挑战性题目DSCT101：硬币找换问题

问题描述：假设你手里有面值为1元、2元、4元、…、2*n*元的硬币，每种面值的硬币刚好有2枚。任意输入一个正整数金额，请问有多少种不同的找零方式？例如，输入6，输出3。

数学描述：对于无穷集合S={1, 2, 4, 8,…, 2*n*}，给定一个正整数*M*，用集合S中的元素进行和分解，求不同的分解数。仅允许使用S中的元素至多1次，那么对于*M*的分解数有且只有1种可行解。

* 若允许使用S中的元素至多2次，请问有多少种不同的分解数？（要求解答）
* 若允许使用S中的元素至多*k*次，请问有多少中不同的分解数？（作为思考）

思路分析[[1]](#footnote-1)与算法描述[[2]](#footnote-2)：

考虑到集合S中的元素全部为2的n次幂，故我们考虑将给定的正整数M用二进制形式进行表示。

容易考虑到，我们可以在可能的方案中选择不超过M的最大的2的n次幂形式的硬币进行组合，但也可以不这样选择。由此思路，我们将不超过M的最大的2的n次幂的硬币数学化为，并将求解算法记为函数solve(),则本题将可被分解为以下两个子问题：

* 使用硬币F：

在使用最大的2次幂面值硬币F的情况下，我们将这部分原问题就化为了求相应的面值为(M-F)的硬币的分解方案数，即我们只需求解solve(M-F)；

* 不使用硬币F：

我们将原硬币面值M化为二进制来讨论这个问题。以二进制数为例，不使用F意即不使用最高位所代表的硬币面值。

为此，我们将首位1进行退位，使得次高位的值+2，即变为。注意到S中的硬币至多只能使用两次，故次高位上的“3枚”硬币至多只有2枚可以直接使用。因此，剩下的一枚硬币必须由其他方式表出，也即必须用更低位的硬币表出。故将被化为。

类似的，中第3位上的3枚硬币也将被化为。注意到此时第4位上的值为2，即该位可以由2枚面值为的硬币表出。需要注意的是，此时可行的方案包括：

a.不使用该位的硬币；

b.仅使用1枚该位的硬币；

c.使用2枚该位的硬币。

容易发现的是，方案a是不可行的，因为若不使用任何一枚该位上的硬币，则即使将剩余低位的硬币面值全部利用2次，也一定无法凑满原来应得的面值。数学表示为。同时，方案c显然是简单且可行的；但我们也可以将其中1

枚硬币的面值利用剩余低位进行表出(因)，故方案b也是可能的。

综上，我们发现，此时我们将至少使用一枚中第4位上的硬币，而另外一枚将可能用剩余低位的硬币进行表示，也可能用本位的1枚硬币进行表出。因此，对于不使用硬币F的情况，我们将求解面值为的硬币的可行方案数，即求解solve()。

至此，我们可以利用递归方法计算solve(M)，方案与递归边界如下：



if(M==0) return 1;

else return 0;

其中表示向下取整函数，表示将中出于最高位置的 0变为1，并将该位左侧的1

全部置零后的结果。若使用自顶向下的方法实现本递归过程，则算法时间复杂度将为；但若使用逆序的递归算法来实现，则复杂度降为。

在本例中，S中的每个元素至多使用2次。若拓展至K次，则相应的在退位后可用的硬币数将增多，因此在不使用硬币F的情况下的讨论将变得更为复杂多样，但总体思路将预计可以继续使用本文的思路。

1. 算法思路不但要给出解题的算法内涵，还应该分析该算法的时间复杂度。此页背面也可以答题，但不应续页。 [↑](#footnote-ref-1)
2. 代码的写作和测试建议使用GCC、G++等通用C/C++编译器进行编译以利于跨平台的性能测试。作业纸质版本和测试的可执行文件请提交给主管助教。测试样例命令格式为：DSCT101\_2018270103012.exe 6，输出结果样例为：3。 [↑](#footnote-ref-2)