

**Αλγόριθμοι και Στοιχεία Πολυπλοκότητας.**

**Ενότητα 11:** Κατακερματισμός.

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:

αντιληφθεί την έννοια του κατακερματισμού.

# ΠΕΡΙΕΧΌΜΕΝΑ ΕΝΌΤΗΤΑΣ.

[Σκοποί ενότητας. 2](#_Toc368164117)

[ΠΕΡΙΕΧΌΜΕΝΑ ΕΝΌΤΗΤΑΣ. 3](#_Toc368164118)

[11. ΚΑΤΑΚΕΡΜΑΤΙΣΜΌΣ (Hash tables). 3](#_Toc368164119)

[11.1 Εισαγωγικές έννοιες ⁄ Ορισμοί. 3](#_Toc368164120)

# 11. ΚΑΤΑΚΕΡΜΑΤΙΣΜΌΣ (Hash tables).

## 11.1 Εισαγωγικές έννοιες ⁄ Ορισμοί.

* **Πίνακες Άμεσης Προσπέλασης:** Εάν έχουμε μία συλλογή *n* στοιχείων, των οποίων τα κλειδιά είναι μοναδικοί ακέραιοι αριθμοί στο διάστημα, **, όπου , τότε μπορούν να αποθηκευτούν σε ένα πίνακα **, όπου κάθε στοιχείο $T\_{i}$ είναι ή άδειο, ή περιέχει κάποιο αντικείμενο της συλλογής με κλειδί *i*. Η αναζήτηση είναι προφανές, ότι εκτελεί  λειτουργίες. Ο αλγόριθμος αναζήτησης είναι:

a. Δεδομένου ενός κλειδιού k,

b. Άμεση προσπέλαση του στοιχείου **, όπου εάν δεν είναι άδειο, τότε το υπό αναζήτηση στοιχείο βρέθηκε, αλλιώς δεν υπάρχει καθόλου.

* Στην περίπτωση κατά την οποία τα κλειδιά δεν είναι μοναδικά, τότε πρέπει να δημιουργηθεί ένα σύνολο από *m* λίστες, όπου η κεφαλή αυτών των λιστών θα είναι αποθηκευμένη στον πίνακα **.
* Όταν η απεικόνιση δεν είναι άμεση, αλλά υπάρχει μία συνάρτηση , δηλαδή απεικονίζει κάθε τιμή του κλειδιού *k* στο πεδίο τιμών , τότε μπορούμε να γενικεύσουμε θέτοντας,  αντί **.
* **Συναρτήσεις Απεικόνισης.** Η άμεση προσπέλαση απαιτεί από την ***hashing*** συνάρτηση *,* μία ένα προς ένα απεικόνιση, όπου το κάθε κλειδί *k,* απεικονίζεται σε ένα μοναδικό ακέραιο του διαστήματος, . Αυτή είναι η περίπτωση της τέλειας *hashing* συνάρτησης. Δυστυχώς, η εύρεση μίας τέτοιας συνάρτησης δεν είναι πάντα εφικτή. Συνήθως, βρίσκουμε μία *hash* συνάρτηση, η οποία απεικονίζει τα περισσότερα από τα κλειδιά σε μοναδικούς ακέραιους, αλλά υπάρχει και ένας μικρός αριθμός κλειδιών, που απεικονίζονται σε ίδιους ακέραιους. Σε αυτή την περίπτωση, θεωρούμε ότι έχουμε **σύγκρουση** (collision) κλειδιών. Εάν ο αριθμός των συγκρούσεων είναι μικρός, τότε η τεχνική δουλεύει πολύ καλά, και πάλι λειτουργεί με πολυπλοκότητα .
* **Χειρισμός Συγκρούσεων.** Στη περίπτωση όπου πολλαπλά κλειδιά απεικονίζονται στον ίδιο ακέραιο, τότε στοιχεία με διαφορετικά κλειδιά, διεκδικούν να αποθηκευτούν στον ίδιο χώρο. Έχουν αναπτυχθεί αρκετές τεχνικές για την αντιμετώπιση αυτού του προβλήματος, με βασικότερες τις εξής:
1. Δημιουργία Περιοχών Υπερχείλισης,
2. Re-hashing,
3. Χρήση Γειτονικών Θέσεων,
4. Χρήση Τυχαίων Θέσεων και άλλες.

Για παράδειγμα, στην χρήση γειτονικών θέσεων (+1 ή -1), στην περίπτωση σύγκρουσης ψάχνεται η γειτονική προς την κανονική θέση, και εάν είναι άδεια, το στοιχείο τοποθετείται εκεί, αλλιώς συνεχίζεται η αναζήτηση κενής θέσης, στην επόμενη γειτονική και ούτω καθεξής. Αυτή είναι μία αρκετά καλή μέθοδος, αλλά θα πρέπει να προσεχθεί ιδιαιτέρως η περίπτωση, όπου υπάρχουν ενδεχόμενες διαγραφές από τον πίνακα.

**ΠΑΡΆΔΕΙΓΜΑ.**

Εάν υποθέσουμε ότι το κλειδί του κάθε στοιχείου ⁄ εγγραφής είναι ένα μοναδικό αλφαριθμητικό, πχ αριθμός ταυτότητας, Α.Φ.Μ., κωδικός προϊόντος, και τα λοιπά. Επειδή δεν αντιπροσωπεύει άμεσα κάποιον ακέραιο, θα μπορούσε η hashing συνάρτηση να προσθέτει τους αντίστοιχους ASCII κωδικούς, και επειδή n = 10, να υπολογίζει και το υπόλοιπο της διαίρεσης του ακέραιου που προκύπτει με το 10. Δηλαδή εάν $k= k\_{1}k\_{2}… k\_{x}$, όπου $k\_{x}$ είναι κάποιο αλφαριθμητικό, τότε .

**Αλγόριθμος Εισαγωγής.**

Δεδομένα: Κλειδί k, hashing συνάρτηση **, n μέγεθος αποθηκευτικού πίνακα.

1. Υπολογισμός **.
2. Εάν το , είναι κενό, εισαγωγή της εγγραφής στη θέση *,* και τέλος.
3. Εάν το  , είναι μη κενό, τότε έστω x = *,*
4. x = x + 1.
5. Εάν x μεγαλύτερο του 1, τότε x = 1.
6. Εάν ** είναι κενό, εισαγωγή της εγγραφής στη θέση ** και τέλος, αλλιώς συνέχισε στο βήμα 4.

 **Αλγόριθμος Αναζήτησης.**

Δεδομένα: Κλειδί k, hashing συνάρτηση **, n μέγεθος αποθηκευτικού πίνακα.

1. Υπολογισμός *,* και έστω x = **.
2. Εάν το ** είναι το ζητούμενο, τότε επιστροφή του x και τέλος.
3. Εάν το ** είναι κενό, τότε αποτυχία αναζήτησης, και τέλος.
4. Εάν το ** είναι μη κενό και όχι το ζητούμενο, τότε,
	1. x = x + 1.
	2. Εάν x μεγαλύτερο του n, τότε x = 1.
	3. Συνέχιση στο βήμα 2.

Το πρόβλημα που θα προκύψει με ενδεχόμενες διαγραφές, είναι ότι η ανακάλυψη ενός κενού χώρου στον αλγόριθμο αναζήτησης, δεν συνεπάγεται πάντα την αποτυχία της αναζήτησης. Εάν ο κενός χώρος δημιουργήθηκε λόγω διαγραφής, τότε είναι πιθανό το υπό αναζήτηση στοιχείο, να υπάρχει σε ένα από τους επόμενους γειτονικούς χώρους. Μία λύση αυτού του προβλήματος, είναι να τίθεται ένα σύμβολο διαγραφής σε κάθε κενό χώρο, που προέκυψε από διαγραφή. Για παράδειγμα, το κενό κελί είναι null ή 0 ενώ το διαγραμμένο είναι ‘\*\*\*’.

Τέλος ενδέκατης ενότητας.

 