Πανεπιστήμιο Πατρών

Πολυτεχνική Σχολή

Τμήμα Μηχανικών Η/Υ και Πληροφορικής

Πολυδιάστατες Δομές Δεδομένων

2024-2025

Project-1: Multi-dimensional Data Indexing and Similarity Query Processing

Συνεργάτες

Ον/μο	ΑΜ	Email	Εξάμηνο
Βασίλειος Κουτρουμπέλας	1093397	up1093397@ac.upatras.gr	7
Φίλιππος Μινώπετρος	1093431	up1093431@ac.upatras.gr	7
Χαράλαμπος Αναστασίου	1093316	up1093316@ac.upatras.gr	7
Θεόφραστος Παξιμάδης	1093460	up1093460@ac.upatras.gr	7
Κωνσταντίνος Αναστασόπουλος	1093320	up1093320@ac.upatras.gr	7

Table of Contents

Αρχικές παράμετροι

Δεδομένα

Το σύνολο δεδομένων που χρησιμοποιήθηκε ήταν το
https://www.kaggle.com/datasets/schmoyote/coffee-reviews-dataset?select=simplified_coffee.csv

Διαστάσεις Δομών

Οι δομές υλοποιήθηκαν με indexing σε τρεις στήλες του dataset για τα δέντρα (100g_USD, rating, review_date) και σε μία (review) για το hashing. Συνολικά χρησιμοποιήθηκαν τέσσερεις στήλες της κάθε εγγραφής για τη δεικτοδότηση αλλά κάθε κόμβος των δομών έχει όλες τις στήλες ως περιεχόμενο. Οι στήλες 100g_USD και rating περιέχουν αριθμητικά δεδομένα, ενώ η review_date περιέχει ημερομηνίες τις οποίες με τη βοήθεια της συνάρτησης date_to_number.py μετατρέπουμε σε αριθμούς που διατηρούν τη διάταξη των ημερομηνιών. Η στήλη review περιέχει ολόκληρες προτάσεις και είναι η μόνη κατάλληλη για χρήση LSH.

Δομή Προγράμματος

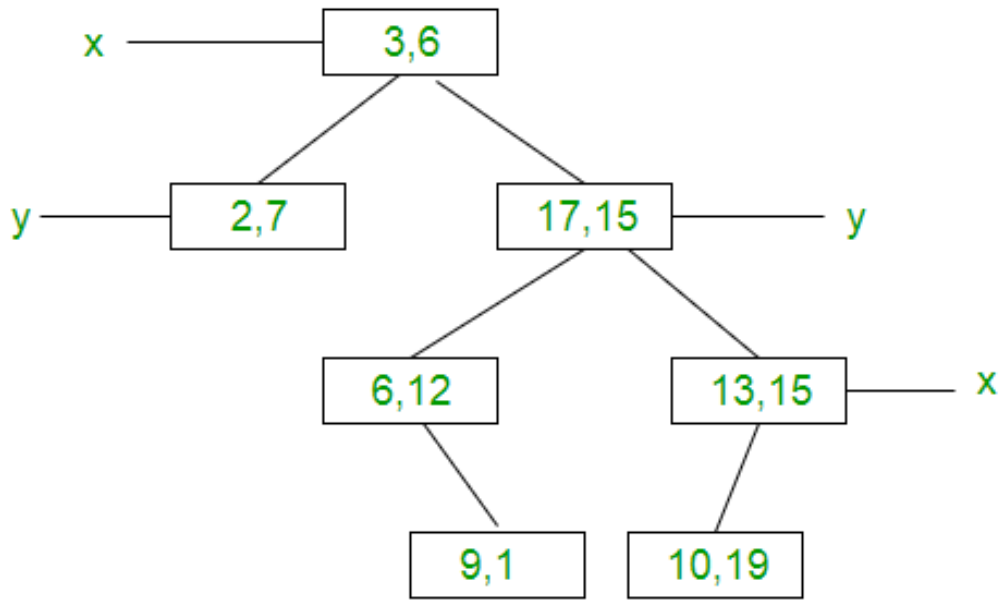
Το πρόγραμμα ακολουθεί αντικειμενοστραφή προσέγγιση και όλες οι δομές αρχικοποιούνται ως αντικείμενα κλάσεων. Στο root directory βρίσκεται ένας φάκελος για την κάθε δομή, με όνομα το όνομα της δομής. Υπάρχει επίσης ένας φάκελος archive που περιέχει το dataset και ένας ακόμη φάκελος functions που περιέχει τις βοηθητικές συναρτήσεις που χρησιμοποιήθηκαν. Εκτός των φακέλων υπάρχει η main.py το οποίο είναι και το κύριο πρόγραμμα. Βάσει του input του χρήστη κατασκευάζει το αντίστοιχο δέντρο, του επιτρέπει να κάνει range query σε αυτό και τέλος, εκτελεί LSH. Για να τρέξει το πρόγραμμα απαιτείται η δομή των directories να μείνει ακέραια.

K-D Tree

Μέρος 1: Εισαγωγή

Ένα KD-Tree (K-Dimensional Tree) είναι ένα δυαδικό δέντρο αναζήτησης, όπου τα δεδομένα σε κάθε κόμβο αναπαριστούν ένα σημείο σε έναν K-διάστατο χώρο. Πρόκειται για μια δομή δεδομένων που χρησιμοποιείται για την οργάνωση σημείων σε έναν πολυδιάστατο χώρο μέσω διαμερισμού του χώρου (space partitioning). Ένας κόμβος που δεν είναι στα φύλλα, ενός KD-Tree διαιρεί τον χώρο σε δύο μέρη, τα οποία ονομάζονται υποχώροι (half-spaces). Τα σημεία που βρίσκονται αριστερά από τον διαχωριστικό άξονα αναπαρίστανται από το αριστερό υποδέντρο, ενώ τα σημεία που βρίσκονται δεξιά αναπαρίστανται από το δεξί υποδέντρο.

Για παράδειγμα, σε ένα δέντρο δύο διαστάσεων (2D Tree), η ρίζα έχει ένα επίπεδο διαχωρισμού που ευθυγραμμίζεται με τον άξονα x. Τα παιδιά της ρίζας έχουν επίπεδα που ευθυγραμμίζονται με τον άξονα y, ενώ τα εγγόνια της επιστρέφουν στον άξονα x, και ούτω καθεξής. Αυτή η εναλλαγή μεταξύ των διαστάσεων συνεχίζεται σε κάθε επίπεδο του δέντρου, διασφαλίζοντας ότι ο χώρος διαιρείται αναδρομικά και ότι κάθε κόμβος περιορίζει τη χωρική περιοχή που πρέπει να ελεγχθεί κατά την αναζήτηση. Με αυτόν τον τρόπο, το KD-Tree αποτελεί ένα αποδοτικό εργαλείο για λειτουργίες όπως η εύρεση του πλησιέστερου γείτονα (nearest neighbor search) και η αναζήτηση σε περιοχές (range search), ιδίως σε χώρους με λίγες διαστάσεις.
(https://www.geeksforgeeks.org/search-and-insertion-in-k-dimensional-tree/)



Πηγή: geeksforgeeks

Μέρος 2: Κατασκευή

Το δέντρο κατασκευάζεται αναδρομικά με την συνάρτηση build_kd_tree η οποία λαμβάνει ως παραμέτρους το DataFrame από το pandas, τα ονόματα των στηλών του DataFrame με βάση τα οποία θα γίνει η κατασκευή και το βάθος στο οποίο βρίσκεται ο κόμβος. Ο κάθε κόμβος δημιουργείται από την κλάση KDTreeNode η οποία περιέχει μια εγγραφή του dataset και τον αριστερό και δεξί κόμβο.

Αρχικά, καθορίζεται ποιος από τους άξονες θα χρησιμοποιηθεί βάση το βάθος του κόμβου. Έπειτα, το dataset ταξινομείται βάση της επιλεγμένης στήλης και επιλέγεται ο διάμεσος. Τέλος "χτίζονται" αναδρομικά το αριστερό και δεξί υποδέντρο λαμβάνοντας ως παράμετρο ό,τι βρίσκεται ως τον διάμεσο και ό,τι βρίσκεται ακριβώς μετά, αντίστοιχα. Η κατασκευή τελειώνει όταν δεν υπάρχουν άλλα στοιχεία μέσα στο dataset μετά από αναδρομική κλήση και η συνάρτηση επιστρέφει τον κόμβο - ρίζα.

Μέρος 3: Αναζήτηση

Με την αναδρομική συνάρτηση range_search μπορούμε να εκτελέσουμε αναζήτηση σε ένα εύρος τιμών των στηλών για τις οποίες έχουμε κατασκευάσει το δέντρο. Η συνάρτηση μπορεί να εκτελέσει και exact match query αν θέσουμε για όλες τις στήλες το ίδιο min – max ($x_{min} = x_{max}$, $y_{min} = y_{max}$, $z_{min} = z_{max}$).

Η συνάρτηση δέχεται ως ορίσματα τον κόμβο και το βάθος το οποίο ξεκινάει η αναζήτηση, τις στήλες στις οποίες έχει κατασκευαστεί το δέντρο και το εύρος τιμών (min_range, max_range). Αρχικά, ελέγχουμε αν οι τιμές των στηλών του κόμβου είναι εντός του καθορισμένου εύρους, αν είναι, τότε προστίθενται στα αποτελέσματα. Αφού ολοκληρωθεί ο έλεγχος αυτός, αναδρομικά ερευνά τα αριστερά και δεξιά υποδέντρα, ανάλογα με το αν το σημείο του κόμβου βρίσκεται εντός του εύρους για την τρέχουσα διάσταση. Η διαδικασία συνεχίζεται μέχρι να εξεταστούν όλοι οι σχετικοί κόμβοι του δέντρου.

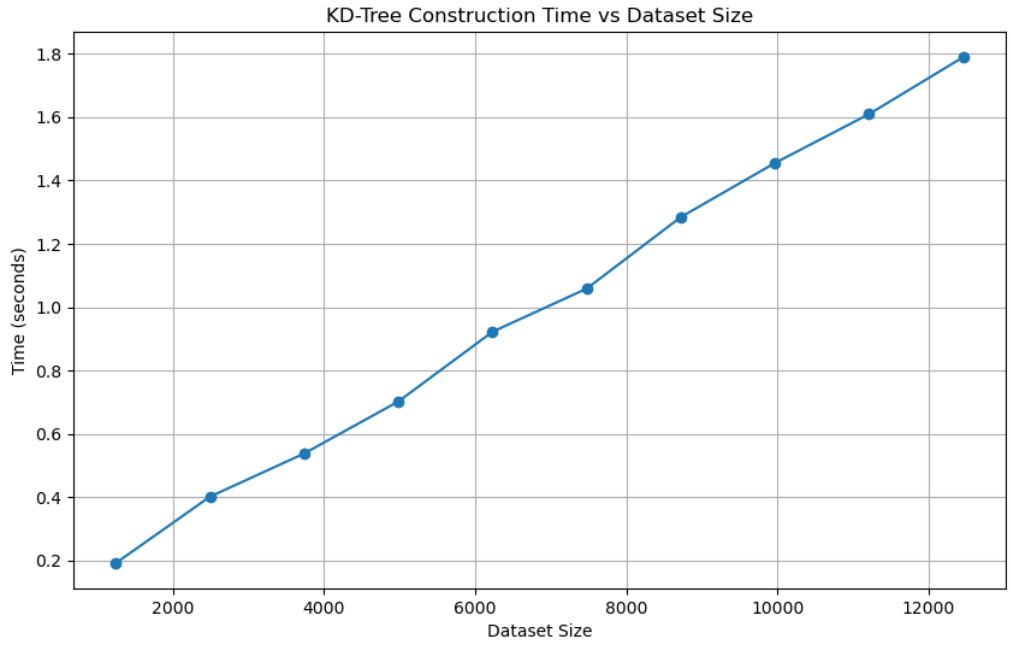
Μέρος 4: Χρονικές Μετρήσεις

Για την κατασκευή του δέντρου η πολυπλοκότητα είναι *O(dnlogn)* όπου d είναι το βάθος του δέντρου. Συμπεραίνουμε το παραπάνω αφού για κάθε αναδρομική κλήση της συνάρτησης ταξινομούμε τα δεδομένα βάση της επιλεγμένης διάστασης (*O(nlogn)*) και καθορίζονται οι τιμές για το αριστερό και δεξί υποδέντρο βάση του διάμεσου (O(n)). Επομένως, η πολυπλοκότητα χειρότερης περίπτωσης είναι *nlogn* επί το βάθος d του δέντρου, δηλαδή *dnlogn*.

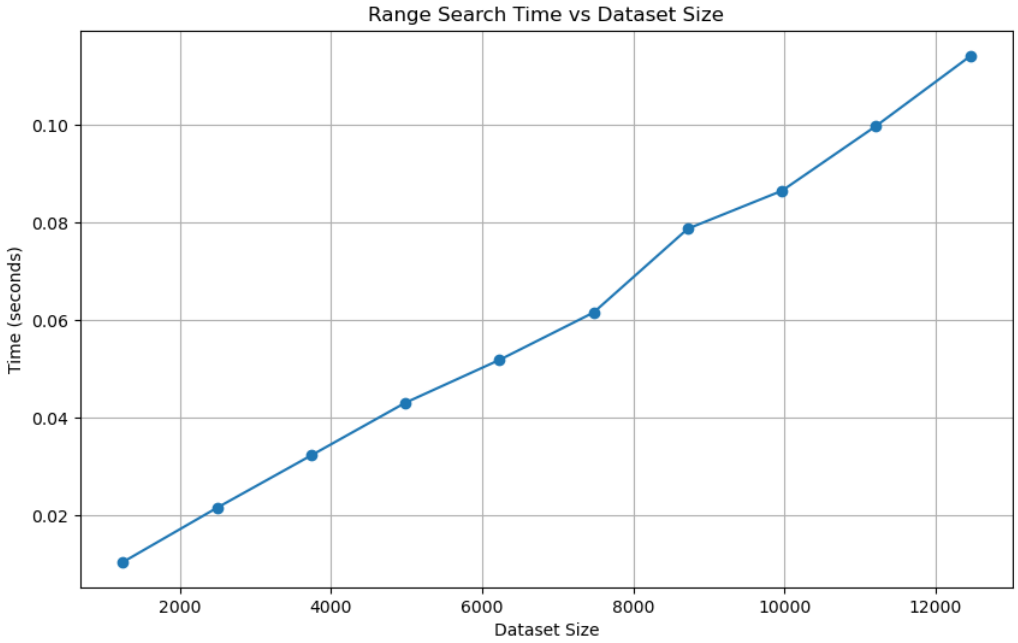
Για την αναζήτηση εύρους στη χειρότερη περίπτωση, μπορεί να χρειαστεί να επισκεφτούμε όλους τους κόμβους, δίνοντας πολυπλοκότητα *O(n)*. Στη μέση περίπτωση, εάν το k-d tree είναι ισοζυγισμένο και το ερώτημα καλύπτει μικρό εύρος η πολυπλοκότητα γίνεται $O(n^{1-1/k} + R)$ όπου *k* ο αριθμός των διαστάσεων που εξετάζουμε και *R* το πλήθος των αποτελεσμάτων του query. Ο όρος $n^{1-1/k}$ δείχνει το αντίκτυπο του πλήθος διαστάσεων στην πολυπλοκότητα, όσο ανεβαίνει ο αριθμός αυτός τόσο η πολυπλοκότητα πλησιάζει το *O(n+k)*.

Για το test δημιουργήθηκαν αντίγραφα του dataset, που για μέγεθος από το μέγεθος του dataset μέχρι δέκα φορές αυτό, έγινε η κατασκευή του δέντρου και έπειτα εξαντλητική αναζήτηση που

επέστρεφε όλες τις τιμές. Οι χρόνοι που μετρήθηκαν φαίνονται παρακάτω.



Εργαλείο: matplotlib

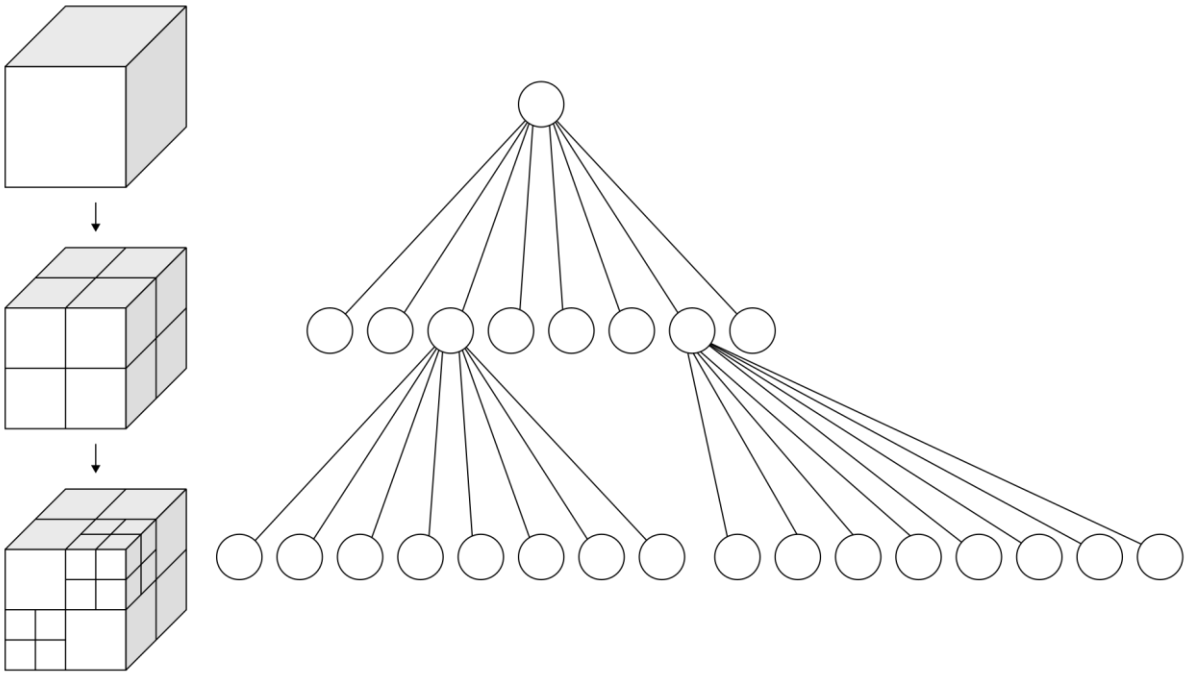


Εργαλείο: matplotlib

Quadtree

Μέρος 1: Εισαγωγή

Τα Quadtrees είναι δέντρα που διαμερίζουν το χώρο σε τέσσερα τεταρτημόρια αλλά στην περίπτωσή μας που οι διαστάσεις είναι τρείς, ο χώρος διαμερίζεται σε οκτώ κύβους. Το δέντρο λοιπόν πρόκειται για octree. Η αναζήτηση στα δέντρα αυτά έγκειται στην αναζήτηση του κύβου που περιέχει το σημείο γεωμετρικά. Η εισαγωγή στοιχείου επιτυγχάνεται με τον ίδιο τρόπο αλλά κάθε φορά που ένας κύβος φτάσει να έχει k (στην τρέχουσα υλοποίηση k = 1) στοιχεία μέσω διαδοχικών inserts, τότε διαμερίζεται σε οκτώ κύβους-παιδιά και το στοιχείο εισάγεται αναδρομικά στο κατάλληλο παιδί. Η δομή του octree βασίζεται σε επέκταση υλοποίησης quadtree των GeeksforGeeks. (https://www.geeksforgeeks.org/quad-tree/)



Left: Recursive subdivision of a cube into octants. Right: The corresponding octree.

Πηγή: Wikipedia

Πρώτο δομικό στοιχείο του octree είναι το σημείο στο χώρο που αντιπροσωπεύει η κλάση Point. Το σημείο δέχεται κατά την αρχικοποίησή του τρείς συνιστώσες x, y και z. Δεύτερο δομικό στοιχείο είναι ο κόμβος του δέντρου που αντιπροσωπεύει η κλάση Node. Αυτό δέχεται σαν ορίσματα ένα σημείο και δεδομένα. Τελευταίο δομικό στοιχείο του octree είναι το ίδιο το δέντρο το οποίο αντιπροσωπεύει η κλάση Octree. Δέχεται ως ορίσματα τα δύο σημεία (επάνω-μπροστά-αριστερά και κάτω-πίσω-δεξιά) που βρίσκονται πάνω στη διαγώνιο του κύβου ο οποίος συμβολίζει το χώρο αναζήτησης και τον οριοθετούν.

Μέρος 2: Κατασκευή

Το δέντρο κατασκευάζεται μέσω διαδοχικών inserts. Το insert επιτυγχάνεται μέσω κλήσης της ομώνυμης μεθόδου ή οποία τρέχει αναδρομικά. Δέχεται ως ορίσματα έναν κόμβο και τις διαστάσεις ενός κύβου και αν ο κόμβος είναι κενός ή δεν περιέχεται στις διαστάσεις του κύβου, το οποίο ελέγχεται μέσω της μεθόδου isBoundary.py, η συνάρτηση επιστρέφει. Στη συνέχεια ελέγχει αν ο τρέχων κύβος έχει δική του ρίζα (ο κάθε κύβος μπορεί να αντιμετωπιστεί ως ξεχωριστό υποδέντρο) και αν όχι, εισαγάγει τον κόμβο ως ρίζα. Αν έχει ρίζα, τότε αποφασίζει σε ποιο παιδί θα εισαχθεί ο κόμβος με κλήση της μεθόδου child_index.py. Ο κύβος τότε διαμερίζεται σε 8 υποκύβους-παιδιά και με βάση τις συντεταγμένες του κόμβου, επιλέγεται ένα παιδί και ο κόμβος εισάγεται αναδρομικά με κλήση της insert.

Μέρος 3: Αναζήτηση

Το δέντρο διαθέτει μία μόνο μέθοδο range_query.py η οποία δέχεται ως όρισμα έναν κύβο και αναζητεί όλους τους κόμβους που περιέχονται σε αυτόν. Παρά το όνομά της είναι προσαρμοσμένη να τρέχει και exact match queries ή και queries που δε χρησιμοποιούν όλες τις διαστάσεις. Αν κάποια διάσταση πρέπει να αγνοηθεί, μπορούμε απλώς να κάνουμε τη μέθοδο να αναζητήσει σε ολόκληρο το εύρος της διάστασης αυτής, κάτι που ισοδυναμεί με το να αγνοήσουμε τη διάσταση. Αν θέλουμε επίσης να κάνουμε exact match query, μπορούμε να ορίσουμε ίσα άνω και κάτω φράγματα στις διαστάσεις του κύβου ($x_{min} = x_{max}$, $y_{min} = y_{max}$, $z_{min} = z_{max}$). Αρχικά η range_query.py αρχικοποιεί ένα κενό σύνολο στο οποίο θα εισαχθεί το αποτέλεσμα και ελέγχει αν ο τρέχων κύβος επικαλύπτεται με την περιοχή αναζήτησης. Αν δεν επικαλύπτεται επιστρέφει αμέσως, ενώ αν επικαλύπτεται, ελέγχει αν ο τρέχων κύβος έχει ρίζα. Αν έχει ρίζα, ελέγχει αν περιέχεται στην περιοχή αναζήτησης και αν ναι, την εισάγει στο αποτέλεσμα. Στη συνέχεια, η μέθοδος καλείται αναδρομικά για όλους τους υποκύβους-παιδιά και επιστρέφει το τελικό αποτέλεσμα.

Μέρος 4: Χρονικές Μετρήσεις

Η κατασκευή του δέντρου αποτελείται από διαδοχικά inserts τα οποία στην καλύτερη και στη μέση περίπτωση που τα δεδομένα είναι ομοιόμορφα ή σχεδόν ομοιόμορφα κατανεμημένα, αναγκάζουν το δέντρο να κάνει διαπεράσεις από τη ρίζα προς τα φύλλα σε χρόνο $\log_8(n)$. Συνολικά λαμβάνουν χώρα n τέτοιες διαπεράσεις και ο συνολικός χρόνος κατασκευής είναι $n\log_8(n) = O(n\log(n))$. Στη χειρότερη, τα σημεία είναι συγγραμμικά και ο χρόνος γίνεται $O(n^2)$. Όσο για τα range queries, στην καλύτερη περίπτωση σε μικρή αναζήτηση που προσπελάζει μόνο ένα octant ο χρόνος είναι ο χρόνος προσπέλασης του octant, δηλαδή O(log(n)). Στη μέση περίπτωση που προσπελάζονται m octants ο χρόνος ανεβαίνει σε O(m + k) που στη δική μας περίπτωση για k = 1 είναι O(m) και στη χειρότερη περίπτωση που πρέπει να επιστραφούν όλα τα σημεία, ο χρόνος γίνεται γραμμικός, δηλαδή O(n).

Για το test δημιουργήθηκαν αντίγραφα του dataset, που για μέγεθος από το μέγεθος του dataset μέχρι δέκα φορές αυτό, έγινε η κατασκευή του δέντρου και έπειτα εξαντλητική αναζήτηση που επέστρεφε όλες τις τιμές. Οι χρόνοι που μετρήθηκαν φαίνονται παρακάτω.

Εργαλείο: MATLAB

Εργαλείο: MATLAB

Range Tree

Μέρος 1: Εισαγωγή

Το Range Tree είναι μια εξειδικευμένη δομή δεδομένων που βασίζεται στη λογική του Binary Search Tree (BST) και έχει σχεδιαστεί για την αποδοτική επεξεργασία πολυδιάστατων ερωτημάτων εύρους. Κάθε επίπεδο του Range Tree αντιπροσωπεύει μία διάσταση του δεδομένου χώρου, ενώ η δομή επιτρέπει την αποθήκευση και αναζήτηση δεδομένων με πολυπλοκότητα που είναι υπολογιστικά βέλτιστη για πολλά σενάρια χρήσης.

Στο μονοδιάστατο επίπεδο, ένα Range Tree λειτουργεί όπως ένα παραδοσιακό BST, ταξινομώντας τα δεδομένα και επιτρέποντας ερωτήματα εύρους σε χρόνο O(logn+k), όπου n είναι το πλήθος των σημείων και k το πλήθος των αποτελεσμάτων. Σε πολυδιάστατους χώρους, όπως ο τρισδιάστατος χώρος (3D), η δομή επεκτείνεται μέσω μιας ιεραρχικής οργάνωσης:

- Κύρια δομή: Αντιστοιχεί στη διάσταση x και λειτουργεί ως το βασικό BST.
- Δευτερεύουσα δομή: Ενσωματώνεται σε κάθε κόμβο της κύριας δομής για τη διάσταση y.
- Τριτεύουσα δομή: Ενσωματώνεται σε κάθε κόμβο της δευτερεύουσας δομής για τη διάσταση z.

Αυτή η ιεραρχία εξασφαλίζει ότι κάθε επίπεδο επεξεργάζεται μόνο τη δική του διάσταση, επιτρέποντας αποδοτική αναζήτηση. Η εφαρμογή των Range Trees είναι ευρεία, καλύπτοντας περιοχές όπως η γεωμετρική αναζήτηση, η επεξεργασία χωρικών δεδομένων και οι βάσεις δεδομένων που περιλαμβάνουν πολυδιάστατα ερωτήματα.

Μέρος 2: Κατασκευή

Η κατασκευή του 3D Range Tree στον κώδικα υλοποιείται μέσω μιας ταξινομημένης προσέγγισης. Ακολουθεί η διαδικασία, με παραπομπές στις αντίστοιχες μεθόδους του αρχείου rangetree.py:

- Κύρια δομή (x-dimension):

Τα σημεία ταξινομούνται με βάση την x-συντεταγμένη (γραμμή 7: `sorted(points, key=lambda p: p[0])`). Ο κόμβος του δέντρου αντιπροσωπεύει το μεσαίο σημείο της ταξινομημένης λίστας, και το δέντρο χτίζεται αναδρομικά για τις αριστερές και δεξιές υπολίστες μέσω της μεθόδου build_primary_tree (γραμμές 12-25).

- Δευτερεύον δέντρο (y-dimension):

Σε κάθε κόμβο της κύριας δομής, κατασκευάζεται ένα δευτερεύον range tree χρησιμοποιώντας τα ίδια σημεία, ταξινομημένα όμως με βάση την y-συντεταγμένη (γραμμές 48-62).

- Τριτεύον δέντρο (z-dimension):

Σε κάθε κόμβο του δευτερεύοντος δέντρου, δημιουργείται ένα τριτεύον range tree, όπου τα σημεία ταξινομούνται κατά την z-συντεταγμένη (γραμμή 79: `self.points = sorted(points, key=lambda p: p[2])`).

Η διαδικασία αυτή διασφαλίζει ότι κάθε επίπεδο του δέντρου μπορεί να επεξεργαστεί αποδοτικά ερωτήματα εύρους για την αντίστοιχη διάσταση.

Μέρος 3: Αναζήτηση

Η επεξεργασία ερωτημάτων εύρους υλοποιείται μέσω αναδρομικής αναζήτησης σε κάθε επίπεδο του δέντρου. Η υλοποίηση περιλαμβάνεται στις μεθόδους range_query και _query_primary για την κύρια δομή, range_query και _query_secondary για το δευτερεύον δέντρο, καθώς και range_query για το τριτεύον δέντρο:

Κύρια Αναζήτηση (x-dimension):

Ξεκινά από τη ρίζα και ελέγχει αν η x-συντεταγμένη του κόμβου βρίσκεται εντός του εύρους μέσω της μεθόδου _query_primary (γραμμές 27-43).

Αν ναι, εκτελείται αναζήτηση στο δευτερεύον δέντρο του κόμβου μέσω της μεθόδου secondary_tree.range_query (γραμμή 35).

Στη συνέχεια, η αναζήτηση επεκτείνεται στους αριστερούς και δεξιούς υποκόμβους ανάλογα με το εύρος.

Δευτερεύουσα Αναζήτηση (y-dimension):

Για κάθε κόμβο που περνά τον έλεγχο της κύριας δομής, η αναζήτηση συνεχίζεται στο δευτερεύον δέντρο, όπου ελέγχεται αν η y-συντεταγμένη βρίσκεται εντός του εύρους μέσω της μεθόδου _query_secondary (γραμμές 64-80).

Αν ναι, εκτελείται αναζήτηση στο τριτεύον δέντρο μέσω της μεθόδου tertiary_tree.range_query (γραμμή 72).

Τριτεύουσα Αναζήτηση (z-dimension):

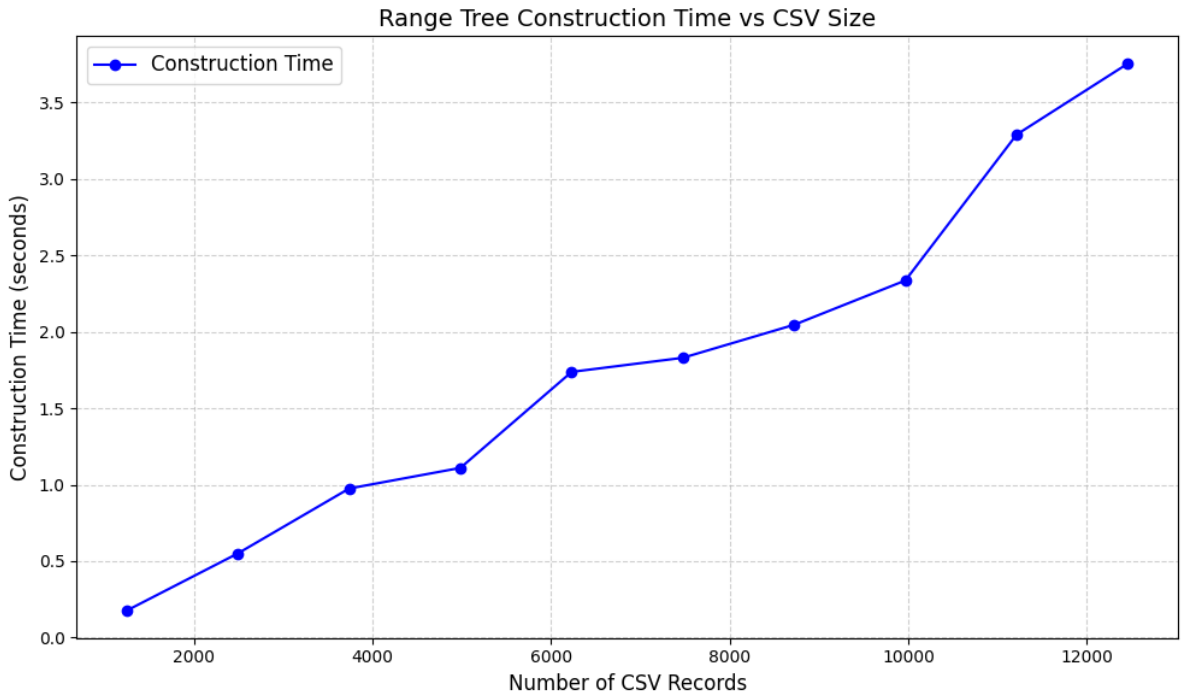
Στο τριτεύον δέντρο, τα σημεία ελέγχονται για την z-συντεταγμένη μέσω της μεθόδου range_query (γραμμές 82-86), και τα κατάλληλα σημεία επιστρέφονται ως αποτέλεσμα.

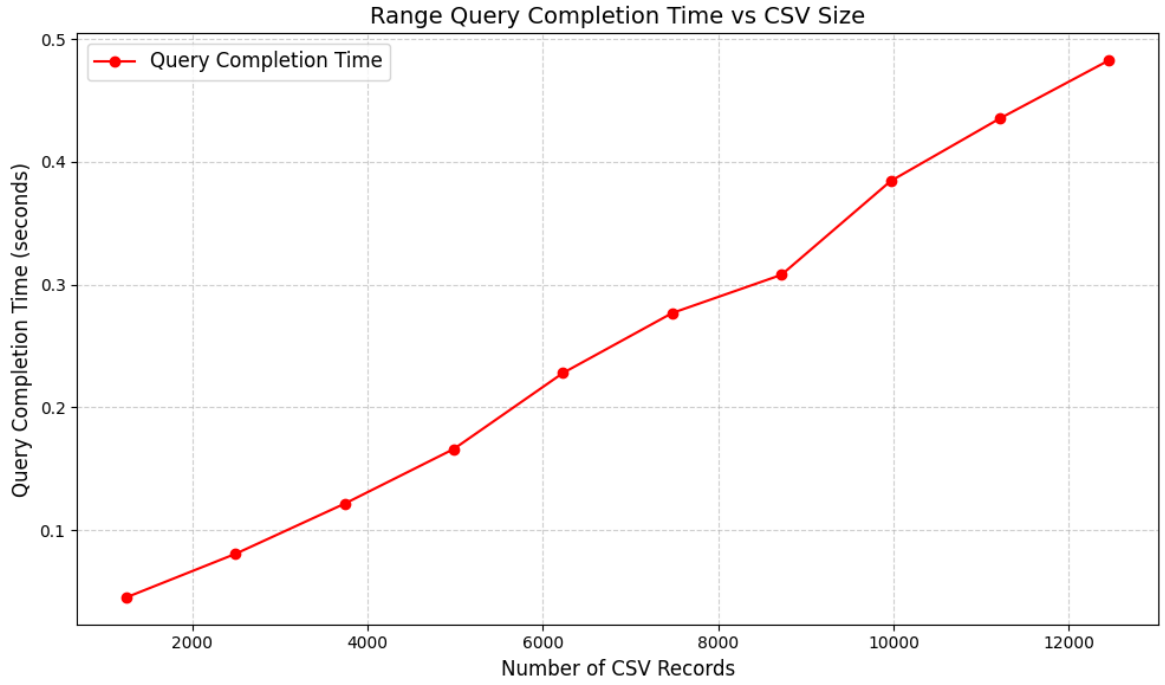
Η διαδικασία είναι βελτιστοποιημένη, καθώς χρησιμοποιεί τοπικές αναζητήσεις σε κάθε διάσταση, αποφεύγοντας την πλήρη επεξεργασία του δέντρου.

Μέρος 4: Χρονικές Μετρήσεις

Η συνολική πολυπλοκότητα κατασκευής ενός Range Tree d-διαστάσεων είναι O(nlog d−1n), ενώ η πολυπλοκότητα επεξεργασίας ενός ερωτήματος είναι O(log dn+k).

Για το test δημιουργήθηκαν αντίγραφα του dataset, που για μέγεθος από το μέγεθος του dataset μέχρι δέκα φορές αυτό, έγινε η κατασκευή του δέντρου και έπειτα εξαντλητική αναζήτηση που επέστρεφε όλες τις τιμές. Οι χρόνοι που μετρήθηκαν φαίνονται παρακάτω.


Range Tree Construction Time vs CSV Size
Construction Time
Construction Time (seconds)
Number of CSV Records
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
2000
4000
6000
8000
10000
12000

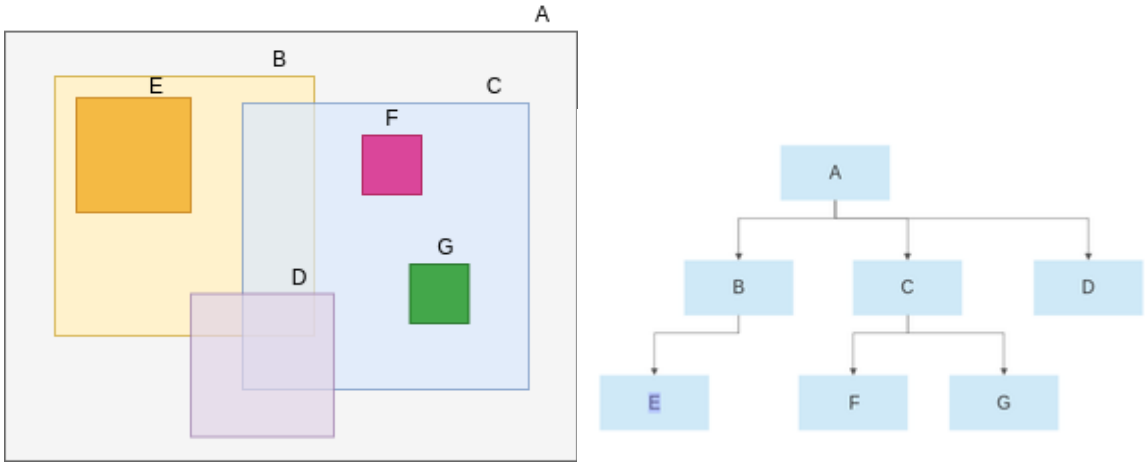

Range Query Completion Time vs CSV Size
Query Completion Time
Query Completion Time (seconds)
Number of CSV Records
0.1
0.2
0.3
0.4
0.5
2000
4000
6000
8000
10000
12000

R-Tree

Μέρος 1: Εισαγωγή

Τα R-Trees αποτελούν δενδρική δομή για την αποτελεσματική αποθήκευση χωρικών δεδομένων σε τουλάχιστον δύο διαστάσεις. Το δέντρο αποτελείται από μία ρίζα, ενδιάμεσους κόμβους και κόμβους φύλλα. Η ρίζα αποτελεί τον κόμβο που καλύπτει χωρικά όλο το δέντρο , ενώ κάθε κόμβος γονέας περιέχει δείκτες προς τους κόμβους παιδιά του. Κύριο χαρακτηριστικό των μελών του δέντρου είναι το MBR ή Minimum Bounding Region, που περιγράφει την περιοχή που καλύπτει το μέλος αυτό, ιδιότητα που είναι κρίσιμη για τη διμιουργία του δέντρου αλλά και την αναζήτηση κόμβου σε αυτό, καθώς ο αλγόριθμος στηρίζεται στην δημιουργία παραλληλογράμμων που περιέχουν τα παραλληλόγραμμα-παιδιά τους και στην προσθήκη νέου μέλους αποφασίζεται ο πατέρας βάση της ελάχιστης επέκτασης του παραλληλογράμου των υποψηφίων ώστε να καλύπτεται πλήρως. Για την υλοποίηση της άσκησης η δομή του R Tree έχει επαυξηθεί από δύο σε τρείς διαστάσεις πράγμα που συνεπάγεται την μετάβαση από παραλληλόγραμμα σε κύβους.

(Πηγή: https://www.geeksforgeeks.org/introduction-to-r-tree/)



Μέρος 2: Κατασκευή

Για να οριστεί ένα MBR σε τρείς διαστάσεις χρειάζονται έξι τιμές, για το ελάχιστο και το μέγιστο σε κάθε άξονα. Έτσι κάθε εγγραφή του συνόλου δεδομένων ορίζεται ως ένα σημείο στον τρισδιάστατο χώρο βάση των τριών τιμών *review_date, rating, 100g_USD,* έχει δηλαδή xmin = xmax, ymin =ymax, zmin = zmax. Η δομή και οι λειτουργίες του δέντρου στηρίζονται πάνω στην κλάση Node όπου πρώτο στιγμιότυπο είναι το root, πάνω σε αυτό καλείται η μέθοδος insert για κάθε εγγραφή στη συνάρτηση build_r_tree. Επίσης ως παραδοχή έχρι οριστεί ότι κάθε κόμβος μπορεί να περιέχει μέχρι 2 γονείς ή μέχρι 4 φύλλα.

Αναλύοντας περαιτέρω την μέθοδο insert, αρχικά γίνεται εισαγωγή στα μέλη-παιδιά του κόμβου root έτσι μέχρι ο αριθμός τους να φτάσει 4 προσθέτονται πολύ απλά ως μέλη και ξαναυπολογίζεται το MBR του γονέα ώστε ο κύβος του να περιέχει όλα τα νέα σημεία.

Στην εισαγωγή πέμπτου στοιχείου ως μέλος προκύπτει η ανάγκη για Quadratic Split , η λογική του οποίου έχει παρθεί από τις διαφάνειες του μαθήματος, έτσι προστίθεται και το πέμπτο μέλος και

στη συνέχεια καλείται η μέθοδος quadraticSplit του γονέα. Η μέθοδος αυτή αναλαμβάνει να υπολογίσει ποια από τα μέλη του γονέα βρίσκονται πιο μακριά στο χώρο και στη συνέχεια χρησιμοποιεί το ζεύγος αυτό ως seed για τα δύο νέα παιδιά γονείς που έχουν πλέον και τις ιδιότητες isgroup ενώ ο γονέας αποκτά την ιδιότητα hasgroup. Στη συνέχεια για κάθε από τα υπολειπόμενα μέλη κρίνεται ποιο από τα δύο νέα Nodes που δημιουργήθηκαν χρειάζεται τη μικρότερη επέκταση στο χώρο για να τα περιέχει και εισάγεται σε αυτό το γκρουπ.

Στην επόμενη κλήση της insert στο root θα εκτελεσεί το μέρος κώδικα όταν το Node έχει την ιδιότητα hasGroup. Ο κώδικας θα μεταβεί από όλους τους κόμβους που έχουν και αυτοί την ίδια ιδιότητα ανανεώνοντας το MBR τους με την mbrCompare και κρίνοντας ποιο από τα παιδιά τους θα είναι το επόμενο βάση της leastExpansionGroup. Όταν πλέον φτάσει σε Node που δεν περιέχει groups τότε θα κληθεί insert σε αυτό το Node.

Η λογική αυτή επαναλαμβάνεται καθώς το δέντρο μεγαλώνει. Επίσης δικλείδα ασφαλείας ώστε το δέντρο να μη μεγαλώσει άνισα είναι η σύγκριση του όγκου των group στην συνάρτηση leastExpansionGroup που αν το πρώτο group έχει όγκο μεγαλύτερο του μηδενός και το δεύτερο είναι μηδέν τότε επιλέγεται το δεύτερο. Αν δεν υπήρχε αυτή η λεπτομέρεια με την πρώτη εισαγωγή τα μεγάλωνε μόνο το ένα group το οποίο και στην επόμενη θα χρειαζόταν την ελάχιστη μεγέθυνση για να περιέχει οποιοδήποτε σημείο στο χώρο , άρα η άλλη πλευρά δεν θα μεγάλωνε ποτέ.

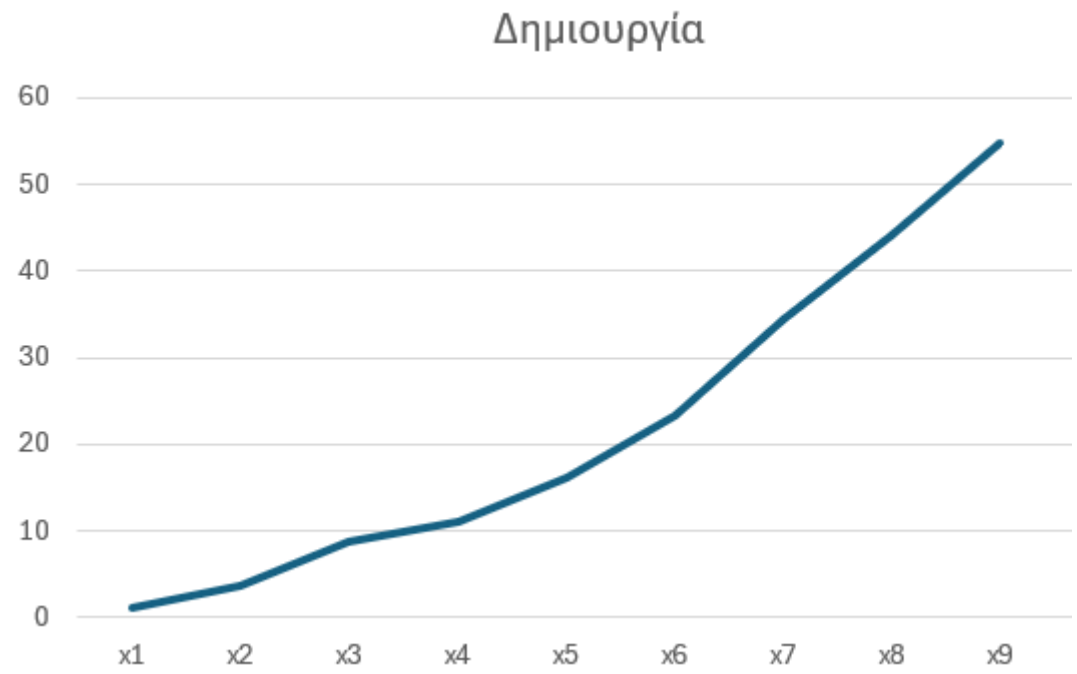
Μέρος 3: Αναζήτηση

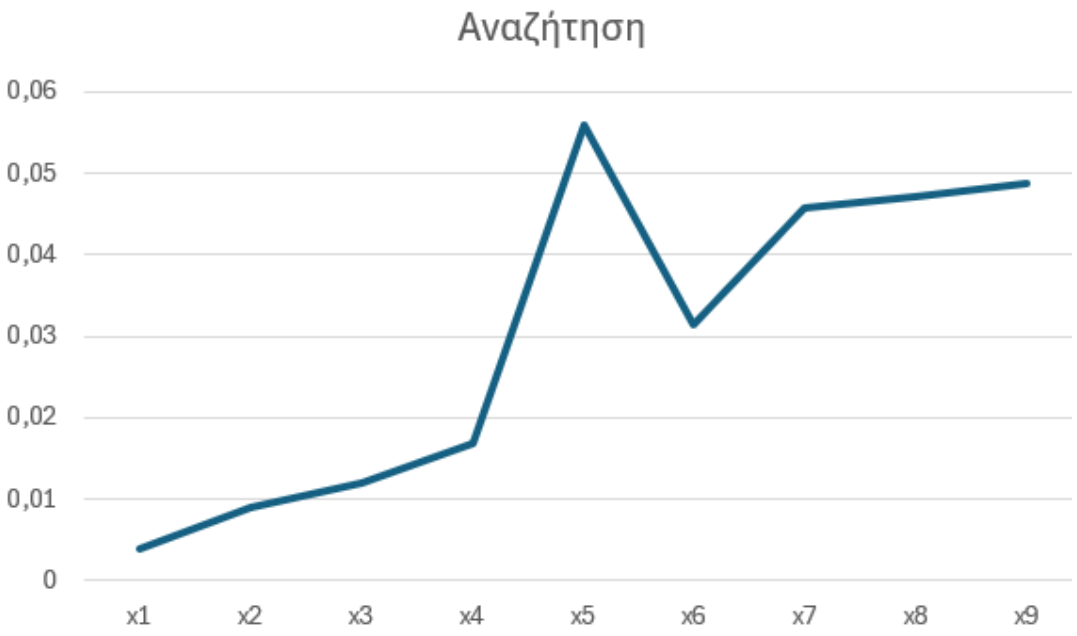
Για την αναζήτηση καλείται η συνάρτηση search πάνω στα μέλη του κόμβου root. Μέσα σε αυτή τη συνάρτηση περιέχεται και η υποσυνάρτηση containMbr που αναλαμβάνει τη σύγκριση των mbr είτε για groups είτε για leafs, αν το Node που συγκρίνεται είναι γκρουπ τότε θα πρέπει να περιέχει τον κύβο αναζήτησης ενώ αν είναι φύλλο τότε θα πρέπει η αναζήτηση να περιέχει το Node.

Η πρώτη εκτέλεση της συνάρτησης γίνεται πάνω στα μέλη του root όπου σε ένα πλήρες δέντρο θα έχει παιδιά που έχουν την ιδιότητα hasgroup οπότε αν η containMbr επιστρέψει true τότε θα μεταβεί ο κώδικας στην αναδρομική κλήση της συνάρτησης στα μέλη μέλη αυτό του Node μέχρι θα βρεθεί φύλλο που πληροί τις συνθήκες της containMbr ώστε να προστεθεί στο nodes array. Η έξοδος από τις αναδρομικές κλήσεις επιστρέφει στην αρχική που θα περιέχει πλέον όλα τα Nodes φύλλα που ικανοποιούν την αναζήτηση τα οποία και επιστρέφει.

Μέρος 4: Χρονικές Μετρήσεις

Η πολυπλοκότητα χειρότερης περίπτωσης του R Tree ειναι O(n) για τη δημιουργία αλλα και την αναζήτηση, ο αλγόριθμος εκτελέστηκε για σταδιακή αύξηση του dataset και παρακάτω δίνονται τα αποτελέσματα σε γράφημα.


Δημιουργία
60
50
40
30
20
10
0
x1
x2
x3
x4
x5
x6
x7
x8
x9


Αναζήτηση
0,06
0,05
0,04
0,03
0,02
0,01
0
x1
x2
x3
x4
x5
x6
x7
x8
x9

LSH

Μέρος 1: Εισαγωγή

Ο κώδικας για το lsh χωρίζεται σε δύο βασικά μέρη, στις συναρτήσεις που περιέχονται στον φάκελο lsh, και στον κύριο κώδικα που δέχεται σαν είσοδο τα αποτελέσματα των range querries από τα δέντρα. Ο κώδικας είναι ίδιος για κάθε δέντρο, οπότε δεν εξαρτάται από ποιο δέντρο χρησιμοποιείται στην κάθε περίπτωση. Επιπλέον, ο κώδικας είναι σύμφωνος με τις διαφάνειες του μαθήματος και βασίζεται πάνω στον κώδικα της σελίδας:
https://www.codemotion.com/magazine/backend/fast-document-similarity-in-python-minhashlsh/
και του αντίστοιχου Github Repository:
https://github.com/nicoDs96/Document-Similarity-using-Python-and-PySpark/blob/master/LSH/DM_HW2_Ex2.ipynb

Μέρος 2: Περιγραφή κώδικα

Ο κώδικας ξεκινάει με την φόρτωση των δεδομένων από τα αποτελέσματα των δέντρων και τον υπολογισμών των shingles για το review της κάθε εγγραφής με την συνάρτηση *get_shingles()*. Τα shingles έχουν μέγεθος *shingling_size* και όταν παραχθούν, περνάνε από την συνάρτηση *get_hashed_shingles()* ώστε να μετατραπούν σε αριθμητικές τιμές που αντιπροσωπεύουν τα shingles της κάθε εγγραφής (μέσω της συνάρτησης κατακερματισμού *hashFamily*).

Στην συνέχεια, με την κλήση της κλάσης *minhashSinger*, δημιουργείται η υπογραφή της κάθε εγγραφής, δηλαδή ένας μικρότερος πίνακα τιμών, ο οποίος όμως διατηρεί την ομοιότητα με τις υπόλοιπες εγγραφές. Αυτό επιτυγχάνεται με την κλήση της *compute_signature_matrix()* που εφαρμόζει την διαδικασία του MinHash. Το MinHash εφαρμόζει πολλές συναρτήσεις hash (πλήθους *sig_size*) και αποθηκεύει την μικρότερη τιμή (min hash) που προκύπτει. Η διαδικασία επαναλαμβάνεται για κάθε συνάρτηση hash και το τελικό αποτέλεσμα είναι μια λίστα με τις ελάχιστες τιμές (min hashes) που αντιστοιχούν σε κάθε εγγραφή. Επίσης, οι υπογραφές αυτές αποθηκεύονται και στον πίνακα *signature_matrix*. Ωστόσο, λόγο της τυχαιότητας των συναρτήσεων hash που χρησιμοποιούνται, δεν διατηρείται όλη η πληροφορία. Αυτό σημαίνει ότι η ομοιότητα της κάθε εγγραφής με τις υπόλοιπες αλλάζει, αλλά συνήθως όχι σημαντικά.

Το επόμενο βήμα είναι το lsh. Αρχικά, το lsh χωρίζει το signature matrix σε bands πλήθους *bands_nr* με την συνάρτηση *get_signature_matrix_bands()*. Μετά, για κάθε αντίστοιχο band των εγγραφών, υπολογίζεται το hash τους και αυτά που έχουν την ίδια τιμή οδηγούνται στο ίδιο bucket (*get_band_buckets()*). Οι εγγραφές στο ίδιο bucket είναι υποψήφια similar pairs και εξετάζονται στην συνέχεια. Στα buckets, δημιουργούνται ζεύγη εγγράφων (που έχουν το ίδιο hash) και γίνεται έλεγχος της ομοιότητάς τους με την Jaccard Similarity. Αν η ομοιότητα ξεπερνάει την τιμή που ορίζεται από τον χρήστη (είναι η μεταβλητή *user_defined_threshold* και ορίζεται στην γραμμή 6 του αρχείου lsh.py στον φάκελο του lsh), τότε το ζεύγος θεωρείται παρόμοιο και προστίθεται στο σύνολο των παρόμοιων εγγράφων. Όλη η διαδικασία που περιεγράφηκε γίνεται στο πλαίσιο της συνάρτησης *get_similar_items*.

Μετά από την διαδικασία του lsh, υπολογίζεται το Jackard Similarity των παρόμοιων εγγραφών και γίνεται sorting με φθίνουσα σειρά ομοιότητας για να δει ο χρήστης τις N πιο όμοιες εγγραφές.

Αυτή είναι η βασική υλοποίηση του lsh. Παρακάτω, ακολουθεί ένας έλεγχος της ποιότητας (quality) του lsh. Σε τετραγωνικό χρόνο (O(n^2)), υπολογίζεται το Jackard Similarity όλων των εγγραφών μεταξύ τους και προβάλλονται πάλι τα N πιο όμοια με φθίνουσα σειρά ομοιότητας. Έτσι μπορούμε να εξετάσουμε τα N πιο όμοια του lsh και τα N πραγματικά πιο όμοια. Ωστόσο, δεν εξετάζετε η σειρά αλλά μόνο αν το ζεύγος εγγραφών ανήκει και στα δύο σύνολα. Δηλαδή αν το lsh εμφανίζει τα ζεύγη {(3,6), (1,5), (2,9)} με αυτήν την σειρά, και το πραγματικό εμφανίζει {(3,6), (2,9), (1,5)}, τότε το quality θεωρείται 100%.

Μέρος 3: Σημαντικές μεταβλητές

Στο δεύτερο μέρος της αναφοράς, αναφέρθηκαν αρκετές μεταβλητές όπως είναι το πλήθος των bands (*bands_nr*), το μέγεθος των shingles (*shingling_size*), το πλήθος των hash συναρτήσεων για το MinHash (*sig_size*) κ.λπ. Όλες αυτές οι μεταβλητές μπορούν να επεξεργαστούν από τον χρήστη για να αλλάξει αντίστοιχα το quality του lsh. Για τις πειραματικές μετρήσεις που θα ακολουθήσουν, θα γίνει έμφαση στις δύο πιο σημαντικές μεταβλητές, δηλαδή στο *user_defined_threshold* για το πότε δύο εγγραφές θεωρούνται όμοιες, καθώς και στο *shingling_size*.

Μέρος 4: Πειραματικά αποτελέσματα

Για μία πρώτη μέτρηση, θα χρησιμοποιηθεί το δέντρο Quadtree (δεν έχει σημασία ποιο δέντρο χρησιμοποιείται) και το lsh θα έχει *user_defined_threshold* = 0.5 και *shingling_size* = 3. Επίσης, για το query του δέντρου θα χρησιμοποιηθούν οι μέγιστες διαστάσεις, δηλαδή τα δεδομένα δεν θα περιοριστούν καθόλου και το Lsh θα εξετάσει ολόκληρο το csv.

```
Define the box for the range query:
Enter min review_date as YYYYMM: 201711
Enter max review_date as YYYYMM: 202211
Enter min rating: 84
Enter max rating: 97
Enter min 100g_USD: 1
Enter max 100g_USD: 130
Time taken for query completion: 0.00200653076171875 seconds.
```

Στην συνέχεια εμφανίζεται στον χρήστη αν θέλει να προχωρήσει στο lsh.

```
Would you like to run the LSH phase of the query? (no for exit):
```

Αν ο χρήστης δεν επιλέξει ‘no’, τότε το lsh εκτυπώνει τα εξής:

```
Would you like to run the LSH phase of the query? (no for exit):
Shingles produced in:    0.33 seconds.
Signature Matrix computed in:    10.36 seconds.
LSH Similarity computed in:  0.17 seconds.
Similar Elements Found: 75
Enter the number of top similar pairs to display:
```

Δηλαδή χρειάζεται περίπου 11 δευτερόλεπτα και βρίσκει 75 εγγραφές με ομοιότητα μεγαλύτερη του *user_defined_threshold*. Αν ο χρήστης εισάγει την τιμή 10, βλέπει τις 10 πιο όμοιες εγγραφές σύμφωνα με το Jackard Similarity τους.

```
review        Richly sweet, floral-toned. Magnolia, almond, ...
doc_id                                                      676
Name: 676, dtype: object
Jaccard Similarity: 0.7073

Pair 20:
Document 1:
name                                              Monte Alban
roaster                                    JBC Coffee Roasters
roast                                            Medium-Light
loc_country                                     United States
origin                                                 Mexico
100g_USD                                                 5.88
rating                                                     92
review_date                                            202010
review        Crisply sweet, nut-toned. Almond brittle, tart...
doc_id                                                     454
Name: 454, dtype: object

Document 2:
name                                           Finca Patzibir
roaster                                         El Gran Cafe
roast                                           Medium-Light
loc_country                                        Guatemala
origin                                             Guatemala
100g_USD                                                 4.7
rating                                                     92
review_date                                            202211
review        Crisply sweet, nut-toned. Almond brittle, pie ...
doc_id                                                     610
Name: 610, dtype: object
Jaccard Similarity: 0.6996

Would you like to compute the LSH quality compared to exact Jaccard similarity? (no to exit):
```

Ο υπολογισμός του quality θα γίνει εκτός αν ο χρήστης εισάγει την τιμή ‘no’. Εκτός από τις πραγματικές N πιο όμοιες εγγραφές, βλέπουμε και τις πληροφορίες για το quality.

```
Time taken to compute similarity for all documents: 27.57 seconds

Evaluating the performance of LSH on displayed pairs...

Performance Metrics (For Displayed Pairs):
Total pairs checking: 10
Common pairs between LSH and exact method (Displayed): 8
LSH Quality (% of displayed LSH pairs found in exact method): 80.00%
```

Δηλαδή το lsh προσεγγίζει τις πραγματικά N πιο όμοιες εγγραφές κατά 80% και απαιτεί το $\frac{10.86}{27.56} \approx 40\,\%$ του χρόνου από το υπολογιστούν όλες οι ομοιότητες σε O(n^2).

Στην περίπτωση όπου το δέντρο περιορίσει αρκετά τις εγγραφές και δεν υπάρχουν αρκετές εγγραφές να εξεταστούν, πιθανών το Lsh να βγάλει ότι δεν υπάρχουν όμοια reviews. Σε αυτήν την περίπτωση πρέπει να μειωθεί το *user_defined_threshold*.

Επειδή υπάρχουν πολλές μεταβλητές που συνεισφέρουν στο quality του lsh, όπως είναι το πλήθος εγγραφών εισόδου, το N που εισάγει ο χρήστης, το user_defined_threshold και το *shingling_size,* είναι δύσκολο να παραστεί γραφικά το quality του lsh σε συνάρτηση με όλα τα προηγούμενα. Για αυτό τον λόγο, το quality θα αποτυπωθεί στον παρακάτω πίνακα.

Εγγραφές εισόδου	N	user_defined_threshold	shingling_size	Quality
1247	10	0.5	3	80%
1247	20	0.5	3	75%
1247	40	0.5	3	65%
1247	10	0.5	4	70%
1247	10	0.6	3	80%
613	10	0.5	3	60%
613	20	0.4	3	85%
613	50	0.4	3	54%
613	6	0.4	4	66.67%
300	10	0.4	3	50%
300	20	0.4	3	50%
96	8	0.4	3	75%
96	10	0.3	3	70%
96	20	0.3	3	40%

Όπου φαίνεται πως το quality είναι ικανοποιητικό για μεγάλο αριθμό εγγραφών εισόδου αλλά για μικρό πλήθος εγγραφών, το quality μειώνεται σημαντικά και μερικές φορές δεν υπολογίζεται σωστά.