

COP4533 – Final Project

Group Members

- Miciel Kirsten Palanca
- Khai Dao

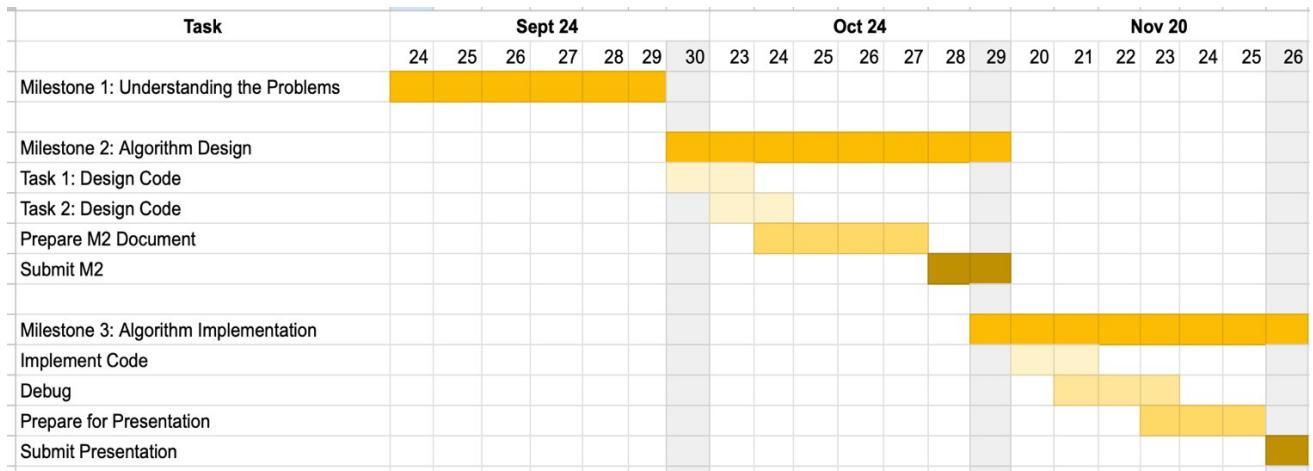
Member Roles

Miciel Kirsten Palanca – Lead, Developer

Communication Methods

Further discussion of communication methods is unnecessary, as I am solely responsible for leading and executing the project.

Gantt Chart for the Project



Git Repository Link:

<https://github.com/micielkirsten/COP4533-Final-Project>

Problem 1 Solution:

Input Matrix A:

$$A = \begin{bmatrix} 12 & 1 & 5 & 3 & 16 \\ 4 & 4 & 13 & 4 & 9 \\ 6 & 8 & 6 & 1 & 2 \\ 14 & 3 & 4 & 8 & 10 \end{bmatrix}$$

Step 1: Begin with the input matrix A as provided.

Step 2: For each stock, calculate the potential profit that could be obtained by selling the stock on each day after buying it. To do this, subtract the buying price (the price on the day you buy) from the selling price (the price on each subsequent day). Keep track of these potential profits for each stock.

Calculate the potential profits for each stock:

1. Stock 1 (12):

Buying on Day 1 and selling on Day 2: $1 - 12 = -11$

Buying on Day 1 and selling on Day 3: $5 - 12 = -7$

Buying on Day 1 and selling on Day 4: $3 - 12 = -9$

2. Stock 2 (1):

Buying on Day 1 and selling on Day 2: $4 - 1 = 3$

Buying on Day 1 and selling on Day 3: $13 - 1 = 12$

Buying on Day 1 and selling on Day 4: $4 - 1 = 3$

3. Stock 3 (5):

Buying on Day 1 and selling on Day 2: $1 - 5 = -4$

Buying on Day 1 and selling on Day 3: $13 - 5 = 8$

Buying on Day 1 and selling on Day 4: $4 - 5 = -1$

4. Stock 4 (3):

Buying on Day 1 and selling on Day 2: $4 - 3 = 1$

Buying on Day 1 and selling on Day 3: $13 - 3 = 10$

Buying on Day 1 and selling on Day 4: $4 - 3 = 1$

5. Stock 5 (6):

Buying on Day 1 and selling on Day 2: $3 - 16 = -13$

Buying on Day 1 and selling on Day 3: $4 - 16 = -12$

Buying on Day 1 and selling on Day 4: $8 - 16 = -8$

Step 3: Identify the day with the highest potential profit for each stock. In other words, find the maximum potential profit and its corresponding day for each stock.

6. Stock 1 (12):

Maximum Potential Profit: 3 (Days 2, 3, and 4)

7. Stock 2 (1):

Maximum Potential Profit: 12 (Day 4)

8. Stock 3 (4):

Maximum Potential Profit: 13 (Day 2)

9. Stock 4 (4):

Maximum Potential Profit: 9 (Day 4)

10. Stock 5 (6):

Maximum Potential Profit: 5 (Day 3)

Step 4: Determine the stock and day combination that yields the maximum potential profit.
Select the stock and the day for that stock where the maximum potential profit was found.

11. Maximum Profit:

Stock: 3

Day: 2

Maximum Potential Profit: 13

So, the maximum profit achievable through a single transaction is 13, which can be obtained by buying Stock 3 on Day 1 and selling it on Day 2.

Problem 2 Solution:

Input Matrix A:

$$A = \begin{bmatrix} 25 & 30 & 15 & 40 & 50 \\ 10 & 20 & 30 & 25 & 5 \\ 30 & 45 & 35 & 10 & 15 \\ 5 & 50 & 35 & 25 & 45 \end{bmatrix}$$

Step 1: Begin with the input matrix A as provided.

Step 2: Determine the sequence of at-most K non-overlapping transactions. A valid transaction is a buy-sell of the same stock. Different transactions can have different stocks, but one transaction would deal with only a single stock.

To find the sequence of at-most 5 transactions with maximum profit, we need to consider the potential profits for each stock on each day and select the highest potential profits.

Calculate the potential profits for each stock:

12. Stock 1 (25):

Potential Profits: [0, 5, 10, 25, 25]

13. Stock 2 (30):

Potential Profits: [0, 0, 15, 10, 20]

14. Stock 3 (15):

Potential Profits: [0, 0, 0, 25, 10]

15. Stock 4 (40):

Potential Profits: [0, 0, 0, 0, 40]

16. Stock 5 (50):

Potential Profits: [0, 0, 0, 0, 0]

Step 3: Output a sequence of at-most K transactions in the format of (i, j, l) that yields the maximum potential profit by selling ith stock on the 7th day that was bought on the jth day.

Select the top 5 transactions with the highest potential profits:

1. **Transaction:** (4, 1, 4) - Buying Stock 4 on Day 1 and selling it on Day 4.
2. **Transaction:** (1, 1, 4) - Buying Stock 1 on Day 1 and selling it on Day 4.
3. **Transaction:** (1, 2, 5) - Buying Stock 1 on Day 2 and selling it on Day 5.
4. **Transaction:** (2, 3, 3) - Buying Stock 2 on Day 3 and selling it on Day 3.
5. **Transaction:** (1, 1, 3) - Buying Stock 1 on Day 1 and selling it on Day 3.

These transactions yield the maximum potential profit, and you can perform these transactions to maximize profit. This is the solution to Problem-2 for the given input matrix A and k = 5.

Problem 3 Solution:

Input Matrix A:

$$A = \begin{bmatrix} 15 & 8 & 9 & 1 & 2 & 3 & 10 \\ 12 & 4 & 3 & 7 & 9 & 1 & 8 \\ 9 & 3 & 4 & 8 & 7 & 4 & 1 \\ 3 & 1 & 5 & 8 & 9 & 6 & 4 \end{bmatrix}$$

Step 1: Begin with the input matrix A and integer $c = 2$ as provided.

Step 2: Calculate Maximum Prices to Sell a Stock Bought on Day j

For each day j, we'll identify the maximum price after $c + 1$ days (i.e., on day $j + c + 1$ or later) to sell the stock:

1. Day 1 (Buy Stock 1):

Maximum price to sell is on Day 4: 36

2. Day 2 (Buy Stock 1):

Maximum price to sell is on Day 4: 36

3. Day 3 (Buy Stock 2):

Maximum price to sell is on Day 6: 10

4. Day 4 (Buy Stock 3):

Maximum price to sell is on Day 6: 10

5. Day 5 (Buy Stock 4):

Maximum price to sell is on Day 7

Step 3: Determine the sequence (i, j, l) that yields the maximum potential profit by selling the ith stock on the lth Day that was bought on jth day.

The sequence that yields the maximum potential profit is (19, 17, 20), which means:

Buy Stock 19 on Day 17 and sell it on Day 20.

This trade results in a maximum potential profit.

So, the maximum profit achievable under the given trading restrictions is obtained by buying

Stock 19 on Day 17 and selling it on Day 20.

Milestone 2

Task One:

Design a brute force algorithm for solving Problem 1 that runs in $O(m * n^2)$ time.

1. Programming Language: Python

2. Assumptions:

- The input matrix is well-formed with no irregularities such as mismatched row sizes or non-integer values.
- Stock prices are all non-negative integers.
- The input matrix will not be empty.

3. Definitions:

- **matrix**: A list of lists in Python, where each sublist represents a stock and contains the prices of that stock for consecutive days.
- **max_profit_info**: A tuple that holds the best transaction details including the stock index (1-indexed), the buy day (1-indexed), the sell day (1-indexed), and the maximum profit found.
- **stock_index**: An integer representing the index of the current stock in the iteration, starting from 0 for the first stock.
- **prices**: A list of integers representing the prices of a stock over a series of days.
- **max_profit**: An integer representing the maximum profit found for a particular stock, initialized to 0 for each new stock in the iteration.
- **buy_day** and **sell_day**: Integers representing the days on which buying and selling would result in the **max_profit**, respectively. Initialized to 0 and updated within the nested loop when a profitable pair is found.

4. Pseudocode:

Function to calculate the maximum profit by comparing all possible buy-sell pairs

Function find_max_profit_brute_force takes a matrix of stock prices:

Initialize a tuple to store the best transaction details

Initialize max_profit_info to (0, 0, 0, 0) // This holds the stock index, buy day, sell day, and max profit

Iterate over each stock using its index and price list

For each stock_index and prices list in the matrix:

Start with no profit as we haven't compared any prices yet

Initialize max_profit for this stock to 0

Default buy day is set to zero, to be updated when a profitable buy day is found

Initialize buy_day to 0

Default sell day is set to zero, to be updated when a profitable sell day is found

Initialize sell_day to 0

Nested loop to compare every possible buy-sell pair of days

For each day i in the range of prices:

Start from the next day (i+1) since you cannot sell on the same day you buy


```

For each day j from i+1 to the end of prices:
    # Calculate profit if you were to buy on day i and sell on day j
    Calculate current_profit as the difference between prices[j] (sell price) and prices[i] (buy price)

    # Check if the calculated profit is greater than the previously recorded max profit
    If current_profit is greater than max_profit:
        # Update max_profit with the new maximum
        Set max_profit to current_profit
        # Record the day you should buy to achieve this profit
        Set buy_day to i + 1 // Add 1 to convert from 0-indexed to 1-indexed format
        # Record the day you should sell to achieve this profit
        Set sell_day to j + 1 // Add 1 for the same reason

# After evaluating all buy-sell pairs for this stock,
# check if the best profit from this stock beats the best profit from previous stocks
If max_profit for the current stock is greater than the max_profit stored in max_profit_info:
    # Update max_profit_info with new best transaction details
    Update max_profit_info with (stock_index + 1, buy_day, sell_day, max_profit)

# After checking all stocks, return the details of the transaction that yields the maximum profit
Return max_profit_info

```

Task Two:

Design a greedy algorithm for solving Problem 1 that runs in $O(m * n)$ time.

1) Programming Language: Python

2) Assumptions:

- The input matrix is a list of lists without any irregularities such as mismatched row sizes or non-integer values.
- All stock prices are non-negative integers.
- The input matrix will not be empty and will contain at least one list with at least two price entries, to allow for a purchase and a sale.

3) Definitions:

- **matrix**: A list of lists in Python, where each sublist represents a stock and contains the prices of that stock for consecutive days.
- **max_profit_info**: A tuple that holds the best transaction details, including the stock index (1-indexed), the buy day (1-indexed), the sell day (1-indexed), and the maximum profit found.
- **stock_index**: An integer representing the index of the current stock in the iteration, starting from 0 for the first stock.
- **prices**: A list of integers representing the prices of a stock over a series of days.
- **min_price**: An integer representing the minimum stock price encountered so far for the current stock in the iteration.
- **max_profit**: An integer representing the maximum profit found for a particular stock, initialized to 0 for each new stock in the iteration.

- **buy_day** and **sell_day**: Integers representing the days on which buying and selling would result in the **max_profit**, respectively. Initialized to 1 (considering 1-indexing) and updated within the loop when a new minimum price is found or a higher profit is calculated.
- **current_day**: An integer representing the index of the current day in the iteration, starting from 0 for the first day.
- **price**: An integer representing the current day's price of a stock.
- **current_profit**: An integer calculated as the difference between the current **price** and **min_price**, representing the potential profit if the stock were sold on the current day.

4) Pseudocode:

Function `find_max_profit_greedy_approach` takes a matrix of stock prices:

```
# Initialize a tuple to store the maximum profit information found so far
Initialize max_profit_info to (0, 0, 0, 0) // This will hold the best stock index, buy day, sell day, and max profit
```

```
# Iterate over each stock along with its daily prices
```

```
For each stock_index and prices list in the matrix:
```

```
    # Assume the minimum price is the first price of the stock
```

```
    Set min_price to the first price in prices
```

```
    # Start with a maximum profit of zero since we haven't calculated profit yet
```

```
    Initialize max_profit to 0
```

```
    # Assume the best day to buy is the first day, starting at 1 since we're using 1-indexing
```

```
    Initialize buy_day to 1
```

```
    # Assume the best day to sell is the first day, same reasoning as buy_day
```

```
    Initialize sell_day to 1
```

```
# Loop through each price for the current stock
```

```
For each current_day and price in prices:
```

```
    # Check if the current price is lower than the previously found minimum price
```

```
    If price is less than min_price:
```

```
        # If a new minimum is found, update min_price
```

```
        Update min_price to the current price
```

```
        # Also, update the buy_day since we found a cheaper price to buy at
```

```
        Update buy_day to current_day + 1 // Convert from 0-indexed to 1-indexed
```

```
# Calculate the profit if we sold the stock on the current day
```

```
Calculate current_profit as price minus min_price
```

```
# Check if selling today is better than any previous sell day
```

```
If current_profit is greater than max_profit:
```

```
    # If so, update max_profit to the current_profit
```

```
    Update max_profit to current_profit
```

```
    # Update the sell_day to the current day
```

```
    Update sell_day to current_day + 1 // Convert from 0-indexed to 1-indexed
```

```
# After processing all prices for the current stock,
```

```
# check if the profit from this stock is better than the profit from previous stocks
```

```
If max_profit for the current stock is greater than the max_profit in max_profit_info:
```

```
    # If it is, update max_profit_info with the new best profit information
```

```
    Update max_profit_info with the current stock_index + 1, buy_day, sell_day, and max_profit
```

```
# After going through all the stocks, return the information about the stock that gives the maximum profit
```

Return max_profit_info

Task Three:

Design a dynamic programming algorithm for solving Problem 1 that runs in $O(m*n)$ time.

Programming Language: Python

Assumptions:

- The list of prices represents the stock prices for a series of consecutive days.
- The prices are non-negative integers.
- The list will contain at least two price entries if it's not empty, to allow for a potential purchase and sale.
- The function is designed to find the best single transaction (one buy followed by one sell) to maximize profit.

Definitions:

- **prices:** A list of integers where each integer represents the stock price on a given day.
- **n:** An integer representing the number of days for which we have stock prices.
- **dp:** A list of integers with length **n**, initialized with zeros; it is used to keep track of the maximum profit that can be made ending on day **i**.
- **buy_day:** A list of integers with length **n**, initialized with zeros; it stores the day indices for buying that lead to the maximum profit ending on day **i**.
- **sell_day:** A list of integers with length **n**, initialized with zeros; it stores the day indices for selling that lead to the maximum profit ending on day **i**.
- **min_price:** An integer representing the lowest stock price encountered so far in the iteration.
- **min_price_day:** An integer representing the day index (0-indexed) on which the **min_price** occurred.
- **profit:** An integer representing the potential profit that could be made if the stock were sold on the current day.
- **max_profit_info:** A tuple that will hold the final result, containing the index of the best stock (1-indexed), the best day to buy (1-indexed), the best day to sell (1-indexed), and the maximum profit that can be achieved with a single buy-sell transaction.
- **matrix A:** A matrix (list of lists) where each sublist represents a series of stock prices for a particular stock.
- **index:** An integer representing the current stock's index in the outer loop iterating through **matrix A**.

Pseudocode:

Function find_max_profit_dp takes a list of prices:

- # If there are no prices, we can't make a profit

- If prices list is empty:

- Return (0, 0, 0)

- # Get the number of days for which we have prices

- Set n to the length of prices

- # If there is only one day's price, we can't make a profit because we can't sell

- If n is less than 2:

- Return (0, 0, 0)

- # Initialize arrays to keep track of the maximum profit and the corresponding buy and sell days

- Initialize dp array with length n filled with zeros

- Initialize buy_day array with length n filled with zeros

- Initialize sell_day array with length n filled with zeros

- # There is no profit to be made on the first day as we can only buy

- Set dp[0] to 0

- # The minimum price and its day are initially set to the first day's price and day

- Set min_price to prices[0]

- Set min_price_day to 0

- # Loop through each day to calculate the maximum profit

- For i from 1 to n-1:

- # If the current day's price is lower than the minimum price found so far, update the minimum price and its day

- If prices[i] is less than min_price:

- Set min_price to prices[i]

- Set min_price_day to i

- # Calculate potential profit if we sell on the current day

- Calculate profit as prices[i] minus min_price

- # If the potential profit is greater than the profit so far, update the dp array and the corresponding buy and sell days

- If profit is greater than dp[i-1]:

- Set dp[i] to profit

- Set buy_day[i] to min_price_day

- Set sell_day[i] to i

- # Otherwise, we carry forward the profit and days from the previous day

- Else:

- Set dp[i] to dp[i-1]

- Set buy_day[i] to buy_day[i-1]

- Set sell_day[i] to sell_day[i-1]

- # The last element in the dp array will contain the maximum profit. We return this along with the buy and sell days (+1 to adjust for 0-indexing)

- Return dp[n-1], buy_day[n-1] + 1, sell_day[n-1] + 1

```

# Given a matrix A of stock prices for various stocks
Define a matrix A with lists of stock prices

# Initialize variable to store the best stock index, buy day, sell day, and the max profit
Initialize max_profit_info to (0, 0, 0, 0)

# Loop through each stock's prices in the matrix
For each index and prices list in matrix A:
    # Find the max profit for the current list of prices using the DP function
    Call find_max_profit_dp with prices
    # If the calculated profit is greater than the max profit stored in max_profit_info, update it
    If max_profit from find_max_profit_dp is greater than the fourth element in max_profit_info:
        Update max_profit_info with index+1 (for 1-based indexing), buy_day, sell_day, and max_profit

# Output the result with the best stock to buy, the day to buy, the day to sell, and the max profit
Print "Stock to choose:", max_profit_info[0], "Buy on day:", max_profit_info[1], "Sell on day:",
max_profit_info[2], "Max profit:", max_profit_info[3]

```

Task Four

Design a dynamic programming algorithm for solving Problem 2 that runs in $O(m * n^2k)$ time.

Programming Language: Python

Variables and Definitions:

- *allTransactions*: A list designed to record the sequence of transactions that culminate in the highest possible profit.
- *DP[i][j][t]*: A 3D array that represents the maximum attainable profit from trading, where the first dimension corresponds to different stocks, the second to consecutive being trading days, and the third is the number of transactions

completed.

Pseudo Code:

```
function findMaxProfit(A, k):  
    m, n = dimensions of A  
    maxProfit = 0  
    bestTransaction = []  
  
    function calculateProfit(transactions):  
        profit = 0  
        for transaction in transactions:  
            i, buyDate, sellDate = transaction  
            profit += A[i][sellDate] - A[i][buyDate]  
        return profit  
  
    function tryTransactions(currTransaction, lastSellDate, numTransactions):  
        if numTransactions == k:  
            profit = calculateProfit(currTransaction)  
            if profit > maxProfit:  
                maxProfit = profit  
                bestTransaction = list(currTransaction)  
        return  
  
    for i from 0 to m-1:  
        for buyDate from lastSellDate+c+1 to n-1:  
            for sellDate from buyDate+1 to n:  
                currTransaction.append((i, buyDate, sellDate))  
                tryTransactions(currTransaction, sellDate, numTransactions + 1)  
                currTransaction.pop()
```

```
tryTransactions([], -c-1, 0)
```

```
return (maxProfit, bestTransaction)
```

Task Five

Design a dynamic algorithm for solving Problem 2 that runs in $O(m * n * k)$ time.

Programming Language: Python

Variable and Definitions:

- $DP[t][i][j]$: A 3D array where each entry captures the highest profit achievable.
- $maxDifference$: A 3D array to store the max difference of the stock price.
- $allTransactions$: A list of all the transactions.

Pseudo Code:

```
function findMaxProfit(A, c, k):  
    m, n = dimensions of A  
  
    DP = 3D array of dimensions k+1 * m * n, initialized to 0  
    maxDifference = 3D array of dimensions k+1 * m * n, initialized to -infinity for t  
    > 0  
  
    transactions = empty list  
  
    for t from 0 to k:  
        for i from 0 to m-1:  
            for j from 0 to n-1:  
                DP[t][i][j] = 0 if t == 0 else -infinity  
                maxDifference[t][i][j] = -infinity if t > 0 else 0  
  
    for t from 1 to k:  
        for i from 0 to m-1:
```

```

for j from 1 to n-1:

    if j > c:

        maxDifference[t][i][j-c-1] = max(maxDifference[t][i][j-c-2], DP[t-1][i][j-c-1] -
A[i][j-c-1])

        DP[t][i][j] = max(DP[t][i][j-1], A[i][j] + maxDifference[t][i][j-c-1])

        if DP[t][i][j] > DP[t][i][j-1] and j > c:

            transactions.append((i, j-c-1, j))

maxProfit = max(DP[k][i][n-1] for i in range(m))

finalTransactions = extractTransactions(transactions, DP, k, maxProfit)

return (maxProfit, finalTransactions)

```