Data Structures and Algorithms Design

Academic Year 2021-2022

Design Document - Assignment 2 – PS02 - [Medical Distributors]

**Authors**

Vinaya K

Muthuraja M

R Balaji

**Contribution Table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl. No.** | **Name (as appears in Canvas)** | **ID NO** | **Contribution** |
|  | Vinaya K | 2021C104200 | 100% |
|  | R Balaji | 2021C104109 | 100% |
|  | Muthuraja M | 2021C104112 | 100% |

**Problem Statement:**

A medical distributor has to keep track of all the retail pharmacies for smooth operations of supply.

They have a number of shops to supply stock to and need help retrieving details of these shops

quickly to handle the supply efficiently. Design a system that can quickly find the shop details based on the name of the pharmacy. The details that need to be included are:

* Pharmacy Name
* Phone Number
* Locality
* Delivery Status (Out for Delivery, Delivered, Yet to Deliver)

The medical distributor would like to use the system to provide the following functionality

* Add pharmacy names and other details into the system.
* Find a list of pharmacies with different delivery status (yet to deliver, out for delivery).
* Generate a list of pharmacies located in a given locality.
* Generate a report on the number of pharmacies to whom delivery has been done

**Overview of the program:**

The system/application is designed using Python 3.7 and using Hash table.

The pharmacy details that include, the name of the pharmacy, Phone number, area where the pharmacy is located, and the delivery status is reach from the input file “InputsPS02.txt”

Ex: Apollo / 9879314616 / Indiranagar / Delivered

Ramdev / 9879451461 / Hoodi / Yet to Deliver

Balaji Medicals / 989243582 / Hebbal / Out for Delivery

Angel / 9879431121 / Madambakam / Yet to Deliver

Angel Medicals / 9679431121 / Selaiyur / Delivered

Balaji / 9679467821 / Selaiyur / Yet to Deliver

The system is designed to have one of these values for the DeliveryStatus:

* Out for Delivery
* Delivered
* Yet to Deliver

The program is written to handle the following scenarios:

1. Read the contents of the input file “InputsPS02.txt” and write into the Hash table. After the Hash table is created, the status is updated in the OutputPS02.txt

**Ex**: “Successfully inserted XX records into the system.”

1. Reading the values in the PromtsPS02.txt and call different methods based on the values.
2. Read the Promptps02.txt and perform different operations based on the contents in the file
   1. If the keyword read from the Promptps02.txt file is “Delivery Status”, the program lists down all the pharmaceuticals based on the values in the InputPS02.txt and their corresponding delivery status that has the delivery status as “Yet to Deliver” and “Out for Delivery”:

**Ex:** Balaji Medicals : Out for Delivery

Ramdev : Yet to Deliver

Angel : Yet to Deliver

Balaji : Yet to Deliver

* 1. If the keyword read from the Promptps02.txt file is “Delivered”, the program lists down all the pharmaceuticals based on the values in the InputPS02.txt for which the delivery is completed.

**Ex:** Angel Medicals

Apollo

* 1. If the keyword read from the PromptsPS02.txt is name of the neither “Delivery Status” nor “Delivered” then, the programs lists all the pharmaceuticals belonging to that locality assuming the keyword read is the name of the locality.

Ex: Angel Medicals

Balaji

**Scope:**

In-scope:

The program is designed to read the InputsPS02.txt and PromptsPS02.txt and create a hash table for the contents in the InputsPS02.txt. Perform different operations based on the contents in the PromptsPS02.txt.

Out of Scope:

The program is designed to handle only the predefined delivery status (Out for Delivery, Yet To Deliver and Delivered). Any other delivery status will be ignored.

**Assumptions:**

1. The input file, Prompts file and output file will be located in the default python folder
2. If there are 2 entries with the same pharmacy name in the input file, the latest record/entry in the file is considered overwriting the previously read values.
3. The input file has to maintain the data in the same order separated by “/”. 1st Value would be name of pharmacy, 2nd value would be phone number, 3rd value would be locality and the 4th value is delivery status. This order has to be maintained across all the lines/data in the input file implying input is read based on index and not based on key/attribute name

**Flow Chart:**

Start

**No**

End

**Yes**

Is end of file

Print Output File

Read the next line

**For other Keywrods**

**Delivered**

**DeliveryStatus**

List all the Pharmacies belonging to the location mentioned in the file

List all the pharmacies with status as “Delivered”

List all the Pharmacies with status as “Yet to Deliver” or “Out For Delivery”

Process keyword from the Prompts file

Read Prompts file

Create a Hash table

Read Input file

**Time Complexity Analysis:**

**Reading the Input file:**

# Iterating the lines present in the input file  
for line in inputLines:  
    # Strips the newline character and identify the empty lines  
    if not line.strip():  
        outputFile.write('Empty Line found in the input file.' +'\n')  
    else:  
        #Split the line by '/' character  
        inputKeys = line.strip().split('/')  
        inputKeysLength = len(inputKeys)  
        if(inputKeysLength == 4):  
            # Inserting into the HashTable if the input is valid  
            customHashTable.put(inputKeys[0].strip(), [inputKeys[1].strip(), inputKeys[2].strip(), inputKeys[3].strip()])  
            # Incrementing this inputCount counter variable on each successful insertion into the Hashtable  
            inputCount += 1  
        else:  
            # Input is not valid if the structure is not proper  
            outputFile.write('Invalid Input.' +'\n')

This method reads the input file and iterates through each line in the input file and creates an entry or Key Vaue Pair in the hash table.

**Time Complexity is O(n)**

**Reading the Prompts file:**

for line in promptLines:

    # Strips the newline character

    if not line.strip():

        print('Empty Line')

    else:

        print(line.strip())

        if(line.strip().casefold() == 'Delivery Status'.casefold()):

                customHashTable.fetchNonDeliveredPharmacies()

                promptCount += 1

        elif(line.strip().casefold() == 'Delivered'.casefold()):

                customHashTable.fetchDeliveredPharmacies()

                promptCount += 1

        else: # Iterating the lines present in the prompt file  
for line in promptLines:  
    # Strips the newline character and identify the empty lines  
    if not line.strip():  
        outputFile.write('Empty Line found in the prompt file.' +'\n')  
    else:  
        # Invoking fetchNonDeliveredPharmacies function if the prompt code is 'Delivery Status'  
        if(line.strip().casefold() == 'Delivery Status'.casefold()):  
                customHashTable.fetchNonDeliveredPharmacies()  
        # Invoking fetchDeliveredPharmacies function if the prompt code is 'Delivered'  
        elif(line.strip().casefold() == 'Delivered'.casefold()):  
                customHashTable.fetchDeliveredPharmacies()  
        # Invoking fetchPharmaciesByLocality function if the prompt code is neither 'Delivery Status' nor 'Delivered'  
        else:  
                customHashTable.fetchPharmaciesByLocality(line.strip())

                customHashTable.fetchPharmaciesByLocality(line.strip())

This method calls different functions based on the keyword read from the prompts file. If the keyword read from the file is “Delivery Status”, then fetchNonDeliveredPharmacies function is called. If the keyword read from the file is “Delivered”, then fetchDeliveredPharmacies function is called. For all other keywords fetchPharmaciesByLocalityis called.

**Time Complexity is O(n)**

**fetchNonDeliveredPharmacies:**

# Function to write to output file the name and delivery status of pharmacies of those not received their deliveries yet     
    def fetchNonDeliveredPharmacies(self):  
        for entry in self.entriesArray:  
            if(entry is not None and (entry[0][1][2].casefold() == 'yet to deliver'.casefold() or entry[0][1][2].casefold() == 'out for delivery'.casefold())):  
                outputFile.write(entry[0][0] + ' : ' + entry[0][1][2] +'\n')

This function fetches all the pharmaceuticals with the delivery status as “Yet to Deliver” and “Out For Delivery” and writes the same in the output file.

**Time Complexity is O(n)**

**fetchDeliveredPharmacies:**

# Function to write to output file the name of pharmacies whose delivery status is Delivered     
    def fetchDeliveredPharmacies(self):  
        for entry in self.entriesArray:  
            if(entry is not None and entry[0][1][2].casefold() == 'Delivered'.casefold()):  
                outputFile.write(entry[0][0] +'\n')

This function fetches all the pharmaceuticals with the delivery status as “Delivered” and writes the same in the output file.

**Time Complexity is O(n)**

**fetchPharmaciesByLocality:**

# Function to write to output file the name of pharmacies from the given locality  
    def fetchPharmaciesByLocality(self, locality):  
        for entry in self.entriesArray:  
            if(entry is not None and entry[0][1][1].casefold() == locality.casefold()):  
                outputFile.write(entry[0][0] +'\n')

This function fetches all the pharmaceuticals belonging to the Locality that is read in the prompts file and the same is written into the output file. Since the hash table is created with “Pharmacy Name” and not based on the locality hence to list down all the pharmacies belonging to any location, we have to loop through all the records in the hash table.

**Time Complexity is O(n)**

**Create Hash table:**

# Function to insert Entries into the Hash Table in [key,value] format  
    def put(self, key, value):  
        # Getting the position of the key based on hash  
        position = self.getHash(key)  
        # Check if already an entry exists in the position  
        if self.entriesArray[position] is not None:  
            # Replacing the old value with the new value if the same key exists  
            if self.entriesArray[position][0][0] == key:  
                self.entriesArray[position] = []  
                self.entriesArray[position].append([key, value])  
            # Looking for the next empty slot  
            else:                  
                nextslot = self.rehash(position,len(self.entriesArray))  
                # Identifying the next empty slot  
                while self.entriesArray[nextslot] != None and self.entriesArray[nextslot] != key:  
                    nextslot = self.rehash(nextslot,len(self.entriesArray))  
                # Inserting the [key,value] in the identified slot  
                self.entriesArray[nextslot] = []  
                self.entriesArray[nextslot].append([key, value])  
  
        else:  
            # Inserting the [key,value] in the position(based on hash of key) index  
            self.entriesArray[position] = []  
            self.entriesArray[position].append([key, value])

This method is used to create the hash table based on the contents of the input file.

**The time complexity is O(1).**

**GetHash:**

# Function to get the index of the array to be used based on the key  
    def getHash(self, key):  
        size = len(self.entriesArray)  
        return hash(key) % size

This function has only one statement and as a constant number binds the entire function the time complexity is represented as O(1).

**Time Complexity is O(1)**

**ReHash:**

# Function to get the index of the array to be used based on the previous index in case of hash collissions  
    def rehash(self,oldhash,size):  
        return (oldhash+1)%size

This function has only one statement and as a constant number binds the entire function the time complexity is represented as O(1).

**Time Complexity is O(1)**

**Acceptance Criteria:**

The following test scenarios are verified

|  |  |  |
| --- | --- | --- |
| **Test Scenario** | **Description** | **Sample Output (desired)** |
| Reading Input File | Process the input file and create a hash table | Update the Output file with:  **Ex: Successfully inserted 6 records into the system** |
| Read Prompts file | If the keyword read from the prompts file is “Delivered” | List all the pharmacies with status as delivered in the output file:  **Ex:** Apollo  Angel |
| Read Prompts file | If the keyword read from the prompts file is “DeliveryStatus” | List all the pharmacies with status as “Out For Delivery” or “Yet To Deliver” in the output file:  **Ex:** Balaji Medicals : Out for Delivery  Ramdev : Yet to Deliver |
| Read Prompts file | If the keyword read from the prompts file is neither “Delivered” nor “DeliveryStatus” | List all the pharmacies belonging to the location mentioned in the prompt file:  Ex: Angela Medicals  Balaji |