Διάλεξη 22 - Δέντρα

Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

Εισαγωγή στον Προγραμματισμό

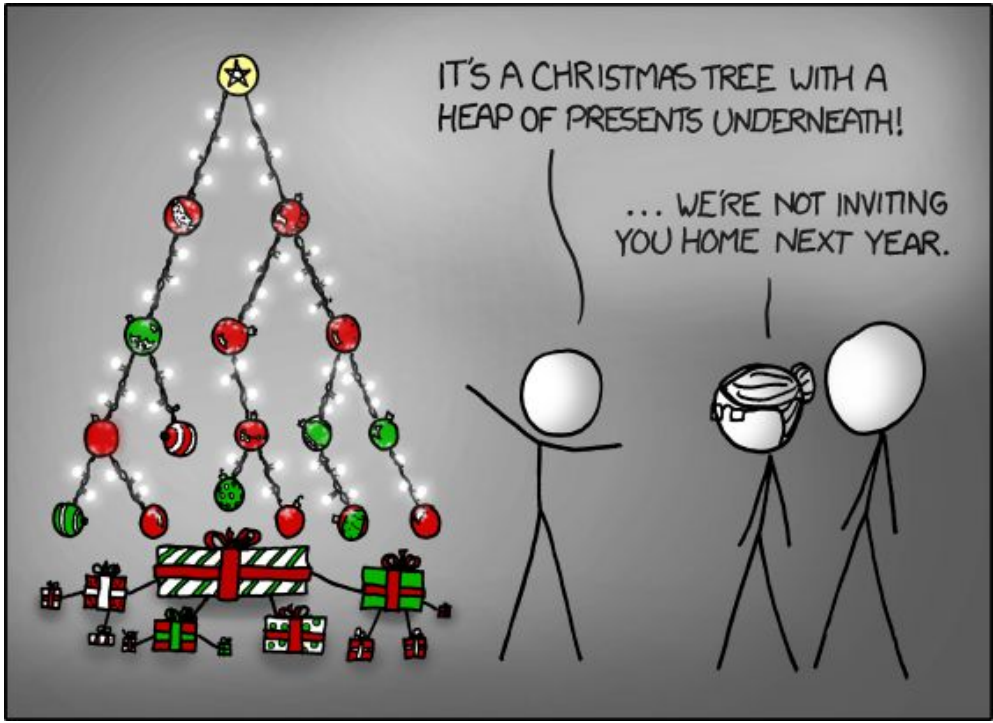
Θανάσης Αυγερινός / Τάκης Σταματόπουλος

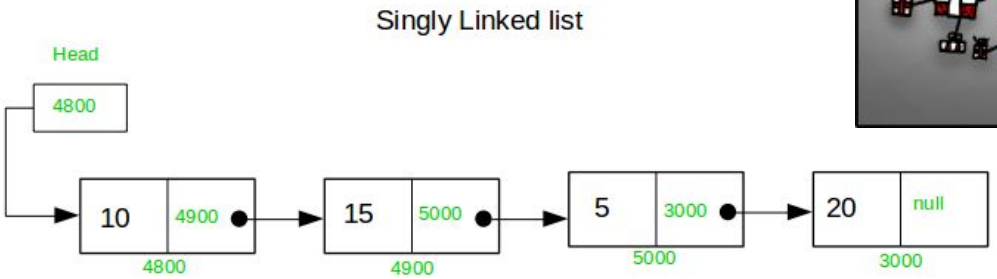
Ανακοινώσεις / Διευκρινήσεις

- Η εργασία #2 βγήκε την Παρασκευή - προθεσμία 10 Ιανουαρίου 23:59

Την προηγούμενη φορά

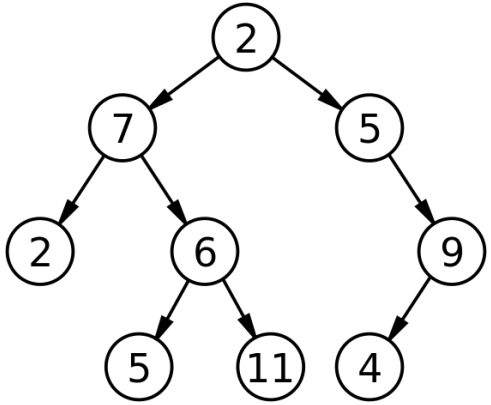
- Αυτοαναφορικές Δομές
  - Λίστες
  - Λίγο για Δέντρα
  - Αλγόριθμοι χρήσης και διάσχισης

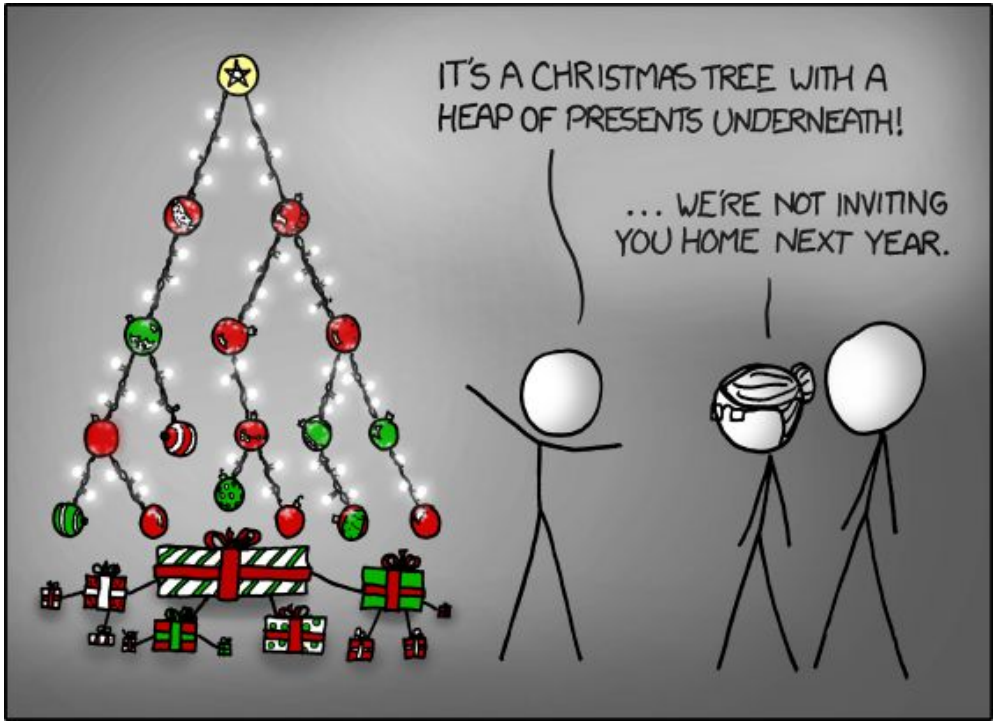




Σήμερα

- Αυτοαναφορικές Δομές
  - Δέντρα
  - Αλγόριθμοι χρήσης και διάσχισης





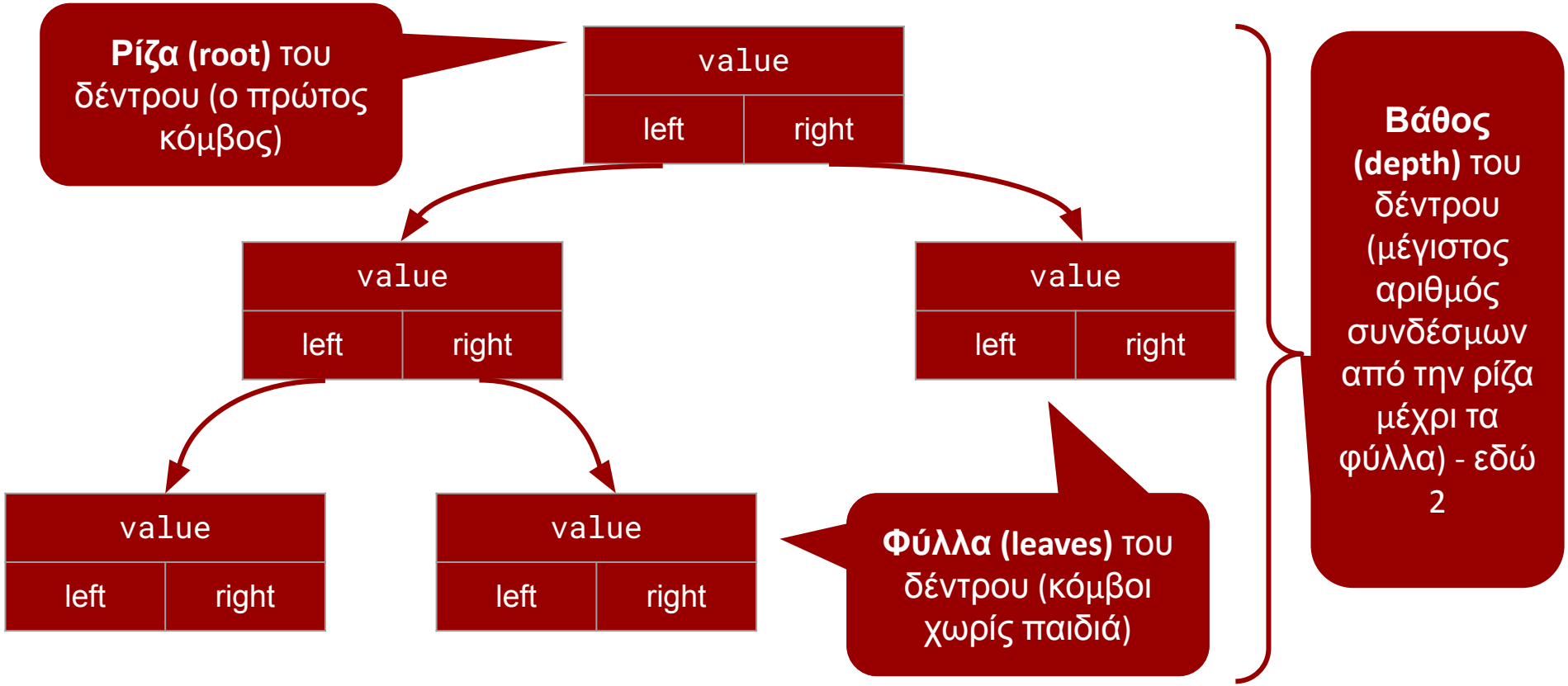
Δυαδικό Δέντρο (Binary Tree)

Το δυαδικό δέντρο (binary tree) είναι ένας τύπος δεδομένων που μας επιτρέπει να οργανώνουμε τα δεδομένα μας σε δενδρική διάταξη, καθένας από τους κόμβους του δέντρου μπορεί να έχει από 0 μέχρι 2 κόμβους-παιδιά.

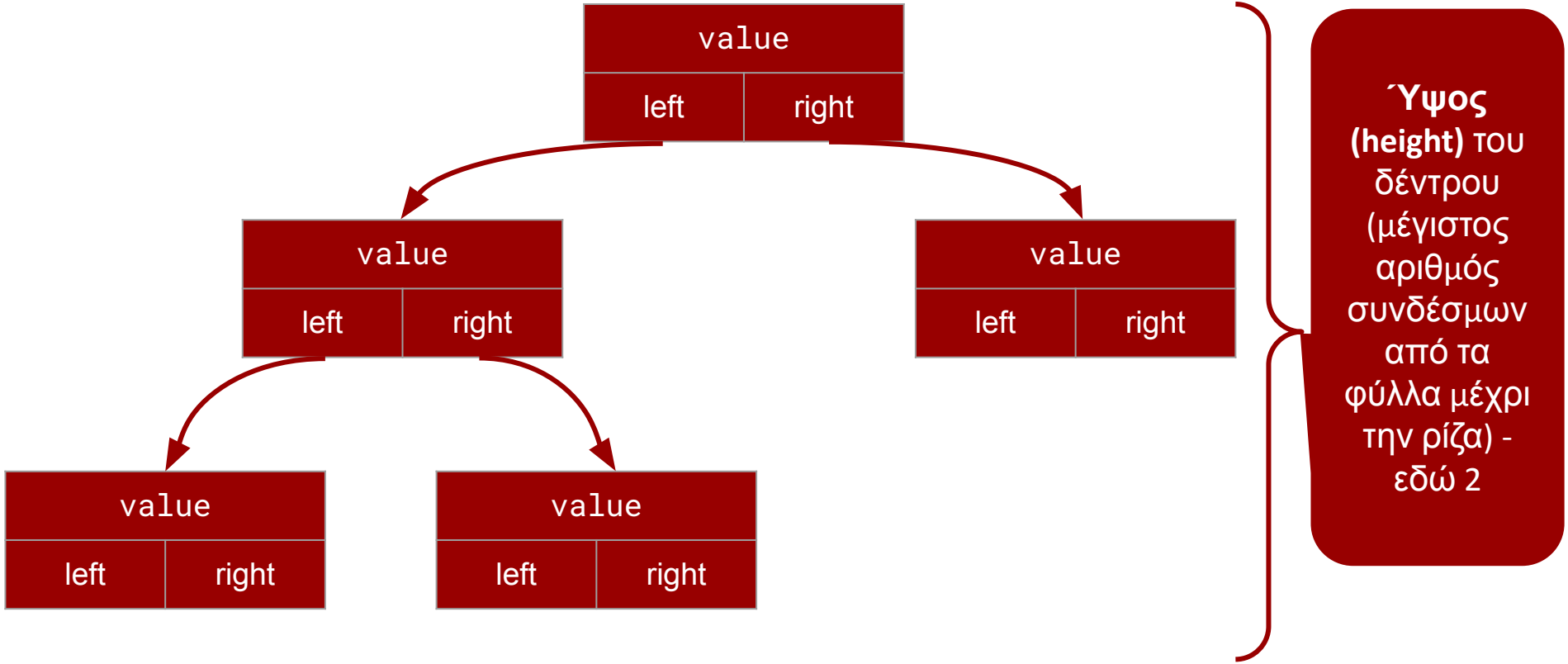
```
struct treenode {
  int value;
  struct treenode * left;
  struct treenode * right;
};
```

Εφαρμογές: από βάσεις δεδομένων/αναζήτηση μέχρι μεταγλωττιστές και από συμπίεση δεδομένων μέχρι κρυπτογραφία (όπου απαιτείται αναπαράσταση γνώσης)

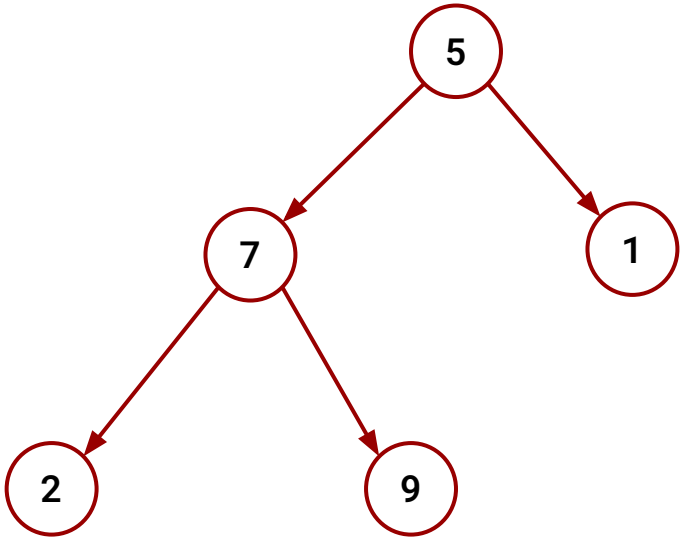
Δυαδικά Δέντρα (Binary Trees)



Δυαδικά Δέντρα (Binary Trees)

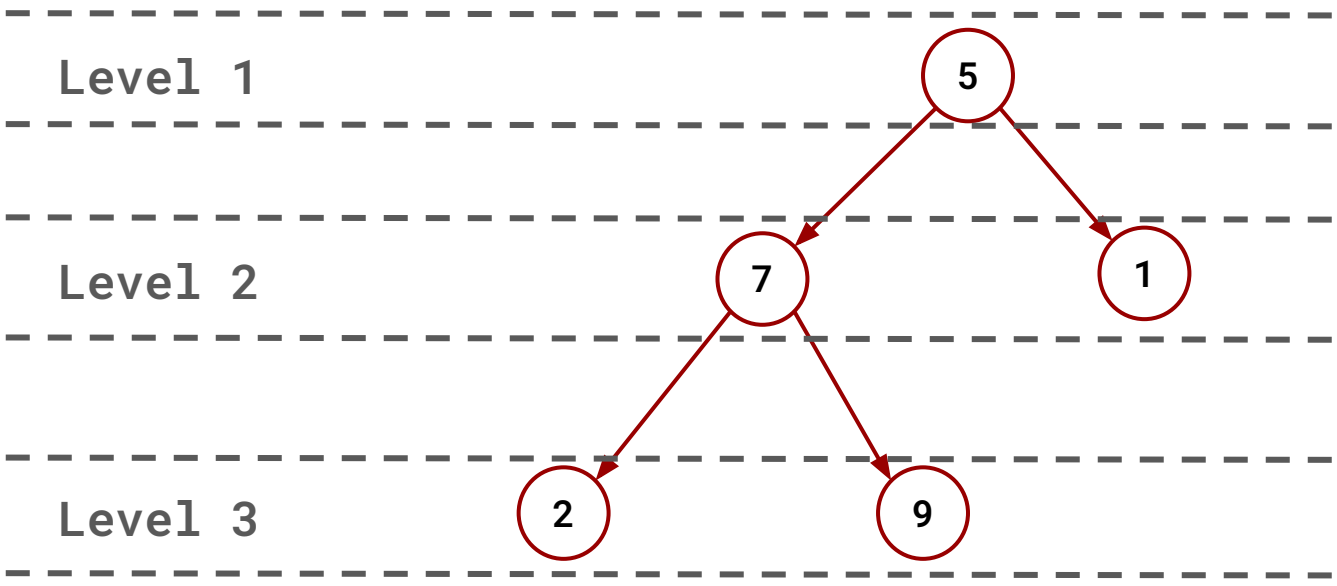


Δυαδικά Δέντρα (Binary Trees)



Επίπεδο Κόμβου (Node Level)

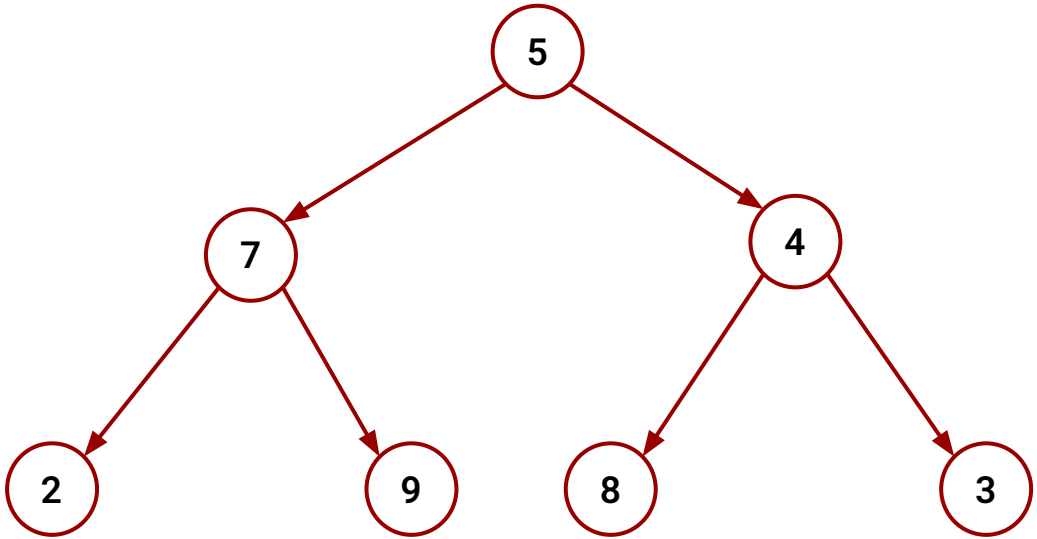
Το επίπεδο ενός κόμβου σε ένα δέντρο ισούται με τον αριθμό των κόμβων που μεσολαβούν μέχρι την ρίζα του δέντρου.



Τύποι Δυαδικών Δέντρων - Τέλειο (Perfect)

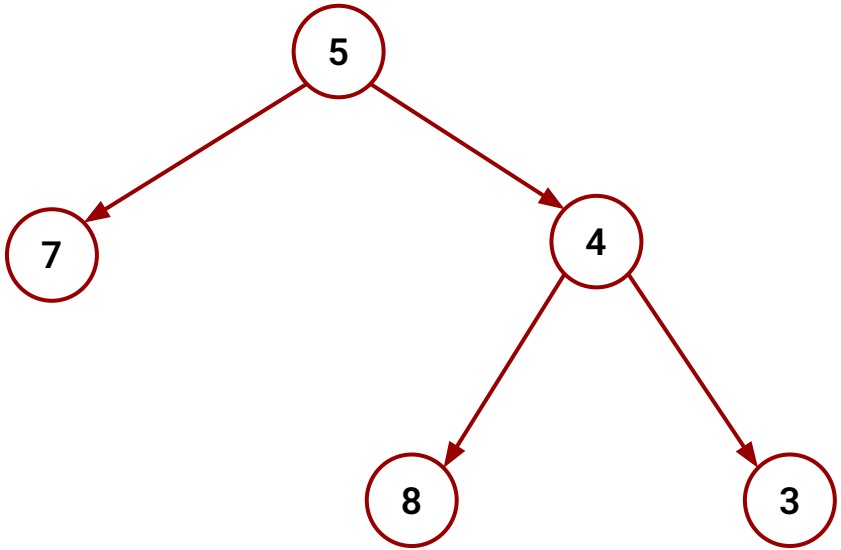
Τέλειο Δυαδικό Δέντρο (Perfect Binary Tree): ένα δυαδικό δέντρο που όλοι οι εσωτερικοί κόμβοι έχουν δύο παιδιά και όλα τα φύλλα βρίσκονται στο ίδιο επίπεδο.

Πόσοι κόμβοι σε n επίπεδα;



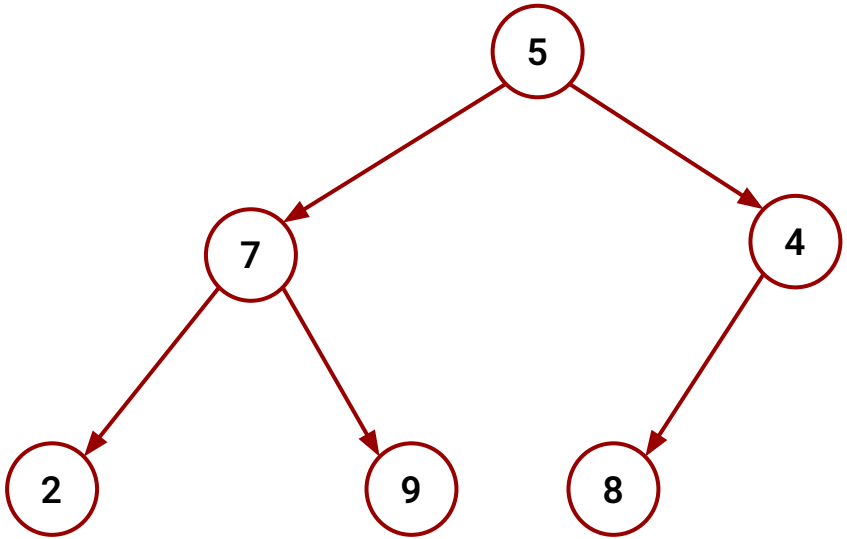
Τύποι Δυαδικών Δέντρων - Γεμάτο (Full)

Γεμάτο Δυαδικό Δέντρο (Full Binary Tree): ένα δυαδικό δέντρο που όλοι οι κόμβοι έχουν 0 ή 2 παιδιά.



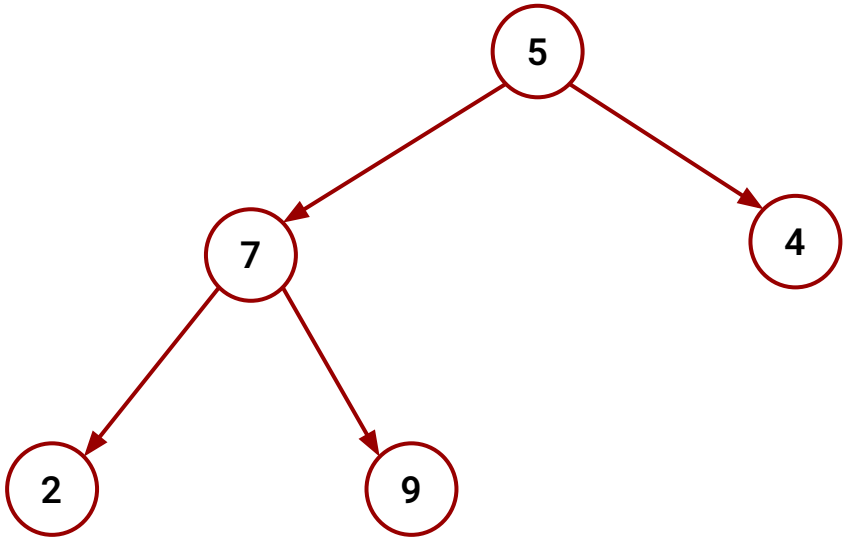
Τύποι Δυαδικών Δέντρων - Πλήρες (Complete)

Πλήρες Δυαδικό Δέντρο (Complete Binary Tree): ένα δυαδικό δέντρο που σε κάθε επίπεδο εκτός πιθανώς από το τελευταίο είναι γεμάτο και όλοι οι κόμβοι στο τελευταίο επίπεδο είναι όσο πιο αριστερά γίνεται.



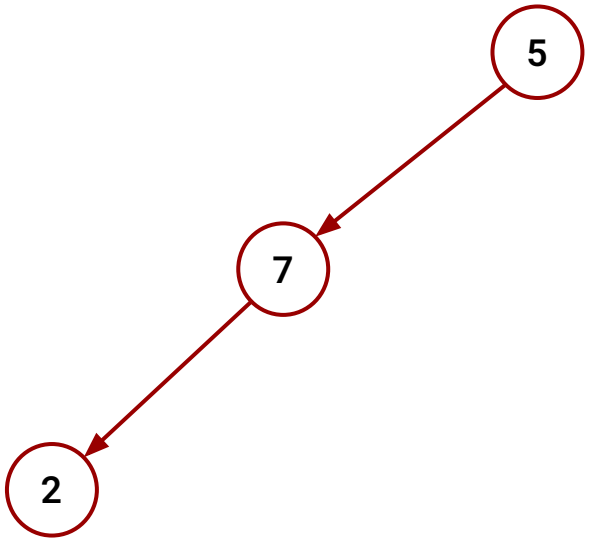
Τύποι Δυαδικών Δέντρων - Ισορροπημένο (Balanced)

Ισορροπημένο Δυαδικό Δέντρο (Balanced Binary Tree): ένα δυαδικό δέντρο που σε κάθε κόμβο το ύψος του αριστερού και το δεξιού υποδέντρο μπορούν να διαφέρουν μέχρι 1.



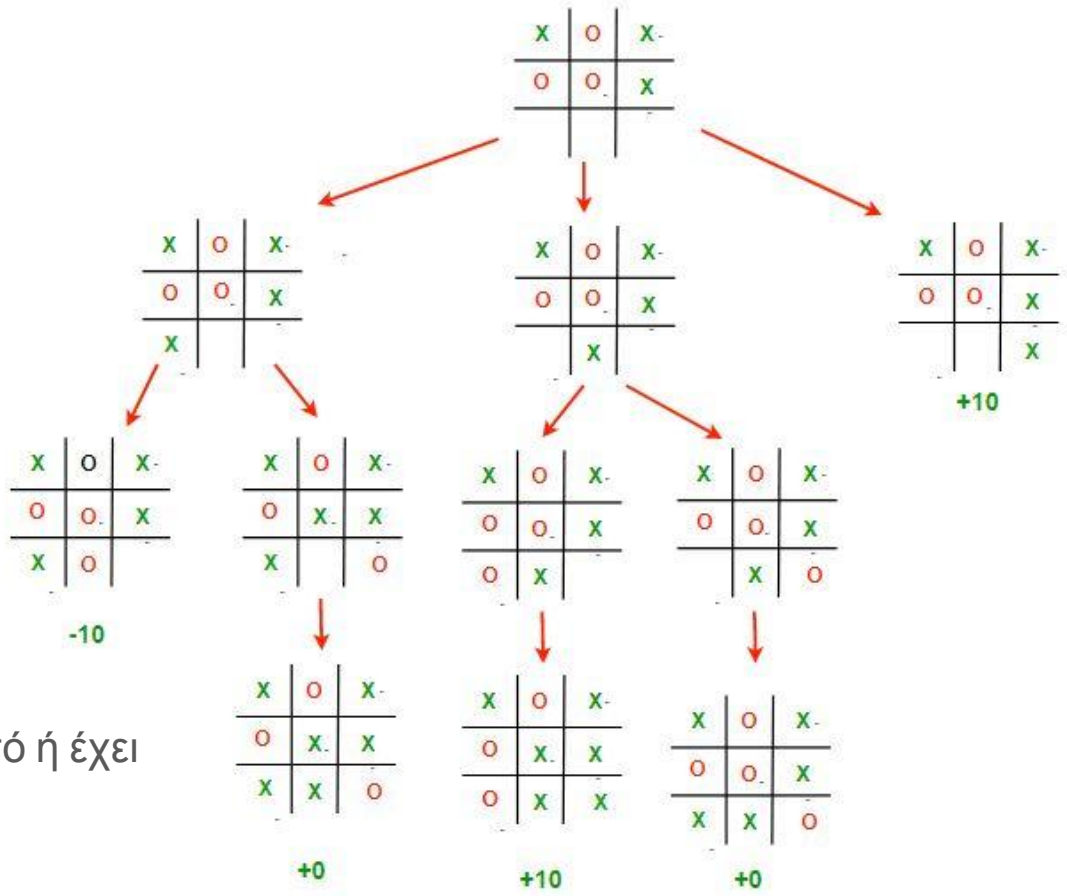
Τύποι Δυαδικών Δέντρων - Εκφυλισμένο (Degenerate)

Εκφυλισμένο Δυαδικό Δέντρο (Degenerate Binary Tree): ένα δυαδικό δέντρο που ο κάθε κόμβος έχει μέχρι ένα παιδί.



Ν-αδικά Δέντρα

Είναι συνηθισμένο να αναπαριστούμε καταστάσεις χρησιμοποιώντας δομές όπως τα δέντρα, κάποιες φορές με περισσότερα από 2 παιδιά κόμβους.



Αυτό το δέντρο καταστάσεων είναι σωστό ή έχει σφάλματα; Πως θα τα βρίσκαμε με έναν αλγόριθμο αυτόματα;

Βασικές Λειτουργίες Με Δυαδικά Δέντρα

1. is_empty: Έλεγχος εάν το δέντρο είναι άδειο
2. depth: Εύρεση βάθους δέντρου
3. print: Τύπωμα στοιχείων δέντρου
4. find: Εύρεση στοιχείου σε δέντρο
5. insert: Προσθήκη στοιχείου στο δέντρο (μόνοι σας)
6. delete: Αφαίρεση στοιχείου από δέντρο (μόνοι σας)

is_empty: Έλεγχος αν το δέντρο είναι άδειο

```c
#include <stdio.h>
#include <stdlib.h>
typedef struct treenode {int value;  struct treenode * left;  struct treenode * right;} * Tree;
int is_empty(Tree t) {
  return t == NULL;
}
int main() {
  Tree t = NULL;
  printf("Empty: %d\n", is_empty(t));
  return 0;
}
```

```
$ ./tree
Empty: 1
```

depth: Εύρεση βάθους δέντρου

depth: Εύρεση βάθους δέντρου

```c
#include <stdio.h>

#include <stdlib.h>

typedef struct treenode {  int value;  struct treenode * left;  struct treenode * right;} * Tree;

int depth(Tree t) {

  if (t == NULL) return -1;

  int left_depth = depth(t->left);

  int right_depth = depth(t->right);

  return 1 + ((left_depth > right_depth) ? left_depth : right_depth);

}

int main() {

  struct treenode t2 = {2, NULL, NULL},   t9 = {9, NULL, NULL}, t1 = {1, NULL, NULL};

  struct treenode t7 = {7, &t2, &t9}; struct treenode t5 = {5, &t7, &t1};

  Tree t = &t5;

  printf("Depth: %d\n", depth(t));

  return 0;

}
```

depth: Εύρεση βάθους δέντρου

```c
#include <stdio.h>

#include <stdlib.h>

typedef struct treenode {  int value;  struct treenode * left;  struct treenode * right;} * Tree;

int depth(Tree t) {

  if (t == NULL) return -1;

  int left_depth = depth(t->left);

  int right_depth = depth(t->right);

  return 1 + ((left_depth > right_depth) ? left_depth : right_depth);

}

int main() {

  struct treenode t2 = {2, NULL, NULL},   t9 = {9, NULL, NULL}, t1 = {1, NULL, NULL};

  struct treenode t7 = {7, &t2, &t9}; struct treenode t5 = {5, &t7, &t1};

  Tree t = &t5;

  printf("Depth: %d\n", depth(t));

  return 0;

}
```

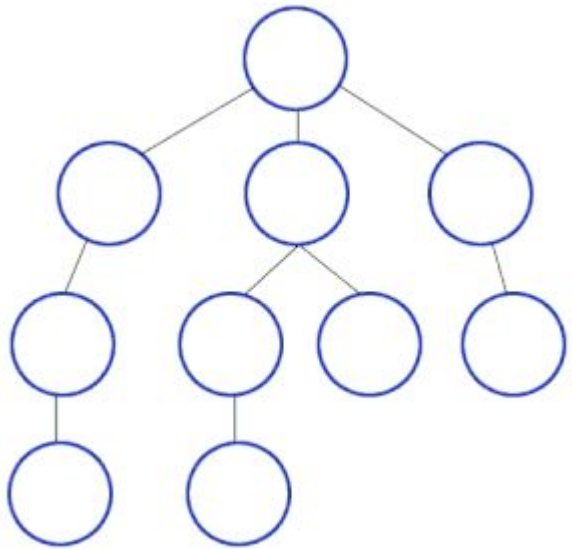
```
$ ./depth
Depth: 2
```

Ποια η πολυπλοκότητα για ένα τέλειο δυαδικό δέντρο;

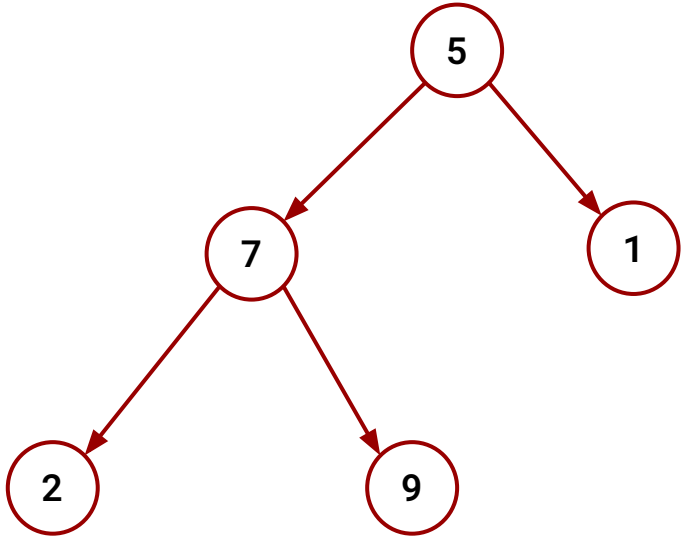
Χρόνος: O(n)
Χώρος: O(logn)

Αναζήτηση κατά Βάθος (Depth-First Search or DFS)

Η αναζήτηση κατά βάθος (depth-first search) είναι ένας αλγόριθμος διάσχισης/αναζήτησης σε δέντρα και γράφους. Ο αλγόριθμος ξεκινά από τον αρχικό κόμβο και εξερευνά όσο περισσότερο μπορεί προτού οπισθοδρομήσει (backtracking).



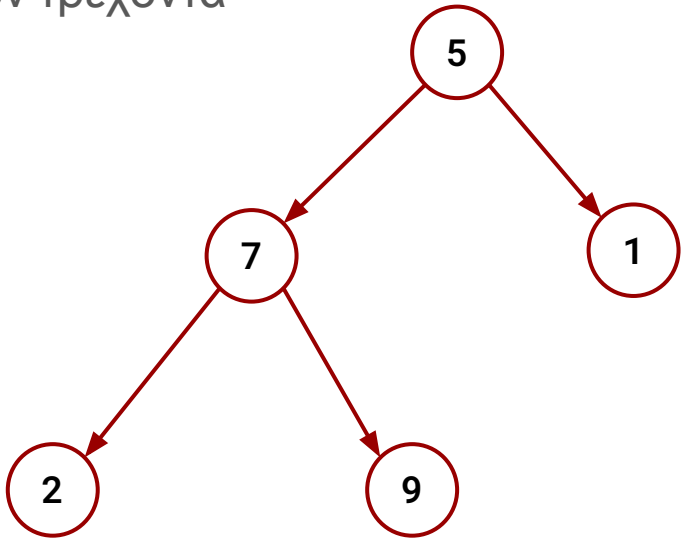
print: Τύπωμα & Διάσχιση (Traversal) Δέντρου



print: Τύπωμα & Διάσχιση (Traversal) Δέντρου

Pre-order Traversal: Τύπωσε (επεξεργάσου) πρώτα τον τρέχοντα κόμβο και μετά τα παιδιά

```
void print(Tree t) {
  if (t == NULL) return;
  printf("%d ", t->value);
  print(t->left);
  print(t->right);
}
```

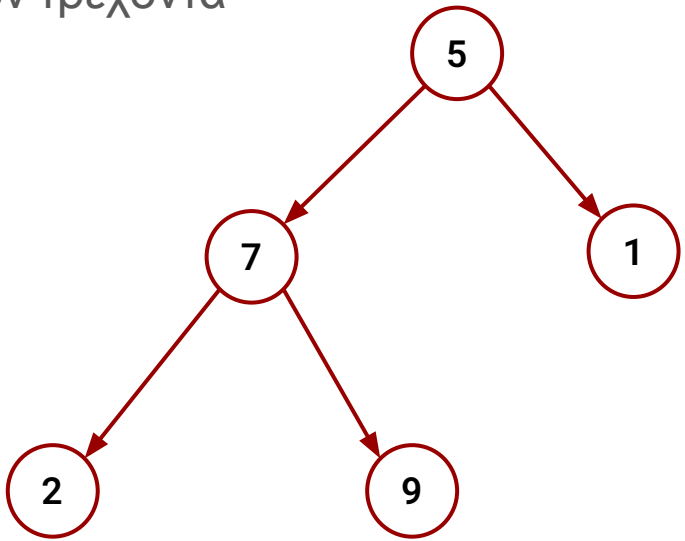


print: Τύπωμα & Διάσχιση (Traversal) Δέντρου

Pre-order Traversal: Τύπωσε (επεξεργάσου) πρώτα τον τρέχοντα κόμβο και μετά τα παιδιά

```
void print(Tree t) {
  if (t == NULL) return;
  printf("%d ", t->value);
  print(t->left);
  print(t->right);
}
```

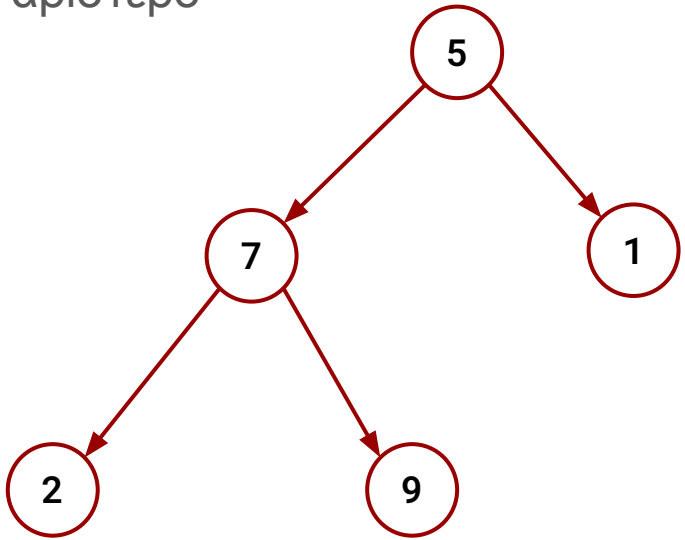
```
$ ./preorder
5 7 2 9 1
```



print: Τύπωμα & Διάσχιση (Traversal) Δέντρου

In-order Traversal: Τύπωσε (επεξεργάσου) πρώτα τον αριστερό κόμβο, μετά τον τρέχοντα και τέλος τον δεξιό

```
void print(Tree t) {
  if (t == NULL) return;
  print(t->left);
  printf("%d ", t->value);
  print(t->right);
}
```

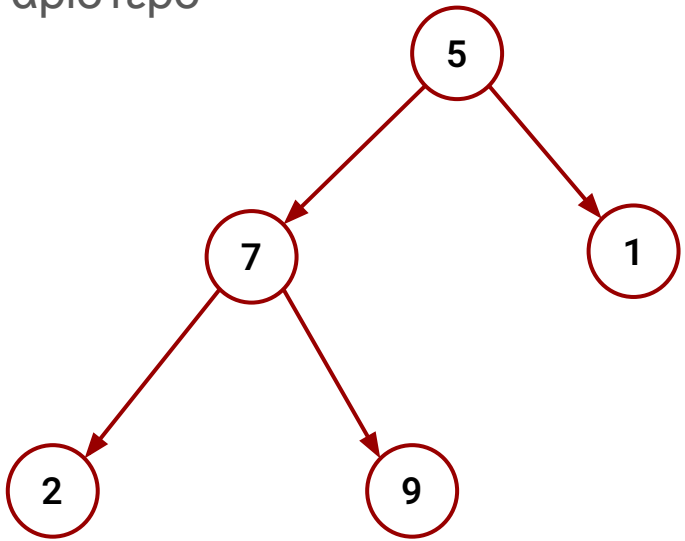


print: Τύπωμα & Διάσχιση (Traversal) Δέντρου

In-order Traversal: Τύπωσε (επεξεργάσου) πρώτα τον αριστερό κόμβο, μετά τον τρέχοντα και τέλος τον δεξιό

```
void print(Tree t) {
  if (t == NULL) return;
  print(t->left);
  printf("%d ", t->value);
  print(t->right);
}
```

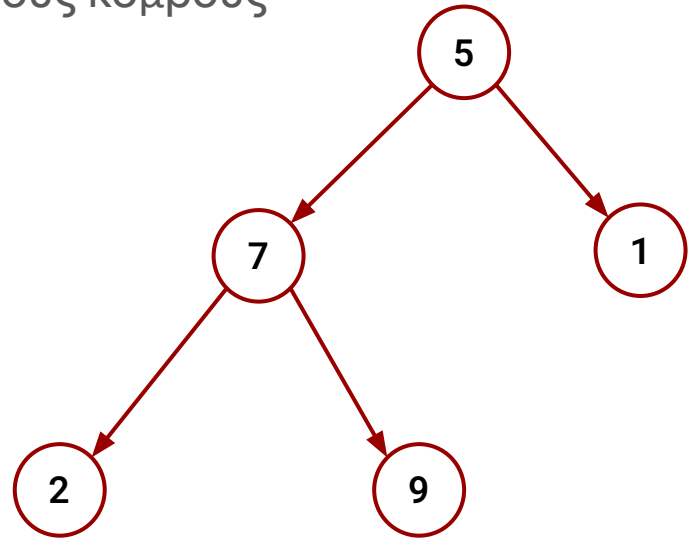
```
$ ./inorder
2 7 9 5 1
```



print: Τύπωμα & Διάσχιση (Traversal) Δέντρου

Post-order Traversal: Τύπωσε (επεξεργάσου) πρώτα τους κόμβους παιδιά και στο τέλος τον τρέχοντα κόμβο

```
void print(Tree t) {
  if (t == NULL) return;
  print(t->left);
  print(t->right);
  printf("%d ", t->value);
}
```

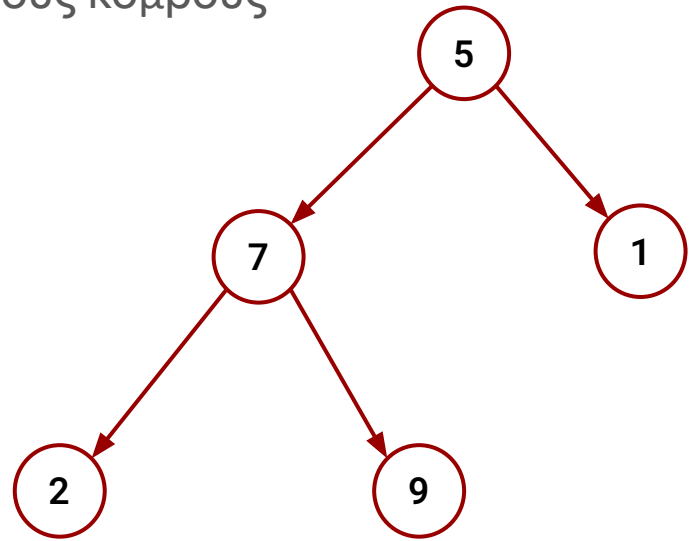


print: Τύπωμα & Διάσχιση (Traversal) Δέντρου

Post-order Traversal: Τύπωσε (επεξεργάσου) πρώτα τους κόμβους παιδιά και στο τέλος τον τρέχοντα κόμβο

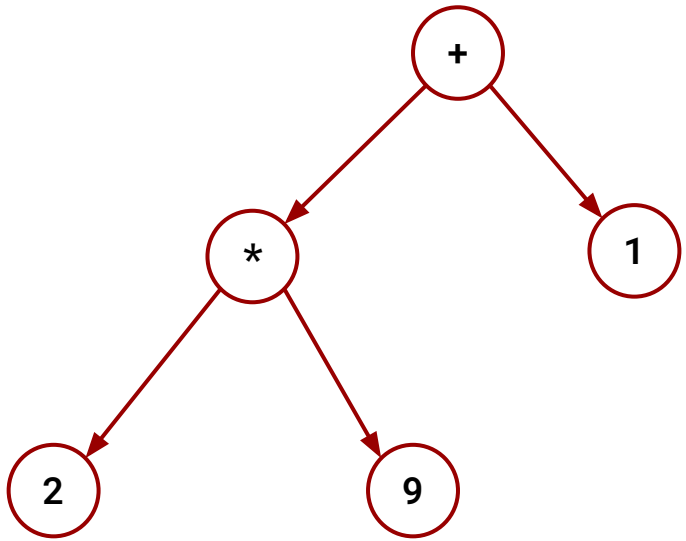
```
void print(Tree t) {
  if (t == NULL) return;
  print(t->left);
  print(t->right);
  printf("%d ", t->value);
}
```

```
$ ./postorder
2 9 7 1 5
```



Χρόνος: O(n)
Χώρος: O(logn)

Θέλω να γράψω έναν αποτιμητή εκφράσεων (calculator / evaluator / interpreter). Ποια διάσχιση θα χρησιμοποιήσω;



find: Αναζήτηση Στοιχείου σε Δυαδικό Δέντρο

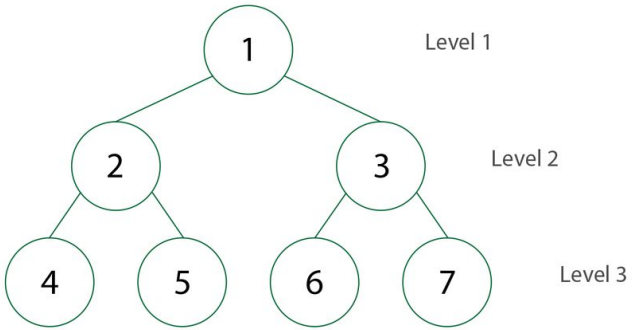
find: Αναζήτηση Στοιχείου σε Δυαδικό Δέντρο

```
Tree find(Tree t, int value) {
  if (t == NULL) return NULL;
  if (t->value == value) return t;
  Tree left = find(t->left, value);
  if (left != NULL) return left;
  return find(t->right, value);
}
```

Χρόνος: O(n)
Χώρος: O(logn)

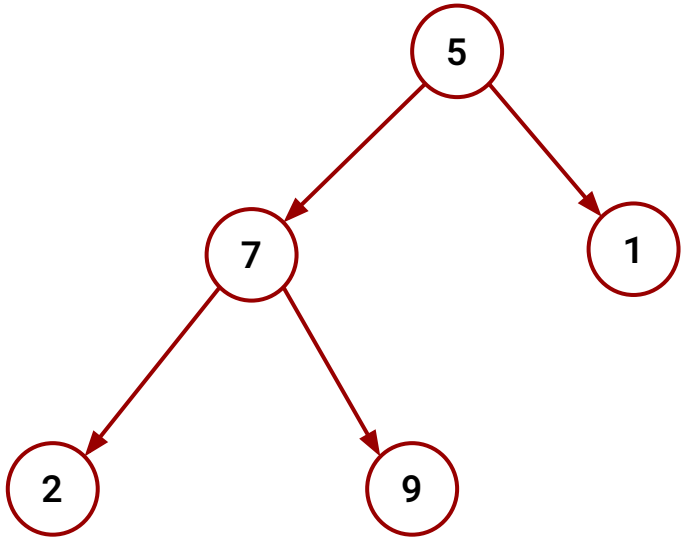
Αναζήτηση κατά Πλάτος (Breadth-First Search or BFS)

Η αναζήτηση κατά πλάτος ή κατά επίπεδα (breadth-first search - BFS) είναι ένας αλγόριθμος διάσχισης/αναζήτησης σε δέντρα και γράφους. Ο αλγόριθμος ξεκινά από τον αρχικό κόμβο και σε κάθε βήμα εξερευνά όλους τους κόμβους στο τρέχον επίπεδο προτού περάσει στο επόμενο.



Τύπωμα με Διάσχιση κατά Πλάτος - BFS

```
void bfs(Tree t) {
  List worklist = NULL;
  Tree tmp;
  insert(&worklist, t);
  while(worklist) {
    tmp = pop_last(&worklist);
    printf("%d ", tmp->value);
    if (tmp->left) insert(&worklist, tmp->left);
    if (tmp->right) insert(&worklist, tmp->right);
  }
}
```



Σημείωση: Ο τύπος List παραπάνω δεν περιέχει πλέον ακεραίους. Τι περιέχει; Πως θα έπρεπε να αλλάξει προκειμένου να υποστηρίξει μια τέτοια υλοποίηση; Τι θα κάνατε ώστε οι λίστες σας να δουλεύουν με κάθε τύπου δεδομένα;

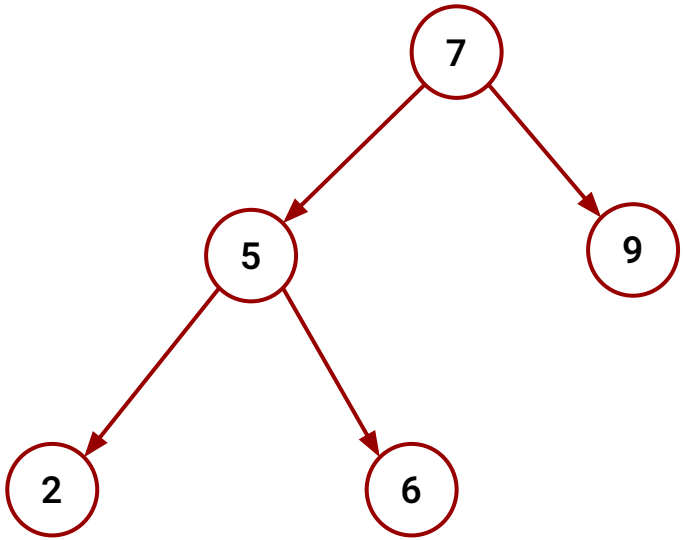
Τύπωμα με Διάσχιση κατά Πλάτος - BFS

```
void bfs(Tree t) {
  List frontier = NULL;
  Tree tmp;
  insert(&frontier, t);
  while(frontier) {
    tmp = pop_last(&frontier);
    printf("%d ", tmp->value);
    if (tmp->left) insert(&frontier, tmp->left);
    if (tmp->right) insert(&frontier, tmp->right);
  }
}
```

Χρόνος: O(n)
Χώρος: O(n)

Δυαδικό Δέντρο Αναζήτησης (Binary Search Tree - BST)

Ένα Δυαδικό Δέντρο Αναζήτησης (Binary Search Tree - BST ή Ordered Tree) είναι ένα ταξινομημένο δέντρο έτσι ώστε για κάθε κόμβο, όλοι οι κόμβοι στο αριστερό του υποδέντρο να είναι μικρότεροι σε τιμή και όλοι οι κόμβοι στο δεξί του υποδέντρο να είναι μεγαλύτεροι.



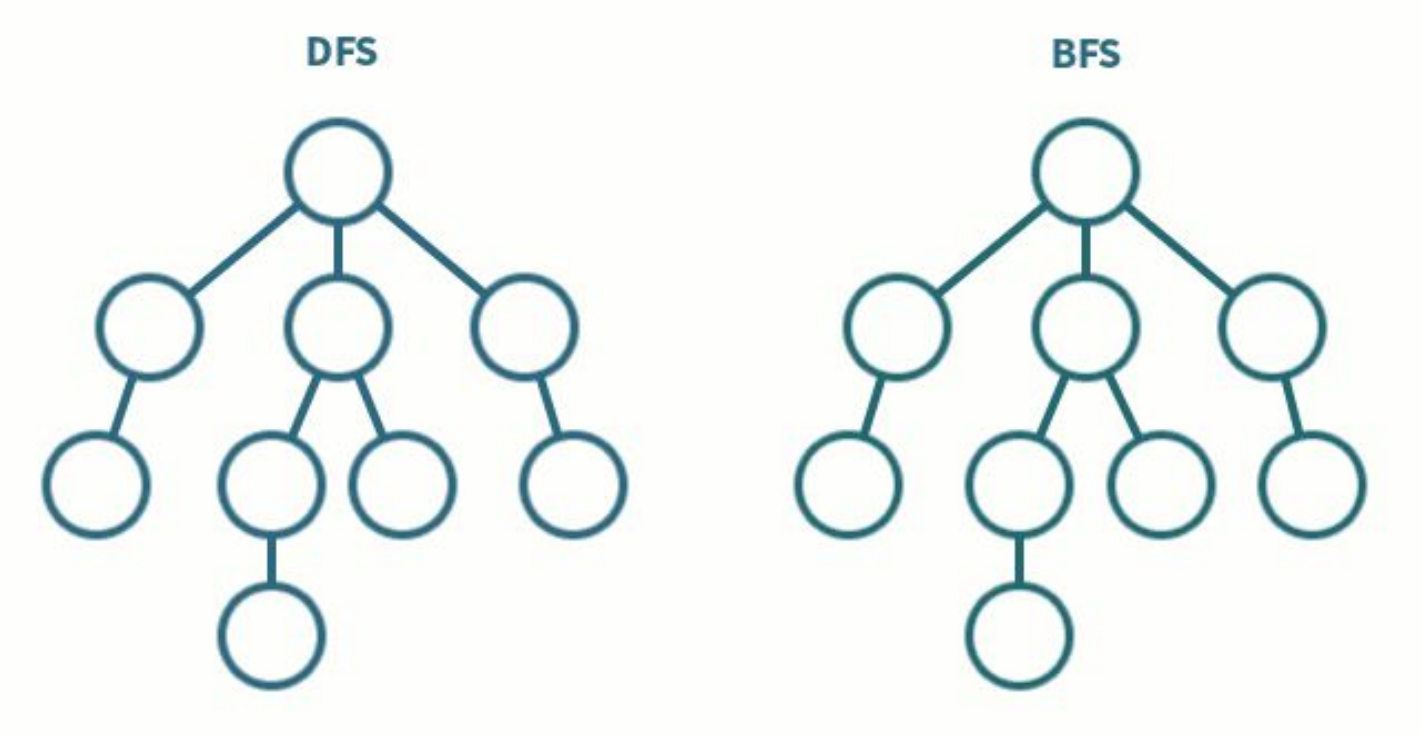
Έλεγχος αν υπάρχει (exists) ένα στοιχείο σε δυαδικό δέντρο. Πως;

Έλεγχος αν υπάρχει (exists) ένα στοιχείο σε δυαδικό δέντρο. Πως;

```
int exists(Tree t, int value) {
  if (t == NULL) return 0;
  if (t->value == value) return 1;
  if (value < t->value) return exists(t->left, value);
  return exists(t->right, value);
}
```

Χρόνος: O(logn)
Χώρος: O(logn)

Ποιος αλγόριθμος αναζήτησης είναι καλύτερος;



Έχεις να αποθηκεύσεις 1 PetaByte (10^6 GB) δεδομένων για μια υπηρεσία (service). Πως θα αποθηκεύσεις τα δεδομένα σου; Πως θα κάνεις αναζήτηση;

Γράφεις ένα πρόγραμμα που βρίσκει λύσεις σε δέντρα-λαβυρίνθους. Αναζητάς το συντομότερο μονοπάτι για την έξοδο. BFS ή DFS; Γιατί;

Θέλεις να βρεις το μέγιστο στοιχείο ενός δέντρου. Προτιμάς BFS ή DFS; Γιατί;

Για την επόμενη φορά

Από τις διαφάνειες του κ. Σταματόπουλου προτείνω να διαβάσετε τις σελίδες 120-135

- Binary Tree Problems
- Tree traversal
- Depth-first search
- Breadth-first search
- Binary Search Tree and Visualization
- Αφηρημένοι Τύποι Δεδομένων (ΑΤΔ) - Abstract Data Types

Ευχαριστώ και καλή μέρα εύχομαι!
Keep Coding ;)