Δυαδικά Δένδρα

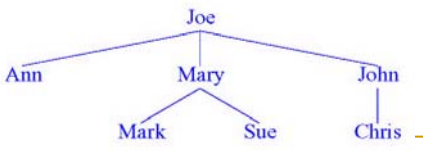
Γιάννης Θεοδωρίδης, Νίκος Πελέκης, Άγγελος Πικράκης
Τμήμα Πληροφορικής

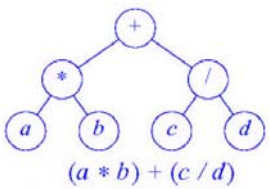
Ορισμοί

(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 (συν.)

(αναδρομικός ορισμός)

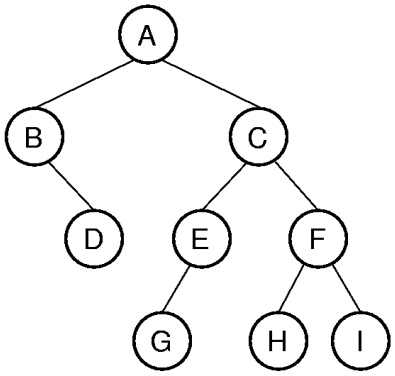
Ένα δυαδικό δένδρο t είναι μία πεπερασμένη (πιθανώς κενή) συλλογή στοιχείων. Όταν το δυαδικό δένδρο δεν είναι κενό, τότε έχει μία ρίζα και τα υπόλοιπα στοιχεία (αν υπάρχουν) επιμερίζονται σε δύο δυαδικά δένδρα που ονομάζονται το αριστερό και το δεξιό υποδένδρο του t

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 (συν.)

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



Πλήρη και Συμπληρωμένα Δυαδικά Δένδρα

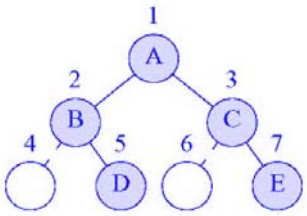
- Πλήρες (full) δυαδικό δένδρο ύψους h είναι εκείνο το δυαδικό δένδρο, το οποίο περιέχει ακριβώς 2^h-1 στοιχεία
  - Μπορούμε να αριθμήσουμε από 1 έως 2^h-1 τα στοιχεία ενός πλήρους δυαδικού δένδρου ύψους h, ξεκινώντας από το επίπεδο 1 προς το επίπεδο h και, μέσα σε κάθε επίπεδο, από αριστερά προς τα 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 ($1 \leq i \leq n$) 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 $2i \leq n$, αλλιώς δεν έχει αριστερό παιδί)
  - Το δεξιό του παιδί έχει αριθμό 2i+1 (αν $2i+1 \leq n$, αλλιώς δεν έχει δεξιό παιδί)
- Αποδείξεις με μαθηματική επαγωγή

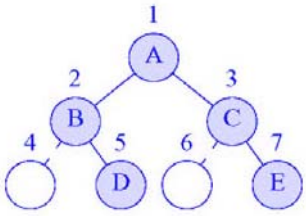
Στατική αναπαράσταση δυαδικού δένδρου (με πίνακα)

- 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)



A	B	C		D		E

Υλοποίηση δυαδικού δένδρου με πίνακα



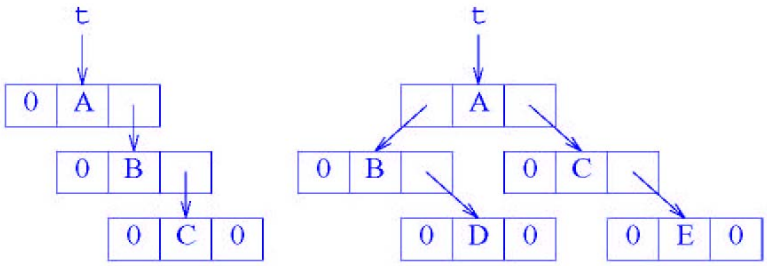
A	B	C		D		E

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 συμπληρωμένων 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)



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

}

Κλάση BinaryTreeNode

```
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
}
```


Κλάση BinaryTree

```
class BinaryTree {
  public:
     BinaryTree() { root = 0; };
     ~BinaryTree() { };
     bool IsEmpty () const {
        return ((root) ? false : true);
     }
     bool Root (T& x) const {
        // Set x to root data - Return false if no root
        if (root) { x = root->data; return true;}
        else return false;       // no root
     }
     ...
   private:
     BinaryTreeNode<T> *root;      // pointer to root
     ...
};
```

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 (συν.)

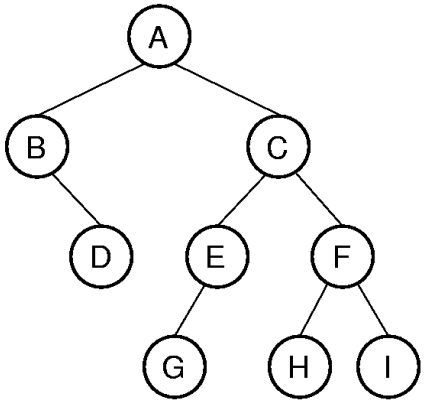
...

- **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 4η είναι επαναληπτική

Διάσχιση δένδρου (συν.)

- 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

```
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

```
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;}
   }
}
```

Εφαρμογή διάσχισης: υπολογισμός ύψους ΔΔ

```
int Height(BinaryTreeNode<T> *t) const {
  // Return height of tree *t - PostOrder traversal!!!
  if (!t) return 0;               // empty tree
  int hl = Height(t->LeftChild);  // height of
                                     left subtree
  int hr = Height(t->RightChild); // height of
                                     right subtree
  if (hl > hr) return ++hl;
  else return ++hr;
  }
```