**ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ**

**(ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ)**

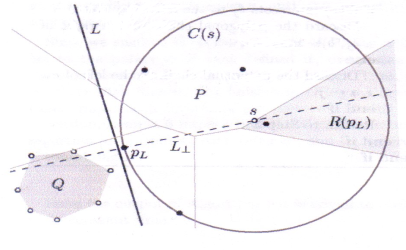
**ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ**

**ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ**

**ΕΡΩΤΗΜΑΤΑ ΔΙΑΧΩΡΙΣΜΟΥ ΚΥΚΛΟΥ ΣΕ**

**ΛΟΓΑΡΙΘΜΙΚΟ ΧΡΟΝΟ**

Χαράλαμπος Ντάλλας

****

**Ιωάννινα 20013**

**Επιβλέπων καθηγητής : Λεωνίδας Παλιός**

# ΕΥΧΑΡΙΣΤΙΕΣ

Η ολοκλήρωση αυτής της πτυχιακής υλοποιήθηκε με την υποστήριξη ενός αριθμού ανθρώπων στους οποίους θα ήθελα να εκφράσω τις θερμότερες ευχαριστίες μου. Θα ήθελα πρωτίστως να ευχαριστήσω τον καθηγητή μου Κ. Παλιό Λεωνίδα που είχε την κύρια επίβλεψη της παρούσας πτυχιακής εργασίας για το ενδιαφέρον, την υποστήριξη και τον πολύτιμο χρόνο του που μου διέθεσε. Ευχαριστώ επίσης, τους καθηγητές: Κ. Κοντογιάννη Σπυρίδων και Κ. Γεωργιάδη Λουκά, που πρόθυμα συμμετείχαν στην τριμελή επιτροπή και με βοήθησαν τόσο στη συγγραφή όσο και στην αρτιότερη παρουσίαση της διπλωματικής . Ακόμη θα ήθελα να εκφράσω τις ευχαριστίες μου σε όλους τους διδάσκοντες στο ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ ΙΩΑΝΝΙΝΩΝ για την προσφορά τους στην επιστημονική και παιδαγωγική μου κατάρτιση κατά τη διάρκεια των σπουδών μου.

# ΠΕΡΙΛΗΨΗ

Σε αυτή την πτυχιακή εργασία μελετάμε το εξής πρόβλημα: επεξεργαζόμαστε ένα σύνολο **P** με **n** σημεία στο επίπεδο ώστε να μπορέσουμε να απαντήσουμε σε ερωτήματα της ακόλουθης μορφής:

«Δοθέντος ενός κυρτού πολυγώνου **Q** με **m** κορυφές, να βρεθεί ο μικρότερος κύκλος, εάν υπάρχει, που να διαχωρίζει το σύνολο **P** από το πολύγωνο **Q**». Η δομή μας θα κατασκευάζεται σε **O(nlogn)** χρόνο και χρησιμοποιεί **O(n)** χώρο.

Προς επίλυση αυτού του προβλήματος μελετάμε τον αλγόριθμο των Alupis, Barba και Langerman τον οποίο και υλοποιούμε σε γλώσσα προγραμματισμού C++.

# ABSTRACT

In this paper we study the following problem: a set **P** of **n** points in the plane is preprocessed so that queries of the following form can be answered: Given a convex **m**-gon **Q** , report the minimum circle containing **P** and excluding **Q**. Our data structure can be constructed in **O(nlogn)** time using **O(n)** space, and answer queries in **O(logn + logm)** time. To solve this problem we study the algorithm of Alupis, Barba and Langerman. We implement the algorithm in C++.

# ΠΕΡΙΕΧΟΜΕΝΑ

ΕΥΧΑΡΙΣΤΙΕΣ……………………………………………………………………………………...2

[ΠΕΡΙΛΗΨΗ 3](#_Toc274127342)

[ABSTRACT 4](#_Toc274127343)

[ΠΕΡΙΕΧΟΜΕΝΑ 5](#_Toc274127345)

[ΕΥΡΕΤΗΡΙΟ ΣΧΗΜΑΤΩΝ 6](#_Toc274127346)

[ΕΥΡΕΤΗΡΙΟ ΠΙΝΑΚΩΝ .6](#_Toc274127347)

[ΚΕΦΑΛΑΙΟ 1: ΕΙΣΑΓΩΓΗ 8](#_Toc274127349)

1.1 [ΒΑΣΙΚΟΙ ΟΡΙΣΜΟΙ 9](#_Toc274127351)

1.2 [ΑΝΤΙΚΕΙΜΕΝΟ ΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ 14](#_Toc274127353)

1.3 ΣΧΕΤΙΚΑ ΕΡΕΥΝΗΤΙΚΑ ΑΠΟΤΕΛΕΣΜΑΤΑ…………………………………………..15

1.4 ΔΟΜΗ ΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ…………………………………………………………17

[ΚΕΦΑΛΑΙΟ 2](#_Toc274127355): [Ο ΑΛΓΟΡΙΘΜΟΣ ΤΩΝ ALUPIS, BABRBA, LANGERMAN 18](#_Toc274127356)

2.1 [ΣΥΜΒΟΛΙΣΜΟΣ 19](#_Toc274127352)

2.2 [ΘΕΩΡΗΤΙΚΟ ΥΠΟΒΑΘΡΟ 20](#_Toc274127358)

2.3 [ΠΕΡΙΓΡΑΦΗ ΤΟΥ ΑΛΓΟΡΙΘΜΟΥ 22](#_Toc274127359)

2.3.1 [ΠΡΟΕΠΕΞΕΡΓΑΣΙΑ 22](#_Toc274127360)

2.3.2 [ΕΡΩΤΗΤΣΗ 22](#_Toc274127361)

2.4 [ΠΑΡΑΔΕΙΓΜΑΤΑ ΕΦΑΡΜΟΓΗΣ ΤΟΥ ΑΛΓΟΡΙΘΜΟΥ 27](#_Toc274127361)

[ΚΕΦΑΛΑΙΟ 3: ΥΛΟΠΟΙΗΣΗ](#_Toc274127363)..............................................................................................36

3.1 [ΕΙΣΟΔΟΣ ΕΞΟΔΟΣ 3](#_Toc274127366)

3.2 [ΚΑΛΑΣΕΙΣ 3](#_Toc274127367)

3.3 ΕΠΙΛΕΓΜΕΝΕΣ ΣΥΝΑΡΤΗΣΕΙΣ……….……………………………………..…………..3

3.4 ΕΝΔΕΙΚΤΙΚΕΣ ΕΚΤΕΛΕΣΕΙΣ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ ……………………………….3

[ΚΕΦΑΛΑΙΟ 4 : ΣΥΜΠΕΡΑΣΜΑΤΑ -ΕΠΕΚΤΑΣΕΙΣ 3](#_Toc274127369)

4.1 [ΣΥΜΠΕΡΑΣΜΑΤΑ 3](#_Toc274127372)

4.2 [ΕΠΕΚΤΑΣΕΙΣ 3](#_Toc274127373)

[ΒΙΒΛΙΟΓΡΑΦΙΑ 3](#_Toc274127377)

[ΟΔΗΓΟΣ ΧΡΗΣΗΣ ΛΟΓΙΣΜΙΚΟΥ 3](#_Toc274127379)

# Ευρετήριο σχημάτων

Σχήμα 1.1 "Απλό ή μη απλό πολύγωνο"..........................................................................................9

Σχήμα 1.2 "Κυρτότητα πολυγώνου"…………………………………………………………………………………………….10

Σχήμα 1.3 "Διάγραμμα Voronoi 20 σημείων"..................................................................................11 Σχήμα 1.4 "Η Delaunay διαίρεση σε τρίγωνα για τους κόμβου του Σχήματος 1.2".………...............13 Σχήμα 2.1 "Κύκλος και διάγραμμα Voronoi "..................................................................................21 Σχημα 2.2 "Διαχωριστική γραμμή L μεταξύ του συνόλου P και του πολυγώνου Q και σημείο διαχωρισμού s"................................................................................................................................23 Σχήμα 2.3 "Εφαπτόμενες L και L’ μεταξύ συνόλου P και πολυγώνου Q και η πολυγωνική αλυσίδα qq’" ..................................................................................................................................................24 Σχήμα 2.4 "Τασημεία του συνόλου P"............................................................................................28 Σχήμα 2.5 "Τασημεία του συνόλου P και του πολυγώνου Q"........................................................29 Σχήμα 2.6 "Το κυρτό περίβλημα των σημείων του συνόλου Pκαι του πολυγώνου Q"...................30 Σχήμα 2.7 "Delauny τριγωνοποίηση των σημείων του συνόλου P".........................................32 Σχήμα 2.8 "Τα περίκεντρα των πολυγώνων απο την τριγωνοποίηση Delauny και το διάγραμμα furthest neighbor Voronoi"........................................................................................33 Σχήμα 2.9 "Εσωτερικές εφαπτομένες ε1 και ε2 μεταξύ των δύο κυρτών σχημάτων μας" .....34

# Ευρετήριο πινάκων

Πίνακας 2.1 "Πίνακας σημείων" 3

Πίνακας 2.2 "Πίνακας σημείων πολυγώνου" 3

Πίνακας 2.3 "Πίνακας σημείων κυρτού περιβλήματος" 3

Πίνακας 2.4 " Πίνακας σημείων του κυρτού περιβλήματος του πολυγώνου " 3

Πίνακας 2.5 "Πίνακας περικέντρων" 3

Πίνακας 2.6 "Πίνακας Τριάδων" 3

Πίνακας 2.7 "Ταξινομημένος πίνακας τριάδων " 3

Πίνακας 3.2.1 " Κώδικας κλάσης Point" 3

Πίνακας 3.2.2 " Κώδικας κλάσης Edge". 3

Πίνακας 3.2.3 " Κώδικας κλάσης VoronoiPoint" 3

Πίνακας 3.2.4 " Κώδικας κλάσης Triad" 3

Πίνακας 3.2.5 " Κώδικας κλάσης ListEdgeItem" 3

Πίνακας 3.2.6 " Κώδικας κλάσης EdgesList" 3

Πίνακας 3.2.7 " Κώδικας κλάσης FileInputData" 3

# 

Κεφάλαιο 1

### Εισαγωγή

Σε αυτό το κεφάλαιο αρχικά παρουσιάζουμε κάποιους βασικούς ορισμούς της εργασίας. Έπειτα παραθέτουμε το αντικείμενο της πτυχιακής εργασίας . Δίδουμε τον συμβολισμό για τα διάφορα αντικείμενα της εργασίας μας. Αναφερόμαστε και σε άλλα παρόμοια με την εργασία μας ερευνητικά αποτελέσματα. Για κάθε ερευνητικό αποτέλεσμα λέμε το όνομα του ερευνητή/ερευνητών, την ημερομηνία δημοσίευσης και την πολυπλοκότητα. Το κεφάλαιο ολοκληρώνεται με τη δομή της εργασίας.

## ΒΑΣΙΚΟΙ ΟΡΙΣΜΟΙ

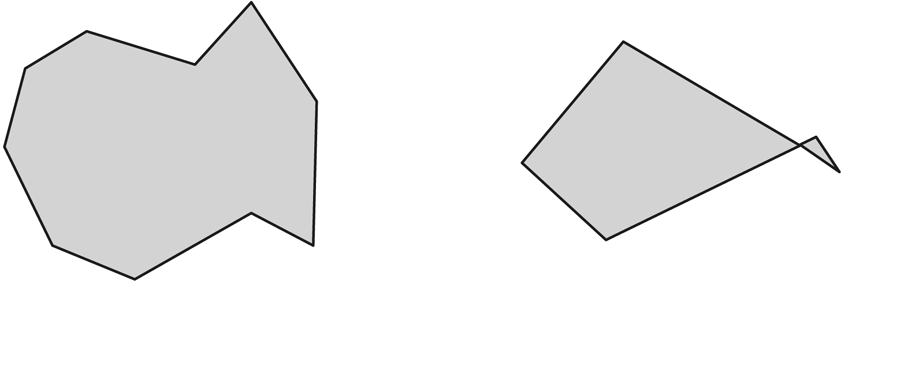
Αναφέρουμε κάποιους βασικούς ορισμούς τους οποίους χρησιμοποιούμε σε αυτή την πτυχιακή εργασία. Θα ορίσουμε το κυρτό σχήμα και στη συνέχεια το κυρτό περίβλημα ενός συνόλου σημείων και το κυρτό πολύγωνο . Στη συνέχεια θα ορίσουμε το διάγραμμα Voronoi και το furthest neighbor διάγραμμα Voronoi το οποίο προκύπτει από το σύνολο των σημείων . Ορίζουμε τον μικρότερο κύκλο διαχωρισμού μεταξύ του συνόλου P και του πολυγώνου Q και το κέντρο του.

* **Σύνολο σημείων** **P:** είναι ένα σύνολο από **n** σημεία στο χώρο. Αυτά δίνονται σαν είσοδος στο προγραμμά μας.
* **Απλό πολύγωνο (simple polygon):** .Ένα απλό πολύγωνο (*simple polygon*) είναι η περιοχή του επιπέδου που περικλείεται από ένα πεπερασμένο σύνολο ευθυγράμμων τμημάτων τα οποία σχηματίζουν μια απλή κλειστή πολυγωνική γραμμή. Έστω u1, u2, …, un είναι n σημεία στο επίπεδο και έστω τα e1 = u1u2, e2=u2u3, …, en = unu1 είναι n ευθύγραμμα τμήματα που συνδέουν τα σημεία. Τότε τα τμήματα αυτά ορίζουν ένα απλό πολύγωνο αν και μόνο αν

1. Η τομή κάθε ζεύγους διαδοχικών τμημάτων είναι το μοναδικό κοινό τους σημείο: ei ∩ ei+1 = ui+1 για κάθε i = 1, 2, …, n en ∩ e1 = u1
2. Μηδιαδοχικά τμήματα δεν τέμνονται: ei ∩ ej  ≠ Ø για κάθε 1 < i ≤ j - 1 και

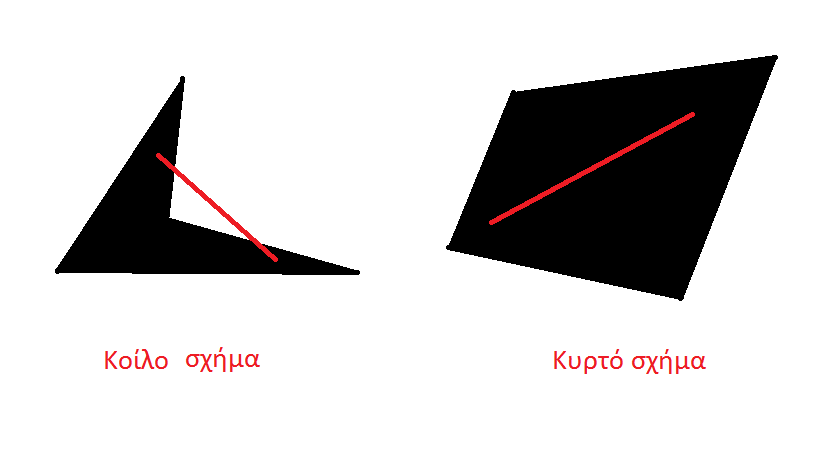
j ≤ n

Ο λόγος που τα ευθύγραμμα τμήματα σχηματίζουν μια *πολυγωνική γραμμή* είναι γιατί είναι συνδεδεμένα στη σειρά το ένα με το άλλο. Ο λόγος ποyαυτή η γραμμή είναι *κλειστή* είναι γιατί τα τμήματα κλείνουν σε ένα “κύκλο”. Ο λόγος που αυτή η κλειστή γραμμή είναι απλή είναι γιατί μη διαδοχικά τμήματα δεν τέμνονται. Τα σημεία ui ονομάζονται **κορυφές**(vertices) και τα τμήματα ei ονομάζονται **ακμές**(edges). Ένα πολύγωνο με *n* κορυφές έχει *n* ακμές. [1]



**Σχήμα 1.1:** Απλό και μη απλό πολύγωνο

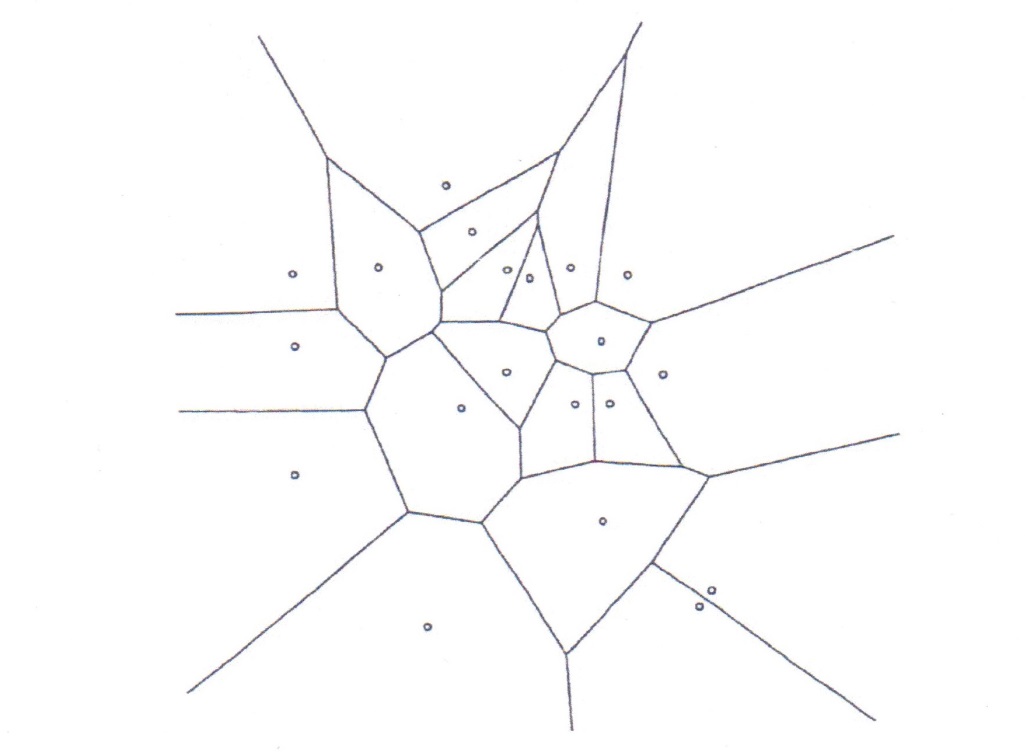
* **Κυρτό** **σχήμα (convex shape):** στη γεωμετρία ονομάζεται κάθε σχήμα το οποίο δεν διαθέτει ευθύγραμμο τμήμα το οποίο να έχει και τα δύο του άκρα μέσα στο σχήμα, και κάποια σημεία του εκτός σχήματος. Διαφορετικά, το σχήμα ονομάζεται **μη κυρτό** ή **κοίλο**. Βλέπε σχήμα 1.2.



**Σχήμα 1.2:** Κυρτότητα Πολυγώνου

* **Κυρτό πολύγωνο (convex polygon):** είναι ένα πολύγωνο το οποίο είναι κυρτό .
* **Κυρτό περίβλημα σημείων (convex hull):** είναι το κυρτό πολύγωνο με το ελάχιστο εμβαδόν το οποίο περιλαμβάνει τα δεδομένα σημεία.
* **Κύκλος Διαχωρισμού (Separating circle):** λέγετε ο κύκλος ο οποίος περιέχει όλα τα σημεία του συνόλου P.
* **Διάγραμμα Voronoi (Voronoi diagram):**

Έστω P = {p1, p2, … , pn} ένα σύνολο από σημεία στο επίπεδο. Αυτά τα σημεία λέγονται **κόμβοι** (*sites*) . Σε κάθε ένα από τους κόμβους αναθέτουμε όλα τα σημεία τοθ επιπέδου που είναι πιο κοντά σε αυτόν παρά σε οποιονδήποτε άλλο κόμβο (με βάση την ευκλείδεια απόσταση). Όλα αυτά τα σημεία σχηματίζουν την περιοχή Voronoi (*Voronoi region*) του κόμβου. Ο μαθηματικός ορισμός της περιοχής Voronoi V(pi) του κόμβου pi είναι: V(pi) = { x | j ≠ i , d(x, pi) ≤ d(x, pj) }



**Σχήμα 1.3:** Διάγραμμα Voronoi 20 σημείων

όπου με d(a, b) συμβολίζουμε την ευκλείδεια απόσταση των a, b . Σημειώνεται ότι κάθε περιοχή Voronoi είναι ένα κλειστό σύνολο, δηλ. , έχει σύνορο· το σύνορό της αποτελείται από σημεία τα οποία δεν έχουν κοντινότερο κόμβο (ή κοντινότερο γείτονα (nearest neighbor) ) αλλά ισαπέχουν από δύο ή περισσότερους κοντινότερους κόμβους. Τα σημεία αυτά για όλες τις περιοχές Voronoi σχηματίζουν το διάγραμμα Voronoi του συνόλου των κόμβων.

Το διάγραμμα Voronoi για δύο κόμβους συμπίπτει με την μεσοκάθετο *l* του ευθυγράμμου τμήματος που τα συνδέει. Η *l* είναι μία **ακμή Voronoi,** ενώ οι περιοχές Voronoi για τους 2 κόμβους είναι τα (κλειστά) ημιεπίπεδα που ορίζει η *l.* Για τρείς κόμβους, το διάγραμμα Voronoi αποτελείται από τρείς ημιευθείες (μεσοκάθετοι των αντίστοιχων ευθυγράμμων τμημάτων) που ξεκινούν από το ίδιο σημείο, το κέντρο του κύκλου που περνάει από τα τρία σημεία. Το σημείο αυτό είναι η κορυφή Voronoi. Το διάγραμμα Voronoi ενός συνόλου 20 σημείων φαίνεται στο Σχήμα 1.3 .

* **Διάγραμμα Voronoi του πιο απομακρυσμένου σημείου(Furthest site Voronoi diagram):**

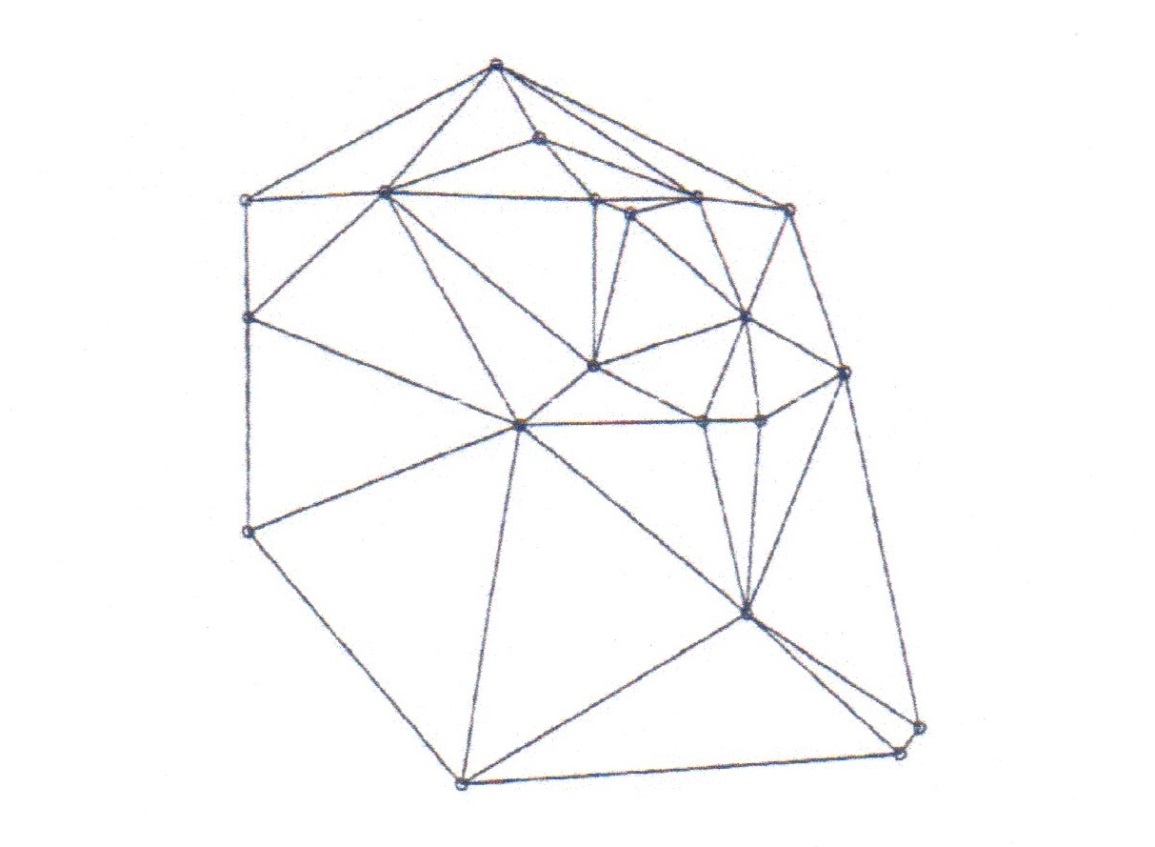
Για ένα σύνολο n σημείων το διάγραμμα Voronoi τάξης (n – 1)th ονομάζεται διάγραμμα Voronoi του πιο απομακρυσμένου σημείου.

Για ένα σύνολο σημείων S = {p1, p2, …, pn} το διάγραμμα Voronoi του πιο απομακρυσμένου σημείου χωρίζει το επίπεδο σε περιοχές στις οποίες το ίδιο σημείο του συνόλου P είναι το πιο απομακρυσμένο σημείο. Να σημειώσουμε ότι ένα σημείο του P έχει μια περιοχή στο διάγραμμα Voronoi του πιο απομακρυσμένου σημείου αν και μόνο αν είναι κορυφή του κυρτού περιβλήματος του συνόλου P. Έστω H = {h1, h2, …, hk} το κυρτό περίβλημα του συνόλου P.Ορίζουμε το διάγραμμα Voronoi του πιο απομακρυσμένου σημείου σαν μία διαίρεση του επιπέδου σε k περιοχές, μία για κάθε σημείο του H, με την ιδιότητα ότι ένα σημείο q βρίσκεται σε μια περιοχή η οποία αντιστοιχεί σε ένα σημείο hi αν και μόνο αν dist(q, hi) > dist(q, pj) για κάθε *pj* ∈ *S* με *hi* ≠ *pj*. Όπου dist(*p*, *q*) είναι η ευκλείδεια απόσταση μεταξύ δύο σημείων p και q.

* **Delaunay διαίρεση σε τρίγωνα (Delaunay triangulation):**

Έστω ότι μας δίνεται ένα σύνολο κόμβων P και έστω V(P) το αντίστοιχο διάγραμμα Voronoi. Υποθέτουμε ότι καμία τετράδα δεν ορίζει ομοκυκλικά σημεία, οπότε κάθε κορυφή Voronoi έχει βαθμό ακριβώς 3. Κατασκευάζουμε το δυικό γράφημα G του V(P): οι κορυφές του G αντιστοιχούν στις περιοχές του V(P) (δηλ. στους κόμβους του P), και οι δύο κορυφές του G συνδέονται με ακμή εάν οι αντίστοιχες περιοχές Voronoi έχουν κοινή ακμή. Παρατηρούμε ότι το G είναι ένα επίπεδο γράφημα: μπορούμε να τοποθετήσουμε κάθε κορυφή του G επάνω στον αντίστοιχο κόμβο pi του P, οπότε οι προσκείμενες ακμές παρουσιάζουν την ίδια γωνιακή ταξινόμηση όπως οι ακμές της περιοχής Voronoi V(pi) . Επιπλέον όλες οι περιοχές του G είναι τρίγωνα καθώς αντιστοιχούν στις περιοχές Voronoi οι οποίες έχουν βαθμό τρία.

Το 1934, ο Delaunay απέδειξε ότι όταν το γράφημα G σχεδιαστεί με ευθείες ακμές παράγει μια διαίρεση του κυρτού περιβλήματος του συνόλου P σε τρίγωνα που είναι γνωστή ως **Delaunay διαίρεση σε τρίγωνα**(*Delaunay triangulation*). Το Σχήμα 4.2 απεικονίζει την Delaunay διαίρεση σε τρίγωνα που αντιστοιχεί στο διάγραμμα Voronoi του Σχήματος 4.1, ενώ το Σχήμα 4.3 την επικάλυψή τους. Αξίζει να σημειωθεί ότι η Delaunay ακμή που συνδέει δύο κόμβου δεν τέμνει απαραίτητα την αντίστοιχη ακμή Voronoi, όπως φαίνεται στο Σχήμα 4.3. Επειδή το διάγραμμα Voronoi και η διαίρεση Delaunay σε τρίγωνα είναι δυικές δομές, περιέχουν την ίδια πληροφορία αλλά την αναπαριστούν σε διαφορετική μορφή. [1]



**Σχήμα 1.4:** Η Delaunay διαίρεση σε τρίγωνα για τους κόμβου του Σχήματος 1.2

## 1.2 ΑΝΤΙΚΕΙΜΕΝΟ ΤΗΣ ΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

Σε αυτή την παράγραφο θα παρουσιάσουμε το αντικείμενο με το οποίο ασχολούμαστε σε αυτή την πτυχιακή εργασία. Το πρόβλημα που θα μελετήσουμε είναι αυτό του διαχωρισμού του επιπέδου σε δύο μέρη έτσι ώστε δύο οποιαδήποτε γεωμετρικά αντικείμενα να μπορούν να διαχωριστούν με ένα σύνορο. Στη συνέχεια θα υλοποιήσουμε τον αλγόριθμο των Alupis, Barba και Langerman για το παραπάνω πρόβλημα. Παρακάτω δίνεται αναλυτικά το πρόβλημά μας.

Ο διαχωρισμός του επιπέδου προϋποθέτει την κατασκευή ενός συνόρου το οποίο θα χωρίζει το επίπεδο σε δυο μέρη έτσι ώστε δύο οποιαδήποτε γεωμετρικά αντικείμενα να μπορούν να απομονωθούν. Αυτό το σύνορο τυπικά είναι μια απλή καμπύλη όπως μια γραμμή, κύκλος ή ένα απλό πολύγωνο που σημαίνει ότι κάθε αντικείμενο του επιπέδου είναι συνδεδεμένο. Πιθανώς η πιο κλασσική περίπτωση αυτού του προβλήματος είναι να διαχωρίσουμε δύο σύνολα από σημεία με έναν κύκλο (ή με μία γραμμή που είναι ισοδύναμο με έναν απείρως μεγάλο κύκλο). Μια τέτοια γραμμή διαχωρισμού μπορεί να βρεθεί αν υπάρχει με γραμμικό προγραμματισμό. Σε αυτή την εργασία θα δείξουμε ότι ένα σύνολο P από n σημεία μπορεί να προεπεξεργαστεί σε O(nlogn) χρόνο , χρησιμοποιώντας O(n) χώρο, ώστε για κάθε πολύγωνο Q με m κορυφές που μας δίνετε να μπορούμε να βρίσκουμε τον μικρότερο κύκλο ο οποίος θα περιέχει το σύνολο P και δεν θα περιέχει το Q σε O(logn +logm) χρόνο. Ο αλγόριθμός βελτιώνει το σύνορο O(logn \* logm) που παρουσιάζεται στο [3].

**1.3 ΣΧΕΤΙΚΑ ΕΡΕΥΝΗΤΙΚΑ ΑΠΟΤΕΛΕΣΜΑΤΑ**

Σε αυτή την παράγραφο θα αναφέρουμε κάποια ερευνητικά αποτελέσματα που σχετίζονται με την πτυχιακή μας εργασία. Γενικά το πρόβλημα του διαχωρισμού του επιπέδου είναι η κατασκευή ενός συνόρου που να το χωρίζει σε δύο μέρη ώστε αν μας δοθούν δύο γεωμετρικά αντικείμενα να μπορούν αυτά να απομονωθούν. Τυπικά αυτό το σύνορο είναι μία καμπύλη όπως μία γραμμή, κύκλος ή απλό πολύγωνο. Πιθανώς η πιο κλασσική περίπτωση αυτού του προβλήματος είναι να ξεχωρίσεις δύο σύνολα από σημεία με έναν κύκλο (ή με μία γραμμή που είναι ίδιο με έναν άπειρα μεγάλο κύκλο).

Μια τέτοια διαχωριστική γραμμή μπορεί να βρεθεί, αν υπάρχει, χρησιμοποιώντας γραμμικό προγραμματισμό. Αυτό απαιτεί γραμμικό χρόνο σύμφωνα με τον αλγόριθμο του Megiddo το 1984. Δύο χρόνια αργότερα, το 1986 οι O’Rourke, Kosaraju, Megiddo δώσανε έναν γραμμικό αλγόριθμο για τον διαχωρισμό δύο συνόλων σημείων με έναν κύκλο βελτιώνοντας προηγούμενα όρια στο πρόβλημα της απόφασης. Δώσανε επίσης την ίδια χρονιά άλλους δύο αλγορίθμους. Έναν αλγόριθμο πολυπλοκότητας O(nlogn) που βρίσκει τον μέγιστο κύκλο διαχωρισμού και ο άλλος αλγόριθμος βρίσκει τον μικρότερο κύκλο διαχωρισμού σε γραμμικό χρόνο. Με αυτές τις ιδέες αργότερα το 2001 οι Boissonat , Cryzowicz, Devillers, Yvinnec έδωσαν έναν αλγόριθμο που βρίσκει τον μικρότερο κύκλο διαχωρισμού μεταξύ δύο απλών πολυγώνων σε γραμμικό χρόνο.

Μετά από 10 χρόνια το 2011 οι Augustine, Das, Maheshwari, Nandy, Roy, Sarvattomanand δείξανε πώς να προεπεξεργαστείς ένα σύνολο σημείων P, ώστε ο μέγιστος κύκλος διαχωρισμού μεταξύ του συνόλου P και ενός σημείου να μπορεί να βρεθεί σε Ο(logn) χρόνο.

Για το πρόβλημα διαχωρισμού με γραμμή , ο Edelsbrunner το 1985 έδειξε ότι ένα σύνολο σημείων P μπορεί να προεπεξεργαστεί σε O(nlogn) ώστε μία διαχωριστική γραμμή μεταξύ του P και ενός κυρτού πολυγώνου Q με m κορυφές μπορεί να υπολογιστεί σε O(logn + logm) χρόνο.

Στις 3 διαστάσεις οι Dobkin και Kirkpatrick έδειξαν ότι δύο κυρτά πολύεδρα μεγέθους n και m εδρών αντίστοιχα μπορούν να προεπεξεργαστούν σε γραμμικό χρόνο έτσι ώστε ένα επίπεδο διαχωρισμού εάν υπάρχει να μπορεί να υπολογιστεί σε O(logn \* logm) χρόνο.

## 1.4 ΔΟΜΗ ΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

Στην παράγραφο αυτή θα περιγράψουμε την δομή της πτυχιακής μας εργασίας. Δηλαδή τι περιέχει το κάθε κεφάλαιο με μια σύντομη περιγραφή.

Στο 2ο κεφάλαιο περιγράφουμε τον αλγόριθμο των Alupis, Barba, Langerman. Η περιγραφή αποτελείται από 2 μέρη, την προεπεξεργασία και την ερώτηση. Υπάρχουν και κάποια βασικά θεωρήματα, λήμματα σημειώσεις και παρατηρήσεις που αποτελούν απαραίτητο θεωρητικό υπόβαθρο για την κατανόηση της εργασίας και την περιγραφή της. Έχουμε βάλει στο τέλος του κεφαλαίου και κάποια παραδείγματα εφαρμογής του αλγορίθμου.

Στο 3ο κεφάλαιο παρουσιάζουμε την υλοποίηση του αλγορίθμου. Περιλαμβάνει δηλαδή το προγραμματιστικό μέρος της εργασίας και περιγράφει την είσοδο και την έξοδο του προγράμματος καθώς επίσης και τον κώδικα του. Περιγράφονται αναλυτικά οι κλάσεις που χρησιμοποιούνται και οι συναρτήσεις του προγράμματος με μια μικρή περιγραφή για το τι κάνουν. Έχουμε επίσης προσθέσει και τον κώδικά της κάθε συνάρτησης . Τέλος έχουμε βάλει και κάποιες ενδεικτικές εκτελέσεις του προγράμματος.

Η πτυχιακή εργασία ολοκληρώνεται με το 4ο κεφάλαιο στο οποίο παρουσιάζονται τα συμπεράσματα από την εργασία και τι θα μπορούσαμε να συμπληρώσουμε ώστε να βελτιστοποιήσουμε τα αποτελέσματα και γενικότερα τον αλγόριθμο.

Κεφάλαιο 2

**O αλγόριθμος των Alupis, Barba και Langerman**

Στο κεφάλαιο αυτό θα ασχοληθούμε με την περιγραφή του αλγορίθμου των Alupis, Barba και Langerman . Την περιγραφή του αλγορίθμου θα την θα την χωρίσουμε σε 2 μέρη. Στο πρώτο θα περιγράψουμε όλη την διαδικασία της προεπεξεργασίας .Στο δεύτερο μέρος θα παραθέσουμε την ερώτηση. Αρχικά αναφέρουμε κάποια θεωρήματα , λήμματα και κάποιες σημαντικές παρατηρήσεις και σημειώσεις που θα μας βοηθήσουν στην καλύτερη κατανόηση του αλγορίθμου. Τα θεωρήματα και τα λήμματα θα συνοδεύονται και με σχήματα ώστε να είναι όσο το δυνατόν πιο κατανοητά στον αναγνώστη. Στην τελευταία ενότητα του κεφαλαίου αυτού θα δώσουμε ένα παράδειγμα εφαρμογής του αλγορίθμου.

## 2.1 ΣΥΜΒΟΛΙΣΜΟΣ

Θα δώσουνε τον συμβολισμό των διάφορων οντοτήτων που παρουσιάζονται στην εργασία μας ώστε στη συνέχεια να αναφερόμαστε σε αυτά με τα σύμβολά τους.

* **P:** ένα σύνολο από **n** σημεία
* **Q:** πολύγωνο με m κορυφές
* **Κύκλος-P:** κύκλος που περιέχει τα σημεία του συνόλου P
* **V(P):** το furthest neighbor διάγραμμα Voronoi των σημείων του P
* **Cp:** είναι ο μικρότερος Κύκλος-P

Εάν ο Cp έχει 2 σημεία του P στην περίμετρό του τότε το κέντρο cp του κύκλου Cp βρίσκεται σε μία ακμή του V(P). Τα σημεία αυτά σχηματίζουν τη διάμετρο του Cp.

Εάν ο Cp έχει 3 σημεία του P στην περίμετρό του τότε το κέντρο cp του κύκλου Cp βρίσκεται σε μία κορυφή του V(P).

Έτσι θεωρούμε το V(P) σαν ένα δέντρο με ρίζα το cp. Για οποιοδήποτε x πάνω στο V(P) υπάρχει μοναδικό μονοπάτι πx από το cp στο σημείο x.

* **Κύκλος διαχωρισμού:** είναι ένας Κύκλος-P ο οποίος διαχωρίζει το σύνολο των σημείων P από το πολύγωνο Q.

Το κέντρο ενός κύκλου διαχωρισμού λέγεται **σημείο διαχωρισμού**

* **C\* :** είναι ο μικρότερος κύκλος διαχωρισμού και **c\*** είναι το κέντρο του

Εάν ο C\* περνάει από τουλάχιστον 2 σημεία του συνόλου P τότε το κέντρο του C\* θα βρίσκεται σε μία ακμή ή μια κορυφή του furthest neighbor διαγράμματος Voronoi V(P).

* **CH(P)** : είναι το **κυρτό περίβλημα** του συνόλου P

Υποθέτουμε ότι τα CH(P)και Q δεν τέμνονται διότι αν τέμνονταν δεν θα υπήρχε κύκλος διαχωρισμού. Επίσης αφού CH(P) και Q δεν τέμνονται τότε το Cp θα είναι αναγκαστικά και ο μικρότερος κύκλος διαχωρισμού.

Cp = C\* και cp = c\*.

* **C(y)** : ο μικρότερος Κύκλος-P με κέντρο το y και p(y) είναι η ακτίνα του

Εάν ο C(y) είναι ένας κύκλος διαχωρισμού τότε το κέντρο του y είναι ένα σημείο διαχωρισμού.

## 2.2 ΘΕΩΡΗΤΙΚΟ ΥΠΟΒΑΘΡΟ

Σε αυτή την ενότητα παραθέτουμε κάποια θεωρήματα, λήμματα, σημειώσεις και παρατηρήσεις, που θα μας βοηθήσουν στην υλοποίηση της πτυχιακής εργασίας. Κάνουμε απλή αναφορά και δεν τα αποδεικνύουμε διότι είναι ήδη αποδεδειγμένα στο paper των G.Alupis, L.Barba και S. Langerman “Circle Separability Cueries in Logarithmic time”.

Σαν πρώτη παρατήρηση , κάθε κύκλος-P που περιέχεται σε έναν κύκλο διαχωρισμού είναι και αυτός ένας κύκλος διαχωρισμού.

* **Θεώρημα 1**

Έστω s ένα σημείο πάνω στο V(P). Εάν το s είναι κέντρο ενός κύκλου διαχωρισμού τότε το c\* ( κέντρο του C\* μικρότερου κύκλου διαχωρισμού ) θα βρίσκεται πάνω στο μονοπάτι πs.

* **Λήμμα 1**

Δοθέντος ενός σημείου s, αν μετακινήσουμε ένα σημείο y πάνω στο πs προς το κέντρο cp του Cp, τότε ο C(y) θα συρρικνωθεί και θα πλησιάσει το P, και θα είναι εφαπτόμενο στο P όταν το y συμπέσει με το c\*.

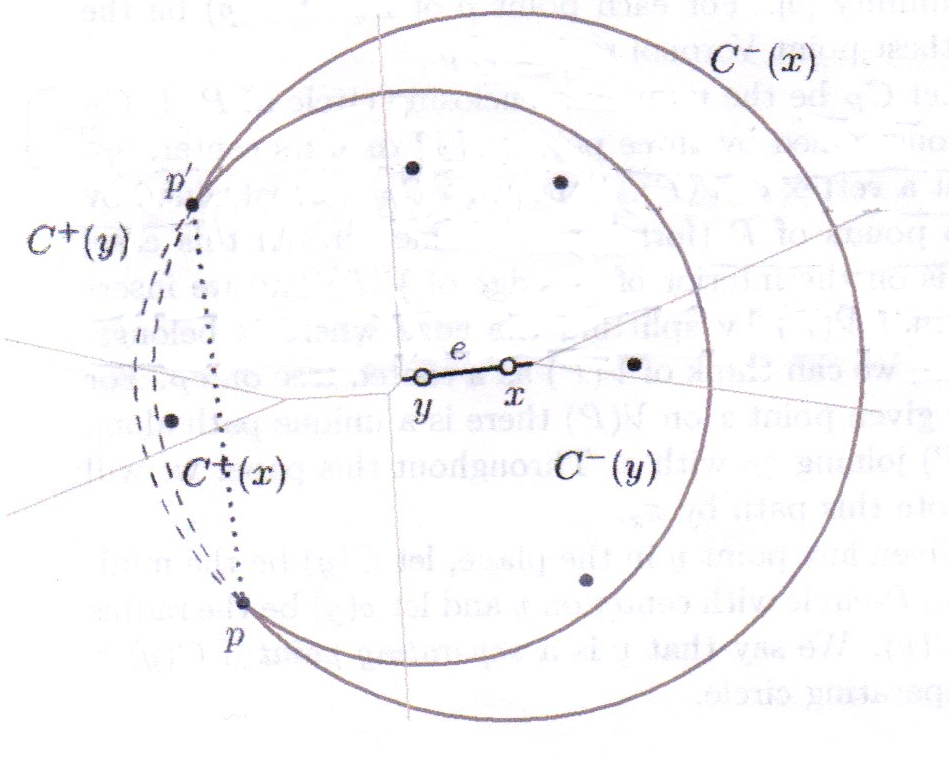
* **Σημείωση 1**

Έστω x σημείο πάνω σε ακμή e του farthest neighbor διαγράμματος Voronoi V(P), τέτοιο ώστε η e να βρίσκεται στη μεσοκάθετο του ευθύγρ. τμήματος pp’ (p, p’ ∈ P). Έστω C-(x) και C+(x) είναι οι δυο περιοχές στις οποίες χωρίζεται ο κύκλος C(x) από την ευθεία pp’ (υποθέτουμε ότι το κέντρο x είναι στο C-(x)).

* **Παρατήρηση 2**

Έστω x , y δύο σημεία πάνω στην ίδια ακμή e του διαγράμματος V(P).

Εάν ρ(x)>ρ(y), τότε C+(x) ⊂ C+(y) και C-(y) ⊂ C-(x).Βλέπε παρακάτω στο σχήμα 2.1



**Σχήμα 2.1 :** Κύκλος και διάγραμμα Voronoi

* **Λήμμα 2**

Έστω x ένα σημείο πάνω στο V(P). Η ακτίνα ρ του μικρότερου περιγεγραμμένου κύκλου αυξάνει μονότονα κατά μήκος του μονοπατιού πx ξεκινώντας από το cp .

* **Λήμμα 3**

Έστω s ένα σημείο διαχωρισμού. Εάν x είναι ένα σημείο πάνω στο πs, τότε το C(x) είναι ένας κύκλος διαχωρισμού αν και μόνο αν ρ(x)>=ρ(c\*).Επιπλέον ο C\* είναι ο μόνος κύκλος διαχωρισμού του οποίου τα όρια τέμνουν το Q.

* **Σημείωση 2**

Το C\* = C(c\*) πρέπει να είναι εφαπτόμενο στο P αλλιώς το κέντρο c\* μπορεί να πάει πιο κοντά στη ρίζα του V(P), όταν το έχουμε ως σημείο διαχωρισμού μέχρι να φτάσει το Q.

## 2.3 ΠΕΡΙΓΡΑΦΗ ΤΟΥ ΑΛΓΟΡΙΘΜΟΥ

Σε αυτή την παράγραφο θα περιγράψουμε τον αλγόριθμο των Alupis, Barba και Langerman. Ο αλγόριθμος αρχικά χρειάζεται μία προεπεξεργασία του συνόλου των σημείων P σε O(nlogn) χρόνο ώστε στη συνέχεια να προχωρήσουμε στο βασικό μέρος του αλγορίθμου που είναι η ερώτηση . Για κάθε κυρτό πολύγωνο που μας δίνεται να μπορούμε να βρίσκουμε τον μικρότερο κύκλο που περικλείει τα σημεία του P και αφήνει το πολύγωνο Q έξω από τα όρια του σε χρόνο O(logn + logm).

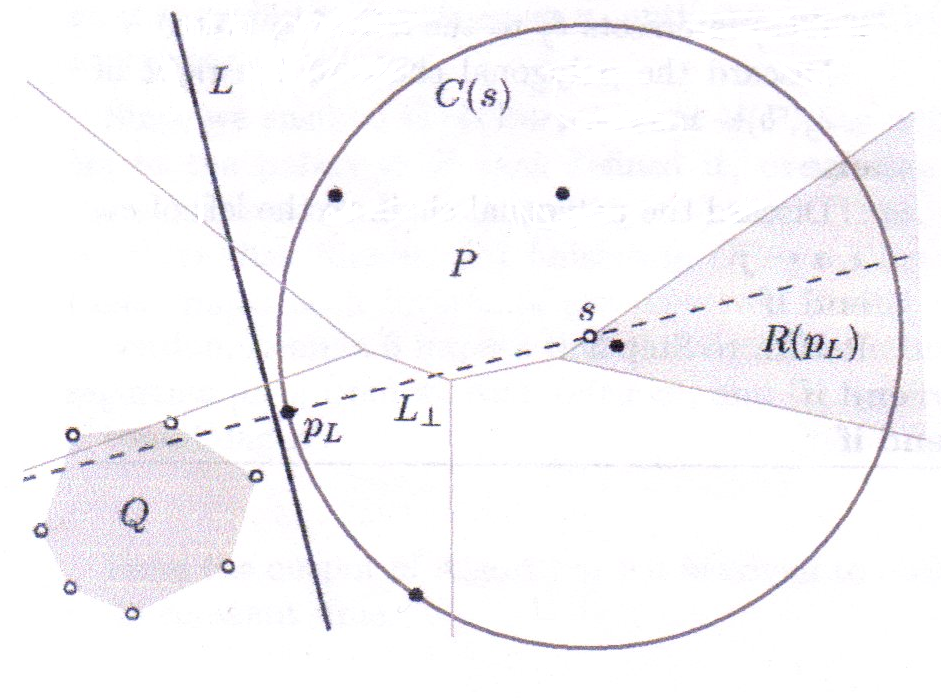
## 2.3.1 ΠΡΟΕΠΕΞΕΡΓΑΣΙΑ

Μας δίνεται αρχικά το σύνολο των σημείων P και υπολογίζουμε το διάγραμμα Voronoi του πιο απομακρυσμένου σημείου V(P) (furthest neighbor Voronoi diagram) για αυτό το σύνολο P. Το V(P) είναι ένα δυαδικό δέντρο με n φύλλα έτσι ώστε κάθε ακμή και κάθε κορυφή του διαγράμματος V(P) να έχει ένα ζεύγος δεικτών προς τα σημεία του P ορίζοντάς τα. Κάθε περιοχή Voronoi R(p) αποθηκεύεται σαν κυρτό πολύγωνο και κάθε σημείο του P έχει ένα δείκτη σε αυτό. Θέλουμε η δομή μας να υποστηρίζει δυαδική αναζήτηση σε οποιοδήποτε πιθανό μονοπάτι πs πάνω στο V(P). Στη συνέχεια ψάχνουμε το κέντρο του μικρότερου κύκλου που περικλείει τα σημεία του συνόλου P. Αυτό θα βρίσκεται είτε σε μία κορυφή του διαγράμματος furthest neighbor Voronoi αν τρία σημεία του P βρίσκονται στην περίμετρο του μικρότερου κύκλου CP. Αλλιώς αν έχει δύο σημεία του P στην περίμετρο του κύκλου CP θα βρίσκεται σε ακμή του διαγράμματος και μάλιστα θα είναι ακριβώς στο κέντρο της ακμής αυτής . Όλη αυτή η διαδικασία της προεπεξεργασίας γίνεται σε O(nlogn) χρόνο όπου n είναι το σύνολο των σημείων του P.

## 2.3.2 ΕΡΩΤΗΣΗ

Έστω ότι το κυρτό πολύγωνο Q έχει m κορυφές , τότε μπορούμε να ελέγξουμε αν ο CP είναι κύκλος διαχωρισμού. Υποθέτουμε ότι ο CP δεν είναι ο μικρότερος κύκλος διαχωρισμού. Για να υπολογίσουμε την θέση του c\* που είναι το κέντρο του μικρότερου κύκλου διαχωρισμού C\* αρχικά βρίσκουμε ένα σημείο διαχωρισμού s και στη συνέχεια ψάχνουμε για το c\* πάνω στο μονοπάτι πs χρησιμοποιώντας την δομή μας.

Για να βρούμε το s κατασκευάζουμε μία διαχωριστική γραμμή L μεταξύ του συνόλου P και του πολυγώνου Q σε O(logn + logm) χρόνο. Έστω το pL είναι το σημείο του P το οποίο βρίσκετε πιο κοντά στην γραμμή L αλλιώς μετακινούμε την L ελαφρώς. Έστω L┴ είναι κάθετη στην L και περιέχει το σημείο pL και το s θα είναι η τομή της L┴ καιτης περιοχής R(pL) όπως φαίνεται στο σχήμα 2.1.



**Σχήμα 2.2 :** Διαχωριστική γραμμή L μεταξύ του συνόλου P και του πολυγώνου Q και σημείο διαχωρισμού s

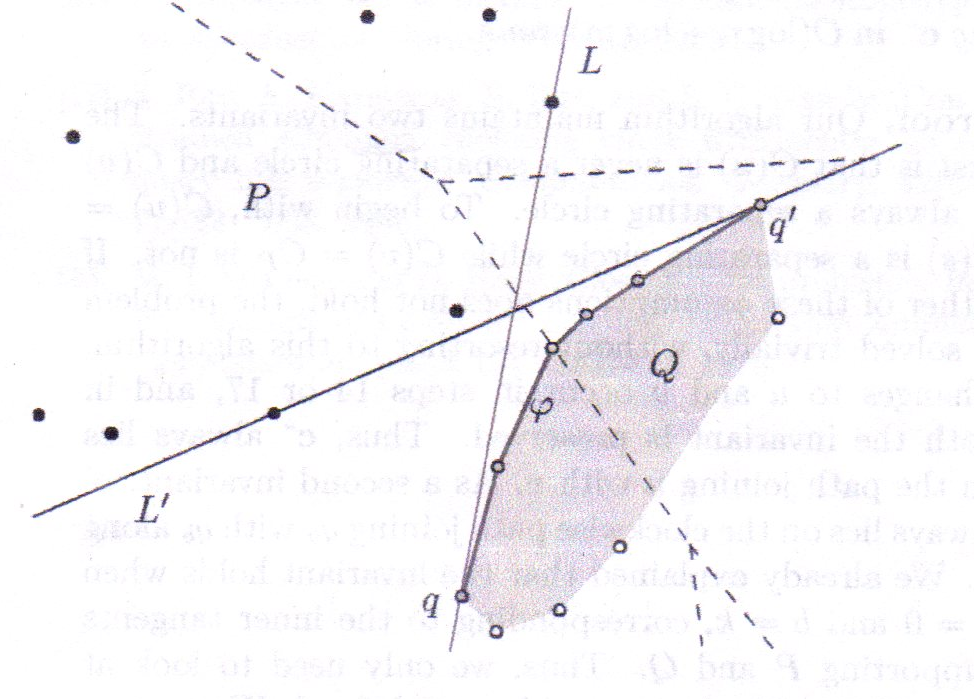
Ξέρουμε ότι η L┴ τέμνει την περιοχή R(pL) διότι το L μπορεί να θεωρηθεί σαν ένας Κύκλος-P που περιέχει μόνο το pL και με κέντρο στο άπειρο πάνω στο L┴. Επειδή το s είναι στο περίβλημα του R(pL) , αυτό σημαίνει ότι ο κύκλος C(s) με κέντρο το s περνάει από το pL. Επιπλέον ο C(s) βρίσκεται στο ίδιο ημιεπίπεδο που ορίζει η L και περιέχει το σύνολο P. Έτσι λοιπόν ο C(s) είναι ένας κύκλος διαχωρισμού.

Ας υποθέσουμε τώρα ότι το s βρίσκεται σε μία τυχαία ακμή xy του V(P) με p(x) > p(y) και έστω πs = ( u0 = s, u1 = y, …, ur = cP ) είναι το μονοπάτι μήκους r + 1 που ενώνει το s με το cP πάνω στο διάγραμμα V(P). Σύμφωνα με το θεώρημα 1 το c\* θα ανήκει στο μονοπάτι πs.

Μπορούμε πλέον να χρησιμοποιήσουμε την δομή μας και να εφαρμόσουμε δυαδική αναζήτηση στις κορυφές του μονοπατιού πs, υπολογίζοντας σε κάθε κορυφή u την ακτίνα του C(u) και την απόσταση από το Q σε χρόνο O(logm). Με αυτόν τον τρόπο μπορούμε να αποφασίσουμε αν ο C(u) είναι κύκλος διαχωρισμού. Αυτή η προσέγγιση βρίσκει το cP σε O(logn \* logm) και είναι ο αλγόριθμος που δίνεται στο [3]. Παρόλα αυτά μπορούμε να βελτιώσουμε τον αλγόριθμο χρησιμοποιώντας την κυρτότητα του πολυγώνου Q.

Για να αποφασίσουμε αν κάποιο σημείο u πάνω στο μονοπάτι πs είναι σημείο διαχωρισμού δεν χρειάζεται πάντα να υπολογίζουμε την απόσταση του u από το Q. Μπορούμε να δούμε αν ο κύκλος C(u), τέμνει την γραμμή που είναι εφαπτομένη στο Q σε χρόνο O(1). Αν όχι τότε ο C(u) είναι κύκλος διαχωρισμού και μπορούμε να προχωρήσουμε στην δυαδική αναζήτηση. Αλλιώς δοκιμάζουμε να βρούμε άλλη διαχωριστική γραμμή εφαπτόμενη στο Q που να μην τέμνει τον C(u). Το πλεονέκτημα αυτού του τρόπου είναι ότι καθώς προχωράμε στην αναζήτηση της διαχωριστικής γραμμής που είναι εφαπτόμενη στο Q ελαττώνουμε το κομμάτι του Q που θα έπρεπε να σκεφτούμε στο μέλλον. Αυτό γίνεται όπως φαίνετε παρακάτω:

Υπολογίζουμε τις δύο εσωτερικές εφαπτόμενες L και L’μεταξύ του κυρτού περιβλήματος του P και του Q σε χρόνο O(logn + logm). Οι τεχνικές για να κατασκευάσεις αυτές τις εφαπτόμενες αναφέρονται στο κεφάλαιο 4 του [12]. Έστω q και q’ είναι τα σημεία τομής των εφαπτόμενων L και L’ με το Q.Θεωρούμε μία δεξιόστροφη πολυγωνική αλυσίδα φ = [ q = q0, …, qk = q’ ] που ενώνει το q με το q’ όπως φαίνεται στο σχήμα 2.3. Να θυμίσουμε ότι φ’ είναι το σημείο τομής μεταξύ του C\* και του Q και να σημειώσουμε ότι η εφαπτόμενη γραμμή του C\* στο φ’ είναι γραμμή διαχωρισμού. Για αυτό το φ’ πρέπει να βρίσκετε σε μία ακμή του φ καθώς καμία άλλη διαχωριστική γραμμή δεν περνάει από κανένα σημείο του περιβλήματος του Q.



**Σχήμα 2.3 :** Εφαπτόμενες L και L’ μεταξύ συνόλου P και πολυγώνου Q και η πολυγωνική αλυσίδα qq’

Εάν q = q’ τότε φ’ = q τότε παρατηρούμε ότι μπορούμε να αγνοήσουμε το Q και να υπολογίσουμε τον μικρότερο κύκλο διαχωρισμού μεταξύ του P και του q. Όπως αναφέραμε προηγουμένως αυτό υπολογίζεται σε χρόνο O(logn). Υποθέτουμε από δω και πέρα ότι q ≠ q’ όπως φαίνεται στο σχήμα 2.3.

Για κάθε ακμή ei = qiqi+1 ( 0 ≤ i ≤ k -1 ) του φ, έστω li να είναι η γραμμή που επεκτείνει την ακμή. Από την κατασκευή ξέρουμε ότι κάθε γραμμή li διαχωρίζει το P με το Q. Επίσης λέμε ότι ένα σημείο x που βρίσκεται στην li  αλλά όχι στην ακμή ei θα είναι στα αριστερά του της ei αν είναι πιο κοντά στο qi ή αν θα είναι δεξιά της ei αν είναι πιο κοντά στο qi+1.

Ο αλγόριθμός μας θα εφαρμόζει ουσιαστικά δύο παράλληλες δυαδικές αναζητήσεις, η πρώτη πάνω στο μονοπάτι πs και η δεύτερη στην αλυσίδα φ, έτσι ώστε σε κάθε βήμα να απορρίπτουμε είτε ένα κομμάτι του πs είτε ένα κομμάτι της αλυσίδας φ. Καθώς ψάχνουμε στο πs κάθε φορά που βρίσκουμε έναν κύκλο διαχωρισμού μετακινούμαστε προς το cP. Όταν ο Κύκλος-P τέμνει το πολύγωνο Q μετακινούμαστε μακριά από το cP. Για να δούμε αν μία κορυφή υ είναι σημείο διαχωρισμού κοιτάμε αν ο κύκλος C(u) τέμνεται από μια γραμμή διαχωρισμού li σε συνεχή χρόνο. Εάν ο C(u) είναι κύκλος διαχωρισμού πετάμε το κομμάτι του μονοπατιού που είναι κάτω από υ στο διάγραμμα V(P). Εάν ο C(u) τέμνεται από κάνουμε έναν γρήγορο έλεγχο να δούμε αν ο C(u) τέμνει το Q βλέποντας αν η ακμή ei τέμνει τον κύκλο C(u). Εάν τέμνονται τότε το υ δεν είναι σημείο διαχωρισμού και συνεχίζουμε με την δυαδική αναζήτηση στο μονοπάτι πs. Αλλιώς η τομή του C(υ) με την li θα βρίσκεται είτε αριστερά είτε στα δεξιά του ei παρόλα αυτά δεν είμαστε σε θέση να συμπεράνουμε γρήγορα αν ο C(u) και ο Q τέμνονται. Έτσι αναστέλλουμε για λίγο την δυαδική αναζήτηση στο V(P) και επικεντρωνόμαστε στο C(u) χρησιμοποιώντας την τομή του με την li για να αποκλείσουμε το μισό φ. Ειδικά το γεγονός ότι η li τέμνει τον C(u) στη μία πλευρά του ei (δεξιά ή αριστερά) μας υποδεικνύει ότι κανένας άλλος κύκλος Κύκλος-P από δω και πέρα δεν θα τέμνει την li στην άλλη πλευρά του ei. Αυτό εμμέσως απορρίπτει το μισό φ από μια μελλοντική θεώρηση και δίνεται με περισσότερη λεπτομέρεια στο θεώρημα 7.

**Θεώρημα 7**: Ο αλγόριθμος 1 βρίσκει την ακμή του μονοπατιού πs που περιέχει το c\* σε O(logn + logm) χρόνο.

**Απόδειξη**: Ο αλγόριθμός μας έχει 2 αναλλοίωτες συνθήκες. Η πρώτη είναι ότι ο C(υ) δεν είναι ποτέ κύκλος διαχωρισμού και ότι ο C(u) είναι πάντα κύκλος διαχωρισμού. Ξεκινάμε με C(u) = C(s) είναι κύκλος διαχωρισμού όταν C(u) = CP  δεν είναι. Εάν καμία από τις υποθέσεις δεν στέκει τότε το πρόβλημα λύνετε επιπόλαια, χωρίς να χρειάζεται καταφύγουμε σε αυτόν τον αλγόριθμο. Αλλαγές στο u και στο v συμβαίνουν στα βήματα 14 ή 17 και στα δύο η σταθερά παραμένει αναλλοίωτη. Έτσι το c\* βρίσκεται πάνω στο μονοπάτι που ενώνει το v με το u. Σαν δεύτερη συνθήκη το φ’

**Algorithm 1** Δίνονται φ = [ q=q0, …, qk=q’ ] και πs = [ u0=s, u1=y, …, ur=cp ] , να βρεθεί η ακμή του πs που περιέχει το **c\***.

1: Προσδιορίζουμε το αρχικό υπομονοπάτι του πs που περιέχει το **c\***, u ← s, v ← cp

2: Προσδιορίζουμε το αρχικό διάστημα αναζήτησης στην αλυσίδα φ, a ← 0, b ← k

3: **if** u και v είναι γειτονικά στο V(P) και b = a + 1 **then**

4: Τερματίζουμε και επιστρέφουμε τα τμήματα S = [ u, v ] και H = [ qa, qb ]

5: **end if**

6: z ← FINDPOINTBETWEEN(u, v), j ← ∟ a+b/2 ˩ (κάτω φράγμα)

7: ej ← qjqj+1 και lj είναι η ευθεία που επεκτίνει την ακμή ej

8: **if** b > a + 1 **then**

9: Υπολογίζουμε p(z) και δ ← d(z, lj), Δ ← (z, ej)

10: **else**

11: Υπολογίζουμε p(z) και δ ← d(z, ej), Δ ← (z, ej)

12: **end if**

13: **if** p(z) ≤ δ, που σημαίνει οτι ο C(z) είναι κύκλος διαχωρισμού **then**

14: Προχωράμε μπροστά στο μονοπάτι πs, u ← z και επιστρέφουμε στο βήμα 3

15: **else**

16: **if** p(z) > Δ, που σημαίνει οτι ο C(z) δεν είναι κύκλος διαχωρισμού **then**

17: Προχωράμε πίσω στο μονοπάτι πs, v ← z και επιστρέφουμε στο βήμα 3

18: **else**

19: **if** C(z) τέμνει την lj στα αριστερά της ej **then**

20: Απορρίπτουμε την αλυσίδα του πολυγώνου που βρίσκεται δεξιά απο την

ej, b ← max{ j, a + 1 }

21: **else**

22: Απορρίπτουμε την αλυσίδα του πολυγώνου που βρίσκεται αριστερά απο την ej

, a ← j

23: **end if**

24: Επιστρέφουμε στο βήμα 3

25: **end if**

26: **end if**

## 2.4 ΠΑΡΑΔΕΙΓΜΑΤΑ ΕΦΑΡΜΟΓΗΣ ΤΟΥ ΑΛΓΟΡΙΘΜΟΥ

Σε αυτο το κεφάλαιο θα δώσουμε ενα παράδειγμα με το οποιο τρέχουμε το πρόγραμμά μας και θα παρουσιάσουμε τα αποτελέσματα. Το προγραμμά μας παίρνει σαν είσοδο 2 αρχεία (.txt) . Απο αυτά τα δύο αρχεία, το πρώτο θα περιέχει τις συντεταγμένες των σημείων του συνόλου P και το άλλο αρχείο θα περιέχει τις συντεταγμένες των σημείων(κορυφών) του πολυγώνου Q.

Το πρώτο αρχείο εχει την παρακάτω μορφή:

9 αριθμός σημείων στο αρχείο

-2 2 συντεταγμένες 1ου σημείου

-3 0 συντεταγμένες 2ου σημείου

-2 -2 συντεταγμένες 3ου σημείου

0 0 συντεταγμένες 4ου σημείου

0 -3 συντεταγμένες 5ου σημείου

2 -2 συντεταγμένες 6ου σημείου

3 0 συντεταγμένες 7ου σημείου

2 2 συντεταγμένες 8ου σημείου

0 3 συντεταγμένες 9ου σημείου

και το δεύτερο αρχείο είναι το:

6 αριθμός σημείων στο αρχείο

5 5 συντεταγμένες 1ου σημείου

7 3 συντεταγμένες 2ου σημείου

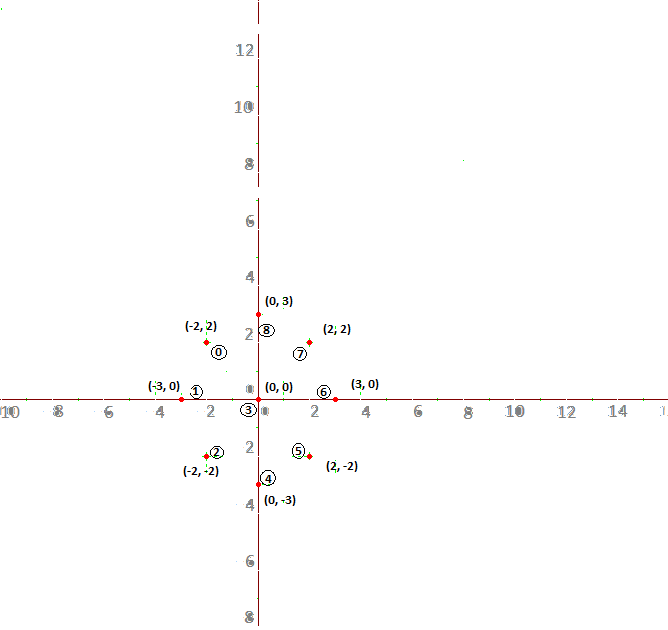
10 5 συντεταγμένες 3ου σημείου

10 10 συντεταγμένες 4ου σημείου

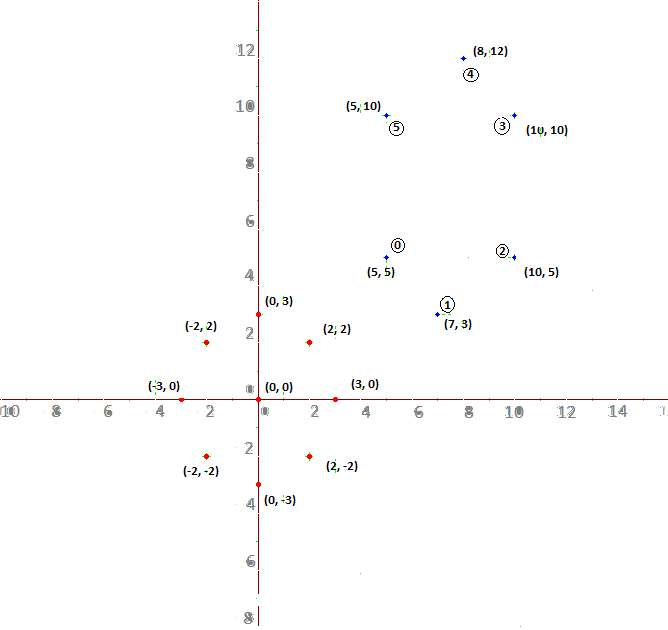
8 12 συντεταγμένες 5ου σημείου

5 10 συντεταγμένες 6ου σημείου

Στο επίπεδο τα σημεία αυτά φένονται στα σχήματα 2.5 και 2.6 .

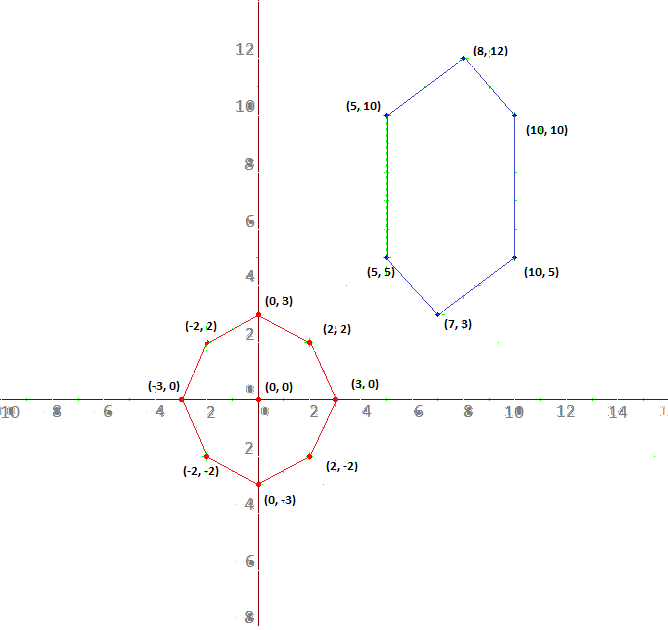


**Σχήμα 2.4 :** Τασημεία του συνόλου P (με κοκκίνο χρώμα)



**Σχήμα 2.5 :** Τασημεία του συνόλου P (με κόκκινο χρώμα) και του πολυγώνου Q (με μπλέ χρώμα)

Εχουμε λοιπόν τα δύο σχήματα και στη συνέχεια θα βρούμε το κυρτό περίβλημα του κάθε σχήματος. Προφανώς το κυρτό περίωλημα του πολυγώνου θα είναιτο ίδιο το πολύγωνο ενώ το κυρτό περίβλημα του συνόλου P τη είναι μόνο τα εξωτερικά σημεία του περιβλήματος. Ετσι λοιπόν φτίαχνουμε τα κυρτά περιβλήματα των δύο σχημάτων μας και αυτά φαίνονται στο σχήμα 2.7 .

 **Σχήμα 2.6 :** Το κυρτό περίβλημα των σημείων του συνόλου P(με κόκκινο χρώμα) και του πολυγώνου Q(με μπλέ χρώμα)

Στη συνέχεια απο τα σημεία μας θα σχεδιάσουμε το διάγραμμα furthest neighbor Voronoi. Με μια συγκεκριμένη εντολή του Qhull φτιάχνουμε ένα αρχείο OFF File που έχει την μορφή:

3

9 5 8

-2 2 5.333333333333333

-3 0 6

-2 -2 5.333333333333333

0 0 0

0 -3 6

2 -2 5.333333333333333

3 0 6

2 2 5.333333333333333

0 3 6

4 4 1 8 6

3 0 8 1

3 8 7 6

3 4 2 1

3 5 4 6

Στα OFF files τo 3 δείχνει τη διάσταση, το 9 δείχνει τον αριθμό των σημείων του συνόλου P, το 5 δείχνει τον αριθμό των πολυγώνων που δημιουργούνται από τα σημεία, το 8 δείχνει τον αριθμό των ακμών που δημιουργούνται. Τα x = -2, y = 2, z = 5,3 είναι συντεταγμένες των σημείων μας στις 3 διαστάσεις. Στο παραδειγμά μας ενδιαφερόμαστε μόνο για τις 2 διαστάσεις οπότε αγνωούμε την τρίτη δίασταση. Και τέλος τo πρώτο 4 δείχνει τον αριθμό των κορυφών του 1ου πολυγώνου που δημιουργείται. Και τα α1= 4, α2 = 1, α3 = 8 , α4 = 6 είναι ο αύξων αριθμός των σημείων μας. Το πρώτο πολύγωνο για παράδειγμα έχει 4 κορυφές οι οποίες είναι

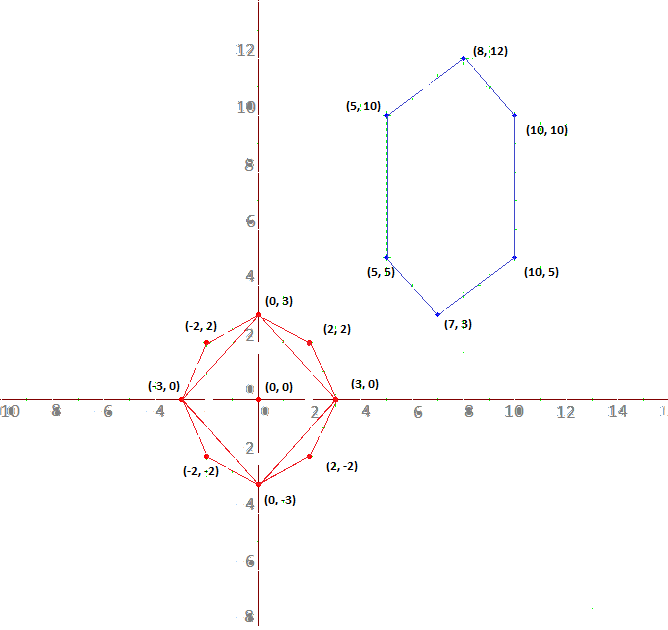
4 = (0, -3)

1 = (-3, 0)

8 = (0, 3)

6 = (3, 0)

Ετσι δημιουργούνται τα πολύγωνά μας με την Delauny τριγωνοποίηση. Η τριγωνοποίηση φαίνεται στο σχήμα 2.8 .



**Σχήμα 2.7 :** Delauny τριγωνοποίηση των σημείων του συνόλου P

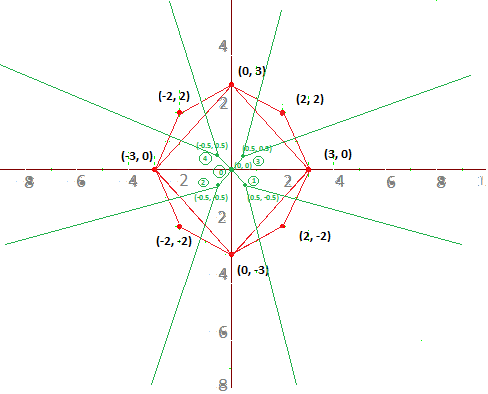
Εφόσον έχουμε τα πολυγωνα απο την Delauny τριγωνοποίηση το επόμενο βήμα είναι να βρούμε τα περίκεντρα των πολυγώνων μας. Τα περίκεντρα είναι τα κέντρα των κύκλων που έχουν στην περιφέριά τους όλες τις κορφές του πολυγώνου. Συγκέκριμένα αυτό ισχύει στα τρίγωνα οπου και οι τρείς κορυφές βρίσκονται πάντα στην περιφέρεια του κύκλου ο οποίος έχει κέντρο το περίκεντρο του τριγώνου. Για κάθε περίκεντρο κρατάμε και την ακτίνα του περιγεγραμμένου κύκλου. Τα περίκεντρα τα αποθηκεύουμε σε ένα πίνακα ο οποίος στο συγκεκριμένο παράδειγμα θα είναι:

i x y radius

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 0.0 | 0.0 | 3.0 |
| 1 | 0.5 | -0.5 | 3.5 |
| 2 | -0.5 | -0.5 | 3.5 |
| 3 | 0.5 | 0.5 | 3.5 |
| 4 | -0.5 | 0.5 | 3.5 |

**Πίνακας 3.5** : Πίνακας περικέντρων

Τα περίκεντρα φαίνονται στο σχήμα 2.9 .

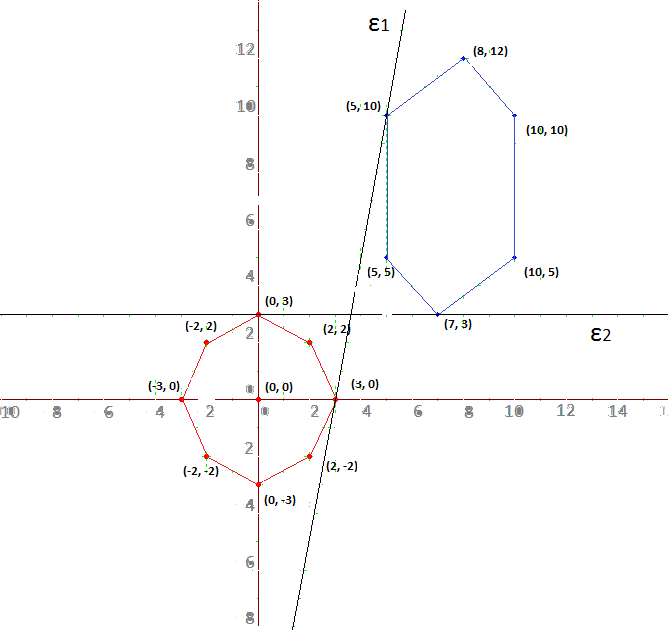


**Σχήμα 2.8 :** Τα περίκεντρα των πολυγώνων απο την τριγωνοποίηση Delauny και το διάγραμμα furthest neighbor Voronoi (με πράσινο χρώμα)

Τα περίκεντρα των πολυγώνων είναι και σημεία του διαγράμματος furthest neighbor Voronoi. Για να ολοκληρώσουμε την κατασκευή του διαγράμματος φέρνουμε απο τα περίκεντρα τις μεσοκαθέτους των πλευρών των πολυγώνων. Εμείς θέλουμε να βρούμε το κέντρο cp του μικρότερου κύκλου που περιέχει το σύνολο των σημείων P. Το cp είναι το περίκεντρο με την μικρότερη ακτίνα περιγεγραμένου κύκλου και στο παραδειγμά μας

cp = (0, 0) .

Συνεχίζοντας βρίσκουμε τις εσωτερικές εφαπτομένες των δύο κυρτών σχημάτων μας. Οι εσωτερικές εφαπτομένες ε1 και ε2 φαίνονται στο σχήμα 2.9 .



**Σχήμα 2.9 :** Εσωτερικές εφαπτομένες ε1 και ε2 μεταξύ των δύο κυρτών σχημάτων μας

Έχοντας τις εσωτερικές εφαπτομένες μπορούμε να πάρουμε απο το πολύγωνό μας την αλυσίδα qq’. Η αλυσίδα qq’ είναι οι κορυφές του πολυγώνου που είναι ενδιάμεσα απο τις εσωτερικές εφαπτομένες μας. Στο παράδειγμά μας η αλυσίδα είναι

qq’ = (5, 10) --- (5, 5) --- (7, 3)

Απομένει να βρούμε το σημείο s ώστε να μπορέσουμε να βρούμε το μονοπάτι πs πάνω στο furthest neighbor Voronoi διάγραμμα. Το σημείο s το βρίσκουμε με τον τρόπο που περιγράψαμε στην ερώτηση του αλγορίθμου. Στο παράδειγμά μας το s είναι

s = (-1.2, 0.8). Επομένως το μονοπάτι πs θα είναι

πs = (-1.2, 0.8) --- (-0.5, 0.5) --- (0, 0)

Έχοντας το **cp** κέντρο του μικρότερου κύκλου που περιέχει τα σημεία του P, το μονοπάτι πάνω στο διάγραμμα furthest neighbor Voronoi **πs** και την αλυσίδα του πολυγώνου **qq’** μπορούμε να τα βάλουμε στον αλγόριθμο των Alupis, Barba και Langerman και να μας επιστρέψει ένα αποτέλεσμα. Ο αλγόριθμος τελικά θα επιστρέψει μία ακμή η οποία θα είναι ακμή του διαγράμματος Voronoi και θα είναι η ακμή που θα περιέχει το κέντρο του μικρότερου κύκλου διαχωρισμού μεταξύ του συνόλου P και του πολυγώνου Q σε O(logn ) + O(logm) χρόνο. Η ακμή αυτή στο παραδειγμά μας είναι η (-0.5, 0.5) --- (0, 0) δηλαδή το **c\*** βρίσκεται στην ακμή (-0.5, 0.5) --- (0, 0) και συγεκριμένα είναι το σημείο **c\*** = (0, 0).

Κεφάλαιο 3

### Υλοποίηση

Στο κεφάλαιο αυτό θα ασχοληθούμε με το προγραμματιστικό μέρος της πτυχιακής εργασίας. Αναφέρουμε την γλώσσα στην οποία έχουμε προγραμματίσει τον αλγόριθμό μας. Παρουσιάζουμε την είσοδο του προγράμματός μας και την έξοδο καθώς επίσης και τον τρόπο αστικοποίησης της εξόδου. Παραθέτουμε αναλυτικά όλες τις κλάσεις που υπάρχουν στο πρόγραμμά μας με τα πεδία και τις συναρτήσεις της κάθε μίας. Το ίδιο κάνουμε και για όλες της συναρτήσεις που χρησιμοποιούμε στο πρόγραμμά μας. Για κάθε συνάρτηση αναφέρουμε τα ορίσματα που παίρνει, τι επιστρέφει και μια περιγραφή για το τι ακριβώς κάνει η συνάρτηση καθώς επίσης και τον κώδικα της .

## 3.1 ΕΙΣΟΔΟΣ – ΕΞΟΔΟΣ

Σε αυτή την παράγραφο θα αναφέρουμε την γλώσσα προγραμματισμού της εργασίας , τι παίρνει σαν είσοδο το πρόγραμμά μας και τι βγάζει σαν έξοδο. Θα αναφέρουμε επίσης και τον τρόπο οπτικοποίησης της εξόδου. Για να είναι πιο κατανοητό θα χρησιμοποιήσουμε ένα παράδειγμα εισόδου.

Το προγραμματιστικό κομμάτι της πτυχιακής εργασίας είναι υλοποιημένο σε C++.Το πρόγραμμά μας παίρνει σαν είσοδο ένα αρχείο τύπου (.txt) οποίο έχει την εξής δομή :

n

x1 y1

x2 y2

x3 y3

…

xn yn

Παράδειγμα

Ένα παράδειγμα αρχείου εισόδου με 9 σημεία

9

-2 2

-3 0

-2 -2

0 0

0 -3

2 -2

3 0

2 2

0 3

Το n είναι το σύνολο τον σημείων του συνόλου P. Τα x1, y1 είναι οι συντεταγμένες του 1ου σημείου του P, τα x2, y2 είναι οι συντεταγμένες του 2ου σημείου κ.ο.κ. μέχρι και το τελευταίο σημείο που έχει συντεταγμένες xn, yn.

Παίρνει επίσης σαν είσοδο άλλο ένα αρχείο τύπου (.txt) το οποίο έχει ακριβώς τη ίδια δομή με το προηγούμενο. Το αρχείο αυτό περιέχει της κορυφές του πολυγώνου.

Παράδειγμα Αρχείου με κορυφές ενός πολυγώνου

6

5 5

7 3

10 5

10 10

8 12

5 10

Τα αρχεία αυτά δίνονται σαν παράμετροι από τη γραμμή εντολών. Έχουμε ονομάσει το αρχείο των σημείων του P , SetOfPoints.txt και το αρχείο με τα σημεία του πολυγώνου PolygonPoints.txt .Η εντολή που δίνουμε για να τρέξει το πρόγραμμα είναι:

**./a.exe SetOfPoints.txt PolygonPoints.txt**

Αυτά τα σημεία τα αποθηκεύουμε σε πίνακες με αντικείμενα τύπου Point (το Point είναι μία κλάση που τη περιγράφουμε στην ενότητα με τις κλάσεις που ακολουθεί). Τα σημεία του συνόλου P τα αποθηκεύουμε σε ένα πίνακα με Point που τον ονομάζουμε **SetPpoints[ ]** και τα σημεία του πολυγώνου τα αποθηκεύουμε στον πίνακα **setOfPolygonPoints[ ]** ο οποίος είναι και αυτός πίνακας από Point.Τα μεγέθη αυτών των πινάκων τα παίρνουμε από το πρώτο στοιχείο του αρχείου που είναι το n το οποίο μας δίνει τον αριθμό των σημείων όπως είπαμε και παραπάνω.

Σαν έξοδο το πρόγραμμά μας αρχικά δημιουργεί τρία επιπλέων αρχεία τύπου (.txt). Αυτά τα αρχεία τα έχουμε ονομάσει PolygonConvexHull.txt, PointsConvexHull.txt και fDT\_off.txt .

Το αρχείο PointsConvexHull.txt έχει δομή:

n

p1

p2

…

pn

όπου n ο αριθμός των σημείων και p1, p2, …, pn είναι ο αύξων αριθμός των σημείων σύμφωνα με τη σειρά που τα δώσαμε.

Παράδειγμα

Παράδειγμα αρχείου PointsConvexHull.txt

8

0

1

2

4

5

6

7

8

Ίδια δομή θα έχει και το αρχείο PolygonConvexHull.txt

Παράδειγμα

Παράδειγμα αρχείου PolygonConvexHull.txt

6

0

1

2

3

4

5

Το PointsConvexHull.txt περιέχει τα σημεία που φτιάχνουν το κυρτό περίβλημα των σημείων του P με βάση τον αριθμό τους με τη σειρά που τα δώσαμε. Τα σημεία αυτά τα αποθηκεύουμε σε ένα πίνακα με Point που τον ονομάζουμε **setPConvexHullPoints[ ]**. Το PolygonConvexHull.txt περιέχει τα σημεία που φτιάχνουν το κυρτό περίβλημα του πολυγώνου με βάση τον αριθμό της σειράς που τα δώσαμε. Φτιάχνουμε έναν πίνακα από Point **polygonsConvexHullPoints[ ]** και τα αποθηκεύουμε. Τα αρχεία αυτά τα δημιουργούμε με μια εντολή του Qhull . Η εντολή είναι:

* qconvex Fx < SetOfPoints.txt > PointsConvexHull.txt

Φτιάχνει το αρχείο PointsConvexHull.txt διαβάζοντας το αρχείο που του δώσαμε σαν είσοδο SetOfPoints.txt.

* qconvex Fx < PolygonPoints.txt > PolygonConvexHull.txt

Φτιάχνει το αρχείο PolygonConvexHull.txt διαβάζοντας το αρχείο που του δώσαμε σαν είσοδο PolygonPoints.txt.

Δεν αναφερθήκαμε ακόμα στο αρχείο fDT\_off.txt που είναι και το σημαντικότερο αρχείο από αυτά που δημιουργούνται διότι από το αρχείο αυτό προκύπτει το διάγραμμα furthest neighbor Voronoi.Το αρχείο fDT\_off.txt δημιουργείται από την εντολή:

* qdelaunay Qu o < SetOfPoints.txt > fDT\_off.txt

Το αρχείο fDT\_off.txt έχει την παρακάτω δομή:

d

n m k

x1 y1 z1

x2 y2 z2

… … …

xn yn zn

a a1 a2 a3 a4

b b1 b2 b3

……………

z z1 z2 z3

Παράδειγμα αρχείου fDT\_off.txt

Αυτό θα είναι το αρχείο fDt\_off.txt που θα δημιουργηθεί από την είσοδο που δώσαμε σαν παράδειγμα.

3

9 5 8

-2 2 5.333333333333333

-3 0 6

-2 -2 5.333333333333333

0 0 0

0 -3 6

2 -2 5.333333333333333

3 0 6

2 2 5.333333333333333

0 3 6

4 4 1 8 6

3 0 8 1

3 8 7 6

3 4 2 1

3 5 4 6

To d = 3 δείχνει τη διάσταση. Το n = 9 δείχνει τον αριθμό των σημείων του συνόλου P. Το m = 5 δείχνει τον αριθμό των πολυγώνων που δημιουργούνται από τα σημεία. Το k = 8 δείχνει τον αριθμό των ακμών που δημιουργούνται. Τα x1 = -2, y1 = 2, z1 = 5,3 είναι συντεταγμένες των σημείων μας στις 3 διαστάσεις. Και τέλος τo a = 4 δείχνει τον αριθμό των κορυφών του 1ου πολυγώνου που δημιουργείται. Και τα α1= 4, α2 = 1, α3 = 8 , α4 = 6 είναι ο αύξων αριθμός των σημείων μας. Το πολύγωνο a για παράδειγμα έχει a = 4 κορυφές οι οποίες είναι

4 = (0, -3, 6)

1 = (-3, 0, 6)

8 = (0, 3, 6)

6 = (3, 0, 6)

Ο αριθμός των πολυγώνων θυμίζουμε δίνεται από το m.

Διαβάζουμε το αρχείο και για κάθε πολύγωνο βρίσκουμε το περίκεντρό του και το αποθηκεύουμε σε έναν πίνακα από Point τον οποίο ονομάζουμε **circumCentersArray [ ]**. Ο πίνακας αυτός έχει μέγεθος m = 5 δηλαδή όσα είναι και τα πολύγωνα.

Φτιάχνουμε στη συνέχεια και έναν πίνακα με Triad (η κλάση Triad περιγράφετε στην επόμενη ενότητα) στον οποίο αποθηκεύουμε τις ακμές των πολυγώνων με τα περίκεντρα τους. Ένα Triad(τριάδα) έχει 3 στοιχεία : το πρώτο είναι ο αύξων αριθμός του 1ου σημείου του πολυγώνου και το παίρνουμε όταν διαβάζουμε από το αρχείο το στοιχείο a1 που είναι το 1ο σημείο του πολυγώνου a. Το δεύτερο στοιχείο του Triad είναι ο αύξων αριθμός του 2ου σημείου του πολυγώνου a δηλαδή το a2 και τα συγκρίνουμε(a1<a2) και βάζουμε στο 1ο στοιχείο του Triad το μικρότερο από τα 2.Ετσι αυτά τα δύο σημεία συμβολίζουν μια ακμή. Στο 3ο στοιχείο του Triad βάζουμε περίκεντρο του πολυγώνου. Αν από το περίκεντρο φέρουμε την μεσοκάθετο στην ακμή τότε θα πάρουμε μία ακμή η οποία θα είναι και ακμή του furthest neighbor διαγράμματος Voronoi. Τον πίνακα με τις τριάδες αυτές τον ονομάζουμε triadsArray [ ] και έχει μέγεθος που δίνεται από τον τύπο

Μέγεθος πίνακα triadsArray [ ] = n + (2 \* m) – 3

Μέγεθος πίνακα triadsArray [ ] = 9 + (2 \* 5) – 3 = 16

Στη συνέχεια ταξινομούμε τον πίνακα triadsArray [ ] με ταξινόμηση Merge Sort και τον αποθηκεύουμε σε έναν καινούριο πίνακα sortedTriasArry [ ] ο οποίος είναι ταξινομημένος ως προς το 1ο στοιχείο του κάθε Triad και αν αυτά είναι ίδια τότε ως προς το 2ο στοιχείο του Triad πάντα με αύξουσα διάταξη.

Παρακάτω δίνουμε τους πίνακες που προκύπτουν από το παράδειγμά μας.

**SetpPoints [ 9 ]**

i x y

|  |  |  |
| --- | --- | --- |
| 0 | -2 | 2 |
| 1 | -3 | 0 |
| 2 | -2 | -2 |
| 3 | 0 | 0 |
| 4 | 0 | -3 |
| 5 | 2 | -2 |
| 6 | 3 | 0 |
| 7 | 2 | 2 |
| 8 | 0 | 3 |
|  | | |

**Πίνακας 3.1** : Πίνακας σημείων

**SetOfPolygonPoints [ 6 ]**

i x y

|  |  |  |
| --- | --- | --- |
| 0 | 5 | 5 |
| 1 | 7 | 3 |
| 2 | 10 | 5 |
| 3 | 10 | 10 |
| 4 | 8 | 12 |
| 5 | 5 | 10 |

**Πίνακας 3.2** : Πίνακας σημείων πολυγώνου

**setPConvexHullPoints [ 8 ]**

i x y

|  |  |  |
| --- | --- | --- |
| 0 | -2 | 2 |
| 1 | -3 | 0 |
| 2 | -2 | -2 |
| 3 | 0 | -3 |
| 4 | 2 | -2 |
| 5 | 3 | 0 |
| 6 | 2 | 2 |
| 7 | 0 | 3 |

**Πίνακας 3.3** : Πίνακας σημείων του κυρτού περιβλήματος

**polygonsConvexHullPoints [ 6 ]**

i x y

|  |  |  |
| --- | --- | --- |
| 0 | 5 | 5 |
| 1 | 7 | 3 |
| 2 | 10 | 5 |
| 3 | 10 | 10 |
| 4 | 8 | 12 |
| 5 | 5 | 10 |

**Πίνακας 3.4** : Πίνακας σημείων του κυρτού περιβλήματος του πολυγώνου

**circumCentersArray [ 5 ]**

i x y

|  |  |  |
| --- | --- | --- |
| 0 | 0.0 | 0.0 |
| 1 | 0.5 | -0.5 |
| 2 | -0.5 | -0.5 |
| 3 | 0.5 | 0.5 |
| 4 | -0.5 | 0.5 |
|  | | |

**Πίνακας 3.5** : Πίνακας περικέντρων

**triadsArray [ 16 ]**

i p1 p2 cc

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 4 | 0 |
| 1 | 1 | 8 | 0 |
| 2 | 6 | 8 | 0 |
| 3 | 4 | 6 | 0 |
| 4 | 0 | 8 | 1 |
| 5 | 1 | 8 | 1 |
| 6 | 0 | 1 | 1 |
| 7 | 7 | 8 | 2 |
| 8 | 6 | 7 | 2 |
| 9 | 6 | 8 | 2 |
| 10 | 2 | 4 | 3 |
| 11 | 1 | 2 | 3 |
| 12 | 1 | 4 | 3 |
| 13 | 4 | 5 | 4 |
| 14 | 4 | 6 | 4 |
| 15 | 5 | 6 | 4 |

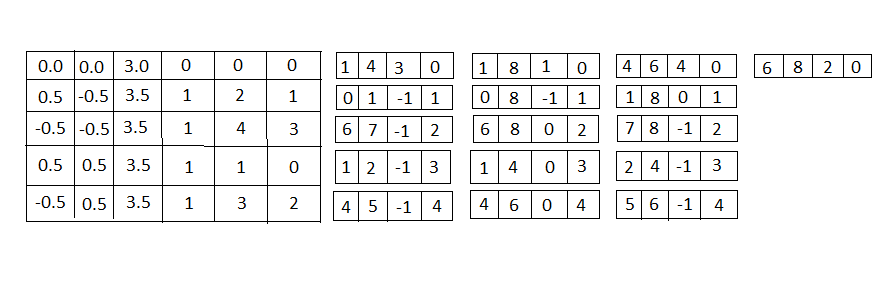
**Πίνακας 3.6** : Πίνακας τριάδων

**sortedTriadsArray [ 16 ]**

i p1 p2 cc

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 0 | 1 | 1 |
| 1 | 0 | 8 | 1 |
| 2 | 1 | 2 | 3 |
| 3 | 1 | 4 | 0 |
| 4 | 1 | 4 | 3 |
| 5 | 1 | 8 | 0 |
| 6 | 1 | 8 | 1 |
| 7 | 2 | 4 | 3 |
| 8 | 4 | 5 | 4 |
| 9 | 4 | 6 | 0 |
| 10 | 4 | 6 | 4 |
| 11 | 5 | 6 | 4 |
| 12 | 6 | 7 | 2 |
| 13 | 6 | 8 | 0 |
| 14 | 6 | 8 | 2 |
| 15 | 7 | 8 | 2 |

**Πίνακας 3.7** : Ταξινομημένος πίνακας τριάδων



**Σχήμα 3.9:** Πίνακας σημείων Voronoi

## 3.2 ΚΛΑΣΕΙΣ

Παρακάτω αναφέρονται όλες οι κλάσεις που έχουμε χρησιμοποιήσει στο πρόγραμμά μας. Για κάθε κλάση δίνουμε τα πεδία της και τις συναρτήσεις της.

* **Κλάση Point**

Την κλάση Point τη χρησιμοποιούμε για αν κρατάμε ένα σημείο με συντεταγμένες x,y. Έχει σαν πεδία τις συντεταγμένες x, y και έχει και συναρτήσεις που επιστρέφουν αυτές τις συντεταγμένες. Το point το χρειαζόμαστε για να κρατάμε τα σημεία μας σε ένα πίνακα από Points.

Πεδία:

**double x**: είναι μεταβλητή τύπου double και κρατάμε την συντεταγμένη x του σημείου μας

**double y**: είναι μεταβλητή τύπου double και κρατάμε την συντεταγμένη y του σημείου μας

Συναρτήσεις:

**Point()**: είναι ο constructor της κλάσης μας

**Point(double a, double b)**: είναι ο constructor της κλάσης μας και παίρνει σαν ορίσματα τα a,b και θέτει το x = a και y = b και φτιάχνει το σημείο point(a, b)

**getx()**: αυτή η συνάρτηση επιστρέφει το x

**gety()**: αυτή η συνάρτηση επιστρέφει το y

**print()**: αυτή η συνάρτηση εκτυπώνει το σημείο με τη μορφή ( x, y)

**move(double a, double b)**: μετακινεί το σημείο x στο a και y στο b

|  |
| --- |
| class Point{    private:  double x;  double y;    public:  Point(){  x = y = 0;  }    Point(double a, double b){  x = a;  y = b;  }  double getx(){  return x;  }  double gety(){  return y;  }    void print(){  printf("(%.1lf, %.1lf)", x, y);  }  void move(double a, double b){  x += a;  y += b;  }  }; |

**Πίνακας 3.2.1 :** Κώδικας της κλάσης Point

* **Κλάση Edge**

Είναι ένα αντικείμενο με τα indexes των 2 σημείων από τον πίνακα των σημείων και τα indexes των 2 περικέντρων . Η ακμή θα είναι είτε τα δύο περίκεντρα αν τα σημεία είναι ίδια είτε μία ημιευθεία που ξεκινάει από το περίκεντρο και πάει στο άπειρο περνώντας από το μέσο των δύο σημείων να αυτά δεν συμπίπτουν.

Πεδία:

**int firstPointIndex**: είναι μεταβλητή τύπου int και κρατάμε τον αύξων αριθμό του 1ου σημείου. Ο αύξων αριθμός είναι από τον πίνακα των σημείων του συνόλου P, και ο αύξων αριθμός ταιριάζει με τη σειρά που δίνουμε τα σημεία μας στο αρχείο SetOfPoints.txt

**int lastPointIndex**: είναι μεταβλητή τύπου int και κρατάμε τον αύξων αριθμό του τελευταίου σημείου

**int circumCenterIndex**: είναι μεταβλητή τύπου int και κρατάμε τον αύξων αριθμό του γειτονικού περικέντρου

**int currCircumCenterIndex**: είναι μεταβλητή τύπου int και κρατάμε τον αύξων αριθμό του περικέντρου

Συναρτήσεις:

**Edge()**: είναι ο constructor της κλάσης μας

**Edge(int fpi, int lpi, int cci, int ccci)**: είναι ο constructor της κλάσης μας και παίρνει σαν ορίσματα τα fpi, lpi, cci, ccci και θέτει το firstPointIndex = fpi , lastPointIndex = lpi, circumCenterIndex = cci και currCircumCenterIndex = ccci και φτιάχνει μια ακμή edge(fpi, lpi, cci, ccci)

**getFirstPointIndex()**: αυτή η συνάρτηση επιστρέφει το firstPointIndeχ

**getLastPointIndex()**: αυτή η συνάρτηση επιστρέφει το lastPointIndex

**getCircumCenterIndex()**: αυτή η συνάρτηση επιστρέφει το circumCenterIndex

**getCurrCircumCenterIndex()**: αυτή η συνάρτηση επιστρέφει το currCircumCenterIndex

**Edge \*getNeighbour()**:αυτή η συνάρτηση επιστρέφει τις γειτονικές ακμές

**void setNeighbour(Edge \*n)**: αυτή η συνάρτηση προσθέτει στη λίστα μία ακμή

**print()**: αυτή η συνάρτηση εκτυπώνει την ακμή

|  |
| --- |
| class Edge{  private:  int firstPointIndex;  int lastPointIndex;  int circumCenterIndex;  int currCircumCenterIndex;      public:    Edge(){    }    Edge(int fp, int lp, int cci, int ccci){  firstPointIndex = fp;  lastPointIndex = lp;  circumCenterIndex = cci;  currCircumCenterIndex = ccci;  }    int getFirstPointIndex(){  return firstPointIndex;  }    int getLastPointIndex(){  return lastPointIndex;  }    int getCircumCenterIndex(){  return circumCenterIndex;  }    int getCurrCircumCenterIndex(){  return currCircumCenterIndex;  }    void print(){    printf("%d\t", firstPointIndex);  printf("%d\t", lastPointIndex);  printf("%d\t",circumCenterIndex);  printf("%d\t",currCircumCenterIndex);  }    }; |

**Πίνακας 3.2.2 :** Κώδικας της κλάσης Edge

* **Κλάση PointsEdge**

Είναι ένα αντικείμενο με 2 σημεία. Ένα σημείο που το λέμε “αρχή “(source) και ένα άλλο σημείο που το λέμε “τέλος”(target). Από τα δύο αυτά σημεία φτιάχνεται το αντικείμενό μας που είναι μία ακμή.

Πεδία:

**Point source**: είναι μεταβλητή τύπου Point και συμβολίζει το ένα σημείο της ακμής. Την αρχή της ακμής.

**Point target**: είναι μεταβλητή τύπου Point και συμβολίζει το άλλο σημείο της ακμής δηλαδή το τέλος της .

Συναρτήσεις:

**PointsEdge()**: είναι ο κενός constructor της κλάσης μας

**PointsEdge(Point p1, int Point p2)**: είναι ο constructor της κλάσης μας και παίρνει σαν ορίσματα 2 Point, Point p1 και Point p2 που είναι τα δύο σημεά που ορίζουν την ακμή

**getSource()**: επιστρέφει το σημείο που δηλώνει την αρχή της ακμής

**getTarget()**: επιστρέφει το σημείο που δηλώνει το τέλος της ακμής

**print()**: αυτή η συνάρτηση εκτυπώνει την ακμή

|  |
| --- |
| class PointsEdge{    private:  Point source;  Point target;    Public:  PointsEdge(){    }  PointsEdge(Point p1, Point p2){  source = p1;  target = p2;  }  Point getSource(){  return source;  }  Point getTarget(){  return target;  }  void print(){  source.print();  printf("----->");  target.print();  printf("\n");    }    }; |

**Πίνακας 3.2.2 :** Κώδικας της κλάσης PointsEdge

* **Κλάση VoronoiPoint**

Το VoronoiPoint είναι μία κορυφή του διαγράμματος furthest neighbour Voronoi. Αυτά τα σημεία τα βρίσκουμε εύκολα καθώς αυτά είναι τα περίκεντρα των πολυγώνων που δίνονται από το off\_File που είδαμε στην προηγούμενη ενότητα. Το περίκεντρο ενός πολυγώνου είναι εκεί που ενώνονται οι μεσοκάθετες τριών οποιονδήποτε ακμών του κάθε πολυγώνου. Το σημείο VoronoiPoint έχει σαν πεδία τις συντεταγμένες του αφού είναι ένα σημείο, έχει επίσης ένα πεδίο το οποίο το λέμε radius το οποίο είναι η απόσταση του VoronoiPoint από ένα οποιοδήποτε σημείο του πολυγώνου.

Έχει και ένα πεδίο που το ονομάζουμε edgesList το οποίο είναι μία λίστα από Edges.

Πεδία:

**double x**: είναι μεταβλητή τύπου double και κρατάμε την συντεταγμένη x του σημείου μας

**double y**: είναι μεταβλητή τύπου double και κρατάμε την συντεταγμένη y του σημείου μας

**double radius**: είναι η ακτίνα του περιγεγραμμένου κύκλου του πολυγώνου που δίνεται από το off\_File

**int depth**: μεταβλητή που κρατάει το βάθος του σημείου VoronoiPoint

**int prenum**: μεταβλητή που κρατάει την θέση στην προδιατεταγμένη διάσχηση

**int postnum**: μεταβλητή που κρατάει την θέση στην μεταδιατεταγμένη διάσχηση

**Edge \* edges**: πίνακας με αντικέιμενα Edge

**Int edgesSize**: μεταβλητή που κρατάει το μέγεθος του πίνακα με τα Edge

Συναρτήσεις:

**VoronoiPoint()**: είναι ο constructor της κλάσης μας

**VoronoiPoint(double a, double b, double r)**: είναι ο constructor της κλάσης μας και παίρνει σαν ορίσματα τα a, b, r και θέτει το x= a , y = b και radius = r και φτιάχνει ένα σημείο voronoiPoint(a, b, r, el)

**getx()**: αυτή η συνάρτηση επιστρέφει το x

**gety ()**: αυτή η συνάρτηση επιστρέφει το y

**getRadius()**: αυτή η συνάρτηση επιστρέφει το το radius

**getEdgesList()**: αυτή η συνάρτηση επιστρέφει τη λίστα με τα edges

**print()**: αυτή η συνάρτηση τυπώνει το σημείο VoronoiPoint

|  |
| --- |
| class VoronoiPoint{    private:  double x;  double y;  double radius;  int depth;  int prenum;  int postnum;  Edge \*edges;  int edgesSize;      public:  VoronoiPoint(){    }  VoronoiPoint(double a, double b, double r){  x = a;  y = b;  radius = r;  depth = 0;  prenum = 0;  postnum = 0;  edgesSize = 0;  edges = (Edge \*) malloc( sizeof(Edge) );    }      double getx(){  return x;  }    double gety(){  return y;  }    double getRadius(){  return radius;  }    int getDepth(){  return depth;  }    void setDepth(int x){  depth = x;  }  int getPreNum(){  return prenum;  }  void setPreNum(int x){  prenum = x;  }  int getPostNum(){  return postnum;  }    void setPostNum(int x){  postnum = x;  }      void addEdge(Edge edge){    if(edgesSize == 0){    edges[0] = edge;  edgesSize++;    } else {    Edge prevEdge = edges[edgesSize];    edges = (Edge \*) realloc(edges, (edgesSize + 1) \* sizeof(Edge));  edges[edgesSize] = edge;    prevEdge.setNeighbour(&edge);    edgesSize++;  }  }      Edge \*getEdges(){  return edges;  }    Edge getEdge(int index){  return edges[index];  }    void print(){  printf("\nVoronoi Point\n");  printf("----------------\n");  printf("Syntetagmeni x: %.1lf\n",x);  printf("Syntetagmeni y: %.1lf\n",y);  printf(" Aktina : %.1lf\n",radius);  printf("Bathos: %d\n",depth);  printf("Pre Num: %d\n",prenum);  printf("Post Num: %d\n\n\n",postnum);  int size = getEdgesSize();  for(int i = 0; i < size; i++){  edges[i].print();  }  }    int getEdgesSize(){  return edgesSize;  }    }; |

**Πίνακας 3.2.3 :** Κώδικας της κλάσης VoronoiPoint

* **Κλάση Triad**

Είναι ένα αντικείμενο με 3 πεδία τα index 2 σημείων από τον πίνακα των σημείων και το index του αντίστοιχου περικέντρου.

Πεδία:

**int firstPointIndex**: είναι μεταβλητή τύπου int και κρατάμε τον αύξων αριθμό του 1ου σημείου μας

**int lastPointIndex**: είναι μεταβλητή τύπου int και κρατάμε τον αύξων αριθμό του τελευταίου σημείου μας

**int circumCenterIndex**: είναι μεταβλητή τύπου int και κρατάμε κρτατάμε τον αύξων αριθμό του περικέντρου

Συναρτήσεις:

**Triad()**: είναι ο constructor της κλάσης μας

**Triad(int fpi, int lpi, int cci, int ccci)**: είναι ο constructor της κλάσης μας και παίρνει σαν ορίσματα τα fpi, lpi, cci, ccci και θέτει το firstPointIndex = fpi , lastPointIndex = lpi, circumCenterIndex = cci και currCircumCenterIndex = ccci και φτιάχνει μια ακμή edge(fpi, lpi, cci, ccci)

**getFirstPointIndex()**: αυτή η συνάρτηση επιστρέφει το firstPointIndeχ

**getLastPointIndex()**: αυτή η συνάρτηση επιστρέφει το lastPointIndex

**getCircumCenterIndex()**: αυτή η συνάρτηση επιστρέφει το circumCenterIndex

**print()**: αυτή η συνάρτηση εκτυπώνει την τριάδα

|  |
| --- |
| class Triad{    private:  int firstPointIndex;  int lastPointIndex;  int circumCentreIndex;  public:  Triad(){    }    Triad(int fp, int lp, int cc){    firstPointIndex = fp;  lastPointIndex = lp;  circumCentreIndex = cc;    }    int getFirstPointIndex(){  return firstPointIndex;  }    int getLastPointIndex(){  return lastPointIndex;  }    int getCircumCentreIndex(){  return circumCentreIndex;  }    void print(){    printf(" %d %d %d\n", firstPointIndex,  lastPointIndex, circumCentreIndex);  }  }; |

**Πίνακας 3.2.4 :** Κώδικας της κλάσης Triad

* **Κλάση Circle**

Τα αντικείμενα Circle τα χρησιμοποιούμε για να δηλώσουμε έναν κύκλο με κέτρο ένα σημείο και μια ακτίνα

Πεδία:

**Point center:** είναι μεταβλητή τύπου Point και συμβολίζει το κέντρο του κύκλου

**Double radius:** μεταβλητή τύπου double που συμβολίζει την ακτίνα του κύκλου

Συναρτήσεις:

**Circle(Point c , double r)** είναι ο constructor της κλάσης μας παίρνει σαν ορίσματα ένα αντικείμενο Point που είναι το κέντρο του κύκλου μας και ένα αριθμό τύπου double που είναι η ακτίνα του κύκλου μας

**getCenter():** η συνάρτηση αυτή επιστρέφει το κέντρο του κύκλου

**getRadius** : αυτή η συνάρτηση επιστρέφει την ακτίνα του κύκλου

**print()**: αυτή η συνάρτηση εκτυπώνει τον κύκλο

|  |
| --- |
| class Circle{    private:  Point center;  double radius;  public:    Circle(Point c, double r){  center = c;  radius = r;  }  Point getCenter(){  return center;  }    double getRadius(){  return radius;  }    void print(){    double r = radius;  double x = center.getx();  double y = center.gety();    cout<<"\n";  cout << "(x - "<<x<<")^2 +  (y - "<<y<<")^2 = "<<r<<"^2";  }        }; |
|  |

**Πίνακας 3.2.5 :** Κώδικας της κλάσης ListEdgeItem

* **Κλάση Line**

Είναι μία λίστα με αντικείμενα ένα ListEdgeItem που κρατάει την αρχή και ένα ListEdgeItem που κρατάει το τέλος της λίστας. Και έχει συνάρτηση που προσθέτει αντικείμενα Edge και έτσι συμπληρώνεται η λίστα.

Πεδία:

**ListEdgeItem \*start:** Δείκτης σε ένα αντικείμενο ListEdgeItem και δηλώνει την αρχή της λίστας

**ListEdgeItem \*end:** Δείκτης σε σε ένα αντικείμενο ListEdgeItem που δηλώνει το τέλος της λίστας

Συναρτήσεις:

**EdgesList()**: είναι ο constructor της κλάσης μας

**void addEdge(Edge e)**: η συνάρτηση αυτή προσθέτει ένα Edge στη λίστα

**Edge getEdge(int index)**: αυτή η συνάρτηση επιστρέφει το Edge με το συγκεκριμένο index

**int getLength()**: αυτή η συνάρτηση επιστρέφει το μήκος της λίστας

**print()**: αυτή η συνάρτηση εκτυπώνει την λίστα

|  |
| --- |
| class Line{  private:  Point p1;  Point p2;  double a;  double b;  double c;  public:  Line(Point q1, Point q2){  p1 = q1;  p2 = q2;  a = p1.gety() - p2.gety();  b = p2.getx() - p1.getx();  c = -a\*p1.getx() - b\*p1.gety();  }    Point getPointa(){  return p1;  }    Point getPointb(){  return p2;  }    double geta(){  return a;  }    double getb(){  return b;  }    double getc(){  return c;  }    int pointOnTheLine(Point p){    double x,y,a,b,c;  x = p.getx();  y = p.gety();  a = geta();  b = getb();  c = getc();    double distance = (a\*x + b\*y + c)/ sqrt( pow(a,2) + pow(b,2) );    if( distance == 0 ){  return 1;  }  else{  return 0;  }  }    void print(){  double a,b,c;  a = geta();  b = getb();  c = getc();  cout<<"\n\n exiswsi eytheias : \n\n";  cout << "\n" << a << "x + " << b << "y + " << c << " = 0 \n";  }  }; |

**Πίνακας 3.2.6 :** Κώδικας της κλάσης EdgesList

* **Κλάση FileInputData**

Είναι ένα αντικείμενο το οποίο το χρησιμοποιούμε για να αποθηκεύσουμε την διάσταση, τον αριθμό των σημείων και έναν πίνακα με Point ο οποίος θα περιέχει τα σημεία που θα διαβάσουμε από το αρχείο..

Πεδία:

**int dimension:** ακέραιος τύπου (int) κρατάει την διάσταση

**int numOfPoints:** ακέραιος τύπου (int) κρατάει τον αριθμό των σημείων

**Point \* pointsArray:** Πίνακας με Point

Συναρτήσεις:

**FileInputData(int d, int nop, Point \* p):** είναι ο constructor της κλάσης μας

**Int getDimension ()**: η συνάρτηση αυτή επιστρέφει τη διάσταση

**Int getNumOfPoints()**: αυτή η συνάρτηση επιστρέφει το πλήθος των σημείων

**Point \* getPoints():** αυτή η συνάρτηση επιστρέφει τον πίνακα με τα σημεία

**print()**: αυτή η συνάρτηση εκτυπώνει το αντικείμενο FileInputData

|  |
| --- |
| class FileInputData{  private:  int dimension;  int numOfPoints;  Point \* pointsArray;  public:  FileInputData(int d, int nop, Point \* p){  dimension = d;  numOfPoints = nop;  pointsArray = p;  }    int getDimension(){  return dimension;  }    int getNumOfPoints(){  return numOfPoints;  }  Point \* getPoints(){  return pointsArray;  }  void print(){  printf("\n\ndimension: %d\n", dimension);  printf("number of points: %d\n", numOfPoints);  for(int i = 0; i < numOfPoints; i++){  printf("\n");  pointsArray[i].print();  }  }  }; |

**Πίνακας 3.2.7 :** Κώδικας της κλάσης FileInputData

## 3.3 ΕΠΙΛΕΓΜΕΝΕΣ ΣΥΝΑΡΤΗΣΕΙΣ

Στην παράγραφο αυτή θα αναφέρουμε όλες τις συναρτήσεις που έχουμε χρησιμοποιήσει στο πρόγραμμά μας με αναλυτικό τρόπο. Για την κάθε μια από αυτές θα λέμε τι κάνει γενικότερα και θα αναφέρουμε τα ορίσματα που παίρνει και τι επιστρέφει. Επίσης θα έχουμε και μια μικρή παράγραφο η οποία θα μας λέει πως δουλεύει η συνάρτηση. Θα δίνεται επίσης και ο κώδικας της κάθε συνάρτησης.

**3.3.1 int getCircumCentersArraySize (char \*filename):**

Αυτή η συνάρτηση διαβάζει το αρχείο filename.txt και επιστρέφει το μέγεθος του πίνακα των περικέντρων.

* **Ορίσματα**

**char \* filename:** δείκτης τύπου char που δείχνει σε ένα string το οποίο είναι το όνομα του αρχείου

* **Επιστρεφόμενη τιμή**

Η συνάρτηση επιστρέφει ένα ακέραιο αριθμό (int) που είναι το πλήθος του πίνακα των περικέντρων

|  |
| --- |
| int getCircumCentersArraySize(char \*filename){    FILE \*OffFile = fopen(filename, "r");    int dimension;  int numOfVertices;  int numOfFacets;    fscanf(OffFile, "%d", &dimension);  fscanf(OffFile, "%d", &numOfVertices);  fscanf(OffFile, "%d", &numOfFacets);  fclose(OffFile);    fclose(OffFile);  return numOfFacets;  } |

**Πίνακας 3.3.1:** Κώδικας της int getCircumCentersArraySize (char \*filename)

**3.3.2 int getVoronoiPointsArraySize (char \*filename):**

Αυτή η συνάρτηση διαβάζει το αρχείο filename.txt και επιστρέφει το μέγεθος του πίνακα των σημείων Voronoi.

* **Ορίσματα**

**char \* filename:** δείκτης τύπου char που δείχνει σε ένα string το οποίο είναι το όνομα του αρχείου

* **Επιστρεφόμενη τιμή**

Η συνάρτηση επιστρέφει ένα ακέραιο αριθμό (int) που είναι το πλήθος του πίνακα των σημείων Voronoi

|  |
| --- |
| int getVoronoiPointsArraySize(char \*filename){    FILE \*OffFile = fopen(filename, "r");    int dimension;  int numOfVertices;  int numOfFacets;    fscanf(OffFile, "%d", &dimension);  fscanf(OffFile, "%d", &numOfVertices);  fscanf(OffFile, "%d", &numOfFacets);    fclose(OffFile);  return numOfFacets;  } |

**Πίνακας 3.3.2:** Κώδικας της int getVoronoiPointsSize (char \*filename)

**3.3.3 int getTriadsArraySize (char \*filename):**

Αυτή η συνάρτηση διαβάζει το αρχείο filename.txt και επιστρέφει το μέγεθος του πίνακα των τριάδων.

* **Ορίσματα**

**char \* filename:** δείκτης τύπου char που δείχνει σε ένα string το οποίο είναι το όνομα του αρχείου

* **Επιστρεφόμενη τιμή**

Η συνάρτηση επιστρέφει ένα ακέραιο αριθμό (int) που είναι το πλήθος του πίνακα των τριάδων

|  |
| --- |
| int getTriadsArraySize(char \*filename){    FILE \*OffFile = fopen(filename, "r");    int dimension;  int numOfVertices;  int numOfFacets;    fscanf(OffFile, "%d", &dimension);  fscanf(OffFile, "%d", &numOfVertices);  fscanf(OffFile, "%d", &numOfFacets);    int TriadsArraySize = numOfVertices + (2 \* numOfFacets) - 3;    fclose(OffFile);  return TriadsArraySize;  } |

**Πίνακας 3.3.3:** Κώδικας της int getTriadsArraySize (char \*filename)

**3.3.4 int getTriadsArraySize (char \*filename):**

Αυτή η συνάρτηση παίρνει 3 σημεία Point σαν ορίσματα και επιστρέφει επίσης ένα σημείο Point το οποίο είναι το περίκεντρο του τριγώνου που δημιουργούν τα 3 σημεία.

* **Ορίσματα**

**Point a:** αντικείμενο Point και είναι το πρώτο από τα τρία σημεία του τριγώνου

**Point b:** αντικείμενο Point και είναι το δεύτερο από τα τρία σημεία του τριγώνου

**Point c:** αντικείμενο Point και είναι το τελευταίο σημείο του πολυγώνου

* **Επιστρεφόμενη τιμή**

Η συνάρτηση επιστρέφει ένα σημείο Point που είναι το περίκεντρο του τριγώνου που δημιουργούν τα 3 σημεία που δίνουμε σαν ορίσματα.

|  |
| --- |
| Point getTriangleCircumCentre(Point a, Point b, Point c){    double x1,y1,x2,y2,x3,y3,u1,u2;  x1 = a.getx();  y1 = a.gety();  x2 = b.getx();  y2 = b.gety();  x3 = c.getx();  y3 = c.gety();  double d;  d = 2\*(x1\*(y2-y3) + x2\*(y3-y1) + x3\*(y1-y2));  u1 = ( ((pow(x1,2.0) + pow(y1,2.0))\*(y2-y3) + (pow(x2,2.0) + pow(y2,2.0))\*(y3-y1) + (pow(x3,2.0) + pow(y3,2.0))\*(y1-y2) ) ) / d ;  u2 = ( ((pow(x1,2.0) + pow(y1,2.0))\*(x3-x2) + (pow(x2,2.0) + pow(y2,2.0))\*(x1-x3) + (pow(x3,2.0) + pow(y3,2.0))\*(x2-x1) ) ) / d ;  Point u(u1,u2);  return (u);  } |

**Πίνακας 3.3.4:** Κώδικας της

Point getTriangleCircumCenter (Point a, Point b, Point c)

**3.3.5 double getPointsDistance (Point p1, Point p2):**

Αυτή η συνάρτηση παίρνει 2 σημεία Point σαν ορίσματα και επιστρέφει επίσης ένα πραγματικό αριθμό double που είναι η απόσταση των 2 σημείων.

* **Ορίσματα**

**Point a:** αντικείμενο Point και είναι το πρώτο από τα 2 σημεία του που θέλουμε να βρούμε την απόσταση μεταξύ τους

**Point b:** αντικείμενο Point και είναι το δεύτερο από τα 2 σημεία των οποίων την απόσταση ψάχνουμε

* **Επιστρεφόμενη τιμή**

Η συνάρτηση επιστρέφει μια μεταβλητή τύπου double η οποία είναι η απόσταση δύο σημείων.

|  |
| --- |
| double getPointsDistance(Point p1, Point p2){    double distance;  double x1,y1,x2,y2;    x1 = p1.getx();  y1 = p1.gety();  x2 = p2.getx();  y2 = p2.gety();    distance = sqrt( pow((x2-x1),2.0) + pow((y2-y1),2.0) );    return distance;  } |

**Πίνακας 3.3.5:** Κώδικας της double getPointsDistance(Point a, Point b, Point c)

**3.3.6 FileInputData readFileInputData (char \* filename):**

Αυτή η συνάρτηση παίρνει σαν όρισμα ένα δείκτη σε πίνακα με char που είναι το όνομα του αρχείου και επιστρέφει ένα αντικείμενο FileInputData . Η συνάρτηση διαβάζει από το αρχείο filename.txt και κρατάει τα διάσταση , το πλήθος των στοιχείων και έναν πίνακα από σημεία. Όλα αυτά διαβάζονται από το αρχείο με την fscanf και αποθηκεύονται στα πεδία του FileInputData.

* **Ορίσματα**

**char \* filename:** είναι ένας δείκτης τύπου char ο οποίος δείχνει σε ένα string(πίνακα από char) που είναι το όνομα του αρχείου το οπό και θέλουμε να διαβάσουμε στην συνάρτηση αυτή

* **Επιστρεφόμενη τιμή**

Η συνάρτηση επιστρέφει ένα αντικείμενο τύπου FileInputData που όπως είδαμε στις κλάσεις κρατάει τα στοιχεία που διαβάζουμε από το αρχείο filename.txt .Δηλαδή κρατάει τη διάσταση το πλήθος των σημείων και έναν πίνακα με αυτά τα σημεία.

|  |
| --- |
| FileInputData readFileInputData(char \*filename){    FILE \*pointsFile = fopen(filename, "r");  int dimension;  int numOfPoints;    if(pointsFile == NULL){  printf("Can't open input file! \n");  exit(1);  }    fscanf(pointsFile, "%d", &dimension);  fscanf(pointsFile, "%d", &numOfPoints);    Point \* pointsArray;  pointsArray =(Point \*) malloc ( numOfPoints\*sizeof(Point) );  if(pointsArray == NULL){  printf("Error: out of bounts! \n");  exit(1);  }    if( numOfPoints < 3 ){  printf("To synolo twn shmeiwn prepei na einai >= 3");  exit(0);  }  else{  printf("Diastasi xwrou : %d\n", dimension);  printf("Arithmos simeiwn : %d \n", numOfPoints);    for(int i = 0; i < numOfPoints; i++){    double x, y;    fscanf(pointsFile, "%lf", &x);  fscanf(pointsFile, "%lf", &y);  Point point(x, y);  pointsArray[i] = point;  point.print();  cout<<"\n";  }    FileInputData inputData(dimension, numOfPoints, pointsArray);    fclose(pointsFile);  return inputData;  }  } |

**Πίνακας 3.3.6:** Κώδικας της FileInputData readFileInputData(char \* filename)

**3.3.7 Point \* readConvexHullData (char \* filename):**

Αυτή η συνάρτηση παίρνει σαν όρισμα ένα δείκτη σε πίνακα με char που είναι το όνομα του αρχείου και παίρνει και έναν δείκτη τύπου Point που δείχνει σε έναν πίνακα με σημεία. Δείκτη τύπου Point που δείχνει σε έναν πίνακα με σημεία τα οποία είναι τα σημεία του κυρτού περιβλήματος. Με λίγα λόγια η συνάρτηση διαβάζει ένα αρχείο filename.txt και από αυτό παίρνει τα σημεία του κυρτού περιβλήματος (που δίνονται με τα index του και όχι σαν κανονικά σημεία τύπου Point) και τα αποθηκεύει σε έναν πίνακα από σημεία. Όλα αυτά διαβάζονται από το αρχείο με την fscanf..

* **Ορίσματα**

**char \* filename:** είναι ένας δείκτης τύπου char ο οποίος δείχνει σε ένα string(πίνακα από char) που είναι το όνομα του αρχείου το οπό και θέλουμε να διαβάσουμε στην συνάρτηση αυτή

**Point \* pointsArray:** παίρνει επίσης σαν όρισμα έναν πίνακα με σημεία. Τα σημεία αυτά είναι τα αρχικά μας σημεία, εκείνα δηλαδή τα σημεία του συνόλου P.

* **Επιστρεφόμενη τιμή**

Η συνάρτηση επιστρέφει ένα δείκτη τύπου Point σε έναν πίνακα με Point. Επιστρέφει δηλαδή έναν πίνακα με σημεία τα οποία αυτά σημεία είναι τα σημεία του κυρτού περιβλήματος.

|  |
| --- |
| Point \* readConvexHullData(char \* filename, Point \* pointsArray){    FILE \* pointsConvexHullFile = fopen(filename, "r");  int numOfVertexes;    if(pointsConvexHullFile == NULL){  printf("Can't open input file! \n");  exit(1);  }    fscanf(pointsConvexHullFile, "%d", &numOfVertexes);  Point \* convexHullArray;  convexHullArray = (Point \*) malloc ( numOfVertexes\*sizeof(Point) );  if(convexHullArray == NULL){  printf("Error: out of bounts! \n");  exit(1);  }    printf("Arithmos koryfwn: %d\n",numOfVertexes);  printf("Ta shmeia pou dimiourgoun to kyrto perivlima einai: \n");      for(int i = 0; i < numOfVertexes; i++){    int x;  fscanf(pointsConvexHullFile, "%d", &x);  convexHullArray[i] = pointsArray[x];  printf("\n");  convexHullArray[i].print();  }    fclose(pointsConvexHullFile);  return convexHullArray;  } |

**Πίνακας 3.3.7:** Κώδικας της Point \* readConveHullData(char \* filename)

**3.3.8 Point \* readConvexHullData (char \* filename):**

Παίρνει σαν ορίσματα 3 σημεία τύπου Point και ανάλογα με το αν αυτά έχουν δεξιά ή αριστερή στροφή επιστρέφει ακέραιο αριθμό ο οποίος είναι αρνητικός αν η στροφή είναι αριστερή και θετικός αν η στροφή είναι δεξιά. Επιστρέφει 0 αν τα 3 σημεία είναι συνευθειακά. Ο τρόπος που το κάνει είναι υπολογίζοντας την ορίζουσα του πίνακα που δημιουργούν οι συντεταγμένες των σημείων στις 3 διατασεις.(για την διάσταση z βάζουμε 1 σε όλα τα σημεία για ευκολία στις πράξεις).Το πρόσημο της ορίζουσας καθορίζει την στροφή.

* **Ορίσματα**

Παίρνει σαν ορίσματα 3 Point

**Point p1:** 1ο σημείο

**Point p2:** 2ο σημείο

**Point p3:** 3ο σημείο

* **Επιστρεφόμενη τιμή**

Η συνάρτηση επιστρέφει ένα έναν ακέραιο αριθμό που ανάλογα με το πρόσημό του μας δείχνει την στροφή. ( ακέραιος > 0 - > δεξιά στροφή , ακέραιος < 0 - > αριστερή στροφή, ακέραιος = 0 - > τα σημεία είναι συνευθειακά)

|  |
| --- |
| int Turn(Point p1, Point p2, Point p3){    double x1,y1,x2,y2,x3,y3;  double det;    x1 = p1.getx();  y1 = p1.gety();  x2 = p2.getx();  y2 = p2.gety();  x3 = p3.getx();  y3 = p3.gety();    double Matrix[3][3]={x1,y1,1,x2,y2,1,x3,y3,1};    det = x1\*y2 - y1\*x2 - x1\*y3 + y1\*x3 + x2\*y3 - y3\*x2 ;    if(det > 0){  return 1;  }  else if(det < 0){  return -1;  }  else{  return 0;  }    } |

**Πίνακας 3.3.8:** Κώδικας της int Turn(Point p1, Point p2, Point p3)

## 3.4 ΕΝΔΕΙΚΤΙΚΕΣ ΕΚΤΕΛΕΣΕΙΣ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ

## 

Κεφάλαιο 4

### Συμπεράσματα - Επεκτάσεις

Στο κεφάλαιο αυτό θα παρουσιάσουμε τα συμπεράσματα της εργασίας μας που μπορεί να χρησιμοποιηθεί και θα προτείνουμε τρόπους για την βελτίωσή της.

## 4.1 ΣΥΜΠΕΡΑΣΜΑΤΑ

## 

## 4.2 ΕΠΕΚΤΑΣΕΙΣ

# ΒΙΒΛΙΟΓΡΑΦΙΑ

**[1]** Σημειώσεις στην υπολογιστική γεωμετρία του καθηγητή Λεωνίδα Παλιού (Πανεπιστήμιο Ιωαννίνων, τμήμα μηχανικών Η/Υ και πληροφορικής).

**[2]** J. Augustine, S.Das, A. Maheshwari, S. Roy και S. Sarvattomananda “ Localized geometric query problems”. CoRR, abs/1111.2918, 2011

**[3]** L. Barba και J. Urrutia . Δυναμικός διαχωρισμός κύκλου μεταξύ δύο κυρτών πολυγώνων. Στην εργασία “ *Proceedings of the Spanish Meetings in Computational Geometry* ” σελίδες 43 46, 2011

**[4]** B. K. Bhattacharya. “ Circular separability of planar point sets ”, σελίδες 25-39, 1988

**[5]** J. – D. Boissonnat , J. Cryzowicz, O. Devillers και M. Yvinec. “ Circular separability of polygons ”. Algorithmica, 30:67-82, 2001

**[6]** M. de Berg, M. van Krevelt, M. Overmars και O. Schwarzkopf ” Algorithms and Applications ”. Springer-Verlag, third edition, 2008

**[7]** D. p. Dopkin and D. G. Kirkpatrick. Fast detection of polyhedral intersection “ Theoretical Computer Sience ” 27(3): 241 – 253, 1983

**[8]** H. Edelsbrunner. Computing the extreme distances between two convex polygons. “ Journal of Algorithms ”, 6(2): 213 – 224, 1985

**[9]** C. E Kim and T. A. Anderson . Digital disks and a digital compactness measure. In STOC ’84: “ Proceedings of the sixteenth annual ACM symposium on Theory of computing ”, pages 117 – 124, New York, NY, USA, 1984. ACM.

**[10]** N. Megiddo. Linear programming in linear time when the dimension is fixed. J. ACM, 31(1): 114 – 127, Jan. 1984.

**[11]** J. o’Rourke, S. Rao Kosaraju and N. Megiddo. Compiuting circular separability. “ Discrete and Computational Geometry ”, 1: 105 – 113, 1986

**[12]** F. Preparata και M. Shamos . Υπολογιστική γεωμετρία: μία εισαγωγή. (*Computational geometry: an introduction*). Springer-Verlag 1985.

**[13]** S. Roy, A. Karmakar, S. Das and S.C. Nandy. Constrained minimum enclosing circle with center on a query line segment. “ Computational Geometry ”, 42(6-7): 632 – 638, 2009.

**[14]** M. I. shamos and D. hoey. Closest-point problems. “ Foundations of Computer Sience ”, 1975, “ 16th Annual Symposium “ on, pages 151 – 162, oct. 1975

# ΗΛΕΚΤΡΟΝΙΚΗ

**[15]** <http://www.qhull.org/html/qvoronoi.htm>

**[16]** <http://www.qhull.org/html/qconvex.htm>

**[17]** <http://www.cs.uoi.gr/~palios/comp_geom/>

# 

# ΟΔΗΓΟΣ ΧΡΗΣΗΣ ΛΟΓΙΣΜΙΚΟΥ

Όπου εφαρμόζεται.