数位计数问题解法研究

清华附中 高逸涵

(gaoyihan@gmail.com)

【摘要】数位计数问题是一类比较麻烦的问题，这类问题难度往往不大，但特殊情况众多，实现较为麻烦，本文将通过对几个此类问题的例子进行分析，利用从简单到复杂的层层递进思路进行代码实现，说明一种较好的对该类问题的处理方式。

【关键字】

数位 计数问题

【正文】

在信息学竞赛中，有一类难度不大但异常麻烦的问题——数位计数问题，这类问题的主要特点是询问的答案和一段连续的数的各个数位相关，并且需要对时间效率有一定要求。由于解决这类问题往往意味着巨大的代码量，而众多的特殊情况又意味着出现错误的巨大可能性，因此很少有人愿意解决此类问题，但只要掌握好的方法，解决这类问题也并非想象中的那样困难。

我们先来看一道最为基础的数位计数问题：

例一、按位求和问题[1]

题目大意：

给定 A,B(1<=A,B<=10^{15})，求[A,B]内的所有数的 k 进制表示下各数位之和（2<=K<=10）

解法分析：

首先，如果能找到方法计算[0,B]的答案，那么同理计算[0,A-1]的答案，然后再相减即可求出需要的数值。因此，问题转化为计算[0,B]的答案。但这一问题仍然难以直接解决。

考虑最简单的情况，如果 B 是形如k^n-1的数，那么也就是说，需要求所有不超过 n

[1] 经典问题

位的数的数位之和，将所有数补前导 0 使所有数为 n 位，那么显然在[0,B]中每个数字出现的次数相等。因此答案 $ans=(B+1)n\dfrac{0+1+2+......+k-1}{k}=\dfrac{(B+1)n(k-1)}{2}$

实现一下该函数：

```
long long getsum1(int n, int k)
//n 为自由位个数，k 为进制
{
    long long B = 1;
    for (int i = 0; i < n; i++ ) B *= k;
    return B * n * (k - 1) / 2;
}
```

同时编写 check 程序：

```
long long check(int n, int k)
{
    long long ret = 0 ;
    for (int i = 1; i <= n; i++ )
    {
        int t = i;
        while ( t > 0 )
        {
            ret += t % k;
            t /= k;
        }
    }
    return ret;
}
```

（编写 check 程序的目的是检验每一个函数部分的实现是否正确，这对于我们正确的解决问题有至关重要的意义）

利用 check 程序检测我们已实现的程序，排查错误。

回归原问题，我们已经成功解决了问题的一个最简单的部分，那么接下来的任务就是利

用这一部分解决原问题，或是解决相对更为复杂一些的部分。

对于一个询问，我们可以利用以上函数处理所有数位比它少的数的数字和，所以只需计算数位和他一样多的数的数字和即可。

对于一个数，若其首位已经比询问上界小，则剩余位没有任何限制。此时如果能直接处理这一情况，则问题距离解决又会迈出一大步。

例如，在十进制下，计算[10000,54321]内的数字和，我们可以将其分解为：

[10000,19999],[20000,29999],[30000,39999],[40000,49999],[50000,54321]。

前四个区间如果可以直接解决，则只需处理最后一个区间，进一步将最后一个区间划分为：[50000,50999],[51000,51999],[52000,52999],[53000,53999],[54000,54321]。同理将最后一个区间划分下去，最后可以得到以下区间划分：

[10000,19999],[20000,29999],[30000,39999],[40000,49999],

[50000,50999],[51000,51999],[52000,52999],[53000,53999],

[54000,54099],[54100,54199],[54200,54299],

[54300,54309],[54310,54319],

[54320,54321]

为了直接处理每个区间，我们编写函数 getsum2。(事实上在有了 getsum2 函数之后，getsum1 函数便可以用 getsum2(0,n,k)代替)

```
long long getsum2(int prefixsum, int n, int k)
//prefix 为公共前缀数字和，n 为自由位长度，k 为进制
{
    long long B = getsum1( n, k);
    long long C = prefixsum;
    for (int i = 0; i < n; i++ ) C *= k;
    return B + C;
}
```

此时，可以利用递归函数完成区间划分。

```
long long getsum3(int prefixsum, long long n, int k)
//计算带有前缀的 0 到 n 的数字和之和，k 为进制，prefixsum 为前缀和
{
    //此函数为递归函数，需要注意边界条件
    if ( n < k )
    {
       long long ret = 0;
       for (int i = 0; i <= n; i++ ) ret += prefixsum + i;
       return ret;
    }
    long long t = 1, tn = n;
    int d = 0;
    while ( tn >= k )
    {
       tn /= k;
       t *= k;
       d ++;
    }
    long long ret = 0;
    for (int i = 0; i < tn; i++ )
        ret += getsum2( prefixsum + i, d, k );
    ret += getsum3( prefixsum + tn, n - t * tn, k );
    return ret;
}
```

最后调用函数 getsum3(0, n, k)计算答案。

小结：

虽然这是一道简单的题目，但从中我们也可以看出解决许多数位计数问题的通用方法。

1. 首先考虑最简单的情况，即只有数位数确定，其他所有数位都任意的情况下，在尽量短的时间复杂度内解决。

2. 考虑带有前缀的情况，对问题进行解决。

3. 将原问题拆分成若干段分别利用上述两种情况解决。

若一个问题可以拆分成若干段子问题之和，则可由以上原则处理，否则需要想办法将原问题进行转化然后处理。

另外就是关于检查的问题，由于数位计数问题的特殊性，其 check 程序往往非常容易编写。因此建议将原问题尽量多的分割成不同块。保证每一块的代码量尽量小。

对每一块单独用 check 函数进行检测，另外需要明确每个函数参数的范围，对函数间调用的部分需要特别谨慎。

例二、The Sum[1]

题目大意：

将 1~N（$1<=N<=10^{15}$）写在纸上，然后在相邻的数字间交替插入”+”和”-“，求最后的结果。例如当 N 为 12 时，答案为：+1-2+3-4+5-6+7-8+9-1+0-1+1-1+2=5

解法分析：

这是一道稍微复杂一点的数位计数问题。

根据上述原则，我们首先探查数位确定，所有数字自由的情况。

若数位数为偶数，以 6 位为例(不妨设第一个符号为+)：

+0	-0	+0	-0	+0	-0
+0	-0	+0	-0	+0	-1
+0	-0	+0	-0	+0	-2
…………………………………………………………					
+9	-9	+9	-9	+9	-9

此时，每一个数位的符号都是确定的，因此只需要分别计算每一位的所有数字和即可，因为所有位 0~9 出现机会均等，因此和必然为 0。

若数位数为奇数，此情况更加简单，以 5 位为例（不妨设第一个符号为+）：

+0	-0	+0	-0	+0
-0	+0	-0	+0	-1
+0	-0	+0	-0	+2
-0	+0	-0	+0	-3

[1] Sphere Online Judge 1433 KPSUM

……………………………………………………………

+9 -9 +9 -9 +8

-9 +9 -9 +9 -9

可以注意到，相邻两行的和必然为-1，因此整个和很容易求出。

于是我们编写函数 getsum1 和检验函数 check。（getsum1 的另一个参数 k 的由来在下页有说明）

```
long long getsum1( int n, int k )
//n 为自由位个数，k 为总位数（k>=n>=1）
{
    if ( k % 2 == 0 )
    {
        if ( n % 2 == 0 ) return 0;
        else
        {
            long long d = -45;
            for (int i = 0; i < n - 1; i++ ) d *= 10;
            return d;
        }
    }
    else
    {
        long long d = -1;
        for (int i = 0; i < n; i++ ) d *= 10;
        return d / 2;
    }}
```

```
long long check( int n )
{
    long long ret = 0;
    int t = 1, a[10];
```

```
    for (int i = 1; i <= n; i++ )
    {
        int r = 0, p = i;
        while ( p > 0 )
        {
            a[r++] = p % 10;
            p /= 10;
        }
        for (int j = r - 1; j >= 0; j-- )
        {
            ret += a[j] * t;
            t = -t;
        }
    }
    return ret;
}
```

接下来，考虑带有前缀的情况，因为前缀对符号的影响，所以需要在 getsum1 处追加总位数的参数。此时由 getsum1 处可求出自由位的数字和，因此只需再求出前缀的数字和即可。

当总位数=自由位+前缀位数为偶数时：

+1	-2	+0	-0	+0	-0
+1	-2	+0	-0	+0	-1
+1	-2	+0	-0	+0	-2

……………………………………………………

+1	-2	+9	-9	+9	-9

前缀数字和符号不变，因此只需要乘总行数即可。

总位数为奇数时：

+1	-2	+0	-0	+0
-1	+2	-0	+0	-1
+1	-2	+0	-0	+2
-1	+2	-0	+0	-3

…………………………………………………………

+1　　-2　　+9　　-9　　+8

-1　　+2　　-9　　+9　　-9

前缀两两相消，和为 0。

依照以上分析编写 getsum2。

```
long long getsum2( long long prefix, int n )
//prefix 为前缀，n 为自由位个数(n >= 1, prefix >= 1)
{
    int d = 0, t = 1;
    long long p = prefix, presum = 0;
    while ( p > 0 )
    {
        presum += (p % 10) * t;
        p /= 10;
        d ++;
        t = -t;
    }
    presum *= -t;
    for (int i = 0; i < n; i++ ) presum *= 10;
    long long ret = getsum1( n, n + d );
    if ( (d + n) % 2 == 0 ) ret += presum;
    return ret;
}
```

沿用上例的思路，再有了上述两个函数之后，我们继续将整个区间划分为若干段，分别利用上述函数求和，这里不再重复叙述，只是将函数实现展示如下。(不能沿用上例递归程序是由于上例添加前导 0 对结果不影响，而本例则不同)

```
long long getsum3( long long n )
//对原问题进行求和[1,n], n>=1
{
```

```
if ( n < 10 )
{
    long long ret = 0;
    for (int i = 1; i <= n; i++ )
    if ( i % 2 == 0 ) ret -= i; else ret += i;
    return ret;
}
long long tn = n, p = 1;
int d = 0;
while ( tn > 0 )
{
    tn /= 10;
    d ++;
}
for (int i = 1; i < d; i++ ) p *= 10;
long long prefix = 0, ret = 5;
for (int j = 1; j < d - 1; j++ )
for (int i = 1; i <= 9; i++ )
    ret -= getsum2( i, j );
tn = n;
while (d > 1)
{
    for (int i = 0; i < tn / p; i++ )
    {
        if ( prefix != 0 )
            ret -= getsum2( prefix, d - 1 );
        prefix ++;
    }
    tn %= p; p /= 10;
```

```
        d--; prefix *= 10;
    }
    int a[20], t = -1;
    for (int i = 0; i <= tn; i++ )
    {
        long long p = prefix + i;
        int r = 0;
        while ( p > 0 )
        {
            a[r++] = p % 10;
            p /= 10;
        }
        for (int j = r - 1; j >= 0; j-- )
        {
            ret += a[j] * t;
            t = -t;
        }
    }
    return ret;
}
```

小结：

通过对问题从简单到复杂的层层递进分析，逐步将程序实现，使得一个原本比较复杂的问题轻松被解决。程序编写过程中思路明确，程序模块化合理，每个模块功能明确，并且单独可以通过 check 函数进行检验。使得出错的可能性大大降低。

虽然整体代码量有所增加，但由于思考的时间减少，代码编写时间甚至还会缩短。

最后来看一道难度较大的题目。

例三、Graduated Lexicographical Ordering[1]

题目大意：

[1] Zhejiang University Online Judge Problem 2599

定义两个数的大小比较方法为首先比较各位数字之和，如果不相等则和大的数比较大，否则按字典序比较两个数的大小关系。

例如 120 小于 4，因为 120 的数字之和为 3，而 4 的数字之和为 4。

555 小于 78，因为在字典序意义下"555"<"78"

20 小于 200，因为在字典序意义下"20"<"200"

求 1~N 中第 k 小的数，以及 k 在 1~N 中的位置。

解法分析：

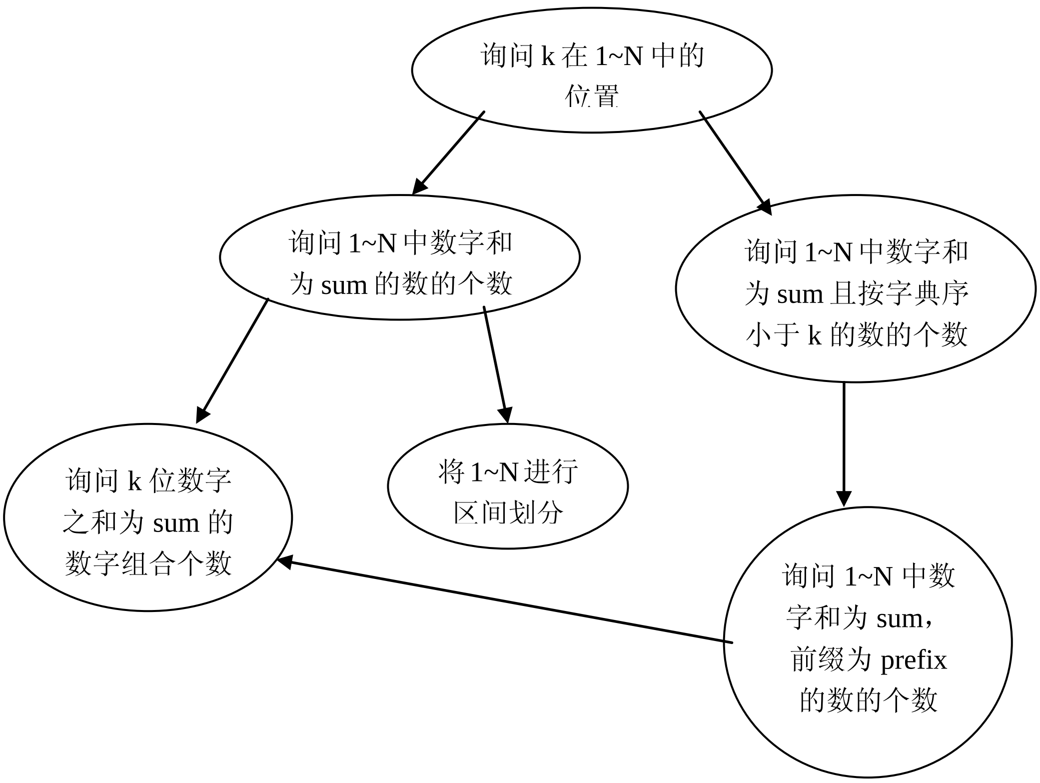
问题有两类询问，而其中第一类询问不能直接套用模式，因此我们需要将该类询问转化为我们能解决的类型。

事实上，类似第一类询问的问题，之所以难以解决，是因为它们不能通过将整个区间划分为若干段子区间，然后通过合并子区间的解来得到整个区间的解。对于这种情况，我们往往采用二分答案的方法将问题加以转化。

例如，在本题中，如果二分答案 C，询问在 1~N 中有多少个数比 C 小，如果答案为 K - 1，那么我们便可以肯定 C 就是我们要得到的原询问的答案。

至此，问题的核心在于解决第二类询问，即询问 k 在 1~N 中的位置。

为了方便叙述本题的分析过程，这里用一个图展示。



按照从简单到复杂的顺序实现函数：

1. getsum1:询问 k 位数字之和为 sum 的数字组合个数。
2. getsum2:询问 1~N 中数字和为 sum 的数的个数。
3. getsum3:询问 1~N 中数字和为 sum，前缀为 prefix 的数的个数。
4. getsum4:询问 1~N 中数字和为 sum，且字典序小于 k 的数的个数。
5. getsum5:询问 k 在 1~N 中的位置。

由于代码长度普遍较长，这里便不再列举各个函数的具体实现，仅举出各函数的实现思路：

1. getsum1:采用递归进行处理，记忆化搜索可以更好的保证复杂度。
2. getsum2:类似于前两例中将区间进行划分的方法。
3. getsum3:同样采用递归的处理方法，注意当前缀 prefix 不是 N 的前缀时，后面的数位便没有限制，可以用 getsum1 函数直接求解。
4. getsum4:通过对前缀分类用 getsum3 进行计算，然后相加。例如 k 为 2423，则分别询问前缀为 1，20，21，22，23，240，241，2420，2421，2422，注意字典序小于 k 的数必然拥有以上前缀之一。
5. getsum5:将数字和小于 k 的数字和的数的个数和数字和等于 k 的数字和但字典序较小的数的个数相加即可。

关于第一类询问，在本问题中难以直接进行二分答案，可以采用按位枚举前缀的方式确定答案。

【总结】

对于数位计数类问题，本文提出一种从简单到复杂逐步实现的方法，通过对问题的分析，将询问逐步细致简化，直到能够直接处理。在实现时由简单到复杂的顺序保证了程序模块化的清晰，便于检查和调试，多模块的函数实现方式保证了每一部分的代码量不至于太长，因而降低了出错的可能性，函数模块之间互相关联，从而降低了实现难度。最终得以快速准确地解决问题。

【感谢】

感谢寿鹤鸣同学（SHiningMoon）对我的帮助。

【参考文献】

刘汝佳，黄亮《算法艺术与信息学竞赛》 清华大学出版社

【附录】

例题二原题网址：

https://www.spoj.pl/problems/KPSUM/

例题三原题网址：

http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemCode=2599