**实验四 贪心算法应用**

**学号： 20211104040**

**姓名： 曹文星**

**班级：软工二班**

**日期： 5.8, 2023**

**一、实验目的**

1、深入理解贪心策略的基本思想。

2、能正确采用贪心策略设计相应的算法，解决实际问题。

3、掌握贪心算法时间空间复杂度分析，以及问题复杂性分析方法

**二、实验内容**

1、随机生成500个0/1背包问题（问题规模可以相对较小），分别使用贪心算法和动态规划进行求解，

要求：1）统计贪心算法求得最优值的概率，

2）计算比值，

3）应用贪心算法求解时，统计最坏的情况下误差有多大，

4）实验结果跟实验设置的参数（如：背包容量、物品的体积）关系很大，简要分析参数对结果的影响。

2：用贪心算法编程求解以下任务安排问题

一个单位时间任务是恰好需要一个单位时间完成的任务。给定一个单位时间任务的有限集S。关于S的一个时间表用于描述S中单位时间任务的执行次序。时间表中第1个任务从时间0 开始执行直至时间1 结束，第2 个任务从时间1 开始执行至时间2 结束，…，第n个任务从时间n-1 开始执行直至时间n结束。  
具有截止时间和误时惩罚的单位时间任务时间表问题可描述如下。  
(1) n个单位时间任务的集合S={1,2,…,n}；  
(2) 任务i的截止时间di ,1≤i≤n,1≤ di ≤n，即要求任务i在时间di 之前结束；  
(3) 任务i的误时惩罚wi ,1≤i≤n,即任务i未在时间di 之前结束将招致wi 的惩罚；若按时完成则无惩罚。

***已知：***给定的n 个单位时间任务，各任务的截止时间di ,各任务的误时惩罚wi ,1≤i≤n，

***要求***：确定S的一个时间表（最优时间表）使得总误时惩罚达到最小。

**三、设计分析（**要说明贪心选择）

对于实验一,需要随机生成500个0/1背包问题（问题规模可以相对较小），分别使用贪心算法和动态规划进行求解. 对此,需要完成贪心算法以及动态规划的函数式调用,同时生成正确的测试用例.贪心选择为: 如果容量还有剩余,那么就将目标对象放入背包. 动态规划为DP方式经典实现(见以下代码)

对于实验二,计划将所有的任务按照误时惩罚wi 从大到小排序，得到一个序列S。之后初始化一个空的时间表solution。然后对于S中的每个任务，按顺序选择下一个任务i，并检查是否可以在截止时间di 之前完成。如果可以，就将任务i 加入到solution中；如果不可以，就跳过任务i。最后返回结果

**四、算法描述及程序（说明）**

代码及说明如下:

实验一 python实现:

# 随机生成500个0/1背包问题（问题规模可以相对较小），分别使用贪心算法和动态规划进行求解

import random

# 贪心

def greedy\_knapsack(weight, value, capacity):

items = sorted(zip(weight, value), key=lambda x: x[1] / x[0], reverse=True) # 按单位价值排序物品, 使用了lambda表达式计算单位价值

total\_weight = 0 # 初始化结果

total\_value = 0

for w, v in items: # 遍历物品

if total\_weight + w <= capacity: # 如果当前物品可以放入背包,就给我放入狠狠的贪心

total\_weight += w # 更新结果

total\_value += v

else:

continue # 否则跳过当前物品

return total\_value

# 动态规划算法

def dp\_knapsack(weight, value, capacity):

n = len(weight) # 获取物品数量

dp = [[0] \* (capacity + 1) for \_ in range(n + 1)] # 初始化状态数组

for i in range(1, n + 1): # 遍历物品

for j in range(1, capacity + 1): # 遍历背包容量

# 如果当前物品可以放入背包

if weight[i - 1] <= j:

dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - weight[i - 1]] + value[i - 1]) # 比较放入和不放入的价值，取较大者

else:

dp[i][j] = dp[i - 1][j] # 否则继承上一个状态

return dp[n][capacity] # 返回结果

def generate\_test\_data(): # 生成测试数据的函数

n = random.randint(1, 10) # 随机生成物品数量和背包容量，范围为1到10

capacity = random.randint(1, 10)

# 随机生成物品重量和价值，范围为1到5

weight = [random.randint(1, 5) for \_ in range(n)]

value = [random.randint(1, 5) for \_ in range(n)]

return n, capacity, weight, value # 返回测试数据

# 应用算法求解: 测试数据

# for u in range(5):

# n, capacity, weight, value = generate\_test\_data()

# print(f"Example {u + 1}:")

# print(f"n = {n}")

# print(f"capacity = {capacity}")

# print(f"weight = {weight}")

# print(f"value = {value}")

# print(f"Greedy: {greedy\_knapsack(weight, value, capacity)}")

# print(f"DP: {dp\_knapsack(weight, value, capacity)}")

# print()

# 统计贪心算法求得最优值的概率

# 计算比值:贪心/动态规划:

# 统计应用贪心算法求解时，统计最坏的情况下误差有多大:

n = 1000

count = ratio = max\_error = 0

for i in range(n):

n, capacity, weight, value = generate\_test\_data()

if (dp\_knapsack(weight, value, capacity) == 0):

continue

else:

if greedy\_knapsack(weight, value, capacity) == dp\_knapsack(weight, value, capacity):

count += 1

ratio += greedy\_knapsack(weight, value, capacity) / dp\_knapsack(weight, value, capacity)

max\_error = max(max\_error, greedy\_knapsack(weight, value, capacity) / dp\_knapsack(weight, value, capacity))

print(f"求得最优值的概率: {count / n}")

print(f"比值: {ratio / n}")

print(f"最坏的情况下误差: {max\_error}")

实验二:

# 将所有的任务按照误时惩罚wi 从大到小排序，得到一个序列S。

# 初始化一个空的时间表solution。

# 对于S中的每个任务，按顺序选择下一个任务i，并检查是否可以在截止时间di 之前完成。如果可以，就将任务i 加入到solution中；如果不可以，就跳过任务i。

# 返回solution作为最优时间表。

import random

# 定义一个任务类，包含任务编号，截止时间和误时惩罚

class Task:

def \_\_init\_\_(self, id, deadline, penalty):

self.id = id

self.deadline = deadline

self.penalty = penalty

# 定义一个比较函数，按照误时惩罚从大到小排序

def \_\_lt\_\_(self, other):

return self.penalty > other.penalty

# 定义一个函数，根据给定的任务集合，返回一个最优时间表

def schedule(tasks):

tasks.sort() # 将任务按照误时惩罚从大到小排序

solution = [] # 初始化一个空的时间表

slots = [False] \* len(tasks) # 初始化一个空的时间槽数组，用于记录每个时间槽是否被占用

for task in tasks: # 对于每个任务，按顺序选择下一个任务

for i in range(task.deadline - 1, -1, -1): # 从任务的截止时间开始向前寻找一个空闲的时间槽

if not slots[i]: # 贪心:如果找到了一个空闲的时间槽，就将任务放入该时间槽，并将其加入到时间表中

slots[i] = True

solution.append(task)

break

# 返回最优时间表

return solution

# 定义一个函数，根据给定的参数，生成一组随机的任务

def generate\_tasks(n, d\_min, d\_max, p\_min, p\_max):

# 初始化一个空的任务集合

tasks = []

# 对于每个编号，生成一个随机的截止时间和误时惩罚，并创建一个任务对象

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

deadline = random.randint(d\_min, d\_max)

penalty = random.randint(p\_min, p\_max)

task = Task(i, deadline, penalty)

# 将任务对象加入到任务集合中

tasks.append(task)

# 返回任务集合

return tasks

# 测试代码

# 指定任务的数量为10，截止时间的范围为1到10，误时惩罚的范围为10到100

n = 10

d\_min = 1

d\_max = p\_min = p\_max = 10

# 调用函数，生成一组随机的任务

tasks = generate\_tasks(n, d\_min, d\_max, p\_min, p\_max)

# 打印生成的任务信息

print("The generated tasks are:")

for task in tasks:

print("Task", task.id, "deadline:", task.deadline, "penalty:", task.penalty)

print("\n\n\n")

# 调用函数，得到最优时间表

solution = schedule(tasks)

# 打印结果

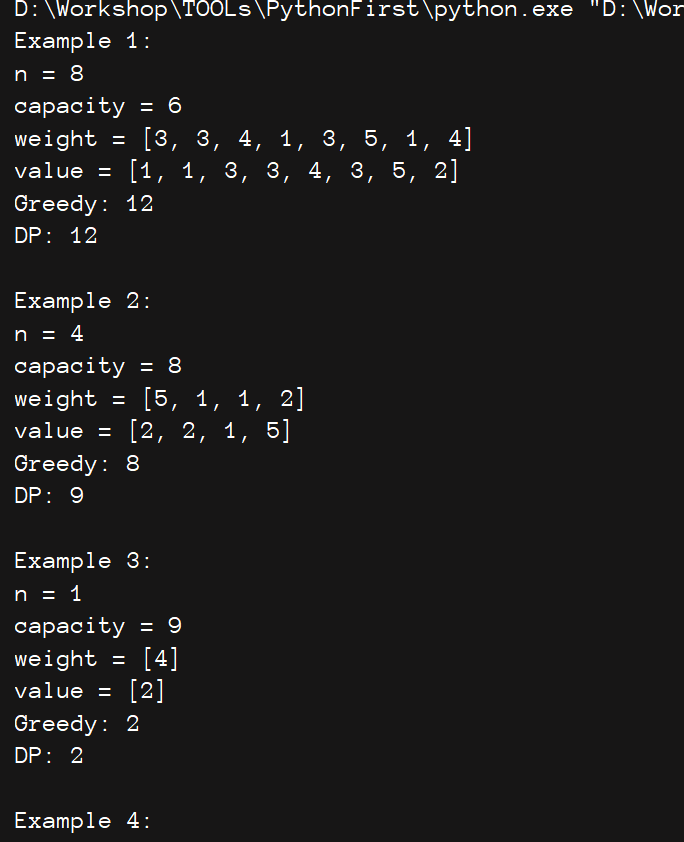
print("The optimal schedule is:")

for task in solution:

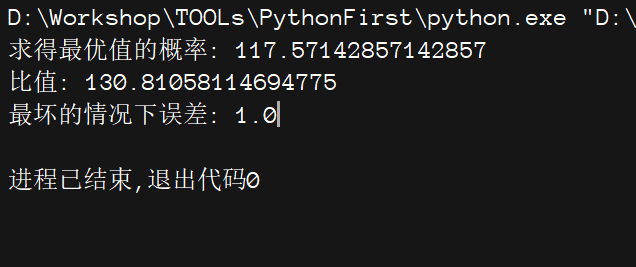
print("Task", task.id, ' + ', end='')

**五、测试与分析**

一:



测试样例输出



要求输出结果

问题四解答:

0/1背包问题中，主要有三个参数：物品的数量n，物品的重量w\_i和价值v\_i，以及背包的容量W。这些参数会影响问题的难度和最优解。

一般来说，物品的数量n越大，问题的难度越高，因为需要考虑更多的组合情况。如果使用动态规划求解，时间复杂度和空间复杂度都和n成正比。

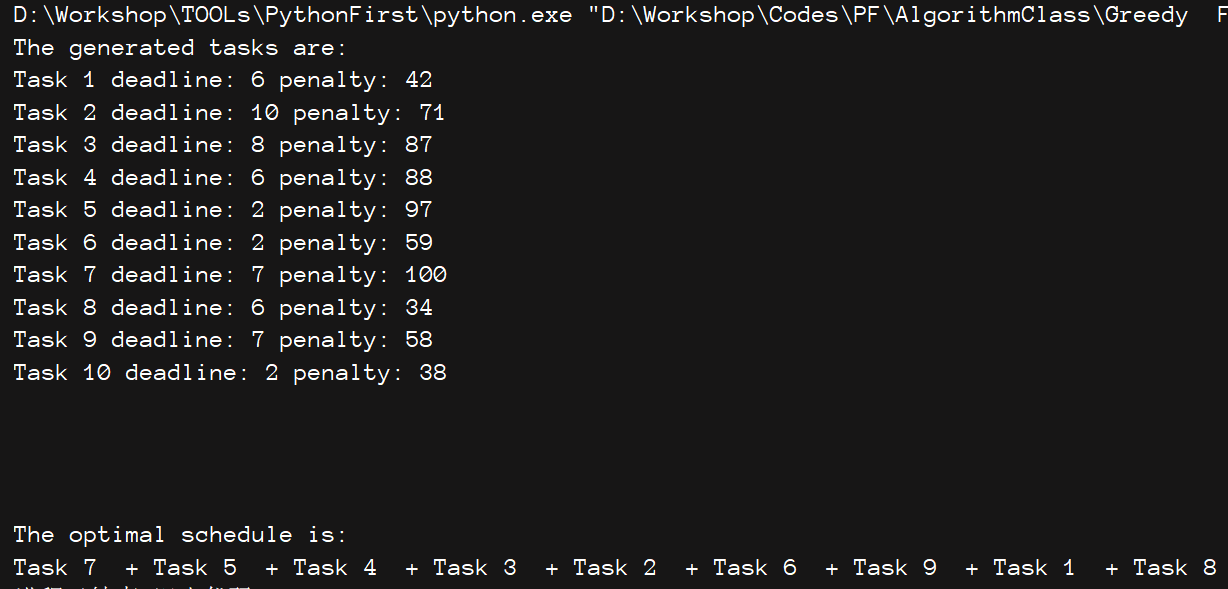
物品的重量w\_i和价值v\_i会影响问题的最优解。如果物品的重量和价值成正比，那么贪心算法可以得到最优解，只需要按照单位价值排序后依次放入背包即可。如果物品的重量和价值不成正比，那么贪心算法可能得不到最优解

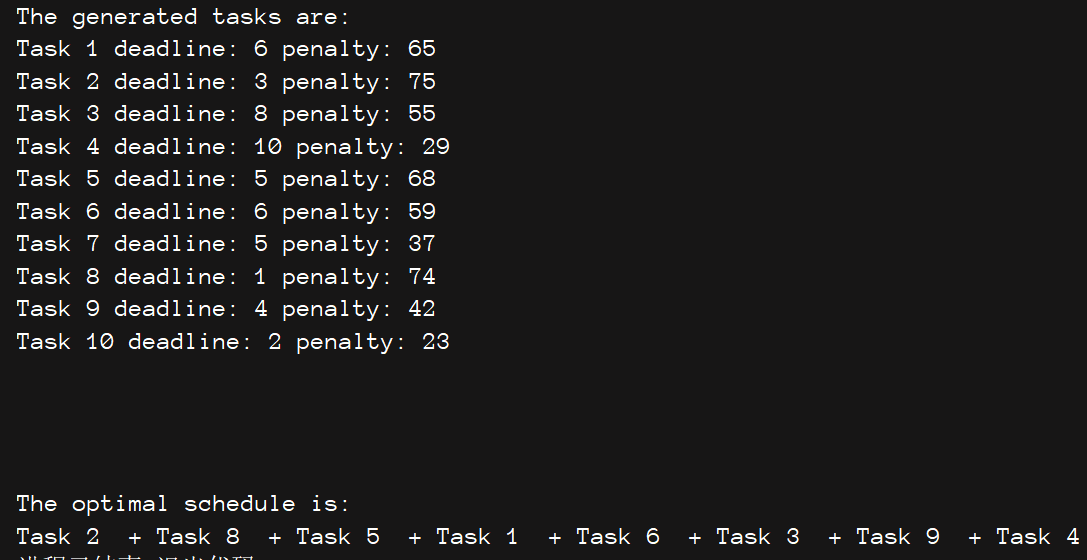
背包的容量W也会影响问题的难度和最优解。如果W很小，那么可能无法放入任何物品或者只能放入少数物品，问题比较简单。如果W很大，那么可能可以放入所有物品或者大部分物品，问题比较复杂。如果使用动态规划求解，时间复杂度和空间复杂度都和W成正比。

综上所述，0/1背包问题中参数对结果的影响是多方面的

二:

下面是一些生成例子





**六、实验总结与体会**

贪心算法是一种在每一步选择中都采取当前状态下最好或最优（即最有利）的选择，从而希望导致结果是最好或最优的算法。贪心算法不是对所有问题都能得到全局最优解，但对一些具有最优子结构的问题，贪心算法可以得到全局最优解或者近似解。所谓最笨的方法不一定就是好方法, 贪心算法对于我们生活中的应用也有很多, 今后也将在实际中运用.