StockYourBookshelf

Benjamin Katz

May 2022

1 Key Insight

Thinking of this problem as sub problems that build off each other. The sub problem is what if she could only change a single T in a given C. In that case, it is simple to get every possible budget by swapping in every book. Once you are allowed to swap out from a second C, you can do so based on any of the previous budgets you created the last time you swapped. However, it is only relevant if it can create a new budget that was unachievable before. Being able to save the previous unique set of achievable budgets allows you to create more unique achievable budgets by only looking at one additional C at a time.

2 Optimal Substructure

If for C_0 (where you must take the cheapest T of every C aside from C_0) the optimal substructure for any given budget is to vary the T in C_0 to get as many of the possible budgets as possible within the given budget. Then, the optimal substructure of C_1 would either be the same as C_0 or varying a single T in C_1 from a given solution in C_0 to meet previously unattainable budgets. To do this, we would $\forall q$ find $A_q = C_1T_q – C_1T_0$ to find all possible additions that we can add to any solution given in the previous iteration to be able to meet budgets that were not reachable before.

3 Recurrence

Let i = the number of Classes we are considering.
Let j = a given budget we are trying to achieve with the given number of classes.
Let $K[i][j]$ = the closest possible reachable budget to j.
$K[i][j] = j$ if $K[i-1][j-A_q] = j - A_q$
$K[i][j] = max(K[i-1][j], K[i][j-1])$ if there is no way to get to j exactly

4 Runtime

The code traverses the entire 2d array which is (M x C). At every step it makes 2+T comparisons. Overall the runtime is $O(M * C * T)$ for maxAmountThatCouldBespent. The actual solution is calculated along the way which takes up additional space but allows for solution() to run in O(1) constant time.