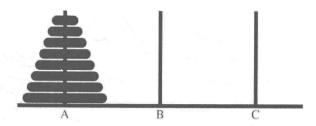
Towers of Hanoi

这个谜题是由法国数学家爱德华·卢卡斯在1880年提出的,汉诺塔问题迅速在欧洲流行。它的成功部分归功于围绕这一谜题的逐渐增长的传奇,它在法国数学家亨利·德·巴微 (Henri de Parville) 所著的《自然之谜》(La Nature)(由数学史学家鲍尔翻译)中描述如下:

在世界中心贝拿勒斯圆顶之下的一座圣庙里,放置了一个黄铜板,有三根宝石针固定在上面,每一根针都有一时高,厚度和一个蜜蜂的身体一样。创建世界时,梵天在其中一根针上面放了64个纯金圆盘,最大的金圆盘放在铜板上,其他的金圆盘随着高度升高,越来越小,最上面的一个是最小的一个。这就是梵天寺之塔。昼夜不断,牧师将金圆盘从一根宝石针移动到另一根上,根据梵天寺固定不变的法则,它要求牧师的职责是每次移动的金圆盘数不能超过一个,他必须把这个金圆盘放在一根针上,并且在这个金圆盘下面没有更小的金圆盘。当所有的金圆盘都从梵天穿好的那根针上移到另外一根针上时,世界将在一声霹雳中毁灭,而梵塔、庙宇和众生也都将同归于尽。

在这个问题的商业版本中,64个传奇的金圆盘被替换成8个木质的或塑料的圆盘,这 使得问题更易于解决(而不是说成本更低)。问题的初始状态看起来如下图所示:



一开始,8个圆盘都在塔柱A上,你的目的是将这8个圆盘从塔柱A移到塔柱B,同时要遵循以下规则:

- 每次只能移动一个圆盘。
- 不允许将一个大圆盘移动到小圆盘之上。

对于这个问题, 如果想应用递归, 首先要确定是否符合递归的想法。

- 存在简单情况
- 存在化简。

```
1  def moveTower(n: int, start: str, finish: str, tmp: str) → None:
2   if n = 1:
3     Move a single disk from start to finish.
4   else:
5     Move a tower of size n-1 from start to tmp.
6     Move a single disk from start to finish.
7     Move a tower of size n -1 from tmp to finish.
```

一个可行的伪码

```
def moveTower(n: int, start: str, finish: str, tmp: str) → None:
"""
n: 需要移动的圆盘数目。
start: 所有圆盘开始所在的塔柱名字。
finish: 所有圆盘最终应该放置的塔柱名字。
tmp: 用于临时存储圆盘的塔柱名字。
```

```
7
8
         example:
              moveTower(8, 'A', 'B', 'C')
9
10
11
         \texttt{def moveSingleDisk}(\texttt{start: str, finish: str}) \, \rightarrow \, \texttt{None:}
              print(f"{start} \rightarrow {finish}")
12
13
14
         if n = 1:
15
              moveSingleDisk(start, finish)
16
17
              moveTower(n - 1, start, tmp, finish)
18
              moveSingleDisk(start, finish)
              moveTower(n - 1, tmp, finish, start)
19
```

子集求和(包含/排除模式)

本节所涉及的问题被称为子集求和问题 (subset-sum problem) , 可定义如下:

给定一个整数集合和一个目标值,确定是否可以找到这个整数集合的一个子集,子集的和等于指定的目标 值。

例如,给定集合 $\{-2, 1, 3, 8\}$ 和目标值7,子集求和问题的答案是"是",因为子集 $\{-2, 1, 8\}$ 的元素之和等于7。

然而,如果目标值是5,答案将为"否",因为没有一种方法能选择出整数集合{-2,1,3,8}的一个子集,其元素之和为5。

```
def subsetSumExits(st: set, target: int) → bool:
    if not st:
        return target = 0
    else:
        rset = st.copy()
        element = rset.pop()
        // pop 从集合中随机取出一个元素,并且删除。
        return subsetSumExits(rset, target) or subsetSumExits(rset, target - element)
```

全排列

```
1
   def generatePermutations(S):
2
       if not S:
3
           return [[]]
4
       else:
5
           res, perms = [], generatePermutations(S[1:])
6
           for i, _ in enumerate(S):
7
               for perm in perms:
8
                   res.append(perm[:i] + [S[0]] + perm[i:])
9
           return res
```

```
1
    def generatePermutations(Str: str) \rightarrow set:
 2
        """ BUG !!!"""
 3
        result: set = set({})
 4
        if not Str:
 5
            result.add('')
 6
        else:
 7
            for i, ch in enumerate(sr):
 8
                 rest = Str[: i] + Str[i + 1:]
9
                 for s in generatePermutations(rest):
10
                     result.add(ch + s)
11
        return result
```

最难的点在于如何将原问题分解成几个简单问题的组合。