COSC 2123/1285 Algorithms and Analysis
Semester 2, 2018

Assignment 1 Description

Due date: 11:59pm Sunday, 9 September, 2018 **Weight:** 15%
Pair (Group of 2) Assignment

1 Objectives

There are two key objectives for this project:

- Use a number of fundamental data structures to implement the *Multiset* abstract data type.
- Evaluate and contrast the performance of the data structures with respect to different usage scenarios and input data.

2 Background

Multisets, also known as bags, are an important abstract data type. Sets are an unordered collection of elements with unique values. A multiset is a set that can have repeated values.

c	a	b	a	c	a

Figure 1: Example of a multiset of characters.

In this assignment, you will implement the multiset abstract data type using a number of basic data structures and evaluate their performance.

3 Tasks

The assignment is broken up into a number of tasks, to help you progressively complete the project.

Task A: Implement Multiset using Different Data Structures (7 marks)

In this task, you will implement the multiset abstract data type using a number of data structures, namely *singly linked list*, *sorted (singly) linked list* and *binary search tree*. Your implementation should support the following operations:

- Create an empty multiset (implemented as a constructor that takes zero arguments).
- Add an element.
- Search for an element.
- Remove all instances of an element.
- Remove one instance of an element.
- Print out the multiset.

Data Structure Details

Multisets can be implemented using a number of data structures. We provide a hash table based implementation (we will learn about hash tables in week 8) and a balanced tree implementation (week 4) to help you get started and also as additional implementations for you to evaluate in task B. You need to implement the multiset abstract data type using the following data structures:

- Singly linked list. Nodes of the linked list should store an element and the number of instances of that element in the multiset.
- Sorted singly linked list. The (singly) linked list should always be maintained in ascending order after any operation, and nodes of the linked list should store an element and the number of instances of that element in the multiset. Operations that can take advantage of, or need to maintain, the ascending ordering of the list should do so. These include:
  - adding an element;
  - searching for an element;
  - removal of element(s).
- Binary search tree (BST), which does *not* necessary have to be *balanced*. Nodes of the tree should store an element and the number of instances of that element in the multiset. Otherwise, the BST should have usual semantics of "left subtree contain values that are less than to the parent node" and "right subtree contains values that are greater than the parent node."

For the above three data structures, you must program your own implementation, and not use the LinkedList or Tree type of data structures in java.util or any other libraries. You must implement your own nodes and methods to handle the operations. If you use java.util or other implementation from libraries, this will be considered as an invalid implementation and attract 0 marks for that data structure.

Operations Details

Operations to perform on the implemented multiset abstract data type are specified on the command line. They are in the following format:

```
<operation> [arguments]
```

where operation is one of {A, S, RA, RO, P, Q} and arguments is for optional arguments of some of the operations. The operations take the following form:

- `A <element>` – add element into the multiset.
- `S <element>` – search for the number of instances that element have in the multiset and prints this number out. See below for the required format.
- `RA <element>` – delete all instances of element from the multiset.
- `RO <element>` – delete one instance of element from the multiset.
- `P` – prints the contents of the multiset. See below for the required format. The elements can be printed in any order.
- `Q` – quits the program.

The format of the output of a search operation should take the form:

```
<element> <number of instances in the multiset>
```

If element does not exist in the multiset, then the 0 should be the number of instances returned.

The print operation should output a number of lines. Each line specifies an element and the number of instances of it in the multiset:

```
<element> | <number of instances in the multiset>
```

As an example of the operations, consider the output from the following list of operations:

```
A robot
A fortune
A macbook
A robot
S robot
S book
A macbook
A macbook
A fortune
RO fortune
P
Q
```

The output from the two search operations (S robot, S book) should be:

```
robot 2
book 0
```

The output from the print operation (P) should be:

```
robot | 2
fortune | 1
macbook | 3
```

Testing Framework

We provide Java skeleton code (see Table 1) to help you get started and automate the correctness testing.

In addition, we provide a python script that automates testing, based on input files of operations (such as example above). These are fed into the java framework which calls your implementations. The outputs resulting from any search or print operations are stored, then compared with the expected output. We have provided two sample input and expected files for your testing and examination. As the instructions for the assignment is getting too lengthy, these are available on Canvas and within the header of the python script.

For our evaluation of the correctness of your implementation, we will use the same python script and input/output/expected files, so to avoid unexpected failures, please do not change the python script nor MultisetTester.java. If you wish to use the script for your timing evaluation, make a copy and use the unaltered script to test the correctness of your implementations, and modify the copy for your timing evaluation. Same suggestion applies for MultisetTester.java.

file	**description**
`MultisetTester.java`	Code that reads in operation commands from stdin then executes those on the selected multiset implementation. *No need to modify this file.*
`Multiset.java`	Abstract class for multisets. All implementions should extend the Multiset class defined in this file. *No need to modify this file.*
`LinkedListMultiset.java`	Code that implements a singly linked list implementation of a multiset. Complete the implementation (implement parts labelled "Implement me!").
`SortedLinkedListMultiset.java`	Code that implements a sorted linked list implementation of a multiset. Complete the implementation (implement parts labelled "Implement me!").
`BstMultiset.java`	Code that implements a BST implementation of a multiset. Complete the implementation (implement parts labelled "Implement me!").
`HashMultiset.java`	Code that implements a hash table implementation of a multiset. *No need to modify this file.*
`BalTreeMultiset.java`	Code that implements a balanced tree implementation of a multiset. *No need to modify this file.*

Table 1: Table of Java files.

Notes

- Use the output of the provided hash tables and balance tree implementations to help you determine the right output format for the operations. If you correctly implement the "Implement me!" parts, you in fact do not need to do anything else to get the correct output formatting. MultisetTest.java will handle this.
- We will run the supplied test script on your implementation on the university's core teaching servers, e.g., `titan.csit.rmit.edu.au`, `jupiter.csit.rmit.edu.au`, `saturn.csit.rmit.edu.au`. If you develop on your own machines, please ensure your code compiles and runs on these machines. You don't want to find out last minute that your code doesn't compile on these machines. If your code doesn't run on these machines, we unfortunately do not have the resources to debug each one and cannot award marks for testing.
- All submissions should be compile with no warnings on Oracle Java 1.8 when compiling the files specified in Table 1.

Task B: Evaluate your Data Structures (8 marks)

In this second task, you will evaluate your three implemented structures, along with the two provided, in terms of their time complexities for the different operations and different use case scenarios. Scenarios arise from the possible use cases of a multiset. We know, or can calculate, the best and worst cases for each operation. However, it is also interesting to evaluate how the data structures perform under different use cases.

Write a report on your analysis and evaluation of the different implementations. Consider in which scenarios each type of implementation would be most appropriate, and subsequently which data structures you recommend to implement those in. The report should be **6 pages or less**, in font size 12. See the assessment rubric (Appendix A) for the criteria we are seeking in the report.

Use Case Scenarios

Typically, you may real usage data to evaluate your data structures. However, for this assignment, you will write data generators to enable testing over different scenarios of interest. There are many possibilities, hence to help you we suggest you consider the following factors.

- The size of the multiset.
- Different scenarios (see below).

For this assignment, consider the following scenarios:

Addition and Removal:

- The number of additions versus removals vary from a growing multiset (i.e., only have additions), to one that fluctuates between roughly equal number of additions and removals, to a multiset that is shrinking (only have removals).

Searches:

- The number of searches are greater than the number of additions and removals (consider varying a ratio between the number of searches versus the total number of additions and removals).

When generating elements to insert, remove and search for, the distribution of these elements, compared to what is in the multiset already, will have an effect on the timing performance (recall the average case for sequential search we talked about in class). However, without knowing the actual application the multiset will be used in, e.g., as a shopping cart, it is difficult to specify what these distributions might be. Instead, in this assignment, assume the elements come from a fixed set (you determine these sets, but you may want to consider fixing the ratio of the size of your fixed set to the size of the tested multiset) and uniformly sample from this fixed set when generating elements for additions, removals and searching, i.e., each element has equally likely chance of being selected for the mentioned operations.

Analysis

In your analysis, you should evaluate each of your data structures in terms of the different scenarios outlined above and for different sized multisets. For generating different multiset sizes, you may want to either generate a series of add operations ('A') to initially grow the multiset to the desired size, then evaluate the appropriate scenarios. Alternatively can consider writing a data generator within Java to insert directly into the data structures. Whichever method you decide to use, remember to generate multisets of different sizes and scenarios to evaluate on. Due to the randomness of the data, you may wish to generate a few datasets with the same parameters settings (same multiset size and a scenario) and take the average across a number of runs.

Note, you may be generating and evaluating a significant number of datasets, hence we advise you to get started on this part relatively early.

4 Report Structure

As a guide, the report could contain the following sections:

- Explain your data and experimental setup. Things to include are (brief) explanations of the data scenarios you decide to evaluate on (e.g., what ratio of the number of searches to additions/removals did you evaluate and why these), the range of multiset sizes (add some brief explanation of why this range selection), describe how the scenarios were generated (a paragraph and perhaps a figure or high level pseudo code suffice), which approach(es) you decide to use for measuring the timing results, and briefly describe the fixed set(s) you used to generate the elements for searching, addition, removal and the initial population of the multisets.

- Evaluation of the data structures using the generated data (different scenarios and multiset sizes). Analyse, compare and discuss your results. Provide your explanation on why you think the results are as you observed. You may consider using the known theoretical time complexities of the operations of each data structure to help in your explanation.

- Summarise your analysis as recommendations, e.g., for this certain data scenario of this data size, I recommend to use this data structure because... We suggest you refer to your previous analysis to help.

5 Submission

The final submission will consist of three parts:

- Your **Java source code** of your implementations. Your source code should be placed into in a flat structure, i.e., all the files should be in the same directory/folder, and that directory/folder should be named as `Assign1-<your student number>`. Specifically, if your student number is s12345, when `unzip Assign1-s12345.zip` is executed then all the source code files should be in directory Assign1-s12345.

- Your **written report for part B** in PDF format, called "assign1.pdf". Place this pdf within the Java source file directory/folder.

- Your **data generation code**. Create a sub-directory/sub-folder called "generation" within the Java source file directory/folder. Place your generation code within that folder. We will not run the code, but may examine their contents.

- Your groups contribution sheet. See the following Team Structure section for more details. This sheet should also be placed within your source file folder.

Note: **submission of the report and code will be done via Canvas**. We will provide details closer to the submission deadline. We will also provide details on how the setup we use to test the correctness of your code will be like, so you can ensure your submitted structure will conform to the automated testing we will perform.

6 Assessment

The project will be marked out of 15. Late submissions will incur a deduction of 3 marks per day, and no submissions will be accepted 5 days beyond the due date.

The assessment in this project will be broken down into two parts. The following criteria will be considered when allocating marks.

Implementation and checkpoint (**7/15**):

- You implementation will be assessed based on the number of tests it passes in our automated testing.

- While the emphasis of this project is not programming, we would like you to maintain decent coding design, readability and commenting, hence commenting and coding style will make up a portion of your marks.

- **Checkpoint (2 marks):** As part of the implementation work and to help you progress, during the laboratory of week 6, one of you should at least attend your allocated laboratory[1], where your demonstrator will assess your progress on your implementations. We will run the provided script on some input to test if your *linked list* implementation compiles, runs and passes a number of basic tests. This does not guarantee your code is bug free and will pass all our final tests. Please conduct your own further testing.

Report (**8/15**): The marking sheet in Appendix A outlines the criteria that will be used to guide the marking of your evaluation report[2]. Use the criteria and the suggested report structure (Section 4) to inform you of how to write the report.

7 Team Structure

This project should be done in **pairs** (group of two). If you have difficulty in finding a partner, post on the discussion forum or contact your lecturer. If there are issues with work division and workload in your group, please contact your lecture as soon as possible. In addition, please submit what percentage each partner made to the assignment (a contribution sheet is made available for you to fill in), and submit this sheet in your submission. The contributions of your group should add up to 100%. If the contribution percentages are not 50-50, the partner with less than 50% will have their marks reduced. Let student A has contribution X%, and student B has contribution Y%, and $X > Y$. The group is given a group mark of M. Student A will get M for assignment 1, but student B will get $\frac{M}{\frac{X}{Y}}$.

8 Plagiarism Policy

University Policy on Academic Honesty and Plagiarism: You are reminded that all submitted project work in this subject is to be the work of you and your partner. It should not be shared with other groups. **Multiple automated similarity checking software will be used to compare submissions**. It is University policy that cheating by students in any form is not permitted, and that work submitted for assessment purposes must be the independent work of the students concerned. Plagiarism of any form may result in zero marks being given for this assessment and result in disciplinary action.

For more details, please see the policy at `http://rmit.info/browse;ID=sg4yfqzod48g1`.

9 Getting Help

There are multiple venues to get help. There are weekly consultation hours (see Canvas for time and location details). In addition, you are encouraged to discuss any issues you have with your Tutor or Lab Demonstrator. We will also be posting common questions on the Assignment 1 Q&A discussion thread on Canvas and we encourage you to check and participate in the discussion forum on Canvas. However, please **refrain from posting solutions**.

[1] If this is not possible, contact your lecturer.

[2] Note for the marking guide, if one of the criteria is not demonstrated at all, then 0 marks will be awarded for that criteria.

A Marking Guide for the Report

Design of Evaluation (Maximum = 2 marks)	**Analysis of Results** (Maximum = 4 marks)	**Report Clarity and Structure** (Maximum = 2 marks)
2 marks Data generation is well designed, systematic and well explained. All suggested scenarios, data structures and a reasonable range of multiset sizes were evaluated. Each type of test was run over a number of runs and results were averaged.	4 marks Analysis is thorough and demonstrates understanding and critical analysis. Well-reasoned explanations and comparisons are provided for all the data structures, scenarios and multiset size. All analysis, comparisons and conclusions are supported by empirical evidence and/or theoretical complexities. Well reasoned recommendations are given.	2 marks Very clear, well structured and accessible report, an undergraduate student can pick up the report and understand it with no difficulty.
1.5 marks Data generation is reasonably designed, systematic and explained. There are at least one obvious missing suggested scenarios, data structures or reasonable multiset size. Each type of test was run over a number of runs and results were averaged.	3 marks Analysis is reasonable and demonstrates good understanding and critical analysis. Adequate comparisons and explanations are made and illustrated with most of the suggested scenarios and multiset sizes. Most analysis and comparisons are supported by empirical evidence and theoretical analysis. Reasonable recommendations are given.	1.5 marks Clear and structured for the most part, with a few unclear minor sections.
1 mark Data generation is somewhat adequately designed, systematic and explained. There are at several obvious missing suggested scenarios, data structures or reasonable multiset size. Each type of test may only have been run once.	2 marks Analysis is adequate and demonstrates some understanding and critical analysis. Some explanations and comparisons are given and illustrated with one or two scenarios and multiset sizes. A portion of analysis and comparisons are supported by empirical evidence and theoretical analysis. Adequate recommendations are given.	1 mark Generally clear and well structured, but there are notable gaps and/or unclear sections.
0.5 marks Data generation is poorly designed, systematic and explained. There are at many obvious missing suggested scenarios, data structures or reasonable multiset size. Each type of test has only have been run once.	1 mark Analysis is poor and demonstrates minimal understanding and critical analysis. Few explanations or comparisons are made and illustrated with one scenario and multiset size. Little analysis and comparisons are supported by empirical evidence and theoretical analysis. Poor or no recommendations are given.	0.5 marks The report is unclear on the whole and the reader has to work hard to understand.