DATA STRUCTURE INDIVIDUAL ASSIGNMENT

Kamran Gasimov

June 20, 2021

1 BACKGROUND

Me and my friends (from Azerbaijan) are working on opening a restaurant on the wheels in Baku, Azerbaijan (and this is not a joke, we have already registered our company, but it is still in the designing stage). However, as a just established company, we would like to fully analyze, some important models, like: What food/drink is being sold most? How much did we earn from every order? How many orders in total have been executed during the day? Just like all other restaurants, we will have cheque system, however, the data cannot be retrieved from there in the massive amounts, since it will be overwhelming. Furthermore, if this program was accessible to customers, they could be checking the ingredients/recipes of all the waffle that we offer. My program can easily solve this problem. In addition to that, in this project, I used a few extra non-essential features to mimic the real buying process as naturally as possible. They will be noted both in the overview and in the code.

2 THE GOAL OF THE CODE, AND CHALLENGES TO FACE

I put forward several essential goals that my code should be able to execute. The overall goal of the whole code is to allow the user to add all the items in menu that the customer wants to buy, and then give efficiently calculate out the "cheque" of what has been ordered. The user instructions have to be user-friendly, since they should be able to check the ingredients of any waffle/drink in the menu. Moreover, the given cheque should be price-sorted for a better view. So, we need efficient data structure for: insertions, sorting, and storing the data.

The main question was, which type of data structure should be used to solve this problem most effectively, and I firstly was going for the singly-linked lists. Adding an item to singly-linked list takes O(1) time complexity, while adding an item to arrays to last position takes O(1). However, searching an item in the known index takes O(1) time complexity for arrays, while it takes O(n) for singly-linked lists. It is easy to understand that if you want to search for a positions, no matter what we do next, our time complexity in singly linked lists will be O(n), which makes it unreasonable to use it when search is present. For that reason, I went for arrays as my preferred data structure for data search and insertion.

For the sorting data structure, I went for the in-place Quick-Sort algorithm, because its expected time complexity is just O(nlog(n)), and is one of the fastest for any kind of inputs.

For storing the data about the instructions, I decided to use the mapping data structure. Using it, I store the instructions in the dictionary, which are divided into relevant categories. Using keys, we will easily access necessary data.

3 SOLUTION

In order to solve this problem, I decided to divide my code into 3 main parts:

The first part consists only and only of the in-place quick sort algorithm. I implemented it the way that it has got two input arrays, but both of them will be sorted on the basis of only 1. So, if one array has got the items in the Menu, and the other array has their respective prices, my sorting algorithm will sort both arrays in the increasing price, and thus, item-price relationship will not change.

The second part is the most essential part of the code, which is a function dealing with the arrays of the menu (divided into food and drinks), and dictionary for the instructions. The function initializes the input inside the loop. The input can initialize special codes, which either accesses the dictionary, or actually breaks out from the loop. If no "special" input is given, then the function will treat it as an item in menu. If no such an item is present, the function will repeat the loop, and it will let you know that it does not exist. If the input actually exists as a food or drink in our restaurant, we will ad it to another array, which includes all the ordered items. Inside this function, we will sort the order, and then return the order together with the price of every item, and we will give full order price as well.

The third part is a simple loop which executes our function repeatedly. After every order, it will ask us if the work day is over, and we will choose if we should close or not. If we close, my code will return all the orders that have been given in the whole day.

4 PSEUDO-CODE

Algorithm 1:

```
def sorting(S1, S2, a, b):
        if a more than or equal to b do
                return
        endif

        pivot     S1[b]
        left      a
        right      b     1

        while left less than or equal to right do
                while left less than or equal to right and S1[right]
                 less than pivot do
                        add 1 to left
                endwhile
                while left less than or equal to right and pivot less
                 than S1[right] do
                        subtract 1 from right
                endwhile
                if left less than or equal to right do
                        S1[left], S1[right]     S1[right], S1[left]
                        S2[left], S2[right]     S2[right], S2[left]
                        left , right     left + 1, right     1
```

```
                endif

        S1[left], S1[b]     S1[b], S1[left]
        S2[left], S2[b]     S2[b], S2[left]
        sortsys(S1, S2, a, left - 1)
        sortsys(S1, S2, left + 1, b)
enddef
```

Algorithm 2:

```
def restaurant():
        number     generate random number from 1000 to 9999
        food     [some array of food in menu]
        drinks     [some array of drinks in menu]
        info  {dictionary of instructions}
        price_food     [some array of prices of food]
        price_drinks     [some array of prices of drinks]
        full     [empty array]
        qiymet  [empty array]
        total_order  [empty array]

        while True:
                order     request input
                if order = "exit" do
                        break
                endif
                if order = info do
                        choice_type     request input
                        if choice_type = "food" do
                                choice_food     request food
                                ind2  find index of choice_food in food
                                print info with key food and with index ind2
                        endif
                        if choice_type = "drink" do
                                choice_drinks     request drink
                                ind4     drink index of choice_drinks in drinks
                                print info with key drinks and index ind4
                        endif
                endif
                if order is not in food do
                        if order not in drinks do
                                print "Food not in menu"
                        endif
                        if order is in drinks do
                                ind3     index of order in drinks
                                add item from drinks with index
                                 ind4 to full
                                add item from price_drinks with index
                                 ind4 to qiymet
```

```
                endif
            if order is in food do
                ind  index of order in food
                add item from food with inde ind to full
                add item from price_food with index ind to qiymet
            endif
        endwhile
        fee = 0
        Algorithm 1 input (qiymet, full, 0, length of full minus 1)
        for i in range of length of full do
        add qiymet with index i to fee
        add an array of (full with index i, qiymet with index i) to total_order
        endfor
        full_order      number + total_order
        full_price      fee
        return full_order, full_price
enddef
```

Algorithm 3:

```
all_orders = [empty array]
count = 0
while True do
        a = request USERINPUT
        if a = 'yes' or a = 'y' or a = 'Yes' or a = "Yep" do
                b = call restaurant
                print b
                add b to all_orders
                add 1 to count
        endif
        else do
                print count
                break
        endelse
endwhile
print all_orders
```

5 Additional Info

Imports used: random

Python version: 3.8