# Εργασία Υπολογιστική Γεωμετρία

## Εαρινό Εξάμηνο 2023

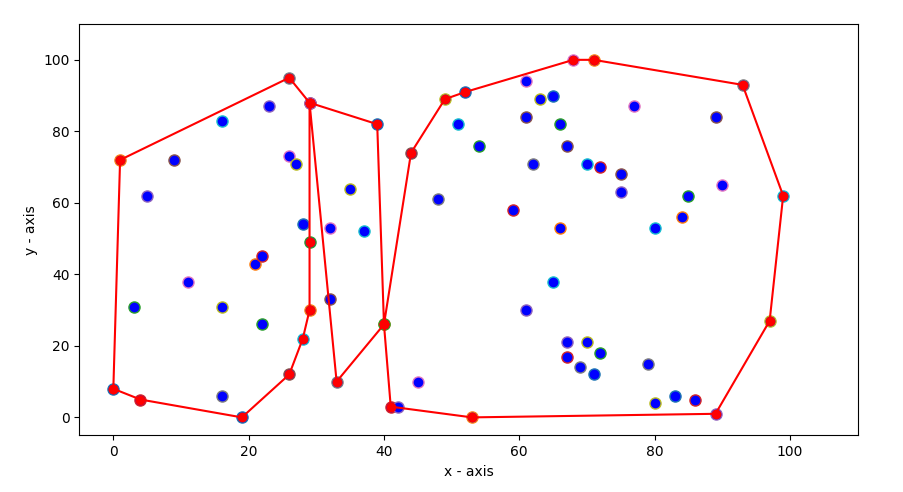
## Ονοματεπώνυμο: Αλέξανδρος Γαλανός

## Αριθμός Μητρώου : 1115201900032

Συμπεριφορά σε Εκφυλισμένες Περιπτώσεις

* Αυξητικός Αλγόριθμος

Ο αυξητικός Αλγόριθμος δε λειτουργεί σωστά στην περίπτωση που τα σημεία δεν είναι σε γενική θέση δηλαδή είναι συνευθειακά. Στην παρακάτω εικόνα φαίνεται μία τέτοια περίπτωση.



Στις περιπτώσεις κινητής υποδιαστολής ίσως περιμέναμε κάποιο σφάλμα στρογγύλευσης ωστόσο ο αλγόριθμος μέσω της γλώσσας python λειτουργεί σωστά και με αριθμούς κινητής υποδιαστολής.

* Αλγόριθμος Περιτυλίγματος

Ο αλγόριθμος περιτυλίγματος λειτουργεί σωστά στην περίπτωση που τα σημεία βρίσκονται σε γενική θέση όπως και για αριθμούς κινητής υποδιαστολής.

* Αλγόριθμος Διαίρει και Βασίλευε

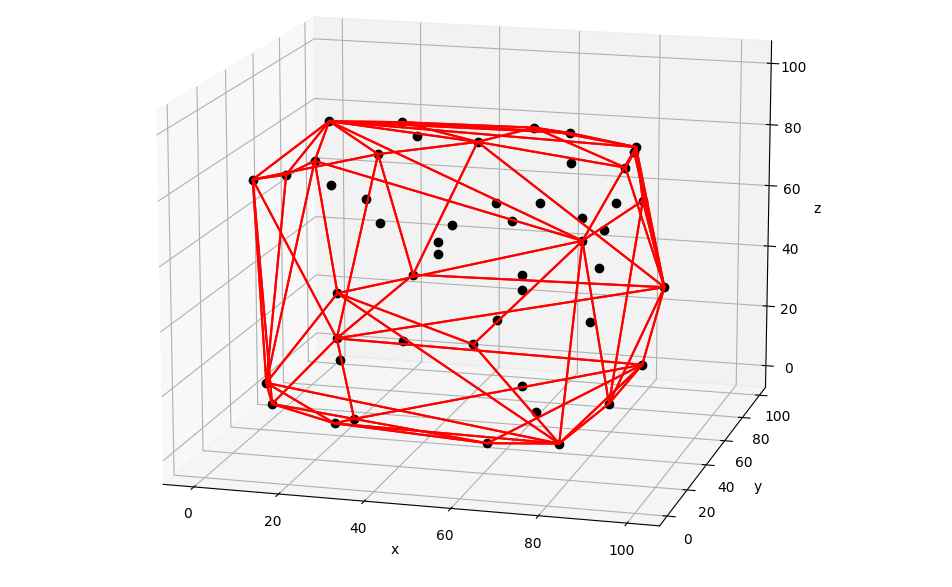
Ο αλγόριθμος Divide and Conquer λειτουργεί επίσης σωστά στις εκφυλισμένες περιπτώσεις.

* Αλγόριθμος QuickHull

O QuickHull επίσης λειτουργεί σωστά στις εκφυλισμένες περιπτώσεις. Σχόλιο τα σημεία του κυρτού περιβλήματος τα αποθηκεύω σε μία δομή set διότι σε λίστα προέκυπταν διπλότυπα σημεία. Ωστόσο μέσω του set τα σημεία δεν αποθηκεύονται σε ωρολόγια διάταξη ώστε να μπορέσω να εκτυπώσω το γράφημα αλλά έχω ελέγξει πως είναι τα σωστά σημεία του κυρτού περιβλήματος.

Convex Hull σε 3 διαστάσεις

Για την υλοποίηση του convex Hull σε τρεις διαστάσεις έχω χρησιμοποιήσει την συνάρτηση “ConvexHull()” της spicy. Παράδειγμα γραφικής παράστασης :



Εφαρμογή 1

1. (α’) Ο κάθε αλγόριθμος που έχω υλοποιήσει επιστρέφει μία λίστα σημείων και εμφανίζει το κυρτό περίβλημα των σημείων.

(β’) Έχω υλοποιήσει οπτικοποίηση των βημάτων του Αθξητικού Αλγόριθμου. Για να γίνει η οπτικοποίση κάθε βήματος θέτουμε “visualization=True”:



2.

|  |  |  |
| --- | --- | --- |
|  | Χρόνος Εκτέλεσης(80 σημεία) | Πολυπλοκότητα |
| Αυξητικός Αλγόριθμος | 0.00022554397583007812 | Ο(nlogn) |
| Gift Wrapping | 0.0005962848663330078 | O(n2) |
| Divide and Conquer | 0.0003712177276611328 | Ο(nlogn) |
| QuickHull | 0.0018698061828613281 | Ο(nlogn)  (Worst Case O(n2)) |

Δεδομένου των εκφυλισμένων περιπτώσεων που αναφέρθηκαν παραπάνω και βάση πολυπλοκότητας θεωρώ πως ο πιο κατάλληλος αλγόριθμος είναι ο Divide and Conquer.

Υλοποίηση 2: Γεωμετρική Αναζήτηση

Έχω υλοποιήσει τον αλγόριθμο γεωμετρικής Αναζήτησης μέσω kd-trees. Ενδεικτικό παράδειγμα. Για τα σημεία : points = [[3, 6], [17, 15], [13, 16], [6, 12], [9, 1], [2, 7], [10, 19]]

και περιοχή area\_=area(xmin=1,xmax=10,ymin=1,ymax=20) έχουμε το αποτέλεσμα

[[3, 6], [2, 7], [6, 12], [9, 1], [10, 19]].

Σχόλιο: Στον αλγόριθμο που έχω υλοποιήσει τα σημεία στο kd-tree εισάγονται με τη σειρά με αποτέλεσμα οι κατακόρυφες και οριζόντιες ευθείες να μη γίνονται με βάση τη διάμεση τετμημένη και διάμεση τεταγμένη των σημείων αντίστοιχα. Αυτό έχει ως αποτέλεσμα το δέντρο να μην είναι ισορροπημένο.

**Πολυπλοκότητα:** Ενα kd-δέντρο για ένα σύνολο P από n σημεία στο επίπεδο καταλαμβάνει χώρο O(n) και μπορεί να κατασκευαστεί σε χρόνο O(nlogn). Ένα ορθογωνικό εκτασιακό ερώτημα προς το kd-δέντρο μπορεί να απαντηθεί σε χρόνο O( n1/2 + k), όπου k είναι το πλήθος των αναφερόμενων σημείων.

Εφαρμογή 2

Παράδειγμα εκτέλεσης του προγράμματος μου , είχα τα σημεία :

points : [(30, 21), (77, 76), (22, 47), (67, 100), (19, 9), (97, 69), (98, 97), (24, 28), (91, 0), (53, 36), (37, 85), (18, 66), (20, 65), (50, 4), (66, 86), (27, 25), (87, 68), (29, 2), (21, 95), (38, 88), (18, 8), (75, 52), (95, 94), (89, 8), (54, 68), (51, 88), (50, 23), (5, 76), (76, 81), (40, 55), (79, 3), (24, 30), (2, 67), (60, 35), (96, 82), (44, 56), (86, 76), (25, 55), (62, 61), (25, 84), (62, 56), (30, 20), (68, 99), (28, 7), (62, 45), (10, 74), (43, 54), (41, 64), (62, 35), (56, 8), (28, 34), (69, 82), (99, 34), (5, 56), (27, 76), (84, 8), (77, 7), (20, 17), (11, 72), (19, 0)]

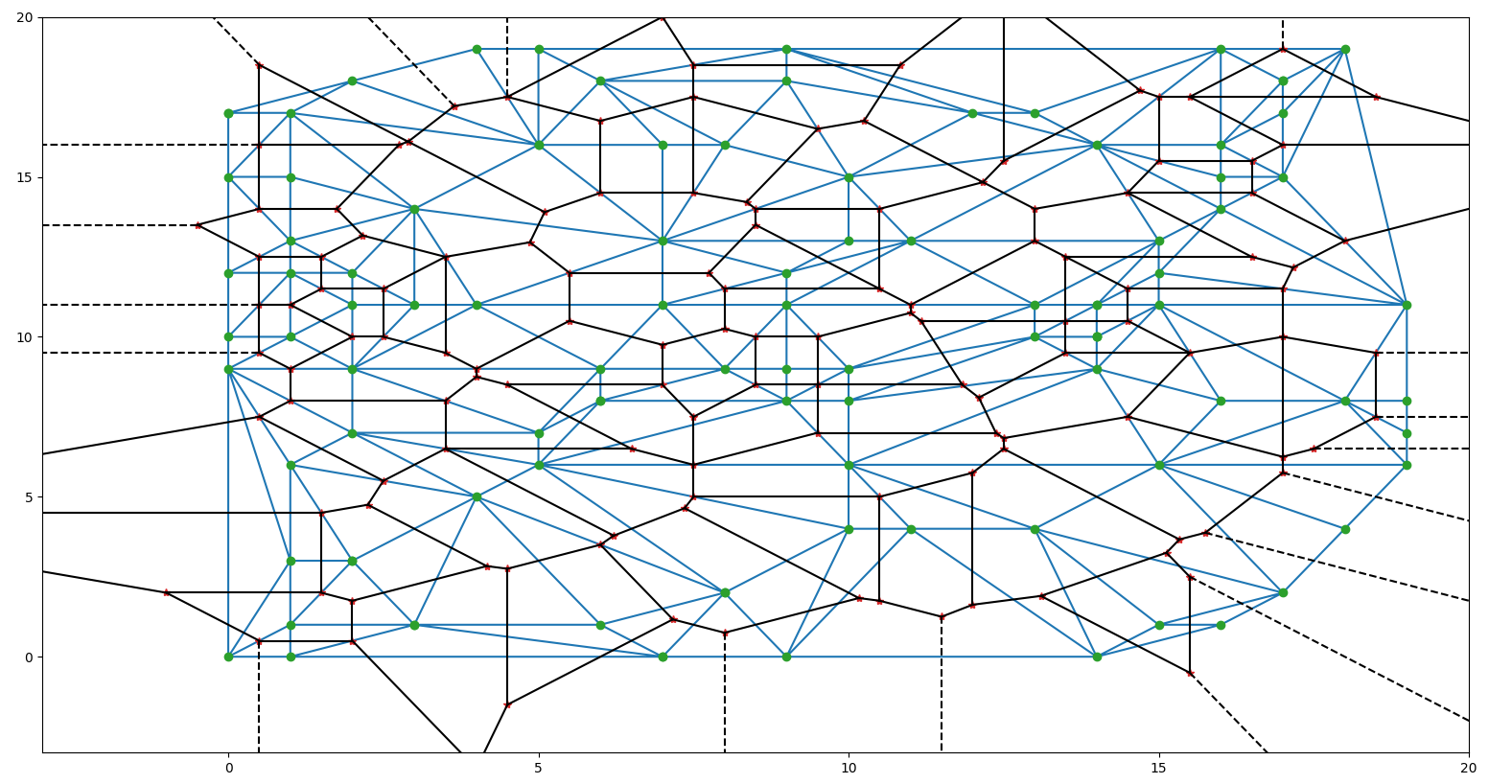
Δεδομένου ορθογωνίου έκτασης: area : xmin= 1 xmax= 30 ymin= 1 ymax= 30

Είχα ως αποτέλεσμα τα σημεία: [(30, 21), (19, 9), (18, 8), (24, 28), (20, 17), (27, 25), (29, 2), (28, 7), (30, 20), (24, 30)]

Η συνάρτηση “def fully\_contained(node,area)” ελέγχει αν το τρέχον δέντρο εμπεριέχεται ολόκληρο μέσα στην έκταση “area”. Αντίστοιχα η συνάρτηση “def intersects(node.area)” ελέγχει αν το τρέχον μέρος του δέντρου τέμνει την έκταση “area”. Η συνάρτηση “def find\_min\_x(root)” επιστρέφει τον κόμβο κάτω από τον κόμβο root με τη μικρότερη τετμημένη x. Αντίστοιχη δουλειά κάνουν και οι υπόλοιπες συναρτήσεις. Η συνάρτηση “def traverse(node)” κάνει αναφορά του υποδέντρου από τον κόμβο node.

Εφαρμογή 3

Στην εφαρμογή 3 χρησιμοποιώντας έτοιμες τις συναρτήσεις της python για το διάγραμμα Voronoi και την τριγονωποίηση Delaunay δεδομένου 20 σημείων στο επίπεδο το διάγραμμα που προκύπτει είναι για παράδειγμα το ακόλουθο.



Με μπλε χρώμα είναι οι ακμές των τριγώνων Delaunay και πράσινο οι κορυφές των τριγώνων που αντιστοιχούν σε εστίες κελιών Voronoi. Οι διακεκομμένες γραμμές αποτελούν ακμές του διαγράμματος Voronoi που εκτείνονται στο άπειρο. Οι κορυφές του διαγράμματος Voronoi είναι με ένα κόκκινο αστέρι και οι ακμές με μαύρο χρώμα.

**Πολυπλοκότητα:**

Κατασκευή διαγράμματος Voronoi : O(nlogn)

Τριγονοποίηση Dealaunay : Ο(nlogn) (Χείριστη Περίπτωση Ο(n2))

|  |  |
| --- | --- |
|  | Χρόνος(sec) |
| 20 σημεία | 0.22033476829528809 |
| 40 σημεία | 0.2325150966644287 |
| 80 σημεία | 0.267317533493042 |
| 160 σημεία | 0.3270692825317383 |

Βάση αυτών των αποτελεσμάτων παρατηρούμε πως ο χρόνος αυξάνεται προσθέτοντας όλο και πιο πολλά σημεία. Βάσει αυτών φαίνεται ότι ο χρόνος αυξάνεται με γεωμετρική πρόοδο καθώς αυξάνεται ο αριθμός των σημείων. Αυτό υποδηλώνει πως ο αλγόριθμος έχει σχέση με την πολυπλοκότητα Ο(nlogn).

Ωστόσο, παρατηρώντας τους χρόνους εκτέλεσης, πρέπει να έχουμε υπόψη ότι οι ακριβείς τιμές μπορεί να διαφέρουν ανάλογα με το περιβάλλον εκτέλεσης, την υλοποίηση των βιβλιοθηκών που χρησιμοποιούνται, και τις ιδιαιτερότητες του συστήματος. Γενικά, όμως, περιμένουμε να δούμε αύξηση στον χρόνο εκτέλεσης καθώς αυξάνουμε τον αριθμό των σημείων στην είσοδο του προγράμματος.