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.

### **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).

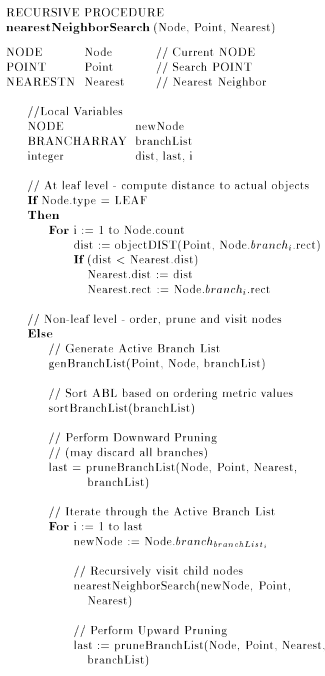
Συνεχίζουμε δημιουργώντας ένα αρχείο datafile.xml, το οποίο θα περιέχει την λίστα των blocks με τα records σε μορφή xml.

### **Υλοποίηση R\*- tree**

### **Ερωτήματα πάνω στο R\* - tree**

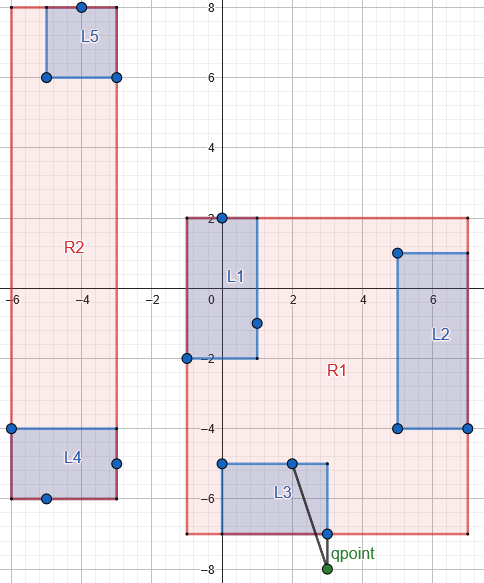
* Range Query
* KNN

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]**

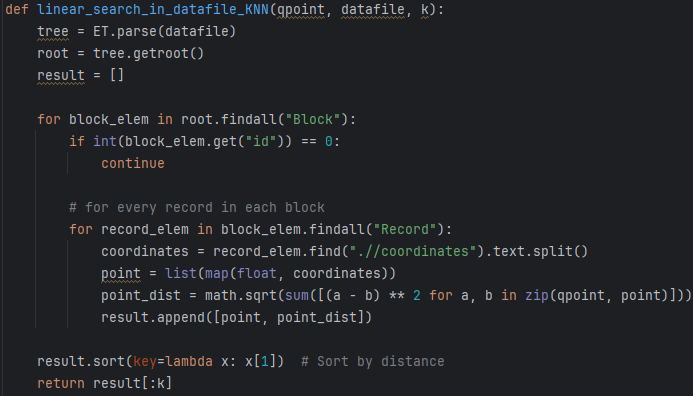
Στο αρχείο υλοποιήθηκε και η συνάρτηση **linear\_search\_in\_datafile\_KNN(qpoint,datafile,k)** στην οποία παίρνουμε στοιχεία από το XML αρχείο που θα επιλέξουμε και υπολογίζουμε την ευκλίδεια απόσταση κάθε σημείου από το qpoint προσθέτωντάς το στην λίστα αποτελεσμάτων. Επομένως, για κάθε record του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

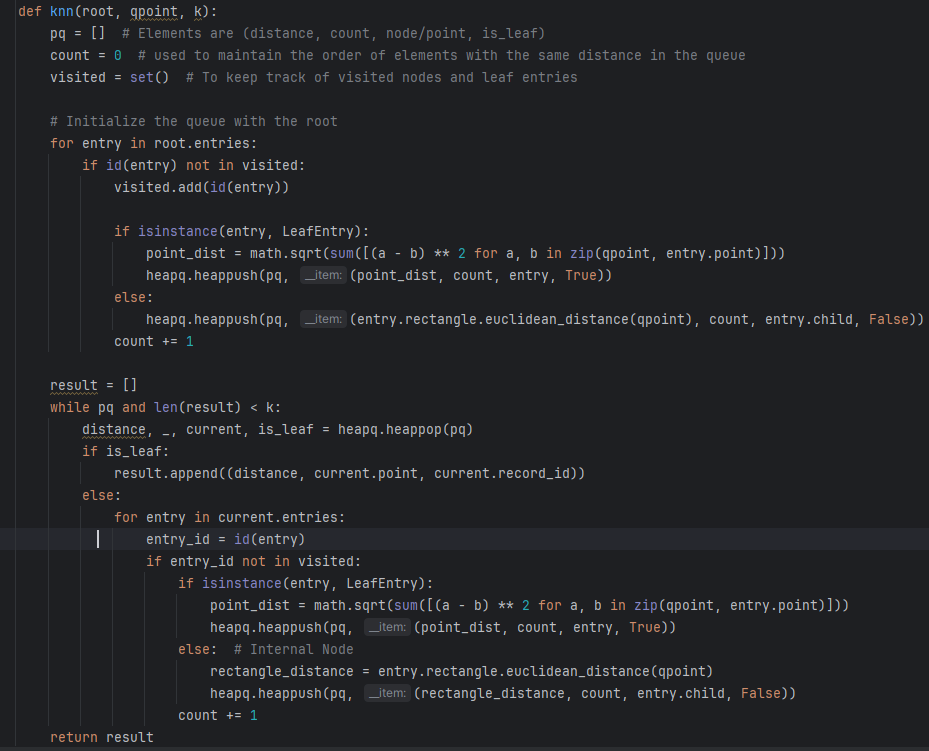
Για το indexfile.xml προκύπτουν τα παρακάτω αποτελέσματα για διαφορετικές τιμές του k:

Σύγκρ

Στιγμιότυπα κώδικα:

|  |  |  |
| --- | --- | --- |
| **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 |

****

****

* 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.

### **Παραδείγματα και Παρατηρήσεις**