

ACM TEMPLATE



Fibonacci's Rabbit

Last build at May 11, 2019

Contents

1	数论	2
1.1	素数筛	2
1.2	唯一分解定理	2
1.3	欧拉函数	3
1.4	扩展欧几里得	3
1.5	逆元	3
1.6	快速幂取模	4
1.7	大整数取模	4
1.8	中国剩余定理	4
2	数学	5
2.1	矩阵	5
2.2	杜教筛	6
2.3	快速傅里叶变换	8
2.4	快速数论变换	9
3	字符串	10
3.1	字符串最小最大表示	10
3.2	kmp 算法	11
3.3	z 函数	11
3.4	manacher	11
3.5	字典树	12
3.6	ac 自动机	13
3.7	后缀数组	14
3.7.1	后缀数组-倍增法	14
3.7.2	后缀数组-dc3	16
3.7.3	后缀数组-快排	17
3.8	字符串哈希	18
3.9	字符串分割	18
3.9.1	按字符分割	18
3.9.2	按字符串分割	18
3.9.3	按字符分割 (STL)	19
4	动态规划	19
4.1	物品无限的背包问题	19
4.2	0-1 背包	20
4.3	线性结构上的动态规划	20
4.3.1	LIS (最长上升子序列)	20
4.3.2	LCS (最长公共子序列)	21
5	数据结构	21
5.1	线段树递归	21
5.2	线段树非递归	22
5.3	主席树	23
6	常用 STL	24
6.1	pair 的 hash	24

1 数论

1.1 素数筛

```

1 bool vis[maxn];
2 int primes[maxn];
3 int primes_len;
4
5 void sieve(int n) {
6     int m = (int)sqrt(n + 0.5);
7     memset(vis, 0, sizeof(vis));
8     for (int i = 2; i <= m; i++)
9         if (!vis[i]) {
10             for (int j = i * i; j <= n; j += i) vis[j] = true;
11         }
12 }
13
14 int gen_primes(int n) {
15     sieve(n);
16     int c = 0;
17     for (int i = 2; i <= n; i++)
18         if (!vis[i]) {
19             primes[c++] = i;
20         }
21     return c;
22 }

```

1.2 唯一分解定理

```

1 const int maxn = 100;
2
3 // 求因子个数
4 int cnt(int n) {
5     int s = 1;
6     for (int i = 2; i * i <= n; i++) {
7         if (n % i == 0) {
8             int a = 0;
9             while (n % i == 0) {
10                 n /= i;
11                 a++;
12             }
13             s = s * (a + 1);
14         }
15     }
16     if (n > 1) s = s * 2;
17     return s;
18 }
19
20 // 求因子的和
21 int sum(int n) {
22     int s = 1;
23     for (int i = 2; i * i <= n; i++) {
24         if (n % i == 0) {
25             int a = 1;
26             while (n % i == 0) {
27                 n /= i;
28                 a *= i;
29             }
30             s = s * (a * i - 1) / (i - 1);
31         }
32     }
33     if (n > 1) s = s * (1 + n);
34     return s;
35 }
36
37 const int MOD = 1e9 + 7;
38
39 // 同时求cnt和sum
40 // sum取模
41 void solve(int n, ll& sum, ll& cnt) {
42     for (int i = 2; i * i <= n; i++) {
43         if (n % i == 0) {
44             ll a = 1;
45             ll t = 0;
46             while (n % i == 0) {
47                 n /= i;
48                 a = a * i % MOD;
49                 t++;

```

```

50     }
51     cnt *= t + 1;
52     sum = sum * ((a * i - 1) / (i - 1) % MOD) % MOD;
53 }
54 }
55 if (n > 1) {
56     sum = sum * (1 + n) % MOD;
57     cnt *= 2;
58 }
59 }
60
61 int primes[maxn];
62 int primes_len;
63
64 // 打素数表, 只遍历素数
65 ll cnt(ll n) {
66     ll s = 1;
67     for (int i = 0; i < primes_len && primes[i] * primes[i] <= n; ++i) {
68         if (n % primes[i] == 0) {
69             ll a = 0;
70             while (n % primes[i] == 0) {
71                 n /= primes[i];
72                 a++;
73             }
74             s = s * (a + 1);
75         }
76     }
77     if (n > 1) s = s * 2;
78     return s;
79 }

```

1.3 欧拉函数

```

1 int euler_phi(int n) {
2     int m = (int)sqrt(n + 0.5);
3     int ans = n;
4     for (int i = 2; i <= m; i++)
5         if (n % i == 0) {
6             ans = ans / i * (i - 1);
7             while (n % i == 0) n /= i;
8         }
9     if (n > 1) ans = ans / n * (n - 1);
10 }
11
12 int phi[maxn];
13
14 // 求1~n的欧拉函数值
15 void phi_table(int n) {
16     for (int i = 2; i <= n; i++) phi[i] = 0;
17     phi[1] = 1;
18     for (int i = 2; i <= n; i++)
19         if (!phi[i]) {
20             for (int j = i; j <= n; j += i) {
21                 if (!phi[j]) phi[j] = j;
22                 phi[j] = phi[j] / i * (i - 1);
23             }
24         }
25 }

```

1.4 扩展欧几里得

```

1 void ex_gcd(ll a, ll b, ll& d, ll& x, ll& y) {
2     if (!b) {
3         d = a;
4         x = 1;
5         y = 0;
6     } else {
7         ex_gcd(b, a % b, d, y, x);
8         y -= x * (a / b);
9     }
10 }

```

1.5 逆元

```

1 // 计算模n下a的逆。如果不存在逆，返回-1
2 ll inv(ll a, ll n) {
3     ll d, x, y;
4     ex_gcd(a, n, d, x, y);
5     return d == 1 ? (x + n) % n : -1;
6 }

```

1.6 快速幂取模

```

1 ll fast_pow_mod(ll a, ll p, ll n) {
2     if (p == 0) return 1;
3     a = a % n;
4     ll ans = pow_mod(a, p / 2, n) % n;
5     ans = (ans * ans) % n;
6     if (p % 2 == 1) ans = (ans * a) % n;
7     return ans;
8 }
9
10 ll faster_pow_mod(ll a, ll b, ll c) {
11     ll ans = 1;
12     a = a % c;
13     while (b != 0) {
14         if (b & 1) ans = (ans * a) % c;
15         b >>= 1;
16         a = (a * a) % c;
17     }
18     return ans;
19 }

```

1.7 大整数取模

```

1 // b>0
2 ll big_mod(const char* a, ll b) {
3     int st = 0;
4     if (a[0] == '-') st = 1;
5     int len = strlen(a);
6     if (b < 0) b = -b;
7     long long ans = 0;
8     for (int i = st; i < len; i++) {
9         ans = (ans * 10 + a[i] - '0') % b;
10    }
11    return ans;
12 }

```

1.8 中国剩余定理

```

1 // n个方程: x=a[i](mod m[i]) (0<=i<n)
2 ll china(int n, ll* a, ll* m) {
3     ll M = 1, d, y, x = 0;
4     for (int i = 0; i < n; i++) M *= m[i];
5     for (int i = 0; i < n; i++) {
6         ll w = M / m[i];
7         ex_gcd(m[i], w, d, d, y);
8         x = (x + y * w * a[i]) % M;
9     }
10    return (x + M) % M;
11 }
12
13 // unused
14 long long ex_crt(long long a[], long long n[], int num) {
15     long long n1 = n[0], a1 = a[0], n2, a2, k1, k2, x0, gcd, c;
16     for (int i = 1; i < num; i++) {
17         n2 = n[i], a2 = a[i];
18         c = a2 - a1;
19         gcd = ex_gcd(n1, n2, k1, k2); //解得: n1*k1+n2*k2=gcd(n1,n2)
20         if (c % gcd) {
21             flag = 1;
22             return 0; //无解
23         }
24         x0 = c / gcd * k1; // n1*x0+n2*(c/gcd*k2)=c PS:k1/gcd*c错误!
25         t = n2 / gcd;
26         x0 = (x0 % t + t) % t; //求n1*x0+n2*y=c的x0的最小解
27         a1 += n1 * x0;
28         n1 = n2 / gcd * n1;
29     }
30     return a1;

```

2 数学

2.1 矩阵

线性递推

$$f(n) = a_1 f(n-1) + a_2 f(n-2) + a_3 f(n-3) + \cdots + a_d f(n-d)$$

$$\begin{bmatrix} f(n-d) \\ \vdots \\ f(n-2) \\ f(n-1) \\ f(n) \end{bmatrix} = \begin{bmatrix} & & 1 & & \\ & & & 1 & \\ & & & & \ddots \\ \vdots & & & & & 1 \\ a_d & a_{d-1} & a_{d-2} & \cdots & a_1 \end{bmatrix} \times \begin{bmatrix} f(n-d-1) \\ \vdots \\ f(n-3) \\ f(n-2) \\ f(n-1) \end{bmatrix}$$

二项式定理

1 计算 k 次方和

$$S_n = \sum_{i=1}^n i^k$$

$$\begin{aligned} S_{n+1} &= S_n + (n+1)^k \\ &= S_n + C_k^0 n^k + C_k^1 n^{k-1} + \cdots + C_k^k n^0 \end{aligned}$$

$$\begin{bmatrix} S_{n+1} \\ (n+1)^k \\ (n+1)^{k-1} \\ \vdots \\ (n+1)^0 \end{bmatrix} = \begin{bmatrix} 1 & C_k^0 & C_k^1 & \cdots & C_k^k \\ 0 & C_k^0 & C_k^1 & \cdots & C_k^k \\ 0 & 0 & C_{k-1}^0 & \cdots & C_{k-1}^{k-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & C_0^0 \end{bmatrix} \times \begin{bmatrix} S_n \\ n^k \\ n^{k-1} \\ \vdots \\ n^0 \end{bmatrix}$$

2 更一般的例子

$$S_n = \sum_{i=1}^n (ai + b)^k$$

$$\begin{aligned} S_n &= [a(n-1) + (a+b)]^k + S_{n-1} \\ &= C_k^0 a^k (n-1)^k + C_k^1 a^{k-1} (a+b)(n-1)^{k-1} + \cdots + C_k^k (a+b)^k + S_{n-1} \end{aligned}$$

$$\begin{bmatrix} cn^k \\ n^{k-1} \\ \vdots \\ n^0 \\ S_n \end{bmatrix} = \begin{bmatrix} C_k^0 & C_k^1 & \cdots & C_k^k & 0 \\ 0 & C_{k-1}^0 & \cdots & C_{k-1}^{k-1} & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & \cdots & C_0^0 & 0 \\ C_k^0 a^k & C_k^1 a^{k-1} (a+b) & \cdots & C_k^k (a+b)^k & 1 \end{bmatrix} \times \begin{bmatrix} n^k \\ n^{k-1} \\ \vdots \\ n^0 \\ S_n \end{bmatrix}$$

3 更复杂的例子

$$S_n = \sum_{i=1}^n i^k k^i$$

$$\begin{aligned} S_n &= n^k k^n + S_{n-1} \\ &= (n-1+1)^k k^{(n-1)+1} + S_{n-1} \\ &= C_k^0 (n-1)^k k(n-1) + 1 + C_k^1 (n-1)^{k-1} k^{(n-1)+1} + \cdots + C_k^k (n-1)^0 k^{(n-1)+1} + S_{n-1} \end{aligned}$$

$$\begin{bmatrix} n^k k^{n+1} \\ n^{k-1} k^{n+1} \\ \vdots \\ n^0 k^{n+1} \\ S_n \end{bmatrix} = \begin{bmatrix} kC_k^0 & kC_k^1 & \cdots & kC_k^k & 0 \\ 0 & kC_{k-1}^0 & \cdots & kC_{k-1}^{k-1} & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & \cdots & kC_0^0 & 0 \\ C_k^0 & C_k^1 & \cdots & C_k^k & 1 \end{bmatrix} \times \begin{bmatrix} (n-1)^k k^{(n-1)+1} \\ (n-1)^{k-1} k^{(n-1)+1} \\ \vdots \\ (n-1)^0 k^{(n-1)+1} \\ S_{n-1} \end{bmatrix}$$

```

1  const int maxn=100;
2  const int MOD=1e9+7;
3
4  struct Matrix {
5      double a[3][3];
6      Matrix inverse() { //求三阶矩阵的行列式和逆矩阵
7          double det = a[0][0] * a[1][1] * a[2][2] + a[0][1] * a[1][2] * a[2][0] + a[0][2] * a[1][0] * a[2][1] -
8              a[0][2] * a[1][1] * a[2][0] -
9              a[0][1] * a[1][0] * a[2][2] - a[0][0] * a[1][2] * a[2][1];
10         Matrix ret;
11         ret.a[0][0] = a[1][1] * a[2][2] - a[1][2] * a[2][1];
12         ret.a[1][0] = (a[1][0] * a[2][2] - a[1][2] * a[2][0]) * (-1);
13         ret.a[2][0] = (a[1][0] * a[2][1] - a[1][1] * a[2][0]);
14         ret.a[0][1] = (a[0][1] * a[2][2] - a[0][2] * a[2][1]) * (-1);
15         ret.a[0][2] = a[0][1] * a[1][2] - a[0][2] * a[1][1];
16         ret.a[1][1] = a[0][0] * a[2][2] - a[0][2] * a[2][0];
17         ret.a[2][1] = (a[0][0] * a[2][1] - a[0][1] * a[2][0]) * (-1);
18         ret.a[1][2] = (a[0][0] * a[1][2] - a[0][2] * a[1][0]) * (-1);
19         ret.a[2][2] = a[0][0] * a[1][1] - a[0][1] * a[1][0];
20         for(int i=0;i<3;i++){
21             for(int j=0;j<3;j++){ ret.a[i][j] /= det; }
22         }
23     }
24 };
25
26 struct Matrix {
27     ll a[maxn][maxn];
28 };
29
30 //若矩阵太大，返回值写在参数里
31 //中间结果用全局变量保存，最好不要重复使用
32 Matrix mul(const Matrix& l, const Matrix& r, int len) {
33     Matrix c;
34     for (int i = 0; i < len; i++) {
35         for (int j = 0; j < len; j++) {
36             c.a[i][j] = 0;
37             for (int k = 0; k < len; k++) {
38                 c.a[i][j] = (c.a[i][j] + (l.a[i][k] * r.a[k][j]) % MOD) % MOD;
39             }
40         }
41     }
42     return c;
43 }
44
45 Matrix pow_mod(Matrix x, ll n, int len) {
46     Matrix ans;
47     memset(ans.a, 0, sizeof(ans.a));
48     for (int i = 0; i < len; i++) ans.a[i][i] = 1;
49     while (n) {
50         if (n & 1) ans = mul(ans, x, len);
51         x = mul(x, x, len);
52         n >>= 1;
53     }
54     return ans;
55 }
56
57 Matrix add(const Matrix& l, const Matrix& r, int len) {
58     Matrix c;
59     for (int i = 0; i < len; i++) {
60         for (int j = 0; j < len; j++) {
61             c.a[i][j] = l.a[i][j] + r.a[i][j];
62             c.a[i][j] %= MOD;
63         }
64     }
65     return c;
66 }
67
68 //倍增法求解 $a^1 + a^2 + \dots + a^n$ 
69 Matrix ad(const Matrix& x, int p) {
70     if (p == 1) return x;
71     Matrix tmp = ad(x, p / 2);
72     Matrix sum = add(tmp, mul(tmp, pow_mod(x, p / 2, N), N), N);
73     if (p & 1) sum = add(sum, pow_mod(x, p, N), N);
74     return sum;
75 }

```

2.2 杜教筛

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #define rep(i, a, n) for (long long i = a; i < n; i++)
5 #define per(i, a, n) for (long long i = n - 1; i >= a; i--)
6 #define pb push_back
7 #define mp make_pair
8 #define all(x) (x).begin(), (x).end()
9 #define fi first
10 #define se second
11 #define SZ(x) ((long long)(x).size())
12 typedef vector<long long> VI;
13 typedef long long ll;
14 typedef pair<long long, long long> PII;
15 const ll mod = 1e9 + 7;
16
17 ll powmod(ll a, ll b) {
18     ll res = 1;
19     a %= mod;
20     assert(b >= 0);
21     for (; b; b >>= 1) {
22         if (b & 1) res = res * a % mod;
23         a = a * a % mod;
24     }
25     return res;
26 }
27 // head
28
29 long long _, n;
30 namespace linear_seq {
31     const long long N = 10010;
32     ll res[N], base[N], _c[N], _md[N];
33
34     vector<long long> Md;
35     void mul(ll *a, ll *b, long long k) {
36         rep(i, 0, k + k) _c[i] = 0;
37         rep(i, 0, k) if (a[i]) rep(j, 0, k) _c[i + j] = (_c[i + j] + a[i] * b[j]) % mod;
38         for (long long i = k + k - 1; i >= k; i--)
39             if (_c[i]) rep(j, 0, SZ(Md)) _c[i - k + Md[j]] = (_c[i - k + Md[j]] - _c[i] * _md[Md[j]]) %
                mod;
40         rep(i, 0, k) a[i] = _c[i];
41     }
42     long long solve(ll n, VI a, VI b) { // a 系数 b 初值 b[n+1]=a[0]*b[n]+...
43         // printf("%d\n",SZ(b));
44         ll ans = 0, pnt = 0;
45         long long k = SZ(a);
46         assert(SZ(a) == SZ(b));
47         rep(i, 0, k) _md[k - 1 - i] = -a[i];
48         _md[k] = 1;
49         Md.clear();
50         rep(i, 0, k) if (_md[i] != 0) Md.push_back(i);
51         rep(i, 0, k) res[i] = base[i] = 0;
52         res[0] = 1;
53         while ((1ll << pnt) <= n) pnt++;
54         for (long long p = pnt; p >= 0; p--) {
55             mul(res, res, k);
56             if ((n >> p) & 1) {
57                 for (long long i = k - 1; i >= 0; i--) res[i + 1] = res[i];
58                 res[0] = 0;
59                 rep(j, 0, SZ(Md)) res[Md[j]] = (res[Md[j]] - res[k] * _md[Md[j]]) % mod;
60             }
61         }
62         rep(i, 0, k) ans = (ans + res[i] * b[i]) % mod;
63         if (ans < 0) ans += mod;
64         return ans;
65     }
66     VI BM(VI s) {
67         VI C(1, 1), B(1, 1);
68         long long L = 0, m = 1, b = 1;
69         rep(n, 0, SZ(s)) {
70             ll d = 0;
71             rep(i, 0, L + 1) d = (d + (ll)C[i] * s[n - i]) % mod;
72             if (d == 0) ++m;
73             else if (2 * L <= n) {
74                 VI T = C;
75                 ll c = mod - d * powmod(b, mod - 2) % mod;
76                 while (SZ(C) < SZ(B) + m) C.pb(0);
77                 rep(i, 0, SZ(B)) C[i + m] = (C[i + m] + c * B[i]) % mod;
78                 L = n + 1 - L;
79             }

```



```

80         B = T;
81         b = d;
82         m = 1;
83     } else {
84         ll c = mod - d * powmod(b, mod - 2) % mod;
85         while (SZ(C) < SZ(B) + m) C.pb(0);
86         rep(i, 0, SZ(B)) C[i + m] = (C[i + m] + c * B[i]) % mod;
87         ++m;
88     }
89 }
90 return C;
91 }
92 long long gao(VI a, ll n) {
93     VI c = BM(a);
94     c.erase(c.begin());
95     rep(i, 0, SZ(c)) c[i] = (mod - c[i]) % mod;
96     return solve(n, c, VI(a.begin(), a.begin() + SZ(c)));
97 }
98 }; // namespace linear_seq
99
100 int main() {
101     while (~scanf("%I64d", &n)) {
102         printf("%I64d\n", linear_seq::gao(VI{1, 5, 11, 36, 95, 281, 781, 2245, 6336, 18061, 51205}, n - 1));
103     }
104 }

```

2.3 快速傅里叶变换

```

1  #include <math.h>
2  #include <stdio.h>
3  #include <string.h>
4  #include <algorithm>
5  #include <iostream>
6  using namespace std;
7
8  const double PI = acos(-1.0);
9
10 struct Complex {
11     double r, i;
12     Complex(double _r = 0.0, double _i = 0.0) {
13         r = _r;
14         i = _i;
15     }
16     Complex operator+(const Complex &b) { return Complex(r + b.r, i + b.i); }
17     Complex operator-(const Complex &b) { return Complex(r - b.r, i - b.i); }
18     Complex operator*(const Complex &b) {
19         return Complex(r * b.r - i * b.i, r * b.i + i * b.r);
20     }
21 };
22 /*
23  * 进行FFT和IFFT前的反转变换。
24  * 位置i和 (i二进制反转后位置) 互换
25  * len必须去2的幂
26  */
27 void change(Complex y[], int len) {
28     int i, j, k;
29     for (i = 1, j = len / 2; i < len - 1; i++) {
30         if (i < j) swap(y[i], y[j]);
31         // 交换互为小标反转的元素, i<j保证交换一次
32         // i做正常的+1, j左反转类型的+1,始终保持i和j是反转的
33         k = len / 2;
34         while (j >= k) {
35             j -= k;
36             k /= 2;
37         }
38         if (j < k) j += k;
39     }
40 }
41 /*
42  * 做FFT
43  * len必须为2^k形式,
44  * on==1时是DFT, on==-1时是IDFT
45  */
46 void fft(Complex y[], int len, int on) {
47     change(y, len);
48     for (int h = 2; h <= len; h <= 1) {
49         Complex wn(cos(-on * 2 * PI / h), sin(-on * 2 * PI / h));
50         for (int j = 0; j < len; j += h) {

```

```

51     Complex w(1, 0);
52     for (int k = j; k < j + h / 2; k++) {
53         Complex u = y[k];
54         Complex t = w * y[k + h / 2];
55         y[k] = u + t;
56         y[k + h / 2] = u - t;
57         w = w * wn;
58     }
59 }
60 }
61 if (on == -1)
62     for (int i = 0; i < len; i++) y[i].r /= len;
63 }
64
65 const int MAXN = 200010;
66 Complex x1[MAXN], x2[MAXN];
67 char str1[MAXN / 2], str2[MAXN / 2];
68 int sum[MAXN];
69
70 int main() {
71     while (scanf("%s%s", str1, str2) == 2) {
72         int len1 = strlen(str1);
73         int len2 = strlen(str2);
74         int len = 1;
75         while (len < len1 * 2 || len < len2 * 2) len <<= 1;
76         for (int i = 0; i < len1; i++)
77             x1[i] = Complex(str1[len1 - 1 - i] - '0', 0);
78         for (int i = len1; i < len; i++) x1[i] = Complex(0, 0);
79         for (int i = 0; i < len2; i++)
80             x2[i] = Complex(str2[len2 - 1 - i] - '0', 0);
81         for (int i = len2; i < len; i++) x2[i] = Complex(0, 0);
82         //求DFT
83         fft(x1, len, 1);
84         fft(x2, len, 1);
85         for (int i = 0; i < len; i++) x1[i] = x1[i] * x2[i];
86         fft(x1, len, -1);
87         for (int i = 0; i < len; i++) sum[i] = (int)(x1[i].r + 0.5);
88         for (int i = 0; i < len; i++) {
89             sum[i + 1] += sum[i] / 10;
90             sum[i] %= 10;
91         }
92         len = len1 + len2 - 1;
93         while (sum[len] <= 0 && len > 0) len--;
94         for (int i = len; i >= 0; i--) printf("%c", sum[i] + '0');
95         printf("\n");
96     }
97     return 0;
98 }

```

2.4 快速数论变换

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  inline int read() {
5      int x = 0, f = 1;
6      char ch = getchar();
7      while (ch < '0' || ch > '9') {
8          if (ch == '-') f = -1;
9          ch = getchar();
10     }
11     while (ch <= '9' && ch >= '0') {
12         x = 10 * x + ch - '0';
13         ch = getchar();
14     }
15     return x * f;
16 }
17
18 void print(int x) {
19     if (x < 0) putchar('-'), x = -x;
20     if (x >= 10) print(x / 10);
21     putchar(x % 10 + '0');
22 }
23
24 const int N = 300100, P = 998244353;
25
26 inline int qpow(int x, int y) {
27     int res(1);
28     while (y) {

```

```

29     if (y & 1) res = 1ll * res * x % P;
30     x = 1ll * x * x % P;
31     y >>= 1;
32 }
33 return res;
34 }
35
36 int r[N];
37
38 void ntt(int *x, int lim, int opt) {
39     int i, j, k, m, gn, g, tmp;
40     for (i = 0; i < lim; ++i)
41         if (r[i] < i) swap(x[i], x[r[i]]);
42     for (m = 2; m <= lim; m <= 1) {
43         k = m >> 1;
44         gn = qpow(3, (P - 1) / m);
45         for (i = 0; i < lim; i += m) {
46             g = 1;
47             for (j = 0; j < k; ++j, g = 1ll * g * gn % P) {
48                 tmp = 1ll * x[i + j + k] * g % P;
49                 x[i + j + k] = (x[i + j] - tmp + P) % P;
50                 x[i + j] = (x[i + j] + tmp) % P;
51             }
52         }
53     }
54     if (opt == -1) {
55         reverse(x + 1, x + lim);
56         int inv = qpow(lim, P - 2);
57         for (i = 0; i < lim; ++i) x[i] = 1ll * x[i] * inv % P;
58     }
59 }
60
61 int A[N], B[N], C[N];
62
63 char a[N], b[N];
64
65 int main() {
66     while (~scanf("%s%s", a, b)) {
67         memset(A, 0, sizeof(A));
68         memset(B, 0, sizeof(B));
69         int i, lim(1), n;
70         n = strlen(a);
71         for (i = 0; i < n; ++i) A[i] = a[n - i - 1] - '0';
72         while (lim < (n <= 1)) lim <= 1;
73         n = strlen(b);
74         for (i = 0; i < n; ++i) B[i] = b[n - i - 1] - '0';
75         while (lim < (n <= 1)) lim <= 1;
76         for (i = 0; i < lim; ++i) r[i] = (i & 1) * (lim >> 1) + (r[i >> 1] >> 1);
77         ntt(A, lim, 1);
78         ntt(B, lim, 1);
79         for (i = 0; i < lim; ++i) C[i] = 1ll * A[i] * B[i] % P;
80         ntt(C, lim, -1);
81         int len(0);
82         for (i = 0; i < lim; ++i) {
83             if (C[i] >= 10) len = i + 1, C[i + 1] += C[i] / 10, C[i] %= 10;
84             if (C[i]) len = max(len, i);
85         }
86         while (C[len] >= 10) C[len + 1] += C[len] / 10, C[len] %= 10, len++;
87         for (i = len; ~i; --i) putchar(C[i] + '0');
88         putchar('\n');
89     }
90     return 0;
91 }

```

3 字符串

3.1 字符串最小最大表示

```

1 #include <algorithm>
2 using namespace std;
3
4 // T = sec[k..n-1]+sec[0..k-1]
5 // k为返回值,n为sec的大小,T为sec的最小表示法
6 int get_min(const char* sec, int n) {
7     int k = 0, i = 0, j = 1;
8     while (k < n && i < n && j < n) {
9         if (sec[(i + k) % n] == sec[(j + k) % n]) {
10             k++;

```

```

11     } else {
12         sec[(i + k) % n] > sec[(j + k) % n] ? i = i + k + 1 : j = j + k + 1;
13         if (i == j) i++;
14         k = 0;
15     }
16 }
17 i = min(i, j);
18 return i;
19 }
20
21 int get_max(const char* sec, int n) {
22     int k = 0, i = 0, j = 1;
23     while (k < n && i < n && j < n) {
24         if (sec[(i + k) % n] == sec[(j + k) % n]) {
25             k++;
26         } else {
27             sec[(i + k) % n] < sec[(j + k) % n] ? i = i + k + 1 : j = j + k + 1;
28             if (i == j) i++;
29             k = 0;
30         }
31     }
32     i = min(i, j);
33     return i;
34 }

```

3.2 kmp 算法

以 i 结尾的最小循环节: $i - f[i]$

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 const int maxn = 10000 + 5;
6
7 int f[maxn];
8
9 void get_next(const char *P, int n) {
10     f[0] = 0;
11     f[1] = 0; // 递推边界初值
12     for (int i = 1; i < n; i++) {
13         int j = f[i];
14         while (j && P[i] != P[j]) j = f[j];
15         f[i + 1] = (P[i] == P[j] ? j + 1 : 0);
16     }
17 }

```

3.3 z 函数

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 const int maxn = 1000000 + 5;
6
7 int z[maxn];
8
9 // s 为待匹配的字符串指针
10 // n 为字符串长度
11 // z[i] 是 s 和 s+i 的最大公共前缀长度。
12 void z_function(const char* s, int n) {
13     fill_n(z, n, 0);
14     for (int i = 1, l = 0, r = 0; i < n; ++i) {
15         if (i <= r) z[i] = min(r - i + 1, z[i - l]);
16         while (i + z[i] < n && s[z[i]] == s[i + z[i]]) ++z[i];
17         if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
18     }
19 }

```

3.4 manacher

回文匹配算法 (可用后缀数组代替, 但是比后缀数组简洁得多)

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4

```

```

5  const int maxn = 1000000;
6  int d1[maxn], d2[maxn];
7
8  // s 为字符串,也可以是const string&
9  // n 是字符串长度,即为s.length()
10 // d1为奇数回文长度(算上起点),总长度为d1[*]*2-1
11 // d2为偶数回文长度(算上起点),总长度为d2[*]*2
12 void Manacher(const char* s, int n) {
13     for (int i = 0, l = 0, r = -1; i < n; i++) {
14         int k = (i > r) ? 1 : min(d1[l + r - i], r - i);
15         while (0 <= i - k && i + k < n && s[i - k] == s[i + k]) {
16             k++;
17         }
18         d1[i] = k--;
19         if (i + k > r) {
20             l = i - k;
21             r = i + k;
22         }
23     }
24
25     for (int i = 0, l = 0, r = -1; i < n; i++) {
26         int k = (i > r) ? 0 : min(d2[l + r - i + 1], r - i + 1);
27         while (0 <= i - k - 1 && i + k < n && s[i - k - 1] == s[i + k]) {
28             k++;
29         }
30         d2[i] = k--;
31         if (i + k > r) {
32             l = i - k - 1;
33             r = i + k;
34         }
35     }
36 }
37
38 // 判断[l,r)是否回文
39 bool is_palindrome(int l, int r) {
40     if (l == r) return true;
41     if ((r - l) & 1) {
42         return d1[l + (r - l) / 2] >= (r - l + 1) / 2;
43     } else {
44         return d2[l + (r - l) / 2] >= (r - l) / 2;
45     }
46 }

```

3.5 字典树

```

1  #include <cstring>
2  #include <vector>
3  using namespace std;
4
5  const int wordnum = 100;
6  const int wordlen = 4000;
7  const int maxnode = wordnum * wordlen + 10;
8  const int sigma_size = 26;
9
10 // 字母表为全体小写字母的Trie
11 struct Trie {
12     int ch[maxnode][sigma_size];
13     int val[maxnode];
14     int sz; // 结点总数
15     void clear() {
16         sz = 1;
17         memset(ch[0], 0, sizeof(ch[0]));
18     }
19     int idx(char c) { return c - 'a'; } // 字符c的编号
20
21 // 插入字符串s, 附加信息为v。注意v必须非0, 因为0代表“本结点不是单词结点”
22 void insert(const char *s, int v) {
23     int u = 0, n = strlen(s);
24     for (int i = 0; i < n; i++) {
25         int c = idx(s[i]);
26         if (!ch[u][c]) { // 结点不存在
27             memset(ch[sz], 0, sizeof(ch[sz]));
28             val[sz] = 0; // 中间结点的附加信息为0
29             ch[u][c] = sz++; // 新建结点
30         }
31         u = ch[u][c]; // 往下走
32     }
33     val[u] = v; // 字符串的最后一个字符的附加信息为v
34 }

```

35 |};

3.6 ac 自动机

```

1  #include <cstring>
2  #include <queue>
3
4  using namespace std;
5
6  const int SIGMA_SIZE = 128;
7  const int WORD_SIZE = 55;
8  const int WORD_NUM = 1005;
9  const int MAXNODE = WORD_SIZE * WORD_NUM + 10;
10
11 struct AhoCorasickAutomata {
12     int ch[MAXNODE][SIGMA_SIZE];
13     int f[MAXNODE];    // fail函数
14     int val[MAXNODE];  // 每个字符串的结尾结点都有一个非0的val
15     int last[MAXNODE]; // 输出链表的下一个结点
16     bool vis[MAXNODE];
17     int cnt[WORD_NUM];
18     int sz;
19
20     void init() {
21         sz = 1;
22         memset(ch[0], 0, sizeof(ch[0]));
23         memset(vis, 0, sizeof(vis));
24         memset(cnt, 0, sizeof(cnt));
25     }
26
27     // 字符c的编号
28     int idx(char c) const { return c; }
29
30     // 插入字符串。v必须非0
31     void insert(char* s, int v) {
32         int u = 0, n = strlen(s);
33         for (int i = 0; i < n; i++) {
34             int c = idx(s[i]);
35             if (!ch[u][c]) {
36                 memset(ch[sz], 0, sizeof(ch[sz]));
37                 val[sz] = 0;
38                 ch[u][c] = sz++;
39             }
40             u = ch[u][c];
41         }
42         val[u] = v;
43     }
44
45     // 递归打印以结点j结尾的所有字符串
46     void print(int j) {
47         int ret = 0;
48         if (j) {
49             cnt[val[j]]++;
50             print(last[j]);
51         }
52     }
53
54     // 在T中找模板
55     void find(const char* T) {
56         int n = strlen(T);
57         int j = 0; // 当前结点编号, 初始为根结点
58         for (int i = 0; i < n; i++) { // 文本串当前指针
59             int c = idx(T[i]);
60             while (j && !ch[j][c]) j = f[j]; // 顺着细边走, 直到可以匹配
61             j = ch[j][c];
62             if (val[j])
63                 print(j);
64             else if (last[j])
65                 print(last[j]); // 找到了!
66         }
67     }
68
69     // 计算fail函数
70     void getFail() {
71         queue<int> q;
72         f[0] = 0;
73         // 初始化队列
74         for (int c = 0; c < SIGMA_SIZE; c++) {
75             int u = ch[0][c];

```

```

76         if (u) {
77             f[u] = 0;
78             q.push(u);
79             last[u] = 0;
80         }
81     }
82     // 按BFS顺序计算fail
83     while (!q.empty()) {
84         int r = q.front();
85         q.pop();
86         for (int c = 0; c < SIGMA_SIZE; c++) {
87             int u = ch[r][c];
88             if (!u) continue;
89             q.push(u);
90             int v = f[r];
91             while (v && !ch[v][c]) v = f[v];
92             f[u] = ch[v][c];
93             last[u] = val[f[u]] ? f[u] : last[f[u]];
94         }
95     }
96 }
97 }; // namespace AhoCorasickAutomata
98
99 AhoCorasickAutomata ac;

```

3.7 后缀数组

全字符串找循环节 要有长度为 i 的循环节，就要满足以下条件：

$$\begin{aligned}
 rank[0] - rank[i] &= 1 \\
 height[rank[0]] &= len - i \\
 len \% i &= 0
 \end{aligned}$$

找字符串循环节最大重复次数 枚举长度 len ，枚举起点 j ，求 $lcp(j, j + len)$

$$\begin{aligned}
 ans &= lcp / len + 1 \\
 k &= j - (len - ans \% len) \\
 if (k > 0 \&\& lcp(k, k + len) \geq len) \{ ans ++; \}
 \end{aligned}$$

求 ans 最大值

复杂度

- * 后缀数组倍增法（时间 $O(n \log n)$ ，空间 $O(4n)$ ）
- * 后缀数组 dc3 法（时间 $O(n)$ ，空间 $O(10n)$ ）
- * 后缀数组快排（适用于最大值很大的情况，除了 sa 数组，其他暂未测试）

3.7.1 后缀数组-倍增法

```

1 #include <algorithm>
2 #include <cstdio>
3 #include <cstring>
4 using namespace std;
5
6 namespace SuffixArray {
7     using std::printf;
8
9     const int maxn = 1e7 + 5; // max(字符串长度, 最大字符值加1)
10
11     int s[maxn];           // 原始字符数组 (最后一个字符应必须是0, 而前面的字符必须非0)
12     int sa[maxn];          // 后缀数组
13     int rank[maxn];         // 名次数组. rank[0]一定是n-1, 即最后一个字符
14     int height[maxn];       // height数组
15     int t[maxn], t2[maxn], c[maxn]; // 辅助数组
16     int n;                 // 字符个数 (包括最后一个0字符)
17
18     void init() { n = 0; }
19
20     // m为最大字符值加1. 调用之前需设置好s和n
21     void build_sa(int m) {
22         int i, *x = t, *y = t2;
23         for (i = 0; i < m; i++) c[i] = 0;
24         for (i = 0; i < n; i++) c[x[i]] = s[i]++;

```

```

25     for (i = 1; i < m; i++) c[i] += c[i - 1];
26     for (i = n - 1; i >= 0; i--) sa[—c[x[i]]] = i;
27     for (int k = 1; k <= n; k <= 1) {
28         int p = 0;
29         for (i = n - k; i < n; i++) y[p++] = i;
30         for (i = 0; i < n; i++)
31             if (sa[i] >= k) y[p++] = sa[i] - k;
32         for (i = 0; i < m; i++) c[i] = 0;
33         for (i = 0; i < n; i++) c[x[y[i]]]++;
34         for (i = 0; i < m; i++) c[i] += c[i - 1];
35         for (i = n - 1; i >= 0; i--) sa[—c[x[y[i]]]] = y[i];
36         swap(x, y);
37         p = 1;
38         x[sa[0]] = 0;
39         for (i = 1; i < n; i++) x[sa[i]] = y[sa[i - 1]] == y[sa[i]] && y[sa[i - 1] + k] == y[sa[i] + k]
40             ? p - 1 : p++;
41         if (p >= n) break;
42         m = p;
43     }
44
45     void build_height() {
46         int i, k = 0;
47         for (i = 0; i < n; i++) rank[sa[i]] = i;
48         for (i = 0; i < n; i++) {
49             if (k) k--;
50             int j = sa[rank[i] - 1];
51             while (s[i + k] == s[j + k]) k++;
52             height[rank[i]] = k;
53         }
54     }
55 } // namespace SuffixArray
56
57 // 编号辅助
58 namespace SuffixArray {
59     int idx[maxn];
60
61     // 给字符串加上一个字符, 属于字符串i
62     void add(int ch, int i) {
63         idx[n] = i;
64         s[n++] = ch;
65     }
66 } // namespace SuffixArray
67
68 // LCP 模板
69 namespace SuffixArray {
70     using std::min;
71     int dp[maxn][20];
72     void initRMQ(int n) {
73         for (int i = 1; i <= n; i++) dp[i][0] = height[i];
74         for (int j = 1; (1 << j) <= n; j++)
75             for (int i = 1; i + (1 << j) - 1 <= n; i++) dp[i][j] = min(dp[i][j - 1], dp[i + (1 << (j - 1))
76                 ][j - 1]);
77         return;
78     }
79     void initRMQ() { initRMQ(n - 1); }
80
81     int lcp(int a, int b) {
82         int ra = rank[a], rb = rank[b];
83         if (ra > rb) swap(ra, rb);
84         int k = 0;
85         while ((1 << (k + 1)) <= rb - ra) k++;
86         return min(dp[ra + 1][k], dp[rb - (1 << k) + 1][k]);
87     }
88 } // namespace SuffixArray
89
90 // 调试信息
91 namespace SuffixArray {
92     using std::printf;
93     void debug() {
94         printf("n:%d\n", n);
95
96         printf("%8s", "");
97         for (int i = 0; i < n; i++) {
98             printf("%4d", i);
99         }
100         printf("\n");
101
102         printf("%8s", "s:");

```



```

103     for (int i = 0; i < n; i++) {
104         printf("%4d", s[i]);
105     }
106     printf("\n");
107
108     printf("%8s", "sa:");
109     for (int i = 0; i < n; i++) {
110         printf("%4d", sa[i]);
111     }
112     printf("\n");
113
114     printf("%8s", "rank:");
115     for (int i = 0; i < n; i++) {
116         printf("%4d", rank[i]);
117     }
118     printf("\n");
119
120     printf("%8s", "height:");
121     for (int i = 0; i < n; i++) {
122         printf("%4d", height[i]);
123     }
124     printf("\n");
125 }
126 } // namespace SuffixArray

```

3.7.2 后缀数组-dc3

```

1  #include <algorithm>
2
3  using namespace std;
4
5  /*
6   注意:
7   1.maxn开n的十倍大小;
8   2.dc3(r,sa,n+1,Max+1);r为待后缀处理的数组,sa为存储排名位置的数组,n+1和Max+1都和倍增一样
9   3.calheight(r,sa,n);和倍增一样
10  */
11  // DC3 算法
12  namespace SuffixArray {
13  #define F(x) ((x) / 3 + ((x) % 3 == 1 ? 0 : tb))
14  #define G(x) ((x) < tb ? (x)*3 + 1 : ((x)-tb) * 3 + 2)
15
16      const int maxn = 1e7 + 5;
17
18      int wa[maxn], wb[maxn], wv[maxn], ws[maxn];
19      int s[maxn], sa[maxn];
20      int rank[maxn], height[maxn];
21      int n;
22
23      void init() { n = 0; }
24
25      int c0(int *r, int a, int b) { return r[a] == r[b] && r[a + 1] == r[b + 1] && r[a + 2] == r[b + 2]; }
26
27      int c12(int k, int *r, int a, int b) {
28          if (k == 2)
29              return r[a] < r[b] || (r[a] == r[b] && c12(1, r, a + 1, b + 1));
30          else
31              return r[a] < r[b] || (r[a] == r[b] && wv[a + 1] < wv[b + 1]);
32      }
33
34      void sort(int *r, int *a, int *b, int n, int m) {
35          int i;
36          for (i = 0; i < n; i++) wv[i] = r[a[i]];
37          for (i = 0; i < m; i++) ws[i] = 0;
38          for (i = 0; i < n; i++) ws[wv[i]]++;
39          for (i = 1; i < m; i++) ws[i] += ws[i - 1];
40          for (i = n - 1; i >= 0; i--) b[ws[wv[i]]] = a[i];
41          return;
42      }
43
44      void dc3(int *r, int *sa, int n, int m) {
45          int i, j, *rn = r + n, *san = sa + n, ta = 0, tb = (n + 1) / 3, tbc = 0, p;
46          r[n] = r[n + 1] = 0;
47          for (i = 0; i < n; i++)
48              if (i % 3 != 0) wa[tbc++] = i;
49          sort(r + 2, wa, wb, tbc, m);
50          sort(r + 1, wb, wa, tbc, m);
51          sort(r, wa, wb, tbc, m);

```

```

52     for (p = 1, rn[F(wb[0])] = 0, i = 1; i < tbc; i++) rn[F(wb[i])] = c0(r, wb[i - 1], wb[i]) ? p - 1
53         : p++;
54     if (p < tbc)
55         dc3(rn, san, tbc, p);
56     else
57         for (i = 0; i < tbc; i++) san[rn[i]] = i;
58     for (i = 0; i < tbc; i++)
59         if (san[i] < tb) wb[ta++] = san[i] * 3;
60     if (n % 3 == 1) wb[ta++] = n - 1;
61     sort(r, wb, wa, ta, m);
62     for (i = 0; i < tbc; i++) wv[wb[i] = G(san[i])] = i;
63     for (i = 0, j = 0, p = 0; i < ta && j < tbc; p++) sa[p] = c12(wb[j] % 3, r, wa[i], wb[j]) ? wa[i]
64         : wb[j++];
65     for (; i < ta; p++) sa[p] = wa[i++];
66     for (; j < tbc; p++) sa[p] = wb[j++];
67     return;
68 }
69
70 void build_height(int n) {
71     int i, j, k = 0;
72     for (i = 1; i <= n; i++) rank[sa[i]] = i;
73     for (i = 0; i < n; height[rank[i++]] = k)
74         for (k ? k-- : 0, j = sa[rank[i] - 1]; s[i + k] == s[j + k]; k++);
75     return;
76 }
77
78 void build_height() { build_height(n - 1); }
79
80 void build_sa(int m) { dc3(s, sa, n, m); }
81 } // namespace SuffixArray

```

3.7.3 后缀数组-快排

```

1  #include <cstdio>
2  #include <algorithm>
3  #include <cstring>
4  using namespace std;
5
6  namespace SuffixArray {
7      using std::printf;
8
9      const int maxn = 1e7 + 5; // max(字符串长度, 最大字符值加1)
10
11     int s[maxn]; // 原始字符数组 (最后一个字符应必须是0, 而前面的字符必须非0)
12     int sa[maxn]; // 后缀数组
13     int t[maxn], rank[maxn], c[maxn]; // 辅助数组
14     int n; // 字符个数 (包括最后一个0字符)
15
16     void init() { n = 0; }
17
18     int k;
19     bool compare_sa(int i, int j) {
20         if (rank[i] != rank[j]) {
21             return rank[i] < rank[j];
22         } else {
23             int ri = i + k < n ? rank[i + k] : -1;
24             int rj = j + k < n ? rank[j + k] : -1;
25             return ri < rj;
26         }
27     }
28
29     void build_sa(int _) {
30         for (int i = 0; i < n; i++) {
31             sa[i] = i;
32             rank[i] = i < n ? s[i] : -1;
33         }
34         for (k = 1; k < n; k <= 1) {
35             sort(sa, sa + n, compare_sa);
36             t[sa[0]] = 0;
37             for (int i = 1; i < n; i++) {
38                 t[sa[i]] = t[sa[i - 1]] + (compare_sa(sa[i - 1], sa[i]) ? 1 : 0);
39             }
40             for (int i = 0; i < n; i++) {
41                 rank[i] = t[i];
42             }
43         }
44     }
45 }

```

```

45
46     int height[maxn]; // height数组
47     void build_height() {
48         int i, k = 0;
49         for (i = 0; i < n; i++) {
50             if (k) k--;
51             int j = sa[rank[i] - 1];
52             while (s[i + k] == s[j + k]) k++;
53             height[rank[i]] = k;
54         }
55     }
56 } // namespace SuffixArray

```

3.8 字符串哈希

```

1  #include <algorithm>
2  #include <cstdio>
3  #include <cstring>
4  using namespace std;
5
6  const int maxn = 40000 + 10; // 字符串长度
7
8  // 字符串哈希（概率算法）
9  struct StringHash {
10     const int x; // 随便取
11     unsigned long long H[maxn], xp[maxn];
12     int n;
13     StringHash() : x(123) {}
14     // n为字符串长度
15     void init(const char* s, int n) {
16         this->n = n;
17         H[n] = 0;
18         for (int i = n - 1; i >= 0; i--) H[i] = H[i + 1] * x + (s[i] - 'a');
19         xp[0] = 1;
20         for (int i = 1; i <= n; i++) xp[i] = xp[i - 1] * x;
21     }
22     // 从i开始，长度为L的字串的hash
23     unsigned long long getHash(int i, int L) const {
24         return H[i] - H[i + L] * xp[L];
25     }
26 };

```

3.9 字符串分割

3.9.1 按字符分割

```

1  #include <iostream>
2  #include <cstring>
3  #include <vector>
4  using namespace std;
5
6  // 字符串分割，分隔符为字符，可为多字符，前后不留空字符串
7  // *a,b*c,d, 按,*分割 -> {"a","b","c","d"}
8  // 注意：源字符串s将会被改变，请勿使用string.c_str()
9  // s源字符串 t传出结果 sep分隔符字符串(分隔符为每个单字符)
10 void split(char *s, vector<string> &v, const char *sep) {
11     char *p = strtok(s, sep);
12     while (p) {
13         v.push_back(string(p));
14         p = strtok(NULL, sep);
15     }
16 }

```

3.9.2 按字符串分割

```

1  #include <iostream>
2  #include <string>
3  #include <vector>
4
5  using namespace std;
6
7  // 字符串分割，分隔符为字符串，前后留空字符串
8  // cabcacac 按c分割 -> {"", "ab", "a", "a", ""}
9  // s源字符串 v传出结果 c分隔符字符串
10 void split(const string& s, vector<string>& v, const string& c) {
11     string::size_type pos1, pos2;

```

```

12 pos2 = s.find(c);
13 pos1 = 0;
14 while (string::npos != pos2) {
15     v.push_back(s.substr(pos1, pos2 - pos1));
16
17     pos1 = pos2 + c.size();
18     pos2 = s.find(c, pos1);
19 }
20 if (pos1 <= s.length()) v.push_back(s.substr(pos1));
21 // 如果要去除最后空串,用下方语句替代上一条
22 // if (pos1 != s.length()) v.push_back(s.substr(pos1));
23 }

```

3.9.3 按字符分割 (STL)

```

1 #include <iostream>
2 #include <string>
3 #include <vector>
4 using namespace std;
5
6 // 字符串分割, 分隔符为字符, 可为多字符, 前后不留空字符串
7 // **a,b*c,d, 按,*分割 -> {"a","b","c","d"}
8 // strtok 的实现
9 // s源字符串 t传出结果 sep分隔符字符串(分隔符为每个单字符)
10 void split(const string &s, vector<string> &v, const string &sep) {
11     typedef string::size_type string_size;
12     string_size i = 0;
13     while (i != s.size()) {
14         //找到字符串中首个不等于分隔符的字母;
15         int flag = 0;
16         while (i != s.size() && flag == 0) {
17             flag = 1;
18             for (string_size x = 0; x < sep.size(); ++x) {
19                 if (s[i] == sep[x]) {
20                     ++i;
21                     flag = 0;
22                     break;
23                 }
24             }
25         }
26
27         //找到又一个分隔符, 将两个分隔符之间的字符串取出;
28         flag = 0;
29         string_size j = i;
30         while (j != s.size() && flag == 0) {
31             for (string_size x = 0; x < sep.size(); ++x) {
32                 if (s[j] == sep[x]) {
33                     flag = 1;
34                     break;
35                 }
36             }
37             if (flag == 0) ++j;
38         }
39         if (i != j) {
40             v.push_back(s.substr(i, j - i));
41             i = j;
42         }
43     }
44 }

```

4 动态规划

4.1 物品无限的背包问题

模型

- 有 n 种物品, 每种物品有无限多个。第 i 种物品的体积为 V_i , 重量为 W_i 。选一些物品装到一个容量为 C 的背包, 使得背包内物品在总体积不超过 C 的前提下重量尽量大
- $d[i]$: 体积为 i 时选择物品的最大价值
- 转移方程: $d[i] = \max\{d[i - V[j]] + W[j]\}$
- 结果: $d[C]$???

4.2 0-1 背包

1 模型

- 有 n 种物品，每种只有一个。第 i 种物品的体积为 V_i ，重量为 W_i 。选一些物品装到一个容量为 C 的背包，使得背包内物品在总体积不超过 C 的前提下重量尽量大
- $d[i][j]$: 把第 $i, i+1, i+2, \dots, n$ 个物品装入容量为 j 的背包中的最大重量之和
- 转移方程: $d[i][j] = \max\{d[i+1][j], d[i+1][j - V[i]] + W[i]\}$
- 结果: $d[0][C]$

2 LG-P1060

- 有 n 件物品，第 i 件物品价格为 V_i ，权重为 W_i 。选一些物品，求物品总价格不超过 N 的条件下最大价格与权重乘积之和
- $d[i][j]$: 把第 $i, i+1, i+2, \dots, n$ 个物品装入容量为 j 的背包中的最大价值与权重乘积之和
- 转移方程: $d[i][j] = \max\{d[i+1][j], d[i+1][j - V[i]] + V[i] \times W[i]\}$
- 结果: $d[0][N]$

3 LG-P1164

- 有 n 种物品，每种只有一个。第 i 种价格为 V_i 元，求花光 C 元的方案都多少种
- $d[i][j]$: 第 $i, i+1, i+2, \dots, n$ 个物品花光 j 元的方案数
- 转移方程: $d[i][j] = d[i+1][j] + d[i+1][j - V[i]]$
- 结果: $d[0][C]$

4 LG-P1064

- 有 n 种物品，每种只有一个，分为主件与附件两类。主件可以有 $0-2$ 个附件，附件不再拥有附件。若选择附件，则其主件必选。第 i 种物品的价格为 V_i ，权重为 W_i 。选一些物品，求物品总价格不超过 N 的条件下最大价格与权重乘积之和
- 方法: 只遍历主件。对每个主件来说，最多有 5 种决策（不再是标准 0-1 背包问题的 2 种），即不选该物品，只选该物品，选择该物品和附件 1，选择该物品和附件 2，选择该物品和附件 1,2。其他均与 0-1 背包相同

4.3 线性结构上的动态规划

4.3.1 LIS (最长上升子序列)

1 模型

- $d[i]$: 以 i 为终点的最长上升子序列的长度
- 转移方程: $d[i] = \max\{0, d[j] | j < i, A_j < A_i\} + 1$
- 结果: $\max\{d[i]\}$

2 最少不上升子序列的个数 = 最长上升子序列的长度

3 $O(n \log n)$ 求 LIS

- e.g. luogu-P1020
- $d[]$: 当前最长上升子序列
- 若 $A_i > d[len]$, $d[len] = A_i$
- 否则，在 $d[]$ 中找到第一个大于等于 A_i 的数，用 A_i 替换
- 结果: len

4 LG-P1091: 变形

- $inc[i]$: 以 i 为终点的最长上升子序列的长度
- $dwn[i]$: 以 i 为起点的最长下降子序列的长度
- 答案: $\max\{inc[i] + dwn[i] - 1\}$

4.3.2 LCS (最长公共子序列)

$$d[i][j] = \begin{cases} d[i-1][j-1] + 1, A_i = A_j \\ \max\{d[i-1][j], d[i][j-1]\}, A_i \neq A_j \end{cases}$$

5 数据结构

5.1 线段树递归

```

1 #define maxn 100007 //元素总个数
2 #define ls l, m, rt << 1
3 #define rs m + 1, r, rt << 1 | 1
4 int Sum[maxn << 2], Add[maxn << 2]; // Sum求和, Add为懒惰标记
5 int A[maxn], n; //存原数组数据下标[1,n]
6 // PushUp函数更新节点信息, 这里是求和
7 void PushUp(int rt) { Sum[rt] = Sum[rt << 1] + Sum[rt << 1 | 1]; }
8 // Build函数建树
9 void Build(int l, int r, int rt) { // l,r表示当前节点区间, rt表示当前节点编号
10     if (l == r) { //若到达叶节点
11         Sum[rt] = A[l]; //储存数组值
12         return;
13     }
14     int m = (l + r) >> 1;
15     //左右递归
16     Build(l, m, rt << 1);
17     Build(m + 1, r, rt << 1 | 1);
18     //更新信息
19     PushUp(rt);
20 }
21 //点修改
22
23 void Update(int L, int C, int l, int r,
24             int rt) { // l,r表示当前节点区间, rt表示当前节点编号
25     if (l == r) { //到叶节点, 修改
26         Sum[rt] += C;
27         return;
28     }
29     int m = (l + r) >> 1;
30     //根据条件判断往左子树调用还是往右
31     if (L <= m)
32         Update(L, C, l, m, rt << 1);
33     else
34         Update(L, C, m + 1, r, rt << 1 | 1);
35     PushUp(rt); //子节点更新了, 所以本节点也需要更新信息
36 }
37 //区间修改
38 void Update1(
39     int L, int R, int C, int l, int r,
40     int rt) { // L,R表示操作区间, l,r表示当前节点区间, rt表示当前节点编号
41     if (L <= l && r <= R) { //如果本区间完全在操作区间[L,R]以内
42         Sum[rt] += C * (r - l + 1); //更新新数字和, 向上保持正确
43         Add[rt] +=
44             C; //增加Add标记, 表示本区间的Sum正确, 子区间的Sum仍需要根据Add的值来调整
45         return;
46     }
47     int m = (l + r) >> 1;
48     PushDown(rt, m - l + 1, r - m); //下推标记
49     //这里判断左右子树跟[L,R]有无交集, 有交集才递归
50     if (L <= m) Update1(L, R, C, l, m, rt << 1);
51     if (R > m) Update1(L, R, C, m + 1, r, rt << 1 | 1);
52     PushUp(rt); //更新本节点信息
53 }
54 //下推标记
55
56 void PushDown(int rt, int ln, int rn) {
57     // ln,rn为左子树, 右子树的数字数量。
58     if (Add[rt]) {
59         //下推标记
60         Add[rt << 1] += Add[rt];
61         Add[rt << 1 | 1] += Add[rt];
62         //修改子节点的Sum使之与对应的Add相对应
63         Sum[rt << 1] += Add[rt] * ln;
64         Sum[rt << 1 | 1] += Add[rt] * rn;
65         //清除本节点标记
66         Add[rt] = 0;
67     }
68 }
69 //查询

```

```

70
71 int Query(int L, int R, int l, int r,
72           int rt) { // L,R表示操作区间, l,r表示当前节点区间, rt表示当前节点编号
73     if (L <= l && r <= R) {
74         //在区间内, 直接返回
75         return Sum[rt];
76     }
77     int m = (l + r) >> 1;
78     //下推标记, 否则Sum可能不正确
79     PushDown(rt, m - l + 1, r - m);
80
81     //累计答案
82     int ANS = 0;
83     if (L <= m) ANS += Query(L, R, l, m, rt << 1);
84     if (R > m) ANS += Query(L, R, m + 1, r, rt << 1 | 1);
85     return ANS;
86 }
87 int main() {
88     int L, C, R;
89     //建树
90     Build(1, n, 1);
91     //点修改
92     Update(L, C, 1, n, 1);
93     //区间修改
94     Update1(L, R, C, 1, n, 1);
95     //区间查询
96     int ANS = Query(L, R, 1, n, 1);
97     return 0;
98 }

```

5.2 线段树非递归

```

1
2 #define maxn 100007
3 int A[maxn], n, N; //原数组,n为原数组元素个数,N为扩充元素个数
4 int Sum[maxn << 2]; //区间和
5 int Add[maxn << 2]; //懒惰标记
6 void Build(int n) {
7     //计算N的值
8     N = 1;
9     while (N < n + 2) N <<= 1;
10    //更新叶节点
11    for (int i = 1; i <= n; ++i) Sum[N + i] = A[i]; //原数组下标+N=存储下标
12    //更新非叶节点
13    for (int i = N - 1; i > 0; --i) {
14        //更新所有非叶节点的统计信息
15        Sum[i] = Sum[i << 1] + Sum[i << 1 | 1];
16        //清空所有非叶节点的Add标记
17        Add[i] = 0;
18    }
19 }
20
21 //点修改
22 void Update(int L, int C) {
23     for (int s = N + L; s; s >>= 1) {
24         Sum[s] += C;
25     }
26 }
27 //区间查询
28 int Query(int L, int R) {
29     int ANS = 0;
30     for (int s = N + L - 1, t = N + R + 1; s ^ t ^ 1; s >>= 1, t >>= 1) {
31         if (~s & 1) ANS += Sum[s ^ 1];
32         if (t & 1) ANS += Sum[t ^ 1];
33     }
34     return ANS;
35 }
36 //区间修改
37 void Update1(int L, int R, int C) {
38     int s, t, Ln = 0, Rn = 0, x = 1;
39     // Ln: s一路走来已经包含了几个数
40     // Rn: t一路走来已经包含了几个数
41     // x: 本层每个节点包含几个数
42     for (s = N + L - 1, t = N + R + 1; s ^ t ^ 1; s >>= 1, t >>= 1, x <<= 1) {
43         //更新Sum
44         Sum[s] += C * Ln;
45         Sum[t] += C * Rn;
46         //处理Add
47         if (~s & 1) Add[s ^ 1] += C, Sum[s ^ 1] += C * x, Ln += x;

```

```

48     if (t & 1) Add[t ^ 1] += C, Sum[t ^ 1] += C * x, Rn += x;
49 }
50 //更新上层Sum
51 for (; s; s >>= 1, t >>= 1) {
52     Sum[s] += C * Ln;
53     Sum[t] += C * Rn;
54 }
55 }
56 //区间查询
57 int Query(int L, int R) {
58     int s, t, Ln = 0, Rn = 0, x = 1;
59     int ANS = 0;
60     for (s = N + L - 1, t = N + R + 1; s ^ t ^ 1; s >>= 1, t >>= 1, x <<= 1) {
61         //根据标记更新
62         if (Add[s]) ANS += Add[s] * Ln;
63         if (Add[t]) ANS += Add[t] * Rn;
64         //常规求和
65         if (~s & 1) ANS += Sum[s ^ 1], Ln += x;
66         if (t & 1) ANS += Sum[t ^ 1], Rn += x;
67     }
68     //处理上层标记
69     for (; s; s >>= 1, t >>= 1) {
70         ANS += Add[s] * Ln;
71         ANS += Add[t] * Rn;
72     }
73     return ANS;
74 }

```

5.3 主席树

```

1  #include <algorithm>
2  #include <cstdio>
3  #include <iostream>
4  #include <vector>
5
6  using namespace std;
7  const int MAXN = 1e5 + 10;
8  struct Tree {
9      int l, r, sum;
10 } T[MAXN * 40];
11 vector<int> v;
12 int cnt, root[MAXN], a[MAXN];
13
14 void Init() {
15     cnt = 0;
16     T[cnt].l = 0;
17     T[cnt].r = 0;
18     T[cnt].sum = 0;
19     root[cnt] = 0;
20     v.clear();
21 }
22
23 int getId(int x) { return lower_bound(v.begin(), v.end(), x) - v.begin() + 1; }
24
25 void Update(int l, int r, int &x, int y, int pos) {
26     T[++cnt] = T[y], T[cnt].sum++, x = cnt;
27     if (l == r) return;
28     int mid = (l + r) >> 1;
29     if (mid >= pos)
30         Update(l, mid, T[x].l, T[y].l, pos);
31     else
32         Update(mid + 1, r, T[x].r, T[y].r, pos);
33 }
34
35 int Query(int l, int r, int x, int y, int k) {
36     if (l == r) return l;
37     int mid = (l + r) >> 1;
38     int sum = T[T[y].l].sum - T[T[x].l].sum;
39     if (sum >= k)
40         return Query(l, mid, T[x].l, T[y].l, k);
41     else
42         return Query(mid + 1, r, T[x].r, T[y].r, k - sum);
43 }
44
45 int main() {
46     Init();
47     int n, m;
48     scanf("%d%d", &n, &m);
49     for (int i = 1; i <= n; i++) {

```



```

50     scanf("%d", &a[i]);
51     v.push_back(a[i]);
52 }
53 sort(v.begin(), v.end());
54 v.erase(unique(v.begin(), v.end()), v.end());
55 for (int i = 1; i <= n; i++) Update(1, n, root[i], root[i - 1], getid(a[i]));
56 int l, r, k;
57 for (int i = 1; i <= m; i++) {
58     scanf("%d%d%d", &l, &r, &k);
59     printf("%d\n", v[Query(1, n, root[l - 1], root[r], k) - 1]);
60 }
61 return 0;
62 }

```

6 常用 STL

6.1 pair 的 hash

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 struct pair_hash {
6     template <class T1, class T2>
7     std::size_t operator()(const std::pair<T1, T2> &p) const {
8         auto h1 = std::hash<T1>{}(p.first);
9         auto h2 = std::hash<T2>{}(p.second);
10        return h1 ^ h2;
11    }
12 };
13
14 unordered_set<pair<int, int>, pair_hash> s; // 用法

```