

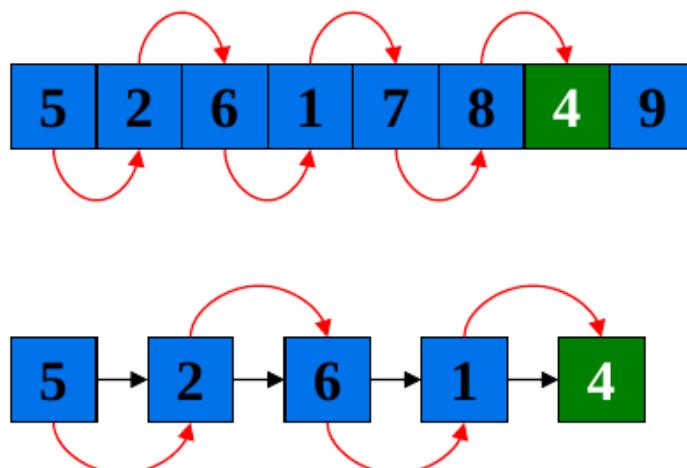
# ARVORES

Prof. Marcelo Dib

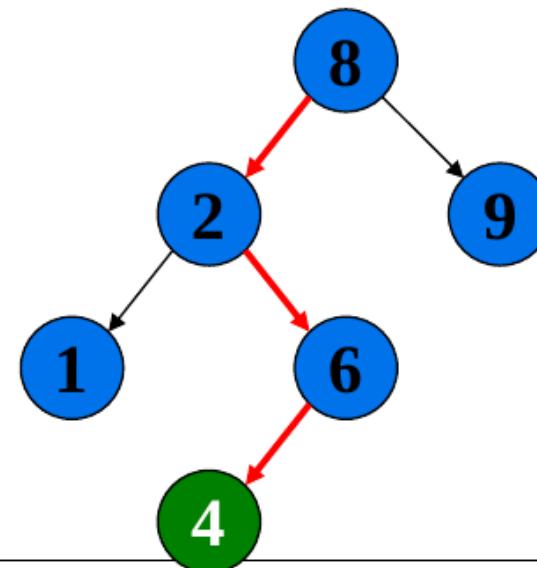
# Árvores

- Motivação

Ex: Arrays e listas



Ex: Árvores



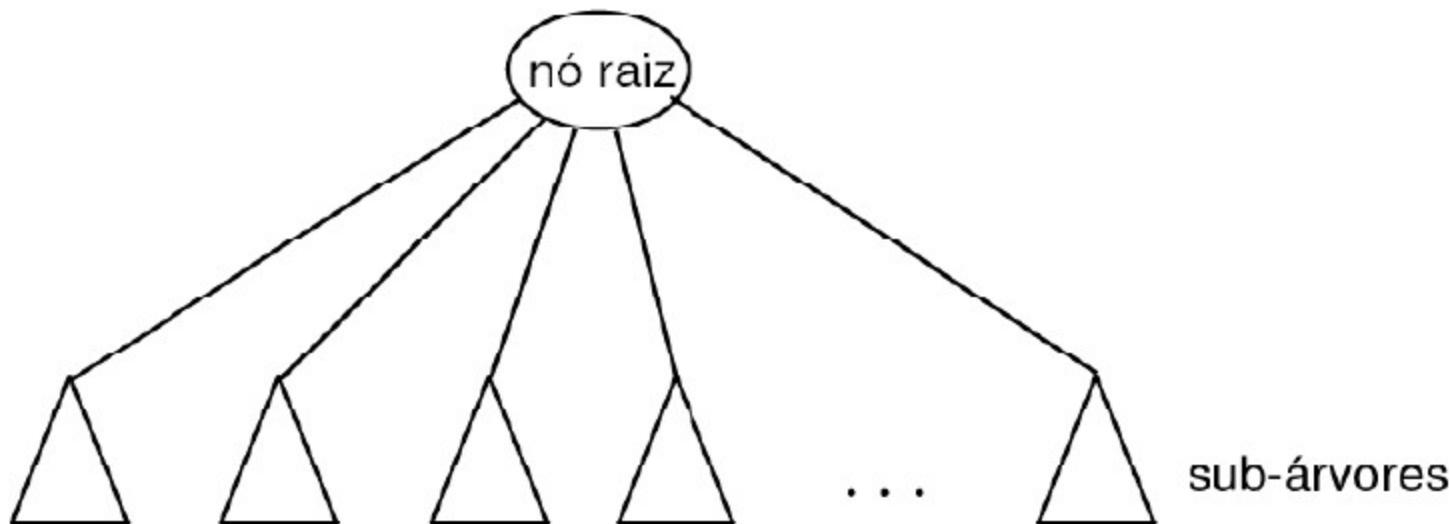
# Árvores

- **Motivação**
  - Em diversas aplicações, necessita-se de estruturas mais complexas do que as sequenciais;
  - Entre essas, destacam-se as árvores, por existirem inúmeros problemas práticos, que podem ser modelados através delas ;
  - As árvores possuem um tratamento computacional simples e eficiente ;

# Árvores

- **Definição**
  - Uma árvore enraizada  $T$ , ou simplesmente árvore, é um conjunto finito de elementos denominados nós ou vértices, tais que :
  - $T = \emptyset$  , e a árvore é dita vazia , ou
  - Existe um nó especial,  $r$  , denominado raiz de  $T$  ; os restantes constituem um único conjunto vazio ou são divididos em  $m \geq 1$  conjunto disjuntos não vazios, as subárvores, cada qual por sua vez uma árvore.
  - Uma floresta é um conjunto de árvores.
  - Se  $v$  é um nó de  $T$ , a notação  $T_v$  indica a subárvore de  $T$  com raiz  $v$  ;

# Árvores

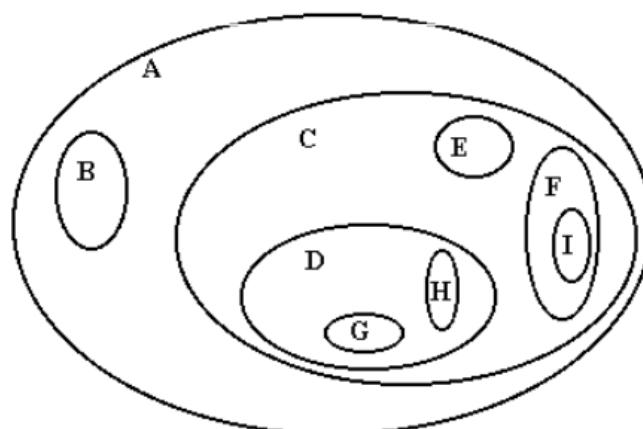


# Árvores

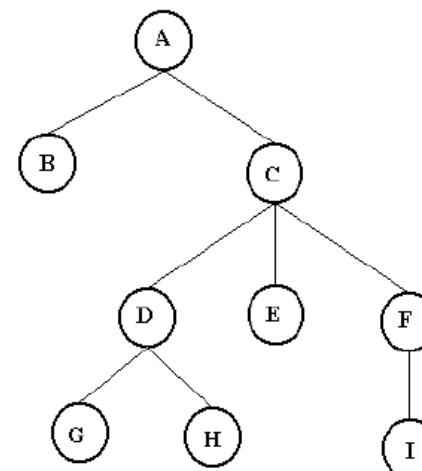
- **Formas de Representação**

- Representação por parênteses aninhados
  - ( A (B) ( C (D (G) (H)) (E) (F (I))))

Diagrama de Inclusão



Representação Hierárquica



# Árvores

## Conceitos Básicos

- Seja  $v$  o nó raiz da subárvore  $T_v$  de  $T$ ;
  - Os nós raízes  $w_1, w_2 \dots w_j$  das subárvores de  $T_v$  são chamados filhos de  $v$  ;
  - Os nós  $w_1, w_2 \dots w_j$  são denominados irmãos ;
  - Se  $z$  é filho de  $w_1$ , então  $w_2$  é tio de  $z$  ; e  $v$  é avô de  $z$  ;
  - O número de filhos de um nó é denominado grau de saída do nó ;

# Árvores

## Conceitos Básicos

- Seja  $v$  o nó raiz da subárvore  $T_v$  de  $T$ ;
  - Se  $x$  pertence a subárvore  $T_v$ ,  $x$  é descendente de  $v$  e ,  $v$  é ancestral de  $x$  ;
  - Neste caso, sendo  $x$  diferente de  $v$  ,  $x$  é descendente próprio de  $v$  e ,  $v$  é ancestral próprio de  $x$  ;
  - Um nó que não possui descendentes próprios é chamado folha ;
  - Um nó não folha é dito interior;

# Árvores

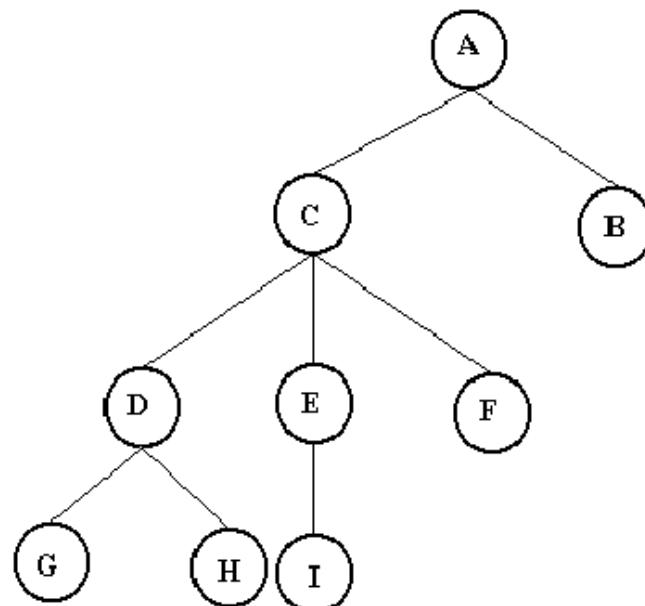
- **Conceitos Básicos**
- Caminho
  - Uma sequência de nós distintos  $v_1, v_2, \dots, v_k$ , tal que existe sempre entre nós consecutivos (isto é, entre  $v_1$  e  $v_2$ , entre  $v_2$  e  $v_3$ , ...,  $v_{(k-1)}$  e  $v_k$ ) a relação "é filho de" ou "é pai de" é denominada um caminho na árvore.
- Comprimento do Caminho
  - Um caminho de  $v_k$  vértices é obtido pela sequência de  $k-1$  pares. O valor  $k-1$  é o comprimento do caminho.
- Nível ou profundidade de um nó
  - número de nós do caminho da raiz até o nó.

# Árvores

- Nível da raiz é portanto 1 ;
- A altura de um nó  $v$  é o numero de nós do maior caminho de  $v$  até um de seus descendentes;
- As folhas tem altura 1 ;
- A altura da árvore  $T$  é igual ao nível máximo de seus nós ;

# Árvores

- Arvore Ordenada: é aquela na qual filhos de cada nó estão ordenados. Assume-se ordenação da esquerda para a direita. Esta árvore é ordenada?

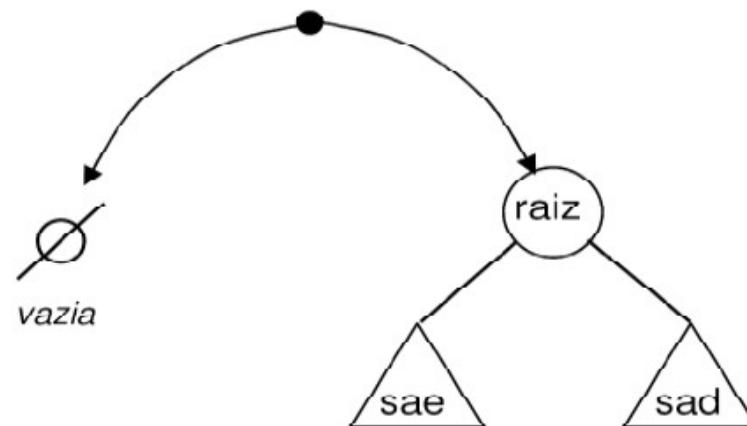


# Árvores

## Árvore Binária

12

- Uma árvore em que cada nó tem zero, um ou dois filhos
- Uma árvore binária é:
  - uma árvore vazia; ou
  - um nó raiz com duas sub-árvore:
    - a subárvore da direita (sad)
    - a subárvore da esquerda (sae)



# Árvores

- Árvore Cheia
    - todos os seus nós internos têm duas sub-árvores associadas
    - número  $n$  de nós de uma árvore cheia de altura  $h$
    - $n = 2^{h+1} - 1$
- 
- The diagram illustrates a full binary tree with four levels. Level 0 (the root) has 1 node. Level 1 has 2 nodes. Level 2 has 4 nodes. Level 3 has 8 nodes. Each node is represented by a circle, and each connection is a straight line.
- | Nível | Expoente de 2 | Quantidade de Nós |
|-------|---------------|-------------------|
| 0     | 0             | $2^0 = 1$ nó      |
| 1     | 1             | $2^1 = 2$ nós     |
| 2     | 2             | $2^2 = 4$ nós     |
| 3     | 3             | $2^3 = 8$ nós     |

# Árvores

## Implementação em C

```
struct no
{ int info ;
  struct no * esq;
  struct no * dir ;
};
typedef struct no no ;
```

# Árvores

- Percurso em Arvore Binária
  - Pré-ordem

```
void pre_ordem( no * pt)
{ printf("\n %d ",pt->info);
  if (pt->esq != NULL) pre_ordem(pt->esq);
  if (pt->dir != NULL) pre_ordem(pt->dir);
}
```

# Árvores

- Percurso em Arvore Binária
  - Em ordem simétrica

```
void sim_ordem( no * pt)
{ if (pt->esq != NULL) pre_ordem(pt->esq);
  printf("\n %d ",pt->info);
  if (pt->dir != NULL) pre_ordem(pt->dir);
}
```

# Árvores

- Percurso em Arvore Binária
  - Pos-ordem

```
void pos_ordem( no * pt)
{
    if (pt->esq != NULL) pre_ordem(pt->esq);
    if (pt->dir != NULL) pre_ordem(pt → dir);
    printf("\n %d ",pt->info);
}
```

# Árvores

- Árvores Binárias de Busca
  - Estrutura de dados adequado a problemas de busca ;
  - Dado um conjunto de elementos, onde cada um é identificado por uma chave, o objetivo é localizar, neste conjunto, o elemento procurado ;

# Árvores

Definição :

Seja  $S = \{s_1, s_2, \dots, s_n\}$  o conjunto de chaves satisfazendo  $s_1 < s_2 < s_3 \dots < s_n$ . Seja  $x$  um valor dado. O objetivo é verificar se  $x$  pertence a  $S$ .

Para resolver este problema emprega-se uma árvore binária rotulada  $T$ , com as seguintes características :

- (i)  $T$  possui  $n$  nós. Cada chave corresponde a uma chave distinta  $s_j \in S$  e possui como rótulo o valor  $r(v) = s_j$ ;
- (ii) Seja um nó  $v$  de  $T$ . Seja também  $v_1$ , pertencente a subárvore esquerda de  $v$ . Então  $r(v_1) < r(v)$  ;
- (iii) Seja um nó  $v$  de  $T$ . Seja também  $v_2$ , pertencente a subárvore direita de  $v$ . Então  $r(v_2) > r(v)$  ;

# Árvores

- Função Busca
  - $f = 0$  , se a árvore é vazia ;
  - $f = 1$  , se  $x$  foi encontrado ; neste caso pt aponta para o nó procurado
  - $f > 1$  ,  $x$  não foi encontrado ;

# Árvores

```
no * busca_arvore (no * pt, int x,int *f)
{if (pt==NULL) {*f=0;    return(NULL);}
 else
 if (pt->info == x) { *f = 1 ; return(pt); }
 else
 if (pt->info > x)
 if (pt->esq == NULL) { *f = 2 ; return(pt);}
 else
 { pt = pt->esq ; busca_arvore(pt,x,f);}
 else
 if (pt->dir == NULL) { *f = 2 ;  return(pt);}
 else
 { pt = pt->dir ;  busca_arvore(pt,x,f);}
}
```

# Árvores

- Função Inserção

```
no * insercao(no * pt, int x )
{no * pt1 , *pt2;  int f ;
pt1 = busca_arvore(pt,x,&f);
if (f==1) printf("inserao Invalida");
else
{ pt2 = malloc (sizeof(no));
pt2->info = x;
pt2->esq = NULL ;
pt2->dir = NULL ;
if (f==0) {return(pt2);}
else
if (f==2)
if (x<pt1->info )
pt1->esq = pt2 ;
else
pt1->dir = pt2 ;
}
}
```

# Árvores

- Função Remoção

```
no * deletar_no(no *r , int x)
{ no *pt , *pt1 ;
  if (r==NULL) return(NULL);
  else
    if (r->info == x)
      {if (r->esq==r->dir) {free(r);  return(NULL);}
       else
         if (r->esq==NULL) {pt = r ;  r = r->dir;  free(pt);  return(r);}
           else
             if (r->dir==NULL) {pt = r ;  r = r->esq;  free(pt);  return(r);}
               else
                 if ((r->esq)->esq == (r->dir)->dir){r->info = (r->esq)->info ;  free(r->esq);  r->esq = NULL;}
                   else
                     {pt = r ;  pt1 = r->esq ;
                      if (pt1->dir == NULL) {r->info = pt1->info ;  r->esq = pt1->esq ; free(pt1);}
                        eles {  while(pt1->dir != NULL)
                          {pt = pt1 ;  pt1 = pt1->dir ;}
                            r->info = pt1->info ;
                            pt->dir = pt1->esq;
                            free(pt1); }
                      }
                    }
      }
    if(r->info >x) r->esq = deletar_no(r->esq,x);
    eles   r->dir = deletar_no(r->dir,x);
  return (r);
}
```