

*Πολυδιάστατες Δομές Δεδομένων*

*Εργαστηριακή Άσκηση 2021 – 2022*

*Εμμανουέλα – Ηλέκτρα Οικονομοπούλου - 1057466*

*Κωνσταντίνα Υφαντή - 1054972*

*Μιχάλης Νικολάου - 1049788*

*Άγγελος Μαργκάς -*

Πίνακας περιεχομένων

[Συλλογή Δεδομένων 3](#_Toc96127176)

[Συλλογή Δεδομένων για την δημιουργία των: KD – Tree, Range Tree 3](#_Toc96127177)

[KD – Tree 4](#_Toc96127178)

[Εισαγωγή 4](#_Toc96127179)

[Τρόπος δημιουργίας και ερμηνείας KD – δέντρου 4](#_Toc96127180)

[Διαχείριση πραγματικών δεδομένων – αρχείο kd-tree.py 7](#_Toc96127181)

[*Πειραματικά αποτελέσματα* 9](#_Toc96127182)

[10](#_Toc96127183)

[Διαχείριση 2D και 3D δεδομένων χρήστη και τυχαίων δεδομένων 11](#_Toc96127184)

[– αρχείο kd-tree1.py 11](#_Toc96127185)

[*Πειραματικά αποτελέσματα* 16](#_Toc96127186)

[18](#_Toc96127187)

[24](#_Toc96127188)

[Το KD – δέντρο: 25](#_Toc96127189)

[Quad – Tree 28](#_Toc96127190)

[Range – Tree 29](#_Toc96127191)

[Δημιουργία Range Tree – Αρχείο range\_tree.py 29](#_Toc96127192)

[Εκτέλεση πειραμάτων – Αρχείο menu.py 31](#_Toc96127193)

[R – Tree 34](#_Toc96127194)

[Διαχείριση πραγματικών δεδομένων – αρχείο R-tree.py 34](#_Toc96127195)

# *Συλλογή Δεδομένων*

## *Συλλογή Δεδομένων για την δημιουργία των: KD – Tree, Range Tree*

Τα δεδομένα που χρησιμοποιήθηκαν για την κατασκευή των δομών καθώς και για την εκτέλεση των απαιτούμενων ενεργειών, αποτελούν πραγματικά δεδομένα. Η συλλογή των δεδομένων αφορά νοσοκομεία που βρίσκονται στην Αμερική, η συλλογή έχει όνομα **kx-us-hospitals-CSV** και η λήψη τους έγινε από την ιστοσελίδα **koordinates.com .**

Το σύνολο δεδομένων αυτό αποτελείται από τα πεδία:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Latitude | Longitude | Gaz\_name | Feature\_ID | County\_name | State\_alpha | Gaz\_featureclass |

Κάθε ένα από τα παραπάνω πεδία δηλώνουν :

* **Latitude** : γεωγραφικό πλάτος
* **Longitude** : γεωγραφικό μήκος
* **Gaz\_name** : όνομα νοσοκομείου
* **Feature\_ID** : ID νοσοκομείου
* **County\_name** : κομητεία που ανήκει το νοσοκομείο
* **State\_alpha** : πολιτεία που ανήκει το νοσοκομείο
* **Gaz\_featureclass** : την κλάση που ανήκει το νοσοκομείο

Έγινε μετατροπή του αρχείου CSV σε Excel, και στην συνέχεια έγινε απαραίτητη μορφοποίηση των δεδομένων έτσι ώστε να έχουν την κατάλληλη μορφή για την ευκολότερη διαχείριση και επεξεργασία τους. Τέλος, το αρχείο μετατράπηκε σε μορφή .txt .

Το αρχείο περιέχει 14.955 διαφορετικά νοσοκομεία.

# *KD – Tree*

## *Εισαγωγή*

Δημιουργήθηκαν 3 αρχεία για την επεξεργασία – διαχείριση των δεδομένων με τη χρήση του KD – δέντρου. Το ένα αρχείο με όνομα **hospitals.py**, αποτελεί την κλάση με την οποία κάθε γραμμή του αρχείου που θα διαβαστεί, αρχείου **data**, θα δημιουργηθεί ένα object νοσοκομείου με τα χαρακτηριστικά του, όπως περιγράφονται αυτά στην προηγούμενη ενότητα «Συλλογή Δεδομένων». Στο αρχείο **kd-tree.py,** γίνεται η επεξεργασία αυτών των πραγματικών δεδομένων. Αρχικά διαβάζεται το αρχείο, δημιουργείται το KD – δέντρο και στην συνέχεια μπορεί να γίνει kNN ή range αναζήτηση με βάση τα δεδομένα αυτά. Στο αρχείο **kd-tree1.py,** γίνεται επεξεργασία δεδομένων που μπορεί να εισάγει ο χρήστης ανάλογα με τη διάσταση που επιθυμεί ( k <= 3) ή ακόμη και τυχαίων δισδιάστατων ή τρισδιάστατων δεδομένων. Στη συνέχεια, ο χρήστης έχει την επιλογή να εκτελέσει κάποια βασική πράξη του δέντρου όπως να προσθέσει έναν νέο κόμβο, να αφαιρέσει έναν κόμβο της επιλογής του , να εκτελέσει kNN και Range αναζήτηση.

## *Τρόπος δημιουργίας και ερμηνείας KD – δέντρου*

Και στις δύο περιπτώσεις δημιουργίας του δέντρου, είτε όταν οι συντεταγμένες είναι πραγματικές είτε δίνονται από τον χρήστη είτε είναι τυχαίες, το δέντρο έχει την ίδια μορφή.

Κάθε κόμβος στο δέντρο, είναι μία λίστα η οποία περιγράφεται από την σχέση:

**[Left\_Subtree, Right\_Subtree, Point\_Root]**

* Στην περίπτωση που ένα σημείο έχει την τιμή **None** σε μια από τις θέσεις Left\_Subtree ή Right\_Subtree , αυτό δηλώνει την μη ύπαρξη του αριστερού ή δεξιού υποδέντρου αντίστοιχα.
* Στην περίπτωση που ένα σημείο έχει την τιμή **None** και στις δύο θέσεις, Left\_Subtree και Right\_Subtree, αυτό δηλώνει πως ο συγκεκριμένος κόμβος αποτελεί φύλλο του δέντρου
* Στην περίπτωση που ένα σημεία έχει σημεία ή λίστες σημείων στις θέσεις Left\_Subtree και Right\_Subtree, αυτό δηλώνει πως ο κόμβος αυτός έχει και δεξί αλλά και αριστερό υποδέντρο.
* Έστω ότι έχουμε ένα δισδιάστατο δέντρο, το οποίο αποτελείται από 6 τυχαία σημεία.

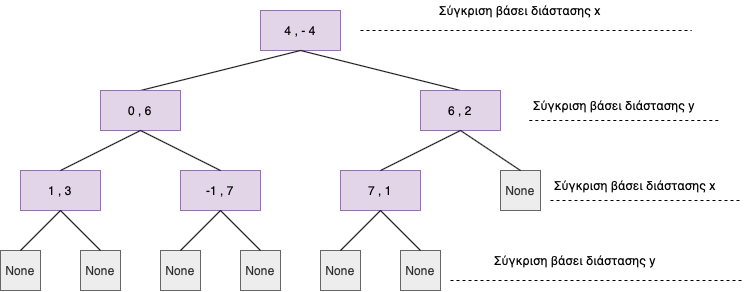
Η λίστα των σημείων αυτών είναι η : [[-1, 7], [0, 6], [1, 3], [4, -4], [6, 2], [7, 1]]

Το δέντρο δημιουργείται από αυτά τα σημεία είναι το:

[[[None, None, [1, 3]], [None, None, [-1, 7]], [0, 6]], [[None, None, [7, 1]], None, [6, 2]], [4, -4]]

|  |  |  |
| --- | --- | --- |
| **Left\_Subtree** | **Right\_Subtree** | **Point\_Root** |
| [[None, None, [1, 3]], [None, None, [-1, 7]], [0, 6]] | [[None, None, [7, 1]], None, [6, 2]] | [4 , -4] |
| [[None, None, [7, 1]] | None | [6 , 2] |
| None | None | [7 , 1] |
| [None, None, [1, 3]] | [None, None, [-1, 7]] | [0 , 6] |
| None | None | [-1 , 7] |
| None | None | [1 , 3] |

Ερμηνεία και δημιουργία του παραπάνω δέντρου:



Προφανώς οι τιμές None στην οπτικοποίηση μπορούν να παραλειφθούν στην πραγματικότητα, αλλά τις χρησιμοποιήσαμε για να γίνει ευκολότερη η κατανόηση της δομής του 2D – δέντρου.

* Έστω ότι έχουμε ένα τρισδιάστατο δέντρο, το οποίο αποτελείται από 8 τυχαία σημεία.

Η λίστα των σημείων αυτών είναι η :

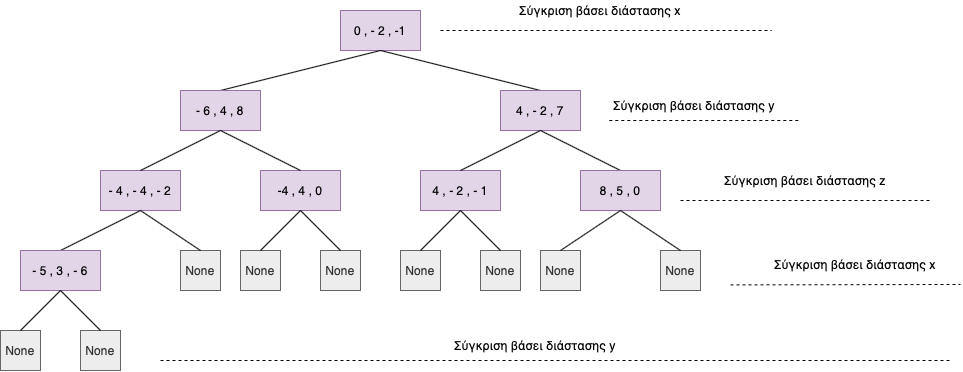
[[-6, 4, 8], [-5, 3, -6], [-4, 4, 0], [-4, -4, -2], [0, -2, -1], [4, -2, -1], [4, -2, 7], [8, 5, 0]]

Το δέντρο δημιουργείται από αυτά τα σημεία είναι το:

[[[[None, None, [-5, 3, -6]], None, [-4, -4, -2]], [None, None, [-4, 4, 0]], [-6, 4, 8]], [[None, None, [4, -2, -1]], [None, None, [8, 5, 0]], [4, -2, 7]], [0, -2, -1]]

Ερμηνεία και δημιουργία του παραπάνω δέντρου:

|  |  |  |
| --- | --- | --- |
| **Left\_Subtree** | **Right\_Subtree** | **Point\_Root** |
| [[[None, None, [-5, 3, -6]], None, [-4, -4, -2]], [None, None, [-4, 4, 0]], [-6, 4, 8]] | [[None, None, [4, -2, -1]],  [None, None, [8, 5, 0]], [4, -2, 7]] | [0, -2, -1] |
| [[None, None, [4, -2, -1]] | [None, None, [8, 5, 0]] | [4 , -2 , 7] |
| None | None | [8 , 5 , 0] |
| None | None | [4, -2, -1] |
| [[None, None, [-5, 3, -6]], None, [-4, -4, -2]] | [None, None, [-4, 4, 0]] | [-6 , 4 , 8] |
| None | None | [-4 , 4 , 0] |
| [None, None, [-5, 3, -6]] | None | [-4 , -4 , -2] |
| None | None | [-5 , 3 , -6] |



Προφανώς οι τιμές None στην οπτικοποίηση μπορούν να παραλειφθούν στην πραγματικότητα, αλλά τις χρησιμοποιήσαμε για να γίνει ευκολότερη η κατανόηση της δομής του 3D – δέντρου.

## *Διαχείριση πραγματικών δεδομένων – αρχείο kd-tree.py*

Εφόσον μιλάμε για πραγματικά δεδομένα, τα δεδομένα μας σε αυτή την περίπτωση αφού μας ενδιαφέρει το latitude και το longitude του κάθε νοσοκομείου θα είναι των 2 διαστάσεων. Για τον λόγο αυτό αρχικοποιούμε μια global μεταβλητή που αντιπροσωπεύει την διάσταση του δέντρου μας, dim = 2.

Για την διαχείριση των πραγματικών αυτών δεδομένων που αφορούν τα νοσοκομεία, δημιουργήθηκαν 7 μέθοδοι στο αρχείο:

* Η μέθοδος **main()** :

Η μέθοδος **main()**, αποτελεί την βασική μέθοδο του προγράμματος. Αρχικά διαβάζει το αρχείο με όνομα **data** και αποθηκεύει τα στοιχεία του στην μεταβλητή **file.** Στη συνέχεια γίνεται κλήση των κατάλληλων μεθόδων που θα περιγραφούν παρακάτω, ώστε να δημιουργηθεί το KD – δέντρο. Έπειτα δίνεται στο χρήστη η δυνατότητα να επιλέξει σύμφωνα με ένα μενού που εμφανίζεται στην οθόνη του, αν επιθυμεί να εκτελέσει kNN αναζήτηση ή Range αναζήτηση. Στην περίπτωση που ο χρήστης επιθυμεί να εκτελέσει kNN αναζήτηση, εμφανίζεται στην οθόνη του μήνυμα που του ζητάει να δώσει το όνομα του νοσοκομείου του οποίου θα ήθελε να βρει τους k κοντινότερους γείτονες και ακόμη ένα μήνυμα στο οποίο θα ζητά από τον χρήστη να δώσει το k, δηλαδή πόσους κοντινότερους γείτονες του νοσοκομείου που πληκτρολόγησε νωρίτερα επιθυμεί να επιστραφούν από το πρόγραμμα. Στην περίπτωση που ο χρήστης επιθυμεί να εκτελέσει Range αναζήτηση, εμφανίζονται 4 μηνύματα στην οθόνη του, που αφορούν τις minimum και maximum συντεταγμένες οι οποίες οριοθετούν το range μέσα στο οποίο τα αποτελέσματα είναι επιθυμητά (ζητά από τον χρήστη min\_x, min\_y, max\_x,max\_y).

* Η μέθοδος **put\_into\_list(file)** :

Η μέθοδος αυτή, είναι υπεύθυνη για την δημιουργία της λίστας στην οποία θα βασιστεί η επεξεργασία των δεδομένων αργότερα. Αρχικά δέχεται ως παράμετρο την μεταβλητή **file** στην οποία έχουν αποθηκευτεί τα δεδομένα από την ανάγνωση του αρχείου **data.** Αφού το αρχείο μας μέσα αποτελείται από 7 διαφορετικές στήλες, χρησιμοποιούμε 7 προσωρινές μεταβλητές, καθεμία από τις οποίες αντιπροσωπεύει την κάθε στήλη του αρχείου μας. Δημιουργούμε επίσης μια global μεταβλητή που αντιπροσωπεύει τον αριθμό των στηλών στο αρχείο μας , **NUM\_OF\_COLUMNS = 7**. Δηλώνουμε μια κενή λίστα, την **hospital\_list,** στην οποία θα αποθηκεύσουμε μορφοποιημένα τα δεδομένα μας και με βάση την οποία θα γίνει η δημιουργία του KD – δέντρου. Στην συνέχεια, μέσω εμφολευμένων επαναληπτικών δομών for, δίνουμε σε κάθε προσωρινή μεταβλητή την τιμή που πρέπει ανάλογα με την στήλη στην οποία βρίσκεται. Tέλος αποθηκεύουμε τις παραπάνω προσωρινές μεταβλητές στην λίστα **hospital\_list.**

* Η μέθοδος **get\_points(list)** :

Η μέθοδος αυτή δέχεται ως παράμετρο την λίστα hospital\_list που υπολογίστηκε από την προηγούμενη μέθοδο**.** Για την δημιουργία του KD – δέντρου, μας ενδιαφέρουν μόνο οι τιμές των πεδίων latitude και longitude. Για τον λόγο αυτό, δημιουργούμε μια νέα κενή αρχικά λίστα, την **points** ,στην οποία αποθηκεύονται μόνο τα πεδία latitude και longitude της λίστας hospital\_list.

* Η μέθοδος **construct\_kd\_tree(points, dim)** :

Η μέθοδος αυτή είναι υπεύθυνη για την δημιουργία του KD – δέντρου. Δέχεται ως παραμέτρους την λίστα **points** που δημιουργήθηκε από την προηγούμενη μέθοδο, την διάσταση του δέντρου που στην περίπτωση μας είναι 2 και το i που θα βοηθήσει στη επιλογή βάσει ποιας συντεταγμένης θα γίνει η σύγκριση των σημείων που θα μπουν στο δέντρο. Η μέθοδος αυτή, ελέγχει αρχικά αν το πλήθος των σημείων της λίστας points είναι περισσότερα του ενός. Αν η συνθήκη αυτή ισχύει, τότε μέσω της συνάρτησης lambda γίνεται ανάλογη ταξινόμηση της λίστας, επαναϋπολογίζεται το i για να γίνει βάσει της σωστής συντεταγμένης η σύγκριση, χωρίζουμε την λίστα points στη μέση, και με αναδρομικό δρόμο χτίζουμε το αριστερό και το δεξί υποδέντρο του KD – δέντρου. Στην περίπτωση ωστόσο που καταλήξουμε να έχουμε 1 στοιχείο στην λίστα μας, αυτό εισέρχεται ως φύλλο στο δέντρο.

* Η μέθοδος **dist\_sq(a, b, dim)** :

Η μέθοδος αυτή δέχεται τρείς παραμέτρους , το **α** και **b** που είναι τα σημεία που επιθυμούμε να βρούμε την απόσταση του και το **dim** που είναι η διάσταση στην οποία θέλουμε να γίνει ο υπολογισμός. Πρακτικά αυτό που κάνει είναι να υπολογίζει το άθροισμα των τετραγώνων της διαφοράς μεταξύ δύο σημείων για κάθε συντεταγμένη (την συνάρτηση αυτή την χρειαζόμαστε για τον υπολογισμό των kNN).

* Η μέθοδος **get\_knn(kd\_node, point, k ,dim, dist\_func)** :

Η μέθοδος αυτή δέχεται ως παραμέτρους το KD - δέντρο, το σημείο από το οποίο θα αναζητήσουμε τους κοντινότερους γείτονες, τον αριθμό των γειτόνων που θέλουμε να βρούμε, τις διαστάσεις και τη συνάρτηση απόστασης. Αρχικά η μέθοδος, υπολογίζει την απόσταση μεταξύ του σημείου και του κόμβου χρησιμοποιώντας τη συνάρτηση απόστασης και στη συνέχεια υπολογίζει την απόσταση μεταξύ του σημείου και του κόμβου. Στη συνέχεια ελέγχουμε αν το μέγεθος της δομής heap είναι μικρότερο από τον αριθμό των γειτόνων που αναζητούμε, αν είναι, τότε κάνουμε push την απόσταση (αλλά αρνητική) και τις συντεταγμένες του κόμβου στο σωρό. Μας ενδιαφέρει η κάθε απόσταση που θα μπει στο heap να είναι αρνητική γιατί με αυτό τον τρόπο η μεγαλύτερη απόσταση γίνεται η μικρότερη. Διαφορετικά, εάν η απόσταση είναι μικρότερη από το πρώτο στοιχείο στο heap, γίνεται push το dist και οι συντεταγμένες του κόμβου στο heap και, στη συνέχεια, γίνεται pop και επιστρέφεται το μικρότερο στοιχείο από το heap. Το μικρότερο στοιχείο είναι η μεγαλύτερη απόσταση. Έπειτα επαναϋπολογίζεται το i ώστε να γίνει rotation άξονα. Και τέλος, με αναδρομικό τρόπο καλείται η συνάρτηση και ελέγχονται το δεξί και το αριστερό υποδένδρο για την εύρεση των k κοντινότερων γειτόνων.

* Η μέθοδος **range\_search(points, range\_min, range\_max)** :

Η μέθοδος αυτή δέχεται ως παραμέτρους την λίστα των σημείων **points,**  τα **range\_min** και **range\_max** που έχει δώσει ο χρήστης από το πληκτρολόγιο. Ελέγχει ποια από τα σημεία ανήκουν μέσα σε αυτό το range τιμών και τα επιστρέφει στον χρήστη.

### *Πειραματικά αποτελέσματα*

**Για τον έλεγχο των ορθών αποτελεσμάτων, αρχικά εκτελέσαμε την επιλογή kNN αναζήτησης και επιλέξαμε τυχαία ένα νοσοκομείο από το dataset μας. Το όνομα του νοσοκομείου που επιλέχθηκε είναι το: ***Houston County Health Center*** και για αριθμό k κοντινότερων γειτόνων επιλέξαμε 5.

**

Εικόνα που περιέχει κείμενο

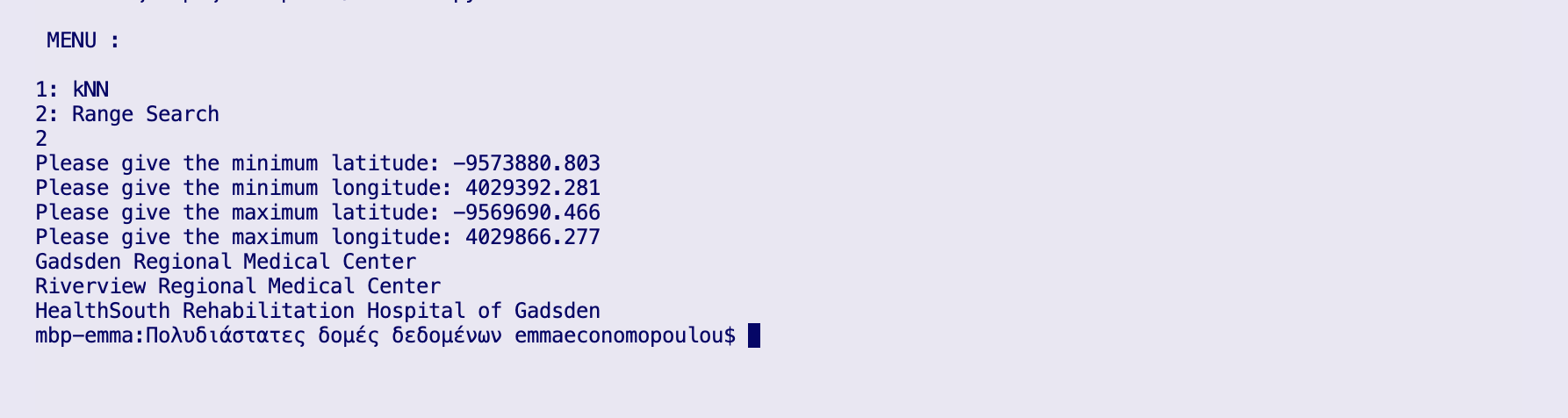
Περιγραφή που δημιουργήθηκε αυτόματα

Παρατηρούμε ότι και τα αποτελέσματα βρίσκονται στην Αλαμπάμα. Τα παραπάνω νοσοκομεία αποτελούν τους 5 κοντινότερους γείτονες του νοσοκομείου **Houston County Health Center.**

Στην συνέχεια εκτελέσαμε Range αναζήτηση μεταξύ των σημείων

Min : -9573880,803, 4029392,281

Max: -9569690,466, 4029866,276

**

## 

## 

**

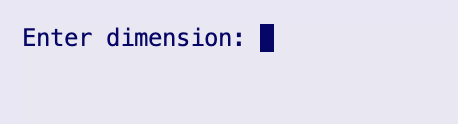
Παρατηρούμε ότι και τα τρία αποτελέσματα βρίσκονται εντός του διαστήματος που δόθηκε ως είσοδος από τον χρήστη και αποτελούν τα ορθά αποτελέσματα της Range αναζήτησης.*Διαχείριση 2D και 3D δεδομένων χρήστη και τυχαίων δεδομένων*

## *– αρχείο kd-tree1.py*

Σε αυτό το αρχείο, μπορεί να δημιουργηθεί ένα 3D - δέντρο ή ένα 2D – δέντρο από δεδομένα που είτε προέρχονται από τον χρήστη είτε από τυχαία δεδομένα. Για την διαχείριση των δεδομένων αυτών δημιουργήθηκαν 17 μέθοδοι στο αρχείο:

* Η μέθοδος **main()** :

Η μέθοδος **main()**, αποτελεί την βασική μέθοδο του προγράμματος. Αρχικά, εμφανίζεται στον χρήστη ένα μήνυμα που του ζητά να δώσει την διάσταση του KD – δέντρου που θα δημιουργηθεί.

**

Αφού ο χρήστης πληκτρολογήσει την διάσταση που επιθυμεί (είτε 2 για 2D είτε 3 για 3D), εμφανίζεται στην οθόνη του νέο μήνυμα το οποίο του ζητά να επιλέξει τον τρόπο με τον οποίο θα δημιουργηθούν οι συντεταγμένες.

**

Σε αυτή την περίπτωση ο χρήστης μπορεί να πληκτρολογήσει είτε 1 ώστε να εισάγει μόνος του τις συντεταγμένες, είτε να 2 ώστε να δημιουργηθούν τυχαίες συντεταγμένες, ανάλογα πάντα με την διάσταση που έχει επιλέξει. Έπειτα, εμφανίζεται μήνυμα το οποίο ζητάει από τον χρήστη να εισάγει το πλήθος των συντεταγμένων που θα αποτελέσουν το δέντρο.

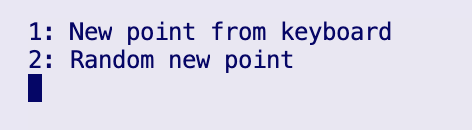
Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Αφού εκτελεστεί και αυτό το βήμα, εμφανίζεται στην οθόνη του χρήστη το KD – δέντρο που δημιουργήθηκε βάσει όλων των δεδομένων που εισήγαγε ο χρήστης μέχρι τώρα.

*Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα*Τώρα που δημιουργήθηκε το KD – δέντρο, εμφανίζεται στον χρήστη νέο μενού από το οποίο μπορεί να εκτελέσει οποιαδήποτε από τις παρακάτω ενέργειες επιθυμεί.

**Στην περίπτωση που ο χρήστης επιλέξει να προσθέσει κάποιον κόμβο στο ήδη υπάρχον δέντρο, δηλαδή πληκτρολογήσει 1, εμφανίζεται μήνυμα που του ζητά να επιλέξει αν θα εισάγει μόνος του τις συντεταγμένες του νέου σημείου ή αν επιθυμεί να εισάγει νέο τυχαίο σημείο στο δέντρο.

Ανάλογα με την επιλογή του, θα προστεθεί νέο σημείο στο δέντρο.

Στην περίπτωση που ο χρήστης επιλέξει να διαγράψει κάποιο σημείο από το ήδη υπάρχον δέντρο, δηλαδή πληκτρολογήσει το νούμερο 2 του μενού, τότε το πρόγραμμα ζητά από τον χρήστη να εισάγει τις συντεταγμένες του σημείου που επιθυμεί να διαγράψει, ανάλογα με την διάσταση του δέντρου.

Αν ο χρήστης πληκτρολογήσει το νούμερο 3 του μενού, δηλαδή επιθυμεί να αναζητήσει τους k κοντινότερους γείτονες ενός σημείου, τότε το πρόγραμμα εμφανίζει μήνυμα που ζητά από τον χρήστη να εισάγει να εισάγει τις συντεταγμένες του σημείου του οποίου θα αναζητήσουμε τους k κοντινότερους γείτονες καθώς και τον αριθμό των κοντινότερων γειτόνων, k, που επιθυμεί να του επιστρέψει το πρόγραμμα.

Αν επιλέξει ο χρήστης να εκτελέσει Range αναζήτηση, δηλαδή πληκτρολογήσει το νούμερο 5 του μενού, τότε το πρόγραμμα του ζητά να πληκτρολογήσει τις minimum και maximum συντεταγμένες οι οποίες οριοθετούν το range μέσα στο οποίο τα αποτελέσματα είναι επιθυμητά (ζητά από τον χρήστη min\_x, min\_y, max\_x, max\_y για 2 διαστάσεις και min\_x, min\_y, min\_z, max\_x, max\_y, max\_z για 3 διαστάσεις).

Τέλος, αν πληκτρολογήσει το νούμερο 5 του μενού, τότε ο χρήστης τερματίζει την εκτέλεση του προγράμματος.

* Η μέθοδος **clearConsole()** :

Δημιουργήθηκε η μέθοδος αυτή, επειδή οι είσοδοι που ζητάει το πρόγραμμα είναι πολλές, καλείται αυτή η μέθοδος ώστε να κάνει clear το τερματικό παράθυρο του χρήστη για να εμφανίζονται τα αποτελέσματα πιο ευανάγνωστα.

* Η μέθοδος **random\_coordinates(num\_coords, dim)** :

Η μέθοδος αυτή καλείται στην περίπτωση που ο χρήστης έχει επιλέξει τα σημεία που θα αποτελέσουν το δέντρο να είναι τυχαία. Η μέθοδος δέχεται δύο παραμέτρους, η πρώτη αφορά το πλήθος των τυχαίων συντεταγμένων που θα δημιουργηθούν και η δεύτερη παράμετρος αφορά την διάσταση των συντεταγμένων που θα δημιουργηθούν. Με μία επαναληπτική δομή for που θα εκτελεστεί τόσες φορές όσες το πλήθος των συντεταγμένων που επιθυμεί ο χρήστη να έχει το δέντρο, δημιουργούνται οι τυχαίες συντεταγμένες ανάλογα με την διάσταση του δέντρου.

* Η μέθοδος **keyboard\_coordinates(num\_coords, dim)** :

Η μέθοδος αυτή καλείται στην περίπτωση που ο χρήστης έχει επιλέξει τα σημεία που θα αποτελέσουν το δέντρο να δοθούν από τον ίδιο. Η μέθοδος δέχεται δύο παραμέτρους, η πρώτη αφορά το πλήθος των τυχαίων συντεταγμένων που θα δημιουργηθούν και η δεύτερη παράμετρος αφορά την διάσταση των συντεταγμένων που θα δημιουργηθούν. Με μία επαναληπτική δομή for που θα εκτελεστεί τόσες φορές όσες το πλήθος των συντεταγμένων που επιθυμεί ο χρήστη να έχει το δέντρο, ο χρήστης κλείνεται να δώσει τις συντεταγμένες για κάθε ένα από τα σημεία που θα δημιουργήσουν το δέντρο, ανάλογα με την διάσταση του δέντρου.

* Η μέθοδος **construct\_kd\_tree(points, dim)** :

Η μέθοδος αυτή είναι υπεύθυνη για την δημιουργία του KD – δέντρου. Δέχεται ως παραμέτρους την λίστα **points** που δημιουργήθηκε από την προηγούμενη μέθοδο, την διάσταση του δέντρου που στην περίπτωση μας είναι 2 και το i που θα βοηθήσει στη επιλογή βάσει ποιας συντεταγμένης θα γίνει η σύγκριση των σημείων που θα μπουν στο δέντρο. Η μέθοδος αυτή, ελέγχει αρχικά αν το πλήθος των σημείων της λίστας points είναι περισσότερα του ενός. Αν η συνθήκη αυτή ισχύει, τότε μέσω της συνάρτησης lambda γίνεται ανάλογη ταξινόμηση της λίστας, επαναϋπολογίζεται το i για να γίνει βάσει της σωστής συντεταγμένης η σύγκριση, χωρίζουμε την λίστα points στη μέση, και με αναδρομικό δρόμο χτίζουμε το αριστερό και το δεξί υποδέντρο του KD – δέντρου. Στην περίπτωση ωστόσο που καταλήξουμε να έχουμε 1 στοιχείο στην λίστα μας, αυτό εισέρχεται ως φύλλο στο δέντρο.

* Η μέθοδος **new\_point\_keyboard( dim)** :

Η μέθοδος αυτή καλείται όταν ο χρήστης επιλέξει να εισάγει ένα νέο σημείο στο ήδη υπάρχον δέντρο, και επιλέξει το νέο σημείο που θα εισαχθεί στο δέντρο να το επιλέξει ο ίδιος. Η μέθοδος αυτή έχει μια παράμετρο που είναι η διάσταση του δέντρου. Ανάλογα με αυτή τη διάσταση, ο χρήστης θα κληθεί να εισάγει τις συντεταγμένες του νέου σημείου που επιθυμεί να εισάγει στο δέντρο.

* Η μέθοδος **new\_random\_point( dim)** :

Η μέθοδος αυτή καλείται όταν ο χρήστης επιλέξει να εισάγει ένα νέο σημείο στο ήδη υπάρχον δέντρο, και επιλέξει το νέο σημείο που θα εισαχθεί στο δέντρο να είναι τυχαίο. Η μέθοδος αυτή έχει μια παράμετρο που είναι η διάσταση του δέντρου. Ανάλογα με αυτή τη διάσταση, το πρόγραμμα δημιουργεί τυχαίες συντεταγμένες του νέου σημείου που θα εισαχθεί στο δέντρο.

* Η μέθοδος **add\_point( kd\_node, point, dim)** :

Η μέθοδος αυτή είναι υπεύθυνη για την εισαγωγή του νέου σημείου στο δέντρο, είτε αυτό το σημείο δόθηκε από τον χρήστη είτε είναι τυχαίο. Η μέθοδος αυτή δέχεται τρείς παραμέτρους, η πρώτη είναι το ήδη υπάρχον KD – δέντρο, η δεύτερη είναι το νέο σημείο που θα εισαχθεί στο δέντρο και η τρίτη είναι η διάσταση του δέντρου. Αρχικά ελέγχεται αν το δέντρο δεν είναι κενό, αν ικανοποιηθεί αυτή η συνθήκη, τότε υπολογίζεται η απόσταση μεταξύ του νέου σημείου και της ρίζας. Έπειτα, στην μεταβλητή i που αρχικά είναι ίση με 0, (για i = 0 συγκρίνεται η διάσταση x, για i =1 συγκρίνεται το y και αν υπάρχει z τότε i =2 ) γίνεται rotation του άξονα έτσι ώστε σε κάθε κλήση της μεθόδου να συγκρίνονται οι σωστές συντεταγμένες. Και τέλος, με αναδρομικό τρόπο καλείται η μέθοδος αυτή και ελέγχονται το δεξί και το αριστερό υποδένδρο για την εύρεση της σωστής θέσης για την εισαγωγή του νέου σημείου.

* Η μέθοδος **dist\_sq(a, b, dim)** :

Η μέθοδος αυτή δέχεται τρείς παραμέτρους , το **α** και **b** που είναι τα σημεία που επιθυμούμε να βρούμε την απόσταση του και το **dim** που είναι η διάσταση στην οποία θέλουμε να γίνει ο υπολογισμός. Πρακτικά αυτό που κάνει είναι να υπολογίζει το άθροισμα των τετραγώνων της διαφοράς μεταξύ δύο σημείων για κάθε συντεταγμένη (την συνάρτηση αυτή την χρειαζόμαστε για τον υπολογισμό των kNN).

* Η μέθοδος **get\_knn(kd\_node, point, k ,dim, dist\_func)** :

Η μέθοδος αυτή καλείται στην περίπτωση που ο χρήστης επιλέξει να κάνει kNN αναζήτηση. Δέχεται ως παραμέτρους το KD - δέντρο, το σημείο από το οποίο θα αναζητήσουμε τους κοντινότερους γείτονες, τον αριθμό των γειτόνων που θέλουμε να βρούμε, τις διαστάσεις και τη συνάρτηση απόστασης. Αρχικά η μέθοδος , υπολογίζει την απόσταση μεταξύ του σημείου και του κόμβου χρησιμοποιώντας τη συνάρτηση απόστασης και στη συνέχεια υπολογίζει την απόσταση μεταξύ του σημείου και του κόμβου. Στη συνέχεια ελέγχουμε αν το μέγεθος της δομής heap είναι μικρότερο από τον αριθμό των γειτόνων που αναζητούμε, αν είναι, τότε κάνουμε push την απόσταση (αλλά αρνητική) και τις συντεταγμένες του κόμβου στο σωρό. Μας ενδιαφέρει η κάθε απόσταση που θα μπει στο heap να είναι αρνητική γιατί με αυτό τον τρόπο η μεγαλύτερη απόσταση γίνεται η μικρότερη. Διαφορετικά, εάν η απόσταση είναι μικρότερη από το πρώτο στοιχείο στο heap, γίνεται push το dist και οι συντεταγμένες του κόμβου στο heap και, στη συνέχεια, γίνεται pop και επιστρέφεται το μικρότερο στοιχείο από το heap. Το μικρότερο στοιχείο είναι η μεγαλύτερη απόσταση. Έπειτα επαναϋπολογίζεται το i ώστε να γίνει rotation άξονα. Και τέλος, με αναδρομικό τρόπο καλείται η συνάρτηση και ελέγχονται το δεξί και το αριστερό υποδένδρο για την εύρεση των k κοντινότερων γειτόνων.

* Η μέθοδος **range\_search(points, range\_min, range\_max)** :

Η μέθοδος αυτή καλείται στην περίπτωση που ο χρήστης επιλέξει να κάνει Range αναζήτηση. Δέχεται ως παραμέτρους την λίστα των σημείων **points,**  τα **range\_min** και **range\_max** που έχει δώσει ο χρήστης από το πληκτρολόγιο. Ελέγχει ποια από τα σημεία ανήκουν μέσα σε αυτό το range τιμών και τα επιστρέφει στον χρήστη.

* Η μέθοδος **delete\_point(points, point)** :

Η μέθοδος αυτή καλείται όταν ο χρήστης επιλέξει να διαγράψει έναν σημείο από το δέντρο. Δέχεται δύο παραμέτρους, η πρώτη είναι τα σημεία του KD - δέντρου και η δεύτερη είναι το σημείο που επιθυμεί ο χρήστης να διαγράψει από το δέντρο. Αρχικά, ελέγχεται αν το σημείο που επιθυμεί ο χρήστης να διαγράψει ανήκει στο KD – δέντρο, αν ανήκει τότε αφαιρεί το σημείο αυτό και με αναδρομικό τρόπο ξαναδημιουργείται το δεξί και το αριστερό υποδέντρο του KD – δέντρου. Αν το σημείο δεν ανήκει στο KD – δέντρο, τότε εμφανίζεται μήνυμα λάθους στον χρήστη.

* Η μέθοδος **plot\_2Dpoints\_knn(points, knn\_list)** :

Η μέθοδος αυτή, καλείται όταν ο χρήστης επιλέξει να εφαρμόσει kNN αναζήτηση. Δέχεται δύο παραμέτρους, η πρώτη είναι η λίστα των σημείων του KD – δέντρου και η δεύτερη είναι η λίστα με τους κοντινότερους γείτονες του σημείου που ο χρήστης επέλεξε να βρει τους kNN. Αυτό που κάνει αυτή η μέθοδος, είναι να οπτικοποιήσει τα σημεία στις δύο διαστάσεις x και y, στην περίπτωση που η εκτέλεση του προγράμματος αφορά τον δισδιάστατο χώρο. Το σημείο που επιλέχτηκε από τον χρήστη για να βρεθούν οι k κοντινότεροι γείτονες εμφανίζεται με πράσινο χρώμα στο διάγραμμα, οι κοντινότεροι του γείτονες εμφανίζονται με κόκκινο χρώμα και τα υπόλοιπα σημεία εμφανίζονται με μπλε χρώμα.

* Η μέθοδος **plot\_3Dpoints\_knn(points, knn\_list)** :

Η μέθοδος αυτή, καλείται όταν ο χρήστης επιλέξει να εφαρμόσει kNN αναζήτηση. Δέχεται δύο παραμέτρους, η πρώτη είναι η λίστα των σημείων του KD – δέντρου και η δεύτερη είναι η λίστα με τους κοντινότερους γείτονες του σημείου που ο χρήστης επέλεξε να βρει τους kNN. Αυτό που κάνει αυτή η μέθοδος, είναι να οπτικοποιήσει τα σημεία στις τρεις διαστάσεις x, y και z, στην περίπτωση που η εκτέλεση του προγράμματος αφορά τον τρισδιάστατο χώρο. Το σημείο που επιλέχτηκε από τον χρήστη για να βρεθούν οι k κοντινότεροι γείτονες εμφανίζεται με πράσινο χρώμα στο διάγραμμα, οι κοντινότεροι του γείτονες εμφανίζονται με κόκκινο χρώμα και τα υπόλοιπα σημεία εμφανίζονται με μπλε χρώμα.

* Η μέθοδος **plot\_2Dpoints\_ range(points, range\_min, range\_max)** :

Η μέθοδος αυτή, καλείται όταν ο χρήστης επιλέξει να εφαρμόσει Range αναζήτηση. Δέχεται τρείς παραμέτρους, η πρώτη είναι η λίστα των σημείων του KD – δέντρου και η δεύτερη είναι η λίστα με τις minimum συντεταγμένες x και y και η τρίτη είναι λίστα με τις maximum συντεταγμένες x και y έτσι ώστε να δημιουργήσουμε ένα εύρος μέσα στο οποίο θα εμφανίζονται τα επιθυμητά σημεία που ικανοποιούν την αναζήτηση αυτή. Αυτό που κάνει αυτή η μέθοδος, είναι να οπτικοποιήσει τα σημεία στις δύο διαστάσεις x και y, στην περίπτωση που η εκτέλεση του προγράμματος αφορά τον δισδιάστατο χώρο. Τα σημεία τα οποία βρίσκονται μέσα σε αυτό το εύρος τιμών που δόθηκε από τον χρήστη εμφανίζονται με κόκκινο χρώμα και τα υπόλοιπα σημεία εμφανίζονται με μπλε χρώμα.

* Η μέθοδος **plot\_3Dpoints\_ range(points, range\_min, range\_max)** :

Η μέθοδος αυτή, καλείται όταν ο χρήστης επιλέξει να εφαρμόσει Range αναζήτηση. Δέχεται τρείς παραμέτρους, η πρώτη είναι η λίστα των σημείων του KD – δέντρου και η δεύτερη είναι η λίστα με τις minimum συντεταγμένες x, y και z και η τρίτη είναι λίστα με τις maximum συντεταγμένες x, y και z έτσι ώστε να δημιουργήσουμε ένα εύρος μέσα στο οποίο θα εμφανίζονται τα επιθυμητά σημεία που ικανοποιούν την αναζήτηση αυτή. Αυτό που κάνει αυτή η μέθοδος, είναι να οπτικοποιήσει τα σημεία στις τρεις διαστάσεις x και y και z, στην περίπτωση που η εκτέλεση του προγράμματος αφορά τον τρισδιάστατο χώρο. Τα σημεία τα οποία βρίσκονται μέσα σε αυτό το εύρος τιμών που δόθηκε από τον χρήστη εμφανίζονται με κόκκινο χρώμα και τα υπόλοιπα σημεία εμφανίζονται με μπλε χρώμα.

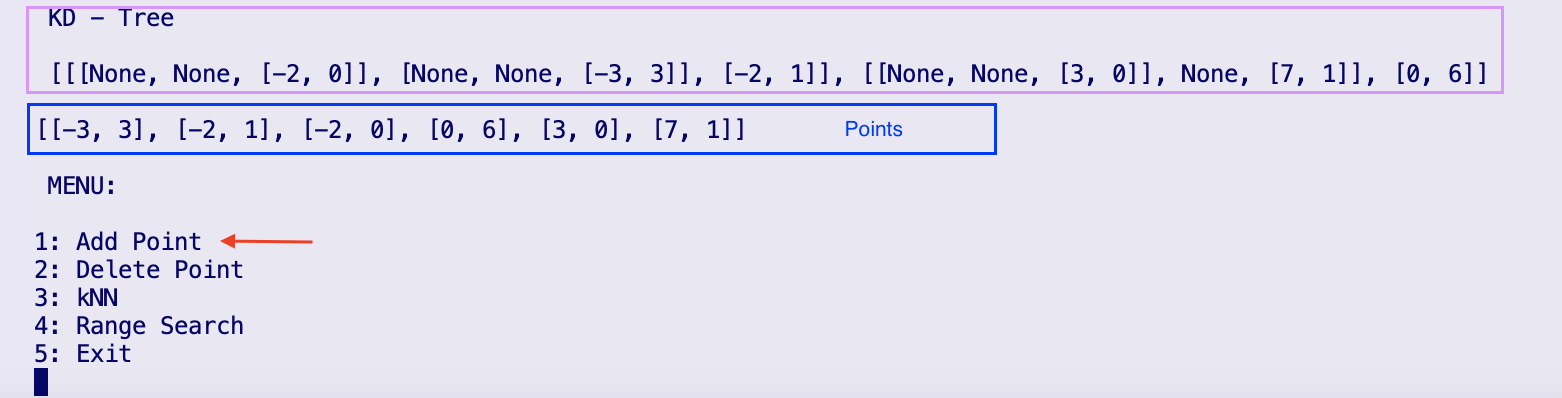
### *Πειραματικά αποτελέσματα*

Αρχικά θα εκτελέσουμε κάθε μία από τις επιλογές του χρήστη για 2 διαστάσεις για λίγα σημεία για να ελέγξουμε πιο εύκολα την ορθότητα των αποτελεσμάτων μας και στην συνέχεια για μεγάλο πλήθος σημείων.

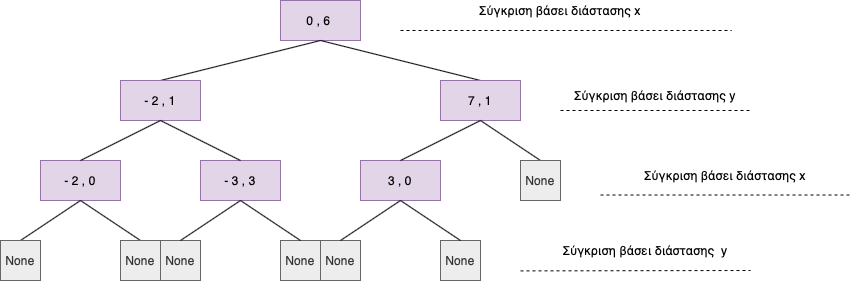
**Πείραμα για δημιουργία 2D – δέντρου και εκτέλεση ενεργειών:**

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

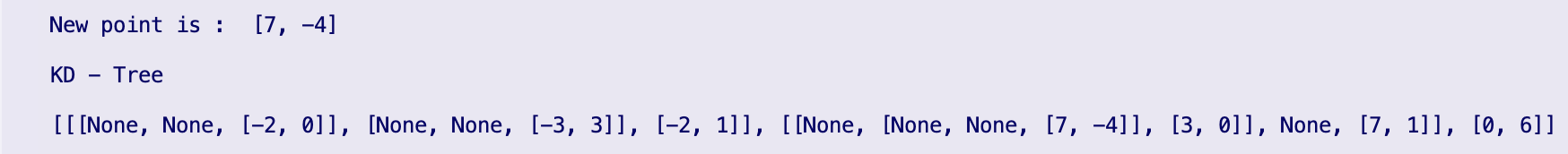


Οπτικοποίηση 2D – δέντρου



***Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα*Προσθέτουμε τυχαίο σημείο:**

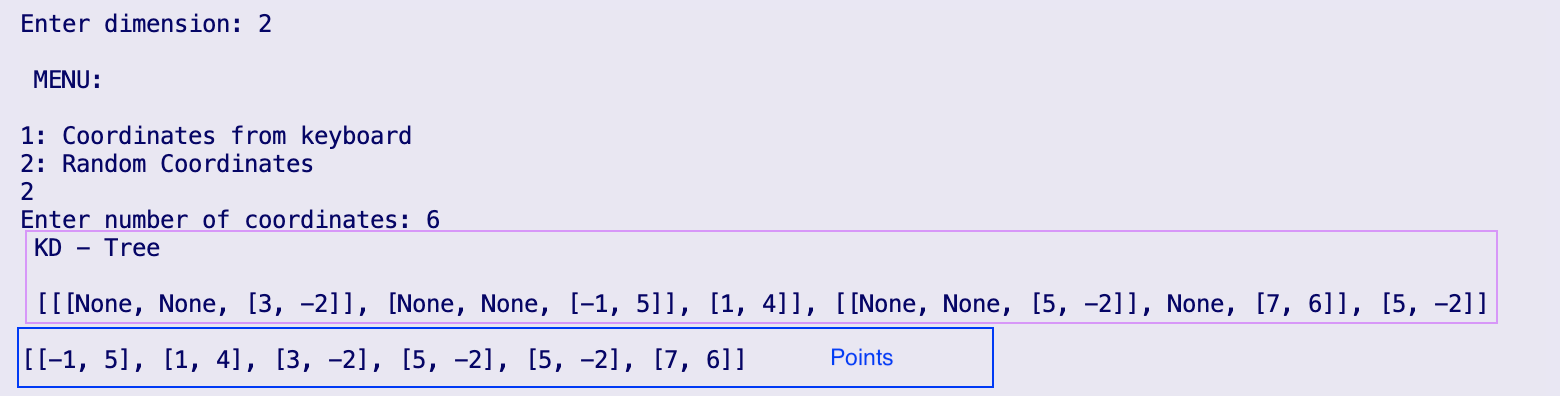
**Το KD – δέντρο μετά την εισαγωγή νέου σημείου:

Η οπτικοποίηση του KD – δέντρου μετά την εισαγωγή:

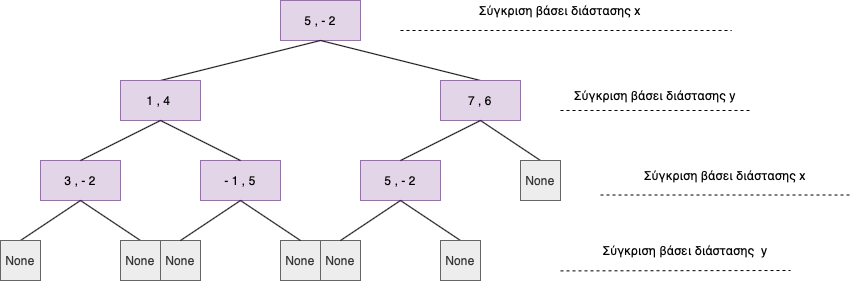
# 

Ακριβώς τα ίδια αποτελέσματα θα λαμβάναμε και αν η εισαγωγή γινόταν με την επιλογή να πληκτρολογήσει ο χρήστης τις νέες συντεταγμένες.

**Τώρα ο χρήστης επιλέγει να διαγράψει ένα σημείο**:



Οπτικοποίηση KD – δέντρου:

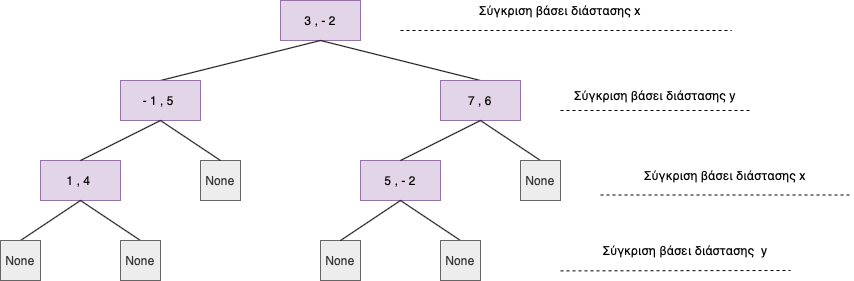


Το KD - δέντρο μετά την διαγραφή του σημείου :

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

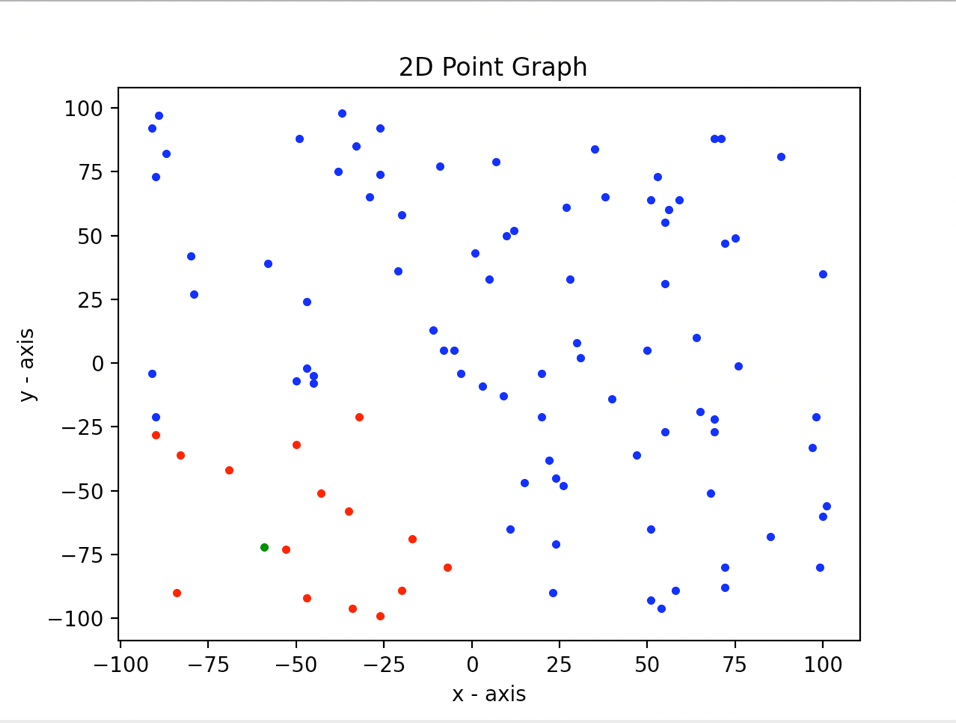
Η οπτικοποίηση του KD – δέντρου μετά την εισαγωγή:

**

**Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματαΕικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματαΟ χρήστης επιλέγει να εκτελέσει kNN αναζήτηση:**

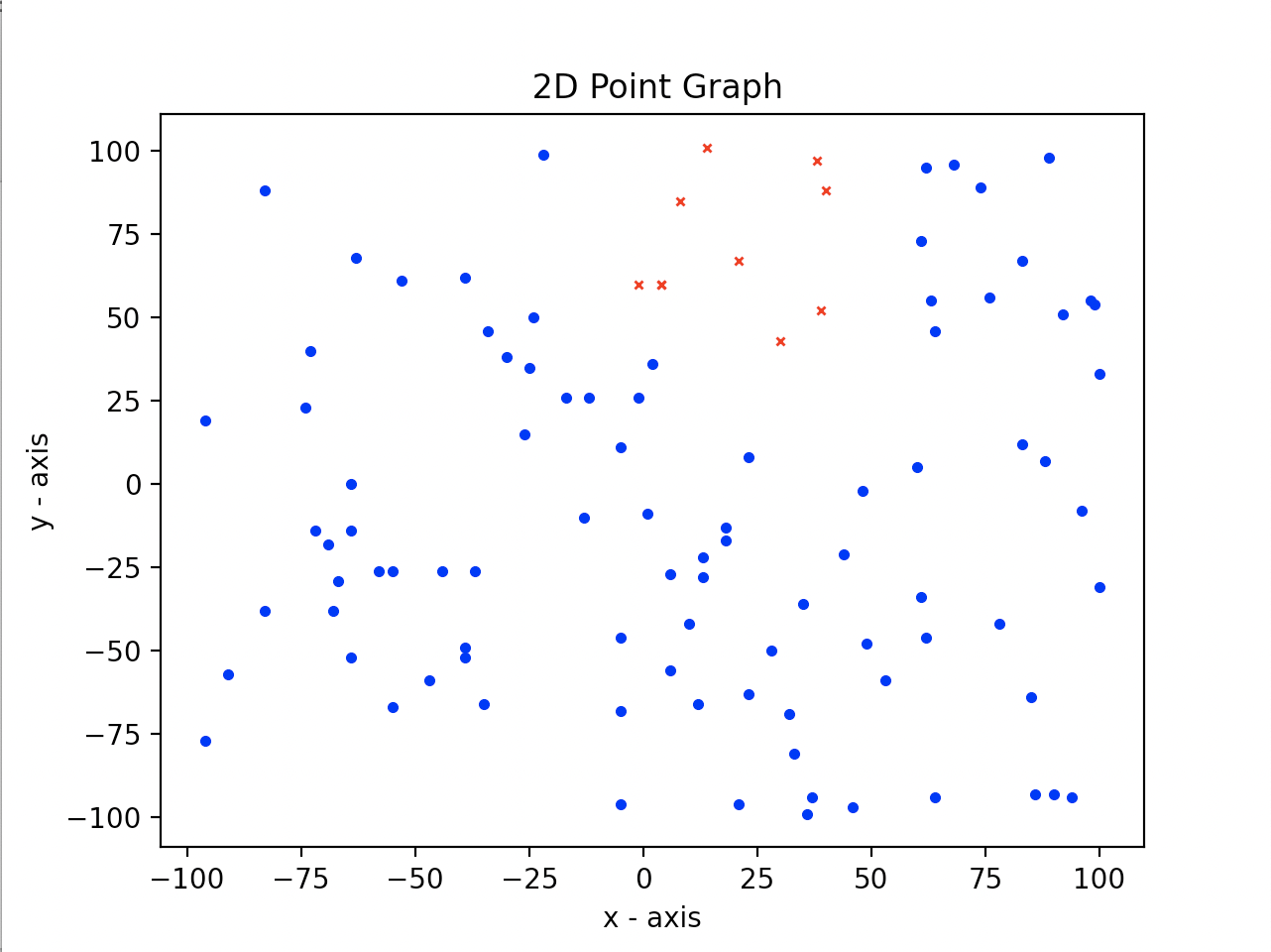
****

**Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματαΟ χρήστης επιλέγει να εκτελέσει Range αναζήτηση:**

**Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα**

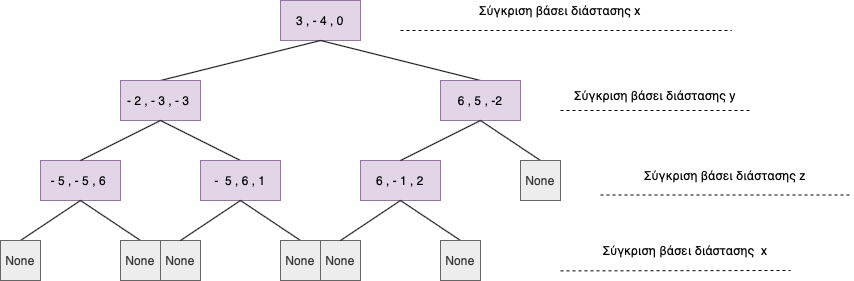
****

**Πείραμα για δημιουργία 3D – δέντρου και εκτέλεση ενεργειών:**

**Επιλογή χρήστη για προσθήκη νέου σημείου στο KD – δέντρο:**

*Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα*

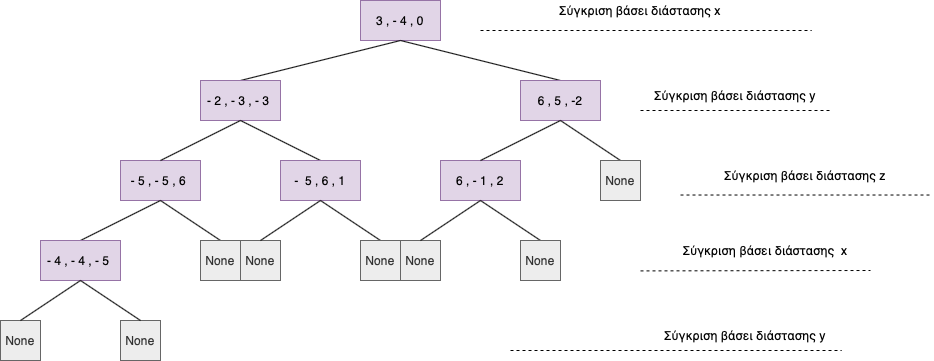
**Οπτικοποίηση KD – δέντρου:

Το KD – δέντρο μετά την εισαγωγή νέου σημείου:

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Οπτικοποίηση KD – δέντρου μετά την εισαγωγή σημείου :

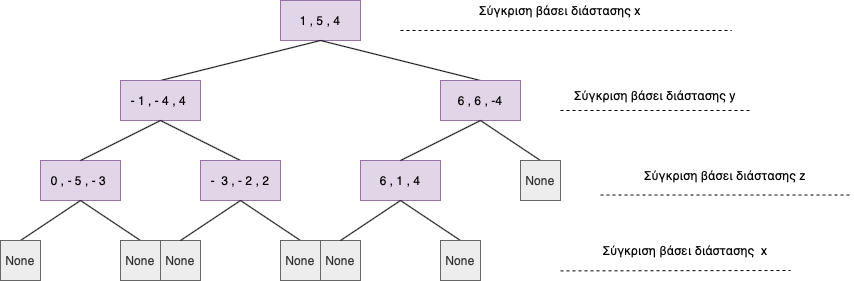


**Επιλογή χρήστη για διαγραφή σημείου στο KD – δέντρο:**

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Οπτικοποίηση KD – δέντρου :

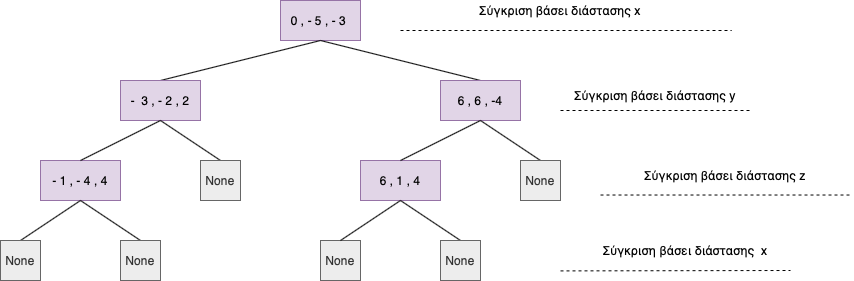


Το KD – δέντρο μετά την διαγραφή σημείου:

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Οπτικοποίηση KD – δέντρου μετά την διαγραφή σημείου :

**

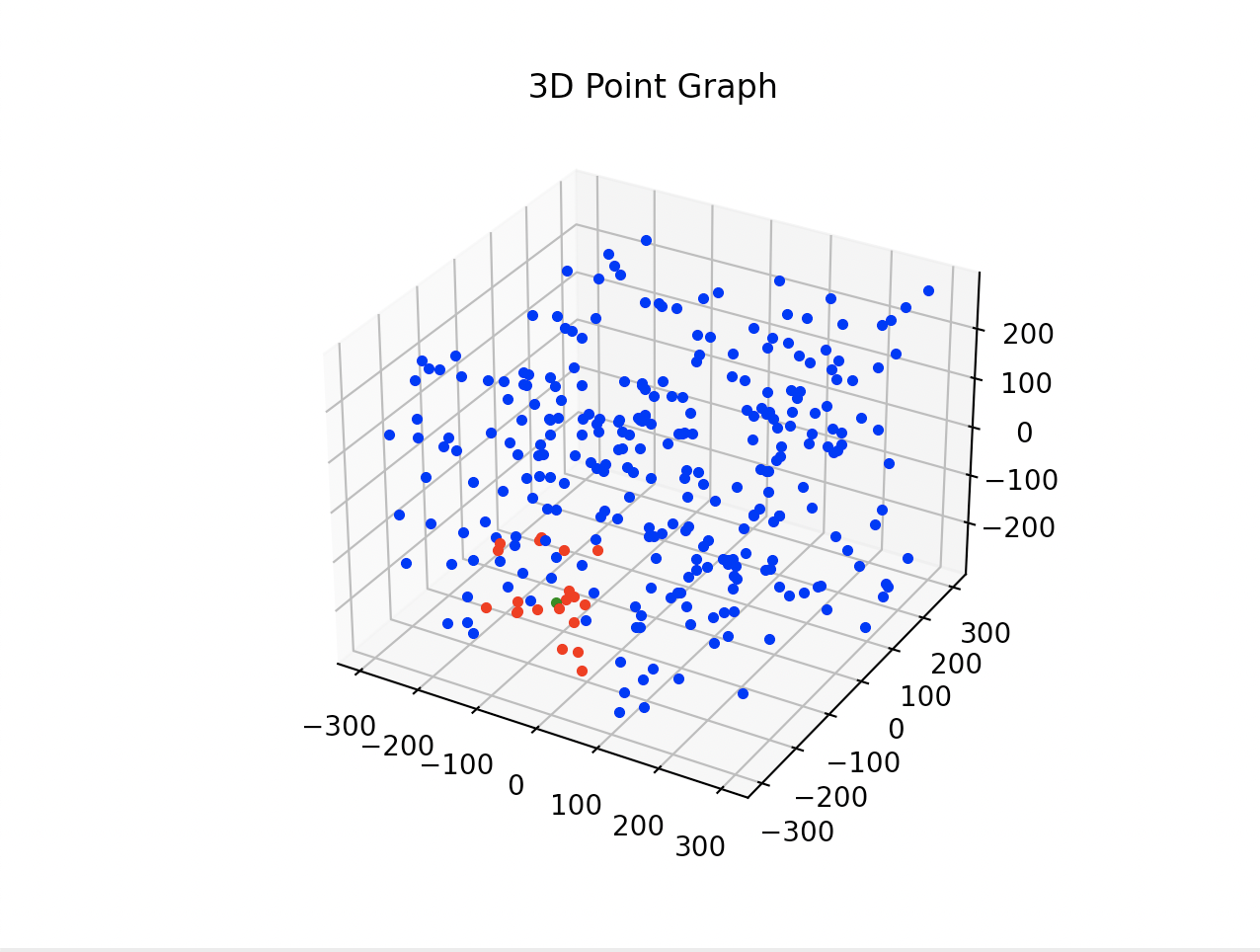
**Επιλογή χρήστη kNN αναζήτησης στο KD – δέντρο:**

# **Εικόνα που περιέχει κείμενο Περιγραφή που δημιουργήθηκε αυτόματα**

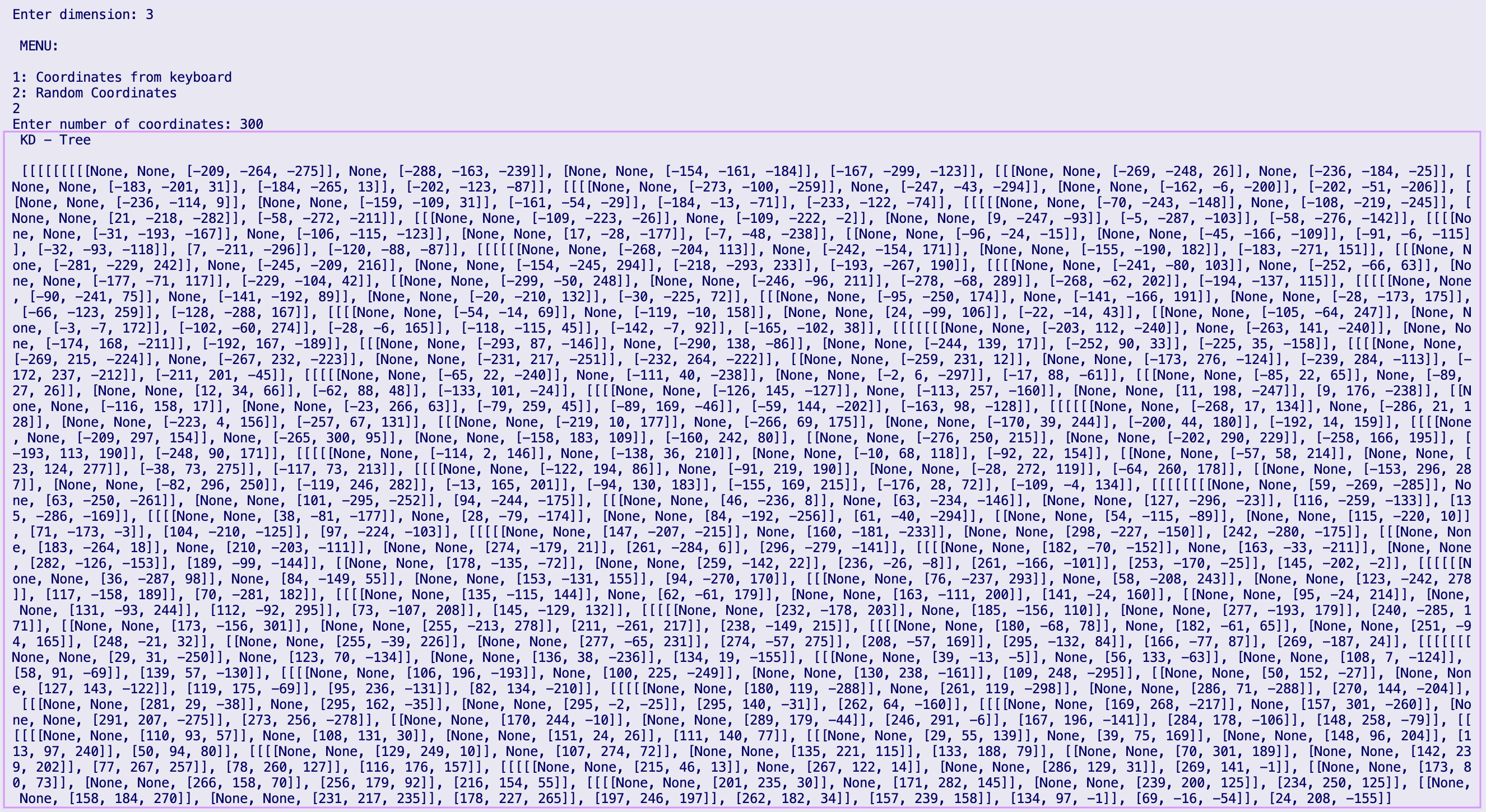
# Το KD – δέντρο:

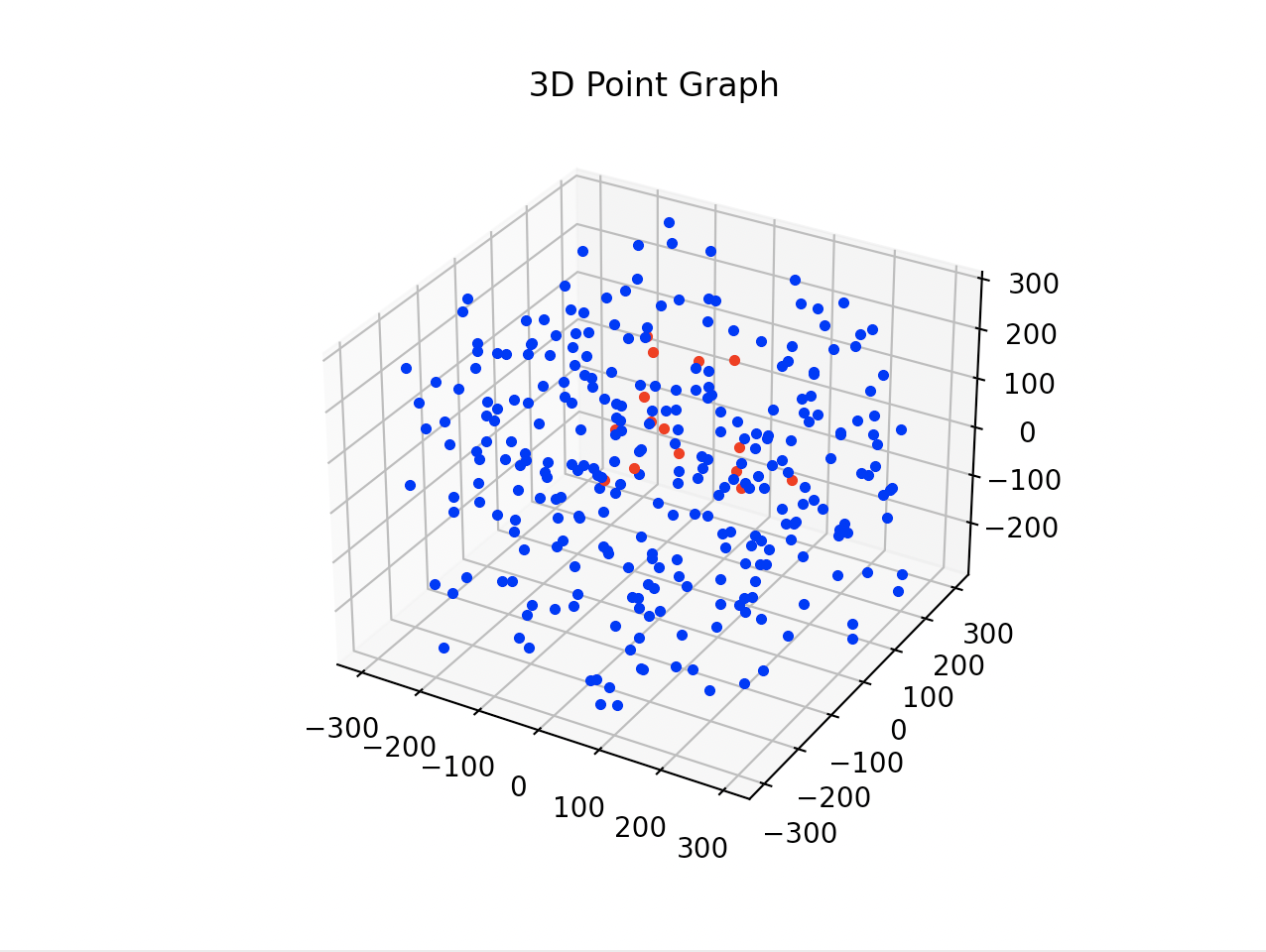
Τα σημεία του KD – δέντρου:

**

**

**Επιλογή χρήστη Range αναζήτησης στο KD – δέντρο:**

**

*Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα*

# *Quad – Tree*

# *Range – Tree*

## Δημιουργία Range Tree – Αρχείο *range\_tree.py*

Στο αρχείο *range\_tree.py* βρίσκονται όλες οι συναρτήσεις για τη δημιουργία του δέντρου, αλλά και τις απαραίτητες πράξεις: *insert, delete, update, range search.* Στη συνέχεια, θα μελετήσουμε τη κάθε μια από αυτές τις συναρτήσεις ξεχωριστά, ωστόσο πρώτα αξίζει να αναφέρουμε επιγραμματικά κάποια σημεία της θεωρίας.

Diagram

Description automatically generatedΈνα range tree είναι μια διατεταγμένη δομή δεδομένων με σκοπό να διατηρεί μια λίστα σημείων. Επιτρέπει την αποτελεσματική αναφορά όλων των σημείων, εντός ενός δεδομένου εύρους και συνήθως χρησιμοποιείται σε δύο ή περισσότερες διαστάσεις, ωστόσο δεν γενικέυει πάντα καλά. Συγκεκριμένα, για ένα σύνολο μονοδιάστατων σημείων, θα είναι ένα ισορροπημένο δυαδικό δέντρο αναζήτησης (BST) σε αυτά τα σημεία. Επεκτείνοντας αυτή την ιδέα, ένα *d*-διάστατο range tree είναι ένα *αναδρομικά καθορισμένο πολυεπίπεδο δυαδικό δέντρο αναζήτησης*. Κάθε επίπεδο της δομής δεδομένων, είναι ένα BST σε μία από τις διαστάσεις *d*. Το πρώτο επίπεδο, είναι ένα δυαδικό δέντρο αναζήτησης στην πρώτη από τις συντεταγμένες *d*. Κάθε κορυφή *v* αυτού του δέντρου περιέχει μια συσχετισμένη δομή που είναι ένα δέντρο εύρους διαστάσεων *(d−1)* στις τελευταίες *(d−1)-*συντεταγμένες των σημείων που είναι αποθηκευμένα στο υποδέντρο του *v*. Τα *range trees*, επιπλέον, αποτελούν μια εναλλακτική των *KD* δέντρων. Σε σύγκριση με τα τελευταία, τα *range trees* προσφέρουν ταχύτερο *query time*, όπου με χρήση *Big O annotation*, αντιστοιχεί σε: , αλλά χειρότερη αποθήκευση, με , όπου *n* είναι ο αριθμός των δεδομένων που είναι αποθηκευμένα στο δέντρο, *d* είναι η διάσταση κάθε σημείου και *k* είναι ο αριθμός των σημείων που αναφέρονται από ένα δεδομένο ερώτημα.

Στην υλοποίηση μας, δηλώνουμε αρχικά τις διαστάσεις του δέντρου με την μεταβλητή *DIMENSIONS*.

***class Node***

Δημιουργούμε ένα object Node, για κάθε κόμβο, όπου θα έχει τις εξής ιδιότητες:

* coords
* name
* left
* right
* next\_dimension

***create\_range\_tree***

Με αυτή τη συνάρτηση κατασκευάζουμε το δέντρο. Δέχεται σαν όρισμα την λίστα με τις σορταρισμένες συντεταγμένες των κόμβων και επιστρέφει τη ρίζα του δέντρου (ως Node object) και μια λίστα με τις συντεταγμένες για το δέντρο επόμενης διάστασης.

Καθώς οι συντεταγμένες είναι διατεταγμένες, διαλέγουμε την μεσαία για τη ρίζα του δέντρου και καλούμε πάλι εσωτερικά την συνάρτηση για τους πρώτους μισούς κόμβους ως το αριστερό υποδέντρο και τους υπόλοιπους για το δεξί υποδέντρο. Χρησιμοποιούμε την (αμέσως επόμενη) συνάρτηση *merge*, ώστε να στοιχίσουμε τις δυο λίστες μαζί με την εκάστοτε ρίζα, ως προς την επόμενη διάσταση και να δημιουργήσουμε το δέντρο που αντιστοιχεί σε αυτήν.

***merge***

Όπως αναφέρθηκε παραπάνω, κατασκευάστηκε η συνάρτηση merge, που παίρνει ως ορίσματα την λίστα του αριστερού και δεξί υποδέντρου και την ρίζα για την επόμενη διάσταση. Αρχικά, στοιχίζει τις δύο λίστες με βάση τους δείκτες και μετά βρίσκει τη σωστή θέση για τη ρίζα.

***insert***

Εδώ εισάγουμε ένα νέο κόμβο στο δέντρο μας. Δίνουμε ως όρισμα τη ρίζα και τον νέο κόμβο ως Node object. Συγκρίνοντας κάθε φορά με τις συντεταγμένες της ρίζας, αποφασίζουμε τη θέση του νέου κόμβου και τον προσθέτουμε, για κάθε διάσταση. Έξοδος είναι η ρίζα του νέου δέντρου.

***leftmost\_node***

Η συνάρτηση αυτή είναι βοηθητική για την *delete.* Δίνοντας σαν είσοδο ένα κόμβο βρίσκουμε το πιο αριστερό παιδί του.

***delete***

Με την *delete* πραγματοποιούμε την πράξη της διαγραφής. Παίρνει ως είσοδο την ρίζα του δέντρου και τις συντεταγμένες του κόμβου που θέλουμε να διαγράψουμε και επιστρέφει την ρίζα του νέου δένδρου. Καλούμε εσωτερικά την συνάρτηση στο αριστερό ή δεξί υποδέντρο, ανάλογα με τη σχέση των συντεταγμένων που δόθηκαν με τη ρίζα. Όταν βρεθεί ο κόμβος, τον διαγράφουμε δίνοντας του τιμή *None*. Για να διαγράψουμε τον κόμβο, όμως, πρέπει να βρούμε αν έχει παιδιά και πόσα, ώστε να διορθωθεί το δέντρο σύμφωνα με τη θεωρία μας.

***search***

Με τη συγκεκριμένη συνάρτηση βρίσκουμε ένα συγκεκριμένο κόμβο, δίνοντας την ρίζα του δέντρου και τις συντεταγμένες του. Για να κατευθυνθούμε μέσα στο δέντρο συγκρίνουμε και πάλι με τις συντεταγμένες του root. Όταν βρούμε τον κόμβο ελέγχουμε και αριστερά του για τυχόν όμοιους. Ως έξοδο παίρνουμε μια λίστα με τους κόμβους που επέστρεψε η συνάρτηση.

***find\_split\_node***

Η συγκεκριμένη αποτελεί βοηθητική συνάρτηση για την *range search.* Θέλουμε να βρούμε το κόμβο στον οποίο χωρίζουν τα μονοπάτια αναζήτησης για το *min, max* του διαστήματος. Παίρνει ως είσοδο τη ρίζα και το διάστημα αναζήτησης και επιστρέφει τον ζητούμενο κόμβο. Γίνεται κλήση της συνάρτησης εσωτερικά και ελέγχοντας κάθε φορά τη σχέση του εκάστοτε root με τα *min, max* καταλήγουμε στο *split node.*

***is\_in\_range***

Δίνοντας ως είσοδο συντεταγμένες και το διάστημα αναζήτησης, παίρνουμε ως έξοδο *True* αν οι συντεταγμένες ανήκουν στο διάστημα, αλλιώς *False*. Λειτουργεί, επίσης, ως βοηθητική συνάρτηση για την *range search.*

***range\_search***

Με αυτή τη συνάρτηση θέλουμε να βρούμε όλους τους κόμβους που βρίσκονται μέσα σε ένα διάστημα. Δίνουμε ως είσοδο την ρίζα και το ζητούμενο διάστημα και παίρνουμε στην έξοδο μια λίστα με όλους τους κόμβους. Χρησιμοποιώντας τις δύο παραπάνω συναρτήσεις πραγματοποιούμε σε κάθε διάσταση *2 binary searches*, για το *min* και *max*. Όσο πηγαίνουμε αριστερά, για την ελάχιστη τιμή του διαστήματος, φροντίζουμε να αποθηκεύουμε στην λίστα και τους κόμβους των δεξιών υποδένδρων και αντίστοιχα, όσο πηγαίνουμε δεξιά για την μέγιστη τιμή του διαστήματος.

***update***

Για την πράξη update, δίνουμε σαν είσοδο τη ρίζα του δέντρου και τις νέες συντεταγμένες του κόμβου και παίρνουμε στην έξοδο την ρίζα του νέου δέντρου. Ελέγχουμε, αρχικά, αν υπάρχει ο κόμβος που θέλουμε να αλλάξουμε τις συντεταγμένες του. Φτιάχνουμε ένα νεο Node object. Χρησιμοποιούμε την *delete* για να διαγράψουμε τον προηγούμενο και με την *insert* τον εισάγουμε στο δέντρο. Επιστρέφουμε και πάλι την ρίζα του νέου δέντρου.

## Εκτέλεση πειραμάτων – Αρχείο menu.py

Για την εκτέλεση των πειραμάτων χρησιμοποιουμε το αρχείο *menu.py*, στο οποίο έχουμε κάνει import τα *range\_tree.py* και το *hospitals.py*, όπου περιέχεται η κλάση *hospital* και η συνάρτηση *put\_into\_list*, όπως φτιάχτηκαν για την υλοποίηση των πειραμάτων του *K-D tree.*

Για τα πειράματά μας, έχουν φτιαχτεί 4 επιπλέον συναρτήσεις.

***pre\_order***

Για να μπορούμε να τυπώνουμε το δέντρο που δημιουργούμε κάθε φορά, επιλέχθηκε το pre – order traversal, όπου κατευθυνόμαστε στους κόμβους από πάνω προς τα κάτω και από αριστερά προς τα δεξιά (άλλες διατάξεις είναι η in – order και post – order).

***print\_nodes***

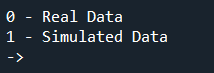
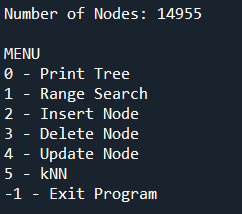
Για να τυπώνουμε τους κόμβους που επιστρέφει η *range\_search.*

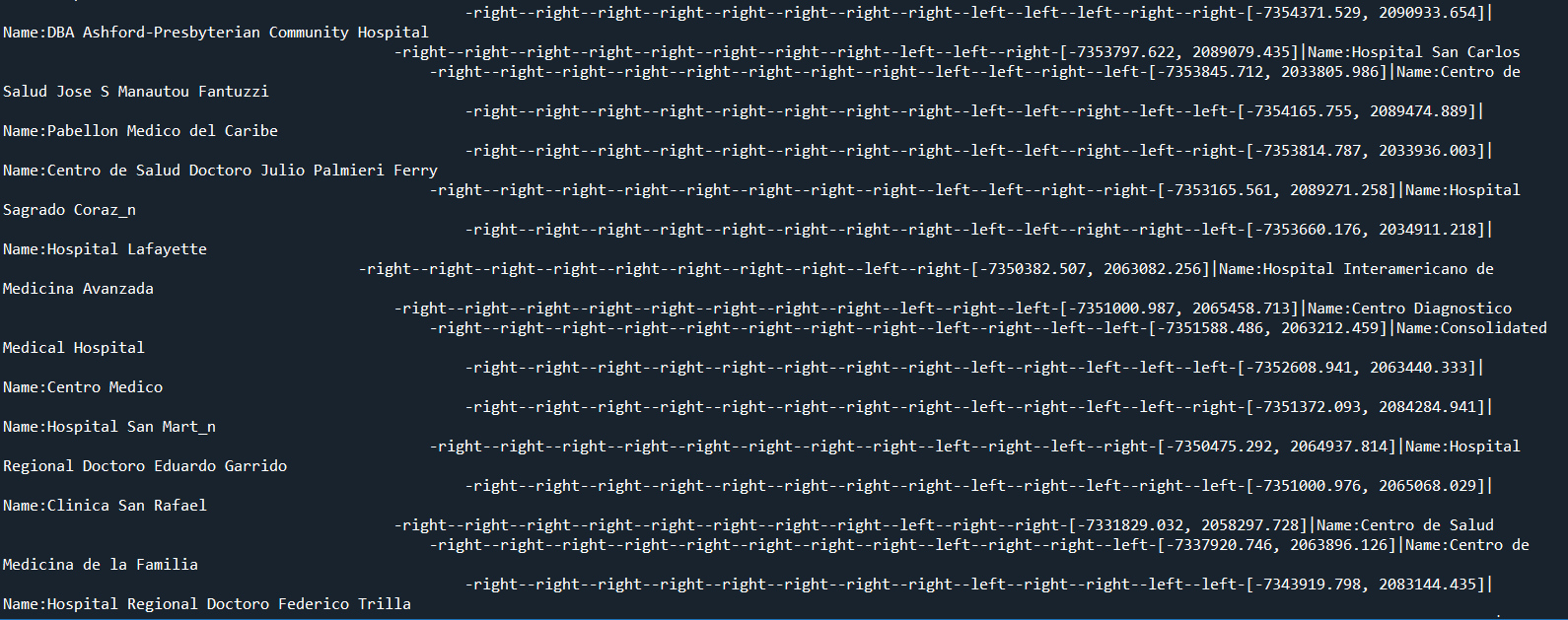
***euclidean\_distance***

Υπολογίζουμε την ευκλείδια απόσταση μεταξύ δύο κόμβων για την υλοποίηση του *kNN*. Συγκεκριμένα, η απόσταση υπολογίζεται από τον τύπο ωστόσο σε αυτή την υλοποίηση δεν υπολογίζουμε την ρίζα.

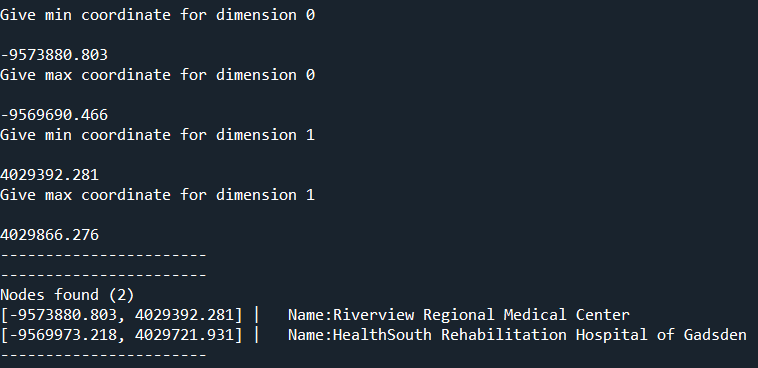
***kNN\_algorithm***

Για την υλοποίηση του *kNN* αλγορίθμου, έχουμε λάβει υπόψιν τις ιδιότητες της δομής. Αρχικά, η συνάρτηση παίρνει ως ορίσματα τους γείτονες που ψάχνουμε (+1 γιατί ο πρώτος κόμβος θα είναι ο εαυτός του), τον κόμβο, ως Node object, του οποίου τους γείτονες ψάχνουμε και την ρίζα του δέντρου. Καθώς οι κόμβοι του δέντρου είναι διατεταγμένοι, γνωρίζουμε ότι γύρω του βρίσκονται κόμβοι με κοντινές συντεταγμένες. Επομένως, αρχικοποιούμε τυχαία ένα *range* και χρησιμοποιούμε την συνάρτηση *range\_search*, που παρουσιάστηκε παραπάνω, για να πάρουμε τους κόμβους που βρίσκονται στην άμεση γειτονιά του. Υπολογίζουμε την απόσταση, σύμφωνα με την *euclidean\_distance* και σορτάρουμε τους κόμβους. Σε περίπτωση που είναι λιγότεροι από *k*, επαναλαμβάνουμε τη διαδικασία ανοίγοντας κάθε φορά το διάστημα.

Ξεκινώντας το *menu,* ο χρήστης πρέπει να διαλέξει μεταξύ *real ή simulated data*, όπου τα πραγματικά αφορούν τα δεδομένα νοσοκομείων που παρουσιάστηκαν στην εισαγωγή, με μόνη διαφορά ότι σε αυτή την υλοποίηση διαβάζονται από το csv. Αντίθετα, τα *simulated data* αποτελούν τυχαίες συντεταγμένες. Στη συνέχεια, δημιουργείται το δέντρο και εμφανίζεται στον χρήστη ένα μενού, με τις 5 επιλογές που βλέπουμε στην εικόνα. Ο χρήστης μπορεί να επιλέγει διαφορετικά έως ότου αποφασίσει να βγει από το πρόγραμμα. Στην περίπτωση των πραγματικών δεδομένων, όπως εδώ, εμφανίζεται επιπλέον και το σύνολο των κόμβων που υλοποιήθηκαν. Παρακάτω φαίνονται τα αποτελέσματα για την επιλογή 0 – Print Tree, όπου μαζί με τα ονόματα και τις συντεταγμένες των κόμβων, μπορούμε να δούμε και την θέση τους στο δέντρο.



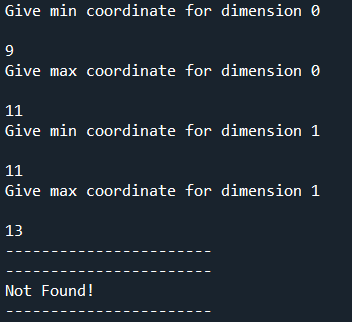
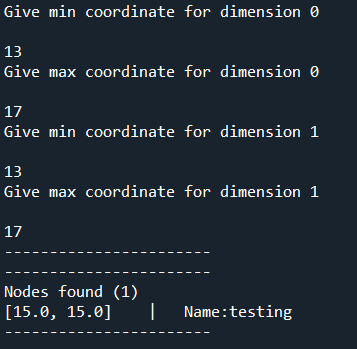
Στη συνέχεια, θα εκτελέσουμε την επιλογή 1 – Range Search χρησιμοποιώντας το ίδιο διάστημα με το πείραμα που έγινε στο *KD* δέντρο, για σύγκριση των αποτελεσμάτων και παρατηρούμε ότι είναι σωστά.



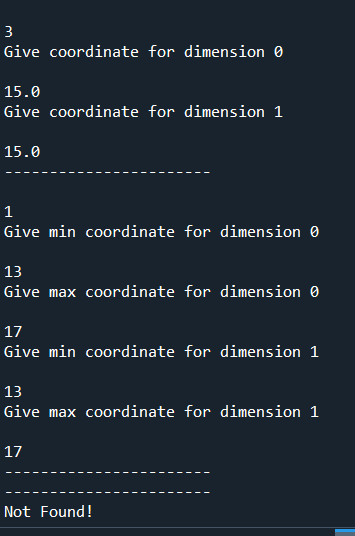
Για την επιλογή 2 – Insert θα προσθέσουμε ένα κόμβο με συντεταγμένες [10.0, 12.0] και ύστερα θα πραγματοποιήσουμε range search στις περιοχές [9, 11] και [9, 13], αντίστοιχα για κάθε διάσταση.



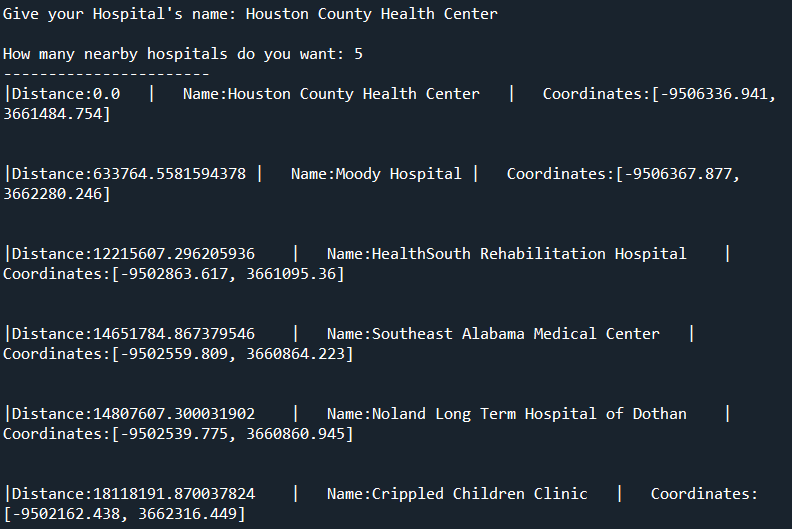
Θα κάνουμε, πρώτα, update στις συντεταγμένες αυτού του κόμβου, δίνοντάς του τις νέες [15.0, 15.0]. Κάνοντας αναζήτηση στο διάστημα [9, 11] και [9, 13] για κάθε διάσταση, βλέπουμε ότι ο κόμβος [10.0, 12.0] δεν υπάρχει. Για να τον εντοπίσουμε, όμως, θα πραγματοποιήσουμε range search στο διάστημα [13, 17] και [13, 17], όπου βλέπουμε ότι υπάρχει.

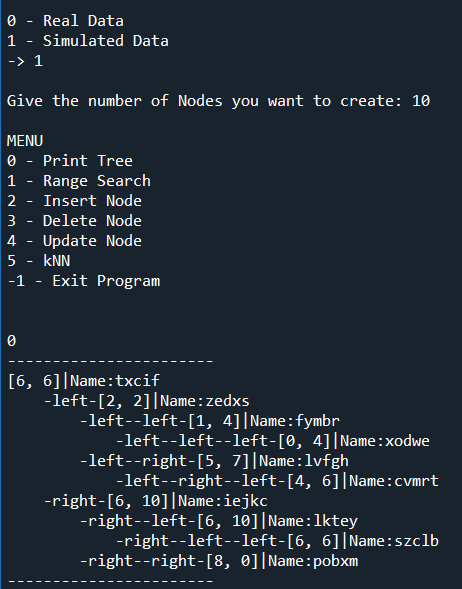
Για να δούμε το αποτέλεσμα της *delete*, θα διαγράψουμε τον κόμβο που προσθέσαμε και ανανεώσαμε και θα κάνουμε *range search* στο διάστημα [13, 17] για κάθε διάσταση. Όπως φαίνεται στην παρακάτω εικόνα, ο κόμβος δεν υπάρχει πια.

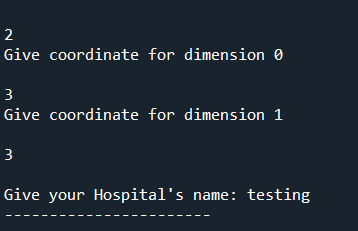


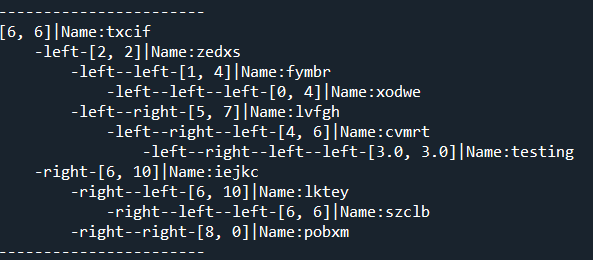
Ολοκληρώνοντας με τα πραγματικά δεδομένα, θα υλοποιήσουμε τον *kNN.* Για επαλήθευση των αποτελεσμάτων, θα δώσουμε τα ίδια ορίσματα με το πείραμα που έγινε στο *KD* δέντρο. Βλέπουμε ότι τα αποτελέσματα είναι και πάλι σωστά.



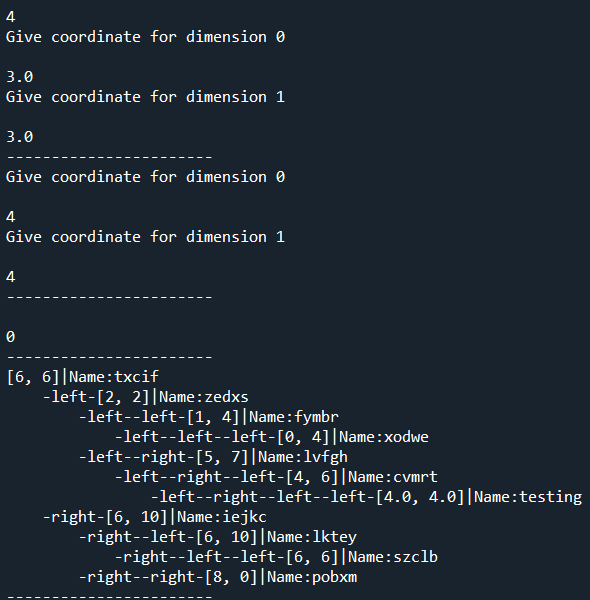
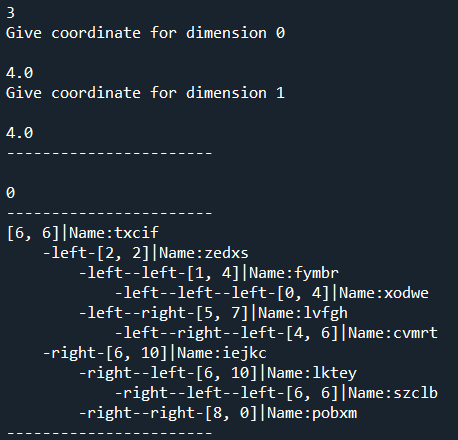
Θα συνεχίσουμε και για την περίπτωση των *simulated* δεδομένων. Κατασκευάζουμε ένα δέντρο με 10 κόμβους και έπειτα το εκτυπώνουμε. Όπως και προηγουμένως, για να δούμε τις πράξεις θα προσθέσουμε ένα νέο κόμβο, θα τον ανανεώσουμε και ύστερα, θα τον διαγράψουμε.







Στην συνέχεια, θα αλλάξουμε τις συντεταγμένες του κόμβου *testing* με νέες, [4, 4] και θα τυπώσουμε και πάλι το δέντρο για να δούμε το αποτέλεσμα.

# *R – Tree*

## *Διαχείριση πραγματικών δεδομένων – αρχείο R-tree.py*

Εφόσον μιλάμε για πραγματικά δεδομένα, τα δεδομένα μας σε αυτή την περίπτωση αφού μας ενδιαφέρει το latitude και το longitude του κάθε νοσοκομείου θα είναι των 2 διαστάσεων. Για τον λόγο αυτό δημιουργούμε μια διασυνδεδεμένη λίστα (List) και την ταξινομούμε κατά αύξουσα σειρά προς το latitude. Για την διαχείριση των πραγματικών αυτών δεδομένων που αφορούν τα νοσοκομεία, δημιουργήθηκαν 9 μέθοδοι στο αρχείο:

Η μέθοδος **length(data,j,D):**

* Η μέθοδος αυτή παίρνει σαν είσοδο την λίστα με τις συντεταγμένες τα minnodes που μας δείχνει στα πόσα τμήματα θα μοιραστή η λίστα μας και το πλήθος τον κόμβων που περιέχεται στη λίστα.

Η μέθοδος **min\_may(List,data):**

* Η μέθοδος αυτή παίρνει σαν είσοδο την λίστα με τις συντεταγμένες και τη λίστα με τα τμήματα που θα μοιραστή η λίστα. Αυτή η μέθοδος υπολογίζει τα νεα bounding box που θα προκύψουν από τα νέα minx , miny , xmax , ymax και τα προσθέτει στο δέντρο.

Η μέθοδος **make\_tree(self,data)**

* Η μέθοδος αυτή καλείται για να δημιουργηθεί το δέντρο. Η μέθοδος αυτή ελέγχει σε πιο υπο- bounding box ανήκει το νέο bounding box

Η μέθοδος **clear\_temp(self,k):**

* Η μέθοδος αυτή καλείται μετά από κάθε εισαγωγή στο δέντρο για να καθαρίσει το πίνακα temp από τα αχρείαστα πλέον δεδομένα.

Η μέθοδος **dist(self,node):**

* Η μέθοδος αυτή καλείται όταν χρειάζεται να υπολογιστεί η απόσταση από ένα σημείο.

Η μέθοδος **Add\_ node(self,node):**

* Η μέθοδος αυτή καλείται όταν ο χρήστης επιλέξει να προσθέσει ένα σημείο. Δέχεται μια παράμετρο, το σημείο που θέλει να προσθέσει. Αρχικά, ελέγχεται πιο bounding box είναι πιο κοντά στο σημείο και το επεκτείνει.

Η μέθοδος **find\_node(sel,node):**

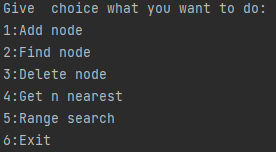
* Η μέθοδος αυτή καλείται όταν ο χρήστης επιλέξει να βρεί αν υπάρχει κάποιο σημείο στο δέντρο. Στη περίπτωση που υπάρχει επιστρέφει Node found.

Η μέθοδος **delete\_node(self, node)** :

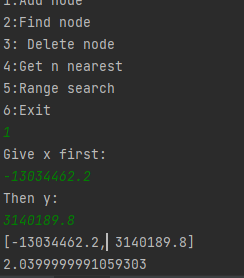
* Η μέθοδος αυτή καλείται όταν ο χρήστης επιλέξει να διαγράψει ένα σημείο από το δέντρο. Δέχεται μια παράμετρο, το σημείο που θέλει να διαγραφεί. Αρχικά, ελέγχεται αν το σημείο που επιθυμεί ο χρήστης να διαγράψει ανήκει στο R– δέντρο, αν ανήκει τότε αφαιρεί το σημείο αυτό και ξαναδημιουργείται Bounding box του R – δέντρου. Αν το σημείο δεν ανήκει στο R – δέντρο, τότε εμφανίζεται μήνυμα λάθους στον χρήστη.

Η μέθοδος **range\_Search(self,node):**

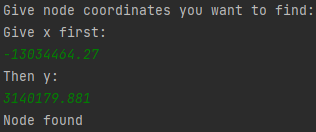
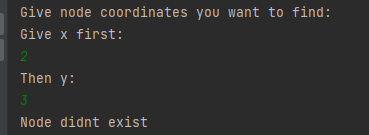
* Η μέθοδος αυτή καλείται όταν ο χρήστης επιλέξει να δει πόσα νοσοκομεία υπάρχουν σε εκείνο το Bounding box.
* Αυτό εδώ είναι το Main Menu με τις επιλογές του χρήστη.



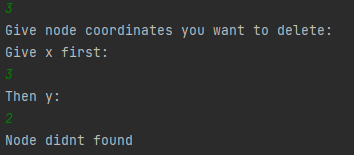
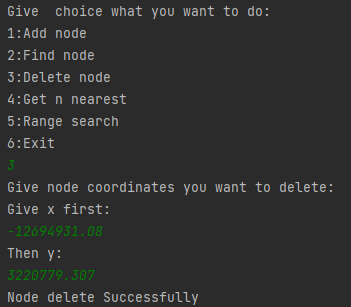
* Όταν επιλέξει το Add node ζητείτε από το χρήστη να βάλει το X και το Y. Αυτόματα υπολογίζεται πιο Bounding box είναι πιο κοντά και το επεκτείνει.



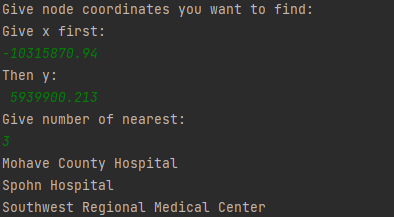
* Όταν επιλέγει το Find\_node ζητείτε από το χρήστη να βάλει το X και το Y. Αμέσως ψάχνει αν υπάρχει αυτό το σημείο. Σε περίπτωση που υπάρχει τυπώνει Node found αλλιώς τυπώνει Node didn’t’t exist.



* Όταν επιλέγει το Delete node ζητείτε από το χρήστη να βάλει το X και το Y. Αυτόματα



* Όταν επιλέγει το Get n nearest ζητείτε από το χρήστη να βάλει το X , Y και τον αριθμό των πλησιέστερων γειτόνων που θέλει να βρουν.



* Όταν επιλέγει το Range search ζητείτε από το χρήστη να βάλει το Xmin, Xmax,Ymin,Ymax για να δημιουργηθεί το Bounding box και να βρει πόσα νοσοκομεία είναι μέσα στο Bounding box.

