CO project

Theory

Algorithm

I chose Greedy algorithm to find sensible solutions for the problem.

Why was chosen?

Greedy algorithms are usually easy to implement and gives nice solutions when the search space is big, just like in case of knapsack problem.

Implementation

Library class

```
class Library:
    def __init__(self, idx : int, num_of_books : int, days_to_signup : int,
                books_per_day : int):

        self.id = idx
        self.num_of_books = num_of_books
        self.days_to_sign_up = days_to_signup
        self.books_per_day = books_per_day
        self.book_ids = dict()
        self.book_order = list()
        self.score = 0
```

This class stores a few important things:

- book_ids: ids of books kept in a dictionary in which key is a book's id and value is a book's value. To keep it in decreasing order I used `dict.fromkeys()` method which preserves the order of added keys, which are in this case book_ids sorted in descending order, which all of it takes $O(n * \log n)$ time to sort and add elements to dict
- book_order: is a list which keeps the order in which books should be sent, it was used during randomizing order of books

Problem class

The main idea was to sort libraries according to some heuristic function.

The ProblemDummyGreedy uses function

$$H(\texttt{Lib}) = \texttt{Lib.days\_needed\_to\_assign}$$

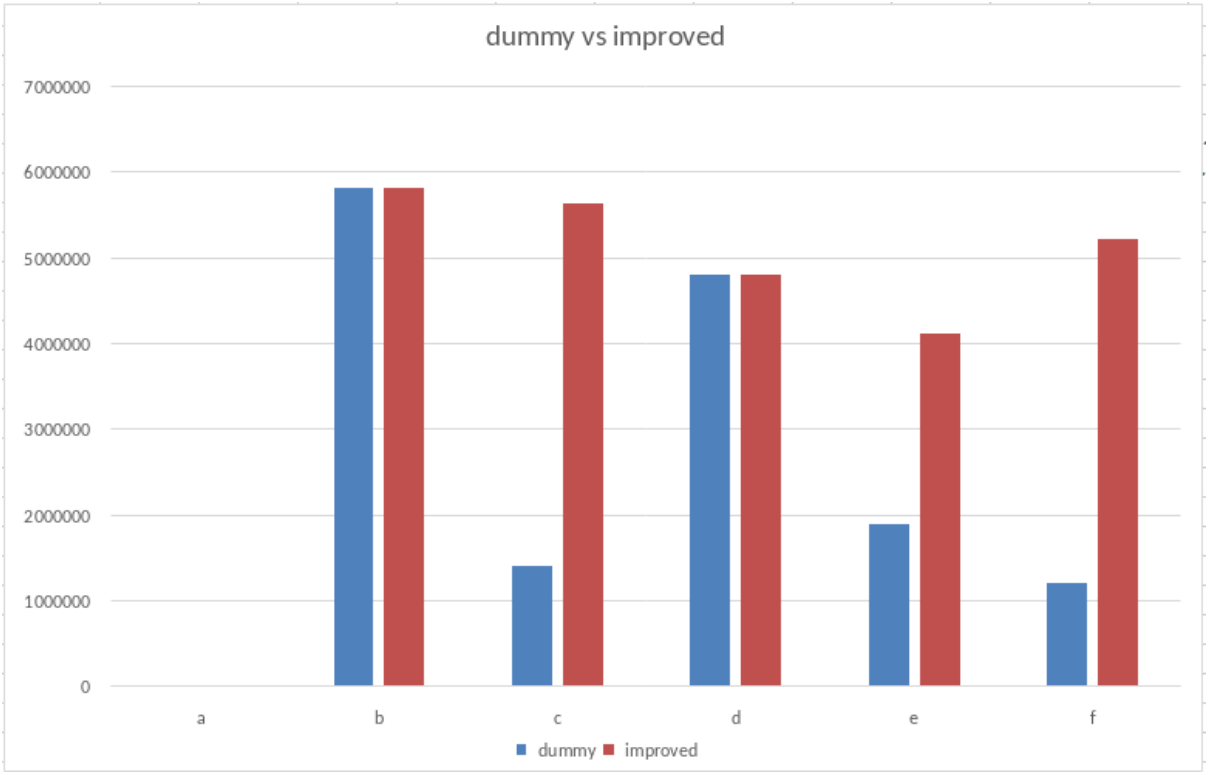
To sort libraries using this function as comperator it takes $O(n \log n)$ time.

The ProblemImprovedGreedy uses function

$$H(\texttt{Lib, start\_day, last\_day}) = \frac{\sum_{i=0}^{n} \texttt{Lib.book\_order[i]}}{\texttt{Lib.days\_needed\_to\_assign}}$$
$$\text{where } n = \texttt{Lib.days\_needed\_to\_assign*(last\_day - start\_day)}$$

The complexity of this method takes $O(m)$ for a one library where m is number of books, sorting takes $O(n \log n)$ where n is number of libraries, so together finding order of libraries takes $O(m * n \log n)$ time.

Comparison between dummy and improved greedy



Score accumulated on the all test cases: 25 646 732