Assignment 7

Bhavya[U21CS100], Garvit[U21CS089], Swayam[U21CS079]

April 24, 2023

1 Hardware Specifications

Machine Type	Laptop
CPU	11th Gen Intel(R) Core(TM) i5-1135G7
Frequency	2.40 GHz
L1 Cache	320 KB
L2 Cache	5.0 MB
L3 Cache	8.0 MB
RAM	16 GB
Operating System	Windows 11 Home
Implementation Language	C++
Compiler	g++ 12.1.0
Libaries Used	iostream chrono vector

2 Buy and Sell Stock

2.1 Question

The following assignment you can do as a group (maximum size: 4 students) or indi- vidually. Only one student from the group should upload the assignment at Google
classroom. On the first page of submission, you have to clearly mention the names and roll numbers of students of the group. The assessment will be based on the novelty of the problem (how different the computation problem is from problems submitted by other groups.), how interesting the problem is, how elegantly you have written the problem statement, pseudocode, and analysis, how elegantly you implemented and analyzed the algorithms, and the viva voce.

2.2 Input Format:

Matrix Size - n
Position of the Statue - x y

3 Solution

3.1 Algorithm

1. Define a recursive function named c that takes four arguments: a. i, an integer representing the current index in the prices list. b. buy, an integer representing whether the current state is a "buy" state (1) or "not buy" state (0). c. prices, a vector of integers representing the stock prices at each index. d. dp, a 2D vector of integers representing the memoization table.

   The function returns an integer representing the maximum profit that can be earned from the current state.

2. Check if the index i is greater than or equal to the length of the prices list. a. If yes, return 0 as there are no more stocks left to buy or sell.

3. Check if the memoization table dp at the index [i][buy] is not equal to -1. a. If yes, return the value stored in the memoization table as the result.

4. Define a variable ans to store the maximum profit that can be earned from the current state.

5. Check if the current state is a "buy" state (buy = 1). a. If yes, set ans to the maximum value of: i. -1 times the stock price at index i, added to the result of the c function called with the arguments (i+1, 0, prices, dp). This represents the case where the stock is bought at index i, and then sold later at some point after i+1. ii. The result of the c function called with the arguments (i+1, 1, prices, dp). This represents the case where the stock is not bought at index i, and the state remains a "buy" state.

   b. If no, set ans to the maximum value of: i. The stock price at index i, added to the result of the c function called with the arguments (i+1, 1, prices, dp). This represents the case where the stock is sold at index i, and then the state changes to a "not buy" state. ii. The result of the c function called with the arguments (i+1, 0, prices, dp). This represents the case where the stock is not sold at index i, and the state remains a "not buy" state.

6. Store the value of ans in the memoization table dp at the index [i][buy].

7. Return ans as the result of the function.

8. Define a function named maxProfit that takes one argument: a. prices, a vector of integers representing the stock prices at each index.

   The function returns an integer representing the maximum profit that can be earned from buying and selling stocks based on the given prices.

9. Create a 2D vector dp of size n x 2, where n is the length of the prices vector, and initialize all elements to -1.

10. Call the c function with the arguments (0, 1, prices, dp), which represents starting at the first index with a "buy" state.

11. Return the value returned by the c function as the result of the maxProfit function.

12. Define a vector named prices with some initial stock prices.

13. Create an instance of the Solution class named s.

14. Call the maxProfit function with the prices vectore as an argument, and store the result in a variable named max-profit

15. Print the string "Max Profit:" followed by the value of the max-profit variable.

3.2 Psuedo Code

Greedy Approach

```
        function c(i: int, buy: int, prices: List[int], dp: List[List[int]]) -> int:
        if i >= len(prices):
            return 0
        if dp[i][buy] != -1:
            return dp[i][buy]
        ans: int
        if buy:
            ans = max(-1 * prices[i] + c(i + 1, 0, prices, dp), c(i + 1, 1, prices, dp))
        else:
            ans = max(prices[i] + c(i + 1, 1, prices, dp), c(i + 1, 0, prices, dp))
        dp[i][buy] = ans
        return ans

    function maxProfit(prices: List[int]) -> int:
        dp: List[List[int]] = [[-1 for _ in range(2)] for _ in range(len(prices))]
        return c(0, 1, prices, dp)

prices: List[int] = [7, 1, 5, 3, 6, 4]
s: Solution = Solution()
max_profit: int = s.maxProfit(prices)
print("Max Profit:", max_profit)
```

3.3 C++ Code

```cpp
#include <iostream>
#include <vector>

using namespace std;

class Solution {
public:
    int c(int i, int buy, vector<int>& prices, vector<vector<int>>& dp) {
        if (i >= prices.size()) {
            return 0;
        }
        int ans;
        if (dp[i][buy] != -1) {
            return dp[i][buy];
        }
        if (buy) {
            ans = max(-1 * prices[i] + c(i + 1, 0, prices, dp), c(i + 1, 1, prices, dp));
        }
        else {
            ans = max(prices[i] + c(i + 1, 1, prices, dp), c(i + 1, 0, prices, dp));
        }
        dp[i][buy] = ans;
        return ans;
    }

    int maxProfit(vector<int>& prices) {
        vector<vector<int>> dp(prices.size(), vector<int>(2, -1));
        return c(0, 1, prices, dp);
    }
};

int main() {
    vector<int> prices = {7, 1, 5, 3, 6, 4};
    Solution s;
    int maxProfit = s.maxProfit(prices);
    cout << "Max Profit: " << maxProfit << endl;
    return 0;
}
```