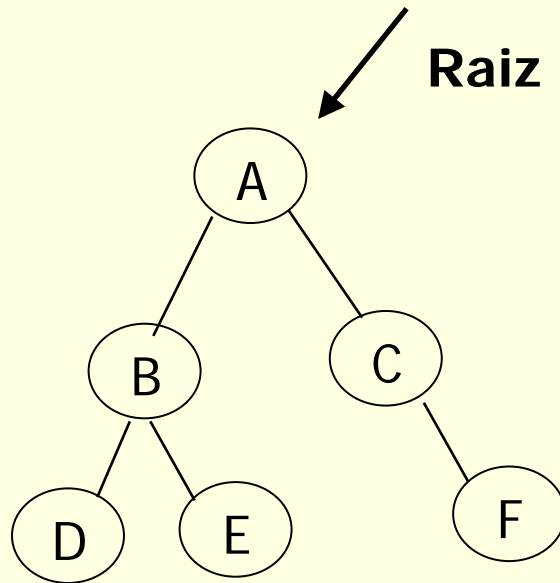


Árvores

SCC-502 – Algoritmos e Estruturas de
Dados I

Árvores binárias

- Árvores com grau 2, ou seja, cada nó pode ter 2 filhos, no máximo



Terminologia:

- filho esquerdo
- filho direito
- informação

Árvores binárias

■ Declaração

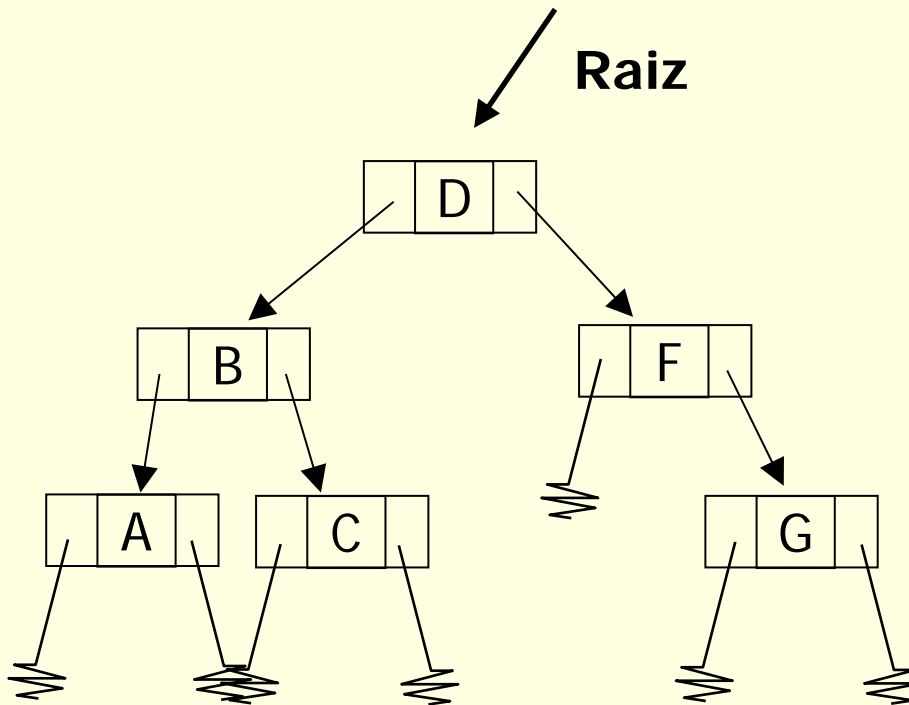
```
typedef char elem;
```

```
typedef struct bloco {  
    elem info;  
    struct bloco *esq, *dir;  
} no;
```

```
typedef struct {  
    no *raiz;  
} Arvore;
```

Árvores binárias

- Representação dinâmica e encadeada de uma árvore binária



Árvores binárias

■ Exercício

- Implementar o TAD árvore binária: dinâmico e encadeado
 - Criar árvore
 - Verificar se a árvore está vazia
 - Buscar um elemento
 - Buscar pai de um elemento
 - Inserir elemento à esquerda de outro elemento
 - Inserir elemento à direita de outro elemento
 - Imprimir elementos da árvore
 - Determinar altura da árvore
 - Simulação da execução da função
 - Finalizar árvore

Exercício

- Em duplas
 - Esquematize/desenhe/explique como seria a função de remoção de um elemento da árvore
 - Implemente a função de remoção

Percurso em árvores binárias

- **Percorrer uma árvore** visitando cada nó uma única vez gera uma sequência linear de nós
 - Listagem de todos os elementos
 - Busca por um elemento
- Passa a ter sentido falar em sucessor e predecessor de um nó segundo um determinado percurso

Percurso em árvores binárias

- Há três maneiras de se percorrer árvores binárias
 - Em função da ordem de visitas aos nós
 - **Pré-ordem**: visita-se nó raiz primeiro e depois as subárvores esquerda e direita, nessa ordem
 - **Em-ordem**: visita-se subárvore esquerda, nó raiz e subárvore direita, nessa ordem
 - **Pós-ordem**: visita-se subárvore esquerda, subárvore direita, e, depois, o nó raiz, nessa ordem

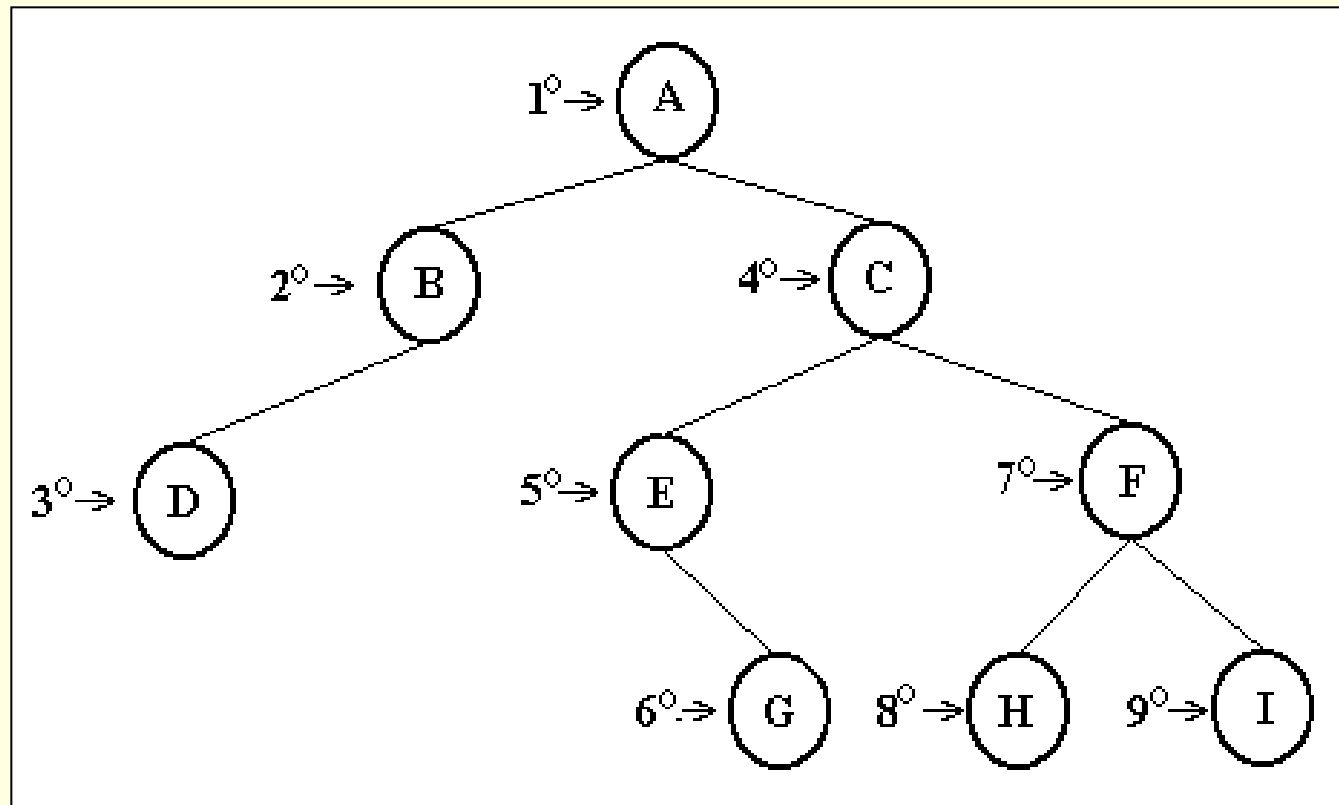
Percurso em árvores binárias

■ Pré-ordem

1. se árvore vazia, então fim
2. visitar o nó raiz
3. percorrer em pré-ordem a subárvore esquerda
4. percorrer em pré-ordem a subárvore direita

Percurso em árvores binárias

■ Pré-ordem



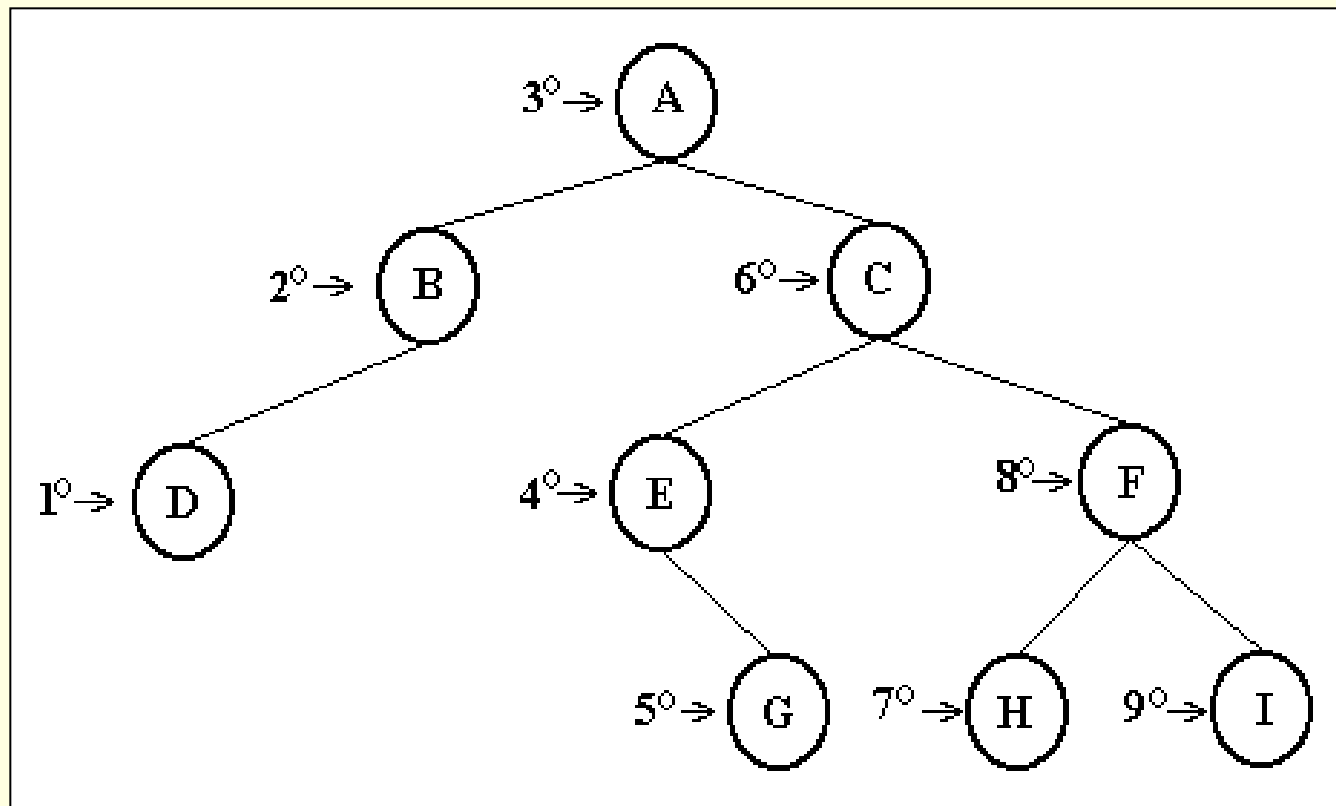
Percurso em árvores binárias

■ Em-ordem

1. se árvore vazia, então fim
2. percorrer em em-ordem a subárvore esquerda
3. visitar o nó raiz
4. percorrer em em-ordem a subárvore direita

Percurso em árvores binárias

- Em-ordem



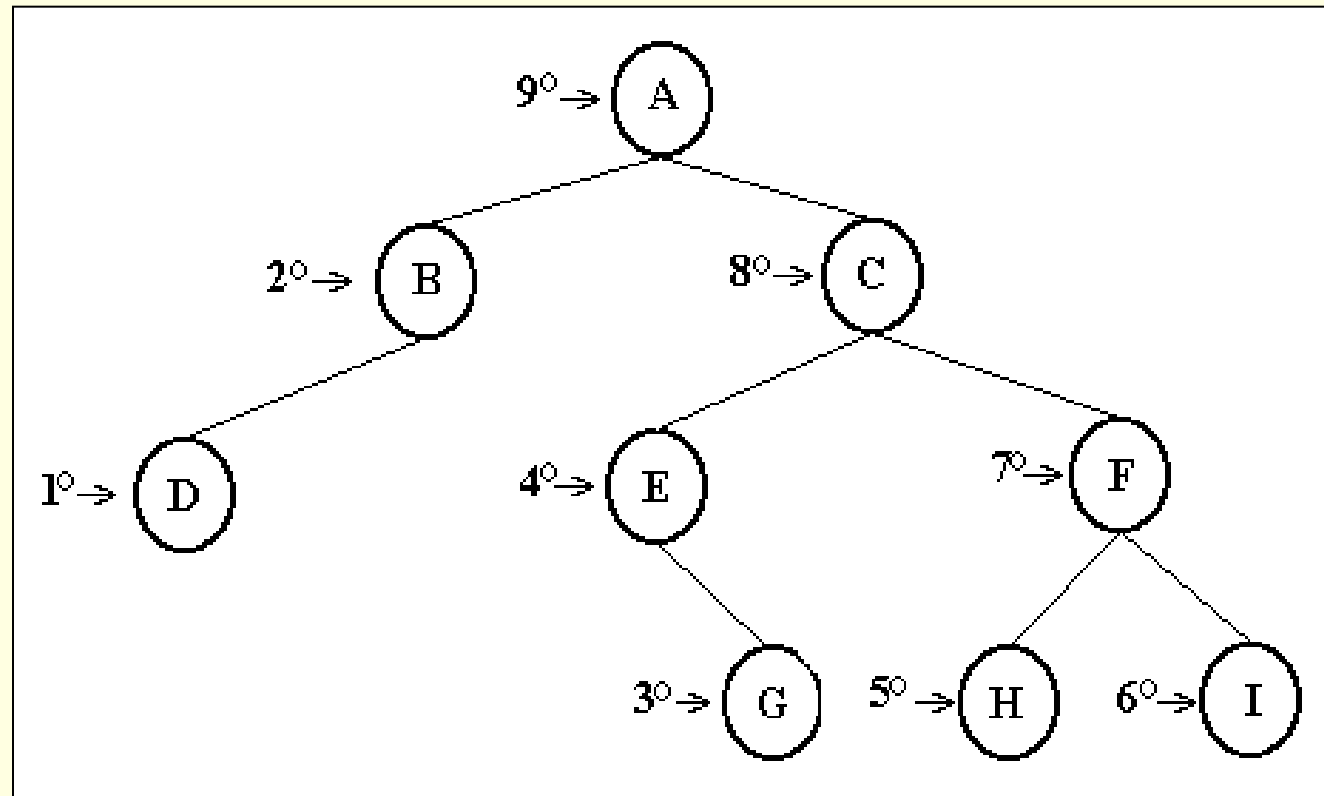
Percurso em árvores binárias

■ Pós-ordem

1. se árvore vazia, então fim
2. percorrer em pós-ordem a subárvore esquerda
3. percorrer em pós-ordem a subárvore direita
4. visitar o nó raiz

Percurso em árvores binárias

■ Pós-ordem



Percurso em árvores binárias

- Exercícios

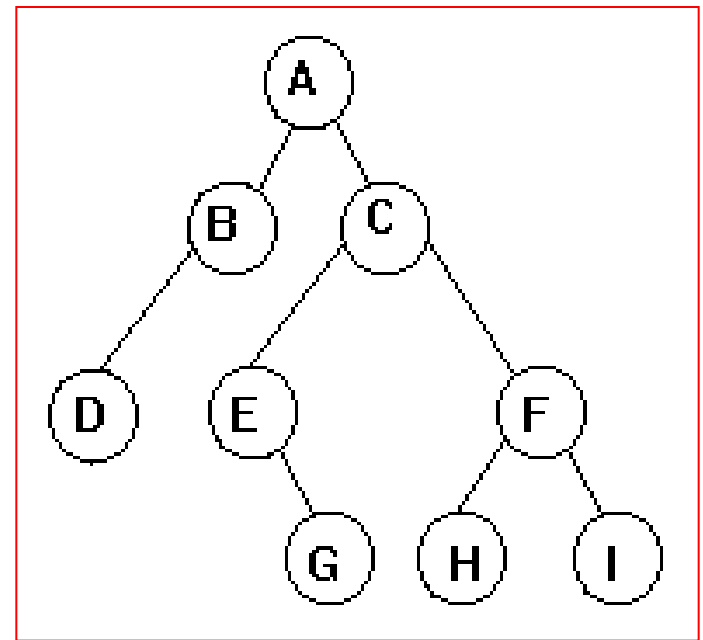
- Implementar sub-rotinas recursivas de percurso pré-ordem, em-ordem e pós-ordem

Percurso em árvores binárias

- Possível implementação não recursiva para o percurso em-ordem

```
...  
#include "pilha.h"  
...  
  
void EmOrdem(no *raiz) {  
    no *p=raiz;  
    Pilha s;  
    cria_pilha(s);  
    do  
        while (p!=NULL) {  
            push(s,p);  
            p=p->esq;  
        }  
        if (!IsEmpty(s)) {  
            pop(s,p);  
            printf("%d\n",p->info);  
            p=p->dir;  
        }  
    while ((!IsEmpty(s)) || (p!=NULL));  
}
```

Execute a sub-rotina para
a árvore abaixo

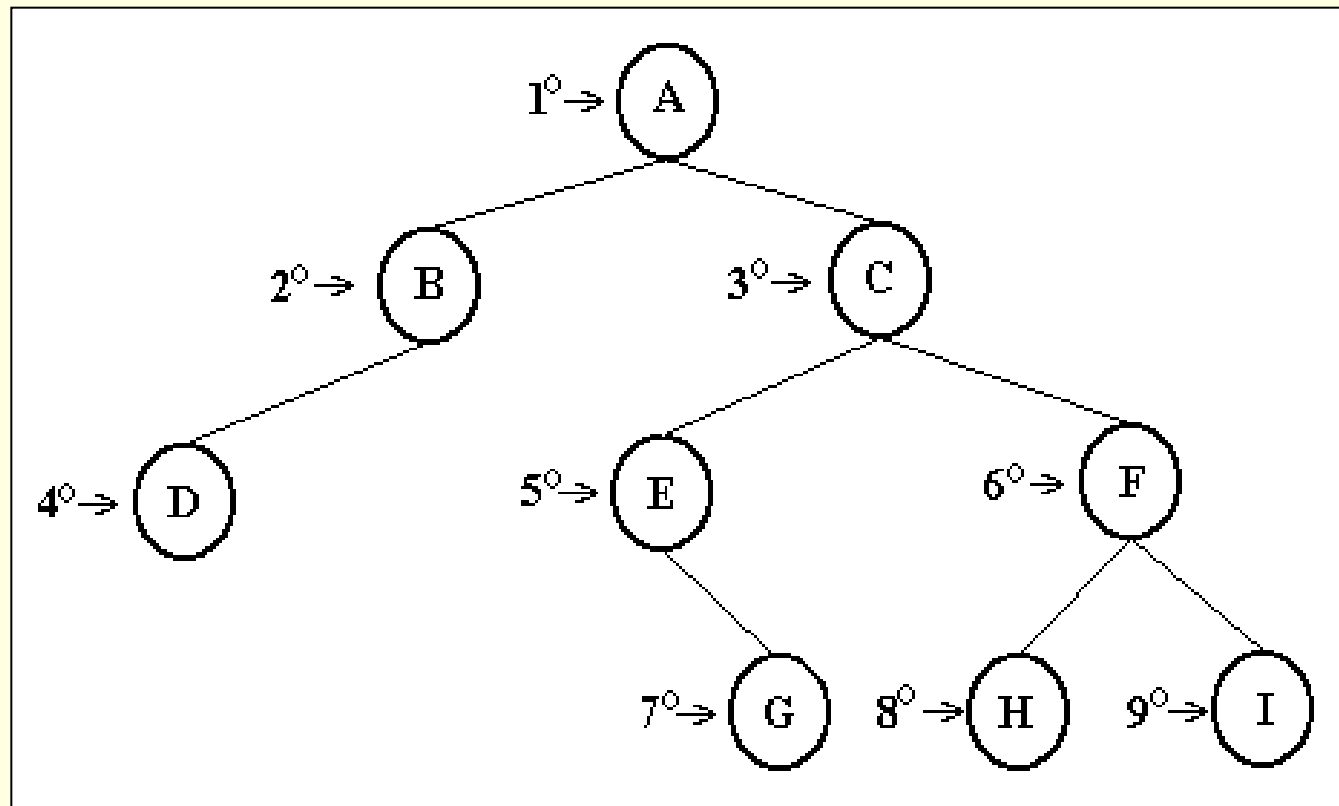


Percurso em árvores

- Outras formas/nomenclaturas
 - Busca/percurso em largura
 - Busca/percurso em profundidade
- Independente do tipo de árvore
- Tradicionalmente da esquerda para a direita

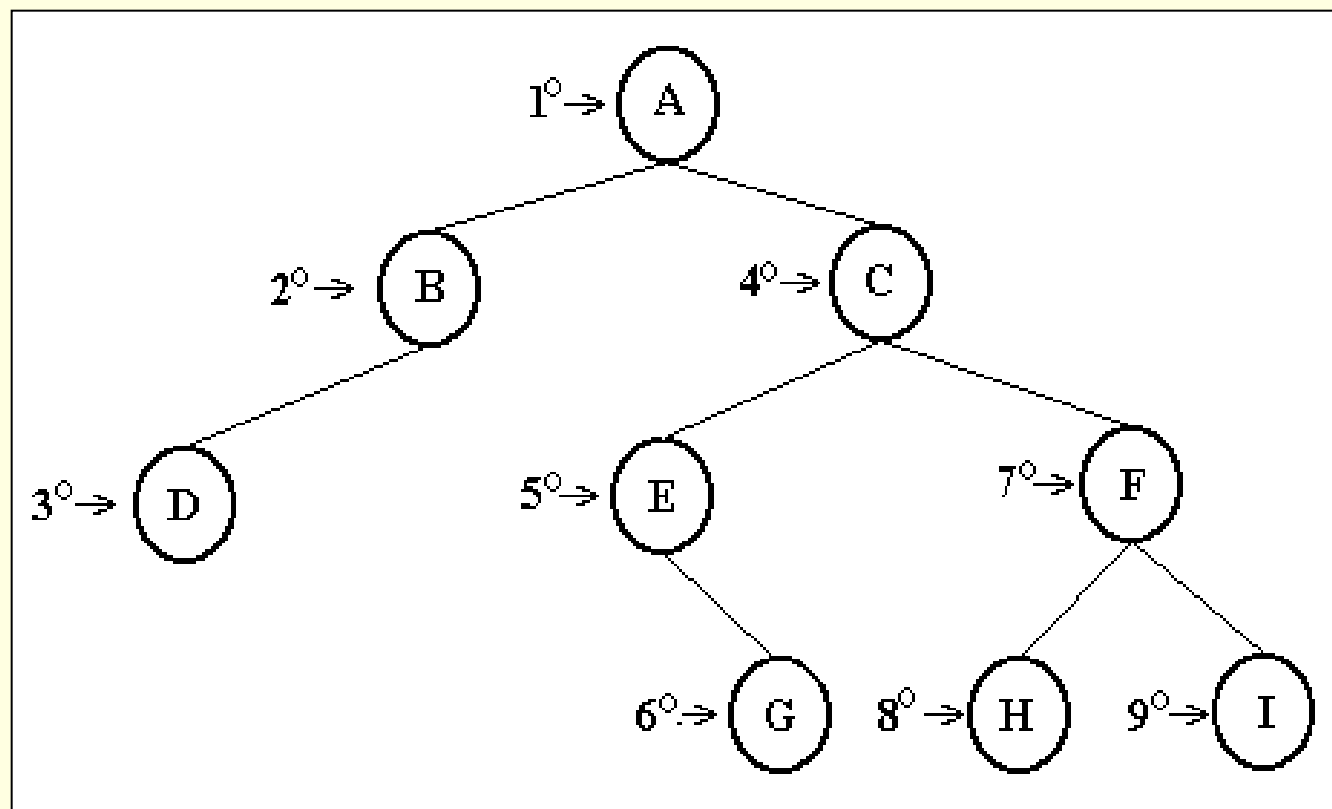
Percurso em árvores

- Em largura
 - Um nível de cada vez



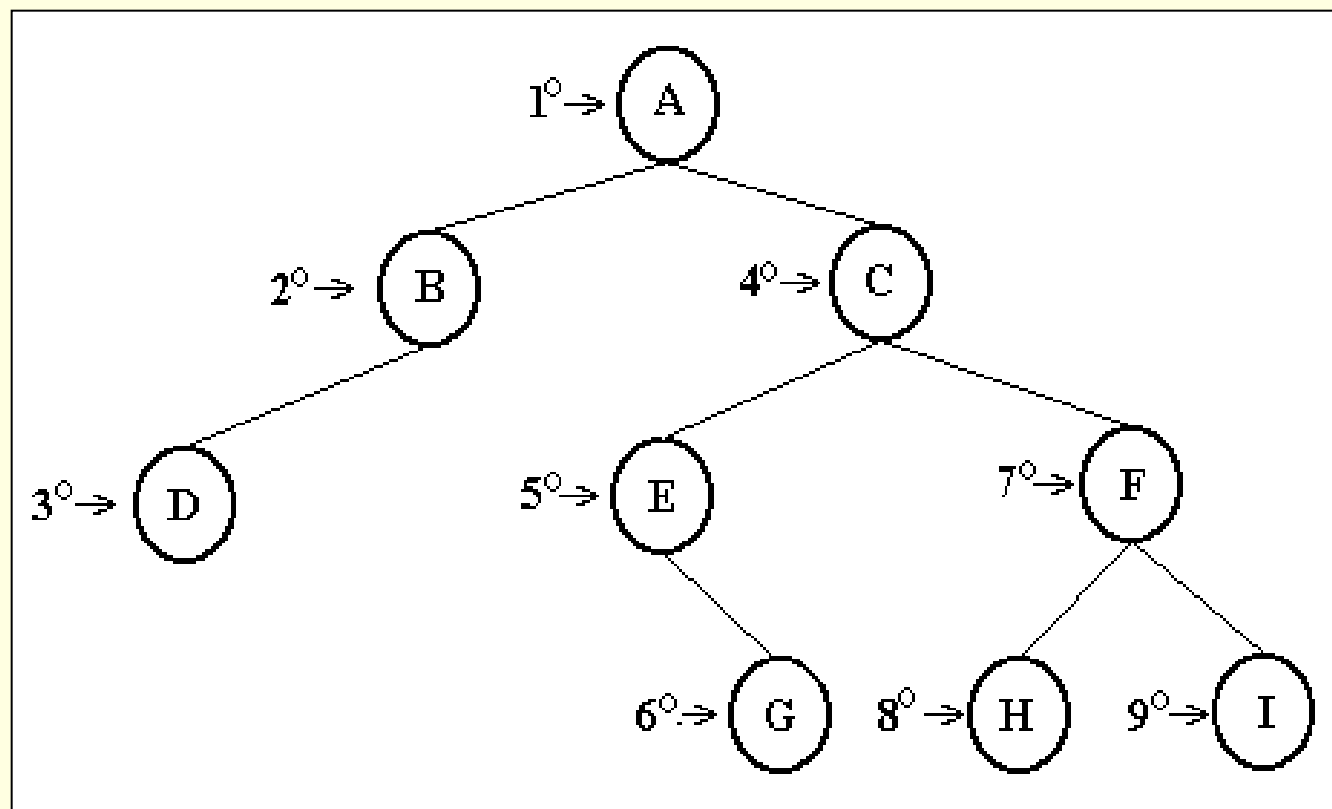
Percurso em árvores

- Em profundidade
 - Um ramo da árvore de cada vez (= ???)



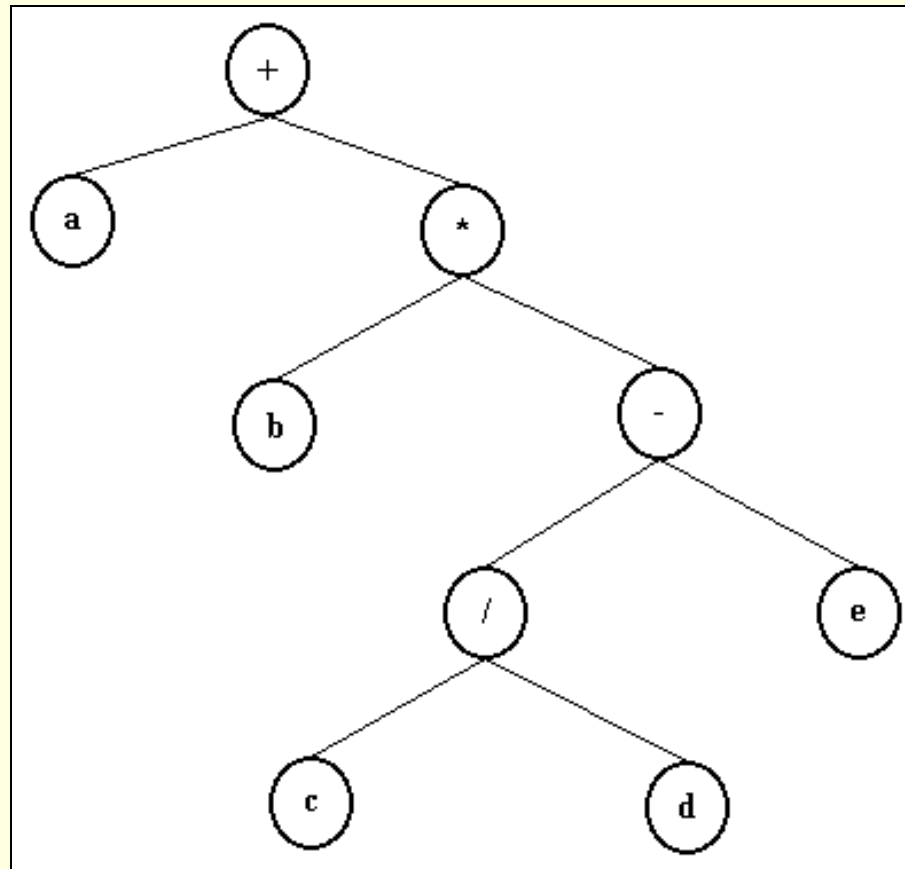
Percurso em árvores

- Em profundidade
 - Um ramo da árvore de cada vez (= pré-ordem)



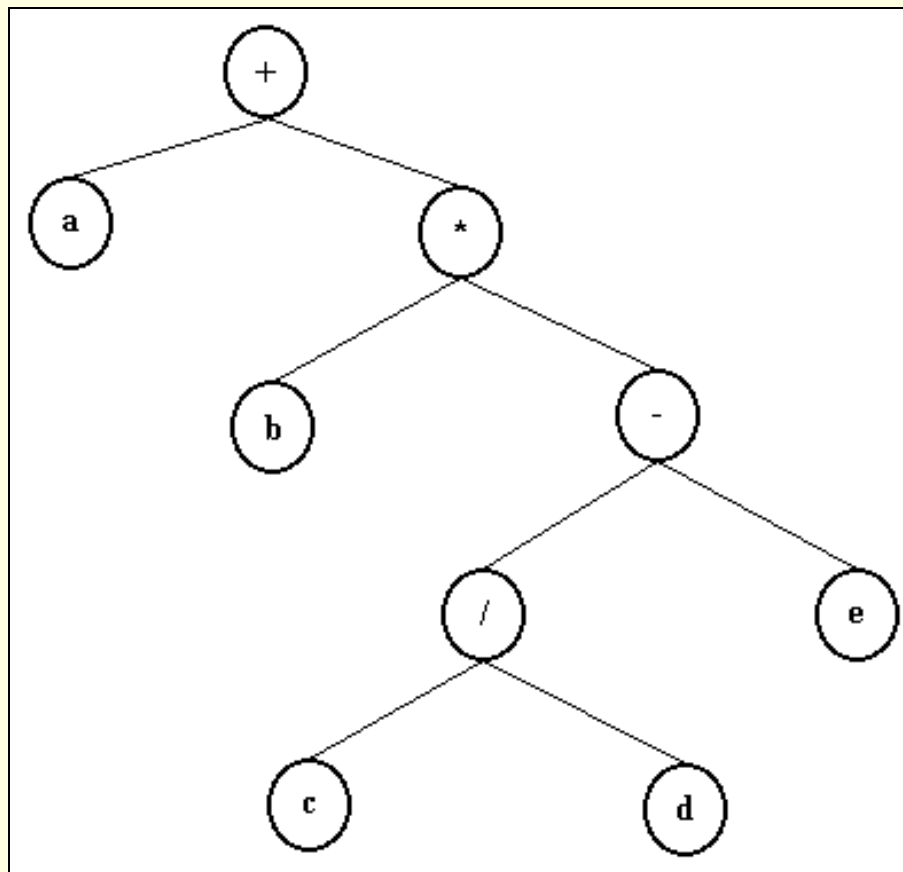
Exercício: percurso em árvores

- Para a árvore abaixo, mostre quais seriam as saídas para os percursos em largura, profundidade (pré-ordem), em-ordem e pós-ordem



Exercício: percurso em árvores

- Para a árvore abaixo, mostre quais seriam as saídas para os percursos em largura, profundidade (pré-ordem), em-ordem e pós-ordem



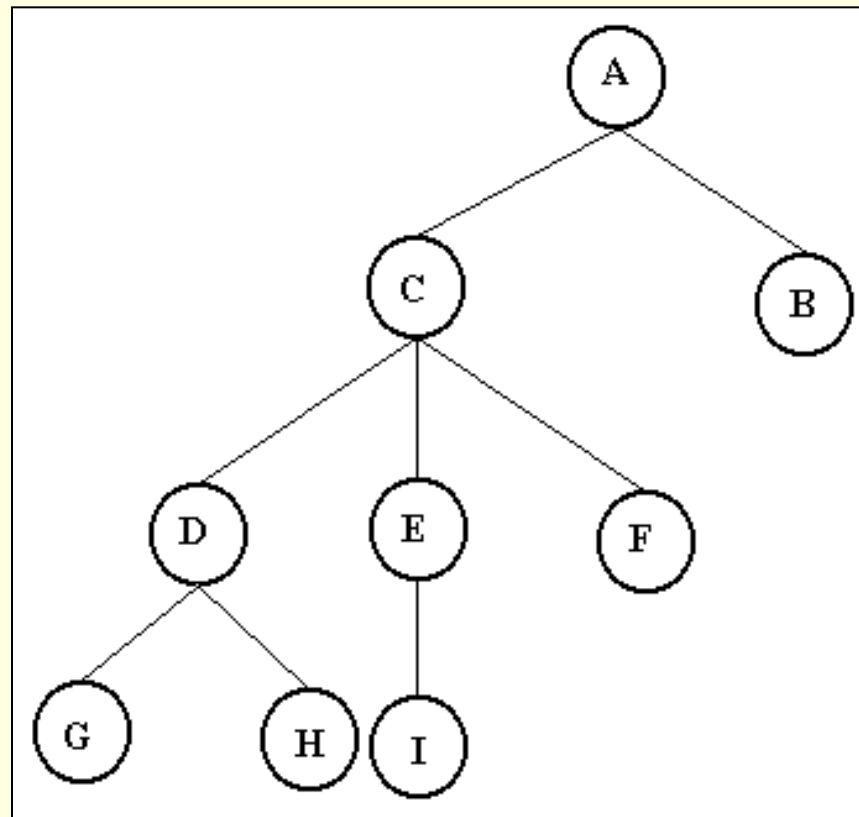
Qual é o percurso necessário para se resolver a expressão?

Percurso em árvores

- Em **profundidade** = pré-ordem
 - Usa **pilha** (explícita ou implicitamente)

Percurso em árvores

- Teste a estratégia com a pilha explícita



Percurso em árvores

- Em **largura**
 - Como implementar?

Percurso em árvores

- Em **largura**

- **FILA**

- Se o nó raiz for diferente de NULL, ele entra na fila

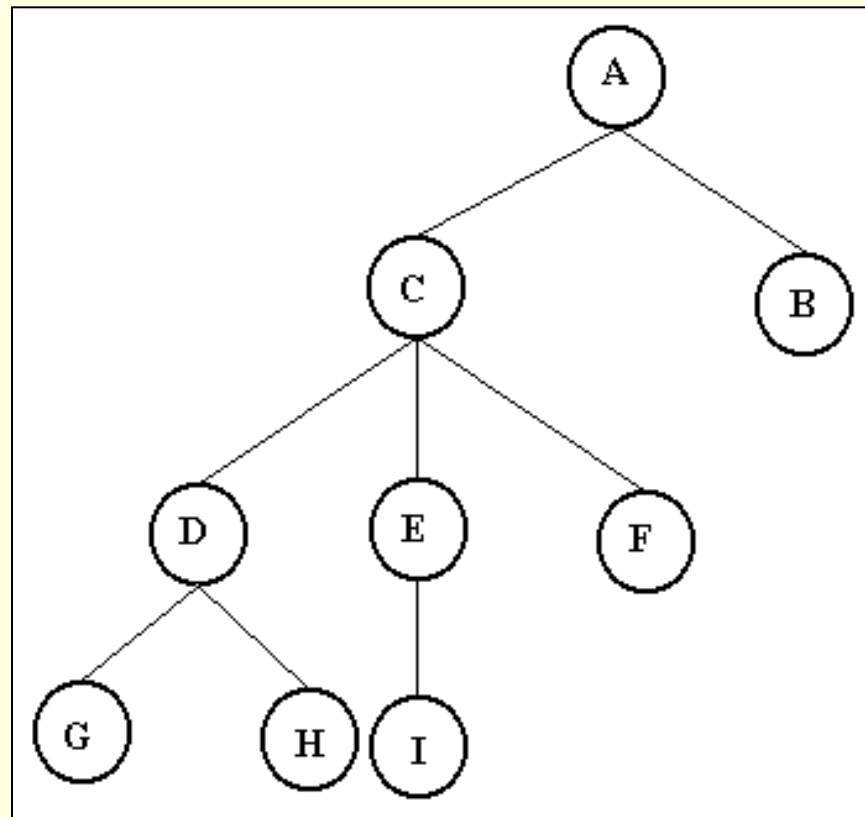
- Enquanto fila não vazia

- Retira-se/visita-se o primeiro da fila

- Se houver filhos desse nó, eles entram na fila

Percurso em árvores

- Teste a estratégia



Largura vs. profundidade

- Ao buscar um elemento
 - **Profundidade**
 - Vantagens?
 - Desvantagens?

Largura vs. profundidade

- Ao buscar um elemento

- **Profundidade**

- Menos memória para guardar nós não visitados
- Pode buscar “para sempre”, demorando mais para achar o elemento

Largura vs. profundidade

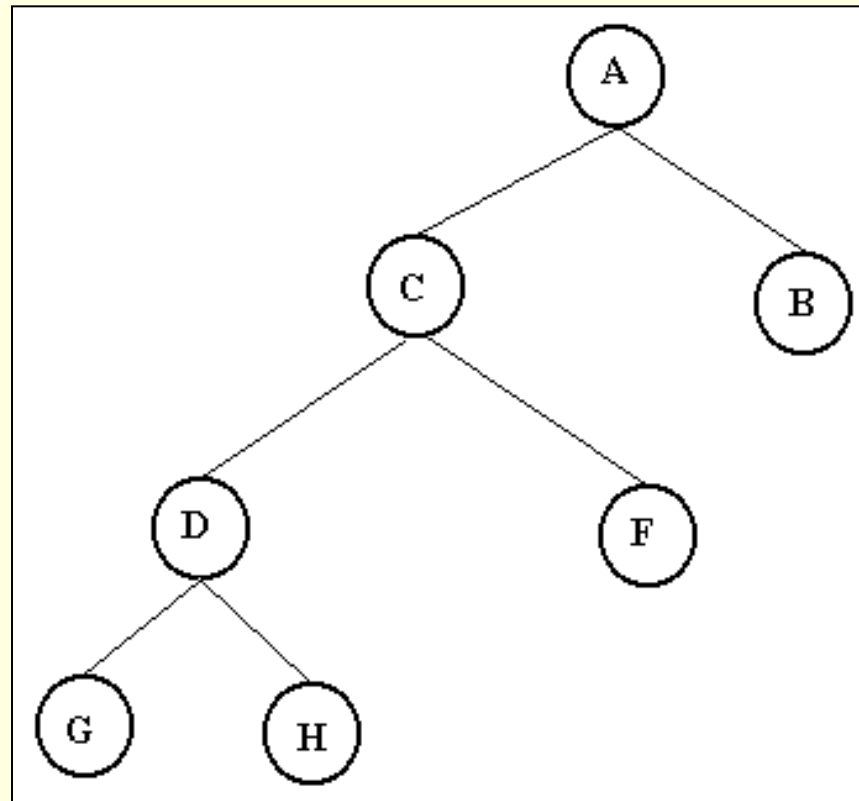
- Ao buscar um elemento
 - **Largura**
 - Vantagens?
 - Desvantagens?

Largura vs. profundidade

- Ao buscar um elemento
 - **Largura**
 - Acha o elemento mais rapidamente
 - Mais memória para guardar nós não visitados

Largura vs. profundidade

- Use as estratégias para buscar o nó F
 - Quem se sai melhor em termos de nós visitados e de memória?



Percurso em árvores

- Árvores com encadeamentos variados
 - Mais ponteiros, mas pode facilitar busca e outras operações
 - Ponteiros podem ser configurados para qualquer tipo de busca
 - Filhos podem apontar para pais, para irmãos, para avôs, etc.

