ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ ΙΑΝΟΥΑΡΙΟΣ 2019 ΕΡΓΑΣΙΑ 3

ΚΩΤΣΗΣ-ΠΑΝΑΚΑΚΗΣ ΒΑΣΙΛΕΙΟΣ-ΕΚΤΩΡ (Ρ3180094)

ΓΕΩΡΓΙΟΥ ΑΛΕΞΙΟΣ - ΛΑΖΑΡΟΣ (Ρ3180027)

Μέρος Α (Υλοποίηση κλάσης Rectangle)

Μέθοδος contains(Point p): η μέθοδος contains() έχει αρκετά απλή υλοποίηση. Εάν η συντεταγμένη χ του p είναι μικρότερη απο το xmax και μεγαλύτερη του xmin, και η συντεταγμένη y του p είναι μικρότερη απο το ymax και μεγαλύτερη του ymin του ορθογονίου, τότε αυτό σημαίνει πως το p βρίσκεται μέσα στο παραλληλόγραμμο, επομένως επιστρέφει true. Αλλιώς, επιστρέφει false.

Μέθοδος intersects(Rectangle r): Η μέθοδος intersects() με απλά λόγια κοιτάει αν το παραλληλόγραμμο r είναι έξω απο το παραλληλόγραμμο το οποίο την καλεί(Εστω Π). Αυτό γίνεται με την εξής λογική. Αν το ymin του r είναι μεγαλύτερο απο το ymax του Π ή το ymax του r μικρότερο απο το ymin του Π ή το xmin του r μεγαλύτερο απο το xmax του Π ή το xmax του r μικρότερο απο το xmin του Π, τότε επιστρέφει false. Αλλιώς, επιστρέφει true.

Μέθοδος distanceTo(Point p): Η μέθοδος distanceTo() λειτουργεί ελέγχοντας που "βρίσκεται" το p σε σχέση με το παραλληλόγραμμο (έστω Π). Μολις το βρεί, υπολογίζει την απόσταση του p από το κοντινότερο σημείο πάνω στο παραλληλόγραμμο, με την μέθοδο distanceTo(Point p) της κλάσης Point. Αυτή η απόσταση είναι η ελάχιστη απόσταση του p απο το παραλληλόγραμμο. Ας δούμε τις περιπτώσεις.

1. Αν το p είναι πάνω αριστερά του Π, τότε η ελάχιστή του απόσταση απο το παραλληλόγραμμο είναι η απόστασή του με το point **(xmin, ymax).**
2. Αν το p είναι πάνω δεξιά του Π, τότε η ελάχιστή του απόσταση απο το παραλληλόγραμμο είναι η απόστασή του με το point **(xmax, ymax).**
3. Αν το p είναι κάτω αριστερά, τότε η ελάχιστή του απόσταση απο το παραλληλόγραμμο είναι η απόστασή του με το point **(xmin, ymin).**
4. Αν το p είναι κάτω δεξιά του Π, τότε η ελάχιστή του απόσταση απο το παραλληλόγραμμο είναι η απόστασή του με το point **(xmax, ymin).**
5. Αν το p είναι πάνω του Π, τότε η απόσταση είναι η διαφορά της συντεταγμένης y του p με την ymax (απόλυτο)
6. Αν το p είναι αριστερά του Π, τότε η απόσταση είναι η διαφορά της συνταταγμένης x του p με την xmin (απόλυτο).
7. Αν το p είναι δεξιά του Π, τότε η απόσταση είναι η διαφορά της συντεταγμένης x του p με την xmax (απόλυτο).
8. Αν το p είναι κάτω του Π, τότε η απόσταση είναι η διαφορά της συντεταγμένης y του p, με την ymin (απόλυτο).
9. Σε άλλη περίπτωση, η απόσταση είναι 0, γιατί το p είναι είτε μέσα, είτε επάνω στο Π.

Μέρος Β (Υλοποίηση μεθόδων rangeSearch και nearestNeighbor)

Προτού εξηγήσουμε την υλοποίηση των μεθόδων μας, να τονιστεί πως στην TreeNode προσθέσαμε και ένα επιπλέον πεδίο **Rectangle rect**, το οποίο αντιστοιχεί στο νοητό παραλληλόγραμμο ενός point στο TwoDTree. Αυτό το παραλληλόγραμμο αρχικοποιείται όταν καλείται η **insert(Point p)** της TwoDTree, της οποίας η δουλειά είναι να εκχωρήσει στο δυαδικό δέντρο τον κόμβο με το p. Οι διαστάσεις του παραλληλογράμμου rect δημιουργούνται την στιγμή που κάνουμε κατάδυση στο δέντρο. Εάν συγκρίνουμε τις x συντεταγμένες κατα την κατάδυση, και δούμε πως ο p πρέπει να πάει στο αριστερό παιδί του τρέχοντος κόμβου, τότε στο xmax εκχωρείται η συντεταγμένη x του τρέχοντος κόμβου (Μείωση του μεγέθους του παραλληλογράμμου απο τα δεξιά), αλλιώς εκχωρείται στο xmin (Μείωση του μεγέθους του παραλληλογράμμου απο τα αριστερά). Την ίδια ακριβώς λογική ακολουθούμε και αν συγκρίνουμε τις y συντεταγμένες των Point (Αντίστοιχη μείωση του μεγέθους του παραλληλογράμμου απο τα πάνω και μείωση του μεγέθους του παραλληλογράμμου απο τα κάτω). Έτσι, κάθε κόμβος TreeNode θα έχει τον Point που θέλουμε, μαζί και με το νοητό παραλληλόγραμμο που φτιάξαμε κατα την διάρκεια της εκχώρησης στο δέντρο

Μέθοδος rangeSearch(Rectangle r): Η μέθοδος rangeSearch() χρησιμοποιεί την αναδρομική μέθοδο **rangeSearchR()**, η οποία έχει 3 ορίσματα: τον τρέχον κόμβο **current**, το **query Rectangle r**, και την λίστα **List<Point> points**, η οποία θα είναι και αυτή που θα τροποποιηθεί κατα την διάρκεια της εκτέλεσης της rangeSearchR(). Ας εξηγήσουμε την λογική. Στην αρχή της rangeSearch(), αρχικοποιούμε την άδεια λίστα points. Ύστερα, καλούμε για πρώτη φορά την rangeSearchR() με αρχικό κόμβο τον head του TwoDTree. Η rangesearchR() με την βοήθεια της **intersects()** αρχικά κοιτάζει εάν υπάρχουν κοινά σημεία μεταξύ του r και του νοητού παραλληλόγραμμου του τρέχοντος point. Αν υπάρχουν, τότε με την **contains()** ελέγχει να δει αν το r περιέχει το τρέχον point, και αν το περιέχει, το βάζει στην λίστα points με την **insertAtFront()**. Στη συνέχεια, αν υπάρχει αριστερό παιδί στον κόμβο, τότε καλείται αναδρομικά η rangeSearchR για το αριστερό παιδί. Το ίδιο και στο δεξί παιδί. Αυτό συμβαίνει μέχρις ότου να μην μπορούμε να προσπελάσουμε άλλους κόμβους στο δέντρο, είτε επειδή φτάσαμε στα φύλλα του δέντρου, είτε επειδή δεν υπάρχουν κόμβοι με τους οποίους το rec να εχει σημεία τομής με το παραλληλόγραμμό τους. Με αυτόν τον τρόπο ελέγχουμε μόνο όσα υποδέντρα μπορούν πιθανά να έχουν σημεία που να περιέχονται στο r. Τέλος, αφότου βγει απο την αναδρομή, επιστρέφει την λίστα **points**.

Μέθοδος nearestNeighbor(Point p): Η μέθοδος nearestNeighbor() χρησιμοποιεί την αναδρομική μέθοδο **nearestNeighborSearchR()**, η οποία έχει 4 ορίσματα: τον τρέχον κόμβο **current,** το **query Point p**, μια array **Point[] minDistancePoint** η οποία **<u>πάντοτε</u>** είναι κατα σύμβαση μεγέθους 1 (γιατί μας επιτρέπει να έχουμε το πιο κοντινό ως τότε point), και μια **boolean compareX**, της οποίας η τιμή μας βοηθά να ξεχωρίσουμε το πότε θα συγκρίνουμε x ή y συντεταγμένες στα Point κατα την διάρκεια της κατάδυσης. Ας εξηγήσουμε την λογική. Στην αρχή της nearestNeighbor, (Αν φυσικά δεν είναι κενή, οπου επιστρέφει null) Αρχικοποιούμε την μόνη θέση της minDistancePoint με το item που βρίσκεται στην ρίζα του δέντρου. Ύστερα, καλούμε την nearestNeighborSearchR() με

αρχικό κόμβο την ρίζα (head) και με **xCompare = true**. Στην αρχή, κοιτάει (με την βοήθεια της **squareDistanceTo()**, μέθοδος της Point) εάν το τωρινό ελάχιστης απόστασης σημείο έχει μεγαλύτερη απόσταση άπο το p απο την απόσταση του p με το Point του τρέχοντος κόμβου και αν είναι, το τρέχον Point καταχωρείται στην minDistancePoint[0] ως το πιο κοντινό σημείο στο p. Στη συνέχεια, αυτό που κάνουμε είναι να κατευθυνθούμε σε ένα απο τα φύλλα του δέντρου, το οποίο είναι το φύλλο του οποίου θα ήταν παιδί το p, εάν το κάναμε insert() στο δέντρο. Με αυτόν τον τρόπο, θα έχουμε ένα πολύ κοντινό σημείο στο p, και θα το χρησιμοποιήσουμε ώστε να αποφύγουμε την προσπέλαση άχρηστων υποδέντρων. Φυσικά, καθώς κάνουμε προσπέλαση του κάθε κόμβου, μετά την αναδρομική κλήση της nearestNeighborSearchR() για τον επόμενο κόμβο που επιλέγεται με βάση την σύγκριση των συντεταγμένων, κάνουμε έναν έλεγχο για το εαν το άλλο παιδί που δεν επιλέχθηκε (δεξί ή αριστερό) υπάρχει. Εαν υπάρχει, τότε έλεγχουμε εάν η απόσταση του τωρινού ελαχίστου σημείου απο το p είναι μεγαλύτερη ή ίση απο την απόσταση του p απο το Rectangle που αντιστοιχεί στο Point εκείνου του παιδιού, και αν είναι, τότε καλούμε την nearestNeighborSearchR() αναδρομικά και για εκείνον τον κόμβο. Με αυτόν τον τρόπο αποφεύγουμε τον έλεγχο ολόκληρων υποδέντρων, γιατί δεν μπορούν να υπάρχουν σημεία που να απέχουν λιγότερο απο τον εκάστοτε minDistancePoint αν δεν ικανοποιείται η παραπάνω συνθήκη. Η διαδικασία τερματίζεται όταν δεν υπάρχουν άλλοι κόμβοι για να προσπελαστούν (είτε λόγω της παραπάνω συνθήκης, είτε επειδή δεν υπάρχουν). Τέλος, μόλις έχουν τελειώσει όλες οι αναδρομικές κλήσεις, επιστρέφεται το τελικό ελάχιστης απόστασης σημείο απο το p **minDistancePoint[0].**

Όπως αναφέρθηκε προηγουμένως, το κύριο πλεονέκτημα αυτών των δύο μεθόδων, είναι πως βοηθάνε πολύ στην επιτάχυνση της ανάκτησης του επιθυμητού αποτελέσματος, καθώς δεν είναι πια αναγκαστικό να προσπελαστούν όλοι οι κόμβοι όλου του δέντρου.