Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 2 REPORT

GÖKTUĞ ALİ AKIN
161044018

Course Assistant:Ayşe Şerbetçi Turan

1 INTRODUCTION

1.1 Problem Definition

In this homework, the problem is design a list class that keeps machine learnig experiments and their results. This class keeps experiments with specific properties.

Adding experiments to the list

ExperimentList keep experiments ordered by day. When user of this class, adds new Experiment, this Experiment will be added to true area according the day. This class, keeps the experiments ascending order according to day values of experiments.

Other capabilites of user of ExperimentList class

- Remove an experiment from the list.
- Remove total experiments from the list by giving a day.
- Change the experiment in the list by giving day and index number.
- Order experiments in specific day according to accuracy.
- Order all experiments in the list according to accuracy and get the new list.
- List all completed experiments in the list with specified day.

1.2 System Requirements

Experiment Class

Firstly, in order to keep an experiments in the ExperimentList class, we need an Experiment class that represents experiments with theese properties:

- Setup(definition) of an experiment.
- Day of an experiment.
- Time of an experiment.
- Accuracy value of an experiment.
- Describing experment is completed or not completed.

ExperimentList Class

In order to keep the experiments in the ExperimentList class which is the main purpose of this homework, we need to implement a list structrue based on linked list. Like linkedlist, ExperimentList class have inner private static class named **Node** to represent a **Node** that keeps data which is Experiment and node to the next experiment. This properties provide us to out class can behave like a single linked class.

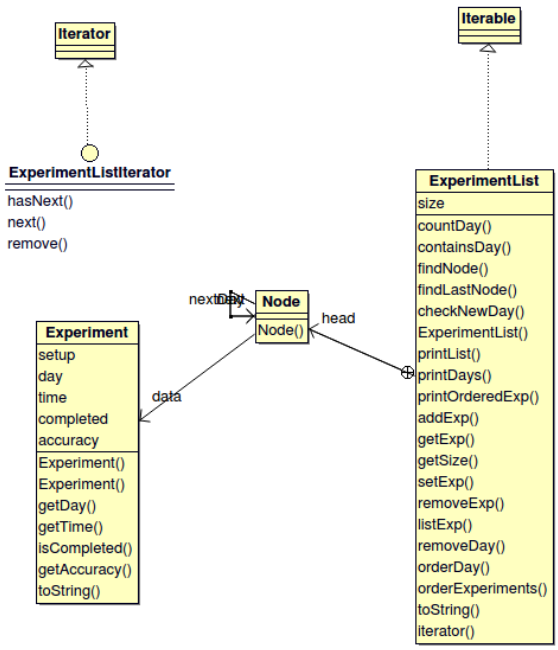
An additional property is, in order to speed up add and remove operations, an additional list structure must be defined for link the days level by level. This structure, provides to jump from a day to another day without traverse experiments which are in the list one by one.

Custom Iterator Class

In order to iterator over the list and use this list in for-each loop statements, ExperimentList class must be *iterable.* Since ExperimentList is iterable, we must provide an iterator method for this class.So a custom we must implement a custom iterator class named *ExperimentListIterator* in implementation.

2 METHOD

2.1 Class Diagrams



2.2 Problem Solution Approach

Time data member in Experiment class

Instead of giving a time value to Experimet class constructor, time value will be automatically gets from the system to an date object, and this date object converted to the string and assigned to instance variable Time(String). So, when an Experiment object created, the time of an experiment will be exact time during constructor invoking.

Linking all experiments together

In order to link all experiments together, private static inner class named *Node* implemented in ExperimentList class. This class provides us to link all experiments together like most of the single

linked lists. Like most of the single linked lists, inner class *Node* has 3 data members. Theese are data(*Experiment*), next(*Node*) and nextDay(*Node*).

Linking experiments according to day

In order to speed up add and remove operations, additional list structure implemented in this list class. In this class, first experiment of a specific day linked other experiment of first day of next day. For example, the first experiment of a 1st day linked to the first experiment of a 2nd day. This property provides us speed. To do this property, an additional node added to the inner class *Node* type *Node* like next data member. This member named nextDay and points to the next day of current day.

Adding a new experiment to the list and construct list structure correctly

We have **addExp()** method to the add experiment to the list.There are two things to consider when adding a new experiment to the list.

1) Linking experiment to the true next node.
2) Linking experiment to the true nextDay node if it is a new day.

1) Linking next node

First, when adding an experiment to the list, I get the day of an experiment which will be added to the list to determine the true space. An experiment can be added to the end of the list, end of the day or between two experiments. Now, I have the day of new experiment. So;

1) If list is empty, add the new experiment and head will be pointed this experiments node.
2) If list is not empty, then I must need to check the given day is a new day in the list. There will be an two different scenarios.
    1. If given day is a new day, I must to find the last node that has a day value that is less than the day of the new day, and greater than the all day values in the list. For example, we have list that contains this days : 1-1-2-2-4-4. And we want to add an experiment which has an day value of 3. In there, the main point is, “Is there a node that has an experiment which has a day value that less than 3 and, greater than all experiments in the list ?”. Yes, in this example, we have a day 2 that is less than 3 and greater than the all days which is less than 3. When during this checking process, nextDay list structure used to speed up the adding method, program not traverse node-by-node, it traverses nextday-by-nextday. Details will be explained below(**2) Linking nextday node**).
    2. If given day is not a new day means that there is day in the list that equals day of new experiment explicitly.

Both of two scenraios, I must to find the true last node to adding. For first scenario, I must find the first node which is in the true space. For second scenario, I must find first the node that equals new experiments day value. So both of two scenarios, we have the first true node.. Then we must to find the last node by the founded first node. In implementation, we have private helper method named

findLastNode(Node node) returns the last occurence of the node that has the same day value.When I found the last node, program is ready to add the newday.

Examples given :

- Scenario 1 new day :

Current list (day values) : 1-2-2-2-4-4. Now we call **addExp(exp3)** to add new experiment which has 3 day value to the list. First, program finds the bigger day value but less than '3' in the list. So program found the experiment which has day value of 2(written blue) and continues to find the last node of experiment which has day value of 2. Now, last occurence of 2 day value founded. And program add the new day to the this last node's next node.(written red). Same algorithm works for adding the end of the list. Ex: adding 3 day value to the this list : (1-2-2-2).

- Scenario 2 not new day:

Current list (day values) : 1-2-2-2**.** Now we call addExp(exp2) to the add new experiment which has day value of '2' to the list. Program checked '2' is not a new day, so program finds the first occurence of day value '2'(written in blue) by nextDay list structure easily. After this, program continues to the find the last occurence of 2 day value(written in red). When program finds the last node, adds the new day to the this last node's next node. Same algorithm works for adding between two experiments. Ex : adding 2 day value to the this list : (1-2-3-3-3).

2) Linking nextDay node

As mentioned above, this structure implemented for speed reasons. Program traverses the nodes not node-by-node, it traverses day-by-day and jump easily to the nextDay if it is exists.

When linking the nextDay node, the new experiment's day must be new day in the list. After passing the new day test(checkNewDay() private helper method), First, program finds the bigger day value but less than 'x' where x is the new experiment's day in the list. After this, we know the fist node and we can easily assign new experiment to this node's nextDay.

Ex : Current list : (1-2-2), adding an experiment which has day value of '3'. Program finds the bigger day value but less than '3'. Now founded (written red). Then assign new node to nextDay node of this node.

These are the main structures of the ExperimentList.

3 RESULT

3.1 Running Results and Complexity of Methods

1) addExp(Experiment e) method

```
{
ExperimentList list1 = new ExperimentList();

for (int i = 0; i < 3; i++)
    list1.addExp(new Experiment( setupVal: "exp4_"+i, dayVal: 4, accuracyVal: 100-i*2.5f, isCompleted: true));
for (int i = 0; i < 3; i++)
    list1.addExp(new Experiment( setupVal: "exp1_"+i, dayVal: 1, accuracyVal: 100-i*2.5f, isCompleted: true));
for (int i = 0; i < 3; i++)
    list1.addExp(new Experiment( setupVal: "exp2_"+i, dayVal: 2, accuracyVal: 100-i*2.5f, isCompleted: true));
for (int i = 0; i < 3; i++)
    list1.addExp(new Experiment( setupVal: "exp3_"+i, dayVal: 3, accuracyVal: 100-i*2.5f, isCompleted: true));

System.out.println("# Test 1 : addExp() method tested.");
System.out.println("adding order for days of experiments is : 4,1,2,3.\nSo, all adding posibilites
```

Adding to the ExperimentList with all posibilities. We expect;

1) First for loop executes and finishes, list1 is : (4,4,4) (according to day).
2) Second for loop executes and finishes, list1 is : (1-1-1-4-4-4).
3) Third for loop executes and finishes, list1 is : (1-1-1-2-2-2-4-4-4)
4) Fourth for loop executed and finishes, list1 is : (1-1-1-2-2-2-3-3-3-4-4-4)

Now lets look up the real result.

```
-----------------------------------------------
# Test 1 : addExp() method tested.
adding order for days of experiments is : 4,1,2,3.
So, all adding posibilites are tested.
Printing the list :
--------------------------
# ExperimentList | Size : 12
--------------------------
Day:1 03:26:20 | exp1_0 | Acc: %100.0 | completed
Day:1 03:26:20 | exp1_1 | Acc: %97.5 | completed
Day:1 03:26:20 | exp1_2 | Acc: %95.0 | completed

Day:2 03:26:20 | exp2_0 | Acc: %100.0 | completed
Day:2 03:26:20 | exp2_1 | Acc: %97.5 | completed
Day:2 03:26:20 | exp2_2 | Acc: %95.0 | completed

Day:3 03:26:20 | exp3_0 | Acc: %100.0 | completed
Day:3 03:26:20 | exp3_1 | Acc: %97.5 | completed
Day:3 03:26:20 | exp3_2 | Acc: %95.0 | completed

Day:4 03:26:20 | exp4_0 | Acc: %100.0 | completed
Day:4 03:26:20 | exp4_1 | Acc: %97.5 | completed
Day:4 03:26:20 | exp4_2 | Acc: %95.0 | completed
--------------------------
# List day view:
Day:1 03:26:20 | exp1_0 | Acc: %100.0 | completed
Day:2 03:26:20 | exp2_0 | Acc: %100.0 | completed
Day:3 03:26:20 | exp3_0 | Acc: %100.0 | completed
Day:4 03:26:20 | exp4_0 | Acc: %100.0 | completed
--------------------------
```

List is correctly printed, nexts and nextDays are correctly constructed.
Time complexity : First loop in the addExp() method, jumps to the nextDay if it is exist. It's time complexity is O(N/x) where x is an constan number that describes number of jumps. Second for loop, continues to the last item, to it's time complexity is O(N). O(N/x) and O(N) gives us to : Time complexity of addExp() method is **O(N)**.

2) orderDay(int day) method

```
System.out.println("# Test 1 Fi
for (int i = 1; i <= 4; i++)
    list1.orderDay(i);
```

By this for loop, all days are ordered with list (current list above in 1. test). Experiments are ordered according to their accuracy. Lets look up the results.

```
--------------------------
# Test 2 : orderDay() method tested.
orderDay(1),orderDay(2),orderDay(3),orderDay(4) executed.
Printing the list :
--------------------------
# ExperimentList | Size : 12
--------------------------
Day:1 03:26:20 | exp1_2 | Acc: %95.0 | completed
Day:1 03:26:20 | exp1_1 | Acc: %97.5 | completed
Day:1 03:26:20 | exp1_0 | Acc: %100.0 | completed

Day:2 03:26:20 | exp2_2 | Acc: %95.0 | completed
Day:2 03:26:20 | exp2_1 | Acc: %97.5 | completed
Day:2 03:26:20 | exp2_0 | Acc: %100.0 | completed

Day:3 03:26:20 | exp3_2 | Acc: %95.0 | completed
Day:3 03:26:20 | exp3_1 | Acc: %97.5 | completed
Day:3 03:26:20 | exp3_0 | Acc: %100.0 | completed

Day:4 03:26:20 | exp4_2 | Acc: %95.0 | completed
Day:4 03:26:20 | exp4_1 | Acc: %97.5 | completed
Day:4 03:26:20 | exp4_0 | Acc: %100.0 | completed
--------------------------
# List day view:
Day:1 03:26:20 | exp1_2 | Acc: %95.0 | completed
Day:2 03:26:20 | exp2_2 | Acc: %95.0 | completed
Day:3 03:26:20 | exp3_2 | Acc: %95.0 | completed
Day:4 03:26:20 | exp4_2 | Acc: %95.0 | completed
--------------------------
```

Experiments are correctly ordered, and by looking the (# List day view) part, we can see nextDay structure not damaged.
Time complexity : There is a nested loop used while sorting the ExperimentList, this for loop executes N’2. So the time complexity is **O(N’2)**.

3) removeExp() method

```
list1.removeExp( day: 1, index: 0);
list1.removeExp( day: 3, index: 0);
list1.removeExp( day: 2, index: 0);
list1.removeExp( day: 4, index: 0);
```

These remove operations executed.(Current list above in 2. test). Results :

```
--------------------------
# Test 3 : removeExp() method tested.
removeExp(1,0),removeExp(2,0),removeExp(3,0),removeExp(4,0) executed.
Printing the list :
--------------------------
# ExperimentList | Size : 8
--------------------------
Day:1 03:26:20 | exp1_1 | Acc: %97.5 | completed
Day:1 03:26:20 | exp1_0 | Acc: %100.0 | completed

Day:2 03:26:20 | exp2_1 | Acc: %97.5 | completed
Day:2 03:26:20 | exp2_0 | Acc: %100.0 | completed

Day:3 03:26:20 | exp3_1 | Acc: %97.5 | completed
Day:3 03:26:20 | exp3_0 | Acc: %100.0 | completed

Day:4 03:26:20 | exp4_1 | Acc: %97.5 | completed
Day:4 03:26:20 | exp4_0 | Acc: %100.0 | completed
--------------------------
# List day view:
Day:1 03:26:20 | exp1_1 | Acc: %97.5 | completed
Day:2 03:26:20 | exp2_1 | Acc: %97.5 | completed
Day:3 03:26:20 | exp3_1 | Acc: %97.5 | completed
Day:4 03:26:20 | exp4_1 | Acc: %97.5 | completed
--------------------------
```

Experiments are removed succesfully. We can see, 0th index of every experiments is gone and by looking the List day view part, nextDay structure is not damaged.

Time complexity : Method removeExp() is finds the correct node to delete with a single while loop with using nextDay structure to jump to nextDays easily. So this while loop is N/x where x defines the jumps. After this, calling the iterator's remove. In this method, again we have a single while loop finds the correct node, so this is N too. So we have O(N) and O(N) gives us to **O(N)**.

4) removeDay() method

```
list1.removeDay(2);
list1.removeDay(4);
```

Theese removeDay operations executed. (Current list in the test 3.) Look up the results :

```
--------------------------
# Test 4 : removeDay() method tested.
removeDay(2), removeDay(4) executed.
Printing the list :
--------------------------
# ExperimentList | Size : 4
--------------------------
Day:1 03:26:20 | exp1_1 | Acc: %97.5 | completed
Day:1 03:26:20 | exp1_0 | Acc: %100.0 | completed

Day:3 03:26:20 | exp3_1 | Acc: %97.5 | completed
Day:3 03:26:20 | exp3_0 | Acc: %100.0 | completed
--------------------------
# List day view:
Day:1 03:26:20 | exp1_1 | Acc: %97.5 | completed
Day:3 03:26:20 | exp3_1 | Acc: %97.5 | completed
--------------------------
```

We can see, 2nd and 4th days are gone from the list. By looking the (#List day view), nextDay list structure is not damaged.

Time complexity : In removeDay() method, we call removeExp() method in a while loop to delete all days. Since removeExp() is O(N) and we call it in a while loop, the time complexity of removeDay() method is **O(N'2)**.

5) setExp() method

```
list1.setExp( day: 1, index: 1,new Experiment( setupVal: "new_exp1", dayVal: 1, accuracyVal: 15f, isCompleted: false));
list1.setExp( day: 3, index: 0,new Experiment( setupVal: "new_exp2", dayVal: 3, accuracyVal: 35f, isCompleted: false));
```

Theese setExp() methods executed. (Current list above in test 4.). Lets look up the results :

```
# Test 5 : setExp() method tested. setExp(1,1), setExp(3,0)
executed with experiment 15 and 35 accuracy values.
For testing purposes of listExp() method which are will be tested next step,completed selected false.
Printing the list :
--------------------------
# ExperimentList | Size : 4
--------------------------
Day:1 03:26:20 | exp1_1 | Acc: %97.5 | completed
Day:1 03:26:20 | new_exp1 | Acc: %15.0 | not completed

Day:3 03:26:20 | new_exp2 | Acc: %35.0 | not completed
Day:3 03:26:20 | exp3_0 | Acc: %100.0 | completed
--------------------------
# List day view:
Day:1 03:26:20 | exp1_1 | Acc: %97.5 | completed
Day:3 03:26:20 | new_exp2 | Acc: %35.0 | not completed
--------------------------
```

We can see, experiments setted succesfully and nextDay list structure is not damaged by looking the list day view.

Time complexity : In this set method, private helper method findNode(day,index) method used. This helper method contains a single while loop so this method is O(N). Set method, just invokes this helper method. So time complexity of setExp() method is **O(N)**.

6) listExp() method

```
list1.listExp( day: 1);
list1.listExp( day: 3);
```

Theese listExp() method executed. Result :

```
# Test 6 : listExp() method tested.
listExp(1), listExp(3) executed.
--------------------------
Day:1 03:26:20 | exp1_1 | Acc: %97.5 | completed
Day:3 03:26:20 | exp3_0 | Acc: %100.0 | completed
Completed experiments succesfully printed.
# Test 6 finished.
```

Time complexity : This method, traverses in list by single while loop. So the time complexity of listExp() method is **O(N)**.

7) orderExperiment() method

```
--------------------------
# Test 7 : orderExperiment() method tested via printOrderedExp method.
Scroll up and see list before ordered in Test 5.
--------------------------
Day:1 03:26:20 | new_exp1 | Acc: %15.0 | not completed
Day:3 03:26:20 | new_exp2 | Acc: %35.0 | not completed
Day:1 03:26:20 | exp1_1 | Acc: %97.5 | completed
Day:3 03:26:20 | exp3_0 | Acc: %100.0 | completed
--------------------------
# Experiments are ordered succesfully.
# Test 7 finished.
--------------------------
```

Time complexity : This method tested with printOrderedExp() method. OrderedExperiment() method creates a new ExperimentList, copy the original list to the new ExperimentList, this operation is done by single while loop, so this give us to O(N) time complexity. After copying with copy constructor, sorting process is started. Sorting is done by two nested for loops, this give us to O(N'2) time complexity. Combining O(N) and O(N'2) gives that, orderExperiment() has **O(N'2)** time complexity.

8) getExp() method

```
--------------------------
# Test 8 : getExp() method tested. getExp(3,0) invoked.
Scroll up and see in Test 5.
getExp(3,0) returns : Day:3 03:26:20 | new_exp2 | Acc: %35.0 | not completed
# Test 8 finished. Experiment printed succesfully
--------------------------
```

Time complexity : This method, gets the experiment with private helper method findNode(day,index). This private helper method findNode() has O(N) time complexity since it uses single while loop to find the experiment. So getExp() method has **O(N)** time complexity.