Προσδιορισμός της τρέχουσας διάμεσης τιμής (running median problem)

Αναφορά

Ανάλυση κώδικα

Ο κώδικας υπολογίζει τη διάμεση τιμή της θερμοκρασίας από ένα σύνολο N μετρήσεων, που είναι αντιστοιχισμένες με συγκεκριμένα σημεία στον χώρο. Ο κώδικας είναι γραμμένος σε Python και χρησιμοποιεί 2 σωρούς (heaps) για να διατηρεί τη σειρά των μετρήσεων και να βρίσκει τη διάμεση τιμή σε κάθε βήμα.

Ο κώδικας χρησιμοποιεί 2 σωρούς, έναν **min heap (largeHeap)** και έναν **max heap (smallHeap)**, για να διατηρεί τις μετρήσεις σε 2 ομάδες, ανάλογα με το αν είναι μεγαλύτερες ή μικρότερες από τη διάμεση τιμή. Ο **min heap** περιέχει τις μεγαλύτερες μετρήσεις, ενώ ο **max heap** περιέχει τις μικρότερες μετρήσεις. Η διάμεση τιμή είναι η ελάχιστη τιμή του **min heap** ή ο μέσος όρος των ελαχίστων τιμών των 2 σωρών, ανάλογα με το αν οι σωροί έχουν ίσο ή διαφορετικό μέγεθος.

✓ Ο κώδικας ακολουθεί τα εξής βήματα:

1. Αρχικοποιεί τους 2 σωρούς με κενούς πίνακες και ορίζει μια λίστα για να αποθηκεύει τα αποτελέσματα της διάμεσης τιμής.

```
def medianFromMeas(NtM, N):
    #NtM: N temperature Measurements & N: Πλήθος μετρήσεων
    largeHeap = MH([]) #min Heap
    smallHeap = MH([]) #Max Heap

    resultList = [None] * 2
    resultIndex = 0
```

2. Εισάγει τις πρώτες 10 μετρήσεις στον **max heap**, χρησιμοποιώντας ως κλειδί τον συνδυασμό των συντεταγμένων του σημείου και ως τιμή την αρνητική τιμή της θερμοκρασίας. Αυτό γίνεται για να επιτευχθεί η ιδιότητα του **max heap**, αφού ο **min heap** επιστρέφει το στοιχείο με τη μικρότερη τιμή. Επίσης, ελέγχει αν η μέτρηση υπάρχει ήδη στον **min heap** και αν ναι, τη διαγράφει από εκεί.

```
    bias = 10

    for i in range(bias):
        key = str(NtM[i][1][0]) + str(NtM[i][1][1])
        value = NtM[i][0]

        if smallHeap.isInMinHeap(key):
            smallHeap.changeKey((key, -value))
        else:
            smallHeap.insert((key, -value)) #push(smallHeap, -temp)
            if largeHeap.isInMinHeap(key):
                largeHeap.deleteKey((key, value)) #Άρα, το σημείο μπορεί να υπάρχει μόνο
σε έναν από τους 2 heap!
```

3. Εξάγει το μέγιστο στοιχείο από τον **max heap** και το εισάγει στον **min heap**, χρησιμοποιώντας την αντίθετη τιμή της θερμοκρασίας. Αυτό γίνεται για να ισορροπήσει τα μεγέθη των σωρών και να εξασφαλίσει ότι η διάμεση τιμή είναι η ελάχιστη τιμή του **min heap**.

```
        (smallHeap_maxKey, smallHeap_maxValue) = smallHeap.extractMin()
        largeHeap.insert((smallHeap_maxKey, -smallHeap_maxValue)) #push(largeHeap, -
smallHeap_max)

        if largeHeap.size > smallHeap.size:
            (largeHeap_minKey, largeHeap_minValue) = largeHeap.extractMin()
            smallHeap.insert((largeHeap_minKey, -largeHeap_minValue)) #push(smallHeap, -
largeHeap_min)
```

4. Επαναλαμβάνει τα βήματα 2 και 3 για τις υπόλοιπες μετρήσεις, με τη διαφορά ότι ελέγχει κάθε φορά αν η μέτρηση είναι μεγαλύτερη ή μικρότερη από τη διάμεση τιμή και ανάλογα την εισάγει στον κατάλληλο σωρό, διατηρώντας την ισορροπία των μεγεθών τους **(ο αλγόριθμος αλλάχτηκε για να επιτευχθεί μια πιο αποδοτική υλοποίηση)**.

```
        ########################################################################## value > median
        if value > median and largeHeap.size > smallHeap.size:
            (largeHeap_minKey, largeHeap_minValue) = largeHeap.extractMin()
            smallHeap.insert((largeHeap_minKey, -largeHeap_minValue))
            if largeHeap.isInMinHeap(key):
                largeHeap.changeKey((key, value))
            else:
                largeHeap.insert((key, value))
                if smallHeap.isInMinHeap(key):
                    smallHeap.deleteKey((key, -value))
        elif value >= median and largeHeap.size <= smallHeap.size:
            if largeHeap.isInMinHeap(key):
                largeHeap.changeKey((key, value))
            else:
                largeHeap.insert((key, value))
                if smallHeap.isInMinHeap(key):
                    smallHeap.deleteKey((key, -value))
        ########################################################################## value < median
        elif value <= median and largeHeap.size >= smallHeap.size:
            if smallHeap.isInMinHeap(key):
                smallHeap.changeKey((key, -value))
            else:
                smallHeap.insert((key, -value))
                if largeHeap.isInMinHeap(key):
                    largeHeap.deleteKey((key, value))
        elif value < median and largeHeap.size < smallHeap.size:
            (smallHeap_maxKey, smallHeap_maxValue) = smallHeap.extractMin()
            largeHeap.insert((smallHeap_maxKey, -smallHeap_maxValue))
            if smallHeap.isInMinHeap(key):
                smallHeap.changeKey((key, -value))
            else:
                smallHeap.insert((key, -value))
                if largeHeap.isInMinHeap(key):
                    largeHeap.deleteKey((key, value))
        ################################################## largeHeap.size == smallHeap.size
        else:
            if largeHeap.isInMinHeap(key):
                largeHeap.changeKey((key, value))
            else:
                largeHeap.insert((key, value))
                if smallHeap.isInMinHeap(key):
                    smallHeap.deleteKey((key, -value))
```

5. Υπολογίζει τη διάμεση τιμή σε κάθε βήμα, ανάλογα με το αν οι σωροί έχουν ίσο ή διαφορετικό μέγεθος. Αν ο **min heap** έχει ένα στοιχείο παραπάνω από τον **max heap**, τότε η διάμεση τιμή είναι η ελάχιστη τιμή του **min heap**. Αν οι σωροί έχουν το ίδιο μέγεθος, τότε η διάμεση τιμή είναι ο μέσος όρος των ελαχίστων τιμών των 2 σωρών.

```
    for i in range(bias, N - bias):
        key = str(NtM[i][1][0]) + str(NtM[i][1][1])
        value = NtM[i][0]

        if largeHeap.size != smallHeap.size:
            median = -smallHeap.getMin()[1]
        else:
            median = round(((largeHeap.getMin()[1] - smallHeap.getMin()[1]) / 2), 2)
```

6. Αποθηκεύει τη διάμεση τιμή στη λίστα των αποτελεσμάτων, αν ο αριθμός των μετρήσεων είναι ο μισός του συνόλου ή ο τελευταίος πριν από το τέλος του συνόλου. Αυτό γίνεται για να επιστρέψει τη διάμεση τιμή μετά τη δημιουργία των μισών και όλων των μετρήσεων, όπως ζητείται από την εκφώνηση.

```
if (i == ((N - 1) // 2)) or (i == (N - 1 - bias)):
            resultList[resultIndex] = median
            resultIndex += 1
```

7. Επιστρέφει τη λίστα των αποτελεσμάτων και τα τελικά μεγέθη των σωρών, ως μια πλειάδα (tuple).

```
    return (resultList, (largeHeap.size, smallHeap.size));
```

Ο κώδικας χρησιμοποιεί επίσης μια συνάρτηση main που δημιουργεί τυχαίες μετρήσεις και σημεία, καλεί τη συνάρτηση medianFromMeas με διαφορετικές τιμές του N και εμφανίζει τα αποτελέσματα και τον χρόνο εκτέλεσης.

```
def main():
    seed(5168131)
    N_pair = (500_000, 1_000_000) #Πλήθος μετρήσεων

    for N in N_pair:
        rMP = [(randomInt(0, 999), randomInt(0, 999)) for _ in range(100_000)] #100000 random
Measured Points
        NtM = [(round(randomFloat(-10, 90), 2), choice(rMP)) for _ in range(N)] #NtM: N
temperature Measurements

        start = time()
        (mediansResults, heapsSize) = medianFromMeas(NtM, N)
        stop = time()
        executionTime = stop - start

        print("Χρόνος εκτέλεσης για N = {} μετρήσεις: {} sec".format(N, executionTime))
        print("Διάμεση τιμή της θερμοκρασίας, μετά την δημιουργία των μισών[N/2] μετρήσεων:
{}".format(mediansResults[0]))
        print("Διάμεση τιμή της θερμοκρασίας, μετά την δημιουργία όλων των μετρήσεων:
{}".format(mediansResults[1]))
        print("Τελικό μέγεθος της δομής δεδομένων Large Heap[min]: {}".format(heapsSize[0]))
        print("Τελικό μέγεθος της δομής δεδομένων small Heap[Max]: {}\n".format(heapsSize[1]))

    return;
```

Ο αλγόριθμος που υλοποιεί ο κώδικας έχει πολυπλοκότητα χρόνου $O(N \log N)$, αφού κάθε εισαγωγή και εξαγωγή σε έναν σωρό έχει πολυπλοκότητα $O(\log N)$ και γίνονται $O(N)$ τέτοιες λειτουργίες.

Απάντηση ερωτήσεων:

```
Χρόνος εκτέλεσης για N = 500000 μετρήσεις: 11.981420278549194 sec
Διάμεση τιμή της θερμοκρασίας, μετά την δημιουργία των μισών[N/2] μετρήσεων: 39.88
Διάμεση τιμή της θερμοκρασίας, μετά την δημιουργία όλων των μετρήσεων: 40.0
Τελικό μέγεθος της δομής δεδομένων Large Heap[min]: 46814
Τελικό μέγεθος της δομής δεδομένων small Heap[Max]: 46814

Χρόνος εκτέλεσης για N = 1000000 μετρήσεις: 25.057960987091064 sec
Διάμεση τιμή της θερμοκρασίας, μετά την δημιουργία των μισών[N/2] μετρήσεων: 39.93
Διάμεση τιμή της θερμοκρασίας, μετά την δημιουργία όλων των μετρήσεων: 39.9
Τελικό μέγεθος της δομής δεδομένων Large Heap[min]: 47096
Τελικό μέγεθος της δομής δεδομένων small Heap[Max]: 47096
```