Project 2: Exhaustive vs. Dynamic Programming

Fall 2023 CPSC 335.02/09 - Algorithm Engineering

Instructor: Ornella Irene Dsouza (odsouza@fullerton.edu)

# Abstract

In this project, you will design, implement and analyze an exhaustive search algorithm, dynamic programming algorithm for solving the same problem.

**The Stock Purchase Maximization Problem**

The problem seeks to maximize the number of stocks an investor may purchase, given a limited amount of available financial resources. The future values of these stocks are not considered at the time of purchase.

This problem can be mathematically represented as:

|  |
| --- |
| **Stock purchase maximation problem** |
| ***input****:* a set of items (, each having a number of stocks and a value is the total available investment sum.  ***output:*** ﻿maximum number of stocks, such that the total value of those stocks is at  most *M*  Such that |
|  |

Assume you are given a set of stocks in an array of arrays. Each entry in a subarray represents a company, with the number of stocks available and the cumulative value of all available stocks. You are also given a specific sum of money to invest. Buying a fraction of any item is not allowed. Your task is to design an algorithm to determine the maximum number of stocks you are able to purchase. The cumulative value of all selected stocks must be equal to or less than the given sum.

A sample is given below. It contains 4 items corresponding to company 0, 1, 2 and 3. Each company has attributes, where = numbers of trading stocks and monetary value of the trading stocks. Purchasing a fraction of trading stock is not allowed.

Sample input

N = 4 # size of the input array

Stocks\_and\_values = [ [1, 2], [4, 3], [3, 6], [6, 7]]

Amount = 12

Sample Output = 11 # 1+4+6 at index 0,1,3, sum of the values at these indices = 2+3+7<=12

N = 4

Stocks\_and\_values = [ [3, 2], [4, 3], [5, 3], [6, 7]]

Amount = 10

Sample output = 12 # 3+4+5 at index 0,1,2, sum of the values at these indices = 2+3+3<=12

Constraints:

0<=N<=100000

[x,y]>0

The Exhaustive Search Approach (Part A)

An exhaustive search algorithm can be used to solve this problem. The approach evaluates the number of stocks and value of all possible subsets, then selects the subset with the highest value that is still under the available fund limit. It recomputes combination at each state. This approach provides an effective, but expensive solution.

﻿def stock\_maximization (M, items):

best = None

for candidates in <stocks\_combinations>(items):

if verify\_combinations(M, items, candidate):

if best is None or total\_value(candidate) > total\_value(best)

best = candidate

return best

This approach was extensively discussed in class.

The Dynamic Programming Approach (Part B)

This problem can also be solved using the dynamic programming approach.  It uses a top-down dynamic programming to handle the overlapping subproblems. Since there are two changing values, the resultant amount and the current index, a two-dimensional array can be used to store the results of all the solved sub-problems.

To calculate the maximum value obtainable with the selection of item a comparison of its cost is made with the total purchase capacity. If item cost more than the available investment sum, it cannot be used. If a new candidate potentially increases the value of purchase and is less or equal to the maximum available sum, it is selected. The approach evaluates the number of stocks and value of all possible subsets, then selects the subset with the highest value that is still under the weight limit.

# Implementation

# To Do

1. Create a Readme file and include your name(s) and email address(es). The Readme file should also contain instructions on how to run your program.
2. Study the sample input and output above.
3. Develop and implement the algorithm for part A and part B using either Python or C++.
4. Run your code using different data inputs/sample arrays (minimum 10)
5. Mathematically analyze each code, your algorithms and observations and represent time complexity in Big O notation. Step count and proof is not required for this project.
6. Upload the code on github <keep the repository private until due date>
7. Submit a report with group information, time complexity and logic used for Part A and B, write a paragraph explaining which approach is better and why? Attach github link in the report.
8. Submit report, code, Readme, input and output files in the zip folder on Canvas

# Grading Rubric

The suggested grading rubric is below.

1. Part A. Exhaustive Implementation Algorithm = 50 points, divided as follows:
   1. Clear and complete code = 20 points
   2. Successful compilation = 10 points
   3. Produces accurate result = 10 points
   4. Time complexity analysis in Big O notation = 10 points
2. Part B: Dynamic programming algorithm design and implementation = 50 points, divided as follows:
   1. Clear and complete code = 20 points
   2. Successful compilation = 10 points
   3. Produces accurate result = 10 points
   4. Time complexity analysis in Big O notation = 5 points
   5. Comparison of performance with exhaustive algorithm = 5 points

Ensure your submissions are your own works. Your submissions will be checked for similarities using a software.

# Submitting your code

Submit your files to the Project 2 on Canvas. It allows for multiple submissions. You can submit your files as a zip folder. One submission per group is enough.

# Deadline

The project deadline is Nov 15, 11:59 pm on Canvas.

Penalty for late submission is as stated in the syllabus. Projects submitted more than 48 hours after the deadline will not be accepted.