DOI:10.16749/j.cnki.jecjtu.2014.04.001

文章编号:1005-0523(2014)04-0054-05

亲和数的数值算法和数量估计

徐明毅

(武汉大学水利水电学院,湖北 武汉 430072)

摘要:总结了搜寻亲和数的分解算法和递推算法,计算出1 000亿内的亲和数3 261对,根据数值结果给出了在一定范围内亲和数的数量估计式,得出10^{18}内的亲和数约为百万对。

关键词:数论; 素因数; 亲和数

中图分类号:O157.1 **文献标志码**:A

一个正整数z的所有因数之和,包括1和该数本身,记为$\sigma(z)$。亲和数为不包含本身的其它所有因数之和等于另一个数的一对数,即对于数p,q,如果$\sigma(p)-p=q$,$\sigma(q)-q=p$,或者$\sigma(p)=p+q=\sigma(q)$,则p,q为一对亲和数。如果$p=q$,则称为完全数[1]。最小的一对亲和数为220和284,是由古希腊的毕达哥拉斯学派提出的[2],后来直到1636年,法国数学家费马给出第二对亲和数17 296和18 416,而数学家欧拉曾找出59对新的亲和数。在计算机出现后,人们加快了亲和数的寻找。文献[3]计算出90亿以下的亲和数1 360对,文献[4]和文献[5]又找到了更大一些的亲和数。另外,也有学者提出了判别一些特定的数不构成亲和数对[6],或寻找一些特别的亲和数对[7-8]的理论公式,但离完全解决亲和数问题尚有距离。在总结亲和数的基本定理基础上,采用数值方法计算了1 000亿内的亲和数,并根据数值计算结果,给出了一定范围内亲和数的数量估计方法。

1 亲和数的基本定理

对任一个正整数z,因数分解后可以表达为以下的形式

$$z=a^m b^n\cdots c^l \tag{1}$$

其中:a,b,c为素数;m,n,l为大于或等于1的正整数。

1.1 素数幂的全因数和

对于素数a,因数只有1和它本身,因此全部因子和为$\sigma(a)=1+a$,因此素数不可能构成亲和数对。如果某数分解为m个a的乘积,其中a为素数,可以通过组合得到所有的因数。任选出其中的一个,因数为a,任选出其中的两个,因数为a^2,以此类推,因此,包括因数1和该数本身a^m,所有的因数之和为

$$\sigma(a^m)=1+a+a^2+\cdots+a^m=\frac{a^{m+1}-1}{a-1} \tag{2}$$

该式退化形式对于素数同样成立,此时退化为m=1,$\sigma(a)=1+a$。特殊地,当$a=2$时,$\sigma(2^m)=2^{m+1}-1=2\cdot 2^m-1$,对小的素数可通过查表方法加速计算。

该表达式在计算机求解时,有m次乘法和1次除法,如果先完成乘法,就是求出一个大数再除以一个数,大数可能超界,限制了使用范围。如果直接用迭代法计算,则没有此顾虑。迭代格式为

收稿日期:2014-05-30

作者简介:徐明毅(1973—),男,副教授,主要研究方向为水工结构数值模拟。

$$\sigma(a^m)=a^m+a^{m-1}+\cdots+1=a\sigma(a^{m-1})+1 \tag{3}$$

考虑到不需计算m=0的情况,于是可用代码表示为:int s=a+1; while(--m) s=a*s+1; 这样对于素数,不需要进入循环增加一次乘法。

1.2　多因数的全因数和

先考虑素数分解后,只有两个互相独立的素数: $z=a\cdot b$,此时的因子有1, a , b , $a\cdot b$,全部因数和为

$$\sigma(z)=\sigma(ab)=1+a+b+a\cdot b=(1+a)\cdot(1+b)=\sigma(a)\sigma(b) \tag{4}$$

其实,只要是独立的两个因子,都可以表示为以上的形式。设 $z=x\cdot y$, x 和 y 为互相独立的两个数,不管是素数还是合数,即它们的因数没有重合的部分。于是,考虑组合的情况,只有 x 部分的组合为全因数和减去1,为 $\sigma(x)-1$;只有 y 部分的组合为 $\sigma(y)-1$,考虑两者的相互组合为 $[\sigma(x)-1][\sigma(y)-1]$,于是总的全因数和为

$$\begin{aligned}&\sigma(z)=\sigma(x\cdot y)=1+[\sigma(x)-1]+[\sigma(y)-1]+[\sigma(x)-1][\sigma(y)-1]=\\&1+\sigma(x)+\sigma(y)-2+\sigma(x)\sigma(y)-\sigma(x)-\sigma(y)+1=\sigma(x)\sigma(y)\end{aligned} \tag{5}$$

因此,可以逐次分解直到求最后的素数幂的全因数和。例如3个素数的乘积: $z=a\cdot b\cdot c$,用组合方法可知所有因子为:1, a , b , c , $a\cdot b$, $b\cdot c$, $a\cdot c$, $a\cdot b\cdot c$,用逐步分解的方法也同样得到全因数和

$$\sigma(a\cdot b\cdot c)=\sigma(a\cdot b)\sigma(c)=\sigma(a)\sigma(b)\sigma(c)=(1+a)(1+b)(1+c) \tag{6}$$

这样,分解素因子后,求全因数和比较简单,比直接组合的方式要简便很多。

2　判断亲和数的分解算法

由以上亲和数的基本定理,可得到判断亲和数的主要步骤为

1) 分解某数的素因子,并统计出每个素因子的个数,即得到每个素因子的幂次。

2) 通过公式求出该数的全因数和,再减去本身,就得到可能的亲和数对的另一个数。

3) 对另一个数同样进行计算,如果重新得到原来的数,则构成亲和数对,否则重新搜索。

其中较为困难的部分为分解素因数,考虑采用简单的方法。先准备素数表,如果搜索范围到n,则素数表的范围只需到 $\sqrt{n}$ 即可。然后从小到大依次测试素因数,如果测试出一个素因子,就记录下来,然后原数除以该因子,剩余的部分再继续判断素因子。在计算时,对小的素因子只要能够整除,就一直提取到没有该素因子为止,这样剩余部分就不用从素数表的开头进行判断,而是直接往后判断是否有更大的素因子,提高计算效率。

以下用代码表示素因子的提取过程,假设素数表用数组reserve[]表示,某数的独立的素因子用数组childs[]保存,对应的素因子的阶次用数组rank[]表示,代码如下:

```
int j = 0;                          // 质数表的当前判断位置
int nfactors = 0;          // 独立的素因子个数
int prime = reserve[0];
int limit = (int) sqrt((double) number) + 1;   // 检测范围的上限
while (prime < limit) {                        // 只需要检测到 √n
      if (number % prime == 0) {
            childs[nfactors] = prime;          // 记录该素因子
            rank[nfactors] = 1;                // 该素因子的阶次至少为1
            number /= prime;
            while (number % prime == 0) {   // 检测重复的素因子
                  rank[nfactors] += 1;   // 增加该素因子的阶次
                  number /= prime;
                  }
```

```
            ++nfactors;                                  // 相异的素因子个数增加
        limit = (int) sqrt((double) number) + 1;         // 减小检测上限
    }
    prime = reserve[++j];          //在素数表中取出下一个更大的素数
}
```

分解算法中,主要的计算量为素因子分解。在从某数得到可能的另一个亲和数时,如果更小,则显然已搜索过,为避免重复计算和记录,就直接跳过,减少了计算量。

现在主要估算分解过程所需的计算量。由以上素因子分解算法,分解正数数 x 的素因子最多需对该范围内的 $\rho\sqrt{x}$ 个素数依次进行求余测试,因此对该范围内所有正整数的素因子分解的测试总数为

$$C(x)=\rho\left(1+\sqrt{2}+\sqrt{3}+\cdots+\sqrt{x}\right)\approx\rho\left(\frac{2}{3}x\sqrt{x}+\frac{1}{2}\sqrt{x}-\frac{5}{24}\right)\approx\frac{2\rho}{3}x\sqrt{x} \tag{7}$$

式中:$C(x)$ 为总的求余次数;ρ 为该范围内的平均素数密度。

3 判断亲和数的递推算法

在小范围搜索时,上述的分解算法并不是最快的。因为要用除法来分解出素因子,也要用乘法来求出全因数和。其实,可以从因数的原始定义出发,可以将乘除法计算变换为加法计算。因为一个数的所有倍数对应的数都有该数为因子,因此将该因子计入因数和,逐步累加多个因子就可得到所有因数和。算法就转换为找出具有某因数的所有数的位置,然后不断累加因数就可以了。

开辟一个连续的数组来表示某数的真因数和,即全因数和减去某数本身。首先1是所有数的因数,故所有数都计入该因数,初始化为1。然后从2开始的每一个可能的真因数,在它的倍数位置加上该数,而不是它倍数的位置就不用累加该数,那么在循环结束之后,就得到该位置对应数的真因数和,其中为1的就是素数,因为没有除1以外的小于它的任一因数。然后再对数组只需要一次遍历就可以轻松找到该范围内所有的亲和数了。假设数组为sum[ ],长度为len,初始的位置为ini,列出求真因数和的代码如下:

```
for (i = 0; i < len; i++) sum[i] = 1; //1是所有数的真因数,所以全部置1
int maxfac = (ini + len) / 2;  //最大的真因数不超过它的一半
for (i = 2; i < maxfac; i++) {        //从2开始直到所有可能的真因数
    int m = ini / i;
    if (m <= 1) j = i + i - ini;    //在数组区段内,跳过本身所在位置
    else  j = i - (ini - m*i);      //在数组区段前,计算在该区段的起始位置
    while (j < len) {
        sum[j] += i;      //将所有i的倍数的位置上加i
        j += i;                     //下一个倍数位置
    }
}
```

然后扫描一遍数组,只要某数的真因数和在该区段内就可判断了,代码如下:

```
    for (i = 0; i < len; i++) {           //扫描,O(N)
    int num1 = ini + i;                   //实际的某数
    int num2 = sum[i];                    //该数的真因数和
    int pos = num2 - ini;  //反查询的位置
    if(pos<0 || pos >=len) continue;                  // 越界不考虑
    if (num1< num2 && sum[pos] == num1)      //去重,满足亲和
    printf("%d,%d\n", num1, num2);
}
```

可以看到,因为问题的特殊性,方便和巧妙地利用了下标作为伴随数组,用数组空间来节约计算时间。同时将回溯的思想转换成递推的思想,分解算法的乘除操作大部分转化为累加操作,因此加快了计算速度。

递推算法中的主要计算量为求真因数和的累加过程,如对于因数1,累加次数为x,对于因数2,累加次数为$x/2$,以此类推,总的累加次数为:

$$D(x)=x(1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{x/2})\approx x[\ln(x/2+1)+r]\approx x(\ln x-0.116) \tag{8}$$

式中:$D(x)$ 为正整数 x 范围内总的求和次数,r 为调和级数的欧拉常数项,约为0.577 2。

与前面比较,分解算法的主要计算量为 $O(x\sqrt{x})$ 次求余,而递推算法的主要计算量为 $O(x\ln x)$ 次求和,故效率大大提高。遗憾的是,由于不能将很大范围内的所有正整数的真因数和同时保存在内存中,只能保存有限区段内的真因数和,故该算法并不能在大范围搜索时达到理想状态。

实际计算中,如果范围超过划定的区段容量,则可将得到的真因数和保留到一个缓冲数组中,这样超出区段的数同样可以查询判断是否是亲和数。缓冲区越大,能够增加的搜索范围越大,一般能增加十倍范围以上。但范围更大时,暂时不能判断是否亲和的数将溢出缓冲区,这时就必须结合分解算法,将缓冲区清空一部分,再进行下一个区段的搜索。

判断亲和数的递推算法在小范围搜索时十分有效,但在搜索空间扩大后,效率下降,此时反而是采用分解算法更有效了,两者效率相当时的搜索范围与使用的内存空间大小和具体的算法有关,必须通过实际的数值实验才能确定。

4　大整数的定义方法

为表示较大的数据范围,需要采用64位二进制整数。在VC++中,int是4字节整数,而__int64(前面为2个下横线)是8字节整数。如果是较新的版本,符合C99标准的话,也可以用long long来表示8字节整数。为便于使用,可以用条件定义语句:typedef __int64 xint;于是用重定义类型xint来表示64位二进制整数。输入采用scanf("%I64d", &n)来读入一个数,输出用printf("%I64d", n),如果是64位无符号整数,格式字符串为%I64u。如果要表示更大的整数范围,C/C++编译器不能直接支持,必须用自定义的方法来处理。

在编译为32位程序还是64位程序时,以上的数据定义格式都不用变化,只有表示地址的指针大小会自动更改。32位程序中,指针用4字节表示,而在64位程序中,指针用8字节表示。由于在64位系统中运行64位程序更快,并且可以分配更大的内存,这时只需要选择目标运行平台为x64即可。如在vs2008中,建立项目后,用快捷建"alt+f7"配置项目属性,在对话框中点击"配置管理器",在弹出对话框中"活动解决方案平台"栏下选择"<新建…>",再选择x64平台即可。但在一些精简版本的编译器中可能没有该选项,使用时要注意。

5　亲和数的分布估计

亲和数是较为稀少的,但现在还不能确定亲和数对的数量是否有限。在搜寻出的亲和数对中,均同为偶数或奇数,还没有一奇一偶的情况,但这是否是普遍规律,还未见证明[2]。使用较为快速的递推算法,已计算得到1 000亿内的所有亲和数对,数量统计如表1所示。

表1　1 000亿内亲和数的数目统计

Tab. 1　Statistics of amicable numbers within one hundred billion

搜寻范围	100万	1 000万	1亿	10亿	100亿	1 000亿
亲和数对	40	100	231	564	1 391	3 261
增长比率	\	2.50	2.31	2.44	2.47	2.34

从计算结果可以看出，亲和数的分布有较好的规律性，在搜索范围增加为原来的10倍时，亲和数对的数量为原来的2倍多，随着范围增大，亲和数越稀少，搜索也越困难。假设增长比率为a，按a=2.3计算，以1亿为计算起点，则100亿亿(10^{18})内的亲和数对估计为231×2.3^{10}=956 952；以100亿为计算起点，则估计为$1\ 391\times2.3^{8}$=1 089 305，差别不大，估计为100万对左右。在该搜索范围内平均万亿个数才能找到一对亲和数，可用“沙里淘金”来形容寻找难度。归纳得到在正整数x内亲和数的数量为：

$$n(x)=n(x_0)a^{\log\frac{x}{x_0}} \tag{9}$$

式中：$n(x)$为正整数x范围内的亲和数对的数目；x_0为已知亲和数对的正整数范围；a为增长比例，从当前成果看似乎应介于2和3之间，其是否随范围增大会趋于一个定值，还需要更多的计算支持或从理论上加以证明。

另外，采用分解算法，寻找到1万亿以上的5对亲和数：(1 000 452 085 744，1 023 608 366 096)，(1 000 539 285 525，1 015 331 690 475)，(1 000 607 505 404，1 147 934 333 956)，(1 001 352 481 250，1 117 674 392 350)，(1 001 583 011 750，1 019 368 284 250)，可见在该范围内，平均约搜索3亿个数才发现一对亲和数。

6 结语

亲和数是数学中的有趣问题，本文给出了搜寻亲和数的分解算法和递推算法，并得到1 000亿内的所有亲和数。分析计算结果得出，亲和数随范围增大愈加稀疏，相邻亲和数对之间相距越来越远，估计10^{18}内的亲和数约100万对。

参考文献：

[1] 华罗庚.数论导引[M].北京:科学出版社,1979.
[2] 徐品方.寻找亲和数的艰辛岁月[J].数学通报,1999(6):37-38.
[3] 周尚超.亲和数[J].华东交通大学学报,1998(4):67-68.
[4] 周尚超,刘二根,邓毅雄,等.15对亲和数[J].华东交通大学学报,1999,26(3):66-67.
[5] 周尚超,刘二根,邓毅雄,等.亲和数的若干性质与10对亲和数[J],华东交通大学学报,2000,27(3):68-70.
[6] 沈忠华.关于亲和数的一个结果[J].哈尔滨师范大学自然科学学报,2001(5):15-19.
[7] 邓淙.探求亲和数的一种方法[J].昭通师专学报,1984(1):10-13.
[8] 向红军,杨吉清.亲和数的一种求法[J].湖南理工学院学报:自然科学版,2009(3):14-15.

Numeric Algorithm and Magnitude Estimation of Amicable Numbers

Xu Mingyi

(School of Water Resources and Hydropower Engineering, Wuhan University, Wuhan 430072, China)

Abstract: This paper proposes the decomposition algorithm and recursive algorithm to search amicable numbers. 3 261 pairs of amicable numbers are searched out within 100 billion. The numerical estimated expression of amicable numbers in a certain range is obtained according to numeric results. There are approximately one million amicable numbers in 10^{18} range.

Key words: number theory; prime factor; amicable number