p == 1 || b == 1) return 0;
5     int cnt = 0, t = 1;
6     for (int g = __gcd(a, p); g != 1; g = __gcd(a, p)) {
7         if (b % g) return -1;
8         p /= g;
9         ++cnt;
10        b /= g;
11        t = 1LL * t * (a / g) % p;
12        if (b == t) return cnt;
13    }
14    std::map<int, int> mp;
15    int m = ceil(sqrt(p));
16    int base = b;
17    for (int i = 0; i != m; ++i) {
18        mp[base] = i;
19        base = 1LL * base * a % p;
20    }
21    base = powMod(a, m, p);
22    for (int i = 1; i <= m; ++i) {
23        t = 1LL * t * base % p;
24        if (mp.count(t))
25            return (1LL * i * m - mp[t]) % p + cnt;
26    }
27    return -1;
28 }
29 // https://www.luogu.com.cn/problem/P4195

```

## 1.4 筛法

### 筛质数

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4  const int N = 1e7 + 10;
5  // minp[i] 为 i 的最小素因子 http://oj.daimayuan.top/course/10/problem/733
6  int primes[N], pcnt, minp[N]; // 可用于 Log 级别分解质因数
7  bool vis[N]; //合数 true
8  int n, q;
9  //Linear
10 void get_prime(int n) {
11     for(int i = 2; i <= n; i++) {
12         if(!vis[i]) primes[ ++ pcnt] = i, minp[i] = i;
13         for(int j = 1; j <= pcnt && i * primes[j] <= n; ++ j) {
14             vis[i * primes[j]] = 1;
15             minp[primes[j] * i] = primes[j];
16             if(i % primes[j] == 0) break;
17         }
18     }
19 }
20
21 //about Linear :O(nloglogn)
22 bool isprime[N];
23 inline void getprime(int n) {
24     for (int i = 2; i <= n; i++) isprime[i] = 1;
25     for (int i = 2; i <= n; i++) {
26         if(isprime[i]) {
27             primes[++pcnt] = i;
28             if((ll)i*i<=n)
29                 for (int j = i * i; j <= n; j+=i){
30                     isprime[j] = 0;
31                 }
32         }
33     }
34 }

```

### 筛欧拉函数

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  /*phi compute
5  根据给定 n 计算 phi(n) O(aqrt(n))
6  核心公式 phi(n) = n*(1-1/p1)*(1 - 1/p2)*...
7  */
8  int get_phi(int n) {
9      int res = n;
10     for (int i = 2; i <= n / i; i++) {

```

```

11         if(n % i == 0) {
12             res = res / i * (i - 1); // res *= (1 - 1/n)
13             while(n % i == 0)    n /= i;
14         }
15     }
16     if(n > 1) res = res / n * (n - 1);
17     return res;
18 }
19
20 using ll = long long;
21 const int N = 1e6 + 10;
22
23 int phi[N], prime[N];
24 bool vis[N]; //合数 true
25
26 void sel_phi(int n) {
27     int cnt = 0;
28     phi[1] = 1;
29     for (int i = 2; i <= n; i++) {
30         if(!vis[i]) {
31             prime[cnt++] = i;
32             phi[i] = i - 1;
33         }
34         for (int j = 0; prime[j] <= n / i; j++) {
35             vis[prime[j] * i] = true;
36             if(i % prime[j] == 0) {
37                 phi[i * prime[j]] = phi[i] * prime[j];
38                 break;
39             }
40             else
41                 phi[prime[j] * i] = phi[i] * (prime[j] - 1);
42         }
43     }
44 }
45
46 /*
47 简短的  $n\log(n)$  时间求  $\phi[]$ 
48 vector<int> phi(n + 1);
49 iota(phi.begin(), phi.end(), 0);
50 for (int i = 1; i <= n; i++) {
51     for (int j = i; j <= n; j += i) {
52         phi[j] -= phi[i];
53     }
54 }
55 */

```

筛莫比乌斯函数

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  const int N = 50010;

```

```

4  int mu[N], p[N]; // p 为素数数组
5  bool flg[N];
6  void init() {
7      int tot = 0; mu[1] = 1;
8      for (int i = 2; i < N; ++i) {
9          if (!flg[i]) {
10             p[++tot] = i;
11             mu[i] = -1;
12         }
13         for (int j = 1; j <= tot && i * p[j] < N; ++j) {
14             flg[i * p[j]] = 1;
15             if (i % p[j] == 0) {
16                 mu[i * p[j]] = 0;
17                 break;
18             }
19             mu[i * p[j]] = -mu[i];
20         }
21     }
22     // 常用 mu 前缀和
23     // for (int i = 1; i <= N; ++i) mu[i] += mu[i - 1];
24 }

```

## 1.5 分解质因数（预处理因子）

```

1  //acwing 867
2  #include <bits/stdc++.h>
3  using namespace std;
4
5  const int N = 1e6 + 10;
6  vector<pair<int, int>> pfac[N];
7  //预处理质因数（当需要经常 1~N 所有数的质因数时候）
8  void init() {
9      for (int i = 2; i < N; i++) {
10         if(pfac[i].size()) continue;
11         for (int j = i; j < N; j += i) {
12             int t = j, cnt = 0;
13             while(t%i == 0) t/= i, cnt++;
14             pfac[j].push_back({i, cnt});
15         }
16     }
17 }
18
19 //根号 n 算法（n 为数值大小）
20
21
22
23 //Log(n) 算法 利用最小质因数筛
24 //先用欧拉筛筛出一定数值内的所有数字的质因子
25 //同时维护一个 minp[i] 表示 i 的最小质因子

```



```

26 //用的时机：一般求  $n$  个数每个数的质因子
27 //          且数值域大于预处理质因子所能承受范围  $>1e5$ ,
28 //          而又小于线性筛所能承受范围  $<1e8$ 
29 // http://oj.daimayuan.top/course/10/problem/733 例题（需要稍加改进）
30 int primes[N], minp[N], vis[N];
31 void get_prime(int n) {
32     int pcnt = 0;
33     for(int i = 2; i <= n; i++) {
34         if(!vis[i]) primes[ ++ pcnt] = i, minp[i] = i;
35         for(int j = 1; j <= pcnt && i * primes[j] <= n; ++ j) {
36             vis[i * primes[j]] = 1;
37             minp[primes[j] * i] = primes[j];
38             if(i % primes[j] == 0) break;
39         }
40     }
41 }
42
43 void decompose_log(int n) {
44     int cnt = 0;
45     while (n > 1) {
46         int t = minp[n];
47         int cnt = 0;
48         while (n % t == 0) {
49             cnt++;
50             n /= t;
51         }
52         printf("%d %d\n", t, cnt);
53     }
54 }
55
56 vector<int> divisors[N + 10];
57 // 预处理因子:
58 void init() {
59     //  $N$   $1e5$  左右
60     for (int i = 1; i <= N; i++) {
61         for (int j = 1; j <= i / j; j++) {
62             if(i % j == 0) {
63                 divisors[i].push_back(j);
64                 if(j * j != i) { //  $j*j$  一定不会爆  $int$ , 因为时间复杂度为  $\sqrt{n}$ 
65                     divisors[i].push_back(i / j);
66                 }
67             }
68         }
69     }
70 }
71
72 int main() {
73     int n; cin >> n;
74     while(n --) {

```

```

75         int x;
76         cin >> x;
77         decompose(x);
78     }
79     return 0;
80 }

```

## 1.6 组合数

1.  $C_n^m = C_n^{n-m}$
2.  $C_n^m = C_{n-1}^m + C_{n-1}^{m-1}$
3.  $C_n^0 + C_n^1 + \cdots + C_n^n = 2^n$
4. *lucas*:  $C_n^m \equiv C_{n \bmod p}^{m \bmod p} * C_{n/p}^{m/p}$

多重集组合数:

设  $S = \{n_1 \cdot a_1, n_2 \cdot a_2, \cdots, n_k \cdot a_k\}$  是一个由  $n_1$  个  $a_1, n_2$  个  $a_2, \cdots, n_k$  个  $a_k$  组成的多重集。设  $n = \sum_{i=1}^k n_i$ , 对于任意整数  $r \leq n$ , 从  $S$  中取出  $r$  个元素组成一个多重集 (不考虑顺序), 产生的不同多重集的数量为:

$$C_{k+r-1}^{k-1} - \sum_{i=1}^k C_{k+r-n_i-2}^{k-1} + \sum_{1 \leq i < j \leq k} C_{k+r-n_i-n_j-3}^{k-1} - \cdots + (-1)^k C_{k+r-\sum_{i=1}^k n_i-(k+1)}^{k-1}$$

多重集排列数:

多重集  $S = \{n_1 \cdot a_1, n_2 \cdot a_2, \cdots, n_k \cdot a_k\}$  生成的排列是  $\frac{(\sum_{i=1}^k n_i)!}{n_1! \cdot n_2! \cdots n_k!}$

```

1  //求组合数的几种方法
2  //不确定的时候都开 long long
3  #include <bits/stdc++.h>
4  using namespace std;
5  using ll = long long;
6  const int mod = 1e9 + 7, N = 1e6 + 10;
7  //C(a, b) a 上 b 下
8
9  /*1. 依照定义 适用于 a, b 很小的时候 (几十) */
10 int C(ll a, int b) /* a 上 b 下 */{
11     if(a < b) return 0;
12     int up = 1, down = 1;
13     for (ll i = a; i > a - b; i -- ) up = i % mod * up % mod; //up *= i
14     for (int j = 1; j <= b; j ++ ) down = (ll)j * down % mod; // down *= j
15     return (ll)up * qpow(down, mod - 2) % mod; // (up/down)
16 }
17
18 /*2. 递推 杨辉三角 a, b 在 2000 这个数量级 */
19 //O(N^2) 1e6~1e7
20 void init() {
21     for (int i = 0; i < N; i ++ )
22         for (int j = 0; j <= i; j ++ )

```

```

23         if(!j) C[i][j] = 1;
24         else C[i][j] = (C[i - 1][j] + C[i - 1][j - 1]) % mod;
25     }
26
27     //最常用
28     /*3. 预处理 fac[], invfac[]*/
29     /**
30      * //调用 :
31      * 1ll * fac[b] * invfac[a] % mod * invfac[b - a] % mod;
32      */
33     // O(N) 1e6 左右 看 N 大小
34     int fac[N], invfac[N];
35     void init() {
36         fac[0] = 1;
37         for (int i = 1; i < N; i++) fac[i] = (1ll)fac[i - 1]*i% mod;
38         invfac[N - 1] = qpow(fac[N - 1], mod - 2);
39         for (int i = N - 2; i >= 0; i--)
40             invfac[i] = (1ll)invfac[i + 1] * (i + 1) % mod;
41     }
42
43     /*4. Lucas 定理 当 a, b 的值特别大 如 1e9 以上...1e18 等 */
44     int C(int a, int b) {
45         int res = 1;
46         for (int i = 1, j = a; i <= b; i++, j--) {
47             res = (1ll)res * j % p;
48             res = (1ll)res * binpow(i, p - 2) % p;
49         }
50         return res;
51     }
52
53     ll lucas(ll a, ll b) { //p 为质 (模) 数
54         if(a < p && b < p) return C(a, b);
55         return (1ll)C(a % p, b % p) * lucas(a / p, b / p) % p;
56     }

```

## 1.7 卡特兰数 (catalan)

- 有  $2n$  个人排成一行进入剧场。入场费 5 元。其中只有  $n$  个人有一张 5 元钞票，另外  $n$  人只有 10 元钞票，剧院无其它钞票，问有多少种方法使得只要有 10 元的人买票，售票处就有 5 元的钞票找零？
- 一位大城市的律师在她住所以北  $n$  个街区和以东  $n$  个街区处工作。每天她走  $2n$  个街区去上班。如果他从不穿越（但可以碰到）从家到办公室的对角线，那么有多少条可能的道路？
- 在圆上选择  $2n$  个点，将这些点成对连接起来使得所得到的  $n$  条线段不相交的方法数？
- 对角线不相交的情况下，将一个凸多边形区域分成三角形区域的方法数？
- 一个栈（无穷大）的进栈序列为  $1, 2, 3, \dots, n$  有多少个不同的出栈序列？
- $n$  个结点可构造多少个不同的二叉树？

- $n$  个  $+1$  和  $n$  个  $-1$  构成  $2n$  项  $a_1, a_2, \dots, a_{2n}$ , 其部分和满足  $a_1 + a_2 + \dots + a_k \geq 0 (k = 1, 2, 3, \dots, 2n)$  对与  $n$  该数列为?

$$catalan(n) = C_{2n}^n - C_{2n}^{n-1}$$

$$catalan(n) = \frac{C_{2n}^n}{n+1}$$

前几项 {1, 2, 5, 14, 42, 132 ... }

```

1 // fac[N] 记得开两倍
2
3 void init() { // 预处理阶乘及其逆元
4     fac[0] = 1;
5     for (int i = 1; i < N; i++) fac[i] = (ll)fac[i - 1] * i % mod;
6     ifac[N - 1] = qpow(fac[N - 1], mod - 2);
7     for (int i = N - 1; i; i--) ifac[i - 1] = (ll)ifac[i] * i % mod;
8 }
9
10 ll C(int n, int m) { // 组合数 C(n,m)
11     return fac[n] * ifac[m] % mod * ifac[n - m] % mod;
12 }
13
14 ll ans = C(2 * n, n) * qpow(n + 1, mod - 2) % mod;

```

## 1.8 容斥原理

$S_i$  为有限集,  $|S|$  为  $S$  的大小 (元素个数), 则:

$$|\bigcup_{i=1}^n S_i| = \sum_{i=1}^n |S_i| - \sum_{1 \leq i < j \leq n} |S_i \cap S_j| + \sum_{1 \leq i < j < k \leq n} |S_i \cap S_j \cap S_k| + \dots + (-1)^{n+1} |S_1 \cap \dots \cap S_n|$$

```

1 // 容斥原理
2 // 给定素数集合 A(大小为 k), 求 [L, R] 中素数集合的任意元素的倍数的个数
3 // 1<=L<=R<=10^18, 1<=k<=20, 2<=ai<=100
4 #include <bits/stdc++.h>
5 using ll = long long;
6 using namespace std;
7
8 int main() {
9     ll l, r, k, f[25];
10    cin >> l >> r >> k;
11    for (int i = 0; i < k; i++) cin >> f[i];
12
13    ll ans = 0;
14
15    for (int i = 1; i < 1 << k; i++) { // 枚举集合中全部的非空子集
16        ll cnt = 0, a = r, b = l - 1; // cnt 用来表示所取的数的个数
17        for (int j = 0; j < k; j++) {
18            if(i >> j & 1) {

```

```

19         cnt++;
20         a /= f[j], b /= f[j];
21     }
22 }
23 if(cnt & 1) ans += (a - b);
24 else ans -= (a - b);
25 }
26 cout << ans << endl;
27 return 0;
28 }

```

## 1.9 数论分块

考虑和式:  $\sum_{i=1}^n f(i) \lfloor \frac{n}{i} \rfloor$ , 由于  $\lfloor \frac{n}{i} \rfloor$  的值成一个块状分布, 故可以一块一块运算。我们先求出  $f(i)$  的前缀和, 每次以  $[l, r] = [l, \lfloor \frac{n}{\lfloor \frac{n}{l} \rfloor} \rfloor]$  为一块分块求出贡献累加到结果中。(常配合莫反使用) 常见转换:

- $\lceil \frac{a}{b} \rceil = \lfloor \frac{a-1}{b} \rfloor + 1$
- $a \bmod b = a - \lfloor \frac{a}{b} \rfloor * b$

```

1 // for(int i = st; i <= ed; i++) ans += num/i
2 ll block(ll st, ll ed, ll num) {
3     //sum(num/i i in [st,ed])
4     ll L = 0, res = 0;
5     ed = min(ed, num);
6     for (ll i = st; i <= ed; i = L + 1) {
7         L = min(ed, num / (num / i)); //该区间的最后一个数
8         res += (L - i + 1) * (num / i); //区间 [i,L] 的 num/i 都是一个值
9         // res += (s(L) - s(i-1)) * (num/i); //s(i) 为 f(i) 前缀和
10    }
11    return res;
12 }

```

## 1.10 Möbius 反演

$\mu$  为莫比乌斯函数, 定义为

$$\mu(x) = \begin{cases} 1 & n = 1 \\ 0 & n \text{ 含有平方因子} \\ (-1)^k & k \text{ 为 } n \text{ 本质不同的质因子个数} \end{cases}$$

性质:

$$\sum_{d|n} \mu(d) = \begin{cases} 1 & n = 1 \\ 0 & n \neq 1 \end{cases}$$

证: 设  $n = \prod_{i=1}^k p_i^{c_i}, n' = \prod_{i=1}^k p_i$

那么  $\sum_{d|n} \mu(d) = \sum_{d|n'} \mu(d) = \sum_{i=0}^k C_k^i \cdot (-1)^i = (1 + (-1))^k = 1$

反演:

形式一:

$$f(n) = \sum_{d|n} g(d) \Leftrightarrow g(n) = \sum_{d|n} \mu(d) f\left(\frac{n}{d}\right)$$

证:

$$\sum_{d|n} \mu(d) f\left(\frac{n}{d}\right) = \sum_{d|n} \mu(d) \sum_{k|\frac{n}{d}} g(k) = \sum_{k|n} g(k) \sum_{d|\frac{n}{k}} \mu(d) = g(n)$$

用  $\sum_{d|n} g(d)$  来替换  $f(\frac{n}{d})$ , 再变换求和顺序。最后一步变换的依据:  $\sum_{d|n} \mu(d) = [n=1]$ , 因此在  $\frac{n}{k}=1$  时第二个和式的值才为 1。此时  $n=k$ , 故原式等价于  $\sum_{k|n} [n=k] \cdot g(k) = g(n)$

形式二:

$$f(n) = \sum_{n|d} g(d) \Leftrightarrow g(n) = \sum_{n|d} \mu\left(\frac{d}{n}\right) f(d)$$

## 1.11 高斯消元

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  const int N = 110;
4  const double eps = 1e-6;
5  int n;
6  double a[N][N];
7
8  int gauss() {
9      int c, r;
10     for(c = 0, r = 0; c < n; c++) {
11         int t = r;
12         for(int i = r; i < n; i++) //找到首元素最大
13             if(fabs(a[i][c]) > fabs(a[t][c]))
14                 t = i;
15
16         if(fabs(a[t][c]) < eps) continue;
17
18         for(int i = c; i <= n; i++) swap(a[t][i], a[r][i]);
19         for(int i = n; i >= c; i--) a[r][i] /= a[r][c];
20         for(int i = r + 1; i < n; i++)
21             if(fabs(a[i][c]) > eps)
22                 for(int j = n; j >= c; j--)
23                     a[i][j] -= a[r][j] * a[i][c];
24         r++;
25     }
26     if(r < n) {
27         for(int i = r; i < n; i++)
28             if(fabs(a[i][n]) > eps)
29                 return 2;
30         return 1;
31     }
32
33     for(int i = n - 1; i >= 0; i--)

```

```

34         for(int j = i + 1; j < n; j++)
35             a[i][n] -= a[i][j] * a[j][n];
36
37     return 0; //有唯一解
38 }
39
40 int main() {
41     cin >> n;
42     for(int i = 0; i < n; i++)
43         for(int j = 0; j < n + 1; j++)
44             cin >> a[i][j];
45
46     int t = gauss();
47     if(t == 0)
48         for(int i = 0; i < n; i++) printf("%.2f\n", a[i][n]);
49     else if(t == 1)
50         puts("Infinite group solutions");
51     else puts("No solution");
52
53     return 0;
54 }

```

## 1.12 Miller Rabin 素数测试

```

1  //loj143 prime test
2  #include <bits/stdc++.h>
3  using namespace std;
4  using ull = unsigned long long;
5  using ll = long long;
6  /* O(sqrt(n))
7  bool is_prime(ll x)
8  {
9      if(x < 2) return false;
10     for(ll i = 2; i <= x / i; ++i)
11         if(x % i == 0) return false;
12     return true;
13 }
14 */
15 //常常是大素数测试，要用到 int128
16 inline ll qmul(ll a, ll b, ll p) { return (ll)((__int128)a * b % p); }
17 ll qpow(ll a, ll b, ll p) {
18     ll res = 1;
19     while(b) {
20         if(b & 1) res = qmul(res, a, p);
21         a = qmul(a, a, p);
22         b >>= 1;
23     }
24     return res;
25 }

```

```

26  const int test_time = 8;
27
28  bool mr_test(ll n) {
29      if(n < 3 || n % 2 == 0) return n == 2;
30      ll a = n - 1, b = 0;
31      while(a % 2 == 0) a /= 2, ++b;
32
33      for (int i = 1, j; i <= test_time; ++i) {
34          ll x = rand() % (n - 2) + 2, v = qpow(x, a, n);
35          if(v == 1) continue;
36          for (j = 0; j < b; ++j) {
37              if(v == n - 1) break;
38              v = qmul(v, v, n);
39          }
40          if(j >= b) return 0;
41      }
42      return 1;
43  }
44
45  int main() {
46      srand(time(0));
47      ll x;
48      while(cin >> x) {
49          if(mr_test(x)) puts("Y");
50          else puts("N");
51      }
52      return 0;
53  }

```

### 1.13 FFT

```

1  #include <bits/stdc++.h>
2  #include <any>
3  #define rep(i, a, b) for (int i = (a); i <= (b); i++)
4  using namespace std;
5
6  namespace FFT {
7      const double PI = acos(-1);
8      using C = complex<double>;
9      vector<int> rev;
10     vector<C> roots{C(0, 0), C(1, 0)};
11     void dft(vector<C>& a) {
12         int n = (int)a.size();
13         if ((int)rev.size() != n) {
14             int k = __builtin_ctz(n) - 1;
15             rev.resize(n);
16             for (int i = 0; i < n; ++i) {
17                 rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
18             }

```



```

19     }
20     if ((int)roots.size() < n) {
21         int k = __builtin_ctz(roots.size());
22         roots.resize(n);
23         while ((1 << k) < n) {
24             C e = polar(1.0, PI / (1 << k));
25             for (int i = 1 << (k - 1); i < (1 << k); ++i) {
26                 roots[2 * i] = roots[i];
27                 roots[2 * i + 1] = roots[i] * e;
28             }
29             ++k;
30         }
31     }
32     for (int i = 0; i < n; ++i) if (rev[i] < i) {
33         swap(a[i], a[rev[i]]);
34     }
35     for (int k = 1; k < n; k *= 2) {
36         for (int i = 0; i < n; i += 2 * k) {
37             for (int j = 0; j < k; ++j) {
38                 auto u = a[i + j], v = a[i + j + k] * roots[k + j];
39                 a[i + j] = u + v;
40                 a[i + j + k] = u - v;
41             }
42         }
43     }
44 }
45 void idft(vector<C>& a) {
46     int n = (int)a.size();
47     reverse(a.begin() + 1, a.end());
48     dft(a);
49     for (auto& x : a) x /= n;
50 }
51 } // namespace FFT
52
53
54 vector<int> mul(const vector<int> &A, const vector<int> &B) {
55     int n = max(A.size(), B.size()), tot = max(1, n * 2 - 1);
56     int sz = 1 << __lg(tot * 2 - 1);
57     vector<complex<double>> C(sz);
58     for (int i = 0; i < A.size(); ++i) C[i].real(A[i]);
59     for (int i = 0; i < B.size(); ++i) C[i].imag(B[i]);
60     FFT::dft(C);
61     for (auto &x : C) x *= x;
62     FFT::idft(C);
63     vector<int> ans(A.size() + B.size() - 1);
64     for (int i = 0; i < ans.size(); ++i) ans[i] = int(C[i].imag() / 2 + 0.2);
65     return ans;
66 }
67

```

```

68 int main() {
69
70     cin.tie(nullptr)->sync_with_stdio(false);
71     int n, m;
72     cin >> n >> m;
73     vector<int> a(n + 1), b(m + 1);
74     for (auto &x : a) cin >> x;
75     for (auto &x : b) cin >> x;
76     auto c = mul(a, b);
77     for (auto &x : c) cout << x << ' ';
78     cout << '\n';
79     std::any a = 34;
80     return 0;
81
82 }

```

## 2 数据结构

### 2.1 (带权) 并查集

```

1 struct DSU {
2     vector<int> f, siz;
3     DSU(int n) : f(n), siz(n, 1) { iota(f.begin(), f.end(), 0); }
4     int leader(int x) {
5         while (x != f[x]) x = f[x] = f[f[x]];
6         return x;
7     }
8     // int leader(int x) { // 带权并查集
9     //     if(x == f[x]) return x;
10    //     int rt = leader(f[x]); //这和下面一行顺序很重要
11    //     d[x] += d[f[x]]; //可以改成 d[x] ^= d[fa[x]], 根据权值意义的需要修改
12    //     return f[x] = rt;
13    // }
14    bool same(int x, int y) { return leader(x) == leader(y); }
15    bool merge(int x, int y) {
16        x = leader(x);
17        y = leader(y);
18        if (x == y) return false;
19        siz[x] += siz[y];
20        f[y] = x;
21        return true;
22    }
23    int size(int x) { return siz[leader(x)]; }
24 };

```

### 2.2 Sparse Table

时间复杂度  $\mathcal{O}(1)$ , 空间复杂度  $\mathcal{O}(n \log n)$

静态区间查询可重复贡献信息, 如“区间最值”、“区间接位和”、“区间接位或”、“区间 GCD”

```

1 //f[i][j] 表示左闭右开 [i, i + 2^j) 的最大值
2 template<class T,
3     class Cmp = std::less<T>>
4 struct RMQ {
5     const int n; // 从零开始
6     const Cmp cmp;
7     std::vector<std::vector<T>> a;
8     RMQ(const std::vector<T> &init) : n(init.size()), cmp(Cmp()) {
9         int lg = std::__lg(n);
10        a.assign(n, std::vector<T>(lg + 1));
11        for (int j = 0; j <= lg; j++) {
12            for (int i = 0; i + (1 << j) <= n; i++) {
13                a[i][j] = (j == 0 ? init[i] : std::min(a[i][j - 1], a[i + (1 << (j - 1))][j
14                    ↪ - 1], cmp));
15            }
16        }
17        // 左闭右开
18        T rangeMin(int l, int r) {
19            int k = std::__lg(r - l);
20            return std::min(a[l][k], a[r - (1 << k)][k], cmp);
21        }
22    };

```

### 2.3 01Trie

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int N = 1e5 + 10, M = N * 31;
4 int a[N];
5 int son[M][2], idx;
6
7 void insert(int x) {
8     int p = 0;
9     for (int i = 30; i >= 0; --i) {
10        int u = (x >> i) & 1;
11        if (!son[p][u]) son[p][u] = ++idx;
12        p = son[p][u];
13    }
14 }
15 // 集合内和 x 异或的最大值
16 int query(int x) {
17     int p = 0, res = 0;
18     for (int i = 30; i >= 0; --i) {
19        int u = (x >> i) & 1;
20        if (son[p][u ^ 1]) p = son[p][u ^ 1], res |= (1 << i);
21        else p = son[p][u];
22        // 集合内和 x 异或的最小值
23        // if(son[p][u]) p = son[p][u];

```

```

24         // else res /= (1 << i), p = son[p][u ^ 1];
25     }
26     return res;
27 }
28
29 int main() {
30     int n, res = 0;
31     cin >> n;
32     for(int i = 0; i < n; i++) cin >> a[i];
33     for(int i = 0; i < n; i++) {
34         insert(a[i]);
35         res = max(res, query(a[i]));
36     }
37     cout << res;
38     return 0;
39 }

```

## 2.4 树状数组

```

1  // fenwick-tree 写区间修改，区间查询
2  //记录两个数组 b[i] = a[i] - a[i - 1]; c[i] = i * b[i];
3  // a[1~x] = \sum_{i=1}^x \sum_{j=1}^i b[j]
4  // = \sum_{i=1}^x (x-i+1)*b[i]
5  // = (x+1)\sum_{i=1}^x b[i] - \sum_{i=1}^x i*b[i]
6
7  #include <bits/stdc++.h>
8  #define rep(i, a, b) for (int i = (a); i <= (b); i++)
9  using namespace std;
10 using ll = long long;
11
12 template<typename T/*, class OP = plus<T>*/>
13 struct fenwick {
14     int n;
15     // const OP op;
16     vector<T> c;
17     fenwick(int _n) : n(_n), c(_n + 1, 0)/*, op(OP())*/ {}
18     void add(int x, T v) {
19         for (int i = x; i <= n; i += i & -i) {
20             c[i] += v; // c[i] = op(c[i], v)
21         }
22     }
23     T sum(int x) {
24         T res{};
25         for (int i = x; i; i -= i & -i) {
26             res += c[i];
27         }
28         return res;
29     }
30 };

```

```

31
32 int main() {
33     ios::sync_with_stdio(false);
34     cin.tie(nullptr);
35     int n, m;
36     cin >> n >> m;
37     vector<int> a(n + 1);
38     fenwick<ll> t1(n), t2(n); //维护 b[i], b[i] * i 的前缀和
39     rep(i, 1, n) cin >> a[i];
40     rep(i, 1, n) {
41         int b = a[i] - a[i - 1];
42         t1.add(i, b);
43         t2.add(i, 1ll * b * i);
44     }
45     auto preSum = [&](int x) {
46         return t1.sum(x)*(x + 1) - t2.sum(x);
47     };
48     while(m --) {
49         int op, l, r, d;
50         cin >> op >> l >> r;
51         if(op == 2) {
52             cout << preSum(r) - preSum(l - 1) << '\n';
53         }
54         else {
55             cin >> d;
56             t1.add(l, d);
57             t2.add(l, 1ll * l * d);
58             t1.add(r + 1, -d);
59             t2.add(r + 1, 1ll * (r + 1) * -d);
60         }
61     }
62     return 0;
63 }

```

## 2.5 线段树

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4
5  struct Info {
6  };
7
8  Info operator+(const Info& a, const Info& b) {
9  }
10
11 struct Tag {
12 };
13

```

```

14 void apply(Info &a, Tag b) {
15 }
16
17 void apply(Tag &a, Tag b) {
18 }
19
20 template <typename Info, typename Tag,
21           typename Merge = plus<Info>>
22 struct SegmentTree {
23     const int n;
24     const Merge merge;
25     vector<Info> info;
26     vector<Tag> tag;
27     SegmentTree(int n) : n(n), merge(Merge()),
28         info(4 << __lg(n)), tag(4 << __lg(n)) {}
29     SegmentTree(vector<Info> init) : SegmentTree(init.size()) {
30         function<void(int, int, int)> build = [&](int p, int l, int r) {
31             if (r - l == 1) {
32                 info[p] = init[l];
33                 return;
34             }
35             int m = (l + r) / 2;
36             build(2 * p, l, m);
37             build(2 * p + 1, m, r);
38             pull(p);
39         };
40         build(1, 0, n);
41     }
42     void pull(int p) {
43         info[p] = merge(info[2 * p], info[2 * p + 1]);
44     }
45     void apply(int p, const Tag &v) {
46         ::apply(info[p], v);
47         ::apply(tag[p], v);
48     }
49     void push(int p) {
50         apply(2 * p, tag[p]);
51         apply(2 * p + 1, tag[p]);
52         tag[p] = Tag();
53     }
54     void modify(int p, int l, int r, int x, const Info &v) {
55         if (r - l == 1) {
56             info[p] = v;
57             return;
58         }
59         int m = (l + r) / 2;
60         push(p);
61         if (x < m) {
62             modify(2 * p, l, m, x, v);

```

```

63         } else {
64             modify(2 * p + 1, m, r, x, v);
65         }
66         pull(p);
67     }
68     void modify(int p, const Info &v) {
69         modify(1, 0, n, p, v);
70     }
71     Info rangeQuery(int p, int l, int r, int x, int y) {
72         if (l >= y || r <= x) {
73             return Info();
74         }
75         if (l >= x && r <= y) {
76             return info[p];
77         }
78         int m = (l + r) / 2;
79         push(p);
80         return merge(rangeQuery(2 * p, l, m, x, y), rangeQuery(2 * p + 1, m, r, x, y));
81     }
82     Info rangeQuery(int l, int r) {
83         return rangeQuery(1, 0, n, l, r);
84     }
85     void rangeApply(int p, int l, int r, int x, int y, const Tag &v) {
86         if (l >= y || r <= x) {
87             return;
88         }
89         if (l >= x && r <= y) {
90             apply(p, v);
91             return;
92         }
93         int m = (l + r) / 2;
94         push(p);
95         rangeApply(2 * p, l, m, x, y, v);
96         rangeApply(2 * p + 1, m, r, x, y, v);
97         pull(p);
98     }
99     void rangeApply(int l, int r, const Tag &v) {
100         return rangeApply(1, 0, n, l, r, v);
101     }
102 };
103
104 int main() {
105     return 0;
106 }

```

扫描线: (面积)

```

1 //p1502 线段树扫描线算法
2 #include<bits/stdc++.h>
3 using namespace std;
4 using ll = long long;

```

```

5  const ll N = 1e4 + 10;
6  struct L {
7      ll x, y1, y2;
8      ll c;
9      //当左矩形的右边界与右矩形的左边界重合时, 该线上的点应属于能被两个窗户都能看见的状态所以先加
10     bool operator<(const L &rhs) const { return x == rhs.x ? c < rhs.c : x < rhs.x; }
11 }line[2 * N];
12
13 ll n, w, h, m;
14 ll b[2 * N]; //离散化前的 y 轴
15
16 struct node {
17     ll l, r;
18     ll maxv, add;
19 } t[8 * N];
20
21
22 void pushdown(ll p) {
23     node &root = t[p], &n1 = t[p << 1], &nr = t[p << 1 | 1];
24     if(root.add) {
25         n1.add += root.add, n1.maxv += root.add;
26         nr.add += root.add, nr.maxv += root.add;
27         root.add = 0;
28     }
29 }
30
31 void pushup(ll p) {
32     t[p].maxv = max(t[p << 1].maxv, t[p << 1 | 1].maxv);
33 }
34
35 void modify(ll p, ll l, ll r, ll c) {
36     if(t[p].l >= l && t[p].r <= r) {
37         t[p].maxv += c;
38         t[p].add += c;
39         return;
40     }
41     pushdown(p);
42     ll mid = t[p].l + t[p].r >> 1;
43     if(l <= mid) modify(p << 1, l, r, c);
44     if(r > mid) modify(p << 1 | 1, l, r, c);
45     pushup(p);
46
47 }
48
49 void build(ll p, ll l, ll r) {
50     if(l == r) {
51         t[p] = {l, r, 0, 0};
52         return;
53     }

```



```

54     t[p].l = l, t[p].r = r;
55     ll mid = l + r >> 1;
56     build(p << 1, l, mid);
57     build(p << 1 | 1, mid + 1, r);
58     //pushup(p); //初始化都是 0 不用 pushup()
59 }
60
61 int main() {
62     ll T;
63     scanf("%lld", &T);
64     while( T -- ) {
65         memset(line, 0, sizeof(line));
66         memset(b, 0, sizeof(b));
67         memset(t, 0, sizeof(t));
68
69         scanf("%lld%lld%lld", &n, &w, &h);
70         for (ll i = 1, j = 0; i <= n; i++) {
71             ll x, y, l;
72             scanf("%lld%lld%lld", &x, &y, &l);
73             line[i] = {x, y, y + h - 1, l};
74             line[i + n] = {x + w - 1, y, y + h - 1, -l};
75             b[ ++ j] = y;
76             b[ ++ j] = y + h - 1;
77         }
78         n <<= 1;
79         sort(b + 1, b + 1 + n);
80         m = unique(b + 1, b + 1 + n) - b - 1; //unique 得到 end() 迭代器
81         sort(line + 1, line + 1 + n);
82
83         for (ll i = 1; i <= n; i++) {
84             line[i].y1 = lower_bound(b + 1, b + m + 1, line[i].y1) - b - 1;
85             line[i].y2 = lower_bound(b + 1, b + m + 1, line[i].y2) - b - 1;
86         }
87         build(1, 1, m - 1);
88
89         ll res = 0;
90         for (ll i = 1; i <= n; i++) {
91             modify(1, line[i].y1, line[i].y2, line[i].c);
92             res = max(res, t[1].maxv);
93         }
94         printf("%d\n", res);
95     }
96     return 0;
97 }

```

## 2.6 可持久化线段树

```

1 //Luogu 3824 kth-number
2 #include <bits/stdc++.h>

```

```

3  using namespace std;
4  const int N = 2e5 + 10, M = (N << 2) + 17 * N;
5
6  struct node {
7      int l, r;
8      int cnt;
9  } t[M];
10 int idx, a[N];
11 vector<int> num;
12 int find(int x) { return lower_bound(num.begin(), num.end(), x) - num.begin(); }
13
14 int insert(int now, int l, int r, int x) {
15     int p = ++ idx;
16     t[p] = t[now];
17     if (l == r) {
18         t[p].cnt ++ ;
19         return p;
20     }
21     int mid = l + r >> 1;
22     if(x <= mid) t[p].l = insert(t[now].l, l, mid, x);
23     else t[p].r = insert(t[now].r, mid + 1, r, x);
24     t[p].cnt = t[t[p].l].cnt + t[t[p].r].cnt;
25
26     return p;
27 }
28
29 int build(int l, int r) {
30     int p = ++ idx;
31     if (l == r) return p;
32     int mid = l + r >> 1;
33     t[p].l = build(l, mid), t[p].r = build(mid + 1, r);
34     return p;
35 }
36
37 int query(int x, int y, int l, int r, int k) {
38     if(l == r) return l;
39     int cnt = t[t[y].l].cnt - t[t[x].l].cnt;
40     int mid = l + r >> 1;
41     if(k <= cnt) return query(t[x].l, t[y].l, l, mid, k);
42     else return query(t[x].r, t[y].r, mid + 1, r, k - cnt);
43 }
44
45 int n, m, root[N];
46
47 int main() {
48     scanf("%d%d", &n, &m);
49     for (int i = 1; i <= n; i ++ ) {
50         scanf("%d", &a[i]);
51         num.push_back(a[i]);

```

```

52     }
53
54     sort(num.begin(), num.end());
55     num.erase(unique(num.begin(), num.end()), num.end());
56
57     root[0] = build(0, num.size() - 1);
58
59     for (int i = 1; i <= n; i ++ )
60         root[i] = insert(root[i - 1], 0, num.size() - 1, find(a[i]));
61     while (m -- ) {
62         int l, r, k;
63         scanf("%d%d%d", &l, &r, &k);
64         printf("%d\n", num[query(root[l - 1], root[r], 0, num.size() - 1, k)]);
65     }
66
67     return 0;
68 }

```

1 // <http://oj.daimayuan.top/course/10/problem/464>  
2 // 在给定  $N$  长的数组  $\{A\}$  中进行  $Q$  次询问  $[Li, Ri]$  区间中不大于  $Hi$  的元素个数。  
3 //主席树的在线做法 还有树状数组的对询问离线做法  
4 #include <bits/stdc++.h>  
5 using namespace std;  
6 const int N = 1e5 + 10;  
7 int n, q, L[N], R[N], idx, H[N], rt[N], a[N];  
8 vector<int> alls;  
9  
10 struct node {  
11 int l, r, cnt;  
12 } t[N \* 75];  
13  
14 int insert(int now, int l, int r, int x) {  
15 int p = ++idx;  
16 t[p] = t[now];  
17 if(l == r) {  
18 t[p].cnt++;  
19 return p;  
20 }  
21 int mid = l + r >> 1;  
22 if(x <= mid) t[p].l = insert(t[now].l, l, mid, x);  
23 else t[p].r = insert(t[now].r, mid + 1, r, x);  
24 t[p].cnt = t[t[p].l].cnt + t[t[p].r].cnt;  
25 return p;  
26 }  
27  
28 int query(int version, int l, int r, int h) {  
29 if(l == r) return t[version].cnt;  
30  
31 if(r <= h) return t[version].cnt;  
32 int mid = l + r >> 1, res = 0;

```

33     res += query(t[version].l, l, mid, h);
34     if(h > mid) res += query(t[version].r, mid + 1, r, h);
35     return res;
36 }
37
38 int build(int l, int r) {
39     int p = ++ idx;
40     if (l == r) return p;
41     int mid = l + r >> 1;
42     t[p].l = build(l, mid), t[p].r = build(mid + 1, r);
43     return p;
44 }
45
46 int find(int x) { return lower_bound(alls.begin(), alls.end(), x) - alls.begin(); }
47
48 void init() {
49     idx = 0;
50     memset(rt, 0, sizeof rt);
51     alls.clear();
52     memset(t, 0, sizeof t);
53 }
54
55 int main() {
56     int T; scanf("%d", &T);
57     while(T --) {
58         init();
59         scanf("%d%d", &n, &q);
60         for (int i = 1; i <= n; i++) {
61             scanf("%d", &a[i]);
62             alls.push_back(a[i]);
63         }
64         //虽然是在线做法, 但为了好处理  $h$  离散化后的值, 就将  $H_i$  也加入  $alls$ 
65         for (int i = 1; i <= q; i++) {
66             scanf("%d%d%d", &L[i], &R[i], &H[i]);
67             alls.push_back(H[i]);
68         }
69         sort(alls.begin(), alls.end());
70         alls.erase(unique(alls.begin(), alls.end()), alls.end());
71
72         //rt[0] = build(0, alls.size() - 1);
73         for (int i = 1; i <= q; i++) H[i] = find(H[i]);
74         for (int i = 1; i <= n; i++) a[i] = find(a[i]);
75
76         for (int i = 1; i <= n; i++)
77             rt[i] = insert(rt[i - 1], 0, alls.size() - 1, a[i]);
78
79         for (int i = 1; i <= q; i++) {
80             printf("%d ", query(rt[R[i]], 0, alls.size() - 1, H[i])

```

```

82         - query(rt[L[i] - 1], 0, alls.size() - 1, H[i]));
83     }
84     puts("");
85 }
86 return 0;
87 }

```

## 2.7 线段树合并

```

1  int merge(int p, int q, int l, int r) {
2      if(!p || !q) return p + q;
3      if(l == r) {
4          //维护信息, 一般是 t[p].val += t[q].val 等
5          // t[p].val.first += t[q].val.first;
6          return p;
7      }
8      int mid = l + r >> 1;
9      t[p].l = merge(t[p].l, t[q].l, l, mid);
10     t[p].r = merge(t[p].r, t[q].r, mid + 1, r);
11     // pushup();
12     // t[p].val = max(t[t[p].l].val, t[t[p].r].val);
13     return p;
14 }

```

```

1  // codeforces600E
2  #include <bits/stdc++.h>
3
4  using namespace std;
5  using ll = long long;
6
7  namespace SegTree {
8
9  struct Info {
10     ll cnt, ans; // 颜色的个数, 颜色编号和
11     Info(ll v = 0, ll x = 0) : cnt(v), ans(x) {}
12 };
13
14 Info operator+(const Info& a, const Info& b) {
15     if(a.cnt > b.cnt) {
16         return a;
17     } else if(a.cnt < b.cnt) {
18         return b;
19     } else {
20         return Info(a.cnt, a.ans + b.ans);
21     }
22 }
23
24 Info merge(const Info &a, const Info &b) {
25     return a + b;
26 }

```

```

27
28 struct Node {
29     Node *l,*r;
30     Info val;
31     Node() : l(nullptr), r(nullptr), val() {}
32 };
33
34 void pull(Node *&p) {
35     p->val = merge(p->l == nullptr ? Info() : p->l->val, p->r == nullptr ? Info() :
        ↪ p->r->val);
36 }
37
38 // query(t, θ, n, query_left, query_right)
39 Info query(Node *t, int l, int r, int le, int ri) { // [le, ri)
40     //   l   r
41     // 00xxxxx000
42     // L R LR L R
43     // xx00x00xx0
44     if(ri <= l || r <= le || t == nullptr) {
45         return Info();
46     }
47     if(le <= l && r <= ri) {
48         return t->val;
49     }
50     int m = (l + r) / 2;
51     return merge(query(t->l, l, m, le, ri), query(t->r, m, r, le, ri));
52 }
53
54 void modify(Node *&t, int l, int r, int x, int v) { // [l,r)
55     if(t == nullptr) {
56         t = new Node();
57     }
58     if(r - l == 1) {
59         t->val.cnt += v;
60         t->val.ans = x;
61         return;
62     }
63
64     int m = (l + r) / 2;
65     if(x < m) {
66         modify(t->l, l, m, x, v);
67     }else {
68         modify(t->r, m, r, x, v);
69     }
70     pull(t);
71 }
72
73 Node* merge(Node *&t, Node *&o, int l, int r) { // [l,r)
74     if(t == nullptr) return o;

```

```

75     if(o == nullptr) return t;
76
77     Node *p = new Node();
78
79     if(r - l == 1) {
80         p->val = t->val;
81         p->val.cnt += o->val.cnt;
82         return p;
83     }
84
85     int m = (l + r) / 2;
86     p->l = merge(t->l, o->l, l, m);
87     p->r = merge(t->r, o->r, m, r);
88     pull(p);
89     return p;
90 }
91
92 }
93
94 using namespace SegTree;
95
96
97
98 int main() {
99     ios::sync_with_stdio(false);
100     cin.tie(nullptr);
101
102     int n;
103     cin >> n;
104     vector<int> col(n);
105     for (int i = 0; i < n; i++) {
106         cin >> col[i];
107     }
108     vector<Node*> rt(n + 1, nullptr);
109     vector<vector<int>> G(n);
110     for (int i = 1; i < n; i++) {
111         int u, v;
112         cin >> u >> v;
113         u--, v--;
114         G[u].push_back(v);
115         G[v].push_back(u);
116     }
117     vector<ll> ans(n);
118     function<void(int, int)> dfs = [&](int u, int fa) {
119         modify(rt[u], 1, n + 1, col[u], 1);
120         for (auto v:G[u]) {
121             if(v == fa) continue;
122             dfs(v, u);
123             rt[u] = merge(rt[u], rt[v], 1, n + 1);

```

```

124     }
125     ans[u] = rt[u]->val.ans;
126 };
127 dfs(0, -1);
128 for (int i = 0; i < n; i++) {
129     cout << ans[i] << " \n"[i == n - 1];
130 }
131
132 return 0;
133 }

```

## 2.8 DSU on tree

静态统计树上所有子树具有某些性质的个数

$O(n^2)$  的方式:

1. 递归每个子树，记录某种性质出现
2. 退出子树时候清空 `cnt` 数组（一般用于记录性质啥的），以免影响到其他子树的统计
3. 这样对于父节点而言，就要重新统计这颗子树的性质

优化:

我们发现，某个节点的最后一棵子树可以不用清空统计。这貌似是一个常数优化，但当我们最后一棵子树改成重儿子的时候，可以证明这样的时间复杂度是  $O(n \log n)$  的。

流程:

1. 第一次 `dfsHson` 得到重儿子
2. `dfs` 中，先递归其所有轻儿子，最后递归重儿子

```

for (auto v:G[u]) {
    if(v == fa || v == son[u]) continue;
    dfs(v, u, false);
}
if(son[u] != -1) {
    dfs(son[u], u, true);
}

```

3. `calc` 当前点 (`u`) 以及其所有轻儿子 (在 `calc` 中递归完成)
4. 如果当前点是其父亲的轻儿子，消除影响 (同样通过 `calc` 完成, `calc` 传入一个影响 `val` 可正可负)

```

1 // https://codeforces.com/problemset/problem/600/E
2 // 树的节点有颜色，我们称一种颜色占领了一个子树，
3 // 当且仅当没有其他颜色在这个子树中出现得比它多。
4 // 求占领每个子树的所有颜色之和。
5

```



```

6  #include <bits/stdc++.h>
7
8  using namespace std;
9  using ll = long long;
10
11 int main() {
12     ios::sync_with_stdio(false);
13     cin.tie(nullptr);
14     int n;
15     cin >> n;
16     vector<int> col(n);
17     vector<vector<int>> G(n);
18     for (int i = 0; i < n; i++) {
19         cin >> col[i];
20     }
21
22     for (int i = 1; i < n; i++) {
23         int u, v;
24         cin >> u >> v;
25         u--, v--;
26         G[u].push_back(v), G[v].push_back(u);
27     }
28     vector<int> sz(n), son(n, -1); // son[u] 表示 u 的重儿子
29     function<void(int, int)> dfsHson = [&](int u, int fa) {
30         sz[u] = 1;
31         for (auto v : G[u]) {
32             if(v == fa) continue;
33             dfsHson(v, u);
34             sz[u] += sz[v];
35             if(son[u] == -1 || sz[son[u]] < sz[v]) {
36                 son[u] = v;
37             }
38         }
39     };
40     dfsHson(0, -1);
41
42     vector<ll> cnt(n + 1), ans(n);
43     ll mx = 0, sum = 0;
44     function<void(int, int, int, int)> calc = [&](int u, int fa, int val, int Hson) {
45         cnt[col[u]] += val;
46         if(cnt[col[u]] > mx) {
47             mx = cnt[col[u]];
48             sum = col[u];
49         } else if(cnt[col[u]] == mx) {
50             sum += col[u];
51         }
52         for (auto v : G[u]) {
53             if(v == fa || v == Hson) continue;
54             calc(v, u, val, Hson);

```

```

55     }
56 };
57
58 function<void(int, int, int)> dfs = [&](int u, int fa, bool isCurHson) {
59     for (auto v:G[u]) {
60         if(v == fa || v == son[u]) continue;
61         dfs(v, u, false);
62     }
63     if(son[u] != -1) {
64         dfs(son[u], u, true);
65     }
66     calc(u, fa, 1, son[u]);
67     ans[u] = sum;
68
69     // 如果当前点是其父节点的轻儿子，则消除其影响
70     if(!isCurHson) {
71         calc(u, fa, -1, -1);
72         sum = mx = 0;
73     }
74 };
75 dfs(0, -1, false);
76
77 for (int i = 0; i < n; i++) {
78     cout << ans[i] << " \n"[i == n - 1];
79 }
80 return 0;
81 }

```

## 2.9 树链剖分

```

1  #include<bits/stdc++.h>
2  #define pb push_back
3  using namespace std;
4
5  using ll = long long;
6  const int N = 1e5 + 10;
7
8  struct node {
9      int l, r;
10     ll add, sum;
11 } t[N << 2];
12 int n, m, w[N], nw[N];
13 vector<int> G[N];
14
15 int dep[N], top[N], son[N], dfn[N], sz[N], fa[N], cnt;
16 ///////////////////////////////////////////////////线段树部分////////////////////////////////////
17
18 void pushdown(int p) {
19     auto &rt = t[p], &n1 = t[p << 1], &n2 = t[p << 1 | 1];

```

```

20     if(rt.add) {
21         nl.add += rt.add, nl.sum += (ll)(nl.r - nl.l + 1) * rt.add;
22         nr.add += rt.add, nr.sum += (ll)(nr.r - nr.l + 1) * rt.add;
23         rt.add = 0;
24     }
25 }
26
27 void pushup(int p) { t[p].sum = t[p << 1].sum + t[p << 1 | 1].sum; }
28
29 void build(int p, int l, int r) {
30     t[p] = {l, r, 0, nw[l]};
31     if(l == r) return;
32
33     int mid = l + r >> 1;
34     build(p << 1, l, mid);
35     build(p << 1 | 1, mid + 1, r);
36     pushup(p);
37 }
38
39 ll query(int p, int l, int r) {
40     if(t[p].l >= l && t[p].r <= r) return t[p].sum;
41
42     pushdown(p);
43     int mid = t[p].l + t[p].r >> 1;
44     ll res = 0;
45     if(l <= mid) res += query(p << 1, l, r);
46     if(r > mid) res += query(p << 1 | 1, l, r);
47     //pushup(p);
48     return res;
49 }
50
51 void modify(int p, int l, int r, int k) {
52     if(t[p].l >= l && t[p].r <= r) {
53         t[p].sum += (t[p].r - t[p].l + 1) * k;
54         t[p].add += k;
55         return;
56     }
57
58     pushdown(p);
59     int mid = t[p].l + t[p].r >> 1;
60     if(l <= mid) modify(p << 1, l, r, k);
61     if(r > mid) modify(p << 1 | 1, l, r, k);
62     pushup(p);
63 }
64
65 //////////////////////////////////////////////////树剖部分//////////////////////////////////////
66 //第一次 dfs 维护 sz, 重儿子, dep[], fa[]
67 void dfs1(int u, int fath) {
68     sz[u] = 1, dep[u] = dep[fath] + 1, fa[u] = fath;

```

```

69     for(int v:G[u]) {
70         if(v == fath) continue;
71         dfs1(v, u);
72         sz[u] += sz[v];
73         if(sz[son[u]] < sz[v]) son[u] = v;
74     }
75 }
76 //第二次 dfs, 维护 dfs 序,
77 void dfs2(int u, int tp) {
78     dfn[u] = ++cnt, nw[cnt] = w[u], top[u] = tp;
79     if(!son[u]) return;
80     dfs2(son[u], tp); //递归重儿子
81     //维护轻儿子信息
82     for(int v:G[u]) {
83         if(v == fa[u] || v == son[u]) continue;
84         dfs2(v, v);
85     }
86 }
87
88 void modify_path(int u, int v, int k) {
89     while(top[u] != top[v]) {
90         if(dep[top[u]] < dep[top[v]]) swap(u, v);
91         modify(1, dfn[top[u]], dfn[u], k);
92         u = fa[top[u]];
93     }
94     if(dep[u] < dep[v]) swap(u, v);
95     modify(1, dfn[v], dfn[u], k);
96 }
97
98 void modify_tree(int u, int k) {
99     modify(1, dfn[u], dfn[u] + sz[u] - 1, k);
100 }
101
102 ll query_tree(int u) {
103     return query(1, dfn[u], dfn[u] + sz[u] - 1);
104 }
105
106 ll query_path(int u, int v) {
107     ll res = 0;
108     while(top[u] != top[v]) {
109         if(dep[top[u]] < dep[top[v]]) swap(u, v);
110         res += query(1, dfn[top[u]], dfn[u]);
111         u = fa[top[u]];
112     }
113     if(dep[u] < dep[v]) swap(u, v);
114     res += query(1, dfn[v], dfn[u]);
115     return res;
116 }
117

```

```

118 //////////////////////////////////////
119 int main() {
120
121     scanf("%d", &n);
122     for(int i = 1; i <= n; i++) scanf("%d", &w[i]);
123     for(int i = 1; i < n; i++) {
124         int u, v; scanf("%d%d", &u, &v);
125         G[u].pb(v), G[v].pb(u);
126     }
127     dfs1(1, 0);
128     dfs2(1, 1);
129
130     build(1, 1, n);
131
132     scanf("%d", &m);
133     while(m -- ) {
134         int op, u, v, k;
135         scanf("%d%d", &op, &u);
136         if(op == 1) {
137             scanf("%d%d", &v, &k);
138             modify_path(u, v, k);
139         }
140         else if(op == 2) {
141             scanf("%d", &k);
142             modify_tree(u, k);
143         }
144         else if(op == 3) {
145             scanf("%d", &v);
146             printf("%lld\n", query_path(u, v));
147         }
148         else
149             printf("%lld\n", query_tree(u));
150     }
151     return 0;
152 }

```

## 2.10 珂朵莉

```

1 // 珂朵莉树
2 // 区间赋值 且数据随机（或者操作种类有限）时使用
3 struct ODT {
4     const int n;
5     map<int, int> mp;
6     ODT(int n) : n(n) { mp[-1] = 0; // mp[0] = 0, mp[n] = -1;}
7     void split(int x) {
8         auto it = prev(mp.upper_bound(x)); //找到左端点小于等于 x 的区间
9         mp[x] = it->second; //设立新的区间，并将上一个区间储存的值复制给本区间。
10    }
11    void assign(int l, int r, int v) { // 注意，这里的 r 是区间右端点 +1

```

```

12     split(l);
13     split(r);
14     auto it = mp.find(l);
15     while (it->first != r) {
16         // 一般在此处可以同时完成相关的更新操作
17         // 若重写一个 update 或者 query
18         // 与 assign 一致, 就是把下一行换成 it = next(it);
19         it = mp.erase(it);
20     }
21     mp[l] = v;
22 }
23 };

```

## 2.11 CDQ 分治

```

1 // 三维偏序模板
2 #include <bits/stdc++.h>
3
4 using namespace std;
5
6 struct Node {
7     int x, y, z, cnt, ans; // cnt: 坐标都为 {x, y, z} 的点数量
8     bool operator<(const Node&rhs) const {
9         if(x != rhs.x) return x < rhs.x;
10        if(y != rhs.y) return y < rhs.y;
11        return z < rhs.z;
12    }
13 };
14
15 struct fenwick {
16     const int n;
17     vector<int> c;
18     fenwick(int n) : n(n), c(n + 1) {}
19     void add(int pos, int val) {
20         for (int i = pos; i <= n; i += i & -i) {
21             c[i] += val;
22         }
23     }
24     int sum(int pos) {
25         int res{0};
26         for (int i = pos; i; i -= i & -i) {
27             res += c[i];
28         }
29         return res;
30     }
31 };
32
33 int main() {
34     ios::sync_with_stdio(false);

```

```

35     cin.tie(nullptr);
36     int n, k;
37     cin >> n >> k;
38     map<array<int, 3>, int> s;
39     fenwick fen(k);
40     for (int i = 0; i < n; i++) {
41         int x, y, z;
42         cin >> x >> y >> z;
43         s[{x, y, z}]++;
44     }
45     vector<Node> node(s.size());
46     int m = 0;
47     for (auto it = s.begin(); it != s.end(); ++it) {
48         auto [x, y, z] = it->first;
49         auto cnt = it->second;
50         node[m++] = {x, y, z, cnt, 0};
51     }
52     assert(m == (int)s.size());
53     sort(node.begin(), node.end());
54     // for (int i = 0; i < m; i++) {
55     //     auto [x, y, z, cnt, _] = node[i];
56     //     cout << "x = " << x << ' ' << y << ' ' << z << ' ' << cnt << '\n';
57     // }
58
59     auto cmp2nd = [&](const Node &lhs, const Node &rhs) -> bool {
60         if(lhs.y != rhs.y) return lhs.y < rhs.y;
61         return lhs.z < rhs.z;
62     };
63
64     function<void(int, int)> cdq = [&](int l, int r) {
65         if(r - l == 1) {
66             return;
67         }
68         int m = (l + r) / 2;
69         cdq(l, m); cdq(m, r);
70         sort(node.begin() + l, node.begin() + m, cmp2nd);
71         sort(node.begin() + m, node.begin() + r, cmp2nd);
72
73         int j = l;
74         for (int i = m; i < r; i++) {
75             while(j < m && node[i].y >= node[j].y) {
76                 // 当涉及修改操作和查询操作时候
77                 // 一般 Node 内会有一个标识
78                 // 修改操作: if(node[j].type == 0) {
79                 //     fen.add(node[j].z, node[j].cnt)
80                 // }
81                 // 统计答案时也只需要对查询操作统计
82                 // 清空树状数组也是对修改操作
83                 fen.add(node[j].z, node[j].cnt);

```

```

84         j++;
85     }
86     node[i].ans += fen.sum(node[i].z);
87 }
88 // 注意不能清空整个树状数组
89 for (int t = 1; t < j; t++) {
90     fen.add(node[t].z, -node[t].cnt);
91 }
92 };
93 cdq(0, m);
94
95 vector<int> res(n);
96 for (int i = 0; i < m; i++) {
97     res[node[i].ans + node[i].cnt - 1] += node[i].cnt;
98 }
99 for (int i = 0; i < n; i++) {
100     cout << res[i] << '\n';
101 }
102
103 return 0;
104 }

```

## 2.12 莫队

普通莫队:

```

1  #include <bits/stdc++.h>
2  #define endl '\n'
3  #define rep(i, a, b) for (int i = (a); i <= (b); i++)
4  using namespace std;
5
6  const int N = 5e4 + 10, M = 2e5 + 10, S = 1e6 + 10; //值域
7
8  int n, A[N], ans[M], cnt[S], m, sq, cur;
9
10 struct query {
11     int l, r, id;
12     bool operator<(const query &rhs) const { //奇偶化排序
13         if (l / sq != rhs.l / sq)
14             return l < rhs.l;
15         if (l / sq & 1)
16             return r < rhs.r;
17         return r > rhs.r;
18     }
19 } q[M];
20
21 void add(int p) {
22     if(cnt[A[p]] == 0) cur++;
23     cnt[A[p]]++;
24 }

```



```

25
26 void del(int p) {
27     cnt[A[p]]--;
28     if(cnt[A[p]] == 0) cur--;
29 }
30
31 int main() {
32     ios::sync_with_stdio(false);
33     cin.tie(nullptr);
34
35     cin >> n;
36     sq = sqrt(n);
37     rep(i, 1, n) cin >> A[i];
38     cin >> m;
39     rep(i, 1, m) {
40         int l, r;
41         cin >> l >> r;
42         q[i] = {l, r, i};
43     }
44     sort(q + 1, q + 1 + m);
45
46     int l = 1, r = 0;
47     rep(i, 1, m) {
48         while(l > q[i].l) add(--l);
49         while(r < q[i].r) add(++r);
50         while(l < q[i].l) del(l++);
51         while(r > q[i].r) del(r--);
52         ans[q[i].id] = cur;
53     }
54     rep(i, 1, m) cout << ans[i] << endl;
55
56     return 0;
57 }

```

带修莫队:

```

1  #include <bits/stdc++.h>
2  #define endl '\n'
3  #define rep(i, a, b) for (int i = (a); i <= (b); i++)
4  using namespace std;
5
6  const int N = 134000, S = 1e6 + 10; //值域
7
8  int n, m, mq, mc, len, cur;
9  int w[N], cnt[S], ans[N];
10 struct Query {
11     int id, l, r, tim;
12 } q[N];
13 struct Modify {
14     int pos, val;
15 } c[N];

```

```

16
17 int get(int x) {
18     return x / len;
19 }
20
21 bool cmp(const Query& a, const Query& b) {
22     int al = get(a.l), ar = get(a.r);
23     int bl = get(b.l), br = get(b.r);
24     if (al != bl) return al < bl;
25     if (ar != br) return ar < br;
26     return a.tim < b.tim;
27 }
28
29 void add(int val) {
30     if(cnt[val] == 0) cur++;
31     cnt[val]++;
32 }
33
34 void del(int val) {
35     cnt[val]--;
36     if(cnt[val] == 0) cur--;
37 }
38 int main() {
39     ios::sync_with_stdio(false);
40     cin.tie(nullptr);
41     cin >> n >> m;
42     rep(i, 1, n) cin >> w[i];
43
44     rep(i, 1, m) {
45         char op[2];
46         int a, b;
47         cin >> op >> a >> b;
48         if (*op == 'Q') mq++, q[mq] = {mq, a, b, mc};
49         else c[ ++ mc] = {a, b};
50     }
51
52     len = cbrt((double)n * max(1, mc)) + 1;
53     sort(q + 1, q + mq + 1, cmp);
54
55     int l = 1, r = 0, t = 0;
56     rep(i, 1, mq) {
57         auto [id, ql, qr, qt] = q[i];
58         while (l < ql) del(w[l++]);
59         while (l > ql) add(w[--l]);
60         while (r < qr) add(w[++r]);
61         while (r > qr) del(w[r--]);
62         while (t < qt) {
63             t ++ ;
64             if (ql <= c[t].pos && qr >= c[t].pos) {

```

```

65         del(w[c[t].pos]);
66         add(c[t].val);
67     }
68     swap(w[c[t].pos], c[t].val);
69 }
70 while (t > qt) {
71     if (ql <= c[t].pos && qr >= c[t].pos) {
72         del(w[c[t].pos]);
73         add(c[t].val);
74     }
75     swap(w[c[t].pos], c[t].val);
76     t--;
77 }
78 ans[id] = cur;
79 }
80
81 rep(i, 1, mq) printf("%d\n", ans[i]);
82 return 0;
83 }

```

## 3 图论

### 3.1 spfa

```

1  #include <bits/stdc++.h>
2  #define pb push_back
3  using namespace std;
4
5  const int N = 1e5 + 10, inf = 0x3f3f3f3f;
6  struct node{int v, w;};
7  vector<node> G[N];
8  int dis[N], n, m;
9  bool inq[N];
10
11 void spfa() {
12     memset(dis, 0x3f, sizeof dis);
13     dis[1] = 0;
14     inq[1] = 1;
15     queue<int> q;
16     q.push(1);
17     while(q.size()) {
18         int u = q.front(); q.pop();
19         inq[u] = 0;
20         for(auto [v, w]:G[u]) {
21             if(dis[v] > w + dis[u]) {
22                 dis[v] = dis[u] + w;
23                 if(!inq[v])
24                     q.push(v), inq[v] = true;
25             }
26         }
27     }
28 }

```

```

26     }
27 }
28 }
29
30 int main() {
31     cin >> n >> m;
32     while(m -- ) {
33         int u, v, w;
34         cin >> u >> v >> w;
35         G[u].pb({v, w});
36     }
37     spfa();
38     if(dis[n] == inf)    cout << "impossible";
39     else                cout << dis[n];
40     return 0;
41 }

```

### 3.2 dijkstra

稀疏图 dijkstra:

```

1  //acwing 849
2  #include <bits/stdc++.h>
3  using namespace std;
4  const int N = 510, inf = 0x3f3f3f3f;
5  int dis[N], G[N][N], n, m;
6  bool vis[N];
7
8  void dij() {
9      memset(dis, 0x3f, sizeof dis);
10     dis[1] = 0;
11     for (int j = 0; j < n; j++) {
12         int minv = inf, pos = -1;
13         for(int i = 1; i <= n; i++)
14             if (!vis[i] && minv > dis[i])
15                 minv = dis[i], pos = i;
16
17         if(pos == -1) break;
18         vis[pos] = 1;
19         for (int i = 1; i <= n; i++)
20             if(!vis[i] && dis[pos] + G[pos][i] < dis[i])
21                 dis[i] = dis[pos] + G[pos][i];
22     }
23 }
24
25 int main() {
26     cin >> n >> m;
27     scanf("%d %d", &n, &m);
28     memset(G, 0x3f, sizeof(G));
29     while(m --) {

```

```

30         int u, v, w; scanf("%d %d %d", &u, &v, &w);
31         G[u][v] = min(G[u][v], w);
32     }
33
34     dij();
35
36     cout << (dis[n] == inf ? -1 : dis[n]);
37     return 0;
38 }

```

稠密图 dijkstra:

```

1  #include <bits/stdc++.h>
2  #define pb push_back
3  #define fi first
4  #define se second
5  using namespace std;
6
7  using P = pair<int, int>;
8  const int N = 151000, inf = 0x3f3f3f3f;
9  struct node{int v, w;};
10 vector<node> G[N];
11 int dis[N], n, m;
12 bool vis[N];
13
14 void dij() {
15     memset(dis, 0x3f, sizeof dis);
16     priority_queue<P, vector<P>, greater<P>> q;
17     q.push({0, 1});
18     while(q.size()) {
19         auto t = q.top(); q.pop();
20         int u = t.se, d = t.fi;
21         if(vis[u]) continue;
22         vis[u] = true;
23         for(auto [v, w] : G[u]) {
24             if(dis[v] > d + w) {
25                 dis[v] = d + w;
26                 q.push({dis[v], v});
27             }
28         }
29     }
30 }
31
32 int main() {
33     ios::sync_with_stdio(false);
34     cin >> n >> m;
35     while(m -- ) {
36         int u, v, w; cin >> u >> v >> w;
37         G[u].pb({v, w});
38     }
39     dij();

```

```

40     cout << (dis[n] == inf ? -1 : dis[n]);
41     return 0;
42 }

```

### 3.3 最小生成树

```

1  // kruskal
2  using ll = long long;
3  struct Edge {
4      int u, v, w;
5      bool operator<(const edge &rhs) const { return w < rhs.w; }
6  };
7
8  struct DSU {
9      int n;
10     vector<int> f;
11     DSU(int n): n(n), f(n) { iota(f.begin(), f.end(), 0); }
12     int find(int x) { return x == f[x] ? x : f[x] = find(f[x]); }
13     bool merge(int u, int v) {
14         u = find(u), v = find(v);
15         if(u == v) return false;
16         f[v] = u;
17         return true;
18     }
19 };
20
21 int kruskal() {
22     cin >> n >> m;
23     ll ans = 0;
24     vector<Edge> edge(m);
25     for (int i = 0; i < m; i++) {
26         int u, v, w;
27         cin >> u >> v >> w;
28         u--, v--;
29         edge[i] = {u, v, w};
30     }
31     sort(edge.begin(), edge.end());
32     DSU dsu(n);
33     for (auto [u, v, w]: edge) {
34         if(dsu.merge(u, v)) {
35             ans += w;
36         }
37     }
38     return ans;
39 }
40
41
42 //prim
43 const int N = 510, inf = 0x3f3f3f3f;

```

```

44 int G[N][N], dis[N];
45 int n, m;
46 bool vis[N];
47
48 int prim() {
49     int res = 0;
50     memset(dis, 0x3f, sizeof dis);
51     dis[1] = 0;    //随便选一点进入 mst 集合
52     for(int j = 0; j < n; j++) {
53         int minv = inf, pos = -1;
54         for(int i = 1; i <= n; i++)
55             if(!vis[i] && dis[i] < minv)
56                 pos = i, minv = dis[i];
57
58         if(pos == -1) return inf;
59         vis[pos] = true;
60         res += dis[pos];
61
62         for(int i = 1; i <= n; i++)
63             if(!vis[i] && dis[i] > G[pos][i])
64                 dis[i] = G[pos][i];
65     }
66     return res;
67 }

```

另外，对于完全图的 *MST* 问题，可以考虑使用 *Boruvka* 算法。我们要在  $n \log n$  或  $n \log^2 n$  时间内求出每个连通块最小的连接的边，而这个边权一般可通过点权以一定方式求出。通常不用直接写出，运用该思想求解。

### 3.4 kruskal 重构树

```

1 //kruskal 重构树
2
3 //性质：
4 //两个点之间的所有简单路径上最大边权的最小值
5 // = 最小生成树上两个点之间的简单路径上的最大值
6 // = Kruskal 重构树上两点之间的 LCA 的权值。
7 //Loj136
8 #include <bits/stdc++.h>
9 #define pb push_back
10 using namespace std;
11
12 const int N = 1010 << 1, M = 3e5 + 10;
13 int n, m, k, val[N]; // kruskal 重构树的点权
14 int idx; // 重构树的节点数
15
16 struct Edge{
17     int u, v, w;
18     bool operator<(const Edge &rhs) const { return w < rhs.w; }
19 }edges[M];

```

```

20
21 vector<int> G[N];
22
23 int p[N];
24 int find(int x) { return x == p[x] ? x : p[x] = find(p[x]); }
25
26 int dep[N], fa[N][21];
27
28 void bfs(int s) {
29     dep[0] = 0, dep[s] = 1;
30     queue<int> q;
31     q.push(s);
32     while(q.size()) {
33         int u = q.front(); q.pop();
34         for(int v:G[u]) {
35             if(dep[v] > dep[u] + 1) {
36                 dep[v] = dep[u] + 1;
37                 q.push(v);
38                 fa[v][0] = u;
39                 for (int i = 1; i <= 20; i++)
40                     fa[v][i] = fa[fa[v][i - 1]][i - 1];
41             }
42         }
43     }
44 }
45
46 int lca(int a, int b) {
47     if(dep[a] < dep[b]) swap(a, b);
48     for (int k = 20; k >= 0; k--)
49         if(dep[fa[a][k]] >= dep[b])
50             a = fa[a][k];
51     if(a == b) return a;
52     for (int k = 20; k >= 0; k--)
53         if(fa[a][k] != fa[b][k])
54             a = fa[a][k], b = fa[b][k];
55     return fa[a][0];
56 }
57
58 void build() {
59     idx = n;
60     int cnt = 0;
61     for (int i = 1; i <= m; i++) {
62         int u = edges[i].u, v = edges[i].v, w = edges[i].w;
63         int fu = find(u), fv = find(v);
64         if(fu != fv) {
65             val[++idx] = w;
66             G[idx].pb(fu), G[idx].pb(fv);
67             G[fu].pb(idx), G[fv].pb(idx);
68             p[fu] = p[fv] = idx;

```



```

69         cnt++;
70     }
71     if(cnt >= n - 1) break;
72 }
73 }
74
75 int main() {
76     scanf("%d %d %d", &n, &m, &k);
77     for (int i = 1; i <= m; i++) {
78         int u, v, w; scanf("%d %d %d", &u, &v, &w);
79         edges[i] = {u, v, w};
80     }
81     sort(edges + 1, edges + m + 1);
82     for (int i = 1; i <= (n << 1); i++) p[i] = i;
83
84     build(); // kruskal 重构树
85
86     memset(dep, 0x3f, sizeof dep);
87     bfs(idk); //bfs 的根节点一定要是重构树的最高点
88
89     while(k -- ) {
90         int s, t;
91         scanf("%d %d", &s, &t);
92         if(find(s) != find(t)) puts("-1");
93         else
94             printf("%d\n", val[lca(s, t)]);
95     }
96     return 0;
97 }

```

### 3.5 二分图匹配

二分图匹配的模型有两个要素：

1. 节点能分成独立的两个集合，每个集合内部有 0 条边
2. 每个节点只能与 1 条匹配边相连

二分图最小覆盖模型特点是：每条边有 2 个端点，二者至少选择一个。

**könig** 定理：二分图最小点覆盖包含的点数等于二分图最大匹配数包含的边数。

图的最大独立集：点集  $S$  中任意两点之间都没有边相连。其大小等于  $n - \text{最大匹配数}$ 。（ $n$  是二分图总点数）

```

1  /* 染色法判断二分图
2  bool vis[N];
3  int col[N], flag = 1, n, m;
4  void dfs(int u, int t) {
5      if (vis[u]) {
6          if (col[u] != t) flag = 0;
7          return;

```

```

8     }
9     vis[u] = 1; col[u] = t;
10    for (int v : g[u]) {
11        dfs(v, t ^ 1);
12    }
13 }
14 bool isbit() { // 是否为二分图
15     for (int u = 1; u <= n; u++) {
16         if (!vis[u]) dfs(u, 0);
17     }
18     return flag;
19 }
20 */
21 int G[N][M]; // 左半部 n, 右半部 m
22 int n, m, p[M], vis[M];
23 bool match(int u) {
24     for (int i = 1; i <= m; i++) {
25         if (G[u][i] && !vis[i]) {
26             vis[i] = true;
27             if (p[i] == 0 || match(p[i])) {
28                 p[i] = u; return true;
29             }
30         }
31     }
32     return false;
33 }
34 int main() {
35     /* 建图 */
36     int res = 0;
37     for (int i = 1; i <= n; i++) {
38         memset(vis, 0, sizeof vis);
39         if (match(i)) res++;
40     }
41     return 0;
42 }

```

### 3.6 强连通分量缩点

时间复杂度  $O(m + n)$ , 反向枚举 `scc_cnt` 即是新图拓扑序。

```

1  #include <bits/stdc++.h>
2  #define pb push_back
3  using namespace std;
4
5  const int N = 1e4 + 10;
6  vector<int> G[N], G2[N];
7  stack<int> s;
8  int n, m, tim, scc_cnt;
9  int w[N], dfn[N], low[N], id[N];
10 int dist[N], ind[N], W[N];

```

```

11  bool ins[N];
12
13  void tarjan(int u) {
14      low[u] = dfn[u] = ++tim;
15      s.push(u); ins[u] = true;
16      for(int v:G[u]) {
17          if(!dfn[v]) {
18              tarjan(v);
19              low[u] = min(low[v], low[u]);
20          }
21          else if(ins[v])
22              low[u] = min(low[u], dfn[v]);
23      }
24      if(low[u] == dfn[u]) {
25          int y; ++scc_cnt;
26          do {
27              y = s.top(); s.pop();
28              ins[y] = false;
29              id[y] = scc_cnt;
30              W[scc_cnt] += w[y];
31          } while (y != u);
32      }
33  }
34
35  int sol() {
36      queue<int> q;
37      for (int i = 1; i <= scc_cnt; i++)
38          if(!ind[i]) {
39              q.push(i);
40              dist[i] = W[i];
41          }
42
43      while(q.size()) {
44          //cout << "cnt = " << ++cnt << endl;
45          int u = q.front(); q.pop();
46          for (int v:G2[u]) {
47
48              ↪ //当有重边时, dist[v] 被更新的值始终不变, 即 dist[v] = dist[u] + W[v]; 所以不会影响
49              dist[v] = max(dist[v], dist[u] + W[v]);
50              if(--ind[v] == 0)
51                  q.push(v);
52          }
53
54          int ans = 0;
55          for (int i = 1; i <= scc_cnt; i++)
56              ans = max(ans, dist[i]);
57          return ans;
58      }

```

```

59
60
61 int main() {
62     ios::sync_with_stdio(false), cin.tie(0);
63     cin >> n >> m;
64     for (int i = 1; i <= n; i++) cin >> w[i];
65     while(m--) {
66         int u, v;
67         cin >> u >> v;
68         G[u].pb(v);
69     }
70     for (int i = 1; i <= n; i++)
71         if(!dfn[i])
72             tarjan(i);
73     //缩点
74     for (int u = 1; u <= n; ++u) {
75         for(int v : G[u]) {
76             if(id[v] != id[u]) {
77                 G2[id[u]].pb(id[v]);
78                 ind[id[v]]++;
79                 //printf("ind[%d] = %d\n", id[v], ind[id[v]]);
80             }
81         }
82     }
83     // debug
84     // for (int i = 1; i <= scc_cnt; i++)
85     //     printf("ind[%d] = %d\n", i, ind[i]);
86     // for (int i = 1; i <= scc_cnt; i++)
87     // {
88     //     printf("%d->", i);
89     //     for (int v:G2[i])
90     //         printf("%d ", v);
91     //     puts("");
92     // }
93     printf("%d\n", sol());
94     return 0;
95 }
96

```

### 3.7 无向图的双连通分量

桥:

- 1 // 一个有桥的连通图，如何把它通过加边变成边双连通图？
- 2 // 1. 求出所有的桥，然后删除这些桥边，剩下的每个连通块都是一个双连通子图。
- 3 // 把每个双连通子图收缩为一个顶点，再把桥边加回来，最后的这图一定是一棵树，边连通度为 1。
- 4 // 2 统计出树中度为 1 的节点的个数，即为叶节点的个数，记为 *cnt*。
- 5 // 3. 则至少在树上添加  $(cnt+1)/2$  条边，就能使树达到边二连通，所以至少添加的边数就是  $(cnt+1)/2$ 。
- 6 #include <bits/stdc++.h>
- 7 #define pb push\_back

```

8  using namespace std;
9  const int N = 5010;
10 int n, m;
11 vector<int> G[N];
12 int low[N], dfn[N], id[N], deg[N];
13 int dcc_cnt, tim, stk[N], top;
14 vector<int> bridge[N];
15
16 void tarjan(int u, int fa) {
17     low[u] = dfn[u] = ++tim;
18     stk[++top] = u;
19
20     for (int i = 0; i < G[u].size(); i++) {
21         int v = G[u][i];
22         if(!dfn[v]) {
23             tarjan(v, u);
24             low[u] = min(low[v], low[u]);
25             if(dfn[u] < low[v])
26                 bridge[u].pb(v), bridge[v].pb(u);
27         }
28         else if(fa != v)
29             low[u] = min(low[u], dfn[v]);
30     }
31     if (dfn[u] == low[u]) {
32         int y;
33         ++dcc_cnt;
34         do {
35             y = stk[top--];
36             id[y] = dcc_cnt;
37         } while (u != y);
38     }
39 }
40
41 int main() {
42     ios::sync_with_stdio(false), cin.tie(0);
43     cin >> n >> m;
44     while(m -- ) {
45         int u, v;
46         cin >> u >> v;
47         G[u].pb(v), G[v].pb(u);
48     }
49     tarjan(1, -1);
50     for (int u = 1; u <= n; u++)
51         deg[id[u]] += bridge[u].size();
52     int cnt = 0;
53     for (int i = 1; i <= dcc_cnt; i++)
54         if(deg[i] == 1)
55             cnt++;
56     cout << (cnt + 1) / 2;

```

```

57     return 0;
58 }

```

割点:

```

1  #include<bits/stdc++.h>
2  #define pb push_back
3  using namespace std;
4  const int N = 2e4 + 10;
5  vector<int> G[N], cut;
6  int tim, n, m, root;
7  int dfn[N], low[N];
8
9  void tarjan(int u) {
10     low[u] = dfn[u] = ++tim;
11     int tot = 0;
12     for(int v:G[u]) {
13         if(!dfn[v]) {
14             tarjan(v);
15             low[u] = min(low[u], low[v]);
16             if (dfn[u] <= low[v])
17                 tot++;
18         }
19         else
20             low[u] = min(low[u], dfn[v]);
21     }
22     if ( (tot > 0 && u != root) || (tot > 1 && u == root))
23         cut.pb(u);
24 }
25
26
27 int main() {
28     cin >> n >> m;
29     while(m --) {
30         int u, v;
31         cin >> u >> v;
32         G[u].pb(v), G[v].pb(u);
33     }
34
35     for (root = 1; root <= n; root ++){
36         if(!dfn[root])
37             tarjan(root);
38     }
39     //不用 sort, 就开一个 bool cut[N];
40     sort(cut.begin(), cut.end());
41     printf("%d\n", cut.size());
42     for(int v : cut)
43         printf("%d ", v);
44
45     return 0;
46 }

```

### 3.8 lca

```

1  /*
2  求 lca: 1. 倍增 2. 树剖 3.tarjan 离线
3  lca 用处
4  1. 树上两点之间的距离 (多维护一个 dist 数组,  $dis[u] + dis[v] - 2 * dis[lca(u, v)]$ )
5  2. 树上两条路径是否相交 (如果两条路径相交, 那么一定有一条路径的 LCA 在另一条路径上)
6  */
7  #include <bits/stdc++.h>
8  #define pb push_back
9  using namespace std;
10 const int N = 1e4 + 10;
11
12 struct node{int v, w;};
13 vector<node> G[N];
14 int fa[N][19], dep[N], dis[N];
15 int n, m;
16
17 void bfs(int s) {
18     memset(dep, 0x3f, sizeof dep);
19     dep[0] = 0, dep[s] = 1;
20     dis[s] = 0;
21     queue<int> q; q.push(s);
22     while(q.size()) {
23         int u = q.front(); q.pop();
24         for(auto [v, w] : G[u]) {
25             if(dep[v] > dep[u] + 1) {
26                 dis[v] = dis[u] + w;
27                 dep[v] = dep[u] + 1;
28                 fa[v][0] = u;
29                 q.push(v);
30                 for(int i = 1; i < 19; ++i)
31                     fa[v][i] = fa[fa[v][i - 1]][i - 1];
32             }
33         }
34     }
35 }
36
37 int lca(int a, int b) {
38     if(dep[a] < dep[b]) swap(a, b);
39     for(int k = 18; k >= 0; k--)
40         if(dep[fa[a][k]] >= dep[b])
41             a = fa[a][k];
42     if(a == b) return a;
43
44     for(int k = 18; k >= 0; --k)
45         if(fa[a][k] != fa[b][k])
46             a = fa[a][k], b = fa[b][k];
47     return fa[a][0];
48 }

```

## 3.9 2-SAT

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  // O(n + m)
4  struct TwoSat {
5      int n;
6      vector<vector<int>> e;
7      vector<bool> ans;
8      TwoSat(int n) : n(n), e(2 * n), ans(n) {}
9      void addClause(int u, bool f, int v, bool g) {
10         e[2 * u + !f].push_back(2 * v + g);
11         e[2 * v + !g].push_back(2 * u + f);
12     }
13     bool satisfiable() {
14         vector<int> id(2 * n, -1), dfn(2 * n, -1), low(2 * n, -1);
15         vector<int> stk;
16         int now = 0, cnt = 0;
17         function<void(int)> tarjan = [&](int u) {
18             stk.push_back(u);
19             dfn[u] = low[u] = now++;
20             for (auto v : e[u]) {
21                 if (dfn[v] == -1) {
22                     tarjan(v);
23                     low[u] = min(low[u], low[v]);
24                 } else if (id[v] == -1) {
25                     low[u] = min(low[u], dfn[v]);
26                 }
27             }
28             if (dfn[u] == low[u]) {
29                 int v;
30                 do {
31                     v = stk.back();
32                     stk.pop_back();
33                     id[v] = cnt;
34                 } while (v != u);
35                 ++cnt;
36             }
37         };
38         for (int i = 0; i < 2 * n; ++i) if (dfn[i] == -1) tarjan(i);
39         for (int i = 0; i < n; ++i) {
40             if (id[2 * i] == id[2 * i + 1]) return false;
41             ans[i] = id[2 * i] > id[2 * i + 1];
42         }
43         return true;
44     }
45     vector<bool> answer() { return ans; }
46 };
47
48 int main() {

```



```

49     ios::sync_with_stdio(false);
50     cin.tie(nullptr);
51     int n, m;
52     cin >> n >> m;
53     TwoSat ts(n);
54     for (int i = 0; i < m; i++) {
55         int u, f, v, g;
56         cin >> u >> f >> v >> g;
57         u--, v--;
58         ts.addClause(u, f, v, g);
59     }
60     if(ts.satisfiable()) {
61         cout << "POSSIBLE" << '\n';
62         for (int i = 0; i < n; i++) {
63             cout << ts.ans[i] << " \n"[i == n - 1];
64         }
65     } else {
66         cout << "IMPOSSIBLE" << '\n';
67     }
68
69     return 0;
70 }

```

### 3.10 基环树

基环树的性质：点数等于边数；度数是点数两倍。一般题目中出现“从一个点到另一个点建一条边”，“ $N$  个点通过恰好  $N$  条双向道路连接起来，不存在任何两条道路连接了相同的两个点”等类似信息可以判定该图是基环树森林。以下是求基环树（森林）直径（和）代码

```

1  //基环树森林求直径和最大
2  #include <bits/stdc++.h>
3  #define endl '\n'
4  #define pb push_back
5  using ll = long long;
6  using namespace std;
7  const int N = 1e6 + 10, M = N << 1;
8  int h[N], e[M], w[M], ne[M], idx;
9  ll s[N], sum[M], d[M]; //环上的前缀和数组，破环成链后两倍的前缀和
10 bool ins[N], vis[N];
11 int n, cir[M], ed[M], cnt; //cnt 环的个数
12 int fa[N], fw[N]; //父节点，反向权值
13 int q[M];
14 ll ans;
15
16 void add(int a, int b, int c) {
17     e[idx] = b, ne[idx] = h[a], w[idx] = c, h[a] = idx++;
18 }
19
20 //深搜 + 栈 找环
21 void dfs(int u, int from) {

```

```

22     vis[u] = ins[u] = true;
23     for (int i = h[u]; ~i; i = ne[i]) {
24         //如果是反向边则跳过, 必须用边来判断, 这样才能确定是通过反向变回到父节点
25         if (i == (from ^ 1)) continue;
26         int v = e[i];
27         fa[v] = u, fw[v] = w[i];
28         if (!vis[v]) dfs(v, i);
29         else if (ins[v]) {
30             cnt++;
31             ed[cnt] = ed[cnt - 1];
32             ll tot = w[i];
33             for (int k = u; k != v; k = fa[k]) {
34                 s[k] = tot;
35                 tot += fw[k];
36                 cir[++ ed[cnt]] = k;
37             }
38             s[v] = tot, cir[++ ed[cnt]] = v;
39         }
40     }
41     ins[u] = false;
42 }
43
44 // 求以 u 为根节点的子树的最大深度
45 ll dfs_d(int u) {
46     vis[u] = true;
47     ll d0 = 0, d1 = 0; //最大距离, 次大距离
48     for (int i = h[u]; ~i; i = ne[i]) {
49         int v = e[i];
50         if (vis[v]) continue;
51         ll d = dfs_d(v) + w[i];
52         if (d >= d0) d1 = d0, d0 = d;
53         else if (d > d1) d1 = d;
54     }
55     ans = max(ans, d1 + d0);
56     return d0;
57 }
58
59 int main() {
60     ios::sync_with_stdio(false), cin.tie(0);
61     cin >> n;
62     memset(h, -1, sizeof h);
63     for (int u = 1; u <= n; u++) {
64         int v; ll w; cin >> v >> w;
65         add(u, v, w), add(v, u, w);
66     }
67
68     for (int i = 1; i <= n; i++)
69         if (!vis[i])
70             dfs(i, -1);

```

```

71
72     memset(vis, 0, sizeof vis);
73     for (int i = 1; i <= n; i++) vis[cir[i]] = 1; //标记环上所有点
74
75     ll res = 0;
76     for (int i = 1; i <= cnt; i++) {
77         ans = 0; // 当前基环树的直径
78         int sz = 0; // 当前基环树的环的大小
79         for (int j = ed[i - 1] + 1; j <= ed[i]; j++) {
80             int k = cir[j];
81             d[sz] = dfs_d(k); // 求以当前点为根的子树的最大深度
82             sum[sz] = s[k];
83             sz++;
84         }
85         // 破环成链, 前缀和数组和 d[] 数组延长一倍
86         for (int j = 0; j < sz; j++)
87             d[sz + j] = d[j], sum[sz + j] = sum[j] + sum[sz - 1];
88
89         // 做一遍滑动窗口, 比较依据是 d[k] - sum[k]
90         int hh = 0, tt = -1;
91         for (int j = 0; j < sz * 2; j++) {
92             while (hh <= tt && q[hh] <= j - sz) hh++;
93             if (hh <= tt) ans = max(ans, d[j] + sum[j] + d[q[hh]] - sum[q[hh]]);
94             while (hh <= tt && d[j] - sum[j] >= d[q[tt]] - sum[q[tt]]) tt--;
95             q[++tt] = j;
96         }
97         res += ans;
98     }
99     cout << res << endl;
100     return 0;
101 }

```

### 3.11 dinic

```

1  #include <bits/stdc++.h>
2  #define pb push_back
3  using namespace std;
4  using ll = long long;
5  const int N = 1e4 + 10;
6  const ll inf = 0x3f3f3f3f3f3f3f3f;
7  int n, m, s, t, dep[N];
8  struct node {int v, cap, rec;};
9  vector<node> G[N];
10
11 bool bfs() {
12     queue<int> q;
13     q.push(s);
14     memset(dep, -1, sizeof dep);
15     dep[s] = 0;

```

```

16     while (q.size()) {
17         int u = q.front(); q.pop();
18         for(auto [v, cap, rev] : G[u])
19             if(dep[v] == -1 && cap)
20                 dep[v] = dep[u] + 1, q.push(v);
21     }
22     return dep[t] != -1;
23 }
24
25 ll dfs(int u, ll lim) {
26     if(u == t || lim == 0) return lim;
27     ll tot_flow = 0;
28     for(auto& [v, cap, rev] : G[u]) {
29         if(dep[v] == dep[u] + 1 && cap > 0) {
30             ll d = dfs(v, min(lim, (ll)cap));
31             cap -= d, G[v][rev].cap += d;
32             lim -= d, tot_flow += d;
33             if(lim == 0) return tot_flow;
34         }
35     }
36     if(lim != 0) dep[u] = -1;
37     return tot_flow;
38 }
39
40 ll dinic() {
41     ll max_flow = 0;
42     while(bfs())
43         max_flow += dfs(s, inf);
44     return max_flow;
45 }
46
47 int main() {
48     scanf("%d%d%d", &n, &m, &s, &t);
49     while(m --) {
50         int u, v, cap; scanf("%d%d", &u, &v, &cap);
51         G[u].pb({v, cap, G[v].size()});
52         G[v].pb({u, 0, G[u].size() - 1});
53     }
54     printf("%lld\n", dinic());
55     return 0;
56 }

```

## 4 动态规划

### 4.1 背包

```

1  #include<bits/stdc++.h>
2  using namespace std;
3

```

```

4  void pack01 {
5      int n, m;
6      cin >> n >> m;
7      vector<int> v(n + 1), w(n + 1);
8      for (int i = 1; i <= n; i++) {
9          cin >> v[i] >> w[i];    // volume, weight
10     }
11
12     for (int i = 1; i <= n; i++) {
13         for (int j = m; j >= v[i]; j--) {
14             f[j] = max(f[j], f[j - v[i]] + w[i]);
15             // 求方案 f[j] += f[j - a[i]];
16         }
17     }
18     cout << f[m] << '\n';
19 }
20
21 void completePack {
22     int n, m;
23     cin >> n >> m;
24     vector<int> v(n + 1), w(n + 1);
25     for(int i = 1; i <= n; i++)    cin >> v[i] >> w[i];
26
27     // vector<vector<int>> f(n + 1, vector<int>(m + 1));
28     // for(int i = 1; i <= n; i++)
29     //     for(int j = 0; j <= m; j++) {
30     //         if(j < v[i]) f[i][j] = f[i - 1][j];
31     //         else f[i][j] = max(f[i - 1][j], f[i][j - v[i]] + w[i] );
32     //     }
33     // cout << f[n][m] << '\n';
34     for(int i = 1; i <= n; i++) {
35         for(int j = v[i]; j <= m; j++) { // 正序循环, 表示 i->i 转移, 每个物品用无限次
36             f[j] = max(f[j], f[j - v[i]] + w[i] );
37             // 方案数: f[j] = f[j] + f[j - w[i]]
38         }
39     }
40     cout << f[m] << '\n';
41     return 0;
42 }
43
44 // 多重背包, 分组背包 见进阶指南 p280
45
46 int main() {
47
48     return 0;
49 }

```

## 4.2 数位 dp

```

1 // lead 前导 0, lim 是否到限制
2 int a[N], dp[N][N];
3 int dfs(int pos, int pre, bool lead, bool limit) {
4     if (!pos) {
5         // 边界条件
6     }
7     if (!limit && !lead && dp[pos][pre] != -1) return dp[pos][pre];
8     int res = 0, up = limit ? a[pos] : 无限限制;
9     for (int i = 0; i <= up; i++) {
10        if (不合法条件) continue;
11        res += dfs(pos - 1, 未定参数, lead && !i, limit && i == up);
12    }
13    return limit ? res : (lead ? res : dp[pos][sum] = res);
14 }
15
16 int cal(int x) {
17     // 一般 dp 初始化成 -1, len = 0;
18     memset(dp, -1, sizeof dp);
19     len = 0;
20     while (x) a[++ len] = x % 进制, x /= 进制;
21     return dfs(len, 未定参数, 1, 1);
22 }
23
24 int main() {
25     cin >> l >> r;
26     cout << cal(r) - cal(l - 1) << endl;
27 }

```

## 4.3 换根 dp

换根 dp 一般时间复杂度为  $\mathcal{O}(n)$ , 需要对树处理得到大规模答案, 如对每个点得到一个答案。

```

1 // 求树上 对某个点来说包含他的连通点集个数
2 #include <bits/stdc++.h>
3 #define pb push_back
4 #define endl '\n'
5 using ll = long long;
6 using namespace std;
7 const int N = 1e6 + 10, mod = 1e9 + 7;
8
9 ll f[N], ans[N], n;
10 vector<int> G[N];
11
12 ll qpow(ll a, ll b) {
13     ll res = 1;
14     while(b) {
15         if(b & 1) res = res * a % mod;
16         a = a * a % mod;

```

```

17         b >>= 1;
18     }
19     return res;
20 }
21
22 void dfs(int u, int fa) {
23     f[u] = 1;
24     for (auto v:G[u]) {
25         if(v == fa) continue;
26         dfs(v, u);
27         f[u] = f[u] * (f[v] + 1) % mod;
28     }
29 }
30
31 /*
32 考虑换根,  $ans[u]$  记为以  $u$  为根, 和整棵树其他点能形成的所有子树数量。(即最终答案)
33 换根方程:  $ans[v] = (ans[u] / (f[v] + 1) + 1) * f[v]$ 
34 解释:  $u$  点答案除以  $v$  点贡献  $(f[v] + 1)$  为与  $v$  无关的  $u$  点答案,  $+1$  后为其余点对  $v$  点贡献, 再乘上  $f[v]$ 
35
36 有一个很坑的地方, 就是  $(f[v] + 1)$  求逆元可能得到  $0$  ( $f[v]$  可能为  $mod - 1$ ), 这时相当于除以  $0$ , 出错
37 当逆元  $inv$  为  $0$  时,  $ans[u]$  实际是由在树形  $dp$  的时候求出的  $f[u]$ , 而  $f[u]$  又等于 (他所有儿子  $f$  的值  $+1$ ) 的乘积。
38 所以  $ans[u] / (f[v] + 1)$  又可以变成  $u$  的其他儿子的乘积:  $u$  除  $v$  外的其他儿子记  $brother$ 。
39  $(f[brother_1] + 1) * (f[brother_2] + 1) * \dots$  他的所有兄弟的值乘积。
40 */
41
42 void dp(int u, int fa) {
43     for (int v:G[u]) {
44         if(v == fa) continue;
45         ll inv = qpow(f[v] + 1, mod - 2);
46         if(inv) ans[v] = (ans[u] * inv % mod + 1) % mod * f[v] % mod;
47         else {
48             ll t = 1;
49             for (auto other:G[u]) {
50                 if(other == v || other == fa) continue;
51                 t = t * (f[other] + 1) % mod;
52             }
53             ans[v] = (t + 1) * f[v] % mod;
54         }
55         dp(v, u);
56     }
57 }
58
59 int main() {
60     cin >> n;
61     for (int i = 1; i < n; i++) {
62         int u, v; cin >> u >> v;
63         G[u].pb(v), G[v].pb(u);
64     }
65     dfs(1, 0);

```

```

66     ans[1] = f[1];
67     dp(1, 0);
68
69     for (int i = 1; i <= n; i++) cout << ans[i] << endl;
70     return 0;
71 }

```

#### 4.4 数据结构优化 dp

##### 1D/1D dp 转移

#### 4.5 斜率优化 dp

(待完善)

1. 单调队列优化
2. 二分
3. cdq 分治

```

1  // p3195
2  #include <bits/stdc++.h>
3
4  using namespace std;
5  using ll = long long;
6
7  const int N = 50050;
8  int n, L;
9  ll dp[N], sum[N], f[N], g[N], h, t, Q[N];
10
11 long double slope(int i, int j) {
12     return (long double) (dp[j] + g[j] - dp[i] - g[i]) / (f[j] - f[i]);
13 }
14
15 int main() {
16     ios::sync_with_stdio(false);
17     cin.tie(nullptr);
18     cin >> n >> L;
19
20     for(int i = 1; i <= n; i++) {
21         cin >> sum[i];
22         sum[i] += sum[i - 1];
23         f[i] = sum[i] + i;
24         g[i] = (f[i] + L + 1) * (f[i] + L + 1);
25     }
26     g[0] = (ll)(L + 1) * (L + 1);
27     for(int i = 1; i <= n; i++) {
28         while(h < t && slope(Q[h], Q[h + 1]) <= 2 * f[i]) h++;
29         dp[i] = dp[Q[h]] + (f[i] - f[Q[h]]) * (f[i] - f[Q[h]]) - L - 1;

```



```

30     while(h < t && slope(Q[t], i) < slope(Q[t - 1], Q[t])) t--;
31     Q[++t] = i;
32 }
33
34 cout << dp[n] << '\n';
35 return 0;
36 }

```

例: LIS 计数

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  constexpr int mod = 1e9 + 7;
5
6  struct Info {
7      int x, y;
8      Info(int _x = 0, int _y = 0):x(_x), y(_y) {}
9  };
10
11 Info operator+(const Info& a, const Info& b) {
12     if(a.x == b.x) {
13         return {a.x, (a.y + b.y) % mod};
14     } else {
15         if(a.x > b.x) {
16             return a;
17         } else {
18             return b;
19         }
20     }
21 }
22
23 template<typename T, typename OP = plus<T>>
24 struct fenwick {
25     const int n;
26     vector<T> c;
27     const OP op;
28     fenwick(int _n) : n(_n), c(_n + 1), op(OP()) {}
29     void add(int pos, T v) {
30         for (int i = pos; i <= n; i += i & -i) {
31             c[i] = op(c[i], v);
32         }
33     }
34     T query(int pos) {
35         T res;
36         for (int i = pos; i; i -= i & -i) {
37             res = op(res, c[i]);
38         }
39         return res;
40     }
41 };

```

```

42
43
44 void discrete(vector<int>& a) {
45     vector<int> b = a;
46     sort(b.begin(), b.end());
47     b.erase(unique(b.begin(), b.end()), b.end());
48     for (size_t i = 0; i < a.size(); i++) {
49         a[i] = lower_bound(b.begin(), b.end(), a[i]) - b.begin() + 1;
50     }
51 }
52
53 int main() {
54     ios::sync_with_stdio(false);
55     cin.tie(nullptr);
56     int n;
57     cin >> n;
58     vector<int> a(n);
59     for (auto &x: a) cin >> x;
60     discrete(a);
61     vector<int> f(n, 1), dp(n, 1);
62
63     fenwick<Info> fen(n);
64
65     int ans = 0, LIS = 0;
66     for (int i = 0; i < n; i++) {
67         // cout << "i = " << i << '\n';
68
69         auto t = fen.query(a[i] - 1);
70         f[i] = t.x + 1;
71         dp[i] = max(1, t.y);
72         fen.add(a[i], {f[i], dp[i]});
73         LIS = max(f[i], LIS);
74     }
75
76
77     for (int i = 0; i < n; i++) {
78         if(f[i] == LIS)
79             (ans += dp[i]) %= mod;
80     }
81     cout << ans << '\n';
82     return 0;
83 }

```

## 5 字符串

### 5.1 字符串 Hash

```

1 #include <bits/stdc++.h>
2 using namespace std;

```

```

3  struct Hash {
4      using ull = unsigned long long;
5      const int base = 131;
6      int siz;
7      vector<ull> pow_base, hash_val; // or p, h due to time budget
8      Hash() { }
9      Hash(const string &s) {
10         siz = s.size();
11         pow_base.resize(siz);
12         hash_val.resize(siz);
13         pow_base[0] = 1;
14         hash_val[0] = s[0];
15         for (int i = 1; i < siz; i++){
16             pow_base[i] = pow_base[i - 1] * base;
17             hash_val[i] = hash_val[i - 1] * base + s[i];
18         }
19     }
20     // 下标 0 开始, 闭区间
21     ull operator[](const array<int, 2>& range) const {
22         // if(r < l || l > n) return 0; //根据题目需要处理边界情况
23         auto l = range[0], r = range[1];
24         if(l == 0) return hash_val[r];
25         return hash_val[r] - hash_val[l - 1] * pow_base[r - l + 1];
26     }
27
28     ull get(int l, int r) {
29         return this->operator[]({l, r});
30     }
31 };
32
33 struct doubleHash {
34     using ll = long long;
35     int size;
36     array<int, 2> mod = {2000000011, 2000000033}, base = {20011, 20033};
37     vector<array<ll, 2>> hash, pow_base;
38     doubleHash() { }
39     doubleHash(const string& s) {
40         size = s.size();
41         hash.resize(size);
42         pow_base.resize(size);
43         pow_base[0][0] = pow_base[0][1] = 1;
44         hash[0][0] = hash[0][1] = s[0];
45         for(int i = 1; i < size; i++){
46             hash[i][0] = (hash[i - 1][0] * base[0] + s[i]) % mod[0];
47             hash[i][1] = (hash[i - 1][1] * base[1] + s[i]) % mod[1];
48             pow_base[i][0] = pow_base[i - 1][0] * base[0] % mod[0];
49             pow_base[i][1] = pow_base[i - 1][1] * base[1] % mod[1];
50         }
51     }

```

```

52     array<ll, 2> operator[](const array<int, 2>& range) const {
53         auto l = range[0], r = range[1];
54         if(l == 0) return hash[r];
55         return {
56             (hash[r][0] - hash[l - 1][0] * pow_base[r - l + 1][0] % mod[0] + mod[0]) %
57             ↪ mod[0],
58             (hash[r][1] - hash[l - 1][1] * pow_base[r - l + 1][1] % mod[1] + mod[1]) %
59             ↪ mod[1]};
60     }
61     //double hash to A hash_val
62     ll get(int l, int r) {
63         auto h = this->operator[]({l, r});
64         return h[0] * 1000000000ll + h[1];
65     }
66 };
67 int main() {}

```

## 5.2 Trie

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  const int N = 1e5 + 10;
5  char str[N];
6  int son[N][26], cnt[N], idx;
7
8  void insert(char *str) {
9      int p = 0;
10     for (int i = 0; str[i]; i++) {
11         int u = str[i] - 'a';
12         if(!son[p][u]) son[p][u] = ++idx;
13         p = son[p][u];
14     }
15     ++cnt[p];
16 }
17
18 int query(char *str) {
19     int p = 0;
20     for (int i = 0; str[i]; ++i) {
21         int u = str[i] - 'a';
22         if(!son[p][u]) return 0;
23         p = son[p][u];
24     }
25     return cnt[p];
26 }

```

### 5.3 KMP

```

1 //poj2406
2 #include <bits/stdc++.h>
3 using namespace std;
4 const int N = 1e6 + 10;
5 char s[N];
6 int nxt[N], n;
7 /*
8 //区间 [l, r] 的 kmp
9     nxt[l] = 0;
10    for (int i = l + 1; i <= r; i++) {
11        int j = nxt[i - 1];
12        while(j && s[i] != s[l + j]) j = nxt[l + j - 1];
13        if(s[i] == s[l + j]) j++;
14        nxt[i] = j;
15    }
16 */
17 void get_nxt() {
18     nxt[1] = 0;
19     for (int i = 2, j = 0; i <= n; i++) {
20         while(j && s[i] != s[j + 1]) j = nxt[j];
21         if(s[i] == s[j + 1]) j++;
22         nxt[i] = j;
23     }
24 }
25
26 int main() {
27     while(~scanf("%s", s + 1)) {
28         if(s[1] == '.') break;
29         n = strlen(s + 1);
30         get_nxt();
31         int period = n - nxt[n];
32         if(n % period == 0) printf("%d\n", n / period);
33         else puts("1");
34     }
35     return 0;
36 }

```

### 5.4 Z-algorithm

- 给出字符串  $a, b$ , 求  $a$  的每个后缀与  $b$  的 LCP:  
 设  $\$$  为字符集外字符, 求  $b + \$ + a$  的 Z 函数, 则  $a$  的后缀  $a[i..]$  与  $b$  的 LCP 为  $Z(|b| + 1 + i)$ 。
- 求  $s$  的每个前缀的出现次数:  
 求  $s$  的 Z 函数。对于每一个  $i$ , 如果  $Z(i)$  不等于 0, 说明长度为  $Z(i), Z(i) - 1, \dots, 1$  的前缀在此处各出现了一次, 所以求一个后缀和即可。在这个问题中一般令  $Z(0) = |s|$ 。

```
for (int i = n + 1; i < 2 * n + 1; ++i)
```

```

        S[z[i]]++;
    for (int i = n; i >= 1; --i)
        S[i] += S[i + 1];

```

- 求  $s$  的所有 border:

KMP 就可以，也可以用 Z 算法。求  $s$  的 Z 函数。对于每一个  $i$ ，如果  $i + Z(i) = |s|$ ，说明这个 Z-Box 对应一个 border。（注：与 KMP 不同，这里只是求所有 border，不是求所有前缀的 border）

```

1 //给定两个字符串 a,b,
2 // 要求出两个数组: b 的 z 函数数组 z、
3 // b 与 a 的每一个后缀的 LCP 长度数组 p。
4 #include <bits/stdc++.h>
5 #define rep(i, a, b) for (int i = (a); i < (b); i++)
6 #define sz(a) int((a).size())
7 using namespace std;
8 using ll = long long;
9 const int N = 2e7;
10 ll ansz, ansp;
11 string a, b;
12
13 // Zfunction
14 int z[N << 1];
15 void getz(string s) {
16     int l = 0;
17     for (int i = 1; i <= s.size(); i++) {
18         if (l + z[l] > i) z[i] = min(z[i - 1], l + z[l] - i);
19         while (i + z[i] < s.size() && s[z[i]] == s[i + z[i]]) z[i]++;
20         if (i + z[i] > l + z[l]) l = i;
21     }
22     // rep(i,0,s.size()) cout<<z[i]<<" ";cout<<'\\n';
23 }
24
25
26 int main(){
27     ios::sync_with_stdio(0);
28     cin.tie(0),cout.tie(0);
29     cin >> a >> b, getz(b + a);
30     ansz ^= 1ll * (sz(b)+1)*(0+1);
31     rep(i,1,sz(b)) ansz^=1ll*(min(z[i],sz(b)-i)+1)*(i+1);
32     rep(i,0,sz(a)) ansp^=1ll*(min(z[i+sz(b)],sz(b))+1)*(i+1);
33     cout << ansz << '\\n'
34         << ansp << '\\n';
35     return 0;
36 }

```

## 5.5 AC 自动机

```

1 #include <bits/stdc++.h>
2

```

```

3  using namespace std;
4
5  constexpr int N = 2e5;
6
7  struct ACM {
8
9      int state_num;
10     struct Node {
11         array<int, 26> nxt;
12         int fail;
13         Node(): nxt{}, fail(0) {}
14     };
15     vector<Node> vec;
16     ACM(int n) : vec(n + 1) {}
17     int insert(const string& s) {
18         int p = 0;
19         for (auto ch: s) {
20             int &u = vec[p].nxt[ch - 'a'];
21             if(!u) {
22                 u = ++state_num;
23             }
24             p = u;
25         }
26         return p;
27     }
28     void build() {
29         queue<int> q;
30         for(auto i: vec[0].nxt) {
31             if(i) q.push(i);
32         }
33         while(q.size()) {
34             int u = q.front(); q.pop();
35             for (int i = 0; i < 26; i++) {
36                 int &v = vec[u].nxt[i];
37                 int w = vec[vec[u].fail].nxt[i];
38                 if(v) {
39                     vec[v].fail = w;
40                     q.push(v);
41                 } else {
42                     v = w;
43                 }
44             }
45         }
46     }
47 };
48
49 void solve() {
50     int n;

```

```

52     cin >> n;
53     vector<string> patterns(n);
54     ACM ac(N);
55
56     vector<int> id(n); // id[i] 表示第 i 个字符串结尾的 state_num
57     for (int i = 0; i < n; i++) {
58         cin >> patterns[i];
59         id[i] = ac.insert(patterns[i]);
60     }
61     ac.build();
62     // cerr << "state_num == " << ac.state_num << '\n';
63     // for (int i = 1; i <= ac.state_num; i++) {
64     //     cerr << i << ".fail = " << ac.vec[i].fail << "\n";
65     // }
66     // cerr << "/////////////////" << '\n';
67
68     vector<vector<int>> G(ac.state_num + 1);
69     for (int i = 1; i <= ac.state_num; i++) {
70         G[ac.vec[i].fail].push_back(i);
71     }
72
73     string t;
74     cin >> t;
75
76     int u = 0;
77     vector<int> cnt(ac.state_num + 1);
78     for (auto ch: t) {
79         u = ac.vec[u].nxt[ch - 'a'];
80         cnt[u]++;
81         // cerr << u << '\n';
82     }
83
84     function<void(int)> dfs = [&](int u) {
85         for (auto v: G[u]) {
86             dfs(v);
87             cnt[u] += cnt[v];
88         }
89     };
90     dfs(0);
91     // cerr << "/////////////////" << '\n';
92     for (int i = 0; i < n; i++) {
93         cout << cnt[id[i]] << '\n';
94     }
95
96 }
97
98 int main() {
99     ios::sync_with_stdio(false);
100    cin.tie(nullptr);

```



```

101     int T = 1; // cin >> T;
102     while(T --) {
103         solve();
104     }
105     return 0;
106 }

```

## 5.6 SA

$lcp(i, j)$  表示后缀  $i, j$  的最长公共前缀 (的长度)

$height$  数组定义:  $ht[i] = lcp(sa[i], sa[i - 1])$

性质:  $lcp(sa[i], sa[j]) = \min\{ht[i + 1..j]\}$

由此, 求两子串 (排名为  $i, j$ ) 最长公共前缀就转化为了  $RMQ$  问题 (求  $ht[i + 1]$  到  $ht[j]$  的最小值)。

本质不同的子串:  $\frac{n*(n+1)}{2} - \sum_{i=2}^n ht[i]$

$ht$  数组连续一段不小于  $h$  的区间长度代表长  $h$  的这个子串的出现次数

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  class SuffixArray {
5  //得到的 sa[], rk[] 下标从 0 开始, ht 下标从 1 开始 (因为是长度)
6  private:
7      int n, m;
8      vector<int> x, y, cnt;
9      void radixSort() {
10         for (int i = 0; i < m; ++ i) cnt[i] = 0;
11         for (int i = 0; i < n; ++ i) cnt[x[i]] ++;
12         for (int i = 1; i < m; ++ i) cnt[i] += cnt[i - 1];
13         for (int i = n - 1; i >= 0; -- i) sa[-- cnt[x[y[i]]]] = y[i];
14     }
15 public:
16     vector<int> sa, rk, ht;
17
18     SuffixArray(const string &s) :
19         n(s.size()), m(256), //m 为字符集最大数量
20         x(n), y(n), cnt(max(n, m)),
21         sa(n), rk(n), ht(n) {
22         init_sa(s);
23         init_ht(s);
24     }
25     void init_sa(const string &s) {
26         for (int i = 0; i < n; ++ i) {
27             x[i] = s[i];
28             y[i] = i;
29         }
30         radixSort();
31         for (int w = 1; w <= n; w <= 1) {
32             int p = 0;

```

```

33     for (int i = n - w; i < n; ++ i) y[p ++] = i;
34     for (int i = 0; i < n; ++ i)
35         if (sa[i] >= w) y[p ++] = sa[i] - w;
36
37     radixSort();
38     swap(x, y);
39     x[sa[0]] = 0;
40     p = 1;
41     auto cmp = [&](int i, int j) {
42         if (i < n && j < n) return y[i] == y[j];
43         return i >= n && j >= n;
44     };
45     for (int i = 1; i < n; ++ i)
46         x[sa[i]] = (cmp(sa[i], sa[i - 1]) && cmp(sa[i - 1] + w, sa[i] + w))
47             ? p - 1 : p++;
48
49     if (p >= n) break;
50     m = p;
51 }
52 for (int i = 0; i < n; ++ i) rk[sa[i]] = i;
53 }
54 void init_ht(const string &s) {
55     for (int i = 0, k = 0; i < n; ++ i) {
56         if (rk[i] == 0) continue ;
57         if (k) k --;
58         int j = sa[rk[i] - 1];
59         while (i + k < n && j + k < n && s[i + k] == s[j + k]) k ++;
60         ht[rk[i]] = k;
61     }
62 }
63 };
64
65 void solve() {
66     string s;
67     cin >> s;
68     SuffixArray f(s);
69     for (int i = 0; i < s.size(); ++ i)
70         cout << f.sa[i] + 1 << " \n"[i + 1 == s.size()];
71     for (int i = 0; i < s.size(); ++ i)
72         cout << f.ht[i] << " \n"[i + 1 == s.size()];
73 }
74
75 /* nlogn 版
76
77 int n, m;
78 char s[N];
79 int sa[N], rk[N], ht[N], x[N], id[N], cnt[N];
80
81 void init_sa() {

```

```

82     m = 300;
83     for (int i = 1; i <= m; i++) cnt[i] = 0;
84     for (int i = 1; i <= n; i++) ++ cnt[x[i] = s[i]];
85     for (int i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
86     for (int i = n; i; i--) sa[cnt[x[i]]--] = i;
87     for (int w = 1; w <= n; w <= 1) {
88         int num = 0;
89         for (int i = n - w + 1; i <= n; i++) id[++ num] = i;
90         for (int i = 1; i <= n; i++)
91             if (sa[i] > w) id[++ num] = sa[i] - w;
92         for (int i = 1; i <= m; i++) cnt[i] = 0;
93         for (int i = 1; i <= n; i++) cnt[x[i]] ++;
94         for (int i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
95         for (int i = n; i; i--) sa[cnt[x[id[i]]] --] = id[i], id[i] = 0;
96     for(int i = 1; i <= n; i++) swap(x[i], id[i]);
97     x[sa[1]] = 1, num = 1;
98     auto cmp = [&](int a, int b, int w) {
99         return id[a] == id[b] && id[a + w] == id[b + w];
100     };
101     for (int i = 2; i <= n; i++)
102         x[sa[i]] = cmp(sa[i], sa[i - 1], w) ? num : ++num;
103     if (n == num) break;
104     m = num;
105 }
106 for (int i = 1; i <= n; i++) rk[sa[i]] = i;
107 }
108
109 */

```

## 5.7 SAM

1. SAM 的总状态数不超过  $2n - 1$ ，当字符串形如 *abbb...* 时取到上界
2. SAM 的总转移数不超过  $3n - 4$ ，当字符串形如 *abbb...bc* 时取到上界
3. 构建 SAM 的复杂度为  $|\Sigma|n$ ， $\Sigma$  为字符集。（瓶颈是复制节点，所以如果字符集过大，可以考虑用 `map` 代替数组）

SAM 的总转移数不超过，当字符串形如时取到上界构建 SAM 的复杂度为，为字符集。（瓶颈是复制节点，所以如果字符集过大，可以考虑用 `map` 代替数组）

```

1 // https://ac.nowcoder.com/acm/contest/33188/H
2 #include <bits/stdc++.h>
3
4 using namespace std;
5 using i64 = long long;
6
7 struct SuffixAutomaton {
8     static constexpr int ALPHABET_SIZE = 26, N = 1e5;
9     struct Node {

```

```

10     int len, link;
11     int next[ALPHABET_SIZE];
12     Node() : len(0), link(0), next{} {}
13 } t[2 * N];
14 int cntNodes;
15 SuffixAutomaton() {
16     cntNodes = 1;
17     fill(t[0].next, t[0].next + ALPHABET_SIZE, 1);
18     t[0].len = -1;
19 }
20 int extend(int p, int c) {
21     if (t[p].next[c]) {
22         int q = t[p].next[c];
23         if (t[q].len == t[p].len + 1)
24             return q;
25         int r = ++cntNodes;
26         t[r].len = t[p].len + 1;
27         t[r].link = t[q].link;
28         copy(t[q].next, t[q].next + ALPHABET_SIZE, t[r].next);
29         t[q].link = r;
30         while (t[p].next[c] == q) {
31             t[p].next[c] = r;
32             p = t[p].link;
33         }
34         return r;
35     }
36     int cur = ++cntNodes;
37     t[cur].len = t[p].len + 1;
38     while (!t[p].next[c]) {
39         t[p].next[c] = cur;
40         p = t[p].link;
41     }
42     t[cur].link = extend(p, c);
43     return cur;
44 }
45 };
46
47 int main() {
48     ios::sync_with_stdio(false);
49     cin.tie(nullptr);
50
51     int n, m, k;
52     cin >> n >> m >> k;
53
54     string A;
55     cin >> A;
56     int p = 1;
57     SuffixAutomaton sam;
58     for (auto c : A) {

```

```

59     p = sam.extend(p, c - 'a');
60 }
61
62 vector<int> w(m);
63 for (int i = 0; i < m; i++) {
64     cin >> w[i];
65 }
66
67 vector<i64> s(m + 1);
68 for (int i = 0; i < m; i++) {
69     s[i + 1] = s[i] + w[i];
70 }
71
72 while (k--) {
73     string B;
74     cin >> B;
75
76     int p = 1;
77     i64 ans = 0;
78     deque<int> q{0};
79     for (int i = 0, j = 0, len = 0; i < m; i++) {
80         while (j < m && sam.t[p].next[B[j] - 'a']) {
81             p = sam.t[p].next[B[j] - 'a'];
82             len++;
83             j++;
84             while (!q.empty() && s[j] > s[q.back()]) {
85                 q.pop_back();
86             }
87             q.push_back(j);
88         }
89         while (q.front() < i) {
90             q.pop_front();
91         }
92         ans = max(ans, s[q.front()] - s[i]);
93         len--;
94         if (len <= sam.t[sam.t[p].link].len) {
95             p = sam.t[p].link;
96         }
97     }
98     cout << ans << "\n";
99 }
100
101 return 0;
102 }

```

## 5.8 Manacher

用 *Manacher* + *hash* 可以求出所有本质不同的回文子串 (存 *hash* 值), 时间复杂度  $\mathcal{O}(|s|)$ 。但是不用于求每个本质不同回文子串出现次数相关统计, 因为统计出现次数时,  $while(l \leq r)$  中

不可以 *break*, 复杂度  $n^2$

```

    auto p = manacher(s);
    Hash hs(s); //or doubleHash
    set<ull> res; // ll when doubleHash
    for (int mid = 1; mid < p.size() - 1; mid++) {
        //枚举回文子串的左右端点
        int l = (mid - p[mid] + 1) / 2, r = (mid + p[mid] - 1) / 2;
        while(l <= r) {
            if(res.count(hash.get(l, r))) break;
            res.insert(hash.get(l++, r--));
        }
    }
}

1  #include<bits/stdc++.h>
2  using namespace std;
3  // return p, p[i] 表示修改后的串中以 i 为中心的最长回文半径
4  vector<int> manacher(const string& _s) {
5      vector<int> p(_s.size() * 2 + 1);
6      string s(_s.size() * 2 + 1, '$');
7      for (int i = 0; i < _s.size(); i++) s[2 * i + 1] = _s[i];
8      for(int i = 0, maxr = 0, mid = 0; i < s.size(); i++) {
9          if(i < maxr) p[i] = min(p[mid * 2 - i], maxr - i);
10         while(i - p[i] - 1 >= 0 && i + p[i] + 1 < s.size()
11             && s[i - p[i] - 1] == s[i + p[i] + 1])
12             ++p[i];
13         if(i + p[i] > maxr) maxr = i + p[i], mid = i;
14     }
15     return p;
16 }
17
18
19 int main() {
20     string s;
21     cin >> s;
22     auto p = manacher(s);
23     // for (int i = 0; i < p.size(); i++) {
24     //     cout << p[i] << " \n"[i == p.size() - 1];
25     // }
26     cout << (*max_element(p.begin(), p.end())) << endl;
27 }

```

## 6 其他

### 6.1 glibc 内置函数

```

1 // Returns the number of 1-bits in x.
2 int __builtin_popcount(unsigned int x);

```

```

3  int __builtin_popcountll(unsigned long long x);
4
5  // Returns the number of trailing 0 (undefined when x == 0)
6  int __builtin_ctz(unsigned int x);
7  int __builtin_ctzll(unsigned long long x);
8
9  // Returns log_2(x)
10 int __lg(int x);
11
12 int __gcd(int x, int y);

```

## 6.2 \_\_int128 读写

```

1  inline __int128 read(){
2      __int128 x = 0, f = 1;
3      char ch = getchar();
4      while (ch < '0' || ch > '9') { if(ch == '-') f = -1; ch = getchar(); }
5      while (ch >= '0' && ch <= '9') { x = x * 10 + ch - '0'; ch = getchar(); }
6      return x * f;
7  }
8
9  inline void print(__int128 x) {
10     if(x < 0) { putchar('-'); x = -x; }
11     if(x > 9) print(x / 10);
12     putchar(x % 10 + '0');
13 }

```

## 6.3 大整数运算

```

1  #include <iostream>
2  #include <vector>
3
4  using namespace std;
5
6  vector<int> add(vector<int> &A, vector<int> &B) {
7      if (A.size() < B.size()) return add(B, A);
8
9      vector<int> C;
10     int t = 0;
11     for (int i = 0; i < A.size(); i++) {
12         t += A[i];
13         if (i < B.size()) t += B[i];
14         C.push_back(t % 10);
15         t /= 10;
16     }
17     if (t) C.push_back(t);
18     return C;
19 }
20

```

```

21 int main() {
22     string a, b;
23     vector<int> A, B;
24     cin >> a >> b;
25     for (int i = a.size() - 1; i >= 0; i -- ) A.push_back(a[i] - '0');
26     for (int i = b.size() - 1; i >= 0; i -- ) B.push_back(b[i] - '0');
27
28     auto C = add(A, B);
29     for (int i = C.size() - 1; i >= 0; i -- ) cout << C[i];
30     cout << endl;
31
32     return 0;
33 }
34
35 //减法
36
37 bool cmp(vector<int> &A, vector<int> &B) {
38     if (A.size() != B.size()) return A.size() > B.size();
39
40     for (int i = A.size() - 1; i >= 0; i -- )
41         if (A[i] != B[i])
42             return A[i] > B[i];
43     return true;
44 }
45
46 vector<int> sub(vector<int> &A, vector<int> &B) {
47     vector<int> C;
48     for (int i = 0, t = 0; i < A.size(); i ++ ) {
49         t = A[i] - t;
50         if (i < B.size()) t -= B[i];
51         C.push_back((t + 10) % 10);
52         if (t < 0) t = 1;
53         else t = 0;
54     }
55     while (C.size() > 1 && C.back() == 0) C.pop_back();
56     return C;
57 }
58
59 int main() {
60     string a, b;
61     vector<int> A, B;
62     cin >> a >> b;
63     for (int i = a.size() - 1; i >= 0; i -- ) A.push_back(a[i] - '0');
64     for (int i = b.size() - 1; i >= 0; i -- ) B.push_back(b[i] - '0');
65     vector<int> C;
66     if (cmp(A, B)) C = sub(A, B);
67     else C = sub(B, A), cout << '-';
68
69     for (int i = C.size() - 1; i >= 0; i -- ) cout << C[i];

```



```

70     cout << endl;
71     return 0;
72 }
73
74 // 乘法
75 // 高精相乘见 fft
76
77 vector<int> mul(vector<int> &A, int b) {
78     vector<int> C;
79     int t = 0;
80     for (int i = 0; i < A.size() || t; i++) {
81         if (i < A.size()) t += A[i] * b;
82         C.push_back(t % 10);
83         t /= 10;
84     }
85     while (C.size() > 1 && C.back() == 0) C.pop_back();
86     return C;
87 }
88
89 int main() {
90     string a;
91     int b;
92     cin >> a >> b;
93     vector<int> A;
94     for (int i = a.size() - 1; i >= 0; i--) A.push_back(a[i] - '0');
95     auto C = mul(A, b);
96     for (int i = C.size() - 1; i >= 0; i--) printf("%d", C[i]);
97     return 0;
98 }
99
100 vector<int> div(vector<int> &A, int b, int &r) {
101     vector<int> C;
102     r = 0;
103     for (int i = A.size() - 1; i >= 0; i--) {
104         r = r * 10 + A[i];
105         C.push_back(r / b);
106         r %= b;
107     }
108     reverse(C.begin(), C.end());
109     while (C.size() > 1 && C.back() == 0) C.pop_back();
110     return C;
111 }
112
113 int main() {
114     string a;
115     vector<int> A;
116     int B;
117     cin >> a >> B;
118     for (int i = a.size() - 1; i >= 0; i--) A.push_back(a[i] - '0');

```

```

119     int r;
120     auto C = div(A, B, r);
121     for (int i = C.size() - 1; i >= 0; i -- ) cout << C[i];
122     cout << endl << r << endl;
123     return 0;
124 }

```

## 6.4 整数二分

```

1 // 区间 [l, r] 被划分成 [l, ans] 和 [ans + 1, r] 时使用:
2 int bsearch_1(int l, int r) {
3     while (l < r) {
4         int mid = l + r >> 1;
5         if (check(mid)) r = mid;    // check() 判断 mid 是否满足性质
6         else l = mid + 1;
7     }
8     return l;
9 }
10 // 区间 [l, r] 被划分成 [l, ans - 1] 和 [ans, r] 时使用:
11 int bsearch_2(int l, int r) {
12     while (l < r) {
13         int mid = l + r + 1 >> 1;
14         if (check(mid)) l = mid;
15         else r = mid - 1;
16     }
17     return l;
18 }

```

## 6.5 单调栈

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int N = 1000100;
4 //单调栈, 记录每个数左边比他小 (大) 的第一个数 (也可以记录其下标)
5 int stk[N], tt, a[N];
6
7 int main() {
8     ios::sync_with_stdio(false), cin.tie(0), cout.tie(0);
9     int n; cin >> n;
10    for (int i = 1; i <= n; i ++ ) cin >> a[i];
11
12    for (int i = 1; i <= n; i++) {
13        while(tt && stk[tt] >= a[i]) tt--;
14        if(tt) cout << stk[tt] << ' ';
15        else cout << -1 << ' ';
16        stk[++tt] = a[i];
17    }
18    return 0;
19 }

```

## 6.6 单调队列

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  const int N = 1e6 + 10;
4  int a[N], q[N], n, k;
5  //滑动窗口
6  int main() {
7      cin >> n >> k;
8      for(int i = 0; i < n; i++) cin >> a[i];
9      int hh = 0, tt = -1;
10     for(int i = 0; i < n; i++) {
11         //判断队头是否已经划出窗口
12         if( hh <= tt && i - k + 1 > q[hh]) hh++;
13         while(hh <= tt && /* 后面改成要维护的最小值 */a[q[tt]] >= a[i]) tt -- ;//求区间最小
14         q[ ++ tt ] = i;
15         if(i >= k-1) printf("%d ",a[q[hh]]);
16     }
17     return 0;
18 }

```

## 6.7 矩阵快速幂

```

1  template<typename T>
2  struct matrix {
3      size_t dim;
4      vector<vector<T>> mtx;
5
6      matrix(int n) : dim(n), mtx(n, vector<T>(n, 0)){}
7      void I() {
8          for (size_t i = 0; i < dim; i++) {
9              for (size_t j = 0; j < dim; j++) {
10                 mtx[i][j] = static_cast<T>(i == j);
11             }
12         }
13     }
14     matrix operator*(const matrix &rhs) {
15         assert(this->dim == rhs.dim && "Matrix dimension must be the same.");
16         matrix<T> res(dim);
17         for (size_t i = 0; i < dim; i++) {
18             for (size_t j = 0; j < dim; j++) {
19                 for (size_t k = 0; k < dim; k++) {
20                     res.mtx[i][j] += (mtx[i][k] * rhs.mtx[k][j]) % mod;
21                     res.mtx[i][j] %= mod;
22                 }
23             }
24         }
25         return res;
26     }
27 }

```

```

28     matrix operator^(ll n) {
29         matrix<T> res(dim);
30         res.I();
31         for (; n; n >>= 1) {
32             if(n & 1) res = res * (*this);
33             *this = *this * (*this);
34         }
35         return res;
36     }
37
38 };

```

## 6.8 GossersHack

生成  $n$  元集合所有  $k$  元子集的算法。这个算法复杂度与答案个数是同阶的，比暴力枚举  $2^n$  个数然后分别算 *popcount* 要好。

```

1 void GossersHack(int k, int n) {
2     int cur = (1 << k) - 1;
3     int limit = (1 << n);
4     while (cur < limit) {
5         // do something
6         int lb = cur & -cur;
7         int r = cur + lb;
8         cur = ((r ^ cur) >> __builtin_ctz(lb) + 2) | r;
9         // 或: cur = ((r ^ cur) >> 2) / lb | r;
10    }
11 }

```

## 6.9 C++17-STL

```

1 #include <bits/stdc++.h>
2 using ll = long long;
3 using namespace std;
4
5 /*
6  Lambda expression
7  */
8
9 /* structure binding */
10 {
11     // Graph with weighted edges
12     vector<vector<pair<int, int>>> G(n);
13     // somewhere in dfs,
14     for (auto [v, w] : G[u]) {
15         ...
16     }
17 }
18
19 {

```

```

20 // using std::tie for multi key compare in structure:
21 struct Node{
22     int x, y, z;
23     bool operator<(const Node& rhs) const {
24         return tie(x, y, z) < tie(rhs.x, rhs.y, rhs.z);
25     }
26 };
27 }
28
29 /* discrete manipulate*/
30 {
31     // vector<int> v. vv = v;
32     sort(v.begin(), v.end());
33     v.erase(unique(v.begin(), v.end()), v.end());
34 }
35
36
37 /*
38 some useful function in STL
39 */
40 {
41     int n = 256;
42     vector<int> v(n);
43     // generate v as 0, 1, 2, ...
44     // often used when you initialize DSU or generate a permutation
45     iota(v.begin(), v.end(), 0);
46
47     ll res = accumulate(v.begin(), v.end(), 1LL,
48         [](int a, int b){return (ll)a*b;});
49
50     gcd() and lcm(); // not __gcd() version
51
52     // sum of k largest numbers in linear time
53     nth_element(a.begin(), a.end(), k);
54     ll ans = accumulate(a.begin(), a.begin() + k, 0LL);
55
56     // max_element and min_element which can be used with user-defined op
57     cout << *max_element(v.begin(), v.end() /*[]()->bool{}/);
58
59     //std::clamp
60     // Returns x if it is in the interval [low, high] or,
61     // otherwise, the nearest value.No more max of min of max of...
62     cout << clamp(7, 0, 10); //7
63     cout << clamp(7, 0, 5); //5
64     cout << clamp(7, 10, 50); //10
65
66     /*don't do*/ max(ans, max(t1, t2));
67     /*just*/ max({ans, t1, t2}); // initalizer list
68

```

```

69
70  /* partial_sum and adjacent_difference 前缀和与差分 */
71  vector<int> a(n, 2), b(n, 2);
72  partial_sum(a.begin(), a.end(), a.begin());
73  partial_sum(a.begin(), a.end(), b.begin());
74
75  adjacent_difference(v.begin(), v.end(), v.begin());
76
77  count(v.begin(), v.end(), target);
78  count_if(v.begin(), v.end(), [](int x) {x % 4 == 0});
79
80
81  // 常用在分治中的一些 STL: inplace_merge, merge, partition, stable_partition
82  // Merges two consecutive sorted ranges [first, middle) and [middle, last)
83  // into one sorted range [first, last).
84  void inplace_merge(BidirIt first, BidirIt middle, BidirIt last);
85  // example
86  template<class Iter>
87  void merge_sort(Iter first, Iter last) {
88      if (last - first > 1) {
89          Iter middle = first + (last - first) / 2;
90          merge_sort(first, middle);
91          merge_sort(middle, last);
92          std::inplace_merge(first, middle, last);
93      }
94  }
95  // used in merge, cdq
96
97  // partition, 将满足条件 p 的元素放在前部分
98  template <class ForwardIt, class UnaryPredicate>
99  constexpr ForwardIt partition(ForwardIt first, ForwardIt last, UnaryPredicate p);
100
101  // partition 的 stable 版本, 常在分支中使用
102  template <class BidirIt, class UnaryPredicate>
103  BidirIt stable_partition(BidirIt first, BidirIt last, UnaryPredicate p);
104
105  #include <algorithm>
106  #include <array>
107  #include <iostream>
108
109  int main()
110  {
111      std::array<int, 9> v{};
112      std::iota(v.begin(), v.end(), 1);
113
114      auto is_even = [](int i){ return i % 2 == 0; };
115      std::cout.setf(std::ios_base::boolalpha);
116      std::cout << std::is_partitioned(v.begin(), v.end(), is_even) << ' ';
117

```

```

118         std::partition(v.begin(), v.end(), is_even);
119         std::cout << std::is_partitioned(v.begin(), v.end(), is_even) << ' ';
120
121         std::reverse(v.begin(), v.end());
122         std::cout << std::is_partitioned(v.cbegin(), v.cend(), is_even) << ' ';
123         std::cout << std::is_partitioned(v.crbegin(), v.crend(), is_even) << '\n';
124         // output:
125         // false true false true
126     }
127 }
128
129 /*
130 template< class R, class... Args >
131 class function<R(Args...)>;
132 */
133 {
134     // a callable obj
135     function<int(int, int)> dfs = [&](int u, int fa) {
136         ...
137     };
138 }
139
140 /* Set operations (on sorted ranges)*/
141 {
142     set<int> s1, s2;
143     set<int> ans;
144     set_intersection(s1.begin(), s1.end(),
145                     s2.begin(), s2.end(),
146                     std::inserter(ans, ans.begin()));
147     // 若为 vector, 可以用 back_inserter;
148 }
149
150 /*
151 Initializer in if and switch:
152 */
153 {
154     set<int> s;
155     if (auto [iter, ok] = s.insert(42); ok) {
156         //...
157     }
158     else {
159         //`ok` and `iter` are available here
160     }
161     //But not here
162 }
163
164 /*
165 about string
166 */

```

```

167 {
168     substr(npos, count);
169
170     string to_string(/*numeric type*/);
171
172     //将字符串按照字面转化为数字类型
173     int stoi(const string &str);
174     long long stoll(const string &str);
175     float stof(const string &str);
176     double stod(const string &str);
177     long double stold(const string &str);
178
179 }
180
181 int main() {
182 }

```

## 6.10 pbds

头文件:

```

#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>
#include<bits/stdc++.h>
using namespace std;
using namespace __gnu_pbds;

```

带有 find\_by\_order(), order\_of\_key() 的 set 和 map  
代替 std::set<int>

```

using tree = __gnu_pbds::tree<int, null_type, less<int>,
    rb_tree_tag, tree_order_statistics_node_update>;

```

代替 std::multiset<int>

```

using tree = __gnu_pbds::tree<pair<int, int>, null_type, less<pair<int, int>>,
    rb_tree_tag, tree_order_statistics_node_update>;
// 插入时: insert({value, count++});
// 查找时 lower_bound({value, 0});

```

示例

```

1  #include<ext/pb_ds/assoc_container.hpp>
2  #include<ext/pb_ds/tree_policy.hpp>
3  #include<bits/stdc++.h>
4  using namespace std;
5  using namespace __gnu_pbds;
6  using ll = long long;
7  // use this as multiset with find_by_order
8  using RBT = __gnu_pbds::tree<pair<ll, int>, null_type, less<pair<ll, int>>, rb_tree_tag,
    ⇨ tree_order_statistics_node_update>;

```



```

9  RBT rbt;
10 int t = 0;
11 int main() {
12     int n, m, k;
13     cin >> n >> m >> k;
14     std::vector<ll> a(n);
15     for (int i = 0; i < n; i++) {
16         cin >> a[i];
17     }
18
19     for (int i = 0; i < m; i++) {
20         rbt.insert({a[i], t++});
21     }
22
23     int count = 0;
24     std::vector<ll> res(n);
25     ll kth = 0;
26     for (auto it = rbt.begin(); count < k; count++, it++) {
27         res[m - 1] += it->first;
28         kth = it->first;
29     }
30
31     for (int i = m; i < n; i++) {
32         rbt.erase(rbt.lower_bound({a[i - m], 0}));
33         rbt.insert({a[i], t++});
34         auto new_kth = rbt.find_by_order(k - 1)->first;
35         if (a[i - m] <= kth && a[i] <= kth) {
36             res[i] = res[i - 1] + (-a[i - m] + a[i]);
37         } else if (a[i - m] > kth && a[i] <= kth) {
38             res[i] = res[i - 1] - kth + a[i];
39         } else if (a[i - m] <= kth && a[i] > kth) {
40             res[i] = res[i - 1] + new_kth - a[i - m];
41         }
42         else if (a[i - m] > kth && a[i] > kth) {
43             res[i] = res[i - 1];
44         }
45         kth = new_kth;
46     }
47
48     for (int i = m - 1; i < n; i++) {
49         std::cout << res[i] << ' ';
50     }
51     return 0;
52 }

```

## 6.11 一些结论

1. 维护区间加，区间查 gcd 利用  $\text{gcd}(a, b) = \text{gcd}(a-b, b)$  // 常用结论

考虑维护差分序列，则  $\text{gcd}(a[1], a[1 + 1], \dots, a[r]) = \text{gcd}(a[1], b[1 + 1], \dots,$

$b[r])$

2. 一个序列的异或和小于等于数值和  $a + b = ab + 2(a \& b)$   
一个序列的数值和和异或和奇偶性相同
3. 一般估计因子数是  $\mathcal{O}(\sqrt{n})$ , 但这个上界是比较松的。 $1e18$  内的数最多有 103680 个因子,  
 $1e9$  内有 1344 个因子
4. 除法上取整:  $(a + b - 1)/b$ , 或者  $\text{ceil}(a.0/b.0)$
5.  $\sum_{k=1}^{n-1} k \cdot k! = n! - 1$
6. 根号 trick: 若  $a[i] > 0$ ,  $\sum_{i=1}^n a[i] = n$ , 则  $a$  中不同的数只有  $\sqrt{n}$  种