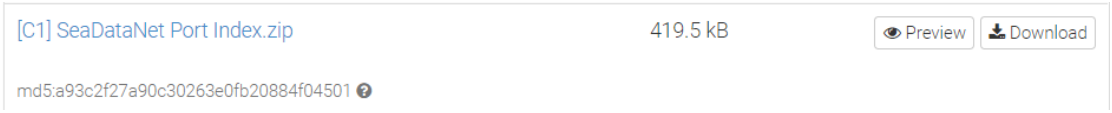
Τεχνική Αναφορά

Τεχνικές Επεξεργασίας Δεδομένων (Data Processing Techniques)

Εισαγωγή

Στο πρόγραμμα επεξεργάζονται γεωγραφικά δεδομένα με τη μορφή points και υλοποιούνται οι αλγόριθμοι Nearest Neighbor και Range Search με τη χρήση κύκλου, χρησιμοποιώντας μια δομή k-d tree.

Για την εργασία χρησιμοποιήθηκε το αρχείο *SeaDataNet Port Index.zip* από την ιστοσελίδα https://zenodo.org/record/1167595#.Y66S53bP2Uk. Το αρχείο περιλαμβάνει ονόματα και συντεταγμένες από αλιευτικούς λιμένες παγκοσμίως.



Το αρχείο *Fishing Ports.dbf* έχει μετατραπεί σε *.csv* για λόγους διευκόλυνσης.

- Από το excel
- Save as .csv

Για να τρέξει το πρόγραμμα, πρέπει το αρχείο *Fishing Ports.csv* να βρίσκεται στο ίδιο directory με το script. Διαφορετικά, μπορεί να χρησιμοποιηθεί path στη γραμμή 181:

Βιβλιοθήκες που χρησιμοποιήθηκαν:

csv: Για να ανοιχτεί το αρχείο Fishing Ports.csv.

folium: Για τη δημιουργία του χάρτη.

webbrowser: Για να ανοιχτεί το αρχείο του χάρτη στον browser.

geodesic: Υπολογίζει την απόσταση δύο σημείων με συντεταγμένες του GWS84 Standard και τη μετατρέπει σε χιλιόμετρα.

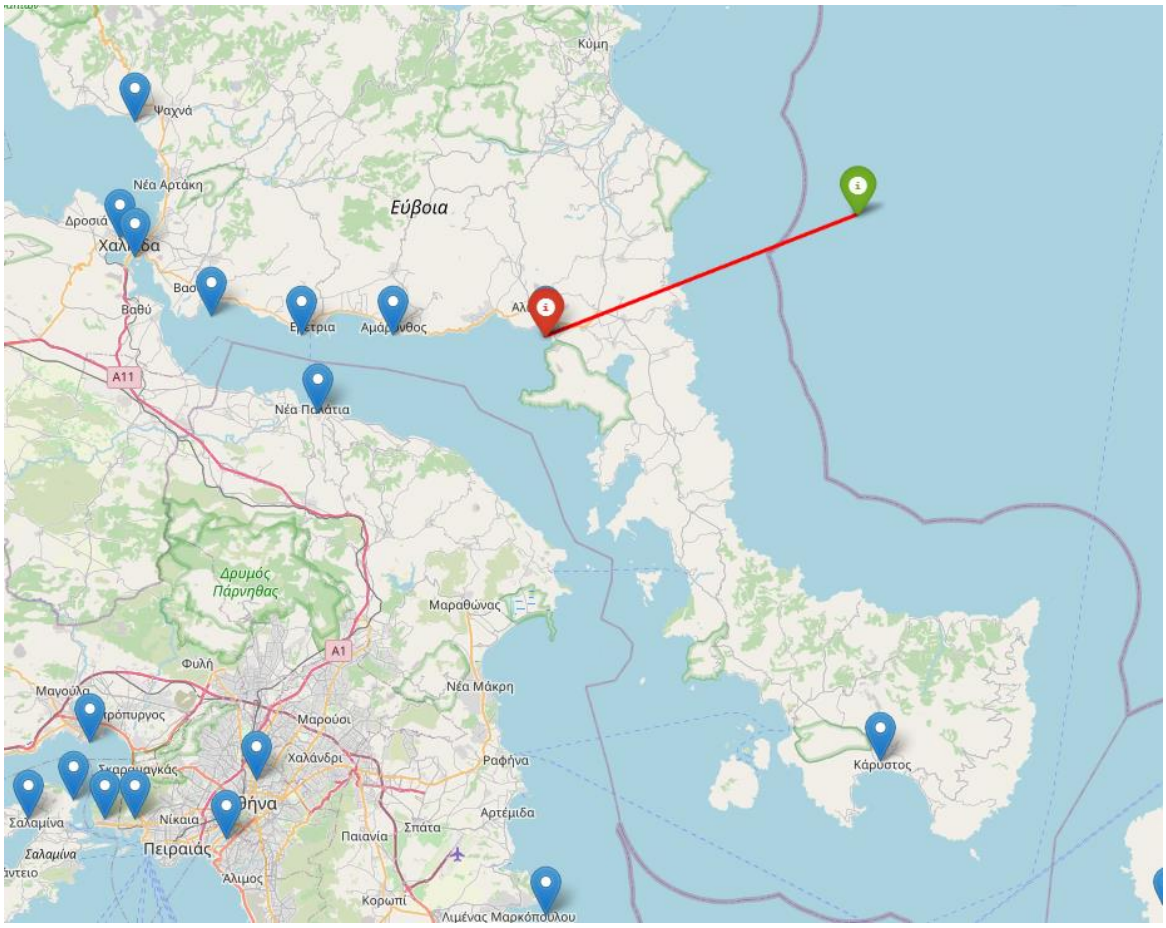
Οδηγός Χρήσης

Πληκτρολογήστε «NN» για τον αλγόριθμο “Nearest Neighbor” ή «RS» για τον αλγόριθμο “Range Search” ή «exit» για να σταματήσει το πρόγραμμα.

Nearest Neighbor

Πληκτρολογήστε το γεωγραφικό πλάτος του σημείου που θέλετε να ελέγξετε και έπειτα το γεωγραφικό μήκος. Το γεωγραφικό πλάτος και μήκος πρέπει να είναι της μορφής: **38.45327, 24.60513**. Μπορούν να χρησιμοποιηθούν συντεταγμένες από το Google Maps.

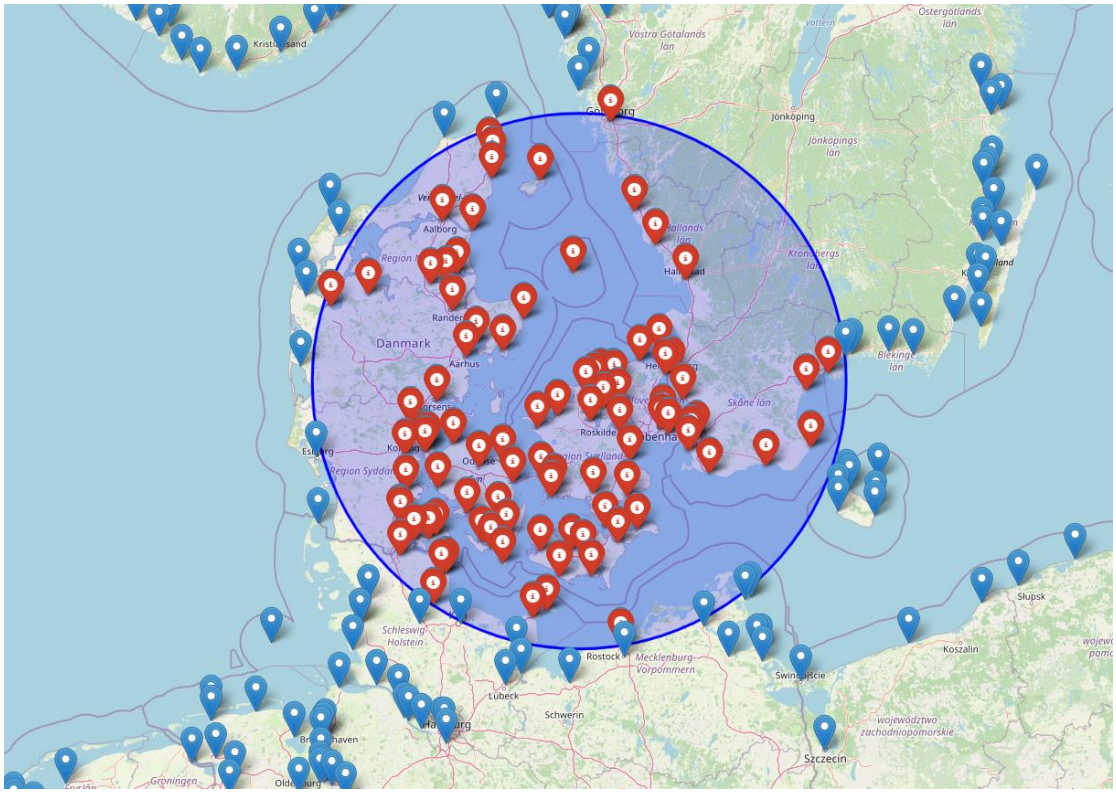
Το πρόγραμμα θα εκτυπώσει τον κοντινότερο λιμένα από το σημείο που εισάχθηκε. Επίσης, θα ανοίξει χάρτη στον browser και θα απεικονίζει το σημείο που εισάχθηκε με πράσινο χρώμα και τον κοντινότερο λιμένα με κόκκινο χρώμα, καθώς και μια γραμμή μεταξύ τους. Τα υπόλοιπα λιμάνια απεικονίζονται με μπλε marker.



Range Search

Πληκτρολογήστε το γεωγραφικό πλάτος του σημείου που θέλετε να ελέγξετε και μετά το γεωγραφικό μήκος με τη μορφή: **55.94625, 11.592099**. Έπειτα, την ακτίνα της απόστασης που θέλετε να ελέγξετε σε χιλιόμετρα.

Το πρόγραμμα θα τυπώσει όλους τους λιμένες που βρίσκονται μέσα στην ακτίνα του σημείου που πληκτρολογήσατε. Αν δεν υπάρχει λιμένας στην ακτίνα, δε θα τυπώσει τίποτα. Επιπλέον, θα ανοίξει χάρτη ο οποίος απεικονίζει τους λιμένες αυτούς με κόκκινο, μέσα σε έναν κύκλο με κέντρο και ακτίνα από τα δεδομένα που δόθηκαν. Οι υπόλοιποι λιμένες απεικονίζονται με μπλε.



Περιγραφή Αλγορίθμων

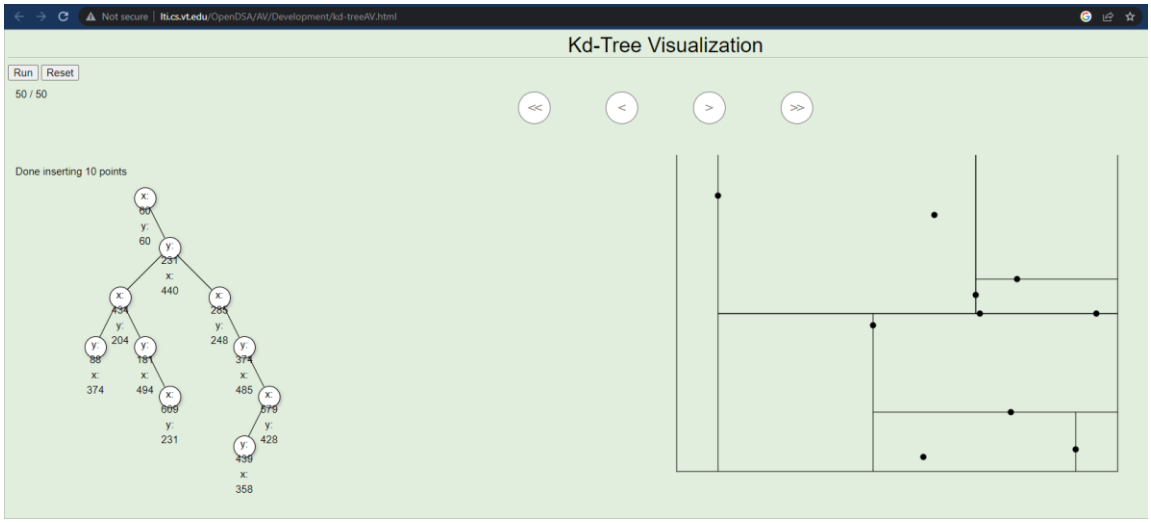
Κλάση Node

Υλοποιεί κόμβο με παραμέτρους γεωγραφικό πλάτος και μήκος, χώρα, όνομα λιμανιού/πόλη, και αριστερό και δεξί παιδί

Κλάση KDTree

Λειτουργία insert_elem

Εισάγει κόμβους στο k-d tree. Χρησιμοποιεί τη μεταβλητή *level* για να ελέγχει σε ποιο ύψος του δέντρου βρίσκεται. Αν το ύψος είναι μονός αριθμός, τότε συγκρίνει το γεωγραφικό πλάτος των κόμβων, διαφορετικά, συγκρίνει το γεωγραφικό μήκος και κινείται αριστερά ή δεξιά στο δέντρο αν ο κόμβος που εισάγεται είναι μικρότερος ή μεγαλύτερος αντίστοιχα από τον κόμβο που βρίσκεται στο συγκεκριμένο επίπεδο. Συνεχίζει το μονοπάτι μέχρι να γίνει η εισαγωγή.



Λειτουργία nearest_neighbor

Δέχεται τέσσερις παραμέτρους:

Target: Το σημείο το οποίο δίνει ο χρήστης για την εύρεση του κοντινότερου λιμένα

iNode: Η πρώτη κλήση της λειτουργίας δέχεται τη ρίζα του δέντρου στη θέση της παραμέτρου. Οι επόμενες, αναδρομικές κλήσεις, δέχονται ένα από τα παιδιά της ρίζας και ούτω καθεξής.

best: Η απόσταση του κοντινότερου λιμένα μέχρι στιγμής. Η πρώτη κλήση της λειτουργίας θα επιστρέψει, εν τέλει, τον κοντινότερο λιμένα.

***depth*:** Το επίπεδο του δέντρου στην κάθε αναδρομή.

Η μεταβλητή *candidate* παίρνει ως τιμή την απόσταση μεταξύ του σημείου που έδωσε ο χρήστης και του κόμβου που κάθε φορά εισέρχεται ως παράμετρος.

Στη συνέχεια, ελέγχεται αν η μεταβλητή candidate είναι μικρότερη από την απόσταση της παραμέτρου best, και αν είναι, η τιμή της μεταβλητής best παίρνει την τιμή της candidate.

```
68          candidate = GD(tmp_target, tmp_iNode).km
69
70          if best is None:    # first
71              best = iNode
72          else:
73              tmp_best = (best.lat_x, best.lon_y)
74              if candidate < GD(tmp_target, tmp_best).km:
75                  best = iNode
```

Έπειτα, ελέγχεται σε ποιο επίπεδο του δέντρου βρίσκεται εκείνη τη στιγμή ο αλγόριθμος χρησιμοποιώντας τη μεταβλητή depth. Αν το depth είναι ζυγός ή μονός

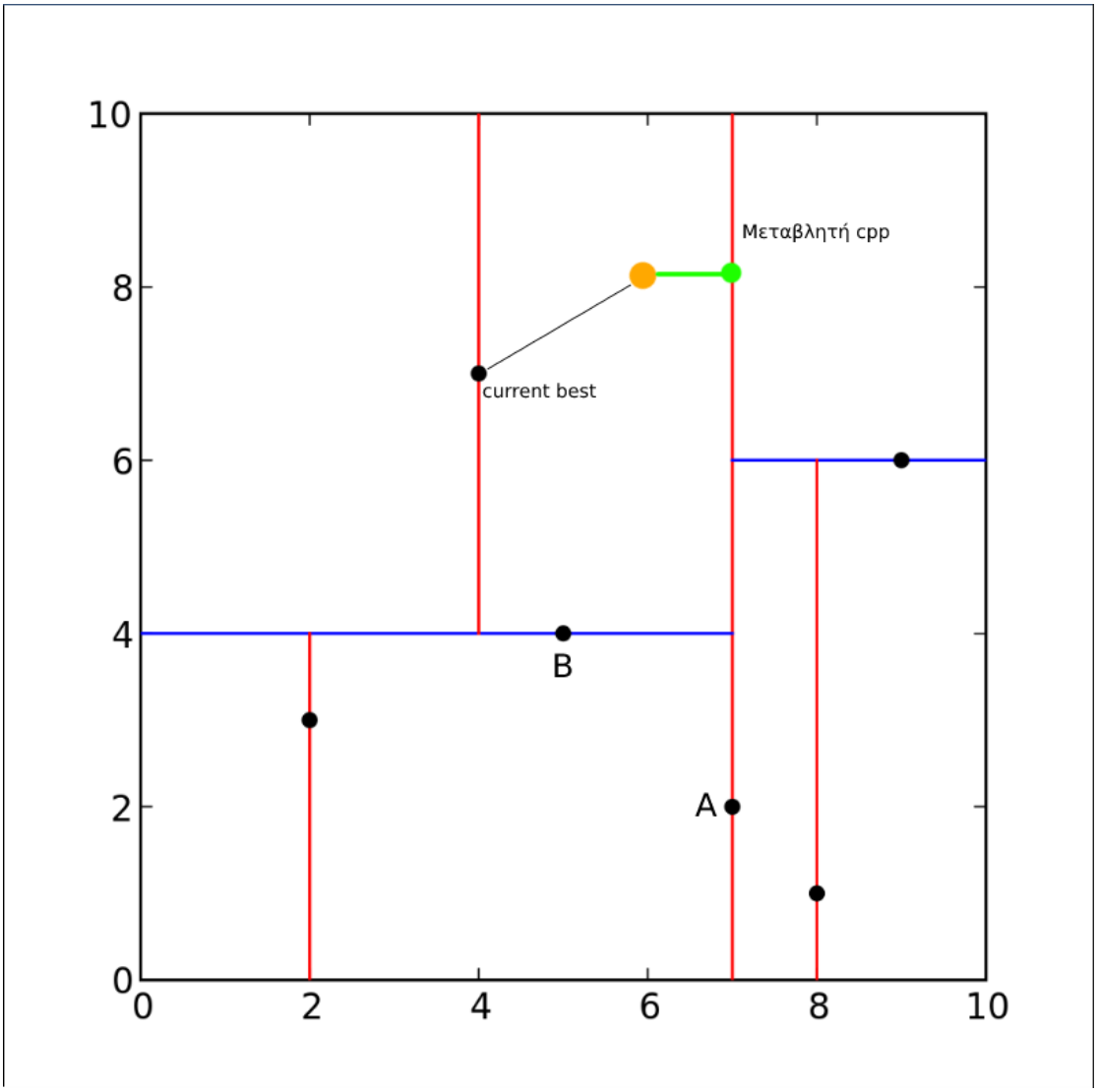
αριθμός, συγκρίνει το γεωγραφικό μήκος ή πλάτος αντίστοιχα και καλείται ξανά η λειτουργία nearest_neighbor με παράμετρο τον επόμενο κόμβο, είτε το αριστερό είτε το δεξί παιδί μαζί με την μεταβλητή best. Αν γίνει η κλήση της λειτουργίας στον δεξί κόμβο, τότε μια μεταβλητή *temp_node* αποθηκεύει τον αριστερό κόμβο, και το αντίθετο αν η κλήση γίνει στον αριστερό.

```
77          if depth % 2 == 0:  # compare y
78              if target.lon_y <= iNode.lon_y:
79                  if iNode.right_child is not None:
80                      temp_node = iNode.right_child
81                  if iNode.left_child is not None:
82                      best = self.nearest_neighbor(target, iNode.left_child, best, depth + 1)
83              else:
84                  if iNode.left_child is not None:
85                      temp_node = iNode.left_child
86                  if iNode.right_child is not None:
87                      best = self.nearest_neighbor(target, iNode.right_child, best, depth + 1)
88          else:   # compare x
89              if target.lat_x <= iNode.lat_x:
90                  if iNode.right_child is not None:
91                      temp_node = iNode.right_child
92                  if iNode.left_child is not None:
93                      best = self.nearest_neighbor(target, iNode.left_child, best, depth + 1)
94              else:
95                  if iNode.left_child is not None:
96                      temp_node = iNode.left_child
97                  if iNode.right_child is not None:
98                      best = self.nearest_neighbor(target, iNode.right_child, best, depth + 1)
```

Στη συνέχεια, ο αλγόριθμος ελέγχει αν υπάρχει κοντινότερος κόμβος στους κόμβους που δεν έλεγξε. Αν για παράδειγμα στο επόμενο σχήμα βρισκόμαστε στον κόμβο A και το σημείο που δόθηκε από το χρήστη βρίσκεται στο (6, 8) τότε μέχρι τώρα έχουν ελεγχθεί οι κόμβοι που είναι μικρότεροι από το X (γεωγραφικό πλάτος) του A, αν το A βρίσκεται σε μονό επίπεδο. Αν η απόσταση μεταξύ του point που έδωσε ο χρήστης και του κοντινότερου σημείου (μεταβλητή ccp (closest point possible)) ως την δεξιά μεριά του A είναι μικρότερη από τον κοντινότερο κόμβο που έχει βρεθεί μέχρι τώρα (μεταβλητή best), τότε υπάρχει πιθανότητα να υπάρχει κόμβος ο οποίος να βρίσκεται πιο κοντά.

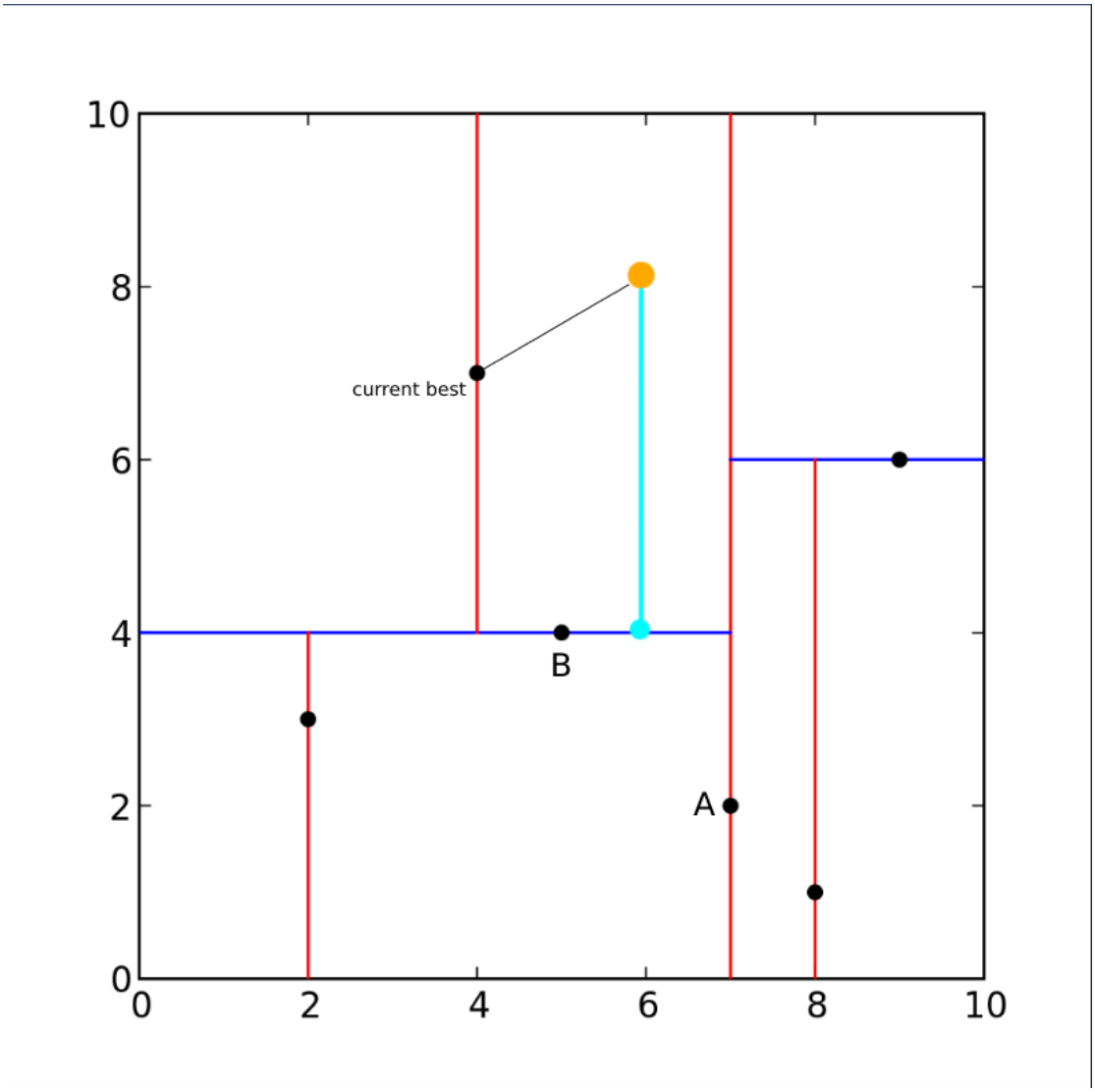
Το κοντινότερο σημείο cpp, έχει συντεταγμένες το Y του point και το X του κόμβου που ελέγχουμε (του A στο παράδειγμα) αν ο κόμβος βρίσκεται σε μονό αριθμό επιπέδου του δέντρου.

Αφού ελεγχθεί η απόσταση, καλείται ή όχι η λειτουργία nearest_neighbor ξανά για το κλαδί του δέντρου που δεν ελέγχθηκε.




```
111         else:   # compare x
112             cpp = Node(iNode.lat_x, target.lon_y)   # closest point possible
113             if temp_node is not None:
114 
115                 tmp_target = (target.lat_x, target.lon_y)
116                 tmp_cpp = (cpp.lat_x, cpp.lon_y)
117                 tmp_best = (best.lat_x, best.lon_y)
118 
119                 if GD(tmp_target, tmp_cpp).km < GD(tmp_target, tmp_best).km:  # if closest point possible < current best
120                     best = self.nearest_neighbor(target, temp_node, best, depth + 1)
```

Στο κάτω σχήμα το cpp μεταξύ του B και του point είναι κάθετο διότι ο κόμβος B βρίσκεται σε ζυγό αριθμό επιπέδου. Οι συντεταγμένες του cpp είναι το X του point και το Y του B. Σύμφωνα με το συγκεκριμένο σχήμα, η λειτουργία nearest_neighbor δε θα καλεστεί διότι το cpp βρίσκεται πιο μακριά από το κοντινότερο που έχουμε ήδη βρει. Άρα δεν υπάρχει καμία περίπτωση να βρεθεί κόμβος με μικρότερη απόσταση.




```
101         if depth % 2 == 0:  # compare y
102             cpp = Node(target.lat_x, iNode.lon_y)   # closest point possible
103             if temp_node is not None:
104
105                 tmp_target = (target.lat_x, target.lon_y)
106                 tmp_cpp = (cpp.lat_x, cpp.lon_y)
107                 tmp_best = (best.lat_x, best.lon_y)
108
109                 if GD(tmp_target, tmp_cpp).km < GD(tmp_target, tmp_best).km:  # if closest point possible < current best
110                     best = self.nearest_neighbor(target, temp_node, best, depth + 1)
```

Λειτουργία circular_range_search

Δέχεται τέσσερις παραμέτρους:

point: Το σημείο που δίνει ο χρήστης ως κέντρο του κύκλου

radius: Το μήκος της ακτίνας που δίνει ο χρήστης σε χιλιόμετρα

iNode: Ο κόμβος που εξετάζεται. Στην πρώτη κλήση της λειτουργίας δίνεται ως παράμετρος η ρίζα του δέντρου και στις επόμενες αναδρομικές κλήσεις, τα επόμενα παιδιά της ρίζας.

depth: Το επίπεδο του δέντρου σε κάθε κλήση της λειτουργίας.

Στην αρχή ελέγχεται αν ο κόμβος βρίσκεται μέσα στην ακτίνα. Αν ναι, τότε τυπώνεται το όνομα του λιμανιού και οι συντεταγμένες του και δημιουργείται marker στον χάρτη με κόκκινο χρώμα.

```
131         if GD(tmp_point, tmp_iNode).km <= radius:
132             print(iNode.city, "is in the radius. ", iNode.lat_x, iNode.lon_y)
133             folium.Marker(location=[iNode.lat_x, iNode.lon_y], popup=iNode.city, icon=folium.Icon(color='red')).add_to(
134                 my_map)
```

Το υπόλοιπο του αλγορίθμου λειτουργεί ακριβώς όπως της nearest_neighbor. Ελέγχεται το επίπεδο του κάθε κόμβου και το δέντρο συνεχίζει τις αναζητήσεις και τους ελέγχους των κόμβων.

```
136         if depth % 2 == 0:
137             if point.lon_y <= iNode.lon_y:
138                 if iNode.left_child is not None:
139                     self.circular_range_search(point, radius, iNode.left_child, depth + 1)
140                 if iNode.right_child is not None:
141                     temp_node = iNode.right_child
142 
143             else:
144                 if iNode.right_child is not None:
145                     self.circular_range_search(point, radius, iNode.right_child, depth + 1)
146                 if iNode.left_child is not None:
147                     temp_node = iNode.left_child
148         else:
149             if point.lat_x <= iNode.lat_x:
150                 if iNode.left_child is not None:
151                     self.circular_range_search(point, radius, iNode.left_child, depth + 1)
152                 if iNode.right_child is not None:
153                     temp_node = iNode.right_child
154             else:
155                 if iNode.right_child is not None:
156                     self.circular_range_search(point, radius, iNode.right_child, depth + 1)
157                 if iNode.left_child is not None:
158                     temp_node = iNode.left_child
```

```
161         if depth % 2 == 0:
162             cpp2 = Node(point.lat_x, iNode.lon_y)   # closest point possible
163             if temp_node is not None:
164 
165                 tmp_point = (point.lat_x, point.lon_y)
166                 tmp_cpp2 = (cpp2.lat_x, cpp2.lon_y)
167 
168                 if GD(tmp_point, tmp_cpp2).km <= radius:
169                     self.circular_range_search(point, radius, temp_node, depth + 1)
170         else:
171             cpp2 = Node(iNode.lat_x, point.lon_y)   # closest point possible
172             if temp_node is not None:
173 
174                 tmp_point = (point.lat_x, point.lon_y)
175                 tmp_cpp2 = (cpp2.lat_x, cpp2.lon_y)
176 
177                 if GD(tmp_point, tmp_cpp2).km <= radius:
178                     self.circular_range_search(point, radius, temp_node, depth + 1)
```

Ανάγνωση του αρχείου Fishing Ports.csv

Δημιουργείται αντικείμενο της κλάσης KDTree και έπειτα χάρτης με το όνομα my_map. Στη συνέχεια, γίνεται ανάγνωση των στηλών 5, 6, 4, 3 του αρχείου και αποθηκεύεται η κάθε γραμμή σε έναν κόμβο Node. Στις στήλες που περιέχουν συντεταγμένες, αντικαθίσταται το κόμμα (,) με τελεία (.) για να μετατραπούν οι τιμές από String σε Float. Τέλος, δημιουργείται marker πάνω στον χάρτη για το κάθε λιμάνι στις ανάλογες συντεταγμένες και εισέρχεται ο κάθε κόμβος στο δέντρο.

```
181    with open(r'Fishing Ports.csv', 'r') as csv_file:
182        csv_reader = csv.reader(csv_file, delimiter=';')
183        next(csv_reader)        # skip first row
184
185        kdt = KDTree()
186
187        my_map = folium.Map(location=[37.98520100051584, 23.751189777734144], zoom_start=7)
188
189        iNode = None
190        for line in csv_reader:
191            iNode = Node(line[5], line[6], line[4], line[3])
192
193            iNode.lat_x = float(iNode.lat_x.replace(",", "."))
194            iNode.lon_y = float(iNode.lon_y.replace(",", "."))
195
196            folium.Marker(location=[iNode.lat_x, iNode.lon_y], popup=iNode.city).add_to(my_map)
197
198            kdt.insert_elem(iNode)
```

Ο υπόλοιπος κώδικας ζητάει από το χρήστη ποιον αλγόριθμο θέλει να χρησιμοποιήσει, ζητώντας στη συνέχεια συντεταγμένες ή και ακτίνα, ανάλογα τον αλγόριθμο που επέλεξε.

Αν επιλέχθηκε ο αλγόριθμος nearest_neighbor, γίνεται κλήση του με παραμέτρους το σημείο του χρήστη (point), τη ρίζα (r) του δέντρου, None αφού δεν υπάρχει best ακόμα, και το επίπεδο της ρίζας (depth) και αποθηκεύει το αποτέλεσμα στον κόμβο aNode ο οποίος χρησιμοποιείται για να εκτυπωθούν τα αποτελέσματα καθώς και αν δημιουργηθεί marker στον χάρτη.

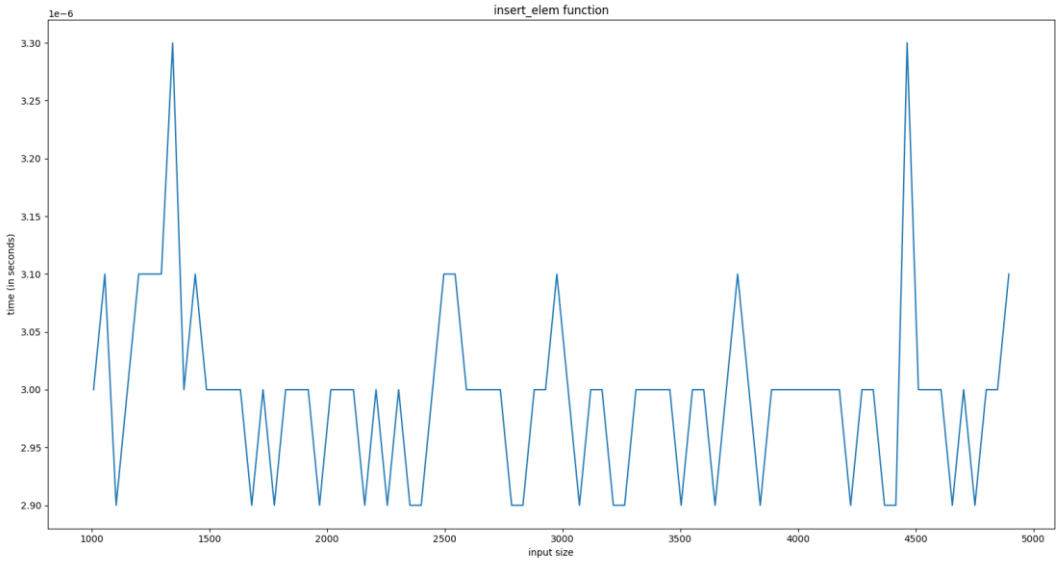
```
    r = kdt.root    # root node
    point = Node

    print("Type 'NN' (Nearest Neighbor) or 'RS' (Range Search) or 'exit':")
    inputx = input()
    while inputx != 'NN' or inputx != 'RS' or inputx != 'exit':
        if inputx == 'NN':
            print("Type point latitude:")
            point.lat_x = input()
            print("Type point longitude:")
            point.lon_y = input()

            point.lat_x = float(point.lat_x)
            point.lon_y = float(point.lon_y)

            aNode = kdt.nearest_neighbor(point, r, None, 1)
            print("Nearest neighbor coordinates: ", aNode.country, aNode.city, aNode.lat_x, aNode.lon_y)

            folium.Marker(location=[aNode.lat_x, aNode.lon_y], popup=aNode.city, icon=folium.Icon(color='red')).add_to(
                my_map)

            folium.Marker(location=[point.lat_x, point.lon_y], popup='Starting location',
                          icon=folium.Icon(color='green')).add_to(my_map)

            folium.PolyLine([(aNode.lat_x, aNode.lon_y), (point.lat_x, point.lon_y)], color='red').add_to(my_map)

            my_map.fit_bounds([[point.lat_x, point.lon_y], [aNode.lat_x, aNode.lon_y]])

            my_map.save('my_map.html')
            webbrowser.open(('my_map.html'))

            break

        elif inputx == 'RS':
            print("Type point latitude:")
            point.lat_x = input()
            print("Type point longitude:")
            point.lon_y = input()
            print("Type radius (in km):")
            radius = input()
```

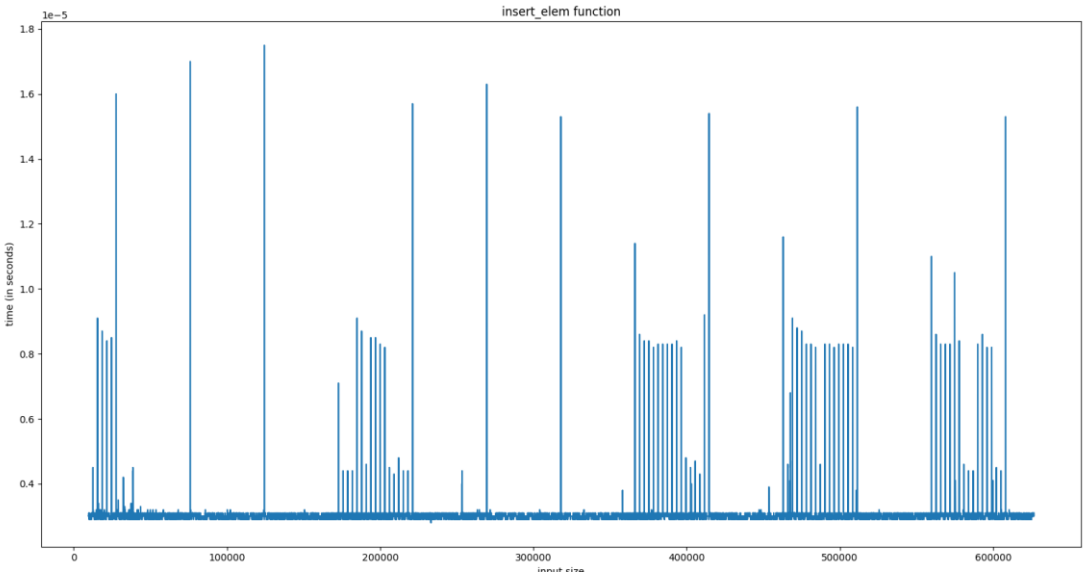
Η κλήση της circular_range_search δέχεται σαν παραμέτρους το σημείο του χρήστη (point), την ακτίνα (radius), τη ρίζα (r) του δέντρου, και το επίπεδο της ρίζας (depth). Στη συνέχεια, δημιουργεί κύκλο πάνω στον χάρτη με κέντρο και ακτίνα το σημείο και την ακτίνα που όρισε ο χρήστης.

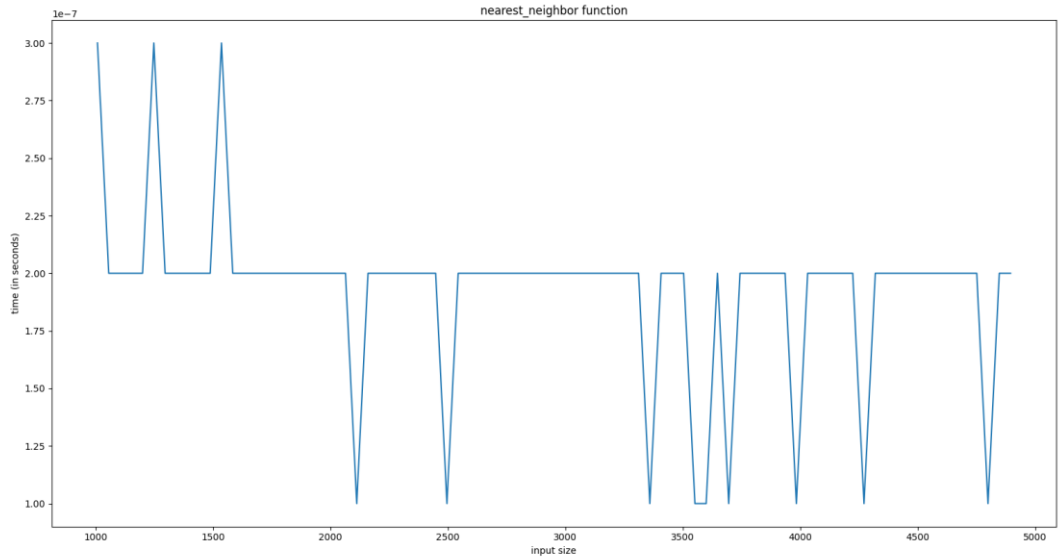
```
            point.lat_x = float(point.lat_x)
            point.lon_y = float(point.lon_y)
            radius = float(radius)

            kdt.circular_range_search(point, radius, r, 1)

            folium.Circle(location=[point.lat_x, point.lon_y], radius=(radius * 1000), color='blue', fill=True).add_to(
                my_map)  # radius in meters

            my_map.location = [point.lat_x, point.lon_y]

            my_map.save('my_map.html')
            webbrowser.open(('my_map.html'))

            break
        elif inputx == 'exit':
            print("Program stopped.")
            break
        else:
            print("Wrong input.")
            print("Type 'NN' (Nearest Neighbor) or 'RS' (Range Search) or 'exit':")
            inputx = input()
```

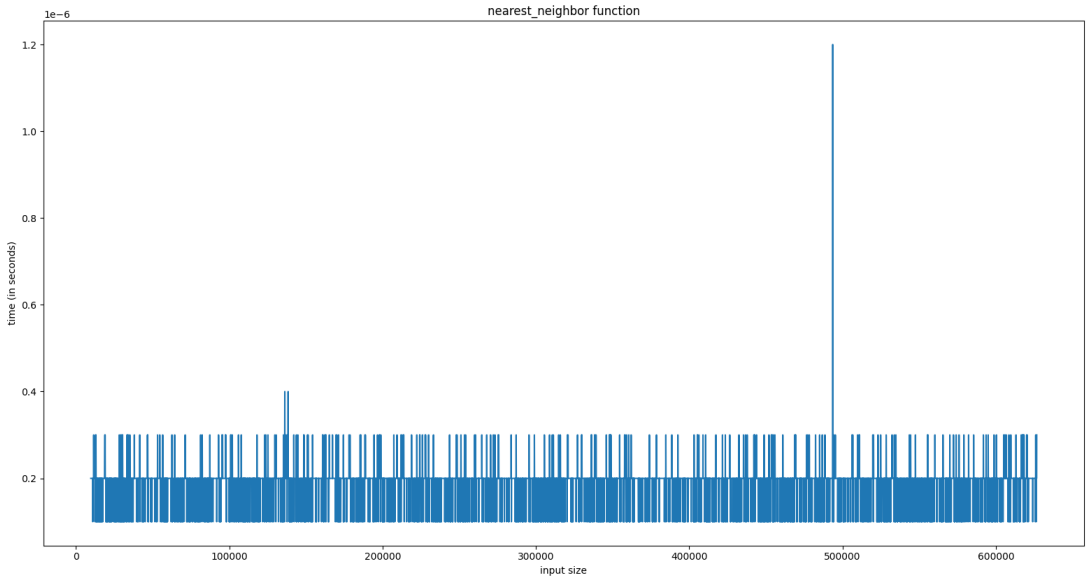
Πειραματική Μελέτη

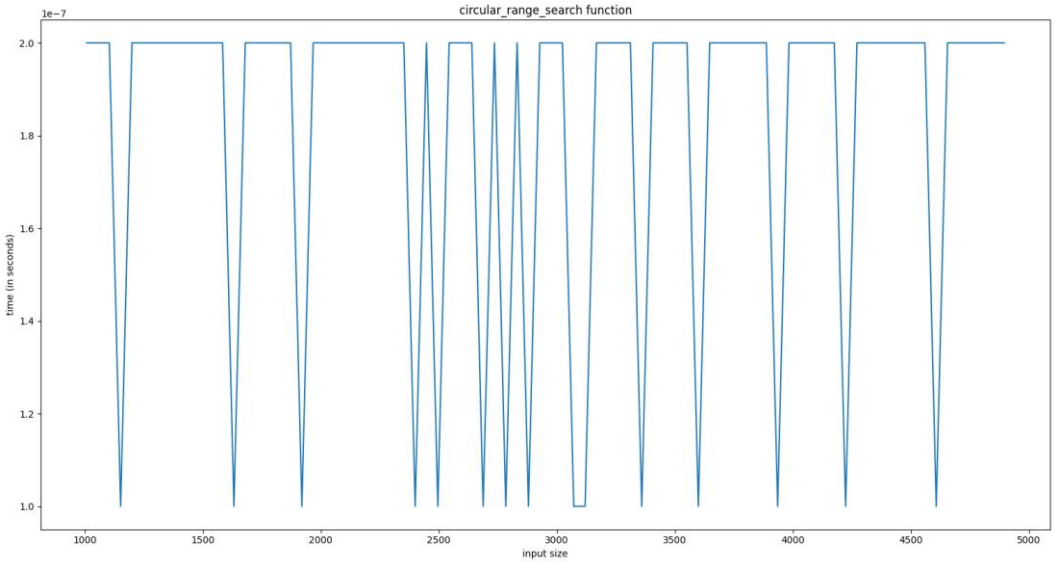
Οι αλγόριθμοι μελετήθηκαν με το ίδιο σύνολο δεδομένων που χρησιμοποιείται στο πρόγραμμα (Fishing Ports.csv), με ≈5.000 στοιχεία και με ≈600.000 στοιχεία για τον καθένα.

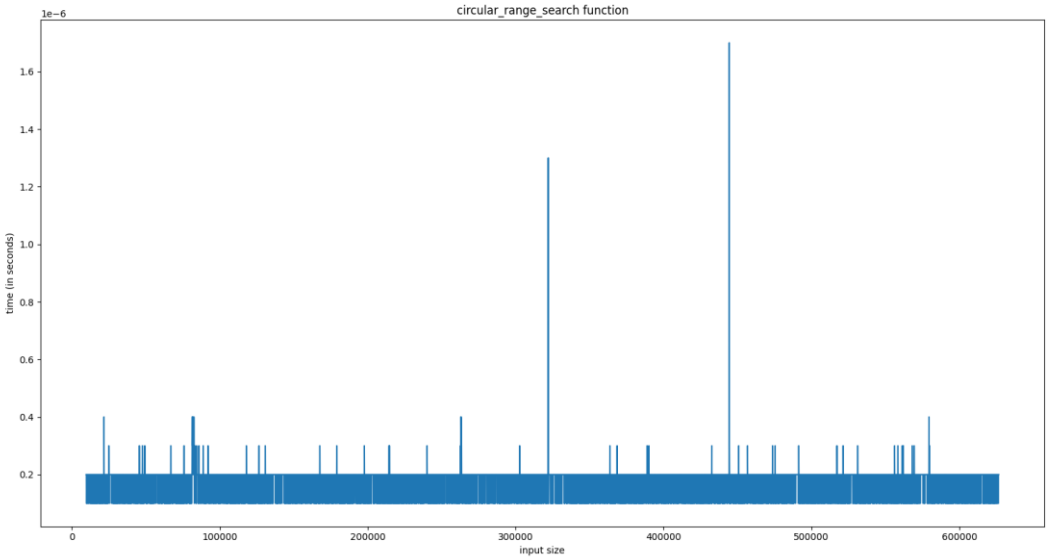





nearest_neighbor function
1e−7
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
time (in seconds)
1000
1500
2000
2500
3000
3500
4000
4500
5000
input size


nearest_neighbor function
1e−6
1.2
1.0
0.8
0.6
0.4
0.2
time (in seconds)
0
100000
200000
300000
400000
500000
600000
input size


circular_range_search function
1e−7
2.0
1.8
1.6
1.4
1.2
1.0
time (in seconds)
1000
1500
2000
2500
3000
3500
4000
4500
5000
input size



Πηγές

Για τους αλγορίθμους nearest neighbor και kd-tree:

https://www.youtube.com/watch?v=Glp7THUpGow

https://www.youtube.com/watch?v=mxrUFkdXaR8

https://www.youtube.com/watch?v=nll58oqEsBg (Αλγόριθμος που χρησιμοποιήθηκε για τη nearest_neighbor)

http://lti.cs.vt.edu/OpenDSA/AV/Development/kd-treeAV.html

https://en.wikipedia.org/wiki/K-d_tree

Για τον αλγόριθμο range search:

https://en.wikipedia.org/wiki/Range_searching

https://iq.opengenus.org/range-searching/