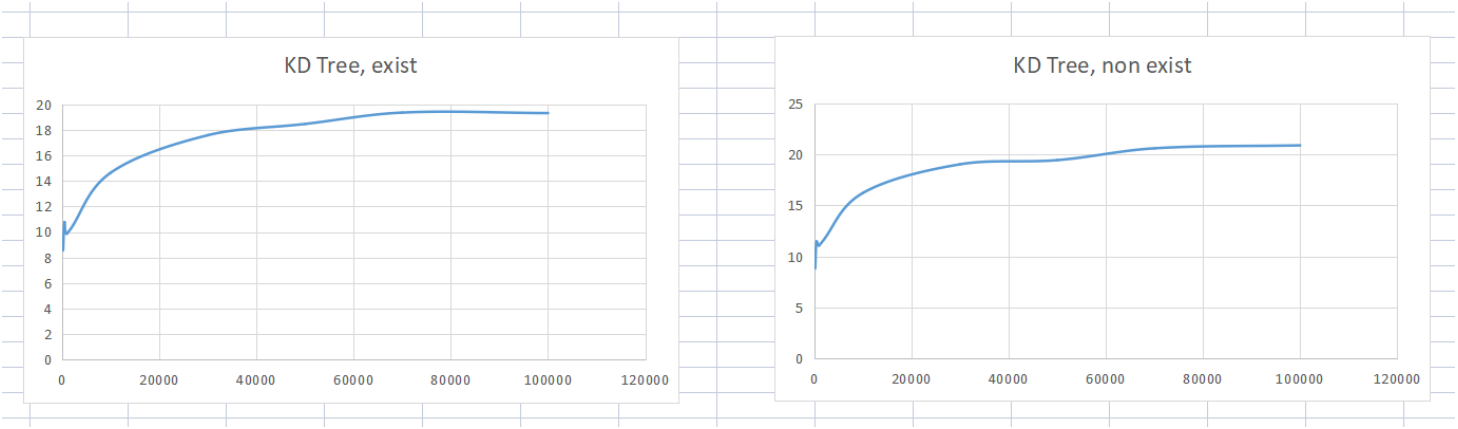
ΣΧΟΛΙΑΣΜΟΣ ΑΡΧΕΙΟΥ EXCEL

Στο αρχείο excel περιέχονται : ο πίνακας με τα αποτελέσματα των αναζητήσεων στις 2 δομές δέντρων καθώς και οι γραφικές παραστάσεις των αναζητήσεων σε συνάρτηση με το πλήθος των ζευγαριών.

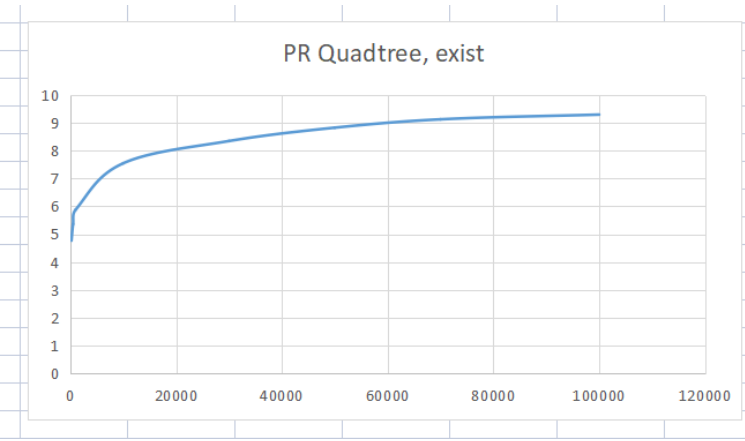
		KD Tree		PR Quadtree	
		exist	non exist	exist	non exist
M	200	8.56	8.82	4.77	3.55
M	500	10.76	11.46	5.36	4.23
M	1000	9.88	11.08	5.88	4.74
M	10000	14.64	16.22	7.55	6.29
M	30000	17.58	19.04	8.35	7.15
M	50000	18.46	19.45	8.83	7.52
M	70000	19.35	20.61	9.13	7.67
M	100000	19.31	20.9	9.3	7.88

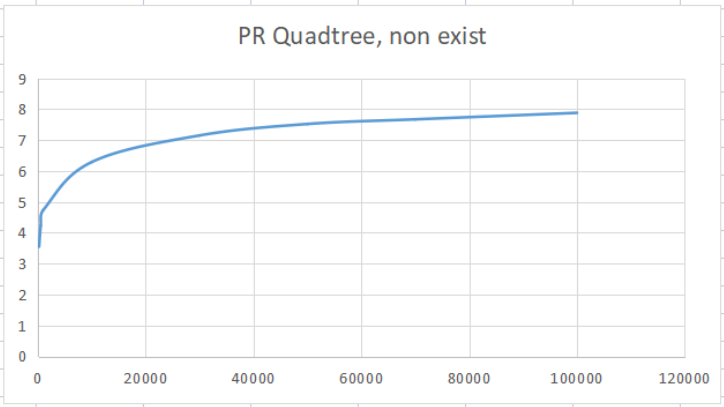
Το M είναι το πλήθος των ζευγαριών. Για κάθε τιμή του M έχουμε για κάθε δέντρο τον μέσω όρο του depth που έφτασε το δέντρο για 100 αναζητήσεις. Για κάθε δέντρο έγιναν 100 αναζητήσεις αρχικά με μοναδικά ζευγάρια που σίγουρα υπάρχουν στο δέντρο, όπου τα αποτελέσματα είναι κάτω από τον τίτλο exist και έπειτα έγιναν 100 αναζητήσεις αρχικά με μοναδικά ζευγάρια που σίγουρα ΔΕΝ υπάρχουν στο δέντρο, όπου τα αποτελέσματα είναι κάτω από τον τίτλο non exist.

1) Το KD Tree όταν αναζητεί ζευγάρια που υπάρχουν στο δέντρο φτάνει σε μικρότερο βάθος, διότι τις περισσότερες φορές χρειάζεται να διασχίσει ολόκληρο το δέντρο όταν αναζητείται στοιχείο που δεν υπάρχει.



Το PR Quadtree θα εκτελέσει μία φορά αναζήτηση για να επαληθευτεί η ύπαρξη ενός στοιχείου που αναζητούμε. Η αναζήτηση αυτή γίνεται στον κόμβο ο οποίος είναι φύλλο (leaf) και μέχρι να φτάσει στο φύλλο, οδηγείται μέσω των κόμβων που είναι pointers σε 4 κόμβους (είτε φύλλα είτε δείκτες). **Οι γραφικές παραστάσεις των αναζητήσεων έχουν μικρή διαφορά (για μη επιτυχημένες αναζητήσεις το depth είναι περίπου κατά 1 μικρότερο) η οποία οφείλεται στο γεγονός ότι κάθε node έχει 4 sub-nodes, οπότε το depth μπορεί να μειώνεται, και τα περισσότερα nodes είναι pointers σε 4 sub-nodes και πιθανόν να δείχνουν στο στοιχείο που δεν υπάρχει πιο νωρίς.**





2) Και οι 2 δομές δέντρων παρουσιάζουν πολυπλοκότητα $\mathcal{O}(log_2 N)$

3) Πιο γρήγορη δομή φαίνεται να είναι το PR Quadtree και στις 2 περιπτώσεις αναζητήσεων γλυτώνει levels επειδή κάθε node έχει 4 sub-nodes καθώς επίσης και με την ιδιότητα των nodes να είναι pointers οπότε δείχνουν πιο γρήγορα και κατευθύνουν στα στοιχεία που αναζητούμε.

4) Η μέθοδος αναζήτησης του KD Tree απαιτεί σε κάθε level να γίνεται σύγκριση των τιμών x, y με τις αντίστοιχες του ήδη υπάρχοντος κόμβου οπότε χρειάζεται συνεχώς να εκτελούνται αριθμητικές πράξεις από τον υπολογιστή. Αντίθετα στην περίπτωση την αναζήτησης του PR Quadtree οι κόμβοι που

είναι δείκτες θα οδηγήσουν σε έναν κόμβο που είναι φύλλο και θα πραγματοποιηθεί μία και μόνο αναζήτηση μεταξύ τιμών x, y.

ΣΧΟΛΙΑΣΜΟΣ ΚΩΔΙΚΑ

Το πρόγραμμα έχει γραφτεί στη γλώσσα python στο IDE PyCharm.

Περιέχει 6 classes : **1) KD_tree**

2) Node

3) PR_quadtree

4) Leaf

5) Main

6) RandomPairs

Για την υλοποίηση της δομής **Kd tree**:

Χρησιμοποιούνται οι κλάσεις Node και KD_tree. **Η κλάση Node περιέχει τις τιμές (x, y) και τα left και right nodes, τα οποία είναι επίσης objects τύπου Node.** Το KD_tree περιγράφει το δέντρο και περιέχει τις βασικές συναρτήσεις insert, search.

Η KD_tree έχει ένα root node το οποίο είναι το πρώτο node που γίνεται insert στο δέντρο.

Η συνάρτηση insert παίρνει ως όρισμα ένα node, όπου είναι ο κόμβος που θα εισχωρήσει στο δέντρο, ορίζει μια μεταβλητή level = 0 και καλείται η συνάρτηση insert recursive ,με ορίσματα το root-node, το node και το level, με την εντολή

self.root = self.insert_recursive(self.root, node, level)

def insert_recursive(self, root, node, level) : Είναι αναδρομική συνάρτηση. Ξεκινάει και ελέγχει αν το root είναι None (αντίστοιχο null), αν ισχύει αυτή η συνθήκη επιστρέφει το node. Έπειτα (στην

περίπτωση που δεν ισχύει η 1η συνθήκη) εκτελεί μία πράξη που ελέγχει αν θα γίνει σύγκριση με την τιμή του X ή του Y.
Αν level % 2 == 0 , τότε θα αυξηθεί η μεταβλητή level κατά 1 και θα συγκρίνει με την τιμή του X. Αν level % 2 == 1, θα αυξηθεί η μεταβλητή level κατά 1 και θα συγκρίνει με την τιμή του Y. Την 1η φορά που level = 0 θα συγκρίνει τα X, σε επόμενη αναδρομική κλήση της συνάρτησης το level = 1 οπότε θα συγκρίνει τα Y και όσο γίνεται αναδρομή θα γίνεται σύγκριση εναλλάξ.

Αναδρομή γίνεται με τον εξής τρόπο : Όταν συγκρίνονται οι τιμές είτε με το X είτε με το Y, συγκρίνεται η μία τιμή του node με την αντίστοιχη τιμή του root. Αν η τιμή του node είναι μικρότερη από αυτή του root τότε ξανακαλείται η συνάρτηση με την ανανεωμένη τιμή του level, το node και το αριστερό sub-node του τρέχοντος root το root.left με την εντολή :
root.left = self.insert_recursive(root.left, node, level). Ό,τι γίνει return από τη συνάρτηση θα γίνει assign στο root.left (δηλαδή από την 1η συνθήκη το root.left θα πάρει την τιμή του node ή θα γίνουν κι άλλες αναδρομές) . Αν η τιμή του node είναι μεγαλύτερη από αυτή του root , τότε εκτελείται η ίδια διαδικασία και με αυτόν τον τρόπο σχηματίζεται το δέντρο !

Η αναζήτηση στο KD_tree έχει σχεδιαστεί με παρόμοιο τρόπο με το insert. Η συνάρτηση search Παίρνει ως όρισμα το node, το οποίο θα αναζητήσει, ορίζει μια μεταβλητή level = 0 και μία depth με την οποία καλεί την αναδρομική συνάρτηση depth_search με ορίσματα το root-node, το node και το level. Η τιμή που επιστρέφει η depth_search είναι το depth στο οποίο έφτασε το δέντρο όταν εκτελούσε την αναζήτηση, εκχωρείται στο depth και γίνεται return από τη search.

def depth_search(self, root, node, level): Είναι αναδρομική συνάρτηση, έχει τους ίδιους ακριβώς ελέγχους με την insert με διαφορετικά return statements. Ο πρώτος έλεγχος για None root επιστρέφει level – 1 διότι τότε θα είμαστε ένα επίπεδο κάτω από το max depth του δέντρου. Ο επόμενος έλεγχος ελέγχει ταυτόχρονα τις τιμές του X και του Y και αν είναι ίσες αυτές του root με αυτές του node τότε επιστρέφει το level. Έπειτα κάνει τους ελέγχους για το ποιες τιμές να συγκρίνει και όταν τις συγκρίνει

κάνει return την ίδια τη συνάρτηση με το σωστό sub-node και την αυξημένη κατά 1 τιμή του level.

Για την υλοποίηση της δομής **PR quadtree**:

Χρησιμοποιούνται οι κλάσεις PR_quadtree και Leaf.

Η Leaf είναι η αντίστοιχη Node για το PR_quadtree, περιέχει τις τιμές X, Y και άλλες 4 μεταβλητές τύπου Leaf αρχικοποιημένες σε None, τις nw, sw, ne, se. Όταν η περιοχή στην οποία βρίσκεται ένα Leaf χωρίζεται σε 4 ίσα κομμάτια αυτές οι μεταβλητές είναι το κάθε κομμάτι. Σημείο αναφοράς είναι το Leaf και οι μεταβλητές αυτές είναι οι περιοχές : north-west, south-west, north-east, south-east. **Η class Leaf έχει επίσης μια συνάρτηση, την is_leaf**, που ελέγχει αν το object είναι leaf ή pointer στα 4 sub-leaves και μία ακόμα συνάρτηση **την def expand_leaf (self, x, y)** , η οποία μετατρέπει το object σε pointer, δηλαδή περνάει τις τιμές X, Y σε μία από τις 4 υπο-περιοχές με βάση τις συντεταγμένες του κέντρου που τις χωρίζει.

Η κλάση PR_quadtree περιέχει τις βασικές συναρτήσεις insert και search και τη μεταβλητή N = 2**16(δηλαδή 2^{16}).

Η συνάρτηση insert παίρνει ως όρισμα ένα leaf τύπου Leaf που θέλουμε να προσθέσουμε στο δέντρο και καλεί την αναδρομική συνάρτηση insert_recursive με ορίσματα το root-leaf, το leaf, ως X το N/2, ως Y το N/2 (και ως div το 2).

def insert_recursive(self, root, leaf, x, y, div): Είναι αναδρομική συνάρτηση, το X, Y είναι οι συντεταγμένες του κέντρου της περιοχής που βρισκόμαστε σε μία κλήση της συνάρτησης. Το δέντρο όταν δημιουργείται καταλαμβάνει μια περιοχή μεγέθους N×N και οι αρχικές συντεταγμένες του κέντρου είναι (N/2 , N/2) . Η συνάρτηση ελέγχει πρώτα αν το root is None, αν είναι γίνεται return το leaf. Αν δεν είναι τότε ελέγχονται τα επόμενα :

- Αν ισχύει root.isLeaf() = true τότε καλείται η συνάρτηση expand_leaf() για το current root (εκτελείται η γραμμή root.expand_leaf(X, Y)),όπου τα X, Y τα παίρνει από τα ορίσματα της συνάρτησης.

- Είτε ισχύει η root.isLeaf() είτε όχι ελέγχονται οι συντεταγμένες του leaf με τις current συντεταγμένες του κέντρου της περιοχής που εξετάζουμε για να γίνει insert το leaf. Με βάση το ποια σύγκριση θα είναι σωστή, πρόκειται να γίνει αναδρομή με ένα από τα sub-leaf του root.

Αναδρομή γίνεται με τον εξής τρόπο:

Όταν βρεθεί η σωστή σύγκριση (μπει το πρόγραμμα στο σωστό if ή else if) υπολογίζονται οι νέες συντεταγμένες του κέντρου της περιοχής που πρόκειται να εξετάσουμε όταν κληθεί η συνάρτηση αμέσως μετά και καλείται η συνάρτηση με root-leaf το sub-leaf με το σωστό προσανατολισμό. Για παράδειγμα
root.sw = self.insert_recursive(root.sw, leaf, x, y, div), όπου τώρα θα τσεκάρει το south-west κόμβο του προηγουμένως root.

Οι υπολογισμοί των κέντρων γίνονται με τον εξής τρόπο :

Όταν είμαστε σε μια περιοχή η οποία θεωρείται τετράγωνο μέσα στην περιοχή N×N με κέντρο (X, Y) και θέλουμε να μεταπηδήσουμε σε ένα από τα 4 τετράγωνα της περιοχής, πρέπει να υπολογίσουμε τις συντεταγμένες του κέντρου του. Τα νέα X, Y θα είναι ίσα με τα προηγούμενα X ή Y και ανάλογα + ή – με N/ 2^{div}. Το div δείχνει πόσες φορές πρέπει να διαιρεθεί το N με το 2. Αυτή η διαίρεση του N μετατοπίζει το προηγούμενο κέντρο στο κέντρο της νέας περιοχής. **Πριν γίνει η αναδρομική κλήση αυξάνουμε το div κατά 1 έτσι ώστε όταν ξαναγίνει η αναδρομή η μετατόπιση των κέντρων να είναι μικρότερη όσο δηλαδή μικραίνουν και τα τετράγωνα.** Οι operators + - καθορίζουν ποιον προσανατολισμό θα ακολουθήσει το πρόγραμμα. Για παράδειγμα αν θέλουμε να πάμε στο τετράγωνο north-west (πάνω αριστερά) θα μειώσουμε τα X, Y για να πάμε πάνω αριστερά.

<u>Συνοψίζοντας για το insert</u>: Η συνάρτηση insert_recursive αρχικά ελέγχει αν το root is None και βλέπει αν υπάρχει άλλο στοιχείο σε εκείνη την περιοχή. Αν υπάρχει τότε ελέγχει αν το root.is_leaf(). Αν είναι leaf τότε σημαίνει ότι στην περιοχή που πρόκειται να γίνει insert ο καινούριος κόμβος υπάρχει ήδη στοιχείο οπότε ο υπάρχων κόμβος πρέπει να κάνει root.expand_leaf() έτσι ώστε να γίνει

pointer, περαστεί η τιμή (X, Y) σε κάποια από τις υπο-περιοχές και στη συνέχεια να γίνουν οι συγκρίσεις έτσι ώστε να γίνει αναδρομή με το σωστό sub-leaf του root με τα ανανεωμένα κέντρα.

Η αναζήτηση σε ένα PR_quadtree έχει παρόμοιες συνθήκες με το insert. Η συνάρτηση search παίρνει ως όρισμα ένα leaf που θέλουμε να δούμε αν υπάρχει στο δέντρο και καλεί (κάνει return) την αναδρομική συνάρτηση depth_search με τιμές κέντρων το N/2, N/2.

def depth_search(self, root, leaf, x, y, div, level) : Είναι αναδρομική συνάρτηση όπου τα ορίσματα αντιπροσωπεύουν τα ίδια μεγέθη με αυτά του *insert_recursive* . Αρχικά αν το root is None επιστρέφεται level – 1. Έπειτα αν root.is_leaf() = true τότε ελέγχουμε τις τιμές X, Y του root και leaf αν είναι ίσες και αν είναι επιστρέφεται το level. Αν δεν ισχύει καμία από τις προηγούμενες συνθήκες αυξάνεται το level κατά 1 και γίνεται αναδρομή με τον ίδιο τρόπο με το *insert_recursive* (υπολογισμός κέντρων και nw, sw κτλ) και γίνεται return η depth_search με το σωστό sub-leaf και τις ανανεωμένες τιμές των x, y , level.

Περιγραφή της class RandomPairs

Η κλάση αυτή δημιουργεί τα σύνολα τον μοναδικών κλειδιών με βάση το πλήθος των ζευγών και επιπλέον δημιουργεί τα σύνολα των αναζητήσεων, τα 100 existing μοναδικά ζεύγη και τα 100 non existing μοναδικά ζεύγη. Η κλάση όταν δημιουργείται ένα στιγμιότυπό της αρχικοποιεί ένα self.pairs = set(), δηλαδή ορίζει ένα αρχικό set. **Τα python sets είναι σύνολα που χειρίζονται dynamically, δεν μας ενδιαφέρει η σειρά με την οποία κατατάσσονται τα στοιχεία και κυρίως κάθε στοιχείο είναι μοναδικό και δεν φροντίζουν να μην έχουν duplicates**.

def random_pairs(self, m, n):

Τρέχει με μια while όσο το μέγεθος του αρχικού set είναι μικρότερο από m και προσθέτει στο αρχικό set 2 random x, y (0 έως n-1).

def existing_pairs(self, n):

Ορίζει ένα exist = set(), τρέχει με μια while όσο το μέγεθος του exist είναι μικρότερο από 100 και προσθέτει στο exist τυχαία στοιχεία από το αρχικό set (self.pairs).

def non_existing_pairs(self, n):

Ορίζει ένα non_exist = set(), τρέχει με μια while όσο το μέγεθος του non_exist είναι μικρότερο από 100 και φτιάχνει ένα temp όπου έχει μόνο ένα ζεύγος random x, y και το non_exist θα γίνει update και η νέα τιμή του που θα εισαχθεί θα είναι η διαφορά του temp set από το αρχικό set. Επομένως αν το ζεύγος random x, y του temp set δεν υπάρχει στο αρχικό set θα προστεθεί στο non_exist set.

Η συνάρτηση main είναι η εκτελέσιμη συνάρτηση του προγράμματος και δημιουργεί τα sets, εκτελεί τις συναρτήσεις insert και search των 2 δομών και τυπώνει τα τελικά αποτελέσματα.