# Hashing

# Εισαγωγή

- Λεξικό: μία συλλογή από στοιχεία
- Στοιχείο i: $(k_i, I_i)$ όπου $k_i$ είναι το κλειδί του στοιχείου i και $I_i$ είναι η υπόλοιπη πληροφορία του στοιχείου
- Τα κλειδιά των στοιχείων είναι μοναδικά
- Υποστηρίζονται οι λειτουργίες
  - Εισαγωγή στοιχείου στο λεξικό με συγκεκριμένη τιμή κλειδιού
  - Αναζήτηση του λεξικού για στοιχείο με συγκεκριμένη τιμή κλειδιού
  - Διαγραφή ενός στοιχείου με συγκεκριμένη τιμή κλειδιού
- Έχουμε δει υλοποίηση με βάση τα δυαδικά δέντρα αναζήτησης ή με απλές γραμμικές λίστες στοιχείων
- Hashing: άλλος ένας τρόπος υλοποίησης της δομής του λεξικού που χρησιμοποιείται ευρέως.

# Hashing

- Χρήση ενός μονοδιάστατου πίνακα για τη αποθήκευση των στοιχείων του λεξικού (Hash Table)
- Συνάρτηση κατακερματισμού (hash function): απεικονίζει τα κλειδιά των στοιχείων σε θέσεις (δείκτες) του πίνακα.
- Ιδανική περίπτωση:
  - αν f είναι η συνάρτηση κατακερματισμού, το στοιχείο με κλειδί κ αποθηκεύεται στη θέση f(κ).
  - το κόστος αναζήτησης είναι O(1)
  - Εισαγωγή , Διαγραφή στοιχείου O(1).
  - Θα πρέπει όμως το πλήθος των θέσεων  του πίνακα να είναι τουλάχιστον ίσος με το πλήθος των στοιχείων του λεξικού
  - Διαφορετικά δύο στοιχεία με διαφορετικά κλειδιά απεικονίζονται  στην ίδια θέση πίνακα

# Παράδειγμα

- Αν οι εγγεγραμμένοι φοιτητές ενός τμήματος δεν ξεπερνούν τους 1000 και οι αριθμοί μητρώου αρχίζουν από 100:
  - Πίνακας κατακερματισμού: πίνακας 1000 θέσεων τουλάχιστον
  - Κάθε θέση του πίνακα περιέχει μία εγγραφή φοιτητή
  - Το πεδίο του κλειδιού σε όλες τις εγγραφές του πίνακα αρχικοποιείται στο 0
  - Συνάρτηση κατακερματισμού: f(κ)=κ-100 δηλ. η εγγραφή του φοιτητή με κλειδί κ αποθηκεύεται στη θέση f(κ) του πίνακα.
  - Διαγραφή φοιτητή με κλειδί κ: θέτουμε μηδέν το κλειδί της εγγραφής στη θέση f(κ) του πίνακα

# Παράδειγμα - Συνέχεια

- Αν χρησιμοποιήσουμε για κλειδί το επώνυμο των φοιτητών:
- Υπόθεση: 1) τα επώνυμα είναι διαφορετικά, 2) ένα επώνυμο έχει το πολύ 12 γράμματα
- Ορίζουμε τη βοηθητική συνάρτηση c:
  - c(α)=0, c(β)=1,….,c(ω)=23
- Τότε η συνάρτηση hash f έχει είσοδο το επώνυμο και επιστρέφει ένα μοναδικό ακέραιο:

$$f(a_0 a_1 \ldots \quad \ldots \quad \sum_{i=0}^{11} c(a_i) \cdot 24^i$$

- Συνολικά υπάρχουν $24^{12}$ -1 συμβολοσειρές μήκους 12.
- Τόσες πρέπει να είναι τουλάχιστον και οι θέσεις του πίνακα κερματισμού.
- Μεγάλες απαιτήσεις μνήμης
- Σε πολλές περιπτώσεις οι περισσότερες θέσεις παραμένουν κενές

# Κατακερματισμός με γραμμική ανοικτή διευθυνσιοδότηση

- Εύρος κλειδιών πολύ μεγάλος συνήθως.
- Το μέγεθος του πίνακα κατακερματισμού μικρότερο από το εύρος των κλειδιών
- Επομένως η συνάρτηση κατακερματισμού αντιστοιχεί πολλά διαφορετικά κλειδιά στην ίδια θέση του πίνακα
- Συχνή συνάρτηση κατακερματισμού:

  $f(\kappa)=\kappa \bmod D$

  όπου D είναι το μέγεθος του πίνακα κατακερματισμού
- Κάθε θέση του πίνακα καλείται κάδος (bucket)
- Ο κάδος f(k) καλείται κάδος-σπίτι για το κλειδί κ
- Στην ιδανική περίπτωση, το κλειδί κ είναι αποθηκευμένο στο κάδο-σπίτι.

# Κατακερματισμός με γραμμική ανοικτή διευθυνσιοδότηση – Συν.

- Παράδειγμα:

| ht | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | 80 | | | | 40 | | | 65 |

- Συνάρτηση κατακερματισμού: f(κ)=κ mod 11
- Σύγκρουση (collision) κατά την εισαγωγή του 58: στη θέση 3(=58 mod 11) είναι αποθηκευμένο το 80
- Αποθηκεύουμε το 58 στο επόμενο ελεύθερο κάδο θεωρώντας τον πίνακα κυκλικό

# Κατακερματισμός με γραμμική ανοικτή διευθυνσιοδότηση – Συν.

- Μετά την εισαγωγή του 58 και του 24:

| ht | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 24 | 80 | 58 | | | 40 | | | 65 |

- Μετά την εισαγωγή του 35 (σύγκρουση)

| ht | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 24 | 80 | 58 | 35 | | 40 | | | 65 |

- Μετά την εισαγωγή του 98

| ht | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 98 | | 24 | 80 | 58 | 35 | | 40 | | | 65 |

# Αναζήτηση στοιχείου

- Η αναζήτηση αρχίζει από το κάδο σπίτι f(κ) του κλειδιού κ
- Συνεχίζουμε την αναζήτηση σε διαδοχικούς κάδους μέχρι ένα από τα παρακάτω να συμβούν:
  1. Συναντούμε ένα κάδο που περιέχει ένα στοιχείο με κλειδί κ
  2. Συναντούμε ένα άδειο κόμβο
  3. Επιστρέφουμε στο κάδο-σπίτι
- Στις περιπτώσεις 2 και 3, ο πίνακας δεν περιέχει στοιχείο με κλειδί κ

# Διαγραφή στοιχείου

- Διαγραφή του στοιχείου κ:
  1. Αναζήτηση του κάδου που περιέχει το κλειδί κ
  2. Άδειασε τον κάδο
  3. Μετά κάνε ένα από τα ακόλουθα:
     1. Μετακίνηση μηδέν ή περισσότερων στοιχείων για να γεμίσει ο άδειος κάδος.
        - » Συγκεκριμένα, ξεκινώντας από τον άδειο κάδο και προχωρώντας προς τα δεξιά, ελέγχουμε αν το στοιχείο στην τρέχουσα θέση θα επηρεαστεί από αυτή τη διαγραφή.
        - » Αν το στοιχείο στην τρέχουσα θέση, έχει κάδο-σπίτι πριν από τον άδειο κάδο τότε αυτό μετακινείται στην άδεια θέση. Στην αντίθετη περίπτωση, το εξεταζόμενο στοιχείο δεν επηρεάζεται από τη διαγραφή.
        - » Αν το στοιχείο στην τρέχουσα θέση μετατοπιστεί για να καλύψει τον άδειο κάδο, δημιουργείται άδειος κάδος στην τρέχουσα θέση.
        - » Αυτή η μετακίνηση στοιχείων επαναλαμβάνεται μέχρι να συναντήσουμε άδειο κάδο ή να επιστρέψουμε στο κάδο που έγινε η διαγραφή
     2. Χρησιμοποιούμε ένα επιπλέον πεδίο NeverUsed και θέτουμε τη τιμή του πεδίου 0 (αρχική τιμή 1)
- Αν παραλείψουμε το βήμα 3, η αναζήτηση στοιχείων αποτυγχάνει

# Επιλογή του Διαιρέτη D

- Συνάρτηση κατακερματιμού:
  f(κ)=κ mod D
- Χώρος κλειδιών = όλοι οι ακέραιοι.
- Για κάθε D, το πλήθος των ακεραίων που απεικονίζονται σε ένα κάδο είναι περίπου $2^{32}$ /D
- Ομοιόμορφη κατανομή των κλειδιών στους κάδους όταν όλα τα κλειδιά ισοπίθανα
- Στη πράξη, τα κλειδιά που χρησιμοποιούνται στις διάφορες εφαρμογές παρουσιάζουν συσχέτιση μεταξύ τους.
- Η επιλογή του διαιρέτη D επηρεάζει την κατανομή των κλειδιών στους κάδους.

# Απόδοση κατακερματισμού με γραμμική ανοιχτή διευθυνσιοδότηση

- Χρόνος για αναζήτηση/εισαγωγή/διαγραφή:
  - Θ(n) στη χειρότερη περίπτωση, όπου n είναι το πλήθος των στοιχείων στο πίνακα κατακερματισμού
- Η χειρότερη περίπτωση συμβαίνει όταν τα κλειδιά έχουν το ίδιο κάδο-σπίτι
- Επομένως, ίδια απόδοση στη χειρότερη της μεθόδου κατακερματισμού με αυτή της γραμμικής λίστας
- Πάντως, στη μέση περίπτωση η μέθοδος του κατακερματισμού παρουσιάζει πολύ καλύτερη συμπεριφορά

# Απόδοση κατακερματισμού με γραμμική ανοιχτή διευθυνσιοδότηση

- α = n / D όπου n είναι το πλήθος των στοιχείων που πρόκειται να εισαχθούν στο πίνακα κατακερματισμού
- $0 \leq \alpha \leq 1$
- Το πλήθος των κάδων που εξετάζονται σε μία επιτυχημένη αναζήτηση κατά μέσο όρο = min(D, Sn)

  $Sn \sim \frac{1}{2} (1 + 1/(1-\alpha))$
- Το πλήθος των κάδων που εξετάζονται σε μία μη επιτυχημένη αναζήτηση κατά μέσο όρο = min(D, Un)

  $Un \sim \frac{1}{2} (1+1/(1-a)^2)$
- Ο χρόνος για εισαγωγή και διαγραφή καθορίζεται από τη ποσότητα Un.

# Παράδειγμα

- α= 0,5 => Sn= 1,5  Un= 2,5
- α= 0,9 => Sn= 5,5  Un= 50,5 (Υπόθεση D ≥ 51)
- Στη μέση περίπτωση, όταν ο πίνακας έχει αρκετές άδειες θέσεις (μικρό α), η μέθοδος κατακερματισμού είναι ανώτερη της γραμμικής λίστας.

# Επιλογή του Διαιρέτη D

- Όταν ο διαιρέτης είναι άρτιος αριθμός, οι περιττοί αριθμοί απεικονίζονται σε περιττό κάδο, και οι άρτιοι σε άρτιους κάδους
- 20 mod 14 = 6, 30 mod 14 = 2, 8 mod 14 = 8
- 15 mod 14 = 1, 3 mod 14 = 3, 23mod 14 = 9
- Αν σε μία εφαρμογή χρησιμοποιεί μόνο άρτια ή περιττά κλειδιά, τα κλειδιά απεικονίζονται στις μισές μόνο θέσεις του πίνακα κατακερματισμού

# Επιλογή του Διαιρέτη D

- Όταν ο διαιρέτης D είναι περιττός , τόσο οι περιττοί όσο και οι άρτιοι μπορούν να απεικονισθούν σε περιττό ή άρτιο κάδο.

    20 mod 15 = 5, 30 mod 15 = 0, 8 mod 15 = 8

    15 mod 15 = 0, 3 mod 15 = 3, 23 mod 15 = 8

- Μεγαλύτερη η πιθανότητα, τα κλειδιά να κατανεμηθούν ομοιόμορφα στους κάδους
- Άρα προτιμάται η χρήση περιττού διαιρέτη

# Επιλογή του Διαιρέτη D

- Μη ομοιόμορφη κατανομή έχει παρατηρηθεί όταν ο διαιρέτης είναι πολλαπλάσιο πρώτων αριθμών όπως οι αριθμοί 3, 5, 7, …
- Το πρόβλημα μειώνεται όσο μεγαλύτεροι είναι οι πρώτοι αριθμοί που δίνουν το διαιρέτη.
- Ιδανικά, επιλέγουμε πρώτο αριθμό ως διαιρέτη D.
- Εναλλακτικά, επιλέγουμε τον D έτσι ώστε να είναι πολλαπλάσιο πρώτων αριθμών μεγαλύτερων του 20.

# Επιλογή του Διαιρέτη D

- Μέγεθος του διαιρέτη D
- 1η Μέθοδος:
  - Καθορισμός της αποδεκτής απόδοσης: πόσο χρόνο θέλουμε για κάθε μία πράξη στη χειρότερη περίπτωση
  - Χρήση των εξισώσεων των Un και Sn για το προσδιορισμό του μέγιστου α μπορεί να χρησιμοποιηθεί
  - Από τη τιμή του n και τη τιμή του α, προσδιορισμός της μικρότερης επιτρεπόμενης τιμής του D
  - Τελικά το D προσδιορίζεται ως ο μικρότερος ακέραιος που είναι μεγαλύτερος από το παραπάνω όριο και ταυτόχρονα είναι πρώτος αριθμός ή είναι γινόμενο πρώτων αριθμών μεγαλύτερων του 20.
- 2η Μέθοδος:
  - Προσδιορισμός της μεγαλύτερης επιτρεπόμενης τιμής του D ανάλογα με το διαθέσιμο χώρο
  - Εύρεση του μεγαλύτερου D που είναι μικρότερος από το παραπάνω όριο και ταυτόχρονα είναι πρώτος αριθμός ή είναι γινόμενο πρώτων αριθμών μεγαλύτερων του 20.

```
template<class E, class K>
class HashTable {
  public:
    HashTable(int divisor = 11);
    ~HashTable() {delete [] ht;
                  delete [] empty;}
    bool Search(const K& k, E& e) const;
    HashTable<E,K>& Insert(const E& e);
  private:
    int hSearch(const K& k) const;
    int D; // hash function divisor
    E *ht; // hash table array
    bool *empty; // 1D array
};
```

**Πρόγραμμα 7.11** Ορισμός σε C++ της κλάσης για πίνακες κατ

```
template<class E, class K>
HashTable<E,K>::HashTable(int divisor)
{// Constructor.
   D = divisor;

   // allocate hash table arrays
   ht = new E [D];
   empty = new bool [D];

   // set all buckets to empty
   for (int i = 0; i < D; i++)
      empty[i] = true;
}
```

**Πρόγραμμα 7.12** Μέθοδος κατασκευής για την κλάση HashTable

```
template<class E, class K>
int HashTable<E,K>::hSearch(const K& k) const
{// Search an open addressed table.
 // Return location of k if present.
 // Otherwise return insert point if there is space.
   int i = k % D; // home bucket
   int j = i;     // start at home bucket
   do {
      if (empty[j] || ht[j] == k) return j;
      j = (j + 1) % D;  // next bucket
      } while (j != i); // returned to home?

   return j;  // table full
}

template<class E, class K>
bool HashTable<E,K>::Search(const K& k, E& e) const
{// Put element that matches k in e.
 // Return false if no match.
   int b = hSearch(k);
   if (empty[b] || ht[b] != k) return false;
   e = ht[b];
   return true;
}
```

**Πρόγραμμα 7.13** Συναρτήσεις αναζήτησης

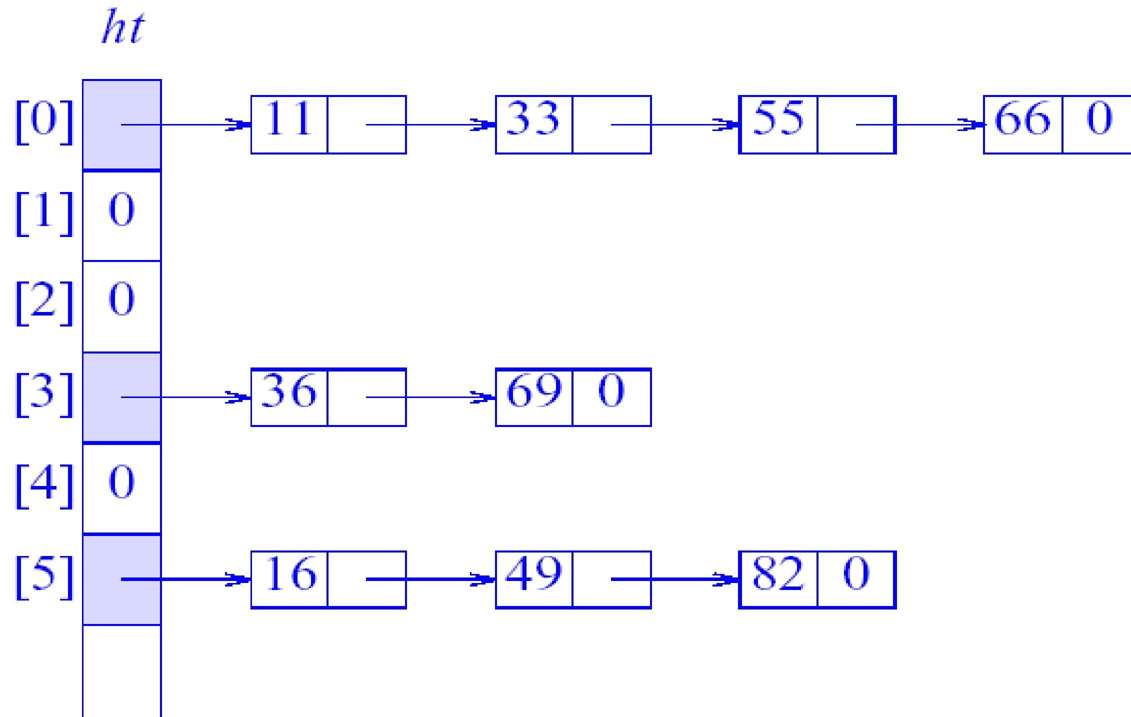
```
template<class E, class K>
HashTable<E,K>& HashTable<E,K>::Insert(const E& e)
{// Hash table insert.
  K k = e; // extract key
  int b = hSearch(k);

  // check if insert is to be done
  if (empty[b]) {empty[b] = false;
                 ht[b] = e;
                 return *this;}

  // no insert, check if duplicate or full
  if (ht[b] == k) throw BadInput(); // duplicate
  throw NoMem(); // table full
}
```

**Πρόγραμμα 7.14** Εισαγωγή σε πίνακα κατακερματισμού

# Κατακερματισμός με αλυσίδες




- Χρήση αλυσίδων για την αποθήκευση στοιχείων που απεικονίζονται στο ίδιο κάδο-σπίτι
- Πιο γρήγορη αναζήτηση αν κάθε αλυσίδα είναι ταξινομημένη

```
template<class E, class K>
class ChainHashTable {
   public:
      ChainHashTable(int divisor = 11)
         {D = divisor;
          ht = new SortedChain<E,K> [D];}
      ~ChainHashTable() {delete [] ht;}
      bool Search(const K& k, E& e) const
         {return ht[k % D].Search(k, e);}
      ChainHashTable<E,K>& Insert(const E& e)
         {ht[e % D].DistinctInsert(e);
          return *this;}
      ChainHashTable<E,K>& Delete(const K& k, E& e)
         {ht[k % D].Delete(k, e);
          return *this;}
   private:
      int D;                          // divisor
      SortedChain<E,K> *ht;           // array of chains
};
```

**Πρόγραμμα 7.15** Πίνακας κατακερματισμού με αλυσίδες

```
template<class E, class K>
class SortedChain {
  public:
    SortedChain() {first = 0;}
    ~SortedChain();
    bool IsEmpty() const {return first == 0;}
    int Length() const;
    bool Search(const K& k, E& e) const;
    SortedChain<E,K>& Delete(const K& k, E& e);
    SortedChain<E,K>& Insert(const E& e);
    SortedChain<E,K>& DistinctInsert(const E& e);
  private:
    SortedChainNode<E,K> *first;
};
```

**Πρόγραμμα 7.1** Κλάση `SortedChain`

```
template<class E, class K>
bool SortedChain<E,K>::Search(const K& k, E& e) const
{// Put element that matches k in e.
 // Return false if no match.

   SortedChainNode<E,K> *p = first;

   // search for match with k
   for (; p && p->data < k; p = p->link);

   // verify match
   if (p && p->data == k) // yes, found match
      {e = p->data; return true;}
   return false; // no match
}
```

```
template<class E, class K>
SortedChain<E,K>& SortedChain<E,K>
                  ::Delete(const K& k, E& e)
{// Delete element that matches k.
 // Put deleted element in e.
 // Throw BadInput exception if no match.

  SortedChainNode<E,K> *p = first,
                       *tp = 0; // trail p

  // search for match with k
  for (; p && p->data < k; tp = p, p = p->link);

  // verify match
  if (p && p->data == k) {// found a match
        e = p->data;  // save data

        // remove p from chain
        if (tp) tp->link = p->link;
        else first = p->link;  // p is first node

        delete p;
        return *this;}
  throw BadInput();// no match
}
```

```
template<class E, class K>
SortedChain<E,K>& SortedChain<E,K>
              ::DistinctInsert(const E& e)
{// Insert e only if no element with same key
 // currently in list.
 // Throw BadInput exception if duplicate.

   SortedChainNode<E,K> *p = first,
                         *tp = 0; // trail p

   // move tp so that e can be inserted after tp
   for (; p && p->data < e; tp = p, p = p->link);

   // check if duplicate
   if (p && p->data == e) throw BadInput();

   // not duplicate, set up node for e
   SortedChainNode<E,K> *q = new SortedChainNode<E,K>;
   q->data = e;

   // insert node just after tp
   q->link = p;
   if (tp) tp->link = q;
   else first = q;

   return *this;
}
```

# Πολυπλοκότητα

- Ανεπιτυχής αναζήτηση:
  - Αν το στοιχείο απεικονίζεται σε μία αλυσίδα με i στοιχεία, το πλήθος των συγκρίσεων:

$$\frac{1}{i}\sum_{j=1}^{i} j = \frac{i+1}{2}$$

  - Κατά μέσο όρο i=α=n/b
  - Άρα $U_n$= (α+1)/2
- Επιτυχής αναζήτηση:
  - Το πλήθος των συγκρίσεων για την αναζήτηση του i-οστου μεγαλύτερου στοιχείου στην αλυσίδα που απεικονίζεται:

1+(i-1)/b

  - Άρα: $S_n = \frac{1}{n}\sum_{i=1}^{n}\left(1+\frac{i-1}{b}\right) = 1+\frac{i-1}{b} \sim 1+\frac{a}{2}$

# Σύγκριση

- $\alpha=0,9$
- Γραμμική ανοικτή Διεύθυνση:
  - $U_n=50,5$
  - $S_n=5,5$
- Χρήση αλυσίδων:
  - $U_n = 0,9$
  - $S_n= 1,45$