UNIVERSIDADE FEDERAL DE JUIZ DE FORA

Departamento de Ciência da Computação

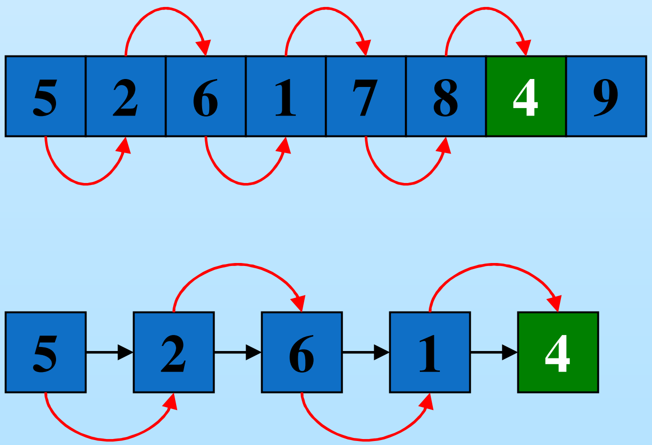
ESTRUTURA DE DADOS

DCC013

Árvores

Preliminares

- Pesquisa sequencial
  - Elementos pesquisados sucessivamente
  - Comparação determina se o elemento foi encontrado ou não
  - Exemplo: buscar 4 (Arrays e lista encadeada)



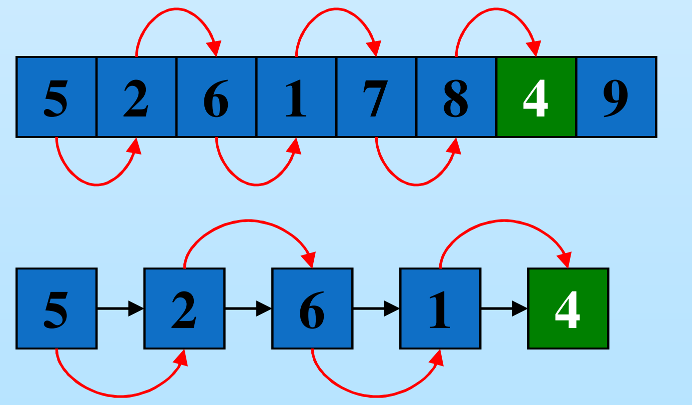
Tempo de busca: O(n)

Comparações: (n + 1) / 2

Busca sequencial

- Pesquisa sequencial em vetor não ordenado
  - Necessário pesquisar em todo o vetor:

```
tipo vet = vetor[1..tamanho_vetor]tipo-t;
funcão Busca(vet: v, tipo-t: chave):int
   int: I;
   I ← 1;
   enquanto vetor[I] ≠ chave e I <= tamanho_vetor
  faca
          I ← I + 1;
   fim-enquanto;
   se I <= tamanho_vetor entao
          Busca ← I;
   senao
          Busca ← -1;
   fim-se;
fim-funcão;
```



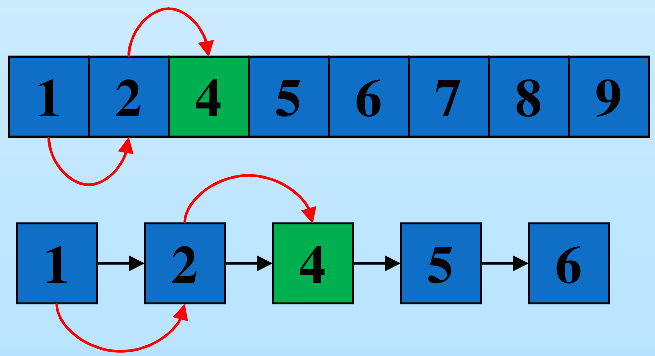
Tempo de busca: O(n)

Comparações: (n + 1) / 2

Busca sequencial

- Pesquisa sequencial em vetor ordenado (crescente)
  - Caso encontre um número maior que o buscado, pare.

```
tipo vet = vetor[1..tamanho_vetor]tipo-t;
funcão Busca(vet: v, tipo-t: chave):int
    int: I;
    I ← 1;
    enquanto chave > vetor[I] e I <= tamanho_vetor
  faca
       I ← I + 1;
    fim-enquanto;
    se chave = vetor[I] entao
       Busca ← I;
    senao
       Busca ← -1;
    fim-se;
fim-funcão;
```



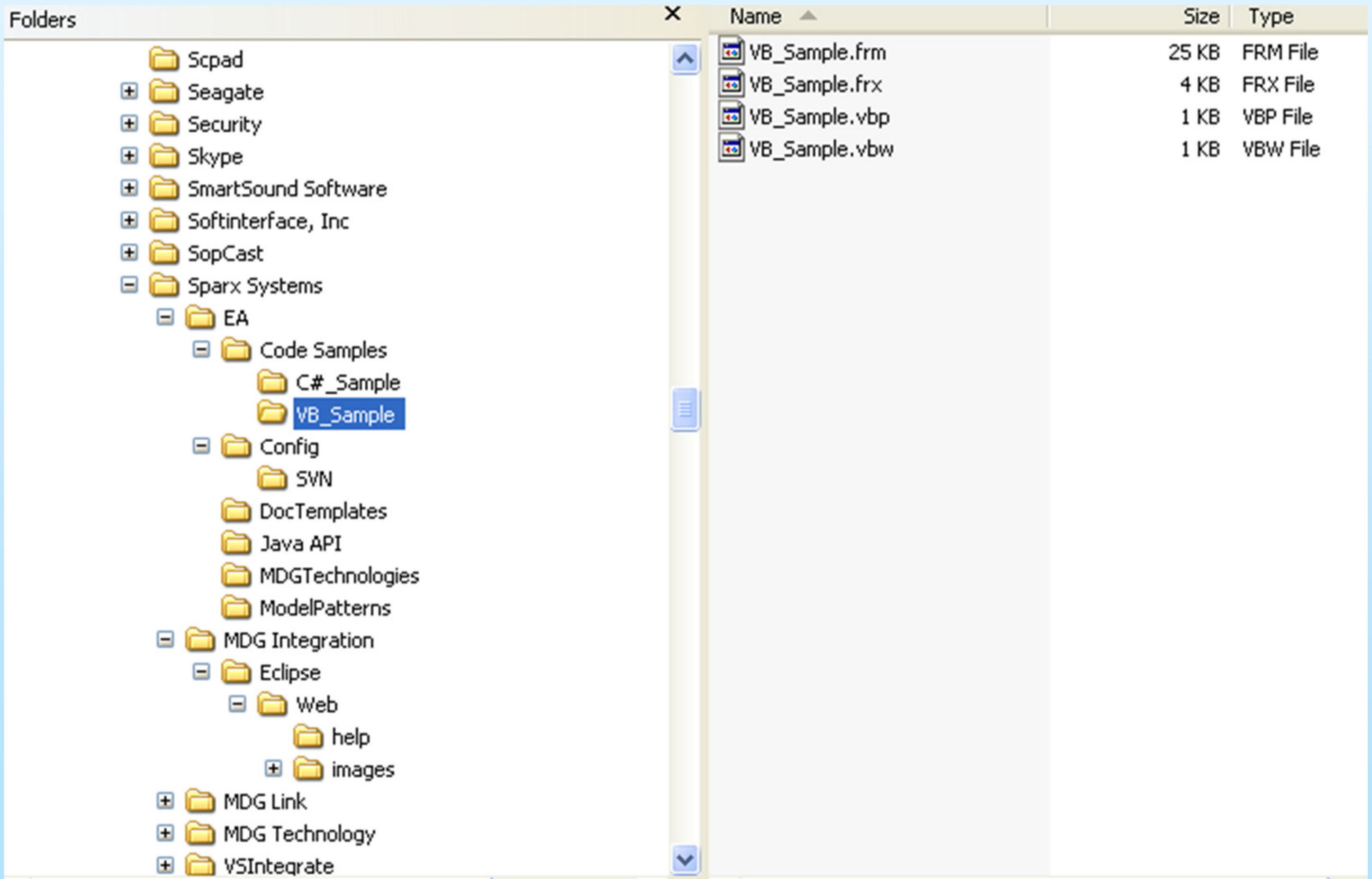
Ainda assim...

Tempo de busca: O(n)

Comparações: (n + 1) / 2

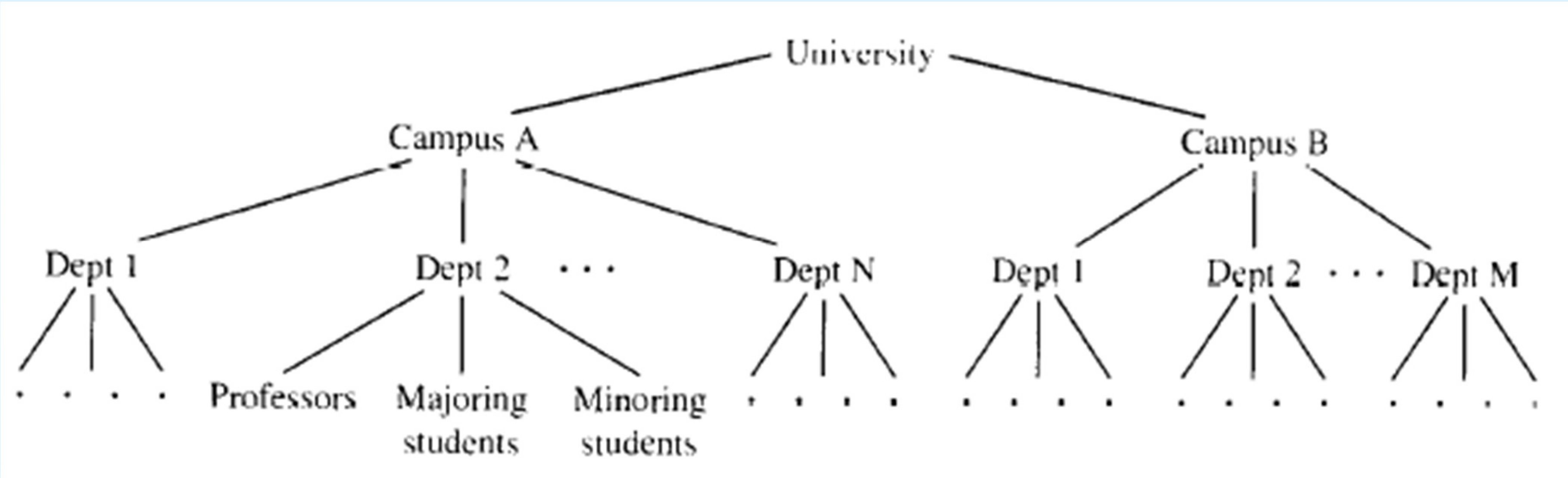
Árvores: Motivação

- Qual estrutura de dados o Windows Explorer deve utilizar para gerenciar os arquivos?



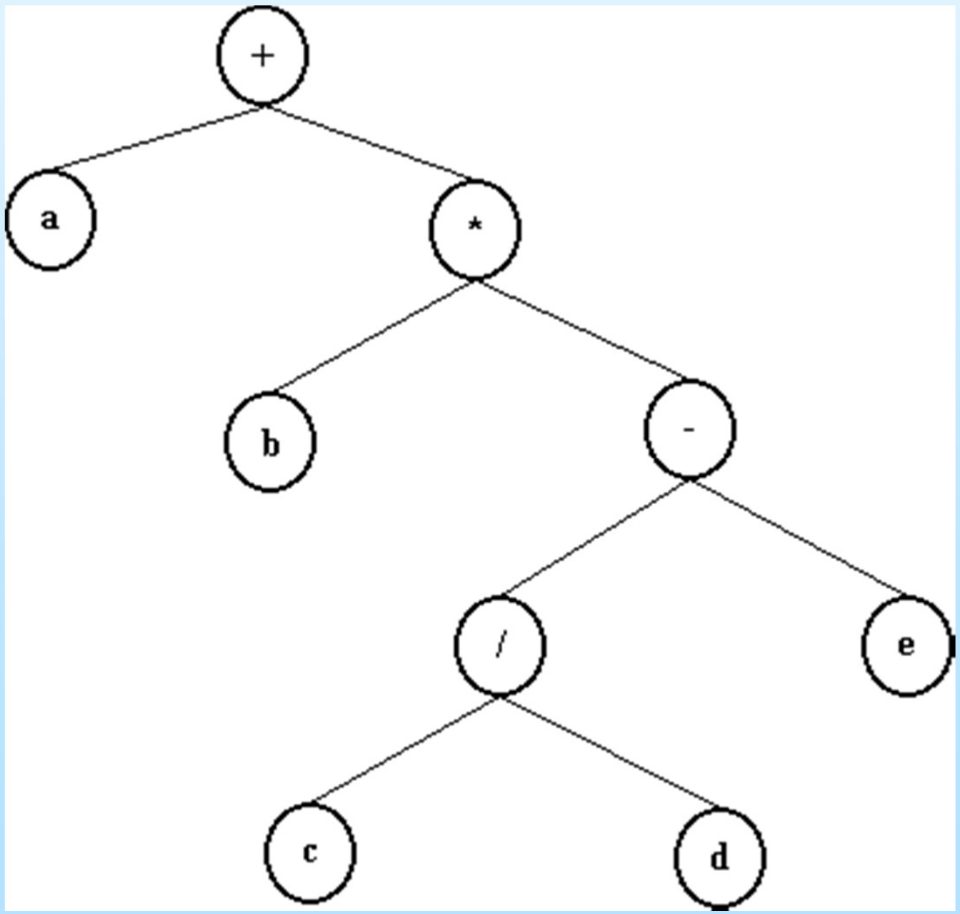
Árvores: Motivação

- Ex: Hierarquia Universitária



Árvores: Motivação

Representação da expressão aritmética: (a + (b *((c / d) - e)))

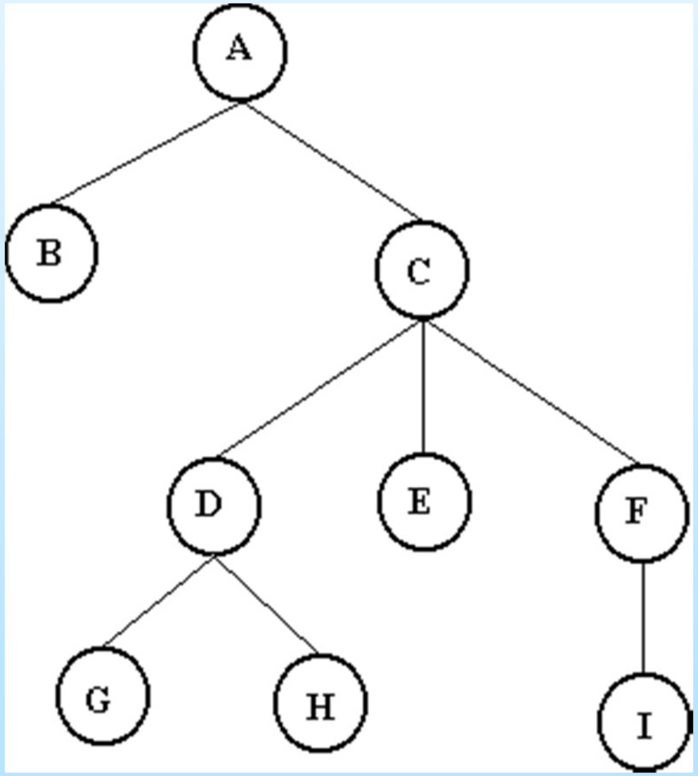


Árvores: Motivação

- Listas ligadas
  - São mais flexíveis do que matrizes;
  - Mas são estruturas lineares sendo difícil utilizá-las para organizar representação hierárquica de objetos.
- Pilhas e filas
  - Refletem alguma hierarquia;
  - Mas são limitadas a somente uma dimensão.
- Árvore
  - Estrutura criada para superar limitações de listas ligadas, pilhas e filas;
  - Consiste de nós e de arcos;
  - São representadas com a raiz no topo e as folhas na base (*diferente de árvore natural*).

Árvores: Representação

- Hierárquica



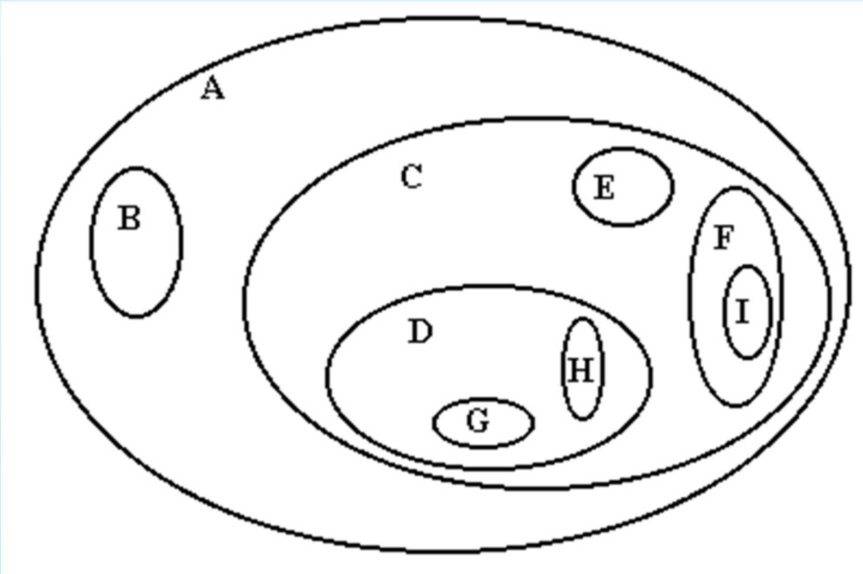
- Alinhamento dos nós

A
B
C
D
G
H
E
F
I

Árvores: Representação

- Parênteses aninhados
  (A (B) (C (D (G) (H)) (E) (F (I))))

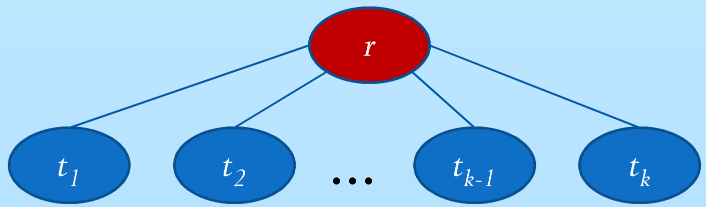
- Diagramas de inclusão



Árvores: Definição

- Raiz
  - Não possui ancestrais
  - Só pode ter filhos
- Folhas
  - Não tem filhos (ou melhor, seus filhos são estruturas vazias)
- Definição recursiva de árvore

1. Uma estrutura vazia é uma árvore vazia.
2. Se $t_1,\ldots,t_k$ são árvores disjuntas, então a estrutura cuja raiz r tem como suas filhas as raízes $t_1,\ldots,t_k$ também é uma árvore.

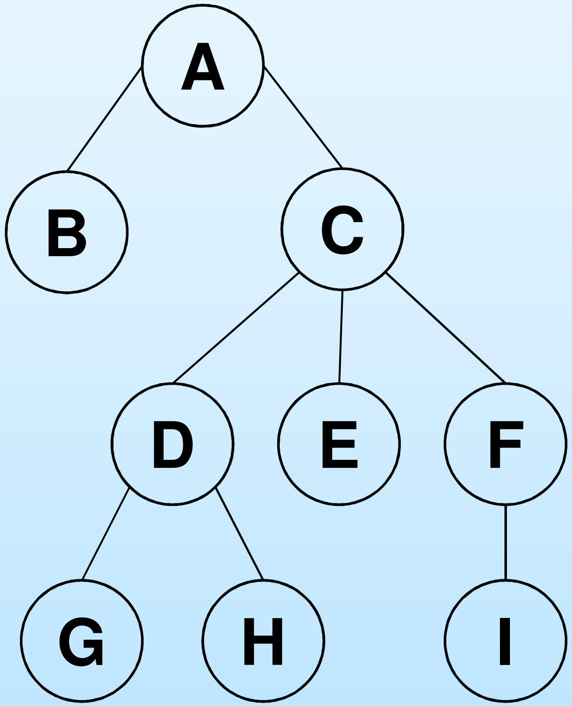


3. Somente estruturas geradas pelas regras 1 e 2 são árvores.

Elementos de uma árvore

- **Nó**: Elemento que contém a informação
- **Arco**: Liga dois nós
- **Pai**: nó superior de um arco
- **Filho**: nó inferior de um arco
- **Raiz**: nó topo – não tem um nó pai
- **Folhas**: nós das extremidades inferiores – não têm nós filhos.
- **Grau**: Representa o número de subávores de um nó. Ver exemplo no próximo slide.
- **Grau de uma árvore** (aridade): é definido como sendo igual ao máximo dos graus de todos os seus nós. A árvore do próximo slide tem grau 3.

Elementos de uma árvore



- **Graus dos nós**
  - **G(A)=2**
  - **G(B)=0**
  - **G(C)=3**
  - **G(D)=2**
  - **G(E)=0**
  - **G(F)=1**
  - **G(G)=0**
  - **G(H)=0**
  - **G(I)=0**

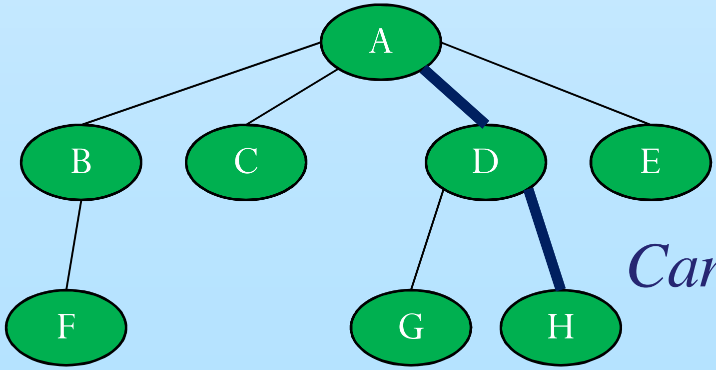
Nós Internos

Folhas

Grau(T) = 3

Árvore

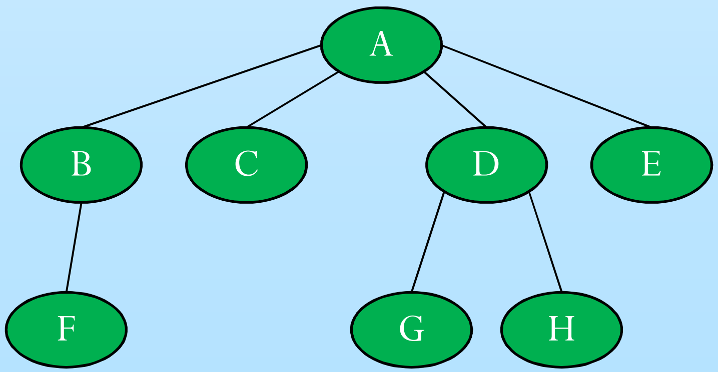
- Cada nó tem que ser atingível a partir da raiz através de uma sequência única de arcos, chamados de **caminho**.
- Comprimento do caminho: o número de arcos do caminho
  - O caminho de A até H tem comprimento 2
- **Nível de um nó**: é a sua distância da raiz da árvore. A raiz tem nível 0. Na fig. abaixo, o nó *A* tem nível 0; os nós *B*, *C*, *D* e *E* têm nível 1 e os nós F, G e H têm nível 2.



Caminho do nó A ao H.

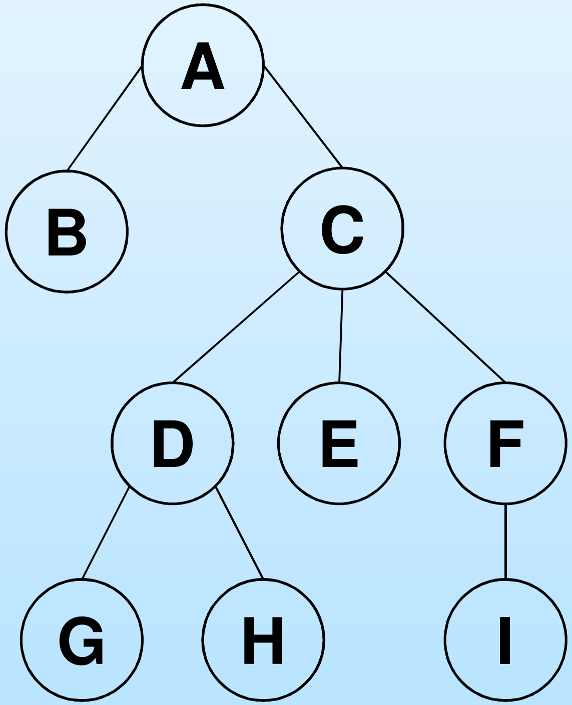
Árvore

- **Altura** (ou **profundidade**) é o nível do nó folha que tem o mais longo caminho até a raiz, somando 1.
  - A altura da árvore abaixo é igual a 3.
  - A árvore vazia é uma árvore de altura -1, por definição.
  - Uma árvore com um único nó tem altura 1.
    - O nó é raiz e folha ao mesmo tempo.
- Toda árvore com n>1 nós possui no mínimo 1 e no máximo n-1 folhas.



Árvore

Exemplo de níveis e altura da árvore)

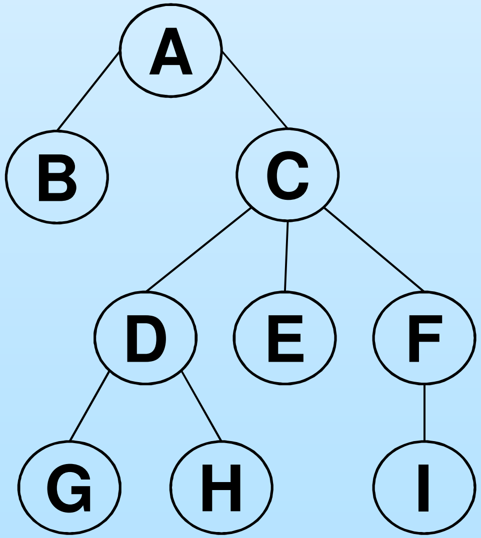


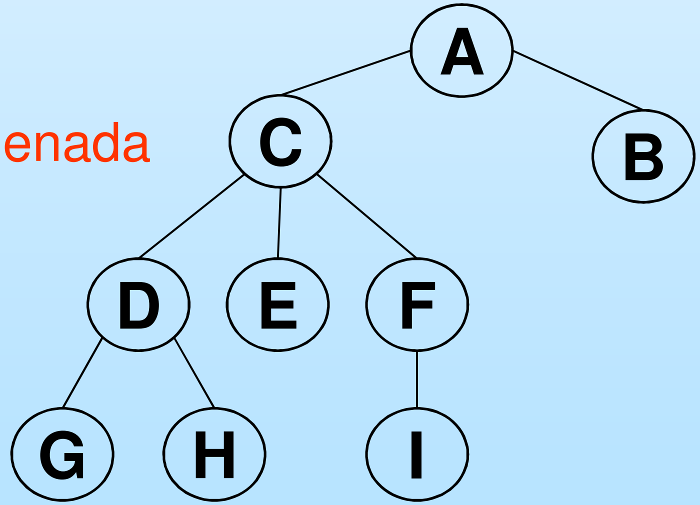
NÍVEIS	
A	0
B, C	1
D, E, F	2
G, H, I	3

h(T) = 4

Árvore

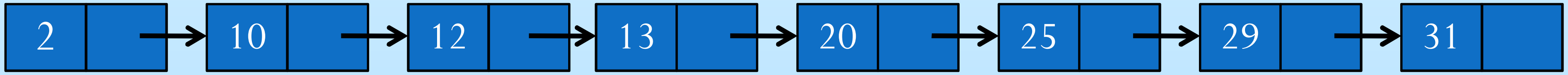
- Árvore Ordenada
  - Os filhos de cada nó estão ordenados (assume-se ordenação da esquerda para a direita)

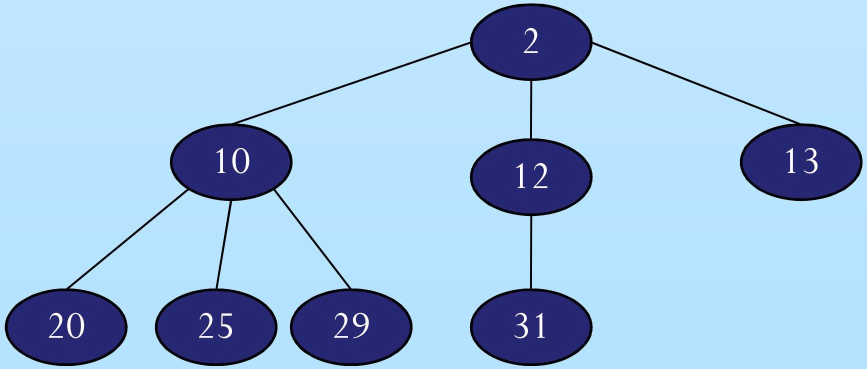




Árvore

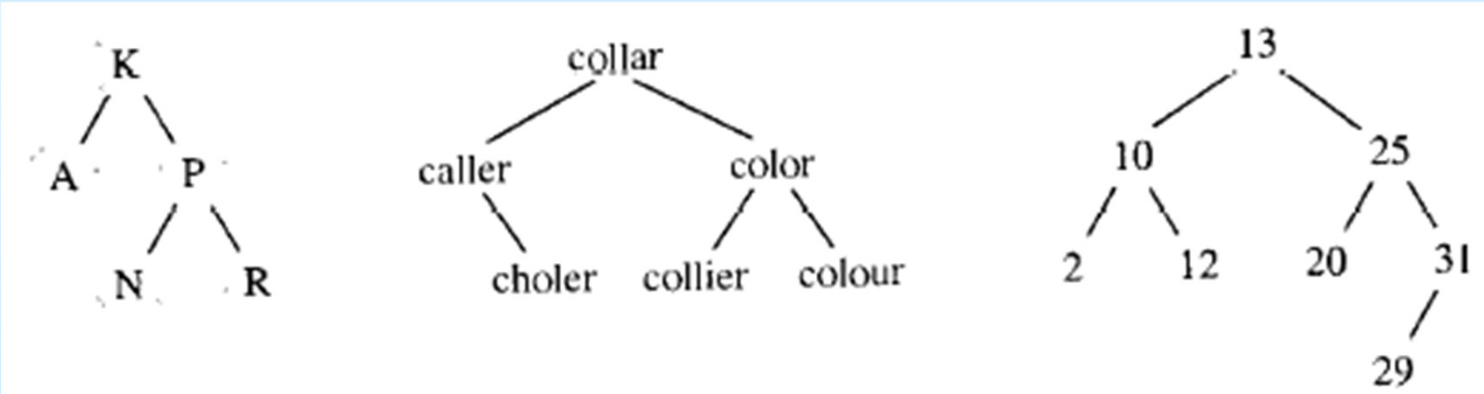
- A definição de árvore não impõe qualquer condição sobre o número de filhos de um nó:
  - Pode variar de 0 a qualquer inteiro
- Árvores são muito utilizadas em sistemas gerenciadores de banco de dados.
- Considerando a lista encadeada e a árvore abaixo, qual pesquisa é mais rápida para achar um valor (chave)?





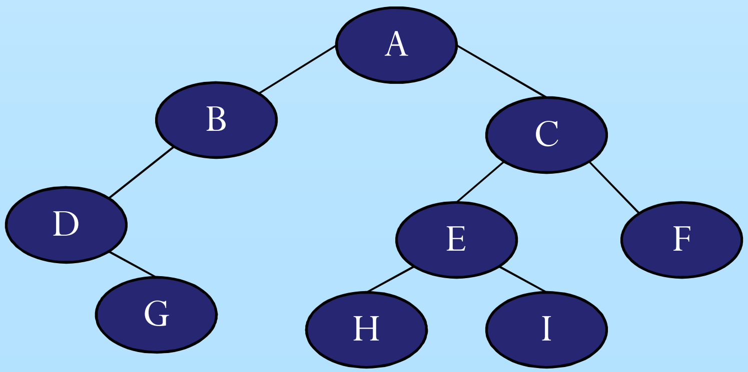
Árvore binária

- Uma árvore binária é uma árvore cujos nós tem dois filhos (alguns vazios) e cada filho é designado como filho à esquerda ou filho à direita. Portanto, a árvore binária tem grau máximo 2.
- Nó filho ESQUERDO e Nó filho DIREITO.



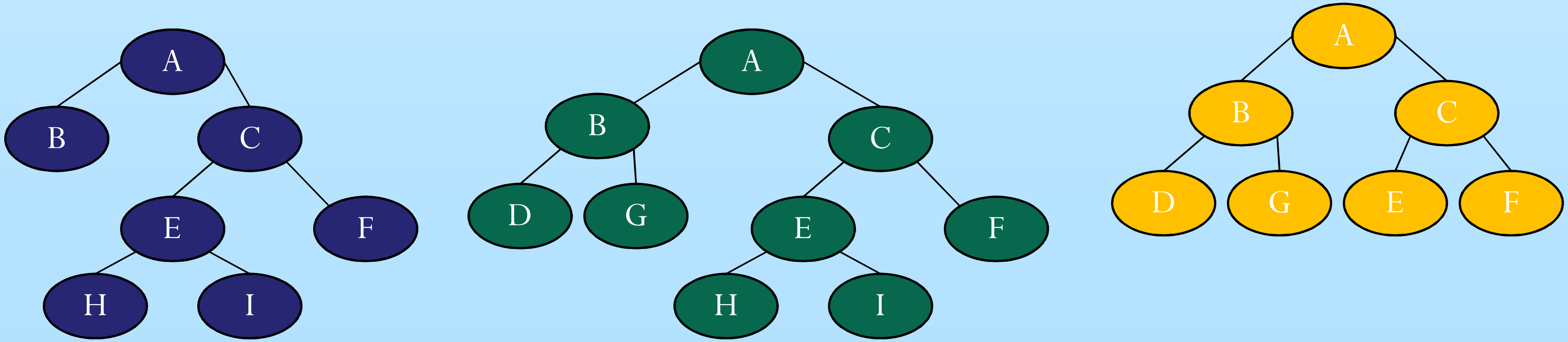
Árvore binária

- Definição
  - Uma *árvore binária T* é um conjunto finito de elementos denominados *nós* ou vértices, tal que:
    - $T = \emptyset$ e a árvore é dita vazia, ou
    - Existe um nó especial *r*, chamado *raiz* de T, e os restantes podem ser divididos em dois subconjuntos disjuntos, T_r^L e T_r^R a subárvore esquerda e a direita de *r*, respectivamente, as quais são também árvores binárias.



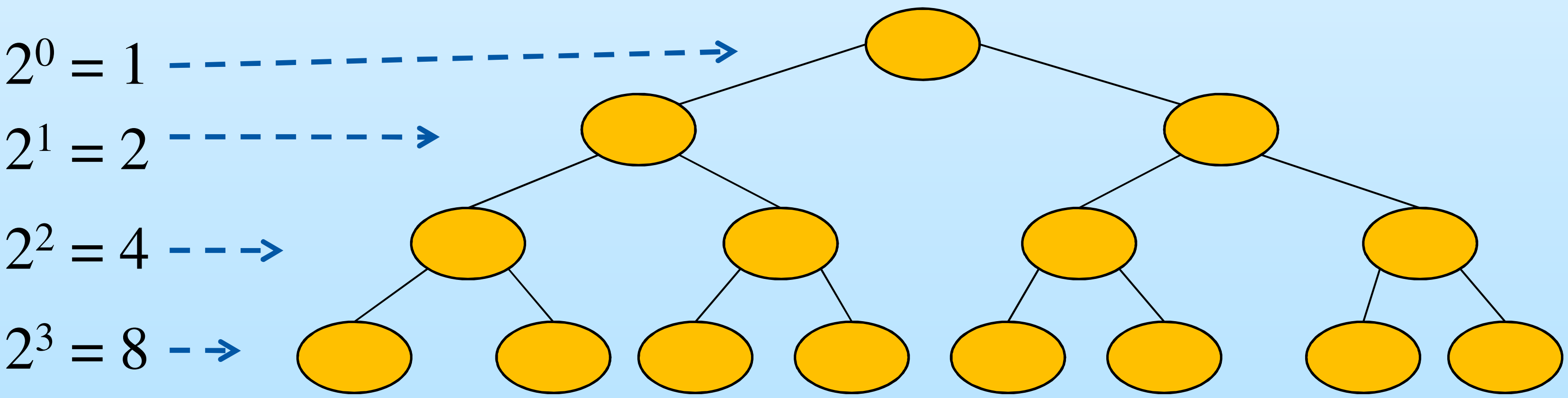
Árvore binária

- **Árvore estritamente binária**
  - Cada nó possui 0 ou 2 filhos.
- **Árvore binária completa** apresenta a seguinte propriedade
  - Se *v* é um nó tal que alguma subárvore de *v* é vazia, então *v* se localiza ou no último (maior) ou no penúltimo nível da árvore.
- **Árvore binária cheia** apresenta a seguinte propriedade:
  - Se *v* é um nó tal que alguma subárvore de *v* é vazia, então *v* se localiza no último (maior) nível da árvore. *v* é um nó folha.



Árvore binária

- Em árvore binária cheia o número de nós do nível i é igual a 2^i.
- Consequentemente, em qualquer árvore binária existe no máximo 2^i nós no nível i.



Árvore binária

Representação sequencial de árvores binárias

- Árvore binária pode ser implementada como matrizes:
  - Nó: estrutura com um:
    - Campo de informação
    - Dois "ponteiros" para filho esquerdo e direito
  - Raiz localiza-se na primeira célula (índice 0)
  - -1 indica filho nulo

Índice	Info	Esquerda	Direita
0	13	4	2
1	31	6	-1
2	25	7	1
3	12	-1	-1
4	10	5	3
5	2	-1	-1
6	29	-1	-1
7	20	-1	-1

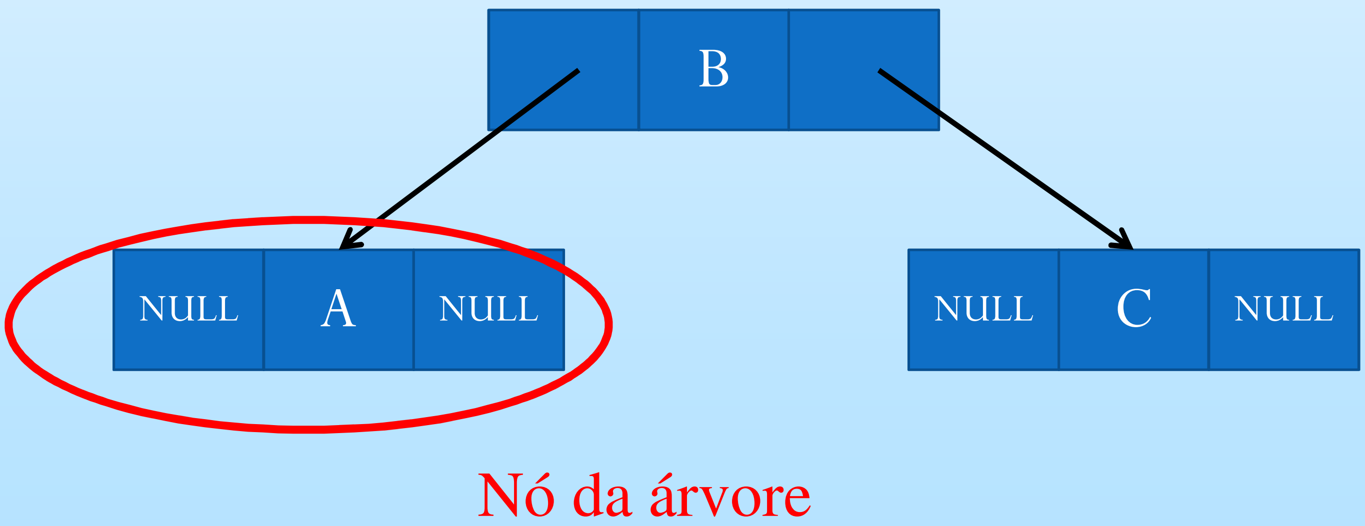
Qual o problema nesta implementação?

Exercício: desenhe a árvore representada acima.

Árvore binária

Representação encadeada de árvores binárias

- Um nó será formado por um registro composto de:
  - Campo de informação
  - Ponteiro para nó esquerdo
  - Ponteiro para nó direito



Árvore binária

Representação encadeada de árvores binárias

- **Definição do tipo de dado árvore binária:**

```
tipo R = ref NO;
tipo NO = reg ( R :  ESQ,  tipot :  X,  R :  DIR );
..................................................
R : RAIZ;
..................................................
```

Árvore binária

Representação encadeada de árvores binárias

- **Inicializa uma árvore vazia (considere os tipos definidos anteriormente)**

```
funcao INICIALIZA:R;
inicio
      INICIALIZA = nil;
fim;
```

- **Verifica se uma árvore está vazia**

```
funcao VAZIA(R: ARV):logico;
inicio
   se ARV = nil entao
      VAZIA ← verdadeiro;
      senao
         VAZIA ← falso;
   fim-se;
fim;
```

Árvore binária

Representação encadeada de árvores binárias

- **Cria uma árvore, dados as suas sub-árvores da esquerda e da direita**

```
funcao CRIA(tipot:C; R:SAE, SAD):R;
inicio
      R:P;
      aloque(P);
      P↑.X ← C;
      P↑.ESQ ← SAE;
      P↑.DIR ← SAD;
      CRIA = P;
fim;
```

Árvore binária

Representação encadeada de árvores binárias

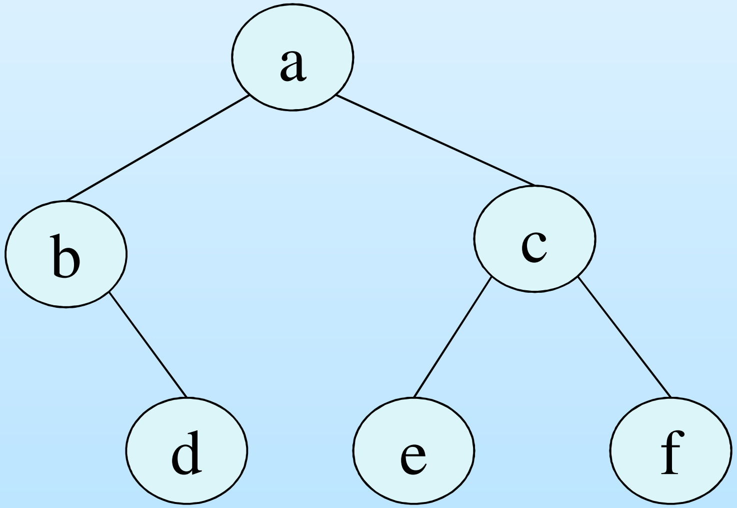
- **Liberando a memória alocada para armazenar uma árvore binária**

```
funcao LIBERA(R:RAIZ):R;
inicio
   se nao VAZIA(RAIZ) entao
       RAIZ↑.ESQ ← LIBERA(RAIZ↑.ESQ);
       RAIZ↑.DIR ← LIBERA(RAIZ↑.DIR);
       desaloque(RAIZ);
       RAIZ ← nil;
   fim-se;
   LIBERA ← nil;
fim;
```

Árvore binária

Representação encadeada de árvores binárias

- **Exemplo: usando as operações inicializa e cria, crie uma estrutura que represente a seguinte árvore.**



- **X (info armazenada em cada nó) será considerado como sendo do tipo `caracter`.**

Árvore binária

Representação encadeada de árvores binárias

- **Exemplo: continuação...**

```
...............................
inicio
      R:A1,A2,A3,A4,A5,A;
      {sub-árvore com raiz ‘d’}
      A1 ← CRIA(‘d’, inicializa, inicializa);
      {sub-árvore com raiz ‘b’}
      A2 ← CRIA(‘b’, inicializa, A1);
      {sub-árvore com raiz ‘e’}
      A3 ← CRIA(‘e’, inicializa, inicializa);
      {sub-árvore com raiz ‘f’}
      A4 ← CRIA(‘f’, inicializa, inicializa);
      {sub-árvore com raiz ‘c’}
      A5 ← CRIA(‘c’, A3, A4);
      {árvore com raiz ‘a’}
      A ← CRIA(‘a’, A2, A5);
      ...............................
      A = LIBERA(A);
fim.
```

Árvore binária

Representação encadeada de árvores binárias

- Vale a pena notar que a definição de árvore, por ser recursiva, não faz distinção entre árvores e sub-árvores. Assim, `cria` pode ser usada para acrescentar ("enxertar") uma sub-árvore em um ramo de uma árvore, e `libera` pode ser usada para remover ("podar") uma sub-árvore qualquer de uma árvore dada.

Árvore binária

Representação encadeada de árvores binárias

- Considerando a criação da árvore feita anteriormente, podemos acrescentar alguns nós, com:

```
A↑.ESQ↑. ESQ ← CRIA('x',
  CRIA('y',INICIALIZA,INICIALIZA),
  CRIA('z',INICIALIZA,INICIALIZA));
```

- E liberar alguns outros com:

```
A↑.DIR↑.ESQ ← LIBERA(A↑.DIR↑. ESQ);
```

- Exercício: verificar o resultado destas operações

Árvore binária

Representação encadeada de árvores binárias

- A função a seguir tem como retorno um valor lógico indicando a ocorrência ou não do caractere `C` na árvore.

```
funcao BUSCA(R: A, car: C):log
inicio
   se VAZIA(A) entao
        BUSCA ← falso
   senao se A↑.X = C entao
        BUSCA ← verdadeiro;
   senao se BUSCA(A↑.ESQ),C) entao
        BUSCA ← verdadeiro;
   senao
        BUSCA ← BUSCA(A↑.DIR,C);
   fim-se; {fim de todos se}
fim.
```

Árvore binária

Percurso em árvores binárias

- Percurso corresponde a uma visita sistemática a cada um dos nós da árvore
  - Esta é uma das operações básicas relativas à manipulação de árvores
- Uma árvore é essencialmente uma estrutura não sequencial
  - Pode ser utilizada em aplicações que demandem acesso direto
- Em qualquer tipo de aplicação é imprescindível conhecer métodos eficientes para percorrer toda a estrutura
  - Exemplo: para listar conteúdo de um arquivo é necessário utilizar algoritmos para percurso.

Árvore binária

Percurso em árvores binárias

- Para percorrer a árvore deve-se, então, visitar cada um de seus nós.
- Visitar um nó significa operar com a informação do nó
  - Por exemplo: imprimir, atualizar informações etc.
- Percorrer uma árvore significa visitar os seus nós exatamente uma vez
  - Contudo, durante um percurso pode-se passar várias vezes por alguns dos nós sem visitá-los.

Árvore binária

Passos básicos do percurso em árvores binárias

- Passo básico: *visitar a raiz v* de cada subárvore T.
- Pode-se assumir que o percurso seja feito como uma combinação de percursos na subárvores de T.
  - Percorrer subárvores esquerda e direita de *v*.
- Estas 3 operações compõem um algoritmo:
  - Visitar a raiz e percorrer as subárvores esquerda e direita
- Questão: definir a ordem em que estas operações são realizadas de acordo com o problema a ser resolvido.

Árvore binária

Percurso em profundidade em árvores binárias

- Algoritmo
  - Seguir tanto quanto possível à esquerda (ou direita)
  - Então mover para trás até a primeira encruzilhada
  - Seguir um passo para direita (ou esquerda)
  - Novamente, seguir tanto quanto possível para a esquerda (ou direita)
  - Repetir o processo até que todos os nós tenham sido visitados
- Envolve 3 tarefas:
  - V - Visitar um nó
  - L – Percorrer subárvore esquerda (*left*)
  - R – Percorrer subárvore direita (*right*)
- 3! Possibilidades: VLR, LVR, LRV, VRL, RVL, RLV.

Árvore binária

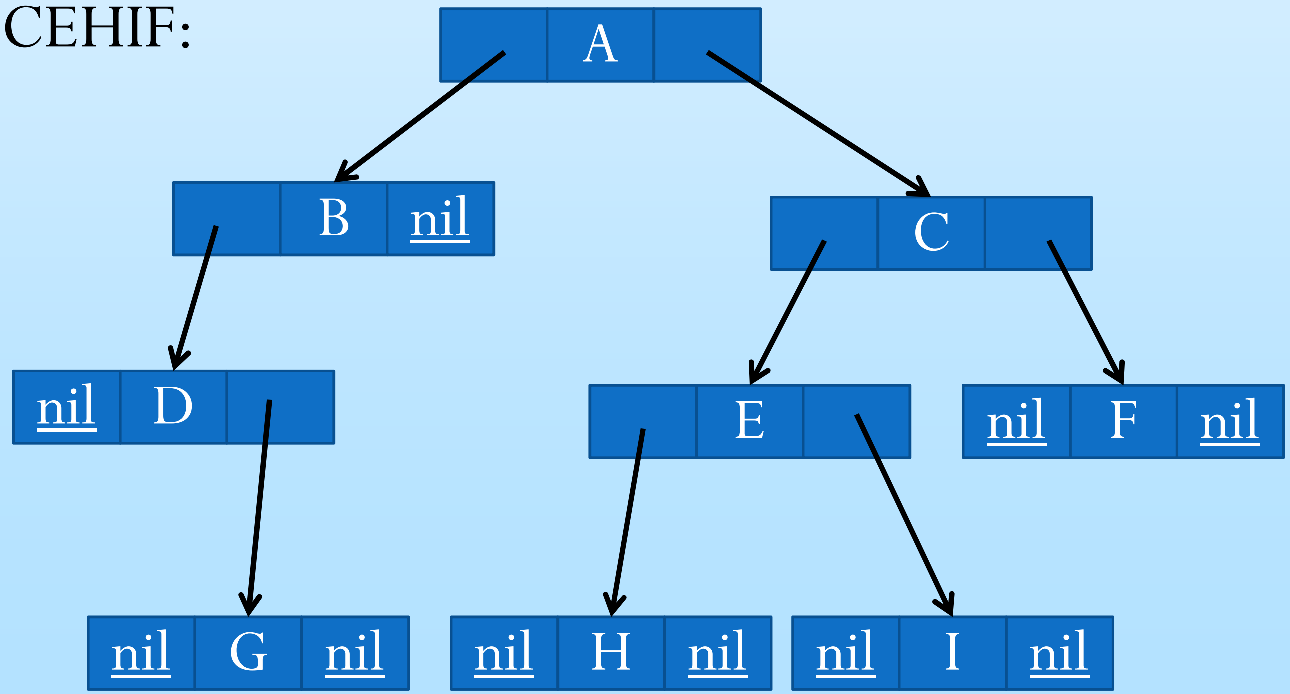
Percurso em profundidade em árvores binárias

- VLR: percurso em pré-ordem (ou pré-fixado)
- LVR: percurso em ordem simétrica (central ou in-ordem)
- LRV: percurso em pós-ordem (ou pós-fixado)

Árvore binária

Percurso em em pré-ordem (pré-fixado)

- Passos:
  - Visitar a raiz;
  - Percorrer sua subárvore esquerda, em pré-ordem (VLR);
  - Percorrer sua subárvore direita, em pré-ordem (VLR).
- Exemplo: ABDGCEHIF:



Árvore binária

Percurso em em pré-ordem (pré-fixado)

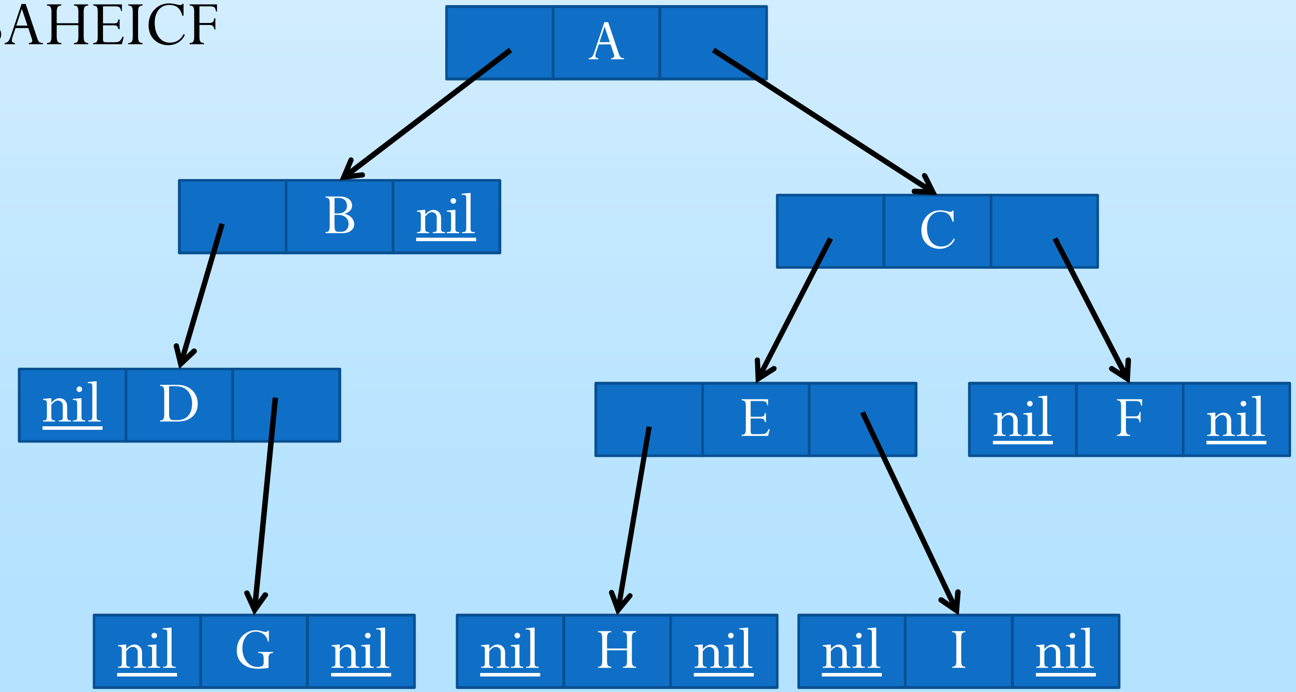
- Algoritmo: parâmetro RAIZ, apontador para a raiz;
- Procedimento recursivo:

```
.............................................................................
tipo R = ref NO;
tipo NO = reg (R: ESQ, tipo-t: X, R: DIR);
.............................................................................


proc PREFIXADO  (R: RAIZ)
   se  RAIZ ≠ nil  então
        VISITA(RAIZ↑);
        PREFIXADO(RAIZ↑.ESQ);
        PREFIXADO(RAIZ↑.DIR);
   fim-se;
fim-proc; {PREFIXADO}
```

Árvore binária

Percurso em em in-ordem ou central

- Passos:
  - Percorrer sua subárvore esquerda, em in-ordem;
  - Visitar a raiz;
  - Percorrer sua subárvore direita, em in-ordem.
- Exemplo: DGBAHEICF



Árvore binária

Percurso em em in-ordem ou central

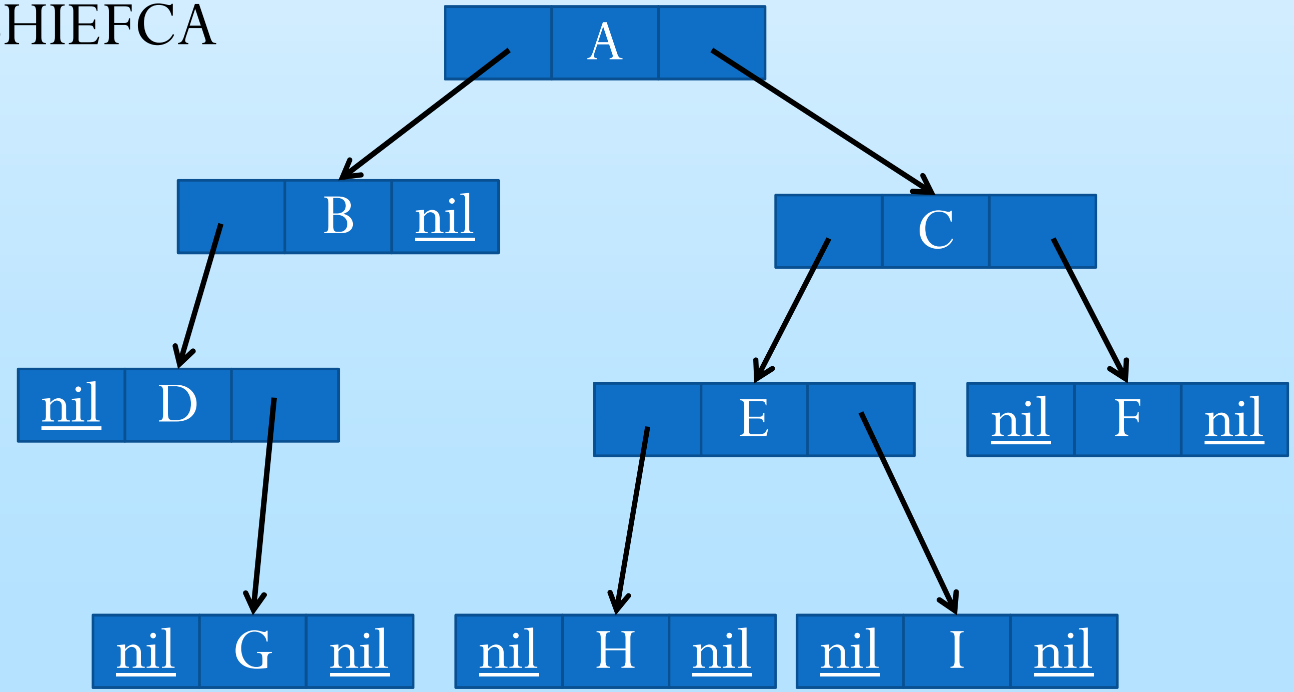
- Algoritmo: parâmetro `RAIZ`, apontador para a raiz;
- Procedimento recursivo:

```
.......................................................................
tipo R = ref NO;
tipo NO = reg (R: ESQ, tipo-t: X, R: DIR);
.......................................................................


proc CENTRAL (R: RAIZ)
   se  RAIZ ≠ nil  então
      CENTRAL(RAIZ↑.ESQ);
      VISITA(RAIZ↑);
      CENTRAL(RAIZ↑.DIR);
   fim-se;
fim-proc; {CENTRAL}
```

Árvore binária

Percurso em pós-ordem

- Passos:
  - Percorrer sua subárvore esquerda, em pós-ordem;
  - Percorrer sua subárvore direita, em pós-ordem;
  - Visitar a raiz.
- Exemplo: GDBHIEFCA



Árvore binária

Percurso em pós-ordem

- Algoritmo: parâmetro `RAIZ`, apontador para a raiz;
- Procedimento recursivo:

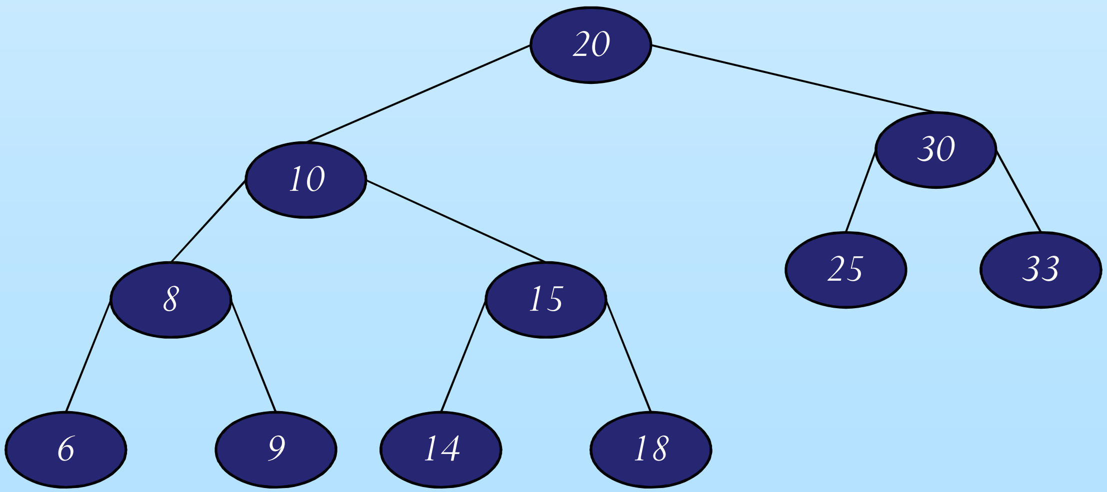
```
.............................................................................
tipo R = ref NO;
tipo NO = reg (R: ESQ, tipo-t: X, R: DIR);
.............................................................................


proc POSFIXADO(R: RAIZ)
   se  RAIZ ≠ nil então
       POSFIXADO(RAIZ↑.ESQ);
       POSFIXADO(RAIZ↑.DIR);
       VISITA(RAIZ↑);
   fim-se;
fim-proc; {POSFIXADO}
```

Árvore binária

Percurso em profundidade

Exercício 1 – percorrer em profundidade

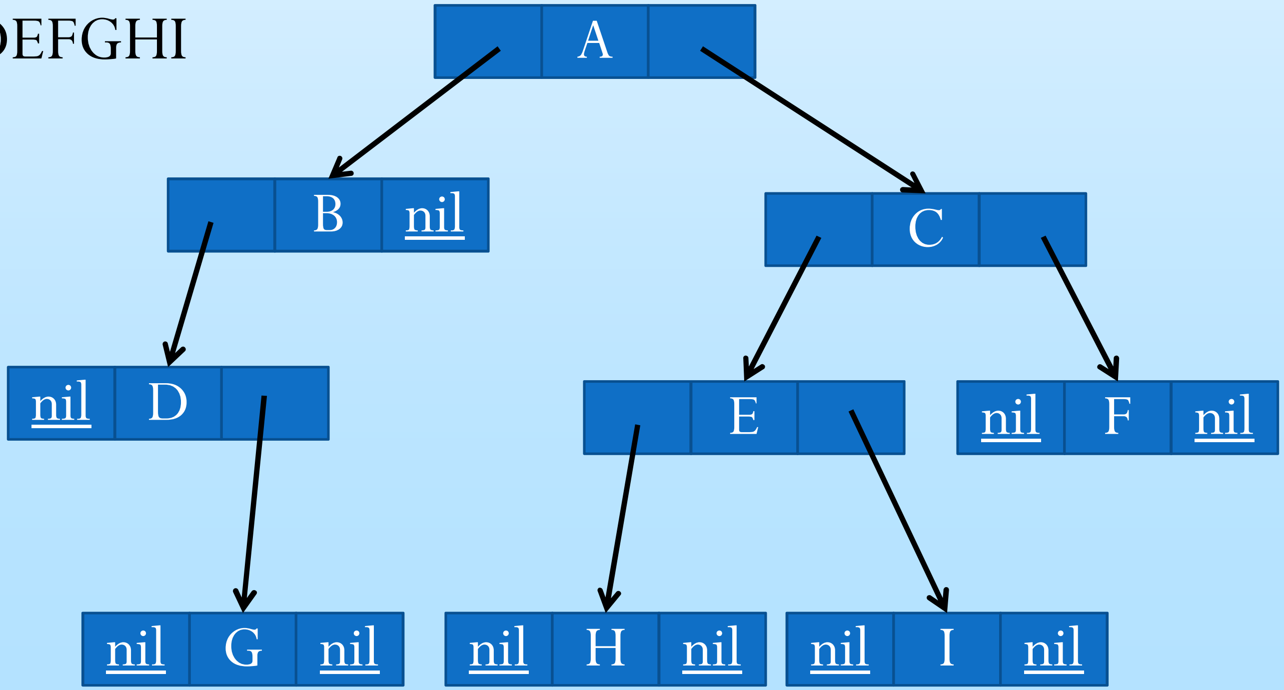
- Para a árvore binária a seguir:
  - Percorrer em pré-ordem (pré-fixado)
  - Percorrer em pós-ordem (pós-fixado)
  - Percorrer em in-ordem (central)



Árvore binária

Percurso em extensão (largura)

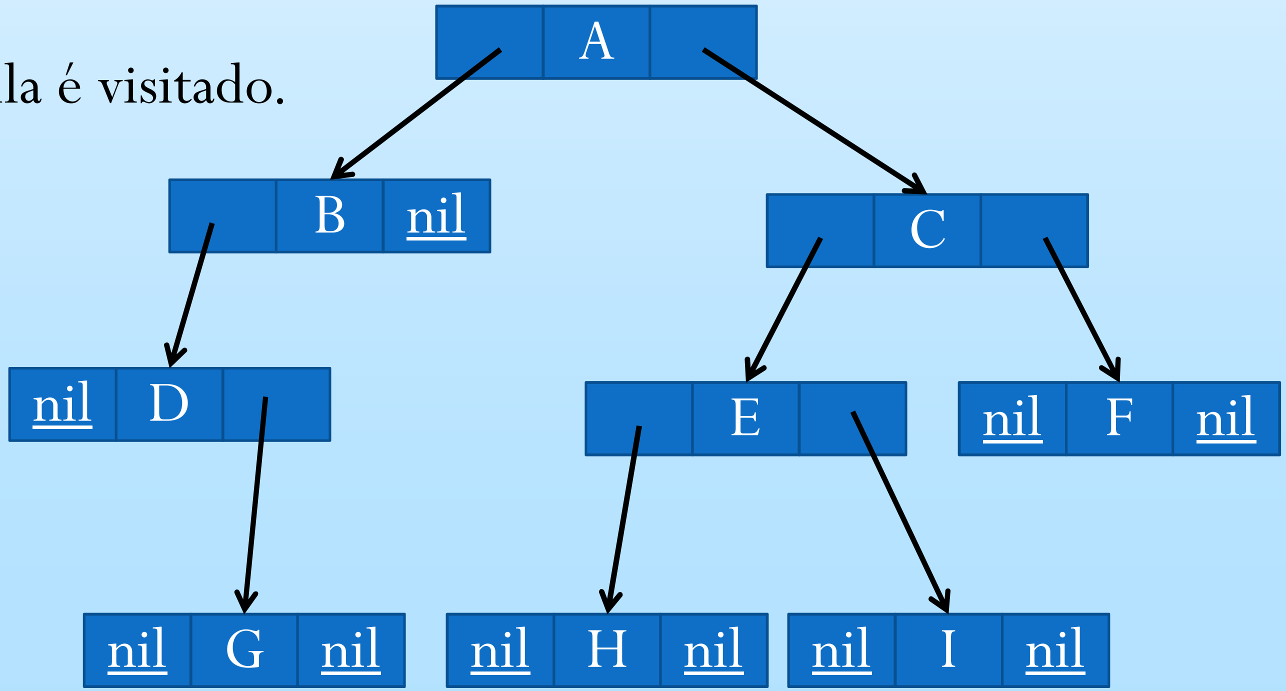
- Corresponde a visitar cada nó começando o nível mais baixo (ou mais alto) e movendo para baixo (ou para cima) nível a nível, visitando nós em cada nível da esquerda para direita (ou da direita para a esquerda).
- Exemplo: ABCDEFGHI



Árvore binária

Percurso em extensão (largura)

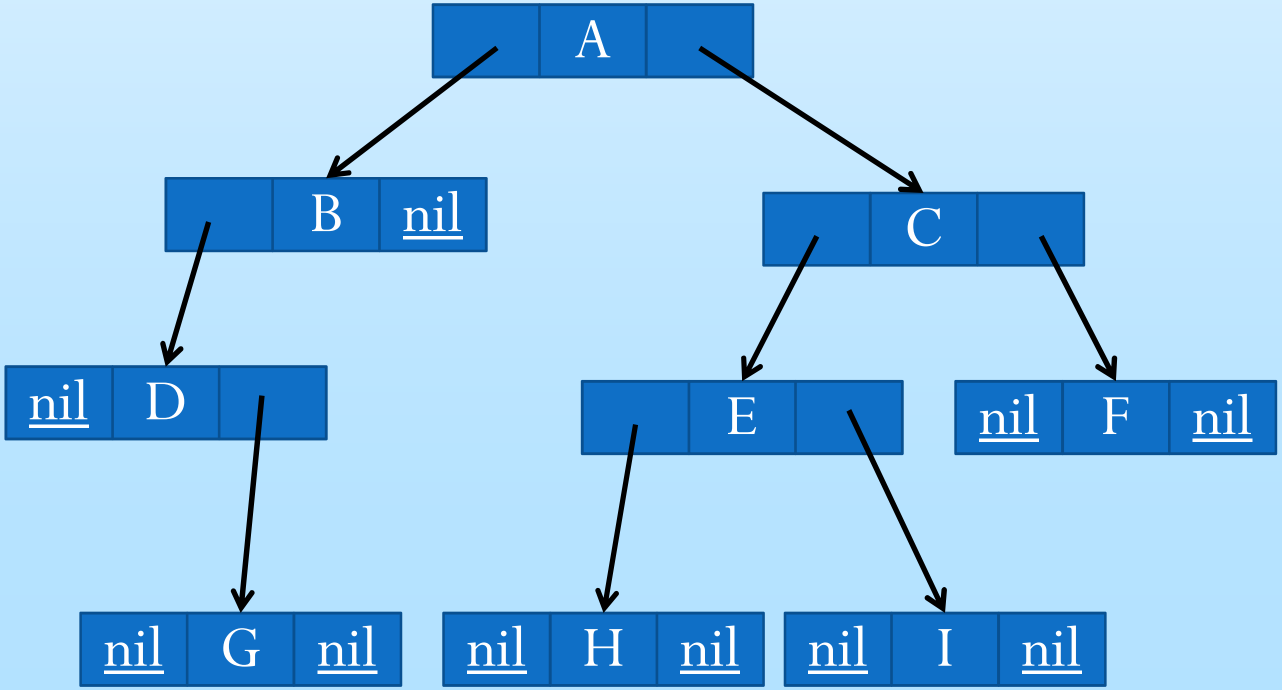
- Implementação utilizando fila.
- Por exemplo, para um percurso em extensão de cima para baixo, da esquerda para a direita usando fila.
  - Quando um nó é visitado, seus filhos (se houver) são colocados no final da fila
  - Nó no início da fila é visitado.



Árvore binária

Percurso em extensão (largura)

A	Visita A → insere B e C	B C
B C	Visita B → insere D	C D
C D	Visita C → insere E e F	D E F



Árvore binária

Percurso em extensão (largura)

- Para um nó no nível *n*, seus filhos estão no nível $n + 1$
- Após a visita do nó *n*, seus filhos são colocados no final da fila
  - Eles serão visitados depois que todos os nós do nível *n* forem visitados
- Assim a restrição de que todos os nós no nível *n* precisam ser visitados antes de visitar quaisquer nós no nível $n + 1$ será satisfeita.

Árvore binária

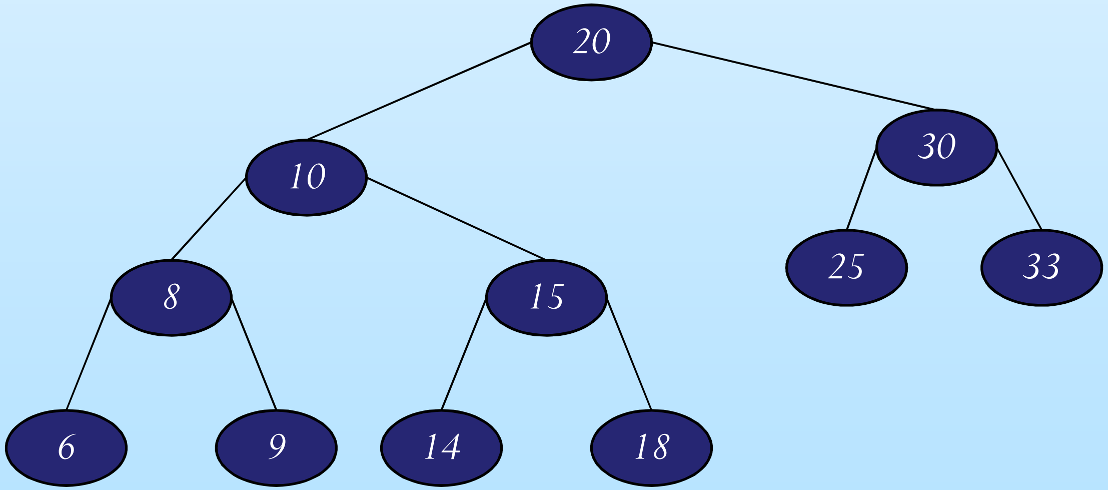
Percurso em extensão (largura)

- Como implementar:
  - Utilizar uma fila.
- Algoritmo:
  - Se árvore não está vazia
    - Coloca nó raiz na fila
    - Enquanto tem nó na fila
      - Tira nó da fila e coloca na lista resultado
      - Se tem nó à esquerda
        - Coloca na fila o nó à esquerda
      - Se tem nó à direita
        - Coloca na fila o nó à direita

Árvore binária

Percurso em extensão (largura)

- Exercício 2 – Percorrer em extensão a árvore binária a seguir:
  - Apresentar o caminho percorrido



Árvore binária

Exercícios

1) Fazer o algoritmo descrito acima para realizar o percurso em extensão.
2) Escreva um algoritmo para calcular a altura de um árvore binária.
3) Escreva um algoritmo que conte o número de nós de uma árvore binária.
4) Escreva um algoritmo que conte o número de folhas de uma árvore binária.
5) Escreva um algoritmo para excluir todas as folhas de uma árvore binária, deixando a raiz e os nós intermediários no respectivo lugar. Dica: use o percurso pré-ordem.

Árvore binária

Exercícios

6) Escreva um algoritmo que determine se uma árvore binária é cheia ou não.

7) Reescreva o algoritmo de percurso em pré-ordem usando uma pilha em vez da recursão.

8) Reescreva o algoritmo de percurso em in-ordem usando uma pilha em vez da recursão.

9) Escreva um algoritmo que cria uma imagem espelho de uma árvore binária, isto é, todos os filhos à esquerda tornam-se filhos à direita, e vice-versa.

10) Ache a raiz de cada uma das seguintes árvores binárias:

Árvore com percurso pós-ordem: FCBDG

Árvore com percurso pré-ordem: IBCDFEN

Árvore com percurso in-ordem: CBIDFGE

Árvore binária

Exercícios

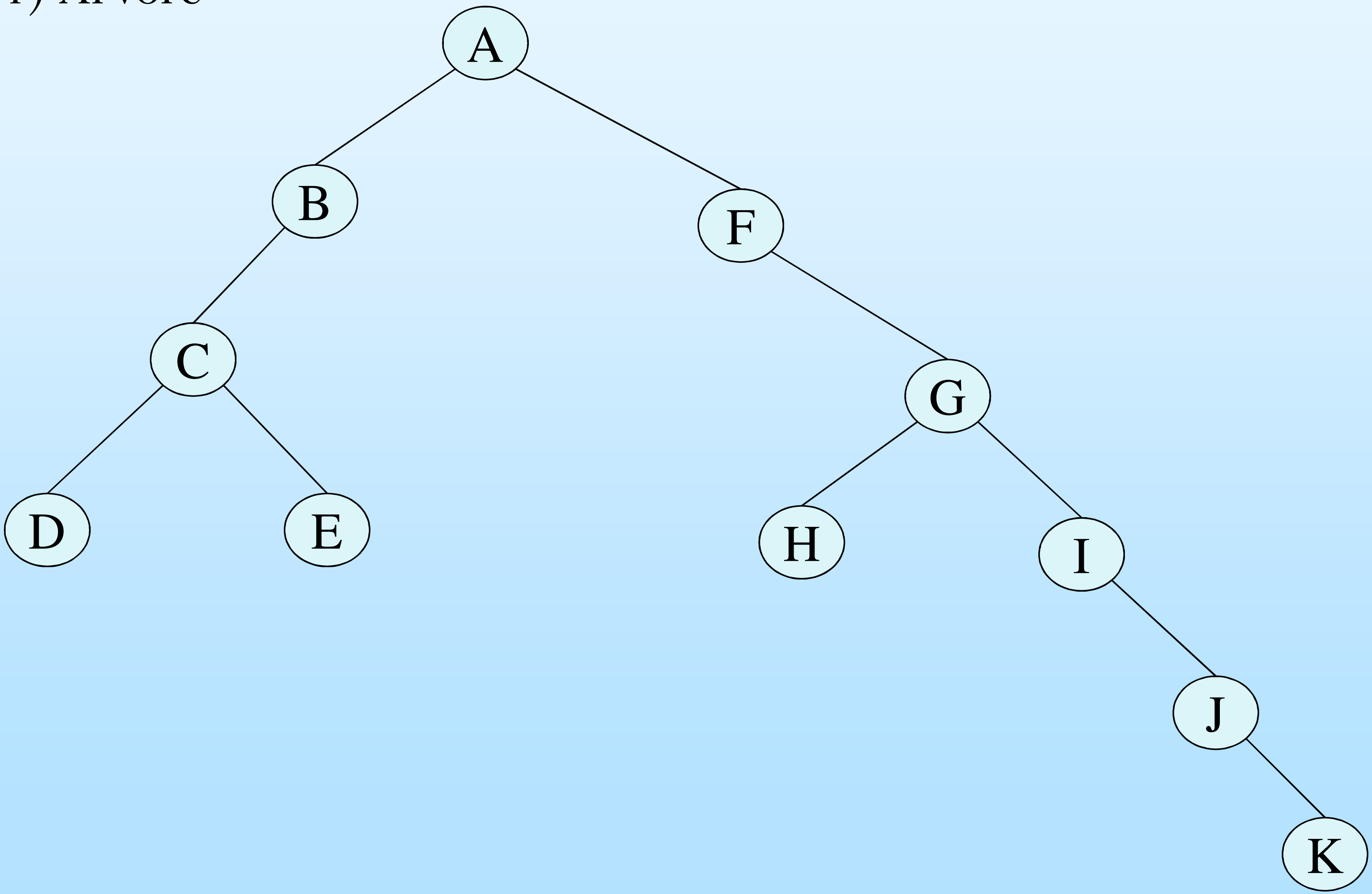
11) Para a árvore binária do próximo slide, faça as seguintes atividades:

- Ache o grau para cada um dos nós.
- Determine os nós folhas.
- Complete a árvore binária com tantos nós quanto forem necessários, para transformá-la em árvore binária cheia.

Árvore binária

Exercícios

11) Árvore



Árvore binária

Exercícios

12) Mostre o resultado do seguinte procedimento usando a árvore do exercício 11.

```
tipo R = ref NO;
tipo NO = reg ( R :  ESQ,  tipot :  X,  R :  DIR );
.....................................................
procedimento PERCURSO(R: RAIZ)
  se  RAIZ = nil  então
     imprima(“Nulo”);
  senao
     PERCURSO(RAIZ↑.DIR);
     imprima(“Lado direito feito”);
     PERCURSO(RAIZ↑.ESQ);
     imprima(“Lado esquerdo feito”);
  fim-se;
fim-procedimento; {PERCURSO}
```

Árvore binária

Exercícios

13) Considerando a árvore do exercício 11, mostre o resultado dos três tipos de passeio em profundidade (ordem, pré-ordem e pós-ordem).

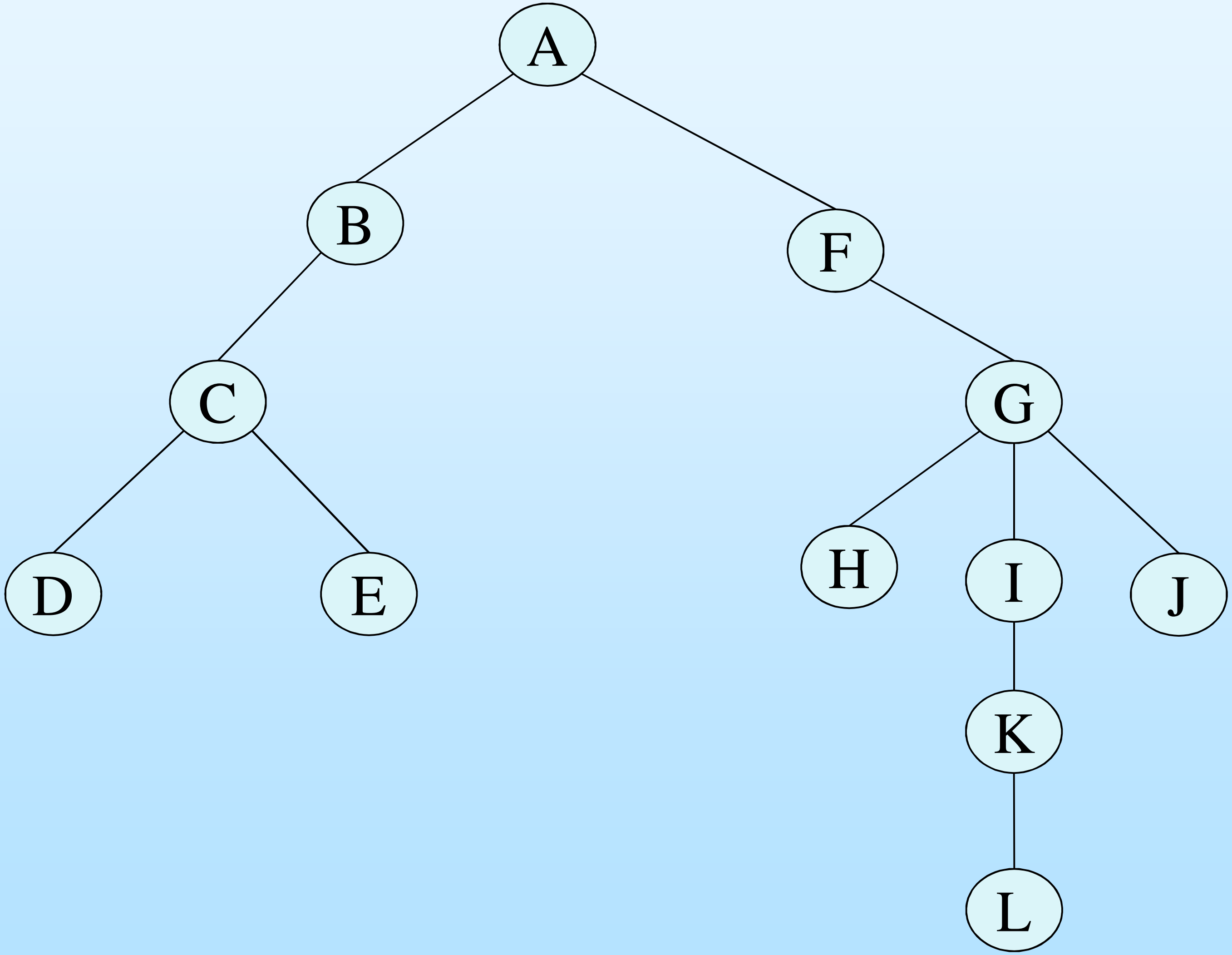
14) Ainda usando a árvore do exercício 11, mostre o resultado do passeiro em largura (em nível).

15) Desenhe uma árvore cheia de nível 4.

16) A seguir encontra-se uma árvore

Árvore binária

Exercícios



Árvore binária

Exercícios

a) Ache os nós filhos de H.
b) Ache o nó pai de K.
c) Ache o nó filho de C.

17) Em relação à árvore do exercício 16, determine:

a) A altura da árvore.
b) Altura do nó G.
c) Nível do nó G.
d) Nível do nó A.
e) Altura do nó E.

18) Em relação à árvore do exercício 16, mostre as subárvores do nó F.

Árvore binária

Exercícios

19) Qual a altura máxima e mínima de uma árvore com 28 nós?

20) Em uma árvore binária, qual é o número máximo de nós que pode ser achado nos níveis 3, 4 e 12?

21) Qual é o menor número de níveis que uma árvore binária com 42 nós pode apresentar?

22) Qual é o número máximo de nós no nível 5 de uma árvore binária.

23) Fazer um procedimento para percorrer uma árvore que possui uma representação encadeada e imprimi-la em numa representação por parênteses aninhados.

Árvore binária

Exercícios resolvidos

1. Fazer o algoritmo descrito acima para realizar o percurso em extensão.

   O algoritmo é o seguinte:

- Se árvore não está vazia
  - Coloca nó raiz na fila
  - Enquanto tem nó na fila
    - Tira nó da fila e coloca na lista resultado
    - Se tem nó à esquerda
      - Coloca na fila o nó à esquerda
    - Se tem nó à direita
      - Coloca na fila o nó à direita

Árvore binária

Exercícios resolvidos

1. Uma solução iterativa poderia ser:

```
tipo RNA = ref NOA;
tipo NOA = reg (RNA: ESQ, tipo-t: X, RNA: DIR);
tipo RDF = ref DESC;
tipo DESC = reg (RNF: C, F);
tipo RNF = ref NOF;
tipo NOF = reg (RNA: Y, RNF: PROX);
```

Observações:

1. O procedimento PERCIMP, a seguir, percorre a árvore em extensão, imprimindo a informação (X) de seus nós.
2. A informação (Y) de um nó da fila é um ponteiro para um nó da árvore.

Árvore binária

Exercícios resolvidos solução iterativa (continuação)

```
procedimento PERCIMP (RNA: RAIZ){percorre e imprime}
    RNF: FL;
    RNA: P;
    {cria fila}
    aloque (FL);
    FL↑.C ← nil;
    FL↑.F ← nil;
   se RAIZ ≠ nil então
     ENTRAF (RAIZ, FL);
      enquanto FL↑.C ≠ nil faça
        P ← SAIF (FL);
        imprima (P↑.X);
        se P↑.ESQ ≠ nil então
           ENTRAF (P↑.ESQ, FL);
        fim-se;
        se P↑.DIR ≠ nil então
           ENTRAF (P↑.DIR, FL);
        fim-se;
      fim-enquanto;
   fim-se;
fim-procedimento; {PERCIMP}
```

Árvore binária

Exercícios resolvidos

solução iterativa (continuação)

```
tipo RNA = ref NOA;
tipo NOA = reg (RNA: ESQ, tipo-t: X, RNA: DIR);
tipo RDF = ref DESC;
tipo DESC = reg (RNF: C, F);
tipo RNF = ref NOF;
tipo NOF = reg (RNA: Y, RNF: PROX);

procedimento ENTRAF (RNA:PA, RNF:FL);{insere nó em FL}
   RNF: PF;
   {cria nó da fila}
   aloque (PF);
   PF↑.Y ← PA;
   PF↑.PROX ← FL↑.C;
   {atualiza descritor}
   FL↑.C ← PF;
   se FL↑.F ≠ nil então
      FL↑.F ← PF;
   fim-se;
fim-procedimento; {ENTRAF}
```

Árvore binária

Exercícios resolvidos solução iterativa (continuação)

```
tipo RNA = ref NOA;
tipo NOA = reg (RNA: ESQ, tipo-t: X, RNA: DIR);
tipo RDF = ref DESC;
tipo DESC = reg (RNF: C, F);
tipo RNF = ref NOF;
tipo NOF = reg (RNA: Y, RNF: PROX);

função SAIF (RNF: FL): RNA;{exclui nó de FL e retorna Y}
   RNF: PF;
   RNA: PA;
   PF ← FL↑.C;
   FL↑.C ← PF↑.PROX;
   se FL↑.C ≠ nil então    {fila esvaziou}
      FL↑.F ← nil;
   fim-se;
   PA ← PF↑.Y;
   desaloque(PF);
   SAIF ← PA;    {ou retorne (PA);}
fim-função;    {SAIF}
```

Árvore binária

Exercícios resolvidos

2. Uma solução recursiva:

```
tipo RNA = ref NOA;
tipo NOA = reg (RNA: ESQ, tipo-t: X, RNA: DIR);
tipo RDF = ref DESC;
tipo DESC = reg (RNF: C, F);
tipo RNF = ref NOF;
tipo NOF = reg (RNA: Y, RNF: PROX);
```

Observações:

1. `PERCIMPR`, a seguir, percorre a árvore em extensão, imprimindo a informação (`X`) de seus nós.
2. A informação (`Y`) de um nó da fila é um ponteiro para um nó da árvore.

Árvore binária

Exercícios resolvidos

solução recursiva (continuação...)

```
procedimento PERCIMPR (RNA: RAIZ);{percorre e imprime}
   RNF: FL;
   {cria fila}
   aloque (FL);
   FL↑.C ← nil;
   FL↑.F ← nil;
   se RAIZ ≠ nil então
      ENTRAF (RAIZ, FL);
      PERCORRE (FL);
   fim-se;
fim-procedimento;    {PERCIMPR}
```

Árvore binária

Exercícios resolvidos solução recursiva (continuação...)

```
procedimento PERCORRE (RNF: FL);    {percorre e imprime}
   RNA: P;
   se FL↑.C ≠ nil então
      P ← SAIF (FL);
      imprima (P ↑.X);
      se P↑.ESQ ≠ nil então
         ENTRAF (P↑.ESQ, FL);
      fim-se;
      se P↑.DIR ≠ nil então
         ENTRAF (P↑.DIR, FL);
      fim-se;
      PERCORRE (FL);    {chamada recursiva}
   fim-se;
fim-procedimento;    {PERCORRE}
```

Observação: a árvore não é parâmetro do procedimento PERCORRE, mas os seus nós são acessados. Isso está correto?