Approach Document Self-Organizing Lists

# Assignment Objective

Implement each Self-Organizing-list heuristics efficiently and effectively

* Apply common software data structures to applications.
* Apply a biblical worldview to data structures and algorithms.
* Understand the concept of each of the heuristics, their differences, and why they are important.
* Understanding the concept of each Self-Organizing list heuristic and their applications where they are useful and when they are useful
* Using the linked-list data structure to help implement these heuristics
* Understanding how to read files into string objects

# Assignment Requirements

The functions I need to create with their respective header files:

For reference I edited two of the given header files below, llist (linked-list) and the link classes.

Within the link header file I created two functions

* setFrequency and getFrequency even though the frequency instance variable was public, for me I could just see things better if I created two functions to set and get the frequency.

Within the llist (linked-list) header file that is inherited from the list header file:

* getFreq 🡪 gets the frequency of the element within the linkedlist
* setFreq 🡪 This sets the frequency of the element within the linkedlist
* getHead 🡪 This function gets the head of the linked list (i.e. the first element)
* getCurrent 🡪 This function gets the current element that is being pointed to in the linked list

Within the Transpose header file that is inherited from the SelfOrderedListADT header file:

* Two private variables compares and list
* The add function simply appends whatever element into the list using the append function from the llist class
* getCompares returns the number of comparisons that were made trying to find the value to help the user understand what is “more efficient”
* size returns the size of the linked list
* printlist prints out the entirety of the list
* However, printlist with the integer parameter n specifies how much of the list is to be printed out, in our case for this assignment we printed out the first 10 values of the linked list
* I added the clear function in case I wanted to clear the linked list
* The find function tries to find if the element is in the list. If it is, then it will call the reorder function, which is simply the implementation of the Transpose heuristic, if that value was found in the list. If that value was not found in the list, then it will call the add function to append that value to the list.
* Lastly, the reorder function essentially just implements the Transpose heuristic on the item found in the list.

Within the Count header file that is inherited from the SelfOrderedListADT header file:

* Two private variables compares and list
* The add function simply appends whatever element into the list using the append function from the llist class
* getCompares returns the number of comparisons that were made trying to find the value to help the user understand what is “more efficient”
* size returns the size of the linked list
* printlist prints out the entirety of the list
* However, printlist with the integer parameter n specifies how much of the list is to be printed out, in our case for this assignment we printed out the first 10 values of the linked list
* I added the clear function in case I wanted to clear the linked list
* The find function tries to find if the element is in the list. If it is, then it will call the reorder function, which is simply the implementation of the Count heuristic, if that value was found in the list. If that value was not found in the list, then it will call the add function to append that value to the list.
* Lastly, the reorder function essentially just implements the Count heuristic on the item found in the list.

Within the MoveToFront header file that is inherited from the SelfOrderedListADT header file:

* Two private variables compares and list
* The add function simply appends whatever element into the list using the append function from the llist class
* getCompares returns the number of comparisons that were made trying to find the value to help the user understand what is “more efficient”
* size returns the size of the linked list
* printlist prints out the entirety of the list
* However, printlist with the integer parameter n specifies how much of the list is to be printed out, in our case for this assignment we printed out the first 10 values of the linked list
* I added the clear function in case I wanted to clear the linked list
* The find function tries to find if the element is in the list. If it is, then it will call the reorder function, which is simply the implementation of the MoveToFront heuristic, if that value was found in the list. If that value was not found in the list, then it will call the add function to append that value to the list. Since this is a move-to-front heuristic, that value will be immediately moved to the front of the list regardless of if it was just added.
* Lastly, the reorder function essentially just implements the MoveToFront heuristic on the item regardless if it was found in the list or not.

# Approach

* Review the lab requirements
* Search for websites to help
* Review lecture slides and read the book
* Watch YouTube videos to understand what is happening
* Set up the approach document
* Tackle each function one at a time with the main influence on the reorder function that is altered for each heuristic.

# Build Log

10/30/23

To start, I am going to review the slides, despite having a good understanding of how I believe each heuristic works. Just in case I have overlooked how something is implemented. Next, I am going to review the instructions to see what is desired and implement my ideas beforehand with the instructions to build a seamless result. Afterward, I will write my algorithm in pseudocode to figure out how each function should be implemented.

10/31/23:

After reviewing everything and writing some pseudocode, I am now going to implement it. I decided yesterday I am going to create three separate header files, as the instructions do not have any restrictions about that. From there, I am going to use inheritance on each heuristic, and I am going to implement the easy functions first i.e., add, getCompares, printlist (both iterations of it), size, and clear in case I want to clear the list. I will do this for each heuristic as those will all be the same. From there, I will call it a day.

11/02/23:

Now, I am going to complete the find function. Since I will be implementing the frequency during the find function, I will create two functions within the linkedlist header file, one that sets the frequency and another that gets the frequency, which will help me increment the frequency if the value is found in the list. Essentially, the find function will follow the same algorithm for each heuristic, except for the MoveToFront where regardless of the item being in the list or not in the list the heuristic will need to be used on it to move that value to the front. In the other cases, the value will simply be appended to the list, if it is not in the list.

11/07/23:

The last function(s) I will build is the reorder function which will be the implementation of each heuristic. Since my understanding of this concept is quite well, the implementation should presumably be easy. However, last week I realized that I would have to create two more functions within the linkedlist class getCurrent and getHead where getCurrent gets the current element being pointed to and getHead gets the head of the linked list. Essentially, the algorithm will be if the item is found, and it is not the head of the list then apply the heuristic to it. However, in the case of MoveToFront regardless of if the item is in the list, the item will always be moved to the front of the list.

11/08/23:

Now, all the functions are complete, and I have accounted for the debugging that will go into this algorithm. I will implement my main, first with the char elements to assist with the debugging. Once those issues are fixed, then I will read from the file and implement it the same way listing the size of the list, number of compares, and the first 10 items of the list.