Estruturas de Dados

Árvores

Universidade Estadual Vale do Acaraú – UVA

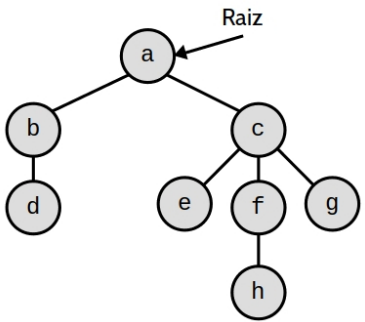
Paulo Regis Menezes Sousa

paulo_regis@uvanet.br

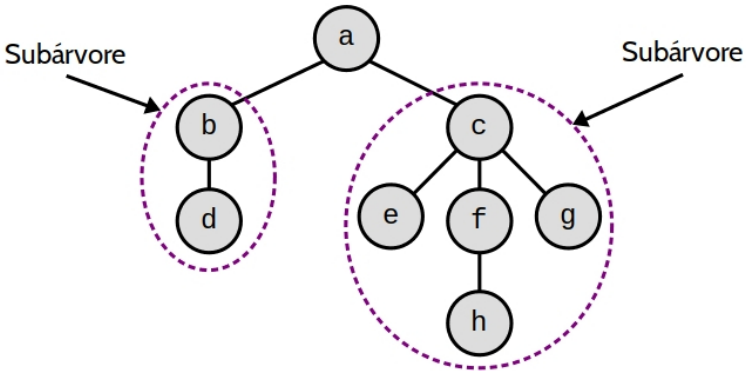
Árvores

- Fundamentos
- Percurso em árvores

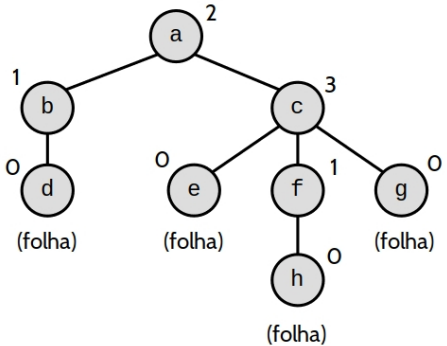
- Uma *árvore* A é uma coleção de $n \geq 0$ nós organizados de forma hierárquica.
- Se $n = 0$, então A é uma árvore vazia.
- Se a árvore não é vazia existe um nó especial em A, denominado **raiz** de A, a partir do qual todos os demais nós de A podem ser acessados.



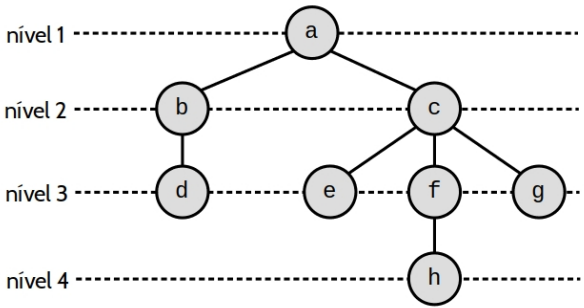
- Por definição, árvores são estruturas recursivas. Quando a raiz de uma árvore é removida, o que sobra é uma coleção de árvores (**subárvores**).
- Enquanto pertencem à raiz essas subárvores são chamadas de **nós filhos**.



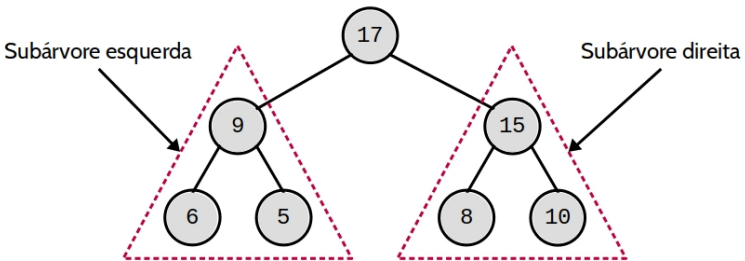
- Um nó que não tem filhos é chamado de **folha**.
- O número de filhos de um nó é o **grau** do nó.
- Um nó folha tem grau zero.
- O grau de uma árvore é o máximo grau de seus nós.



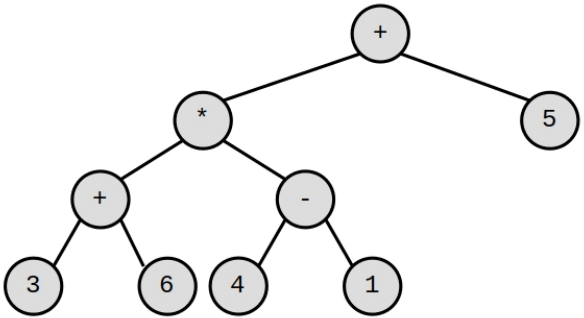
- O **nível** da raiz de uma árvore é 1.
- O nível dos filhos de um nó num nível h é $h + 1$.
- A **altura** de uma árvore é o máximo nível de seus nós (ou 0, se a árvore é vazia).



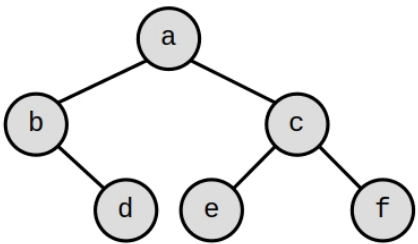
- **Árvore binária** é uma árvore de grau 2, ou seja, uma árvore em que todo nó tem no máximo dois filhos.



- Árvore binária representando expressões aritméticas binárias
  - Nós folhas representam os operandos
  - Nós internos representam os operadores
  - $(3 + 6) * (4 - 1) + 5$



- Notação textual
  - a árvore vazia é representada por <>
  - árvores não vazias por <**raiz** <**sae**> <**sad**> >
    - sae: sub-árvore esquerda
    - sad: sub-árvore direita
- Exemplo:
  - **<a <b <> <d <><> > > <c <e <><> > <f <><> > > >**



- Podemos criar a **representação** de um nó na árvore através de uma **estrutura** em C contendo:
  - A informação armazenada no nó (exemplo: um caractere, ou inteiro)
  - Um ponteiro para a sub-árvore esquerda.
  - Um ponteiro para a sub-árvore direita.

```
1  struct Tree {
2      void *value;
3      struct Tree *left;
4      struct Tree *right;
5  };
```

Código 1: Tree.h

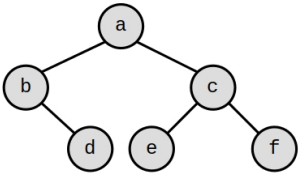
```
1  typedef struct Tree Tree;
2
3  struct Tree {
4      int value;
5      Tree *left;
6      Tree *right;
7  };
8
9  Tree *Tree_alloc(int value, Tree *left, Tree *right);
10 void  Tree_free(Tree *tree);
11 void  Tree_print(Tree *t);
```

- Função `Tree_alloc`
  - Cria um nó raiz dadas a informação e as duas sub-árvores, a da esquerda e a da direita.
  - Retorna o endereço do nó raiz criado.

```
1  Tree *Tree_alloc(int value, Tree *left, Tree *right) {
2      Tree *t = NULL;
3
4      if (value) {
5          t = (Tree*) malloc(sizeof(Tree));
6          t->value = value;
7          t->left  = left;
8          t->right = right;
9      }
10
11     return t;
12 }
```

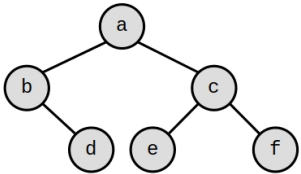
Exemplo

- Criar a árvore <a <b <> <d <><> > > <c <e <><> > <f <><> > > >




```
1  Tree *t1 = Tree_alloc('d', NULL, NULL);
2  Tree *t2 = Tree_alloc('b', NULL, t1);
3  Tree *t3 = Tree_alloc('e', NULL, NULL);
4  Tree *t4 = Tree_alloc('f', NULL, NULL);
5  Tree *t5 = Tree_alloc('c', t3, t4);
6  Tree *t0 = Tree_alloc('a', t2, t5 );
```

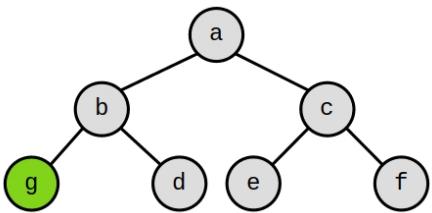
Exemplo



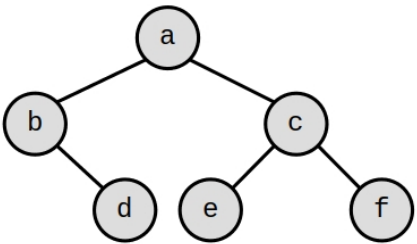

```
1  Tree *t = Tree_alloc('a',
2                       Tree_alloc('b',
3                                  NULL,
4                                  Tree_alloc('d', NULL,NULL)),
5                       Tree_alloc('c',
6                                  Tree_alloc('e', NULL, NULL),
7                                  Tree_alloc('f', NULL, NULL)) );
```

- Acrescentar um nó 'g' como filho esquerdo de 'b'

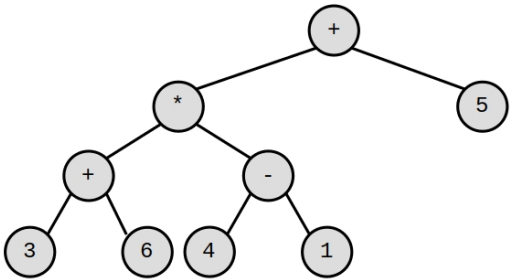
```
1  //(a)->(b)->( ) = g
2  t->left->left = Tree_alloc('g', NULL, NULL);
```



- O percurso (ou a travessia) em uma árvore pode ser realizado de três maneiras distintas.
- Pré-ordem:
  - processa raiz, percorre **sae**, percorre **sad**
  - exemplo: a b d c e f
- Ordem simétrica (ou In-Ordem):
  - percorre **sae**, processa raiz, percorre **sad**
  - exemplo: a b d c e f
- Pós-ordem:
  - percorre **sae**, percorre **sad**, processa raiz
  - exemplo: d b e f c a



- Pré-ordem: $+ * + 36 - 415$
- In-Ordem: $3 + 6 * 4 - 1 + 5$
- Pós-ordem: $36 + 41 - * 5 +$



```
1  void Tree_preOrder(Tree *t) {
2      process(t);
3      Tree_preOrder(t->left);
4      Tree_preOrder(t->right);
5  }
6
7  void Tree_inOrder(Tree *t) {
8      Tree_preOrder(t->left);
9      process(t);
10     Tree_preOrder(t->right);
11 }
12
13 void Tree_postOrder(Tree *t) {
14     Tree_preOrder(t->left);
15     Tree_preOrder(t->right);
16     process(t);
17 }
```

- A função `Tree_free` pode ser implementada como:
  - pré-ordem,
  - pós-ordem ou
  - in-ordem?

```
1  void Tree_free(Tree *t) {
2      //...
3  }
```

- função `Tree_free`
  - libera memória alocada pela estrutura da árvore
    – as sub-árvores devem ser liberadas antes de se liberar o nó raiz

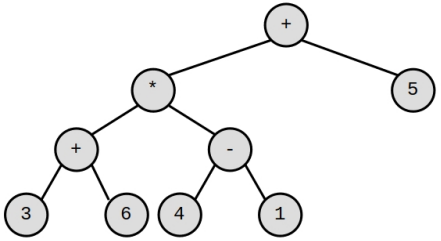
```
1  void Tree_free(Tree *t) {
2      if (t) {
3          Tree_free(t->left);
4          t->left  = NULL;
5
6          Tree_free(t->right);
7          t->right = NULL;
8
9          free(t);
10     }
11 }
```

- função `Tree_print`
  - percorre recursivamente a árvore, visitando todos os nós e imprimindo sua informação

```
1  void Tree_print(Tree *t) {
2      if (t) {
3          if (t->value) {
4              printf("<");
5              printf("%c ", t->value);
6              Tree_print(t->left, print);
7              Tree_print(t->right, print);
8              printf(">");
9          }
10     }
11     else
12         printf("<> ");
13 }
```

Exercício 1

Implemente um TAD Árvore Binária binária capaz de armazenar uma expressão aritmética. Por exemplo, a expressão (3+6)*(4-1)+5 é representada pela árvore binária ilustrada na figura abaixo. As folhas da árvore armazenam operandos e os nós internos operadores. Crie uma função que calcule o resultado final da expressão. Teste sua implementação com a expressão desta árvore que resulta no valor 32.



Exercício 2

Fazer função para retornar o pai de um dado nó de uma árvore

- Dado um item, a função o procura na árvore
- Caso o item seja encontrado, a função retorna o conteúdo do pai do nó

Exercício 3

Escrever uma função que calcule a altura de uma árvore binária dada. A altura de uma árvore é igual ao máximo nível de seus nós.

Exercício 4

Escrever o algoritmo de visita em Pré-Ordem sem utilizar recursividade.