

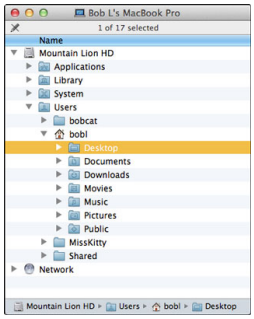

Estruturas de Dados / Programação 2
Árvores

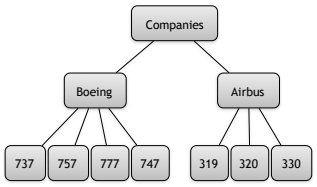
Márcio Ribeiro
marcio@ic.ufal.br
twitter.com/marciomribeiro

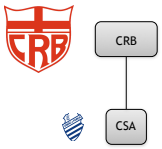
Introduction

- Linear structures
- Removing this idea, treasure of applications

Hierarchical structure







Trees

The structure of Trees

- At most one predecessor but many successors
- Root has no predecessor
- Every position except the root has exactly one predecessor
- Every position except the root has the root as a (perhaps not immediate) predecessor

Definitions

- Let *T* be a tree, and let *n* and *m* be nodes of *T*...

① *n* and *m* are siblings if they have the same parent

② A node is a leaf of *T* if it has no children

③ A node is an internal node if it is not a leaf

④ An edge of *T* connects two nodes (parent and child)

⑤ A path in T is a collection of edges that join two nodes; the length of a path consists of the number of edges in the path

Definitions

- Let *T* be a tree, and let *n* and *m* be nodes of *T*…

⑥ The height of a node is the length of the longest path from the node to a leaf

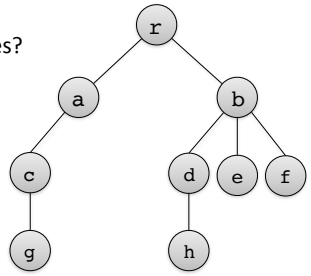
⑦ The depth of a node is the length of the path from that node to the root

⑧ The height of a tree *T* is the height of the root, namely, the length of the longest path from the root to a leaf

⑨ A subtree *S*, of *T* rooted at n is a tree that is made from *T* by considering *n* to be the root of *S* and including all *S* descendants of *n*

Exercise

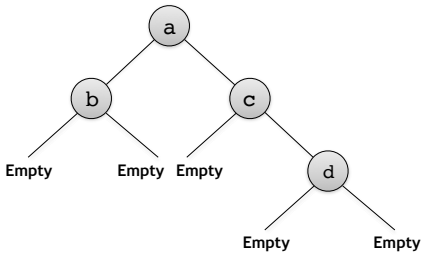
- *a* and *b* are…?
- *h* is a descendant of…?
- What are the leaves?
- What are the internal nodes?
- height(*b*)?
- depth(*b*)?
- height(tree)?
- subtree(*b*)?



Binary Tree

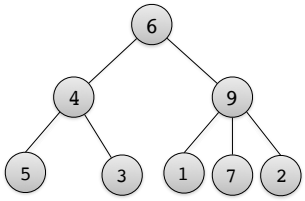
Definition

- A binary tree consists of either:
  - Nothing at all (empty tree) or
  - A node (called root) of the binary tree along with a left subtree and a right subtree, both of which are binary trees



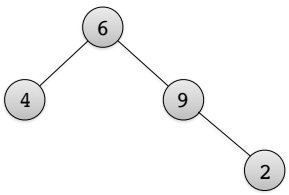
Quiz

- Is the following tree a binary one?



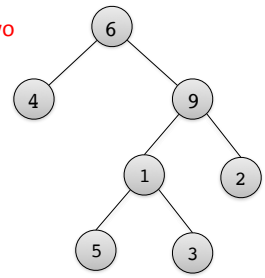
Degree of a node

- Each node can have the following degrees:
  - Zero (no children)
  - One
  - Two



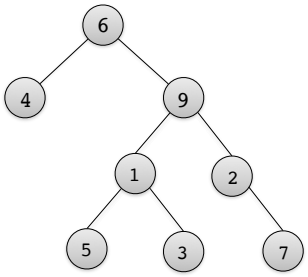
Strictly Binary Tree…

- … is a tree in which every node other than the leaves has two children
- Each node has degree zero or two



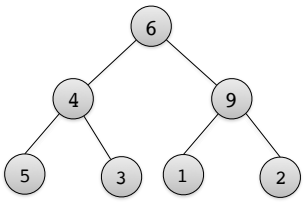
Is the following tree strictly binary?!

- Why?



Complete Binary Tree

- If all leaves of a strictly binary tree are at the same depth



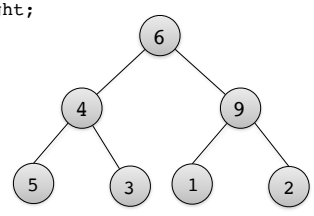
Abstract Data Type: Binary Tree

Binary Tree ADT

```
binary_tree* create_empty_binary_tree();

binary_tree* create_binary_tree(
    int item, binary_tree *left, binary_tree *right);

binary_tree* search(binary_tree *bt, int item);

int is_empty(binary_tree *bt);

void print_in_order(binary_tree *bt);

void print_pre_order(binary_tree *bt);

void print_post_order(binary_tree *bt);
```

Struct

```
struct binary_tree {
  int item;
  binary_tree *left;
  binary_tree *right;
};
```

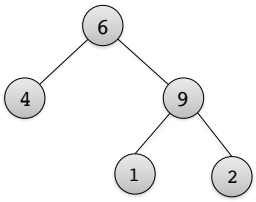


Functions to create Trees…

```
binary_tree* create_empty_binary_tree()
{
  return NULL;
}


binary_tree* create_binary_tree(
      int item, binary_tree *left, binary_tree *right)
{
  binary_tree *new_binary_tree =
      (binary_tree*) malloc(sizeof(binary_tree));
  new_binary_tree->item = item;
  new_binary_tree->left = left;
  new_binary_tree->right = right;
  return new_binary_tree;
}
```

Creating a Tree

```
binary_tree *bt =
    create_binary_tree(6,
        create_binary_tree(4, NULL, NULL),
        create_binary_tree(9,
            create_binary_tree(1, NULL, NULL),
            create_binary_tree(2, NULL, NULL)));
```

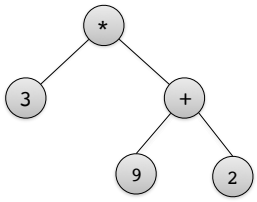


Binary Tree Traversals

Binary Tree Traversals

- Inspecting each node
  - Inorder
  - Preorder
  - Postorder

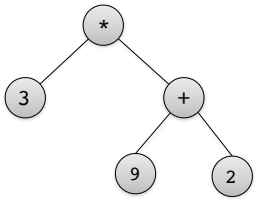
```
3 * (9 + 2)

* 3 + 9 2

3 9 2 + *
```



Inorder

```
void print_in_order(binary_tree *bt)
{
  if (!is_empty(bt)) {
    print_in_order(bt->left);
    printf("%d", bt->item);
    print_in_order(bt->right);
  }
}
```

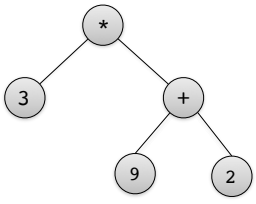
```
3 * (9 + 2)
```



Preorder

```
void print_pre_order(binary_tree *bt)
{
  if (!is_empty(bt)) {
    printf("%d", bt->item);
    print_pre_order(bt->left);
    print_pre_order(bt->right);
  }
}
```

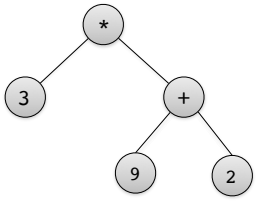
```
* 3 + 9 2
```



Postorder

```
void print_post_order(binary_tree *bt)
{
  if (!is_empty(bt)) {
    print_post_order(bt->left);
    print_post_order(bt->right);
    printf("%d", bt->item);
  }
}
```

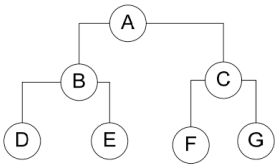
3 9 2 + *



Poscomp 2009

Questão 33. [FUN]

Percorrendo a árvore binária a seguir em pré-ordem, obtemos que seqüência de caracteres?

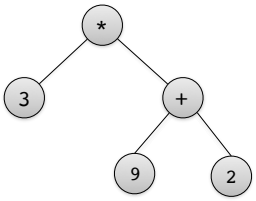


A) A C G F B E D
B) G C F A E B D
C) A B C D E F G
D) D B E A F C G
E) A B D E C F G

Exercise

- Which order would you use to deallocate all nodes?

Postorder!



Binary Search Tree

Binary search in arrays

- O(log n)

12	23	45	67	99	101	212
0	1	2	3	4	5	6

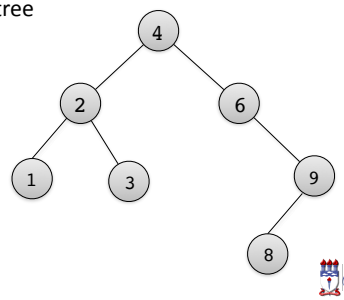
- Array must be sorted
- Insert/Delete operations
  - Rearrange the array
  - Expensive

Dynamic structure

- So we need a dynamic data structure to add/remove nodes in constant time…
- Linked list:
  - Insert/Remove
  - O(1)
- But where is the middle?!

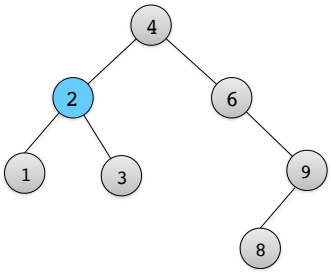
Binary Search Tree

- We have a way of finding the middle element
- The root of a subtree is to the right of its left subtree and to the left of its right subtree



Binary Search Tree

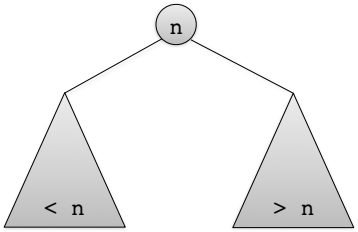
- The item of a node is larger than any item node in its left subtree and smaller than any item node in its right subtree



2 > 1

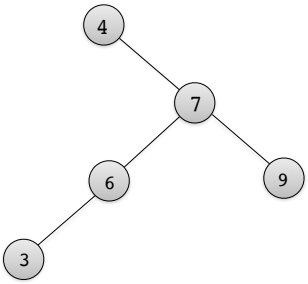
2 < 3

Intuition



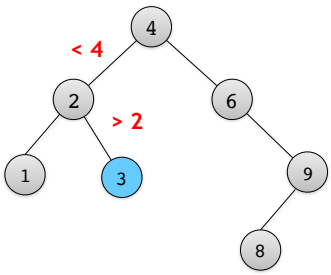
Exercise

- Is the following tree a Binary Search one?



No!
3 < 4

Searching for 3



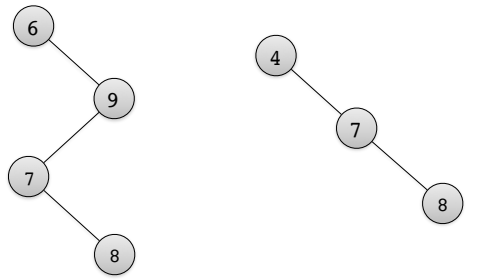
Exercise: implement the search function

- Recursive version
- Try to implement a non-recursive version at home

```
binary_tree* search(binary_tree *bt, int item)
{
  if ((bt == NULL) || (bt->item == item)) {
    return bt;
  } else if (bt->item > item) {
    return search(bt->left, item);
  } else {
    return search(bt->right, item);
  }
}
```

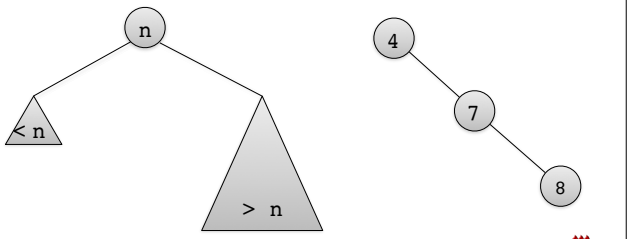
Degenerate Tree

- All nodes have only one subtree



What happens with our binary search?

- O(n)
- Unbalanced trees (AVLs to solve this problem; next class!)



Now we need to keep everything sorted!

Adding nodes…

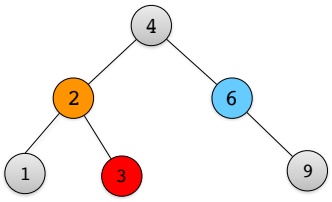
```
binary_tree* add(binary_tree *bt, int item)
{
  if (bt == NULL) {
    bt = create_binary_tree(item, NULL, NULL);
  } else if (bt->item > item) {
    bt->left = add(bt->left, item);
  } else {
    bt->right = add(bt->right, item);
  }
  return bt;
}
```

Using double pointer…

```
void add(binary_tree **bt, int item)
{
  if (*bt == NULL) {
    *bt = create_binary_tree(item, NULL, NULL);
  } else if ((*bt)->item > item) {
    add(&(*bt)->left, item);
  } else {
    add(&(*bt)->right, item);
  }
}
```

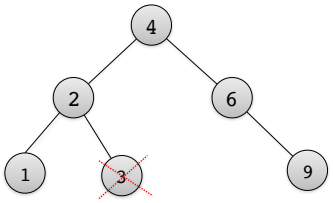
Remove

1. Removing a leaf
2. Removing a node which contains only one subtree
3. Removing a node which contains two subtrees



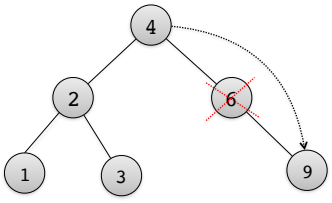
Removing a leaf

- Removing leaf 3



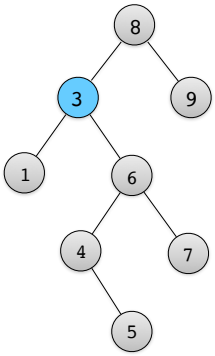
Removing a node with one subtree

- Removing node 6

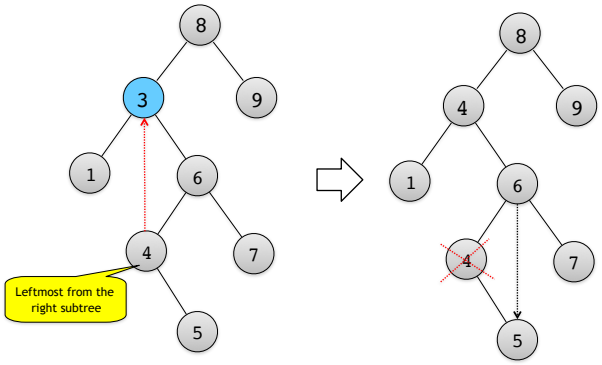


Removing a node with two subtrees

- Removing node N
- Choose node R:
  - Successor (leftmost item from the right subtree)
  - Predecessor (rightmost item from the left subtree)
- Replace the item of N with the R (successor or predecessor)
- Delete R

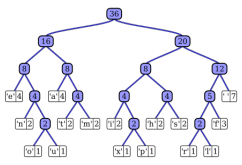


Removing node 3

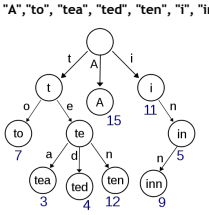


Applications

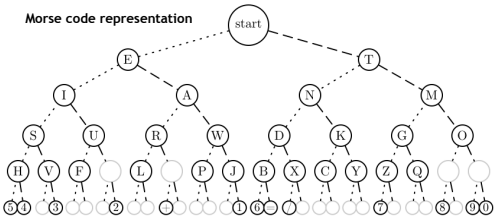
Huffman coding



A trie for keys "A","to", "tea", "ted", "ten", "i", "in", and "inn".

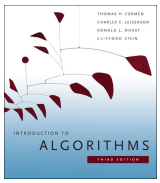


Morse code representation



From *http://apfelmus.nfshost.com/articles/fun-with-morse-code.html*
From *http://en.wikipedia.org/wiki/Trie and /Huffman_coding*

References



Chapter 12

Chapter 6