Πίνακες Κατακερματισμού (Hash Tables)

Ορέστης Τελέλης

telelis@unipi.gr

Τμήμα Ψηφιακών Συστημάτων, Πανεπιστήμιο Πειραιώς

Πίνακες (Μια παλιά άσκηση)

Σε πίνακα A n θέσεων, χρόνοι χειρότερης περίπτωσης:

- **Εισαγωγή:** $O(1)$ (στην πρώτη ελεύθερη θέση στο τέλος του πίνακα).
- **Αναζήτηση:** $O(n)$ (διατρέχουμε τον πίνακα γραμμικά).
- **Διαγραφή:** $O(n)$ (προϋποθέτει αναζήτηση + μετακίνηση στοιχείων).

Ερώτημα: Μπορούμε καλύτερα αν θέλουμε να αποθηκεύσουμε υποσύνολο του $\{0, 1, \ldots, n-1\}$;

Πίνακες (Μια παλιά άσκηση)

Απάντηση: Ναι!
Χρησιμοποιούμε έναν πίνακα n δυαδικών στοιχείων (με τιμές `true/false`)

- Εισαγωγή του $a \in \{0, 1, \ldots, n-1\}$: $A[a] = \mathit{true}$ (χρόνος $O(1)$).
- Διαγραφή του $a \in \{0, 1, \ldots, n-1\}$: $A[a] = \mathit{false}$ (χρόνος $O(1)$).
- Αναζήτηση του $a \in \{0, 1, \ldots, n-1\}$: `return A[a];` (χρόνος $O(1)$).

Ερώτημα: Μπορούμε να επιτύχουμε το ίδιο για οποιοδήποτε υποσύνολο οποιουδήποτε συνόλου δεδομένων n στοιχείων;

Εισαγωγή

Πίνακας Κατακερματισμού:

- Προσφέρει πολύ γρήγορη **Εισαγωγή** και **Αναζήτηση**.
- **Αναζήτηση**, **Εισαγωγή** (μερικές φορές και η **Διαγραφή**) *σε σχεδόν σταθερό χρόνο* $O(1)$.
- Σημαντικά πιο ταχύς από τα δέντρα (που αποδίδουν σε χρόνο $O(\log n)$).

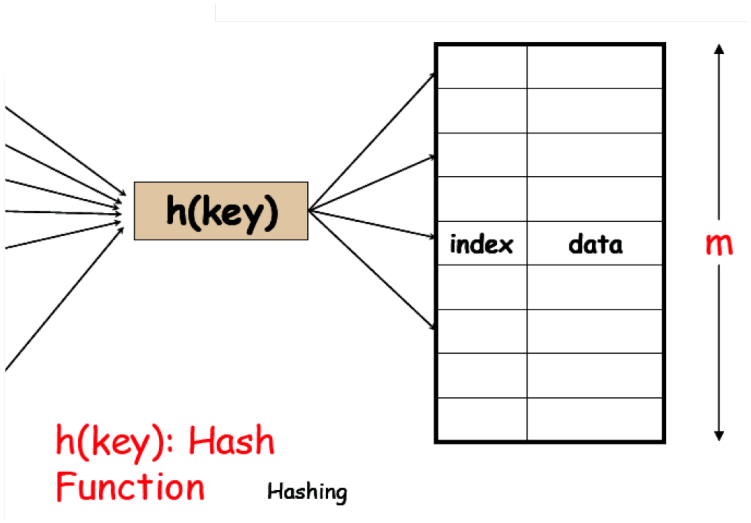
Μειονεκτήματα

- Βασίζονται σε πίνακες: δύσκολη η επέκταση μετά την δημιουργία τους.
- Η απόδοσή τους μειώνεται σημαντικά όταν ο πίνακας υπερκορεστεί.
- Δεν υπάρχει τρόπος επίσκεψης στοιχείων με συγκεκριμένη σειρά.

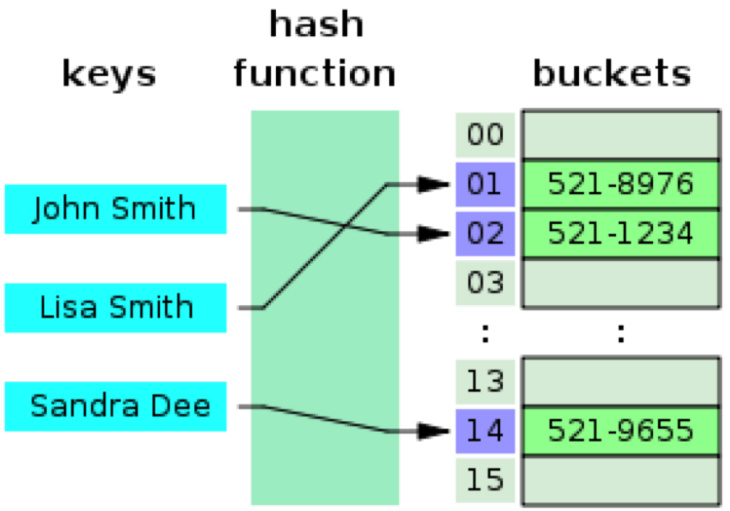
Πίνακες/Συναρτήσεις Κατακερματισμού

- Βασικό στοιχείο: ο μετασχηματισμός κλειδιών σε δείκτες πίνακα.
- Επιτυγχάνεται με μία **συνάρτηση κατακερματισμού.**
- Απλή περίπτωση, χωρίς χρήση συνάρτησης κατακερματισμού.
- Τιμές κλειδιών $\Rightarrow$ δείκτες πίνακα

Τρόπος Λειτουργίας



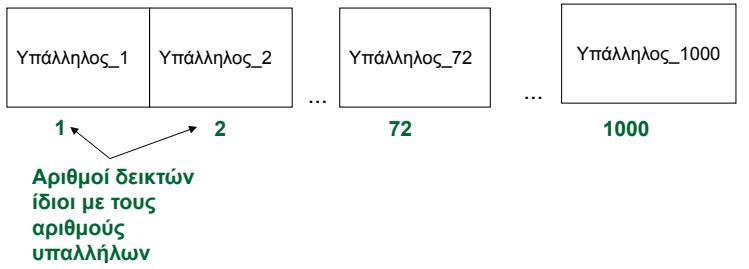
Παράδειγμα Τρόπου Λειτουργίας: Τηλεφωνικός Κατάλογος



Παραδείγματα Χρήσης

- Πρόσβαση σε εγγραφές υπαλλήλων.
- Αριθμοί υπαλλήλων $\Rightarrow$ 1-1000.
- Οι αριθμοί υπαλλήλων κλειδιά στην προσπέλαση εγγραφών.
- Τί είδους δομή πρέπει να χρησιμοποιηθεί σ' αυτή την περίπτωση;

Παραδείγματα Χρήσης (Συνέχεια)



- Προσπέλαση στοιχείου πίνακα:

```
empRecord rec = databaseArray[72];
```

- Προσθήκη στοιχείου:

```
databaseArray[totalEmployees++] = newRecord;
```


Παραδείγματα Χρήσης (Συνέχεια)

- Χρήση πίνακα: απλή & αποδοτική δομή για την προσπέλαση των στοιχείων.
- Περιορισμοί:
  - Καλά τακτοποιημένα κλειδιά.
  - Περιορισμένος αριθμός διαγραφών – οδηγούν σε κενά στην μνήμη.
  - Στοιχεία μπορούν να προστίθενται μόνο στο τέλος του πίνακα.
  - Μέγεθος πίνακα σταθερό.
- Τί κάνουμε όταν τα κλειδιά δεν πληρούν τους περιορισμούς του πίνακα;
- Ο πίνακας κατακερματισμού είναι μία κατάλληλη δομή δεδομένων.

Εφαρμογές Πινάκων Κατακερματισμού

- Μεταγλωττιστές γλωσσών προγραμματισμού.
- Διατήρηση πίνακα συμβόλων σε πίνακα κατακερματισμού (ονόματα μεταβλητών/συναρτήσεων, διευθύνσεις στη μνήμη).
- Λεξικό όρων.
  - Κάθε λέξη καταλαμβάνει ένα κελί σε έναν πίνακα.
  - Προσπέλαση της λέξης χρησιμοποιώντας έναν αριθμό δείκτη.

Συνάρτηση Κατακερματισμού

- Μετατρέπει το αντικείμενο σε ακέραιο για δεικτοδότηση θέσης πίνακα:

$$h : O \mapsto [0, M)$$

όπου είναι αποθηκευμένο (ή θα αποθηκευθεί) το αντικείμενο.

- Αν είναι 1-1, προσπελαύνουμε το αντικείμενο με το δείκτη του στον πίνακα.

Ιδιότητες Συνάρτησης Κατακερματισμού

Επιθυμητές Ιδιότητες

1. Κατανομή κλειδιών ομοιόμορφα στο χώρο δεικτών του πίνακα.
2. *Εγγραφές διαφορετικών κλειδιών να μην καταλαμβάνουν την ίδια θέση:*

$$\mathbf{k} \neq \mathbf{k}', \implies \mathbf{h}(\mathbf{k}) \neq \mathbf{h}(\mathbf{k}')$$

3. *Διατήρηση της σειράς:* $\mathbf{k} \leq \mathbf{k}' \implies \mathbf{h}(\mathbf{k}) \leq \mathbf{h}(\mathbf{k}')$.

- Δύσκολο να βρεθεί μία τέτοια συνάρτηση κατακερματισμού.
- Η 2η ιδιότητα είναι η πιο σημαντική !!
- **Σύγκρουση:** Δύο ή περισσότερα διαφορετικά αντικείμενα/κλειδιά αντιστοιχίζονται μέσω της συνάρτησης στην ίδια θέση:

$$\mathbf{k} \neq \mathbf{k}' \implies \mathbf{h}(\mathbf{k}) = \mathbf{h}(\mathbf{k}')$$

Πρωτογενείς Τύποι Δεδομένων

- Μπορούμε να αντιστοιχίσουμε κάθε τιμή βασικού τύπου σε τιμή τύπου `int`.
- Τιμή τύπου `char`: κατακερματίζεται σε μοναδικό θετικό ακέραιο (όπως ορίζεται στο UNICODE).
- Αφηρημένοι τύποι δεδομένων δεν έχουν προκαθορισμένες τιμές.
- Υπολογισμός τιμής κατακερματισμού για κάθε αντικείμενο με πολυωνυμική συνάρτηση.

Συνάρτηση Κατακερματισμού

- Έστω ο αριθμός 1234: είναι μία συλλογή από ψηφία 1, 2, 3 και 4, όπου:

$$1 \cdot 10^3 + 2 \cdot 10^2 + 3 \cdot 10^1 + 4 \cdot 10^0$$

- Έστω το string "junk" – συλλογή από χαρακτήρες 'j', 'u', 'n', 'k'.
ASCII αναπαράσταση με 7 bits σαν ένας αριθμός μεταξύ 0 και 127:

$$128^3 \cdot \text{'j'} + 128^2 \cdot \text{'u'} + 128^1 \text{'n'} + 128^0 \cdot \text{'k'}$$

- Για αντιστοίχιση μεγάλων αριθμών (όπως προκύπτουν από μετατροπή strings σε αριθμούς) χρειαζόμαστε συνάρτηση κατακερματισμού.
- Αν **tableSize** το μέγεθος του πίνακα και x ακέραιος:

$$x \bmod \textbf{tableSize}$$

παράγει αριθμό στο διάστημα $[0, \textbf{tableSize} - 1]$.

Κατακερματισμός Αντικειμένων

- Έστω αντικείμενα της κλάσης **C** που έχουν k πεδία τύπου `int`:

$$O = (x_0, \ldots, x_{k-1})$$

- Η τιμή κατακερματισμού για κάθε αντικείμενο O επιλέγοντας μία θετική σταθερά α, υπολογίζεται ως:

$$h(x_0)\alpha^{k-1} + h(x_1)\alpha^{k-2} + \cdots + h(x_{k-2})\alpha + h(x_{k-1})$$

ή: $h(x_{k-1}) + \alpha(h(x_{k-2}) + \alpha(h(x_{k-2}) + \cdots + \alpha(h(x_1) + \alpha h(x_0))\ldots))$

Διαχείριση Συγκρούσεων

- Έχοντας μία συνάρτηση κατακερματισμού, τι θα κάνουμε σε περίπτωση σύγκρουσης (collision);
- Εάν το στοιχείο X κατακερματιστεί σε θέση που είναι ήδη κατειλημμένη, πού θα το τοποθετήσουμε;

Ανοιχτή Διευθυνσιοδότηση

Open Addressing

- Όταν ένα στοιχείο δεν μπορεί να τοποθετηθεί στο δείκτη που υπολογίστηκε από τη συνάρτηση κατακερματισμού, το πρόγραμμα ψάχνει για άλλη θέση στον πίνακα.

  Εκδοχές Ανοικτής Διεύθυνσιοδότησης (Open Addressing):

  - Γραμμική Διερεύνηση (Linear Probing).
  - Δευτεροβάθμια Διερεύνηση (Quadratic Probing).
  - Διπλός Κατακερματισμός (Double Hashing)

Γενική Μορφή Ανοιχτής Διευθυνσιοδότησης

Σύγκρουση: κλειδιά $k \neq k'$ με $h(k) = h(k')$ και η θέση $h(k)$ έχει δοθεί στο k.

- Δεδομένης $F : \{0, \ldots, \texttt{tableSize-1}\} \mapsto \mathbb{N}$, αναζητείται διαθέσιμη θέση:

$$(h(k') + F(i)) \bmod \texttt{tableSize}, \quad \text{για } i = 1, 2, 3, \ldots$$

- **Τελικά, κάθε εισαγωγή επιχειρείται στην πρώτη διαθέσιμη θέση:**

$$(h(k') + F(i)) \bmod \texttt{tableSize}, \quad i = 0, 1, 2, \ldots, \text{όπου } F(0) = 0$$

- **Η συνάρτηση F καλείται Στρατηγική Επίλυσης Σύγκρουσης** (*Collision Resolution Strategy*).

Ανοιχτή Διεύθυνση: Γραμμική Διερεύνηση

- **΄Οταν η συνάρτηση επίλυσης σύγκρουσης, F, είναι γραμμική.** Τυπικά:

$$\mathbf{F(i)} = \mathbf{i}$$

- Τότε, αναζητείται σειριακά ελεύθερη θέση στον πίνακα, μεταξύ των:

$$(\mathbf{h(k')} + \mathbf{i}) \bmod \texttt{tableSize}, \quad i = 0, 1, 2, \ldots$$

Παράδειγμα Γραμμικής Διερεύνησης

1. $h(89) = 89 \bmod 10 = 9$
2. $h(18) = 18 \bmod 10 = 8$
3. $h(49) = 49 \bmod 10 = 9$
4. $h(58) = 58 \bmod 10 = 8$
5. $h(9) = 9 \bmod 10 = 9$

Θέση	Μετά το 89	Μετά το 18	Μετά το 49	Μετά το 58	Μετά το 9
0			49	49	49
1				58	58
2					9
3					
4					
5					
6					
7					
8		18	18	18	18
9	89	89	89	89	89

Ανοιχτή Διεύθυνση: Γραμμική Διερεύνησης

Ένα αντικείμενο στον πίνακα κατακερματισμού «συνδέει» και άλλα αντικείμενα (μέσω της Γραμμικής Διερεύνησης).

Χρόνος Βασικών Πράξεων:

- **Αναζήτηση:** ανάλογος του μέγιστου «μήκους» γραμμικής διερεύνησης.
- **Εισαγωγή:** ανάλογος του μέγιστου «μήκους» γραμμικής διερεύνησης.
- **Διαγραφή:** σταθερός $O(1)$, **γίνεται εικονικά** (δεν ελευθερώνεται θέση):
  - αν ελευθερωθεί θέση, η αναζήτηση στον πίνακα μπορεί να αποτύχει.

Κλάση `DataItem`

```
import java.io.*;

class DataItem {                        // (could have more data)
   private int iData;                   // data item (key)

   public DataItem(int ii) {            // constructor
        iData = ii;
   }

   public int getKey() {
        return iData;
   }

}                                       // end class DataItem
```

Υλοποίηση Πίνακα Κατακερματισμού (1/4)

```
class HashTable {
   private DataItem[] hashArray;       // array holds hash table
   private int arraySize;
   private DataItem nonItem;           // for deleted items
// -------------------------------------------------------------
   public HashTable(int size) {        // constructor
      arraySize = size;
      hashArray = new DataItem[arraySize];
      nonItem = new DataItem(-1);      // deleted item key is -1
   }
// -------------------------------------------------------------
   public void displayTable() {
      System.out.print("Table: ");
      for(int j=0; j<arraySize; j++) {
         if(hashArray[j] != null)
            System.out.print(hashArray[j].getKey() + " ");
         else System.out.print("** ");
      }
      System.out.println("");
   }
```


Υλοποίηση Πίνακα Κατακερματισμού (2/4)

```
   public int hashFunc(int key) {
      return key % arraySize;          // hash function
   }
// -------------------------------------------------------------

   public void insert(DataItem item) { // insert a DataItem
                                       // (assumes not full)
      int key = item.getKey();         // extract key
      int hashVal = hashFunc(key);     // hash the key

                                       // until empty or -1,
      while(hashArray[hashVal] != null &&
                     hashArray[hashVal].getKey() != -1) {

         ++hashVal;                    // go to next cell
         hashVal %= arraySize;         // wraparound if needed
      }

      hashArray[hashVal] = item;       // insert item
   }                                   // end insert()
```


Υλοποίηση Πίνακα Κατακερματισμού (3/4)

```
  public DataItem delete(int key)  {       // delete a DataItem
      int hashVal = hashFunc(key);         // hash the key

      while(hashArray[hashVal] != null) {  // until empty cell,

         if(hashArray[hashVal].getKey()==key) {  // if found

            DataItem temp = hashArray[hashVal];  // save item
            hashArray[hashVal] = nonItem;        // delete item
            return temp;                         // return item
         }

         ++hashVal;                    // go to next cell
         hashVal %= arraySize;         // wraparound if needed
      }
      return null;                     // can't find item
   }                                   // end delete()
```


Υλοποίηση Πίνακα Κατακερματισμού (4/4)

```
   public DataItem find(int key) {        // find item with key

      int hashVal = hashFunc(key);        // hash the key

      while(hashArray[hashVal] != null) {// until empty cell,

                                          // found the key?
         if(hashArray[hashVal].getKey() == key)
            return hashArray[hashVal];    // yes, return item

         ++hashVal;                    // go to next cell
         hashVal %= arraySize;         // wraparound if needed
      }
      return null;                     // can't find item
   }

}                                      // end class HashTable
```

Γραμμική Διερεύνηση: Δημιουργία Δεσμών

- **Μειονέκτημα Γραμμικής Διερεύνησης:** *Δημιουργία δεσμών (clusters).*
- Οι δεσμοί δημιουργούν μεγάλες ακολουθίες διερεύνησης.
- Αργή προσπέλαση κελιών στο τέλος της ακολουθίας.
- Το πρόβλημα δεσμών μεγαλώνει καθώς γεμίζει ο πίνακας. (primary clustering)
- Όχι σημαντικό όσο ο πίνακας είναι κατά το πολύ κατά το ήμισυ γεμάτος.
- **Συντελεστής Φόρτου (Load Factor):**

$$\textbf{loadFactor} = \textbf{nItems} \,/\, \textbf{tableSize}$$

Επέκταση Πίνακα – Ανακατακερματισμός

Rehashing

- *Όταν ο πίνακας «παρα-γεμίζει» τον επεκτείνουμε.*
- **Δημιουργία μεγαλύτερου πίνακα και επανεισαγωγή των στοιχείων** με `insert`.
  Προσοχή: *δεν αντιγράφουμε «απλώς» τα στοιχεία στις ίδιες θέσεις*
- *Συνήθως επιλέγεται διπλάσιο μέγεθος πίνακα.* (όχι ακριβώς...)
- **Η διαδικασία λέγεται «ανακατακερματισμός» (rehashing).**
- **Όμως το μέγεθος απαιτείται να είναι πρώτος αριθμός (γιατί;)**

```
private boolean isPrime(int n) {
  for(int j = 2; (j*j <= n); j++)
    if (n%j == 0) return(false);
  return(true);
}
```


Ανοιχτή Διεύθυνση: Δευτεροβάθμια Διερεύνηση

- *Αποσκοπεί στην αντιμετώπιση Πρωτογενούς Δημιουργίας Δεσμών* (*της Γραμμικής Διρεύνησης.*)
- Εξετάζει συγκεκριμένα κελιά μακριά από το αρχικό σημείο σύγκρουσης.
- **Δευτεροβάθμια Συνάρτηση Επίλυσης Συγκρούσεων**, τυπικά:

$$\mathbf{F(i)} = \mathbf{i^2}$$

- Για κάθε εισαγωγή κλειδιού, *k*, αναζητά ελεύθερη θέση στον πίνακα, εξετάζοντας σειριακά τις ακόλουθες θέσεις:

$$\left(\mathbf{h(k)} + \mathbf{i^2}\right) \bmod \texttt{tableSize}, \quad \mathbf{i} = \mathbf{0}, \mathbf{1}, \mathbf{2}, \ldots$$

Παράδειγμα Δευτεροβάθμιας Διερεύνησης

1. $h(89) = 89 \mod 10 = 9$
2. $h(18) = 18 \mod 10 = 8$
3. $h(49) = 49 \mod 10 = 9$
4. $h(58) = 58 \mod 10 = 8$
5. $h(9) = 9 \mod 10 = 9$

Θέση	Μετά το 89	Μετά το 18	Μετά το 49	Μετά το 58	Μετά το 9
0			49	49	49
1					
2				58	58
3					9
4					
5					
6					
7					
8		18	18	18	18
9	89	89	89	89	89

Απόδοση Δευτεροβάθμιας Διερεύνησης

Θεώρημα

Όταν χρησιμοποιείται δευτεροβάθμια διερεύνηση και το μέγεθος του πίνακα είναι πρώτος αριθμός, πάντα μπορεί να εισαχθεί ένα νέο κλειδί, αν ο πίνακας είναι το πολύ κατά το ήμισυ γεμάτος (δηλαδή αν **loadFactor** $\leq$ **0.5**).

Απόδειξη (1/2)

- Έστω **tableSize** το μέγεθος του πίνακα: θεωρούμε ότι είναι **περιττός πρώτος αριθμός**, αυστηρά μεγαλύτερος του 3.
- Θα αποδείξουμε ότι οι πρώτες $\lfloor$**tableSize**$/2\rfloor$ εναλλακτικές θέσεις του πίνακα είναι όλες διαφορετικές.
- Για κάποια i, j με $0 < i, j \leq \lfloor$**tableSize**$/2\rfloor$, δύο τέτοιες θέσεις είναι οι:

$$h(k) + i^2 \bmod \textbf{tableSize}$$
$$h(k) + j^2 \bmod \textbf{tableSize}$$

- Θα θεωρήσουμε (με στόχο την απαγωγή σε άτοπο), ότι $i \neq j$, αλλά:

$$h(k) + i^2 \bmod \textbf{tableSize} = h(k) + j^2 \bmod \textbf{tableSize}$$

Απόδειξη (2/2)

- Τότε:
$$[(h(k) + i^2) - (h(k) + j^2)] \bmod \textbf{tableSize} = 0$$
- Άρα:
$$(i^2 - j^2) \bmod \textbf{tableSize} = 0$$
- Επομένως:
$$(i - j)(i + j) \bmod \textbf{tableSize} = 0$$
- Όμως, ο **tableSize** δε μπορεί να διαιρεί τον $i + j$, διότι $0 < i + j <$ **tableSize** $- 1$.
- Επίσης, ο **tableSize** δε μπορεί να διαιρεί τον $i - j$, εκτός κι αν $i = j$, διότι $|i - j| <$ **tableSize**$/2$. Όμως, επίσης, $i \neq j$, από υπόθεση.
- Τέλος, ο **tableSize** δε μπορεί να διαιρεί το γινόμενο, γιατί είναι πρώτος (και θα έπρεπε να διαιρεί κάποιον από τους $i + j$, $i - j$).
- Επομένως, οι πρώτες $\lfloor$**tableSize**$/2\rfloor$ εναλλακτικές θέσεις του πίνακα είναι διαφορετικές. Αν συμπεριλάβουμε και την πρώτη θέση στην οποία κατακερματίζεται ένα κλειδί, έχουμε $\lceil$**tableSize**$/2\rceil$ διαφορετικές εναλλακτικές θέσεις.

Πρόβλημα με Δευτεροβάθμια Διερεύνηση

- Οδηγεί σε **Δευτερεύουσα Δημιουργία Δεσμών** (*Secondary Clustering*)
- Ο αλγόριθμος που παράγει την ακολουθία βημάτων στη δευτεροβάθμια διερεύνηση παράγει πάντα τα ίδια βήματα: 1, 4, 9, 16
- Π.χ. 184, 302, 420, 544 μετασχηματίζονται στο 7 και εισάγονται στον πίνακα με αυτήν τη σειρά
  - 302 $\Rightarrow$ διερεύνηση ενός βήματος.
  - 420 $\Rightarrow$ διερεύνηση 4 βημάτων.
  - 544 $\Rightarrow$ διερευνηση 9 βημάτων.
- Κάθε επιπλέον στοιχείο με κλειδί που μετασχηματίζεται σε 7 θα χρειάζεται μία ακόμα μακρύτερη διερεύνηση.

Ανοιχτή Διεύθυνση: Διπλός Κατακερματισμός (1)

- **Λύση στο πρόβλημα της δημιουργίας δεσμών.**
- Παραγωγή σειράς «διερευνήσεων» που εξαρτώνται από το κλειδί:
  - αντί η ίδια σειρά για κάθε κλειδί.
- *Μέγεθος βήματος σταθερό σε όλη τη διερεύνηση για δεδομένο κλειδί,*
  - **αλλά είναι διαφορετικό για διαφορετικά κλειδιά.**
- **Δευτερεύουσα Συνάρτηση Κατακερματισμού:**
  (σαν μέρος της Στρατηγικής Επίλυσης Συγκρούσεων)

$$\mathbf{F(i,k) = i \cdot h_2(k)}$$

- Αναζητείται σειριακά η επόμενη ελεύθερη θέση στον πίνακα, μεταξύ των:

$$\big(\mathbf{h(k) + F(i,k)}\big) \bmod \texttt{tableSize}, \quad \mathbf{i = 0, 1, 2, \ldots}$$

Ανοιχτή Διεύθυνση: Διπλός Κατακερματισμός (2)

Επιθυμητά Χαρακτηριστικά Δευτερεύουσας Συνάρτησης Κατακερματισμού:

- *Πρέπει να διαφέρει από την πρωτεύουσα συνάρτηση κατακερματισμού.*
- *Δεν πρέπει ποτέ να έχει έξοδο 0*
  (αλλιώς ατέρμονος βρόχος από συνεχόμενες διερευνήσεις ίδιου κελιού).
- Έχει δειχθεί ότι συναρτήσεις της παρακάτω μορφής αποδίδουν καλά:

$$\mathbf{h_2(key) = constant - (key \bmod constant)}$$

όπου **constant** πρώτος αριθμός και μικρότερος από το μέγεθος πίνακα.

Παράδειγμα Διπλού Κατακερματισμού

- $\mathbf{h_1(k) = k \bmod 10}$.
- $\mathbf{h_2(k) = 7 - (k \bmod 7)}$.

Θέση	Μετά το 89	Μετά το 18	Μετά το 49	Μετά το 58	Μετά το 69
0					69
1					
2					
3				58	58
4					
5					
6			49	49	49
7					
8		18	18	18	18
9	89	89	89	89	89

Υλοποίηση Διπλού Κατακερματισμού

```
   public int hashFunc1(int key) { return key % arraySize; }
// ------------------------------------------------------------

   public int hashFunc2(int key) {
   // non-zero, less than array size, different from hashFunc1
   // array size must be relatively prime to 5, 4, 3, and 2
      return 5 - key % 5;
   }
// ------------------------------------------------------------
  public void insert(int key, DataItem item) {//assume not full
      int hashVal = hashFunc1(key);           // hash the key
      int stepSize = hashFunc2(key);          // get step size
                                        // until empty cell or -1
      while(hashArray[hashVal] != null &&
                    hashArray[hashVal].getKey() != -1) {
         hashVal += stepSize;                 // add the step
         hashVal %= arraySize;                // for wraparound
      }
      hashArray[hashVal] = item;              // insert item
  }                                           // end insert()
```

Το Μέγεθος Πίνακα να είναι Πρώτος Αριθμός

- Έστω πίνακας 15 θέσεων (δείκτες 0 ως 14).
- Έστω κλειδί που κατακερματίζεται στη θέση 0 με βήμα 5.
  - Τότε η ακολουθία διερευνήσεων θα είναι:

    0, 5, 10, 0, 5, 10, . . . **ΠΡΟΒΛΗΜΑ**
  - Εάν το μέγεθος πίνακα ήταν 13 (πρώτος):

    0, 5, 10, 2, 7, 12, 4, 9, 1, 6, 11, 3, . . .

Χωριστή Αλυσίδωση (1)

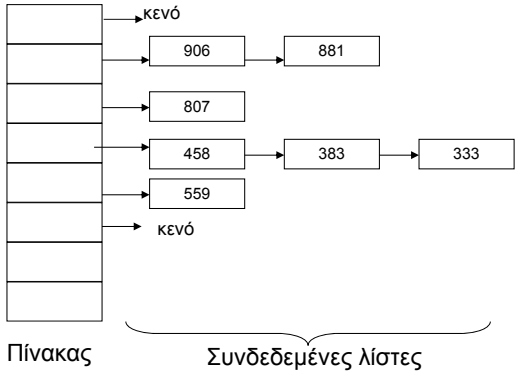
Separate Chaining

- *Συνδεδεμένη Λίστα σε κάθε κελί του πίνακα κατακερματισμού.*
- Το κλειδί ενός στοιχείου μετασχηματίζεται στο δείκτη με βάση τη συνάρτηση κατακερματισμού και το στοιχείο εισάγεται στη συνδεδεμένη λίστα.
- Στοιχεία που κατακερματίζονται στον ίδιο δείκτη απλώς εισάγονται στη λίστα.

Χωριστή Αλυσίδωση (2)

- Μπορούμε να αποθηκεύσουμε οσαδήποτε στοιχεία σε πίνακα n κελιών.
- Ο χρόνος προσπέλασης στοιχείων αυξάνει με το μήκος των λιστών.
- **Επιτρέπονται Διπλοεγγραφές**
- **Διαγραφή:**
  Κατακερματισμός κλειδιού στον πίνακα και διαγραφή στοιχείου από την αντίστοιχη λίστα.

Παράδειγμα Χωριστής Αλυσίδωσης



Υλοποίηση με Χωριστή Αλυσίδωση (1/2)

```
class HashTable {
   private SortedList[] hashArray;              // array of lists
   private int arraySize;
// --------------------------------------------------------------
   public HashTable(int size) {                 // constructor
      arraySize = size;
      hashArray = new SortedList[arraySize];    // create array
      for(int j=0; j<arraySize; j++)            // fill array
         hashArray[j] = new SortedList();       // with lists
   }
// --------------------------------------------------------------
   public void displayTable() {
      for(int j=0; j<arraySize; j++) {          // for each cell,
         System.out.print(j + ". ");            // display cell number
         hashArray[j].displayList();            // display list
      }
   }
```


Υλοποίηση με Χωριστή Αλυσίδωση (2/2)

```
   public int hashFunc(int key) { return key % arraySize; }
// --------------------------------------------------------------
   public void insert(Link theLink) {      // insert a link
      int key = theLink.getKey();
      int hashVal = hashFunc(key);         // hash the key
      hashArray[hashVal].insert(theLink); // insert at hashVal
   }                                       // end insert()
// --------------------------------------------------------------
   public void delete(int key) {           // delete a link
      int hashVal = hashFunc(key);         // hash the key
      hashArray[hashVal].delete(key);      // delete link
   }  // end delete()
// --------------------------------------------------------------
   public Link find(int key) {             // find link
      int hashVal = hashFunc(key);         // hash the key
      Link theLink = hashArray[hashVal].find(key); // get link
      return theLink;                      // return link
   }
}                                          // end class HashTable
```


Καλές Επιλογές για Συναρτήσεις Κατακερματισμού

Η συνάρτηση κατακερματισμού πρέπει να:

- Υπολογίζεται γρήγορα.
- Κατανέμει με ομοιόμορφο τυχαίο τρόπο τα κλειδιά στις θέσεις του πίνακα.
- Μη χρησιμοποιεί άχρηστη πληροφορία.
- Π.χ. Κωδικοί της μορφής 033-400-03-94-05-0-535.
- Χρησιμοποιεί όλα τα δεδομένα.
- Αν είναι η mod (%), πρώτο αριθμό για το διαιρέτη.

Απόδοση Κατακερματισμού

- Η **Εισαγωγή** και η **Αναζήτηση** προσεγγίζουν χρόνο $O(1)$ (αν δε συμβεί καμία σύγκρουση).
- Αν συμβούν συγκρούσεις, ο χρόνος εξαρτάται από το μήκος της ακολουθίας διερευνήσεων.
- Το μέσο μήκος διερεύνησης εξαρτάται από το συντελεστή φόρτου.
- Έστω P το μήκος διερεύνησης (probes) και L ο συντελεστής φόρτου.

Απόδοση Γραμμικής Διερεύνησης

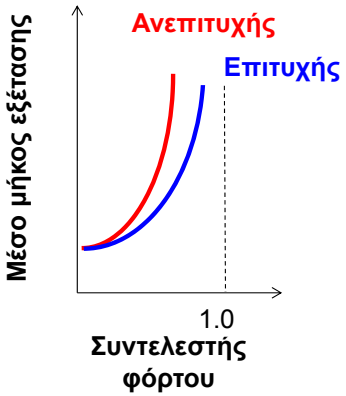
- **Επιτυχής Αναζήτηση:**

  $P = \left(1 + 1/(1-L)^2\right)/2$

- **Ανεπιτυχής Αναζήτηση:**

  $P = \left(1 + 1/(1-L)\right)/2$

- Ο συντελεστής φόρτου πρέπει να διατηρείται κάτω από το 2/3 και, ιδανικά, κάτω από το 1/2.



Donald E. Knuth: "The Art of Computer Programming", Τόμος 3, ``Sorting and Searching''

Απόδοση Δευτεροβάθμιας Διερεύνησης και Διπλού Κατακερματισμού

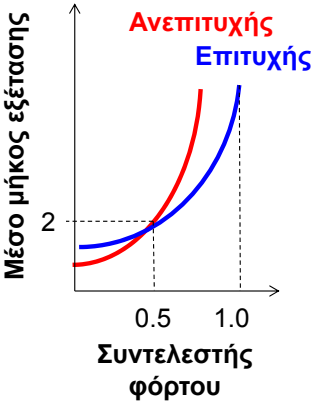
- **Επιτυχής Αναζήτηση:**

  $P = -\log(1-L)/L$

- **Ανεπιτυχής Αναζήτηση:**

  $P = 1/(1-L)$

- Με συντελεστή φόρτου 0.5 επιτυχείς και ανεπιτυχείς αναζητήσεις απαιτούν κατά μέσο όρο 2 διερευνήσεις.

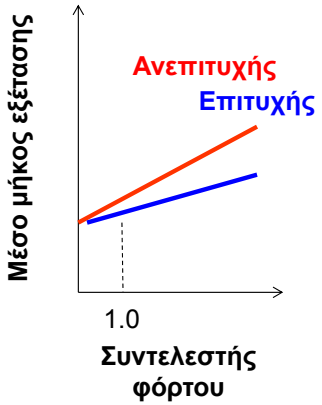


Απόδοση Χωριστής Αλυσίδωσης

- *n = πλήθος εισηγμένων στοιχείων.*
- *Μέσο μήκος κάθε λίστας:*

  AVGListLength = $n/$`tableSize`
- *Ίδιο με το συντελεστή φόρτου:*

  $L = n/$`tableSize`
- **Μέσος Χρόνος Αναζήτησης:**
  - *Επιτυχής:* $1 + L/2$.
  - *Ανεπιτυχής:* $1 + L$.
- **Μέσος Χρόνος Εισαγωγής:**
  - Μη ταξινομημένες λίστες: $O(1)$
  - Ταξινομημένες λίστες: $1 + L/2$.



Συμπερασματικά

- Διπλός Κατακερματισμός μάλλον καλύτερος τύπος ανοιχτής διεύθυνσης.
  - Με μικρή υπεροχή ως προς τη δευτεροβάθμια διερεύνηση.
  - Αν υπάρχει άφθονη μνήμη, πιο απλή η Γραμμική Διερεύνηση (μικρή επιβάρυνση για συντελεστή φόρτου μικρότερο από 0.5).
- Για άγνωστο πλήθος στοιχείων: προτιμότερη η Χωριστή Αλυσίδωση.
- Η απόδοσή της μειώνεται γραμμικά με την αύξηση του συντελεστή φόρτου
- Προτιμάται για δεδομένα αβέβαιου πλήθους και ποιότητας (εύρους κλειδιών).

Εξωτερικός Κατακερματισμός: (External Hashing)

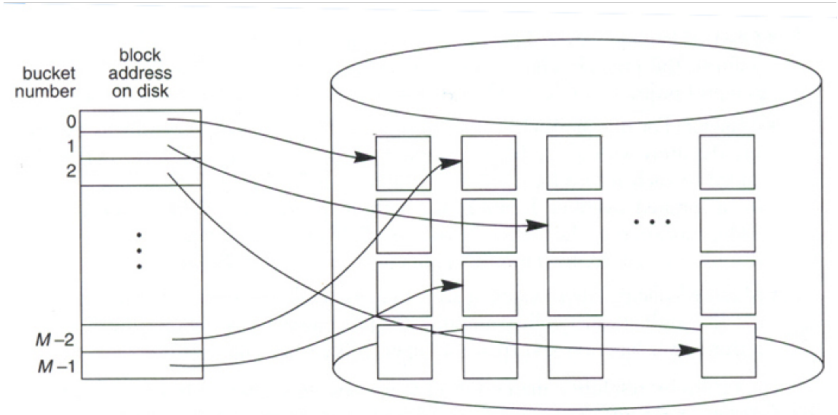


Figure 4.9 Matching bucket numbers to disk blocks.

Πίνακες Κατακερματισμού στη Java

Κλάση HashMap

Παρέχεται από το πακέτο `java.util`

Απαιτείται: `import java.util.HashMap;`

Constructor	*Κατασκευάζει:*
`HashMap()`	Άδειο πίνακα 16 θέσεων με μέγιστο συντελεστή φόρτου 0.75.
`HashMap(int size)`	Άδειο πίνακα `size` θέσεων με μέγιστο συντελεστή φόρτου 0.75.
`HashMap(int size, float lF)`	Άδειο πίνακα `size` θέσεων με μέγιστο συντελεστή φόρτου `lF`.
`HashMap(HashMap m)`	Πίνακα με χαρακτηριστικά και τα στοιχεία του `m`.

Δημιουργία Πίνακα Κατακερματισμού στη Java

Η κλάση HashMap έχει οριστεί με χρήση Java «Generics»

Στη δήλωση αναφοράς σε αντικείμενο τύπου HashMap πρέπει να δηλώσουμε:

- Τον τύπο των κλειδιών, **K** (αυτά κατακερματίζονται)
- Τον τύπο των δεδομένων (τιμών), **V**
- Με δήλωση του είδους: `HashMap<K,V> hmap;`
- *Δε μπορούν να είναι βασικοί τύποι (`int`, `float`, `double`, `char`,...)*
  - Υποκαθιστούνται από σύνθετους τύπους `Integer`, `Float`, `Double`... (με κεφαλαίο το πρώτο γράμμα)
- *Επιτρέπονται σύνθετοι τύποι (κλάσεις) που έχουμε ορίσει.*

Παραδείγματα

Πίνακας για κλειδιά τύπου `Integer`, τιμές τύπου `String`:

```
HashMap<Integer,String> hm = new HashMap<Integer,String>();
```

Πίνακας για κλειδιά τύπου `String`, τιμές τύπου `MyClass`:

```
HashMap<String,MyClass> hm = new HashMap<String,MyClass>(20);
```

Μέθοδοι Πινάκων Κατακερματισμού στη Java

Μέθοδος	*Λειτουργία*
`V put(K key, V value)`	Εισάγει το ζεύγος `key,value`
`V remove(Object key)`	Διαγράφει την εγγραφή για κλειδί `key`
`V get(Object key)`	Επιστρέφει την τιμή για κλειδί `key`
`boolean isEmpty()`	Επιστρέφει **true** αν ο πίνακας είναι άδειος
`boolean containsKey(Object key)`	Επιστρέφει **true** αν ο πίνακας περιέχει ζεύγος για το κλειδί `key`
`boolean containsValue(Object value)`	Επιστρέφει **true** αν ο πίνακας περιέχει ζεύγος με την τιμή `value`
`int size()`	Επιστρέφει πλήθος ζευγών

Παράδειγμα

```
import java.util.HashMap;

public class HashMapDemo {

  public static void main(String args[]) {

    // Create a hash map: String for key, Integer for Value
    HashMap<String,Integer> hm = new HashMap<String,Integer>();

    // Put elements to the map
    hm.put("Orestis", new Integer(37));

    // Print Value for key "Orestis"
    System.out.println(hm.get("Orestis"));
  }
}
```


Άσκηση

Δεδομένης εισόδου $\{ 4371,\ 1323,\ 6173,\ 4199,\ 4344,\ 9679,\ 1989 \}$ και:

$$\text{Συνάρτηση Κατακερματισμού:}\quad h(k) = k \bmod 10$$

να βρεθούν οι πίνακες κατακερματισμού:

a. με ξεχωριστή αλυσίδα.

b. με ανοιχτή διευθυνσιοδότηση γραμμικής διερεύνησης.

c. με ανοιχτή διευθυνσιοδότητη δευτεροβάθμιας διερεύνησης.

d. με ανοιχτή διευθυνσιοδότηση δευτερεύουσας συνάρτησης κατακερματισμού:

$$h_1(k) = 7 - (k \bmod 7)$$