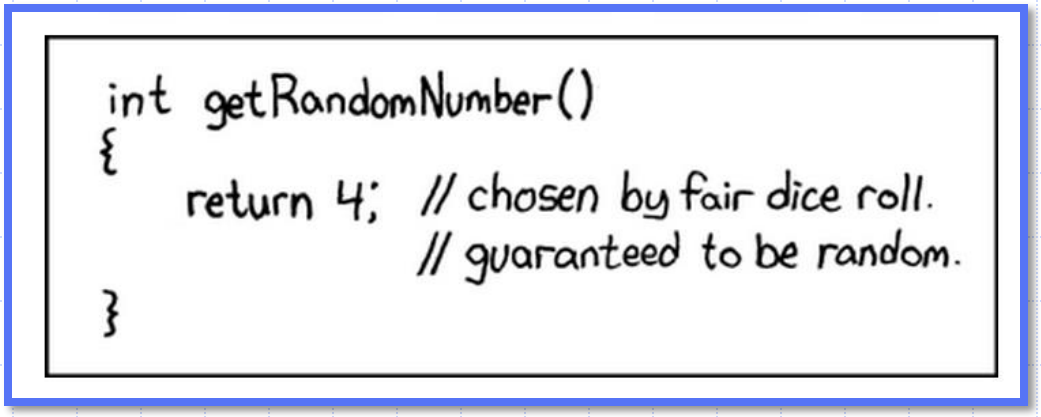
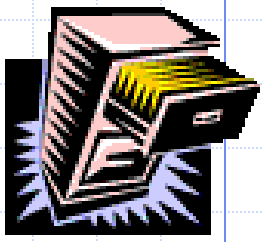
Παρουσίαση για χρήση με το σύγγραμμα, Αλγόριθμοι Σχεδίαση και Εφαρμογές, των M. T. Goodrich and R. Tamassia, Wiley, 2015 (στα ελληνικά από εκδόσεις Μ. Γκιούρδας)

Πίνακες Κατακερματισμού



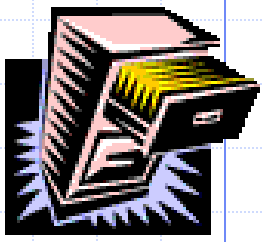
xkcd. http://xkcd.com/221/. "Random Number." Used with permission under Creative Commons 2.5 License.

Θυμηθείτε τις λειτουργίες του πίνακα αντιστοίχισης (map)

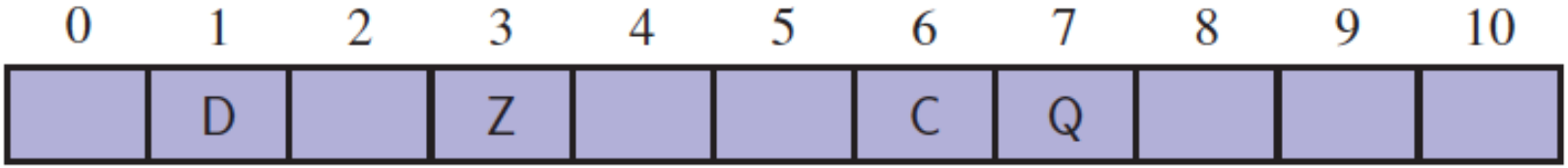


- get(k): Αν ο πίνακας αντιστοίχισης M περιέχει ένα στοιχείο με κλειδί ίσο με k επιστρέφει την τιμή του, αλλιώς επιστρέφει null.
- put(k, v): εισαγωγή στοιχείου (k, v) στον πίνακα αντιστοίχισης M, εάν το κλειδί k δεν υπάρχει στον M επιστρέφει null, αλλιώς επιστρέφει την προηγούμενη τιμή στην οποία αντιστοιχούσε το k.
- remove(k): εάν ο πίνακας αντιστοίχισης M έχει ένα στοιχείο με κλειδί k, αφαίρεση του από τον M και επιστρέφει την τιμή του, αλλιώς επιστρέφει null.
- size(), isEmpty().

Διαισθητική αντίληψη πίνακα αντιστοίχισης

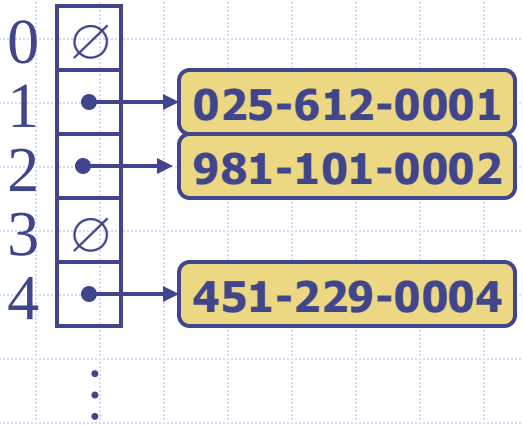


- Διαισθητικά, ένας πίνακας αντιστοίχισης M αποτελεί έναν αφηρημένο τύπο δεδομένων στον οποίο επιτρέπεται η χρήση κλειδιών ως δείκτες με σύνταξη της μορφής M[k].
- Σκεφτείτε μια ειδική περίπτωση όπου γνωρίζουμε ότι ένας πίνακας αντιστοίχισης με n στοιχεία χρησιμοποιεί κλειδιά που είναι ακέραιοι αριθμοί από 0 μέχρι $N - 1$, για κάποιο $N \geq n$.

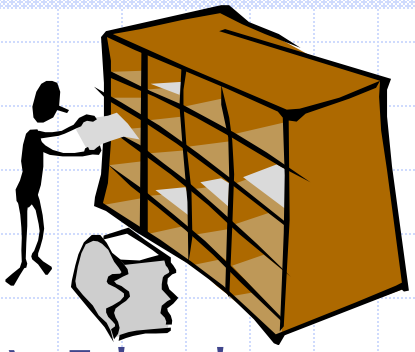


Γενικού τύπου κλειδιά

- Αλλά, τι θα μπορούσαμε να κάνουμε εάν τα κλειδιά δεν είναι ακέραιοι από 0 έως N – 1;
  - Μπορούμε να χρησιμοποιήσουμε **συναρτήσεις κατακερματισμού (hash functions)** για να αντιστοιχήσουμε γενικού τύπου κλειδιά με δείκτες ενός πίνακα.
  - Για παράδειγμα τα 4 τελευταία ψηφία ενός κωδικού (π.χ., ΑΜΚΑ, ΑΦΜ κ.λπ.).



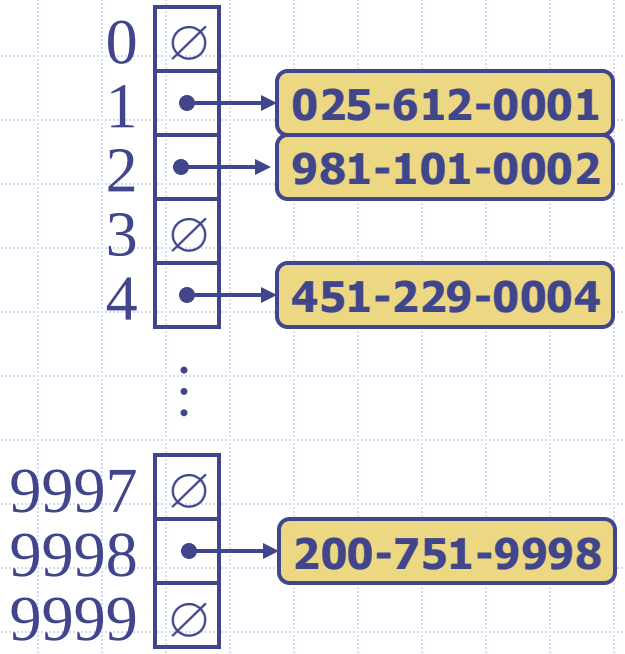
Συναρτήσεις κατακερματισμού και πίνακες κατακερματισμού



- Μία συνάρτηση κατακερματισμού h αντιστοιχίζει κλειδιά ενός δεδομένου τύπου σε ακεραίους που βρίσκονται στο διάστημα $[0, N-1]$.
- Παράδειγμα:
  $$h(x) = x \bmod N$$
  είναι μία συνάρτηση κατακερματισμού για ακέραια κλειδιά.
- Ο ακέραιος $h(x)$ καλείται τιμή κατακερματισμού του κλειδιού x.
- Ένας πίνακας κατακερματισμού για ένα δεδομένο τύπο κλειδιού αποτελείται από:
  - Μια συνάρτηση κατακερματισμού h.
  - Έναν πίνακα μεγέθους N.
- Στην υλοποίηση πίνακα αντιστοίχισης με πίνακα κατακερματισμού ο σκοπός είναι η αποθήκευση του στοιχείου (k, o) στο δείκτη $i = h(k)$.

Παράδειγμα

- Ως παράδειγμα εξετάζουμε έναν **πίνακα αντιστοίχισης** στον οποίο αποθηκεύουμε εγγραφές ως (Κωδικός, Όνομα), όπου ο κωδικός υποθέτουμε ότι είναι ένας θετικός δεκαψήφιος ακέραιος
- Ο **πίνακας κατακερματισμού** είναι ένας πίνακας μεγέθους $N = 10{,}000$ και η συνάρτηση κατακερματισμού είναι $h(x) =$ τα τέσσερα τελευταία ψηφία του x.



Συναρτήσεις κατακερματισμού

- Μία συνάρτηση κατακερματισμού συνήθως ορίζεται ως η σύνθεση δύο συναρτήσεων:

  Κώδικας κατακερματισμού:

  h_1: κλειδιά $\rightarrow$ ακέραιοι

  Συνάρτηση συμπίεσης:

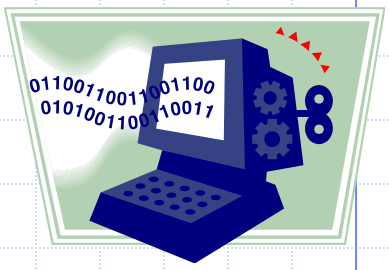
  h_2: ακέραιοι $\rightarrow [0, N-1]$

- Ο κώδικας κατακερματισμού υπολογίζεται πρώτα και στη συνέχεια εφαρμόζεται η συνάρτηση συμπίεσης:

  $$h(x) = h_2(h_1(x))$$

- Ο σκοπός της συνάρτησης κατακερματισμού είναι να “απλώσει” τα κλειδιά με όσο το δυνατόν πιο τυχαίο τρόπο.

Υλοποιήσεις συναρτήσεων κατακερματισμού



- Διεύθυνσης μνήμης:
  - Ερμηνεύουμε τη διεύθυνση μνήμης στην οποία βρίσκεται το κλειδί ως μια ακέραια τιμή. Καλή λύση γενικά, αλλά όχι για κλειδιά που είναι αριθμοί ή συμβολοσειρές.
- Ακέραια μετατροπή:
  - Ερμηνεύουμε τα bits του κλειδιού ως ακέραια τιμή
  - Κατάλληλο για κλειδιά μεγέθους ίσου ή μικρότερου πλήθους από τα bits του ακέραιου τύπου που θα επιλεγεί (π.χ. byte, short, int and float).
- Συνιστώσες αθροίσματος:
  - Χωρίζουμε τα bits του κλειδιού σε τμήματα (π.χ. 16 ή 32 bits) και αθροίζουμε τα τμήματα (αγνοώντας τις υπερχειλίσεις που μπορεί να συμβούν).
  - Κατάλληλο για αριθμητικά κλειδιά σταθερού μεγέθους ίσου ή μεγαλύτερου από τα bits του ακέραιου τύπου που θα επιλεγεί.

Υλοποιήσεις συναρτήσεων κατακερματισμού (συνέχεια)

- Πολυωνυμική συσσώρευση:
  - Χωρίζουμε τα bits του κλειδιού σε ακολουθίες τμημάτων σταθερού μεγέθους (π.χ. 8, 16 ή 32 bits):
    $$a_0 a_1 \ldots a_{n-1}$$
  - Για μία σταθερή τιμή z υπολογίζουμε το πολυώνυμο:
    $$p(z) = a_0 + a_1 z + a_2 z^2 + \ldots + a_{n-1} z^{n-1}$$
    αγνοώντας τυχόν υπερχειλίσεις
  - Ιδιαίτερα χρήσιμο για συμβολοσειρές (π.χ. η επιλογή $z = 33$ δημιουργεί το πολύ 6 συγκρούσεις σε ένα σύνολο 50,000 Αγγλικών λέξεων).
- Το πολυώνυμο $p(z)$ μπορεί να υπολογιστεί σε $O(n)$ χρόνο χρησιμοποιώντας τον κανόνα του Horner:
  - Τα ακόλουθα πολυώνυμα υπολογίζονται διαδοχικά, κάθε ένα από το προηγούμενο σε χρόνο $O(1)$:
    $$p_0(z) = a_{n-1}$$
    $$p_i(z) = a_{n-i-1} + z p_{i-1}(z) \quad (i = 1, 2, \ldots, n-1)$$
- Έχουμε $p(z) = p_{n-1}(z)$.

Κατακερματισμός που βασίζεται σε πίνακα

- Ας υποθέσουμε ότι ένα κλειδί μπορεί να θεωρηθεί ως μία πλειάδα, $k = (x_1, x_2, \ldots, x_d)$, για σταθερό d, όπου κάθε x_i είναι στο διάστημα $[0, M - 1]$.
- Υπάρχει μία κατηγορία συναρτήσεων κατακερματισμού, γνωστή ως **συναρτήσεις κατακερματισμού που βασίζονται σε πίνακα**.
- Αρχικοποιούμε d πίνακες, $T_1, T_2, \ldots, T_d$, μεγέθους M ο καθένας, έτσι ώστε κάθε $T_i[j]$ να είναι ένας ομοιόμορφα επιλεγμένος ανεξάρτητος τυχαίος αριθμός στο διάστημα $[0, N - 1]$.
- Μπορούμε να υπολογίσουμε την συνάρτηση κατακερματισμού h(k) ως

$$h(k) = T_1[x_1] \oplus T_2[x_2] \oplus \ldots \oplus T_d[x_d],$$

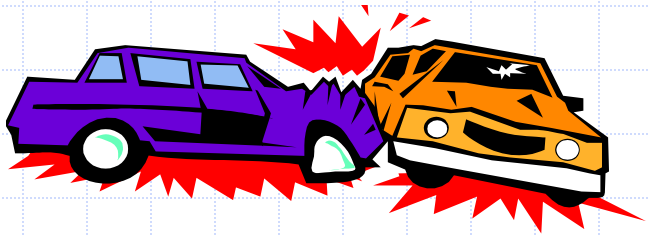
όπου το σύμβολο "$\oplus$" αναπαριστά τη λειτουργία XOR.

- Επειδή οι τιμές στους πίνακες επιλέγονται τυχαία, μία τέτοια συνάρτηση είναι αρκετά τυχαία. Για παράδειγμα, μπορούμε να αποδειχθεί ότι μια τέτοια συνάρτηση θα προκαλέσει σύγκρουση μεταξύ δύο διαφορετικών κλειδιών στην ίδια τιμή κατακερματισμού με πιθανότητα 1/N, η οποία είναι αυτή που θα επέστρεφε μία τέλεια τυχαία συνάρτηση.

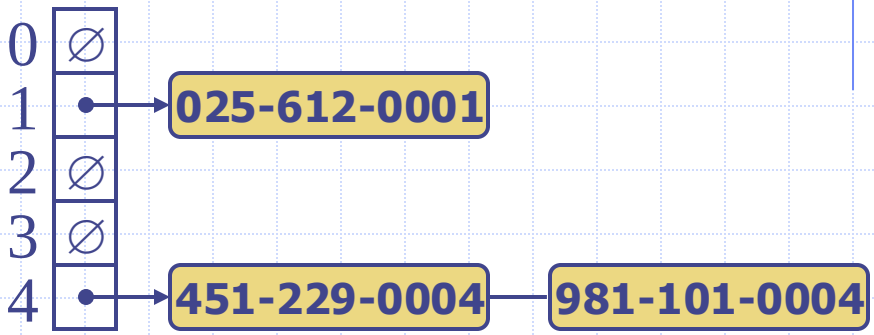
Συναρτήσεις συμπίεσης

- Διαίρεση:
  - $h_2(y) = y \bmod N$
  - Το μέγεθος N του πίνακα κατακερματισμού συνήθως επιλέγεται να είναι ένας πρώτος αριθμός (θεωρία αριθμών).
- Τυχαία γραμμική συνάρτηση κατακερματισμού:
  - $h_2(y) = (ay + b) \bmod N$
  - a και b είναι τυχαίοι μη αρνητικοί ακέραιοι έτσι ώστε:
    $a \bmod N \neq 0$
  - Αλλιώς κάθε ακέραιος θα αντιστοιχιζόταν στην ίδια τιμή b.

Διαχείριση συγκρούσεων



- Οι συγκρούσεις συμβαίνουν όταν διαφορετικά στοιχεία αντιστοιχίζονται στο ίδιο κελί
- Ξεχωριστή αλυσίδωση (separate chaining): κάθε κελί του πίνακα δείχνει σε μία συνδεδεμένη λίστα από στοιχεία που αντιστοιχίζονται σε αυτό.



- Η ξεχωριστή αλυσίδωση είναι απλή αλλά απαιτεί επιπλέον μνήμη πέρα από τον ίδιο τον πίνακα.

Πίνακας αντιστοίχισης με ξεχωριστή αλυσίδωση

Ανάθεση λειτουργιών σε έναν πίνακα αντιστοίχισης που χρησιμοποιεί λίστα για κάθε κλειδί:

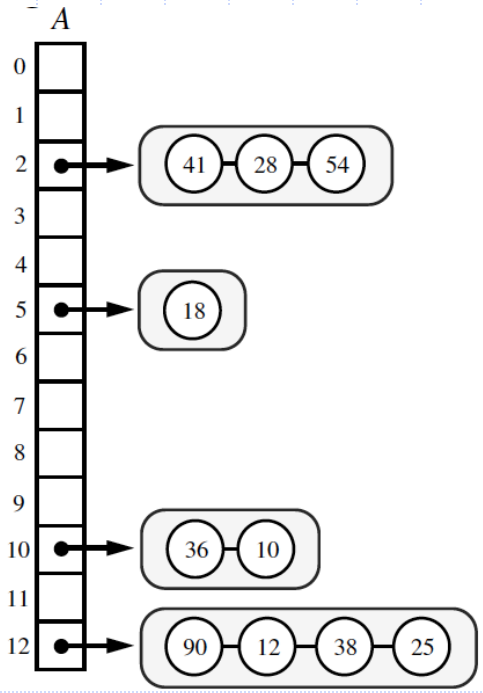
```
Algorithm get(k):
return A[h(k)].get(k)

Algorithm put(k,v):
t = A[h(k)].put(k,v)
if t = null then                    {το k είναι το νέο κλειδί}
    n = n + 1
return t

Algorithm remove(k):
t = A[h(k)].remove(k)
if t ≠ null then                    {το k βρέθηκε}
    n = n - 1
return t
```

Απόδοση ξεχωριστής αλυσίδωσης

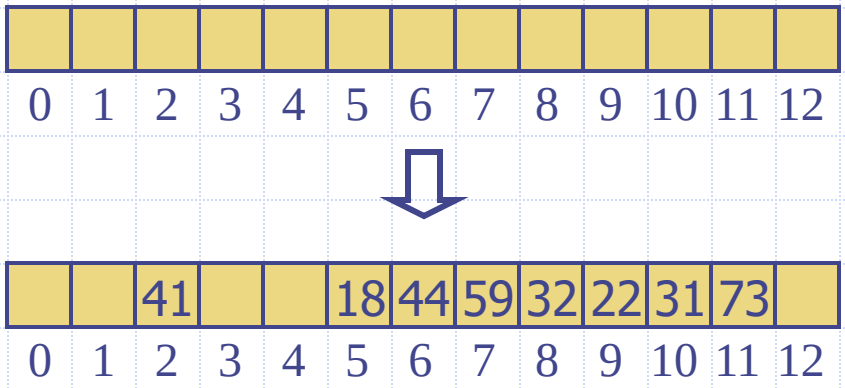
- Ας υποθέσουμε ότι η συνάρτηση κατακερματισμού, h, αντιστοιχίζει κλειδιά σε ανεξάρτητες ομοιόμορφες τυχαίες τιμές στο διάστημα [0, N−1].
- Συνεπώς, εάν θεωρήσουμε το X ως μία τυχαία μεταβλητή που αναπαριστά το πλήθος των στοιχείων που αντιστοιχίζονται σε μια θέση i του πίνακα A, τότε η αναμενόμενη τιμή του X είναι E(X) = n/N, όπου n είναι το πλήθος των στοιχείων στον χάρτη επειδή κάθε στοιχείο έχει ίδιες πιθανότητες να τοποθετηθεί σε καθεμία απ' τις θέσεις N του A.
- Αυτή η παράμετρος, n/N, δηλαδή ο λόγος του πλήθους των στοιχείων σ' έναν πίνακα κατακερματισμού n προς την χωρητικότητα του πίνακα N ονομάζεται **συντελεστής φόρτωσης (load factor)** του πίνακα κατακερματισμού.
- Ο αναμενόμενος χρόνος για λειτουργίες σε έναν πίνακα κατακερματισμού είναι O(1) όταν οι συγκρούσεις αντιμετωπίζονται με ξεχωριστή αλυσίδωση.



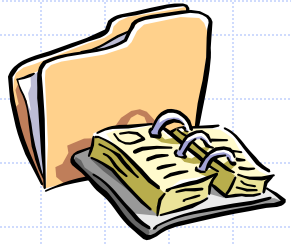
Γραμμική ανίχνευση (linear probing)

- Ανοιχτή διευθυνσιοδότηση (open addressing): το στοιχείο που συγκρούεται τοποθετείται σε διαφορετικό κελί του πίνακα.
- Γραμμική ανίχνευση: διαχειρίζεται τις συγκρούσεις τοποθετώντας το στοιχείο που συγκρούεται στο επόμενο (κυκλικά) διαθέσιμο κελί του πίνακα.
- Κάθε κελί του πίνακα που εξετάζεται αναφέρεται ως "ανίχνευση" (probe).
- Τα στοιχεία που συγκρούονται τείνουν να μαζεύονται μαζί, προκαλώντας μελλοντικές συγκρούσεις με μεγαλύτερες ακολουθίες ανιχνεύσεων.

- Παράδειγμα:
  - $h(x) = x \bmod 13$
  - Εισαγωγή των κλειδιών 18, 41, 22, 44, 59, 32, 31, 73, με αυτήν τη σειρά.



Αναζήτηση με γραμμική ανίχνευση



- Θεωρείστε έναν πίνακα κατακερματισμού A που χρησιμοποιεί γραμμική ανίχνευση.
- get(k)
  - Ξεκινάμε στο κελί $h(k)$
  - Πραγματοποιούμε ανίχνευση σε διάφορες θέσεις μέχρι να συμβεί ένα από τα ακόλουθα:
    - Να βρεθεί ένα στοιχείο με κλειδί k.
    - Να βρεθεί ένα κενό κελί.
    - Να ανιχνευθούν N κελιά χωρίς επιτυχία.

Algorithm *get*(k)
  $i \leftarrow h(k)$
  $p \leftarrow 0$
  repeat
    $c \leftarrow A[i]$
    if $c = \emptyset$
      return *null*
    else if $c.getKey() = k$
      return $c.getValue()$
    else
      $i \leftarrow (i + 1) \bmod N$
      $p \leftarrow p + 1$
  until $p = N$
  return *null*

Εισαγωγές και διαγραφές με γραμμική ανίχνευση

- Για να χειριστούμε τις διαγραφές και τις εισαγωγές, ορίζουμε ένα νέο ειδικό αντικείμενο με όνομα ***DEFUNCT (ανενεργό)***, το οποίο τοποθετείται στη θέση των διαγραμμένων στοιχείων **ή εναλλακτικά χρησιμοποιούμε τις υλοποιήσεις των remove, put που ακολουθούν**:
- remove(k)
  - Αναζητούμε την εγγραφή με κλειδί k.
  - Εάν η εγγραφή (k, v) βρεθεί, μετακινούμε στοιχεία έτσι ώστε να καλυφθεί η "τρύπα" που δημιουργήθηκε με την αφαίρεση του.
- put(k, v)
  - Επιστρέφει σφάλμα εάν ο πίνακας είναι πλήρης.
  - Ξεκινάμε στο κελί $h(k)$.
  - Ανιχνεύουμε συνεχόμενα κελιά μέχρι να βρεθεί κάποιο κελί i που να είναι κενό.
  - Αποθηκεύουμε το στοιχείο (k, v) στο κελί i.

Ψευδό-κώδικας για get και put

- get(k):

```
i ← h(k)
while A[i] ≠ NULL do
    if A[i].key = k then
        return A[i]
    i ← (i + 1) mod N
return NULL
```

- put(k, v):

```
i ← h(k)
while A[i] ≠ NULL do
    if A[i].key = k then
        A[i] ← (k, v)        // replace the old (k, v')
    i ← (i + 1) mod N
A[i] ← (k, v)
```

Ψευδό-κώδικας για remove

- remove(k):

```
i ← h(k)
while A[i] ≠ NULL do
    if A[i].key = k then
        temp ← A[i]
        A[i] ← NULL
        Call Shift(i) to restore A to a stable state without k
        return temp
    i ← (i + 1) mod N
return NULL
```

- Shift(i):

```
s ← 1        // the current shift amount
while A[(i + s) mod N] ≠ NULL do
    j ← h(A[(i + s) mod N].key)        // preferred index for this item
    if j ∉ (i, i + s] mod N then
        A[i] ← A[(i + s) mod N]        // fill in the "hole"
        A[(i + s) mod N] ← NULL        // move the "hole"
        i ← (i + s) mod N
        s ← 1
    else
        s ← s + 1
```

Απόδοση της γραμμικής ανίχνευσης

- Στην χειρότερη περίπτωση οι αναζητήσεις, εισαγωγές και διαγραφές απαιτούν $O(n)$ χρόνο.
- Η χειρότερη περίπτωση συμβαίνει όταν όλα τα κλειδιά που εισάγονται συγκρούονται.
- Ο συντελεστής φόρτωσης (load factor) $\alpha = n/N$ επηρεάζει την απόδοση του πίνακα κατακερματισμού.
- Υποθέτοντας ότι οι τιμές προς κατακερματισμό συμπεριφέρονται σαν τυχαίοι αριθμοί, μπορεί να αποδειχθεί ότι ο αναμενόμενος αριθμός αναζητήσεων για μία εισαγωγή με ανοιχτή διευθυνσιοδότηση θα είναι $1 / (1 - \alpha)$
- Ο αναμενόμενος χρόνος για τις λειτουργίες του ΑΤΔ (Αφηρημένου Τύπου Δεδομένων) λεξικού (dictionary) σε έναν πίνακα κατακερματισμού είναι $O(1)$, όταν ο συντελεστής φόρτωσης δεν είναι κοντά στο 100%.
- Στην πράξη ο κατακερματισμός είναι πολύ γρήγορος.
- Εφαρμογές πινάκων κατακερματισμού:
  - Μικρές βάσεις δεδομένων.
  - Μεταγλωττιστές.
  - Λανθάνουσα μνήμη web browsers.

Μία πιο προσεκτική ανάλυση της γραμμικής ανίχνευσης

- Θυμηθείτε ότι στην τεχνική γραμμικής ανίχνευσης για τη διαχείριση συγκρούσεων, όταν η εισαγωγή προκαλεί σύγκρουση στο κελί i, τότε εισάγεται το νέο στοιχείο στο πρώτο κελί από τα i+1, i+2, κτλ. μέχρι να βρεθεί ένα κενό κελί.

Let $X_1, X_2, \ldots, X_n$ be a set of mutually independent indicator random variables, such that each X_i is 1 with some probability $p_i > 0$ and 0 otherwise. Let $X = \sum_{i=1}^{n} X_i$ be the sum of these random variables, and let μ denote the mean of X, that is, $\mu = E(X) = \sum_{i=1}^{n} p_i$. The following bound, which is due to Chernoff (and which we derive in Section 19.5), establishes that, for $\delta > 0$,

$$\Pr(X > (1+\delta)\mu) < \left[\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right]^{\mu}.$$

- Για αυτήν την ανάλυση ας υποθέσουμε ότι αποθηκεύουμε n στοιχεία σε έναν πίνακα κατακερματισμού μεγέθους N = 2n, οπότε ο πίνακας κατακερματισμού έχει συντελεστή φόρτωσης 1/2.

Μία πιο προσεκτική ανάλυση της γραμμικής αναζήτησης, 2

Let X denote a random variable equal to the number of probes that we would perform in doing a search or update operation in our hash table for some key, k. Furthermore, let X_i be a 0/1 indicator random variable that is 1 if and only if $i = h(k)$, and let Y_i be a random variable that is equal to the length of a run of contiguous nonempty cells that begins at position i, wrapping around the end of the table if necessary. By the way that linear probing works, and because we assume that our hash function $h(k)$ is random,

$$X = \sum_{i=0}^{N-1} X_i(Y_i + 1),$$

which implies that

$$\begin{aligned} E(X) &= \sum_{i=0}^{N-1} \frac{1}{2n} E(Y_i + 1) \\ &= 1 + (1/2n)E\left(\sum_{i=0}^{N-1} Y_i\right). \end{aligned}$$

Έτσι αν μπορούμε να φράξουμε την αναμενόμενη τιμή του αθροίσματος των Y_i's, τότε μπορούμε να φράξουμε τον αναμενόμενο χρόνο για λειτουργία αναζήτησης ή ενημέρωσης με την τεχνική της γραμμικής ανίχνευσης.

Μία πιο προσεκτική ανάλυση της γραμμικής αναζήτησης, 3

Consider, then, a maximal contiguous sequence, S, of k nonempty table cells, that is, a contiguous group of occupied cells that has empty cells next to its opposite ends. Any search or update operation that lands in S will, in the worst case, march all the way to the end of S. That is, if a search lands in the first cell of S, it would make k wasted probes, if it lands in the second cell of S, it would make $k-1$ wasted probes, and so on. So the total cost of all the searches that land in S can be at most k^2. Thus, if we let $Z_{i,k}$ be a 0/1 indicator random variable for the existence of a maximal sequence of nonempty cells of length k, then

$$\sum_{i=0}^{N-1} Y_i \leq \sum_{i=0}^{N-1} \sum_{k=1}^{2n} k^2 Z_{i,k}.$$

Put another way, it is as if we are "charging" each maximal sequence of nonempty cells for all the searches that land in that sequence.

Μία πιο προσεκτική ανάλυση της γραμμικής αναζήτησης, 4

So, to bound the expected value of the sum of the Y_i's, we need to bound the probability that $Z_{i,k}$ is 1, which is something we can do using the Chernoff bound given above. Let Z_k denote the number of items that are mapped to a given sequence of k cells in our table. Then,

$$\Pr(Z_{i,k} = 1) \leq \Pr(Z_k \geq k).$$

Because the load factor of our table is $1/2$, $E(Z_k) = k/2$. Thus, by the above Chernoff bound,

$$\begin{aligned} \Pr(Z_k \geq k) &= \Pr(Z_k \geq 2(k/2)) \\ &\leq (e/4)^{k/2} \\ &< 2^{-k/4}. \end{aligned}$$

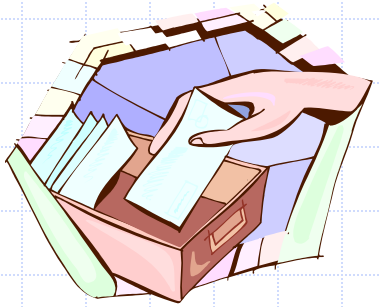
Therefore, putting all the above pieces together,

$$\begin{aligned} E(X) &= 1 + (1/2n)E\left(\sum_{i=0}^{N-1} Y_i\right) \\ &\leq 1 + (1/2n)\sum_{i=0}^{N-1}\sum_{k=1}^{2n} k^2\, 2^{-k/4} \\ &\leq 1 + \sum_{k=1}^{\infty} k^2\, 2^{-k/4} \\ &= O(1). \end{aligned}$$

Συμπέρασμα!

That is, the expected running time for doing a search or update operation with linear probing is $O(1)$, so long as the load factor in our hash table is at most $1/2$.

Διπλός κατακερματισμός



- Ο διπλός κατακερματισμός χρησιμοποιεί μία δευτερεύουσα συνάρτηση κατακερματισμού $d(k)$ και διαχειρίζεται τις συγκρούσεις τοποθετώντας ένα στοιχείο στο πρώτο διαθέσιμο κελί της σειράς: $(i + jd(k)) \bmod N$ για $j = 0, 1, \ldots, N - 1$.
- Η δευτερεύουσα συνάρτηση κατακερματισμού $d(k)$ δεν μπορεί να επιστρέφει μηδενικές τιμές.
- Το μέγεθος N του πίνακα πρέπει να είναι πρώτος αριθμός ώστε να επιτρέπει την ανίχνευση (probing) όλων των κελιών.

- Συνηθισμένη επιλογή συνάρτησης συμπίεσης για τη δευτερεύουσα συνάρτησης κατακερματισμού είναι η ακόλουθη:

  $d_2(k) = q - k \bmod q$

  όπου
  - q πρώτος αριθμός και $q < N$
- Οι πιθανές τιμές $d_2(k)$ είναι $1, 2, \ldots, q$

Παράδειγμα διπλού κατακερματισμού

- Έστω ένας πίνακας κατακερματισμού που αποθηκεύει ακέραια κλειδιά:
  - $N = 13$
  - $h(k) = k \bmod 13$
  - $d(k) = 7 - k \bmod 7$
- Εισαγωγή των κλειδιών 18, 41, 22, 44, 59, 32, 31, 73, με αυτή τη σειρά.

k	$h(k)$	$d(k)$	Probes		
18	5	3	5		
41	2	1	2		
22	9	6	9		
44	5	5	5	10	
59	7	4	7		
32	6	3	6		
31	5	4	5	9	0
73	8	4	8