Εργασία 2023 - 2024

Τεχνολογια βασεων δεδομενων

Υλοποίηση Χωρικών Μεθόδων Προσπέλασης

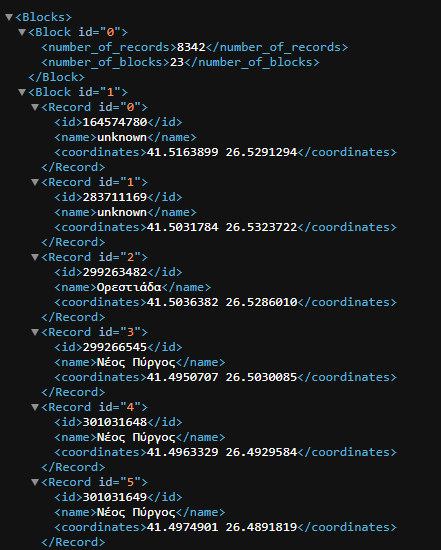
Ναλμπάντη παναγιωτα, 4050

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.

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)

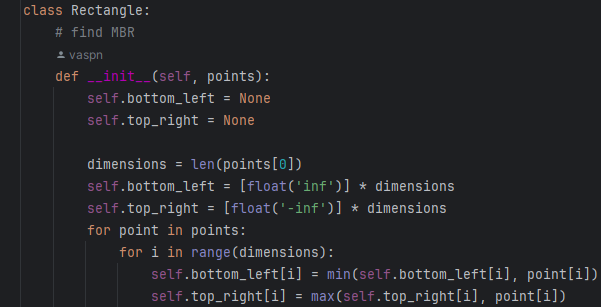
1. Δημιουργία του R\*-tree

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 :

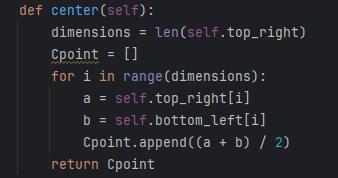
* **Rectangle**: Αναπαριστά ένα πολυδιάστατο ορθογώνιο που ορίζεται από δύο σημεία. Έχει δύο πεδία :
* **bottom\_left** που αντιπροσωπεύει το κάτω αριστερά σημείο
* **top\_right** που αντιπροσωπεύει το πάνω δεξία σημείο

Περιλαμβάνει τις εξής μεθόδους:

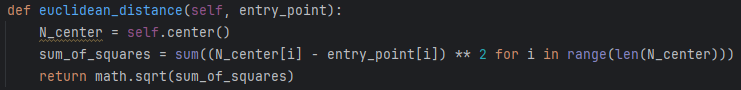
* **\_\_init\_\_(self, points)**: παίρνει ως όρισμα μία λίστα από σημεία (πχ [[7,5, 5.2], [0,3, 1.2], [2,5, 1.7]]) και υπολογίζει το minimum bounding rectangle(ΜΒR), βρίσκοντας και αρχικοποιώντας το bottom\_left και το top\_right υπολογίζοντας τη μικρότερη και μεγαλύτερη τιμή κάθε διάστασης αντίστοιχα:



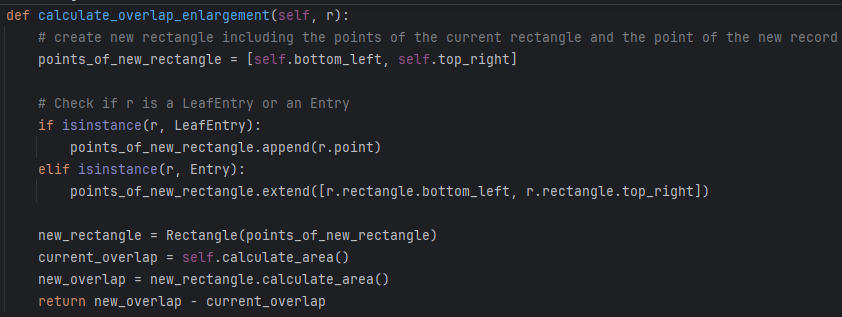
* **center(self):** υπολογίζει και επιστρέφει ως λίστα τις συντεταγμένες του κεντρικού σημείου ενός ορθογωνίου:



* **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:

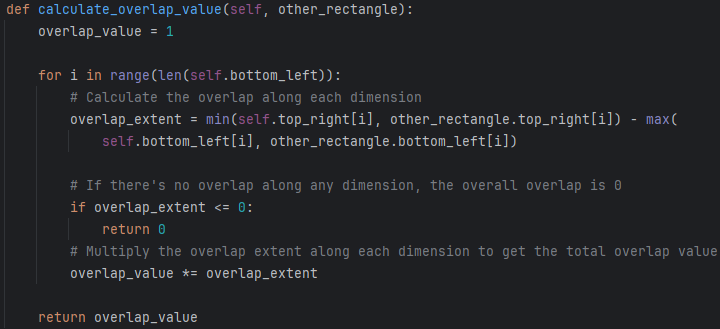


* **calculate\_overlap\_enlargement(self, r)**: υπολογίζει την αλλαγή (αύξηση) στο εμβαδόν της επικάλυψης ενός ορθογωνίου όταν προστίθεται ένα νέο σημείο ή ένα άλλο ορθογώνιο. Η συνάρτηση υπολογίζει το εμβαδόν του τρέχοντος ορθογωνίου όπως και του νέου με την calculate\_area(self). Τέλος, επιστρέφεται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

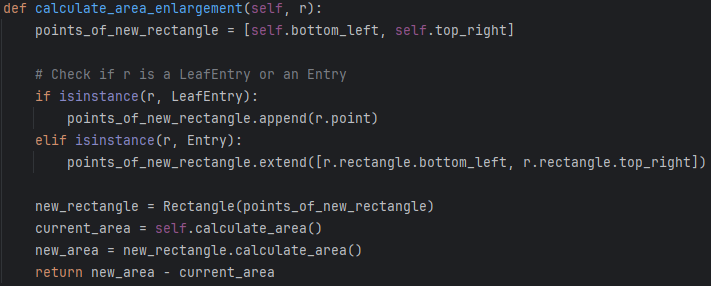


* **calculate\_overlap\_value(self, other\_rectangle)**: υπολογίζει το εμβαδόν επικάλυψης μεταξύ δύο ορθογωνίων. Συγκεκριμένα, υπολογίζει την επικάλυψη μεταξύ του ορθογωνίου που καλεί τη συνάρτηση (self) και ενός άλλου ορθογωνίου (other\_rectangle) κατά μήκος κάθε διάστασης. Για κάθε διάσταση i (χρησιμοποιώντας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

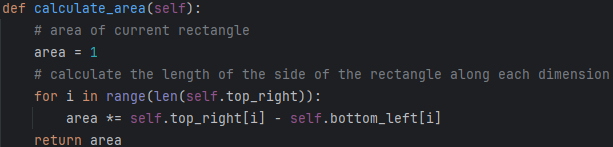
Υπολογίζεται το μέγιστο από τα κάτω άκρα των δύο ορθογωνίων και το ελάχιστο από τα άνω άκρα τους. Η διαφορά μεταξύ του ελάχιστου άνω άκρου και του μέγιστου κάτω άκρου δίνει το μήκος της επικάλυψης για τη συγκεκριμένη διάσταση. Αν το μήκος της επικάλυψης (overlap\_extent) είναι μικρότερο ή ίσο με το μηδέν 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.



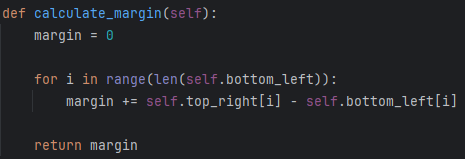
* **calculate\_area\_enlargement(self, r):** υπολογίζει την αύξηση της επιφάνειας ενός ορθογωνίο όταν επεκτείνεται για να συμπεριλάβει ένα νέο στοιχείο, που μπορεί να είναι είτε ένα σημείο είτε ένα άλλο ορθογώνιο. Ένα νέο ορθογώνιο δημιουργείται χρησιμοποιώντας τα σημεία της λίστας points\_of\_new\_rectangle. 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.

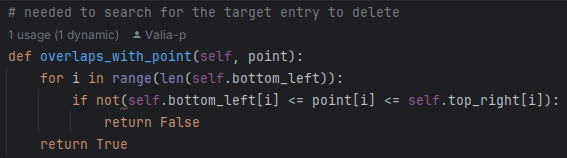


* **calculate\_margin(self)**: υπολογίζει και επιστρέφει την περίμετρο του ορθογωνίου. Ο υπολογισμός της περιμέτρου γίνεται ως το άθροισμα των διαφορών top\_right[i]-bottom\_left[i] για κάθε διάσταση i.

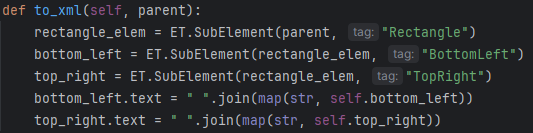


* **overlaps\_with\_point(self, point)**: ελέγχει αν ένα σημείο βρίσκεται μέσα σε ένα ορθογώνιο. Χρησιμοποιεί ένα βρόχο για να περάσει από όλες τις διαστάσεις του ορθογωνίου και για κάθε διάσταση, ελέγχει αν η συντεταγμένη του σημείου point σε αυτή τη διάσταση βρίσκεται εντός των ορίων του ορθογωνίου. 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.



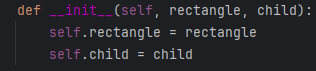
* **to\_xml(self, parent)**: Κωδικοποιεί ένα αντικείμενο της κλάσης Rectangle σε μορφή xml. Πιο συγκεκριμένα, δέχεται το parent element που θα έχει το αντικέιμενο Rectangle στο indexfile.xml και δημιουργεί ένα subelement με tag “Rectangle” και μέσα σε αυτό το tag βάζει δύο subelement: το “BottomLeft”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



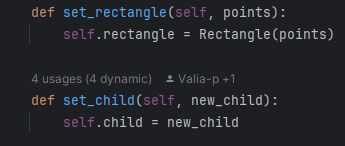
* **Entry**: Αναπαριστά μία εσωτερική εγγραφή, δηλαδή εγγραφή εσωτερικού κόμβου. Αποτελείται από δύο πεδία :
* **rectangle**: αναπαριστά το minimum bounding rectangle του κόμβου παιδιού του συγκεκριμένου entry.
* **child**: αναπαριστά τον κόμβο παιδί του συγκεκριμένου entry.

Περιλαμβάνει τις εξής μεθόδους:

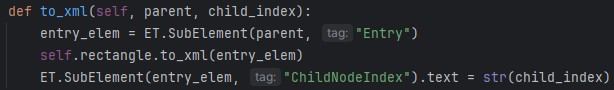
* **\_\_init\_\_(self, rectangle, child)**: η οποία δέχεται ως ορίσματα ένα ορθογώνιο και ένα κόμβο ώστε να αρχικοποιήσει τα πεδία rectangle και child.



* **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.



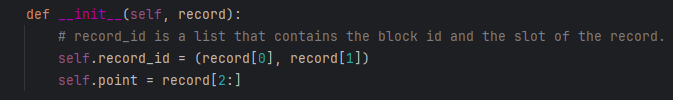
* **to\_xml(self, parent, child\_index)**: Κωδικοποιεί ένα αντικείμενο της κλάσης Entry σε μορφή xml. Συγκεκριμένα, δέχεται ένα γονικό στοιχείο xml (parent) που θα περιέχει το αντικείμενο Entry στο αρχείο indexfile.xml. Στη συνέχεια, δημιουργεί ένα υποστοιχείο με το tag "Entry". Μέσα σε αυτό το tag, 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.



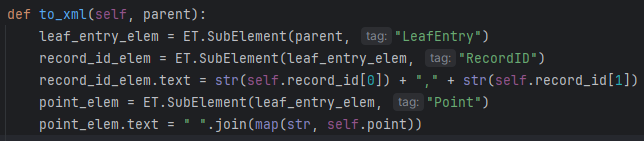
* **LeafEntry**: Αναπαριστά μία εγγραφή φύλλου δηλαδή μια εγγραφή ενός κόμβου φύλλου. Έχει δύο πεδία:
* **record**: Αποτελεί μία λίστα που περιέχει το id του block στο οποίο βρίσκεται η συγκεκριμένη εγγραφή στο datafile και το slot στο οποίο βρίσκεται μέσα στο block.
* **point**: Αποτελεί μία λίστα που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

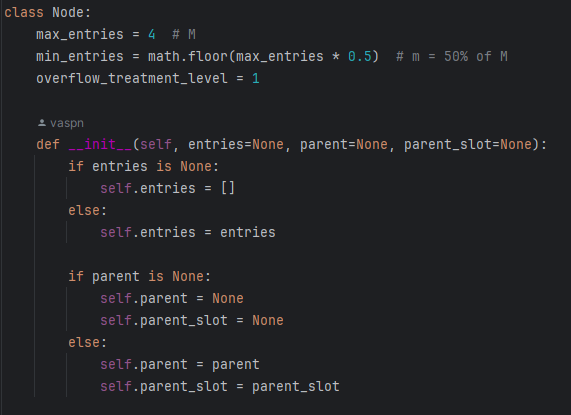
* **Node** : Αναπαριστά έναν κόμβο του R-tree και αποτελείται από τρία πεδία:
* **entries** : αποτελεί μία λίστα που περιέχει τις εγγραφές του κάθε κόμβου οι οποίες είναι όλες είτε αντικείμενα της κλάσης Entry, αν ο κόμβος είναι εσωτερικός, είτε αντικείμενα της κλάσης LeafEntry, αν ο κόμβος είναι φύλλο.
* **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 :

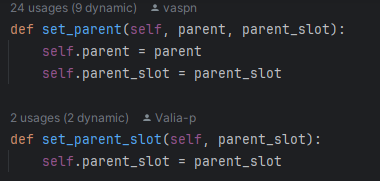
* + - **max\_entries** : εκφράζει το μέγιστο επιτρεπτό πλήθος εγγραφών, M, που μπορεί να περιέχει ένας κόμβος (άμα δεν οριστεί μέσα στον κώδικα , η default τιμή του είναι 4)
    - **min\_entries** : εκφράζει το ελάχιστο επιτρεπτό πλήθος εγγραφών, m, που μπορεί να περιέχει ένας κόμβος. Ορίζεται ως το 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:

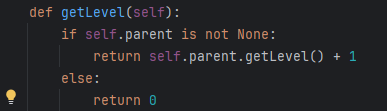
* **\_\_init\_\_(self, entries = None, parent = None, parent\_slot = None)**: Αρχικοποιεί τα πεδία που δέχεται ως ορίσματα. Σε περίπτωση που δεν δεχτεί κανένα όρισμα ή δεχτεί μόνο entries, αρχικοποιεί τα πεδία τα οποία δεν είχαν ανάλογο όρισμα με None και αναμένεται να γίνουν set ύστερα με τις ανάλογες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



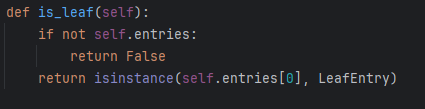
* **set\_parent(self, parent, parent\_slot)**: αποτελεί έναν setter για τα πεδία parent και parent\_slot.
* **set\_parent\_slot(self, parent\_slot)**: αποτελεί έναν setter μόνο για το πεδίο parent\_slot και χρησιμοποιείται όταν δεν αλλάζει ο γονικός κόμβος αλλά χρειάζεται ενημέρωση του parent\_slot λόγω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

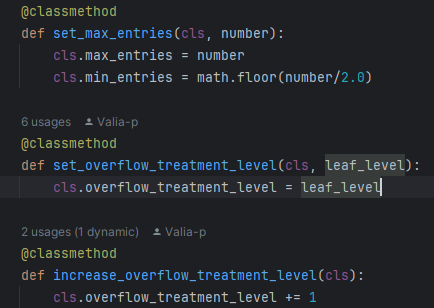


* **is\_leaf(self)**: Ελέγχει αν ο κόμβος είναι εσωτερικός ή φύλλο,εξετάζοντας την εγγραφή που βρίσκεται σε αυτόν. Επιστρέφει True αν είναι LeafEntry διαφορετικά False αν είναι Entry.



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.

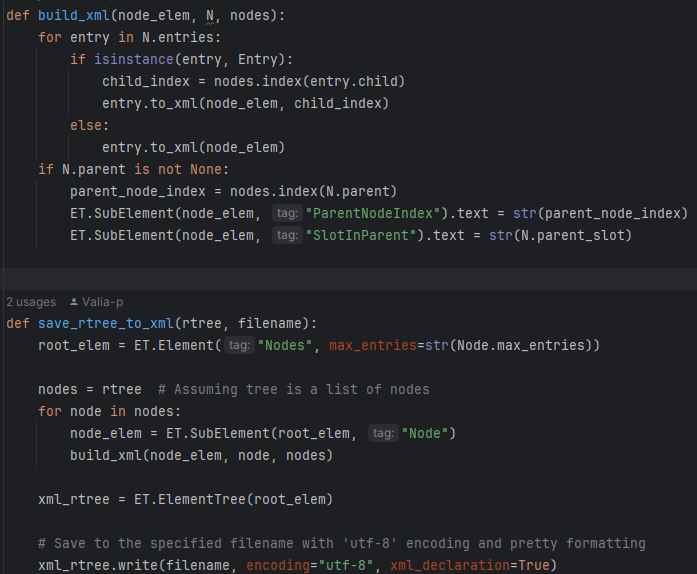


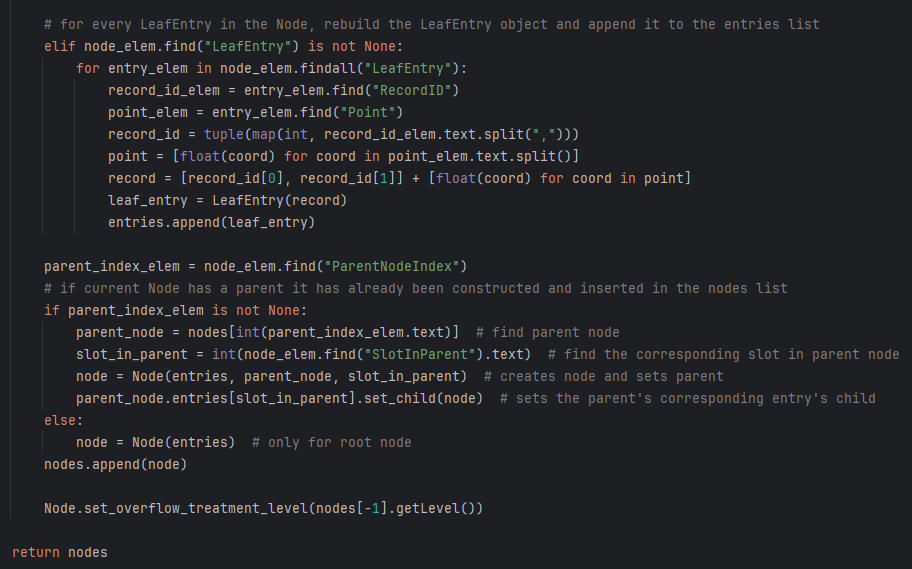
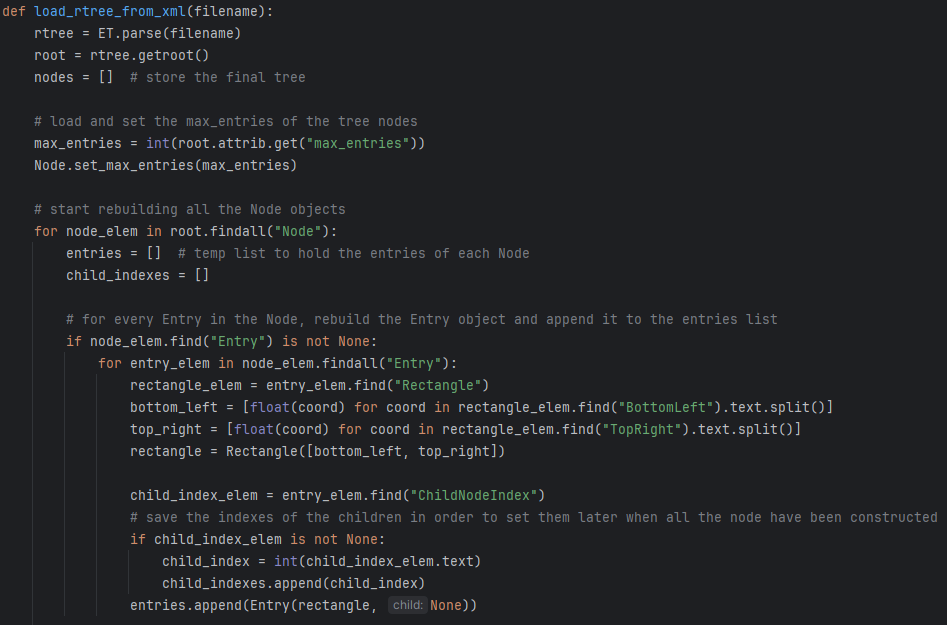
H δομή χτίζεται και αποθηκεύεται ως μία λίστα rtree που περιέχει αντικείμενα της κλάσης Node τα οποία διατηρούν μια συγκεκριμένη ιεραρχία με βάση το επίπεδο στο οποίο βρίσκεται ο κάθε κόμβος. Πιο συγκεκριμένα, στην θέση 0 της λίστας βρίσκεται η ρίζα, στις επόμενες θέσεις βρίσκονται οι κόμβοι-παιδιά της 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:

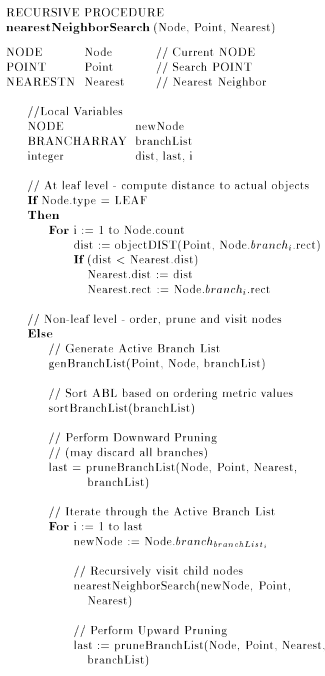


1. Εισαγωγή Εγγραφής

### **Ερωτήματα πάνω στο R\* - tree**

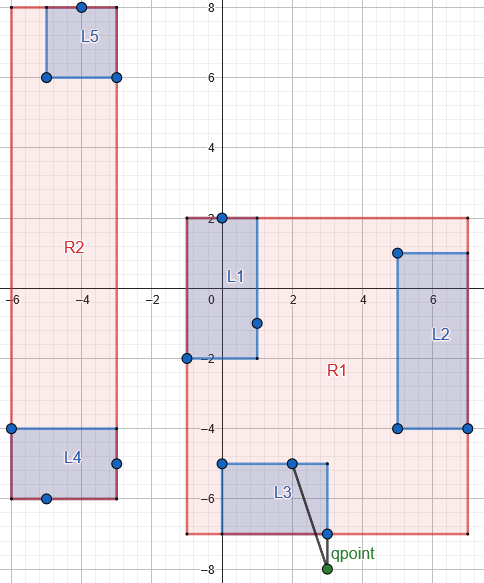
* Range Query
* KNN

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)

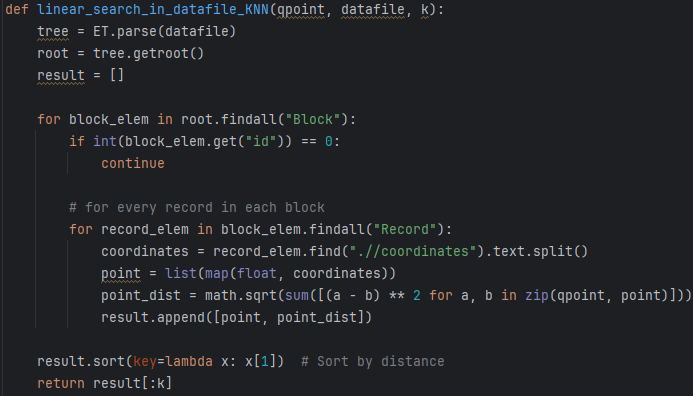
**Point: [2.0, -5.0]**

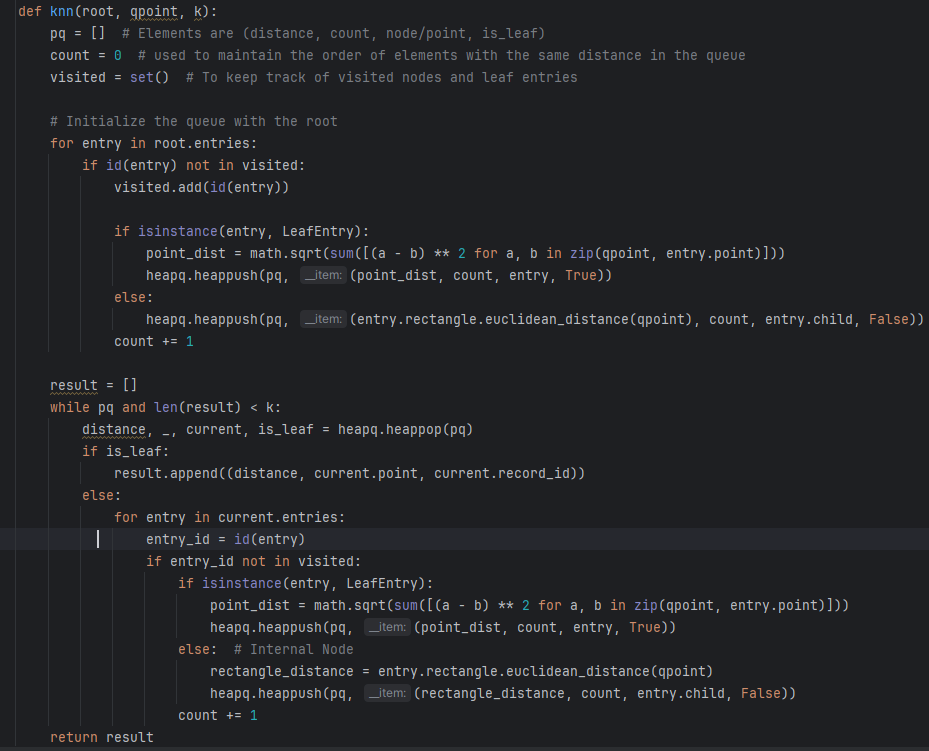
Στο αρχείο υλοποιήθηκε και η συνάρτηση **linear\_search\_in\_datafile\_KNN(qpoint,datafile,k)** στην οποία παίρνουμε στοιχεία από το XML αρχείο που θα επιλέξουμε και υπολογίζουμε την ευκλίδεια απόσταση κάθε σημείου από το qpoint προσθέτωντάς το στην λίστα αποτελεσμάτων. Επομένως, για κάθε record του 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.

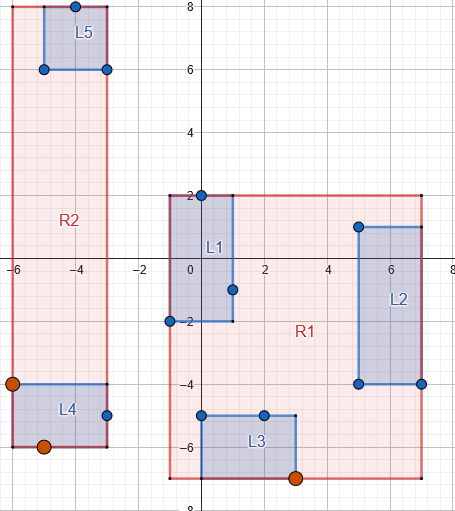
H κλάση **QEntry** εξυπηρετεί ως αντικείμενο που περιέχει δεδομένα για την ουρά προτεραιότητας pq. Περιλαμβάνει τις εξής μεταβήτές: μεταβλητή **mindist** που εκφράζει την ελάχιστη απόσταση ανάμεσα στο σημείο ερωτήματος και στον κόμβο ή την καταχώριση του δένδρου, τη μεταβλητή **node\_or\_entry** που 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).

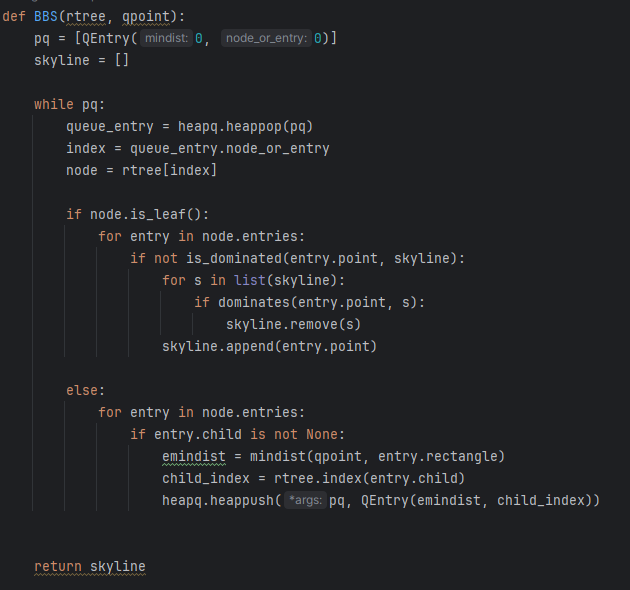
Αυτές οι πληροφορίες αποθηκεύονται σε μια λίστα που αντιπροσωπεύει τις εγγραφές του block και επιστρέφεται.

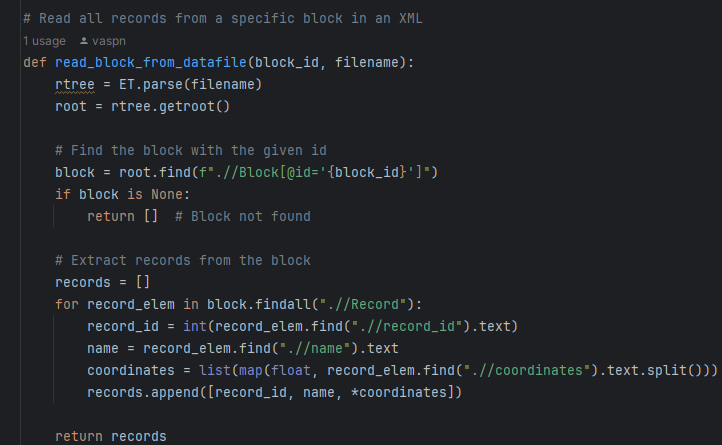
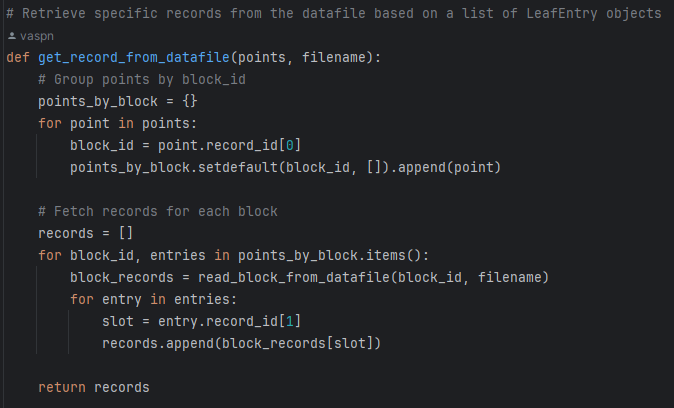
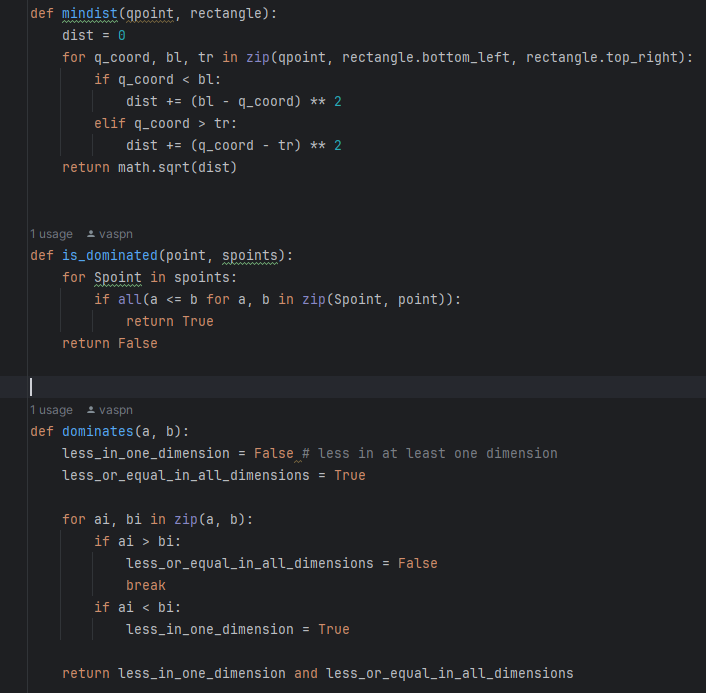
Εκτέλεση παραδείγματος :

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**