Assignment 2

r e a d m e

ΣΩΤΗΡΙΑ ΚΑΣΤΑΝΑ, 2995

Εκκίνηση προγράμματος : py assignment2.py data_rectangles.txt

<u>ΜΕΡΟΣ Ι</u> → *Κατασκευή R-tree στην μνήμη*

Τεχνική STR bulk loading : →

- Αρχικά διαβάζονται τα ορθογώνια από το αρχείο, ταξινομούνται με βάση την x-low τιμή τους και τα αποθηκεύονται σε μια λίστα ‘rects_sorted’.
- Γνωρίζοντας ότι το ‘block_size’ είναι 1024 bytes και κάθε εγγραφή χρειάζεται 36 bytes για να αποθηκευτεί (record_size), με την διαίρεση block_size/record_size υπολογίζεται το πλήθος των εγγραφών που χωράνε σε κάθε κόμβο (‘capacity’).
- Παίρνοντας το μήκος της ταξινομημένης λίστας ‘rects_sorted’ βρίσκεται το πλήθος των αντικειμένων (‘rects_objs’). Για να βρεθεί το πλήθος των φύλλων (‘leaves’) εφαρμόζεται ο τύπος : ⌈rects_objs/capacity⌉ . Αφού θέλουμε το ανώφλι αυτής της διαίρεσης, και γνωρίζοντας πως το ανώφλι ενός πραγματικού αριθμού είναι ο μικρότερος ακέραιος που είναι μεγαλύτερος ή ίσος αυτού του πραγματικού, γίνεται cast το αποτέλεσμα αυτής της διαίρεσης σε int και προστίθεται σε αυτό 1 (προστίθενται η μονάδα, γιατί με το cast ενός πραγματικού σε int γίνεται στρογγυλοποίηση προς τα κάτω, ενώ εμείς θέλουμε το άνω φράγμα) .
- Έπειτα βρίσκοντας την ρίζα του συνολικού πλήθους των φύλλων (και εδώ ανώφλι, όπως πάνω) διαβάζεται ανά τμήματα τέτοιου μεγέθους η ταξινομημένη λίστα ‘rects_sorted’ και αναταξινομείται με βάση την y-low τιμή. Στην συνέχεια δημιουργούνται όλα τα φύλλα, έπειτα όλοι οι κόμβοι του αμέσως επόμενου επιπέδου (γονείς φύλλων), έπειτα όλοι οι κόμβοι του αμέσως επόμενου επιπέδου των γονέων των φύλλων κοκ μέχρι να δημιουργηθεί στο τέλος η ρίζα. Έτσι επιτυγχάνεται η κατασκευή του R-tree.

Δημιουργία φύλλων (*layer 1*) **:**

- Αρχικά πρώτα δημιουργούνται όλα τα φύλλα και μετά δημιουργούνται οι γονείς τους
- Μέσα σε μια επανάληψη για όλα τα φύλλα, πλην του δεξιότερου (τελευταίου), δημιουργείται ο κάθε κόμβος/φύλλο και αφού δημιουργηθεί προστίθενται στον μεγάλο πίνακα με όλους τους κόμβους ‘nodes’. Ο κάθε κόμβος/φύλλο είναι ένας πίνακας 32 θέσεων (ARRA) όπου στις 4 πρώτες θέσεις του αποθηκεύει τα στοιχεία του κόμβου, δηλαδή το id του, το πλήθος των εγγραφών που περιέχει, αν είναι τελευταίος κόμβος ή όχι και το επίπεδο που βρίσκεται . Στις επόμενες 28 θέσεις, κάθε θέση είναι ένας μικρότερος πίνακας 5 θέσεων

(arra) όπου κρατάει το object id του αντικειμένου που δείχνει και στις επόμενες 4 θέσεις τις συντεταγμένες του mbr αυτού του αντικειμένου. Η δημιουργία του πίνακα arra και η προσθήκη του στον πίνακα ARRA γίνεται με χρήση της συνάρτησης ‘leaf_node’ .

- Στην συνέχεια ακολουθείται η ίδια διαδικασία και για το δεξιότερο φύλλο, με την μόνη διαφορά ότι η επανάληψη που διατρέχει τα ορθογώνια των αντικειμένων ώστε να τα προσθέσει ως εγγραφές στο φύλλο τώρα δεν φτάνει μέχρι το capacity αναγκαστικά, καθώς είναι ο μόνος κόμβος που μπορεί να περιέχει λιγότερες εγγραφές .
- Έτσι αφού έχει ολοκληρωθεί η διαδικασία δημιουργίας των φύλλων, και αφού καθ’ολη την διάρκεια υπολογιζόταν το εμβαδό κάθε εγγραφής και αποθηκευόταν σε κάθε θέση της λίστας ‘areas’ , με πρόσθεση όλων των θέσεων και διαίρεση με το πλήθος όλων των εμβαδών βρίσκεται το μέσο εμβαδό του επιπέδου των φύλλων όπου και αποθηκεύεται στην λίστα ‘numofnodes_meanarea’μαζί με το επίπεδο και το πλήθος των φύλλων.

Δημιουργία υπόλοιπων κόμβων :

- Πρώτα δημιουργούνται όλοι οι γονείς των φύλλων, έπειτα όλοι οι γονείς αυτών κοκ μέχρι να δημιουργηθεί η ρίζα. Έτσι η παρακάτω διαδικασία ακολουθείται για όλα τα επίπεδα :
- Αρχικά γίνονται προσπελάσεις στον πίνακα ‘nodes’ ξεκινώντας από την πρώτη θέση(n =0) μέχρι να βρεθεί ο πρώτος κόμβος που βρίσκεται στο ακριβώς προηγούμενο επίπεδο . Όταν βρεθεί, αποθηκεύεται στην μεταβλητή ‘my_layer_first_node’ το id του, ώστε να χρησιμοποιηθεί παρακάτω για να εντοπιστεί αν ένας κόμβος είναι ο τελευταίος του επιπέδου.
- Έπειτα για κάθε κόμβο του ‘nodes’ γίνεται η ίδια διαδικασία ώστε να προστεθούν οι εγγραφές των παιδιών στον κόμβο του πατέρα μέχρις ότου βρεθεί ο τελευταίος κόμβος του επιπέδου.
- Στη δημιουργία τόσο των τελευταίων κόμβων του επιπέδου, όσο και των υπόλοιπων χρησιμοποιείται η συνάρτηση ‘node_record’ η οποία είναι αντίστοιχη της ‘leaf_record’ με την διαφορά ότι εδώ για την δημιουργία των συντεταγμένων κάθε mbri που εγγράφεται στον γονέα , καθώς είναι ορθογώνιο ορθογωνίων, ταξινομούνται 4 φορές οι εγγραφές του κάθε παιδιού του μια με βάση το x-low, μια με βάση το x-high, μια με βάση το y-low και μια με βάση το y-high ώστε ως x-low του νέου ορθογωνίου των ορθογωνίων (max_low_X) αποθηκεύεται η 1η θέση της πρώτης θέσης (δηλαδή το min από όλα τα x-lows) , ως x-high (max_high_X) αποθηκεύεται η 2η θέση της τελευταίας θέσης (δηλαδή το max από όλα τα x-highs) και αντίστοιχα για τα y-low και y-high (σημειώνεται ότι χρησιμοποιούνται οι μεταβλητές m1 και m2 για την περίπτωση που είναι ο τελευταίος κόμβος και δεν είναι γεμισμένες όλες οι θέσεις με τις εγγραφές) .
- Όταν δημιουργείται και ο τελευταίος κόμβος κάθε επιπέδου, όπως και στο επίπεδο των φύλλων υπολογίζεται ο μέσος όρος των εμβαδών των mbrs του επιπέδου και αποθηκεύονται στον ίδιο πίνακα ‘numofnodes_meanarea’ .
- Η παραπάνω διαδικασία ακολουθείται μέχρις ότου φτάσουμε σε κάποιον κόμβο όπου θα είναι και ο πρώτος και ο τελευταίος του επιπέδου, δηλαδή έχει χωρητικότητα εγγραφών < από την προκαθορισμένη χωρητικότητα (ή δλδ όλοι οι κόμβοι του προηγούμενου επιπέδου <= capacity). Αυτό σημαίνει ότι δημιουργήθηκε η ρίζα και με την βοήθεια μιας μεταβλητής ‘root’ σταματάει η διαδικασία δημιουργίας κόμβων και συνεπώς έχει ολοκληρωθεί και η κατασκευή του R-tree.
- Στο τέλος γράφονται στο αρχείο ‘rtree.txt’ μαζί με το id της ρίζας, το ύψος του δέντρου και όλοι οι κόμβοι με την σειρά δημιουργίας και την μορφή : node-id, n, (ptr1, MBR1), (ptr2, MBR2), …, (ptrn, MBRn), όπου node-id είναι το id το κόμβου και η θέση του στον πίνακα ‘nodes’, n είναι το πλήθος εγγραφών σε κάθε κόμβο και (ptri, MBRi) οι εγγραφές μέσα στον κόμβο.

<u>ΜΕΡΟΣ ΙΙ</u> → ***Ερωτήσεις στο R-tree***

- Αρχικά διαβάζεται το αρχείο ‘query_rectangles.txt’ και δημιουργείται μια λίστα ‘q’ από τις ερωτήσεις που περιέχονται σε αυτό.
- Για κάθε τύπο ερώτησης δημιουργείται μια αντίστοιχη συνάρτηση (‘Range_Intersect_Query’, ’Range_Inside_Query’ και ’Containment_Query’), οι οποίες καλούνται για κάθε ερώτημα από μια φορά η κάθε μια, με ορίσματα: το ερώτημα(‘qr’), την ρίζα του r-tree(‘nodes[len(nodes)-1]’ → γιατί είναι στη τελευταία θέση του πίνακα nodes), τον αριθμό αποτελεσμάτων(‘rects_intersect/inside/contain’) και των κόμβων(‘N’) που προσπελαύνονται για να αποτιμηθεί η ερώτηση. Στην συνέχεια με κλήση άλλης συνάρτησης(‘f’) αποφασίζουν είτε αν θα καλέσουν τον εαυτό τους αναδρομικά, με κόμβο τώρα αυτόν που έχει id το ptr της εγγραφής που ικανοποίησε τον έλεγχο, είτε αν θα συνεχίσουν στην επόμενη εγγραφή του κόμβου. (Υπενθυμίζεται πως για να προσπελαστούν οι εγγραφές κόμβου ξεκινάει η προσπέλαση από το 4, καθώς αυτή είναι η θέση που έχει οριστεί σαν την θέση της πρώτης εγγραφής, και φτάνει μέχρι node[1]+4, καθώς στο node[1] υπάρχει το μήκος των εγγραφών που υπάρχουν στον κόμβο) . Πιο συγκεκριμένα οι συναρτήσεις αποτίμησης ερωτήσεων επιλογής ακολουθούν την μορφή :

```
Range-Query(query q, R-tree node n) :
if n is leaf node
      for each index entry e in n
            such that f ( q, e. MBR )
                  rectangles ++
else   // n is a non-leaf
      for each index entry e in n
            such that f ( q, e. MBR )
                  nodes ++
                        visit node n’ pointed by e.ptr Range-Query ( q, n’ )
```

(solution 1, Tutorial 2: Spatial Data Management)

- Με την συνάρτηση f να συμβολίζει μια από τις ακόλουθες:
  ‘intersects’ , ‘inside’ και ‘contains’ οι οποίες συγκρίνουν με ελέγχους τις συντεταγμένες του ορθογωνίου του ερωτήματος και του ορθογώνιου της εγγραφής του κόμβου, και επιστρέφουν True σε περίπτωση που τα 2 ορθογώνια τέμνονται/περιέχονται/περιέχουν το ένα το άλλο, διαφορετικά επιστρέφουν False. Πιο αναλυτικά:
  - intersect : αν οι low συντεταγμένες (σε έναν από τους 2 άξονες x,y) του ενός είναι μεγαλύτερες από τις high του άλλου τότε σημαίνει ότι δεν τέμνονται, δηλαδή δεν έχουν κανένα κοινό σημείο και συνεπώς επιστρέφει False, διαφορετικά επιστρέφει True.
  - Inside : αν οι low συντεταγμένε του qr είναι <= και οι high του >= από τις αντίστοιχες του ορθογωνίου που περιέχει το αντικείμενο τότε σημαίνει ότι το ορθογώνιο περιέχεται μέσα στο ορθογώνιο qr
  - contains : αντίστοιχα με το inside, με διαφορετική φορά των >, <
    (για τις inside, contains υπάρχει η μια if κάτω από την άλλη και όχι με and γιατί δεν το έπαιρνε σωστά για κάποιο λόγο)
- Σημειώνεται πως το Intersection είναι υπερσύνολο των Inside και Containment, έτσι στην ’Range_Inside_Query’ όπου ζητείται να βρεθούν τα ορθογώνια που περιέχονται στο ορθογώνιο qr, ο έλεγχος καλώντας την ‘inside’ γίνεται μόνο στα φύλλα και σε όλα τα υπόλοιπα επίπεδα γίνεται με κλήση της ‘intersect’ για να ελεγχθεί αν τέμνονται οι χώροι των 2 ορθογωνίων. Αυτό συμβαίνει καθώς αν ο έλεγχος με ‘inside’ ξεκινούσε από την ρίζα θα χανόταν αποτελέσματα, αφού θα υπήρχαν κόμβοι που τα mbr τους θα ήταν μεγαλύτερα από

το ορθογώνιο qr και θα απορρίπτονταν, ενώ το μονοπάτι τους θα κατέληγε σε φύλλα με ορθογώνια αντικειμένων μικρότερα από αυτά του qr

- Από την άλλη ενώ και στην ’Containment_Query’, όπου στόχος είναι να βρεθούν τα ορθογώνια που περιέχουν το qr, η ίδια ιδέα (δηλαδή καλώντας την ‘contains’ μόνο στα φύλλα και στα υπόλοιπα επίπεδα ‘intersects’) προφανώς δουλεύει, προτιμάται η χρήση της ‘contains’ εξαρχής από την ρίζα για τον μόνο λόγο ότι οι προσπελάσεις των κόμβων για την αποτίμηση του ερωτήματος θα είναι πολύ λιγότερες. Αυτό συμβαίνει καθώς οι εγγραφές των ορθογωνίων που υπάρχουν στην ρίζα σ’ένα r-tree, στην χειρότερη θα έχουν συντεταγμένες >= από αυτές των φύλλων αφού τα mbrs των εγγραφών αυτών περιέχουν άλλα mbrs ορθογωνίων τα οποία με την σειρά τους περιέχουν άλλα κοκ φτάνοντας στο επίπεδο των φύλλων. Έτσι εφόσον περιέχουν και τα ορθογώνια των αντικειμένων στα φύλλα, θα περιέχουν σίγουρα και όσα ορθογώνια qr περιέχονται μέσα σε αυτά.
- Η κάθε συνάρτηση Range-Query καλείται κάθε φορά για έναν κόμβο, όπου προσπελαύνονται οι εγγραφές του. Έτσι το πλήθος κόμβων που προσπευλαύνονται αποθηκεύεται στον μετρητή ‘N’ και αυξάνεται κάθε φορά με το που ισχύει η συνθήκη (‘f’→ return True), δηλαδή πριν κληθεί αναδρομικά η συνάρτηση Range-Query με καινούριο κόμβο (δλδ ο κόμβος με id το ptr της εγγραφής με την οποία η ‘f’ return True) .
- Τέλος για κάθε ερώτημα τυπώνονται τα αποτελέσματα των ορθογωνίων του rtree (δηλαδή των εγγραφών των φύλλων) που αποτιμούν την κάθε ερώτηση καθώς και το πλήθος των κόμβων

Παρακάτω παρατίθενται screenshots με τα αποτελέσματα:

ΜΕΡΟΣ Ι

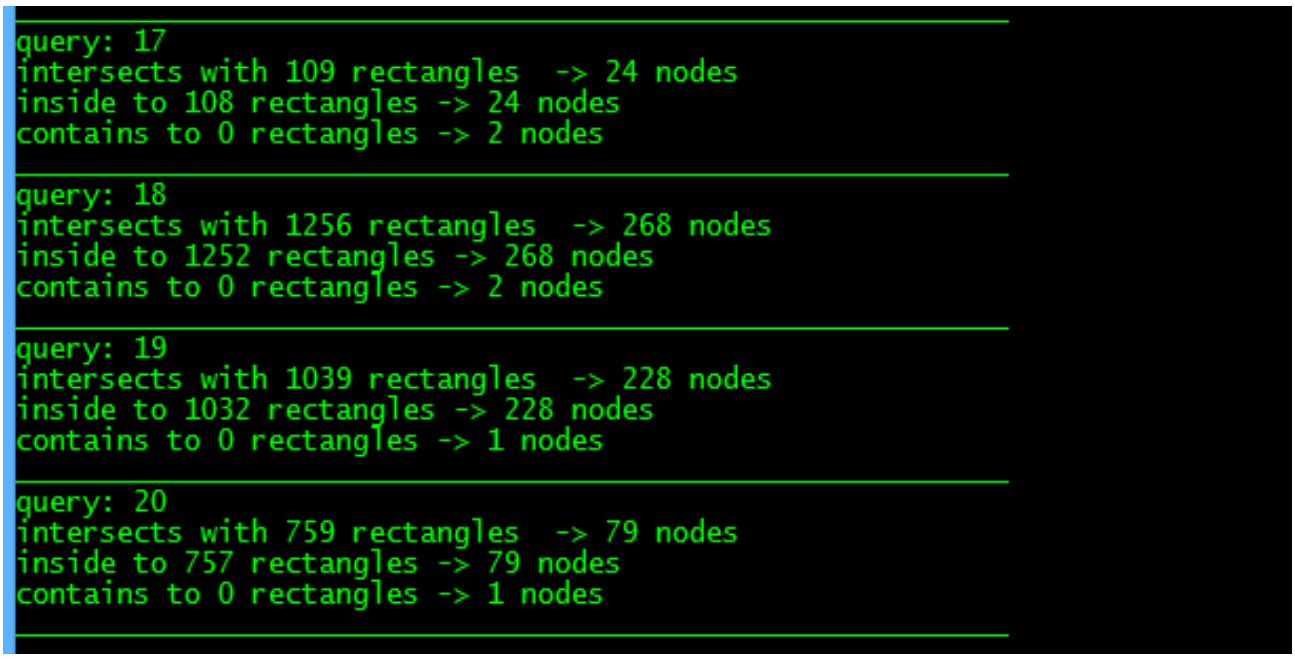
```
C:\Users\Σωτηρια\Desktop\mye041>py assigment2.py data_rectangles.txt
h arithmisi twn nodes ksekinaei apo 0!!
height of R-tree is: 4 layers
# nodes in layer: 1 are: 4516 with mean_area: 5.79871534203e-09
# nodes in layer: 2 are: 162 with mean_area: 1.83283903137e-05
# nodes in layer: 3 are: 6 with mean_area: 0.000673195525645
# nodes in layer: 4 are: 1 with mean_area: 0.0224490708243
```

ΜΕΡΟΣ ΙΙ

```
query: 0
intersects with 513 rectangles  -> 97 nodes
inside to 512 rectangles -> 97 nodes
contains to 0 rectangles -> 2 nodes
_____________________________________
query: 1
intersects with 1446 rectangles  -> 214 nodes
inside to 1437 rectangles -> 214 nodes
contains to 0 rectangles -> 1 nodes
_____________________________________
query: 2
intersects with 1346 rectangles  -> 347 nodes
inside to 1336 rectangles -> 347 nodes
contains to 0 rectangles -> 1 nodes
_____________________________________
query: 3
intersects with 46 rectangles  -> 23 nodes
inside to 46 rectangles -> 23 nodes
contains to 0 rectangles -> 2 nodes
_____________________________________
query: 4
intersects with 1529 rectangles  -> 367 nodes
inside to 1523 rectangles -> 367 nodes
contains to 0 rectangles -> 1 nodes
_____________________________________
```

```
query: 5
intersects with 144 rectangles  -> 38 nodes
inside to 141 rectangles -> 38 nodes
contains to 0 rectangles -> 2 nodes
_____________________________________________________
query: 6
intersects with 357 rectangles  -> 47 nodes
inside to 355 rectangles -> 47 nodes
contains to 0 rectangles -> 2 nodes
_____________________________________________________
query: 7
intersects with 222 rectangles  -> 59 nodes
inside to 220 rectangles -> 59 nodes
contains to 0 rectangles -> 2 nodes
_____________________________________________________
query: 8
intersects with 319 rectangles  -> 59 nodes
inside to 315 rectangles -> 59 nodes
contains to 0 rectangles -> 2 nodes
_____________________________________________________
query: 9
intersects with 656 rectangles  -> 150 nodes
inside to 654 rectangles -> 150 nodes
contains to 0 rectangles -> 1 nodes
_____________________________________________________
query: 10
intersects with 1 rectangles  -> 7 nodes
inside to 0 rectangles -> 7 nodes
contains to 1 rectangles -> 7 nodes
_____________________________________________________
query: 11
intersects with 117 rectangles  -> 50 nodes
inside to 116 rectangles -> 50 nodes
contains to 0 rectangles -> 2 nodes
_____________________________________________________
query: 12
intersects with 1837 rectangles  -> 331 nodes
inside to 1828 rectangles -> 331 nodes
contains to 0 rectangles -> 1 nodes
_____________________________________________________
query: 13
intersects with 1137 rectangles  -> 239 nodes
inside to 1131 rectangles -> 239 nodes
contains to 0 rectangles -> 2 nodes
```

```
query: 14
intersects with 1365 rectangles  -> 252 nodes
inside to 1358 rectangles -> 252 nodes
contains to 0 rectangles -> 2 nodes
_____________________________________________________
query: 15
intersects with 1 rectangles  -> 7 nodes
inside to 0 rectangles -> 7 nodes
contains to 1 rectangles -> 7 nodes
_____________________________________________________
query: 16
intersects with 1479 rectangles  -> 215 nodes
inside to 1469 rectangles -> 215 nodes
contains to 0 rectangles -> 2 nodes
```


query: 17
intersects with 109 rectangles -> 24 nodes
inside to 108 rectangles -> 24 nodes
contains to 0 rectangles -> 2 nodes
query: 18
intersects with 1256 rectangles -> 268 nodes
inside to 1252 rectangles -> 268 nodes
contains to 0 rectangles -> 2 nodes
query: 19
intersects with 1039 rectangles -> 228 nodes
inside to 1032 rectangles -> 228 nodes
contains to 0 rectangles -> 1 nodes
query: 20
intersects with 759 rectangles -> 79 nodes
inside to 757 rectangles -> 79 nodes
contains to 0 rectangles -> 1 nodes