Project 2:
Exhaustive vs. Dynamic Programming

CPSC 335 - Algorithm Engineering

Nathaniel Gries - nathanelgries@csu.fullerton.edu
Nevan Nguyen - Nevanem@csu.fullerton.edu
Patrick Valera - patrickv@csu.fullerton.edu

Approach for Exhaustive Solution (Part A)

To get the stock combination that gives the maximum possible value based on the amount given, we analyze **all** possible combinations. For each stock and value pair, we consider two cases: buying the stock or not buying the stock. We then **recursively** explore the possibilities for the next stock. In each recursion call we return the max between adding or not adding the current stock.

Approach for Dynamic Programming Solution(Part B)

The maximum number of stocks that can be purchased can be found by the given set of stocks and the given amount of money needed to invest. The approach that is used needs to consider the following: whether to include or exclude the current stock. Our dp array is a 2D table, where dp[i][j] represents the maximum number of stocks that can be purchased with the first i stocks and remaining amount j. We fill up the table and once the table is filled the result will be at the bottom right - dp[N][Amount]

Time Complexity Analysis and Conclusion

Our exhaustive algorithm has to explore all combinations, which is inefficient for large values of N. It also has to recursively explore combinations and in doings so results to an O(N^2) Time complexity

On the other hand Our Dynamic Programming algorithm has a time complexity of O(N * amount), which is more efficient than the exhaustive search algorithm for larger input sizes. It uses a bottom-up approach to build the solution iteratively, avoiding redundant computations by storing intermediate results in a dp array.

GitHub Repo

GitHub Repo: https://github.com/NateGries1/CPSC335-Project2/tree/main

Screen Shots