Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών Και Μηχανικών Υπολογιστών

Προχωρημένα Θέματα Βάσεων Δεδομένων

9ο Εξάμηνο

Εξαμηνιαία Εργασία

Καναβάκης Ελευθέριος 03114180
Δαζέα Ελένη 03114060

Σύντομη εισαγωγή : Στην εργασία αυτή θα χρησιμοποιήσουμε αλγορίθμους MapReduce για να κάνουμε αποθήκευση και ανάλυση σε έναν αριθμό από σετ δεδομένων . Για την υλοποίηση της εργασίας αυτής έχουμε στην διάθεση μας ένα cluster στον ωκεανό το οποίο αποτελείται από 2 υπολογιστές καθένας από τους οποίους διαθέτει 2 πυρήνες cpu , 2gb Ram και 30gb μνήμη . Επιπρόσθετα , για την υλοποίηση της εργασίας αυτής θα χρησιμοποιήσουμε τόσο το Hadoop distributed filesystem όσο και το Spark RDD API. Πιο συγκεκριμένα θα χρησιμοποιήσουμε το hdfs για να αποθηκεύσουμε το input και output του κώδικα μας σε ένα distributed σύστημα , ενώ θα χρησιμοποιήσουμε το Spark RDD API για να προγραμματίσουμε τα MapReduce που θα εκτελεί ο κώδικας μας πάνω στα δεδομένα εισόδου. Τέλος , η γλώσσα προγραμματισμού που επιλέξαμε για να γράψουμε τον κώδικα μας είναι η γλώσσα Python . Ο λόγος που επιλέξαμε την γλώσσα αυτή δεν είναι τυχαίος . Παρόλο που η Scala (εναλλακτική επιλογή γλώσσας για προγραμματισμό στο Spark Api) είναι καλύτερα optimized επιλέξαμε την Python καθώς στην εργασία μας θα κληθούμε να πραγματοποιήσουμε πράξεις πινάκων και αλγορίθμους μηχανικής μάθησης , τομείς στους οποίους η Python διαθέτει πολλές βιβλιοθήκες με συναρτήσεις οι οποίες μπορούν να φανούν ιδιαίτερα χρήσιμες κατά την ανάπτυξη του κώδικας μας !

Άσκηση 1 – Αναλυτική επεξεργασία δεδομένων

Στην άσκηση αυτή θα μελετήσουμε και θα επεξεργαστούμε τα δεδομένα από τις διαδρομές των ταξί της Νέα Υόρκης για τον μήνα Μάρτιο . Πιο συγκεκριμένα μας δίνεται ένα dataset μεγέθους περίπου 2GB το οποίο αποτελείται από 13 εκατομμύρια διαδρομές . Το dataset αυτό , αποτελείται από δύο αρχεία . Το αρχείο yellow_tripvendors_1m.csv και το αρχεία yellow_tripdata_1m.csv . Παρατηρούμε ότι τα αρχεία αυτά είναι .csv , δηλαδή πρόκειται για comma separated values . Το γεγονός αυτό θα το αξιοποιήσουμε κατάλληλα στην επεξεργασία του dataset μας κατά την εισαγωγή του.

Ερώτημα α): Αρχικά μας ζητείται να βρούμε την μέση διάρκεια διαδρομής (σε λεπτά ανά ώρα). Για το λόγο αυτό αρχικά διαβάζουμε το αρχείο yellow_tripdata_1m.csv και το κάνουμε split σε 100 αρχεία για να είναι καλύτερα distributed στην μνήμη και να μην έχουμε προβλήματα τύπου OOM . Στην συνέχεια , κάνουμε split την κάθε γραμμή του πίνακα χρησιμοποιώντας σαν στοιχείο διαχωρισμού το κόμμα (‘,’) και κρατάμε το πρώτο και δεύτερο στοιχείο της κάθε γραμμής στα οποία περιέχονται τα δεδομένα που χρειαζόμαστε (ημερομηνία και ώρα έναρξης και λήξης διαδρομής). Τέλος , ολοκληρώνουμε το preprocessing του dataset μας κρατώντας σαν key μόνο

την ώρα , αγνοώντας τα λεπτά και δευτερόλεπτα (πχ 23) και σαν value την χρονική διάρκεια διαδρομής σε λεπτά . Για τον υπολογισμό της χρονικής διάρκειας διαδρομής σε λεπτά αρχικά μετατρέψαμε την ημερομηνία και ώρα σε ένα αντικείμενο της κλάσης datetime και στην συνέχεια αφαιρέσαμε από το αντικείμενο που αντιστοιχεί στην αποβίβαση το αντικείμενο που αντιστοιχεί στην επιβίβαση. Τέλος, μετατρέψαμε το τελικό αντικείμενο αυτό σε δευτερόλεπτα και τα δευτερόλεπτα σε λεπτά .

Στην συνέχεια παραθέτουμε σε ψευδοκώδικα map reduce την παραπάνω διαδικασία :

```
map(key,value):
    String record = value.toString()
    String parts = record.split(",")
    int hourofday = gethour(parts[1])
    float duration = duration(parts[1],parts[2])
    emit(hourdofday,duration)
```

Αφού ολοκληρώσαμε την προεπεξεργασία του dataset μας θα προχωρήσουμε στην βασική υλοποίηση του κώδικα μας κατά την οποία ουσιαστικά αθροίζουμε για όλα τα στοιχεία με το ίδιο κλειδί την χρονική διάρκεια διαδρομή και στην συνέχεια για κάθε κλειδί διαιρούμε με το πλήθος των διαδρομών για την ώρα αυτή . Τα παραπάνω συνοψίζονται με την εξής γραμμή κώδικα σε python :

```
dataset = dataset.mapValues(lambda v: (v,1))
.reduceByKey(lambda a,b: (a[0]+b[0],a[1]+b[1]))
.mapValues(lambda v: v[0]/v[1])
```

Στην συνέχεια , θα παραθέσουμε ψευδοκώδικα που περιγράφει αναλυτικά τι κάνουν οι παραπάνω map και reduce .

Αρχικά , μετατρέπουμε τα values στην μορφή (value,1) έτσι ώστε να μπορέσουμε να μετρήσουμε το πλήθος των διαδρομών ανά ώρα .

```
map(key,value):
    d = value
    emit(key,(d,1))
```

Στην συνέχεια προσθέτουμε για κάθε κλειδί προσθέτουμε όλα τα values . Πιο συγκεκριμένα , επειδή τα values είναι σε μορφή τούπλας προσθέτουμε όλα τα πρώτα στοιχεία της τούπλας μεταξύ τους και αντίστοιχα τα δεύτερα .

```
reduce(key,values):
    result = 0
    counter = 0
    for each v in values:
        result += v[0]
        counter += v[1]
    emit(key,(result,counter))
```

Τέλος , αυτή την στιγμή τα values μας είναι στην μορφή (sum of minutes,number of taxi rides) οπότε για να υπολογίσουμε τον μέσο όρο αρκεί να διαιρέσουμε το πρώτο στοιχείο κάθε τούπλας με το δεύτερο στοιχείο της τούπλας αυτής .

```
map(key,value):
    v = value
    emit(key,v[0]/v[1])
```

Στο σημείο αυτό υπολογίσαμε το ζητούμενου του ερωτήματος αυτού και επομένων αποθηκεύουμε το αποτέλεσμα μας σε ένα αρχείο στο hdfs distributed file system .

Παράλληλα ταξινομήσαμε τα αποτελέσματα μας και τα τυπώσαμε στην οθόνη . Για την ταξινόμηση χρησιμοποιήσαμε τις εντολές .sortByKey().collect() . Ο ψευδοκώδικας για τις εντολές αυτές φαίνεται παρακάτω .

```
map(key,value):
    emit(1,(key,value))      // τα βάζουμε όλα στο ίδιο bucket
```

```
reduce(key,values):
    mergersort(values)
    foreach value in values:
        emit(value[0],value[1])
```

Στην συνέχεια , παραθέτουμε τα αποτελέσματα που εξάγαμε από την εκτέλεση του κώδικα μας :

```
HourOfDay      AverageTripDuration

(0,            14.017793737362238)
(1,            13.975069898907131)
(2,            13.035635592676696)
(3,            13.322282520526894)
(4,            13.799857931121982)
(5,            13.275583221175415)
(6,            12.487420237563242)
(7,            13.395006418527391)
(8,            14.627504543367815)
(9,            14.670106419765618)
(10,           14.657939169698290)
(11,           14.935821221905542)
(12,           15.130881322885687)
(13,           15.553918733195127)
(14,           16.523138380789470)
(15,           30.223498632126258)
(16,           17.213072069374658)
(17,           16.510825654408613)
(18,           15.290453748601190)
(19,           14.221208805985722)
(20,           13.575899636364245)
(21,           13.510855327392878)
(22,           14.231797625637371)
(23,           13.958471125232721)
```

Ερώτημα β): Στο ερώτημα αυτό μας ζητείται να υπολογίσουμε το μέγιστο ποσό που πληρώθηκε σε μία διαδρομή για κάθε εταιρεία ταξί (vendor) . Για τον υπολογισμό αυτό θα χρειαστούμε και τα δύο αρχεία που μας δίνονται σαν είσοδο . Αφού τα διαβάσουμε κατάλληλα (παρόμοια λογική που περιγράφηκε παραπάνω) έχουμε δυο RDD . Το πρώτο είναι της μορφής (trip_id,vendor_id) και το δεύτερο είναι της μορφής (trip_id,money_paid) . Ο ψευδοκώδικας για το διάβασμα αυτό φαίνεται παρακάτω :

```
map(key,value):
    // file tripdata
    String record = value.toString()
    String parts = record.split(",")
    emit(parts[0],parts[7])

map(key,value):
    // file tripvendors
    String record = value.toString()
    String parts = record.split(",")
    emit(parts[0],parts[1])
```

Επιθυμούμε, να ενώσουμε τα παραπάνω αρχεία. Ως εκ τούτου , θα πραγματοποιήσουμε inner join (reduce side join) για τα δύο RDD's .Έτσι , το καινούργιο RDD που δημιουργείται είναι της μορφής (trip_id,(money_paid,vendor_id)) .

```
map(key,value):
    // file tripdata
    emit(key,("data",value))

map(key,value):
    // file tripvendors
    emit(key,("vendors",value))

reduce(key,values):

    int vendorid ;
    foreach value in values:
        if(value[0]=="vendors") {
            vendorid = int(value[1])
        }
    foreach value in values:
        if(value[0]=="data") {
            emit(key,(float(value[1]),vendorid))
        }
```

Εμείς επιθυμούμε να το μετατρέψουμε σε ένα RDD της μορφής (vendor_id,money_paid). Για τον σκοπό αυτό χρησιμοποιήσαμε μια απλή συνάρτηση map ο ψευδοκώδικας της οποίας παρατίθεται παρακάτω :

```
map(key,value):
    v = value
    emit(v[1],v[0])
```

Στην συνέχεια , αρκεί να επιλέξουμε το μέγιστο ποσό ανά κλειδί με την χρήση της παρακάτω συνάρτησης :

```
reduce(key,values):
    max = -1
    for each v in values:
        if(v > max):
            max = v
    emit(key,max)
```

Στο σημείο αυτό υπολογίσαμε το ζητούμενου του ερωτήματος αυτού και όπως και προηγουμένως αποθηκεύσαμε το αποτέλεσμα μας σε ένα αρχείο στο hdfs distributed file system .

Στην συνέχεια , παραθέτουμε τα αποτελέσματα που εξάγαμε από την εκτέλεση του κώδικα μας :

```
VendorID        MaxAmountPaid
1               503326.33
2               548463.35
```

Παρατηρούμε ότι τα ποσά που εξάγαμε είναι αστρονομικό και κατά πάσα πιθανότητα λάθος . Αρχικά , ελέγξαμε το ενδεχόμενο του να έχουμε κάνει κάποιο λάθος . Αναζητήσαμε λοιπόν τις τιμές αυτές στο αρχείο εισόδου και επιβεβαιώσαμε ότι ο κώδικας μας είναι σωστός καθώς τα ποσά αυτά όντως υπάρχουν στο dataset μας . Στο σημείο αυτό προβληματιστήκαμε αρκετά σχετικά με το αν πρέπει να κάνουμε filtering στο dataset μας ή όχι . Τελικά , επιλέξαμε να μην κάνουμε filtering για δύο λόγους . Ο πρώτος αφορά το ότι μας ζητείται το μέγιστο ποσό χωρίς να αναφέρεται κάτι για filtering . Έτσι, το σωστό αποτέλεσμα είναι αυτό που παραθέσαμε . Επιπρόσθετα , επισκεφθήκαμε την επίσημη ιστοσελίδα του dataset (https://data.cityofnewyork.us/Transportation/2015-Yellow-Taxi-Trip-Data) στην οποία και είδαμε ότι στα μέγιστα ποσά υπάρχουν τα ποσά που υπολογίσαμε . Τα ποσά φαίνονται στο επόμενο screenshot :

total_amount	vendor_id	pickup_datetime	dropoff_datetime
3,950,611.6	2	2015 Jan 18 07:24:15 PM	2015 Jan 18 07:51:55 PM
826,039.97	1	2015 Dec 27 03:45:54 AM	2015 Dec 27 03:45:54 AM
826,039.97	1	2015 Dec 27 03:44:25 AM	2015 Dec 27 03:45:53 AM
650,262.85	2	2015 Apr 12 10:03:19 AM	2015 Apr 12 10:10:27 AM
548,463.35	2	2015 Mar 01 06:18:48 PM	2015 Mar 01 06:27:06 PM
503,326.33	1	2015 Mar 22 01:08:58 PM	2015 Mar 22 01:19:08 PM
410,267.66	1	2015 Sep 28 11:32:21 AM	2015 Sep 28 11:32:21 AM
335,414.49	1	2015 Jun 26 06:47:32 AM	2015 Jun 26 06:50:36 AM
322,743.82	1	2015 Feb 20 11:04:39 PM	2015 Feb 20 11:38:56 PM
210,523.79	1	2015 Apr 13 07:53:47 AM	2015 Apr 13 08:05:12 AM
187,444.26	1	2015 Oct 24 01:12:44 PM	2015 Oct 24 01:25:50 PM

Φυσικά παρατηρώντας την χρονική διάρκεια της διαδρομής επιβεβαιώνουμε ότι τα ποσά αυτά είναι προφανώς λάθος . Θα μπορούσαμε να αποφύγουμε το πρόβλημα αυτό γράφοντας την παρακάτω γραμμή κώδικα :

```
dataset = dataset.filter(lambda x: x[1]<5000 )
```

Με την παραπάνω γραμμή θα κρατάγαμε μόνο τα ποσά που ήταν μικρότερα από 5000 δολάρια .

Ενδεικτικά θα παραθέσουμε και τα αποτελέσματα εκτέλεσης μετά το παραπάνω filtering .

```
VendorID        MaxAmountPaid
1               4864.51
2               3336.8
```

Άσκηση 2 – Machine Learning , εκτέλεση kmeans με fixed k

Στην άσκηση αυτή θα χρησιμοποιήσουμε τα δεδομένα που είχαμε και στην άσκηση 1 για να εκτελέσουμε έναν αλγόριθμο μηχανικής μάθησης . Πιο συγκεκριμένα θα εκτελέσουμε τον αλγόριθμο ομαδοποίησης kmeans . Με τον αλγόριθμο αυτό θα ομαδοποιήσουμε τα σημεία επιβίβασης των πελατών σε k=5 περιοχές (clusters) και θα βρούμε το κέντρο των σημείων της κάθε περιοχής. Αρχικά διαβάσαμε και πάλι το dataset μας και κρατήσαμε τις συντεταγμένες x,y. Στην συνέχεια κάναμε filtering στο dataset μας καθώς όπως είδαμε υπήρχαν missing values στις συντεταγμένες x,y . Πιο συγκεκριμένα υπήρχαν συντεταγμένες x,y οι οποίες είχαν τιμή ίση με 0 . Φυσικά οι συντεταγμένες αυτές ήταν λάθος καθώς μελετάμε τις διαδρομές των ταξί στην Νέα Υόρκη και οι συντεταγμένες αυτές αντιστοιχούσαν κάπου βαθιά στην θάλασσα . Στην συνέχεια , παρατίθεται ψευδοκώδικας για τα παρακάτω .

```
map(key,value):
    String record = value.toString()
    String parts = record.split(",")
    float coordsx,coordsy
    coordsx = float(parts[3])
    coordsy = float(parts[4])
    if(coordsx!=0 && coordsy!=0) {
        emit(coordsx,coordsy)
    }
```

Αφού λοιπόν διαβάσαμε και φιλτράραμε το dataset μας για missing values εκτελέσαμε επαναληπτικά τον αλγόριθμο μας (για αριθμό επαναλήψεων ίσο με 3). Η επαναληπτική εκτέλεση αυτή αναλύεται παρακάτω :

Αρχικά , για κάθε σημείο x,y υπολογίσαμε το τετράγωνο της ευκλείδειας απόστασης(μας ενδιαφέρει να συγκρίνουμε τις αποστάσεις για αυτό αρκεστήκαμε στο τετράγωνο της ευκλείδειας απόστασης) του από όλα τα κέντρα . Στην συνέχεια , βρήκαμε το πιο κοντινό σε αυτό κέντρο και επιστρέψαμε το index του .

```
map(key,value):
    index = 0
    mindst = inf
    tempdst = 0
    for i in range(len(centroid)):
        tempdst =(value[0]-centroid[i][0])**2+(value[1]-centroid[i][1])**2
        if(mindst>tempdst):
            mindst = tempdst
            index = i
    emit(index,(value,1))
```

Μετά την εκτέλεση της παραπάνω map το RDD που δημιουργείτε είναι της μορφής (index of closest centroid,(points,1)) . Στην συνέχεια επιθυμούμε να επαναυπολογίσουμε τα κέντρα . Ως εκ τούτου αρχικά θα υπολογίσουμε το άθροισμα των σημείων και το πλήθος τους ανάλογα με το index .

```
reduce(key,values):
    result_x = 0
    result_y = 0
    counter = 0
    for each tuple v in values:
        result_x +=v[0][0]
        result_y +=v[0][1]
        counter +=v[1]
    emit(key,((result_x,result_y),counter))
```

Τέλος , για τον υπολογισμών των κέντρων αρκεί να διαιρέσουμε το άθροισμα των σημείων με το πλήθος τους .

```
map(key,value):
    total = value[1]
    sum_x = value[0][0]
    sum_y = value[0][1]
    emit(key,(sum_x/total,sum_y/total))
```

Τέλος , αφού εκτελέσουμε το παραπάνω κομμάτι κώδικα 3 φορές τυπώνουμε τα κέντρα που υπολογίσαμε και τα αποθηκεύουμε σε ένα αρχείο στο hdfs distributed file system .

Στην συνέχεια , παραθέτουμε τα αποτελέσματα που εξάγαμε από την εκτέλεση του κώδικα μας :

```
Id   Centroid
1    [-78.55245188347199 , 40.57994150215725]
2    [-73.83948279658780 , 40.71842786195664]
3    [-73.99643817071644 , 40.71580962445436]
4    [-73.99104123343132 , 40.74429505585700]
5    [-73.96875877280982 , 40.77102816561475]
```

Άσκηση 3 – Γράφοι , PageRank computation

Στην άσκηση αυτή θα υλοποιήσουμε τον αλγόριθμο του PageRank σε map-reduce context . Ο αλγόριθμος του PageRank είναι ο πρώτος και πιο γνωστός αλγόριθμος ταξινόμησης ιστοσελίδων . Πιο συγκεκριμένα ο αλγόριθμος αυτός χρησιμοποιήθηκε από την Google για την ταξινόμηση των ιστοσελίδων στα αποτελέσματα αναζήτησης .

Ο αλγόριθμος του PageRank μετράει το πόσο σημαντική είναι μια ιστοσελίδα με βάση τα inbound links , δηλαδή τις ιστοσελίδες που δείχνουν σε αυτή . Στον αλγόριθμο αυτό το διαδίκτυο μοντελοποιείται σαν ένας γράφος , στον οποίο οι κόμβοι αναπαριστούν τις ιστοσελίδες ενώ η κάθε ακμή «σημαίνει» ότι μια

ιστοσελίδα δείχνει σε μία άλλη μέσα από κάποιο hyperlink . Όπως είπαμε και προηγουμένως , αλγόριθμος του PageRank υπολογίζει το ποσό σημαντική είναι μια ιστοσελίδα . Πιο συγκεκριμένα , η σημασία μιας ιστοσελίδας μπορεί να μεταφραστεί σαν την πιθανότητα να επισκεφτεί ένας χρήστης μια ιστοσελίδα κάνοντας τυχαία clicks . Η πιθανότητα αυτή υπολογίζεται από τον παρακάτω τύπο :

$$PR(p_j) = \frac{1-d}{N} + d \times \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)}$$

Στην άσκηση αυτή ουσιαστικά καλούμαστε να υλοποιήσουμε τον παραπάνω τύπο κάνοντας χρήση του προγραμματιστικού μοντέλου MapReduce . Ως εκ τούτου , θα αναλύσουμε σταδιακά την παραπάνω μαθηματική σχέση .Πριν όμως προχωρήσουμε στην ανάλυση αυτή θα ασχοληθούμε λίγο με τα δεδομένα εισόδου μας . Τα δεδομένα εισόδου μας δίνονται στο αρχείο web-Google.txt , στο οποίο περιέχονται μοναδικές ακμές ενός κατευθυνόμενου γράφου ! Έτσι , γνωρίζουμε ότι κάθε ακμή που μας δίνεται δεν είναι bidirectional και είναι μοναδική άρα δεν χρειάζεται να κάνουμε καμία προεπεξεργασία του input μας. Έτσι μπορούμε να συνεχίσουμε με την ανάλυση των υπολογισμών του αλγορίθμου μας .Αρχικά , γνωρίζουμε ήδη από την εκφώνηση της άσκησης ότι η μεταβλητή d (συντελεστής απόσβεσης) είναι μία σταθερά με τιμή ίση με 0.85 . Έτσι , εύκολα καταλαβαίνει κανείς ότι η συγκεκριμένη σταθερά δεν ιδιαίτερα την υλοποίηση μας . Στην συνέχεια , παρατηρούμε ότι και η μεταβλητή N μπορεί να θεωρηθεί ως σταθερά καθώς η μεταβλητή αυτή αποτελεί το συνολικό πλήθος όλων των ιστοσελίδων . Την τιμή για την μεταβλητή αυτή υπάρχουν δύο τρόποι για να την υπολογίσουμε . Είτε να μετρήσουμε μία φορά τους κόμβους κατά την διάρκεια εκτέλεσης του προγράμματος μας (πχ nodes.count() , αφού έχει προηγηθεί groupByKey() φυσικά) είτε να πάρουμε απ' ευθείας την τιμή αυτή από την εκφώνηση της άσκησης ή τις πρώτες γραμμές του dataset μας . Εμείς επιλέξαμε να μετράμε στον κώδικα μας τον αριθμό των ιστοσελίδων έτσι ώστε ο κώδικας μας να δουλεύει για περισσότερα από ένα dataset . Εφόσον λοιπόν , οι μεταβλητές d και N είναι σταθερές (ή σχεδόν σταθερές στην περίπτωση της N) συμπεραίνουμε ότι όλος ο υπολογισμός μας συμβαίνει κατά το άθροισμα $\sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)}$. Στο άθροισμα αυτό υπολογίζουμε το πηλίκο του score μίας ιστοσελίδας δια το πλήθος των ιστοσελίδων στην οποία δείχνει αυτή η ιστοσελίδα , για κάθε ιστοσελίδα που δείχνει στην ιστοσελίδα της οποίας υπολογίζουμε το score. Ως εκ τούτου , συνειδητοποιούμε ότι μιας ενδιαφέρει σε κάθε iteration του αλγορίθμου να γνωρίζουμε σε ποιες ιστοσελίδες δείχνει μία ιστοσελίδα και ποιο είναι το PageRank score της ιστοσελίδας αυτής. Εφόσον γνωρίζουμε τις πληροφορίες αυτές η άσκηση μας έχει λυθεί καθώς

για κάθε ιστοσελίδα αρκεί να υπολογίσουμε την συνεισφορά της σε κάθε ιστοσελίδα που δείχνει και στην συνέχεια τις συνεισφορές αυτές να τις αθροίσουμε για κάθε ιστοσελίδα . Στην συνέχεια παραθέτουμε τον ψευδοκώδικα της υλοποίησης μας έτσι ώστε η παραπάνω περιγραφή να γίνει καλύτερα κατανοητή .

Αρχικά , θα διαβάσουμε το αρχείο εισόδου μας κάνοντας και το απαραίτητο preprocessing στα δεδομένα μας . Πιο συγκεκριμένα χρειάστηκε να κόψουμε τις γραμμές που ξεκινούν με # . Ο ψευδοκώδικας της υλοποίησης μας φαίνεται παρακάτω :

```
map(key,value):
    String record = value.toString()
    if(!(record.startswith("#"))) {
        String parts = record.split("\t")
        emit(int(parts[0]),int(parts[1]))
    }
```

Αφού διαβάσαμε το αρχείο εισόδου μας , δημιουργήσαμε από αυτό δύο rdd's .Ένα rdd που περιλαμβάνει τις ακμές και είναι της μορφής (pageid,list of outbound page id's) και ένα rdd που περιλαμβάνει το rank της κάθε ιστοσελίδας και είναι της μορφής (pageid,rank).

```
// (page_id,list_of_outbounds)
map(key,value):
    emit(key,value)

reduce(key,values):
    emitA(key,values)

// inital pagerank
map(key,value):
    emit(key,0.5)
```

Στην συνέχεια θεωρούμε ότι ήδη έχουμε υπολογίσει τον όρο $\frac{1-d}{N}$ καθώς όπως αναλύσαμε και παραπάνω πρόκειται για έναν σταθερό όρο. Σε κάθε iteration του αλγορίθμου μας τώρα αρχικά εκτελούμε ένα join μεταξύ των rdd's edges,rank έτσι ώστε να δημιουργήσουμε ένα νέο rdd της μορφής (pageid,(list of outbound pageids,rank)) και στην συνέχεια εκτελούμε ένα flatMap για να δημιουργήσουμε ένα νέο rdd στο οποίο για κάθε pageid που ανήκει στην λίστα με τα outbound pageids έχουμε υπολογίσουμε το rank contribution της σελίδας που δείχνει σε αυτό. Ψευδοκώδικας τόσο του reduce-side join όσο και του flatMap αυτού παραδίδεται παρακάτω :

```
(key,values):
    float pagerank
    foreach value in values:
        if(value[0]=="rank") pagerank = value[1]
    foreach value in values:
        if(value[0]=="rank") continue
        emit(key,(value[1],pagerank))
```

```
map(key,value):
    outbound = value[0]
    rank = value[1]
    counter = 0
    for each link in outbound:
        counter++
    Lpj = counter
    score = rank/Lpj
    for each link in outbound:
        emit(link,score)
```

Στον παραπάνω ψευδοκώδικα , το emit επιστρέφει παραπάνω από ένα αποτελέσματα . Ουσιαστικά το emit αυτό μπορεί να προσομοιωθεί στην γλώσσα προγραμματισμού python από την εντολή yield η οποία επιστρέφει έναν generator αντί για ένα αντικείμενο (που επιστρέφει η return).

Μετά την εκτέλεση της παραπάνω συνάρτησης για κάθε key του rdd τα δεδομένα μας είναι της μορφής (pageid,rank_contribution). Για να υπολογίσουμε τον όρο $\sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)}$ αρκεί πλέον να προσθέσουμε όλα τα values που έχουν το ίδιο key . Η πρόσθεση αυτή υλοποιείται εύκολα με ένα reduceByKey(). Στην συνέχεια παραθέτουμε ψευδοκώδικα της συγκεκριμένη reduce .

```
reduce(key,values):
    total = 0
    for each rank in values:
        total += rank
    emit(total,rank)
```

Τέλος , αφού υπολογίσαμε τον όρο $\sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)}$ μένει να τον πολλαπλασιάσουμε επί την σταθερά d και να προσθέσουμε τον σταθερό όρο $\frac{1-d}{N}$. Προφανώς , η παραπάνω πράξη υλοποιείται εύκολα με μία mapValues . Ο ψευδοκώδικας της συνάρτησης map αυτής φαίνεται παρακάτω :

```
map(key,value):
    constant = (1-d)/N
    PR = constant+(value*d)
    emit(key,PR)
```

Τέλος , εκτελούμε τον παραπάνω ψευδοκώδικα 5 φόρες (όπως ορίζεται στην εκφώνηση της άσκησης μας) . Αφού ολοκληρωθεί η εκτέλεση του επαναληπτικού αλγορίθμου μας αποθηκεύουμε τα αποτελέσματα μας σε ένα αρχείο στο hdfs distributed file system .

Δεδομένου ότι το dataset μας αποτελείται από περίπου 850000 ιστοσελίδες δεν υπάρχει ιδιαίτερο νόημα στο να παραθέσουμε όλα τα αποτελέσματα που εξάγαμε . Ενδεικτικά παραθέτουμε ένα μέρος των αποτελεσμάτων μας παρακάτω :

NodeId	***PageRank***
(0,	6.0986762555401740000)
(1,	0.3566907417193308000)
(131077,	0.0031997574323273286)
(786433,	0.0302677064291229230)
(393229,	0.0136250158940944600)
(633517,	0.0083295367806301180)
(524305,	0.1524530896161032000)
(655381,	0.1155811140582307100)
(873817,	0.0407109669896978900)
(393241,	0.1884712600320331000)
(524317,	0.0008990606302872310)
(720901,	0.0006467777739878668)
(655393,	0.0183908918912001960)
(393253,	0.0245648459587892100)
(174769,	0.0377271763500902840)
(524329,	0.1867653386277535200)
(602305,	0.0279731042026613970)
(655405,	0.0497853230498535440)
(786481,	0.0457011880873668900)
(524341,	0.4705146807019137700)
(199021,	0.0006529359543181205)
(65545,	0.0319384106874031000)
(262201,	0.1526611804220313300)
(393277,	0.0495334377111961640)
(524353,	0.1913289260721190400)
(91657,	0.0118957476819996900)
(262213,	0.2216124169901704800)
(148393,	0.0737928913699734700)
(222589,	0.0164450650990087820)
(73,	0.1325690076967904800)

Άσκηση 4 – Γραμμική άλγεβρα - πολλαπλασιασμός πίνακα

Στην άσκηση αυτή θα υλοποιήσουμε την πράξη του πολλαπλασιασμού πινάκων σε map-reduce context. Για τις ανάγκες τις άσκησης αυτής μας δίνονται δύο αρχεία csv σαν είσοδος .Πιο συγκεκριμένα μας δίνονται δύο πίνακες A.csv, B.csv, οι οποίοι είναι της μορφής (row, column, number). Το αποτέλεσμα το οποίο θα παραχθεί μετά την εκτέλεση του αλγορίθμου μας θα είναι της ίδιας μορφής και θα περιέχει το αποτέλεσμα του πολλαπλασιασμού των πινάκων A,B. Πριν προχωρήσουμε όμως στην ανάλυση και επεξήγηση του αλγορίθμου μας θεωρούμε σκόπιμο να παραθέσουμε μία οπτική αναπαράσταση του πολλαπλασιασμού πινάκων για την καλύτερη κατανόηση της πράξης αυτής . Η εικόνα αυτή παρατίθεται παρακάτω .

$$\begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ 1 & 2 & 3 & 4 \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix} \begin{bmatrix} \cdot & \cdot & 4 & \cdot \\ \cdot & \cdot & 3 & \cdot \\ \cdot & \cdot & 2 & \cdot \\ \cdot & \cdot & 1 & \cdot \end{bmatrix} = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 20 & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix}$$

$$c_{23} = \sum_{k=1}^{4} a_{2k} b_{k3} = 1 \cdot 4 + 2 \cdot 3 + 3 \cdot 2 + 4 \cdot 1 = 20$$

Ουσιαστικά μετά των πολλαπλασιασμό δύο πινάκων διαστάσεων MxN και NxK θα προκύψει ένας νέος πίνακας διαστάσεων MxK κάθε στοιχείο του οποίου υπολογίζεται από την παρακάτω σχέση $c_{ij} = \sum_k a_{ik} b_{kj}$!

Στην συνέχεια λοιπόν θα περιγράψουμε και θα επεξηγήσουμε τον αλγόριθμο της υλοποίησης μας. Αρχικά, χωρίσαμε τα δεδομένα του A με βάση την κάθε γραμμή.

```
map(key, value):
    emit(key[0],(key[1],value))
```

Στην συνέχεια, χωρίσαμε τα δεδομένα του B με βάση την κάθε στήλη του πίνακα αυτού .

```
map(key, value):
    emit(key[1],(key[0],value))
```

Μετά από τις παραπάνω δύο map συναρτήσεις έχουμε ομαδοποιήσει τα στοιχεία του πίνακα Α κατά γραμμές και τα στοιχεία του πίνακα Β κατά στήλες. Έπειτα, πρέπει να αντιστοιχίσουμε όλα τα στοιχεία των γραμμών του Α μεταξύ τους καθώς και τα στοιχεία των στηλών του Β αντίστοιχα:

```
reduce(keyA, valuesA):
    emit(keyA,valuesA)

reduce(keyB, valuesB):
    emit(keyB,valuesB)
```

Επειδή όμως τα δεδομένα μας δεν έρχονται ταξινομημένα, θα πρέπει να κάνουμε μία ταξινόμηση σε όλες τις γραμμές του Α και τις στήλες του Β, ώστε να έχουμε τα στοιχεία στην κατάλληλη σειρά ώστε να τα προσθέσουμε αργότερα.

```
map(keyA, valuesA):
    sort_by_first_element_of_tuple(valuesA)
    emit(keyA,valuesA)

map(keyB, valuesB):
    sort_by_first_element_of_tuple(valuesB)
    emit(keyB,valuesB)
```

Έπειτα, πρέπει να αντιστοιχίσουμε όλες τις γραμμές του Α με όλες τις στήλες του Β. Ουσιαστικά , επιθυμούμε να εκτελέσουμε την πράξη του εξωτερικού γινομένου των δύο παραπάνω πινάκων .Για την υλοποίηση αυτού στον κώδικα μας χρησιμοποιήσαμε την συνάρτηση του spark cartesian. Για τις ανάγκες τις αναφοράς αυτής όμως θα υλοποιήσουμε την πράξη του εξωτερικού γινομένου με χρήση map-reduce (όπως στην ουσία υλοποιείται και εσωτερικά στο Spark).

Για την υλοποίηση της πράξης αυτής λοιπόν επιθυμούμε να φέρουμε όλες τις γραμμές του Α και όλες τις στήλες του Β στο ίδιο bucket (ή καλύτερα στον ίδιο reducer).
Για τον πίνακα Α έχουμε:

```
map(row,list(column,value)):
    emit(*,A:(row, list(column,value)))
```

Για τον πίνακα Β έχουμε:

```
map(column,list(row,value)):
    emit(*,B:(column,list(row,value)))
```

Τέλος , θα περάσουμε τα παραπάνω από τον ίδιο reducer . Στον reducer αυτόν θα κάνουμε emit κάθε γραμμή του πίνακα A με κάθε στήλη του πίνακα B.

```
reduce(*, list(values)):
    arrayA=[]
    arrayB=[]
    for i in list(values):
        if(vaule contains A) arrayA+=list[i]
        else arrayB+=list[i]
    for i in arrayA:
        for j in arrayB:
            emit((arrayA[i].row, arrayB[j].column),
list(arrayA[i].list,arrayB[j].list))
```

Τέλος, θέλουμε να υπολογίσουμε για όλα τα επιμέρους στοιχεία τον πολλαπλασιασμό ανάμεσα στα αντίστοιχα στοιχεία της γραμμής του A με τη στήλη του B, και μετά το άθροισμα όλων μεταξύ τους:

```
map((rowA, columnB), list(values)):
    sum=0
    rowofA=list[0]
    columnofB=list[1]
    for i in rowofA:
        sum+=rowofA[i]*columnofB[i]
    emit(rowA,columnB,sum)
```

Αφού ολοκληρωθεί η εκτέλεση του αλγορίθμου μας αποθηκεύουμε τα αποτελέσματα μας σε ένα αρχείο στο hdfs distributed file system .Παρακάτω , παρατίθεται ένα μέρος των αποτελεσμάτων μας.

Δεδομένου ότι ο πίνακας που θα δημιουργηθεί από τον πολλαπλασιασμό πινάκων θα είναι μεγέθους 10K x 5K δεν υπάρχει ιδιαίτερο νόημα στο να παραθέσουμε όλα τα αποτελέσματα που εξάγαμε . Ενδεικτικά παραθέτουμε ένα μέρος των αποτελεσμάτων (στοιχεία της πρώτης στήλης του πίνακα) μας παρακάτω :

Row,Column	***Value***
((0, 0),	23946)
((0, 1),	24956)
((0, 2),	23670)
((0, 3),	24211)
((0, 4),	24502)
((0, 5),	25529)
((0, 6),	25094)
((0, 7),	24531)
((0, 8),	25205)
((0, 9),	25067)
((0, 10),	25113)
((0, 11),	24580)
((0, 12),	23794)
((0, 13),	23795)
((0, 14),	24862)
((0, 15),	24544)
((0, 16),	25031)
((0, 17),	24467)
((0, 18),	24639)
((0, 19),	24502)
((0, 20),	24476)
((0, 21),	23617)
((0, 22),	23976)
((0, 23),	25557)
((0, 24),	22989)
((0, 25),	25118)
((0, 26),	24031)
((0, 27),	24318)
((0, 28),	24146)
((0, 29),	24856)
((0, 30),	24209)

Τέλος , θα παραθέσουμε τους συνδέσμους για το hdfs site τόσο των αρχικών dataset όσο και των τελικών αποτελεσμάτων .

Datasets →

A.csv:
http://83.212.74.80:50075/webhdfs/v1/user/user0023/inputs/A.csv?op=OPEN&namenoderpcaddress=master:9000&offset=0

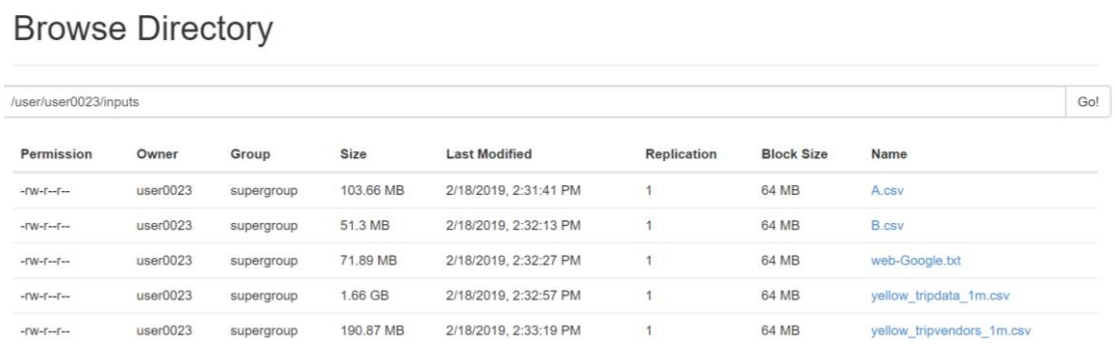
B.csv:
http://83.212.74.80:50075/webhdfs/v1/user/user0023/inputs/B.csv?op=OPEN&namenoderpcaddress=master:9000&offset=0

web-Google.txt:
http:// 83.212.74.80:50075/webhdfs/v1/user/user0023/inputs/web-Google.txt?op=OPEN&namenoderpcaddress=master:9000&offset=0

yellow_tripdata_1m.csv:
http://83.212.74.80:50075/webhdfs/v1/user/user0023/inputs/yellow_tripdata_1m.csv?op=OPEN&namenoderpcaddress=master:9000&offset=0

yellow_tripvendors_1m.csv:
http://83.212.74.80:50075/webhdfs/v1/user/user0023/inputs/yellow_tripvendors_1m.csv?op=OPEN&namenoderpcaddress=master:9000&offset=0

Συνολική παρουσίαση :
http://83.212.74.80:50070/explorer.html#/user/user0023/inputs



Αποτελέσματα →

Άσκηση 1α :
http://83.212.74.80:50075/webhdfs/v1/user/user0023/outputs/ex1a.out?op=OPEN&namenoderpcaddress=master:9000&offset=0

Άσκηση 1β:
http://83.212.74.80:50075/webhdfs/v1/user/user0023/outputs/ex1b.out?op=OPEN&namenoderpcaddress=master:9000&offset=0

Άσκηση 2:
http://83.212.74.80:50075/webhdfs/v1/user/user0023/outputs/ex2.out?op=OPEN&namenoderpcaddress=master:9000&offset=0

Άσκηση 3:
http://83.212.74.80:50075/webhdfs/v1/user/user0023/outputs/ex3.out?op=OPEN&namenoderpcaddress=master:9000&offset=0

Άσκηση 4:
http://83.212.74.80:50075/webhdfs/v1/user/user0023/outputs/ex4.out?op=OPEN&namenoderpcaddress=master:9000&offset=0

Συνολική παρουσίαση :
http://83.212.74.80:50070/explorer.html#/user/user0023/outputs

Browse Directory

/user/user0023/outputs Go!

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	user0023	supergroup	583 B	2/19/2019, 10:55:58 PM	1	64 MB	ex1a.out
-rw-r--r--	user0023	supergroup	30 B	2/19/2019, 10:56:48 PM	1	64 MB	ex1b.out
-rw-r--r--	user0023	supergroup	224 B	2/19/2019, 11:09:49 PM	1	64 MB	ex2.out
-rw-r--r--	user0023	supergroup	18.88 MB	2/19/2019, 10:54:46 PM	1	64 MB	ex3.out
-rw-r--r--	user0023	supergroup	1.01 GB	2/23/2019, 6:22:24 PM	1	64 MB	ex4.out