Εργασία αποθήκευσης & αναζήτησης δεδομένων με χρήση HASHING

Σκοπός:

Ο στόχος της εργασίας είναι η υλοποίηση μιας αποτελεσματικής λύσης για τον υπολογισμό συνολικών χρεώσεων σε κατόχους πιστωτικών καρτών, χρησιμοποιώντας την τεχνική του HASHING. Ζητείται η δημιουργία μιας λίστας πιστωτικών καρτών, η προσομοίωση χρεώσεων σε αυτές τις κάρτες και η ανάλυση των δεδομένων για τον προσδιορισμό των καρτών με το μεγαλύτερο/μικρότερο ποσό πληρωμών και το μεγαλύτερο/μικρότερο πλήθος συναλλαγών. Επιπλέον, πρέπει να κατασκευαστεί ένας πίνακας κατακερματισμού με ανοικτή διευθυνσιοδότηση.

- **Εκτίμηση της πολυπλοκότητας του main (pythonDictionary/myHashTable) αρχείου:**

```
def main():
    seed(1691961)

    cardsNum = 20_000
    iterationsNumList = [1_000_000, 2_000_000, 5_000_000]
    cardsList = [None] * cardsNum
    for i in range(cardsNum):
        cardsList[i] = makeCardID()

    for i in iterationsNumList:
        print("- Εκτέλεση αλγορίθμου για {} χρεώσεις -\n".format(i))
        startTime = time();

        dictionary = runEconomy(cardsList, i)
        printAllCardsInfo(dictionary)

        endTime = time();
        print("Χρόνος εκτέλεσης: {}\n".format(endTime - startTime))

    print("-- ΤΕΛΟΣ ΠΡΟΓΡΑΜΜΑΤΟΣ --")

    return;
```

```
def runEconomy(cardsList, num):
    dictionary = {}                                          # O(1)
    for i in range(num):                                     # O(n) == O(num)
        cardID = choice(cardsList)                           # O(1)
        charge = randint(5, 500)                             # O(1)
        day = randint(0, 5)                                  # O(1)
        if cardID in dictionary:                             # O(1), πρόκειται για dict!
            dictionary[cardID].makeTransaction(charge, day)  # O(1)
        else:
            dictionary[cardID] = Card(cardID)                # O(1), πρόκειται για dict!
            dictionary[cardID].makeTransaction(charge, day)  # O(1), πρόκειται για dict!

    return dictionary;                                       # O(1)
```

Αναλυτικότερα:

- Δημιουργία ενός κενού λεξικού (dictionary)
- Επανάληψη της λούπας num φορές:
    - Επιλογή τυχαίας κάρτας από τη λίστα cardsList
    - Τυχαία επιλογή ενός ποσού φόρτωσης (charge) και μιας ημέρας (day)
    - Εάν η κάρτα υπάρχει ήδη στο λεξικό, πραγματοποιείται μια συναλλαγή
    - Διαφορετικά, δημιουργία μιας νέας κάρτας και πραγματοποίηση συναλλαγής
- Επιστροφή του λεξικού dictionary

Δηλαδή, η χρονική πολυπλοκότητα εξαρτάται κυρίως από τον αριθμό των συναλλαγών που πραγματοποιούνται (num). Επομένως, **συνολικά η πολυπλοκότητα της συνάρτησης είναι O(n)**.

```
def printAllCardsInfo(dictionary):
    maxCharge = -1                      # O(1)
    maxChargeCard = None                # O(1)
    minCharge = float("inf")            # O(1)
    minChargeCard = None                # O(1)
    maxTransactionsNumber = -1          # O(1)
    maxTransactionsNumberCard = None    # O(1)

    for card in dictionary.values():                            # O(n) == O(πλήθος values)
        if maxCharge < card.charge:                             # O(1)
            maxCharge = card.charge                             # O(1)
            maxChargeCard = card                                # O(1)
        if minCharge > card.charge:                             # O(1)
            minCharge = card.charge                             # O(1)
            minChargeCard = card                                # O(1)
        if maxTransactionsNumber < card.transactionsNumber:     # O(1)
            maxTransactionsNumber = card.transactionsNumber     # O(1)
            maxTransactionsNumberCard = card                    # O(1)

    totalDayTransactions = [0] * 6                              # O(1)
    for card in dictionary.values():                            # O(n) == O(πλήθος values)
        for day in range(6):                                    # O(1)
            totalDayTransactions[day] += card.week[day]         # O(1)

    minTransactions = min(totalDayTransactions)                             # O(n)
    minTotalDayTransactions = totalDayTransactions.index(minTransactions)   # O(n)

    print("- Μέγιστο κόστος συναλλαγών = {}:".format(maxCharge))               # O(1)
    maxChargeCard.printCardInfo()                                             # O(1)
    print("- Ελάχιστο κόστος συναλλαγών = {}:".format(minCharge))             # O(1)
    minChargeCard.printCardInfo()                                             # O(1)
    print("- Μέγιστος αριθμός συναλλαγών = {}:".format(maxTransactionsNumber)) # O(1)
    maxTransactionsNumberCard.printCardInfo()                                 # O(1)
    print("- Ημέρα ελάχιστων συναλλαγών: {} | Ελάχιστες συναλλαγές: {}"\
        .format(minTotalDayTransactions, minTransactions))                    # O(1)

    return;
```

Αναλυτικότερα:

- Δημιουργία των μεταβλητών maxCharge, maxChargeCard, minCharge, minChargeCard, maxTransactionsNumber, και maxTransactionsNumberCard
- Επανάληψη για κάθε κάρτα του λεξικό:
    - Ενημέρωση των μεταβλητών maxCharge, maxChargeCard, minCharge, minChargeCard, maxTransactionsNumber, και maxTransactionsNumberCard ανάλογα με τις τιμές των συναλλαγών και των αριθμών συναλλαγών της κάθε κάρτας
- Δημιουργία της λίστας totalDayTransactions που αθροίζει τις συναλλαγές ανά ημέρα για όλες τις κάρτες
- Εύρεση της ελάχιστης συνολικής συναλλαγής και την αντίστοιχης ημέρας

Δηλαδή, η χρονική πολυπλοκότητα εξαρτάται από τον αριθμό των καρτών στο λεξικό. Επομένως, **συνολικά η πολυπλοκότητα της συνάρτησης είναι γραμμική ως προς τον αριθμό των καρτών, δηλαδή O(n), όπου n είναι το μέγεθος του λεξικού**.

Τα παραπάνω ισχύουν και στην υλοποίηση του πίνακα κατακερματισμού (Hash Table) που ανέπτυξα!

✍ **Άρα, η πολυπλοκότητα και των 2 αλγορίθμων που υλοποίησα, είναι O(n).**

- **Επαλήθευση της εκτίμησης της πολυπλοκότητας:**

Το αποτέλεσμα της εκτέλεσης του κώδικα **με τη δομή δεδομένων της γλώσσας προγραμματισμού (dictionary)**:

- Εκτέλεση αλγορίθμου για 1000000 χρεώσεις -

- Μέγιστο κόστος συναλλαγών = 21596:
Κάρτα: ID = 7676-1115-3095-5279 | χρέωση = 21596 | κινήσεις = 75 | εβδομάδα = 19 12 14 9 9 12
- Ελάχιστο κόστος συναλλαγών = 5981:
Κάρτα: ID = 8449-6373-1956-7259 | χρέωση = 5981 | κινήσεις = 29 | εβδομάδα = 5 3 8 4 5 4
- Μέγιστος αριθμός συναλλαγών = 77:
Κάρτα: ID = 9307-5533-3449-3123 | χρέωση = 20509 | κινήσεις = 77 | εβδομάδα = 13 15 6 19 12 12
- Ημέρα ελάχιστων συναλλαγών: 0 | Ελάχιστες συναλλαγές: 166302
Χρόνος εκτέλεσης: 3.2179677486419678

- Εκτέλεση αλγορίθμου για 2000000 χρεώσεις -

- Μέγιστο κόστος συναλλαγών = 37331:
Κάρτα: ID = 9443-1072-5817-7854 | χρέωση = 37331 | κινήσεις = 149 | εβδομάδα = 22 15 29 32 28 23
- Ελάχιστο κόστος συναλλαγών = 13979:
Κάρτα: ID = 3845-9788-7493-2210 | χρέωση = 13979 | κινήσεις = 62 | εβδομάδα = 10 7 8 13 14 10
- Μέγιστος αριθμός συναλλαγών = 149:
Κάρτα: ID = 9443-1072-5817-7854 | χρέωση = 37331 | κινήσεις = 149 | εβδομάδα = 22 15 29 32 28 23
- Ημέρα ελάχιστων συναλλαγών: 4 | Ελάχιστες συναλλαγές: 332847
Χρόνος εκτέλεσης: 6.510864496231079

- Εκτέλεση αλγορίθμου για 5000000 χρεώσεις -

- Μέγιστο κόστος συναλλαγών = 83544:
Κάρτα: ID = 4748-5207-7257-3524 | χρέωση = 83544 | κινήσεις = 304 | εβδομάδα = 57 55 43 46 57 46
- Ελάχιστο κόστος συναλλαγών = 46516:
Κάρτα: ID = 3438-7617-8436-8197 | χρέωση = 46516 | κινήσεις = 189 | εβδομάδα = 30 30 28 35 38 28
- Μέγιστος αριθμός συναλλαγών = 319:
Κάρτα: ID = 1260-2301-1400-5164 | χρέωση = 82243 | κινήσεις = 319 | εβδομάδα = 58 55 60 53 48 45
- Ημέρα ελάχιστων συναλλαγών: 3 | Ελάχιστες συναλλαγές: 830519
Χρόνος εκτέλεσης: 15.758031129837036

-- ΤΕΛΟΣ ΠΡΟΓΡΑΜΜΑΤΟΣ --

Το αποτέλεσμα της εκτέλεσης του κώδικα **με την υλοποίηση του πίνακα κατακερματισμού που ανέπτυξα**:

- Εκτέλεση αλγορίθμου για 1000000 χρεώσεις -

- Μέγιστο κόστος συναλλαγών = 21596:
Κάρτα: ID = 7676-1115-3095-5279 | χρέωση = 21596 | κινήσεις = 75 | εβδομάδα = 19 12 14 9 9 12
- Ελάχιστο κόστος συναλλαγών = 5981:
Κάρτα: ID = 8449-6373-1956-7259 | χρέωση = 5981 | κινήσεις = 29 | εβδομάδα = 5 3 8 4 5 4
- Μέγιστος αριθμός συναλλαγών = 77:
Κάρτα: ID = 9307-5533-3449-3123 | χρέωση = 20509 | κινήσεις = 77 | εβδομάδα = 13 15 6 19 12 12
- Ημέρα ελάχιστων συναλλαγών: 0 | Ελάχιστες συναλλαγές: 166302
Χρόνος εκτέλεσης: 11.781339406967163

- Εκτέλεση αλγορίθμου για 2000000 χρεώσεις -

- Μέγιστο κόστος συναλλαγών = 37331:
Κάρτα: ID = 9443-1072-5817-7854 | χρέωση = 37331 | κινήσεις = 149 | εβδομάδα = 22 15 29 32 28 23
- Ελάχιστο κόστος συναλλαγών = 13979:
Κάρτα: ID = 3845-9788-7493-2210 | χρέωση = 13979 | κινήσεις = 62 | εβδομάδα = 10 7 8 13 14 10
- Μέγιστος αριθμός συναλλαγών = 149:
Κάρτα: ID = 9443-1072-5817-7854 | χρέωση = 37331 | κινήσεις = 149 | εβδομάδα = 22 15 29 32 28 23
- Ημέρα ελάχιστων συναλλαγών: 4 | Ελάχιστες συναλλαγές: 332847
Χρόνος εκτέλεσης: 22.80915594100952

- Εκτέλεση αλγορίθμου για 5000000 χρεώσεις -

- Μέγιστο κόστος συναλλαγών = 83544:
Κάρτα: ID = 4748-5207-7257-3524 | χρέωση = 83544 | κινήσεις = 304 | εβδομάδα = 57 55 43 46 57 46
- Ελάχιστο κόστος συναλλαγών = 46516:
Κάρτα: ID = 3438-7617-8436-8197 | χρέωση = 46516 | κινήσεις = 189 | εβδομάδα = 30 30 28 35 38 28
- Μέγιστος αριθμός συναλλαγών = 319:
Κάρτα: ID = 1260-2301-1400-5164 | χρέωση = 82243 | κινήσεις = 319 | εβδομάδα = 58 55 60 53 48 45
- Ημέρα ελάχιστων συναλλαγών: 3 | Ελάχιστες συναλλαγές: 830519
Χρόνος εκτέλεσης: 55.7244508266449

-- ΤΕΛΟΣ ΠΡΟΓΡΑΜΜΑΤΟΣ --

✍ **Παρατηρούμε ότι υπάρχει συμφωνία εκτιμήσεων. Αναλυτικότερα, οι χρόνοι εκτέλεσης των αλγορίθμων επιβεβαιώνουν τις αρχικές εκτιμήσεις πολυπλοκότητας. Παράλληλα, παρατηρήθηκε συνέπεια μεταξύ των προβλεπόμενων και των πραγματικών χρόνων εκτέλεσης για διάφορες περιπτώσεις εισόδου. Συμπερασματικά, οι αλγόριθμοι λειτουργούν με την αναμενόμενη απόδοση και η πολυπλοκότητα είναι σύμφωνη με τη θεωρητική ανάλυση.**