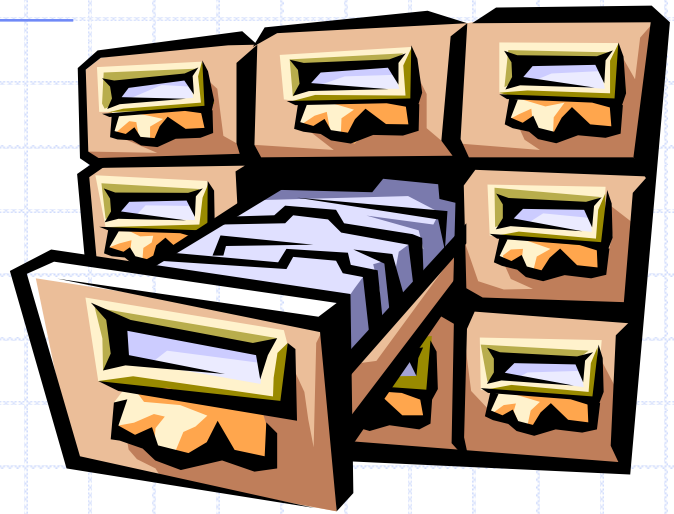
Χάρτες



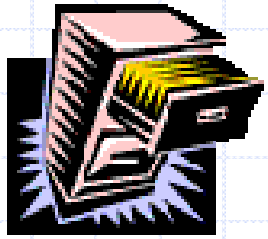
Χάρτες

- Ένας χάρτης μοντελοποιεί μια συλλογή τιμών-κλειδιών με δυνατότητα αναζήτησης
- Οι βασικές πράξεις ενός χάρτη είναι η αναζήτηση, η εισαγωγή, και η διαγραφή στοιχείων
- Δεν επιτρέπονται καταχωρήσεις με το ίδιο κλειδί
- Εφαρμογές:
  - ατζέντα διευθύνσεων
  - βάση δεδομένων πελατών

Λεξικά

- Σε αντίθεση με τους χάρτες τα λεξικά επιτρέπουν καταχωρήσεις με το ίδιο κλειδί.

Ο ΑΤΔ Χάρτη



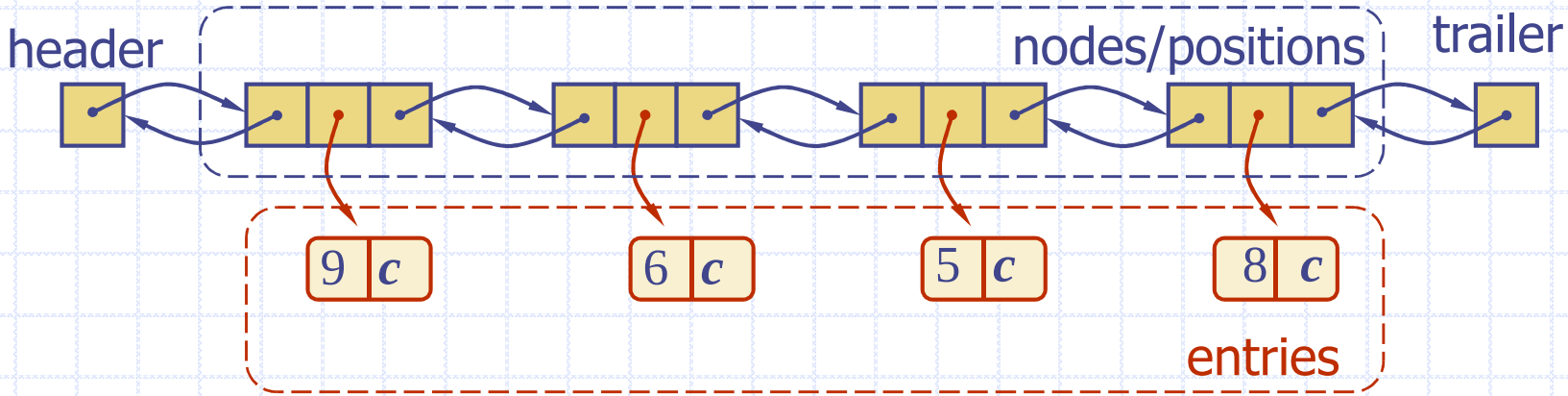
- get(k): αν ο χάρτης M έχει μια καταχώρηση με κλειδί k, επιστρέφεται η αντίστοιχη τιμή· διαφορετικά επιστρέφει την τιμή null
- put(k, v): εισάγεται στο χάρτη M η καταχώρηση (k, v) · αν το κλειδί k δεν είναι ήδη στον M, και επιστρέφει null· διαφορετικά, επιστρέφει την παλαιά τιμή που αντιστοιχεί στο k
- remove(k): αν ο χάρτης M έχει μια καταχώρηση με κλειδί k, την διαγράφει από τον M και επιστρέφει την αντίστοιχη τιμή· διαφορετικά, επιστρέφει null
- size(), isEmpty()
- entrySet(): επιστρέφει μια συλλογή καταχωρήσεων στον M
- keySet(): επιστρέφει μια συλλογή κλειδιών του M
- values(): επιστρέφει έναν επαναλήπτη των τιμών του M

Παράδειγμα

Πράξη	*Έξοδος*	*Χάρτης*
isEmpty()	**true**	Ø
put(5,*A*)	**null**	(5,*A*)
put(7,*B*)	**null**	(5,*A*),(7,*B*)
put(2,*C*)	**null**	(5,*A*),(7,*B*),(2,*C*)
put(8,*D*)	**null**	(5,*A*),(7,*B*),(2,*C*),(8,*D*)
put(2,*E*)	*C*	(5,*A*),(7,*B*),(2,*E*),(8,*D*)
get(7)	*B*	(5,*A*),(7,*B*),(2,*E*),(8,*D*)
get(4)	**null**	(5,*A*),(7,*B*),(2,*E*),(8,*D*)
get(2)	*E*	(5,*A*),(7,*B*),(2,*E*),(8,*D*)
size()	4	(5,*A*),(7,*B*),(2,*E*),(8,*D*)
remove(5)	*A*	(7,*B*),(2,*E*),(8,*D*)
remove(2)	*E*	(7,*B*),(8,*D*)
get(2)	**null**	(7,*B*),(8,*D*)
isEmpty()	**false**	(7,*B*),(8,*D*)

Ένας απλός χάρτης βασισμένος σε λίστα

- Μπορούμε να υλοποιήσουμε αποτελεσματικά έναν χάρτη με χρήση μιας μη ταξινομημένης λίστας
  - Αποθηκεύουμε τα στοιχεία του χάρτη σε μια λίστα S (που βασίζεται σε διπλά συνδεδεμένη λίστα), με αυθαίρετη διάταξη



Ο αλγόριθμος get(k)

```
Algorithm get(k):
    B = S.positions() {Β είναι ένας επαναπλήπτης των
    θέσεων της S}
    while B.hasNext() do
        p = B.next()   { η επόμενη θέση του Β }
        if p.element().getKey() = k        then
            return p.element().getValue()
    return null {δεν υπάρχει καταχώρηση με κλειδί ίσο με
    το k}
```

Ο αλγόριθμος put(k,v)

```
Algorithm put(k,v):
B  = S.positions()
while B.hasNext() do
    p = B.next()
    if p.element().getKey() = k  then
          t = p.element().getValue()
          S.set(p,(k,v))
          return t        {επιστρέφει την παλαιά τιμή}
S.addLast((k,v))
n = n + 1       {καταχωρεί το πλήθος των καταχωρήσεων}
return null     { δεν υπάρχει καταχώρηση με κλειδί ίσο με k }
```

Ο αλγόριθμος remove(k)

```
Algorithm remove(k):
B =S.positions()
while B.hasNext() do
    p = B.next()
    if p.element().getKey() = k  then
        t = p.element().getValue()
        S.remove(p)
        n = n – 1      {μείωση του πλήθους των
    καταχωρήσεων}
        return t       {επιστρέφει την διαγραφείσα τιμή}
return null            {δεν υπάρχει καταχώρηση με κλειδί ίσο
    με  k}
```

Απόδοση ενός χάρτη που βασίζεται σε λίστα

- Απόδοση:
  - put απαιτεί χρόνο $O(1)$ αφού μπορούμε να εισάγουμε το νέο στοιχείο στην αρχή ή το τέλος της ακολουθίας
  - get και remove απαιτούν χρόνο $O(n)$ αφού στη χειρότερη περίπτωση (δεν βρίσκεται το στοιχείο) σαρώνουμε όλη της ακολουθία αναζητώντας ένα στοιχείο με το δοθέν κλειδί
- Η υλοποίηση με μη ταξινομημένη λίστα είναι αποτελεσματική μόνο για μικρούς χάρτες στους οποίους η πιο συνήθης πράξη είναι η εισαγωγή, ενώ σπάνια εκτελούνται αναζητήσεις και διαγραφές (π.χ., ιστορικό προσβάσεων σε σταθμό εργασίας)

Άλλες Προσεγγίσεις Υλοποίησης

- Κατακερματισμός (hashing)
- Πίνακες
- Λίστες Παράλειψης
- Δενδρικές Δομές