ΕΝΟΤΗΤΑ 5
ΣΥΝΟΛΑ & ΛΕΞΙΚΑ

Σύνολα (Sets)

- ✓ Τα μέλη ενός συνόλου προέρχονται από κάποιο χώρο αντικειμένων/στοιχείων (π.χ., σύνολα αριθμών, λέξεων, ζευγών αποτελούμενα από έναν αριθμό και μια λέξη, κοκ).

- ✓ Αν S είναι ένα σύνολο και x είναι ένα αντικείμενο του χώρου από όπου το S προέρχεται, είτε $x \in S$, ή $x \notin S$.

- ✓ Ένα σύνολο δεν μπορεί να περιέχει το ίδιο στοιχείο 2 ή περισσότερες φορές. Σύνολα που επιτρέπουν πολλαπλά στιγμιότυπα του ίδιου στοιχείου λέγονται πολυ-σύνολα (multi-sets).

Χρησιμότητα

Πολλές εφαρμογές χρησιμοποιούν σύνολα και απαιτούν να είναι δυνατή η απάντηση ερωτήσεων του στυλ «Είναι το $x \in S$»;

- Υπάρχει αυτό το κλειδί στον πίνακα συμβόλων του μεταγλωτιστή;
- Υπάρχει αυτός ο εργαζόμενος στη βάση δεδομένων των εργαζομένων;
- Υπάρχει αυτό το τηλέφωνο στον ηλεκτρονικό τηλεφωνικό κατάλογο;

Χρήσιμες Λειτουργίες στα Σύνολα

- ✓ Member(x,S): επιστρέφει true αν το x είναι μέλος του συνόλου S, διαφορετικά false.
- ✓ Union(S,T): επιστρέφει S $\cup$ T, δηλαδή το σύνολο που αποτελείται από τα στοιχεία εκείνα που είναι μέλη είτε του S είτε του T.
- ✓ Intersection(S,T): επιστρέφει S $\cap$ T, δηλαδή το σύνολο που αποτελείται από τα στοιχεία εκείνα που είναι μέλη και του S και του T.
- ✓ Difference(S,T): Επιστρέφει S \ T, δηλαδή το σύνολο τω στοιχείων που ανήκουν στο S, αλλά δεν ανήκουν στο T.
- ✓ MakeEmptySet(): Επιστρέφει το κενό σύνολο $\varnothing$.
- ✓ IsEmptySet(S): Επιστρέφει true αν S είναι το κενό σύνολο, και false διαφορετικά.
- ✓ Size(S): Επιστρέφει |S|, δηλαδή τον αριθμό των στοιχείων του S.
- ✓ Insert(x,S): Προσθέτει το στοιχείο x στο σύνολο S ή δεν κάνει καμία ενέργεια αν x $\in$ S.
- ✓ Delete(x,S): Διαγράφει το στοιχείο x από το S. Δεν κάνει τίποτα αν x $\notin$ S.
- ✓ Equal(S,T): Επιστρέφει true αν S=T, διαφορετικά false.
- ✓ Iterate(S,F): Εφαρμόζει τη λειτουργία F σε κάθε στοιχείο του S.

Χρήσιμες Λειτουργίες στα Σύνολα - Λεξικά

Για χώρους στοιχείων με την ιδιότητα της γραμμικής διάταξης:

- *Min(S):* επιστρέφει το μικρότερο στοιχείο του S.

Θεωρούμε ότι κάθε στοιχείο του συνόλου είναι ένα ζεύγος <K,I> όπου K είναι ένα κλειδί για το στοιχείο, και I είναι η πληροφορία που συνοδεύει το στοιχείο με κλειδί K.

Η τιμή του κλειδιού ενός στοιχείου είναι μοναδική.

- *LookUp(K,S):* Δεδομένου ενός κλειδιού K, επιστρέφει μια πληροφορία I, τέτοια ώστε <K,I> $\in$ S. Αν δεν υπάρχει στοιχείο με κλειδί K στο S, επιστρέφει Λ.

- *Insert(K,I,S)*
- *Delete(K,S)*

Λεξικά

Ο αφηρημένος τύπος δεδομένων που υποστηρίζει μόνο τις λειτουργίες MakeEmptySet, IsEmptySet, Insert, Delete, και LookUp, λέγεται **λεξικό**.

Κατάλληλες Δομές Δεδομένων για Λεξικά

Λίστες

Αποθήκευση των στοιχείων του λεξικού σε μια λίστα:

- Σειριακή λίστα (δηλαδή πίνακα)
- Δυναμική Λίστα (απλά ή διπλά συνδεδεμένη).

Τα στοιχεία βρίσκονται στη λίστα διατεταγμένα κατά τον τρόπο εισαγωγής τους.

Υλοποίηση Λειτουργιών

Συνδεδεμένη Λίστα

- LookUp(): Θ(n)
- Insert(): Θ(n)
- Delete(): Θ(n)

Πίνακας

- Το μέγιστο μέγεθος του λεξικού πρέπει να είναι γνωστό εξ αρχής
- LookUp, Insert, Delete: *Χρονική Πολυπλοκότητα;*

Αναμενόμενο Κόστος

- Η πιθανότητα p_j η LookUp() να ψάχνει για το στοιχείο x_j είναι η ίδια για κάθε x, δηλαδή αν έχουμε n στοιχεία είναι 1/n. Έστω ότι c_j είναι το κόστος που πληρώνουμε για να βρούμε το στοιχείο x_j, δηλαδή $c_j = j$.
- Το αναμενόμενο κόστος είναι το άθροισμα των $(c_j * p_j)$ για κάθε j.

$$(\sum_{i=1}^{n} i) / n = (n+1) / 2$$

Άρα το αναμενόμενο κόστος είναι επίσης Θ(n).

Διαφορετικές Πιθανότητες για κλειδιά

- $K_1, \ldots, K_n$: τα κλειδιά στο λεξικό σε φθίνουσα διάταξη αναζήτησης από την LookUp().
- $p_1 \geq p_2 \geq \ldots \geq p_n$: πιθανότητα μια LookUp() να ψάχνει για το $K_1, K_2, \ldots, K_n$, αντίστοιχα.

Ο αναμενόμενος χρόνος αναζήτησης ελαχιστοποιείται όταν τα στοιχεία έχουν τη διάταξη $K_1, K_2, \ldots, K_n$ στη λίστα:

$$C_{opt} = \sum_{i=1}^{n} i p_i$$

Γιατί αυτό είναι βέλτιστο;

Ευριστικά

- Η πραγματική κατανομή πιθανότητας συνήθως δεν είναι γνωστή.
- Το λεξικό μπορεί να αλλάζει μέγεθος κατά τη χρήση του.

➢ Η μελέτη πιθανοτικών μοντέλων απέχει αρκετά από το τι γίνεται στην πράξη!

Heuristic "Move-To-Front" (Ευριστικό «Μετακίνησε στην Αρχή»): Μετά από κάθε επιτυχημένη αναζήτηση, μετακίνησε το στοιχείο που βρέθηκε στην αρχή της λίστας.

Πόσο ακριβό είναι αυτό;
Συνδεδεμένη Λίστα – Σειριακή Λίστα.

Το αναμενόμενο κόστος του ευριστικού Move-To-Front είναι το πολύ 2 φορές χειρότερο από εκείνο του βέλτιστου αλγόριθμου.

Heuristic Transpose (Αλληλομετάθεσης):
Αν το στοιχείο που αναζητήθηκε δεν είναι το πρώτο της λίστας, μετακινείται μια θέση προς τα εμπρός και ανταλλάσσεται με το προηγούμενό του στη λίστα.

- Καλύτερο αναμενόμενο κόστος από MoveToFront
- Σταθεροποιείται σε μια σταθερή "καλή" κατάσταση πιο αργά από το MoveToFront.

Διατεταγμένες Λίστες

Υλοποίηση LookUp() ως Δυαδική Αναζήτηση

- Χρήση Πίνακα για αποθήκευση στοιχείων + BinarySearch() για LookUp().

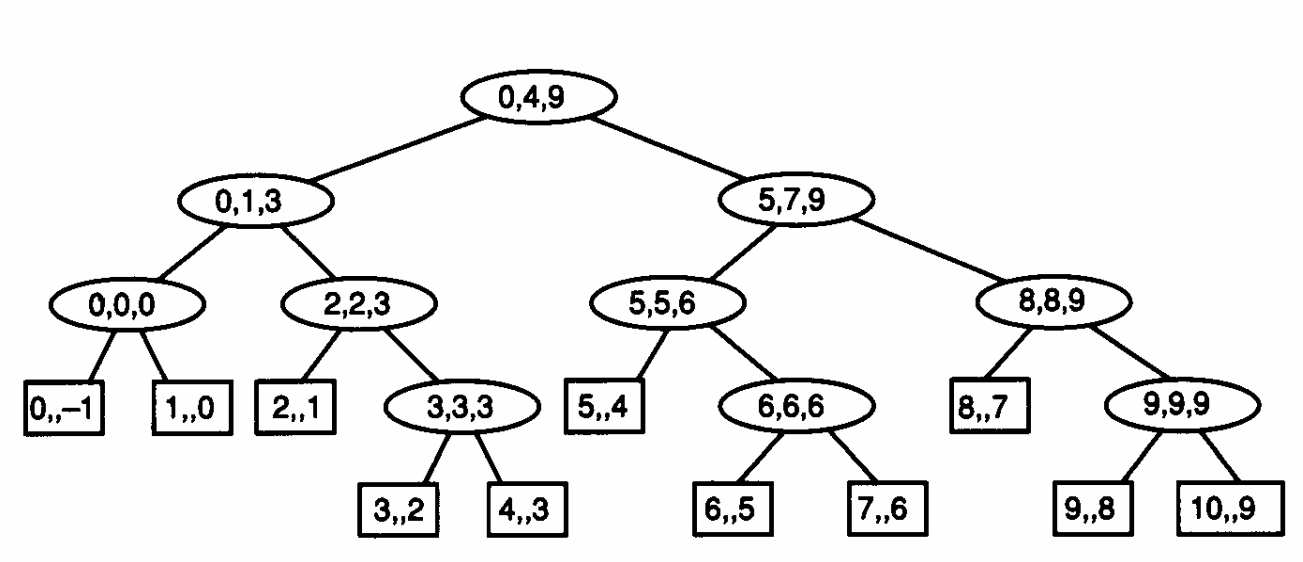
- Τα στοιχεία του πίνακα είναι ταξινομημένα με βάση το Key τους.

BinarySearch - Μη Αναδρομική Έκδοση

function *BinarySearchLookUp*(**key** K, **table** T[0..n-1]):**info**
/* Return information stored with key K in T, or NULL if K is not in T */

```
left = 0;
right = n-1;

repeat forever
    if (right < left) then
        return NULL
    else
        middle = ⌊(left+right)/2⌋;
        if (K == T[middle]->Key) then
            return T(middle]->Info;
        else if (K < T[middle]->Key) then
            right = middle-1;
        else left = middle+1;
```

Δυνατές Εκτελέσεις BinarySearchLookUp



Κυκλικοί κόμβοι: εσωτερικοί
Τετραγωνισμένοι κόμβοι: εξωτερικοί

Θεώρημα 1
Ο αλγόριθμος δυαδικής αναζήτησης εκτελεί O(logn) συγκρίσεις για κάθε αναζήτηση στοιχείου σε πίνακα με n στοιχεία.

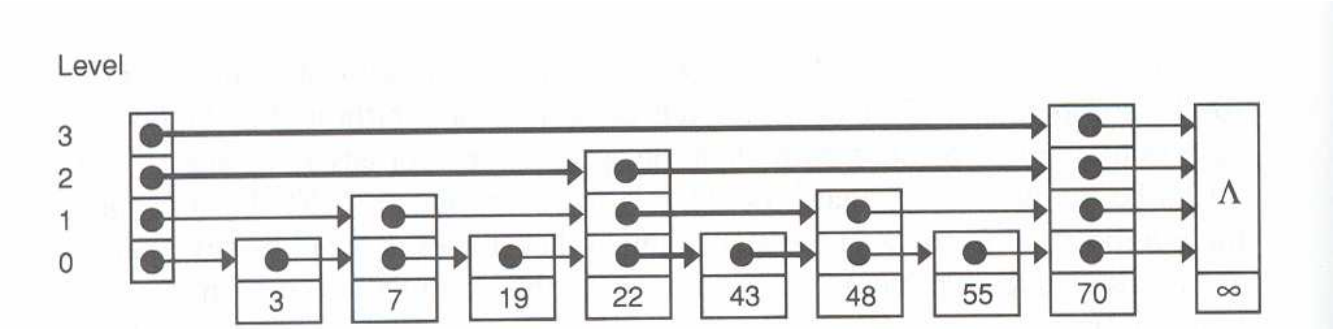
Απόδειξη: Αν h το ύψος του δένδρου, οι εσωτερικοί κόμβοι αποδεικνύεται (επαγωγικά) ότι ανήκουν σε ένα τέλειο δυαδικό δένδρο με ύψος h-1. Αν n είναι οι κόμβοι του τέλειου δένδρου έχουμε δείξει πως $n = 2^h - 1$. Άρα,

μέγιστο βάθος εσωτερικού κόμβου = logn.

Εξωτερικοί κόμβοι;

Skip Lists (Λίστες Παράλειψης)

Είναι δυνατό να εφαρμόσει κάποιος (αποδοτικά) δυαδική αναζήτηση σε μια συνδεδεμένη λίστα;



- Περίπου logn δείκτες ανά στοιχείο χρειάζονται για μια πλήρως συνδεδεμένη λίστα με n στοιχεία.
- Τα περισσότερα στοιχεία δεν χρειάζεται να κρατούν τόσους πολλούς δείκτες:

  Μόνο οι μισοί χρειάζονται έναν ακόμη δείκτη, και από αυτούς μόνο οι μισοί έναν ακόμη, κοκ.
- Οι standard δείκτες της λίστας ονομάζονται δείκτες επιπέδου 0.
- Οι δείκτες που δείχνουν στον 2^i–οστό επόμενο κόμβο λέγονται δείκτες επιπέδου i.
- Ένας κόμβος κεφαλή δείχνει στους αρχικούς κόμβους κάθε επιπέδου.

Skip Lists (Λίστες Παράλειψης)

Αλγόριθμος Αναζήτησης

Ξεκινώντας με τους δείκτες του υψηλότερου επιπέδου:

Ακολούθησε δείκτες μέχρι να βρεθεί ένα στοιχείο με κλειδί > ή ίσο Κ.
Αν ίσο επέστρεψε με επιτυχία.
Αν μεγαλύτερο, οπισθοδρομούμε κατά ένα δείκτη και επαναλαμβάνουμε ακολουθώντας δείκτες του επόμενου χαμηλότερου επιπέδου.
Αν ένας δείκτης επιπέδου 0 οδηγεί σε κλειδί > Κ, επέστρεψε ανεπιτυχώς.

Πως γίνεται η οπισθοδρόμηση;

Σε μια τέλεια οργανωμένη λίστα παράλειψης, ποιά θα ήταν η πολυπλοκότητα του αλγορίθμου;

Είναι οι τέλεια οργανωμένες λίστες παράλειψης πρακτικές στην περίπτωση που έχουμε εισαγωγές και εξαγωγές στοιχείων;

Skip Lists (Λίστες Παράλειψης)

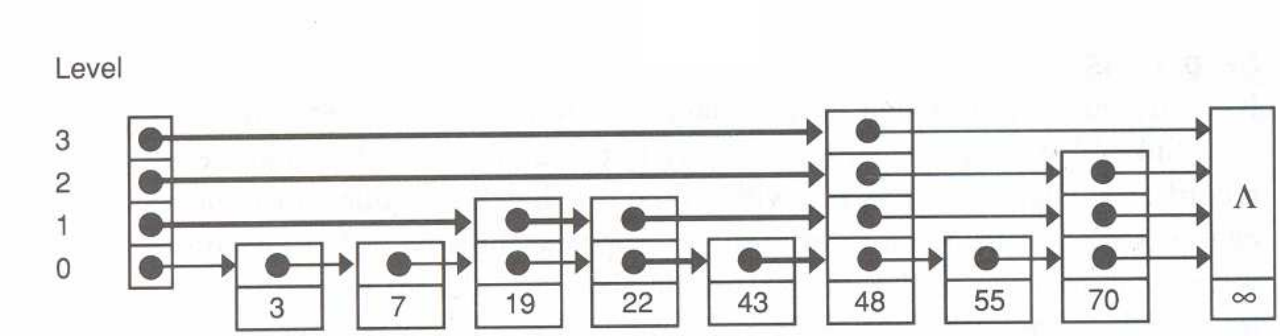
- ✓ Αντί τέλεια οργανωμένων λιστών παράλειψης χρησιμοποιούμε μη-τέλεια οργανωμένες λίστες παράλειψης που τις φτιάχνουμε εισάγοντας κάποια τυχαιότητα στο σύστημα:

- ➢ Ένας κόμβος με (j+1) δείκτες, ένα για κάθε επίπεδο 0, 1, ..., j, ονομάζεται κόμβος επιπέδου j.

Βασική Ιδέα

Οι κόμβοι των διαφόρων επιπέδων εξακολουθούν να υπάρχουν στη λίστα σε «περίπου» ίδια αναλογία, αλλά είναι διασκορπισμένη τυχαία μέσα στη λίστα.

Κόμβοι μεγαλύτερου επιπέδου πρέπει να συναντούνται λιγότερο συχνά.



Skip Lists (Λίστες Παράλειψης)

Με τα νέα μας δεδομένα η εισαγωγή γίνεται αρκετά πιο εύκολη:

- Βρες την κατάλληλη θέση εισαγωγής στη λίστα;
- Επέλεξε το επίπεδου του κόμβου με τυχαίο τρόπο, αλλά με βάση τον κανόνα:

  «Για κάθε επίπεδο j, η πιθανότητα να επιλεγεί το j ως επίπεδου του κόμβου είναι διπλάσια από την πιθανότητα να επιλεγεί το j+1».

✓ N: ο μέγιστος δυνατός αριθμός στοιχείων της λίστας

✓ MaxLevel = $\lfloor \log N \rfloor$ -1: μέγιστο επίπεδο κάθε κόμβου

Κάθε κόμβος μιας λίστας παράλειψης έχει τρία πεδία:

- Key
- Info
- ένα πίνακα Forward από δείκτες.

Η **λίστα παράλειψης** είναι struct με δύο πεδία:

- Header: δείκτης σε dummy node (που περιέχει πίνακα Forward από MaxLevel δείκτες). Ο δείκτης Header->Forward[j] δείχνει στον πρώτο κόμβο του επιπέδου j.
- Level: ακέραιος

Αρχικά, Level = 0, και όλοι οι δείκτες NULL.

Skip Lists (Λίστες Παράλειψης): Εισαγωγή

Αλγόριθμος Αναζήτησης

function *SkipListLookUp*(**key** K, **pointer** L): info
/* επιστρέφει το Info του στοιχείου με κλειδί K στη λίστα παράλειψης αν υπάρχει, διαφορετικά Λ */

```
P = L->Header;
for j from L->Level downto 0 do
      while ((P->Forward[j])->Key < K) do
            P = P->Forward[j];
P = P->Forward[0];
if (P->Key == K) return P->Info;
else return Λ;
```

function *RandomLevel*():**integer**
/* παράγει ένα τυχαίο επίπεδο μεταξύ 0 και MaxLevel */

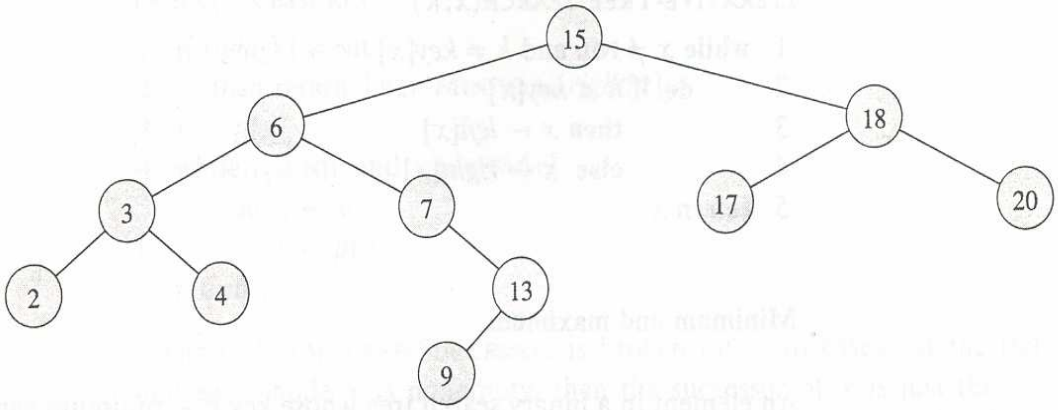
```
v = 0;
while (Random() < ½ && v < MaxLevel) do
      v = v+1;
return v;
```

Skip Lists (Λίστες Παράλειψης): Εισαγωγή

procedure *SkipListInsert*(**key** K, **info** I, **pointer** L):**pointer**

```
/* Εισάγει την πληροφορία I με κλειδί K στην λίστα παράλειψης
                                    L */

P = L->Header;
for j from L->Level downto 0 {
        while ((P->Forward[j])->Key < K) do
                P = P->Forward[j];
        Update[j] = P;
}

P = P->Forward[0];
if (P->Key == K) P->Info = I;
else {
        NewLevel = RandomLevel();
        if (NewLevel > L->Level) {
                for j from L->Level+1 to NewLevel do
                        Update[j] = L->Header;
                L->Level = NewLevel;
        }
        P = NewCell(Node);
        P->Key = K;
        P->Info = I;
        for i from 0 to NewLevel {
             P->Forward[i] = ((Update[i])->Forward)[i]
             ((Update[i])->Forward)[i] = P;
        }
}
```

Ταξινομημένα Δυαδικά Δένδρα
(ή Δυαδικά Δένδρα Αναζήτησης)



Είναι δυαδικά δένδρα που με κάθε κόμβο τους έχει συσχετιστεί μια τιμή από ένα χώρο τιμών στον οποίο είναι ορισμένη μια γραμμική διάταξη. Σε κάθε κόμβο η τιμή είναι μεγαλύτερη από όλες τις τιμές των κόμβων του αριστερού υποδένδρου και μικρότερη από όλες τις τιμές των κόμβων του δεξιού υποδένδρου.

Κάθε κόμβος είναι ένα struct με πεδία Key, data, LC, RC.

Το μέγιστο ύψος δυαδικού δένδρου με n κόμβους είναι n-1.

Το ελάχιστο ύψος δυαδικού δένδρου με n κόμβους είναι logn.

Ποια η σχέση ταξινομημένων δυαδικών δένδρων και ενδο-διατεταγμένης διάσχισης?

Υλοποίηση λειτουργίας LookUp σε Ταξινομημένα Δυαδικά Δένδρα

Αναδρομική Έκδοση

function *BinaryTreeLookUp*(**key** K, **pointer** P):**info**

/* Εύρεση του κλειδιού K στο δένδρο P, με αναδρομική αναζήτηση και επιστροφή του Info πεδίου του ή Λ αν το κλειδί δεν υπάρχει στο δένδρο */

```
if (P == NULL) return Λ;
else if (K == P->Key) return P->data;
else if (K < P->Key)
        return(BinaryTreeLookUp(K, P->LC));
else return(BinaryTreeLookUp(K, P->RC));
```

Μη-Αναδρομική Έκδοση

function *BinaryTreeLookUp*(**key** K, **pointer** P):**info**

```
while (P != NULL) {
        if (K == P->Key) return(P->data);
        else if (K < P->Key) P = P->LC;
        else P = P->RC;
}
return Λ;
```

- *Χρονική πολυπλοκότητα?*
- *Κόμβος Φρουρός?*

Ελάχιστο και Μέγιστο Στοιχείο

function *TreeMinimum*(**pointer** P): **info**
/* P είναι δείκτης στη ρίζα του δένδρου */

```
if (P == NULL) return error;
while (P->LC != NULL) P = P->LC;

return(P->data);
```

function *TreeMaximum*(**pointer** P): **info**
/* P είναι δείκτης στη ρίζα του δένδρου */

```
if (P == NULL) return error;
while (P->RC != NULL) P = P->RC;

return (P->data);
```

Πολυπλοκότητα?

Επόμενο και Προηγούμενο Στοιχείο

Το πρόβλημα είναι ίδιο με την εύρεση του επόμενου και προηγούμενου κόμβου στην ενδο-διατεταγμένη διάσχιση του δένδρου.

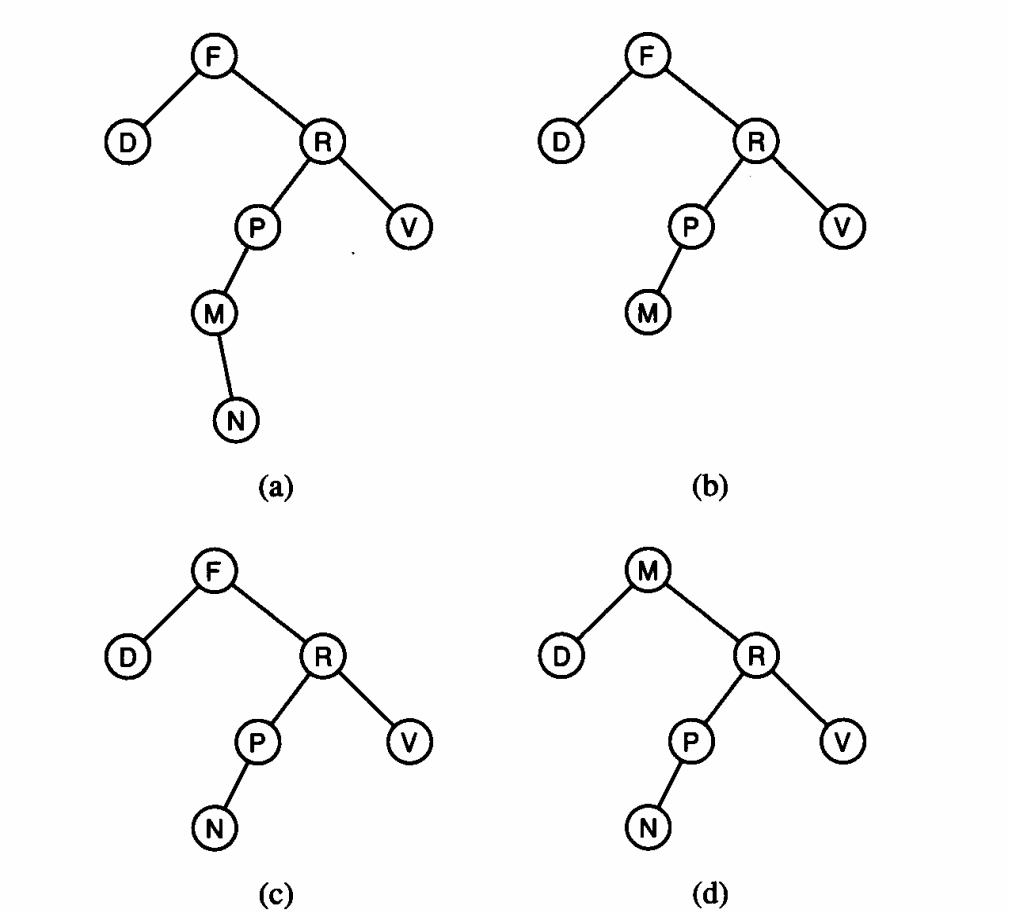
Εισαγωγές σε Ταξινομημένο Δυαδικό Δένδρο

```
function BinSearchTreeInsert(key K, info I,
                             pointer R): pointer
        /* R είναι δείκτης στη ρίζα του δένδρου */

    pointer P, Q, Prev = NULL;

    P = R;
    while (P != NULL) {
        if (P->Key == K) {
            P->data = I;
            return R;
        }
        Prev = P;
        if (K < P->Key) P = P->LC;
        else P = P->RC;
    }

    /* Δημιουργία & προσθήκη νέου κόμβου */
    Q = NewCell(Node);
    Q->Key = K; Q->data = I;
    Q->LC = Q->RC = NULL;

    if (Prev == NULL) return Q;
    else if (K < Prev->Key) Prev->LC = Q;
    else Prev->RC = Q;
    return R;
```

Οι νέοι κόμβοι εισάγονται πάντα σαν παιδιά φύλλων. *Είναι αυτό καλό?*

Διαγραφές: Ταξινομημένο Δυαδικό Δένδρο

Όταν ένας κόμβος διαγράφεται, η ενδο-διατεταγμένη διάσχιση των υπόλοιπων κόμβων πρέπει να «δίνει» τα κλειδιά στη διάταξη που είχαν πριν τη διαγραφή.



Περιπτώσεις:

1) **Ο προς διαγραφή κόμβος είναι φύλλο.** *Απλά τον διαγράφουμε.*

2) **Ο προς διαγραφή κόμβος είναι εσωτερικός αλλά έχει μόνο ένα παιδί.** *Αντικαθιστούμε τον κόμβο με το μοναδικό παιδί του.*

3) **Ο προς διαγραφή κόμβος είναι εσωτερικός με 2 παιδιά.** *Αντικαθιστούμε τον κόμβο με τον επόμενο του στην ενδο-διατεταγμένη διάσχιση.*

Διαγραφή: Ταξινομημένο Δυαδικό Δένδρο

Υποθέτουμε πως το δένδρο είναι διπλά συνδεδεμένο, δηλαδή κάθε κόμβος έχει ένα δείκτη p που δείχνει στο γονικό κόμβο.

function *BinaryTreeDelete*(**pointer** *R, **pointer** z): **pointer**

```
/* Ο R είναι η διεύθυνση ενός δείκτη στη ρίζα του δένδρου */
/* Ο z είναι δείκτης στον προς διαγραφή κόμβο */

if (z->LC == NULL || z->RC == NULL) y = z;
else y = TreeSuccessor(z);

if (y->LC != NULL) x = y->LC;
else x = y->RC;
if (x != NULL) x->p = y->p;
if (y->p == NULL) return x;
else if (y == y->p->LC) y->p->LC = x;
else y->p->RC = x;

if (y != z) z->Key = y->Key;
if y has other fields copy them two;
return y;
```

Διαγραφές σε Ταξινομημένο Δυαδικό Δένδρο

Γιατί ο επόμενος και όχι ο προηγούμενος στην ενδο-διατεταγμένη διάσχιση?

Ποιο πρόβλημα μπορεί να προκύψει με συνεχή αντικατάσταση του κόμβου με τον επόμενό του στην ενδο-διατεταγμένη διάσχιση?

Στατικά Ταξινομημένα Δυαδικά Δένδρα

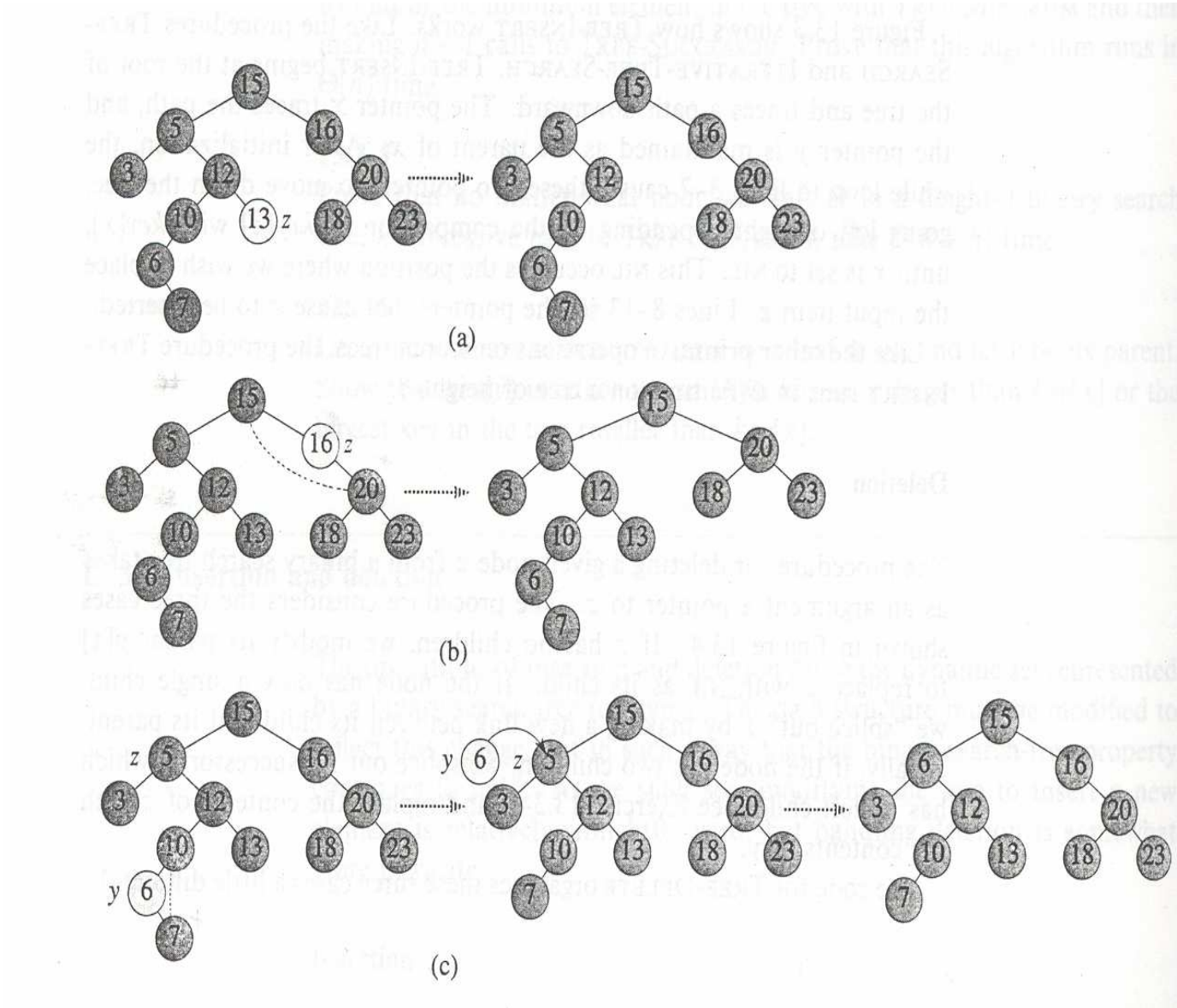
Υπάρχουν κλειδιά που αναζητούνται πιο συχνά και άλλα που αναζητούνται πιο σπάνια.

Σε μια λίστα συχνά ζητούμενα κλειδιά κρατούνται όσο το δυνατόν πιο κοντά στην αρχή της λίστας.

Ποιο είναι το ανάλογο σε ένα ταξινομημένο δυαδικό δένδρο?

Διαγραφή σε Ταξινομημένο Δυαδικό Δένδρο

Παράδειγμα



Ταξινομημένο Δυαδικό Δένδρο Αναζήτησης με Κόμβο Φρουρό