ASSIGNMENT 2 Complex Data Management on Top-K & Skyline Queries

ΙΩΑΝΝΑ ΠΑΠΑΓΙΑΝΝΗ 2790

Η γλώσσα προγραμματισμού που χρησιμοποιήθηκε και για τα δύο μέρη της άσκησης είναι python3.
Επιπλέον τα αρχεία 2017_ALL.csv, 2017_AST.csv, 2017_BLK.csv, 2017_PTS.csv, 2017_STL.csv, 2017_TRB.csv είναι κατεβασμένα ήδη και έχουν τοποθετηθεί στον ίδιο φάκελο με τα αρχεία part1.py και part2.py

PART 1

Στο μέρος 1 υλοποιούμε τον αλγόριθμο NRA ο οποίος υπολογίζει και βρίσκει τους k κορυφαίους παίχτες με βάση τις επιδόσεις τους στα στατιστικά που θα επιλέξουμε απο command line.

Αρχικά κάνουμε import csv για να γράφουμε το csv αρχείο μας με το κατάλληλο format και import itemgetter απο operator που θα μας βοηθήσει στο να ταξηνομήσουμε κάθε φορά με το κατάλληλο value το αρχείο μας.

```
import csv
from operator import itemgetter
```

Παρακάτω για λόγους ευκολίας, το κάθε αρχείο το θέτουμε σε μια μεταβλήτη.

```
allStats='2017_ALL.csv'
assistStats='2017_AST.csv'
pointStats='2017_PTS.csv'
stealStats='2017_STL.csv'
blockStats='2017_BLK.csv'
reboundStats='2017_TRB.csv'

if __name__ == '__main__':

    inputs=getInput()
    chosenCategories=inputs[0]
    k=inputs[1]
    counter=0

    with open('topks.csv', 'w', encoding='UTF-8') as rp1:
        csv_writer = csv.writer(rp1, delimiter=',')

        for topks in topKEvaluation(chosenCategories, k):

            if k==0:

                print('No results. 0 num of numOfAccesses. Terminate.')
                break

            elif counter<k:

                csv_writer.writerow(topks[:-1])
                counter+=1

            else:
                accesses=['#ACCESSES', topks[-1]]
                csv_writer.writerow(accesses)
                break
```

Η main() αρχικά καλεί την *getInput()* και το αποτέλεσμά της το καταχωρεί στην μεταβλητή inputs.

Το inputs[0] είναι για τις κατηγορίες που έχουμε επιλέξει και το inputs[1] για το k.

Αρχικοποιούμε counter=0.

Ανοίγουμε το αρχείο *topks.csv* για γράψιμο, ορίζουμε csv_writer με delimiter comma και καλούμε μια generator συνάρτηση *topKEvaluation()* με ορίσματα τις επιλεγμένες κατηγορίες και το k, η οποία θα μας επιστρέφει τα αποτελέσματα στο τέλος του evaluation τους.

Αν το k=0 δεν υπάρχουν αποτελέσματα οπότε εκτυπώνουμε το κατάλληλο μήνυμα το αρχείο *topks.csv* είναι κενό και τερματίζουμε.

Αλλίως αν counter<k τότε γράφουμε σε γραμμή χωρισμένο με κόμμα στο αρχείο μας τον k παίχτη κάθε φορά και αυξάνουμε τον counter κατά 1.

Όταν ο counter φτάσει το k τότε σημαίνει ότι έχουμε γράψει όλους τους k παίχτες οπότε είναι ώρα να γράψουμε και το αριθμό προσβάσεων που είχαμε (για κάθε αρχείο το κάθε id του παίχτη αντιστοιχίζεται σε ένα access).

```
def getInput():
```

```
chosenCategories=[]
args=[]
checked=0

print('------------------------------NBA STATS 2017------------------------------')
print('------------------------------TOK K PLAYERS ------------------------------\n')
print('Choose which of the following statCategories you are interested in: ')
print('1. Rebounds          2. Assists          3. Steals          4. Blocks          5. Points')

while checked==0 or len(args)!=2:
        args=input('\nGive the numbers with comma in [ ] and then, the number of the Top-k players you are looking for:\n').split(' ')
        checked=checkArgs(args, chosenCategories)

k=int(args[1])
return [chosenCategories, k]
```

Η *getInput()* είναι υπεύθυνη για να επιστρέψει στην *main()* τις παραμέτρους που δώσαμε στη γραμμή διαταγών μέσα σε αγγύλες [,] και το επιθυμητό αριθμό k παιχτών που θέλουμε να βρούμε.

Αρχικοποιούμε *checked=0* το οποίο ειναι ένα flag και γίνεται 1 μόνο όταν η συνάρτηση *checkArgs()* μας το επιστρέψει. Διαφορετικά, λαμβάνουμε μήνυμα ότι πρέπει να ξαναδώσουμε input.

Εφόσον περάσει τον “έλεγχο” επιστρέφει τις επιλεγμένες κατηγορίες (*chosenCategories[]*) και το k.

---

def checkArgs(args, cat):

```
if args[0][0]!='[' or args[0][len(args[0])-1]!=']':
        print('\nInsert the chosen categories in [ ]\n')
        return 0

for i in args[0]:

        if i.isdigit():

                if int(i)>=1 and int(i)<=5:
                        cat.insert(len(cat), int(i))
                else:
                        print('\nInsert number from 1-5.\n')
                        return 0

        elif i=='[' or i==']' or i==',':
                pass
        elif i=='-':
                print('\nInsert number from 1-5.\n')
                return 0

        else:
                print('\nInsert number from 1-5.\n')
                return 0

try:
        k=int(args[1])

except ValueError:
        print('\nInsert integer for k.\n')
        return 0

except IndexError:
        print('Insert integer for k.\n')
        return 0

if k<0 or k>595:
        print( '\nAll players are 595.\n')
        return 0

return 1
```

Η συνάρτηση *checkArgs()* καλείται από την *getInput()* όπως είδαμε και σκοπός είναι να ελέγξει το input για τυχόν αστοχίες (αστοχίες τρόπου γραφής, αστοχίες τιμών).

Τρόπος γραφής: <**[**> <num> <**,**> <num> <**]**> <**1 space**> <num>

ASSIGNMENT 2 Complex Data Management on Top-K & Skyline Queries

Τιμές: Για τις κατηγορίες μέσα στις αγγύλες θέλουμε int από 1 ως 5 και απο ένα ως πέντε #αριθμών.
Για το k θέλουμε int από 0 ως 595 που είναι όλοι μας οι παίχτες.

Επιστρέφει 0 αν βρεί οτιδήποτε εκτός απο την επιθυμητή “φόρμα” που θέλουμε να είναι γραμμένα και 1 αν όλα είναι εντάξει.

---

def topKEvaluation(cat, k):

```
    hashMap={}
    hashMap1={}
    hashMap2={}
    hashMap3={}
    hashMap4={}
    hashMap5={}
    upperBoundsDict={}
    W=[]
    u=5.0
    t=0
    canYield=0
    numOfAccesses=0
    firstRow=1
...
```

Αρχικοποιούμε τα dictionaries που θα χρησιμοποιήσουμε παρακάτω στο κενό .

Το *hashMap* θα κρατάει τα πάντα από το αρχείο 2017_ALL ώστε γνωρίζοντας μετά το id του κάθε παίχτη να μπορούμε να γραφτεί με το όνομα του στο αρχείο *topks.csv* και με τα στατιστικά του δίπλα για τις επιλεγμένες κατηγορίες.

Τα *hashMap1,...hashMap5* είναι για να αποθηκεύουμε κάθε φορά τα data που συναντάμε διαβάζοντας γραμμή γραμμή τα αρχεία.

Το *upperBoundsDict* είναι υπεύθυνο να αποθηκεύει κάθε φορά τα αντίστοιχα upper bounds του κάθε id εκτός αυτών που βρίσκονται στην W λίστα.

Το W είναι μια priorityQueue (στη κορυφή της έχει τα μεγαλύτερα ids και τις normalized αποδόσεις τους) και είναι υπεύθυνη να αποθηκεύει τα μεγαλύτερα k-οστά lower bounds.

Όπου u το μεγαλύτερο upper bound της upperBoundsDict (το αρχικοποιούμε στο 5.0 γιατί τόσο είναι το μαξ που μπορεί να πάρει για 5 κατηγορίες που είναι το μέγιστο που μπορούμε να δώσουμε ως input).

Όπου t το μικρότερο lower bound της W (το αρχικοποιούμε στο 0 γιατί τόσο είναι το κατώτερο όριο που μπορεί να πάρει).

Η μεταβλητή canYield είναι flag αρχικοποιημένο στο 0 και γίνεται 1 μόνο όταν έχει οριστικοποιηθεί η W.

Η μεταβλήτη numOfAccesses αυξάνεται κάθε φορά που έχουμε νέα access σε εγγραφή των αρχείων.

Η μεταβλητή firstRow είναι flag 1 με το που ανοίγουμε τα αρχεία γιατί θέλουμε να κρατήσουμε πάντα τη πρώτη γραμμή των αρχείων ως το max που συναντάμε (θα χρειαστεί για τον υπολογισμό των normalized αποδόσεων μετά) και κατευθείαν μετά γίνεται 0 και δεν ξαναλλάζει.

```
...
    with open(allStats, 'r', encoding='UTF-8') as df:

        row=df.readline()

        for row in df:
            data=row.split(',')
            data[-1] = data[-1].strip()
            name=data[1]
            team=data[2]
            trb=int(data[3])
            ast=int(data[4])
            stl=int(data[5])
            blk=int(data[6])
            pts=int(data[7])
            hashMap.update({int(data[0]): [name, team, trb, ast, stl, blk, pts]})

    optionFiles={0: allStats, 1: reboundStats, 2: assistStats, 3: stealStats, 4: blockStats, 5: pointStats}
    numOfChoices=len(cat)

    if numOfChoices<5:
```

```
leftovers=5-numOfChoices
for i in range(leftovers):
    cat.append(0)
```

...
Ανοίγουμε το αρχείο πρώτα με όλα τα στατιστικά κα τα ονόματα – id των παιχτών και προσπερνάμε την πρώτη γραμμή που ειναι τα labels.

Για κάθε γραμμή σπάμε τα δεδομένα της γραμμής, αφαιρούμε απο το τελευταίο κομμάτι που πήραμε το \n που έχει στο τέλος και το καθένα το τοποθετούμε στην αντίστοιχη μεταβλητή του (name, team κλπ).

Κάνουμε update στο *hashMap* με όλα τα από πάνω και μετά ορίζουμε ένα dictionary ονόματι *optionFiles* όπου 0 θα δηλώνει ότι είναι το αρχείο με τα allStats, 1 το αρχείο με τα reboundStats κ.ο.κ

Η μεταβλητή numOfChoices είναι ίση με το αριθμό των categories (cat) που έχουμε πάρει ως παράμετρο στη συνάρτηση.

Αν είναι λιγότερο σε πλήθος από 5, κάνουμε append μηδενικά για όσα leftovers έχουμε (δηλαδή το cat μας θα έχει μετά από αυτή την επεξεργασία πάντα 5 στοιχεία πχ πριν: [1,4] → μετά: [1,4,0,0,0]).
Συνεπώς, για numOfChoices :
1, πάντα θα κοιτάμε το cat[0]
2, πάντα θα κοιτάμε το cat[0], cat[1]
3, πάντα θα κοιτάμε το cat[0], cat[1], cat[2]
4, πάντα θα κοιτάμε το cat[0], cat[1], cat[2], cat[3]
5, πάντα θα κοιτάμε το cat[0], cat[1], cat[2], cat[3], cat[4]

...

```
with    open(optionFiles[cat[0]], 'r', encoding='UTF-8') as df1,
        open(optionFiles[cat[1]], 'r', encoding='UTF-8') as df2,
        open(optionFiles[cat[2]], 'r',encoding='UTF-8') as df3,
        open(optionFiles[cat[3]], 'r', encoding='UTF-8') as df4,
        open(optionFiles[cat[4]], 'r', encoding='UTF-8') as df5:

        if numOfChoices==1:

            for row in df1:
                data1=row.split(',')
                topKPlayer=hashMap.get(int(data1[0]))
                yield [str(topKPlayer[0]), int(topKPlayer[cat[0]+1]), numOfAccesses]
                numOfAccesses+=1
```

...
Ανοίγουμε τα αντίστοιχα optionFiles για διάβασμα.

Αν οι επιλογή για στατιστικά που δώσαμε είναι μόνο μία, τότε για κάθε γραμμή στο αντίστοιχο αρχείο κάνουμε split την γραμμή όπου συναντάμε κόμμα και απο το *hashMap* με κλείδι το id του παίχτη παίρνουμε το αντίστοιχο value και κάνουμε yield το όνομα, την κατηγορία που έχει επιλεχθεί (το cat[0] +1 γιατί θέλουμε σε κάθε περίπτωση να προσπερνάμε τη κολώνα με τα ονόματα των ομάδων team) και τον αριθμό των προσβάσεων.
Κάθε φορά μετά το yield αυξάνουμε τις προσβάσεις κατά 1.

...

```
        elif numOfChoices==2:

            for row in zip(df1,df2):
                data1,data2=fixData(row, numOfChoices)

                if firstRow==1:
                    maxv1=data1[1]
                    maxv2=data2[1]
                    firstRow=0

                performance1=normalization(int(data1[1]), maxv1)
                performance2=normalization(int(data2[1]), maxv2)
                hashMap1.update({int(data1[0]): performance1})
                hashMap2.update({int(data2[0]): performance2})
                T=performance1+performance2
                currentIds=[data1[0], data2[0]]
                currentPerformances=[performance1, performance2]
                hashMapsList=[hashMap1, hashMap2]
                R=myNRA(currentIds,currentPerformances,hashMapsList,t,u,T,W,upperBoundsDict,
                                                                numOfChoices, numOfAccesses, k)
                W=R[0]
                t=R[1]
```

```
                    u=R[2]
                    T=R[3]
                    upperBoundsDict=R[4]
                    numOfAccesses=R[5]
                    canYield=R[6]

                    if canYield==1:

                        for ks in W:
                            topKPlayer=hashMap.get(ks[0])
                            yield[str(topKPlayer[0]),int(topKPlayer[cat[0]+1]),int(topKPlayer[cat[1]+1]),
                                                                                    numOfAccesses]



elif numOfChoices==3:

        for row in zip(df1,df2,df3):
                    data1,data2, data3=fixData(row, numOfChoices)

                    if firstRow==1:
                        maxv1=data1[1]
                        maxv2=data2[1]
                        maxv3=data3[1]
                        firstRow=0


                    performance1=normalization(int(data1[1]), maxv1)
                    performance2=normalization(int(data2[1]), maxv2)
                    performance3=normalization(int(data3[1]), maxv3)
                    hashMap1.update({int(data1[0]): performance1})
                    hashMap2.update({int(data2[0]): performance2})
                    hashMap3.update({int(data3[0]): performance3})
                    currentIds=[data1[0], data2[0], data3[0]]
                    T=performance1+performance2+performance3
                    currentPerformances=[performance1, performance2, performance3]
                    hashMapsList=[hashMap1, hashMap2, hashMap3]
                    R=myNRA(currentIds,currentPerformances,hashMapsList,t,u,T,W,upperBoundsDict,
                                                                    numOfChoices, numOfAccesses, k)
                    W=R[0]
                    t=R[1]
                    u=R[2]
                    T=R[3]
                    upperBoundsDict=R[4]
                    numOfAccesses=R[5]
                    canYield=R[6]

                    if canYield==1:

                        for ks in W:
                            topKPlayer=hashMap.get(ks[0])
                            yield[str(topKPlayer[0]), int(topKPlayer[cat[0]+1]), int(topKPlayer[cat[1]+1]),
                                                            int(topKPlayer[cat[2]+1]), numOfAccesses]


elif numOfChoices==4:

        for row in zip(df1,df2,df3,df4):
                    data1,data2, data3, data4=fixData(row, numOfChoices)

                    if firstRow==1:
                        maxv1=data1[1]
                        maxv2=data2[1]
                        maxv3=data3[1]
                        maxv4=data4[1]
                        firstRow=0

                    performance1=normalization(int(data1[1]), maxv1)
                    performance2=normalization(int(data2[1]), maxv2)
                    performance3=normalization(int(data3[1]), maxv3)
                    performance4=normalization(int(data4[1]), maxv4)
                    hashMap1.update({int(data1[0]): performance1})
                    hashMap2.update({int(data2[0]): performance2})
                    hashMap3.update({int(data3[0]): performance3})
                    hashMap4.update({int(data4[0]): performance4})
```

```
                    T=performance1+performance2+performance3+performance4
                    currentIds=[data1[0], data2[0], data3[0], data4[0]]
                    currentPerformances=[performance1,performance2, performance3, performance4]
                    hashMapsList=[hashMap1, hashMap2, hashMap3, hashMap4]
                    R=myNRA(currentIds,currentPerformances,hashMapsList,t,u,T,W,upperBoundsDict,
                                                                numOfChoices, numOfAccesses, k)
                    W=R[0]
                    t=R[1]
                    u=R[2]
                    T=R[3]
                    upperBoundsDict=R[4]
                    numOfAccesses=R[5]
                    canYield=R[6]

                    if canYield==1:

                        for ks in W:
                            topKPlayer=hashMap.get(ks[0])
                            yield[str(topKPlayer[0]), int(topKPlayer[cat[0]+1]), int(topKPlayer[cat[1]+1]),
                                int(topKPlayer[cat[2]+1]), int(topKPlayer[cat[3]+1]), numOfAccesses]


elif numOfChoices==5:

    for row in zip(df1,df2,df3,df4,df5):
                    data1,data2, data3, data4, data5=fixData(row, numOfChoices)

                    if firstRow==1:
                        maxv1=data1[1]
                        maxv2=data2[1]
                        maxv3=data3[1]
                        maxv4=data4[1]
                        maxv5=data5[1]
                        firstRow=0

                    performance1=normalization(int(data1[1]), maxv1)
                    performance2=normalization(int(data2[1]), maxv2)
                    performance3=normalization(int(data3[1]), maxv3)
                    performance4=normalization(int(data4[1]), maxv4)
                    performance5=normalization(int(data5[1]), maxv5)
                    hashMap1.update({int(data1[0]): performance1})
                    hashMap2.update({int(data2[0]): performance2})
                    hashMap3.update({int(data3[0]): performance3})
                    hashMap4.update({int(data4[0]): performance4})
                    hashMap5.update({int(data5[0]): performance5})
                    T=performance1+performance2+performance3+performance4+performance5
                    currentIds=[data1[0], data2[0], data3[0], data4[0], data5[0]]
                    currentPerformances=[performance1,performance2, performance3, performance4,
                                                                        performance5]
                    hashMapsList=[hashMap1, hashMap2, hashMap3, hashMap4, hashMap5]
                    R=myNRA(currentIds,currentPerformances,hashMapsList,t,u,T,W,upperBoundsDict,
                                                                numOfChoices, numOfAccesses, k)
                    W=R[0]
                    t=R[1]
                    u=R[2]
                    T=R[3]
                    upperBoundsDict=R[4]
                    numOfAccesses=R[5]
                    canYield=R[6]

                    if canYield==1:

                        for ks in W:
                            topKPlayer=hashMap.get(ks[0])
                            yield[str(topKPlayer[0]), int(topKPlayer[cat[0]+1]), int(topKPlayer[cat[1]+1]),
                                int(topKPlayer[cat[2]+1]), int(topKPlayer[cat[3]+1]),
                                int(topKPlayer[cat[4]+1]), int(topKPlayer[cat[4]+1]), numOfAccesses]
```

Για τα υπόλοιπα τώρα numOfChoices η λογική που ακολουθούμε είναι όμοια.

Ανοίγουμε με zip μαζικά γραμμή γραμμή από όλα τα αρχεία που έχουμε ως παράμετρο, και καλούμε για την μαζική "row" που έχει προκύψει την συνάρτηση *fixData()* με παράμετρο την "row" και το numOfChoices.

Έχοντας πλέον "φτιαγμένα" data αν είναι η πρώτη γραμμή του αρχείου κρατάμε τα max.
Στην συνέχεια, υπολογίζουμε αντίστοιχα για το καθένα την normalized performance του καλώντας την συνάρτηση

normalization() με παράμετρο την αντίστοιχη απόδοση και το αντίστοιχο max του στατιστικού.

Αφού λάβουμε τα normalized performances τα τοποθετούμε στο αντίστοιχο dictionary με το id ως κλειδί και την απόδοση ως value.

Υπολογίζουμε το T (Threshold) που είναι το άθροισμα κάθε φορά όλων των αποδόσεων της εκάστοτε μαζικής "row".

Τοποθετούμε σε μια λίστα τα currentIds , σε μία άλλη τα currentPerformances και σε μία άλλη τα current dictionaries (hashMapsList).

Όπου R, ότι μας κάνει return κάθε φορά η συνάρτηση *myNRA().*

Το πρώτο που μας επιστρέφει θα είναι η τροποποιημένη λίστα W.
Το δεύτερο που μας επιστρέφει θα είναι το t.
Το τρίτο που μας επιστρέφει θα είναι το νέο u.
Το τέταρτο που μας επιστρέφει θα είναι η τροποποιημένη upperBoundsDict.
Το πέμπτο που μας επιστρέφει θα είναι το numOfAccesses.
Και το τελευταίο που μας επιστρέφει είναι αν είμαστε σε θέση να κάνουμε yield (canYield).

Αν είμαστε έτοιμοι για yield, κάνουμε την αντίστοιχη διαδικασία όπως στο numOfChoices==1 που εξηγήσαμε παραπάνω.

---

def fixData(row, numOfChoices):

```
    if numOfChoices==2:

        data1=list(row[:1])
        data1[-1] = data1[-1].strip()
        data1=data1[0].split(',')
        data1=[int(l) for l in data1]

        data2=list(row[1:2])
        data2[-1] = data2[-1].strip()
        data2=data2[0].split(',')
        data2=[int(l) for l in data2]

        return [data1, data2]

    elif numOfChoices==3:

        data1=list(row[:1])
        data1[-1] = data1[-1].strip()
        data1=data1[0].split(',')
        data1=[int(l) for l in data1]

        data2=list(row[1:2])
        data2[-1] = data2[-1].strip()
        data2=data2[0].split(',')
        data2=[int(l) for l in data2]

        data3=list(row[2:3])
        data3[-1] = data3[-1].strip()
        data3=data3[0].split(',')
        data3=[int(l) for l in data3]

        return [data1, data2, data3]

    elif numOfChoices==4:

        data1=list(row[:1])
        data1[-1] = data1[-1].strip()
        data1=data1[0].split(',')
        data1=[int(l) for l in data1]

        data2=list(row[1:2])
        data2[-1] = data2[-1].strip()
        data2=data2[0].split(',')
        data2=[int(l) for l in data2]

        data3=list(row[2:3])
        data3[-1] = data3[-1].strip()
        data3=data3[0].split(',')
        data3=[int(l) for l in data3]
```

```
            data4=list(row[3:4])
            data4[-1] = data4[-1].strip()
            data4=data4[0].split(',')
            data4=[int(l) for l in data4]

            return [data1, data2, data3, data4]

    elif numOfChoices==5:

            data1=list(row[:1])
            data1[-1] = data1[-1].strip()
            data1=data1[0].split(',')
            data1=[int(l) for l in data1]

            data2=list(row[1:2])
            data2[-1] = data2[-1].strip()
            data2=data2[0].split(',')
            data2=[int(l) for l in data2]

            data3=list(row[2:3])
            data3[-1] = data3[-1].strip()
            data3=data3[0].split(',')
            data3=[int(l) for l in data3]

            data4=list(row[3:4])
            data4[-1] = data4[-1].strip()
            data4=data4[0].split(',')
            data4=[int(l) for l in data4]

            data5=list(row[4:5])
            data5[-1] = data5[-1].strip()
            data5=data5[0].split(',')
            data5=[int(l) for l in data5]

            return [data1, data2, data3, data4, data5]
```

Η συνάρτηση αυτή από raw data μας τα μετατρέπει σε fixed κάνοντας κάθε φορά την επεξεργασία που χρειάζεται για το καθένα.

Πχ:

data1=list(row[:1]) →	['138,1116\n']
data1[-1] = data1[-1].strip() →	['138, 1116']
data1=data1[0].split(',') →	['138', '1116']
data1=[int(l) for l in data1] →	[138, 1116]

Από την κάθε μαζική “row” αρχικά κάνουμε split και κρατάμε σε μια μεταβλητή αντίστοιχα ποιο είναι το data1, data2, .,,κλπ

Έπειτα αφαιρούμε το \n.

Σπάμε τώρα το id απο την απόδοση και στη συνέχεια απο str τα κάνουμε int .

Τέλος επιστρέφουμε τα data1, data2,.. κλπ φτιαγμένα έτοιμα για χρήση.

---

def normalization(rowValue, maxv):

```
    return rowValue/maxv
```

Εδώ κάνουμε το normalization των αποδόσεων, διαιρώντας την εκάστοτε απόδοση με την max της αντίστοιχης κατηγορίας και την επιστρέφουμε.

Για την συνάρτηση *myNRA()* αρκεί να εξηγήσουμε την λογική για numOfChoices=2 γιατί όλες μετά ακολουθούν την ίδια λογική.

def myNRA (currentIds, currentPerformances, hashMapsList, t, u, T, W, upperBoundsDict,numOfChoices,numOfAccesses, k):

```
    if t<u and T!=0:

        if numOfChoices==2:
```

```
numOfAccesses+=2

for i in hashMapsList[0]:

    if i not in hashMapsList[1]:
        f1Lb=hashMapsList[0].get(i)
        f1Ub=hashMapsList[0].get(i)+currentPerformances[1]

    else:
        f1Lb=hashMapsList[0].get(i)+hashMapsList[1].get(i)
        f1Ub=hashMapsList[0].get(i)+hashMapsList[1].get(i)

    upperBoundsDict.update({i:f1Ub})
```

...

Όσο το t<u και T!=0 σημαίνει ότι οποιοδήποτε από τα επόμενα objects είναι πιθανό να ανήκει στα top Ks.
Άρα:
- πρέπει να αυξήσουμε το numOfAccesses κατά τον αντίστοιχο αριθμό numOfChoices που έχουμε
- Για κάθε key (ως τώρα) στην κάθε hashMapsList τσεκάρουμε αν υπάρχει και στις άλλες.

Ανάλογα με το αν υπάρχει ή όχι θα υπολογίζουμε και το lower bound, upper bound.

Για lower bound:

Αν δεν υπάρχει πουθενά το συγκεκριμένο id σε κάποια άλλο hashMap τότε το lower bound είναι μόνο η απόδοση του id του συγκεκριμένου hashMap.

Αν υπάρχει σε κάποιο άλλο hashMap τότε το lower bound θα είναι το αποτέλεσμα της άθροισης των αποδόσεων των hashMap που περιέχουν το συγκεκριμένο id μόνο!

Για upper bound:

Αν δεν υπάρχει πουθενά το συγκεκριμένο id σε κάποια άλλο hashMap τότε το upper bound είναι η απόδοση του id του συγκεκριμένου hashMap, συν όλες τις υπόλοιπες currentPerformances των άλλων hashMap.

Αν υπάρχει σε κάποιο άλλο hashMap τότε το upper bound θα είναι το αποτέλεσμα της άθροισης των αποδόσεων των hashMap που περιέχουν το συγκεκριμένο id,συν όλες τις υπόλοιπες currentPerformances των άλλων hashMap.

Αν υπάρχει σε όλα τα hashMap τότε προφανώς είναι το άθροισμα όλων των αποδόσεων όλων των hashMap του συγκεκριμένου id.

Κάνουμε update έπειτα το upperBoundsDict με το νέο κάθε φορά upper bound για το κάθε id.

...

```
if (f1Lb>t and len(W)==k) or len(W)<k:

    found=0
    pos=0

    for ks in W:
        if ks[0]==i:
            found=1
            W.pop(pos)
            W.insert(pos, [i,f1Lb])
            Wk=sorted(W, reverse=True, key=itemgetter(1))
            W=Wk
            t=W[-1][1]
            break
        else:
            pos+=1

    if found==0 and len(W)<k:
        W.insert(len(W), [i, f1Lb])

    elif found==0 and len(W)==k:
        W.pop(-1)
        W.insert(len(W), [i, f1Lb])

    Wk=sorted(W, reverse=True, key=itemgetter(1))
    W=Wk
    t=W[-1][1]
```

...
Εάν το νέο lower bound που έχει προκύψει είναι μεγαλύτερο από το μικρότερο που υπάρχει ως τώρα στην W και το μέγεθος της W έχει φουλάρει (έχει φτάσει ήδη το k) ή **εάν δεν έχει γεμίσει ακόμα η W,** τότε ανάλογα ακολουθούμε μια στρατηγική.

Χρησιμοποιούμε μια μεταβλητή pos που δηλώνει το position στην W και ένα flag found που από 0 θα γίνει 1 μόνο εάν βρεθεί το συγκεκριμένο id μέσα στην W.

Σαρώνουμε την W και εάν βρούμε το συγκεκριμένο id το αφαιρούμε και στη θέση του εισάγουμε το ίδιο με την πλέον τροποποιημένη του τιμή σε lower bound, οπότε στην συνέχεια το W ταξηνομείται πάλι για να διατηρεί την φθίνουσα σειρά του.

Εάν δεν βρεθεί, και η W δεν έχει ακόμα γεμίσει, απλώς το εισάγουμε στην W.

Εάν δεν βρεθεί, και η W έχει γεμίσει, τότε αφαιρούμε το τελευταίο της W, και εισάγουμε το νέο.

Κάνουμε και μία ταξινόμηση στην συνέχεια και έχουμε την W έτοιμη για να ξαναελέγξουμε για τα επόμενα hashMap.

```
#-------------------------------------------------------------------------

for i in hashMapsList[1]:

        if i not in hashMapsList[0]:
                f2Lb=hashMapsList[1].get(i)
                f2Ub=hashMapsList[1].get(i)+currentPerformances[0]

        elif i in hashMapsList[0]:
                f2Lb=hashMapsList[1].get(i)+hashMapsList[0].get(i)
                f2Ub=hashMapsList[1].get(i)+hashMapsList[0].get(i)

        upperBoundsDict.update({i:f2Ub})

        if (f2Lb>t and len(W)==k) or len(W)<k:

                found=0
                pos=0

                for ks in W:
                        if ks[0]==i:
                                found=1
                                W.pop(pos)
                                W.insert(pos, [i,f2Lb])
                                Wk=sorted(W, reverse=True, key=itemgetter(1))
                                W=Wk
                                t=W[-1][1]
                                break
                        else:
                                pos+=1

                if found==0 and len(W)<k:
                        W.insert(len(W), [i, f2Lb])

                elif found==0 and len(W)==k:
                        W.pop(-1)
                        W.insert(len(W), [i, f2Lb])

                Wk=sorted(W, reverse=True, key=itemgetter(1))
                W=Wk
                t=W[-1][1]
```

...
Είναι ακριβώς η ίδια διαδικασία με από πάνω τώρα ελέγχοντας για το άλλο hashMap.

...
```
for ks in W:
        if ks[0] in upperBoundsDict:
                del upperBoundsDict[ks[0]]

upperBoundsDictK=sorted(upperBoundsDict.items(), reverse=True, key=lambda kv: kv[1])
upperBoundsDict=dict(upperBoundsDictK)

try:
        u=next(iter(upperBoundsDict.values()))
except StopIteration:
        pass
```

...
Αφού ελέγξουμε για όλα τα hashMap, τέλος, για όποιο id βρίσκεται στην W, ελέγχουμε αν υπάρχει στο dictionary του upperBoundsDict και αν το βρούμε το αφαιρούμε από το upperBoundsDict.

Κάνουμε ένα sort στο upperBoundsDict και try να πάρουμε ως u το πρώτο object του upperBoundsDict (υπάρχει περίπτωση στην αρχή να μην έχει γι'αυτο έχουμε pass στην εξαίρεση).

PART 2

Στο μέρος 2, θέλουμε για τους παίχτες του NBA να υπολογίζεται το skyline των παιχτών που δεν κυριαρχούνται από κάποιον άλλον.

Κάνουμε import csv και εδώ για να μπορέσουμε να γράψουμε με την κατάλληλη μορφή στο αρχείο και για λόγους ευκολίας θέτουμε το αρχείο που θα χρειαστούμε ως allStats.

import csv

allStats='2017_ALL.csv'

---

if __name__ == '__main__':

```
chosenCategories=getInput()
skyline=myBNL(chosenCategories)

with open('skyline.csv', 'w', encoding='UTF-8') as rp2:
        csv_writer = csv.writer(rp2, delimiter=',')
        for s in skyline:
                csv_writer.writerow(s)
```

Στην *main()* καλούμε την συνάρτηση *getInput()* για να πάρει από το command line τις επιλεγμένες κατηγορίες, και στην συνέχεια καλούμε την *myBNL()* με παράμετρο τις κατηγορίες που μας επέστρεψε η *getInput().*

Εφόσον η *myBNL()* μας επιστρεψει το skyline, ανοίγουμε το αρχείο *skyline.csv* για γράψιμο, με delimiter comma, και για κάθε στοιχείο της λίστας του skyline γράφουμε την γραμμή.

def getInput():

```
chosenCategories=[]
args=[]
checked=0

print('------------------------------NBA STATS 2017------------------------------')
print('------------------------------SKYLINE PLAYERS------------------------------\n')
print('Choose which of the following statCategories you are interested in: ')
print('1. Rebounds          2. Assists          3. Steals          4. Blocks          5. Points')

while checked==0:
        args=input('\nGive the numbers with comma in [ ]:\n')
        checked=checkArgs(args, chosenCategories)

return chosenCategories
```

Η συγκεκριμένη συνάρτηση παίρνει από τον χρήστη μέσω command line τις κατηγορίες μέσα σε [,] και καλεί την συνάρτηση *checkArgs() (όπως κάναμε και στο part1)* για να ελέγξει αν η είσοδος είναι ορθή.
Η μεταβλητή checked γίνεται 1 μόνο όταν έχει περαστεί και είναι ορθά τα δεδομένα από τον έλεγχο της *checkArgs()* .

def checkArgs(args, cat):

```
for i in args:

        if i.isdigit():

                if int(i)>=1 and int(i)<=5:
                        cat.insert(len(cat), int(i))
                else:
                        print('insert number from 1-5.\n')
                        return 0

        elif i=='[' or i==']' or i==',':
                pass
        elif i=='-':
                print('Insert number from 1-5.\n')
                return 0

        else:
```

```
                print('Insert number from 1-5.\n')
                return 0

        return 1
```

Ελέγχει αν τα δεδομένα μας είναι int θετικοί αριθμοί από 1 ως 5 και εάν είναι της μορφής:
<[> <num> <,> <num> <]>

Επιστρέφει 1 αν είναι εντάξει, αλλιώς 0.

def myBNL(cat):

```
    numOfChoices=len(cat)
    if numOfChoices<5:
        leftovers=5-numOfChoices
        for i in range(leftovers):
            cat.append(0)

     slHashMap={}
     firstTime=1
...
```

Για αρχή, βρίσκουμε τον πλήθος των κατηγοριών που έδωσε ο χρήστης και αν είναι λιγότερες από 5 συμπληρώνουμε τη λίστα cat με 0 (δηλαδή το cat μας θα έχει μετά από αυτή την επεξεργασία πάντα 5 στοιχεία πχ πριν: [1,4] → μετά: [1,4,0,0,0]).

Το hashMap του kyline μας το αρχικοποιούμε στο κενό.

Το flag firstTIme είναι 1 όταν μπαίνουμε για πρώτη φορά ώστε για αυτή και μόνο την φορά να εισάγουμε το πρώτο παίχτη στο skyline χωρίς να πρέπει να υπακούει κάποια συνθήκη και γίνεται 0 μετά και δεν ξαναλλάζει.

```
...
     with open(allStats, 'r',encoding='UTF-8') as df:

        row=df.readline()

        for row in df:

            data=row.split(',')
            data[-1] = data[-1].strip()
            iD=data[0]
            delList=[]
            needToInsert=0
            pl=0
...
```

Ανοίγουμε το allStats για διάβασμα και προσπερνάμε την πρώτη γραμμή με τα labels (*row=df.readline()*).

Για κάθε γραμμή στο αρχείο επεξεργαζόμαστε τώρα τα δεδομένα μας για να μπορέσουμε να τα χρησιμοποιήσουμε.

Πρώτα κάνουμε split όπου κόμμα, μετά αφαιρούμε απο το τελευταίο στοιχείο της λίστας *data[]* το \n και ορίζουμε ως iD το πρώτο στοιχείο της λίστας.

Ορίζουμε ως *delList[]* μια λίστα που θα συλλέγει τους παίχτες που μπήκαν στο skyline αλλά τελικά βρέθηκε ένας νέος παίχτης που τους κυριαρχεί, ώστε να γνωρίζουμε ποιους να διαγράψουμε.

Το flag needToInsert είναι 0 όταν δεν έχουμε βρεί νέο παίχτη που να ικανοποιεί την σχέση κυριαρχίας και 1 όταν την ικανοποιεί.

Το pl είναι το πλήθος των παιχτών (players) που έχουν τουλάχιστον ένα χαρακτηριστικό μεγαλύτερο από τα όσα υπάρχουν μέσα στην slHashMap.

```
...
            if numOfChoices==1:

                if firstTime==1:
                    slHashMap.update({iD : [data[1], int(data[cat[0]+2])]})
                    firstTime=0

                for idd in slHashMap:
                    val1=slHashMap.get(idd)
                    val1=val1[1]
```

```
                    if int(data[cat[0]+2])>val1:
                        delList.insert(len(delList), idd)
                        needToInsert=1

            for idss in delList:
                del slHashMap[idss]

            if needToInsert==1:
                slHashMap.update({iD : [data[1], int(data[cat[0]+2])]})
                needToInsert=0
```

...

Αν το πλήθος των κατηγοριών είναι 1 τότε αν μπαίνουμε για πρώτη φορά εισάγουμε τον πρώτο παίχτη που βρίσκουμε στο αρχείο και κάνουμε το firstTIme 0.

Για κάθε idd στο slHashMap βάζουμε στο val1 το value που βρίσκουμε στο slHashMap από το συγκεκριμένο idd ως κλειδί.

Το cat[0]+2 είναι έτσι γιατί πρέπει σε κάθε περίπτωση να προσπερνάμε τις κολώνες με τα ονόματα και τις ομάδες.

Η συνθήκη κυριαρχίας λέει ότι εάν η απόδοση του συγκεκριμένου παίχτη είναι μεγαλύτερη από εκείνη του παίχτη που τσεκάρουμε τώρα στη slHashMap τότε βάζουμε το idd του παίχτη της hashMap που βρήκαμε να είναι χειρότερος στην delList και κάνουμε την μεταβλητή needToInsert 1.

Σβήνουμε στην συνέχεια αφού έχουμε σκανάρει όλο το slHashMap τους παίχτες που ήταν χειρότεροι από το slHashMap και εισάγουμε στο slHashMap τον παίχτη που βρίκαμε να τους κυριαρχεί.

...

```
        elif numOfChoices==2:

            if firstTime==1:
                slHashMap.update({iD : [data[1], int(data[cat[0]+2]), int(data[cat[1]+2])]})
                firstTime=0

            for idd in slHashMap:
                val1=slHashMap.get(idd)
                val2=slHashMap.get(idd)
                val1=val1[1]
                val2=val2[2]

                if (int(data[cat[0]+2])>val1 and int(data[cat[1]+2])>=val2) or
                   (int(data[cat[1]+2])>val2 and int(data[cat[0]+2])>=val1):

                    delList.insert(len(delList), idd)
                    needToInsert=1

                elif int(data[cat[0]+2])>val1 or int(data[cat[1]+2])>val2 :
                    pl+=1

            for idss in delList:
                del slHashMap[idss]

            if len(slHashMap)==pl:
                needToInsert=1

            if needToInsert==1:
                slHashMap.update({iD : [data[1], int(data[cat[0]+2]), int(data[cat[1]+2])]})
                needToInsert=0
```

...

Οι υπόλοιπες περιπτώσεις για numOfChoices 2,3,4 και 5 έχουν την ίδια λογική.

Έστω numOfChoices =2.

Όπως και πριν, αν μπαίνουμε για πρώτη φορά εισάγουμε τον πρώτο παίχτη που βρίσκουμε στο αρχείο και κάνουμε το firstTIme 0.

Για κάθε idd στο slHashMap βάζουμε στο val1 το value της μιας κατηγορίας που βρίσκουμε στο slHashMap από το συγκεκριμένο idd ως κλειδί και στο val2 αντίστοιχα το value που βρίσκουμε στο slHashMap της άλλης επιλεγμένης κατηγορίας κ.ο.κ για περισσότερα από numOfChoices=2.

Το cat[0]+2 είναι έτσι γιατί πρέπει σε κάθε περίπτωση να προσπερνάμε τις κολώνες με τα ονόματα και τις ομάδες.

ASSIGNMENT 2 Complex Data Management on Top-K & Skyline Queries

Η συνθήκη κυριαρχίας λέει ότι εάν η απόδοση του συγκεκριμένου παίχτη σε ένα (τουλάχιστον) από τα δυο χαρακτηριστικά είναι αυστηρά μεγαλύτερη από εκείνη του παίχτη που τσεκάρουμε τώρα στη slHashMap και η απόδοση στο άλλο χαρακτηριστικό (στα άλλα χαρακτηριστικά αν έχουμε numOfChoices 3 και πάνω) είναι μεγαλύτερη ή ίση τότε βάζουμε το idd του παίχτη της hashMap που βρήκαμε να είναι χειρότερος στην delList και κάνουμε την μεταβλητή needToInsert 1.

Με λίγα λόγια ο παίχτης για να κυριαρχεί ώστε να μπει στο skyline πρέπει να είναι καλύτερος τουλάχιστον σε ένα χαρακτηριστικό και να μην είναι χειρότερος σε κανένα από τα άλλα.

Εάν δεν εκπληρώνεται η παραπάνω συνθήκη, ελέγχουμε αν τουλάχιστον ο παίχτης έχει κάποια από όλες τις αποδόσεις του μεγαλύτερη από αυτή των υπολοίπων στο συγκεκριμένο χαρακτηριστικό,
Αν ναι, αυξάνουμε τη μεταβλήτη pl κατά 1.

Σβήνουμε στην συνέχεια, αφού έχουμε σκανάρει όλο το slHashMap, τους παίχτες που ήταν χειρότεροι από το slHashMap και εισάγουμε στο slHashMap τον παίχτη που βρίκαμε να τους κυριαρχεί.

Εδώ αξίζει να σημειωθεί ότι αν το μήκος του hashMap είναι ίσο με το pl (δηλαδή για όλους τους παίχτες μέσα στο slHashMap έχει τουλάχιστον ένα χαρακτηριστικό καλύτερο ο παίχτης μας) πάλι χρειάζεται να κάνουμε insert τον παίχτη γιατί δεν κυριαρχείται από όσους υπάρχουν μέσα ως τώρα. **δεν κυριαρχείται αν το καθένα υπερτερεί σε διαφορετικό χαρακτηριστικό.**

```
elif numOfChoices==3:

        if firstTime==1:
                slHashMap.update({iD : [data[1], int(data[cat[0]+2]), int(data[cat[1]+2]),
                                                                        int(data[cat[2]+2])]})
                firstTime=0

        for idd in slHashMap:

                val1=slHashMap.get(idd)
                val2=slHashMap.get(idd)
                val3=slHashMap.get(idd)
                val1=val1[1]
                val2=val2[2]
                val3=val3[3]

                if (int(data[cat[0]+2])>val1 and int(data[cat[1]+2])>=val2 and
                   int(data[cat[2]+2])>=val3) or (int(data[cat[1]+2])>val2 and
                    int(data[cat[0]+2])>=val1 and int(data[cat[2]+2])>=val3) or
                    (int(data[cat[2]+2])>val3 and int(data[cat[0]+2])>=val1 and
                                                int(data[cat[1]+2])>=val2):

                        delList.insert(len(delList), idd)
                        needToInsert=1

                elif int(data[cat[0]+2])>val1 or int(data[cat[1]+2])>val2 or
                                                int(data[cat[2]+2])>val3:
                        pl+=1


        for idss in delList:
                del slHashMap[idss]

        if len(slHashMap)==pl:
                needToInsert=1

        if needToInsert==1:
                slHashMap.update({iD : [data[1], int(data[cat[0]+2]), int(data[cat[1]+2]),
                                                                        int(data[cat[2]+2])]})
                needToInsert=0



elif numOfChoices==4:

        if firstTime==1:
                slHashMap.update({iD : [data[1], int(data[cat[0]+2]), int(data[cat[1]+2]),
                                                        int(data[cat[2]+2]), int(data[cat[3]+2])]})
                firstTime=0

        for idd in slHashMap:
                val1=slHashMap.get(idd)
```

```
                    val2=slHashMap.get(idd)
                    val3=slHashMap.get(idd)
                    val4=slHashMap.get(idd)
                    val1=val1[1]
                    val2=val2[2]
                    val3=val3[3]
                    val4=val4[4]

                    if (int(data[cat[0]+2])>val1 and int(data[cat[1]+2])>=val2 and
                       int(data[cat[2]+2])>=val3 and int(data[cat[3]+2])>=val4) or
                       (int(data[cat[1]+2])>val2 and int(data[cat[0]+2])>=val1 and
                       int(data[cat[2]+2])>=val3 and int(data[cat[3]+2])>=val4) or
                      (int(data[cat[2]+2])>val3 and int(data[cat[0]+2])>=val1 and
                       int(data[cat[1]+2])>=val2 and int(data[cat[3]+2])>=val4) or
                       (int(data[cat[3]+2])>val4 and int(data[cat[0]+2])>=val1 and
                       int(data[cat[1]+2])>=val2 and int(data[cat[2]+2])>=val3):

                            delList.insert(len(delList), idd)
                            needToInsert=1

                    elif int(data[cat[0]+2])>val1 or int(data[cat[1]+2])>val2 or
                         int(data[cat[2]+2])>val3 or int(data[cat[3]+2])>val4:

                            pl+=1


            for idss in delList:
                    del slHashMap[idss]

            if len(slHashMap)==pl:
                    needToInsert=1

            if needToInsert==1:
                    slHashMap.update({iD : [data[1], int(data[cat[0]+2]), int(data[cat[1]+2]),
                                                        int(data[cat[2]+2]), int(data[cat[3]+2])]})
                    needToInsert=0



    elif numOfChoices==5:

            if firstTime==1:
                    slHashMap.update({iD : [data[1], int(data[cat[0]+2]), int(data[cat[1]+2]),
                                    int(data[cat[2]+2]), int(data[cat[3]+2]), int(data[cat[4]+2])]})


                    firstTime=0

            for idd in slHashMap:
                    val1=slHashMap.get(idd)
                    val2=slHashMap.get(idd)
                    val3=slHashMap.get(idd)
                    val4=slHashMap.get(idd)
                    val5=slHashMap.get(idd)
                    val1=val1[1]
                    val2=val2[2]
                    val3=val3[3]
                    val4=val4[4]
                    val5=val5[5]

                    if (int(data[cat[0]+2])>val1 and int(data[cat[1]+2])>=val2 and
                       int(data[cat[2]+2])>=val3 and int(data[cat[3]+2])>=val4 and
                       int(data[cat[4]+2])>=val5) or (int(data[cat[1]+2])>val2 and
                       int(data[cat[0]+2])>=val1 and int(data[cat[2]+2])>=val3 and
                       int(data[cat[3]+2])>=val4 and int(data[cat[4]+2])>=val5) or
                       (int(data[cat[2]+2])>val3 and int(data[cat[0]+2])>=val1 and
                    int(data[cat[1]+2])>=val2 and int(data[cat[3]+2])>=val4 and
                    int(data[cat[4]+2])>=val5) or (int(data[cat[3]+2])>val4 and
                    int(data[cat[0]+2])>=val1 and int(data[cat[1]+2])>=val2 and
                    int(data[cat[2]+2])>=val3 and int(data[cat[4]+2])>=val5) or
                    (int(data[cat[4]+2])>val5 and int(data[cat[0]+2])>=val1 and
                    int(data[cat[1]+2])>=val2 and int(data[cat[2]+2])>=val3 and
                    int(data[cat[3]+2])>=val4):
```

```
                                        delList.insert(len(delList), idd)
                                needToInsert=1

                        elif int(data[cat[0]+2])>val1 or int(data[cat[1]+2])>val2 or
                                int(data[cat[2]+2])>val3 or int(data[cat[3]+2])>val4 or
                                int(data[cat[4]+2])>val5:

                                pl+=1


                for idss in delList:
                        del slHashMap[idss]

                if len(slHashMap)==pl:
                        needToInsert=1

                if needToInsert==1:
                        slHashMap.update({iD : [data[1], int(data[cat[0]+2]), int(data[cat[1]+2]),
                                        int(data[cat[2]+2]), int(data[cat[3]+2]), int(data[cat[4]+2])]})

                        needToInsert=0
...
```

Ακολουθείται η ίδια διαδικασία με πριν.

...

```
        sk=list(slHashMap.values())
        return sk
```

Τέλος σε όποια περίπτωση και αν έχουμε μπει, βάζουμε τα αποτελέσματα που συλλέξαμε στο slHashMap σε μια λιστα *sk[]* και την επιστρέφουμε στην *main().*

Έχουμε βρει το skyline.