ECE241 PROJECT 1: Sorting and Searching

Due: Oct 14th, 2021, 11:50PM on Gradescope

Introduction:

The stock market is very unpredictable, there are many factors that can impact the trend of stocks in the share market. Recently we have seen how covid-19 has impacted the stock prices, which is why on financial data performing a reliable trend analysis is very difficult.

In this project, you will manage stock price data using data structures and algorithms you learned about in class and perform analysis on historical stock price data. This includes tasks to load the data from a given file and process and store the data in memory. Then, you will perform multiple searches to demonstrate the efficiency of using different data structures to accomplish different tasks.

The stock price data are collected from a subset of the top 2000 valued companies in NASDAQ and stored in a file "stock_database.csv". For simplicity, we only list the Company name, Symbol, Market Value, and the close market stock price from 2021-01-01 to 2021-02-01. Each attribute is separated by '|'.

Task Overview

In this project, you will perform the following specific tasks.

1. Manage the data in array-based objects with Python. Specifically, you can follow the instructions below to start. **Note, you should use the required name for the classes and functions**.
    a. Each stock is managed using a **Stock class.** In the Stock class, you should have variables for the different attributes of the stock.
        1. A string ***sname*** for the stock name.
        2. A string ***symbol*** for the stock symbol.
        3. A float ***val*** for market value.
        4. A list ***prices*** for a list that contains the close market price for one month.
    b. Implement a function ***__str__(self)*** in Stock class to show the stock information in a string. The function returns a string including name, symbol, market value, and the price on the last day (2021-02-01). For example, the string of the first stock should be returned as: "name: Exxon Mobil Corporation; symbol: XOM; val: 384845.80; price:44.84".
    c. Implement a **StockLibrary class** as a library to store the whole dataset. In the StockLibrary class, you should have basic information about the library.
        1. A list ***stockList*** for all the Stock objects.
        2. An integer ***size*** that indicates how many stocks are in the library.
        3. A Boolean variable ***isSorted*** that shows whether the library is sorted or not.
2. Implement a ***loadData(self, filename: str)*** method in StockLibrary class. The method takes the file name ("stock_database.csv", string type) of the input dataset and stores the data of stocks into the library. Make sure the order of the stocks is the **same** as the one in the input file.

3. Implement a linear search algorithm to search the stocks based on sname or symbol. The ***linearSearch(self, name: str, attribute: str)*** method takes two arguments as the string for search and an attribute field that we want to search ("name" or "symbol"). It returns the details of the stock as described in __str__() function or a "Stock not found" message when there is no match.
4. Implement a ***quickSort(self)*** algorithm to sort the stock library database based on the stock symbol. The sorted stocks should be stored in the same stock list. Make sure you change the status of the Boolean variable that shows whether the library is sorted or not.
5. Implement a function ***buildBST(self)*** to build a balanced BST of the stocks based on the symbol. Store the root of the BST as attribute ***bst***, which is a **TreeNode** type. The TreeNode type you can refer to the lecture code for BinarySearchTree in

   https://gist.github.com/mikezink/4aa0f6d2c3fce47f5f6cf9bbb2ed6bb8

   Record the time spent on building the BST. Ideally, the height of a balanced BST for 1118 stocks is 11. In this task, you need to build a tree with a height of **AT MOST 20**.
6. Implement a search function ***searchBST(self, symbol: str)*** based on the symbol attribute. It returns the details of the stock as described in __str__() function or a "Stock not found" message when there is no match.
7. Perform a linear search based on ***symbol*** in the sorted database for the 100 stocks. You can use a random number generator to arbitrarily select 100 stock symbols. Record the average time spent on the linear search.
8. Perform a search of the same 100 stocks in the BST. Record the average time spent.
9. In order to perform a search on BST, we need to spend additional time to build the BST. If we have many stocks to search for, it is worthwhile for the additional sorting time. Using the information recorded in Task 5, 7, 8 to calculate how many linear searches $\boldsymbol{n}$ would be necessary before the overall run time of the linear searches would be more than the combined total of building a BST (only performed once) and $\boldsymbol{n}$ searches in the BST.
10. Plot a figure to shows the prices for the stock with the longest company name.
11. Find the two stocks with the largest percentage change in price within a month, one for the price increase and one for the price decrease. You should list the stock information for the two stocks, also the percentage changes.

Hints and Suggestions:

1. Start early and try to get portions of the program to work step-by-step. Debug your program using a small list of stocks first. Only use the complete file once you feel confident when your program works.
2. *LoadData*(self, filename: str):
   a. You can use file = open(filename, 'r') to open the file or use the csv reader module.
   b. You can use the readline() or readlines() function to read data line by line. The first line in the file is the title. You need to skip it.
   c. You need to split each line based on '|' to obtain the attributes for Stock objects.

What to submit:

For Task 1-6, you should submit your code to Gradescope for auto-grading. Remember to comment your code properly.

For Task 5, 7-11, put your recorded running time of building BST, the average time for linear search and search on BST into a Document (.doc or .pdf). Explain how you compute the number of searches $\boldsymbol{n}$ for task 10 in the document. Submit the document to Gradescope as well.

Reminder: The course honesty policy requires you to write all code yourself, except for the code that we give to you. Your submitted code will be compared with all other submitted codes for the course to identify similarities. Note that our checking program is not confused by changed variables or method names.

Grading

- Code works on Gradescope (70%)
- Assignment results (search number $\boldsymbol{n}$, plot, top stocks) (20%)
- Program structure and comments (10%)