StockYourBookshelf

Isaac Gutt

May 5, 2022

1 Key Insights

The main insight was that we want to be able to track every possible combination of sefarim without having to go back and check the prices of any kind of sefarim a second time. In short, we need to track every single possible solution, greedy will not work because it's possible she'll run out of money and then need to start from the beginning again. Therefore, we need to gather all the information for each sefer after the first time we run through that sefer's type, then we could add all the possibilities we stored and use them to purchase the next sefer at it's prices, and on like that until we find all the possibilites and choose the best one.

2 Optimal Substructure

Since this problem contains many sub problems, we can simply store all the subproblems after each run through a certain sefer type, then take those subproblems and create new ones with the next sefer type. Let's define $b = budget$,and $s = NumOfSefarimTypes$. The way I did this was with a 2d array, $size = s*b$. In this way, at each sefer type, we can store how much money is being spent with each respective subproblem, then with the next sefer (next row) we can build off of the previous problems which are above by using $c = CurrentSefer$ and $r = RemainingMoney$ to maximize the amount of money spent with those pre determined variables. For example, if b = 20 and the sefarim are gemara and chumash, we can buy a \$3 gemara and the first row will have a 1 (instead of 0) at spot 17 to symbolize that there are \$17 left, and a \$7 chumash will result in a 1 in spot 10 of the next row. By the time we get to the last sefer row, we'll have the most expensive possibility stored (lowest spot with a 1) and that will be the answer, or no possibilities and the result will have no solution.

3 Recursive Formulation

$$spendMax = \begin{cases} spendMax(c+1, (r - price[c][n])), & \text{if } r \geq 0 \text{ and } c < s. \\ b - r, & \text{if c = s.} \end{cases} \quad (1)$$

4 Performance

The 2d array used is size $s * b$. Since in the worst case all the boxes in the 2d array will be filled, the time complexity is O(sb).

The solution that returns the list is O(n), because I create a sorted set based on the lexicographical ordering of the sefarim strings as the solution happens with each add to the sorted set being an O(log n) operation, and a Hashmap with the sefarim and their prices in maxAmountThatCanBeSpent. So by the time solution is called, I simply run through the sorted set, get the price of the sefer from the hashmap and add it to the new list.