Κάλυψη Ορθογωνίων Πολυγώνων Κλάσης-3 με Ελάχιστο Πλήθος r-Αστέρων

Η ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ ΕΞΕΙΔΙΚΕΥΣΗΣ

υποβάλλεται στην

ορισθείσα από τη Γενική Συνέλευση Ειδικής Σύνθεσης

του Τμήματος Πληροφορικής Εξεταστική Επιτροπή

από τον

Πέτρο Τζίμα

ως μέρος των υποχρεώσεων του για τη λήψη του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΣΤΗΝ ΠΛΗΡΟΦΟΡΙΚΗ

ΜΕ ΕΞΕΙΔΙΚΕΥΣΗ

ΣΤΗΝ ΘΕΩΡΙΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

Μάρτιος 2013

Αφιέρωση

Αφιερώνω αυτή την εργασία στην οικογένεια μου.

Ειδικά στη σύζυγό μου Ειρήνη.

Ευχαριστίες

Ευχαριστώ θερμά τον επιβλέποντα της μεταπτυχιακής μου εργασίας κ. Λεωνίδα Παληό, αναπληρωτή καθηγητή του Τμήματος Πληροφορικής του Πανεπιστημίου Ιωαννίνων. Η καθοδήγησή του ήταν ουσιαστικότατη για να τελειώσει αυτή η μεταπτυχιακή εργασία. Η υπομονή και η κατανόηση που έδειξε σε όλη την πορεία της εργασίας μου υπήρξε καθοριστικός παράγοντας της επιτυχούς έκβασής της.

Ευχαριστώ επίσης τον κ. Σταύρο Νικολόπουλο, καθηγητή του Τμήματος Πληροφορικής του Πανεπιστημίου Ιωαννίνων και τον κ. Λουκά Γεωργιάδη, επίκουρο καθηγητή του Τμήματος Πληροφορικής του Πανεπιστημίου Ιωαννίνων για το χρόνο που διέθεσαν να μελετήσουν τη μεταπτυχιακή μου εργασία. Τα σχόλιά τους υπήρξαν εποικοδομητικά και συντέλεσαν στην βελτίωση της εργασίας.

Ευχαριστώ τη σύζυγό μου Ειρήνη για την αμέριστη συμπαράσταση και υποστήριξη που μου παρείχε καθώς επίσης και τα δυο παιδιά μας Ανθία και Κωνσταντίνο για το χρόνο που μου δάνεισαν.

Τέλος ευχαριστώ τους γονείς μου που με πολλές θυσίες μου έδωσαν τη δυνατότητα να σπουδάσω στο τμήμα που αγαπώ.

Περίληψη

Μελετούμε το πρόβλημα της κάλυψης ορθογωνίων πολυγώνων με το ελάχιστο πλήθος από r-αστέρες. Ένα σημείο q σε ένα ορθογώνιο πολύγωνο P είναι r-ορατό από ένα σημείο $p \in P$ αν το ορθογώνιο παραλληλόγραμμο (με τις πλευρές του παράλληλες προς τους άξονες) με τα p και q ως απέναντι κορυφές ανήκει στο P. Ένα ορθογώνιο πολύγωνο P είναι r-αστέρας αν υπάρχει σημείο $p \in P$ τέτοιο ώστε κάθε σημείο του P είναι r-ορατό από το p. Με το πρόβλημα της κάλυψης απλών ορθογωνίων πολυγώνων με r-αστέρες ασχολήθηκαν οι Worman και Keil, οι οποίοι περιέγραψαν έναν αλγόριθμο τάξης $O(n^{17}\text{polylog}n)$ όπου n είναι το πλήθος κορυφών του πολυγώνου.

Σε αυτή την εργασία, ασχολούμαστε με το παραπάνω πρόβλημα σε απλά ορθογώνια πολύγωνα κλάσης-3. Ένα ορθογώνιο πολύγωνο κλάσης-3 επιτρέπεται να έχει εσοχές κατά μήκος το πολύ τριών διαφορετικών προσανατολισμών. Εκμεταλλευόμενοι γεωμετρικές ιδιότητες αυτής της κλάσης πολυγώνων, επιτυγχάνουμε να δώσουμε έναν αλγόριθμο $O(n^2)$ πολυπλοκότητας χρόνου. Πιστεύουμε ότι αυτός ο αλγόριθμος μπορεί να μας οδηγήσει σε καλύτερες λύσεις από αυτές των Worman και Keil για το πρόβλημα της κάλυψης γενικών απλών ορθογωνίων πολυγώνων με το ελάχιστος πλήθος από r-αστέρες.

Abstract

We consider the problem of covering simple orthogonal polygons with the minimum number of r-stars. A point q in an orthogonal polygon P is r-visible from a point $p \in P$ if the axis parallel rectangle with p, q at opposite corners lies within P; then, an orthogonal polygon P is an r-star if there exists a point $p \in P$ such that every point in P is r-visible from p. The problem of covering a simple orthogonal polygon with the minimum number of r-stars has been considered by Worman and Keil [10] who described an $O(n^{17}\text{poly log } n)$-time algorithm where n is the size of the given polygon.

In this paper, we consider the above problem on simple class-3 orthogonal polygons; a class-3 orthogonal polygon is defined to have dents along at most 3 different orientations. By taking advantage of geometric properties of these polygons, we are able to provide an $O(n^2)$-time algorithm; our algorithm paves the way for obtaining algorithms for the problem on general simple orthogonal polygons that are faster than Worman and Keil's.

Περιεχόμενα

1. Εισαγωγή

Σε αυτό το κεφάλαιο δίνουμε τον ορισμό βασικών εννοιών οι οποίες κρίνονται απαραίτητες για την κατανόηση του προβλήματος που θα μας απασχολήσει σε αυτήν τη μεταπτυχιακή εργασία. Έπειτα θα αναφερθούμε στο αντικείμενο της μεταπτυχιακής εργασίας και θα δώσουμε κάποια σχετικά ερευνητικά αποτελέσματα. Τέλος θα δοθεί η δομή της εργασίας μας.

1.1 Ορισμοί Βασικών Εννοιών

Απλό πολύγωνο είναι η περιοχή του επιπέδου που περικλείεται από ένα πεπερασμένο σύνολο ευθυγράμμων τμημάτων τα οποία σχηματίζουν μια απλή κλειστή πολυγωνική γραμμή. Έστω ότι τα u_1, u_2, ..., u_n είναι n σημεία στο επίπεδο, και έστω ότι τα $e_1=u_1u_2$, $e_2=u_2u_3$, ..., $e_n=u_nu_1$ 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 άκρο: $e_i \cap e_{i+1}=u_{i+1}$ για κάθε i=1,....,n-1 και $e_n \cap e_1=u_1$.

2. Μη διαδοχικά τμήματα δεν τέμνονται: ei ∩ ej = ∅ για κάθε 1≤i<j-1 και j ≤ n.
3. Ο λόγος που τα ευθύγραμμα τμήματα σχηματίζουν μια πολυγωνική γραμμή είναι γιατί είναι συνδεδεμένα στη σειρά το ένα μετά το άλλο. Ο λόγος που αυτή η γραμμή είναι κλειστή είναι γιατί τα τμήματα κλείνουν σε έναν "κύκλο". Ο λόγος που αυτή η κλειστή γραμμή είναι απλή είναι γιατί μη

διαδοχικά τμήματα δεν τέμνονται (στο Σχήμα 1.1 δίνονται δύο πολύγωνα μη-απλά).

Σχήμα 1.1: Μη απλά πολύγωνα

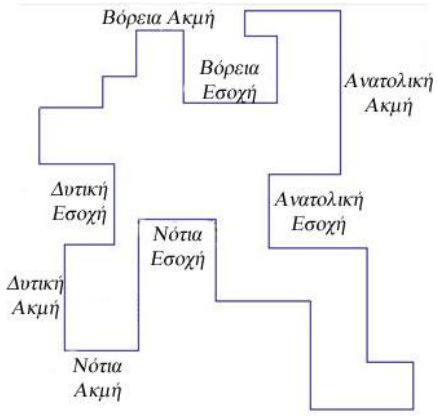
Κορυφές ονομάζονται τα σημεία u_i του πολυγώνου ενώ *ακμές* ονομάζονται τα τμήματα e_i του πολυγώνου.

Ορθογώνιο πολύγωνο είναι ένα πολύγωνο του οποίου οι ακμές είναι παράλληλες προς τους άξονες του καρτεσιανού συστήματος συντεταγμένων. Επειδή ένα ορθογώνιο πολύγωνο έχει μόνο είτε οριζόντιες είτε κάθετες ακμές, μπορούμε εύκολα να χαρακτηρίσουμε τις ακμές του σύμφωνα με τα σημεία του ορίζοντα (Βορράς, Νότος, Ανατολή, Δύση). Συγκεκριμένα:

Ως *Βόρεια ακμή* χαρακτηρίζουμε μία ακμή ενός ορθογωνίου πολυγώνου όταν το κάθετο διάνυσμα στην ακμή με φορά προς τα έξω δείχνει προς τον Βορρά. Αντίστοιχα χαρακτηρίζουμε μία ακμή ως *Νότια*, *Ανατολική*, *Δυτική*.

Εσοχές ενός ορθογωνίου πολυγώνου ονομάζονται οι ακμές των οποίων τα άκρα είναι μη-κυρτές κορυφές. Εάν η ακμή που ορίζει μία εσοχή είναι Βόρεια τότε η εσοχή χαρακτηρίζεται ως Βόρεια εσοχή, και παρόμοια έχουμε Νότιες εσοχές, Ανατολικές εσοχές και Δυτικές εσοχές. (Σχήμα 1.2)

Κυρτό ορθογώνιο πολύγωνο είναι ένα πολύγωνο όταν το ευθύγραμμο τμήμα μεταξύ δύο οποιωνδήποτε σημείων του στην ίδια οριζόντια ή στην ίδια κατακόρυφη γραμμή ανήκει στο πολύγωνο.

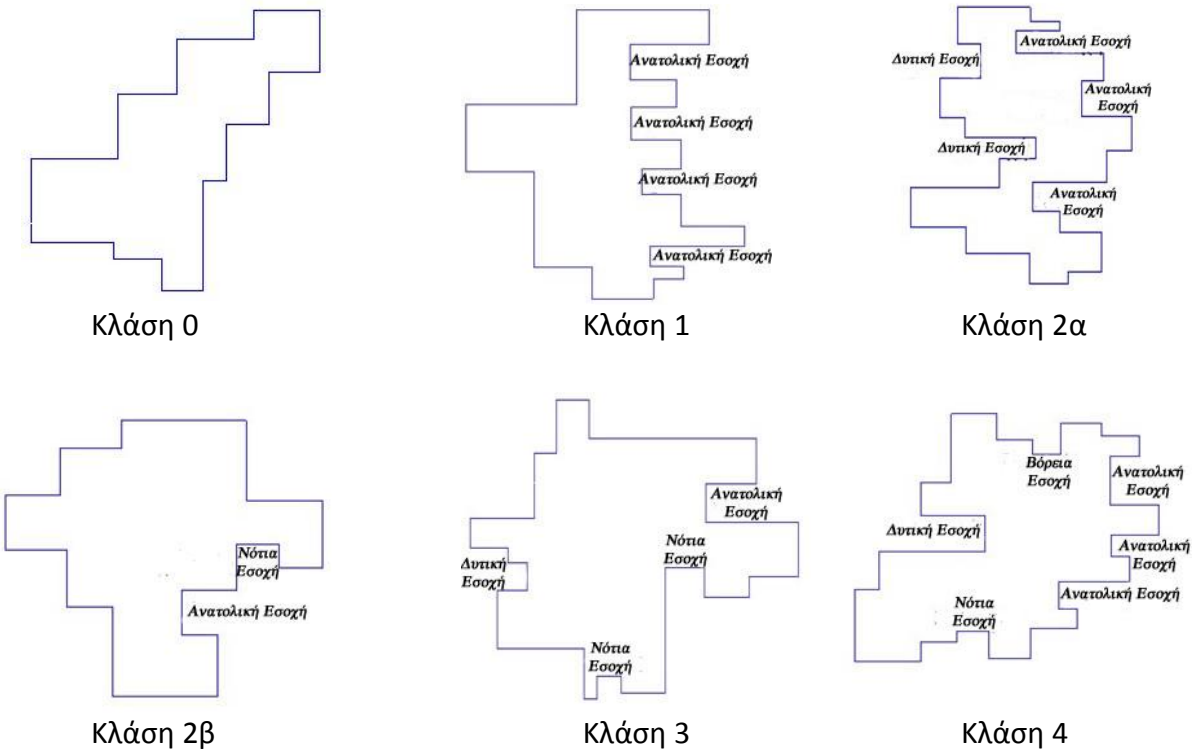


Σχήμα 1.2: Τα είδη των ακμών και των εσοχών

Κλάσεις ορθογωνίων πολυγώνων: Ένα ορθογώνιο πολύγωνο *κλάσης k* ($0 \leq k \leq 4$) έχει εσοχές με το πολύ k διαφορετικούς προσανατολισμούς. Έτσι έχουμε:

- *Κλάση 0*: δεν έχει καμία εσοχή άρα είναι ένα κυρτό ορθογώνιο πολύγωνο
- *Κλάση 1*: έχει ένα είδος εσοχών (είτε Βόρειες, είτε Νότιες, είτε Δυτικές, είτε Ανατολικές)
- *Κλάση 2*: έχει δύο είδη εσοχών και χωρίζεται σε δυο υποκατηγορίες:
    - *Κλάση 2α*: όπου τα δύο είδη εσοχών είναι «απέναντι» (Δυτικές και Ανατολικές, ή Βόρειες και Νότιες)
    - *Κλάση 2β*: όπου τα δύο είδη εσοχών είναι κάθετα μεταξύ τους (Δυτικές-Βόρειες, Βόρειες-Ανατολικές, Ανατολικές-Νότιες, Νότιες-Δυτικές)
- *Κλάση 3*: έχει τρία είδη εσοχών
- *Κλάση 4*: έχει τέσσερα είδη εσοχών

Στο πάνω μέρος του Σχήματος 1.3 παρατηρούμε από αριστερά προς τα δεξιά: ένα κυρτό ορθογώνιο πολύγωνο κλάσης 0, ένα ορθογώνιο πολύγωνο κλάσης 1 με Ανατολικές Εσοχές, ένα ορθογώνιο πολύγωνο κλάσης 2α με Ανατολικές και Δυτικές Εσοχές. Στο κάτω μέρος παρατηρούμε από αριστερά προς τα δεξιά ένα ορθογώνιο πολύγωνο κλάσης 2β με μία Ανατολική και μία Νότια Εσοχή, ένα ορθογώνιο πολύγωνο κλάσης 3 που δεν έχει Βόρειες Εσοχές και ένα ορθογώνιο πολύγωνο κλάσης 4 που αποτελεί τη γενική περίπτωση ορθογωνίου πολυγώνου.



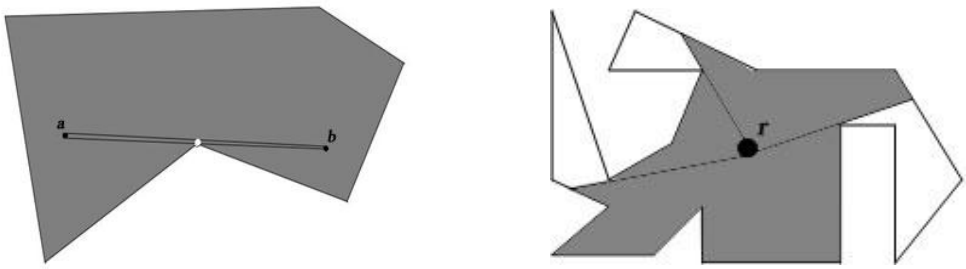
Κλάση 0 Κλάση 1 Κλάση 2α

Κλάση 2β Κλάση 3 Κλάση 4

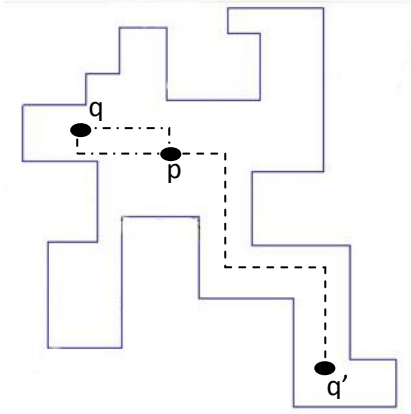
Σχήμα 1.3: Κλάσεις ορθογωνίων πολυγώνων

Ορατότητα μεταξύ των σημείων x και y (ή το σημείο y είναι ορατό από το x) υπάρχει αν και μόνο αν το κλειστό ευθύγραμμο τμήμα xy δεν τέμνει το εξωτερικό του πολυγώνου P: $xy \subseteq P$. Ο ορισμός αυτός επιτρέπει την περίπτωση όπου το τμήμα xy έχει κοινά σημεία με το σύνορο του πολυγώνου.

Στο Σχήμα 1.4 αριστερά το σημείο b είναι ορατό από το σημείο a (το ευθύγραμμο τμήμα μεταξύ τους έχει και ένα σημείο κοινό με το σύνορο του πολυγώνου). Στο Σχήμα 1.4 δεξιά η γκριζαρισμένη περιοχή είναι η περιοχή του πολυγώνου που είναι ορατή από το σημείο r.



Σχήμα 1.4: Ορατότητα



Σχήμα 1.5: r-ορατότητα, s-ορατότητα

s-ορατότητα: Έστω δύο σημεία a,b ενός ορθογωνίου πολυγώνου P. Το σημείο a είναι s-ορατό από το σημείο b όταν υπάρχει ένα κυρτό ορθογώνιο πολύγωνο εντός του πολυγώνου P που τα περιέχει (στο Σχήμα 1.5 το σημείο q' είναι s-ορατό από το σημείο p) .

s-αστέρας: Είναι ένα πολύγωνο P όταν υπάρχει ένα σημείο του από το οποίο όλα τα σημεία του πολυγώνου είναι s-ορατά.

r-ορατότητα: Έστω δύο σημεία a,b ενός ορθογωνίου πολυγώνου P. Το σημείο a είναι r-ορατό από το σημείο b όταν υπάρχει ένα ορθογώνιο παραλληλόγραμμο (με πλευρές παράλληλες στους άξονες του καρτεσιανού συστήματος συντεταγμένων) με τα a,b να βρίσκονται σε δυο απέναντι κορυφές του το οποίο ανήκει στο πολύγωνο P (στο Σχήμα 1.5 το σημείο q είναι r-ορατό από το σημείο p).

r-αστέρας: Είναι ένα πολύγωνο P όταν υπάρχει ένα σημείο του από το οποίο όλα τα σημεία του πολυγώνου είναι r-ορατά.

Ένα σύνολο πολυγώνων $S=\{P_1,P_2,\ldots,P_k\}$ αποτελεί μία *κάλυψη* ενός πολυγώνου P εάν η ένωση όλων των πολυγώνων στο S είναι το P. Αν τα κομμάτια αυτού του συνόλου S είναι και ξένα μεταξύ τους (με εξαίρεση τα σύνορά τους) τότε έχουμε μια *διαμέριση* του πολυγώνου P στο σύνολο S.

1.2 Το αντικείμενο της μεταπτυχιακής εργασίας

Στόχος της μεταπτυχιακής εργασίας είναι η εύρεση ενός αποδοτικού αλγορίθμου ο οποίος θα υπολογίζει τις θέσεις ενός συνόλου φυλάκων ελαχίστου πληθαρίθμου οι οποίοι είναι απαραίτητοι για να εποπτευθεί με r-ορατότητα ένα απλό ορθογώνιο πολύγωνο κλάσης 3. Το πρόβλημα αυτό αποτελεί υποπερίπτωση του γενικότερου προβλήματος της επόπτευσης πινακοθήκης σε πολύγωνα.

1.3 Σχετικά ερευνητικά αποτελέσματα

Πρώτος ο Klee το 1973 έθεσε το ερώτημα για το πόσοι φύλακες είναι αρκετοί να εποπτεύσουν ένα πολύγωνο n κορυφών. Με αυτό ως αφορμή οι Chvátal και Fisk [8] παρουσίασαν το Θεώρημα της Πινακοθήκης. Σύμφωνα με αυτό, $\lfloor n/3 \rfloor$ ακίνητοι φύλακες αρκούν και μερικές φορές είναι απαραίτητοι για να εποπτεύσουν ένα απλό n-γωνο. Ο Aggarwal [1] απέδειξε ότι το πρόβλημα του να βρεθεί το ελάχιστο πλήθος από φύλακες που εποπτεύουν ένα πολύγωνο είναι NP-πλήρες, για αυτό και πολλές υποκατηγορίες πολυγώνων έχουν μελετηθεί ξεχωριστά για καλύτερα αποτελέσματα [8,9].

Μία τέτοια υποκατηγορία αποτελούν και τα ορθογώνια πολύγωνα. Το Θεώρημα της Πινακοθήκης αναφέρει ότι $\lfloor n/4 \rfloor$ ακίνητοι φύλακες είναι αρκετοί και μερικές φορές απαραίτητοι για να εποπτευθεί ένα ορθογώνιο πολύγωνο n κορυφών [4]. Με τα ορθογώνια πολύγωνα ασχολήθηκαν οι Culberson και Reckhow οι οποίοι τα διέκριναν στις κλάσεις 0, 1, 2, 3, 4 [2].

Το 1986 ο J.M. Keil [5] περιέγραψε έναν αλγόριθμο για τη βέλτιστη κάλυψη ενός ορθογωνίου πολυγώνου κλάσης 2α με r-αστέρες σε πολυπλοκότητα $O(n^2)$. Οι Culberson και Reckhow [2] τρία χρόνια μετά παρουσίασαν έναν O(n) αλγόριθμο που βρίσκει το ελάχιστο πλήθος r-αστέρων που απαιτούνται για την κάλυψη ενός κλάσης 2α ορθογωνίου πολυγώνου, καθώς επίσης κι ένα διαφορετικό αλγόριθμο $O(n^2)$ από αυτόν του Keil που επιλύει το πρόβλημα και για τα ορθογώνια πολύγωνα κλάσης 2β. Αργότερα, το 1992, οι L. Gewali, M. Keil και S.C. Ntafos [3] έδωσαν έναν O(n) αλγόριθμο που υπολογίζει το ελάχιστο πλήθος από φύλακες για τα ορθογώνια πολύγωνα κλάσης 2α. Αλγόριθμο τάξης $O(n^{17}\text{polylogn})$ που επιλύει τη γενική

περίπτωση παρουσίασαν οι Worman και Keil [10] το 2007, με μία γραφοθεωρητική προσέγγιση του προβλήματος.

1.4 Η δομή της μεταπτυχιακής εργασίας

Στο Κεφάλαιο 2 θα αναλύσουμε το πρόβλημά μας και θα δώσουμε το θεωρητικό υπόβαθρο για την κατανόηση του προβλήματος. Έπειτα θα δώσουμε τον αλγόριθμο που επινοήσαμε μαζί με την πολυπλοκότητά του. Τελικά, θα κλείσουμε με κάποια παραδείγματα εφαρμογής του αλγορίθμου.

Στο Κεφάλαιο 3 θα δοθεί η υλοποίηση του αλγορίθμου. Επίσης, θα αναφερθούμε στη μορφή της εισόδου του προγράμματός μας και στην έξοδο που δίνει. Ακόμη, θα αναφέρουμε τις δομές δεδομένων που έχουμε χρησιμοποιήσει καθώς και επιλεγμένες συναρτήσεις του κώδικά μας. Ακολούθως θα δώσουμε ενδεικτικές εκτελέσεις του προγράμματος.

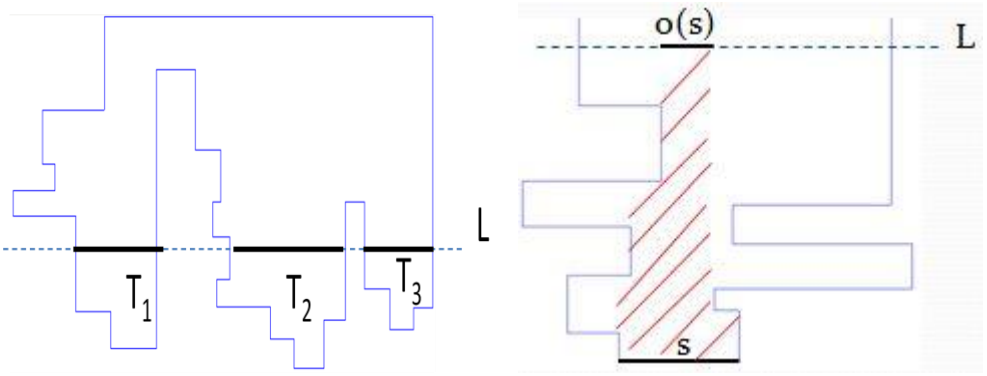
Στο Κεφάλαιο 4 θα ακολουθήσουν τα συμπεράσματα καθώς και κάποιες επεκτάσεις που θα μπορούσαν να γίνουν με βάση τον αλγόριθμο που δίνουμε.

2. Ο αλγόριθμος

2.1 Θεωρητικό υπόβαθρο

Με τον όρο ορθογώνια πολύγωνα στο εξής θα αναφερόμαστε σε απλά ορθογώνια πολύγωνα. Έστω ορθογώνιο πολύγωνο p το οποίο δεν έχει βόρειες εσοχές. Η τομή ενός τέτοιου πολυγώνου με μία οριζόντια γραμμή L αποτελείται από διάφορα οριζόντια τμήματα. Επειδή το P δεν έχει βόρειες εσοχές αυτά τα τμήματα αντιστοιχούν σε ξένα τμήματα του πολυγώνου P κάτω από τη γραμμή L. Για ευκολία ονομάζουμε κάθε τέτοιο μέρος του P *σκέλος*. Στο Σχήμα 2.2.1 αριστερά παρατηρούμε τρία σκέλη κάτω από την γραμμή L τα οποία είναι ξένα μεταξύ τους. Ακολούθως, επεκτείνουμε τις έννοιες της «στάθμης» και του «ευθυγράμμου τμήματος τομής» [3, 6] τις οποίες χρησιμοποιούμε στον αλγόριθμο: *στάθμη* ενός σημείου ή μιας οριζόντιας ακμής του πολυγώνου είναι η y-συντεταγμένη του, το *ευθύγραμμο τμήμα τομής* του P ή ενός σκέλους T είναι ένα μεγιστοτικό (κλειστό) οριζόντιο τμήμα του P ή του T. Για συντομία θα αναφερόμαστε στο ευθύγραμμο τμήμα τομής ως *ευθ. τμήμα τομής* στη συνέχεια της εργασίας. Μία *ορθογώνια προβολή o(s)* ενός οριζοντίου τμήματος s στη στάθμη l του P στο ευθ. τμήμα τομής s′ στη στάθμη l′ ≥ l είναι το μεγιστοτικό υποτμήμα του s′ τέτοιο ώστε για κάθε σημείο α της o(s) να υπάρχει ένα κατακόρυφο ευθ. τμήμα στο P που να περνά από το α και να τέμνει το s. Στο Σχήμα 2.2.1 δεξιά παρατηρούμε την ορθογώνια προβολή του s στη στάθμη L. Τέλος, ορίζουμε ως x-εύρος μιας οριζόντιας ακμής e ενός

πολυγώνου το σύνολο των x συντεταγμένων των σημείων της e. Με άλλα λόγια, αν τα άκρα της οριζόντιας ακμής έχουν x-συντεταγμένες x_l και x_r, τότε το x-εύρος της ακμής είναι (x_l, x_r). Σημειώνεται ότι μολονότι ένα πολύγωνο θεωρείται ως ένα κλειστό σύνολο, θεωρούμε τις ακμές να είναι ανοιχτά σύνολα (χωρίς δηλαδή τα άκρα τους) και έτσι τα x-εύρη είναι επίσης ανοιχτά σύνολα.



Σχήμα 2.2.1

Το επόμενο λήμμα μας παρέχει δυο σημαντικές ιδιότητες των ορθογωνίων πολυγώνων κλάσης 3.

Λήμμα 2.1 Έστω P ένα ορθογώνιο πολύγωνο κλάσης 3 και ας υποθέσουμε ότι το P δεν έχει βόρειες εσοχές. Έτσι, ισχύουν τα παρακάτω:

i. Το πολύγωνο P έχει μοναδική υψηλότερη ακμή.
ii. Έστω ότι σαρώνουμε το πολύγωνο από κάτω προς τα πάνω. Κάθε ακμή που συναντούμε είναι προσπίπτουσα στο σύνορο του σαρωμένου πολυγώνου.
iii. Έστω T ένας σκέλος τη στιγμή που το P τέμνεται από τη γραμμή σάρωσης στη στάθμη l. Έστω s_1 και s_2 τα ευθ. τμήματα τομής του T στις στάθμες l_1 και l_2 αντίστοιχα, όπου $l_1 < l_2 \leq l$, τέτοια ώστε να υπάρχει ένα κατακόρυφο ευθύγραμμο τμήμα στο T που να τέμνει και το s_1 και το s_2. Τότε, η ορθογώνια προβολή του s_1 στη στάθμη l είναι υποσύνολο της ορθογώνιας προβολής του s_2 στην l.

Απόδειξη: Το (i) και το (ii) προκύπτουν εύκολα λόγω της έλλειψης βόρειων εσοχών. Το (iii) προκύπτει από την παρατήρηση ότι η ορθογώνια προβολή του ευθ.

τμήματος τομής του T της στάθμης l_1 στη στάθμη l_2 είναι υποσύνολο του ευθ. τμήματος τομής του T στη στάθμη l_2.

2.2 Ο Αλγόριθμος

Ο αλγόριθμος εφαρμόζει τη σάρωση επιπέδου (plane-sweep) όπως κάνουν οι αλγόριθμοι στο [3,6]. Υποθέτουμε ότι το δοθέν πολύγωνο κλάσης 3 δεν έχει βόρειες εσοχές και το σαρώνουμε από κάτω προς τα πάνω (έτσι εκμεταλλευόμαστε το Λήμμα 2.1). Ο αλγόριθμος μας επιστρέφει τις θέσεις ενός ελάχιστου συνόλου από r-φύλακες οι οποίοι επιβλέπουν ολόκληρο το πολύγωνο. Στη συνέχεια δίνουμε τις βασικές ιδέες του αλγορίθμου, έπειτα την περιγραφή του με λεπτομέρεια και τέλος αναλύουμε την πολυπλοκότητά του.

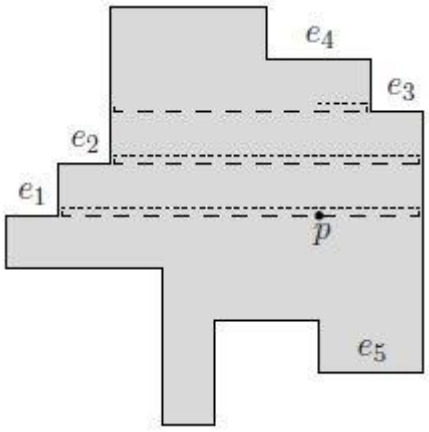
2.2.1 Καταχώριση και επεξεργασία φυλάκων

Θα διατηρήσουμε τη σύμβαση των αλγορίθμων του Gewali, Keil και Ntafos [3] και αργότερα των Lingas, Wasylewicz και Zylinski [6] σύμφωνα με την οποία οι φύλακες θα τοποθετούνται στο αριστερότερο άκρο του υψηλότερου δυνατού επιπέδου. Έτσι κάθε φύλακας θα τοποθετείται στη στάθμη μιας βόρειας ακμής. Προκειμένου να βρούμε τις κατάλληλες τοποθεσίες των φυλάκων, για κάθε φύλακα θα διατηρούμε:

- Το εύρος θέσης, το οποίο είναι το εύρος των x-συντεταγμένων των σημείων στα οποία μπορεί να τοποθετηθεί ο φύλακας.
- Το εύρος επόπτευσης, το οποίο είναι το εύρος των x-συντεταγμένων των σημείων τα όποια είναι ορατά από το φύλακα πάνω από την τρέχουσα θέση της γραμμής σάρωσης.

Επειδή δεν υπάρχουν βόρειες εσοχές, το καθένα από τα παραπάνω εύρη είναι ένα συνεχές διάστημα x-συντεταγμένων, και επειδή τοποθετούμε φύλακες έτσι ώστε να μπορούν να εποπτεύσουν όσο το δυνατόν μεγαλύτερο μέρος του πολυγώνου το οποίο βρίσκεται πάνω από αυτούς. Ισχύει πάντα ότι το εύρος θέσης είναι υποσύνολο του εύρους επόπτευσης.

Αρχικά, όταν τοποθετούμε ένα φύλακα σε μία στάθμη l σε ένα σκέλος T, το εύρος θέσης και το εύρος επόπτευσης συμπίπτουν με το εύρος x-συντεταγμένων του ευθ. τμήματος τομής του T στη στάθμη l. Όσο η γραμμή σάρωσης προχωράει προς τα πάνω, τα δύο εύρη περικόπτονται τόσο όσο και η τομή τους με το x-εύρος των βόρειων ακμών. Τέλος, όταν βρούμε μια βόρεια ακμή e η οποία υπερκαλύπτει το (πιθανώς περικομμένο) εύρος θέσης ενός φύλακα, τότε ένας φύλακας g τοποθετείται στο σημείο (x_l,l) όπου το x_l είναι το αριστερό όριο του εύρους θέσης του φύλακα ακριβώς αμέσως πριν συναντήσουμε τη βόρεια ακμή e. Ο φύλακας g που τοποθετήσαμε δεν μπορεί να δει κανένα σημείο του πολυγώνου P πάνω από τη στάθμη της γραμμής e.



Σχήμα 2.2.1

Στο Σχήμα 2.2.1 παρουσιάζουμε τις αλλαγές στο εύρος θέσης και στο εύρος επόπτευσης ενός φύλακα καθώς σαρώνουμε το ορθογώνιο πολύγωνο από κάτω προς τα πάνω. Πριν φτάσουμε στην ακμή e_1 όλα τα σημεία του σαρωμένου μέρους του πολυγώνου μπορούν να εποπτευθούν από φύλακες (σε κατάλληλες θέσεις) από φύλακες σε υψηλότερη στάθμη από τη γραμμή σάρωσης. Αυτό παύει να ισχύει (π.χ. τα σημεία κάτω από τη βόρεια ακμή e_1) όταν φτάνουμε στην e_1. Σύμφωνα με αυτή την παρατήρηση βλέπουμε ότι ένας φύλακας πρέπει αναγκαστικά να τοποθετηθεί στη στάθμη της e_1 αρχικοποιώντας το εύρος θέσης του και το εύρος ορατότητάς του με το x-εύρος του ευθ. τμήματος τομής στη στάθμη της e_1. Αφού επεξεργαστούμε και την e_1 και τα δυο εύρη περικόπτονται. Αμέσως μετά έρχεται η e_2 και περικόπτονται κι άλλο. Μετά έρχεται η e_3 και παρατηρούμε ότι το εύρος θέσης του

φύλακα πρέπει να περιοριστεί στο εύρος x-συντεταγμένων του e_5 (το εύρος επόπτευσης όμως δεν αλλάζει). Έπειτα και το εύρος θέσης και το εύρος επόπτευσης περικόπτονται από την e_3 και καταλήγουν όπως φαίνεται στο Σχήμα 2.2.1 στη στάθμη της e_3. Το τελικό εύρος θέσης πριν τοποθετηθεί οριστικά ο φύλακας δημιουργείται όταν έρχεται η e_4 και είναι το υποσύνολο της τομής του x-εύρους της βόρειας ακμής του e_4 με το εύρος θέσης. Έτσι ο φύλακας g τοποθετείται στο σημείο p που φαίνεται στο σχήμα.

2.2.2 Τοποθέτηση Φυλάκων

Έστω μια οποιαδήποτε νότια ακμή e. Όσο δεν έχουμε βρει μια βόρεια ακμή που να τέμνει την ορθογώνια προβολή της e στη στάθμη της βόρειας ακμής, τότε ένας φύλακας σε μία στάθμη υψηλότερη από τη στάθμη της βόρειας ακμής μπορεί να δει όλη τη νότια ακμή e (π.χ. η e_1 στο Σχήμα 2.2.2α). Αν όμως μια βόρεια ακμή d τμήσει την ορθογώνια προβολή της νότιας ακμής e στη στάθμη της d τότε ένας φύλακας πρέπει να τοποθετηθεί σε μία στάθμη l όπου l $\in$ [στάθμη e, στάθμη d] επειδή κανένας φύλακας ψηλότερα από τη στάθμη της d δεν μπορεί να δει όλη την e (π.χ. η e_2 στο Σχήμα 2.2.2α). Επιπλέον, έστω l η στάθμη που πρέπει να τοποθετηθεί ένας φύλακας, τότε πρέπει να τοποθετηθεί στην ορθογώνια προβολή του ευθ. τμήματος τομής του επιπέδου e στη στάθμη l για να μπορεί να δει ολόκληρη την ακμή e.

Σχήμα 2.2.2

Για να εκμεταλλευτούμε τις παραπάνω παρατηρήσεις, για κάθε νότια ακμή e κρατούμε:

- Ένα εύρος ανάθεσης, όπου είναι το x-εύρος της ακμής e.

- Ένα εύρος τοποθέτησης, όπου είναι το εύρος των x-συντεταγμένων του ευθ. τμήματος τομής στη στάθμη της e.

Καθένα από αυτά τα εύρη είναι ένα συνεχές διάστημα x-συντεταγμένων (το εύρος ανάθεσης είναι ανοιχτό, το εύρος τοποθέτησης είναι κλειστό), και ισχύει πάντα ότι το εύρος ανάθεσης της νότιας ακμής είναι υποσύνολο του εύρους τοποθέτησης. Το Σχήμα 2.2.2.b δείχνει το εύρος ανάθεσης (με τελείες) και το εύρος τοποθέτησης (με παύλες) για τη νότια ακμή e.

Παρακάτω θα δούμε πώς χρησιμοποιούνται τα δυο αυτά εύρη μιας νότιας ακμής e. Κατά τη διάρκεια της σάρωσης, όσο βρίσκουμε x-εύρη βόρειων ακμών τα οποία δεν τέμνουν κανένα από τα δύο είδη ευρών της νότιας ακμής e, δε συμβαίνει καμία αλλαγή. Αν συναντήσουμε μια βόρεια ακμή της οποίας το x-εύρος τέμνει το εύρος τοποθέτησης της e, τότε απλά περικόπτουμε το εύρος τοποθέτησης. Αν όμως συναντήσουμε μια βόρεια ακμή d της οποίας το x-εύρος τέμνει το εύρος ανάθεσης της e τότε πρέπει άμεσα να ορίσουμε κάποιο φύλακα για να επιβλέπει τη νότια ακμή e. Ο φύλακας που θα ορίσουμε πρέπει να πληροί τις δυο παρακάτω προϋποθέσεις:

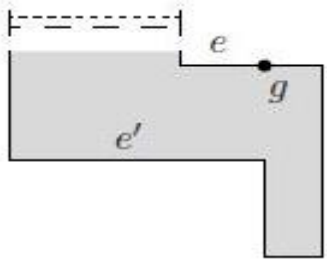
- Να βρίσκεται σε στάθμη μεταξύ (και περιέχοντας) της στάθμης e και της στάθμης d και
- Το εύρος θέσης του φύλακα θα πρέπει να τέμνει το εύρος τοποθέτησης της ακμής e.

Έτσι, το εύρος θέσης του φύλακα που θα επιλεγεί για να επιβλέπει τη νότια ακμή e ορίζεται ίσο με την τομή του εύρους θέσης του φύλακα και του εύρους τοποθέτησης της e. Με αυτό τον τρόπο ο φύλακας είναι ικανός να επιβλέπει και την e και όσο το δυνατόν περισσότερο από το μη εποπτευόμενο κομμάτι του πολυγώνου (αυτός είναι κι ο λόγος της περικοπής του εύρους τοποθέτησης της e). Επειδή το εύρος ανάθεσης και το εύρος τοποθέτησης μιας νότιας ακμής μας βοηθούν:

i. να αποφασίσουμε αν χρειάζεται ένας φύλακας ή όχι για να εποπτευθεί αυτή η ακμή

ii. το εύρος θέσης αυτού του φύλακα,

πραγματοποιούμε ένα αίτημα φύλακα για κάθε νότια ακμή σχετιζόμενο με το εύρος ανάθεσης και το εύρος τοποθέτησης της ακμής.



Σχήμα 2.2.3

Στην πραγματικότητα υπάρχει άλλη μια περίπτωση που χρειάζεται ένα αίτημα φύλακα. Ας μελετήσουμε το Σχήμα 2.2.3. Μόλις μας έρθει η βόρεια ακμή e, ένας φύλακας g τοποθετείται όπως φαίνεται για να επιβλέψει τη δεξιότερη νότια ακμή. Ο ίδιος φύλακας επιβλέπει και τη νότια ακμή e΄. Αυτό μας οδηγεί στο να αφαιρέσουμε το αίτημα φύλακα που παράγει η e΄. Βλέπουμε όμως ότι αν δε χειριστούμε ειδικά αυτή την περίπτωση τίποτα δε θα μας δείχνει ότι χρειάζεται ένας φύλακας για να επιβλέψει το ευθ. τμήμα τομής μιας στάθμης ελαφρώς πάνω από τη στάθμη της e. Αυτό είναι λάθος και φαίνεται ξεκάθαρα στο Σχήμα 2.2.3 .

Έτσι λοιπόν πρέπει να χειριστούμε ειδικά αυτή την περίπτωση. Όταν έχουμε μια βόρεια ακμή e σε μία στάθμη l, αφού πρώτα την επεξεργαστούμε, στο τέλος ελέγχουμε το ευθ. τμήμα τομής της στάθμης l πλην το x-εύρος της βόρειας (έστω C αυτή η διαφορά). Έστω D η ένωση των ευρών επόπτευσης των διαθέσιμων φυλάκων με τα εύρη ανάθεσης των αιτημάτων φυλάκων. Ελέγχουμε την τομή της D με την περιοχή ακριβώς δεξιότερα του αριστερού άκρου της C (έστω E αυτή η τομή) και την τομή της D με την περιοχή ακριβώς αριστερότερα του δεξιού άκρου της C (έστω F αυτή η τομή). Διακρίνουμε τέσσερις περιπτώσεις:

i. Αν οι τομές E και F είναι μη κενές δεν κάνουμε τίποτα.
ii. Αν μία από τις τομές E και F είναι το κενό σύνολο τότε πρέπει να προσθέσουμε ένα αίτημα φύλακα. Αν υποθέσουμε ότι η τομή E είναι κενό σύνολο το αίτημα φύλακα που πρέπει να προσθέσουμε έχει εύρος

τοποθέτησης τη διαφορά C και εύρος ανάθεσης το ευθύγραμμο τμήμα με άκρα το αριστερό άκρο της C και δεξί άκρο το αριστερό άκρο της D.

iii. Αν οι τομές E και F είναι κενά σύνολα αλλά η ένωση D είναι διάφορη του κενού τότε προσθέτουμε δύο αιτήματα φύλακα. Και τα δύο έχουν εύρος τοποθέτησης τη διαφορά C. Το ένα αίτημα φύλακα έχει εύρος ανάθεσης το ευθύγραμμο τμήμα με αριστερό άκρο το αριστερό άκρο της C και δεξί άκρο το αριστερό άκρο της D. Το δεύτερο αίτημα φύλακα έχει εύρος ανάθεσης το ευθύγραμμο τμήμα με αριστερό άκρο το δεξί άκρο της D και δεξί άκρο το δεξί άκρο της C.

iv. Αν οι τομές E και F και η ένωση D είναι κενά σύνολα προσθέτουμε ένα αίτημα φύλακα με εύρος τοποθέτησης και εύρος ανάθεσης τη διαφορά C.

Από τα παραπάνω εξάγεται η ακόλουθη παρατήρηση.

Παρατήρηση 3.1 Ανά πάσα στιγμή τα εύρη ανάθεσης των αιτημάτων φυλάκων δεν επικαλύπτονται.

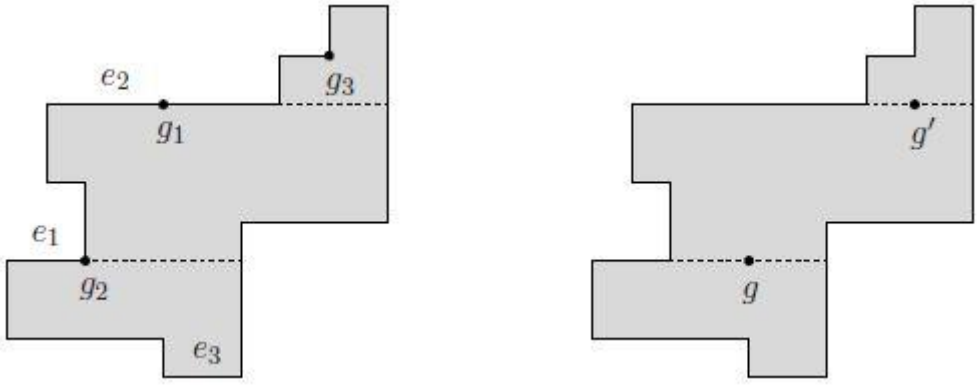
2.2.3 Επιλογή φύλακα για επόπτευση μιας νότιας ακμής

Πολλοί φύλακες σε διαφορετικές στάθμες του πολυγώνου μπορεί να είναι ικανοί να εποπτεύσουν μια νότια ακμή e′ όταν το εύρος ανάθεσης ενός αιτήματος φύλακα της e′ τέμνεται από το x-εύρος μιας βόρειας ακμής. Για να κάνουμε μια καλή επιλογή μεταξύ τους ακολουθούμε την τακτική που μας προτείνει το ακόλουθο λήμμα.

Λήμμα 2.2.3.1 Κάθε φορά που πρέπει να ικανοποιηθεί ένα αίτημα φύλακα αρκεί να επιλέξουμε τον χαμηλότερο μεταξύ όλων των φυλάκων που μπορούν να το εκπληρώσουν.

Η απόδειξη του λήμματος βασίζεται στο Λήμμα 2.1(iii). Θεωρούμε δυο φύλακες g_1 και g_2 στις στάθμες l_1 και l_2 αντίστοιχα με $l_1 < l_2$. Θυμίζουμε ότι το εύρος επόπτευσης ενός φύλακα αρχικοποιείται με το x-εύρος του ευθ. τμήματος τομής στη στάθμη του φύλακα και μετέπειτα περικόπτεται από βόρειες ακμές που συναντούμε. Έτσι, σε μία στάθμη l ψηλότερα από τη στάθμη του, ο φύλακας μπορεί να εποπτεύσει όλα τα σημεία της ορθογώνιας προβολής του αρχικού εύρους

επόπτευσης στη στάθμη l. Σύμφωνα όμως με το Λήμμα 2.1.(iii) το σύνολο των σημείων στη στάθμη $l>l_2$ τα οποία εποπτεύονται από το φύλακα g_1 είναι ένα υποσύνολο του συνόλου των σημείων στη στάθμη l που εποπτεύονται από το g_2. Στην πραγματικότητα υπάρχουν περιπτώσεις όπου διαλέγοντας ένα φύλακα διαφορετικό από το χαμηλότερο διαθέσιμο παίρνουμε λάθος αποτέλεσμα. Δείτε το Σχήμα 2.2.4. Όταν συναντούμε τις βόρειες ακμές e_1 και e_2, βλέπουμε ότι χρειάζονται φύλακες σε αυτά τα επίπεδα. Καθώς αναθέτουμε ένα φύλακα να επιβλέπει τη νότια ακμή e_3 βλέπουμε ότι αν διαλέξουμε φύλακα στη στάθμη του e_2 (τον g_1 στα αριστερά του Σχήματος 2.2.4) τότε ένας τρίτος φύλακας g_3 θα μας χρειαστεί για να εποπτευθεί το πολύγωνο. Βλέπουμε όμως στα δεξιά του Σχήματος 2.2.4 ότι δύο φύλακες αρκούν για να επιβλέψουν ολόκληρο το πολύγωνο.



Σχήμα 2.2.4

2.2.4 Περιγραφή του αλγορίθμου

Όπως έχουμε αναφέρει θα σαρώσουμε το δοθέν ορθογώνιο πολύγωνο κλάσης 3 από κάτω προς τα πάνω διατηρώντας κατάλληλες πληροφορίες για τα τρέχοντα σκέλη (στη συγκεκριμένη θέση της γραμμής σάρωσης). Επειδή πρέπει να εισάγουμε νέα σκέλη, να διαγράφουμε σκέλη και να αναζητούμε σε ποιο από τα τρέχοντα σκέλη προσπίπτει η ακμή (δες Λήμμα 2.1.(ii)), διατηρούμε τα σκέλη σε ένα ισοζυγισμένο δυαδικό δέντρο αναζήτησης αποθηκεύοντας από αριστερά προς τα δεξιά. Με αυτή τη δομή αυτές οι λειτουργίες απαιτούν O(logn) χρόνο. Όσον αφορά τα εύρη ανάθεσης των αιτημάτων φυλάκων, αυτά χρειάζονται τις λειτουργίες των εισαγωγών στα άκρα ,των διαγραφών από τα άκρα και τη σύνδεση δομών (η μία στο τέλος της άλλης). Για αυτό το λόγο χρησιμοποιούμε μία διπλά συνδεδεμένη

λίστα με δείκτες στα δύο άκρα τους. Έτσι οι λειτουργίες αυτές μπορούν να γίνουν σε σταθερό χρόνο. Χρειαζόμαστε επίσης μία λίστα με τα εύρη θέσης και εύρη επόπτευσης των φυλάκων ανά σκέλος Τ. Επίσης, κρατούμε δύο σύνολα, το σύνολο Διαθέσιμοι(Τ) και το σύνολο Τοποθετημένοι(Τ). Στους πρώτους αποθηκεύουμε τους φύλακες του Τ των οποίων τα εύρη θέσης δεν αποτελούν ένα σημείο και στους δεύτερους εκείνους που έχουν τοποθετηθεί σε ένα συγκεκριμένο σημείο.

Κατά τη διάρκεια της σάρωσης σταματάμε σε κάθε οριζόντια ακμή e και την επεξεργαζόμαστε. Αν η e είναι νότια ακμή ενημερώνουμε τις πληροφορίες που κρατάει το προσπίπτον (στην e) σκέλος και εισάγουμε ένα κατάλληλο αίτημα φύλακα. Αν η e είναι βόρεια ακμή επεξεργαζόμαστε τα αιτήματα φύλακες των οποίων τα εύρη ανάθεσης τέμνονται με το x-εύρος της e. Ανάλογα μπορεί να εισάγουμε ένα αίτημα φύλακα του δεύτερου είδους εφόσον χρειάζεται. Τοποθετούμε τους φύλακες των οποίων τα εύρη τοποθέτησης είναι υποσύνολα του x-εύρους της e και περικόπτουμε τα εύρη τοποθέτησης όλων των αιτημάτων φυλάκων καθώς επίσης και τα εύρη θέσης και εύρη επόπτευσης των φυλάκων. Αφού έχουμε επεξεργαστεί όλες τις οριζόντιες ακμές το σύνολο φυλάκων Τοποθετημένοι του μοναδικού (τελικά) σκέλους που «έκλεισε» από την τελευταία υψηλότερη ακμή (δες Λήμμα 2.1.(i)) μας δίνει τη θέση ενός ελαχίστου συνόλου από φύλακες r-ορατότητας.

Παρακάτω δίνουμε μια αναλυτική περιγραφή του αλγορίθμου σε ψευδοκώδικα.

g= φύλακας

g.εύρος_θέσης= εύρος θέσης φύλακα

g.εύρος_επόπτευσης= εύρος επόπτευσης φύλακα

r= αίτημα φύλακα

r.εύρος_ανάθεσης= εύρος ανάθεσης αιτήματος φύλακα

r.εύρος_τοποθέτησης= εύρος τοποθέτησης αιτήματος φύλακα

e= ακμή

e.x-εύρος= x-εύρος της ακμής e

Τοποθετημένοι= δεν βλέπουν κανένα σημείο πάνω από την τρέχουσα γραμμή

σάρωσης}

Αναλυτική περιγραφή

Αλγόριθμος Class3_rStar_Cover(P)
Είσοδος: ένα απλό ορθογώνιο πολύγωνο P κλάσης 3 (χωρίς Βόρειες Εσοχές)
Έξοδος: οι θέσεις ενός ελάχιστου συνόλου από r-φύλακες οι οποίοι επιβλέπουν ολόκληρο το πολύγωνο

```
1. ταξινόμησε τις Βόρειες και Νότιες ακμές του P κατά αύξουσα σειρά
   της y-συντεταγμένης του;
   δημιούργησε μία κενή δομή δεδομένων D_t για να αποθηκεύονται τα
   σκέλη

2. {σάρωση από κάτω προς τα πάνω}
   για κάθε Βόρεια ή Νότια ακμή e που διατρέχουμε κάνε
     αν η e είναι Νότια ακμή
     τότε δημιούργησε το αντίστοιχο αίτημα φύλακα r;
          εντόπισε την e στη δομή δεδομένων των σκελών;
          αν η e δεν ανήκει σε κάποιο από τα τρέχοντα σκέλη
          τότε δημιούργησε μία καινούρια εγγραφή για το καινούριο
               σκέλος T (το οποίο θα περιέχει μόνο την ακμή e) και
               εισάγαγέ το στη δομή D_t;
               εισάγαγε το r στη δομή με τα αιτήματα φύλακα του
               σκέλους T;
               Τοποθετημένοι(T)←∅;
               Διαθέσιμοι(T) ←∅;
          αλλιώς αν η e είναι μία Νότια Εσοχή
          τότε {συγχώνευση των σκελών T_1 και T_2 αριστερά και δεξιά της
                                                                    e}
               αφαίρεσε τις εγγραφές του T_1 και του T_2 από την D_t και
               εισάγαγε ένα νέο σκέλος T;
               συγχώνευσε τις δομές δεδομένων των φυλάκων και των
               αιτημάτων φυλάκων του T_1 και του T_2 (εισάγοντας ένα r
               στη (συγχωνευμένη) δομή δεδομένων με τα αιτήματα
               φυλάκων, και συσχέτισέ τες με το T;
          αλλιώς    {η e ανήκει σε ένα μοναδικό σκέλος T}
               εισάγαγε το r στη δομή δεδομένων με τα αιτήματα φύλακα;
     αλλιώς         {η e είναι Βόρεια ακμή}
          εντόπισε την e στη δομή δεδομένων των σκελών, έστω T το
          σκέλος του οποίου το σύνορο επεκτείνεται από την e;

          {επεξεργασία αιτημάτων φύλακα του T, των οποίων το εύρος
          ανάθεσης τέμνεται με το x-εύρος της e}
          για κάθε αίτημα φύλακα r στο T τέτοιο ώστε
          r.εύρος_ανάθεσης ∩ e.x-εύρος ≠ ∅ κάνε
          {η e είναι η πρώτη ακμή της οποίας το x-εύρος τέμνει το
          r.εύρος_ανάθεσης}
               αν ∃ φύλακες ∈ Διαθέσιμοι(T) των οποίων το εύρος θέσης
               είναι υποσύνολο του r.εύρους_τοποθέτησης
               τότε g← ο χαμηλότερος τέτοιος φύλακας;
               αλλιώς αν ∃ φύλακες ∈ Διαθέσιμοι(T)των οποίων το εύρος
               θέσης τέμνει το r.εύρος_τοποθέτησης
               τότε g← ο χαμηλότερος τέτοιος φύλακας;
                    g.εύρος_θέσης← g.εύρος_θέσης∩r.εύρος_τοποθέτησης;
               αλλιώς δημιούργησε ένα καινούριο φύλακα g και
               εισάγαγέ τον στους Διαθέσιμοι(T);
               g.στάθμη ← στάθμη της e;
```

```
            g.εύρος_επόπτευσης ← x-εύρος του ευθ. τμήματος τομής
            στη στάθμη της e;
            g.εύρος θέσης ← r.εύρος_τοποθέτησης;
        αφαίρεσε το r από τη δομή δεδομένων με τα αιτήματα φυλάκων
        του T;
    {επεξεργασία φυλάκων των οποίων το εύρος θέσης «καλύφθηκε» από
    την e}
    για κάθε φύλακα g τέτοιον ώστε το g.εύρος_θέσης ⊆ e.x-εύρος
    κάνε
        x_g←η x συντεταγμένη του αριστερού άκρου του g.εύρους_θέσης;
        τοποθέτησε το g στο (x_g,y_g)όπου y_g είναι η στάθμη του g;
        αφαίρεσε το g από το σύνολο Διαθέσιμοι(T) και εισάγαγέ το
        στο σύνολο Τοποθετημένοι(T);
    περίκοψε τα εύρη θέσης και τα εύρη επόπτευσης (όπου χρειάζεται)
    των φυλάκων g ∈ Διαθέσιμοι(T);
    περίκοψε τα εύρη τοποθέτησης (όπου χρειάζεται) των αιτημάτων
    φύλακα του T;
    {έλεγχος για τη δημιουργία αιτήματος φύλακα 2ου είδους}
    I ← x-εύρος του ευθ. τμήματος τομής – το x-εύρος της e;
    U ← η ένωση των ευρών επόπτευσης των φυλάκων g∈ Διαθέσιμοι(T)
        μαζί με τα εύρη ανάθεσης των αιτημάτων φύλακα του T;

    αν U=∅
    τότε πρόσθεσε ένα αίτημα φύλακα r;
        r.εύρος_ανάθεσης ← I
        r.εύρος_τοποθέτησης ← I;
    αλλιώς
        L ←διάστημα από αριστερό άκρο του I έως αριστερό άκρο του U
        αν L≠∅
        τότε πρόσθεσε ένα αίτημα φύλακα r;
        r.εύρος_τοποθέτησης ← I;
        r.εύρος_ανάθεσης ← (x_l,x_r),όπου x_l το αριστερό άκρο του I
        και x_r το αριστερό άκρο της U;
        R ← διάστημα από δεξί άκρο του U έως δεξί άκρο του I
        αν R≠∅
        τότε πρόσθεσε ένα αίτημα φύλακα r';
        r'.εύρος_τοποθέτησης ← I;
        r'.εύρος_ανάθεσης ← (x_l,x_r),όπου x_l το δεξί άκρο της U
        και x_r το δεξί άκρο του I;
3. αναφορά των θέσεων των φυλάκων που είναι αποθηκευμένοι στο
τελευταίο σύνολο Τοποθετημένοι.
```

Η ορθότητα του παραπάνω αλγορίθμου στηρίζεται στα Λήμματα 2.1 και 2.2.3.1 και στο γεγονός ότι ένας νέος φύλακας ανατίθεται όποτε οι μέχρι στιγμής φύλακες δεν μπορούν να εποπτεύσουν κάποιο τμήμα του πολυγώνου.

2.3 Πολυπλοκότητα του αλγορίθμου

Έστω n το πλήθος των κορυφών του δοθέντος ορθογωνίου πολυγώνου κλάσης 3. Το πλήθος των σκελών είναι O(n). Το πλήθος των αιτημάτων-φυλάκων είναι επίσης O(n) επειδή σε κάθε στιγμή έχουμε το πολύ ένα αίτημα-φύλακα για καθεμία από τις νότιες ακμές που συναντούμε (ή το πολύ δύο στην περίπτωση του δεύτερου είδους αιτήματος-φύλακα). Ο αριθμός των φυλάκων είναι πάλι O(n) (αν τοποθετήσουμε ένα φύλακα σε κάθε βόρεια και νότια ακμή προφανώς μπορούμε να εποπτεύσουμε όλο το πολύγωνο).

Όπως είχαμε πει, τα σκέλη αποθηκεύονται σε ένα ισοζυγισμένο δυαδικό δέντρο αναζήτησης. Επομένως, επειδή το πλήθος των σκελών είναι O(n), τόσο είναι και το μέγεθος του δέντρου. Κάθε λειτουργία εισαγωγής, διαγραφής και αναζήτησης χρησιμοποιεί O(logn) χρόνου. Επίσης, έχουμε επισημάνει στην Παρατήρηση 3.1 ότι τα εύρη ανάθεσης των αιτημάτων φυλάκων δεν τέμνονται μεταξύ τους. Έτσι φυλάσσοντάς τα σε μία διπλά συνδεδεμένη λίστα (με δείκτες και στα δύο άκρα) μας επιτρέπεται η διάσχισή τους και από τα δύο άκρα καθώς επίσης η συγχώνευσή τους χρησιμοποιεί σταθερό χρόνο.

Το Βήμα 1 του αλγορίθμου προφανώς χρησιμοποιεί O(nlogn) χρόνο. Στο Βήμα 2 για κάθε νότια ακμή πρέπει να εντοπίσουμε την ακμή στο αντίστοιχο σκέλος πραγματοποιώντας το πολύ μια εισαγωγή και το πολύ δύο διαγραφές σκελών. Επίσης, χρειάζεται να ενημερώσουμε και τις κατάλληλες πληροφορίες στο αντίστοιχο σκέλος. Αυτά απαιτούν O(logn) συνολικό χρόνο για κάθε ακμή (O(logn) χρόνο για την εισαγωγή, διαγραφή και αναζήτηση και σταθερό χρόνο για τις ενημερώσεις της δομής).

Ας δούμε τώρα την επεξεργασία μιας βόρειας ακμής στο Βήμα 2. Η αναζήτηση του σκέλους T στο οποίο προσπίπτει η ακμή e μπορεί να γίνει σε O(logn) χρόνο χρησιμοποιώντας τη δομή D(t). Επειδή γνωρίζουμε ότι η e βρίσκεται είτε στο αριστερό είτε στο δεξί μέρος του συνόρου του σκέλους, μπορούμε να βρούμε τα αιτήματα φύλακες που μας ενδιαφέρουν ακριβώς διασχίζοντας τη διπλά συνδεδεμένη λίστα των αιτημάτων φυλάκων είτε από αριστερά είτε από δεξιά αντίστοιχα (θυμηθείτε το Λήμμα 2.1.(ii)). Για κάθε τέτοιο αίτημα φύλακα ξοδεύουμε

O(n) χρόνο για βρούμε ένα διαθέσιμο φύλακα και να ενημερώσουμε τα καινούρια εύρη του. Το αίτημα φύλακα απομακρύνεται και δε θα χρειαστεί να το χειριστούμε ξανά. Δεν πρέπει να ξεχνάμε όμως και την περίπτωση όπου θα χρειαστεί ένα αίτημα φύλακα-δεύτερου είδους. Σε αυτή την περίπτωση χρειαζόμαστε O(n) χρόνο για να ελέγξουμε αν κάποιος φύλακας εποπτεύει το ευθ. τμήμα τομής λίγο ψηλότερα από την e (δες Σχήμα 2.2.3) και σταθερό χρόνο για να εισάγουμε το καινούριο αίτημα (ή αιτήματα στη περίπτωση που πρέπει να προσθέσουμε δύο) φύλακα στη δομή των αιτημάτων φυλάκων του T. Η επεξεργασία ενός φύλακα του οποίου το εύρος θέσης υπερκαλύπτεται από το x-εύρος της e παίρνει σταθερό χρόνο για κάθε φύλακα και συμβαίνει μόνο μία φορά για κάθε φύλακα. Τέλος, η περικοπή των ευρών των φυλάκων και των ευρών-τοποθέτησης των αιτημάτων φυλάκων (όποτε χρειάζεται) μπορεί να γίνει προφανώς σε O(n) χρόνο. Συνοψίζοντας, αν αγνοήσουμε το χρόνο που χρειάζεται για να επεξεργαστούμε τα αιτήματα φυλάκων των οποίων το εύρος ανάθεσης είναι υποσύνολο του x-εύρους της e και το χρόνο που χρειάζεται για να επεξεργαστούμε τους φύλακες των οποίων το εύρος θέσης είναι υποσύνολο του x-εύρους της e, η επεξεργασία κάθε βόρειας ακμής χρειάζεται O(n) χρόνο.

Έτσι, επειδή το εύρος ανάθεσης των αιτημάτων φυλάκων και το εύρος θέσης των φυλάκων θα είναι υποσύνολο του x-εύρους μιας βόρειας ακμής ακριβώς μια φορά, το Βήμα 2 χρειάζεται $O(n^2)$ χρόνο. Το βήμα 3 προφανώς χρειάζεται O(n) χρόνο, συνεπώς:

Θεώρημα 2.3.1 Έστω P ένα απλό ορθογώνιο πολύγωνο κλάσης 3 με n κορυφές. Ο αλγόριθμος Class 3_rStar_Cover υπολογίζει ένα ελάχιστο σύνολο από r-ορατούς φύλακες σε $O(n^2)$ χρόνο και O(n) χώρο.

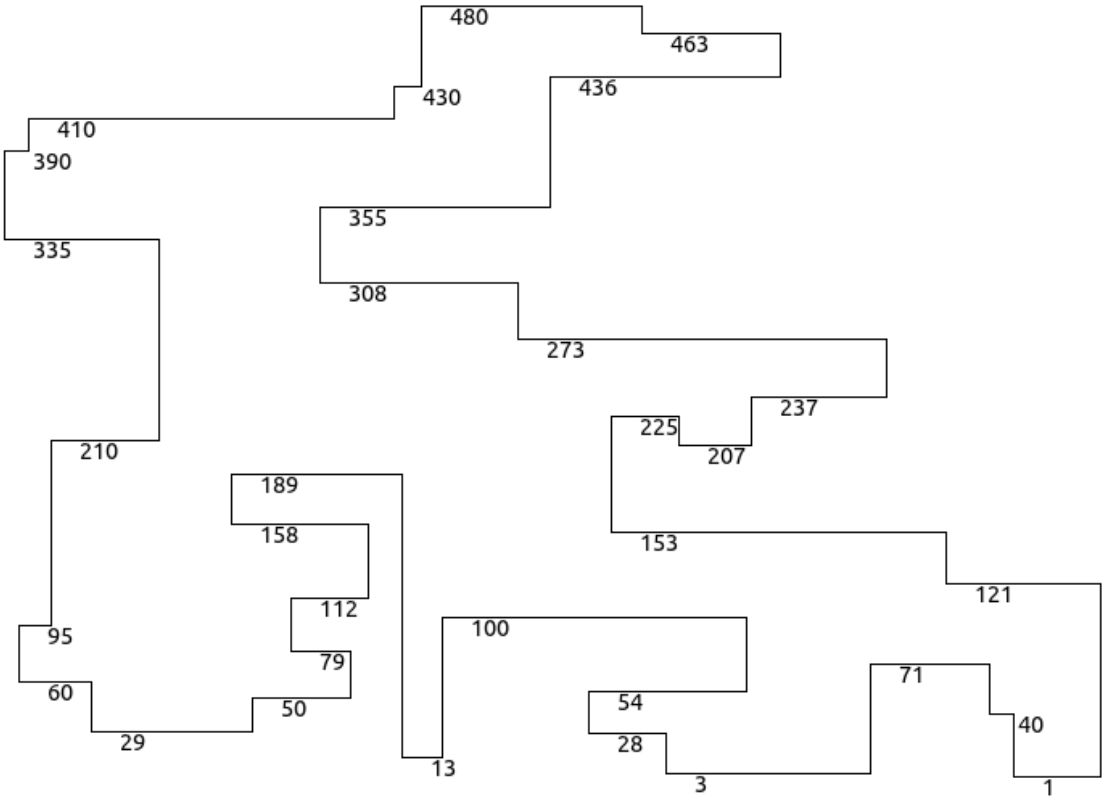
Σημειώνουμε ότι με χρήση κατάλληλων δομών δεδομένων μπορούμε σε σταθερό χρόνο να βρίσκουμε το χαμηλότερο φύλακα για ένα αίτημα φύλακα καθώς επίσης μπορούμε σε σταθερό χρόνο να ελέγξουμε αν υπάρχει ανάγκη για ένα αίτημα φύλακα δευτέρου είδους. Δυστυχώς η περικοπή των ευρών των φυλάκων και των ευρών τοποθέτησης των αιτημάτων φυλάκων (όποτε αυτή χρειάζεται) είναι δυσκολότερο να επιταχυνθεί (σε σύγκριση με το O(n)). Αυτό που χρειάζεται είναι μια δομή δεδομένων που αρχικά θα κάνει την περικοπή με έμμεσο τρόπο και είτε θα είναι συγχωνεύσιμη (επειδή και τα σκέλη συγχωνεύονται) είτε θα επιτρέπει την

έμμεση περικοπή ενός συνδεδεμένου υποσυνόλου των αποθηκευμένων διαστημάτων.

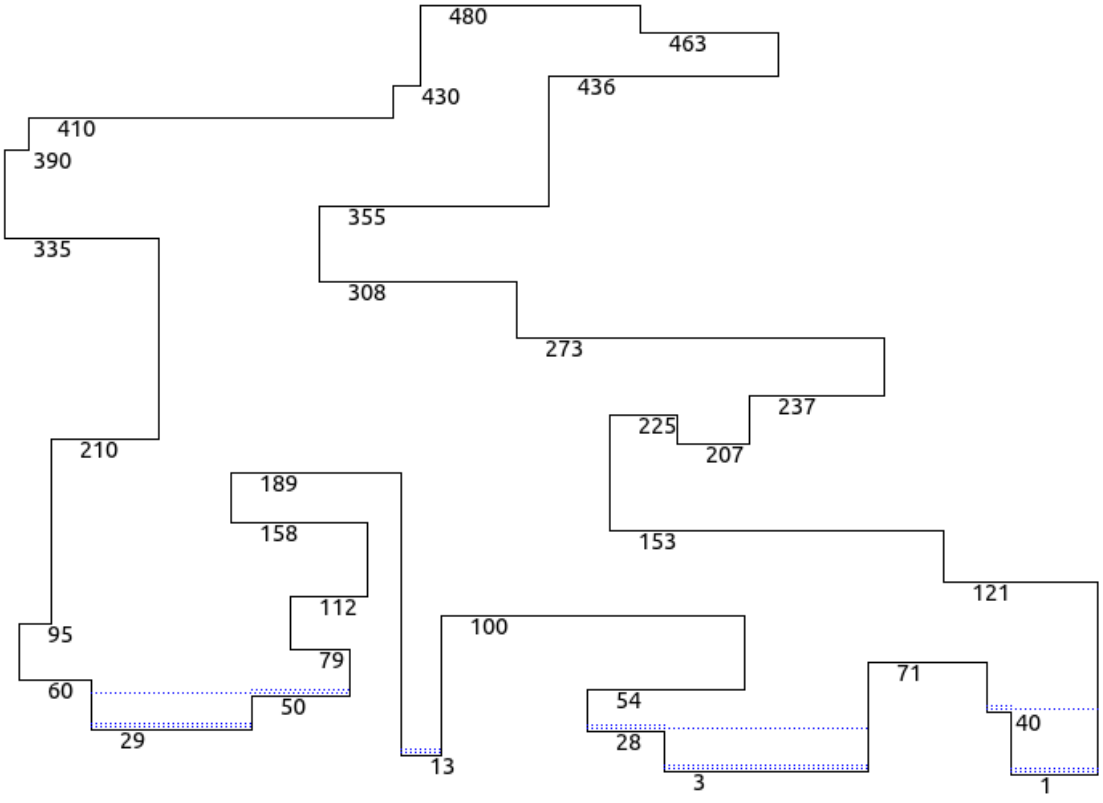
2.4 Παράδειγμα εφαρμογής του αλγορίθμου

Παρακάτω θα ακολουθήσει η εφαρμογή του αλγορίθμου Class3_rStar_Cover(P) με τη βοήθεια σχημάτων.

Στο Σχήμα 2.4.1 βλέπουμε ένα ορθογώνιο πολύγωνο κλάσης 3 χωρίς βόρειες εσοχές. Οι αριθμοί που βρίσκονται κάτω από τις οριζόντιες ακμές ισοδυναμούν με την y-συντεταγμένη τους, ή αλλιώς τη στάθμη που έχουμε αναφέρει πιο πάνω. Κατά την περιγραφή της εφαρμογής του αλγορίθμου θα χρησιμοποιούμε αυτό τον αριθμό ως χαρακτηριστικό όνομα αυτών των ακμών.

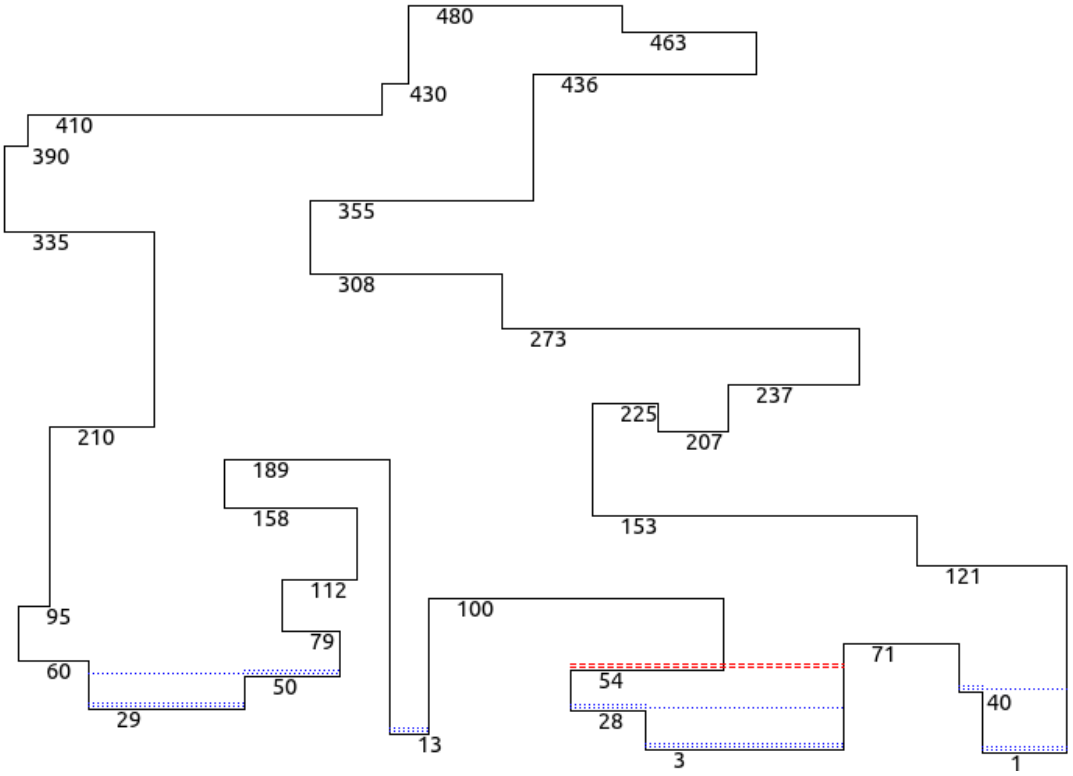


Σχήμα 2.4.1



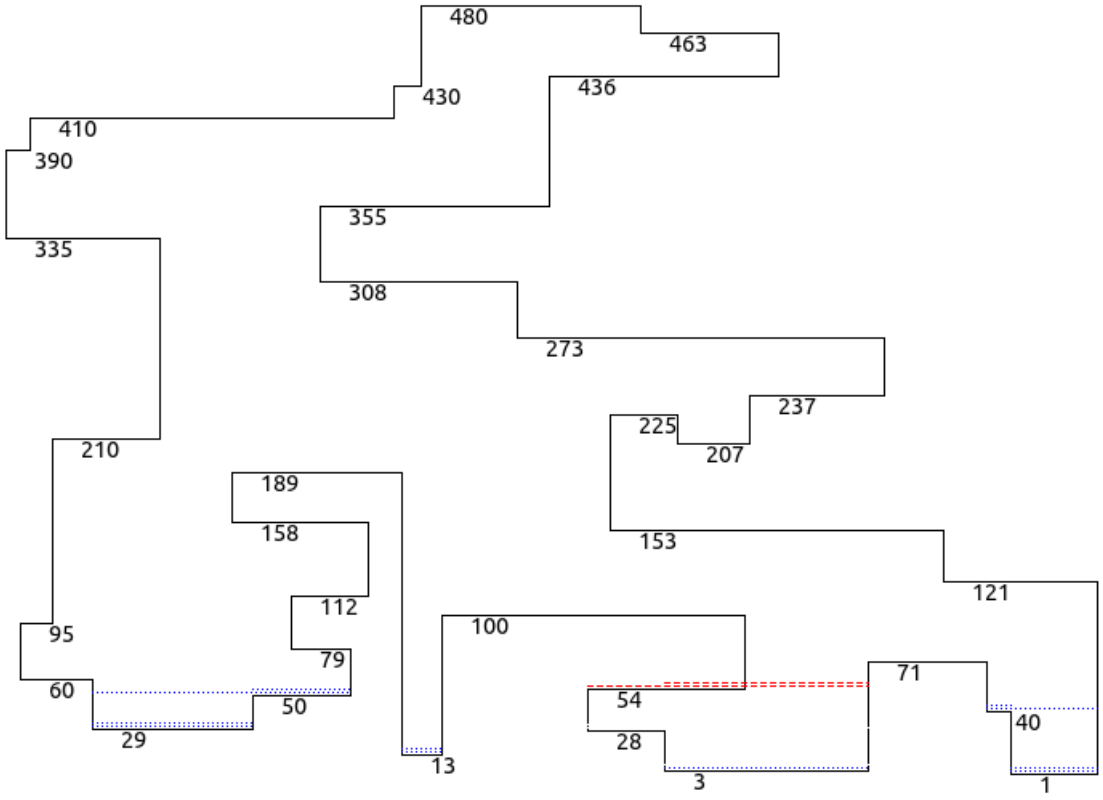
Σχήμα 2.4.2

Στο Σχήμα 2.4.2 βρισκόμαστε ακριβώς πριν την επεξεργασία της Βόρειας ακμής 54. Μέχρι στιγμής έχουμε συναντήσει μόνο νότιες ακμές και έχουμε δημιουργήσει τα αντίστοιχα αιτήματα φυλάκων τα οποία και τα συμβολίζουμε με τις διπλές μπλε διακεκομμένες γραμμές. Στο Σχήμα 2.4.2 το εύρος ανάθεσης είναι ακριβώς πάνω από το εύρος τοποθέτησης του αιτήματος φύλακα (Υπενθυμίζεται ότι το εύρος ανάθεσης είναι πάντα υποσύνολο του εύρους τοποθέτησης).



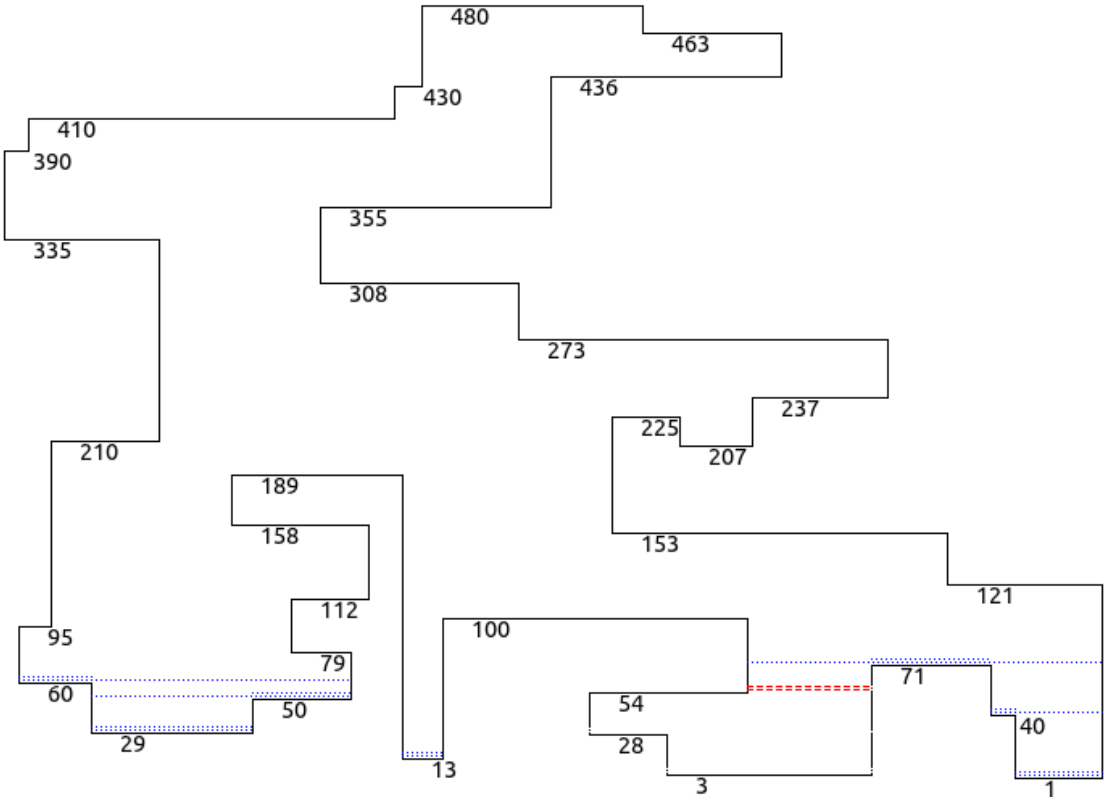
Σχήμα 2.4.3

Στο Σχήμα 2.4.3 έχουμε αρχίσει να επεξεργαζόμαστε την Βόρεια ακμή 54 και παρατηρούμε ότι αυτή τέμνει ένα εύρος ανάθεσης (αυτό της ακμής 28). Αναζητούμε λοιπόν ένα ελεύθερο φύλακα κατάλληλο να επιτηρήσει την ακμή 28 αλλά δε βρίσκουμε για αυτό και δημιουργείται ένας καινούριος φύλακα. (Ο φύλακας συμβολίζεται με δύο κόκκινες γραμμές, η υψηλότερη συμβολίζει το εύρος θέσης ενώ η χαμηλότερη το εύρος επόπτευσης του φύλακα). Παρατηρούμε λοιπόν ότι ο φύλακας στη στάθμη της ακμής 54 αρχικοποιείται με εύρος θέσης ίσο με το εύρος τοποθέτησης της ακμής 28 και εύρος επόπτευσης το ευθ. τμήμα τομής του σκέλους (εδώ ταυτίζονται).



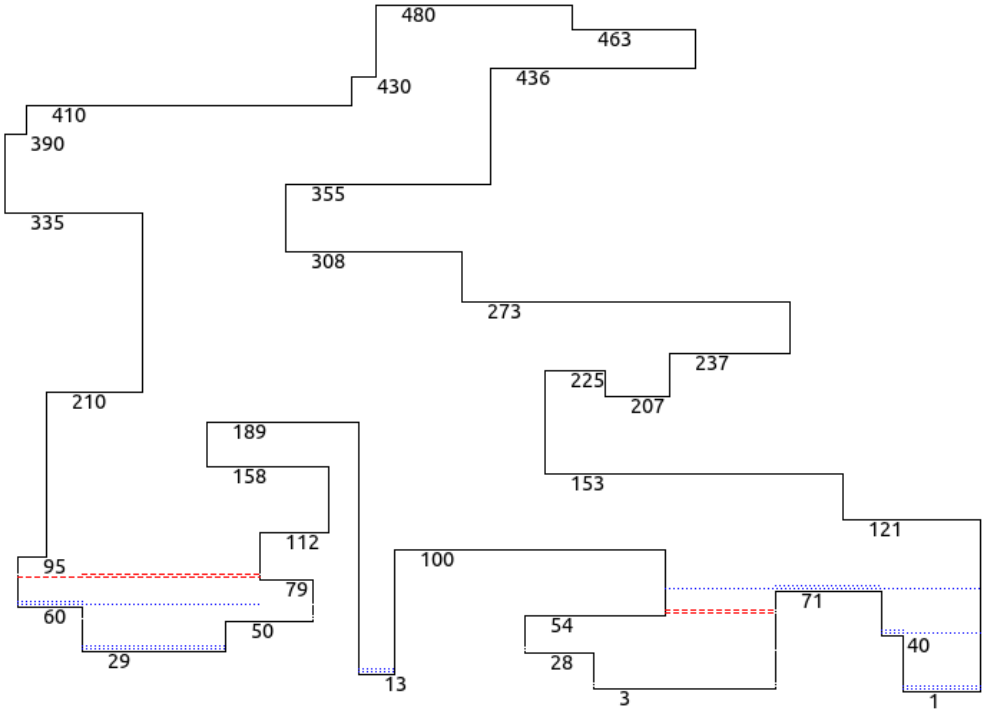
Σχήμα 2.4.4

Αφού βρήκαμε (δημιουργήσαμε στη συγκεκριμένη περίπτωση) κάποιον φύλακα και τον τροποποιήσαμε κατάλληλα για να είναι ικανός να εποπτεύει την ακμή 28 ο έλεγχος συνεχίζεται για να δούμε μήπως και κάποιο άλλο εύρος ανάθεσης τέμνεται. Σημειώνεται ότι επειδή η Βόρεια ακμή 54 έρχεται από αριστερά προς τα δεξιά και εμείς ελέγχουμε τα εύρη ανάθεσης από αριστερά προς τα δεξιά. Πράγματι η ακμή 54 τέμνει και το εύρος ανάθεσης της ακμής 3. Αυτή την φορά βρίσκουμε κάποιον ελεύθερο φύλακα (αυτόν που δημιουργήσαμε αμέσως πριν στη στάθμη 54) και τροποποιούμε το εύρος θέσης να είναι ίσο με το εύρος τοποθέτησης της ακμής 3. Το εύρος επόπτευσης δεν αλλάζει.



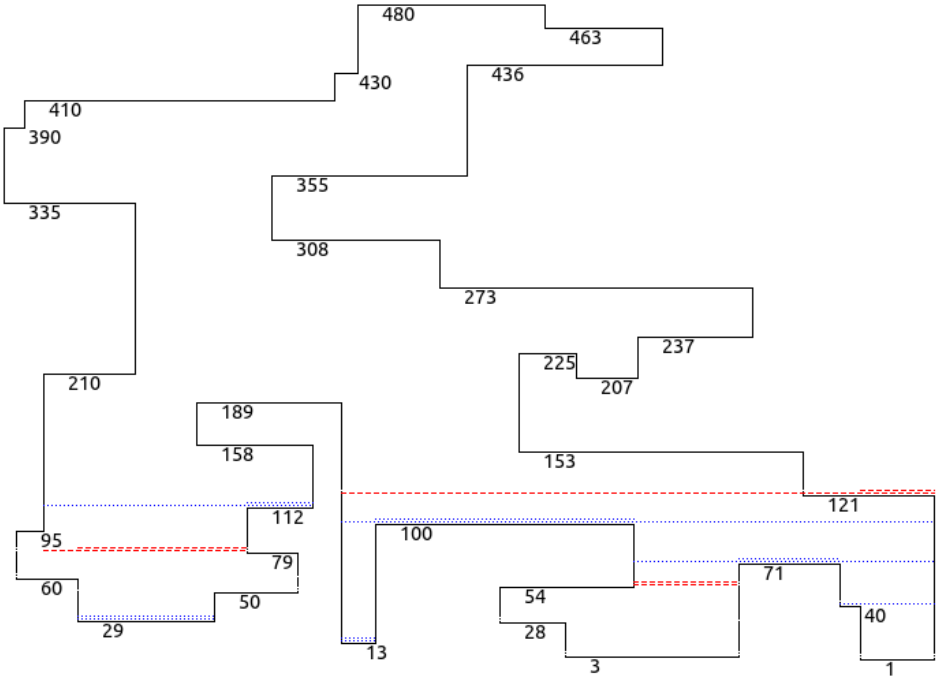
Σχήμα 2.4.5

Αφού τροποποιήσαμε τον φύλακα 54 έτσι ώστε να μπορεί να επιβλέπει την ακμή 3 διαγράψαμε και το αίτημα φύλακα της ακμής 3. Η ακμή 54 δε τέμνει άλλο εύρος ανάθεσης οπότε κοιτάμε τι γίνεται με την τομή αυτής με τους διαθέσιμους φύλακες. Έτσι περικόπτουμε το εύρος θέσης αλλά και επόπτευσης του φύλακα 54 ώστε ο διαθέσιμος φύλακας να μπορεί να επιβλέπει και το πολύγωνο σε επίπεδα πιο ψηλά. Η ακμή 54 δεν τέμνει κάποιο εύρος τοποθέτησης κάποιου αιτήματος φύλακα οπότε δεν κάνουμε τίποτα ως προς αυτόν τον τομέα. Ακολούθως έρχεται η Νότια ακμή 71. Συνενώνουμε εδώ το αριστερό και το δεξί σκέλος σε ένα. Επιπλέον συνενώνουμε τα αιτήματα φύλακα από το αριστερό και τα δεξί της σκέλος καθώς και τους ελεύθερους φύλακες.



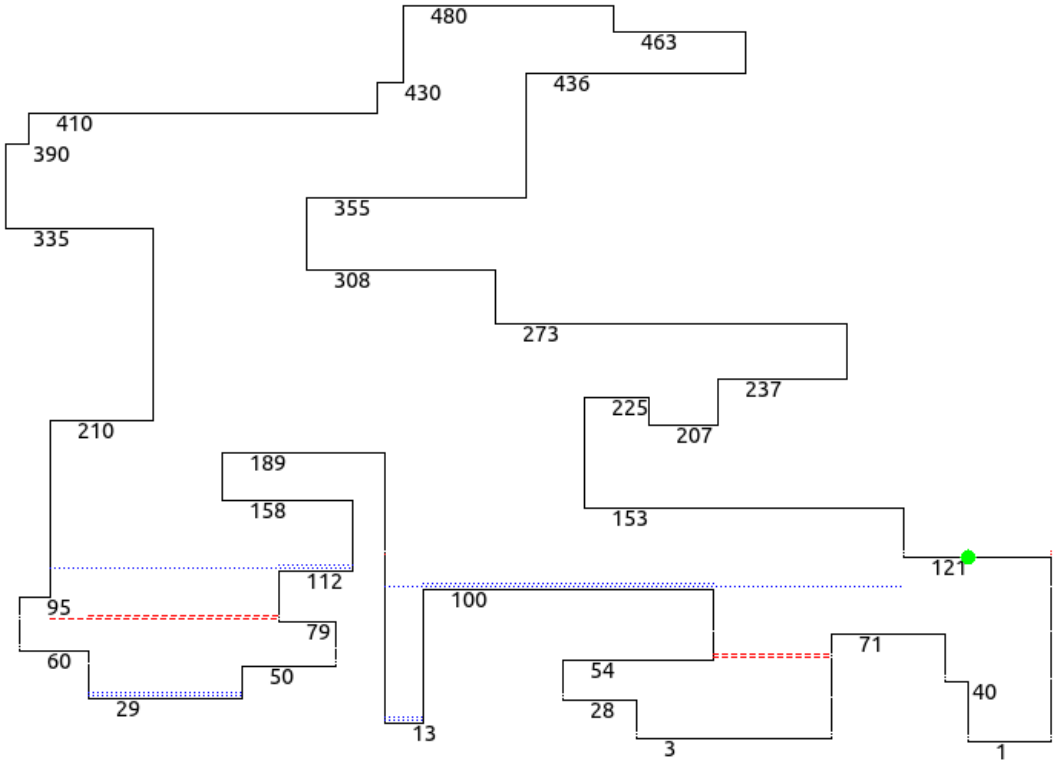
Σχήμα 2.4.6

Στο Σχήμα 2.4.6 έχει τελειώσει η επεξεργασία της Βόρειας ακμής 79. Επειδή τεμνόταν με το εύρος ανάθεσης της ακμής 50 πήρε το ανάλογο εύρος θέσης (ίσο με το εύρος τοποθέτησης της ακμής 50 δηλαδή). Αφού περιέκοψε το εύρος θέσης και επόπτευσης του φύλακα περικόπτει και το εύρος τοποθέτησης της ακμής 60.



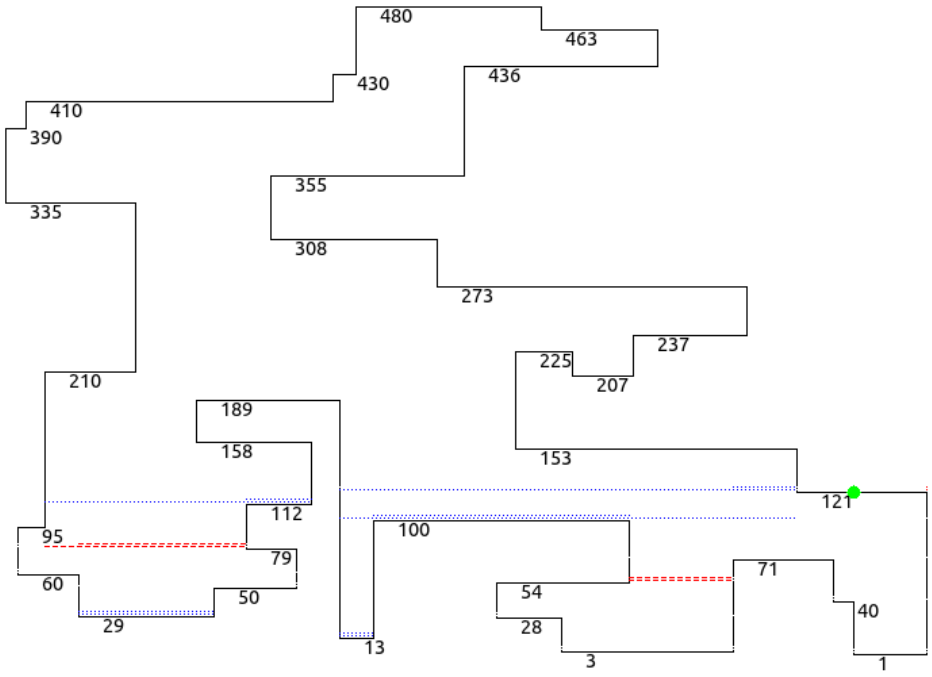
Σχήμα 2.4.7

Εδώ παρατηρούμε την ακμή 95 να έχει βρει φύλακα για την ακμή 60 της οποίας έτεμνε το εύρος ανάθεσης και ακολούθως περιέκοψε το εύρος επόπτευσης του φύλακα στη στάθμη 79. Έπειτα βλέπουμε την ακμή 121 να τέμνει από δεξιά προς τα αριστερά το εύρος ανάθεσης των ακμών 1, 40, 71. Έτσι λοιπόν αφού δημιούργησε έναν ελεύθερο φύλακα για την ακμή 1 με εύρος θέσης ίσο με το εύρος τοποθέτησης της ακμής 1 διέγραψε και τα αιτήματα φύλακα των ακμών 40 και 71. (Σχήμα 2.4.8)



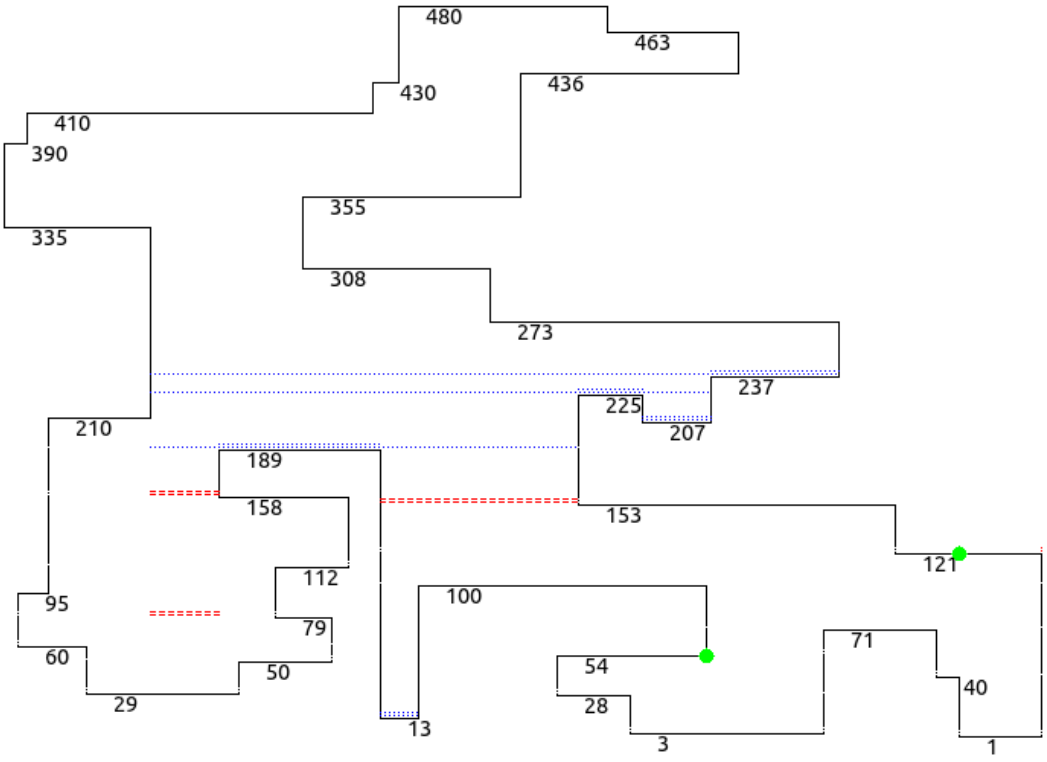
Σχήμα 2.4.8

Έπειτα επειδή υπερκάλυπτε το εύρος θέσης του φύλακα 121 τοποθετεί οριστικά φύλακα στην πράσινη κουκίδα και τον αφαιρεί από τη λίστα με τους φύλακες στο σύνολο Διαθέσιμοι φύλακες. Έπειτα περιορίζει το εύρος τοποθέτησης της ακμής 100.



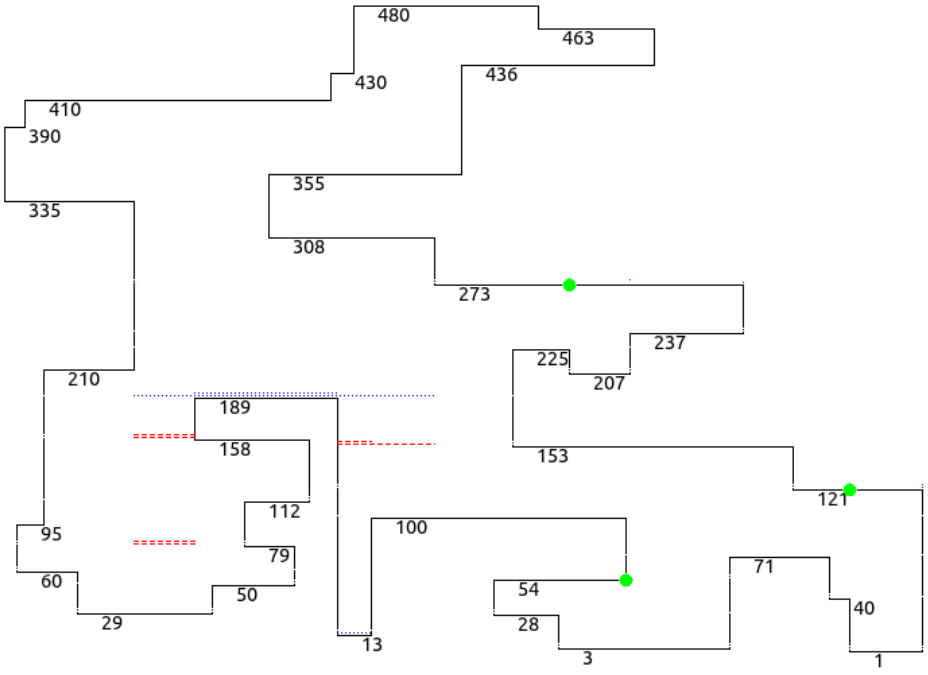
Σχήμα 2.4.9

Μέχρι στιγμής όταν ερχόταν κάποια βόρεια το ευθ. τμήμα τομής του σκέλους που έμενε (χωρίς τη βόρεια) εποπτευόταν. Αυτό δεν ισχύει στην περίπτωση της ακμής 121. Έτσι ο έλεγχος που γίνεται στα δύο άκρα του ευθ. τμήματος τομής – η ακμή 121 μας οδηγεί στο να προσθέσουμε ένα καινούριο αίτημα φύλακα στη στάθμη 121 όπως αυτός φαίνεται το Σχήμα 2.4.9. Συγκεκριμένα παρατηρούμε ότι το αριστερό άκρο αυτής της διαφοράς ούτε επιβλέπεται από κάποιο φύλακα πιο χαμηλά αλλά ούτε υπάρχει κάποιο εύρος ανάθεσης με αυτό ως άκρο.

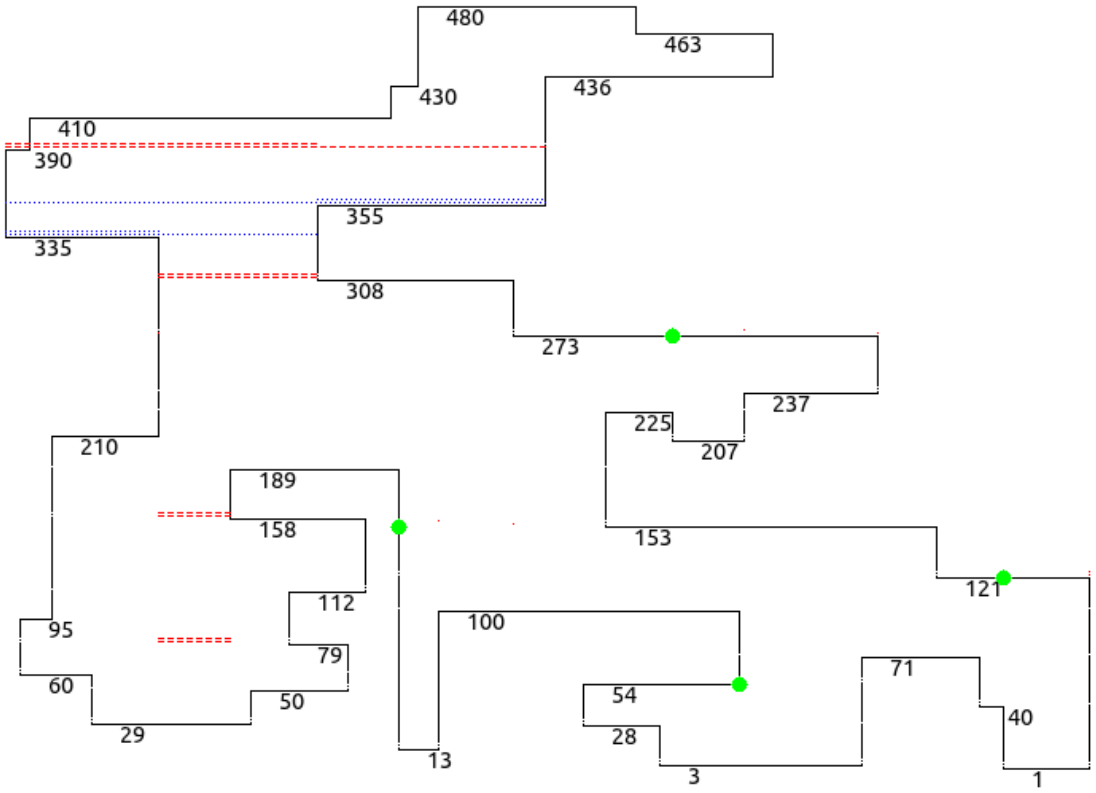


Σχήμα 2.4.10

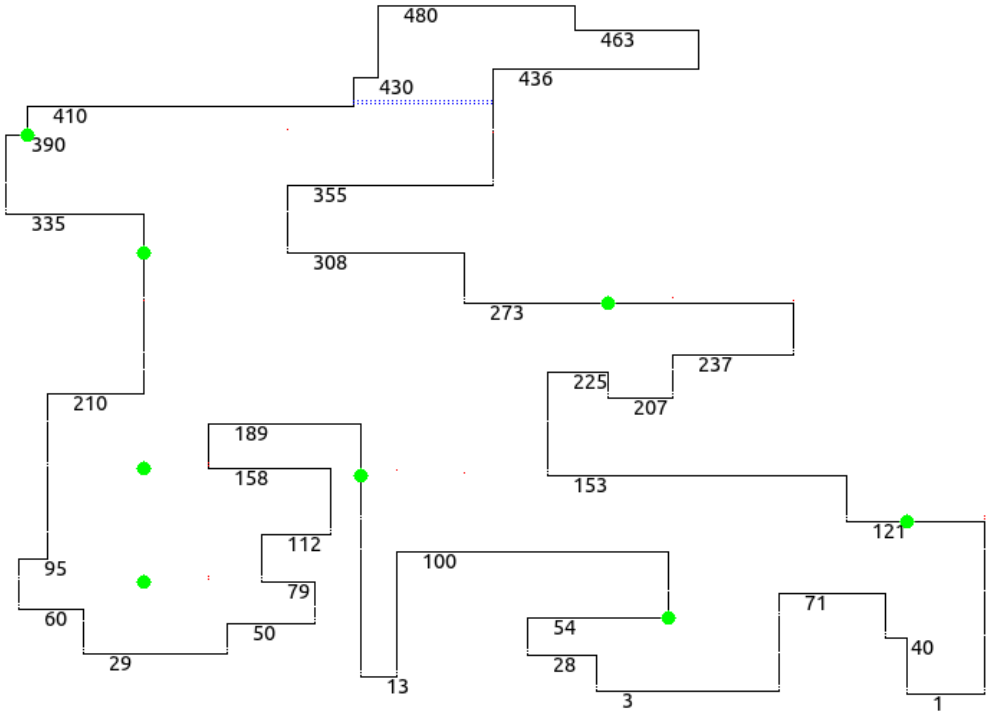
Στο Σχήμα 2.4.10 αξίζει να δούμε ότι όταν ήρθε η ακμή 210, επειδή επιβλεπόταν το αριστερό και το δεξί άκρο του σκέλους δε δημιουργήσαμε καινούριο αίτημα φύλακα.



Σχήμα 2.4.11

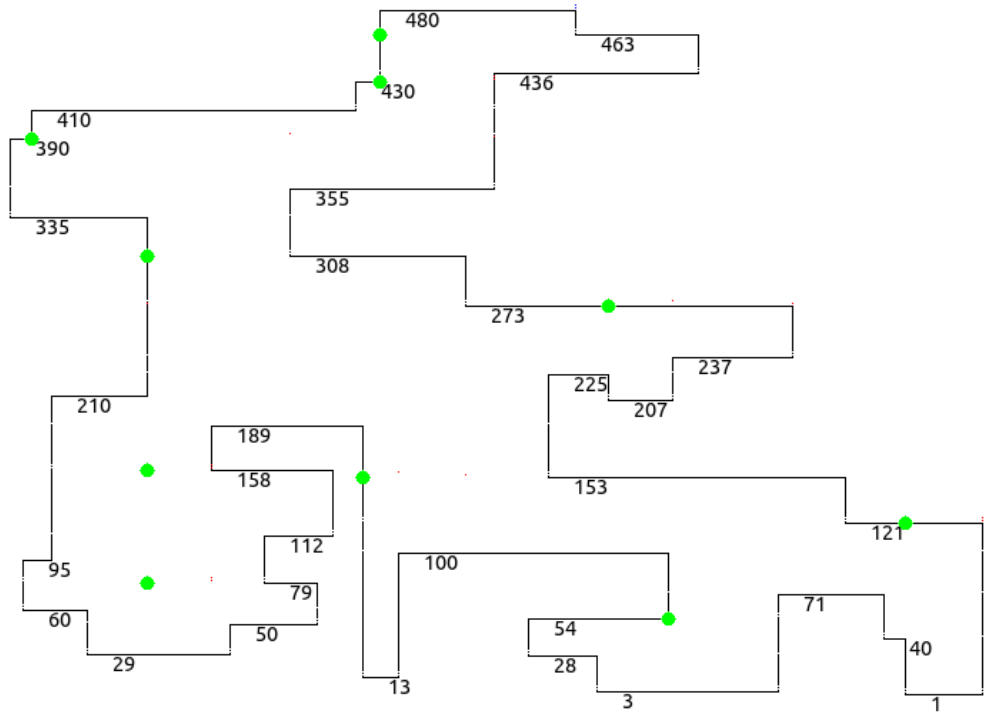


Σχήμα 2.4.12



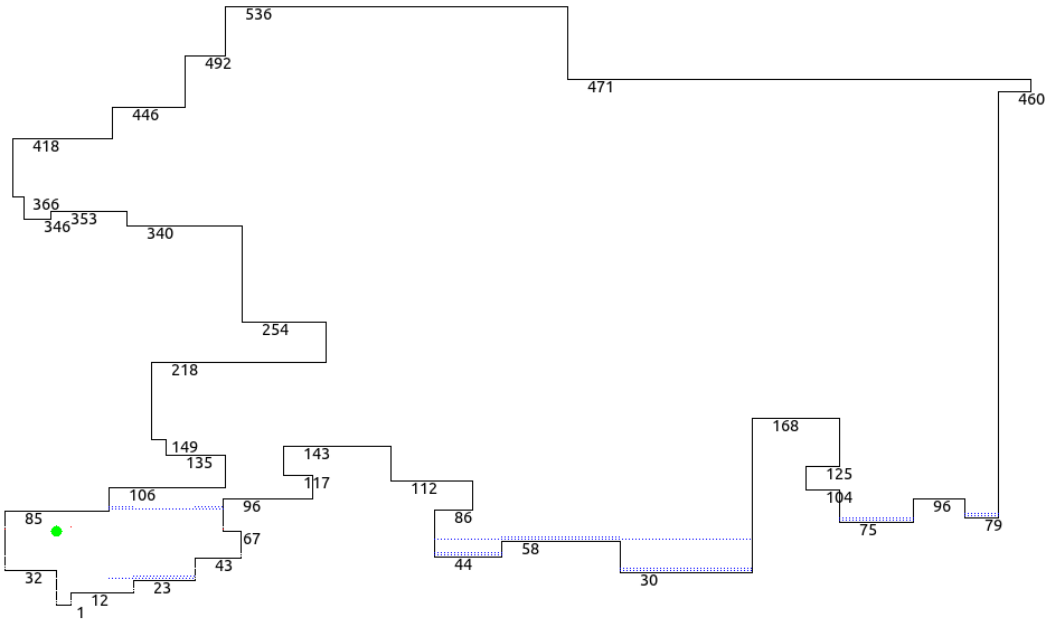
Σχήμα 2.4.13

Στο Σχήμα 2.4.13 φαίνεται ξεκάθαρα ότι αν δε δημιουργούσαμε ένα καινούριο αίτημα φύλακα στη στάθμη 410 θα οδηγούμαστε σε λάθος (δείτε την περιοχή ακριβώς κάτω από την ακμή 430).



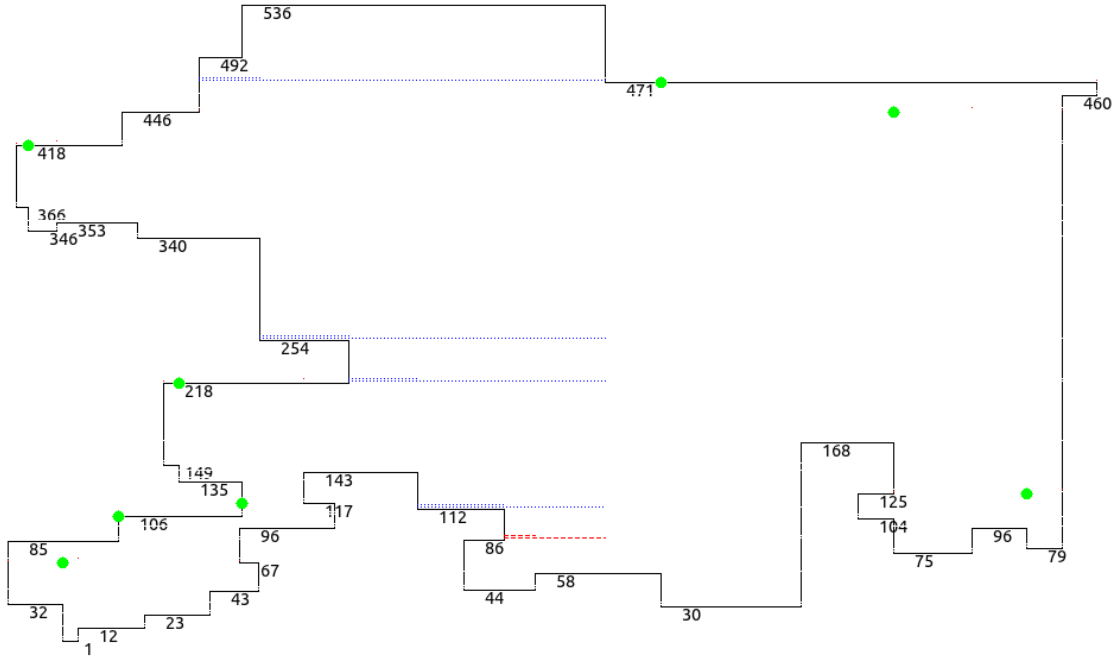
Σχήμα 2.4.14

Στο Σχήμα 2.4.14 βλέπουμε το ορθογώνιο πολύγωνο πλήρως εποπτευμένο.



Σχήμα 2.4.15

Στο Σχήμα 2.4.15 παρατηρούμε ένα διαφορετικό ορθογώνιο πολύγωνο κλάσης 3. Αξιοσημείωτο σε αυτό είναι όταν χρειάζεται να βάλουμε ένα καινούριο αίτημα φύλακα δευτέρου είδους στη στάθμη 85. Παρατηρούμε ότι τελικά χρειάζεται να βάλουμε δύο καινούρια αιτήματα φύλακα όπως φαίνονται στο σχήμα.



Σχήμα 2.4.16

Στο Σχήμα 2.4.16 παρατηρούμε ένα επιπλέον αίτημα φύλακα 2^{ου} είδους στη στάθμη 471.

3. Η υλοποίηση

Σε αυτό το κεφάλαιο θα περιγράψουμε τα εργαλεία, τι δομές που χρησιμοποιήσαμε καθώς και τις συναρτήσεις που υλοποιήσαμε για την επίλυση του προβλήματος. Να σημειωθεί ότι εκτός από την υλοποίηση του αλγορίθμου έχει πραγματοποιηθεί και οπτικοποίησή του με δυνατότητα σχεδιασμού καινούριων παραδειγμάτων.

Οι συναρτήσεις των αλγορίθμων είναι υλοποιημένες σε C++. Η οπτικοποίηση του αλγορίθμου έχει γίνει με την βοήθεια της βιβλιοθήκης της Qt.

3.1 Είσοδος – Έξοδος

Ως είσοδο, χρησιμοποιώντας το γραφικό περιβάλλον που έχει υλοποιηθεί σε Qt, παίρνουμε οποιοδήποτε απλό ορθογώνιο πολύγωνο. Ο χρήστης απλά ζωγραφίζει την αλληλουχία των κορυφών που σχηματίζει το πολύγωνο. Ιδιαίτερη προσοχή πρέπει να δοθεί ότι για να δουλέψει σωστά ο αλγόριθμος θα πρέπει το ζωγραφισμένο ορθογώνιο πολύγωνο να είναι κλάσης 3 και μάλιστα να μην έχει βόρειες εσοχές. Επίσης δίνεται η δυνατότητα αποθήκευσης των παραδειγμάτων για αναπαραγωγή τους στο μέλλον. Οι συναρτήσεις είναι σε C++ και δε θα παρουσιαστούν εδώ. Πάντως με το που τελειώσει η σχεδίαση γίνεται ταξινόμηση

των οριζόντιων ακμών ως προς το ύψος τους και καταλήγουμε να έχουμε τον πίνακα sorted[ ] όπου περιέχει τις ταξινομημένες ακμές.

3.2 Δομές Δεδομένων

- Το struct που χρησιμοποιούμε για την αποθήκευση των οριζοντίων ακμών:

```
typedef struct h_edge
{
      int  xl,  xr, level;
      int eidos;
}edge;
```

- Το struct που χρησιμοποιούμε για τους φύλακες:

```
typedef struct guard
{
    int level,visibilityl,visibilityr,thesil,thesir;
    struct guard *next;
    struct guard *prev;
}guard;
```

Οι δύο δείκτες χρησιμοποιούνται για τη δημιουργία και τη συντήρηση μίας διπλά συνδεδεμένης λίστας ελεύθερων φυλάκων.

- Το struct που χρησιμοποιούμε για τα αιτήματα φύλακα:

```
typedef struct g_req
{
    int forcingl,forcingr,thesil,thesir,level;
    struct g_req *next;
    struct g_req *prev;
}g_req;
```

Οι δείκτες next και prev χρησιμοποιούνται για τη διπλά συνδεδεμένη λίστα των αιτημάτων φυλάκων.

- Το struct που χρησιμοποιούμε για τα σκέλη:

```
typedef struct nodeofatree
{
      int  xl,  xr, lastlevel;
      g_req *g_req_front;
      g_req *g_req_end;
      struct nodeofatree *prev;
      struct nodeofatree *next;
      guard * g_front;
      guard * g_end;
} node;
```

Το xl και το xr μας δείχνουν τα άκρα ενός σκέλους, οι δείκτες αιτημάτων φυλάκων και φυλάκων μας δείχνουν την αρχή και το τέλος της αντίστοιχης

λίστας ενώ ο next κι ο prev χρησιμοποιούνται για τη δομή των σκελών. Εδώ σημειώνεται ότι για λόγους απλότητας της κωδικοποίησης δε χρησιμοποιήθηκε ένα ισοζυγισμένο δυαδικό δέντρο αναζήτησης αλλά μία διπλά συνδεδεμένη λίστα.

- `edge sorted[N];`

  Περιέχει ταξινομημένες της οριζόντιες ακμές ως προς το ύψος τους.

- `QPoint fylakes[N];`

  Εδώ αποθηκεύονται οι τοποθετημένοι φύλακες. Σημειώνεται ότι ο τύπος QPoint είναι τύπος της Qt και είναι της μορφής (int x,int y)

- `node * start;`

  Είναι ο δείκτης που μας δείχνει ποιο είναι το αριστερότερο σκέλος.

3.3 Συναρτήσεις

3.3.1 void Class3_rStar_Cover (edge sorted[ ],int k)

Αυτός είναι ο βασικός βρόγχος του προγράμματος. Παίρνει ως όρισμα τον πίνακα sorted[] όπου περιέχει ταξινομημένες τις οριζόντιες ακμές του πολυγώνου και τον ακέραιο k που είναι το πλήθος των ακμών. Εδώ θα δούμε πως εισάγονται καινούρια σκέλη, καινούρια αιτήματα φύλακα και πως γίνονται οι διάφορες συνενώσεις όταν έρχονται οι νότιες εσοχές.

```
#define LeftChain -1
#define RightChain 1
void ScribbleArea::sd(int k)
{//k einai o arithmos twn akmwn
/*****Arxikopoiisi*****/
    int i;
    int chain;
    addednode=newnode();
    addednode->xl=sorted[0].xl;
    addednode->xr=sorted[0].xr;
    addednode->eidos=0; //H prwti einai south
    addednode->lastlevel=sorted[0].level;
    addg_req(addednode,sorted[0].xl,sorted[0].xr,LeftChain);
```

```
    start=addednode;
    end =addednode;
    cursor=start;
    for(i=1;i<k;i++)
    {
        cursor=start;
/*****erxetai notia*****/
        if(sorted[i].eidos==0)
        {
            while(cursor!=NULL)
            {
                if(sorted[i].xr==cursor->xl)//ayksanetai apo aristera
                {
                    cursor->xl=sorted[i].xl;
                    cursor->lastlevel=sorted[i].level;
                    cursor->eidos=0;
                    addg_req(cursor,sorted[i].xl,sorted[i].xr,
                    LeftChain);
                    break;
                }
                else if(sorted[i].xl==cursor->xr)
                   //ayksanetai apo deksia
                {
                    cursor->xr=sorted[i].xr;
                    cursor->lastlevel=sorted[i].level;
                    cursor->eidos=0;
                    addg_req(cursor,sorted[i].xl,sorted[i].xr,
                    RightChain);
                    break;
                }
                else if(sorted[i].xr<cursor->xl)
                {
                    addednode=newnode();
                    addednode->xl=sorted[i].xl;
                    addednode->xr=sorted[i].xr;
                    if(cursor!=start)
                    {
                        addednode->prev=cursor->prev;
                        cursor->prev->next=addednode;
                    }
                    else
                       start=addednode;
                    cursor->prev=addednode;
                    addednode->next=cursor;
                    addednode->eidos=0;
                    addednode->lastlevel=sorted[i].level;
                    addg_req(addednode,sorted[i].xl,sorted[i].xr,
                    LeftChain);
                    break;
                }
                cursor=cursor->next;
            }
            if(sorted[i].xl>end->xr)
              //simainei oti mpanei sto telos kainoyrio
            {
                addednode=newnode();
                addednode->xl=sorted[i].xl;
                addednode->xr=sorted[i].xr;
                addednode->eidos=0;
                addednode->lastlevel=sorted[i].level;
                addg_req(addednode,sorted[i].xl,sorted[i].xr,
                LeftChain);
```

```
                end->next=addednode;
                addednode->prev=end;
                end=addednode;
            }
        }
        /*****erxetai voreia*****/
        else if(sorted[i].eidos==1)
        {
            cursor=start;
            while(cursor!=NULL)
            {
                if(sorted[i].xl==cursor->xl)
                {
                    chain=LeftChain;
                    break;
                }
                else if(sorted[i].xr==cursor->xr)
                {
                    chain=RightChain;
                    break;
                }
                 cursor=cursor->next;
            }
             erxetaivoreia(sorted[i],cursor,chain);
        }
        /*****erxetai south dent*****/
        else if(sorted[i].eidos==2)
        {
            cursor=start->next;
          //sigoyra exoyme apo to deytero kombo kai meta
            while(cursor!=NULL)
            {
                if(sorted[i].xr==cursor->xl)
                {
                    addednode=newnode();
                    addednode->xl=cursor->prev->xl;
                    addednode->xr=cursor->xr;
                    addednode->eidos=2;
                    addednode->lastlevel=sorted[i].level;
                    /***synenwsi listas aitimatwn fylaka***/
                    addednode->g_req_front=cursor->prev->g_req_front;
                    addednode->g_req_end=cursor->prev->g_req_end;
                    addg_req(addednode,sorted[i].xl,sorted[i].xr,
                    RightChain);
//edw to right chain simainei na valei to kainoyrio g_req sto telos

                    if(cursor->g_req_front!=NULL)
                    {
                        cursor->g_req_front->prev=
                                            addednode->g_req_end;
                        addednode->g_req_end->next=
                                            cursor->g_req_front;
                       addednode->g_req_end=cursor->g_req_end;
                    }

                    /***synenwsi listas fylakwn***/
                    if(cursor->prev->g_front!=NULL)
                    {
                        addednode->g_front=cursor->prev->g_front;
                        addednode->g_end=cursor->prev->g_end;
                        if(cursor->g_front!=NULL)
                        {
```

```
                        cursor->prev->g_end->next=
                                                cursor->g_front;
                        cursor->g_front->prev=
                                             cursor->prev->g_end;
                        addednode->g_end=cursor->g_end;
                    }
                }
                else
                {
                     addednode->g_front=cursor->g_front;
                     addednode->g_end=cursor->g_end;
                }
                if(cursor->next==NULL)
                {
                    end=addednode;
                   if(cursor->prev->prev==NULL)
                      start=addednode;
                   else
                    {
                        cursor->prev->prev->next=addednode;
                        addednode->prev=cursor->prev->prev;
                    }
                }
                else if(cursor->prev->prev==NULL)
                {
                    addednode->next=cursor->next;
                    start=addednode;
                    cursor->next->prev=addednode;
                }
                else
                {
                    addednode->next=cursor->next;
                    addednode->prev=cursor->prev->prev;
                    cursor->prev->prev->next=addednode;
                    cursor->next->prev=addednode;
                }
                addednode->eidos=2;
                addednode->lastlevel=sorted[i].level;
                break;
            }
            cursor=cursor->next;
        }
    }
}
```

Στην παραπάνω συνάρτηση υπάρχει ο βασικός βρόγχος του προγράμματος. Εξετάζουμε τις οριζόντιες ακμές κατά αύξουσα σειρά της y-συντεταγμένης τους (έτσι βρίσκονται στον πίνακα sorted από προεργασία που έχει γίνει) και ανάλογα με το τι είδος είναι η κάθε ακμή κάνω διαφορετικά πράγματα. Μία ακμή μπορεί να είναι Βόρεια, Νότια ή Νότια Εσοχή (η Νότια Εσοχή είναι και αυτή Νότια ακμή μόνο που πρέπει να τη χειριστούμε λίγο διαφορετικά). Στη περίπτωση που είναι Νότια ακμή εξετάζουμε το αν προσπίπτει σε κάποιο από τα τρέχοντα σκέλη που έχουμε ήδη συναντήσει, διαφορετικά προσθέτουμε ένα καινούριο μπαντζάκι. Όπως και να

έχει σίγουρα προσθέτουμε ένα καινούριο αίτημα φύλακα για κάθε Νότια ακμή. Ειδικά στην περίπτωση που η Νότια ακμή είναι και Νότια Εσοχή, συγχωνεύουμε το σκέλος στα αριστερά της με το σκέλος στα δεξιά της. Επίσης συγχωνεύουμε τις δομές με τους διαθέσιμους φύλακες και τα αιτήματα φυλάκων. Στη περίπτωση που η οριζόντια ακμή είναι βόρεια εκτελούμε τη συνάρτηση void erxetaivoreia(edge voreia,node * cursor, int chain).

3.3.2 void erxetaivoreia(edge voreia, node * cursor, int chain)

Η συνάρτηση erxetaivoreia παίρνει ως όρισμα τη Βόρεια ακμή στην οποία αναφερόμαστε, ένα δείκτη cursor που δείχνει το σκέλος στο οποίο «ακούμπησε» η Βόρεια ακμή καθώς επίσης και έναν ακέραιο αριθμό που δείχνει αν η Βόρεια ακμή «ακουμπάει» στο αριστερό ή στο δεξί μέρος του σκέλους cursor.

```
void ScribbleArea::erxetaivoreia(edge voreia,node * cursor, int
chain)
{
   cursor->lastlevel=voreia.level;
   cursor->eidos=1;
   //elegxei an plireitai kapoio guard request
  isforced(cursor,voreia.xl,voreia.xr,chain);
  //clip-topothetisi guards
  closurevoreiawithguard(cursor,voreia);
  //clip mono p ranges
    closurevoreiawithprange(cursor,voreia.xl,voreia.xr);
    if(chain==LeftChain)
        cursor->xl=voreia.xr;
    else if(chain==RightChain)
        cursor->xr=voreia.xl;
//elegxos gia aitima fylaka 2ou  eidoys
elegxosgiakainoyriog_req(cursor);
}
```

Όταν έρχεται μία βόρεια ακμή κάνουμε με τη σειρά τα εξής:

i. Ελέγχουμε αν το x-εύρος της τέμνει κάποιο εύρος ανάθεσης και αν ναι ψάχνουμε να βρούμε κάποιον ελεύθερο φύλακα.
ii. Ελέγχουμε αν υπερκαλύφθηκε κάποιος ελεύθερος φύλακας οπότε και τοποθετούμε οριστικά έναν.
iii. Περικόπτουμε το εύρος επόπτευσης και το εύρος θέσης των φυλάκων όπου χρειάζεται.
iv. Περικόπτουμε τα εύρη τοποθέτησης των αιτημάτων-φυλάκων όπου πρέπει.

v. Ελέγχουμε για την ανάγκη δημιουργίας αιτημάτων φυλάκων 2ου είδους. Αν υπάρχει τέτοια ανάγκη προσθέτουμε το καινούριο αίτημα φύλακα (για την ακρίβεια μπορεί να χρειαστεί να προσθέσουμε μέχρι δύο 2ου είδους αιτήματα φύλακες) στη διπλά συνδεδεμένη λίστα με τα αιτήματα φυλάκων.

3.3.3 bool isforced (node * cursor, int xl, int xr, int meria)

Η συνάρτηση isforced παίρνει ως όρισμα τον δείκτη cursor που δείχνει σε ποιο σκέλος έχει «ακουμπήσει» η Βόρεια ακμή στην οποία αναφερόμαστε. Τα xl και xr είναι το αριστερό και το δεξί άκρο της Βόρειας ακμής ενώ η meria μας δείχνει από ποιο μέρος του σκέλους έχει «ακουμπήσει» η Βόρεια ακμή. Αυτή η συνάρτηση μας επιστρέφει true αν η Βόρεια ακμή «ακούμπησε» κάποιο εύρος ανάθεσης και false διαφορετικά.

```
bool  isforced(node * cursor,int xl,int xr,int meria)
{
    g_req *g_reqcursor;
    int flag=0;
    if(meria==LeftChain)
        g_reqcursor=cursor->g_req_front;
    else if(meria==RightChain)
        g_reqcursor=cursor->g_req_end;
    while(g_reqcursor!=NULL && flag==0)
    {
        if(meria==LeftChain)
        {
            if(xr>g_reqcursor->forcingl)
            //estw ki ligo na akoympisei mia force range
            {
                //psakse gia eleythero fylaka
                 if(!lookforfreeguard(cursor, cursor->g_front,
                                                 g_reqcursor))
                {
                    addguard(cursor);
                    cursor->g_end->thesil=g_reqcursor->thesil;
                    cursor->g_end->thesir=g_reqcursor->thesir;
                }
                delete_g_req(cursor,meria); //diegrapse to aitima
fylaka
                g_reqcursor=cursor->g_req_front;
            }
            else
               flag=1;
        }
        else if(meria==RightChain)
        {
            if(xl<g_reqcursor->forcing)
            {
                if(!lookforfreeguard(cursor,cursor->g_front,
                                                 g_reqcursor))
                {
```

```
                    addguard(cursor);
                    cursor->g_end->thesil=g_reqcursor->thesil;
                    cursor->g_end->thesir=g_reqcursor->thesir;
                }
                delete_g_req(cursor,meria);
                g_reqcursor=cursor->g_req_end;
            }
            else
               flag=1;
        }
    }
}
```

Αν η συνάρτησή μας εντοπίσει τομή της Βόρειας ακμής με κάποιο εύρος ανάθεσης ενός αιτήματος φύλακα καλεί τη συνάρτηση lookforfreeguard και ψάχνει να βρει αν υπάρχει κάποιος φύλακας στους διαθέσιμους που να είναι ικανός να εποπτεύσει αυτό το αίτημα φύλακα. Αν δεν υπάρχει τέτοιος φύλακας δημιουργεί έναν με τη συνάρτηση addguard(). Όπως και να έχει το αίτημα φύλακα για το οποίο μιλάμε διαγράφεται.

3.3.4 bool lookforfreeguard (node * cursor, guard * cursorg, g_req * theg_req)

Η συνάρτηση lookforfreeguard παίρνει ως όρισμα τον δείκτη cursor που δείχνει σε ποιο σκέλος «βρισκόμαστε». Ο δείκτης cursorg δείχνει στην αρχή της λίστας του συνόλου Διαθέσιμοι για το σκέλος cursor. Ο δείκτης theg_req δείχνει στην αρχή ή στο τέλος (ανάλογα με το αν η Βόρεια ακμή είχε έρθει από αριστερά ή από δεξιά) της διπλά συνδεδεμένης λίστας με τα αιτήματα φυλάκων. Ο ίδιος δείκτης είναι και αυτός για τον οποίο ψάχνουμε ένα διαθέσιμο φύλακα. Αυτή η συνάρτηση μας επιστρέφει true αν βρεθεί κατάλληλος φύλακας για το δοθέν αίτημα φύλακα και false στην αντίθετη περίπτωση.

```
bool lookforfreeguard(node * cursor, guard * cursorg, g_req *
theg_req)
{
    while(cursorg!=NULL)
    {
        if(closureguardwithprange(cursorg,cursorg->thesil,
        cursorg->thesir, theg_req->thesil,theg_req->thesir) &&
        cursorg->level>theg_req->level)
             return true;
        else
            cursorg=cursorg->next;
    }
    return false;
}
```

Για να ελεγχθεί η ύπαρξη ή μη κατάλληλου φύλακα καλούμε τη συνάρτηση closureguardwithprange.

3.3.5 bool closureguardwithprange(guard * cursorg, int x1,int x2, int x3, int x4)

Η συνάρτηση closureguardwithprange παίρνει ως όρισμα τον δείκτη cursorg που δείχνει σε ένα διαθέσιμο φύλακα και τέσσερις ακεραίους. Το (x1, x2) είναι το εύρος θέσης του υποψήφιου διαθέσιμου φύλακα ενώ το (x3,x4) είναι το εύρος τοποθέτησης του αιτήματος φύλακα. Αυτή η συνάρτηση μας επιστρέφει true αν τέμνονται τα δύο αυτά εύρη ή false διαφορετικά.

```
bool closureguardwithprange(guard * cursorg, int x1, int x2, int x3,
                                                              int x4)
//guard(x1,x2),prange(x3,x4)
{
    if((x1<=x3 && x2>=x4))//superset
    {
        cursorg->thesil=x3;
        cursorg->thesir=x4;
        return true;
    }
    else if(x1>=x3 && x2<=x4)
        return true;
    else if(x1<=x3 && x2>x3)
    {
        cursorg->thesil=x2;
        cursorg->thesir=x4;
        return true;
    }
    else if(x1<x4 && x2>=x4)
    {
        cursorg->thesil=x3;
        cursorg->thesir=x1;
        return true;
    }
    else
        return false;
}
```

Το εύρος θέσης του κατάλληλου φύλακα τροποποιείται ανάλογα με τις διάφορες περιπτώσεις τομής των δύο ευρών.

3.3.6 void closurevoreiawithguard(node * cursor, edge voreia)

Η συνάρτηση closurevoreiawithguard παίρνει ως όρισμα την Βόρεια ακμή voreia και τον δείκτη cursor που δείχνει το σκέλος στο οποίο έχει «ακουμπήσει» η voreia. Με το τέλος αυτής της συνάρτησης τα εύρη θέσης και τα εύρη επόπτευσης των διαθέσιμων φυλάκων έχουν περικοπεί κατάλληλα όπου χρειάζεται. Αυτή είναι επίσης η συνάρτηση που τοποθετεί οριστικά κάποιο φύλακα και τον προσθέτει στο σύνολο Τοποθετημένοι.

```
void closurevoreiawithguard(node * cursor,edge
voreia)//voreia(x1,x2),fylakas(x3,x4)
{
    guard *cursorg;
    QPoint guard;
    cursorg=cursor->g_end;
    while(cursorg!=NULL)
    {
        if((voreia.xl<=cursorg->thesil && voreia.xr>=
                                              cursorg->thesir))
          //topothetisi oristikoy fylaka!!!
        {
            guard.setX(cursorg->thesil);
            guard.setY(cursorg->level);
            diagrafieleytheroyfylaka(cursor,cursorg);
            fylakes[numofguards]=guard;
            numofguards++;
        }
        else if(voreia.xl<=cursorg->thesil && voreia.xr>
                                              cursorg->thesil)
        {
            cursorg->thesil=voreia.xr;
            cursorg->visibilityl=voreia.xr;
        }
        else if(voreia.xl<cursorg->thesir && voreia.xr>=
                                              cursorg->thesir)
        {
            cursorg->thesir=voreia.xl;
            cursorg->visibilityr=voreia.xl;
        }
        else if(voreia.xl<=cursorg->visibilityl && voreia.xr>
                                          cursorg->visibilityl)
           cursorg->visibilityl=voreia.xr;
        else if(voreia.xl<cursorg->visibilityr && voreia.xr>=
                                          cursorg->visibilityr)
            cursorg->visibilityr=voreia.xl;
    cursorg=cursorg->prev;
    }
}
```

Με τη συνάρτηση closurevoreiawithguard ελέγχουμε την τομή της Βόρειας ακμής που μόλις ήρθε με τα εύρη επόπτευσης και τα εύρη θέσης των διαθέσιμων φυλάκων. Στη περίπτωση που το εύρος θέσης ενός διαθέσιμου φύλακα

υπερκαλύπτεται από την Βόρεια ακμή τότε ο φύλακας διαγράφεται από το σύνολο Διαθέσιμοι και προστίθεται στο σύνολο Τοποθετημένοι. Όταν απλά «ακουμπάμε» είτε το εύρος θέσης είτε το εύρος επόπτευσης τότε το περικόπτουμε αναλόγως.

3.3.7 void closurevoreiawithprange(node * cursor, int x1,int x2)

Η συνάρτηση closurevoreiawithprange παίρνει ως όρισμα το δείκτη cursor που δείχνει το σκέλος στο οποίο έχει «ακουμπήσει» η Βόρεια ακμή. Οι ακέραιοι x1και x2 είναι τα άκρα της Βόρειας ακμής. Με το τέλος της συνάρτησης αυτής έχουμε περικόψει κατάλληλα τα εύρη τοποθέτησης των αιτημάτων φυλάκων.

```
void closurevoreiawithprange(node *cursor, int x1,int x2)
//voreia(x1,x2),g_req(x3,x4)
{
      int x3,x4;
      g_req * g_reqcursor;
      g_reqcursor=cursor->g_req_front;
     //ki apo end na ksekinoysa pali to idio
      while(g_reqcursor!=NULL)
      {
          x3=g_reqcursor->thesil;
          x4=g_reqcursor->thesir;
          if(x1<=x3 && x2>x3)
              g_reqcursor->thesil=x2;
          else if(x1<x4 && x2>=x4)
              g_reqcursor->thesir=x1;
          g_reqcursor=g_reqcursor->next;
      }
  }
```

Με τη συνάρτηση closurevoreiawithprange ελέγχουμε την τομή της Βόρειας ακμής που μόλις ήρθε με το εύρος τοποθέτησης των αιτημάτων φυλάκων. Όταν αυτά τέμνονται περικόπτουμε ανάλογα το εύρος τοποθέτησης των αιτημάτων φυλάκων.

3.3.8 void elegxosgiakainourio_req(node * cursor)

Η συνάρτηση elegxosgiakainourio_req παίρνει ως όρισμα το δείκτη cursor που δείχνει το σκέλος στο οποίο έχει «ακουμπήσει» η Βόρεια ακμή. Η συνάρτηση τοποθετεί το πολύ δύο αιτήματα φύλακα επιπλέον όπου είναι απαραίτητο.

```
void elegxosgiakainoyriog_req(node *cursor)
{//2ou eidous
    int flag,aristeroterotero,deksiotero;
```

```
bool eidaaristeromeros,eidadeksimeros;
guard *cursorg;
aristerotero=999;
deksiotero=0;
flag=0;
eidaaristeromeros=false;
eidadeksimeros=false;
cursorg=cursor->g_end;
while(cursorg!=NULL)
{
    if(cursorg->visibilityl<=cursor->xl&&
     cursorg->visibilityr>=cursor->xr)//epopteyetai plirws
    {
        eidaaristeromeros=true;
        eidadeksimeros=true;
        break;
    }
    else if(cursorg->visibilityl<=cursor->xl &&
          cursorg->visibilityr>=cursor->xl)
    {
        eidaaristeromeros=true;
        if(cursorg->visibilityr>deksiotero)
            deksiotero=cursorg->visibilityr;
    }
    else if(cursorg->visibilityr>=cursor->xr &&
           cursorg->visibilityl<=cursor->xr)
    {
        eidadeksimeros=true;
        if(cursorg->visibilityl<aristerotero)
            aristerotero=cursorg->visibilityl;
    }
     else
    {
        if(cursorg->visibilityr>deksiotero)
            deksiotero=cursorg->visibilityr;
        if(cursorg->visibilityl<aristerotero)
            aristerotero=cursorg->visibilityl;
    }
    cursorg=cursorg->prev;
}
 if(cursor->g_req_front!=NULL)
{
    if(cursor->g_req_front->forcing <= cursor->xl &&
     cursor->g_req_end->forcingr >= cursor->xr)
     //epopteyetai plirws
    {
        eidaaristeromeros=true;
        eidadeksimeros=true;
    }
    else if(cursor->g_req_front->forcing <= cursor->xl)
    {
        eidaaristeromeros=true;
        if(cursor->g_req_end->forcingr > deksiotero)
            deksiotero = cursor->g_req_end->forcingr;
```

```
        }
        else if(cursor->g_req_end->forcingr>=cursor->xr)
        {
            eidadeksimeros=true;
            if(cursor->g_req_front->forcing < aristerotero)
                aristerotero=cursor->g_req_front->forcingl;
        }
        else
        {
            if(cursor->g_req_end->forcingr > deksiotero)
                deksiotero=cursor->g_req_end->forcingr;
            if(cursor->g_req_front->forcing < aristerotero)
                aristerotero=cursor->g_req_front->forcingl;
        }
}
    if(eidaaristeromeros && eidadeksimeros)
     //epopteyetai apo aristera kai deksia miakneis tipota
        ;
    else if(eidaaristeromeros)
        addg_req(cursor, deksiotero, cursor->xr, RightChain);
    else if(eidadeksimeros)
        addg_req(cursor, cursor->xl,aristerotero,LeftChain);
    else//den eida tipota
    {
        if(aristerotero == 999 && deksiotero == 0)
            addg_req(cursor, cursor->xl,cursor->xr,LeftChain);
        else
        {
            addg_req(cursor, cursor->xl, aristerotero, LeftChain);
            addg_req(cursor, deksiotero, cursor->xr, RightChain);
        }
     }
}
```

Η συνάρτηση elegxosgiakainourio_req αρχικά ελέγχει αν εποπτεύεται το ευθ. τμήμα τομής στη στάθμη της Βόρειας ακμής που μόλις μας ήρθε μείον το x-εύρος της Βόρειας ακμής. Αν εποπτεύεται ολόκληρο δεν κάνει τίποτε. Το ίδιο και αν εποπτεύονται τα δύο άκρα του. Εκτός όμως από την επόπτευση αυτού του τμήματος ελέγχουμε και αν υπάρχει εύρος ανάθεσης που να το «εποπτεύει». Αυτό το κάνουμε γιατί αν υπάρχει ήδη εύρος ανάθεσης δεν κερδίζουμε κάτι με το να προσθέσουμε αίτημα φύλακα με επικαλυπτόμενο εύρος ανάθεσης. Επιπλέον, με αυτό τον τρόπο φροντίζουμε να συνεχίσει να ισχύει η Παρατήρηση 3.1. Αν όμως ένα από τα δύο ή και τα δύο άκρα δεν εποπτεύονται (είτε από διαθέσιμο φύλακα είτε από κάποιο εύρος ανάθεσης ενός αιτήματος φύλακα με τον τρόπο που ανέφερα παραπάνω) τότε η συνάρτηση προσθέτει το πολύ δύο αιτήματα φύλακα. Το εύρος τοποθέτησης είναι όλο το ευθ. τμήμα τομής – x-εύρος της Βόρειας ακμής, ενώ το

εύρος ανάθεσης είναι το (ή τα) μη επιβλεπόμενο άκρο(-α) του ίδιου τμήματος. Σημειώνεται ότι φροντίζουμε να προσθέσουμε το καινούριο αίτημα φύλακα είτε στην αρχή είτε στο τέλος της διπλά συνδεδεμένης λίστας ανάλογα με το που υπάρχει έλλειψη επόπτευσης.

3.3.9 void addguard(node * cursor)

Η συνάρτηση addguard παίρνει ως όρισμα το δείκτη cursor που δείχνει το σκέλος στο οποίο έχει «ακουμπήσει» η Βόρεια ακμή. Η συνάρτηση προσθέτει ένα διαθέσιμο φύλακα στο σκέλος cursor με εύρος θέσης και εύρος επόπτευσης το ευθ. τμήμα τομής – χεύρος της Βόρειας ακμής.

```
void addguard(node * cursor)
{
    guard * aguard;
    aguard=newguard();
    aguard->visibilityl=cursor->xl;
    aguard->visibilityr=cursor->xr;
    aguard->thesil=cursor->xl;
    aguard->thesir=cursor->xr;
    aguard->level=cursor->lastlevel;
    aguard->prev=cursor->g_end;
    if(cursor->g_front==NULL)
    {
        cursor->g_front=aguard;
        cursor->g_end=aguard;
    }
    else
    {
        aguard->prev=cursor->g_end;
        cursor->g_end->next=aguard;
        cursor->g_end=aguard;
    }
}
```

3.3.10 void delete_g_req(node * cursor,int meria)

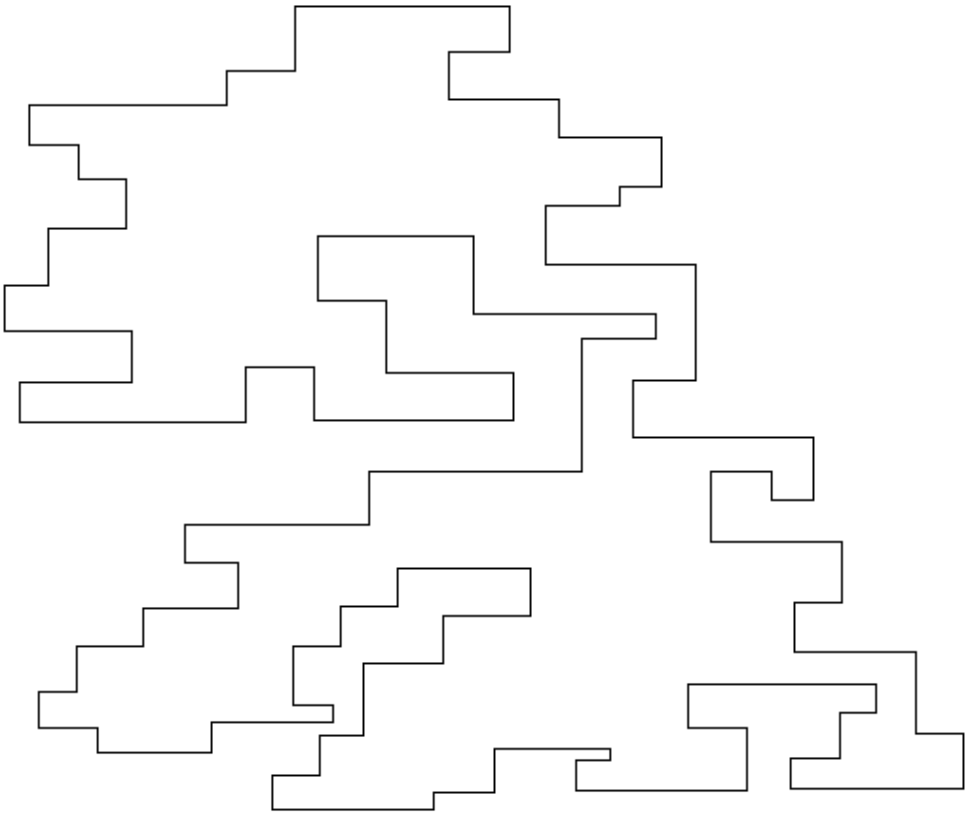
Η συνάρτηση delete_g_req παίρνει ως όρισμα το δείκτη cursor που δείχνει το σκέλος στο οποίο έχει «ακουμπήσει» η Βόρεια ακμή, και τον ακέραιο μεριά που μας δηλώνει από ποια μεριά έχει ακουμπήσει η Βόρεια ακμή το σκέλος. Η συνάρτηση διαγράφει ή το αριστερότερο ή το δεξιότερο αίτημα φύλακα αναλόγως τη meria.

```
void delete_g_req(node * cursor,int meria)
{
    g_req * g_reqcursor;
    if (meria==LeftChain)
    {
        g_reqcursor=cursor->g_req_front;
        svistrag_req(g_reqcursor);
```

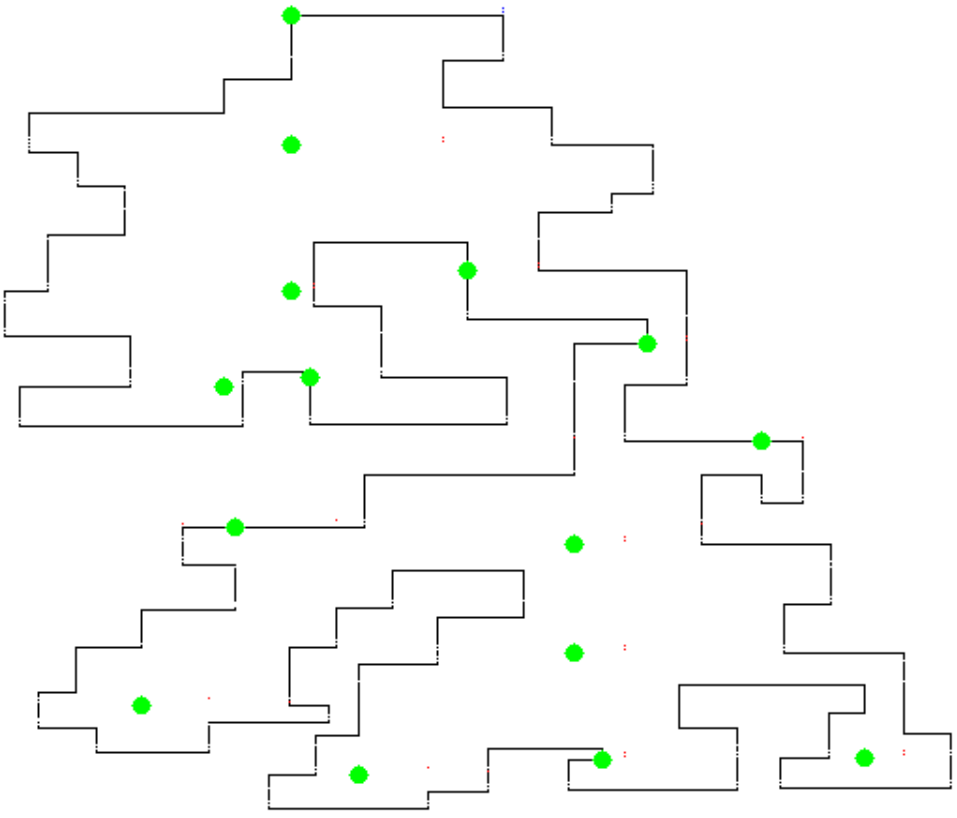
```
        if(g_reqcursor->next!=NULL)
        {
            g_reqcursor=g_reqcursor->next;
            g_reqcursor->prev=NULL;
            cursor->g_req_front=g_reqcursor;
        }
        else
        {
            g_reqcursor=NULL;
            cursor->g_req_end=NULL;
            cursor->g_req_front=NULL;
        }
    }
    else if (meria==RightChain)
    {
        g_reqcursor=cursor->g_req_end;
         svistrag_req(g_reqcursor);
         if(g_reqcursor->prev!=NULL)
        {
            g_reqcursor=g_reqcursor->prev;
            g_reqcursor->next=NULL;
            cursor->g_req_end=g_reqcursor;
        }
        else
        {
            g_reqcursor=NULL;
            cursor->g_req_end=NULL;
            cursor->g_req_front=NULL;
        }
    }
}
```

Χρησιμοποιώντας την Παρατήρηση 3.1, αν η Βόρεια ακμή έχει «ακουμπήσει» στο αριστερό μέρος του σκέλους, έτσι και χρειαστεί να διαγράψουμε ένα αίτημα φύλακα γνωρίζουμε ότι αυτό θα είναι το αριστερότερο. Ανάλογα χειριζόμαστε και την περίπτωση που η Βόρεια ακμή ακουμπήσει στο δεξί μέρος του σκέλους.

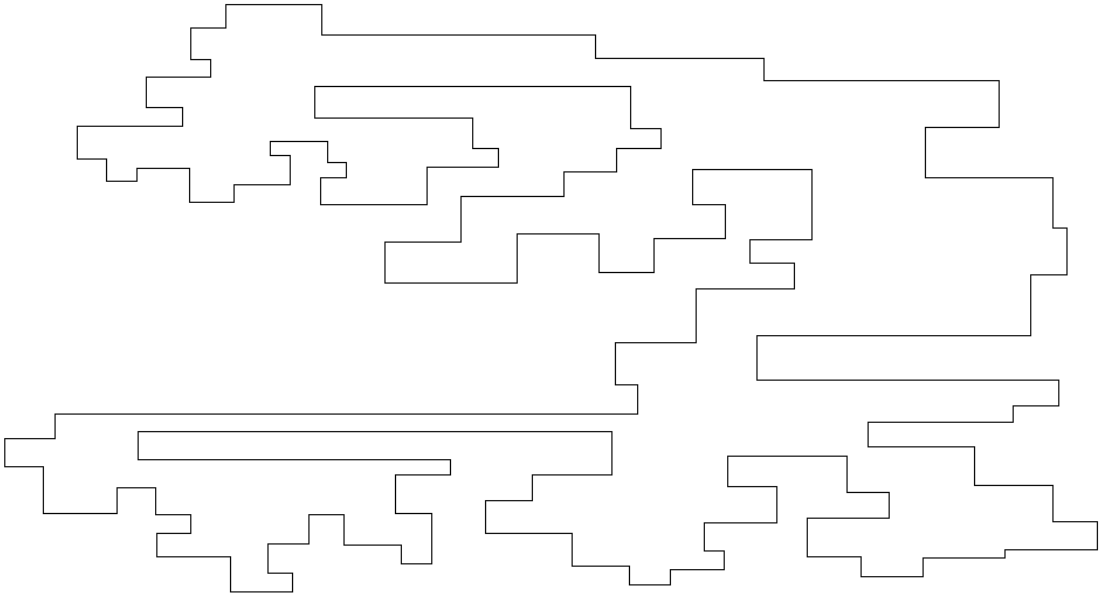
3.4 Ενδεικτικές εκτελέσεις του προγράμματος



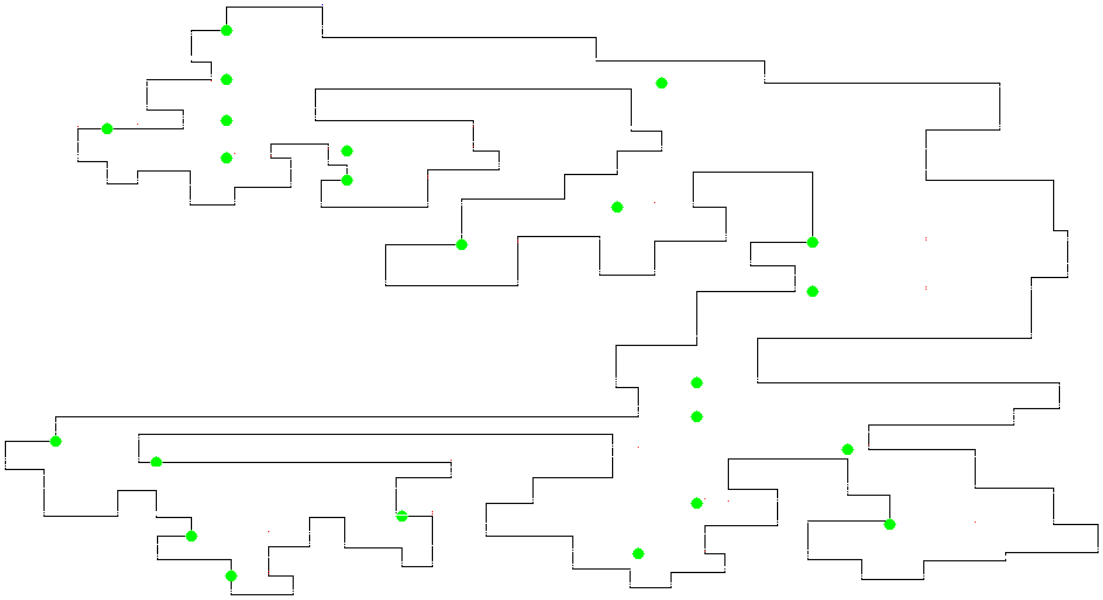
Σχήμα 3.4.1



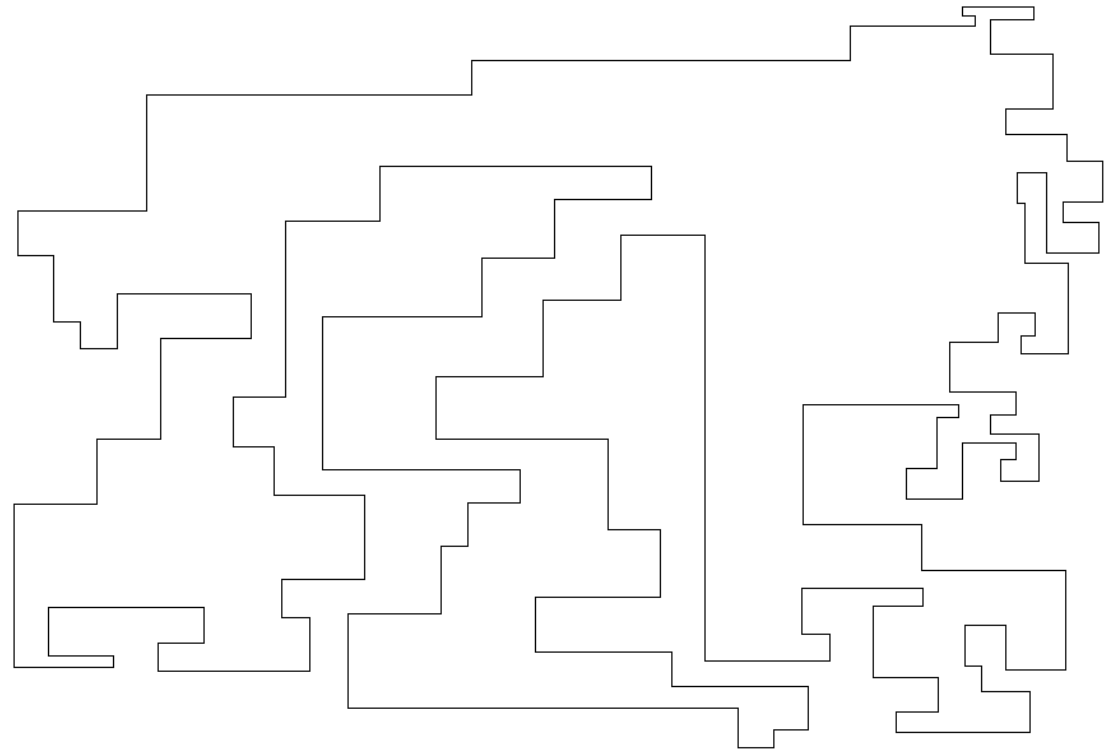
Σχήμα 3.4.2



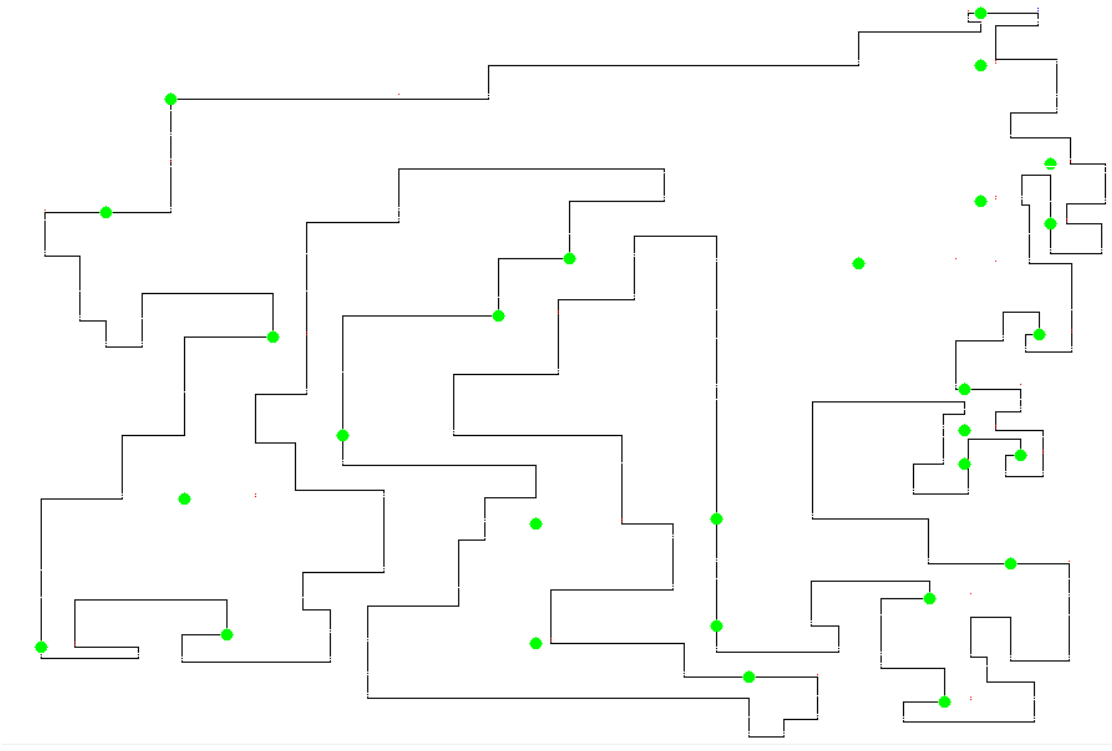
Σχήμα 3.4.3



Σχήμα 3.4.4



Σχήμα 3.4.5



Σχήμα 3.4.6

4. Συμπεράσματα-Επεκτάσεις

Σε αυτή τη μεταπτυχιακή εργασία πραγματευτήκαμε το πρόβλημα της κάλυψης ενός απλού ορθογωνίου πολυγώνου κλάσης 3 με το ελάχιστο πλήθος από r-αστέρες και περιγράψαμε και υλοποιήσαμε έναν αλγόριθμο τάξης $O(n^2)$ χρόνου όπου n είναι το πλήθος των κορυφών του δοθέντος πολυγώνου. Πιστεύουμε ότι ο αλγόριθμός μας θα οδηγήσει σε αλγορίθμους για το παραπάνω πρόβλημα σε γενικά ορθογώνια πολύγωνα οι οποίοι θα είναι πολύ ταχύτεροι από τον αλγόριθμο χρόνου $O(n^{17}polylogn)$ των Worman και Keil [10]. Επομένως, ένα άμεσο ανοιχτό ερώτημα είναι πώς μπορούμε να εκμεταλλευτούμε τις ιδέες από αυτή την εργασία για να βρούμε έναν αλγόριθμο που θα λύνει το πρόβλημα της κάλυψης με r-αστέρες στη γενική περίπτωση απλών ορθογωνίων πολυγώνων (κλάσης 4).

Ένα άλλο ενδιαφέρον ανοιχτό ερώτημα είναι αν μπορούμε να βρούμε ταχύτερους αλγορίθμους για το πρόβλημα κάλυψης ορθογωνίων πολυγώνων κλάσης 4 με s-αστέρες (ο ταχύτερος αλγόριθμος αυτή τη στιγμή απαιτεί $O(n^8)$ χρόνο [7] και βασίζεται σε μια γραφοθεωρητική προσέγγιση). Τέλος, θα ήταν ενδιαφέρον να προσπαθήσουμε να βελτιώσουμε την πολυπλοκότητα του δικού μας

αλγορίθμου. Πιστεύουμε ότι με κατάλληλες ειδικές δομές δεδομένων θα μπορέσουμε να χειριστούμε την περικοπή ευρών ταχύτερα κι αυτό θα μας επιτρέψει να βελτιώσουμε την πολυπλοκότητα χρόνου σε O(nlogn).

Βιβλιογραφία

1. A. Aggarwal, The Art Gallery Theorem: its Variations, Applications, and Algorithmic Aspects, PhD Thesis, Department of Electrical Engineering and Computer Science, John Hopkins University, 1984
2. J. Culberson and R.A. Reckhow, Orthogonally convex coverings of orthogonal polygons without holes, J. Comput. Systems Science 39(2), 166-204, 1989
3. L. Gewali, M. Keil, and S.C. Ntafos, On covering orthogonal polygons with star-shaped polygons, Information Sciences 65, 45-63, 1992
4. J. Kahn, M. Klawe, and D. Kleitman, Traditional galleries require fewer watchmen, SIAM J. Algebraic Discrete Methods 4(2), 194-206, 1983
5. J.M. Keil, Minimally covering a horizontally convex orthogonal polygon, Proc. 2nd Annual ACM Symp. Computational Geometry, 43-51, 1986
6. A. Lingas, A. Wasylewicz, and P. Zyli´nski, Note on covering orthogonal polygons with star-shaped polygons, Information Processing Letters 104(6), 220-227, 2007
7. R. Motwani, A. Raghunathan, and H. Saran, Covering orthogonal polygons with star polygons: the perfect graph approach, J. Comput. Systems Science 40, 19-48, 1990
8. J. O'Rourke, Art Gallery Theorems and Algorithms, Oxford University Press, 1987
9. J. Urrutia, Art gallery and illumination problems, Handbook of Computational Geometry, Elsevier Science, Amsterdam, 973-1027, 2000
10. C. Worman and J.M. Keil, Polygon decomposition and the orthogonal art gallery problem, International Journal of Computational Geometry & Applications 17(2), 105-138, 2007