COSC 320 – 001
Analysis of Algorithms
2020 Winter Term 2

Project Milestone 2
Wall Street Bets Data Scraper

Group Lead: Tahmeed Hossain
Group Members: Barret Jackson, Omar Mourad, Kenji Niyokindi

Abstract

For this milestone, we explored a different algorithm than in the first milestone for sorting data input from the Reddit scraper as it comes. While a max heap was considered in the last milestone, we are sorting the stock tickers and storing them in a dictionary. Taking the keys with the highest weighted values and storing them within a stack data structure. Filtering them within the stack as more stock tickers with higher values come along. By focusing on the top 3 stocks it drastically minimizes our run-time complexity because within the stack values will only be compared to 3 other values.

Algorithm Analysis

This implementation differs only slightly from the form indicated in Milestone 1, however rather than using a heap to sort our data, we will store all values in a simple dictionary, then keep a second data structure containing the top three stock symbols in terms of their scores. When the data scraper analyzes a post and comes across a valid stock symbol, the score for the post is calculated as before and the daily score for that symbol is either added to the dictionary if the symbol has not been mapped, or is incremented accordingly if it does exist. The score is then compared to the score of the symbol at the top of the stack; if it is smaller, then the algorithm moves to the next post. If it is larger, then the top node of the stack is popped out, the new symbol is added to the stack, and then pushed down the stack until the next node has a larger value.

As an example, say our stack contains stock:value pairs A:4, B:5, and C:9. The scraper comes across a post mentioning stock D and calculates its score to be 7. Since 7 is greater than 4, stock A is popped out of the stack, stock D is added, then stock B and D swap positions since 7 is greater than 5. Our new ordering is B:5, D:7, and C:9, with B at the top of our stack.

Proof of Correctness

Our loop invariant for our algorithm is that the stack will contain the three stock tickers with the highest scores. For the initial condition, the dictionary and stack are both empty, meaning the loop invariant holds.

For maintenance, the algorithm first calls the Reddit API, looks for an occurrence of a stock symbol, then calculates its corresponding score, adding it to the dictionary if it does not yet exist, or incrementing its score if it does. Next, the new score is compared to the stack, giving two cases: i) the score is lower than the lowest score in the stack, or ii) the score is higher than the lowest score in the stack. For case i), we know the stack already holds the top three scoring symbols, and since the score of the new symbol is lower, the stack is not changed, which maintains the loop invariant. For case ii), the top of the stack is popped out because the third highest key value is now smaller than the new symbol, the new symbol is added from the dictionary, and then trickles down so that the next item of the stack has a higher value. This means the top three scores are in the stack, maintaining our loop invariant.

At termination, each post has been examined, and our stack holds the top three scoring stock symbols as in the maintenance section of the proof. Since no stock symbols remain to be examined, the stack contains the top three scoring stock symbols, maintaining our loop invariant and completing the proof.

Time Complexity

The time complexity of our sorting algorithm can be divided into two sections. Time complexity of the dictionary as well as the time complexity of stack operations.

First, we must consider the running time of creating the dictionary and inserting keys/values into it. In a given amount of posts that contain n different stocks; creating the full simple dictionary will take O(n) time. The other main operation to do through our dictionary is to update values for keys that already exist in the list. Whenever a post contains a stock symbol, the algorithm will check the dictionary if there is a matching key, hence for a dictionary with size n, a search operation would normally take up O(n). Therefore, the worst-case scenario for using a dictionary to sort our data is O(n*n) considering insert and update operations.

Second, we will consider the running time of all of our stack operations. Initially, the stack will be empty, however it will have a predetermined constant size given to it. Inserting values into a stack data structure is always O(1) as it is as simple as pushing the new value on top. Another operation we must consider with the stack is the deletion operation, which is similar to the insertion such that it will always take O(1) time. Stack deletion operation relies on the fact that we can only remove the top element from the stack. Other operations on the stack include the search operation to help us sort our stack whenever a new value has pushed into it. This will help place the new value in the appropriate position within the list. A search operation in a stack is generally regarded as O(n) if a stack has n elements, however, since we are restricting the size of stack to 3, the search operation will always be O(1) instead.

To conclude, the full running time of our sorting algorithm will be O(1 + n*n) = O(n^2).

Data Structure

Our choice for data structures are dictionaries and stacks. Dictionaries will be used to store the complete set of stock tickers as well as their associated values, whereas a stack will hold the top 3 values at any given day. The stack's lowest value will always be compared to the key values as they are being added/updated. The stack will always have a size of 3, that makes comparisons to it have a time complexity of O(1).

The reason dictionaries/maps are chosen as our main data structure is because they allow us to simply store keys and values associated with different keys. Through Python, maps or chain maps are used to manage multiple dictionaries as one unit. Combined dictionaries will have key and value pairs in a specific sequence while eliminating any duplicate keys. In our case, keys will represent the stock ticker name i.e - $XYZ, whereas the values will represent the given/calculated score assigned to each stock symbol as they are collected i.e - 3 (resembling 3 mentions over a 100 posts). Implementation of a chainmap on python requires us to import the collections library. Using a dictionary will make it easier for us to add unique stock symbols strings (without duplicates), and better accessibility of values/scores as they relate to different stock symbols.

Finally, a stack is also used within our algorithm to constantly keep track of the top 3 (in terms of value/score) in a given date range of posts. Stacks are a simple data structure that focuses on the element on the top of the stack i.e; the element that was last added to the stack. This is also known as the Last In-First Out (LIFO) feature. In order to keep track of the top 3 results of our key values, the size of the stack will have a constant value of 3 (this value can be increased if we wish to look at more results, e.g. top 5 instead or top 3). A stack with a fixed size is advantageous as it allows us to compare the top value fast, and sorting the stack can also be done quickly if needed. If we wish to replace a value on the dictionary with the top element in the stack, we generally only need to perform the pop (top element) then push (new value that is presumably larger) operations that are afforded by this data structure. In order for us to implement the stack data structure in python, we must import the Stack library.

Unexpected Cases/Difficulties

There are multiple different unexpected difficulties encountered in the implementation of the algorithm. One of the major hurdles we faced was that some of the posts that were scraped by the algorithm were not actually present on the wallstreetbets subreddit. After some examinations, we concluded that the code was also scraping posts that were deleted on the subreddit. To solve this issue, we added a checking condition so that the implemented code would disregard posts that were deleted or removed. Another difficulty we had to solve was that the code would at some point encounter a subreddit post with no body, and since the algorithm is supposed to scrape the body of these subreddits, the implemented code would crash and return an error message. We had to again include a checking condition so that the code would only scrape posts that actually have a body.

The original design of the pseudocode included that designed code would scrape posts from the wallstreetbets subreddit, returning from each post different words that started with a "$" character. However, the issue with that was that the algorithm, aside from stock tickers, was also returning irrelevant words such as monetary prices (e.g $2.0). In order to solve that issue, we added a checking condition to the code, so that it would also look at the following character after a "$" and would only return words that had a letter after the "$".

Lastly, we noticed that some stock tickers were mentioned in posts without a "$" character (e.g GME instead of $GME). We concluded that this was an issue that was too big to solve and that it would be considered as a limitation to our algorithm.

Task Separation and Responsibilities

Tahmeed: Helped formulate problem into algorithmic problem. Broke down the data processing of the algorithm. Write project formulation and milestone abstract.

Barret: Helped formulate problem into algorithmic problem. Elucidated algorithm's proof of correctness, the implementation of the stack data structure and how every insertion of an element will run through (LIFO).

Omar: Data Structure implementation and fundamental understanding of Stack within the milestone. Time complexity of algorithm.

Kenji: Unexpected cases/difficulties and implemented the first algorithm from the first milestone.