**Employee Tracker – Binary Search Tree Design**

**Problem Statement:**

The Indian space agency ISRO must deal with budgeting decisions to choose how to optimally divide the budget among its N competing missions. Each mission head has submitted the cost to be incurred and its overall value (or profit) for consideration. The budget available with the agency is 100 crores. They need your help in selecting as many projects as possible with the budget constraints such that the total value returned is maximized.

For this problem we can assume that there is no dependency of launching one project on the other. If there are multiple solutions with same value, choose the one which results in maximum utilization of budget and selection of missions as well.

**Requirements:**

1. Formulate an efficient recursive algorithm using Dynamic Programming to determine how to select the missions to be funded and maximize value.

2. Analyse the time complexity of your algorithm.

3. Implement the above problem statement using Python 3.7.

**Design:**

The class name is ISROMissionApprover, which is defined in main.py.

The ISROMissionApprover has the following attributes:

**Input**

allocated\_budget – allocated budget for the year.

missions- mission name array

budgets- budget corresponding to each mission as array.

values- value corresponding to each mission as array.

no\_of\_missions- Total number of missions proposed.

**output**

selected\_missions – Approved/Selected missions.

total\_profit – total profit for the approved/selected missions.

remaining\_budget – Remaining budget after approved/selected budget is allocated.

**Functions:**

|  |  |  |
| --- | --- | --- |
| **Function Name** | **Time complexity** | **Comments** |
| read\_input(self, file\_name) | O(1)/ Constant | This function is used to read the input from inputPS8.txt file and fill the input variables listed above. |
| write\_output(self,output\_file\_name) | O(1)/ Constant | This function is used to write output to file outputPS8.txt |
| prepare\_output(self) | **O(n)** | This function is used to prepared the necessary output variables in class for printing or giving it to write\_output. |
| find\_maximum\_profit(self,budgets, values, allocated\_budget, no\_of\_missions) | **O(n\*allocated\_budget)** | This function calculates the total value matrix. This function fills the matrix with the appropriate profit values. |

**Time Complexity Analysis:**

**find\_maximum\_profit complexity analysis:**

|  |  |  |
| --- | --- | --- |
| Mission-1 | Mission-2 | Mission-3 |
| 0 | 0 | 0 |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 0 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |
| 1 | 0 | 1 |

This function is usually takes O(2n) since we have n missions and those can be arranged with multiple combinations O(2n)

Example:  
if have 3 values like

Budget-{10,20,30}

Values-{20,40,50}

Suppose we have max budget capacity is 50

We can have combination matrix(0/1 Knapsack) as

1. represents not selected
2. represents selected

The possible combinations are 23 = 8, so we need to try and find the best solution which will satisfy our condition of max capacity.

So the complexity is O(2n) where n is number of elements in given input.

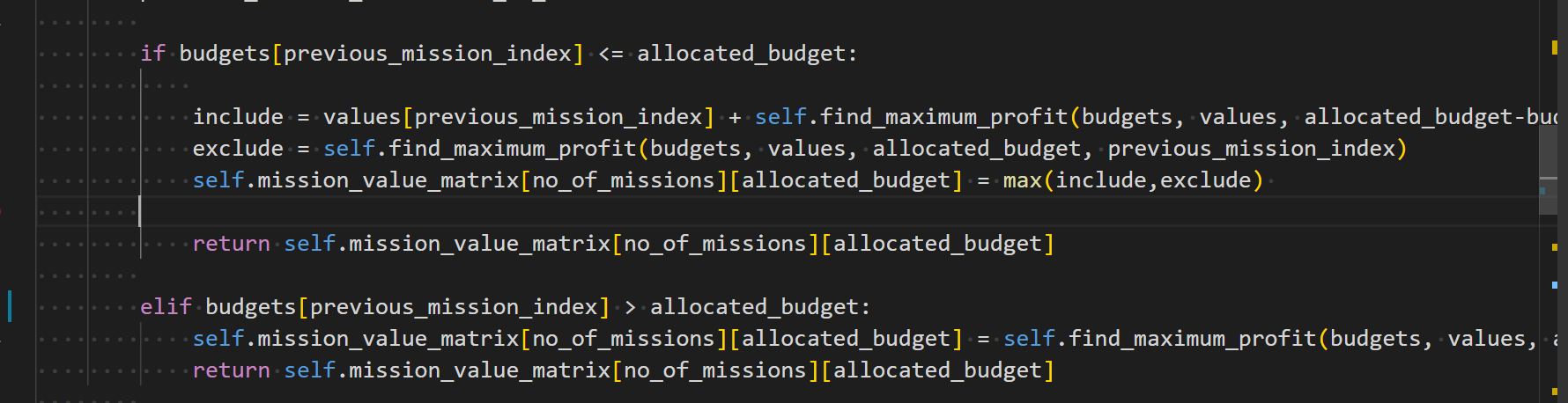
In this we are trying

1. Figure out optimal solution.
2. We are taking sequence of decisions.

So we can apply dynamic programming.

So in the current function we are following Top-down Dynamic Programming with Memoization.

Which is we are using recursive function



For this

f(n) = 0 for all N =0 or C= 0

f(n) = max(val[n-1]+f(n-1,C-val[n-1], f(n-1,C) ) where val is array of values, n is the element C is capacity if current\_budget < allocated\_budget

f(n) = f(n-1,C-val[n-1) if if current\_budget > allocated\_budget

so f(n) = 0 for all N= 0 or C = 0

f(n) = g(n-1) where g(k) = max(val[k]+f(k,C-val[k], f(k,C) ) where k is n-1

in a mathematical form it can be represented as

A similar dynamic programming solution for the 0-1 knapsack problem also runs in pseudo-polynomial time. Assume

w1,w2,w3....wn, W are strictly positive integers. Define m[i,w] to be the maximum value that can be attained with weight less than or equal to w using items up to

*i* (first *i* items).

m{0,*w*} = 0

m{*i*,*w*} = m[*i-1,W] if* wi >*w (the new item is more than the current weight limit)*

m{*I,w*} = max(m[*i-1,W],*m[*i-1,w-wi]+vi*) if wi ­­<= w

The solution can then be found by calculating m[n,W]. To do this efficiently, we can use a table to store previous computations.

So the complexity is in between O(N\*C) to O(2n)

**memorization**

We do not need to re-calculate the repeated sequence, since we know the value of the resultant sequence we are reducing the recursive O(2n)to O(N\*C)

if self.mission\_value\_matrix[no\_of\_missions][allocated\_budget] != -1:

    return self.mission\_value\_matrix[no\_of\_missions][allocated\_budget]

So if we continue like this we will fill the value matrix of capacity N\*C.

So the complexity of the algorithm is O(N\*C).

**prepare\_output** **complexity analysis:**

For printing results we need to use bottom approach

  while itemNo > 0:

            if (itemNo == 1 or current\_profit != self.mission\_value\_matrix[itemNo-1][self.remaining\_budget]) and self.budgets[itemNo-1] < self.remaining\_budget: #item was included

                self.selected\_missions = [itemNo] + self.selected\_missions

                self.remaining\_budget = self.remaining\_budget - self.budgets[itemNo-1] #subtract the capacity

                current\_profit = current\_profit - self.values[itemNo-1]

            itemNo = itemNo - 1

here the complexity function can be defined as

f(n) = return if n<= 0

f(n) = select n if (n=1 or current\_value != value\_matix(n-1,remaing\_budget) ) and budget[n-1] < remaining\_budget

so the loop gets over n to 0 so the complexity is O(n)

**Contribution Table:**

**Contribution** (This table should contain the list of all the students in the group. Clearly mention each student’s contribution towards the assignment. Mention “No Contribution” in cases applicable.)

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl. No.** | **Name (as appears in Canvas)** | **ID NO** | **Contribution** |
| **1** | **Gopi A V V** | 2020FC04797 | **33.33%** |
| **2** | **Kamaraj Kumar Sivva** | 2020FC04801 | **33.33%** |
| **3** | **Chakradhar Kausik P** | 2020FC04737 | **33.33%** |
|  |  |  |  |