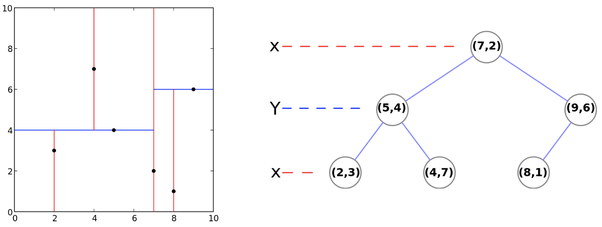
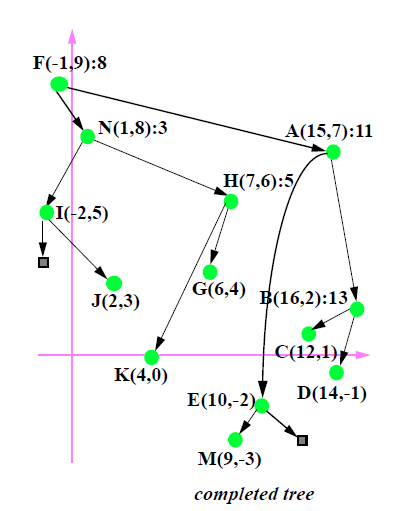
**Αναφορά Εργασίας Πολυδιάστατων δομών Δεδομένων**

K-D Trees και Priority Search Trees (8)

1)**Βασικές Έννοιες:**

* **K-D Tree:**
  + Δημιουργήθηκε το 1975 από τον Jon Louis Bentley,στο Stanford University.
  + Αποτελεί έναν τύπο πολυδιάστατου δυαδικού δέντρου αναζήτησης.
  + Το δέντρο περιέχει κόμβους που αντιπροσωπεύουν σημεία στον k-διάστατο χώρο. Το κάθε επίπεδο της δομής είναι οργανωμένο με βάση τις διαφορετικές διαστάσεις (X, Y, Z,…) κυκλικά.
* **Priority Search Tree:**
  + Δημιουργήθηκε το 1985 από τον Edward M. McCreight.
  + Ομοίως με τα K-D Tree, αποτελεί έναν τύπο πολυδιάστατου δυαδικού δέντρου αναζήτησης.
  + Το δέντρο περιέχει κόμβους που αντιπροσωπεύουν σημεία στον 2-διάστατο χώρο. Είναι οργανωμένο κατά βάθος σε φθίνουσα σειρά ως προς την μία από τις 2 διαστάσεις, ενώ τα παιδιά του κάθε κόμβου είναι οργανωμένα ως προς την άλλη διάσταση
* **3 Sided Query:**

Το ερώτημα εύρεσης όλων των σημείων που βρίσκονται μέσα σε μία ημι-άπειρη «λωρίδα». Η «λωρίδα» αυτή περιγράφεται από 3 μεταβλητές, το κατώτατο και ανώτερο όριο ως προς την μία διάσταση και το κατώτατο ή ανώτατο ως προς την άλλη.

2)**Παρουσίαση και Ανάλυση κώδικα:**

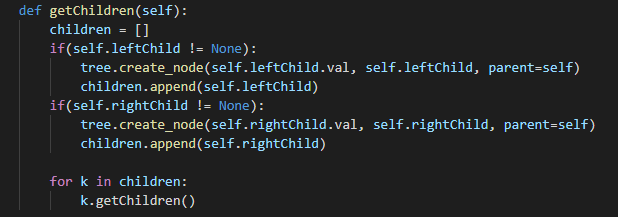
(Ο κώδικας μπορεί να έχει μικρές διαφορές από αυτόν που θα σταλθεί.)

**KD Tree:**

**Kλάσεις:**

* **Node:** Αντιπροσωπεύει τουςκόμβους του δέντρου.

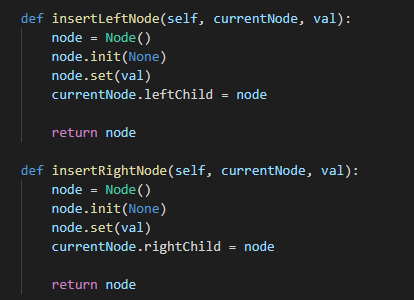
**Συναρτήσεις:**

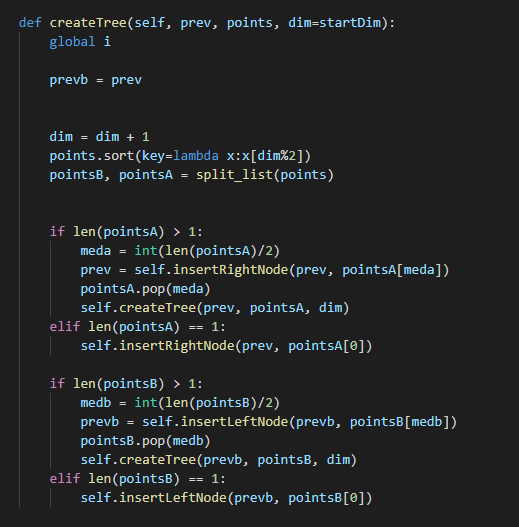
* + **init(self, val):** Initialisation των αντικειμένων με τις τιμές val, leftChild, rightChild.
  + **getChildren(self):** Εύρεση και αποθήκευση όλων των «παιδιών» κόμβου στην λίστα children και βοηθάει στην απεικόνιση του δέντρου.

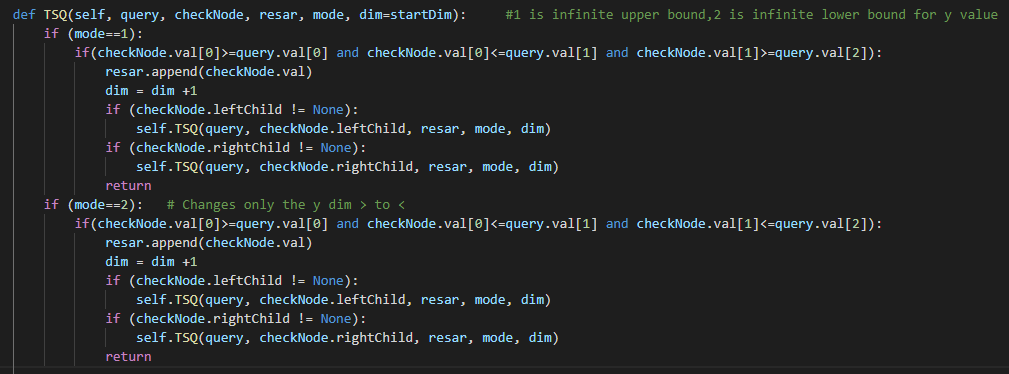
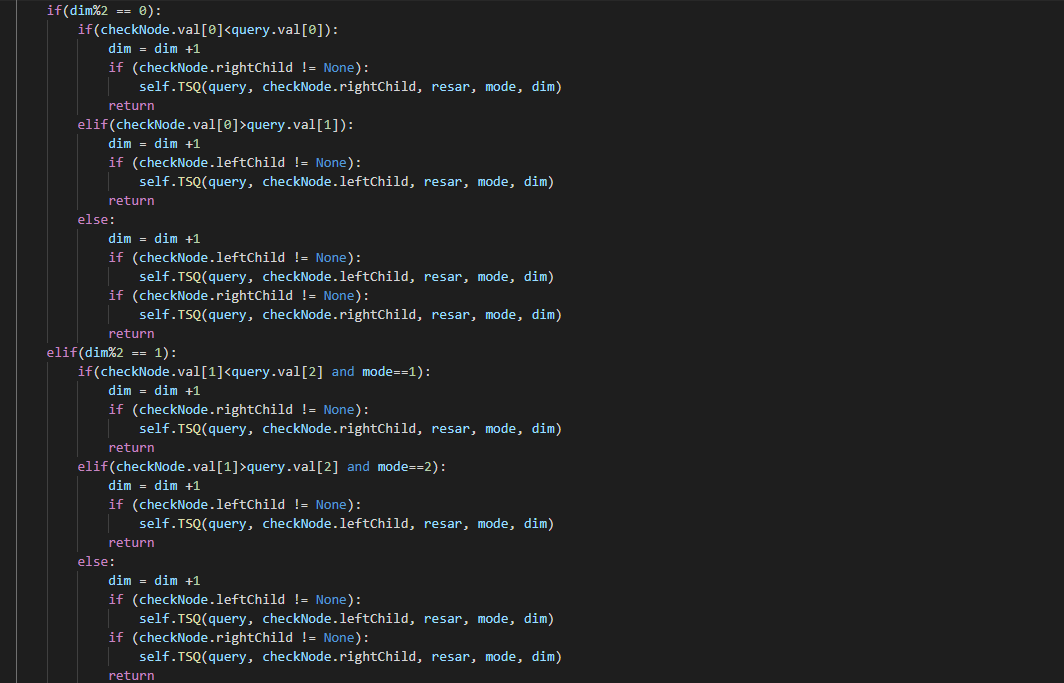
1. **KDT:** Αντιπροσωπεύει το K-D Tree.

**Συναρτήσεις:**

* + **init(self):** Initialisation των αντικειμένων με την τιμή root.
  + **setRoot(self, val)**
  + **insertLeftNode(self, currentNode, val):** Ορισμός κόμβου με την τιμή val ως αριστερό παιδί του currentNode και επιστροφή του.
  + **insertRightNode(self, currentNode, val):** Ορισμός κόμβου με την τιμή val ως δεξί παιδί του currentNode και επιστροφή του.

****

* + **createTree(self, prev, points, dim=startDim):** Δημιουργία του K-D Treeμε αναδρομική κλήση της συνάρτησης για κάθε υποδέντρο. Η μεταβλητή prev αντιπροσωπεύει την ρίζα του κάθε υποδέντρου. Points είναι η λίστα όλων των σημείων που θέλουμε να περιέχει το δέντρο. dim είναι μια integer μεταβλητή που έχει αποθηκευμένη την διάσταση στην οποία είναι ταξινομημένα τα σημεία στο βάθος του δέντρου στο οποίο καλείται η συνάρτηση κάθε φορά. startDim είναι η αρχική κατεύθυνση με την οποία ξεκινάμε τον «χωρισμό» των στοιχείων.
  + **ΤSQ(self, query, checkNode, resar, mode, dim=startDim):** Πραγματοποίηση και αποθήκευση των αποτελεσμάτων 3-sided Query. Το query είναι αντικείμενο της κλάσης Query και περιέχει τις τιμές των ορίων που αυτό θέλουμε να έχει. Το checkNode περιέχει τον κόμβο που είναι ρίζα του υποδέντρου που θέλουμε με κάνουμε την αναζήτηση.resar ονομάζεται η λίστα στην οποία θα αποθηκευτούν όλα τα σημεία που βρίσκονται στα επιθυμητά όρια του 3-sided query. Η integer μεταβλητή mode χρησιμοποιείται για να καθορίσουμε αν η τιμή που ορίζεται μέσω του αντικειμένου query για τον άξονα Y είναι το ανώτατο ή το κατώτατο άκρο. Το dim όπως αναφέρθηκε στο createTree().

1. **Query:**

**Συναρτήσεις:**

* + **init(self, minx, maxx, y):** Initialisation των αντικειμένων με τις τιμές minx, maxx, y.

**Συναρτήσεις Γενικής Χρήσης:**

* + **read\_file(file\_name, array):** Διάβασμα json αρχείου με όνομα file\_name και αποθήκευση των σημείων στην λίστα array.
  + **Split\_list(alist, wanted\_parts=2):** Διαχωρισμός λίστας alist σε δύο ίσα μέρη. Χρησιμοποιείται για να χωρίσουμε τα σημεία και να δημιουργήσουμε 2 υποδέντρα.
  + **Visual(KDT):** Απεικόνιση του δέντρου με την βοήθεια της βιβλιοθήκης treelib.

**Priority Search Tree:**

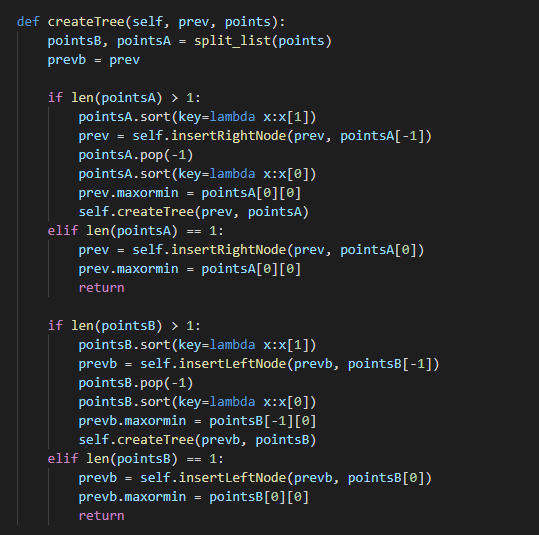
**Kλάσεις:**

* **Node:** Ομοίως με το K-D Tree.

**Επιπλέον Συναρτήσεις:**

* + Η Συνάρτηση init() αρχικοποιεί επίσης την μεταβλητή maxormin. Αντιπροσωπεύει την μικρότερη τιμή ως προς Χ του υποδέντρου που έχει το σημείο αυτό ως ρίζα, όταν βρίσκεται στο δεξί υποδέντρο ή την μεγαλύτερη τιμή αν βρίσκεται στο αριστερό.
  + **calcMed(self):** Υπολογισμός σημείου διαχωρισμού σε δύο υποδέντρα.
* **PST:** Αντιπροσωπεύει το Priority Search Tree.

**Συναρτήσεις:**

* + **init(self):** Initialisation των αντικειμένων με την τιμή root.
  + **setRoot, insertLeftNode και insertRightNode:** Ομοίως με το K-D Tree
  + **createTree(self, prev, points):** Δημιουργεί το Priority Search Tree με αναδρομικές κλήσεις του εαυτού του. Η τιμή prev είναι η ρίζα του κάθε υποδέντρου και points η λίστα όλων των σημείων που θέλουμε να εισάγουμε στο δέντρο.
  + **Tsq(self, query, checkNode, resar):**  Πραγματοποίηση και αποθήκευση των αποτελεσμάτων 3-sided Query. Το query είναι αντικείμενο της κλάσης Query και περιέχει τις τιμές των ορίων που αυτό θέλουμε να έχει. Το checkNode περιέχει τον κόμβο που είναι ρίζα του υποδέντρου που θέλουμε με κάνουμε την αναζήτηση.resar ονομάζεται η λίστα στην οποία θα αποθηκευτούν όλα τα σημεία που βρίσκονται στα επιθυμητά όρια του 3-sided query.

1. **Query:**
   * Ομοίως με το K-D Tree.

**Συναρτήσεις Γενικής Χρήσης:**

* + read\_file, split\_list και Visual ομοίως με το το K-D Tree.

**Λειτουργικότητα K-D Tree και Priority Search Tree:**

**createTree:**

**Priority Search Tree:**

1. Αρχικά, πριν να κληθεί η συνάρτηση createTree,έχει ταξινομηθεί ο πίνακας των σημείων ως προς τον Y άξονα και έχει οριστεί το μεγαλύτερο από αυτά ως ρίζα του δέντρου. Έπειτα το σημείο αυτό αφαιρείται από τον πίνακα.
2. Ο πίνακας των σημείων ταξινομείται ως προς Χ και καλείται η createTree.
3. Ο πίνακας χωρίζεται σε 2 ίσα μέρη και αποθηκεύουμε την μεταβλητή prev(ρίζα) στην prevb.
4. Aν ο πίνακας pointsA είναι μεγαλύτερος του 1, ταξινομούνται tα σημεία του ως προς την Υ μεταβλητή. Το μεγαλύτερο από αυτά εισάγεται ως δεξί παιδί της ρίζας, αφαιρείται από την λίστα και ανανεώνεται η μεταβλητή prev παίρνοντας την τιμή του.
5. Η υπολειπόμενη λίστα των σημείων ταξινομείται πάλι ως προς Χ και το σημείο με το μικρότερο Χ εισάγεται στην μεταβλητή maxormin του νέου prev.
6. H συνάρτηση createTree καλείται ξανά με το νέο prev και την νέα λίστα σημείων.
7. Αν ο πίνακας pointsA έχει μόνο ένα σημείο, τότε το σημείο αυτό εισάγεται αμέσως ως δεξί παιδί της ρίζας, ανανεώνεται η μεταβλητή prev παίρνοντας την τιμή του και η μεταβλητή maxormin την τιμή του στον Χ άξονα.
8. Αντίστοιχα για τον πίνακα pointsB, με την διαφορά πως τα σημεία εισάγονται ως αριστερά παιδιά της ρίζας(prevb) και η μεταβλητή maxormin παίρνει την μεγαλύτερη τιμή του Χ άξονα των σημείων.

**K-D Tree:**

1. Αρχικά, πριν να κληθεί η συνάρτηση createTree ταξινομείται η λίστα points, που περιέχει όλα τα σημεία ως προς τον άξονα Χ ή Υ που καθορίζει η μεταβλητή startDim.
2. Έπειτα, αποθηκεύεται το σημείο που βρίσκεται στην μέση του πίνακα ως ρίζα του δέντρου και αφαιρείται από τον πίνακα.
3. Στην συνέχεια καλείται η συνάρτηση createTree με ορίσματα την ρίζα του δέντρου, τα σημεία που έχουν απομείνει και την αρχική διάσταση χωρισμού του πίνακα.
4. Η μεταβλητή prev,που αντιπροσωπεύει την ρίζα του υποδέντρου αποθηκεύεται στην μεταβλητή prevb.
5. Έπειτα, η μεταβλητή dim αυξάνεται κατά ένα και έτσι εξασφαλίζεται το σορτάρισμα του πίνακα να πραγματοποιηθεί ως προς τον άλλο άξονα.
6. Στην συνέχεια, χωρίζεται ο πίνακας των σημείων στην μέση με την βοήθεια της συνάρτησης split\_list.
7. Αν η λίστα pointsA είναι μεγαλύτερη του 1, βρίσκουμε το μέσο σημείο της και το εισάγουμε ως δεξί παιδί της ρίζας. Ανανεώνεται η τιμή της ρίζας με την τιμή του παιδιού που μόλις εισήγαμε στο δέντρο και το σημείο αυτό αφαιρείται από τον πίνακα τον σημείων.
8. Η Συνάρτηση createTree ξανακαλείται με ορίσματα την νέα τιμή του prev (ρίζα) και τα υπόλοιπα σημεία του πίνακα pointsA.
9. Aν o πίνακας pointsA περιέχει μόνο ένα σημείο, το σημείο αυτό εισάγεται αμέσως ως δεξί παιδί της ρίζας.
10. Ομοίως με τον πίνακα pointsA γίνεται και η επεξεργασία του πίνακα pointsB με την διαφορά πως τα σημεία εισάγονται ως αριστερά παιδιά της ρίζας(prevb).

**TSQ:**

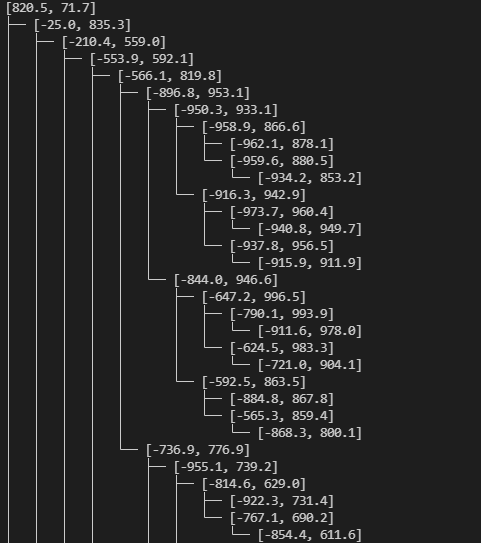
**Priority Search Tree:**

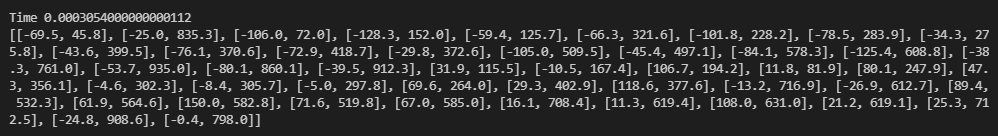
1. Αρχικά δημιουργούμε ένα αντικείμενο της κλάσης Query, το οποίο περιέχει τα όρια του query που επιθυμούμε.
2. Έπειτα, καλείται η συνάρτηση Tsq με ορίσματα το αντικείμενο query, το η ρίζα του δέντρου ή υποδέντρου που θέλουμε να περιέχει τα σημεία και την λίστα στην οποία θέλουμε να αποθηκεύσουμε το αποτέλεσμα του query.
3. Σε πρώτο στάδιο γίνεται έλεγχος της y τιμής του σημείου. Αν αυτή είναι μικρότερη από την κατώτατη τιμή για το Y του 3 sided query, σταματάει η αναζήτηση.
4. Αν ο κόμβος checkNode βρίσκεται στα όρια του query, προστίθεται στην λίστα resar.
5. Αλλιώς ελέγχεται αν ο κόμβος checkNode έχει και τα δύο παιδία. Σε αυτήν την περίπτωση καλείται η calcMed, που υπολογίζει την κάθετο ως προς τον Χ που χωρίστηκαν τα σημεία της λίστας. Έτσι, αν η κάθετος βρίσκεται δεξιά του query, καλείται η Tsq μόνο για το αριστερό υποδέντρο ή αν είναι δεξιά του, καλείται για το δεξί υποδέντρο.
6. Ειδάλλως, αν έχει μόνο ένα από τα δύο παιδιά, καλείται μόνο για αυτό η Tsq.

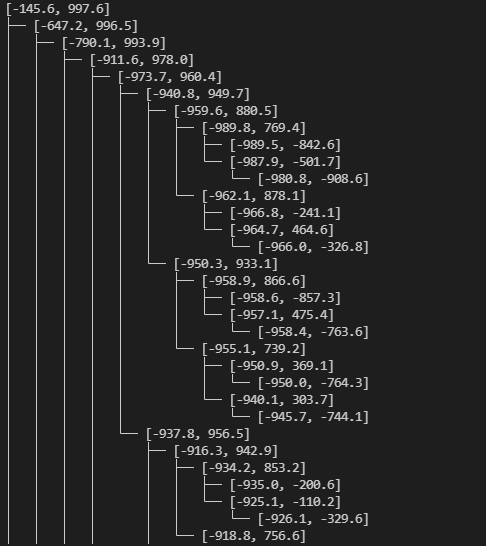
**K-D Tree:**

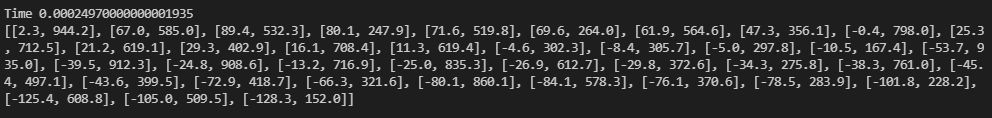
1. Αρχικά δημιουργούμε ένα αντικείμενο της κλάσης Query, το οποίο περιέχει τα όρια του query που επιθυμούμε.
2. Έπειτα, καλείται η συνάρτηση Tsq με ορίσματα το αντικείμενο query, το η ρίζα του δέντρου ή υποδέντρου που θέλουμε να περιέχει τα σημεία, την λίστα στην οποία θέλουμε να αποθηκεύσουμε το αποτέλεσμα του query και την τιμή mode, που καθορίζει αν η Υ τιμή του query είναι το άνω ή το κάτω όριο.
3. Αρχικά, αν η μεταβλητή mode ισούται με 1 και άρα το Υ είναι το κάτω όριο του query, γίνεται έλεγχος αν το σημείο checkNode βρίσκεται μέσα στα όρια. Αν βρίσκεται, προστίθεται στην λίστα των αποτελεσμάτων και αυξάνεται κατά ένα η τιμή της μεταβλητής dim.
4. Ελέγχεται αν υπάρχουν δεξί και αριστερό παιδί της checkNode και αν υπάρχουν καλείται πάλι η συνάρτηση TSQ με όρισμα τον αντίστοιχο παιδί.
5. Ομοίως γίνεται η ίδια ακριβώς διαδικασία αν το mode είναι 2.
6. Αν τα σημεία δεν ανήκουν στα όρια του query τα σημεία που μελετάμε και βρισκόμαστε στο επίπεδο που τα σημεία χωρίζονται ως προς τον Χ άξονα, ελέγχουμε αν βρίσκονται στα αριστερά της ελάχιστης τιμής του Χ, αν βρίσκονται στα δεξιά της μέγιστης τιμής του Χ ή ενδιάμεσα και καλούμε την TSQ για το δεξί, αριστερό ή και τα δύο παιδία αντίστοιχα.
7. Αν τα σημεία δεν ανήκουν στα όρια του query τα σημεία που μελετάμε και βρισκόμαστε στο επίπεδο που τα σημεία χωρίζονται ως προς τον Υ άξονα, αν το σημείο βρίσκεται στο επιθυμητό κομμάτι που ορίζει η μεταβλητή Υ καλούμε την TSQ και για τα δύο του παιδιά. Αλλιώς αν η μεταβλητή mode είναι 1 καλούμε την TSQ για το δεξί παιδί, ενώ αν είναι mode είναι 2, την καλούμε για το αριστερό παιδί.

**3)Αποτελέσματα:**

** K-D Tree:** Αποτύπωση Δέντρου K-D Tree:

Απότελεσμα 3 Sided Query και υπολογισμός χρόνου ολοκλήρωσης:

**Priority Search Tree:** Αποτύπωση Priority Search Tree:

Απότελεσμα 3 Sided Query και υπολογισμός χρόνου ολοκλήρωσης:

**Συμπεράσματα Χρονικής σύγκρισης K-D Tree και Priority Search Tree:**

Έπειτα από δειγματοληπτικές μετρήσεις για τους χρόνους τόσο για το Priority Search Tree, όσο και για το K-D Tree(αναφερόμενοι στις δικές μας υλοποιήσεις των δέντρων και των αλγορίθμων 3 sided query), καταλήξαμε στο συμπέρασμα ότι τα Priority Search Trees επιτυγχάνουν μικρότερους χρόνους σε σχέση με τα K-D Trees. Αυτό το συμπέρασμα θωρούμε πως είναι αναμενόμενο, καθώς η σημαντική διαφορά μεταξύ τους είναι πως το priority search tree είναι από την ρίζα μέχρι τα φύλλα οργανωμένο ως προς την Υ μεταβλητή και επομένως με την σύγκριση αυτής είναι πολύ πιο γρήγορος αποκλεισμός υποδέντρων. Οι κόμβοι των υποδένδρων αυτών δεν χρειάζεται να τους επισκεφθούμε και γλυτώνεται χρόνος. Ειδικά στην περίπτωση που η ρίζα του δέντρου είναι μικρότερη του κατώτατου ορίου Υ του query, η αναζήτηση τερματίζει από την πρώτη κιόλας σύγκριση.

(Η πειραματική σύγκριση έγινε με αρχείο 2000 σημείων.)