mean_trip_duration_byhour

στο yellow_tripdata.csv:

```
MAP(key, value):
record = value.toString();
String parts = record.split(“,”);
String key2 = parts[0];
value2 = (parts[1], 1)
emit(key2,value2)

REDUCE(key, list(value):
    totalDuration = 0
    totalTrips=0
    for item in list:
        totalDuration += duration
        totalTrips ++
emit (departureTime, (totalDuration, totalTrips))

MAP(departureTime, (totalDuration, totalTrips)):
emit (departureTime, totalDuration/totalTrips)

REDUCE(key, list(mean_duration)):
    for mean in list:
        sortbykey(mean_duaration)
emit(list)
```

---

mean_payment_byvendor

στο yellow_tripdata.csv:

```
MAP(key, value):
    record = value.toString()
    parts = record.split(“,”)
    key2 = parts[0] #tripID
    value2 = “payment,” + parts[7]) #payment
emit(key2,value2)
```

στο yellow_tripvendors.csv:

```
MAP(key, value):
        record = value.toString()
        parts = record.split(“,”)
        key2 = parts[0] #tripID
        value2 = “vendor,” + parts[1] #vendorID
emit(key2,value2)
```

REDUCE(key,list(value)):

```
        for t in values:
                parts[] = t.toString().split(“,”)
                 if parts[0] == “payments”
                        payment=parts[1]
                elif parts[0] == “vendor”:
                        vendorID=parts[1]
emit (key, (payment, vendorID))
```

on the above created data set:

MAP(key, value):

```
        record = value.toString()
        parts = record.split(“,”)
        αkey2 = parts[1] #company
        value2 = parts[0] #payment
emit(key2,value2)
```

REDUCE(key,list(value)):

```
        #find the max payment per company
        max = 0
        for t in values:
                if t>max:
                        max = t
emit(key, max)
```

on the above created data set:

MAP(key, value):

```
        emit(key, value)
```

REDUCE(key, list(value)):

```
        sortbykey
        emit(list)
```

---

top5_hopon_clusters

στο yellow_tripdata_1m.csv:

```
MAP(key, value):
        record = value.toString()
        parts = record.split(“,”)
        key2 = (parts[0], parts[1]) #lon, lat
emit(key2)
```

στο παραπάνω dataset και επαναληπτικά κάνουμε τα παρακάτω:

```
MAP(key, value):
        record = value.toString()
        parts = record.split(“,”)
        find_min_distance_from_key_to_centroids()
emit(centroid_id, (lon, lat, 1))

REDUCE(key,list(value)):
        count = 0
        total_lon = 0
        total_lat = 0
        for t in values:
                parts[] = t.toString().split(“,”)
                total_lon += parts[0]
                total_lat += parts[1]
                count+= parts[2]
emit (key, (total_lon, total_lat, count))
```

στα προηγούμενα δεδομένα:

```
MAP(key, value):
        record = value.toString()
        parts = record.split(“,”)
emit((parts[0]/parts[2], parts[1]/parts[2])
```

ό,τι προκύπτει από την τελευταία MAP χρησιμοποιείται ως entry στο loop

---

page_rank

στο δοθεν dataset: δημιουργούμε το rdd “edges”

```
MAP(key, value):
    record = value.toString()
    parts = record.split(“,”)
    key2 = parts[0] #dep_page
    value2 = parts[1] #dest_page
emit(key2,value2)
```

στο edges: δημιουργούμε το rdd “L”

```
MAP(key, value):
    record = value.toString()
    parts = record.split(“,”)
    key2 = key #dep_page
    value2 = 1
emit(key2,value2)

REDUCE(key,list(value)):
    #for each page find the number of outbound links from that page
    sum = 0
    for t in values:
        sum += t
emit(key, sum:q)
```

στο edges: δημιουργούμε το rdd “M”

```
MAP(key, value):
    #reverse the initial data
    record = value.toString()
    parts = record.split(“,”)
    key2 = value #dest_page
    value2 = key #dep_page
emit(key2,value2)
```

στο edges: δημιουργούμε το rdd “PR” με τις αρχικές πιθανότητες (0.5)

```
MAP(key, value):
    #initial probabilities 0.5
    record = value.toString()
    parts = record.split(“,”)
    key2 = key #dep_page
    value2 = 0.5
emit(key2,value2)
```

Για τις 5 επαναλήψεις:
join PR & L => klasma (dep_page, pr) kai (dep_page, sum) => (dep_page, (pr, sum)
στο PR:

```
MAP(key, value):
    record = value.toString()
    parts = record.split(“,”)
    key2 = key #dep_page
    value2 = “pr,”+parts[0]
emit(key2,value2)
```

στο L:

```
MAP(key, value):
    record = value.toString()
    parts = record.split(“,”)
    key2 = key #dep_page
    value2 = “sum,”+ parts[0])
emit(key2,value2)

REDUCE(key,list(value)):
    for t in values:
        parts[] = t.toString().split(“,”)
         if parts[0] == “pr”
            pr=parts[1]
        elif parts[0] == “sum”:
            sum =parts[1]
emit (key, (pr, sum))
```

Κάνουμε join το προηγούμενο dataset με το αρχικό για να πάρουμε (dep_page, (dest_page, klasma)):

δημιουργώ το rdd “klasma”

```
MAP(key, value):
        record = value.toString()
        parts = record.split(“,”)
        key2 = key #dep_page
        value2 = “klasma,”+parts[0]/parts[1] #pr/sum
emit(key2,value2)
```

στην edges:

```
MAP(key, value):
        record = value.toString()
        parts = record.split(“,”)
        key2 = key #dep_page
        value2 = “dest_page,”+parts[0]
emit(key2,value2)

REDUCE(key,list(value)):
        for t in values:
                parts[] = t.toString().split(“,”)
                 if parts[0] == “klasma”
                        klasma=parts[1]
                elif parts[0] == “dest_page”:
                        dest_page=parts[1]
emit (key, (dest_page, klasma))
```

στο προηγούμενο dataset:

```
MAP(key, value):
        record = value.toString()
        parts = record.split(“,”)
        key2 = parts[0] #dest_page (pi)
        value2 = parts[1] #klasma
emit(key2,value2)

REDUCE(key,list(value)):
        sum = 0
        for t in values:
                sum += t
emit (key, sum) #(dest_page, sum of klasmata)
```

στο παραπάνω dataset:

```
MAP(key, value):
        record = value.toString()
        parts = record.split(“,”)
        key2 = key
        value2 = (1 - d)/N + d*parts[0]
emit(key2,value2)
```

---

array_multiplication

στο A:

```
MAP(key, value):
        record = value.toString()
        parts = record.split(“,”)
        key2 = parts[0] #rowA
        value2 = (parts[1], parts[2] #colA, valA
emit(key2, value2)
```

```
**REDUCE(key,list(value)):**
        for t in value:
                lista.append(t)
emit (key, lista)

MAP(key, value):
        sortByFirstElement(value)
emit(key2, value2)
```

στο Β:

```
MAP(key, value):
        record = value.toString()
        parts = record.split(“,”)
        key2 = parts[1] #colB
        value2 = (parts[0], parts[2] #rowB, valB
emit(key2, value2)

REDUCE(key,list(value)):
        for t in value:
                lista.append(t)
emit (key, lista)
```

στα παραπάνω datasets:

```
MAP(key, value):
emit( *, ("A", (k,v) )

MAP(key, value):
emit( *, (“B”, (k,v) )
```

Μετά τις παραπάνω δύο MAP έχουμε μαζεμένα όλα τα στοιχεία των δύο αρχικών πινάκων Α και Β μετά το μεταχρηματισμό τους από την προηγούμενη διαδικασία MAP-REDUCE για να μπορέσουμε να υπολογίσουμε το καρτεσιανό τους γινόμενο, όπως φαίνεται παρακάτω:

```
REDUCE(key,list(value)):
        for elem in value:
                if elem[0] == “A”:
```

```
            newA.append(elem[1])
        else:
            newB.append(elem[1])
    for row in newA:
        for col in newB:
            emit( (i,j), (valRow, valCol) )
```

Στη συνέχεια μπορούμε να υπολογίσουμε το τελικό αποτέλεσμα:

```
MAP(key, value): #key = (i,j), value = (valRow, valCol)
    sum = 0;
    for i in range (len(value[0])
        sum += value[0][i] * value[1][i]
emit((i,j),sum)
```