Δένδρα & Αλγόριθμοι Γράφων

Βιβλιογραφία

- Γεωργιάδης, Νικολόπουλος, Παληός: «Δομές Δεδομένων» Κάλλιπος, Αθήνα, 2016 http://hdl.handle.net/11419/6217
- Νικολόπουλος, Γεωργιάδης, Παληός: «Αλγοριθμική Θεωρία Γράφων» Κάλλιπος, Αθήνα, 2016. https://repository.kallipos.gr/
- Τσίχλας, Γούναρης, Μανωλόπουλος: «Σχεδίαση και Ανάλυση Αλγορίθμων», Κάλλιπος, 2016 https://repository.kallipos.gr/

Δομές Δεδομένων & Αλγόριθμοι

- Δυαδικά Δένδρα (binary trees)

- Δυαδικά Δένδρα Αναζήτησης (binary search trees)

Δυαδικά Δένδρα

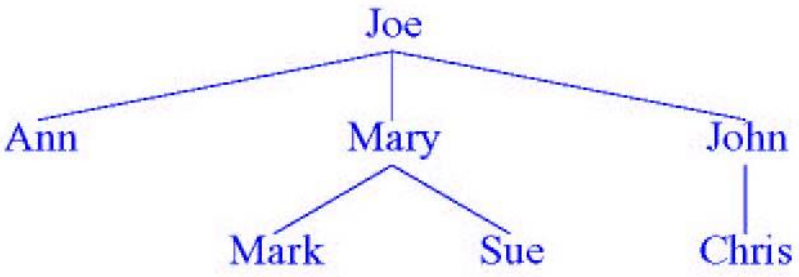
- Ορισμοί
- Λειτουργίες
- Υλοποιήσεις
- ΑΤΔ
- Εφαρμογές

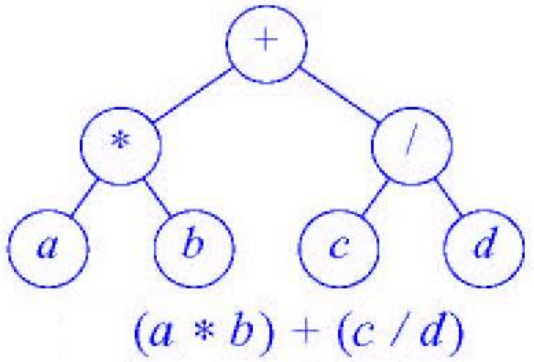
Ορισμοί

(αναδρομικός ορισμός)

Ένα δένδρο t είναι ένα πεπερασμένο μη κενό σύνολο στοιχείων. Ένα από τα στοιχεία αυτά ονομάζεται ρίζα, ενώ τα υπόλοιπα στοιχεία (αν υπάρχουν) επιμερίζονται σε δένδρα που ονομάζονται υποδένδρα του t

- Βαθμός ενός στοιχείου είναι ο αριθμός των παιδιών που έχει
- Βαθμός του δένδρου είναι ο μέγιστος βαθμός των στοιχείων του





Ορισμοί (συνέχεια)

(αναδρομικός ορισμός)

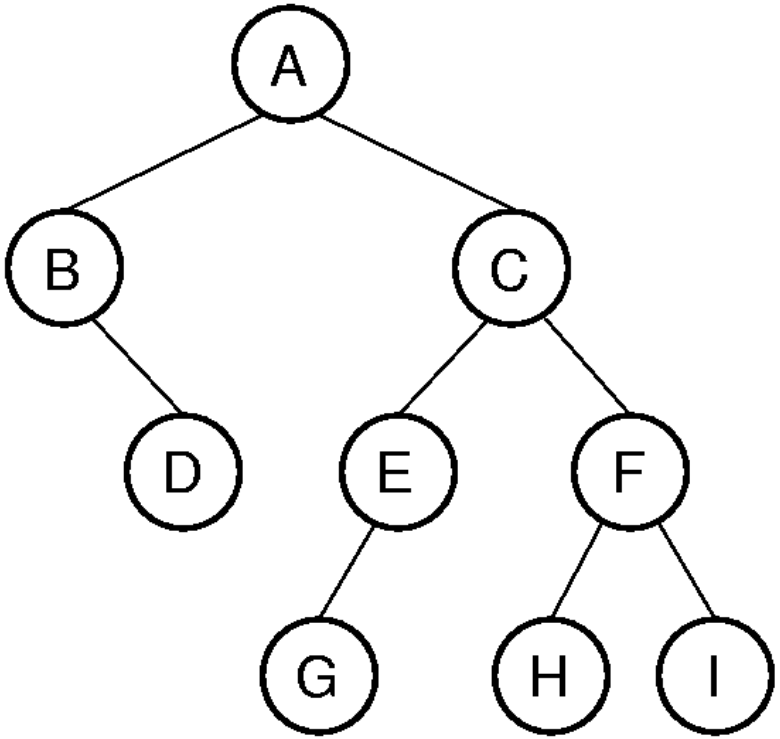
Ένα δυαδικό δένδρο t είναι μία πεπερασμένη (πιθανώς κενή) συλλογή στοιχείων. Όταν το δυαδικό δένδρο δεν είναι κενό, τότε έχει μία ρίζα και τα υπόλοιπα στοιχεία (αν υπάρχουν) επιμερίζονται σε δύο δυαδικά δένδρα που ονομάζονται το αριστερό και το δεξιό υποδένδρο του t

Ιδιότητες:

- Το σχέδιο ενός δυαδικού δένδρου με n στοιχεία ($n > 0$) έχει ακριβώς n-1 ακμές
- Ένα δυαδικό δένδρο ύψους h ($h \geq 0$) έχει τουλάχιστον h και το πολύ 2^h-1 στοιχεία

Ορισμοί (συνέχεια)

- στοιχείο (κόμβος)
- ακμή
- υποδένδρο
- παιδιά, γονιός, πρόγονος, απόγονος
- μονοπάτι
- ύψος (βάθος)
- επίπεδο
- φύλλα, εσωτερικοί κόμβοι



Πλήρη και Συμπληρωμένα Δυαδικά Δένδρα

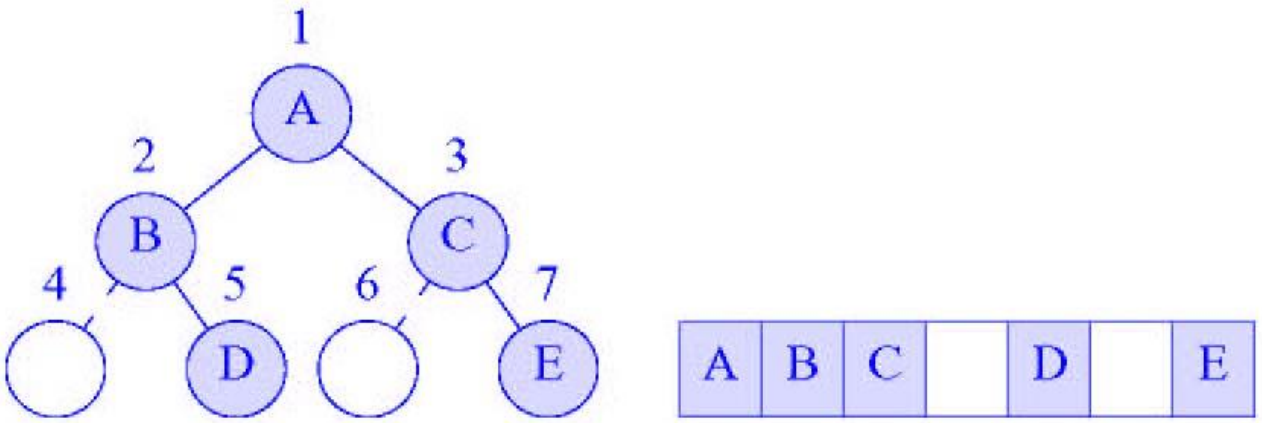
- Πλήρες (full) δυαδικό δένδρο ύψους h είναι εκείνο το δυαδικό δένδρο, το οποίο περιέχει ακριβώς 2^h-1 στοιχεία
  - Μπορούμε να αριθμήσουμε από 1 έως 2^h-1 τα στοιχεία ενός πλήρους δυαδικού δένδρου ύψους h, ξεκινώντας από το επίπεδο 1 προς το επίπεδο h και, μέσα σε κάθε επίπεδο, από αριστερά προς τα δεξιά
- Συμπληρωμένο (complete) δυαδικό δένδρο ύψους h είναι εκείνο το δυαδικό δένδρο, το οποίο προκύπτει από ένα πλήρες δυαδικό δένδρο ύψους h εάν διαγράψουμε k στοιχεία με αρίθμηση 2^h-i, για $1 \leq i \leq k$ ($k \geq 0$)

Ιδιότητες πλήρων / συμπληρωμένων δυαδικών δένδρων

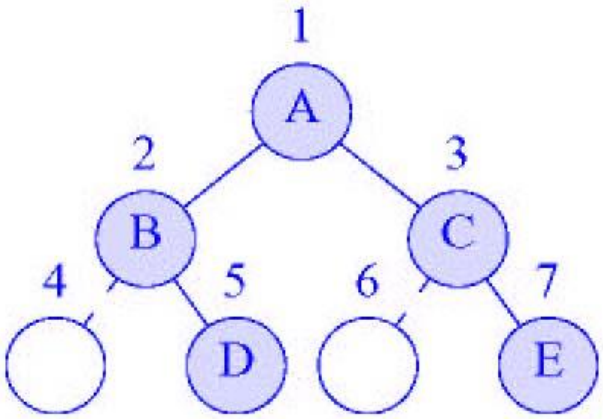
- Το πλήθος των φύλλων σε ένα μη άδειο πλήρες δυαδικό δένδρο είναι κατά 1 μεγαλύτερο από το πλήθος των εσωτερικών κόμβων (2^{h-1} και $2^{h-1}-1$, αντίστοιχα)
- Έστω i ($1 \le i \le n$) ο αριθμός ενός στοιχείου ενός συμπληρωμένου δυαδικού δένδρου:
  - Αν i = 1, το στοιχείο είναι η ρίζα, αλλιώς είναι παιδί του κόμβου με αριθμό $\lfloor i/2 \rfloor$
  - Το αριστερό του παιδί έχει αριθμό 2i (αν $2i \le n$, αλλιώς δεν έχει αριστερό παιδί)
  - Το δεξιό του παιδί έχει αριθμό 2i+1 (αν $2i+1 \le n$, αλλιώς δεν έχει δεξιό παιδί)
- Αποδείξεις με μαθηματική επαγωγή

Στατική αναπαράσταση δυαδικού δένδρου (με πίνακα)

- Βασίζεται στην προηγούμενη ιδιότητα (των συμπληρωμένων δυαδικών δένδρων)
- Πρόβλημα: σπατάλη χώρου όταν λείπουν πολλά στοιχεία (για να γίνει το δένδρο πλήρες)
  - Ένα δένδρο με n στοιχεία θα μπορούσε να απαιτήσει πίνακα μεγέθους μέχρι και 2^n-1 (η περίπτωση των δεξιών λοξών δυαδικών δένδρων)



Υλοποίηση δυαδικού δένδρου με πίνακα



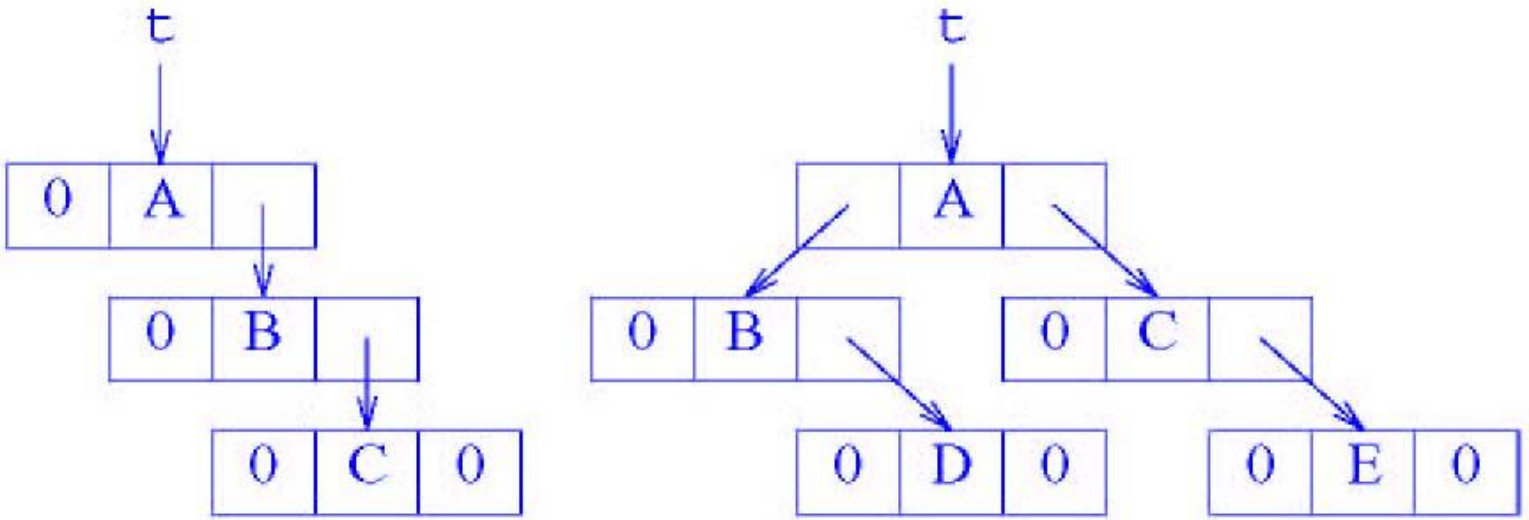
A	B	C		D		E

Θέση	1	2	3	4	5	6	7
Γονιός	--	1	1	2	2	3	3
Αριστερό παιδί	2	4	6	--	--	--	--
Δεξί παιδί	3	5	7	--	--	--	--
Αριστερός αδελφός	--	--	2	--	4	--	6
Δεξιός αδελφός	--	3	--	5	--	7	--

(οι μαθηματικοί τύποι προκύπτουν από την ιδιότητα των συμπληρω-μένων δυαδικών δένδρων)

Συνδεδεμένη αναπαράσταση δυαδικού δένδρου (με δείκτες)

- Η πιο δημοφιλής υλοποίηση
- Κάθε στοιχείο αντιστοιχεί σε ένα κόμβο με ένα πεδίο δεδομένων (data) και δύο πεδία συνδέσμων (LeftChild, RightChild)



Κλάση κόμβου για συνδεδεμένα δυαδικά δένδρα

```
class BinaryTreeNode {
  public:
      BinaryTreeNode() {LeftChild = RightChild = 0;}
      BinaryTreeNode(const T& e)
            {data = e; LeftChild = RightChild = 0;}
      BinaryTreeNode(const T& e, BinaryTreeNode *l,
                     BinaryTreeNode *r)
       {data = e; LeftChild = l; RightChild = r;}
   private:
      T data;
      BinaryTreeNode<T> *LeftChild,  // left subtree
                        *RightChild; // right
  subtree
}
```

Πράξεις πάνω σε δυαδικά δένδρα

- Προσδιορισμός ύψους, πλήθους στοιχείων
- Δημιουργία αντιγράφου
- Παρουσίαση δένδρου σε μονάδα εξόδου
- Διαγραφή δένδρου
- Υπολογισμός έκφρασης (αν είναι δένδρο έκφρασης)

Όλα τα παραπάνω εκτελούνται με συστηματικό τρόπο με τη λειτουργία διάσχισης του δένδρου

Διάσχιση δένδρου

- Κάθε διαδικασία επίσκεψης όλων των κόμβων ενός δένδρου, ακριβώς μια φορά τον καθένα, ονομάζεται διάσχιση (traversal).
  - **Προδιατεταγμένη** διάσχιση (preorder): Για κάθε κόμβο, επισκεπτόμαστε πρώτα τον ίδιο τον κόμβο, έπειτα τους κόμβους του αριστερού του υποδένδρου και στη συνέχεια τους κόμβους του δεξιού του υποδένδρου.
  - **Μεταδιατεταγμένη** διάσχιση (postorder): Για κάθε κόμβο, επισκεπτόμαστε πρώτα τους κόμβους του αριστερού του υποδένδρου, έπειτα τους κόμβους του δεξιού του υποδένδρου και στη συνέχεια τον ίδιο τον κόμβο.

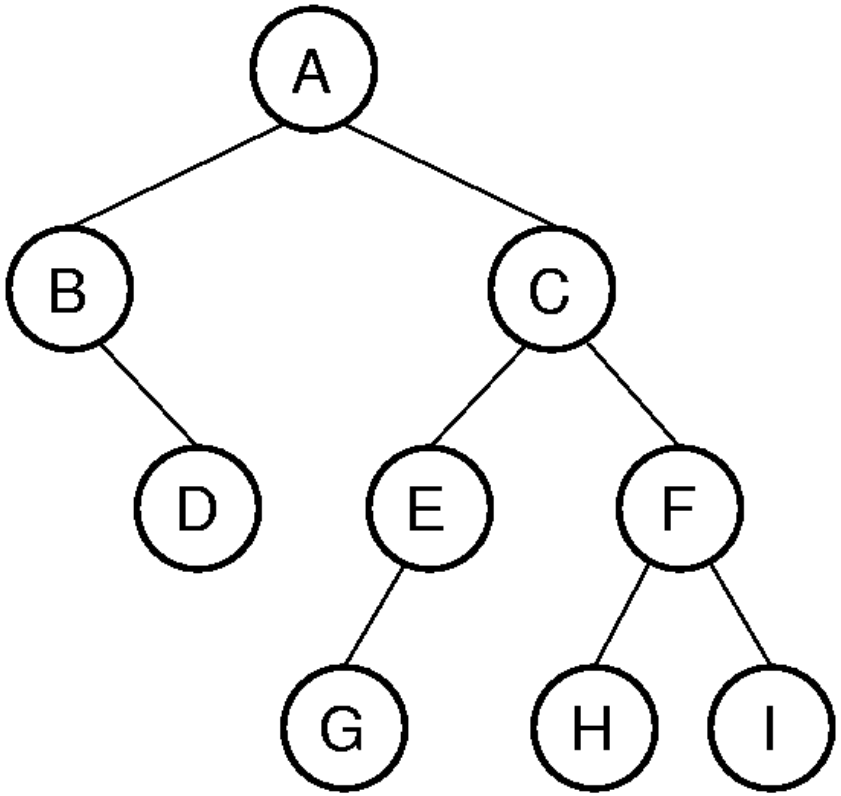
Διάσχιση δένδρου

- **Ενδοδιατεταγμένη** διάσχιση (inorder): Για κάθε κόμβο, επισκεπτόμαστε πρώτα τους κόμβους του αριστερού του υποδένδρου, έπειτα τον ίδιο τον κόμβο και στη συνέχεια τους κόμβους του δεξιού του υποδένδρου.
- **Κατά σειρά επιπέδων** (level order): Επισκεπτόμαστε τους κόμβους κατά επίπεδα, από πάνω (τη ρίζα) προς τα κάτω, και μέσα σε ένα επίπεδο από αριστερά προς τα δεξιά.

- Οι 3 πρώτες μέθοδοι είναι αναδρομικές ενώ η $4^{η}$ είναι επαναληπτική

Διάσχιση δένδρου (συνέχεια)

- Προδιατεταγμένη:
  A, B, D, C, E, G, F, H, I
- Μεταδιατεταγμένη:
  D, B, G, E, H, I, F, C, A
- Ενδοδιατεταγμένη:
  B, D, A, G, E, C, H, F, I
- Κατά σειρά επιπέδων:
  A, B, C, D, E, F, G, H, I



Αναδρομικές μέθοδοι διάσχισης

Το διασχιζόμενο δυαδικό δέντρο αναπαριστάται με το συνδεδεμένο σχήμα που παρουσιάσθηκε προηγουμένως και το BinaryTreeNode ορίζεται ως μια πρότυπη δομή ή κλάση

```
void PreOrder(BinaryTreeNode<T> *t)
{// Preorder traversal of *t.
   if (t) {Visit(t); PreOrder(t->LeftChild);
      PreOrder(t->RightChild); }
}

void InOrder(BinaryTreeNode<T> *t)
{// Inorder traversal of *t.
   if (t) {InOrder(t->LeftChild);
      Visit(t); InOrder(t->RightChild); }
}

void PostOrder(BinaryTreeNode<T> *t)
{// Postorder traversal of *t.
   if (t) {PostOrder(t->LeftChild);
      PostOrder(t->RightChild); Visit(t); }
}
```

Διάσχιση κατά σειρά επιπέδων

Επισκεπτόμαστε τα στοιχεία κατά επίπεδα από επάνω προς τα κάτω.

```
void LevelOrder(BinaryTreeNode<T> *t)
{// Level-order traversal of *t.
   LinkedQueue<BinaryTreeNode<T>*> Q;
   while (t) {
      Visit(t);   // visit t
      // put t's children on queue
      if (t->LeftChild) Q.Add(t->LeftChild);
      if (t->RightChild) Q.Add(t->RightChild);
      // get next node to visit
      try {Q.Delete(t);}
      catch (OutOfBounds) {return;}
   }
}
```

Χρησιμοποιείται η συνδεδεμένη λίστα κλάση LinkedQueue. Όπου τα στοιχεία της ουράς είναι δείκτες προς κόμβους του δυαδικού δέντρου. Όταν το δέντρο δεν είναι κενό εισέρχεται μέσα στο while.

Επισκεπτόμαστε τη ρίζα ενώ τα παιδιά εάν υπάρχουν προστίθενται στην ουρά. Μετά την προσθήκη των παιδιών της t στην ουρά , προσπαθούμε να διαγράψουμε ένα στοιχείο από την ουρά

Αφηρημένος Τύπος Δεδομένων ΑΤΔ BinaryTree

AbstractDataType *BinaryTree* {

instances: collection of elements; if not empty, the collection is partitioned into a root, left subtree, and right subtree; each subtree is also a binary tree;

operations

Create (): create an empty binary tree

IsEmpty (): return `true` if the tree is empty, return `false` otherwise

Root (*x*): *x* is set to root element; return `false` if the operation fails, return `true` otherwise

MakeTree (*root*, *left*, *right*): create a binary tree with *root* as the root element, *left* (*right*) as the left (right) subtree.

BreakTree (*root*, *left*, *right*): inverse of create

PreOrder: preorder traversal of binary tree

InOrder: inorder traversal of binary tree

PostOrder: postorder traversal of binary tree

LevelOrder: level-order traversal of binary tree

}

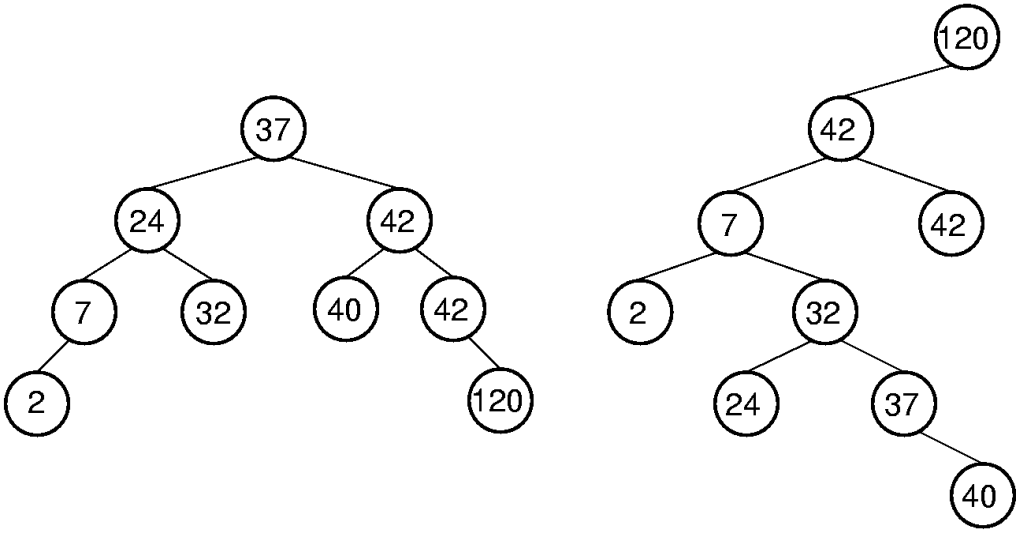
Δυαδικά Δένδρα Αναζήτησης

- Ορισμοί
- Λειτουργίες
- Υλοποιήσεις
- ΑΤΔ
- Εφαρμογές

Δυαδικό Δένδρο Αναζήτησης (ΔΔΑ)

Ορισμός: Ένα ΔΔΑ είναι δυαδικό δένδρο με διακριτά κλειδιά (τιμές) και τις εξής ιδιότητες:

- Τα κλειδιά (αν υπάρχουν) στο αριστερό υποδένδρο της ρίζας είναι μικρότερα από το κλειδί της ρίζας
- Τα κλειδιά (αν υπάρχουν) στο δεξιό υποδένδρο της ρίζας είναι μεγαλύτερα από το κλειδί της ρίζας
- Το αριστερό και το δεξιό υποδένδρο είναι επίσης ΔΔΑ



Κίνητρο:

να μειώσουμε τους χρόνους ενημέρωσης και αναζήτησης σε λιγότερο από Θ(n)

ΑΤΔ για Δυαδικό Δένδρο Αναζήτησης (ΔΔΑ)
ΑΤΔ BinarySearchTree

AbstractDataType *BSTree* {

instances: binary trees, each node has an element with a key field; all keys are distinct; keys in the left subtree of any node smaller than the key in the node; those in the right subtree are larger;

operations

Create (): create an empty binary search tree

Search (*k*, *e*): return in *e* the element with key *k*; return `false` if the operation fails, return `true` if it succeeds

Insert (*e*): insert element *e* into the search tree

Delete (*k*, *e*): delete the element with key *k* and return it in *e*

Ascend (): Output all elements in ascending order of key

}

Αναζήτηση στοιχείου μέσα σε ΔΔΑ

```
bool BSTree<E,K>::Search(const K& k, E &e) const
{// Search for element that matches k.
   // pointer p starts at the root and moves through
   // the tree looking for an element with key k
   BinaryTreeNode<E> *p = root;
   while (p)                          // examine p->data
      if (k < p->data) p = p->LeftChild;
      else if (k > p->data) p = p->RightChild;
           else {                     // found element
                 e = p->data;
                 return true;}
   return false;
}
```

Κόστος αναζήτησης = κόστος κατάβασης = O(h)

Προκειμένου να αναζητήσουμε ένα στοιχείο με κλειδί k, ξεκινάμε από τη ρίζα. Ένα η ρίζα είναι NULL, αναζήτηση δεν επιτυχής, διαφορετικά συγκρίνουμε το k με το κλειδί στη ρίζα. Εάν είναι μικρότερο ελέγχεται το αριστερό υποδέντρο κτ..τλ

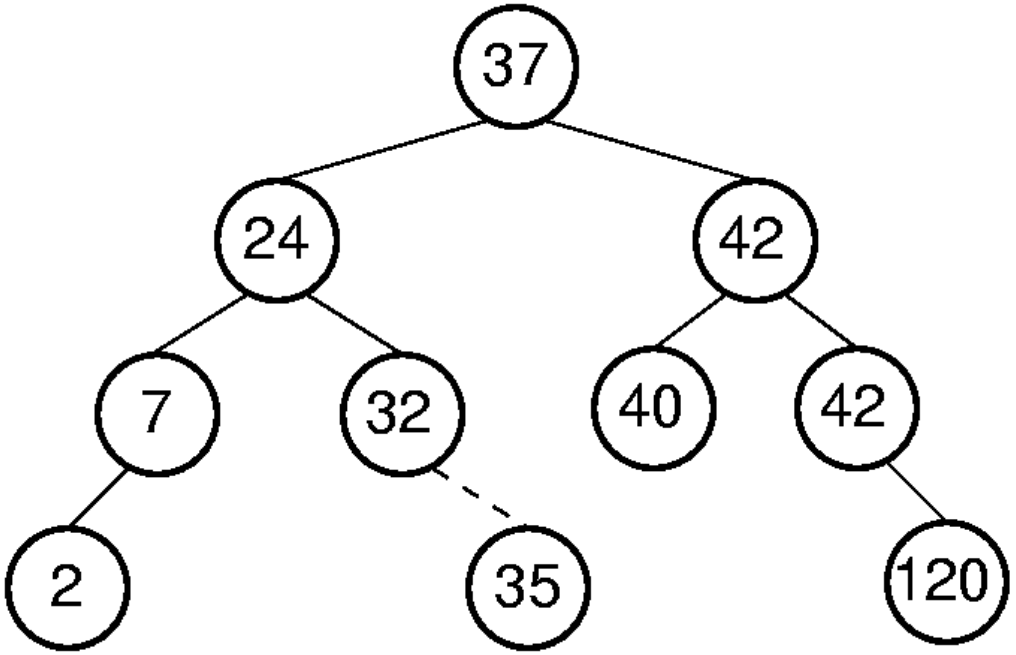
Εισαγωγή στοιχείου σε ΔΔΑ (1)

Βασική ιδιότητα ΔΔΑ:

- Η εισαγωγή γίνεται πάντα σε κάποιο (νέο) φύλλο

Διαδικασία:

- Πρώτα επιβεβαιώνουμε ότι είναι διαφορετικό από τα υπάρχοντα κλειδιά εκτελώντας αναζήτηση για ένα στοιχείο με το ίδιο κλειδί e
- Αναζήτηση του στοιχείου (οπότε καταλήγουμε σε κόμβο-φύλλο)
- Εισαγωγή του ως παιδί εκείνου του κόμβου



Εισαγωγή στοιχείου σε ΔΔΑ (2)

```
BSTree<E,K>& BSTree<E,K>::Insert(const E& e)
{// Insert e if not duplicate.
   BinaryTreeNode<E> *p = root,  // search pointer
                     *pp = 0;    // parent of p
   // find place to insert
   while (p) {                   // examine p->data
      pp = p;
      // move p to a child
      if (e < p->data) p = p->LeftChild;
      else if (e > p->data) p = p->RightChild;
           else throw BadInput(); // duplicate
      }

   // get a node for e and attach to pp
   …
```

Εισαγωγή στοιχείου σε ΔΔΑ (3)

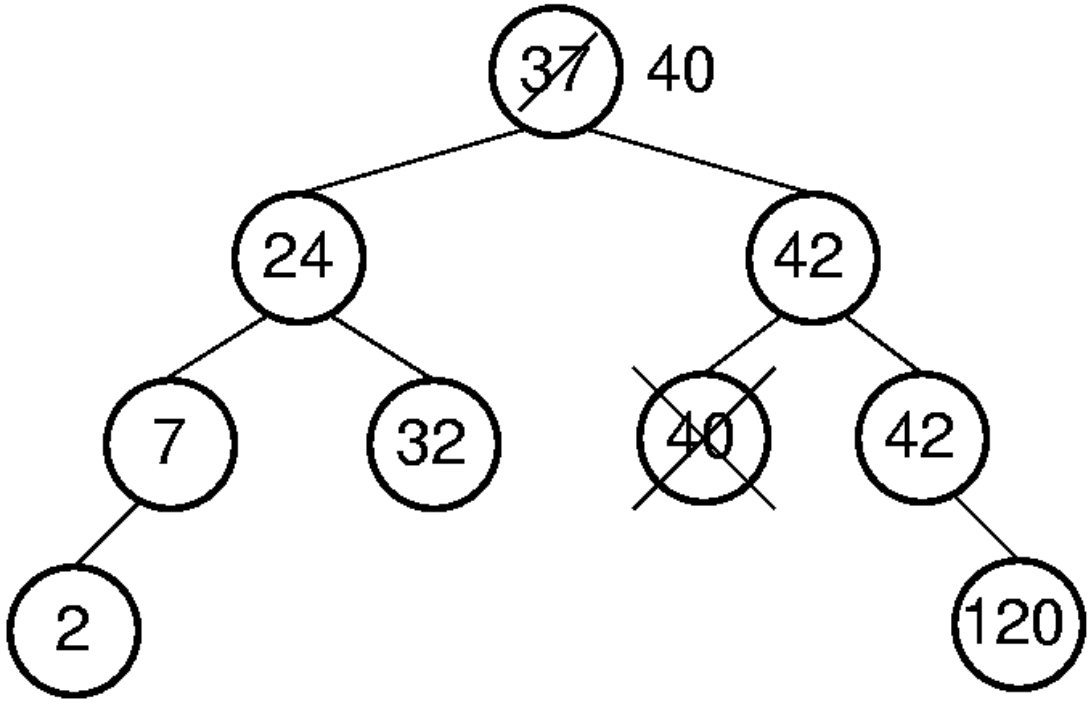
```
…
  // get a node for e and attach to pp
  BinaryTreeNode<E> *r = new
BinaryTreeNode<E> (e);
  if (root) {// tree not empty
     if (e < pp->data) pp->LeftChild = r;
     else pp->RightChild = r;}
  else // insertion into empty tree
       root = r;

  return *this;
}
```

Κόστος = Κόστος αναζήτησης + κόστος 'συγκόλλησης'
νέου κόμβου στον πατέρα-κόμβο = O(h) + O(1) = O(h)

Διαγραφή στοιχείου από ΔΔΑ (1)

3 περιπτώσεις:

- Ο κόμβος p (που περιέχει το στοιχείο) είναι φύλλο
- Το p έχει μόνο ένα μη κενό υποδένδρο
- Το p έχει ακριβώς δύο μη κενά υποδένδρα *(αντικαθιστούμε το στοιχείο αυτό είτε με το μεγαλύτερο στοιχείου του αριστερού υποδέντρου είτε με το μικρότερο του δεξιού υποδέντρου)*



Διαγραφή στοιχείου από ΔΔΑ (2)

Όταν διαγράφουμε κόμβο με δύο μη κενά υποδέντρα, αυτός ο κώδικας πάντοτε χρησιμοποιεί το μεγαλύτερο στοιχείο του αριστερού υποδέντρου για αντικατάσταση

```
BSTree<E,K>& BSTree<E,K>::Delete(const K& k, E& e)
{// Delete element with key k and put it in e.

   // set p to point to node with key k
   BinaryTreeNode<E> *p = root, // search pointer
                     *pp = 0;   // parent of p
   while (p && p->data != k){// move to a child of p
      pp = p;
      if (k < p->data) p = p->LeftChild;
      else p = p->RightChild;
      }
   if (!p) throw BadInput(); // no element with key k

   e = p->data;  // save element to delete
   …
```

Διαγραφή στοιχείου από ΔΔΑ (3)

```
...
 // restructure tree
 // handle case when p has two children
 if (p->LeftChild && p->RightChild) {// two children
    // convert to zero or one child case
    // find largest element in left subtree of p
    BinaryTreeNode<E> *s = p->LeftChild,
                      *ps = p;  // parent of s
    while (s->RightChild) {// move to larger element
       ps = s;
       s = s->RightChild;}

    // move largest from s to p
    p->data = s->data;
    p = s;
    pp = ps;}
...
```

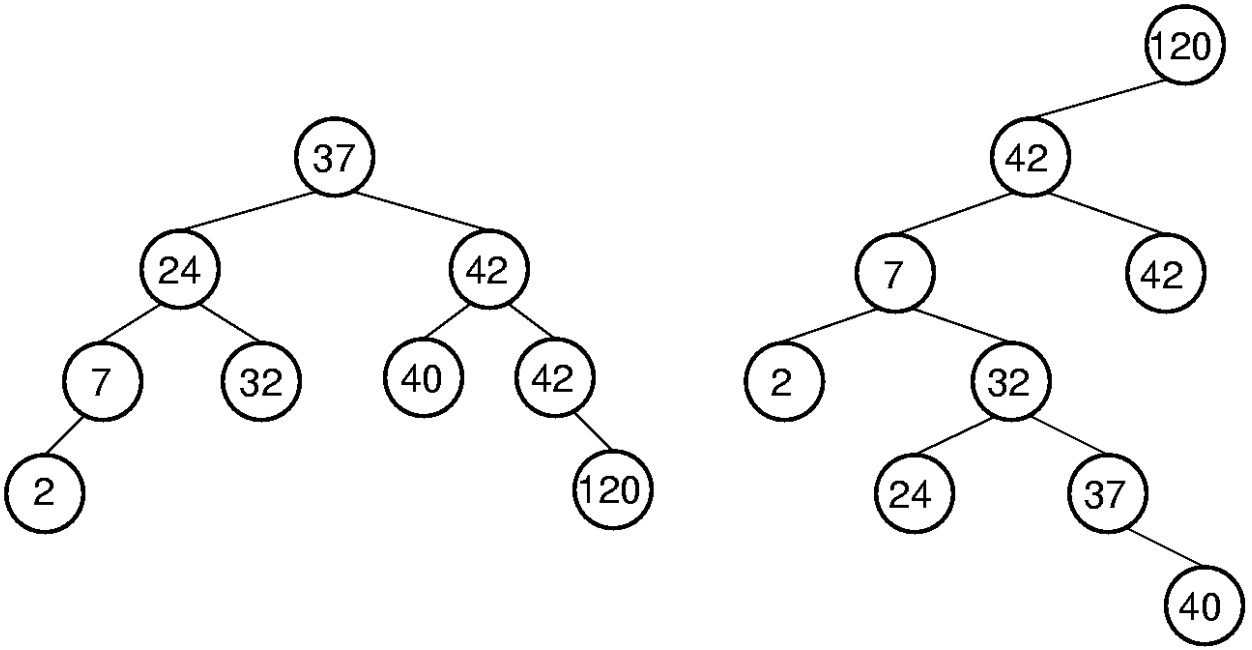
Διαγραφή στοιχείου από ΔΔΑ (4)

```
   ...
   // p has at most one child
   // save child pointer in c
   BinaryTreeNode<E> *c;
   if (p->LeftChild) c = p->LeftChild;
   else c = p->RightChild;

   // delete p
   if (p == root) root = c;
   else {// is p left or right child of pp?
         if (p == pp->LeftChild)
               pp->LeftChild = c;
         else pp->RightChild = c;}
   delete p;

   return *this;
}
```

Κόστος = ?

Υψος ΔΔΑ

- Το ύψος ενός ΔΔΑ με n στοιχεία μπορεί να φτάσει μέχρι και n.
- Στη γενική περίπτωση όμως (όταν οι εισαγωγές και οι διαγραφές γίνονται τυχαία), το ύψος είναι O(log*n*) κατά μέσο όρο.



Τυχαίο Δένδρο

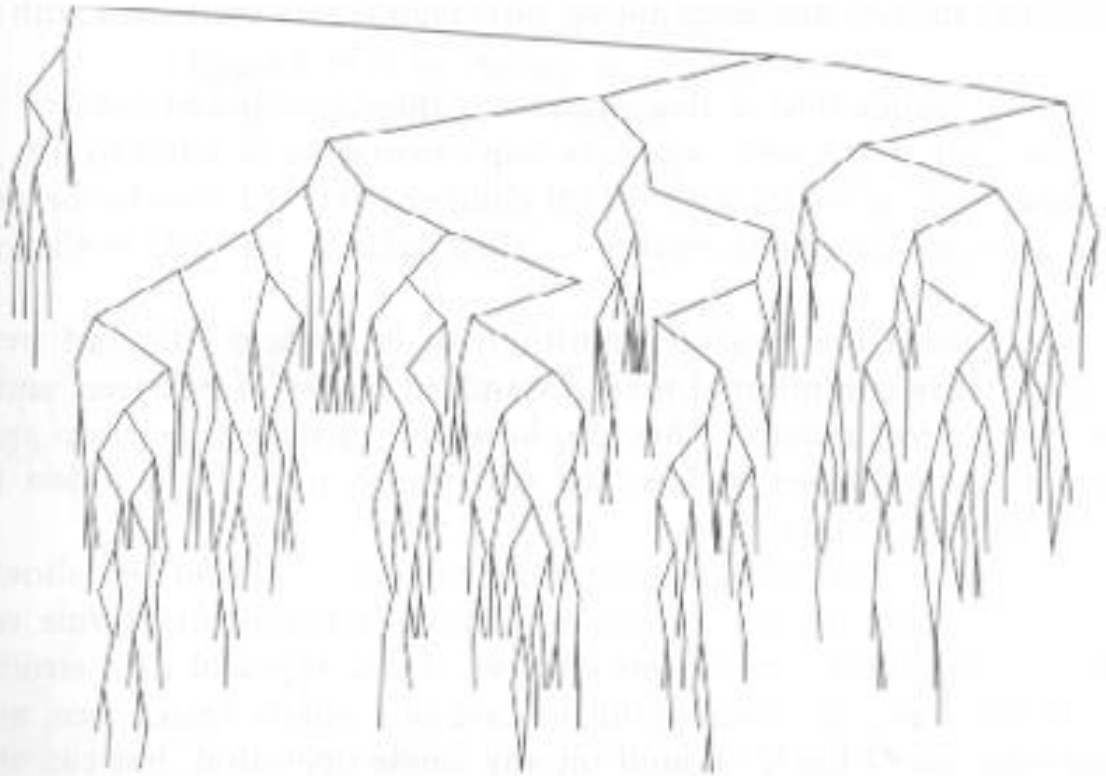
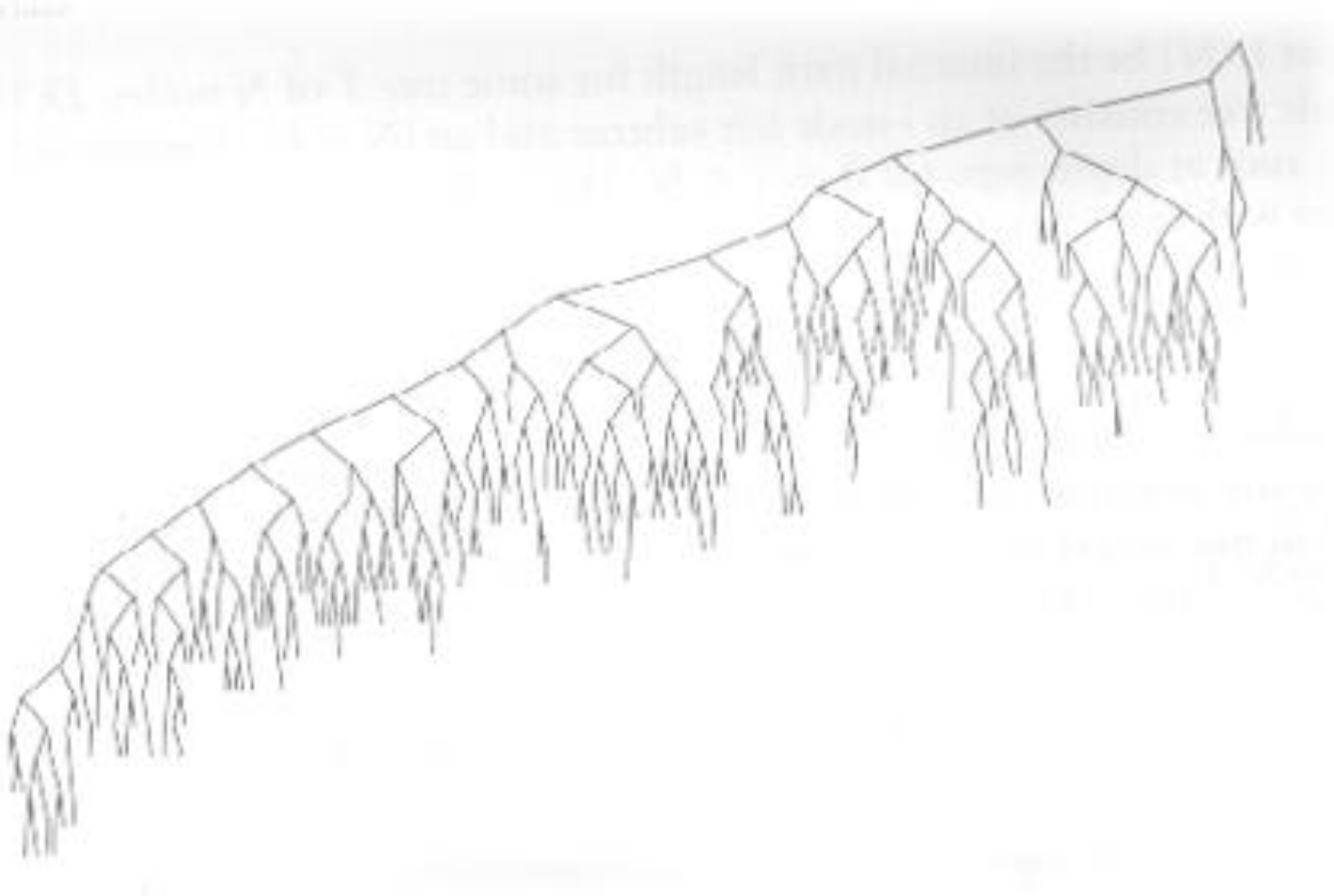
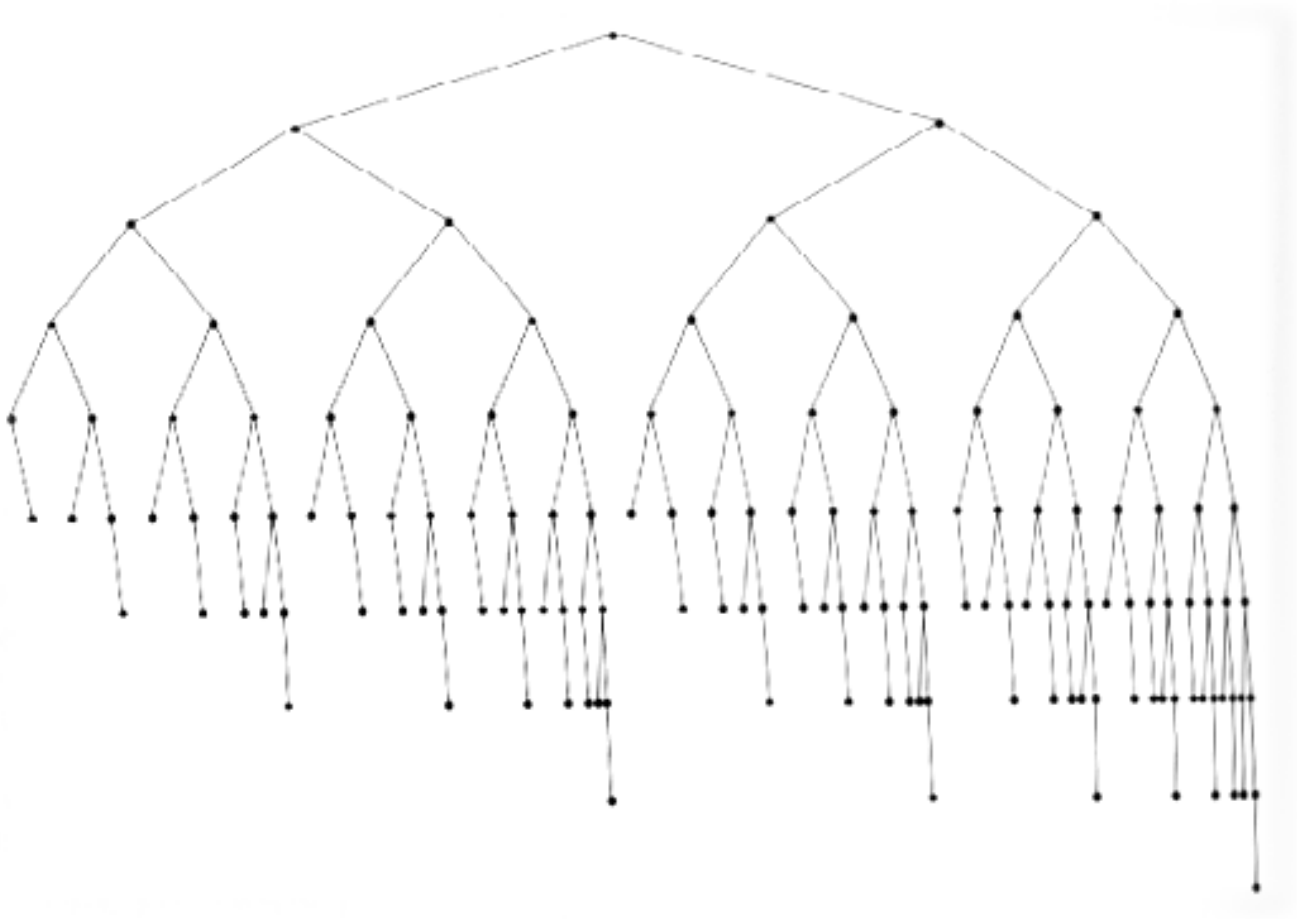


Figure 4.29 A randomly generated binary search tree

Απαράδεκτο !

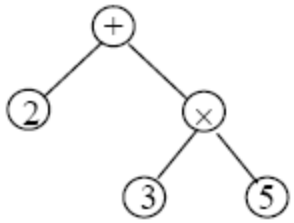


Ένα Καλύτερο Δένδρο



Παράδειγμα

- Μια αλγεβρική παράσταση που περιέχει ακέραιους και τους τελεστές +, ×, μπορεί να απεικονιστεί ως ένα δυαδικό δένδρο. Στο οποίο κάθε κόμβος μπορεί να είναι είτε φύλλο που περιέχει κάποιο ακέραιο, είτε να είναι εσωτερικός κόμβος που περιέχει έναν τελεστή (+, ×) και τα παιδιά του αντιστοιχούν στους τελεστέους του τελεστή.
- Για παράδειγμα, η ένθετη παράσταση 2+3×5 απεικονίζεται με το δυαδικό δένδρο



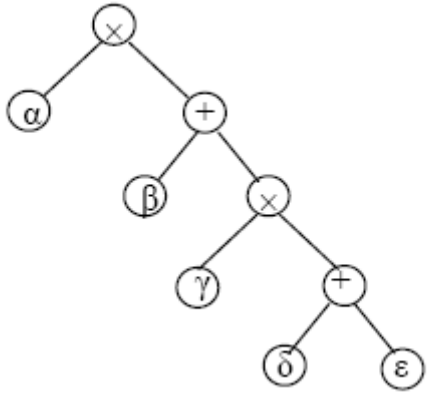
α) Να απεικονίσετε τις πιο κάτω ένθετες παραστάσεις με δένδρα.

(ι) $\alpha \times (\beta + \gamma \times (\delta + \varepsilon))$

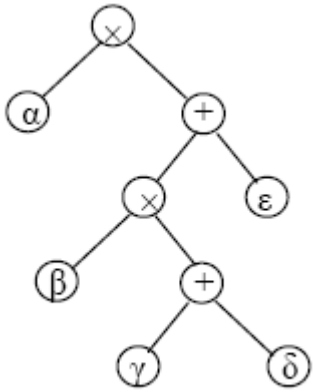
(ιι) $\alpha \times (\beta \times (\gamma + \delta) + \varepsilon)$

(β) Να γράψετε αναδρομική διαδικασία, η οποία παίρνοντας ως δεδομένο εισόδου το δείκτη στη ρίζα κάποιου δυαδικού δέντρου Τ να υπολογίζει και να επιστρέφει την τιμή της παράστασης η οποία απεικονίζεται με το δένδρο Τ. (Να περιγράψετε την υλοποίηση της δομής *δυαδικό δένδρο* την οποία θα χρησιμοποιήσετε.

Απεικόνιση της παράστασης α × (β + γ×(δ + ε))



Απεικόνιση της παράστασης α × ((β × (γ+δ))+ ε)



β) Υποθέτουμε απλοποιητικά ότι το δένδρο που αντιστοιχεί στην αλγεβρική έκφραση δεν περιέχει λάθη και ότι οι κόμβοι του δένδρου είναι υλοποιημένοι ώς εγγραφές με τρία πεδία:

```
struct Node{
        int val;
        Node *left;
        Node *right;
}
```

όπου το πεδίο val καταγράφει το στοιχείο που είναι αποθηκευμένο στον κόμβο, το πεδίο left είναι δείκτης στο αριστερό παιδί του κόμβου και το πεδίο right είναι δείκτης στο δεξιό παιδί.

Αναδρομική διαδικασία υπολογισμού της τιμής δυαδικού δένδρου

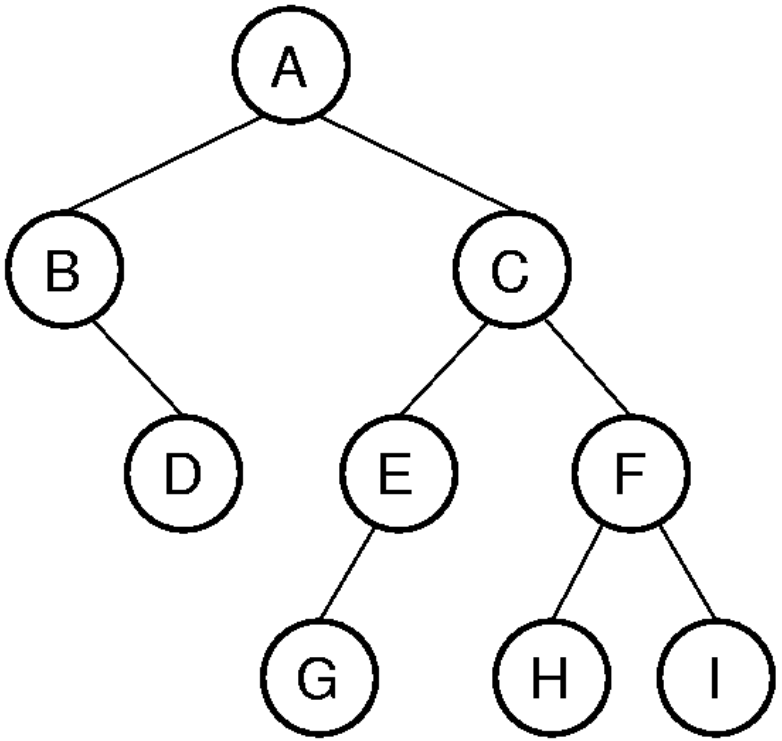
```
int Evaluate (Node *T){
if (*T).val = '×'
return (Evaluate((*T).left) · Evaluate((*T).right));
if (*T).val = '+'
return (Evaluate((*T).left) + Evaluate((*T).right));
else return ((*T).val);
}
```

Δομές Δεδομένων & Αλγόριθμοι

- **Ισοζυγισμένα Δυαδικά Δένδρα (balanced binary trees)**
- **Δένδρα Αναζήτησης m-δρόμων**
- **Πολυδιάστατα Δένδρα (multidimensional trees)**

Ορισμοί

- στοιχείο (κόμβος)
- ακμή
- υποδένδρο
- παιδιά, γονιός, πρόγονος, απόγονος
- μονοπάτι
- ύψος (βάθος)
- επίπεδο
- φύλλα, εσωτερικοί κόμβοι



Ισοζυγισμένα Δένδρα

Στόχος:

Να μην αφήσουμε το ένα κλαδί του δένδρου να γίνει κατά πολύ μεγαλύτερο του άλλου. Τα δύο υποδένδρα θέλουμε να έχουν περίπου το ίδιο ύψος.

Η Λύση

Δένδρα AVL –Ισοζυγισμένα Δέντρα

Adel'son

Vel'skii

Landis

Δένδρα AVL

Ή διαφορά των υψών του αριστερού και του δεξιού υποδένδρου δεν πρέπει να ξεπερνά το 1. Αυτό πρέπει να ισχύει αναδρομικά για όλους τους κόμβους του δένδρου.

Δένδρα AVL

Τρόπος Λειτουργίας

Η αναζήτηση είναι ίδια όπως και σε ένα απλό δυαδικό δένδρο αναζήτησης.

Μετά από κάθε εισαγωγή και διαγραφή πρέπει να γίνει έλεγχος για το αν ικανοποιείται ο περιορισμός του AVL (διαφορά των υψών των υποδένδρων).

Δένδρα AVL

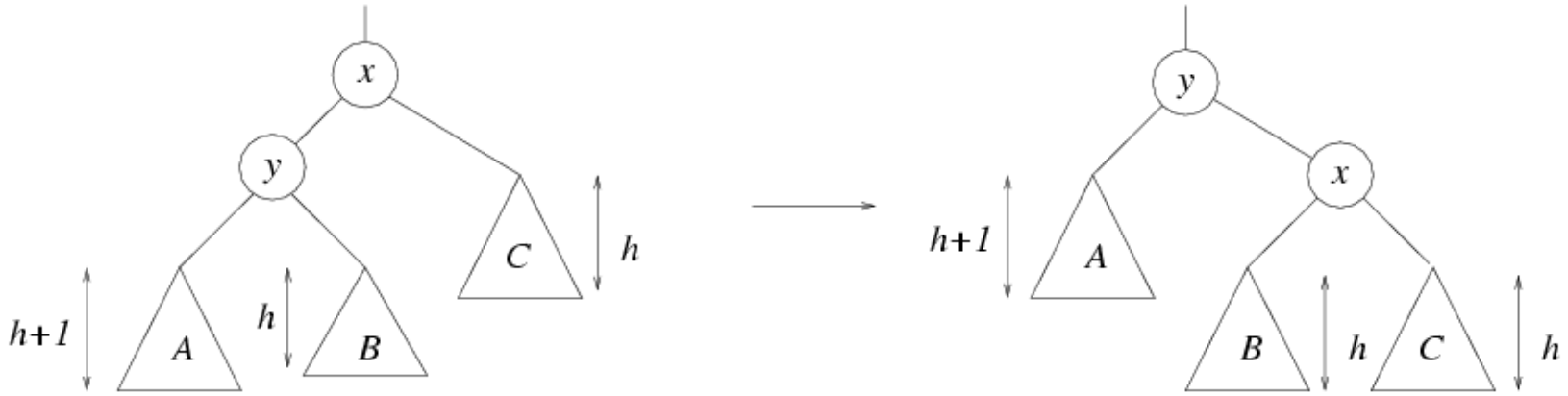
Σε περίπτωση που δεν υπάρχει πρόβλημα στη δομή του δένδρου, τότε δεν απαιτείται καμία άλλη ενέργεια.

Διαφορετικά θα πρέπει να γίνουν δομικές αλλαγές στο δένδρο ώστε να προκύψει πάλι ένα δένδρο AVL.

Περιστροφές

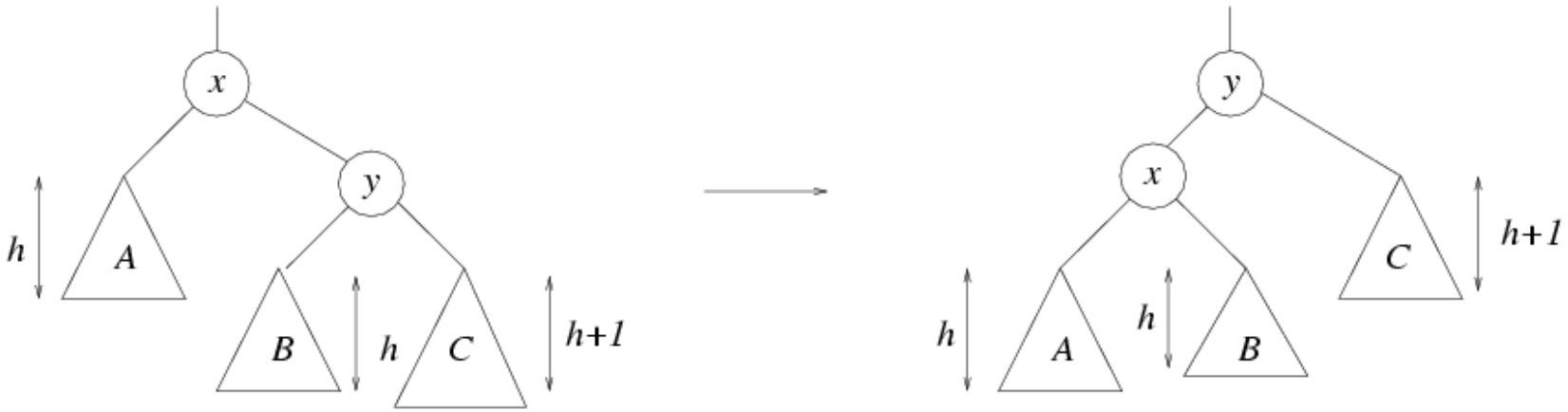
- Έχουμε δύο είδη περιστροφών
  - Απλή περιστροφή
    - Περιστρέφονται δύο κόμβοι
  - Διπλή περιστροφή
    - Περιστρέφονται τρεις κόμβοι
- Πρώτα θα δούμε τι είναι αυτές οι περιστροφές και στη συνέχεια θα εξετάσουμε τη χρήση τους

Απλή Περιστροφή



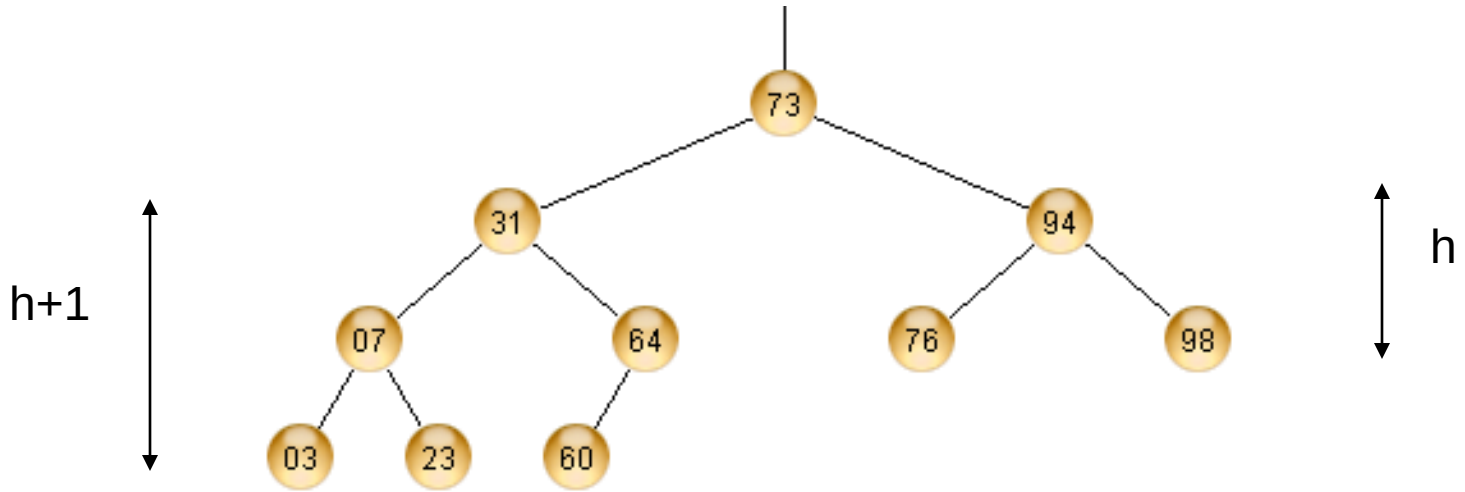
Περίπτωση 1

Απλή Περιστροφή



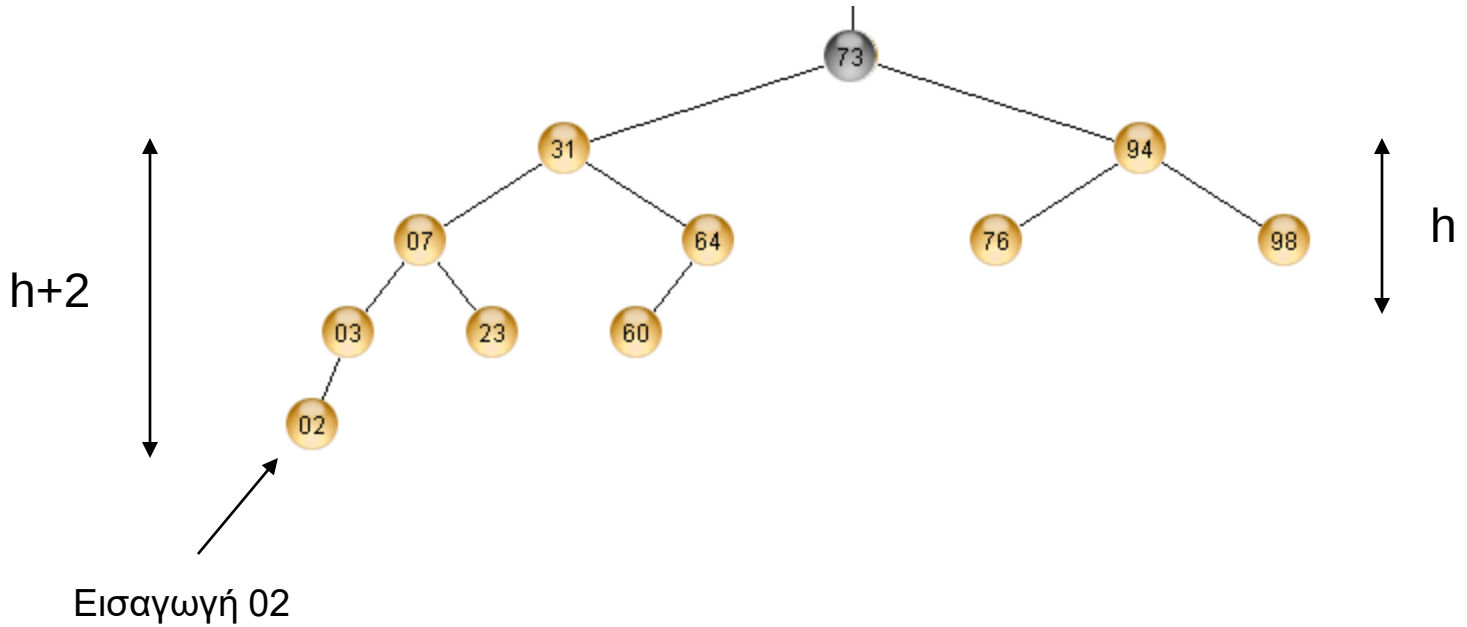
Περίπτωση 2

Απλή Περιστροφή - Παράδειγμα



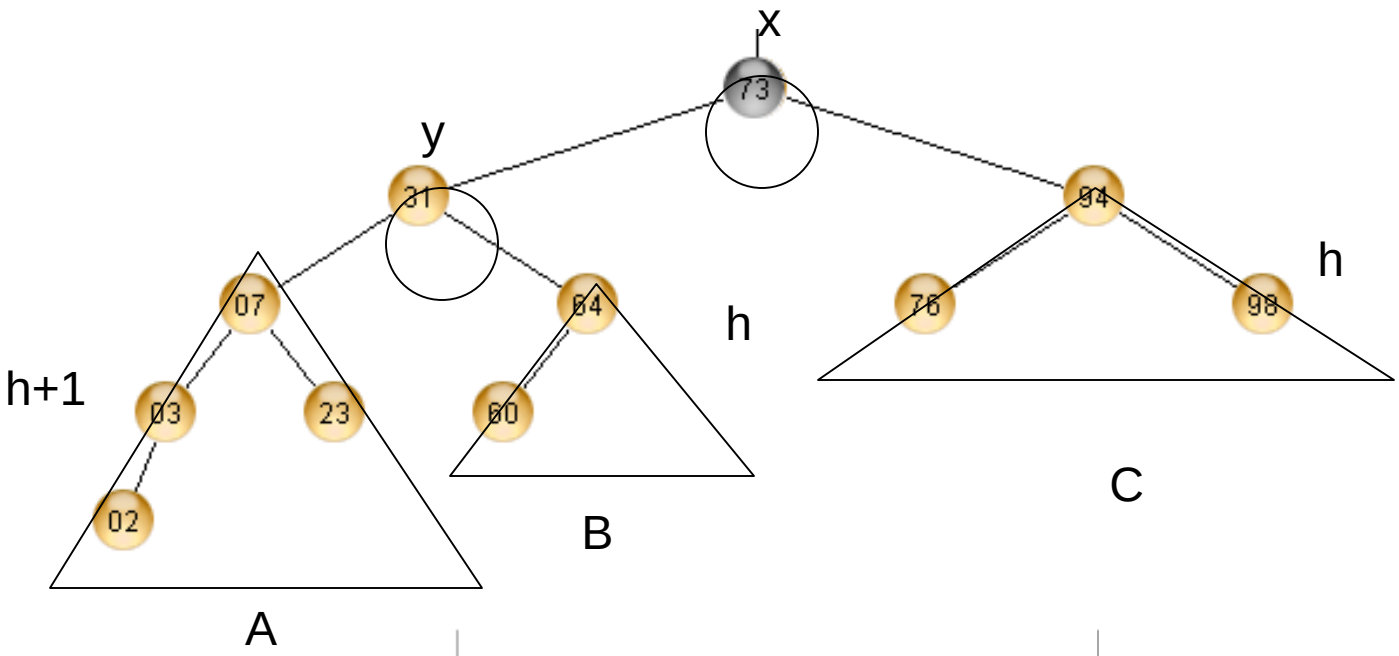
Αυτό είναι ένα δένδρο AVL

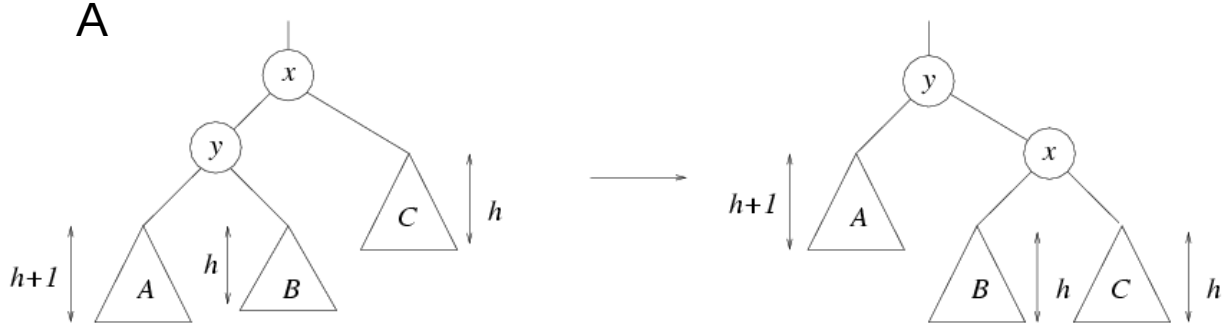
Απλή Περιστροφή - Παράδειγμα



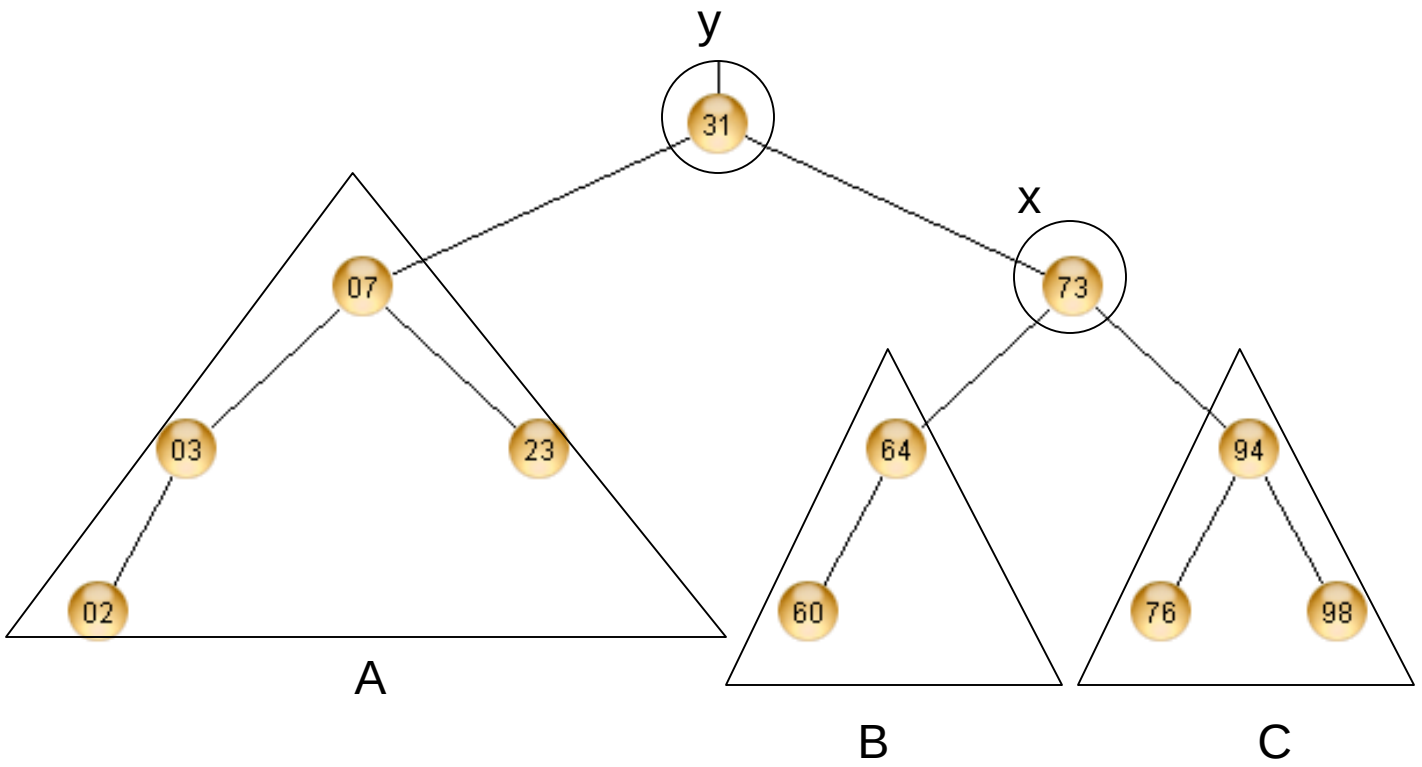
Μετά την εισαγωγή του κόμβου 2 το δένδρο που προκύπτει ΔΕΝ είναι AVL

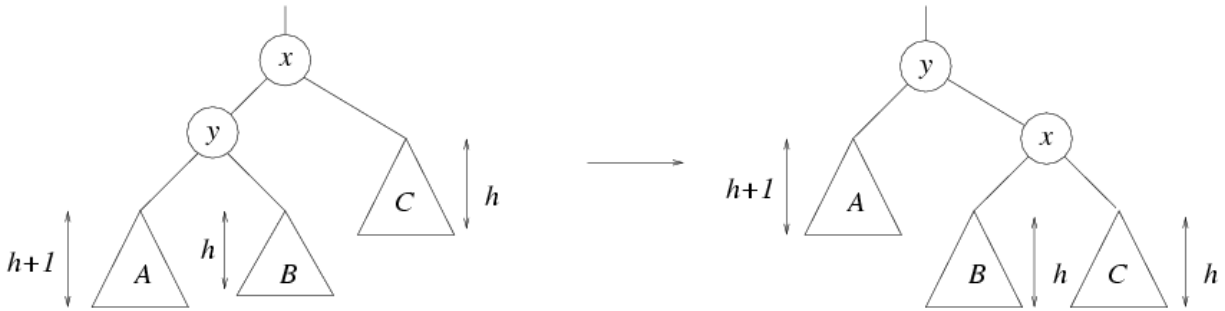
Απλή Περιστροφή - Παράδειγμα





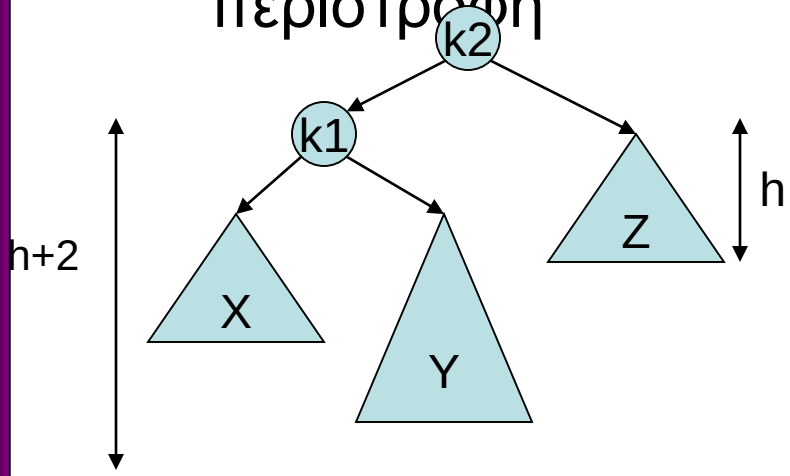
Μετά την Περιστροφή

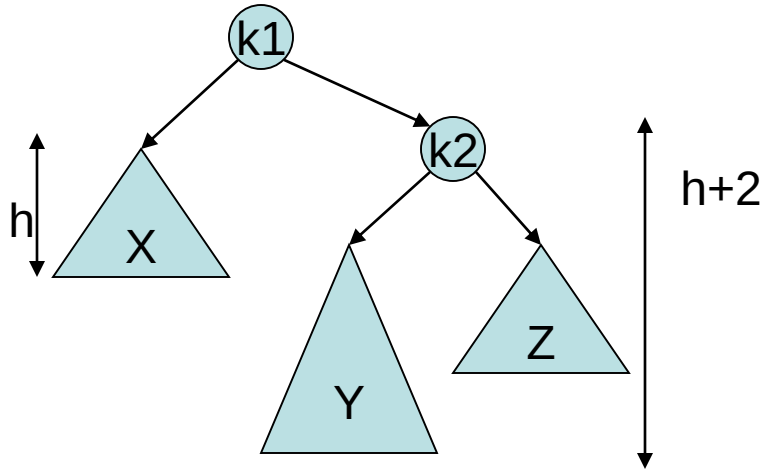




Απλή Περιστροφή

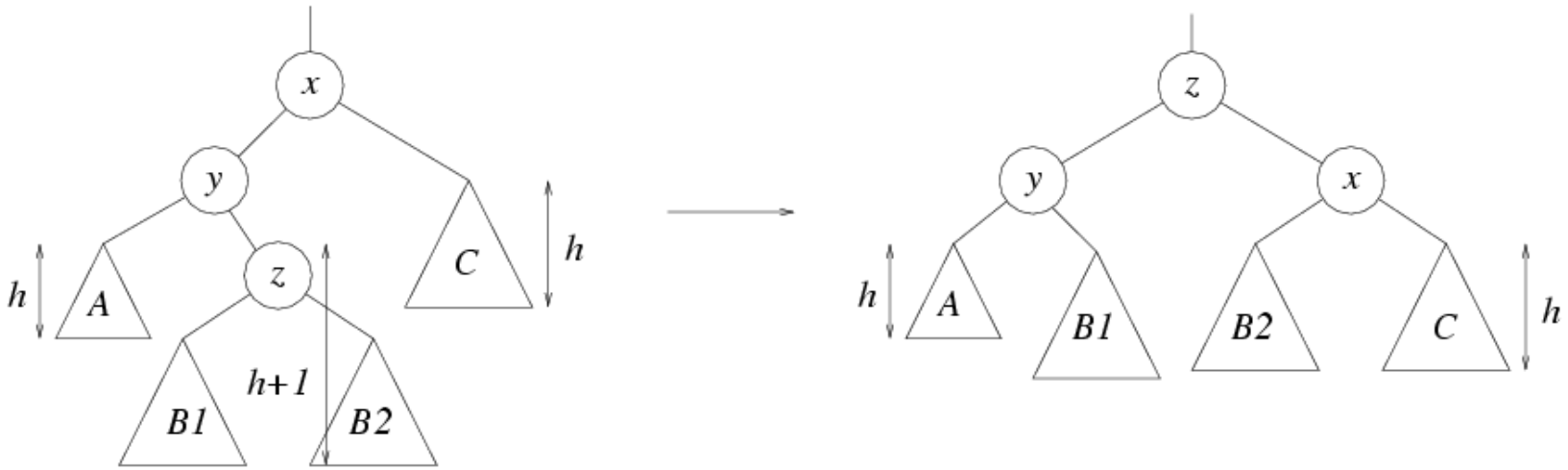
Μερικές φορές το πρόβλημα δε λύνεται με απλή περιστροφή





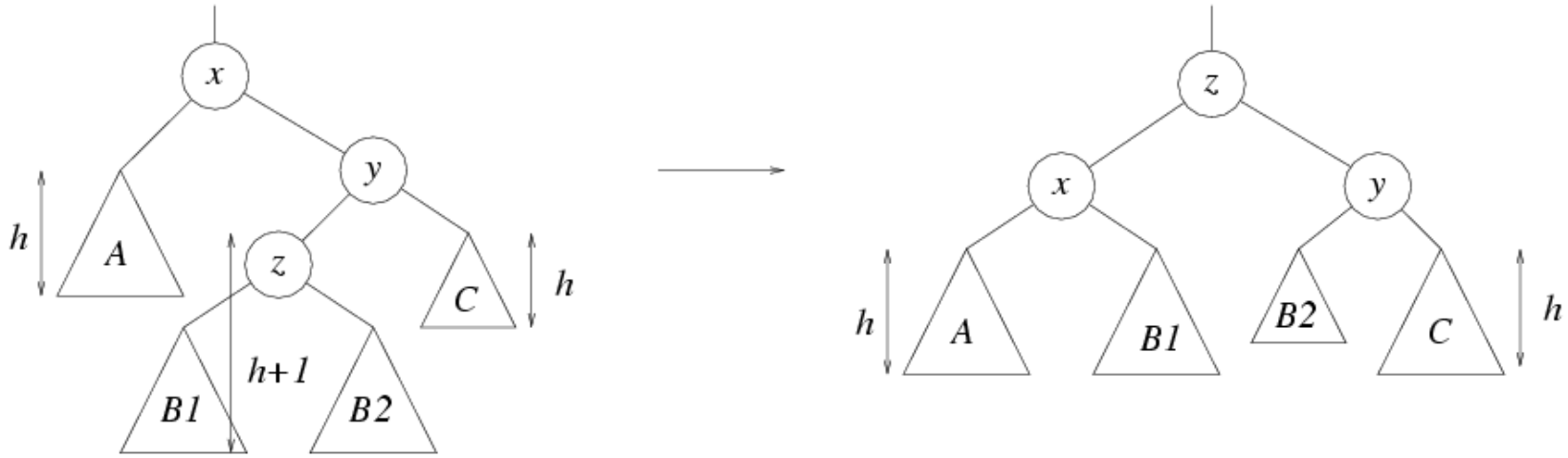
Τι μπορούμε να κάνουμε; Διπλή περιστροφή !

Διπλή Περιστροφή



Περίπτωση 3

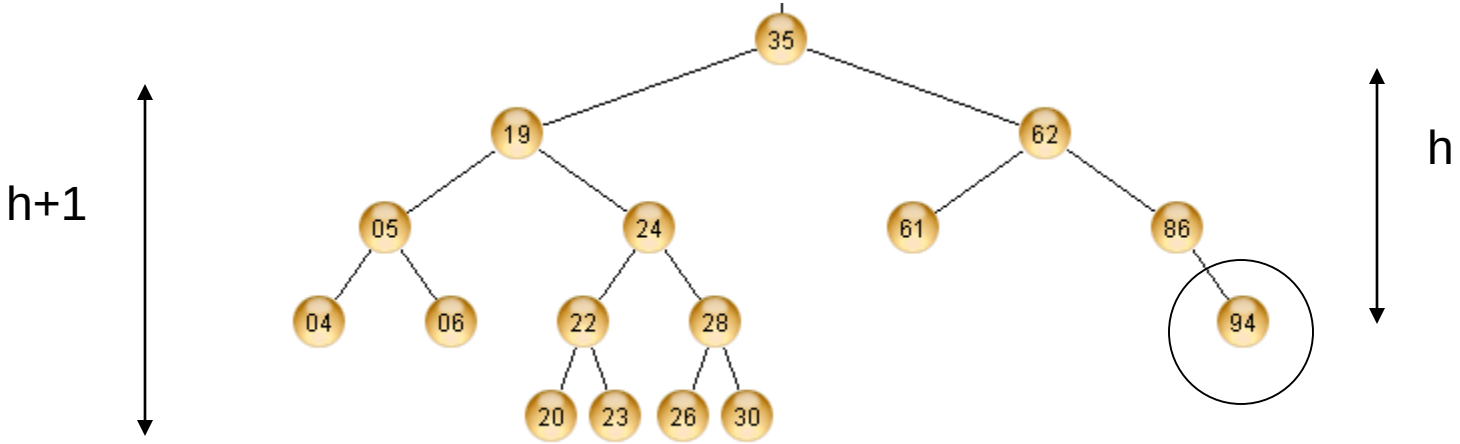
Διπλή Περιστροφή



Περίπτωση 4

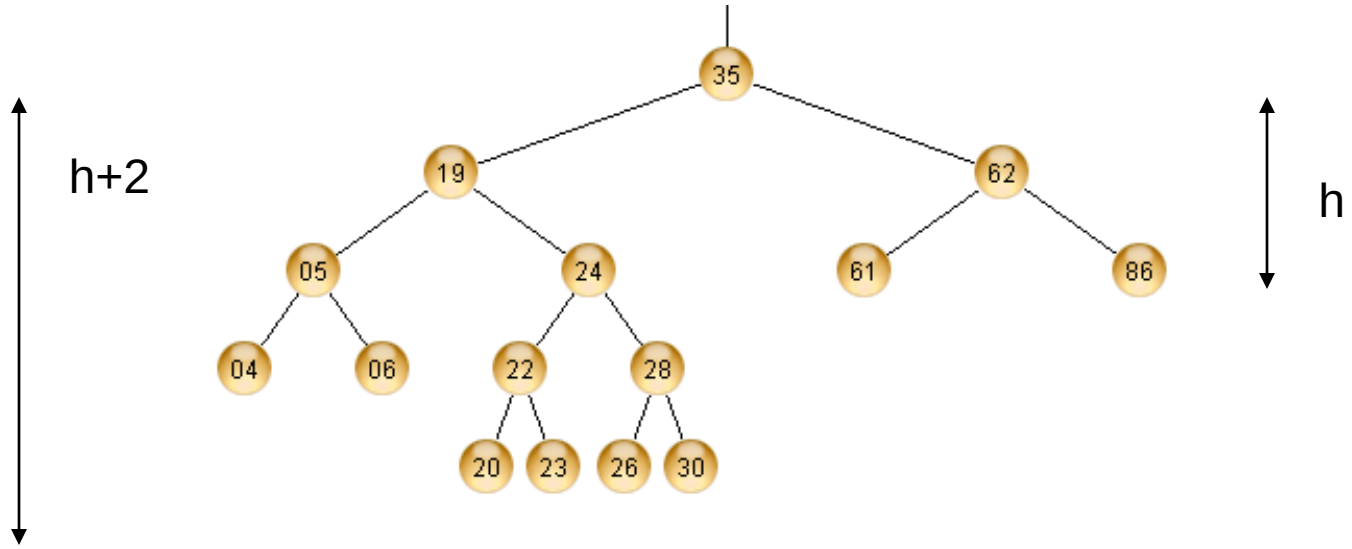
Διπλή Περιστροφή - Παράδειγμα

Αυτό είναι ένα δένδρο AVL



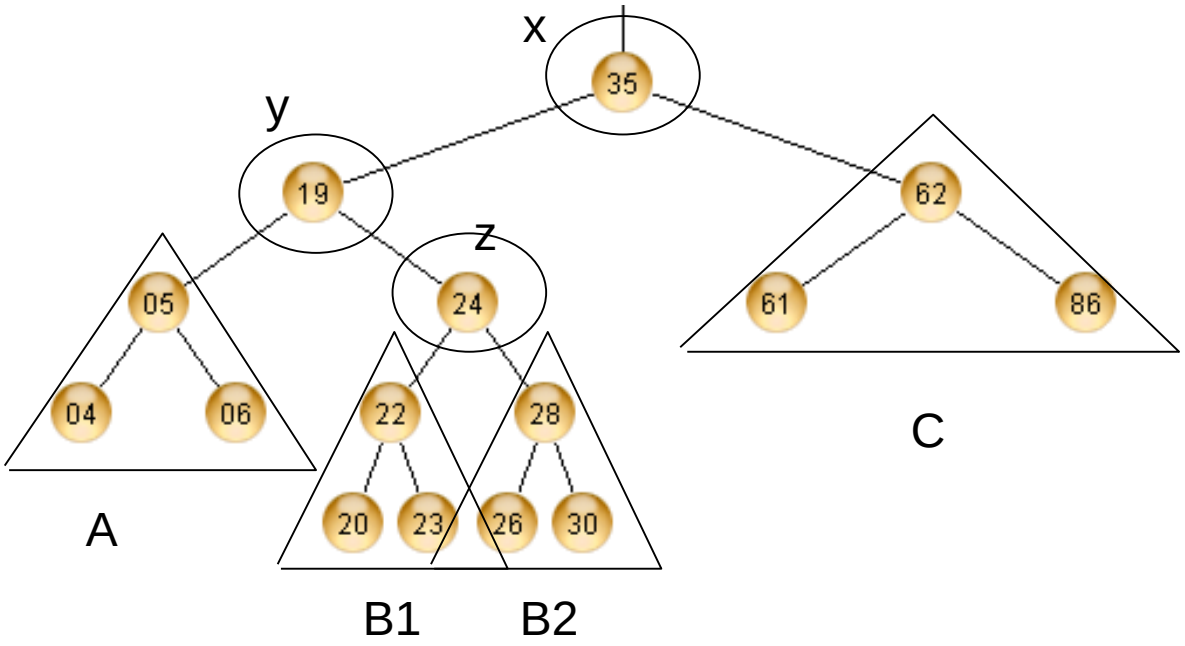
Διαγραφή του κόμβου 94

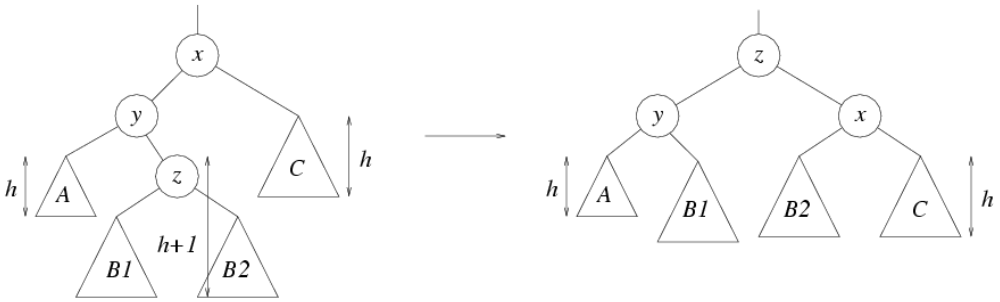
Διπλή Περιστροφή - Παράδειγμα



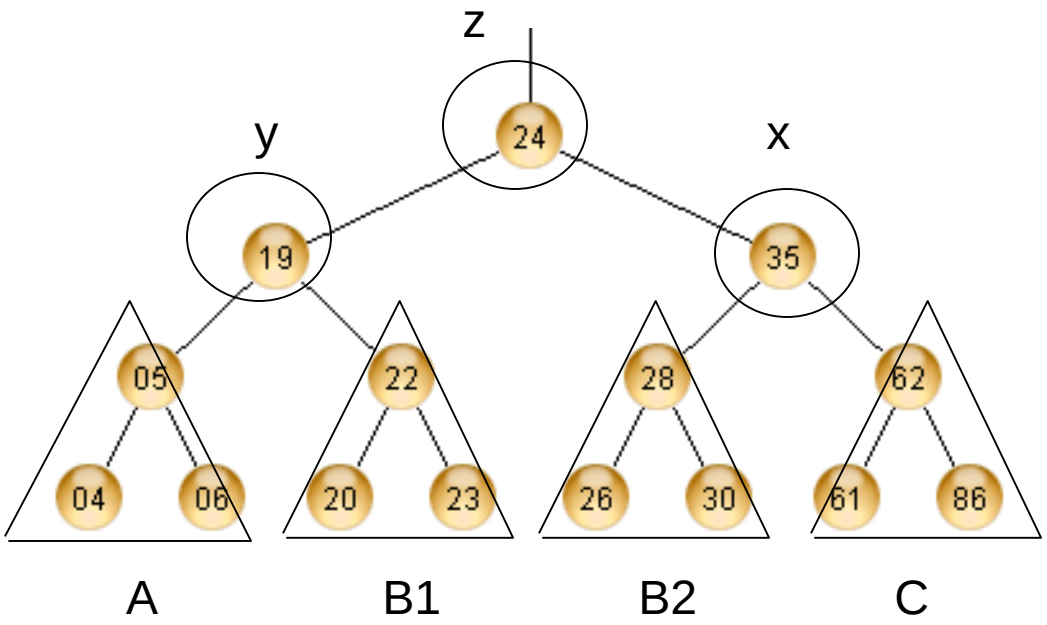
Παραβιάζεται ο ορισμός του AVL

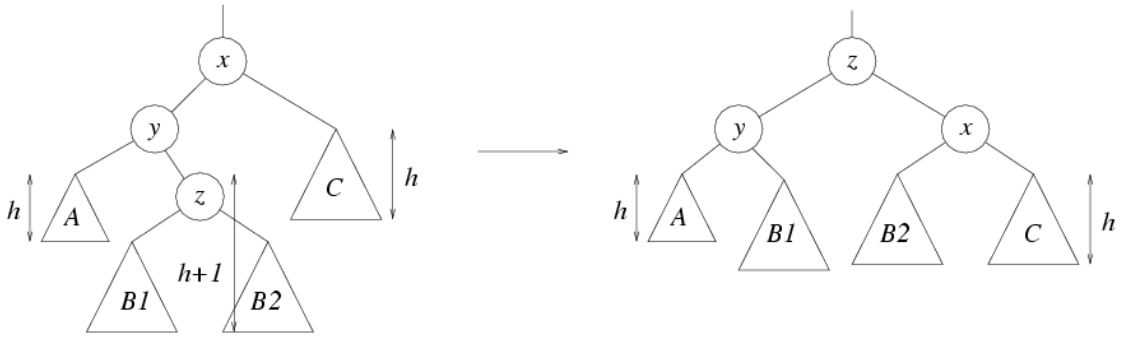
Διπλή Περιστροφή - Παράδειγμα



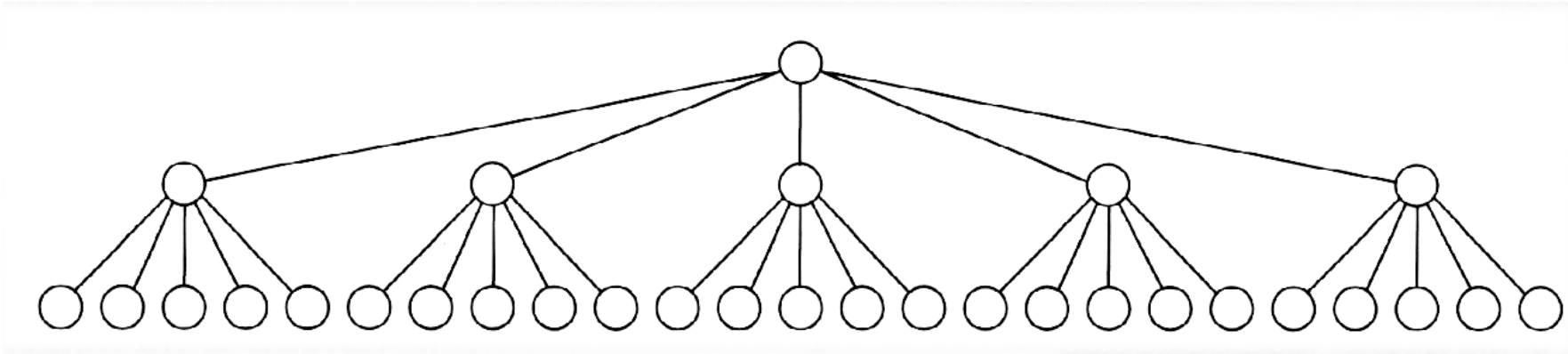


Μετά τη Διπλή Περιστροφή



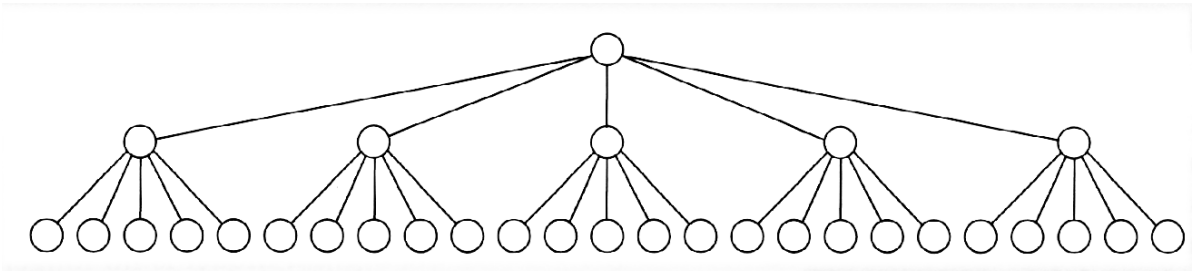


Δένδρα Πολλών Δρόμων

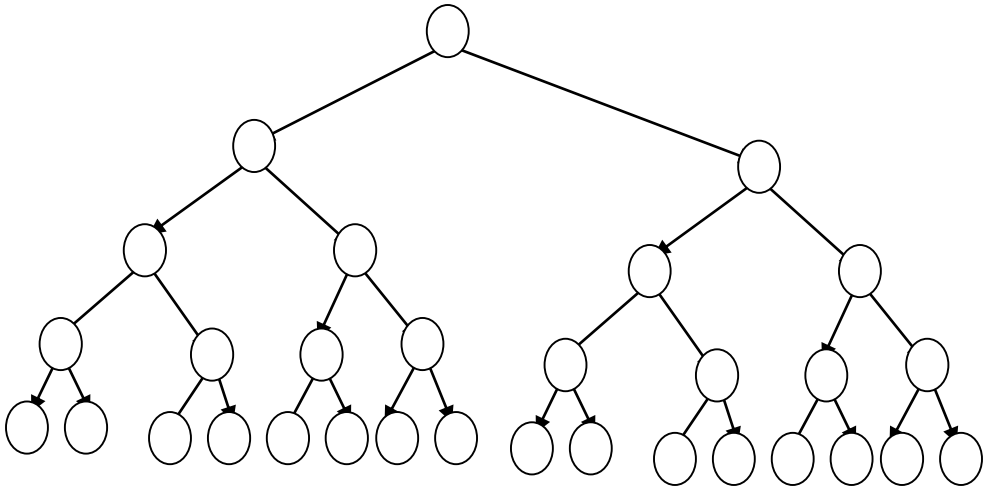


Δένδρο 5 δρόμων

Παράδειγμα



Δένδρο 5 δρόμων
31 κόμβοι
3 επίπεδα



Δυαδικό δένδρο
31 κόμβοι
5 επίπεδα

Παράδειγμα

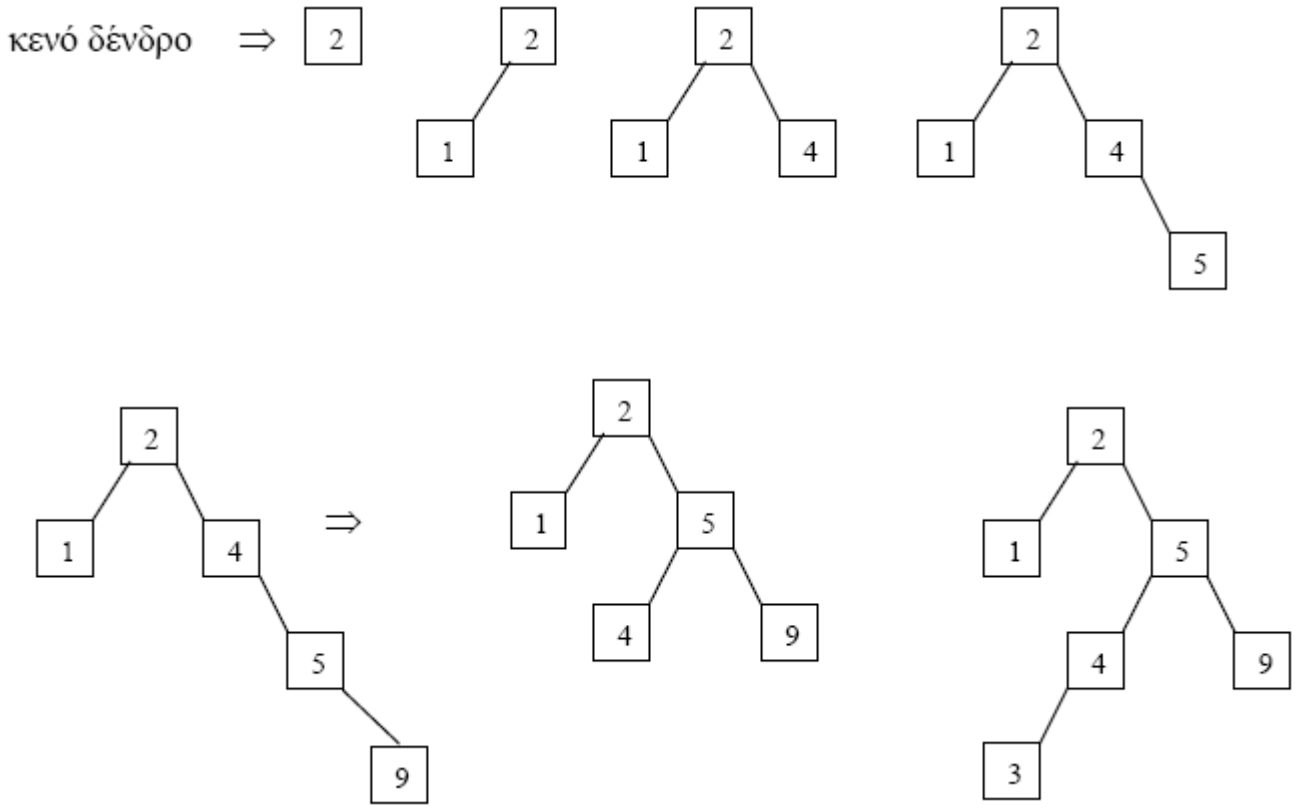
Σε ένα άδειο δένδρο να εισάγετε διαδοχικά τα παρακάτω στοιχεία 2, 1,4, 5, 9, 3, 6, 7, 16, 0.

Να δείχνετε το αποτέλεσμα της κάθε μιας από τις εισαγωγές, στην περίπτωση που το δένδρο είναι:

AVL-δένδρο

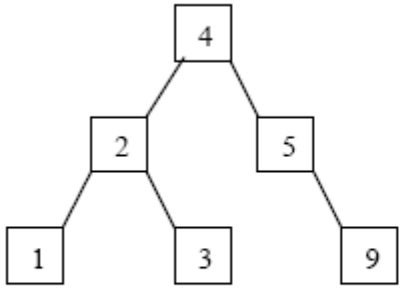
Λύση

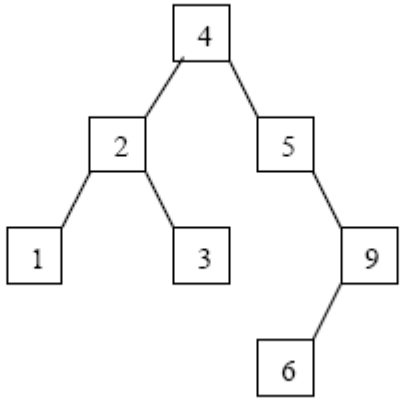
στοιχεία 2, 1,4, 5, 9, 3, 6, 7, 16, 0.

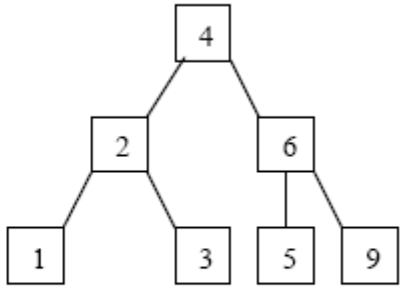


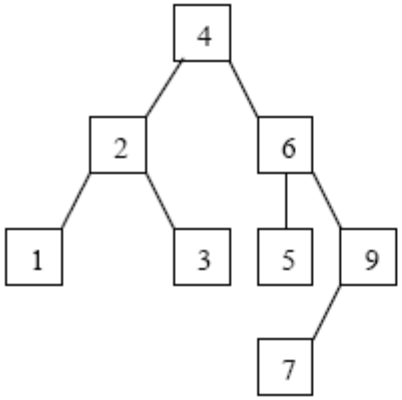
Λύση

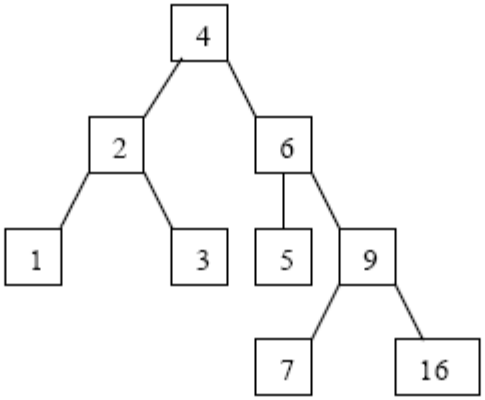
στοιχεία 2, 1,4, 5, 9, 3, 6, 7, 16, 0.

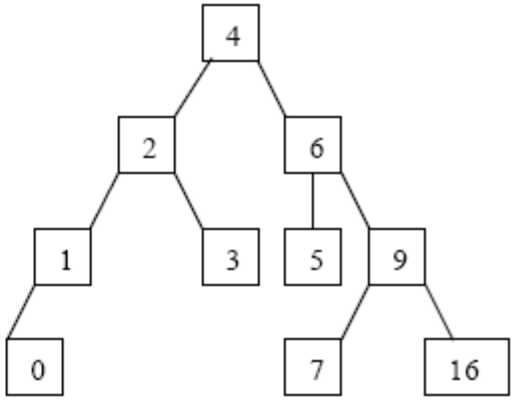








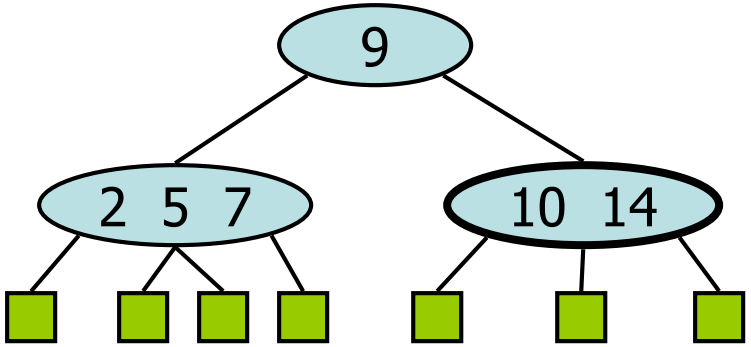




Συνδέσεις

- http://webpages.ull.es/users/jriera/Docencia/AVL/AVL%20tree%20applet.htm
- http://www.site.uottawa.ca/~stan/csi2514/applets/avl/BT.html

(2,4) Δέντρα

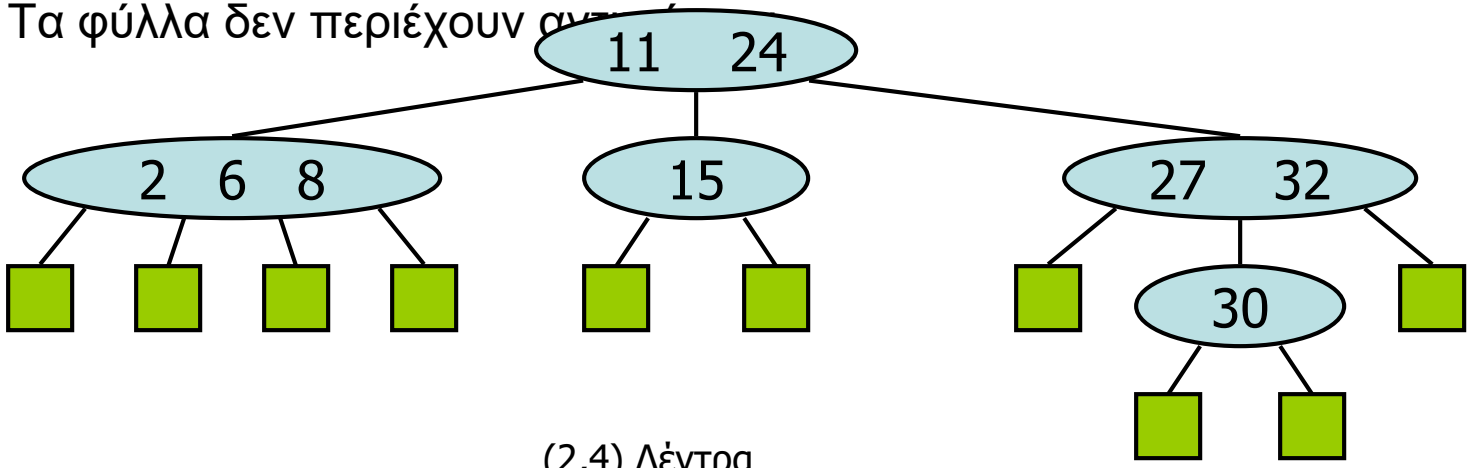


Περίληψη

- Πολυκατευθυνόμενο δέντρο αναζήτησης
  - Ορισμός
  - Αναζήτηση
- (2,4) δέντρο
  - Ορισμός
  - Αναζήτηση
  - Εισαγωγή
  - Διαγραφή

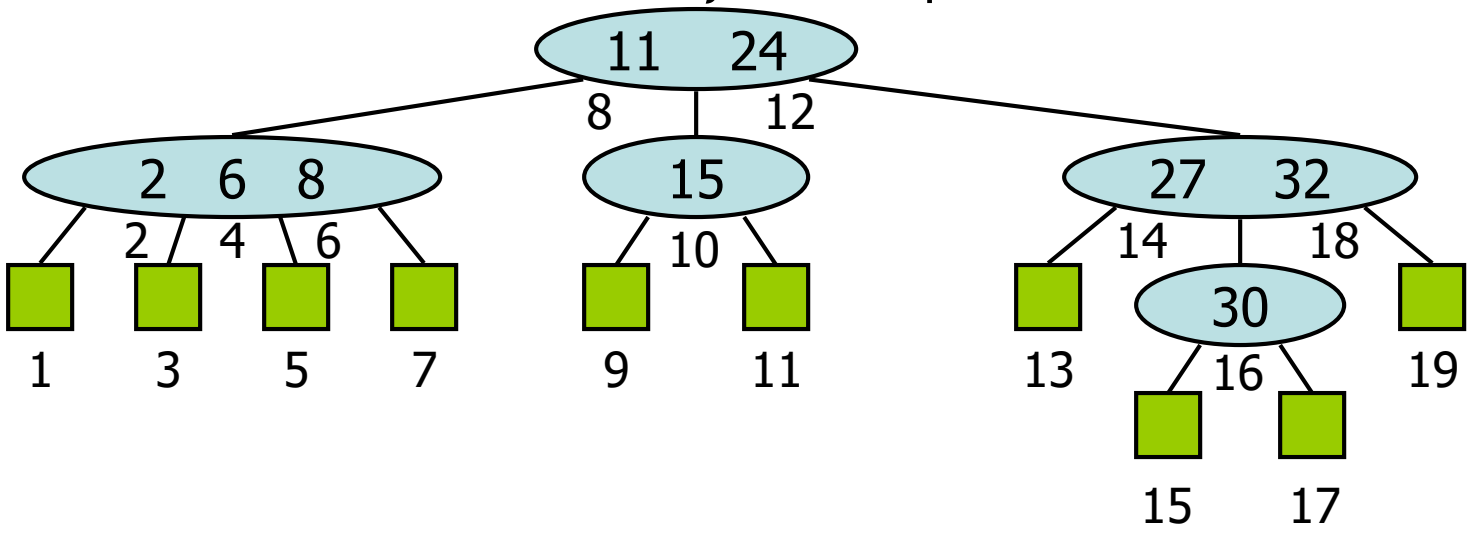
Πολυκατευθυνόμενο δέντρο αναζήτησης

- Ένα πολυκατευθυνόμενο δέντρο αναζήτησης είναι ένα διατεταγμένο δέντρο τέτοιο ώστε
  - Κάθε εσωτερικός κόμβος έχει τουλάχιστον δύο παιδιά και περιέχει $d-1$ κλειδιά-στοιχεία αντικείμενα (k_i, o_i), όπου d είναι ο αριθμός των παιδιών
  - Για κόμβο με παιδιά $v_1 v_2 \ldots v_d$ που περιέχουν κλειδιά $k_1 k_2 \ldots k_{d-1}$
    - τα κλειδιά στο υποδέντρο του v_1 είναι λιγότερα από k_1
    - τα κλειδιά στο υποδέντρο του v_i είναι μεταξύ k_{i-1} και k_i $(i = 2, \ldots, d-1)$
    - τα κλειδιά στο υποδέντρο του v_d είναι μεγαλύτερα από k_{d-1}
  - Τα φύλλα δεν περιέχουν αντ[illegible]



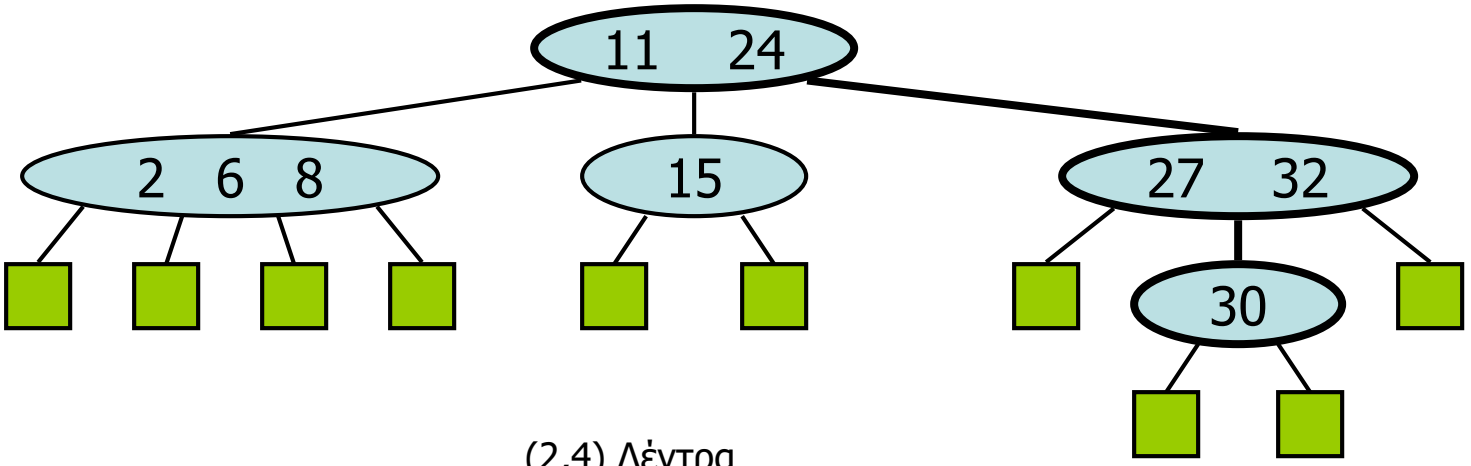
Πολυκατευθυνόμενη Εν Σειρά Διάσχιση

- Μπορούμε να επεκτείνουμε την έννοια της εν σειρά διάσχισης από τα δυαδικά δέντρα στα πολυκατευθυνόμενα
- Επισκεπτόμαστε το αντικείμενο (k_i, o_i) του κόμβου v ανάμεσα στις αναδρομικές διασχίσεις των υποδέντρων του v που είναι ρίζα των παιδιών v_i και v_{i+1}
- Μια εν σειρά διάσχιση πολυκατευθυνόμενου δέντρου επισκέπτεται τα κλειδιά σε αύξουσα σειρά



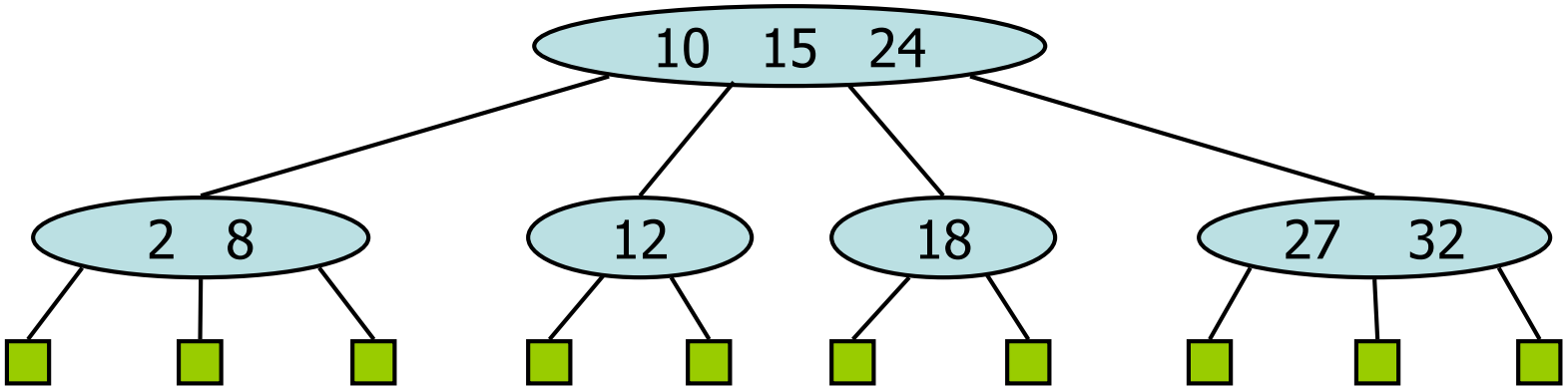
Πολυκατευθυνόμενη Αναζήτηση

- Παρόμοια με την αναζήτηση σε δυαδικό δέντρο αναζήτησης
- Για κάθε εσωτερικό κόμβο με παιδιά $v_1 v_2 \ldots v_d$ και κλειδιά $k_1 k_2 \ldots k_{d-1}$
  - $k = k_i$ ($i = 1, \ldots, d - 1$): η αναζήτηση τερματίζει επιτυχώς
  - $k < k_1$: συνεχίζουμε την αναζήτηση στο παιδί v_1
  - $k_{i-1} < k < k_i$ ($i = 2, \ldots, d - 1$): συνεχίζουμε την αναζήτηση στο παιδί v_i
  - $k > k_{d-1}$: συνεχίζουμε την αναζήτηση στο παιδί v_d
- Η αναζήτηση τερματίζει ανεπιτυχώς όταν φτάσει σε εξωτερικό κόμβο
- Παράδειγμα: αναζήτηση για το 30



(2,4) Δέντρο

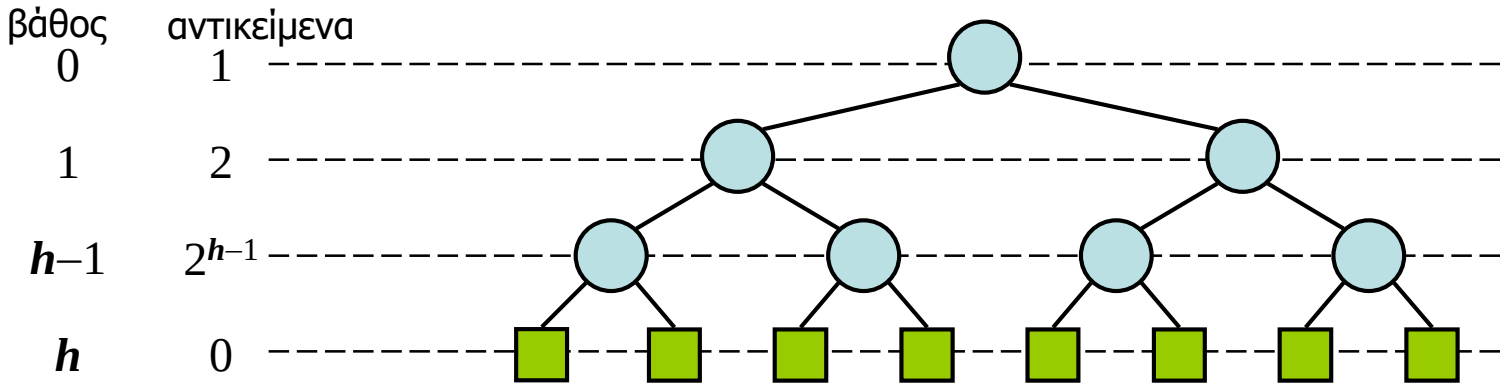
- Ένα (2,4) δέντρο (καλείται επίσης 2-4 δέντρο ή 2-3-4 δέντρο) είναι ένα πολυκατευθυνόμενο δέντρο αναζήτησης με τις παρακάτω ιδιότητες
  - Μέγεθος κόμβου: κάθε εσωτερικός κόμβος έχει το πολύ 4 παιδιά
  - Βάθος: όλοι οι εξωτερικοί κόμβοι έχουν το ίδιο βάθος
- Ανάλογα με το πλήθος των παιδιών, ένας εσωτερικός κόμβος ενός (2,4) δέντρου λέγεται 2-node, 3-node ή 4-node



Ύψος ενός (2,4) Δέντρου

- Θεώρημα: ένα (2,4) δέντρο που περιέχει n αντικείμενα έχει ύψος $O(\log n)$

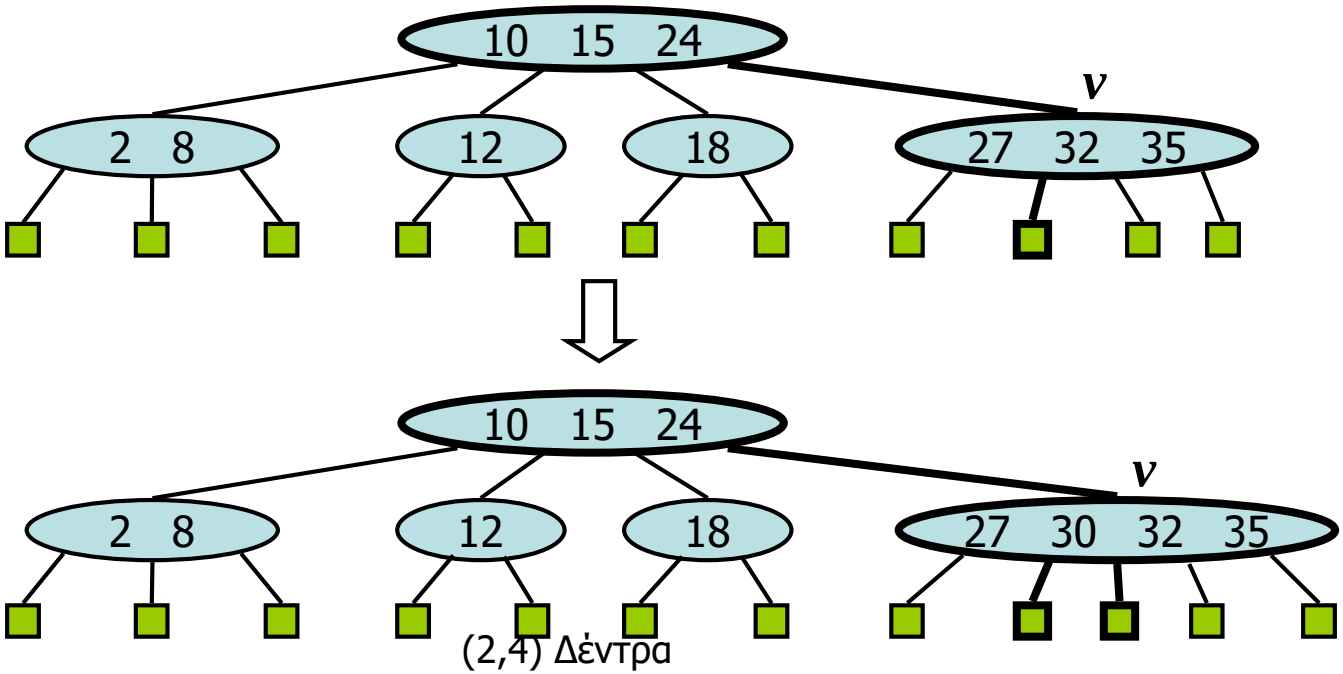
  Απόδειξη:
  - Έστω h το ύψος ενός (2,4) δέντρου με n αντικείμενα
  - Αφού υπάρχουν τουλάχιστον 2^i αντικείμενα σε βάθος $i = 0, \ldots, h-1$ και καθόλου αντικείμενα σε βάθος h, έχουμε
    $$n \geq 1 + 2 + 4 + \ldots + 2^{h-1} = 2^h - 1$$
  - Άρα, $h \leq \log (n+1)$
- Η αναζήτηση σε ένα (2,4) δέντρο με n αντικείμενα παίρνει $O(\log n)$ χρόνο



(2,4) Δέντρα

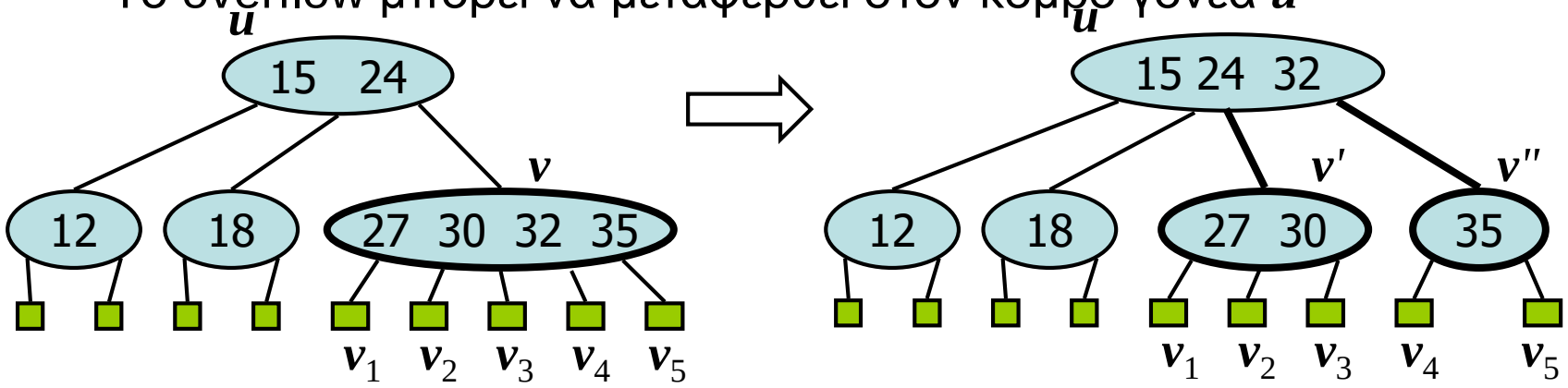
Εισαγωγή

- Εισάγουμε νέο αντικείμενο (k, o) στον γονέα v του φύλλου που φτάσαμε ψάχνοντας για το k
  - Διατηρούμε την ιδιότητα του βάθους αλλά
  - Μπορεί να προκληθεί overflow (π.χ., ο κόμβος v μπορεί να γίνει ένας 5-node)
- Παράδειγμα: η εισαγωγή του κλειδιού 30 προκαλεί overflow



Overflow και Split

- Αντιμετωπίζουμε ένα overflow σε έναν 5-node v με μία split διεργασία:
  - έστω $v_1 \ldots v_5$ τα παιδιά του v και $k_1 \ldots k_4$ τα κλειδιά του v
  - ο κόμβος v αντικαθίσταται από τους κόμβους v' και v''
    - v' είναι ένας 3-node με κλειδιά k_1 k_2 και παιδιά v_1 v_2 v_3
    - v'' είναι ένας 2-node με κλειδί k_4 και παιδιά v_4 v_5
  - το κλειδί k_3 εισάγεται στο γονέα u του v (μια νέα ρίζα μπορεί να δημιουργηθεί)
- Το overflow μπορεί να μεταφερθεί στον κόμβο γονέα u



Ανάλυση της Εισαγωγής

Algorithm *insertItem*(*k*, *o*)

1. Ψάχνουμε για το κλειδί ***k*** για να εντοπίσουμε τον κόμβο εισαγωγής ***v***

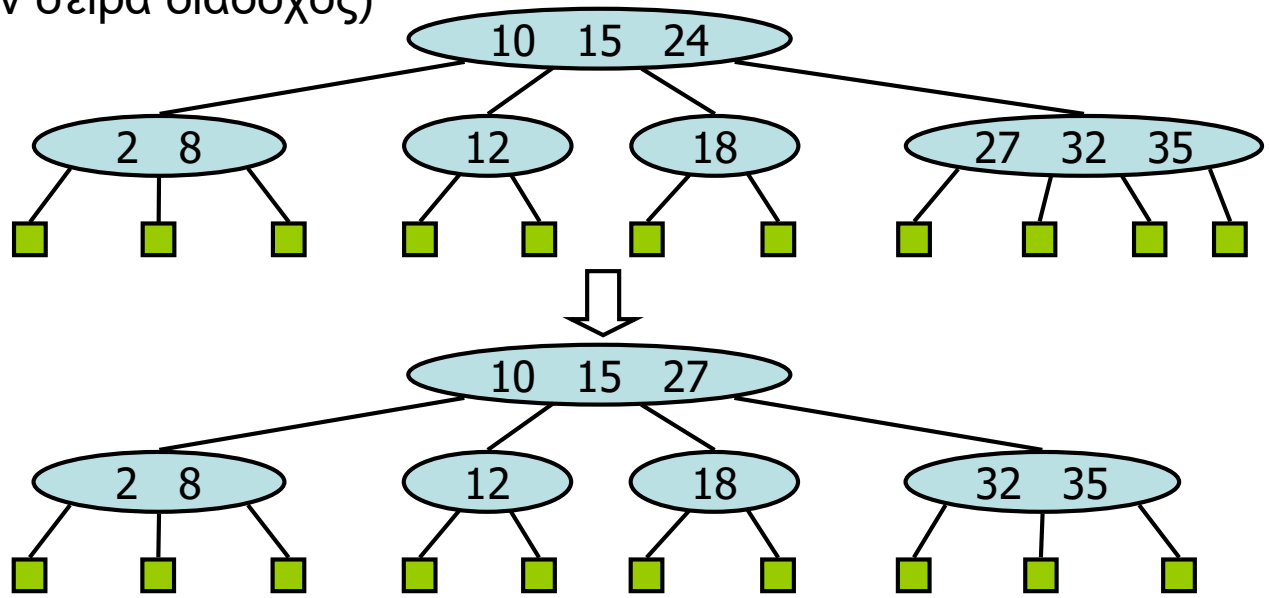
2. Προσθέτουμε το νέο στοιχείο (***k***, ***o***) στον κόμβο ***v***

3. **while *overflow*(*v*)**
   if *isRoot*(*v*)
   δημιούργησε νέα άδεια ρίζα πάνω από το ***v***
   v ← ***split*(*v*)**

- Έστω ***T*** ένα (2,4) δέντρο με ***n*** στοιχεία
  - Το δέντρο ***T*** έχει $O(\log n)$ ύψος
  - Το βήμα 1 παίρνει $O(\log n)$ χρόνο γιατί επισκεπτόμαστε $O(\log n)$ κόμβους
  - Το βήμα 2 παίρνει $O(1)$ χρόνο
  - Το βήμα 3 παίρνει $O(\log n)$ χρόνο γιατί κάθε split παίρνει $O(1)$ χρόνο και πραγματοποιούμε $O(\log n)$ splits
- Άρα, μια εισαγωγή σε ένα (2,4) δέντρο παίρνει $O(\log n)$ χρόνο

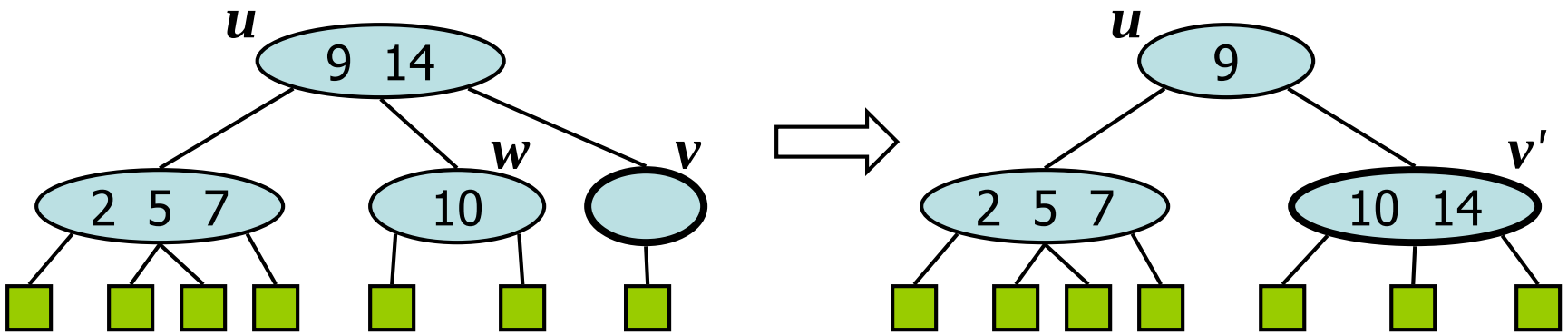
Διαγραφή

- Επικεντρώνουμε τη διαδικασία της διαγραφής στην περίπτωση που ένα στοιχείο είναι σε κόμβο με παιδιά φύλλα
- Αλλιώς, αντικαθιστούμε το στοιχείο με το εν σειρά του διάδοχο (ή αντίστοιχα με το εν σειρά προκάτοχό του) και διαγράφουμε το δεύτερο στοιχείο
- Παράδειγμα: για τη διαγραφή του κλειδιού 24, το αντικαθιστούμε με το 27 (εν σειρά διάδοχος)



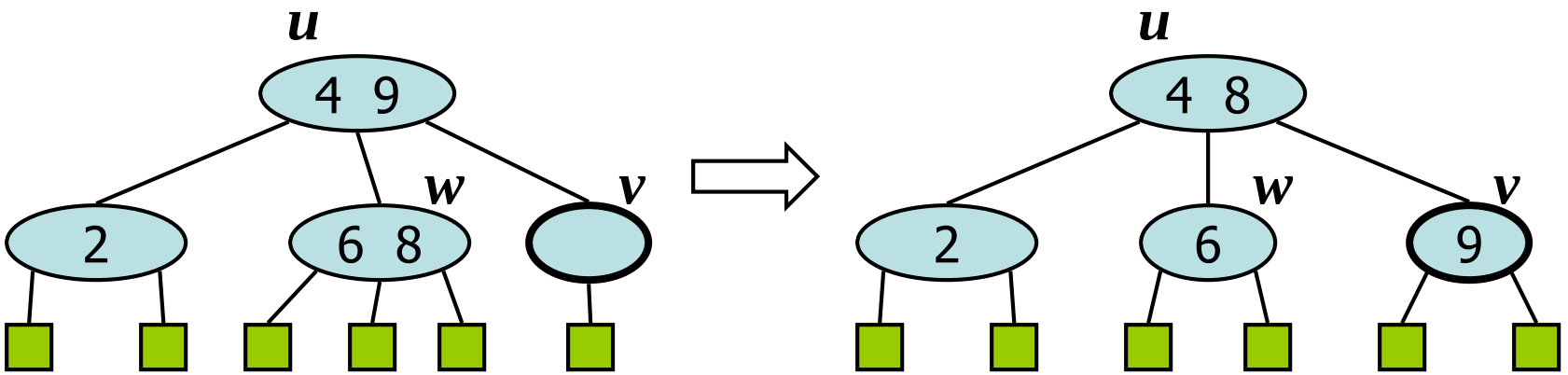
Underflow και Fusion

- Η διαγραφή ενός στοιχείου από κόμβο v μπορεί να προκαλέσει underflow, όπου ο κόμβος v γίνεται ένας 1-node με ένα παιδί και χωρίς κλειδιά
- Για την αντιμετώπιση ενός underflow στον κόμβο v με γονέα u, θεωρούμε δύο πρειπτώσεις
- Περίπτωση 1: τα γειτονικά αδέρφια του v είναι 2-nodes
  - Διεργασία Fusion: ενώνουμε το v με ένα γειτονικό αδερφό w και μετακινούμε ένα στοιχείο από το u στον ενωμένο κόμβο v'
  - Μετά την fusion, το underflow μπορεί να μεταφερθεί στον γονέα u



Underflow και Transfer

- Για την αντιμετώπιση ενός underflow σε κόμβο v με γονέα u, θεωρούμε δύο περιπτώσεις
- Περίπτωση 1: ένας γειτονικός αδερφός w του v είναι ένας 3-node ή 4-node
  - Διεργασία Transfer:
    1. μετακινούμε ένα παιδί του w στο v
- Περίπτωση 2: ένας γειτονικός αδερφός w του v είναι ένας 2-node ή 1-node
  - Διεργασία Transfer:
    1. μετακινούμε ένα στοιχείο του u στο v
    2. μετακινούμε ένα στοιχείο του w στο u
  - Μετά την μεταφορά, δεν παρουσιάζεται underflow



Ανάλυση της Διαγραφής

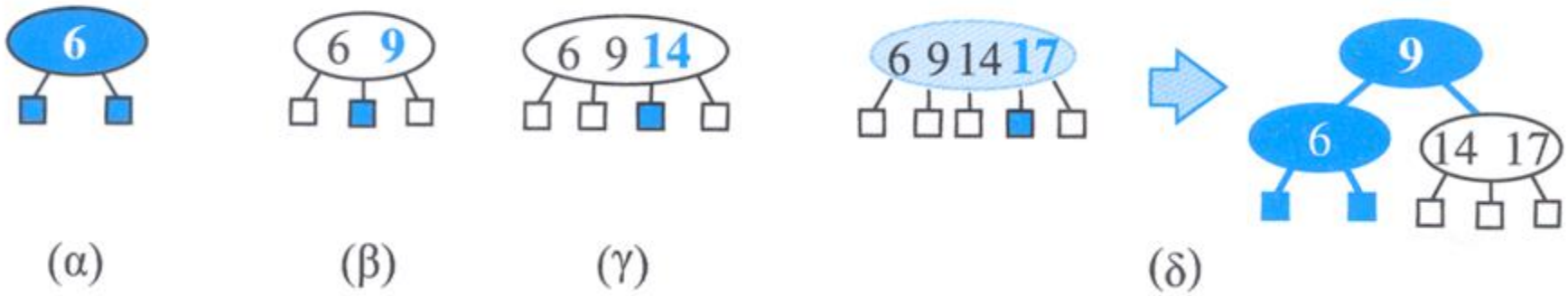
- Έστω T ένα (2,4) δέντρο με n στοιχεία
  - Το δέντρο T έχει $O(\log n)$ ύψος
- Κατά την διαγραφή
  - Επισκεπτόμαστε $O(\log n)$ κόμβους για να εντοπίσουμε τον κόμβο απ'όπου θα διαγραφεί το στοιχείο
  - Αντιμετωπίσουμε ένα underflow με μια σειρά από $O(\log n)$ fusions, ακολουθούμενες το πολύ από ένα transfer
  - Κάθε fusion και transfer παίρνει $O(1)$ χρόνο
- Άρα, η διαγραφή ενός στοιχείου από ένα (2,4) δέντρο παίρνει $O(\log n)$ χρόνο

Παράδειγμα

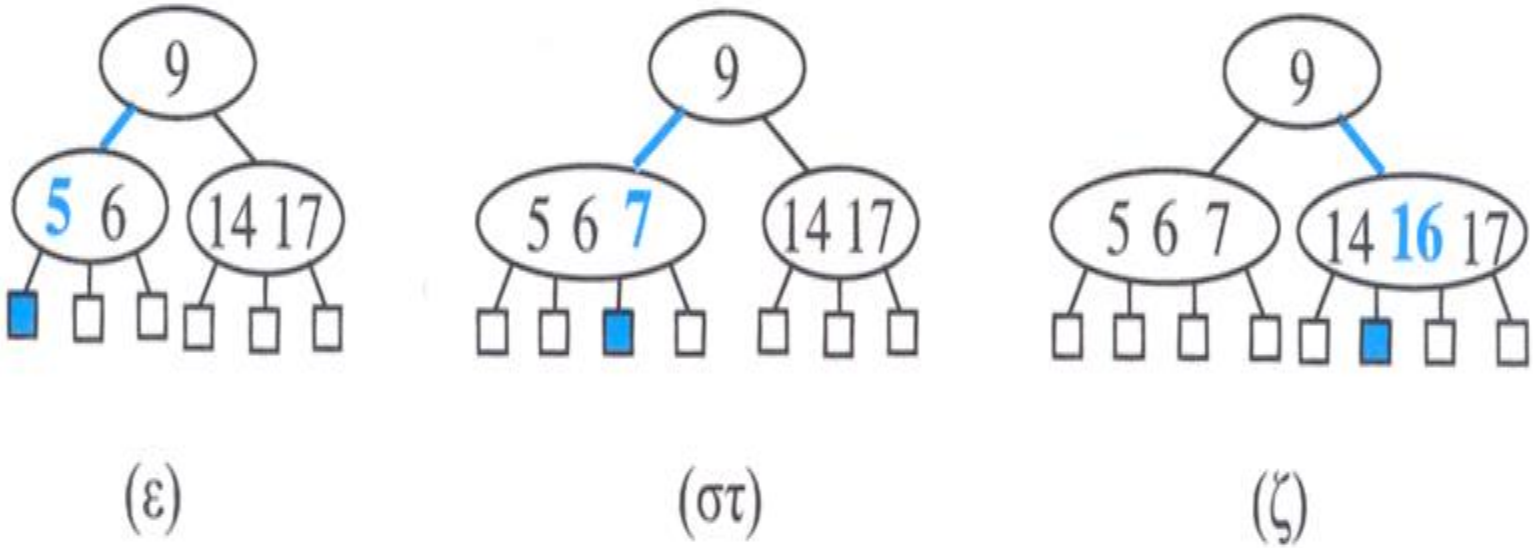
Εισάγονται τα παρακάτω στοιχεία σε ένα δέντρο (2,4) με την ακόλουθη σειρά: 6,9, 14,17, 5, 7, 16, 20, 18, 19, 4, 11.

Στην συνέχεια διαγράφονται τα στοιχεία: 6, 9, 14, 4, 5, 20, 17.

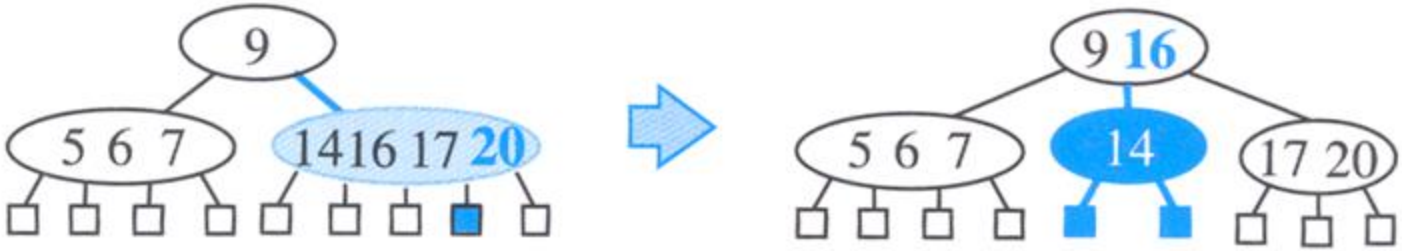
Να δείξετε το αποτέλεσμα μετά από κάθε εισαγωγή / διαγραφή



Εισαγωγή 6, 9, 14, 17

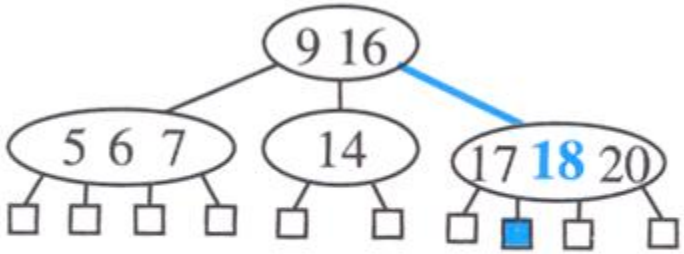


Εισαγωγή 5, 7, 16

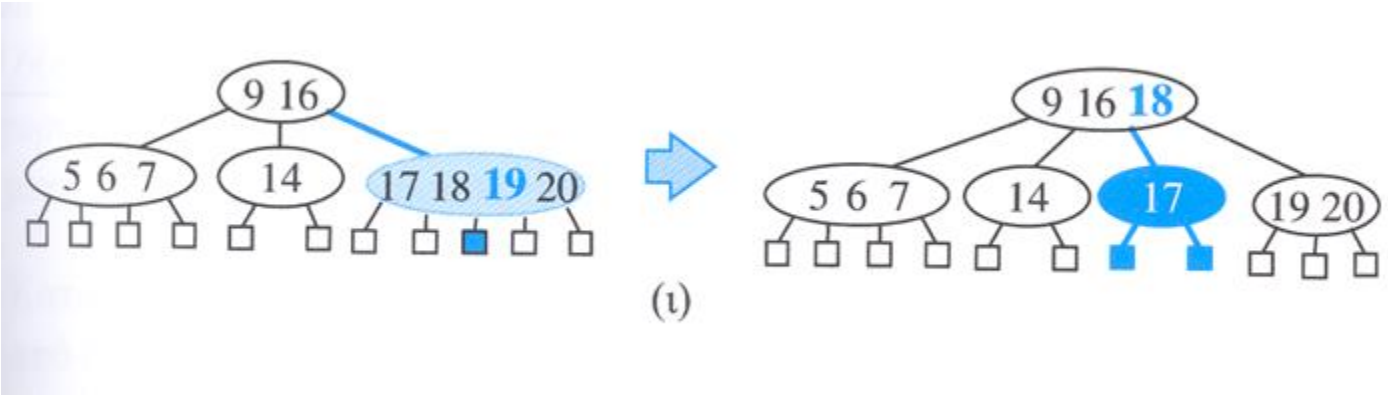


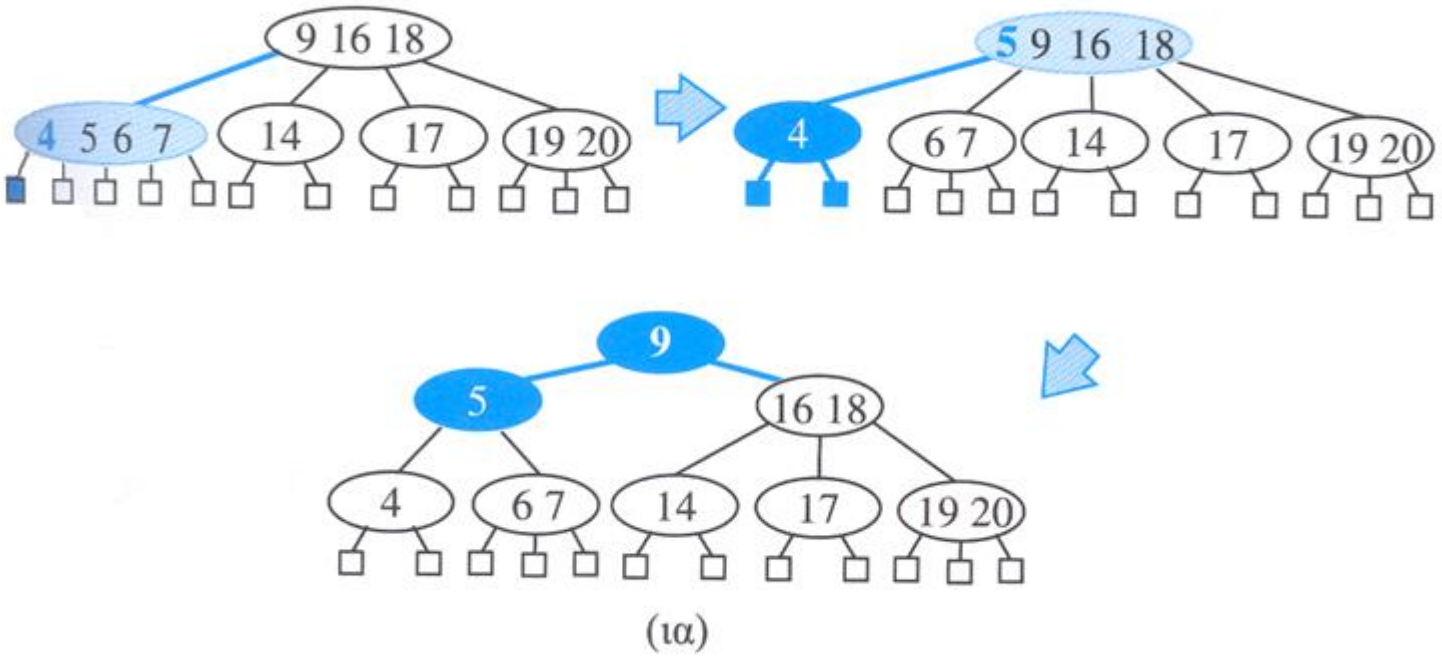
(η)

Εισαγωγή 20



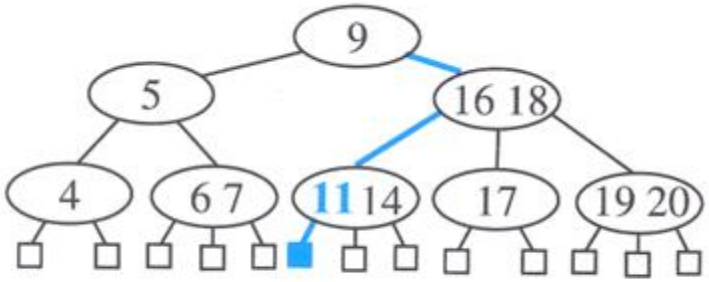
(θ)


9 16
5 6 7
14
17 18 19 20
(ι)
9 16 18
5 6 7
14
17
19 20

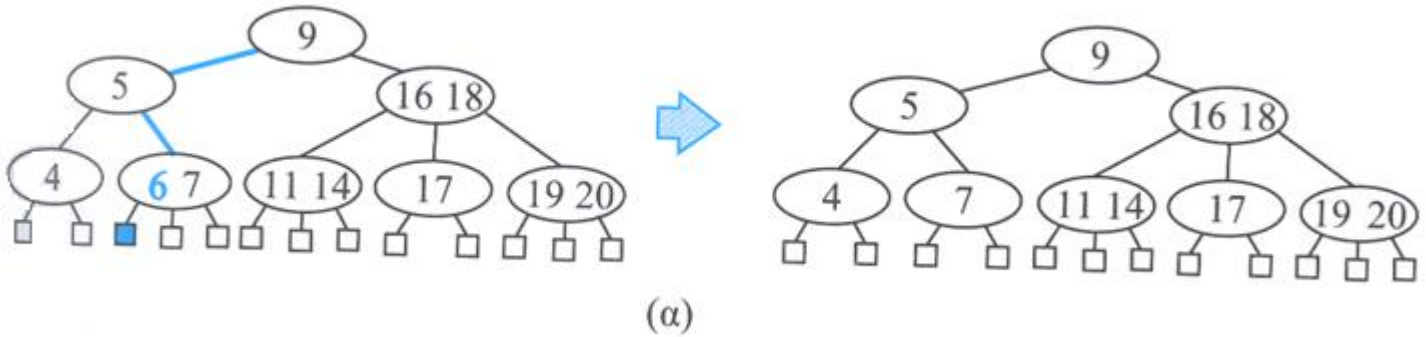


(ια)

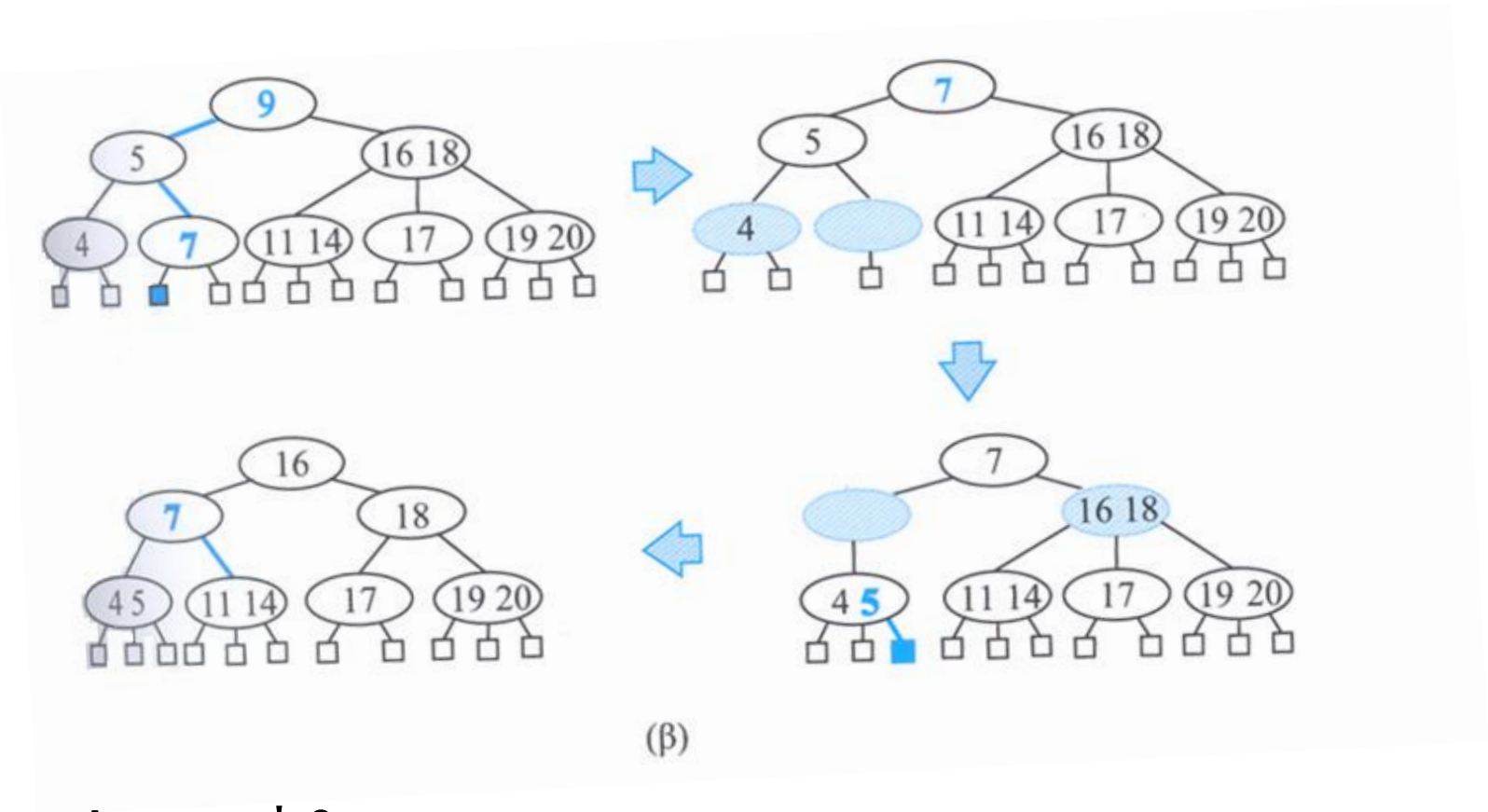
Εισαγωγή 4



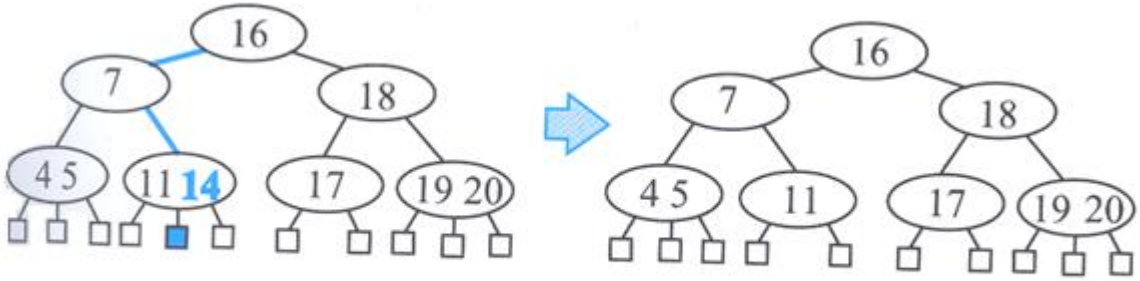
(ιβ)



Διαγραφή 6

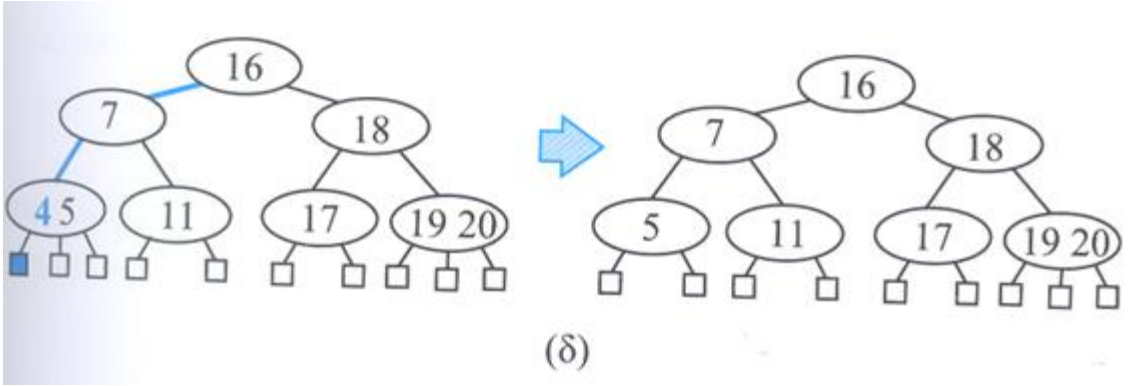

9
5
16 18
4
7
11 14
17
19 20
7
5
16 18
4
11 14
17
19 20
16
7
18
4 5
11 14
17
19 20
7
16 18
4 5
11 14
17
19 20
(β)

Διαγραφή 9

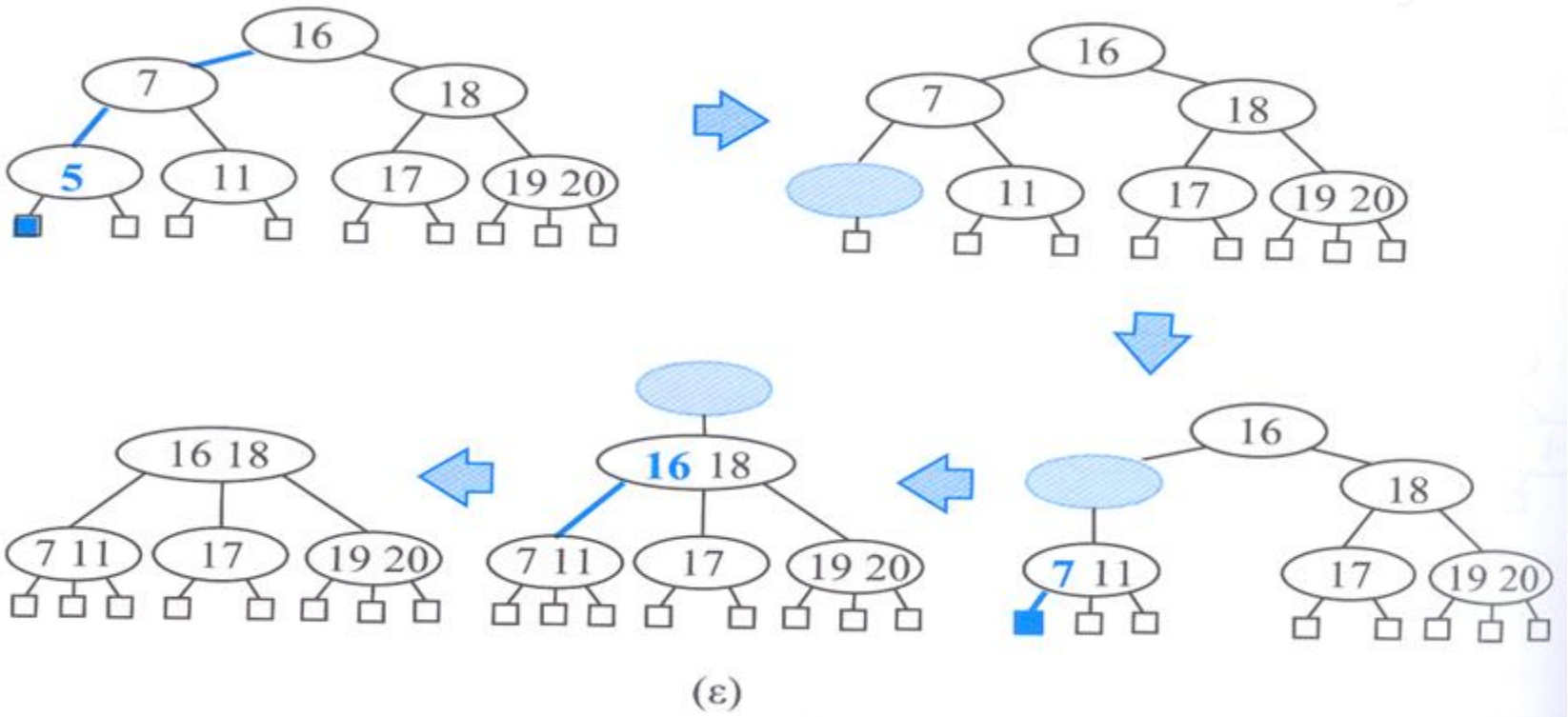


(γ)

Διαγραφή 14



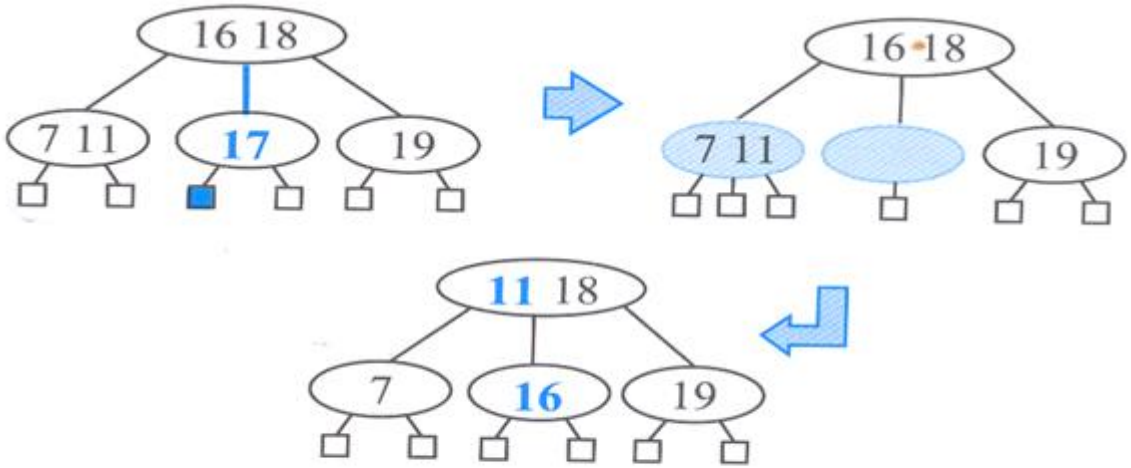
Διαγραφή 4


16
7
18
5
11
17
19 20
16
7
18
11
17
19 20
16
18
7 11
17
19 20
16 18
7 11
17
19 20
16 18
7 11
17
19 20
(ε)

Διαγραφή 5

(στ)

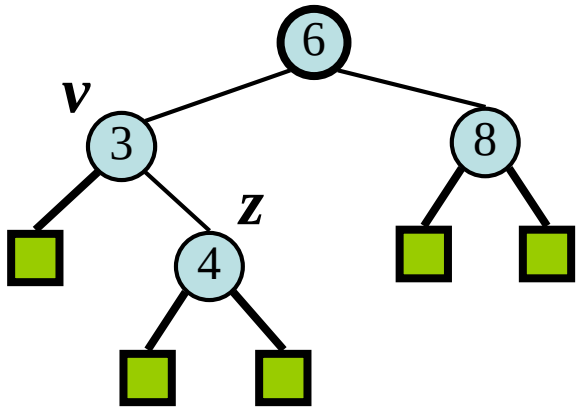
Διαγραφή 20



(ζ)

Διαγραφή 17

Red-Black Δέντρα



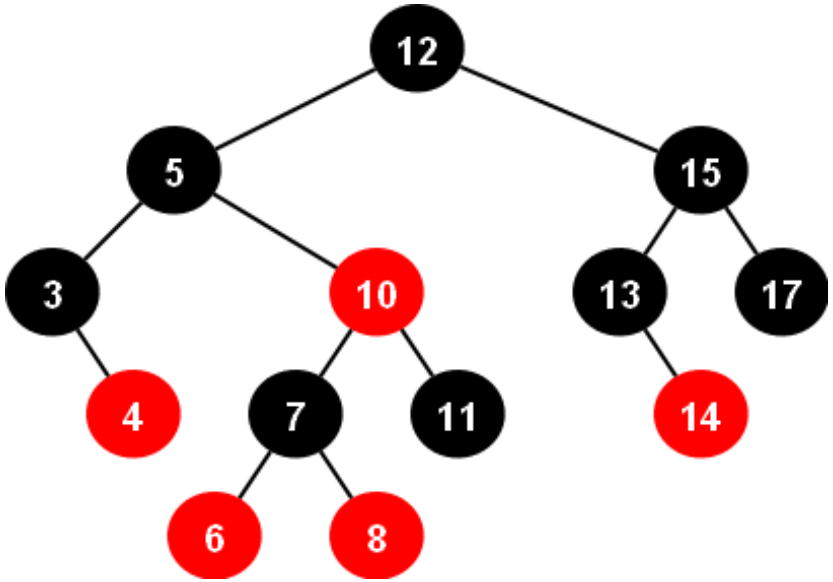
Περίληψη

- Από τα (2,4) δέντρα στα red-black δέντρα
- Red-black δέντρο
  - Ορισμός
  - ΄Υψος
  - Εισαγωγή
    - αναδόμηση
    - επαναχρωματισμός
  - Διαγραφή
    - αναδόμηση
    - επαναχρωματισμός
    - προσαρμογή

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/

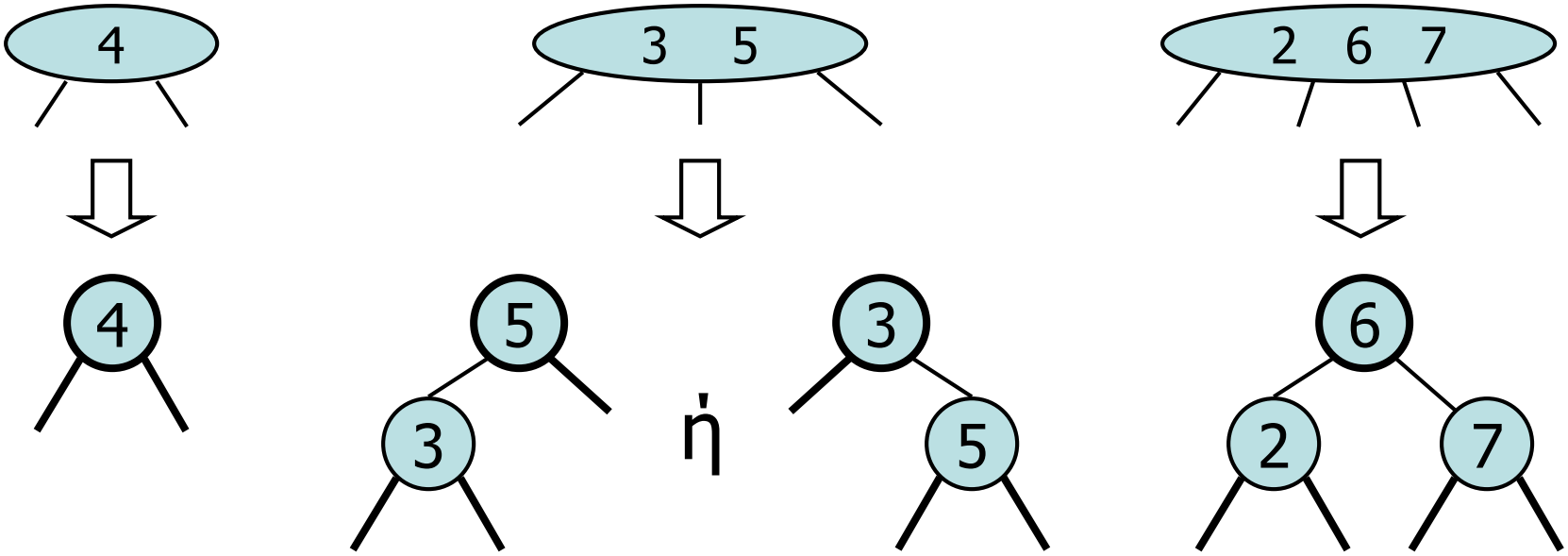
Ύψος κόκκινων μαύρων δένδρων

- 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
```

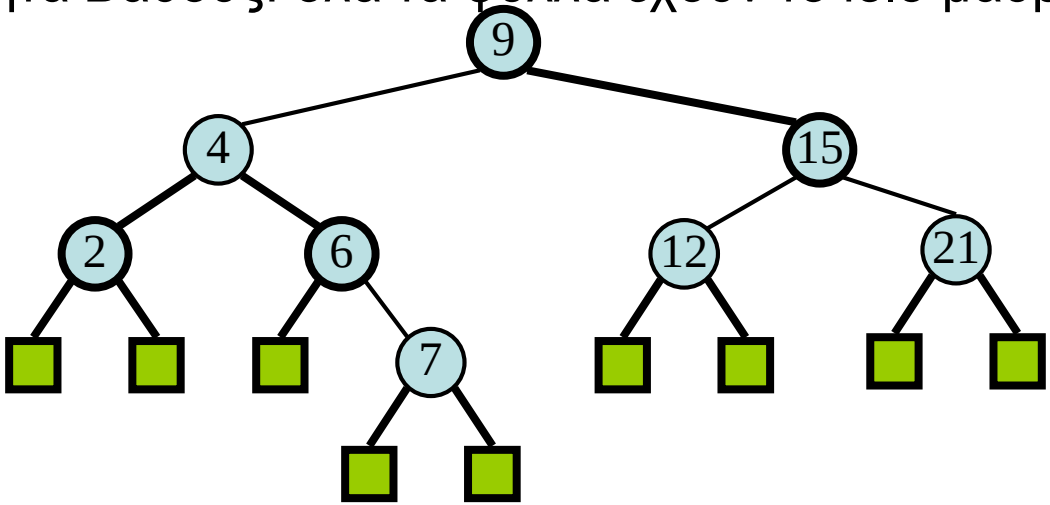
Από τα (2,4) στα Red-Black Δέντρα

- Ένα red-black δέντρο είναι η απεικόνιση ενός (2,4) δέντρο με τρόπο δυαδικού δέντρου του οποίου οι κόμβοι είναι χρωματισμένοι **κόκκινοι** ή **μαύροι**
- Σε σύγκριση με το (2,4) δέντρο, ένα red-black δέντρο έχει
  - ίδια λογαριθμική απόδοση χρόνου
  - απλούστερη υλοποίηση με ένα τύπο κόμβων



Red-Black Δέντρο

- Ένα red-black δέντρο μπορεί να οριστεί σαν ένα δυαδικό δέντρο που ικανοποιεί τις παρακάτω ιδιότητες:
  - Ιδιότητα Ρίζας: η ρίζα είναι μαύρη
  - Εξωτερική Ιδιότητα: κάθε φύλλο είναι μαύρο
  - Εσωτερική Ιδιότητα: τα παιδιά ενός κόκκινου κόμβου είναι μαύρα
  - Ιδιότητα Βάθους: όλα τα φύλλα έχουν το ίδιο μαύρο βάθος



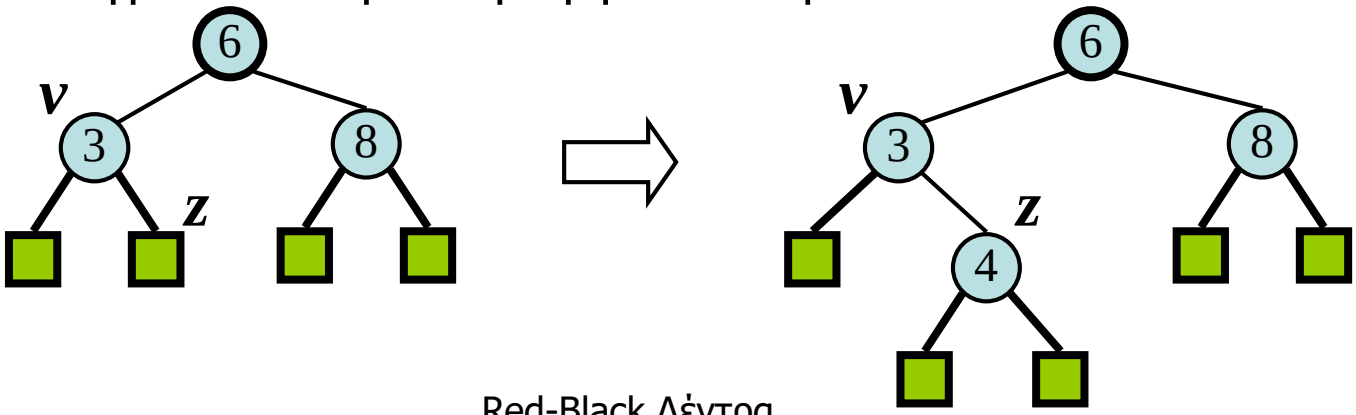
Ύψος Red-Black Δέντρου

- Θεώρημα: Ένα red-black δέντρο που περιέχει n στοιχεία έχει ύψος $O(\log n)$

  Απόδειξη:
  - Το ύψος ενός red-black δέντρου είναι το πολύ δύο φορές το ύψος του αντίστοιχου (2,4) δέντρου, που είναι $O(\log n)$
- Ο αλγόριθμος αναζήτησης για ένα red-black δέντρο είναι ίδιος με αυτόν για ένα δυαδικό δέντρο αναζήτησης
- Σύμφωνα με το παραπάνω θεώρημα, η αναζήτηση σε ένα red-black δέντρο παίρνει $O(\log n)$ χρόνο

Εισαγωγή

- Για την εκτέλεση της διεργασίας insertItem(k, o), εκτελούμε τον αλγόριθμο εισαγωγής για δυαδικά δέντρα αναζήτησης και χρωματίζουμε κόκκινο το νεοεισαγμένο κόμβο z εκτός και αν είναι η ρίζα
  - Διατηρούμε τις ιδιότητες ρίζας, βάθους και την εξωτερική
  - Αν ο γονέας v του z είναι μαύρος, διατηρούμε και την εσωτερική ιδιότητα και η διαδικασία έχει ολοκληρωθεί
  - Αλλιώς (v είναι κόκκινο) έχουμε ένα διπλό κόκκινο (π.χ., παραβίαση της εσωτερικής ιδιότητας), που απαιτεί αναδόμηση του δέντρου
- Παράδειγμα όπου η εισαγωγή του 4 προκαλεί διπλό κόκκινο:

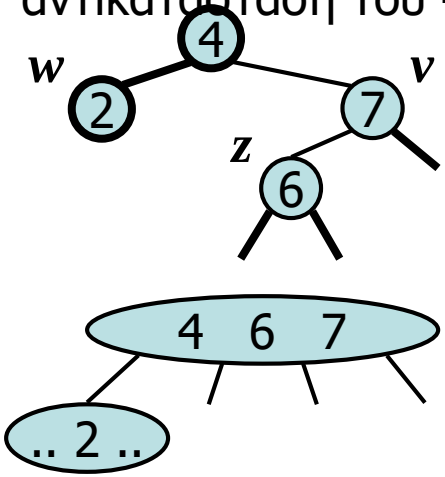


Αποκατάσταση Διπλού Κόκκινου

- Έστω ένα διπλό κόκκινο με παιδί z και γονέα v, και w γείτονας του v

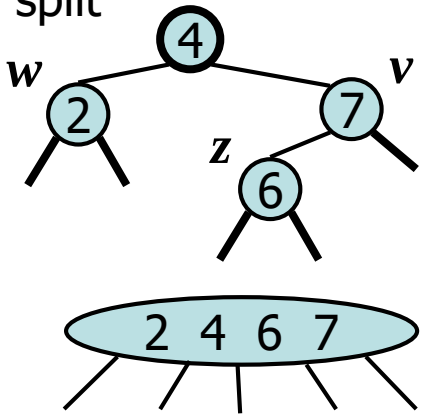
Περίπτωση 1: το w είναι μαύρο

- Το διπλό κόκκινο είναι μια λανθασμένη αντικατάσταση ενός 4-node
- Αναδόμηση: αλλάζουμε την αντικατάσταση του 4-node



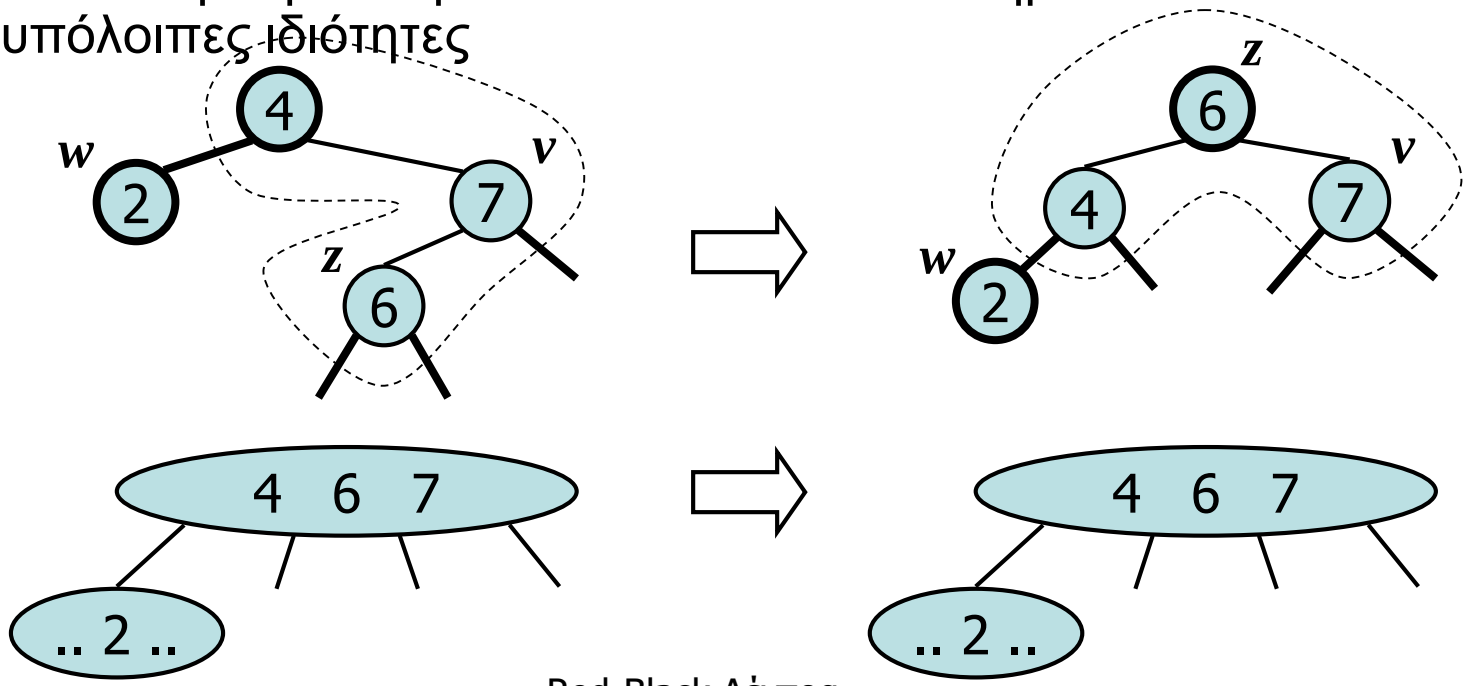
Περίπτωση 2: το w είναι κόκκινο

- Το διπλό κόκκινο αντιστοιχεί σε overflow
- Επαναχρωματισμός: εκτελούμε το αντίστοιχο του split



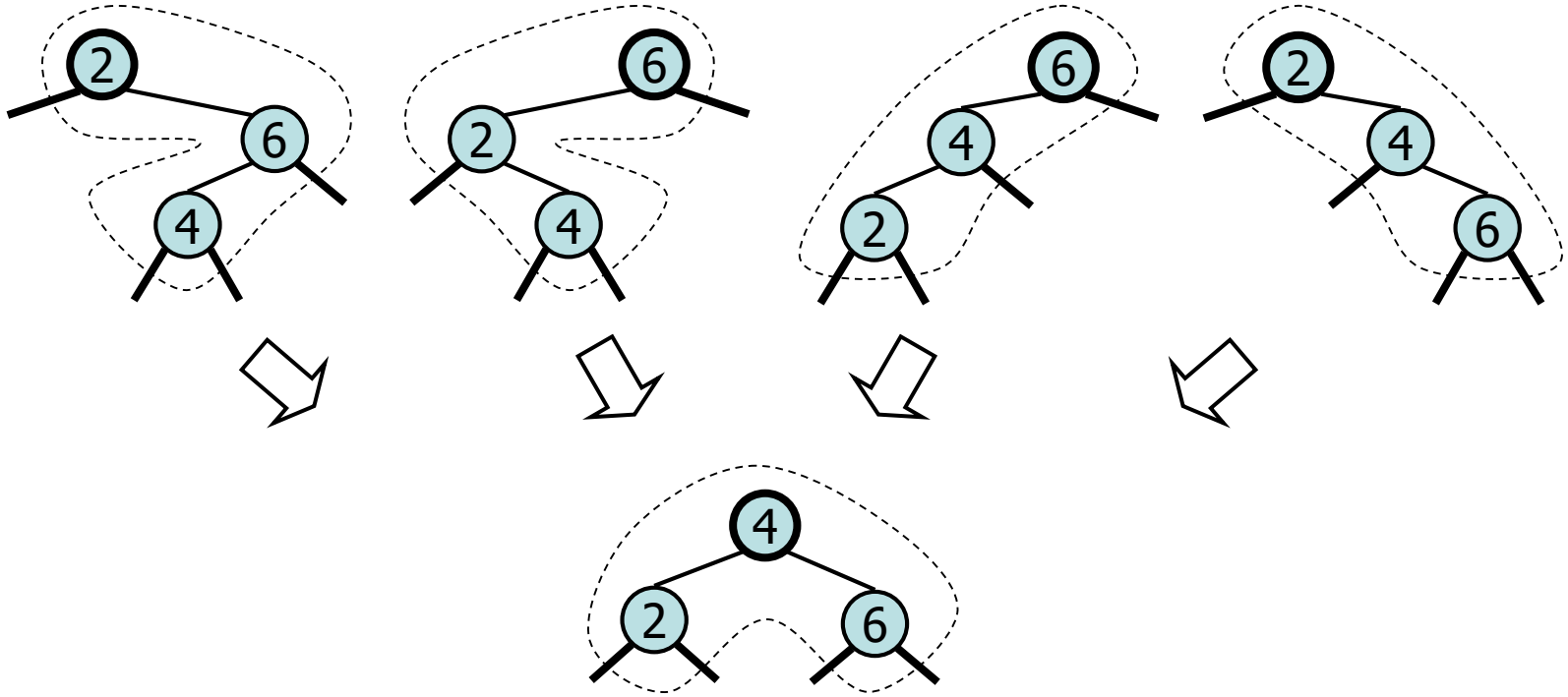
Αναδόμηση

- Μια αναδόμηση αποκαθιστά το διπλό κόκκινο ενός παιδιού-γονέα όταν ο κόκκινος γονέας έχει μαύρο γείτονα
- Είναι αντίστοιχο με την ανάκτηση της σωστής αντικατάστασης ενός 4-node
- Η εσωτερική ιδιότητα επανακτάται και διατηρούνται οι υπόλοιπες ιδιότητες



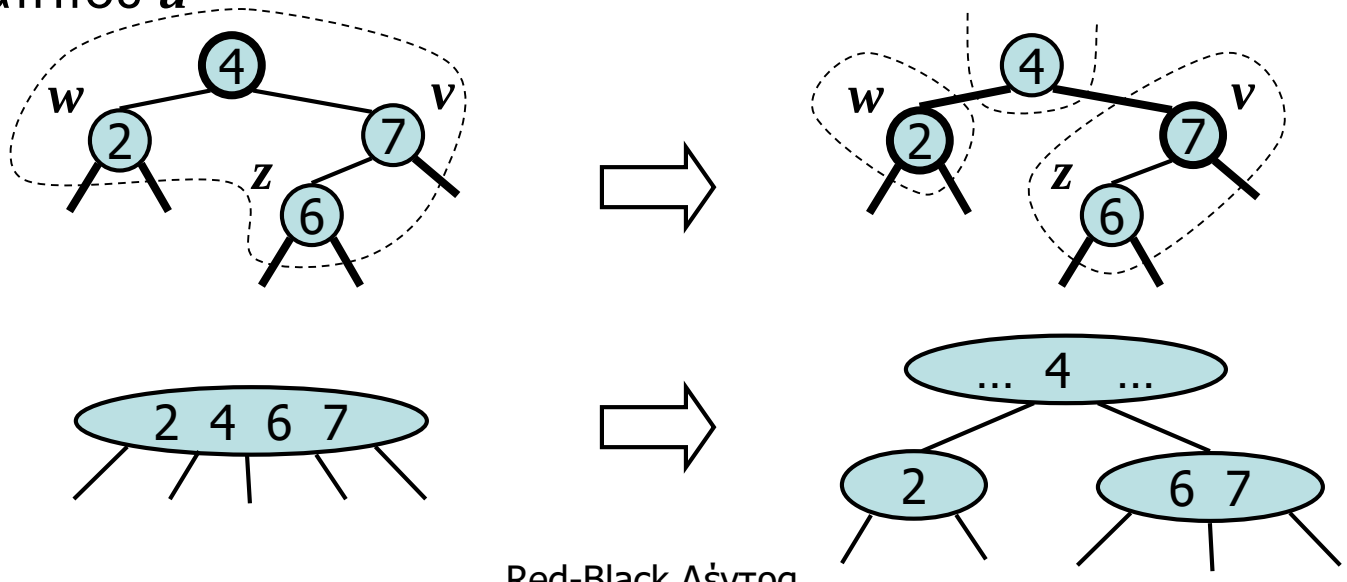
Αναδόμηση (συνέχεια)

- Υπάρχουν τέσσερις ρυθμίσεις αναδόμησης ανάλογα με το αν όι διπλοί κόκκινοι κόμβοι είναι αριστερά ή δεξιά παιδιά



Επαναχρωματισμός

- Ο επαναχρωματισμός αποκαθιστά το διπλό κόκκινο ενός παιδιού-γονέα όταν ο κόκκινος γονέας έχει κόκκινο γείτονα
- Ο γονέας v και ο γείτονας w γίνονται μαύροι και ο παππούς u γίνεται κόκκινος, εκτός αν είναι η ρίζα
- Είναι αντίστοιχο με την εκτέλεση split σε ένα 5-node
- Η παραβίαση του διπλού κόκκινου μπορεί να μεταφερθεί στον παππού u



Ανάλυση της Εισαγωγής

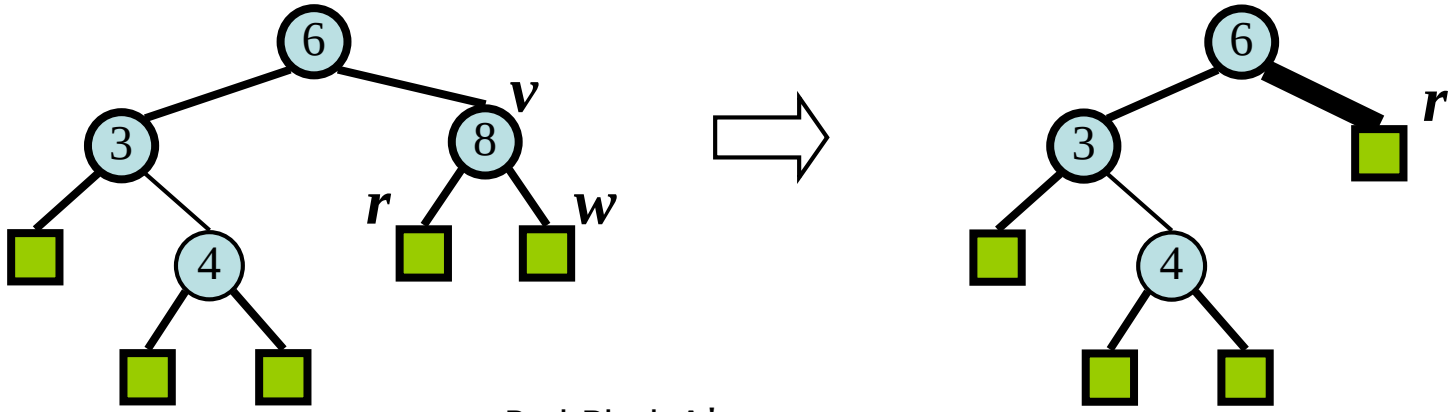
Algorithm ***insertItem***(***k***, ***o***)

1. Αναζητούμε το κλειδί ***k*** για να εντοπίσουμε τον κόμβο εισαγωγής ***z***
2. Προσθέτουμε το νέο αντικείμενο (***k***, ***o***) στον κόμβο ***z*** και χρωματίζουμε το ***z*** κόκκινο
3. **while** ***doubleRed***(***z***)
   if ***isBlack***(***sibling***(***parent***(***z***)))
      z ← ***restructure***(***z***)
      return
   else { ***sibling***(***parent***(***z***) is red }
      z ← ***recolor***(***z***)

- Ένα red-black δέντρο έχει ύψος $O(\log n)$
- Το Βήμα 1 παίρνει $O(\log n)$ χρόνο γιατί επισκεπτόμαστε $O(\log n)$ κόμβους
- Το Βήμα 2 παίρνει $O(1)$ χρόνο
- Το Βήμα 3 παίρνει $O(\log n)$ χρόνο γιατί εκτελούμε
  - $O(\log n)$ επαναχρωματισμούς, ο καθένας σε $O(1)$ χρόνο, και
  - το πολύ μια αναδόμηση σε $O(1)$ χρόνο
- Άρα, μια εισαγωγή σε ένα red-black δέντρο παίρνει $O(\log n)$ χρόνο

Διαγραφή

- Για την εκτέλεση της διεργασίας remove(***k***), πρώτα εκτελούμε τον αλγόριθμο διαγραφής για δυαδικά δέντρα αναζήτησης
- Έστω ***v*** ο εσωτερικός κόμβος που θα διαγραφεί, ***w*** ο εξωτερικός κόμβος που θα διαγραφεί, και ***r*** ο γείτονας του ***w***
  - Αν είτε ο ***v*** ή ο ***r*** ήταν κόκκινοι, χρωματίζουμε τον ***r*** μαύρο
  - Αλλιώς (οι ***v*** και ***r*** ήταν και οι δύο μαύροι) χρωματίζουμε τον ***r*** ***διπλό μαύρο***, που είναι παραβίαση της εσωτερικής ιδιότητας και απαιτεί αναοργάνωση του δέντρου
- Παράδειγμα όπου η διαγραφή του 8 προκαλεί διπλό μαύρο:



Αποκατάσταση Διπλού Μαύρου

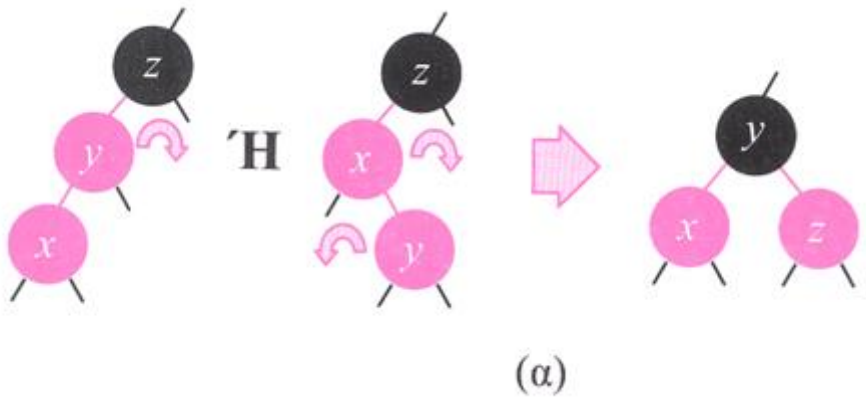
- Ο αλγόριθμος για την αποκατάσταση ενός διπλά μαύρου κόμβου w με γείτονα y θεωρεί τρεις περιπτώσεις

  Περίπτωση 1: ο y είναι μαύρος και έχει ένα κόκκινο παιδί
  - Εκτελούμε μία αναδόμηση, αντίστοιχα με μία transfer, και τερματίζουμε

  Περίπτωση 2: ο y είναι μαύρος και το παιδιά του είναι και τα δύο μαύρα
  - Εκτελούμε επαναχρωματισμό, αντίστοιχα με μία fusion, που μπορεί να μεταφέρει προς τα πάνω την παραβίαση του διπλού μαύρου

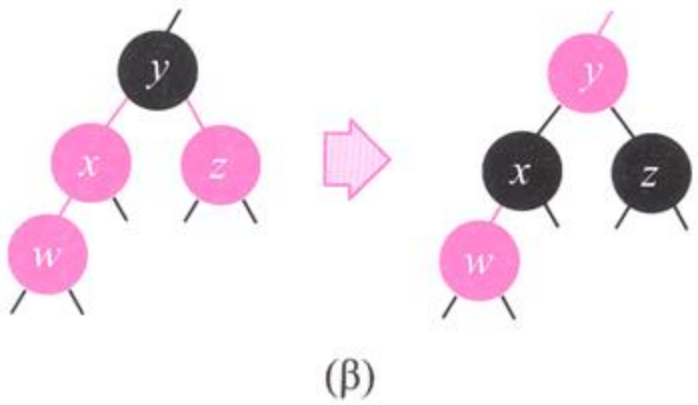
  Περίπτωση 3: ο y είναι κόκκινος
  - Εκτελούμε adjustment, αντίστοιχα της επιλογής διαφορετικής αναπαράστασης ενός 3-node, και μετά καταλήγουμε είτε στην Περίπτωση 1 είτε στην Περίπτωση 2
- Η διαγραφή σε ένα red-black δέντρο παίρνει χρόνο $O(\log n)$

Αναδιατάξεις (1)



Απλή ή διπλή περιστροφή

Αναδιατάξεις (2)



(β)

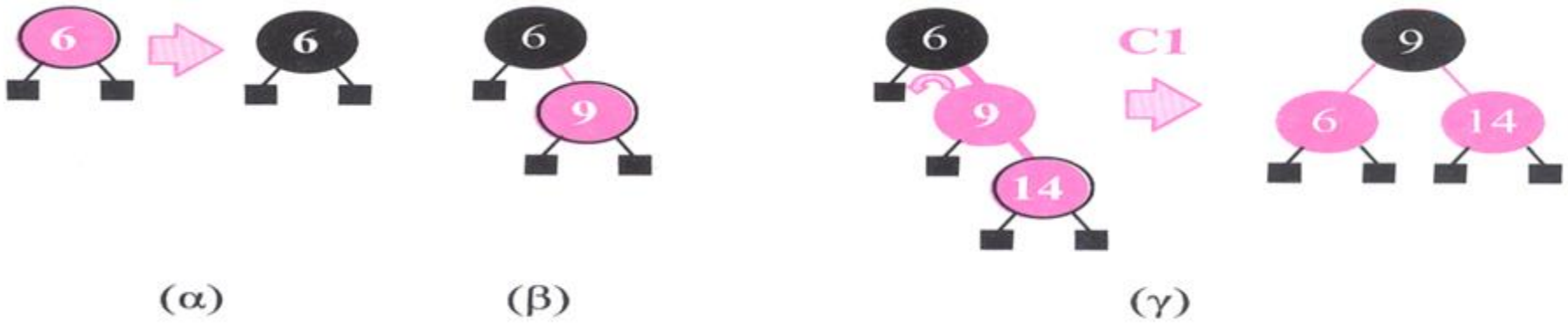
Αντιστροφή χρωμάτων

Παράδειγμα

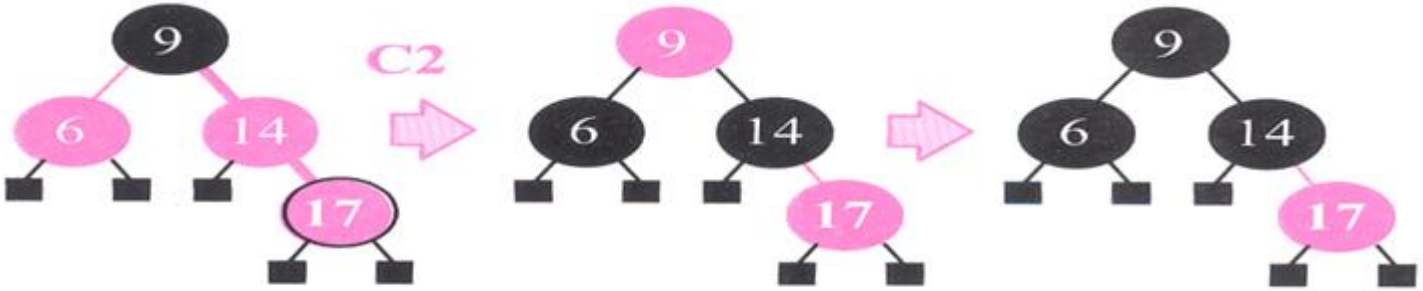
Εισάγονται τα επόμενα κλειδιά διαδοχικά το ένα μετά το άλλο, σε ένα μαύρο-κόκκινο δέντρο: 6, 9, 14, 17, 5, 7, 16, 20, 18, 19, 4 ,11

Να δείχνετε το αποτέλεσμα μετά από κάθε εισαγωγή.

Στην συνέχεια διαγράφονται τα επόμενα κλειδιά 6, 9, 14, 4, 5, 20

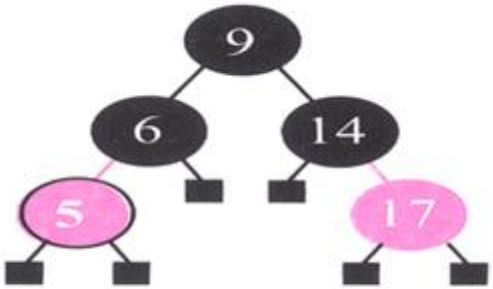


Εισαγωγή 6, 9, 14, 17

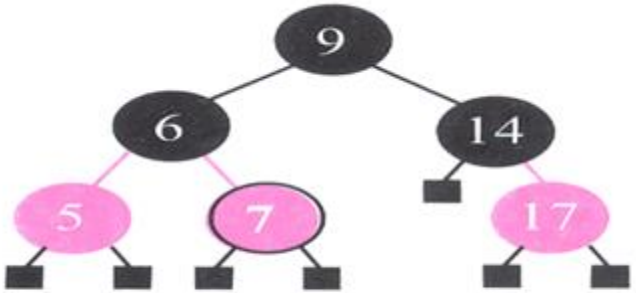


(δ)

Εισαγωγή 17, 5

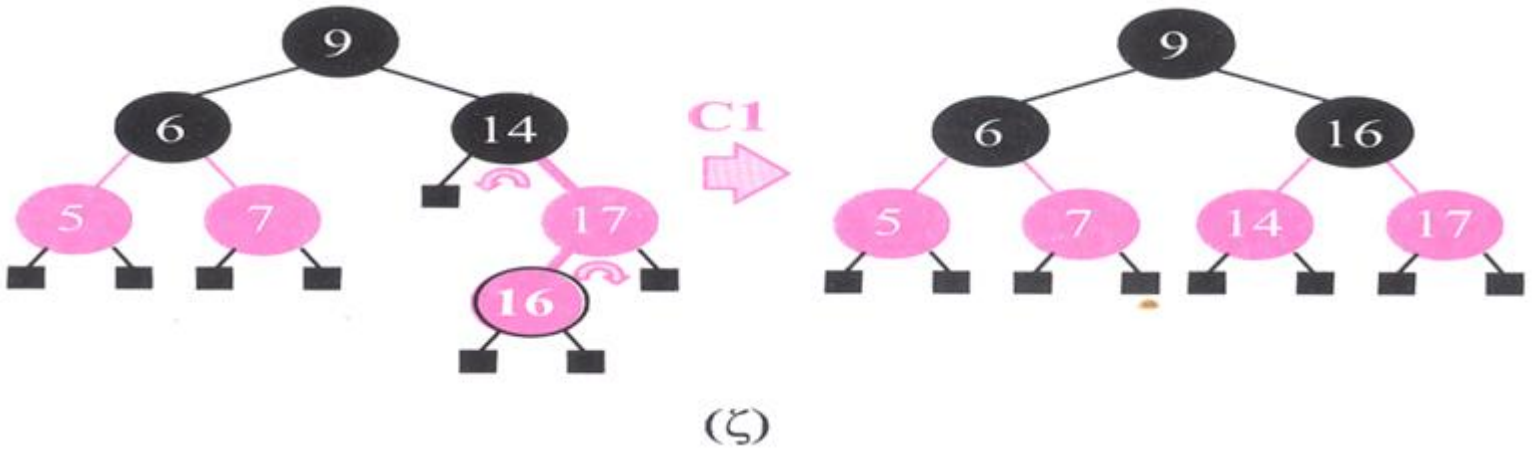


(ε)



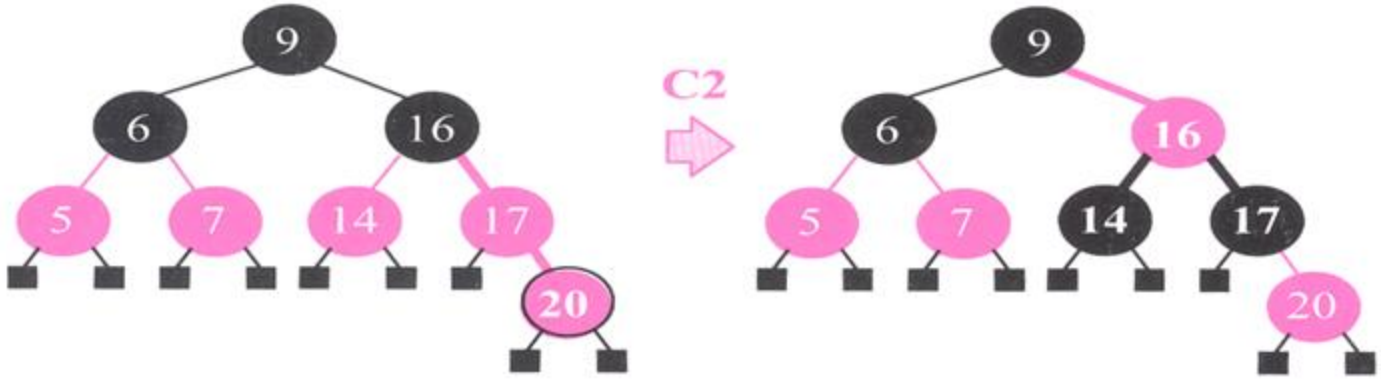
(στ)

Εισαγωγή 5, 7, 16



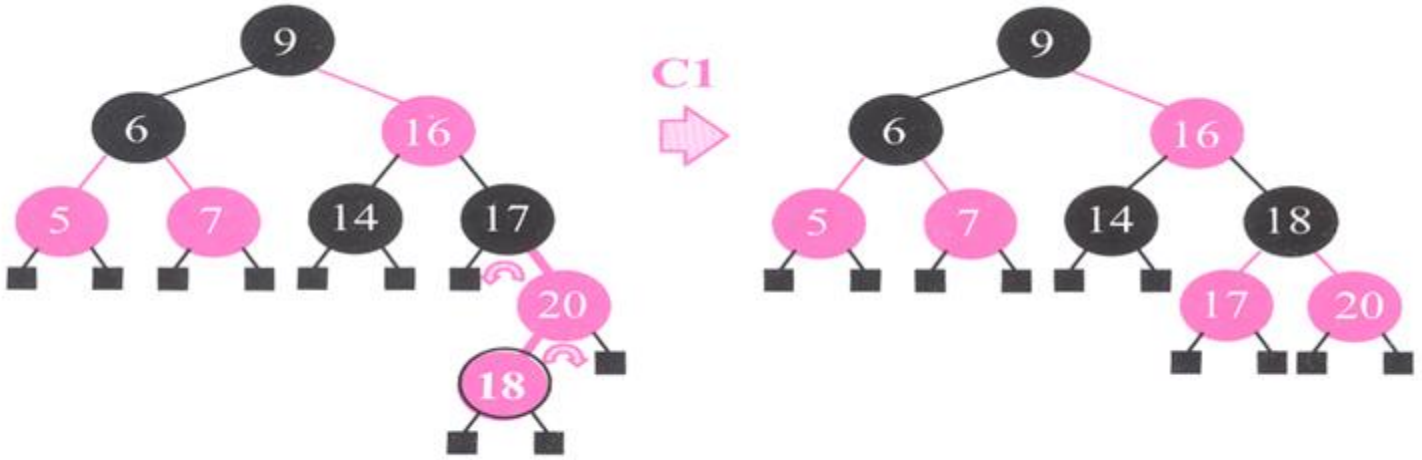
(ζ)

Εισαγωγή 16, 20



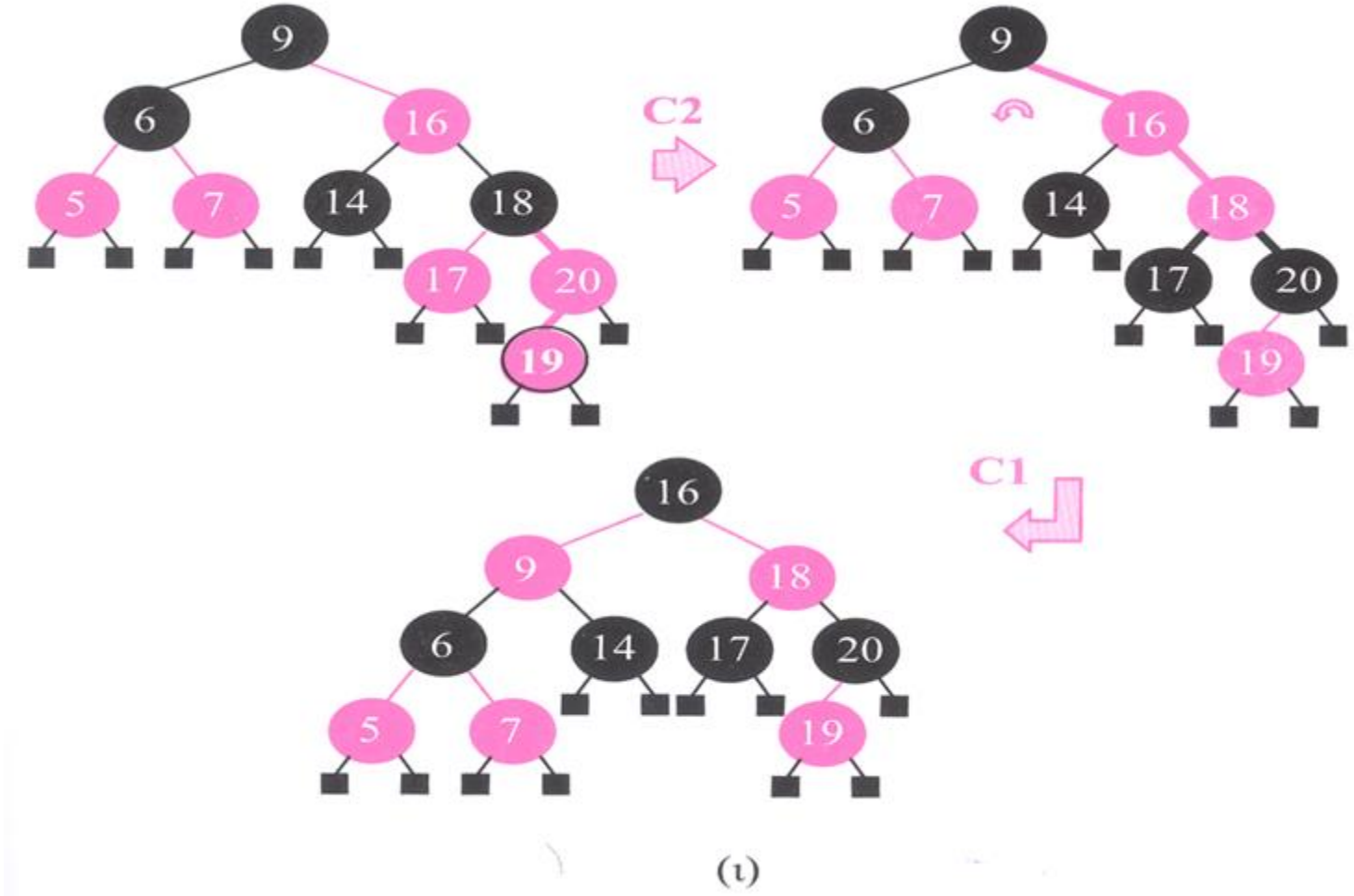
(η)

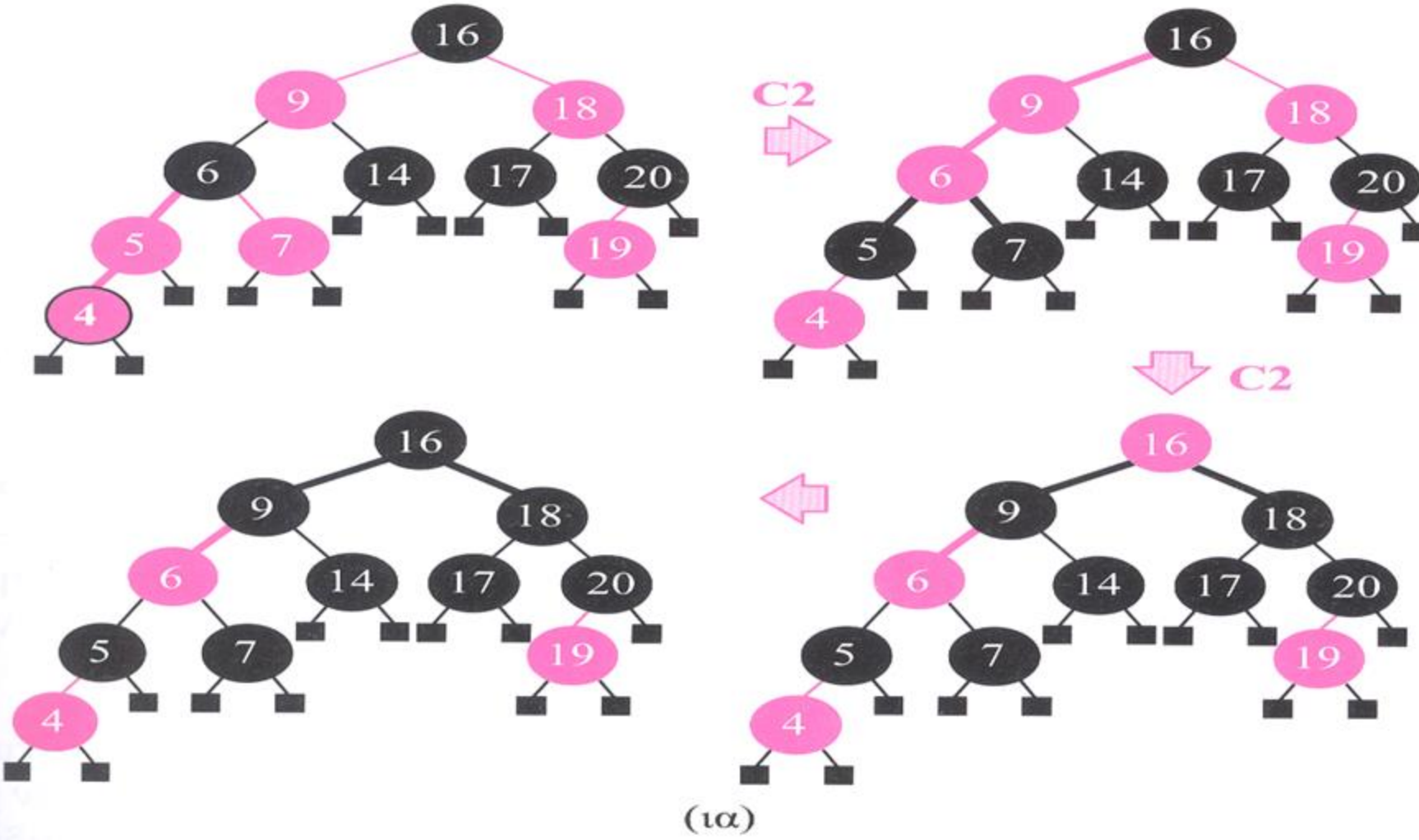
Εισαγωγή 20, 18



(θ)

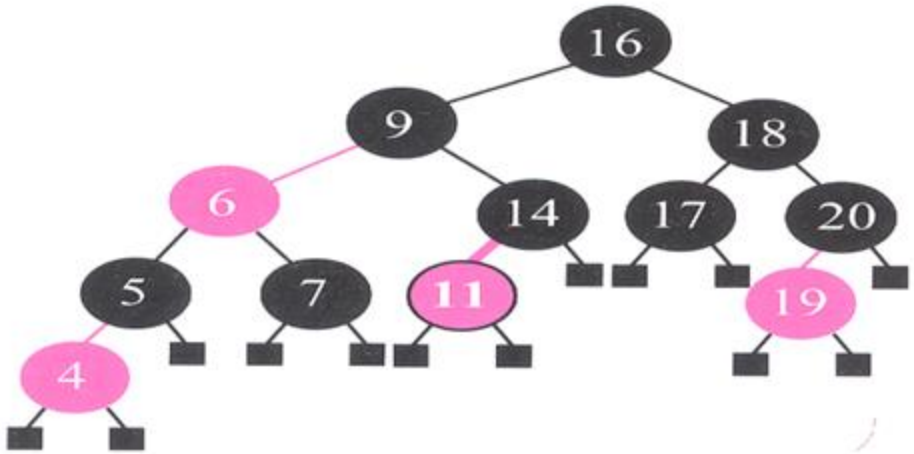
Εισαγωγή 18, 19


9
6
16
5
7
14
18
17
20
19
C2
9
6
16
5
7
14
18
17
20
19
C1
16
9
18
6
14
17
20
5
7
19
(ι)

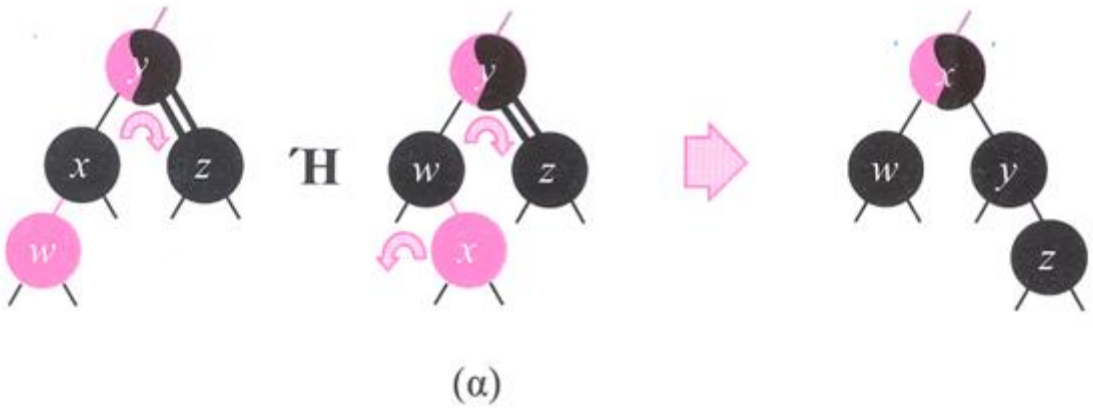


(ια)

Εισαγωγή 4, 11



(ιβ)

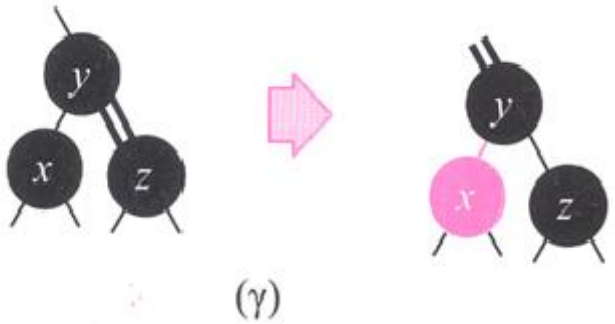


(α)

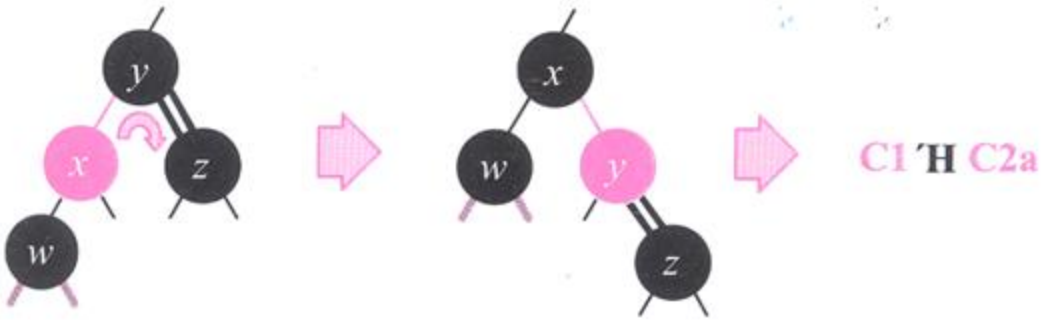
Επαναζυγιστικές πράξεις αποσβέσεις ερυθρόμαυρου δέντρου. Η διπλή μαύρη γραμμή δείχνει απώλεια μαύρου δέντρου.
α) Απλή ή διπλή περιστροφή C1

(β)

β) Τερματικός αναχρωματισμός C2a

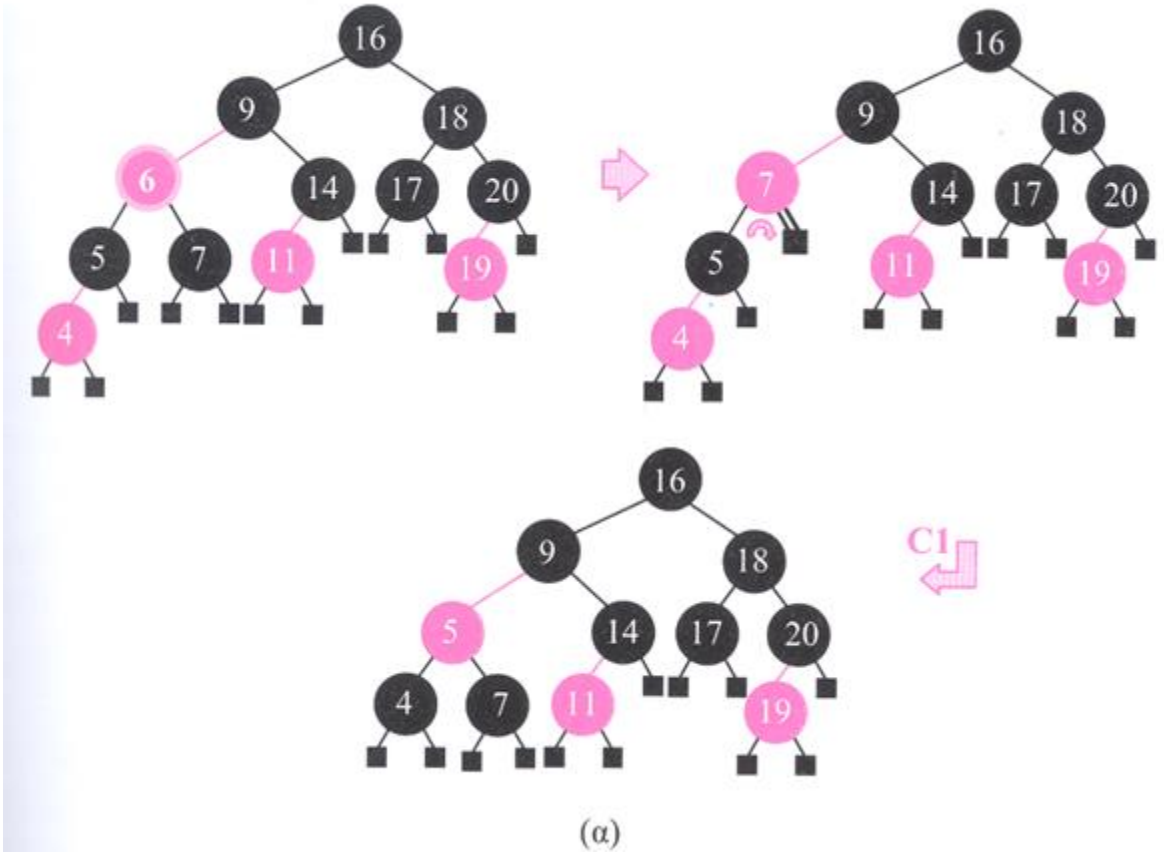


γ) Μη τερματικός αναχρωματισμός C2b

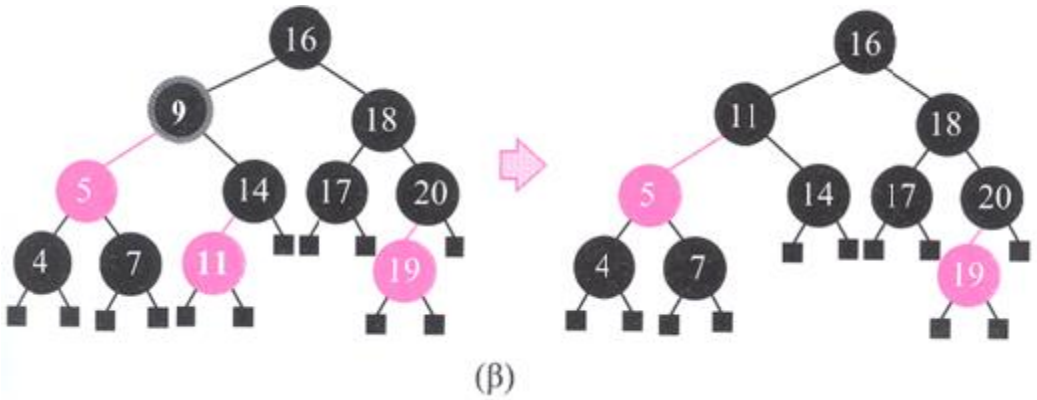


(δ)

Περιστροφή C3, που οδηγεί σε C1 ή C2a

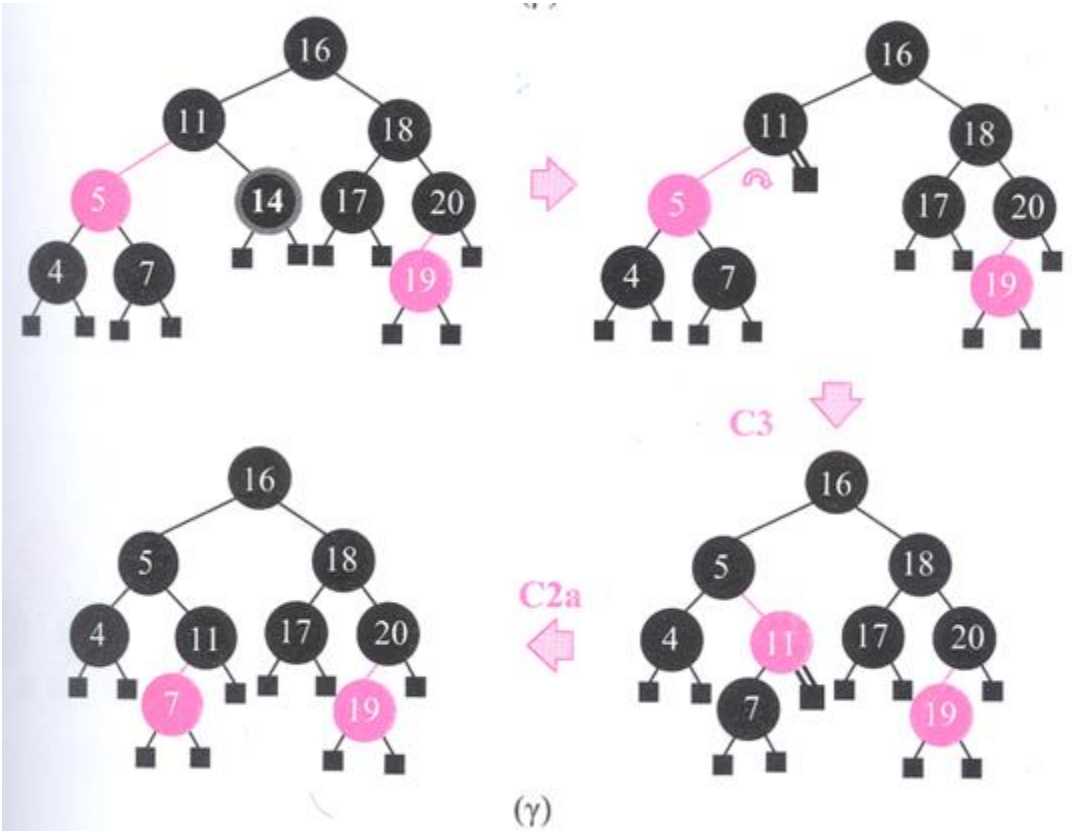


Διαγραφή του 6, στη συνέχεια του 9

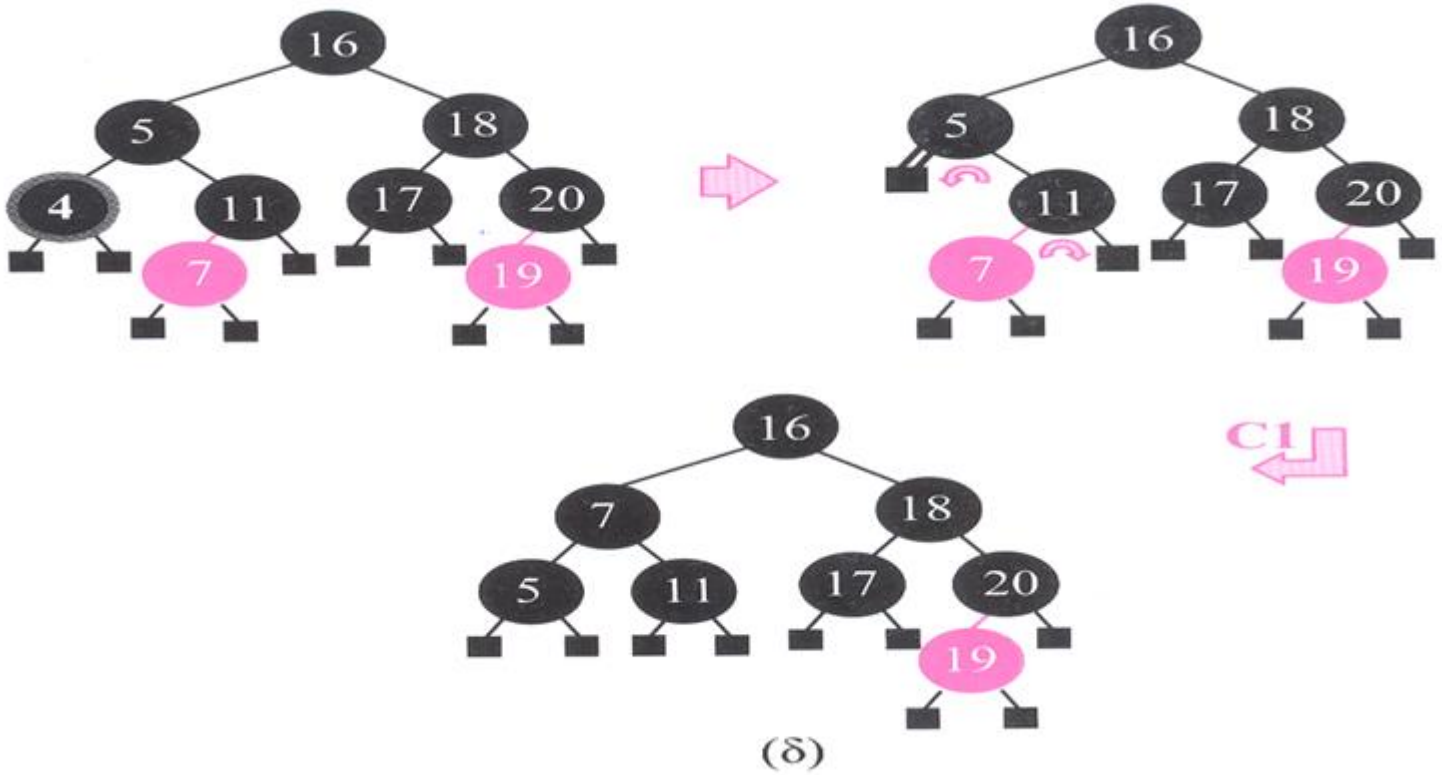


(β)

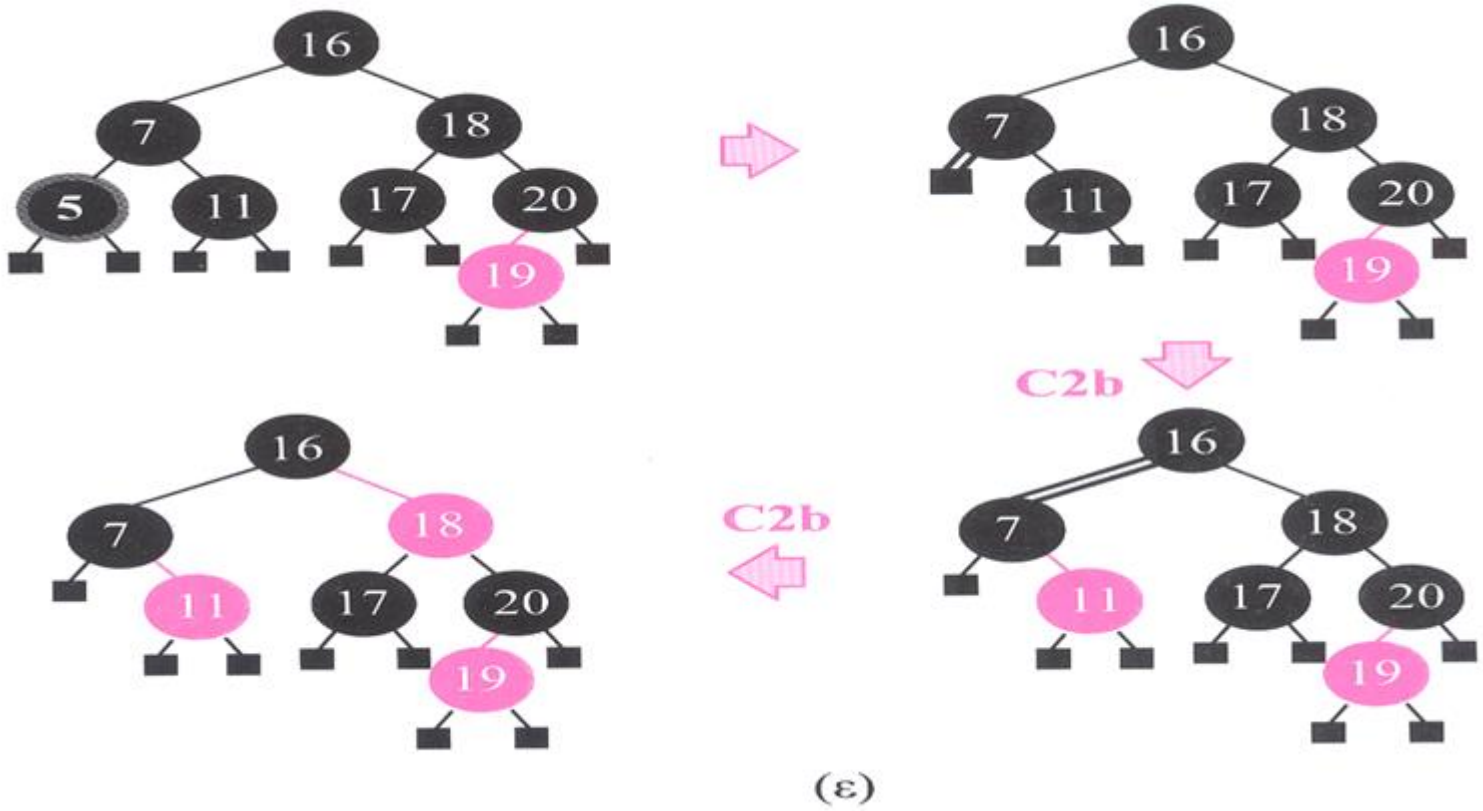
Διαγραφή του 9



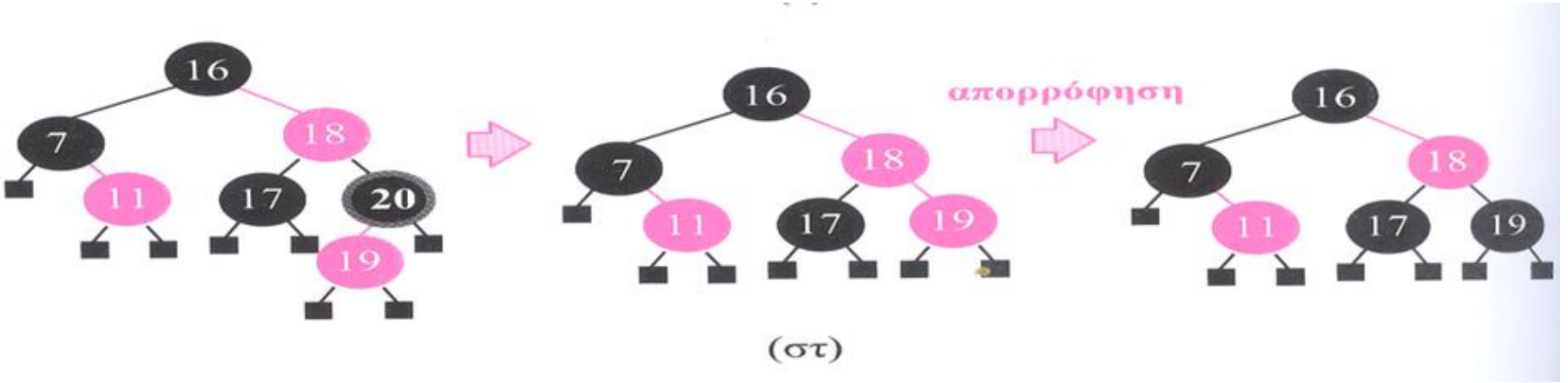
Διαγραφή του 14



Διαγραφή του 4



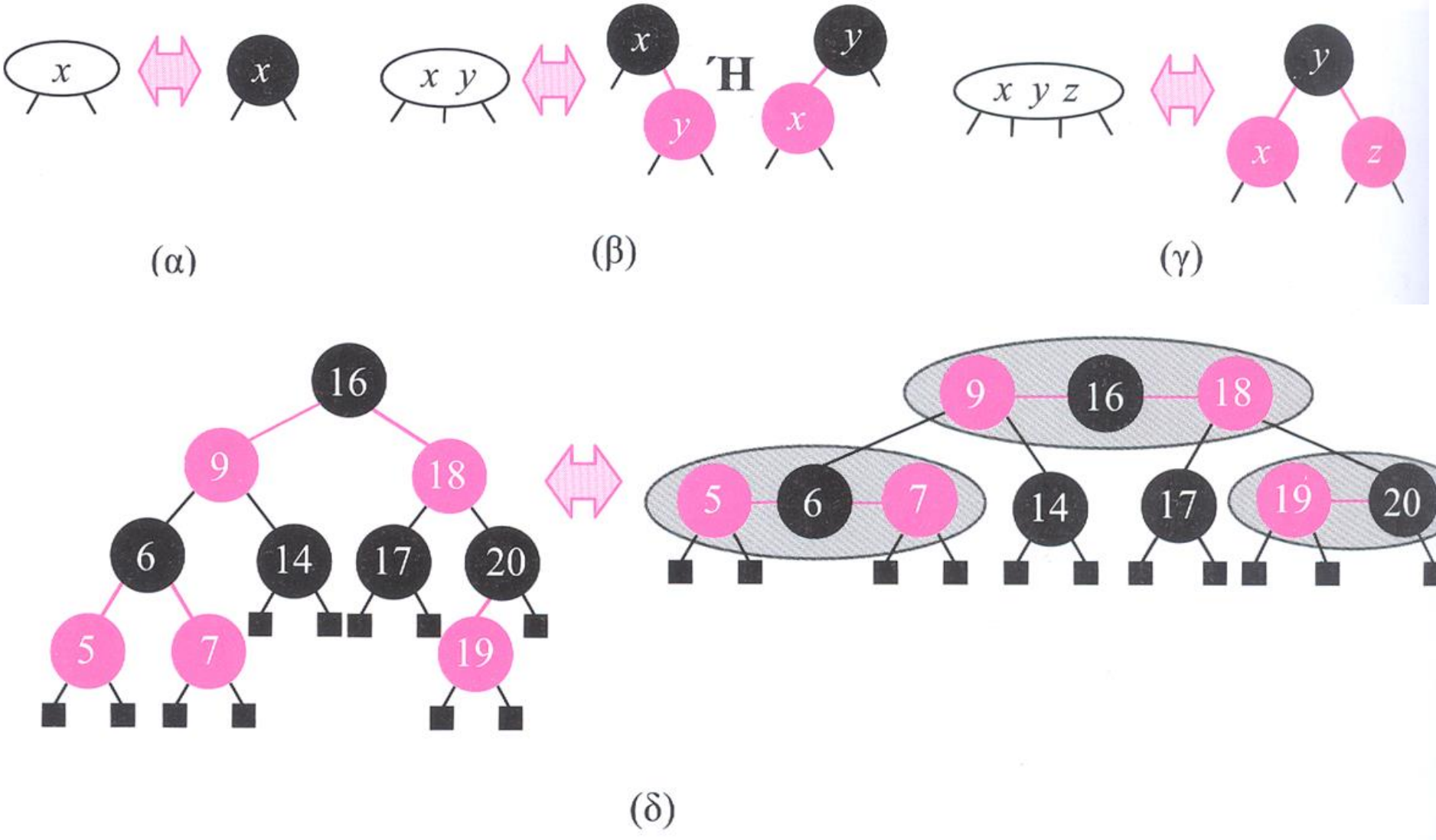
Διαγραφή του 5

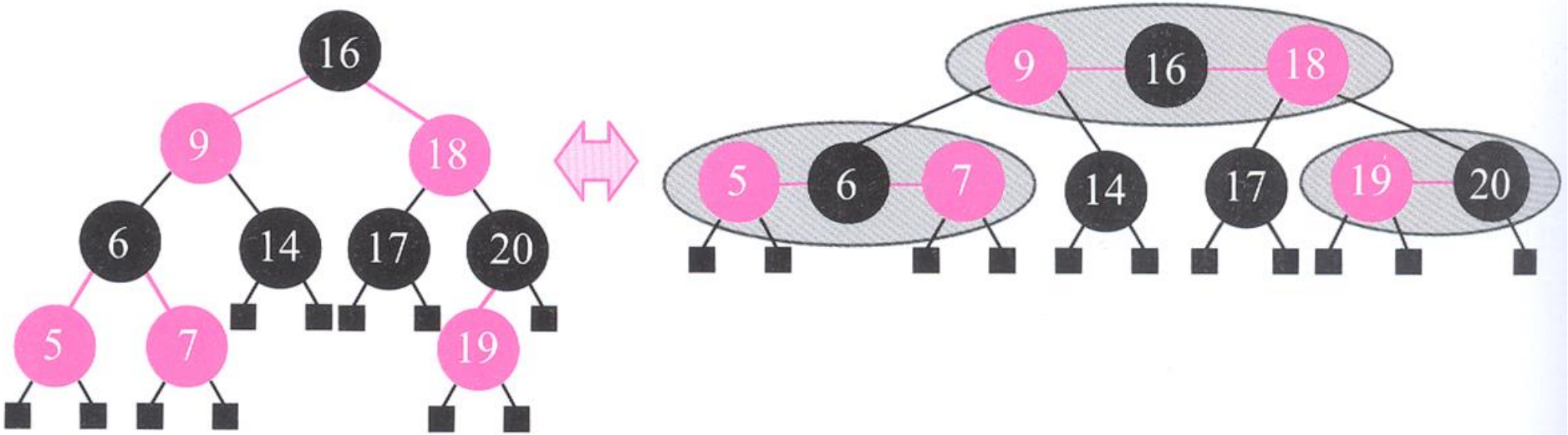


(στ)

Διαγραφή του 20

Ισοδυναμία ερυθρόμαυρων δέντρων με δέντρα (2-4)





(δ)