ΘΕΜΑ ΕΡΓΑΣΙΑΣ: 3.K-D Trees και Quad Trees: Πειραματική Σύγκριση των δύο δομών (Storage/Build/Insert/Delete/Update/Search performance comparison) για 2 διαστάσεις. Υλοποίηση kNN Queries με τις παραπάνω δύο δομές και πειραματική σύγκριση.

ΥΛΟΠΟΙΗΣΗ K-D TREE

Τα K-D Trees υλοποιήθηκαν σε python και συγκεκριμένα οι συναρτήσεις storage, build, searching, insertion, deletion, updating, searchingPath, searchingNeighbor και στη συνέχεια παρουσιάζεται ένα παράδειγμα χρήσης του.

Αρχικά, ορίζεται μια κλάση Node για ένα κόμβο του δένδρου, ο οποίος περιέχει το σημείο (location: ζεύγος δύο αριθμών), το αριστερό υποδέντρο (left_child) και το δεξί υποδέντρο (right_child).

Storage-Build (create_tree)

Η συνάρτηση αυτή χτίζει το δέντρο.

Ως είσοδο δίνουμε μία λίστα σημείων (point list) και η συνάρτηση αρχικοποιεί το βάθος στο 0 που αντιπροσωπεύει την ρίζα. Γίνεται έλεγχος για το αν η λίστα είναι άδεια, και αν είναι επιστρέφει το κενό. Εάν δεν είναι υπολογίζει τις k διαστάσεις των σημείων της λίστας και βάση αυτών προκύπτει ο εκάστοτε άξονας (axis) της σύγκρισης, δηλαδή αν θα συγκρίνει την x ή y συντεταγμένη του σημείου. Αυτό στις επόμενες συναρτήσεις ορίζεται κατευθείαν για τις 2 διαστάσεις που μας ενδιαφέρουν (k=2). Η λίστα ταξινομείται βάση του άξονα που υπολογίσαμε και βρίσκουμε την διάμεσο. Έτσι μπορούμε να καλέσουμε αναδρομικά την create_tree() 2 φορές, μία για το δεξί και μία για το αριστερό κομμάτι της λίστας. Η συνάρτηση τερματίζει όταν δεν υπάρχουν άλλα παιδιά.

Searching

Η συνάρτηση αυτή αναζητά εάν υπάρχει συγκεκριμένος κόμβος στο δέντρο.

Ως είσοδο η συνάρτηση δέχεται το δέντρο tree, το σημείο που αναζητούμε και αρχικοποιεί το βάθος στο 0 για να ξεκινήσει την αναζήτηση. Ελέγχεται το δέντρο για να διαπιστωθεί εάν υπάρχει ή εάν είναι άδειο και στην περίπτωση αυτή η συνάρτηση επιστρέφει «False» και τερματίζει. Εφ' όσον το δέντρο υπάρχει, γίνεται σύγκριση του κόμβου όπου βρισκόμαστε (αρχικά η ρίζα) με το ζητούμενο σημείο. Σε περίπτωση ταύτισης, επιστρέφεται «True» και η συνάρτηση τερματίζεται έχοντας βρει το ζητούμενο σημείο. Διαφορετικά, καλείται αναδρομικά η searching(), βάσει του άξονα axis της κάθε επανάληψης, είτε στο δεξί είτε στο αριστερό υποδέντρο.

Insertion

Η συνάρτηση αυτή εισάγει νέο κόμβο στο δέντρο.

Ως είσοδο δίνουμε ένα υπάρχον δέντρο, ένα σημείο και η συνάρτηση αρχικοποιεί το βάθος στο 0 που αντιπροσωπεύει την ρίζα. Ελέγχει αν το δέντρο είναι κενό και αν αυτό ισχύει το δοθέν σημείο αποτελεί όλο το δέντρο, το οποίο και επιστρέφεται. Διαφορετικά, αποθηκεύουμε τη θέση στην οποία βρισκόμαστε στο δέντρο και υπολογίζουμε τον εκάστοτε άξονα (axis). Οι συντεταγμένες ελέγχονται εναλλάξ και ανάλογα εάν το σημείο είναι μεγαλύτερο, μικρότερο ή ίσο βάση της συντεταγμένης από τη θέση μας καλούμε αναδρομικά την συνάρτηση insertion() για το αριστερό ή το δεξί υποδέντρο αυξάνοντας κάθε φορά το βάθος κατά 1.

Deletion

Η συνάρτηση αυτή διαγράφει έναν κόμβο από το δέντρο.

Ως είσοδο δίνουμε ένα υπάρχον δέντρο, ένα σημείο και η συνάρτηση αρχικοποιεί το βάθος στο 0 που αντιπροσωπεύει την ρίζα. Η συνάρτηση τερματίζει αν το δέντρο είναι άδειο επιστρέφοντας το κενό. Εάν η συνάρτηση δεν τερματίσει αποθηκεύουμε τη θέση στην οποία βρισκόμαστε στο δέντρο και ελέγχουμε αν αυτή ισούται με τη θέση του σημείου που ψάχνουμε να διαγράψουμε. Σ’ αυτή την περίπτωση, προχωράμε στην διαδικασία διαγραφής. Διαφορετικά, υπολογίζουμε τον εκάστοτε άξονα (axis) και ανάλογα τη θέση του σημείου σε σχέση με τη θέση μας, καλείται αναδρομικά η συνάρτηση deletion() για το δεξί ή το αριστερό υποδέντρο.

Κατά την διαδικασία διαγραφής, αρχικά ελέγχουμε αν υπάρχουν παιδιά. Εάν βρισκόμαστε σε φύλλο τότε απλά το διαγράφει, επιστρέφοντας κενό. Εναλλακτικά, πρέπει ο κόμβος να αντικατασταθεί με το επόμενο υπόδεντρο. Δημιουργούμε δύο πίνακες, τον nodes, που περιέχει όλους τους απόγονους του κόμβου που διαγράφεται και τον already_passed, στον οποίο αποθηκεύονται κάθε φορά τα πρώτα στοιχεία του πίνακα nodes και συγκεκριμένα η τιμή του. Στην συνέχεια αντικαθιστούν τον κόμβο τα παιδιά του και αυτός διαγράφεται. Αφού ολοκληρωθεί η διαδικασία, καλείται η συνάρτηση create_tree() με ορίσματα τον πίνακα already_passed και το βάθος στο οποίο βρισκόμαστε, προκειμένου να κατασκευαστεί το νέο δέντρο το οποίο και επιστρέφεται.

Updating

Η συνάρτηση αυτή διαγράφει έναν παλιό κόμβο και εισάγει έναν νέο.

Ως είσοδο η συνάρτηση δέχεται το δέντρο tree, το σημείο που αναζητούμε να αντικαταστήσουμε (old) και το νέο κόμβο που θα αντικαταστήσει τον παλιό (new). Καλείται η συνάρτηση searching(), με ορίσματα το tree και τον κόμβο old. Εφ’ όσον

αυτή επιστρέψει true, βρίσκοντας τον κόμβο στο δέντρο, το διαγράφει μέσω της deletion() και εισάγει το νέο κόμβο new μέσω της insertion(). Επιστρέφεται το ανανεωμένο δέντρο.

SearchingPath

Η συνάρτηση αυτή υπολογίζει τα μονοπάτια από τη ρίζα προς έναν κόμβο προορισμού του δέντρου.

Ως είσοδο η συνάρτηση δέχεται το δέντρο tree, ένα σημείο point και αρχικοποιεί το μονοπάτι path στο None (άδειο) και το βάθος depth στο 0. Στα πρώτα βήματα, εάν βρισκόμαστε δηλαδή στο 0 βάθος, στο path ανατίθεται μια άδεια λίστα μέσω του list(). Έπειτα γίνεται έλεγχος για το αν το tree είναι άδειο και, εάν είναι, το path επιστρέφεται. Αποθηκεύουμε τη θέση (location) που βρισκόμαστε και ορίζουμε το βάθος του κόμβου αυτού στο βάθος που έχουμε. Έπειτα προσθέτουμε τον κόμβο αυτό στο μονοπάτι path.

Στη συνέχεια ελέγχουμε εάν ο κόμβος που βρισκόμαστε είναι ο ζητούμενος κόμβος προορισμού. Εάν είναι, τότε επιστρέφεται το path. Εάν όχι, ορίζουμε τον άξονα axis που θα ελεγχθεί βάσει του βάθους και καλούμε αναλόγως αναδρομικά τη συνάρτηση searchingPath() στο αριστερό ή το δεξί υποδέντρο, αυξάνοντας κάθε φορά το βάθος κατά 1. Τέλος, επιστρέφεται το ζητούμενο path.

SearchingNeighbor

Η συνάρτηση αυτή υπολογίζει το κοντινότερο σημείο σε έναν συγκεκριμένο κόμβο του δέντρου.

Ως είσοδο δέχεται το δέντρο tree και ένα σημείο point και υπολογίζει το μονοπάτι path μέσω της συνάρτησης searchingPath(). Αφαιρεί από τη λίστα path τον τελευταίο κόμβο, αποθηκεύοντάς τον ως lastNode και υπολογίζει την απόστασή του, Distance από τον κόμβο προορισμού point. Αυτό γίνεται μέσω της συνάρτησης range(), η οποία έχει οριστεί από εμάς και υπολογίζει τη διανυσματική απόσταση μεταξύ δύο σημείων στο χώρο. Η απόσταση Distance αποθηκεύεται ξανά σε μεταβλητή radius (ακτίνα) και αρχικοποιείται ο κόμβος ελάχιστης απόστασης (minimum) ως ο lastNode.

Ξεκινά βρόγχος επανάληψης, ο οποίος εκτελείται όσο υπάρχει υπολειπόμενο μονοπάτι, δηλαδή όσο η λίστα path δεν είναι άδεια. Η παραπάνω διαδικασία αφαίρεσης του τελευταίου κόμβου και υπολογισμού της απόστασής του επαναλαμβάνεται. Για την ανανέωση της τιμής της ακτίνας radius και του κόμβου minimum, ελέγχουμε ότι βρισκόμαστε εντός της ακτίνας, δηλαδή ότι η απόσταση Distance που υπολογίσαμε είναι μικρότερη από την αποθηκευμένη ακτίνα radius.

Ύστερα, υπολογίζεται ο εκάστοτε άξονας axis μέσω του βάθους του lastNode. Εάν η απόλυτη τιμή της διαφοράς της θέσης του lastNode με το σημείο προορισμού στον άξονα axis που υπολογίζουμε είναι μικρότερη της ακτίνας radius, τότε ελέγχουμε αν η θέση του σημείου point είναι μικρότερη από τη θέση του σημείου lastNode στον άξονα. Εάν αυτό αληθεύει, γίνεται έλεγχος για την ύπαρξη του δεξιού υποδέντρου. Εάν αυτό υπάρχει, το προσθέτουμε στο path, αυξάνοντας το βάθος του κατά 1. Στη περίπτωση που η θέση του σημείου point στον άξονα είναι μεγαλύτερη της θέσης του lastNode, κάνουμε την ίδια διαδικασία για το αριστερό υποδέντρο.

Η συνάρτηση επιστρέφει τον κοντινότερο γείτονα (σημείο minimum) του σημείου προορισμού point καθώς και την μεταξύ τους απόσταση.

ΥΛΟΠΟΙΗΣΗ QUAD TREE

Τα Quad Trees υλοποιήθηκαν με παρόμοιο τρόπο σε python. Υλοποιήθηκαν επίσης οι συναρτήσεις storage, searching, insertion, deletion, updating, searchingPath, searchingNeighbor.

Storage-Build

Η συνάρτηση αυτή κατασκευάζει το δέντρο.

Η συνάρτηση δέχεται ως είσοδο ένα point list σημείων δύο συντεταγμένων και κτίζει το δέντρο, το οποίο αποτελείται από κόμβους (Nodes), που περιλαμβάνουν την τιμή τους (location) καθώς και 4 παιδιά (tLeft, tRight, bRight, bLeft), που αποθηκεύουν τα σημεία που βρίσκονται στην αντίστοιχη κατεύθυνση. Γίνεται έλεγχος για το αν η λίστα είναι άδεια και, αν είναι, επιστρέφει το κενό. Τα σημεία ταξινομούνται στους κόμβους ανάλογα με το αποτέλεσμα της σύγκρισής τους με τον αρχικό κόμβο.

Επίσης, υλοποιείται μια συνάρτηση child() που συγκρίνει ένα σημείο με ένα κόμβο και επιστρέφει την κατεύθυνση που χρειάζεται να κινηθούμε, ορίζεται ακέραιος με τιμές 1-4, που αντιστοιχεί σε tLeft, tRight, bRight, bLeft αντίστοιχα ανάλογα την τιμή.

Searching

Η συνάρτηση αυτή αναζητά το δοθέντα κόμβο στο δέντρο και λειτουργεί ακριβώς όπως η αντίστοιχη των K-D Trees. Γίνεται έλεγχος, αν το κλειδί είναι ο τρέχον κόμβος, αλλιώς με τη συνάρτηση child() οδηγούμαστε στο παιδί με αναδρομικό τρόπο. Αν φτάσουμε σε φύλλο (επιστρέφεται τιμή None), η τιμή δεν βρέθηκε.

Insertion

Η συνάρτηση αυτή εισάγει έναν κόμβο στο δέντρο.

Η συνάρτηση λειτουργεί με τον ίδιο τρόπο με την insertion() των K-D Trees με την εξαίρεση πως η αναδρομή καλείται για κάποιο από τα 4 παιδιά του κόμβου. Γίνεται διάσχιση του δένδρου με χρήση αυτής της συνάρτησης μέχρι να φτάσει σε φύλλο, όπου εισάγουμε τον κόμβο.

Deletion

Η συνάρτηση αυτή διαγράφει έναν κόμβο από το δέντρο.

Η συνάρτηση λειτουργεί με τον ίδιο τρόπο με τη deletion() των K-D Trees, μέσω αποθήκευσης των already_passed nodes σε προσωρινό πίνακα και επανακατασκευή των υποδέντρων που «χαλάνε» με τη διαγραφή του κόμβου. Η διαφοροποίηση έγκειται στο ότι και πάλι το υποδέντρο που θα κληθεί στην αναδρομή υπολογίζεται με χρήση της συνάρτησης child().

Updating

Η συνάρτηση αυτή ενημερώνει έναν υπάρχον κόμβο του δέντρου, αντικαθιστώντας τον με τον κόμβο-όρισμά της, με χρήση των deletion() και insertion().

SearchingPath

Η συνάρτηση αυτή αναζητά και αποθηκεύει το μονοπάτι κόμβων του δέντρου προς τον κόμβο-όρισμά της. Λειτουργεί με τον ίδιο τρόπο με την αντίστοιχη συνάρτηση των K-D Trees, χρησιμοποιώντας την αντίστοιχη συνάρτηση distance για το Quad δέντρο.

SearchingNeighbor

Η συνάρτηση αυτή υπολογίζει τον κοντινότερο σημείο σε έναν συγκεκριμένο κόμβο του δέντρου. Λειτουργεί όπως η searchingNeighbor() των K-D Trees, με τη διαφορά ότι υπάρχουν 3 συνθήκες που ελέγχει για να προσθέσει το κατάλληλο παιδί στο μονοπάτι: υπολογίζει την απόλυτη τιμή της διαφοράς της απόστασης του σημείου με τον εκάστοτε τελευταίο κόμβο ανά άξονα, και τη συγκρίνει με την ακτίνα radius. Για κάθε συνθήκη, ανάλογα το αποτέλεσμα της σύγκρισης, προστίθενται στο μονοπάτι τα κατάλληλα παιδιά, τα οποία θα περιέχουν τον κοντινότερο γείτονα.

ΣΥΓΚΡΙΣΗ - ΣΥΜΠΕΡΑΣΜΑΤΑ

Από το σύνδεσμο που δόθηκε, κατέβηκε ένα σύνολο δεδομένων που βρίσκεται στο αρχείο data.csv, το οποίο φορτώνεται με κατάλληλο τρόπο στην Python.
Γίνονται οι βασικές λειτουργίες κάθε δομής και γίνεται μέτρηση του χρόνου που χρειάζεται η κάθε λειτουργία. Τα αποτελέσματα φαίνονται παρακάτω.

K-D Trees

```
Creation
0.0004095999999999961
Search a node
0.0001708000000000126
Delete a node
3.549999999999387e-05
Insert a node
1.180000000000625e-05
Find the nearest neighbor
4.479999999999762e-05

Process finished with exit code 0
```

Quad Trees

```
Creation
0.00043390000000000095
Search a node
8.600000000004437e-06
Delete a node
2.0199999999997997e-05
Insert a node
2.889999999999837e-05
Find the nearest neighbor
0.00012240000000000167

Process finished with exit code 0
```

Παρατηρούμε, λοιπόν, ότι η διαγραφή και η αναζήτηση κοντινότερου γείτονα είναι πιο γρήγορες στα Quad Trees. Επίσης, στα Quad Trees κάθε κόμβος έχει τέσσερα παιδιά, το δέντρο αποτελείται από λιγότερα επίπεδα και επομένως απαιτούνται λιγότερα βήματα για τη διάσχισή του, με αποτέλεσμα το μονοπάτι να εντοπίζεται γρηγορότερα στην περίπτωση του NN search.

Στις περιπτώσεις των creation, search και insert, παρατηρούμε καλύτερους χρόνους στα K-D Trees, πιθανόν γιατί αυτά έχουν δύο παιδιά ανά κόμβο, ενώ τα Quad Trees έχουν τέσσερα, επομένως απαιτούνται περισσότερες πράξεις στην περίπτωση των Quad Trees.