Εργασία 2023 - 2024

Τεχνολογια βασεων δεδομενων

Υλοποίηση Χωρικών Μεθόδων Προσπέλασης

Ναλμπάντη παναγιωτα, 4050

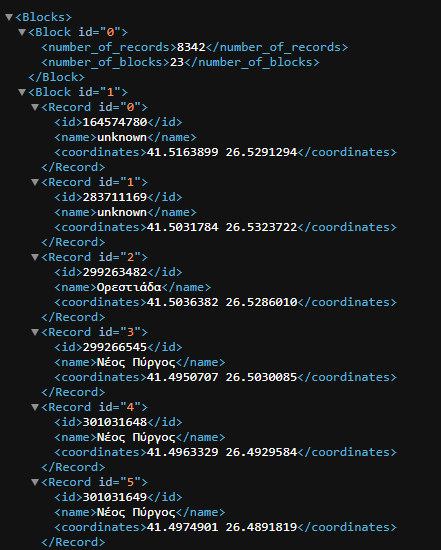
παντελοπουλου βασιλικη, 4153

### **Εισαγωγή**

Η εργασία αυτή ασχολείται με την οργάνωση και την επεξεργασία δεδομένων χώρου με πολλαπλές διαστάσεις. Συγκεκριμένα, γίνεται υλοποίηση της κατασκευής μίας δομής δεδομένων δευτερεύουσας μνήμης, του R\*- tree. Αυτή υποστηρίζει βασικά ερωτήματα (range query, knn, skyline), καθώς και κατασκευή καταλόγου με τις βασικές λειτουργίες του.

Για την υλοποίηση των παραπάνω χρησιμοποιήσαμε το δοθέν αρχείο osm, το οποίο περιέχει δεδομένα από το OpenStreetMap που αναφέρονται σε περιοχές του κόσμου με σημεία ενδιαφέροντος. Είναι απαραίτητο να αντιμετωπίσουμε τα δεδομένα ως δισδιάστατα καθώς τα δεδομένα από το osm αρχείο έχουν μόνο συντεταγμένες lat και lon. Όλες οι υπόλοιπες συναρτήσεις του project λειτουργούν θεωρώντας ότι μπορούμε να έχουμε περισσότερες απο δύο διαστάσεις. Επιλέχθηκε η εκπόνησή της με γλώσσα Python διότι παρέχει πολλές χρήσιμες βιβλιοθήκες οι οποίες δεν χρειάζονται εγκατάσταση καθώς ανήκουν στο Python Standard Library. Σε όλο το project χρησιμοποιήθηκαν οι εξείς βιβλιοθήκες:

* xml.etree.ElementTree
* heapq
* math
* time
* sys

Για την αποθήκευση και αναπαράσταση των εγγραφών χρησιμοποιήθηκε μορφή xml καθώς παρέχει έναν εύκολα διαχειρίσιμο και πλήρως κατανοητό τρόπο αποθήκευσης δεδομένων, ενώ ταυτόχρονα επιτρέπει το read και write ανά Block, το οποίο ήταν το κύριο ζητούμενο από το datafile ώστε να προσομοιωστεί η λειτουργία read και write στην μνήμη/buffer. Το μόνο μειονέκτημα της xml αναπαράστασης είναι η απόδοσή του καθώς, όπως είναι γνωστό, δεν φημίζεται για την ταχύτητά του. Παρόλαυτα, εφόσον εξετάζονται χρονικά οι διεργασίες της δομής R\*-Tree ως δευτερεύον κατάλογος, και η ταχύτητα εκτέλεσης των διάφορων queries δεν επηρεάζονται από την αναπαράσταση αυτή, κρίθηκε ως αποδεκτό format για το datafile αφού επηρεάζει μόνο την ταχύτητα κατασκευής του καταλόγου.Ένα ενδεικτικό τμήμα ενός datafile.xml αρχείου δίνεται παρακάτω :

### **Μεθοδολογία**

Πριν την έναρξη της κατασκευής του R\* δένδρου, χρειάζεται η δημιουργία μιας λίστας, η οποία θα αποτελείται από ένα πλήθος blocks των 32KB. Αυτά θα περιέχουν λίστες από records, τα οποία λαμβάνουμε από το osm αρχείο. Τα records περιέχουν πληροφορίες σχετικά με το αναγνωριστικό, το όνομα και τις συντεταγμένες ενός σημείου (id, name, lat/ lon/ κλπ). Οι διάφορες συναρτήσεις βρίσκουν τον αριθμό των διαστάσεων μέσα από τα δεδομένα. Επομένως, **αν θέλουμε η δομή να λειτουργήσει για δεδομένα πολλών διαστάσεων αρκεί να δώσουμε στην συνάρτηση createBlocks μια λίστα record\_data θα περιέχει records της μορφής** : **[ id, name, coord\_1, coord\_2, …, coord\_n ]**

Αυτό θα έχει ως αποτέλεσμα να δημιουργηθεί το επιθυμητό datafile που περιέχει σημεία πολλαπλών διαστάσεων.

Συνεχίζουμε δημιουργώντας ένα αρχείο datafile.xml, το οποίο θα περιέχει την λίστα των blocks με τα records σε μορφή xml.

### **Υλοποίηση R\*- tree**

1. Δημιουργία και διαχείρηση datafile

Ο κώδικας που υλοποιήθηκε στο **create\_datafile.py** παράγει ως αποτέλεσμα το αρχείο **datafile.xml**. Για την ευκολότερη διαχείριση των δεδομένων και την δημιουργία του datafile χρησιμοποιήθηκε η βιβλιοθήκη της Python xml.etree.ElementTree για την ανάγνωση του αρχικού osm αρχείου. Πιο αναλυτικά, μετά την ληψή του osm αρχείου και αρχικοποιώντας μία λίστα **record\_data** προστέθηκαν προσωρινά σε αυτή μόνο τα δεδομένα του osm με ετικέτα **‘node’** εφόσον μας ενδιαφέρουν μόνο τα σημεία. Από κάθε node διατηρούνται μόνο το **id**, το **όνομα(name)** και οι **lat-lon** συντεταγμένες του. Σε περίπωση που κάποιο node δεν διαθέται όνομα( tag name), ονομάζεται ‘**unknown’**. Αν τα σημεία προσδιορίζονται από περισσότερες διαστάσεις και όχι μόνο lat-lon, αυτές μπορούν να προστεθούν σε αυτό το σημείο : record\_data.append([id, name, lat, lon]).

Έπειτα για την αποθήκευση των εγγραφών σε **blocks των 32KB**, η λίστα με τα δεδομένα (record\_data) δίνεται ως παράμετρος στην συνάρτηση **createBlocks(record\_data)** η οποία είναι υπεύθυνη για τη δημιουργία των blocks μεγέθους 32ΚΒ που προσδιορίζεται από το block\_size = 32 \* 1024. Για να αναπαρασταθεί ο διαχωρισμός των εγγραφών σε blocks επιστρέφεται μία λίστα από λίστες **listOfBlocks**. Στη πρώτη θέση της λίστας βρίσκεται το **block0** το οποίο περιέχει βοηθητικές πληροφορίες σχετικά με το πλήθος των εγγραφών και τον αριθμό των blocks που δημιουργήθηκαν.

Μετά την διαδιακασία διαχωρισμού των εγγραφών σε blocks, αυτά αποθηκεύονται στο αρχείο datafile.xml μέσω της συνάρτησης **create\_xml(blocks,records,xml)** που δέχεται ως παραμέτρους τη λίστα με τα blocks, το πλήθος των εγγραφών και το όνομα του αρχείου(προαιρετικά).

Στιγμιότυπα κώδικα:

(πρεπει να σβηστουν τα testing και να μπει το καθαρό)

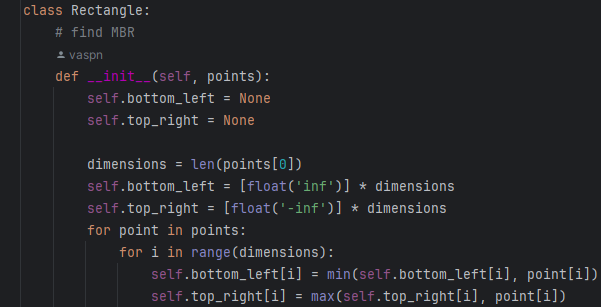
1. Δημιουργία του R\*-tree

Για την αναπαράσταση της δομής δημιουργήθηκαν βοηθητικές κλάσεις που βρίσκονται στο Entry.py και Node.py οι οποίες αντιπροσωπεύουν το είδος των εγγραφών και τους κόμβους αντίστοιχα. Πιο συγκεκριμένα, στο αρχείο Entry.py βρίσκονται οι εξής κλάσεις :

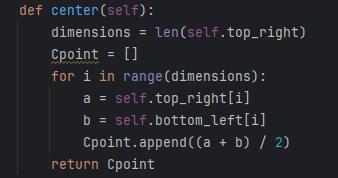
* **Rectangle**: Αναπαριστά ένα πολυδιάστατο ορθογώνιο που ορίζεται από δύο σημεία. Έχει δύο πεδία :
* **bottom\_left** που αντιπροσωπεύει το κάτω αριστερά σημείο
* **top\_right** που αντιπροσωπεύει το πάνω δεξία σημείο

Περιλαμβάνει τις εξής μεθόδους:

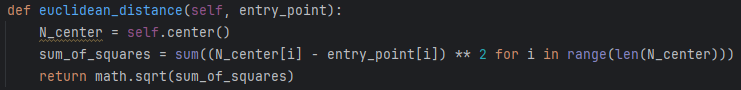
* **\_\_init\_\_(self, points)**: παίρνει ως όρισμα μία λίστα από σημεία (πχ [[7,5, 5.2], [0,3, 1.2], [2,5, 1.7]]) και υπολογίζει το minimum bounding rectangle(ΜΒR), βρίσκοντας και αρχικοποιώντας το bottom\_left και το top\_right υπολογίζοντας τη μικρότερη και μεγαλύτερη τιμή κάθε διάστασης αντίστοιχα:



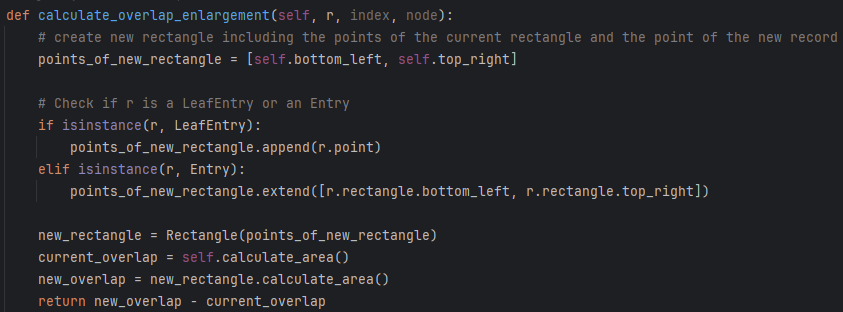
* **center(self):** υπολογίζει και επιστρέφει ως λίστα τις συντεταγμένες του κεντρικού σημείου ενός ορθογωνίου:



* **euclidean\_distance(self, entry\_point)**: δέχεται ως όρισμα ένα σημείο και υπολογίζει και επιστρέφει την ευκλείδια απόστασή του από το κέντρο του ορθογωνίου. Η ευκλείδια απόσταση υπολογίζεται ως η τετραγωνική ρίζα του αθροίσματος των τετραγώνων των διαφορών center[i] – point[i] για κάθε διάσταση i:

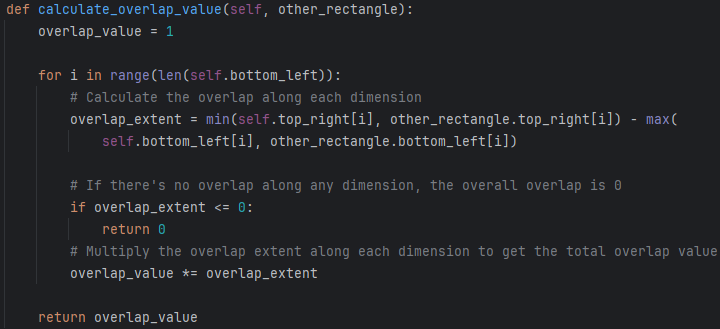


* **calculate\_overlap\_enlargement(self, r, index, node)**: υπολογίζει την αλλαγή (αύξηση) στο εμβαδόν της επικάλυψης ενός ορθογωνίου όταν προστίθεται ένα νέο σημείο ή ένα άλλο ορθογώνιο. Υπολογίζει το εμβαδόν του τρέχοντος ορθογωνίου όπως και του νέου με την calculate\_area(self). Τέλος, επιστρέφεται η διαφορά μεταξύ του νέου εμβαδού (new\_overlap) και του τρέχοντος εμβαδού (current\_overlap), που αντιπροσωπεύει την αύξηση του εμβαδού επικάλυψης.

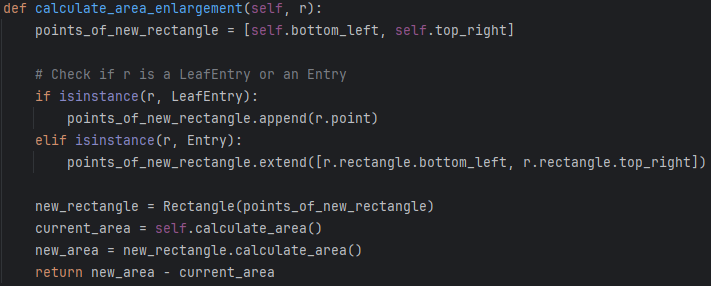


* **calculate\_overlap\_value(self, other\_rectangle)**: υπολογίζει το εμβαδόν επικάλυψης μεταξύ δύο ορθογωνίων. Συγκεκριμένα, υπολογίζει την επικάλυψη μεταξύ του ορθογωνίου που καλεί τη συνάρτηση (self) και ενός άλλου ορθογωνίου (other\_rectangle) κατά μήκος κάθε διάστασης. Για κάθε διάσταση i (χρησιμοποιώντας το μήκος της λίστας bottom\_left για τον αριθμό διαστάσεων), υπολογίζεται το μήκος της επικάλυψης κατά μήκος της συγκεκριμένης διάστασης. Αυτό γίνεται ως εξής:

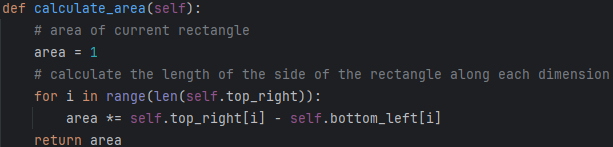
Υπολογίζεται το μέγιστο από τα κάτω άκρα των δύο ορθογωνίων και το ελάχιστο από τα άνω άκρα τους. Η διαφορά μεταξύ του ελάχιστου άνω άκρου και του μέγιστου κάτω άκρου δίνει το μήκος της επικάλυψης για τη συγκεκριμένη διάσταση. Αν το μήκος της επικάλυψης (overlap\_extent) είναι μικρότερο ή ίσο με το μηδέν σε οποιαδήποτε διάσταση, αυτό σημαίνει ότι τα ορθογώνια δεν επικαλύπτονται σε αυτή τη διάσταση, και συνεπώς η συνολική επικάλυψη είναι μηδενική. Σε αυτή την περίπτωση, η συνάρτηση επιστρέφει 0. Εάν υπάρχει επικάλυψη σε όλες τις διαστάσεις, τότε οι τιμές επικάλυψης για κάθε διάσταση πολλαπλασιάζονται για να δώσουν το συνολικό εμβαδόν επικάλυψης.



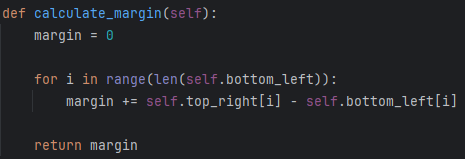
* **calculate\_area\_enlargement(self, r):** υπολογίζει την αύξηση της επιφάνειας ενός ορθογωνίο όταν επεκτείνεται για να συμπεριλάβει ένα νέο στοιχείο, που μπορεί να είναι είτε ένα σημείο είτε ένα άλλο ορθογώνιο. Ένα νέο ορθογώνιο δημιουργείται χρησιμοποιώντας τα σημεία της λίστας points\_of\_new\_rectangle. Το νέο αυτό ορθογώνιο περιλαμβάνει τόσο το αρχικό ορθογώνιο όσο και το νέο στοιχείο. Υπολογίζεται η επιφάνεια του τρέχοντος και του νέου ορθογωνίου χρησιμοποιώντας τη μέθοδο calculate\_area. Τελικά, επιστρέφει την αύξηση της επιφάνειας, δηλαδή τη διαφορά μεταξύ της επιφάνειας του νέου ορθογωνίου και της αρχικής επιφάνειας (new\_area - current\_area).



* **calculate\_area(self)**: υπολογίζει και επιστρέφει το εμβαδόν του ορθογωνίου. Ο υπολογισμός του εμβαδού γίνεται ως το γίνομενο των διαφορών top\_right [i]-bottom\_left [i] για κάθε διάσταση i.

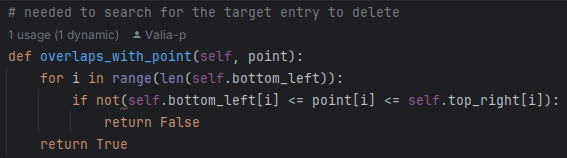


* **calculate\_margin(self)**: υπολογίζει και επιστρέφει την περίμετρο του ορθογωνίου. Ο υπολογισμός της περιμέτρου γίνεται ως το άθροισμα των διαφορών top\_right[i]-bottom\_left[i] για κάθε διάσταση i.

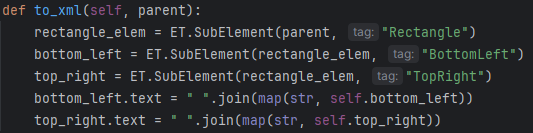


* **overlaps\_with\_point(self, point)**: ελέγχει αν ένα σημείο βρίσκεται μέσα σε ένα ορθογώνιο. Χρησιμοποιεί ένα βρόχο για να περάσει από όλες τις διαστάσεις του ορθογωνίου και για κάθε διάσταση, ελέγχει αν η συντεταγμένη του σημείου point σε αυτή τη διάσταση βρίσκεται εντός των ορίων του ορθογωνίου. Αυτό γίνεται με τον έλεγχο: self.bottom\_left[i] <= point[i] <= self.top\_right[i].

Αν το σημείο βρίσκεται εκτός του ορθογωνίου σε οποιαδήποτε διάσταση επιστρέφει False, σηματοδοτώντας ότι το σημείο δεν είναι μέσα στο ορθογώνιο. Διαφορετικά επιστρέφει True αν η συνθήκη ικανοποιείται για όλες τις διαστάσεις.



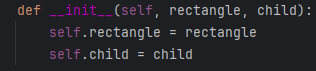
* **to\_xml(self, parent)**: Κωδικοποιεί ένα αντικείμενο της κλάσης Rectangle σε μορφή xml. Πιο συγκεκριμένα, δέχεται το parent element που θα έχει το αντικέιμενο Rectangle στο indexfile.xml και δημιουργεί ένα subelement με tag “Rectangle” και μέσα σε αυτό το tag βάζει δύο subelement: το “BottomLeft” και το “TopRight”, τα οποία περιέχουν το καθένα τις αντίστοιχες συντεταγμένες.



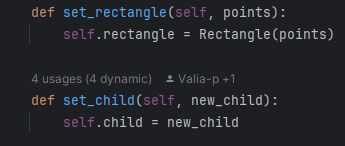
* **Entry**: Αναπαριστά μία εσωτερική εγγραφή, δηλαδή εγγραφή εσωτερικού κόμβου. Αποτελείται από δύο πεδία :
* **rectangle**: αναπαριστά το minimum bounding rectangle του κόμβου παιδιού του συγκεκριμένου entry.
* **child**: αναπαριστά τον κόμβο παιδί του συγκεκριμένου entry.

Περιλαμβάνει τις εξής μεθόδους:

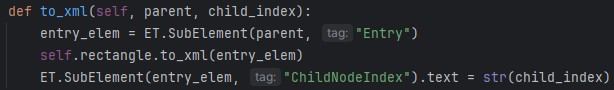
* **\_\_init\_\_(self, rectangle, child)**: η οποία δέχεται ως ορίσματα ένα ορθογώνιο και ένα κόμβο ώστε να αρχικοποιήσει τα πεδία rectangle και child.



* **set\_rectangle(self,points)**: δέχεται ως όρισμα μία λίστα από σημεία και ορίζει το πεδίο rectangle με το MBR των σημείων που δίνονται.
* **set\_child(self,new\_child)**: δέχεται ως όρισμα ένα κόμβο και ορίζει το πεδίο child με το κόμβο που δίνεται.



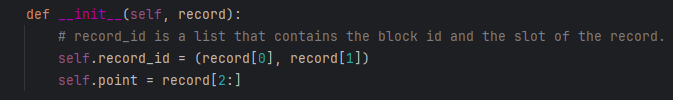
* **to\_xml(self, parent, child\_index)**: Κωδικοποιεί ένα αντικείμενο της κλάσης Entry σε μορφή xml. Συγκεκριμένα, δέχεται ένα γονικό στοιχείο xml (parent) που θα περιέχει το αντικείμενο Entry στο αρχείο indexfile.xml. Στη συνέχεια, δημιουργεί ένα υποστοιχείο με το tag "Entry". Μέσα σε αυτό το tag, προσθέτει την xml αναπαράσταση του self.rectangle, χρησιμοποιώντας τη μέθοδο to\_xml της κλάσης Rectangle. Επιπλέον, προσθέτει ένα βοηθητικό υποστοιχείο με το όνομα "ChildNodeIndex", το οποίο περιέχει το child\_index που δέχεται η μέθοδος ως παράμετρο. Αυτός ο δείκτης αντιπροσωπεύει τη θέση του παιδιού του συγκεκριμένου Entry στη λίστα που περιλαμβάνει ιεραρχικά όλους τους κόμβους που απαρτίζουν το R-tree.



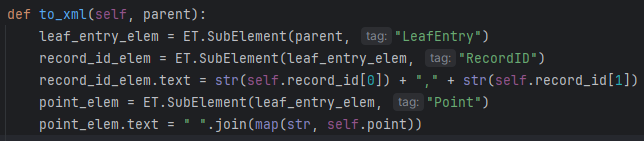
* **LeafEntry**: Αναπαριστά μία εγγραφή φύλλου δηλαδή μια εγγραφή ενός κόμβου φύλλου. Έχει δύο πεδία:
* **record**: Αποτελεί μία λίστα που περιέχει το id του block στο οποίο βρίσκεται η συγκεκριμένη εγγραφή στο datafile και το slot στο οποίο βρίσκεται μέσα στο block.
* **point**: Αποτελεί μία λίστα που περιέχει τις συντεταγμένες του σημείου.

Περιλαμβάνει τις εξής μεθόδους:

* **\_\_init\_\_(self, record)**: αρχικοποιεί τα δύο πεδία της κλάσης παίρνοντας ως όρισμα ένα record από το datafile που έχει τη μορφή λίστας όπου στις θέσεις 0 και 1 έχει το record\_id και στις υπόλοιπες θέσεις περιέχει τις συντεταγμένες του σημείου.



* **to\_xml(self, parent)**: Κωδικοποιεί ένα αντικείμενο της κλάσης LeafEntry σε μορφή xml. Ειδικότερα, δέχεται ένα γονικό στοιχείο xml (parent) που θα φιλοξενήσει το αντικείμενο LeafEntry στο αρχείο indexfile.xml. Στη συνέχεια, δημιουργεί ένα υποστοιχείο με την ετικέτα "LeafEntry". Μέσα σε αυτό το tag, προσθέτει δύο υποστοιχεία: το "RecordID", το οποίο περιέχει τις δύο τιμές του self.record διαχωρισμένες με κόμμα, και το "Point", το οποίο περιλαμβάνει τις συντεταγμένες του σημείου διαχωρισμένες με κενά διαστήματα.



Στο αρχείο Node.py βρίσκονται οι εξής κλάσεις :

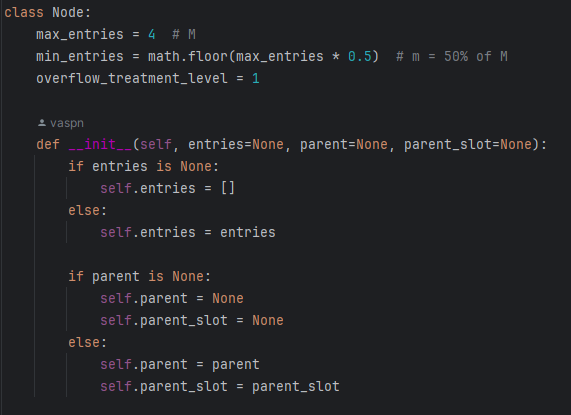
* **Node** : Αναπαριστά έναν κόμβο του R-tree και αποτελείται από τρία πεδία:
* **entries** : αποτελεί μία λίστα που περιέχει τις εγγραφές του κάθε κόμβου οι οποίες είναι όλες είτε αντικείμενα της κλάσης Entry, αν ο κόμβος είναι εσωτερικός, είτε αντικείμενα της κλάσης LeafEntry, αν ο κόμβος είναι φύλλο.
* **parenet** : αναπαριστά τον γονικό κόμβο Node μέσα στον οποίο βρίσκεται η εγγραφή Entry που έχει ως παιδί τον συγκεκριμένο κόμβο. Ο μόνος κόμβος που δεν έχει parent είναι η ρίζα του δέντρου.
* **parent\_slot** : αναπαριστά την θέση που καταλαμβάνει το γονικό Entry στην λίστα entries του γονικού κόμβου.

Επίσης έχει τρεις μεταβλητές κλάσεις, δηλαδή κοινές μεταβλητές για κάθε αντικείμενο της κλάσης αυτής :

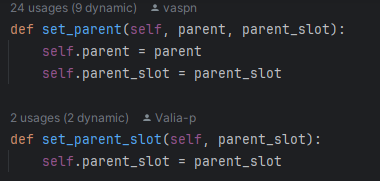
* + - **max\_entries** : εκφράζει το μέγιστο επιτρεπτό πλήθος εγγραφών, M, που μπορεί να περιέχει ένας κόμβος (άμα δεν οριστεί μέσα στον κώδικα , η default τιμή του είναι 4)
    - **min\_entries** : εκφράζει το ελάχιστο επιτρεπτό πλήθος εγγραφών, m, που μπορεί να περιέχει ένας κόμβος. Ορίζεται ως το μισό του max\_entries δηλαδή min\_entries = max\_entries\*50% (με στρογγυλοποίηση προς τον πλησιέστερο ακέραιο).
    - **overflow\_treatment\_level** : αποτελεί βοηθητική μεταβλητή που χρησιμοποιείται στην συνάρτηση overflow\_treatment κατά την διαδικασία του insert. Ορίζεται ως default τιμή του 1.

Περιλαμβάνει τις εξής μεθόδους:

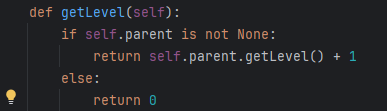
* **\_\_init\_\_(self, entries = None, parent = None, parent\_slot = None)**: Αρχικοποιεί τα πεδία που δέχεται ως ορίσματα. Σε περίπτωση που δεν δεχτεί κανένα όρισμα ή δεχτεί μόνο entries, αρχικοποιεί τα πεδία τα οποία δεν είχαν ανάλογο όρισμα με None και αναμένεται να γίνουν set ύστερα με τις ανάλογες μεθόδους(set\_parent και set\_parent\_slot).



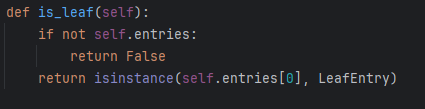
* **set\_parent(self, parent, parent\_slot)**: αποτελεί έναν setter για τα πεδία parent και parent\_slot.
* **set\_parent\_slot(self, parent\_slot)**: αποτελεί έναν setter μόνο για το πεδίο parent\_slot και χρησιμοποιείται όταν δεν αλλάζει ο γονικός κόμβος αλλά χρειάζεται ενημέρωση του parent\_slot λόγω κάποιας διαγραφής ή εισαγωγής.



* **getLevel(self)**: Αναδρομικά βρίσκει το επίπεδο του συγκεκριμένου κόμβου (η αρίθμηση ξεκινάει από το 0 που αποτελεί πάντα το επίπεδο της ρίζας.Πιο συγκεκριμένα, ακολουθεί τα πεδία των γονικών του κόμβων μέχρι να φτάσει στην ρίζα που δεν έχει γονέα και έπειτα μετράει από πόσους γονικούς κόμβους πέρασε.

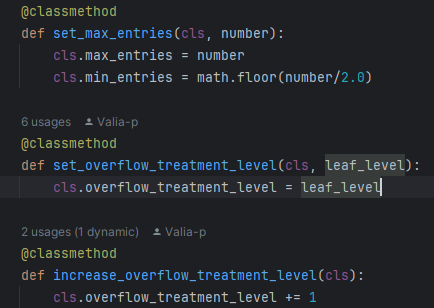


* **is\_leaf(self)**: Ελέγχει αν ο κόμβος είναι εσωτερικός ή φύλλο,εξετάζοντας την εγγραφή που βρίσκεται σε αυτόν. Επιστρέφει True αν είναι LeafEntry διαφορετικά False αν είναι Entry.



Περιλαμβάνει και τρεις μεθόδους κλάσης (λειτουργούν σε επίπεδο κλάσης και όχι αντικειμένου) και μπορούν να αλλάξουν την κατάσταση της ίδιας της κλάσης και όχι μόνο συγκεκριμένων αντικειμένων :

* **set\_max\_entries(cls, number)**: είναι setter των πεδίων max\_entries και min\_entries. Δέχεται ως όρισμα τον μέγιστο αριθμό εγγραφών που θα πρέπει να έχει κάθε κόμβος και αρχικοποιεί το max\_entries με τον αριθμό number και το min\_entries με το μισό αυτού του αριθμού.
* **set\_overflow\_treatment\_level(cls, leaf\_level)**: είναι setter του πεδίου overflow\_treatment\_level. Δέχεται ως όρισμα το επίπεδο των φύλλων του δέντρο και αρχικοποιεί το overflow\_treatment\_level.
* **increase\_overflow\_treatment\_level(cls)**: κάθε φορά που καλείται αυξάνει το πεδίο overflow\_treatment\_level κατά ένα.

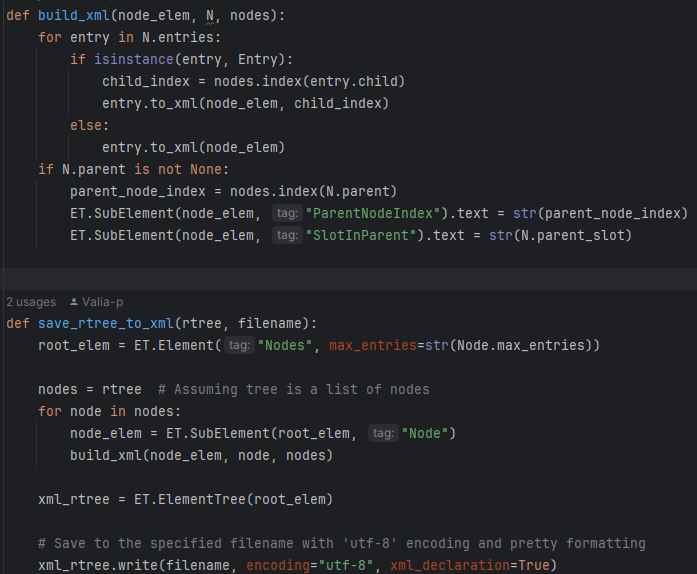


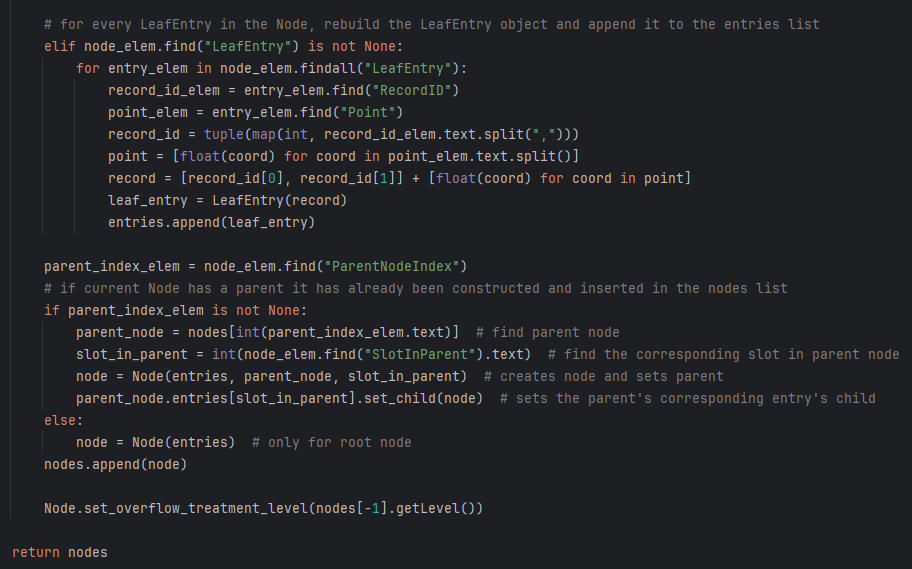
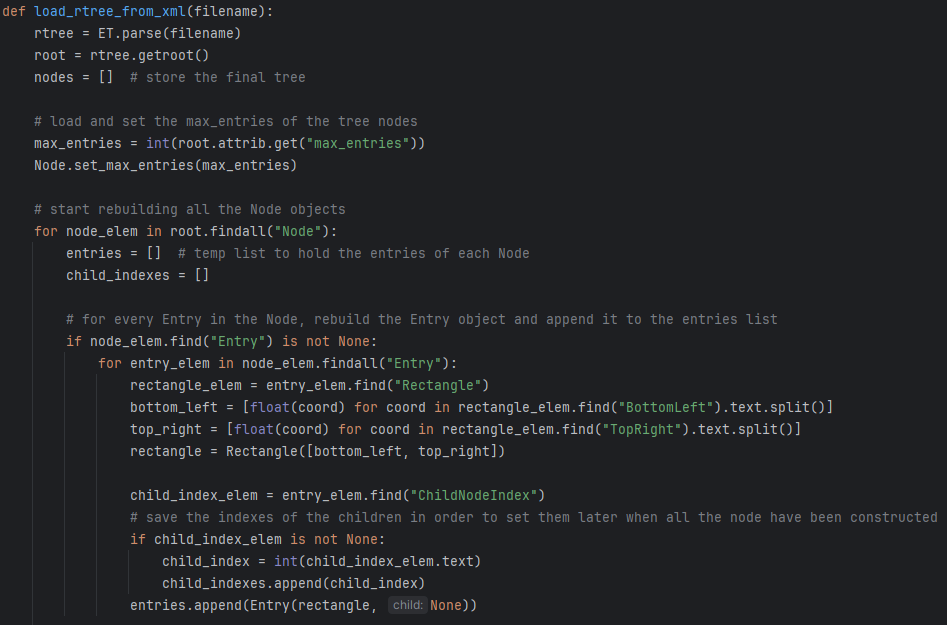
H δομή χτίζεται και αποθηκεύεται ως μία λίστα rtree που περιέχει αντικείμενα της κλάσης Node τα οποία διατηρούν μια συγκεκριμένη ιεραρχία με βάση το επίπεδο στο οποίο βρίσκεται ο κάθε κόμβος. Πιο συγκεκριμένα, στην θέση 0 της λίστας βρίσκεται η ρίζα, στις επόμενες θέσεις βρίσκονται οι κόμβοι-παιδιά της ρίζας, δηλαδή οι κόμβοι του πρώτου επιπέδου, έπειτα ακολουθούν τα παιδιά των παιδιών της ρίζας, δηλαδή οι κόμβοι του δεύτερου επιπέδου, κοκ. Επομένως, η ρίζα βρίσκεται στην αρχή της λίστας και τα φύλλα στο τέλος. Παρολαυτα, η ιεραρχία δεν επηρεάζει τη σωστή λειτουργία της δομής, απλά διευκολύνει τον έλεγχο και την αποθήκευσή της στο indexfile.

1. Δημιουργία και διαχείρηση indexfile

Ο κώδικας που υλοποιήθηκε στο **create\_indexfile.py** παράγει ως αποτέλεσμα το αρχείο **indexfile.xml**. Η κατασκευή του indexfile γίνεται από τη συνάρτηση **save\_rtree\_to\_xml(rtree, filename)** που δέχεται ως ορίσματα τη λίστα rtree που περιέχει ιεραρχικά όλους τους κόμβους και επιθυμητό όνομα του αρχείου. Δημιουργεί αρχικά το στοιχείο ρίζας που ονομάζεται “Nodes” και αποθηκεύει σε αυτό τη μέγιστη χωριτηκότητα των κόμβων (max\_entries). Έπειτα, για κάθε κόμβο στο rtree, δημιουργείται ένα υποστοιχείο “Node” και η συνάρτηση build\_xml (node\_elem, N, nodes) χρησιμοποιείται για να προσθέσει τις πληροφορίες του κάθε κόμβου (και των εγγραφών του) στο xml. Στην build\_xml, για κάθε εγγραφή του κόμβου, αν η εγγραφή είναι τύπου Entry, καλείται η μέθοδος to\_xml του Entry με το κατάλληλο child\_index για να εισαχθούν τα στοιχεία στο xml. Εάν η εγγραφή είναι τύπου LeafEntry, τα δεδομένα της εγγραφής αποθηκεύονται άμεσα. Επιπλέον, αν ο κόμβος έχει γονέα, προστίθενται στοιχεία ParentNodeIndex και SlotInParent για να καθοριστεί η θέση του στον γονικό κόμβο. Αφού ολοκληρωθεί αυτή η διαδικασία, δημιουργεί και αποθηκεύει στο indexfile το δέντρο.

Για την διαχείρηση του indexfile.xml χρησιμοποιείται η συνάρτηση **load\_rtree\_from\_xml(filename)** η οποία δέχεται ως όρισμα το όνομα του αρχείου στο οποίο είναι αποθηκευμένο το R\*-Tree και αναδομεί και επιστρέφει το δέντρο στην αρχική μορφή λίστας που είχε πριν την αποθήκευσή του.





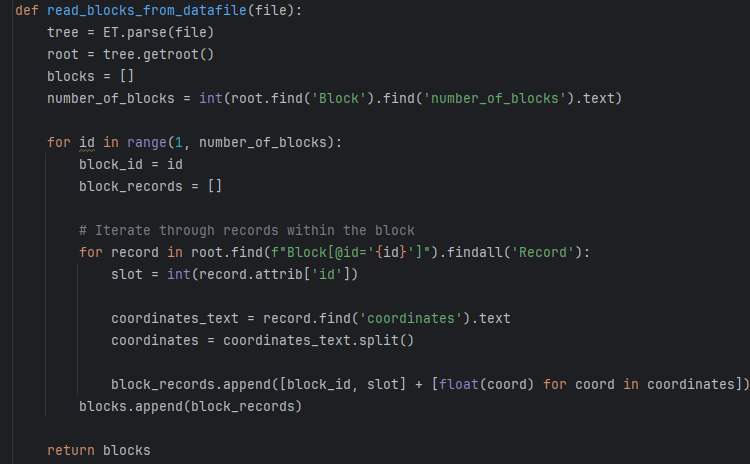
Το τελικό indexfile.xml που παράγεται έχει την εξής μορφή:



1. Εισαγωγή Εγγραφής

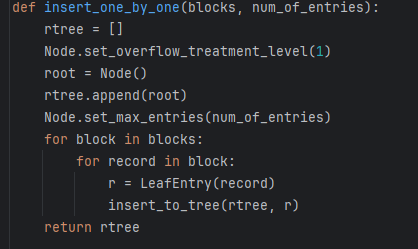
Η υλοποίηση της εισαγωγής εγγραφής στο rtree βρίσκεται στο αρχείο insert.py και η ορθή λειτουργία του βασίζεται στις παρακάτω συναρτήσεις(παρουσιάζονται με τη σειρά που καλούνται):

* **read\_blocks\_from\_datafile(file)**: διαβάζει τα δεδομένα από το αρχείο datafile.xml και εξάγει τα μπλοκ και τις εγγραφές τους. Το αρχείο xml περιέχει μπλοκ, καθένα από τα οποία περιέχει πολλές εγγραφές. Αρχικά, διαβάζει το block0 ώστε να πάρει απο αυτό τον συνολικό αριθμό των blocks. Κάθε εγγραφή περιλαμβάνει έναν μοναδικό αριθμό θέσης και συντεταγμένες. Η συνάρτηση δημιουργεί μια λίστα από μπλοκ, όπου κάθε μπλοκ είναι μια λίστα με τις εγγραφές του.

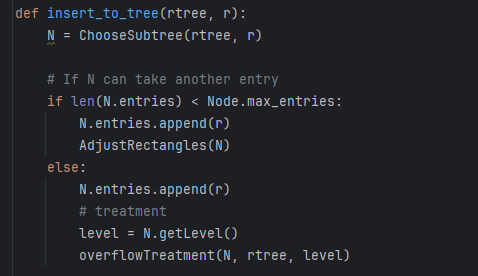


* **insert\_one\_by\_one(blocks, num\_of\_entries)**: κατασκευή του καταλόγου με εισαγωγή των εγγραφών μία προς μία από τα δεδομένα που έχουν ανακτηθεί από το read\_blocks\_from\_datafile. δέχεται ως ορίσματα τον μέγιστο αριθμό εγγραφών που μπορεί να έχει ένας κόμβος και μία λίστα που περιέχει τα

blocks απο το datafile. Ξεκινάει με μία κενή λίστα rtree και προσθέτει σε αυτή τη ρίζα ενώ αρχικοποιεί το επίπεδο αντιμετώπισης υπερφόρτωσης (overflow treatment level) στο 0. Η μέγιστη χωρητικότητα κάθε κόμβου καθορίζεται από την παράμετρο num\_of\_entries. Συνεχίζει, διαπερνώντας την λίστα με τα blocks και για κάθε record του εκάστοτε block δημιουργείται ένα αντικείμενο LeafEntry και εισάγεται στο δέντρο χρησιμοποιώντας τη συνάρτηση insert\_to\_tree. Στο τέλος, επιστρέφει την λίστα rtree η οποία περιέχει με ιεραρχική σειρά το δέντρο που προκύπτει απο την εισαγωγή των εγγραφών μία προς μία.



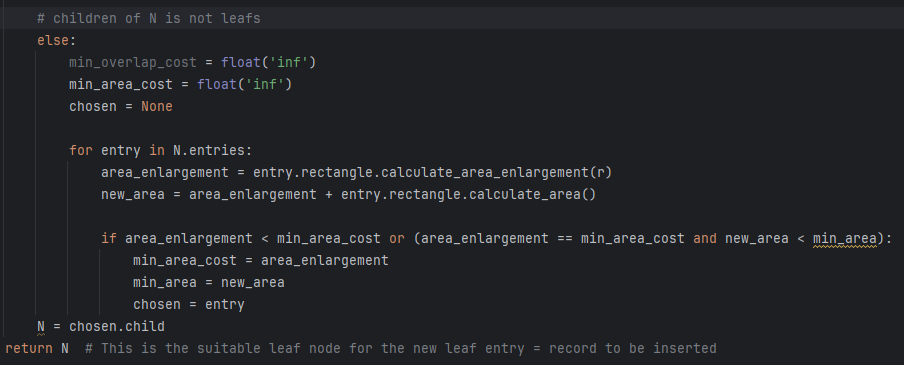
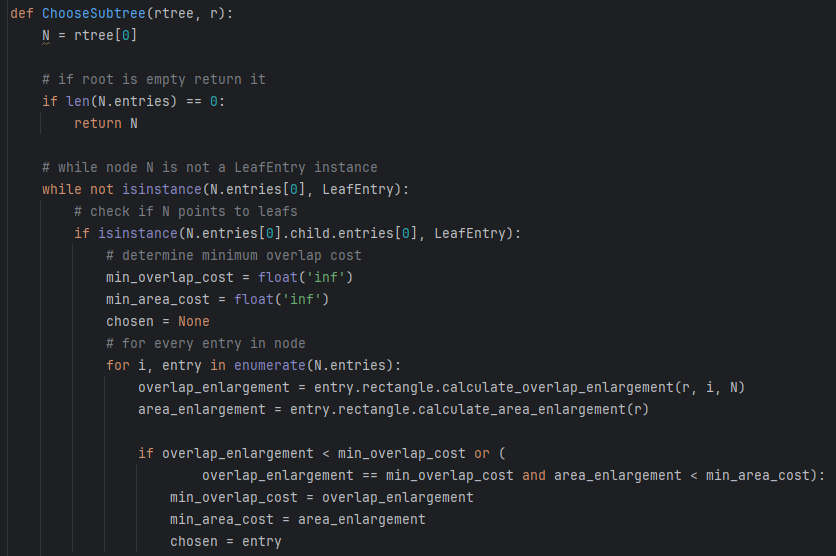
* **insert\_to\_tree(rtree, r)**: δέχεται ως ορίσματα την λίστα με τους κόμβους του δέντρου και την εγραφή τύπου LeafEntry που εισάγεται στο δέντρο. Χρησιμοποιεί τη συνάρτηση ChooseSubtree για να επιλέξει τον κατάλληλο κόμβο (leaf node) για την εισαγωγή της εγγραφής. Αν ο κόμβος έχει διαθέσιμο χώρο, η εγγραφή προστίθεται απευθείας, ακολουθούμενη απο μία κλήση της συνάρτησης AdjustRectangles(N) για τον συγκεκριμένο κόμβο. Διαφορετικά, αν ο κόμβος είναι πλήρης, γίνεται εισγωγή της νέας εγγραφής στον κόμβο και έπειτα ενεργοποιείται η διαδικασία υπερφόρτωσης μέσω της συνάρτησης overflowTreatment(N, rtree, level).



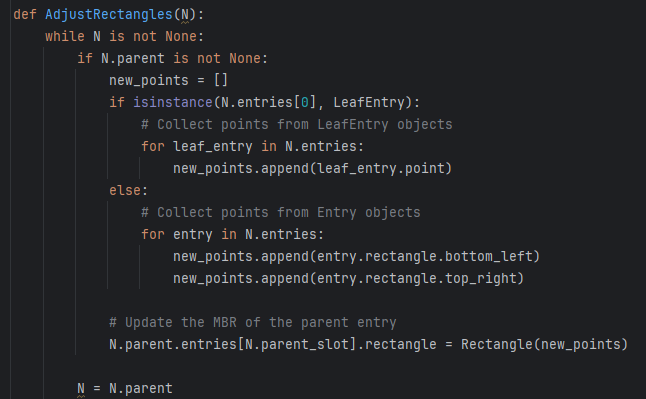
* **ChooseSubtree(rtree, r)**: βασίζεται στον ψευδοκώδικα της συνάρτησης ChooseSubtree που δίνεται στο original paper για το R\*-Tree:



Πιο συγκεκριμένα, δέχεται ως ορίσματα την νέα εγγραφή που είναι προς εισαγωγή στο δέντρο ως αντικείμενο της κλάσης LeafEntry και την λίστα που περιέχει τους κόμβους του δέντρου ώστε να επιλεγεί το κατάλληλο υποδέντρο για της. Επομένως, ξεκινώντας από τη ρίζα του αρχίζει μια επαναληπτική διαδικασία για την εύρεση του κόμβου-φύλλο που αποτελεί τον καταλληλότερο προορισμό για την νέα εγγραφή. Η σύγκληση της επαναληπτικής διαδικασίας βασίζεται στο εξής: Αν ο κόμβος δεν έχει ως παιδιά φύλλα, επιλέγεται κάθε φορά ο κόμβος που θα χρειαστεί να μεγαλώσει λιγότερο το εμβαδόν του MBR του. Αν ο κόμβος έχει παιδιά φύλλα, επιλέγεται ο κόμβος που θα ελαχιστοποιεί το εμβαδόν της τομής του νέου MBR του με τα MBR των γειτονικών κόμβων. Στο τέλος, επιστρέφει τον κόμβο-φύλλο του δέντρου στον οποίο θα πρέπει να γίνει η εισαγωγή της νέας εγγραφής.



* **AdjustRectangles(N)**: αποσκοπεί στην ενημέρωση των MBR όλων των κόμβων σε ένα δέντρο όταν έχει υπάρξει κάποια αλλαγή μέσα σε ένα κόμβο και πρέπει να επαναπροσαρμοστούν. Αυτή η διαδικασία είναι κρίσιμη για τη διατήρηση της ακρίβειας των MBR, που χρησιμοποιούνται για την αναζήτηση και την εισαγωγή νέων δεδομένων στο δέντρο. Πιο συγκεκριμένα, δέχεται ως όρισμα τον κόμβο που υπέστη κάποια αλλαγή και προσαρμόζει το MBR του κόμβου ώσετε να αντικατοπτρίζει την αλλαγή. Η συνάρτηση συνεχίζει αναδρομικά μέχρι την ρίζα, ώστε να διορθωθούν τα MBR όλων των προηγούμενων κόμβων που επηρεάζονται από την αρχική αλλαγή. Αν οι εγγραφές του κόμβου N είναι τύπου LeafEntry, συλλέγονται τα σημεία (δηλαδή, οι συντεταγμένες) όλων των LeafEntry που περιέχονται στον κόμβο.Διαφορετικά, αν οι εγγραφές του κόμβου N είναι τύπου Entry, συλλέγονται οι συντεταγμένες των κάτω αριστερών και άνω δεξιών γωνιών των MBR αυτών των εγγραφών. Αυτές οι συντεταγμένες αποτελούν τα όρια των MBR που περιέχουν τους απογόνους κόμβους. Με τα συγκεντρωμένα σημεία, η συνάρτηση δημιουργεί ένα νέο MBR.Αυτό καθορίζεται ως το MBR της εγγραφής του γονικού κόμβου που αντιστοιχεί στον κόμβο N. Μετά την ενημέρωση του MBR του γονικού κόμβου, ο τρέχων κόμβος N ενημερώνεται ώστε να είναι ο γονικός κόμβος και η διαδικασία επαναλαμβάνεται.



* **overflowTreatment(N, rtree, level)**: βασίζεται στον ψευδοκώδικα της συνάρτησης OverflowTreatment που δίνεται στο original paper για το R\*-Tree:



Πιο συγκεκριμένα, αναλαμβάνει τη διαχείριση της υπερφόρτωσης ενός κόμβου στο R\*-Tree, όταν δηλαδή ο αριθμός των εγγραφών υπερβεί τον μέγιστο επιτρεπόμενο. Δέχεται ως ορίσματα τον κόμβο που έχει γεμίσει, το επίπεδο αυτού του κόμβου και την λίστα με τους κόμβους του δέντρου. Η διαδικασία ξεκινά με τον έλεγχο αν ο υπερφορτωμένος κόμβος είναι η ρίζα του δέντρου. Σε αυτή την περίπτωση, η συνάρτηση προχωρά στη διάσπαση (split) του κόμβου σε δύο νέους κόμβους. Δημιουργείται ένας νέος κόμβος ρίζας που περιλαμβάνει εγγραφές για τους δύο νέους κόμβους, οι οποίοι γίνονται τα παιδιά της νέας ρίζας. Η λίστα των κόμβων του δέντρου ενημερώνεται ανάλογα, όπως και οι δείκτες που συνδέουν τους κόμβους μεταξύ τους.Αν δεν είναι η ρίζα, η συνάρτηση ελέγχει αν είναι η πρώτη φορά που καλείται για αυτό το επίπεδο του δέντρου, αυξάνοντας τη μεταβλητή overflow\_treatment\_level κατά ένα και καλώντας τη συνάρτηση ReInsert. Η ReInsert προσπαθεί να αναδιανείμει τις εγγραφές του υπερφορτωμένου κόμβου σε άλλα μέρη του δέντρου, ώστε να αποφευχθεί η διάσπαση και να επιτευχθεί καλύτερη κατανομή του φορτίου.Σε όλες τις άλλες περιπτώσεις, η συνάρτηση απλώς πραγματοποιεί διάσπαση του κόμβου.

Εάν ο κόμβος περιέχει εγγραφές φύλλων (LeafEntry), τότε οι νέες εγγραφές ανατίθενται σε δύο νέους κόμβους φύλλων. Εάν οι εγγραφές είναι εσωτερικές (Entry), τότε αυτές διαχωρίζονται σε δύο νέους κόμβους που αναπαριστούν ορθογώνιες περιοχές (bounding rectangles). Κατόπιν, οι δείκτες προς τους γονικούς κόμβους και άλλους σχετικούς κόμβους ενημερώνονται, ώστε να διατηρηθεί η δομή του δέντρου.

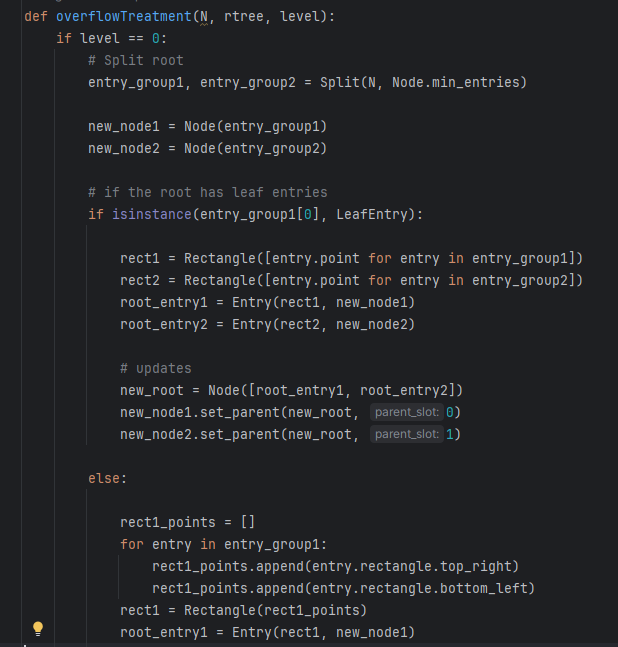
**(εναλλακτικό κείμενο)** : Εάν ο υπερφορτωμένος κόμβος (N) είναι η ρίζα του δέντρου (δηλαδή, level == 0): Καλείται η συνάρτηση Split για να διαχωρίσει τις εγγραφές του κόμβου σε δύο ομάδες (entry\_group1 και entry\_group2). Αυτό γίνεται ώστε να δημιουργηθούν δύο νέοι κόμβοι (new\_node1 και new\_node2), οι οποίοι θα αντικαταστήσουν τον αρχικό κόμβο.Αν οι εγγραφές είναι φύλλα (LeafEntry), τότε δημιουργούνται δύο νέα ορθογώνια (Rectangle) που περικλείουν τις εγγραφές της κάθε ομάδας ενώ δημιουργούνται επίσης δύο νέες εγγραφές ρίζας (root\_entry1 και root\_entry2) που περιέχουν τα νέα ορθογώνια και τους νέους κόμβους.Εάν οι εγγραφές δεν είναι φύλλα, οι νέες εγγραφές ρίζας δημιουργούνται με βάση τα όρια των ορθογωνίων που περιβάλλουν τις εγγραφές κάθε ομάδας. Δημιουργείται ένας νέος κόμβος ρίζας (new\_root) και ενημερώνονται οι δείκτες γονέα-παιδιού μεταξύ των νέων κόμβων και της νέας ρίζας.Τέλος, ο αρχικός κόμβος (N) αφαιρείται από το δέντρο και οι νέοι κόμβοι προστίθενται.

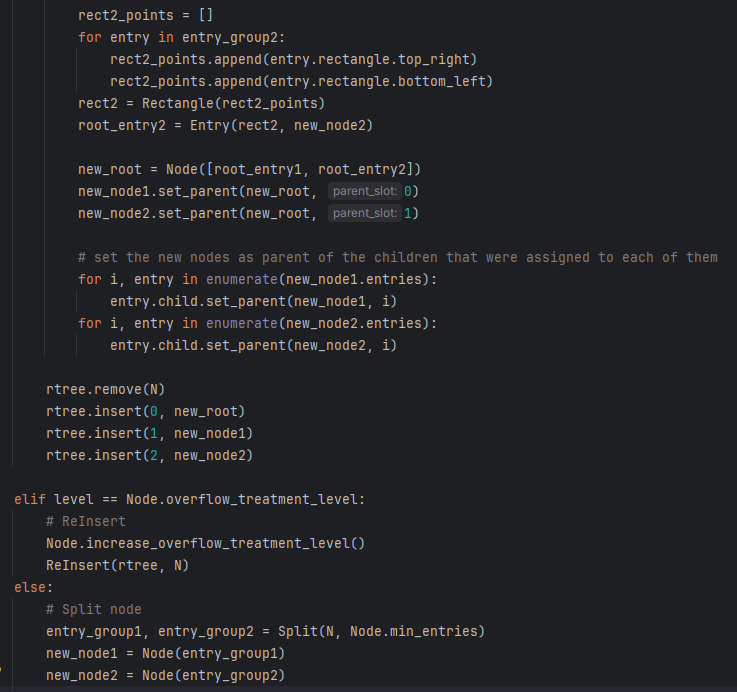
Εάν ο υπερφορτωμένος κόμβος δεν είναι η ρίζα και το επίπεδο του είναι ίσο με την τιμή της μεταβλητής overflow\_treatment\_level (level == Node.overflow\_treatment\_level):Αυξάνεται η τιμή της μεταβλητής overflow\_treatment\_level κατά ένα με την κλήση της Node.increase\_overflow\_treatment\_level().Έπειτα, καλείται η συνάρτηση ReInsert, η οποία προσπαθεί να αναδιατάξει τις εγγραφές του κόμβου (N) σε άλλους κόμβους του δέντρου, ελπίζοντας να αποφευχθεί η διάσπαση (split).

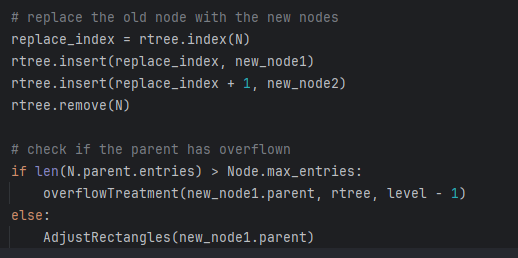
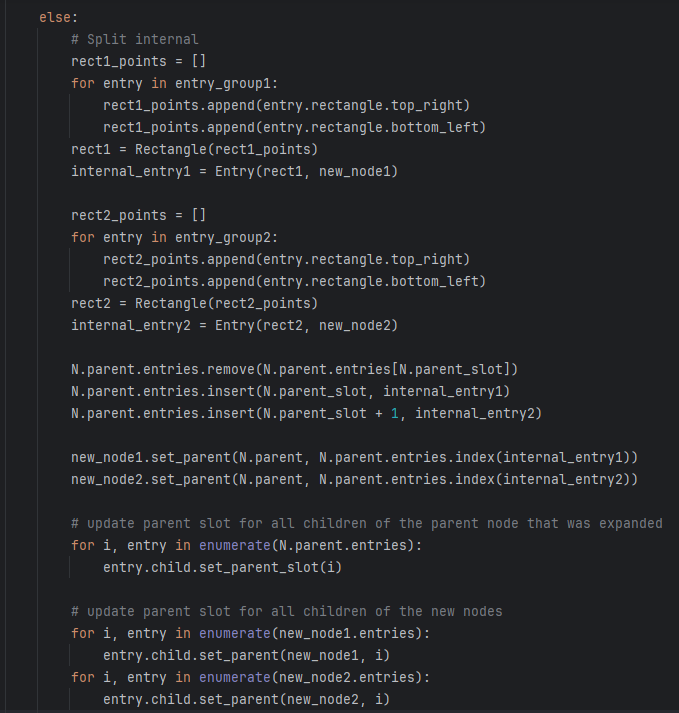
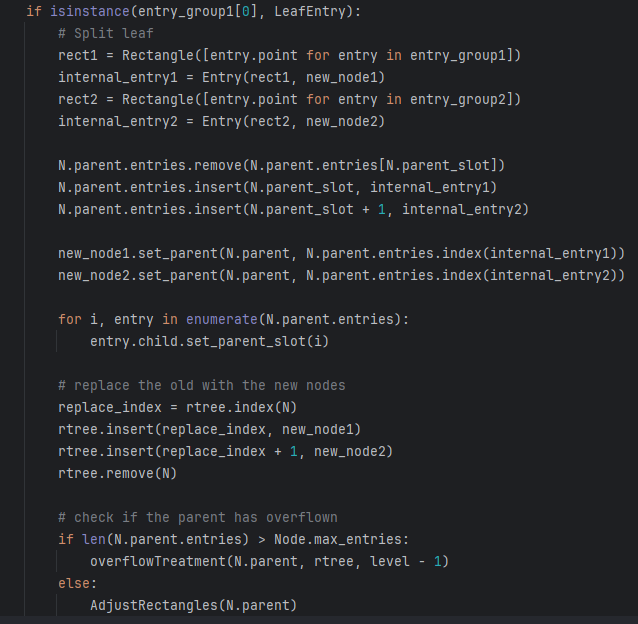
Για άλλες περιπτώσεις, όπου ο κόμβος δεν είναι η ρίζα και δεν γίνεται αναδιανομή:Ο κόμβος (N) διασπάται σε δύο νέους κόμβους (new\_node1 και new\_node2) με τη χρήση της Split, η οποία χωρίζει τις εγγραφές σε δύο ομάδες.Αν οι εγγραφές είναι φύλλα (LeafEntry), τότε δημιουργούνται νέες εσωτερικές εγγραφές (internal\_entry1 και internal\_entry2) που αντιπροσωπεύουν τα ορθογώνια που περικλείουν τις εγγραφές της κάθε ομάδας.Οι νέες εγγραφές αντικαθιστούν την παλιά εγγραφή στο γονικό κόμβο και οι δείκτες γονέα-παιδιού ενημερώνονται αναλόγως.Τέλος, οι παλιές εγγραφές αντικαθίστανται στο δέντρο από τους νέους κόμβους και οι δείκτες γονέα-παιδιού ανανεώνονται.

Η διαδικασία ολοκληρώνται , ελέγχοντας αν ο γονικός κόμβος έχει υπερχειλίσει. Αν ναι, καλείται η overflowTreatment για τον γονικό κόμβο, αλλιώς καλείται η AdjustRectangles για να προσαρμόσει τα ορθογώνια που περικλείουν τις εγγραφές.

**Τελος εναλλακτικού κειμένου**





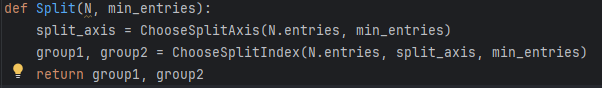


Χρησιμοποιεί δύο κύριες βοηθητικές συναρτήσεις: τη Split και τη ReInsert που περιγράφονται παρακάτω:

* **Split(N, min\_entries)**: βασίζεται στον ψευδοκώδικα της συνάρτησης Split που δίνεται στο original paper για το R\*-Tree:



Πιο συγκεκριμένα, χωρίζει έναν κόμβο σε δύο νέους. Δέχεται ως ορίσματα τον κόμβο με Μ+1 εγγραφές που χρειάζεται να διασπαστεί και τον ελάχιστο επιτρεπτό αριθμό εγγραφών που μπορεί να περιέχει ένας κόμβος. Αρχικά για τον υπολογισμό του άξονα διάσπασης καλεί την συνάρτηση ChooseSplitAxis, δίνοντάς της ως όρισμα τα Μ+1 entries του κόμβου. Στην συνέχεια, για τον υπολογισμό του σημείου διάσπασης καλεί την ChooseSplitIndex με ορίσματα το αποτέλεσμα της προηγούμενης συνάρτησης και τα Μ+1 entries του κόμβου. Στο τέλος επιστρέφει ως αποτέλεσμα τα δύο σύνολα από entries που παράγει η ChooseSplitIndex.



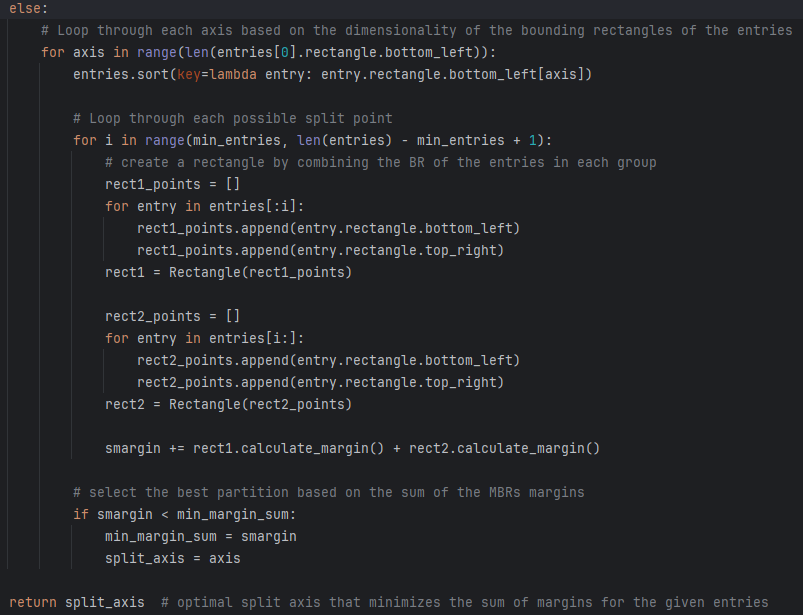
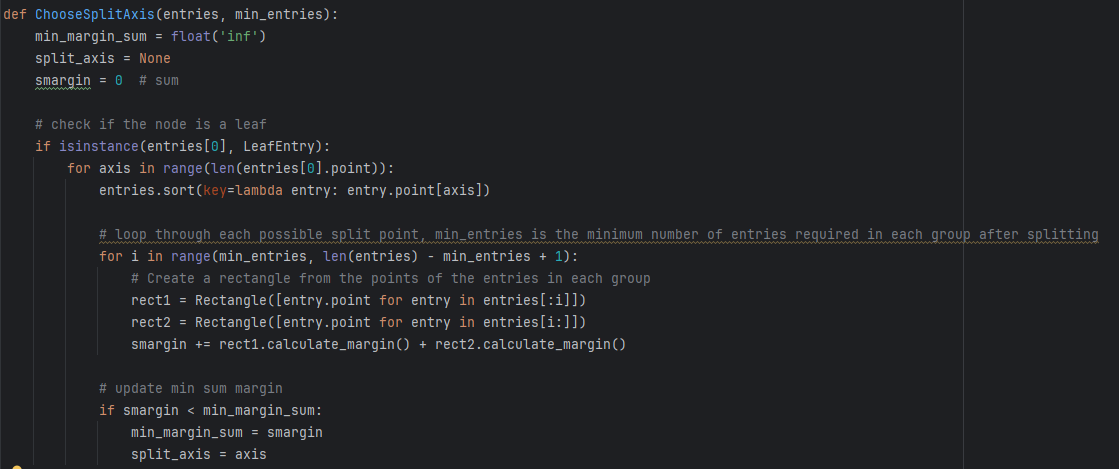
Η λειτουργία της βασίζεται σε δύο βοηθιτηκές συναρτήσεις:

* **ChooseSplitAxis(entries, min\_entries)**: βασίζεται στον ψευδοκώδικα της συνάρτησης ChooseSplitAxis που δίνεται στο original paper για το R\*-Tree:



Πιο συγκεκριμένα, χρησιμοποιείται για να επιλέξει τον καλύτερο άξονα για τη διαίρεση των εγγραφών σε δύο ομάδες, κατά τη διαδικασία διαχείρισης της υπερχείλισης στους κόμβους.Δέχεται ως ορίσματα μία λίστα από εγγραφές δέντρου (είτε Entry είτε LeafEntry) και τον ελάχιστο επιτρεπτό αριθμό εγγραφών που μπορεί να περιέχει ένας κόμβος. Η διαδικασία επιλογής του καταλληλότερου άξονα διαχωρισμού περιλαμβάνει τα εξής βήματα:

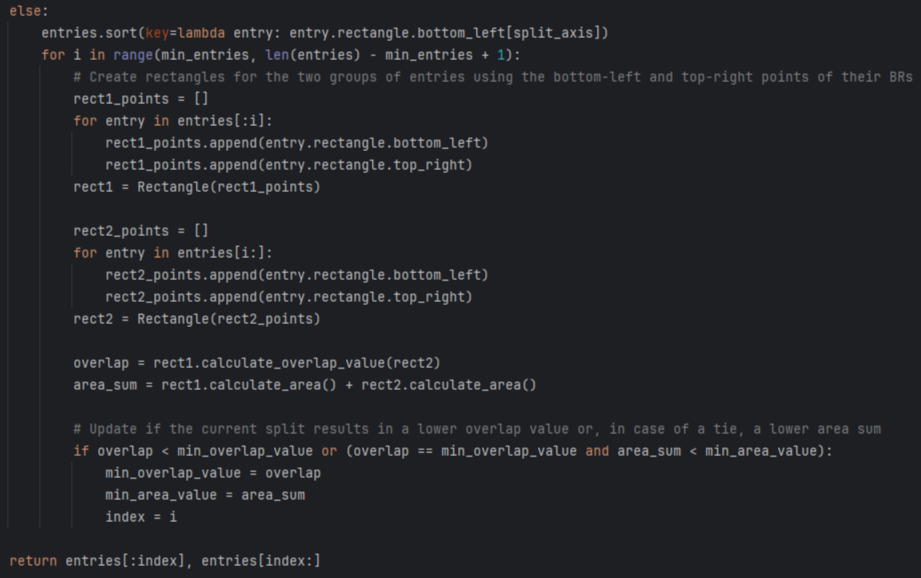
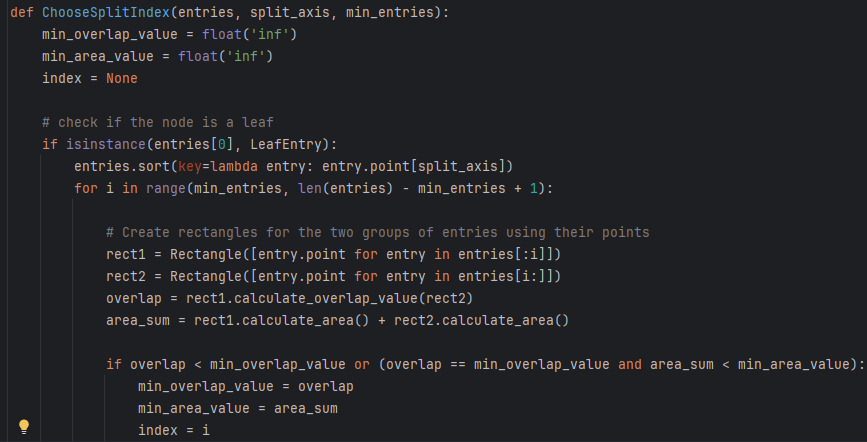
Για κάθε άξονα των συντεταγμένων, οι εγγραφές ταξινομούνται σε αύξουσα σειρά με βάση τη συντεταγμένη τους σε αυτόν τον άξονα. Στη συνέχεια, για κάθε πιθανή διαίρεση των ταξινομημένων εγγραφών, υπολογίζεται το άθροισμα των περιμέτρων των MBR για τα δύο υποσύνολα που προκύπτουν από τη διαίρεση. Μια διαίρεση θεωρείται έγκυρη αν και τα δύο υποσύνολα έχουν τουλάχιστον τον ελάχιστο αριθμό εγγραφών. Κατά τη διάρκεια αυτής της διαδικασίας, η συνάρτηση παρακολουθεί και καταγράφει το index του άξονα που προσφέρει τη διαίρεση με το μικρότερο άθροισμα περιμέτρων MBR. Αυτός ο άξονας θεωρείται ως ο καταλληλότερος για τη διαίρεση, καθώς ελαχιστοποιεί την αύξηση της περιμέτρου των MBR, κάτι που είναι σημαντικό για τη διατήρηση της αποδοτικότητας του R\*-Tree. Ο τρόπος με τον οποίο υπολογίζονται τα MBR και ταξινομούνται οι εγγραφές εξαρτάται από το αν οι εγγραφές είναι τύπου Entry ή LeafEntry. Για το λόγο αυτό, χρησιμοποιείται μια συνθήκη if isinstance() για να διαφοροποιήσει τη μέθοδο υπολογισμού. Τέλος, η συνάρτηση επιστρέφει το index του άξονα που βρέθηκε ως ο καλύτερος για τη διαίρεση, δηλαδή αυτόν που ελαχιστοποιεί το άθροισμα των περιμέτρων των MBR των δύο υποσυνόλων.



* **ChooseSplitIndex(entries, split\_axis, min\_entries)**: βασίζεται στον ψευδοκώδικα της συνάρτησης ChooseSplitIndex που δίνεται στο original paper για το R\*-Tree:



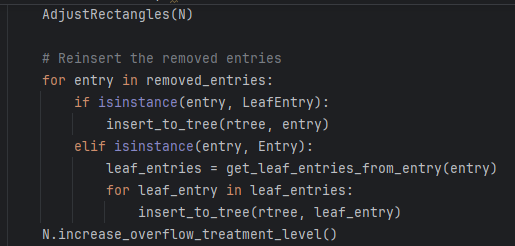
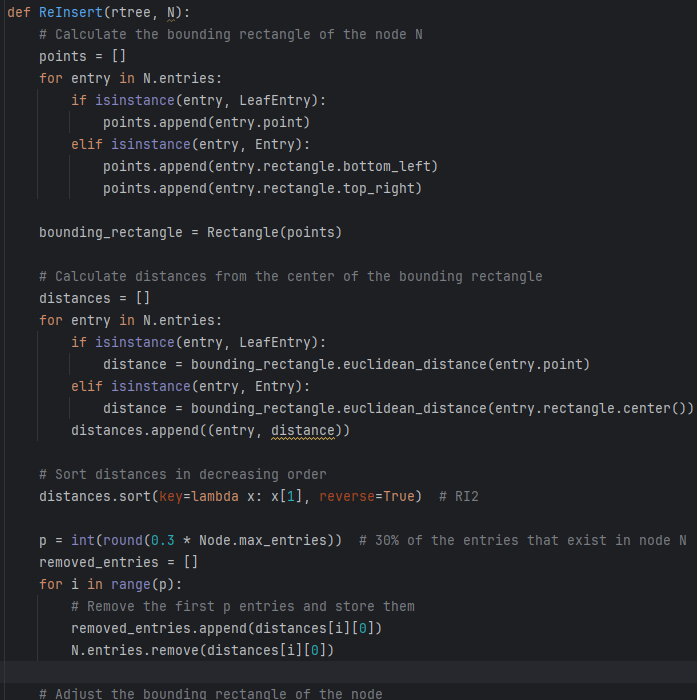
Πιο συγκεκριμένα, έχει ως στόχο την εύρεση του βέλτιστου σημείου διαχωρισμού μιας λίστας από εγγραφές σε δύο ομάδες κατά μήκος ενός καθορισμένου άξονα. Αυτό το σημείο διαχωρισμού επιλέγεται με βάση δύο κύρια κριτήρια: την ελαχιστοποίηση της επικάλυψης των ορθογωνικών περιοχών που δημιουργούνται από τις δύο ομάδες και την ελαχιστοποίηση του συνολικού εμβαδού αυτών των περιοχών. Δέχεται ως ορίσματα μία λίστα από εγγραφές δέντρου (είτε Entry είτε LeafEntry), το index του άξονα πάνω στον οποίο θα εκτελεστεί το split και τον ελάχιστο επιτρεπτό αριθμό εγγραφών που μπορεί να περιέχει ένας κόμβος. Αρχικά,, ταξινομεί τις εγγραφές με βάση τις συντεταγμένες του επιλεγμένου άξονα και για κάθε επιτρεπτή διαμέριση υπολογίζει το εμβαδόν την τομής των MBR των δύο συνόλων. Στο τέλος επιστρέφει τα δύο σύνολα της διαμέρισης η οποία ελαχιστοποιεί το εμβαδόν την τομής των MBR που είναι δύο λίστες: εγγραφές πριν από το βέλτιστο σημείο διαχωρισμού και εγγραφές μετά από αυτό. Αν υπάρχουν περισσότερες από μία διαμερίσεις που ελαχιστοποιούν το εμβαδόν της τομής των MBR, τότε επιλέγεται αυτή με το ελάχιστο άθροισμα εμβαδού.



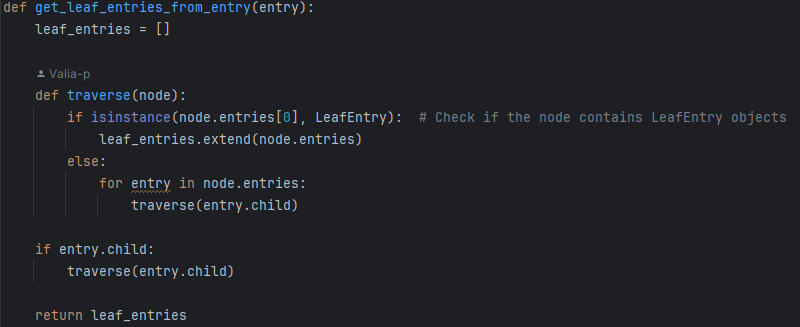
* **ReInsert(rtree, N)**: βασίζεται στον ψευδοκώδικα της συνάρτησης ReInsert που δίνεται στο original paper για το R\*-Tree:



Πιο συγκεκριμένα, αποσκοπεί στη βελτιστοποίηση της δομής ενός R\*-Tree με την αναδιάταξη ορισμένων εγγραφών από έναν κόμβο που έχει υπερφορτωθεί. Η διαδικασία αυτή βοηθά στη μείωση της ανάγκης για διάσπαση του κόμβου, διατηρώντας παράλληλα την αποτελεσματικότητα του δέντρου στην αναζήτηση. Για κάθε εγγραφή στον κόμβο N, συγκεντρώνει τα σημεία που χρειάζονται για τον υπολογισμό του MBR. Αν η εγγραφή είναι τύπου LeafEntry, χρησιμοποιείται το σημείο της εγγραφής, ενώ αν είναι τύπου Entry, χρησιμοποιούνται οι συντεταγμένες της κάτω αριστερής και της πάνω δεξιάς γωνίας του MBR της εγγραφής. Στη συνέχεια, υπολογίζονται οι αποστάσεις από το κέντρο του MBR προς τα σημεία των εγγραφών (για LeafEntry) ή προς το κέντρο MBR της κάθε εγγραφής (για Entry). Οι αποστάσεις αυτές καταγράφονται μαζί με τις αντίστοιχες εγγραφές. Έπειτα, οι εγγραφές ταξινομούνται κατά φθίνουσα σειρά με βάση τις αποστάσεις αυτές. Από την κορυφή της ταξινομημένης λίστας, αφαιρείται το 30% των εγγραφών (που προσδιορίζεται από το Node.max\_entries), οι οποίες θεωρούνται οι πλέον απομακρυσμένες από το κέντρο του MBR. Αυτές οι εγγραφές αφαιρούνται από τον κόμβο N και αποθηκεύονται σε μια λίστα removed\_entries εφόσον δεν πρέπει να βρίσκονται στο ίδιο ορθογώνιο. Μετά την αφαίρεση των εγγραφών, το MBR του κόμβου N αναπροσαρμόζεται στις νέες του διαστάσεις. Οι εγγραφές που αφαιρέθηκαν επανεισάγονται στο δέντρο. Αν η εγγραφή είναι τύπου LeafEntry, εισάγεται απευθείας, ενώ αν είναι τύπου Entry, οι εγγραφές των φύλλων εισάγονται μία προς μία. Για την απόκτηση των εγγραφών αυτών χρησιμοποιείται η βοηθητική συνάρτηση get\_leaf\_entries\_from\_entry. Στο τέλος, αυξάνεται το overflow\_treatment\_level για να υποδείξει ότι ο κόμβος N έχει υποστεί ReInsert.



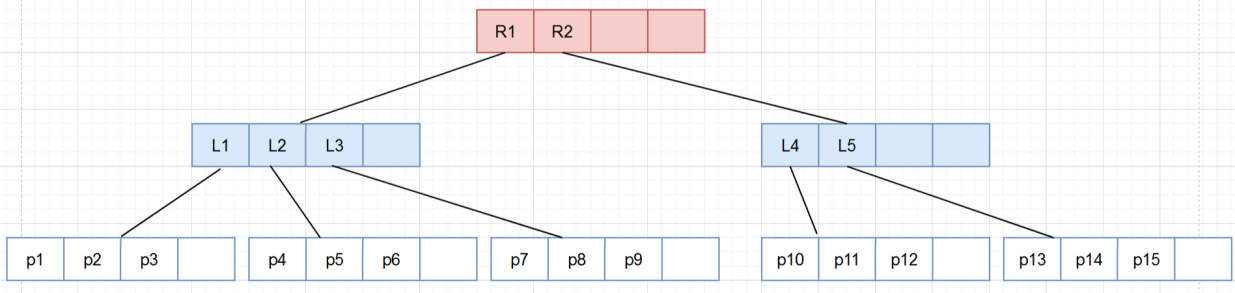
* **get\_leaf\_entries\_from\_entry(entry)**: χρησιμοποιείται για την ανάκτηση των φύλλων από έναν εσωτερικό κόμβο . Αυτή η διαδικασία πραγματοποιείται αναδρομική από τις εγγραφές των παιδιών του κόμβου, συλλέγοντας όλες τις εγγραφές τύπου LeafEntry.



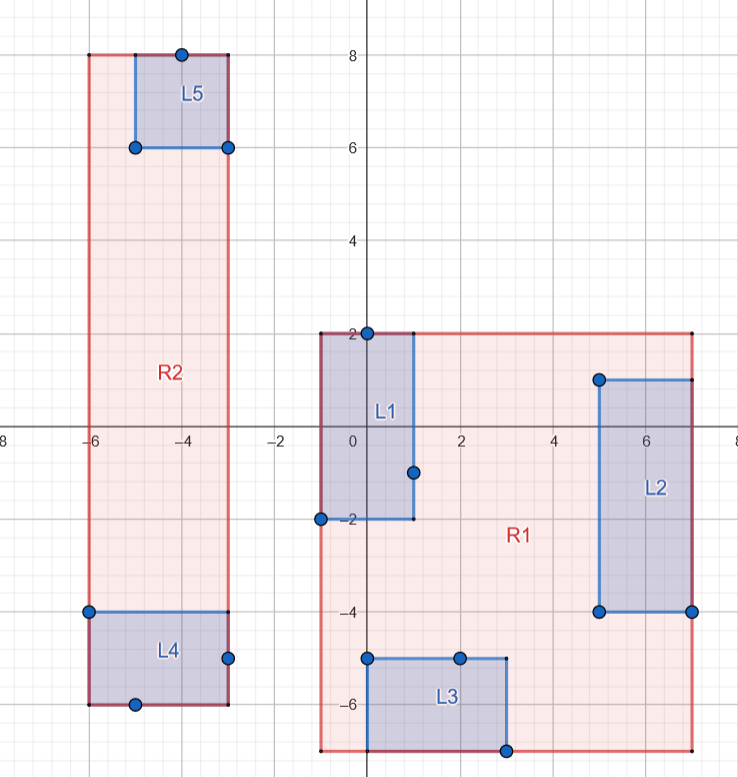
**Εκτέλεση Παραδείγματος:**

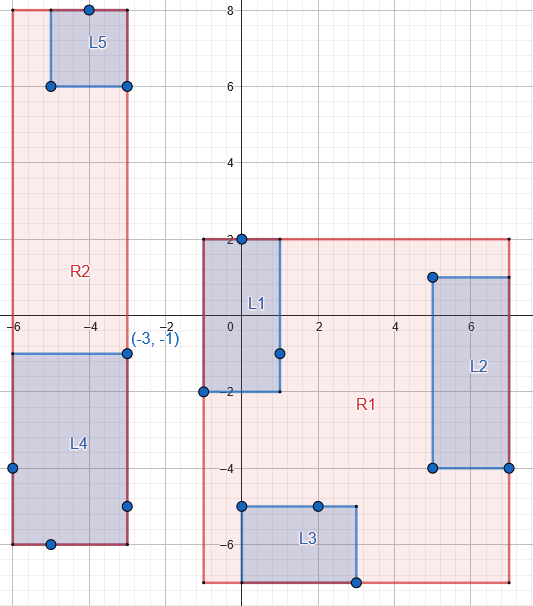
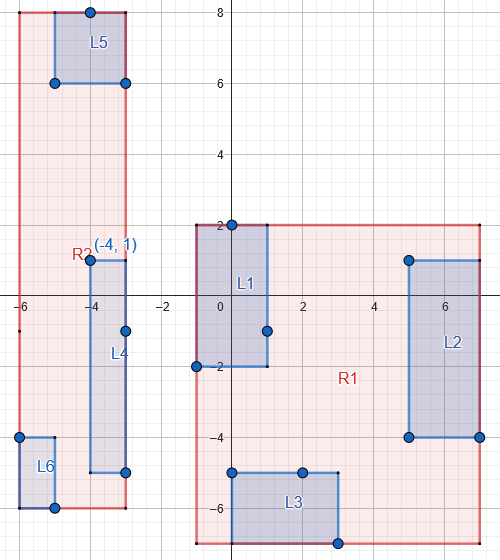
Για την καλύτερη κατανόηση της λειτουργίας της εισαγωγής στο R\*-tree, παρουσιάζεται ένα παράδειγμα διαδοχικών εισαγωγών για ένα μικρό δέντρο που φτιάξαμε(αρχείο testing.py) και είναι αποθηκευμένο στο indexfile1.xml. Δεν περιέχει αντίστοιχο datafile.

Η αρχική μορφή του δέντρου είναι η εξής:

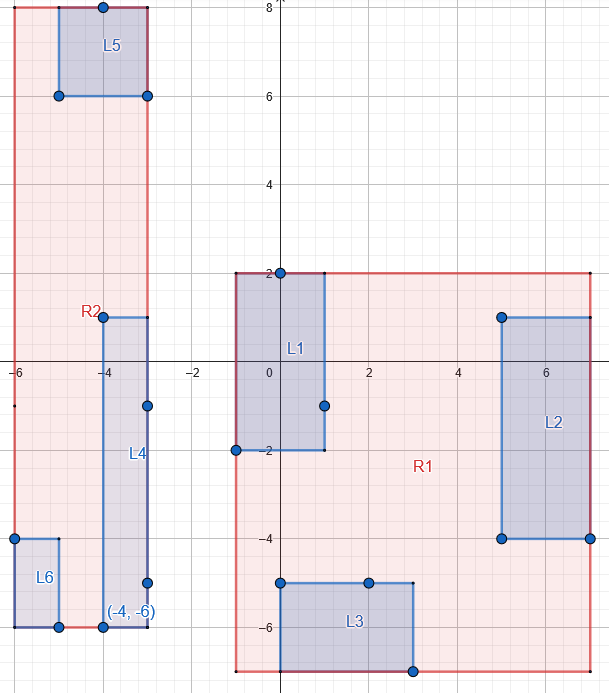
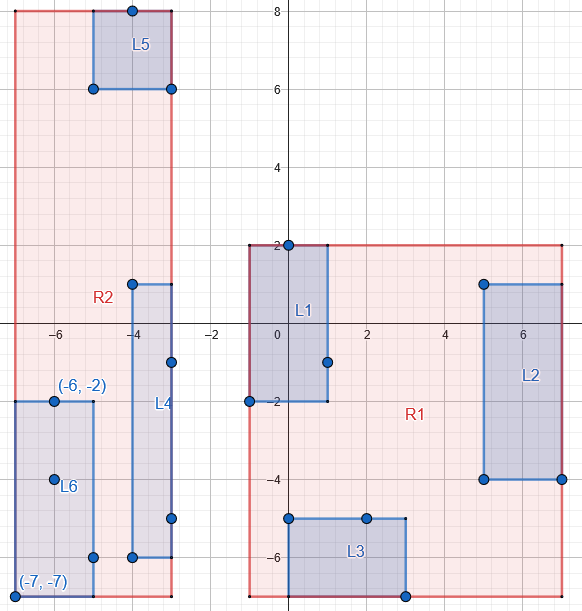


Max\_entries = 4, Min\_entries = 2

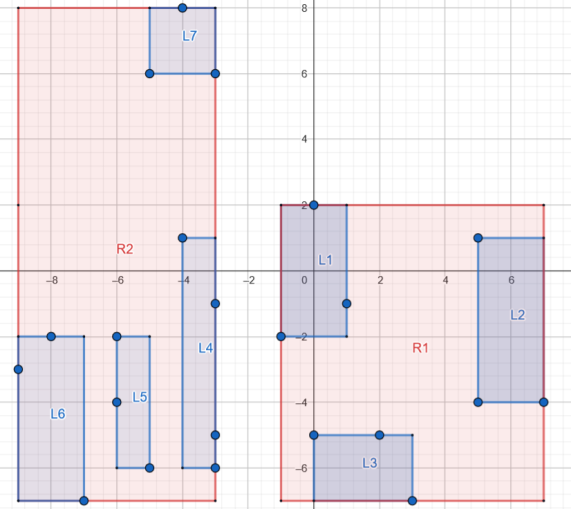
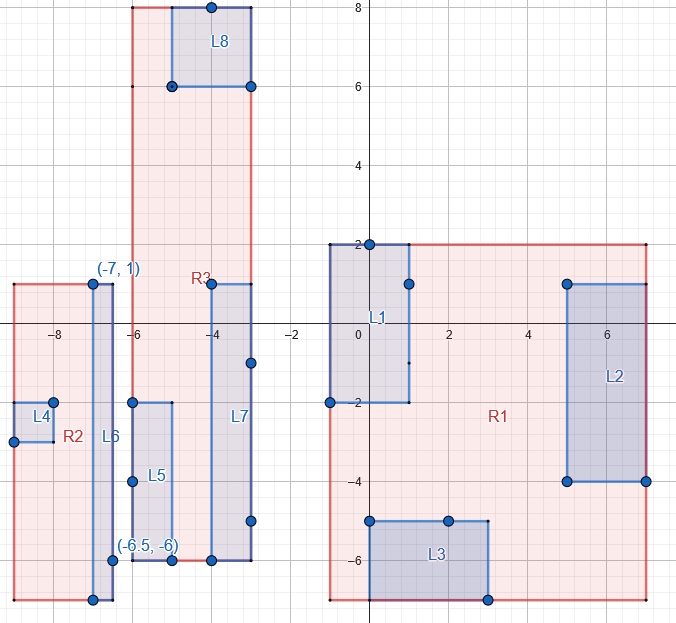


insert(-3,-1) insert(-4,1)  

**insert (-4,-6) insert (-6,-2) and insert(-7,-7)**

** **

**insert (-8,-2) and insert(-9,-3) insert (-7,-1) and insert(-6.5,-6)**

** **

Εξετάστηκε επίσης ο χρόνος κατασκεύης του καταλόγου με εισαγωγή των στοιχείων ένα ένα για διαφορετικό αριθμό records κάθε φορά. Με αυτό τον τρόπο γίνεται εμφανής η προοδευτική αύξηση του χρόνου κατασκευής όσο αυξάνεται ο αριθμός των εγγραφών στο datafile. Ακολουθεί πίνακας ενδεικτικών αποτελεσμάτων:

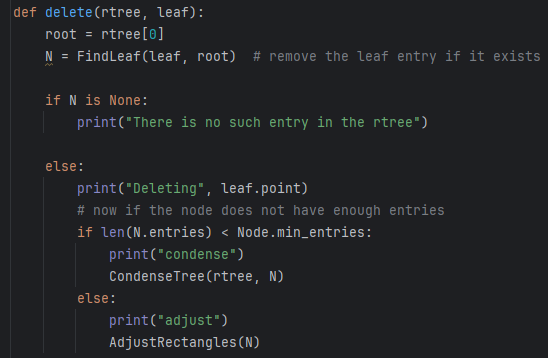
|  |  |
| --- | --- |
| **Αριθμός Εγγραφών στο datafile** | **Χρόνος Κατεσκεύης R\*-tree** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

5.Διαγραφή Εγγραφής

Η υλοποίηση της διαγραφής εγγραφής στο rtree βρίσκεται στο αρχείο delete.py και η ορθή λειτουργία του βασίζεται στις παρακάτω συναρτήσεις (παρουσιάζονται με τη σειρά που καλούνται) και υλοποιήθηκε βάση του ψευδοκώδικα ης συνάρτησης Delete που δίνεται στο original paper για το R-Tree:



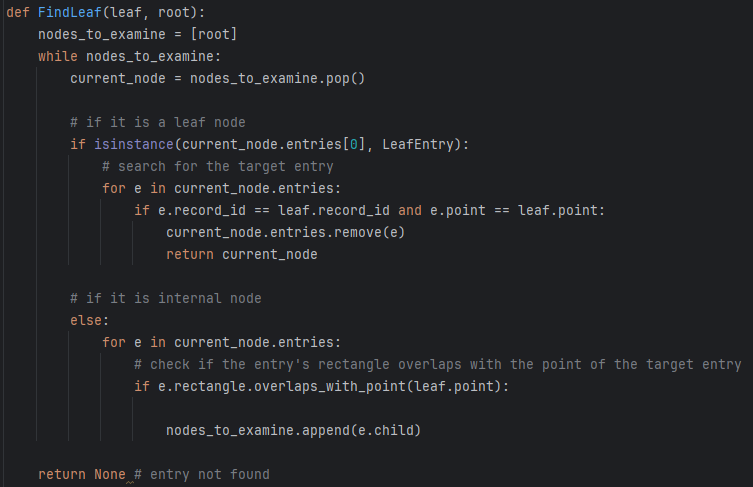
* **delete(rtree, leaf)**: δέχεται ως ορίσματα την λίστα με τους κόμβους του δέντρου, rtree, και την εγγραφή που θέλουμε να διαγράψουμε. Αρχικά καλεί την συνάρτηση FindLeaf για να εντοπίσει τον κόμβο N που περιέχει την εγγραφή. Αν η εγγραφή δεν βρεθεί εκτυπώνει μήνυμα "There is no such entry in the rtree" και σταματά. Αν η εγγραφή υπάρχει μέσα στο δέντρο, τη διαγράφει και έπειτα ελέγχει αν ο κόμβος N έχει λιγότερες από τις ελάχιστες απαιτούμενες εγγραφές. Αν ισχύει αυτό καλει τη συνάρτηση CondenseTree δίνοντας ως όρισμα τον κόμβο που την εμπεριέχει, διαφορετικά καλεί την AdjustRectangles ώστε να ενημερωθούν τα MBRs των κόμβων που επηρεάστηκαν από την διαγραφή της εγγραφής.



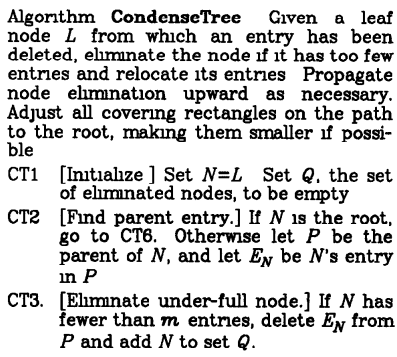
* **FindLeaf(leaf, root)**: αναζητά και διαγράφει το φύλλο από το δέντρο και βασίζεται στον ψευδοκώδικα της συνάρτησης FindLeaf που δίνεται στο original paper για το R-Tree.

****

Πιο συγκεκριμένα, δέχεται ως ορίσματα την εγγραφή του δέντρουπου θέλουμε να διαγράψουμε και την ρίζα του R\*-Tree. Η διαδικασία ξεκινάει με τη δημιουργία μιας λίστας κόμβων προς εξέταση (nodes\_to\_examine), η οποία αρχικά περιέχει μόνο τη ρίζα του δέντρου. Στη συνέχεια, η συνάρτηση επαναλαμβάνει τις εξής ενέργειες μέχρι να εξαντληθούν οι κόμβοι προς εξέταση: εξάγει τον τελευταίο κόμβο από τη λίστα και τον αποθηκεύει στη μεταβλητή current\_node.Αν ο τρέχων κόμβος είναι φύλλο, δηλαδή οι εγγραφές του είναι αντικείμενα τύπου LeafEntry, η συνάρτηση διατρέχει τις εγγραφές του κόμβου και αναζητά την εγγραφή που ταιριάζει με την εγγραφή που αναζητάμε(με βάση τα record\_id και point). Αν βρεθεί η αντίστοιχη εγγραφή, αφαιρείται από τον κόμβο και η συνάρτηση επιστρέφει τον κόμβο στον οποίο βρέθηκε η εγγραφή.Αν ο τρέχων κόμβος είναι εσωτερικός, διατρέχει τις εγγραφές του κόμβου και για κάθε εγγραφή ελέγχει αν το MBR της εγγραφής επικαλύπτεται με το σημείο της εγγραφής που αναζητείται χρησιμοποιώντας τη μέθοδο overlaps\_with\_point. Αν υπάρχει επικάλυψη, το παιδί της εγγραφής προστίθεται στη λίστα nodes\_to\_examine για μελλοντική εξέταση.Αν εξαντληθούν οι κόμβοι στη λίστα και δεν έχει βρεθεί η εγγραφή, η συνάρτηση επιστρέφει None, υποδεικνύοντας ότι η εγγραφή δεν βρέθηκε στο δέντρο.

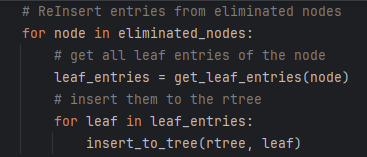
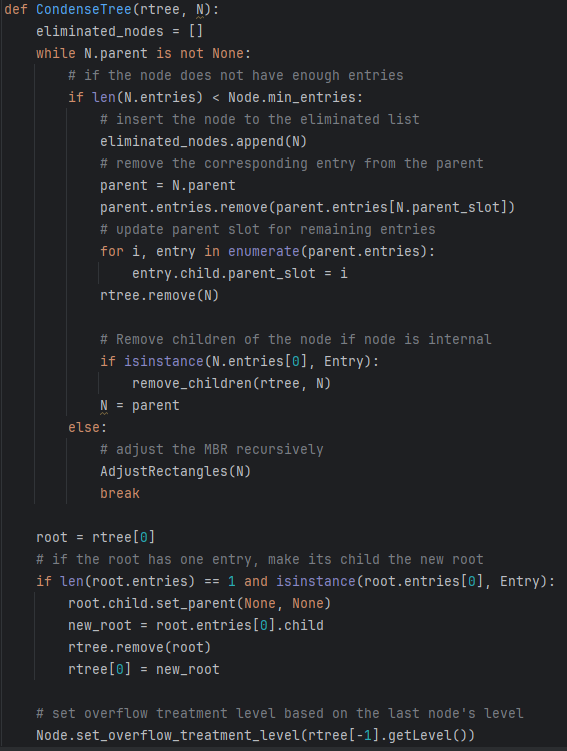


* **CondenseTree(rtree, N)**: συμπυκνώνει το δέντρο μετά από διαγραφή για να εξασφαλίσει ότι όλοι οι κόμβοι πληρούν τις ελάχιστες απαιτήσεις εγγραφών και βασίζεται στον ψευδοκώδικα της CondenseTreeπου δίνεται στο original paper για το R-Tree.



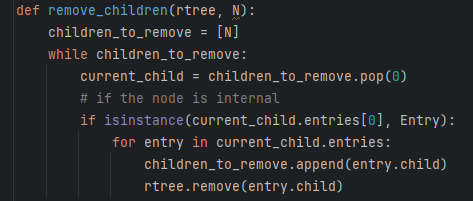


Πιο συγκεκριμένα, δέχεται ως όρισμα τον κόμβο Node που έχει διαγραφθεί και βρίσκει και επιστρέφει μία λίστα με όλα τα LeafEntries που έχουν ως πρόγονό τους τον συγκεκριμένο κόμβο. Αρχικά, δημιουργείται μια κενή λίστα (eliminated\_nodes) για την αποθήκευση των κόμβων που θα αφαιρεθούν. Στη συνέχεια, ξεκινώντας από τον κόμβο N, η συνάρτηση επαναλαμβάνεται μέχρι να φτάσει στη ρίζα ή μέχρι να βρεθεί κόμβος που πληροί το ελάχιστο πλήθος εγγραφών. Αν ο τρέχων κόμβος N έχει λιγότερες εγγραφές από το ελάχιστο επιτρεπτό, προστίθεται στη λίστα eliminated\_nodes, ενώ ο γονικός κόμβος του N ενημερώνεται αφαιρώντας την εγγραφή που αντιστοιχεί στον κόμβο N, ενώ ταυτόχρονα ενημερώνονται και οι θέσεις των υπόλοιπων παιδιών του γονέα. Ο κόμβος N αφαιρείται από το δέντρο, και αν είναι εσωτερικός, αφαιρούνται επίσης τα παιδιά του μέσω της συνάρτησης remove\_children. Η μεταβλητή N ενημερώνεται ώστε να αναφέρεται στον γονικό κόμβο, και ο βρόχος συνεχίζεται. Αν ο κόμβος N έχει επαρκές πλήθος εγγραφών, καλείται η συνάρτηση AdjustRectangles(περιγράφηκε στην εισαγωγή εγγραφής) για την προσαρμογή των MBR των προγόνων του, και ο βρόχος διακόπτεται. Μετά το τέλος του βρόχου, η συνάρτηση ελέγχει τη ρίζα του δέντρου. Αν η ρίζα έχει μόνο μία εγγραφή και είναι εσωτερικός, τότε το παιδί της ρίζας γίνεται η νέα ρίζα του δέντρου. Το overflow treatment level του δέντρου ενημερώνεται με βάση το επίπεδο του τελευταίου κόμβου στο δέντρο. Τέλος, οι εγγραφές των κόμβων που εξαλείφθηκαν επανεισάγονται στο δέντρο. Η συνάρτηση get\_leaf\_entries χρησιμοποιείται για να συλλέξει όλες τις εγγραφές φύλλων από τους κόμβους που εξαλείφθηκαν και εισάγονται στο δέντρο χρησιμοποιώντας τη συνάρτηση insert\_to\_tree.

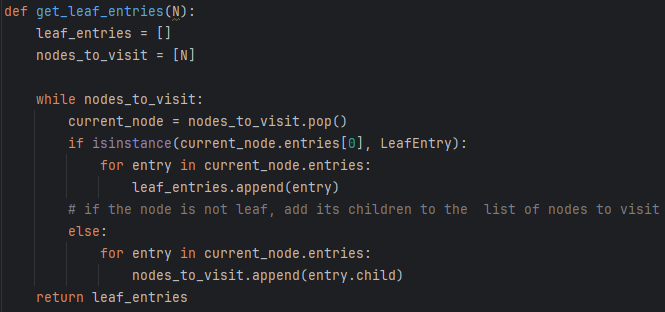


Συνοπτικά, η λειτουργία της βασίζεται στις παρακάτω βοηθητικές συναρτήσεις:

* **remove\_children(rtree, N)**: αφαιρεί όλους τους κόμβους παιδιά ενός εσωτερικού κόμβου N από το δέντρο. Αρχικοποιεί μια λίστα, children\_to\_remove, με τον κόμβο N και επαναληπτικά αφαιρεί τον πρώτο κόμβο της λίστας, ελέγχοντας αν είναι εσωτερικός κόμβος. Αν ναι, προσθέτει τα παιδιά του στη λίστα και τα αφαιρεί από το rtree.



* **get\_leaf\_entries(N)**: συλλέγει όλες τις εγγραφές φύλλων από έναν κόμβο N. Αρχικοποιεί μια λίστα, nodes\_to\_visit, με τον κόμβο N και επαναληπτικά αφαιρεί τον τελευταίο κόμβο της λίστας. Αν ο τρέχων κόμβος είναι φύλλο, προσθέτει τις εγγραφές του στη λίστα leaf\_entries. Αν ο τρέχων κόμβος δεν είναι φύλλο, προσθέτει τα παιδιά του στη λίστα nodes\_to\_visit. Στο τέλος, επιστρέφει τη λίστα leaf\_entries.

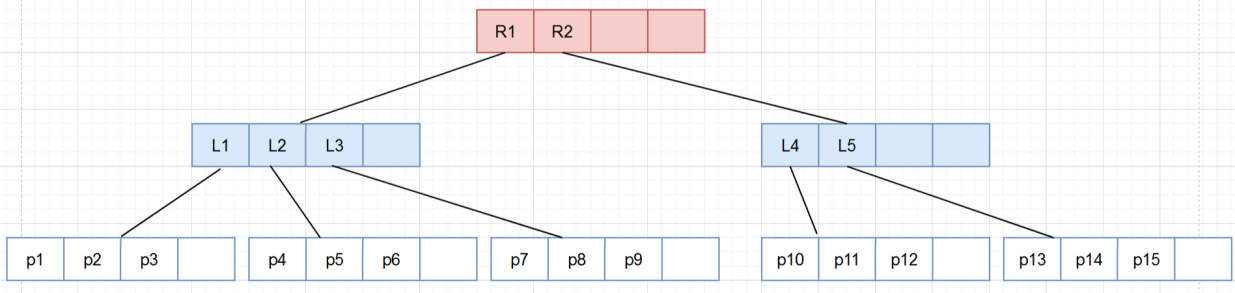


* **insert\_to\_tree(rtree, r)**: αναλύθηκε στο κομμάτι της εισαγωγής εγγραφής.

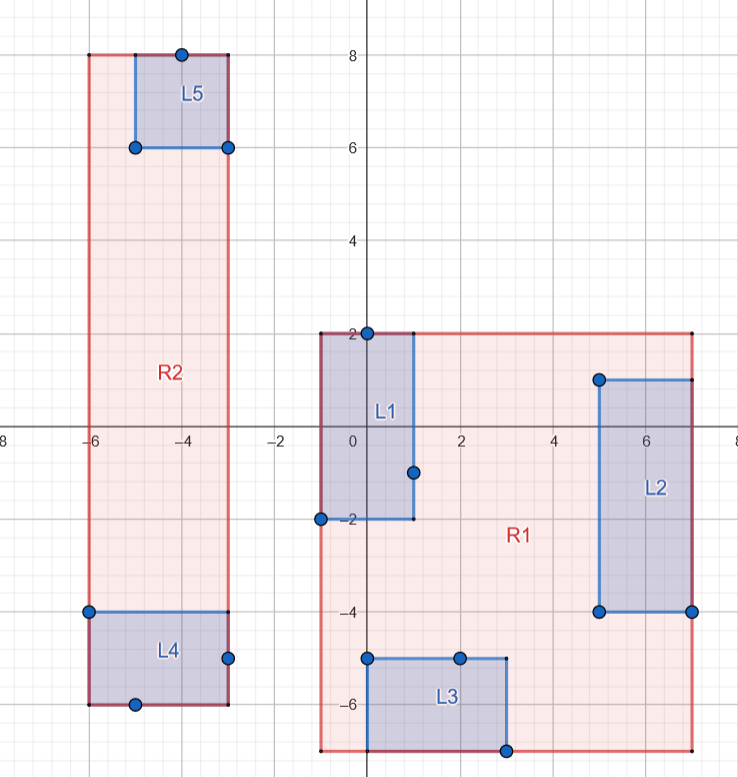
**Εκτέλεση Παραδείγματος:**

Για την καλύτερη κατανόηση της λειτουργίας της διαγραφής στο R\*-tree, παρουσιάζεται ένα παράδειγμα διαδοχικών διαγραφών για ένα μικρό δέντρο που φτιάξαμε(αρχείο testing.py) και είναι αποθηκευμένο στο indexfile1.xml. Δεν περιέχει αντίστοιχο datafile.

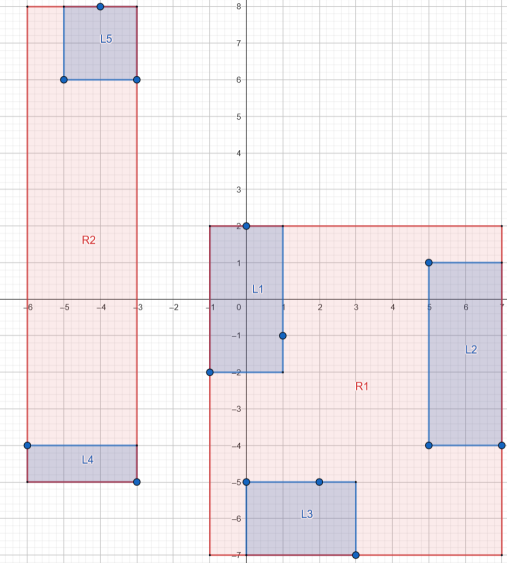
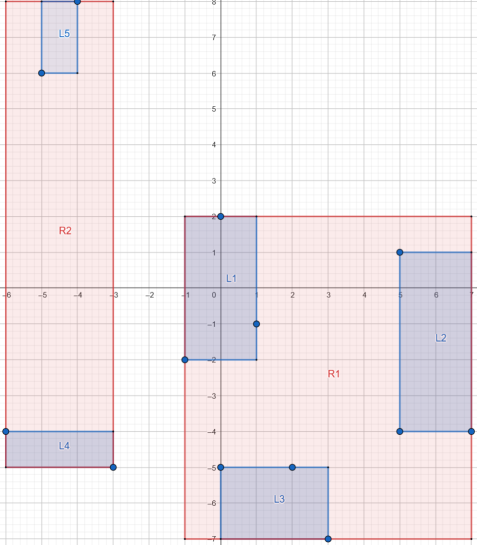
Η αρχική μορφή του δέντρου είναι η εξής:



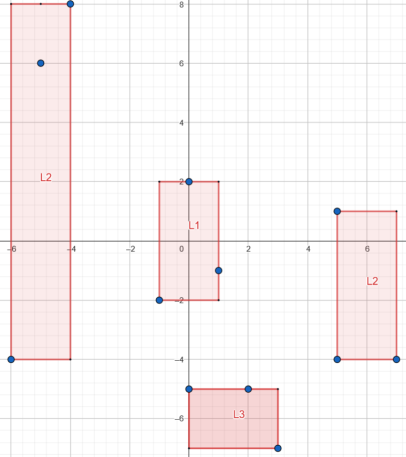
Max\_entries = 4, Min\_entries = 2



**delete(-5,-6) delete(-3,6)**

** **

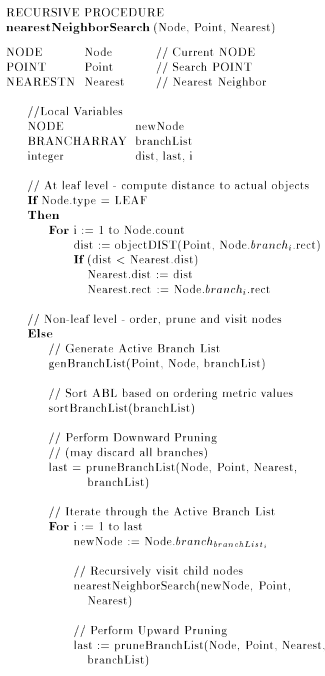
**delete(-3,-5)**

****

### **Ερωτήματα πάνω στο R\* - tree**

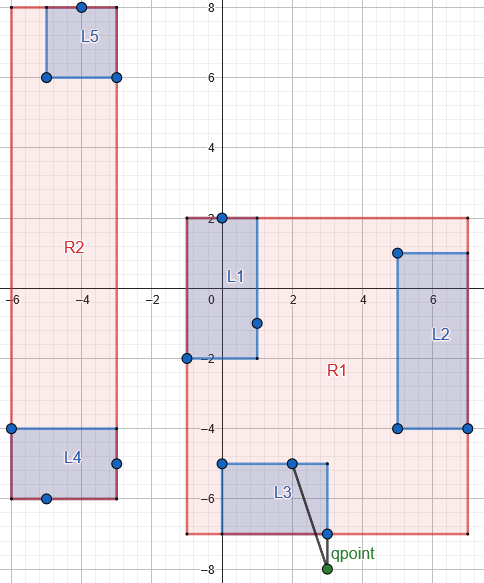
* Range Query
* KNN

Η δομή που υλοποιήσαμε στο αρχείο **KNN.py** υποστηρίζει ερωτήμα k πλησιέστερων γειτόνων και βασίζεται στο παρακάτω κώδικα :



Πιο συγκεκριμένα, η συνάρτηση **knn(root, qpoint, k)** είναι υπεύθυνη για τον υπολογισμό των k πλησιέστερων σημείων σε ένα σημείο ενδιαφέροντος(qpoint) όπου qpoint σημείο τύπου (0,….,0) για τις n διαστάσεις. Για την υλοποίηση της συνάρτησης, χρησιμοποιούνται δύο δομές δεδομένων: η ουρά προτεραιότητας **pq** και το σύνολο **visited**. Η pq αποθηκεύει και οργανώνει κόμβους /σημεία με βάση την απόστασή τους από το qpoint. Αυτή η ουρά εξασφαλίζει ότι τα πιο κοντινά στοιχεία επεξεργάζονται πρώτα. Κάθε στοιχείο της ουράς είναι μια τετράδα (distance, count, node\_or\_point, is\_leaf), όπου το distance είναι η απόσταση του στοιχείου από το qpoint, το count είναι ένας μετρητής που διασφαλίζει τη σωστή σειρά για στοιχεία με την ίδια απόσταση, το node/point αναφέρεται στον κόμβο/σημείο που εξετάζεται, και το is\_leaf είναι ένας δείκτης που υποδεικνύει αν το στοιχείο είναι φύλλο (LeafEntry). Ταυτόχρονα, το σύνολο visited διατηρεί το σύνολο των σημείων/κόμβων που έχουν προσπελασθεί ώστε να μην υπάρχει επανάληψη εισαγωγής τους στην ουρά. Η διαδικασία ξεκινάει από τη ρίζα του δέντρου εξετάζοντας όλες τις εγγραφές του. Κάθε εγγραφή μπορεί να είναι είτε ένας κόμβος φύλλου που περιέχει ένα σημείο **LeafEntry,** είτε ένας εσωτερικός κόμβος **Entry** που περιέχει ένα **ορθογώνιο** και έναν κόμβο παιδί **child**. Πρώτα ελέγχεται αν υπάρχει στο σύνολο visited με τη βοήθεια της συνάρτησης **id()** και αν δεν υπάρχει τότε το id της προστίθεται στο σύνολο. Έπειτα, αν η εγγραφή είναι φύλλο αυτό σημαίνει ότι περιέχει σημείο(entry.point) οπότε υπολογίζεται η ευκλίδεια απόστασή του από το qpoint και προστίθεται στην ουρά pq. Διαφορετικά, αν είναι εσωτερικός κόμβος, η εγγραφή περιέχει ένα ορθογώνιο(entry.rectangle) και έναν κόμβο παιδί (entry.child). Σε αυτή τη περίπτωση, υπολογίζεται η απόσταση μεταξύ του qpoint και του ορθογωνίου και στη συνέχεια προστίθεται στην ουρά προτεραιότητας. Έπειτα, μέχρι να αδειάσει η ουρά ή μέχρι να συγκεντρωθούν τα k πλησιέστερα σημεία, επαναλαμβάνονται τα εξής : Αρχικά, εξάγεται το distance, το current που αναφέρεται στο τρέχον σημείο και το is\_leaf που υποδηλώνει το είδος του σημείο(True) ή κόμβος(False). Αν είναι φύλλο, το σημείο προστίθεται στη λίστα αποτελεσμάτων **result** με την απόσταση του από το qpoint, τη θέση του και το record\_id του. Αν είναι εσωτερικός κόμβος, εξετάζεται κάθε εγγραφή του εάν αυτή δεν βρίσκεται στο σύνολο visited. Αν δεν βρίσκεται τότε, αν η εγγραφή είναι φύλλο υπολογίζεται η ευκλίδεια απόστασή του από το qpoint και προστίθεται στο pq. Διαφορετικά, υπολογίζεται η απόσταση του ορθογωνίου (MBR) από το qpoint με τη βοηθητική συνάρτηση της κλάσης rectangle **euclidean\_distance(qpoint)** και προστίθεται στο pq. Τέλος, μετά την επεξεργασία κάθε εγγραφής αυξάνεται ο μετρητής count.

Για τον έλεγχο της ορθής λειτουργίας του ΚΝΝ.py ενσωματώθηκε και η συνάρτηση **load\_rtree\_from\_xml(filename)** και χρησιμοποιήθηκε το indexfile1.xml για την ευκολότερη εξέταση των αποτελεσμάτων του.



Εκτέλεση παραδείγματος :

Για qpoint = [3,-8] και k = 2:

Distance: 1.0

RecordID: (1, 5)

**Point: [3.0, -7.0]**

---------------------

Distance: 3.1622776601683795

RecordID: (1, 4)

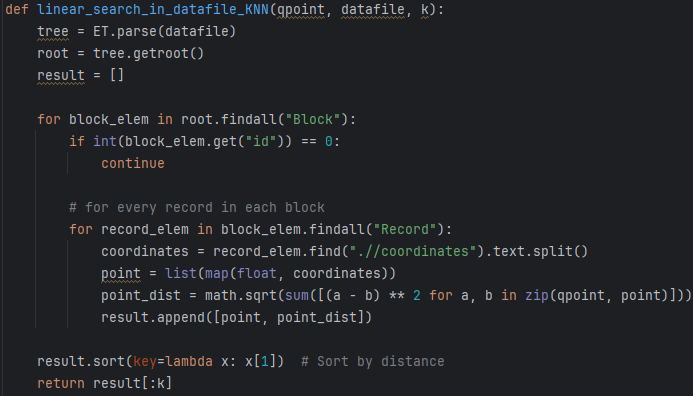
**Point: [2.0, -5.0]**

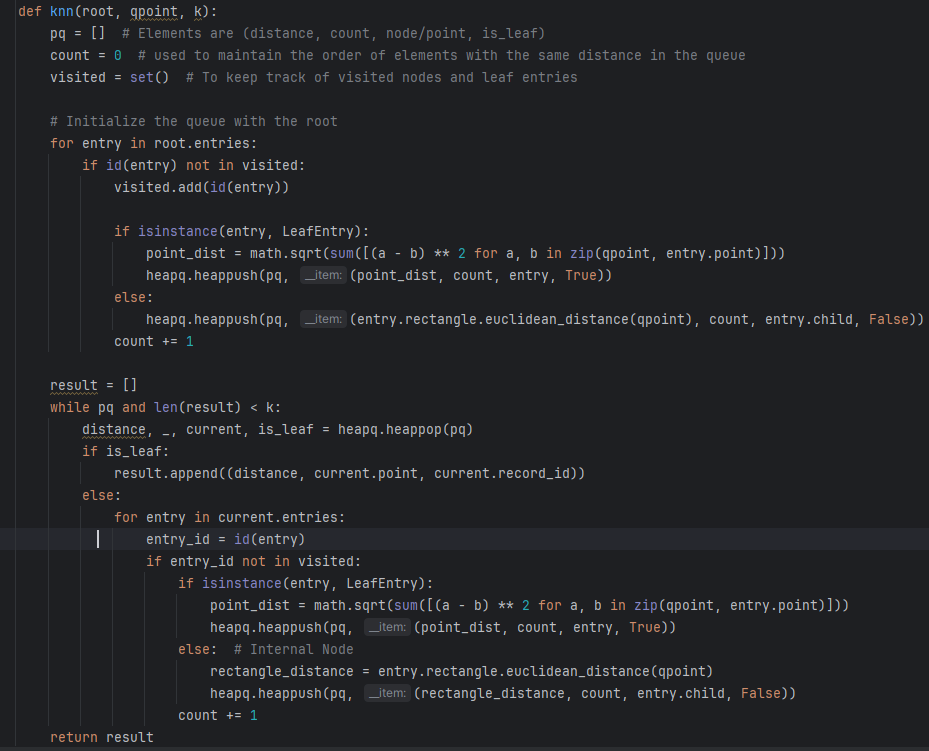
Στο αρχείο υλοποιήθηκε και η συνάρτηση **linear\_search\_in\_datafile\_KNN(qpoint,datafile,k)** στην οποία παίρνουμε στοιχεία από το XML αρχείο που θα επιλέξουμε και υπολογίζουμε την ευκλίδεια απόσταση κάθε σημείου από το qpoint προσθέτωντάς το στην λίστα αποτελεσμάτων. Επομένως, για κάθε record του block υπολογίζεται η ευκλίδεια απόστασή του από το qpoint και προστίθεται στο result. Τελικά, επιστρέφονται τα k πρώτα σημεία του result.

Για το indexfile.xml προκύπτουν τα παρακάτω αποτελέσματα για διαφορετικές τιμές του k:

|  |  |  |
| --- | --- | --- |
| **Κ** | **KNN algorithm time** | **Linear Search time** |
| 10 | 0.0000123121368443232 | 0.13809490203857422 |
| 100 | 0.0010123252868652344 | 0.15636754035949707 |
| 500 | 0.004174947738647461 | 0.15974618911743164 |
| 1000 | 0.019946575164794922 | 0.16841510772705078 |
| 3000 | 0.026024580001831055 | 0.17599272727966309 |
| 5000 | 0.09165358543395996 | 0.2206110954284668 |

Στιγμιότυπα κώδικα:

****

****

* Skyline

Η δομή που υλοποιήσαμε στο αρχείο **skyline.py** υποστηρίζει ερώτημα κορυφογραμμής και βασίζεται στον αλγόριθμο BBS (Branch and Bound):

Πιο συγκεκριμένα, η συνάρτηση **BBS(rtree,qpoint)** είναι υπεύθυνη για τον υπολογισμό της κορυφογραμμής με βάση ενός σημείου ενδιαφέροντος (qpoint) όπου qpoint σημείο τύπου (0,….,0) για τις n διαστάσεις. Για την υλοποίηση της συνάρτησης, χρησιμοποιούνται δύο δομές δεδομένων: η ουρά προτεραιότητας(min-heap) **pq** που περιέχει αντικείμενα **Qentry** και μια λίστα **skyline** για την αποθήκευση των σημείων κορυφογραμμής. Η διαδικασία ξεκινάει από τη ρίζα του δέντρου η οποία προστίθεται στην pq. Όσο η pq δεν είναι άδεια, εξάγεται από αυτή ο κόμβος με τη μικρότερη απόσταση **mindist** από το qpoint. Στη συνέχεια, αν ο κόμβος είναι φύλλο(**node.is\_leaf()**), τότε εξετάζεται κάθε σημείο του για να αποφασιστεί αν πρέπει να προστεθεί στο skyline(μόνο αν αυτό δεν κυριαρχείται από κανένα σημείο του skyline). Για τον έλεγχο κυριαρχίας χρησιμοποιείται η βοηθητική συνάρτηση **is\_dominated(entry.point,skyline)**. Αν δεν κυριαρχείται εισάγεται στο skyline και επείτα για κάθε σημείο του skyline εξετάζεται αν αυτό κυριαρχείται από το νεοεισαχθέν σημείο ώστε να διαγραφεί από το skyline. Για τον έλεγχο της κυριαρχίας σημείου σε άλλο σημείο χρησιμοποιείται η βοηθητική συνάρτηση **dominates(entry.point,s)**. Εάν ο κόμβος δεν είναι φύλλο, τότε για κάθε καταχώριση του κόμβου, υπολογίζεται η ελάχιστη απόσταση ανάμεσα στο qpoint και στο ορθογώνιο της καταχώρησης και ο κόμβος παιδί της καταχώρησης προστίθεται στην pq. Για τον υπολογισμό της απόστασης μεταξύ του qpoint και του ορθογωνίου χρησιμοποιείται η βοηθητική συνάρτηση **mindist(qpoint,rectangle)**.Όταν η pq αδειάσει, επιστρέφεται το αποτέλεσμα του ερωτήματος κορυφογραμμής.

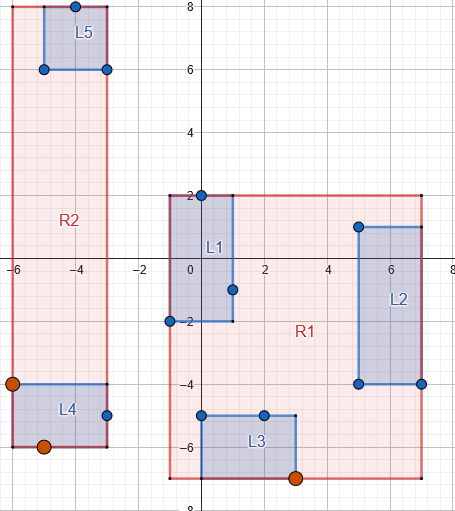
H κλάση **QEntry** εξυπηρετεί ως αντικείμενο που περιέχει δεδομένα για την ουρά προτεραιότητας pq. Περιλαμβάνει τις εξής μεταβήτές: μεταβλητή **mindist** που εκφράζει την ελάχιστη απόσταση ανάμεσα στο σημείο ερωτήματος και στον κόμβο ή την καταχώριση του δένδρου, τη μεταβλητή **node\_or\_entry** που μπορεί να είναι είτε ένας κόμβος του δένδρου είτε μια καταχώριση σε έναν κόμβο φύλλο. Η μέθοδος **\_\_lt\_\_(self, other)** υποστηρίζει τη σύγκριση δύο αντικειμένων QueueEntry με βάση την mindist. Χρησιμοποιείται από την heapq για να διατηρεί την ουρά προτεραιότητας σε ταξινομημένη σειρά.

Πιο αναλυτικά, η συνάρτηση **mindist(qpoint,rectangle)** υπολογίζει την ελάχιστη ευκλίδεια απόσταση μεταξύ ενός qpoint και ενός ορθογωνίου. Αρχικά, αρχικοποιείται μεταβλητή dist ίση με μηδέν και στη συνέχεια η επανάληψη διατρέχει κάθε διάσταση του σημείου ερωτήματος(q\_coord) καθώς και τα αντίστοιχα όρια του ορθογωνίου (bl για το κάτω αριστερό όριο και tr πάνω δεξί όριο). Αν q < bl, αυτό σημαίνει ότι το σημείο ερωτήματος βρίσκεται προς την αριστερή/κάτω πλευρά του ορθογωνίου σε αυτή τη διάσταση. Σε αυτή την περίπτωση, προστίθεται το τετράγωνο της απόστασης (bl – q\_coord) \*\* 2 στο dist. Διαφορετικά, αν q > tr, αυτό σημαίνει ότι το σημείο ερωτήματος βρίσκεται προς την δεξιά/πάνω πλευρά του ορθογωνίου σε αυτή τη διάσταση, οπότε προστίθεται το τετράγωνο της απόστασης (q\_coord - tr) \*\* 2 στο dist. Αν κανένα από τα δύο δεν ικανοποιείται τότε η απόσταση είναι ίση με το μηδέν.Τελικά, μετά το τέλος της επανάληψης , επιστρέφεται η τετραγωνική ρίζα του dist.

Η συνάρτηση **is\_dominated(point,spoints)** ελέγχει αν ένα σημείο κυριαρχείται από οποιοδήποτε άλλο σημείο που βρίσκεται για την ώρα στο skyline επιστρέφοντας True διαφορετικά False. Με τη σειρά της η συνάρτηση **dominates(a,b)** εξετάζει αν ένα σημείο κυριαρχεί πάνω σε κάποιο άλλο. Το σημείο b κυριαρχείται από το a, όταν ισχύουν τα εξής: κάθε διάσταση του σημείου a πρέπει να είναι μικρότερη ή ίση με την αντίστοιχη διάσταση του σημείου b και, τουλάχιστον σε μία διάσταση, το σημείο a πρέπει να είναι αυστηρά μικρότερη του σημείου b. Στην συνάρτηση δημιουργούνται δύο μεταβλητές less\_in\_one\_dimension και less\_or\_equal\_in\_all\_dimensions. Η πρώτη θα γίνει True εάν βρεθεί τουλάχιστον μια διάσταση όπου το a είναι αυστηρά μικρότερο του b. Η δεύτερη θα γίνει False εάν βρεθεί μια διάσταση όπου το a είναι μεγαλύτερο του b. Επαναληπτικά, διατρέχονται οι διαστάσεις των δύο σημείων ταυτόχρονα ελέγχοντας για κάθε διάσταση τα εξής. Εάν ai > bi: το σημείο a δεν κυριαρχεί το b για τη συγκεκριμένη διάσταση και η less\_or\_equal\_in\_all\_dimensions γίνεται False διακόπτοντας την επανάληψη.

Διαφορετικά, ai < bi: η less\_in\_one\_dimension γίνεται True, επειδή βρέθηκε μια διάσταση όπου το a είναι αυστηρά μικρότερο του b. Τέλος, επιστρέφεται True αν και οι δύο συνθήκες ισχύουν.

Αναφορικά με τις συναρτήσεις **get\_record\_from\_datafile(points,filename)** και τη **read\_block\_from\_datafile(block\_id, filename)** αυτές χρησιμοποιούνται για την αποδοτική ανάκτηση συγκεκριμένων εγγραφών από ένα αρχείο δεδομένων, χρησιμοποιώντας πληροφορίες σχετικά με το πού βρίσκονται οι εγγραφές αυτές. Πιο συγκεκριμένα η πρώτη έχει ως σκοπό να ανακτήσει συγκεκριμένες εγγραφές από ένα αρχείο XML, χρησιμοποιώντας μια λίστα από αντικείμενα LeafEntry που περιέχουν πληροφορίες για τα έγγραφα που επιθυμούμε να ανακτήσουμε. Για να επιτευχθεί αυτό γίνεται αρχικά ομαδοποίηση σημείων σε ένα λεξικό με κλειδί το block\_id και τιμή μια λίστα των αντίστοιχων σημείων LeafEntry. Στη συνέχεια, γίνεται ανάκτηση εγγραφών για κάθε block καλώντας για κάθε block τη read\_block\_from\_datafile για να ανακτυθούν όλες οι εγγραφές του συγκεκριμένου block από το XML. Τέλος, επιστρέφονται οι εγγραφές που ανακτήθηκαν. Η δεύτερη συνάρτηση, διαβάζει όλες τις εγγραφές από ένα συγκεκριμένο block ενός XML, προσδιοριζόμενο από το block\_id. Αν το block βρεθεί, διαβάζει όλα τα στοιχεία Record εντός αυτού του block. Κάθε εγγραφή περιλαμβάνει ένα record\_id, ένα name, και coordinates (συντεταγμένες).

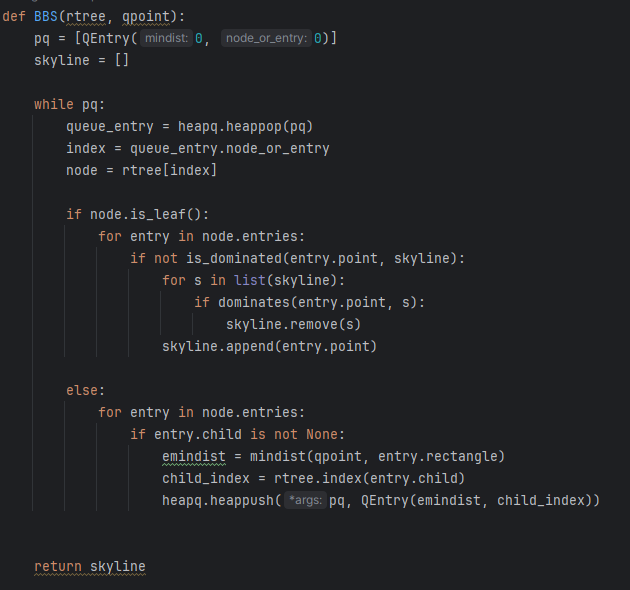
Αυτές οι πληροφορίες αποθηκεύονται σε μια λίστα που αντιπροσωπεύει τις εγγραφές του block και επιστρέφεται.

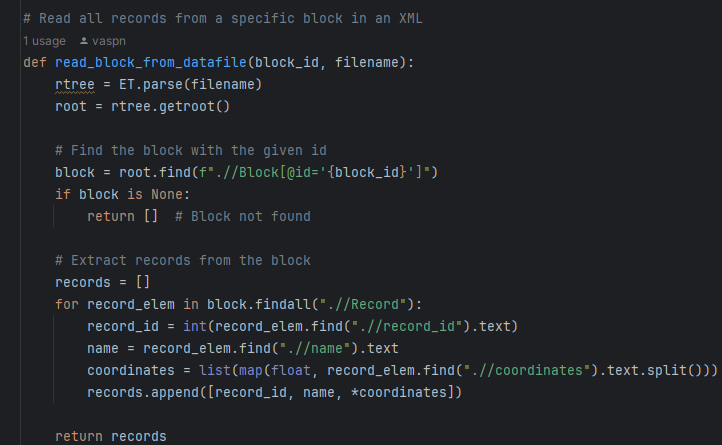
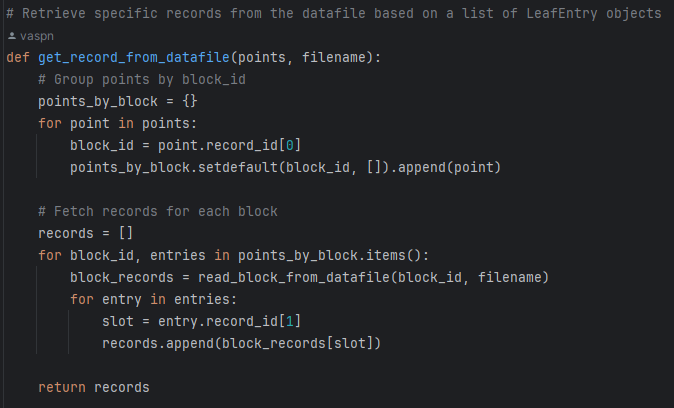
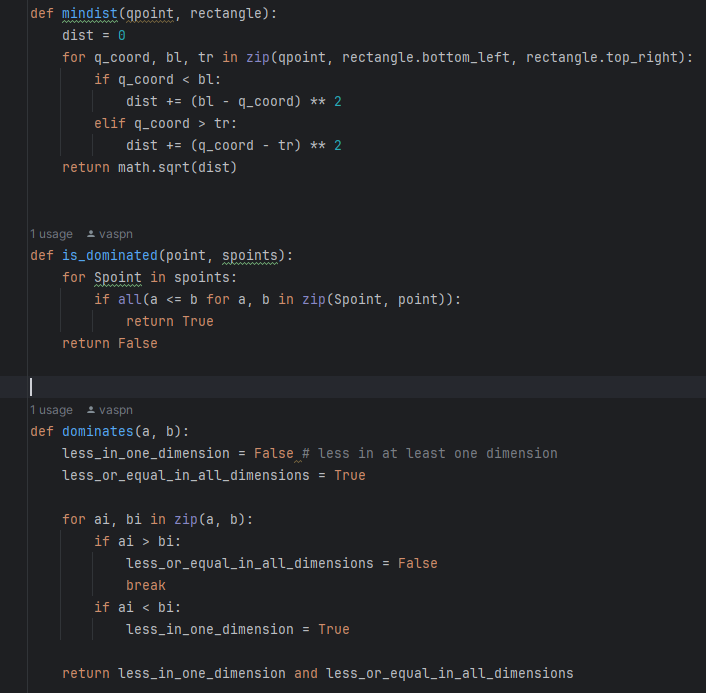
Εκτέλεση παραδείγματος :

Για qpoint = [0,0] :

Result of skyline algorithm: [[3.0, -7.0], [-6.0, -4.0],[-5.0, 6.0]]

Στιγμιότυπα κώδικα:





### **Παραδείγματα και Παρατηρήσεις**