# Alocação Encadeada

O mesmo que discutimos a respeito das filas ocorre com as pilhas.

Ou seja, uma pilha nada mais é do que uma lista com uma disciplina de acesso.

Logo, podemos nos utilizar de todos os conceitos vistos em listas para implementarmos pilhas.

Por exemplo, podemos utilizar uma lista encadeada com nó cabeçalho para armazenar uma pilha.

# Árvores

# Árvores - Conceitos

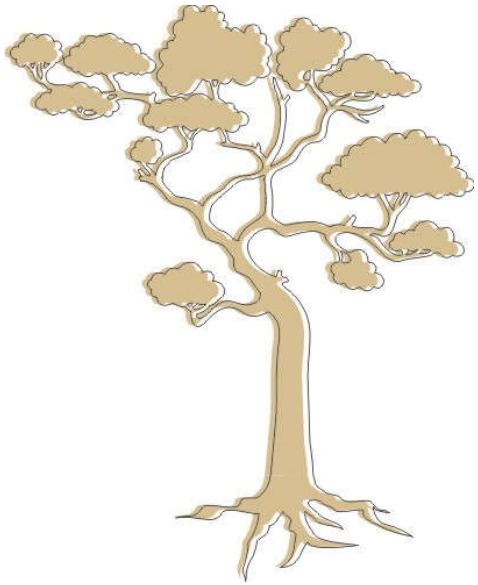
Qual a carência das pilhas e filas?

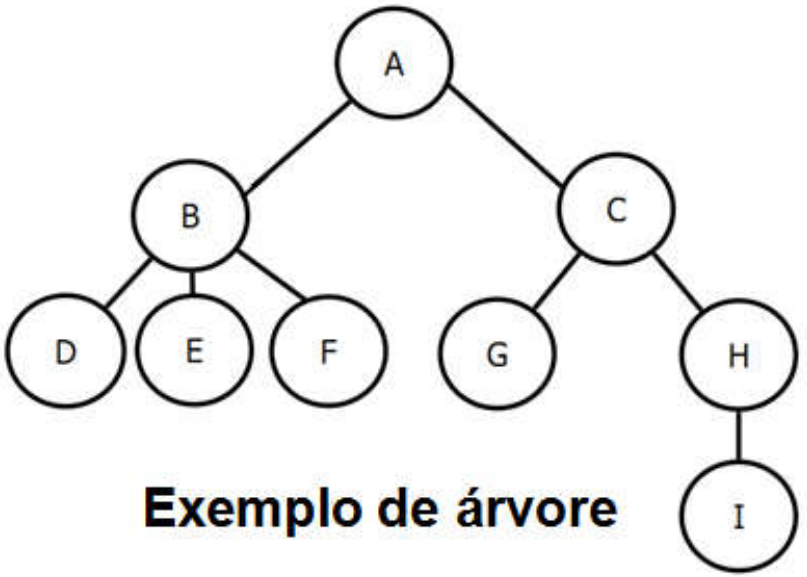
**Estas são de difícil utilização para a representação hierárquica de elementos.**

Devido a…

**Serem limitadas a apenas uma dimensão.**

Visando eliminar esta limitação foi criado o conceito de árvore.

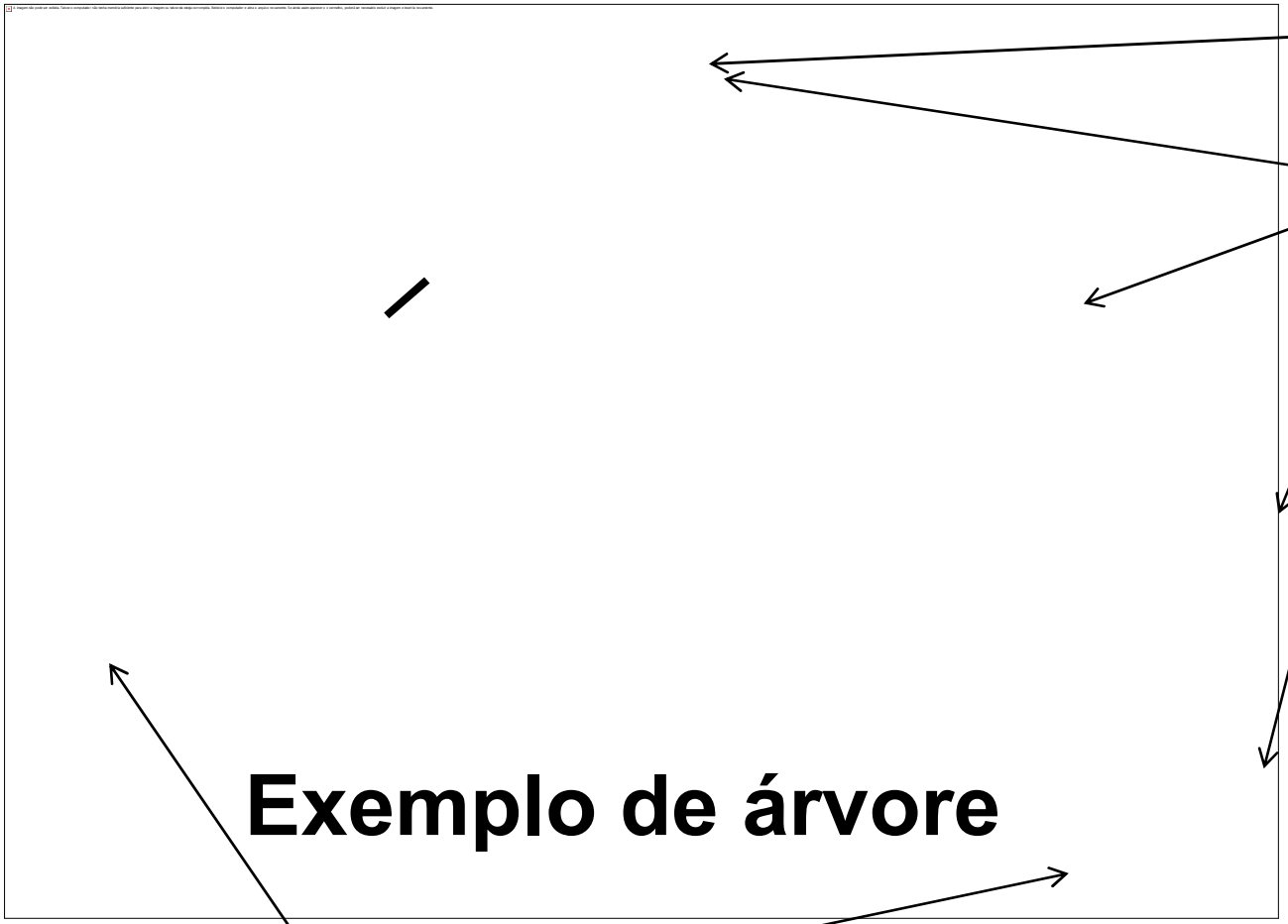





Exemplo de árvore

# Árvores - Conceitos

Conceitos:



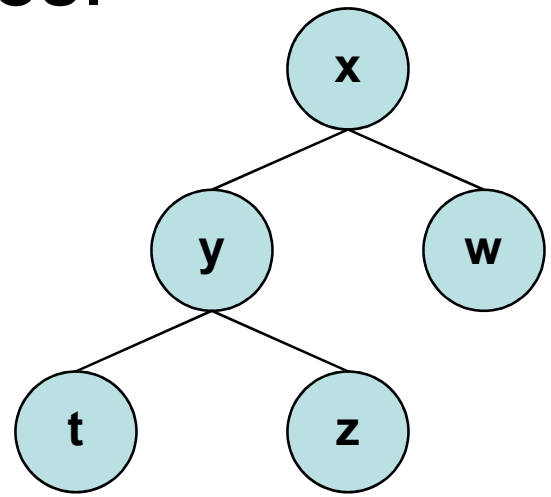
# Árvores - Conceitos

Uma definição recursiva para árvore é:

1. Uma estrutura vazia é uma árvore.

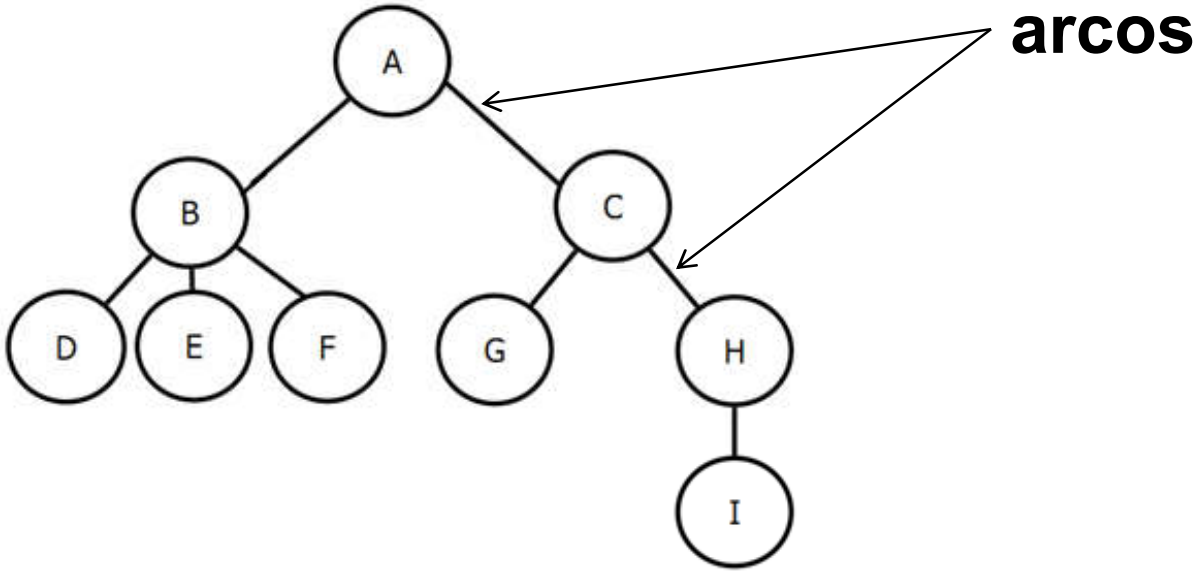
2. Se $t_1$, …, $t_k$ são árvores disjuntas, então a estrutura cuja raiz tem como suas filhas as raízes de $t_1$, …, $t_k$ também é uma árvore.

3. Somente estruturas geradas pelas regras 1 e 2 são árvores.



# Árvores - Conceitos

Qual o conceito de caminho?

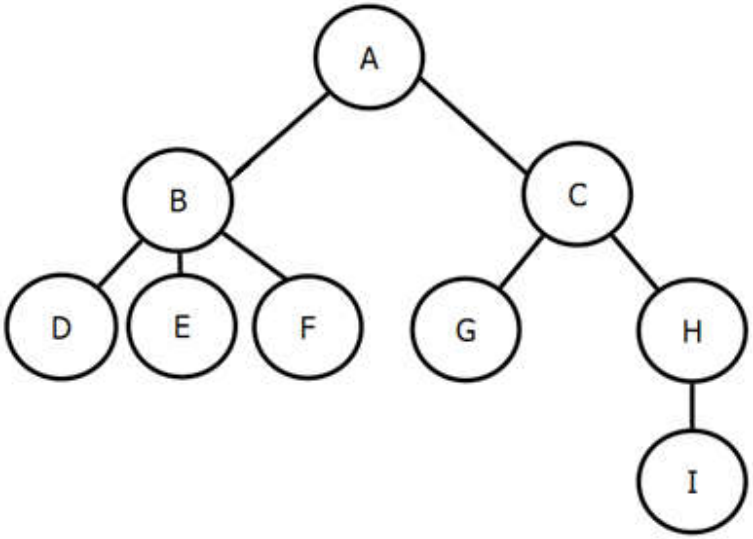



Caminho é a sequência de arcos, com origem na raiz e final em um determinado nó.

Quantos caminhos existem para se atingir um determinado nó?

Apenas um.

# Árvores - Conceitos




O que determina o tamanho de um caminho?

**O número de arcos no mesmo.**

Qual o nível de um determinado nó?

**O nível de um nó em uma árvore é definido da seguinte forma: a raiz da árvore tem nível zero e o nível de qualquer outro nó na árvore é um nível a mais que o de seu pai.**

# Árvores - Conceitos

Qual a altura (ou profundidade) de uma árvore não vazia?

**O nível máximo de um nó na árvore.**

A árvore vazia é uma árvore legítima de altura 0 (zero).

A definição de árvore impõe alguma restrição sobre a quantidade de filhos de um nó?

**Não. Esta pode variar de zero até qualquer inteiro positivo.**

Porém, existem alguns tipos particulares de árvores, cuja suas definições impõem algumas restrições. Por exemplo, árvores binárias.



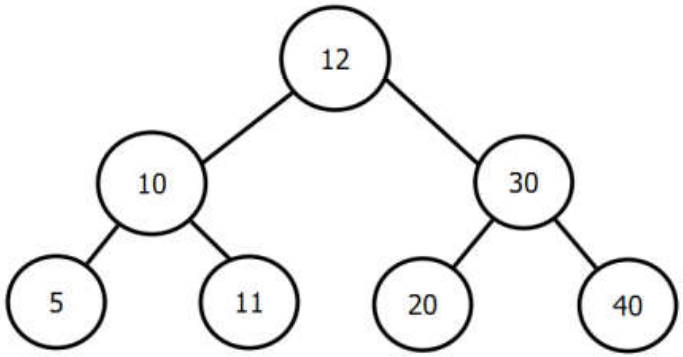
UNIVASF

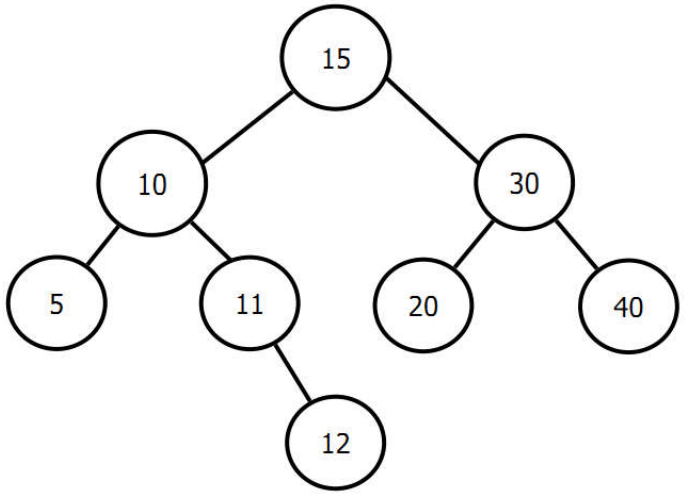
# Árvores Binárias

O que é uma árvore binária?

**É uma árvore cujos nós têm dois filhos (possivelmente vazios) e cada filho é designado como filho à esquerda ou filho à direita.**

Ex.:




Em uma árvore binária existem no máximo quantos nós em um determinado nível?

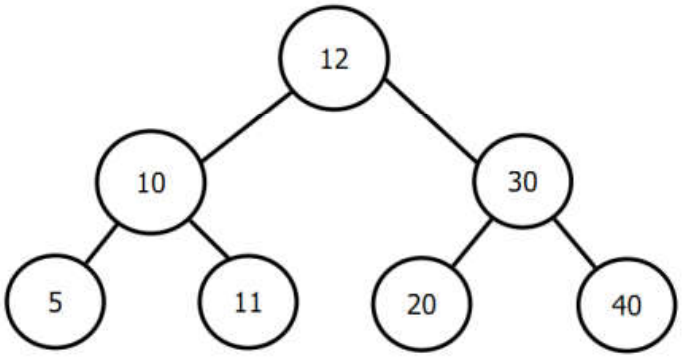
**Existem no máximo $2^i$ nós no nível *i*.**

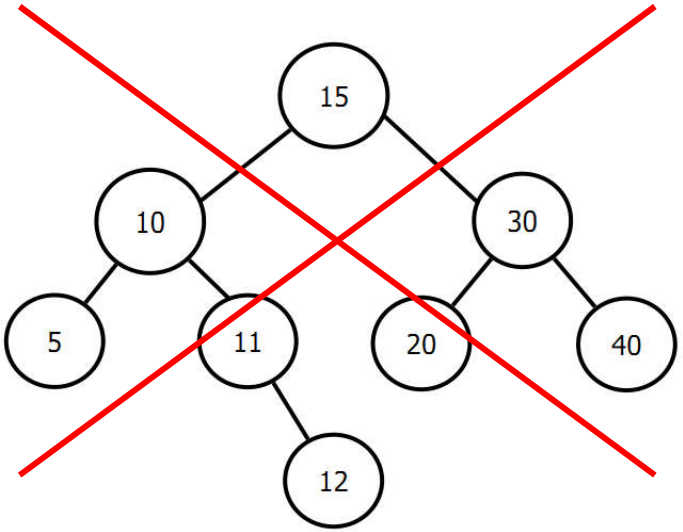
# Árvores Binárias

## Árvore estritamente binária

**É uma árvore onde todos os nós que não são folha possuem dois filhos.**

Ex.:




## Árvore binária completa

**Uma árvore binária completa de profundidade *d* é uma árvore estritamente binária onde todas as folhas estão no nível *d*.**

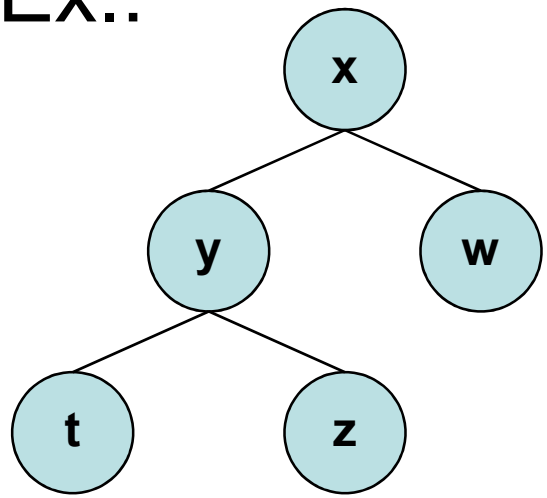
# Árvores Binárias

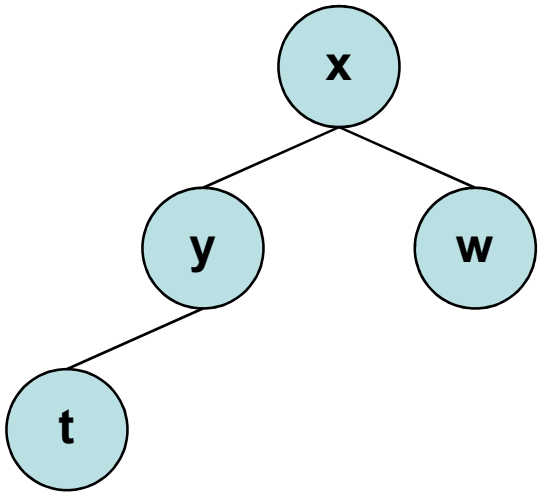
## Árvore binária quase completa

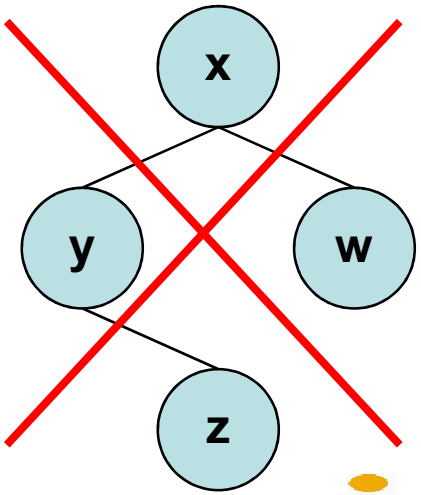
**Uma árvore binária de profundidade *d* será uma árvore binária quase completa se:**

1. **Cada folha da árvore estiver no nível *d* ou no nível *d*-1.**
2. **Para todo nó *nd* que possui um descendente direito no nível *d*, todo descendente esquerdo de *nd* é folha no nível *d* ou tem 2 filhos.**

Ex.:



# Árvores Binárias

Assim como vimos em nosso estudo sobre listas, árvores também podem ser armazenadas de forma estática ou dinâmica.

Começaremos nosso estudo com o armazenamento estático.

Quais campos seriam relevantes para cada nó?

Os campos *info*, *left*, *right* e *father*. Onde estes campos representam respectivamente a informação armazenada no nó, seu filho à esquerda, seu filho à direita e seu pai.



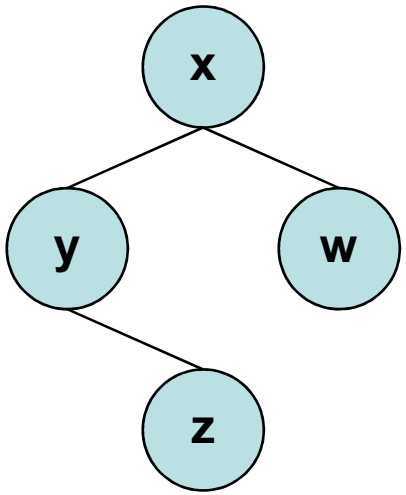
UNIVASF

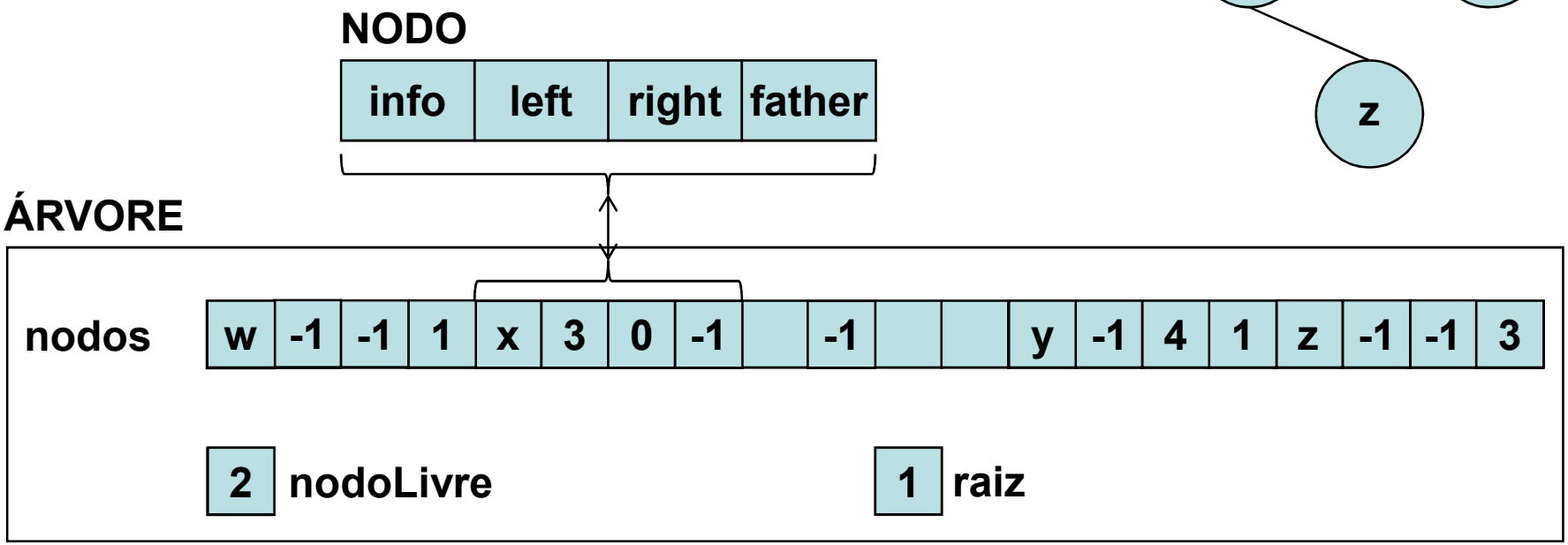
# Árvores Binárias

Como vocês sugeririam o armazenamento estático?

Em um vetor?

Como?



## Árvores Binárias

Quais operações primitivas seriam relevantes?

Existem várias operações primitivas que podem ser aplicadas a uma árvore binária.

Se ***p*** é uma referência para um nó ***nd*** de uma árvore binária associada a ***t***, a função ***info(t, p)*** retorna o conteúdo de ***nd***. As funções ***left(t, p)***, ***right(t, p)***, ***father(t, p)*** e ***brother(t, p)*** retornam referências para o filho esquerdo de ***nd***, filho direito de ***nd***, pai de ***nd*** e irmão de ***nd***, respectivamente. Estas funções retornaram referência inválidas se ***nd*** não tiver filho esquerdo, filho direito, pai ou irmão.

# Árvores Binárias

Temos também as funções lógicas ***isleft(t, p)*** e ***isright(t, p)*** que retornaram o valor ***true*** se ***nd*** for um filho esquerdo ou direito, respectivamente, de algum outro nó na árvore associada a ***t***, e ***false***, caso contrário.

Para construir uma árvore binária, as operações ***maketree***, ***setleft*** e ***setright*** são úteis.

***maketree(t, x)*** cria uma nova árvore associada a ***t*** consistindo num único nó com o campo de informação ***x***.

# Árvores Binárias

***setleft(t, p, x)*** recebe uma referência ***p*** para um nó da árvore binária associada a ***t*** sem filho esquerdo e cria um novo filho esquerdo para o nó referenciado por ***p*** com o campo informação ***x***.

***setright(t, p, x)*** análoga a ***setleft***, exceto pelo fato de que ela cria um filho direito no nó referenciado por ***p***.

Com o que foi definido podemos definir, na linguagem C, uma árvore binária como sendo:



UNIVASF

```
/*Definição a ARV_BIN_SEQ*/
#define NUMNODES 100
typedef struct {
   int info;
   int left;
   int right;
   int father;
} NODE;
typedef struct{
   int root;
   int nodeFree;
   NODE nodes[NUMNODES];
}ARV_BIN_SEQ;
void maketree(ARV_BIN_SEQ *, int);
void setleft(ARV_BIN_SEQ *, int, int);
void setright(ARV_BIN_SEQ *, int, int);
int info(ARV_BIN_SEQ *, int);
int left(ARV_BIN_SEQ *, int);
int right(ARV_BIN_SEQ *, int);
int father(ARV_BIN_SEQ *, int);
int brother(ARV_BIN_SEQ *, int);
int isleft(ARV_BIN_SEQ *, int);
int isright(ARV_BIN_SEQ *, int);
```

# Árvores Binárias - Alocação Sequencial

Com base no que foi visto implemente as operações que compõem o TAD ARV_BIN_SEQ.

```
/*Definição a ARV_BIN_SEQ*/
#define NUMNODES 100
typedef struct {
   int info;
   int left;
   int right;
   int father;
} NODE;
typedef struct{
   int root;
   int nodeFree;
   NODE nodes[NUMNODES];
}ARV_BIN_SEQ;
```

```
void maketree(ARV_BIN_SEQ *, int);
void setleft(ARV_BIN_SEQ *, int, int);
void setright(ARV_BIN_SEQ *, int, int);
int info(ARV_BIN_SEQ *, int);
int left(ARV_BIN_SEQ *, int);
int right(ARV_BIN_SEQ *, int);
int father(ARV_BIN_SEQ *, int);
int brother(ARV_BIN_SEQ *, int);
int isleft(ARV_BIN_SEQ *, int);
int isright(ARV_BIN_SEQ *, int);
```

```
void maketree(ARV_BIN_SEQ *t, int x)
{
   int i, ind;
   for (i=0; i<NUMNODES-1; i++)
      t->nodes[i].left = i+1;
   t->nodes[i].left = -1;
   t->nodeFree=0;
   ind = getNode(t);
   if (ind != -1)
   {
      t->nodes[ind].info = x;
      t->nodes[ind].left = -1;
      t->nodes[ind].right = -1;
      t->nodes[ind].father = -1;
      t->root = ind;
   }
   else
   {
      printf("Impossivel construir a arvore!");
      exit(1);
   }
}
```

```
int getNode(ARV_BIN_SEQ *t)
{
   if (t->nodeFree != -1)
   {
      int i = t->nodeFree;
      t->nodeFree = t->nodes[t->nodeFree].left;
      return i;
   }
   else
      return -1;
}


void freeNode(ARV_BIN_SEQ *t, int node)
{
   t->nodes[node].left = t->nodeFree;
   t->nodeFree = node;
}
```

UNIVASF

# Árvores Binárias - Alocação Sequencial

Com base no que foi visto implemente as operações que compõem o TAD ARV_BIN_SEQ.

```
/*Definição a ARV_BIN_SEQ*/
#define NUMNODES 100
typedef struct {
   int info;
   int left;
   int right;
   int father;
} NODE;
typedef struct{
   int root;
   int nodeFree;
   NODE nodes[NUMNODES];
}ARV_BIN_SEQ;
```

```
void maketree(ARV_BIN_SEQ *, int);
void setleft(ARV_BIN_SEQ *, int, int);
void setright(ARV_BIN_SEQ *, int, int);
int info(ARV_BIN_SEQ *, int);
int left(ARV_BIN_SEQ *, int);
int right(ARV_BIN_SEQ *, int);
int father(ARV_BIN_SEQ *, int);
int brother(ARV_BIN_SEQ *, int);
int isleft(ARV_BIN_SEQ *, int);
int isright(ARV_BIN_SEQ *, int);
```

```
void setleft(ARV_BIN_SEQ *t, int p, int x)
{
   int ind = getNode(t);
   if (ind != -1)
   {
      t->nodes[p].left = ind;
      t->nodes[ind].info = x;
      t->nodes[ind].left = -1;
      t->nodes[ind].right = -1;
      t->nodes[ind].father = p;
   }
   else
   {
      printf("Impossivel inserir filho a esquerda!");
      exit(2);
   }
}
```

# Árvores Binárias - Alocação Sequencial

Com base no que foi visto implemente as operações que compõem o TAD ARV_BIN_SEQ.

```
/*Definição a ARV_BIN_SEQ*/
#define NUMNODES 100
typedef struct {
   int info;
   int left;
   int right;
   int father;
} NODE;
typedef struct{
   int root;
   int nodeFree;
   NODE nodes[NUMNODES];
}ARV_BIN_SEQ;
```

```
void maketree(ARV_BIN_SEQ *, int);
void setleft(ARV_BIN_SEQ *, int, int);
void setright(ARV_BIN_SEQ *, int, int);
int info(ARV_BIN_SEQ *, int);
int left(ARV_BIN_SEQ *, int);
int right(ARV_BIN_SEQ *, int);
int father(ARV_BIN_SEQ *, int);
int brother(ARV_BIN_SEQ *, int);
int isleft(ARV_BIN_SEQ *, int);
int isright(ARV_BIN_SEQ *, int);
```

```
void setright(ARV_BIN_SEQ *t, int p, int x)
{
   int ind = getNode(t);
   if (ind != -1)
   {
      t->nodes[p].right = ind;
      t->nodes[ind].info = x;
      t->nodes[ind].left = -1;
      t->nodes[ind].right = -1;
      t->nodes[ind].father = p;
   }
   else
   {
      printf("Impossivel inserir filho a direita!");
      exit(2);
   }
}
```

# Árvores Binárias - Alocação Sequencial

Com base no que foi visto implemente as operações que compõem o TAD ARV_BIN_SEQ.

```
/*Definição a ARV_BIN_SEQ*/
#define NUMNODES 100
typedef struct {
   int info;
   int left;
   int right;
   int father;
} NODE;
typedef struct{
   int root;
   int nodeFree;
   NODE nodes[NUMNODES];
}ARV_BIN_SEQ;
```

```
void maketree(ARV_BIN_SEQ *, int);
void setleft(ARV_BIN_SEQ *, int, int);
void setright(ARV_BIN_SEQ *, int, int);
int info(ARV_BIN_SEQ *, int);
int left(ARV_BIN_SEQ *, int);
int right(ARV_BIN_SEQ *, int);
int father(ARV_BIN_SEQ *, int);
int brother(ARV_BIN_SEQ *, int);
int isleft(ARV_BIN_SEQ *, int);
int isright(ARV_BIN_SEQ *, int);
```

```
int info(ARV_BIN_SEQ *t, int p)
{
  return t->nodes[p].info;
}

int left(ARV_BIN_SEQ *t, int p)
{
  return t->nodes[p].left;
}

int right(ARV_BIN_SEQ *t, int p)
{
  return t->nodes[p].right;
}

int father(ARV_BIN_SEQ *t, int p)
{
  return t->nodes[p].father;
}
```

UNIVASF

# Árvores Binárias - Alocação Sequencial

Com base no que foi visto implemente as operações que compõem o TAD ARV_BIN_SEQ.

```
/*Definição a ARV_BIN_SEQ*/
#define NUMNODES 100
typedef struct {
   int info;
   int left;
   int right;
   int father;
} NODE;
typedef struct{
   int root;
   int nodeFree;
   NODE nodes[NUMNODES];
}ARV_BIN_SEQ;
```

```
void maketree(ARV_BIN_SEQ *, int);
void setleft(ARV_BIN_SEQ *, int, int);
void setright(ARV_BIN_SEQ *, int, int);
int info(ARV_BIN_SEQ *, int);
int left(ARV_BIN_SEQ *, int);
int right(ARV_BIN_SEQ *, int);
int father(ARV_BIN_SEQ *, int);
int brother(ARV_BIN_SEQ *, int);
int isleft(ARV_BIN_SEQ *, int);
int isright(ARV_BIN_SEQ *, int);
```