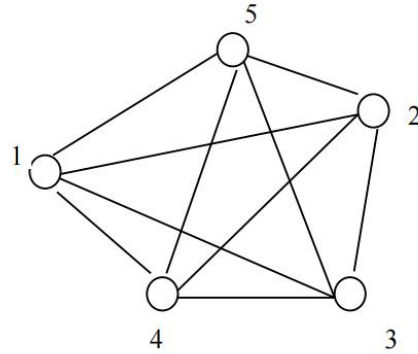


## 算法分析第七章习题

1. 假设对称旅行商问题的邻接矩阵如图 1 所示，试用优先队列式分枝限界算法给出最短环游。画出状态空间树的搜索图，并说明搜索过程。

$$\begin{pmatrix} \infty & 20 & 30 & 10 & 11 \\ & \infty & 16 & 4 & 2 \\ & & \infty & 6 & 7 \\ & & & \infty & 12 \\ & & & & \infty \end{pmatrix}$$

邻接矩阵



旅行商问题

程序见 Travel-PriorityHeap.py

#OUTPUT: 45 [[0, 3, 2, 4, 1, 0], [0, 3, 2, 1, 4, 0], [0, 1, 4, 2, 3, 0], [0, 4, 1, 2, 3, 0]]

搜索过程:

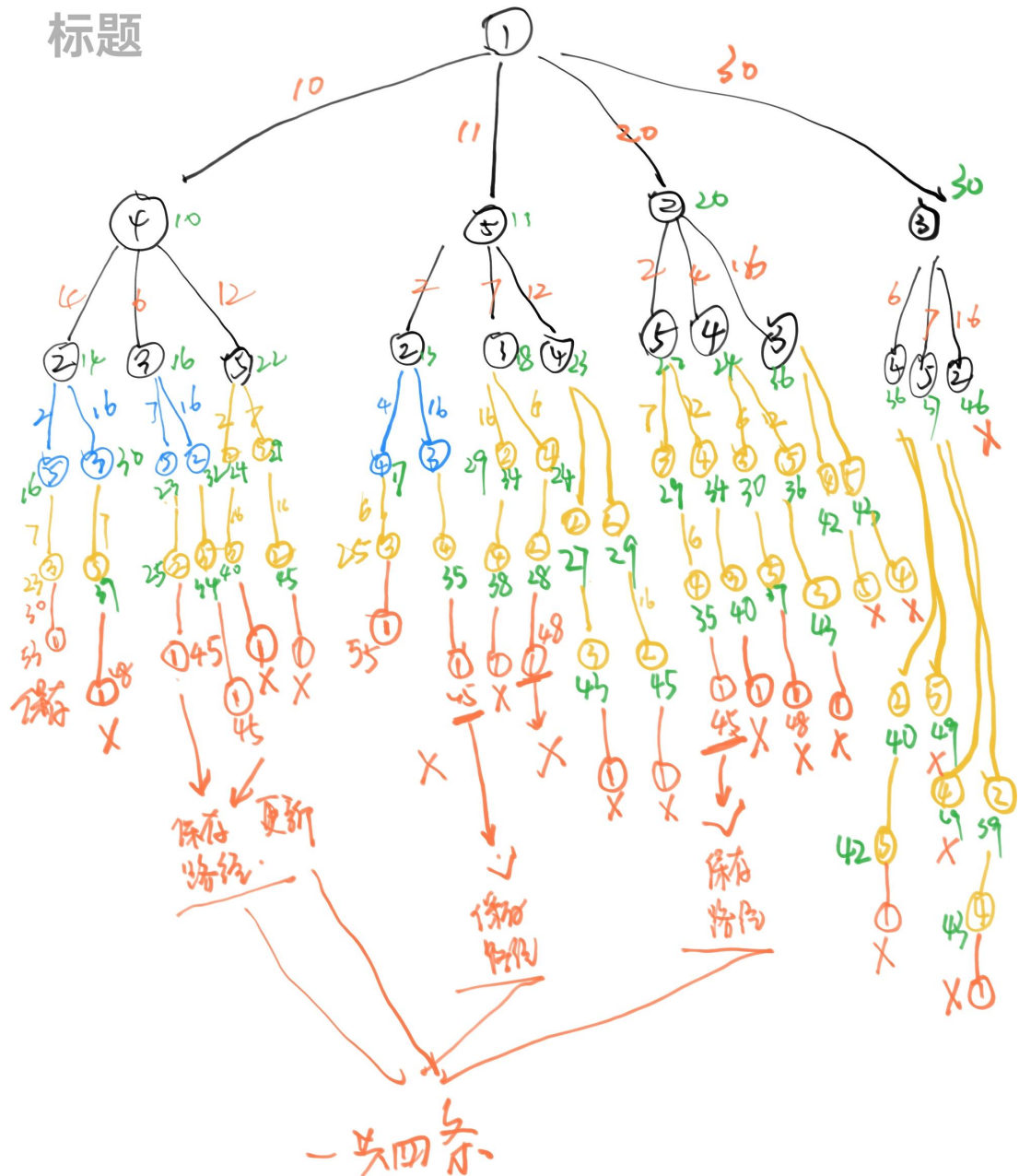
注: 这里对于最后返回起点是进行直接判断的, 当 visited\_cities 达到 n 时, 进行比较判断 (current\_node.node, current\_node.visited\_cities, current\_node.distance)

```
0 [0] 0
3 [0, 3] 10
4 [0, 4] 11
1 [0, 4, 1] 13
1 [0, 3, 1] 14
2 [0, 3, 2] 16
4 [0, 3, 1, 4] 16
3 [0, 4, 1, 3] 17
2 [0, 4, 2] 18
1 [0, 1] 20
4 [0, 3, 4] 22
4 [0, 1, 4] 22
2 [0, 3, 1, 4, 2] 23
2 [0, 4, 1, 3, 2] 23
4 [0, 3, 2, 4] 23
3 [0, 4, 3] 23
1 [0, 3, 4, 1] 24
3 [0, 1, 3] 24
3 [0, 4, 2, 3] 24
1 [0, 3, 2, 4, 1] 25
1 [0, 4, 3, 1] 27
1 [0, 4, 2, 3, 1] 28
2 [0, 4, 3, 2] 29
```

2 [0, 3, 4, 2] 29  
2 [0, 1, 4, 2] 29  
2 [0, 4, 1, 2] 29  
2 [0, 1, 3, 2] 30  
2 [0, 2] 30  
2 [0, 3, 1, 2] 30  
1 [0, 3, 2, 1] 32  
1 [0, 4, 2, 1] 34  
3 [0, 1, 4, 3] 34  
4 [0, 3, 2, 1, 4] 34  
3 [0, 1, 4, 2, 3] 35  
3 [0, 4, 1, 2, 3] 35  
3 [0, 2, 3] 36  
2 [0, 1, 2] 36  
4 [0, 1, 3, 4] 36  
4 [0, 1, 3, 2, 4] 37  
4 [0, 2, 4] 37  
4 [0, 3, 1, 2, 4] 37  
3 [0, 4, 2, 1, 3] 38  
1 [0, 2, 4, 1] 39  
2 [0, 1, 4, 3, 2] 40  
1 [0, 2, 3, 1] 40  
2 [0, 3, 4, 1, 2] 40  
3 [0, 1, 2, 3] 42  
4 [0, 2, 3, 1, 4] 42  
3 [0, 2, 4, 1, 3] 43  
2 [0, 4, 3, 1, 2] 43  
2 [0, 1, 3, 4, 2] 43  
4 [0, 1, 2, 4] 43  
1 [0, 3, 4, 2, 1] 45  
1 [0, 4, 3, 2, 1] 45  
1 [0, 2, 1] 46  
4 [0, 2, 3, 4] 48  
3 [0, 2, 4, 3] 49  
4 [0, 1, 2, 3, 4] 54  
3 [0, 1, 2, 4, 3] 55

搜索图见下图:

## 标题



2. 试写出 0/1 背包问题的优先队列式分枝限界算法程序，并找一个物品个数至少是 16 的例子检验程序的运行情况。

The screenshot shows a Python IDE with several tabs: WorkAssignment.py, Travel-PriorityHeap.py, 01Bag-PriorityHeap.py (active), Huffman.py, and MinWeightMechine-DFS.py. The active tab contains the following code:

```
82 if __name__ == "__main__":
83     item_num = 20
84     bag_capacity = 500
85
86     item_prices = [random.randint(1, 100) for _ in range(item_num)]
87     item_capacities = [random.randint(1, 600) for _ in range(item_num)]
88
89
90     print("Item prices:", item_prices)
91     print("Item capacities:", item_capacities)
92     knap = Knap(item_prices, item_capacities, bag_capacity, item_num)
93     knap.solve()
94     knap.print_result()
95
```

The terminal output shows the execution of the code:

```
PS D:\研究生\研一上课程\12-26算法分析与设计\作业程序> & D:/ProgramFilesFolder/05-Anaconda3/python.exe d:/研究生/研一上课程/12-26算法分析与设计/作业程序/Travel-PriorityHeap.py
45 [0, 3, 2, 4, 1, 0]
PS D:\研究生\研一上课程\12-26算法分析与设计\作业程序> & D:/ProgramFilesFolder/05-Anaconda3/python.exe d:/研究生/研一上课程/12-26算法分析与设计/作业程序/Travel-PriorityHeap.py
45 [[0, 3, 2, 4, 1, 0], [0, 3, 2, 1, 4, 0], [0, 1, 4, 2, 3, 0], [0, 4, 1, 2, 3, 0]]
PS D:\研究生\研一上课程\12-26算法分析与设计\作业程序> cd d:/研究生/研一上课程/12-26算法分析与设计/作业程序
PS D:\研究生\研一上课程\12-26算法分析与设计\作业程序> & D:/ProgramFilesFolder/05-Anaconda3/python.exe d:/研究生/研一上课程/12-26算法分析与设计/作业程序/01Bag-PriorityHeap.py
Maximum profit: 402
Solution vector: [1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0]
PS D:\研究生\研一上课程\12-26算法分析与设计\作业程序> cd d:/研究生/研一上课程/12-26算法分析与设计/作业程序
PS D:\研究生\研一上课程\12-26算法分析与设计\作业程序> & D:/ProgramFilesFolder/05-Anaconda3/python.exe d:/研究生/研一上课程/12-26算法分析与设计/作业程序/01Bag-PriorityHeap.py
Item prices: [91, 18, 49, 66, 83, 54, 35, 94, 27, 67, 32, 92, 86, 76, 77, 33, 18, 20, 71, 16]
Item capacities: [411, 593, 111, 540, 9, 584, 441, 361, 396, 534, 244, 124, 223, 461, 540, 354, 392, 176, 248, 592]
Maximum profit: 310
Solution vector: [0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0]
PS D:\研究生\研一上课程\12-26算法分析与设计\作业程序> |
```

3. 最佳调度问题:假设有  $n$  个任务要由  $k$  个可并行工作的机器来完成, 完成任务  $i$  需要的时间为  $t_i$ 。试设计一个分枝限界算法, 找出完成这  $n$  个任务的最佳调度, 使得完成全部任务的时间 (从机器开始加工任务到最后停机的时间) 最短。

这里有个问题, 就是任务  $i$  在不同的机器上的完成时间是相同的吗?

看着应该是相同的, 如果不同的话对于任务  $i$  应当有  $t_i \cdot k$ , 算法见

WorkAssignment-PriorityHeap.py

每个节点表示一个任务的分配策略, 并且通过对比节点的最短可能完成时间来决定是否继续扩展该节点。算法采用 优先级队列 (最小堆) 来存储和扩展节点, 以保证每次扩展的都是当前可能最优的节点。

完成时间不同的也有一个: WorkAssignment.py