**算法设计与分析第六章作业**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **姓名** | **王靳** | **班级** | **计科十班** | **学号** | 220111012 |
| **第1题** |  | | | | |
| **第2题** |  | | | | |
| **第3题** |  | | | | |
| **第4题** |  | | | | |
| **第5题** |  | | | | |
| **总分** |  | | | | |
| **备注** | 作业提交截止时间：2023年11月4日24:00，超过提交截至时间的作业视为无效。确因网络等特殊原因无法及时提交作业的学生，应至少提前1小时与助教联系沟通。作业提交邮箱：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、用本章知识解决下面的问题，写出你的思路和伪代码。

给你一个整数数组coins，表示不同面额的硬币；以及一个整数amount，表示总金额。

计算并返回可以凑成总金额所需的最少硬币的个数。如果没有任何一种硬币组合能组成总金额，返回-1。（你可以认为每种硬币的数量是无限的）

示例1：

输入：coins = [1,2,5], amount = 11

输出：3

解释：11 = 5 + 5 + 1

示例2：

输入：coins = [2], amount = 3

输出：-1

示例3：

输入：coins = [1], amount = 0

输出：0

提示：

1 <= coins.length <= 12

1 <= coins[i] <= 2^31 -1

0 <= amount <= 10^4

想法：深度优先搜索，通过递归来实现。

Base case: 如果rest==0，那么就可以搜集结果；如果rest<0，那么说明找零失败。

Else: 迭代器是i，遍历coins中所有的值。

搜索树中的节点就是函数的递归调用的信息，其中信息有：剩余的钱rest，已经找了几张钞票num。那么子节点就是：utils(rest-coins[i], num + 1)

我们可以在这个搜索树上做一些优化：剪枝。通过分支限界法：因为我们要的是最少的钱币数量，因此我们需要直到当前节点的下界是多少，如果下界比已知的解还要大的话，就没有继续遍历下去的必要了，这就可以直接剪枝了。下界的设计也非常的直观，rest/max(coins)+num，我们至少需要找rest/max(coins)这么多钱，然后再加上已经找过的钱num。

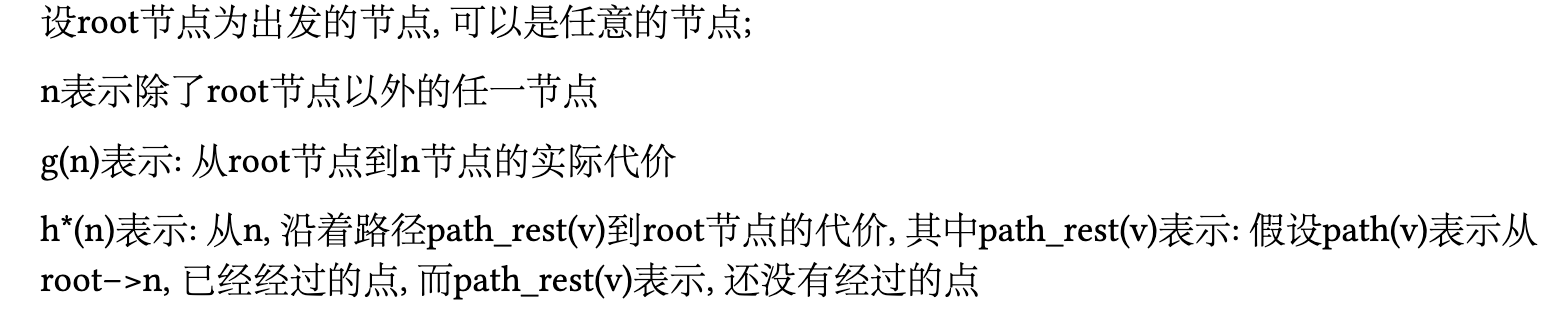
1. class solution:
2. def \_\_init\_\_(self, amount: int, coins: list) -> None:
3. self.amount: int = amount
4. self.coins: list = sorted(coins, reverse=True)
5. self.up: int = max(coins)
6. self.ok: int = None
7. def utils(self, rest\_money: int, num: int) -> None:
8. """递归函数
9. Args:
10. rest\_money (int): 剩余的钱
11. num (int): 已有的钱的数量
12. """
13. if rest\_money == 0:
14. *# 搜集结果*
15. if (self.ok == None) or (num < self.ok):
16. self.ok = num
17. return
18. if rest\_money < 0:
19. *# 找零失败*
20. return
21. if (self.ok != None) and (rest\_money / self.up + num > self.ok):
22. *# 分支限界*
23. return
24. for i in self.coins:
25. self.utils(rest\_money - i, 1 + num)
26. pass
27. def handle(self):
28. self.utils(self.amount, 0)
29. if self.ok == None:
30. print(-1)
31. else:
32. print(self.ok)
33. if \_\_name\_\_ == "\_\_main\_\_":
34. print("hello world")
35. s = solution(0, [1])
36. s.handle()

2、给定一个4个点的连通有向图，其邻接矩阵如下：

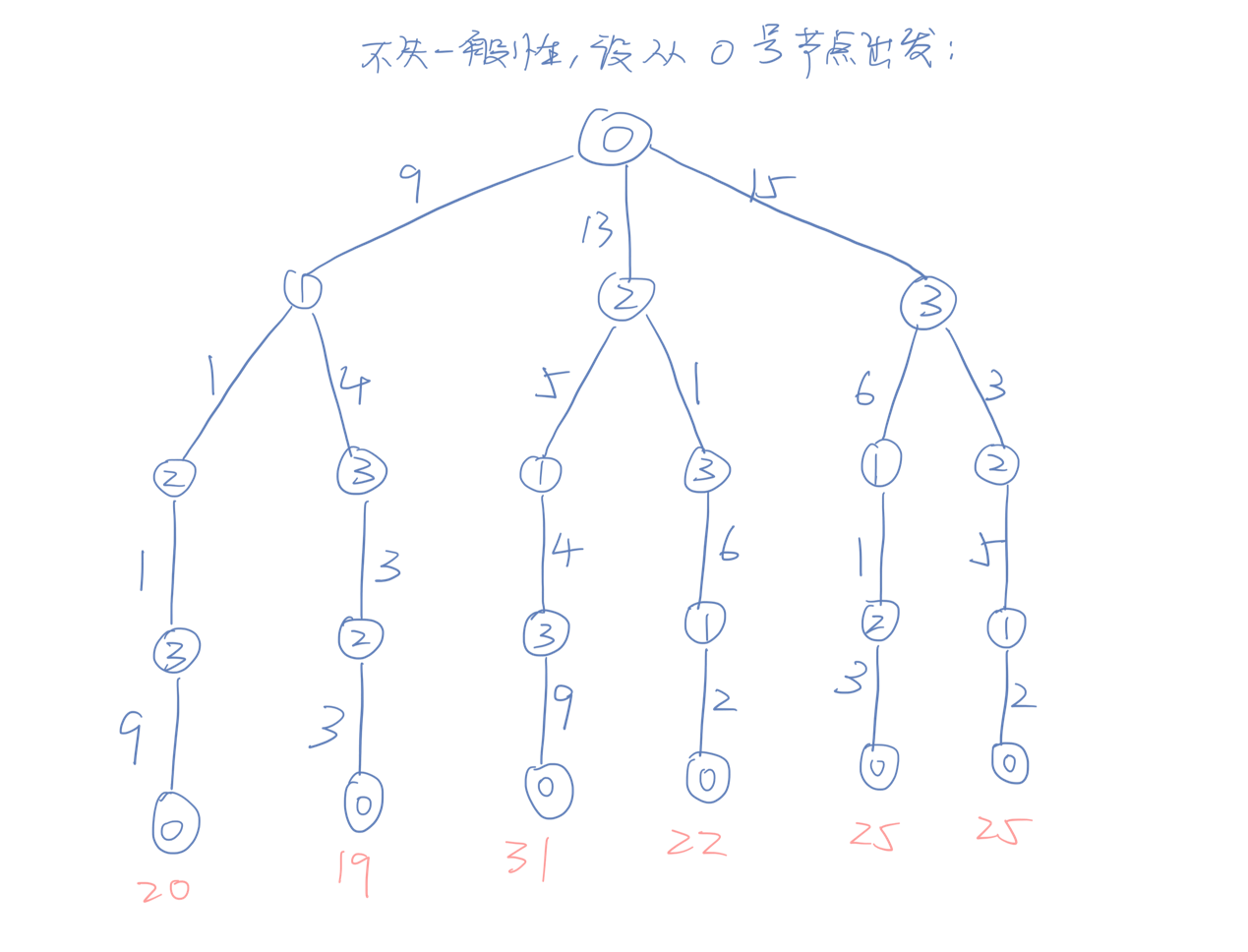
****

可用使用A\*算法求这个图的旅行商问题。

(1)请写出你的g(n)和h\*(n)的定义。



(2)画出求解此图的搜索树。



1. import heapq
2. import random
3. class A\_star:
4. def \_\_init\_\_(self, map: list[list[float]]) -> None:
5. self.map = map  *# 用二维数组来存储图*
6. self.num = len(map)  *# 有多少个节点*
7. self.all\_bitmap: str = "1" \* self.num  *# 用来记录全部节点的位图*
8. self.path: list = []  *# 用来搜集结果*
9. self.cost: float = float("inf")  *# 代价*
10. self.handle()
11. self.path.reverse()
12. def handle(self):
13. root\_index: int = random.randint(0, self.num - 1)  *# 随机产生一个出发的节点*
14. *# root\_index = 0*
15. root\_node: node = node(self.map, root\_index, "0" \* self.num, 0, None)
16. my\_heap: list[node] = []
17. heapq.heappush(my\_heap, root\_node)
18. epoch: int = 0
19. while len(my\_heap) != 0:
20. cur\_node: node = heapq.heappop(my\_heap)
21. cur\_index: int = cur\_node.index
22. if cur\_node.bitmap == self.all\_bitmap:
23. if epoch >= 10:
24. return
25. else:
26. epoch += 1
27. temp\_cost: float = cur\_node.g + self.map[cur\_index][root\_index]
28. *# 到叶节点了, 搜集结果*
29. if self.cost > temp\_cost:
30. del self.path
31. self.path: int = []
32. self.cost = temp\_cost
33. self.path.append(root\_index)
34. self.path.append(cur\_index)
35. parent\_node: node = cur\_node.parent
36. while parent\_node != None:
37. parent\_index = parent\_node.index
38. self.path.append(parent\_index)
39. parent\_node = parent\_node.parent
40. else:
41. rest\_index: list[int] = [
42. c for c in range(self.num) if cur\_node.bitmap[c] == "0"
43. ]  *# 剩余的下标*
44. *# 没有到叶子节点*
45. for next\_index in rest\_index:
46. next\_node\_g: float = self.map[cur\_index][next\_index] + cur\_node.g
47. next\_node: node = node(
48. self.map, next\_index, cur\_node.bitmap, next\_node\_g, cur\_node
49. )
50. heapq.heappush(my\_heap, next\_node)  *# 插入堆中*
51. class node:
52. *# 搜索树上的节点的结构体*
53. def \_\_init\_\_(
54. self, map: list[list[float]], index: int, bitmap: str, g: float, parent
55. ) -> None:
56. self.index = index  *# 当前节点的下标*
57. bitmap\_list = list(bitmap)
58. bitmap\_list[self.index] = "1"
59. self.bitmap: str = "".join(bitmap\_list)  *# 位图, 用来存储已经遍历的节点(包含当前节点)*
60. self.g: float = g  *# 已有代价*
61. self.parent = parent  *# 搜索树的, 当前节点的父节点(谁产生他的)*
62. self.num = len(self.bitmap)  *# 有多少节点*
63. self.f = self.get\_f(map)  *# f = g + h*
64. def \_\_lt\_\_(self, other):
65. return self.f < other.f
66. def get\_f(self, map: list[list[float]]):
67. rest = [c for c in range(self.num) if self.bitmap[c] == "0"]  *# 没有被访问过节点*
68. arr = map[self.index]
69. h = float("inf")
70. for i in rest:
71. if arr[i] < h:
72. h = arr[i]
73. return self.g + h + len(rest) - 1
74. if \_\_name\_\_ == "\_\_main\_\_":
75. test\_map = [
76. [float("inf"), 10, 15, 20, 25],
77. [10, float("inf"), 35, 25, 20],
78. [15, 35, float("inf"), 30, 10],
79. [20, 25, 30, float("inf"), 50],
80. [25, 20, 10, 50, float("inf")]
81. ]
82. test\_a\_star = A\_star(test\_map)
83. print(test\_a\_star.cost)
84. print(test\_a\_star.path)

测试了一下，对于5个节点的完全图，差不多可以在10次epoch，也就是10次到达叶子节点的时候，得到可能是最优解