Εργασία Τεχνολογίας Βάσεων Δεδομένων

Γραμμένος Θεόδωρος
Α.Ε.Μ: 3294

Μπουας Χρήστος
Α.Ε.Μ.: 3315

3 Ιουλίου 2021

Η εργασία υλοποιήθηκε στη γλώσσα C++. Για το build χρησιμοποιείται το σύστημα CMake. Επίσης μέσα στην εργασία συμπεριλαμβάνεται η βιβλιοθήκη pugixml και χρησιμοποιείται για το parsing XML αρχείων.

1 Datafile

1.1 Τεχνικές Λεπτομέρειες

Στο datafile υπάρχουν 3 βασικές οντότητες:

- Point: Ένα σημείο που περιέχει OSM ID, όνομα και συντεταγμένες.
- Block: Περιέχει πολλά points. Υπάρχει ένα κοινό block size για όλα τα blocks.
- Database: Περιέχει πολλά blocks.

1.1.1 Point

Τα points αποθηκεύονται στο δίσκο ως μια σειρά πεδίων στη σειρά. Παρακάτω φαίνεται η σειρά των πεδίων καθώς και η ερμηνεία τους:

Περιγραφή	Τύπος
Record ID	32-bit unsigned integer > 0
OSM ID	64-bit unsigned integer
Coordinates	Πολλοί double αριθμοί στη σειρά. (Όσοι και οι διαστάσεις)
Μέγεθος Ονόματος	8-bit unsigned integer
Όνομα	Μέγεθος ονόματος char στη σειρά

Πίνακας 1: Point

Από το παραπάνω φαίνεται ότι το μέγεθος του κάθε point είναι διαφορετικό ακόμα και μέσα στο ίδιο datafile αφού υπάρχει το πεδίο του ονόματος το οποίο μπορεί να έχει από 0 έως 255 χαρακτήρες, ανάλογα με το σημείο.

1.1.2 Block

Κάθε Block περιέχει απλώς πολλά Point στη σειρά, το ένα μετά το άλλο. Τα points εισέρχονται σειριακά, από την αρχή του block. Εφόσον δεν γίνονται διαγραφές, ένα Block θεωρείται ότι είναι γεμάτο μέχρι τη θέση που υπάρχει το τελευταίο Point, ενώ από εκεί και μετά είναι κενό.

Κατά την αρχικοποίηση ενός νέου block το datafile επεκτείνεται και ο χώρος του νέου block γεμίζεται με μηδενικά. Αυτό επιτρέπει τον εντοπισμό της αρχής του κενού χώρου, αφού εάν διαβαστεί Record ID = 0 τότε δεν υπάρχουν περεταίρω σημεία στο block.

Το πρώτο block είναι διαφορετικό από τα υπόλοιπα διότι δεν αποθηκεύει δεδομένα, αλλά πληροφορίες για τη βάση δεδομένων. Αυτό το block (metadata block) έχει διαφορετική δομή από τα υπόλοιπα (data blocks) και διαφορετικό μέγεθος.

Περιγραφή	Τύπος
Σύνολο Blocks	32-bit unsigned integer
Block Size (in KiBs)	8-bit unsigned integer
Σύνολο σημείων	32-bit unsigned integer
Διαστάσεις	8-bit unsigned integer

Πίνακας 2: Metadata Block

1.1.3 Database

Η Database είναι υπεύθυνη για τη διαχείριση των block. Κατά την εισαγωγή ενός σημείου ελέγχεται μόνο το τελευταίο block για επαρκή χώρο και θεωρείται ότι τα προηγούμενα είναι αρκετά γεμάτα. Αν υπάρχει επαρκής χώρος, το νέο point εισάγεται σε αυτό το block, αλλιώς δημιουργείται καινούριο.

Μία ακραία περίπτωση είναι αν γίνει εισαγωγή ενός πολύ μικρού σημείου στην αρχή του block και στη συνέχεια γίνει εισαγωγή ενός Point το οποίο είναι ελάχιστα μικρότερο ή ίσο του block size. Αυτό θα προκαλέσει το αρχικό block να έχει ελεύθερο χώρο σχεδόν ίσο με το block size. Αυτη η ακραία περίπτωση μπορεί να αποφευχθεί αν το max name length είναι αρκετά μικρότερο του block size, αφού το name είναι το μόνο πεδίο που μπορεί να έχει μεταβλητό μέγεθος. Με αυτόν τον τρόπο το μέγιστο μέγεθος ενός record μπορεί να κρατηθεί αρκετά μικρότερο του block size, περιορίζοντας τη σπατάλη χώρου. Στην υλοποίησή μας το μέγιστο μέγεθος του ονόματος είναι 256 bytes, ενώ το ελάχιστο μέγεθος block 1KB.

Στην εισαγωγή των σημείων η βάση δεδομένων αναθέτει στο καθένα ένα μοναδικό record ID. Το record ID είναι ένας μοναδικός αριθμός ο οποίος χρησιμοποιείται από τη βάση για να χαρακτηρίζει το κάθε record.

Κατά τη δημιουργία μιας βάσης δεδομένων δίνονται ως παράμετροι το όνομα του αρχείου, το block size και οι διαστάσεις των σημείων που θα αποθηκεύονται (το μέγεθος του metadata block μπορεί να οριστεί μέσω μιας private σταθεράς στην κλάση Database). Οι παράμετροι αυτοί αποθηκεύονται στο metadata block από το οποίο διαβάζονται όταν γίνεται άνοιγμα της βάσης δεδομένων. Με αυτόν τον τρόπο διασφαλίζεται ότι δεν θα γίνει άνοιγμα μιας βάσης με λάθος παραμέτρους.

Ο τρόπος με τον οποίο είναι οργανωμένα τα δεδομένα επιτρέπει μία ακόμα βελτιστοποίηση. Κατά τη φόρτωση της βάσης ελέγχεται ο ελεύθερος χώρος του τελευταίου block και αποθηκεύεται. Οι μετέπειτα εισαγωγές αντί να διαβάζουν συνεχώς το τελευταίο block και να ελέγχουν τον ελευθερό του χώρο, απλώς διαβάζουν και ενημερώνουν τις μεταβλητές. Με αυτόν τον τρόπο η μόνη πρόσβαση στο δίσκο για την εισαγωγή ενός σημείου είναι κατά την εγγραφή των δεδομένων του στο datafile.

2 Index

Η απόδοση του index είναι σε γενικές γραμμές καλή, με εξαίρεση την κατασκευή όπου απαιτούνται γύρω στα 3-4 λεπτά προκειμένου να γίνουν import όλα τα σημεία του αρχείου saloniki.osm.

Το indexfile είναι οργανωμένο σε Nodes. Κάθε node περιέχει ένα byte στην αρχή το οποίο είναι 0 ή 1 ανάλογα με το αν ο κόμβος είναι φύλλο ή όχι. Στη συνέχεια κάθε node περιέχει τα entries του. Η δομή των entries διαφέρει ανάλογα με το αν ο κόμβος είναι φύλλο ή όχι.

Περιγραφή	Τύπος
offset	64-bit unsigned integer
coordinates	Πολλοί double στη σειρά (όσες και οι διαστάσεις)

Πίνακας 3: Entry σε φύλλο

Το offset που αποθηκεύεται στο φύλλο δηλώνει το offset στο datafile στο οποίο βρίσκεται το συγκεκριμένο point + 1. Η προσθήκη του 1 γίνεται για να μην υπάρχει σύγχυση σε περίπτωση που κάποιο point είναι αποθηκευμένο στο offset 0, αφού αν το offset = 0, θεωρούμε ότι ένα entry είναι κενό. Κατά το διάβασμα των offset από το index η κλάση Database κάνει την αφαίρεση προκειμένου να εξάγει το σωστό offset.

Περιγραφή	Τύπος
childOffset	64-bit unsigned integer
MBR	2 πίνακες double στη σειρά μεγέθους όσες και οι διαστάσεις

Πίνακας 4: Entry σε ενδιάμεσο κόμβο

Τα entries στους ενδιάμεσους κόμβους περιλαμβάνουν το MBR τους το οποίο αποθηκεύεται ως 2 πίνακες double στη σειρά μεγέθους όσες και οι διαστάσεις. Ο πρώτος πίνακας δηλώνει το min του rectangle (κάτω αριστερά στις 2 διαστάσεις) και ο δεύτερος το max (πάνω δεξιά στις 2 διαστάσεις).

Αξίζει να σημειωθεί ότι τα offsets που αναφέρονται μέσα στο Index πάντα αναφέρονται στο κομμάτι του node META το αρχικό byte που δηλώνει αν ο κόμβος είναι φύλλο. Αυτό είναι μία παραδοχή που γίνεται σε όλο το Index.

Η υλοποίησή δουλεύει με δυναμικά offsets, δηλαδή τα παιδιά ενός κόμβου μπορεί να είναι διασκορπισμένα οπουδήποτε μέσα στο

αρχείο, ανάλογα με το που δείχνει το πεδίο childOffset. Αυτό επιτρέπει την απαλοιφή μετακινήσεων δεδομένων από το ένα μέρος του αρχείου στο άλλο, παρά μόνο όταν γίνεται κάποιο split και δημιουργείται κάποιος νέος κόμβος στο τέλος του αρχείου. Και σε αυτήν την περίπτωση όμως δημιουγείται μόνο ένας νέος κόμβος στο τέλος, ενώ επαναχρησιμοποιείται ο χώρος που καταλαμβάνει ο αρχικός.

Στην υλοποίηση χρησιμοποιούνται 2 βοηθητικές κλάσεις, η LeafEntry και η NonLeafEntry, που περιέχουν τις πληροφορίες ενός entry σε ένα φύλλο και σε έναν ενδιάμεσο κόμβο αντίστοιχα. Αυτή η διαφοροποίηση έχει οδηγήσει στη δημιουργία 2 παραλλαγών των περισσότερων συναρτήσεων, μία για φύλλα και μία για ενδιάμεσους κόμβους(π.χ. writeLeaf, writeNonLeaf).

Μια επιπλέον βοηθητική κλάση είναι η Rectangle η οποία υλοποιεί ένα απλό ορθογώνιο. Το ορθογώνιο χαρακτηρίζεται από τις συντεταγμένες του low σημείου του και του high σημείου του. Στις 2 διαστάσεις αυτά τα σημεία είναι το κάτω αριστερά και το πάνω δεξιά.

Εκτός από αυτές τις κλάσεις δημιουργήθηκε και ένα γενικό interface Entry το οποίο περιέχει μία μόνο μέθοδο getRectangle. Και οι δύο κλάσεις LeafEntry και NonLeafEntry υλοποιούν το interface. Η μέθοδος LeafEntry επιστρέφει ένα ορθογώνιο του οποίου τα low και high σημεία ταυτίζονται, ενώ η NonLeafEntry επιστρέφει το MBR. Το interface χρησιμοποιείται κυρίως κατά τη διαδικασία του split ενός node προκειμένου να μειωθεί το duplication στον κώδικα.

Οι αλγόριθμοι που χρησιμοποιούνται έχουν αντληθεί από το paper του R*-Tree και τις διαφάνειες του μαθήματος και χρησιμοποιούν τα ίδια ονόματα μέσα στον κώδικα.

Έχει επίσης υλοποιηθεί η συνάρτηση walkTree η οποία κάνει μία BFS διάσχιση του δέντρου, έτσι ώστε χρησιμοποιώντας debugger και breakpoints να μπορεί κάποιος να δει το περιεχόμενό του ανα πάσα στιγμή.

Η συνάρτηση testIndex περιλαμβάνει ένα παράδειγμα ενός απλού R-Tree το οποίο περιλαμβάνει πολλά split node και 2 split της ρίζας. Χρησιμοποιώντας τη συνάρτηση walkTree μπορεί κανείς να δει την κατάσταση του δέντρου και να επιβεβαιώσει ότι όντως οι διαδικασίες γίνονται σωστά.

3 Παραδείγματα

Σημείωση: Στον κώδικα των παραδειγμάτων χρησιμοποιείται το αρχείο saloniki.osm το οποίο περιέχει όλα τα σημεία της Θεσσαλονίκης και

δεν περιλαμβάνεται λόγω υψηλού μεγέθους (70MB), αλλά μπορεί να κατέβει με την εντολή στο αρχείο download.sh

Οι συναρτήσεις testRange και testNearestNeighbors εκτελούν απλά ερωτήματα περιοχής και κοντινότερων γειτόνων αντίστοιχα. Στο output δίνουν μόνο τα σημεία τα οποία έχουν όνομα έτσι ώστε να μπορεί να γίνει ευκολότερα αντιληπτό το αποτέλεσμα.

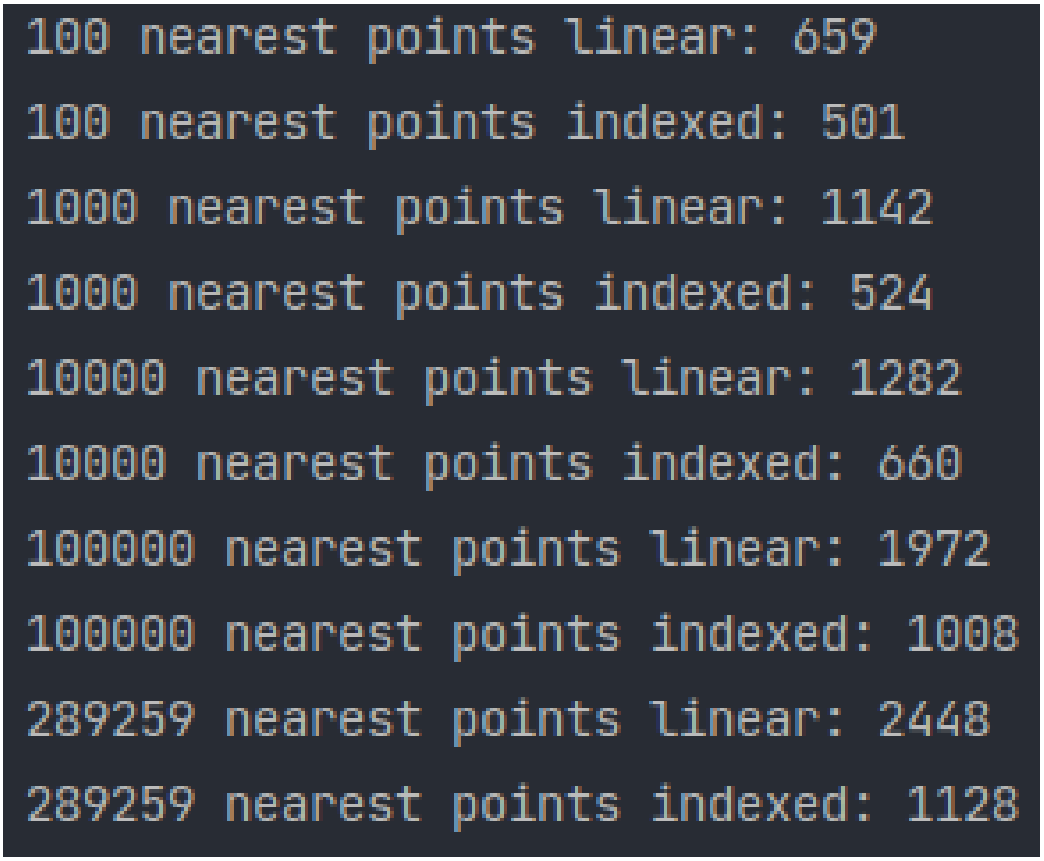
Στη συνάρτηση testIndex δημιουγείται ένα απλό R-Tree το οποίο μπορεί να δημιουργηθεί και να σχεδιαστεί και στο χαρτί. Η κατάσταση του δέντρου μπορεί να ελεγχθεί με τη συνάρτηση walkTree.

4 Benchmarks

Τα tests εκτελέστηκαν σε υπολογιστή με επεξεργαστή Intel Core i5 6500 και με SSD.

4.1 Nearest Neighbors

Το benchmark υλοποιείται στη συνάρτηση timeNearestNeighbors. Παρακάτω φαίνονται τα αποτελέσματα του benchmark. Οι χρόνοι είναι σε ms.



Σχήμα 1: Αποτελέσματα benchmark nearest neighbors.

Αριθμός Σημείων	Χωρίς Index	Με Index
100	659	501
1000	1142	524
10000	1282	660
100000	1972	1008

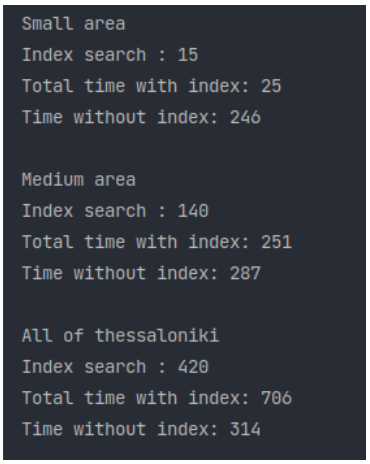
Πίνακας 5: Αποτελέσματα benchmark nearest neighbors.

Παρατηρούμε ότι σε αντίθεση με τα ερωτήματα κορυφής τα ερωτήματα nearest-neigbors είναι σταθερά πιο γρήγορα χρησιμοποιώντας το index. Η διαφορά στην ταχύτητα αυξάνεται καθώς μέσω του index περιορίζονται τα σημεία που χρειάζεται να εξετάσουμε και να υπολογίσουμε τις αποστάσεις.

4.2 Range Search

Το benchmark υλοποιείται στη συνάρτηση timeRange. Εκτελούνται αναζητήσεις σε 3 περιοχές της Θεσσαλονίκης:

- Την small η οποία είναι μία περιοχή κοντά στο μέγαρο μουσικής.
- Την medium η οποία περιέχει περίπου τη μισή Θεσσαλονίκη.
- Την large η οποία περιέχει όλη τη θεσσαλονίκη.



Σχήμα 2: Αποτελέσματα της timeRange

Περιοχή	Small	Medium	Large
Χρόνος Index	15	140	420
Χρόνος index και ανάκτησης σημείων	25	251	706
Χρόνος γραμμικής αναζήτησης	246	287	314

Πίνακας 6: Αποτελέσματα timeRange

Παρατηρούμε ότι η επίδοση του index είναι πολύ καλύτερη όταν η περιοχή αναζήτησης είναι μικρή, αλλά σταδιακά χειροτερεύει καθώς αυξάνεται το μέγεθος της περιοχής. Όταν η περιοχή αναζήτησης καλύπτει περίπου τα μισά σημεία τότε η απόδοση της αναζήτησης με index είναι λίγο καλύτερη από την απλή γραμμική αναζήτηση. Όταν η περιοχή αναζήτησης καλύπτει όλα τα σημεία παρατηρούμε ότι η απόδοση του index είναι πολύ κακή σε σχέση με την απλή γραμμική αναζήτηση λόγω του επιπλέον overhead που υπάρχει κατά την

διάσχιση του δέντρου και την ανάκτηση όλων των σημείων. Επίσης παρατηρούμε ότι οι χρόνοι της γραμμικής αναζήτησης παρουσιάζουν μικρότερη διακύμανση σε σχέση με αυτούς του index.