

Templates for ICPC

iNx

Contents

1	Basics	7
1.1	高精度	7
2	Number Theory	11
2.1	筛法	11
2.1.1	线性筛 (欧拉筛)	11
2.1.2	杜教筛	11
2.1.3	min25 筛	12
2.2	Euclid 算法	14
2.2.1	最大公约数算法	14
2.2.2	扩展欧几里得	15
2.2.3	类欧几里得	15
2.3	中国剩余定理 (CRT)	15
2.3.1	CRT	15
2.3.2	EX-CRT	15
2.4	Lucas 定理	16
2.4.1	Lucas	16
2.4.2	EX-Lucas	17
2.5	原根	19
2.6	离散对数问题 (DLP)	20
2.6.1	BSGS(Baby-step Giant-step)	20
3	Math	23
3.1	公式 (Formula)	23
3.1.1	切/曼距离	23
3.1.2	贝尔级数	23
3.2	数值积分	24
3.3	康拓展开	24
4	Polynomial	27
4.1	快速傅立叶变换 FFT	27
4.2	快速数论变换 NTT	28
5	Geometry	31
5.1	基础知识	31
5.1.1	Point 类	31
5.1.2	定比分点 (求两直线交点)	32
5.1.3	Line 类	32
5.1.4	距离	32

5.1.5	判断点在线上	32
5.1.6	交点	33
5.1.7	与直线位置	33
5.1.8	线性变换	33
5.1.9	对称	33
5.2	圆 (Circle)	34
5.2.1	Circle 类	34
5.2.2	两圆交点	34
5.2.3	两圆公切线	35
5.2.4	三角形外接圆	35
5.2.5	最小圆覆盖 (Minimum Enclosing Circle)	35
5.3	多边形 (Polygon)	36
5.3.1	面积 (Area)	36
5.3.2	凸包 (Convex Hull)	36
5.3.3	旋转卡壳	37
5.3.4	半平面交	37
6	Linear Algebra	39
6.1	异或线性基	39
6.2	矩阵 (Matrix)	39
6.3	高斯约当消元	40
7	Graph	41
7.1	树	41
7.1.1	直径	41
7.1.2	重心	41
7.1.3	树上差分	42
7.1.4	重链剖分	42
7.1.5	树上启发式合并	43
7.1.6	虚树	45
7.1.7	点分治	46
7.2	差分约束	50
7.3	强连通分量 SCC	50
7.3.1	tarjan 算法	50
7.4	双连通分量 BCC	51
7.4.1	点双	51
7.4.2	割点	52
7.4.3	边双连通分量/桥	52
7.5	2-SAT	53
7.6	斯坦纳树	54
7.7	网络流	56
7.7.1	最大流/最小割	56
7.7.2	最小费用最大流 (MCMF)	57
8	Data Structure	59
8.1	并查集 (Union-Find)	59
8.2	哈希表	59
8.3	Splay Tree	60
8.4	Treap (Tree-heap)	63

9 String Theory	67
9.1 AC 自动机	67
9.2 前缀函数 (Prefix-Function)	69
9.3 KMP 自动机	69
9.4 后缀数组 (SA)	70
9.4.1 后缀排序	70
9.4.2 最长公共前缀 (LCP)	71
9.5 后缀自动机 (SAM)	71
9.6 Manacher	72

Chapter 1

Basics

1.1 高精度

```
1 struct Bign {
2     int len, sgn, w[MAX_L];
3     void clear() {
4         len = sgn = 1;
5         memset(w, 0, sizeof(w));
6     }
7     Bign operator=(int x) {
8         clear();
9         if (x != 0) {
10             if (x < 0) {
11                 sgn = -1, x = -x;
12             }
13             len = 0;
14             while (x) {
15                 w[len++] = x % 10;
16                 x /= 10;
17             }
18         }
19         return *this;
20     }
21     Bign operator=(string s) {
22         clear();
23         reverse(s.begin(), s.end());
24         if (s.back() == '-') {
25             sgn = -1, s.pop_back();
26         }
27         for (int i = 0; i < s.size(); i++) {
28             w[i] = s[i] - '0';
29         }
30         len = s.size();
31         return *this;
32     }
33     Bign() {
34         clear();
35     }
36     Bign(int x) {
```

```
37     *this = x;
38 }
39 friend istream& operator>>(istream& in, Bign& a) {
40     string s;
41     in >> s;
42     a = s;
43     return in;
44 }
45 friend ostream& operator<<(ostream& out, const Bign& a) {
46     if (a.sgn < 0) {
47         out << "-";
48     }
49     for (int i = a.len - 1; i >= 0; i--) {
50         out << a.w[i];
51     }
52     return out;
53 }
54 friend int cmp(const Bign& a, const Bign& b) {
55     if (a.sgn != b.sgn) {
56         return a.sgn > b.sgn ? 1 : -1;
57     }
58     int res = a.sgn;
59     if (a.len != b.len) {
60         return a.len > b.len ? res : -res;
61     }
62     for (int i = a.len - 1; i >= 0; i--) {
63         if (a.w[i] != b.w[i]) {
64             return a.w[i] > b.w[i] ? res : -res;
65         }
66     }
67     return 0;
68 }
69 friend bool eq(const Bign& a, const Bign& b) {
70     return cmp(a, b) == 0;
71 }
72 friend Bign add(const Bign& a, const Bign& b) {
73     Bign c;
74     c.len = max(a.len, b.len);
75     for (int i = 0; i < c.len; i++) {
76         if (i < a.len) {
77             c.w[i] += a.w[i];
78         }
79         if (i < b.len) {
80             c.w[i] += b.w[i];
81         }
82         c.w[i + 1] += c.w[i] / 10;
83         c.w[i] %= 10;
84     }
85     if (c.w[c.len] > 0) {
86         c.len++;
87     }
88     return c;
89 }
```



```

90     friend Bign sub(Bign a, Bign b) {
91         Bign c;
92         if (a < b) {
93             swap(a, b), c.sgn = -1;
94         }
95         c.len = a.len;
96         for (int i = 0; i < c.len; i++) {
97             c.w[i] += a.w[i];
98             if (i < b.len) {
99                 c.w[i] -= b.w[i];
100             }
101             if (c.w[i] < 0) {
102                 c.w[i] += 10, c.w[i + 1]--;
103             }
104         }
105         while (c.w[c.len - 1] == 0) {
106             c.len--;
107         }
108         return c;
109     }
110     friend Bign operator+(const Bign& a, const Bign& b) {
111         return add(a, b);
112     }
113     friend Bign operator-(const Bign& a, const Bign& b) {
114         return sub(a, b);
115     }
116     friend bool operator<(const Bign& a, const Bign& b) {
117         return cmp(a, b) < 0;
118     }
119     friend bool operator>(const Bign& a, const Bign& b) {
120         return cmp(a, b) > 0;
121     }
122     friend bool operator==(const Bign& a, const Bign& b) {
123         return eq(a, b);
124     }
125     int remove() {
126         int cnt = 0;
127         while (cnt < len && w[cnt] == 0) {
128             cnt++;
129         }
130         for (int i = cnt, j = 0; i < len; i++, j++) {
131             w[j] = w[i];
132         }
133         for (int i = 0, j = len - 1; i < cnt; i++, j--) {
134             w[j] = 0;
135         }
136         len -= cnt;
137         return cnt;
138     }
139 };

```


Chapter 2

Number Theory

2.1 筛法

2.1.1 线性筛 (欧拉筛)

$O(n)$. 欧拉筛还可以将可乘函数计算出来, 如下面的代码计算欧拉函数 $\varphi(n)$.

```
1 void Euler(int n) {
2     phi[1] = 1;
3     for (int i = 2; i <= n; i++) {
4         if (!vis[i]) {
5             phi[i] = i - 1;
6             vis[i] = true;
7             pri[++cnt] = i;
8         }
9         for (int j = 1; j <= cnt && i * pri[j] <= n; j++) {
10             vis[i * pri[j]] = true;
11             if (i % pri[j] == 0) {
12                 phi[i * pri[j]] = phi[i] * pri[j];
13                 break;
14             }
15             phi[i * pri[j]] = phi[i] * phi[pri[j]];
16         }
17     }
18 }
```

2.1.2 杜教筛

$O(n^{\frac{2}{3}})$. 对于积性函数 $f(x)$ 的前缀和 $S(n)$, 对于任意一个数论函数 $g(x)$, 均有:

$$g(1)S(n) = \sum_{i=1}^n (f * g)(i) - \sum_{i=2}^n g(i)S(\lfloor \frac{n}{i} \rfloor)$$

可以利用贝尔级数等找到 $g, f * g$ 都易于求和的 g .
快速求莫比乌斯函数前缀和.

```
1 int S_mu(int n) {
2     if (n <= MAX_N) return mu[n];
3     if (hsmu.count(n)) return hsmu[n];
4     int res = 1;
```

```

5   for (int i = 2, j; i <= n; i = j + 1) {
6       j = n / (n / i);
7       res -= (j - i + 1) * S_mu(n / i);
8   }
9   return hsmu[n] = res;
10 }
```

2.1.3 min25 筛

min25 筛主要分为两个步骤，首先自下而上计算一个素数函数前缀和，然后自上而下递归地计算积性函数前缀和，这个过程是类似埃氏筛的。

为了做到这一点，那么首先定义两个函数 $g(n, j)$ 和 $S(n, j)$ ：

$$g(n, j) = \sum_{i=1}^n f(i) [i \in P \text{ 或 } i \text{ 的最小素因子} > p_j]$$

$$S(n, j) = \sum_{i=1}^n f(i) [i \text{ 的最小素因子} > p_j]$$

$f(x)$ 为要求前缀和的积性函数， p_j 表示第 j 个素数， P 为小于等于 n 的素数组成的集合，特别的我们定义 $p_0 = 1$ 。为什么要这么做呢？首先看第一个问题，要把素数函数的前缀和筛出来，如果利用埃氏筛，其时间复杂度为 $O(n \log \log n)$ 。仔细思考其过程，如果筛到一个素数，要把它所有的倍数全部划去，但是很显然如果这个素数大于 $\lfloor \sqrt{n} \rfloor$ 的时候，后面根本没有数要被划去！但是由于要筛出所有素数，这一步骤必须进行；再看现在的问题，如果要筛出前缀和，那么不需要处理出所有素数，也就是说，如果先把所有的数和算出来，然后用素数倍数一个个把不符合的划去，就可以计算出素数函数的前缀和了。而由于大于根号的素数对答案不会产生贡献（不会划去任何数），所以我们只要预处理前面 $\lfloor \sqrt{n} \rfloor$ 的素数就行了。利用素数分布的结果大概可以知道这个过程差不多是 $O(\frac{n}{\log n})$ 的。

接下来处理细节，考虑 $g(n, j)$ 的转移：由刚才的分析， $g(n, j)$ 显然要由 $g(n, j-1)$ 转移过来，前一个状态划去了所有素因子小于等于 p_{j-1} 的，接下来显然要把最小素因子为 p_j 的划去。因为需要划去的数其最小的素数已经是 p_j 了，所以我们可以发现这些数的和就是 $g(\frac{n}{p_j}, j-1)$ ，但是需要注意，这里面不止包含了素因子大于等于 p_j 的数，还有所有素数，因此需要把小于 p_j 的素数加回来。故有：

$$g(n, j) = g(n, j-1) - f(p_j) \times \left(g\left(\frac{n}{p_j}, j-1\right) - g(p_{j-1}, j-1) \right)$$

由于如果有必要转移的话（就是如果 $p_j^2 > n$ 时， $g(n, j) = g(n, j-1)$ ），所以 $g(p_{j-1}, j-1)$ 也就是前 $(j-1)$ 个素数函数前缀和是可以预处理的时候算出来的。所以转移需要用到的状态是 $g(n, j-1), g(\frac{n}{p_j}, j-1)$ ，那么很明显这个可以用滚动数组进行优化，并且注意到所有对答案有贡献的状态 $g(n', j)$ 中， n' 必然是某个 $\lfloor \frac{n}{i} \rfloor$ ，由整除分块的知识知道有用的状态只有 $O(\sqrt{n})$ 个。考虑第二个问题，也是类似的，我们考虑它的转移。因为以我们的定义答案就是 $S(n, 0) + 1$ ，所以这个转移方向和 g 是相反的。考虑 $S(n, j)$ ，可以认为其值由两部分组成，一部分是素数的和，这个由前面已经计算出来的 g 值就可以直接统计出答案，所以问题变成合数部分如何求解。我们考虑该合数的最小素数，那么由 $S(n, j)$ 的定义，这个素数是 p_{j+1}, p_{j+2}, \dots ，我们枚举它的幂次 e ，暴力由 $S(\frac{n}{p_k^e}, k)$ 转移过来，但是需要注意，这时候 $p_k^e (e > 1)$ 同样被删去了，把它加回来就行。故有：

$$S(n, j) = g(n, |P|) - \sum_{i=1}^j f(p_i) + \sum_{k=j+1}^{|P| \text{ 且 } p_k^2 \leq n} \sum_{e=1}^{\infty} \left(f(p_k^e) \times \left(S\left(\frac{n}{p_k^e}, k\right) + (e > 1) \right) \right)$$

这个时间复杂度不会分析，据说是亚线性 $O(n^{1-\varepsilon})$ 。

$O(n^{1-\varepsilon})$ 。

```

1 // luogu P5325 f(p^k) = p^k(p^k-1)
2 #include <bits/stdc++.h>
3 #define int long long
4
5 const int MOD = 1e9 + 7;
6 const int MAX_SQRT = 2e5 + 7;
7 int pri[MAX_SQRT], id1[MAX_SQRT << 1], id2[MAX_SQRT], w[MAX_SQRT], sp2[MAX_SQRT],
    sp1[MAX_SQRT], g2[MAX_SQRT], g1[MAX_SQRT];
8 bool vis[MAX_SQRT];
9 int n, cnt, sqr, tot;
10
11 inline int add(const int &x, const int &y) {
12     return (x + y) >= MOD ? x + y - MOD : x + y;
13 }
14
15 inline int dec(const int &x, const int &y) {
16     return (x - y) < 0 ? x - y + MOD : x - y;
17 }
18
19 void Euler(int n) {
20     pri[0] = 1;
21     for (int i = 2; i <= n; ++i) {
22         if (!vis[i]) {
23             pri[++cnt] = i;
24             sp2[cnt] = add(sp2[cnt - 1], i * i % MOD);
25             sp1[cnt] = add(sp1[cnt - 1], i);
26         }
27         for (int j = 1; j <= cnt && i * pri[j] <= n; ++j) {
28             vis[i * pri[j]] = true;
29             if (i % pri[j] == 0) break;
30         }
31     }
32 }
33
34 int f_pow(int base, int b, int mod = MOD) {
35     int res = 1;
36     while (b) {
37         if (b & 1) res = res * base % mod;
38         base = base * base % mod;
39         b >>= 1;
40     }
41     return res;
42 }
43
44 inline int f(const int &p, const int &e) {
45     int tmp = f_pow(p, e);
46     return tmp * (tmp - 1) % MOD;
47 }
48
49 inline int g(const int &k) { return dec(g2[k], g1[k]); }

```

```

50
51 inline int sp(const int &y) { return dec(sp2[y], sp1[y]); }
52
53 int S(int x, int y) {
54     if (pri[y] >= x) return 0;
55     int k = (x <= sqr) ? id1[x] : id2[n / x], res = dec(g(k), sp(y));
56     for (int i = y + 1; i <= cnt && pri[i] * pri[i] <= x; ++i) {
57         for (int e = 1, prod = pri[i]; prod <= x; ++e, prod *= pri[i]) {
58             int tmp = prod % MOD;
59             res = add(res, tmp * (tmp - 1) % MOD * (S(x / prod, i) + (e != 1)) % MOD);
60             // res = add(res, f(pri[i], e) * (S(n / prod, i) + (e != 1)) % MOD);
61         }
62     }
63     return res < 0 ? res + MOD : res;
64 }
65
66 int min_25(const int &n) {
67     sqr = sqrt(n);
68     Euler(sqr);
69     int inv6 = f_pow(611, MOD - 2);
70     for (int i = 1, j, tmp; i <= n; i = j + 1) {
71         j = n / (n / i);
72         w[++tot] = tmp = n / i;
73         if (tmp >= MOD) tmp %= MOD;
74         g1[tot] = dec(tmp * (tmp + 1) / 2 % MOD, 1);
75         g2[tot] = dec(tmp * (tmp + 1) % MOD * (tmp + tmp + 1) % MOD * inv6 % MOD, 1);
76         if (w[tot] <= sqr) id1[w[tot]] = tot;
77         else id2[n / w[tot]] = tot;
78     }
79     for (int i = 1; i <= cnt; ++i) {
80         for (int j = 1; j <= tot && pri[i] * pri[i] <= w[j]; ++j) {
81             int k = (w[j] / pri[i] <= sqr) ? id1[w[j] / pri[i]] : id2[n / (w[j] / pri[i])];
82             g1[j] = dec(g1[j], pri[i] * dec(g1[k], sp1[i - 1]) % MOD);
83             g2[j] = dec(g2[j], pri[i] * pri[i] % MOD * dec(g2[k], sp2[i - 1]) % MOD);
84         }
85     }
86     return add(S(n, 0), 1);
87 }
88
89 signed main() {
90     scanf("%lld", &n);
91     printf("%lld\n", min_25(n));
92     return 0;
93 }

```

2.2 Euclid 算法

2.2.1 最大公约数算法

$O(\log n)$.

```
1 int gcd(int a, int b) { return b ? gcd(b, a % b) : a; }
```

2.2.2 扩展欧几里得

$O(\log n)$, 求裴蜀定理系数.

$$ax_0 + by_0 = d$$

则不定方程通解为:

$$\begin{cases} x = (x_0 + \frac{b}{d}t) \\ y = (y_0 - \frac{a}{d}t) \end{cases}$$

```
1 int ex_gcd(int a, int b, int &x, int &y) {
2     if (!b) {
3         x = 1, y = 0;
4         return a;
5     }
6     int res = ex_gcd(b, a % b, x, y);
7     int tmp = x; x = y; y = tmp - a / b * y;
8     return res;
9 }
```

2.2.3 类欧几里得

求两个分数中间分子最小的分数.

$O(\log n)$.

```
1 pii find(int a, int b, int c, int d) {
2     if ((a + b - 1) / b <= c / d) return pii((a + b - 1) / b, 1);
3     int t = a / d;
4     pii tmp = find(d, c - d * t, b, a - t * b);
5     return pii(tmp.second + t * tmp.first, tmp.first);
6 }
```

2.3 中国剩余定理 (CRT)

2.3.1 CRT

$n \log \text{MAX}$.

```
1 inline int crt(int a, int m, int M) {
2     return a * inv(M / m, m) % M * (M / m) % M;
3 }
```

2.3.2 EX-CRT

对方程进行两两合并, 由于若干个方程的解为一个特解的关于最小公倍数的剩余类, 所以合并是正确的。对于当前要合并的两个方程:

$$\begin{cases} x \equiv a_1 \pmod{m_1} \\ x \equiv a_2 \pmod{m_2} \end{cases}$$

也即

$$\begin{cases} x = m_1 t_1 + a_1 \\ x = m_2 t_2 + a_2 \end{cases}$$

两式相减即得

$$m_1 t_1 - m_2 t_2 = a_2 - a_1 = c$$

那么很明显, 若 $(m_1, m_2) \nmid c$ 则方程无解。否则也就是要求解同余方程:

$$m_1 t_1 \equiv c \pmod{m_2}$$

扩展欧几里得即可。

$O(n \log \text{MAX})$.

```

1 int ex_crt(int *a, int *m, int n) {
2     int x, y, m1 = 1, ans = 0;
3     for (int i = 1; i <= n; ++i) {
4         int m2 = m[i], c = (a[i] - ans) % m2, _gcd = ex_gcd(m1, m2, x, y);
5         if (c % _gcd) return -1;
6         x = (x * (c / _gcd)) % m2;
7         ans += x * m1;
8         m1 *= m2 / _gcd;
9         ans %= m1;
10    }
11    return ans < 0 ? ans + m1 : ans;
12 }

```

2.4 Lucas 定理

2.4.1 Lucas

定理 2.4.1 (Lucas's Theorem).

$$C_a^b \equiv C_{\lfloor \frac{a}{p} \rfloor}^{\lfloor \frac{b}{p} \rfloor} \cdot C_a^b \pmod{p} \pmod{p}$$

Lucas 定理揭示了一个组合数取模可以将 m, n 分别写成 p 进制数再进行计算。其主要原因是由于

$$p \mid C_p^n (n = 1, 2, \dots, p-1)$$

因而有

$$(1+x)^p \equiv 1+x^p \pmod{p}$$

由带余数除法将 m, n 写成:

$$\begin{cases} m = pq_m + r_m \\ n = pq_n + r_n \end{cases}$$

于是有

$$\begin{aligned}
(1+x)^m &= (1+x)^{pq_m+r_m} \\
&= [(1+x)^p]^{q_m} \cdot (1+x)^{r_m} \\
&\equiv (1+x^p)^{q_m} \cdot (1+x)^{r_m}
\end{aligned}$$

考虑左边 x^n 的系数 C_m^n ；而右边第二项由于 $r_m < p$ ，而第一项全是 p 的若干倍数，故只能是第一项取 q_n ，第二项取 r_n ，于是有：

$$C_m^n = C_{\lfloor \frac{n}{p} \rfloor}^{\lfloor \frac{n}{p} \rfloor} \cdot C_{m \bmod p}^{n \bmod p}$$

Lucas 定理另一形式：令 $m = m_0 + m_1p + \dots + m_dp^d, n = n_0 + n_1p + \dots + n_dp^d$ ，则：

$$C_m^n \equiv C_{m_0}^{n_0} \cdot C_{m_1}^{n_1} \cdot \dots \cdot C_{m_d}^{n_d} \pmod{p}$$

推论 2.4.1.

$$C_n^p \equiv \lfloor \frac{n}{p} \rfloor! \pmod{p}$$

定理 2.4.2 (Fine's Theorem).

$$n = n_0 + n_1p + n_2p^2 + \dots + n_dp^d$$

则 $C_n^k (k = 0, 1, \dots, n)$ 中，有 $\prod_{i=1}^d (1 + n_i)$ 个数不被 p 整除。

如果预处理前 p 个数的阶乘及其逆元，则时间复杂度为 $O(p)$.

```

1 void init(int p) {
2     fac[0] = 1;
3     for (int i = 1; i < p; ++i) fac[i] = fac[i - 1] * i % p;
4     inv[p - 1] = f_pow(fac[p - 1], p - 2, p);
5     for (int i = p - 1; i; --i) inv[i - 1] = i * inv[i] % p;
6 }
7
8 inline int C(int n, int m, int p) {
9     return n >= m ? fac[n] * inv[n - m] % p * inv[m] % p : 0;
10 }
11
12 int lucas(int n, int m, int p) {
13     return m ? C(n % p, m % p, p) * lucas(n / p, m / p, p) % p : 1;
14 }

```

2.4.2 EX-Lucas

扩展 Lucas 定理是将 p 进行分解：

$$p = \prod_{i=1}^k p_i^{\alpha_i}$$

然后计算 $C_n^m \equiv a \pmod{p^{\alpha_i}}$ ，最后中国剩余定理进行合并。考虑

$$\frac{n!}{m!(n-m)!} = \frac{\frac{n!}{p^a}}{\frac{m!}{p^b} \frac{(n-m)!}{p^c}} \cdot p^{a-b-c}$$

所以考虑

$$\frac{n!}{p^\alpha} \% p^\alpha$$

稍加观察可以发现上式

$$\frac{n!}{p^\alpha} \equiv \left\lfloor \frac{n}{p} \right\rfloor! \cdot \left(\prod_{i=1 \text{ 且 } (i,p)=1}^{p^\alpha} i \right)^{\left\lfloor \frac{n}{p^\alpha} \right\rfloor} \cdot \left(\prod_{i=1 \text{ 且 } (i,p)=1}^{(n \% p^\alpha)} i \right) \pmod{p^\alpha}$$

$$O(p \log_p n)$$

```

1  #include <bits/stdc++.h>
2  #define int long long
3
4  using namespace std;
5
6  int n, m, p;
7
8  inline int add(const int &x, const int &y, const int &mod) {
9      return x + y >= mod ? x + y - mod : x + y;
10 }
11 inline int dec(const int &x, const int &y, const int &mod) {
12     return x - y < 0 ? x - y + mod : x - y;
13 }
14
15 int f_pow(int base, int b, int mod) {
16     int res = 1;
17     while (b) {
18         if (b & 1) res = res * base % mod;
19         base = base * base % mod;
20         b >>= 1;
21     }
22     return res;
23 }
24
25 int ex_gcd(int a, int b, int &x, int &y) {
26     if (!b) {
27         x = 1, y = 0;
28         return a;
29     }
30     int res = ex_gcd(b, a % b, x, y);
31     int tmp = x; x = y; y = tmp - a / b * y;
32     return res;
33 }
34
35 inline int inv(int a, int mod) {
36     int x, y;
37     ex_gcd(a, mod, x, y);
38     return (x % mod) < 0 ? x + mod : x;
39 }
40
41 int fac(int n, int p, int pk) {
42     if (!n) return 1;

```

```

43     int res = 1;
44     for (int i = 2; i < pk; ++i) if (i % p)
45         res = res * i % pk;
46     res = f_pow(res, n / pk, pk);
47     for (int i = 2; i <= n % pk; ++i) if (i % p)
48         res = res * i % pk;
49     return res * fac(n / p, p, pk) % pk;
50 }
51
52 int C(int n, int m, int pk, int p) {
53     int k = 0;
54     for (int i = n; i; i /= p) k += (i / p);
55     for (int i = m; i; i /= p) k -= (i / p);
56     for (int i = n - m; i; i /= p) k -= (i / p);
57     return fac(n, p, pk) * inv(fac(m, p, pk), pk) % pk * inv(fac(n - m, p, pk), pk)
58         % pk * f_pow(p, k, pk) % pk;
59 }
60 inline int crt(int a, int m, int M) {
61     return a * inv(M / m, m) % M * (M / m) % M;
62 }
63
64 int ex_lucas(int n, int m, int p) {
65     if (n < m) return 0;
66     int res = 0, t = p;
67     for (int i = 2, pk; i * i <= p; ++i) {
68         if (t % i) continue;
69         pk = 1;
70         while (t % i == 0) t /= i, pk *= i;
71         res = add(res, crt(C(n, m, pk, i), pk, p), p);
72     }
73     if (t > 1) res = add(res, crt(C(n, m, t, t), t, p), p);
74     return res;
75 }
76
77 signed main() {
78     scanf("%lld%lld%lld", &n, &m, &p);
79     printf("%lld\n", ex_lucas(n, m, p));
80     return 0;
81 }

```

2.5 原根

寻找最小原根, $O(n^{0.25} \log n + \sqrt{n})$.

```

1 void getPrime(int n) {
2     cntn = 0;
3     int x = n;
4     for (int i = 2; i * i <= n; i++) {
5         if (x % i == 0) {
6             npri[++cntn] = i;
7             while (x % i == 0) x /= i;

```

```

8     }
9     }
10    if (x > 1) npri[++cntn] = x;
11 }
12
13 bool chk(int g, int n) {
14     if (f_pow(g, phi[n], n) != 1) return false;
15     getPrime(phi[n]);
16     for (int i = 1; i <= cntn; i++) {
17         if (f_pow(g, phi[n] / npri[i], n) == 1) return false;
18     }
19     return true;
20 }
21
22 int findG(int n) {
23     for (int i = 1; i < n; i++) {
24         if (chk(i, n)) return i;
25     }
26     return -1;
27 }

```

寻找所有原根, $O(\varphi(n)\log\varphi(n))$.

```

1 void findG(int g, int n) {
2     int base = g, prod = g;
3     for (int i = 2; i <= phi[n]; i++) {
4         prod = g * prod % n;
5         if (gcd(i, phi[n]) == 1) prt[++num] = prod;
6     }
7 }

```

2.6 离散对数问题 (DLP)

2.6.1 BSGS(Baby-step Giant-step)

BSGS 算法, $O(\sqrt{p})$ 求解满足 $a^x \equiv b \pmod{p}$, $(a, p) = 1$ 的最小自然数解 x .

算法原理: 注意到当 a, p 互素时, 令 $x = A[p] - B$ ($0 \leq B < [p]$), 则原问题等价于

$$a^{A[p]} \equiv ba^B \pmod{p}$$

先 \sqrt{p} 下枚举 B , 将 ba^B 模 p 存到 hash 里面, 再 \sqrt{p} 下枚举 A 即可.

```

1 int bsgs(int a, int p, int b) {
2     int nsqrt = (long long)sqrt(p) + 1, base = f_pow(a, nsqrt, p);
3     unordered_map<int, int> mp;
4     for (int i = 0, prod = b; i < nsqrt; i++, prod = (prod * a) % p) {
5         mp[prod] = i;
6     }
7     for (int i = 1, prod = base; i <= nsqrt; i++, prod = (prod * base) % p) {
8         if (mp.count(prod)) {
9             return i * nsqrt - mp[prod];
10        }
11    }

```

```
12     return -1;
13 }
```

扩展 BSGS 算法, $O(\sqrt{p})$ 求离散对数, 但是不要求 $(a, p) = 1$. 找到最大的 k , 使得 $(a^k, p) > 1$, 如果 b 不能被最大公约数整除, 方程无解, 否则把逆元乘到后面再做 BSGS 即可.

```
1 int ex_bsgs(int a, int p, int b) {
2     int k = 0, d, x, y, down = 1;
3     if (b == 1) return 0;
4     while ((d = ex_gcd(a, p, x, y)) != 1) {
5         if (b % d) return -1;
6         k++, b /= d, p /= d, down = (down * a / d) % p;
7         if (down == b) return k;
8     }
9     ex_gcd(down, p, x, y);
10    b = (b * x % p + p) % p;
11    return (d = bsgs(a, p, b)) < 0 ? -1 : d + k;
12 }
```


Chapter 3

Math

3.1 公式 (Formula)

3.1.1 切/曼距离

切比雪夫距离转化成曼哈顿距离：

$$\begin{cases} x' = \frac{x+y}{2} \\ y' = \frac{x-y}{2} \end{cases}$$

曼哈顿距离转化成切比雪夫距离：

$$\begin{cases} x' = x + y \\ y' = x - y \end{cases}$$

3.1.2 贝尔级数

定义数论函数 f 在模素数 p 意义下的贝尔级数：

$$f_p(x) = \sum_{i=0}^{\infty} f(p^i) x^i$$

则有：

$$(f * g)_p = f_p \times g_p$$

即可以用级数乘法来刻画卷积，这点和拉普拉斯变换有点类似。

常用数论函数的贝尔级数

$$\begin{aligned}\mu_p(x) &= 1 - x \\ \varphi_p(x) &= \frac{1 - x}{1 - px} \\ 1_p(x) &= \frac{1}{1 - x} \\ \varepsilon_p(x) &= 1 \\ ID_p(x) &= \frac{1}{1 - px} \\ ID_{k_p}(x) &= \frac{1}{1 - p^k x} \\ \mu_p^2(x) &= 1 + x \\ (ID \cdot \mu)(x) &= 1 - px \\ \sigma_p(x) &= (1 * ID)_p(x) = (1_p \times ID_p)(x) = \frac{1}{(1 - x)(1 - px)} \\ \sigma_{k_p}(x) &= \frac{1}{(1 - x)(1 - p^k x)} \\ \lambda_p(x) &= \frac{1}{1 + x}\end{aligned}$$

3.2 数值积分

辛普森积分, $O(\text{能过})...$

```
1 double simpson(double a, double b) {
2     double c = (a + b) / 2;
3     return (f(a) + f(b) + 4 * f(c)) * (b - a) / 6;
4 }
5 double asr(double a, double b, double eps, double A) {
6     double c = (a + b) / 2;
7     double L = simpson(a, c), R = simpson(c, b);
8     if (fabs(L + R - A) < 15 * eps)
9         return (L + R + (L + R - A) / 15.0);
10    return asr(a, c, eps / 2, L) + asr(c, b, eps / 2, R);
11 }
12 double asr(double a, double b, double eps) { return asr(a, b, eps, simpson(a, b)); }
```

3.3 康拓展开

对于排列 $b_1 b_2 \dots b_n$, 其排名:

$$X = \sum_{i=1}^n a_i (n - i)! + 1$$

a_i 代表后面有多少个小于当前元素的元素个数。

$O(n \log n)$ 。(树状数组优化)

```
1 int cantor(int *a, int n) {
2     int res = 1;
```



```
3   for (int i = 1; i <= n; ++i) {
4       int cnt = get_sum(a[i] - 1);
5       add(a[i], -1);
6       res = (res + cnt * fac[n - i] % MOD) % MOD;
7   }
8   return res;
9 }
```

逆康托展开: $O(n \log^2 n)$, 树状数组优化版本。(实际上一般 n 都很小, 暴力即可)

```
1 vector<int> decantor(int x, int n) {
2     vector<int> res;
3     for (int i = 1; i <= n; ++i) {
4         int pos = find_pos(x / fac[n - i]);
5         x %= fac[n - 1];
6         res.push_back(pos);
7         add(pos, -1);
8     }
9     return res;
10 }
```


Chapter 4

Polynomial

4.1 快速傅立叶变换 FFT

$O(n \log n)$.

```
1 //luogu P3803 【模板】多项式乘法 (FFT)
2 #include <iostream>
3 #include <cstdio>
4 #include <cstring>
5 #include <complex>
6 #include <algorithm>
7 #include <cmath>
8 #define PI 3.141592653589
9
10 using namespace std;
11
12 typedef complex <double> cd;
13 const int N = 3e6 + 7;
14 int rev[N];
15 cd f[N], g[N];
16 int n, m;
17
18 void fft(cd *a, int n, int dft) {
19     for(int i = 0; i < n; i++) {
20         if(i < rev[i]) swap(a[i], a[rev[i]]);
21     }
22     for(int i = 1; i < n; i <= 1) {
23         cd wn = exp(cd(0, 1.0 * dft * PI / i));
24         for(int j = 0; j < n; j += (i < 1)) {
25             cd wnk = cd(1, 0);
26             for(int k = j; k < j + i; k++) {
27                 cd a1 = a[k], a2 = a[k + i];
28                 a[k] = a1 + wnk * a2;
29                 a[k + i] = a1 - wnk * a2;
30                 wnk *= wn;
31             }
32         }
33     }
34     if(dft == -1) {
35         for(int i = 0; i < n; i++) a[i] /= n;
```

```

36     }
37 }
38 int main() {
39     scanf("%d%d", &n, &m);
40     for(int i = 0; i <= n; i++) {
41         int x; scanf("%d", &x);
42         f[i] = x;
43     }
44     for(int i = 0; i <= m; i++) {
45         int x; scanf("%d", &x);
46         g[i] = x;
47     }
48     int N = 1, p = 0;
49     while(N < (m + n + 1)) N <= 1, p++;
50     for(int i = 0; i < N; i++) rev[i] = ((rev[i >> 1] >> 1) | ((i & 1) << (p - 1)));
51     fft(f, N, 1);
52     fft(g, N, 1);
53     for(int i = 0; i < N; i++) {
54         f[i] *= g[i];
55     }
56     fft(f, N, -1);
57     for(int i = 0; i <= (n + m); i++) {
58         printf("%d ", (int)(f[i].real() + 0.5));
59     }
60     puts("");
61     return 0;
62 }

```

4.2 快速数论变换 NTT

$O(n \log n)$.

```

1  //luogu P3803 【模板】多项式乘法 (FFT)
2  #include <iostream>
3  #include <cstdio>
4  #include <cstring>
5  #include <complex>
6  #include <algorithm>
7  #include <cmath>
8  #define int long long
9  #define G 3
10
11 using namespace std;
12
13 const int N = 3e6 + 7;
14 const int MOD = 998244353;
15 int rev[N];
16 int f[N], g[N];
17 int n, m;
18
19 int f_pow(int base, int b, int mod) {
20     int res = 1;

```

```

21     while(b) {
22         if(b & 1) res = res * base % mod;
23         base = base * base % mod;
24         b >>= 1;
25     }
26     return res;
27 }
28 void ntt(int *a, int n, int dft) {
29     for(int i = 0; i < n; i++) {
30         if(i < rev[i]) swap(a[i], a[rev[i]]);
31     }
32     for(int i = 1; i < n; i <= 1) {
33         int wn = f_pow(G, (MOD - 1) / (i << 1), MOD);
34         if(dft < 0) wn = f_pow(wn, MOD - 2, MOD);
35         for(int j = 0; j < n; j += (i << 1)) {
36             int wnk = 1;
37             for(int k = j; k < j + i; k++) {
38                 int a1 = a[k], a2 = a[k + i];
39                 a[k] = (a1 + wnk * a2 % MOD) % MOD;
40                 a[k + i] = (a1 - wnk * a2 % MOD) % MOD;
41                 wnk = wnk * wn % MOD;
42             }
43         }
44     }
45     if(dft == -1) {
46         int inv = f_pow(n, MOD - 2, MOD);
47         for(int i = 0; i < n; i++) a[i] = a[i] * inv % MOD;
48     }
49 }
50 signed main() {
51     scanf("%lld%lld", &n, &m);
52     for(int i = 0; i <= n; i++) scanf("%lld", f + i);
53     for(int i = 0; i <= m; i++) scanf("%lld", g + i);
54     int N = 1, p = 0;
55     while(N < (m + n + 1)) N <= 1, p++;
56     for(int i = 0; i < N; i++) rev[i] = ((rev[i] >> 1) >> 1) | ((i & 1) << (p - 1));
57     ntt(f, N, 1);
58     ntt(g, N, 1);
59     for(int i = 0; i < N; i++) {
60         f[i] *= g[i];
61     }
62     ntt(f, N, -1);
63     for(int i = 0; i <= (n + m); i++) printf("%lld ", (f[i] + MOD) % MOD);
64     puts("");
65     return 0;
66 }

```


Chapter 5

Geometry

5.1 基础知识

5.1.1 Point 类

```
1  template<class T> int sgn(T x) { return (x > 0) - (x < 0); }
2  struct Point {
3      double x, y;
4      Point(double _x = 0, double _y = 0) : x(_x), y(_y) {}
5      Point operator+(const Point &p) const {
6          return Point(x + p.x, y + p.y);
7      }
8      Point operator-(const Point &p) const {
9          return Point(x - p.x, y - p.y);
10     }
11     Point operator*(double w) const {
12         return Point(x * w, y * w);
13     }
14     Point operator/(double w) const {
15         return Point(x / w, y / w);
16     }
17     bool operator==(const Point &p) const {
18         return (!sgn(x - p.x)) && (!sgn(y - p.y));
19     }
20     bool operator<(const Point &p) const {
21         return (!sgn(x - p.x)) ? x < p.x : y < p.y;
22     }
23     Point unit() const { return *this / sqrt(x * x + y * y); }
24     Point perp() const { return Point(-y, x); }
25     Point normal() const { return perp().unit(); }
26     void print() {
27         printf("%1f %1f\n", x, y);
28     }
29 };
30 double dist2(const Point &p) {
31     return p.x * p.x + p.y * p.y;
32 }
33 double dist(const Point &p) {
34     return sqrt(p.x * p.x + p.y * p.y);
```

```

35 }
36 double dot(const Point &a, const Point &b) {
37     return a.x * b.x + a.y * b.y;
38 }
39 double cross(const Point &a, const Point &b) {
40     return a.x * b.y - b.x * a.y;
41 }
42 Point rotate(const Point &a, double theta) {
43     return Point(a.x * cos(theta) - a.y * sin(theta), a.y * cos(theta) + a.x * sin(
        theta));
44 }
45 double angle(const Point &p) {
46     return atan2(p.y, p.x);
47 }
48 pair<int, Point> line_inter(Point &s1, Point &e1, Point &s2, Point &e2) {
49     double d = cross(e1 - s1, e2 - s2);
50     if (!sgn(d)) return {-(sgn(cross(e1 - s1, e2 - s1)) == 0), Point(0, 0)};
51     double p = cross(e1 - s2, e2 - s2), q = cross(e2 - s2, s1 - s2);
52     return {1, (s1 * p + e1 * q) / d};
53 }

```

5.1.2 定比分点（求两直线交点）

坐标上 A, B 连线上有一点 P 满足：

$$\overrightarrow{AP} = \lambda \overrightarrow{PB}$$

则有：

$$P = \frac{A + \lambda B}{1 + \lambda}$$

5.1.3 Line 类

5.1.4 距离

```

1 // 左正右负
2 double line_dist(Point &s, Point &e, Point &p) {
3     return cross(e - s, p - s) / dist(e - s);
4 }
5
6 double seg_dist(Point &s, Point &e, Point &p) {
7     if (s == e) return dist(p - s);
8     double d = dist2(e - s), t = min(d, max(0.0, dot(p - s, e - s)));
9     return dist((p - s) * d - (e - s) * t) / d;
10 }

```

5.1.5 判断点在线上

```

1 bool on_seg(Point &s, Point &e, Point &p) {
2     return seg_dist(s, e, p) < EPS;
3 }
4
5 bool on_line(Point &s, Point &e, Point &p) {

```



```

6   return line_dist(s, e, p) < EPS;
7 }

```

5.1.6 交点

```

1 vector<Point> seg_inter(Point &s1, Point &e1, Point &s2, Point &e2) {
2     double oa = cross(e2 - s2, s1 - s2), ob = cross(e2 - s2, e1 - s2);
3     double oc = cross(e1 - s1, s2 - s1), od = cross(e1 - s1, e2 - s1);
4     if (sgn(oa) * sgn(ob) < 0 && sgn(oc) * sgn(od) < 0) {
5         return {(s1 * ob - e1 * oa) / (ob - oa)};
6     }
7     set<Point> s;
8     if (on_seg(s2, e2, s1)) s.insert(s1);
9     if (on_seg(s2, e2, e1)) s.insert(e1);
10    if (on_seg(s1, e1, s2)) s.insert(s2);
11    if (on_seg(s1, e1, e2)) s.insert(e2);
12    return {s.begin(), s.end()};
13 }
14
15 //1: 一个交点, -1: 无穷多, 0: 无交点
16 pair<int, Point> line_inter(Point &s1, Point &e1, Point &s2, Point &e2) {
17     double d = cross(e1 - s1, e2 - s2);
18     if (!sgn(d)) return {-(sgn(cross(e1 - s1, e2 - s1)) == 0), Point(0, 0)};
19     double p = cross(e1 - s2, e2 - s2), q = cross(e2 - s2, s1 - s2);
20     return {1, (s1 * p + e1 * q) / d};
21 }
22
23 Point line_inter(Line &l1, Line &l2) {
24     return line_inter(l1[0], l1[1], l2[0], l2[1]).second;
25 }

```

5.1.7 与直线位置

左边返回 1, 右边返回 -1, 在直线上返回 0。

```

1 int side_of(const Point &s, const Point &e, const Point &p) {
2     double a = cross(e - s, p - s), l = dist(e - s) * EPS;
3     return (a > l) - (a < -l);
4 }

```

5.1.8 线性变换

```

1 Point linear_tran(Point &p0, Point &p1, Point &q0, Point &q1, Point &r) {
2     Point dp = p1 - p0, dq = q1 - q0, num(cross(dp, dq), dot(dp, dq));
3     return q0 + Point(cross(r - p0, num), dot(r - p0, num)) / dist2(dp);
4 }

```

5.1.9 对称

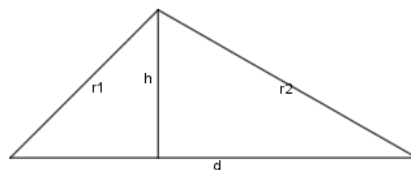
```
1 // 轴对称
2 Point symmetry(Point &s, Point &e, Point &r) {
3     if (s == e) return s * 2 - r;
4     Point p = e - s, q = r - s, num(dot(p, q), cross(p, q));
5     return s + Point(dot(num, p), cross(num, p)) / dist2(p);
6 }
7 // 中心对称
8 Point symmetry(Point &s, Point &r) {
9     return symmetry(s, s, r);
10 }
```

5.2 圆 (Circle)

5.2.1 Circle 类

```
1 struct Circle {
2     Point o; double r;
3     Circle(Point _o = {0, 0}, double _r = 0) {
4         o = _o, r = _r;
5     }
6     int position(const Point &p) const {
7         return sgn(r - dist(p - o));
8     }
9 };
```

5.2.2 两圆交点



如图有：

$$\begin{aligned}\sqrt{r_1^2 - h^2} + \sqrt{r_2^2 - h^2} &= d \\ \Rightarrow r_1^2 - r_2^2 &= d(x_1 - x_2) \\ \Rightarrow x_1 + x_2 &= d \\ \Rightarrow x_1 &= \frac{r_1^2 - r_2^2 + d^2}{2d}\end{aligned}$$

```

1 pair<bool, pair<Point, Point>> circle_inter(Circle &a, Circle &b) {
2     if (a.o == b.o) return {false, {Point(0, 0), Point(0, 0)}};
3     Point d = b.o - a.o;
4     double d2 = dist2(d), sum = a.r + b.r, dif = a.r - b.r,
5         p = (d2 + a.r * a.r - b.r * b.r) / (d2 * 2), h2 = a.r * a.r - p * p * d2;
6     if (sum * sum < d2 || dif * dif > d2) return {false, {Point(0, 0), Point(0, 0)}};
7     Point mid = a + d * p, per = d.perp() * sqrt(max(0.0, h2) / d2);
8     return {true, {mid + per, mid - per}};
9 }

```

5.2.3 两圆公切线

可以返回 0,1,2 三条切线，0 表示没有切线，1 表示两圆相切，2 会返回两条外公切线，并且当 $b.r < 0$ 时返回内公切线。公切线用切点表示。

```

1 vector<pair<Point, Point>> circle_tan(Circle &a, Circle &b) {
2     Point d = b.o - a.o;
3     double dr = a.r - b.r, d2 = dist2(d), h2 = d2 - dr * dr;
4     if (d2 < EPS || h2 < 0) return {};
5     vector<pair<Point, Point>> res;
6     for (double sign : {-1, 1}) {
7         Point v = (d * dr + d.perp() * sqrt(h2) * sign) / d2;
8         res.push_back({a.o + v * a.r, b.o + v * b.r});
9     }
10    if (h2 < EPS) res.pop_back();
11    return res;
12 }

```

5.2.4 三角形外接圆

张角定理可证。

```

1 Circle circumcircle(Point &A, Point &B, Point &C) {
2     Point b = C - A, c = B - A;
3     Point out = (b * dist2(c) - c * dist2(b)).perp() / cross(b, c) / 2;
4     return Circle(A + out, dist(out));
5 }

```

5.2.5 最小圆覆盖 (Minimum Enclosing Circle)

期望复杂度: $O(n)$ 。

```

1 Circle mec(vector<Point> ps) {
2     shuffle(ps.begin(), ps.end(), mt19937(time(0)));
3     Circle C(ps[0], 0);
4     for (int i = 0; i < ps.size(); ++i) if (C.position(ps[i]) == -1) {
5         C = {ps[i], 0};
6         for (int j = 0; j < i; ++j) if (C.position(ps[j]) == -1) {
7             C = {(ps[i] + ps[j]) / 2, dist(ps[j] - ps[i]) / 2};
8             for (int k = 0; k < j; ++k) if (C.position(ps[k]) == -1) {
9                 C = circumcircle(ps[i], ps[j], ps[k]);
10            }
11        }
12    }
13    return C;
14 }

```

5.3 多边形 (Polygon)

5.3.1 面积 (Area)

$O(n)$ 。

```

1 // 逆正顺负
2 double get_area(vector<Point> &p) {
3     double res = 0;
4     int n = p.size();
5     if (n < 3) return res;
6     for (int i = 1; i < n - 1; ++i) {
7         res += cross(p[i] - p[0], p[i + 1] - p[0]);
8     }
9     return res / 2;
10 }
11
12 double get_area(vector<Line> &L) {
13     vector<Point> p;
14     for (int i = 0; i < L.size(); ++i) {
15         p.push_back(line_inter(L[i], L[(i + 1) % L.size()]));
16     }
17     return get_area(p);
18 }

```

5.3.2 凸包 (Convex Hull)

Andrew 算法 ($O(n \log n)$)。

```

1 vector<Point> convex_hull(vector<Point> p) {
2     sort(p.begin(), p.end());
3     vector<Point> h(p.size() + 1);
4     h[0] = p[0];
5     int top = 1;
6     for (int i = 1; i < p.size(); ++i) {
7         while (top > 1 && cross(h[top - 1] - h[top - 2], p[i] - h[top - 1]) <= 0) --
            top;

```

```

8     h[top++] = p[i];
9 }
10 int down = top;
11 for (int i = p.size() - 2; i; --i) {
12     while (top > down && cross(h[top - 1] - h[top - 2], p[i] - h[top - 1]) <= 0)
13         --top;
14     h[top++] = p[i];
15 }
16 return {h.begin(), h.begin() + top};
17 }

```

5.3.3 旋转卡壳

$O(n \log n)$ 。

```

1 pair<Point, Point> hull_diameter(vector<Point> h) {
2     int n = h.size(), j = n < 2 ? 0 : 1;
3     pair<double, pair<Point, Point>> res({0, {h[0], h[0]}});
4     for (int i = 0; i < j; ++i) {
5         for (; j = (j + 1) % n; {
6             res = max(res, {dist2(h[i] - h[j]), {h[i], h[j]}});
7             if (cross(h[(j + 1) % n] - h[j], h[i + 1] - h[i]) >= 0) break;
8         }
9     }
10    return res.second;
11 }

```

5.3.4 半平面交

$O(n \log n)$ 。

```

1 // 求半平面交，半平面是逆时针方向，输出按照逆时针
2 vector<Line> half_plane_inter(vector<Line> &L){
3     sort(L.begin(), L.end());
4     deque<Line> q;
5     vector<Line> res;
6     for (int i = 0; i < L.size(); ++i){
7         if (i && same_dir(L[i], L[i - 1])) continue;
8         while (q.size() > 1 && (!check_pos(q[q.size() - 2], q[q.size() - 1], L[i])))
9             q.pop_back();
10        while (q.size() > 1 && (!check_pos(q[1], q[0], L[i]))) q.pop_front();
11        q.push_back(L[i]);
12    }
13    while (q.size() > 2 && (!check_pos(q[q.size() - 2], q[q.size() - 1], q[0]))) q.
14        pop_back();
15    while (q.size() > 2 && (!check_pos(q[1], q[0], q[q.size() - 1]))) q.pop_front();
16    for (int i = 0; i < q.size(); ++i) res.push_back(q[i]);
17    return res;
18 }

```


Chapter 6

Linear Algebra

6.1 异或线性基

$O(\log n)$ 。

```
1 bool insert(int x) {
2     for (int i = 63; i >= 0; --i) {
3         if (x & (1ll << i)) {
4             if (!b[i]) {
5                 b[i] = x;
6                 return true;
7             }
8             x ^= b[i];
9         }
10    }
11    return false;
12 }
```

6.2 矩阵 (Matrix)

矩阵类：

```
1 const int MAX_ML = 7;
2 struct Matrix{
3     int m[MAX_ML][MAX_ML];
4     int n;
5     Matrix(int _n = 0) : n(_n) {
6         for (int i = 1; i <= n; ++i)
7             for (int j = 1; j <= n; ++j)
8                 m[i][j] = 0;
9     }
10    Matrix I() {
11        Matrix res(n);
12        for (int i = 1; i <= n; ++i) res[i][i] = 1;
13        return res;
14    }
15    int* const operator [] (const int k) {
16        return m[k];
17    }
18 }
```

```

18 Matrix operator * (const Matrix &M) {
19     Matrix res(n);
20     for (int i = 1; i <= n; ++i)
21         for (int j = 1; j <= n; ++j)
22             for (int k = 1; k <= n; ++k)
23                 res[i][j] = (res[i][j] + m[i][k] * M.m[k][j]);
24     return res;
25 }
26 Matrix& operator = (const Matrix &M) {
27     for (int i = 1; i <= n; ++i)
28         for (int j = 1; j <= n; ++j) m[i][j] = M.m[i][j];
29     return *this;
30 }
31 Matrix f_pow(Matrix base, int b) {
32     Matrix res(n); res = I();
33     while (b) {
34         if (b & 1) res = res * base;
35         base = base * base;
36         b >>= 1;
37     }
38     return res;
39 }
40 };

```

6.3 高斯约当消元

$O(n^3)$.

```

1 bool Gauss() {
2     for (int i = 1; i <= n; ++i) {
3         int mpos = i;
4         for (int j = i + 1; j <= n; ++j) {
5             if (fabs(a[j][i]) > fabs(a[mpos][i])) mpos = j;
6         }
7         if (fabs(a[mpos][i]) < EPS) return false;
8         for (int j = i; j <= n + 1; ++j) swap(a[i][j], a[mpos][j]);
9         for (int j = 1; j <= n; ++j) {
10             if (j == i) continue;
11             double tmp = a[j][i] / a[i][i];
12             for (int k = i + 1; k <= n + 1; ++k) {
13                 a[j][k] -= tmp * a[i][k];
14             }
15         }
16     }
17     return true;
18 }

```


Chapter 7

Graph

7.1 树

7.1.1 直径

树形 DP 做法，用 $dp[u]$ 表示当前结点向下最大距离，用最大及次大更新树的直径即可， $O(n)$.

```
1 void dfs(int u, int fa) {
2     for (int i = head[u]; i; i = e[i].next) {
3         int v = e[i].to;
4         if (v == fa) continue;
5         dfs(v, u);
6         dia = max(dia, dp[u] + dp[v] + 1);
7         dp[u] = max(dp[u], dp[v] + 1);
8     }
9 }
```

7.1.2 重心

定义

子树大小最大值最小的点.

性质

1. 重心子树的大小不会超过所有结点数目的一半.

Proof. 反证即可. □

2. 树中所有点到某一个点的距离和中，到重心的距离和最小.

Proof. 考虑树形 DP 的转移，记一个结点 u 的答案为 $dis[u]$ ，那么其儿子 v 可以由他转移

$$dis[v] = dis[u] + (n - 2 * siz[v])$$

也就是说，如果当前树的大小小于全部结点数 n 的一半，那么其父亲的答案肯定更小，以重心为根结点建立一颗树，则由上一条性质知此结论成立. □

3. 把两棵树连接，其重心在原重心路径上.

Proof. 若不再该路径上, 由上一条结论证明中的方法, 可以不断向上更新答案, 矛盾. \square

4. 在树上添加或删除一个叶子结点, 重心最多移动一条边.

5. 重心之间有边相连.

Proof. 先确定一个重心, 很显然, 此重心最大的子树只有一个并且另一个重心在该子树中. 如果其他子树大小都小于 $siz[son] - 1$, 那么显然最大子树根结点才是重心, 故可以发现其他子树大小最大为 $siz[son] - 1$, 故另一个根结点即为最大子树的根. 故相连. \square

6. 推论: 树的重心最多有两个.

DFS 算法

选一个根 DFS, 每次通过向上和向下更新当前答案即可.

7.1.3 树上差分

树上差分数组定义为

$$\text{diff}[u] = w[u] - \sum_{v's \text{ father is } u} w[v]$$

点差分: 将结点 u 和 v 之间的所有点权值 $+x$, 则操作为:

```
1 diff[u] += x;
2 diff[v] += x;
3 diff[lca] -= x;
4 diff[fa[lca]] -= x;
```

则每个结点的答案为其子树所有结点权值之和。

边差分: 将结点 u 和 v 之间的所有边的权值 $+x$, 则操作为:

```
1 diff[u] += x;
2 diff[v] += x;
3 diff[lca] -= 2 * x;
```

则当前子树的权值和是当前结点到其父亲结点的边的权值。

7.1.4 重链剖分

$O(n)$.

```
1 void dfs1(int u) {
2     siz[u] = 1;
3     for (int i = head[u]; i; i = e[i].next) {
4         int v = e[i].to;
5         if (v == fa[u]) continue;
6         fa[v] = u;
7         dep[v] = dep[u] + 1;
8         dfs1(v);
9         siz[u] += siz[v];
10        if (siz[v] > siz[son[u]]) son[u] = v;
11    }
12 }
13
14 void dfs2(int u, int tp) {
15     top[u] = tp;
16     dfn[u] = ++order;
```

```

17   rk[order] = u;
18   if (son[u]) dfs2(son[u], tp);
19   for (int i = head[u]; i; i = e[i].next) {
20       int v = e[i].to;
21       if (v == fa[u] || v == son[u]) continue;
22       dfs2(v, v);
23   }
24 }

```

树链剖分求 LCA, $O(\log n)$ 。

```

1 int lca(int u, int v) {
2     while (top[u] != top[v]) {
3         if (dep[top[u]] > dep[top[v]]) swap(u, v);
4         v = fa[top[v]];
5     }
6     return dep[u] < dep[v] ? u : v;
7 }

```

7.1.5 树上启发式合并

某类计算每棵子树的答案的问题时，需要先由子树信息得到当前树的答案，但是需要清空去计算兄弟子树的答案导致时间复杂度变为 $O(n^2)$ 。注意到最后一棵子树不会再影响后面的兄弟子树，故其不需要清空，所以贪心地想必然选子树最大的作为最后一棵，也就是重儿子。算法设计过程如下：

1. 递归计算轻儿子的答案，并将记录的信息清空；
2. 计算重儿子的答案，不清空；
3. 合并其他轻儿子的答案。

复杂度分析：可以发现，每个结点被计算的次数即为从根走到当前结点的轻边数量加 1，基于重连剖分的性质，每个点被计算的次数为 $O(\log n)$ ，故总的时间复杂度为 $O(n \log n)$ 。

```

1 //CF600E. Lomsat gelral
2 #include <iostream>
3 #include <cstdio>
4 #include <cstring>
5 #define int long long
6
7 using namespace std;
8
9 struct Edge {
10     int to, next;
11 };
12
13 const int MAX_N = 1e5 + 7;
14 Edge e[MAX_N << 1];
15 int head[MAX_N], siz[MAX_N], cnt[MAX_N], son[MAX_N], ans[MAX_N], col[MAX_N];
16 int n, cnt_e, sum, mx;
17
18 void add(int u, int v) {
19     e[++cnt_e].to = v;

```

```
20     e[cnt_e].next = head[u];
21     head[u] = cnt_e;
22 }
23
24 void dfs1(int u, int fa) {
25     siz[u] = 1;
26     for (int i = head[u]; i; i = e[i].next) {
27         int v = e[i].to;
28         if (v == fa) continue;
29         dfs1(v, u);
30         siz[u] += siz[v];
31         if (siz[v] > siz[son[u]]) son[u] = v;
32     }
33 }
34
35 void del(int u, int fa) {
36     --cnt[col[u]];
37     for (int i = head[u]; i; i = e[i].next) {
38         int v = e[i].to;
39         if (v == fa) continue;
40         del(v, u);
41     }
42 }
43
44 void calc(int u) {
45     ++cnt[col[u]];
46     if (mx < cnt[col[u]]) {
47         sum = col[u];
48         mx = cnt[col[u]];
49     } else if (mx == cnt[col[u]]) {
50         sum += col[u];
51     }
52 }
53
54 void dfs2(int u, int fa) {
55     calc(u);
56     for (int i = head[u]; i; i = e[i].next) {
57         int v = e[i].to;
58         if (v == fa) continue;
59         dfs2(v, u);
60     }
61 }
62
63 void dsu(int u, int fa) {
64     for (int i = head[u]; i; i = e[i].next) {
65         int v = e[i].to;
66         if (v == fa || v == son[u]) continue;
67         dsu(v, u);
68         del(v, u);
69         sum = mx = 0;
70     }
71     if (son[u]) dsu(son[u], u);
72     calc(u);
```

```

73     for (int i = head[u]; i; i = e[i].next) {
74         int v = e[i].to;
75         if (v == fa || v == son[u]) continue;
76         dfs2(v, u);
77     }
78     ans[u] = sum;
79 }
80
81 signed main() {
82     scanf("%lld", &n);
83     for (int i = 1; i <= n; ++i) scanf("%lld", col + i);
84     for (int i = 1; i < n; ++i) {
85         int u, v;
86         scanf("%lld%lld", &u, &v);
87         add(u, v); add(v, u);
88     }
89     dfs1(1, 0);
90     dsu(1, 0);
91     for (int i = 1; i <= n; ++i) printf("%lld ", ans[i]);
92     puts("");
93     return 0;
94 }

```

7.1.6 虚树

$O(\sum k \log k)$.

```

1  bool cmp(const int &x, const int &y) {
2      return dfn[x] < dfn[y];
3  }
4
5  void build() {
6      sort(h + 1, h + 1 + k, cmp);
7      st.push(1), vhead[1] = 0, cntve = 0;
8      for (int i = 1, _lca; i <= k; i++) {
9          if (h[i] == 1) continue;
10         _lca = lca(h[i], st.top());
11         if (_lca != st.top()) {
12             int tp = st.top(); st.pop();
13             while (dfn[_lca] < dfn[st.top()]) vadd(st.top(), tp), tp = st.top(), st.
                pop();
14             if (dfn[_lca] > dfn[st.top()]) {
15                 vhead[_lca] = 0, vadd(_lca, tp), st.push(_lca);
16             } else {
17                 vadd(_lca, tp);
18             }
19         }
20         vhead[h[i]] = 0, st.push(h[i]);
21     }
22     int tp = st.top(); st.pop();
23     while (!st.empty()) vadd(st.top(), tp), tp = st.top(), st.pop();
24 }

```

7.1.7 点分治

$O(n \log n)$.

```

1 //luogu P3806 【模板】点分治1
2 #include <iostream>
3 #include <cstdio>
4 #include <cstring>
5 #include <queue>
6 #define INF 1e7
7
8 using namespace std;
9
10 struct Edge {
11     int to, next, w;
12     Edge() {}
13 };
14
15 const int MAX_N = 1e4 + 7;
16 const int MAX_K = 1e7 + 7;
17 const int MAX_M = 1e2 + 7;
18 Edge e[MAX_N << 1];
19 int head[MAX_N], siz[MAX_N], dp[MAX_N], dis[MAX_N], qu[MAX_N], k[MAX_M];
20 bool mp[MAX_K], vis[MAX_N], ans[MAX_M];
21 int n, m, rt, cnt, num;
22 queue<int> q;
23
24 void add(int u, int v, int w) {
25     e[++cnt].to = v;
26     e[cnt].next = head[u];
27     head[u] = cnt;
28     e[cnt].w = w;
29 }
30
31 void get_rt(int u, int fa, int size_all) {
32     siz[u] = 1;
33     dp[u] = 0;
34     for (int i = head[u]; i; i = e[i].next) {
35         int v = e[i].to;
36         if (v == fa || vis[v]) continue;
37         get_rt(v, u, size_all);
38         dp[u] = max(dp[u], siz[v]);
39         siz[u] += siz[v];
40     }
41     dp[u] = max(dp[u], size_all - siz[u]);
42     if (dp[u] < dp[rt]) rt = u;
43 }
44
45 void get_dis(int u, int fa) {
46     qu[++num] = dis[u];
47     for (int i = head[u]; i; i = e[i].next) {
48         int v = e[i].to;
49         if (v == fa || vis[v]) continue;
50         dis[v] = dis[u] + e[i].w;

```

```

51     get_dis(v, u);
52 }
53 }
54
55 void solve(int u) {
56     vis[u] = mp[0] = true;
57     q.push(0);
58     for (int i = head[u]; i; i = e[i].next) {
59         int v = e[i].to;
60         if (vis[v]) continue;
61         num = 0; dis[v] = e[i].w;
62         get_dis(v, u);
63         for (int t = 1; t <= num; ++t) {
64             for (int j = 1; j <= m; ++j) {
65                 if (qu[t] <= k[j]) ans[j] |= mp[k[j] - qu[t]];
66             }
67         }
68         for (int t = 1; t <= num; ++t) if (qu[t] <= INF) q.push(qu[t]), mp[qu[t]] = true;
69     }
70     while (!q.empty()) mp[q.front()] = false, q.pop();
71     for (int i = head[u]; i; i = e[i].next) {
72         int v = e[i].to;
73         if (vis[v]) continue;
74         rt = 0;
75         get_rt(v, u, siz[v]);
76         solve(rt);
77     }
78 }
79
80 int main() {
81     scanf("%d%d", &n, &m);
82     for (int i = 1; i < n; ++i) {
83         int u, v, w;
84         scanf("%d%d%d", &u, &v, &w);
85         add(u, v, w); add(v, u, w);
86     }
87     for (int i = 1; i <= m; ++i) scanf("%d", k + i);
88     rt = 0; dp[0] = INF;
89     get_rt(1, 0, n);
90     solve(rt);
91     for (int i = 1; i <= m; ++i) {
92         if (ans[i]) {
93             puts("AYE");
94         } else {
95             puts("NAY");
96         }
97     }
98     return 0;
99 }

```

luogu P4178 Tree: 求树上距离小于等于 k 的点对数, 如果仿照上一种方法, 可以直接利用权值线段树进行区间查询单点修改, 时间复杂度 $O(n \log n \log k)$; 也可以基于桶排序的双指针法 (需

要容斥), 时间复杂度 $O(\max\{n, k\} \log n)$ 。

```

1 //luogu P4178 Tree
2 // 计算距离小于等于 k 的点对, 利用双指针法
3 #include <iostream>
4 #include <cstdio>
5 #include <cstring>
6 #include <queue>
7 #include <complex>
8 #include <vector>
9 #define INF 2e4
10
11 using namespace std;
12
13 struct Edge {
14     int to, next, w;
15     Edge() {}
16 };
17
18 const int MAX_N = 4e4 + 7;
19 const int MAX_K = 2e4 + 7;
20 Edge e[MAX_N << 1];
21 int head[MAX_N], siz[MAX_N], dp[MAX_N], dis[MAX_N], qu[MAX_N], box[MAX_K];
22 bool vis[MAX_N];
23 int n, m, rt, cnt, num, k, ans;
24 queue<int> q;
25
26 void add(int u, int v, int w) {
27     e[++cnt].to = v;
28     e[cnt].next = head[u];
29     head[u] = cnt;
30     e[cnt].w = w;
31 }
32
33 void get_rt(int u, int fa, int size_all) {
34     siz[u] = 1;
35     dp[u] = 0;
36     for (int i = head[u]; i; i = e[i].next) {
37         int v = e[i].to;
38         if (v == fa || vis[v]) continue;
39         get_rt(v, u, size_all);
40         dp[u] = max(dp[u], siz[v]);
41         siz[u] += siz[v];
42     }
43     dp[u] = max(dp[u], size_all - siz[u]);
44     if (dp[u] < dp[rt]) rt = u;
45 }
46
47 void get_dis(int u, int fa) {
48     ++num;
49     ++box[dis[u]];
50     for (int i = head[u]; i; i = e[i].next) {
51         int v = e[i].to;

```



```

52     if (v == fa || vis[v]) continue;
53     dis[v] = dis[u] + e[i].w;
54     get_dis(v, u);
55 }
56 }
57
58 int calc(int u, int w) {
59     dis[u] = w; num = 0; get_dis(u, 0);
60     for (int i = 0, j = 1; i <= MAX_K && j <= num; ++i) {
61         while (box[i]) qu[j++] = i, --box[i];
62     }
63     int l = 1, r = num, res = 0;
64     while (l < r) {
65         qu[l] + qu[r] <= k ? res += r - l, ++l : --r;
66     }
67     return res;
68 }
69
70 void solve(int u) {
71     vis[u] = true; ans += calc(u, 0);
72     for (int i = head[u]; i; i = e[i].next) {
73         int v = e[i].to;
74         if (vis[v]) continue;
75         dis[v] = e[i].w;
76         ans -= calc(v, dis[v]);
77     }
78     for (int i = head[u]; i; i = e[i].next) {
79         int v = e[i].to;
80         if (vis[v]) continue;
81         rt = 0;
82         get_rt(v, u, siz[v]);
83         solve(rt);
84     }
85 }
86
87 signed main() {
88     scanf("%d", &n);
89     for (int i = 1; i < n; ++i) {
90         int u, v, w;
91         scanf("%d%d%d", &u, &v, &w);
92         add(u, v, w); add(v, u, w);
93     }
94     scanf("%d", &k);
95     rt = 0; dp[0] = INF + INF;
96     get_rt(1, 0, n);
97     solve(rt);
98     printf("%d\n", ans);
99     return 0;
100 }

```

7.2 差分约束

$O(|V||E|)$.

```

1  bool bellman_ford(int s) {
2      memset(dis, 0x3f, sizeof dis);
3      dis[s] = 0;
4      int cnt = 0;
5      while(cnt <= n) {
6          bool upd = false;
7          for(int i = 1; i <= cntE; i++) {
8              int x = e[i].from, y = e[i].to, w = e[i].w;
9              if(dis[y] > dis[x] + w) {
10                 dis[y] = dis[x] + w;
11                 upd = true;
12             }
13         }
14         if(!upd) return false;
15         cnt++;
16     }
17     return true;
18 }
19 int main() {
20     scanf("%d%d", &n, &m);
21     for(int i = 1; i <= m; i++) {
22         int x, y, w;
23         scanf("%d%d%d", &x, &y, &w);
24         add(y, x, w);
25     }
26     for(int i = 1; i <= n; i++) {
27         add(0, i, 0);
28     }
29     if(bellman_ford(0)) puts("NO");
30     else {
31         for(int i = 1; i <= n; i++) printf("%d ", dis[i]);
32         puts("");
33     }
34     return 0;
35 }

```

7.3 强连通分量 SCC

7.3.1 tarjan 算法

$O(n)$.

```

1  void tarjan(int u) {
2      dfn[u] = low[u] = ++order;
3      st.push(u);
4      in_st[u] = true;
5      for (int i = head[u]; i; i = e[i].next) {
6          int v = e[i].to;
7          if (!dfn[v]) {

```

```

8         tarjan(v);
9         low[u] = min(low[u], low[v]);
10    } else if (in_st[v]) {
11        low[u] = min(low[u], dfn[v]);
12    }
13    }
14    if (dfn[u] == low[u]) {
15        int tmp;
16        num++;
17        do {
18            tmp = st.top();
19            st.pop();
20            scc[tmp] = num;
21            in_st[tmp] = false;
22        } while(tmp != u);
23    }
24 }

```

7.4 双连通分量 BCC

7.4.1 点双

$O(n)$.

```

1 void tarjan(int u, int fa) {
2     dfn[u] = low[u] = ++order;
3     for (int i = head[u]; i; i = e[i].next) {
4         int v = e[i].to;
5         if (v == fa) continue;
6         if (!dfn[v]) {
7             int id = e[i].index;
8             st.push(id);
9             tarjan(v, u);
10            low[u] = min(low[u], low[v]);
11            if (low[v] >= dfn[u]) {
12                num++;
13                mn[num] = INF;
14                int tmp;
15                do {
16                    tmp = st.top();
17                    belong[tmp] = num;
18                    st.pop();
19                    mn[num] = min(mn[num], tmp);
20                } while(tmp != id);
21            }
22        } else if (dfn[v] < dfn[u]) {
23            st.push(e[i].index);
24            low[u] = min(low[u], dfn[v]);
25        }
26    }
27 }

```

7.4.2 割点

$O(n)$.

```

1 void tarjan(int u, int fa) {
2     dfn[u] = low[u] = ++order;
3     int child = 0;
4     for (int i = head[u]; i; i = e[i].next) {
5         int v = e[i].to;
6         if (v == fa) continue;
7         if (!dfn[v]) {
8             child++;
9             tarjan(v, u);
10            low[u] = min(low[u], low[v]);
11            if (low[v] >= dfn[u] && fa) {
12                if (!point[u]) point[u] = true, ans++;
13            } else if (!fa && child > 1) {
14                if (!point[u]) point[u] = true, ans++;
15            }
16        } else if (dfn[v] < dfn[u]) {
17            low[u] = min(low[u], dfn[v]);
18        }
19    }
20 }

```

7.4.3 边双连通分量/桥

$O(n)$.

```

1 void tarjan(int u, int fa) {
2     dfn[u] = low[u] = ++order;
3     bool flag = true;
4     st.push(u);
5     for (int i = head[u]; i; i = e[i].next) {
6         int v = e[i].to;
7         if (v == fa && flag) {
8             flag = false;
9             continue;
10        }
11        if (!dfn[v]) {
12            tarjan(v, u);
13            low[u] = min(low[u], low[v]);
14            if (low[v] > dfn[u]) {
15                bridge[i] = 1;
16                if (i & 1) bridge[i + 1] = 1;
17                else bridge[i - 1] = 1;
18            }
19        } else if (dfn[v] < dfn[u]) {
20            low[u] = min(low[u], dfn[v]);
21        }
22    }
23    if (dfn[u] == low[u]) {
24        num++;
25        int tmp;

```

```

26     do{
27         tmp = st.top();
28         belong[tmp] = num;
29         st.pop();
30     } while(tmp != u);
31 }
32 }

```

7.5 2-SAT

$O(n)$.

```

1  //HDU 3062 Party
2  #include <iostream>
3  #include <cstdio>
4  #include <cstring>
5  #include <stack>
6
7  using namespace std;
8
9  struct Edge {
10     int to, next;
11 };
12
13 const int MAX_N = 2e3 + 7;
14 Edge e[MAX_N * MAX_N];
15 int head[MAX_N], dfn[MAX_N], low[MAX_N], scc[MAX_N];
16 bool vis[MAX_N];
17 int n, m, cnt, num, order;
18 stack <int> st;
19
20 void add(int x, int y) {
21     e[++cnt].to = y; e[cnt].next = head[x]; head[x] = cnt;
22 }
23
24 void tarjan(int u) {
25     dfn[u] = low[u] = ++order;
26     st.push(u); vis[u] = true;
27     for (int i = head[u]; i; i = e[i].next) {
28         int v = e[i].to;
29         if (!dfn[v]) {
30             tarjan(v);
31             low[u] = min(low[u], low[v]);
32         } else if (vis[v]) {
33             low[u] = min(low[u], dfn[v]);
34         }
35     }
36     if (dfn[u] == low[u]) {
37         num++;
38         int tmp;
39         do {
40             tmp = st.top();

```

```

41         st.pop();
42         scc[tmp] = num;
43         vis[tmp] = false;
44     } while(tmp != u);
45 }
46 }
47
48 bool twoSat() {
49     for (int i = 0; i < (n + n); i++) {
50         if (!dfn[i]) tarjan(i);
51     }
52     for (int i = 0; i < n; i++) {
53         if (scc[i] == scc[i + n]) return false;
54     }
55     return true;
56 }
57
58 int main() {
59     while (scanf("%d%d", &n, &m) != EOF) {
60         cnt = order = num = 0;
61         for (int i = 0; i < (n + n); i++) {
62             dfn[i] = low[i] = head[i] = 0;
63         }
64         for (int i = 1; i <= m; i++) {
65             int a1, a2, c1, c2;
66             scanf("%d%d%d%d", &a1, &a2, &c1, &c2);
67             add(a1 + c1 * n, a2 + (c2 ^ 1) * n);
68             add(a2 + c2 * n, a1 + (c1 ^ 1) * n);
69         }
70         if (twoSat()) puts("YES");
71         else puts("NO");
72     }
73     return 0;
74 }

```

7.6 斯坦纳树

$$O(n \times 3^k + m \log m \times 2^k)$$

```

1  //luogu P6192 【模板】最小斯坦纳树
2  #include <iostream>
3  #include <cstdio>
4  #include <cstring>
5  #include <queue>
6  #define INF 0x3f3f3f3f
7
8  using namespace std;
9
10 struct Edge {
11     int to, next, w;
12     Edge() {}
13 };

```

```

14 struct Node {
15     int v, w;
16     bool operator < (const Node & x) const {
17         return x.w < w;
18     }
19     Node (int _v = 0, int _w = 0) : w(_w), v(_v) {}
20 }p;
21
22 const int MAX_N = 1e2 + 7;
23 const int MAX_S = 1 << 10;
24 int dp[MAX_N][MAX_S + 7], head[MAX_N];
25 bool vis[MAX_N];
26 Edge e[10 * MAX_N];
27 priority_queue <Node> q;
28 int n, m, k, cnt, key;
29
30 void add(int x, int y, int w) {
31     e[++cnt].to = y; e[cnt].next = head[x]; head[x] = cnt; e[cnt].w = w;
32 }
33
34 void dijkstra(int s) {
35     for (int i = 1; i <= n; i++) vis[i] = false;
36     while (!q.empty()) {
37         p = q.top(); q.pop();
38         if (vis[p.v]) continue;
39         vis[p.v] = true;
40         for (int i = head[p.v]; i; i = e[i].next) {
41             int v = e[i].to;
42             if (dp[v][s] > dp[p.v][s] + e[i].w) {
43                 dp[v][s] = dp[p.v][s] + e[i].w;
44                 q.push(Node(v, dp[v][s]));
45             }
46         }
47     }
48 }
49
50 int main() {
51     scanf("%d%d%d", &n, &m, &k);
52     for (int i = 1; i <= m; i++) {
53         int x, y, w;
54         scanf("%d%d%d", &x, &y, &w);
55         add(x, y, w); add(y, x, w);
56     }
57     memset(dp, 0x3f, sizeof dp);
58     for (int i = 0; i < k; i++) {
59         scanf("%d", &key);
60         dp[key][1 << i] = 0;
61     }
62     for (int s = 1; s < (1 << k); s++) {
63         for (int i = 1; i <= n; i++) {
64             for (int subs = s & (s - 1); subs; subs = s & (subs - 1)) {
65                 dp[i][s] = min(dp[i][s], dp[i][subs] + dp[i][subs ^ s]);
66             }
67         }
68     }
69 }

```

```

67         if (dp[i][s] < INF) q.push(Node(i, dp[i][s]));
68     }
69     dijkstra(s);
70 }
71 printf("%d\n", dp[key][(1 << k) - 1]);
72 return 0;
73 }

```

7.7 网络流

7.7.1 最大流/最小割

Ford-Fulkerson 算法

步骤一（贪心）

1. 找到一条由 s 到 t 的只经过 $f(e) < c(e)$ 的路径；
2. 如果不存在该路径，算法结束。否则，沿该路径尽可能增加 $f(e)$ ，返回上一步。

步骤二 1. 利用残余网络寻找 s 到 t 的路径； 2. 若不存在该路径，算法结束。否则，沿该路径尽可能增加流，返回上一步。

残余网络定义了新的边，为原来边的反向边，其容量为：

$$c_f(e) = \begin{cases} f(e), e \notin E \\ c(e) - f(e), e \in E \end{cases}$$

即允许流量回流。

Dinic 算法

$O(|E||V|^2)$.

```

1 struct Edge {
2     int to, cap, rev;
3     Edge(int _to = 0, int _cap = 0, int _rev = 0) : to(_to), cap(_cap), rev(_rev) {}
4 };
5
6 const int MAX_N = 1e2 + 7;
7 vector<Edge> e[MAX_N];
8 int level[MAX_N], iter[MAX_N];
9 int n, m;
10
11 void add(int from, int to, int cap) {
12     e[from].push_back((Edge){to, cap, (int)e[to].size()});
13     e[to].push_back((Edge){from, 0, (int)e[from].size() - 1});
14 }
15
16 bool bfs(int s, int t) {
17     memset(level, 0, sizeof level);
18     queue<int> q;
19     level[s] = 1;
20     q.push(s);
21     while (!q.empty()) {
22         int v = q.front(); q.pop();
23         for (int i = 0; i < e[v].size(); i++) {

```



```

24     Edge &ed = e[v][i];
25     if (ed.cap > 0 && !level[ed.to]) {
26         level[ed.to] = level[v] + 1;
27         q.push(ed.to);
28     }
29 }
30 }
31 return level[t] > 0;
32 }
33
34 int dfs(int v, int t, int f) {
35     if (v == t) return f;
36     for (int &i = iter[v]; i < e[v].size(); i++) {
37         Edge &ed = e[v][i];
38         if (ed.cap > 0 && level[v] < level[ed.to]) {
39             int d = dfs(ed.to, t, min(f, ed.cap));
40             if (d) {
41                 ed.cap -= d;
42                 e[ed.to][ed.rev].cap += d;
43                 return d;
44             }
45         }
46     }
47     return 0;
48 }
49
50 int dinic(int s, int t) {
51     int flow = 0, f;
52     while (bfs(s, t)) {
53         memset(iter, 0, sizeof iter);
54         while (f = dfs(s, t, INF)) {
55             flow += f;
56         }
57     }
58     return flow;
59 }

```

7.7.2 最小费用最大流 (MCMF)

```

1 struct Edge {
2     int to, cap, cost, rev;
3     Edge() {}
4     Edge(int _to, int _cap, int _cost, int _rev)
5         : to(_to), cap(_cap), cost(_cost), rev(_rev) {}
6 };
7
8 int dis[MAX_N];
9 int pre[MAX_N];
10 int tag[MAX_N];
11 bool vis[MAX_N];
12 vector<Edge> e[MAX_N];
13 queue<int> q;

```

```
14
15 void add(int x, int y, int z, int w) {
16     e[x].push_back(Edge(y, z, w, e[y].size()));
17     e[y].push_back(Edge(x, 0, -w, e[x].size() - 1));
18 }
19
20 bool spfa(int s, int t) {
21     memset(dis, 0x3f, sizeof(int) * (n + 5));
22     dis[s] = 0, q.push(s), vis[s] = true;
23     while (!q.empty()) {
24         int x = q.front();
25         q.pop(), vis[x] = false;
26         for (int i = 0; i < e[x].size(); i++) {
27             Edge &ed = e[x][i];
28             if (ed.cap > 0 && dis[ed.to] > dis[x] + ed.cost) {
29                 dis[ed.to] = dis[x] + ed.cost;
30                 pre[ed.to] = x, tag[ed.to] = i;
31                 if (!vis[ed.to]) {
32                     q.push(ed.to), vis[ed.to] = true;
33                 }
34             }
35         }
36     }
37     return dis[t] < INF;
38 }
39
40 pii mcmf(int s, int t) {
41     int flow = 0, cost = 0;
42     while (spfa(s, t)) {
43         int f = INF;
44         for (int i = t; i != s; i = pre[i]) {
45             f = min(f, e[pre[i]][tag[i]].cap);
46         }
47         flow += f, cost += f * dis[t];
48         for (int i = t; i != s; i = pre[i]) {
49             Edge &ed = e[pre[i]][tag[i]];
50             ed.cap -= f, e[i][ed.rev].cap += f;
51         }
52     }
53     return pii(flow, cost);
54 }
```

Chapter 8

Data Structure

8.1 并查集 (Union-Find)

```
1 struct UF {
2     vector<int> fa, rk;
3     UF(int n = 0) {
4         fa.resize(n + 1);
5         rk.resize(n + 1);
6         for (int i = 1; i <= n; ++i) fa[i] = i, rk[i] = 1;
7     }
8     int find(int x) { return x == fa[x] ? x : fa[x] = find(fa[x]); }
9     int merge(int x, int y) {
10         x = find(x), y = find(y);
11         if (rk[x] < rk[y]) swap(x, y);
12         fa[y] = x, rk[x] = max(rk[x], rk[y] + 1);
13         return x;
14     }
15 };
```

8.2 哈希表

```
1 const int HASHSIZE = 100033, KEYSIZE = 102000;
2 template<typename Key, typename Mapped>
3 struct HashMap {
4     typedef pair<Key, Mapped> pii;
5     pii *End, kv[KEYSIZE];
6     int head[HASHSIZE], nxt[HASHSIZE], tot;
7
8     HashMap() {
9         clear(); End = new pii;
10    }
11    pii *end() {
12        return End;
13    }
14    inline void clear() {
15        memset(head, 0, sizeof head); tot = 0;
16    }
```

```

17 inline void insert(const Key &x, const Mapped &y) {
18     kv[++tot] = pii(x, y);
19     int xm = x % HASHSIZE;
20     nxt[tot] = head[xm];
21     head[xm] = tot;
22 }
23 inline pii *find(const Key &x) {
24     int xm = x % HASHSIZE;
25     for (int _p = head[xm]; _p; _p = nxt[_p]) {
26         if (kv[_p].first == x) return kv + _p;
27     }
28     return End;
29 }
30 Mapped &operator[] (const Key &idx) {
31     if (find(idx) == End) insert(idx, 0);
32     return find(idx)->second;
33 }
34 int count(const Key &idx) {
35     if (find(idx) == End) return 0;
36     return 1;
37 }
38 };

```

8.3 Splay Tree

$O(\log n)$ 。

```

1 struct Splay {
2     vector<int> key, siz, cnt, fa, tag;
3     vector<vector<int>>> ch;
4     int rt, tot;
5     Splay(int len = 0) {
6         rt = tot = 0;
7         key.resize(len, 0);
8         siz.resize(len, 0);
9         cnt.resize(len, 0);
10        fa.resize(len, 0);
11        tag.resize(len, 0); // lazy tag when using interval operations
12        ch.resize(len, vector<int> (2, 0));
13    }
14
15    void pushup(int x) {
16        if (x) {
17            siz[x] = cnt[x] + siz[ch[x][0]] + siz[ch[x][1]];
18        }
19    }
20    inline int get(const int &x) { return ch[fa[x]][1] == x; }
21    inline void clear(const int &x) {
22        ch[x][0] = ch[x][1] = siz[x] = cnt[x] = fa[x] = key[x] = 0;
23    }
24    void pushdown(int x) { // pushdown the lazy tag
25        if (x && tag[x]) {

```

```

26         swap(ch[x][0], ch[x][1]);
27         tag[x] = 0;
28         tag[ch[x][0]] ^= 1;
29         tag[ch[x][1]] ^= 1;
30     }
31 }
32 // blanced BST
33 void rotate(int x) {
34     int f = fa[x], ff = fa[f], p = get(x);
35     ch[f][p] = ch[x][p ^ 1], fa[ch[f][p]] = f;
36     fa[f] = x, ch[x][p ^ 1] = f, fa[x] = ff;
37     if (ff) ch[ff][ch[ff][1] == f] = x;
38     pushup(f), pushup(x);
39 }
40 void splay(int x, int g) {
41     for (int f; (f = fa[x]) != g; rotate(x)) {
42         if (fa[f] != g) rotate(get(x) == get(f) ? f : x);
43     }
44     if (!g) rt = x;
45 }
46 int find(int x) {
47     if (!rt) return -1;
48     int u = rt, ans = 1, flag;
49     while (ch[u][flag = (x > key[u])] && key[u] != x) ans += (siz[ch[u][0]] + cnt
        [u]) * flag, u = ch[u][flag];
50     ans += siz[ch[u][0]];
51     splay(u, 0);
52     return key[u] == x ? ans : -1;
53 }
54 void insert(int x) {
55     int u = rt, f = 0;
56     while (u && key[u] != x) f = u, u = ch[u][x > key[u]];
57     if (u) {
58         ++cnt[u];
59     } else {
60         u = ++tot;
61         if (f) ch[f][x > key[f]] = u;
62         ch[u][0] = ch[u][1] = 0;
63         fa[u] = f, key[u] = x, cnt[u] = siz[u] = 1;
64     }
65     splay(u, 0);
66 }
67 int pre(int x) {
68     find(x);
69     int u = rt;
70     if (key[u] < x) return u;
71     for (u = ch[u][0]; ch[u][1];) u = ch[u][1];
72     return u;
73 }
74 int nxt(int x) {
75     find(x);
76     int u = rt;
77     if (key[u] > x) return u;

```

```

78     for (u = ch[u][1]; ch[u][0];) u = ch[u][0];
79     return u;
80 }
81 bool del(int x) {
82     find(x);
83     if (!cnt[rt]) return false;
84     int of = rt;
85     if (cnt[rt] > 1) {
86         --cnt[rt];
87     } else if (!ch[rt][0] && !ch[rt][1]) {
88         clear(rt), rt = 0;
89     } else if (!ch[rt][0]) {
90         rt = ch[rt][1], fa[rt] = 0, clear(of);
91     } else if (!ch[rt][1]) {
92         rt = ch[rt][0], fa[rt] = 0, clear(of);
93     } else {
94         int l = pre(x);
95         splay(l, 0);
96         ch[rt][1] = ch[of][1], fa[ch[of][1]] = rt, clear(of);
97         pushup(rt);
98     }
99     return true;
100 }
101 int kth(int x) {
102     if (tot < x) return -1;
103     int u = rt, tmp, left;
104     while (true) {
105         pushdown(u);
106         left = ch[u][0];
107         if (x <= siz[left]) {
108             u = left;
109         } else {
110             tmp = siz[left] + cnt[u];
111             if (tmp >= x) return u;
112             x -= tmp;
113             u = ch[u][1];
114         }
115     }
116 }
117 inline int pre_key(const int &x) { return key[pre(x)]; }
118 inline int nxt_key(const int &x) { return key[nxt(x)]; }
119 inline int kth_key(const int &x) { return key[kth(x)]; }
120 // interval operations
121 int build(int l, int r, int p) { // a little slow
122     if (l > r) return 0;
123     int mid = (l + r) >> 1, u = ++tot;
124     key[u] = mid, fa[u] = p, ++cnt[u];
125     ch[u][0] = build(l, mid - 1, u);
126     ch[u][1] = build(mid + 1, r, u);
127     pushup(u);
128     return u;
129 }
130 void reverse(int l, int r) {

```

```

131     if (l >= r) return;
132     int L = kth(l), R = kth(r);
133     splay(L, 0);
134     splay(R, L);
135     tag[ch[ch[rt][1]][0]] ^= 1;
136 }
137 };

```

8.4 Treap (Tree-heap)

$O(\log n)$ 。

```

1  mt19937 rnd(time(0));
2  struct Treap {
3      vector<int> siz, key, pri;
4      vector<vector<int>> ch;
5      int rt, tot;
6      Treap(int len = 0) {
7          siz.resize(len, 0);
8          key.resize(len, 0);
9          pri.resize(len, 0);
10         ch.resize(len, vector<int>(2, 0));
11         rt = tot = 0;
12     }
13
14     void pushup(int x) {
15         if (x) {
16             siz[x] = siz[ch[x][0]] + siz[ch[x][1]] + 1;
17         }
18     }
19     int new_node(int x) {
20         int u = ++tot;
21         siz[u] = 1, key[u] = x, pri[u] = rnd() % 998244353;
22         ch[u][0] = ch[u][1] = 0;
23         return u;
24     }
25     void clear(int x) {
26         siz[x] = key[x] = pri[x] = ch[x][0] = ch[x][1] = 0;
27     }
28
29     int merge(int u, int v) {
30         if (!u || !v) return u + v;
31         if (pri[u] > pri[v]) {
32             ch[u][1] = merge(ch[u][1], v);
33             pushup(u);
34             return u;
35         } else {
36             ch[v][0] = merge(u, ch[v][0]);
37             pushup(v);
38             return v;
39         }
40     }

```

```

41  int merge(pii p) { return merge(p.first, p.second); }
42  pii split(int u, int x) { // ch[u][0] <= k, ch[u][1] > k
43      if (!u) return pii(0, 0);
44      if (x < key[u]) {
45          pii o = split(ch[u][0], x);
46          ch[u][0] = o.second;
47          pushup(u);
48          return pii(o.first, u);
49      } else {
50          pii o = split(ch[u][1], x);
51          ch[u][1] = o.first;
52          pushup(u);
53          return pii(u, o.second);
54      }
55  }
56  void insert(int x) {
57      int u = new_node(x);
58      pii o = split(rt, x);
59      o.first = merge(o.first, u);
60      rt = merge(o);
61  }
62  bool del(int x) {
63      pii o = split(rt, x - 1), p = split(o.second, x);
64      if (!p.first) return false;
65      int u = merge(ch[p.first][0], ch[p.first][1]);
66      clear(p.first);
67      rt = merge(o.first, merge(u, p.second));
68      return true;
69  }
70  int find(int x) {
71      pii o = split(rt, x - 1);
72      int res = siz[o.first] + 1;
73      rt = merge(o);
74      return res;
75  }
76  int kth(int rt, int x) {
77      int u = rt;
78      if (tot < x) return -1;
79      while (siz[ch[u][0]] + 1 != x) {
80          if (siz[ch[u][0]] >= x) {
81              u = ch[u][0];
82          } else {
83              x -= siz[ch[u][0]] + 1;
84              u = ch[u][1];
85          }
86      }
87      return key[u];
88  }
89  int kth(int x) { return kth(rt, x); }
90  int pre(int x) {
91      pii o = split(rt, x - 1);
92      int res = kth(o.first, siz[o.first]);
93      rt = merge(o);

```



```
94     return res;
95 }
96 int nxt(int x) {
97     pii o = split(rt, x);
98     int res = kth(o.second, 1);
99     rt = merge(o);
100    return res;
101 }
102 };
```


Chapter 9

String Theory

9.1 AC 自动机

$$O(\sum_{i=1}^n |s_i| + n|\Sigma| + |T|)$$

```
1 //luogu P3796 【模板】AC自动机 (加强版)
2 #include <iostream>
3 #include <cstdio>
4 #include <cstring>
5 #include <queue>
6 #include <vector>
7
8 using namespace std;
9
10 const int MAX_N = 157;
11 const int L = 77;
12 const int MAX_T = 1e6 + 7;
13
14 void clear(queue<int> &q) {
15     queue<int> empty;
16     swap(empty, q);
17 }
18
19 struct AC {
20     int tot;
21     vector<vector<int>> tr;
22     vector<int> fail, id, val, cnt;
23     queue<int> q;
24
25     AC(int len, int N, int chSize) : tot(0) {
26         fail.resize(len, 0);
27         id.resize(len, 0);
28         val.resize(len, 0);
29         cnt.resize(N, 0);
30         tr.resize(len, vector<int> (chSize, 0));
31         clear(q);
32     }
33 }
```

```

34 void insert(char *s, int index) {
35     int u = 0, n = strlen(s + 1);
36     for (int i = 1; i <= n; ++i) {
37         int e = s[i] - 'a';
38         if (!tr[u][e]) tr[u][e] = ++tot;
39         u = tr[u][e];
40     }
41     id[u] = index;
42 }
43
44 void build() {
45     int u;
46     for (int i = 0; i < 26; ++i) {
47         if (tr[0][i]) q.push(tr[0][i]);
48     }
49     while (!q.empty()) {
50         u = q.front(), q.pop();
51         for (int i = 0; i < 26; ++i) {
52             if (tr[u][i]) {
53                 fail[tr[u][i]] = tr[fail[u]][i];
54                 q.push(tr[u][i]);
55             } else {
56                 tr[u][i] = tr[fail[u]][i];
57             }
58         }
59     }
60 }
61
62 int query(char *t) {
63     int u = 0, res = 0, n = strlen(t + 1);
64     for (int i = 1; i <= n; ++i) {
65         u = tr[u][t[i] - 'a'];
66         for (int j = u; j; j = fail[j]) ++val[j];
67     }
68     for (int i = 1; i <= tot; ++i) {
69         if (id[i]) res = max(res, val[i]), cnt[id[i]] = val[i];
70     }
71     return res;
72 }
73 };
74
75 char s[MAX_N][L], t[MAX_T];
76 int n, mx;
77
78 int main() {
79     while (true) {
80         scanf("%d", &n);
81         if (!n) break;
82         AC ac((n + 1) * L, n + 1, 27);
83         for (int i = 1; i <= n; ++i) scanf("%s", s[i] + 1), ac.insert(s[i], i);
84         ac.build();
85         scanf("%s", t + 1);
86         mx = ac.query(t);

```

```

87     printf("%d\n", mx);
88     for (int i = 1; i <= n; ++i) if (ac.cnt[i] == mx) puts(s[i] + 1);
89 }
90 return 0;
91 }

```

9.2 前缀函数 (Prefix-Function)

$O(n)$ 计算 π 函数。

```

1 void calc_pi(string s) {
2     int n = s.size();
3     vector<int> pi(n);
4     for (int i = 1; i < n; ++i) {
5         int j = pi[i - 1];
6         while (j && s[j] != s[i]) j = pi[j - 1];
7         pi[i] = j + (s[i] == s[j]);
8     }
9 }

```

9.3 KMP 自动机

$O(n|\Sigma|)$ 。

```

1 struct KMP {
2     int n, st;
3     vector<int> pi;
4     vector<vector<int>> tr;
5
6     KMP(int _n) : n(_n) {
7         pi.resize(_n, 0);
8         tr.resize(_n, vector<int> (26, 0));
9     }
10
11     void reset(int _st = 0) {
12         st = _st;
13     }
14
15     void calc_pi(char *s) {
16         for (int i = 1; i < n; ++i) {
17             int j = pi[i - 1];
18             while (j && s[i] != s[j]) j = pi[j - 1];
19             pi[i] = j + (s[i] == s[j]);
20         }
21     }
22
23     void build(char *s) {
24         calc_pi(s);
25         for (int i = 0; i < n; ++i) {
26             for (int c = 0; c < 26; ++c) {
27                 if (i && ('A' + c != s[i])) {

```

```

28         tr[i][c] = tr[pi[i - 1]][c];
29     } else {
30         tr[i][c] = i + ('A' + c == s[i]);
31     }
32 }
33 }
34 }
35
36 int query(int c) {
37     st = tr[st][c];
38     return st;
39 }
40 };

```

9.4 后缀数组 (SA)

9.4.1 后缀排序

倍增

基于倍增的排序, $O(n \log n)$ 。

```

1 bool cmp(const int &x, const int &y, const int &w) {
2     return oldrk[x] == oldrk[y] && oldrk[x + w] == oldrk[y + w];
3 }
4
5 void suffix_sort(char *s) {
6     int m = 300, p;
7     for (int i = 1; i <= n; ++i) ++cnt[rk[i] = s[i]];
8     for (int i = 1; i <= m; ++i) cnt[i] += cnt[i - 1];
9     for (int i = n; i; --i) sa[cnt[rk[i]]--] = i;
10    for (int w = 1; w < n; w <= 1, m = p) {
11        p = 0;
12        for (int i = n; i > n - w; --i) id[++p] = i;
13        for (int i = 1; i <= n; ++i) {
14            if (sa[i] > w) id[++p] = sa[i] - w;
15        }
16        for (int i = 0; i <= m; ++i) cnt[i] = 0;
17        for (int i = 1; i <= n; ++i) ++cnt[px[i] = rk[id[i]]];
18        for (int i = 1; i <= m; ++i) cnt[i] += cnt[i - 1];
19        for (int i = n; i; --i) sa[cnt[px[i]]--] = id[i];
20        swap(oldrk, rk);
21        p = 0;
22        for (int i = 1; i <= n; ++i) {
23            rk[sa[i]] = cmp(sa[i], sa[i - 1], w) ? p : ++p;
24        }
25    }
26 }

```

DC3

9.4.2 最长公共前缀 (LCP)

定义 9.4.1 (最长公共前缀). 对于两个字符串 S 和 T , 其最长公共前缀 (LCP) 即为 x , x 是满足 $\forall 1 \leq i \leq x, S_i = T_i$ 的最大整数。

并记 $lcp(i, j)$ 表示后缀 i 和后缀 j 的 LCP, $LCP(i, j) = lcp(SA[i], SA[j])$ 。

性质 9.4.1. $LCP(i, j) = LCP(j, i)$ 。

性质 9.4.2. $LCP(i, i) = n - |SA[i]| + 1$ 。

引理 9.4.1 (LCP 引理). 对任意 $1 \leq i < j < k \leq n$, $LCP(i, k) = \min\{LCP(i, j), LCP(j, k)\}$ 。

证明. 记 $LC(i, k) = p$, $SA[i]$ 表示的字符串为 AB , $SA[k]$ 表示 AD , $|A| = p$ 。然后根据 $SA[i, i+1, \dots, k]$ 表示字符串有序再随便反证一下就行了。□

由 LCP 引理立即得到 LCP 定理:

定理 9.4.1 (LCP 定理). 对任意 $1 \leq i < j \leq n$, 有 $LCP(i, j) = \min\{LCP(k-1, k) | i < k \leq j\}$ 。

推论 9.4.1. 对 $i \leq j < k$, 有 $LCP(i, k) \leq LCP(j, k)$ 。

由此进一步定义 $height$ 和 h 数组。

定义 9.4.2 ($height$ 数组).

$$height[i] = \begin{cases} LCP(i-1, i), & 1 < i \leq n \\ 0, & i = 1 \end{cases}$$

定义 9.4.3 (h 数组). $h[i] = height[Rank[i]]$, 也即 $height[i] = h[SA[i]]$ 。

定理 9.4.2. 对任意 $i > 1$ 且 $Rank[i] > 1$, 均有 $h[i] \geq h[i-1] - 1$ 。

证明. 仿照引理的证明方法。□

根据上面定理, 即可 $O(n)$ 暴力求解 $height$ 数组。

```

1 void calc_height(char *s) {
2     for (int i = 1, k = 0; i <= n; ++i) {
3         if (k) --k;
4         int j = sa[rk[i] - 1];
5         while (s[i + k] == s[j + k]) ++k;
6         height[rk[i]] = k;
7     }
8 }

```

9.5 后缀自动机 (SAM)

$O(n)$ 。

```

1 struct SAM {
2     int cnt, last;
3     vector<int> len, link;
4     vector<vector<int>> tr;
5     // vector<map<int, int>> tr; // Use map but there'll be a log.
6 }

```

```

7   SAM(int strLen, int chSize) : cnt(1), last(1) {
8       len.resize(strLen * 2, 0);
9       link.resize(strLen * 2, 0);
10      tr.resize(strLen * 2, vector<int> (chSize, 0));
11      // tr.resize(strLen * 2, map<int, int> ());
12  }
13
14  void extend(int c) {
15      int p, cur = ++cnt;
16      len[cur] = len[last] + 1;
17      for (p = last; p && (!tr[p][c]); p = link[p]) tr[p][c] = cur;
18      if (!p) {
19          link[cur] = 1;
20      } else {
21          int q = tr[p][c];
22          if (len[q] == len[p] + 1) {
23              link[cur] = q;
24          } else {
25              int clone = ++cnt;
26              len[clone] = len[p] + 1, tr[clone] = tr[q], link[clone] = link[q];
27              for (; p && tr[p][c] == q; p = link[p]) tr[p][c] = clone;
28              link[cur] = link[q] = clone;
29          }
30      }
31      last = cur;
32  }
33  };

```

9.6 Manacher

$O(n)$ 。

```

1  vector<int> manacher(string s) {
2      n = s.size();
3      vector<int> d(n);
4      for (int i = 0, l = 0, r = -1; i < n; ++i) {
5          int k = (i > r) ? 1 : min(d[l + r - i], r - i + 1);
6          while ((i - k >= 0) && (i + k < n) && (s[i - k] == s[i + k])) ++k;
7          d[i] = k--;
8          if (i + k > r) l = i - k, r = i + k;
9      }
10     return d;
11 }

```