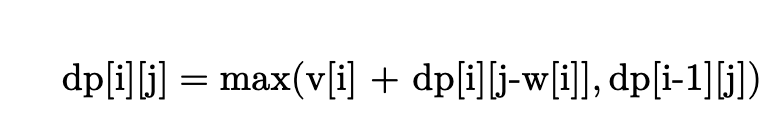
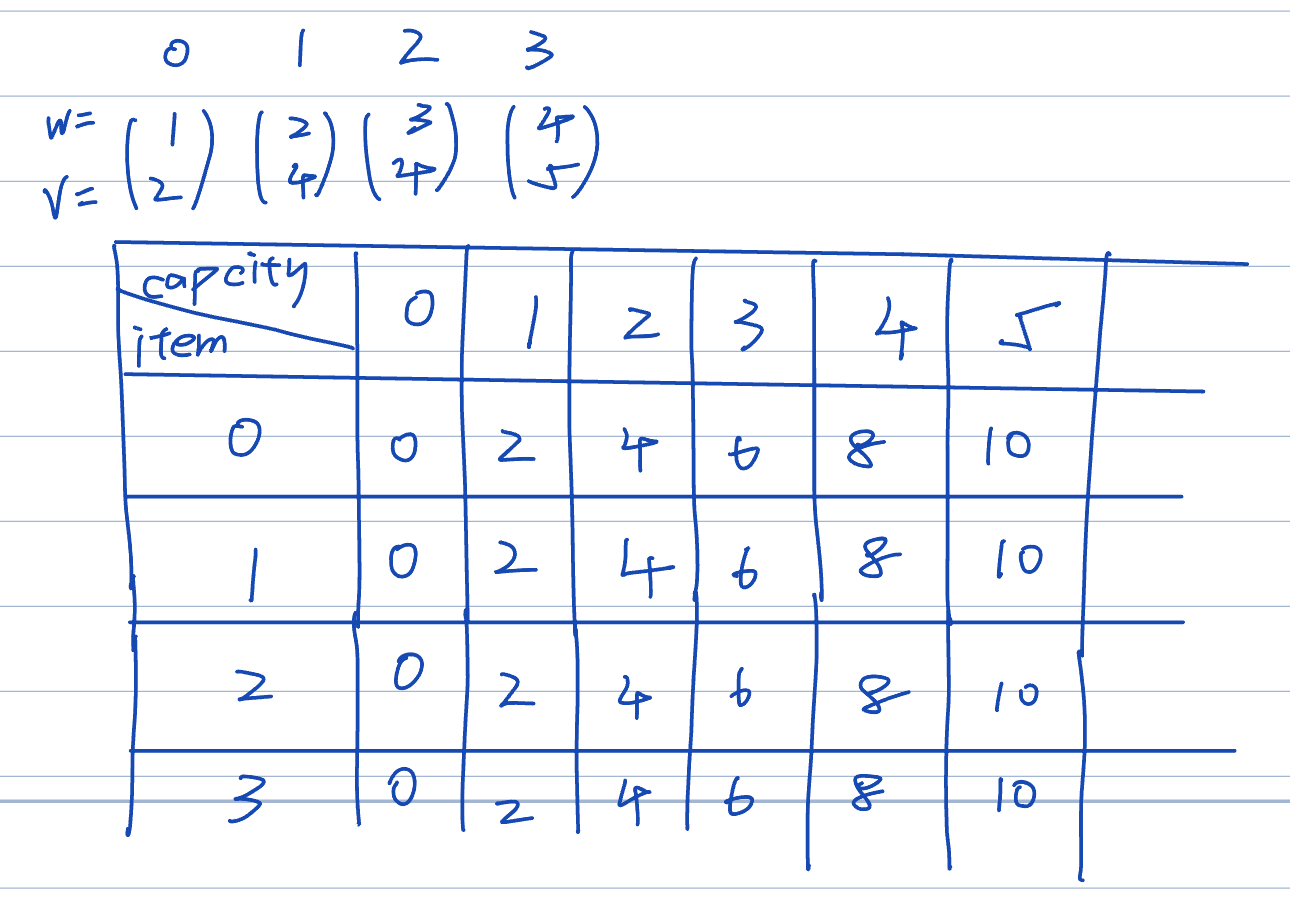
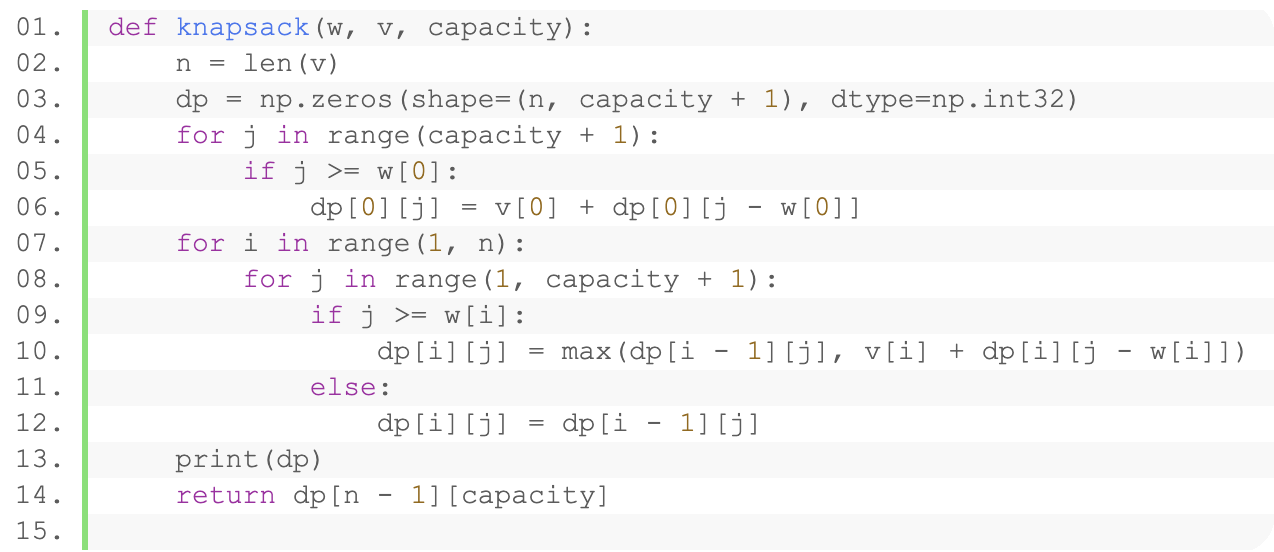
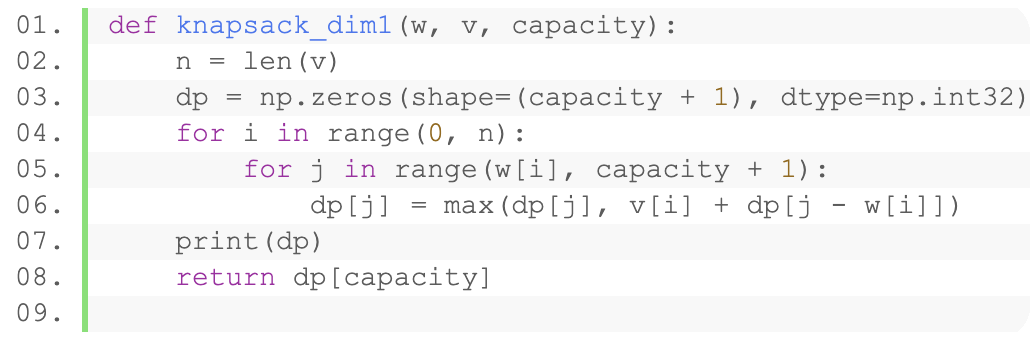
**算法设计与分析第四章作业**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **姓名** | **王靳** | **班级** | **计科十班** | **学号** | 220111012 |
| **第1题** |  | | | | |
| **第2题** |  | | | | |
| **第3题** |  | | | | |
| **总分** | **(满分：100分+附加10分)** | | | | |
| **备注** | 作业提交截止时间：2023-10-13日24:00，超过提交截至时间的作业视为无效。作业提交邮箱：[hitsz\_algorithm@126.com。](mailto:hitsz_algorithm@126.com。)作业文件名命名方式： 第x章-x班-姓名-学号（例， 第1章-1班-张三-220110101.docx）； 邮件主题为：第x章作业, x班，姓名，学号（例， 第1章作业，1班，张三，220110101）。缺少这些信息的作业将被酌情扣分。 | | | | |

1. 给定n种物品和一个背包，物品的重量是，价值, 背包容量为, **每个物品的数量无限且可重复选取**，问如何选择装入背包的物品，使装入背包中的物品的总价值最大？（40 分）
2. 请写出该问题的递推方程（定义为第种物品到第种物品装进限重为的背包可获得的最大价值）（10分）
3. 假设背包容量为**5**，有**4**种物品，其重量分别为，其价值分别为，请写出对应的矩阵（10分）
4. 请写出该问题的伪代码（10分）
5. 如果上述伪代码是用二维数组实现的，请问是否有空间更优化的的实现版本？提示：可否将2维数组降至1维数组。

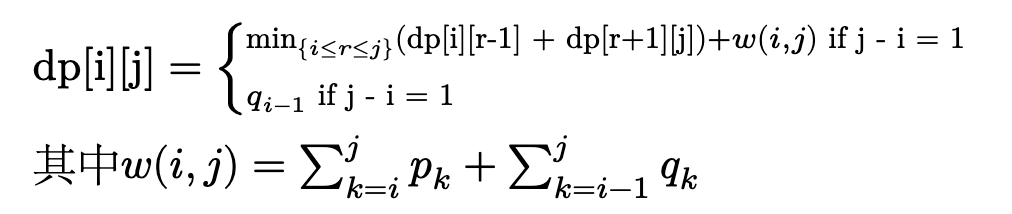
（**附加题**10 分）

可以进行状态压缩，因为dp[i][j]只依赖于dp[i-1][j]和dp[i][j-w[i]]这两个状态，也就是：只依赖于当前行和上一行，并且得到结果是dp[n-1][capacity]也就是结果只是最后一行就可以得到，因此可以进行状态压缩。依赖于当前行，所以要求遍历从左到右；依赖于上一行，所以要求遍历从上到下。并且可以看出，有元素进行了重复计算，所以是完全背包问题。

1. 若7个关键字的概率如下所示，求其最优二叉搜索树的结构和代价，要求必须写出递推方程。（30分）

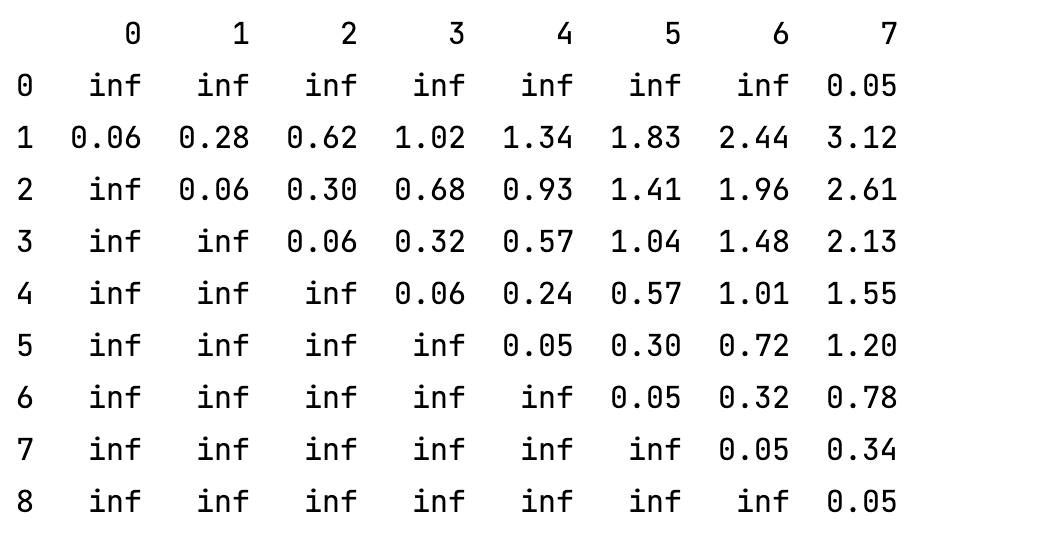
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|  |  | 0.04 | 0.06 | 0.08 | 0.02 | 0.10 | 0.12 | 0.14 |
|  | 0.06 | 0.06 | 0.06 | 0.06 | 0.05 | 0.05 | 0.05 | 0.05 |

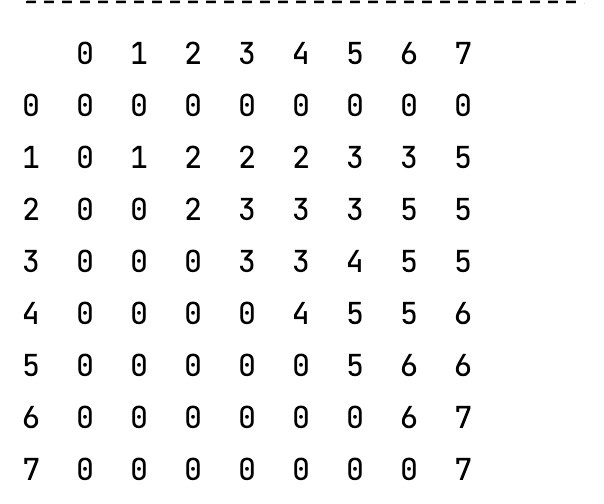
解：



1. import numpy as np
2. import pandas as pd
3. import networkx as nx
4. import matplotlib.pyplot as plt
5. def register\_w(p, q):
6. n = len(p)
7. *# 计算前缀和数组p\_sum*
8. p\_sum = [0]
9. for r in range(1, n):
10. p\_sum.append(p[r] + p\_sum[r - 1])
11. *# 计算前缀和数组q\_sum*
12. q\_sum = [q[0]]
13. for r in range(1, n):
14. q\_sum.append(q[r] + q\_sum[r - 1])
15. def inner(i, j):
16. if i == 1:
17. sigma\_i\_j\_p = p\_sum[j]
18. sigma\_i\_j\_q = q\_sum[j]
19. return sigma\_i\_j\_p + sigma\_i\_j\_q
20. else:
21. sigma\_i\_j\_p = p\_sum[j] - p\_sum[i - 1]
22. sigma\_i\_j\_q = q\_sum[j] - q\_sum[i - 1]
23. sigma\_i\_j\_q += q[i - 1]
24. return sigma\_i\_j\_p + sigma\_i\_j\_q
25. return inner
26. class Node:
27. def \_\_init\_\_(self, val):
28. self.value = val
29. self.left = None
30. self.right = None
31. *# 创建最优二叉搜索树*
32. *# 接收的root是一个二维数组*
33. def set\_opti\_bst(root, i, j, keys):
34. *# base case*
35. if i > j:
36. return None
37. if i == j:
38. k = keys[i]
39. n = Node("k" + str(k))
40. n.left = Node("d" + str(k - 1))
41. n.right = Node("d" + str(k))
42. return n
43. *# 根节点*
44. r = root[i][j]
45. k = keys[r]
46. n\_root = Node("k" + str(k))
47. n\_left = set\_opti\_bst(root, i, r - 1, keys)
48. n\_right = set\_opti\_bst(root, r + 1, j, keys)
49. if n\_left == None:
50. n\_left = Node("d" + str(k - 1))
51. n\_root.left = n\_left
52. else:
53. n\_root.left = n\_left
54. if n\_right == None:
55. n\_right = Node("d" + str(k))
56. n\_root.right = n\_right
57. else:
58. n\_root.right = n\_right
59. return n\_root
60. def create\_graph(G, node, pos={}, x=0, y=0, layer=1):
61. pos[node.value] = (x, y)
62. if node.left:
63. G.add\_edge(node.value, node.left.value)
64. l\_x, l\_y = x - 1 / 2 \*\* layer, y - 1
65. l\_layer = layer + 1
66. create\_graph(G, node.left, x=l\_x, y=l\_y, pos=pos, layer=l\_layer)
67. if node.right:
68. G.add\_edge(node.value, node.right.value)
69. r\_x, r\_y = x + 1 / 2 \*\* layer, y - 1
70. r\_layer = layer + 1
71. create\_graph(G, node.right, x=r\_x, y=r\_y, pos=pos, layer=r\_layer)
72. return (G, pos)
73. def draw(node):  *# 以某个节点为根画图*
74. graph = nx.DiGraph()
75. graph, pos = create\_graph(graph, node)
76. fig, ax = plt.subplots(figsize=(8, 10))  *# 比例可以根据树的深度适当调节*
77. nx.draw\_networkx(graph, pos, ax=ax, node\_size=300)
78. plt.show()
79. def opti\_bst(p, q):
80. n = len(p) - 1  *# 一共有7个关键字*
81. w = register\_w(p, q)
82. e = np.full((n + 1 + 1, n + 1), fill\_value=np.inf, dtype=np.float32)
83. root = np.zeros((n + 1, n + 1), dtype=np.int32)
84. for l in range(0, n + 2):  *# 从1到8初始化，多了一行，为了防止出现越界的情况*
85. e[l][l - 1] = q[l - 1]
86. *# i从n到1*
87. for i in range(n, 0, -1):
88. *# j从1到n*
89. for j in range(i, n + 1):
90. *# r从i到j*
91. for r in range(i, j + 1):
92. temp = e[i][r - 1] + e[r + 1][j] + w(i, j)
93. if temp < e[i][j]:
94. e[i][j] = temp
95. root[i][j] = r
96. return e, root
97. if \_\_name\_\_ == "\_\_main\_\_":
98. p = [0, 0.04, 0.06, 0.08, 0.02, 0.10, 0.12, 0.14]
99. q = [0.06, 0.06, 0.06, 0.06, 0.05, 0.05, 0.05, 0.05]
100. keys = [0, 1, 2, 3, 4, 5, 6, 7]
101. *# p = [0, 0.15, 0.1, 0.05, 0.1, 0.2]*
102. *# q = [0.05, 0.1, 0.05, 0.05, 0.05, 0.1]*
103. w = register\_w(p, q)
104. e, root = opti\_bst(p, q)
105. print(pd.DataFrame(e))
106. print('-' \* 100)
107. print(pd.DataFrame(root))
108. node = set\_opti\_bst(root, 1, len(keys) - 1, keys)
109. draw(node)







1. 灭鼠人计划消灭菜田中田鼠，每个鼠洞内都藏有一定数量的田鼠。这个菜田所有的鼠洞不互通且都围成一圈 ，**这意味着第一个鼠洞和最后一个鼠洞是紧挨着的**。同时，由于灭鼠动静过大，**左右相邻的鼠洞中的田鼠会听见灭鼠动静而全部迅速逃窜，逃窜的田鼠不会去往其他鼠洞，而是逃离这片菜田**。

给定一个代表每个鼠洞藏匿老鼠数量的非负整数数组，计算能够消灭田鼠的最大数量。（40分）

示例 1：

输入：nums = [2,3,2]

输出：3

解释：你不能先灭1 号鼠洞（数量 = 2），然后灭 3 号鼠洞（数量 = 2）, 因为他们是相邻的。

示例 2：

输入：nums = [1,2,3,1]

输出：4

解释：你可以先灭 1 号鼠洞（数量 = 1），然后灭 3 号鼠洞（数量 = 3）。

消灭田鼠最大数量 = 1 + 3 = 4 。

提示：

1 <= nums.length <= 100

0 <= nums[i] <= 1000

**要求：**

**运用动态规划的思想作答，请写出分析过程（10分）和状态转移方程（10分），并用一种语言实现你的思路（附完整源代码，20分），并保证代码能正确运行，复杂度尽可能低。**

解：这是一个打家劫舍的题目，因为是循环的，并不好考虑，我们可以掐头去尾。

有三种情况：1.首尾都不考虑；2.只考虑首；3.只考虑尾。但实际上，“只考虑首”+“只考虑尾”就已经包含了“首尾都不考虑”的情况。

我们定义一个线性的处理函数process。然后我们将：“只考虑首”和“只考虑尾”的情况分别代入process中，然后取得一个最大值即可.

Dp数组的含义：dp[i]表示考虑i位置，i位置有两种状态：灭鼠，不灭鼠

单个线性数组，process最终求解的目标是dp[nums.size-1]。

递推公式：

灭鼠：dp[i] = dp[i-2] + nums[i]

不灭鼠，延续上一次的状态：dp[i] = dp[i-1]

因此：dp[i] = max{ dp[i-2] + nums[i], dp[i-1] }

因为i是只增补减的，所以是线性的时间复杂度

初始化：dp[0] = nums[0]，如果只有一个老鼠洞，直接打；dp[1] = max{ nums[1],nums[0] }，其他位置可以是任何值，因为后面会覆盖掉

遍历顺序：因为第i号位置是依赖i-1位置和i-2位置，所以从左到右遍历即可

1. import numpy as np
2. def process(nums):
3. n = len(nums)
4. dp = np.zeros(shape=n, dtype=np.int32)
5. dp[0] = nums[0]
6. dp[1] = max(nums[0], nums[1])
7. for i in range(2, n):
8. dp[i] = max(dp[i - 1], dp[i - 2] + nums[i])
9. return dp[n - 1]
10. def kill\_mice(arr):
11. *# 掐头*
12. arr1 = arr[1:]
13. *# 去尾*
14. arr2 = arr[:-1]
15. return max(process(arr1), process(arr2))
16. if \_\_name\_\_ == "\_\_main\_\_":
17. arr = [1, 2, 3, 1]
18. ans = kill\_mice(arr)
19. print(ans)