**《算法分析与设计》**

**实验一 蛮力法 实验报告**

**授课教师： 李 朝 晖**

**实验指导教师： 李 朝 晖**

**教学对象： 大三年级**

**开课时间： 秋季学期**

**姓名： 朱安民**

**班级：2016211502**

**学号：2016211974**

2018年10月21日

**一、**实验目的

1. 深刻理解并掌握蛮力法的设计思想；
2. 提高应用蛮力法设计算法的技能；

二、实验任务

用蛮力法编程解决实验内容基本题，分析算法的时间复杂度。学有余力者继续解决提高题。

**三、**实验内容

**基本题：** **0—1背包问题**

用蛮力法编程实现0/1背包问题：给定n个重量为{w1, w2, … ,wn}、价值为{v1, v2, … ,vn}的物品和一个容量为C的背包，求这些物品中的一个最有价值的子集，并且要能够装到背包中。

**提高题：字符串匹配问题**

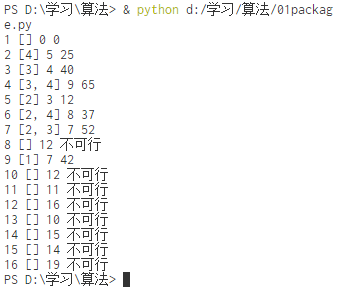
给定一个文本，在该文本中查找并定位任意给定字符串，实现BF算法或BF算法的改进算法KMP算法。

**四、**实验环境

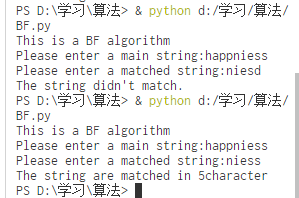
Python3.7 win10系统

1. 实验结果

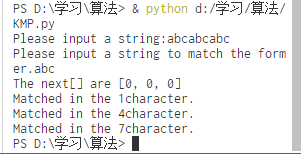
**基本题： 0—1背包问题**

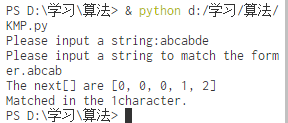


**提高题：字符串匹配问题 BF算法实现**

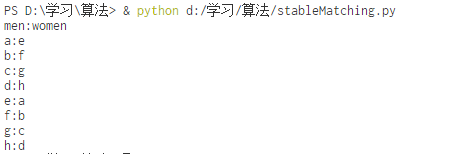


**提高题：字符串匹配问题 KMP算法实现**





**选做：稳定婚姻匹配问题**



1. 实验代码

**基本题：**

# 0/1背包问题

# 初始化物品的重量和价值，0,0表示未选中时的重量和价值

item1=[[0,0],[7,42]]

item2=[[0,0],[3,12]]

item3=[[0,0],[4,40]]

item4=[[0,0],[5,25]]

MAX=10 #背包最大重量

son=[] #所能容纳的背包的子集

no=0 #挨个测试的序号

# 增加背包子集

def addson(a,b,c,d,s):

if(a!=0):

s.append(1)

if(b!=0):

s.append(2)

if(c!=0):

s.append(3)

if(d!=0):

s.append(4)

# 主程序，穷举测试每种情况

for i in range(2):

for j in range(2):

for k in range(2):

for l in range(2):

sumH=item1[i][0]+item2[j][0]+item3[k][0]+item4[l][0]

if(sumH < MAX):

sumM=item1[i][1]+item2[j][1]+item3[k][1]+item4[l][1]

addson(i,j,k,l,son)

else:

sumM="不可行"

no=no+1

print(no,son,sumH,sumM)

son.clear()

**提高题：BF算法**

# BF算法 蛮力法解决字符串匹配问题

# 获取输入

print("This is a BF algorithm")

mainStr = input("Please enter a main string:")

subStr = input("Please enter a matched string:")

# 求长度

n = len(mainStr)

m = len(subStr)

# 判断字符串是否有效

if(m > n):

print("The main string should larger than matched string!")

else:

i = 0 # 主串浮动指针

j = 0 # 子串浮动指针

while(i < (n-m+1)): # 当主串指针已经到n-m+1时，肯定无法匹配

while(j < m): # 子串的指针不超过其长度m

if(mainStr[i] != subStr[j]): # 如果对应字符不相等，则跳出内循环

i=i-j+1########################

j=0

break

else: # 如果相等，继续比较

i = i+1

j = j+1

if(j == m): # 跳出内循环后测试是否已经完成了匹配

print("The string are matched in "+str(i-m+1)+"character")

if(i >= (n-m+1) and j != m): # 测试当i全部尝试匹配时，j是否没有完成匹配

print("The string didn't match.")

**提高题：KMP算法**

def getNext(s):

next = [] # 保存子串的next值

k = 0 # 匹配串（重复部分）的索引

i = 1 # 子串字符索引

next.append(0) # 初始化为0

l = len(s) # 子串长度

while(i < l):

# 当有一个字符匹配时，就看接下来的是否匹配，如果否，则后退匹配串，直到重新发现一个匹配或者其他，最后k为0

while(k > 0 and s[i] != s[k]):

k = next[k-1]

if(s[i] == s[k]): # 如果上一个字符匹配，下一个字符匹配时会增加k，即k是叠加而来

k = k+1

next.append(k) # 每次匹配后，给当前字符的next赋值

i = i+1

return next

s1 = input("Please input a string:")

s2 = input("Please input a string to match the former.")

m1 = len(s1)

m2 = len(s2)

next = getNext(s2)

print("The next[] are "+str(next))

i, j = 0, 0

while(i < m1 and m1>=m2):# 当主串指针没有到尾部，主串比子串长时循环继续

while(j>0 and s1[i] != s2[j]):# 当子串指针大于0时，如果不等，i不变，重新寻找适合子串的指针j

j=next[j-1]

if(s1[i]==s2[j]):#如果字符相等，子串指针向后移

j=j+1

if(j==m2): # 判断条件，多次匹配

print("Matched in the "+str(i+2-m2)+"character.")

j = 0

i=i+1#主串指针向后移

**选做：稳定婚姻匹配问题**

men = [

{

"name": "a",

"dream": ['e', 'f', 'g', 'h'],

"mate":""

},

{

"name": "b",

"dream": ['f', 'g', 'h', 'e'],

"mate":""

},

{

"name": "c",

"dream": ['g', 'e', 'h', 'f'],

"mate":""

},

{

"name": "d",

"dream": ['h', 'e', 'f', 'g'],

"mate":""

}

]

women = [

{

"name": "e",

"dream": ['d', 'a', 'c', 'b'],

"mate":""

},

{

"name": "f",

"dream": ['c', 'd', 'a', 'b'],

"mate":""

},

{

"name": "g",

"dream": ['b', 'a', 'c', 'd'],

"mate":""

},

{

"name": "h",

"dream": ['a', 'b', 'd', 'c'],

"mate":""

}

]

def compare(woman, man, singledogs):#比较并决定是否抛弃配偶

if(woman["dream"].index(woman["mate"]) > woman["dream"].index(man["name"])):

singledogs.append(woman["mate"])

return True

else:

return False

#初识的单身狗们

singledogs = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']

while(singledogs):#当还存在单身狗，则继续循环

for man in men:#对于每个男人

for dreamer in man["dream"]:#那么他将按照先后顺序依次向女士求婚

if man["name"] not in singledogs:#如果这个男人不是单声狗，则没必要继续求婚

break

for woman in women:##对于每个女人

if(woman["name"] == dreamer):#如果他找到了这个女人

if(woman["mate"]==""):#女人未婚，他们美好的生活

woman["mate"] = man["name"]

man["mate"] = woman["name"]

singledogs.remove(man["name"])

singledogs.remove(woman["name"])

break

elif(compare(woman, man, singledogs)):#如果已婚，女人进行比较，淘汰不喜欢的人

singledogs.remove(man["name"])

break

else:

pass#如果女人已婚而且被该女人淘汰，则寻找下一个女人

print("men:women")

for man in men:

print(man["name"]+":"+man["mate"])

for woman in women:

print(woman["name"]+":"+woman["mate"])