4. 确定集合

Algorithm 1 分成两个子集，使它们和的差值最大

实现思路是用 dp[i][j] 记录 i 个 S 中的元素是否能 sum=j，然后找到 dp[n/2][j]=True，这个值为 sum 最大的子集，答案易得。若是要记录元素则将二维数组每个元素改为 pair<sum,vector< 元素 > 即可。

时间复杂度：$O(n^2 \cdot sum)$

输入： 集合 S，大小为 n，n 是偶数

sum 为集合 S 的总和

初始化大小为 $(n/2+1) \times (sum+1)$ 的二维数组 dp, 元素均设置为 false

$dp[0][0] \leftarrow$ True

for $k = 1$ to n **do**
  for $i = n/2$ down to 1 **do**
    for $j = sum$ down to 0 **do**
      if $j \geq S[k]$ **and** $dp[i-1][j-S[k]]$ **then**
        $dp[i][j] \leftarrow$ True
      end if
    end for
  end for
end for

找到最大的 j，有 $dp[n/2][j] =$ True

return $2 \times dp[n/2][j] - sum$

Algorithm 2 分成两个子集，使它们和的差值最小

与算法 1 基本一致, 所以不再赘述, 只说明不同的地方

与算法 1 唯一不同之处正在于, 最后一步, 改为: 从 $sum/2 \rightarrow sum$ 找到最小的 j 使 $dp[n/2][j] =$ True

5. 均衡

Algorithm 3 在两种方法间寻找均衡

只要把两种方法结合起来, 在还剩 2 个及以上罐子时用二分寻找答案或者缩小范围, 在还没找到答案且剩最后一个罐子时逐个测试.

输入: 梯子级数 n, 罐子数量 k, 且 $k \geq 1$

初始化 $range = [1, n], range$

while $k > 1$**and**$len(range) >= 2$ **do**

  在 middle(range 中间) 级阶梯上测试

  if 罐子摔碎 **then**

    range 改为 [left,middle-1]

  else

    range 改为 [middle+1,right]

  end if

end while

if len(range) == 1 **then**

  return middle

else

  for i=left(range) to right(range) **do**

    if 摔碎罐子 **then**

      return i-1

    end if

  end for

end if