COL106: Data Structures and Algorithms

Assignment 3

Instructor: Prof. Saroj Kaushik *Deadline: TBD*

1 Statement

Having worked on a few projects for small tech companies, you plan to try out your luck in the financial markets. You think that data driven computing is the key to succeed in such financial markets. Inspired from the HFT ideology, you are ambitious about building some computing facility of your own that will help you earn profits in all market conditions.

But before you can process or analyze any live data, you need a way to quickly get data from the stock exchange to your compute servers. Ideally you would want your compute servers to be located right at the stock exchange so that you can get instant access to the data and make trading decisions using your algorithms. This, however is not possible in a real world scenario. Luckily you have a friend Merlin, who stays very close to the exchange. Merlin has allowed you to run a small setup at his residence which won't really be sufficient for your for all the compute you need. So you plan to use this setup to get data to you as fast as possible.

In this assignment, we will together build a Market Data Publisher (MDP). This is a component that receives price data for various symbols and broadcasts them (prints on console for our simple system) as per certain rules. A naive publisher would simply publish any data that it gets instantly, however assume that we are operating in a limited bandwidth environment, and do not want to send out price changes unless they exceed some threshold (θ).

Rule: Do not send out any price unless

$$|newPrice - lastPublishedPrice| > threshold$$

(We will skip some other rules that a MDP might need for brevity of the assignment)

You are required to implement an MDP using an AVL Tree. The stocks will be described as a pair (company-id, stock-price). Since the stock exchange exists even before you created your MDP, you first have to bulk load the existing stocks into your MDP. By bulk loading, we mean inserting into the AVL tree in O(n) time, instead of the usual O(nlog(n)) (Hint: Day-Stout-Warren Algorithm). This creates the need of pre-sorting the stock values on their company-id. Thus, your assignment gets divided into 2 subparts as follows:

Goal 1: Given a input file containing various symbols (stock symbols) and their corresponding prices at the end of some trading session (at some point in time) in $COL106\_STOCK\_EXCHANGE$, you are required to output them in sorted order of the stock symbols.

Implement external merge sort algorithm and output the share symbols along with their prices in sorted order of their stock symbol in a single file. The output format is required to be same as the input format. (Sample input file format is given below). You are allowed to create temporary files in the implementation of your program. If two stocks have the same price, sort them in the order of their stock price.

Sample input file:

```
21 32
65 19
19 10001
```

You are fond of numbers and further believe that only stocks with symbols below some number (k) are lucky for you and offer the best possibility of making profits and so you are only interested in trading only those securities. So given this parameter k, you are required to first bulk load all the (company-id, stock-price), such that $company - id < k$ into an AVL tree and then support the operations given below.

Goal 2:

- Bulk load the key value pairs into an AVL tree
- Support the following operations:
  - **register-company** <company-id> <initial-price>: Insert a new node in the AVL Tree with the given company-id and price
  - **deregister-company** <company-id>: Remove all data of the company with the given company-id
  - **update-price** <company-id> <new-price>: You must update the price of the given company-id, if the above given rule is satisfied. Also in case the rule is satisfied, you must broadcast (print) the previous price
  - **stock-split** <company-id> <x:y>: Readjust the price of stock corresponding to company-id given that the stocks are split in the ratio $x - for - y$

2 Input Format

- All IO operations are to be done using standard input/output.
- Your program will be given five command line arguments: the input file name, the output file name and the run size (the number of stock value pairs that can be read into memory in a go), parameter k, parameter θ, n.
- The first line of the input contains a single integer N - the no. of subsequent operations to be performed on the AVL Tree
- Next N lines consist of 1 operations each in the following format:

<operation-code> <params>

Code	**Operation**
1	register-company
2	deregister-company
3	update-price
4	stock-split
5	reverse-stock-split

3 Output Format

- Each line in output will have exactly 2 space separated integers: ¡company-id¿ ¡stock-price¿
- There is no output for Operation 2.
- Operation 1 outputs the initial price of the stock.
- Remaining operators may or may not publish the new price of the stock based on the condition $|newPrice - lastPublishedPrice| > threshold$.

4 Limits

- //FIXME

5 Sample IO

5.1 Input

//FIXME

6 Submission Instructions

- You are required strictly follow the instructions given below.
- You must submit a single program file which must be named with your *KerberosID.cpp* in small case. For example, if your kerberos id is cs1150999 then the file name should be cs1150999.cpp.
- You will be penalized if your submission does not conform to this requirement.

7 Other Instructions

- The assignment is to be done individually.
- All submissions must use C/C++ for the programming assignment.
- You are **not** allowed to used any standard implementations/libraries (except for IO) in any part of the program.
- You are not allowed to discuss or take help from anybody outside the class. You may only discuss but are not allowed to share code with anyone inside the class.
- We will run plagiarism detection on your submissions. People found guilty will be penalised as per the instructions mentioned in the beginning of the course