Αλέξανδρος Αριστόβουλος

1063199

Εργασία αποθήκευσης και αναζήτησης δεδομένων με χρήση Hashing

Δημιουργία επίσκεψης πιστωτικής κάρτας

Για τη δημιουργία του τυχαίου αριθμού της κάρτας διαλέγουμε συνεχώς ακέραιους αριθμούς μεταξύ του 0 (μηδέν) και του 9 (εννέα) μέχρι να φτάσουμε το επιθυμητό μήκος. Έπειτα με τη χρήση της συνάρτησης np.array.choice() διαλέγουμε τις θέσεις που θα αντικαταστήσουμε με τα γράμματα. Επιπλέον διαλέγουμε τυχαία ένα ποσό που θα πληρώσει η κάρτα καθώς και τη μέρα επίσκεψης.

```
def createCard():
    #create a random number with the specified length
    cardDigits = [str(np.random.randint(0, 9)) for i in range(CARD_DIGIT_LENGTH)]

    #find the spots you will replace with letters
    spotsToReplace = np.random.choice(CARD_DIGIT_LENGTH - 1, len(CARD_LETTERS), replace=False)

    #replace these spots with letters
    for i in range(len(CARD_LETTERS)):
        cardDigits[spotsToReplace[i]] = CARD_LETTERS[i]

    #when you are finished convert the list to a string
    cardDigits = "".join(cardDigits)

    #pick a random value between 20 and 100 for the price they paid
    value = np.random.randint(LOWEST_PRICE, HIGHEST_PRICE)

    #pick a random day (1-7 for Monday-Sunday)
    day = np.random.randint(0, 6)

    return cardDigits, value, day
```

Εύρεση κοντινότερου πρώτου

Για να μάθουμε αν ένας αριθμός είναι πρώτος αρκεί να ελέγξουμε αν υπάρχει κάποιος θετικός ακέραιος από το 2 μέχρι και τη ρίζα του αριθμού που ελέγχουμε που να τον διαιρεί. Για αυτό το λόγο για να βρούμε τον κοντινότερο πρώτο ξεκινάμε από το νούμερο που μας δίνεται και ελέγχουμε διαδοχικά όλους τους επόμενους ακέραιους μέχρι να βρούμε κάποιον πρώτο με τον τρόπο που προαναφέρθηκε.

```python
def isPrime(number):
    #we only need to check until the square root of the number
    for i in range (2, math.ceil(math.sqrt(number))+1):
        if number % i == 0:
            return False
    return True


def findClosestPrime(number):
    while isPrime(number) == False:
        number += 1
    return number
```

Εύρεση hash της κάρτας

Καθώς ο αριθμός της κάρτας περιλαμβάνει και γράμματα δεν μπορούμε να κάνουμε διαίρεση και να βρούμε το υπόλοιπο. Για αυτό το λόγο πριν κάνω την διαίρεση αντικαθιστώ τα γράμματα με νούμερα (πχ το Α γίνεται 11, το Β γίνεται 12 κλπ.). Ο τρόπος με τον οποίο το κάνω με κώδικα αντιστρέφει επίσης και τη σειρά των ψηφίων αλλά δεν μας νοιάζει γιατί κάνουμε το ίδιο σε όλες τις κάρτες. Παραδείγματα μετατροπών των αριθμών καρτών για την εύρεση hash:

12345ABCD -> 1543154321, 1A2B3C4D -> 145342311

```python
def hash(key):
    #this where we store the result pf the hash
    result = 0

    #convert string to list
    keyList = []
    keyList[:0] = key

    for i in range(len(keyList)):

        #if its a number then we multiply with the appropriate power of 10
        if keyList[i].isnumeric():
            result += int(keyList[i])*10**i
        #if its a letter convert it to a number first
        else:
            if keyList[i] == "A":
                val = 11
            elif keyList[i] == "B":
                val = 12
            elif keyList[i] == "C":
                val = 13
            else:
                val = 14

            result += val*10**i

    return result
```

Δημιουργία κλάσης κάρτας

Για να βάζουμε οργανωμένα τα στοιχεία της κάρτας (αναγνωριστικό, ποσό πληρωμής, ημέρες επισκέψεων) στο hash table χρησιμοποιούμε κλάση Card

```
class Card:
    def __init__(self, key, value, day):
        self.key = key
        self.val = value
        self.days = [day]
```

Τοποθέτηση καρτών στο hash table

Για τη τοποθέτηση των καρτών στο hash table χρησιμοποιούμε τη συνάρτηση hash για να δημιουργήσουμε νούμερο από το αναγνωριστικό της. Αυτό το νούμερο το διαιρούμε με το μήκος του hash table για να πάρουμε την θέση στην οποία πρέπει να τοποθετήσουμε τη κάρτα. Αν η θέση είναι άδεια τότε απλά τοποθετούμε τη κάρτα. Αν δεν είναι τότε περνάμε διαδοχικά τις επόμενες θέσεις μέχρι να βρούμε κάποια άδεια θέση ή μέχρι να βρούμε την ίδια κάρτα ήδη τοποθετημένη. Στη δεύτερη περίπτωση δεν τοποθετούμε πάλι την ίδια κάρτα, αλλά, απλά ενημερώνουμε τα δεδομένα της στη θέση που την έχουμε ήδη (αυξάνουμε το συνολικό πληρωτέο ποσό και προσθέτουμε την μέρα επίσκεψης).

```
def putInHashTable(card, hashTable):
    #get the hash result
    result = hash(card.key) % len(hashTable)

    #check if that spot is free
    if type(hashTable[result]) != Card:
        #if it is free it also means that we haven't used that card
        hashTable[result] = card

    else:
        while True:
            #loop until you find an empty spot to put the card
            if type(hashTable[result]) != Card:
                hashTable[result] = card
                break
            #or loop until you have the same card in that spot
            #(in that case you need to update it)
            elif hashTable[result].key == card.key:
                hashTable[result].val += card.val
                hashTable[result].days.extend(card.days)
                break

            #this is to move spots
            result = (result+1) % len(hashTable)

    return hashTable
```

Αύξηση του μεγέθους του hash table

Για να μεγαλώσουμε το hash table δεν μπορούμε απλά να προσθέσουμε κενές θέσεις στο τέλος επειδή τα στοιχεία που έχουμε ήδη μέσα θα είναι όλα τοποθετημένα λάθος γιατί έχουν μπει με βάση το παλιό μέγεθος του hash table. Για αυτό το λόγο, δημιουργούμε καινούργιο hash table με μέγεθος ένα διπλάσιο πρώτο αριθμό σε σχέση με το προηγούμενο μέγεθος. Έπειτα για κάθε κάρτα του παλιού hash table ξανακάνουμε hash για να το βάλουμε στη σωστή καινούργια θέση.

```
def redefineHashTable(hashTable):
    #create a new hash table with double the size
    newHashTable = [None] * findClosestPrime(2*len(hashTable))

    #copy all the elements to the new hash table
    #(they will have a different position)
    for i in range (len(hashTable)):
        if type(hashTable[i]) == Card:
            newHashTable = putInHashTable(hashTable[i], newHashTable)

    return newHashTable
```

Έλεγχος του load factor

Για να ελέγξουμε αν έχουμε ξεπεράσει το load factor μετράμε τις κάρτες που περιέχει το hash table και διαιρούμε τον αριθμό τους με το μήκος του hash table. Αν το αποτέλεσμα είναι μεγαλύτερο από αυτό που έχουμε ορίσει τότε έχουμε ξεπεράσει το load factor.

```
def exceedLoadFactror(hashTable):
    #get the number of cards in the list
    cardCount = sum(isinstance(i, Card) for i in hashTable)

    #if the percentage of Cards exceeds the limit return True
    if cardCount / len(hashTable) > MAX_LOAD_FACTOR:
        return True
    return False
```

Δημιουργία του τελικού hash table

Αρχικά ξεκινάμε με ένα άδειο hash table μεγέθους ίσο με το κοντινότερο πρώτο που μας έχει δοθεί. Έπειτα, δημιουργούμε τόσες επισκέψεις καρτών όσες έχουμε ορίσει και προσέχουμε να διπλασιάζουμε το hash table με τον τρόπο που προαναφέρθηκε

όταν ξεπερνάμε το load factor που ορίσαμε. Για να μην χάνουμε χρόνο να μετράμε τις κάρτες στο hash table ενώ δεν γίνεται να έχουμε ξεπεράσει το load factor έχω βάλει ένα παραπάνω έλεγχο πριν εκτελέσω τη συνάρτηση ελέγχου load factor. Αυτό που κάνω είναι να κοιτάζω πρώτα αν έχω αρκετές επισκέψεις έτσι ώστε αν όλες ήταν με διαφορετική κάρτα να είχα ξεπεράσει το load factor και τότε μόνο «μετράω» ακριβώς πόσο γεμάτο είναι το hash table. Αυτό το επιτυγχάνω με τη χρήση της μεταβλητής που χρησιμοποιώ στην επανάληψη για την δημιουργία των επισκέψεων.

```
def createFinalHashTable():
    print("Creating hash table for {} card visits and a load factor of {}" .format(TOTAL_CARD_VISITS, MAX_LOAD_FACTOR))
    #create an empty the hash table
    hashTable = [None] * findClosestPrime(INITIAL_CAPACITY)
    print("The initial size of the hash table is: {}" .format(len(hashTable)))

    #fill the table with cards
    for i in range (TOTAL_CARD_VISITS):
        #get all the values for a card
        id, val, day = createCard()
        #create the card
        card = Card(id, val, day)
        #put the card in the table
        hashTable = putInHashTable(card, hashTable)

        #if the table is too filled then create a bigger one
        #and move everything to it
        if i >= MAX_LOAD_FACTOR*len(hashTable):
            if exceedLoadFactror(hashTable):
                hashTable = redefineHashTable(hashTable)
                print("The new size of the hash table is: {}" .format(len(hashTable)))

    return hashTable
```

Απαντήσεις στα ερωτήματα

Τα τεστ με έγιναν με τις επισκέψεις και τα μεγέθη που μας δόθηκαν. Για να απαντηθούν τα ερωτήματα 1-4 παίρνουμε το hash table και για κάθε στοιχείο του ελέγχουμε αν περιέχει κάρτα. Αν έχει τότε κάνουμε τους απαραίτητους ελέγχους για να ενημερώσουμε τις μεταβλητές στις οποίες αποθηκεύουμε τις απαντήσεις.

```
def getStats(hashTable):
    #set the min payment value to the max of the system so that we definately find a smaller number in a card
    minPaymentsCard = Card('', sys.maxsize, [])
    #in the card with the max visits we put an empty list to change it once we find another card
    maxVisitsCard = Card('', 0, None)
    #create a dictionary with the visits of each day
    dayVisits = 6 * [0]
    #total number of conflicts
    conflictsNum = 0
```

```
    for i in range(len(hashTable)):
        #first check if we have a card
        if type(hashTable[i]) == Card:

            #to get the card with the least amount payed
            if hashTable[i].val < minPaymentsCard.val:
                minPaymentsCard = hashTable[i]

            #to get the card with the most visits
            if len(hashTable[i].days) > len(maxVisitsCard.days) or maxVisitsCard.days == [None]:
                maxVisitsCard = hashTable[i]

            #update the visits for each day
            for j in range(len(hashTable[i].days)):
                dayVisits[hashTable[i].days[j]] += 1

            #if the hash of the card doesn't match its position in the list it means that we had a conflict
            if hash(hashTable[i].key)%len(hashTable) != i:
                conflictsNum += 1

    return minPaymentsCard, maxVisitsCard, dayVisits, conflictsNum
```

Αποτελέσματα για 1.000.000 επισκέψεις, 1.000 αρχικό μέγεθος hash table, 16 ψηφία κάρτας και 0.6 load factor

```
Creating hash table for 1000000 card visits and a load factor of 0.6
The initial size of the hash table is: 1009
The new size of the hash table is: 2027
The new size of the hash table is: 4057
The new size of the hash table is: 8117
The new size of the hash table is: 16249
The new size of the hash table is: 32503
The new size of the hash table is: 65011
The new size of the hash table is: 130027
The new size of the hash table is: 260081
The new size of the hash table is: 520193
The new size of the hash table is: 1040387
The new size of the hash table is: 2080777
Total time to create a hash table for 1000000 cards with a load factor of 0.6 is 274.548855304718 seconds
The card with the least amount payed is 36A8B53CD4273401 with 20 payed
The card with the most visits is B4A248C241D56845 with 1 visit(s)
The day with the least visits is 3 with 166109 visit(s)
The total number of conflicts is 239642
```

Αποτελέσματα για 2.000.000 επισκέψεις, 1.000 αρχικό μέγεθος hash table, 16 ψηφία κάρτας και 0.6 load factor

```
Creating hash table for 2000000 card visits and a load factor of 0.6
The initial size of the hash table is: 1009
The new size of the hash table is: 2027
The new size of the hash table is: 4057
The new size of the hash table is: 8117
The new size of the hash table is: 16249
The new size of the hash table is: 32503
The new size of the hash table is: 65011
The new size of the hash table is: 130027
The new size of the hash table is: 260081
The new size of the hash table is: 520193
The new size of the hash table is: 1040387
The new size of the hash table is: 2080777
The new size of the hash table is: 4161557
Total time to create a hash table for 2000000 cards with a load factor of 0.6 is 552.7620334625244 seconds
The card with the least amount payed is C4840325D5AB3025 with 20 payed
The card with the most visits is 814C58AD22B65276 with 1 visit(s)
The day with the most visits is 2 with 333756 visit(s)
The total number of conflicts is 480084
```

Αποτελέσματα για 1.000.000 επισκέψεις, 1.000 αρχικό μέγεθος hash table, 16 ψηφία κάρτας και 0.7 load factor

```
Creating hash table for 1000000 card visits and a load factor of 0.7
The initial size of the hash table is: 1009
The new size of the hash table is: 2027
The new size of the hash table is: 4057
The new size of the hash table is: 8117
The new size of the hash table is: 16249
The new size of the hash table is: 32503
The new size of the hash table is: 65011
The new size of the hash table is: 130027
The new size of the hash table is: 260081
The new size of the hash table is: 520193
The new size of the hash table is: 1040387
The new size of the hash table is: 2080777
The total number of conflicts is 240114
```

Αποτελέσματα για 1.000.000 επισκέψεις, 1.000 αρχικό μέγεθος hash table, 16 ψηφία κάρτας και 0.8 load factor

```
Creating hash table for 1000000 card visits and a load factor of 0.8
The initial size of the hash table is: 1009
The new size of the hash table is: 2027
The new size of the hash table is: 4057
The new size of the hash table is: 8117
The new size of the hash table is: 16249
The new size of the hash table is: 32503
The new size of the hash table is: 65011
The new size of the hash table is: 130027
The new size of the hash table is: 260081
The new size of the hash table is: 520193
The new size of the hash table is: 1040387
The new size of the hash table is: 2080777
The total number of conflicts is 239633
```

Αποτελέσματα για 2.000.000 επισκέψεις, 1.000 αρχικό μέγεθος hash table, 16 ψηφία κάρτας και 0.7 load factor

```
Creating hash table for 2000000 card visits and a load factor of 0.7
The initial size of the hash table is: 1009
The new size of the hash table is: 2027
The new size of the hash table is: 4057
The new size of the hash table is: 8117
The new size of the hash table is: 16249
The new size of the hash table is: 32503
The new size of the hash table is: 65011
The new size of the hash table is: 130027
The new size of the hash table is: 260081
The new size of the hash table is: 520193
The new size of the hash table is: 1040387
The new size of the hash table is: 2080777
The new size of the hash table is: 4161557
The total number of conflicts is 479827
```

Αποτελέσματα για 2.000.000 επισκέψεις, 1.000 αρχικό μέγεθος hash table, 16 ψηφία κάρτας και 0.8 load factor

```
Creating hash table for 2000000 card visits and a load factor of 0.8
The initial size of the hash table is: 1009
The new size of the hash table is: 2027
The new size of the hash table is: 4057
The new size of the hash table is: 8117
The new size of the hash table is: 16249
The new size of the hash table is: 32503
The new size of the hash table is: 65011
The new size of the hash table is: 130027
The new size of the hash table is: 260081
The new size of the hash table is: 520193
The new size of the hash table is: 1040387
The new size of the hash table is: 2080777
The new size of the hash table is: 4161557
The total number of conflicts is 479519
The thread 'MainThread' (0x1) has exited with code 0 (0x0).
The program 'python.exe' has exited with code 0 (0x0).
```

Παρατηρήσεις

- Το τελικό μέγεθος του hash table ήταν ίδιο για 0.6, 0.7 και 0.8 load factor και για αυτό το λόγο δεν βλέπουμε ουσιαστική διαφορά στον αριθμό των συγκρούσεων. Αν είχαμε διαφορετικό αριθμό επισκέψεων μπορεί με το 0.6 να αναγκαζόμασταν να αυξήσουμε το hash table ενώ με το 0.7 και το 0.8 να κρατάγαμε πιο μικρό μέγεθος. Σε εκείνη τη περίπτωση η διαφορά των συγκρούσεων θα ήταν πιο σημαντική.
- Με 16 ψηφία για αναγνωριστικό κάρτας η πιθανότητα να δημιουργήσουμε την ίδια κάρτα σε 1.000.000 επισκέψεις είναι αστρονομικά μικρή. Αν δεν μετρήσουμε καν τα γράμματα σαν ψηφία έχουμε 10^12 συνδυασμούς για το αναγνωριστικό κάρτας το οποίο είναι 10^6 φορές μεγαλύτερο νούμερο από τις 10^6 επισκέψεις. Αυτός είναι και ο λόγος που η κάρτα με τις περισσότερες επισκέψεις έχει 1 επίσκεψη.