Δομές Δεδομένων και Αλγόριθμοι

Χρήστος Γκόγκος

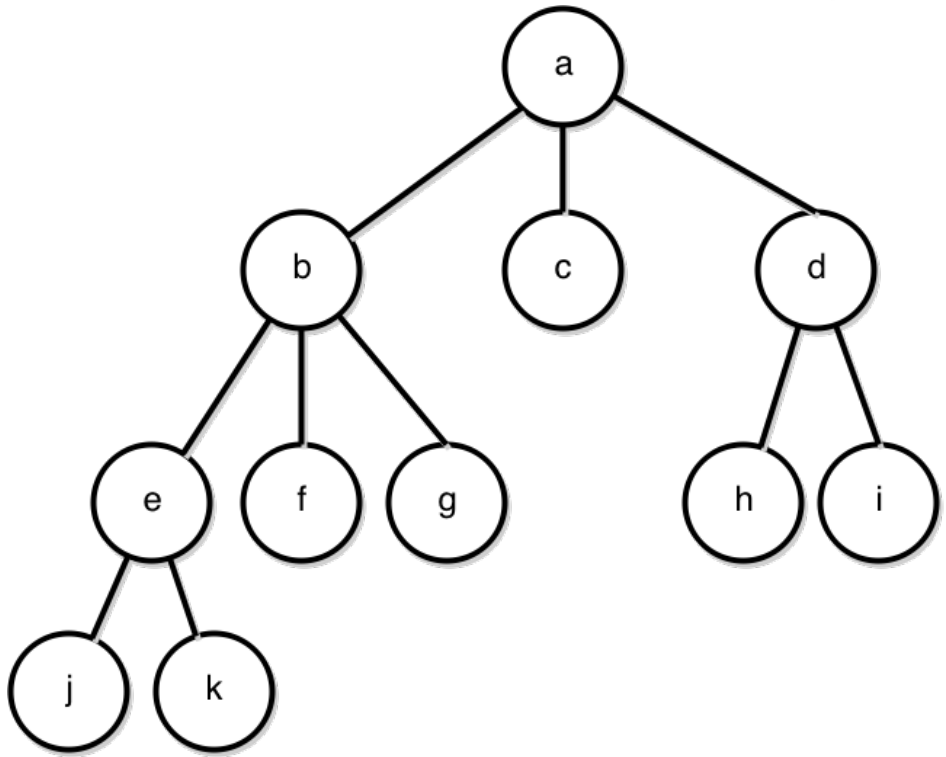
ΤΕΙ Ηπείρου

Χειμερινό Εξάμηνο 2014-2015
Παρουσίαση 16. Δένδρα (Trees)

Δένδρα (Trees)

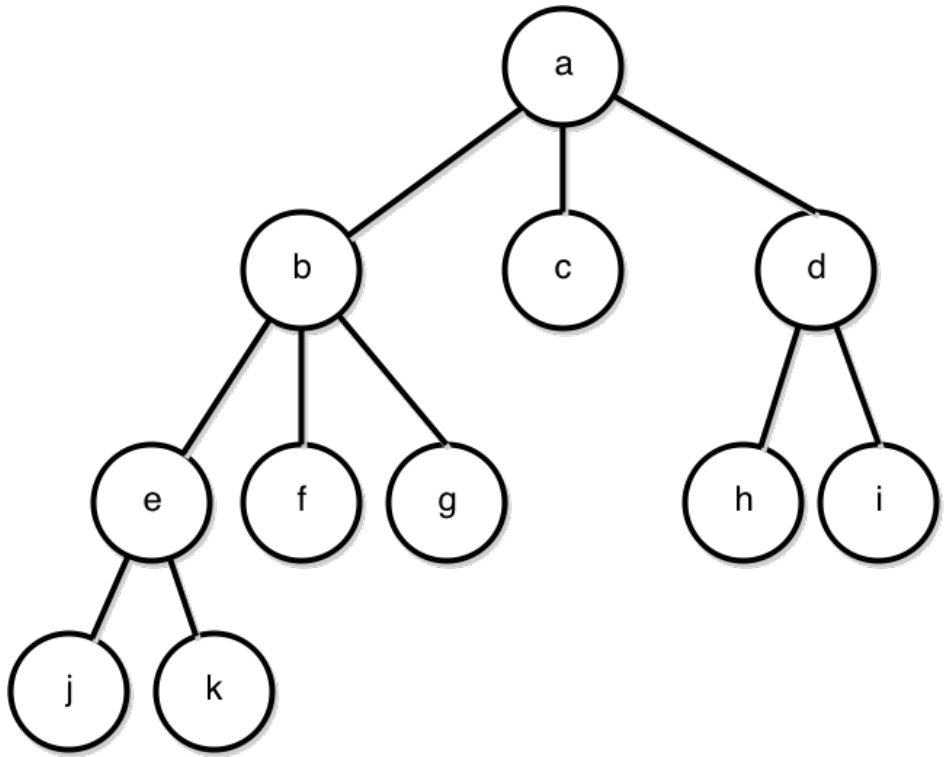
Ένα δένδρο είναι ένα συνδεδεμένο γράφημα χωρίς κύκλους.

- Για κάθε 2 κόμβους ενός δένδρου υπάρχει ακριβώς 1 διαδρομή που οδηγεί από τον ένα κόμβο στον άλλο.
- Το πλήθος των ακμών ενός δένδρου είναι κατά ένα μικρότερο από το πλήθος των κόμβων του (δηλαδή ισχύει ότι $|E| = |V| - 1$).



Όροι που χρησιμοποιούνται σε δένδρα

- Ρίζα (root)
- Υποδένδρα (subtrees)
- Γονέας (parent)
- Παιδί (child)
- Φύλλο (leaf)
- Πρόγονοι (ancestors)
- Απόγονοι (descendants)
- Αδέλφια (siblings)



Το ύψος (height) ενός δένδρου είναι το μήκος του μακρύτερου μονοπατιού από τη ρίζα προς ένα φύλλο του δένδρου.
Το βάθος ενός κόμβου είναι το μήκος του μονοπατιού από τη ρίζα προς τον κόμβο.

Για το παραπάνω δένδρο

Ρίζα είναι το a, το d είναι γονέας των παιδιών h και i που είναι και φύλλα. Το b είναι πρόγονος του j. Το k είναι απόγονος του b. Τα b και d είναι αδέλφια. Το ύψος του δένδρου είναι 3 και το βάθος του e είναι 2.

Εφαρμογές δένδρων

1. Περιγραφή ιεραρχιών
2. Υλοποίηση λεξικών (dictionaries): Ένα λεξικό περιέχει ζεύγη key, value
3. Αποδοτική αποθήκευση μεγάλου όγκου δεδομένων
4. Κωδικοποίηση δεδομένων (π.χ. κωδικοποίηση Huffman)

Διατεταγμένα δένδρα

Στα διατεταγμένα δένδρα (ordered trees) τα παιδιά κάθε κορυφής είναι διατεταγμένα από αριστερά προς τα δεξιά.

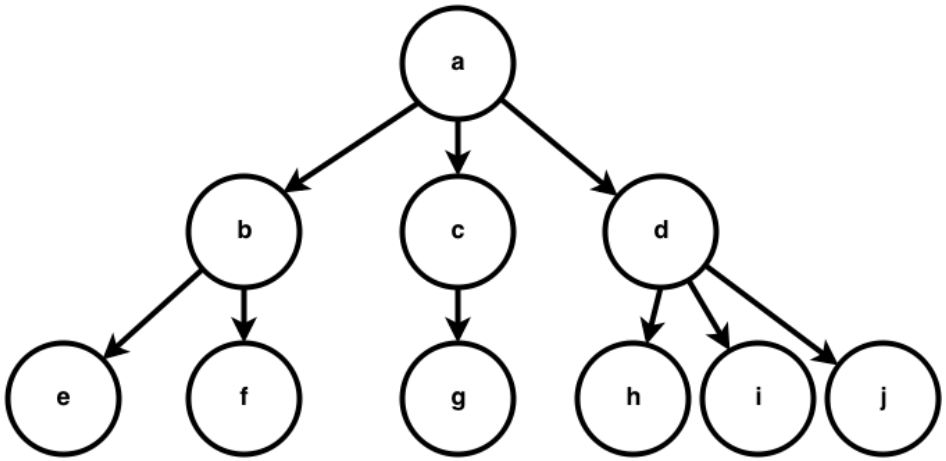
- Δυαδικά δένδρα (Binary Trees). Κάθε κόμβος έχει το πολύ 2 παιδιά.
- Δυαδικά δένδρα αναζήτησης (Binary Search Trees). Δυαδικά δένδρα για τα οποία ισχύει ότι το αριστερό παιδί έχει μικρότερο κλειδί από το γονέα και το δεξιό παιδί έχει μεγαλύτερο κλειδί από το γονέα.
- Δένδρα πολλαπλών δρόμων. Κάθε κόμβος μπορεί να έχει κανένα ένα ή περισσότερα παιδιά.

Μετατροπή ενός δένδρου πολλαπλών δρόμων σε δυαδικό δένδρο

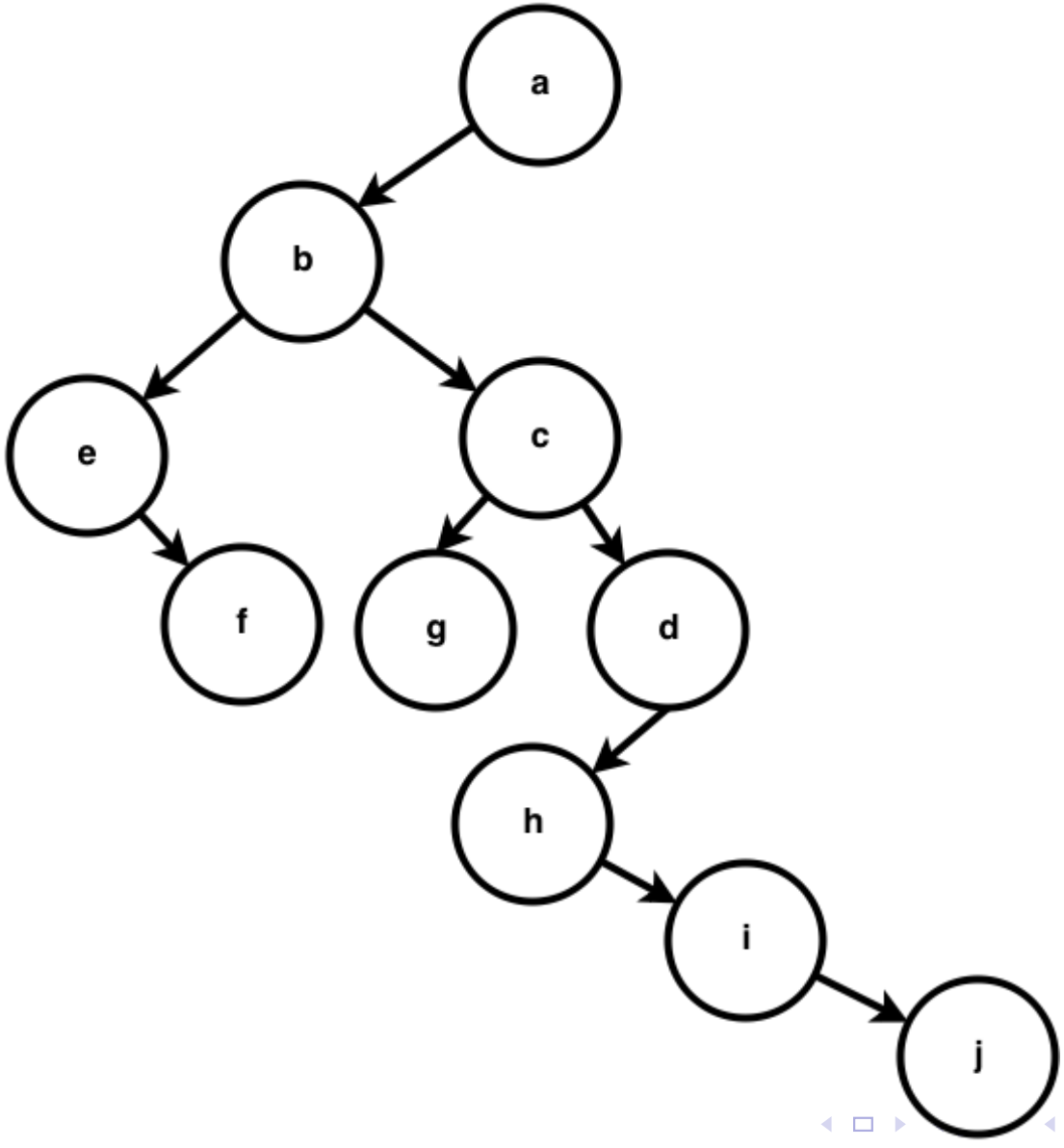
first child-next sibling

Με δεδομένο ένα δένδρο πολλαπλών δρόμων μπορεί να κατασκευαστεί ένα ισοδύναμο δυαδικό δένδρο ακολουθώντας την ακόλουθη διαδικασία που είναι γνωστή ως first child next sibling. Για κάθε κόμβο x με παιδιά του δένδρου πολλαπλών δρόμων:

1. Ο κόμβος x εισάγεται στο δυαδικό δένδρο.
2. Το παιδί του x που βρίσκεται αριστερότερα γίνεται το αριστερό παιδί του κόμβου x στο δυαδικό δένδρο.
3. Κάθε παιδί του x που ακολουθεί γίνεται δεξί παιδί του πλησιέστερου αδερφού του που έχει ήδη τοποθετηθεί στο δυαδικό δένδρο.

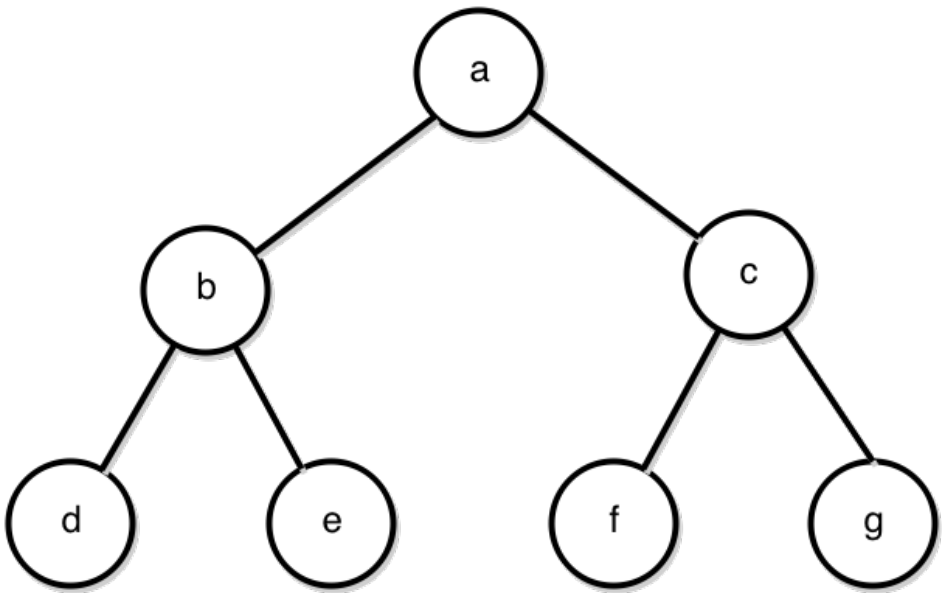


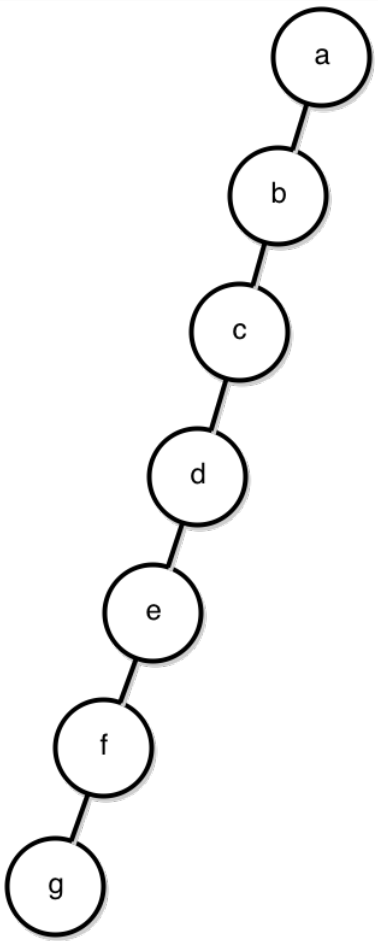
Το δυαδικό δένδρο που προκύπτει από το προηγούμενο δένδρο πολλαπλών δρόμων



Όρια για το ύψος ενός δυαδικού δένδρου

Το ύψος (height) ενός δυαδικού δένδρου είναι:
$\lfloor \log_2 n \rfloor \leq \text{height} \leq n - 1$
όπου n είναι ο αριθμός των κόμβων του δένδρου.





Πλήρες δυαδικό δένδρο - συμπληρωμένο δυαδικό δένδρο

Πλήρες δυαδικο δένδρο

Είναι ένα δυαδικό δένδρο για το οποίο όλοι οι κόμβοι (πλην των φύλλων) έχουν 2 παιδιά.

Συμπληρωμένο δυαδικο δένδρο

Είναι ένα δυαδικό δένδρο που προκύπτει από το πλήρες δένδρο διαγράφοντας κόμβους διανύοντας το δένδρο από κάτω προς τα πάνω και από δεξιά προς τα αριστερά

Υλοποίηση δυαδικού δένδρου

- Στατική αναπαράσταση δυαδικού δένδρου (με χρήση πίνακα). Έχει το μειονέκτημα ότι μπορεί πολλές θέσεις του πίνακα να είναι κενές.
- Συνδεδεμένη αναπαράσταση δυαδικού δένδρου. Κάθε κόμβος αποτελείται από ένα κλειδί και 2 δείκτες που δείχνουν στο αριστερό και στο δεξιό υποδένδρο του. Εναλλακτικά μπορεί να υπάρχει και τρίτος δείκτης ο οποίος δέχνει στο γονέα του κάθε κόμβου.

Συνδεδεμένη αναπαράσταση δυαδικού δένδρου

```
struct node {
    int key;
    struct node* left;
    struct node* right;
};

struct node* new_node(int key) {
    struct node* node = new (struct node);
    node->key = key;
    node->left = NULL;
    node->right = NULL;
    return (node);
}
```

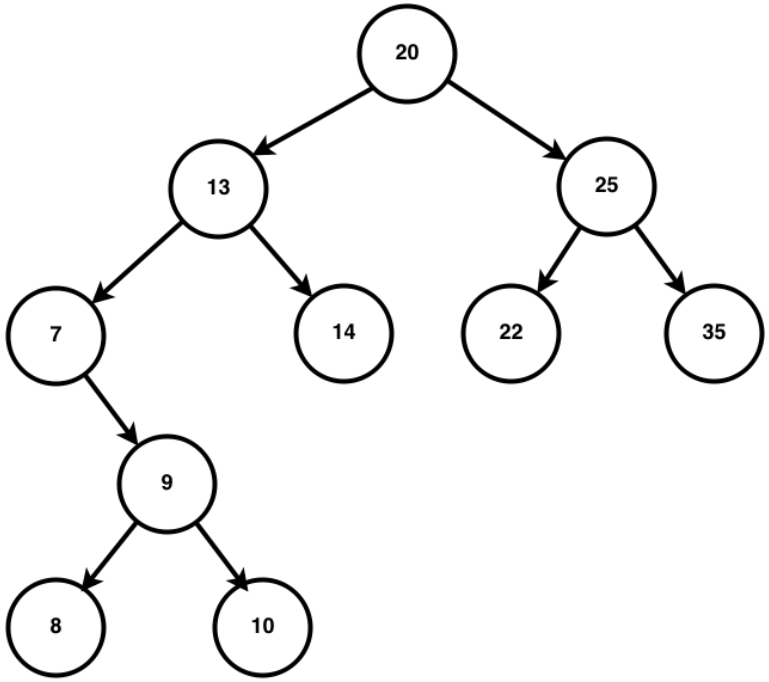
Διάσχιση δυαδικού δένδρου

- Προδιατεταγμένη (preorder)
- Ενδοδιατεταγμένη (inorder)
- Μεταδιατεταγμένη (postorder)
- Κατά σειρά επιπέδων (levelorder)

Στην προδιατεταγμένη, ενδοδιατεταγμένη και μεταδιατεταγμένη διάσχιση διανύονται πρώτα οι κόμβοι του αριστερού υποδένδρου και μετά οι κόμβοι του δεξιού υποδένδρου.

Προδιατεταγμένη (preorder) διάσχιση δένδρου

```
void preorder_traversal(struct node*
      node) {
    if (node == NULL)
       return;
    printf ("%d ", node->key);
   preorder_traversal(node->left);
   preorder_traversal(node->right);
}
```

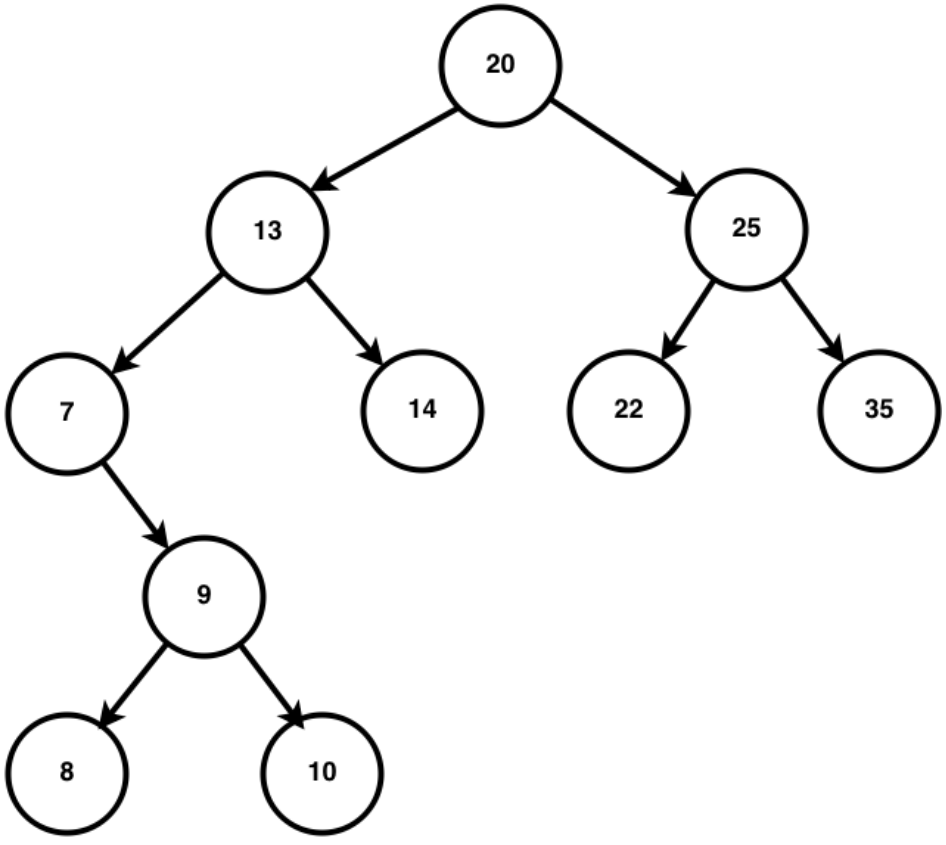


Έξοδος

20, 13, 7, 9, 8, 10, 14, 25, 22, 35

Ενδοδιατεταγμένη (inorder) διάσχιση δένδρου

```
void inorder_traversal(struct node*
     node) {
   if (node == NULL)
      return;
   inorder_traversal(node->left);
   printf("%d ", node->key);
   inorder_traversal(node->right);
}
```

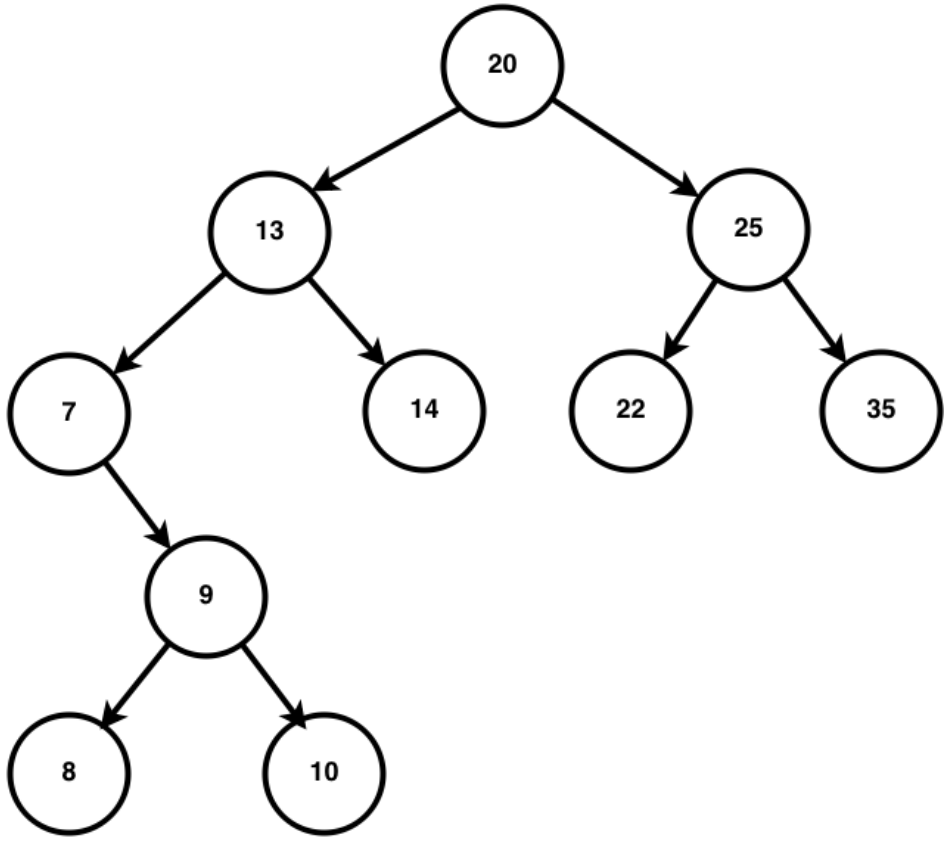


Έξοδος

7, 8, 9, 10, 13, 14, 20, 22, 25, 35

Μεταδιατεταγμένη (postorder) διάσχιση δένδρου

```
void postorder_traversal(struct node*
      node) {
    if (node == NULL)
       return;
    postorder_traversal(node->left);
    postorder_traversal(node->right);
    printf ("%d ", node->key);
}
```

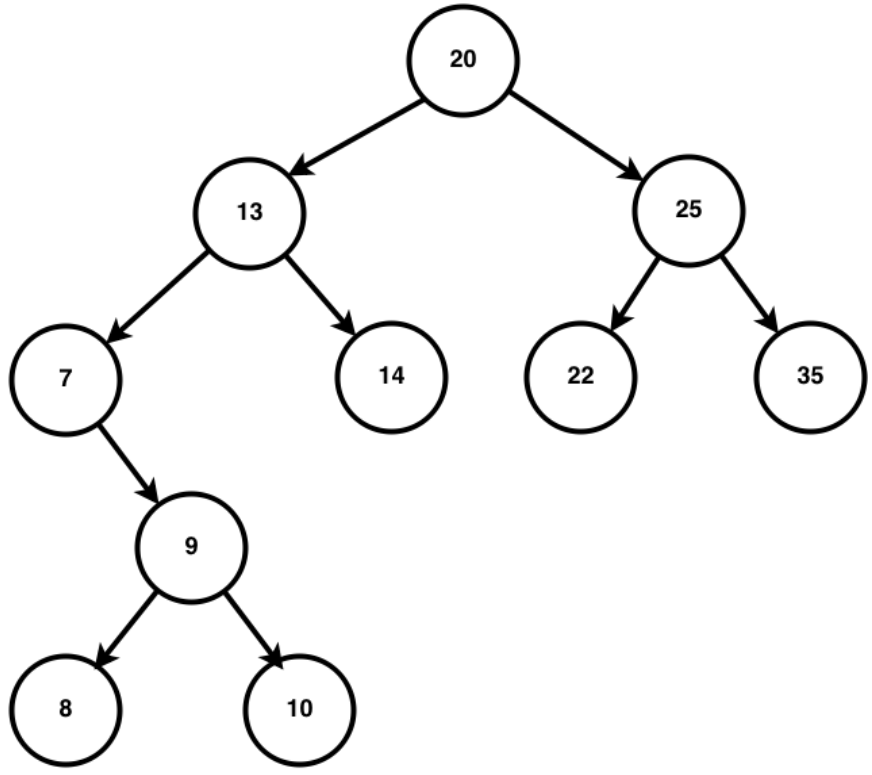


Έξοδος

8, 10, 9, 7, 14, 13, 22, 35, 25, 20

Διάσχιση δένδρου κατά σειρά επιπέδων (levelorder)

```
#include <queue>
 ...
void levelorder_traversal(struct node*
      node) {
    queue<struct node*> q;
    q.push(node);
    while (!q.empty()) {
        struct node* tmp = q.front();
        printf("%d ", tmp->key);
        if (tmp->left != NULL)
            q.push(tmp->left);
        if (tmp->right != NULL)
            q.push(tmp->right);
        q.pop();
    }
}
```



Έξοδος

20, 13, 25, 7, 14, 22, 35, 9, 8, 10

Τυπικές λειτουργίες δυαδικού δένδρου

- Εύρεση ύψους του δένδρου
- Υπολογισμός αριθμού κόμβων του δένδρου
- Αντιγραφή του δένδρου
- Παρουσίαση του δένδρου σε μονάδα εξόδου

Οι παραπάνω ενέργειες μπορούν να πραγματοποιηθούν χρησιμοποιώντας έναν από τους τρόπους διάσχισης του δένδρου: preorder, inorder, postorder και levelorder.

Δυαδικά δένδρα αναζήτησης (Binary Search Trees)

BST

Σε ένα δυαδικό δένδρο αναζήτησης:

- Κάθε κόμβος έχει διαφορετικό (διακριτό) κλειδί σε σχέση με τους υπόλοιπους κόμβους του δένδρου.
- Τα κλειδία που είναι στο αριστερό υποδένδρο ενός κόμβου είναι μικροτερα από το κλειδί του κόμβου και τα κλειδία που είναι στο δεξί υποδένδρο είναι μεγαλύτερα από το κλειδί του κόμβου.

Κάθε υποδένδρο ενός δυαδικού δένδρου αναζήτησης είναι και αυτό δυαδικό δένδρο αναζήτησης

Ένα δυαδικό δένδρο αναζήτησης έχει παρόμοιες αποδόσεις με ένα ταξινομημένο πίνακα για λειτουργίες όπως η αναζήτηση και επιπλέον επιτυγχάνει γρήγορες εισαγωγές νέων στοιχείων και διαγραφές υπαρχόντων στοιχείων.

Αναζήτηση σε δυαδικό δένδρο αναζήτησης

- Εκκίνηση στη ρίζα.
- Σε κάθε κόμβο εξέταση της τιμής κλειδιού σε σχέση με αυτή που αναζητείται και επιλογή του αριστερού ή του δεξιού υποδένδρου για συνέχεια.
- Η διαδικασία τερματίζει όταν βρεθεί το στοιχείο ή όταν φτάσουμε σε NULL.

```
struct node* find(struct node* node, int
      key) {
   if (node == NULL) {
      return (NULL);
   } else {
      if (key == node->key)
         return (node);
      else if (key < node->key)
         return (find(node->left, key));
      else
         return (find(node->right, key));
   }
}
```

Εισαγωγή νέου κλειδιού K σε δυαδικό δένδρο αναζήτησης

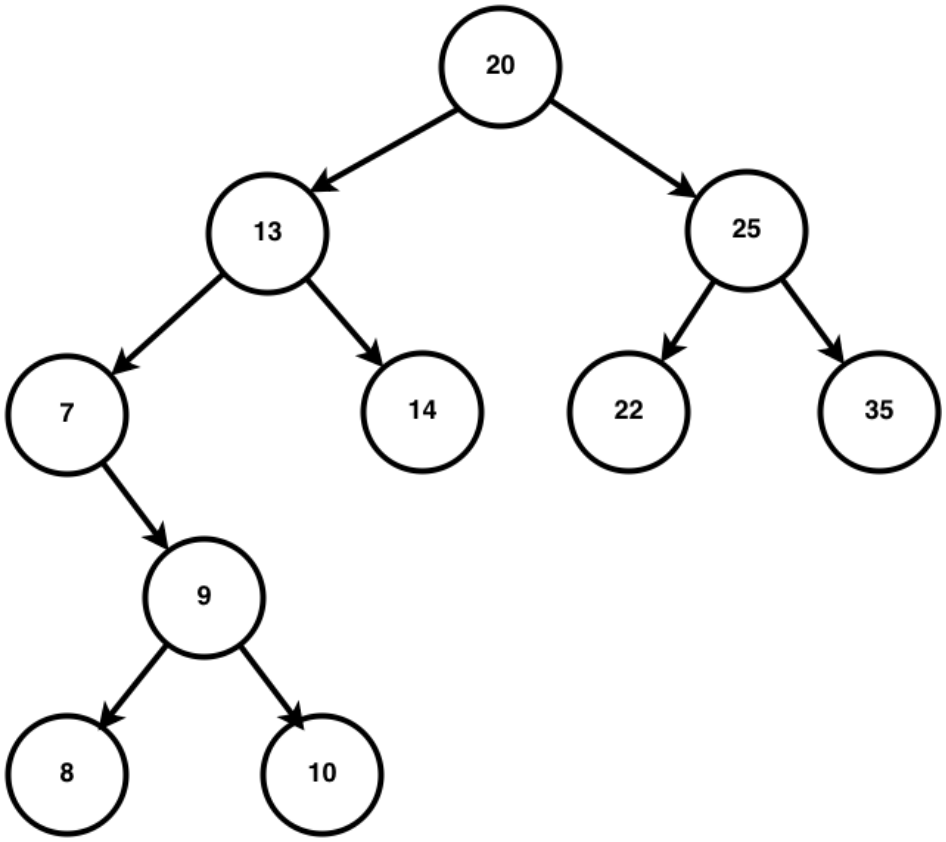
- Αναζήτηση για το κλειδί K.
- Αν το κλειδί δε βρεθεί τότε ο τελευταίος δείκτης που διανύθηκε προκειμένου να φτάσουμε στο NULL αλλάζει έτσι ώστε πλέον να δείχνει στο K.

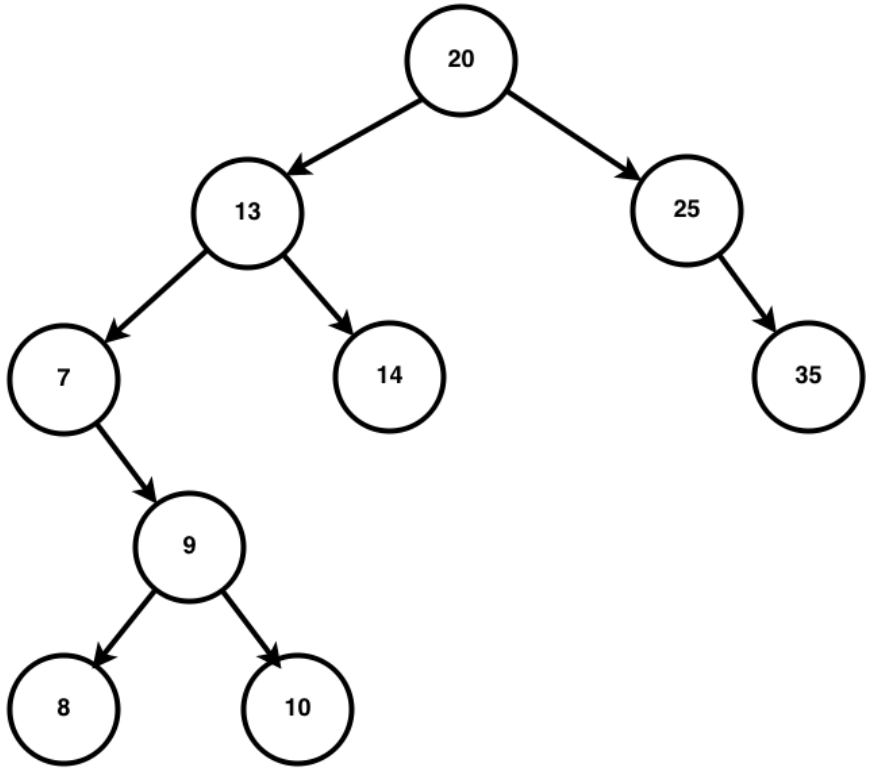
```
struct node* insert_node(struct node*
      node, int key) {
   if (node == NULL) {
      return (new_node(key));
   } else {
      if (key == node->key) {
         cerr << "only distinct
               values are accepted!"
               << endl;
         return node;
      } else if (key < node->key)
         node->left =
               insert_node(node->left,
               key);
      else
         node->right =
               insert_node(node->right,
               key);
      return (node);
   }
}
```

Διαγραφή κλειδιού K από ένα δυαδικό δένδρο αναζήτησης

- Αναζήτηση για το κλειδί K. Διακρίνονται στη συνέχεια 3 περιπτώσεις
- Αν το κλειδί K είναι φύλλο του δένδρου τότε απλά διαγράφεται.
- Αν το κλειδί K είναι κόμβος του δένδρου με ένα μόνο υποδένδρο τότε διαγράφεται ο κόμβος και το υποδένδρο "ανεβαίνει" προς τα πάνω.
- Αν το κλειδί K είναι κόμβος του δένδρου με δύο υποδένδρα τότε αντικαθιστούμε το κλειδί K είτε με το μεγαλύτερο στοιχείο του αριστερού υποδένδρου είτε με το μικρότερο στοιχείο του δεξιού υποδένδρου.

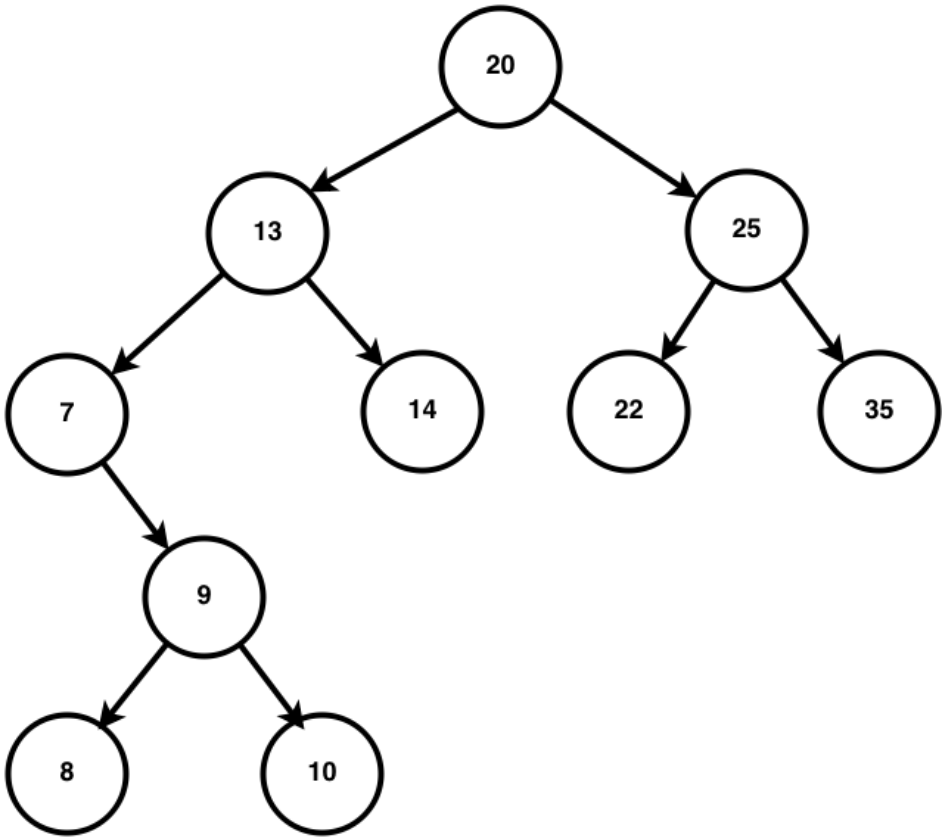
Διαγραφή φύλλου (κλειδί 22)

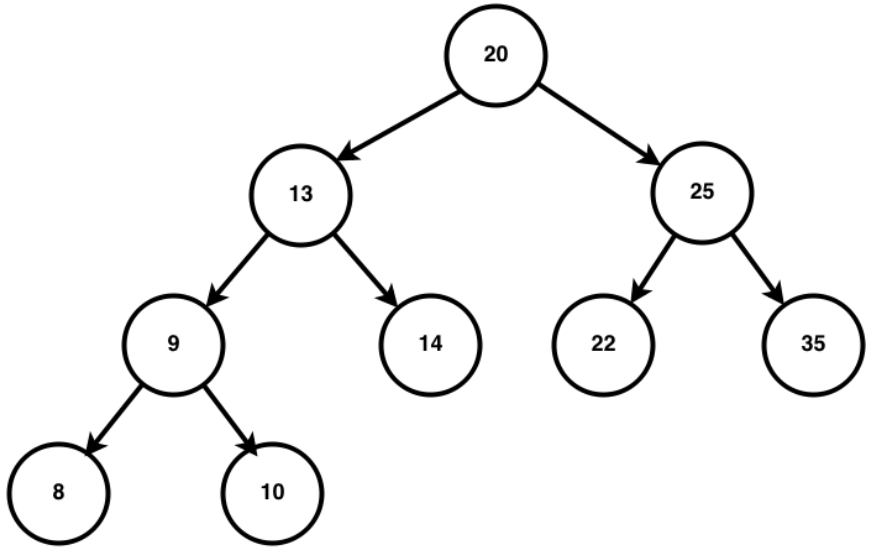




απλά διαγράφεται το κλειδί 22

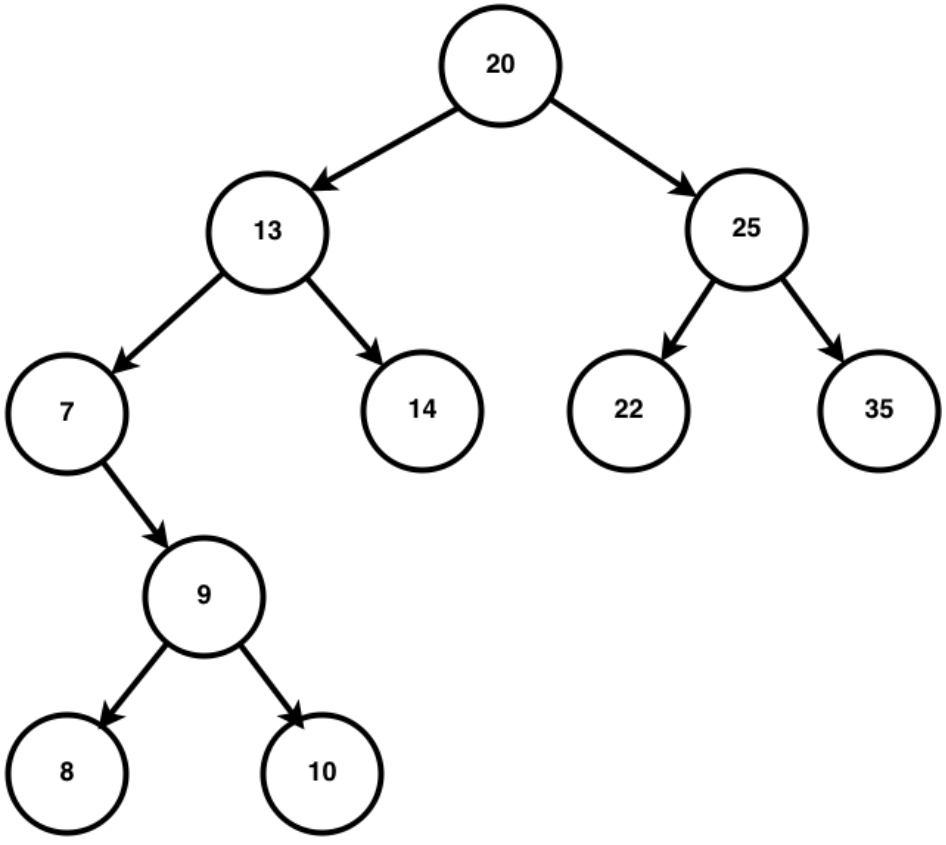
Διαγραφή κόμβου με ένα μόνο υποδένδρο (κλειδί 7)

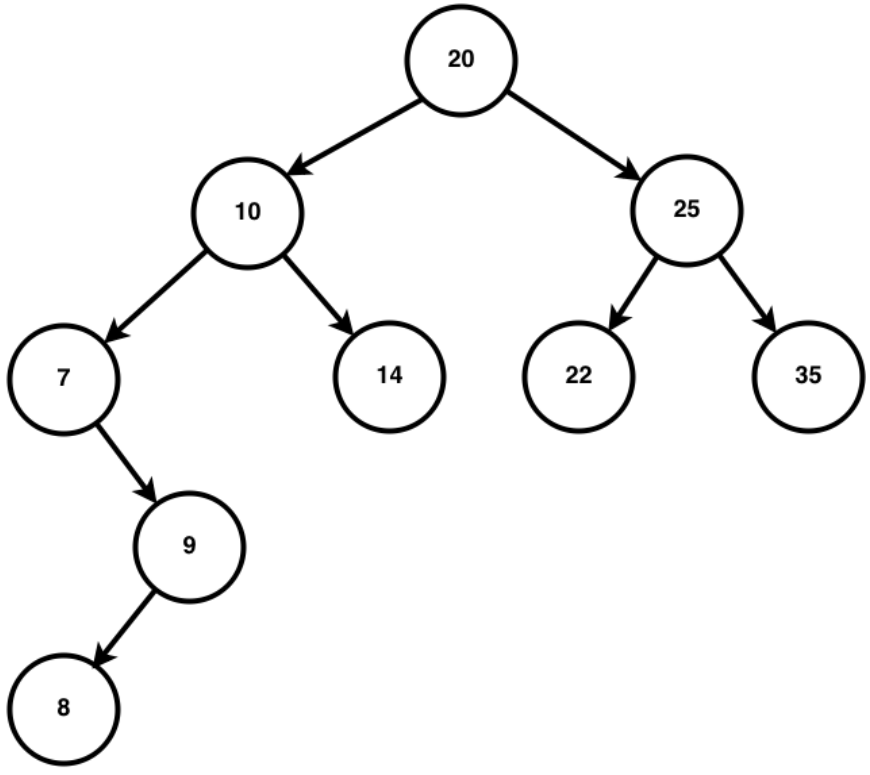




το κλειδί 7 αντικαθίσταται από το κλειδί 9 και όλο το υποδένδρο κάτω από το 9 ανεβαίνει και αυτό προς τα πάνω

Διαγραφή κόμβου με δύο υποδένδρα (κλειδί 13)





Το κλειδί 13 αντικαθίσταται από το μεγαλύτερο κλειδί του αριστερού υποδένδρου του (δλδ το 10) το οποίο και διαγράφεται. Εναλλακτικά θα μπορούσε να αντικατασταθεί με το μικρότερο κλειδί του δεξιού υποδένδρου του (δλδ το 14).

Διαγραφή φύλλου από το δυαδικό δέντρο αναζήτησης

```
void delete_node(struct node* tree, int key) {
    struct node* ptr = tree; struct node* parent = NULL;bool turn_left =
        false;
    while (ptr != NULL && ptr->key != key) {
        parent = ptr;
        if (key < ptr->key) {turn_left = true; ptr = ptr->left;}
        else { turn_left = false; ptr = ptr->right;}}
    if (ptr == NULL) {cout << "Value " << key << " not found" << endl;}
    else if (ptr->left == NULL && ptr->right == NULL) {// case 1: leaf
        if ( turn_left ) parent->left = NULL;else parent->right = NULL;
        delete ptr;
    } else if (ptr->left == NULL) {// case 2: single subtree
        if ( turn_left )parent->left = ptr->right;
        else parent->right = ptr->right;
        delete ptr;
    } else if (ptr->right == NULL) {
        if ( turn_left ) parent->left = ptr->left;
        else parent->right = ptr->left;
        delete ptr;
    } else { // case 3: two subtrees. Replace node to be deleted with
        //the largest value of its left subtree
        int max_value_left_subtree = max_value(ptr->left);
        struct node* tmp = ptr->left;
        while (tmp->right->key != max_value_left_subtree)
            tmp = tmp->right;
        ptr->key = max_value_left_subtree;delete tmp->right;
        tmp->right = NULL;}
}
```

Άλλες λειτουργίες σε δυαδικό δένδρο αναζήτησης

- Εύρεση μεγίστου - ελαχίστου
- Υπολογισμός ύψους δένδρου

```
int max_value(struct node* node) {
    struct node* current = node;
    while (current->right != NULL) {
        current = current->right;
    }
    return (current->key);
}

int height(struct node* node) {
    if (node == NULL) {
        return (-1);
    } else {
        int l_height = height(node->left);
        int r_height = height(node->right);
        if (l_height > r_height)
            return (l_height + 1);
        else
            return (r_height + 1);
    }
}
```

Δυαδικά δένδρα αναζήτησης με εγγύηση μέγιστου ύψους

Ιδέα

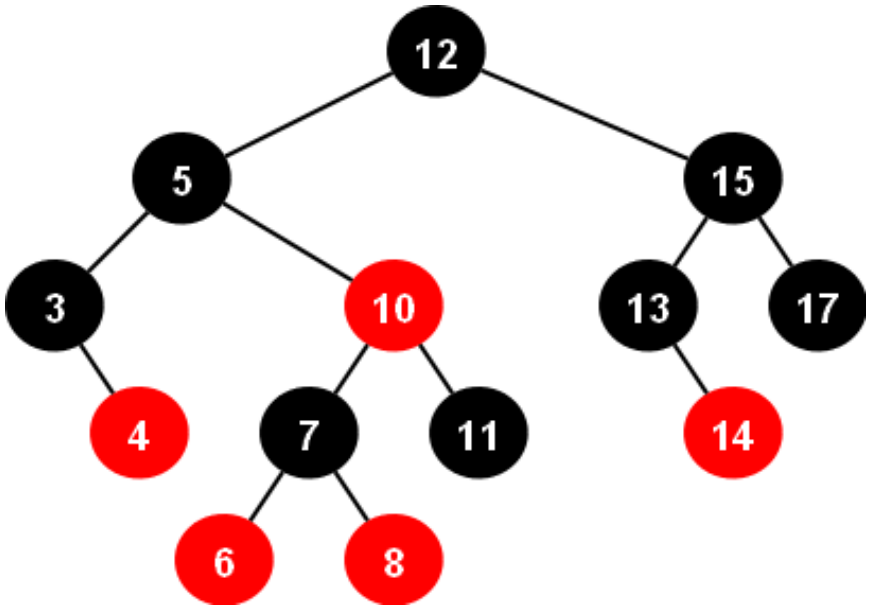
Οργάνωση των κόμβων ενός δυαδικού δένδρου αναζήτησης με τέτοιο τρόπο έτσι ώστε το ύψος του να είναι κοντά στο $\log_2 n$ όπου n είναι το πλήθος των κόμβων του δένδρου. Έτσι λειτουργίες όπως η αναζήτηση, η εισαγωγή, η διαγραφή, η εύρεση μεγίστου κλπ εκτελούνται σε χρόνο $O(\log_2 n)$.

- Κόκκινα μαύρα δένδρα (Red Black Trees)
- AVL δένδρα

Κόκκινα Μαύρα Δένδρα

Τα κόκκινα μαύρα δένδρα είναι δυαδικά δένδρα αναζήτησης για τα οποία ισχύουν κανόνες που έχουν να κάνουν με μια επιπλέον ιδιότητα χρώματος η οποία προσαρτάται σε κάθε κόμβο. Προτάθηκαν από τον R. Bayer το 1972 και τους L. Guibas και R. Sedgewick το 1978.

- Κάθε κόμβος μπορεί να είναι κόκκινος ή μαύρος.
- Η ρίζα είναι μαύρη.
- Δεν μπορεί να υπάρχουν δύο συνεχόμενοι κόκκινοι κόμβοι σε οποιαδήποτε διαδρομή από την κορυφή προς ένα φύλλο του δένδρου (δλδ κάθε κόκκινος κόμβος επιτρέπεται να έχει μόνο μαύρα παιδιά).
- Κάθε διαδρομή από τη ρίζα προς τα φύλλα έχει τον ίδιο αριθμό από μαύρους κόμβους.



http://cs.lmu.edu/~ray/notes/redblacktrees/

Ύψος κόκκινων μαύρων δένδρων

- Το ύψος ενός κόκκινου μαύρου δένδρου με n κόμβους είναι μικρότερο ή ίσο του $2\log_2(n+1)$.
- Κατά την εισαγωγή νέων κόμβων ή τη διαγραφή υπαρχόντων κόμβων θα πρέπει να διασφαλιστεί ότι το δένδρο εξακολουθεί να έχει τις ιδιότητες που πρέπει να ισχύουν για τα κόκκινα μαύρα δένδρα. Για να επιτευχθεί αυτό πραγματοποιούνται περιστροφές (rotations) τμημάτων του δένδρου και επαναχρωματισμοί κόμβων.
- Τα κόκκινα μαύρα δένδρα χρησιμοποιούνται σε διάφορες βιβλιοθήκες καθώς και στην STL της C++ για να υλοποιήσουν τα std::set και std::map.

```
#include <set>
#include <map>
 ...
set<int> a_set;
a_set.insert(5); a_set.insert(7);
a_set.insert(5); // already exists
a_set.insert(3); a_set.insert(10);
for (int x : a_set)
    cout << x << " ";
cout << endl;
// outputs: 3 5 7 10
 ...
map<string, string> agenta;
agenta["Christos"] = "12345";
agenta["Maria"] = "23456";
agenta["Petros"] = "34567";
for (pair<string, string> k_v :
      agenta)
    cout << k_v.first << "-->" <<
         k_v.second << endl;
//outputs
// Christos-->12345
// Maria-->23456
// Petros-->34567
```

AVL δένδρα

Προτάθηκαν από τους G.M.Adelson-Velsky και E.M. Landis το 1962. Η διαφορά στο ύψος του αριστερού και του δεξιού υποδένδρου δεν θα πρέπει να υπερβαίνει τη μονάδα ή ισοδύναμα η διαφορά στα ύψη του αριστερού με το δεξιό υποδένδρο μπορεί να είναι +1, 0 ή -1 (το ύψος του άδειου υποδένδρου ορίζεται ως -1).
Το ύψος h οποιουδήποτε AVL δένδρου με n κόμβους είναι:
$\lfloor \log_2 n \rfloor \leq h \leq 1.4405 \log_2(n+2) - 1.3277$.

Αν η εισαγωγή ενός νέου κόμβου ή η διαγραφή ενός υπάρχοντος κόμβου κάνει το AVL δένδρο μη ισοζυγισμένο τότε το δένδρο θα πρέπει να επιδιορθωθεί με περιστροφές (rotations) τμημάτων του. Οι περιστροφές μπορούν να γίνουν σε σταθερό χρόνο κατά μέσο όρο οπότε η χρήση των AVL δένδρων είναι αποδοτική.

Σύγκριση AVL δένδρων και Red Black δένδρων

Τα AVL δένδρα είναι αυστηρότερα ισοζυγισμένα (balanced) σε σχέση με τα Red Black δένδρα. Αυτό οδηγεί σε πιο αργές εισαγωγές και διαγραφές αλλά ταχύτερες αναζητήσεις για τα AVL δένδρα σε σχέση με τα Red Black δένδρα.

Περιστροφές (rotations) σε AVL δένδρα

Παράγοντας ισορροπίας (bf = balance factor)

Είναι η τιμή που προκύπτει αν αφαιρέσουμε από το ύψος του δεξιού υποδένδρου το ύψος του αριστερού υποδένδρου.

- Αν το bf είναι 1 ή 0 ή -1 τότε ο κόμβος θεωρείται ισοζυγισμένος. Αν αυτό ισχύει για όους τους κόμβους του δένδρου τότε το δένδρο είναι AVL.
- Η περιστροφές αλλάζουν τη δομή του δένδρου χωρίς όμως να αλλάζουν τη σειρά των στοιχείων.
- Ο σκοπός των περιστροφών είναι να μειώσουν το ύψος ενός υποδένδρου κατά 1 και να αυξήσουν το ύψος ενός άλλου υποδένδρου κατα 1.

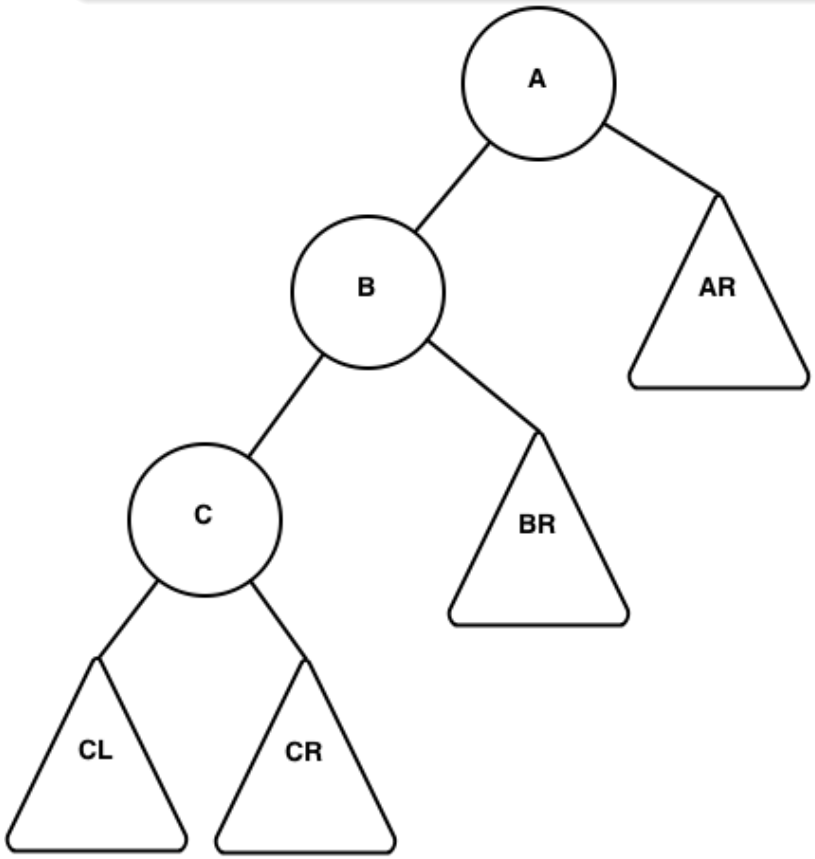
Είδη περιστροφών σε AVL δένδρα

- δεξιά περιστροφή (RR = Right Rotation)
- αριστερή περιστροφή (LR = Left Rotation)
- δεξιά και αριστερή περιστροφή (RLR = Right Left Rotation)
- αριστερή και δεξιά περιστροφή (LRR = Left Right Rotation)

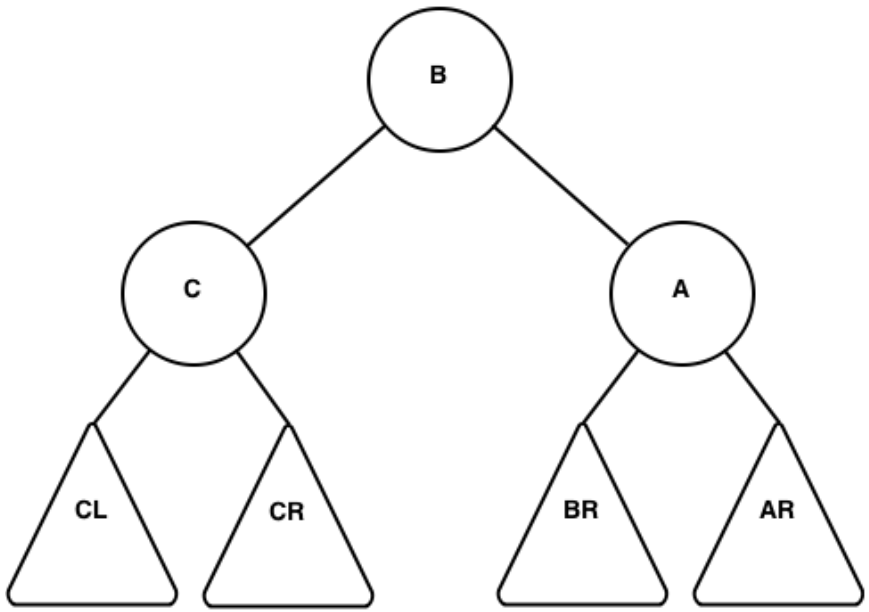
Δεξιά περιστροφή σε AVL δένδρα

Δεξιά περιστροφή

- Το B γίνεται η νέα ρίζα του υποδένδρου
- Το δεξί παιδί του B γίνεται αριστερό παιδί του A
- Το A γίνεται δεξί παιδί του B



CL<C<CR<B<BR<A<AR

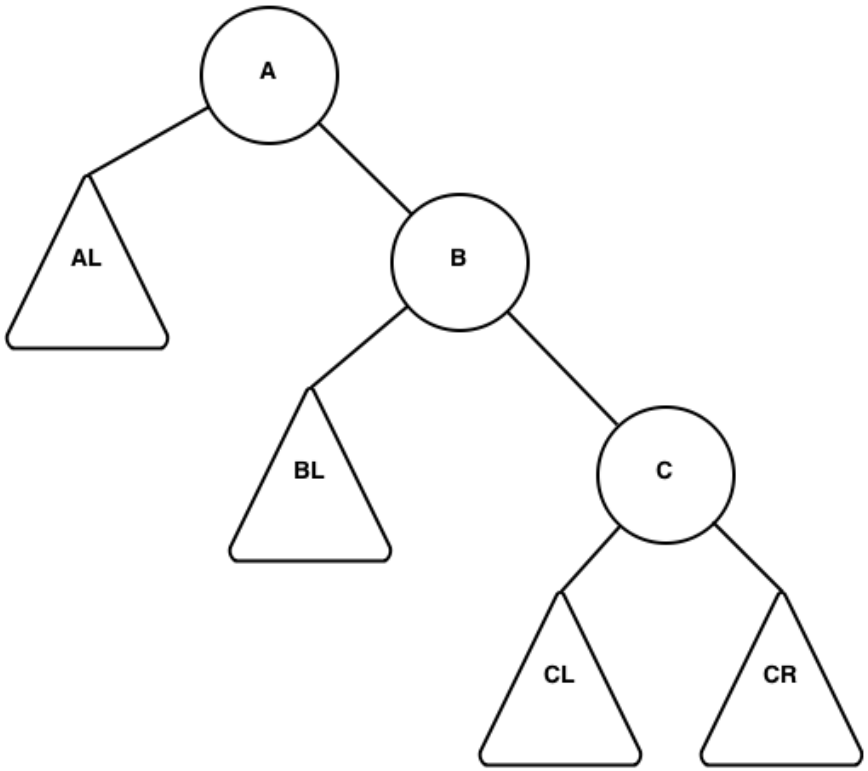


CL<C<CR<B<BR<A<AR

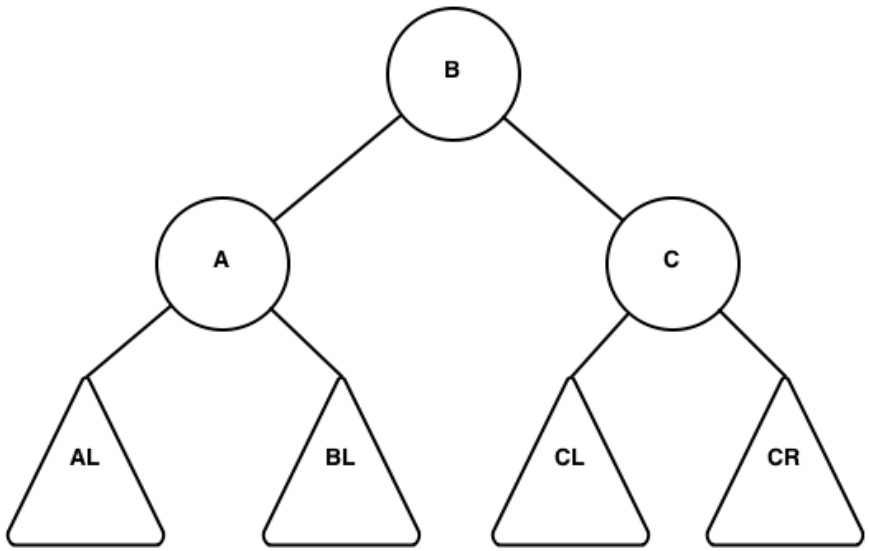
Αριστερή περιστροφή σε AVL δένδρα

Αριστερή περιστροφή

- Το B γίνεται η νέα ρίζα του υποδένδρου
- Το αριστερό παιδί του B γίνεται δεξί παιδί του A
- Το A γίνεται αριστερό παιδί του B



AL<A<BL<B<CL<C<CR

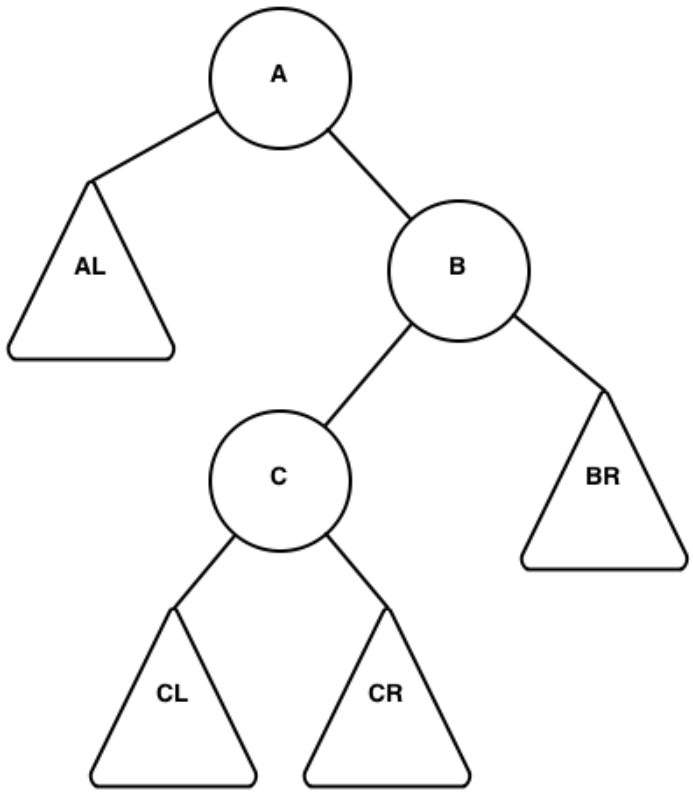


AL<A<BL<B<CL<C<CR

Αριστερή - δεξιά περιστροφή σε AVL δένδρα

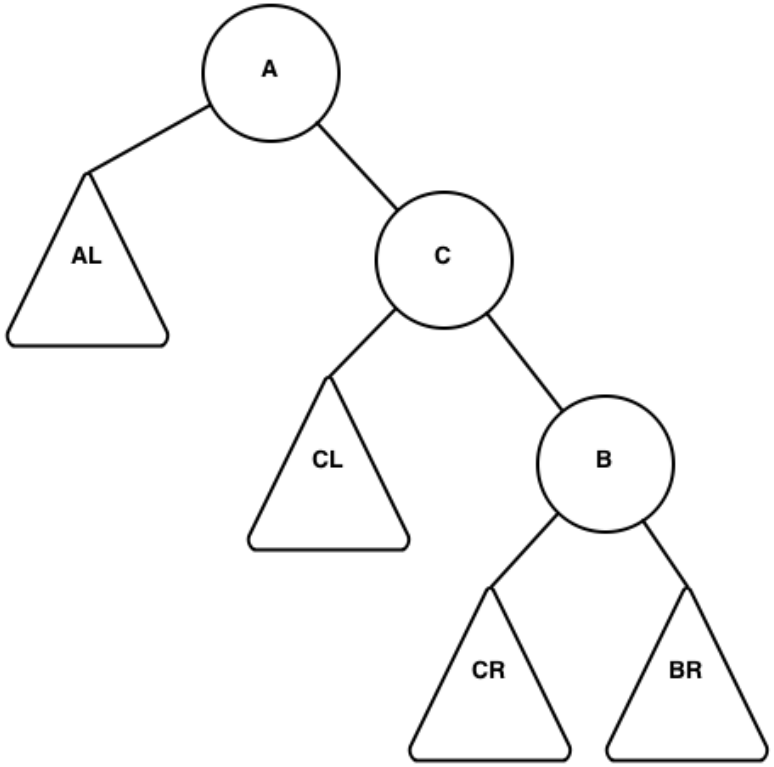
Αριστερή - δεξιά περιστροφή (LRR)

- Πρώτα γίνεται μια δεξιά περιστροφή στο κάτω μέρος του δένδρου και στη συνέχεια μια αριστερή περιστροφή στο πάνω μέρος του.

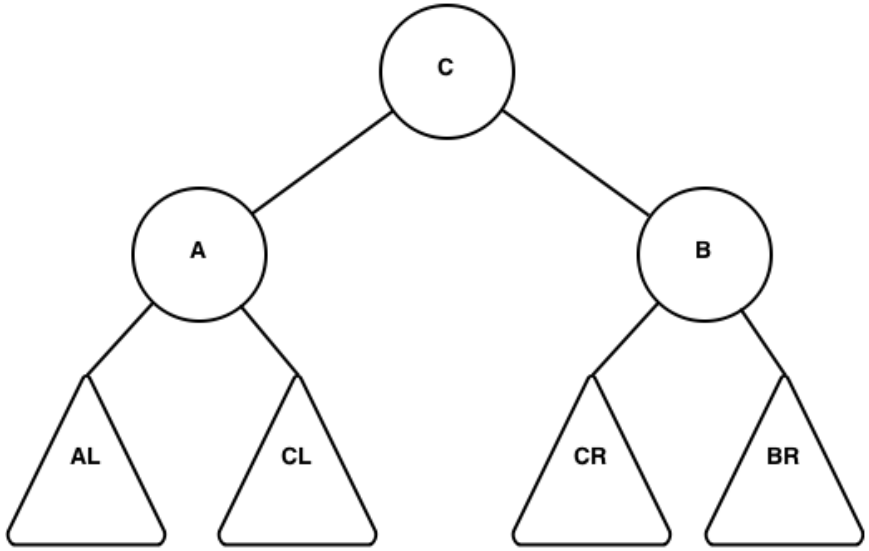


AL<A<CL<C<CR<B<BR

Αριστερή - δεξιά περιστροφή σε AVL δένδρα



AL<A<CL<C<CR<B<BR

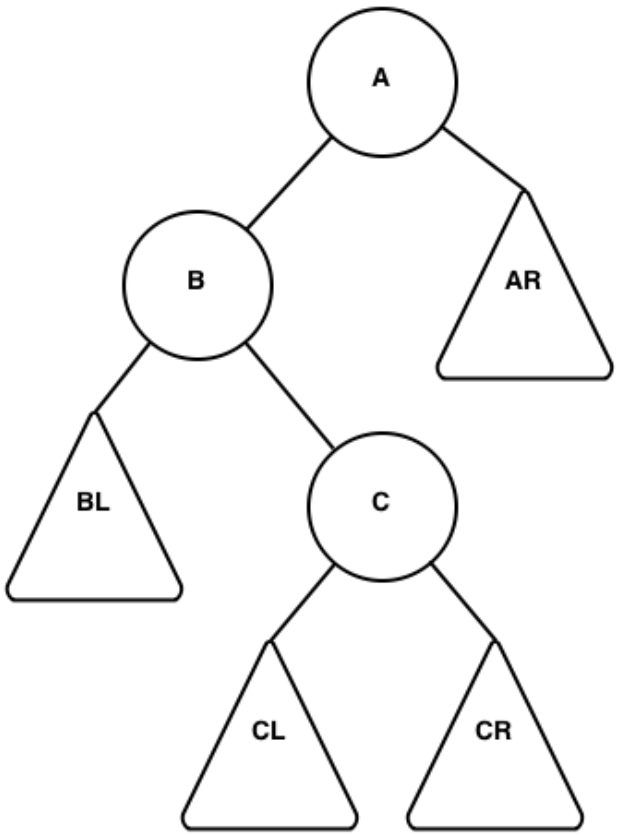


AL<A<CL<C<CR<B<BR

Δεξιά - αριστερή περιστροφή σε AVL δένδρα

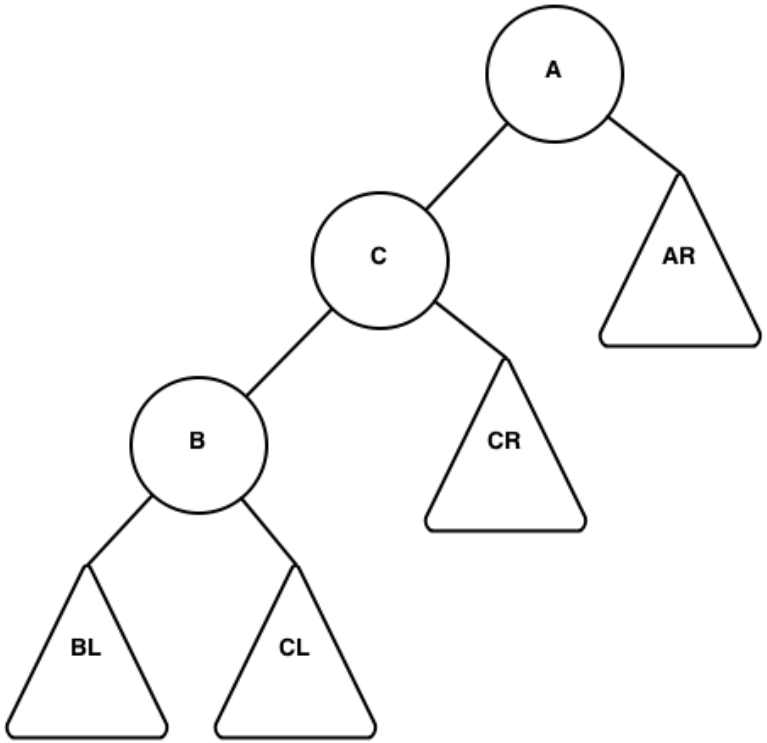
Δεξιά - αριστερή περιστροφή (RLR)

- Πρώτα γίνεται μια αριστερή περιστροφή στο κάτω μέρος του δένδρου και στη συνέχεια μια δεξιά περιστροφή στο πάνω μέρος του.

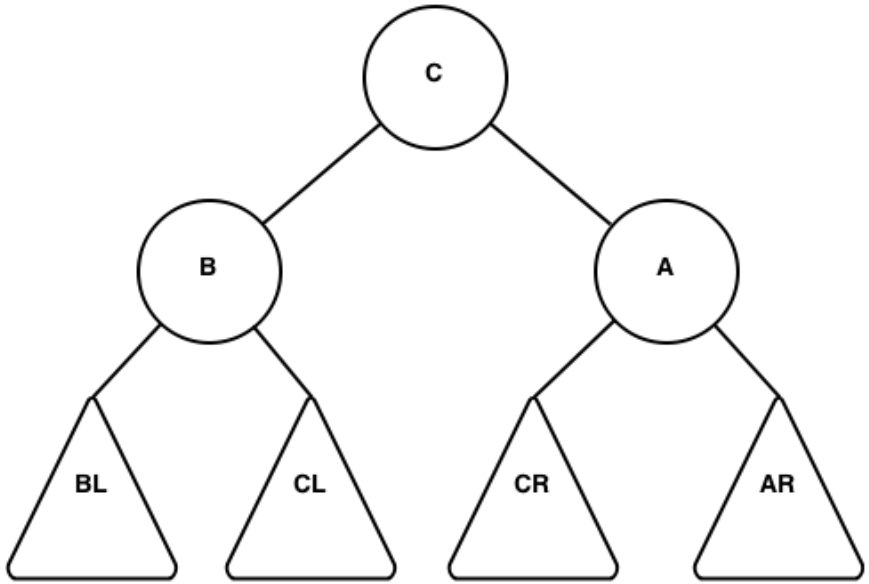


BL<B<CL<C<CR<A<AR

Δεξιά - αριστερή περιστροφή σε AVL δένδρα



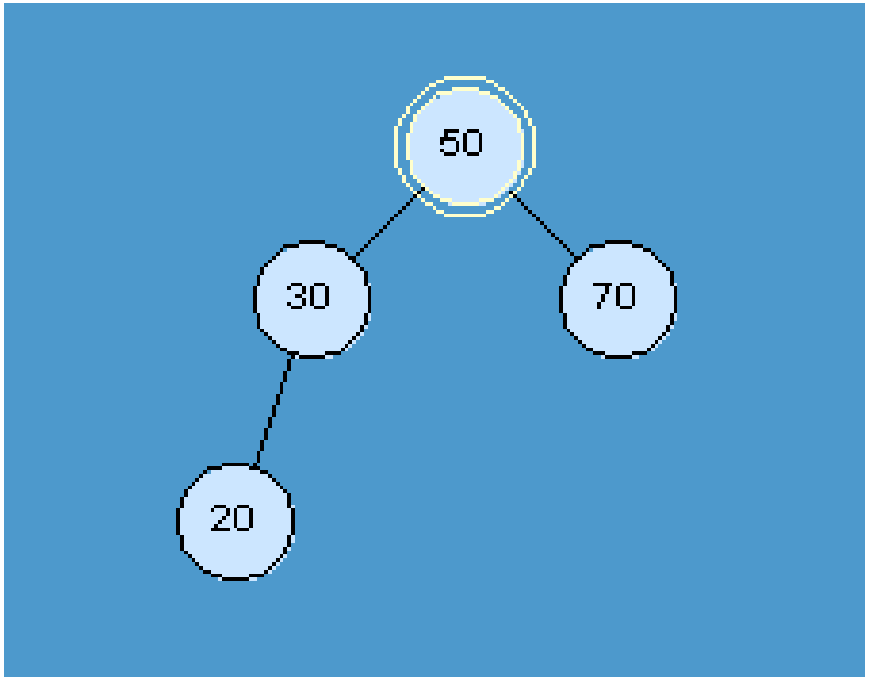
BL<B<CL<C<CR<A<AR



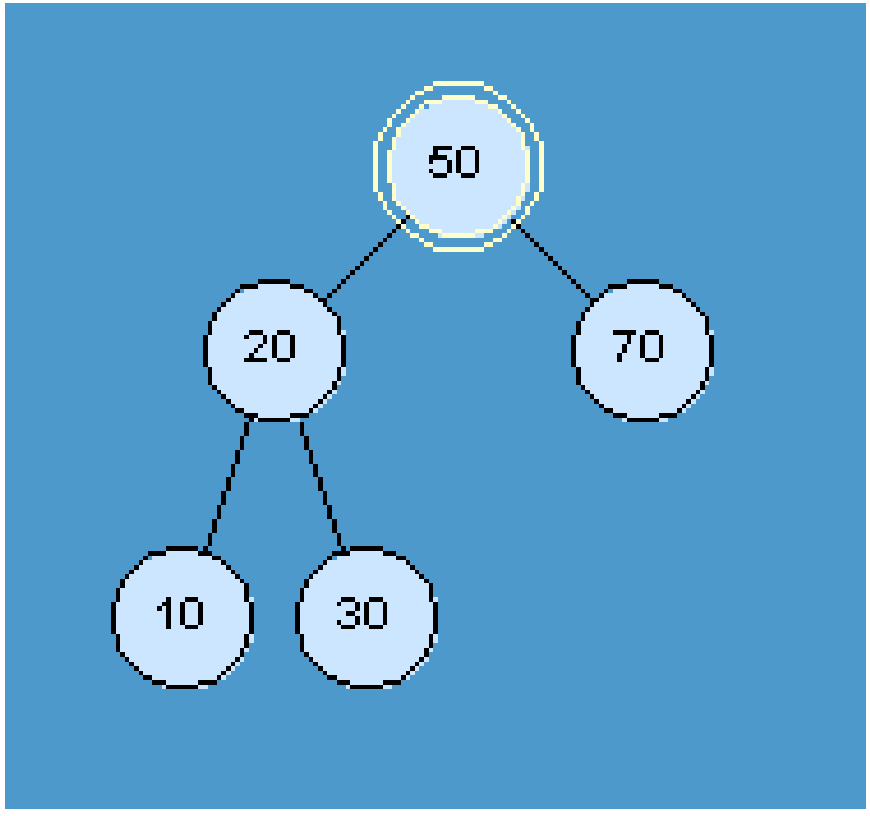
BL<B<CL<C<CR<A<AR

Παράδειγμα (1/3)

Παράδειγμα σταδιακής εισαγωγής σε ένα AVL δένδρο των τιμών 50,30,70,20,10,25,28,27,26



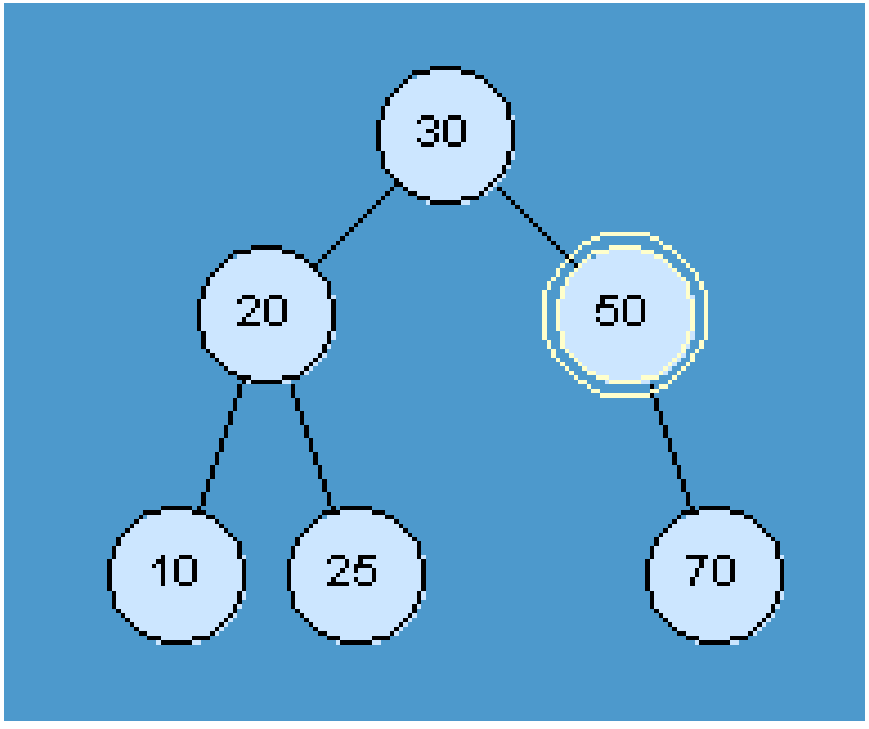
Εισαγωγή των τιμών 50, 30, 70, 20



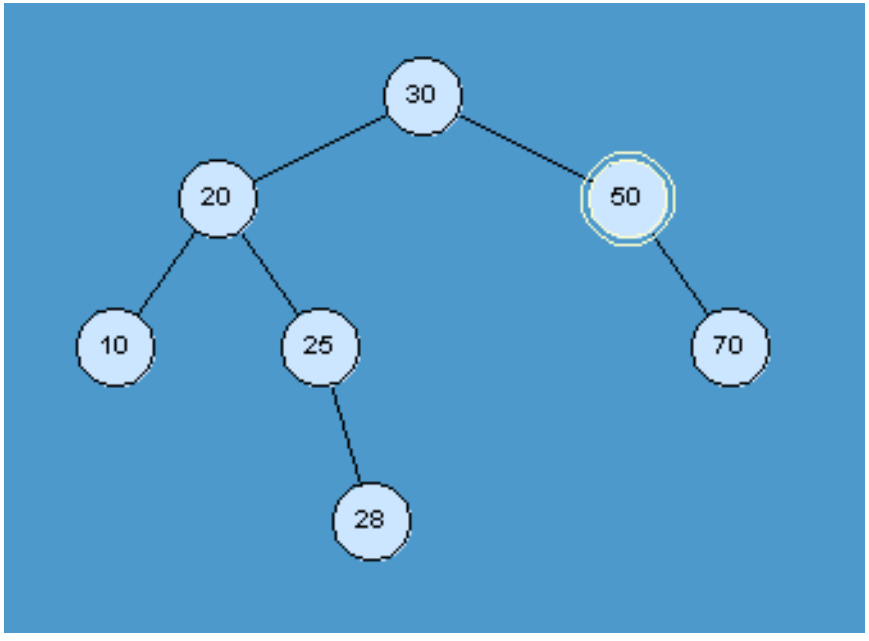
Εισαγωγή της τιμής 10

Παράδειγμα (2/3)

Παράδειγμα σταδιακής εισαγωγής σε ένα AVL δένδρο των τιμών 50,30,70,20,10,25,28,27,26



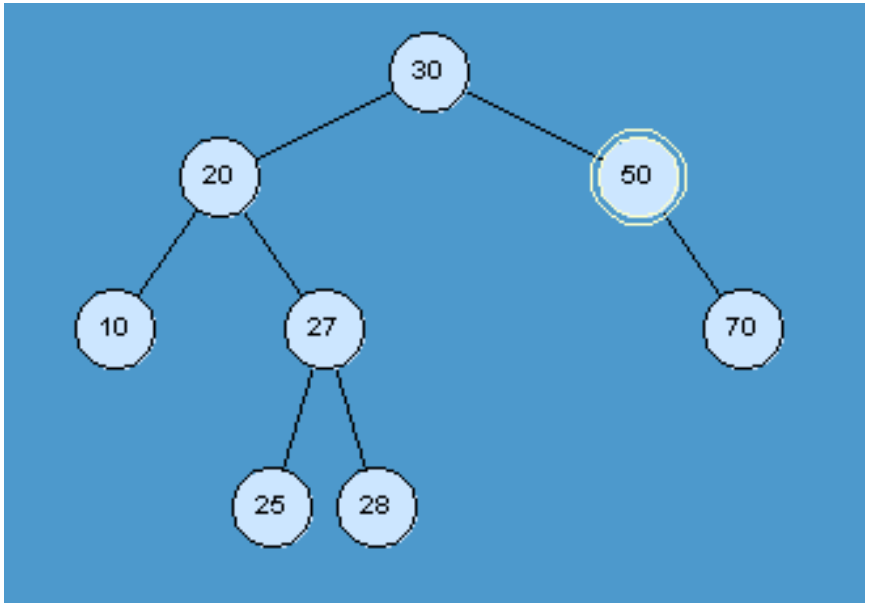
Εισαγωγή της τιμής 25



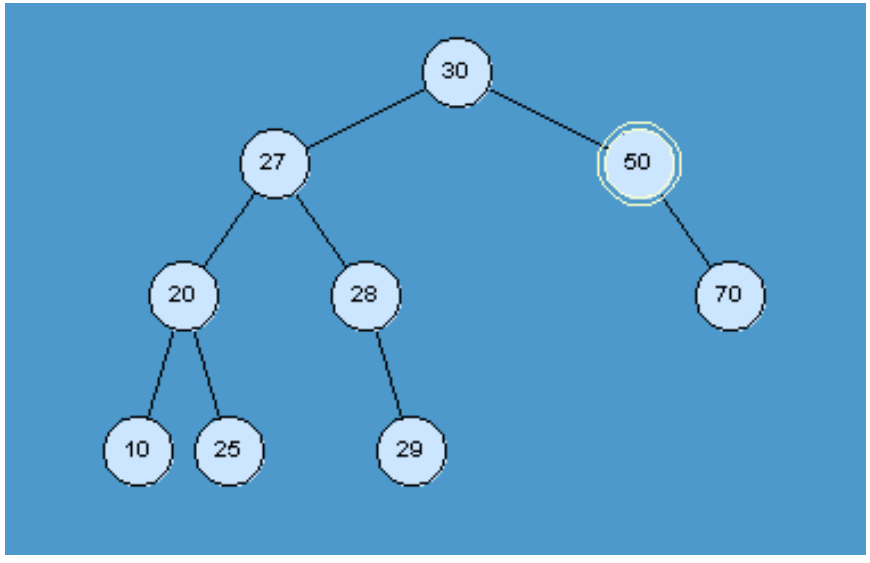
Εισαγωγή της τιμής 28

Παράδειγμα (3/3)

Παράδειγμα σταδιακής εισαγωγής σε ένα AVL δένδρο των τιμών 50,30,70,20,10,25,28,27,26



Εισαγωγή της τιμής 27



Εισαγωγή της τιμής 26