Data Structures and Algorithms Design

Assignment - 2 PS16 - Paint Production

Academic Year 2020-21

Group - 240

Section - 3

**DESIGN DOCUMENTATION**

## Introduction

### Purpose

The purpose of this design document is to elaborate on the technical implementation of solving the Paint Production optimisation problem for India Paints factory to help increase their profits and simultaneously maximise their production keeping it within production capacity limits. The problem is to be approached through a recursive solution using Dynamic programming on Python.

### Problem Statement

Major highlights of the given problem are as follows :

1. India Paints factory produces paints in various colours with a single production line that can produce only one colour in a given moment.
2. Each colour has its on production cost and associated profits which would be provided in an input text file.
3. The factory has a maximum capacity of production fixed at 1000 litres per day.
4. Considering all the above information, an algorithm has to be defined to select the paints to be funded to maximum profits. The algorithm also needs to return the profit realised and also the appropriate paints chosen for production.

### Solution overview

The solution is based on the Recursive Dynamic Programming algorithm. It is developed using the recursion technique with memoization to avoid repeating computational steps by storing each previous computation in an array/list. This solution is completely developed on Python. Once the recursion algorithm reaches its stopping criterion, the optimised profit is generated. Thereby, using the auxiliary list, which is used to induce memoization, the paints which are to be funded to realise the optimised profit are deduced and returned.

In addition to the algorithm design, 2 auxiliary functions have been used; one to read input and parse data - parse\_input() and the other to write back the results into the output file - write\_output()

As per the above design, the solution is developed and run time analysis has been performed on the same to get an insight into the time efficiency of the program.

## Design and Analysis

### Design Methodology

As an assumption, it is taken that there is no dependency of one paint on the other with respect to production.

As discussed in the solution overview, a recursive approach is taken to solve the optimisation problem. In solving recursive problems, there can be cases where the same subproblem is encountered multiple times in different parts of the recursion tree. In naive recursive solutions, these subproblems are solved each time adding to the computational inefficiency of the algorithm.

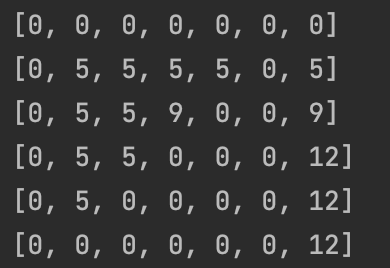
Thus to overcome the above problem, an auxiliary 2D list is being used in this solution to store the intermediate results thereby reducing overall number of computations. This method uses the top-down manner of solving the problem.

Figure 1

As shown in the above figure for a sample case of optimisation with 5 items (Figure 1), the auxiliary list stores the computation of profits at each level thereby aiding us to reach the final answer quickly.

Once the maximum optimised profit has been calculated using the above list (i.e. the last entry in the table, 12 in the example), there is an additional step added to backtrack on the table to find the paints corresponding to the profit calculated and store it in a list. Finally, all the required results are formatted and stored in a text file.

### Run-Time Analysis

As part of run-time analysis, below is a detailed view on the different functions being used to modularise the complete solution.

*\*Considering that number of paint inputs=****n****, maximum-capacity of production=****m***

1. **parse\_input(file\_path)** - This function reads the input file and iterates over each line in the file and parses the inputs as separate lists. The pseudo code of the program can be given as follows:

for i in input\_line\_split:

i[0] => paint-codes

i[1] => capacity list

………

Thus, since it iterates over the number of paint codes available, it runs in time complexity of **O(n).**

2. **run\_optimisation()** - This is the function which provides the overall hindsight on the algorithm implemented. It has 2 major functionalities : i. to run the helper function optimiser\_func() and calculate optimised profit. ii. to identify which paints need to be produced to realise the profit.

The optimiser\_func() implements the recursion with memoization algorithm to solve the optimisation problem. Since, we are using an 2D auxiliary list to avoid redundant computation, this runs in **O(n\*m)**.

The second operation iterates over the total number of paints(i.e. n), thus it is seen that it can be neglected in the overall time complexity expression i.e. O(n(m+1)) ~ O(nm).

### Alternate way of modelling

Since optimisation problems can be modelled in multiple ways, an alternate way of modelling this problem was explored as below.

*\*Considering that number of paint inputs=****n****, maximum-capacity of production=****m***

TABULATION AND ITERATIVE METHOD

Instead of a recursion approach to Dynamic programming, the solution can be arrived at using an iterative module and maintaining a similar tabular structure. Iterating over the (length\*breadth) of the auxiliary table, we can arrive at the optimised profit using the bottom-up approach in the iterative manner. This solution was also implemented and the time complexity was analysed. The time complexity of the iterative implementation was also seen to be **O(n\*m)** and a space complexity of **O(n\*m)**, since we were utilising a 2D list/array to store temp results.

However, with respect to space complexity and memory consumption, this algorithm can be modified and made to work at a space complexity of **O(m)** by using lower dimensional lists for result storage(1D array).

Also, it was seen that process for extracting the individual paint codes to be funded can be included with the algorithm itself to help reduce time taken for the process but the time-complexity would still remain to be O(n\*m).