

**Αλγόριθμοι και Στοιχεία Πολυπλοκότητας.**

**Ενότητα 3:** Βελτιωμένη Μέθοδος Υπολογισμού Δισδιάστατων Μεγίστων (Plane – Sweep Algorithm).

Διδάσκων: Ηλίας Κ Σάββας, Αναπληρωτής Καθηγητής.

Τμήμα Μηχανικών Πληροφορικής, Τεχνολογικής Εκπαίδευσης.

**Άδειες χρήσης.**

* Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons (C C). **Αναφορά δημιουργού (B Y), Μη εμπορική χρήση (N C), Μη τροποποίηση (N D), 3.0, Μη εισαγόμενο.**
* Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.



**Χρηματοδότηση.**

* Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
* Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.



# Σκοποί ενότητας.

Ο αναγνώστης να μπορεί να:

1) βελτιώνει ήδη λυμένα προβλήματα, χρησιμοποιώντας ειδικές τεχνικές,

2) υπολογίζει με ακρίβεια την πολυπλοκότητά τους.

ΠΕΡΙΕΧΌΜΕΝΑ ΕΝΌΤΗΤΑΣ.

[Σκοποί ενότητας. 2](#_Toc368170910)

[ΑΛΓΌΡΙΘΜΟΙ 3](#_Toc368170911)

[ΠΡΟΓΡΆΜΜΑΤΑ 3](#_Toc368170912)

[ΣΧΉΜΑΤΑ 3](#_Toc368170913)

[3. ΒΕΛΤΙΩΜΈΝΗ ΜΈΘΟΔΟΣ ΥΠΟΛΟΓΙΣΜΟΎ ΔΙΣΔΙΆΣΤΑΤΩΝ ΜΕΓΊΣΤΩΝ (Plane–Sweep Algorithm). 3](#_Toc368170914)

[3.1 Αλγόριθμος σάρωσης του επιπέδου (Plane–Sweep algorithm). 4](#_Toc368170915)

[3.2 Πολυπλοκότητα του αλγόριθμου σάρωσης του επιπέδου. 6](#_Toc368170916)

[3.3 Ασκήσεις. 6](#_Toc368170917)

# ΑΛΓΌΡΙΘΜΟΙ.

|  |  |
| --- | --- |
| ***Αλγόριθμος 1****: Βελτιωμένος αλγόριθμος επίλυσης του προβλήματος εύρεσης μέγιστων δισδιάστατου χώρου (Sweep – Plane).*  | 6 |

# ΠΡΟΓΡΆΜΜΑΤΑ.

|  |  |
| --- | --- |
| ***Πρόγραμμα 1****: Πρόγραμμα υλοποίησης αλγόριθμου σάρωσης επιπέδου.* | 7 |

# ΣΧΉΜΑΤΑ.

|  |  |
| --- | --- |
| ***Σχήμα 3.1*** *Βελτιωμένος αλγόριθμος υπολογισμού δισδιάστατων μεγίστων.* | 5 |

# 3. ΒΕΛΤΙΩΜΈΝΗ ΜΈΘΟΔΟΣ ΥΠΟΛΟΓΙΣΜΟΎ ΔΙΣΔΙΆΣΤΑΤΩΝ ΜΕΓΊΣΤΩΝ (Plane–Sweep Algorithm).

Όπως έχει ήδη αποδειχθεί, η προηγούμενη μέθοδος (brute-force), υπολογισμού δισδιάστατων μεγίστων έχει πολυπλοκότητα O(n2). Το ερώτημα που τίθεται είναι εάν υπάρχει μία πιο γρήγορη μέθοδος. Το πρόβλημα με τον brute-force αλγόριθμο, είναι ότι εξετάζονται όλα τα ζεύγη σημείων, όπου ακόμη και στην περίπτωση που ένα σημείο ήδη έχει βρεθεί, ότι δεν είναι μέγιστο θα εξεταστεί ξανά και ξανά, μέχρι να τελειώσουν όλοι οι πιθανοί συνδυασμοί. Επομένως, είναι ένας μη ευφυής αλγόριθμος, που στηρίζεται στη πλήρη και πολλαπλή διερεύνηση του χώρου λύσεων. Εάν ένα σημείο px, καλύπτεται από ένα άλλο έστω py, τότε, το px δεν θα έπρεπε να επανεξεταστεί με κανένα άλλο σημείο. Η παρατήρηση αυτή, μπορεί να οδηγήσει σε μια καλύτερη επίλυση του προβλήματος. Εάν βέβαια όλα τα υπό εξέταση σημεία, τύχει να είναι μέγιστα, τότε αυτή η παρατήρηση δεν οδηγεί σε πιο γρήγορη λύση, αλλά αυτή αποτελεί μια ακραία περίπτωση.

## 3.1 Αλγόριθμος σάρωσης του επιπέδου (Plane–Sweep algorithm).

Ο αλγόριθμος σάρωσης του επιπέδου, στηρίζεται στην προαναφερθείσα παρατήρηση, ότι δηλαδή, εάν κατά την σύγκριση δύο σημείων κάποιο υπερκαλύπτει το άλλο, τότε αυτό το άλλο σημείο δεν πρέπει να επανεξετασθεί. Η τεχνική σάρωσης του επιπέδου λειτουργεί ως εξής: Με μία κάθετη γραμμή σαρώνεται το επίπεδο από αριστερά προς τα δεξιά, κρατώντας τα μέγιστα σημεία που βρίσκονται αριστερά της γραμμής. Όταν η γραμμή φθάσει στο τελευταίο (δεξιότερο) σημείο του επιπέδου, τότε θα έχουν βρεθεί όλα τα μέγιστα.

Αν και φαίνεται ότι η γραμμή κινείται κατά ένα συνεχή τρόπο, στην πραγματικότητα πρέπει να κινείται με διακριτά βήματα. Για να επιτευχθεί αυτό, αρχικά ταξινομούνται τα σημεία σε αύξουσα διάταξη σε σχέση με τις τετμημένες τους. Στη συνέχεια εφαρμόζεται η σάρωση του επιπέδου μετακινώντας την γραμμή από σημείο σε σημείο. Φυσικά, η ταξινόμηση επιφέρει ένα επιπλέον κόστος στο όλο πρόβλημα, το οποίο όμως είναι της τάξης , εάν χρησιμοποιηθεί κάποια καλή μέθοδος ταξινόμησης, όπως η quick sort. Επομένως, το πρόβλημα ανάγεται μόνο στο πως θα κρατούνται τα μέγιστα, και πως θα ενημερώνονται όταν ένα νέο μέγιστο ανακαλύπτεται. Κατ’ αρχήν, υποτίθεται ότι κάθε νέο στοιχείο που εξετάζεται είναι εν δυνάμει μέγιστο, μιας και η τετμημένη του είναι μεγαλύτερη από όλα τα άλλα, (έχει ήδη ταξινομηθεί το σύνολο των σημείων). Το μοναδικό πρόβλημα είναι μήπως αυτό το σημείο που εξετάζεται, είναι μεγαλύτερο από κάποιο από τα μέγιστα που έχουν βρεθεί μέχρι τώρα, και το οποίο θα πρέπει να αφαιρεθεί από το σύνολο των μεγίστων. Φυσικά, κάθε σημείο που βρίσκεται μικρότερο από κάποιο άλλο δεν πρέπει να επανεξετασθεί.

Όπως φαίνεται στο *Σχήμα 3.1*, ενώ πριν εξετασθεί το σημείο (12, 12), τα μέγιστα σημεία είναι τα (7, 13), τα (9, 10), και τα (11, 5), μετά την εξέταση του (12, 12), μέγιστα είναι τα (7, 13), και τα (12, 12).



***Σχήμα 3.1*** *Βελτιωμένος αλγόριθμος υπολογισμού δισδιάστατων μεγίστων.*

*Ανάλυση:* Εάν υποτεθεί ότι αυτή τη στιγμή εξετάζεται το σημείο p. Σημειωτέον ότι, αφού το p.x είναι μεγαλύτερο από όλα όσα έχουν εξετασθεί μέχρι τώρα, τότε το p για να αποτελέσει μέγιστο αρκεί να εξετασθεί μόνο αν η τεταγμένη του, είναι μεγαλύτερη ή ίση από όλα τα μέχρι τώρα μέγιστα. Επομένως, από όλα τα μέχρι τώρα μέγιστα αρκεί να βρεθούν αυτά με μικρότερη ή ίση τεταγμένη, και στη συνέχεια να αφαιρεθούν από το σύνολο των μεγίστων.

Εδώ, είναι χρήσιμη η ακόλουθη παρατήρηση. Όπως διαβάζονται τα μέγιστα σημεία από αριστερά προς τα δεξιά, και οι τετμημένες είναι σε αύξουσα διάταξη, οι τεταγμένες τους μικραίνουν (φθίνουσα διάταξη). Αυτό συμβαίνει γιατί, εάν υποτεθεί ότι υπάρχουν δύο μέγιστα, έστω p και q, με p.x ≥ q.x, αλλά p.y ≥ q.y, τότε θα έπρεπε το p να μην είναι μέγιστο.

Για να γίνει λοιπόν ο υπολογισμός των μεγίστων αρκεί να σαρωθεί το επίπεδο γραμμικά. Η επόμενη και τελευταία ερώτηση είναι η φορά της σάρωσης, δηλαδή από αριστερά προς τα δεξιά ή ανάποδα. Η απάντηση είναι από αριστερά προς τα δεξιά, διότι κάθε σημείο που εξετάζεται, και η τεταγμένη του είναι μικρότερη από κάποιο άλλο, τότε πρέπει να απαλειφθεί από το σύνολο των μεγίστων, και δεν υπάρχει λόγος επανεξέτασης. Στην αντίθετη περίπτωση (από δεξιά προς τα αριστερά), και στην χειρότερη δυνατή περίπτωση όπου όλα τα σημεία είναι μέγιστα, θα πρέπει το κάθε σημείο του συνόλου των μεγίστων να επανεξετάζεται κάθε φορά, και αυτό οδηγεί σε πολυπλοκότητα της τάξης .

Παρακάτω περιγράφεται αναλυτικά ο αλγόριθμος. Επειδή τα μέγιστα εισάγονται στο τέλος μίας λίστας, και διαγράφονται πάλι από το τέλος της, μπορεί να χρησιμοποιηθεί η δομή «στοίβα ή σωρός», και έστω S το όνομά της. Το τελευταίο σημείο του σωρού είναι το . Η διαδικασία εισόδου στο σωρό λέγεται Push, και εξόδου (κατά συνέπεια και διαγραφής), Pop.

***Αλγόριθμος 1****: Βελτιωμένος αλγόριθμος επίλυσης του προβλήματος εύρεσης μέγιστων δισδιάστατου χώρου (Sweep – Plane).*

1: Δεδομένα: Πλήθος σημείων n, και πίνακας σημείων Π(n), και πίνακας μεγίστων S(n).

2: Αρχή,

3: Ταξινόμηση του πίνακα Π, με κλειδί την τετμημένη των σημείων.

4: S = κενό.

5: Για i από 1 μέχρι n,

6: Εφόσον (S όχι κενή ΚΑΙ S.top.y ≤ Π(i).y),

7: Pop(S),

8: Τέλος Εφόσον.

9: Push(S, Π(i)).

10: Τέλος Για (i).

11: Τέλος Αλγόριθμου.

## 3.2 Πολυπλοκότητα του αλγόριθμου σάρωσης του επιπέδου.

Κατ’ αρχήν, η ταξινόμηση του πίνακα των σημείων είναι της τάξης . Ο αλγόριθμος αυτός, περιλαμβάνει επίσης δύο βρόγχους. Ο εξωτερικός βρόγχος επαναλαμβάνεται n φορές. Το ερώτημα που τίθεται είναι πόσες φορές επαναλαμβάνεται ο εσωτερικός. Αρχικά, ενώ δείχνει ότι επαναλαμβάνεται και αυτός το πολύ n -1 φορές για κάθε επανάληψη του εξωτερικού βρόγχου, που σημαίνει πολυπλοκότητα της τάξης n2, στην πραγματικότητα επαναλαμβάνεται το πολύ n-1 φορές συνολικά. Αυτό ισχύει γιατί δεν είναι δυνατό να εξαχθούν πιο πολλά σημεία από την στοίβα, από όσα θα εισαχθούν, και θα εισαχθούν n ακριβώς σημεία. Και επειδή τουλάχιστον ένα μέγιστο θα υπάρχει κατά συνέπεια, θα επαναληφθεί n-1 φορές συνολικά. Επομένως, η συνολική πολυπλοκότητα του αλγόριθμου είναι της τάξης , δηλαδή .

Συγκριτικά με τον προηγούμενο αλγόριθμο (brute-force), του οποίου η πολυπλοκότητα είναι της τάξης , ο sweep–plane είναι σαφώς πιο γρήγορος. Ο λόγος των πολυπλοκοτήτων τους είναι:

,

όπου ενώ για μικρό n είναι σχεδόν ισοδύναμοι, όσο όμως αυξάνεται το πλήθος των σημείων, η διαφορά τους είναι εμφανής. Για παράδειγμα, εάν n = 1.000.000, ο λόγος είναι . Αυτό σημαίνει σε πραγματικούς χρόνους, ότι εάν ο αλγόριθμος σάρωσης του επιπέδου χρειάζεται 1 δευτερόλεπτο, ο πρώτος αλγόριθμος (brute-force) θα χρειαζόταν 13,88 ώρες.

## 3.3 Ασκήσεις.

1. Να γραφεί πρόγραμμα υλοποίησης του αλγόριθμου σάρωσης του επιπέδου. Η στοίβα να υλοποιείται με πίνακα, και τα σημεία να εισάγονται από αρχείο δεδομένων.

**Λύση.**

***Πρόγραμμα 1****: Πρόγραμμα υλοποίησης αλγόριθμου σάρωσης επιπέδου.*

#include <stdio.h>

#define M 1000 /\* Μέγιστος αριθμός σημείων \*/,

struct Simio {

 int x;

 int y;

};

struct Simio P[M], S[M], temp;

main()

{

 FILE f; **/** Αρχείο αποθήκευσης σημείων \*/,

 int i=0, j=0, k=0, l=0, N;

 int megisto;

 /\* Εισαγωγή σημείων \*/,

 F = fopen("2dmax.dat", "r");

 while (! feof(f) ) {

 fscanf(f, "%d %d", &P[i].x, &P[i].y);

 if (! feof(f)) {

printf("\n %2d ==> %5d, %5d", i, P[i].x, P[i].y);

i++;

 }

 }

 fclose(f);

 printf("\n\n");

 system("pause");

 N = i;

 /\* Ταξινόμηση πίνακα σημείων – Bubble sort \*/,

 for (i=0; i<N; i++)

 for (j=N-1; j>0; j--)

if (P[j].x < P[j-1].x) {

/\* Αντιμετάθεση \*/,

temp = P[j];

P[j] = P[j-1];

P[j-1] = temp;

}

 /\* Ταξινομημένος ως προς την τετμημένη x \*/,

 printf("\n\n Ταξινομημένα σημεία \n");

 for (i=0; i<N; i++)

 printf("\n %3d --> %5d, %5d", i, P[i].x, P[i].y);

 printf("\n\n");

 system("pause");

 /\* Αρχικοποίηση S \*/,

 for (i=0; i<N; i++) {

 S[i].x = 0;

 S[i].y = 0;

 }

/\* Υπολογισμός 2-d μεγίστων (plane sweep algorithm) \*/,

 j = 0;

 for (i=0; i<N; i++) {

 while (j>=0 && S[j-1].y <= P[i].y)

j--; /\* Απομάκρυνση σημείου \*/,

S[j++] = P[i]; /\* Εισαγωγή σημείου \*/,

 }

/\* Εκτυπώσεις \*/,

 printf("\n\n ΜΈΓΙΣΤΑ \n");

 for (i=0; i<j; i++)

 printf("\n %3d --> %5d, %5d", i, S[i].x, S[i].y);

printf("\n\n Σύνολο μεγίστων: %5d \n\n", j);

}

Τέλος τρίτης ενότητας.

 