Prathyoosha Chaya

CS163: Data Structures

**Program 1 – Efficiency Review Write-Up**

The overlying abstract data type used in this program was a relative doubly linked list (with the two traversal pointers pointing to different threads within the same list, instead of a “prev.” and “next”). In this scenario, a relative list is definitely the right choice to make in ordering the list – allowing the news data to be sorted relative to other members. The underlying data structures used were a struct type to create the data classification within the node, and a struct node type. Conceptually, the idea of having the underlying data form structures made sense. The data types could be easily grouped together and is easily accessible by the node class.

Most of the effort done was to make sure that all the cases of failure for a linear linked list were covered – requiring situation-dependent code such as the case of an empty list, a list with 1 node, and multiple nodes, each requiring different methods depending if data was being sorted by time or topic. While this would have been made much easier with direct access in an absolute dynamic array data structure, this is not appropriate for the situation. The news needs to be flexibly placed and using a dynamic array would lead to a lot of memory wastage. While the run-time performance would be boosted as a result of not needing traversals with direct access, it would not be worth it as direct access does not allow us to place our information freely.

The major drawback of this design is the efficiency loss through traversals. I think in this case a singly linked list of arrays would work well, especially for large archives of news. I think this especially will work well because there are parts of this assignment that require both an absolute list and a relative list implementation. For example, if all the news stories are arranged chronologically, then having the direct access will provide useful for displaying purposes, where the data structure can be treated as a queue, having each story dequeue off the queue in order of their time. Direct access could also be used here to find the data members that have a certain time stamp without traversing, using the closest and farthest time of the data members in the queue. Also, in some parts of my traversal, I had used a forward-looking approach, which is not the most efficient. This can be changed to not require as many fetches and assignments and save on runtime performance. Another inefficiency was in having to have different traversal methods for both next pointers.

While adding another pointer to the linked list (one “next” pointer for the time, and the other for the topic) will have doubled the amount of memory these pointers use in the list, it also allowed for some efficiency. Instead of having two separate linked lists (one organizing the data by topic, and the other by time), using just one and having two data threads run through it made the design more complicated, but saved on efficiency.

If I had more time, I would like to try exploring how to implement a similar design with different data structures, and finding how to get around some of the limitations of certain data structures. The traversal methods can definitely be refined and made more efficient to reduce the number or operations done. I also would like to make this program more universal, by expanding some of the functionality (display favorite articles, write your archive into a file/read from it, and check better for different error cases).