Υλοποίηση Αλγορίθμων για το Πρόβλημα της Επόπτευσης μιας x-Μονότονης Πολυγωνικής Γραμμής στο Επίπεδο από Δύο Πύργους

Παναγιώτης Μουζούρης

Διπλωματική Εργασία

Επιβλέπων: Λεωνίδας Παληός

Φεβρουάριος 2025

ΤΜΗΜΑ ΜΗΧ. Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF IOANNINA

Ευχαριστίες

Θα ήθελα να εκφράσω την ειλικρινή μου ευγνωμοσύνη στον Καθηγητή Λεωνίδα Παλήο για την ανεκτίμητη καθοδήγησή του, την αμέριστη στήριξη και την ενθάρρυνσή του καθ' όλη τη διάρκεια της έρευνας αυτής. Οι πολύτιμες παρατηρήσεις του και η εκτεταμένη εξειδίκευσή του διαμόρφωσαν καθοριστικά την πορεία της διατριβής. Η υπομονή του και η αφοσίωσή του στη μετάδοση της γνώσης στην υπολογιστική γεωμετρία εμπλούτισαν σημαντικά την εκπαιδευτική μου εμπειρία. Είμαι βαθιά ευγνώμων για τον χρόνο και την προσπάθειά του στην ανασκόπηση της εργασίας μου, καθώς και για τις εποικοδομητικές του προτάσεις που συνέβαλαν ουσιαστικά στη βελτίωση της ποιότητας της έρευνάς μου. Δεν θα μπορούσα να παραλείψω να εκφράσω την ειλικρινή ευγνωμοσύνη μου προς τους γονείς μου, οι οποίοι με στήριξαν σε κάθε βήμα αυτής της διαδρομής. Η αδιάκοπη συμπαράστασή τους, η εμπιστοσύνη τους στις ικανότητές μου και η αφοσίωσή τους αποτέλεσαν την κινητήρια δύναμη πίσω από την ολοκλήρωση αυτής της διατριβής. Χωρίς τη διαρκή τους ενθάρρυνση, την αγάπη και την υπομονή τους, το ταξίδι αυτό δεν θα ήταν το ίδιο.

ABSTRACT

This thesis examines the planar two-watchtower problem, in which we are given a terrain in the plane and we are asked to place two watchtowers of the minimum possible height so that every point of the terrain is watched by at least one of the watchtowers. We focus on the discrete version, in which both watchtowers are built on vertices of the terrain, and the semi-continuous version of the problem, in which at least one of the watchtowers is built on a terrain vertex (the other can be anywhere on the terrain).

The thesis presents the description and an implementation of a decision procedure for the discrete version, an extension of it for the semi-continuous version, and an algorithm for the discrete version of the problem. The implementation has been done using the python programming language.

Περίληψη

Η παρούσα διπλωματική εργασία εξετάζει το πρόβλημα των δύο παρατηρητηρίων (πύργων παρατήρησης) στο επίπεδο, στο οποίο μάς δίδεται μια x-μονότονη πολυγωνική γραμμή στο επίπεδο και μάς ζητείται να τοποθετήσουμε δύο παρατηρητήρια ελαχίστου ύψους έτσι ώστε κάθε σημείο της πολυγωνικής γραμμής να είναι ορατό από τουλάχιστον το ένα παρατηρητήριο. Εστιάζουμε στη διακριτή εκδοχή, στην οποία και τα δύο παρατηρητήρια τοποθετούνται σε κορυφές της πολυγωνικής γραμμής, και την ημι-συνεχή εκδοχή του προβλήματος, στην οποία τουλάχιστον το ένα παρατηρητήριο τοποθετείται σε κορυφή της γραμμής.

Η εργασία παρουσιάζει την περιγραφή και μια υλοποίηση μιας διαδικασίας απόφασης για τη διακριτή εκδοχή, μια επέκτασή της για την ημι-συνεχή περίπτωση και έναν αλγόριθμο για τη διακριτή εκδοχή του προβλήματος. Η υλοποίηση έγινε στη γλώσσα προγραμματισμού python.

Πίνακας περιεχομένων

Κεφάλαιο 1. Εισαγωγή

1.1 Βασικοί Ορισμοί

x-Μονοτονία

Μια πολυγωνική γραμμή (terrain) ονομάζεται x-μονότονο όταν κάθε κατακόρυφη ευθεία τέμνει τη γραμμή σε το πολύ ένα σημείο. Αν T είναι μια x-μονότονη πολυγωνική γραμμή με κορυφές p1, p2, …, pn με x-συντεταγμένες x1, x2, …, xn αντίστοιχα, τότε ισχύει:

$$x1 < x2 < \cdots < xn.$$

Ορατότητα

Σε μια x-μονότονη γραμμή, ένα σημείο p είναι ορατό από ένα σημείο q εάν το ευθύγραμμο τμήμα που τα ενώνει δεν διαπερνά τη γραμμή, δηλαδή δεν υπάρχει κάποιο σημείο του ευθυγράμμου τμήματος που να βρίσκεται κάτω από τη x-μονότονη γραμμή.

Συνθήκη ορατότητας

Αν q=(xq,yq) είναι το σημείο παρατήρησης και p=(xp,yp) είναι ένα σημείο του εδάφους, τότε η ευθεία που τα ενώνει είναι:

$$y= m*x + b \text{ όπου } m= (yp-yq)/(xp-xq) \text{ και } b= yq - m*xq.$$

Το σημείο p είναι ορατό από το q αν και μόνο αν δεν υπάρχει άλλο σημείο r=(xr,yr) του εδάφους μεταξύ xq και xp που να βρίσκεται πάνω από την ευθεία ορατότητας, δηλαδή να μην ισχύει

$$yr > m*xr + b.$$

Αυτή η έννοια είναι κεντρική στο πρόβλημα των δύο πύργων, καθώς ο στόχος είναι να τοποθετηθούν δύο πύργοι ώστε να επιτυγχάνεται πλήρης ορατότητα του εδάφους.

Άνω Κάλυμμα (Upper Envelope)

Το άνω κάλυμμα ενός συνόλου ευθειών στο επίπεδο είναι το ανώτερο επίπεδο που ορίζεται από αυτές, δηλαδή το σύνολο των σημείων των ευθειών που δεν έχουν άλλη ευθεία ψηλότερά τους. Στο πρόβλημα των δύο παρατηρητηρίων, το άνω κάλυμμα των φορέων των (μερικώς ή ολικώς) μη ορατών ακμών του εδάφους προσδιορίζει το ελάχιστο ύψος που πρέπει να φτάσει το δεύτερο παρατηρητήριο για να εξασφαλιστεί η πλήρης επόπτευση του εδάφους από τα δύο παρατηρητήρια.

Το άνω κάλυμμα μπορεί να υπολογιστεί με χρήση της δυϊκότητας και αλγορίθμους κυρτού περιβλήματος όπως ο Graham Scan.

Παρατηρητήριο (Watchtower)

Ένα παρατηρητήριο (πύργος παρατήρησης) είναι ένας κατακόρυφος στύλος τοποθετημένος σε ένα σημείο του εδάφους. Το κατώτερο σημείο του πύργου βρίσκεται στο έδαφος, ενώ το ανώτερο είναι το σημείο παρατήρησης**.**

Αν το σημείο βάσης του πύργου είναι u=(xu,yu) και το ύψος του είναι h, τότε η κορυφή του πύργου είναι:

$$u' = (xu, yu+h).$$

Μη Ορατές Αλυσίδες (Non-Visible Chains)

Οι μη ορατές αλυσίδες είναι αλληλουχίες διαδοχικών ακμών του εδάφους που δεν είναι ολικώς ορατές από το πρώτο παρατηρητήριο. Οι μη ορατές αλυσίδες προσδιορίζουν τα τμήματα του εδάφους που πρέπει να καλυφθούν από το δεύτερο πύργο. Ο υπολογισμός τους γίνεται μέσω εξέτασης των ακμών του εδάφους για ορατότητα και συνένωση των γειτονικών μη ορατών ακμών.

Δυϊκότητα

Η δυϊκότητα (duality) είναι ένας μετασχηματισμός που επιτρέπει την αντιστοίχιση σημείων του επιπέδου σε (μη κατακόρυφες) ευθείες του επιπέδου και αντίστροφα.

Ένα σημείο p=(a,b) μπορεί να μετατραπεί στη δυϊκή του ευθεία: y=ax−b

Αντίστοιχα, μία ευθεία y=mx+c μετατρέπεται στο σημείο (m,−c) στον δυϊκό χώρο**.**

1.2 Αντικείμενο της Διπλωματικής Εργασία

Η παρούσα διπλωματική εργασία εξετάζει το πρόβλημα των δύο πύργων (παρατηρητηρίων) στο επίπεδο, στο οποίο δίδεται μια x-μονότονη πολυγωνική γραμμή T στο επίπεδο και ζητείται η τοποθέτηση δύο παρατηρητηρίων του ελάχιστου ύψους έτσι ώστε κάθε σημείο της πολυγωνικής γραμμής T να είναι ορατό από τουλάχιστον ένα παρατηρητήριο.

Το πρόβλημα εμφανίζεται σε τρεις εκδοχές:

1. *Διακριτή Εκδοχή*: οι βάσεις των δύο παρατηρητηρίων είναι σε κορυφές της πολυγωνικής γραμμής T.
2. *Ημι-Συνεχής Εκδοχή*: η βάση του ενός παρατηρητηρίου είναι σε κορυφή της T, ενώ η βάση του άλλου μπορεί να είναι σε οποιοδήποτε σημείο της T.
3. *Συνεχής Εκδοχή*: οι βάσεις των δύο παρατηρητηρίων μπορούν να είναι σε οποιοδήποτε σημείο κατά μήκος των ακμών του T.

Ο στόχος της διπλωματικής εργασίας είναι η παρουσίαση και η υλοποίηση των ακόλουθων

αλγορίθμων [P25]:

- μιας διαδικασίας απόφασης για τη διακριτή εκδοχή (δηλαδή δοθέντων μιας x-μονότονης πολυγωνικής γραμμής T, μιας κορυφής v της T και ενός ύψους h, να υπολογίσουμε το ελάχιστο ύψος του δεύτερου παρατηρητηρίου σε κορυφή της T έτσι ώστε αυτό σε συνδυασμό με ένα παρατηρητήριο ύψους h τοποθετημένου στην κορυφή v να εποπτεύουν ολόκληρη την πολυγωνική γραμμή T),
- μιας επέκτασης της παραπάνω διαδικασίας απόφασης για την ημι-συνεχή εκδοχή και
- ενός ολοκληρωμένου αλγορίθμου για τη διακριτή εκδοχή του προβλήματος.

Η υλοποίηση έγινε στη γλώσσα προγραμματισμού python.

1.3 Δομή της Διπλωματικής Εργασίας

Η παρούσα διπλωματική εργασία διαρθρώνεται σε τέσσερα κύρια κεφάλαια, καθένα από τα οποία επικεντρώνεται σε διαφορετικές πτυχές του προβλήματος των δύο παρατηρητηρίων και των αλγορίθμων που χρησιμοποιούνται για την επίλυσή του.

Στο Κεφάλαιο 2, περιγράφονται οι αλγόριθμοι που χρησιμοποιούνται στην υλοποίηση, παρέχοντας αναλυτικές επεξηγήσεις και μαθηματικό υπόβαθρο για τον καθένα. Οι κύριοι αλγόριθμοι που αναπτύσσονται περιλαμβάνουν τον αλγόριθμο εντοπισμού μη ορατών αλυσίδων, τον αλγόριθμο υπολογισμού του άνω καλύμματος μέσω δυαδικότητας, καθώς και τους αλγόριθμους αναζήτησης, που περιλαμβάνουν την παραμετρική αναζήτηση, τη δυαδική αναζήτηση, και την ημι-συνεχή αναζήτηση.

Το Κεφάλαιο 3 εστιάζει στην υλοποίηση των αλγορίθμων, παρουσιάζοντας τις σχετικές δομές δεδομένων, την ανάλυση του κώδικα και παραδείγματα εκτέλεσης. Γίνεται περιγραφή της εισόδου και εξόδου των δεδομένων, ενώ παρέχονται λεπτομέρειες για κάθε συνάρτηση που υλοποιεί τις διάφορες μεθόδους ανάλυσης και εύρεσης του ελάχιστου ύψους του δεύτερου παρατηρητηρίου.

Τέλος, το Κεφάλαιο 4 αποτελεί τον επίλογο της εργασίας, όπου συνοψίζονται τα κύρια ευρήματα, συζητούνται τα πλεονεκτήματα και οι περιορισμοί της προτεινόμενης μεθοδολογίας, και παρουσιάζονται πιθανές προεκτάσεις της έρευνας για μελλοντική βελτίωση και εφαρμογή σε άλλα προβλήματα ορατότητας και φύλαξης εδαφών.

Κεφάλαιο 2. Περιγραφές των Αλγορίθμων

2.1 Περιγραφή των Αλγορίθμων

Αλγόριθμος Εντοπισμού Μη Ορατών Αλυσίδων

Ο αλγόριθμος find_non_visible_chains χρησιμοποιείται για τον εντοπισμό των τμημάτων του εδάφους που δεν είναι ορατά από το πρώτο παρατηρητήριο. Για να το πετύχει αυτό:

1. Διαιρεί το terrain σε δύο περιοχές:
    - Αριστερά του παρατηρητηρίου.
    - Δεξιά του παρατηρητηρίου.
2. Εξετάζει τη γεωμετρική φορά των σημείων με βάση το προσημασμένο εμβαδόν τριγώνου:
    - Για το αριστερό τμήμα, απαιτείται αρνητικό πρόσημο για να θεωρηθεί ορατό.
    - Για το δεξιό τμήμα, απαιτείται θετικό πρόσημο.
    - Αν το εμβαδόν είναι 0, τότε τα σημεία είναι συνευθειακά και το τμήμα θεωρείται ορατό.
3. Εντοπίζει ημι-ορατές ακμές:
    - Αν μία ακμή είναι ημι-ορατή (δηλαδή το παρατηρητήριο βλέπει τη μία κορυφή αλλά όχι την άλλη), τότε θεωρείται εξ ορισμού μη ορατή.

Ο αλγόριθμος βασίζεται στον υπολογισμό του προσημασμένου εμβαδού ενός τριγώνου που σχηματίζεται από τα εξής τρία σημεία: το σημείο A στην κορυφή του πύργου, το τελευταίο ορατό σημείο B της x-πολυγωνικής γραμμής και το τρέχον σημείο C της x-πολυγωνικής γραμμής πού δεν γνωρίζουμε ακόμη αν είναι ορατό ή όχι.

Έστω τα σημεία A(xA,yA), B(xB,yB) και C(xC,yC). Ορίζουμε το προσημοποιημένο εμβαδόν:

$\Delta = xA*(yB-yC) + xB*(yC-yA) + xC*(yA-yB)$

- Αν $\Delta > 0$: Τα σημεία σχηματίζουν αριστερόστροφη στροφή (counterclockwise turn).
- Αν $\Delta < 0$: Σχηματίζουν δεξιόστροφη στροφή (clockwise turn).
- Αν $\Delta = 0$: Τα σημεία είναι συνευθειακά.

Παράδειγμα Εφαρμογής:
Έστω το terrain με κορυφές:

[(0,0), (1,3), (2,0), (3,3), (4,0), (5,8), (7,0), (9,1), (10,0)]

και το παρατηρητήριο βρίσκεται στο x=5 με h1=4. Ο αλγόριθμος εντοπίζει ως μη ορατές αλυσίδες τις διαδοχικές ακμές: [((0, 0), (1, 3))].
Αυτό σημαίνει ότι τα συγκεκριμένα τμήματα δεν είναι ορατά από το πρώτο παρατηρητήριο, καθώς παρεμποδίζονται από υψηλότερα σημεία του terrain.

Αλγόριθμος Υπολογισμού του Άνω Καλύμματος μέσω Δυαδικότητας

Ο υπολογισμός του Άνω Καλύμματος (Upper Envelope) των μη ορατών αλυσίδων βασίζεται στη γεωμετρική δυαδικότητα. Η ιδέα είναι η μετατροπή κάθε ευθείας που ορίζει μία μη ορατή ακμή σε ένα δυαδικό σημείο, επιτρέποντας την αναδιατύπωση του προβλήματος σε έναν πιο απλό υπολογιστικό χώρο.

Η διαδικασία ακολουθεί τα εξής στάδια:

1. Μετατροπή μη ορατών ακμών σε δυαδικά σημεία:

   Κάθε ευθεία γραμμή y=m*x+b μετατρέπεται στο δυαδικό σημείο (m,−b).

2. Υπολογισμός της κυρτής θήκης των δυαδικών σημείων:

   Χρησιμοποιείται ο αλγόριθμος Graham Scan, ο οποίος ταξινομεί τα σημεία και κατασκευάζει την κυρτή θήκη τους σε χρόνο O(nlogn).

3. Μετατροπή των σημείων πίσω στον αρχικό χώρο (primal space):

   Επαναφέρουμε το αποτέλεσμα στην αρχική μορφή, κατασκευάζοντας την ορατή ανώτατη περιβάλλουσα των μη ορατών αλυσίδων.

4. Υπολογισμός της Άνω Εφαπτομένης (Upper Tangent Line)

Το τελευταίο τμήμα του Άνω Καλύμματος συνδέθηκε είτε με το τελευταίο σημείο του terrain είτε με μια οριζόντια επέκταση. Αν το terrain τελείωνε χαμηλότερα από το τελευταίο σημείο του Άνω Καλύμματος, προστέθηκε μια οριζόντια γραμμή (γέφυρα).Αν το terrain τελείωνε ψηλότερα, υπολογίστηκε μια τελική εφαπτομένη προς το terrain, εξασφαλίζοντας μια ομαλή μετάβαση.

Το τελικό αποτέλεσμα του άνω Καλύμματος (Upper Envelope) προέκυψε ως μία ακολουθία από ευθείες, οι οποίες προσδιορίζουν το ελάχιστο απαιτούμενο ύψος για το δεύτερο παρατηρητήριο.

Μαθηματική Θεμελίωση

Η προσέγγιση βασίζεται στη μετατροπή γραμμών σε δυαδικά σημεία και στην εύρεση της κυρτής θήκης.

1. Δυαδική Μετατροπή

Έστω μια ευθεία της μορφής:

$y=m*x+b$

Η δυαδική αναπαράσταση αυτής της ευθείας δίνεται από τη μετατροπή:

$\text{Dual}(y=m*x+b) \longrightarrow (m,-b)$

Αυτό σημαίνει ότι κάθε γραμμή στο primal επίπεδο αντιστοιχεί σε ένα σημείο στο dual επίπεδο.

2.Μετατροπή της Κυρτής Θήκης Πίσω στο Primal Space

Αφού βρούμε την κυρτή θήκη των δυαδικών σημείων, επιστρέφουμε στο primal επίπεδο. Για κάθε σημείο $(m_i,-b_i)$ της κυρτής θήκης, δημιουργούμε ξανά τη γραμμική εξίσωση:

$y = m_i x + b_i$

Αυτό σημαίνει ότι το Άνω Κάλυμμα αποτελείται από ένα σύνολο γραμμικών τμημάτων, τα οποία είναι τα ανώτερα μέρη των αρχικών ευθειών.

3. Υπολογισμός των Τομών των Ευθειών Γραμμών

Για κάθε δύο διαδοχικές ευθείες $y = m_i x + b_i$ και $y = m_{i+1} x + b_{i+1}$, βρίσκουμε το σημείο τομής:

$$x = (b_{i+1} - b_i) / (m_i - m_{i+1})$$

Το τελικό Upper Envelope προκύπτει από τις αλυσίδες τομών αυτών των γραμμικών τμημάτων.

4. Προσθήκη της Άνω Εφαπτομένης (Upper Tangent)

Η Άνω Εφαπτομένη (Upper Tangent) είναι απαραίτητη για να εξασφαλίσουμε τη σωστή σύνδεση του τελευταίου τμήματος του Άνω Καλύμματος με το τελευταίο σημείο του terrain.

Υπολογισμός της Άνω Εφαπτομένης

Έστω το τελευταίο σημείο του Άνω Καλύμματος (xlast,ylast) και το τελευταίο σημείο του terrain (xT,yT).

Η τελική ευθεία που συνδέει αυτά τα δύο σημεία έχει εξίσωση: $y=mTx+bT$

όπου: $mT=yT-ylastxT-xlast$ και $bT=ylast-mTxlast$

Αν το yT είναι χαμηλότερο από το ylast, τότε τοποθετούμε μια οριζόντια γραμμή ως εφαπτομένη.

Παράδειγμα Εφαρμογής

Είσοδος

Ας υποθέσουμε ότι έχουμε τις παρακάτω μη ορατές ακμές:

Non-visible chains: [((7, 5), (8, 6)), ((5, 0), (7, 5)), ((3, 0), (5, 0)), ((2, 4), (3, 0)), ((1, 5), (2, 4)), ((0, 15), (1, 5))]

Έξοδος

Upper Envelope: [(-1.0, 25.0), (-0.875, 23.75), (7.666666666666667, 6.666666666666666), (10.0, 12.5)]

Αλγόριθμος Απόφασης για τη Διακριτή Εκδοχή

Η εύρεση του ελάχιστου ύψους του δεύτερου παρατηρητηρίου αποτελεί κρίσιμο βήμα για την πλήρη κάλυψη του terrain. Στόχος μας είναι να εξετάσουμε διαδοχικά όλες τις κορυφές του Upper Envelope και υπολογίζουμε το απαιτούμενο ύψος h2 για το δεύτερο παρατηρητήριο για κάθε κορυφή για να προσδιορίσουμε το μικρότερο ύψος του δεύτερου παρατηρητηρίου.

Ο αλγόριθμος εκτελεί τα εξής βήματα:

1. Διατρέχει κάθε σημείο (vx,vy) του upper envelope.
2. Για κάθε ακμή ((x1,y1),(x2,y2)) του terrain, αν το vxβρίσκεται εντός της περιοχής της ακμής,
    1. Υπολογίζει το ύψος του terrain στη θέση αυτή yupper
    2. Υπολογίζει το απαιτούμενο ύψος ώστε το σημείο να είναι ορατό h2= max(0,yupper − vy)
    3. Επιλέγει το σημείο του upper envelope που απαιτεί το ελάχιστο h2.
    4. Το αποτέλεσμα είναι το h2 και η βέλτιστη τοποθεσία του δεύτερου παρατηρητηρίου.

Μαθηματική Θεμελίωση

Η παραμετρική αναζήτηση δεν βασίζεται στη μονοτονικότητα, αλλά στην εξέταση όλων των πιθανών τοποθεσιών του upper envelope.

Για κάθε σημείο (vx,vy) του upper envelope, πρέπει να ελέγξουμε όλες τις ακμές του terrain και να προσδιορίσουμε ποια είναι η μέγιστη απαίτηση ύψους h2 ώστε το σημείο να έχει οπτική επαφή με το πρώτο παρατηρητήριο.

Αυτό επιτυγχάνεται ως εξής:

1. Για κάθε ακμή του terrain ((x1,y1),(x2,y2)), αν το σημείο vx βρίσκεται εντός της περιοχής αυτής (x1≤vx≤x2), τότε χρησιμοποιούμε γραμμική παρεμβολή (linear interpolation) για να υπολογίσουμε το yupper , δηλαδή το ύψος του terrain σε εκείνο το σημείο:

    $$yupper=(y2−y1)(x2−x1)\cdot(vx−x1)+y1$$

Το απαιτούμενο ύψος h2 που χρειάζεται ώστε το σημείο (vx,vy) του upper envelope να υπερβαίνει την οπτική παρεμπόδιση είναι:

h2=max(0,yupper − vy)

2. Για κάθε σημείο του upper envelope, διατηρούμε το μέγιστο απαιτούμενο h2 μεταξύ όλων των ακμών.

3. Το σημείο του upper envelope που απαιτεί το μικρότερο h2 είναι η βέλτιστη τοποθεσία του δεύτερου παρατηρητηρίου.

Παράδειγμα Εφαρμογής

Είσοδος

Έστω ένα terrain:

T=[(0,0),(1,3),(2,0),(3,3),(4,0),(5,8),(7,0),(9,1),(10,0)]

και ένα πρώτο παρατηρητήριο στη θέση x=0 με ύψος h1=0.

Έξοδος

Ο αλγόριθμος επιστρέφει: h2=1, Τοποθεσία: (5,8)

Αλγόριθμος Απόφασης για την Ημι_συνεχή Εκδοχή

Ο αλγόριθμος απόφασης για την ημι-συνεχή έκδοχή αποτελεί επέκταση του αλγορίθμου απόφασης για τη διακριτή εκδοχή, επιτρέποντας την τοποθέτηση του δεύτερου παρατηρητηρίου όχι μόνο σε κορυφές του terrain, αλλά και στα μέσα των ακμών.

Διαφορές από άλλες εκδοχές

- Σε σχέση με τη διακριτή εκδοχή: Επεκτείνει τον χώρο αναζήτησης πέρα από τις κορυφές, δίνοντας περισσότερες επιλογές.

- Σε σχέση με τη συνεχή εκδοχή: Δεν επιτρέπει την τοποθέτηση σε οποιοδήποτε σημείο της ακμής, αλλά μόνο στα μέσα των ακμών, μειώνοντας έτσι την υπολογιστική πολυπλοκότητα.

Αυτή η μέθοδος επιτρέπει μεγαλύτερη ευελιξία στη θέση του δεύτερου παρατηρητηρίου, διατηρώντας ταυτόχρονα την αποδοτικότητα του αλγορίθμου. Ωστόσο, απαιτεί πρόσθετους ελέγχους για να διασφαλιστεί ότι η μετάβαση μεταξύ σημείων και μεσαίων σημείων των ακμών διατηρεί την ορθότητα της ορατότητας.

Μαθηματική Θεμελίωση

Έστω ένα terrain T που ορίζεται από n κορυφές P={p1,p2,…,pn}, όπου κάθε κορυφή pi=(xi,yi) συνδέεται με την επόμενη μέσω ευθύγραμμων ακμών.

Για κάθε σημείο τοποθέτησης s του δεύτερου παρατηρητηρίου, το ελάχιστο ύψος του h2 υπολογίζεται με βάση το Άνω Κάλυμμα (Upper Envelope) των μη ορατών ακμών της x-μονότονης πολυγωνικής γραμμής.

Το Άνω Κάλυμμα υπολογίζεται μέσω γεωμετρικής δυαδικότητας και του αλγορίθμου Graham Scan.

Η επιλογή της βέλτιστης τοποθεσίας γίνεται επιλέγοντας το σημείο που ελαχιστοποιεί το απαιτούμενο ύψος h2.

Παράδειγμα Εφαρμογής

Είσοδος

Έστω ένα terrain:

T =[(0, 3), (1, 5), (2, 4), (3, 0), (5, 0), (7, 5), (8, 6), (9, 4)]

και ένα πρώτο παρατηρητήριο στη θέση x=0 με ύψος h1=0.

Έξοδος

Ο αλγόριθμος επιστρέφει: h2= 0.9999999999999991, Τοποθεσία: (7.666666666666667, 5.666666666666667)

Διακριτός Αλγόριθμος χωρίς παραμετρική αναζήτηση

Ο διακριτός αλγόριθμος χωρίς παραμετρική αναζήτηση λύνει το πρόβλημα εύρεσης βέλτιστων θέσεων και ελάχιστου ύψους για δύο παρατηρητήρια, που τοποθετούνται αποκλειστικά σε κορυφές του terrain, έτσι ώστε να καλύπτουν πλήρως την περιοχή.

Η βασική διαφορά από παραμετρικές μεθόδους είναι ότι αντί για συνεχείς αναζητήσεις ύψους, ο αλγόριθμος υπολογίζει και ταξινομεί κρίσιμα ύψη, και χρησιμοποιεί δυαδική αναζήτηση (binary search) για την εύρεση της βέλτιστης λύσης.

Προεπεξεργασία: Ταξινόμηση Ακμών κατά Κλίση

Αρχικά, ταξινομούμε τις ακμές του terrain κατά αύξουσα κλίση και τις αποθηκεύουμε σε έναν πίνακα.
Αυτό διευκολύνει τον αποτελεσματικό υπολογισμό του Άνω Καλύμματος των ακμών.

2. Υπολογισμός Κρίσιμων Υψών Τύπου 1

Για κάθε κορυφή u του terrain, κατασκευάζουμε το δέντρο συντομότερων μονοπατιών (Shortest Path Tree - SPT).

Αυτό μας επιτρέπει να βρούμε τα κρίσιμα ύψη τύπου 1, δηλαδή τις τιμές ύψους στις οποίες αλλάζει η ορατότητα του terrain από την κορυφή u.

Αντιστοιχεί στα σημεία τομής της καθέτου στο u με τις ευθείες που ορίζουν οι ακμές του SPT.

Το αποτέλεσμα είναι ένα σύνολο ταξινομημένων κρίσιμων υψών:

$H(u)=\{h0(u)=0,h1(u),h2(u),...,hk(u)\}$

Ιδιότητα:
Για οποιοδήποτε hhh μεταξύ δύο διαδοχικών κρίσιμων υψών hi(u) και hi+1(u)η ορατότητα του terrain παραμένει σταθερή.

3. Υπολογισμός Κρίσιμων Υψών Τύπου 2

Για κάθε ζευγάρι κορυφών u,v του terrain:

- Υπολογίζουμε το δέντρο συντομότερων μονοπατιών για το v και εξάγουμε το σύνολο κρίσιμων υψών του v, δηλαδή το H(v).
- Δημιουργούμε μια κοινή λίστα κρίσιμων υψών για το ζεύγος (u,v): $B=H(u)\cup H(v)$

Για κάθε ακμή e μεταξύ u και v:

1. Υπολογίζουμε τις βραχύτερες διαδρομές από u και v στα άκρα της ακμής.
2. Βρίσκουμε το ελάχιστο ύψος $h_{u,e,v}$ ώστε τα παρατηρητήρια σε u και v να καλύπτουν πλήρως την ακμή e.

   Αυτό γίνεται λύνοντας μια τετραγωνική εξίσωση που προκύπτει από τη συνθήκη συνευθειακότητας $Ah2+Bh+C=0$ όπου οι συντελεστές A,B,C καθορίζονται από τη γεωμετρία της ακμής.
3. Τοποθετούμε το $h_{u,e,v}$ στο B.

Στο τέλος, έχουμε ένα ταξινομημένο σύνολο κρίσιμων υψών για το ζεύγος (u,v).

4. Δυαδική Αναζήτηση στο Σύνολο Κρίσιμων Υψών

Χρησιμοποιούμε δυαδική αναζήτηση στο BBB για να βρούμε το ελάχιστο ύψος $h^*_{u,v}$ για το οποίο:

1. Η απόφαση για το αν το terrain καλύπτεται πλήρως από τα παρατηρητήρια μπορεί να ληφθεί γρήγορα.

2. Το ύψος του δεύτερου παρατηρητηρίου δεν υπερβαίνει το αντίστοιχο κρίσιμο ύψος.

Κατά την αναζήτηση:

- Εφαρμόζουμε την απόφαση της διακριτής εκδοχής του προβλήματος για ένα δοκιμαστικό ύψος h.

- Ενημερώνουμε τα όρια της αναζήτησης σύμφωνα με το αποτέλεσμα (Θεώρημα 2(b)) μέχρι να βρούμε το ελάχιστο κρίσιμο ύψος $h^*_{u,v}$.

Το αποτέλεσμα είναι $h^* = \min_{(u,v)\in V\times V} h^*_{u,v}$ το οποίο αποτελεί το τελικό βέλτιστο ύψος των παρατηρητηρίων.

Παράδειγμα Εφαρμογής

Είσοδος

Έστω ένα terrain:

T = [(0, 0), (1, 5), (2, 4), (3, 0),(4,5),(6,0) ,(7, 5), (8, 6), (9, 4),(10,8),(11,0),(12,0)]

Έξοδος

Optimal minimum height for towers: 3.0

Best pair of vertices (tower positions): (Tower1X=1,Tower2X= 9)

Κεφάλαιο 3: Η Υλοποίηση

3.1 Είσοδος – Έξοδος

Είσοδος

Η είσοδος της υλοποίησης αποτελείται από μια σειρά δεδομένων που περιγράφουν τη x-μονότονη πολυγωνική γραμμή και τις αρχικές παραμέτρους των παρατηρητηρίων.

1. Δεδομένα Εδάφους (x-μονότονη πολυγωνική γραμμή):
   Το terrain αναπαρίσταται ως μια λίστα σημείων(οπού δίνονται από χρήστη) της μορφής:

   $$T=[(x1,y1), (x2,y2), \ldots, (xn,yn)]$$

   όπου κάθε σημείο (xi,yi) εκφράζει τη θέση μιας κορυφής της x-μονότονης πολυγωνικής γραμμής.

2. Δείκτης του Πρώτου Παρατηρητηρίου (u_idx):
   Ένας ακέραιος αριθμός που δηλώνει τη θέση του πρώτου παρατηρητηρίου μέσα στη λίστα T.

3. Ύψος του Πρώτου Παρατηρητηρίου (h1):
   Ένας δεκαδικός αριθμός που εκφράζει το ύψος του πρώτου παρατηρητηρίου πάνω από τη θέση του στη x-μονότονη πολυγωνική γραμμή.

4. Τύπος Αναζήτησης για το Δεύτερο Παρατηρητήριο:
   Ορίζει αν η αναζήτηση θα γίνει διακριτά (σε κορυφές μόνο) ή ημι-συνεχώς (σε κορυφές και ακμές της x-μονότονης πολυγωνικής γραμμής).

5. Έλεγχος Εγκυρότητας Δεδομένων:
   Για να διασφαλιστεί η ορθότητα των δεδομένων, ελέγχεται ότι:

   - Η πολυγωνική γραμμή είναι x-μονότονη, δηλαδή κάθε κατακόρυφη ευθεία τέμνει την πολυγωνική γραμμή σε το πολύ ένα σημείο.

   - Ο δείκτης του πρώτου παρατηρητηρίου είναι εντός των ορίων της λίστας T.

Παράδειγμα Δεδομένων Εισόδου:

Έστω ότι η πολυγωνική γραμμή αποτελείται από τις εξής κορυφές:

$$T=[(0,0),(1,3),(2,0),(3,3),(4,0),(5,5),(6,2)]$$

και το πρώτο παρατηρητήριο βρίσκεται στο u_idx=1 (δηλαδή στο σημείο (1,3)) με ύψος h1=4

Έξοδος

Η έξοδος της υλοποίησης περιλαμβάνει τα αποτελέσματα των αλγορίθμων για τον εντοπισμό του βέλτιστου ύψους και θέσης του δεύτερου παρατηρητηρίου.

1. Ελάχιστο Ύψος του Δεύτερου Παρατηρητηρίου (h2):
   Ένας δεκαδικός αριθμός που αντιπροσωπεύει το ελάχιστο ύψος που απαιτείται για το δεύτερο παρατηρητήριο ώστε να καλύψει το terrain.

2. Βέλτιστη Τοποθεσία του Δεύτερου Παρατηρητηρίου:

Ένα σημείο (x,y) που αναπαριστά τη θέση όπου πρέπει να τοποθετηθεί το δεύτερο παρατηρητήριο.

3. Μη Ορατές Αλυσίδες:

Μια λίστα από segments που αναπαριστούν τα μη ορατά τμήματα του terrain.

4. Άνω Κάλυμμα (Upper Envelope):

Μια λίστα από segments που ορίζουν το ανώτατο περίγραμμα των μη ορατών αλυσίδων.

5. Μέθοδος Υπολογισμού Βέλτιστης Λύσης:
   Ανάλογα με τον τύπο της αναζήτησης, επιστρέφεται και η μέθοδος που χρησιμοποιήθηκε:

   - Παραμετρική Αναζήτηση: Αν χρησιμοποιήθηκε η προσέγγιση εύρεσης του βέλτιστου ύψους μέσω του upper envelope.
   - Δυαδική Αναζήτηση: Αν έγινε δυαδική αναζήτηση για το h2.
   - Ημι-συνεχής Αναζήτηση: Αν εξετάστηκαν σημεία και πάνω σε ακμές του terrain.

6. Οπτικοποίηση της Λύσης:
   Δημιουργείται ένα γράφημα το οποίο περιλαμβάνει:

   - Την απεικόνιση του terrain.
   - Τα σημεία των μη ορατών αλυσίδων.

- Το κάτω κάλυμμα.
- Τη θέση των παρατηρητηρίων.
- Τις τοποθεσίες των παρατηρητηρίων.
  Επιπλέον, στην οπτική αναπαράσταση μπορούν να χρησιμοποιηθούν συγκεκριμένα χρώματα ή σύμβολα για κάθε στοιχείο (μπλε για μη ορατές αλυσίδες, μαύρο για το terrain, κόκκινο για πρώτου παρατηρητή, μπλε ο δεύτερος παρατηρητής).

7. Μήνυμα Σφάλματος:
   Σε περίπτωση που δεν βρεθεί λύση λόγω μη ικανοποίησης κάποιου κριτηρίου ορατότητας, το πρόγραμμα επιστρέφει αντίστοιχο μήνυμα σφάλματος ή προειδοποίησης.

Παράδειγμα Δεδομένων Εξόδου:
Αν εφαρμοστεί η αναζήτηση σε ένα terrain, το πρόγραμμα μπορεί να επιστρέψει την ακόλουθη έξοδο:

Non-visible chains: [((3, 3), (4, 0)), ((5, 5), (6, 2))]

Upper envelope: {(4, 0), (6, 2), (3, 3), (5, 5)}

Optimal height of second tower: 0, Location: (5, 5)

3.2 Δομές Δεδομένων

Η υλοποίηση του προγράμματος οργανώνεται γύρω από μερικές βασικές δομές δεδομένων και αλγοριθμικές διαδικασίες που συνεργάζονται για την επίλυση του προβλήματος των δύο παρατηρητηρίων. Το βασικό αντικείμενο είναι το **Terrain**, που αναπαρίσταται ως λίστα σημείων (Points) κατά μήκος του x-άξονα. Από το Terrain εξάγονται οι **Μη Ορατές Αλυσίδες** (Non-visible Chains), οι οποίες αποθηκεύονται ως λίστα segments. Στη συνέχεια, οι μη ορατές ακμές μετατρέπονται σε δυαδικά σημεία μέσω της τεχνικής της γεωμετρικής δυαδικότητας και, χρησιμοποιώντας τον αλγόριθμο Graham Scan, υπολογίζεται το **Άνω Κάλυμμα** (Upper Envelope) των δυαδικών σημείων. Τέλος, οι μέθοδοι αναζήτησης (παραμετρική, δυαδική και ημι-συνεχής αναζήτηση) αξιοποιούν το παραπάνω αποτέλεσμα για να εντοπίσουν το ελάχιστο απαιτούμενο ύψος και τη βέλτιστη τοποθεσία του δεύτερου παρατηρητηρίου.

Σημείο (Point)

Το σημείο αναπαριστά μια κορυφή του terrain και αποθηκεύεται ως δυάδα (x,y), όπου x είναι η x-συντεταγμένη και y η y-συντεταγμένη του σημείου.

Ακμή (Segment)

Μια ακμή αναπαριστά ένα τμήμα του terrain που συνδέει δύο διαδοχικές κορυφές(Point). Αποθηκεύεται ως ζεύγος σημείων ((x1,y1),(x2,y2)).

Terrain (Λίστα σημείων του εδάφους)

Το terrain αναπαρίσταται ως λίστα σημείων, η οποία περιέχει όλες τις κορυφές του εδάφους με τη σειρά που εμφανίζονται κατά μήκος του x-άξονα.

Λίστα Μη Ορατών Αλυσίδων (Non-visible Chains)

Η λίστα non_visible_chains περιέχει τα τμήματα του terrain που δεν είναι ορατά από το πρώτο παρατηρητήριο. Κάθε στοιχείο της λίστας είναι ένα segment που περιγράφει ένα μη ορατό τμήμα.

Λίστα Άνω Καλύμματος (Upper Envelope)

Η λίστα upper_envelope περιέχει τα σημεία του άνω καλύμματος, τα οποία υπολογίζονται μέσω δυαδικότητας και του αλγορίθμου Graham Scan.

Δομές για την Επεξεργασία Δυαδικότητας

Για την εφαρμογή της γεωμετρικής δυαδικότητας, χρησιμοποιούμε μια αντιστοίχιση μεταξύ των γραμμών του terrain και των δυαδικών σημείων που αντιστοιχούν σε αυτές.

Λεξικό Αντιστοίχισης Δυαδικών Σημείων (Dual Mapping)

Ένα λεξικό χρησιμοποιείται για να συνδέει τις μη ορατές ακμές με τα δυαδικά τους σημεία.

Δομές για τη Διαδικασία Αναζήτησης

Για την εύρεση του ελάχιστου ύψους του δεύτερου παρατηρητηρίου, χρησιμοποιούνται επιπλέον δομές δεδομένων.

Βέλτιστο Ύψος και Τοποθεσία

Το ελάχιστο ύψος του δεύτερου παρατηρητηρίου και η τοποθεσία του αποθηκεύονται σε μία μεταβλητή τύπου Tuple.

Παράμετροι Δυαδικής Αναζήτησης

Για την εφαρμογή της δυαδικής αναζήτησης, αποθηκεύονται τα όρια της αναζήτησης.

Συμπέρασμα

Οι δομές δεδομένων που χρησιμοποιούνται στην υλοποίηση του προβλήματος των δύο παρατηρητηρίων επιτρέπουν την αποδοτική αναπαράσταση του terrain, την ανίχνευση των μη ορατών αλυσίδων, την εύρεση του άνω καλύμματος και την αναζήτηση της βέλτιστης θέσης του δεύτερου παρατηρητηρίου. Η χρήση λίστας σημείων, segments, λεξικών αντιστοίχισης δυαδικών σημείων και ορίων αναζήτησης επιτρέπει τη βελτιστοποίηση των υπολογισμών, ενώ η οργάνωση των δεδομένων σε κλάσεις συμβάλλει στην καθαρότερη διαχείριση της υλοποίησης.

3.3 Συνάρτηση find_non_visible_chains

Η συνάρτηση find_non_visible_chains εντοπίζει τις μη ορατές αλυσίδες του terrain, δηλαδή τις ακμές για τις οποίες, χρησιμοποιώντας τη μέθοδο του υπολογισμού του εμβαδού τριγώνου, διαπιστώνεται ότι δεν είναι ορατές από το πρώτο παρατηρητήριο. Συγκεκριμένα με τη βοήθεια της συνάρτησης split_and_reverse και, μέσω της συνάρτησης cross_product, ελέγχει τη στροφή με χρήση του προσημοποιημένου εμβαδού Δ για να καταγράψει τις ακμές που δεν πληρούν το κριτήριο ορατότητας. Άμεσος μετα περνώ της ακμές και δημιουργεί την άνω εφαπτομένη καλώντας compute_upper_tangents

Η συνάρτηση επιστρέφει μια λίστα από segments (δηλαδή, ζεύγη σημείων) που αντιπροσωπεύουν τις μη ορατές αλυσίδες και ζεύγη για τα άνω εφαπτόμενα τμήματα.

Κώδικας:

```
def find_non_visible_chains(terrain: List[Point], u_idx: int, h1: float) ->
Tuple[List[Segment], List[Tuple[float, float]]]:
    global previous, towerOne, non_visible, general, half_visible
    previous = ()
    general = ()
    non_visible = []
    half_visible = ()
    left, right, towerOne = split_and_reverse(terrain, u_idx)
```

```
(x, y) = towerOne
y = y + h1
towerOne = (x, y)
if len(left) > 1:
    for i in range(len(left)):
        (x, y) = left[i]
        if left[0] == (x, y):
            previous = (x, y)
            general = left[i+1]
        else:
            if cross_product(towerOne, previous, left[i]) <= 0:
                if half_visible == left[i-1]:
                    non_visible.append((left[i], half_visible))
                    previous=(x,y)
                else:
                    previous = (x, y)
            else:
                non_visible.append(((x, y), general))
                half_visible = left[i]
        general = left[i]
previous = ()
general = ()
half_visible = ()
if len(right) > 1:
    for i in range(len(right)):
        (x, y) = right[i]
        if right[0] == (x, y):
            previous = (x, y)
        else:
            evaluation=cross_product(towerOne, previous, right[i])
            if  evaluation>= 0:

                if half_visible == right[i-1]:

                    non_visible.append((half_visible,right[i]))
                    previous=(x,y)
                else:
                    previous = (x, y)

            elif evaluation<0:


                non_visible.append((general,(x, y)))
                half_visible=(x,y)

        general = right[i]


L_r,right_endpoints=compute_upper_tangents(non_visible, terrain)

return non_visible, L_r
```

3.3.1 Συνάρτηση compute_upper_tangents

Η συνάρτηση compute_upper_tangents υπολογίζει τις άνω εφαπτόμενες (upper tangents) που συνδέουν τις κορυφές του terrain με τις ακμές των μη ορατών αλυσίδων. Αυτή η διαδικασία είναι κρίσιμη για την κατασκευή του Άνω Καλύμματος (Upper Envelope), καθώς επιτρέπει την ομαλή μετάβαση μεταξύ των σημείων που ορίζουν το άνω όριο της μη ορατής περιοχής.

Η συνάρτηση εκτελεί τα εξής βήματα:

1. Εντοπίζει τα δεξιά τελικά σημεία των μη ορατών αλυσίδων.

2. Βεβαιώνει ότι περιλαμβάνεται το τελευταίο σημείο της δοθείσας x-μονότονης πολυγωνικής γραμμής.

3. Ταξινομεί τα σημεία και υπολογίζει τις εφαπτόμενες.

4. Προσθέτει μια οριζόντια γέφυρα ή μια τελική εφαπτομένη προς το terrain.

Η συνάρτηση επιστρέφει δύο λίστες:

1. Λίστα άνω εφαπτόμενων Lr: Περιέχει τις εξισώσεις ευθειών (m,b) που ορίζουν το ανώτατο περίγραμμα.

2. Λίστα δεξιών άκρων right_end_points: Περιέχει τις κορυφές που ορίζουν το ανώτατο περίγραμμα.

Κώδικας:

```
def compute_upper_tangents(non_visible, terrain):
    right_dict = {}
    for seg in non_visible:
        pt = seg[1]
        if pt[0] in right_dict:
            if pt[1] > right_dict[pt[0]][1]:
                right_dict[pt[0]] = pt
        else:
            right_dict[pt[0]] = pt

    right_endpoints = list(right_dict.values())

    # Ensure the terrain's last point is included:
    p_end = terrain[-1]
    if not any(isclose(p_end[0], pt[0], abs_tol=1e-9) and isclose(p_end[1], pt[1],
abs_tol=1e-9)
               for pt in right_endpoints):
        right_endpoints.append(p_end)

    # Sort right endpoints by x (and if equal, by descending y)
    right_endpoints.sort(key=lambda p: (p[0], -p[1]))

    L_r = []
    # Compute tangents between consecutive right endpoints.
    for i in range(len(right_endpoints) - 1):
        r_i = right_endpoints[i]
        r_next = right_endpoints[i + 1]
        # Skip vertical segments (if x's are essentially the same)
        if not np.isclose(r_i[0], r_next[0], atol=1e-9):
            m = (r_next[1] - r_i[1]) / (r_next[0] - r_i[0])
            b = r_i[1] - m * r_i[0]
            L_r.append((m, b))

    # Get the last right endpoint:
    if right_endpoints:
        r_last = right_endpoints[-1]
        p_end = terrain[-1]
        # Instead of checking arbitrary atol differences, we compare the y-values.
        # If the terrain's end y is lower than r_last's y, then extend
horizontally.
        if p_end[1] < r_last[1]:
            # Extend horizontally at the last y value (bridge)
            L_r.append((0.0, r_last[1]))
            print("Added horizontal bridge: slope=0.0, intercept =", r_last[1])
        else:
            # Otherwise, add a tangent connecting r_last to the terrain end
            if not np.isclose(r_last[0], p_end[0], atol=1e-9):
```

```
                m = (p_end[1] - r_last[1]) / (p_end[0] - r_last[0])
                b = r_last[1] - m * r_last[0]
                L_r.append((m, b))
                print("Added tangent from r_last to terrain end: slope =", m,
"intercept =", b)
    return L_r, right_endpoints
```

3.3.2 Συνάρτηση split_and_reverse

Η συνάρτηση **split_and_reverse** διαχωρίζει το terrain σε δύο τμήματα – αριστερό και δεξιό – με βάση τη θέση του πρώτου παρατηρητηρίου. Τα σημεία στα αριστερά του παρατηρητηρίου επιστρέφονται αντεστραμμένα ώστε να διευκολύνεται η επεξεργασία των μη ορατών αλυσίδων.

Η συνάρτηση επιστρέφει τρεις τιμές:

- Μια λίστα σημείων που ανήκουν στο αριστερό τμήμα του terrain (με αντίστροφη σειρά).
- Μια λίστα σημείων που ανήκουν στο δεξιό τμήμα του terrain.
- Το σημείο του πρώτου παρατηρητηρίου.

Κώδικας:

```
def split_and_reverse(points, x_split):
    left = []
    right = []
    towerone = ()

    found = False
    for x, y in points:
        if x == x_split:
            found = True
            towerone = (x, y)
            continue
        if not found:
            left.append((x, y))
        else:
            right.append((x, y))

    return left[::-1], right, towerone
```

3.3.3 Συνάρτηση cross_product

Η συνάρτηση cross_product υπολογίζει το προσημασμένο εμβαδόν του τριγώνου που σχηματίζεται από τρία σημεία στο επίπεδο. Το αποτέλεσμα χρησιμοποιείται για τον καθορισμό της σχετικής θέσης των σημείων και αν ένα σημείο είναι "αριστερόστροφο", "δεξιόστροφο" ή συνευθειακά σε σχέση με τα υπόλοιπα.

Η συνάρτηση cross_product επιστρέφει έναν ακέραιο αριθμό, ο οποίος καθορίζει την γεωμετρική σχέση των σημείων:

- Θετικός αριθμός → Τα σημεία έχουν αριστερόστροφη διάταξη.

- Αρνητικός αριθμός → Τα σημεία έχουν δεξιόστροφη διάταξη.
- Μηδέν (0) → Τα σημεία είναι συνευθειακά (βρίσκονται στην ίδια ευθεία).

Κώδικας:

```
def cross_product(a, b, c):
    (ax, ay) = a
    (bx, by) = b
    (cx,cy) = c
    return (ax*by)-(ay*bx)+(ay*cx)-(ax*cy)+(bx*cy)-(by*cx)
```

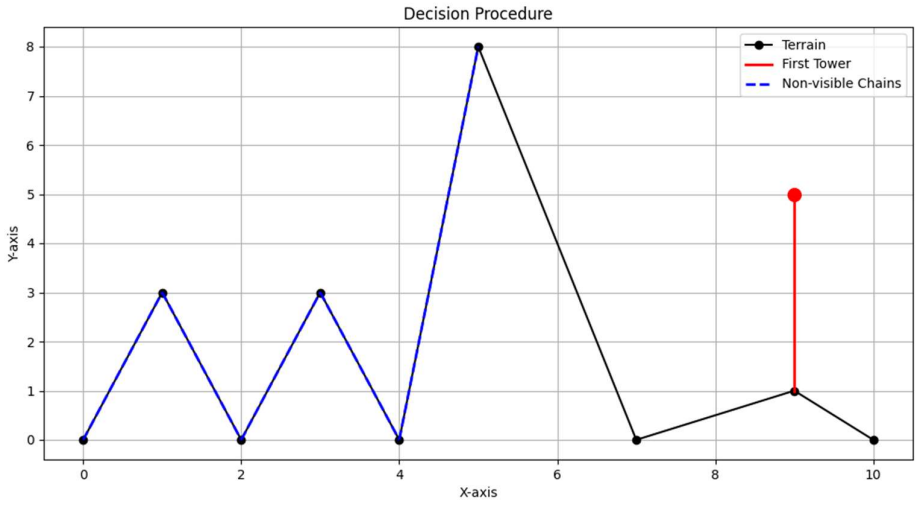
3.3.4 Παράδειγμα εξόδου

Για εισόδους:

terrain = [(0, 0), (1, 3), (2, 0), (3, 3), (4, 0), (5, 8), (7, 0), (9,1), (10,0)]
u_idx = 9
h1 = 4

Έξοδος (με κόκκινο χρώμα φαίνεται το πρώτο παρατηρητήριο ενώ οι χρωματισμένες μπλε γραμμές απεικονίζουν τις μη ορατές ακμές από αυτό):

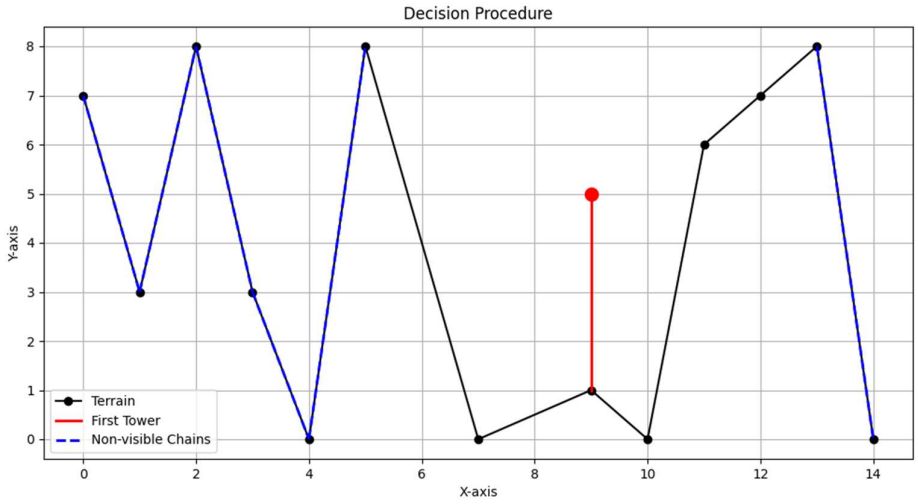
Non-visible Chains: [((4, 0), (5, 8)), ((3, 3), (4, 0)), ((2, 0), (3, 3)), ((1, 3), (2, 0)), ((0, 0), (1, 3))]



Για εισόδους:

terrain = [(0, 7), (1, 3), (2, 8), (3, 3), (4, 0), (5, 8), (7, 0), (9,1), (10,0),(11,6),(12,7),(13,8),(14,0)]
u_idx = 9
h1 = 4

Έξοδος:



3.4 Συνάρτηση compute_upper_envelope_discrete_gra\-ham_modified

Η συνάρτηση compute_upper_envelope_discrete_graham_modified υπολογίζει το Άνω Κάλυμμα (Upper Envelope) στη διακριτή εκδοχή του προβλήματος. Σε αντίθεση με την κλασική προσέγγιση, χρησιμοποιεί μια τροποποιημένη εκδοχή του Graham Scan, ώστε να επεξεργαστεί τις μη ορατές ακμές και να προσδιορίσει το ανώτερο περίγραμμα του terrain.

Ουσιαστικά, εκτελεί τα εξής βήματα:

1. Υπολογίζει τις εξισώσεις των μη ορατών ακμών και τις μετατρέπει σε δυαδικά σημεία.

2. Εφαρμόζει τον Graham Scan για να βρει το άνω κυρτό περίβλημα των δυαδικών σημείων.

3. Μετατρέπει πίσω το κυρτό περίβλημα στο primal επίπεδο και υπολογίζει το τελικό Άνω Κάλυμμα.

Η συνάρτηση επιστρέφει μια λίστα segments που αντιπροσωπεύουν το upper envelope.

Κώδικας:

```
def compute_upper_envelope_discrete_graham_modified(non_visible_segments, last,
terrain, L_r):
    from math import isclose  # make sure to import isclose
    domain_min, domain_max = -1.0, last + 1.0


    group = non_visible_segments

    all_env_pts = []
    lines = []
    # For each segment in our grouped segments, compute its line (slope,
```

```
intercept)
    for (x1, y1), (x2, y2) in group:
        if isclose(x2, x1, abs_tol=1e-9):
            continue  # skip vertical segments
        m = (y2 - y1) / (x2 - x1)
        b = y1 - m * x1
        lines.append((m, b))
    if not lines:
        return []


    lines.extend(L_r)

    # Convert lines to dual points (each line y = m*x + b becomes (m, -b))
    dual_points = [(m, -b) for (m, b) in lines]
    dual_points.sort(key=lambda p: p[0])
    hull_dual = graham_scan(dual_points)
    hull_lines = [(m, -negb) for (m, negb) in hull_dual]

    env_pts = []
    def line_intersection(m1, b1, m2, b2):
        denom = (m1 - m2)
        if isclose(denom, 0, abs_tol=1e-9):
            return None
        x_int = (b2 - b1) / denom
        y_int = m1 * x_int + b1

        return (x_int, y_int)

    for i in range(len(hull_lines) - 1):
        m1, b1 = hull_lines[i]
        m2, b2 = hull_lines[i+1]
        pt = line_intersection(m1, b1, m2, b2)
        print("Intersection point between lines:", pt)
        if pt is not None:
            env_pts.append(pt)

    print("Hull lines:", hull_lines)
    # Create envelope endpoints at the left and right ends of the group domain
    group_min = min(seg[0][0] for seg in group)
    group_max = max(seg[1][0] for seg in group)
    left_pt = (domain_min, hull_lines[0][0] * domain_min + hull_lines[0][1])
    right_pt = (domain_max, hull_lines[-1][0] * domain_max + hull_lines[-1][1])
    group_env = [left_pt] + env_pts + [right_pt]
    all_env_pts.extend(group_env)
    if not all_env_pts:
        return []
    all_env_pts.sort(key=lambda p: p[0])
    unique_x = sorted(set(x for (x, _) in all_env_pts))
    merged = []
    for x in unique_x:
        y_max = max(y for (xx, y) in all_env_pts if isclose(xx, x, abs_tol=1e-9)
or xx == x)
        merged.append((x, y_max))
    if merged[0][0] > domain_min:
        merged.insert(0, (domain_min, merged[0][1]))
    if merged[-1][0] < domain_max:
        merged.append((domain_max, merged[-1][1]))
    return merged
```

3.4.1 Συνάρτηση Graham_Scan

Η συνάρτηση **graham_scan** υπολογίζει το κυρτό περίβλημα (convex hull) ενός συνόλου σημείων χρησιμοποιώντας τον αλγόριθμο Graham Scan. Στο πρόβλημα των δύο παρατηρητηρίων,

χρησιμοποιείται για να βρούμε το άνω κάλυμμα (upper envelope), αφού πρώτα μετατρέψουμε τις μη ορατές ακμές σε δυαδικά σημεία.

Η συνάρτηση λειτουργεί σε δύο στάδια:

1. Ταξινόμηση των σημείων με βάση τις x-συντεταγμένες (και, σε περίπτωση ισοβαθμίας, τις y-συντεταγμένες).

2. Κατασκευή της κυρτής θήκης χρησιμοποιώντας το διανυσματικό γινόμενο (cross product) για να προσδιορίσει αν ένα σημείο πρέπει να διατηρηθεί ή να αφαιρεθεί από το αποτέλεσμα.

Η συνάρτηση επιστρέφει μια λίστα από σημεία που αποτελούν το κυρτό περίβλημα του συνόλου των εισαγόμενων σημείων

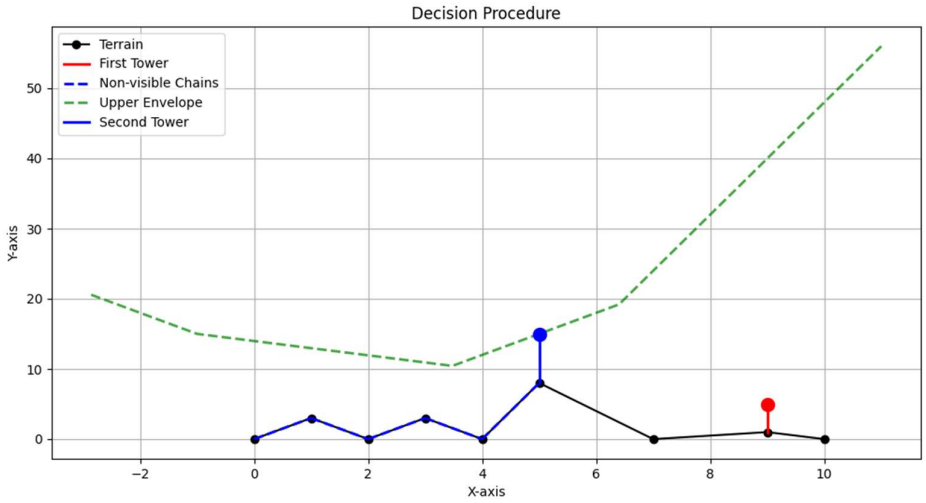
Κώδικας:

```
def graham_scan(points):
    if len(points) <= 2:
        return points
    points.sort(key=lambda p: (p[0], p[1]))
    lower = []
    for p in points:
        while len(lower) >= 2 and (
                (lower[-1][0] - lower[-2][0]) * (p[1] - lower[-2][1]) - (lower[-
1][1] - lower[-2][1]) * (
                p[0] - lower[-2][0])) <= 0:
            lower.pop()
        lower.append(p)
    return lower
```

3.4.2 Παράδειγμα εξόδου

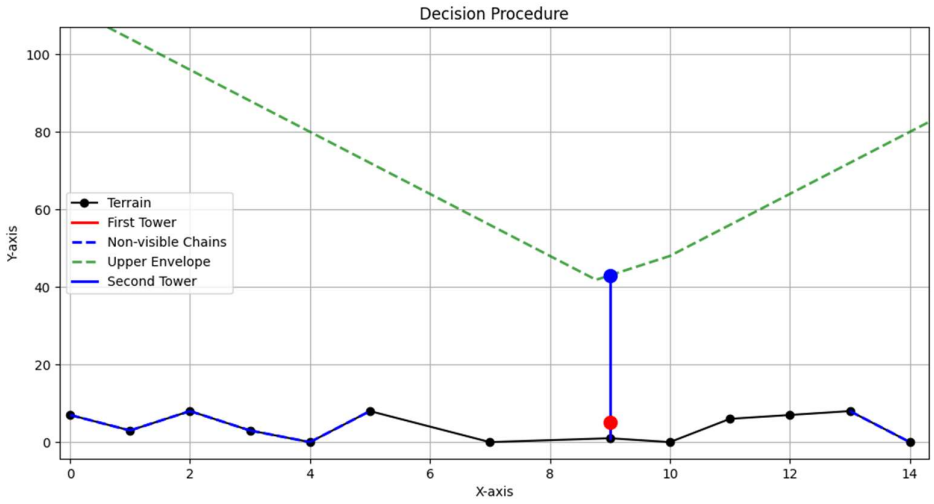
Είσοδο ιδιά με 3.3

Έξοδος Upper Envelope: [(-2.857142857142857, 20.57142857142857), (-1.0, 15.0), (3.4782608695652177, 10.434782608695652), (6.4, 19.200000000000003), (11.0, 56.0)]



Είσοδος ίδια με 3.3.

Έξοδος Upper Envelope: [(-1.0, 120.0), (8.76923076923077, 41.84615384615384), (10.0, 48.0), (15.0, 88.0)]



3.5 Αλγόριθμος παραμετρικής αναζήτησης

Η συνάρτηση parametric_search εκτελεί την παραμετρική αναζήτηση για τον υπολογισμό του ελάχιστου απαιτούμενου ύψους h2 του δεύτερου παρατηρητηρίου. Διατρέχει τα σημεία του upper envelope και, για κάθε σημείο, υπολογίζει την απαιτούμενη κατακόρυφη απόσταση ώστε όλες οι μη ορατές ακμές του terrain να γίνουν ορατές. Το σημείο με τη μικρότερη τέτοια απόσταση καθορίζει τη βέλτιστη τοποθεσία και το ελάχιστο ύψος του δεύτερου παρατηρητηρίου.

Η συνάρτηση επιστρέφει ένα ζεύγος (tuple) που αποτελείται από το ελάχιστο ύψος h2 και την τοποθεσία (σε μορφή Point) του δεύτερου παρατηρητηρίου.

Κώδικας:

```
def parametric_search(terrain, envelope):

    min_h2 = float('inf')
    best_location = None

    # Iterate over each terrain vertex (discrete candidate location for the second
tower)
    for vx, ground in terrain:
        # Evaluate the envelope at this x-coordinate:
        env_y = evaluate_envelope(envelope, vx)
        # The extra height required is the gap between envelope and ground:
        required_h2 = max(0, env_y - ground)


        if required_h2 <= min_h2:
            min_h2 = required_h2
            best_location = (vx, ground)
    return round(min_h2), best_location
```

3.5.1 Συνάρτηση Evaluate_Envelope

Η συνάρτηση evaluate_envelope υπολογίζει την τιμή του Άνω Καλύμματος (Upper Envelope) σε μια δεδομένη τιμή x, χρησιμοποιώντας γραμμική παρεμβολή (linear interpolation) μεταξύ των αποθηκευμένων σημείων του Άνω Καλύμματος.

Συγκεκριμένα:

1. Αν το x βρίσκεται εκτός του ορισμένου εύρους του Άνω Καλύμματος, επιστρέφει το πλησιέστερο άκρο.

2. Αν το x βρίσκεται εντός του ορισμένου εύρους, εντοπίζει τα δύο γειτονικά σημεία του Άνω Καλύμματος και υπολογίζει την τιμή του χρησιμοποιώντας γραμμική παρεμβολή.

Η συνάρτηση επιστρέφει το y-συντεταγμένο ύψος του Άνω Καλύμματος για την τιμή x.

Κώδικας:

```
def evaluate_envelope(envelope, x):

    if x <= envelope[0][0]:
        return envelope[0][1]
    if x >= envelope[-1][0]:
        return envelope[-1][1]

    for i in range(len(envelope) - 1):
        x1, y1 = envelope[i]
        x2, y2 = envelope[i + 1]
        if x1 <= x <= x2:
            # Linear interpolation:
            t = (x - x1) / (x2 - x1)
            return y1 + t * (y2 - y1)
    # Fallback (should not happen if envelope is non-empty)
    return envelope[-1][1]
```

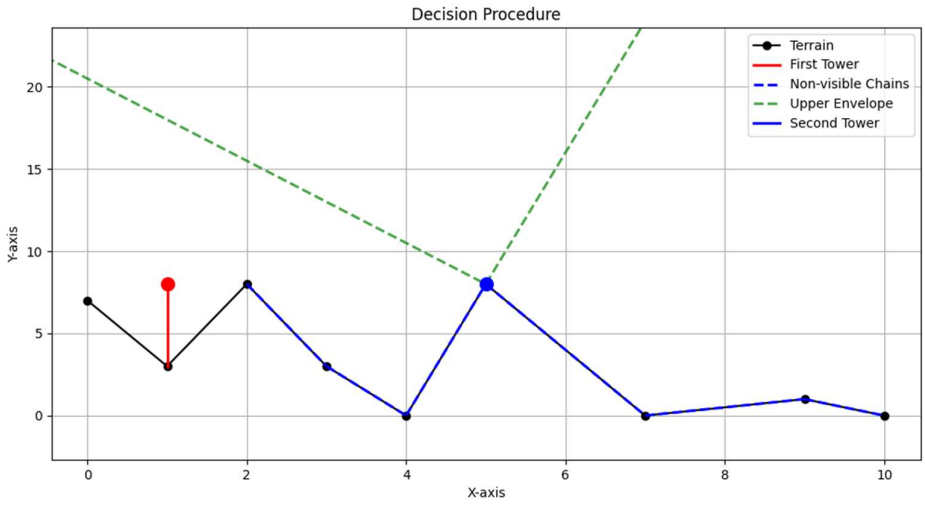
3.5.2 Παράδειγμα εξόδου

Είσοδος

terrain = [(0, 7), (1, 3), (2, 8), (3, 3), (4, 0), (5, 8), (7, 0), (9, 1), (10, 0)]
u_idx = 1
h1 = 5

Έξοδος

Optimal height of second tower: 0, Location: (5, 8)



3.6 Αλγόριθμος αναζήτησης για την Ημι-συνεχή Εκδοχή

Η συνάρτηση **semi_search** εκτελεί την ημι-συνεχή αναζήτηση (semi-continuous search) για την εύρεση του ελάχιστου ύψους του δεύτερου παρατηρητηρίου. Σε αντίθεση με την διακριτή εκδοχή, όπου το δεύτερο παρατηρητήριο μπορεί να τοποθετηθεί μόνο σε κορυφές του terrain, εδώ μπορεί να τοποθετηθεί και πάνω σε ακμές, επιτρέποντας μεγαλύτερη ευελιξία στη βέλτιστη επιλογή της τοποθεσίας.

Η μέθοδος λειτουργεί ως εξής:

1. Εύρεση όλων των πιθανών σημείων για την τοποθέτηση του δεύτερου παρατηρητηρίου (κορυφές και ενδιάμεσα σημεία σε ακμές).

2. Καθορισμός των ορίων του upper envelope, δηλαδή της ελάχιστης και μέγιστης x-συντεταγμένης.

3. Εξερεύνηση κάθε πιθανού σημείου και υπολογισμός του ύψους που απαιτείται ώστε το σημείο να είναι ορατό από το πρώτο παρατηρητήριο.

4. Επιλογή του σημείου με το ελάχιστο απαιτούμενο ύψος h2.

Η συνάρτηση επιστρέφει ένα tuple που περιέχει:

- Το ελάχιστο απαιτούμενο ύψος h2 του δεύτερου παρατηρητηρίου.
- Την καλύτερη τοποθεσία (x, y) για το δεύτερο παρατηρητήριο.

Κώδικας:

```
def parametric_search_semi(terrain, envelope):

    candidates = set()
    for (x, _) in terrain:
        candidates.add(x)
    for (x, _) in envelope:
        candidates.add(x)

    candidates.add(terrain[0][0])
    candidates.add(terrain[-1][0])
    candidates = sorted(candidates)

    min_h2 = float('inf')
    best_location = None
    for x in candidates:
        y_terrain = evaluate_terrain(terrain, x)
        y_envelope = evaluate_envelope(envelope, x)
        required_h2 = max(0, y_envelope - y_terrain)
        if required_h2 < min_h2:
            min_h2 = required_h2
            best_location = (x, y_terrain)
    return min_h2, best_location
```

3.6.1 Evaluate Terrain

Η συνάρτηση evaluate_terrain υπολογίζει το ύψος του terrain για μια δεδομένη τιμή x, χρησιμοποιώντας γραμμική παρεμβολή (linear interpolation) μεταξύ των σημείων του εδάφους.

Η διαδικασία περιλαμβάνει:

1. Έλεγχο των οριακών τιμών: Αν το x βρίσκεται εκτός του terrain, επιστρέφεται η πλησιέστερη διαθέσιμη τιμή.
2. Εύρεση του κατάλληλου διαστήματος στο terrain: Εντοπίζονται τα δύο σημεία του terrain μεταξύ των οποίων ανήκει το x.
3. Εφαρμογή γραμμικής παρεμβολής: Υπολογίζεται το ύψος y στο σημείο x με βάση τα δύο γειτονικά σημεία.

Η συνάρτηση επιστρέφει το ύψος y του terrain στη θέση x.

Κώδικας:

```
def evaluate_terrain(terrain, x):

    if x <= terrain[0][0]:
        return terrain[0][1]
    if x >= terrain[-1][0]:
```

```
    return terrain[-1][1]
for i in range(len(terrain) - 1):
    x1, y1 = terrain[i]
    x2, y2 = terrain[i + 1]
    if x1 <= x <= x2:
        t = (x - x1) / (x2 - x1)
        return y1 + t * (y2 - y1)
return terrain[-1][1]
```

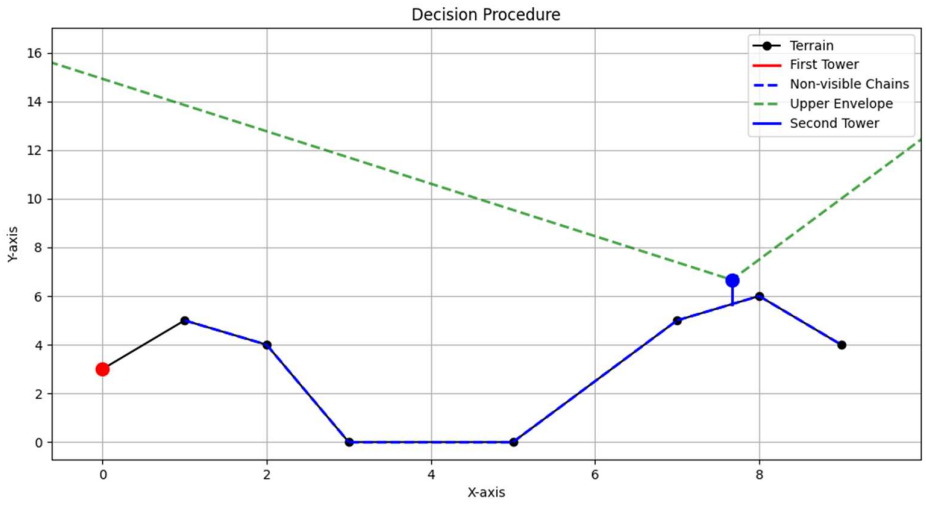
3.6.1 Παράδειγμα εξόδου

Είσοδος

terrain = [(0, 3), (1, 5), (2, 4), (3, 0), (5, 0), (7, 5), (8, 6), (9, 4)]
u_idx = 0
h1 = 0

Έξοδος

Optimal h2: 0.9999999999999991, Location: (7.666666666666667, 5.666666666666667)



3.7 Αλγόριθμος χωρίς παραμετρική αναζήτηση

Η συνάρτηση two_watchtowers_min_height_envelope υπολογίζει το ελάχιστο δυνατό κοινό ύψος για δύο παρατηρητήρια, έτσι ώστε να καλύπτουν ολόκληρο το terrain.

Η διαδικασία περιλαμβάνει:

1. Υπολογισμό των Δέντρων Συντομότερων Διαδρομών (SPT) για κάθε κορυφή του terrain.

2. Εξαγωγή κρίσιμων υψών τύπου 1 από τα SPT, τα οποία αντιστοιχούν σε αλλαγές στην ορατότητα.

3. Δοκιμή όλων των πιθανών ζευγών κορυφών (u,v) του terrain ως υποψήφιες θέσεις παρατηρητηρίων.

4. Υπολογισμό του ελάχιστου ύψους h που απαιτείται για το ζευγάρι (u,v) μέσω του Upper Envelope.

5. Επιστροφή του βέλτιστου ύψους h και του αντίστοιχου ζευγαριού (u,v).

Η συνάρτηση επιστρέφει δύο τιμές:

1. Το ελάχιστο κοινό ύψος h για τα δύο παρατηρητήρια.

2. Το καλύτερο ζευγάρι κορυφών (u,v) όπου πρέπει να τοποθετηθούν τα παρατηρητήρια.

Κώδικας:

```
def two_watchtowers_min_height_envelope(terrain: List[Point]) -> Tuple[float,
Tuple[int, int]]:
    n = len(terrain)
    SPT = compute_all_SPTs(terrain)
    type1 = compute_type1_from_SPT(SPT)
    best_h = float('inf')
    best_pair = (-1, -1)
    for u in range(n):
        for v in range(u + 1, n):
            h_pair = find_min_height_for_pair_envelope(terrain, SPT, type1, u, v)
            if h_pair < best_h:
                best_h = h_pair
                best_pair = (u, v)
    return best_h, best_pair
```

3.7.1 Συνάρτηση compute_SPT_upper_envelope

Η συνάρτηση compute_SPT_upper_envelope υπολογίζει το Άνω Κάλυμμα (Upper Envelope) για το Δέντρο Συντομότερων Διαδρομών (SPT) ενός δοσμένου σημείου u στο terrain.

Η διαδικασία περιλαμβάνει:

1. Σάρωση των κορυφών στα δεξιά του u και υπολογισμό της συνεισφοράς τους στο Άνω Κάλυμμα.

2. Σάρωση των κορυφών στα αριστερά του u χρησιμοποιώντας μια προσαρμοσμένη έκδοση του Graham Scan για να διατηρηθεί η κυρτή θήκη.

3. Επιστροφή ενός πίνακα S που περιέχει το επιπλέον ύψος (visibility gap) που απαιτείται για την κάλυψη κάθε κορυφής από το u.

Η συνάρτηση επιστρέφει μια λίστα S που περιέχει τις επιπλέον απαιτήσεις ύψους για να είναι το u ορατό από κάθε κορυφή.(Type 1 critical heights)

Κώδικας:

```
def compute_SPT_upper_envelope(terrain: List[Point], u: int) -> List[float]:
    n = len(terrain)
    u_point = terrain[u]
    S = [0.0] * n

    # Process vertices to the right of u
    envelope = []  # (x_start, m, b)
    for w in range(u + 1, n):
        xw = terrain[w][0]
        m_direct, b_direct = line_from_points(u_point, terrain[w])
        extra = 0.0
        # Check existing envelope segments
        for seg in envelope:
            x_start, m_seg, b_seg = seg
            if xw >= x_start:
                y_seg = m_seg * xw + b_seg
                y_direct = m_direct * xw + b_direct
                extra = max(extra, y_seg - y_direct)
        S[w] = extra
        # Update envelope with the new line (u(h) to w)
        new_line = line_from_points((u_point[0], u_point[1] + S[w]), terrain[w])
        # Prune segments dominated by new_line
        while envelope:
            last_seg = envelope[-1]
            x_start, m_last, b_last = last_seg
            inter = intersection_line(new_line[0], new_line[1], m_last, b_last)
            if inter is None or inter[0] <= x_start + 1e-9:
                envelope.pop()
            else:
                break
        envelope.append((xw, new_line[0], new_line[1]))

    # Process vertices to the LEFT of u using a reversed convex hull approach
    envelope_left = []  # (x_start, m, b), valid for x <= x_start
    for w in range(u - 1, -1, -1):
        xw = terrain[w][0]
        m_direct, b_direct = line_from_points(u_point, terrain[w])
        extra = 0.0
        # Check existing envelope_left segments
        for seg in envelope_left:
            x_start, m_seg, b_seg = seg
            if xw <= x_start:
                y_seg = m_seg * xw + b_seg
                y_direct = m_direct * xw + b_direct
                extra = max(extra, y_seg - y_direct)
        S[w] = extra
        # Update envelope_left with the new line (u(h) to w)
        new_line = line_from_points((u_point[0], u_point[1] + S[w]), terrain[w])
        # Prune segments dominated by new_line (reverse logic)
        while envelope_left:
            last_seg = envelope_left[-1]
            x_start, m_last, b_last = last_seg
            inter = intersection_line(new_line[0], new_line[1], m_last, b_last)
            if inter is None or inter[0] >= x_start - 1e-9:
                envelope_left.pop()
            else:
                break
        envelope_left.append((xw, new_line[0], new_line[1]))

    return S
```

3.7.2 Συναρτήσεις για Type 2 heights

Αυτή η συνδυασμένη συνάρτηση υπολογίζει τα κρίσιμα ύψη τύπου 2, τα οποία είναι τα ύψη στα οποία ένα ζεύγος παρατηρητηρίων (u,v) μπορεί να καλύψει πλήρως μια ακμή του terrain.

Η διαδικασία περιλαμβάνει:

1. Επίλυση του προβλήματος συνευθειακότητας (solve_joint_visibility_exact) μεταξύ των σημείων που συνδέουν τα δύο παρατηρητήρια.

2. Υπολογισμό των κρίσιμων υψών για κάθε ακμή (compute_type2_for_edge_exact).

3. Συγκέντρωση και ταξινόμηση όλων των κρίσιμων υψών για τις ακμές μεταξύ u και v (compute_type2_heights_exact).

Οι συναρτήσεις μαζί επιστέφουν μία λίστα από κρίσιμα ύψη τύπου 2.

Κώδικας:

```
def solve_joint_visibility_exact(h_a: float, h_b: float, h_pa: float, h_pb: float)
-> Optional[float]:

    A = (h_b - h_a) - (h_pb - h_pa)
    B = 2 * (h_a * (h_pb - h_pa) - h_pa * (h_b - h_a))
    C = h_a ** 2 - h_pa ** 2
    # Solve quadratic A*h^2 + B*h + C = 0, take the largest root in (h_b, h_a)
    if isclose(A, 0, abs_tol=1e-12):
        # Fall back to linear solution.
        if isclose((h_b - h_a) - (h_pb - h_pa), 0, abs_tol=1e-12):
            return None
        t = (h_pa - h_a) / ((h_b - h_a) - (h_pb - h_pa))
        candidate = h_a + t * (h_b - h_a)
        return candidate if 0 < t < 1 else None
    disc = B * B - 4 * A * C
    if disc < 0:
        return None
    r1 = (-B + sqrt(disc)) / (2 * A)
    r2 = (-B - sqrt(disc)) / (2 * A)
    candidate = max(r1, r2)
    if h_b < candidate < h_a:
        return candidate
    return None

def compute_type2_for_edge_exact(H_u_a: float, H_u_b: float, H_v_a: float, H_v_b:
float) -> Optional[float]:
    if H_u_a <= H_v_a:
        return H_u_a
    if H_u_b >= H_v_b:
        return H_v_b
    sol = solve_joint_visibility_exact(H_u_a, H_u_b, H_v_a, H_v_b)
    return sol


def compute_type2_heights_exact(SPT: List[List[float]], u: int, v: int) ->
List[float]:

    crits = set()
    if u > v:
        u, v = v, u
    for p in range(u, v):
        H_u_a = SPT[u][p]
        H_u_b = SPT[u][p + 1]
```

```
        H_v_a = SPT[v][p]
        H_v_b = SPT[v][p + 1]
        crit = compute_type2_for_edge_exact(H_u_a, H_u_b, H_v_a, H_v_b)
        if crit is not None:
            crits.add(crit)
    return sorted(crits)
```

3.7.3 Συνάρτηση is_guarded_envelope

Η συνάρτηση is_guarded_envelope ελέγχει αν δύο παρατηρητήρια που τοποθετούνται στα σημεία u και v με ύψος hhh είναι αρκετά για να επιβλέπουν ολόκληρο το terrain.

Η απόφαση βασίζεται στο Άνω Κάλυμμα (Upper Envelope) και στον έλεγχο της ορατότητας κάθε ακμής του terrain, λαμβάνοντας υπόψη:

1. Τις συνθήκες ορατότητας σε σχέση με τα Δέντρα Συντομότερων Διαδρομών (SPT).

2. Τις απαιτήσεις ύψους που προκύπτουν από τον τύπο 2 κρίσιμων υψών.

3. Έλεγχο ότι οι κορυφές που βρίσκονται μεταξύ u και v είναι επίσης ορατές.

Η συνάρτηση επιστρέφει Boolean:

- True αν το terrain καλύπτεται πλήρως από τα δύο παρατηρητήρια.

- False αν υπάρχει έστω και μια ακμή που δεν επιβλέπεται.

Κώδικας:

```
def is_guarded_envelope(terrain: List[Point], SPT: List[List[float]], u: int, v:
int, h: float) -> bool:
    n = len(terrain)

    def edge_required(p: int) -> float:
        # For edge from p to p+1, if one tower does not cover it, compute the
joint requirement.
        # Use the type 2 computation.
        return compute_type2_for_edge_exact(SPT[u][p], SPT[u][p + 1], SPT[v][p],
SPT[v][p + 1])

    for p in range(n - 1):
        # Edges fully left of u:
        if p + 1 <= u:
            if max(SPT[u][p], SPT[u][p + 1]) > h:
                return False
        # Edges fully right of v:
        elif p >= v:
            if max(SPT[v][p], SPT[v][p + 1]) > h:
                return False
        else:
            # For edges between u and v:
            cover_u = max(SPT[u][p], SPT[u][p + 1]) <= h
            cover_v = max(SPT[v][p], SPT[v][p + 1]) <= h
            if cover_u or cover_v:
                continue
            joint = edge_required(p)
            if joint is None or h < joint - 1e-9:
```

```
            return False
    return True
```

3.7.4 Παράδειγμα εξόδου

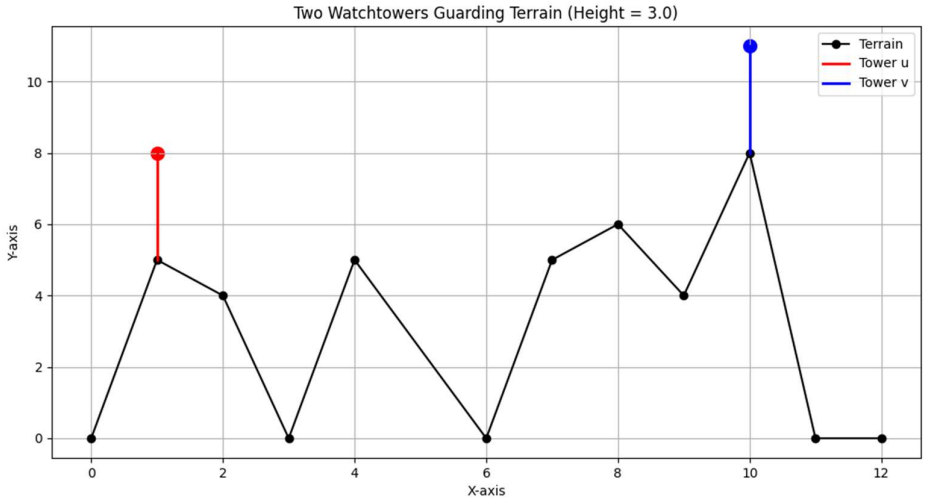
Είσοδος

terrain= [(0, 0), (1, 5), (2, 4), (3, 0),(4,5),(6,0) ,(7, 5), (8, 6), (9, 4),(10,8),(11,0),(12,0)]

Έξοδος

Optimal minimum height for towers: 3.0

Best pair of vertices (tower positions): (1, 9)



Κεφάλαιο 4: Επίλογος

Στην παρούσα εργασία μελετήσαμε το πρόβλημα των δύο παρατηρητηρίων, το οποίο αφορά τον υπολογισμό της βέλτιστης θέσης και του ελαχίστου ύψους δύο παρατηρητηρίων ώστε να καλύπτουν πλήρως μια x-μονότονη πολυγωνική γραμμή. Παρουσιάσαμε μια ολοκληρωμένη προσέγγιση δίνοντας τη μαθηματική της θεμελίωση, την αλγοριθμική της σχεδίαση και την πρακτική της υλοποίηση.

Αρχικά, εντοπίσαμε τις μη ορατές αλυσίδες της x-μονότονης πολυγωνικής γραμμής χρησιμοποιώντας τη μέθοδο του προσημασμένου εμβαδού τριγώνου, επιτρέποντας τον προσδιορισμό των ακμών που δεν είναι πλήρως ορατές από το πρώτο παρατηρητήριο. Στη συνέχεια, εφαρμόσαμε γεωμετρική δυϊκότητα για τη μετατροπή αυτών των ακμών σε δυϊκά σημεία και χρησιμοποιήσαμε τον αλγόριθμο Graham Scan για την κατασκευή του κάτω κυρτού περιβλήματος από το οποίο υπολογίσαμε το άνω κάλυμμα των φορέων αυτών των ακμών.

Έτσι παρουσιάσαμε μια διαδικασία απόφασης για τη διακριτή εκδοχή η οποία, δοθέντων της θέσης και του ύψους του πρώτου παρατηρητηρίου, υπολογίζει τη θέση και το μικρότερο ύψος για το δεύτερο παρατηρητήριο. Κατόπιν επεκτείναμε αυτή τη διαδικασία απόφασης σε μια διαδικασία απόφασης για την ημι-συνεχή εκδοχή. Τέλος, παρουσιάσαμε έναν ολοκληρωμένο αλγόριθμο για τη διακριτή εκδοχή του προβλήματος των δύο παρατηρητηρίων.

Οι παραπάνω αλγόριθμοι υλοποιήθηκαν στη γλώσσα προγραμματισμού python και δώσαμε αναλυτικές περιγραφές συναρτήσεων και παραδείγματα εκτέλεσης για καθέναν από αυτούς.

Βιβλιογραφία

[P25] L. Palios, “On the Discrete and Semi-Continuous Versions of the Two-Watchtower Problem in the Plane”, in Proc. CIAC 2025, 2025.

[HP22] S. Har-Peled, "Duality", 2022. Διαθέσιμο από: har-peled_ch31_duality_2022.pdf

[Zh97] B. Zhu, "Computing the Shortest Watchtower of a Polyhedral Terrain in O(n log n) Time," Computational Geometry 8, pp. 181–193, 1997.

[Urr00] J. Urrutia, "Art Gallery and Illumination Problems," 2000. Διαθέσιμο από: urrutia2000.pdf

[BCWZ01] S. Bespamyatnikh, Z. Chen, K. Wang, and B. Zhu, "On the Planar Two-Watchtower Problem," Information Processing Letters 89, pp. 137–139, 2001.

[GD19] L. Gewali and B. Dahal, "Algorithms for Tower Placement on Terrain," 2019. Διαθέσιμο από: 10.1007@978-3-030-14070-077.pdf

[BMCK04] B. Ben-Moshe, P. Carmi, and M. J. Katz, "Computing All Large Sums-of-Pairs in R^n and the Discrete Planar Two-Watchtower Problem," Information Processing Letters 89, pp. 137–139, 2004.

[CV07] K. L. Clarkson and K. Varadarajan, "Improved Approximation Algorithms for Geometric Set Cover," Discrete & Computational Geometry 37, pp. 43–58, 2007.

[ABDKNSZ09] P. K. Agarwal, S. Bereg, O. Daescu, H. Kaplan, S. Ntafos, M. Sharir, and B. Zhu, "Guarding a Terrain by Two Watchtowers," Algorithmica 58, pp. 352–390, 2009.

[BCKO08] M. de Berg, O. Cheong, M. J. van Kreveld, and M. H. Overmars, Computational Geometry: Algorithms and Applications, 3rd ed., Santa Clara, CA, USA: Springer, 2008.

[KDM16] F. Khodakarami, F. Didehvar, and A. Mohades, "1.5D Terrain Guarding Problem Parameterized by Guard Range," Theoretical Computer Science 661, pp. 65–69, 2016.

[K06] J. King, "A 4-Approximation Algorithm for Guarding 1.5-Dimensional Terrains," in LNCS 3887, pp. 629–640, 2006.

[Sh88] M. Sharir, "The Shortest Watchtower and Related Problems for Polyhedral Terrains," Information Processing Letters 29, pp. 265–270, 1988.

[Ed85] H. Edelsbrunner, "Computing the Extreme Distances between Two Convex Polygons," Journal of Algorithms 6, pp. 213–224, 1985.