Algoritmos e Estrutura de Dados

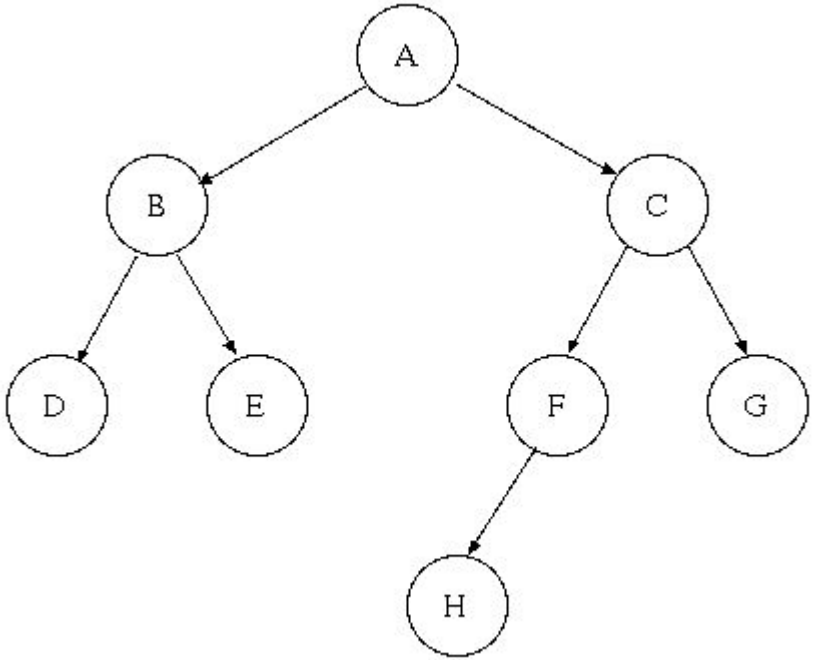
Árvores Binárias

Prof. Nilton Luiz Queiroz Jr.

Árvores binárias

- São um tipo especial de árvore:
  - Nenhum nó possui grau maior que dois;
    - Todos os nós possuem no **máximo** dois filhos;
- Toda subárvore de uma árvore binária é uma sub árvore binária;

Árvores binárias



Árvores binárias

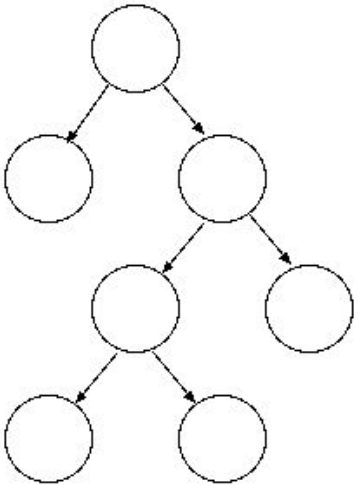
- Uma árvore binária T:
  - Pode ser nula
    - Não possuir raiz;
  - Possuir:
    - Um nó raiz;
      - Nó de referência da árvore
    - Nós particionados em duas estruturas distintas Te e Td;
      - Te denominada subárvore esquerda de T;
      - Td denominada subárvore direita de T;

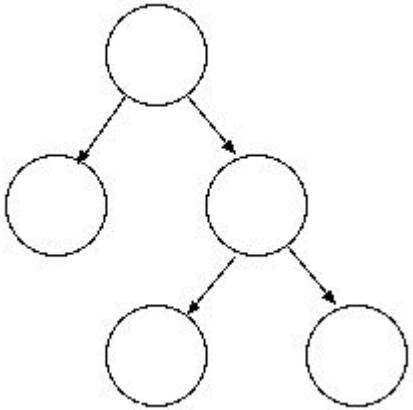
Terminologia

- Existem diferentes tipos de árvores binárias:
  - Estritamente binária;
  - Binária completa;
  - Binária quase completa;
  - Degenerada;
  - Binária de busca;
  - Binária balanceada;

Tipos de árvores binárias

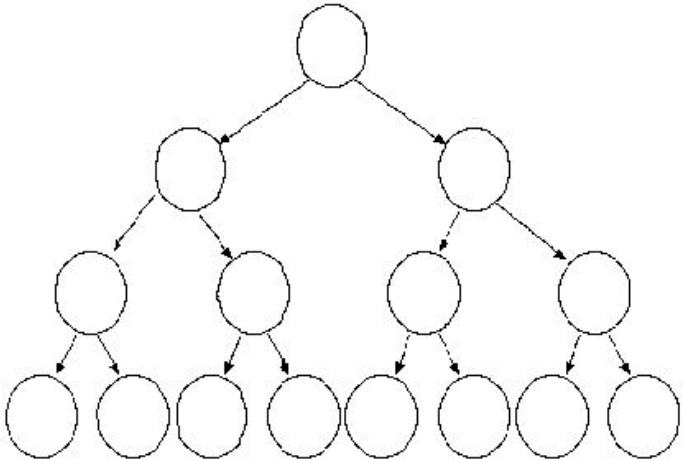
- Árvore estritamente binária:
  - Cada nó da árvore tem grau 2 **ou** grau 0;
    - Não são admitidos nós com grau 1;

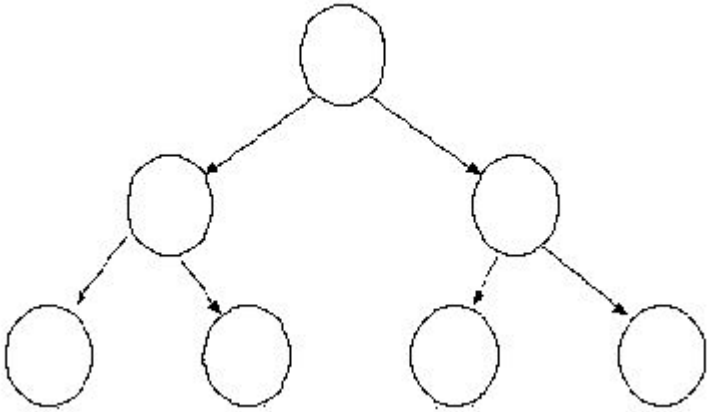




Tipos de árvores binárias

- Árvore binária completa:
  - Todos níveis devem ser completos;
    - Todos nós folhas devem estar no mesmo nível;
    - Todos nós intermediários devem ter grau 2;





Propriedades de árvores binárias completas

- Se uma árvore binária for completa e seu nível d possuir n nós, então seu nível d+1 terá 2n nós;
- Cada nível d possui exatamente 2^d nós

Nível (*d*)	nº max. nós	Potência
0	1	2^0
1	2	2^1
2	4	2^2
3	8	2^3
4	16	2^4
5	32	2^5

Propriedades de árvores binárias completas

- A altura (**h**) de uma árvore binária completa com n nós pode ser calculada por:

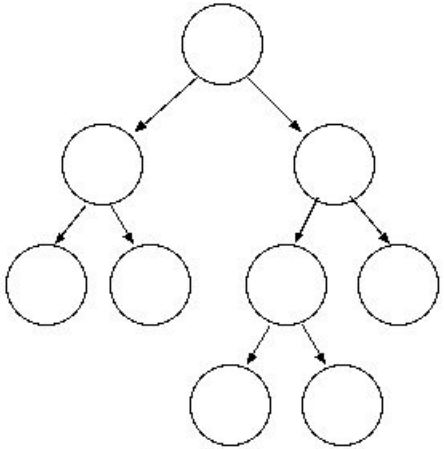
$$h = \log_2 (n+1)$$

- O número de nós (**n**) da árvore pode ser calculado em função da altura da árvore:

$$n = 2^h - 1$$

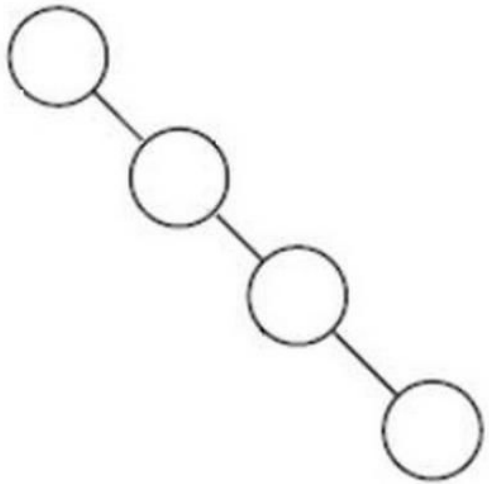
Tipos de árvores binárias

- Árvores binárias quase completas são árvores onde todos os níveis da árvore estão completos, exceto o último;
  - O último nível deve ter nós terminais faltantes (nível incompleto);
  - O penúltimo nível deve ter pelo menos um nó não terminal



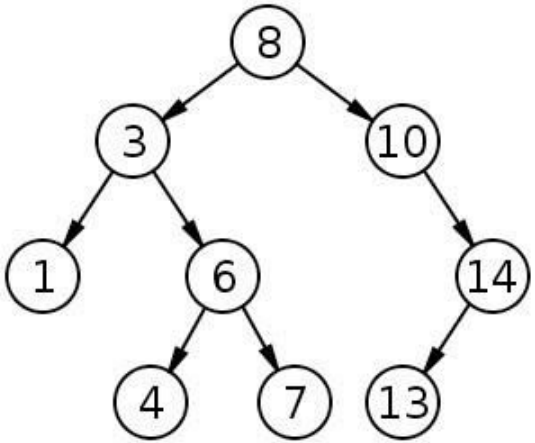
Tipos de árvores binárias

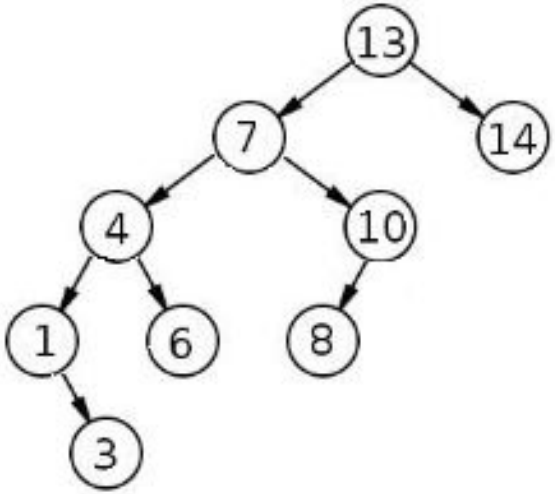
- Árvore degenerada:
  - Cada nó possui apenas um filho;



Tipos de árvores binárias

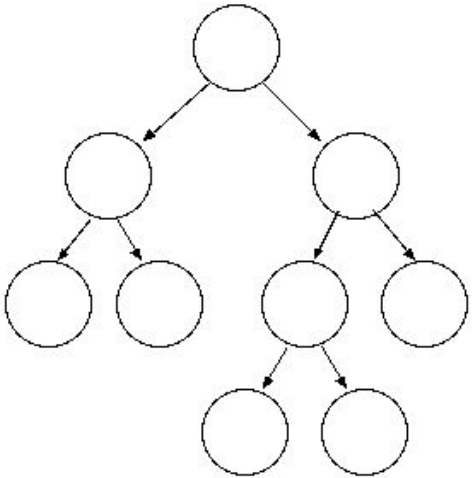
- Árvores binárias de busca:
  - Todos elementos à esquerda são menores que seu elemento raiz;
  - Todos elementos à direita são maiores que o elemento raiz;





Tipos de árvores binárias

- Árvores binárias balanceadas:
  - Para todo nó da árvore suas subárvores esquerda e direita tem diferença de altura de uma unidade



Implementação

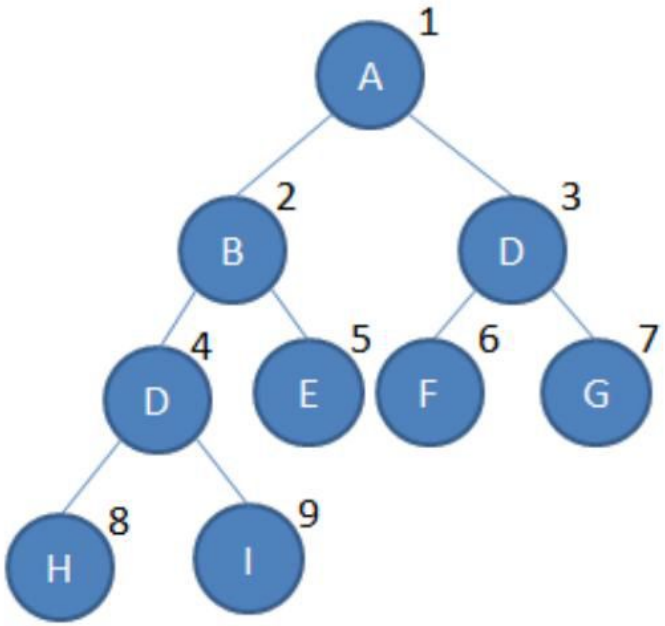
- Árvores binárias podem ser implementadas de duas principais maneiras:
  - Estáticas:
    - Usa-se vetor para implementação;
    - O controle é feito pelos índices;
    - Existe uma restrição;
      - As árvores devem ser completas, ou quase completas;
  - Dinâmicas;
    - Usa-se alocação dinâmica e ponteiros;
    - O controle se torna mais fácil usando recursão;

Implementação estática

- Predefinir o número total de elementos do vetor;
  - Será o número máximo de nós que se quer armazenar;
- Atribuir a primeira posição do vetor como sendo a raiz da árvore;
- Para cada nó n localizado na posição i do vetor, tem-se:
  - O nó filho esquerdo de n se encontra na posição: ne = 2i + 1;
  - O nó filho direito de n se encontra na posição: nd = 2i + 2;

Implementação estática

Árvore T:



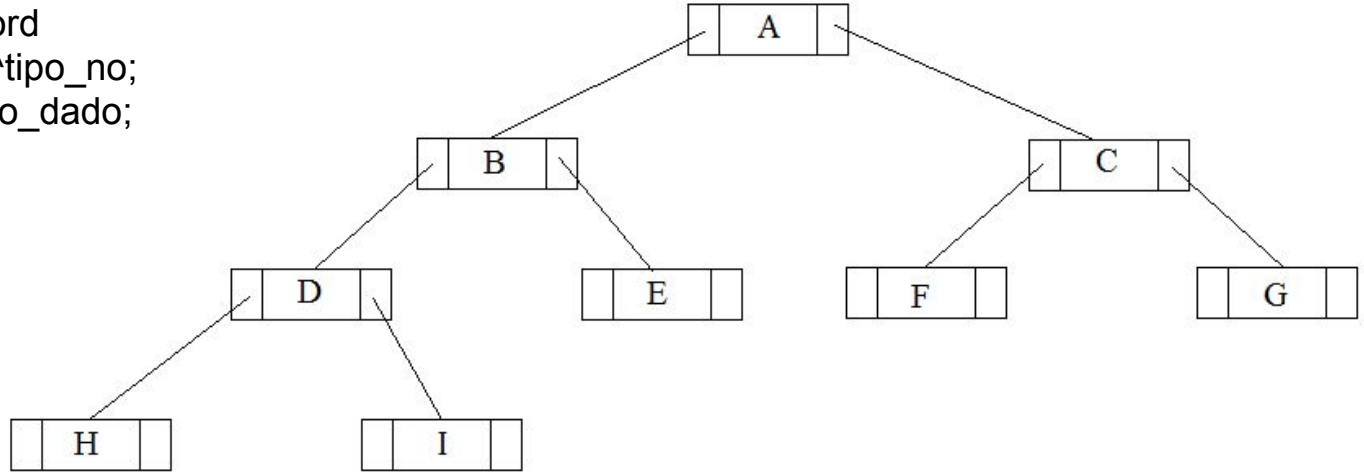
T armazenada em um vetor:

A	B	D	D	E	F	G	H	I

Implementação dinâmica

- Usa-se uma estrutura de dado para cada nó;
  - Essa estrutura contém ponteiros para seus dois filhos e o dado armazenado;

```
tipo_no = record
        esq,dir:^tipo_no;
        dado:tipo_dado;
end;
```



Implementação dinâmica

- A árvore pode ser constituída de um ponteiro que aponta para a raiz;

```
struct tipo_arvore{
    struct tipo_no *raiz;
};
```

Implementação dinâmica

- As inserções são feitas ou a esquerda ou a direita de um determinado nó;
  - Uma exceção é quando a raiz deve ser inserida, como ela não tem pai, é necessário um caso especial para inserir a raiz;
    - Em geral usa-se uma função para inserir a raiz;
- Dessa forma, vamos implementar 3 funções para inserção:
  - Inserção da raiz;
  - Inserção à direita de um nó;
  - inserção à esquerda de um nó;
- Além das inserções serão implementadas funções para:
  - Inicializar a árvore;
  - Verificar se a árvore está vazia;
  - Criar um nó, inicializar seus campos e retornar um ponteiro para o mesmo de nó;

Implementação dinâmica

- Inicialização de árvore:

```
void inicializa(struct tipo_arvore *a){
    a->raiz = NULL;
}
```

- Verificar se a árvore está vazia

```
int vazia(struct tipo_arvore *a){
    return a->raiz == NULL;
}
```

Implementação dinâmica

- Criação de um nó

```
struct tipo_no *cria_no(struct tipo_item x){
    /*algoritmo*/
}
```

Implementação dinâmica

- Criação de um nó

```
struct tipo_no *cria_no(struct tipo_item x){
    struct tipo_no *aux;
    aux=(struct tipo_no *)malloc(sizeof(struct tipo_no));
    aux->dir=NULL;
    aux->esq=NULL;
    aux->dado=x;
    return aux;
}
```

Implementação dinâmica

- Inserção da raiz:
  - É necessário verificar se a árvore já tem uma raiz, pois se existir uma raiz não se pode inserir uma nova;

```
struct tipo_no *insere_raiz(struct tipo_arvore *a, struct tipo_item x){
    /*algoritmo*/
}
```

Implementação dinâmica

- Inserção da raiz:
  - É necessário verificar se a árvore já tem uma raiz, pois se existir uma raiz não se pode inserir uma nova;

```
struct tipo_no *insere_raiz(struct tipo_arvore *a, struct tipo_item x){
    struct tipo_no *aux;
    if (vazia(a)){
        aux = cria_no(x);
        a->raiz = aux;
        return aux;
    }else{
        return NULL;
    }
}
```

Implementação dinâmica

- Inserção do filho direito em um nó *n*;
  - Não é possível inserir um filho direito em um nó *n* quando ele já tem um filho direito;

```
struct tipo_no *insere_dir(struct tipo_no *n, struct tipo_item x){
    /*algoritmo*/
}
```

Implementação dinâmica

- Inserção do filho direito em um nó *n*;
  - Não é possível inserir um filho direito em um nó *n* quando ele já tem um filho direito;

```
struct tipo_no *insere_dir(struct tipo_no *n, struct tipo_item x){
    struct tipo_no *aux;
    if(n->dir==NULL){
        aux=cria_no(x);
        n->dir=aux;
        return aux;
    }else{
        return NULL;
    }
}
```

Implementação dinâmica

- Inserção do filho esquerdo em um nó *n*

```
struct tipo_no *insere_esq(struct tipo_no *n, struct tipo_item x){
    /*algoritmo*/
}
```

Implementação dinâmica

- Inserção do filho esquerdo em um nó *n*

```
struct tipo_no *insere_esq(struct tipo_no *n, struct tipo_item x){
    struct tipo_no *aux;
    if(n->esq==NULL){
        aux=cria_no(x);
        n->esq=aux;
        return aux;
    }else{
        return NULL;
    }
}
```

Percurso em Árvores

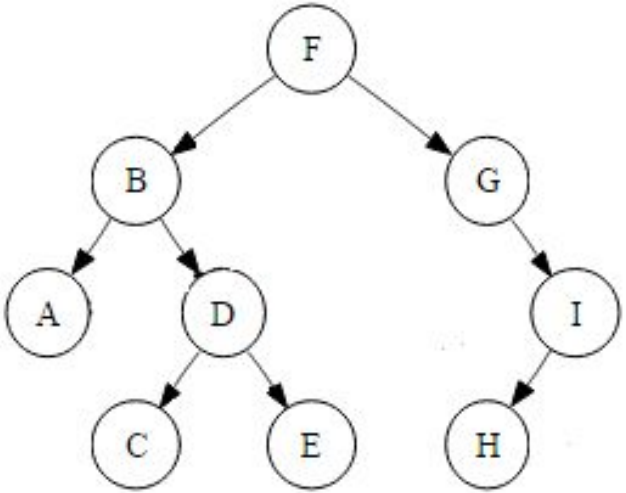
- Muitas vezes se deseja percorrer todos os nós de uma árvore;
- Existem diferentes tipos de percursos levando em consideração a disposição dos elementos:
  - Percurso Pré-Ordem;
  - Percurso Em-Ordem;
  - Percurso Pós-Ordem;

Percurso Pré-Ordem

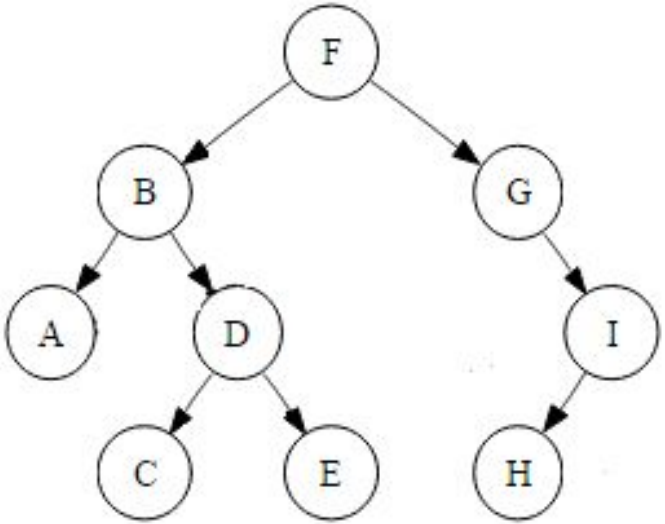
- No percurso pré-ordem a raiz é o primeiro nó a ser visitado;
- Para realizar o percurso são realizadas as seguintes operações:
  - Visita-se o nó;
  - Faz-se o percurso pré-ordem na subárvore esquerda;
  - Faz-se o percurso pré-ordem na subárvore direita;

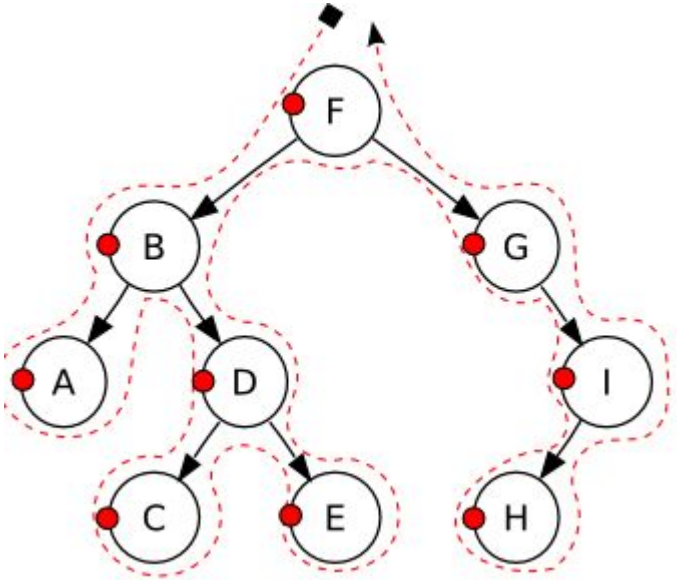
Percurso Pré-Ordem

- Dada a seguinte árvore, de o resultado da impressão dos valores em pré-ordem.



Percurso Pré-Ordem





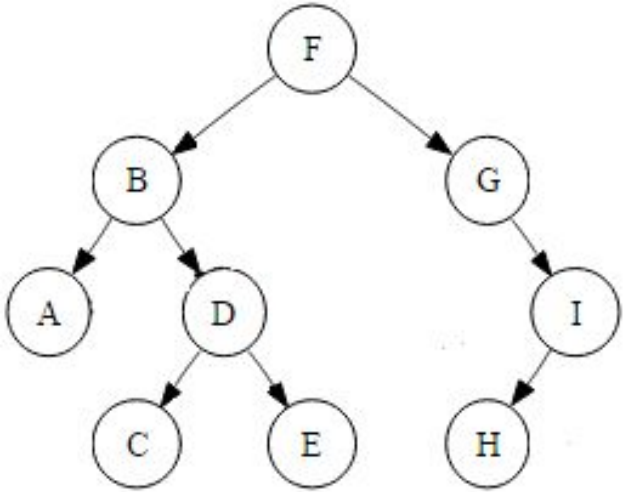
Impressão: {F, B, A, D, C, E, G, I, H}

Percurso Em-Ordem

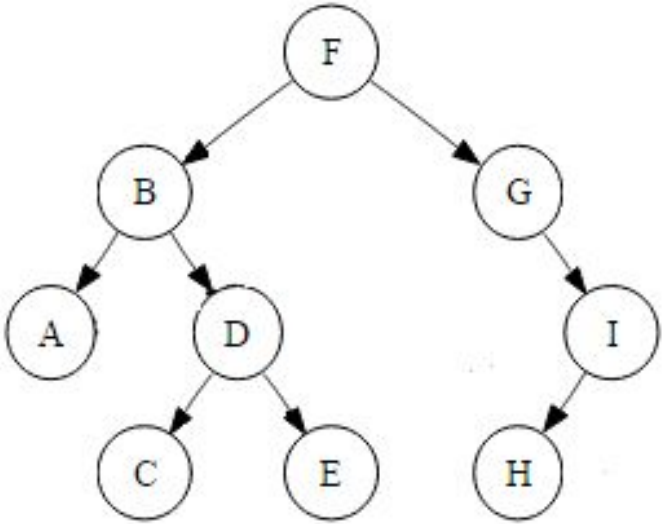
- No percurso em Ordem a raiz é visitada após percorrer toda subárvore esquerda;
- Após visitar a raiz a subárvore direita é visitada;
- Para realizar o percurso são realizadas as seguintes operações:
  - Faz-se o percurso em-ordem na subárvore esquerda;
  - Visita-se o nó;
  - Faz-se o percurso em-ordem na subárvore direita;
- Quando aplicado em uma árvore binária de busca esse percurso visita os elementos de maneira ordenada;

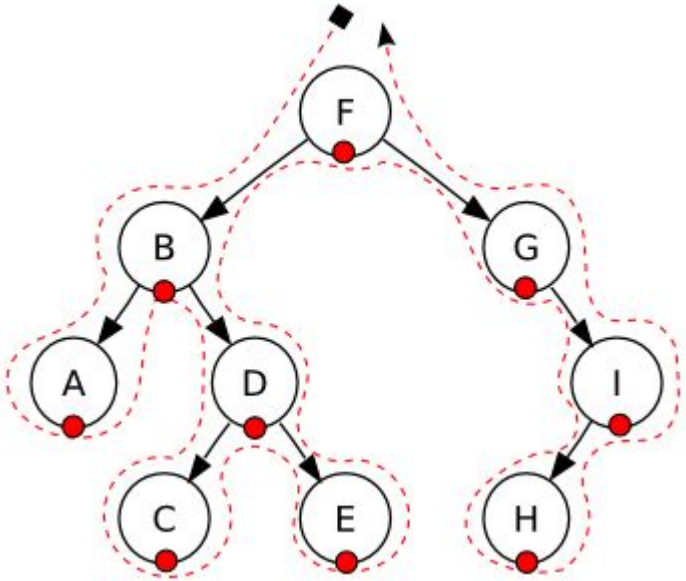
Percurso Em-Ordem

- Dada a seguinte árvore, de o resultado da impressão dos valores em em-ordem.



Percurso Em-Ordem





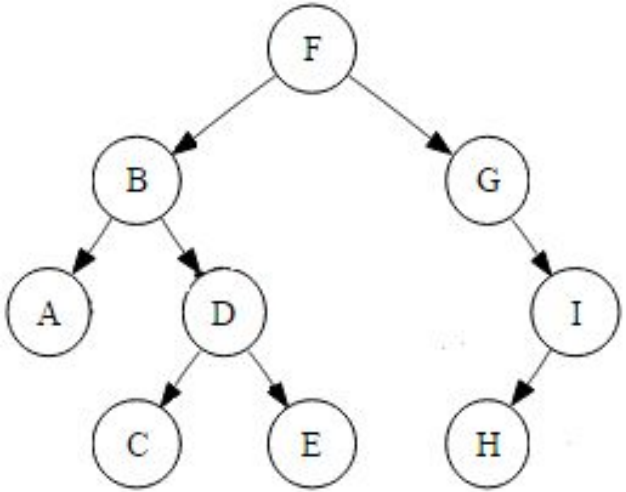
Impressão: {A, B, C, D, E, G, H, I}

Percurso Pós-Ordem

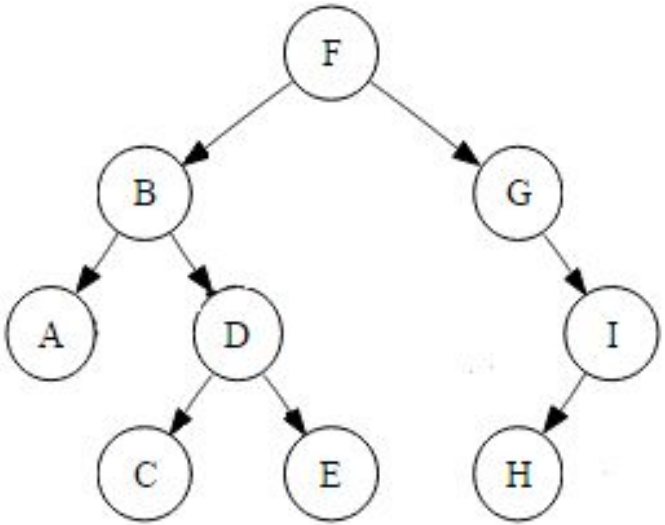
- A raiz é o último nó a ser visitado;
- Para realizar o percurso são realizadas as seguintes operações:
  - Faz-se o percurso pós-ordem na subárvore esquerda;
  - Faz-se o percurso pós-ordem na subárvore direita;
  - Visita-se o nó;

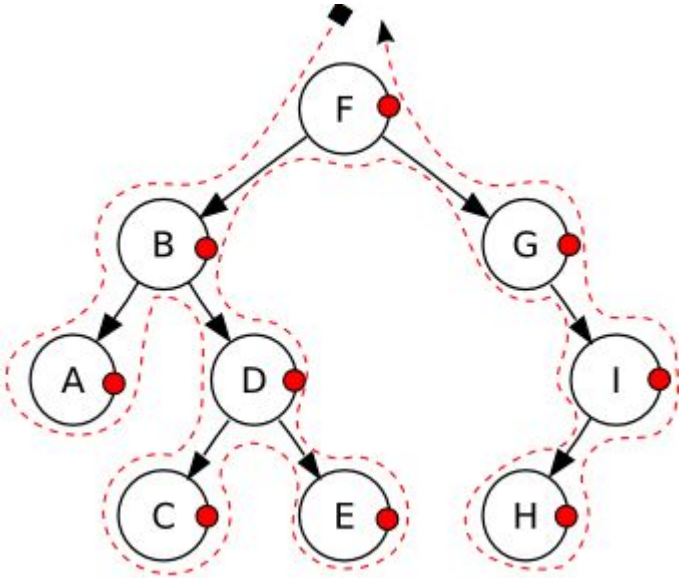
Percurso Pós-Ordem

- Dada a seguinte árvore, de o resultado da impressão dos valores em pós-ordem.



Percurso Pós-Ordem





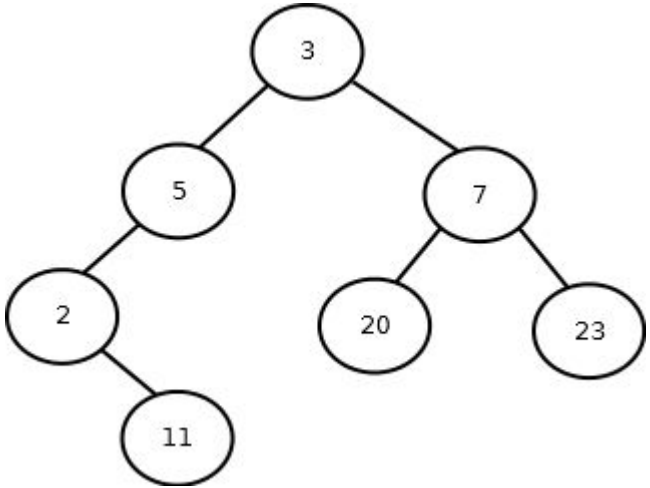
Impressão: {A, C, E, D, B, H, I, G, F}

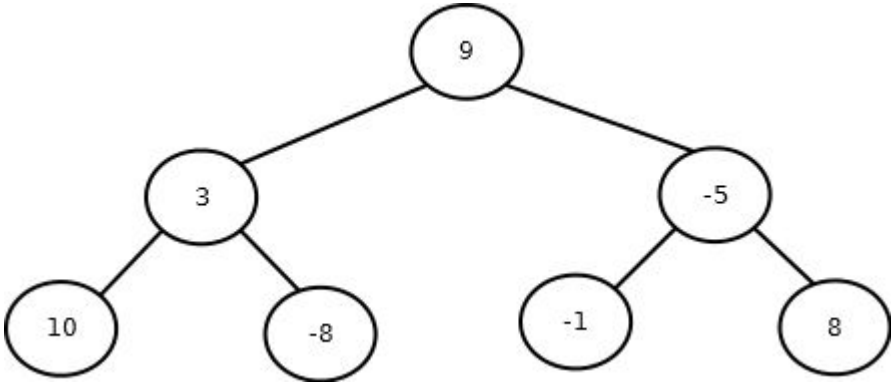
Exercícios

1. Qual estrutura de dados já conhecida é semelhante a uma árvore binária degenerada?
2. Dada uma árvore de altura 4, responda:
   a. Qual o minimo de nós que essa árvore tem?
   b. Qual o número máximo de nós que ela pode ter?
3. Dada uma árvore binária completa com 63 nós, responda:
   a. Qual sua altura?
   b. Quantos nós a árvore tem no nível do nó com maior nível?

Exercícios

4. Dada as seguintes árvores, escreva o resultado da impressão dos valores das árvores para cada um dos 3 percursos na árvore.





Exercícios

5. Construa as árvores do exercício 4 em memória, utilizando as funções de inserção implementadas nos slides anteriores;
6. Implemente os procedimentos de percurso em árvores binárias (em ordem, pré ordem e pós ordem), para escrever a chave dos nós.
   Obs: Teste suas implementações nas árvores construídas no exercício 5, comparando com os resultados do exercício 4.
7. Implemente um subprograma que receba uma árvore e uma chave, e retorne um ponteiro para o nó que armazena aquela chave. Assuma que não existem chaves repetidas na árvore.