Δομές Δεδομένων

Μέρος 8ο: Χάρτες ή λεξικά

Εξάμηνο Σπουδών: 6ο

Κωδικός Μαθήματος: 681

Τμήμα Μαθηματικών
Πανεπιστήμιο Ιωαννίνων

Μιχάλης Α. Μπέκος
bekos@uoi.gr

Χάρτες ή λεξικά

Maps / dictionaries

- Ένας χάρτης ή λεξικό μοντελοποιεί μια συλλογή από εγγραφές τύπου ⟨κλειδί, τιμή⟩ η οποία είναι αναζητήσιμη με βάση το κλειδί.
- Οι βασικές λειτουργίες ενός χάρτη είναι:
  - Αναζήτηση εγγραφής
  - Εισαγωγή εγγραφής
  - Διαγραφή εγγραφής
- Πολλαπλές εγγραφές με το ίδιο κλειδί δεν επιτρέπονται
- Εφαρμογές:
  - Τηλεφωνικός κατάλογος
  - Βάση δεδομένων βαθμολογίας φοιτητών

Ο ΑΤΔ Χάρτης

- Ο ΑΤΔ χάρτης υποστηρίζει τις παρακάτω λειτουργίες:

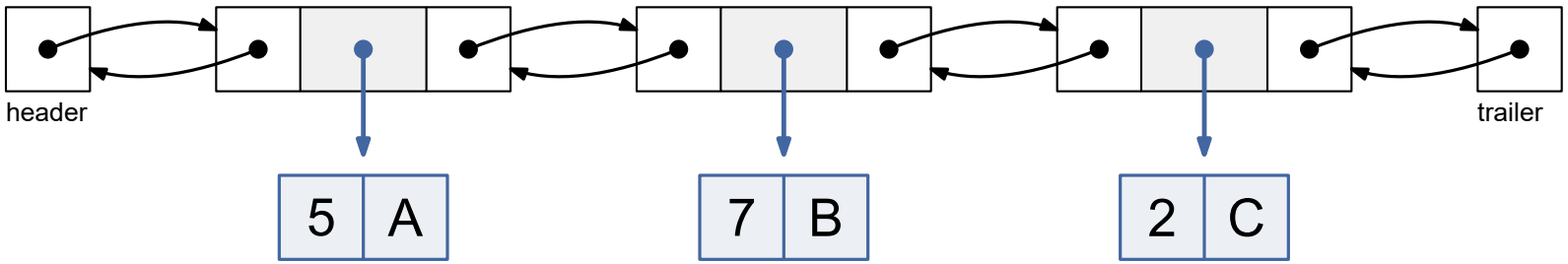
`empty()`	ελέγχει αν η δομή είναι κενή
`size()`	επιστρέφει το πλήθος των εγγραφών της δομής
`find(key k)`	αν υπάρχει εγγραφή με κλειδί `k`, επιστρέφει διαπροσπελαστή σε αυτή διαφορετικά, επιστρέφει “ειδικό” διαπροσπελαστή στο τέλος της δομής
`put(key k, value v)`	αν δεν υπάρχει εγγραφή με κλειδί `k`, εισάγει το ζεύγος `(k, v)` στη δομή, διαφορετικά ανανεώνει την τιμή της υπάρχουσας εγγραφής σε `v`, και επιστρέφει έναν διαπροσπελαστή στη νέα/τροποποιημένη εγγραφή.
`erase(key k)`	αν υπάρχει μια εγγραφή με κλειδί `k`, την αφαιρεί από τη δομή
`begin()`	επιστρέφει διαπροσπελαστή στην αρχή της δομής
`end()`	επιστρέφει διαπροσπελαστή στο τέλος της δομής

Παράδειγμα: Χρήση χάρτη

Μέθοδος	Τιμή επιστροφής	Περιεχόμενα χάρτη
`empty()`	`true`	∅
`put(5,A)`	`[5,A]`	`[5,A]`
`put(7,B)`	`[7,B]`	`[5,A], [7,B]`
`put(2,C)`	`[2,C]`	`[5,A], [7,B], [2,C]`
`empty()`	`false`	`[5,A], [7,B], [2,C]`
`put(2,E)`	`[2,E]`	`[5,A], [7,B], [2,E]`
`find(7)`	`B`	`[5,A], [7,B], [2,E]`
`find(4)`	`end`	`[5,A], [7,B], [2,E]`
`size()`	`3`	`[5,A], [7,B], [2,E]`
`erase(4)`	–	`[5,A], [7,B], [2,E]`
`erase(2)`	–	`[5,A], [7,B]`

Υλοποίηση του ΑΤΔ Χάρτη με λίστα

- Χρησιμοποιούμε μια μη-ταξινομημένη (διπλά συνδεδεμένη) λίστα
- Αποθηκεύουμε τις εγγραφές του χάρτη στη λίστα
- Παράδειγμα:



Υλοποίηση του ΑΤΔ Χάρτη με λίστα: Αναζήτηση

- Για την αναζήτηση εγγραφής με κλειδί `k`:
  - γίνεται διαπέραση της λίστα κατά την οποία ελέγχονται όλα τα στοιχεία της
  - αν υπάρχει εγγραφή με κλειδί `k`, επιστρέφεται διαπροσπελαστής στην αντίστοιχη θέση
  - διαφορετικά, επιστρέφεται διαπροσπελαστής στο τέλος της λίστας

```
1 Algorithm find(k)
2 |   foreach p in [S.begin(),S.end()) do
3 |   |   if p->key() == k then return p;
4 |   return S.end();
```

Υλοποίηση του ΑΤΔ Χάρτη με λίστα: Εισαγωγή

- Για την εισαγωγή εγγραφής με κλειδί `k` και τιμή `v`:
  - αρχικά γίνεται έλεγχος αν υπάρχει εγγραφή με κλειδί `k` στη λίστα
  - αν υπάρχει, η τιμή της υπάρχουσας εγγραφής ανανεώνεται σε `v`
  - διαφορετικά, το ζεύγος `(k, v)` εισάγεται στο τέλος της λίστας

```
1 Algorithm put(k,v)
2 |  foreach p in [S.begin(),S.end()) do
3 |  |  if p->key() == k then
4 |  |  |  p->set_value(v);
5 |  |  |  return p;
6 |  p = S.insert_back(Entry(k,v));
7 |  return p;
```

Υλοποίηση του ΑΤΔ Χάρτη με λίστα: Διαγραφή

- Για τη διαγραφή της εγγραφής με κλειδί `k`:
  - αρχικά γίνεται έλεγχος αν υπάρχει εγγραφή με κλειδί `k` στη λίστα
  - αν υπάρχει, τότε διαγράφεται από τη λίστα.

```
1 Algorithm erase(k)
2 |  foreach p in [S.begin(),S.end()) do
3 |  |  if p->key() == k then S.erase(p);
```

Υλοποίηση του ΑΤΔ Χάρτη με λίστα: Απόδοση

- Η εισαγωγή απαιτεί $O(n)$ χρόνο, αφού γίνεται έλεγχος αν υπάρχει ήδη στην λίστα
- Η εύρεση και η διαγραφή απαιτούν $O(n)$ χρόνο, αφού στη χειρότερη περίπτωση (το στοιχείο δεν βρέθηκε) πρέπει να προσπελάσουμε ολόκληρη τη λίστα για την αναζήτηση του στοιχείου με το δεδομένο κλειδί

	Unsorted list
`empty()`	$\mathcal{O}(1)$
`size()`	$\mathcal{O}(1)$
`put(k,v)`	$\mathcal{O}(n)$
`erase(k)`	$\mathcal{O}(n)$
`find(k)`	$\mathcal{O}(n)$

- Παρατήρηση: Η υλοποίηση με μη-ταξινομημένη λίστα είναι αποδοτική μόνο για μικρού μεγέθους χάρτες

Συναρτήσεις και πίνακες κατακερματισμού

Hash functions / hash tables

- Μια συνάρτηση κατακερματισμού `h` αντιστοιχίζει κλειδιά συγκεκριμένου τύπου σε ακέραιους αριθμούς σε ένα σταθερό διάστημα `[0, N-1]`
  - Ο ακέραιος `h(x)` ονομάζεται τιμή κατακερματισμού του κλειδιού `x`

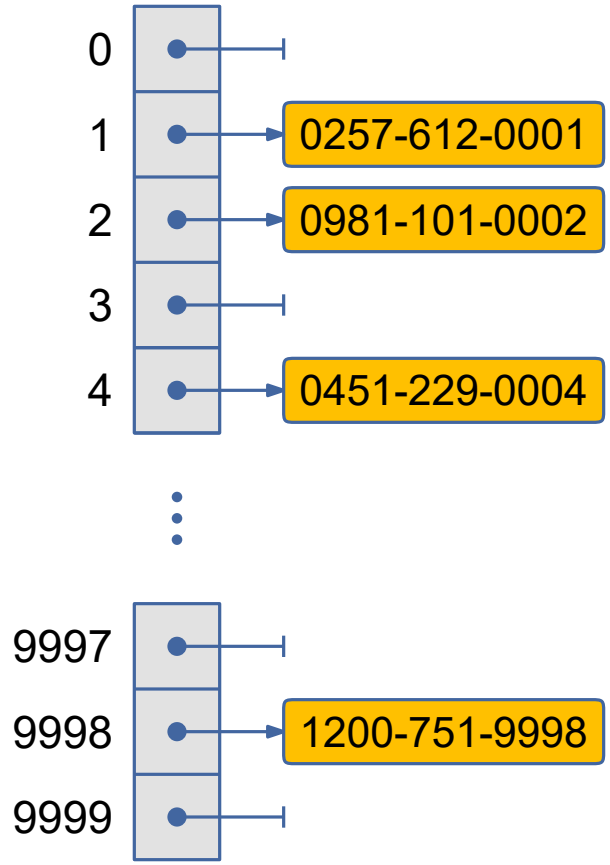
- Παράδειγμα: Μια συνάρτηση κατακερματισμού για ακέραια κλειδιά είναι η:
  $$h(x) = x \mod N$$

- Ένας πίνακας κατακερματισμού για έναν δεδομένο τύπο κλειδιού αποτελείται από
  - μια συνάρτηση κατακερματισμού h
  - ένα πίνακα μεγέθους N

- Κατά την υλοποίηση ενός χάρτη με πίνακα κατακερματισμού, ο στόχος είναι η αποθήκευση της εγγραφής `(k, v)` στη θέση του πίνακα με δείκτη `h(k)`
- Μια σύγκρουση συμβαίνει όταν δύο κλειδιά αντιστοιχίζονται στην ίδια τιμή κατακερματισμού.

Πίνακες κατακερματισμού: Παράδειγμα

- Σχεδιάζουμε ένα πίνακα κατακερματισμού για ένα χάρτη, ο οποίος αποθηκεύει εγγραφές όπως (ΑΜΚΑ, ονοματεπώνυμο), όπου ο ΑΜΚΑ είναι ένας θετικός ακέραιος 11 ψηφιών.



- Ο πίνακας κατακερματισμού χρησιμοποιεί ένα διάνυσμα με `N = 100.000` θέσεις και τη συνάρτηση κατακερματισμού:

  `h(x) =` τα τελευταία 4 ψηφία του `x`.

- Τι γίνεται όμως αν τα κλειδιά δεν είναι ακέραιοι;

Κώδικες κατακερματισμού και συναρτήσεις συμπίεσης

Hash codes / compression functions

- Μία συνάρτηση κατακερματισμού συνήθως ορίζεται ως ο συνδυασμός δύο συναρτήσεων:
  - Κώδικας κατακερματισμού: $h_1 : \text{keys} \to \mathbb{N}$
  - Συνάρτηση συμπίεσης: $h_2 : \mathbb{N} \to [0, N-1]$

- Ο κώδικας κατακερματισμού εφαρμόζεται πρώτος και η συνάρτηση συμπίεσης εφαρμόζεται δεύτερη στο αποτέλεσμα, δηλαδή: $h(x) = h_2(h_1(x))$

- Ο στόχος της συνάρτησης κατακερματισμού είναι να “διασκορπίσει” τα κλειδιά με ένα φαινομενικά τυχαίο τρόπο για την αποφυγή συγκρούσεων.

collisions

Κώδικες κατακερματισμού

- Η διεύθυνση μνήμης
  memory address
  - Επανερμηνεύουμε τη διεύθυνση μνήμης του αντικειμένου του κλειδιού ως ακέραιο (π.χ. μέσω του τελεστή διεύθυνσης & στην C++)
  - Γενικά καλή προσέγγιση, εκτός των περιπτώσεων όπου τα κλειδιά είναι αριθμοί ή συμβολοσειρές

Κώδικες κατακερματισμού

- Μετατροπή τύπου σε ακέραιο
  integer cast
  - Επανερμηνεύουμε τα bits του κλειδιού σαν ακέραιο
  - Κατάλληλη για κλειδιά μήκους μικρότερου ή ίσου του πλήθους των bits των ακέραιων τύπων (π.χ. byte, short, int, float στην C++)

- Πρόσθεση συνιστωσών
  component sum
  - Χωρίζουμε τα bits του κλειδιού σε τμήματα $a_0, a_1, \ldots, a_{n-1}$ σταθερού μήκους (πχ 16 ή 32 bits) και αθροίζουμε τα τμήματα (αγνοώντας υπερχειλίσεις)

    Αν $k = (a_0, a_1, \ldots, a_{n-1})$ τότε $h_1(k) = \sum_{i=0}^{n-1} a_i$

  - Κατάλληλη για κλειδιά μήκους μεγαλύτερου ή ίσου του πλήθους των bits των ακέραιων τύπων (π.χ. long, double στην C++)

Κώδικες κατακερματισμού

- Πολυωνυμική συσσώρευση
  polynomial accumulation

  - Χωρίζουμε τα bits του κλειδιού σε τμήματα $a_0, a_1, \ldots, a_{n-1}$ σταθερού μήκους (πχ 16 ή 32 bits) και υπολογίζουμε το πολυώνυμο $p(z) = a_0 + a_1 z + \ldots + a_{n-1} z^{n-1}$ σε μία σταθερή τιμή z

    Αν $k = (a_0, a_1, \ldots, a_{n-1})$ τότε $h_1(k) = \sum_{i=0}^{n-1} a_i z^i$

  - Ειδικά κατάλληλο για συμβολοσειρές
    (π.χ. επιλέγοντας $z = 33$ προκύπτουν το πολύ 6 “συγκρούσεις” σε ένα σύνολο 50.000 λέξεων)

  - Το πολυώνυμο $p(z)$ μπορεί να υπολογιστεί σε $O(n)$ χρόνο χρησιμοποιώντας τον κανόνα του Horner ως εξής:

    - $p_0(z) = a_{n-1}$
    - $p_i(z) = a_{n-i-1} + z \cdot p_{i-1}(z), \quad i = 1, 2, \ldots, n-1$
    - Τελικά έχουμε: $p(z) = p_{n-1}(z)$

Συνάρτησεις συμπίεσης

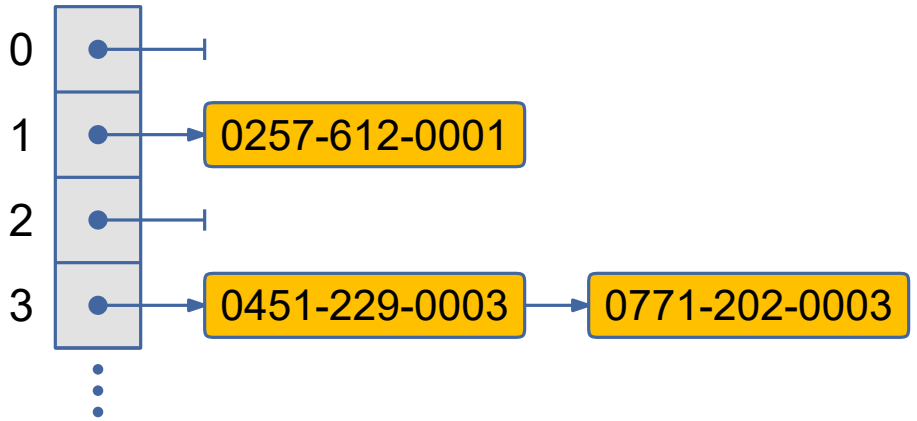
- Διαίρεση
  division
  - $h_2(y) = y \mod N$
  - Το μέγεθος N του πίνακα κατακερματισμού επιλέγεται συνήθως να είναι πρώτος αριθμός

- Πολλαπλασιασμός, Πρόσθεση και Διαίρεση
  MAD - multiplication, addition, division
  - $h_2(y) = (ay + b) \mod N$
  - Οι αριθμοί a και b είναι μη αρνητικοί ακέραιοι, τέτοιοι ώστε: $a \mod N \neq 0$
    (διαφορετικά κάθε ακέραιος y θα αντιστοιχούταν στην ίδια τιμή b)

Διαχείριση συγκρούσεων με χρήση αλυσίδων

collision handling with chaining

- Το κάθε κελί του πίνακα κατακερματισμού δείχνει σε μια συνδεδεμένη λίστα (αλυσίδα) από εγγραφές που αντιστιχούν σε αυτή την τιμή.
- Απλή λύση, αλλά απαιτεί επιπλέον μνήμη πέραν του πίνακα κατακερματισμού.



- Τυπικές υλοποιήσεις (υποθέτοντας ότι η λίστα υποστηρίζει και τη μέθοδο `put`):

```
1 Algorithm find(k)
2 |   return A[h(k)].find(k);
```

```
1 Algorithm put(k,v)
2 |   return A[h(k)].put(k,v);
```

```
1 Algorithm erase(k)
2 |   return A[h(k)].erase(k);
```

Διαχείριση συγκρούσεων με χρήση αλυσίδων: Ανάλυση

collision handling with chaining

- Ομοιόμορφη υπόθεση: Η συνάρτηση κατακερματισμού κατανέμει ομοιόμορφα τα κλειδιά στις διαθέσιμες θέσεις.
- Συντελεστής φόρτου: $\alpha = n/N$
  load factor
- Λήμμα: Σε μία δομή κατακερματισμού με αλυσίδες n στοιχείων:
  - Το μέσο μήκος των αλυσίδων είναι α με πιθανότητα να τείνει στο 1.
  - Μία αναζήτηση (επιτυχημένη ή αποτυχημένη) κοστίζει $O(1+\alpha)$ χρόνο.
- Θεώρημα: Μία ακολουθία από n ενθέσεις, διαγραφές, και αναζητήσεις σε ένα πίνακα κατακερματισμού με χρήση αλυσίδων απαιτεί χρόνο $O(n(1+\alpha/2))$.

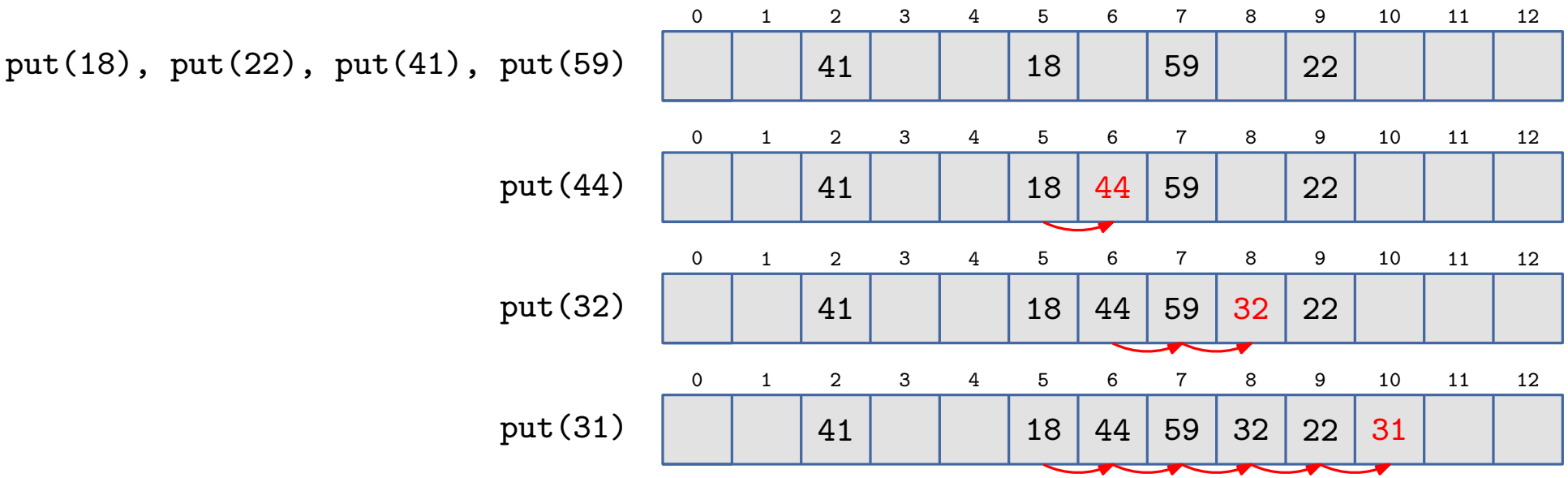
Διαχείριση συγκρούσεων με γραμμική εξέταση

collision handling with linear probing

- Ανοιχτή διευθυνσιοδότηση: Οι συγκρουόμενες εγγραφές τοποθετούνται σε διαφορετικά κελιά του πίνακα κατακερματισμού.
  open addressing

- Γραμμική εξέταση: Η συγκρουόμενη εγγραφή τοποθετείται στο επόμενο (κυκλικά) διαθέσιμο κελί του πίνακα κατακερματισμού.
  linear probing

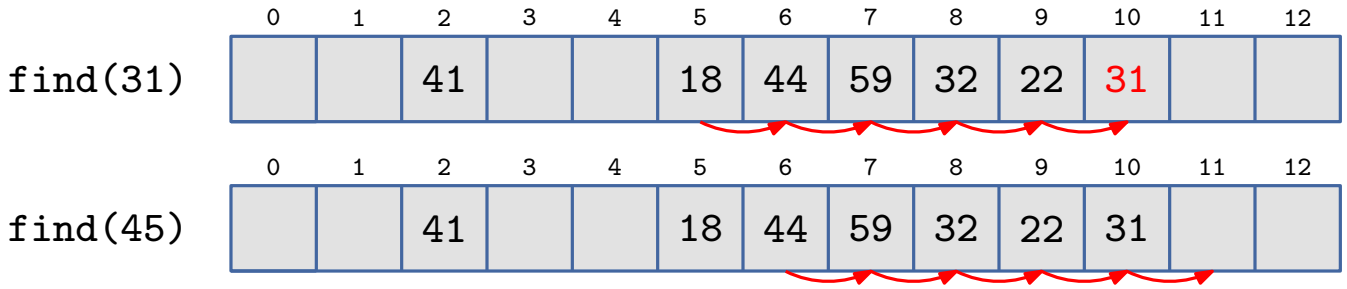
- Παράδειγμα: $h(k) = k \mod 13$



Διαχείριση συγκρούσεων με γραμμική εξέταση: Αναζήτηση

collision handling with linear probing

- Για την αναζήτηση της εγγραφής με κλειδί k:
  - ξεκινάμε την αναζήτηση από το κελί $h(k)$, και
  - εξετάζουμε διαδοχικά τα κελιά του πίνακα κατακερματισμού έως ότου:
    - βρεθεί μία εγγραφή με κλειδί k ή
    - βρεθεί ένα άδειο κελί ή
    - εξεταστούν όλα τα N κελιά
- Παράδειγμα: $h(k) = k \mod 13$



Διαχείριση συγκρούσεων με γραμμική εξέταση: Διαγραφή

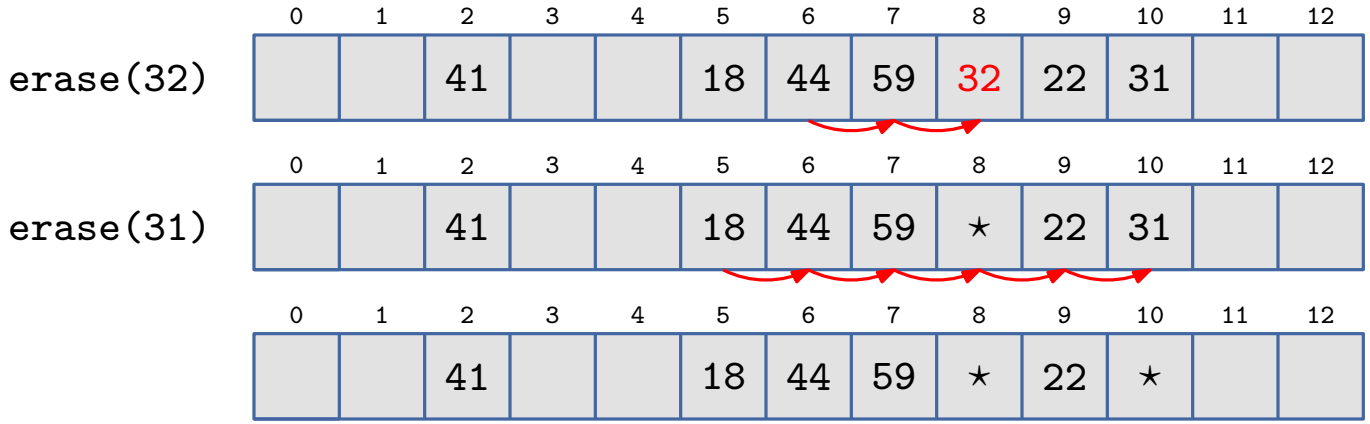
collision handling with linear probing

- Για την διαγραφή της εγγραφής με κλειδί k:
  - Αναζητούμε την εγγραφή με κλειδί k
  - Εάν βρέθει, αντικαθίσταται με την "ανενεργή εγγραφή ($\star$)" και επιστρέφεται η τιμής της v
  - Διαφορετικά, επιστρέφεται `null`

- Ανενεργή θέση: Μια θέση του πίνακα κατακερματισμού που περιέχει την ενενεργή εγγραφή

  defunct position

- Παράδειγμα: $h(k) = k \mod 13$

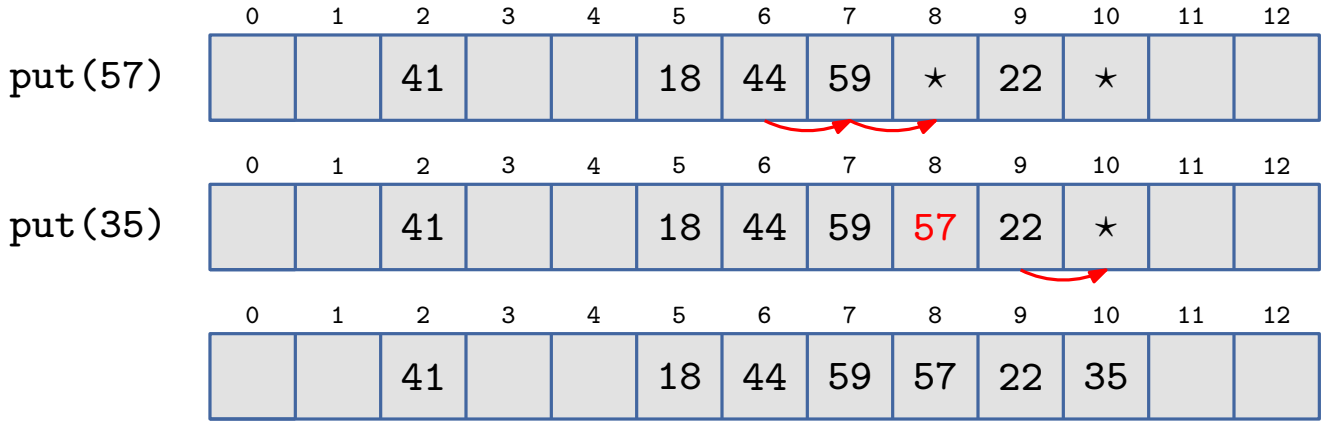


Διαχείριση συγκρούσεων με γραμμική εξέταση: Εισαγωγή

collision handling with linear probing

- Για την εισαγωγή μίας εγγραφής με κλειδί k και τιμή v:
  - Ελέγχουμε αν ο πίνακας κατακερματισμού είναι γεμάτος. Αν είναι, δημιουργείται εξαίρεση (exception)
  - Ξεκινάμε την αναζήτηση άδειας θέσης από το κελί $h(k)$
  - Εξετάζουμε διαδοχικά τα κελιά μέχρι να βρεθεί ένα κελί που να είναι είτε άδειο είτε να περιέχει την ανενεργή εγγραφή ($\star$) και σε αυτό αποθηκεύουμε την εγγραφή (k, v)

- Παράδειγμα: $h(k) = k \mod 13$



Διαχείριση συγκρούσεων με γραμμική εξέταση: Ανάλυση

collision handling with linear probing

- Ομοιόμορφη υπόθεση: Η συνάρτηση κατακερματισμού κατανέμει ομοιόμορφα τα κλειδιά στις διαθέσιμες θέσεις.

- Συντελεστής φόρτου: $\alpha = n/N$
  load factor

- Θεώρημα: Μία επιτυχημένη αναζήτηση σε ένα πίνακα κατακερματισμού με γραμμική διερεύνηση και συντελεστή φόρτου α χρειάζεται, κατά μέσο όρο,

$$\frac{1}{2}\left(1 + \frac{1}{1-\alpha}\right)$$

συγκρίσεις, ενώ μία αποτυχημένη

$$\frac{1}{2}\left(1 + \frac{1}{(1-\alpha)^2}\right)$$

Διαχείριση συγκρούσεων με διπλό κατακερματισμό

collision handling with double hashing

- Η διαχείριση των συγκρούσεων γίνεται χρησιμοποιώντας μια δευτερεύουσα συνάρτηση κατακερματισμού $d(k)$ εισάγοντας την εγγραφή στην πρώτη διαθέσιμη θέση της ακολουθίας:

$$(h(k) + j \cdot d(k)) \mod N, \quad j = 0, 1, \ldots, N-1$$

- Σύνηθης επιλογή: $d(k) = (q - k) \mod q, \quad q < N$

- Ιδιότητες:
  - Η συνάρτηση $d(k)$ δεν μπορεί να έχει μηδενικές τιμές.
  - Για τη διερεύνηση (probing) όλων των θέσεων το μέγεθος N του πίνακα πρέπει να είναι πρώτος.
  - Όμοια, η παράμετρος q της συνήθης επιλογής πρέπει να είναι πρώτος (π.χ. $q = 97$).

Διαχείριση συγκρούσεων με διπλό κατακερματισμό

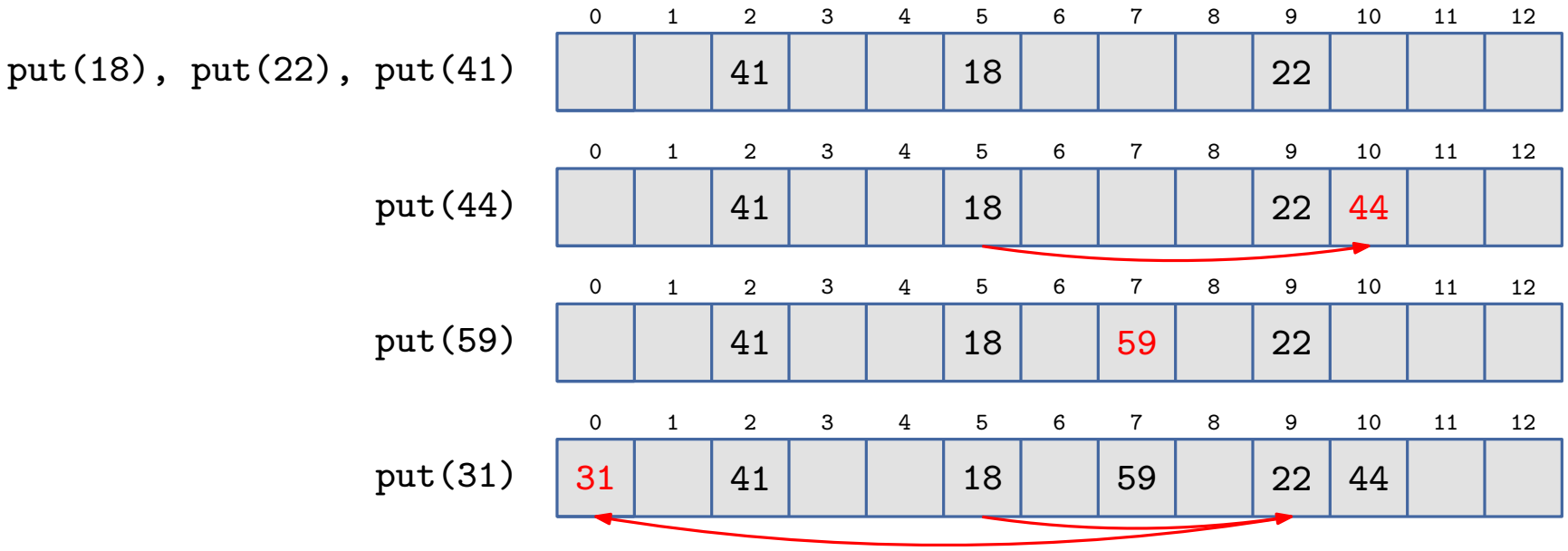
collision handling with double hashing

- Η διαχείριση των συγκρούσεων γίνεται χρησιμοποιώντας μια δευτερεύουσα συνάρτηση κατακερματισμού $d(k)$ εισάγοντας την εγγραφή στην πρώτη διαθέσιμη θέση της ακολουθίας:

$$(h(k) + j \cdot d(k)) \mod N, \quad j = 0, 1, \ldots, N-1$$

- Σύνηθης επιλογή: $d(k) = (q - k) \mod q, \quad q < N$

- Παράδειγμα: $h(k) = k \mod 13, d(k) = (7 - k) \mod 7$



Πίνακες κατακερματισμού: Απόδοση

- Στη χειρότερη περίπτωση, οι πράξεις της αναζήτησης, εισαγωγής και διαγραφής σε ένα πίνακα κατακερματισμού απαιτούν $O(n)$ χρόνο
- Ο συντελεστής φόρτου $\alpha = n/N$ επηρεάζει την απόδοση ενός πίνακα κατακερματισμού
  - Η ιδέα του ανακατακερματισμού: Όταν ο συντελεστής φόρτου, λόγω εισαγωγών (διαγραφών), γινει μεγαλύτερος του $\frac{1}{2}$ (μικρότερος του $\frac{1}{8}$), δεσμεύεται νέος πίνακας διπλάσιου (μισού) μεγέθους και τα στοιχεία ανακατανέμονται στο νέο πίνακα.
- Ο αναμενόμενος χρόνος κάθε πράξης σε ένα πίνακα κατακερματισμού είναι $O(1)$
- Στην πράξη, ο κατακερματισμός είναι πολύ γρήγορος υποθέτοντας ότι ο συντελεστής φόρτου δεν είναι κοντά στο 100%
- Εφαρμογές:
  - Μικρές βάσεις δεδομένων
  - Μεταγλωττιστές (compilers)
  - Προσωρινή αποθήκευση φυλλομετρητή (browser cache)

Επιπλέον υλικό

- Κεφάλαιο 6:
  Michael T. Goodrich, Roberto Tamassia, Αλγόριθμοι σχεδίαση και εφαρμογές
  ISBN: 9789605126971, εκδόσεις Γκιούρδα, 2016.