COSC-160-01
Spring 2019
Project 1 Description

List Efficiency

1. Summary

For the first part of the semester, we have focused on two overarching topics: linear data structures and computational efficiency. Those two foundational topics intersect here in Project 1. For this project, you will design and implement two different List data structures that are designed for efficient data retrieval: **orderedList** structure and **MTFlist** structure. The efficiency of the two list structures will be compared empirically. The details of implementation are largely at your discretion. Major factors to consider for the design should be theoretical efficiency (both time and space), practical efficiency, and appropriate memory management. Although some related concepts and tips are provided below (and have been provided in class), the determination of how to make the structure efficient is largely your charge.

2. Submission and Compilation Requirements

To standardize submissions, you will submit a makefile, which will contain the necessary compilation commands for your code. The target executable will be named **p1**.

Thus, the graders should be able to compile your program by accessing it on the course server and writing the following command:

make p1

Check to make sure that it works on the class server. If the program does not compile (using the above make command) or the program does not run, the submission will not be accepted.

You must also submit a ***cover letter*** for this project. Your cover letter (no more than 1 page total) should explain what your program does and describe the various components. It should also include your estimates for the efficiency of your program (time and space for both types of lists) and a brief explanation for how you optimized the efficiency.

Budget your time well. Include significant time for design / planning and testing / debugging. Please submit early and often (version control)! Your last submission will be graded.

3. Input Requirements

After successful compilation, the following command will run your program:

./p1 [inputFilename]

The program will take one command line argument, which will be a text file (assumed to be located in the same directory as the executable). For example, the command **./p1 uniform.txt** will run the program using as input a text file named uniform.txt that is located in the same directory. The program will load and store a collection of numbers into the List structures. The first line of the input file will be an integer that is the length of the list. This will be followed by a new line character. The second line will contain all the numbers to store; we will refer to this sequence of numbers as the "list". This will be followed by a new line character. The next line will then contain a number (the number of queries) followed by a new line character. Next a sequence of numbers to retrieve from the list; we will refer to this sequence as the "queries".

4. Output Requirements

   The main method should construct the two list structures, read in the input file populating both lists appropriately, perform the sequential retrievals for each list separately, and finally display the retrieval results to the console. (The main method should also be efficient.) The following steps should be executed by the main method:

   A. Read input file and store data items in both lists.
   B. Record the time needed to complete the sequence of retrievals for the Ordered Array Structure. Include the time needed to initially sort the list.
   C. Record the time needed to complete the sequence of retrievals for the MTF Linked List Structure.
   D. Display the total retrieval times for both Lists, and declare which was faster.

   No other outputs should be observed.

5. Structural and Operational Requirements
   A. An orderedList (of integers or template if you wish) structure
      - integers stored in this list must be in increasing order
      - a merge sort method should be implemented (to impose above requirement)
      - a searching method should be implemented (to facilitate queries)
   B. An MTF (self-organizing) List (of integers or template if you wish) structure
      - integers are initially stored in an arbitrary order
      - a searching method should be implemented (to facilitate queries). this method should re-organize the list using the move-to-front strategy.

   Notes: To earn full marks it is expected that you will implement the structures very efficiently (in space and time). If you are faced with a space / time tradeoff, you will opt to improve time complexity (if the cost of space is relatively minor).

6. Rubric

List of Requirements	**Percentage**
OrderedList Structure	0.20
OrderedList Search	0.10
OrderedList Sort	0.10
MTFList Structure	0.20
MTFList Search	0.10
Efficient Input and Output	0.15
Cover letter	0.15
TOTAL	**1.00**

The following sections are boilerplate for this course, but contain valuable information:

7. Programming Languages

   You should submit your projects using C++ (unless you have spoken with me and I have approved a different programming language for your project). Take note that there are many versions of C++; you must use the version that is currently running on the course server. Keep in mind that one of the main goals of our class projects is for you to learn how to construct various data structures from the most elemental programming constructs. Thus you will not receive credit when using any pre-existing structures from programming libraries (or code that has been created or designed by others). For example in C++ you *cannot* use the pre-existing vectors, stacks, lists, etc. For some programming languages, complex data structures (non-elemental constructs) are “built-into” the language. You cannot use any built-in structure. If you have any questions as to what structures are permitted (and which are not permitted), given your language of choice, please ask me.

8. Planning and Design

   Before implementation, you should plan and design your project using standard approaches, e.g. UML class diagrams, flow diagrams, etc. If you have questions pertaining to your project, I will first ask to see your designs. I will not look at your code without first viewing your design documents. You will be faced with many design decisions during this project. It is best to spend the requisite time during the design stages to assure an appropriate and efficient implementation is built. Consider your options, perform a theoretical complexity analysis of the different options, and base your decision on the results of your analysis.

9. Testing and Debugging (Not Submitted)

   You may wish to construct an interactive interface to test the functionality of your structure at intermediate stages of development. This would likely be most efficient with an interactive interface that allowed you to interactively test various functionalities of your structure given different inputs. If you do implement a testing interface, please be sure to comment it (so that it does NOT execute) before submission. I also strongly encourage you to construct and test many input files to test the functionality of your implementation on varying inputs.

10. Version Control (Not submitted, but encouraged)

    I strongly recommend that you back-up your work periodically throughout the development process. This can mitigate a disaster scenario where you might accidentally delete your program files. I also recommend employing a version control strategy which records your development at different stages (versions). If you have time, I encourage you to investigate GitHub to facilitate version control. Otherwise you can make use of a more simplistic naming scheme: each time you save a file, change the filename to indicate a version: filename v1.cpp, filename v2.cpp, ...