# Árvores, Árvores Binárias e Árvores Binárias de Busca

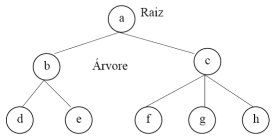
Danielle B. Colturato

## Árvores

- São estruturas de dados que caracterizam uma relação entre os dados;
- A relação existente entre os dados é uma relação de hierarquia ou de composição (um conjunto é subordinado a outro).

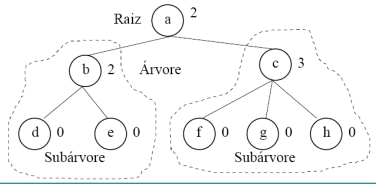
## Árvores - Definição

- Uma Árvore pode ser definida recursivamente como:
  - Uma estrutura vazia é uma árvore vazia;
  - Se $t_1$, ..., $t_k$ são árvores disjuntas, então a estrutura cuja raiz tem como suas filhas as raízes de $t_1$, ..., $t_k$ também é uma árvore;
  - Somente estruturas geradas pelas regras 1 e 2 são árvores.



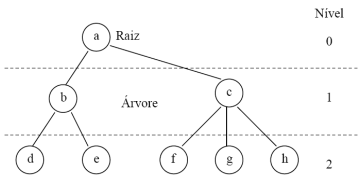

## Árvores

- **Grau**
  - Indica o número de sub-árvores de um nó.
  - ***Observação***: Se um nodo não possuir nenhuma sub-árvore é chamado de **Nó Terminal** ou **Folha**.



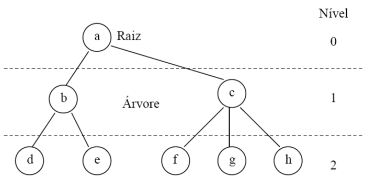

## Árvores

- **Nível**
  - É o comprimento (número de linhas) do caminho (raiz até nó).



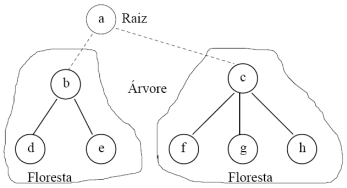

## Árvores

- **Altura**
  - É o nível mais alto da árvore ou o comprimento do caminho mais longo da raiz até uma das folhas.



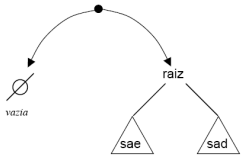
## Árvores

- **Floresta**
  - É um conjunto de zero ou mais árvores disjuntas, ou seja, se for eliminado o nó raiz da árvore, as sub-árvores que restarem chamam-se de florestas.



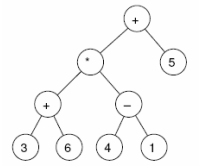

## Árvores Binárias

- São estruturas do tipo árvore onde o grau de cada nó é menor ou igual a dois.
- Uma árvore binária é:
  - uma árvore vazia; ou
  - um nó raiz com duas sub-árvores:
    - a sub-árvore da direita (sad)
    - a sub-árvore da esquerda (sae)



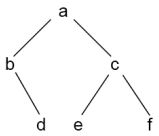

## Árvores Binárias

- Exemplo:
  - Representação de expressões aritméticas
    - Nós folhas representam os operandos;
    - Nós internos representam os operadores;
    - Ex: (3+6)*(4-1)+5



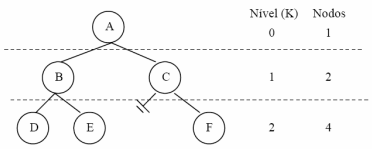

## Árvores Binárias

- Notação textual:
  - a árvore vazia é representada por <>
  - árvores não vazias por <raiz sae sad>
  - exemplo: <a <b <> <d<><>> > <c <e<><>> <f<><>>> >




## Árvores Binárias

- **Propriedades**
  - O número máximo de nodos no k-ésimo nível de uma árvore binária é $2^k$;
  - O número máximo de nodos em uma árvore binária com altura k é $2^{k+1}$-1, para k >= 0;
  - Árvore completa: árvore de altura k com $2^{k+1}$-1 nodos.




## Árvores Binárias

- Representação de uma árvore:
  - através de um ponteiro para o nó raiz
- Representação de um nó da árvore:
  - estrutura em C contendo
    - a informação propriamente dita (exemplo: um caractere)
    - dois ponteiros para as sub-árvores, à esquerda e à direita

```
struct arv {
   char info;
   struct arv* esq;
   struct arv* dir;
};
```

## Árvores Binárias

- Interface do tipo abstrato Árvore Binária:

```
typedef struct arv Arv;

Arv* arv_criavazia (void);
Arv* arv_cria (char c, Arv* e, Arv* d);
Arv* arv_libera (Arv* a);
int  arv_vazia (Arv* a);
int  arv_pertence (Arv* a, char c);
void arv_imprime (Arv* a);
```

## Árvores Binárias

- função arv_criavazia
  - cria uma árvore vazia

```
Arv* arv_criavazia (void)
{
   return NULL;
}
```

## Árvores Binárias

- função arv_cria
  - 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

```
Arv* arv_cria (char c, Arv* sae, Arv* sad)
{
   Arv* p=(Arv*)malloc(sizeof(Arv));
   p->info = c;
   p->esq = sae;
   p->dir = sad;
   return p;
}
```

## Árvores Binárias

- criavazia e cria
  - as duas funções para a criação de árvores representam os dois casos da definição recursiva de árvore binária:
  - uma árvore binária **Arv* a;**
    - é vazia **a=arv_criavazia()**
    - é composta por uma raiz e duas sub-árvores **a=arv_cria(c,sae,sad);**

## Árvores Binárias

- função arv_libera
  - libera memória alocada pela estrutura da árvore
    - as sub-árvores devem ser liberadas antes de se liberar o nó raiz
  - retorna uma árvore vazia, representada por NULL

```
Arv* arv_libera (Arv* a){
   if (!arv_vazia(a)){
      arv_libera(a->esq);     /* libera sae */
      arv_libera(a->dir);     /* libera sad */
      free(a);                /* libera raiz */
   }
   return NULL;
}
```

## Árvores Binárias

- função arv_vazia
  - indica se uma árvore é ou não vazia

```
int arv_vazia (Arv* a)
{
   return a==NULL;
}
```

## Árvores Binárias

- função arv_pertence
  - verifica a ocorrência de um caractere c em um de nós
  - retorna um valor booleano (1 ou 0) indicando a ocorrência ou não do caractere na árvore

```
int arv_pertence (Arv* a, char c){
   if (arv_vazia(a))
      return 0; /* árvore vazia: não encontrou */
   else
      return a->info==c ||
             arv_pertence(a->esq,c) ||
             arv_pertence(a->dir,c);
}
```

## Árvores Binárias

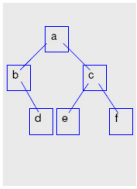
- função arv_imprime
  - percorre recursivamente a árvore, visitando todos os nós e imprimindo sua informação

```
void arv_imprime (Arv* a)
{
   if (!arv_vazia(a)){
      printf("%c ",a->info); /* mostra raiz */
      arv_imprime(a->esq);   /* mostra sae */
      arv_imprime(a->dir);   /* mostra sad */
   }
}
```

## Árvores Binárias

Exemplo: <a <b <> <d <><>> > <c <e <><> > <f <><> > > >

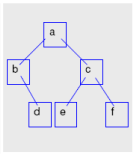
```
/* sub-árvore 'd' */
Arv* a1= arv_cria('d',arv_criavazia(),arv_criavazia());
/* sub-árvore 'b' */
Arv* a2= arv_cria('b',arv_criavazia(),a1);
/* sub-árvore 'e' */
Arv* a3= arv_cria('e',arv_criavazia(),arv_criavazia());
/* sub-árvore 'f' */
Arv* a4= arv_cria('f',arv_criavazia(),arv_criavazia());
/* sub-árvore 'c' */
Arv* a5= arv_cria('c',a3,a4);
/* árvore 'a' */
Arv* a = arv_cria('a',a2,a5 );
```




## Árvores Binárias

- Exemplo: <a <b <> <d <><>> > <c <e <><> > <f <><> > > >

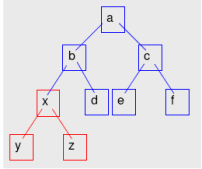
```
Arv* a = arv_cria('a',
           arv_cria('b',
              arv_criavazia(),
              arv_cria('d', arv_criavazia(), arv_criavazia())
           ),
           arv_cria('c',
              arv_cria('e', arv_criavazia(), arv_criavazia()),
              arv_cria('f', arv_criavazia(), arv_criavazia())
           )
         );
```




## Árvores Binárias

- Exemplo - acrescenta nós

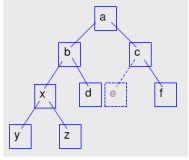
```
a->esq->esq =
   arv_cria('x',
            arv_cria('y',
                     arv_criavazia(),
                     arv_criavazia()),
            arv_cria('z',
                     arv_criavazia(),
                     arv_criavazia())
           );
```




## Árvores Binárias

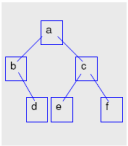
- Exemplo - libera nós

```
a->dir->esq = libera(a->dir->esq);
```



## Árvores Binárias

- Ordens de percurso:
  - *pré-ordem*:
    - trata *raiz*, percorre *sae*, percorre *sad*
    - exemplo: a b d c e f
  - *ordem simétrica*:
    - percorre *sae*, trata *raiz*, percorre *sad*
    - exemplo: b d a e c f
  - *pós-ordem*:
    - percorre *sae*, percorre *sad*, trata *raiz*
    - exemplo: d b e f c a




## Árvores Binárias

```
void PreOrdem(Arv* a){
    if (a!= NULL){
        printf("%c", a->info);
        PreOrdem(a->esq);
        PreOrdem(a->dir);
    }
}
```

```
void InOrdem(Arv* a){
    if (a!= NULL){
        InOrdem(a->esq);
        printf("%c", a->info);
        InOrdem(a->dir);
    }
}
```

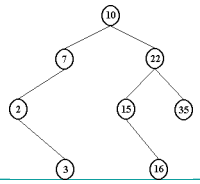
```
void PosOrdem(Arv* a){
    if (a!= NULL){
        PosOrdem(a->esq);
        PosOrdem(a->dir);
        printf("%c", a->info);
    }
}
```

## Árvore Binária de Busca

- Uma **Árvore Binária de Busca T (ABB)** ou **Árvore Binária de Pesquisa** é tal que T = 0 e a árvore é dita vazia ou seu nó raiz contém uma chave e:
  - Todas as chaves da sub-árvore esquerda são menores que a chave da raiz.
  - Todas as chaves da sub-árvore direita são maiores que a chave raiz.
  - As sub-árvores direita e esquerda são também Árvores Binárias de Busca.

## Árvore Binária de Busca

- Em algoritmo de busca a medida de eficiência é dada pelo número de comparações necessárias para se localizar uma chave, ou descobrir que ela não existe.
- Numa lista linear com *n* chaves, temos que, no *pior caso*, faremos *n comparações*. O número de comparações cresce linearmente em função do número de chaves.




## Árvore Binária de Busca

```
int Busca (Arv* a, char x)
{
   if (a == NULL)
      return 0;
   else
      if (a->info < x)
            return Busca(a->dir,x);
        else if (a->info > x)
            return Busca(a->esq,x);
            else
                  return 1;
}
```

## Árvore Binária de Busca

- **Problemas com Árvores Binárias**
  - Deterioração
    - Quando inserimos utilizando a inserção simples, dependendo da distribuição de dados, pode haver deterioração.
    - Árvores deterioradas perdem a característica de eficiência de busca.

## Árvores AVL

- **Manter uma árvore de busca balanceada sob a presença de constantes inserções e deleções é ineficiente.**
- **Para contornar este problema foi criada a árvore AVL (Adelson-Velskii e Landis).**
- **Árvores AVL p**ermitem inserção/deleção e rebalanceamento aceitavelmente rápidos.

## Referências


DROZDEK, Adam. Estruturas de dados e algoritmos em C++. São Paulo: Pioneira Thomson Learning, 2005.

VELOSO, Paulo; SANTOS, Clésio. *Estruturas de Dados.* 4 ed. Rio de Janeiro: Campus, 1986.

WIRTH, Niklaus. *Algoritmos e Estruturas de Dados. Rio de Janeiro: LTC, 1999.*