算法导论 课程设计

成绩	

题　　目：回溯算法

学院班级：1613012

学　　号：16130120191

姓　　名：罗阳豪

主讲教师：覃桂敏

日　　期：2019 年 05 月 30 日

目录

题目一　0-1 背包问题（回溯法）

1.1 实验题目

背包问题。5 个物品的价值和重量列表如下，背包最多可以装下 100 磅。请使用回溯算法解决 0-1 背包问题，并尝试画出生成树。

	1	2	3	4	5
value (US $)	20	30	65	40	60
weight (Lbs)	10	20	30	40	50
value / weight	2	1.5	2.1	1	1.2

1.2 实验目的

认识回溯算法的思想，了解回溯算法解决问题的一般过程。通过经典的 0-1 背包问题理解回溯算法的实现，了解回溯算法与一般的遍历法的区别。

1.3 实验设计与分析

对于背包问题，我们要解决的就是一个物品放入背包还是不放入。对于每一个物品，我们都可以选择放入或者不放入，这样，可以产生一棵决策树，我们要做的就是沿着决策树的每一个树枝走到最底层的叶子（代表一个完整的解），找到这些解当中最优的一个（总价值最大）。

其中这里有大量解是无效的，因为他们超重了，而且幸运的是他们通常不用走到最底层就能被发现，这时只要退出来，选择另一条枝干继续走下去即可。这也是回溯算法跟一般的遍历法最大的区别，因为决策不是直接产生的，而是一步一步产生的，每一步都会影响后面的决策，可以在产生大量无用结果之前就发现决策的荒谬之处并及时停止或绕道，避免大量无意义的判断。

在这个具体问题中，这棵决策树实际上是一棵完全二叉树，因为每一个物品只能装上或者不装上两种可能，我们从根节点以先根序往下遍历，用一个栈来记录从根到当前节点的路径，以便回溯。

找到所有可能解，同时记录并返回其中最优的一个。

这里有一个很有意义的优化，实际上就功能实现来说，先装（或不装）哪个物品并不重要。但是如果我们把决策的顺序从物品重量由小到大排序，这样，如果走到某一步，发现无法装下一个物品，那么其后的物品肯定都无法装入（因为他们比当前的更重），这样可以节省很多宝贵的时间。

1.4 实验环境

Python 3.7.0

1.5 项目测试

将上述测试用例作为输入，得到如下结果

最优解：

(155, [0, 1, 2, 3])

表示最优解装下了总价值 155 的物品，装下的是第 0， 1， 2， 3 号物品

生成树：

由于画一棵树相当复杂，所以就只列出节点序列吧

```
(155, [True, True, True, True])
(115, [True, True, True, False])
(90, [True, True, False, True])
(110, [True, True, False, False, True])
(50, [True, True, False, False, False])
(125, [True, False, True, True])
(145, [True, False, True, False, True])
(85, [True, False, True, False, False])
(120, [True, False, False, True, True])
(60, [True, False, False, True, False])
(80, [True, False, False, False, True])
(20, [True, False, False, False, False])
(135, [False, True, True, True])
(155, [False, True, True, False, True])
(95, [False, True, True, False, False])
(70, [False, True, False, True])
(90, [False, True, False, False, True])
(30, [False, True, False, False, False])
(105, [False, False, True, True])
(125, [False, False, True, False, True])
(65, [False, False, True, False, False])
(100, [False, False, False, True, True])
(40, [False, False, False, True, False])
(60, [False, False, False, False, True])
(0, [False, False, False, False, False])
```

题目二 8 皇后问题

2.1 实验题目

使用回溯算法解决 8 皇后问题。

2.2 实验目的

认识回溯算法的思想，了解回溯算法解决问题的一般过程。通过经典的 8 皇后问题理解回溯算法的实现，了解回溯算法与一般的遍历法的区别。

2.3 实验设计与分析

8 皇后问题非常经典，如果采用一般的遍历法，n 皇后问题的时间复杂度高达 O(n^n)，这是很难让人接受的。但是如果采用逻辑类似，只是更科学的“回溯”算法可以极大减少开销。

对于 n 皇后问题，初始化一个 n * n 的矩阵作为棋盘，在左上角摆上一个皇后。由于皇后每行只能有一个，所以我们逐行填充皇后。每填充一个皇后，后面能填充皇后的位置就会减少（因为皇后的攻击规则），每次都在能放皇后的最左侧放一个皇后。如果能够填充到最后一行，那么记录这么一种解法。最下面的皇后往可以下皇后的右侧移动一次。无法移动或无法放下皇后时就往上撤回一个皇后，从下一个（右侧）可能的位置再尝试。类似暴力的遍历，每次都将最下面的往右移动，移到最右侧之后归 0，上一行往右移一次，只不过现在每次放置一个皇后时都先考虑自己能不能放下，而不是到最后一行确认后再做整体的判断。

2.4 实验环境

Python 3.7.0

2.5 项目测试

对于 8 皇后问题，程序得到的解法是 92 个，由于解法数量较大，而且没有什么意义，所以就不列出了