算法分析与设计
第五章

1020383827@qq.com

doubleh

数论基础

- 1.如何判断一个数是否为质数？
- 2.如何求出一个范围内的所有质数？（0~N）
- 3.如何对一个数做质因子分解？

数论基础

- 1.如何判断一个数是否为质数？
- O(sqrt(n)) （如何是合数，一定存在小于等于sqrt(n))的因子）

```
77 bool is_prime(int n)
78 {
79     for (int i = 2; i * i <= n; i++)
80         if (n % i == 0)
81             return false;
82     return true;
83 }
```

数论基础

- 2.如何求出一个范围内的所有质数？
- 普通版素数筛法：O(nlogn)（枚举所有倍数）

```
85 const int maxn = 100;
86 bool composite[maxn];
87 void gen_prime()
88 {
89     for (int i = 2; i < maxn; i++)
90     {
91         if (composite[i]) continue;
92         for (int j = 2 * i; j < maxn; j += i)
93             composite[j] = true;
94     }
95 }
```

数论基础

- 2.如何求出一个范围内的所有质数？
- 快速版素数筛法：O(nlog(logn))（枚举不小于当前质数倍数的数）

```
 96 void fast_gen_prime()
 97 {
 98     for (int i = 2; i < maxn; i++)
 99     {
100         if (composite[i]) continue;
101         for (int j = i * i; j < maxn; j += i)
102             composite[j] = true;
103     }
104 }
```

数论基础

- 2.如何求出一个范围内的所有质数？
- 快速版素数筛法：O(nlog(logn))

```
 96 void fast_gen_prime()
 97 {
 98     for (int i = 2; i < maxn; i++)
 99     {
100         if (composite[i]) continue;
101         for (int j = i * i; j < maxn; j += i)
102             composite[j] = true;
103     }
104 }
```

数论基础

- 2.如何求出一个范围内的所有质数？
- 线性筛法：O(n)（只枚举不大于当前最小质数的质数）

```
106 void linear_gen_prime()
107 {
108     tot = 0;
109     for (int i = 2; i < maxn; i++)
110     {
111         if (!composite[i]) prime[tot++] = i;
112         for (int j = 0; j < tot; j++)
113         {
114             if (prime[j] * i >= maxn) break;
115             composite[prime[j] * i] = true;
116             if (i % prime[j] == 0) break;
117         }
118     }
119 }
```

数论基础

- 3.如何对一个数做质因子分解？
- O(sqrt(n))

```
121 vector<long long> factor(long long n)
122 {
123     vector<long long> ret;
124     for (long long i = 2; i * i <= n; i++)
125     {
126         while (n % i == 0)
127         {
128             ret.push_back(i);
129             n /= i;
130         }
131     }
132     if (n > 1) ret.push_back(n);
133     return ret;
134 }
```

题目

- 1009 Mersenne Composite N，综合题
- 1020 Big Integer，高精度
- 1259 Sum of Consecutive Primes，简单题
- 1240 Faulty Odometer，简单题
- 1231 The Embarrassed Cryptography，高精度
- 1203 The Cubic End
- 1119 Factstone Benchmark，技巧题
- 1500 Prime Gap，简单题
- Base： 1 题 8 分
- Up：多一题，加一分，不超过10分

Soj 1009 Mersenne Composite N

- 分解所有形如2^p-1的素数，其中p为素数（即梅森素数的定义），且p<=k

Soj 1009 Mersenne Composite N

- 解法：那么对于本题我们只需要计算出小于63的所有素数，并对所有2^p – 1进行因子分解就可以了

Soj 1009 Mersenne Composite N

- 然而，对于p=61太慢了。。。
- 但是我们可以发现2^61 – 1是质数所以可以不用枚举到它。
- 梅森素数还有其他有趣的性质，大家可以在网上查阅。

soj 1020 Big Integer

- 题意：输入n个整数bi（1 <= i <= n)，以及一个大整数x，输出一个n元组(x mod b1,x mod b2,…,x mod bn)
- 约束：n <= 100，1 < bi <= 1000 (1 <= i <= n) 大整数x的位数 m <= 400并且非负

soj 1020 Big Integer

- mod 操作（对应C++中的%操作符）的性质：
- (a + b) % n == (a % n + b % n) % n
- (a * b) % n == ((a % n) * (b % n)) % n

- 所以我们要存储的值都是在模意义下的

soj 1020 Big Integer

- 大整数处理的常用办法
- 例如：1234=(((1 * 10 + 2) * 10 + 3) * 10) + 4
- 再利用前面mod操作的性质我们可以知道：
- 1%7 = 1
- 12%7 = (((1 % 7) * 10) % 7 + 2) % 7 = 5
- 123%7 = (((12 % 7) * 10) % 7 + 3) % 7 = 4
- 1234%7 = (((123 % 7) * 10) % 7 + 4) % 7 = 2

soj 1020 Big Integer

- 代码: O(n*m)

```
13 int mod(int x[], int m, int val)
14 {
15     int ret = 0;
16     for (int i = 0; i < m; i++)
17         ret = (ret * 10 + x[i]) % val;
18     return ret;
19 }
```

Soj 1259 Sum of Consecutive Primes

- 题意：
- 给出一个正整数，求出它有多少种方法可以表示成连续的素数的和。
- 例如53 = 5 + 7 + 11 + 13 + 17 = 53，共有两种方法。

- 限制：
- 数字大小2<=n<=10000

Soj 1259 Sum of Consecutive Primes

- 解法：
- 第一步，显然先把1~10000的所有素数找出来
- 第二步，就通过枚举连续素数的起点，来看是否有一段以它为开头的连续素数和为输入的数

Soj 1259 Sum of Consecutive Primes

- 解法：
- 第一步可以用前面学到的素数筛法，有个结论是1~n范围内的素数个数大概为O(n/logn)
- 那么接下来的枚举完连续素数的开头，即使是在暴力枚举连续素数的结尾，总的复杂度也就O((n/logn)^2)，还算可以接受
- 当然，我们同样可以通过前缀和，用二分的方法找到这个连续的素数和可能的结尾，这样做的复杂度是O(n/logn * (log(n/logn)))

Soj 1259 Sum of Consecutive Primes

- 代码：第二种方法的代码

```
const int maxn = 10001;
bool flag[maxn];
int prime[maxn], tot;
long long sum[maxn];

void pre()
{
    for (int i = 2; i < maxn; i++)
    {
        if (flag[i]) continue;
        prime[++tot] = i;
        for (int j = 2 * i; j < maxn; j += i) flag[j] = true;
    }
    for (int i = 1; i <= tot; i++)
        sum[i] = sum[i - 1] + prime[i];
}
```

Soj 1259 Sum of Consecutive Primes

- 代码：第二种方法的代码

```
int countAns(int n)
{
    int ret = 0;
    for (int i = 1; i <= tot; i++)
    {
        int j = lower_bound(sum + 1, sum + tot + 1, n + sum[i - 1]) - sum;
        if (j == tot || sum[j] - sum[i - 1] > n) continue;
        ret++;
    }
    return ret;
}
```

Soj 1240 Faulty Odometer

- 题意：
- 有个损坏的里程表，不能显示数字4，会从数字3直接跳到数字5
- 给出里程表的读数，求出实际里程。

- 限制：
- 里程表读数

Soj 1240 Faulty Odometer

- 分析：
- 在这个坏掉的里程表上，相当于只能显示“012356789”，这其实不难发现是一种修改过的9进制数
- 于是我们只需要将里程表读数转换成真正的9进制数之后，再将它转换成10进制数就可以了

Soj 1240 Faulty Odometer

- 代码：

```
int realNumber(int n)
{
    int ret = 0;
    for (int nines = 1; n; n /= 10, nines *= 9)
    {
        int v = n % 10;
        if (v > 4) v--;
        ret += v * nines;
    }
    return ret;
}
```

Soj 1231 The Embarrassed Cryptography

- 题意：
- 给出两个正整数K和L，问K是否存在小于L的质因数，有的话则找出最小的质因数。

- 限制：
- $4 <= K <= 10^{100}$ ， $2 <= L <= 10^{6}$

Soj 1231 The Embarrassed Cryptography

- 解法：
- 第一步用素数筛法求出1~1000000内的所有素数
- 第二步就枚举每个素数p，判断L能否整除p，这是一个高精度取模一个低精度的问题

Soj 1231 The Embarrassed Cryptography

- 代码：
- 同样利用到了取模操作的性质

```
int findNumber(char *L, int K)
{
    int n = strlen(L);
    for (int i = 1; i <= tot; i++)
    {
        int p = prime[i], ret = 0;
        for (int j = n - 1; j >= 0; j--)
            ret = (ret * 10 + s[j] - '0');
        if (ret == 0)
            return p;
    }
    return -1;
}
```

Soj 1231 The Embarrassed Cryptography

- 小技巧：
- 许多时候，可能这种数组中一个元素表示一个位上的数字的高精度表示法在时间和空间上会显得浪费
- 可以用到“压位”这一技巧，用int数组来表示一个高精度数，并且每一个元素表示多位数（比如4位数）
- 这里还用到个小技巧a / b - 1） / b

```
int zipNumber(char L[], int a[])
{
    int n = strlen(L);
    reverse(L, L + n);
    int m = (n + 3) / 4; // ceil(n / 4)
    for (int i = 0; i < m; i++)
    {
        a[i] = 0;
        for (int j = 3; j >= 0; j--) if (i * 4 + j < n)
            a[j] = 10 * a[j] + s[i * 4 + j] - '0';
    }
    return m;
}
```

Soj 1231 The Embarrassed Cryptography

- 代码：
- 输出高精度数（带压位的）

```
void printNumber(int a[], int n)
{
    printf("%d", a[n - 1]);
    for (int i = n - 2; i >= 0; i--)
        printf("%04d", a[i]);
    puts("");
}
```

Soj 1203 The Cubic End

- 题意：
- 题目给出了一个有趣的现象：如果一个数字串，以1，3，7，9结尾，则会有一个数，它的三次方以这个数字串结尾，且长度不会超过这个数字串。
- 现在给出一个数字串，找到一个数的三次方以这个数字串结尾。
- 限制：
- 数字串长度1<=n<=10

Soj 1203 The Cubic End

- 分析：
- 注意到一个结论：一个数x做三次方操作得到y，那么只有x的个位会影响y的个位，而y的十位也只受x的个位和十位影响…
- 而且0^3 = 0, 1^3 = 1, 2^3 = 8, 3^3 = 27, 4^3 = 64, 5^3=125
- 6^3 = 216, 7^3 =343 8^3 = 512, 9^3 = 729，三次方之后出现在最低位的数字各不相同
- 于是我们可以从低位开始一位一位的决定x的每一位

Soj 1203 The Cubic End

- 解法：
- 从低到高枚举每一位是哪个数字，再检查是否满足条件
- 假如现在枚举到第i位，枚举的数字现在为b[i]，接着用一个两重循环，判断第i位确定的情况下，做三次方运算后第i位是否与数字串的第i位相同(两重循环)。如果相同，那么说明第i位，必定为b[i]

Soj 1203 The Cubic End

- 代码：
- 时间复杂度O(n^3)
- k保存着进位

```
void theCubicEnd(int a[], int n, int b[])
{
    for (int i = 0, k = 0, tmp; i < n; i++)
    {
        for (b[i] = 0; b[i] < 10; b[i]++)
        {
            tmp = k;
            for (int x = 0; x <= i; x++)
                for (int y = 0; x + y <= i; y++)
                {
                    int z = i - x - y;
                    tmp += b[x] * b[y] * b[z];
                }
            if (tmp % 10 == a[i])
                break;
        }
        k = tmp / 10;
    }
}
```

Soj 1119 Factstone Benchmark

- 题意：
- 1960年发行了4位计算机，从此以后每过10年，计算机的位数变成两倍。输入某一个年份，求出在这个年份的最大的整数n使得n!能被一个字表示。
- 限制：
- 年份1960<=n<=2160，且n%10 == 0

Soj 1119 Factstone Benchmark

- 解法：
- 由于位长最多为2^22，能够表示的数范围很大，所以我们考虑使用log来缩小数值范围
- 如果n!能够被位长为bit_len的字表示，那么应该有
- n!<2^bit_len
- 也就是，
- $\log_2(n!) < bit\_len$
- 那么，其实我们只需要从小到大枚举n，再做判断就可以了

Soj 1119 Factstone Benchmark

- 代码：

```
int maxN(int year)
{
    int n = 1, bit_len = 1 << ((year - 1960) / 10 + 2);
    for (double tmp = 0; ; n++)
    {
        tmp += log(n);
        if (log(n) / log(2) >= bit_len)
            break;
    }
    return n - 1;
}
```

Soj 1500 Prime Gap

- 题意：
- 给出一个正整数k，找到与之相邻的两个素数，并求出两个素数之差。如果不存在两个相邻的素数则输出0。
- 限制：
- 1<=k<=1299709

Soj 1500 Prime Gap

- 分析：
- 有一个结论，素数的分布相对密集，在10^9以内，两个相邻素数距离不超过400
- 所以可以直接向前和向后枚举素数

Soj 1500 Prime Gap

- 代码：

```
bool isPrime(int x)
{
    for (int i = 2; i * i <= x; i++)
        if (x % i == 0)
            return false;
    return true;
}

int primeGap(int k)
{
    int p1 = k, p2 = k;
    while (p1 && !isPrime(p1)) p1--;
    while (!isPrime(p2)) p2++;
    return p1 == 0? 0: p2 - p1;
}
```