Οικονομικό Πανεπιστήμιο Αθηνών

Τμήμα Πληροφορικής

Φθινοπωρινό Εξάμηνο 2016

Δομές Δεδομένων - Εργασία 3

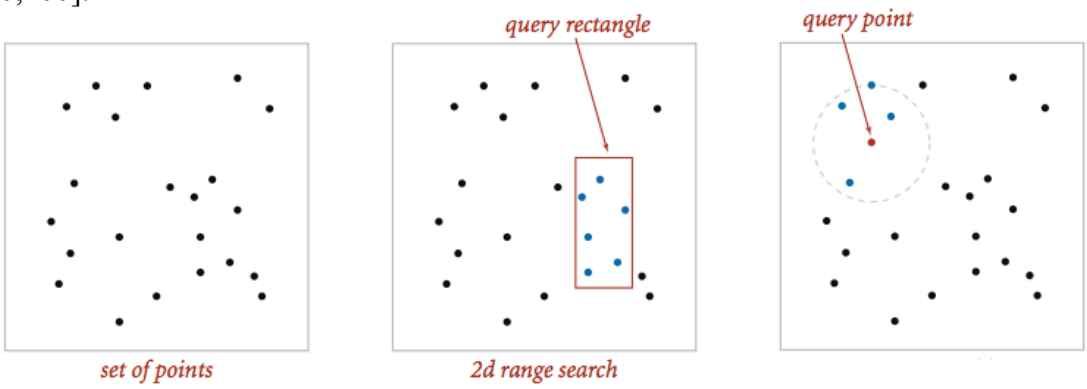
Διδάσκων: Ε. Μαρκάκης

Εφαρμογές των Δέντρων Δυαδικής Αναζήτησης

Σκοπός της εργασίας αυτής είναι η εξοικείωση με δενδρικές δομές και με τις επεκτάσεις των Δέντρων Δυαδικής Αναζήτησης. Συγκεκριμένα, θα πρέπει να υλοποιήσετε μεθόδους για 2d-trees, μια δενδρική δομή δεδομένων κατάλληλη για την αναπαράσταση και την επεξεργασία σημείων στο επίπεδο. Δεδομένων N σημείων στο δισδιάστατο χώρο, οι πιο σημαντικές λειτουργίες που θέλουμε να υλοποιήσουμε είναι

1. *range search:* βρες όλα τα σημεία, από τα N, που περιέχονται σε ένα παραλληλόγραμμο που δίνει ο χρήστης, (δείτε το σχήμα παρακάτω)
2. *nearest neighbor search:* βρες από τα N σημεία αυτό που απέχει τη μικρότερη απόσταση από κάποιο σημείο που καθορίζει ο χρήστης.

Για ευκολία, θα περιοριστούμε στο χώρο [0,100] x [0,100], επομένως όλα τα σημεία που θα θεωρήσουμε θα έχουν συντεταγμένες (x, y) με x, y ακέραιους στο διάστημα [0,100].



Εφαρμογές των 2d-trees: Τα 2d-trees και οι επεκτάσεις τους έχουν αρκετές εφαρμογές σε όραση υπολογιστών, επιτάχυνση νευρωνικών δικτύων, και ανάκτηση εικόνας. Μια χαρακτηριστική εφαρμογή είναι η ανίχνευση σύγκρουσης (collision detection) σε video games 2 διαστάσεων. Για παράδειγμα, σε ένα platform video game, συχνά το πρόγραμμα του παιχνιδιού καλείται να αποφασίσει α) αν ο χαρακτήρας που χειρίζεται ο παίκτης συγκρούστηκε με κάποιο αντικείμενο (π.χ. αντίπαλος παίκτης ή σφαίρα) ή β) αν γίνει μια έκρηξη, τότε πόσα αντικείμενα στο γύρω χώρο πρέπει να ανατιναχτούν. Για να παρθούν αυτές οι αποφάσεις, ένας μη-αποδοτικός τρόπος είναι να συγκριθεί η θέση του εξεταζόμενου αντικειμένου σε σχέση με όλα τα N αντικείμενα που εμφανίζονται στην οθόνη. Αυτό όμως είναι

υπολογιστικά ακριβό καθώς κοστίζει O(N). Με τη χρήση των 2d-trees, η πολυπλοκότητα μειώνεται σημαντικά σε O(logN) κατά μέσο όρο.

Παράδειγμα: Στο παιχνίδι Space Invaders θέλουμε να δούμε αν η σφαίρα που έριξε το αεροπλανάκι βρίσκει στόχο. Επιπλέον, αν η σφαίρα είναι ένα ειδικό όπλο που εκρήγνυται, τότε η υλοποίηση του προγράμματος θα πρέπει να εντοπίζει όλους τους εξωγήινους γύρω από την έκρηξη (δηλαδή, υλοποίηση της range_search) ώστε να τους καταστρέψει και αυτούς.

Μέρος Α: ΑΤΔ γεωμετρικών αντικειμένων: Ξεκινώντας, υλοποιήστε πρώτα μία κλάση για την αναπαράσταση σημείων και μία για την αναπαράσταση παραλληλογράμμων, με πλευρές παράλληλες με τους άξονες. Συγκεκριμένα:

Υλοποιήστε τον ΑΤΔ `Point` για σημεία στο επίπεδο που να υποστηρίζει τις εξής λειτουργίες:

```
public class Point {
   public Point(int x, int y)     // construct the point (x, y)
   public int x()                    // return the x-coordinate
   public int y()                    // return the y-coordinate
   public double distanceTo(Point z)    // Euclidean distance
                                        //between two points
   public int squareDistanceTo(Point z)   // square of the Euclidean
                                        //distance between two points
   public String toString()          // string representation: (x, y)
}
```

Για την κλάση αυτή μπορείτε να βασιστείτε στην κλάση Point από το βιβλίο (Κεφάλαιο 3) και να την τροποποιήσετε κατάλληλα όπου χρειάζεται.

Στη συνέχεια γράψτε τον ΑΤΔ `Rectangle` για την αναπαράσταση παραλληλογράμμων. Για ευκολία, τα παραλληλόγραμμα θα είναι παράλληλα προς τους άξονες. Επομένως, τα παραλληλόγραμμα είναι της μορφής

$$[x_{min}, x_{max}] \times [y_{min}, y_{max}]$$

όπου $x_{min}, x_{max}, y_{min}, y_{max} \in [0,100]$. Άρα η κλάση `Rectangle` αναπαρίσταται με τέσσερις ακεραίους. Οι λειτουργίες που πρέπει να υποστηρίζονται από την κλάση `Rectangle` είναι οι εξής:

```
public class Rectangle {
   public Rectangle(int xmin, int ymin, int xmax, int ymax)
        // construct the rectangle [xmin, ymin] x [xmax, ymax]
```

```
   public int xmin()          // minimum x-coordinate of rectangle
   public int ymin()          // minimum y-coordinate of rectangle
   public int xmax()          // maximum x-coordinate of rectangle
   public int ymax()          // maximum y-coordinate of rectangle
   public boolean contains(Point p) //does p belong to the rectangle?
   public boolean intersects(Rectangle that) // do the two rectangles
                                             // intersect?
   public double distanceTo(Point p)  // Euclidean distance from p
                                      //to closest point in rectangle
   public int squareDistanceTo(Point p)    // square of Euclidean
                    // distance from p to closest point in rectangle
   public String toString()    // string representation:
                               // [xmin, xmax] x [ymin, ymax]
}
```

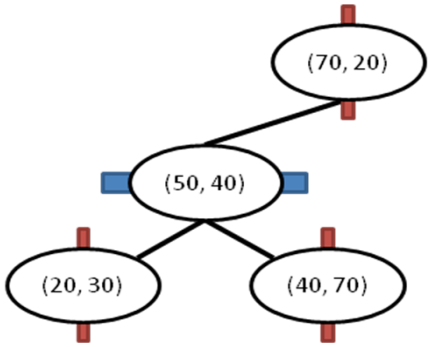
Ελέγξτε προσεκτικά τις μεθόδους σας μέχρι εδώ προτού προχωρήσετε. Δείτε τα σχόλια που περιγράφουν τι κάνει κάθε μέθοδος, καθώς και πότε πρέπει να πετάξετε κάποια εξαίρεση (επιλέξτε έτοιμη εξαίρεση από την Java ή κατασκευάστε δική σας, προκειμένου να εκφράζει καλύτερα τον τύπο του σφάλματος). Π.χ. αν στον constructor του παραλληλογράμμου, τα ορίσματα ικανοποιούν ότι xmin > xmax θα πρέπει να πετάτε κάποιο exception.

Για την μέθοδο contains(p), η απάντηση πρέπει να είναι true είτε όταν το σημείο p βρίσκεται στο εσωτερικό του παραλληλόγραμμου, είτε όταν βρίσκεται πάνω στις πλευρές του. Για την μέθοδο intersects, θα πρέπει να επιστρέφετε true όταν τα 2 παραλληλόγραμμα έχουν ένα τουλάχιστον κοινό σημείο.

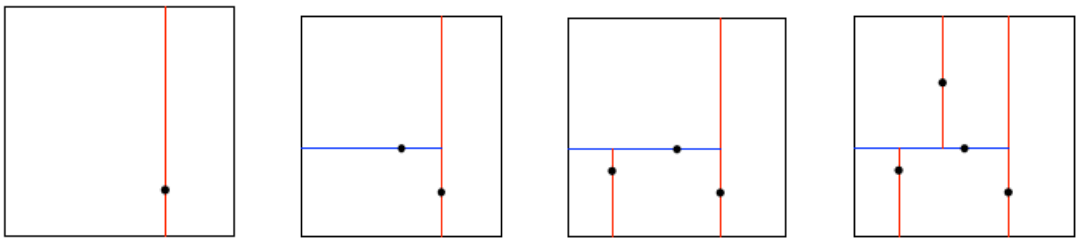
Μέρος Β: ΑΤΔ για 2d-trees. Ένα *2d-tree* είναι μία γενίκευση των Δέντρων Δυαδικής Αναζήτησης, για 2διάστατα κλειδιά. Κάθε κόμβος του 2d-tree θα αντιστοιχεί σε ένα από τα N σημεία που θα δίνει ο χρήστης. Επομένως προτού ορίσετε την κλάση TwoDTree που περιγράφεται παρακάτω, θα πρέπει πρώτα να ορίσετε μία κλάση TreeNode, για τους κόμβους (σε αναλογία με την υλοποίηση για ΔΔΑ του Κεφαλαίου 12). Ένας κόμβος στη δομή σας θα περιέχει τουλάχιστον ένα αντικείμενο Point και 2 δείκτες προς το αριστερό και δεξιό υποδέντρο. Η ιδέα στα *2d-trees* είναι ότι χρησιμοποιούμε εναλλάξ τις συντεταγμένες x, y για να καθορίσουμε το μονοπάτι της αναζήτησης καθώς και το πού θα εισαχθεί ένας νέος κόμβος (δείτε και τα hints στην τελευταία σελίδα).

Μέθοδοι *search* και *insert.* Ο αλγόριθμος αναζήτησης σε ένα *2d-tree* είναι παρόμοιος με τον αντίστοιχο αλγόριθμο σε ΔΔΑ και λειτουργεί ως εξής: ξεκινώντας από τη ρίζα, αν το σημείο που ψάχνουμε έχει συντεταγμένη x μικρότερη από την συντεταγμένη x της ρίζας, τότε συνεχίζουμε την αναζήτηση στο αριστερό υποδέντρο, αλλιώς τη συνεχίζουμε στο δεξιό (ελέγχουμε φυσικά πρώτα αν το σημείο που ψάχνουμε έχει x και y συντεταγμένες ίσες με το σημείο στη ρίζα, και σε τέτοια περίπτωση σταματάμε αφού έχουμε επιτυχή αναζήτηση). Στη συνέχεια **στο επόμενο επίπεδο χρησιμοποιούμε την συντεταγμένη y** για να καθορίσουμε αν θα κινηθούμε προς το αριστερό ή το δεξιό υποδέντρο, ενώ στο μεθεπόμενο επίπεδο χρησιμοποιούμε πάλι τη συντεταγμένη x κ.ο.κ. Με τον ίδιο τρόπο γίνεται και η εισαγωγή ενός νέου κόμβου. Ακολουθούμε το μονοπάτι στο δέντρο χρησιμοποιώντας εναλλάξ τις συντεταγμένες x και y και εισάγουμε το νέο σημείο ως φύλλο. Κάνουμε λοιπόν εισαγωγή σε φύλλο σε αντιστοιχία με το πρόγραμμα 12.15 του βιβλίου και όχι εισαγωγή στη ρίζα. Για παράδειγμα το δέντρο που θα δημιουργηθεί από τις 4

διαδοχικές εισαγωγές που φαίνονται παρακάτω αν γίνουν με σειρά από αριστερά προς δεξιά θα είναι το εξής:



insert (70, 20) *insert (50, 40)* *insert (20, 30)* *insert (40, 70)*



Γράψτε τον ΑΤΔ `TwoDTree` που αναπαριστά ένα 2d-tree και υποστηρίζει τις εξής λειτουργίες (μπορείτε να προσθέσετε όποιες άλλες μεθόδους κρίνετε εσείς απαραίτητες):

```
public class TwoDTree {
   public TwoDTree()                 // construct an empty tree
   public boolean isEmpty()          // is the tree empty?
   public int size()                 // number of points in the tree
   public void insert(Point p)       // inserts the point p to the tree
   public boolean search(Point p)    // does the tree contain p?
   public Point nearestNeighbor(Point p) // point in the tree that is
                                         //closest to p
                                         //(null if tree is empty)
   public List<Point> rangeSearch(Rectangle rect) // Returns a list
// with the Points that are contained in the rectangle

}
```

Προσοχή στην μέθοδο `rangeSearch`: η μέθοδος επιστρέφει τα σημεία (Points) που περιέχονται στο συγκεκριμένο χώρο (Rectangle) μέσα σε μια δομή δεδομένων. Αν δεν υπάρχουν σημεία, θα επιστρέψει μια άδεια δομή (αλλά όχι null). Στο σημείο αυτό μπορείτε να χρησιμοποιήσετε είτε λίστα είτε ουρά (ή ακόμα και ΔΔΑ), και μπορείτε να βασιστείτε στις δομές που έχουμε υλοποιήσει στο εργαστήριο ή να επαναχρησιμοποιήσετε μια δομή που αναπτύξατε στις εργασίες σας και να την τροποποιήσετε αν χρειάζεται. Η δήλωση της συνάρτησης `rangeSearch` να επιστρέφει `List<Point>` είναι **ενδεικτική**. Μπορείτε να την τροποποιήσετε κατά βούληση ώστε να επιστρέφει την δομή που προτιμήσατε.

Όλοι οι παραπάνω ΑΤΔ (`Point`, `Rectangle`, `TwoDTree`) μπορούν να περιέχουν και άλλες βοηθητικές μεθόδους που κρίνετε εσείς απαραίτητες.

Εκτέλεση, input και output

Το πρόγραμμα TwoDTree.java θα έχει μία μέθοδο main η οποία αρχικά θα διαβάζει από ένα αρχείο εισόδου τον αριθμό των σημείων και μετά τα ίδια τα σημεία που θα εισάγει στο δέντρο (το όνομα του αρχείου εισόδου θα δίνεται μαζί με το path όπως και στις άλλες εργασίες). Ένα παράδειγμα για το αρχείο εισόδου είναι το εξής:

```
5
30 50
15 97
0 36
100 0
100 40
```

Στο παραπάνω παράδειγμα, N=5, και μετά την πρώτη γραμμή ακολουθούν τα 5 σημεία. Δεν επιτρέπεται η επανάληψη σημείων. Η main θα διαβάζει τα σημεία και θα δημιουργεί ένα 2d-tree κάνοντας διαδοχικές εισαγωγές. Αν υπάρχει οποιαδήποτε ασυνέπεια στην είσοδο θα πρέπει να τερματίζετε το πρόγραμμα και να τυπώνετε το αντίστοιχο μήνυμα λάθους (π.χ. αν κάποια συντεταγμένη υπερβαίνει το 100 ή αν κάποιο σημείο επαναλαμβάνεται).
Στη συνέχεια η main θα εκτελεί έναν ατέρμονα βρόχο όπου θα ρωτάει το χρήστη αν θέλει να δώσει ένα query rectangle ή ένα query point ή απλώς να τερματίσει το πρόγραμμα.

Αν ο χρήστης δώσει ένα query rectangle, θα πρέπει να βρείτε τα σημεία του δέντρου που περιέχονται μέσα στο δοσμένο παραλληλόγραμμο (rangeSearch) και να τα τυπώσετε. Το παραλληλόγραμμο μπορεί να δίνεται από το χρήστη στη μορφή:

```
20 30
40 80
```

Αυτό θα αντιστοιχεί στο παραλληλόγραμμο [20, 30] x [40, 80].

Αν ο χρήστης δώσει ένα query point, π.χ. στη μορφή:

```
34 76
```

θα πρέπει να τρέξετε τη μέθοδο nearestNeighbor και να τυπώσετε το σημείο του δέντρου που βρίσκεται πιο κοντά στο δοσμένο σημείο καθώς και την απόσταση.

ΠΡΟΣΟΧΗ: Όταν συγκρίνετε Ευκλείδιες αποστάσεις, ο υπολογισμός της τετραγωνικής ρίζας ενδέχεται να κόψει μερικά απο τα δεκαδικά ψηφία. Είναι προτιμότερο να χρησιμοποιείτε το τετράγωνο της απόστασης για τη σύγκριση.

Παραδοτέα: Για την εργασία θα υποβάλετε τα εξής αρχεία:

1) Point.java, ο ΑΤΔ για σημεία.
2) Rectangle.java, ο ΑΤΔ για παραλληλόγραμμα.
3) TwoDTree.java, ο ΑΤΔ για 2d-trees που περιέχει και τη main.
4) Μία σύντομη αναφορά σε pdf αρχείο με όνομα project3-report.pdf, στην οποία θα αναφέρετε τα μέλη της ομάδας σας και θα περιγράψετε συνοπτικά την υλοποίησή σας (2-5 σελίδες).

5) Ό,τι άλλο αρχείο χρειαστείτε, αν π.χ. χρησιμοποιήσετε κάποια λίστα ή κάτι άλλο που έχετε δει στο εργαστήριο. Ως συνήθως **δεν επιτρέπονται έτοιμες δομές της Java.**

Όλα τα παραπάνω αρχεία θα πρέπει να μπουν σε ένα αρχείο zip. Το όνομα που θα δώσετε στο αρχείο αυτό θα είναι ο αριθμός μητρώου σας πχ. 3030056_3030066.zip ή 3030056.zip (αν δεν είστε σε ομάδα). Στη συνέχεια, θα υποβάλλετε το zip αρχείο σας στην περιοχή του μαθήματος «Εργασίες» στο e-class.

Η προθεσμία παράδοσης της εργασίας είναι Κυριακή, 22 Ιανουαρίου 2017 και ώρα 23:59.

Hints για την υλοποίηση των rangeSearch και nearestNeighbor: Το κύριο πλεονέκτημα των 2d-trees είναι ότι κάνουν πιο αποδοτική την υλοποίηση τέτοιων μεθόδων αναζήτησης. Καταρχήν παρατηρήστε ότι κάθε κόμβος σε ένα 2d-tree ουσιαστικά αντιστοιχεί σε ένα παραλληλόγραμμο, που περιέχει όλα τα σημεία που βρισκονται στα 2 υποδέντρα του. Η ρίζα αντιστοιχεί στο παραλληλόγραμμο [0, 100] x [0,100]. Στο παράδειγμα πιο πάνω, το αριστερό παιδί της ρίζας αντιστοιχεί στο παραλληλόγραμμο [0, 70] x [0, 100], δηλαδή στα σημεία που έχουν μικρότερη ή ίση x-συντεταγμένη από αυτή της ρίζας (δείτε το σχήμα πιο πάνω). Το δεξί παιδί της ρίζας αντιστοιχεί στο παραλληλόγραμμο [70, 100] x [0, 100]. Στη συνέχεια το αριστερό παιδί του κόμβου (50, 40) αντιστοιχεί στο παραλληλόγραμμο [0, 70] x [0, 40], χρησιμοποιούμε δηλαδή την y-συντεταγμένη σε αυτό το επίπεδο. Και συνεχίζουμε με αυτόν τον τρόπο εναλλάξ μέχρι τα φύλλα.

- *Range Search.* Για να βρείτε όλα τα σημεία που περιέχονται σε ένα παραλληλόγραμμο που δίνει ο χρήστης, ξεκινήστε από τη ρίζα και αναδρομικά ψάξτε στα 2 υποδέντρα με τον εξής κανόνα: αν το παραλληλόγραμμο που έδωσε ο χρήστης δεν έχει κανένα κοινό σημείο με το παραλληλόγραμμο στο οποίο αντιστοιχεί κάποιος κόμβος (έτσι όπως το περιγράψαμε παραπάνω), τότε δεν χρειάζεται να ψάξετε αυτόν τον κόμβο ούτε και τα υποδέντρα του. Η αναδρομή συνεχίζεται μόνο εάν υπάρχει κάποια πιθανότητα το υποδέντρο να περιέχει σημεία που ανήκουν στο παραλληλόγραμμο. Τέτοιες υλοποιήσεις επιταχύνουν σε μεγάλο βαθμό την εκτέλεση, καθώς ψάχνουν μόνο στα κομμάτια του δέντρου που χρειάζεται. Για την υλοποίηση θα σας είναι χρήσιμη η μέθοδος *intersects* της κλάσης Rectangle.
- *Nearest neighbor search.* Για να βρείτε το πιο κοντινό σημείο σε ένα σημείο p που δίνει ο χρήστης, ξεκινήστε πάλι από τη ρίζα και αναδρομικά ψάξτε στα υποδέντρα με τον εξής κανόνα: αν το τρέχον κοντινότερο σημείο που έχετε βρει μέχρι εκείνη τη στιγμή έχει απόσταση από το p αυστηρά μικρότερη από την απόσταση του p από το παραλληλόγραμμο στο οποίο αντιστοιχεί κάποιος κόμβος (με χρήση της distanceTo από τη Rectangle) τότε πάλι δεν χρειάζεται να ψάξετε αυτόν τον κόμβο και τα υποδέντρα του.
- Για debugging, μπορείτε να έχετε μία μέθοδο που να τυπώνει τους κόμβους του δέντρου ανά επίπεδο (σαν τη διάσχιση level order που είδαμε στο μάθημα). Δεν χρειάζεται να υπάρχει στα παραδοτέα, απλώς για δική σας διευκόλυνση.