**实验三：贪心算法设计**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 学号 | | 202018526 | 姓名 | 高树林 | 成绩 |  |
| 友情提示 | 1. 算法描述及代码实现与网络或他人雷同者，均按0分计算；  2. 要求算法描述明确、代码清晰、格式美观；  3. 纸质版与电子版同时提交（电子版命名格式：完整学号-姓名-实验X-实验名称，其中,不得省略“-”，如：2088166-乔峰-实验1-分治与递归策略）；   1. 电子版中代码需格式化处理，方便查看（http://www.codeinword.com/）。 | | | | | |
| 实验目的 | 1. 掌握贪心法解决问题的一般步骤。。 2. 通过设计与实现贪心算法求解给定问题，学会使用贪心法解决实际问题。 | | | | | |
| 实验内容 | 1. 均分纸牌问题：有堆纸牌，编号分别为1，2……3，每堆上有若干张，但纸牌总数必为的倍数。可以在任一堆上取若干张纸牌，然后移动。移牌的规则为：在编号为1上取的纸牌，只能移到编号为2的堆上；在编号为的堆上取的纸牌，只能移到编号为的堆上；其他堆上取的纸牌，可以移到相邻左边或右边的堆上。现在要求找出一种移动方法，用最少的移动次数使每堆上纸牌数都一样多。 2. 线段覆盖问题：在一维空间中存在条线段，每条线段的起始坐标与终止坐标已知，要求求出这些线段一共覆盖了多大的长度。 | | | | | |
| 算法描述 | 1. 均分纸牌问题解题思路或算法思想   根据题意，每堆纸牌最终一定能都达到平均纸牌数。那么，可以先根据总纸牌数 sum 求出平均纸牌数ave。然后，针对每一堆纸牌，将该堆的纸牌数与平均纸牌数进行比较，相等时就不需要操作，不相等就要 移动 a[i]-ave 个纸牌（此处包括移走或移来），操作数加 1 | | | | | |
| 1. 线段覆盖问题解题思路或算法思想   对于线段覆盖问题，首先要解决的是线段的端点问题，因此必须首先按照左端点出现的先后顺序排列。之后选择一个标记point[0]和一个point[1]作为线段的左右端点，以便后续刷新。此外，令第一条线段的长度为length，length在后续的判断中会增加相应长度，最终能求出最大长度。 | | | | | |
| 程序及运行结果  （附截图） | 1. 均分纸牌问题  代码：   1. **def** Findways(poker,n): 2. ave = sum(poker) // len(poker) 3. cnt = 0 4. **for** i **in** range(1, n): 5. **if** poker[i] == ave: 6. **continue** 7. poker[i + 1] = poker[i + 1] + poker[i] - ave 8. cnt += 1 9. **return** cnt  12. **if** \_\_name\_\_ == '\_\_main\_\_': 13. n = int(input('输入几堆纸牌：')) 14. poker = list(map(int,input('输入每堆纸牌数：').split(' ')))+[0]\*100 15. **print**('要使纸牌一样多，利用贪心算法移动次数最少为%d次'%Findways(poker,n))   截图： | | | | | |
| 1. 线段覆盖问题   代码：   1. **def** Findways(n, a): 2. **for** i **in** range(0, n):  # 将左端点排好序 3. **for** j **in** range(n - 1 - i): 4. **if** a[0][j] > a[0][j + 1]: 5. temp = a[1][j] 6. a[1][j] = a[1][j + 1] 7. a[1][j + 1] = temp 9. temp = a[0][j] 10. a[0][j] = a[0][j + 1] 11. a[0][j + 1] = temp 12. length = a[1][0] - a[0][0] 13. point = [0, 0] 14. point[0] = a[0][0] 15. point[1] = a[1][0] 16. **for** i **in** range(n): 17. **if** a[0][i] >= point[1]: 18. temp = (a[1][i] - a[0][i]) 19. length += temp 20. point[0] = a[0][i] 21. point[1] = a[1][i] 22. **if** a[0][i] < point[1]: 23. **if** a[1][i] > point[1]: 24. tmp = (a[1][i] - point[1]) 25. length += tmp 26. point[0] = a[0][i] 27. point[1] = a[1][i] 28. **return** length  31. **if** \_\_name\_\_ == '\_\_main\_\_': 32. n = int(input('一共有几条线段(n<100)：')) 33. a = [[0] \* n **for** \_ **in** range(2)] 34. **for** i **in** range(n): 35. a[0][i], a[1][i] = input("第%d个点坐标x,y:" % (i + 1)).split(' ') 36. a[0][i] = int(a[0][i]) 37. a[1][i] = int(a[1][i]) 38. **print**(Findways(n, a))   截图： | | | | | |
| 总结 | 贪心算法在几个基本算法是相对简单的算法了，思路也简单，每一步都能做出当前最好的选择。对于贪心算法，最重要的就是找到每次的局部最优解，而动态规划的关键在于找到状态转移方程。贪心算法又称为贪婪法，是用来寻找最优解问题的常用方法。与动态规划不同的是，贪心算法在求解问题时，总是选择对于当前子问题最好的选择。也就是贪心算法的本质是每次只顾眼前利益，并且到最后能获得最大利益。  我对贪心算法的学习一直在路上，过程也付出了努力，有时不是很懂贪心算法的思想时，加上过程也很艰难，自己也想过放弃，但是老师鼓舞人心的话语让我打消了这个念头，再次对自己充满毅力，坚信自己付出了时间和努力，一定会走到最后。在老师布置贪心算法的作业时，我开始很茫然，不停地看着老师的PPT例题讲解，翻看资料书一些例题理解它的思想，也搜过好些代码，慢慢总结规律，自己总算琢磨出贪心算法的思想以及它的思路，对它的限制和不足也有所了解，对于老师布置的作业，自己也总算A掉了几个题。学习贪心算法的过程，几乎都是在琢磨路上，不断翻看资料，借阅优秀的代码，到最后总算熟悉掌握了它的思路。  个人遗憾：感觉自己还是不够努力，花在贪心算法的时间和精力感觉不足，不是很多，过程虽然有点艰难，自己也不会轻易放弃。贪心算法，我一直在路上，程序设计，我也一直在路上。 | | | | | |