PTA 循环结构 H

rogeryoungh

2021 年 04 月 13 日

目录

PTA 循环结构 HARD 部分，PDF。

1 加点优化

掌握一些常见的算法很有必要。

算法学习比较考验数学功底，可能理解起来会比较吃力。

1.1 7-2 含 8 的数字的个数

暴力是显然的，为 $O(n \log n)$。

假如输入的是 `1 123456987`，在我的电脑上大概需要运行 10 秒钟，用 PTA 在线测试则会超时。

这里通过另一道题 P1980 计数问题，来引出 **数位 DP** 。

> 在 $[1, n]$ 的所有整数中，数码 x 出现了几次?

注意 888 在原题中算出现 1 次，在这题算出现 3 次。

利用小学奥数中的一些计数技巧，不用计算机我们也可以求出答案。

假设 $n = 728, x = 7$。

个位：73 个，7，17，$\cdots$，727。

十位：70 个，$70 \sim 79$，$170 \sim 179$，$\cdots$，$670 \sim 679$。

百位：29 个，$700 \sim 728$。

故总计为 $73 + 70 + 29 = 172$。

据此，我们可以写出 $x \neq 0$ 的解。不难看出是 $O(\log n)$ 的。

```
int addup(int n, int x) {
    int ans = 0, m = 1;
    while(m ≤ n){
        int a=n/(m*10), b=n/m%10, c=n%m;
        if(b > x)
            ans += a * m + m;
        else if(b == x)
            ans += a * m + c + 1;
        else
            ans += a * m;
        m *= 10;
    }
    return ans;
}
```

回到原题，我们可以用同样的思想。记 $f_x(n) = f(x)$ 是 1 到 n 中数码 x 出现的次数，所求即 $f_8(b) - f_8(a-1)$。

第一步是预处理 $f(10^t)$，后面将把任意的 n 分解为一系列 10^t 的组合。

- 首先 $f(10) = 1$。对于 $10 \sim 19$ 和 $20 \sim 29$，因为首位数不贡献，可把这些区间看作相似的。两位数中只有 $80 \sim 89$ 是特殊的。
- 对于 $f(10^2)$，除了特殊的 $80 \sim 89$，剩下都可看作相似的 9 个 $f(10)$，共记 $9f(10) + 10 = 19$。
- 同样的对于 $f(10^3)$，除了特殊的 800 至 899，剩下都可看作相似的 9 个 $f(10^2)$，共记 $9f(10^2) + 10^2 = 272$。

记 $a_i = f(10^i)$，可以抽象出递推关系，提前预处理

$$a_{i+1} = 9a_i + 10^i, a_1 = 1$$

假如要计算的 $x = 123456987$，从高到低位逐位分析，记中间变量为 ans，简记为 s。

- 首先，最高位的 1 表示有 1 个 $f(10^8)$，此时 $s = f(100000000)$。
- 次高位的 2 表示有 2 个 $f(10^7)$，此时 $s = f(120000000)$。
- 接下来的 3 表示有 3 个 $f(10^6)$，此时 $s = f(123000000)$。
- 对于当前位不为 8 的都做类似操作，直到 9，表示有 8 个 $f(10^2)$ 和特殊的 100 个 $800 \sim 899$，此时 $s = f(123456900)$。
- 接下来是 8 ，表示有 8 个 $f(10)$，以及特殊且被截断的 $80 \sim 87$，此时 $s = f(123456987)$。
- 因为截断已经计数了，计算完成，跳出循环。

综上，我们可以写出代码，它的复杂度是 $O(\log n)$。

```
int aa[20] = {0};

int addup(int n, int x) {
    int m = 1, i = 0;
    while(m ⩽ n) {
        i++; m *= 10;
    }
    int ans = 0;
    while(m > 0) {
        int t = n / m;
        n = n % m;
        if (t < x) {
            ans += t * aa[i];
        } else if (t > x) {
            ans += (t - 1) * aa[i] + m;
        } else {
            ans += t * aa[i] + n + 1;
```

```
        }
        i--; m /= 10;
    }
    return ans;
}

int main() {
    int a,b;
    scanf("%d %d", &a, &b);
    int m = 1;
    for(int i = 1; i ⩽ 9; i++) {
        aa[i] = aa[i-1] * 9 + m;
        m *= 10;
    }
    int ans = addup(b,8) - addup(a-1, 8);
    printf("%d", ans);
    return 0;
}
```

1.2 7-5 输出 2 到 n 之间的全部素数

暴力试除法是能过的，为 $O(n\sqrt{n})$。

接下来介绍筛法，把 $1 \sim n$ 的所有数字对应到一个数组，在数组上做标记来表示这个数不是质数，即筛去。剩下的自然都是质数。

首先是埃氏筛，它的复杂度是 $O(n \log \log n)$。我们从 2 开始，将 2 的倍数筛去；其次是 3，将 3 的倍数筛去；4 已经被筛去了，接下来是 5，把 5 的倍数筛去……一直筛到 n。

放个维基的 GIF

实现如下。

```
int notp[10000001];
int prime[2000001], cnt;
void init(int n) {
    for (int i = 2; i ⩽ n; i++) {
        if (!notn[i]) {
```

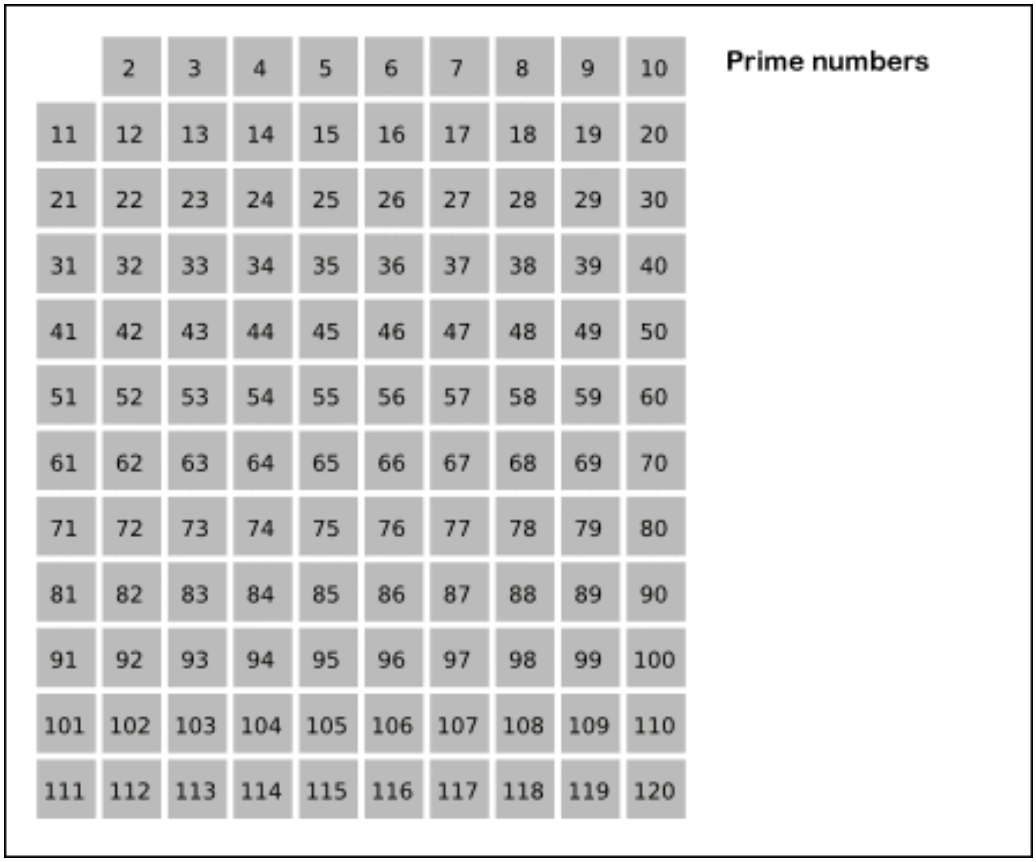

图 1: Eratosthenes

```
            prime[++cnt] = i;
            int tn = n / i;
            for (int j = i; i <= tn; i++)
                notn[i * j] = 1;
        }
    }
}
```

线性筛的复杂度更为优秀，是 $O(n)$，因为每个数只会被筛一次。我们更常用线性筛，线性筛例题 P3383。

```
int notp[10000001];
int prime[2000001], cnt;
void init(int n) {
    for (int i = 2; i <= n; i++) {
        if (!notp[i])
            prime[++cnt] = i;
        int t = n / i;
        for (int j = 1; i <= cnt; i++) {
            if(prime[j] > t)
                break;
            notp[i * prime[j]] = 1;
            if (i%prime[j] == 0)
                break;
        }
    }
}
```

更加深入的，质数的分布 $\pi(x) \sim \dfrac{x}{\ln x}(x \to \infty)$ 是亚线性的，于是还存在着一些亚线性的筛法，感兴趣的可以自行了解。

1.3 7-7 中国余数定理（1）

暴力是显然的。我在这里介绍一下数学做法。即求

$$\begin{cases} n \equiv b & (\bmod 3) \\ n \equiv c & (\bmod 5) \\ n \equiv a & (\bmod 7) \end{cases}$$

题目给了提示，“中国剩余定理 CRT”。

对于 3，我们可以构造 $m_1 = 5 \times 7 = 35$，其对 5 和 7 的模显然都是 0，而对 3 的模是 2。

对于线性同余方程 $ax \equiv 1 \pmod b$，则称 x 为 a 在模 b 意义下的逆元，记作 a^{-1}。可以由欧几里得算法推知 $\gcd(a, b) = 1$ 是逆元存在的充要条件。求逆元例题 L2605，L110。

因此 35 在模 3 意义下的逆元是 2，于是有

$$b \times 35 \times 2 \equiv b \pmod 3$$

同理，对于 5 可以得到 21 和其逆元 1，于是有

$$c \times 21 \times 1 \equiv c \pmod 5$$

最后，对于 7 可以得到 15 和其逆元 1，于是有

$$a \times 15 \times 1 \equiv a \pmod 7$$

注意到我们每次计算时都已经排除其他变量的干扰，因此将全部结果累加也同样具有这些性质

$$n \equiv 70b + 21c + 15a \pmod{105}$$

从而写出程序

```c
int main() {
    int a,b,c;
    while (scanf("%d%d%d", &a, &b, &c) != EOF) {
        int ans = b*70 + c*21 + a*15;
        ans = (ans-1+105)%105+1;
        printf("%d\n", ans);
    }
    return 0;
```

```
}
```

一般的，若 n_i 中任意两个互质，求方程组

$$\begin{cases} x \equiv a_1 & (\bmod n_1) \\ x \equiv a_2 & (\bmod n_2) \\ \vdots & \\ x \equiv a_k & (\bmod n_k) \end{cases}$$

的解。对于第 i 项，我们可以构造数

$$m_i = \frac{1}{n_i} \prod_{j=1}^{k} n_j$$

它对除了 n_i 以外的数 n 模都是 0，再求 m_i 对 n_i 逆元

$$a_i m_i m_i^{-1} \equiv a_i \pmod{n_i}$$

对所有的解累加求和得到全部的解

$$x \equiv \sum_{i=1}^{k} a_i m_i m_i^{-1} \pmod{\prod_{j=1}^{k} n_j}$$

中国剩余定理（CRT）例题 P1495。

```
ll china_crt(int* aa, int* nn, int n) {
    ll prod = 1;
    ll rst = 0;
    for (int i = 1; i ⩽ n; i++)
        prod *= nn[i];
    for (int i = 1; i ⩽ n; i++) {
        ll m = prod / nn[i];
        rst += aa[i] * m * inv(m, nn[i]);
        rst %= prod;
    }
    return rst;
}
```

对于 n_i 不互质的情况，可以使用 exgcd 对问题进行转换。EXCRT 例题 P4777。

1.4 7-26 判断素数

试除法不再详述。

常见的素性测试方法还有 Miller-Rabbin 方法。

费马小定理：如果 p 是素数，a 是小于 p 的正整数，那么 $a^{p-1} \bmod p = 1$。

尽管费马小定理的逆定理并不成立，但是它几乎都是对的，特别是再与二次探测定理结合：如果 p 是素数，x 是小于 p 的正整数，且 $x^2 \bmod p = 1$。

据此可以加强命题：

尽可能提取 $p-1$ 中 2 的因子，使得 $p-1 = d \cdot 2^r$。如果 p 是一个素数：

1. 要么 $a^d \bmod p = 1$
2. 要么存在 $0 \leqslant i < r$ 使得 $a^{d \cdot 2^i} \bmod p = p-1$。

选取 $a = 2, 7, 61$ 为基分别测试，那么在 2^{32} 范围内逆命题都是对的。更多特殊基的情况见 Link。

于是可以写出来 $O(\log n)$ 的算法

```
ll power(ll a, ll b, ll p) {
    ll rst = 1 % p;
    for (; b > 0; b >>= 1) {
        a = a * a % p;
        if (b & 1)
            rst = a * rst % p;
    }
    return rst;
}

int miller_rabbin(int n) {
    if (n < 3)
        return (n == 2);
    int a = n - 1, b = 0;
    while (1 - (a & 1)) {
        a >>= 1;
        ++b;
    }
    int prime[10] = {2, 7, 61};
    for (int i = 0; i ⩽ 2; i++) {
```

```
        int x = prime[i];
        if (n == x)
            return 1;
        ll v = power(x, a, n);
        if (v == 1 || v == n - 1)
            continue;
        int j;
        for (j = 0; j < b; j++) {
            v = v * v % n;
            if (v == n - 1)
                break;
        }
        if (j >= b)
            return 0;
    }
    return 1;
}
```