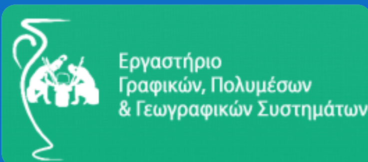
AVL-trees

C++ implementation

Δομές Δεδομένων

Μάριος Κενδέα

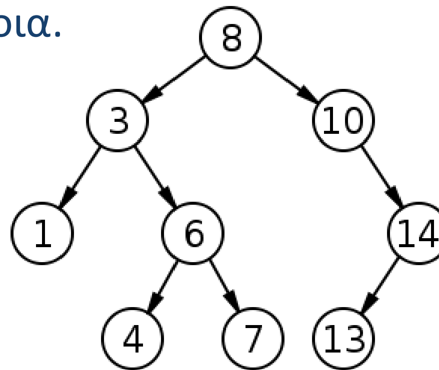


31 Μαρτίου 2015

kendea@ceid.upatras.gr

Εισαγωγή (1/3)

- **Δυαδικά Δένδρα Αναζήτησης**: πολύ καλή δομή δεδομένων για την υλοποίηση maps, sets, και άλλα παρόμοια.



- Κύρια δυσκολία : είναι αποδοτικά μόνο όταν είναι ισοζυγισμένα
- Ορίζονται σχέσεις μεταξύ κόμβων
  - Parent: ο γονεϊκός κόμβος του κόμβου αναφοράς
  - LSON: το αριστερό παιδί του κόμβου αναφοράς
  - RSON: το δεξί παιδί του κόμβου αναφοράς
  - Ancestor: κάποιος κόμβος πρόγονος του κόμβου αναφοράς

Εισαγωγή (2/3)

- Ύψος δέντρου: για ένα δέντρο με n στοιχεία χρειάζονται O(logn) επίπεδα.
- **Το ύψος του δέντρου καθορίζει και το κόστος αναζήτησης γιατί έχουμε μια σύγκριση σε κάθε επίπεδο**
- **Ιδιότητα δυαδικού δέντρου αναζήτησης:**

Για να είναι ένα δέντρο δυαδικό δέντρο αναζήτησης (Binary Search Tree – BST) πρέπει να πληροί την ιδιότητα δυαδικού δέντρου αναζήτησης:

Έστω n κόμβος ενός δυαδικού δέντρου αναζήτησης. Τότε όλοι οι κόμβοι που βρίσκονται στο αριστερό υποδέντρο του n έχουν τιμή περιεχομένου μικρότερη ή ίση με το περιεχόμενο του n. Αντίστοιχα, οι κόμβοι που βρίσκονται στο δεξιό υποδέντρο του n έχουν τιμή περιεχομένου μεγαλύτερη ή ίση από το περιεχόμενο του n.

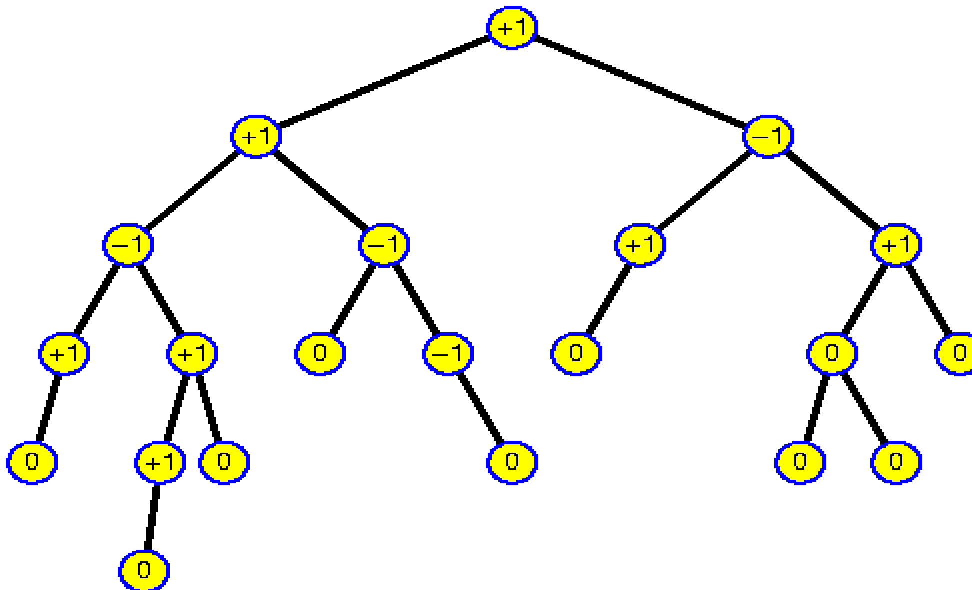
- **Έτσι το δυαδικό δέντρο αναζήτησης εγγυάται χρόνο αναζήτησης O(logn) με μια σύγκριση σε κάθε ένα από τα O(logn) επίπεδα. Στη συνέχεια η ένθεση (insert) και η διαγραφή απαιτούν σταθερό χρόνο.**

Εισαγωγή (3/3)

- Τα ισοζυγισμένα δέντρα απαντούν στο πρόβλημα του ύψους χειρότερης περίπτωσης ενός δυαδικού δέντρου αναζήτησης
  - Διατηρούν μια αναλογία μεταξύ του αριστερού και του δεξιού υποδέντρου σε κάθε κόμβο ώστε το συνολικό ύψος να είναι πολύ κοντά στο logn
  - Ισοζυγισμένα με βάρη (weight-balanced) και ισοζυγισμένα με ύψος (height – balanced)
- **Το AVL tree είναι ένα ισοζυγισμένο δέντρο βάσει ύψους (ισοσκελισμένο) που σε κάθε κόμβο του, το ύψος του αριστερού υποδέντρου επιτρέπεται να διαφέρει μόνο κατά 1 από το ύψος του δεξιού υποδέντρου.**

AVL Trees (1/8)

- Τα AVL trees, είναι μια απλή και αποδοτική δομή δεδομένων για την διατήρηση της ισορροπίας.
- Είναι η πρώτη που προτάθηκε (“An algorithm for the organisation of information” - Proceedings of the USSR Academy of Sciences 146: 263–266, 1962) από τους G.M. Adelson-Velskii και E.M. Landis

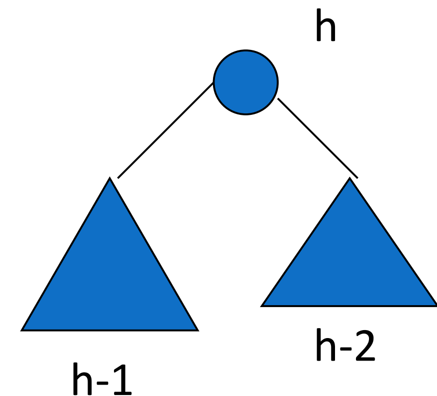


AVL Trees (2/8)

- Σε κάθε κόμβο του δέντρου αντιστοιχεί ένας αριθμός που δείχνει τον παράγοντα ισοσκελισμού.
  - Αν hleft είναι το ύψος του αριστερού υποδέντρου του κόμβου v και hright το ύψος του δεξιού υποδέντρου, τότε ο παράγοντας ισοσκελισμού υπολογίζεται από την πράξη hright - hleft
  - Εφόσον θέλουμε η διαφορά ύψους των υποδέντρων θέλουμε να είναι το πολύ 1, επιτρεπτές τιμές είναι τα -1, 0, 1.
  - Το AVL tree εγγυάται ύψος δέντρου h τέτοιο ώστε

$$\log n \le h < 1.44\log(n+2) - 1$$

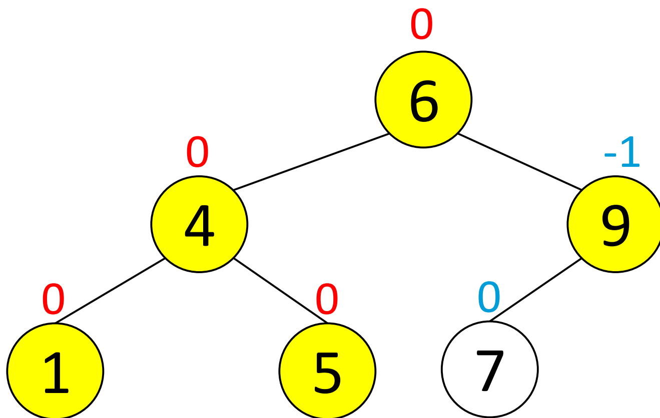
Άρα επί της ουσίας O(logn)!



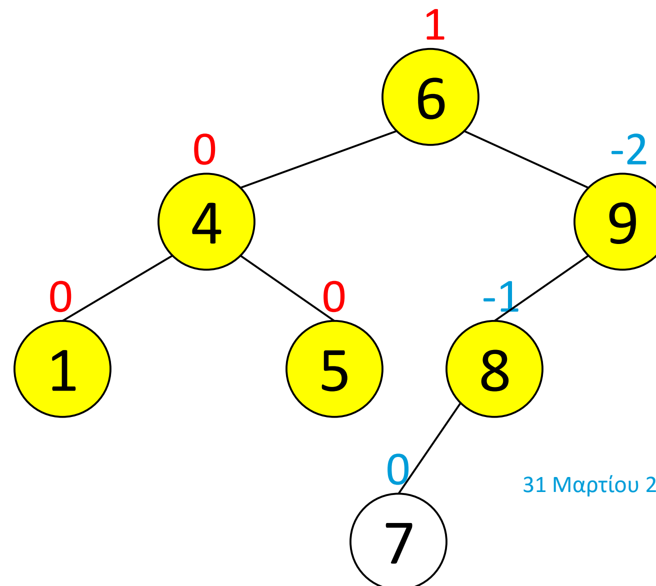
AVL Trees (3/8)
Insert

- Η εισαγωγή γίνεται κανονικά με εύρεση του κατάλληλου σημείου για ένθεση και στη συνέχεια κατασκευή νέου κόμβου στο σημείο αυτό.
  - Πολλές φορές χαλάει η ισορρόπία του AVL tree και εμφανίζονται βάρη 2 ή -2.
  - Για να αποκατασταθεί ο ισοσκελισμός και η ιδιότητα των AVL δέντρων πρέπει να γίνουν κάποιες πράξεις που ονομάζονται περιστρόφές

Tree A (AVL)



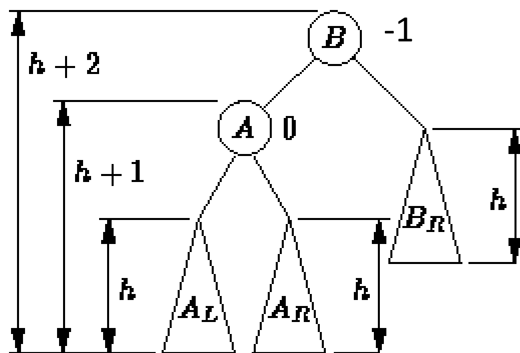
Tree B (not AVL)



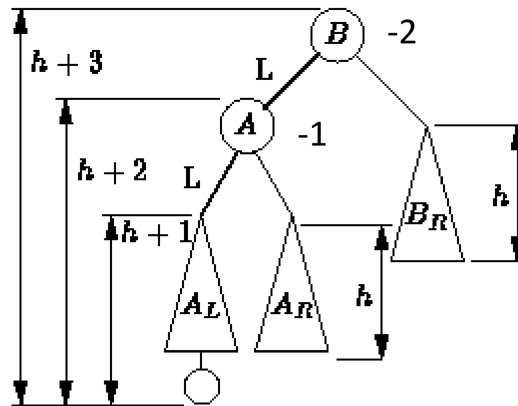
AVL Trees (4/8)
Insert

- Μόνο κόμβοι στο path από το σημείο εισαγωγής προς τον κόμβο της ρίζας μπορεί να αλλάξουν ύψος
- Οπότε, πάμε προς τα πάνω , διορθώνοντας τα ύψη
- (h_{right}-h_{left}) είναι 2 ή –2, κάνουμε περιστροφή γύρο από τον κόμβο
- Αν ο κόμβος που χρειάζεται περιστροφή είναι ο α.
- Υπάρχουν 4 περιπτώσεις:
  - Εξωτερικές περιπτώσεις (απαιτεί απλή περιστροφή) :
    1. Εισαγωγή στο αριστερό υποδέντρο του αριστερού παιδιού του α.
    2. Εισαγωγή στο δεξί υποδέντρο του δεξιού παιδιού του α.
  - Εσωτερικές περιπτώσεις (απαιτεί διπλή περιστροφή) :
    3. Εισαγωγή στο δεξί υποδέντρο του αριστερού παιδιού του α.
    4. Εισαγωγή στο αριστερό υποδέντρο του δεξιού παιδιού του α.
- Με αντίστοιχο τρόπο λειτουργεί και η διαγραφή: αναζήτηση, διαγραφή, εξισορρόπηση.

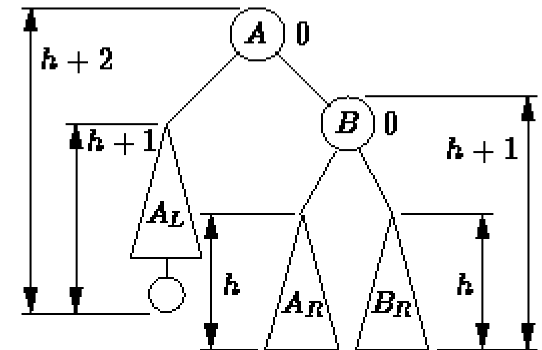
AVL Trees (5/8)
Απλή περιστροφή



(a)

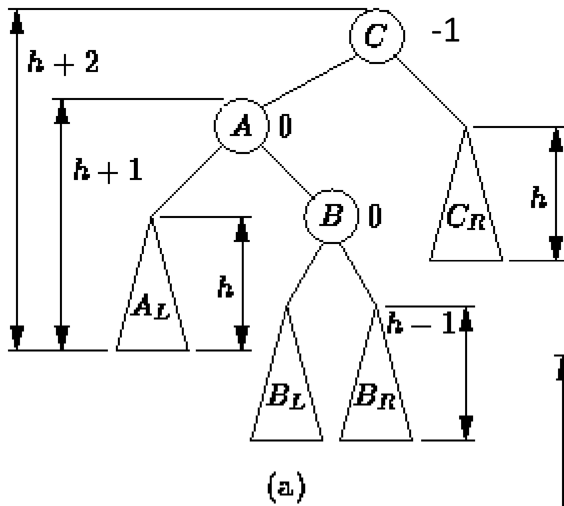


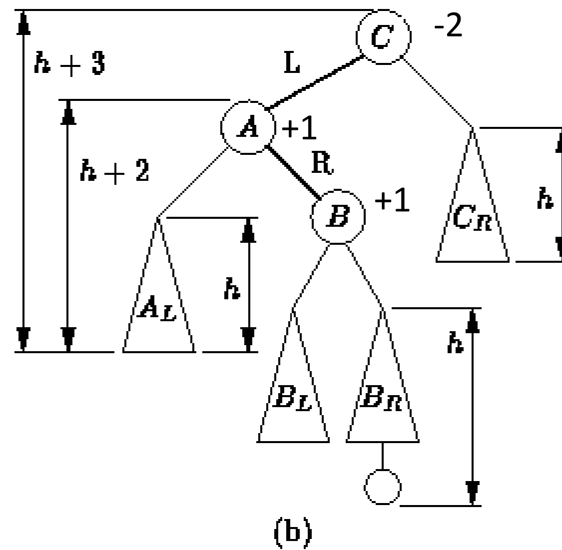
(b)

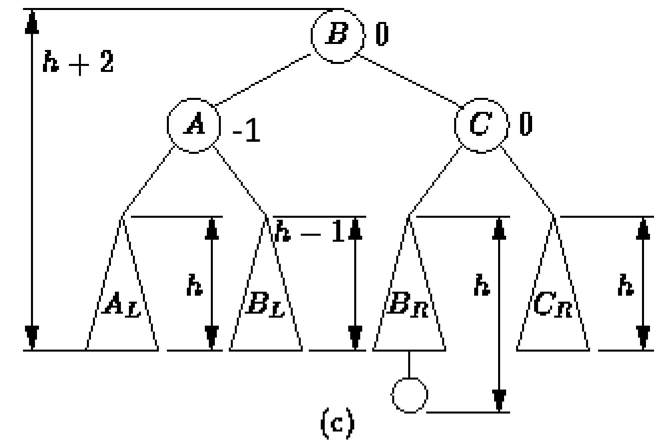


(c)

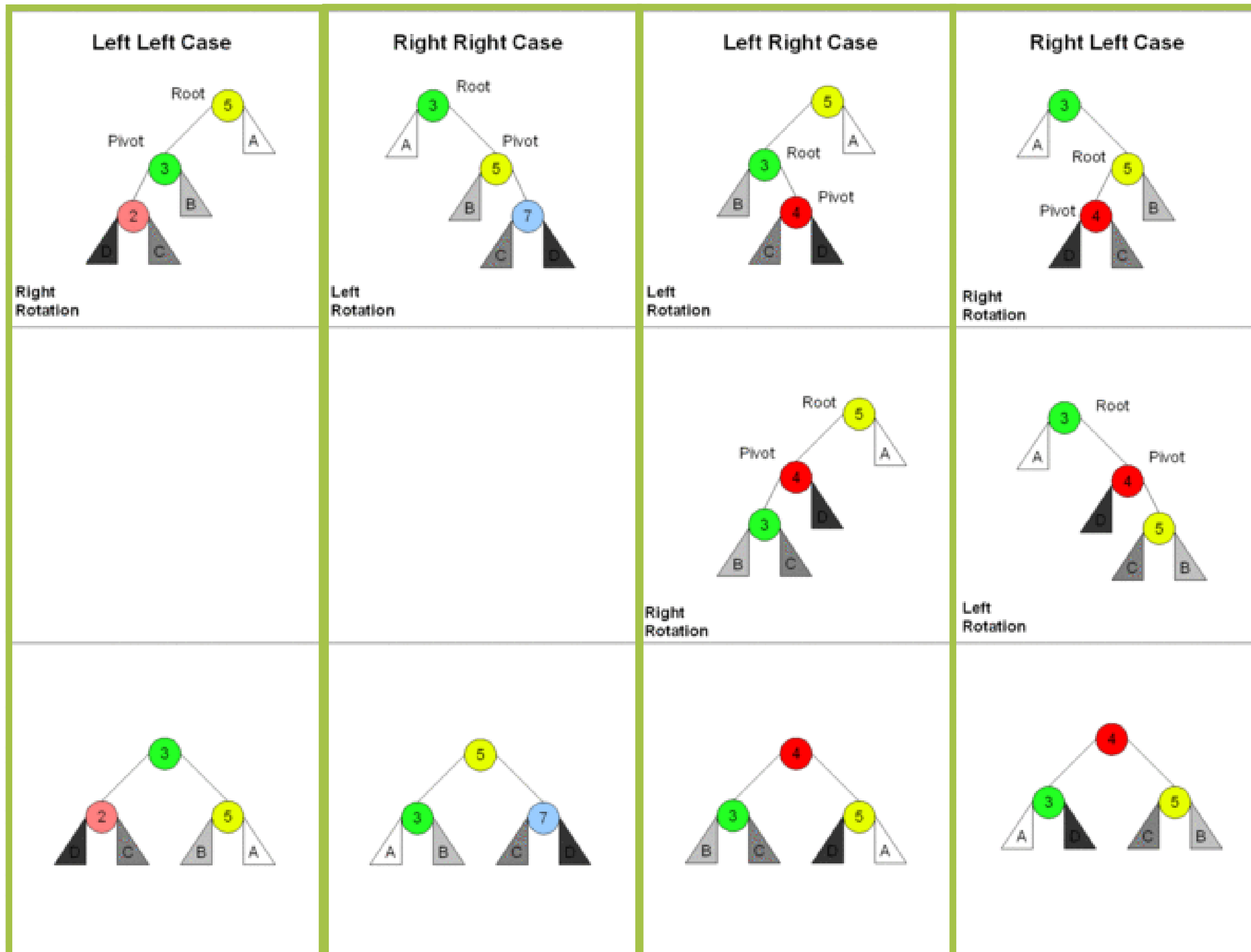
AVL Trees (6/8)
Διπλή περιστροφή







AVL Trees (7/8)
Περιστροφές

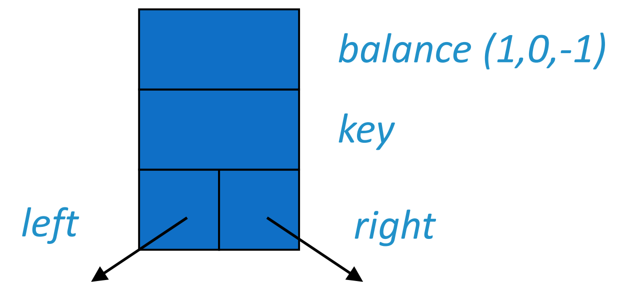


AVL Trees (8/8)
ΨΕΥΔΟΚΩΔΙΚΑΣ ΕΠΙΛΟΓΗΣ ΠΕΡΙΣΤΡΟΦΗΣ

```
IF tree is right heavy {
    IF tree's right subtree is left heavy
        Perform Double Left rotation
    ELSE
        Perform Single Left rotation
}
ELSE IF tree is left heavy {
    IF tree's left subtree is right heavy
        Perform Double Right rotation
    ELSE
        Perform Single Right rotation
}
```

AVL Trees & C++ (1/9)

- **Αρχικά, χρειαζόμαστε μια κλάση για τους κόμβους με βάση την οποία θα δομηθεί το δέντρο:**




```
template <class KeyType>
 class AvlNode {
 private:
      Comparable<KeyType> * myData; // Data field
      AvlNode<KeyType> * mySubtree[2]; // Subtree pointers
      short myBal; // Balance factor

      // ... many details omitted
 };
```

- Παρατηρούμε ότι περιέχει ένα πίνακα για τα δύο παιδιά του κόμβου και μια μεταβλητή για τον παράγοντα ισοσκελισμού.
  - Η κλάση αυτή μας αρκεί για το δέντρο!

AVL Trees & C++ (2/9)

- **Ορίζουμε τα βοηθητικά στοιχεία σε μια κλάση Comparable**

```
enum cmp_t {
    MIN_CMP = -1, // less than
    EQ_CMP = 0, // equal to
    MAX_CMP = 1 // greater than
};
template <class KeyType>
  class Comparable {
  private:
    KeyType myKey;

  public:
    Comparable(KeyType key) : myKey(key) { };
    cmp_t Compare(KeyType key) const;
    KeyType Key() const { return myKey; }
};
```

AVL Trees & C++ (3/9)

- **Κώδικας για την αριστερή περιστροφή:**

```
enum dir_t { LEFT = 0, RIGHT = 1 };
template <class KeyType>
 void
 AvlNode<KeyType>::RotateLeft(AvlNode<KeyType> * & root) {
      AvlNode<KeyType> * oldRoot = root;
      root = root->mySubtree[RIGHT];
      oldRoot->mySubtree[RIGHT] = root->mySubtree[LEFT];
      root->mySubtree[LEFT] = oldRoot;
      // update balances
      oldRoot->myBal -= (1 + MAX(root->myBal, 0));
      root->myBal -= (1 - MIN(oldRoot->myBal, 0));
 }
```

AVL Trees & C++ (4/9)

- **Κώδικας για την δεξιά περιστροφή:**

```
template <class KeyType>
 void
 AvlNode<KeyType>::RotateRight(AvlNode<KeyType> * & root) {
	AvlNode<KeyType> * oldRoot = root;
	// perform rotation
	root = root->mySubtree[LEFT];
	oldRoot->mySubtree[LEFT] = root->mySubtree[RIGHT];
	root->mySubtree[RIGHT] = oldRoot;
	// update balances
	oldRoot->myBal += (1 - MIN(root->myBal, 0));
	root->myBal += (1 + MAX(oldRoot->myBal, 0));
 }
```

AVL Trees & C++ (5/9)

- **Ένωση των δύο προηγούμενων συναρτήσεων σε μια:**

```
template <class KeyType>
 void
 AvlNode<KeyType>::RotateOnce(AvlNode<KeyType> * & root, dir_t dir) {
	AvlNode<KeyType> * oldRoot = root;
	dir_t otherDir = Opposite(dir);
	short factor = (RIGHT - LEFT) * (1 - (2 * dir));

	// rotate
	root = tree->mySubtree[otherDir];
	oldRoot->mySubtree[otherDir] = tree->mySubtree[dir];
	root->mySubtree[dir] = oldRoot;
	// update balances
	oldRoot->myBal -= factor * (1 + MAX(factor * root->myBal, 0));
	root->myBal += factor * (1 + MAX(factor * oldRoot->myBal, 0));
}
```

AVL Trees & C++ (6/9)

- **Διπλή περιστροφή με χρήση της RotateOnce:**

```
template <class KeyType>
 void
 AvlNode<KeyType>::RotateTwice(AvlNode<KeyType> * & root, dir_t dir) {
       dir_t otherDir = Opposite(dir);
       RotateOnce(root->mySubtree[otherDir], otherDir);
       RotateOnce(root, dir);
 }
```

AVL Trees & C++ (7/9)

- **Μέθοδος για τη σύγκριση του περιεχομένου ενός κόμβου με το ζητούμενο:**

```
template <class KeyType>
  cmp_t
  AvlNode<KeyType>::Compare(KeyType key, cmp_t cmp) const {
      switch (cmp) {
          case EQ_CMP : // Standard comparison
              return myData->Compare(key);
          case MIN_CMP : // Find the minimal element in this tree
              return (mySubtree[LEFT] == NULL) ? EQ_CMP : MIN_CMP;
          case MAX_CMP : // Find the maximal element in this tree
              return (mySubtree[RIGHT] == NULL) ? EQ_CMP : MAX_CMP;
          }
}
```

AVL Trees & C++ (8/9)

- **Μέθοδος insert:**

```
template <class KeyType>
  Comparable<KeyType> *
  AvlNode<KeyType>::Insert(Comparable<KeyType> * item, AvlNode<KeyType> * & root,
int & change) {
        // See if the tree is empty
        if (root == NULL) {
                // Insert new node here
                root = new AvlNode<KeyType>(item);
                change = HEIGHT_CHANGE;
                return NULL;
        }
        // Initialize
        Comparable<KeyType> * found = NULL;
        int increase = 0;

        // Compare items and determine which direction to search
        cmp_t result = root->Compare(item->Key());
        dir_t dir = (result == MIN_CMP) ? LEFT : RIGHT;
```

AVL Trees & C++ (9/9)

- **Μέθοδος insert (συνέχεια):**

```
  if (result != EQ_CMP) {
      // Insert into "dir" subtree
      found = Insert(item, root->mySubtree[dir], change);
      if (found)
          return found; // already here - dont insert
      increase = result * change; // set balance factor increment
  }
  else { // key already in tree at this node
      increase = HEIGHT_NOCHANGE; // 0
      return root->myData;
  }

  root->myBal += increase; // update balance factor
  change = (increase && root->myBal) ? (1 - ReBalance(root)) : HEIGHT_NOCHANGE;
  return NULL; // the key was successfully inserted
}
```

Αναφορές

- Ολοκληρωμένος κώδικας AVL tree σε C++:

http://www.cmcrossroads.com/bradapp/ftp/src/libs/C++/AvlTrees.html

- Βασικές Πληροφορίες για AVL tree:

http://en.wikipedia.org/wiki/AVL_tree

- AVL με παραδείγματα:

http://www.cs.ucf.edu/~reinhard/classes/cop3503/lectures/AVLTrees02.pdf

- Standard AVL C++

http://sourceforge.net/projects/standardavl/

- Animated AVL Tree Java applet

http://webpages.ull.es/users/jriera/Docencia/AVL/AVL%20tree%20applet.htm