

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, δημιουργία αρχείου Rtree.txt με contents όλα τα nodes του δέντρου και print τον αριθμο των κομβων σε καθε επιπεδο.

Εξήγηση κώδικα:

Αρχικά ελέγχουμε αν η εντολή που δινουμε κατα την εκτελεση εχει το σωστο format και αν τα αρχια που δινουμε μεσω arguments ειναι valid.

Επειτα φτιαχνουμε μια λιστα “objects” που περιεχει ολα τα αντικειμενα με τις συντεταγμενες τους.

Μετα, με την εντολή `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 $20 \cdot 0.4$ εγγραφες, εκτος απο το root)

Εχω χωρισει τη δημιουργια του δέντρου σε 2 μεθοδους.

~Δημιουργια των φυλλων, `create()` οπου παιρνει σαν ορισμα το `sortedMBR` που υπολογισαμε στην μεθοδο `sortMBR`

~Δημιουργία του υπολοίπου δέντρου επαναληπτικά μέχρι το root, finishUp() που παίζει σαν ορίσμα την λίστα των φύλλων που υπολογίσαμε με το create(sortedMBR)

Δημιουργία Φύλλων create(sortedMBR):

Εχουμε $\text{maxPerLeaf} = 20$, $\text{minPerLeaf} = \text{maxPerLeaf} * 0.4$

Φτιαχνουμε ένα node και το γεμίζουμε με αντικείμενα της λίστας **sortedMBR** μέχρι το length του node που φτιαξαμε να είναι ίσο με το maxPerLeaf. Όσο υπάρχουν κι άλλα αντικείμενα στην sortedMBR, φτιαχνουμε κι άλλα nodes και τα γεμίζουμε με την ίδια λογική.

Όταν φτάσουμε στο τελευταίο φύλλο, τσεκάρουμε αν τα αντικείμενα που βάλαμε στο node είναι λιγότερα από minPerLeaf. Αν ισχύει, τότε διαγράφουμε τα “theloume” (theloume = ποσα αντικείμενα χρειαζόμαστε για να έχουμε ακριβώς minPerLeaf objects στο τελευταίο φύλλο) τελευταία αντικείμενα από το προηγούμενο φύλλο και τα μεταφέρουμε στο τελευταίο φύλλο.

Και επιστρέφουμε την λίστα με τα φύλλα.

Επειτα, παίζουμε αυτή τη λίστα με τα φύλλα και την δίνουμε στο finishUp() έτσι ώστε να ολοκληρωθεί η δημιουργία του δέντρου.

Ακολουθείται ακριβώς η ίδια τακτική με την δημιουργία των φύλλων.

Το `mode` που έχω βάλει είναι:

Την 1η φορά του while θα γίνουν processed τα φύλλα. Τα φύλλα έχουν διαφορετικό format από το υπόλοιπο δέντρο, οπότε όσο γίνονται processed τα φύλλα έχω mode=1. Μετά το mode γίνεται 0.

Χρησιμοποιούνται οι μέθοδοι toNormalMBR και toNormalMBR2 όπου μετατρέπουν πολλαπλές λίστες με συντεταγμένες σε 1 μοναδικό MBR.

Τέλος, για τη δημιουργία του Rtree ζητάτε ένα συγκεκριμένο format για το Rtree.

Για αυτό έχω τη μέθοδο writeRtree(fullTree). Μπαινει σε κάθε επίπεδο και κάνει append τα nodes στο Rtree.txt

Το Rtree.txt στο τέλος θα είναι ένα 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 αντικειμένων που βρίσκονται μέσα στο κάθε query window

Εξήγηση κώδικα:

Αρχικά ελέγχουμε αν η εντολή που δίνουμε κατά την εκτέλεση έχει το σωστό format και αν τα αρχεία που δίνουμε μέσω arguments είναι valid.

Φορτώνουμε το Rtree.txt σαν λίστα σε αυτή την άσκηση με τη βοήθεια της βιβλιοθήκης json αφού το Rtree.txt που φτιαξαμε στην άσκηση 1 έχει valid μορφή array.

Επειτα φτιαχνουμε μια λίστα “queries” που περιέχει τα queries από το αρχείο Rqueries.txt σε μορφή λίστας με floats.

Εφτιαξα μια μεθοδο που ξεκινει απο το root του Rtree και βρισκει την τομη του query με το current node. Αν η τομη υπαρχει, ελεγχουμε το isnotleaf. Αν ειναι φυλλο 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 printαrouμε 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 point

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
}
```

Αλλά με τις εξής αλλαγές:

-Σε κάθε heappush, εκεί που βάζω τα entries κάποιου node στην Q, τα entries έχουν την μορφή:

[distance(q, entryMBR), isnotleaf, entrylist]

-Όταν το current Node (e) είναι entry of a leaf node, θέτω το isnotleaf με 3 και κάνω push το object στην Q έτσι ώστε όταν το current Node είναι το object (δηλαδή isnotleaf = 3), θα πάει στο 2ο elif όπου ελέγχει αν $e[1] == 3$, τότε θα σημαίνει ότι βρήκε το next NN και θα το κάνει return.

Όπου distance(q, entryMBR) είναι η απόσταση του query από το πλησιέστερο σημείο του MBR. Οι περιπτώσεις που παίρνω για τα σημεία του MBR είναι οι γωνίες του MBR, τα κέντρα των πλευρών και το εσωτερικό του MBR, δηλαδή 9 περιπτώσεις συνολικά.

Αυτές οι περιπτώσεις φαινόνται αναλυτικά στην μέθοδο `calc_dist`, η οποία χρησιμοποιεί μια μέθοδο `dist` αν το q βρισκείται “διαγώνια” από το MBR (4 περιπτώσεις).

Η dist απλά παίρνει τα αντιστοιχά σημεία και μέσω πυθαγορείου θεωρήματος υπολογίζει την υποτεινούσα, δηλαδή το distance που θέλουμε.

Τέλος printάρουμε τα ids από την λίστα resultList που επιστραφίκε από τον αλγόριθμο για το κάθε query.