ΕΡΓΑΣΙΑ 3

Μαρία Σχοινάκη

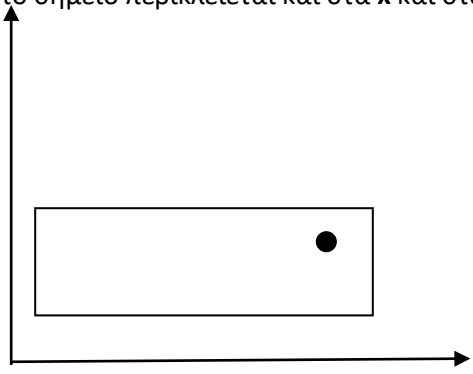
Μέρος Α - ΑΤΔ γεωμετρικών αντικειμένων

ΑΡΧΕΙΑ

Point.java, Rectangle.java

Στην εργασία ξεκινήσαμε υλοποιώντας την κλάση **Point**. Η κλάση **Point** ορίζει στην ουσία αντικείμενα που παριστάνουν σημεία στο επίπεδο(*συντεταγμένες*). Ξεκινήσαμε ορίζοντας τις μεταβλητές **x** και **y** private(*για να μην μπορούν κλάσεις εκτός αρχείου να έχουν πρόσβαση*), οι οποίες αργότερα στον κατασκευαστή παίρνουν τις τιμές τους. Έπειτα υλοποιήσαμε τις μεθόδους **x()**, **y()**, για να μπορούμε να διαβάζουμε τις συντεταγμένες χωρίς να έχουμε πρόσβαση ή να μπορούμε να αλλάξουμε τις μεταβλητές **x**, **y**. Στην συνέχεια, υλοποιήσαμε την μέθοδο **distanceTo(Point z)**, η οποία επιστρέφει την απόσταση 2 σημείων στο επίπεδο. Συγκεκριμένα, την απόσταση του τρέχοντος αντικειμένου, με το αντικείμενο **z** που δίνεται σαν όρισμα. Κατά τα γνωστά μας από τα μαθηματικά, ο τύπος είναι: **√((x2 – x1)² + (y2 – y1)²)**. Στον τύπο αυτό μας βοήθησε η βιβλιοθήκη **Math** της **java** από την οποία και πήραμε τις συναρτήσεις **sqrt(x)** και **pow(x, y)**. Αξίζει να σημειωθεί, ότι οι προαναφερόμενες συναρτήσεις επιστρέφουν **double** τιμές, όπως είναι και προφανές, γιαυτό και η μέθοδος μας είναι τύπου **double**. Έπειτα υλοποιήσαμε την μέθοδο **squareDistanceTo(Point z)**, η οποία σε ακριβώς παρόμοια λογική με την **distanceTo(Point z)**, επιστρέφει τον τύπο που προαναφέρθηκε χωρίς την τετραγωνική ρίζα(*για καλύτερους υπολογισμούς*) και άρα είναι τύπου **int**. Τέλος, υλοποιήσαμε την μέθοδο **toString()**, η οποία επιστρέφει ένα αντικείμενο τύπου **String**.
Π.χ (20, 30). Στο αμέσως επόμενο κομμάτι της εργασίας μας προχωρήσαμε στην υλοποίηση της κλάσης **Rectangle.** Η κλάση **Rectangle**, ορίζει στην ουσία αντικείμενα που παριστάνουν ορθογώνια παραλληλόγραμμα στο επίπεδο. Το παραλληλόγραμμο απαρτίζεται από 4 γωνίες οι οποίες είναι αντικείμενα τύπου **Point**. Με την ίδια λογική όπως και με την **Point** ορίζουμε ως **private** 4 μεταβλητές που ο κατάλληλος συνδυασμός τους απαρτίζει τα σημεία στον χώρο που αποτελούν γωνίες του παραλληλογράμμου. Στον κατασκευαστή οι 4 μεταβλητές παίρνουν τιμή. Έπειτα, ορίσαμε την μέθοδο **contains(Point p)** η οποία επιστρέφει **true** αν το σημείο που δίνεται σαν όρισμα περιέχεται στο τρέχον παραλληλόγραμμο, αλλιώς επιστρέφει **false**. Ένα σημείο περιέχεται σε ένα παραλληλόγραμμο αν και μόνο αν η διάσταση **x** του σημείου "περικλείεται" στα **x** του παραλληλογράμμου και αν η διάσταση **y** "περικλείεται" στα **y** του παραλληλογράμμου.

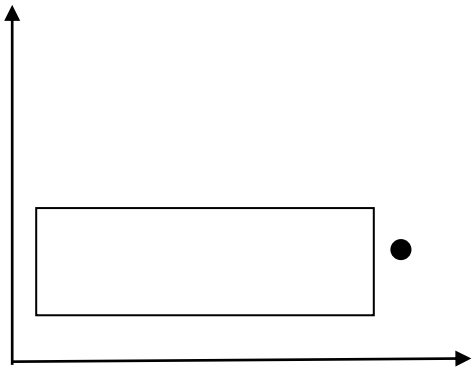
Όπως παρατηρούμε το σημείο περικλείεται και στα **x** και στα **y** του ορθογωνίου.



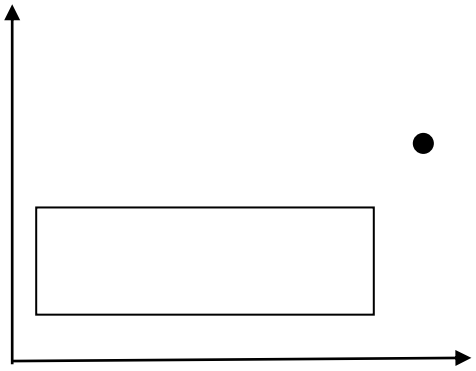
ΕΡΓΑΣΙΑ 3

Μαρία Σχοινάκη

Ενώ στο παρακάτω παράδειγμα όπως βλέπουμε ενώ το **y** περικλείεται στα y του παραλληλογράμμου, το **x** βγαίνει εκτός ορίων.



Βέβαια υπάρχουν πολλές περιπτώσεις για τις οποίες η **contains** θα βγει **false**, όπως π.χ. ούτε το **x** ούτε το **y** του σημείου να περικλείονται στα αντίστοιχα **x, y** του παραλληλογράμμου.

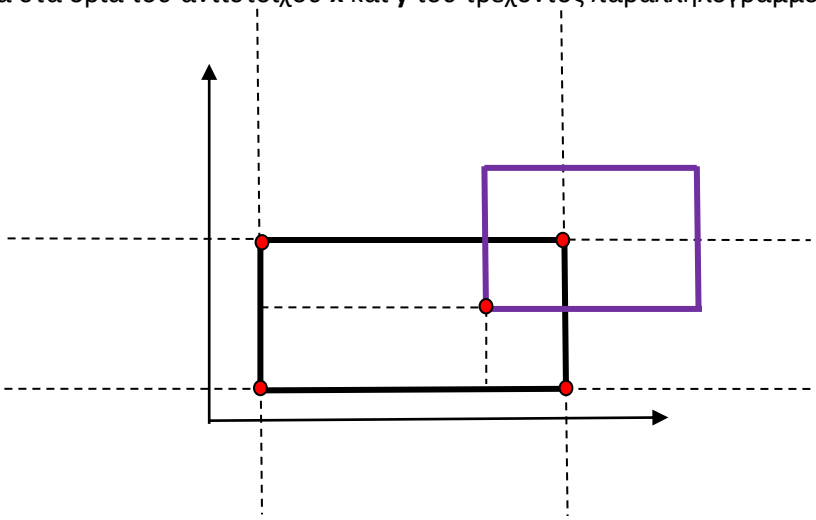


Αμέσως μετά υλοποιήσαμε την μέθοδο **intersects(Rectangle that)**, η οποία δέχεται σαν όρισμα ένα αντικείμενο τύπου **Rectangle** και συγκρίνει αν αυτό έχει ένα τουλάχιστον κοινό σημείο με το τρέχον παραλληλόγραμμο. Αν έχει γυρίζει **true**, αλλιώς γυρίζει **false**. Οπότε αρκεί απλά να αποδείξουμε ότι υπάρχει κενό μεταξύ των διαστάσεων των **2** παραλληλογράμμων **x** ή **y** για να δείξουμε ότι τα **2** παραλληλόγραμμα δεν έχουν κοινά σημεία . Δηλαδή αν η μέγιστη διάσταση του ενός είναι μικρότερη από την ελάχιστη διάσταση του άλλου ή και το αντίστροφο(*η ελάχιστη διάσταση του ενός να είναι μεγαλύτερη από την μέγιστη διάσταση του άλλου*), για οποιαδήποτε από τις **2** διαστάσεις **x**, **y**, τότε τα **2** παραλληλόγραμμα είναι ασυσχέτιστα. Πρακτικά βρήκαμε την συνθήκη αποσυσχέτισης των **2** παραλληλογράμμων, η οποία αν ισχύει επιστρέφει **false**, ενώ σε κάθε άλλη περίπτωση επιστρέφει **true**.

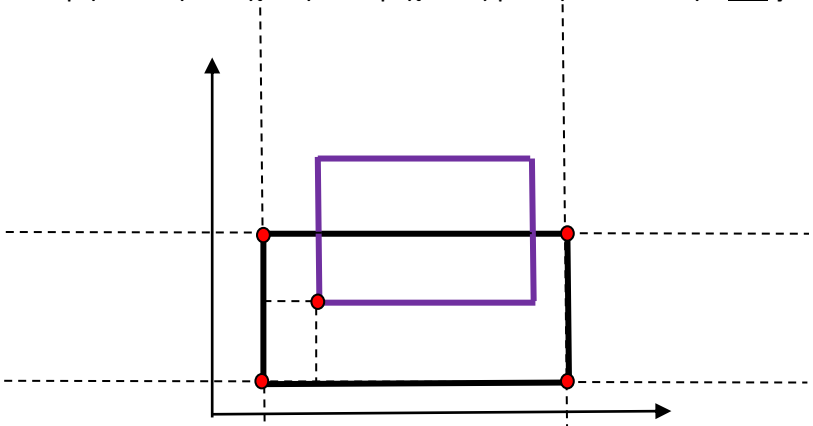
ΕΡΓΑΣΙΑ 3

Μαρία Σχοινάκη

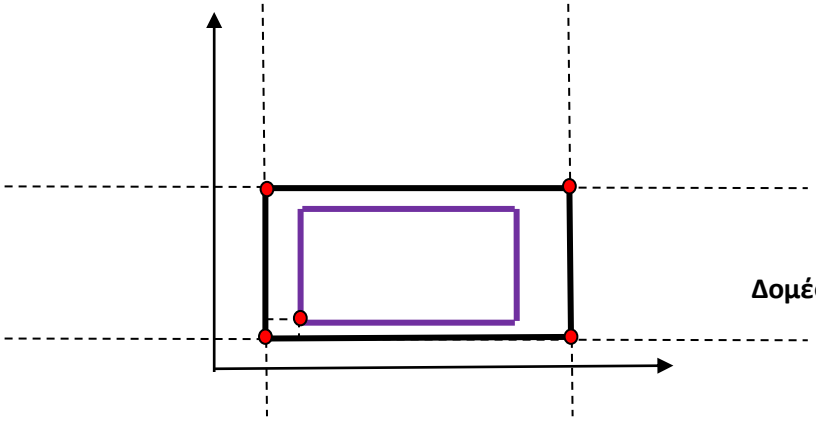
Παρακάτω, η ελάχιστη διάσταση του **x** και του **y** του δοθέντος παραλληλογράμμου, περιέχεται μέσα στα όρια του αντίστοιχου **x** και **y** του τρέχοντος παραλληλογράμμου.



Ακριβώς ανάλογα και στο παρακάτω παράδειγμα τα **2** παραλληλόγραμμα έχουν ΤΟΥΛΑΧΙΣΤΟΝ **1** κοινό σημείο. Όπως επαληθεύεται, η ελάχιστη διάσταση του δοθέντος παραλληλογράμμου είναι μικρότερη από την μέγιστη διάσταση του τρέχοντος και η μέγιστη του δοθέντος είναι μεγαλύτερη από την ελάχιστη του τρέχοντος για την διάσταση **x** ΚΑΙ **y** αντίστοιχα.



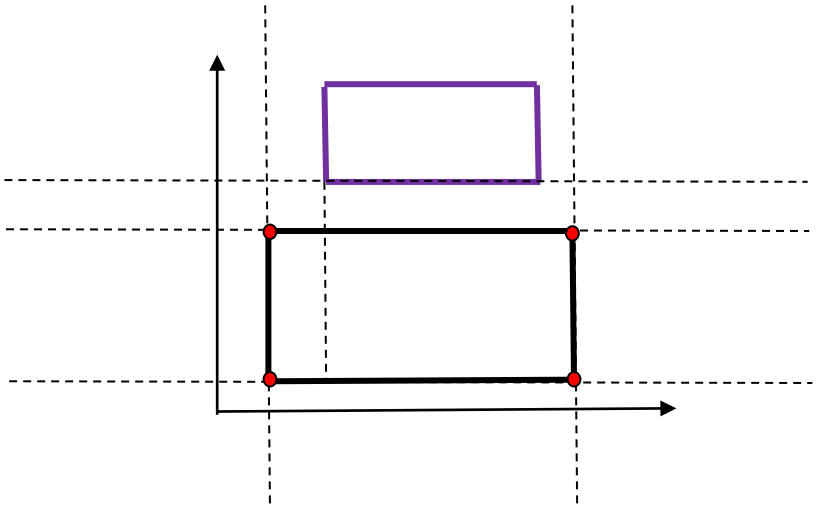
Όπως και παραπάνω, αν το ένα παραλληλόγραμμο περιέχεται στο άλλο, τότε η μέθοδος επιστρέφει **true**. Αξίζει να σημειωθεί ότι έχουν κοινά σημεία τα σημεία του δοθέντος.



ΕΡΓΑΣΙΑ 3

Μαρία Σχοινάκη

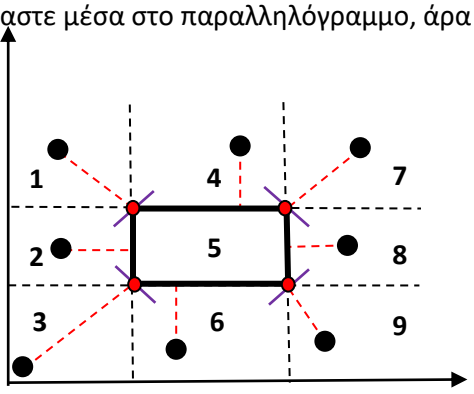
Τέλος, προφανώς αν τα **2** παραλληλόγραμμα είναι άσχετα μεταξύ τους, τότε η ελάχιστη διάσταση του δοθέντος παραλληλογράμμου, θα είναι μεγαλύτερη από την μέγιστη του τρέχοντος παραλληλογράμμου Ή η μέγιστη διάσταση του δοθέντος θα είναι μικρότερη από την ελάχιστη διάσταση του τρέχοντος για τις διαστάσεις **x** Ή **y** αντίστοιχα(*δηλαδή 4 συνθήκες, 2 για κάθε διάσταση*).



Αμέσως μετά υλοποιήσαμε τις μεθόδους **distanceTo(Point p)** και **squareDistanceTo(Point p)**. Η **distanceTo(Point p)**, βρίσκει τον οριοθετημένο χώρο στον οποίο βρισκόμαστε και ανάλογα επιστρέφει την τιμή της απόστασης.

Στους χώρους **1**, **3**, **7**, **9** το κοντινότερο σημείο του p στο παραλληλόγραμμο, είναι η συγκεκριμένη γωνία, άρα καλούμε την **distanceTo** της **Point**. Στους χώρους **2**, **4**, **6**, **8** το κοντινότερο σημείο του **p** στο παραλληλόγραμμο είναι η μεσοκάθετος από το σημείο, στην ευθεία της συγκεκριμένης πλευράς. Συγκεκριμένα, στους χώρους **2**, **8** είναι **|χτρεχ – χδοθ|**, ενώ στους **4**, **6**, **|υτρεχ – υδοθ|**.

Στον χώρο **5** βρισκόμαστε μέσα στο παραλληλόγραμμο, άρα επιστρέφουμε **0**.



Κάτι εντελώς ανάλογο συμβαίνει και στην **squareDistanceTo(Point p)**, μόνο που επιστρέφουμε στους χώρους **1**, **3**, **7**, **9** την **squareDistanceTo(Point p)** της **Point**, ενώ στους χώρους **2**, **4**, **6**, **8** **|χτρεχ – χδοθ|^2** και **|γτρεχ – γδοθ|^2** αντίστοιχα. Τέλος, υλοποιήσαμε την μέθοδο **toString()**, η οποία επιστρέφει ένα αντικείμενο τύπου **String** όπως στην κλάση **Point**, απλά τώρα στην **Rectangle** επιστρέφει τον οριοθετημένο χώρο του παραλληλογράμμου. *Π.χ [30, 40] X [10, 20]*

Αξίζει να σημειωθεί ότι η εργασία βασίζεται στο ανώτερο σύνολο σημείων ***[0, 100] X [0, 100]***, οπότε κάθε νέο παραλληλόγραμμο ή σημείο πρέπει να οριοθετείται και να είναι γνήσιο υποσύνολο του προαναφερθέντος συνόλου.

Μέρος Β - ΑΤΔ για 2d-trees

ΑΡΧΕΙΑ

TwoDTree.java, TreeNode.java, List.java, Node.java

Σε συνέχεια της εργασίας, υλοποιήσαμε την κλάση **TwoDTree**. Αφότου ορίσαμε της βασικές μεθόδους, ορίσαμε και την **nearestNeighbor(Point p)**. Η μέθοδος αυτή, δέχεται σαν όρισμα ένα αντικείμενο τύπου **Point** και επιστρέφει το κοντινότερο **Point** από το τρέχον δέντρο που έχουμε. Ξεκινήσαμε ορίζοντας **2** μεθόδους με ίδιο όνομα, αλλά διαφορετική υπογραφή. Στην **nearestNeighbor(Point p)**, ελέγξαμε αρχικά αν το δέντρο είναι άδειο(*αν ναι επιστρέφουμε null*). Στην συνέχεια, ορίσαμε ως κοντινότερο σημείο, το σημείο που αντιστοιχεί στον κόμβο της ρίζας, δηλαδή του $1^{ου}$ κόμβου και καλέσαμε την $2^{η}$ μέθοδο, **nearestNeighbor(TreeNode current,Point p,Point currclospoint,int count,int xmin,int xmax,int ymin ,int ymax)** όπου θα είναι στην ουσία η αναδρομική μας μέθοδος. Τέλος, επιστρέψαμε το αποτέλεσμα της αναδρομικής μεθόδου. Η αναδρομική μέθοδος καλείται την $1^{η}$ φορά με τα δεδομένα της ρίζας και μετά κάθε φορά με τα δεδομένα των κατάλληλων κόμβων.

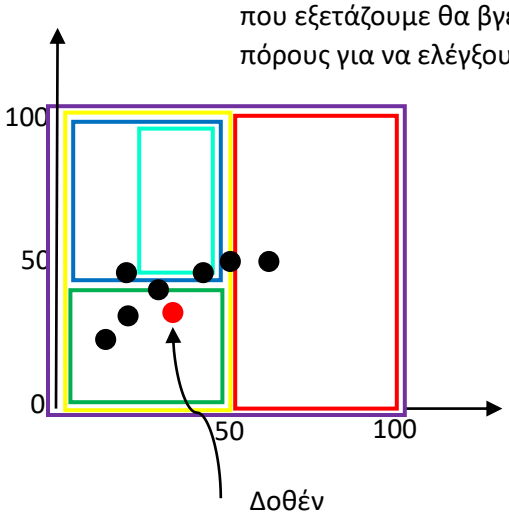
TreeNode current: Το **TreeNode** στο οποίο βρισκόμαστε
Point p: Το **Point** που δίνει ο χρήστης
Point currclospoint: Το κοντινότερο **Point** στο **p** που έχουμε βρει μέχρι στιγμής
int count: Μετρητής που καθορίζει αν συγκρίνουμε με το **x** ή με το **y**
int xmin-xmax-ymin-ymax : Οι **4** συντεταγμένες του παραλληλογράμμου του κόμβου που βρίσκεται ο **current** κόμβος

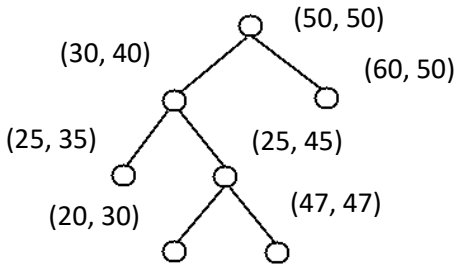
Αν ο τρέχον κόμβος που εξετάζουμε είναι **null**, τότε επιστρέφουμε το κοντινότερο σημείο που έχουμε βρει μέχρι στιγμής. Αμέσως μετά ελέγξαμε αν η απόσταση του **p** από το κοντινότερο σημείο μέχρι στιγμής, είναι μικρότερη από την απόσταση του **p** από το ορθογώνιο του κόμβου που εξετάζουμε. Αν είναι τότε δεν έχει νόημα να ασχοληθούμε με

ΕΡΓΑΣΙΑ 3

Μαρία Σχοινάκη

τον συγκεκριμένο κόμβο και το υποδέντρο του. Αν η απόσταση του **p** από το **Point** του **current** είναι μικρότερη από την απόσταση του **p** από το κοντινότερο **Point** στο **p** που έχουμε βρει μέχρι στιγμής τότε πλέον το κοντινότερο σημείο αλλάζει στο **Point** του **current**. Μετέπειτα ελέγξαμε με μια συνθήκη αν βρισκόμαστε σε μονό ή ζυγό επίπεδο για να γνωρίζουμε αν πρέπει να εξετάσουμε την διάσταση **x** ή **y**. Ανάλογα την διάσταση(*x ή y*) συγκρίνουμε αν η διάσταση του σημείου του ορίσματος είναι μικρότερη από την διάσταση του τρέχοντα κόμβου. Αν είναι, τότε καλούμε την αναδρομική μέθοδο για το αριστερό παιδί με τα κατάλληλα δεδομένα και αμέσως μετά το δεξί. Αν η διάσταση του σημείου του ορίσματος είναι μεγαλύτερη από την διάσταση του τρέχοντα κόμβου, τότε καλούμε την αναδρομική μέθοδο για το δεξί παιδί με τα κατάλληλα δεδομένα και μετά για το αριστερό. Αυτό το κάνουμε για την διάσταση **x** στα ζυγά επίπεδα, ενώ για την διάσταση **y** για τα μονά επίπεδα. Επίσης έχουμε ορίσει έναν μετρητή επιπέδων, που κάθε φορά που καλούμε την αναδρομική μέθοδο, αυξάνεται κατά **ένα**, για να δείξουμε ότι προχωρήσαμε ένα επίπεδο. Ο λόγος που συγκρίνουμε την διάσταση του ορίσματος με την διάσταση του τρέχοντος κόμβου, είναι γιατί αν η πρώτη είναι μικρότερη από την δεύτερη, τότε είναι πιο πιθανό η λύση που ψάχνουμε να βρίσκεται στο αριστερό υποδέντρο, γιαυτό ελέγχουμε πρώτα αυτό, έτσι ώστε να γλιτώσουμε αναδρομές. Αν τώρα είμαστε σε ζυγό επίπεδο και καλούμε αναδρομικά το αριστερό παιδί τότε εκτός του προφανούς ότι για **current** κόμβο στέλνουμε το αριστερό παιδί, δίνουμε για **xmax** το **x** του πατέρα ενώ τα **xmax**, **ymin**, **ymax** παραμένουν ίδια με του γονέα. Αν καλούμε το δεξί παιδί, δίνουμε για **xmin** το **x** του πατέρα ενώ τα **xmin**, **ymin**, **ymax** παραμένουν ίδια με του γονέα. Αν τώρα βρισκόμαστε σε μονό επίπεδο και καλούμε αναδρομικά το αριστερό παιδί τότε εκτός του προφανούς ότι για **current** κόμβο στέλνουμε το αριστερό παιδί, δίνουμε για **ymax** το **y** του πατέρα ενώ τα **xmin**, **xmax**, **ymin** παραμένουν ίδια με του γονέα. Αν καλούμε το δεξί παιδί, δίνουμε για **ymin** το **y** του πατέρα ενώ τα **xmin**, **xmax**, **ymax** παραμένουν ίδια με του γονέα. Η βασική ιδέα πίσω από αυτή τη μέθοδο είναι ότι ξεκινώντας από την ρίζα και ελέγχοντας το επίπεδο και το αν η διάσταση του σημείου του ορίσματος είναι μικρότερη ή μεγαλύτερη από την διάσταση του τρέχοντα κόμβου, καλούμε την αναδρομική μέθοδο για το κατάλληλο παιδί, έτσι ώστε να βρούμε όσο πιο γρήγορα γίνεται το πιο κοντινό σημείο του **p**. Δεδομένου ότι βρίσκουμε το πιο κοντινό σημείο, και είναι σε ένα αριστερό φύλλο, τότε πηγαίνοντας μετά να ψάξουμε τον αδερφό του δεξιά, η συνθήκη για το αν η απόσταση του **p** από το κοντινότερο σημείο μέχρι στιγμής, είναι μικρότερη από την απόσταση του p από το ορθογώνιο του κόμβου που εξετάζουμε θα βγει αληθής, οπότε δεν θα χάσουμε χρόνο, χώρο και υπολογιστικούς πόρους για να ελέγξουμε αυτό τον αδερφό και το υποδέντρο του.





ΕΡΓΑΣΙΑ 3

Μαρία Σχοινάκη

Σειρά έχει, η υλοποίηση της μεθόδου **rangeSearch(Rectangle rect)**. Η μέθοδος αυτή δέχεται ένα αντικείμενο τύπου **Rectangle** και επιστρέφει μία λίστα με τα σημεία του δέντρου που περιέχονται στο ορθογώνιο αυτό. Η λίστα προφανώς έχει στοιχεία τύπου **Point** και αν το δέντρο είναι άδειο, επιστρέφεται μια άδεια λίστα. Η λογική είναι ίδια με της **nearestNeighbor(Point p)**. Υλοποιούμε πρώτα την **rangeSearch(Rectangle rect)**, στην οποία αρχικοποιούμε την λίστα και ελέγχουμε αν το δέντρο είναι άδειο. Αν είναι τότε επιστρέφουμε

την κενή λίστα. Έπειτα, καλούμε την αναδρομική μέθοδο **rangeSearch(Rectangle rect,TreeNode current,int count,int xmin,int xmax,int ymin ,int ymax,List<Point> list)**, η οποία έχει ίδιο όνομα με την πρώτη, αλλά διαφορετική υπογραφή. Τέλος, επιστρέφουμε το αποτέλεσμα της αναδρομικής μεθόδου. Η αναδρομική μέθοδος τώρα, έχει ορίσματα:

Rectangle rect: Το **rectangle** που δίνει ο χρήστης
TreeNode current: Το **TreeNode** στο οποίο βρισκόμαστε τώρα
int count: Μετρητής **count** για να γνωρίζει αν πρέπει να συγκρίνει με βάση το **x** ή το **y**
int xmin-xmax-ymin-ymax: Οι **4** συντεταγμένες του **rectangle** στο οποίο βρίσκεται το **current**
List<Point> list: Η λίστα στην οποία προσθέτουμε τα **Point**

Η αναδρομική μέθοδος καλείται πρώτη φορά για τα δεδομένα του 1ου κόμβου, δηλαδή της ρίζας. Μέσα στην μέθοδο, εξετάζουμε αν ο τρέχον κόμβος είναι null και αν είναι επιστρέφουμε την λίστα που έχουμε μέχρι στιγμής. Έπειτα, ελέγχουμε αν το παραλληλόγραμμο που έδωσε ο χρήστης δεν έχει κανένα κοινό σημείο με το παραλληλόγραμμο στο οποίο αντιστοιχεί ο τρέχον κόμβος, γιατί αν δεν έχει τότε δεν χρειάζεται να ελέγξουμε αυτόν τον κόμβο και τα υποδέντρα του. Μετά ελέγχουμε αν το παραλληλόγραμμο που δόθηκε περιέχει τον συγκεκριμένο κόμβο, για να βάλουμε το **Point** του στην λίστα. Μετέπειτα ελέγξαμε με μια συνθήκη αν βρισκόμαστε σε μονό ή ζυγό επίπεδο για να γνωρίζουμε αν πρέπει να εξετάσουμε την διάσταση **x** ή **y**. Επίσης έχουμε ορίσει έναν μετρητή επιπέδων, που κάθε φορά που καλούμε την αναδρομική μέθοδο, αυξάνεται κατά ένα, για να δείξουμε ότι προχωρήσαμε ένα επίπεδο. Αν τώρα είμαστε σε ζυγό επίπεδο και καλούμε αναδρομικά το αριστερό παιδί τότε εκτός του προφανούς ότι για **current** κόμβο στέλνουμε το αριστερό παιδί, δίνουμε για **xmax** το **x** του πατέρα ενώ τα **xmax**, **ymin**, **ymax** παραμένουν ίδια με του γονέα. Αν καλούμε το δεξί παιδί, δίνουμε για **xmin** το **x** του πατέρα ενώ τα **xmin**, **ymin**, **ymax** παραμένουν ίδια με του γονέα. Αν τώρα βρισκόμαστε σε μονό επίπεδο και καλούμε αναδρομικά το αριστερό παιδί τότε εκτός του προφανούς ότι για **current** κόμβο στέλνουμε το αριστερό παιδί, δίνουμε για **ymax** το **y** του πατέρα ενώ τα **xmin**, **xmax**, **ymin** παραμένουν ίδια με του γονέα. Αν καλούμε το δεξί παιδί, δίνουμε για **ymin** το **y** του πατέρα ενώ τα **xmin**, **xmax**, **ymax** παραμένουν ίδια με του γονέα. Η βασική ιδέα πίσω από την υλοποίηση της μεθόδου είναι, ξεκινώντας από την ρίζα και χρησιμοποιώντας προδιατεταγμένη διάσχιση(*Αν το επιτρέπει η συνθήκη για την μείωση των αναδρομών*), να ελέγχουμε αν κάποιος κόμβος περιέχεται στο παραλληλόγραμμο. Αν το παραλληλόγραμμο που έδωσε ο χρήστης δεν έχει κανένα κοινό σημείο με το παραλληλόγραμμο στο οποίο αντιστοιχεί

ΕΡΓΑΣΙΑ 3

Μαρία Σχοινάκη

ο τρέχον κόμβος δεν χρειάζεται να ελέγξουμε αυτόν τον κόμβο και τα υποδέντρα του. Αυτό χρησιμοποιείται ώστε να μην είναι αναγκαία η διάσχιση όλου του δέντρου για να επιστραφεί η λίστα που χρειαζόμαστε. Αξίζει να σημειωθεί πως για την υλοποίηση της λίστας, χρησιμοποιήσαμε την υλοποίηση που είχαμε κάνει για την Εργασία 1 του μαθήματος.