kd-tree

3Δ ΥΠΟΛΟΓΙΣΤΙΚΗ ΓΕΩΜΕΤΡΙΑ & ΟΡΑΣΗ

5η Εργαστηριακή Άσκηση

ΓΕΡΟΝΤΑΣ ΝΙΚΟΛΑΟΣ
Α.Μ.: 1092813

Περιεχόμενα

Ασκήσεις 1 έως και 4:

Δημιουργήστε ένα kd-tree από ένα νέφος σημείων, συμπληρώνοντας τον constructor της κλάσης KdNode, η οποία κατασκευάζει, με αναδρομικό τρόπο, κόμβους ενός kd-tree.

Απεικονίστε με διαφορετικό χρώμα τα σημεία του κάθε υπό-δέντρου του kd-tree δεδομένου του επιπέδου. Για να το κάνετε αυτό θα χρειαστεί να υλοποιήσετε τις παρακάτω συναρτήσεις:

a. ***getMaxDepth:*** Η συνάρτηση αυτή υπολογίζει, με αναδρομικό τρόπο, το μέγιστο βάθος του kd-tree.
b. ***getNodesBelow:*** Η συνάρτηση αυτή, με αναδρομικό τρόπο, εντοπίζει και επιστρέφει όσους κόμβους βρίσκονται σε βαθύτερο επίπεδο από τον κόμβο που λαμβάνει ως είσοδο.
c. ***getNodesAtDepth***: Η συνάρτηση αυτή με αναδρομικό τρόπο, εντοπίζει και επιστρέφει τους κόμβους του kd-tree που βρίσκονται σε κάποιο ορισμένο βάθος.

Ο κώδικας που αναθέτει το χρώμα στα σημεία είναι ήδη υλοποιημένος. Με τη χρήση των ***up*** και ***down arrows*** μπορείτε να επιλέξετε διαφορετικά επίπεδα του kd-tree. Θα παρατηρήσετε επίσης ότι όλα τα pivot σημεία μέχρι και το επιλεγμένο βάθος απεικονίζονται με άσπρο χρώμα.

Συμπληρώστε τη συνάρτηση ***inSphere***, η οποία, με αναδρομικό τρόπο, εντοπίζει και επιστρέφει όλα τα σημεία του kd-tree που βρίσκονται εντός της σφαίρας που λαμβάνει ως είσοδο.

Συμπληρώστε τη συνάρτηση ***nearestNeighbor***, η οποία, με αναδρομικό τρόπο, εντοπίζει και επιστρέφει το σημείο του kd-tree που είναι κοντινότερο στο κέντρο της σφαίρας.

Απάντηση:

✓ Ολοκληρώθηκαν κατά τη διάρκεια του εργαστηρίου!

Άσκηση 5η:

Συμπληρώστε τη συνάρτηση ***nearestK***, η οποία, με αναδρομικό τρόπο, εντοπίζει και επιστρέφει τα k σημεία του kd-tree που είναι κοντινότερα στο κέντρο της σφαίρας.

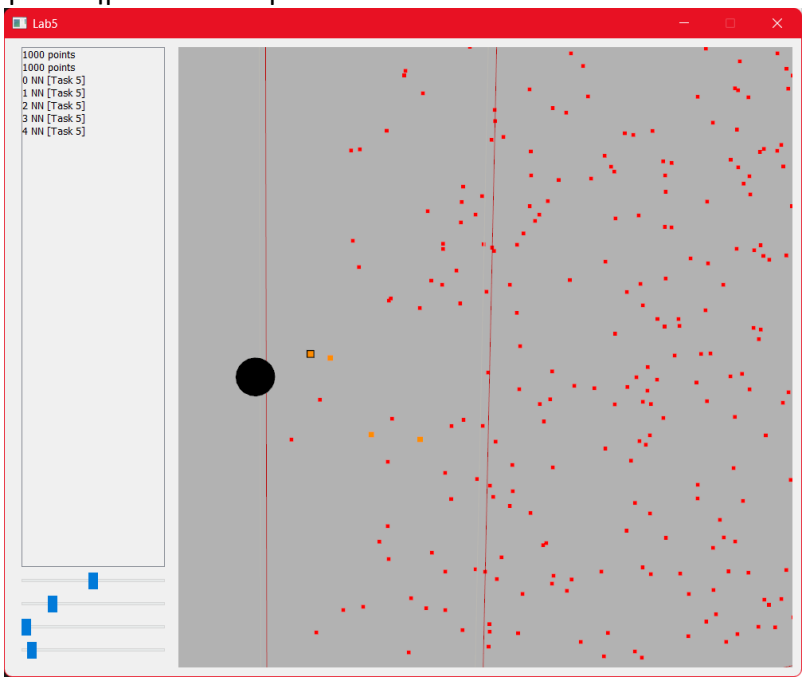
Απάντηση:

Η συνάρτηση `nearestK` υλοποιήθηκε έτσι ώστε αναδρομικά να βρίσκει τους k πλησιέστερους γείτονες του `σημείου self.p = Point3D((-1, 0, 0)) {κέντρο της σφαίρας}` μέσω του kd-tree. Αναλυτικότερα, **επεκτείνει την λογική της συνάρτησης `nearestNeighbor` χρησιμοποιώντας max heap**!

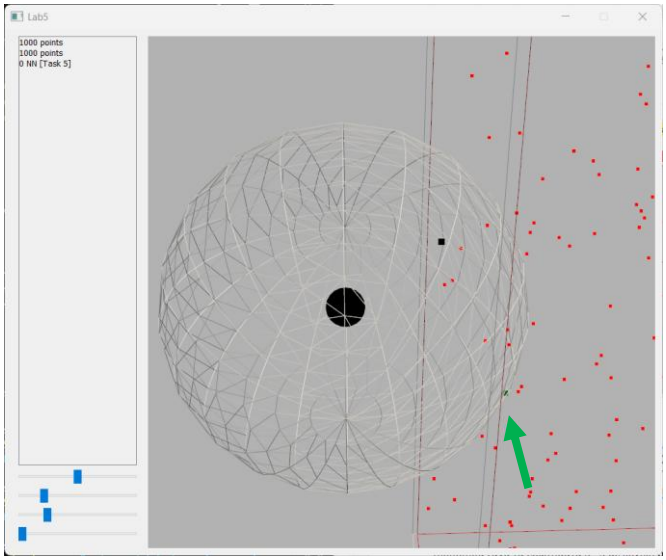
Συγκεκριμένα, κάθε φορά που επισκέπτεται έναν κόμβο του δέντρου, υπολογίζει την απόσταση του pivot από το **σημείο** και:

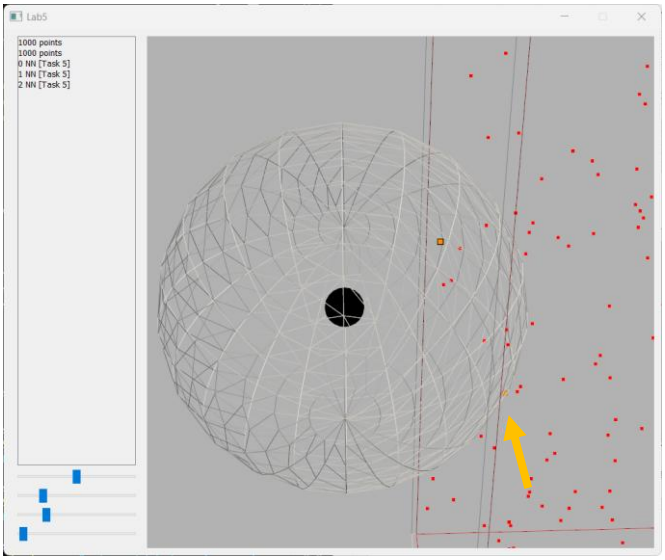
- αν το heap έχει λιγότερα από k σημεία, προστίθεται κανονικά,
- διαφορετικά, συγκρίνεται με το πιο μακρινό σημείο του heap και το αντικαθιστά αν είναι πιο κοντινό!

Το heap χρησιμοποιείται ως max heap, παρότι το heapq της Python είναι min heap, κάνοντας απλώς τις αποστάσεις αρνητικές! Έτσι, στην κορυφή του heap έχουμε πάντα το πιο μακρινό από τα k σημεία.



✍ Με την βοήθεια της σφαίρας του Task 3, για k = 2, <u>έχουμε:</u>





Άσκηση 6η:

> Υλοποιήστε τα ερωτήματα 3-5 χωρίς να χρησιμοποιήσετε το kd-tree. Συγκρίνετε το χρόνο εκτέλεσης των συναρτήσεων και παρουσιάστε τα αποτελέσματά σας για διάφορα πλήθη σημείων. Εξηγείστε τι παρατηρείτε.

Απάντηση:

Υλοποίησα ξανά τα **ερωτήματα 3 - 5 με λογική brute force**! Συγκεριμένα:

- Για το ερώτημα 3, υλοποίησα τη συνάρτηση `inSphere_brute_force`, η οποία υπολογίζει την απόσταση κάθε σημείου από το κέντρο της σφαίρας και **κρατάει μόνο όσα βρίσκονται εντός της**!

`{d_sq <= radius_sq}`

- Για το ερώτημα 4, η `nearestNeighbor_brute_force` βρίσκει το κοντινότερο σημείο στο test_pt υπολογίζοντας την ευκλείδεια απόσταση όλων των σημείων και **επιστρέφει το μικρότερο**.

- Για το ερώτημα 5, η `nearestK_brute_force` ταξινομεί όλα τα σημεία με βάση την απόσταση από το test_pt και **επιστρέφει τα k κοντινότερα**.

🖬 Αποτέλεσμα εκτέλεσης BenchTest:

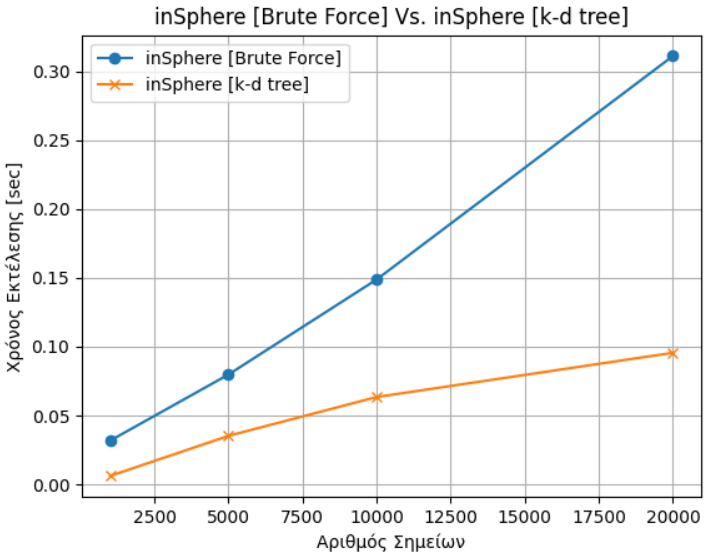
```
BenchTest...
Χρόνος εκτέλεσης: 0.031665 δευτερόλεπτα | Αριθμός σημείων:   1000 | Συνάρτηση: inSphere_brute_force
Χρόνος εκτέλεσης: 0.006022 δευτερόλεπτα | Αριθμός σημείων:   1000 | Συνάρτηση: inSphere
Χρόνος εκτέλεσης: 0.079813 δευτερόλεπτα | Αριθμός σημείων:   5000 | Συνάρτηση: inSphere_brute_force
Χρόνος εκτέλεσης: 0.035203 δευτερόλεπτα | Αριθμός σημείων:   5000 | Συνάρτηση: inSphere
Χρόνος εκτέλεσης: 0.148888 δευτερόλεπτα | Αριθμός σημείων:  10000 | Συνάρτηση: inSphere_brute_force
Χρόνος εκτέλεσης: 0.063442 δευτερόλεπτα | Αριθμός σημείων:  10000 | Συνάρτηση: inSphere
Χρόνος εκτέλεσης: 0.311363 δευτερόλεπτα | Αριθμός σημείων:  20000 | Συνάρτηση: inSphere_brute_force
Χρόνος εκτέλεσης: 0.095578 δευτερόλεπτα | Αριθμός σημείων:  20000 | Συνάρτηση: inSphere

Χρόνος εκτέλεσης: 0.017460 δευτερόλεπτα | Αριθμός σημείων:   1000 | Συνάρτηση: nearestNeighbor_brute_force
Χρόνος εκτέλεσης: 0.002013 δευτερόλεπτα | Αριθμός σημείων:   1000 | Συνάρτηση: nearestNeighbor
Χρόνος εκτέλεσης: 0.060994 δευτερόλεπτα | Αριθμός σημείων:   5000 | Συνάρτηση: nearestNeighbor_brute_force
Χρόνος εκτέλεσης: 0.005906 δευτερόλεπτα | Αριθμός σημείων:   5000 | Συνάρτηση: nearestNeighbor
Χρόνος εκτέλεσης: 0.126397 δευτερόλεπτα | Αριθμός σημείων:  10000 | Συνάρτηση: nearestNeighbor_brute_force
Χρόνος εκτέλεσης: 0.001757 δευτερόλεπτα | Αριθμός σημείων:  10000 | Συνάρτηση: nearestNeighbor
Χρόνος εκτέλεσης: 0.583457 δευτερόλεπτα | Αριθμός σημείων:  20000 | Συνάρτηση: nearestNeighbor_brute_force
Χρόνος εκτέλεσης: 0.002012 δευτερόλεπτα | Αριθμός σημείων:  20000 | Συνάρτηση: nearestNeighbor

Χρόνος εκτέλεσης: 0.002022 δευτερόλεπτα | Αριθμός σημείων:   1000 | Συνάρτηση: nearestK_brute_force
Χρόνος εκτέλεσης: 0.002380 δευτερόλεπτα | Αριθμός σημείων:   1000 | Συνάρτηση: nearestK
Χρόνος εκτέλεσης: 0.047410 δευτερόλεπτα | Αριθμός σημείων:   5000 | Συνάρτηση: nearestK_brute_force
Χρόνος εκτέλεσης: 0.000000 δευτερόλεπτα | Αριθμός σημείων:   5000 | Συνάρτηση: nearestK
Χρόνος εκτέλεσης: 0.132127 δευτερόλεπτα | Αριθμός σημείων:  10000 | Συνάρτηση: nearestK_brute_force
Χρόνος εκτέλεσης: 0.004019 δευτερόλεπτα | Αριθμός σημείων:  10000 | Συνάρτηση: nearestK
Χρόνος εκτέλεσης: 0.225398 δευτερόλεπτα | Αριθμός σημείων:  20000 | Συνάρτηση: nearestK_brute_force
Χρόνος εκτέλεσης: 0.010108 δευτερόλεπτα | Αριθμός σημείων:  20000 | Συνάρτηση: nearestK
```

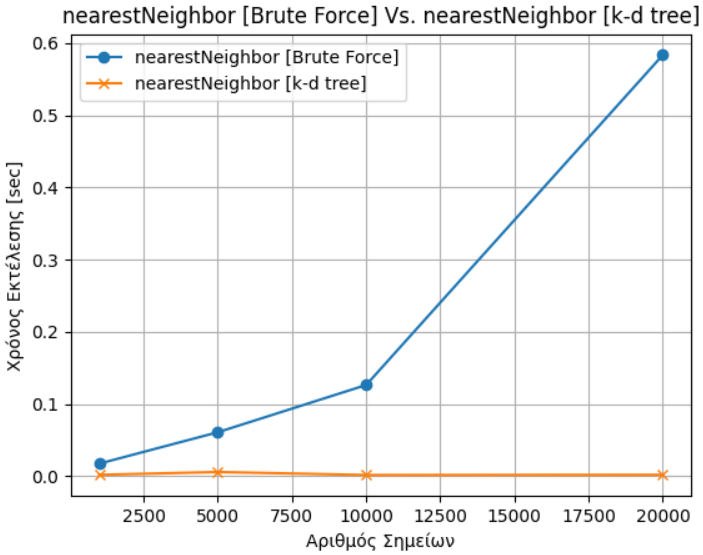
Βάση της μεθοδολογίας που ακολούθησα και στο Lab - 2, παρατηρούμε ότι:

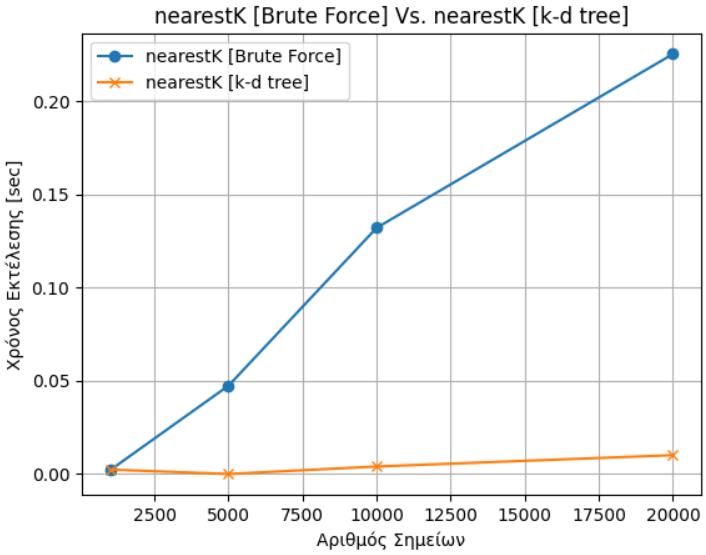
- Οι υλοποιήσεις με **brute force έχουν πολυπλοκότητα O(n) ή και O(n log n)** (ανάλογα με τη συνάρτηση – π.χ. λόγω ταξινόμησης στο nearestK).

- Αντίθετα, οι υλοποιήσεις με **χρήση kd-tree έχουν στην πράξη σχεδόν γραμμική απόδοση**, καθώς εξετάζεται μόνο ένα μικρό υποσύνολο των σημείων!

Γραφικές:


inSphere [Brute Force] Vs. inSphere [k-d tree]
inSphere [Brute Force]
inSphere [k-d tree]
Χρόνος Εκτέλεσης [sec]
0.00
0.05
0.10
0.15
0.20
0.25
0.30
2500
5000
7500
10000
12500
15000
17500
20000
Αριθμός Σημείων


nearestNeighbor [Brute Force] Vs. nearestNeighbor [k-d tree]
nearestNeighbor [Brute Force]
nearestNeighbor [k-d tree]
Χρόνος Εκτέλεσης [sec]
0.0
0.1
0.2
0.3
0.4
0.5
0.6
2500
5000
7500
10000
12500
15000
17500
20000
Αριθμός Σημείων


nearestK [Brute Force] Vs. nearestK [k-d tree]
nearestK [Brute Force]
nearestK [k-d tree]
Χρόνος Εκτέλεσης [sec]
0.00
0.05
0.10
0.15
0.20
2500
5000
7500
10000
12500
15000
17500
20000
Αριθμός Σημείων