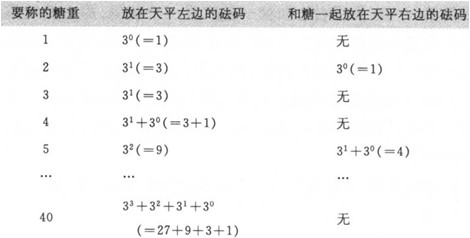
前段时间抽空看了《说谎者悖论和汉诺塔游戏》（[加拿大]马塞尔·丹尼斯著，程云琦译）一书，作者在第一个谜题“斯芬克斯之谜”中介绍了法国耶稣会诗人 Claude Gaspard Bachet de Méziriac（1581-1638）的一个经典谜题：

若天平两端可以任意放置砝码，要称量从1磅到40磅的整磅重的糖，天平所需要的砝码个数最少是多少？

换句话说，我们需要确定若干个读数互不相同的砝码，用它们在天平上组合出1至40的数。当然，前提是这是台特殊的天平——你不能开玩笑说天平上还有游码，也不能说砝码的规格是有限定的。我们假定砝码值可以是任意正整数。传统上我们可能认为1，"Times New Roman";" lang="EN-US">2，4，8，16和32磅共7个砝码，这样最高可以称出63磅重的糖。不过这个谜题有意思的地方就在于它使用的是天平quot;Times New Roman";" lang="EN-US">——两边都可以放砝码。于是我们可以这样来称出2磅重的糖：把3磅重的砝码放在天平的左边，1磅重的砝码放在天平的右边，在天平右边放上糖直到天平平衡。这样，只需要1，3，9和27磅的砝码就可以称出1磅到40磅的糖。

注意到这些砝码的重量都是3的幂：1=30，3=31，9=32，27=33。依次类推，我们可以使用1，3，9，27和81磅的砝码称出1磅到121磅的糖。

那么针对具体数量的糖，怎样放置各个砝码呢？作者给出了如下图所示的称法。



**图1砝码放置**

作者似乎想表达的是，具体的砝码放置法的获得是来源于所谓的“顿悟思维（insight thinking）”。不过写个具体的算法来实现针对具体数值的砝码放置方案并不难。

我们同样假定称量时将糖放置在天平右边。那么算法就是要实现，在输入砝码个数和需要称量的最大数值后，输出如图1所示的列表。若称量的数值超过砝码所能称量的最大数值，则提示出错（比如四个砝码不能称出41磅糖）。

我们假设要称量的数值为n。算法的基本思想是，通过判断n所在的“区间”，迭代地把砝码加到天平左右两侧。这里的“区间”指的是，将砝码的值按从小到大累积相加，形成一个新的数列，其中n所在的前开后闭区间。如，对于砝码1，3，9和27，其累积和形成的数列为1，4，13，40。如3所在的区间为(1，4]，而13所在的区间为(4，13]。

仍假定n=5。算法的运行过程如下：

1．判断得n∈(4，13]。13是1、3和9的累积和，将最大的砝码9放入左边天平。砝码9的符号为正。

2．令n = n – 9 = 5 – 9 = - 4，判断得|n|∈(1，4]。4是1和3的累积和，且由于n的符号为负，故将最大的砝码3放入右边天平。砝码3的符号为负。

3．继续令n = n – (-3) = (-4) – (-3) = - 1，判断得|n|∈(1，1]。1是1的累积和，且由于n的符号为负，故将最大的砝码1放入右边天平。砝码1的符号为负。

4．继续令n = n – (-1) = (-1) – (-1) = 0。结果为零，算法结束。

在以上的每一步中我们都得到了一个所谓的“最大的砝码”，以下这个砝码我们称之为上界砝码

算法是一个迭代的过程，用伪代码描述如下。

输入：数值n

输出：各个砝码的符号

Procedure setBalance( int n) {

While (n != 0) {

      找出n的上界砝码，令其符号与n相同。

      令n = n –上界砝码。

}

}

从伪代码来看，算法非常简洁。不过要成为可运行的程序，还需要有一些辅助性工作。以下是用Java实现该算法的参考代码。

**[java]** [view plaincopyprint?](http://blog.csdn.net/xyilu/article/details/6074622)

1. /\*
2. \* @(#)Balance.java  1.0 10/06/13
3. \*
4. \* Copyright 2010 Onion Lab. All rights reserved.
5. \*/
6. **package** problem.balance;
7. /\*\*
8. \* Solves the classic problem: balance.
9. \* @author Chungtow Lai - Zhejiang University
10. \* @version $Date: 2010-06-13
11. \*/
12. **public** **class** Balance {
13. **private** **static** **final** **int** NUM\_OF\_POISE = 4;
14. **int** number;
15. **int**[] left, right;
16. **int**[] sign;
18. /\*\*
19. \* Construction method.
20. \*/
21. **public** Balance(**int** number) {
22. **this**.number = number;
24. init();
26. **if** (number > getSumToK(NUM\_OF\_POISE)) {
27. System.out.println("Error: Out of bound!");
28. } **else** {
29. setBalance();
30. print();
31. }
32. }
33. /\*\*
34. \* Initialization
35. \*/
36. **private** **void** init() {
37. left = **new** **int**[NUM\_OF\_POISE];
38. right = **new** **int**[NUM\_OF\_POISE];
39. sign = **new** **int**[NUM\_OF\_POISE];
41. **for** (**int** i = 0; i < NUM\_OF\_POISE; i ++) {
42. sign[i] = 0;
43. }
44. }
46. /\*\*
47. \* Main algorithm
48. \*/
49. **private** **void** setBalance() {
50. **int** temp1, temp2, upper;
52. temp1 = number;
53. **while** (temp1 != 0) {
54. temp2 = Math.abs(temp1);
55. upper = getUpper(temp2);
56. sign[upper] = temp1 / temp2;
57. temp1 -= sign[upper] \* power(3, upper);
58. }
59. }
61. **private** **void** print() {
62. **int** i;
63. **int** lj = 0, rj = 0;
64. **for** (i = 0; i < NUM\_OF\_POISE; i ++) {
65. **if** (sign[i] > 0) {
66. left[lj++] = power(3, i);
67. } **else** **if** (sign[i] < 0) {
68. right[rj++] = power(3, i);
69. }
70. }
72. System.out.print(**this**.number + "/t/t");
73. **for** (i = 0; i < lj; i ++) {
74. System.out.print(left[i]);
75. **if** (i != lj-1) {
76. System.out.print(",");
77. }
78. }
79. System.out.print("/t/t");
80. **for** (i = 0; i < rj; i ++) {
81. System.out.print(right[i]);
82. **if** (i != rj-1) {
83. System.out.print(",");
84. }
85. }
86. System.out.print("/t/t");
87. **for** (i = 0; i < NUM\_OF\_POISE; i ++) {
88. **int** t = sign[i] \* power(3, i);
89. **if** (t > 0) {
90. System.out.print("+" + t);
91. } **else** **if** (t < 0) {
92. System.out.print(t);
93. }
94. }
95. System.out.println();
96. }
98. /\*\*
99. \*
100. \*/
101. **private** **int** getUpper(**int** n) {
102. **for** (**int** i = 1; i <= NUM\_OF\_POISE; i ++) {
103. **if** ((n > getSumToK(i)) && (n <= getSumToK(i+1))) {
104. **return** i;
105. }
106. }
107. **return** 0;
108. }
110. /\*\*
111. \* 返回前k个数的和
112. \*/
113. **private** **static** **int** getSumToK(**int** k) {
114. **if** (k == 1) **return** 1;
115. **else** **return** (power(3,k-1) + getSumToK(k-1));
116. }
118. /\*\*
119. \* Returns the value of the first argument raised to the power of the
120. \* second argument.
121. \* @param x the base.
122. \* @param e the exponent.
123. \* @return the value <code>x<sup>e</sup></code>.
124. \*/
125. **private** **static** **int** power(**int** x, **int** e) {
126. **if** (e == 0) **return** 1;
127. **else** **return** x \* power(x, e-1);
128. }
130. /\*\*
131. \* Main entry point for invocation.
132. \* @param args The command line arguments.
133. \*/
134. **public** **static** **void** main(String[] args) {
135. System.out.println("Number/t/tLeft/t/tRight/t/tFormula");
136. **for** (**int** i = 1; i <= 40; i ++) {
137. **new** Balance(i);
138. }
139. }
140. }