ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
ΤΜΗΜΑ ΗΜΜΥ
Προχωρημένα Θέματα Βάσεων Δεδομένων (ΠΛΗ406)
Εαρινό Εξάμηνο – 2016-2017
Υπεύθυνος Μαθήματος: Καθηγητής Μ. Γαροφαλάκης

Εργασία Εξαμήνου: Υλοποίηση αλγορίθμου υπολογισμού Skyline στο Hadoop

Γενικά

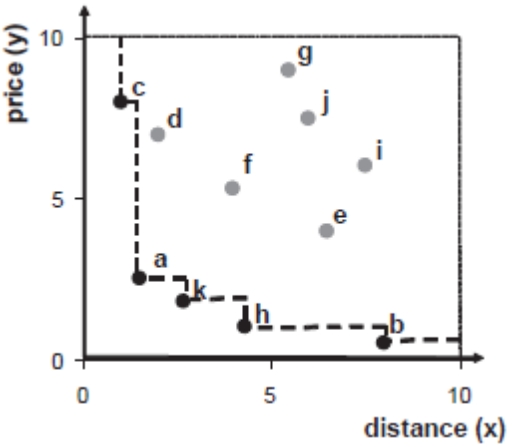
Τα skyline queries έχουν τραβήξει αρκετά το επιστημονικό ενδιαφέρον, μιας και έχουν τη δυνατότητα να βοηθήσουν τους χρήστες στη λήψη πιο έξυπνων αποφάσεων πάνω σε πολύπλοκα δεδομένα όπου πολλά κριτήρια συχνά είναι αντιφατικά. Σε αυτή την εργασία θα προσομοιώσετε το σενάριο αναζήτησης για ενοικίαση διαμερίσματος με κριτήρια την απόσταση από το κέντρο μιας πόλης, την τιμή και την παλαιότητα του διαμερίσματος με τη χρήση Skyline.

Η εργασία σας θα υλοποιηθεί με το πλαίσιο ανάπτυξης λογισμικού Hadoop, προκειμένου να δημιουργήσετε έναν παράλληλο αλγόριθμο εκτέλεσης Skyline Queries. Για να κάνετε την εργασία θα πρέπει να έχετε καταλάβει πως λειτουργεί ο αλγόριθμος BNL (Block Nested Loop) όπως επίσης και το hadoop.

Skyline

Ορισμός: «Ένα σημείο x ανήκει στο Skyline των σημείων S ($x \in SKY_S$) εάν κυριαρχεί έναντι κάθε άλλου σημείου είτε σε όλες τις διαστάσεις του, είτε τουλάχιστον σε μια από αυτές. Το Skyline ορίζεται ως το σύνολο εκείνων των σημείων ($SKY_S \subset S$), τα οποία δεν κυριαρχούνται από κανένα άλλο σημείο από το αρχικό μας σύνολο σημείων».

Ένας απλός τρόπος υπολογισμού του Skyline σε ένα σύνολο σημείων προϋποθέτει για κάθε σημείο να συγκριθεί με όλα τα υπόλοιπα σημεία και να κυριαρχήσει ως skyline σημείο αν αντέξει όλους του ελέγχους με όλα τα άλλα σημεία (Nested Loop Evaluation). Έχει, όμως, κακή συμπεριφορά Εισόδου/Εξόδου (I/O) καθώς απαιτεί την φόρτωση κάθε αντικειμένου |S| φορές από τη βάση δεδομένων. Όσο μεγαλώνουν τα δεδομένα τόσο πιο αργός θα γίνεται ο αλγόριθμος.



Εικόνα 1 Skyline

Block Nested Loop

Βελτιωμένος nested loop αλγόριθμος (βλ. [1]). Αντί να εξετάζουμε ένα υποψήφιο σημείο την φορά, κρατάμε όλα τα υποψήφια σημεία σε ένα παράθυρο.

- Εξωτερικός βρόχος: επανέλαβε πάνω στα δεδομένα
- Εσωτερικός βρόχος: σύγκρινε το δεδομένο με όλα τα υποψηφία skyline σημεία που είναι στο παράθυρο
  - Εάν το δεδομένο υπερισχύετε, βγες από τον εσωτερικό βρόχο
  - Εάν το δεδομένο υπερισχύσει έναντι κάποιου υποψήφιου skyline σημείου, αφαίρεσε το υποψήφιο skyline σημείο από το παράθυρο
- Τα δεδομένα που θα επιβιώσουν τον εσωτερικό βρόχο προστίθενται στο παράθυρο ως υποψήφια skyline σημεία.

Διαμέριση Δεδομένων

Για να έχει καλή απόδοση ο BNL, το παράθυρο πρέπει να χωράει στη κεντρική μνήμη. Προκειμένου να μπορούμε να επεξεργαστούμε πολύ μεγάλα δεδομένα, θα πρέπει να τα χωρίσουμε.

Παρατήρηση: «Ένα σημείο x είναι σημείο Skyline ($x \in SKY_S$) αν και μόνο αν υπάρχει μια διαμέριση των δεδομένων ($D_i \subset S$) στην οποία ανήκει το σημείο αυτό ($x \in D_i$) και είναι σημείο Skyline στη διαμέριση αυτή ($x \in SKY_{D_i}$).»

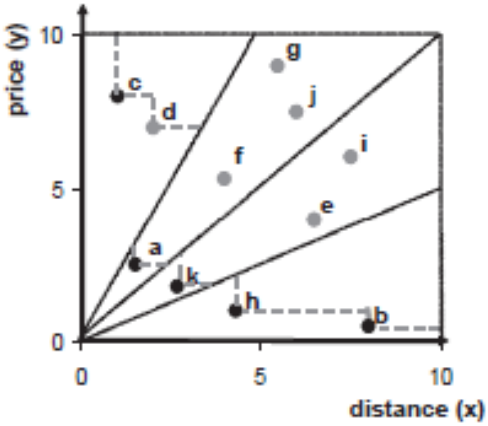
Με άλλα λόγια τα Skyline σημεία των δεδομένων αποτελούν υποσύνολο της ένωσης των Skyline σημείων όλων των επιμέρους διαμερίσεων. Με αυτή την παρατήρηση γίνεται προφανές ότι μπορούμε διαμερίζοντας τα δεδομένα να βρούμε τα τοπικά Skylines για κάθε διαμέριση και συγχωνεύοντας τα τοπικά Skylines μπορούμε να βρούμε το ολικό Skyline όλων των δεδομένων.

Τυχαία Διαμέριση Δεδομένων

Ένας απλός τρόπος διαμοιρασμός των δεδομένων εισόδου σε N διαμερίσεις μπορεί να γίνει με τυχαίο τρόπο για κάθε δεδομένο και ίση πιθανότητα για κάθε μια από τα διαμερίσεις. Η τυχαία διαμέριση έχει χρησιμοποιηθεί για τον παράλληλο υπολογισμό του Skyline (βλ. [3]). Χωρίζοντας τα δεδομένα με αυτό τον τρόπο είναι πιθανό ότι η κάθε διαμέριση να ακολουθεί την κατανομή των αρχικών δεδομένων. Με αυτό τον τρόπο μπορούμε να πούμε ότι γίνεται ένας δίκαιος διαμοιρασμός των δεδομένων και περιμένουμε να παράγεται παρόμοιος αριθμός Skyline δεδομένων και παρόμοιο ποσοστό αυτών να ανήκει στο ολικό Skyline. Επιπλέον, είναι αρκετά εύκολο να υλοποιηθεί για παράλληλη επεξεργασία και προσδίδει ελάχιστο επιπλέον κόστος στην επεξεργασία των δεδομένων.

Γωνιακή Διαμέριση Δεδομένων

Ένας άλλος τρόπος διαμοιρασμός των δεδομένων, που είναι πιο αποδοτικός (βλ. [2]), είναι ο διαχωρισμός των δεδομένων βάσει της γωνίας τους από την αρχή των αξόνων. Αυτή η τεχνική πρώτα μετατρέπει το καρτεσιανό πεδίο σε υπερσφαιρικό πεδίο και έπειτα κατακερματίζει τα δεδομένα βάσει των γωνιακών συντεταγμένων των N διαμερίσεων.



Εικόνα 2 Γωνιακή Διαμέριση 2-d

Οι καρτεσιανές συντεταγμένες για ένα σημείο x = [x_1, x_2, ..., x_d] μετατρέπονται σε μια ακτινική συντεταγμένη (*r*) και *d-1* γωνιακές συντεταγμένες (φ_i) με βάση τις παρακάτω εξισώσεις:

$$\begin{aligned} r &= \sqrt{x_n^2 + x_{n-1}^2 + \ldots + x_1^2} \\ \tan(\phi_1) &= \frac{\sqrt{x_n^2 + x_{n-1}^2 + \ldots + x_2^2}}{x_1} \\ &\ldots \\ \tan(\phi_{d-2}) &= \frac{\sqrt{x_n^2 + x_{n-1}^2}}{x_{n-2}} \\ \tan(\phi_{d-1}) &= \frac{x_n}{x_{n-1}} \end{aligned} \qquad \text{(Εξ.1)}$$

Παρατηρήστε γενικά ότι $0 \le \varphi_i \le \pi$ για $i < d-1$, και $0 \le \varphi_{d-1} \le 2\pi$, αλλά στην περίπτωσή μας $0 \le \varphi_i \le \frac{\pi}{2}$ για $i \le d-1$. Αυτό είναι γιατί θεωρούμε ότι για οποιοδήποτε σημείο x οι συντεταγμένες x_i είναι θετικές για όλες τις διαστάσεις ($x_i \ge 0\ \forall i$) και μέγιστη τιμή το *L*.

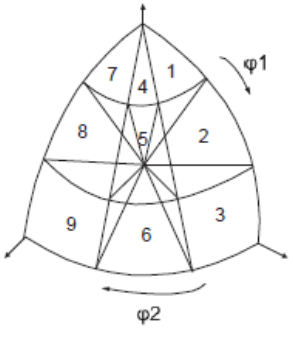
Γενικά, δεδομένου S δεδομένων και *d*-διαστάσεων, με τη γωνιακή διαμέριση μπορούμε να έχουμε *N* διαμερίσεις, όπου 1≤ *N* ≤S με την κάθε διαμέριση *i* (όπου 1≤ *i* ≤N) των δεδομένων να ορίζεται από τις γωνιακές συντεταγμένες ως: $D_i=[\varphi_1^{i-1}, \varphi_1^i]x\ldots x[\varphi_{d-1}^{i-1}, \varphi_{d-1}^i]$ όπου $\varphi_j^0 = 0°$ και $\varphi_j^N = 90°$ για $1 \le j \le d$, ενώ φ_j^{i-1} και φ_j^i είναι τα όρια της γωνιακής συντεταγμένης φ_j για τη διαμέριση *i*.

Αυτό που μένει να οριστεί είναι τα όρια των γωνιακών συντεταγμένων προκειμένου να επιτευχθεί όσον το δυνατόν ισόποσος διαμοιρασμός δεδομένων. Αν θεωρήσουμε ότι έχουμε ομοιόμορφη κατανομή δεδομένων, και οι διαμερίσεις χωριστούν με τέτοιο τρόπο ώστε να έχουν τελικά τον ίδιο περίπου όγκο $V_d^i = \frac{V_d}{N}, (1 \le i \le N)$ είναι αρκετό για να διασφαλίσουμε ότι θα έχουμε περίπου τον ίδιο αριθμό στοιχείων σε κάθε διαμέριση D_i. Ο όγκος γενικά για κάθε διαμέριση υπολογίζεται από την παρακάτω εξίσωση:

$$V_d^i = \int_0^r \int_{\varphi_1^i}^{\varphi_1^{i+1}} \ldots \int_{\varphi_{d-1}^i}^{\varphi_{d-1}^{i+1}} r^{d-1} \sin^{d-2}(\varphi_1) \ldots \sin(\varphi_{d-2})\, dr d\varphi_1 \ldots d\varphi_{d-1} \quad \text{(Εξ. 2)}$$

Στην εικόνα 3 βλέπουμε τη γωνιακή διαμέριση σε δεδομένα d=3 διαστάσεων με τη χρήση των γωνιακών συντεταγμένων φ_1 και φ_2 σε N=9 διαμερίσεις. Ο όγκος των απεικονιζόμενων δεδομένων ισούται με το 1/8 του όγκου της σφαίρας, το οποίο ισούται με $V_3 = \frac{\pi L^3}{6}$. Για κάθε διαμέριση *i* τελικά έχουμε όγκο ίσο με:

$$V_d^i = \int_0^L \int_{\varphi_1^{i-1}}^{\varphi_1^i} \int_{\varphi_2^{i-1}}^{\varphi_2^i} r^2 \sin(\varphi_1)\, dr d\varphi_1 d\varphi_2 = L^3\left(cos(\varphi_1^{i-1}) - \cos(\varphi_1^i)\right)(\varphi_2^i - \varphi_2^{i-1})/3$$



Εικόνα 3 Γωνιακή Διαμέριση 3-d

Για να έχουμε ίδιο όγκο σε κάθε ένα από τις 9 διαμερίσεις θα πρέπει να πάρουμε $\frac{V_3}{9}$ και αν λύσουμε τις παραπάνω εξισώσεις θα πάρουμε $\varphi_1^1 = 48.24°$, $\varphi_1^2 = 70{,}55°$, $\varphi_2^1 = 30°$, $\varphi_2^2 = 60°$ και με $D_i=[\varphi_1^{i-1}, \varphi_1^i]x[\varphi_2^{i-1}, \varphi_2^i]$ έχουμε:

$D_7=[0°, 48.24°]x[60°, 90°]$, $D_4=[0°, 48.24°]x[30°, 60°]$, $D_1=[0°, 48.24°]x[0°, 30°]$

$D_8=[48.24°, 70.52°]x[60°, 90°]$, $D_5=[48.24°, 70.52°]x[30°, 60°]$, $D_2=[48.24°, 70.52°]x\ [0°, 30°]$

$D_9=[70.52°, 90°]x[60°, 90°]$, $D_6=[70.52°, 90°]x[30°, 60°]$, $D_3=[70.52°, 90°]x[0°, 30°]$

με τη σειρά εμφάνισης στην εικόνα 3.

Hadoop

Το Hadoop είναι ένα μεγάλης κλίμακας κατανεμημένο σύστημα κατάλληλο για μαζική επεξεργασία δεδομένων. Το μεγαλύτερο πλεονέκτημα του είναι ότι δίνει την δυνατότητα στον προγραμματιστή να τρέξει σε εκατοντάδες μηχανήματα, κάθε ένα από τα οποία με πολλούς επεξεργαστές. Επιπλέον, το Hadoop έχει ενσωματωμένο κατανεμημένο σύστημα αρχείων ώστε να διαμοιράζεται μεγάλος όγκος δεδομένων στους κόμβους του συστήματος.

Μοντέλο προγραμματισμού Map-Reduce

Μια Map-Reduce εφαρμογή συνήθως σπάει σε ανεξάρτητα κομμάτια την είσοδο της, τα όποια διαβάζονται και επεξεργάζονται ταυτόχρονα από Map συναρτήσεις. Έπειτα τα αποτελέσματα των Map συναρτήσεων ταξινομούνται από το Hadoop και προωθούνται στις Reduce συναρτήσεις. Πιο αναλυτικά η MapReduce εφαρμογής αποτελείται από δύο φάσεις:

1. Το πρώτο στάδιο μιας MapReduce εφαρμογής ονομάζεται mapping. Μια λίστα από στοιχεία δίδονται ένα-ένα σε μία συνάρτηση που ονομάζεται Mapper, η οποία μετατρέπει κάθε στοιχείο ξεχωριστά σε μία έξοδο. Π.χ. Έστω ότι θέλαμε να μετρήσουμε τις λέξεις που εμφανίζονται σε ένα κείμενο. Στην πρώτη φάση θα παίρναμε μία-μία λέξη και η Mapper συνάρτηση θα την μετέτρεπε σε μια τιμή εξόδου που θα ήταν ένα ζευγάρι του τύπου <λέξη,1>
2. Το δεύτερο στάδιο (Reducing) ομαδοποιεί τιμές μεταξύ τους. Μια Reducer συνάρτηση δέχεται σαν είσοδο έναν iterator σε τιμές εισόδου και τις συνδυάζει για να παράγει μία τιμή. Συνεχίζοντας το προηγούμενο παράδειγμα μια κλήση της Reducer θα έπαιρνε έναν iterator για κάθε διαφορετική λέξη που υπήρχε στο κείμενο και θα άθροιζε πόσες φορές εμφανίζεται στο κείμενο. Τέλος, θα δημιουργούσε και αυτό με την σειρά του ένα ζευγάρι <λέξη,σύνολο εμφανίσεων>

Hadoop MapReduce

Το Hadoop Map-Reduce αποτελείται από τις δύο παραπάνω φάσεις για κάθε μια υπάρχει και μία κλάση:

- Mapper: Λαμβάνει μέρος της εισόδου και για κάθε εγγραφή/τιμή τρέχει την map μέθοδο
- Reducer: Δέχεται μία λίστα από συσχετιζόμενες εγγραφές/τιμές και ομαδοποιεί τα αποτελέσματα του προηγούμενου βήματος, έτσι δημιουργεί μια σύνοψη των δεδομένων.

Στο Hadoop καμία τιμή δεν υφίσταται από μόνη της. Κάθε τιμή έχει και κάποιο κλειδί που την συνοδεύει. Το κλειδί χρησιμοποιείται για να αναγνωριστούν συσχετιζόμενες τιμές. Π.χ. Για ένα tuple το id της, ο ταχυδρομικός κώδικας σε ταχυδρομικά γράμματα κ.ο.κ. Οπότε κάθε εγγραφή που παίρνει η κλάση Mapper από την είσοδο έχει την μορφή ζευγαριού <key,value>. Το ίδιο ισχύει και για την κλάση Reducer η μόνη διαφορά είναι ότι δεν παίρνει μία εγγραφή, αλλά παίρνει ένα κλειδί και μία λίστα από values με το ίδιο κλειδί. Τέλος, πριν δοθούν οι τιμές στους Reducers παρεμβάλλεται το Hadoop και σύμφωνα με τα κλειδιά ταξινομεί και ομαδοποιεί σε λίστες τα ζευγάρια που έχουν δημιουργηθεί από τους Mappers, έπειτα αναθέτει σε κάθε Reducer ένα μέρος από το σύνολο αυτών των λιστών.

Παράδειγμα (*Πως θα υλοποιούσατε τον τελεστή count στο Hadoop?)*

Έστω ότι έχετε σαν είσοδο την εξής ακολουθία:
[1, 9 ,12, 89, 8,1,89, 2, 4,90,1,12,9]

Εδώ είναι προφανές ότι και για κλειδί και για τιμή θα χρησιμοποιηθεί ο ίδιος ο αριθμός.
Το Hadoop σπάει την είσοδο από μόνο του σε κομμάτια.

Άρα αν έχουμε 2 Mappers και παίρνουν την παρακάτω είσοδο:
Mapper1: [1,9,12,89,8,1,89]
Mapper2: [2, 4,90,1,12,9]

τότε αυτοί θα έχουν την παρακάτω έξοδο:
Mapper1: {<1,1> , <9,9> , <12,12> , <89,89> , <8,8> , <1,1> , <89,89>}
Mapper2: {<2,2> , <4,4> , <90,90> , <1,1> , <12,12> , <9,9>}

Πριν οι λίστες πάνε στους Reducers το Hadoop ταξινομεί και ομαδοποιεί τα values που έχουν τα ίδια κλειδιά και δημιουργήθηκαν από τους Mappers.
Κλειδί 1 τιμές [<1,1,1>]
κλειδί 2 τιμές [<2>]
κλειδί 4 τιμές [<4>]
κλειδί 8 τιμές [<8>]
κλειδί 9 τιμές [<9,9>]
κλειδί 12 τιμές [<12,12>]
κλειδί 89 τιμές [<89,89>]
κλειδί 90 τιμές [<90>]

Έστω ότι έχουμε 2 Reducers που παίρνουν σαν είσοδο (όχι απαραίτητα συνεχόμενα κλειδιά) :
Reducer1: {
<1 , [1,1,1] >,
<89 , [89,89]>,
<90 , [90]>
}
Reducer2: {
< 2 , [2]>,
< 4 , [4]>,
< 8 , [8]>,
< 9 , [9,9]>
}
Και παράγουν έξοδο (ΣΕ ΞΕΧΩΡΙΣΤΑ ΑΡΧΕΙΑ ΤΟ ΚΑΘΈΝΑ)
Reducer1: { <1,3> , <89,2> , <90,1>
Reducer2: { <2,1> , <4,1> , <8,1>, <9,2> }

ΤΙ ΘΑ ΠΡΕΠΕΙ ΝΑ ΥΛΟΠΟΙΗΣΕΤΕ:

Καλείστε να υλοποιήσετε ένα παράλληλο αλγόριθμο, σε Hadoop Map-Reduce, που να βρίσκει τα Skyline σημεία από 1 μεγάλο πίνακα εισόδου εκτελώντας τον αλγόριθμο BNL με γωνιακή και με τυχαία διαμέριση των δεδομένων σε 2 και σε 3 διστάσεις.

Ο αλγόριθμός σας θα πρέπει να:

1. διαβάζει το αρχείο εισόδου
2. να υπολογίζει τα όρια των διαμερίσεων που θα χωριστούν τα δεδομένα (όπου χρειάζεται)
3. να χωρίζει τα δεδομένα, με βάση την επιλεγμένη τεχνική διαμέρισης
4. να βρίσκει τα τοπικά Skyline δεδομένα για τη κάθε διαμέριση με τον αλγόριθμο BNL
5. να συνενώνει τα τοπικά Skyline δεδομένα πάλι με τη χρήση του BNL
6. να γράφει σε 1 αρχείο τα ολικά Skyline δεδομένα

Επιπλέον θα πρέπει:

1. να ορίζεται με επιλογή το πλήθος των διαμερίσεων
2. να ορίζεται με επιλογή αν θα τρέξει με τυχαία ή με γωνιακή διαμέριση
3. να ορίζεται με επιλογή αν θα τρέξει για τις 2 ή τις 3 διαστάσεις

Ειδικά τη για γωνιακή διαμέριση:

1. Απαιτείται η εύρεση των ορίων της κάθε διαμέρισης σε 2-d.
2. Απαιτείται η εφαρμογή των ορίων της κάθε διαμέρισης του παραδείγματος της εικόνας 3 για 3-d για N=9 διαμερίσεις.
3. ΔΕΝ απαιτείται να υλοποιήσετε την (Εξ. 2) καθώς σε 3-d θα ελεγχθεί μόνο για 9 διαμερίσεις, ακριβώς όπως και στο παράδειγμα της εικόνας 3.
4. Απαιτείται η υλοποίηση της (Εξ. 1) προκειμένου να μπορείτε να χωρίσετε σωστά τα δεδομένα που βρίσκονται μέσα στα όρια των διαμερίσεων.

Είστε ελεύθεροι να χρησιμοποιήσετε όσα στάδια θέλετε για να υλοποιήσετε τον αλγόριθμό σας. Σημαντικό κομμάτι για την εργασία σας είναι να καταλάβετε τις ιδιαιτερότητες που έχει η υλοποίηση σε ένα σύστημα σαν το Hadoop και να βρείτε πως θα υλοποιήσετε τον αλγόριθμο που σας ζητήθηκε.

ΠΑΡΑΔΟΤΕΑ:

1) Ο κώδικας σας,
   - σε exported zip archive μορφή
2) Μια αναφορά, στην οποία θα περιγράφετε:
   - τον αλγόριθμο σας
   - τα στάδια map-reduce που χρησιμοποιήσατε
   - σύγκριση απόδοσης για αυξανόμενο αριθμό διαμερίσεων για κάθε τεχνική διαμέρισης (2-d, N=[1,10])
   - σύγκριση απόδοσης μεταξύ των 2 τεχνικών διαμέρισης (2-d, N=10)
   - μελέτη επιβράδυνσης μεταξύ των 2 και των 3 διαστάσεων (2-d και 3-d, N=9)
3) Ένα jar αρχείο, που να τρέχει με την παρακάτω εντολή:
   - `hadoop jar your_jar.jar SkyLine input_file.csv skyline.csv #paritions random|angle 2d|3d`
   - `πχ: hadoop jar giannis_flouris.jar SkyLine input_file.csv skyline.csv 9 angle 3d`

Δεδομένα Εισόδου

Το αρχείο εισόδου (input_file.csv) θα έχει ως πρώτη γραμμή τα ονόματα των στηλών του πίνακα χωρισμένα με κόμμα και οι υπόλοιπες γραμμές θα έχουν τις τιμές για κάθε tuple του πίνακα επίσης χωρισμένα με κόμμα. Ενδεικτικά, δεδομένα θα μπορείτε να βρείτε στη σελίδα του μαθήματος στο cources.

Τα πεδία, ο τύπος τους και το εύρος τιμών τους θα είναι:

- ✓ ApartmentId *(int)*: [1, ∞)
- ✓ Price *(double)*: [10-30]
- ✓ Age *(double)*: [1-21]
- ✓ DistanceFromCityCenter *(double)*: [5-25]

Ακολουθεί ένα ενδεικτικό παράδειγμα αρχείου εισόδου:

ApartmentId , Price, Age, DistanceFromCityCenter
1 , 12.072748615471102 , 4.147990500682161 , 18.17132919130647
2 , 27.873340564792215 , 1.1110920771467794 , 12.764166027692042
3 , 18.084306186532068 , 9.44781531003055 , 24.601088524188647
4 , 10.455029402671139 , 1.26374275858154 , 12.712385883688263
5 , 19.25489187930485 , 15.898793329134161 , 15.366477540889658

ΔΙΑΔΙΚΑΣΤΙΚΑ

- **Αριθμός Μελών Ομάδας Εργασίας:** 1 (Ατομικές)
- **Ημερομηνία Παράδοσης Εργασίας:** Τα μεσάνυχτα της ημέρας Εξέτασης του μαθήματος
- **Τρόπος Παράδοσης:** Μέσω e-mail στο βοηθό του μαθήματος
  - Προς: giannisflouris at gmail.com
  - Τίτλος: ProjectComp406
  - Περιεχόμενο:
    - Το ονοματεπώνυμο σας
    - Το Α.Μ. σας
    - Ένα DropBox link ή GDrive link στο οποίο θα βρίσκονται τα 3 ζητούμενα παραδοτέα με εμπρόθεσμη ημερομηνία τελευταίας αλλαγής(Date Modified)
- **Ημερομηνία Προφορικής Εξέτασης:** Κατόπιν ανακοίνωσης, μετά την γραπτή εξέταση του μαθήματος

- **ΠΡΟΣΟΧΗ:**
  - Ο κώδικας και οι αναφορές θα ελεγχθούν με ειδικό λογισμικό ανίχνευσης αντιγραφής
  - Η αντιμετώπιση φαινομένων αντιγραφής θα είναι αυστηρή
  - Εκπρόθεσμες Εργασίες ΔΕΝ θα γίνουν αποδεκτές
  - Τυχόν αλλαγές στα παραδοτέα μετά την πάροδο της προθεσμίας ΔΕΝ θα γίνουν αποδεκτές

ΚΑΛΗ ΔΟΥΛΕΙΑ!

Βιβλιογραφία:

[1] Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. 2001. The Skyline Operator. In Proceedings of the 17th International Conference on Data Engineering. IEEE Computer Society, Washington, DC, USA, 421-430.

[2] Akrivi Vlachou, Christos Doulkeridis, and Yannis Kotidis. 2008. Angle-based space partitioning for efficient parallel skyline computation. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data* (SIGMOD '08). ACM, New York, NY, USA, 227-238.

[3] A. Cosgaya-Lozano, A. Rau-Chaplin, and N. Zeh. Parallel computation of skyline queries. In Proc. of Int. Symp. on High Performance Computing Systems and Applications, page 12, 2007.