《面向对象程序设计与训练》实验报告

信息 学院 计算机科学与技术 专业 19 级

实验时间 2020 年 12 月 7 日

姓名 白文强 学号 20191060064

实验名称 CSP认证2018年9月测试真题

实验成绩

一、实验目的

1. 理解Java语言是如何体现面向对象编程基本思想，
2. 了解类的封装方法，以及如何创建类和对象，
3. 了解成员变量和成员方法的特性。

了解类的继承性和多态性的作用。

二、实验仪器设备及软件

个人PC， IDEA

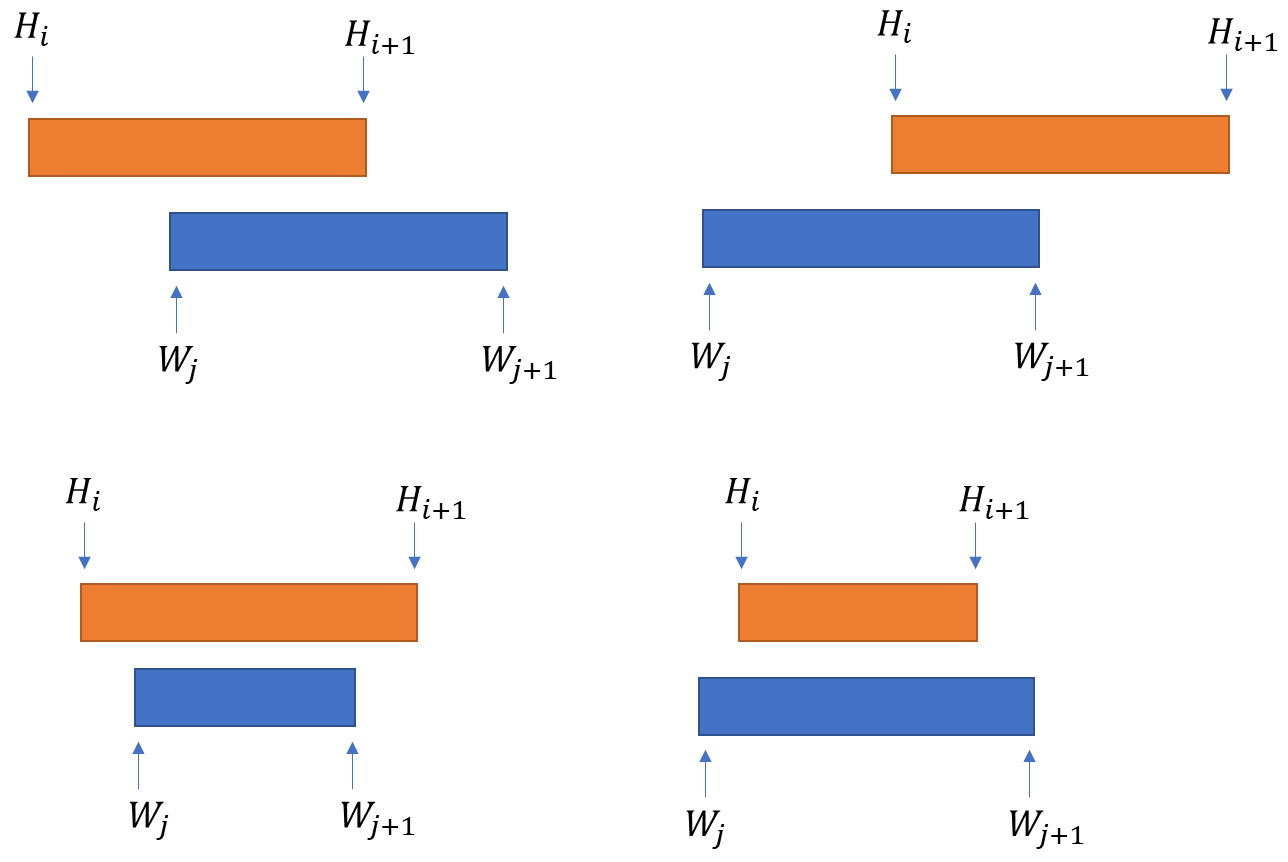
三、实验方案

**卖菜：**

此题比较简单，只需要注意计算第二天菜价时需要用一个新数组存放结果，而不能直接在一个数组中进行操作；此外还需要注意两个边界，计算第一家商户和最后一家商户第二天的价格时，要单独处理。

**买菜：**

此题是计算两个人同时空闲的时间，对于两个人的时间段，如果存在交叉，有以下几种情况：



我采用数组存放一个人的时间段，偶数下标存放开始，奇数下标存放结束，这样就完成了存储的要求；然后，固定H的一个时间段，对W的时间段进行遍历，出现时间段有交叉时，计算重合时间，直到出现大于时，就说明当前H的时间段下，与W重合的时间段计算完毕，退出内层循环，对H的时间段进行迭代。

由此，当对每一个H的时间段遍历都结束后，得出来的重合时间之和就是答案。

**元素选择器：**

Main类中含有三个变量，tag表示标签，id表示id，rank表示元素的等级（rank越大，等级越低，这是根据输入过程中元素前的点个个数计算的）。

对于单级查询，分为查询id和tag两种情况，只需要遍历每个元素，如果查到就记住元素的行数以及计数器更新加一。

对于多级查询，一级一级查找，例如查找html div p，首先查找所有的html，将其存放到集合里，接下来查询div，依次取出集合中的元素，从该元素开始查询div，当遇到等级大于等于取出的元素的等级时，退出循环，取出下一个元素继续向后查询。将查询到的结果放到另一个集合里。每个循环开始时，第一个集合会克隆第二个集合，第二个集合会清空用来记录新查找到的元素。

这样，直到找完最后一个元素，最后集合中的元素中存放的就是要查询的所有元素，至此就完成了查询过程。

四、实验步骤

**卖菜：**

1. **import** java.util.Scanner;
3. **public** **class** Main {
4. **private** **static** Scanner in = **new** Scanner(System.in);
6. **public** **static** **void** main(String[] args) {
7. String[] str = in.nextLine().split(" ");    //读入第一行
8. **int** n = Integer.parseInt(str[0]);
10. **int**[] price = **new** **int**[n];                          //原价格
11. **int**[] newPrice = **new** **int**[n];                       //现价格
13. String[] str1 = in.nextLine().split(" ");   //第二行，每个商店的时菜价
14. **for**(**int** i = 0; i < n; ++i) {
15. price[i] = Integer.parseInt(str1[i]);
16. }
17. StringBuffer result = **new** StringBuffer("");
18. //计算第二天价格
19. **for**(**int** i = 0; i < n; ++i) {
20. **if** (i == 0) {
21. newPrice[i] = (price[i] + price[i + 1]) / 2;
22. } **else** **if** (i == n - 1) {
23. newPrice[i] = (price[i - 1] + price[i]) / 2;
24. } **else** {
25. newPrice[i] = (price[i - 1] + price[i] + price[i + 1]) / 3;
26. }
27. result.append(newPrice[i]).append(" ");
28. }
29. String res = **new** String(result);
30. //去除最后空格输出
31. System.out.println(res.trim());
32. }
33. }

**买菜：**

1. **import** java.util.Scanner;
3. **public** **class** Main {
4. **private** **static** Scanner in = **new** Scanner(System.in);
6. **public** **static** **void** main(String[] args) {
7. String lineFirst = in.nextLine();
8. **int** n = Integer.parseInt(lineFirst);
9. **int**[] HTime = **new** **int**[n \* 2];
10. **int**[] WTime = **new** **int**[n \* 2];
12. //数据读入
13. String[] lineTwo;
14. **for**(**int** i = 0; i < 2 \* n; i += 2) {
15. lineTwo = in.nextLine().split(" ");
16. HTime[i] = Integer.parseInt(lineTwo[0]);
17. HTime[i + 1] = Integer.parseInt(lineTwo[1]);
18. }
19. **for**(**int** i = 0; i < 2 \* n; i += 2) {
20. lineTwo = in.nextLine().split(" ");
21. WTime[i] = Integer.parseInt(lineTwo[0]);
22. WTime[i + 1] = Integer.parseInt(lineTwo[1]);
23. }
24. //计算可聊天的时间
25. **int** sumTime = 0;
26. **for**(**int** i = 0; i < 2 \* n; i += 2) {
27. **for**(**int** j = 0; j < 2 \* n && WTime[j] <= HTime[i + 1]; j += 2) {
28. **if** (HTime[i] <= WTime[j] && HTime[i + 1] >= WTime[j] && WTime[j + 1] >= HTime[i + 1])
29. sumTime += HTime[i + 1] - WTime[j];         //有交叉，W靠后
30. **else** **if** (HTime[i] >= WTime[j] && HTime[i] <= WTime[j + 1] && HTime[i + 1] >= WTime[j + 1])
31. sumTime += WTime[j + 1] - HTime[i];         //有交叉，H靠后
32. **else** **if** (HTime[i] <= WTime[j] && HTime[i + 1] >= WTime[j + 1])
33. sumTime = sumTime + (WTime[j + 1] - WTime[j]);     //W包含H
34. **else** **if** (WTime[j] <= HTime[i] && WTime[j + 1] >= HTime[i + 1])
35. sumTime += HTime[i + 1] - HTime[i];         //H包含W
36. }
37. }
38. System.out.println(sumTime);
39. }
40. }

**元素选择器：**

1. **import** java.util.\*;
3. **public** **class** Main {
4. **private** String tag; //标签
5. **private** String id;  //id
6. **private** **int** rank;   //等级，0 1 2 3 ，数字越高等级越低
8. **private** **static** Scanner in;
10. **public** **static** **void** main(String[] args) {
11. in = **new** Scanner(System.in);
12. List<Main> l = **new** LinkedList<>();          //存放每一个元素
13. List<String> result = **new** ArrayList<>();    //存放输出结果
15. String[] s = in.nextLine().split(" ");
16. **int** n = Integer.parseInt(s[0]);    //标签个数
17. **int** m = Integer.parseInt(s[1]);    //查询次数
19. **int** i;
20. String[] str2;
22. //输入过程
23. **for** (i = 0; i < n; ++i) {
24. str2 = in.nextLine().split(" ");
25. **int** preLength = str2[0].length();
26. str2[0] = str2[0].replace(".", "");
27. **int** afterLength = str2[0].length();
29. Main temp = **new** Main();
30. temp.tag = str2[0].toLowerCase();
31. **if** (str2.length != 1) {
32. temp.id = str2[1];
33. }
34. temp.rank = (preLength - afterLength) / 2;
36. l.add(temp);
37. }
39. //查询过程
40. **for** (i = 0; i < m; ++i) {
41. str2 = in.nextLine().split(" ");
42. StringBuffer outCount = **new** StringBuffer();
43. StringBuilder out = **new** StringBuilder();
45. **int** count = 0;      //查询到的个数
46. **if** (str2.length == 1) {     //只查询tag或者id
47. Main main;
48. **if** (str2[0].charAt(0) == '#') {         //查询id
49. **for** (**int** k = 0; k < n; k++) {
50. main = l.get(k);
51. **if** (str2[0].equals(main.id)) {
52. out.append(k + 1).append(" ");
53. count++;
54. }
55. }
56. } **else** {                                //查询标签
57. **for** (**int** k = 0; k < n; k++) {
58. main = l.get(k);
59. **if** (main.tag.equalsIgnoreCase(str2[0])) {
60. out.append(k + 1).append(" ");
61. count++;
62. }
63. }
64. }
65. } **else** {            //分级查询
66. Set<Integer> buff = **new** HashSet<>();        //临时
67. Set<Integer> res = **new** HashSet<>();         //结果
68. **for** (**int** k = 0; k < n; k++) {
69. **if** (l.get(k).tag.equals(str2[0].toLowerCase()) || str2[0].equals(l.get(k).id)) {
70. res.add(k);                     //查询第一个元素
71. }
72. }
73. **int** start = 1;      //str2[]的索引，开始查询下标为1的元素
74. **while** (start < str2.length) {
75. buff.clear();       //清空集合
76. buff.addAll(res);   //拷贝上次得到的结果
77. res.clear();        //清空上次结果
78. Object[] buf = buff.toArray(); //转数组
79. **for** (Object o : buf) {  //依次取出匹配到的下一级的下标
80. **int** location = (**int**) o;
81. Main main;
82. //找下一级的元素下标，放入res集合中
83. **for** (**int** p = location + 1; p < n && l.get(p).rank > l.get(location).rank; p++) {
84. main = l.get(p);
85. **if** (main.tag.equals(str2[start].toLowerCase()) || str2[start].equals(main.id)) {
86. res.add(p);
87. }
88. }
89. }
90. start++;    //迭代查找下一级
91. }
92. count = res.size();     //结果集合元素的个数就是结果的个数
93. Object[] out2 = res.toArray();
94. **for** (**int** o = 0; o < count; o++) {
95. out.append((**int**) out2[o] + 1).append(" "); //构建找到的下标字符串
96. }
97. }
98. outCount.append(count).append(" ").append(out); //构建结果字符串
99. String newStr = **new** String(outCount);
100. result.add(newStr.trim());      //去掉首尾空格
101. }
102. //输出部分
103. **for** (String newResult : result) {
104. System.out.println(newResult);
105. }
106. }
107. }

五、实验结果及分析







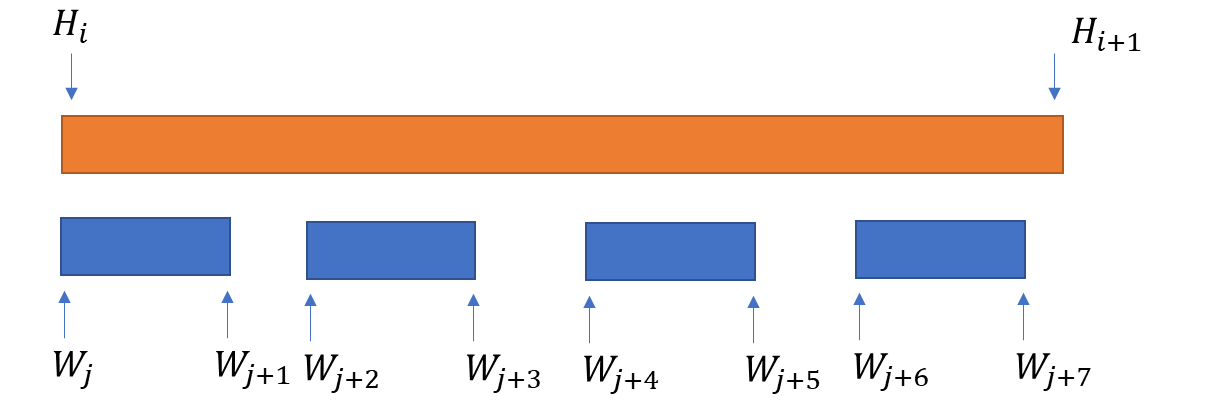
六、实验总结及体会

**卖菜：**

第一次，在一个数组中进行操作，第二天的价格把第一天的价格覆盖，计算后面的商户的第二天的价格时会出现错误。解决方法即采用两个数组进行操作，将计算出的结果放到新数组中。

**买菜：**

此题一开始采用单循环，即一段一段比较，但是会有极端情况的出现：



因此，最终采用双循环的做法，对内层循环做了优化，另外此题还可以采用双指针法完成，不需要每次都从开始。

**元素选择器：**

实现单级查询时，比较简单，当只写完单级查询时，提交得到50分，但是实现多级查询时遇到较大困难，一开始的思路出现较大问题，找到第一个符合条件的元素就会返回，忽略了可能存在多个符合条件的情况。后面改变思路，将每次的结果都存下来，然后查询，但是会出现重复的情况，为了避免出现重复，采用了Set集合，集合的元素不可重复。由此解决了重复的问题。

通过此次实验，对CSP的基本情况有了更深的理解，对题型和要用到的一些知识有了初步掌握。

七、教师评语