Programming Assignment 3 — Stock Market

In hindsight all actions on the stock market would have been easy, some say. Then we would have known which stocks to buy and sell at which moment. In this assignment, we will see how easy or hard this is, when dealing with it computationally.

Problem Statement

We consider a stock trader who at day 0 has a given budget $B0$. Each forthcoming day, he/she can decide whether they wants to buy or sell stocks. That can be done for the bid and load prices of that stock of that day. For simplicity, we assume that the bid and load prices of all stocks are equal, i.e., the trader can buy stocks for the same stocks as they can sell them for. To spread the risk, the trader does not want to own too many stocks of a given company. They want to own at most one stock per company.

Typically, the trader will often also have an amount of money that is not invested in stocks. This amount can not be negative, as such the trader can not always buy all stocks they want. The amount of money is stored in the bank, and as such it will gather interest. On a certain day, the trader wants to sell all their stocks. Their goal is to have a maximum amount of money at this point.

Notation

The amount of money the trader starts with is called $B0$. The number of stocks which the trader can buy is called n, and $1 \leq n \leq 8$. A stock i has on day $t \geq 0$ a price of $\texttt{price}(i, t)$. The price remains the same over the whole day. The interest factor is given and remains constant for the entire period. If the interest is 2%, the interest factor is 1.02, and the trader will obtain the amount of money that is in the bank times 1.02[1].

We will denote the maximum amount of money that the trader can have in the bank on day t, along with a stock portfolio p, as $\texttt{amount}(p, t)$. As such, we will start with $\texttt{amount}(\emptyset, 0) = B0$ (where $\emptyset$ represents the empty portfolio) and we are interested in computing $\texttt{amount}(\emptyset, t_e)$, where t_e is the end of the investment period.

Stock portfolio

The portfolio of stocks that the trader owns can be represented as a bitstring. Suppose that there are 4 companies to invest in, then that would result in a bitstring of length 4. If the ith bit is True, that means that stock i consists in the portfolio, and if the ith bit is False that means that stock i is not contained in the portfolio. Each portfolio can also be represented as an integer index, where the 0th portfolio consists of none of the stocks, and the $2^n - 1$th portfolio consists of all stocks.

In python code, any integer (portfolio index) can be transformed into a bitstring with the command `bstr = "{0:b}".format(portfolio)`. As such, `bstr[0]` represents whether stock $n - 1$ is in `portfolio`, and `bstr[len(bstr)-1]` represents whether stock 0 is in `portfolio`.

We provided a template program. It consists of two files:

- `stock_market.py`: The main assignment file.

[1] Indeed, the interest is not expressed as a percentage, but as the multiplication factor.

Table 1: Stock portfolios

idx	bitstring	stock 0	stock 1	stock 2	stock 3
0	0000	✗	✗	✗	✗
1	0001	✓	✗	✗	✗
2	0010	✗	✓	✗	✗
3	0011	✓	✓	✗	✗
4	0100	✗	✗	✓	✗
5	0101	✓	✗	✓	✗
6	0110	✗	✓	✓	✗
7	0111	✓	✓	✓	✗
8	1000	✗	✗	✗	✓
	...				
15	1111	✓	✓	✓	✓

- `stock_market_test.py`: A Unit Test file.

The file `stock_market.py` contains various functions that are not implemented yet. Read the written documentation about these functions, and implement these. Most of these functions require less than 10 lines of code. None of the functions require more than 20 lines of code. Do not change the headers of the functions provided. You are allowed to add functions yourself, if you feel that that makes it easier. Make sure to document these consistently.

The file `stock_market_test.py` consists of several test functions. We provided these functions to help programming and testing the code. By running the following command, the unit tests can be executed:

```
python -m unittest stock_market_test.py
```

Make sure to have the right packages installed (numpy). Feel free to add more unit tests. Do not alter the unit tests that are provided. If your program does not succeed on all unit tests that are provided, likely, there is still a problem with your code. Make sure that all other unit tests succeed, before submitting the code.

Additionally, also hand in a file named `test_private.py`. Here, you can create additional unit tests to verify the working of your program. Write at least 3 additional unit tests, and hand these in along with the assignment.

Also, keep in mind that all unit tests should be able to run within a matter of seconds, on any computer.

Report

Write a report in LaTeX(at most 3 pages), addressing the following points/research questions:

- Describe the recursive formulation of this problem
- Is it possible to deduct from this implementation what would be the optimal strategy? Describe (in a few words) how to adapt your program. (Note that there is also a Python function that can be programmed for this).
- What is the time complexity of this method? Use Θ-notation, and include the number of days t_e and the number of stocks n.

- Same for the space complexity.
- Can the space complexity be further reduced? If yes, what will the space complexity be, and what are the consequences of this?
- Summary and Discussion. Always end a report with some summarizing remarks.