

Λεξικά, Binary Search Trees, Hash Tables

**Δημήτρης Φωτάκης, Αριστείδης Παγουρτζής,
Δώρα Σούλιου, Παναγιώτης Γροντάς**

Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών

Εθνικό Μετσόβιο Πολυτεχνείο



ADT Λεξικού

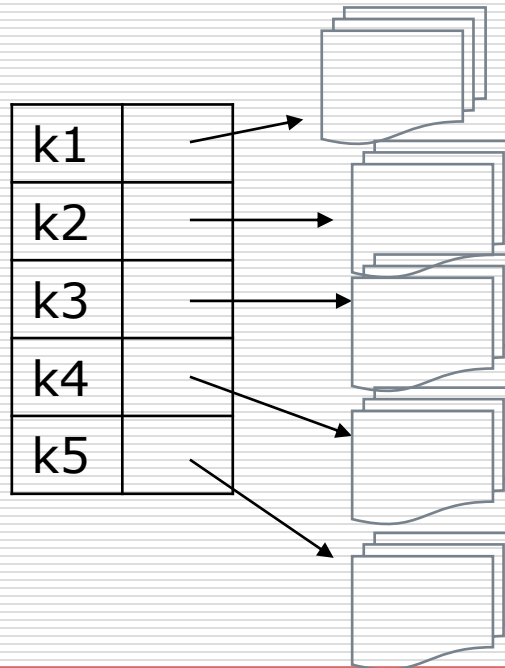
- Δυναμικά μεταβαλλόμενη συλλογή αντικειμένων που αναγνωρίζονται με «κλειδί» (π.χ. κατάλογοι, πίνακες ΒΔ).
- **Λεξικό** : συλλογή αντικειμένων με μοναδικό «κλειδί».
 - «Κλειδί»: αριθμός ή τύπος δεδομένων με ολική διάταξη.
 - Γενίκευση και για μη-μοναδικά κλειδιά.
- ADT λεξικού υποστηρίζει ακολουθίες λειτουργιών:
 - Αναζήτηση στοιχείου με κλειδί k
 - $member(k)$: ελέγχει ύπαρξη στοιχείου με κλειδί k
 - $search(k)$ ή $lookup(k)$: επιστρέφει δείκτη σε θέσεις k
 - Εισαγωγή στοιχείου με κλειδί k
 - $insert(k)$
 - Διαγραφή στοιχείου με κλειδί k
 - $delete(k)$

Λειτουργίες Λεξικού

- Λεξικό μπορεί ακόμη να υποστηρίζει λειτουργίες που σχετίζονται με ταξινόμηση κλειδιών:
 - `outputSorted()`: Εκτύπωση στοιχείων σε αύξουσα / φθίνουσα σειρά
 - `min()`, `max()`: Μέγιστο και ελάχιστο στοιχείο.
 - `predecessor()`, `successor()`: Προηγούμενο και επόμενο στοιχείο.
 - `select(i)`: Επιστρέφει δείκτη στο στοιχείο με το i -οστό μικρότερο κλειδί
 - `rank(i)`: Επιστρέφει πλήθος στοιχείων με κλειδί το πολύ i
- Δεν είναι οι βασικές...
- Δεν υποστηρίζονται από όλες τις δομές δεδομένων λεξικού...

Περί Κλειδιών

- Ασχολούμαστε αποκλειστικά με τα **κλειδιά**
- Όμως αυτά μπορεί να είναι ή να περιέχουν **δείκτες** σε άλλα αντικείμενα



Υλοποιήσεις Λεξικού

- Μη-ταξινομημένη **διασυνδεδεμένη λίστα**:
 - Εισαγωγή: $O(1)$
 - Αναζήτηση / τυχαία διαγραφή: $O(n)$
 - Κατάλληλη όταν **συχνές εισαγωγές**, σπάνιες αναζητήσεις / διαγραφές μεμονωμένες ή στο τέλος (π.χ. log file).
- Ταξινομημένος **πίνακας**:
 - (Διαδική) αναζήτηση: $O(\log n)$
 - min, max: $O(1)$
 - predecessor, successor: $O(\log n)$
 - outputSorted(): $O(n)$
 - select(): $O(1)$
 - rank(): $O(\log n)$
 - **Στατική συλλογή**: «εισαγωγή»/ «διαγραφή» $O(n)$ / στοιχείο
Χρόνος ταξινόμησης : $O(n \log n)$
 - **Κατάλληλη** για **συχνές αναζητήσεις** και δεδομένα μεταβάλλονται σπάνια (π.χ. αγγλο-ελληνικό λεξικό).

Υλοποιήσεις Λεξικού

- (Δυαδικό) Δέντρο Αναζήτησης*:
 - Αναζήτηση / εισαγωγή / διαγραφή: $O(\log n)$
 - Μέγιστο / ελάχιστο / προηγούμενο / επόμενο / k -οστό: $O(\log n)$
 - outputSorted: $O(n)$
 - Range queries σε γραμμικό χρόνο.
 - Πλήρως δυναμική – επιπλέον χώρος για δείκτες!
 - *Balanced
- Hash Table:
 - Αναζήτηση / διαγραφή : $O(1)$
 - Εισαγωγή : $O(1)$ expected amortized, $O(\log n)$ (ακόμη και $O(1)$) whp., $O(n)$ χ.π.
 - Δεν υποστηρίζει αποδοτικά άλλες λειτουργίες.
 - Δυναμική – επιπλέον χώρος στον πίνακα (util $\approx 50\%$)

Εφαρμογές Λεξικού

- Motivating Application: Symbol tables σε πρώτους compilers
- Network filters:
 - Μπλοκάρισμα πακέτων από συγκεκριμένες διευθύνσεις IP (πχ. spammers, DDoS)
- Deduplication
 - Εύρεση & διαγραφή διπλοτύπων
 - Διατήρηση μοναδικού αντιγράφου
 - Πώς βοηθάει το λεξικό;

Κοινό χαρακτηριστικό: Τεράστια ποσά δεδομένων

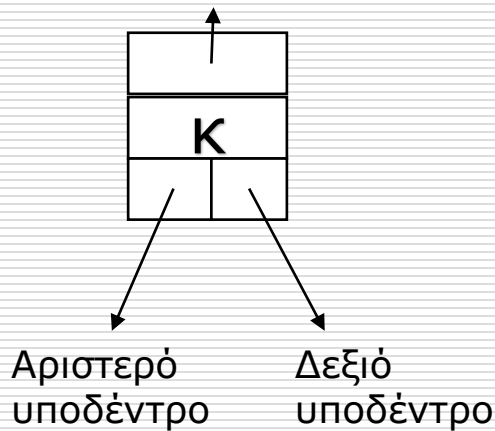
2-SUM Problem

- **Είσοδος:** Δίνεται μη-ταξινομημένος πίνακας n ακεραίων A και ένας ακέραιος t .
- **Έξοδος:** Υπάρχουν ακέραιοι $x, y \in A: x + y = t$?
- **Λύσεις:**
 - $\Theta(n^2)$
 - $O(n \cdot \log n)$
 - $O(n)$

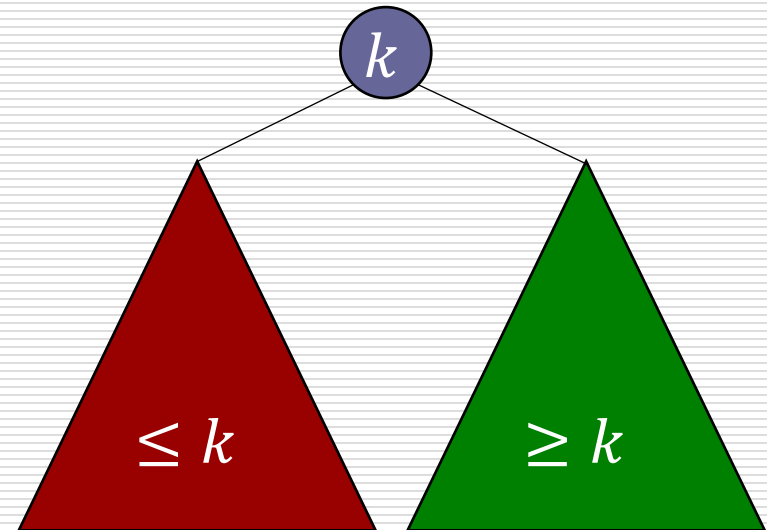
Binary Search Trees

Υλοποίηση - Ιδιότητα BST

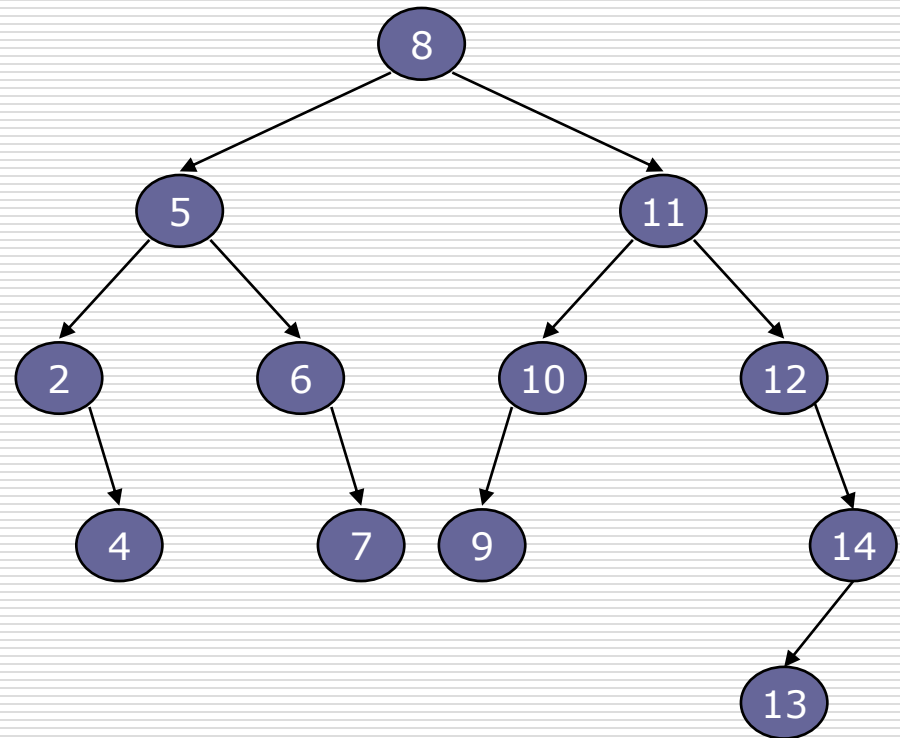
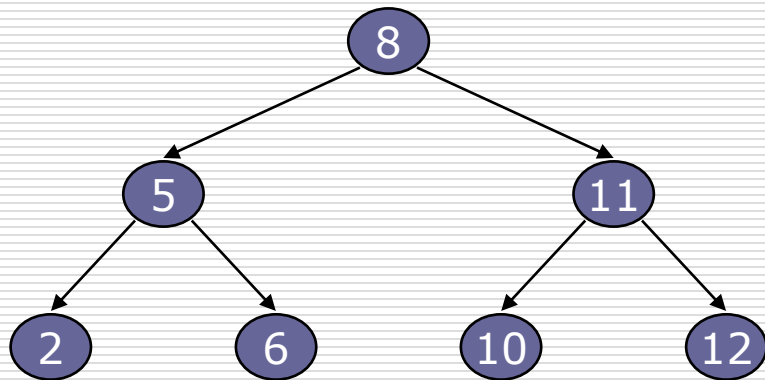
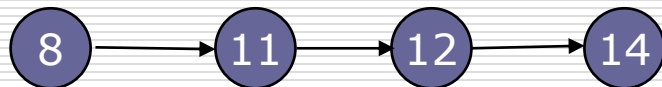
- Κόμβος
 - Κλειδί
 - Γονικός Δείκτης
 - Αριστερός Δείκτης
 - Δεξιός Δείκτης
- Δεν είναι κατ' ανάγκη πλήρη
- Υλοποίηση με πίνακα - **κενά**



- $\forall k: x \in \text{Left}(k) \Rightarrow x \leq k$
- $\forall k: x \in \text{Right}(k) \Rightarrow x \geq k$



Παραδείγματα BST



$$\log_2(n+1) - 1 \leq \text{ύψος} \leq n - 1$$

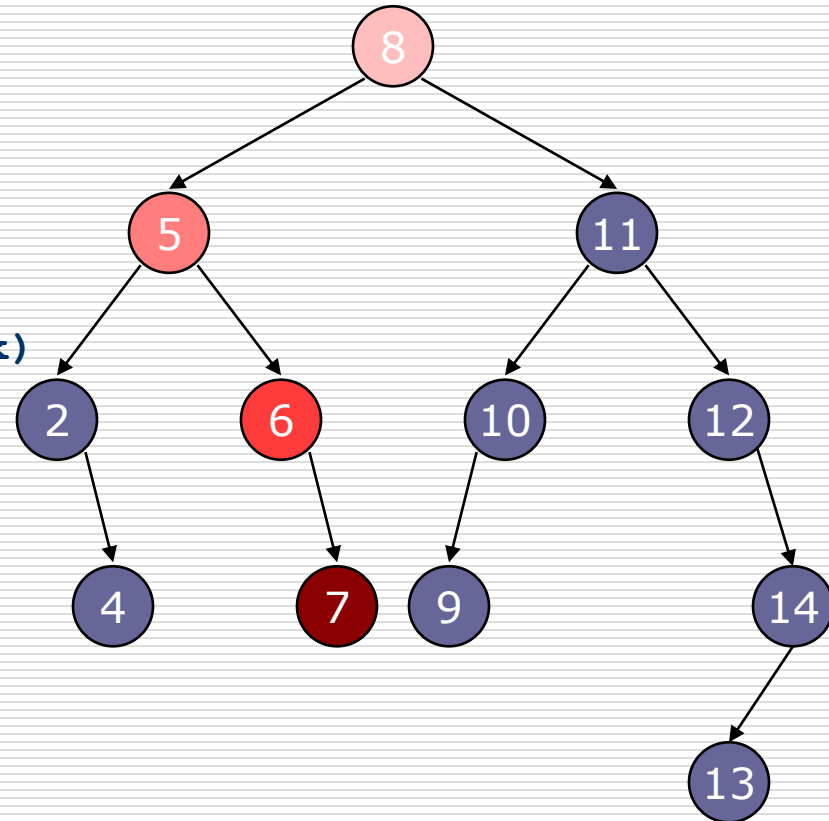
Όλες οι λειτουργίες υλοποιούνται σε: $O(\text{ύψος})$

Αν balanced τότε: $O(\log n)$

Υλοποίηση λειτουργιών – search

```
node search(node r, int k):  
    if (r == null):  
        return null  
    if (r.key == k):  
        return k  
    if (r.key < k):  
        return search(r.left,k)  
    return search(r.right,k)
```

```
node search(node r, int k):  
    x = r  
    while x!=null and x.key!=k:  
        if k < x.key:  
            x = x.left  
        else:  
            x = x.right  
    return null if x==null else x
```

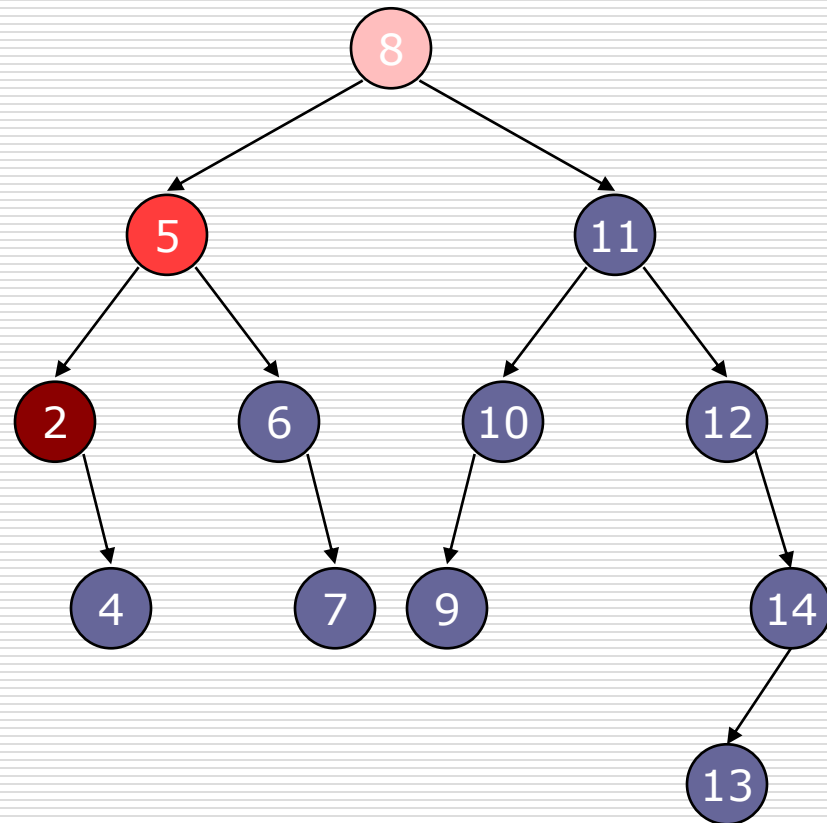


search(8, 7)

Υλοποίηση λειτουργιών – min, max

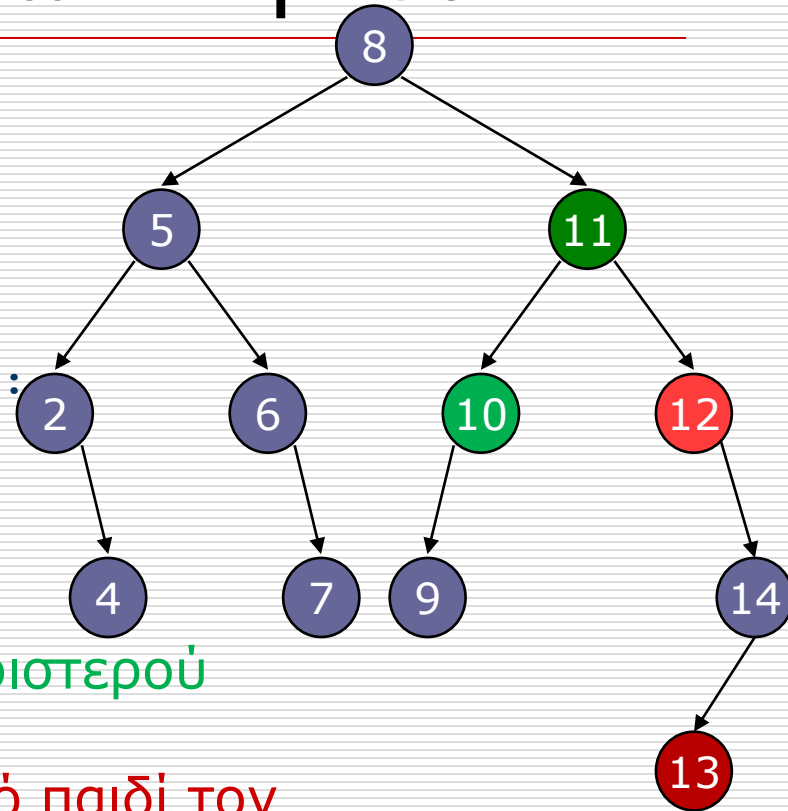
```
node min(node r):  
    while (r.left != null):  
        r=r.left  
    return r.key
```

```
node max(node r):  
    while (n.right != null):  
        r=r.right  
    return r.key
```



Υλοποίηση λειτουργιών – pred

```
node pred(node k):  
    if (k.left != null):  
        return max(k.left)  
    y = k.parent  
    while (y!=null and k == y.left):  
        k=y  
        y=y.parent  
    return y
```



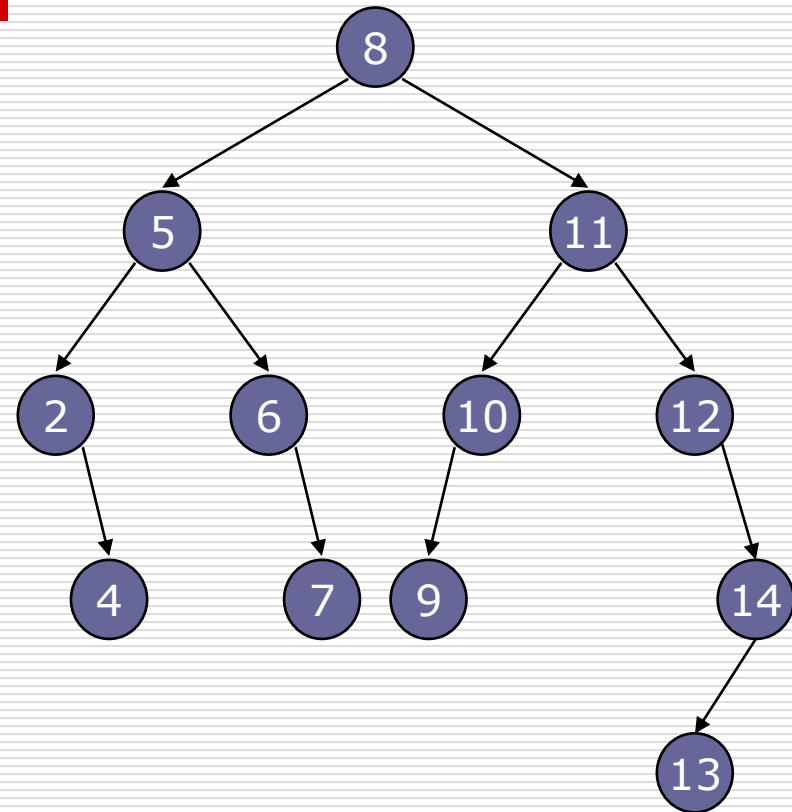
Ο χαμηλότερος δεξιός απόγονος του αριστερού υποδέντρου (max) πχ. $\text{pred}(11)=10$ ή

Ο χαμηλότερος πρόγονος που έχει δεξιό παιδί τον κόμβο πχ. $\text{pred}(13)=12$

Άσκηση: `successor(node k)`

Υλοποίηση Λειτουργιών - output

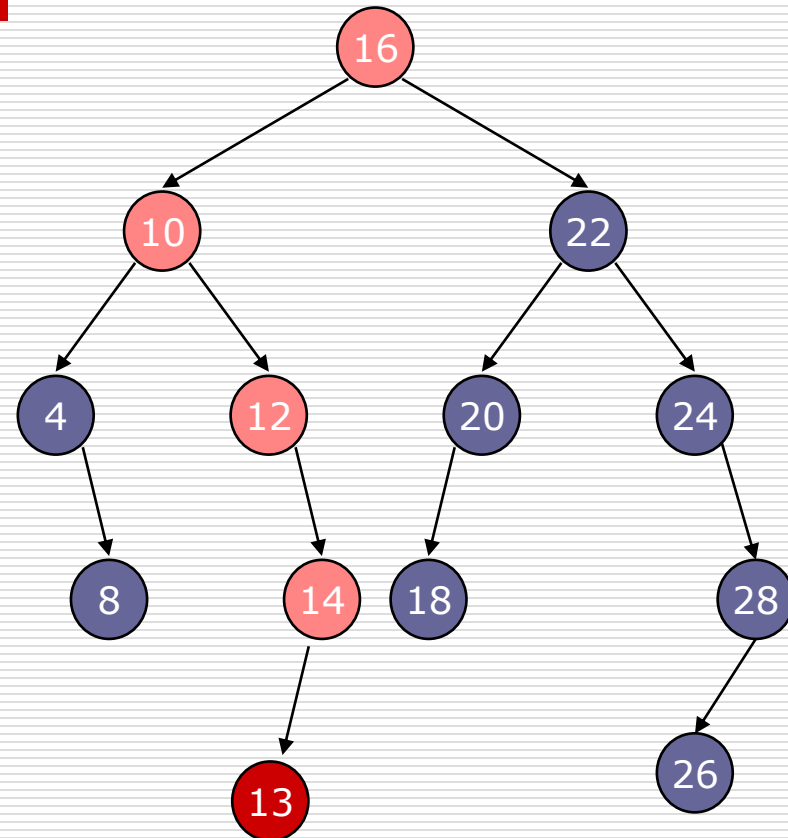
```
outputSorted(node r):  
    outputSorted(r.left)  
    return r.key  
    outputSorted(r.right)
```



$\text{outputsorted}(8) = [2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]$

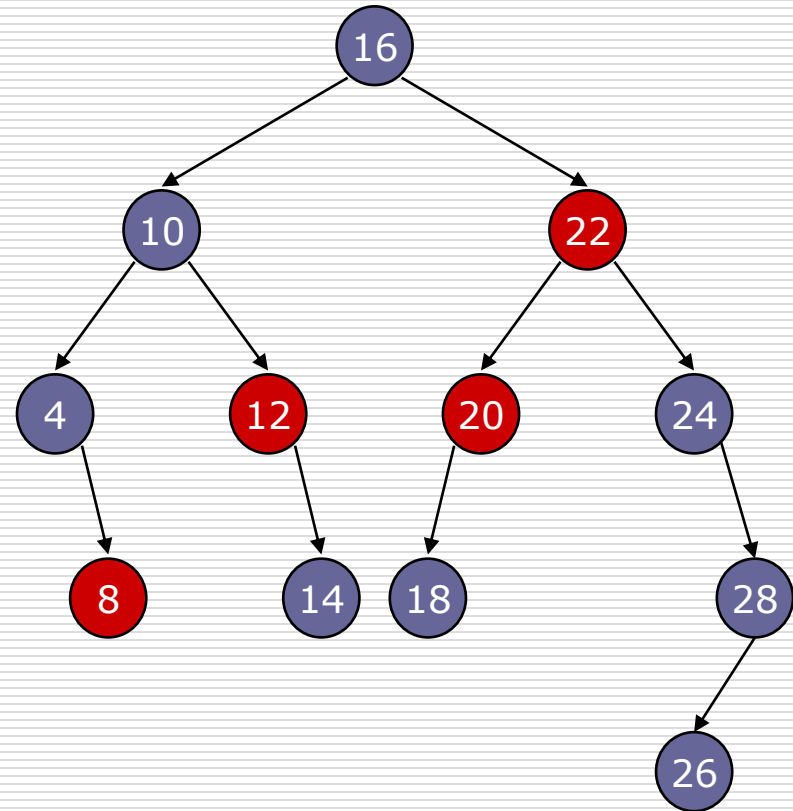
Υλοποίηση Λειτουργιών - insert

```
insert(node r, node k):  
    x=r  
    y=null  
    while (x!=null):  
        y=x  
        if k.key<x.key:  
            x=x.left  
        else:  
            x=x.right  
    k.parent = y  
    if y != null:  
        if k.key < y.key:  
            y.left = k  
        elif k.key < y.key:  
            y.right = k
```



Υλοποίηση Λειτουργιών - delete

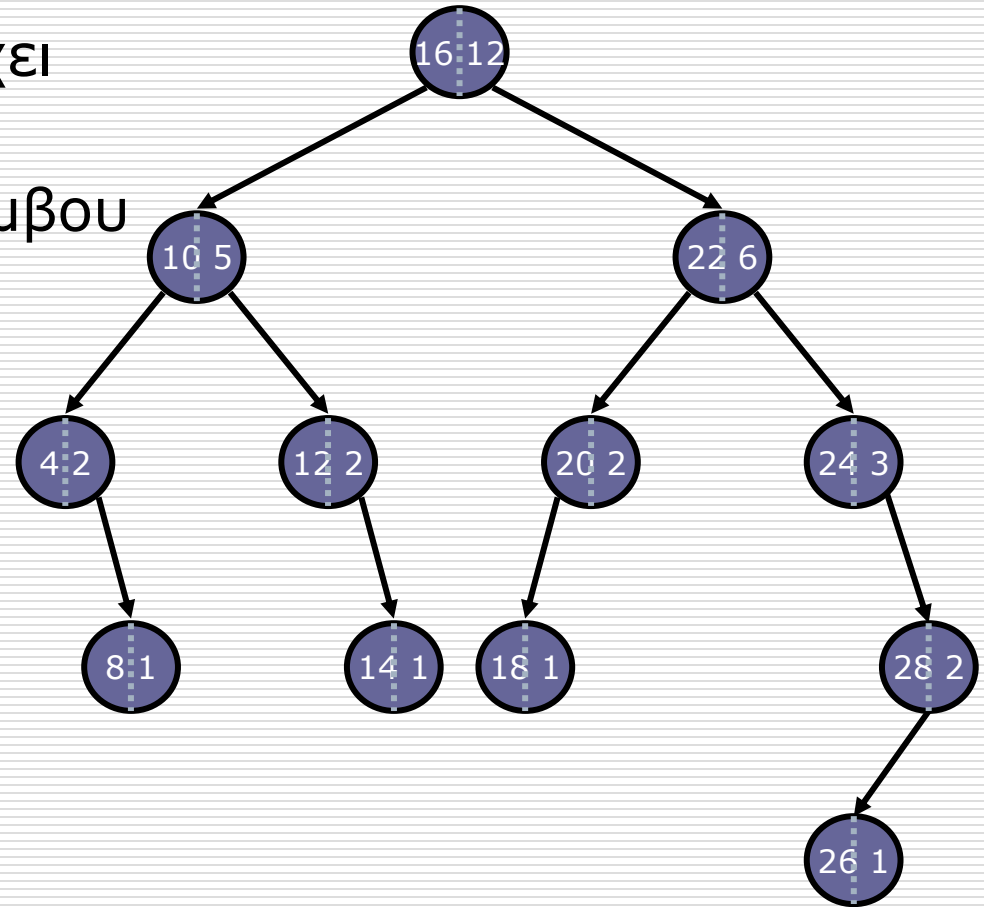
```
node delete(node r, int k):
    if (r == null): return null
    elif k < r.key:
        r.left = delete(r.left, k)
    elif k > r.key:
        r.right = delete(r.right, k)
    else:
        if r.left == null:
            t = r.right
            r = null
            return t
        elif r.right == null:
            t = r.left
            r = null
            return t
        else:
            t = pred(r)
            r.key = t.key
            r.left = delete(r.left, t.key)
    return r
```



Μπορεί και με successor

Order statistics BST

- Νέο πεδίο που περιέχει το μέγεθος του υποδέντρου κάθε κόμβου
- Ποια είναι η σχέση μεταξύ $size(p), size(l), size(r)$;
- $size(p) = size(l) + size(r) + 1$
- **Άσκηση**: Ενημέρωση λειτουργιών delete, insert



Επιλογή (i-οστού μικρότερου) στοιχείου

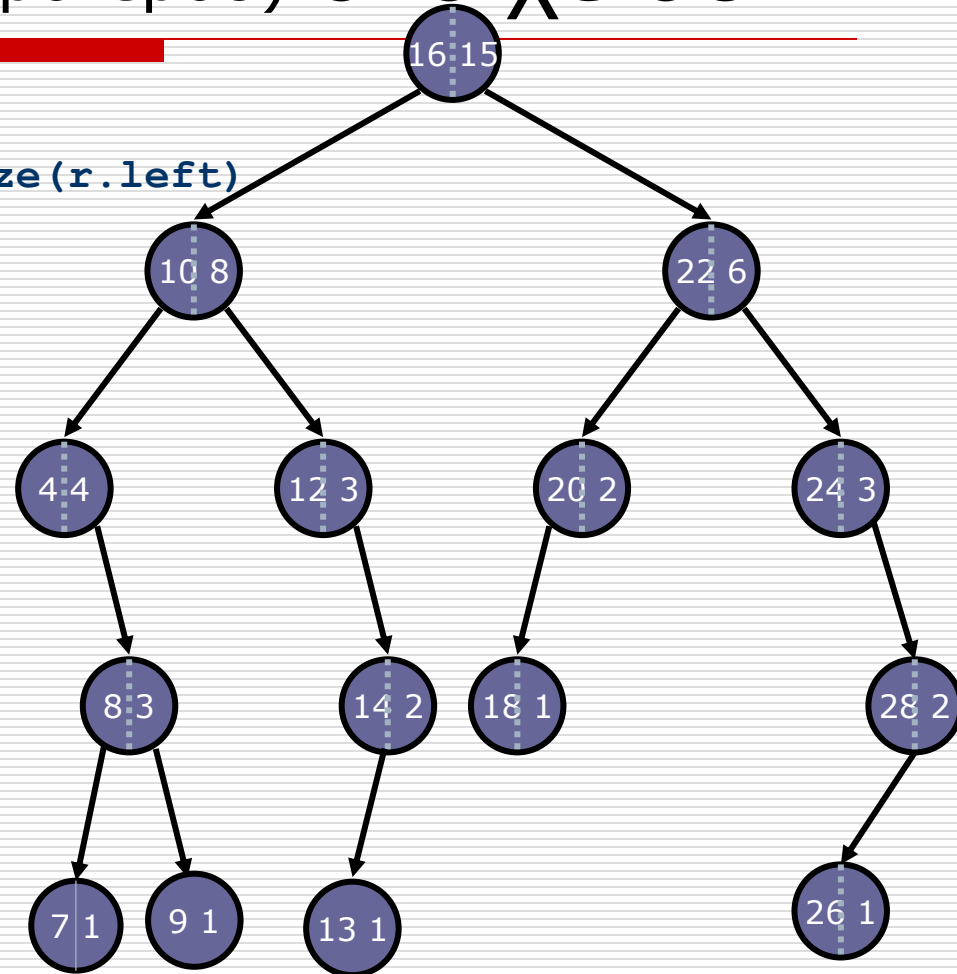
```
node select(node r, int i):  
    l = 0 if r.left == null else size(r.left)  
    if i==l+1:  
        return r.key  
    elif i<l+1:  
        return select(r.left,i)  
    else:  
        return select(r.right,i-l-1)
```

Άσκηση:

rank (node r, int i)

Πόσα στοιχεία είναι το πολύ i

πχ. $rank(\underline{16}, 21) \rightarrow 11$

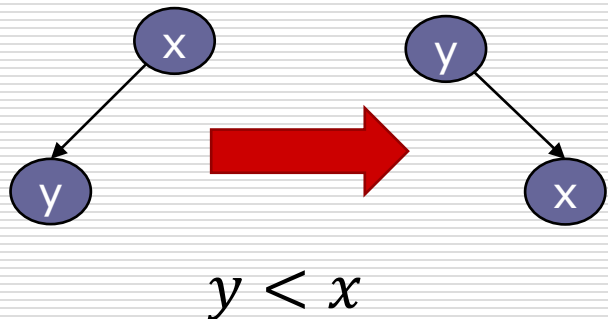


Balancing

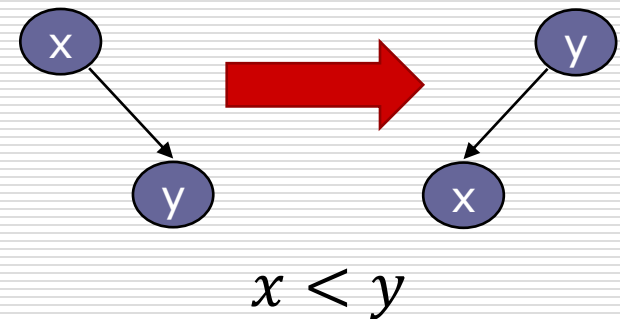
- Επιπλέον δουλειά κατά την εισαγωγή / διαγραφή ώστε το BST να παραμένει ισοζυγισμένο
- ... με αποτέλεσμα: $\text{ύψος} = O(\log n)$
- Βασικό εργαλείο – **Rotations**: μετασχηματισμοί που διατηρούν τη βασική ιδιότητα BST
- Παραλλαγές: 2-3 Trees, AVL Trees

Balancing (2)

Right Rotation

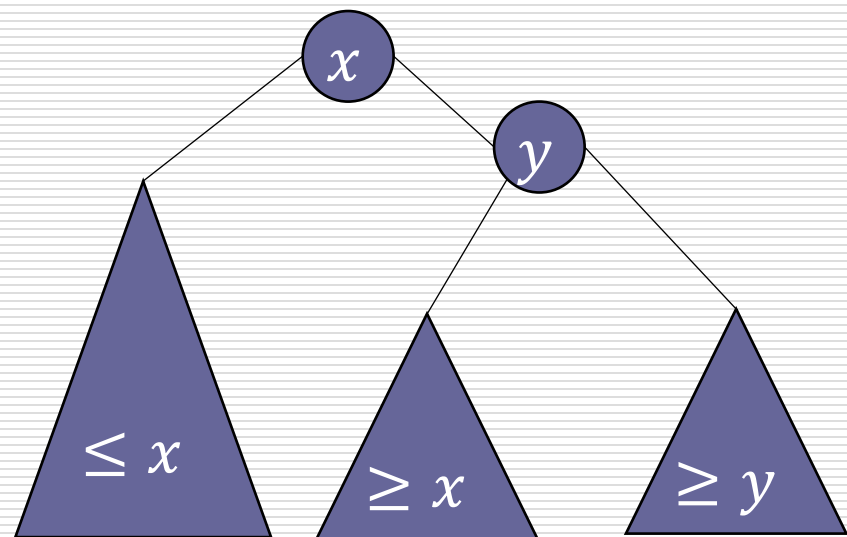
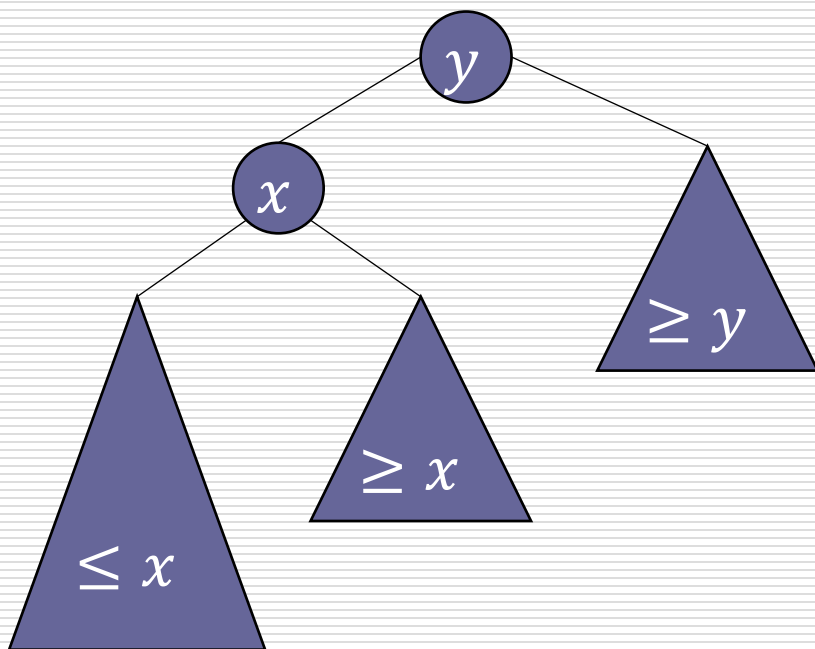


Left Rotation



Διατήρηση ιδιότητας BST

Balancing (3)



Hash Tables

Hash Tables

- Γενίκευση των arrays
- Δηλαδή:
 - Random access $O(1)$
 - Χωρίς ακέραιους δείκτες
 - Με δυναμική αλλαγή μεγέθους
- Υποστηριζόμενες λειτουργίες
 - $\text{Lookup}(k): O(1)$
 - $\text{Insert}(k): O(1)$
 - $\text{Delete}(k): O(1)$
- Κάνουν λίγα πράγματα αλλά τα κάνουν **εξαιρετικά καλά!**

Κλειδιά

- Δίνεται ένα σύνολο U από πιθανά κλειδιά
 - Όλες οι πιθανές διευθύνσεις IP 2^{128} .
 - Όλα τα πιθανά ονόματα 25 χαρακτήρων (26^{25}).
- Δίνεται ένα σύνολο S από πραγματικά κλειδιά
 - Διευθύνσεις IP επισκεπτών
 - Πραγματικά ονόματα
- $|S| \ll |U|$



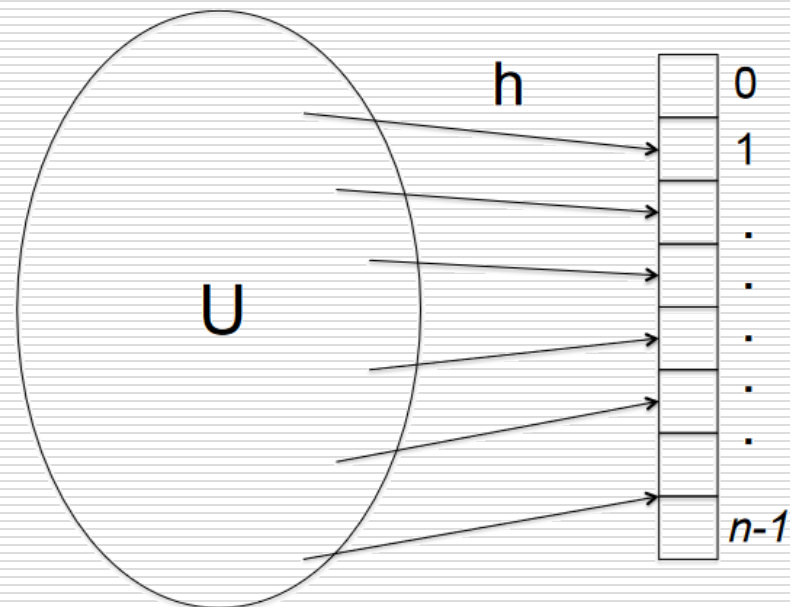
Διαφορές Υλοποίησης

Δομή	Χώρος	Χρόνος Lookup(k)
Πίνακας	$\Theta(U)$	$O(1)$
Γραμμική Λίστα	$\Theta(S)$	$\Theta(S)$
Hash-Table	$\Theta(S)$	$\Theta(1)$

Hash Tables: The best of both worlds

Hash Functions - Collisions

- Μετάφραση στοιχείων του U σε ακέραιους
- $H: U \rightarrow [n]$ με $n = 2|S|$
- Αναζήτηση κλειδιού
 - Υπολογισμός hash
 - Προσπέλαση πίνακα
- Collision
 - $k_1 \neq k_2 \Rightarrow H(k_1) = H(k_2)$
- Σίγουρες λόγω:
 - Αρχή Περιστεριώνα
 - $|S| \ll |U|$
- Συχνές:
 - Παράδοξο Γενεθλίων



Collision Resolution - Chaining

- ❑ Χειρισμός σύγκρουσης:
Συνδεδεμένη λίστα
- ❑ **Lookup(k)**:
 - Υπολογισμός hash $h(k)$
 - Αναζήτηση στη λίστα $A[h(k)]$
- ❑ Θέσεις πίνακα: buckets
- ❑ Για λειτουργίες $O(1)$
 - Σταθερό μέγεθος bucket

Παράδειγμα:

$$H(k) = k \bmod 4, U = [15]$$

0	→	4	8	12		
1	→	1	5	9	13	
2	→	2	4	6	10	14
3	→	3	7	11	15	

Collision Resolution – Probing (Open Addressing)

- Σε κάθε θέση του πίνακα υπάρχουν 0/1 κλειδιά (όχι λίστα)
- Κάθε κλειδί αντιστοιχεί σε ένα σύνολο πιθανών θέσεων (probe positions)
- Εισαγωγή:
 - Υπολογισμός $h(k)$
 - Αναζήτηση μέχρι να βρεθεί κενή θέση σε κάποιες από τις πιθανές θέσεις του συνόλου $A[h(k)]$

Εναλλακτικά Σχήματα Probing

□ Linear Probing

- Υπολογισμός $h(k)$
- $\{h(k), h(k) + 1, h(k) + 2, \dots\} \bmod n$
- Χρήση probe function $\text{pos}(k, i) = h(k) + f(i) \bmod n$

□ Double Hashing

- $h(k) = h_1(k) + i \cdot h_2(k)$
- $h_1(k)$: Αρχική θέση
- $h_2(k)$: Μετατόπιση (σε περίπτωση κατειλημμένης θέσης)
- Παράδειγμα:
 - $h_1(k) \rightarrow 20$
 - $h_2(k) \rightarrow 43$
 - Probing sequence: $\{20, 63, 106, \dots\}$

Ποιότητα hash functions

- Θα διαλέγατε ποτέ την συνάρτηση $h(k) \rightarrow 0, \forall k \in U$;
- Όμως λόγω γενίκευσης αρχής περιστεριώννα:
 - $\forall h: U \rightarrow [n], \exists S \subset U: h(k_1) = h(k_2), \forall k_1, k_2 \in S$
- Όλες οι λειτουργίες του Hash Table είναι $O(1)$ υπό την υπόθεση μιας καλής hash function.
- Μήπως να διαλέξουμε μια τυχαία συνάρτηση;
 - $\Pr[h(k) = h_0] = \frac{1}{n} \forall k \in |U|$
 - Random oracle - Independent Uniform Hashing
 - Δύσκολο στην αποθήκευση και αποτίμηση
- Απαιτούμενα χαρακτηριστικά h:
 - Αποτίμηση σε $O(1)$
 - Αποθήκευση $O(1)$
 - 'Μοιάζει' με τυχαία συνάρτηση (*ομοιόμορφη κατανομή κλειδιών*)

Εκτίμηση Ποιότητας

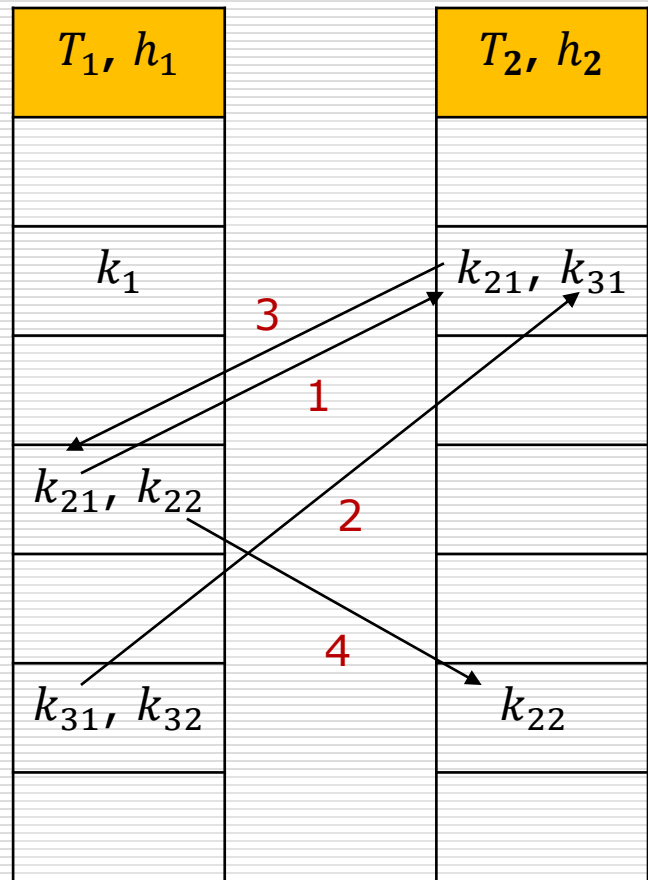
- Φόρτος hash table
 - $\alpha = \text{πλήθος_αποθηκευμένων_κλειδιών} / \text{πλήθος_θέσεων}$
- Εκτίμηση χειρότερης περίπτωσης lookup
 - Chaining: $O(1 + \alpha)$
 - α - αναμενόμενο μέγεθος bucket $\alpha \geq 1$
 - Open Addressing: $O(\frac{1}{1-\alpha})$
 - α - πιθανότητα hash σε κατειλημμένη θέση $\alpha \leq 1$
 - $1 + \alpha + \alpha^2 + \dots = \frac{1}{1-\alpha}$

Πρακτικά θέματα

- Table – Resizing (Rehashing)
 - Όταν $a > \text{threshold}$ αλλαγή μεγέθους πίνακα
 - Συνήθως διπλασιάζουμε το μέγεθος
 - Ξαναυπολογίζουμε τα hashes
- Cryptographic hash functions
 - (Υπολογιστική) Δυσκολία εύρεσης συγκρούσεων
 - Δεν είναι $O(1)$ η αποτίμηση
 - Όμως είναι πολύ γρήγορη (υποστήριξη και από υλικό)

Cuckoo Hashing (Pagh, Rodle 2001)

- Χρήση δύο συναρτήσεων h_1, h_2
- ... και δύο πινάκων T_1, T_2
- Χρήση $h_1 \rightarrow$ τοποθέτηση σε T_1
- Σύγκρουση
 - Νέο κλειδί: $T_1[h_1(x)]$
 - Παλιό κλειδί: $T_2[h_2(x)]$
- Σε νέα σύγκρουση στον T_2
 - Νέο κλειδί: $T_2[h_2(x)]$
 - Παλιό κλειδί: $T_1[h_1(x)]$
 - Νέα σύγκρουση στον T_1
 - Τη χειριζόμαστε κανονικά (διώχνουμε παλιό)
- Κύκλος: rehash



Cuckoo Hashing (με κώδικα)

lookup(k) :

```
return T_1[h_1(k)] or  
       T_2[h_2(k)]
```

O(1)

delete(k) :

```
T_1[h_1(k)] = null  
or  
T_2[h_2(k)] = null
```

O(1)

insert(k) :

```
for I in range(max):  
  
    swap(k, T_1[h_1(k)])  
    if k==null return  
    swap(k, T_2[h_2(k)])  
    if k==null return  
  
rehash()  
insert(k)
```

expected O(n) για n εισαγωγές

Πιθανοτικά Hash Tables: Bloom Filters

- Lookup(k) \in {False, True} (member(k))
- **False Positive: Υπάρχουν σε μικρό βαθμό.**
 - Αν δεις True, δεν είναι βέβαιο ότι υπάρχει αντικείμενο με κλειδί k .
 - Πιθανότητα λάθους $< 1\%$.
- **False Negative: Δεν υπάρχουν.**
 - Αν δεις False ξέρεις σίγουρα ότι δεν υπάρχει αντικείμενο με κλειδί k .
- Πλεονεκτήματα
 - Εγγυημένο Constant Time
 - Space efficient
- Υλοποίηση – επηρεάζει error rate

Bloom Filters – Βασική υλοποίηση

```
insert(k):  
    for i in range(m):  
        A[h_i(k)] = 1  
  
bool lookup(k):  
    for i in range(m):  
        if A[h_i(k)] = 0:  
            return False  
    return True
```

- Πίνακας A n-bit (n θέσεων)
- m hash functions (σταθερός αριθμός)
- Στοιχεία του πίνακα
 - $0 \rightarrow 1$ (ποτέ αντίστροφα)