Σωτήριος Παναγιώτου, 4456

Διαχείριση Σύνθετων Δεδομένων
Assignment 2

Μέρος 1: Κατασκευή R-Tree μέσω Bulk Loading

Input: 2 αρχεία txt, coords.txt και offsets.txt ως arguments στο command line.
Παράδειγμα εκτέλεσης: python3 ask1.py 'coords.txt' 'offsets.txt'

Output: R-tree, δημιουργια αρχείου Rtree.txt με contents ολα τα nodes του δεντρου και print τον αριθμο των κομβων σε καθε επιπεδο.

Εξήγηση κώδικα:

Αρχικά ελέγχουμε αν η εντολη που δινουμε κατα την εκτελεση εχει το σωστο format και αν τα αρχεια που δινουμε μεσω arguments ειναι valid.
Επειτα φτιαχνουμε μια λιστα “objects” που περιεχει ολα τα αντικειμενα με τις συντεταγμενες τους.

Μετα, με την εντολη sortMBR(objects) κανουμε τα εξης:
calculateMBR(objects): για καθε object, παιρνουμε τις συντεταγμενες και βρισκουμε το MBR τους. Ετσι προκυπτει μια λιστα `MBRlist` που ουσιαστικα περιεχει τα MBR του καθε object.
calcZOrder(MBRlist): για καθε αντικειμενο, κανε z-order encode το κεντρο του καθε object, και αποθηκευσε τα objects σε μια λιστα `ZorderList`. Το καθε object μεσα σε αυτη τη λιστα θα εχει format [zOrderEncoded, [objectId, MBR]]
Επειτα κανουμε sort την λιστα ZorderList με κριτηριο sort του string zOrderEncoded και τελος κανουμε return αυτη τη sorted λιστα.

(Χρησιμοποιουνται και οι εντολες MBR, MBR2 οπου μετατρεπουν λιστες με πολλες συντεταγμενες σε MBR. Η διαφορα του MBR απο το MBR2 ειναι απλα οτι χρησιμοποιουν διαφορετικο format λιστας στο ορισμα)

Τωρα εχουμε οτι χρειαζομαστε, οποτε ξεκιναμε τη δημιουργια του R-tree.

Η μεθοδος που υλοποιησα δημιουργει ενα R-tree με το εξης format:

[[επιπεδο0] , [επιπεδο1] , [επιπεδο2] , ….. , [επιπεδοN]] οπου N ειναι ο αριθμος των συνολικων επιπεδων που χρειαστηκαν να δημιουργηθουν.

Για να μπουμε σε ενα επιπεδο πρεπει να κανουμε:
Δεντρο[αριθμοςΕπιπεδου]
Μεσα σε αυτη τη λιστα περιεχονται τα nodes που βρισκονται μεσα στο επιπεδο.

Για να κανουμε access τα nodes μεσα σε ενα επιπεδο κανουμε:
Δεντρο[αριθμοςΕπιπεδου][αριθμοςNode στο επιπεδο (0 μεχρι length του επιπεδου-1)]
(καθε node περιεχει Max 20, min 20*0.4 εγγραφες)

Εχω χωρισει τη δημιουργια του δεντρου σε 2 μεθοδους.
~Δημιουργια των φυλων, create() οπου παιρνει σαν ορισμα το sortedMBR που υπολογισαμε στην μεθοδο sortMBR

~Δημιουργια του υπολοιπου δεντρου επαναληπτικα μεχρι το root, finishUp() που παιρνει σαν ορισμα την λιστα των φυλων που υπολογισαμε με το create(sortedMBR)

Δημιουργια Φυλων create(sortedMBR):
Εχουμε maxPerLeaf = 20, minPerLeaf = maxPerLeaf*0.4
Φτιαχνουμε ενα node και το γεμιζουμε με αντικειμενα της λιστας **sortedMBR** μεχρι το length του node που φτιαξαμε να ειναι ισο με το maxPerLeaf. Οσο υπαρχουν κι αλλα αντικειμενα στην sortedMBR, φτιαχνουμε κι αλλα nodes και τα γεμιζουμε με την ιδια λογικη.
Οταν φτασουμε στο τελευταιο φυλο, τσεκαρουμε αν τα αντικειμενα που βαλαμε στο node ειναι λιγοτερα απο minPerLeaf. Αν ισχυει, τοτε διαγραφουμε τα "theloume" (theloume = ποσα αντικειμενα χρειαζομαστε για να εχουμε ακριβως minPerLeaf objects στο τελευταιο φυλλο) τελευταια αντικειμενα απο το προηγουμενο φυλο και τα μεταφερουμε στο τελευταιο φυλο.
Και επιστρεφουμε την λιστα με τα φυλα.

Επειτα, παιρνουμε αυτη τη λιστα με τα φυλα και την δινουμε στο finishUp() ετσι ωστε να ολοκληρωθει η δημιουργια του δεντρου.

Ακολουθειται ακριβως η ιδια τακτικη με την δημιουργια των φυλων.
Το `mode` που εχω βαλει ειναι:
Την 1η φορα του while θα γινουν processed τα φυλα. Τα φυλα εχουν διαφορετικο format απο το υπολοιπο δεντρο, οποτε οσο γινονται processed τα φυλα εχω mode=1. Μετα το mode γινεται 0.

Χρησιμοποιουνται οι μεθοδοι toNormalMBR και toNormalMBR2 οπου μετατρεπουν πολλαπλες λιστες με συντεταγμενες σε 1 μοναδικο MBR.

Τελος, για τη δημιουργια του Rtree ζητατε ενα συγκεκριμενο format για το Rtree.
Για αυτο εχω τη μεθοδο writeRtree(fullTree). Μπαινει σε καθε επιπεδο και κανει append τα nodes στο Rtree.txt
Το Rtree.txt στο τελος θα ειναι ενα valid array.

Μέρος 2: Ερωτήσεις Εύρους (Range queries)

Input: 2 αρχεία txt, Rtree.txt και Rqueries.txt ως arguments στο command line.
Παράδειγμα εκτέλεσης: python3 ask2.py 'Rtree.txt' 'Rqueries.txt'

Output: ID αντικειμενων που βρισκονται μεσα στο καθε query

Εξήγηση κώδικα:

Αρχικά ελέγχουμε αν η εντολη που δινουμε κατα την εκτελεση εχει το σωστο format και αν τα αρχεια που δινουμε μεσω arguments ειναι valid.
Φορτωνουμε το Rtree.txt σαν λιστα σε αυτη την ασκηση με τη βοηθεια της βιβλιοθηκης json αφου το Rtree.txt που φτιαξαμε στην ασκηση 1 εχει valid μορφη array.

Επειτα φτιαχνουμε μια λιστα "queries" που περιεχει τα queries απο το αρχειο Rqueries.txt σε μορφη λιστας με floats.

Εφτιαξα μια μεθοδο που ξεκιναει απο το root του Rtree και βρισκει την τομη του query με το current node. Αν η τομη υπαρχει, ελεγχουμε το isnotleaf. Αν ειναι φυλο τοτε βαζουμε το id του node μεσα σε μια λιστα “success”, αλλιως ψαχνουμε αναδρομικα μεσα στο notleaf node μεχρι να φτασουμε σε leaf node.
Τελος επιστρεφουμε την λιστα “success” και printαρουμε τα αποτελεσματα για το καθε query.

Μέρος 3: Ερωτήσεις Πλησιέστερου Γείτονα (kNN queries)

Input: 2 αρχεία txt, Rtree.txt, NNqueries.txt και τον αριθμο πλησιεστερων γειτονων k ως arguments στο command line.
Παράδειγμα εκτέλεσης: python3 ask3.py 'Rtree.txt' 'NNqueries.txt' 10

Output: ID k πλησιεστερων αντικειμενων στο καθε query

Εξήγηση κώδικα:

Αρχικά ελέγχουμε αν η εντολη που δινουμε κατα την εκτελεση εχει το σωστο format και αν τα αρχεια που δινουμε μεσω arguments ειναι valid.
Φορτωνουμε το Rtree.txt σαν λιστα σε αυτη την ασκηση με τη βοηθεια της βιβλιοθηκης json αφου το Rtree.txt που φτιαξαμε στην ασκηση 1 εχει valid μορφη array.

Επειτα φτιαχνουμε μια λιστα “NNqueries” που περιεχει τα queries απο το αρχειο NNqueries.txt σε μορφη λιστας με floats.

Μεταφραζω σε python τον ψευδοκωδικα που αναφερεται στην τελευταια σελιδα του Tutorial 2 στο ecourse

```
function BF NN search(object q, R–tree R) {
    initialize a min-heap Q;
        add all entries of R’s root to Q;
        while more NNs are needed:
                get_next BF NN search(q, Q)
    }

function get_next BF NN search(object q, priority queue Q): object nextNN {
    while not empty(Q)
        e := top(Q);
        remove e from Q;
        if e is a directory node entry then
            n := R–tree node with address e.ptr;
            for each entry e' ∈ n
                    add e' on Q;
        elseif e is an entry of a leaf node
            o := object with address e.ptr;
            add o on Q;
        else /* e is an object */
            /* found next NN */
            return o
}
```

Αλλα με τις εξης αλλαγες:
-Σε καθε heappush, εκει που βαζω τα entries καποιου node στην Q, τα entries εχουν την μορφη:
[distance(q, entryMBR), isnotleaf, entrylist]
-Οταν το current Node (e) ειναι entry of a leaf node, θετω το isnotleaf με 3 και το κανω push στην Q ετσι ωστε οταν το current Node ειναι το leaf node, θα παει στο 2ο elif οπου ελεγχει αν e[1] == 3, τοτε θα σημαινει οτι βρηκε το next NN και θα το κανει return.

Οπου distance(q, entryMBR) ειναι η αποσταση του query απο το πλησιεστερο σημειο του MBR. Οι περιπτωσεις που παιρνω για τα σημεια του MBR ειναι οι γωνιες του MBR, τα κεντρα των πλευρων και το εσωτερικο του MBR, δηλαδη 9 περιπτωσεις συνολικα.

Αυτες οι περιπτωσεις φαινονται αναλυτικα στην μεθοδο `calc_dist`, η οποια χρησιμοποιει μια μεθοδο `dist` αν το q “διαγωνια” απο το MBR (4 περιπτωσεις).

Η dist απλα παιρνει τα αντιστοιχα σημεια και μεσω πυθαγωρειου θεωρηματος υπολογιζει την υποτεινουσα, δηλαδη το distance που θελουμε.

Τελος printαρουμε τα ids απο την λιστα resultList που επιστραφηκε απο τον αλγοριθμο για το καθε query.