ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

Δομές Δεδομένων - Εργασία 3

Τμήμα Πληροφορικής

Φθινοπωρινό Εξάμηνο 2019-2020

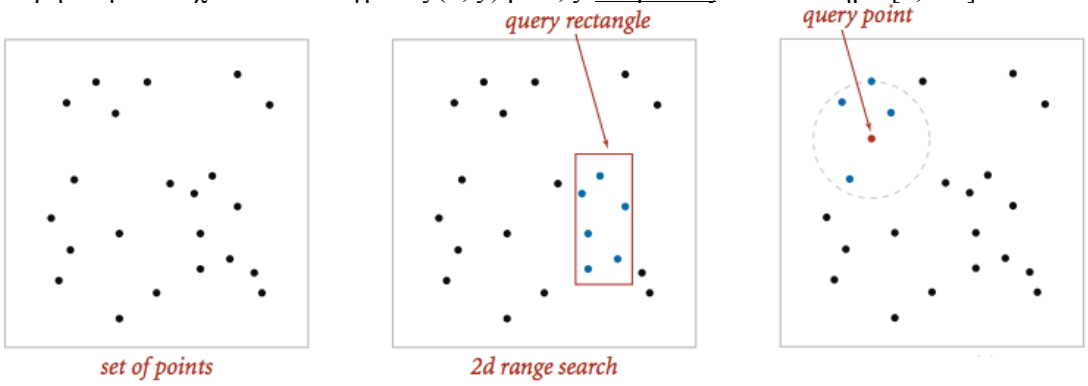
Διδάσκων: Ε. Μαρκάκης

Δέντρα 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 Ν σημείων στο δισδιάστατο χώρο, οι πιο σημαντικές λειτουργίες που θέλουμε να υλοποιήσουμε με χρήση 2d-trees είναι

1. *range search:* βρες όλα τα σημεία, από τα Ν, που περιέχονται σε ένα παραλληλόγραμμο που δίνει ο χρήστης, (δείτε το σχήμα παρακάτω)
2. *nearest neighbor search:* βρες από τα Ν 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. Συχνά, το πρόγραμμα ενός τέτοιου 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. Π.χ., στο παιχνίδι Space Invaders (βλ. εικόνα παρακάτω), 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. 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.

Μέρος Α - ΑΤΔ γεωμετρικών αντικειμένων [20 μονάδες]. 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 (συν όποιες άλλες χρειαστείτε):

```
public class Point {
   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 ένα αντικείμενο της κλάσης Rectangle αναπαρίσταται με τέσσερις ακεραίους. Οι λειτουργίες που πρέπει να υποστηρίζονται από την κλάση Rectangle είναι οι εξής (συν όποιες άλλες χρειαστείτε):

```
public class Rectangle {
   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. 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. Τέλος, όταν θα χρειαστεί να συγκρίνετε Ευκλείδιες αποστάσεις στα υπόλοιπα μέρη της εργασίας, ο υπολογισμός της τετραγωνικής ρίζας ενδέχεται να κόψει μερικά απο τα δεκαδικά ψηφία. Για αυτό το λόγο, είναι συνήθως προτιμότερο να χρησιμοποιείτε το τετράγωνο της απόστασης (squareDistanceTo) για τη σύγκριση.

Μέρος Β [50 μονάδες]. ΑΤΔ για 2d-trees. Ένα *2d-tree* είναι μία γενίκευση των Δέντρων Δυαδικής Αναζήτησης, για 2διάστατα κλειδιά. Επομένως προτού ορίσετε την κλάση TwoDTree, θα πρέπει πρώτα να ορίσετε μία κλάση TreeNode, για τους κόμβους του δέντρου (σε αναλογία με την κλάση Node που χρησιμοποιούμε στο μάθημα για την υλοποίηση ΔΔΑ του Κεφαλαίου 12).
Η κλάση TreeNode. Κάθε κόμβος του δέντρου θα αντιστοιχεί σε ένα σημείο στο επίπεδο και πρέπει να περιέχει ένα αντικείμενο Point και 2 δείκτες προς το αριστερό και δεξιό υποδέντρο. Επομένως στην κλάση Treenode πρέπει να υπάρχουν τουλάχιστον τα εξής πεδία (και ό,τι άλλο θέλετε εσείς να προσθέσετε):

```
private class Treenode {
   Point item // an object of the class Point
   Treenode l // pointer to left subtree
   Treenode r // pointer to right subtree
   ...
}
```

Σε αντίθεση με το μάθημα, δεν θα χρειαστεί να έχετε μέθοδο key() για πρόσβαση στο κλειδί. Το κλειδί εδώ ουσιαστικά αποτελείται από τις 2 συντεταγμένες του σημείου, επομένως για έναν κόμβο h, μπορείτε να έχετε πρόσβαση στις συντεταγμένες μέσω των μεθόδων h.item.x() και h.item.y().

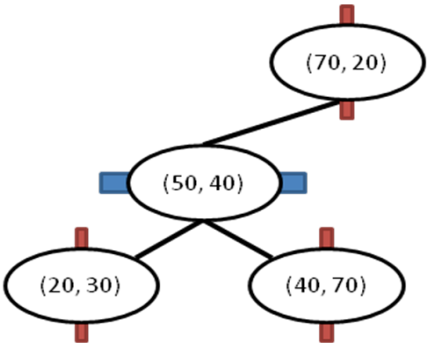
Η κλάση TwoDTree. Ο ΑΤΔ TwoDTree θα πρέπει να υποστηρίζει τις λειτουργίες που φαίνονται παρακάτω (μπορείτε να προσθέσετε όποιες άλλες μεθόδους κρίνετε εσείς απαραίτητες):

```
public class TwoDTree {
   private class TreeNode {
   ...
   };
   private TreeNode head; //ρίζα στο δέντρο
   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?
```

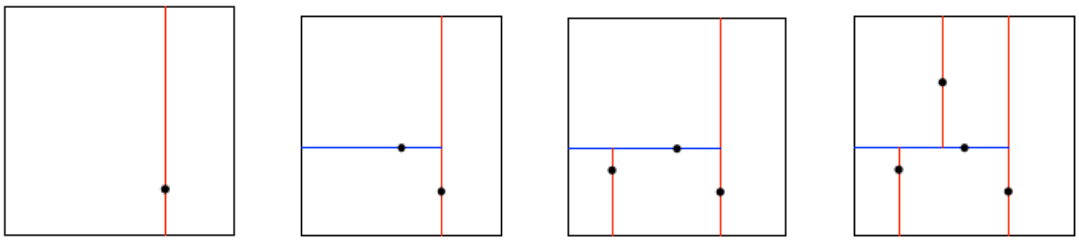
```
    public Point nearestNeighbor(Point p) // point in the tree that is
        //closest to p (null if tree is empty)
    public List<Point> rangeSearch(Rectangle rect) // Returns a list
        //with the Points that are contained in the rectangle
}
```

Υλοποίηση μεθόδων *search* και *insert.* 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. Ο αλγόριθμος αναζήτησης σε ένα *2d-tree* είναι γενίκευση του αλγορίθμου 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. **Δεν κάνουμε εισαγωγή στη ρίζα.** Επίσης, η insert θα πρέπει να ελέγχει ότι το σημείο προς εισαγωγή δεν υπάρχει ήδη στο δέντρο. Αν υπάρχει θα πρέπει να μην το εισάγει και να τυπώνει απλά ένα μήνυμα λάθους. Ως παράδειγμα, παρακάτω δίνονται 4 σημεία και το δέντρο που θα δημιουργηθεί από τις 4 διαδοχικές εισαγωγές αν γίνουν με σειρά από το p1 ως το p4.

Point p1 = (70, 20), p2 = (50, 40), p3 = (20, 30), p4 = (40, 70)



insert (p1) *insert (p2)* *insert (p3)* *insert (p4)*



Μέθοδος rangeSearch: Η δήλωση της συνάρτησης rangeSearch να επιστρέφει List<Point> είναι **ενδεικτική**. Μπορείτε να την τροποποιήσετε κατά βούληση ώστε να επιστρέφει την δομή που προτιμάτε. Μπορείτε να χρησιμοποιήσετε είτε λίστα είτε ουρά, και να βασιστείτε στις δομές που

έχετε υλοποιήσει στο εργαστήριο ή στις εργασίες σας. Αν δεν υπάρχουν σημεία για επιστροφή, η μέθοδος απλά θα επιστρέψει μια άδεια δομή (αλλά όχι null).

Μέρος Γ [20 μονάδες]. Μενού διαχείρισης. Στο πρόγραμμα TwoDTree.java προσθέστε μία μέθοδο main η οποία αρχικά θα διαβάζει από ένα αρχείο εισόδου ένα σύνολο από σημεία στο επίπεδο και μετά θα τα εισάγει στο δέντρο (το όνομα του αρχείου εισόδου θα δίνεται μαζί με το path όπως και στις άλλες εργασίες). Ένα παράδειγμα για το format του αρχείου εισόδου είναι το εξής:

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

Στο παραπάνω παράδειγμα, η 1η γραμμή δηλώνει πόσα σημεία βρίσκονται στο αρχείο (εδώ N=5), και μετά την πρώτη γραμμή ακολουθούν τα 5 σημεία. 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 μήνυμα λάθους (π.χ. αν κάποια συντεταγμένη υπερβαίνει το 100 ή αν το πλήθος των σημείων δεν είναι N).
Στη συνέχεια αφότου έχει δημιουργηθεί το 2d-tree, η main θα εκτελεί έναν ατέρμονα βρόχο όπου θα εμφανίζει στο χρήστη ένα μενού με τις παρακάτω επιλογές:

1. Compute the size of the tree
2. Insert a new point
3. Search if a given point exists in the tree
4. Provide a query rectangle
5. Provide a query point

Το μενού αυτό θα σας είναι χρήσιμο ούτως ή άλλως και για debugging, και μπορείτε να προσθέσετε κι εσείς περισσότερες επιλογές. Αν ο χρήστης επιλέξει την πρώτη επιλογή, θα πρέπει να τρέξετε την size. Αν επιλέξει την $2^{η}$, θα πρέπει να ζητήσετε να δοθούν 2 συντεταγμένες για ένα νέο σημείο και να καλέσετε την insert. Ομοίως στην $3^{η}$ επιλογή, θα πρέπει να χρησιμοποιήσετε την search. Στην $4^{η}$ επιλογή, θα ζητήσετε από τον χρήστη να δώσει ένα παραλληλόγραμμο, και 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 π.χ.:

```
20 30
40 80
```

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

Τελος, στην 5η επιλογή, 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.

Μέρος Δ - Αναφορά παράδοσης [10 μονάδες]. Ετοιμάστε μία σύντομη αναφορά σε pdf αρχείο (μην παραδώσετε Word ή txt αρχεία!) με όνομα project3-report.pdf, στην οποία θα αναφερθείτε στα εξής:

a. Εξηγήστε συνοπτικά πώς υλοποιήσατε τις μεθόδους contains, intersects, distanceTo της κλάσης Rectangle του Μέρους Α. Άνω όριο: 1 σελίδα (μπορείτε αν θέλετε να κάνετε και κάποιο σχήμα για να επεξηγήσετε τις ιδέες των μεθόδων).
b. Εξηγήστε πώς υλοποιήσατε τις μεθόδους nearestNeighbor και rangeSearch του Μέρους Β. Άνω όριο: 2 σελίδες

Το συνολικό μέγεθος της αναφοράς θα πρέπει να είναι τουλάχιστον 2 σελίδες. Μην ξεχνάτε τα ονοματεπώνυμα και τους ΑΜ σας στην αναφορά.

Οδηγίες Παράδοσης

Η εργασία σας θα πρέπει να μην έχει συντακτικά λάθη και να μπορεί να μεταγλωττίζεται. Εργασίες που δεν μεταγλωττίζονται χάνουν το **50%** της συνολικής αξίας.
Η εργασία θα αποτελείται από:

1. Τον πηγαίο κώδικα (source code). Τοποθετήστε σε ένα φάκελο με όνομα **src** τα αρχεία java που έχετε φτιάξει. Επιπλέον, φροντίστε να συμπεριλάβετε όποια άλλα αρχεία πηγαίου κώδικα απαιτούνται για να μεταγλωττίζεται η εργασία σας. Φροντίστε επίσης να προσθέσετε επεξηγηματικά σχόλια όπου κρίνετε απαραίτητο στον κώδικά σας.
2. Την αναφορά παράδοσης.

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. Δεν χρειάζεται υποβολή και από τους 2 φοιτητές μιας ομάδας.

Η προθεσμία παράδοσης της εργασίας είναι Παρασκευή, 17 Ιανουαρίου 2020 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 σκεφτείτε ότι κάθε κόμβος σε ένα 2d-tree ουσιαστικά αντιστοιχεί "νοητά" σε ένα παραλληλόγραμμο, που περιέχει όλα τα σημεία που βρισκονται στα 2 υποδέντρα του. Η ρίζα αντιστοιχεί στο παραλληλόγραμμο [0, 100] x [0,100], δηλαδή σε όλο το κομμάτι του επιπέδου που μας αφορά. Στο παράδειγμα που δόθηκε στο Μέρος Β 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 τα σχήματα στο Μέρος Β). 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 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 π.χ. ανά επίπεδο (σαν τη διάσχιση level order που είδαμε στο μάθημα). Δεν χρειάζεται να υπάρχει στα παραδοτέα, απλώς για δική σας διευκόλυνση.