

# Εργασία Τεχνολογίας Βάσεων Δεδομένων 2022-2023

**Φοιτητές:** Δημήτρης Μάρκου (ΑΕΜ: 3550)

Σωτήριος Λουκάς Καμπύλης (ΑΕΜ:3805)

SOS -> Η παρούσα εργασία υλοποιήθηκε από την αρχή του ακαδημαϊκού εαρινού εξαμήνου 2022-2023 αλλά κατά την πορεία προέκυψαν κάποιες δυσκολίες (κυρίως ως προς τον χρόνο που μας είχε απομείνει ως ομάδα) οπότε επιλέχθηκε να συνεχιστεί ακόμα και μέσα στο καλοκαίρι. Ωστόσο, και οι 2 φοιτητές που πραγματοποίησαν την εργασία έχουν πραγματοποιήσει ένα μεγάλο μέρος της υλοποίησης όλων των κλάσεων (όπου χρησιμοποιήθηκε κάτι αυτούσια αναφέρεται ρητά μέσα στα σχόλια στις κλάσεις που υπάρχουν στο src file). Μάλιστα, η πραγματοποίηση της εργασίας ξεκίνησε με διαφορετικές ομάδες αλλά στην πορεία οι συνεργάτες μας τα παράτησαν και περίπου τον Μάιο ξεκινήσαμε την συνεργασία μεταξύ μας!

## ΕΙΣΑΓΩΓΗ:

Κατά τη διάρκεια της έρευνας μας, πραγματοποιήσαμε μια εις βάθος ανάλυση σχετικά με τις περιπλοκές του R\* Tree, μιας πιο προηγμένης έκδοσης των συμβατικών R δέντρων που έχουμε συναντήσει στο παρελθόν. Η εστίαση μας ήταν κυρίως στην κατανόηση του αντίκτυπου που έχει αυτή η συγκεκριμένη δομή σε δύο συγκεκριμένους τύπους ερωτημάτων: ερωτήματα

εύρους (*Range Queries*) και ερωτήματα πλησιέστερου γείτονα (*KNN Queries*). Εξετάζοντας διεξοδικά αυτούς τους τύπους ερωτημάτων, στοχεύαμε να αποκτήσουμε γνώσεις σχετικά με την αποτελεσματικότητα και την αποδοτικότητα της δομής R\* Tree στον χειρισμό τους. Για να παρέχουμε μια ολοκληρωμένη σύγκριση, πραγματοποιήσαμε επίσης πειραματικές αξιολογήσεις όπου εφαρμόσαμε και τις δύο μεθόδους ερωτημάτων χρησιμοποιώντας μια απλοϊκή προσέγγιση, όπως η σειριακή αναζήτηση. Αυτό μας επέτρεψε να εκτιμήσουμε καλύτερα και να αξιολογήσουμε την αξία που φέρνει στο τραπέζι η συγκεκριμένη δομή του δέντρου R\*. Η δημιουργία των τελικών αλγορίθμων που εκτελούν τις λειτουργίες της δομής πραγματοποιήθηκε με τη χρήση της προγραμματιστικής γλώσσας Java (στο περιβάλλον του IntelliJ). Ο κώδικας αναλαμβάνει τις βασικές λειτουργίες της δομής χρησιμοποιώντας χωρικά δεδομένα που προέρχονται από τη σελίδα του μαθήματος στο e-learning. Επιπλέον, πραγματοποιήθηκαν δοκιμές με διάφορες βάσεις δεδομένων από διάφορες πηγές, όπως το Open Street Maps, το μεγαλύτερο ηλεκτρονικό χάρτη με ελεύθερη άδεια. Προτού αναφερθούμε στις λεπτομέρειες του κώδικα και της υλοποίησης, πρέπει να καθορίσουμε μερικές από τις παραμέτρους που πρέπει να ληφθούν υπόψη όσον αφορά τη λειτουργία της δομής δεδομένων. Καταρχάς, ο κώδικας είναι διαμορφωμένος έτσι ώστε να υποστηρίζει δεδομένα πολλών διαστάσεων, όχι μόνο δυσδιάστατα στοιχεία. Επιπλέον, το πρόγραμμα που έχουμε δημιουργήσει λαμβάνει τα δεδομένα από ένα ξεχωριστό αρχείο που ονομάζεται "datafile.dat". Συγκεκριμένα, έχουμε υλοποιήσει και τις 6 δομές που έπρεπε (+κάποια πράγματα ακόμα όπως η ταξινόμηση Hilbert, η οποία βασίστηκε σε κώδικα στο διαδίκτυο) με απόλυτη επιτυχία.

## **Οι βασικές δομές που ζητάει οι εκφώνηση να υλοποιήσουμε είναι οι εξής:**

### ***1)Εισαγωγή Εγγραφής (Insert)***

Αυτή η κλάση υλοποιεί τη λειτουργία εισαγωγής για το R\* Δέντρο. Περιλαμβάνει μεθόδους για την εισαγωγή μιας εγγραφής, τη διαχείριση της υπερχείλισης και τη μαζική εισαγωγή εγγραφών κατά τη δημιουργία του αρχείου δεδομένων. Παρέχει επίσης μια μέθοδο για τη χειροκίνητη εισαγωγή μιας εγγραφής στο R\* Δέντρο. Αρχικά, η μεταβλητή `overflow_first_time` χρησιμοποιείται για να ελέγξει αν πρώτη φορά συναντήθηκε υπερχείλιση και η `overflowLevel` αποθηκεύει το επίπεδο της υπερχείλισης. Η μέθοδος `insert` εισάγει μια εγγραφή στο R\* Δέντρο. Αρχικά, χρησιμοποιεί τη μέθοδο `ChooseSubtree` για να επιλέξει τον καλύτερο κόμβο για την εισαγωγή της εγγραφής. Στη συνέχεια, διαβάζει το αρχείο δείκτη και αντιγράφει τον κόμβο που πρέπει να ενημερωθεί σε έναν πίνακα από bytes. Ελέγχει αν υπάρχει ακόμα χώρος στον κόμβο για την εισαγωγή της εγγραφής. Αν υπάρχει, εισάγει την εγγραφή και ενημερώνει τον αριθμό των κόμβων στον κόμβο. Αν ο κόμβος είναι γεμάτος, καλεί τη μέθοδο `overflowTreatment` για την αντιμετώπιση της υπερχείλισης. Η μέθοδος `overflowTreatment` χειρίζεται την υπερχείλιση όταν ένας κόμβος είναι γεμάτος. Αν το επίπεδο της υπερχείλισης δεν είναι το ίδιο με το τρέχον επίπεδο του δέντρου, τότε ορίζει το επίπεδο υπερχείλισης και ορίζεται ως πρώτη φορά υπερχείλισης. Αν το επίπεδο του δέντρου δεν είναι μηδέν και η σημαία υπερχείλισης είναι true, τότε καλεί τη μέθοδο `reinsert` για την επανεισαγωγή της εγγραφής. Διαφορετικά, καλεί τη

μέθοδο `split` για τη διάσπαση του κόμβου. Η μέθοδος `datafileMassInsert` εισάγει πολλαπλές εγγραφές στο αρχείο δεδομένων κατά τη διαδικασία δημιουργίας του αρχείου δεδομένων. Διαβάζει το αρχείο δεδομένων και αντιγράφει τον κόμβο που πρέπει να ενημερωθεί σε έναν πίνακα `bytes`. Στη συνέχεια, επαναλαμβάνει για κάθε εγγραφή, ελέγχοντας αν υπάρχει αρκετός χώρος στον κόμβο για την εισαγωγή της εγγραφής. Αν υπάρχει αρκετός χώρος, εισάγει την εγγραφή. Αν όχι, γράφει τον κόμβο στο αρχείο, ενημερώνει τον αριθμό των κόμβων και ξαναγράφει τον κόμβο στο αρχείο. Τέλος, γράφει τον τελευταίο κόμβο στο αρχείο. Η μέθοδος `datafileRecordInsert` εισάγει χειροκίνητα μια εγγραφή στο  $R^*$  Δέντρο και το αρχείο δεδομένων. Διαβάζει τον τελευταίο κόμβο στο αρχείο δεδομένων και αντιγράφει τον κόμβο σε έναν πίνακα από `bytes`. Στη συνέχεια, εντοπίζει το τέλος του κόμβου και ελέγχει αν υπάρχει αρκετός χώρος για την εισαγωγή της εγγραφής. Αν υπάρχει αρκετός χώρος, εισάγει την εγγραφή. Αν όχι, γράφει τον κόμβο στο αρχείο, ενημερώνει τον αριθμό των κόμβων και ξαναγράφει τον κόμβο στο αρχείο. Τέλος, γράφει τον τελευταίο κόμβο στο αρχείο και εισάγει την εγγραφή στο  $R^*$  Δέντρο.

## 2) Διαγραφή Εγγραφής (*Deletion*)

Η κλάση `Delete` παρέχει τις απαραίτητες μεθόδους για τη διαγραφή ενός σημείου από το δέντρο. Η σταθερά `minEntries` υπολογίζει το ελάχιστο πλήθος καταχωρήσεων που πρέπει να υπάρχουν σε έναν κόμβο για να μην θεωρείται υποχρεωτική υπερχείλιση. Αυτό εξαρτάται από το μέγεθος του `block` και την αναλογία 40% που χρησιμοποιείται. Η μέθοδος `delete` δέχεται τις συντεταγμένες `LAT` και `LON` του σημείου που πρέπει να

διαγραφεί και καλεί την `deletePoint` για την πραγματική διαγραφή του. Η μέθοδος `deletePoint` είναι η κύρια μέθοδος για τη διαγραφή ενός σημείου από το δέντρο. Χρησιμοποιεί μια ουρά `pointers` για να διασχίσει το δέντρο και βρίσκει το σημείο που πρέπει να διαγραφεί. Όταν βρει το σημείο, καλεί τη μέθοδο `deletePointFromBlock` για να διαγράψει το σημείο από τον κατάλληλο κόμβο. Επιστρέφει `true` αν το σημείο διαγράφηκε με επιτυχία, αλλιώς επιστρέφει `false`. Η μέθοδος `deletePointFromBlock` διαγράφει ένα σημείο από έναν καθορισμένο κόμβο του δέντρο. Ελέγχει το περιεχόμενο του `block`, βρίσκει το σημείο που πρέπει να διαγραφεί, διαγράφει το σημείο και ενημερώνει το `block` κατάλληλα. Στη συνέχεια, ελέγχει αν χρειάζεται να γίνει επαναδιαμόρφωση του δέντρο ή αν απλά πρέπει να διαγραφεί το `block`. Αν ο κόμβος δεν είναι στη ρίζα και δεν έχει φτάσει στο ελάχιστο πλήθος καταχωρήσεων, τότε γίνεται επαναδιαμόρφωση των ορίων των ορθογωνικών περιοχών που καλύπτουν τα σημεία. Σε διαφορετική περίπτωση, το `block` διαγράφεται και τα σημεία επανεισάγονται στο δέντρο.

### *3) Ερώτημα Περιοχής (RangeQuery)*

Η κλάση `RangeQuery` αντιπροσωπεύει μια ερώτηση εύρους (Range Query) σε ένα  $R^*$  Tree, μια δομή δεδομένων που χρησιμοποιείται για την αναζήτηση δεδομένων χωρικού περιεχομένου. Ας αναλύσουμε τα βασικά χαρακτηριστικά του κώδικα, η `rangeRectangle` αντιπροσωπεύει το ορθογώνιο παράθυρο (rectangle) της ερωτήσεως εύρους που χρησιμοποιείται για την αναζήτηση, η `result` αντιπροσωπεύει μια λίστα που θα περιέχει τα αποτελέσματα της ερωτήσεως εύρους, η `pointers` αντιπροσωπεύει μια ουρά προτεραιότητας

που χρησιμοποιείται για να διαχειριστεί τους δείκτες στα διάφορα επίπεδα του R\* Tree και η dimensions τον αριθμό των διαστάσεων του διανύσματος χωρικού περιεχομένου. Ο κατασκευαστής λαμβάνει ένα ορθογώνιο παράθυρο (rectangle) ως παράμετρο και αρχικοποιεί την ερώτηση εύρους με αυτό το ορθογώνιο παράθυρο. Στη συνέχεια, καλεί τη μέθοδο rangeQuery() για να εκτελέσει την ερώτηση. Η μέθοδος rangeQuery() εκτελεί την ερώτηση εύρους στο R\* Tree. Οι δείκτες που αντιπροσωπεύουν τα επίπεδα του R\* Tree τοποθετούνται σε μια ουρά προτεραιότητας, με αρχικό δείκτη τον ριζικό κόμβο. Η μέθοδος αναζητά το R\* Tree από τη ρίζα προς τα φύλλα, ελέγχοντας αν τα ορθογώνια παράθυρα των εγγραφών επικαλύπτονται με το ορθογώνιο παράθυρο της ερωτήσεως εύρους. Εάν μια εγγραφή βρίσκεται εντός του ερωτήματος εύρους, προστίθεται στη λίστα αποτελεσμάτων.

#### *4) Ερώτημα Πλησιέστερων Γειτόνων (KNNQuery)*

Αυτή η κλάση KnnQuery αντιπροσωπεύει μια ερώτηση K-NearestNeighbors (KNN) σε ένα R\* Tree, μια δομή δεδομένων που χρησιμοποιείται για την αναζήτηση δεδομένων χωρικού περιεχομένου. Ας αναλύσουμε τα βασικά χαρακτηριστικά του κώδικα, η knn μεταβλητή είναι μια ουρά προτεραιότητας που χρησιμοποιείται για την αποθήκευση των k πλησιέστερων γειτόνων (nearestneighbors), η pointers είναι μια ουρά προτεραιότητας που χρησιμοποιείται για τη διαχείριση των δεικτών, η dimensions είναι ο αριθμός των διαστάσεων του χώρου χωρικού περιεχομένου και η coordinates είναι οι συντεταγμένες του σημείου για το οποίο γίνεται η ερώτηση KNN. Ο κατασκευαστής δέχεται τον αριθμό k των πλησιέστερων γειτόνων και τις συντεταγμένες ενός σημείου ως παραμέτρους.

Αρχικοποιεί τις προτεραιότητες ουρές knn και pointers και καλεί τη μέθοδο knnQuery() για να εκτελέσει την ερώτηση KNN. Η μέθοδος knnQuery() εκτελεί την ερώτηση KNN στο R\* Tree. Εξετάζει την απόσταση ανάμεσα στο σημείο και κάθε ορθογώνιο παράθυρο στο αρχείο δεδομένων. Εάν η απόσταση είναι μικρότερη από την απόσταση του πιο μακρινού στοιχείου που έχει βρεθεί μέχρι τώρα, προσθέτει το στοιχείο στην ουρά knn. Οι δείκτες στα επίπεδα του R\* Tree τοποθετούνται στην ουρά προτεραιότητας pointers. Οι μέθοδοι calcDistBetweenPoints() και calcDistBetweenPointAndRectangle() υπολογίζουν την απόσταση μεταξύ σημείων και μεταξύ σημείου και ορθογωνίου παραθύρου αντίστοιχα. Η μέθοδος print() χρησιμοποιείται για να εκτυπώσει τα k πλησιέστερα στοιχεία που βρέθηκαν με την ερώτηση KNN.

### *5) Ερώτημα Κορυφογραμμής (SkylineQuery)*

Η κλάση SkylineQuery φαίνεται να υλοποιεί μια ερώτηση στο R\* Tree για την αναζήτηση των εγγραφών που αποτελούν την "skyline". Ας αναλύσουμε τον κώδικα. Η μεταβλητή pointers είναι μια ουρά προτεραιότητας που χρησιμοποιείται για τη διαχείριση των δεικτών, η result είναι μια λίστα που χρησιμοποιείται για την αποθήκευση των εγγραφών που ανήκουν στη "skyline", η dimensions είναι ο αριθμός των διαστάσεων του χώρου χωρικού περιεχομένου. Ο κατασκευαστής αρχικοποιεί τις μεταβλητές pointers και διαβάζει τις διαστάσεις από το αρχείο FileHandler. Στη συνέχεια, καλεί τη μέθοδο skylineQuery() για να εκτελέσει την ερώτηση "skyline". Η μέθοδος skylineQuery() εκτελεί την ερώτηση "skyline" στο R\* Tree. Η διαδικασία χρησιμοποιεί

διάφορους έλεγχους για την προσθήκη εγγραφών στην "skyline". Οι δείκτες στα επίπεδα του R\* Tree τοποθετούνται στην ουρά προτεραιότητας pointers. Η μέθοδος επαναλαμβάνεται όσο υπάρχουν δείκτες στην ουρά pointers. Η μέθοδος print() χρησιμοποιείται για να εκτυπώσει τις εγγραφές που βρίσκονται στην "skyline".

### *6) Μαζική Κατασκευή του Δένδρου Bottom-Up*

Η κλάση BottomUp φαίνεται να υλοποιεί την δημιουργία ενός R-tree με τη μέθοδο "BottomUp". Ας αναλύσουμε τον κώδικα. Η subset\_recs είναι μια ουρά που χρησιμοποιείται για τη διαχείριση ενός υποσυνόλου των εγγραφών, η records είναι μια λίστα που περιέχει όλες τις εγγραφές από το αρχείο FileHandler, η leafLevelFINAL είναι το επίπεδο των φύλλων του R-tree και η blockID είναι ένα αναγνωριστικό που χρησιμοποιείται για την αναγνώριση των διαφορετικών μπλοκ στο αρχείο. Ο κατασκευαστής αρχικοποιεί τις μεταβλητές της κλάσης, όπως τις εγγραφές, την ουρά subset\_recs, το επίπεδο των φύλλων και το blockID. Η μέθοδος construct() εκτελεί την κατασκευή του R-tree χρησιμοποιώντας τη μέθοδο "BottomUp". Οι εγγραφές ταξινομούνται με τη βοήθεια της μεθόδου HilbertSort. Στη συνέχεια, δημιουργούνται τα μπλοκ για τα φύλλα του R-tree και για τα ενδιάμεσα επίπεδα. Η μέθοδος getLevelsOfTree() υπολογίζει τα επίπεδα του R\* tree βάσει του μεγέθους των εγγραφών. Τέλος, η ιδιωτική μέθοδος calculateLevels() υπολογίζει αναδρομικά τα επίπεδα του R\* tree, όσο το μέγεθος του δέντρου είναι μεγαλύτερο από 1.

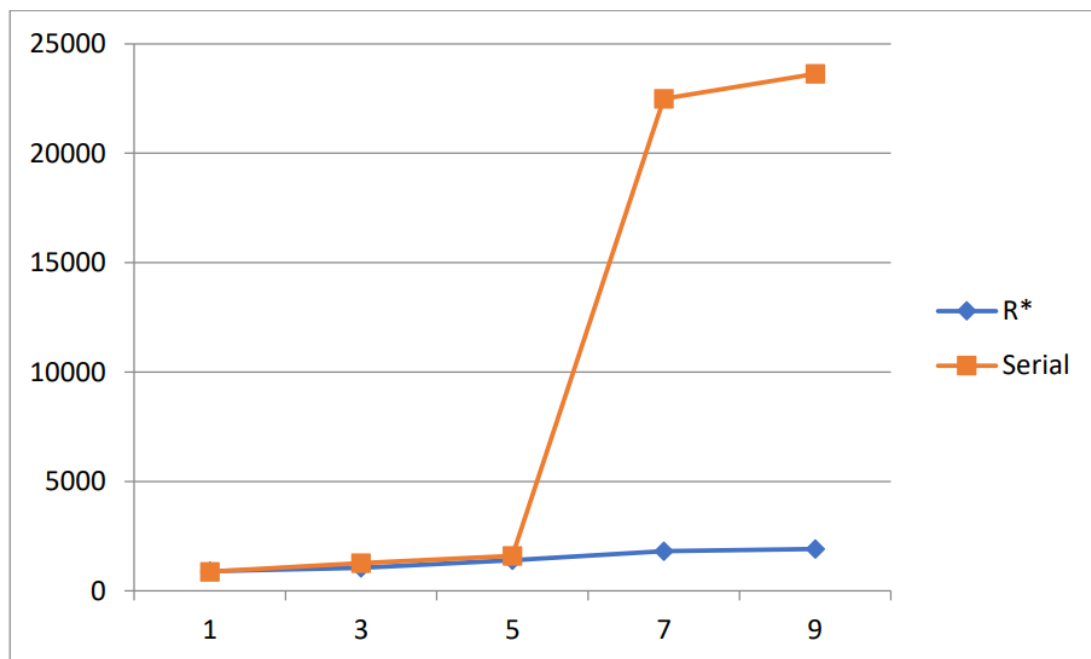


## ΠΕΙΡΑΜΑΤΑ – ΔΟΚΙΜΕΣ

*(Σημείωση: Παρακάτω θα σας παρουσιάσουμε μερικά ενδεικτικά παραδείγματα από χρόνους εκτέλεσης ερωτημάτων των διαφόρων δομών. Δυστυχώς, λόγω έλλειψης χρόνου δεν προλάβσαμε να κάνουμε plots από όλες τις δομές – αν και όλες οι δομές έχουν δοκιμαστεί και τρέχουν σωστά – όπως Skyline Queries κτλ.)*

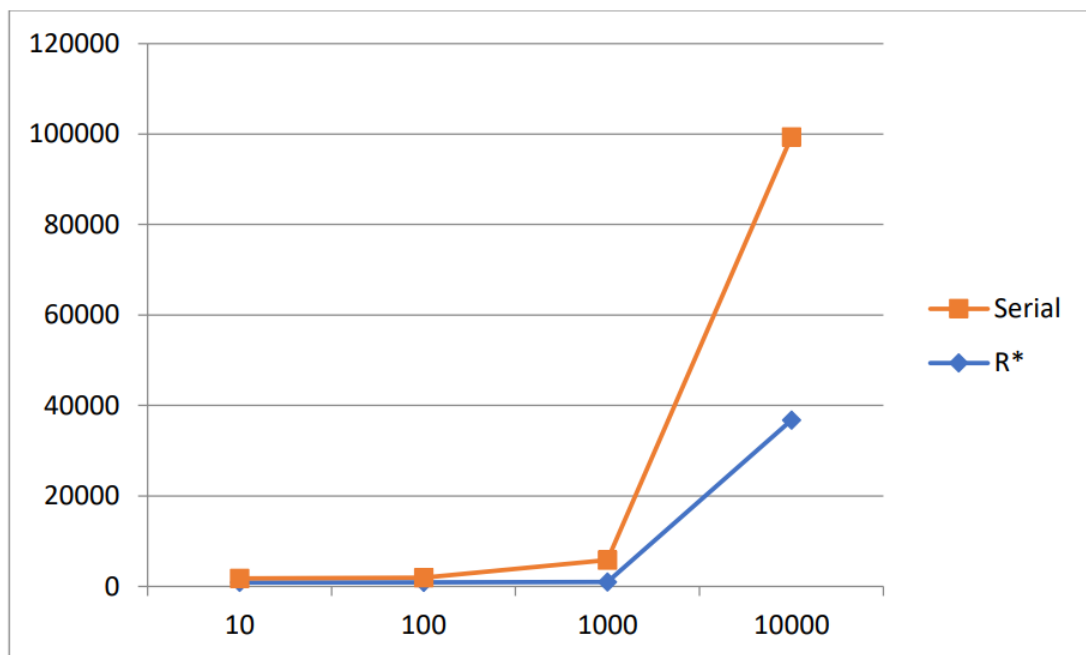
Για να δούμε την αποδοτικότητα της δομής R\* tree που υλοποιήθηκε όσον αφορά τα ερωτήματα περιοχής και των K κοντινότερων γειτόνων τρέξαμε κάποια πειράματα (σε ένα μικρό εύρος δεδομένων) και τα συγκρίναμε σε σχέση με τον αλγόριθμο της σειριακής αναζήτησης. Σε αυτά δεν περιλαμβάνεται ο χρόνος κατασκευής του δέντρου, δεδομένου ότι το δέντρο θα κατασκευαστεί μια φορά και θα αποθηκευτεί. Για τα παραδείγματα χρησιμοποιήθηκε ένα σημείο ενδιαφέροντος και έγινε σταδιακή αύξηση του εύρους και των K γειτόνων.

Range



Σύμφωνα με τα παραπάνω συμπεραίνουμε ότι για μικρές τιμές η γραμμική αναζήτηση έχει την ίδια αποδοτικότητα, ωστόσο για  $R \geq 5$  ο χρόνος εκτέλεσής της αυξάνεται δραματικά.

### K - NN



Σχετικά με τα πειράματα για τον αλγόριθμο KNN έναντι της γραμμικής αναζήτησης παρατηρούμε ότι η γραμμική αναζήτηση καταφέρνει αρκετά καλά να συναγωνιστεί τις αποδόσεις του R\* tree για ένα ικανοποιητικό αριθμό γειτόνων. Παρόλα αυτά και πάλι, όσο το πλήθος γειτόνων αυξάνεται ( $\geq 1000$ ) η επίδοση της γραμμικής αναζήτησης αλλάζει δραματικά καθιστώντας καλύτερη επιλογή την δομή R\*.

Συμπέρασμα: Σύμφωνα λοιπόν με τα παραπάνω μπορούμε να δούμε πως η δομή R\* tree αποτελεί την καλύτερη λύση σε τέτοιου είδους ερωτήματα και έχει καλύτερη κλιμακωσιμότητα όσο αναφορά την είσοδο των παραμέτρων.

# ΕΝΔΕΙΚΤΙΚΑ ΠΑΡΑΔΕΙΓΜΑΤΑ ΤΡΟΠΟΥ ΛΕΙΤΟΥΡΓΙΑΣ

*Παρακάτω παρουσιάζουμε μερικά παραδείγματα (προφανώς όχι όλα γιατί υπάρχουν πολλές διαφορετικές περιπτώσεις) για το πως να ξεκινήσετε την εργασία.*

Αρχικά, πρέπει να δημιουργηθεί το δέντρο (παρουσιάζεται με screenshots μέσα από βήματα):

Έχουμε 2 επιλογές ( α) **Point by Point** ή β) **Bottom Up**). Στο παρακάτω screenshot φαίνεται να το δημιουργούμε Point by Point (έτσι δημιουργείται και το αρχείο *treeOutput.txt*)! Μάλιστα, για να μην δημιουργούμε ξανά και ξανά το δέντρο μπορούμε να κάνουμε Reuse το υπάρχον δέντρο.

```
Menu (Choose something from them)
1) Start,
2) Settings,
3) About
Input: 1

Options:
1) Build,
2) Reuse,
3) Esc
to build the tree, reuse the old files or return to the main menu respectively.
Input: 1

Options (type option or number):
1) Point by point,
2) Bottom-up,
3) ESC
to build the tree inserting the entries one by one, using the bottom-up approach or return to the Start menu respectively
Input: 1

Parsing data...
Tree is building...
Total elapsed time :233.011 seconds

Tree structure is located in treeOutput.txt

Options (type option or number):
1) Insert,
2) Delete,
3) Range Query,
4) K-nn Query,
5) Skyline query,
6) Linear search Range Query,
7) Linear search K-nn Query,
8) ESC
Input:
```

Ύστερα, μπορούμε να χρησιμοποιήσουμε οποιαδήποτε από τις 7 δομές που έχουμε υλοποιήσει. Ενδεικτικά τρέχουμε και παραθέτουμε σε screenshots την **1 (Insert)**, **4 (K-nn Query)** & **5 (Skyline Query)**. Ωστόσο έχουν δοκιμαστεί όλες οι δομές!

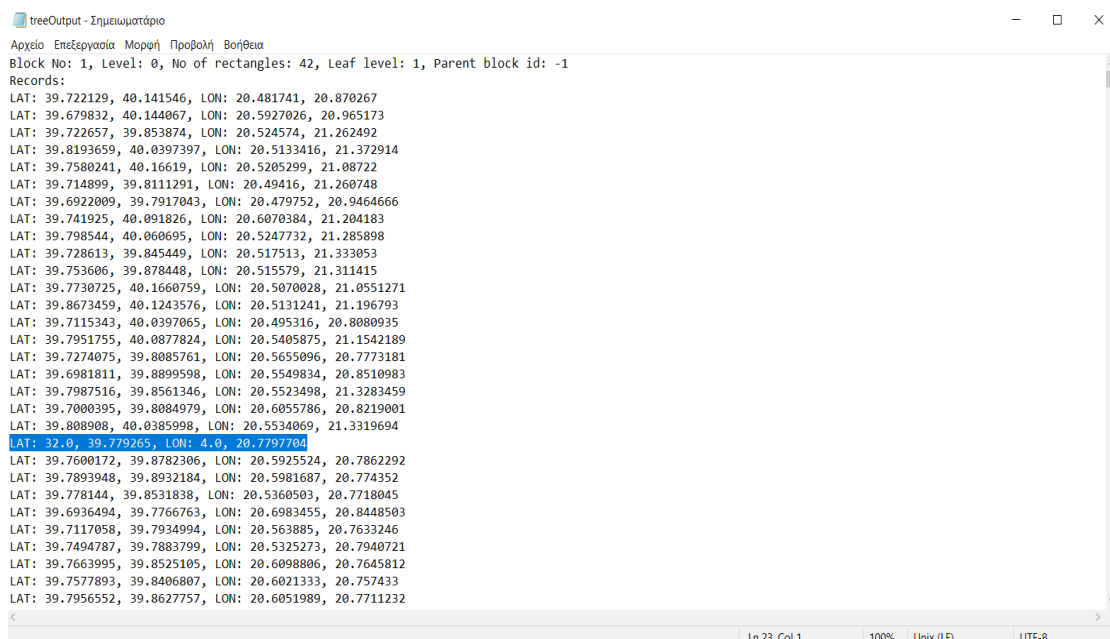
*Μάλιστα, ο αριθμός των διαστάσεων δουλεύει ως παράμετρος (δηλαδή μπορούν να χρησιμοποιηθούν >2 διαστάσεις!)*

## 1 (Insert)

```
Options (type option or number):
1) Insert,
2) Delete,
3) Range Query,
4) K-nn Query,
5) Skyline query,
6) Linear search Range Query,
7) Linear search K-nn Query,
8) ESC
Input: Input: 1

Insert the coordinates of the point (coordinates should be positive Double/Float or Integers)
Lat: 32
Lon: 4
Insert the new node Id (SEE FROM treeOutput.txt): 5
Insert the new node Name or press ENTER if you don't want to name it:
The node with LAT: 32.0, LON: 4.0 and ID: 5 was successfully inserted.

Press ENTER to continue
```



```
treeOutput - Σημειωματάριο
Αρχείο Επεξεργασία Μορφή Προβολή Βοήθεια
Block No: 1, Level: 0, No of rectangles: 42, Leaf level: 1, Parent block id: -1
Records:
LAT: 39.722129, 40.141546, LON: 20.481741, 20.870267
LAT: 39.679832, 40.144067, LON: 20.5927026, 20.965173
LAT: 39.722657, 39.853874, LON: 20.524574, 21.262492
LAT: 39.8193659, 40.0397397, LON: 20.5133416, 21.372914
LAT: 39.7580241, 40.16619, LON: 20.5205299, 21.08722
LAT: 39.714899, 39.8111291, LON: 20.49416, 21.260748
LAT: 39.6922009, 39.7917043, LON: 20.479752, 20.9464666
LAT: 39.741925, 40.091826, LON: 20.6070384, 21.204183
LAT: 39.798544, 40.060695, LON: 20.5247732, 21.285898
LAT: 39.728613, 39.845449, LON: 20.517513, 21.333053
LAT: 39.753606, 39.878448, LON: 20.515579, 21.311415
LAT: 39.7730725, 40.1660759, LON: 20.5070028, 21.0551271
LAT: 39.8673459, 40.1243576, LON: 20.5131241, 21.196793
LAT: 39.7115343, 40.0397065, LON: 20.495316, 20.8080935
LAT: 39.7951755, 40.0877824, LON: 20.5405875, 21.1542189
LAT: 39.7274075, 39.8085761, LON: 20.5655096, 20.7773181
LAT: 39.6981811, 39.8899598, LON: 20.5549834, 20.8510983
LAT: 39.7987516, 39.8561346, LON: 20.5523498, 21.3283459
LAT: 39.7000395, 39.8084979, LON: 20.6055786, 20.8219001
LAT: 39.808908, 40.0385998, LON: 20.5534069, 21.3319694
LAT: 32.0, 39.779265, LON: 4.0, 20.7797704
LAT: 39.7600172, 39.8782306, LON: 20.5925524, 20.7862292
LAT: 39.7893948, 39.8932184, LON: 20.5981687, 20.774352
LAT: 39.778144, 39.8531838, LON: 20.5360503, 20.7718045
LAT: 39.6936494, 39.7766763, LON: 20.6983455, 20.8448503
LAT: 39.7117058, 39.7934994, LON: 20.563885, 20.7633246
LAT: 39.7494787, 39.7883799, LON: 20.5325273, 20.7940721
LAT: 39.7663995, 39.8525105, LON: 20.6098806, 20.7645812
LAT: 39.7577893, 39.8406807, LON: 20.6021333, 20.757433
LAT: 39.7956552, 39.8627757, LON: 20.6051989, 20.7711232
```

## 4 (K-nn Query)

```
Options (type option or number):
1) Insert,
2) Delete,
3) Range Query,
4) K-nn Query,
5) Skyline query,
6) Linear search Range Query,
7) Linear search K-nn Query,
8) ESC
Input: Input: 4

Insert the k (k should be a positive Integer): 4
Insert the coordinates of the point (coordinates should be positive Double/Float or Integers)
Lat: 39
Lon: 40
The 4 nearest neighbors are:
1) Distance: 18.652905042248513, LAT: 39.981088, LON: 21.372914, Datafile block: 11, Block slot: 2288, Node ID: 1506359761
2) Distance: 2.23606797749979, LAT: 40.0, LON: 42.0, Datafile block: 42, Block slot: 19615, Node ID: 20000
3) Distance: 18.68387641902946, LAT: 40.032262, LON: 21.344661, Datafile block: 11, Block slot: 2392, Node ID: 1506359766
4) Distance: 18.686082904461546, LAT: 39.845449, LON: 21.333053, Datafile block: 11, Block slot: 1742, Node ID: 1506359723

Press ENTER to continue
```

## 5 (Skyline Query)

```
8) ESC
Input: 5

There are 33 entries in the skyline:
LAT: 39.728475, LON: 20.479752, Datafile block: 8, Block slot: 14794, Node ID: 1499043109
LAT: 39.727932, LON: 20.47978, Datafile block: 8, Block slot: 14638, Node ID: 1499042892
LAT: 39.727552, LON: 20.479993, Datafile block: 8, Block slot: 14612, Node ID: 1499042789
LAT: 39.727287, LON: 20.48073, Datafile block: 8, Block slot: 14508, Node ID: 1499042739
LAT: 39.726807, LON: 20.48131, Datafile block: 8, Block slot: 14378, Node ID: 1499042622
LAT: 39.72454, LON: 20.481721, Datafile block: 8, Block slot: 13884, Node ID: 1499042151
LAT: 39.72437, LON: 20.483491, Datafile block: 8, Block slot: 13806, Node ID: 1499042123
LAT: 39.72431, LON: 20.484578, Datafile block: 8, Block slot: 13780, Node ID: 1499042118
LAT: 39.723986, LON: 20.489685, Datafile block: 8, Block slot: 13702, Node ID: 1499042057
LAT: 39.72273, LON: 20.49121, Datafile block: 8, Block slot: 13520, Node ID: 1499041872
LAT: 39.722129, LON: 20.494276, Datafile block: 8, Block slot: 13416, Node ID: 1499041809
LAT: 39.721991, LON: 20.495324, Datafile block: 8, Block slot: 12402, Node ID: 1497737668
LAT: 39.721866, LON: 20.502832, Datafile block: 8, Block slot: 13390, Node ID: 1499041778
LAT: 39.720936, LON: 20.503666, Datafile block: 8, Block slot: 13338, Node ID: 1499041680
LAT: 39.719043, LON: 20.504212, Datafile block: 8, Block slot: 13234, Node ID: 1499041295
LAT: 39.717127, LON: 20.506151, Datafile block: 8, Block slot: 13182, Node ID: 1499040936
LAT: 39.715775, LON: 20.506484, Datafile block: 8, Block slot: 13130, Node ID: 1499040645
LAT: 39.715048, LON: 20.507226, Datafile block: 8, Block slot: 13078, Node ID: 1499040433
LAT: 39.714777, LON: 20.509533, Datafile block: 8, Block slot: 13052, Node ID: 1499040342
LAT: 39.7138776, LON: 20.5965595, Datafile block: 42, Block slot: 6760, Node ID: 9001951183
LAT: 39.7117058, LON: 20.6002027, Datafile block: 42, Block slot: 6786, Node ID: 9001951184
LAT: 39.7080156, LON: 20.6019101, Datafile block: 42, Block slot: 6812, Node ID: 9001951185
LAT: 39.7054535, LON: 20.6030766, Datafile block: 42, Block slot: 6838, Node ID: 9001951186
LAT: 39.7020653, LON: 20.6046319, Datafile block: 42, Block slot: 6864, Node ID: 9001951187
LAT: 39.7000395, LON: 20.6055786, Datafile block: 21, Block slot: 32188, Node ID: 5423303307
LAT: 39.700025, LON: 20.7862324, Datafile block: 42, Block slot: 5772, Node ID: 9001918038
LAT: 39.6983195, LON: 20.8243458, Datafile block: 1, Block slot: 5289, Node ID: 299466558
LAT: 39.6961814, LON: 20.8278863, Datafile block: 1, Block slot: 4925, Node ID: 299456893
LAT: 39.6877864, LON: 20.8362292, Datafile block: 1, Block slot: 4535, Node ID: 299456861
LAT: 39.685107, LON: 20.938788, Datafile block: 10, Block slot: 31382, Node ID: 1506359389
LAT: 39.680647, LON: 20.94128, Datafile block: 10, Block slot: 31356, Node ID: 1506359387
LAT: 39.680047, LON: 20.943219, Datafile block: 10, Block slot: 31330, Node ID: 1506359383
LAT: 39.679832, LON: 20.943915, Datafile block: 10, Block slot: 31304, Node ID: 1506359378

Press ENTER to continue
```