222018321062006 宋行健

# 算法实验题4-1 会场安排问题

## 源代码

1. #include <iostream>
2. #include <algorithm>
3. **using** **namespace** std;
5. **int** main()
6. {
7. **int** n;
8. cin >> n;
9. **int** startTime[10005],endTime[10005];
10. **for**(**int** i = 0; i < n; i++) {
11. cin >> startTime[i] >> endTime[i];
12. }
13. sort(startTime,startTime+n);
14. sort(endTime,endTime+n);
15. **int** room = 0;
16. **int** j = 0;
17. **for**(**int** i = 0; i < n; ++i) {
18. **if**(startTime[i] < endTime[j]) room++;
19. **else** j++;
20. }
21. cout << room << endl;
22. **return** 0;
23. }

## 程序运行截图



图 1 会场问题运行结果截图

## 实验心得

解决本问题在于贪心算法的核心思想。按照开始时间排序，来了一个活动就把它做了，如果没有空的会场就增加一个会场（room++），过程中维护一下会场的状态。这里实际维护了这样一个东西——在开的所有会场中，里面正在进行的会议结束时间最早的时间（endTime[j]）。

具体做法是把开始时间和结束时间分别装在两个数组中, startTime[] 和 endTime[]，从早到晚排序，循环遍历开始时间startTime[i]，设置位置标记j, 如果startTime[i] < endTime[j]，说明当前没有空会场可用了，要新开一个会场，room++；否则j++，表示已经有会场能空出来了，更新当前所有会场结束的最早时间。

# PPT补充题目：贪心算法实现最少硬币找零问题

## 源代码

1. #include <iostream>
2. #include <fstream>
3. #include <algorithm>
4. **using** **namespace** std;
6. **int** Coins[6];                                      // 各面值硬币数
7. **int** CoinNum = 0;                                   // 需要的最少硬币数
8. **int** Pay;                                           // 需要支付的钱数
9. **int** CoinValue[6] = {5, 10, 20, 50, 100, 200};      // 硬币面值，以分为单位
10. **int** ChangeValue[7] = {0, 5, 10, 20, 50, 100, 200}; // 用来找零的硬币面值，0为不找零的情况
11. **int** RealMoney = CoinValue[0] - ChangeValue[0];     // 实际支付金额 = 支付硬币 - 找零硬币 【贪心变量】
12. ifstream input("input.txt");
13. ofstream output("output.txt");
15. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
16. \* 函数描述： 从文件中读取测试数据
17. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
18. **void** getData()
19. {
20. **for** (**int** i = 0; i < 6; ++i)
21. input >> Coins[i];
22. **double** costt;
23. input >> costt;
24. cout << "\n目标金额：" << costt << "元"
25. << "\n各币值数量：" << endl;
26. Pay = (**int**)(costt \* 100); //将输入的钱转为分为单位
27. **for** (**int** i = 0; i < 6; ++i)
28. cout << Coins[i] << " ";
29. }
31. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
32. \* 函数描述： 格式化输出结果
33. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
34. **void** outputResult()
35. {
36. **if** (CoinNum == 0 || Pay != 0)
37. {
38. cout << "\n============== impossible" << endl;
39. output << "impossible" << endl;
40. }
41. **else**
42. {
43. cout << endl
44. << "各币值剩余数量:" << endl;
45. **for** (**int** i = 0; i < 6; i++)
46. cout << Coins[i] << " ";
47. cout << endl
48. << "============== 支付和找零需要的最少总硬币数量：" << CoinNum << endl;
49. output << CoinNum << endl;
50. }
51. }
53. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
54. \* 函数描述：查看顾客手中是否还有 a 面值的硬币，
55. \*          如果有则返回面值索引，否则返回 -1
56. \* 函数返回：面值索引 or -1
57. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
58. **int** contains(**int** a)
59. {
60. **for** (**int** i = 0; i < 6; ++i)
61. **if** (CoinValue[i] == a && Coins[i] > 0)
62. **return** i;
63. **return** -1;
64. }
66. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
67. \* 函数描述： 贪心法实现硬币找零问题，以实际支付金额作为贪心变量
68. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
69. **void** Greed()
70. {
71. // 顾客手中的硬币面值从大到小遍历选取
72. **for** (**int** i = 5; i >= 0; --i)
73. {
74. **if** (Coins[i] > 0) // 如果顾客手中当前面值硬币有剩余
75. {
76. // 商家手中的硬币从小到大遍历选取，进行找零操作
77. **for** (**int** j = 0; j <= i; ++j)
78. {
79. RealMoney = CoinValue[i] - ChangeValue[j]; // 实际支付金额 = 支付硬币 - 找零硬币
80. // 只有当实际支付金额小于目标金额的时候才进行选取，否则需要增加找零硬币的金额（即，继续本层循环 j）
81. **if** (Pay >= RealMoney)
82. {
83. **if** (Coins[i] >= Pay / RealMoney) //如果当前面值硬币找零后足够支付余额
84. {
85. **int** TempCoinNum = Pay / RealMoney; // 此步骤用户消耗硬币数量
86. CoinNum += TempCoinNum \* 2;
87. // 如果顾客具有该硬币，则无需使用找零，因为找零需要耗费2个硬币，而支付只需要一个硬币
88. **if** (contains(RealMoney) != -1)
89. {
90. TempCoinNum = min(TempCoinNum, Coins[contains(RealMoney)]);
91. CoinNum -= TempCoinNum;
92. Coins[contains(RealMoney)] -= TempCoinNum; // 顾客手中相应硬币数减少
93. }
94. **else**
95. Coins[i] -= Pay / RealMoney; // 顾客手中当前面值硬币数减少
96. Pay = Pay % RealMoney;           // 更新还需支付的金额
97. **if** (contains(CoinValue[i]) == -1)
98. **break**; // 如果顾客手中该硬币用完，直接用下一个小面值
99. }
100. **else** // 如果当前币值不够支付，则用完该币值
101. {
102. CoinNum += Coins[i];
103. Pay = Pay - CoinValue[i] \* Coins[i];
104. Coins[i] = 0;
105. }
106. }
107. }
108. }
109. }
110. }
112. **int** main()
113. {
114. **while** (!input.eof())
115. {
116. CoinNum = 0;
117. getData();      // 读取测试数据
118. Greed();        // 贪心法实现硬币找零问题
119. outputResult(); // 输出结果
120. }
121. input.close();
122. output.close();
123. }

## 程序运行截图



图 2 最少硬币问题运行结果截图

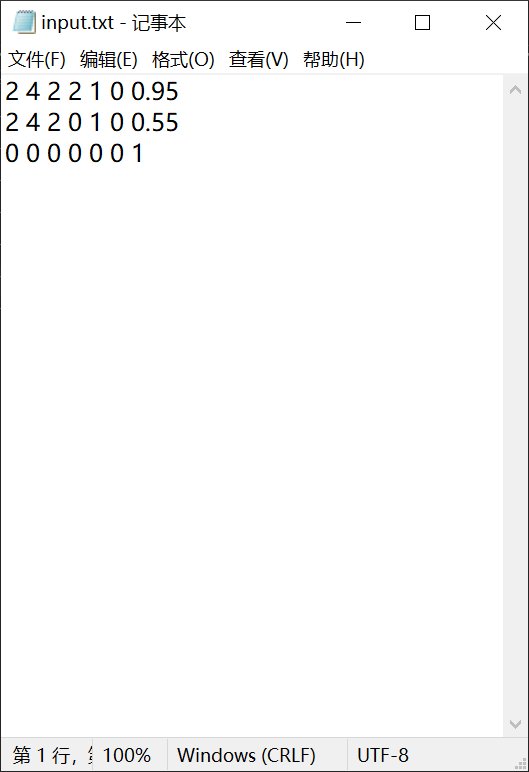


图 3 input.txt输入文件截图

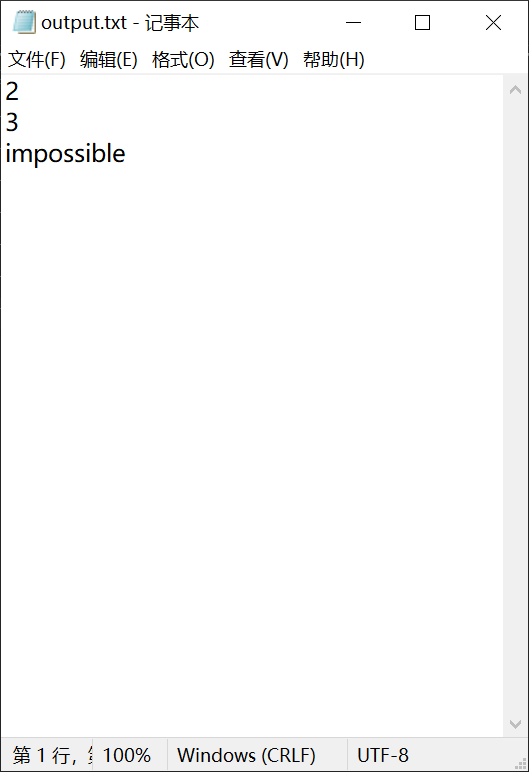


图 4 output.txt输出文件截图

## 实验心得

解决本问题在于贪心变量的找取。原始的硬币问题的贪心变量很直观，就是硬币从大到小选取。而加上了找零的操作后，贪心变量就不那么直观了。本问题中的贪心变量是每个硬币的实际支付金额，实际支付金额 = 支付硬币 - 找零硬币。加上找零的本质就是通过找零的排列组合，将硬币的面值变多了。

其中有一点需要注意的就是，如果找零的排列组合（当前硬币的实际支付金额）存在等面值的真实硬币，则不用找零的操作，直接取硬币，因为找零需要耗费2枚硬币，而支付只需要一枚硬币。