Τεχνολογία Βάσεων Δεδομένων

Ονοματεπώνυμο: Κωνσταντίνος – Ηλίας Χονδρορρίζος
A.E.M. : 3812

Ονοματεπώνυμο: Θεόδωρος Ιωαννίδης
A.E.M. : 3626

Τεχνική αναφορά:

Στην συγκεκριμένη εργασία, υλοποιήσαμε την δομή του R Star Tree, που έχει σκοπό την δημιουργία ενός indexing καταλόγου για την αποθήκευση γεωγραφικών δεδομένων, στην γλώσσα προγραμματισμού Python.
Η δομή είναι πλήρως λειτουργική για in-memory χρήση, καθώς δεν καταφέραμε να την κάνουμε να δουλέψει με την χρήση εξωτερικών αρχείων.

- Γενικές πληροφορίες για την λειτουργία και την λογική της υλοποίησης

Η δομή αρχικοποιείται με την δημιουργία ενός αντικειμένου RTree() και περιλαμβάνει μεταβλητές όπως η ρίζα και πληροφορίες για το ύψος του δένδρου. Για την εισαγωγή ενός αντικειμένου στην δομή, αρκεί να καλέσουμε την συνάρτηση Insert_data(arg) με όρισμα 'arg' την εγγραφή που επιθυμούμε και η κλάση του Rtree θα αναλάβει όλη την διαδικασία.

Εφόσον επιθυμούμε να τρέξουμε το πρόγραμμα από κάποιο τερματικό, θα χρειαστεί να δώσουμε σαν όρισμα, εκτός από το αρχείο, και έναν ακέραιο αριθμό, ο οποίος θα αντιπροσωπεύει την διάσταση των δεδομένων που θέλουμε να εισάγουμε στην δομή.

Π.χ. :

```
\Desktop\Spatial Data> python main.py 3
(1, 1, 3), level: 1
```

Αν δουλεύουμε με τρισδιάστατα δεδομένα. Διαφορετικά, για τον σκοπό της εργασίας που έχουμε 2 διαστάσεις (Lat / Lon coordinates), τρέχουμε:

```
\Desktop\Spatial Data> python main.py 2
(1, 1), level: 1
```

Όπως αναφέραμε και νωρίτερα, η RTree κλάση περιέχει μια μεταβλητή η οποία δείχνει στην ριζά του δένδρου. Η ριζα βρίσκεται στο πρώτο επίπεδο του δένδρου και όσο αυξάνεται το μέγεθος του αυτού, αυξάνονται και τα levels προς τα κάτω .Η ρίζα είναι ένα αντικείμενο της κλάσης Node η οποία αντιπροσωπεύει έναν κόμβο του δένδρου.

```
self.root = None
self.overflow_flags = {}
self.total_levels = 1
```

*overflow_flags είναι ένα dictionary που μας δείχνει αν έχει υπάρξει διάσπαση σε κάποιο level του δένδρου ή όχι. Είναι της μορφής:
{1 :False, 2: True , 3:False …}

- Κλάση Node:

Η συγκεκριμένη κλάση αντιπροσωπεύει ένα κόμβο του δένδρου και έχει τις παρακάτω μεταβλητές:

```
self.MAX_DEGREE = 100
self.MIN_FILL_FACTOR = int(.50 * self.MAX_DEGREE)
self.entries = self.update_belonging_node(entries)
self.parent_entry = parent_entry
```

Όπου entries είναι οι εγγραφές που υπάρχουν μέσα στον κόμβο, MAX_DEGREE και MIN_FILL_FACTOR είναι οι μέγιστες και οι ελάχιστές εγγραφές που μπορεί να περιλαμβάνει αυτος και parent_entry είναι μια εγγραφή από κάποιον κόμβο στο προηγούμενο level του δένδρου, που δείχνει στον συγκεκριμένο (το πεδίο αυτό είναι None την ριζα καθώς βρίσκεται στο πρώτο επίπεδο και δεν έχει γονέα). Όσον αφορά τις εγγραφές:

- Κλάσεις των Entries:

Τα entries που βρίσκονται καταχωρημένα σε κάθε node του δένδρου και μπορεί να είναι αντικείμενα δυο πιθανών κλάσεων:

- Record ή
- Middle_entry

Τα αντικείμενα τύπου Record επιτρέπεται να υπάρχουν μόνο στους κόμβους του τελευταίου level του δένδρου καθώς αναπαριστούν μια γεωγραφική εγγραφή με τις πληροφορίες που αναφέρονται παρακάτω:

```
    self.record_slot = slot
    self.block_id = block
    self.cordinates = cordinates
    self.MBR = MBR(points=self.get_record_mbr())
    self.belonging_node = None
```

Όπου record_slot είναι το slot / id της συγκεκριμένης εγγραφής,
block_id το αναγνωριστικό του block εντός του οποίου βρίσκεται αποθηκευμένη η εγγραφή στο datafile , coordinates οι lat/lon συντεταγμένες ,
belonging node ένας δείκτης, ο οποίος δείχνει στον κόμβο στον οποίο είναι αποθηκευμένη η εγγραφή και τέλος,
MBR οπού είναι ένα αντικείμενο της κλάσης MBR δηλαδή του Minimum Bounding Rectangle του σημείου. (Από την στιγμή που οι συντεταγμένες είναι απλά ένα σημείο στον χώρο το rectangle / box δεν έχει εμβαδόν)

Αντιθέτως, τα αντικείμενα τύπου Middle_entry είναι αναγκαίο να βρίσκονται αποθηκευμένα σε nodes του δένδρου, τα οποία δεν είναι φύλλα. Τα μεσαία entries,

λοιπόν, έχουν τα εξής attributes:

```
self.child_pointer = None
self.MBR = None
self.belonging_node = belonging_node
```

Δηλαδή, αντί για τις πληροφορίες των εγγράφων που έχουν τα Record αντικείμενα, αυτές έχουν ένα pointer που δείχνει σε κάποιο κόμβο, με σκοπό να δημιουργηθεί μονοπάτι από την ρίζα μέχρι τα φύλλα μέσω αυτών.
Η belonging_node μεταβλητή έχει αντίστοιχο σκοπό με αυτήν στα Records αντικείμενα, όμως η MBR μεταβλητή κρατάει το minimum bounding rectangle των σημείων / εγγράφων που βρισκονται στο υποδένδρο κάτω από αυτή την εγγραφή.

- Κλάση MBR:

Με την σειρά της λοιπόν, η κλάση MBR, όπως αναφέραμε και παραπάνω, αντιπροσωπεύει το ελάχιστο περιγεγραμμένο ορθογώνιο / υπερ-ορθογώνιο κάποιων σημείων και ως attributes έχει:

```
self.points = points
self.bottom_left, self.upper_right = MBR.min_bounding_box(self)
```

Points είναι τα σημεία τα οποία θέλουμε να περικλείσουμε μέσα στο ορθογώνιο και bottom_left, upper_right οι δυο συντεταγμένες με τις οποίες μπορούμε να περιγράψουμε το ορθογώνιο αυτό.

Κάθε μια από τις προαναφερθείσες κλάσεις έχει συναρτήσεις που βοηθούν στο σωστό χτίσιμο του δένδρου, με τον τρόπο που περιγράφει το original paper. Περισσότερες πληροφορίες υπάρχουν στα σχόλια του πηγαίου κώδικα που επισυνάπτονται.

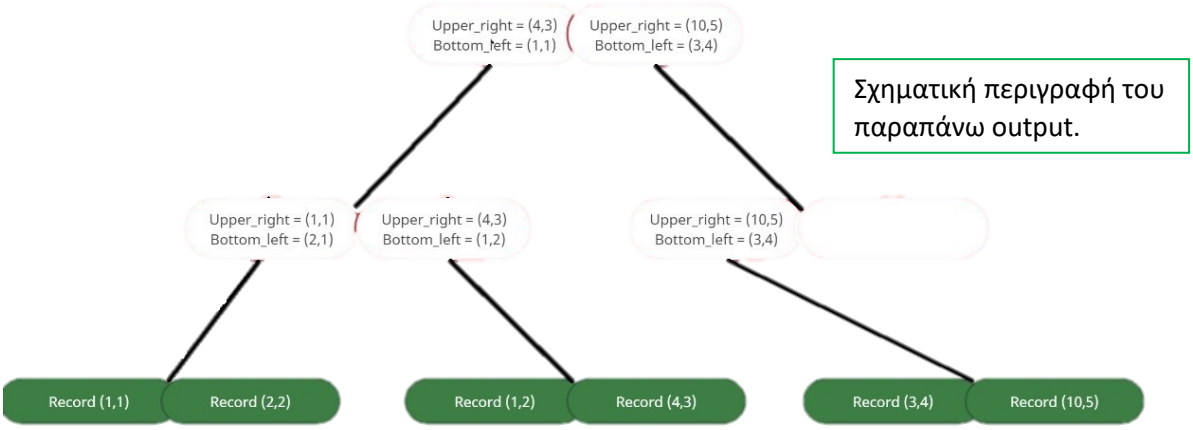
Επόμενος καταλήγουμε ότι:

Το δένδρο αποτελείται από Nodes τα οποία έχουν αποθηκευμένα entries (Record entry στα φύλλα, Middle entry στους υπόλοιπους) μέσω αυτών δημιουργείται ένα μονοπάτι μεταξύ της ρίζας και των εγγράφων και καθώς διασχίζουμε top-down το δένδρο, το MBR των entries γίνεται όλο και πιο μικρό έως ότου φτάσουμε στα Records.

Ένα ενδεικτικό παράδειγμα μιας depth first διάσχισης του δένδρου μέσο της συνάρτηση print_tree() που υπάρχει στο αρχείο R_star_tree.py μετα την εισαγωγή ορισμένων σημείων είναι:

```
Middle entry: ((1, 1), (4, 3)), level: 1
Middle entry: ((1, 1), (2, 1)), level: 2
(1, 1), level: 3
(2, 1), level: 3
Middle entry: ((1, 2), (4, 3)), level: 2
(1, 2), level: 3
(4, 3), level: 3
Middle entry: ((3, 4), (10, 5)), level: 1
Middle entry: ((3, 4), (10, 5)), level: 2
(3, 4), level: 3
(10, 5), level: 3
```

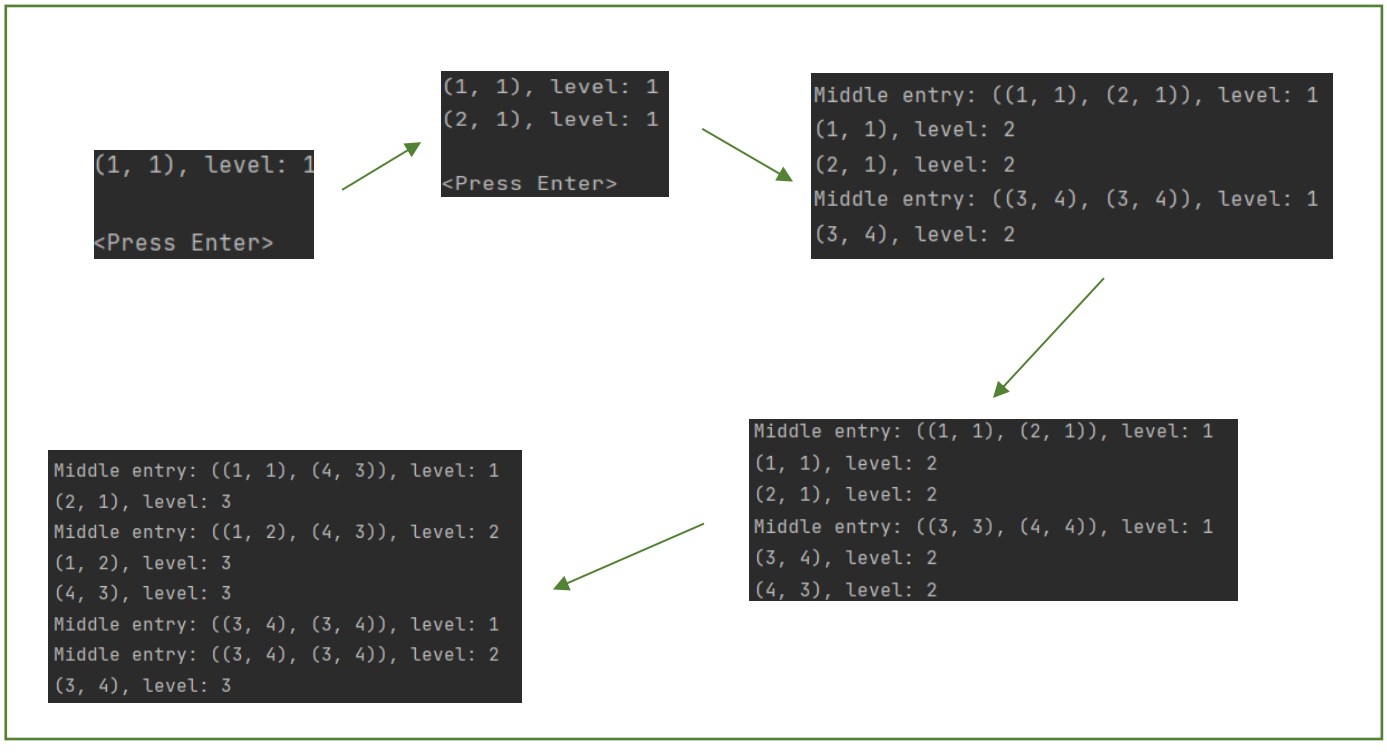
MAX_DEGREE = 2
MIN_FILL_FACTOR = 1



Σχηματική περιγραφή του παραπάνω output.

- <u>Εισαγωγή εγγραφής</u>

Για την εισαγωγή ενός entry στο δένδρο, εκτελείται μια αλυσίδα από συναρτήσεις που περιγράφονται στο paper και στα σχόλια του κώδικα, προκειμένου να υπάρξει κατάλληλος χειρισμός σε πιθανά splits και re-insertion.
Στο αρχείο ‘main.py’ υπάρχει ένα απλό παράδειγμα που εκτυπώνει το δένδρο στην οθόνη μετά από κάθε εισαγωγή.
Ενδεικτικά:

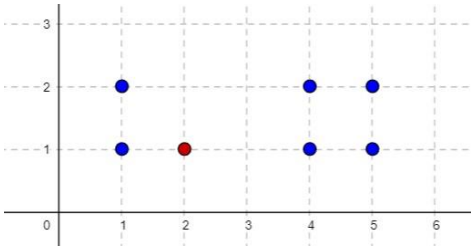


- Διαγραφή εγγραφής

Για να διαγράψουμε κάποιο αντικείμενο που έχει εκχωρηθεί προηγουμένως στην δομή, δεν υπάρχει διαφορετικός τρόπος στο R star tree. Έτσι λοιπόν, η υλοποίηση που κατασκευάσαμε είναι αυτή που χρησιμοποιείται και στο απλό R tree, η οποία περιγράφεται στο σετ διαφανειών 'Indexing with R-trees' του e-learning. Για την διαγραφή της εγγραφής δεν αρκεί να δώσουμε μόνο τις συντεταγμένες (lat / lon) αυτής, αλλά ολόκληρο το Record αντικείμενό της, με όλα τα πεδία ίδια.
*Υπάρχει σχετικό παράδειγμα στο αρχείο 'main.py' μετά από τις εισαγωγές.

- Ερώτημα περιοχής

Για τα ερωτήματα περιοχής (range queries) αρκεί να προσδιορίσουμε ένα σημείο ως το κέντρο του ερωτήματος και μία τιμή radius, που θα δείχνει την απόσταση που πρέπει να ψάξουμε από το κέντρο, προκειμένου να βρούμε τα entries κοντά σε αυτό.
Η διαδικασία είναι αναδρομική, ξεκινώντας από τον κόμβο της ρίζας και ουσιαστικά ελέγχει για κάθε entry αν υπάρχει overlap με την ακτίνα.
(δηλαδη απόσταση από το MBR του entry με το κέντρο <= από την τιμή του radius).
Π.χ.:



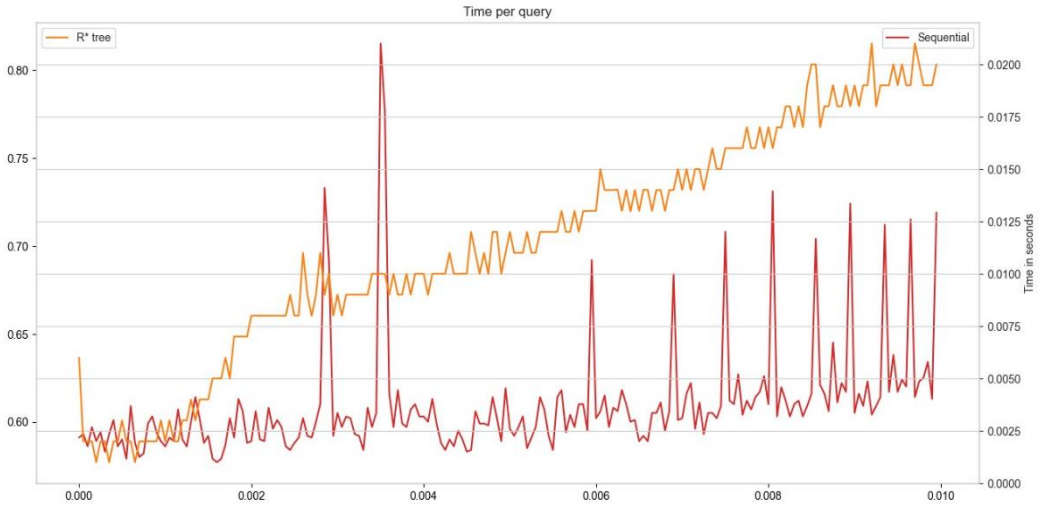
Με κέντρο το κόκκινο σημείο και ακτίνα = 2, παίρνουμε ως αποτέλεσμα:

```
query = range_query(point=(2, 1), radius=2)
query.search(Tree.root)
```

```
----------Range Query----------
Points inside the radius : [(1, 1), (1, 2), (4, 1)]
------End of the Query-------
```

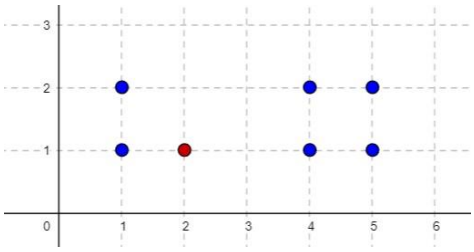
Ακόμη, αν εισάγουμε όλα τα records που έχουμε αποθηκευμένα στο datafile.bin κι αυξήσουμε σταδιακά την τιμή του radius, παρατηρούμε ότι υπάρχει αύξηση του χρόνου εκτέλεσης του κάθε query(παρουσιάζετε σύγκριση με σειριακή αναζήτηση).

```
radius_values = [i for i in np.arange(0, .1, .0005)]
query = range_query(point=(40.5872698, 22.9706448), radius=radius)
```



- Ερώτημα Κ-πλησιέστερων γειτόνων

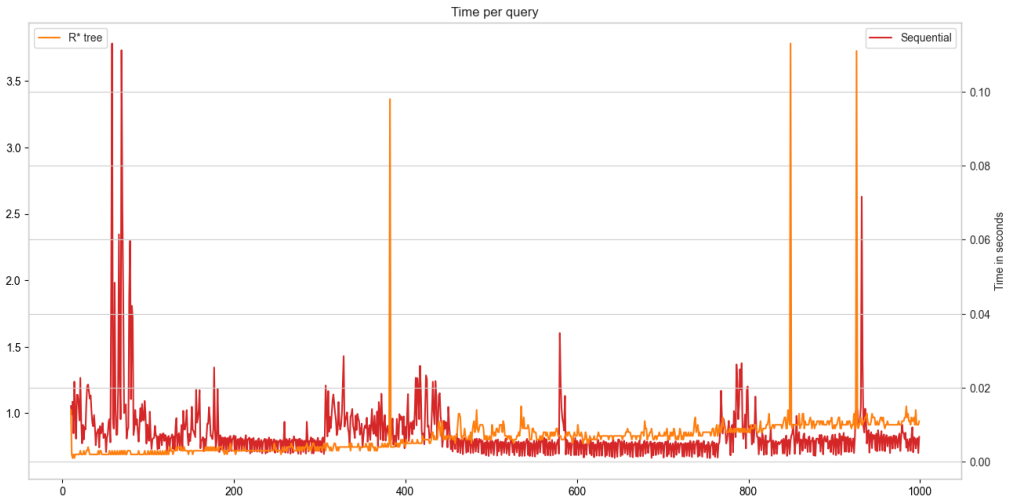
Για τα ερωτήματα πλησιέστερων γειτόνων, χρησιμοποιήσαμε σαν βάση τον αλγόριθμο branch and bound που περιγράφετε στο paper ‘Nearest-Neighbor Queries’. Προκειμένου να αποδείξουμε την σωστή του λειτουργεία, πάλι ως τοy παράδειγμα παρουσιάζουμε το ίδιο που δείξαμε και στα ερωτήματα περιοχής. Έτσι τα αποτελέσματα για αριθμό γειτόνων = 4 και κεντρικό σημείο το (2, 1) είναι:




```
query = Knn(point=(2, 1), k_neighbors=4)
query.Knn_query(Tree.root)
```

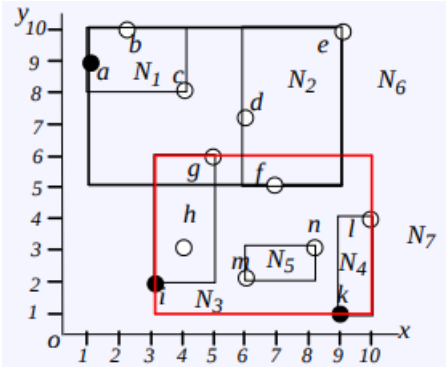
```
----------Knn Query----------
Nearest points : [(1, 1), (1, 2), (4, 1), (4, 2)]
------End of the Knn query-------
```

Ακόμη, αν εισάγουμε και παλι όλα τα records που έχουμε αποθηκευμένα στο datafile.bin κι αυξήσουμε σταδιακά την τιμή του Κ(γειτόνων), παρατηρούμε ότι υπάρχει επίσης σταδιακή αύξηση του χρόνου εκτέλεσης του κάθε query(παρουσιάζετε σύγκριση με σειριακή αναζήτηση).



- Ερώτημα κορυφογραμμής

Με οδηγό τα slides του μαθήματος με τίτλο ‘Ερωτήματα κορυφογραμμής (slides)’, έγινε η υλοποίηση των skyline queries. Για τις δομές δεδομένων που χρησιμοποιήθηκαν, αλλά και το αλγόριθμο αναζήτησης, μπορείτε να ανατρέξετε στο αρχείο ‘skyline.py’, όπου κάνει αναδρομική αναζήτηση των σημείων που κυριαρχούν των υπολοίπων. Προκειμένου να φανεί η σωστή λειτουργία του αλγορίθμου, υλοποιήσαμε το παράδειγμα των διαφανειών με τα ακριβώς ίδια σημεία:



action	heap contents	S
access root	<e_7,4><e_6,6>	⊘
expand e_7	<e_3,5><e_6,6><e_5,8><e_4,10>	⊘
expand e_3	**<i,5>**<e_6,6><e_5,8> <e_4,10>	{i}
expand e_6	<e_5,8><e_1,9><e_4,10>	{i}
remove e_5	<e_1,9><e_4,10>	{i}
expand e_1	**<a,10>**<e_4,10>	{i,a}
expand e_4	**<k,10>**	{i,a,k}

```
data = [(1, 9), (2, 10), (3, 2), (4, 3), (4, 8), (5, 6), (6, 2), (6, 7), (7, 5), (8, 3), (9, 1), (10, 4)]
```

```
----------skyline----------
skyline points : [(3, 2), (9, 1), (1, 9)]
------end of skyline-------
```