ΠΑΝΕΠΙΣΤΉΜΙΟ ΙΩΑΝΝΊΝΩΝ

ΤΜΉΜΑ ΜΗΧΑΝΙΚΏΝ ΗΛΕΚΤΡΟΝΙΚΏΝ ΥΠΟΛΟΓΙΣΤΏΝ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ

Διαχείριση Σύνθετων Δεδομένων

Εργασία 3 – Ερωτήσεις κορυφαίων κ και κορυφογραμμής

ΣΙΑΚΑΒΑΡΑ ΘΕΜΙΣΤΟΚΛΕΙΑ, 4786

ΠΑΡΑΔΟΣΗ: ΠΑΡΑΣΚΕΥΗ 31 ΜΑΪΟΥ 2024

Για την υλοποίηση της τρίτης εργασίας, χρησιμοποιήθηκε η γλώσσα προγραμματισμού python, και πιο συγκεκριμένα η έκδοση 3.10.6.

Σε όλα τα προγράμματα έχει γίνει import sys, για να διαβάσουμε ορίσματα από τη γραμμή διαταγών, import heapq και import time για να βρούμε τον χρόνο εκτέλεσης τους.

1ο Μέρος:

Το ζητούμενο πρόγραμμα για το πρώτο μέρος ονομάζεται **meros1.py** και παίρνει σαν όρισμα έναν αριθμό K. Για παράδειγμα:

```
>python meros1.py 5
```

Στη main συνάρτηση του προγράμματος, ελέγχεται πρώτα αν έχει δοθεί ο σωστός αριθμός ορισμάτων. Αν όχι, το πρόγραμμα τερματίζει, αν ναι τότε αρχίζει να μετράει ο χρόνος εκτέλεσης και ανοίγουν τα δύο αρχεία με τα δεδομένα (females_sorted, males_sorted). Αφού αρχικοποιηθούν κατάλληλα οι μεταβλητές που θα χρησιμοποιηθούν, καλείται η συνάρτηση topKjoin() και σε ένα loop από το 0 μέχρι το K που έχει οριστεί από τη γραμμή εντολών, το πρόγραμμα βρίσκει το επόμενο αποτέλεσμα. Η topKjoin() είναι general function, γι αυτό και όταν καλείται αποθηκεύεται σε μία μεταβλητή, η οποία με το next() βρίσκει το επόμενο αποτέλεσμα κάθε φορά. Όταν τελειώσει, σταματάει και η καταγραφή του χρόνου και τυπώνεται ο συνολικός χρόνος εκτέλεσης.

```python
if __name__ == "__main__":
    if len(sys.argv)<2:
        print("Usage: python program.py K")
        sys.exit(1)

    K = int(sys.argv[1])

    startTime = time.time()
    with open('males_sorted', 'r') as malesFile, open('females_sorted', 'r') as femalesFile:
       Q = []
       T = 0
       p1_max, p1_cur, p2_max, p2_cur = 0, 0, 0, 0
       males = {}
       females = {}

       result = topKjoin(malesFile, femalesFile)
       print(f"Top-{K} pairs:")
       idx=1
       for count in range(0,K):
           joinR = next(result)
           print(f"{idx}. pair: {joinR[0]} score: {joinR[1]:.2f}")
           idx +=1
    endTime = time.time()

    print(f"Runtime: {endTime - startTime: .4f} seconds")
```

Για τη βασική συνάρτηση του προγράμματος, δηλαδή την topKjoin(), ορίστηκαν και κάποιες βοηθητικές συναρτήσεις, οι οποίες είναι οι εξής:

- getImportantFields(fields): Παίρνει σαν όρισμα μία γραμμή αρχείου fields, δηλαδή όλα τα πεδία. Επιστρέφει τα πεδία της εγγραφής, της ηλικίας, και του βάρους.
- writeMalesData(malesFile): Παίρνει σαν όρισμα το αρχείο με τα δεδομένα των αντρών. Αν η γραμμή που διαβάζει είναι έγκυρη, τότε ορίζει κατάλληλα τις p1_cur και p1_max και προσθέτει τον τρέχων άντρα στο hash table males. Αν δεν είναι έγκυρη, ξανά καλεί την συνάρτηση αυτή. Επίσης, ελέγχει αν βρίσκεται στο τέλος του αρχείου, απλά προσθέτει ένα null δεδομένο στο males. Επιστρέφει τον αριθμό του άντρα και τα p1_max, p1_cur.
- writeFemalesData(femalesFile): Αντίστοιχη συνάρτηση με αυτή των αντρών αλλά με το αρχείο των γυναικών. Άρα, παίρνει σαν όρισμα το αρχείο με τα δεδομένα των γυναικών. Αν η γραμμή που διαβάζει είναι έγκυρη, τότε ορίζει κατάλληλα τις p2_cur και p2_max και προσθέτει την τρέχουσα γυναίκα στο hash table females. Αν δεν είναι έγκυρη, ξανά καλεί την συνάρτηση αυτή. Επίσης, ελέγχει αν βρίσκεται στο τέλος του αρχείου.. Επιστρέφει τον αριθμό της γυναίκας και τα p2_max, p2_cur.

```
def getImportantFields(fields):
    recordID = int(fields[0])
    age = int(fields[1])
    instanceWeight = float(fields[25])
    return (recordID, age, instanceWeight)

def writeMalesData(malesFile):
    global p1_max
    global p1_cur

    malesLine = malesFile.readline().strip().split(',')
    if len(malesLine) < 2:
        if 0 not in males:
            males[0] = []
        males[0].append(())
        return None, p1_max, p1_cur
    if not malesLine[8].startswith(" Married") and int(malesLine[1])>=18:
        p1_cur = float(malesLine[25])
        if len(males)==0:
            p1_max = p1_cur
        malesRecord = getImportantFields(malesLine)
        if malesRecord[1] not in males:
            males[malesRecord[1]] = []
        males[malesRecord[1]].append((malesRecord[0], malesRecord[2]))
        return malesRecord, p1_max, p1_cur
    else:
        return writeMalesData(malesFile)

def writeFemalesData(femalesFile):
    global p2_max

    femalesLine = femalesFile.readline().strip().split(',')
    if len(femalesLine) < 2:
        return None, p2_max, p2_cur
    if not femalesLine[8].startswith(" Married") and int(femalesLine[1])>=18:
        p2_cur = float(femalesLine[25])
        if len(females)==0:
            p2_max = p2_cur
        femalesRecord = getImportantFields(femalesLine)
        if femalesRecord[1] not in females:
            females[femalesRecord[1]] = []
        females[femalesRecord[1]].append((femalesRecord[0], femalesRecord[2]))
        return femalesRecord, p2_max, p2_cur
    else:
        return writeFemalesData(femalesFile)
```

Τέλος η συνάρτηση topKjoin(), η οποία περιέχει τον ζητούμενο αλγόριθμο, παίρνει ως ορίσματα τα δύο αρχεία με τα δεδομένα των αντρών και των γυναικών. Αφού οριστούν κάποιες μεταβλητές, ξεκινάει ένα while loop. Σε αυτό, αν το hash table males είναι κενό, σημαίνει ότι βρισκόμαστε στην αρχή, άρα διαβάζει και από τα δύο αρχεία, βρίσκει τα απαραίτητα δεδομένα, και αν οι ηλικίες είναι ίδιες, ελέγχει το συνολικό βάρος και ανάλογα το προσθέτει στο Q. Αν τα δύο hash tables males και females έχουν το ίδιο μέγεθος, τότε διαβάζει δεδομένα από το αρχείο των αντρών και ελέγχει αν υπάρχει γυναίκα ίδιας ηλικίας με τον τρέχοντα άντρα. Αν ναι, πάλι ελέγχεται το συνολικό βάρος. Αν τίποτα από τα παραπάνω δεν ισχύουν, τότε διαβάζεται το αρχείο των γυναικών, με τον ίδιο τρόπο όπως στην παραπάνω περίπτωση. Τέλος, αν υπάρχουν στοιχεία στην Q τότε αφαιρείται το κορυφαίο στοιχείο από αυτή και επιστρέφεται. Εφόσον θέλουμε η topKjoin να είναι general function, έχουμε yield αντί για return.

```
def topKjoin(malesFile, femalesFile):
    global males
    global females
    global p1_max, p1_cur, p2_max, p2_cur
    global T
    global Q

    totalWeight = 0

    while True:
        if len(males)==0:
            malesRecord, p1_max, p1_cur = writeMalesData(malesFile)
            femalesRecord, p2_max, p2_cur = writeFemalesData(femalesFile)
            T = max(p1_max + p2_cur, p1_cur + p2_max)

            if malesRecord[1] == femalesRecord[1]:
                totalWeight = malesRecord[2] + femalesRecord[2]
                if totalWeight>=T:
                    heapq.heappush(Q, (totalWeight, (malesRecord[0], femalesRecord[0])))
        elif len(males) == len(females):
            malesRecord, p1_max, p1_cur = writeMalesData(malesFile)
            if malesRecord == None:
                continue
            T = max(p1_max + p2_cur, p1_cur + p2_max)

            for woman in females.get(malesRecord[1], []):
                totalWeight = malesRecord[2] + woman[1]
                if totalWeight >= T:
                    heapq.heappush(Q, (totalWeight, (malesRecord[0], woman[0])))
        else:
            femalesRecord, p2_max, p2_cur = writeFemalesData(femalesFile)
            if femalesRecord == None:
                break
            T = max(p1_max + p2_cur, p1_cur + p2_max)

            for man in males.get(femalesRecord[1], []):
                totalWeight = femalesRecord[2] + man[1]
                if totalWeight >= T:
                    heapq.heappush(Q, (totalWeight, (man[0], femalesRecord[0])))
        while Q and Q[0][0]>= T:
            topElement = heapq.heappop(Q)
            yield topElement[1], topElement[0]
```

2ο Μέρος:

Το ζητούμενο πρόγραμμα για το δεύτερο μέρος ονομάζεται **meros2.py** και παίρνει σαν όρισμα έναν αριθμό Κ. Για παράδειγμα:

```
>python meros2.py 5
```

Στη main συνάρτηση ελέγχεται πρώτα αν έχει δοθεί ο σωστός αριθμός ορισμάτων. Αν όχι, βγάζει μήνυμα λάθους και τερματίζει το πρόγραμμα, αλλιώς το πρόγραμμα συνεχίζει κανονικά. Γίνεται έναρξη του χρόνου εκτέλεσης, και αφού ορίσουμε ως Κ τον αριθμό από τη γραμμή εντολών, ανοίγουμε τα δύο αρχεία ‘males_sorted’ και ‘females_sorted’ και καλούμε την topKjoinB(). Το αποτέλεσμα της είναι τα Κ κορυφαία joins, δηλαδή το ζητούμενο της άσκησης, τα οποία τα τυπώνουμε. Ο χρόνος τερματίζει, και τυπώνεται στο τέλος.

```python
if __name__ == "__main__":
    if len(sys.argv)<2:
        print("Usage: python program.py K")
        sys.exit(1)

    startTime = time.time()
    K = int(sys.argv[1])
    with open('males_sorted', 'r') as malesFile, open('females_sorted', 'r') as femalesFile:
        minHeap = topKjoinB(malesFile, femalesFile, K)
        print(f"Top-{K} pairs:")
        for idx, (score, pair) in enumerate(minHeap, start=1):
            print(f"{idx}. pair: {pair} score: {score:.2f}")
    endTime = time.time()

    print(f"Runtime: {endTime - startTime: .4f} seconds")
```

Για τη συνάρτηση topKjoinB() χρειαζόμαστε κάποιες βοηθητικές συναρτήσεις, οι οποίες είναι:

- getImportantFields(fields): Η ίδια συνάρτηση με το meros1.py. Παίρνει σαν όρισμα μία γραμμή αρχείου fields, δηλαδή όλα τα πεδία. Επιστρέφει τα πεδία της εγγραφής, της ηλικίας, και του βάρους.
- readMalesFile(malesFile): Η συνάρτηση αυτή διαβάζει εξολοκλήρου το αρχείο males_sorted και βάζεις τις έγκυρες πλειάδες σε ένα hash table ‘males’, με κλειδί το age. Παίρνει ως όρισμα το αρχείο males_sorted (malesFile). Επιστρέφει το hash table males.

```
def getImportantFields(fields):
    recordID = int(fields[0])
    age = int(fields[1])
    instanceWeight = float(fields[25])
    return (recordID, age, instanceWeight)

def readMalesFile(malesFile):
    males = {}
    malesLine = malesFile.readline().strip().split(',')
    while len(malesLine)>1:
        if not malesLine[8].startswith(" Married") and int(malesLine[1])>=18:
            malesRecord = getImportantFields(malesLine)
            if malesRecord[1] not in males:
                males[malesRecord[1]] = []
            males[malesRecord[1]].append((malesRecord[0], malesRecord[2]))
        malesLine = malesFile.readline().strip().split(',')
    return males
```

Τέλος, η συνάρτηση topKjoinB() παίρνει ως ορίσματα τα δύο αρχεία των αντρών και των γυναικών (malesFile, femalesFile), και τον αριθμό K. Αρχικά, δημιουργεί το hash table males καλώντας την readMalesFile() που αναφέρθηκε παραπάνω. Στη συνέχεια, διαβάζει γραμμή προς γραμμή το αρχείο females_sorted. Αν η γραμμή είναι έγκυρη, και η ηλικία της τρέχουσας γυναίκας υπάρχει στο males, τότε υπολογίζεται το συνολικό τους βάρος. Το join αυτό απλά προστίθεται στο min heap στην περίπτωση που δεν έχει ήδη K μήκος, αλλιώς ελέγχεται πρώτα αν είναι μικρότερο από το πρώτο στοιχείο του min heap (το οποίο είναι το μικρότερο στοιχείο προς το παρών), και αν είναι τότε προστίθεται αυτό και αφαιρείται το τελευταίο στοιχείο, αλλιώς δεν αλλάζει τίποτα στο min heap.

Αφού διαβαστεί όλο το αρχείο females_sorted και έχουμε τα αποτελέσματα, ταξινομούμε το min heap από το μεγαλύτερο αρχείο στο μικρότερο. Επιστρέφουμε το min heap.

```
def topKjoinB(malesFile, femalesFile, K):
    males = readMalesFile(malesFile)
    minHeap = []

    for line in femalesFile:
        fields = line.strip().split(',')
        if not fields[8].startswith(" Married") and int(fields[1]) >=18:
            femaleRecord = getImportantFields(fields)
            age = femaleRecord[1]
            if age in males:
                for maleRecord in males[age]:
                    totalWeight = maleRecord[1] + femaleRecord[2]
                    if len(minHeap)<K:
                        heapq.heappush(minHeap, (totalWeight, (maleRecord[0], femaleRecord[0])))
                    else:
                        if totalWeight > minHeap[0][0]:
                            heapq.heappushpop(minHeap, (totalWeight, (maleRecord[0], femaleRecord[0])))
    minHeap.sort(reverse=True, key=lambda x: x[0])
    return minHeap
```

3ο Μέρος:

Το ζητούμενο πρόγραμμα για το δεύτερο μέρος ονομάζεται **meros3.py** και δεν παίρνει κάποιο όρισμα. Γίνεται import matplotlib.pyplot as plt στην αρχή του, για τη δημιουργία του γραφήματος.

```
>python meros3.py
```

Το 3ο μέρος περιέχει ολόκληρους τους κώδικες του 1ου και του 2ου μέρους, με τη διαφορά ότι αντί να τρέχουν οι δύο αλγόριθμοι στην main, δημιουργήθηκαν οι συναρτήσεις algorithmA και algorithmB αντίστοιχα. Παίρνουν σαν όρισμα τον αριθμό K, δηλαδή τον αριθμό των joins. Η πρώτη επιστρέφει τον χρόνο εκτέλεσης και τον αριθμό των έγκυρων γραμμών που διαβάστηκαν από το males_sorted, και τον αριθμό των έγκυρων γραμμών που διαβάστηκαν από το females_sorted. Αυτά υπολογίστηκαν προσθέτοντας κάποιες έξτρα εντολές στον κώδικα της πρώτης άσκησης. Η δεύτερη συνάρτηση επιστρέφει μόνο τον χρόνο εκτέλεσης.

Στη main συνάρτηση του προγράμματος, ορίζουμε έναν πίνακα με τις K τιμές που θα τρέξουμε τους δύο αλγορίθμους. Για κάθε μία από τις τιμές, γεμίζει τους δυο πίνακες με τους χρόνους που κάνει ο κάθε αλγόριθμος. Τυπώνονται επίσης οι έγκυρες γραμμές που διαβάστηκαν από τα δύο αρχεία για κάθε K. Αφού γίνει αυτό για όλες τις τιμές, φτιάχνουμε το γράφημα και το αποθηκεύουμε ως 'topKjoin_comparison.png'.

```
if __name__ == "__main__":
    Kvalues = [1,2,5,10,20,50,100]
    timeValues = [1,10,100,1000,10000]

    totalTimesA = []
    totalTimesB = []

    for K in Kvalues:
        timeA, validM, validF = algorithmA(K)
        totalTimesA.append(timeA *1000)
        print("K=", K, ":Valid lines in Males:", validM, "-Females:", validF)

        timeB = algorithmB(K)
        totalTimesB.append(timeB *1000)


    plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)
    plt.plot(Kvalues, totalTimesA, label='Algorithm A', marker='o')
    plt.plot(Kvalues, totalTimesB, label='Algorithm B', marker='s')
    plt.xscale('log')
    plt.yscale('log')
    plt.xticks(Kvalues, [str(k) for k in Kvalues])
    plt.yticks(timeValues, [str(t) for t in timeValues])
    plt.xlabel('K')
    plt.ylabel('Execution Time (milliseconds)')
    plt.title('Execution Time of Top-K Join Algorithms')
    plt.legend()

    plt.savefig('topKjoin_comparison.png')
```

Το αποτέλεσμα της γραφικής αποθηκεύεται σε εικόνα και είναι το εξής: