4. Self-Organizing Lists Assignment

# Assignment

Implement the three self-organizing list heuristics:

* Count – Whenever a record is accessed it may move toward the front of the list if its number of accesses becomes greater than the record(s) in front of it. If the record is not in the list it is added to the back of the list and its count is set to zero.
* Move-to-front – Whenever a record is accessed it is moved to the front of the list. This heuristic only works well with linked-lists; because, in arrays the cost of shifting all the records down one spot every time you move a record to the front is too expensive.
* Transpose – whenever the record is accessed swap it with the record immediately in front of it provided it is already in the list. If it is not found in the list and is being added to the list for the first time, it goes to the back of the list.

Compare the cost of each heuristic by keeping track of the number of compares required when searching the list.

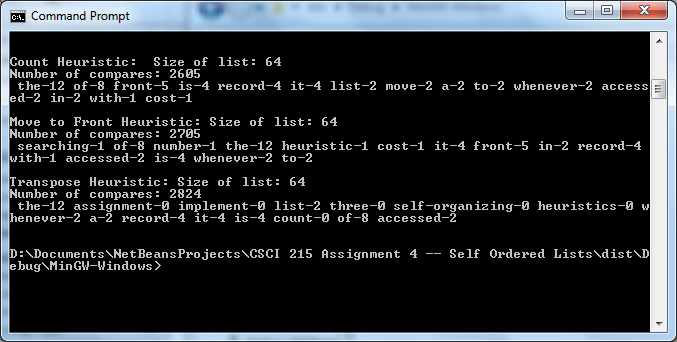
# Additional Instructions

Use the SelfOrderedListADT abstract data type and the linked-list files I have provided to implement your self-ordered lists. You may incorporate the author’s linked list implementation via inheritance or composition, which ever makes the most sense to you (I will not evaluate that aspect of your implementation). You are allowed to make changes to any of the files I have provided, except SelfOrderedListADT and test.txt, to make your implementation of SelfOrderedListADT cleaner. The same applies to the link.h (the link node implementation). You may not change SelfOrderedListADT or test.txt files.

I want you to run two tests

1. The first test is with char types. Use the add() function to build a list in the following order: A B C D E F G H (do not add ‘I’ here). After you have built that initial list I want you to use the find function to input the following characters: F D F G E G F A D F G E H I (note that ‘I’ is not in the initial list; I want to see what your program does when it searches for an item that is not already in the list). For each heuristic display the order of the final list and the number of compares.
2. The second test is using the test.txt file I have provided using the data type string. Do not modify the test text file; because, I am going to compare your results with my own and modifying the test file will throw your results off. For this test I want you to do the following for each heuristic:
   1. Read into your program the test.txt file adding words to your list using your find() function, and then
   2. Print out
      1. the total number of words in your list,
      2. the total number of compares, and
      3. the first 10 words in your list along with their frequency.

Here is a sample of a test run using strings (this sample was not run using the test file I’ve assigned you, so the values you see are not the same as those you should be seeing):



Make sure that you give me enough screen shots so that I can see your entire run. You will be penalized if you do not.

SelfOrderedListADT Functions:

1. find() – finds a value in the list and, if found, increments the frequency. If not found then find() calls add() to append the value to the end of the list (initial frequency of an item added this way is 0). In either case find() calls your reorder function (see below) to reorder the list in accordance to the heuristic being used and find() increments the number of compares made (whereas add() (see below) does not).
2. add – appends the value to the end of the list without doing any compares or adjusting frequencies.
3. getCompares – returns the total number of compares done by find when searching for values in the list.
4. Size – returns the size of the list.
5. printlist – prints the list in the following format: value-## where “value” is the actual value of the node (either a char or a string) and ## is the frequency of that value. You will need a printlist() and a printlist(n) method because for your char tests you will print the entire list but for the string test I only want the first 10 nodes printed.
6. reorder – you can call this method whatever you want but what I am looking for is a method or methods that reorders your list as appropriate based on the heuristic you are using. This method is typically called by find().

# Submit

To receive credit for this assignment you must submit the following files in a zip file to Blackboard:

* Word document describing your approach and calling out changes you made to any of the files I have given you.
* Any source files you have changed
* Your executable file
* The output of your executable run (either in screen shots or an output.txt file)

# Rubrics

To earn any credit at all you must submit a working program that implements and demonstrates at least one of the heuristics and it must be submitted on time. You must also include all the files I asked for above or you will not receive any credit. You must also use the SelfOrderedListADT I have provided (unchanged) by establishing an inheritance relationship between your self-ordered list(s) and the ADT. Failure to do so will disqualify your project and you will not receive any credit for it.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Rubrics** |  | **Count** | **Move-to-front** | **Transpose** | **Totals** |
| **Test List** | **Val%** | **40%** | **25%** | **35%** | **100%** |
| **Chars A-I** | 40% | 16.0 | 10.0 | 14.0 | 40.0 |
| **Strings (test.txt)** | 60% | 24.0 | 15.0 | 21.0 | 60.0 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
| **Rubrics -- Char Test** |  | **Count** | **Move-to-front** | **Transpose** | **Totals** |
| **Method/Heuristic** | **Val%** | **40%** | **25%** | **35%** | **100%** |
| **Compares** | 48% | 7.6 | 4.8 | 6.7 | 19.0 |
| **Size** | 5% | 0.8 | 0.5 | 0.7 | 2.0 |
| **Reorder** | 48% | 7.6 | 4.8 | 6.7 | 19.0 |
| **Total** | 100% | 16.4 | 10.3 | 14.4 | **40.0** |
|  |  |  |  |  |  |
| **Rubrics -- String Test** |  | **Count** | **Move-to-front** | **Transpose** | **Totals** |
| **Method/Heuristic** | **Val%** | **40%** | **25%** | **35%** | **100%** |
| **Compares** | 45% | 10.8 | 6.8 | 9.5 | 27.0 |
| **Size** | 10% | 2.4 | 1.5 | 2.1 | 6.0 |
| **Reorder** | 45% | 10.8 | 6.8 | 9.5 | 27.0 |
| **Total** | 100% | 24.4 | 15.3 | 21.4 | **60.0** |

[Course Learning Outcomes Alignment](https://drive.google.com/file/d/15Vh7QGueewlgVNuIiT44G7NfDVEeVHz0/view?usp=sharing)