ΜΥΕ041 - ΠΛΕ081: Διαχείριση Σύνθετων Δεδομένων

(ΕΑΡΙΝΟ ΕΞΑΜΗΝΟ 2022-23)

ΕΡΓΑΣΙΑ 3- Top-k queries

Ονοματεπώνυμο: Evangelos Iliadis 3117 (Ευάγγελος Ηλιάδης 3117)

Στόχος της εργασίας είναι η ανάπτυξη αλγορίθμου για ερωτήσεις κορυφαίων κ (top-k queries), οι οποίες συναθροίζουν ατομικά σκορ αντικειμένων από τρεις πηγές. Τα αντικείμενα μπορεί να είναι εστιατόρια, η κάθε πηγή ένας ιστότοπος κρίσεων εστιατορίων (π.χ. google, TripAdvisor, yelp), και το ατομικό σκορ ενός αντικειμένου σε μία πηγή ο μέσος όρος των βαθμολογιών (ratings) που πήρε το εστιατόριο στο συγκεκριμένο ιστότοπο.

Πρώτο βήμα είναι η ανάγνωση του ακέραιου αριθμού k από το τερματικό.

```
#Main
if len(sys.argv) == 2: #Check if the user included all the necessary arguments
    try: #Firstly, check if the second argument is an integer
        k = int(sys.argv[1]) #Assign to k-variable the integer value of the second argument
        if (k > 0): #Secondly, check if it is positive
            print()
        else: #Negative integer
            print("Your integer is not positive! Please use a positive integer as your second argument!")
            exit()
    except: #Non-integer second argument
        print("You didn't use an integer as the second argument! Please use a positive integer as your second argument!")
        exit()
else: #Didn't include right amount of arguments
    print("Please use exactly 2 arguments! Your 2nd argument must be a positive integer!")
    exit()
add_from_rnd()
round_robin()
print_results()
```

Όπως βλέπουμε το πρόγραμμα περιμένει να διαβάσει τον αριθμό k από το τερματικό. Πιο συγκεκριμένα ο αριθμός k θα είναι το 2ο κατά σειράν όρισμα. Π.χ για τον αριθμό k = 5 η εντολή εκτέλεσης του προγράμματος θα έχει τη μορφή “python3 Assignment3.py 5”. Το πρόγραμμα αρχικά ελέγχει αν έχουν δοθεί ακριβώς δύο ορίσματα κατά την εκτέλεση. Αν όχι πετάει μήνυμα λάθους και τερματίζει. Εφόσον έχουμε 2 ορίσματα, το πρόγραμμα θα επιχειρήσει να κάνει casting το δεύτερο όρισμα ως int και να το αναθέσει στο k. Εδώ είναι απαραίτητο το try προκειμένου να καλύψουμε την περίπτωση στην οποία ο χρήστης δεν έβαλε έναν ακέραιο αριθμό ως δεύτερο όρισμα εκτέλεσης και κατ’ επέκταση δεν μπορεί να γίνει cast ως int. Σε αυτή τη περίπτωση πάλι εμφανίζεται μήνυμα λάθους και το πρόγραμμα τερματίζει. Τέλος, εφόσον έχουμε τον ακέραιό μας, γίνεται έλεγχος να είναι μεγαλύτερος του 0. Μιας και το νόημα της άσκησης είναι η εκτύπωση πλήθους στοιχείων, ένα αρνητικό πλήθος δεν είναι αποδεκτό, μιας και αρνητικό πλήθος δεν υφίσταται. Και σε αυτή την περίπτωση θα εμφανιστεί το αντίστοιχο μήνυμα λάθους σε περίπτωση αποτυχίας εκπλήρωσης της συνθήκης. Όταν περαστούν όλοι οι έλεγχοι με επιτυχία μπορούμε πλέον να μεταβούμε στο υπόλοιπο πρόγραμμα.

Λίγα λόγια για τις δομές δεδομένων που θα χρησιμοποιηθούν στο πρόγραμμά μας:

```
R = [] #Here will be stored the contents of rnd.txt
seen = {} #Dictionary. Holds all the objects that we've seen so far
sequential_accesses = 0 #Number of sequential accesses
heap_IDs = set() #The k ID's of the objects that are currently in the heap
```

- R: Πρόκειται για την λίστα/ πίνακα που θα αποθηκευτούν τα στοιχεία του rnd.txt
- Seen: Είναι ένα λεξικό το οποίο θα αποθηκεύει κάθε στοιχείο που διαβάζουμε μέσω του round-robin. Τα στοιχεία του θα έχουν την μορφή seen[ID] = (ID, άθροισμα, string από πού το έχουμε διαβάσει). Το τρίτο όρισμα παίρνει τις τιμές: “line1”, “line2” και “both” εάν το έχουμε διαβάσει από το seq1, το seq2 ή και από τα δύο αντίστοιχα.
- Sequential_accesses: Ο αριθμός σειριακών προσπελάσεων που χρειαστήκαμε κατά την αναζήτηση των top k αντικειμένων
- Wk: Μία στοίβα που κρατάει τα top k αντικείμενα που έχουμε δει
- Heap_IDs: ένα σετ που κρατάει τα ID των παρόντων αντικειμένων στη στοίβα

Αρχικά θα γίνει φόρτωση των στοιχείων του rnd.txt σε μία λίστα/ πίνακα. Εφόσον το rnd.txt είναι ταξινομημένο με βάση το ID, η θέση x του πίνακα θα περιέχει το ατομικό σκορ του αντικειμένου με ID=x.

```
#Add contents of rnd to R array.
def add_from_rnd():
    with open(rndPath, mode='r') as rnd: #Open rnd.txt for reading
        for line in rnd: #For each line
            splitLine = line.split() #Split line to get it's contents. id,rating
            R.append(float(splitLine[1])) #Add to array
```

Ανοίγουμε το rnd.txt για ανάγνωση. Διαβάζουμε γραμμή γραμμή και τη χωρίζουμε σε [ID,τιμή]. Προσθέτουμε την cast τιμή στο αντίστοιχο κελί του πίνακα.

Παρακάτω ακολουθεί η συνάρτηση round_robin που αποτελεί τον πυρήνα του προγράμματος. Θα εξηγηθεί ο κώδικας ανάγνωσης γραμμής από το seq1 και κατ' αντιστοιχία είναι η ίδια διαδικασία για τον κώδικα του seq2.

Αρχικά ανοίγουμε και τα δύο αρχεία seq1,seq2 για ανάγνωση. Αρχικοποιούμε τις μεταβλητές last1 και last2 στο 0. Αυτές θα μας φανούν χρήσιμες κατά τον έλεγχο για τερματισμό του προγράμματος. Επίσης ένα flag που είναι 1 εάν έχει δημιουργηθεί η στοίβα. Είναι ζητούμενο της άσκησης η στοίβα να δημιουργηθεί μόλις έχουν διαβαστεί τα πρώτα k αντικείμενα. Γι' αυτό και χρησιμοποιούμε την flag μιας και δεν υπάρχει η δομή στοίβας από την αρχή εκτέλεσης του προγράμματος.

Σε ένα endless loop(θα σταματήσει όταν το αποφασίσουμε εμείς) διαβάζουμε τη γραμμή από το seq1 και την σπάμε στα επιμέρους κομμάτια της. Εάν δεν είναι null και έχουμε όντως διαβάσει "κάτι", αυξάνουμε τις σειριακές προσπελάσεις κατά 1 και βάζουμε στο last1 την τιμή της γραμμής που μόλις διαβάσαμε.

Εάν το id της γραμμής υπάρχει στο λεξικό seen αυτό σημαίνει ότι το εν λόγω αντικείμενο το έχουμε δει και από την άλλη πηγή. Αυτό θα έχει λοιπόν κάποιο κάτω όριο από την πηγή seq2 + R πίνακα. Προσθέτουμε στο άθροισμα κάτω ορίου και την τιμή του seq1, αλλάζουμε το αναγνωριστικό σε "both" μιας και αυτό το αντικείμενο το έχουμε δει και από τις δύο πηγές και το ξανά αποθηκεύουμε ανανεωμένο στην ίδια θέση.

```
def round_robin():
    with open(seq1Path, mode = 'r') as seq1, open(seq2Path, mode = 'r') as seq2: #Open seq1,seq2 for reading
        global sequential_accesses
        last1 = last2 = 0.0 #Here will be stored the last value of the lines we've just read.
        flag = 0 #Binary flag. Will be set to 1 once the heap has been created.
        while True: #Loop until break
            exit = 1 #Binary value. Once we do the condition check, if exit == 1 break.
            # Read next line from seq1. And split it.
            line1 = seq1.readline().split() #line1 = [ID,value]
            if line1: #If next line is not null
                sequential_accesses += 1 #Increase sequential accesses by 1
                last1 = float(line1[1]) #Cast the value field to float
                if line1[0] in seen: #If we've already read this ID from the other source file
                    #Add the previous lower bound with the new line source value
                    #Round it to 2 decimals
                    summary = round(sum([float(seen[line1[0]][1]), float(line1[1])]), 2)
                    #Store in dictionary. seen[id] = (id,value,string)
                    #String can be "line1","line2" or "both" depending on where we've seen..
                    #..it so far. Since this is the second time we encounter the same id we put "both"
                    seen[line1[0]] = ((line1[0]), summary, "both")
```

Αφού έχουμε βάλει το στοιχείο στο λεξικό κοιτάμε εάν έχει δημιουργηθεί η στοίβα. Αν όχι δεν μας απασχολεί άλλο το συγκεκριμένο στοιχείο. Αν ναι, χρειάζεται να ελέγξουμε κάποιες επιπλέον περιπτώσεις καθώς ενδέχεται να πρέπει να μπει και στην στοίβα το συγκεκριμένο στοιχείο. Επομένως κοιτάμε εάν έχει γίνει η στοίβα(flag==1), εάν το άθροισμα του στοιχείου είναι μεγαλύτερο του κορυφαίου στοιχείου της στοίβας(summary>Wk[0][1]) και εάν το στοιχείο τυχαίνει να υπάρχει ήδη στην στοίβα (line1[0] not in heap_IDs). Είναι προφανές πως άμα το άθροισμα περνάει την τιμή κορυφαίου(μικρότερου) στοιχείου της στοίβας πρέπει να μπει. Ωστόσο πρέπει πρώτα να ελέγξουμε εάν το αντικείμενο είναι ήδη στην στοίβα λόγω του κάτω ορίου του. Αν δεν είναι μέσα στη στοίβα ήδη, απλά αφαιρούμε το μικρότερο/ κορυφαίο αντικείμενο από τη στοιβά, καθώς και το αντίστοιχο ID από το set και βάζουμε τα αντίστοιχα του νέου αντικείμενου. Έπειτα ταξινομούμε[1] τη στοίβα. Εάν είναι ήδη μέσα στη στοίβα λόγω του κάτω ορίου του, θα πρέπει να βγει το ίδιο το αντικείμενο προκειμένου να ξανά μπει ανανεωμένο, με την τελική του τιμή.

```
            #If heap has been created and summary > value of the top heap element..
            #..and the object isn't already in the heap
            if flag == 1 and summary > Wk[0][1] and line1[0] not in heap_IDs:
                heap_IDs.discard(Wk[0][0]) #Discard first object id from set
                Wk.pop(0) #Pop first object from heap
                heapq.heappush(Wk, seen[line1[0]])#Push the new object in heap whose sum was greater
                sort_heap()#Sort the heap
                heap_IDs.add(line1[0])#Add the id of the newly added object to the ID's set
            #If heap has been created and summary > value of the top heap element..
            #..but the object is already in the heap
            elif flag == 1 and summary > Wk[0][1] and line1[0] in heap_IDs:
                #Search for the index that shows its position in the heap
                index = next((i for i, tuple in enumerate(Wk) if tuple[0] == line1[0]), -1)
                #Once found, remove the outdated object.Add the updated version and sort the heap
                if index != -1:
                    Wk.pop(index)
                    heapq.heappush(Wk, seen[line1[0]])
                    sort_heap()
        else: #If this is the first time we encounter this object ID
            #Get sum of line value + rnd source value.Round it to 2 decimals
            summary = round(sum([float(line1[1]), float(R[int(line1[0])])]), 2)
            #Since this is the first time we encounter this object in seq1, we put "line1"
            seen[line1[0]] = ((line1[0], summary, "line1"))
```

Επομένως ψάχνουμε τον δείκτη θέσης που βρίσκεται το αντικείμενο και το αφαιρούμε. Έπειτα βάζουμε επανατοποθετούμε βάζοντας την τελική του τιμή και ξανά κάνουμε sort τη στοίβα.

Τώρα στη περίπτωση που μόλις διαβάσαμε ένα νέο αντικείμενο για πρώτη φορά, υπολογίζουμε το κάτω όριό του και το βάζουμε στο λεξικό seen, για κλειδί line1[0] (που είναι το ID), με αναγνωριστικό “line1”, μιας και το έχουμε δει μόνο από το seq1.

```
            #If heap hasn't been created yet and we've reached k objects read, create the heap.
            if flag == 0 and len(seen) == k:
                declare_min_heap()
                sort_heap()
                flag = 1 #Set flag to 1 since the heap has been created
            elif flag == 1 and summary > Wk[0][1]: #If heap already exists and sum> top element
                heap_IDs.discard(Wk[0][0]) #Remove top object id from the id's set
                Wk.pop(0) #Remove top object from heap
                heapq.heappush(Wk, seen[line1[0]]) #Add new object
                sort_heap() #Sort heap
                heap_IDs.add(line1[0]) #Add new object ID to the id's set

        if check_exit(flag,last1,last2,exit) == 1:
            break
```

Αφού μόλις διαβάσαμε ένα νέο αντικείμενο πρέπει να ελέγξουμε άμα φτάσαμε το πλήθος αντικειμένων ίσο με k. Γιατί σύμφωνα με την εκφώνηση τότε πρέπει να δημιουργηθεί η στοίβα. Εάν το έχουμε φτάσει καλούμε τη συνάρτηση δημιουργίας της στοίβας[2], την ταξινομούμε και θέτουμε το flag = 1. Εάν υπάρχει ήδη επαναλαμβάνουμε την ίδια διαδικασία ελέγχου εάν το αντικείμενο πρέπει να τοποθετηθεί στη στοίβα.

Τέλος, καλούμε τη συνάρτηση ελέγχου εξόδου[3].

Κατά αντιστοιχία για το seq2:

```
        exit = 1#Binary value. Once we do the condition check, if exit == 1 break.
        line2 = seq2.readline().split()#line2 = [ID,value]
        if line2: #If next line is not null
            sequential_accesses += 1 #Increase sequential accesses by 1
            last2 = float(line2[1]) #Cast the value field to float
            if line2[0] in seen: #If we've already read this ID from the other source file
                #Add the previous lower bound with the new line source value
                #Round it to 2 decimals
                summary = round(sum([float(seen[line2[0]][1]), float(line2[1])]), 2)
                # Store in dictionary. seen[id] = (id,value,string)
                # String can be "line1","line2" or "both" depending on where we've seen..
                # ..it so far. Since this is the second time we encounter the same id we put "both"
                seen[line2[0]] = ((line2[0]), summary, "both")
                # If heap has been created and summary > value of the top heap element..
                # ..and the object isn't already in the heap
                if flag == 1 and summary > Wk[0][1] and line2[0] not in heap_IDs:
                    heap_IDs.discard(Wk[0][0])#Discard first object id from set
                    Wk.pop(0)#Pop first object from heap
                    heapq.heappush(Wk, seen[line2[0]])#Push the new object in heap whose sum was greater
                    sort_heap()#Sort the heap
                    heap_IDs.add(line2[0])#Add the id of the newly added object to the ID's set
                elif flag == 1 and summary > Wk[0][1] and line2[0] in heap_IDs:
                    #Search for the index that shows its position in the heap
                    index = next((i for i, tuple in enumerate(Wk) if tuple[0] == line2[0]), -1)
```

```
                #Once found, remove the outdated object.Add the updated version and sort the heap
                if index != -1:
                    Wk.pop(index)
                    heapq.heappush(Wk, seen[line2[0]])
                    sort_heap()
            else:#If this is the first time we encounter this object ID
                #Get sum of line value + rnd source value.Round it to 2 decimals
                summary = round(sum([float(line2[1]), float(R[int(line2[0])])]), 2)
                #Since this is the first time we encounter this object in seq2, we put "line2"
                seen[line2[0]] = ((line2[0], summary, "line2"))
                #If heap hasn't been created yet and we've reached k objects read, create the heap.
                if flag == 0 and len(seen) == k:
                    declare_min_heap()
                    sort_heap()
                    flag = 1 #Set flag to 1 since the heap has been created
                elif flag == 1 and summary > Wk[0][1]:#If heap already exists and sum> top element
                    heap_IDs.discard(Wk[0][0])#Remove top object id from the id's set
                    Wk.pop(0)#Remove top object from heap
                    heapq.heappush(Wk, seen[line2[0]])#Add new object
                    sort_heap()#Sort heap
                    heap_IDs.add(line2[0])#Add new object ID to the id's set

            if check_exit(flag, last1, last2, exit) == 1:
                break
```

Τέλος ο έλεγχος εξόδου εάν έχουμε φτάνει στο τέλος του αρχείου για seq1, seq2:

```
        #If EOF for either seq1 or seq2, break
        if not line1 or not line2:
            break
```

1. Συνάρτηση ταξινόμησης:

```
#Sort heap tuples
def sort_heap():
    global Wk
    #In our case heap consists of 3-element tuples(objects)
    #So heapq.heapify() may not work properly.
    Wk = sorted(list(Wk), key=lambda tuple: tuple[1])[:] #Sort Wk, comparing the fied of "summary"
    heap_IDs = list(Wk)[:][0] # : is a slicer. It helps iterate through, keeping the ID fields in the process
```

Καθώς η άσκηση ζητά η στοίβα να δημιουργείται μόνο εφόσον έχουμε διαβάσει τα πρώτα k αντικείμενα, δεν πρέπει να έχει δημιουργηθεί από την αρχή. Αντ' αυτού δημιουργείται μόνο εφόσον διαβάσουμε k αντικείμενα μέσω αυτής της συνάρτησης. Γι' αυτό πρέπει να δηλωθεί ως global Wk για να είναι ορατή από όλες τις υπόλοιπες συναρτήσεις. Αν και η βιβλιοθήκη heapq προσφέρει την συνάρτηση heapq.heapify(), με σκοπό την μετατροπή σε minheap και κατ' επέκταση την ταξινόμησή της, δεν επιφέρει τα επιθυμητά αποτελέσματα. Για το λόγο αυτό δημιουργήθηκε η συγκεκριμένη συνάρτηση προκειμένου να κάνει sorting των αντικειμένων της Wk με βάση την τιμή των αθροισμάτων του εκάστοτε αντικειμένου. Έπειτα για κάθε ένα ταξινομημένο αντικείμενο κρατάμε το ID του σε ένα set.

2. Δημιουργία min heap

```python
#Create min heap
def declare_min_heap():
    global Wk #Since Wk is created inside a function, "global" is needed
    Wk = sorted(seen.values())[:] #Put the elements that we've seen, sorted, inside the min heap
    heap_IDs = {tuple[0] for tuple in Wk} #Keep the heap ID's in a set.
```

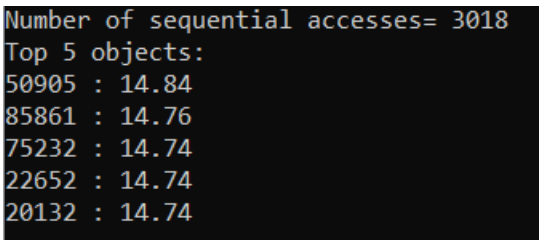
Καθώς η άσκηση ζητά η στοίβα να δημιουργείται μόνο εφόσον έχουμε διαβάσει τα πρώτα k αντικείμενα, δεν πρέπει να έχει δημιουργηθεί από την αρχή. Αντ' αυτού δημιουργείται μόνο εφόσον διαβάσουμε k αντικείμενα μέσω αυτής της συνάρτησης. Γι' αυτό πρέπει να δηλωθεί ως global Wk για να είναι ορατή από όλες τις υπόλοιπες συναρτήσεις. Η seen.values() μας επιστρέφει όλες τις τιμές του λεξικού seen. Η sorted τις επιστρέφει ταξινομημένες και ο slicer [:] κάνει iteration μία – μία αυτές τις τιμές, τις οποίες και αναθέτουμε στην Wk. Τέλος κρατάμε τα IDs στο set.

3. Έλεγχος εξόδου

```python
# Exit check
def check_exit(flag,last1,last2,exit):
    threshold = last1 + last2 + 5.0  # Calculate threshold as instructed
    # We should bother with this only if the heap has been created. In other words only after..
    # ..we've encountered k objects. If threshold <= Wk[k-1][1] we've passed the first exit condition
    if flag == 1 and threshold <= Wk[k - 1][1]:
        # We should exit only if there are no potential objects that we've seen with values greater than..
        # ..the top heap object. In other words, in order to exit, we need the potential upper bound of all the objects..
        # ..that we've seen, that are not in the heap, to be less than the value of the top heap object.
        # For the objects that we've read in seq1, the max possible value is last2 and for the ones from seq2, it is..
        # .. last1. Since those files are sorted, the rest of the unread lines could only be as high as the last one we've read.
        # So even if the rest of the seen elements would get the max possible value, it would still not be enough to get into the heap.
        if any(key not in heap_IDs and (seen[key][2] == "line1" and seen[key][1] + last2 > Wk[0][1]) or \
               (seen[key][2] == "line2" and (seen[key][1] + last1 > Wk[0][1])) for key in seen):
            exit = 0
        return exit
```

Τα ορίσματα είναι το flag, τα last1/ last2 και το exit. Προκειμένου να γίνουν η παρακάτω έλεγχοι χρειαζόμαστε τη στοίβα και άρα το flag == 1. Υπολογίζουμε το threshold όπως περιγράφεται στην εκφώνηση της άσκησης. Το Wk[k-1][1] το πιο μεγάλο αντικείμενο της στοίβας. Εάν το threshold > του μεγαλύτερου στοιχείου, δεν έχει καν νόημα να μιλάμε για έξοδο από το πρόγραμμα ακόμα καθώς, υπάρχει πιθανότητα να υπάρχουν και άλλα αντικείμενα μεγαλύτερα που πρέπει να συμπεριληφθούν στη στοίβα. Εάν όμως είναι μικρότερο ίσο υπάρχει περίπτωση να μπορούμε να βγούμε από τώρα χωρίς να διαβάσουμε όλα τα στοιχεία από seq1, seq2. Σε αυτό το σημείο θα φανούν ιδιαίτερα χρήσιμα τα αναγνωριστικά που μας δείχνουν ποια είναι η πηγή από την οποία έχουμε διαβάσει ένα αντικείμενο. Εάν είναι “both” δεν μας απασχολεί καθώς σημαίνει ότι το έχουμε διαβάσει και από τις δύο πηγές και εάν μας ενδιέφερε, θα βρισκόταν ήδη μέσα στη στοίβα έως τώρα. Αυτά που μας ενδιαφέρουν είναι τα αντικείμενα που έχουν αναγνωριστικά “line1”, “line2”. Επίσης κρατάμε τις τελευταίες τιμές που διαβάστηκαν από τα seq1,seq2 στα last1,last2 αντίστοιχα. Εφόσον τα seq1,seq2 είναι ταξινομημένα αυτό σημαίνει ότι η last τιμή που έχουμε διαβάσει αποτελεί την μέγιστη δυνατή που μπορούν να έχουν από εκεί και πέρα οι υπόλοιπες γραμμές του αντίστοιχου αρχείου, στην καλύτερη. Επομένως κοιτάμε για τα στοιχεία που δεν είναι στην στοίβα, με αναγνωριστικά “line1”, “line2” εάν το άθροισμα της τιμής του κάτω ορίου τους + την τιμή που πιθανώς στην καλύτερη θα μπορούσαν να έχουν από την άλλη πηγή (το last της άλλης πηγής), να είναι μεγαλύτερη από το μικρότερο στοιχείο της στοίβας. Εάν ναι, αυτό σημαίνει ότι πιθανώς, θα μπορούσε να υπάρχει στοιχείο που άμα διαβάσουμε και την τιμή που έχει στην άλλη πηγή να αποκτήσει τελική τιμή, αρκετά μεγάλη για να βρίσκεται στην στοίβα. Αυτό δεν είναι σίγουρο, πάντως σε καμία περίπτωση δεν μπορούμε να τερματίσουμε, μέχρι να κλείσει το ενδεχόμενο ύπαρξης αυτών των πιθανώς μεγαλύτερων τιμών. Το exit είναι αρχικοποιημένο στο 1. Εάν πρέπει να συνεχίσουμε το βάζουμε 0 προκειμένου να μην γίνει break.

Αποτελέσματα:



Εάν και τα αποτελέσματα είναι τα επιθυμητά με το υπόδειγμα που μας δόθηκε, παρατηρείται διαφορετική σειρά σε κάποια από τα στοιχεία με ίδια τιμή. Καθώς κατά το sorting δεν είναι δυνατό να δοθεί προτεραιότητα σε κάποιο στοιχείο, έναντι κάποιου άλλου χωρίς παραπάνω κριτήρια ταξινόμησης.

```
Number of sequential accesses= 5616
Top 10 objects:
50905 : 14.84
85861 : 14.76
75232 : 14.74
20132 : 14.74
22652 : 14.74
21824 : 14.7
9041 : 14.66
97866 : 14.65
83759 : 14.64
96407 : 14.58
```

```
Number of sequential accesses= 9046
Top 20 objects:
50905 : 14.84
85861 : 14.76
75232 : 14.74
22652 : 14.74
20132 : 14.74
21824 : 14.7
9041 : 14.66
97866 : 14.65
83759 : 14.64
96407 : 14.58
35055 : 14.57
78315 : 14.54
594 : 14.54
16564 : 14.54
33288 : 14.52
79330 : 14.51
11283 : 14.49
4885 : 14.47
9745 : 14.45
55165 : 14.44
```

Για k έως και μερικές χιλιάδες ο χρόνος εκτέλεσης είναι μηδαμινός και μεγαλώνει για k >= 10000.

Με σκοπό τον έλεγχο της ορθότητας του προγράμματος και της εγκυρότητας των παραχθέντων αποτελεσμάτων αναπτύχθηκε και κώδικας brute force ο οποίος παράγει ένα αρχείο με όλα τα 100000 αποτελέσματα για κάθε ID. Αν και δεν θα συμπεριληφθεί μεταξύ των παραδοθέντων αρχείων, παρατίθεται παρακάτω:

```
R = []
S1 = []
S2 = []
results = []

#Add contents of rnd to R array.
def add_from_rnd():
    with open(rndPath, mode='r') as rnd: #Open rnd.txt for reading
        for line in rnd: #For each line
            splitLine = line.split() #Split line to get it's contents. id,rating
            R.append(float(splitLine[1])) #Add to array

def add_from_seq1():
    with open(seq1Path, mode='r') as seq1: #Open rnd.txt for reading
        for line in seq1: #For each line
            splitLine = line.split() #Split line to get it's contents. id,rating
            S1.append([int(splitLine[0]),float(splitLine[1])]) #Add to array

def add_from_seq2():
    with open(seq2Path, mode='r') as seq2: #Open rnd.txt for reading
        for line in seq2: #For each line
            splitLine = line.split() #Split line to get it's contents. id,rating
            S2.append([int(splitLine[0]),float(splitLine[1])]) #Add to array
```

```python
def calculate_results():
    for i in range(len(R)):
        sum = round(R[i] + S1[i][1] + S2[i][1],2)
        results.append([i,sum])

def create_results_txt():
    with open(resultsPath, 'w') as res:
        for i in results:
            line = f"{i[0]}: {i[1]}"
            res.write(line + '\n')
```

```python
add_from_rnd()
add_from_seq1()
add_from_seq2()
S1 = sorted(S1, key=lambda x: x[0])
S2 = sorted(S2, key=lambda x: x[0])
calculate_results()
results = sorted(results, key=lambda x: x[1], reverse=True)
create_results_txt()
```