**Project Part 2: Advanced Data Structures (Due 4/30/23)**

**Description:** This phase of the project extends part 1 with two principal tasks: to discuss two theoretical applications related to your dataset for which you believe two “advanced” ADTs would be appropriate, and to choose a single “advanced” data structure implementation and modify it so that you can read your dataset into the structure, as well as print it out via some mechanism.

More specific details follow:

**Problem 1:** *(Note: I will be very flexible about your choice of application for these.)* Write about one paragraph **each** regarding **two** distinct theoretical applications related to your dataset that would be best handled with **two separate** ADTs we looked at beyond the list-based ones (i.e. your options are Stack/Queue, BST, Priority Queue, Hash Table (Map)). ***Note that you can use one of stack or queue, but not both***. Sample applications for the credit-card fraud dataset are below:

*Sample Application 1: Credit card records could be maintained in a self-organizing BST structure sorted on charge amount, with the main goal to have fast and efficient access to outliers in the data (i.e those that are extremely below or extremely above the average). Since the root of a self-organizing BST will have a charge amount close to the median, it should be easy to compare items with the smallest values (via In-order traversal) or largest values (via Reverse In-order traversal) to this quantity. Outlying data could then be investigated further.*

*Sample Application 2: Modified credit-card records could include a “risk” variable based on how likely the transaction is deemed to be fraudulent based on information like price, location, time, etc. The appropriate ADT for this data and application would then be the priority queue, with higher risk transactions being placed higher in the queue. Investigators (manual or automated) would then “dequeue” records in order of risk when investigating potential fraud.*

**Problem 2:** Choose a single existing data structure implementation not including the list ADT implementations (i.e. your options are ArrayStack, CLQueue, BST, Heap) from class. You are to write a C++ program that can read your dataset from part 1 (or you can choose a new one, if preferred) of the project into this structure, and print the elements one-by-one using *any operation of your choice* (ex: for the stack, you can pop items one by one; for the BST, you can utilize in-order traversal, etc.) You will likely want to consult and/or modify the main file you used in part 1 (or create one, if you did not finish!) to handle the reading of data into the structure, as well as writing of it. But remember – **you will also need to modify the new data structure you have chosen**, which includes details such as modifying or removing other operations (ex: DeleteItem for the BST) and modifying intermediate structures (like the traversal queue for BST) if any.

***Your submission should be a .zip file that includes a .csv file for the final, preprocessed dataset, a series of .h/.cpp source files including the modified structure files and main.cpp, and a .doc(x) or .pdf format for the task 1. Upload the file as “LN\_FN\_ProjP2.zip” where LN is your last name and FN is your first name.***

1. In my first approach application for a credit card fraud, I could use a hash Map to index the values and integrating into the hash map. The hash map could help me to detect fraudulent transaction in real time, a hash is like label a bucket for example a bucket of apple, pineapple, and orange. Using a hash map the transactions could be quickly looked up and check the database of known fraudulent patterns, such as unusual purchase on different locations. A hash map gives us an efficient and fast lookup because the value was previously indexed.
2. For my second approach, I could use a Queue to identify which fraud cases are the most significant and should be investigate using the default procedure First in first Out (FIFO). This ADT allows investigators to quickly focus on the most significant cases, reducing the time required to investigate all transactions. As more transactions come in, the queue could be continuously updating as new transactions are received to guarantee that the most crucial cases are always at the front.