Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών
Τμήμα Πληροφορικής και Τηλεπικοινωνιών

Υπολογιστική Γεωμετρία
Υπολογιστική Εργασία

Ονοματεπώνυμο : Κυριακάκος Νικόλαος Αριθμός Μητρώου : 1115202200083

Περιεχόμενα

1 Κυρτό Περίβλημα

1.1 Σύσταση Προγράμματος

Το πρόγραμμα συνίσταται από τα εξής:

- Φάκελος src: Εδώ βρίσκεται ο κώδικας και αποτελείται από τα εξής:
  - main.py, στο οποίο βρίσκεται η κύρια συνάρτηση που εκτελείται
  - utils.py, στο οποίο βρίσκονται οι βοηθητικές συναρτήσεις που καλούνται από την main
  - Φάκελος algorithms: Εδώ περιέχονται οι υλοποιήσεις των 4 αλγορίθμων
- Φάκελος files: Εδώ βρίσκονται οι εξής 2 φάκελοι:
  - points, στον οποίο βρίσκονται τα αρχεία που περιέχουν τα τυχαία σημεία
  - output, στον οποίο βρίσκονται τα αρχεία με τα αποτελέσματα που προκύπτουν από την εκτέλεση των αλγορίθμων

1.2 Ροή Προγράμματος

Αρχικά, εμφανίζονται στον χρήστη οι επιλογές των αλγορίθμων:

- 1.Αυξητικός Αλγόριθμος
- 2.Αλγόριθμος του Περιτυλίγματος
- 3.Αλγόριθμος Διαίρει και Βασίλευε
- 4.Αλγόριθμος QuickHull 2 διαστάσεων
- 5.Αλγόριθμος QuickHull 3 διαστάσεων
- 6.Όλοι οι Αλγόριθμοι 2 διαστάσεων

Έπειτα, ο χρήστης διαλέγει ποιος αλγόριθμος θέλει να εκτελεστεί και ερωτάται αν επιθυμεί να δει τα βήματα (σχηματικά) της εκτέλεσής του, καθώς και αν επιθυμεί στα σημεία εισόδου να υπάρχουν συνεθειακά σημεία ή όχι.
Στη συνέχεια ελέγχεται αν υπάρχει το αρχείο που περιλαμβάνει τα τυχαία σημεία και αν δεν υπάρχει, δημιουργείται.
Έπειτα, εκτελείται ο αντίστοιχος αλγόριθμος και επιστρέφεται το κυρτό περίβλημα των σημείων, το οποίο επίσης καταγράφεται στο αντίστοιχο αρχείο στον φάκελο output.

1.3 Αλγόριθμοι

1.3.1 Αυξητικός Αλγόριθμος (Graham Scan)

- Είσοδος: Ένα σύνολο σημείων σε γενική θέση
- Έξοδος: Μία λίστα των κορυφών του κυρτού περιβλήματος σε ωρολόγια διάταξη
- Πολυπλοκότητα: O(nlog n) εξαιτίας της ταξινόμησης
- Βήματα:

  - Τα σημεία ταξινομούνται με βάση την αύξουσα σειρά των τετμημένων τους. Αν δύο σημεία έχουν ίδια τετμημένη τότε αυτά ταξινομούνται με βάση την τεταγμένη τους.
  - Κατασκευάζουμε το Λάνω (Lup) προσθέτοντας στον Λάνω αρχικά τα δύο πρώτα σημεία και, έπειτα, κάθε σημείο από το τρίτο και μετά. Αν το Λάνω περιέχει περισσότερα από 2 σημεία και η στροφή που ορίζουν δεν είναι δεξιά, αφαιρούμε το μεσαίο.
  - Κατασκευάζουμε το Lκάτω (Ldown) προσθέτοντας στο Lκάτω αρχικά τα δυο τελευταία σημεία και, έπειτα, κάθε σημείο από το τρίτο από το τέλος και πριν. Ομοίως, αν το Lκάτω περιέχει περισσότερα από 2 σημεία και η στροφή που ορίζουν δεν είναι δεξιά, αφαιρούμε το μεσαίο.
  - Αφαιρούμε από το Lκάτω το πρώτο και τελευταίο σημείο, έτσι ώστε να μην υπάρχουν διπλότυπα στο τελικό μας περίβλημα.
  - Επισυνάπτουμε το Λάνω και το Lκάτω και τα επιστρέφουμε.

1.3.2 Αλγόριθμος του Περιτυλίγματος

- Είσοδος: Ενα σύνολο σημείων σε γενική θέση
- Έξοδος: Μια αλυσίδα ακμών και κορυφών του κυρτού πεπριβλήματος
- Πολυπλοκότητα: O(nh) όπου h το πλήθος των κορυφών του κυρτού περιβλήματος
- Βήματα:

  - Αρχικοποιούμε την αλυσίδα με το λεξικογραφικά μικρότερο σημείο.
  - Σε μια δομή επανάληψης κάνουμε τα εξής:
    - Για κάθε σημείο το οποίο είναι διάφορο του τρέχοντος σημείου r ελέγχουμε αν η στροφή είναι ίδια με τη φορά των δεικτών του ρολογιού (δεξιόστροφα - CW).
    - Ελέγχουμε αν φτάσαμε στο αρχικό μας σημείο, οπότε σταματάμε την επανάληψη.
    - Σε διαφορετική περίπτωση, θέτουμε το τρέχον σημείο ως το επόμενο που βρήκαμε και το προσθέτουμε στην αλυσίδα.

1.3.3 Αλγόριθμος Διαίρει και Βασίλευε

- Είσοδος: Ενα σύνολο σημείων σε γενική θέση
- Εξοδος: Μία λίστα των κορυφών του κυρτού περιβλήματος
- Πολυπλοκότητα: O(nlog n)
- Βήματα:

  - Τα σημεία ταξινομούνται με βάση την αύξουσα σειρά των τετμημένων τους. Αν δύο σημεία έχουν ίδια τετμημένη τότε αυτά ταξινομούνται με βάση την τεταγμένη τους
  - Αναδρομικά κάνουμε τα εξής:
    - Χωρίζουμε τα σημεία σε δύο υποσύνολα A,B τα οποία περιέχουν τα $\lceil \frac{n}{2} \rceil$ και $\lfloor \frac{n}{2} \rfloor$ σημεία αντίστοιχα.
    - Καλούμε την αναδρομική συνάρτηση για τα υποσύνολα A και B αντίστοιχα.
    - Υπολογίζουμε την άνω και κάτω γέφυρα.
    - Επιστρέφουμε τα σημεία αυτά που βρίσκονται μεταξύ των 2 γεφυρών για κάθε ένα από τα 2 κυρτά περιβλήματα.
    - Επισυνάπτουμε τα 2 κυρτά περιβλήματα σε ένα, το οποίο επιστρέφουμε.

- Βήματα προσδιορισμού Άνω Γέφυρας:

  - Προσδιορίζουμε το δεξιότερο σημείο του A (A_i) και το αριστερότερο του B (B_j)
  - Εφόσον η ευθεία A_i, A_{i+1} δεν αφήνει στο ίδιο ημιεπίπεδο τα σημεία του A και το B_j, αυξάνουμε το i κατά 1.
  - Εφόσον η ευθεία B_j, B_{j-1} δεν αφήνει στο ίδιο ημιεπίπεδο τα σημεία του A και το A_i, μειώνουμε το j κατά 1.
  - Σε περίπτωση που δεν μεταβλήθηκε το i ή το j ή και τα 2, επιστρέφουμε τα A_i, B_j ως άνω γέφυρα, αλλιώς επιστρέφουμε, εκτελούμε πάλι τα 2 προηγούμενα βήματα.

Ο αντίστοιχος αλγόριθμος χρησιμοποιείται και για τον προσδιορισμό της Κάτω Γέφυρας. Τα σημεία, στα οποία είναι διαφορετικός είναι ότι αρχικά ελέγχονται τα B_j, B_{j+1}, A_i και αυξάνουμε το j κατά 1. Έπειτα, ελέγχονται A_{i-1}, A_i, B_j και μειώνεται το i κατά 1.

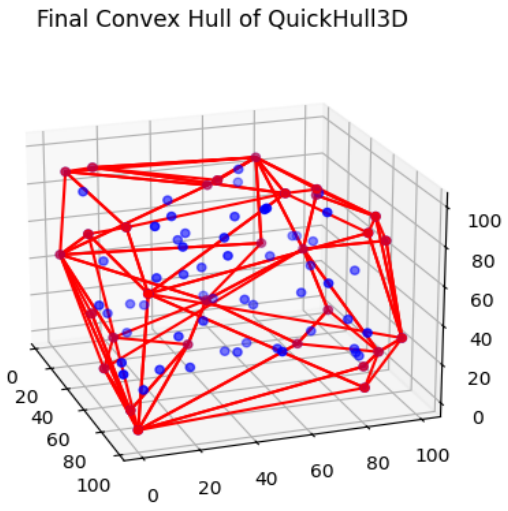
1.3.4 Αλγόριθμος QuickHull 2 διαστάσεων

Στην υλοποίηση χρησιμοποιήθηκε η συνάρτηση ConvexHull που έχει υλοποιηθεί στη βιβλιοθήκη scipy.spatial της Python και υπολογίζει το κυρτό περίβλημα με τον αλγόριθμο QuickHull.
Η συγκεκριμένη συνάρτηση δέχεται τόσο σημεία στον χώρο των2, καθώς και των 3 διαστάσεων και συνεπώς, χρησιμοποιήθηκε και για τις 2 περιπτώσεις.
Ωστόσο, είναι απαραίτητο να αναφερθούν τα βήματα που ο αλγόριθμος ακολουθεί, καθώς και τα χαρακτηριστικά του.
- Είσοδος: Ενα σύνολο σημείων σε γενική θέση
- Εξοδος: Μια αλυσίδα ακμών και κορυφών του κυρτού πεπριβλήματος
- Πολυπλοκότητα: O(nlog r) όπου r ο αριθμός των επεξεργασμένων σημείων (αυτών που έχουν ελεγχθεί). Συνεπώς, κατά μέση περίπτωση O(nlog r), αλλά μπορεί ενδεχομένως να φτάσει χείριστη περίπτωση $O(n^2)$
- Βήματα:

- Αρχικά, βρίσκουμε τα 2 πιο μακρινά σημεία με βάση τις συνεταγμένες τους (μεγαλύτερο-μικρότερο x και y), καθώς αυτά σίγουρα θα αποτελούν μέρος του περιβλήματος.
- Ενώνοντας τα δύο αυτά σημεία με μια γραμμή, χωρίζουμε τα σημεία σε 2 υποσύνολα.
- Αναδρομικά γίνονται τα εξής (μέχρι να συμπληρωθεί το περίβλημα):
  - Βρίσκουμε το σημείο που απέχει τη μεγαλύτερη απόσταση από την παραπάνω γραμμή και σχηματίζεται ένα τρίγωνο.
  - Τα σημεία που βρίσκονται εντός του τριγώνου αγνοούνται, καθώς δεν μπορούν να αποτελούν μέρος του περιβλήματος.
  - Καλούμε την αναδρομική συνάρτηση για τις δυο πλευρές του τριγώνου που προέκυψε.

1.3.5 Αλγόριθμος QuickHull 3 διαστάσεων

Η συγκεκριμένη υλοποίηση χρησιμοποιεί και αυτή την συνάρτηση ConvexHull που έχει υλοποιηθεί στη βιβλιοθήκη scipy.
Η μόνη διαφορά εδώ είναι ότι τα σημεία ανήκουν στον τρισδιάστατο χώρο, οπότε μόνο αυτό το σημείο έχει αλλάξει στον κώδικα προκειμένου να γίνει σωστή διαχείριση και εμφάνισή τους.
Επίσης, έχει δημιουργηθεί διαφορετική συνάρτηση για παρουσίαση των στοιχείων.
Ένα ενδεικτικό αποτέλεσμα του αλγορίθμου αυτού για 3 διαστάσεις είναι το ακόλουθο:



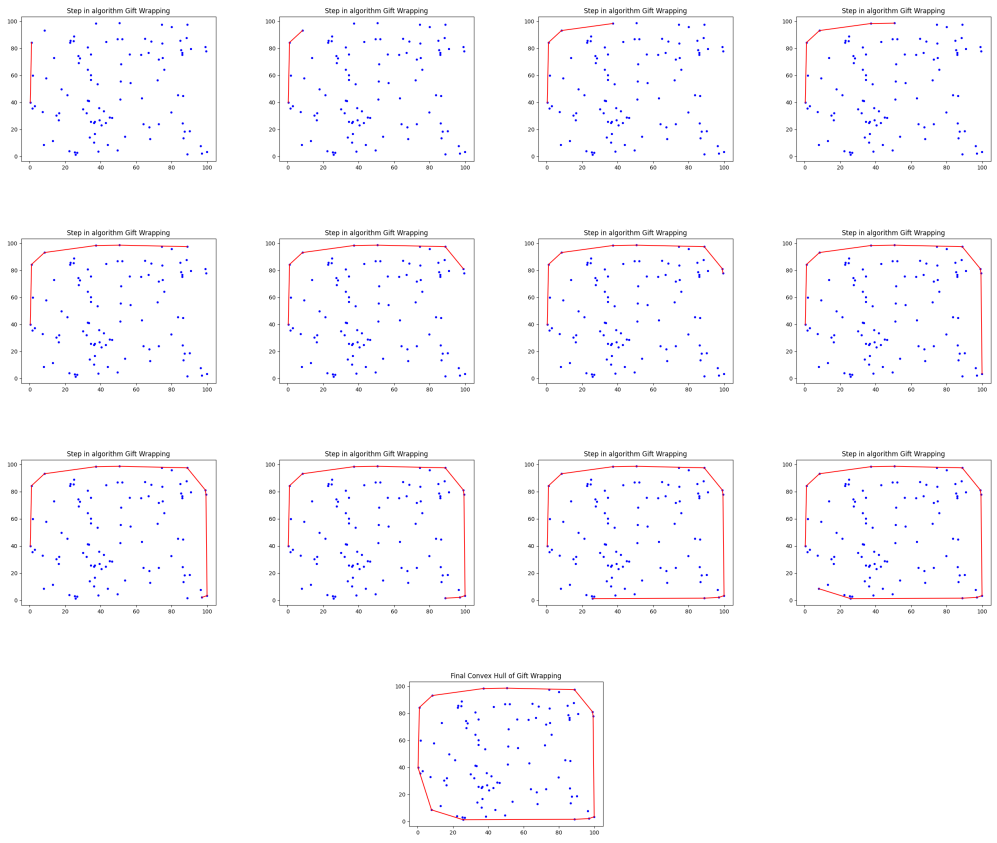
1.4 Αποτελέσματα ενδεικτικής εκτέλεσης

1.4.1 Αριθμητικά

Θεωρούμε 100 τυχαία σημεία στο επίπεδο. Χρησιμοποιώντας όλους τους αλγορίθμους παρατηρούμε ότι προκύπτει το ίδιο αποτέλεσμα, δηλαδή μια λίστα που περιέχει τα ίδια τελικά σημεία ως περίβλημα.
Ωστόσο, παρατηρούμε ότι η σειρά με την οποία είναι καταγεγραμμένα τα σημεία στο αρχείο του κάθε αλγορίθμου είναι διαφορετική. Αυτό προκύπτει από τον διαφορετικό τρόπο με τον οποίο ο κάθε αλγόριθμος παράγει τα τελικά σημεία.
Το παραπάνω φαίνεται και από τα περιεχόμενα των αρχείων των αλγορίθμων, οι οποίοι εκτελέστηκαν με βάση το ίδιο σύνολο σημείων που βρίσκεται στο αρχείο points.txt. Μπορεί βέβαια να επιβεβαιωθεί και με τη διαγραφή του περιεχομένου (ή του αρχείου points.txt) και με την εκ νέου εκτέλεση των αλγορίθμων.

1.4.2 Οπτικοποίηση βημάτων

Όλοι οι αλγόριθμοι (εκτός του QuickHull) υποστηρίζουν οπτικοποίηση των βημάτων για την κατασκευή του κυρτού περιβλήματος.
Στο παραπάνω αποτελεί εξαίρεση ο QuickHull, διότι χρησιμοποιεί την συνάρτηση ConvexHull η οποία είναι ήδη υλοποιημένη. Ωστόσο, μπορούμε να δούμε το τελικό αποτέλεσμα που παράγει, το οποίο είναι το ίδιο με αυτό των υπολοίπων.
Ενδεικτικά, παρακάτω παρουσιάζονται τα βήματα που εκτελεί ο Αλγόριθμο του Περιτυλίγματος για την εύρεση του κυρτού περιβλήματος:



Gift Wrapping

1.5 Συνευθειακά σημεία

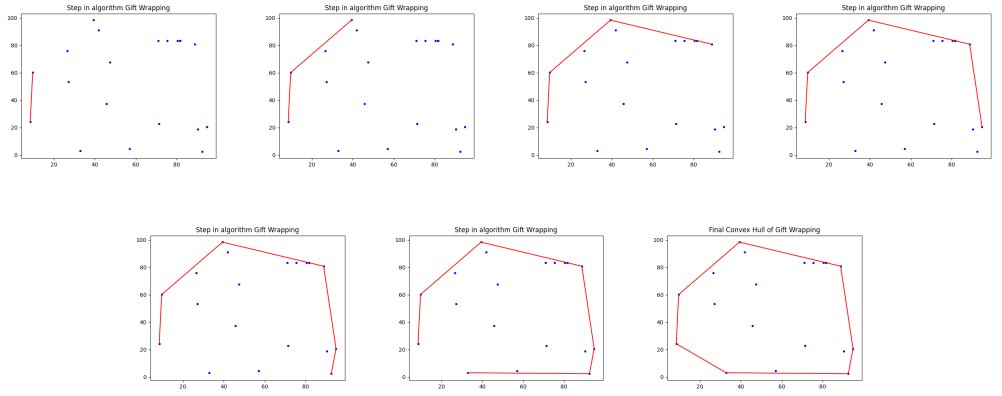
- **Αλγόριθμος Αυξητικός**
  - Όταν προστίθενται συνευθειακά σημεία, ο αλγόριθμος πρέπει να ελέγξει και να αφαιρέσει τα σημεία που δεν ανήκουν στο κυρτό περίβλημα.
  - Ο έλεγχος και η ενημέρωση της κυρτής θήκης μπορεί να αυξήσουν τον χρόνο εκτέλεσης, καθώς απαιτείται επιπλέον υπολογισμός για τον προσδιορισμό των σημείων που βρίσκονται εντός ή εκτός του περιβλήματος. Στην χειρότερη περίπτωση, η πολυπλοκότητα παραμένει $O(n^2)$, αλλά η διαχείριση συνευθειακών σημείων μπορεί να αυξήσει τον πραγματικό χρόνο εκτέλεσης.

- **Αλγόριθμος του Περιτυλίγματος**
  - Σε περίπτωση συνευθειακών σημείων, ο αλγόριθμος πρέπει να ελέγξει ποιο από τα σημεία βρίσκεται πιο μακριά, για να αποφύγει την επιλογή ενδιάμεσων σημείων. - Αυτό μπορεί να αυξήσει τον αριθμό των ελέγχων και να επιβραδύνει τον αλγόριθμο.
  Η πολυπλοκότητα του αλγορίθμου παραμένει O(nh), όπου h είναι ο αριθμός των σημείων του κυρτού περιβλήματος, αλλά οι επιπλέον έλεγχοι μπορεί να αυξήσουν τον χρόνο εκτέλεσης σε περιπτώσεις με πολλά συνευθειακά σημεία.

- **Αλγόριθμος Διαίρει και Βασίλευε**
  - Η συγχώνευση των δύο περιβλημάτων μπορεί να γίνει πιο περίπλοκη όταν υπάρχουν συνευθειακά σημεία, καθώς πρέπει να ελεγχθεί αν τα ενδιάμεσα σημεία ανήκουν στο περίβλημα.
  - Ωστόσο, η πολυπλοκότητα της συγχώνευσης παραμένει γενικά ίδια, αν και ο χρόνος εκτέλεσης μπορεί να αυξηθεί λόγω των επιπλέον ελέγχων.
  Συνεπώς, η πολυπλοκότητα του αλγορίθμου παραμένει O(nlogn), αλλά οι επιπλέον έλεγχοι μπορούν να αυξήσουν τον πραγματικό χρόνο εκτέλεσης.

- **Αλγόριθμος QuickHull**
  - Τα συνευθειακά σημεία μπορούν να προκαλέσουν την περιττή επανάληψη της διαδικασίας διαχωρισμού και τον έλεγχο σημείων που δεν ανήκουν στο κυρτό περίβλημα.
  Ο αλγόριθμος πρέπει να διαχειριστεί τα σημεία που βρίσκονται πάνω στη γραμμή και να αποφύγει την επιλογή τους ως τμήμα του κυρτού περιβλήματος.
  - Η πολυπλοκότητα του αλγορίθμου παραμένει O(nlogn) κατά μέσο όρο, αλλά η χειρότερη περίπτωση μπορεί να είναι $O(n^2)$. Οι συνευθειακές περιπτώσεις μπορούν να αυξήσουν τον πραγματικό χρόνο εκτέλεσης λόγω των επιπλέον ελέγχων.

Ενδεικτικά, παρακάτω παρουσιάζονται τα βήματα που εκτελεί ο Αλγόριθμο του Περιτυλίγματος για την εύρεση του κυρτού περιβλήματος όταν υπάρχουν στα δοσμένα σημεία κάποια που να είναι συνευθειακά:



Gift Wrapping

2 Γραμμικός Προγραμματισμός

2.1 Σύσταση Προγράμματος

Στον φάκελο linear περιλαμβάνονται τα εξής 3 αρχεία:

- constraints.txt : Το αρχείο αυτό περιλαμβάνει τον τύπο προβλήματος που καλούμαστε να λύσουμε (στη συγκεκριμένη περίπτωση max), τους περιορισμούς και το εύρος τιμών των μεταβλητών
- linear.py : Εδώ βρίσκεται το κύριο αρχείο που εκτελείται και λύνει το πρόβλημα γραμμικού προγραμματισμού
- utils.py : Στο αρχείο αυτό περιέχονται βοηθητικές συναρτήσεις για το διάβασμα των περιορισμών από το αρχείο και τη διατύπωσή τους σε ένα πρόβλημα ώστε να επιλυθεί από το κύριο αρχείο

2.2 Ροή Προγράμματος

Το πρόγραμμα αρχικά διαβάζει τους περιορισμούς από το αρχείο constraints.txt και μέσω της βασικής συνάρτησης read_data επιστρέφει το format του προβλήματος ώστε να επιλυθεί από το κυρίως πρόγραμμα. Το format αυτό αποτελείται από:

- c: Ένας κατάλογος (λίστα) με τους συντελεστές της αντικειμενικής συνάρτησης (objective function)
- constraints: Μια λίστα από λίστες που περιέχουν τους συντελεστές των περιορισμών
- numbers: Μια λίστα με τους αριθμούς που αντιστοιχούν στους περιορισμούς (δεξιά πλευρά των ανισοτήτων)
- bounds: Μια λίστα με τα όρια των μεταβλητών (χαμηλότερο και ανώτερο όριο).
- invert: Ένα boolean που δείχνει αν η αντικειμενική συνάρτηση πρέπει να αναστραφεί (maximization problem)

Έχοντας διαβάσει επιτυχώς τα δεδομένα αυτά από το αρχείο, επιστρέφονται στο κύριο πρόγραμμα, το οποίο εκτελεί τη συνάρτηση linprog της βιβλιοθήκης scipy. Η συνάρτηση αυτή χρησιμοποιείται για την επίλυση του προβλήματος γραμμικού προγραμματισμού.
Εάν η λύση του ήταν επιτυχής, την εμφανίζει. Επίσης, ελέγχει αν πρέπει να αναστραφεί η τιμή της αντικειμενικής συνάρτησης. Αν το πρόβλημα ήταν μεγιστοποίησης (οπότε οι συντελεστές της αντικειμενικής συνάρτησης είχαν αντιστραφεί στην αρχή), αναστρέφει την τιμή της αντικειμενικής συνάρτησης.
Τέλος, τυπώνει την βέλτιστη τιμή της αντικειμενικής συνάρτησης.

3 Τριγωνοποίηση Delaunay

3.1 Σύσταση Προγράμματος

Στον φάκελο delaunay περιλαμβάνονται τα εξής 3 αρχεία:

- points.txt : Το αρχείο αυτό περιλαμβάνει τα τυχαία σημεία στον χώρο με τα οποία θα εργαστούμε
- main.py : Εδώ βρίσκεται το κύριο αρχείο που εκτελείται
- utils.py : Στο αρχείο αυτό περιέχονται βοηθητικές συναρτήσεις για το διάβασμα των σημείων και την εμφάνιση των τελικών αποτελεσμάτων

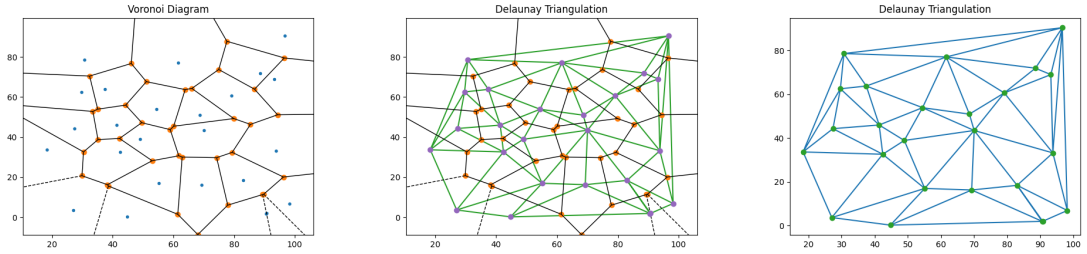
3.2 Ροή Προγράμματος

Το πρόγραμμα αρχικά ελέγχει αν το αρχείο points.txt υπάρχει και αν όχι το δημιουργεί και γράφει σε αυτό 25 τυχαία σημεία.
Στη συνέχεια, δημιουργείται το διάγραμμα Voronoi και η τριγωνοποίηση Delaunay και τέλος αυτά εμφανίζονται. Η σειρά με την οποία εμφανίζονται είναι:

- Αρχικά εμφανίζεται το διάγραμμα Voronoi για το σύνολο σημείων που δημιουργήσαμε
- Γνωρίζουμε ότι η τριγωνοποίηση Delaunay είναι το δυϊκό του διαγράμματος Voronoi και συνεπώς το δεύτερο βήμα είναι η εμφάνιση και των 2
- Στο τελευταίο σχήμα έχουμε την τριγωνοποίηση Delaunay για το σύνολο σημείων που δημιουργήσαμε

3.3 Αποτελέσματα ενδεικτικής εκτέλεσης

Ακολουθεί η εμφάνιση των 3 αυτών σταδίων για ένα ενδεικτικό σύνολο σημείων:



3.4 Πολυπλοκότητα αλγορίθμων

Ο αλγόριθμος για την τριγωνοποίηση Delaunay παρουσιάζει πολυωνυμική πολυπλοκότητα της τάξεως O(nlogn). Αυτό σημαίνει ότι ο χρόνος εκτέλεσης αυξάνεται με γραμμικό ρυθμό καθώς αυξάνεται το πλήθος των σημείων εισόδου n.
Η συμπεριφορά αυτή είναι παρόμοια με αυτή του αλγορίθμου για την κατασκευή διαγράμματος Voronoi, καθώς οι δύο δομές είναι δυϊκές και οι αλγόριθμοι μοιράζονται την ίδια πολυπλοκότητα και εκτελούνται σε περίπου ίδιο χρόνο, με τον αλγόριθμο Delaunay να εκτελείται ελαφρώς ταχύτερα.
Σε μεγάλα πλήθη σημείων, η γραμμική αύξηση του χρόνου εκτέλεσης καθίσταται πιο εμφανής, επιβεβαιώνοντας την πολυωνυμική πολυπλοκότητα των αλγορίθμων. Ωστόσο, οι αλγόριθμοι παραμένουν αποδοτικοί και εφαρμόσιμοι για πρακτικές εφαρμογές όπου η ταχύτητα και η απόδοση είναι κρίσιμες.
Συμπερασματικά, με την αύξηση του πλήθους των σημείων, ο αλγόριθμος τριγωνοποίησης Delaunay διατηρεί αποδεκτή απόδοση με γραμμική αύξηση του χρόνου εκτέλεσης, καταδεικνύοντας την αποτελεσματικότητα και την προσαρμοστικότητά του σε διαφορετικά μεγέθη δεδομένων.

4 Γεωμετρική αναζήτηση

4.1 Σύσταση Προγράμματος

Στον φάκελο geom_search περιλαμβάνονται τα εξής 5 αρχεία:

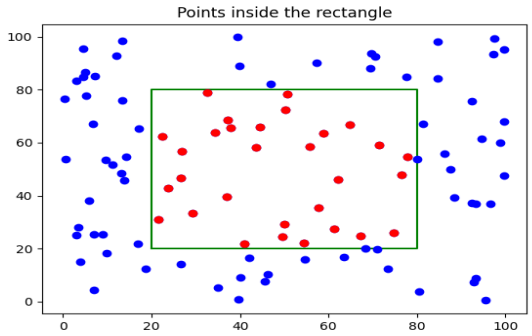
- points.txt : Το αρχείο αυτό περιλαμβάνει τα τυχαία σημεία στον χώρο με τα οποία θα εργαστούμε
- result.txt : Το αρχείο αυτό περιλαμβάνει τα σημεία τα οποία θα βρίσκονται εντός του τελικού σχήματος
- main.py : Εδώ βρίσκεται το κύριο αρχείο που εκτελείται
- kd_tree.py : Εδώ βρίσκονται οι κλάσεις Rectangle και KdTree που θα χρησιμοποιηθούν
- utils.py : Στο αρχείο αυτό περιέχονται βοηθητικές συναρτήσεις για το διάβασμα των σημείων και την εμφάνιση των τελικών αποτελεσμάτων τόσο στο σχήμα όσο και στο αρχείο result.txt

4.2 Ροή Προγράμματος

Το πρόγραμμα αρχικά δημιουργεί 100 τυχαία σημεία στο αρχείο points.txt και τα διαβάζει.
Στη συνέχεια, δημιουργείται το kd_tree με βάση αυτά τα σημεία και ένα ορθογώνιο με ορισμένες ενδεικτικές συντεταγμένες.
Έπειτα, ελέγχεται ποια από τα σημεία αυτά του kd_tree βρίσκονται εντός του ορθογωνίου και εμφανίζονται στο σχήμα που δημιουργείται. Επίσης, τα σημεία αυτά καταγράφονται και στο αρχείο result.txt.

4.3 Αποτελέσματα ενδεικτικής εκτέλεσης

Ακολουθεί η εμφάνιση του αποτελέσματος για ένα ενδεικτικό σύνολο σημείων:



4.4 Πολυπλοκότητα αλγορίθμων

- Κατασκευή: Η κατασκευή ενός kd-δέντρου με n σημεία χρειάζεται χρόνο O(nlogn). Ο αλγόριθμος είναι αναδρομικός και σε κάθε επίπεδο χωρίζει τα σημεία στη μέση.
  Το kd δέντρο με n σημεία καταλαμβάνει χώρο O(n), αφού έχουμε n διαφορετικά φύλλα, αλλά η χωρική πολυπλοκότητα κατασκευής του είναι O(nlogn), καθώς το δέντρο είναι δυαδικό και κάθε φύλλο και κόμβος χρησιμοποιεί χώρο O(1).
- Αναζήτηση: Στην ορθγώνια αναζήτηση σημείων η πολυπλοκότητα αυξάνεται όσο περισσότερα είναι τα σημεία που βρίσκονται μέσα στο ορθογώνιο. Σε μια μέση περίπτωση επισκεπτόμαστε μόνο ένα μέρος των κόμβων του δέντρου, κατά μέσο όρο $\sqrt{n}$ κόμβους. Οπότε η πολυπλοκότητα αναζήτησης είναι περίπου O($\sqrt{n}$+k), όπου k οι κόμβοι εντός του ορθογωνίου.