**ΕΙΣΑΓΩΓΗ**

Η σημερινή εποχή χαρακτηρίζεται ως η εποχή των Μεγάλων Δεδομένων, καθώς ολοένα και μεγαλύτερος όγκος δεδομένων συλλέγεται από τις εταιρείες και τους οργανισμούς. Αυτά τα δεδομένα, που βρίσκονται τόσο σε δομημένη όσο και σε αδόμητη μορφή, περιέχουν πάρα πολλές πληροφορίες, που μπορούν να οδηγήσουν σε χρήσιμα συμπεράσματα σε ένα πάρα πολύ μεγάλο πλήθος πεδίων, τόσο επιστημονικών όσο και επαγγελματικών. Συνεπώς, είναι απολύτως αναμενόμενη η επεξεργασία όλων αυτών των δεδομένων και η αξιοποίηση τους προκειμένου να ληφθούν οι σωστές αποφάσεις και να πραγματοποιηθεί η εξόρυξη της γνώσης.

Φυσικά, προκειμένου να επιτευχθεί ο παραπάνω στόχος είναι απαραίτητη η χρήση των κατάλληλων τεχνολογιών, όπως τεχνολογίες παράλληλης επεξεργασίας δεδομένων και μη σχεσιακές βάσεις δεδομένων.

Σε αυτή την εργασία, η έρευνα επικεντρώθηκε κυρίως στο κομμάτι των μη σχεσιακών βάσεων δεδομένων και συγκεκριμένα στη Mongo DB και τους τρόπους που επιτυγχάνεται η βελτιστοποίηση των ερωτημάτων που αφορούν χωροχρονικά δεδομένα. Ως χωροχρονικά δεδομένα ορίζονται τα δεδομένα που αποτελούνται από 3 βασικές μεταβλητές (longitude, latitude, timestamp), δηλαδή τις συντεταγμένες και τη χρονική στιγμή που η οντότητα βρέθηκε στη συγκεκριμένη θέση. Η ανάλυση των συγκεκριμένων δεδομένων μπορεί να βρει εφαρμογή σε πολλούς τομείς, όπως η ναυτιλία ή ο σχεδιασμός έξυπνων πόλεων.

Στη συνέχεια, παρουσιάζεται ο τρόπος με τον οποίο έχει δομηθεί η εργασία και τα περιεχόμενα των επόμενων ενοτήτων.

**Ανασκόπηση Βιβλιογραφίας:** Στην επόμενη ενότητα πραγματοποιείται μια βιβλιογραφική ανασκόπηση, στην οποία παρουσιάζονται οι τύποι των μη σχεσιακών βάσεων δεδομένων και κάποια σχετικά θεωρήματα.

**Περιγραφή Αλγόριθμων:** Σε αυτή την ενότητα περιγράφονται όλοι οι αλγόριθμοι που χρησιμοποιήθηκαν στην εργασία και παρουσιάζεται ο κώδικας που υλοποιεί τους αλγόριθμους.

**Πειραματική Μελέτη:** Μετά την παρουσίαση των αλγόριθμων εκτελούνται τα πειράματα και παρουσιάζονται τα αποτελέσματα.

**Συμπεράσματα:**

**ΑΝΑΣΚΟΠΗΣΗ ΒΙΒΛΙΟΓΡΑΦΙΑΣ**

**ACID Ιδιότητες**

Το όνομα των ACID ιδιοτήτων προέκυψε από τα αρχικά των ιδιοτήτων που είναι Atomicity, Consistency, Isolation, Durability. Οι συγκεκριμένες ιδιότητες διαβεβαιώνουν ότι το αποτέλεσμα ενός transaction σε μια SQL βάση δεδομένων θα είναι αξιόπιστο. Στη συνέχεια, παρουσιάζεται η επεξήγηση της κάθε ιδιότητας:

* **Atomicity:** Ένα transaction ονομάζεται atomic όταν εκτελεστούν όλες ή καμία από τις προγραμματισμένες ενέργειες. Το atomicity ως ιδιότητα διασφαλίζει ότι αν ένα transaction διακοπεί τότε οι ενέργειες που είχαν εκτελεστεί ως εκείνη τη στιγμή ακυρώνονται και το transaction είναι σαν να μην εκτελέστηκε ποτέ.
* **Consistency:** Η εκτέλεση ενός transaction στη βάση δεδομένων επιστρέφει πάντα την πιο πρόσφατη τιμή.
* **Isolation:** Οι αλλαγές ου πραγματοποιούνται από την εκτέλεση ενός transaction δεν είναι ορατές στα υπόλοιπα transactions μέχρι αυτές να ολοκληρωθούν. Η συγκεκριμένη ιδιότητα διασφαλίζει ότι τα transactions θα εκτελεστούν σειριακά και δεν θα υπάρχει ποτέ ταυτόχρονη εκτέλεση δύο transactions.
* **Durability:** Το durability διασφαλίζει ότι αν ένα transaction ολοκληρωθεί, οι όποιες αλλαγές πραγματοποιήθηκαν στη βάση δεδομένων θα αποθηκευτούν ακόμα και αν στη συνέχεια υπάρξει κάποιο system failure.

Οι παραπάνω απαιτήσεις κατάφεραν για πάρα πολλά χρόνια να βρουν πεδίο εφαρμογής στα RDBMS (Relational DataBase Management Systems), όπου οι βάσεις δεδομένων ήταν μικρότερες, horizontally scalable και schema driven. Πλέον, με την εξέλιξη της τεχνολογίας και την συνεχή και ολοένα αυξανόμενη ανάγκη για αποθήκευση δεδομένων στις βάσεις, το μοντέλο των σχεσιακών βάσεων δεδομένων, καθώς και οι παραπάνω ACID ιδιότητες δεν είναι αρκετές για να καλύψουν την αποθήκευση των Μεγάλων Δεδομένων.

Η σημερινή εποχή, η εποχή των Μεγάλων Δεδομένων, χαρακτηρίζεται από συνεχή αύξηση του όγκου των δεδομένων, καθώς και από αποθήκευση δεδομένων που βρίσκονται σε unstructured ή semi-structured μορφή, τα οποία φυσικά δεν ακολουθούν κάποια relational data structure και είναι αποθηκευμένα σε distributed computing systems.

Με βάση τα παραπάνω δεδομένα, λοιπόν, δημιουργήθηκαν καινούργιες βάσεις δεδομένων που ανταποκρίνονται στις νέες απαιτήσεις.

**ΘΕΩΡΗΜΑ CAP**

Σύμφωνα με τo θεώρημα CAP (Consistency, Availability, Partition tolerance), γνωστό και ως θεώρημα του Brewer, καθώς διατυπώθηκε για πρώτη φορά από τον Eric Brewer το 2000, είναι αδύνατο για ένα σύστημα αποθήκευσης κατανεμημένων δεδομένων να ικανοποιεί ταυτόχρονα και τις τρεις ιδιότητες που παρουσιάζονται παρακάτω:

* **Consistency**: Η εκτέλεση ενός read στη βάση δεδομένων επιστρέφει πάντα την πιο πρόσφατη τιμή των δεδομένων, ανεξαρτήτως του πότε πραγματοποιήθηκε το τελευταίο write.
* **Availability**: Κάθε request τερματίζει με μια απάντηση χωρίς errors.
* **Partition Tolerance**: Ακόμα και αν η σύνδεση μεταξύ κάποιων κόμβων έχει διακοπεί, οι λειτουργίες στη βάση δεδομένων συνεχίζουν να πραγματοποιούνται.

Επομένως, όπως γίνεται κατανοητό όταν δημιουργείται ένα κατανεμημένο σύστημα, πρέπει να αποφασιστεί ποια από τις τρεις ιδιότητες δεν θα ικανοποιείται. Στη συνέχεια, παρουσιάζονται όλα τα δυνατά ζεύγη ιδιοτήτων που μπορούν να ικανοποιούνται και σε κάθε περίπτωση αποδεικνύεται γιατί πάντα μια ιδιότητα δεν θα μπορεί να συμπεριληφθεί στο σύστημα.

**1Η Περίπτωση: Consistency και Availability**

Όταν ένα σύστημα είναι consistent και available, πρακτικά σημαίνει ότι για κάθε request υπάρχει απάντηση, η οποία είναι και consistent. Από τη στιγμή που η απάντηση είναι consistent, υπάρχει συνεχείς επικοινωνία μεταξύ των servers, προκειμένου να υπάρχει εγγύηση για το αποτέλεσμα. Όταν ικανοποιείται και το availability, η επικοινωνία δεν μπορεί να διακοπεί σε καμιά περίπτωση. Επομένως, στο σύστημα δεν υπάρχει partition tolerance γιατί αν υπήρχε θα έπρεπε το σύστημα είτε να επιστρέφει error, δηλαδή να είναι unavailable, είτε η απάντηση να μην είναι ενημερωμένη, δηλαδή να inconsistent που με βάση τον ορισμό κάτι τέτοιο δεν μπορεί να ισχύει.

**2Η Περίπτωση: Consistency και Partition Tolerance**

Όπως και στη πρώτη περίπτωση, εξαιτίας του consistency οι servers επικοινωνούν και οι βάσεις είναι συγχρονισμένες. Αλλά το σύστημα είναι και partition tolerant, που σημαίνει ότι οι servers μπορούν να σταματήσουν να επικοινωνούν. Όταν όμως αυτή η επικοινωνία διακοπεί, τα δεδομένα δεν θα είναι συγχρονισμένα και κατά συνέπεια δεν θα υπάρχει απάντηση εφόσον δεν θα υπάρχει consistency. Όταν όμως το σύστημα δεν μπορεί να απαντήσει είναι unavailable.

**3Η Περίπτωση: Availability και Partition Tolerance**

Όταν το σύστημα είναι partition tolerant υπάρχει δυνατότητα οι servers να μην επικοινωνούν και οι βάσεις να μην είναι συγχρονισμένες. Αλλά λόγω του availability η απάντηση δεν μπορεί να είναι κάποιο error. Κατά συνέπεια, σε ένα request ο κάθε server θα παρουσιάζει την δική του εκδοχή και φυσικά δεν θα υπάρχει consistency.

**ΤΥΠΟΙ NOSQL ΒΑΣΕΩΝ ΔΕΔΟΜΕΝΩΝ**

Υπάρχουν 4 διαφορετικοί τύποι NOSQL βάσεων δεδομένων, οι οποίοι είναι οι εξής: key value databases, document stores databases, columnar databases, graph databases.

**Key Value Databases**

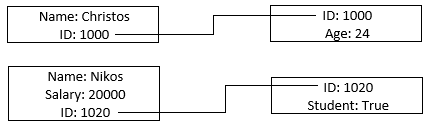
Οι key Value Databases όπως υποδηλώνει και το όνομά τους συνδυάζουν δύο χαρακτηριστικά. Ένα μοναδικό identifier που είναι το key και μια δομή δεδομένων που είναι το value που χαρακτηρίζεται από το key.

Αυτού του είδους οι βάσεις δεδομένων μοιάζουν με hash tables, όπου ο μοναδικός τρόπος να πραγματοποιηθεί ένα query στα δεδομένων είναι με τη βοήθεια του μοναδικού key. Όλα τα keys είναι διατεταγμένα κατά αλφαβητική σειρά και όλα τα objects (key-value ζευγάρια) είναι replicated για μεγαλύτερο availability.

|  |  |
| --- | --- |
| **Key** | **Value** |
| 1 | ID: 1000  Name: Christos  Age: 24 |
| 2 | ID: 1004  Name: Nikos  Age: 27 |
| 3 | ID: 1012  Name: Giannis  Age: 29 |

**Document Stores Databases**

Σε αυτού του είδους τις NOSQL βάσεις δεδομένων τα στοιχεία που αποθηκεύονται είναι σε μορφή εγγράφων. Σε αυτές τις βάσεις αποθηκεύονται είτε unstructured είτε semi-structured έγγραφα. Με τον όρο «έγγραφα» υποδηλώνεται ένα σύνολο key-values, τα οποί σε μεγάλο βαθμό μοιάζουν με τις key-value databases που παρουσιάστηκαν παραπάνω. Κάθε βάση δεδομένων χρησιμοποιεί pointers για τα πεδία, όπως ακριβώς συμβαίνει με την τεχνική του hashing. Επίσης, είναι πολύ σημαντικό το γεγονός ότι οι document stores databases είναι schema free. Στην παρακάτω εικόνα παρουσιάζεται ένα παράδειγμα document store database:



Στη παραπάνω εικόνα παρουσιάζεται τόσο η δομή των εγγράφων και το πως συνδέονται μεταξύ τους, όσο και το γεγονός ότι είναι schema free.

**Columnar Databases**

Οι columnar databases γνωστές και ως column family databases χωρίζονται σε δύο βασικές κατηγορίες, οι οποίες είναι οι εξής:

**Row Oriented Databases**

Σε αυτή τη κατηγορία κάθε row περιλαμβάνει ένα row key και όλες τις τιμές των αντίστοιχων columns όπως παρουσιάζεται παρακάτω:

|  |  |  |
| --- | --- | --- |
| **ID** | **Name** | **Age** |
| 01 | Christos | 24 |
| 02 | Nikos | 27 |
| 03 | Giannis | 29 |

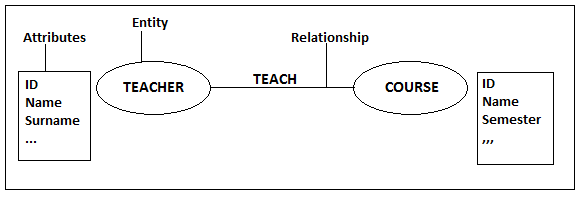
**Column Oriented Databases**

Σε αντίθεση με τις row oriented databases, οι column oriented κάνουν serialize όλες τις τιμές ενός column και στη συνέχεια του επόμενου, όπως παρουσιάζεται στην παρακάτω εικόνα:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | 01: Christos | 02: Nikos | 03: Giannis |
| **Age** | 01: 24 | 02: 27 | 03: 29 |

**Graph Database**

Οι Graph Databases βασίζονται στη θεωρία των γράφων, όπου ένας γράφος αποτελείται από nodes, properties και edges. Στις graph databases τα nodes, properties και edges αντιστοιχούν σε entities, attributes και relationships. Στο παρακάτω σχήμα παρουσιάζεται η μορφή ενός graph database:



**ΠΕΡΙΓΡΑΦΗ ΑΛΓΟΡΙΘΜΩΝ**

Τα ερωτήματα που απαντήθηκαν αφορούν χωροχρονικά δεδομένα και trajectories, δηλαδή τροχιές που ακολουθήθηκαν από ορισμένες οντότητες στο χώρο. Τα βασικά χαρακτηριστικά που έχουν τα δεδομένα που μελετήθηκαν είναι οι συντεταγμένες καθώς και η χρονική στιγμή που συνδέεται με το αντίστοιχο σημείο.

Τα ερωτήματα που απαντήθηκαν είναι τα εξής:

* **k-nearest neighbour:** Στο συγκεκριμένο ερώτημα δεδομένου ενός σημείου και ενός χρονικού διαστήματος αναζητούνται τα k κοντινότερα σημεία που παράλληλα ικανοποιούν και τον χρονικό περιορισμό. Τα ερωτήματα απαντήθηκαν με δύο τρόπους:

1. Ο πρώτος τρόπος περιλαμβάνει έλεγχο μόνο με βάση τον χρόνο και στη συνέχεια εύρεση των k κοντινότερων γειτόνων από τα σημεία που πέρασαν με επιτυχία τον έλεγχο. Πρόκειται για έναν σχετικά μη αποδοτικό τρόπο καθώς δεν χρησιμοποιείται κανένας χωρικός περιορισμός, σε πολλές περιπτώσεις επιστρέφεται μεγάλος αριθμός δεδομένων (υπάρχει περίπτωση όλα τα δεδομένα να ικανοποιούν τον χρονικό περιορισμό) και στη συνέχεια στη μνήμη για όλα αυτά τα δεδομένα πρέπει να πραγματοποιηθεί μέτρηση της απόστασης και σύγκριση μεταξύ τους.
2. Ο δεύτερος τρόπος περιλαμβάνει χρήση στατιστικών στοιχείων. Κατά την εισαγωγή των δεδομένων στη βάση, ο χώρος που βρίσκονται τα δεδομένα χωρίζεται σε κελιά και δημιουργείται ένα αρχείο που δείχνει πόσα δεδομένα βρίσκονται σε κάθε κελί. Αυτό έχει ως αποτέλεσμα όταν το σημείο για το οποίο γίνεται η αναζήτηση των κοντινότερων γειτόνων να βρίσκεται σε ένα κελί με πολλά σημεία να ελέγχονται μόνο τα σημεία που βρίσκονται σε αυτό και στα γειτονικά κελιά και όχι όλα τα σημεία της βάσης δεδομένων. Θεωρητικά, αυτός ο τρόπος είναι περισσότερο αποδοτικός, καθώς δεν εξετάζονται όλα τα δεδομένα.

* **k-nearest neighbour με unique id:** Πρόκειται για το ίδιο ερώτημα με το k-nearest neighbour με τη διαφορά ότι τα k σημεία που θα είναι οι κοντινότεροι γείτονες πρέπει να προέρχονται από διαφορετικές οντότητες, δηλαδή να έχουν διαφορετικό id. Όπως και το παραπάνω ερώτημα, έτσι κι αυτό απαντήθηκε με τους δύο τρόπους που περιγράφονται παραπάνω. Φυσικά, σε σύγκριση με τα αντίστοιχα ερωτήματα του k-nearest neighbour απαιτείται περισσότερος χρόνος, καθώς και ο έλεγχος περισσότερων δεδομένων.
* **Box Range Query:** Σε αυτό το ερώτημα δίνεται ένα ορθογώνιο και ένα χρονικό διάστημα και αναζητούνται όλα τα σημεία που βρέθηκαν μέσα στο ορθογώνιο το συγκεκριμένο χρονικό διάστημα.
* **Circle Range Query:** Όπως στο Box Range Query, αναζητούνται τα σημεία που βρέθηκαν μέσα σε έναν κύκλο ένα συγκεκριμένο χρονικό διάστημα.
* **Aggregate Range Query:** Η διαδικασία εύρεσης των δεδομένων είναι ίδια όπως τα box και circle range query. Η διαφορά είναι ότι στο τέλος εκτελείται κάποια ενέργεια στα δεδομένα που βρέθηκαν όπως για παράδειγμα να επιστρέφεται στον χρήστη το άθροισμα όλων των δεδομένων.

Όλα τα ερωτήματα απαντήθηκαν με 4 διαφορετικούς τρόπους όσο αφορά τη χρήση index:

* Χωρίς κανένα index
* Με index στο χρόνο
* Με index στις συντεταγμένες
* Με index στις συντεταγμένες και στο χρόνο

Επίσης, μια παραλλαγή των ερωτημάτων είναι να μην επιστρέφονται τα id, αλλά για κάθε id να επιστρέφεται όλο το trajectory. Αυτό μπορεί να απαντηθεί με δύο τρόπους: 1) για κάθε id να πραγματοποιείται μια αναζήτηση στο collection και να επιστρέφονται όλα τα documents που έχουν το ίδιο id, 2) να δημιουργηθεί ένα δεύτερο collection, στο οποίο κάθε document θα περιέχει όλο το trajectory. Mε αυτό το τρόπο γίνεται αναζήτηση στο δεύτερο collection ενός document που θα περιέχει το συγκεκριμένο id.

Όλα τα πειράματα πραγματοποιήθηκαν με πολλές διαφορετικές παραμέτρους.

**Σύνδεση στη βάση δεδομένων**

def connect\_to\_mongodb(MONGO\_HOST, MONGO\_PORT, MONGO\_DB, MONGO\_USER, MONGO\_PASSWORD, colllection\_name):

client = MongoClient(MONGO\_HOST, MONGO\_PORT)

db = client[MONGO\_DB]

#db.authenticate(MONGO\_USER, MONGO\_PASSWORD)

print("Connection Succeed")

return db[colllection\_name]

Η παραπάνω συνάρτηση πραγματοποιεί την σύνδεση στη βάση δεδομένων. Τα ορίσματα της συνάρτησης είναι τα host, port, database name, username, password, collection name. H συνάρτηση επιστρέφει το collection της βάσης, έχοντας πραγματοποιήσει τη σύνδεση σε αυτό, έτοιμο για να χρησιμοποιηθεί από τις υπόλοιπες συναρτήσεις, οι οποίες θα παρουσιαστούν παρακάτω. To # μπροστά από το db.authenticate υποδηλώνει ότι η συγκεκριμένη γραμμή έχει μπει σε σχόλιο, καθώς στα παραδείγματα που θα παρουσιαστούν η σύνδεση στη βάση δεν απαιτεί username και password.

Ένα παράδειγμα χρήσης της παραπάνω συνάρτησης είναι το εξής:

mycol = connect\_to\_mongodb('localhost', 27017, 'trajectories', '', '','trajectory')

Στο παραπάνω παράδειγμα το host ισούται με localhost, το port είναι το 27017, το όνομα της βάσης είναι trajectories, η σύνδεση στην βάση δεν απαιτεί username και password και από τη βάση θα χρησιμοποιηθεί το collection objects.

Η παραπάνω γραμμή μπορεί να χρησιμοποιηθεί είτε μέσα σε κάποια άλλη συνάρτηση, είτε το mycol να χρησιμοποιηθεί σαν όρισμα σε κάποια άλλη συνάρτηση.

**Εύρεση k κοντινότερων γειτόνων**

H k\_neighbors συνάρτηση υπολογίζει τους k κοντινότερους γείτονες ενός σημείου (x,y) που ικανοποιούν ένα χρονικό περιορισμό, δηλαδή το t βρίσκεται ανάμεσα σε t\_min και t\_max.

Αρχικά η συνάρτηση λαμβάνει ως ορίσματα τα (mycol, x, y, k, t\_min, t\_max) που αντιστοιχούν στα εξής: το mycol αντιστοιχεί στη σύνδεση σε κάποιο collection της βάσης, όπως παρουσιάστηκε παραπάνω, x και y είναι οι συντεταγμένες για τις οποίες πρέπει να βρεθούν οι κοντινότεροι γείτονες. k είναι ο αριθμός των κοντινότερων γειτόνων και t\_min, t\_max το χρονικό διάστημα μέσα στο οποίο θα πραγματοποιηθεί η αναζήτηση των κοντινότερων γειτόνων.

Στη συνέχεια καλείται η παραπάνω συνάρτηση, η οποία πραγματοποιεί τη σύνδεση με τη βάση δεδομένων, όπως παρουσιάζεται παρακάτω:

def k\_neghbors(mycol,x,y,k,t\_min,t\_max):

Η λογική που ακολουθείται είναι η εξής: Πρώτα βρίσκονται όλες οι εγγραφές που ικανοποιούν τον χρονικό περιορισμό δηλαδή βρίσκονται μεταξύ t\_min και t\_max και στη συνέχεια υπολογίζονται οι k κοντινότερες εγγραφές στο x και y. Ο υπολογισμός των εγγραφών που βρίσκονται μεταξύ t\_min και t\_max γίνεται με τις εξής εντολές:

query='{"$and":[{"t":{"$gt":'+str(t\_min)+'}}, {"t":{"$lt":'+str(t\_max)+'}}]}'

k1 = json.loads(query)

cursor = mycol.find(k1)

Στη συνέχεια τα αποτελέσματα αποθηκεύονται στη λίστα documents

documents = []

counter = 0

for document in cursor:

documents.insert(counter, document)

counter += 1

Το επόμενο βήμα είναι για κάθε εγγραφή του documents να υπολογιστεί η Ευκλείδια απόσταση από τις παραμέτρους (x, y). Για να πραγματοποιηθεί η παραπάνω διαδικασία, οι παράμετροι (x, y) αποθηκεύονται σε έναν πίνακα b1 και τα στοιχεία (lon, lat) των εγγραφών του document σε έναν πίνακα b2. Με χρήση της συνάρτησης linalg.norm(b1-b2) του πακέτου numpy της Python υπολογίζεται η ευκλείδια απόσταση και αποθηκεύεται σε μια λίστα distances μαζί με τον αύξων αριθμό της εγγραφής του document.

b1 = np.array((x,y))

distances = []

counter = 0

for doc in documents:

b2 = np.array((doc['coordinates'][0],doc['coordinates'][1]))

dist = np.linalg.norm(b1-b2)

distances.append((counter,dist))

counter += 1

Τέλος, κάνοντας μια ταξινόμηση στη λίστα distances με τις αποστάσεις σε αύξουσα σειρά, μπορεί να γίνει η επιλογή των k πρώτων εγγραφών και με χρήση του αύξων αριθμού πραγματοποιείται εύρεση των εγγραφών στη λίστα documents.

Στην Python, αν το k είναι μεγαλύτερο από το πλήθος των εγγραφών στο distances, θα υπάρξει error. Επομένως, γίνεται και ο έλεγχος των δύο τιμών και αν το k είναι μεγαλύτερο, τότε επιστρέφονται όλες οι εγγραφές:

distances = sorted(distances, key=lambda distance: distance[1])

document = []

if len(distances) > k:

for j in range(k):

document.append(documents[distances[j][0]])

else:

for j in range(len(distances)):

document.append(documents[distances[j][0]])

return document

**Εύρεση k κοντινότερων γειτόνων με μοναδικό id**

Μια παραλλαγή της συνάρτησης που παρουσιάστηκε παραπάνω είναι να βρεθούν οι k κοντινότεροι γείτονες που θα έχουν διαφορετικό id. Πρακτικά αυτό σημαίνει ότι αν για ένα σημείο (x, y) βρεθούν τα 3 κοντινότερα σημεία, αυτά τα σημεία θα πρέπει να μην έχουν τα ίδια id ή διαφορετικά να μην είναι σημεία του ίδιου trajectory.

O αλγόριθμος είναι ίδιος με την παραπάνω συνάρτηση μέχρι το σημείο που γίνεται η ταξινόμηση στη λίστα distances με τη διαφορά ότι στη λίστα distances, αποθηκεύεται και το id της κάθε εγγραφής, εκτός από την απόσταση από το (x, y) και τον αύξων αριθμό.

Στη συνέχεια, έχοντας ταξινομημένη τη λίστα, η λογική που ακολουθείται είναι να βρεθούν τα πρώτα k στοιχεία με διαφορετικό id.

Επομένως για κάθε στοιχείο της λίστας distances ελέγχεται η τελική λίστα document. Αν η λίστα δεν είναι άδεια, τότε συγκρίνονται τα δύο id, αν είναι διαφορετικά τότε το στοιχείο που αντιστοιχεί στη συγκεκριμένη εγγραφή του distances εισάγεται στην τελική λίστα. Η διαδικασία αυτή επαναλαμβάνεται μέχρι να γίνει εισαγωγή των k στοιχείων.

for j in range(len(distances)):

if len(document) < k:

if document != []:

for doc in document:

traj\_object = documents[distances[j][0]]

if traj\_object['id'] != doc['id']:

document.append(documents[distances[j][0]])

else:

document.append(documents[distances[j][0]])

else:

break

return document

**Box Range Query**

Η Box Range συνάρτηση υπολογίζει πόσες εγγραφές υπήρξαν μέσα σε ένα ορθογώνιο ένα συγκεκριμένο χρονικό διάστημα.

Τα ορίσματα της συνάρτησης είναι (x\_min, y\_min) που αντιστοιχούν στην κάτω αριστερά γωνία του ορθογωνίου, (x\_max, y\_max) που αντιστοιχούν στην πάνω δεξιά γωνία του ορθογωνίου και (t\_min, t\_max) που είναι το χρονικό διάστημα από t\_min ως t\_max.

Επομένως, το query ζητάει όλες τις εγγραφές όπου το lon είναι ανάμεσα σε x\_min και x\_max, το lat ανάμεσα σε y\_min και y\_max και παράλληλα ικανοποιείται και ο χρονικός περιορισμός.

Η mongo περιλαμβάνει τον geospatial query operator $geowithin, ο οποίος μπορεί να ακολουθείται από κάποιους άλλους τελεστές όπως $center για κύκλο ή $box για ορθογώνιο και χρησιμοποιείται για να βρίσκονται συντεταγμένες που περιέχονται μέσα σε μια γεωγραφική περιοχή.

def range\_box(mycol, x\_min, x\_max, y\_min, y\_max, t\_min, t\_max):

query = '{"$and":[{"coordinates":{"$geoWithin": {"$box":[['+str(x\_min)+','+str(y\_min)+'] , ['+str(x\_max)+','+str(y\_max)+']]}}},{"t":{"$gt":'+str(t\_min)+'}},{"t":{"$lt":'+str(t\_max)+'}}]}'

k1 = json.loads(query)

cursor = mycol.find(k1)

documents = []

counter = 0

for document in cursor:

documents.insert(counter, document)

counter += 1

return documents

**Box Range query aggregate**

Ο παραπάνω αλγόριθμος μπορεί να εκτελεστεί και ως εξής: Εκτελούνται όλα τα βήματα μέχρι να βρεθούν όλες οι εγγραφές που ικανοποιούν τους περιορισμούς, με τη διαφορά ότι η συνάρτηση περιέχει μια ακόμα παράμετρο. Στο τέλος, η συνάρτηση δεν επιστρέφει όλες τις εγγραφές, αλλά ότι ορίζει η παράμετρος agg. Παρακάτω παρουσιάζεται η περίπτωση όπου το agg είναι ίσο με count και κατά συνέπεια η συνάρτηση επιστρέφει το πλήθος των εγγραφών.

def range\_box(mycol, x\_min, x\_max, y\_min, y\_max, t\_min, t\_max):

query = '{"$and":[{"coordinates":{"$geoWithin": {"$box":[['+str(x\_min)+','+str(y\_min)+'] , ['+str(x\_max)+',' +str(y\_max)+']]}}},{"t":{"$gt":'+str(t\_min)+'}},{"t":{"$lt":'+str(t\_max)+'}}]}'

k1 = json.loads(query)

cursor = mycol.find(k1)

documents = []

counter = 0

for document in cursor:

documents.insert(counter, document)

counter += 1

if agg == "count":

return len(documents)

**Circle Range Query**

Η Cirlcle Range συνάρτηση υπολογίζει πόσες εγγραφές υπήρξαν μέσα σε έναν κύκλο μια συγκεκριμένη χρονική στιγμή.

Ομοίως με το box range query, εκτελείται πρώτα το query που βρίσκει τις εγγραφές που βρίσκονται μέσα στον κύκλο και ικανοποιούν τον χρονικό περιορισμό.

def range\_circle(mycol, x, y, r, t\_min, t\_max):

query='{"$and":[{"coordinates":{"$geoWithin":{"$center": [['+str(x)+','+str(y)+'],'+str(r)+']}}} ,{"t":{"$gt":'+str(t\_min)+'}},{"t":{"$lt":'+str(t\_max)+'}}]}'

k1 = json.loads(query)

cursor = mycol.find(k1)

documents = []

counter = 0

for document in cursor:

documents.insert(counter, document)

counter += 1

return documents

**Εισαγωγή Δεδομένων στη Βάση και Αποθήκευση Στατιστικών Στοιχείων**

Τα δεδομένα που αποθηκεύτηκαν στη βάση δεδομένων και χρησιμοποιήθηκαν στα πειράματα που θα παρουσιαστούν παρακάτω, προέρχονται από το Zenodo Repository. Συγκεκριμένα, οι εγγραφές είναι 4565 και αφορούν τροχιές πλοίων. Κάθε εγγραφή αποτελείται από τα εξής στοιχεία: sourcemmsi, altitude, speedoverground, courseoverground, lon, lat, ts. Τα στοιχεία που αποθηκεύτηκαν στη βάση δεδομένων είναι το sourcemmsi ως id, τα lon και lat που αφορούν τις συντεταγμένες και το ts που είναι το timestamp για κάθε εγγραφή.

H λογική που ακολουθήθηκε είναι η εξής: χωρίζοντας τον χώρο που βρίσκονται τα δεδομένα σε κελιά και αποθηκεύοντας τον αριθμό των εγγραφών που βρίσκονται σε κάθε κελί, κάποια ερωτήματα μπορούν να απαντηθούν με προσπέλαση λιγότερων εγγραφών από τη βάση δεδομένων (των εγγραφών που βρίσκονται στο συγκεκριμένο κελί ή και των γειτονικών κελιών).

Αρχικά ο χρήστης μπορεί να εισάγει των αριθμό των κελιών στον άξονα x και στον άξονα y με τις εντολές:

xcells = int(input("number of xcells: "))

ycells = int(input("number of ycells: "))

και να διαβάσει το csv αρχείο με την εντολή read\_csv του πακέτου pandas.

df = pd.read\_csv('Zenodo\_Datasets/AISData/nari\_dynamic\_sar.csv')

Στη συνέχεια γίνεται εύρεση των μικρότερων και μεγαλύτερων lon και lat και αποθήκευση στις κατάλληλες μεταβλητές.

min\_lon = min(df['lon'])

max\_lon = max(df['lon'])

min\_lat = min(df['lat'])

max\_lat = max(df['lat'])

Πλέον, γνωρίζοντας τo ορθογώνιο μέσα στο οποίο βρίσκονται όλες οι συντεταγμένες, καθώς και τον αριθμό των κελιών x και y, μπορεί να γίνει ο έλεγχος για κάθε εγγραφή έτσι ώστε να βρεθεί το κελί μέσα στο οποίο βρίσκεται.

Ο έλεγχος που πραγματοποιείται για κάθε εγγραφή είναι αν το lon είναι μεγαλύτερο από το min\_lon+x\*(max\_lon-min\_lon)/xcells που ουσιαστικά είναι η αριστερή πλευρά του ορθογωνίου με την αρίθμηση του x να ξεκινάει από το 0 και μικρότερο από το min\_lon+(x+1)\*(max\_lon-min\_lon)/xcells που είναι η δεξιά πλευρά. Ομοίως για το lat κάθε εγγραφής πρέπει να είναι μεγαλύτερο από το min\_lat+y\*(max\_lat-min\_lat)/ycells και μικρότερο από το min\_lat+(y+1)\*(max\_lat-min\_lat)/ycells.

Εφόσον βρεθεί το ορθογώνιο στο οποίο ανήκει η εγγραφή, εισάγεται στη βάση δεδομένων με τις εντολές:

mydict = { "id": str(df['sourcemmsi'][i]), "t": int(df['ts'][i]), "coordinates":[df['lon'][i], df['lat'][i]] }

mycol.insert\_one(mydict)

Τα min και max lon και lat του ορθογωνίου, τα x και y που αναφέρονται στο κελί καθώς και ένας μοναδικός αύξων αριθμός για κάθε κελί αποθηκεύονται σε μια λίστα.

coords\_list.append([y\*xcells+x, lon\_start, lon\_end, lat\_start, lat\_end, x, y])

Όταν ολοκληρωθεί η διαδικασία και έχει πραγματοποιηθεί η εισαγωγή όλων των εγγραφών στη βάση δεδομένων, για κάθε κελί βρίσκεται το άθροισμα όλων των μοναδικών αριθμών που αναφέρονται σε αυτό και έχουν εισαχθεί στη λίστα.

Τέλος, σε ένα αρχείο statistics.txt για κάθε κελί εισάγονται τα εξής στοιχεία: άθροισμα των εγγραφών, min longitude, max longitude, min latitude, max latitude, x, y.

**Εισαγωγή Δεδομένων σε Δεύτερο Collection**

Όπως θα παρουσιαστεί παρακάτω, σε ορισμένες περιπτώσεις υπάρχει η ανάγκη να επιστραφεί ολόκληρο το trajectory και όχι μόνο μια εγγραφή αυτού. Συνεπώς, είναι περισσότερο αποδοτικό να δημιουργηθεί ένα δεύτερο collection, στο οποίο κάθε document θα περιέχει ολόκληρο το trajectory δηλαδή τα δεδομένα θα είναι της μορφής id και ένας πίνακας με όλες τις τριάδες (lon, lat, t). Αυτό το collection θα λειτουργήσει σαν index. Για την εισαγωγή κάθε δεδομένου στο δεύτερο collection καλείται η συνάρτηση:

def insert\_into\_col(col\_idx, traj\_id, t, lon, lat):

η οποία λαμβάνει ως ορίσματα το collection στο οποίο θα γίνει η σύνδεση καθώς και τα δεδομένα της εγγραφής (id, timestamp και συντεταγμένες).

Στη συνέχεια, πραγματοποιείται αναζήτηση σε αυτό το collection για τυχόν ύπαρξη του του traj\_id και αν υπάρχει, η συγκεκριμένη εγγραφή αποθηκεύεται σε μια λίστα:

query = '{"id":"'+traj\_id+'"}'

k1 = json.loads(query)

cursor = col\_idx.find(k1)

documents = []

counter = 0

for document in cursor:

documents.insert(counter, document)

Αν η λίστα είναι άδεια, δηλαδή δεν υπάρχει το συγκεκριμένο id στο collection, τότε τα ορίσματα της συνάρτησης εισάγονται σαν ένα document στο collection.

if len(documents) == 0:

col\_dict = { "id": traj\_id, "trajectories":[[t, lon, lat]] }

col\_idx.insert(col\_dict)

Διαφορετικά το document που περιέχει το traj\_id γίνεται update και στον πίνακα trajectories προστίθεται η εγγραφή [t, lon, lat]

else:

trajectories = [t, lon, lat]

old\_query = {"id":traj\_id}

push = {"$push":{"trajectories":trajectories}}

col\_idx.update\_one(old\_query, push)

**Εύρεση k κοντινότερων γειτόνων με χρήση στατιστικών στοιχείων**

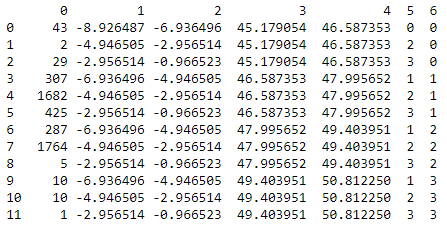
Για την εύρεση των k κοντινότερων γειτόνων θα χρησιμοποιηθεί το .txt αρχείο, στο οποίο έχει αποθηκευτεί ο αριθμός των εγγραφών που βρίσκονται σε κάθε κελί. Με αυτό τον τρόπο η αναζήτηση επικεντρώνεται σε ένα κελί ή και σε κάποια γειτονικά κελιά και ο αριθμός των εγγραφών που θα εξεταστούν είναι μικρότερος από το σύνολο όλων των εγγραφών.

Οι παράμετροι της συνάρτησης είναι το mycol με το οποίο έχει πραγματοποιηθεί η σύνδεση στη βάση δεδομένων, τα x και y που είναι οι συντεταγμένες στις οποίες θα αναζητηθούν οι κοντινότεροι γείτονες, το k που είναι ο αριθμός των κοντινότερων γειτόνων και τα tmin και tmax που είναι τα χρονικό διάστημα μέσα στα οποία θα πρέπει να βρίσκονται οι εγγραφές.

Αρχικά, διαβάζεται το αρχείο στο οποίο βρίσκονται τα δεδομένα και αποθηκεύονται σε ένα dataframe. Αυτό επιτυγχάνεται με την συνάρτηση read\_csv του πακέτου pandas.

df = pd.read\_csv('statistics.txt', header = None)

Τα δεδομένα που αποθηκεύτηκαν στο df έχουν την εξής μορφή:



Η στήλη 0 παρουσιάζει τον αριθμό των εγγραφών που βρίσκονται σε κάθε κελί, οι στήλες 1 και 2 αναφέρονται στα min και max longitude, οι στήλες 3 και 4 στα min και max latitude και οι στήλες 5 και 6 στον αύξων αριθμό του κελιού στον οριζόντιο και κάθετο άξονα αντίστοιχα.

Στη συνέχεια από το df επιλέγεται η εγγραφή για την οποία η παράμετρος x της συνάρτησης είναι μεγαλύτερη από την τιμή της στήλης 1 και μικρότερη από την τιμή της στήλης 2 και αντίστοιχα η παράμετρος y που είναι μεγαλύτερη από την τιμή της στήλης 3 και μικρότερη από την τιμή της στήλης 4. Οι τιμές των στηλών 5 και 6 της εγγραφής που επιλέχθηκε αποθηκεύονται στις μεταβλητές xcell και ycell.

df1 = df.loc[(df[1]<=x) & (df[2]>=x) & (df[3]<=y) & (df[4]>=y)]

has\_docs = True

xcell = int(df1[5])

ycell = int(df1[6])

Για τη συγκεκριμένη εγγραφή αν η τιμή της στήλης 0 είναι μεγαλύτερη από το k, δηλαδή στο συγκεκριμένο κελί βρίσκονται περισσότερες από k εγγραφές, τότε υπολογίζεται ποια γωνία του ορθογωνίου έχει τη μεγαλύτερη απόσταση από το (x, y).

if sum(df1[0]) >= k:

dist = np.array((x,y))

dist13= np.array((df1[1][df1.index[0]],df1[3][df1.index[0]]))

dist24= np.array((df1[2][df1.index[0]],df1[4][df1.index[0]]))

dist14= np.array((df1[1][df1.index[0]],df1[4][df1.index[0]]))

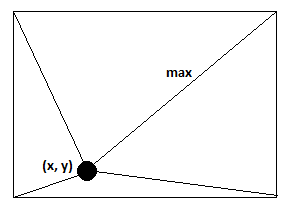
dist23= np.array((df1[2][df1.index[0]],df1[3][df1.index[0]]))

max\_distance = max(np.linalg.norm(dist-dist13),\

np.linalg.norm(dist-dist24),\

np.linalg.norm(dist-dist14),\

np.linalg.norm(dist-dist23))



Μετά την εύρεση της γωνίας με τη μεγαλύτερη απόσταση εκτελείται η range\_circle που παρουσιάστηκε παραπάνω. Η συγκεκριμένη συνάρτηση επιστρέφει όλες τις εγγραφές που βρίσκονται μέσα σε έναν κύκλο με κέντρο το (x, y) και ακτίνα ίση με την μεγαλύτερη απόσταση της γωνίας του ορθογωνίου.

documents = range\_circle(mycol, x, y, max\_distance, tmin, tmax)

Τέλος, για όλες τις εγγραφές που επέστρεψε η range\_circle υπολογίζεται η απόσταση από το (x, y),

τα αποτελέσματα ταξινομούνται από το μικρότερο στο μεγαλύτερο

b1 = np.array((x,y))

distances = []

counter = 0

for doc in documents:

b2 = np.array((doc['coordinates'][0],doc['coordinates'][1]))

dist = np.linalg.norm(b1-b2)

distances.append((counter,dist))

counter += 1

distances = sorted(distances, key=lambda distance: distance[1])

και αν τα αποτελέσματα είναι περισσότερα ή ίσα με k επιλέγονται τα πρώτα k αποτελέσματα.

document = []

if len(distances) >= k:

for j in range(k):

document.append(documents[distances[j][0]])

has\_docs = False

return document

Στην περίπτωση που ο αριθμός των εγγραφών που βρίσκονται στο συγκεκριμένο κελί είναι μικρότερος από k ή το query που εκτελέστηκε για το συγκεκριμένο κελί επέστρεψε λιγότερα από k αποτελέσματα, επιλέγονται όλα τα γειτονικά κελιά και η διαδικασία επαναλαμβάνεται για το ορθογώνιο που θα σχηματιστεί από όλα τα κελιά. Η διαδικασία επαναλαμβάνεται μέχρι ο αριθμός των εγγραφών που βρίσκονται στο ορθογώνιο που σχηματίζεται να είναι μεγαλύτερος από k. Στο παρακάτω παράδειγμα το i παίρνει τιμές από 1 ως 4, καθώς όταν έγινε ο διαχωρισμός του χώρου που βρίσκονται οι συντεταγμένες σε κελιά, θεωρήσαμε ότι ο μεγαλύτερος αριθμός κελιών στον x και στον y άξονα είναι 4.

if sum(df1[0])<k or has\_docs==True:

for i in range(1, 4):

df2 = df.loc[(df[5]>=xcell-i) & (df[5]<=xcell+i) & (df[6]>=ycell-i) & (df[6]<=ycell+i)]

if sum(df2[0]) >= k:

.

.

.

break

**Εύρεση k κοντινότερων γειτόνων με διαφορετικά id με χρήση στατιστικών στοιχείων**

Στη περίπτωση που αναζητούνται k κοντινότεροι γείτονες με διαφορετικά id, η λογική που ακολουθείται είναι ίδια με το παραπάνω ερώτημα, η διαφορά παρουσιάζεται μετά την ταξινόμηση των στοιχείων που επέστρεψε το query.

Όπως παρουσιάζεται στο παρακάτω τμήμα κώδικα δημιουργούνται δύο λίστες, η document στην οποία θα γίνει εισαγωγή των k κοντινότερων εγγραφών και η docs\_id στην οποία θα γίνει εισαγωγή μόνο των ids των k κοντινότερων εγγραφών. Για κάθε στοιχείο του distances, αν η λίστα documents περιέχει λιγότερα από k στοιχεία τότε εξετάζονται δύο περιπτώσεις. Αν η documents περιέχει στοιχεία, συγκρίνονται τα αντίστοιχα ids του docs\_id με το id του distances και αν δεν υπάρχει στο docs\_id τότε πραγματοποιείται η εισαγωγή στο document. Διαφορετικά αν η document είναι άδεια, τότε εισάγεται το στοιχείο toy distances και το αντίστοιχο id στο docs\_id.

distances = sorted(distances, key=lambda distance: distance[1])

document = []

docs\_id = []

for j in range(len(distances)):

if len(document) < k:

if document != []:

traj\_object = documents[distances[j][0]]

if traj\_object['id'] not in docs\_id:

docs\_id.append(traj\_object['id'])

document.append(documents[distances[j][0]])

else:

traj\_object = documents[distances[j][0]]

docs\_id.append(traj\_object['id'])

document.append(traj\_object)

else:

break

if len(document) == k:

return document

**Εξαγωγή όλου του trajectory**

Μια παραλλαγή των ερωτημάτων που παρουσιάστηκαν στην προηγούμενη ενότητα είναι στο τέλος για κάθε id που βρίσκεται, επιστρέφεται ολόκληρο το trajectory, δηλαδή όλες οι εγγραφές που ανήκουν στο συγκεκριμένο id. Προκειμένου να επιστραφεί όλο τo trajectory τα παραπάνω ερωτήματα που επιστρέφουν μέρος αυτών δεν αλλάζουν καθόλου. Η διαφορά είναι στο γεγονός ότι υπάρχει ένα επιπλέον βήμα στο τέλος της διαδικασίας και αυτό είναι η χρήση των unique ids για την εύρεση ολόκληρου του trajectory.

Η εύρεση ολόκληρου του trajectory μπορεί να πραγματοποιηθεί με δύο τρόπους. Ο πρώτος τρόπος είναι για κάθε id που επέστρεψε κάποιο από τα παραπάνω ερωτήματα να αναζητηθούν όλα τα ίδια id στο collection. Πρόκειται για έναν σχετικά μη αποδοτικό τρόπο, καθώς απαιτεί τον έλεγχο όλων των δεδομένων. Φυσικά, η διαδικασία θα μπορούσε να επιταχυνθεί με τη χρήση ενός index στο id. O δεύτερος τρόπος απαιτεί τη δημιουργία ενός δεύτερου collection στο οποίο κάθε document περιλαμβάνει τα εξής δεδομένα id, [(lon, lat, t), (lon, lat, t)…], δηλαδή ένα id και στη συνέχεια έναν πίνακα με όλες τις τριάδες (lon, lat, t). Με αυτό το τρόπο τα documents που θα εξεταστούν είναι λιγότερα.

Και οι δύο τρόποι βασίζονται στον ίδιο κώδικα, η διαφορά τους είναι στο όρισμα που θα εισαχθεί στην τελευταία συνάρτηση, δηλαδή σε ποιο από τα δύο collections θα γίνει η σύνδεση.

Όπως παρουσιάζεται στο παρακάτω τμήμα κώδικα, τα δεδομένα που επιστρέφονται από τη συνάρτηση range\_box() αποθηκεύονται στο results. Στη συνέχεια, όλα τα id αποθηκεύονται στη λίστα ids. To ids μετατρέπεται σε dataframe και πλέον στο ids αποθηκεύονται όλα τα unique ids, δηλαδή κάθε id εμφανίζεται μόνο μια φορά. Για κάθε id της λίστας ids αποθηκεύονται οι τιμές “id”, η τιμή του id και “:” στις λίστες key\_list, value\_list, operator\_list αντίστοιχα. Αυτές οι τρεις λίστες εισάγονται ως ορίσματα στη συνάρτηση filter\_or().

results = range\_box(mycol, -3.4032917,46.61752,0,48.9,1443687610,1943689790)

ids = []

for r in results:

ids.append(r['id'])

ids = pd.DataFrame(ids)

ids = pd.unique(ids[0])

key\_list = []

value\_list = []

operator\_list = []

for i in ids:

key\_list.append("id")

value\_list.append(i)

operator\_list.append(":")

results = filter\_or(mycol1, key\_list, value\_list, operator\_list)

Η συνάρτηση filter\_or() αναζητά στη βάση όλα τα keys του key\_list που συνδέονται με τις τιμές του value\_list μέσω της αντίστοιχης τιμής του operator\_list. Για παράδειγμα αν οι τιμές των key\_list, value\_list, operator\_list είναι [“name”, “age”], [“John”, 27], [“:”, “>”] αντίστοιχα, τότε στο collection αναζητούνται όλα ta document που είτε το πεδίο name ισούται John είτε η τιμή του πεδίου age είναι μεγαλύτερη του 27. Επομένως, στη περίπτωση με τα trajectories που όλες οι τιμές του key\_list είναι “id” και όλες οι τιμές του operator\_list είναι όλες “:”, αναζητούνται όλα τα id με κάποια από τις τιμές του value\_list. Στη συνέχεια παρουσιάζεται ο κώδικας της συνάρτησης filter\_or:

def filter\_or(col,a,b,c):

query = ''

for i in range(len(a)):

a[i] = '"'+a[i]+'"'

if type(b[i]) == str:

b[i] = '"'+b[i]+'"'

if i == 0:

if c[i] == ':':

query += '{'+a[i]+c[i]+str(b[i])+'}'

else:

if c[i] == ':':

query += ',{'+a[i]+c[i]+str(b[i])+'}'

Για κάθε εγγραφή του a, εφόσον γίνουν κάποιες αλλαγές στις τιμές του key και του value αν είναι απαραίτητες στη μεταβλητή query προστίθενται οι αντίστοιχες τιμές που περιέχουν οι τρεις λίστες. Όταν το i δεν ισούται με 0 τότε προστίθεται ένα κόμμα (,) πριν το query προκειμένου η δομή του να είναι σωστή. Φυσικά, στο filter\_or υπάρχουν και οι περιπτώσεις που c[i] ισούται με (>=, >, <=, <) αλλά δεν έχουν συμπεριληφθεί στην παραπάνω περιγραφή καθώς απαιτείται μόνο η περίπτωση της ισότητας.

query = '{"$or":['+query+"]}"

k1 = json.loads(query)

cursor = col.find(k1)

documents = []

counter = 0

for document in cursor:

documents.insert(counter, document)

counter += 1

return documents

Τέλος, στο query προστίθεται το or, πραγματοποιείται η αναζήτηση και τα αποτελέσματα αποθηκεύονται σε μια λίστα.

Αν η σύνδεση έχει πραγματοποιηθεί στο αρχικό collection, τότε στο τέλος εκτελείται και το παρακάτω τμήμα κώδικα, που ουσιαστικά κάνει μια ομαδοποίηση των δεδομένων ανάλογα με το id στο οποίο ανήκουν.

results = pd.DataFrame(results)

results = results.groupby('id')

for traj\_id, data in results:

print(traj\_id)

print(data)

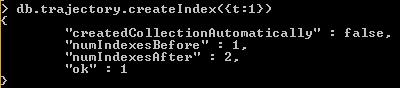
**ΠΕΙΡΑΜΑΤΙΚΗ ΜΕΛΕΤΗ**

Στα πειράματα που πραγματοποιήθηκαν, η μελέτη επικεντρώθηκε στον αριθμό των εγγραφών της βάσης δεδομένων, που πρέπει να προσπελάσει ο κάθε αλγόριθμος προκειμένου να επιστρέψει το τελικό αποτέλεσμα, καθώς και στον χρόνο εκτέλεσης των queries. Προκειμένου να βρεθεί ο αριθμός των εγγραφών που εξετάζει ο κάθε αλγόριθμος χρησιμοποιήθηκε η συνάρτηση explain. Μετά την εκτέλεση της συνάρτησης find, εκτελέστηκε η εντολή print(cursor.explain()['executionStats']). Η συνάρτηση explain μπορεί να πάρει μία από τις 3 παρακάτω παραμέτρους:

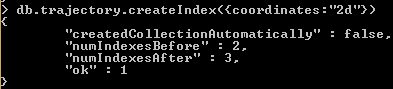
* **queryPlanner:** Η MongoDB εκτελεί ένα query optimizer προκειμένου να αποφασίσει ποιο πλάνο είναι το καλύτερο και επιστρέφει πληροφορίες σχετικά με αυτό.
* **executionStats:** Το καλύτερο πλάνο εκτελείται και επιστρέφονται στατιστικά δεδομένα που περιγράφουν την εκτέλεση του.
* **allPlansExecution:** Το καλύτερο πλάνο εκτελείται και επιστρέφονται στατιστικά δεδομένα για αυτό, όπως επίσης στατιστικά δεδομένα επιστρέφονται και για όλα τα υποψήφια πλάνα.

Όπως είναι προφανές η παράμετρος για την οποία υπάρχει ενδιαφέρον είναι η «executionStats» και συγκεκριμένα τα πεδία που ονομάζεται «totalDocsExamined» και «executionTimeMillis». Η τιμή ατου πεδίου «totalDocsExamined» δείχνει πόσες εγγραφές προσπέλασε ο αλγόριθμος προκειμένου να βρεθεί το σωστό αποτέλεσμα. Το «executionTimeMillis» μετράει το χρόνο εκτέλεσης του query σε milliseconds.

Στα πειράματα πραγματοποιήθηκαν χωρίς τη χρήση index, καθώς επίσης και με τη χρήση index είτε στο πεδίο «t», δηλαδή στο χρόνο είτε στις συντεταγμένες ή και με τη χρήση και των δύο index. Οι εντολές για τη χρήση των index είναι οι εξής:



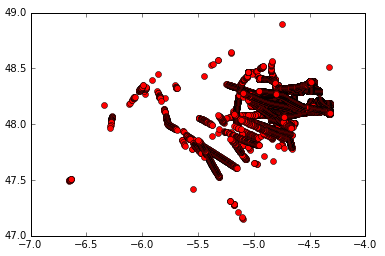
Η τιμή «1» στο t ότι οι τιμές του πεδίου θα ταξινομηθούν σε αύξουσα σειρά, αν η τιμή ήταν «-1», τότε η σειρά θα ήταν φθίνουσα.



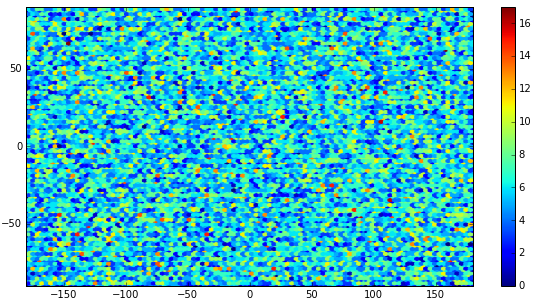
Η τιμή «2d» δείχνει ότι το συγκεκριμένο πεδίο περιέχει ζεύγη τιμών όπως είναι οι συντεταγμένες. Φυσικά, επειδή το συγκεκριμένο index έχει σχεδιαστεί για συντεταγμένες, η πρώτη τιμή, το longitude, πρέπει να βρίσκεται στο διάστημα [-180,180] και το latitude στο [-90, 90]. Σε διαφορετική περίπτωση θα υπάρξει error.

**Datasets**

Το πρώτο dataset προέρχεται από ένα μέρος ενός dataset του zenodo repository. Ουσιαστικά, πρόκειται για 67484 εγγραφές που ανήκουν σε 70 οντότητες, η συγκέντρωση των οποίων βρίσκεται μέσα στο ορθογώνιο που προκύπτει από τις συντεταγμένες (-7, 47) και (-4, 49) όπως παρουσιάζεται και στο παρακάτω σχήμα και το χρονικό διάστημα μέσα στο οποίο βρίσκονται οι εγγραφές είναι το (1443650402, 1443722123).



To δεύτερο dataset δημιουργήθηκε με τυχαίο τρόπο χρησιμοποιώντας την random.uniform() συνάρτηση της Python. Η συγκεκριμένη συνάρτηση χρησιμοποιήθηκε με παραμέτρους (-180,180), (-90, 90) και (1400000000, 16000000000), οι οποίες αντιστοιχούν στο εύρος των τιμών των μεταβλητών lon, lat και time. Στη βάση δεδομένων αποθηκεύτηκαν 20 οντότητες, δηλαδή 20 διαφορετικά id με 5000 εγγραφές το καθένα, δηλαδή σύνολο 70000 εγγραφές. Στο παρακάτω γράφημα παρουσιάζεται η κατανομή των δεδομένων.



**Πειραματική Μελέτη με χρήση του πρώτου dataset**

**Εύρεση k κοντινότερων γειτόνων**

Ο πρώτος αλγόριθμος που μελετήθηκε ήταν ο k\_neighbors χωρίς την χρήση στατιστικών δεδομένων καθώς επίσης και χωρίς την χρήση indexes. Όπως ήταν αναμενόμενο, ο συγκεκριμένος αλγόριθμος προκειμένου να επιστρέψει το τελικό αποτέλεσμα θα έπρεπε να προσπελάσει όλες τις εγγραφές. Οι παράμετροι του αλγόριθμου ήταν οι εξής: (mycol, -4.75, 48.25, 2, 1443690402, 1443700123), δηλαδή αναζητήθηκαν οι 2 κοντινότεροι γείτονες στο σημείο (-4.75, 48.25) που βρίσκονται στο χρονικό διάστημα [-4.75, 48.25]. Το σημείο δεν επιλέχθηκε τυχαία, καθώς όπως παρατηρείται στο παραπάνω γράφημα, υπάρχουν πάρα πολλά σημεία που βρίσκονται κοντά σε αυτό. Βέβαια, θεωρητικά θα υπάρξει βελτίωση όταν χρησιμοποιηθούν στατιστικά δεδομένα σε συνδυασμό με κάποιο index στο χώρο, καθώς όπως παρουσιάστηκε παραπάνω, ο συγκεκριμένος αλγόριθμος επιστρέφει μόνο τα δεδομένα που ικανοποιούν τον χρονικό περιορισμό. Παρακάτω παρουσιάζεται ένα μέρος του αποτελέσματος όπως αυτό παρουσιάστηκε από την συνάρτηση explain.



Ο αλγόριθμος εξέτασε 67484 εγγραφές και επέστρεψε μόλις 13389 σε 38 ms και η διάρκεια όλης της διαδικασίας ήταν 6.44 sec.



Χωρίς την χρήση indexes και ο δεύτερος αλγόριθμος k\_neighbors\_with\_statistics, που χρησιμοποιεί στατιστικά δεδομένα επέστρεψε το ίδιο αποτέλεσμα, δηλαδή χρειάστηκε να ερευνήσει όλα τα έγγραφα προκειμένου να βρει το σωστό αποτέλεσμα. Για τον k\_neighbors\_with\_statistics, ο χώρος χωρίστηκε σε 4x4 ίσα κελιά



Όπως και ο πρώτος αλγόριθμος, έτσι και ο k\_neighbors\_with\_statistics επιστρέφει 13389 αποτελέσματα σε σχεδόν ίδιο χρόνο, 41 ms και η συνολική διάρκεια ήταν 6.21 sec.



Χωρίς τη χρήση κάποιου index στο χώρο ή στο χρόνο, θα εξετάζονται πάντα όλα τα έγγραφα από τον k\_neighbors\_with\_statistics.

Στην περίπτωση που δημιουργηθεί ένα index στο πεδίο t, δηλαδή στο χρόνο, ο πρώτος αλγόριθμος εξετάζει 13389 εγγραφές, αυτές που επιστρέφει και ο χρόνος εκτέλεσης του query είναι τα 15ms και η διάρκεια ήταν 6.14 sec.

Ομοίως και ο k\_neighbors\_with\_statistics εξετάζει 13389 εγγραφές και τις επιστρέφει όλες, αλλά αυτό γίνεται σε 21ms. Η διαφορά είναι πολύ πιθανό να οφείλεται στο γεγονός ότι ο δεύτερος αλγόριθμος εξετάζει και τους χωρικούς περιορισμούς, καθώς όπως παρουσιάστηκε παραπάνω, δημιουργείται ένας κύκλος και ελέγχεται ποια δεδομένα βρίσκονται μέσα σε αυτόν. Παρόλα αυτά, ο αλγόριθμος επιστρέφει το τελικό αποτέλεσμα σε 5.89 sec.

Στην περίπτωση που στον k\_neighbors\_with\_statistics εισαχθεί μόνο το “2d” index, εξετάζονται και πάλι 67484 εγγραφές σε 142ms. Όταν χρησιμοποιηθεί ένα compounding index εξετάζονται 13389 εγγραφές, δηλαδή οι εγγραφές που επιστρέφονται σε 82ms. Ο χρόνος ολοκλήρωσης του αλγόριθμου είναι 6.29 sec και 6.01 sec αντίστοιχα.

Όταν ο χώρος χωριστεί σε 10x10 χωρίς τη χρήση κάποιου index δεν αλλάζει κάτι σε σύγκριση με τα 4x4 κελιά. Εξετάζονται όλες οι εγγραφές σε 41 ms. Όταν χρησιμοποιηθεί ένα index στο χώρο, εξετάζονται και πάλι όλες οι εγγραφές σε 145 ms. Όταν χρησιμοποιηθεί μόνο index στο χρόνο, εξετάζονται 13389 εγγραφές σε 20 ms. Όταν χρησιμοποιηθεί compounding index στο χώρο και στο χρόνο εξετάζονται και πάλι 13389 εγγραφές, αλλά αυτό συμβαίνει σε 84 ms. Οι χρόνοι ολοκλήρωσης του αλγόριθμου είναι 6.46, 6.01, 6.26, 6.17 sec αντίστοιχα

Παρατηρείται, λοιπόν, ότι για το συγκεκριμένο dataset, που όλα τα σημεία βρίσκονται συγκεντρωμένα μέσα σε ένα κελί με διαστάσεις 2x3, για να υπάρξει βελτίωση θα πρέπει ο χώρος να χωριστεί σε πάρα πολλά κελιά.

Όταν ο χώρος χωριστεί σε 1000x1000 κελιά, χωρίς κανένα index εξετάζονται όλες οι εγγραφές και τα αποτελέσματα επιστρέφονται σε 6 sec.



Αν όμως χρησιμοποιηθεί ένα index στο χώρο, τα αποτελέσματα είναι τα εξής:



Η συνολική διάρκεια είναι 5.83 sec. Σίγουρα δεν είναι το καλύτερο δυνατό αποτέλεσμα, αλλά σε σύγκριση με τα προηγούμενα πειράματα, που επέστρεφαν όλο το dataset, υπάρχει μια μικρή βελτίωση, καθώς εξετάζονται λιγότερα έγγραφα και επιστρέφονται ακόμα λιγότερα. Όταν χρησιμοποιηθεί ένα index στο χρόνο, τα αποτελέσματα είναι περισσότερο θετικά με τη διάρκεια όλου του αλγόριθμου να είναι 5.43 sec.



Αλλά τα καλύτερα αποτελέσματα παρουσιάζονται όταν χρησιμοποιηθεί ένα compounding index, με τη συνολική διάρκεια να είναι στα 5.34 sec.



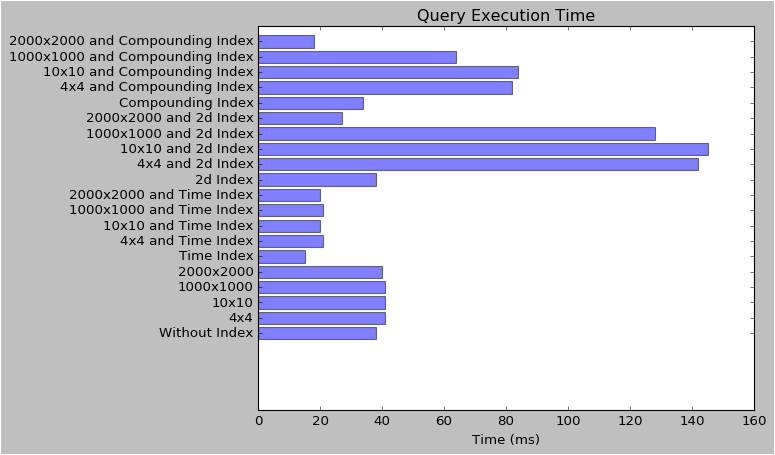
Όταν ο χώρος χωριστεί σε 2000x2000 κελιά τότε χωρίς κανένα index εξετάζονται όλα τα έγγραφα και επιστρέφονται 13389 σε 40 ms και τα τελικά αποτελέσματα σε 5.75 sec. Όταν χρησιμοποιηθεί ένα time index, τότε όπως και στα παραπάνω πειράματα εξετάζονται 13389 έγγραφα σε 20 ms και η συνολική διάρκεια είναι 3.61 sec. Με 2d index ελέγχονται 13836 έγγραφα σε 27 ms και τέλος με τη χρήση του compounding index ελέγχονται 3439 έγγραφα σε 18 ms. Η συνολική διάρκεια είναι 3.79 και 3.59 sec αντίστοιχα. Αξίζει να σημειωθεί ότι με 2000x2000 κελιά, ο αλγόριθμος επιστρέφει 1710 έγγραφα.

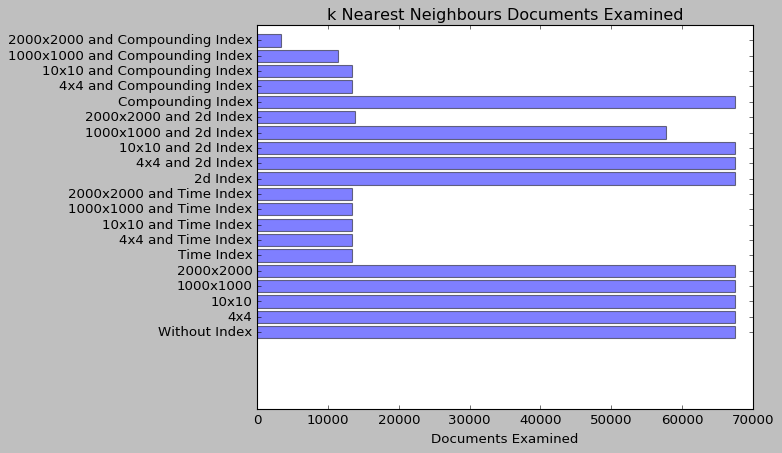
Στον παρακάτω πίνακα παρουσιάζονται συγκεντρωτικά τα αποτελέσματα των αλγόριθμων:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Documents Examined** | **Execution Time (ms)** | **Documents Returned** | **Total Time (sec)** |
| **Without index** | 67484 | 38 | 13389 | 6.44 |
| **Time index** | 13389 | 15 | 13389 | 6.14 |
| **2d index** | 67484 | 38 | 13389 | 6.42 |
| **Compounding index** | 67484 | 34 | 13389 | 5.37 |
| **Statistics (4x4) Without index** | 67484 | 41 | 13389 | 6.21 |
| **Statistics (4x4) Time index** | 13389 | 21 | 13389 | 5.89 |
| **Statistics (4x4) 2d index** | 67484 | 142 | 13389 | 6.29 |
| **Statistics (4x4)**  **Compounding index** | 13389 | 82 | 13389 | 6.01 |
| **Statistics (10x10)**  **Without index** | 67484 | 41 | 13389 | 6.46 |
| **Statistics (10x10) Time index** | 13389 | 20 | 13389 | 6.01 |
| **Statistics (10x10) 2d index** | 67484 | 145 | 13389 | 6.26 |
| **Statistics (10x10) Compounding index** | 13389 | 84 | 13389 | 6.17 |
| **Statistics (1000x1000)**  **Without index** | 67484 | 41 | 11154 | 6.0 |
| **Statistics (1000x1000)**  **Time index** | 13389 | 21 | 11154 | 5.43 |
| **Statistics (1000x1000)**  **2d index** | 57680 | 128 | 11154 | 5.83 |
| **Statistics (1000x1000) Compounding index** | 11415 | 64 | 11154 | 5.34 |
| **Statistics (2000x2000)**  **Without index** | 67484 | 40 | 1710 | 3.99 |
| **Statistics (2000x2000)**  **Time index** | 13389 | 20 | 1710 | 3.61 |
| **Statistics (2000x2000)**  **2d index** | 13836 | 27 | 1710 | 3.79 |
| **Statistics (2000x2000) Compounding index** | 3439 | 18 | 1710 | 3.59 |

Παρατηρείται ότι όταν ο χώρος χωριστεί σε 2000x2000 κελιά και χρησιμοποιηθεί ένα compounding index, τα έγγραφα που εξετάζονται είναι λιγότερα. Σε διαφορετική περίπτωση όταν χρησιμοποιηθούν λίγα κελιά δεν υπάρχει σχεδόν καμία βελτίωση ή η βελτίωση είναι πάρα πολύ μικρή. Επίσης, ο μικρότερος χρόνος ολοκλήρωσης εμφανίζεται όταν χρησιμοποιηθούν στατιστικά δεδομένα, ο χώρος χωριστεί σε 2000x2000 κελιά και χρησιμοποιηθεί κάποιο index.

Στη συνέχεια, παρουσιάζονται κάποια γραφήματα που παρουσιάζουν τον χρόνο εκτέλεσης, καθώς και τα έγγραφα που ελέγχονται.





Με τη χρήση των παραπάνω γραφημάτων είναι περισσότερο εμφανές ότι ο καλύτερος συνδυασμός χρόνου εκτέλεσης του query και εγγράφων που εξετάζονται είναι στην περίπτωση που o χώρος χωρίζεται σε 2000x2000 κελιά και χρησιμοποιείται ένα compounding index.

**Εύρεση k κοντινότερων γειτόνων με διαφορετικό id**

Όπως παρουσιάστηκε στην προηγούμενη ενότητα μια παραλλαγή του παραπάνω αλγόριθμου, που επιστρέφει τους k κοντινότερους γείτονες, είναι η εύρεση των k κοντινότερων γειτόνων που θα προέρχονται από διαφορετικές οντότητες, δηλαδή μεταξύ τους θα έχουν διαφορετικό id.

Θεωρητικά, σε σύγκριση με τον παραπάνω αλγόριθμο ο χρόνος εκτέλεσης των ερωτημάτων θα είναι μεγαλύτερος, όπως και τα έγγραφα που εξετάζονται θα είναι περισσότερα. Αλλά, όπως και στον προηγούμενο αλγόριθμο, τα καλύτερα αποτελέσματα αναμένονται όταν χρησιμοποιηθεί ένα compounding index και ο χώρος χωριστεί σε 2000x2000 κελιά.

Στους παρακάτω πίνακες παρουσιάζονται τα αποτελέσματα όλων πειραμάτων.

**Χωρίς τη χρήση στατιστικών δεδομένων:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Documents Examined** | **Execution Time (ms)** | **Documents Returned** | **Total Time (sec)** |
| **Without index** | 67484 | 34 | 13389 | 10.59 |
| **Time index** | 13389 | 17 | 13389 | 6.51 |
| **2d index** | 67484 | 34 | 13389 | 6.69 |
| **Compounding index** | 67484 | 34 | 13389 | 5.72 |

**Με διαχωρισμό σε 4x4 κελιά**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Documents Examined** | **Execution Time (ms)** | **Documents Returned** | **Total Time (sec)** |
| **Statistics (4x4) Without index** | 67484 | 52 | 13389 | 7.23 |
| **Statistics (4x4) Time index** | 13389 | 23 | 13389 | 6.13 |
| **Statistics (4x4) 2d index** | 67484 | 144 | 13389 | 6.35 |
| **Statistics (4x4)**  **Compounding index** | 67484 | 34 | 13389 | 6.08 |

**Με διαχωρισμό σε 10x10 κελιά**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Documents Examined** | **Execution Time (ms)** | **Documents Returned** | **Total Time (sec)** |
| **Statistics (10x10)**  **Without index** | 67484 | 43 | 13389 | 6.92 |
| **Statistics (10x10) Time index** | 13389 | 20 | 13389 | 6.45 |
| **Statistics (10x10) 2d index** | 67484 | 146 | 13389 | 6.57 |
| **Statistics (10x10) Compounding index** | 67484 | 83 | 13389 | 6.23 |

**Με διαχωρισμό σε 1000x1000 κελιά**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Documents Examined** | **Execution Time (ms)** | **Documents Returned** | **Total Time (sec)** |
| **Statistics (1000x1000)**  **Without index** | 67484 | 41 | 11154 | 6.19 |
| **Statistics (1000x1000)**  **Time index** | 13389 | 21 | 11154 | 5.72 |
| **Statistics (1000x1000)**  **2d index** | 57680 | 129 | 11154 | 5.91 |
| **Statistics (1000x1000) Compounding index** | 11415 | 65 | 11154 | 5.41 |

**Με διαχωρισμό σε 2000x2000 κελιά**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Documents Examined** | **Execution Time (ms)** | **Documents Returned** | **Total Time (sec)** |
| **Statistics (2000x2000)**  **Without index** | 67484 | 42 | 1710 | 4.01 |
| **Statistics (2000x2000)**  **Time index** | 13389 | 21 | 1710 | 3.82 |
| **Statistics (2000x2000)**  **2d index** | 13836 | 27 | 1710 | 3.91 |
| **Statistics (2000x2000) Compounding index** | 3439 | 20 | 1710 | 3.70 |

Όπως ήταν αναμενόμενο ο συνολικός χρόνος εκτέλεσης, σε σύγκριση με τον προηγούμενο αλγόριθμο που δεν ελέγχει το id των εγγραφών που επιστρέφονται, είναι λίγο μεγαλύτερος. Οι υπόλοιπες μετρήσεις είναι σχεδόν ίδιες γιατί ο αλγόριθμος που επιστρέφει τις εγγραφές δεν αλλάζει.

**Circle Range**

Η Circle Range συνάρτηση όπως είναι αναμενόμενο χωρίς κανένα index, θα πρέπει να ελέγξει όλες τις εγγραφές προκειμένου να επιστρέψει κάποιο αποτέλεσμα. Σημαντική βελτίωση υπάρχει με την εισαγωγή index στο χώρο ή στο χρόνο. Οι παράμετροι της συνάρτησης είναι (mycol, -4.75, 48.25, 0.08, 1443690402, 1443700123), δηλαδή το ερώτημα που κλήθηκε να απαντήσει είναι να βρεθούν όλες οι εγγραφές που βρίσκονται μέσα στον κύκλο με κέντρο (-4.75, 48.25) και ακτίνα 0.08, καθώς επίσης και στο χρονικό διάστημα [1443690402, 1443700123].

Αρχικά, όπως αναφέρθηκε χωρίς κανένα index η συνάρτηση εξετάζει όλες τις εγγραφές.



Ο χρόνος εκτέλεσης ήταν 40 ms και το query επέστρεψε 221 έγγραφα.



Αν πραγματοποιηθεί εισαγωγή ενός index στο χώρο, τότε η συνάρτηση εξετάζει 3457 εγγραφές σε 7 ms, δηλαδή η βελτίωση είναι πολύ μεγάλη.

Αν πραγματοποιηθεί και εισαγωγή index μόνο στο χρόνο, τότε οι εγγραφές που εξετάζονται είναι 13389 σε 19 ms.

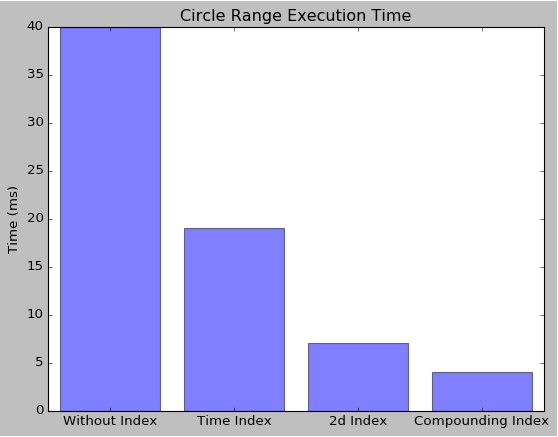
Αν χρησιμοποιούνταν ένα compounding index, τότε τα αποτελέσματα είναι τα καλύτερα σε σύγκριση με τις προηγούμενες μετρήσεις, καθώς εξετάζονται μόνο 766 εγγραφές σε 4 ms.

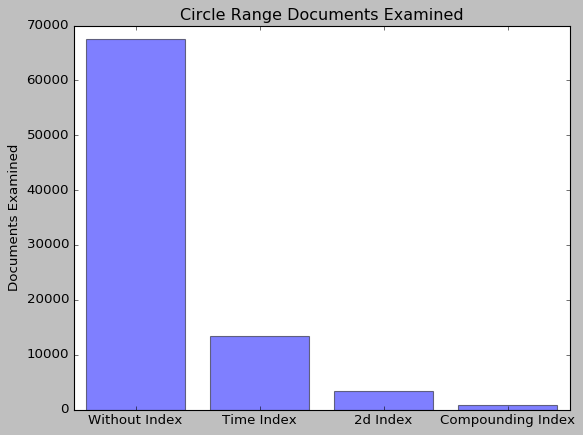
 

Στον παρακάτω πίνακα παρουσιάζονται συγκεντρωτικά τα αποτελέσματα όλων των μετρήσεων για το Circle Range Query.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Documents Examined** | **Execution Time (ms)** | **Total Time (sec)** |
| **Without index** | 67484 | 40 | 2.96 |
| **Time index** | 13389 | 19 | 2.58 |
| **2d index** | 3459 | 7 | 2.56 |
| **Compounding index** | 766 | 4 | 2.53 |

Τα γραφήματα των αποτελεσμάτων είναι τα εξής:





Τα αποτελέσματα είναι απολύτως αναμενόμενα, καθώς όταν δεν χρησιμοποιείται κανένα index εξετάζονται όλα τα έγγραφα και η χρονική διάρκεια είναι υψηλή συγκριτικά με τις υπόλοιπες μετρήσεις. Το γεγονός ότι το 2d index παρουσιάζει καλύτερα αποτελέσματα από το χρονικό index, όσο αφορά το χρόνο εκτέλεσης του query, οφείλεται στο ότι η ακτίνα του κύκλου είναι πολύ μικρή και το χρονικό διάστημα σχετικά μεγάλο. Σε ένα διαφορετικό query, όπου το χρονικό διάστημα θα ήταν πολύ μικρό και η ακτίνα του κύκλου πολύ μεγάλη, προφανώς το χρονικό index θα παρουσίαζε καλύτερα αποτελέσματα. Παρόλα αυτά σε κάθε περίπτωση, ο συνδυασμός των δύο index, δηλαδή το compounding index είναι αυτό που παράγει τα καλύτερα αποτελέσματα.

**Box Range**

H Box Range συνάρτηση χρησιμοποιήθηκε με παραμέτρους (mycol, -4.9, 48.1, -4.7, 48.3, 1443690402, 1443700123), δηλαδή το ορθογώνιο σχηματιζότανε από τις γωνίες (-4.9, 48.1) και (-4.7, 48.3) και το χρονικό διάστημα μέσα στο οποίο πρέπει να βρίσκονται τα δεδομένα είναι [1443690402, 1443700123].

Όπως σε όλα τα παραδείγματα, χωρίς index εξετάζονται όλα τα δεδομένα σε 41 ms. Φυσικά, το query επιστρέφει μόνο 1642 έγγραφα.

Με χρήση index στο χώρο εξετάζονται 8797 εγγραφές σε 20 ms.

Aν χρησιμοποιηθεί μόνο το index στο χρόνο εξετάζονται 13389 έγγραφα σε 30 ms.



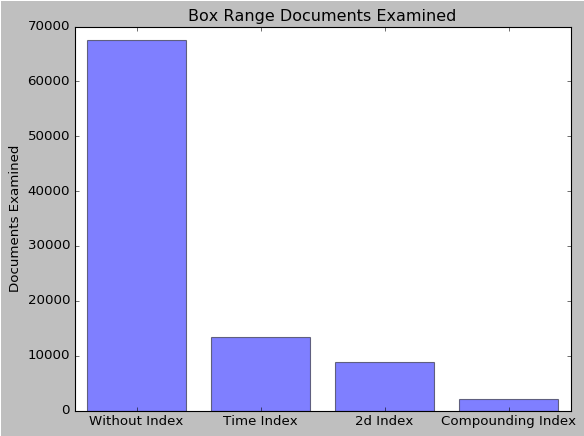
Συνεπώς, με χρήση index είτε στο χώρο, είτε στο χρόνο η αποδοτικότητα του αλγόριθμου αυξάνεται. Παρόλα αυτά όπως και στον Circle Range αλγόριθμο, η μεγαλύτερη αποδοτικότητα εμφανίζεται όταν χρησιμοποιηθεί ένα compounding index. Όπως παρουσιάζεται παρακάτω, εξετάζονται 2173 έγγραφα σε 11 ms.



Συγκεντρωτικά, τα αποτελέσματα του αλγόριθμου είναι τα εξής:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Documents Examined** | **Execution Time (ms)** | **Total Time (sec)** |
| **Without index** | 67484 | 41 | 3.61 |
| **Time index** | 13389 | 30 | 3.50 |
| **2d index** | 8797 | 20 | 3.41 |
| **Compounding index** | 2173 | 11 | 3.33 |





Τα αποτελέσματα μοιάζουν πάρα πολύ με τον Range Circle αλγόριθμο, καθώς το 2d index παρουσιάζει καλύτερα αποτελέσματα από το time index, καθώς λιγότερα έγγραφα ικανοποιούν τον χωρικό περιορισμό, από αυτά που ικανοποιούν τον χρονικό περιορισμό. Τέλος, όπως ήταν αναμενόμενο το compounding index παρουσιάζει τα καλύτερα αποτελέσματα.

**Εξαγωγή όλου του trajectory**

Όπως παρουσιάστηκε παραπάνω μια παραλλαγή των παραπάνω ερωτημάτων είναι για κάθε id που επιστρέφει ο αλγόριθμος, να πραγματοποιείται αναζήτηση όλου του trajectory. Οι μετρήσεις αφορούν τον χρόνο εκτέλεσης του query που θα επιστρέψει όλες τις εγγραφές που περιλαμβάνει το συγκεκριμένο id, όταν η αναζήτηση πραγματοποιηθεί στο αρχικό collection και όταν πραγματοποιηθεί στο collection που λειτουργεί ως inverted index. Φυσικά, αναμένεται η αναζήτηση στο collection που λειτουργεί ως inverted index να πραγματοποιηθεί πολύ γρηγορότερα. Στον παρακάτω πίνακα παρουσιάζονται τα αποτελέσματα των μετρήσεων του χρόνου στα δυο collection. Οι παράμετροι των αλγόριθμων είναι ίδιοι με τα παραπάνω ερωτήματα.

|  |  |  |
| --- | --- | --- |
|  | **Αρχικό Collection Time (ms)** | **Inverted Index Collection Time (ms)** |
| **Box Range** | 79 | 10 |
| **Circle Range** | 48 | 0 |
| **k Nearest Neighbours** | 26 | 0 |

**Πειραματική Μελέτη με χρήση του Δεύτερου Dataset**

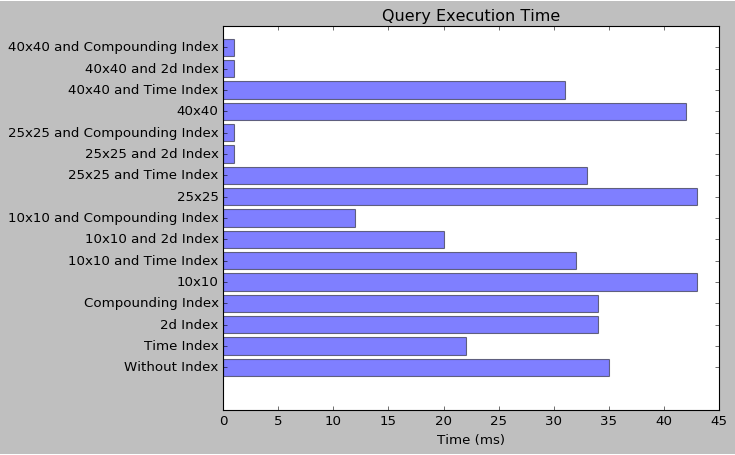
**Εύρεση k κοντινότερων γειτόνων**

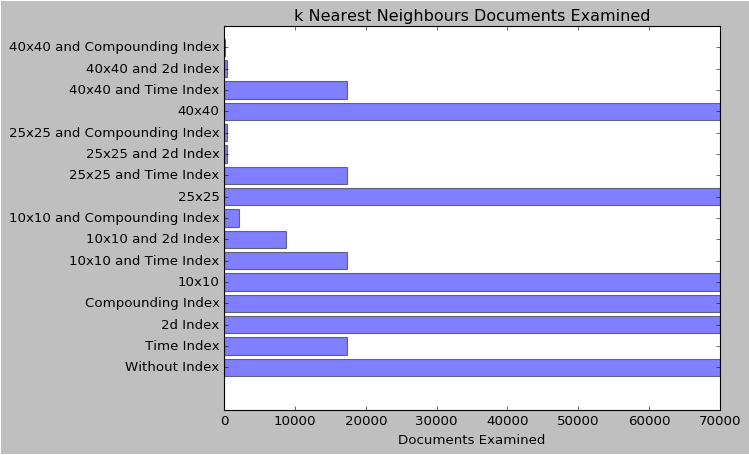
Όπως και στο παραπάνω dataset ο αλγόριθμος μελετήθηκε χωρίς τη χρήση στατιστικών δεδομένων και indexes, καθώς και με τη χρήση μόνο indexes ή μόνο στατιστικών δεδομένων ή συνδυασμό των δύο. Το σημείο στο οποίο θα αναζητηθούν οι κοντινότεροι γείτονες είναι το (1, 1) και το χρονικό διάστημα είναι το (1475000000, 1525000000). Για τη χρήση των στατιστικών δεδομένων, ο χώρος χωρίστηκε σε 10x10, 25x25 και 40x40 κελιά.

Στον παρακάτω πίνακα παρουσιάζονται τα αποτελέσματα όλων των πειραμάτων.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Documents Examined** | **Execution Time (ms)** | **Documents Returned** | **Total Time (sec)** |
| **Without index** | 70000 | 35 | 17356 | 6.57 |
| **Time index** | 17356 | 22 | 17356 | 6.22 |
| **2d index** | 70000 | 34 | 17356 | 6.35 |
| **Compounding index** | 70000 | 34 | 17356 | 6.34 |
| **Statistics (10x10) Without index** | 70000 | 43 | 1222 | 3.15 |
| **Statistics (10x10) Time index** | 17356 | 32 | 1222 | 3.10 |
| **Statistics (10x10) 2d index** | 8680 | 20 | 1222 | 3.01 |
| **Statistics (10x10)**  **Compounding index** | 2124 | 12 | 1222 | 2.98 |
| **Statistics (25x25)**  **Without index** | 70000 | 43 | 69 | 3.18 |
| **Statistics (25x25) Time index** | 17356 | 33 | 69 | 2.79 |
| **Statistics (25x25) 2d index** | 477 | 1 | 69 | 2.17 |
| **Statistics (25x25) Compounding index** | 477 | 1 | 69 | 2.79 |
| **Statistics (40x40)**  **Without index** | 70000 | 42 | 58 | 2.53 |
| **Statistics (40x40)**  **Time index** | 17356 | 31 | 58 | 2.39 |
| **Statistics (40x40)**  **2d index** | 473 | 1 | 58 | 2.16 |
| **Statistics (40x40) Compounding index** | 108 | 1 | 58 | 2.09 |

Παρατηρείται ότι σε όσο περισσότερα κελιά χωρίζεται ο χώρος, τόσο καλύτερα τα αποτελέσματα του αλγόριθμου. Φυσικά, πάντα με τη χρήση κάποιου index είτε στο χώρο, είτε index που συνδυάζει χώρο και χρόνο. Όσο περισσότερα τα κελιά που χωρίζεται ο χώρος, τόσο λιγότερα τα έγγραφα που επιστρέφει ο αλγόριθμος, συνεπώς είναι λιγότερος ο συνολικός χρόνος εκτέλεσης του αλγόριθμου. Στα παρακάτω γραφήματα που απεικονίζουν τον χρόνο εκτέλεσης του query, καθώς και τα έγγραφα που εξετάζει ο αλγόριθμος προκειμένου να επιστρέψει το αποτέλεσμα, είναι περισσότερο εμφανές ότι τα καλύτερα αποτελέσματα παρουσιάζονται όταν ο χώρος χωριστεί σε 40x40 κελιά και χρησιμοποιηθεί 2d ή compounding index. Τα αποτελέσματα είναι πάρα πολύ θετικά, ακόμα και όταν ο χώρος χωριστεί σε 25x25 κελιά και χρησιμοποιηθεί κάποιο από τα δύο index.





**Εύρεση k κοντινότερων γειτόνων με διαφορετικό id**

Γενικότερα, όταν η αναζήτηση αφορά εγγραφές με διαφορετικό id, ο συνολικός χρόνος εκτέλεσης του αλγόριθμου αναμένεται να είναι μεγαλύτερος, καθώς πολλές φορές οι κοντινότερες εγγραφές περιέχουν το ίδιο id, επομένως πρέπει να γίνει επαναϋπολογισμός των κοντινότερων εγγραφών και επίσης πάντα πραγματοποιείται έλεγχος ότι οι εγγραφές δεν περιέχουν το ίδιο id. Στον παρακάτω πίνακα παρουσιάζεται μόνο ο συνολικός χρόνος εκτέλεσης του αλγόριθμου, καθώς τα query δεν αλλάζουν σε σύγκριση με τον παραπάνω αλγόριθμο.

|  |  |
| --- | --- |
|  | **Total Time (sec)** |
| **Without index** | 8.12 |
| **Time index** | 7.07 |
| **2d index** | 7.19 |
| **Compounding index** | 7.14 |
| **Statistics (10x10) Without index** | 3.79 |
| **Statistics (10x10) Time index** | 3.44 |
| **Statistics (10x10) 2d index** | 3.55 |
| **Statistics (10x10)**  **Compounding index** | 3.60 |
| **Statistics (25x25)**  **Without index** | 3.59 |
| **Statistics (25x25) Time index** | 3.01 |
| **Statistics (25x25) 2d index** | 2.82 |
| **Statistics (25x25) Compounding index** | 2.86 |
| **Statistics (40x40)**  **Without index** | 4.72 |
| **Statistics (40x40)**  **Time index** | 3.18 |
| **Statistics (40x40)**  **2d index** | 2.56 |
| **Statistics (40x40) Compounding index** | 2.50 |

Όπως ήταν αναμενόμενο ο συνολικός χρόνος εκτέλεσης του αλγόριθμου που αναζητά εγγραφές με διαφορετικό id όταν συγκρίνεται με τον παραπάνω αλγόριθμο που δεν ελέγχει το id είναι μεγαλύτερος σε όλες τις περιπτώσεις.

**Circle Range**

Αρχικά, ο Circle Range αλγόριθμος εκτελέστηκε με τις εξής παραμέτρους: (1,1, 2, 1475000000, 1525000000), δηλαδή αναζητήθηκε ο κύκλος με κέντρο (1, 1) και ακτίνα 2 και χρονικό διάστημα [1475000000, 1525000000]. Τα αποτελέσματα παρουσιάζονται στον παρακάτω πίνακα.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Documents Examined** | **Execution Time (ms)** | **Documents Returned** |
| **Without index** | 70000 | 43 | 3 |
| **Time index** | 17356 | 31 | 3 |
| **2d index** | 38 | 0 | 3 |
| **Compounding index** | 8 | 0 | 3 |

Παρατηρείται ότι με συνδυασμό των δύο Index, το αποτέλεσμα είναι το βέλτιστο, καθώς το query εκτελείται σε 0 ms και εξετάζονται μόνο 8 έγγραφα.

Αν αλλάξουν οι παράμετροι και πραγματοποιηθεί αναζήτηση των εγγραφών που βρίσκονται μέσα στο κύκλο με κέντρο (1, 1) και ακτίνα 10 στο ίδιο χρονικό διάστημα τότε τα αποτελέσματα είναι τα εξής:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Documents Examined** | **Execution Time (ms)** | **Documents Returned** |
| **Without index** | 70000 | 49 | 73 |
| **Time index** | 17356 | 32 | 73 |
| **2d index** | 500 | 5 | 73 |
| **Compounding index** | 116 | 1 | 73 |

Με αυτά τα νέα δεδομένα και πάλι όπως ήταν αναμενόμενο με τη χρήση του compounding index επιτυγχάνονται τα καλύτερα αποτελέσματα, αλλά σε σύγκριση με τα παραπάνω αποτελέσματα ο χρόνος εκτέλεσης είναι λίγο μεγαλύτερος κυρίως όταν χρησιμοποιείται 2d ή compounding index. Αυτό οφείλεται στο γεγονός ότι εξετάζονται περισσότερα έγγραφα.

Τέλος, οι μετρήσεις θα εκτελεστούν με τις εξής παραμέτρους (1, 1, 15, 1450000000, 1550000000), δηλαδή εκτός από την ακτίνα αυξάνεται και το χρονικό διάστημα μέσα στο οποίο θα αναζητηθούν τα δεδομένα.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Documents Examined** | **Execution Time (ms)** | **Documents Returned** |
| **Without index** | 70000 | 58 | 377 |
| **Time index** | 35005 | 51 | 377 |
| **2d index** | 1738 | 7 | 377 |
| **Compounding index** | 823 | 5 | 377 |

Παρατηρείται ο χρόνος εκτέλεσης του query είναι μεγαλύτερος σε σύγκριση με τα παραπάνω πειράματα σε όλες τις περιπτώσεις.

**Box Range**

Ο Box Range αλγόριθμος εκτελέστηκε με 3 διαφορετικές παραμέτρους. Όπως και στον Circle Range, αρχικά αυξήθηκε ο χώρος μέσα στον οποίο αναζητούνται τα δεδομένα και στη συνέχεια το χρονικό διάστημα. Αρχικά οι παράμετροι ήταν (0, 0, 3, 3, 1475000000, 1525000000).

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Documents Examined** | **Execution Time (ms)** | **Documents Returned** |
| **Without index** | 70000 | 48 | 4 |
| **Time index** | 17356 | 31 | 4 |
| **2d index** | 28 | 3 | 4 |
| **Compounding index** | 6 | 1 | 4 |

Όπως και στον circle range αλγόριθμο, ο συνδυασμός των δύο index οδηγεί στην εξέταση λιγότερων εγγράφων και κατά συνέπεια, στην ελαχιστοποίηση του χρόνου εκτέλεσης του query.

Όταν αυξηθεί μόνο το ορθογώνιο μέσα στο οποίο πραγματοποιείται η αναζήτηση των δεδομένων (0, 0, 10, 10), αυξάνεται ελάχιστα και ο χρόνος εκτέλεσης των query και τα αποτελέσματα είναι τα εξής:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Documents Examined** | **Execution Time (ms)** | **Documents Returned** |
| **Without index** | 70000 | 49 | 28 |
| **Time index** | 17356 | 35 | 28 |
| **2d index** | 178 | 5 | 28 |
| **Compounding index** | 49 | 2 | 28 |

Τέλος, όταν αυξηθεί το χρονικό διάστημα και τον ορθογώνιο παραμείνει όπως το πρώτο πείραμα, δηλαδή οι παράμετροι είναι (0, 0, 3, 3, 1430000000, 1570000000) τότε παρουσιάζονται τα παρακάτω αποτελέσματα.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Documents Examined** | **Execution Time (ms)** | **Documents Returned** |
| **Without index** | 70000 | 65 | 9 |
| **Time index** | 48981 | 60 | 9 |
| **2d index** | 28 | 3 | 9 |
| **Compounding index** | 22 | 1 | 9 |

Η μεγαλύτερη διαφορά σε σύγκριση με τις πρώτες μετρήσεις παρατηρείται όταν χρησιμοποιηθεί κάποιο index στο χρόνο, καθώς όσο αυξάνεται το χρονικό διάστημα τόσο περισσότερα έγγραφα εξετάζονται.

**Εξαγωγή όλου του trajectory**

Όπως και στο πρώτο dataset εξετάζεται ο χρόνος εξαγωγής όλου του trajectory τόσο από το αρχικό collection όσο και από ένα δεύτερο που περιλαμβάνει ένα document με όλα τα στοιχεία για κάθε id.

|  |  |  |
| --- | --- | --- |
|  | **Αρχικό Collection Time (ms)** | **Inverted Index Collection Time (ms)** |
| **Box Range** | 49 | 0 |
| **Circle Range** | 56 | 0 |
| **k Nearest Neighbours** | 39 | 0 |

Στο δεύτερο collection όλοι οι χρόνοι είναι ίσοι με 0 και αυτό είναι απολύτως λογικό, καθώς το συγκεκριμένο dataset περιλαμβάνει μόνο 20 διαφορετικά id, δηλαδή 20 documents, που σε αντίθεση με τα 70000 documents του αρχικού collection, χρειάζονται πολύ λιγότερο χρόνο για να εξεταστούν.