Διάλεξη 21 - Λίστες και Δέντρα

Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

Εισαγωγή στον Προγραμματισμό

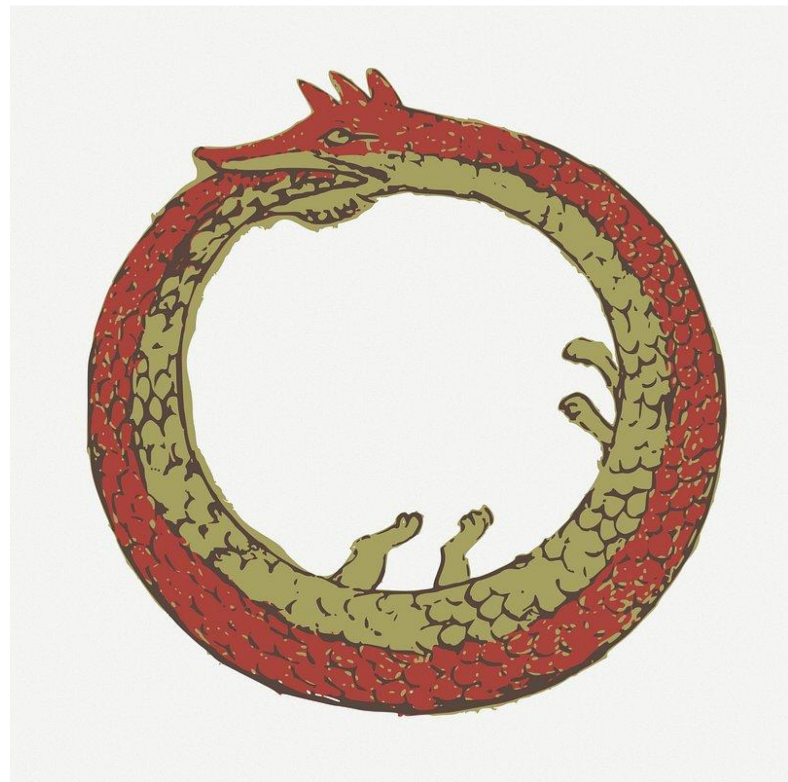
Θανάσης Αυγερινός

Ανακοινώσεις / Διευκρινίσεις

- Προθεσμία για Εργασία #1 αύριο!
- Η Εργασία #2 θα βγει μέσα στην εβδομάδα
- Η Εργασία #3 θα ανακοινωθεί μέσα στις γιορτές

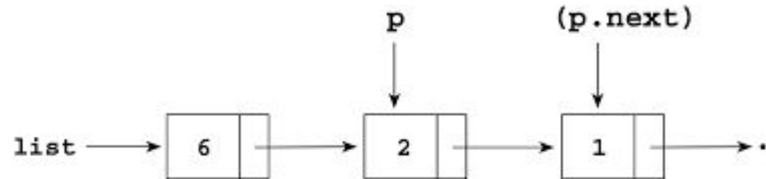
Την προηγούμενη φορά

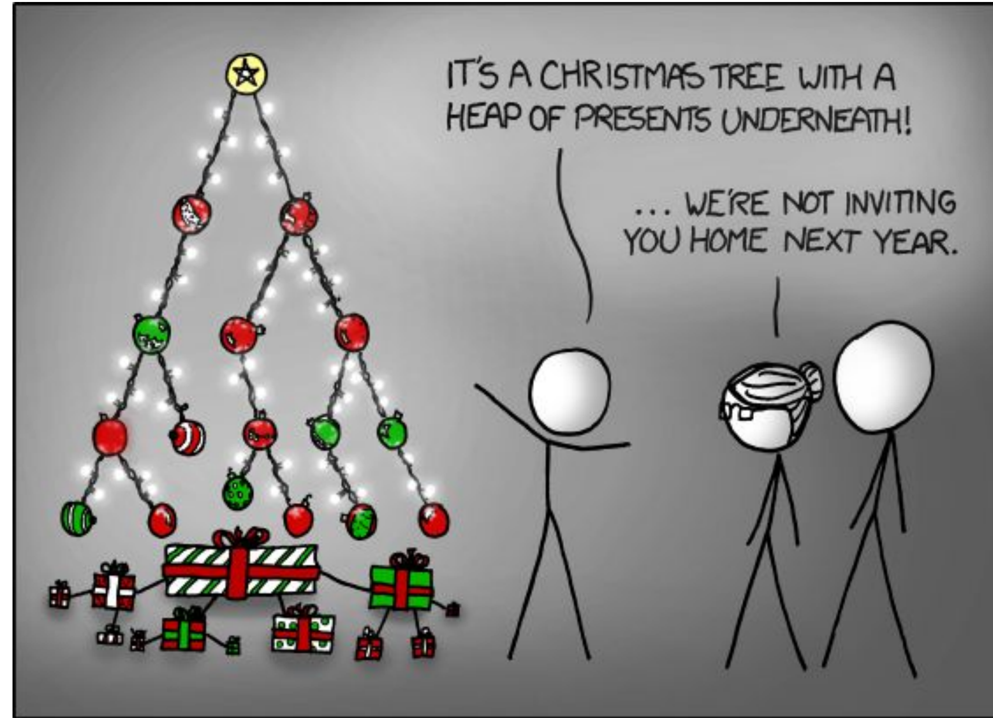
- Προχωρημένες Δομές
  - Πεδία Δυφίων (Bit Fields)
  - Ενώσεις (Unions)
  - Απαριθμήσεις (Enumerations)
  - Αυτοαναφορικές (Self-Referential)



Σήμερα

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





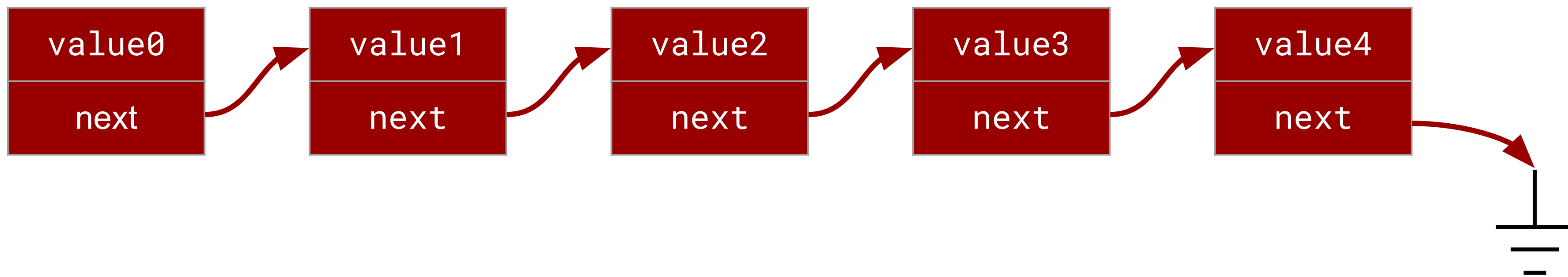
Απλά Συνδεδεμένη Λίστα (Single Linked List)

Στην γενική μορφή δηλώνεται ως:

```
struct listnode {
  int value;
  struct listnode * next;
};
```

Απλά Συνδεδεμένη Λίστα (Single Linked List)

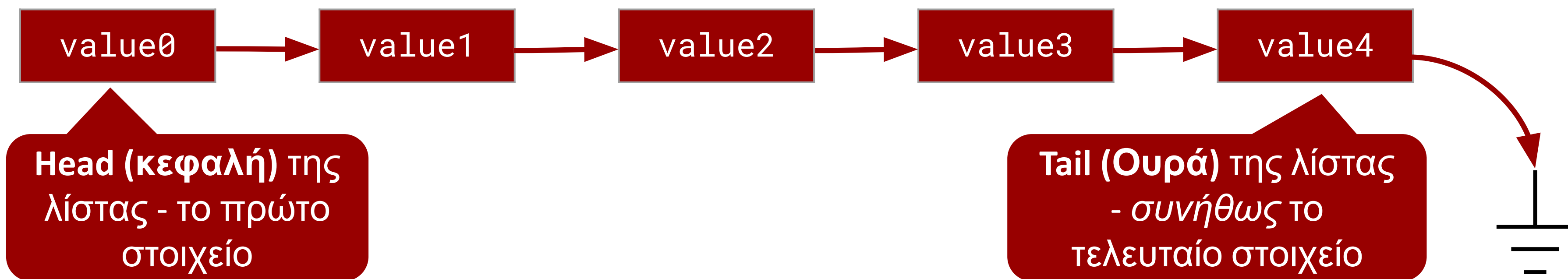
Η **απλά συνδεδεμένη λίστα (single linked list)** είναι ένας τύπος δεδομένων που το κάθε στοιχείο δείχνει (links) στο επόμενο και το τελευταίο δείχνει στο NULL.



Μήκος λίστας (list length): ο αριθμός των στοιχείων που περιέχει (5 παραπάνω)

Απλά Συνδεδεμένη Λίστα (Single Linked List)

Η απλά συνδεδεμένη λίστα (single linked list) είναι ένας τύπος δεδομένων που το κάθε στοιχείο δείχνει (links) στο επόμενο και το τελευταίο δείχνει στο NULL.



Μήκος λίστας (list length): ο αριθμός των στοιχείων που περιέχει (5 παραπάνω)

Βασικές Λειτουργίες με Λίστες

1. is_empty: Έλεγχος αν η λίστα είναι άδεια
2. insert: Προσθήκη στοιχείου στην λίστα
3. print: Τύπωμα στοιχείων λίστας
4. length: Εύρεση μήκους λίστας
5. find: Εύρεση στοιχείου σε λίστα
6. delete: Αφαίρεση στοιχείου από λίστα

is_empty: Έλεγχος αν η λίστα είναι άδεια

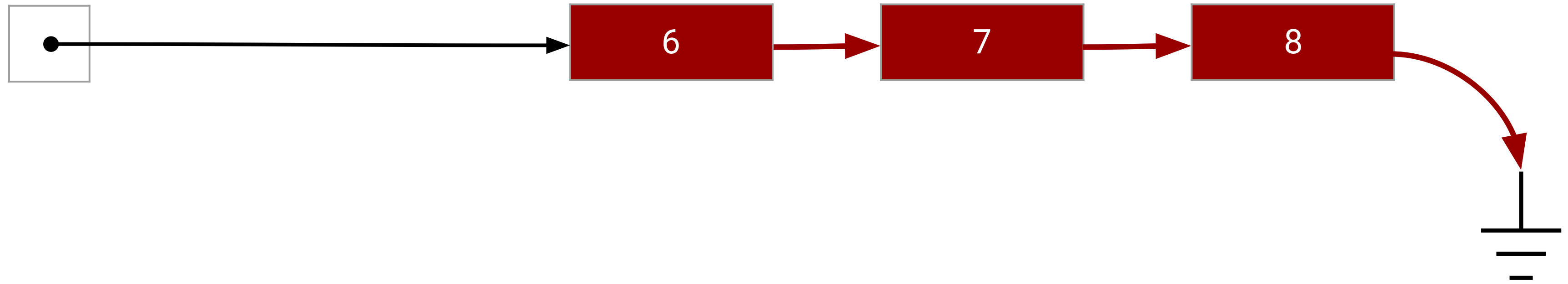
```c
#include <stdio.h>

typedef struct listnode {int value; struct listnode * next;} * List;

int is_empty(List list) {

  return list == NULL;

}

int main() {

  struct listnode node = {42, NULL};

  List list1 = &node;

  List list2 = NULL;

  printf("Is empty: %d\n", is_empty(list1));

  printf("Is empty: %d\n", is_empty(list2));

  return 0;

}
```

is_empty: Έλεγχος αν η λίστα είναι άδεια

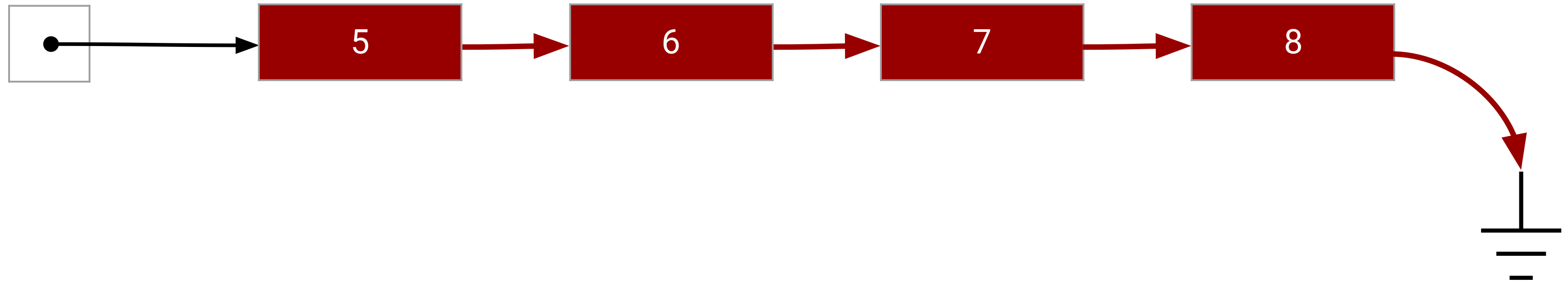
```c
#include <stdio.h>
typedef struct listnode {int value; struct listnode * next;} * List;
int is_empty(List list) {
  return list == NULL;
}
int main() {
  struct listnode node = {42, NULL};
  List list1 = &node;
  List list2 = NULL;
  printf("Is empty: %d\n", is_empty(list1));
  printf("Is empty: %d\n", is_empty(list2));
  return 0;
}
```

```
$ ./list
Is empty: 0
Is empty: 1
```

Θέλω να προσθέσω ένα στοιχείο (π.χ., το 5) σε λίστα. Πως;



Θέλω να προσθέσω ένα στοιχείο (π.χ., το 5) σε λίστα. Πως;



insert: Προσθήκη στοιχείου στην λίστα

```
#include <stdio.h>

#include <stdlib.h>

typedef struct listnode {int value; struct listnode * next;} * List;

void insert(List * list, int value) {

  List current_head = *list;

  List new_head = malloc(sizeof(struct listnode));

  new_head->value = value;

  new_head->next = current_head;

  *list = new_head;

}

int main() {

  List list = NULL;

  insert(&list, 42); insert(&list, 43); insert(&list, 44);

  print(list);

  return 0;

}
```

Δημιουργία νέου κόμβου λίστας στον σωρό (heap)

Αρχικοποίηση κόμβου

Ο νέος κόμβος γίνεται η νέα κεφαλή της λίστας

insert: Προσθήκη στοιχείου στην λίστα

```
#include <stdio.h>

#include <stdlib.h>

typedef struct listnode {int value; struct listnode * next;} * List;

void insert(List * list, int value) {

  List current_head = *list;

  List new_head = malloc(sizeof(struct listnode));

  new_head->value = value;

  new_head->next = current_head;

  *list = new_head;

}

int main() {

  List list = NULL;

  insert(&list, 42); insert(&list, 43); insert(&list, 44);

  print(list); // commented out for size

  return 0;

}
```

Τι θα τυπώσει το πρόγραμμα;

insert: Προσθήκη στοιχείου στην λίστα

```
#include <stdio.h>

#include <stdlib.h>

typedef struct listnode {int value; struct listnode * next;} * List;

void insert(List * list, int value) {

  List current_head = *list;

  List new_head = malloc(sizeof(struct listnode));

  new_head->value = value;

  new_head->next = current_head;

  *list = new_head;

}

int main() {

  List list = NULL;

  insert(&list, 42); insert(&list, 43); insert(&list, 44);

  print(list); // commented out for size

  return 0;

}
```

Τι θα τυπώσει το πρόγραμμα;

```
$ ./insert
list:  -> 44 -> 43 -> 42 -> NULL
```

print: Τύπωμα στοιχείων λίστας

```c
void print(List list) {
  printf("list: ");
  while(list) {
    printf(" -> %d", list->value);
    list = list->next;
  }
  printf(" -> NULL\n");
}
```

length: Εύρεση μήκους λίστας

length: Εύρεση μήκους λίστας

```
int length(List list) {
  int counter = 0;
  while(list) {
    counter++;
    list = list->next;
  }
  return counter;
}
```

Πως θα το κάναμε αναδρομικά;

length: Εύρεση μήκους λίστας

```
int length(List list) {
  int counter = 0;
  while(list) {
    counter++;
    list = list->next;
  }
  return counter;
}
```

```
int length(List list) {
  if (!list) return 0;
  return 1 + length(list->next);
}
```

Ποια η πολυπλοκότητα των παραπάνω ως προς χρόνο και χώρο;

length: Εύρεση μήκους λίστας

```
int length(List list) {
  int counter = 0;
  while(list) {
   counter++;
   list = list->next;
  }
  return counter;
}
```

Χρόνος: O(n)
Χώρος: O(1)

```
int length(List list) {
  if (!list) return 0;
  return 1 + length(list->next);
}
```

Ποια η πολυπλοκότητα των παραπάνω ως προς χρόνο και χώρο;

Χρόνος: O(n)
Χώρος: O(n)

find: Εύρεση στοιχείου σε λίστα

```
#include <stdio.h>

#include <stdlib.h>

typedef struct listnode {int value; struct listnode * next;} * List;

List find(List list, int value) {

  while(list && list->value != value) {

    list = list->next;

  }

  return list;

}

int main() {

  List list = NULL;

  insert(&list, 42); insert(&list, 43); insert(&list, 44);

  printf("Found 43: %x\n", find(list, 43));

  printf("Found 34: %x\n", find(list, 34));

  return 0;

}
```

Τι θα τυπώσει το πρόγραμμα;

find: Εύρεση στοιχείου σε λίστα

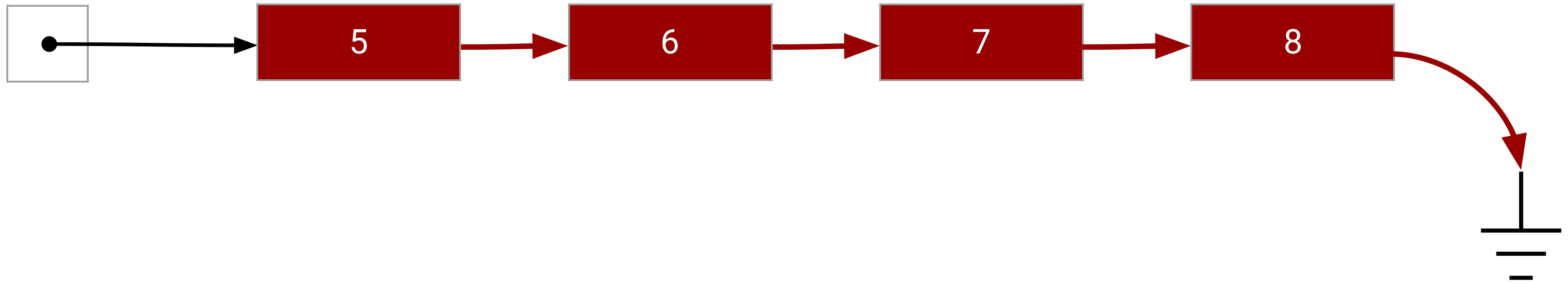
```
#include <stdio.h>

#include <stdlib.h>

typedef struct listnode {int value; struct listnode * next;} * List;

List find(List list, int value) {

  while(list && list->value != value) {

    list = list->next;

  }

  return list;

}

int main() {

  List list = NULL;

  insert(&list, 42); insert(&list, 43); insert(&list, 44);

  printf("Found 43: %x\n", find(list, 43));

  printf("Found 34: %x\n", find(list, 34));

  return 0;

}
```

Τι θα τυπώσει το πρόγραμμα;

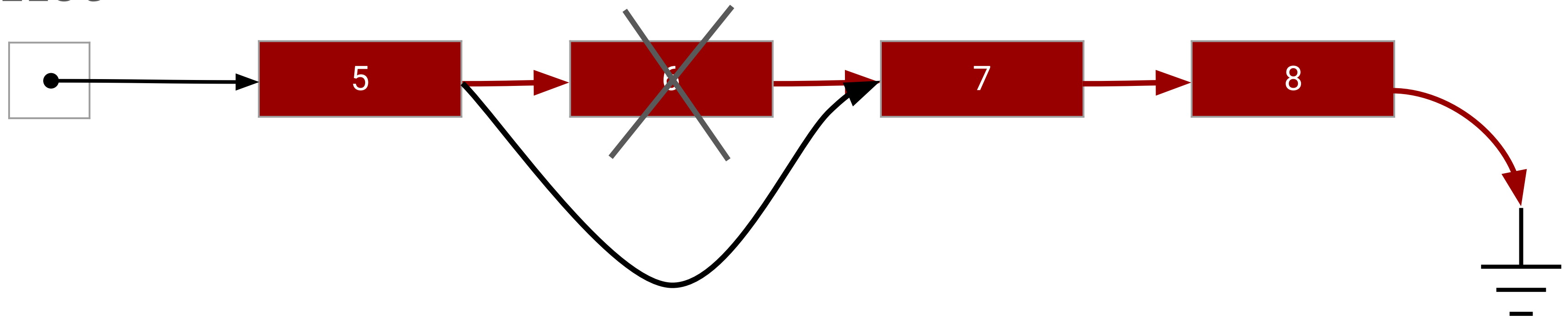
```
$ ./find
Found 43: 161862c0
Found 34: 0
```

Χρόνος: O(n)
Χώρος: O(1)

Θέλω να αφαιρέσω ένα στοιχείο (π.χ., το 6). Πως;



Θέλω να αφαιρέσω ένα στοιχείο (π.χ., το 6). Πως;



delete: Αφαίρεση στοιχείου από λίστα

```
void delete(List * list, int value) {
  List temp;
  while(*list && (*list)->value != value) {
      list = &((*list)->next);
  }
  if (*list) {
      temp = *list;
      *list = temp->next;
      free(temp);
  }
}
```

Πίνακες vs Λίστες

Πίνακες

1. Τα στοιχεία αποθηκεύονται σε **συνεχόμενες θέσεις** στην μνήμη
2. Ο χώρος μνήμης είναι **όσος χρειάζεται** για την αποθήκευση των στοιχείων
3. **Πρόσβαση** σε κάθε στοιχείο **σε σταθερό χρόνο** (O(1)) χρησιμοποιώντας array[i]
4. **Αναδιάταξη** στοιχείων συνήθως παίρνει **O(n) χρόνο** (π.χ., εισαγωγή στο array[0])
5. Στην **δήλωσή** τους χρειάζεται να **ξέρουμε πόσα στοιχεία** θα εισάγουμε (πιο στατικό ως δομή δεδομένων)

Λίστες

1. Τα στοιχεία μπορούν να αποθηκευτούν **σε οποιαδήποτε θέση στην μνήμη**
2. Ο χώρος μνήμης είναι **επαυξημένος κατά ένα sizeof(pointer)** για κάθε στοιχείο
3. **Πρόσβαση** σε κάθε στοιχείο **σε γραμμικό χρόνο** (O(n))
4. Εύκολη / γρήγορη **αναδιάταξη στοιχείων με χρήση δεικτών**
5. Στην **δήλωσή** τους **δεν χρειάζεται να ξέρουμε πόσα στοιχεία** θα εισάγουμε (πιο δυναμικό ως δομή δεδομένων)

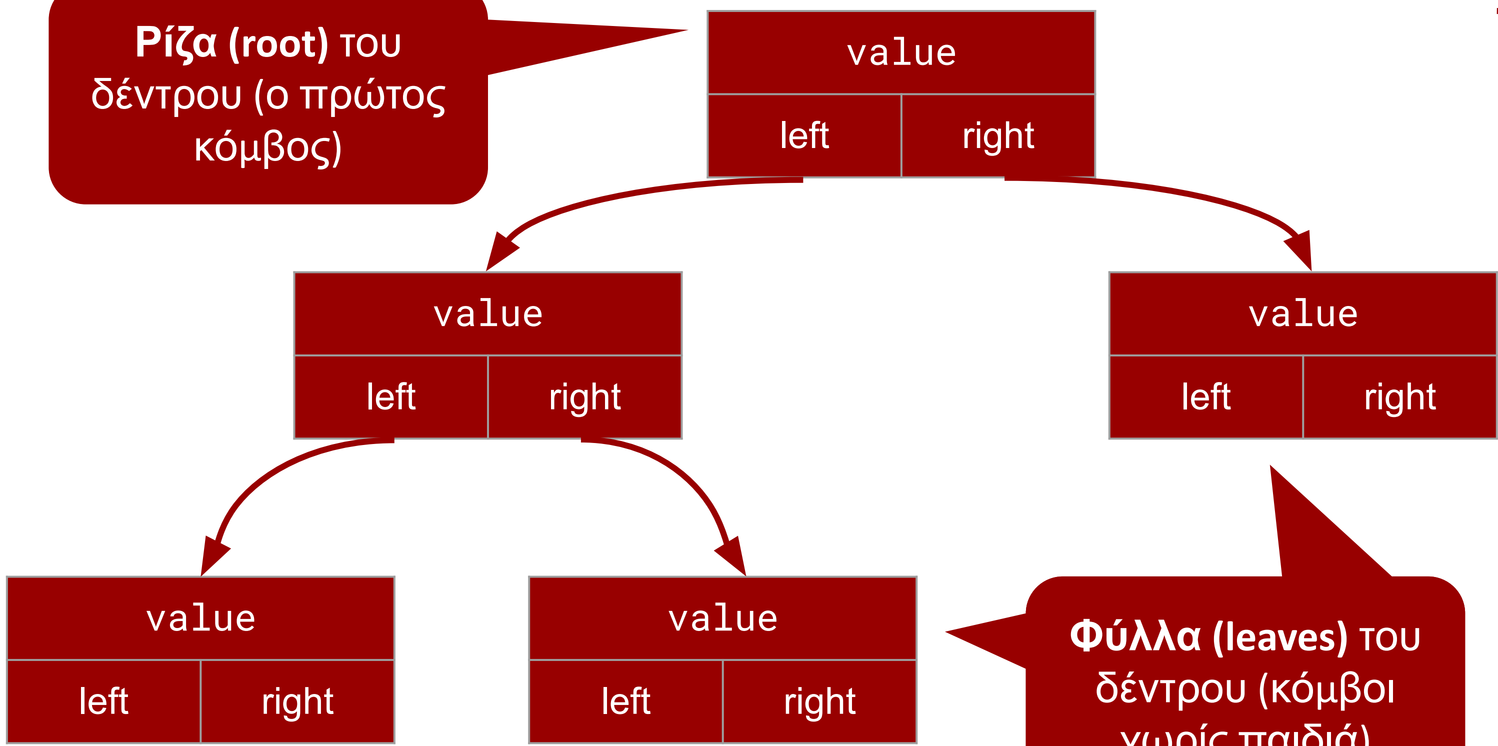
Δυαδικό Δέντρο (Binary Tree)

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;
};
```

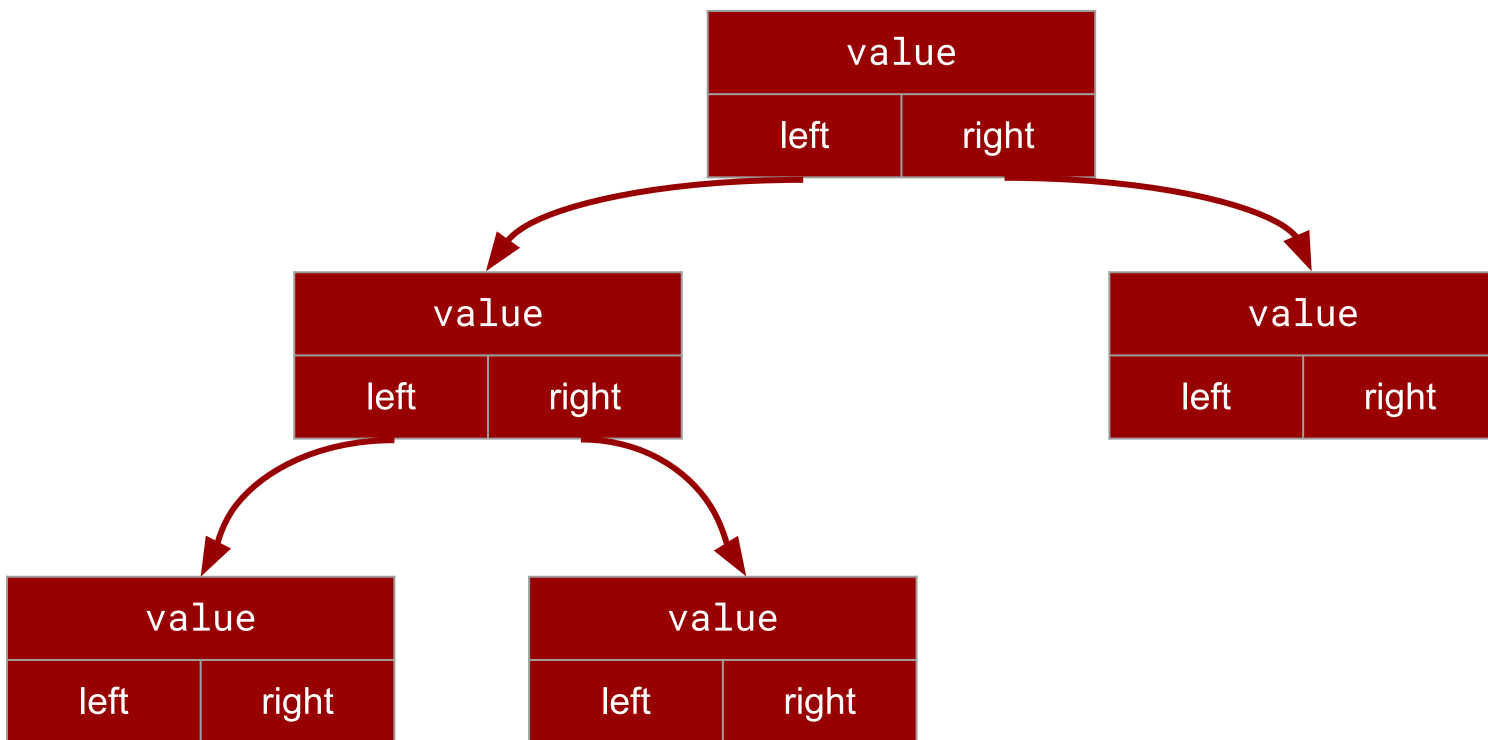
Εφαρμογές: από βάσεις δεδομένων/αναζήτηση μέχρι μεταγλωττιστές και από συμπίεση δεδομένων μέχρι κρυπτογραφία (όπου απαιτείται αναπαράσταση γνώσης)

Δυαδικά Δέντρα (Binary Trees)



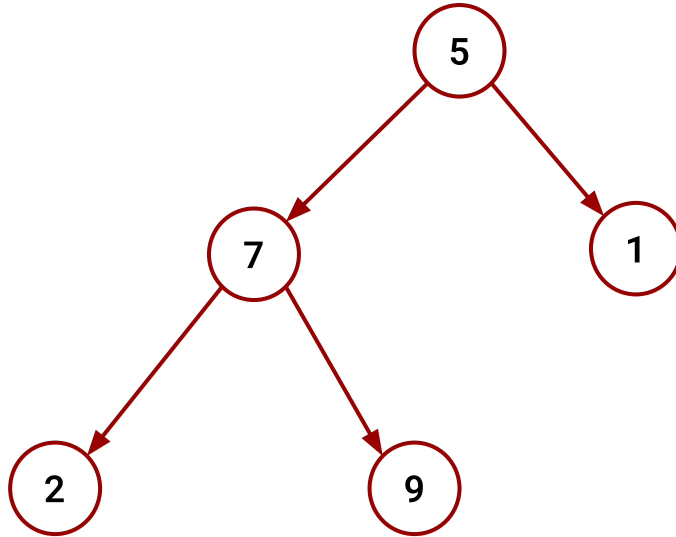
Βάθος (depth) του δέντρου (μέγιστος αριθμός συνδέσμων από την ρίζα μέχρι τα φύλλα) - εδώ 2

Δυαδικά Δέντρα (Binary Trees)



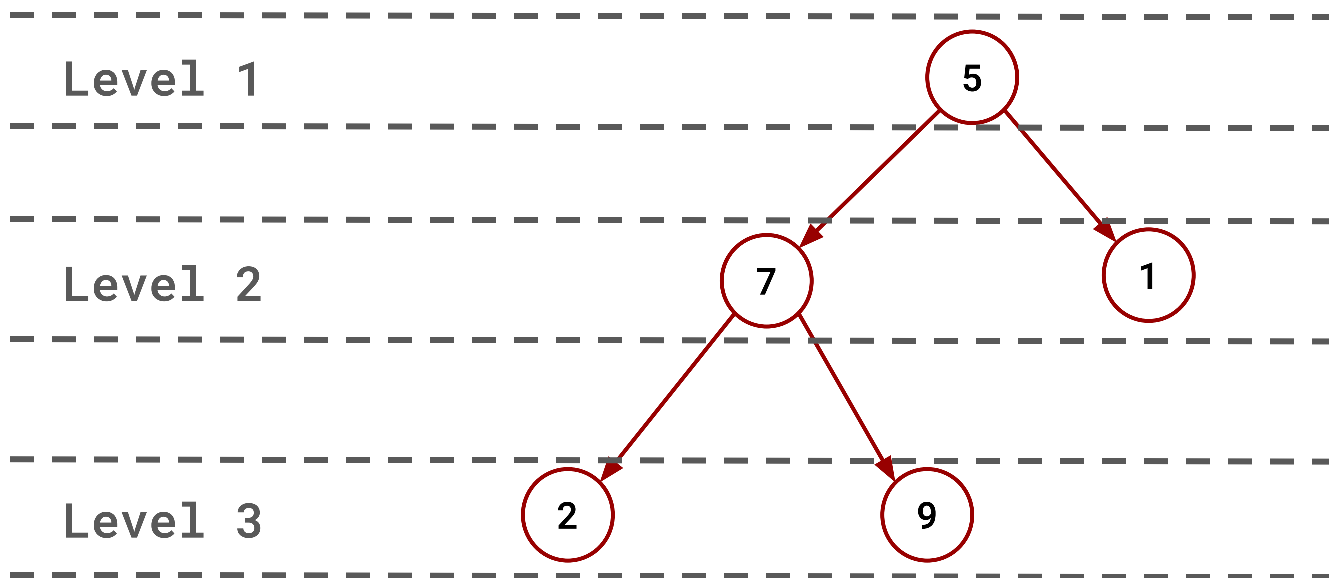
Ύψος (height) του δέντρου (μέγιστος αριθμός συνδέσμων από τα φύλλα μέχρι την ρίζα) - εδώ 2

Δυαδικά Δέντρα (Binary Trees)



Επίπεδο Κόμβου (Node Level)

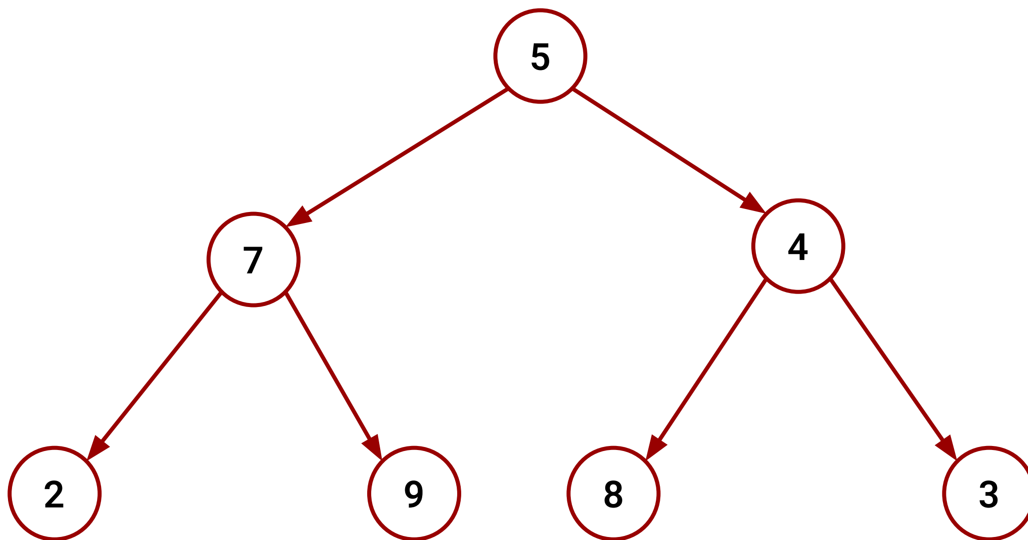
Το επίπεδο ενός κόμβου σε ένα δέντρο ισούται με τον αριθμό των κόμβων που μεσολαβούν μέχρι την ρίζα του δέντρου.



Τύποι Δυαδικών Δέντρων - Τέλειο (Perfect)

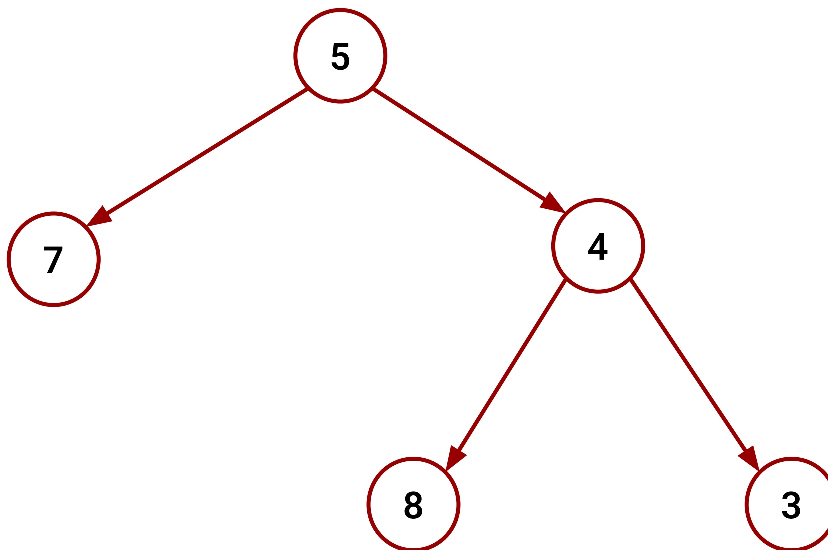
Τέλειο Δυαδικό Δέντρο (Perfect Binary Tree): ένα δυαδικό δέντρο που όλοι οι εσωτερικοί κόμβοι έχουν δύο παιδιά και όλα τα φύλλα βρίσκονται στο ίδιο επίπεδο.

Πόσοι κόμβοι σε n επίπεδα;



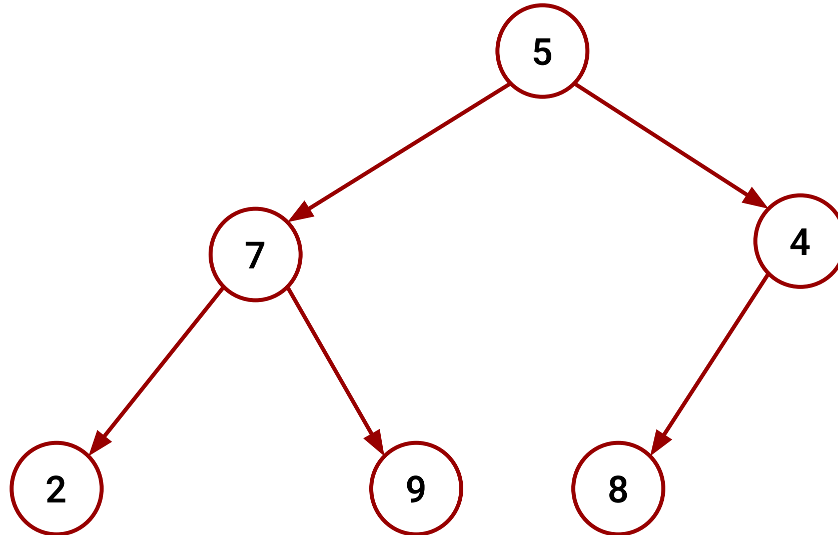
Τύποι Δυαδικών Δέντρων - Γεμάτο (Full)

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



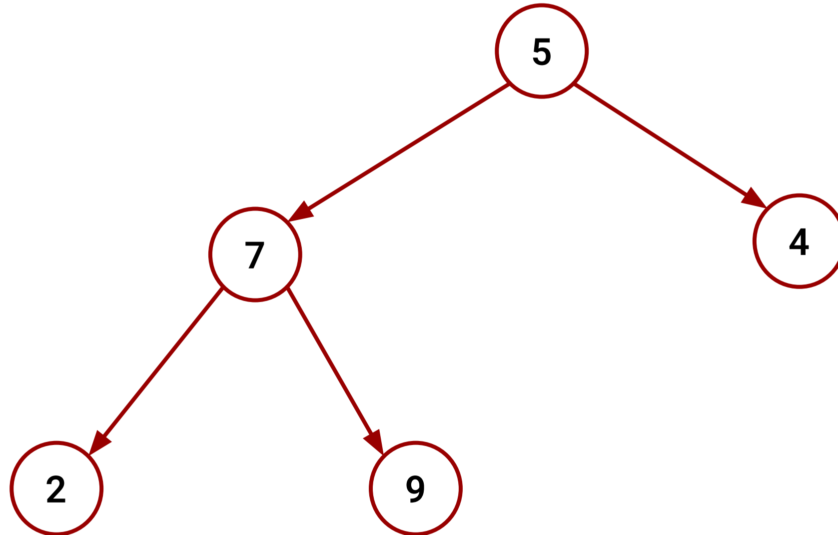
Τύποι Δυαδικών Δέντρων - Πλήρες (Complete)

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.



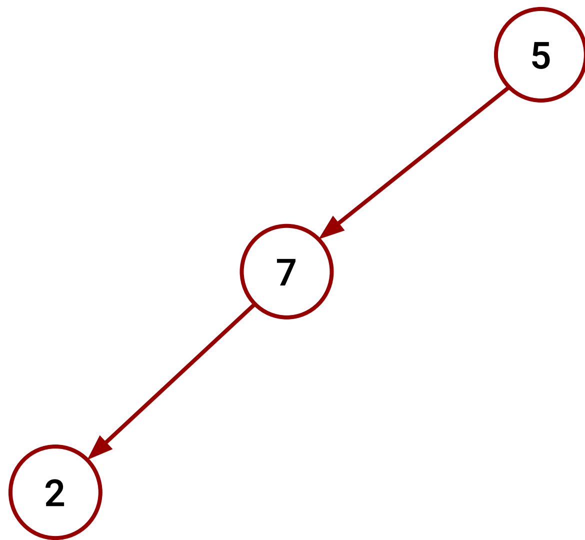
Τύποι Δυαδικών Δέντρων - Ισορροπημένο (Balanced)

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.



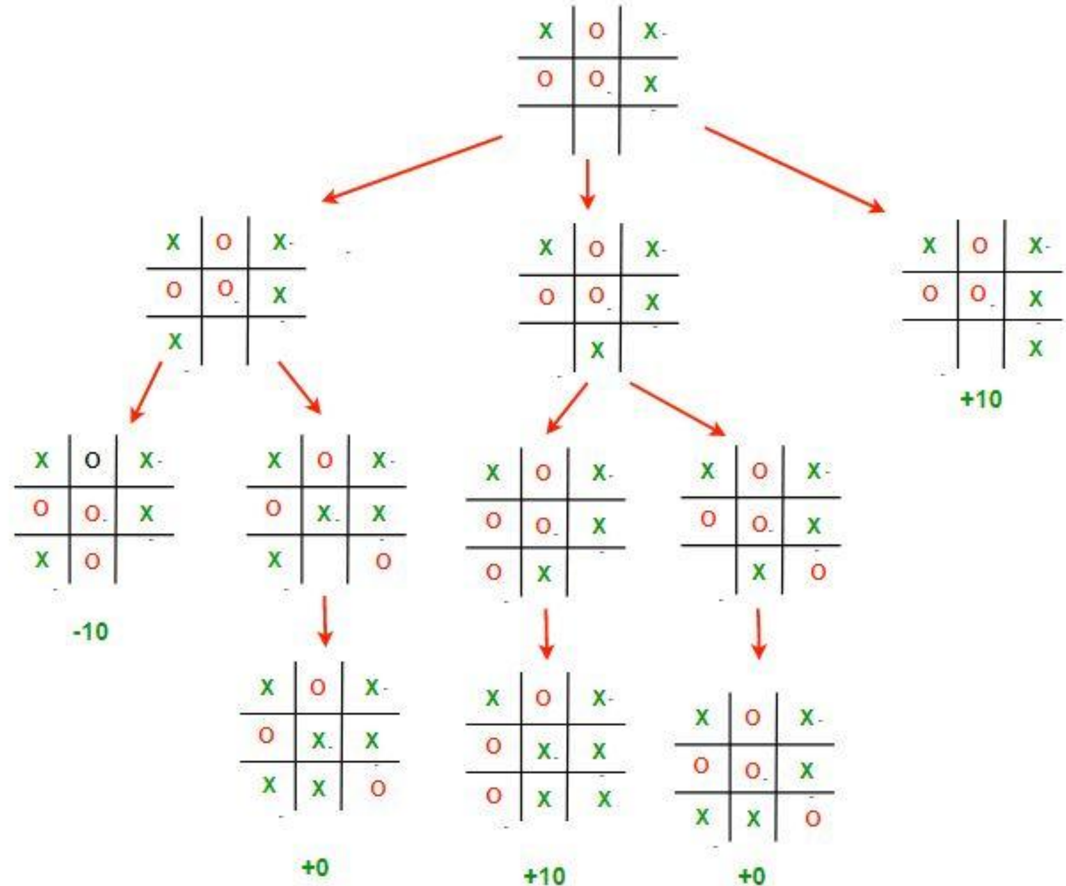
Τύποι Δυαδικών Δέντρων - Εκφυλισμένο (Degenerate)

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.



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

```
#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

```
#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;

}
```

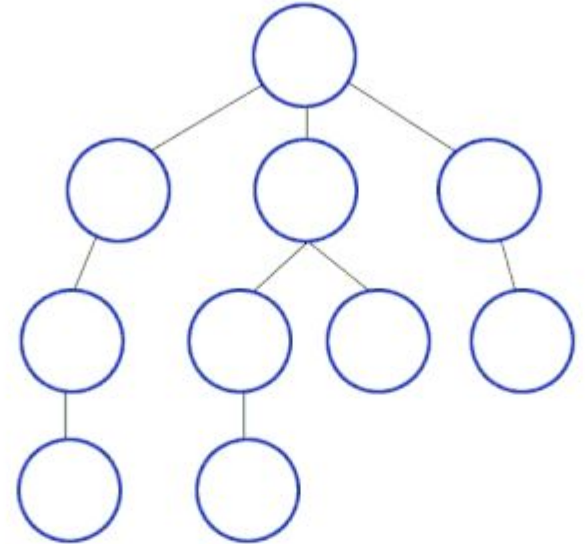
```
$ ./depth
Depth: 2
```

Ποια η πολυπλοκότητα για ένα τέλειο δυαδικό δέντρο;

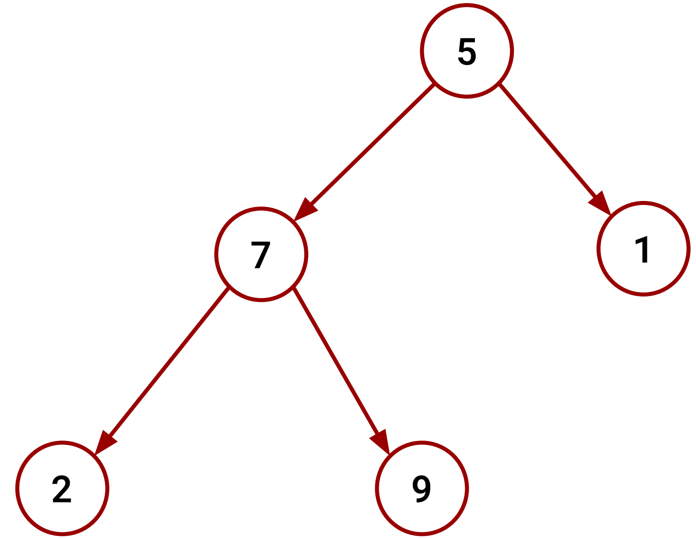
Χρόνος: O(n)
Χώρος: O(logn)

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 οπισθοδρομίσει (backtracking).



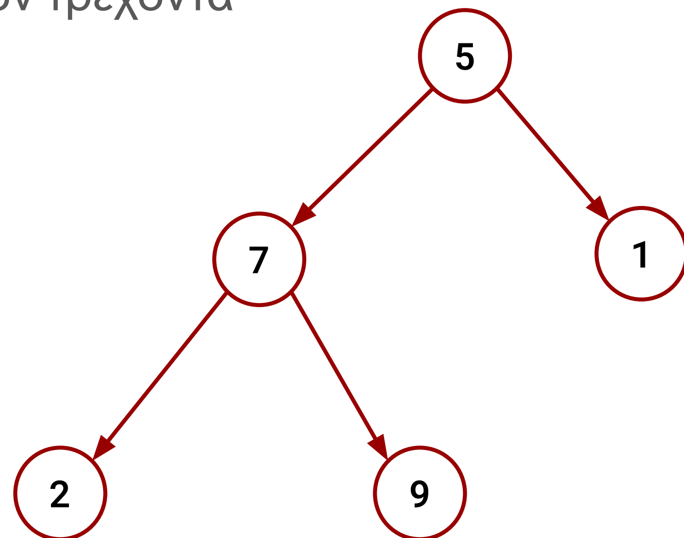
print: Τύπωμα & Διάσχιση (Traversal) Δέντρου



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);
}
```

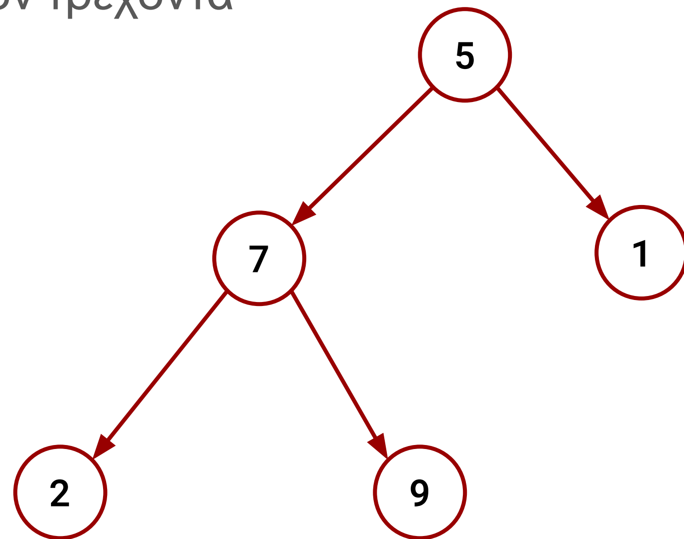


print: Τύπωμα & Διάσχιση (Traversal) Δέντρου

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);
}
```

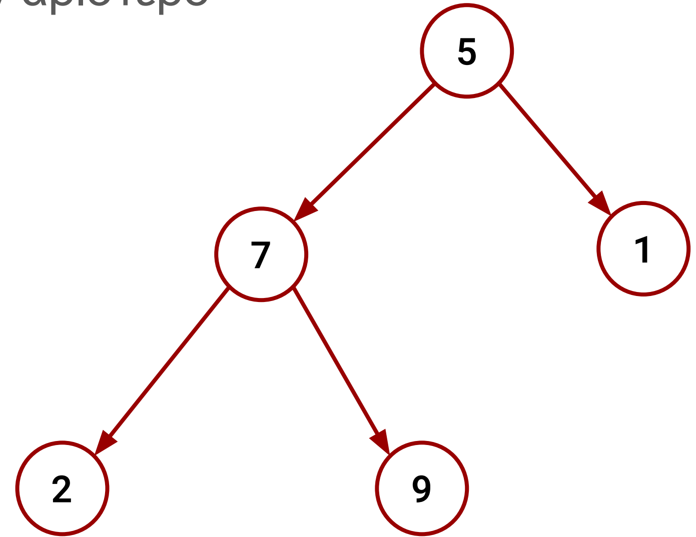
```
$ ./preorder
5 7 2 9 1
```



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);
}
```

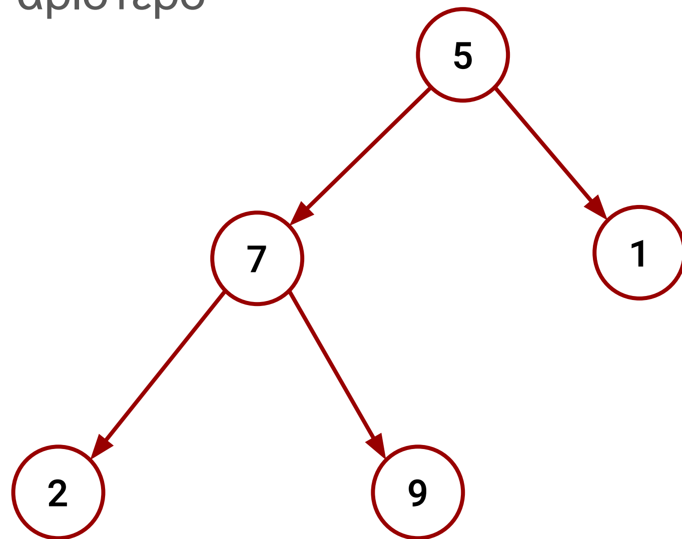


print: Τύπωμα & Διάσχιση (Traversal) Δέντρου

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);
}
```

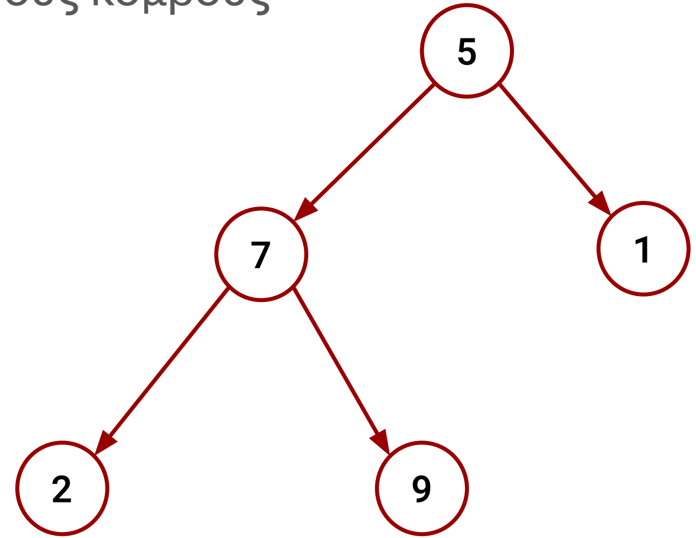
```
$ ./inorder
2 7 9 5 1
```



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);
}
```

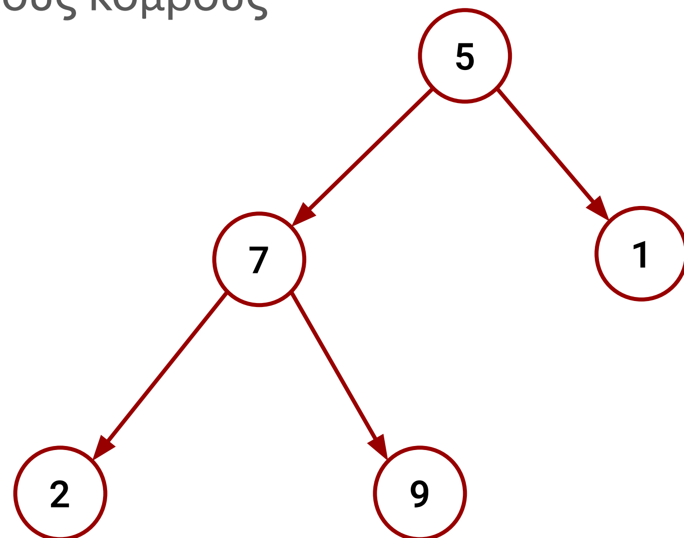


print: Τύπωμα & Διάσχιση (Traversal) Δέντρου

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

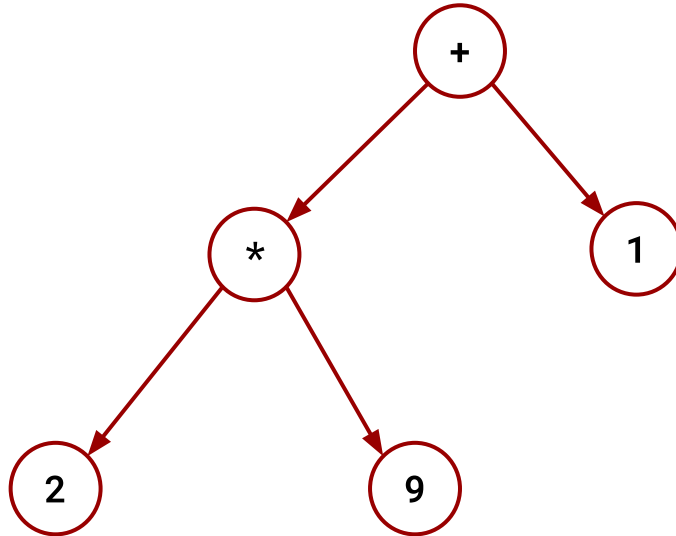
```c
void print(Tree t) {
  if (t == NULL) return;
  print(t->left);
  print(t->right);
  printf("%d ", t->value);
}
```

```
$ ./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

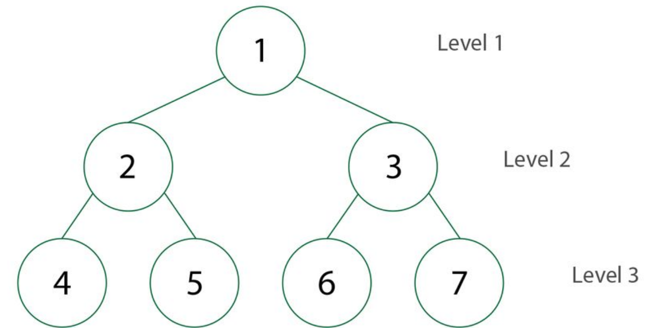
find: Αναζήτηση Στοιχείου σε Δυαδικό Δέντρο

```
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)

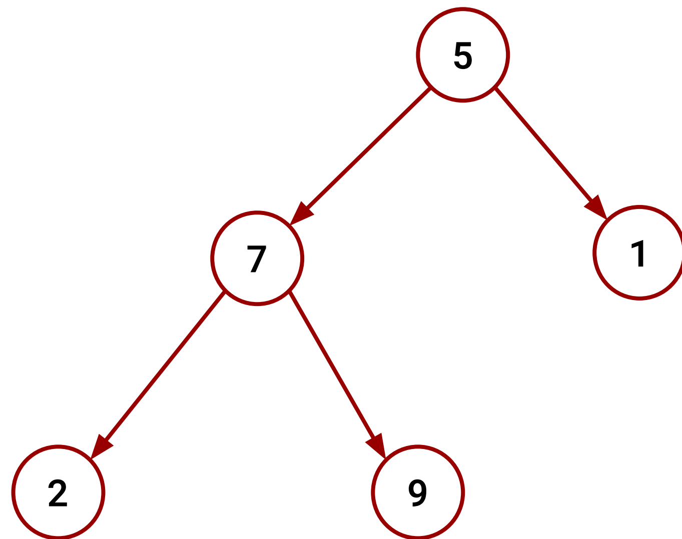
Αναζήτηση κατά Πλάτος (Breadth-First Search or BFS)

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);
  }
}
```



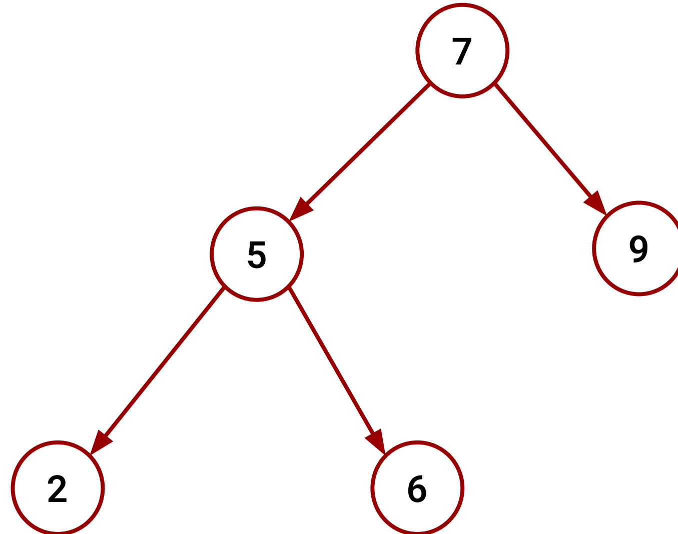
Τύπωμα με Διάσχιση κατά Πλάτος - BFS

```
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 κάθε αριστερός κόμβος να είναι μικρότερος του γονιού του και κάθε δεξιός κόμβος να είναι μεγαλύτερος του.



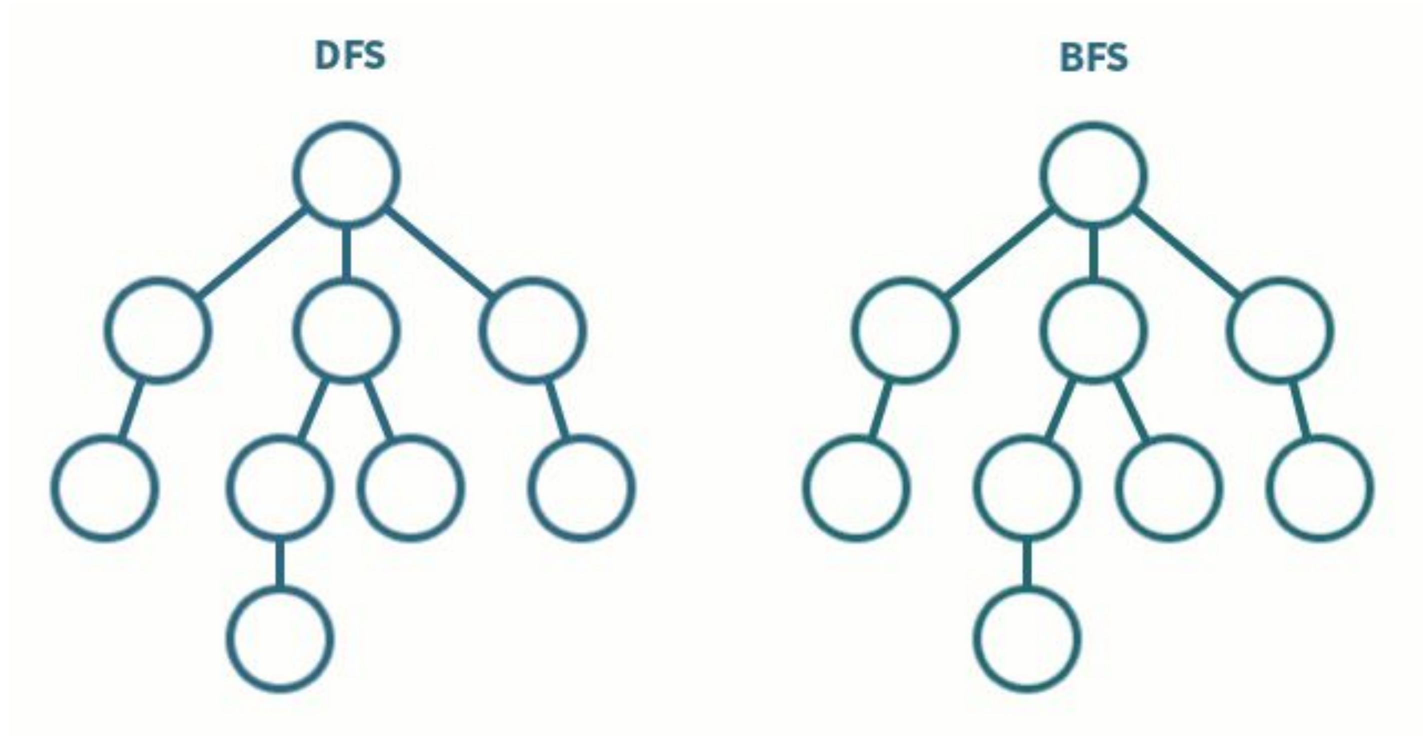
Έλεγχος αν υπάρχει (exists) ένα στοιχείο σε δυαδικό δέντρο. Πως;

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

- Linked List
- 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 ;)