Python算法之旅（第1期）

篇首语：

人生苦短，我用Python。Python语言以其简明优雅、易于开发，代码简短，可扩展性强和丰富的第三方库等特征深受广大人们喜爱。多数人使用Python来进行软件开发和数据处理，作为一名高中信息技术教师，我想从学习算法的角度来使用Python，利用Python简明优雅的特性来描述算法问题，使程序员能直指问题的核心，而不是陷入代码细节的泥淖。

我计划搜集和整理一些有趣有用的算法问题，对其进行算法分析，并用Python语言加以实现。之所以要做这件事情，主要是为了鞭策自己坚持学习Python和算法研究，由于我接触Python语言的时间不长，在算法研究上也只粗通皮毛，写出来的文章基本上属于自娱自乐，如果有不当之处，还请大家海涵。

每期文章的最后都会有课后思考，请各位老师和同学积极参与讨论，我会在下一期公布参考答案和讨论结果。

感谢您阅读我的文章，让我们一起探讨，共同进步。

题目：石子划分问题

难度：3星 有趣：2星 有用：2星

分类：穷举，回溯，动态规划

描述：给出n堆石子，以及每堆石子数。请将它们分为两堆，使得这两堆的总石子数差最小。输入n，以及每堆石子数，输出分为两堆后的最小差值。

比如，n＝4，四堆石子分别有13，6，8，14颗，则可以分为13＋8和14＋6的两堆，它们的最小差为1。

函数：divide\_stones(a, n)

参数说明：a -- 元组，存储了每堆石子的数量。

n -- 正整数，表示总共n堆石子

返回值：返回两堆石子的最小差值。

算法分析：

分堆算法为：

（1）求得所有石子数total，以及它的一半half；

（2）在所有石子堆中作适当选择，对每种选择方案，求不超过half的己选中堆中的石子总数的最大值mx。所求即为（total－max）－max。

最直接的思路是穷举算法，即对每一堆石子都做选择和不选择两种讨论，分析每一种选择模式产生的结果，累计该选择模式中被选中的石子堆的数量，判断该结果是否满足条件，若能得到更好的解则更新最优解。

我们引入一个列表b来存储每一种选择模式，以b[i]的值表示第i堆石子是否被选中，1表示被选中，0表示未被选中。

我们可以按顺序，从b=[00…00]（均不选中），到b=[00…01]（只选中第n堆石子），到b=[00…10]（只选中第n－1堆石子），到b=[00…11]，再到b=[00…101]（选中第n－2堆和第n堆石子），直到b=[11…11]（选中所有n堆石子）。

观察这个顺序，其实是用列表b来模拟一个整数t的n位二进制数，从t=0开始，每次递增1，直到t=2^n-1的过程。

根据上述分析，我们可以写出代码：

#穷举算法：使用列表b存储当前选择模式，直接修改列表b的值来模拟整数t递增的过程

def divide\_stones\_1(a, n):

total = sum(a)

half = total // 2

b =[0] \* n #初始化所有的位都是0

max\_s, i = 0, n - 1 #先选择第n堆

while i >= 0: #遍历从[0,1<<n]的所有选择模式

s, b[i] = 0, 1 #将b[i]右侧的1都改成0，b[i]改成1，相当于二进制数递增1

for i in range(n):

s += a[i] \* b[i] #累计被选中的石子堆的数量

if s > max\_s and s <= half: #更新最优解

max\_s = s

i = n - 1 #i从最右端开始模拟整数t递增的过程

while i >= 0 and b[i] == 1: #修改列表b的值来模拟整数t递增的过程，每次递增1

b[i] = 0

i -= 1 #当b[i]==0时跳出循环，并将b[i]改成1，相当于二进制数递增1

return total - max\_s \* 2 #返回最小差值

穷举法是容易想到的算法，但是效率实在太低，我们可以使用深度优先搜索（回溯加剪枝）来实现同样的功能。

本算法的剪枝条件是已经选择的石子数量超过半数，另外我们只在第一个值为1的二进制位左侧设置1，以避免重复。代码如下：

#深搜算法：使用列表b存储当前选择模式，每层递归函数的选择模式都继承自上一层函数，每层只选择1堆石子

def divide\_stones\_2(a, n):

total = sum(a)

half = total // 2

b =[0] \* n #初始化所有的位都是0

max\_s = 0

def dfs(b, s): #参数介绍：b——列表，存储当前选择模式；s——正整数，表示已经选择的石子数量。

nonlocal max\_s

if s > max\_s: #更新最优解

max\_s = s

for i in range(n):

if b[i] == 0: #只在第一个值为1的二进制位左侧设置1，以避免重复

if s+a[i] <= half: #已经超出半数就无需再选择新的石子堆了

b[i] = 1 #将整数t的第i个二进制位设置成1

dfs(b, s+a[i])

b[i] = 0 #将整数t的第i个二进制位恢复成0

else:

break

dfs(b, 0)

return total - max\_s \* 2 #返回最小差值

进一步思考：当石子堆数量n太大时，对应的n位二进制数太大了，用穷举法和回溯法都不适合；因为我们对每个石子堆的操作都是选择或者不选择，所以本题可以当做一个0-1背包问题来解决（当然石子的总数不能太大，否则需要太大的空间），我们可以用石子总量的一半half来表示背包的容量，用每堆石子的数量来表示每个物品的质量（同时也是价值大小），最大价值就是不超过half的石子数量，这样就可以套用0-1背包问题的基本模型来解决本题。代码如下：

#动态规划：0-1背包问题的基本模型，一维列表存储记录，f[j]初始化为0

def divide\_stones\_3(a, n):

total = sum(a)

half = total // 2

f = [0] \* (total + 1)

for i in range(1, n+1):

for j in range(half, a[i-1]-1, -1): #须先求出列坐标j较大的F[j]，故让循环变量j的值从大到小递减

if f[j] < f[j-a[i-1]] + a[i-1]: #当(j < a[i-1] or f[j] >= f[j-a[i-1]] + a[i-1])时，f[j]的值不变

f[j] = f[j-a[i-1]] + a[i-1]

return total - f[half] \* 2 #返回最小差值

最后是主函数部分，我们从文件szhf.txt中读取数据，每行数据表示n堆石子的数量，一个示例如下：

50,14,5,12,4,45,45,45,42,26,26,11

36,23,7,22,46,15,15,6,4

24,9,16

总共有3行（组）数据，第1组数据给出了12堆石子的数量；第2组数据给出了9堆石子的数量；第3组数据给出了3堆石子的数量。

对于每组数据，预想的答案分别是：1，0，1。

with open('szhf.txt', 'r') as fin:

for line in fin.readlines():

print(line.strip())#依次读取每行

a = tuple(map(int, line.strip().split(",")))

print(divide\_stones\_1(a, len(a)))

print(divide\_stones\_2(a, len(a)))

print(divide\_stones\_3(a, len(a)))

课后思考：

在穷举算法中，我们用列表b来模拟一个整数t的n位二进制数，从t=0开始，每次递增1，直到t=2^n-1的过程。这个方法是很巧妙的，但是不够直观，其实我们可以使用位运算来处理整数t的二进制数，这样就无需引入列表b，可以直接操作整数t了。

请动动你聪明的大脑，写出直接操作整数t的代码吧！

另外，如果你有更 Pythonic（优雅的、地道的、整洁的）代码，或者与本文不同的算法思路和代码实现，请你一定留言或联系我，让我们一起讨论，共同进步。