# Ενότητα 7
# Κατακερματισμός



## Κατακερματισμός - Αρχές Λειτουργίας

Έστω $S \subseteq U$ το προς αποθήκευση σύνολο κλειδιών.

```
Type LookUp(set S, Key k) {
   return(H[k]);
}
```

**Βασική Ιδέα**

«Αν το κλειδί $k \in S$ ενός στοιχείου $s$ ήταν ακέραιος αριθμός, τότε θα ήταν δυνατό το $k$ να αποτελεί δείκτη σε έναν πίνακα $A$, όπου το στοιχείο $s$ είναι αποθηκευμένο, δηλαδή να ισχύει $A[k] = s$».

$\Rightarrow$Αν υπήρχε άπειρος αποθηκευτικός χώρος, η εύρεση ενός στοιχείου στο σύνολο θα είχε χρονική πολυπλοκότητα $O(1)$!!

☹ Η υπόθεση αυτή είναι ανέφικτη:
- Πάντα υπάρχει ένα πάνω όριο $m$ στο διαθέσιμο χώρο.
- Η σπατάλη σε μνήμη είναι μεγάλη όταν το σύνολο έχει λίγα στοιχεία ενώ ο χώρος $U$ των κλειδιών είναι τεράστιος.

Ας υποθέσουμε πως ο χώρος των κλειδιών είναι το σύνολο των φυσικών αριθμών.

**Ιδέα**

Αποθηκεύουμε τα στοιχεία του συνόλου $S$ σε έναν πίνακα $m$ θέσεων, που ονομάζεται πίνακας κατακερματισμού και χρησιμοποιούμε μια συνάρτηση $h$ η οποία απεικονίζει το σύνολο $\{0, \ldots, |U|\}$ στο σύνολο $\{0,\ldots m\}$. Η συνάρτηση αυτή λέγεται συνάρτηση κατακερματισμού. Το στοιχείο με κλειδί $k$ αποθηκεύεται στη θέση $A[h[k]]$ του πίνακα.

# Κατακερματισμός

**Συγκρούσεις (collisions)**
Όταν για δύο κλειδιά $K_i$ και $K_j$ με $K_i \neq K_j$ ισχύει ότι $h(K_i) = h(Kj)$ λέμε πως συμβαίνει σύγκρουση.

Όταν συμβαίνουν συγκρούσεις, θα πρέπει να γίνει ανακατανομή κλειδιών με κατάλληλο τρόπο ώστε να επιλυθεί η σύγκρουση και τα κλειδιά να μπορούν να βρεθούν (μέσω της LookUp()) σε λογικό χρόνο μετά την ανακατανομή.

**Καλές Συναρτήσεις Κατακερματισμού**
Κάνουν καλή διασκόρπιση των κλειδιών στον πίνακα.

**Αρχή Απλής Ομοιόμορφης Κατανομής των Κλειδιών**
«Αν ένα κλειδί K επιλέγεται τυχαία από το χώρο κλειδιών, η πιθανότητα να ισχύει $h(K) = j$, θα πρέπει να είναι $1/m$, ίδια για όλα τα $j$, $1 \leq j \leq m$ (δηλαδή ίδια για όλες τις θέσεις του πίνακα)».

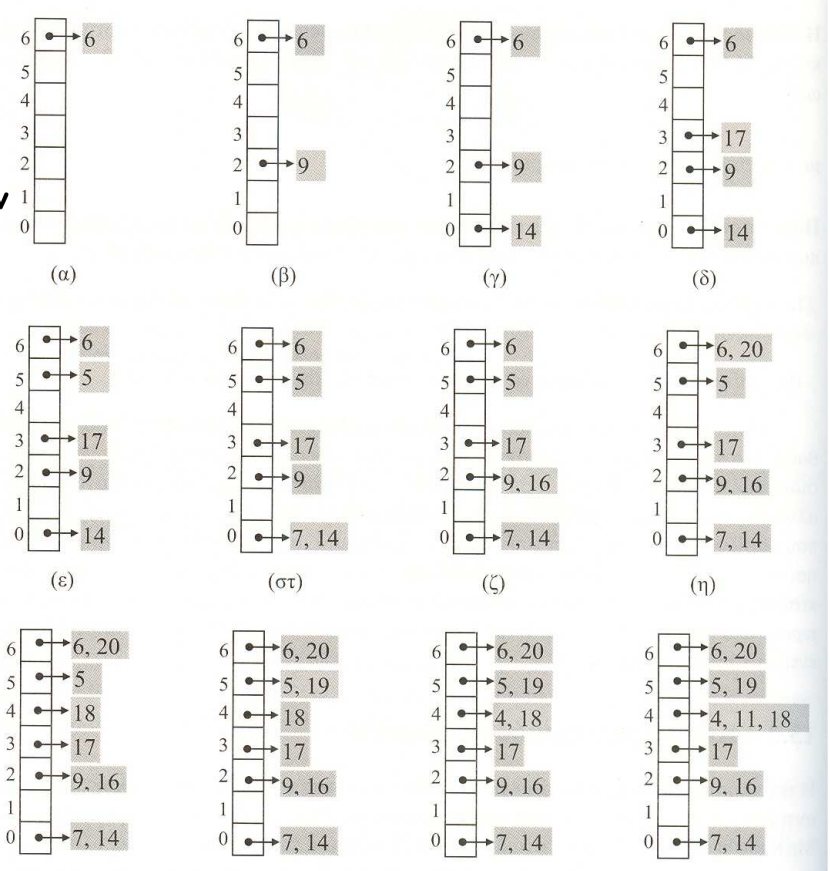
**Παράδειγμα Συνάρτησης Κατακερματισμού**
$h(k) = k \bmod m$



# Μέθοδοι Διαχείρισης Συγκρούσεων

**Μέθοδος ξεχωριστών αλυσίδων**
Η θέση A[j] του πίνακα κατακερματισμού δεν περιέχει ένα στοιχείο αλλά ένα δείκτη σε μια δυναμική δομή (που υλοποιεί ένα λεξικό, η οποία περιέχει κάθε στοιχείο με κλειδί K τέτοιο ώστε $h(K) = j$.

**Παράδειγμα**
Διαδοχική εισαγωγή των κλειδιών 6, 9, 14, 17, 5, 7, 16, 20, 18, 19, 4, 11 σε έναν πίνακα κατακερματισμού 7 θέσεων βάσει της μεθόδου της αλυσίδας χρησιμοποιώντας τη συνάρτηση κατακερματισμού: $h(k) = k \bmod 7$



# Μέθοδος Διαχείρισης Συγκρούσεων των Ξεχωριστών Αλυσίδων (separate chaining)

```
Insert(HashTable A, Key K) {
    int pos;
    pos = h(K);        // εύρεση του κλειδιού του στοιχείου e του U βάσει συναρτήσεως κατακερματισμού
    (εισαγωγή του στοιχείου με κλειδί K στην αλυσίδα εκείνη στης οποίας
     το πρώτο στοιχείο δείχνει ο δείκτης A[pos]);
                                   // η εισαγωγή μπορεί να γίνει είτε στην αρχή ή στο τέλος της αλυσίδας
}

LookUp(HashTable A, Key k) {
   pointer p; int pos;
   pos = h(K);        // εύρεση του κλειδιού του στοιχείου e βάσει της συναρτήσεως κατακερματισμού
   p = A[pos];        // ο p είναι δείκτης στο πρώτο στοιχείο της αλυσίδας
   while (p!= NULL AND p->key != K) p = p->next;
            // διάσχιση της αλυσίδας μέχρι είτε να βρεθεί το κλειδί ή να φθάσουμε στο τέλος της
   return p;
}

Delete(HashTable A, Key K) {
   int pos = h(K);    // εύρεση του κλειδιού του στοιχείου e του U βάσει συναρτήσεως κατακερματισμού
    (εύρεση και διαγραφή του στοιχείου K από την αλυσίδα εκείνη στης οποίας
     το πρώτο στοιχείο δείχνει ο δείκτης A[pos]);
}
```



# Μέθοδοι Διαχείρισης Συγκρούσεων - Μέθοδος Ξεχωριστών Αλυσίδων - Ανάλυση Πολυπλοκότητας

- Συμβολίζουμε με n το μέγεθος του λεξικού και με m το μέγεθος του πίνακα κατακερματισμού.

**Λήμμα 1**
Σε μια δομή κατακερματισμού με αλυσίδες, n στοιχείων, το μέσο πλήθος στοιχείων που είναι αποθηκευμένα σε μια αλυσίδα είναι α = m/n (και μάλιστα με πιθανότητα που τείνει στο 1).

- Το α είναι γνωστό ως παράγοντας φόρτου (load fator).

Ποια είναι η αναμενόμενη χρονική πολυπλοκότητα μιας LookUp() σε πίνακα κατακερματισμού οργανωμένου με τη μέθοδο των αλυσίδων;

# Μέθοδοι Διαχείρισης Συγκρούσεων - Μέθοδος Ξεχωριστών Αλυσίδων - Ανάλυση Πολυπλοκότητας

❑ Συμβολίζουμε με S(α) το αναμενόμενο πλήθος προσβάσεων στη μνήμη που απαιτούνται για την εκτέλεση μιας επιτυχημένης LookUp() (δηλαδή μιας LookUp() σε κλειδί που υπάρχει στην δομή).
❑ Συμβολίζουμε με U(α) το αναμενόμενο πλήθος προσβάσεων στη μνήμη που απαιτούνται για την εκτέλεση μιας αποτυχημένης LookUp() (δηλαδή μιας LookUp() σε κλειδί που δεν υπάρχει στην δομή).

$\Rightarrow U(\alpha) = 1 + \alpha$.

**S(α)**
Ποιος είναι ο μέσος αριθμός προσβάσεων στη μνήμη που απαιτούνται για μια επιτυχημένη αναζήτηση, αν:
❑ το μέγεθος της αλυσίδας είναι 1; 1
❑ το μέγεθος της αλυσίδας είναι 2; (1+2)/2
❑ το μέγεθος της αλυσίδας είναι κ; (1+2+3+...+κ)/κ=(κ+1)/2

Αν όλες οι αλυσίδες ήταν μη-άδειες, το αναμενόμενο μήκος κάθε αλυσίδας θα ήταν α, οπότε $S(\alpha) = 1 + (1+\alpha)/2 = 3/2 + \alpha/2$.

Μια επιτυχημένη LookUp ποτέ δεν εξετάζει άδειες αλυσίδες. Γιατί?
Ωστόσο, το μήκος της αλυσίδας μπορεί να είναι λίγο μεγαλύτερο από α:
$S(\alpha) \approx 2 + \alpha/2$ (αυτό έχει αποδειχθεί πειραματικά).



# Μέθοδοι Διαχείρισης Συγκρούσεων - Μέθοδος Ξεχωριστών Αλυσίδων - Ανάλυση Πολυπλοκότητας

Χειρότερη περίπτωση;

Τα n κλειδιά έχουν την ίδια τιμή κατακερματισμού και έτσι τοποθετούνται όλα στην ίδια αλυσίδα μήκους n. Χρόνος LookUp() στη χειρότερη περίπτωση είναι $O(n)$ + χρόνο υπολογισμού της συνάρτησης κατακερματισμού.

➧ Η επίδοση της μεθόδου του κατακερματισμού εξαρτάται από το πόσο καλά η συνάρτηση κατακερματισμού κατανέμει το σύνολο των προς εισαγωγή κλειδιών στις m θέσεις του πίνακα κατακερματισμού.

Ποια είναι η κατάλληλη επιλογή για το m;

Είναι επιθυμητό το α να είναι μια σταθερά, άρα θα θέλαμε το m να επιλεγεί έτσι ώστε $n/m \in O(1)$.

Πόσο εύκολα μπορούμε να υλοποιήσουμε διαγραφή;

Είναι εφικτό να υλοποιηθεί η διαγραφή με εύκολο τρόπο. Βρίσκουμε μέσω της συνάρτησης κατακερματισμού την κατάλληλη αλυσίδα στην οποία θα πρέπει να είναι αποθηκευμένο το προς διαγραφή κλειδί και το αναζητούμε σε αυτήν. Αν το κλειδί βρεθεί, διαγράφεται.

# Μέθοδος Μικτών Αλυσίδων (coalesced chaining)

- Η αλυσίδα αποθηκεύεται μέσα στον πίνακα που είναι σταθερού μεγέθους (δεν απαιτείται δυναμική παραχώρηση μνήμης) ⇒ ο πίνακας εκτός της αποθήκευσης των άλλων δεδομένων πρέπει να περιέχει και μια στήλη για την αποθήκευση των δεικτών (έστω ότι αυτό είναι το πεδίο next κάθε στοιχείου του πίνακα).
- Όταν συμβεί σύγκρουση και το πεδίο next του στοιχείου που προκαλεί τη σύγκρουση είναι Λ, αναζητείται η πρώτη διαθέσιμη θέση ξεκινώντας από τη θέση 0 του πίνακα. Έστω ότι η θέση αυτή είναι η p.
- Το πεδίο next του στοιχείου που προκαλεί τη σύγκρουση ενημερώνεται να δείχνει στην p.



# Μέθοδος Μικτών Αλυσίδων (coalesced chaining)

**Παράδειγμα:** Διαδοχική εισαγωγή των κλειδιών 71, 52, 12, 56, 1, 10, 90, 19 σε έναν πίνακα κατακερματισμού 10 θέσεων χρησιμοποιώντας τη συνάρτηση κατακερματισμού h(k) = k mod 10

| | |
|---|---|
| | Λ |
| | Λ |
| | Λ |
| | Λ |
| | Λ |
| | Λ |
| | Λ |
| | Λ |
| 71 | Λ |
| | Λ |

| | |
|---|---|
| | Λ |
| | Λ |
| | Λ |
| | Λ |
| | Λ |
| | Λ |
| 52 | Λ |
| 71 | Λ |
| | Λ |

| | |
|---|---|
| | Λ |
| | Λ |
| | Λ |
| | Λ |
| | Λ |
| | Λ |
| 52 | 0 |
| 71 | Λ |
| 12 | Λ |

| | |
|---|---|
| | Λ |
| | Λ |
| | Λ |
| 56 | Λ |
| | Λ |
| | Λ |
| 52 | 0 |
| 71 | Λ |
| 12 | Λ |

| | |
|---|---|
| | Λ |
| | Λ |
| | Λ |
| 56 | Λ |
| | Λ |
| | Λ |
| 1 | Λ |
| 52 | 0 |
| 71 | 3 |
| 12 | Λ |

| | |
|---|---|
| | Λ |
| | Λ |
| | Λ |
| 56 | Λ |
| | Λ |
| 10 | Λ |
| 1 | 4 |
| 52 | 0 |
| 71 | 3 |
| 12 | 4 |

- Μπορεί να διαπλέκονται διαφορετικές αλυσίδες, δηλαδή μια αλυσίδα μπορεί να περιέχει κλειδιά που έχουν διαφορετική τιμή κατακερματισμού (σε αντίθεση με τη βασική μέθοδο της αλυσίδας όπου μια αλυσίδα περιέχει μόνο κλειδιά με την ίδια τιμή κατακερματισμού).

**Πρόβλημα:** Μια θέση στην οποία έχει τοποθετηθεί ένα κλειδί για το οποίο η συνάρτηση κατακερματισμού καθορίζει άλλη θέση εισαγωγής, μπορεί να πρέπει να αποθηκεύσει αργότερα το πρώτο κλειδί μιας νέας αλυσίδας.

**Λύση:** Το κλειδί τοποθετείται στην πρώτη διαθέσιμη θέση και συνδέεται στο τέλος της αλυσίδας του κλειδιού που κατέχει την αρχική θέση.

| | |
|---|---|
| 19 | Λ |
| | Λ |
| | Λ |
| 56 | Λ |
| 90 | Λ |
| 10 | 5 |
| 1 | 4 |
| 52 | 0 |
| 71 | 3 |
| 12 | 4 |

# Μέθοδος Μικτών Αλυσίδων

**LookUp:** Ίδια με εκείνη της βασικής μεθόδου της αλυσίδας:

1. Βρίσκουμε την τιμή κατακερματισμού h(K) του προς αναζήτηση στοιχείου (που έστω ότι έχει κλειδί K).
2. Αν το κλειδί βρίσκεται στον πίνακα θα είναι σε μια από τις θέσεις που καταλαμβάνουν τα στοιχεία της αλυσίδας που ξεκινά από τη θέση h(K) του πίνακα. Ακολουθούμε την αλυσίδα αυτή μέχρι είτε να βρούμε το κλειδί που αναζητάμε ή να φθάσουμε στην τελευταία θέση της αλυσίδας.
   **Παρατηρήση:** Η κάθε αλυσίδα ίσως περιέχει κλειδιά για τα οποία η συνάρτηση κατακερματισμού δίνει διαφορετικές τιμές.

**Insert:** Όπως στη βασική μέθοδο με αλυσίδες:

1. Βρίσκουμε την τιμή κατακερματισμού h(K) του προς εισαγωγή στοιχείου (που έστω ότι έχει κλειδί K).
2. Εξετάζουμε την θέση h(K) του πίνακα. Αν είναι κατειλημμένη, και το πεδίο next της θέσης αυτής είναι Λ, αναζητούμε την πρώτη διαθέσιμη θέση στον πίνακα, ξεκινώντας από τη θέση 0 του πίνακα. Διαφορετικά, ακολουθούμε την αλυσίδα που ξεκινά από αυτή τη θέση και όταν φθάσουμε στην τελευταία θέση της αλυσίδας, ξεκινώντας από εκεί βρίσκουμε την επομένη ελεύθερη θέση στον πίνακα.



# Μέθοδος Μικτών Αλυσίδων

**Κελάρι (cellar):** Κρατάμε τις πρώτες θέσεις του πίνακα μόνο για την επίλυση συγκρούσεων. Δηλαδή, αν ο πίνακας έχει m θέσεις, οι πρώτες c αποτελούν το κελάρι. Η συνάρτηση κατακερματισμού έχει πεδίο τιμών {c, …, m-1}, ενώ για την επίλυση των συγκρούσεων χρησιμοποιείται όλος ο πίνακας.

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 9 | | Λ | | Λ | | Λ | | Λ | | Λ | | Λ |
| 8 | | Λ | | Λ | 56 | Λ | 56 | Λ | 56 | Λ | 56 | Λ |
| 7 | | Λ | | Λ | | Λ | | Λ | | Λ | | Λ |
| 6 | | Λ | | Λ | | Λ | | Λ | | Λ | | Λ |
| 5 | | Λ | | Λ | | Λ | | Λ | | Λ | **90** | Λ |
| 4 | | Λ | 52 | **0** | 52 | Λ | 52 | 0 | 52 | 0 | 52 | 0 |
| 3 | 71 | Λ | 71 | Λ | 71 | Λ | 71 | **1** | 71 | 1 | 71 | 1 |
| 2 | | Λ | | Λ | | | | Λ | 10 | Λ | 10 | **5** |
| 1 | | Λ | | Λ | | Λ | **1** | Λ | 1 | Λ | 1 | Λ |
| 0 | | Λ | **12** | Λ | 12 | Λ | 12 | Λ | 12 | Λ | 12 | Λ |

**Παράδειγμα:** Διαδοχική εισαγωγή των κλειδιών 71, 52, 12, 56, 1, 10, 90, 19 σε έναν πίνακα κατακερματισμού 10 θέσεων χρησιμοποιώντας τη συνάρτηση κατακερματισμού h(k) = (k mod 10) mod 8 + 2

| | |
|---|---|
| | Λ |
| 56 | Λ |
| | Λ |
| **19** | Λ |
| 90 | **6** |
| 52 | 0 |
| 71 | 1 |
| 10 | 5 |
| 1 | Λ |
| 12 | Λ |

c = 2

Πειραματική και θεωρητική δουλειά έχει αποδείξει ότι αν το κελάρι είναι το 14% του συνολικού πίνακα, η απόδοση είναι καλή.

# Κατακερματισμός με Ανοικτή Διευθυνσιοδότηση (Open Addressing Strategies)

- Η μέθοδος στοχεύει στην ορθή διαχείριση του ελεύθερου χώρου του ίδιου του πίνακα κατακερματισμού.
- Όλα τα κλειδιά αποθηκεύονται στον πίνακα χωρίς ωστόσο να σχηματίζονται αλυσίδες (δεν χρησιμοποιούνται δείκτες).
- Για κάθε κλειδί ελέγχεται μια ακολουθία από θέσεις του πίνακα, που ονομάζεται ακολουθία εξέτασης. Η πρώτη διαθέσιμη θέση του πίνακα με τη σειρά που καθορίζεται από την ακολουθία αυτή θα στεγάσει το κλειδί.
- Η ακολουθία καθορίζεται βάσει κάποιου κανόνα (που συνήθως λαμβάνει υπόψη του και το ίδιο το κλειδί).
- Συμβολίζουμε με H(K,i) την i-οστή θέση του πίνακα που ελέγχουμε αν είναι διαθέσιμη για το κλειδί K, i = 0, 1 … (δηλαδή, H(K,i) είναι το i-οστό στοιχείο της ακολουθίας αναζήτησης του κλειδιού K).

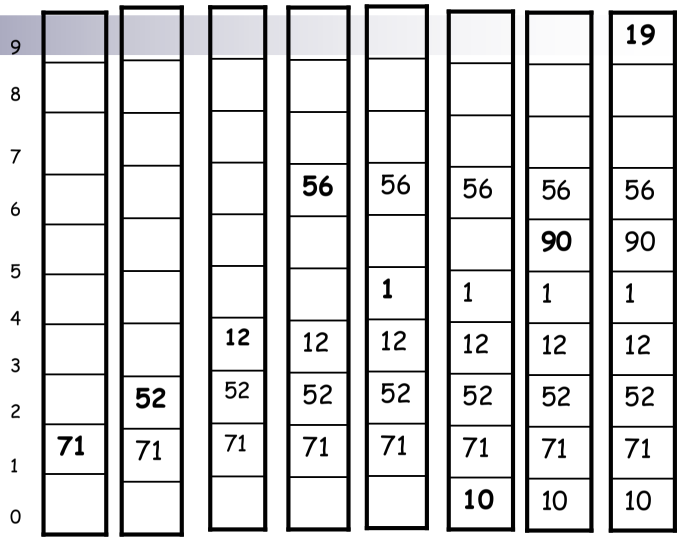
**LookUp(HashTable A, Key K)**
Ψάξε τις διαδοχικές θέσεις στην ακολουθία εξέτασης του K, μέχρι είτε να βρεθεί το κλειδί, ή να βρεθεί μια κενή θέση στον πίνακα.
Στη δεύτερη περίπτωση, το κλειδί δεν υπάρχει στον πίνακα.

**Insert(HashTable A, Key K)**
Εκτέλεση της LookUp(A,K). Αν το κλειδί βρεθεί, ο αλγόριθμος τερματίζει.
Διαφορετικά, το νέο κλειδί εισάγεται στην κενή θέση που επιστρέφεται από την LookUp(A,K).



# Κατακερματισμός με Ανοικτή Διευθυνσιοδότηση

**Γραμμική Αναζήτηση**
Ακολουθία εξέτασης
H(K,0) = h(K);
// η πρώτη θέση στην ακολουθία καθορίζεται από
// την τιμή κατακερματισμού
H(K,i+1) = (H(K,i)+1) mod m;
// η (i+1) θέση είναι η επόμενη της θέσης i (mod m,
// δηλαδή με κυκλικό τρόπο).




**Παράδειγμα**
Διαδοχική εισαγωγή των κλειδιών 71, 52, 12, 56, 1, 10, 90, 19 σε έναν πίνακα κατακερματισμού 10 θέσεων χρησιμοποιώντας τη συνάρτηση κατακερματισμού h(k) = k mod 10.

➡ Η απόδοση της μεθόδου είναι ικανοποιητική όταν ο πίνακας δεν έχει πολλά στοιχεία.

✖ **Πρόβλημα: Φαινόμενο Πρωταρχικής Συγκέντρωσης (primary clustering)**
Όταν ένα μπλοκ (cluster) με συνεχόμενες κατειλημμένες θέσεις δημιουργηθεί, αποτελεί προορισμό για περαιτέρω συγκρούσεις, ενώ νέες τέτοιες συγκρούσεις οδηγούν στην αύξηση του μεγέθους του μπλοκ.

# Κατακερματισμός με Ανοικτή Διευθυνσιοδότηση

**Αριθμός απαιτούμενων προσπελάσεων στη μνήμη για μη επιτυχημένη αναζήτηση;**

- αν η συνάρτηση κατακερματισμού επιστρέφει μια κενή θέση του πίνακα;
  Μία μόνο προσπέλαση απαιτείται!
- αν η συνάρτηση κατακερματισμού επιστρέφει μια κατειλημμένη θέση του πίνακα;
  Η αναζήτηση θα τερματίσει μόνο αφού εξετάσει και το τελευταίο στοιχείο του cluster.

**Προσπάθεια βελτίωσης**

Ακολουθία εξέτασης

H(K,0) = h(K); // η πρώτη θέση στην ακολουθία καθορίζεται από την τιμή κατακερματισμού

H(K,i+1) = (H(K,i)+c) mod m; // η (i+1) θέση προς εξέταση είναι c θέσεις μετά τη θέση i (mod m).

Τι αποτέλεσμα θα έχει αυτό το σχήμα; Θα οδηγούσε σε καλύτερη απόδοση;

Και το σχήμα αυτό υποφέρει από τα φαινόμενο δημιουργίας μακρών ακολουθιών κατειλημμένων θέσεων, γνωστό ως φαινόμενο δευτερεύουσας συγκέντρωσης (secondary clustering).



# Κατακερματισμός με Ανοικτή Διευθυνσιοδότηση Διπλός Κατακερματισμός (Double Hashing)

Χρησιμοποιούνται δύο συναρτήσεις κατακερματισμού, $h_1$ και $h_2$. Η $h_1$ ονομάζεται πρωτεύουσα συνάρτηση κατακερματισμού, ενώ η $h_2$ δευτερεύουσα.

Ακολουθία Εξέτασης

$H(K,0) = h_1(K)$; // η πρώτη θέση στην ακολουθία εξέτασης καθορίζεται
// από την πρωταρχική συνάρτηση κατακερματισμού $h_1$

$H(K, i+1) = (H(K, i) + h_2(K)) \bmod m$; // η επόμενη της θέσης i στην ακολουθία εξέτασης είναι η θέση
// που βρίσκεται $h_2(K)$ θέσεις μετά τη θέση i στον πίνακα (mod m)

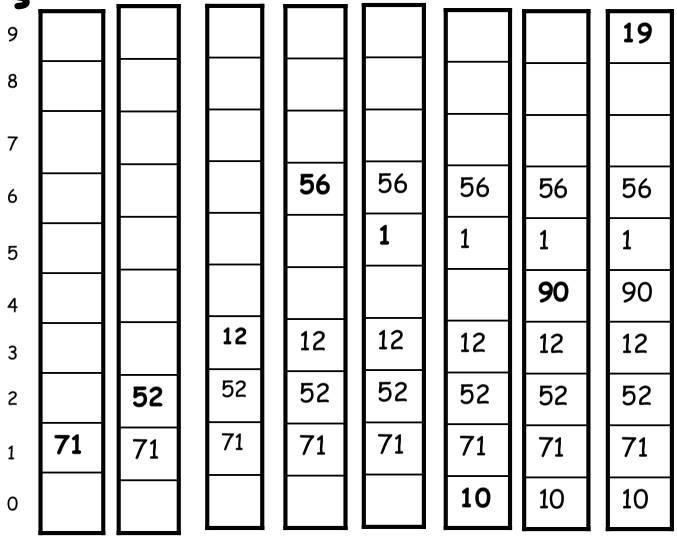
- Η ακολουθία εξέτασης που προκύπτει βάσει της μεθόδου της γραμμικής αναζήτησης είναι ίδια με εκείνη που προκύπτει βάσει της μεθόδου του διπλού κατακερματισμού όταν $h_2(K) = 1$, $\forall$ κλειδί K.

- Η ακολουθία εξέτασης πρέπει να περιέχει όλες τις θέσεις του πίνακα.
- Πρέπει να ισχύει ότι $h_2(K) > 0$.
- Τα $h_2(K)$ και m δεν πρέπει να έχουν κοινούς διαιρέτες.
  Έστω ότι ένας αριθμός d αποτελεί κοινό διαιρέτη του $h_2(K)$ και του m. Τότε:
  $[(m/d)h2(K)] \bmod m = [m(h2(K)/d)] \bmod m = 0$,
  Άρα, η (m/d)-οστή θέση στην ακολουθία θα είναι η ίδια όπως η πρώτη.
  Για το λόγο αυτό, διαλέγουμε τον m να είναι πρώτος αριθμός.

# Κατακερματισμός με Ανοικτή Διευθυνσιοδότηση Διπλός Κατακερματισμός

**Παράδειγμα**
Έστω ότι $h_1(K) = k \bmod m$ και $h_2(K) = K \bmod 3 + 1$.

Διαδοχική εισαγωγή των κλειδιών 71, 52, 12, 56, 1, 10, 90, 19 σε έναν πίνακα κατακερματισμού 10 θέσεων.




Είναι αξιοσημείωτο το τι συμβαίνει όταν γίνεται η εισαγωγή του 1. Η ακολουθία εξέτασης είναι 1, 1+2=3, 3+2=5, 5+2=7, 9, κ.ο.κ., αφού $h_2(1) = 1 \bmod 3 + 1 = 2$.

Η θέση 1 του πίνακα είναι κατειλημμένη, οπότε εξετάζεται το 2ο στοιχείο στην ακολουθία (δηλαδή η θέση 3 του πίνακα) η οποία είναι επίσης κατειλημμένη. Το 3ο στοιχείο της ακολουθίας είναι η θέση 5 του πίνακα, η οποία είναι διαθέσιμη. Άρα, το στοιχείο 1 αποθηκεύεται στην θέση 5 του πίνακα.



# Κατακερματισμός με Ανοικτή Διευθυνσιοδότηση Διπλός Κατακερματισμός

**Υπόθεση**
Κάθε θέση που εξετάζεται στον πίνακα κατακερματισμού είναι ανεξάρτητη από τις υπόλοιπες θέσεις του πίνακα, και η πιθανότητα να επιλεγεί μια κατειλημμένη θέση είναι ίση με τον παράγοντα φόρτου (α).

Η υπόθεση αυτή δεν είναι σωστή για τους ακόλουθους λόγους:
- Διαδοχικές θέσεις μπορεί να εξαρτώνται με κάποιο τρόπο.
- Είναι αδύνατο να εξεταστεί η ίδια θέση 2 φορές.

Ωστόσο, πολλές φορές τέτοιες υποθέσεις μας επιτρέπουν να πραγματοποιούμε μαθηματικές αναλύσεις που περιγράφουν ικανοποιητικά τη συμπεριφορά του συστήματος (όπως αυτή καθορίζεται από πειραματική μελέτη).

**Εισαγωγή n κλειδιών σε πίνακα κατακερματισμού μεγέθους m, δεδομένου ότι η υπόθεση είναι σωστή**
$a_i = i/m$, $i \leq n$: η πιθανότητα σύγκρουσης μετά την εισαγωγή i κλειδιών είναι $a_i$

# Κατακερματισμός με Ανοικτή Διευθυνσιοδότηση Διπλός Κατακερματισμός

**Πλήθος προσπελάσεων στη μνήμη για την εκτέλεση μη επιτυχημένης αναζήτησης**

Το αναμενόμενο πλήθος προσπελάσεων στη μνήμη για την εκτέλεση μιας μη επιτυχημένης αναζήτησης αν n-1 κλειδιά έχουν ήδη εισαχθεί στον πίνακα κατακερματισμού είναι:

$U_{n-1} = 1*(1 - a_{n-1}) + 2*a_{n-1}*(1-a_{n-1})+3a_{n-1}^2(1-a_{n-1}) + \ldots$

$= 1 + a_{n-1} + a_{n-1}^2 + \ldots$

$= 1/(1 - a_{n-1})$

**Πλήθος προσπελάσεων στη μνήμη για την εκτέλεση επιτυχημένης αναζήτησης**

Ισούται με το αναμενόμενο πλήθος προσπελάσεων στη μνήμη για την εκτέλεση της εισαγωγής κάθε ενός από τα n κλειδιά.

Το αναμενόμενο πλήθος προσπελάσεων στη μνήμη για την εισαγωγή του i-οστού κλειδιού = αναμενόμενο πλήθος προσπελάσεων στη μνήμη για την εκτέλεση μιας μη επιτυχημένης αναζήτησης. Επομένως:

$S_n = (1/n)\sum_{i=1}^{n} U_{i-1} = (1/n)\sum_{i=1}^{n}\frac{1}{1-a_{i-1}} = (m/n)\sum_{i=1}^{n} 1/(m-i+1)$

$= (m/n)\,(H_m - H_{m-n})$, όπου $H_i = 1 + \frac{1}{2} + \ldots + 1/i \approx \ln i$.

$S_n \approx (m/n)\,(\ln m - \ln (m-n))$
$= (m/n)\,\ln(m/(m-n))$
$= (1/a_n)\,\ln\,(1/(1-a_n))$.



# Ταξινομημένος Κατακερματισμός (Ordered Hashing)

**Μέθοδος Αλυσίδων**

«Τα κλειδιά διατηρούνται ταξινομημένα σε κάθε αλυσίδα».

Βελτιωμένη Έκδοση LookUp(A,e):
Αν ένα μεγαλύτερο κλειδί από το e προσπελαστεί τερματίζει η αναζήτηση.

**Μέθοδος Ανοικτής Διεύθυνσης**

Τα κλειδιά θα πρέπει να εισαχθούν στον πίνακα κατακερματισμού έτσι ώστε:
«Τα κλειδιά που προηγούνται του K στην ακολουθία εξέτασης (του K), θα πρέπει να είναι μικρότερα από το K».

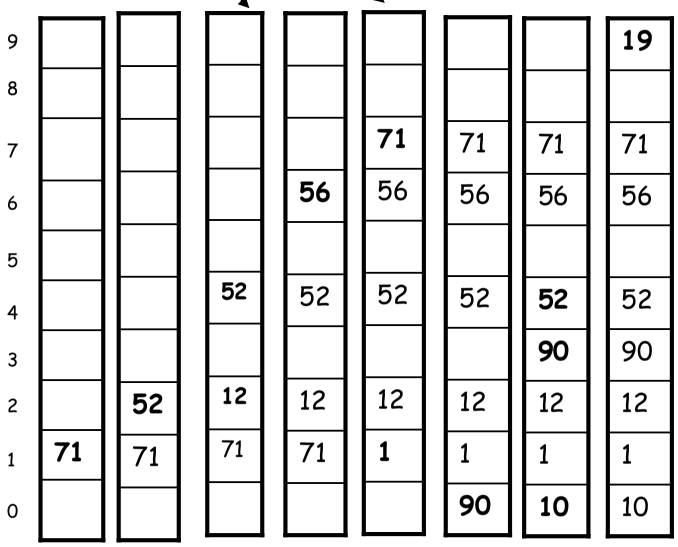
**Ιδέα**

Αν στην ακολουθία εξέτασης για το κλειδί K προσπελάσουμε κάποιο στοιχείο e' με κλειδί K' > K, τότε αντικαθιστούμε το e' με το e και συνεχίζουμε με την εισαγωγή του e' βάσει της ακολουθίας εξέτασης του κλειδιού K' του e'.

# Ταξινομημένος Κατακερματισμός Ανοικτής Διευθυνσιοδότησης

```
procedure OrderedHashTableInsert(
        HashTable A, Key K, Type I) {
   int pos = h1(K);
   while (A[pos] != NULL) {
      if (A[pos]->Key > K) {
         swap(K, A[pos]->Key);
         swap(I, A[pos]->Info);
      }
      else if (K == A[pos]->Key) {
         A[pos]->Info = I;
         return;
      }
      pos = (pos + h2(K)) mod m;
   }
   A[pos]->Key = K;
   A[pos]->Info = I;
}
```

**Παρατηρήστε τις ενέργειες που εκτελούνται κατά την εισαγωγή του κλειδιού 12, του 1 ή του 90!**




**Παράδειγμα**: Έστω ότι [$h_1(K)$ = k mod m] και [$h_2(K)$ = K mod 3 + 1]. Διαδοχική εισαγωγή των κλειδιών 71, 52, 12, 56, 1, 90, 10, 19 σε έναν πίνακα κατακερματισμού 10 θέσεων.



# Ταξινομημένος Κατακερματισμός Ανοικτής Διευθυνσιοδότησης

```
Type OrderedHashTableLookUp(HashTable A, Key K) {
    int pos = h1(K);
    while (A[pos] != NULL && A[pos]->Key < K) do
          pos = (pos + h2(K)) mod m;
    if (A[pos] != NULL && A[pos]->Key == K)
          return A[pos]->Info;
    else return NIL;
}
```

| |
|---|
| 19 |
| |
| 71 |
| 56 |
| |
| 52 |
| 90 |
| 12 |
| 1 |
| 10 |

**Άσκηση για το σπίτι**

Ιχνηλατίστε την εκτέλεση της OrderedHashTableLookUp(A,71), όπου A ο πίνακας του σχήματος.

Ιχνηλατίστε την εκτέλεση της OrderedHashTableLookUp(A,91), όπου A ο πίνακας του σχήματος.

# Ταξινομημένος Κατακερματισμός Ανοικτής Διευθυνσιοδότησης

- Η τελική μορφή του πίνακα κατακερματισμού, μετά την εισαγωγή σε αυτόν (βάσει της τεχνικής του ταξινομημένου κατακερματισμού) ενός συνόλου κλειδιών, θα είναι η ίδια ανεξαρτήτως της σειράς με την οποία τα κλειδιά αυτά εισάγονται στον πίνακα.

**Υπόθεση**
Η πιθανότητα να επιλεγεί ένα συγκεκριμένο κλειδί από το χώρο κλειδιών είναι η ίδια για όλα τα κλειδιά του χώρου.

**Τότε:**
Ο αναμενόμενος χρόνος $S_n$ για μια επιτυχημένη αναζήτηση δεν αλλάζει.
Ο αναμενόμενος χρόνος $U_n$ για μια μη επιτυχημένη αναζήτηση μειώνεται.
Γίνεται περίπου ίδιος με $S_n$.

**Άρα:**
$S_n \approx U_n \approx (1/a_n) \ln(1/(1 - a_n))$.



# Κατακερματισμός - Διαγραφές

**Μέθοδος αλυσίδων**
Υλοποιείται εύκολα. Πως?

**Μέθοδος Ανοικτής Διευθυνσιοδότησης**
Ένα κλειδί δεν μπορεί να διαγραφεί αφήνοντας απλά τη θέση που κατείχε άδεια. Γιατί?

**Παράδειγμα**
Έστω ότι χρησιμοποιούμε τη μέθοδο της γραμμικής αναζήτησης.
Έστω ότι δύο κλειδιά με την ίδια βασική/πρωταρχική τιμή κατακερματισμού εισάγονται στις θέσεις j και j+1 ενός πίνακα κατακερματισμού και στη συνέχεια αυτό στη θέση j διαγράφεται.
Το άλλο θα μείνει στη θέση j+1, αλλά η LookUp() θα σταματήσει όταν βρει τη θέση j κενή. Αυτό θα οδηγήσει την εν λόγω LookUp() σε λάθος απόκριση!

**Ιδέα**
Για κάθε θέση του πίνακα υπάρχει ένα bit, που ονομάζεται Deleted και μπορεί να είναι είτε 0 ή 1. Αρχικά όλα αυτά τα bits είναι 0. Αν διαγράψουμε κάτι από μια θέση του πίνακα, θέτουμε το bit της θέσης αυτής σε 1. Η LookUp() δεν τερματίζει σε άδειες θέσεις για τις οποίες το Deleted bit είναι 1.

# Επεκτάσιμος Κατακερματισμός (Extendible Hashing)

✓ Η μέθοδος ενδείκνυται για την αποθήκευση στοιχείων σε δευτερεύουσα μνήμη δίσκου.

✓ Οι δίσκοι είναι πολύ πιο αργές μονάδες αποθήκευσης συγκριτικά με την κύρια μνήμη. Για το λόγο αυτό, τα δεδομένα οργανώνονται σε σελίδες (pages), δηλαδή αποθηκεύονται σε ομάδες συνεχόμενων θέσεων αποθήκευσης, τα περιεχόμενα των οποίων μεταφέρονται στην κύρια μνήμη όταν ζητηθεί η ανάκτηση ή η τροποποίηση κάποιων εξ αυτών.

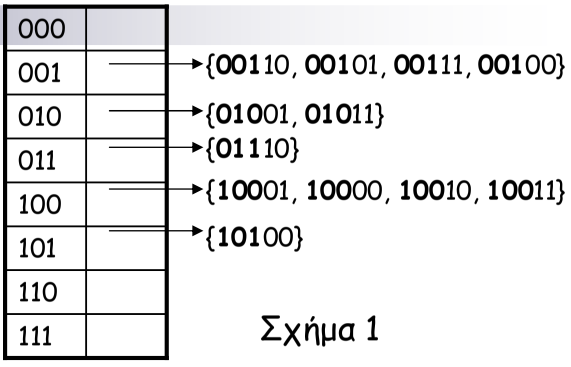
**Ολικό βάθος D(S)** ενός συνόλου S καλείται το μικρότερο μήκος προθέματος των τιμών κατακερματισμού των στοιχείων του S, που απαιτείται για να χωρισθούν τα στοιχεία αυτά σε ομάδες των b στοιχείων, το πολύ, σε κάθε μια, όπου όλα τα στοιχεία μιας ομάδας έχουν κοινό πρόθεμα μήκους D(S) bits.

**Παράδειγμα**
Θεωρούμε ότι b=4. Έστω ότι:
S = {6 $(00110)_2$, 9 $(01001)_2$, 14 $(01110)_2$, 17 $(10001)_2$, 5 $(00101)_2$, 7 $(00111)_2$, 16 $(10000)_2$, 20 $(10100)_2$, 18 $(10010)_2$, 19 $(10011)_2$, 4 $(00100)_2$, 11 $(01011)_2$}.
Είναι D(S) = 3 και οι ομάδες που δημιουργούνται είναι οι εξής:
{**00**110, **00**101, **00**111, **00**100}, {**010**01, **010**11}, {**011**10},
{**100**01, **100**00, **100**10, **100**11}, {**101**00}.



# Επεκτάσιμος Κατακερματισμός

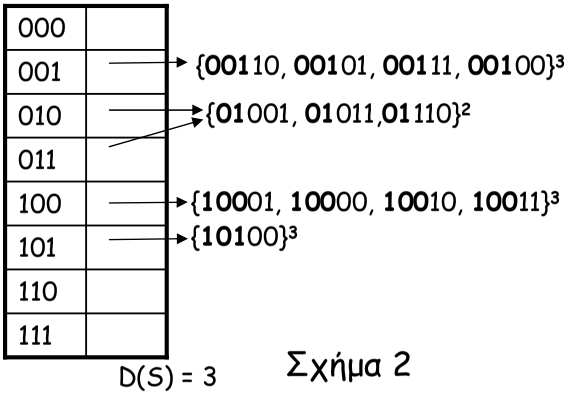
- Κάθε ομάδα αποθηκεύεται σε διαφορετική σελίδα της δομής του επεκτάσιμου κατακερματισμού.
- Μπορούμε να χρησιμοποιήσουμε έναν πίνακα μεγέθους $2^{D(S)}$ για τη δεικτοδότηση των σελίδων (Σχήμα 1).



Σχήμα 1

Είναι δυνατό να συνενωθούν σελίδες με συνολικό πλήθος στοιχείων b και μικρότερο κοινό πρόθεμα. Το πρόθεμα αυτό καλείται **βάθος σελίδας**.

**Παράδειγμα**
{**010**01,**010**11,**011**10} ∪ {**011**10} =
{**010**01, **010**11,**011**10,**011**10} (Σχήμα 2).
Τέτοιου είδους σελίδες χαρακτηρίζονται ως **φιλικές σελίδες (buddies)**.
Τα στοιχεία φιλικών σελίδων διαφέρουν στο τελευταίο ψηφίο του προθέματός τους.

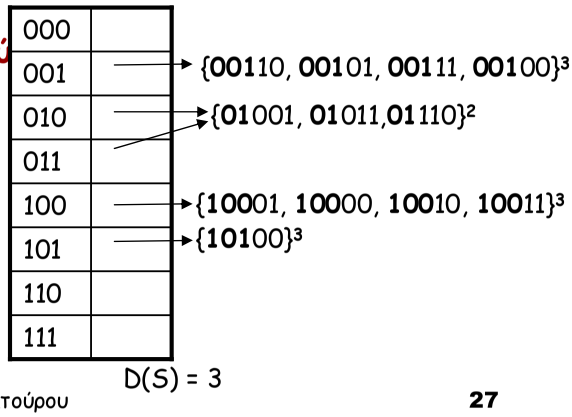


Σχήμα 2

Αν για μια σελίδα ισχύει ότι το βάθος της d είναι μικρότερο από το ολικό βάθος D, τότε $2^{D-d}$ δείκτες σε συνεχόμενες θέσεις του πίνακα θα δείχνουν στη σελίδα αυτή.

# Επεκτάσιμος Κατακερματισμός

- Ένας **επεκτάσιμος πίνακας κατακερματισμού** είναι δομή δύο επιπέδων. Αποτελείται από έναν κατάλογο (directory) που δημιουργεί τη δομή υψηλού επιπέδου και ένα σύνολο από σελίδες (pages) στις οποίες αποθηκεύονται τα δεδομένα.
- Ο **κατάλογος** είναι ένας πίνακας από δείκτες στις σελίδες όπου αποθηκεύονται τα δεδομένα. Οι σελίδες είναι σταθερού μεγέθους, π.χ., χωράει b στοιχεία/δεδομένα η κάθε μια.
- Μια συνάρτηση κατακερματισμού απεικονίζει κάθε κλειδί K σε μια ακολουθία από bits μήκους L. Η ακολουθία αυτή ονομάζεται **τιμή κατακερματισμού** του K.
- Για κάθε $d \leq L$, συμβολίζουμε με $h_d(K)$ τα πρώτα d bits του h(K).
- Το μέγιστο βάθος μεταξύ όλων των σελίδων ονομάζεται ***βάθος* του πίνακα κατακερματισμού** και συμβολίζεται με το D.
- Ο κατάλογος είναι ένας πίνακας με $2^D$ δείκτες σε σελίδες.




**Εύρεση σελίδας που περιέχει το κλειδί K**

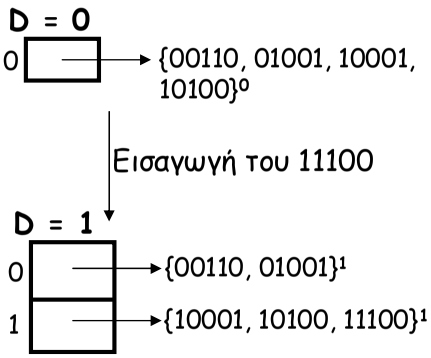
1. Υπολογισμός του $h_D(K)$;
2. Ο δείκτης που περιέχεται στο A[hD(K)] οδηγεί στην σελίδα στην οποία θα πρέπει να είναι αποθηκευμένο το δεδομένο;

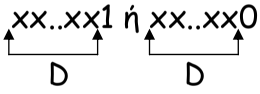


# Επεκτάσιμος Κατακερματισμός

**Εισαγωγές**

1. Εκτελούμε μια αναζήτηση, ώστε να εντοπιστεί η σελίδα τοποθετήσεως p η οποία έστω ότι έχει βάθος d. (Αρχικά D=0 και υπάρχει μια μόνο σελίδα στον πίνακα που περιέχει όλα τα δεδομένα. Ο κατάλογος περιέχει έναν μόνο δείκτη στη σελίδα αυτή και όλες οι αναζητήσεις καταλήγουν εκεί. )
2. Αν τα πλήθος των στοιχείων που είναι αποθηκευμένα στην p είναι < b, το νέο στοιχείο τοποθετείται στην p και ο αλγόριθμος τερματίζει.
3. Διαφορετικά, συμβαίνει υπερχείλιση και η p θα πρέπει να χωριστεί σε δύο σελίδες. Ο χωρισμός γίνεται με αύξηση του βάθους της p στην τιμή d+1 και με τη δημιουργία μιας νέας φιλικής σελίδας p' βάθους επίσης d+1.
4. Τα (b+1) κλειδιά της p κατανέμονται στις p, p' βάσει των d+1 πρώτων ψηφίων τους.
5. Αν αυτό δεν επιλύσει το πρόβλημα (δηλαδή όλα τα b+1 κλειδιά ανήκουν και πάλι σε μια από τις p, p', έστω στην p'), θέτω p = p' και d = d+1 και επαναλαμβάνω το Βήμα 3.
6. Αν μετά από D-d επαναλήψεις, η υπερχείλιση δεν έχει αντιμετωπιστεί, είναι αναγκαίος ο διπλασιασμός του καταλόγου (δηλαδή το ολικό βάθος αλλάζει σε D+1). Ο νέος κατάλογος δεικτοδοτείται από δείκτες της μορφής xx..xx1 ή xx..xx0.
7. Περισσότεροι του ενός διπλασιασμοί του καταλόγου μπορεί να απαιτούνται αν το πρόβλημα δεν επιλυθεί.

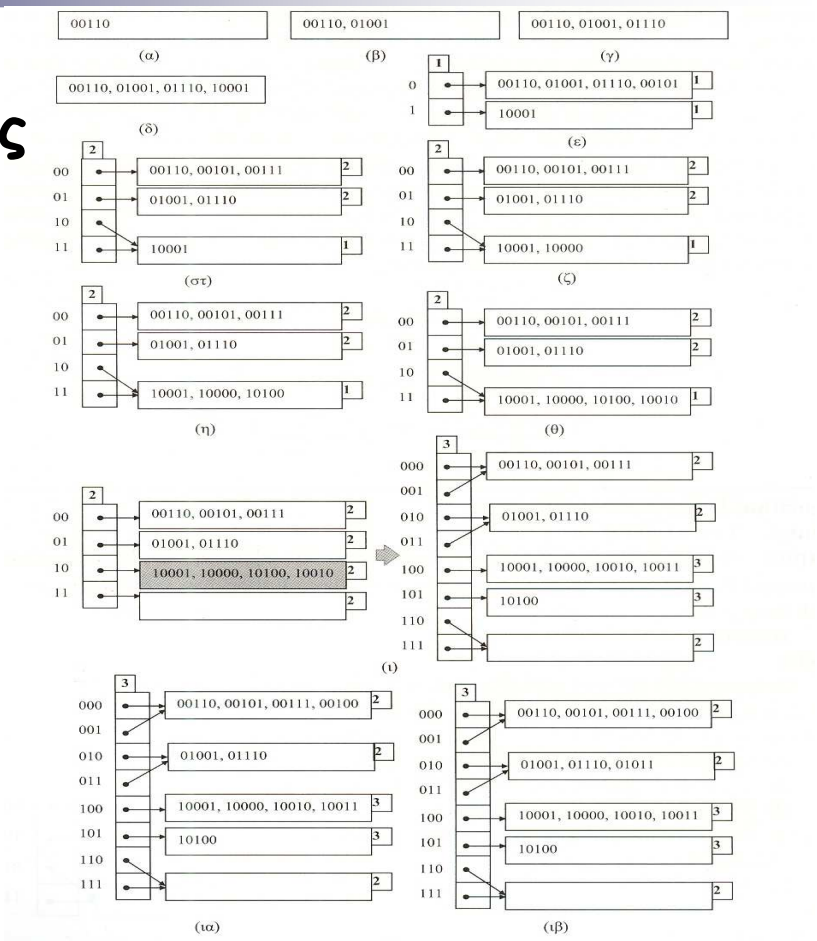


# Επεκτάσιμος Κατακερματισμός Εισαγωγή

**Παράδειγμα**

Διαδοχικές εισαγωγές των 6 (00110), 9 (01001), 14 (01110), 17 (10001), 5 (000101), 7 (000111), 16 (10000), 20 (10100), 18 (10010), 19 (10011), 4 (00100), 11 (01011) σε επεκτάσιμο πίνακα κατακερματισμού.




# Επεκτάσιμος Κατακερματισμός - Διαγραφή

Οι ενέργειες που πραγματοποιούνται είναι «συμμετρικές» της λειτουργίας της εισαγωγής.

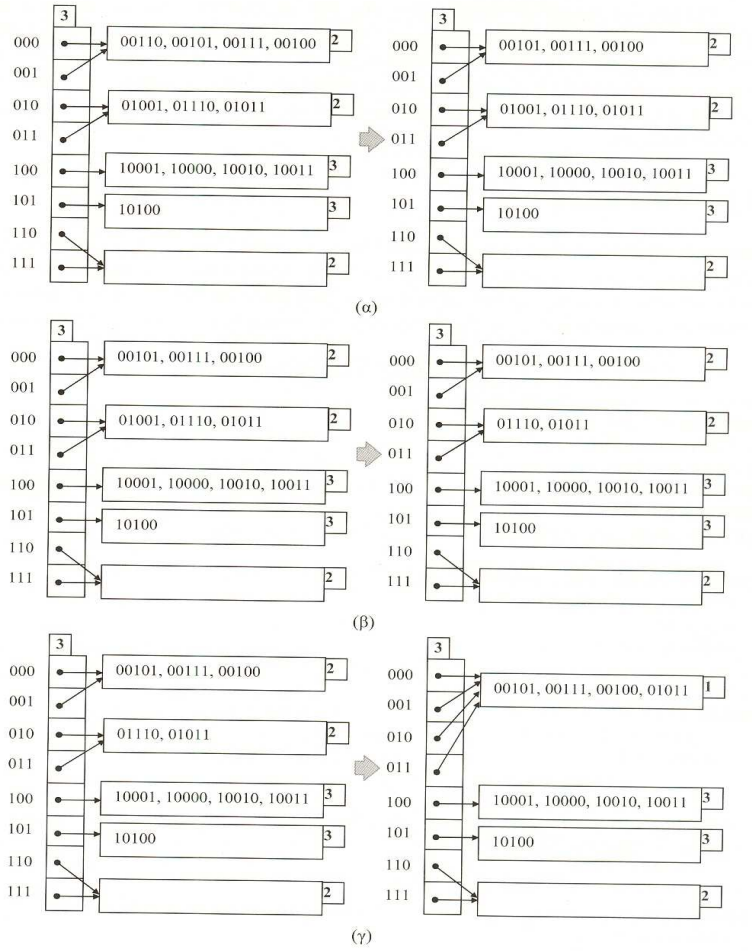
1. Εκτελούμε μια αναζήτηση, ώστε να εντοπιστεί η σελίδα p στην οποία είναι αποθηκευμένο το προς διαγραφή κλειδί. Έστω ότι η σελίδα αυτή έχει βάθος d.
2. Το στοιχείο διαγράφεται από την p.
3. Προσπελάζεται η φιλική σελίδα, έστω p', της p (εάν υπάρχει).
4. Εάν το συνολικό πλήθος στοιχείων των p,p' είναι ≤ b, τα αποθηκεύουμε όλα στην p μειώνοντας το βάθος της σε d-1.
5. Εάν μετά τη μείωση, όλες οι σελίδες έχουν βάθος < D, τότε υποδιπλασιάζουμε τον κατάλογο και τα αποθηκευμένα στοιχεία κατανέμονται στις σελίδες λαμβάνοντας υπ' όψιν το πολύ D-1 ψηφία.
6. Ο κατάλογος υποδιπλασιάζεται, διατηρώντας για κάθε ζεύγος θέσεων της μορφής xx..xx0 και xx...xx1, μια θέση της μορφής xx...xx.

D-1 D-1 D-1

# Επεκτάσιμος Κατακερματισμός Διαγραφή

**Παράδειγμα**

Διαδοχικές διαγραφές των 6 (00110), 9 (01001), 14 (01110) από επεκτάσιμο πίνακα κατακερματισμού.




# Συναρτήσεις Κατακερματισμού

**Αποδοτικές Συναρτήσεις Κατακερματισμού**

Ικανοποιούν την παραδοχή του απλού ομοιόμορφου κατακερματισμού:
«Θα πρέπει να είναι εξίσου πιθανό η τιμή κατακερματισμού του κάθε κλειδιού να είναι οποιαδήποτε από τις τιμές {0, …, m-1}, ανεξάρτητα από τις τιμές κατακερματισμού των υπολοίπων κλειδιών.»

Συνήθως δεν είναι δυνατό να ελέγξουμε αν ισχύει αυτή η παραδοχή:
- Σπανίως γνωρίζουμε την κατανομή πιθανότητας που ακολουθούν τα κλειδιά.
- Μπορεί αυτά να μην είναι ανεξάρτητα μεταξύ τους.

Σε ορισμένες περιπτώσεις, η κατανομή των κλειδιών είναι γνωστή.

**Παράδειγμα**
Αν γνωρίζουμε ότι τα κλειδιά είναι πραγματικοί αριθμοί κατανεμημένοι ανεξάρτητα και ομοιόμορφα στο διάστημα [0,1), τότε η συνάρτηση κατακερματισμού $h(k) = \lfloor km \rfloor$ ικανοποιεί τη συνθήκη του απλού ομοιόμορφου κατακερματισμού.

Στην πράξη, συχνά δημιουργούμε συναρτήσεις κατακερματισμού με καλή συμπεριφορά βασιζόμενοι σε ευρετικές τεχνικές.

# Ερμηνεία των κλειδιών ως Φυσικών Αριθμών

Οι περισσότερες συναρτήσεις κατακερματισμού προϋποθέτουν ότι ο χώρος των κλειδιών είναι το σύνολο N των φυσικών αριθμών.
Συχνά αυτό δεν ισχύει ⇒ θα πρέπει να βρεθεί τρόπος να μετατραπεί το εκάστοτε κλειδί σε φυσικό αριθμό.

**Παράδειγμα**
Έστω ότι ένα κλειδί K είναι αλφαριθμητικό (string).
Υπολογίζουμε το $\sum_{i=0}^{p-1} c_i r^i$, όπου

- p: μήκος του αλφαριθμητικού
- r: ο αριθμός χαρακτήρων στον κώδικα (συνήθως 128 ή 256)
- $c_j$: ο κωδικός (ASCII) κάθε χαρακτήρα

Η συμβολοσειρά εκφράζεται ως ακέραιος στο αριθμητικό σύστημα με βάση το 128 (αφού ο πληθικός αριθμός του συνόλου χαρακτήρων ASCII είναι το 128).



# Συναρτήσεις Κατακερματισμού

**Κατακερματισμός βάσει της πράξης της Διαίρεσης**
Συνάρτηση κατακερματισμού: h(K) = K mod m,
Όπου m είναι το μέγεθος του πίνακα κατακερματισμού και K ένα κλειδί στο U.

**Σκοπός:** Αποφυγή συστηματικών συγκρούσεων σε περιπτώσεις που τα κλειδιά επιλέγονται με ένα συστηματικά μη τυχαίο τρόπο.

**Παραδείγματα**
1. Αν το m είναι r ή $r^2$, το αποτέλεσμα της διαίρεσης είναι ο κωδικός του ενός ή των δύο τελευταίων χαρακτήρων.
   **Όμως,** από τα γράμματα του αλφάβητου είναι πολύ λίγα αυτά που συνήθως συναντούνται ως τελευταίοι χαρακτήρες λέξεων.
2. Αν το m είναι άρτιος, η τιμή h(K) θα είναι άρτια ή περιττή ανάλογα με το αν ο τελευταίος χαρακτήρας στο αλφαριθμητικό έχει περιττό ή άρτιο κωδικό.

# Συναρτήσεις Κατακερματισμού

**Κατακερματισμός βάσει της πράξης της Διαίρεσης**

Το m επιλέγεται να είναι πρώτος αριθμός.

Επίσης, είναι καλύτερο το m να μην διαιρεί τους $r^k+a$, $r^k-1$ για μικρές σταθερές k και a.

**Παράδειγμα**

Έστω m = r-1 και έστω ότι ο r-1 είναι πρώτος. Τότε:

$$\sum_{i=0}^{p-1} c_i r^i \bmod (r-1) = \sum_{i=0}^{p-1} (c_i r^i \bmod (r-1)) \bmod (r-1) = (\sum_{i=0}^{p-1} c_i) \bmod (r-1)$$

Μπορεί να αποδειχθεί επαγωγικά πως, για κάθε i,

$r^i \bmod (r-1) = (1 + (r-1) \sum_{j=0}^{i-1} r^j) \bmod (r-1) = 1.$

Άρα, αν m = r-1, όλες οι μεταθέσεις του ίδιου συνόλου χαρακτήρων (π.χ., ABC, BCA, CBA, κλπ.) έχουν την ίδια τιμή κατακερματισμού.



# Συναρτήσεις Κατακερματισμού

**Κατακερματισμός βάσει της πράξης του Πολλαπλασιασμού**

1. Πολλαπλασιάζουμε το κλειδί K με μια σταθερά A στο διάστημα 0 < A < 1 και κρατάμε το δεκαδικό μέρος του γινομένου KA.
2. Πολλαπλασιάζουμε αυτή την ποσότητα με m και παίρνουμε το κάτω ακέραιο μέρος του αποτελέσματος.

**Συνάρτηση Κατακερματσμού**

$h(K) = \lfloor m(KA - \lfloor KA \rfloor) \rfloor$

- Η ποσότητα $KA - \lfloor KA \rfloor$ είναι το δεκαδικό μέρος της ποσότητας KA.
- Η μέθοδος μπορεί να εφαρμοστεί με οποιαδήποτε τιμή της σταθεράς A. Η βέλτιστη επιλογή εξαρτάται από τα χαρακτηριστικά των δεδομένων που θα αποθηκευθούν στον πίνακα κατακερματισμού.
- Η τιμή $A = (\sqrt{5} - 1)/2 = 0{,}6180339887\ldots$ θεωρείται ότι δίνει καλά αποτελέσματα.

**Πλεονέκτημα της μεθόδου**

1. Η τιμή του m δεν έχει καθοριστική σημασία. Το m μπορεί να είναι οποιαδήποτε δύναμη του 2.

# Συναρτήσεις Κατακερματισμού

**Τέλειος Κατακερματισμός Στατικών Δεδομένων**

- Αν γνωρίζαμε εξ αρχής τα κλειδιά που θέλουμε να αποθηκευτούν, ίσως να μπορούσαμε να σχεδιάσουμε μια συνάρτηση κατακερματισμού που να αποφεύγει εντελώς τις συγκρούσεις.

- Έχουν αναπτυχθεί διάφορες τεχνικές για την εύρεση τέλειων συναρτήσεων κατακερματισμού για δεδομένα σύνολα κλειδιών.

- Η μέθοδος έχει περιορισμένη εφαρμογή, αφού συνήθως το σύνολο των κλειδιών δεν είναι γνωστό και τα δεδομένα αλλάζουν δυναμικά.



# Καθολικές Κλάσεις Συναρτήσεων Κατακερματισμού

**Πρόβλημα**
Ένας κακόβουλος αντίπαλος (adversary) που γνωρίζει τη συνάρτηση κατακερματισμού που χρησιμοποιεί ένας αλγόριθμος μπορεί να επιλέξει τα προς αποθήκευση κλειδιά έτσι ώστε να έχουν όλα την ίδια τιμή κατακερματισμού.

☹ Σε αυτή την περίπτωση η χρονική πολυπλοκότητα της LookUp() είναι γραμμική στο n.

**Επίλυση Προβλήματος**
Επιλέγουμε τη συνάρτηση κατακερματισμού με τυχαίο τρόπο, ανεξάρτητο από τα κλειδιά που πρόκειται να αποθηκευθούν τελικά στη δομή κατακερματισμού, από μια προσεχτικά σχεδιασμένη κλάση συναρτήσεων κατά την εκκίνηση της εκτέλεσης. Η προσέγγιση αυτή ονομάζεται **καθολικός κατακερματισμός**.

Λόγω της τυχαιότητας, ο αλγόριθμος μπορεί να συμπεριφέρεται διαφορετικά σε κάθε εκτέλεση, ακόμη και για την ίδια είσοδο.

➧ Με τον τρόπο αυτό εξασφαλίζεται καλή αναμενόμενη επίδοση για οποιαδήποτε είσοδο.

# Καθολικές Κλάσεις Συναρτήσεων Κατακερματισμού

**Ορισμός**

Έστω H μια πεπερασμένη συλλογή συναρτήσεων κατακερματισμού οι οποίες απεικονίζουν ένα δεδομένο χώρο κλειδιών U στο διάστημα {0,1,…,m-1}.
Μια τέτοια συλλογή λέγεται **καθολική** εάν για κάθε ζεύγος διαφορετικών κλειδιών $k,l \in U$, το πλήθος των συναρτήσεων κατακερματισμού $h \in H$ για τις οποίες $h(k) = h(l)$ είναι μικρότερο ή ίσο του $|H|/m$.

**Παρατήρηση**

Με μια συνάρτηση κατακερματισμού που επιλέγεται τυχαία από μια καθολική κλάση συναρτήσεων H, η πιθανότητα να συμβεί μια σύγκρουση μεταξύ δύο διαφορετικών κλειδιών k και l είναι μικρότερη ή ίση της πιθανότητας 1/m να συνέβαινε μια σύμπτωση αν τα h(k) και h(l) επιλέγονταν τυχαία και ανεξάρτητα μεταξύ τους από το σύνολο {0,…,m-1}.



# Καθολικές Κλάσεις Συναρτήσεων Κατακερματισμού

**Θεώρημα**

Έστω ότι επιλέγουμε μια συνάρτηση κατακερματισμού h από μια καθολική κλάση συναρτήσεων H και έστω ότι χρησιμοποιούμε την h για να αποθηκεύσουμε n κλειδιά σε έναν πίνακα κατακερματισμού A των m θέσεων επιλύοντας τις συγκρούσεις με τη μέθοδο των ξεχωριστών αλυσίδων.

1. Αν ένα κλειδί K δεν βρίσκεται στον πίνακα, τότε το αναμενόμενο μήκος $E[n_{h(K)}]$ της αλυσίδας στην οποία αποθηκεύεται το K είναι μικρότερο ή ίσο του α.
2. Αν το κλειδί K βρίσκεται στον πίνακα κατακερματισμού, τότε το αναμενόμενο μήκος $E[n_{h(K)}]$ της αλυσίδας που περιέχει το K είναι μικρότερο ή ίσο του 1+α.

**Παρατήρηση**

- Οι αναμενόμενες τιμές αναφέρονται στις διάφορες συναρτήσεις κατακερματισμού. Δεν έχει γίνει κάποια παραδοχή όσον αφορά την κατανομή των κλειδιών.

# Καθολικές Κλάσεις Συναρτήσεων Κατακερματισμού

**Απόδειξη Θεωρήματος**

Για κάθε ζεύγος διαφορετικών κλειδιών k και l, ορίζουμε την (δείκτρια) τυχαία μεταβλητή

$$X_{kl} = \begin{cases} 1 & \text{if } h(k) = h(l) \\ 0 & \text{otherwise} \end{cases}$$

Δεδομένου ότι η πιθανότητα σύγκρουσης δύο οποιονδήποτε κλειδιών είναι εξ ορισμού μικρότερη ή ίση του 1/m, έχουμε $Pr\{h(k) = h(l)\} \leq 1/m$.

Άρα, $E[X_{kl}] = 1 * Pr\{h(k)=h(l)\} + 0 * (1-Pr\{h(k)=h(l)\}) \leq 1/m$.

Για κάθε κλειδί k, ορίζουμε την τυχαία μεταβλητή $Y_k$ η οποία ισούται με το πλήθος των διαφορετικών κλειδιών από το k που αποθηκεύονται στην ίδια θέση του πίνακα κατακερματισμού με το k. Ισχύει ότι $Y_k = \sum_{\substack{l \in A \\ l \neq k}} X_{kl}$

Επομένως: $$E[Y_k] = E[\sum_{\substack{l \in A \\ l \neq k}} X_{kl}] = \sum_{\substack{l \in A \\ l \neq k}} E[X_{kl}] \leq \sum_{\substack{l \in A \\ l \neq k}} \frac{1}{m}$$

Αν $k \notin A$, $n_{h(k)} = Y_k$ και $|\{l \in A \text{ και } l \neq k\}| = n$. Επομένως, $E[n_{h(k)}] = E[Y_k] \leq n/m = a$.

Αν $k \in A$, επειδή το k ανήκει στην αλυσίδα $A[h(k)]$ και το πλήθος Yκ δεν περιλαμβάνει το k, έχουμε $n_{h(k)} = Y_k+1$ και $|\{l \in A \text{ και } l \neq k\}| = n-1$. Επομένως, $E[n_{h(k)}] = E[Y_k] +1 \leq (n-1)/m +1 = 1 + a - 1/m < 1+a$.



# Καθολικές Κλάσεις Συναρτήσεων Κατακερματισμού

**Πόρισμα**

Η μέθοδος του καθολικού κατακερματισμού σε συνδυασμό με τη μέθοδο επίλυσης συγκρούσεων των ξεχωριστών αλυσίδων σε πίνακα με m θέσεις εξασφαλίζει ότι ο αναμενόμενος χρόνος διεκπεραίωσης οποιασδήποτε ακολουθίας n λειτουργιών Insert(), LookUp() και Delete() οι οποίες περιλαμβάνουν $O(m)$ λειτουργίες Insert() είναι $\Theta(n)$.

**Απόδειξη**

- Αφού το πλήθος των λειτουργιών εισαγωγής είναι $O(m)$ ισχύει ότι $n = O(m)$ και επομένως $a = O(1)$.
- Βάσει του Θεωρήματος, ο αναμενόμενος χρόνος για κάθε αναζήτηση είναι $O(1)$ (αφού $a=O(1)$).
- Επομένως, ο αναμενόμενος χρόνος για την εκτέλεση όλων των λειτουργιών είναι $O(n)$.
- Το κάτω φράγμα $\Omega(n)$ προκύπτει από το ότι κάθε λειτουργία απαιτεί χρόνο $\Omega(1)$.

# Δημιουργία μιας Καθολικής Κλάσης Συναρτήσεων Κατακερματισμού

Επιλέγουμε έναν πρώτο αριθμό p αρκετά μεγάλο ώστε όλα τα δυνατά κλειδιά K να βρίσκονται στο διάστημα {0,.., p-1}.

**Θεώρημα**

Για κάθε αριθμό $a \in \{1,\ldots,p-1\}$ και $b \in \{0,\ldots,p-1\}$ έστω $h_{a,b}(x) = ((ax+b) \bmod N) \bmod m$.

Τότε, το σύνολο συναρτήσεων $H = \{h_{a,b}: 1 \leq a < p \text{ και } 0 \leq b < p\}$ είναι μια καθολική κλάση συναρτήσεων κατακερματισμού.

**Απόδειξη**

Χρησιμοποιεί απλά επιχειρήματα από τη θεωρία αριθμών.

**Παρατηρήσεις**

✓ Το μέγεθος m του πίνακα μπορεί να είναι οποιοσδήποτε ακέραιος και όχι απαραίτητα πρώτος, ούτε καν περιττός. Π.χ., το m μπορεί να είναι μια δύναμη του 2.

✓ Η μέθοδος μπορεί να χρησιμοποιηθεί σε συνδυασμό με τη μέθοδο του επεκτάσιμου κατακερματισμού.



# Ενότητα 8
# Ουρές Προτεραιότητας

# Ουρές Προτεραιότητας

Θεωρούμε ένα χώρο κλειδιών U και έστω ότι με κάθε κλειδί K (τύπου Key) έχει συσχετισθεί κάποια πληροφορία I (τύπου Type). Έστω επίσης ότι έχει ορισθεί μια διάταξη στα στοιχεία του U έτσι ώστε για κάθε ζεύγος στοιχείων $x,y \in U$, ισχύει είτε $x = y$ ή $x < y$ ή $x > y$.

Τα στοιχεία που είναι επιθυμητό να αποθηκευθούν στη δομή είναι ζεύγη της μορφής <K,I>.

Μια **ουρά προτεραιότητας** είναι ένας αφηρημένος τύπος δεδομένων για ένα σύνολο με στοιχεία ζεύγη <K,I>, που υποστηρίζει τις ακόλουθες λειτουργίες:

- *MakeEmptySet():* επιστρέφει το κενό σύνολο ∅.
- *IsEmptySet(S):* Επιστρέφει true αν S = ∅, false διαφορετικά
- *Insert(K,I,S):* Εισάγει το ζεύγος <K,I> στο S.
- *FindMin(S):* επιστρέφει το info πεδίο I του ζεύγους <K,I>, όπου K είναι το μικρότερο κλειδί στο σύνολο.
- *DeleteMin(S):* Διαγράφει το ζεύγος <K,I>, όπου K είναι το μικρότερο κλειδί στο σύνολο, και επιστρέφει I.



# Ουρές Προτεραιότητας

**Υλοποίηση με Ισοζυγισμένα Δένδρα**

Μπορούμε να υλοποιήσουμε μια ουρά προτεραιότητας με ισοζυγισμένα δένδρα;

AVL δένδρα, 2-3 δένδρα, Red-black δένδρα, B-δένδρα: όλα παρέχουν υλοποιήσεις ουρών προτεραιοτήτων.

Δεδομένου ενός ισοζυγισμένου δένδρου, πως θα μπορούσαμε να υλοποιήσουμε μια ουρά προτεραιότητας;

Ποια είναι η χρονική πολυπλοκότητα για τις λειτουργίες Insert(), FindMin() και DeleteMin();

Υπάρχουν άλλες λειτουργίες που να υποστηρίζονται αποδοτικά;

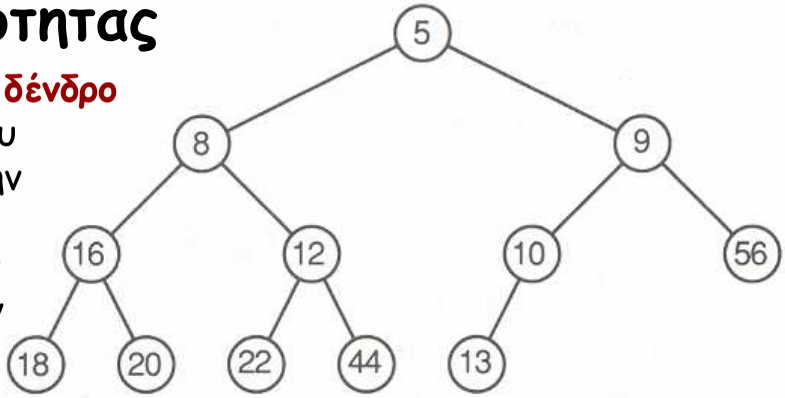
(1) LookUp; (2) Delete; (3) FindMax – DeleteMax;

Ασκήσεις για το σπίτι!

Μια ουρά προτεραιότητας που υποστηρίζει και τις λειτουργίες FindMax() και DeleteMax() ονομάζεται **διπλή ουρά προτεραιότητας** (ή **ουρά προτεραιότητας με δύο άκρα**).

# Ουρές Προτεραιότητας

Ένα **μερικώς διατεταγμένο δένδρο** είναι ένα δυαδικό δένδρο του οποίου τα στοιχεία έχουν την εξής ιδιότητα:
Το κλειδί κάθε κόμβου είναι μικρότερο ή ίσο εκείνου των παιδιών του κόμβου.




Θεωρούμε ότι η προτεραιότητα ενός κόμβου καθορίζεται από το κλειδί του ως εξής μικρό κλειδί -> μεγάλη προτεραιότητα και το αντίστροφο. Έτσι, η διάταξη των κόμβων του σχήματος από μεγαλύτερες προς μικρότερες προτεραιότητες είναι 5, 8, 9, 10, 12, 13, 16, 18, 20, 22, 44, 56.

Ποιος είναι ο κόμβος με τη μεγαλύτερη προτεραιότητα σε ένα μερικώς διατεταγμένο δένδρο;

- Σε κάθε μονοπάτι από τη ρίζα προς οποιοδήποτε κόμβο, οι κόμβοι που διατρέχουμε είναι φθίνουσας προτεραιότητας.

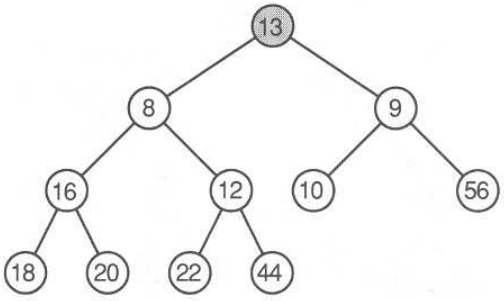
Πως υλοποιούμε την FindMin() σε μερικώς διατεταγμένο δένδρο;



# Ουρές Προτεραιότητας

Προκειμένου οι λειτουργίες αυτές να υλοποιηθούν αποδοτικά, θα ήταν επιθυμητό το ύψος του δένδρου να είναι $O(\log n)$.

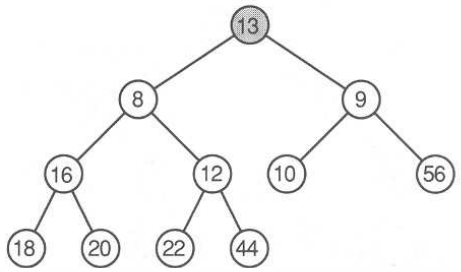
**DeleteMin()**
Δεν διαγράφουμε τη ρίζα, αλλά το δεξιότερο φύλλο του τελευταίου επιπέδου του δένδρου, αφού πρώτα αντιγράψουμε τα δεδομένα του στη ρίζα




**Πρόβλημα που μπορεί να προκύψει**
Το προκύπτον δένδρο μπορεί να μην είναι μερικώς διατεταγμένο δένδρο.

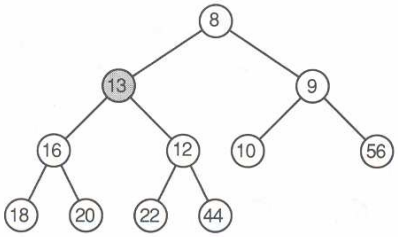
**Παρατήρηση**
Η μερική διάταξη καταστρέφεται μόνο στη ρίζα.

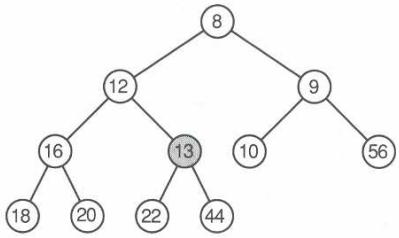
# Μερικώς Διατεταγμένα Δένδρα Διαγραφή




**DeleteMin() - Επανόρθωση διάταξης**

1. Έστω u η ρίζα του δένδρου και w ο θυγατρικός κόμβος του u με τη μεγαλύτερη προτεραιότητα.
2. Αν (w == nill OR κλειδί του u > κλειδί του w) ο αλγόριθμος τερματίζει;
3. Ανταλλάσσουμε τα δεδομένα του u με τα δεδομένα του w;
4. Θέτω (u = w) και (w = θυγατρικός κόμβος του u με τη μεγαλύτερη προτεραιότητα);
5. Επιστρέφουμε στο βήμα 2;

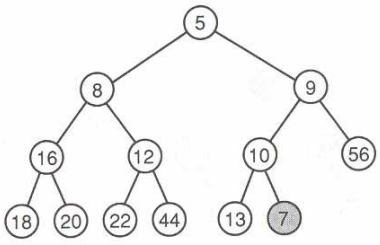


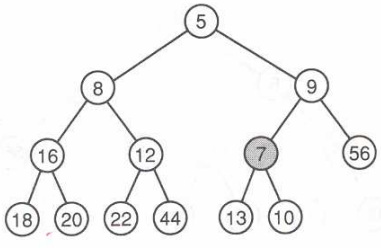

Ποια είναι πολυπλοκότητα της DeleteMin(); O(ύψος δένδρου)



# Μερικώς Διατεταγμένα Δένδρα



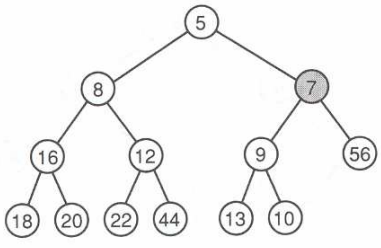

(a)




(b)

**Insert()**

- Εισάγουμε το νέο στοιχείο ως δεξιότερο φύλλο του δένδρου.
- Η ιδιότητα της μερικής διάταξης ίσως καταστρέφεται άλλα μόνο για τον πατέρα του φύλλου αυτού.



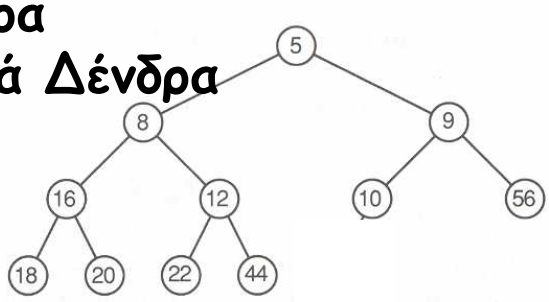

(c)

**Insert() - Επανόρθωση διάταξης**

1. Έστω u η ρίζα του δένδρου και w ο θυγατρικός κόμβος του u με τη μεγαλύτερη προτεραιότητα.
2. Αν (w == nill OR κλειδί του u > κλειδί του w) ο αλγόριθμος τερματίζει;
3. Ανταλλάσσουμε τα δεδομένα του u με τα δεδομένα του w;
4. Θέτω (u = w) και (w = θυγατρικός κόμβος του u με τη μεγαλύτερη προτεραιότητα);
5. Επιστρέφουμε στο βήμα 2;

Ποια είναι η χρονική πολυπλοκότητα της Insert(); O(ύψος δένδρου)

# Μερικώς Διατεταγμένα Δένδρα Υλοποίηση ως Πλήρη Δυαδικά Δένδρα




- Υλοποιούμε το μερικώς διατεταγμένο δένδρο χρησιμοποιώντας πλήρη δυαδικά δένδρα.
- Το δεξιότερο φύλλο με μέγιστο βάθος στο δένδρο είναι το τελευταίο στοιχείο του πίνακα.
- Η θέση του πίνακα που περιέχει αυτό τον κόμβο μπορεί να καθοριστεί αν γνωρίζουμε το πλήθος των στοιχείων και τη διεύθυνση του πρώτου στοιχείου του πίνακα.
- Η διαγραφή του φύλλου αυτού δεν επηρεάζει την μερική διάταξη του δένδρου, ενώ το δένδρο εξακολουθεί να είναι πλήρες.

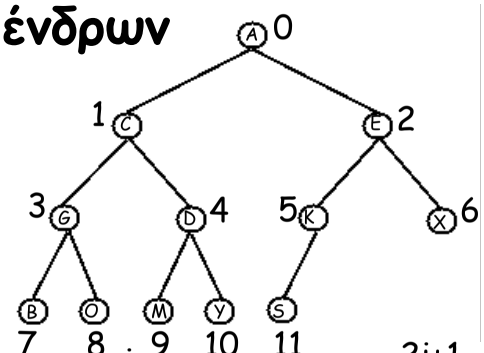
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 8 | 9 | 16 | 12 | 10 | 56 | 18 | 20 | 22 | 44 | |

Ένα μερικώς διατεταγμένο δένδρο υλοποιημένο με στατικό τρόπο (δηλαδή με πίνακα), ονομάζεται **σωρός**.

Το ύψος οποιουδήποτε πλήρους δυαδικού δένδρου είναι $O(\log n) \Rightarrow$ Πολυπλοκότητα HeapInsert() και HeapDeleteMin() = $O(\log n)$. Ο σωρός είναι μια εξαιρετικά αποδοτική δομή για την υλοποίηση ουρών προτεραιότητας.



# Υλοποίηση Πλήρων Δυαδικών Δένδρων




Υπάρχει μόνο ένα πλήρες δυαδικό δένδρο με n κόμβους και το υλοποιούμε με ένα πίνακα Ν στοιχείων.
Αριθμούμε τους κόμβους με αριθμούς στο διάστημα {0,...,n-1} και αποθηκεύουμε τον κόμβο i στο στοιχείο Τ[i] του πίνακα.
Θέλουμε να κάνουμε την αρίθμηση με τέτοιο τρόπο ώστε να πετύχουμε την εκτέλεση χρήσιμων λειτουργιών στο δένδρο σε σταθερό χρόνο.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| A | C | E | G | D | K | X | B | O | M | Y | s |

**Αρίθμηση**
Η ρίζα είναι ο κόμβος 0.
Το αριστερό παιδί του κόμβου i αριθμείται ως κόμβος 2i+1, ενώ το δεξί παιδί του ως κόμβος 2i+2.

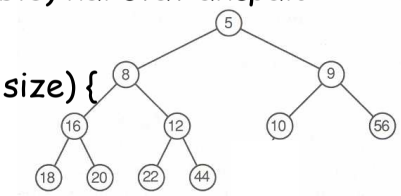
**Υλοποίηση Λειτουργιών**

- IsLeaf(i): return (2i+1>n);
- LeftChild(i): if (2i+1 < n); return (2i+1) else return nill;
- RightChild(i): if (2i+2<n) return(2i+2); else return nill;
- LeftSibling(i): if (i != 0 and i not odd) return (i-1);
- RightSibling(i): if (i != n-1 and i not even) return(i+1);
- Parent(i): if (i != 0) return($\lfloor (i-1)/2 \rfloor$);

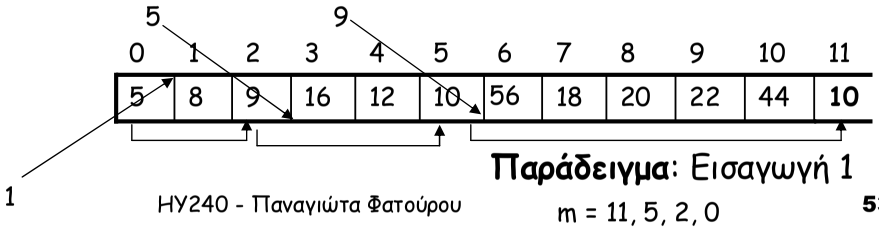
**Χρονική πολυπλοκότητα κάθε λειτουργίας:** $\Theta(1)$

# Σωροί – Ψευδοκώδικας για HeapInsert()

- Τα στοιχεία του σωρού είναι structs με δύο πεδία, ένα κλειδί K τύπου Key και τα δεδομένα data τύπου Type που έχουν συσχετισθεί με το κλειδί.
- Ο σωρός υλοποιείται από έναν πίνακα A (τύπου HeapTable) και έναν ακέραιο size που υποδηλώνει το πλήθος των στοιχείων του σωρού.




```
procedure HeapInsert(Key K, Type I, HeapTable A, int size) {
/* Εισαγωγή του ζεύγους <K,I> στο σωρό <A,size> */
   if (size == N) then error;   /* Heap is full */
   m = size;   // ο m είναι ακέραιος που χρησιμοποιείται ως δείκτης στους κόμβους ενός
               // μονοπατιού του δένδρου. Αρχικά, δείχνει στη θέση που θα εισαχθεί το νέο στοιχείο
   while (m > 0 and K < A[⌊(m-1)/2⌋]->Key) {   // όσο δεν έχω φθάσει στη ρίζα και
                                               // ο πατρικός κόμβος του m έχει μικρότερο κλειδί από το K
      A[m]->key  = A[⌊(m-1)/2⌋]->Key;   // το κλειδί και τα δεδομένα του πατρικού κόμβου
      A[m]->data = A[⌊(m-1)/2⌋]->data;  // του m αντιγράφονται στον m
      m = ⌊(m-1)/2⌋;                   // η διαδικασία επαναλαμβάνεται με m = πατρικός κόμβος του m
   }
   A[m]->Key = K;
   A[m]->data = I;
   size++;
}
```



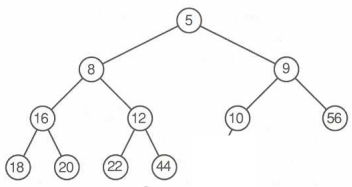

**Παράδειγμα**: Εισαγωγή 1

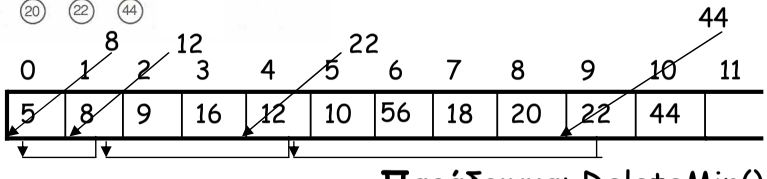
m = 11, 5, 2, 0



# Σωροί – Ψευδοκώδικας για HeapDeleteMin()

```
Type HeapDeleteMin(HeapTable A, int size) {
// διαγραφή του στοιχείου με τη μεγαλύτερη προτεραιότητα και επιστροφή της τιμής του
   if (size == 0) then error;          // Κενός σωρός
   I = A[0]->data;                     // Στοιχείο που θα επιστραφεί
   K = A[size-1]->Key;                 // τιμή κλειδιού που θα μετακινηθεί προς τη ρίζα
   size--;                             // νέο μέγεθος σωρού
   if (size == 1) then return;         // ο σωρός περιέχει μόνο τη ρίζα μετά τη διαγραφή
   m = 0;          /* ο m είναι ακέραιος που δεικτοδοτεί τους κόμβους ενός μονοπατιού προς τη ρίζα
    while ((2m+1 < n AND K > A[2m+1]->Key) OR  // αν υπάρχει αριστερό παιδί και έχει μικρότερο κλειδί
           (2m+2 < n  AND K > A[2m+2]->Key)) {  // ή αν υπάρχει δεξιό παιδί και έχει μικρότερο κλειδί
       if (2m +2 < n) {                        // ο m έχει δύο θυγατρικούς κόμβους
          if (A[2m+1]->Key < A[2m+2]->Key)
                p = 2m+1;                                  ) m με τη μεγαλύτερη προτεραιότητα
           else p = 2m+2;
       }
       else p = n-1;
       A[m]->Key = A[p]->Key;
       A[m]->data = A[p]->data;
       m = p;
   }
   A[m]->Key = A[n-1]->Key;
   A[m]->data = A[n-1]->data;
   return I;
}
```




**Παράδειγμα**: DeleteMin()

m = 0, 1, 4, 9