Algoritmos e Estrutura de Dados

Fabrício Olivetti de França

02 de Fevereiro de 2019

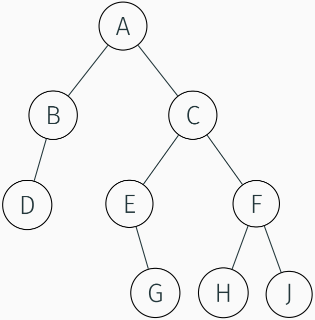
Topics

Árvores

Árvores

Uma árvore é uma estrutura não-linear que permite modelar ramificações, escolhas.

Árvores - Estruturas não-lineares



Algebricamente, uma árvore pode ser representada como:

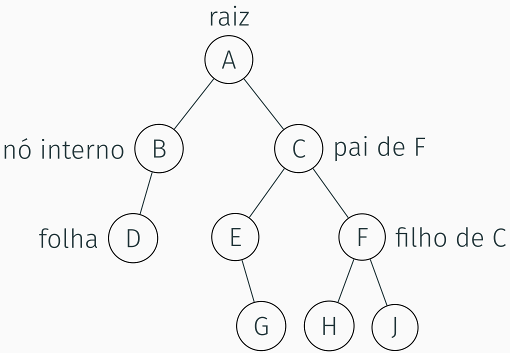
```
T(a) = void | a [T(a)]
```

Ou seja, ou a árvore é vazia ou é uma lista de árvores. A lista vazia representa o final da árvore.

Notações

- **nó:** representa um registro contido na árvore.
- **raiz:** o primeiro elemento da árvore.
- **filho:** um dos elementos que sucede um nó.
- **pai:** elemento que precede um nó.
- **folha:** elementos que não possui filhos.
- **interno:** nó que possui filhos.

Árvores

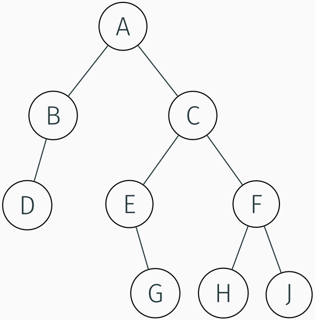


Grau

O **Grau de um nó** é o número de sub-árvores abaixo desse nó (ou quantos filhos ele possui). Note que o grau de um nó folha é 0.

Grau

O Grau do nó A é 2.



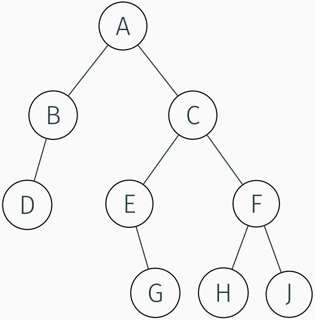
Altura, profundidade e nível

A **altura de um nó** é o caminho mais longo do nó até uma folha. A **altura da árvore** é a altura da raiz.

A **profundidade de um nó** é o número de arestas do nó até a raiz.

O **nível de um nó** é definido como $1+$ o nível de seu pai, sendo o nível da raíz igual a 0.

A altura do nó C é 2, a altura da árvore é 3, a profundidade do nó C é 1 e o nível dele é 1.



Uma árvore pode ser classificada pelo máximo de filhos que cada nó pode ter. O caso trivial é a árvore unária:

```
T(a) = void | a T(a)
```

Isso representa nossa lista ligada!

Árvores Binárias

Árvore Binária

Uma árvore mais comum na computação é a **árvore binária** cujos nós possuem de 0 a 2 filhos:

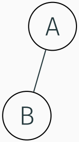
```
T(a) = void | T(a) a T(a)
```

Em C podemos representar como a seguinte estrutura:

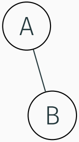
```c
typedef struct bintree {
    TREE_TYPE data;
    struct bintree * left;
    struct bintree * right;
} bintree;
```

Árvore Binária

Note que as seguintes árvores são diferentes!



$\neq$

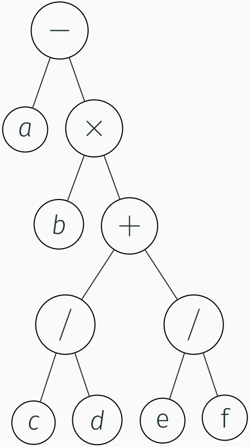


Árvore de Expressão

Um exemplo interessante de árvore binária é a **árvore de expressão** que representa expressões matemáticas prontas para serem avaliadas:

Árvore de Expressão

$$a - b(c/d + e/f)$$



Percurso em Árvores

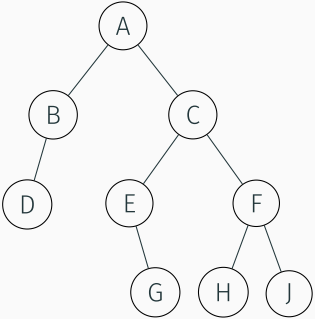
Para percorrer os elementos de uma árvore, a cada nó, temos que decidir qual ramo iremos explorar primeiro.

As três ordens comumente utilizadas são:

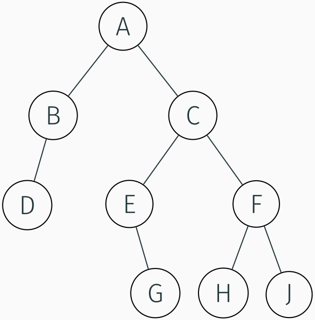
- **Pré-ordem:** raíz - esquerda - direita.
- **Em-ordem:** esquerda - raíz - direita.
- **Pós-ordem:** esquerda - direita - raiz.

Pré-ordem

Qual a ordem dos nós ao fazer o percurso pré-ordem?

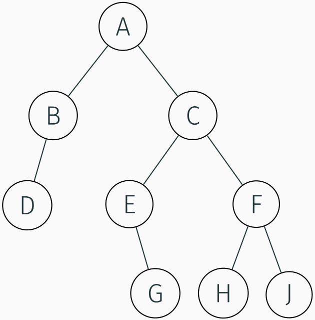


A - B - D - C - E - G - F - H - J



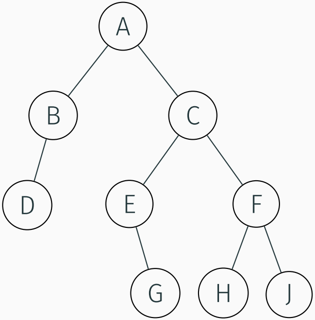
Em-ordem

Qual a ordem dos nós ao fazer o percurso em-ordem?



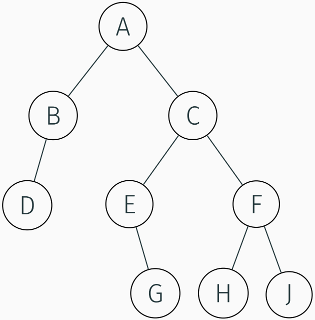
Em-ordem

D - B - A - E - G - C - H - F - J



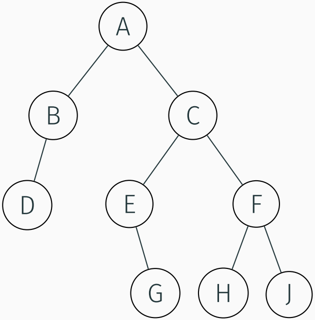
Pós-ordem

Qual a ordem dos nós ao fazer o percurso pós-ordem?



Pós-ordem

D - B - G - E - H - J - F - C - A



Criando uma árvore

Para criar uma árvore precisamos estabelecer o critério de onde inserir cada novo nó.

Criando uma árvore

Vamos adotar o critério de uma árvore ordenada. Para um nó n, todos os nós a esquerda possuem valor menor do que ele e todos os nós a direita valores maiores ou iguais.

Seguindo esse critério, um novo nó será inserido ao encontrarmos um nó folha.

Criando uma árvore

```
tree * create_node (int x) {
  tree * node = malloc(sizeof(tree));
  node->left = node->right = NULL;
  node->x = x;
  return node;
}
```

Criando uma árvore

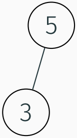
```
tree * insert_sorted (tree * t, tree * node) {

  if (t==NULL) return node;

  if (node->x < t->x)
     t->left = insert_sorted(t->left, node);
  if (node->x > t->x)
     t->right = insert_sorted(t->right, node);

  return t;
}
```

Criando uma árvore

```
insert_sorted(root, create_node(5));
```

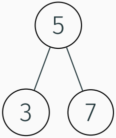
Criando uma árvore

```
insert_sorted(root, create_node(3));
```

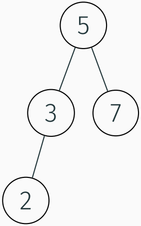


Criando uma árvore

```
insert_sorted(root, create_node(7));
```

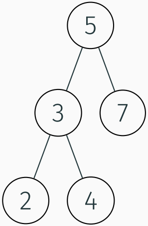


```
insert_sorted(root, create_node(2));
```



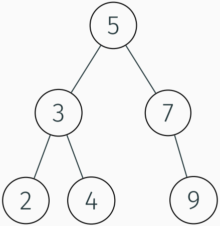
Criando uma árvore

```
insert_sorted(root, create_node(4));
```



Criando uma árvore

```
insert_sorted(root, create_node(9));
```



Para imprimir a sequência de nós no percurso pré-ordem, fazemos:

```
void pre_order(tree * t) {

  if(t==NULL) return;

  printf("%d ", t->x);
  pre_order(t->left);
  pre_order(t->right);

}
```

Porém, para fazer uso da informação da árvore em uma das sequências, o ideal é armazenar em uma lista ligada (esse processo é chamado de *flattening*):

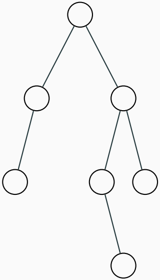
```
List * pre_order(tree * t, List * p) {

  if(t==NULL) return p;

  insere_fim(p, t->x);
  pre_order(t->left, p);
  pre_order(t->right, p);

  return p;
}
```

Percurso em árvore

Como implementar as funções `pos_ordem` e `in_ordem`?

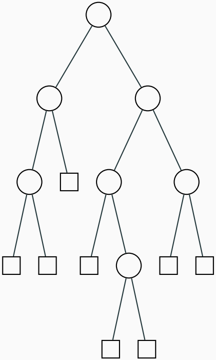
Árvore Estendida

Vamos definir uma árvore binária estendida como uma árvore cujos nós folhas não possuem informação e que todos os nós pais tenham exatamente 2 filhos. Para isso acrescentamos nós falsos em nossa árvore:

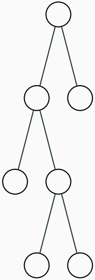


Árvore Estendida

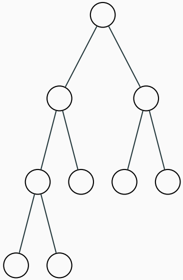
Vamos definir uma árvore binária estendida como uma árvore cujos nós folhas não possuem informação e que todos os nós pais tenham exatamente 2 filhos. Para isso acrescentamos nós falsos em nossa árvore:



Uma árvore binária é **cheia** se todos os nós, exceto os folhas, possuem exatamente 2 filhos.

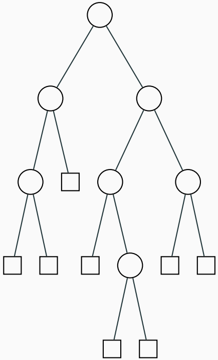


Uma árvore binária é **completa** se todos os níveis, exceto o último, está completamente preenchida.



Árvore Estendida

Voltando a nossa árvore estendida, nela, todo nó possui 2 filhos e todo quadrado possui 0.



Temos um total de $n + s - 1$ arestas pois cada nó induz uma aresta acima, exceto a raiz.

Árvore Estendida

Podemos também dizer que temos um total de $2n$ arestas, pois cada nó circular tem exatamente dois filhos.

Logo: $s = n + 1$

Comprimento Externo

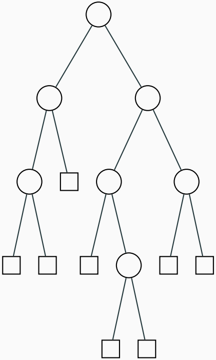
O **comprimento externo** de uma árvore é a soma dos comprimentos da raiz até cada nó externo (quadrado).

Comprimento Interno

O **comprimento interno** de uma árvore é a soma dos comprimentos da raiz até cada nó interno.

$E = 3 + 3 + 2 + 3 + 4 + 4 + 3 + 3 = 25$

$I = 2 + 1 + 0 + 2 + 3 + 1 + 2 = 11$



Uma **árvore binária de busca** é uma árvore ordenada cujo desempenho para encontrar um elemento é equivalente ao de uma busca binária.

Árvore Binária de Busca

Sua característica principal é a de que dado um nó *n*, todos os nós a esquerda possuem um valor menor ou igual a ele e todos os nós a direita possuem um valor maior ou igual a ele.

Essa árvore é ordenada conforme nosso algoritmo de inserção. Para um desempenho ótimo, ela deve ser completa.

Qual o maior valor de comprimento para uma árvore com 1 nó?

Qual o maior valor de comprimento para uma árvore com 1 nó? 0

Árvore Binária de Busca

nós	comprimento
1	0
2	1
3	1
4	2
5	2
6	2
7	2
8	3
9	3
10	3

comprimento $= \lg \lfloor \text{nós} \rfloor$

Temos que o comprimento interno é tão grande quanto a soma:

$\sum_{k=1}^{n} \lg \lfloor k \rfloor = (n+1)\lfloor \lg (n+1) \rfloor - 2^{\lfloor \lg (n+1) \rfloor} + 2$

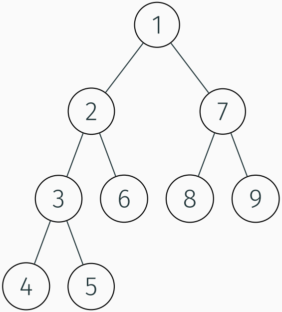
$\approx n \lg n$

Árvore Completa como uma Array

Uma árvore binária completa pode ser representada como uma lista sequencial de tal forma que o pai do índice i está na posição $\lfloor (i-1)/2 \rfloor$ e os filhos de i estão na posição $2i+1$ e $2i+2$.

Árvore Completa como uma Array

0	1	2	3	4	5	6	7	8
1	2	7	3	6	8	9	4	5



Busca com Inserção

Vamos criar agora um algoritmo para buscar por um elemento na árvore, se não encontrar, ele insere na posição correta.

Busca com Inserção

```
tree * search_insert (tree * t, int k) {
  tree * p = t;
  tree * pai = t;

  while(1)
  {
    if (p == NULL)
        return insert_sorted(pai, create_node(k));
    if (p->x == k) return p;
    if (k < p->x) {pai = p; p = p->left;}
    else {pai = p; p = p->right;}
  }
  return NULL;
}
```

Vamos definir como C_N o número de passos médio em uma busca bem sucedida para N nós e C'_N para as buscas não sucedidas.

Temos que:

$$C_N = 1 + \frac{C'_0 + C'_1 + \ldots + C'_{N-1}}{N}$$

pois devemos fazer uma operação a mais (testar a igualdade final).

Temos também que:

$$C_N = \frac{I}{N} + 1$$

$$C'_N = \frac{E}{N+1}$$

Fazendo:

$N(C_N - 1) = I$

e

$(N + 1)C'_N = E$

E sabendo que $E = I + 2N$:

$$(N+1)C'_N - 2N = N(C_N - 1)$$

Substituindo em uma das eqs. anteriores:

$$N(C_N - 1) = C'_0 + C'_1 + \ldots + C'_{N-1}$$

$$(N+1)C'_N = 2N + C'_0 + C'_1 + \ldots + C'_{N-1}$$

Subtraindo

$$NC'_{N-1} = 2(N-1) + C'_0 + C'_1 + \ldots + C'_{N-2}$$

temos:

$$(N+1)C'_N - NC'_{N-1} = 2N - 2(N-1) + C'_{N-1}$$

Resolvendo chegamos a:

$C'_N = C'_{N-1} + \frac{2}{N+1}$

Sabendo que $C'_0 = 0$ temos:

$C'_1 = \frac{2}{2}, C'_2 = \frac{2}{2} + \frac{2}{3}, \ldots$

$C'_N = 1 + 2 \cdot \left(\frac{1}{3} + \frac{1}{4} + \ldots\right) = 2H_{N-1} - 2$

Consequentemente:

$$C_N = \left(1 + \frac{1}{N}\right) 2H_{N-1} - 2 - 1 \approx \ln n$$

Logo a busca tem um custo médio de $O(\ln n)$.

Exercício

Pergunta: o que acontece se eu inserir os elementos de 1 a 10 na sequencia?

Exercício

Por sorte esse tipo de caso é raro, ao adicionar elementos em uma ordem aleatória a chance é que manteremos algo proporcional a $O(\ln n)$.

Próxima Aula

Aprenderemos sobre **árvores AVL** que garantem que nossa árvore estará balanceada (próximo de completa).