电 子 科 技 大 学 实 验 报 告

课程名称： Python语言程序设计及其应用

实验地点： 科A229

指导教师： 张勇

评 分：

完成实验学生信息：

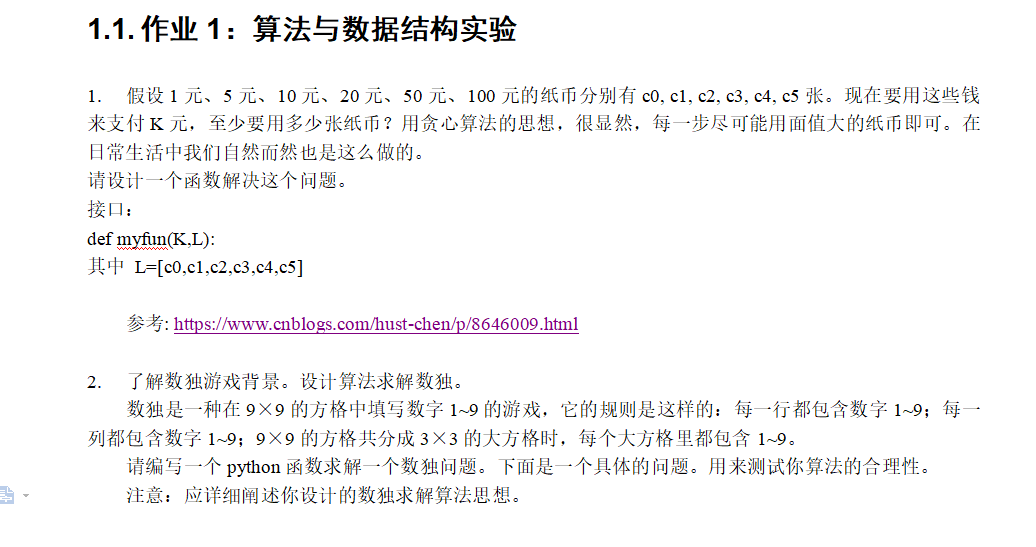
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **序** | **姓名** | **学号** | **选课**  **序号** | **贡献**  **百分比(%)** | **备注**  **（主要工作**） |
| 1 | 付文亮 | 2022080401004 |  | 30 | 1.纸币算法贡献一种递归模式的实现 |
| 2 | 傅若山 | 2022080401023 |  | 30 | 1.纸币算法贡献一种递归模式的实现  2.数独算法有一定方法上的研讨贡献 |
| 3 | 苏徐涛 | 2022080401011 |  | 40 | 1.纸币算法贡献一种循环模式的实现  2.数独算法最终实现者 |
| 4 |  |  |  |  | 1.  2. |

1. 学生人数按照任课教师要求限定；
2. 对于“评价、改进、总结和体会”都要认真填写，和其他内容是评价实验成绩的重要参考。

**实验题目名称: 算法与数据结构实验**

1. **实验内容**

(可以从word拷贝或pdf文档截图或复制文本)



1. **实验目的**

解决上诉问题

1. **实验过程**

（如果有多个问题，请对每个问题分别整理出下列内容）

1. 三人共同讨论后，分别独立给出解答

二．主要在数独的实现中， 我们有考虑更简单的算法（即进行一定程度的剪枝）。但经过试验发现完全不经过剪枝的算法依然具有良好的效率，所以没有进行深度的剪枝。

目 录

**[1.](#_Toc132356664)****[实验1：](#_Toc132356664)**[算法与数据结构实验（最少纸币问题） 5](#_Toc132356664)

[1.1. 问题分析 5](#_Toc132356665)

[1.2. 系统设计与算法设计 5](#_Toc132356666)

[1.3. 编写程序 5](#_Toc132356667)

[1.4. 运行结果 6](#_Toc132356668)

[1.5. 实验结果分析 6](#_Toc132356669)

[1.6. 优缺点及改进方向 7](#_Toc132356670)

[1.7. 心得体会与总结 8](#_Toc132356671)

**[2.](#_Toc132356672)****[实验2：数独问题](#_Toc132356672)** [8](#_Toc132356672)

[2.1. 问题分析 8](#_Toc132356673)

[2.2. 系统设计与算法设计 8](#_Toc132356674)

[2.3. 编写程序 8](#_Toc132356675)

[2.4. 运行结果 8](#_Toc132356676)

[2.5. 实验结果分析 8](#_Toc132356677)

[2.6. 优缺点及改进方向 8](#_Toc132356678)

[2.7. 心得体会与总结 8](#_Toc132356679)

**[3.](#_Toc132356680)****[问题重述](#_Toc132356680)** [8](#_Toc132356680)

**[4.](#_Toc132356681)****[对本次实验的设计提出改进意见](#_Toc132356681)** [9](#_Toc132356681)

**[5.](#_Toc132356682)****[附件](#_Toc132356682)** [9](#_Toc132356682)

[附件1.最少纸币问题的程序 9](#_Toc132356683)

[附件2.YY的程序 10](#_Toc132356684)

[附件3.ZZ程序的输出结果 10](#_Toc132356685)

# **实验1：**算法与数据结构实验（最少纸币问题）

## 问题分析

给定不同面值纸币后要求计算达到固定金额所需要的最少纸币可以采用贪心算法，即从最高面值的纸币开始，只要该面值纸币数量足够并且不超过总金额就可以一直累加，按照这一操作得到的满足总金额的纸币数量就是所需要纸币的最少数量。

## 系统设计与算法设计

模块分解:程序分为两个部分，第一部分作为输入部分，第二部分作为求解部分。

算法设计:从可选择的最高面值纸币开始累加，在该纸币数量范围内累加到不超过总金额的最大数量并记录，然后找到次高面值纸币按照这一逻辑计算，其余全部如此，直到正好等于总金额即可输出结果。如果遍历过程中出现无法满足总金额的情况时，则在此时第一个高于累加中的最小面值(即比最小面值大并且数量不为零)的位置回退，回退后重新计算，以此类推找到最终解，如果无解输出不可能。

## 编写程序

def myfun(k: int, l: list):  
    ind = 5  
    while ind >= 0:  
        if k == 0:  
            return 1  
        else:

  if l[ind] > 0:  
                if k - C[ind] >= 0:  
                    k -= C[ind]  
                    l[ind] -= 1  
                    p = myfun(k, l)  
                    if p >= 1:  
                        return 1 + p  
                    l[ind] += 1  
                    k += C[ind]  
                    ind -= 1  
                else:  
                    ind -= 1  
            else:  
                ind -= 1  
    return -1  
  
if \_\_name\_\_ == "\_\_main\_\_":  
    K = int(input("input denomination in one number please\n"))  
    L = list(map(int, input("input 6 number to show the number of sheet of each type of banknote\n").split()))  
    C = [1, 5, 10, 20, 50, 100]  
    ans = myfun(K, L) - 1  
    if ans == -2:  
        print("operation impossible")  
    else:  
        print(ans)

## 运行结果

输入：金额=150，1元纸币=2,5元纸币=5,10元纸币=4,20元纸币=1,50元纸币=8,100元纸币=1

输出:最小纸币数量=2

输入：金额=60，1元纸币=0,5元纸币=0,10元纸币=0,20元纸币=3,50元纸币=1,100元纸币=0

输出:最小纸币数量=3

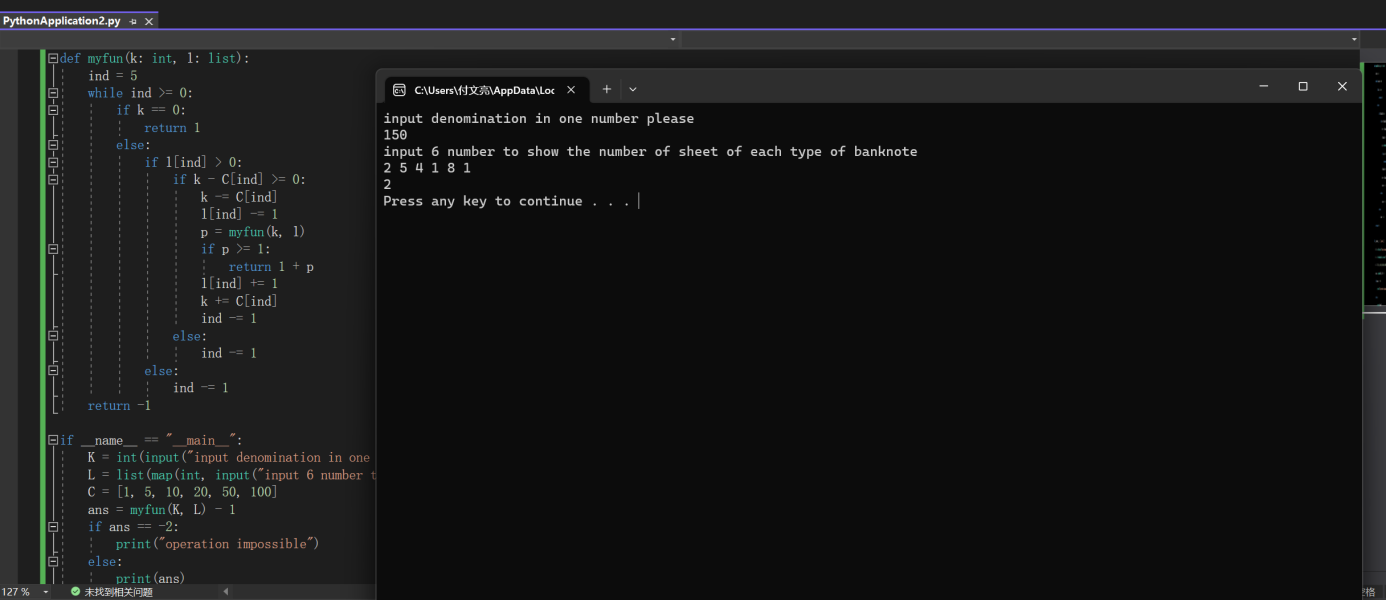
输入：金额=167，1元纸币=16,5元纸币=0,10元纸币=0,20元纸币=3,50元纸币=1,100元纸币=2

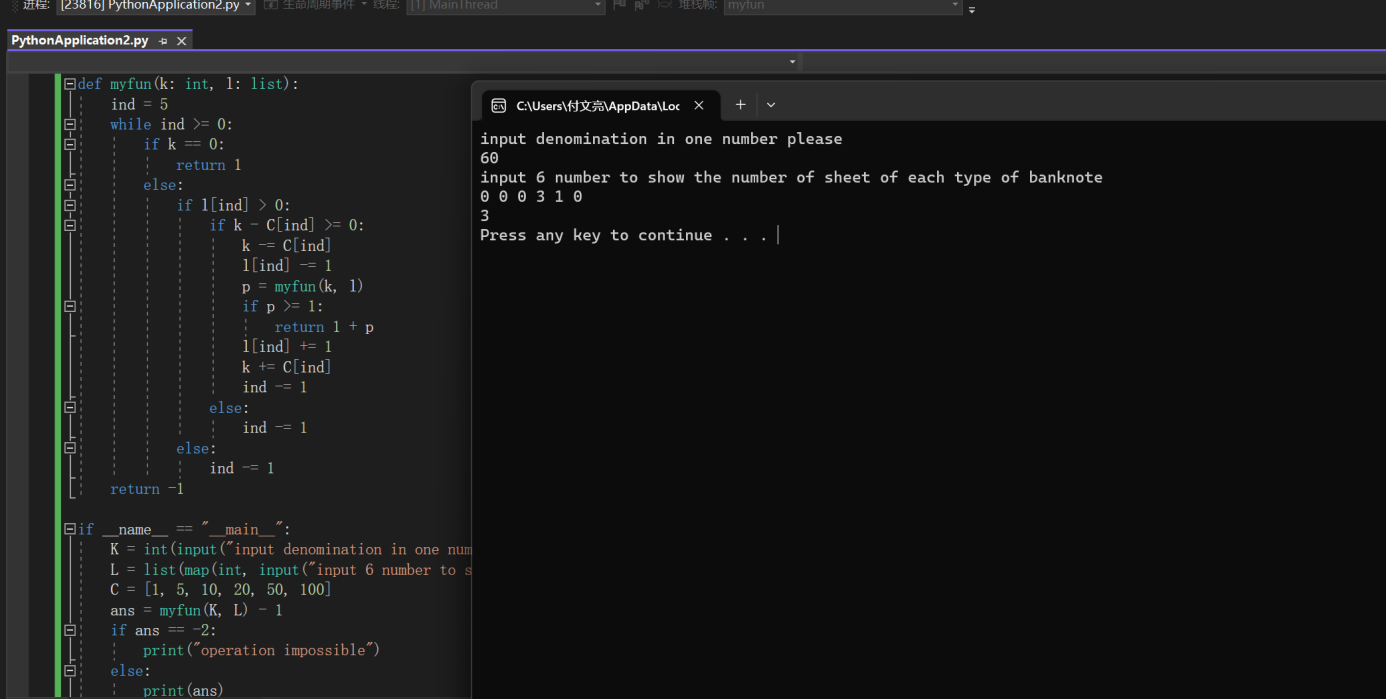
输出:最小纸币数量=11

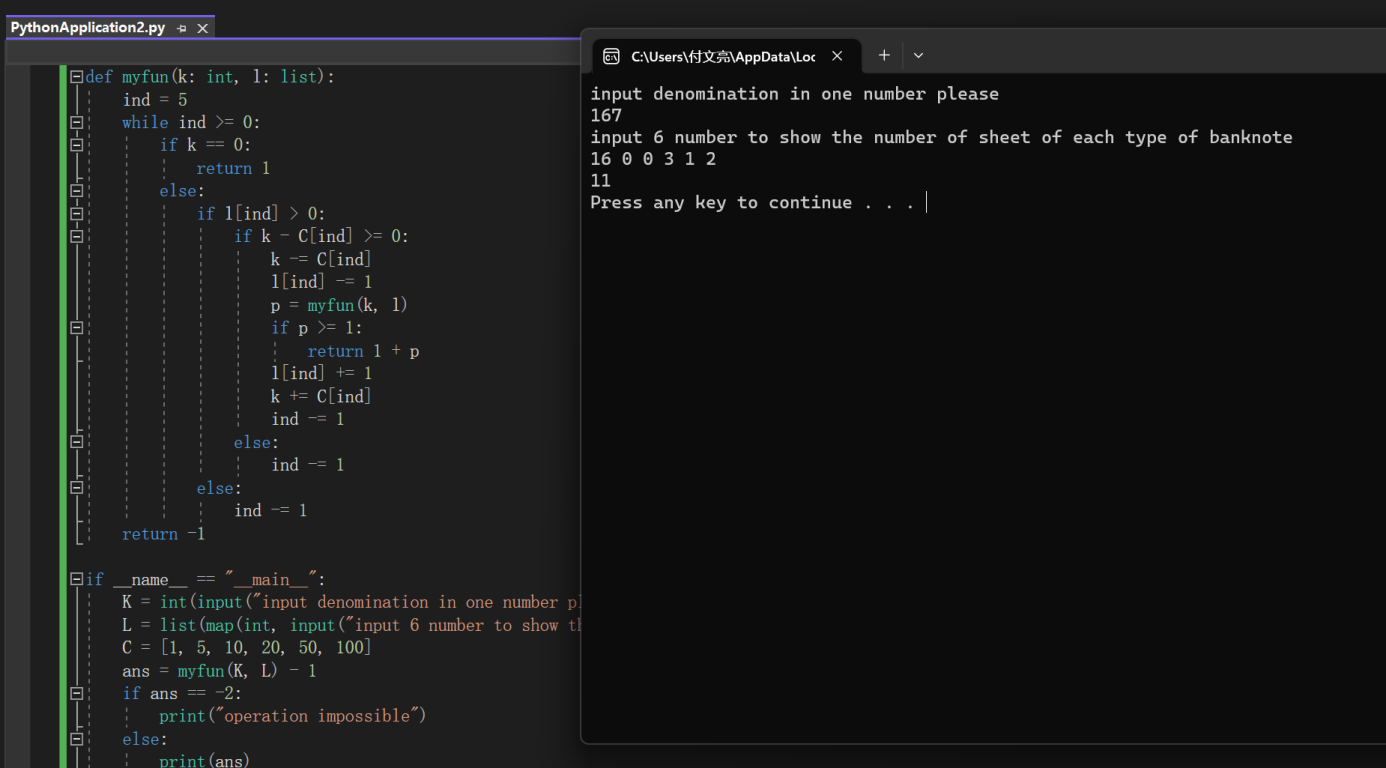
输入：金额=167，1元纸币=3,5元纸币=0,10元纸币=0,20元纸币=3,50元纸币=1,100元纸币=2

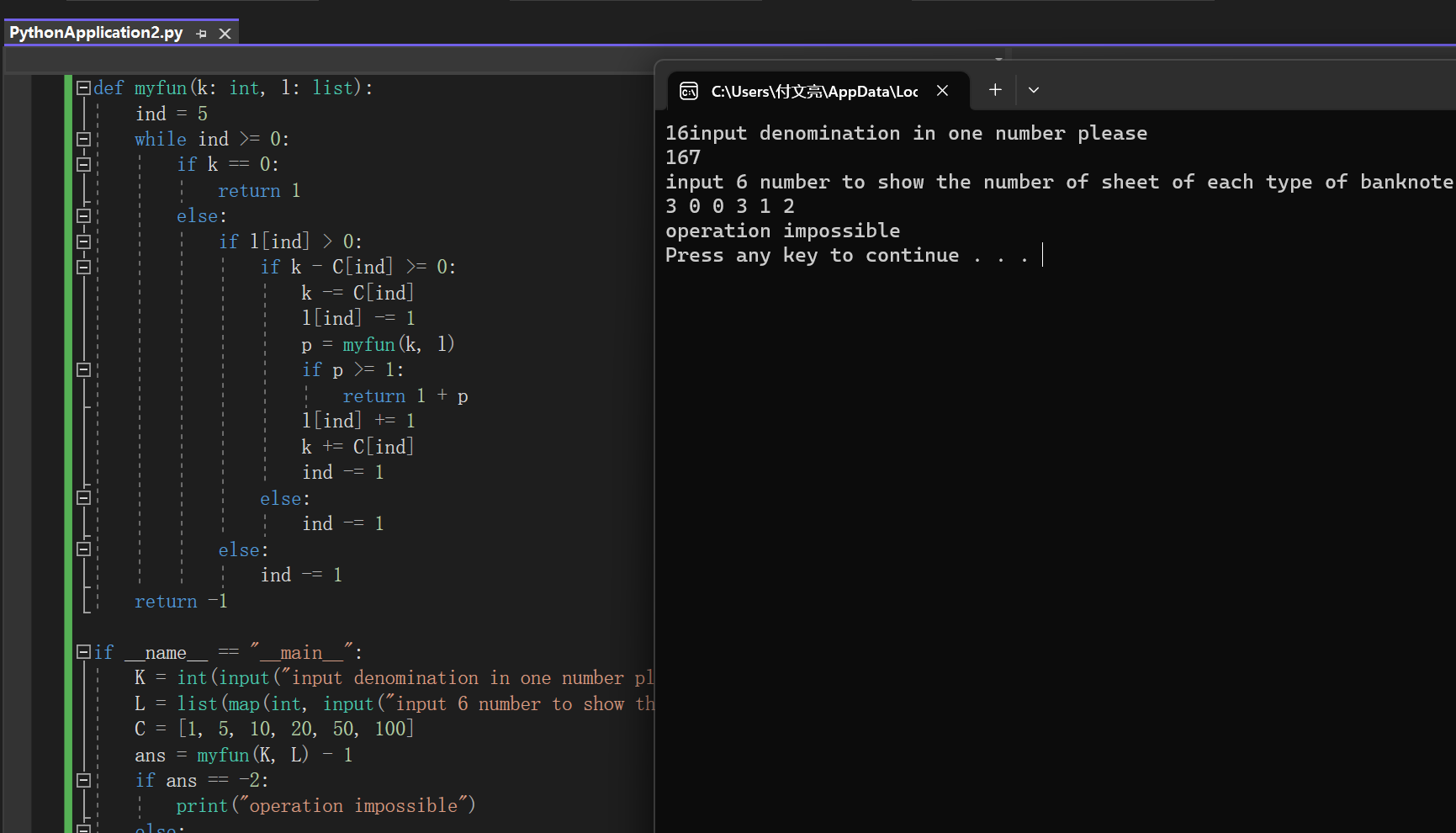
输出:最小纸币数量=不存在

## 实验结果分析









该程序在人为可快速判断的情况下计算出了正确的结果，并且对于无法满足问题要求的解也可正确输出不存在解，输出的结果也十分简洁明了。

## 优缺点及改进方向

优点:该程序让机器可以帮助人们解决计算最少纸币问题，并且当机器错误的引入过大面值的纸币造成无解的情况可以自动更正，遍历所有情况来判断最终是否有解，该程序合理运用了递归思想，让代码看起来更加简洁，让人易于理解。

缺点:当数据巨大的情况下，运行效率低、速度慢，并且程序执行的时间普遍偏长。

改进方向:对核心算法进行优化，降低代码时间复杂度，提高普遍运算速度和巨大数据情况下的运算速度。

## 心得体会与总结

学会了调试程序，并恰当的寻找合适的测试用例，当程序出现某些用例情况下错误时可以及时发现并做出相应调整，同时也认识到了测试的必要性，会有很多遗漏的情况没有考虑进初次代码的设计中来。

学会了贪心算法和递归思想，对循环嵌套有了深一步的了解，为以后实现写出简洁高效的程序代码提供了经验。

# **实验2：数独求解**

## 问题分析

数独游戏，一个古老而有趣的游戏。规则很简单：玩家需要根据9×9盘面上的已知数字，推理出所有剩余空格的数字，并满足每一行、每一列、每一个粗线宫（3\*3）内的数字均含1-9，不重复。这个数独游戏求解出来颇有难度，但是通过计算机的强大计算能力，我们可以使用Python轻松解决这个问题。

## 系统设计与算法设计

模块主要分为三个相互联系的函数组成，输入一个9\*9的二维列表，这三个函数对列表进行一系列的操作，最后使用print()函数输出。

函数isempty()判断数组是否有空元素，如果没有空元素则代表问题已经解决，如果有空元素则返回空元素的坐标。

函数isvalid()判断填入的数字是否为有效数字，即不能跟同一行，同一列，同一个粗线宫的已经存在的数字相同。

函数shudu()是提供的接口函数，主要运用穷举和递归的思想，每递归一次，则会在一个空元素上填一个有效数字。如果递归到后面发现前面填的数字有误，则会返回前面的状态再次填写下一个有效数字。直到填上了所有的有效数字，即为解决问题。

## 编写程序

# 数独

import numpy as np

def is\_valid(k: int,row: int,col: int):

for i in range(9):

if arr[row][i]==k:

return False

for i in range(9):

if arr[i][col]==k:

return False

for i in range(3):

for j in range(3):

l=arr[i+(row//3)\*3][j+(col//3)\*3]

if k==l:

return False

return True

def is\_empty():

for i in range(9):

for j in range(9):

if arr[i][j]==0:

return True,i,j

return False,9,9

def sudoku():

a=is\_empty()

if not a[0]:

return True

else:

i = a[1]

j = a[2]

for k in range(1, 10):

if is\_valid(k, i, j):

arr[i][j] = k

if sudoku():

return True

arr[i][j] = 0

return False

if \_\_name\_\_ == '\_\_main\_\_':

arr = np.arange(81).reshape(9, 9)

print("please input 9 lines and each line has 9 numbers")

for p in range(9):

for q, m in enumerate(list(map(int, input().split()))):

arr[p][q] = m

sudoku()

print("The answer of the sudoku is:\n", arr)

## 运行结果

输入：

1 0 0 0 0 6 8 2 0

0 0 0 3 0 9 5 0 0

0 0 0 0 0 0 0 0 6

0 6 0 0 1 0 0 0 0

0 0 0 5 0 0 1 3 0

4 5 0 0 0 0 9 0 0

6 1 0 0 3 7 0 0 4

5 0 0 0 2 0 0 0 0

0 0 7 0 0 0 3 0 0

输出：

The answer of the sudoku is:

[[1 9 4 7 5 6 8 2 3]

[8 2 6 3 4 9 5 1 7]

[7 3 5 1 8 2 4 9 6]

[3 6 2 9 1 8 7 4 5]

[9 7 8 5 6 4 1 3 2]

[4 5 1 2 7 3 9 6 8]

[6 1 9 8 3 7 2 5 4]

[5 8 3 4 2 1 6 7 9]

[2 4 7 6 9 5 3 8 1]]

输入：

1 0 4 7 5 6 0 2 3

0 2 6 3 4 0 5 1 0

7 3 5 1 0 2 4 0 6

3 0 2 0 1 0 7 4 5

0 0 0 5 6 4 1 3 2

4 5 1 2 0 3 0 6 0

6 1 0 0 3 7 2 5 4

5 0 3 4 2 1 6 7 0

2 4 0 6 0 5 3 0 1

输出：

The answer of the sudoku is:

[[1 8 4 7 5 6 9 2 3]

[9 2 6 3 4 8 5 1 7]

[7 3 5 1 9 2 4 8 6]

[3 6 2 8 1 9 7 4 5]

[8 7 9 5 6 4 1 3 2]

[4 5 1 2 7 3 8 6 9]

[6 1 8 9 3 7 2 5 4]

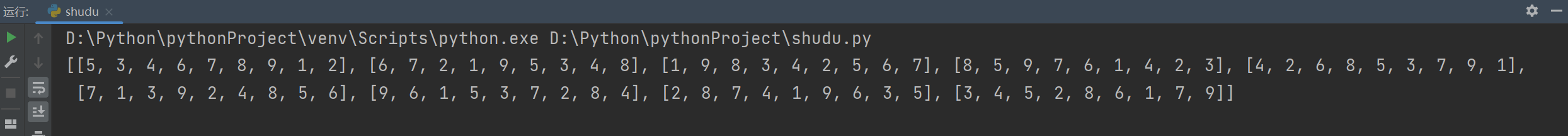
[5 9 3 4 2 1 6 7 8]

[2 4 7 6 8 5 3 9 1]]

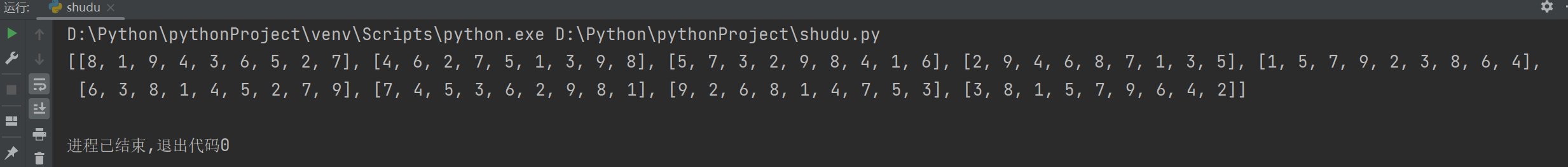
## 实验结果分析

(展示运行结果,对结果进行分析,甚至检验,程序作为附件,附在本doc文档后面;另外邮件包含原始程序文件.)

对第一个例子：



对第二个例子：



## 优缺点及改进方向

优点:该程序让机器可以帮助人们解决计算数独问题，遍历所有情况来判断最终是否有解，该程序合理运用了递归思想，把代码设计得逼近人的思维，让代码看起来更加简洁，让人易于理解。

缺点:当数据巨大的情况下，运行效率低、速度慢，并且程序执行的时间普遍偏长。

改进方向:对核心算法进行优化，降低代码时间复杂度，提高运算速度。

## 心得体会与总结

对递归在程序实现中的重要性有了进一步的认识，对Python中独特的函数中形参和实参的关系有了进一步的了解，加深了对模块化编程思想的理解，为以后实现写出简洁高效的程序代码提供了经验。

# **问题重述**

1. 最少纸币问题

用贪心算法的思想，很显然，每一步尽可能用面值大的纸币即可。如果大的不行，就换小的重新试。可以用循环和递归解决。

1. 数独问题

数独很适合使用递归，本质上来说解决方案就是穷举加递归的结合。

# **对本次实验的设计提出改进意见**

所给的问题比较少且比较局限，应该给三个或三个以上问题，让小组成员各自发挥自己的能力，代码能力更能得到锻炼。

# **附件**

## 附件1.最少纸币问题的程序（递归）

(这里给出程序代码)

def myfun(k: int, l: list):  
    ind = 5  
    while ind >= 0:  
        if k == 0:  
            return 1  
        else:  
            if l[ind] > 0:  
                if k - C[ind] >= 0:  
                    k -= C[ind]

              l[ind] -= 1  
                    p = myfun(k, l)  
                    if p >= 1:  
                        return 1 + p  
                    l[ind] += 1  
                    k += C[ind]  
                    ind -= 1  
                else:  
                    ind -= 1  
            else:  
                ind -= 1

    return -1  
  
if \_\_name\_\_ == "\_\_main\_\_":  
    K = int(input("input denomination in one number please\n"))  
    L = list(map(int, input("input 6 number to show the number of sheet of each type of banknote\n").split()))

    C = [1, 5, 10, 20, 50, 100]

    ans = myfun(K, L) – 1

    if ans == -2:  
        print("operation impossible")  
    else:  
        print(ans)

## 附件2.最少纸币问题的程序（循环）

# 最少要用多少张纸币--贪心算法  
def myfun(K:int, L:list):  
 K\_pre=K  
 L\_pre=[[100, L[0]], [50, L[1]], [20, L[2]], [10, L[3]], [5, L[4]], [1, L[5]]]  
 if K==0:  
 return 0  
 for x in range(6):  
 # 初始化  
 K\_now=K\_pre  
 L\_now = [L\_pre[0].copy(), L\_pre[1].copy(), L\_pre[2].copy(), L\_pre[3].copy(), L\_pre[4].copy(), L\_pre[5].copy(), 0]  
 result=0  
  
 while K\_now >= L\_now[x][0] and L\_now[x][1] > 0:  
 K\_now -= L\_now[x][0]  
 result += 1  
 L\_now[x][1] -= 1  
 if K\_now == 0:  
 return result  
  
 for z in range(x+1,6):  
 L\_now2=[L\_now[0].copy(),L\_now[1].copy(),L\_now[2].copy(),L\_now[3].copy(),L\_now[4].copy(),L\_now[5].copy(),0]  
 K\_now2=K\_now  
 result2=result  
 for y in range(z,6):  
 while K\_now2>=L\_now2[y][0] and L\_now2[y][1]>0:  
 K\_now2-=L\_now2[y][0]  
 result2+=1  
 L\_now2[y][1]-=1  
 if K\_now2==0:  
 return result2  
 return 'impossible'  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 K=int(input())  
 L=list(map(int,input().split()))  
 L.reverse()  
 print(myfun(K,L))

## 附件3.数独问题的程序

# 数独

import numpy as np

def is\_valid(k: int,row: int,col: int):

for i in range(9):

if arr[row][i]==k:

return False

for i in range(9):

if arr[i][col]==k:

return False

for i in range(3):

for j in range(3):

l=arr[i+(row//3)\*3][j+(col//3)\*3]

if k==l:

return False

return True

def is\_empty():

for i in range(9):

for j in range(9):

if arr[i][j]==0:

return True,i,j

return False,9,9

def sudoku():

a=is\_empty()

if not a[0]:

return True

else:

i = a[1]

j = a[2]

for k in range(1, 10):

if is\_valid(k, i, j):

arr[i][j] = k

if sudoku():

return True

arr[i][j] = 0

return False

if \_\_name\_\_ == '\_\_main\_\_':

arr = np.arange(81).reshape(9, 9)

print("please input 9 lines and each line has 9 numbers")

for p in range(9):

for q, m in enumerate(list(map(int, input().split()))):

arr[p][q] = m

sudoku()

print("The answer of the sudoku is:\n", arr)