

Árvores

Prof^a. Rose Yuri Shimizu

Roteiro

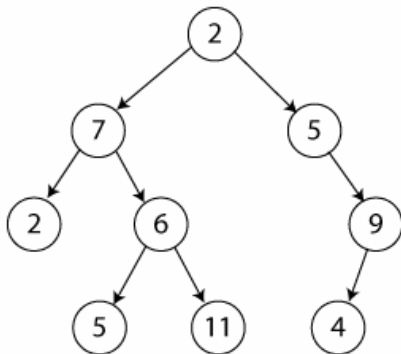
- 1 Árvores
 - Árvore binária

Referências

- <https://www.ic.unicamp.br/~rafael/cursos/2s2018/mc202/slides/unidade17-arvores-binarias.pdf>
- <https://www.inf.ufpr.br/carmem/ci057/apostila.pdf>
- <https://www.ime.usp.br/~pf/algoritmos/aulas/bint.html>
- <https://www.ime.usp.br/~song/mac5710/slides/05tree>
- https://pt.wikipedia.org/wiki/%C3%81rvore_bin%C3%A1ria
- https://ww2.inf.ufg.br/~hebert/disc/aed1/AED1_10_Arvores.pdf

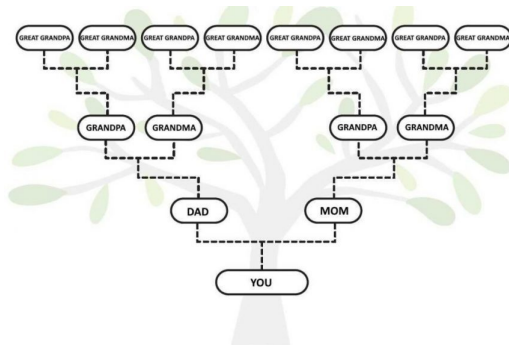
Árvore

- Estrutura não linear com relacionamentos hierárquicos entre os elementos (node, vértices)



Árvores

- Amplamente utilizadas na computação para modelagem de problemas reais: representação computacional
- Representação hierárquica:
 - ▶ Estrutura de organização que favorece consultas eficientes
 - ▶ Cada ramificação possui um conjunto de informações específicas



<https://genealogizando.com.br/arvore-genealogica/afinal-o-que-e-arvore-genealogica/>

Árvores - analisador de expressões

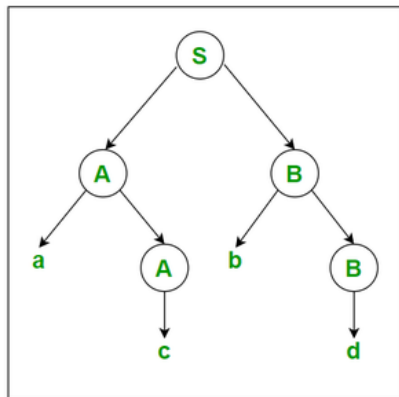
- Regra gramatical (lógica hierárquica): gera-se árvores validam uma entrada
- Exemplo de gramática (regras de produção):

$S := AB$

$A := c \mid aA$

$B := d \mid bB$

- Árvore sintática (parse): resultado da análise sintática
- Dada a entrada "acbd", a árvore de análise sintática gerada:

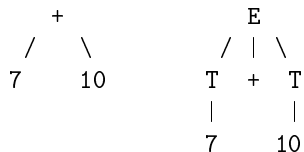


Árvores - analisador de expressões

- Exemplo de gramática (regras de produção):

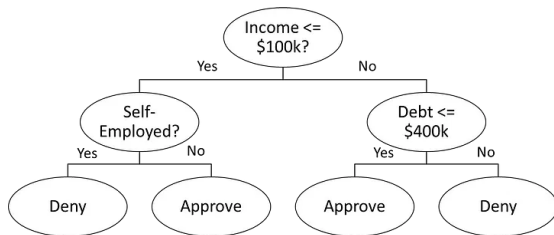
$$E := T + T$$
$$T := \text{numeros}$$

- Dada a entrada "7 + 10", a árvore da análise sintática gerada:

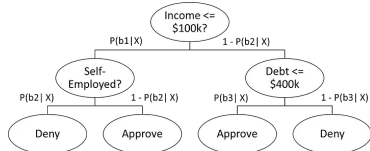


Árvores - aprendizagem de máquina

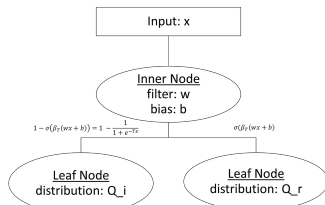
- Árvores de decisão: sequencias de decisões baseadas em uma modelagem hierárquica



Determinística

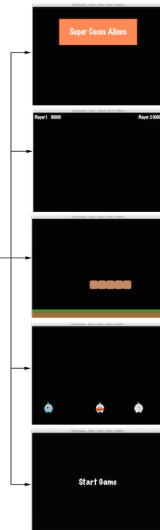
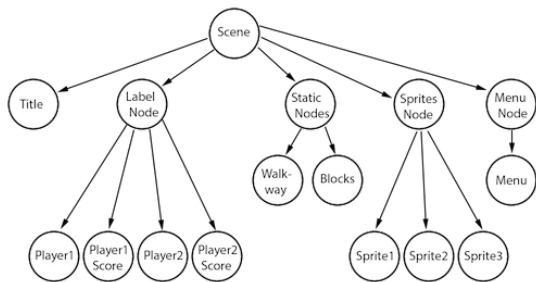


Probabilística (condicional) - Bayes



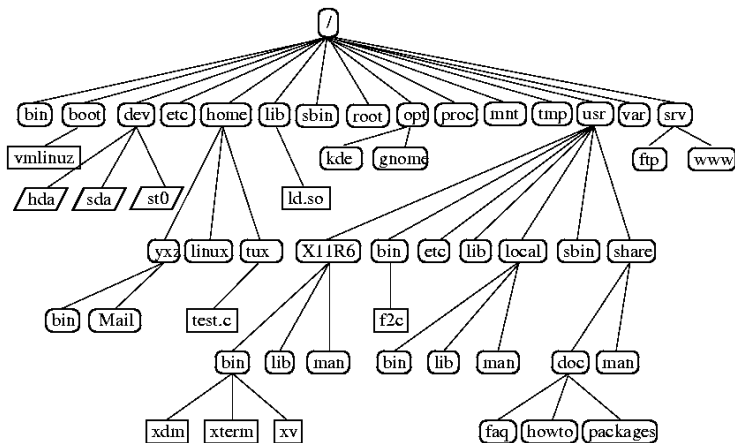
Distribuição de probabilidade - redes neurais

Árvores - Engine



Árvores - sistemas de arquivos

- Árvore de diretórios (Unix-like)

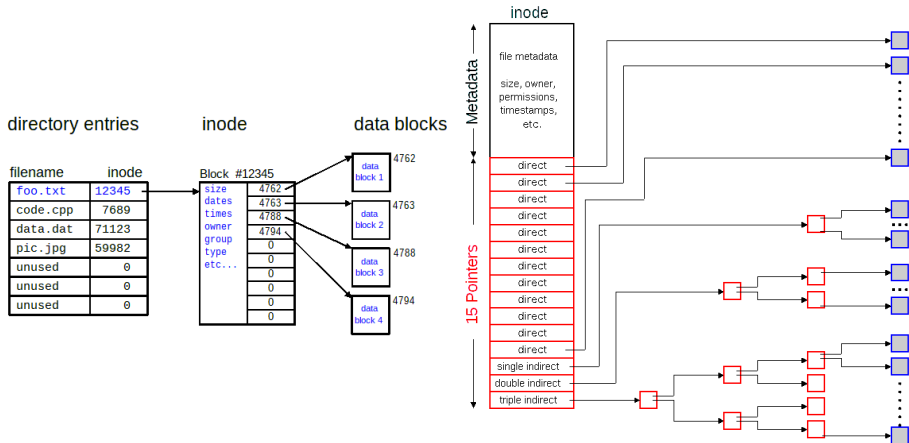


<https://vedslinux.blogspot.com/2013/10/linux-file-system-hierarchy.html>

Árvores - sistemas de arquivos

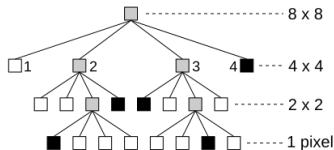
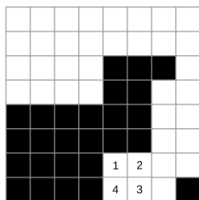
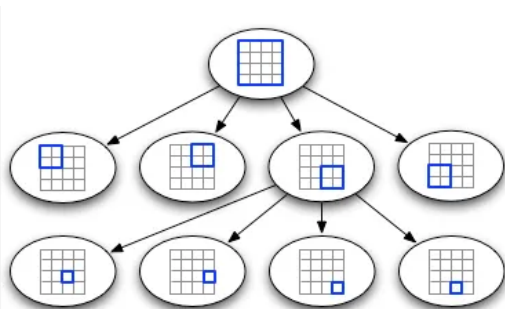
- **Árvore de inodes**

- ▶ Inode (index node): estrutura de dados que descreve um arquivo (Unix-like)
 - ★ Atributos (dados temporais, proprietário, permissões) e endereços de blocos de memória (dados)



<https://azrael.digipen.edu/~mmead/www/Courses/CS180/FileSystems-1.html>

Árvores - representação de imagens

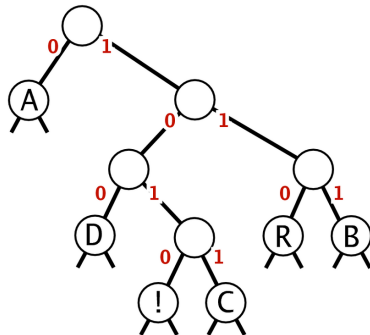


Árvores - compressão de arquivos

Codeword table

| <i>key</i> | <i>value</i> |
|------------|--------------|
| ! | 1010 |
| A | 0 |
| B | 111 |
| C | 1011 |
| D | 100 |
| R | 110 |

Trie representation



A Huffman code

Árvore decodificadora gerada pelo código de Huffman

- ABRACADABRA!: sem compressão $12 \cdot 8 = 96$ bits
- ABRACADABRA!: com compressão 29 bits
- 0 111 110 0 1011 0 100 0 111 110 0 1010

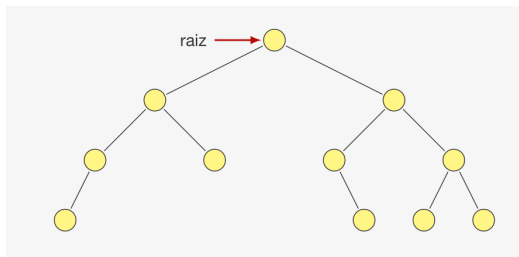
Árvores - Definições

- Árvores

- ▶ Grafo acíclico
- ▶ Conexo/conectado: existe caminho entre quaisquer dois de seus vértices

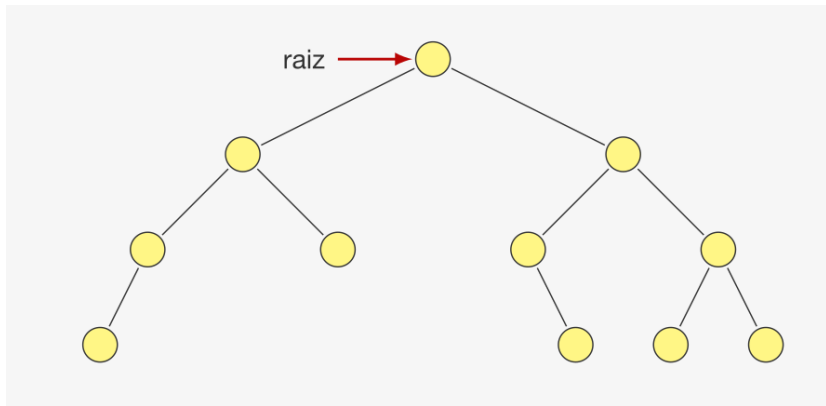
- Árvores enraizadas

- ▶ Possui 1 elemento nó chamado raiz



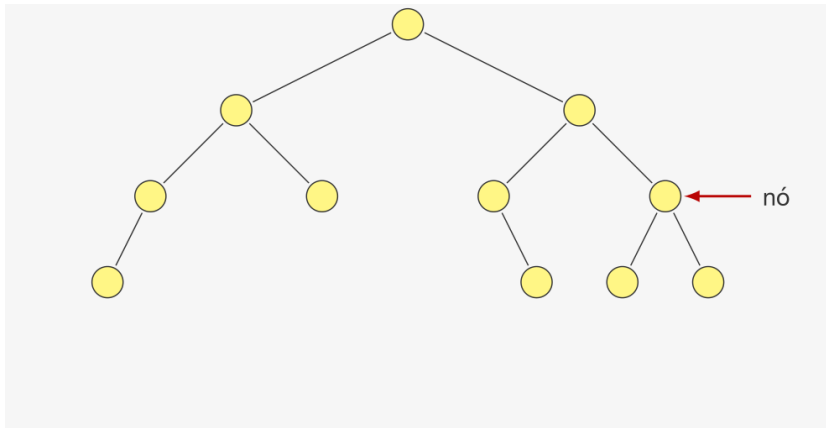
<https://www.ic.unicamp.br/~rafael/cursos/2s2018/mc202/slides/unidade17-arvores-binarias.pdf>

Árvores enraizadas - nós (node)



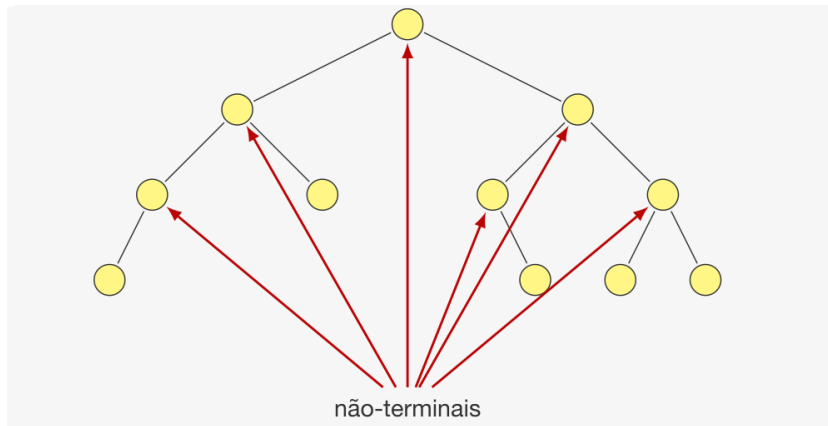
<https://www.ic.unicamp.br/~rafael/cursos/2s2018/mc202/slides/unidade17-arvores-binarias.pdf>

Árvores enraizadas - nós (node)



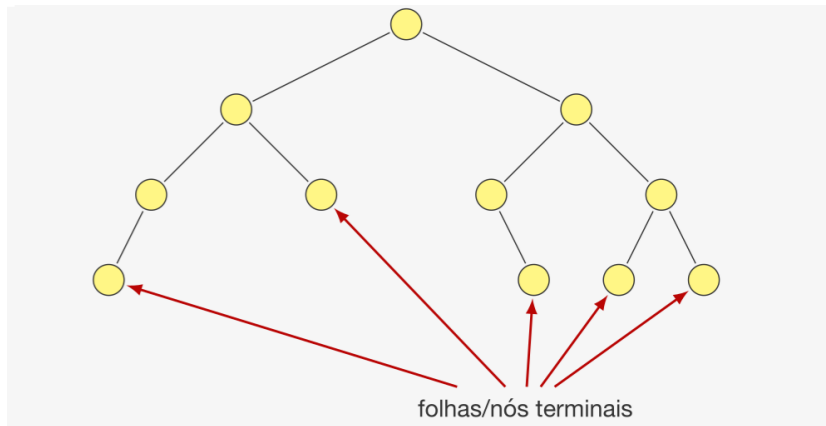
<https://www.ic.unicamp.br/~rafael/cursos/2s2018/mc202/slides/unidade17-arvores-binarias.pdf>

Árvores enraizadas - nós (node)



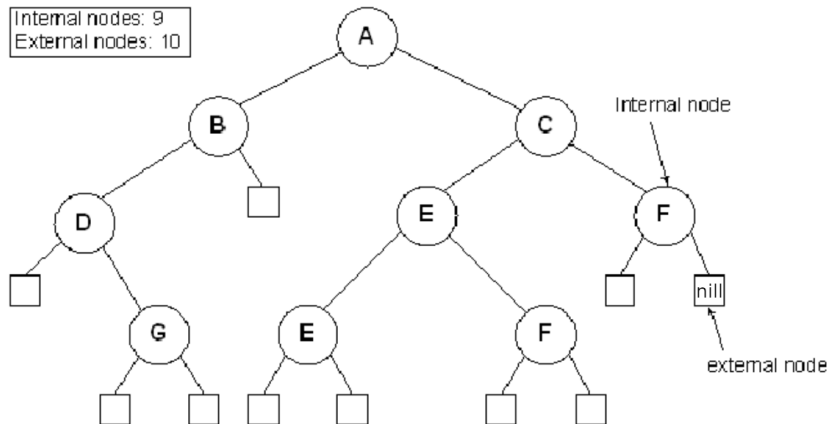
<https://www.ic.unicamp.br/~rafael/cursos/2s2018/mc202/slides/unidade17-arvores-binarias.pdf>

Árvores enraizadas - nós (node)



<https://www.ic.unicamp.br/~rafael/cursos/2s2018/mc202/slides/unidade17-arvores-binarias.pdf>

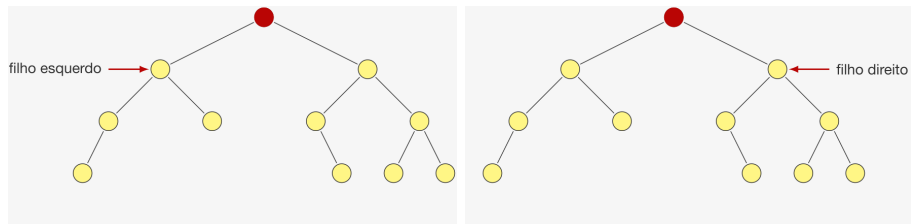
Árvores enraizadas - nós (node)



<https://cs-study.blogspot.com/2012/11/properties-of-binary-tree.html>

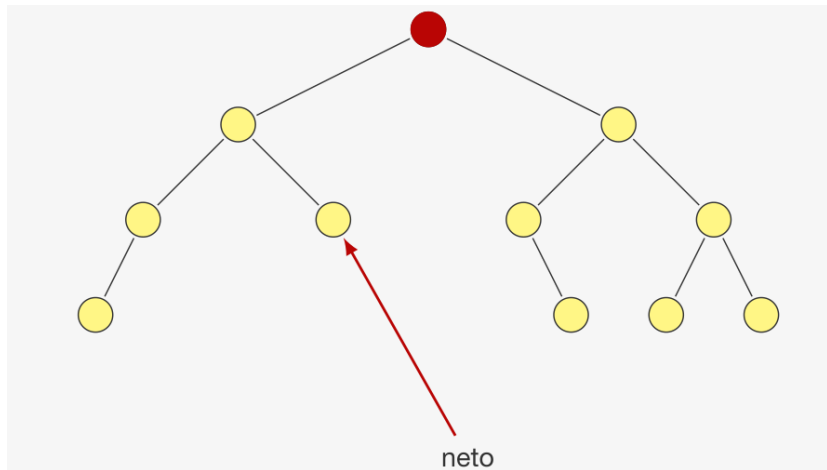
- Nós externos (sem filhos - NULL - NIL - nada/zero)

Árvores enraizadas - Genealogia dos nós



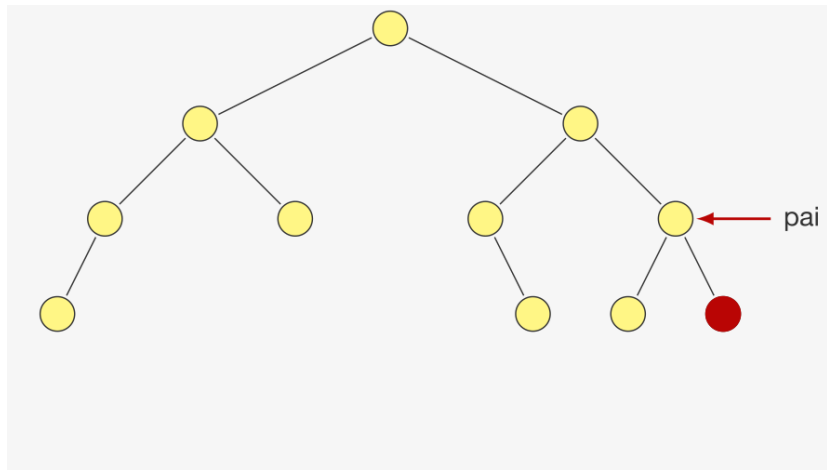
<https://www.ic.unicamp.br/~rafael/cursos/2s2018/mc202/slides/unidade17-arvores-binarias.pdf>

Árvores enraizadas - Genealogia dos nós



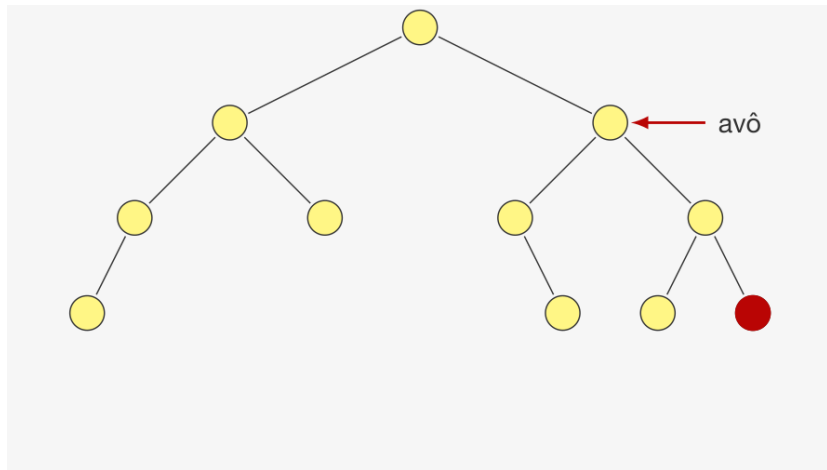
<https://www.ic.unicamp.br/~rafael/cursos/2s2018/mc202/slides/unidade17-arvores-binarias.pdf>

Árvores enraizadas - Genealogia dos nós



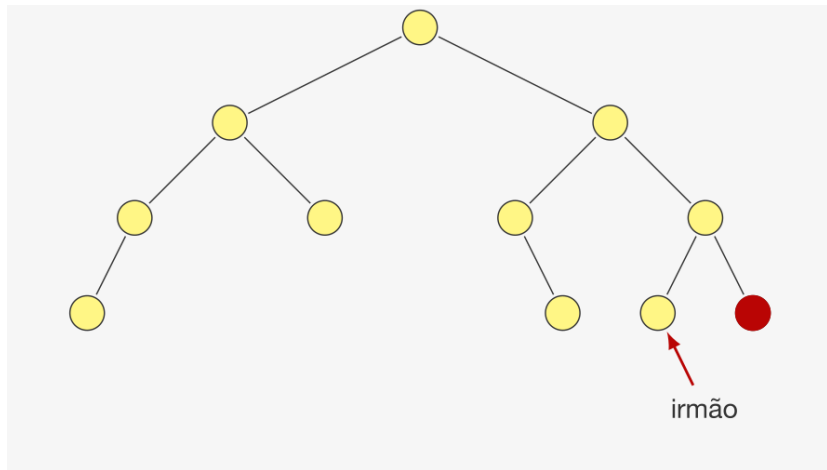
<https://www.ic.unicamp.br/~rafael/cursos/2s2018/mc202/slides/unidade17-arvores-binarias.pdf>

Árvores enraizadas - Genealogia dos nós



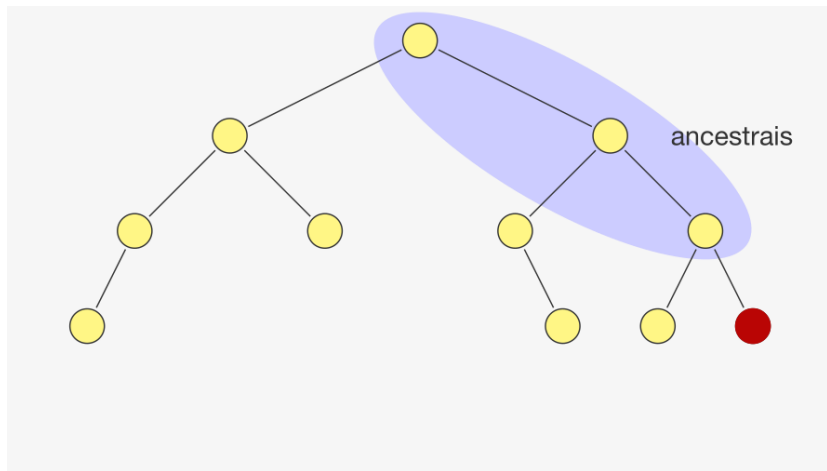
<https://www.ic.unicamp.br/~rafael/cursos/2s2018/mc202/slides/unidade17-arvores-binarias.pdf>

Árvores enraizadas - Genealogia dos nós



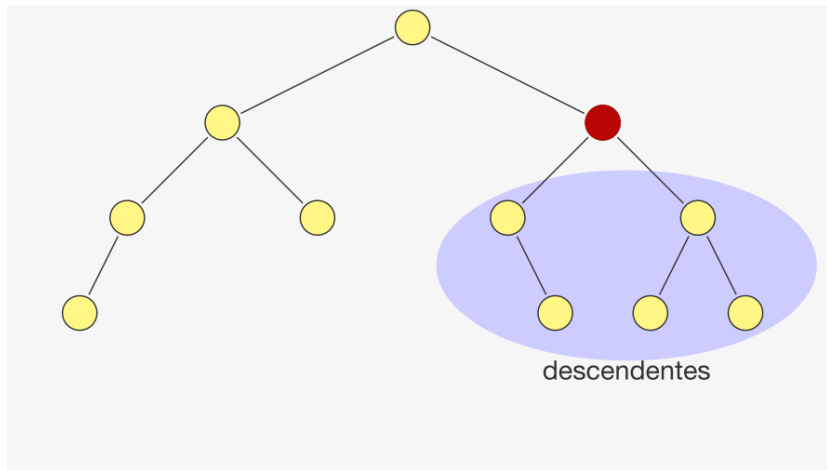
<https://www.ic.unicamp.br/~rafael/cursos/2s2018/mc202/slides/unidade17-arvores-binarias.pdf>

Árvores enraizadas - Genealogia dos nós



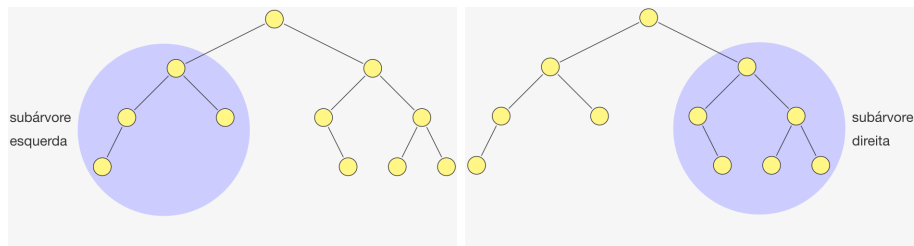
<https://www.ic.unicamp.br/~rafael/cursos/2s2018/mc202/slides/unidade17-arvores-binarias.pdf>

Árvores enraizadas - Genealogia dos nós



<https://www.ic.unicamp.br/~rafael/cursos/2s2018/mc202/slides/unidade17-arvores-binarias.pdf>

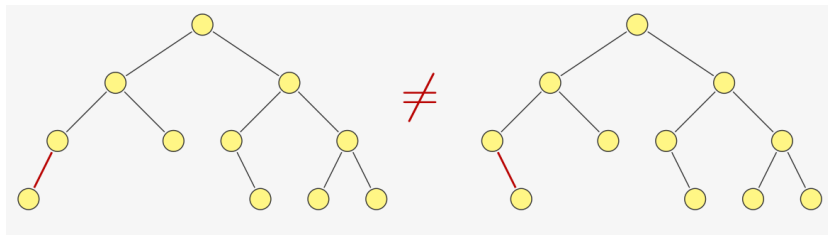
Árvores enraizadas - Sub-árvores



<https://www.ic.unicamp.br/~rafael/cursos/2s2018/mc202/slides/unidade17-arvores-binarias.pdf>

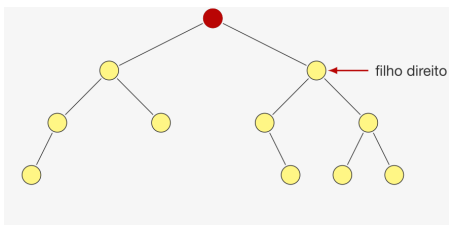
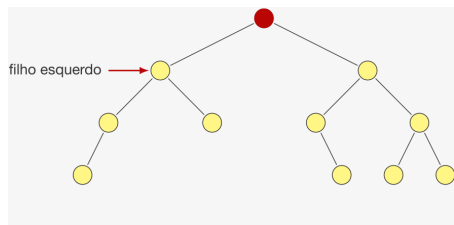
- Subárvore enraizada em x é a árvore composta pelos nós descendentes de x

Árvores enraizadas - Ordem dos nodos



<https://www.ic.unicamp.br/~rafael/cursos/2s2018/mc202/slides/unidade17-arvores-binarias.pdf>

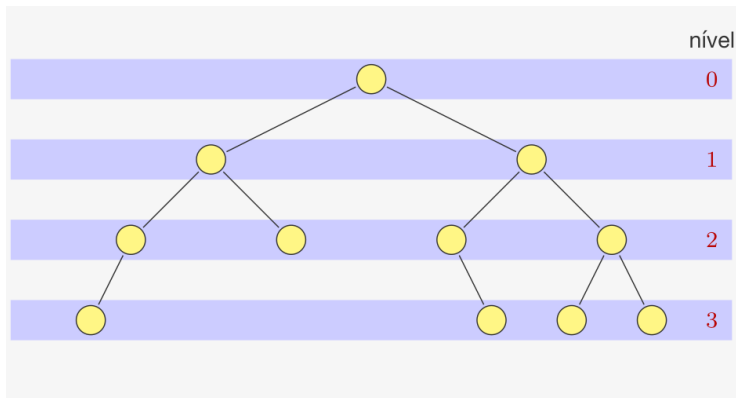
Árvores enraizadas - Grau



<https://www.ic.unicamp.br/~rafael/cursos/2s2018/mc202/slides/unidade17-arvores-binarias.pdf>

- Grau: número de filhos (subárvores) de um nó
- Um nó folha possui grau 0
- Árvores n-árias: quando todos os nós possuem grau máximo "n" (binária, ternária)

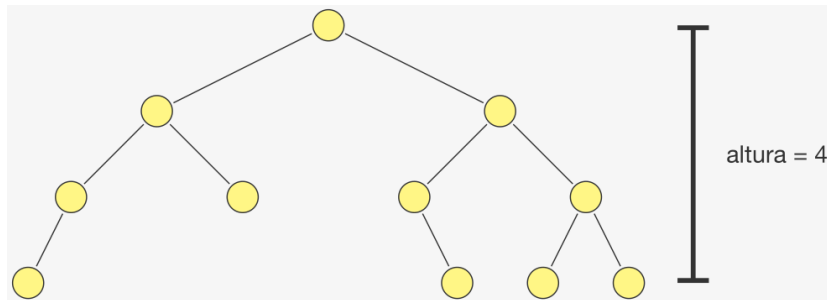
Árvores enraizadas - Profundidade e Nível



<https://www.ic.unicamp.br/~rafael/cursos/2s2018/mc202/slides/unidade17-arvores-binarias.pdf>

- Nível indica a profundidade dos nós
- Profundidade de um nó:
 - ▶ Número de ancestrais: caminho entre desse nó até a raiz
 - ▶ A raiz tem profundidade 0
- Um conjunto de nós com a mesma profundidade estão em um mesmo Nível

Árvores enraizadas - Altura



<https://www.ic.unicamp.br/~rafael/cursos/2s2018/mc202/slides/unidade17-arvores-binarias.pdf>

- Altura: maior profundidade de um nó externo (NULL)

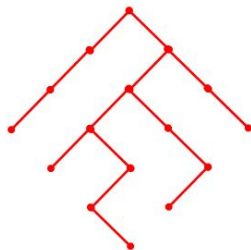
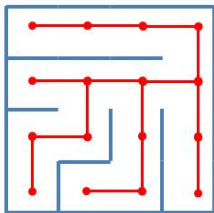
Árvores - Tipos

- Existem diversos tipos de **restrições utilizadas na construção e nas operações** sobre a estrutura de árvore, cada qual realizada com um determinado propósito para **melhor tratamento dos dados em um determinado domínio**
- Dentre esses diferentes tipos de árvores citam-se:
 - ▶ Árvores Binárias, Binárias de Busca
 - ▶ Heaps, fila de propriedades (priority queue)
 - ▶ Árvores AVL, Red-Black (balanceadas)
 - ▶ Árvores Trie e PATRICIA
 - ▶ Árvores B, B+, B*
 - ▶ etc.

- 1 Árvores
 - Árvore binária

Árvores binárias

- Cada nó não tem grau maior que 2
- Formado recursivamente por:
 - ▶ Raiz
 - ▶ Sub-árvores binárias: direita e esquerda, as quais também são árvores binárias
- Base para outros tipos de variações de árvores (Trie, red-black, AVL)

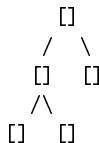


<https://openhome.cc/eGossip/OpenSCAD/SimpleGeneratedMaze.html>

Árvores binárias - Classificação

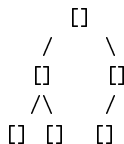
- **Árvore Estritamente Binária**

- ▶ Os nós tem 0 ou 2 filhos



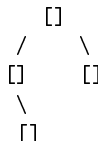
- **Árvore Binária Completa**

- ▶ Todos os nós folhas estão em um mesmo nível



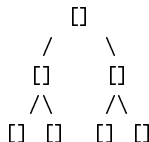
- **Árvore Binária Quase Completa**

- ▶ Todos as folhas estão no nível "d" ou "d-1"

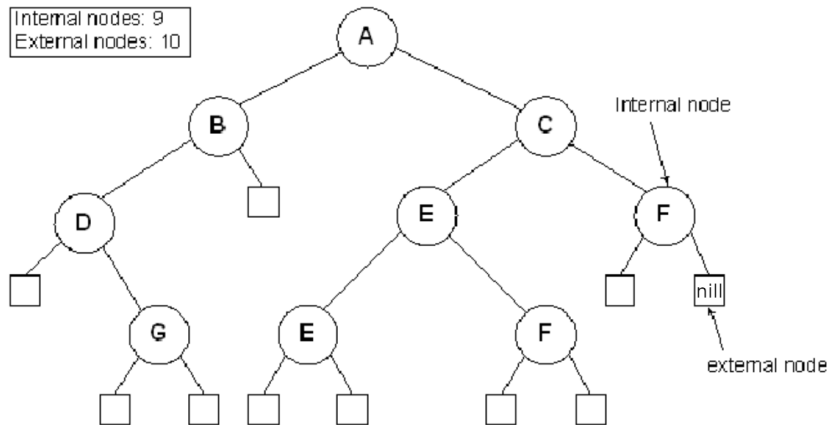


- **Árvore Binária Cheia**

- ▶ Consiste em uma árvore estritamente binária e completa

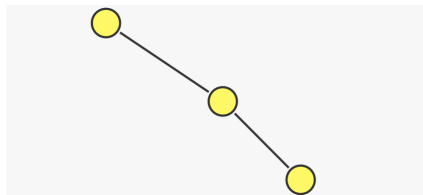


Árvores binárias - Propriedades



- Se há N nós internos, então tem $N + 1$ nós externos

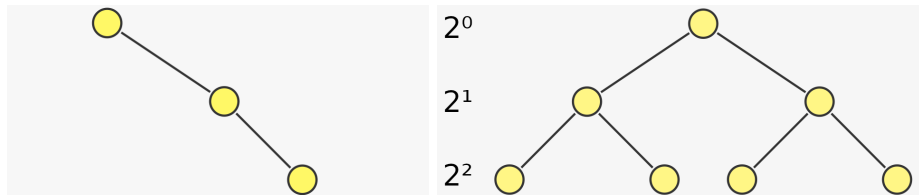
Árvores binárias - Propriedades



<https://www.ic.unicamp.br/~rafael/cursos/2s2018/mc202/slides/unidade17-arvores-binarias.pdf>

- Se a altura é h , então a árvore:
 - ▶ tem no mínimo h nós internos

Árvores binárias - Propriedades

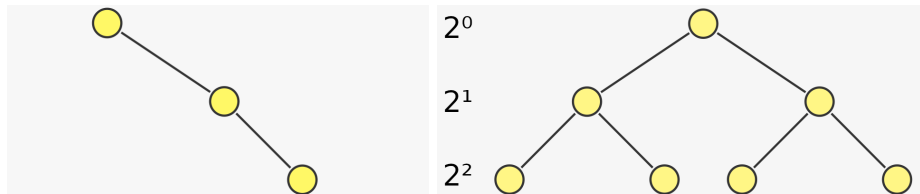


<https://www.ic.unicamp.br/~rafael/cursos/2s2018/mc202/slides/unidade17-arvores-binarias.pdf>

• Se a altura é h , então a árvore:

- ▶ tem no mínimo h nós internos
- ▶ tem no máximo $2^h - 1$ nós internos (árvore binária completa)
 - ★ h vezes: $2^0 + 2^1 + 2^2 + \dots + 2^{h-1}$
 - ★ soma de uma PG $(a_0 * (q^h - 1)) / (q - 1)$
 - ★ $n = \frac{1(2^h - 1)}{(2 - 1)}$
 - ★ $n = 2^h - 1$

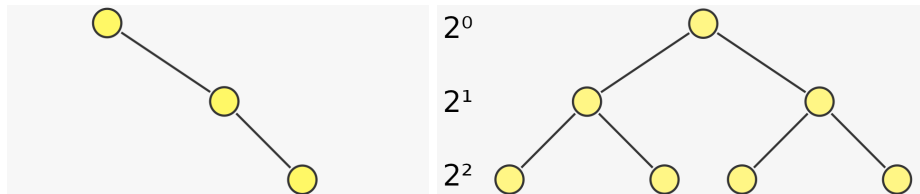
Árvores binárias - Propriedades



<https://www.ic.unicamp.br/~rafael/cursos/2s2018/mc202/slides/unidade17-arvores-binarias.pdf>

- Se a árvore tem $n \geq 1$ nós, então a altura:
 - ▶ é no mínimo $\lceil \lg(n + 1) \rceil$:
 - ★ quando a árvore é completa
 - ★ $n = 2^h - 1$: máximo de nós internos
 - ★ $n + 1 = 2^h$
 - ★ $\lg(n + 1) = \lg(2^h)$
 - ★ $\lg(n + 1) = h \cdot \lg(2)$
 - ★ $\lg(n + 1) = h$

Árvores binárias - Propriedades

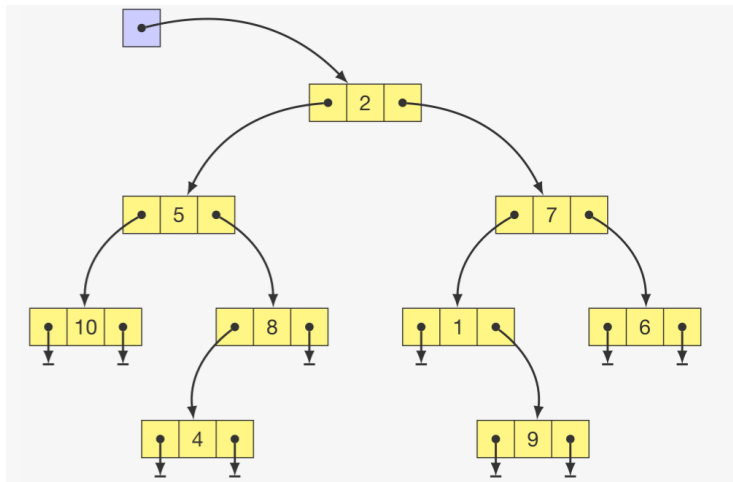


<https://www.ic.unicamp.br/~rafael/cursos/2s2018/mc202/slides/unidade17-arvores-binarias.pdf>

- Se a árvore tem $n \geq 1$ nós, então a altura:
 - ▶ é no mínimo $\lceil \lg(n + 1) \rceil$:
 - ★ quando a árvore é completa
 - ★ $n = 2^h - 1$: máximo de nós internos
 - ★ $n + 1 = 2^h$
 - ★ $\lg(n + 1) = \lg(2^h)$
 - ★ $\lg(n + 1) = h \cdot \lg(2)$
 - ★ $\lg(n + 1) = h$
 - ▶ é no máximo n :
 - ★ quando cada nó não-terminal tem apenas um filho

Árvores binárias - Implementação

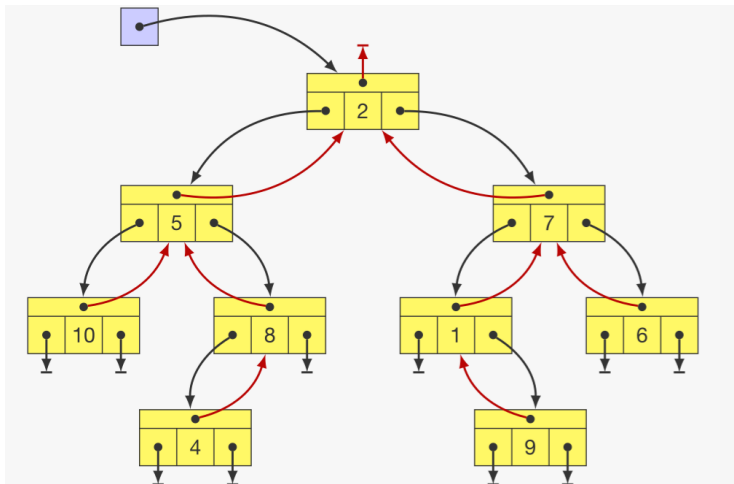
- Lista de adjacências ou matriz de adjacências (aulas anteriores)
- Lista estática ou encadeada



<https://www.ic.unicamp.br/~rafael/cursos/2s2018/mc202/slides/unidade17-arvores-binarias.pdf>

Árvores binárias - Implementação

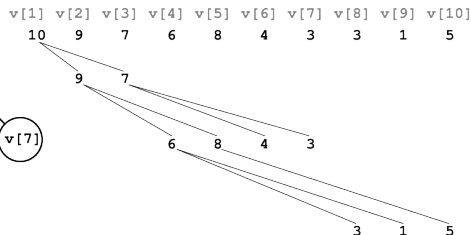
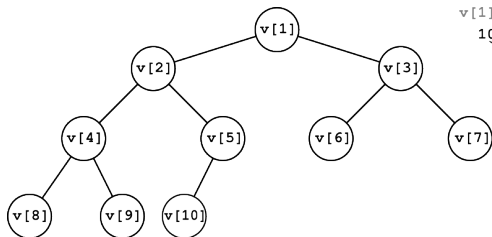
- Lista de adjacências ou matriz de adjacências (aulas anteriores)
- Lista estática ou encadeada



<https://www.ic.unicamp.br/~rafael/cursos/2s2018/mc202/slides/unidade17-arvores-binarias.pdf>

Árvores binárias - Implementação

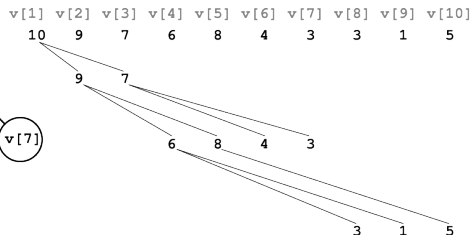
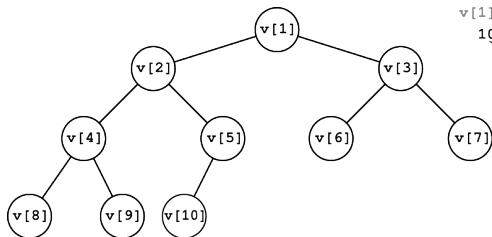
- Lista de adjacências ou matriz de adjacências (aulas anteriores)
- Lista estática ou encadeada



- $v[k]$: pai? filhos?

Árvores binárias - Implementação

- Lista de adjacências ou matriz de adjacências (aulas anteriores)
- Lista estática ou encadeada



- $v[k]$: pai? filhos?
- pai $\rightarrow v[k] : \lfloor \frac{k}{2} \rfloor$
- filhos $\rightarrow v[k] : 2 * k$ e $2 * k + 1$

Árvores binárias - Implementação em C

```
1 typedef int Item;
2
3 typedef struct node no;
4 struct node
5 {
6     Item item;
7     no *pai;
8     no *esq, *dir;
9 };
10
11 no *criar_arvore(Item x, no *p, no *e, no *d)
12 {
13     no *raiz = malloc(sizeof(no));
14     raiz->pai = p;
15     raiz->esq = e;
16     raiz->dir = d;
17     raiz->item = x;
18     return raiz;
19 }
```

Árvores binárias - Implementação em C

```
1 no *avo(no *elemento) {
```

Árvores binárias - Implementação em C

```
1 no *avo(no *elemento) {  
2     if ((elemento != NULL) && (elemento->pai != NULL))
```

Árvores binárias - Implementação em C

```
1 no *avo(no *elemento) {
2     if ((elemento != NULL) && (elemento->pai != NULL))
3         return elemento->pai->pai;
4
5     return NULL;
6 }
7
8 no *tio(no *elemento) {
```


Árvores binárias - Implementação em C

```
1 no *avo(no *elemento) {
2     if ((elemento != NULL) && (elemento->pai != NULL))
3         return elemento->pai->pai;
4
5     return NULL;
6 }
7
8 no *tio(no *elemento) {
9     no *vo = avo(elemento);
10    if (vo == NULL) return NULL;
11
12    if (elemento->pai == vo->esquerda)
```

Árvores binárias - Implementação em C

```
1 no *avo(no *elemento) {
2     if ((elemento != NULL) && (elemento->pai != NULL))
3         return elemento->pai->pai;
4
5     return NULL;
6 }
7
8 no *tio(no *elemento) {
9     no *vo = avo(elemento);
10    if (vo == NULL) return NULL;
11
12    if (elemento->pai == vo->esquerda)
13        return vo->direita;
14
15    return vo->esquerda;
16 }
17
18 no *irmao(no *elemento) {
```

Árvores binárias - Implementação em C

```
1 no *avo(no *elemento) {
2     if ((elemento != NULL) && (elemento->pai != NULL))
3         return elemento->pai->pai;
4
5     return NULL;
6 }
7
8 no *tio(no *elemento) {
9     no *vo = avo(elemento);
10    if (vo == NULL) return NULL;
11
12    if (elemento->pai == vo->esquerda)
13        return vo->direita;
14
15    return vo->esquerda;
16 }
17
18 no *irmao(no *elemento) {
19    if ((elemento != NULL) && (elemento->pai != NULL)){
```

Árvores binárias - Implementação em C

```
1 no *avo(no *elemento) {
2     if ((elemento != NULL) && (elemento->pai != NULL))
3         return elemento->pai->pai;
4
5     return NULL;
6 }
7
8 no *tio(no *elemento) {
9     no *vo = avo(elemento);
10    if (vo == NULL) return NULL;
11
12    if (elemento->pai == vo->esquerda)
13        return vo->direita;
14
15    return vo->esquerda;
16 }
17
18 no *irmao(no *elemento) {
19    if ((elemento != NULL) && (elemento->pai != NULL)){
20        if (elemento == elemento->pai->esquerda)
```

Árvores binárias - Implementação em C

```
1 no *avo(no *elemento) {
2     if ((elemento != NULL) && (elemento->pai != NULL))
3         return elemento->pai->pai;
4
5     return NULL;
6 }
7
8 no *tio(no *elemento) {
9     no *vo = avo(elemento);
10    if (vo == NULL) return NULL;
11
12    if (elemento->pai == vo->esquerda)
13        return vo->direita;
14
15    return vo->esquerda;
16 }
17
18 no *irmao(no *elemento) {
19    if ((elemento != NULL) && (elemento->pai != NULL)){
20        if (elemento == elemento->pai->esquerda)
21            return pai->direita;
22
23        return pai->esquerda;
24    }
25 }
```

Árvores binárias - Implementação em C

```
1 no *busca_linear(no *elemento, Item v) {
```

Árvores binárias - Implementação em C

```
1 no *busca_linear(no *elemento, Item v) {  
2     if(elemento == NULL || elemento->item == v)  
3         return elemento;
```

Árvores binárias - Implementação em C

```
1 no *busca_linear(no *elemento, Item v) {
2     if(elemento == NULL || elemento->item == v)
3         return elemento;
4
5     no *e = busca_linear(elemento->esq, v);
6     if(e != NULL) return e;
```


Árvores binárias - Implementação em C

```
1 no *busca_linear(no *elemento, Item v) {
2     if(elemento == NULL || elemento->item == v)
3         return elemento;
4
5     no *e = busca_linear(elemento->esq, v);
6     if(e != NULL) return e;
7
8     return busca_linear(elemento->dir, v);
9 }
10
11 int numero_nos(no *raiz) {
```

Árvores binárias - Implementação em C

```
1 no *busca_linear(no *elemento, Item v) {
2     if(elemento == NULL || elemento->item == v)
3         return elemento;
4
5     no *e = busca_linear(elemento->esq, v);
6     if(e != NULL) return e;
7
8     return busca_linear(elemento->dir, v);
9 }
10
11 int numero_nos(no *raiz) {
12     if (raiz == NULL) return 0;
```

Árvores binárias - Implementação em C

```
1 no *busca_linear(no *elemento, Item v) {
2     if(elemento == NULL || elemento->item == v)
3         return elemento;
4
5     no *e = busca_linear(elemento->esq, v);
6     if(e != NULL) return e;
7
8     return busca_linear(elemento->dir, v);
9 }
10
11 int numero_nos(no *raiz) {
12     if (raiz == NULL) return 0;
13     return 1 +
```

Árvores binárias - Implementação em C

```
1 no *busca_linear(no *elemento, Item v) {
2     if(elemento == NULL || elemento->item == v)
3         return elemento;
4
5     no *e = busca_linear(elemento->esq, v);
6     if(e != NULL) return e;
7
8     return busca_linear(elemento->dir, v);
9 }
10
11 int numero_nos(no *raiz) {
12     if (raiz == NULL) return 0;
13     return 1 + numero_nos(raiz->esq) +
```

Árvores binárias - Implementação em C

```
1 no *busca_linear(no *elemento, Item v) {
2     if(elemento == NULL || elemento->item == v)
3         return elemento;
4
5     no *e = busca_linear(elemento->esq, v);
6     if(e != NULL) return e;
7
8     return busca_linear(elemento->dir, v);
9 }
10
11 int numero_nos(no *raiz) {
12     if (raiz == NULL) return 0;
13     return 1 + numero_nos(raiz->esq) + numero_nos(raiz->dir);
14 }
15
16 int altura(no *raiz) {
17     if (raiz == NULL) return 0;
```

Árvores binárias - Implementação em C

```
1 no *busca_linear(no *elemento, Item v) {
2     if(elemento == NULL || elemento->item == v)
3         return elemento;
4
5     no *e = busca_linear(elemento->esq, v);
6     if(e != NULL) return e;
7
8     return busca_linear(elemento->dir, v);
9 }
10
11 int numero_nos(no *raiz) {
12     if (raiz == NULL) return 0;
13     return 1 + numero_nos(raiz->esq) + numero_nos(raiz->dir);
14 }
15
16 int altura(no *raiz) {
17     if (raiz == NULL) return 0;
18
19     int h_esq = altura(raiz->esq);
```

Árvores binárias - Implementação em C

```
1 no *busca_linear(no *elemento, Item v) {
2     if(elemento == NULL || elemento->item == v)
3         return elemento;
4
5     no *e = busca_linear(elemento->esq, v);
6     if(e != NULL) return e;
7
8     return busca_linear(elemento->dir, v);
9 }
10
11 int numero_nos(no *raiz) {
12     if (raiz == NULL) return 0;
13     return 1 + numero_nos(raiz->esq) + numero_nos(raiz->dir);
14 }
15
16 int altura(no *raiz) {
17     if (raiz == NULL) return 0;
18
19     int h_esq = altura(raiz->esq);
20     int h_dir = altura(raiz->dir);
```

Árvores binárias - Implementação em C

```
1 no *busca_linear(no *elemento, Item v) {
2     if(elemento == NULL || elemento->item == v)
3         return elemento;
4
5     no *e = busca_linear(elemento->esq, v);
6     if(e != NULL) return e;
7
8     return busca_linear(elemento->dir, v);
9 }
10
11 int numero_nos(no *raiz) {
12     if (raiz == NULL) return 0;
13     return 1 + numero_nos(raiz->esq) + numero_nos(raiz->dir);
14 }
15
16 int altura(no *raiz) {
17     if (raiz == NULL) return 0;
18
19     int h_esq = altura(raiz->esq);
20     int h_dir = altura(raiz->dir);
21
22     return 1 + (h_esq > h_dir ? h_esq : h_dir);
23 }
```

● Exercício: faça versões sem recursão dos algoritmos acima. (estrutura ?)

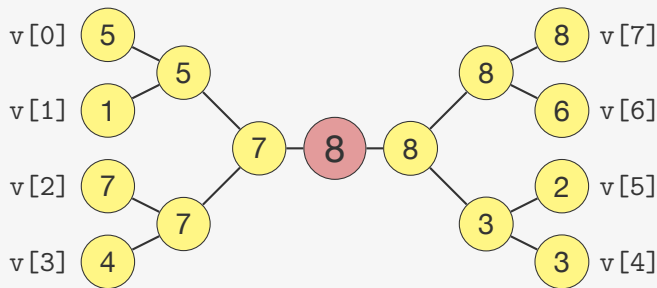
Árvores binárias - Exemplo

- Criar um torneio: representar em árvore
- Montar as chaves: quartas de final, semifinal, final e vencedor
- Slides a seguir:
 - ▶ <https://www.ic.unicamp.br/~rafael/cursos/2s2018/mc202/slides/unidade17-arvores-binarias.pdf>

Exemplo: Criando um torneio

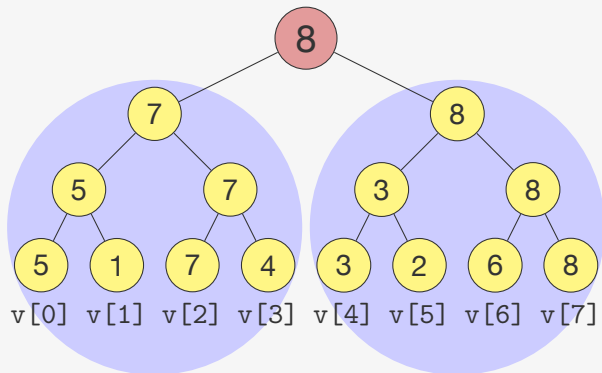
Dado um vetor v com n números, queremos criar um torneio

- Decidir qual é o maior número em um esquema de chaves
 - Ex.: para $n = 8$, temos quartas de final, semifinal e final



É uma árvore binária, onde o valor do pai é o maior valor dos seus filhos

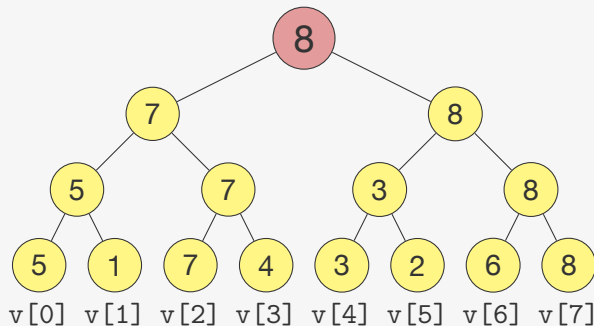
Exemplo: Criando um torneio



Para resolver o torneio:

- resolva o torneio das duas subárvores recursivamente

Exemplo: Criando um torneio



Para resolver o torneio:

- resolva o torneio das duas subárvores recursivamente
- decida o vencedor

Árvores binárias - Exemplo

```
1 no *torneio(int *v, int l, int r) {
2     if (l == r) return criar_arvore(v[l], NULL , NULL);
3
4     int m = (l + r) / 2;
5     no *esq = torneio(v, l, m);
6     no *dir = torneio(v, m+1, r);
7
8     int valor = esq->dado > dir->dado ? esq->dado : dir->dado;
9
10    return criar_arvore(valor , esq , dir);
11
12 }
```

- Com a resolução/representação do torneio, várias informações podem ser obtidas:
 - ▶ Vencedor
 - ▶ Pontuação dos times
 - ▶ Quais times se enfrentaram
 - ▶ Jogos das fases (níveis da árvore)

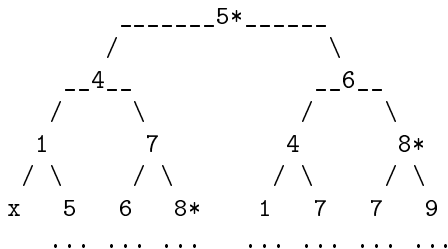
Árvores binárias - exemplo

Problema dos portais - lista de adjacências

Entrada: 10 5 8 4 2 1 3 1 2 1 1 1 2

Saída: 2

1 2 3 4 5 6 7 8 9 10
x[] = {?, 4, 2, 1, 3, 1, 2, 1, 1, 1, 2}



```

1 head *lista = criar_lista(); //tad?
2 inserir(lista, partida);      //posição?
3 distancia[partida] = 0;
4
5 while(!vazia(lista) && distancia[destino] == infinito){
6     Item p = remover(lista); //posição?
7
8     //percorrendo a árvore para resolver o problema
9
10    //subárvore esquerda
11    int passado = p-saltos[p];
12    if(distancia[passado]==infinito) {
13        distancia[passado] = distancia[p] + 1;
14        inserir(lista, passado);
15    }
16
17    //subárvore direita
18    int futuro = p+saltos[p];
19    if(distancia[futuro]==infinito) {
20        distancia[futuro] = distancia[p] + 1;
21        inserir(lista, futuro);
22    }
23 }
24
25 if(distancia[destino] < infinito)
26     printf("%d\n", distancia[destino]);

```

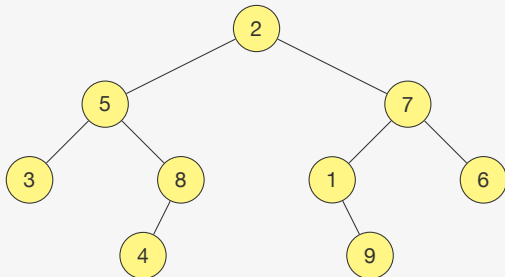
Árvores binárias - Algoritmos de Percurso

- Diferentes diferentes percursos, geram diferentes sequencias de nós (informações)
- Percurso em largura (aulas anteriores)
 - ▶ Percurso por nível: os nós são visitados na ordem dos níveis em que se encontram
 - ▶ Inicialmente são visitados todos os nós do nível 0 (raiz), depois os nós do nível 1, depois do nível 2 e assim por diante
- Percurso em profundidade
 - ▶ Em-Ordem: esquerda raiz/pai direita (ex. notação normal)
 - ▶ Pré-Ordem: raiz/pai esquerda direita (ex. notação polonesa)
 - ▶ Pós-Ordem: esquerda direita raiz/pai (ex. notação polonesa inversa)
- Slides a seguir:
 - ▶ <https://www.ic.unicamp.br/~rafael/cursos/2s2018/mc202/slides/unidade17-arvores-binarias.pdf>

Percorrendo os nós - Pré-ordem

A pré-ordem

- primeiro visita (processa) a raiz
- depois a subárvore esquerda
- depois a subárvore direita

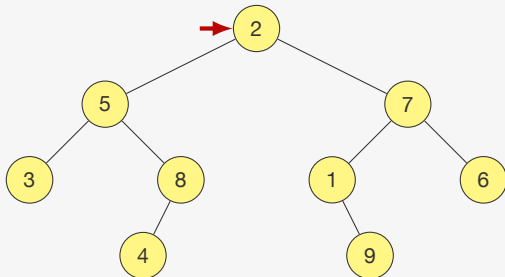


Ex:

Percorrendo os nós - Pré-ordem

A pré-ordem

- primeiro visita (processa) a raiz
- depois a subárvore esquerda
- depois a subárvore direita

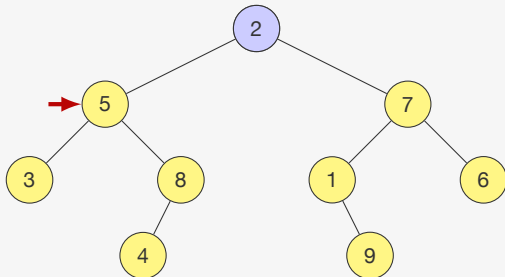


Ex:

Percorrendo os nós - Pré-ordem

A pré-ordem

- primeiro visita (processa) a raiz
- depois a subárvore esquerda
- depois a subárvore direita

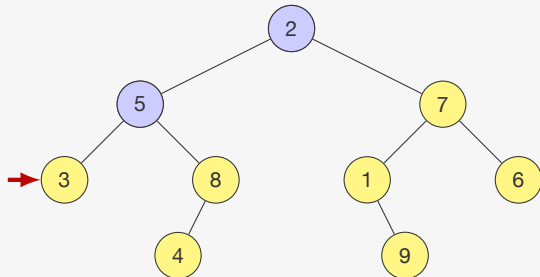


Ex: 2,

Percorrendo os nós - Pré-ordem

A pré-ordem

- primeiro visita (processa) a raiz
- depois a subárvore esquerda
- depois a subárvore direita

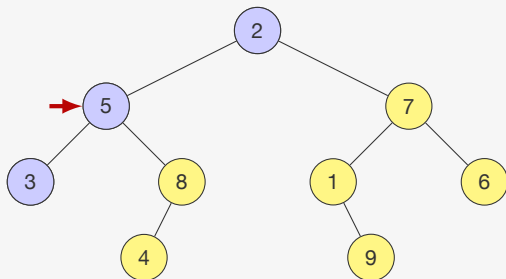


Ex: 2, 5,

Percorrendo os nós - Pré-ordem

A pré-ordem

- primeiro visita (processa) a raiz
- depois a subárvore esquerda
- depois a subárvore direita

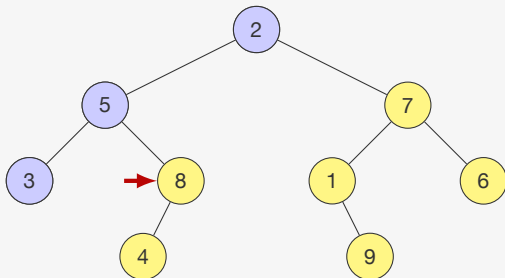


Ex: 2, 5, 3,

Percorrendo os nós - Pré-ordem

A pré-ordem

- primeiro visita (processa) a raiz
- depois a subárvore esquerda
- depois a subárvore direita

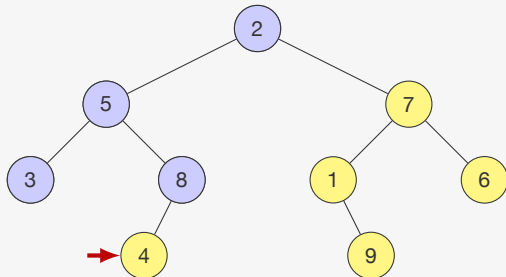


Ex: 2, 5, 3,

Percorrendo os nós - Pré-ordem

A pré-ordem

- primeiro visita (processa) a raiz
- depois a subárvore esquerda
- depois a subárvore direita

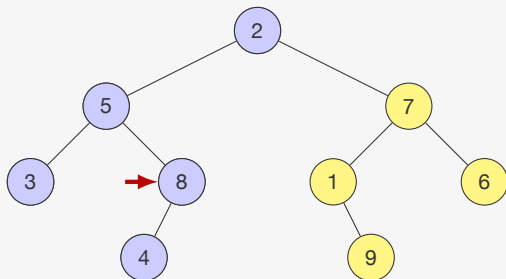


Ex: 2, 5, 3, 8,

Percorrendo os nós - Pré-ordem

A pré-ordem

- primeiro visita (processa) a raiz
- depois a subárvore esquerda
- depois a subárvore direita

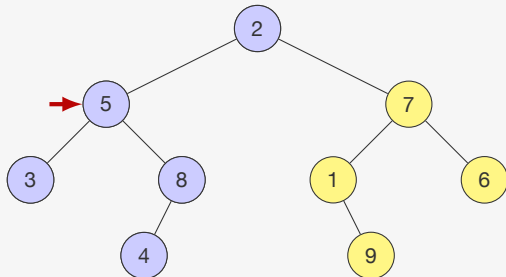


Ex: 2, 5, 3, 8, 4,

Percorrendo os nós - Pré-ordem

A pré-ordem

- primeiro visita (processa) a raiz
- depois a subárvore esquerda
- depois a subárvore direita

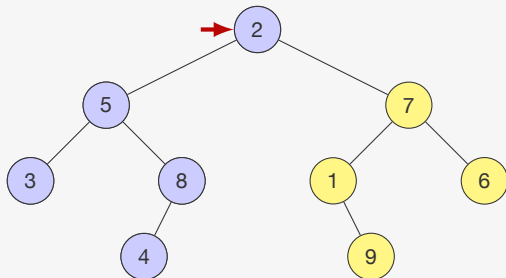


Ex: 2, 5, 3, 8, 4,

Percorrendo os nós - Pré-ordem

A pré-ordem

- primeiro visita (processa) a raiz
- depois a subárvore esquerda
- depois a subárvore direita

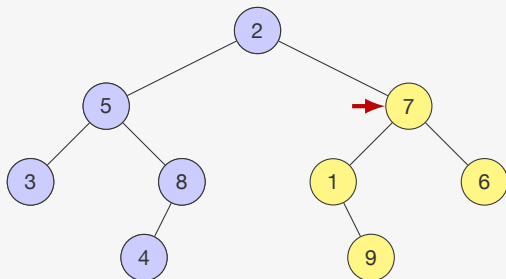


Ex: 2, 5, 3, 8, 4,

Percorrendo os nós - Pré-ordem

A pré-ordem

- primeiro visita (processa) a raiz
- depois a subárvore esquerda
- depois a subárvore direita

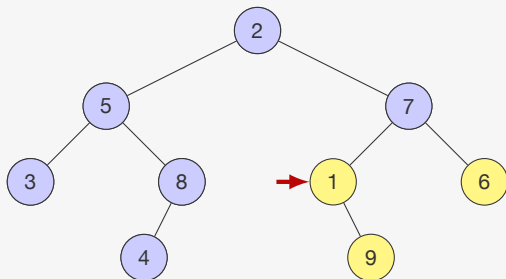


Ex: 2, 5, 3, 8, 4,

Percorrendo os nós - Pré-ordem

A pré-ordem

- primeiro visita (processa) a raiz
- depois a subárvore esquerda
- depois a subárvore direita

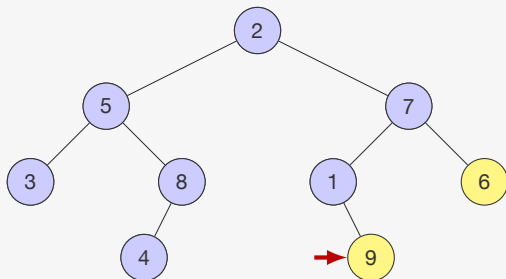


Ex: 2, 5, 3, 8, 4, 7,

Percorrendo os nós - Pré-ordem

A pré-ordem

- primeiro visita (processa) a raiz
- depois a subárvore esquerda
- depois a subárvore direita

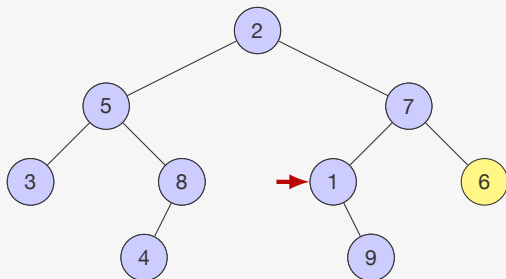


Ex: 2, 5, 3, 8, 4, 7, 1,

Percorrendo os nós - Pré-ordem

A pré-ordem

- primeiro visita (processa) a raiz
- depois a subárvore esquerda
- depois a subárvore direita

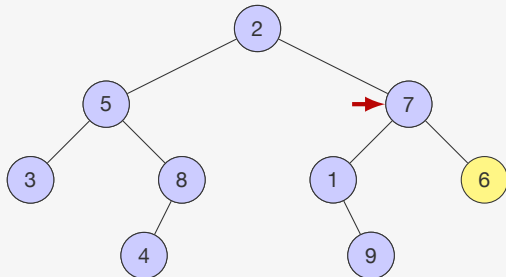


Ex: 2, 5, 3, 8, 4, 7, 1, 9,

Percorrendo os nós - Pré-ordem

A pré-ordem

- primeiro visita (processa) a raiz
- depois a subárvore esquerda
- depois a subárvore direita

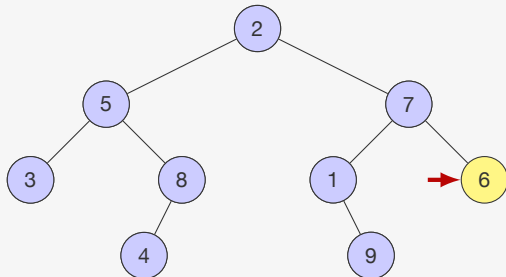


Ex: 2, 5, 3, 8, 4, 7, 1, 9,

Percorrendo os nós - Pré-ordem

A pré-ordem

- primeiro visita (processa) a raiz
- depois a subárvore esquerda
- depois a subárvore direita

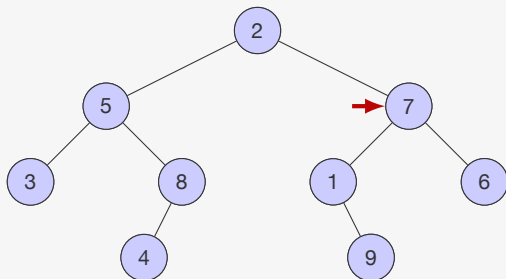


Ex: 2, 5, 3, 8, 4, 7, 1, 9,

Percorrendo os nós - Pré-ordem

A pré-ordem

- primeiro visita (processa) a raiz
- depois a subárvore esquerda
- depois a subárvore direita

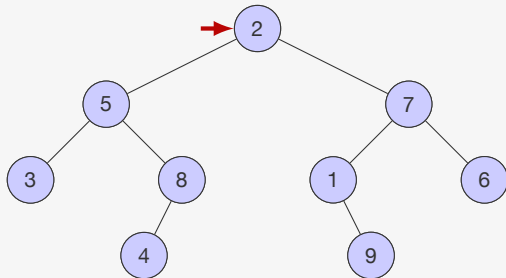


Ex: 2, 5, 3, 8, 4, 7, 1, 9, 6

Percorrendo os nós - Pré-ordem

A pré-ordem

- primeiro visita (processa) a raiz
- depois a subárvore esquerda
- depois a subárvore direita

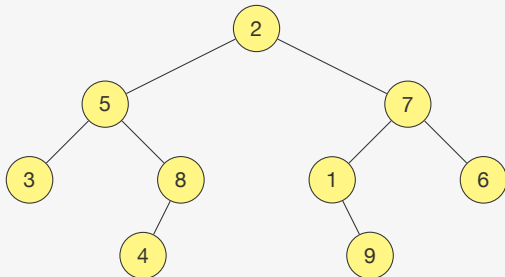


Ex: 2, 5, 3, 8, 4, 7, 1, 9, 6

Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro visita a subárvore esquerda
- depois a subárvore direita
- e por último visita a raiz

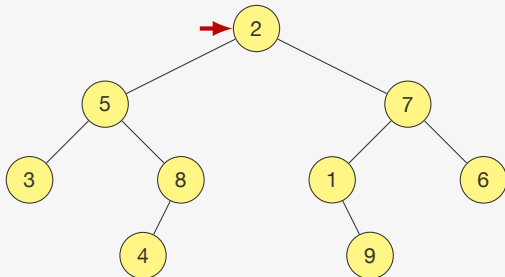


Ex:

Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro visita a subárvore esquerda
- depois a subárvore direita
- e por último visita a raiz

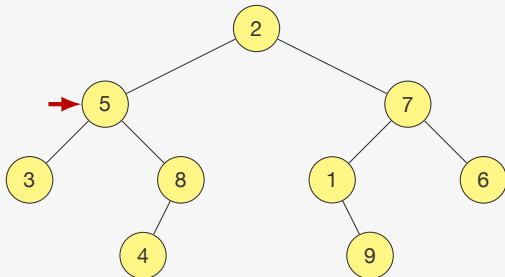


Ex:

Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro visita a subárvore esquerda
- depois a subárvore direita
- e por último visita a raiz

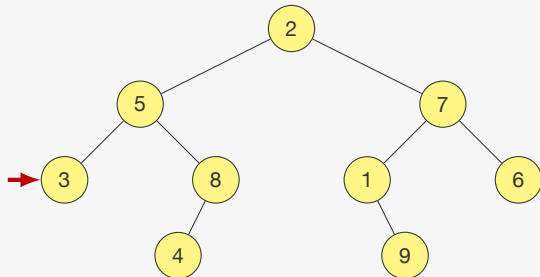


Ex:

Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro visita a subárvore esquerda
- depois a subárvore direita
- e por último visita a raiz

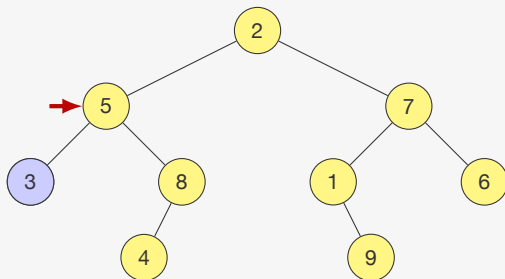


Ex:

Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro visita a subárvore esquerda
- depois a subárvore direita
- e por último visita a raiz

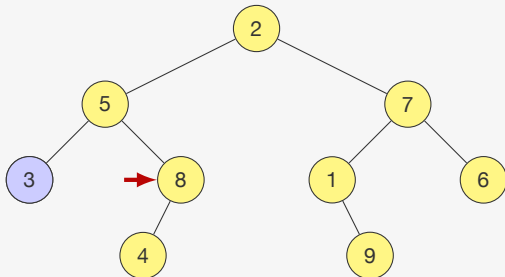


Ex: 3,

Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro visita a subárvore esquerda
- depois a subárvore direita
- e por último visita a raiz

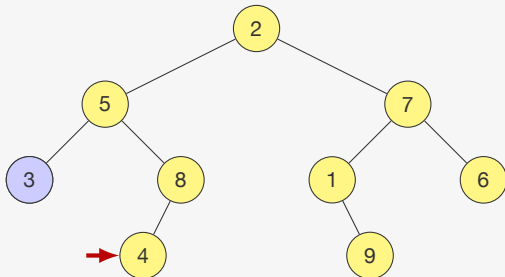


Ex: 3,

Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro visita a subárvore esquerda
- depois a subárvore direita
- e por último visita a raiz

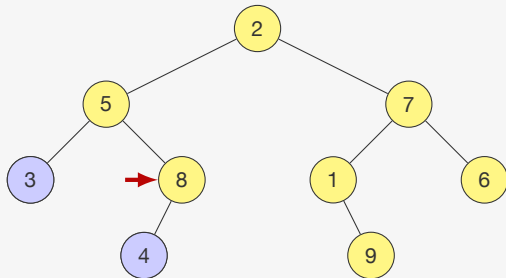


Ex: 3,

Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro visita a subárvore esquerda
- depois a subárvore direita
- e por último visita a raiz

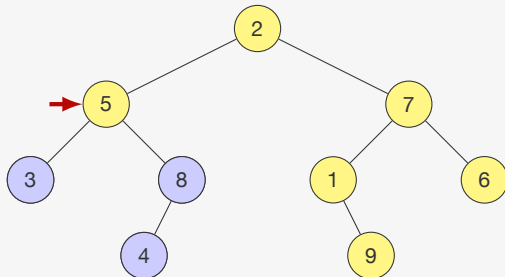


Ex: 3, 4,

Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro visita a subárvore esquerda
- depois a subárvore direita
- e por último visita a raiz

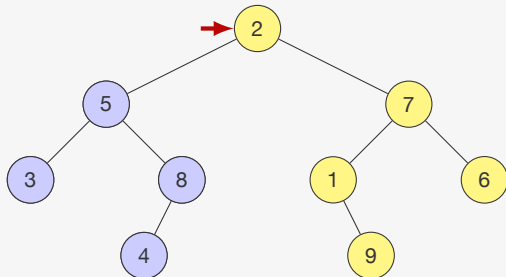


Ex: 3, 4, 8,

Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro visita a subárvore esquerda
- depois a subárvore direita
- e por último visita a raiz

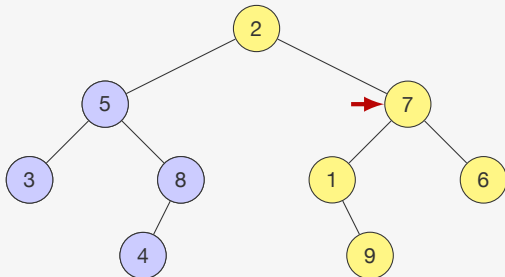


Ex: 3, 4, 8, 5,

Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro visita a subárvore esquerda
- depois a subárvore direita
- e por último visita a raiz

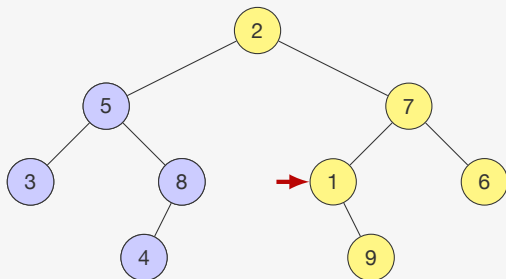


Ex: 3, 4, 8, 5,

Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro visita a subárvore esquerda
- depois a subárvore direita
- e por último visita a raiz

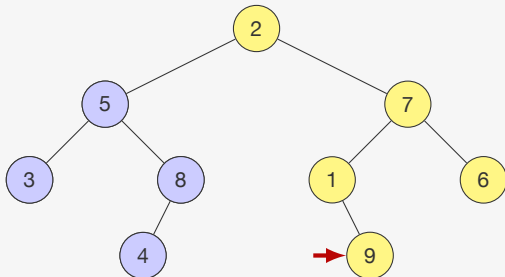


Ex: 3, 4, 8, 5,

Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro visita a subárvore esquerda
- depois a subárvore direita
- e por último visita a raiz

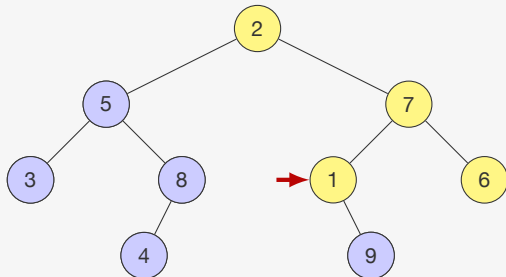


Ex: 3, 4, 8, 5,

Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro visita a subárvore esquerda
- depois a subárvore direita
- e por último visita a raiz

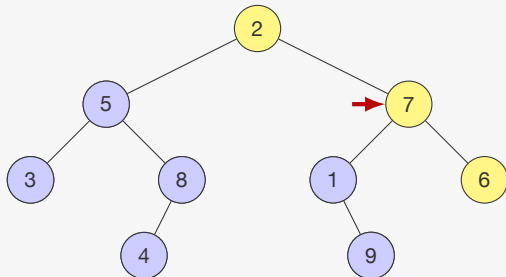


Ex: 3, 4, 8, 5, 9,

Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro visita a subárvore esquerda
- depois a subárvore direita
- e por último visita a raiz

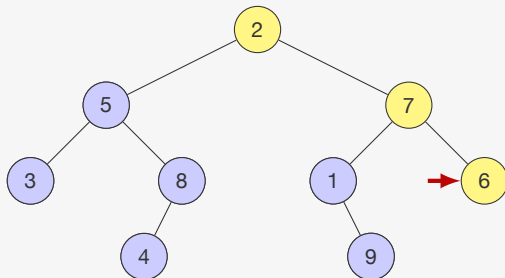


Ex: 3, 4, 8, 5, 9, 1,

Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro visita a subárvore esquerda
- depois a subárvore direita
- e por último visita a raiz

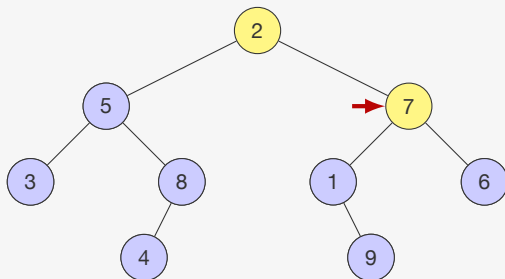


Ex: 3, 4, 8, 5, 9, 1,

Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro visita a subárvore esquerda
- depois a subárvore direita
- e por último visita a raiz

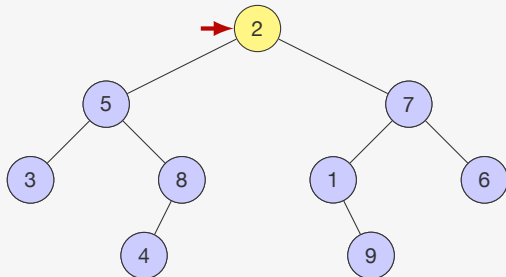


Ex: 3, 4, 8, 5, 9, 1, 6,

Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro visita a subárvore esquerda
- depois a subárvore direita
- e por último visita a raiz

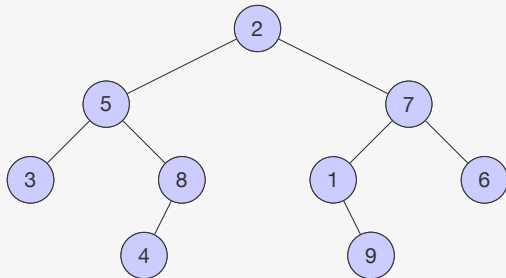


Ex: 3, 4, 8, 5, 9, 1, 6, 7,

Percorrendo os nós - Pós-ordem

A pós-ordem

- primeiro visita a subárvore esquerda
- depois a subárvore direita
- e por último visita a raiz

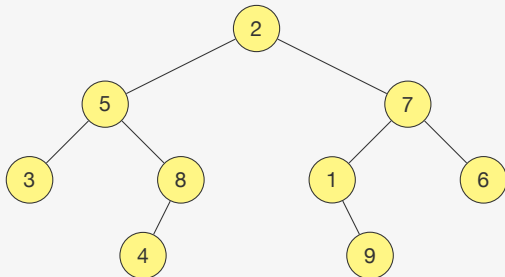


Ex: 3, 4, 8, 5, 9, 1, 6, 7, 2

Percorrendo os nós - Inordem

A inordem

- primeiro visita a subárvore esquerda
- depois visita a raiz
- e por última visita a subárvore direita

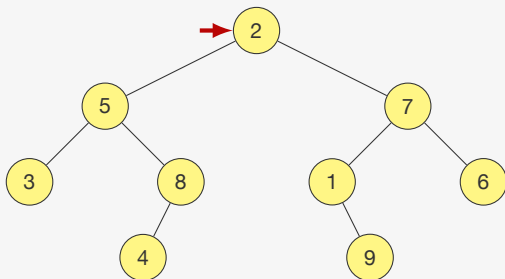


Ex:

Percorrendo os nós - Inordem

A inordem

- primeiro visita a subárvore esquerda
- depois visita a raiz
- e por última visita a subárvore direita

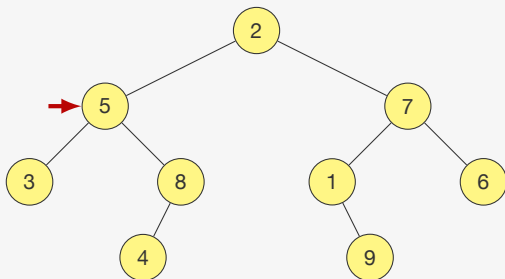


Ex:

Percorrendo os nós - Inordem

A inordem

- primeiro visita a subárvore esquerda
- depois visita a raiz
- e por última visita a subárvore direita

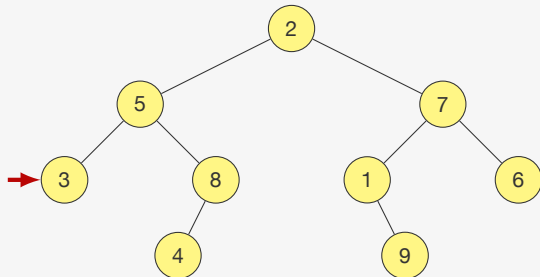


Ex:

Percorrendo os nós - Inordem

A inordem

- primeiro visita a subárvore esquerda
- depois visita a raiz
- e por última visita a subárvore direita

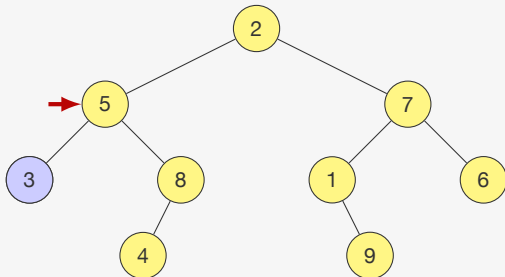


Ex:

Percorrendo os nós - Inordem

A inordem

- primeiro visita a subárvore esquerda
- depois visita a raiz
- e por última visita a subárvore direita

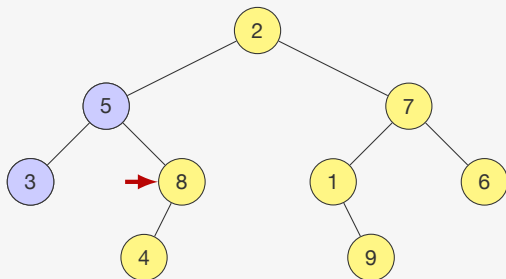


Ex: 3,

Percorrendo os nós - Inordem

A inordem

- primeiro visita a subárvore esquerda
- depois visita a raiz
- e por última visita a subárvore direita

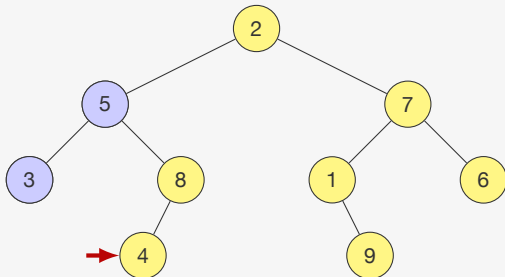


Ex: 3, 5,

Percorrendo os nós - Inordem

A inordem

- primeiro visita a subárvore esquerda
- depois visita a raiz
- e por última visita a subárvore direita

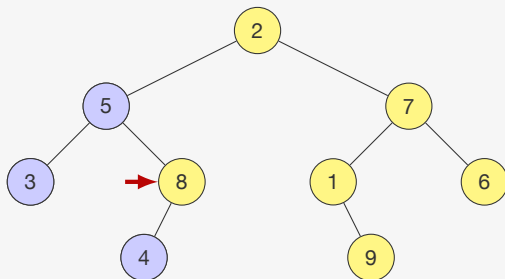


Ex: 3, 5,

Percorrendo os nós - Inordem

A inordem

- primeiro visita a subárvore esquerda
- depois visita a raiz
- e por última visita a subárvore direita

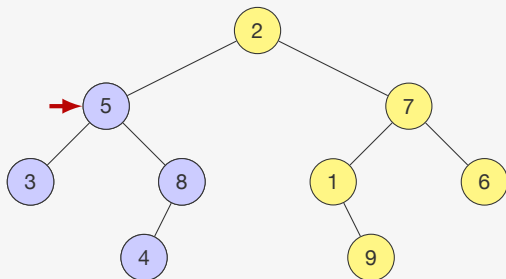


Ex: 3, 5, 4,

Percorrendo os nós - Inordem

A inordem

- primeiro visita a subárvore esquerda
- depois visita a raiz
- e por última visita a subárvore direita

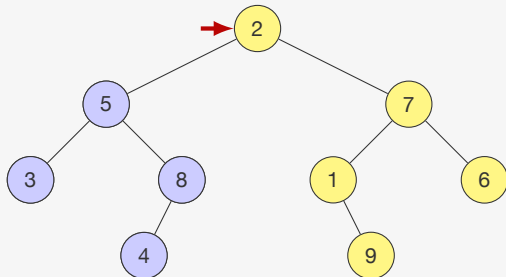


Ex: 3, 5, 4, 8,

Percorrendo os nós - Inordem

A inordem

- primeiro visita a subárvore esquerda
- depois visita a raiz
- e por última visita a subárvore direita

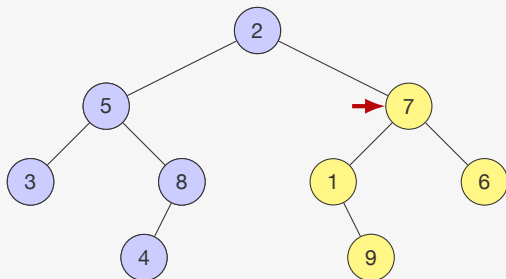


Ex: 3, 5, 4, 8,

Percorrendo os nós - Inordem

A inordem

- primeiro visita a subárvore esquerda
- depois visita a raiz
- e por última visita a subárvore direita

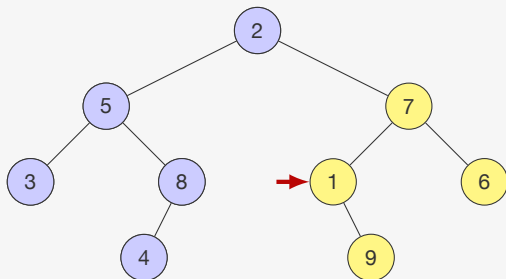


Ex: 3, 5, 4, 8, 2,

Percorrendo os nós - Inordem

A inordem

- primeiro visita a subárvore esquerda
- depois visita a raiz
- e por última visita a subárvore direita

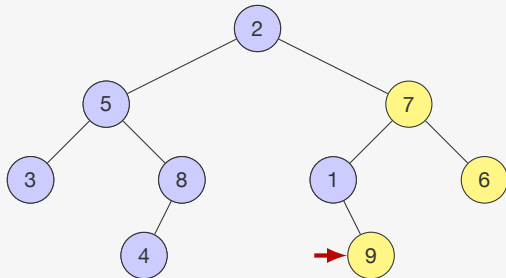


Ex: 3, 5, 4, 8, 2,

Percorrendo os nós - Inordem

A inordem

- primeiro visita a subárvore esquerda
- depois visita a raiz
- e por última visita a subárvore direita

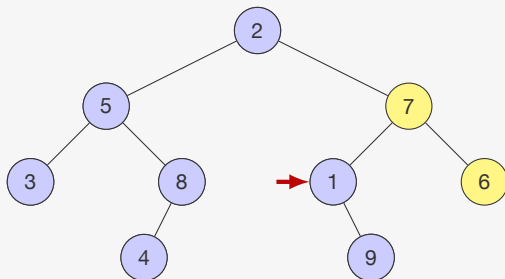


Ex: 3, 5, 4, 8, 2, 1,

Percorrendo os nós - Inordem

A inordem

- primeiro visita a subárvore esquerda
- depois visita a raiz
- e por última visita a subárvore direita

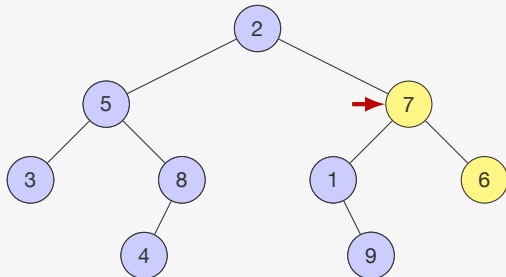


Ex: 3, 5, 4, 8, 2, 1, 9,

Percorrendo os nós - Inordem

A inordem

- primeiro visita a subárvore esquerda
- depois visita a raiz
- e por última visita a subárvore direita

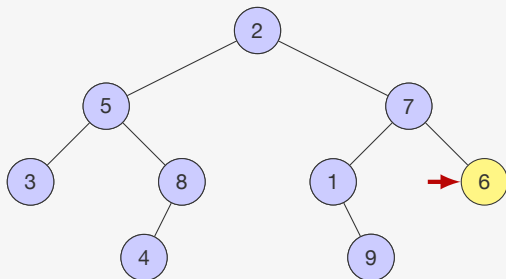


Ex: 3, 5, 4, 8, 2, 1, 9,

Percorrendo os nós - Inordem

A inordem

- primeiro visita a subárvore esquerda
- depois visita a raiz
- e por última visita a subárvore direita

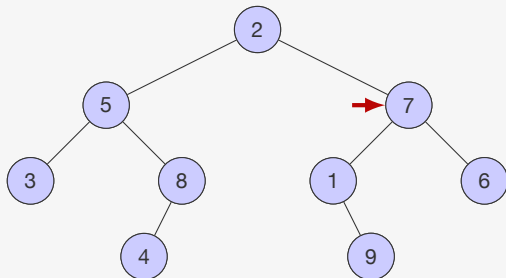


Ex: 3, 5, 4, 8, 2, 1, 9, 7,

Percorrendo os nós - Inordem

A inordem

- primeiro visita a subárvore esquerda
- depois visita a raiz
- e por última visita a subárvore direita

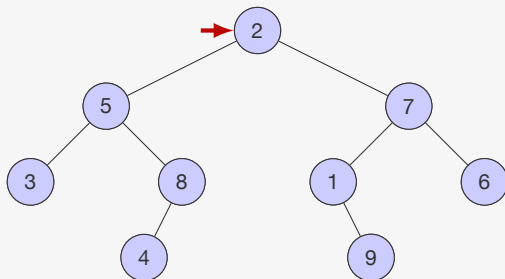


Ex: 3, 5, 4, 8, 2, 1, 9, 7, 6

Percorrendo os nós - Inordem

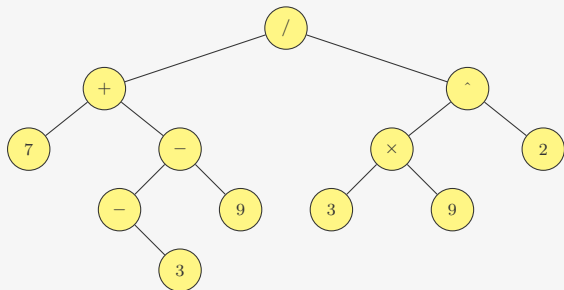
A inordem

- primeiro visita a subárvore esquerda
- depois visita a raiz
- e por última visita a subárvore direita



Ex: 3, 5, 4, 8, 2, 1, 9, 7, 6

Percurso em profundidade e expressões



Notação

- **Pré-fixa:** / + 7 - - 3 9 ^ × 3 9 2
- **Pós-fixa:** 7 3 - 9 - + 3 9 × 2 ^ /
- **Infixa:** 7 + - 3 - 9 / 3 × 9 ^ 2

Árvores binárias - Implementação em C

- Percurso em profundidade:
 - ▶ Visita os nós descendo os ramos (profundidade)
 - ▶ Recursivamente: da esquerda para a direita

```
1 void preordem(no *raiz) {  
2     if (raiz != NULL) {  
3         printf("%d ", raiz->dado); /* visita/processa raiz */
```

```
1 void preordem(no *raiz) {
2     if (raiz != NULL) {
3         printf("%d ", raiz->dado); /* visita/processa raiz */
4         preordem(raiz->esq);
```

```
1 void preordem(no *raiz) {
2     if (raiz != NULL) {
3         printf("%d ", raiz->dado); /* visita/processa raiz */
4         preordem(raiz->esq);
5         preordem(raiz->dir);
6     }
7 }
8
9 void inordem(no *raiz) {
10    if (raiz != NULL) {
```

```
1 void preordem(no *raiz) {
2     if (raiz != NULL) {
3         printf("%d ", raiz->dado); /* visita/processa raiz */
4         preordem(raiz->esq);
5         preordem(raiz->dir);
6     }
7 }
8
9 void inordem(no *raiz) {
10    if (raiz != NULL) {
11        inordem(raiz->esq);
```

```
1 void preordem(no *raiz) {
2     if (raiz != NULL) {
3         printf("%d ", raiz->dado); /* visita/processa raiz */
4         preordem(raiz->esq);
5         preordem(raiz->dir);
6     }
7 }
8
9 void inordem(no *raiz) {
10    if (raiz != NULL) {
11        inordem(raiz->esq);
12        printf("%d ", raiz->dado); /* visita/processa raiz */
```

```
1 void preordem(no *raiz) {
2     if (raiz != NULL) {
3         printf("%d ", raiz->dado); /* visita/processa raiz */
4         preordem(raiz->esq);
5         preordem(raiz->dir);
6     }
7 }
8
9 void inordem(no *raiz) {
10    if (raiz != NULL) {
11        inordem(raiz->esq);
12        printf("%d ", raiz->dado); /* visita/processa raiz */
13        inordem(raiz->dir);
14    }
15 }
16
17 void posordem(no *raiz) {
18    if (raiz != NULL) {
```

```
1 void preordem(no *raiz) {
2     if (raiz != NULL) {
3         printf("%d ", raiz->dado); /* visita/processa raiz */
4         preordem(raiz->esq);
5         preordem(raiz->dir);
6     }
7 }
8
9 void inordem(no *raiz) {
10    if (raiz != NULL) {
11        inordem(raiz->esq);
12        printf("%d ", raiz->dado); /* visita/processa raiz */
13        inordem(raiz->dir);
14    }
15 }
16
17 void posordem(no *raiz) {
18    if (raiz != NULL) {
19        posordem(raiz->esq);
```



```

1 void preordem(no *raiz) {
2     if (raiz != NULL) {
3         printf("%d ", raiz->dado); /* visita/processa raiz */
4         preordem(raiz->esq);
5         preordem(raiz->dir);
6     }
7 }
8
9 void inordem(no *raiz) {
10    if (raiz != NULL) {
11        inordem(raiz->esq);
12        printf("%d ", raiz->dado); /* visita/processa raiz */
13        inordem(raiz->dir);
14    }
15 }
16
17 void posordem(no *raiz) {
18    if (raiz != NULL) {
19        posordem(raiz->esq);
20        posordem(raiz->dir);

```

```

1 void preordem(no *raiz) {
2     if (raiz != NULL) {
3         printf("%d ", raiz->dado); /* visita/processa raiz */
4         preordem(raiz->esq);
5         preordem(raiz->dir);
6     }
7 }
8
9 void inordem(no *raiz) {
10    if (raiz != NULL) {
11        inordem(raiz->esq);
12        printf("%d ", raiz->dado); /* visita/processa raiz */
13        inordem(raiz->dir);
14    }
15 }
16
17 void posordem(no *raiz) {
18    if (raiz != NULL) {
19        posordem(raiz->esq);
20        posordem(raiz->dir);
21        printf("%d ", raiz->dado); /* visita/processa raiz */
22    }
23 }

```

Sem recursão - TAD?

```
1 void pre_ordem(no *raiz) {
```

Sem recursão - TAD?

```
1 void pre_ordem(no *raiz) {
2
3     cabeca *p;
4     p = criar_lista();
5
6     empilha(p, raiz);
7
8     while (!vazia(p)) {
9         raiz = desempilha(p);
10
11         if (raiz != NULL) {
```

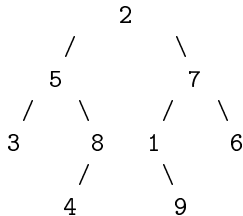
Sem recursão - TAD?

```
1 void pre_ordem(no *raiz) {
2
3     cabeca *p;
4     p = criar_lista();
5
6     empilha(p, raiz);
7
8     while (!vazia(p)) {
9         raiz = desempilha(p);
10
11         if (raiz != NULL) {
12             //processa a raiz
13             printf("%d ", raiz->dado);
14
15             //empilhar nó direita ou esquerda?
```

Sem recursão - TAD?

```
1 void pre_ordem(no *raiz) {
2
3     cabeca *p;
4     p = criar_lista();
5
6     empilha(p, raiz);
7
8     while (!vazia(p)) {
9         raiz = desempilha(p);
10
11         if (raiz != NULL) {
12             //processa a raiz
13             printf("%d ", raiz->dado);
14
15             //empilhar nó direita ou esquerda?
16             empilha(p, raiz->dir);
17             empilha(p, raiz->esq);
18         }
19     }
20 }
21 free(p);
22 }
```

2 5 3 8 4 7 1 9 6



Sem recursão - TAD?

```
1 void in_ordem(no *raiz) {
```

Sem recursão - TAD?

```
1 void in_ordem(no *raiz) {
2   cabeca *p = criar_lista();
3   //empilha todos os nós esquerda
4   while (raiz!=NULL) {
5     empilha(p, raiz);
6     raiz = raiz->esq;
7   }
8
9   //visita os nós empilhados
10  //qual o primeiro nó a ser visitado??
```


Sem recursão - TAD?

```
1 void in_ordem(no *raiz) {
2     cabeca *p = criar_lista();
3     //empilha todos os nós esquerda
4     while (raiz!=NULL) {
5         empilha(p, raiz);
6         raiz = raiz->esq;
7     }
8
9     //visita os nós empilhados
10    //qual o primeiro nó a ser visitado??
11    while (!vazia(p)) {
12        raiz = desempilha(p);
13        if (raiz != NULL) {
14
15            //processa o nó mais a esquerda
16            printf("%d ", raiz->dado);
```

Sem recursão - TAD?

```
1 void in_ordem(no *raiz) {
2   cabeca *p = criar_lista();
3   //empilha todos os nós esquerda
4   while (raiz!=NULL) {
5     empilha(p, raiz);
6     raiz = raiz->esq;
7   }
8
9   //visita os nós empilhados
10  //qual o primeiro nó a ser visitado??
11  while (!vazia(p)) {
12    raiz = desempilha(p);
13    if (raiz != NULL) {
14
15      //processa o nó mais a esquerda
16      printf("%d ", raiz->dado);
17
18      //nós direita do mais esquerda
```

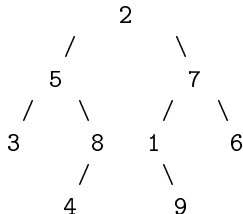
Sem recursão - TAD?

```
1 void in_ordem(no *raiz) {
2   cabeca *p = criar_lista();
3   //empilha todos os nós esquerda
4   while (raiz!=NULL) {
5     empilha(p, raiz);
6     raiz = raiz->esq;
7   }
8
9   //visita os nós empilhados
10  //qual o primeiro nó a ser visitado??
11  while (!vazia(p)) {
12    raiz = desempilha(p);
13    if (raiz != NULL) {
14
15      //processa o nó mais a esquerda
16      printf("%d ", raiz->dado);
17
18      //nós direita do mais esquerda
19      raiz = raiz->dir;
20      while (raiz!=NULL) {
21        empilha(p, raiz);
22        raiz = raiz->esq;
23      }
24
25      //qual o próximo nó a ser visitado??
```

Sem recursão - TAD?

```
1 void in_ordem(no *raiz) {
2   cabeca *p = criar_lista();
3   //empilha todos os nós esquerda
4   while (raiz!=NULL) {
5     empilha(p, raiz);
6     raiz = raiz->esq;
7   }
8
9   //visita os nós empilhados
10  //qual o primeiro nó a ser visitado??
11  while (!vazia(p)) {
12    raiz = desempilha(p);
13    if (raiz != NULL) {
14
15      //processa o nó mais a esquerda
16      printf("%d ", raiz->dado);
17
18      //nós direita do mais esquerda
19      raiz = raiz->dir;
20      while (raiz!=NULL) {
21        empilha(p, raiz);
22        raiz = raiz->esq;
23      }
24
25      //qual o próximo nó a ser visitado??
26    }
27  }
28  free(p);
29 }
```

3 5 4 8 2 1 9 7 6



Sem recursão - TAD?

```
1 void pos_ordem(no *raiz) {
```

Sem recursão - TAD?

```
1 void pos_ordem(no *raiz) {
2   cabeca *p, *raizes;
3   p = criar_lista();           //pilha para percurso
4   raizes = criar_lista();     //pilha para raizes
5
6   empilha(p, raiz);
7   while (!vazia(p)) {
8     raiz = desempilha(p);
9     if (raiz != NULL) {
10      //visita a raiz e processa depois
11      empilha(raizes, raiz);
12
13      //subarvore esquerda
14      if (raiz->esq) empilha(p, raiz->esq);
15      if (raiz->dir) empilha(p, raiz->dir);
16
17      //visitar da dir. p/ esq. (??)
```

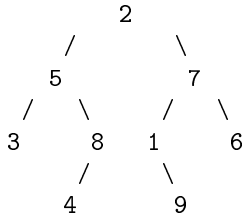
Sem recursão - TAD?

```
1 void pos_ordem(no *raiz) {
2     cabeca *p, *raizes;
3     p = criar_lista();           //pilha para percurso
4     raizes = criar_lista();     //pilha para raizes
5
6     empilha(p, raiz);
7     while (!vazia(p)) {
8         raiz = desempilha(p);
9         if (raiz != NULL) {
10            //visita a raiz e processa depois
11            empilha(raizes, raiz);
12
13            //subarvore esquerda
14            if (raiz->esq) empilha(p, raiz->esq);
15            if (raiz->dir) empilha(p, raiz->dir);
16
17            //visitar da dir. p/ esq. (??)
18            //processar na ordem inversa
19        }
20    }
21
22    //processar raizes
```

Sem recursão - TAD?

```
1 void pos_ordem(no *raiz) {
2   cabeca *p, *raizes;
3   p = criar_lista();           //pilha para percurso
4   raizes = criar_lista();     //pilha para raizes
5
6   empilha(p, raiz);
7   while (!vazia(p)) {
8     raiz = desempilha(p);
9     if (raiz != NULL) {
10      //visita a raiz e processa depois
11      empilha(raizes, raiz);
12
13      //subarvore esquerda
14      if(raiz->esq) empilha(p, raiz->esq);
15      if(raiz->dir) empilha(p, raiz->dir);
16
17      //visitar da dir. p/ esq. (??)
18      //processar na ordem inversa
19    }
20  }
21
22  //processar raizes
23  while (!vazia(raizes)) {
24    raiz = desempilha(raizes);
25    printf("%d ", raiz->dado);
26  }
27 }
```

3 4 8 5 9 1 6 7 2



Árvores binárias - Implementação em C

- Percurso em largura - TAD?
 - ▶ Visita os nós por níveis
 - ▶ Da esquerda para a direita

```
1 void percurso_em_largura (no *raiz) {
2     cabeca *f = criar_lista();
3     enfileira(f, raiz);
4     while (!vazia(f)) {
5         raiz = desenfileira(f);
6         if (raiz != NULL) {
7             //qual nó enfileiramos primeiro:
8             // direita ou esquerda?
```

Árvores binárias - Implementação em C

- Percurso em largura - TAD?
 - ▶ Visita os nós por níveis
 - ▶ Da esquerda para a direita

```
1 void percurso_em_largura (no *raiz) {
2     cabeca *f = criar_lista();
3     enfileira(f, raiz);
4     while (!vazia(f)) {
5         raiz = desenfileira(f);
6         if (raiz != NULL) {
7             //qual nó enfileiramos primeiro:
8             // direita ou esquerda?
9             enfileira(f, raiz->esq);
10            enfileira(f, raiz->dir);

```

Árvores binárias - Implementação em C

- Percurso em largura - TAD?
 - ▶ Visita os nós por níveis
 - ▶ Da esquerda para a direita

```
1 void percurso_em_largura (no *raiz) {
2     cabeca *f = criar_lista();
3     enfileira(f, raiz);
4     while (!vazia(f)) {
5         raiz = desenfileira(f);
6         if (raiz != NULL) {
7             //qual nó enfileiramos primeiro:
8             // direita ou esquerda?
9             enfileira(f, raiz->esq);
10            enfileira(f, raiz->dir);
11            printf("%d ", raiz->dado);
12        }
13    }
14 }
```

Árvores binárias - Algoritmos de Percurso - exercício

- Retirado dos slides do prof. Siang Wun Song (USP)
- Considere uma árvore binária
- Cada nó tem um campo adicional chamado “alt”
- Escreva um algoritmo que coloque no campo “alt” de cada nó x da árvore binária, a altura da subárvore enraizada em x
- Dica: use pós-ordem e uma rotina apropriada de visita
- Tente fazer este exercício e veja o motivo da adoção da pós-ordem