R-Tree ή Rectangle Tree. Ονομάζεται η πολυδιάστατη δομή δεδομένων που όρισε ο Guttman το 1984[1].

Κάνοντας χρήση των Minimum Bounding Rectangles (MBRs) περιγράφουμε το απαιτούμενο χώρο για να υπάρξει indexing ενός οποιουδήποτε σχήματος αντικειμένου.
Βρισκόμαστε στον 2d χώρο, στο επίπεδο.

Ένα ορθογώνιο λοιπόν, όπως και ένα τετράγωνο, περιγράφεται από την διαγώνιο του.

Αντίστοιχα η διαγώνιος αυτή, όπως κάθε ευθεία, περιγράφεται από δύο σημεία στο επίπεδο.

Τα κάτω αριστερά και πάνω δεξιά σημεία του ορθογωνίου μας.

Σε αυτά υπάρχουν οι ελάχιστες και μέγιστες τιμές των x και y, δίνοντας την καλύτερη δυνατή εικόνα για τη τοποθεσία του ορθογωνίου στο επίπεδο.

Το R-Tree, περιγράφοντας στοιχεία μέσω των εφαρμοστών τους ορθογωνίων, του δίνεται η δυνατότητα να τα ομαδοποιήσει. Περιορισμένο εξίσου με ελάχιστο και μέγιστο όριο αντικειμένων, τα οποία κωδικοποιούμε ως Entries, m και M, όπου m<=M/2, καταφέρνει Insert σε O(n) και Search σε O(logm(N)) χρόνο.

Έχοντας μια καθαρή εικόνα για το βασικά στοιχειά της δομής προχωράμε στα εξής :

- Δημιουργία κλάσης Point , Rectangle, Node, Entry και RTree
- Βοηθητικές συναρτήσεις όπως union, find_indices , _pick_seeds
- Καθορισμός της μορφής των data για εισαγωγή

Κλάσεις

Point : ένα δυσδιάστατο σημείο

Rectangle: ένα ορθογώνιο, ορισμένο από τα σημεία του με τις ελάχιστες και τις μέγιστες καρτεσιανές συντεταγμένες.

1. low = το σημείο κάτω αριστερά με x = xmin, y=ymin
2. high = το σημείο πάνω δεξιά με x = xmax, y=ymax

Μέσω των παραπάνω προκύπτει το ύψος, το πλάτος και αντίστοιχα η επιφάνεια ενός Rectangle.

i) changed_rectangle, η οποία επιστρέφει το MBR του Rectangle μας με ένωση αυτό του ορίσματος.
ii) Overlaps, της οποίας η χρήση εξηγείται εδώ [2]
iii) same_as, η οποία ελέγχει αν συμπίπτουν τα Rectangles
iv) intersects, η όποια επιστρέφει ως Rectangle τη τομή των ορθογωνίων μας
v) contains, αν το Rectangle μας είναι περιέχει ολικά το όρισμα

Node: Το βασικό δομικό στοιχείο κάθε δένδρου, ο κόμβος. Χαρακτηρίζεται, ως leaf ή μη, αναλόγως αν περιέχει δεδομένα, από τον αριθμό Entries που περιγράφει, m ως M, και από τον γονικό του κόμβο.

i) Ρίζα του δένδρου μας θα είναι ο κόμβος χωρίς γονεά, και η is_root εξετάζει ακριβώς αυτό

ii) parent_entry , ψάχνουμε να βρούμε το node που εξετάζουμε ως entry στα παιδιά του γονέα του. Μόνο έτσι υπάρχει πρόσβαση στο MBR ενός κόμβου.
iii) get_bounding_rect, συνάρτηση για να υπολογιστεί το MBR ενός Node

Entry: Το περιεχόμενο ενός κόμβου. Έχει ένα MBR, και ένα pointer, είτε σε ένα child node ή στα data μας. Αντιστοίχως το MBR περιγράφει ένα από τα δύο.

RTree: Η κεντρική δομή μας το RTree, καθορίζεται από το ελάχιστο και μέγιστο αριθμό entries που μπορεί να έχει κάθε κόμβος. Με την δημιουργία του φτιάχνεται και ένα leaf root node.

i) Insertion: Γίνεται η επιλογή τού leaf κόμβου για την εισαγωγή των data μας με το MBR τους μέσω της choose_leaf.
   Έλεγχος κόμβου για των αριθμό Entry. Πιθανώς να απαιτείται να γίνει split στο node μας ώστε τα Entries να διαμοιραστούν σε 2, στον ήδη υπάρχων και σε έναν καινούργιο, κόμβους ακολουθώντας τον αλγόριθμο Quadratic_split.
   Καλείται η adjust_tree_strategy ώστε να διορθωθούν οποιαδήποτε προβλήματα ανακρίβειας προκύψουν.
ii) search_tree : range search στα R Trees έχοντας σαν όρισμα έναν rectangle που περιγράφει το χώρο αναζήτησής μας.
   Από τη ρίζα του δέντρου μας προσπελαύνουμε το δέντρο top -down.
   Αναδρομικά σε όλα τα non leaf nodes εξετάζουμε κόμβους μόνο για την τομή τους με τον χώρο αναζήτησης μας.. Όταν φτάσουμε στους κόμβους φύλλα, σιγουρεύουμε ότι τα data είναι στα πλαίσια της αναζήτησής μας αν και μόνο αν το MBR αυτών εμπεριέχεται πλήρως από το τελικό εξεταζόμενο χώρο.
   Όλες οι θετικές περιπτώσεις αποθηκεύουν τα data τους σε μια λίστα, η οποία είναι και το τελικό αποτέλεσμα του search.
iii) choose_leaf: Επιλέγουμε το Node που θα υποστεί την λιγότερη αλλαγή στο mbr του δοσμένου ενός Rectangle, μέσω της find_least_area. Ουσιαστικά, από την ρίζα προς τα κάτω επιλέγουμε το ελάχιστο δυνατό μονοπάτι στο δέντρο όσον αφορά τις προσαυξήσεις στην επιφάνεια των Rectangles του.
iv) **find_least_area** : Βρίσκουμε και επιστρέφουμε το Entry με την λιγότερη επιφάνεια προσαύξησης δοσμένου ενός Rectangle για insertion. Αν έχουμε πάνω από 1 με την ίδια αυτή τιμή, επιλέγουμε αυτό με την λιγότερη αρχική επιφάνεια.
v) quadratic_split: Πολιτική διαχωρισμού κόμβου.
   Σκοπός είναι να επιλέξουμε για να χωριστούν, τα δύο MBRs με την μεγαλύτερη απόσταση μεταξύ τους, μέσω της **_pick_seeds**. Κάθε ένα από τα υπόλοιπα M-1 entries πρέπει να πάνε στο πιο κοντινό τους group τηρώντας πάντα την συνθήκη των ελάχιστων entry ανά κόμβο. Επιλέγουμε το entry με την ελάχιστη απόσταση από οποιοδήποτε group μέσω της **_pick_next** και ξεκαθαρίζουμε σε ποιο από αυτά θα εισαχθεί. Για edge cases έχουμε 2 tie breakers: 1. Το group με τη λιγότερη αρχική επιφάνεια, 2. Το group με τα λιγότερα entries. Ωστόσο, αν σε ένα από τα δύο group πληρείται η συνθήκη Entries=>m, και αντίστοιχα στο άλλο group όχι, τότε απευθείας τα entries θα δοθούν στο group με το μη επαρκές αριθμό Entry.
   Μέχρι στιγμής έχει αποφασιστεί πως θα διαμοιραστούν τα entries χωρίς όμως κάποια «πραγματική» ενέργεια, οπότε καλείται η perfom_node_split.

vi) **_pick_seeds**: Επιλέγουμε ανά 2 τα Entries μας και υπολογίζουμε την προσαύξηση της επιφάνειας σε περίπτωση ένωσης των MBRs τους. Η δυάδα με την περισσότερη σπατάλη χώρου θα είναι αυτή που θα επιλεχθεί.

vii) **_pick_next**: Υπολογίζουμε την προσαύξηση των MBRs όλων υπολειπόμενων entry και για τα 2 πιθανά group. Η διαφορά των 2 αποτελεσμάτων δηλώνει την απόσταση που έχει το Entry μας από τα group. Αν το d1-d2 είναι κοντά στο 0, σημαίνει ότι το εξεταζόμενο MBR ισαπέχει από τα άλλα 2 Rectangles. Ωστόσο, το d1-d2 αυξάνεται γραμμικά όσο πλησιάζει το entry μας σε ένα από τα δύο group. Επιλέγεται το entry που έχει την μεγαλύτερη, κατά απόλυτη τιμή, διαφορά d1-d2

viii) perform_node_split: Δημιουργούμε ένα διπλότυπο δεύτερο node με τον ίδιο γονέα και του αναθέτουμε το 2^{ο} group entry και σε περίπτωση που δεν είναι leaf. Σε περίπτωση που το split έχει γίνει σε πιο πάνω level, κάνουμε adjust και τα παιδιά του νέου split_node μέσω της _fix_children. Ο αρχικός μας κόμβος περιέχει ήδη τα περιεχόμενα του group1 λόγω ότι σε εκείνον έχουν γίνει τα insert.

ix) adjust_tree_strategy: Το MBR ενός node βρίσκεται στο parent του, οπότε στην περίπτωση splitting προκύπτουν ανακρίβειες. Εδώ κοιτάμε να επιλύσουμε αυτά τα ζητήματα όπως και καταστάσεις chain splitting bottom-up. Βρίσκουμε το node μας, ως entry, μέσω της parent_entry, ώστε να κάνουμε adjust το mbr του. Σε περίπτωση που έχει υποστεί splitting, πρέπει να φτιάξουμε νέο Entry για το parent, και αναδρομικά προς τα πάνω αν χρειάζεται, εξετάζουμε αν κάποιο parent node χρειάζεται split. Αν έχει γίνει split η ρίζα τότε καλούμε την grow_tree.

x) grow_tree: Με όρισμα την δυάδα των nodes που προέκυψε, δημιουργεί τα MBR των κόμβων που δέχθηκε σας όρισμα και δημιουργεί ένα νέο non leaf node το οποίο χαρακτηρίζουμε ως root, κάνοντας adjust και τον pointer των παιδιών του. Με αυτό τον τρόπο έχουμε καταφέρει να μεγαλώσει το δέντρο μας κατά 1 επίπεδο.

xi) Η union και union_all είναι βοηθητικές συναρτήσεις για να παίρνουμε τα MBR όλων των Entry με μια μόνο κλήση συνάρτησης.

xii) Η find_indices ψάχνει εσωτερικά σε μια λίστα, με σκοπό να βρει σε ποιες θέσεις υπάρχει ένας συγκεκριμένος αριθμός ή και αντικείμενο. Επιστρέφει σε μια λίστα όλες τις τιμές που βρήκε.