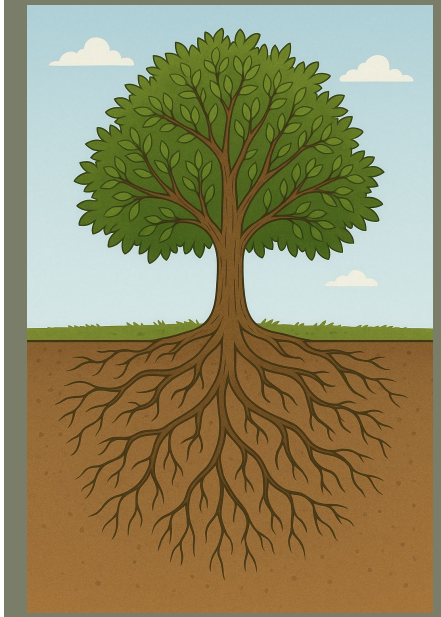


Árvores

Prof. Dr. Marcelo Fernando Rauber
marcelo.rauber@ifc.edu.br

Introdução - Árvores

- Listas, pilhas e filas são adequadas para estruturas lineares.
- Imaginemos, estruturas como
  - Pastas e sub-pastas do sistemas de arquivos
  - Armazenar uma árvore genealógica
- Nesses casos, estruturas lineares **não** são adequadas
- Para isso temos as estruturas de dados de **árvores**!!!

Introdução - Árvores

Formas de representação:

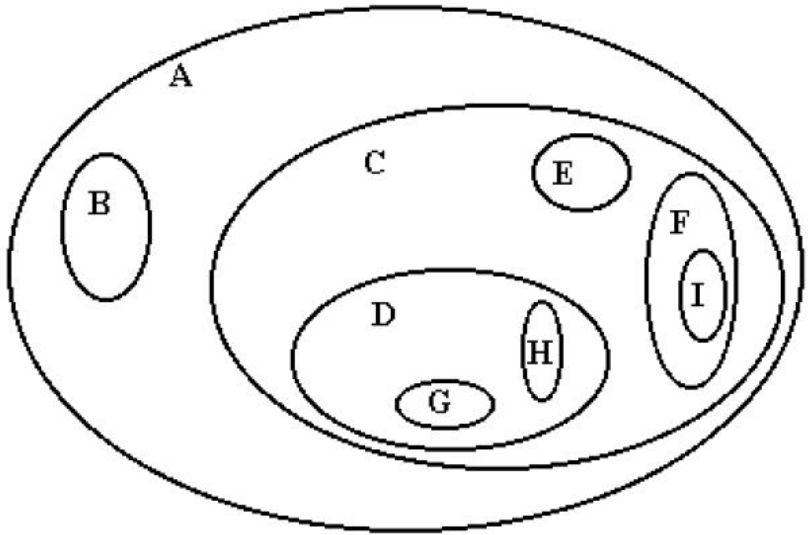
- por parênteses aninhados

(A (B) (C (D (G) (H)) (E) (F (I))))

Introdução - Árvores

Formas de representação:

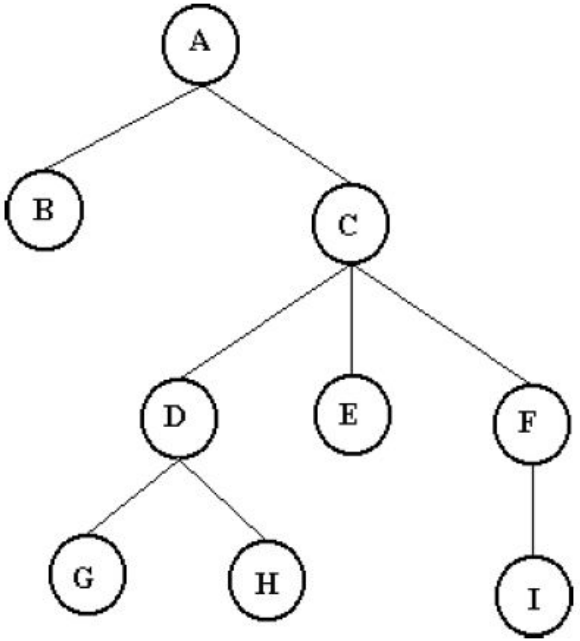
- Diagrama de Inclusão



Introdução - Árvores

Formas de representação:

- Representação Hierárquica
  - Essa é a mais utilizada!



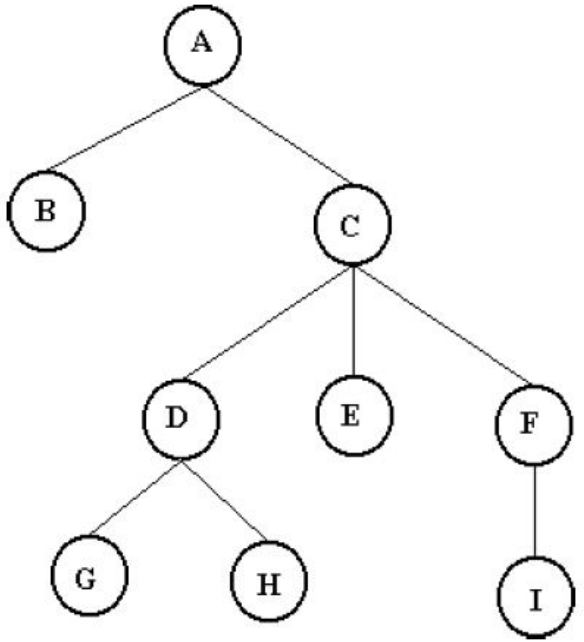
Introdução - Árvores

Definição:

- São estruturas de dados hierárquicas, portanto, não lineares;
- É composta por um número finito de elementos;
- Uma árvore é um único nó ou um nó raiz conectado a um conjunto de árvores;

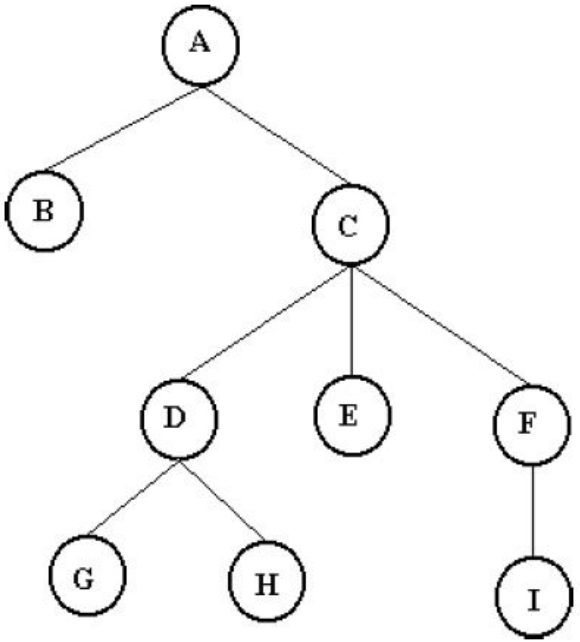
Árvores: Conceitos Fundamentais

- **Nó, Nodo ou vértice:** é um objeto simples que pode ter um nome e mais alguma outra informação associada.
- É cada uma das “bolinhas” na representação ao lado.



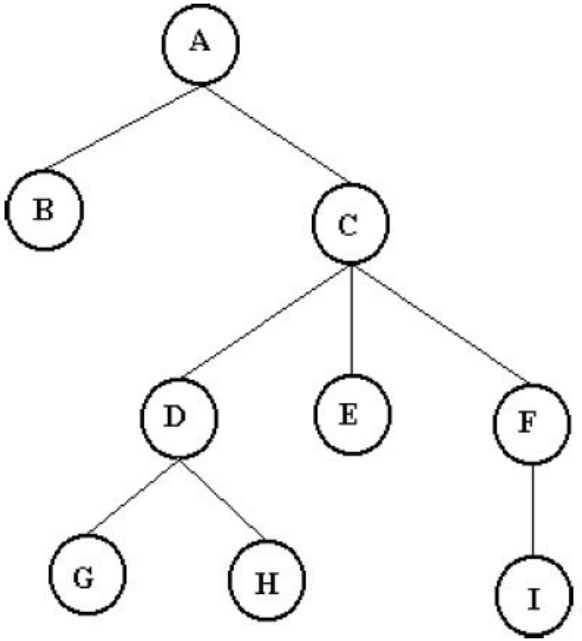
Árvores: Conceitos Fundamentais

- **Arco ou aresta:** é a conexão entre dois nós, que pode ser direcional ou não.
- São as linhas que unem as “bolinhas” na representação ao lado.



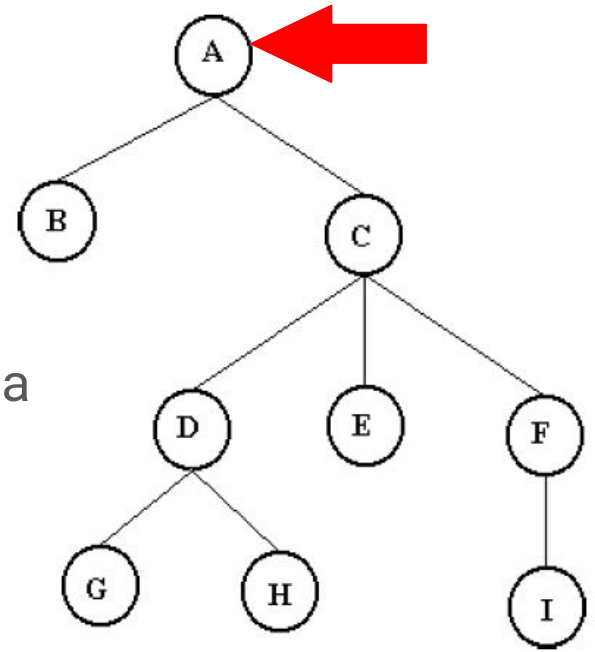
Árvores: Conceitos Fundamentais

- **Arco, aresta ou ligação:** é a conexão entre dois nós, que pode ser direcional ou não.
- São as linhas que unem as “bolinhas” na representação ao lado.



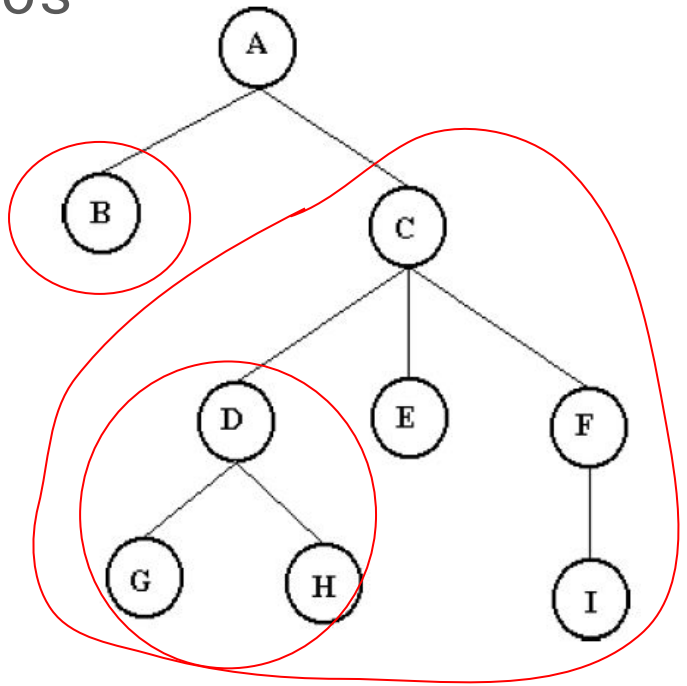
Árvores: Conceitos Fundamentais

- **Nó Raiz:** É o nó especial que fica no topo da hierarquia (indicado pela seta vermelha).
- É o único nó que não tem um "pai".
- Toda árvore não vazia tem exatamente uma raiz.
- Existe exatamente um caminho entre a raiz e cada um dos nós da árvore
  - se existir mais de um caminho ou nenhum temos um **grafo**, e não uma árvore



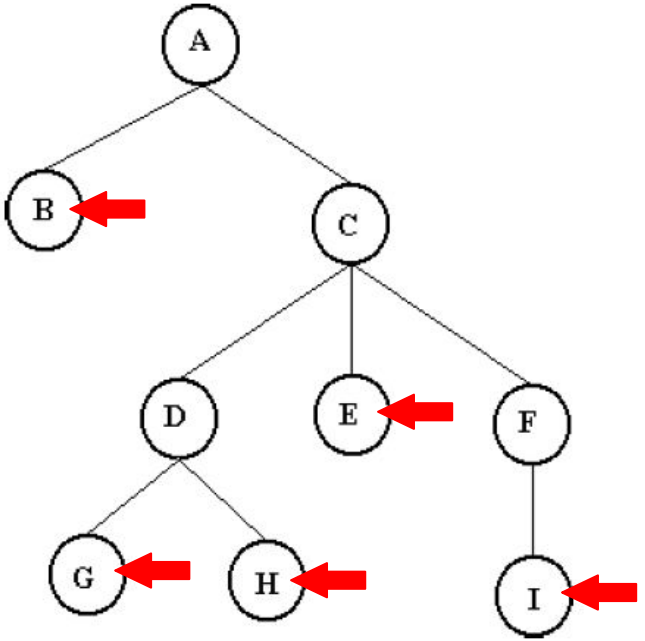
Árvores: Conceitos Fundamentais

- **Subárvore:** Uma árvore formada por um nó e todos os seus descendentes.
- Exemplos nos círculos vermelhos.



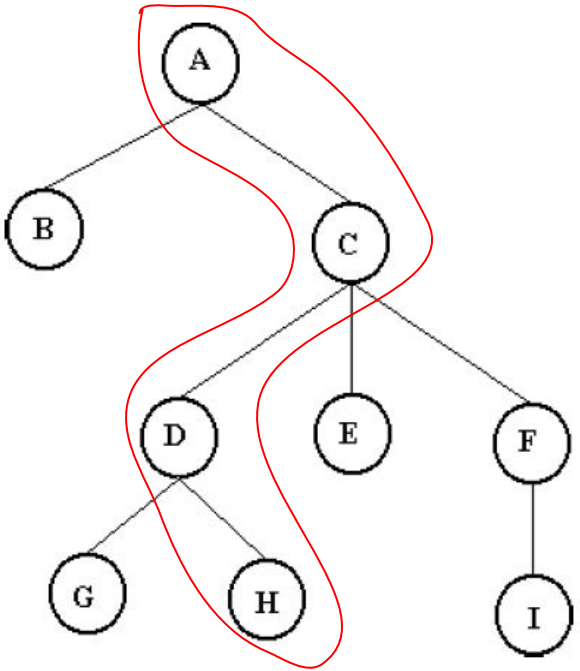
Árvores: Conceitos Fundamentais

- **Folha, nó terminal ou nó externo:** um nó que não possui filhos. São os nós nas extremidades inferiores da árvore (indicados pelas setas vermelhas).



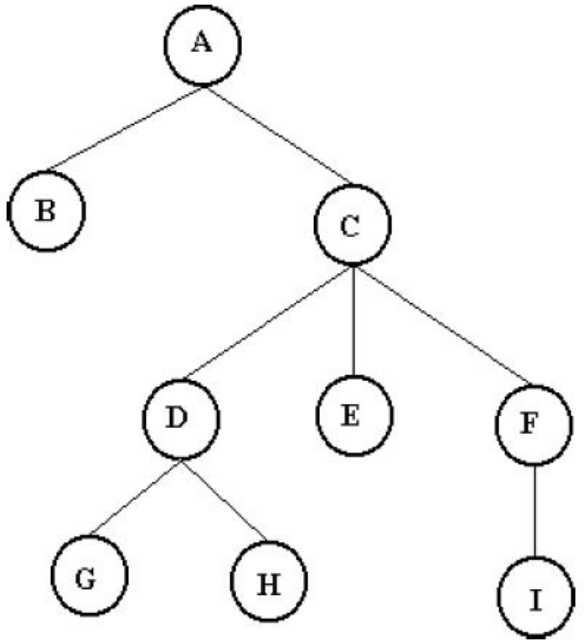
Árvores: Conceitos Fundamentais

- **Caminho:** é é uma lista de vértices distintos e sucessivos,conectados por arcos (arestas) da árvore.
- Exemplo ao lado, marcado o caminho para chegar em “H”.
- Existe exatamente um caminho entre a raiz e cada um dos nós da árvore
  - se existir mais de um caminho ou nenhum temos um grafo, e não uma árvore.



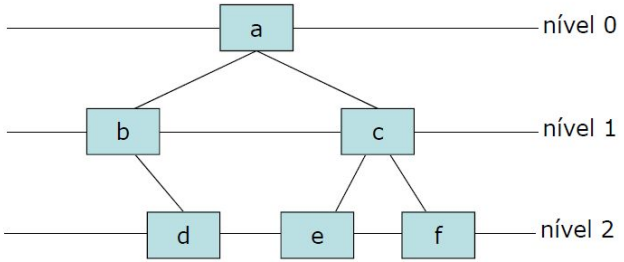
Árvores: Conceitos Fundamentais

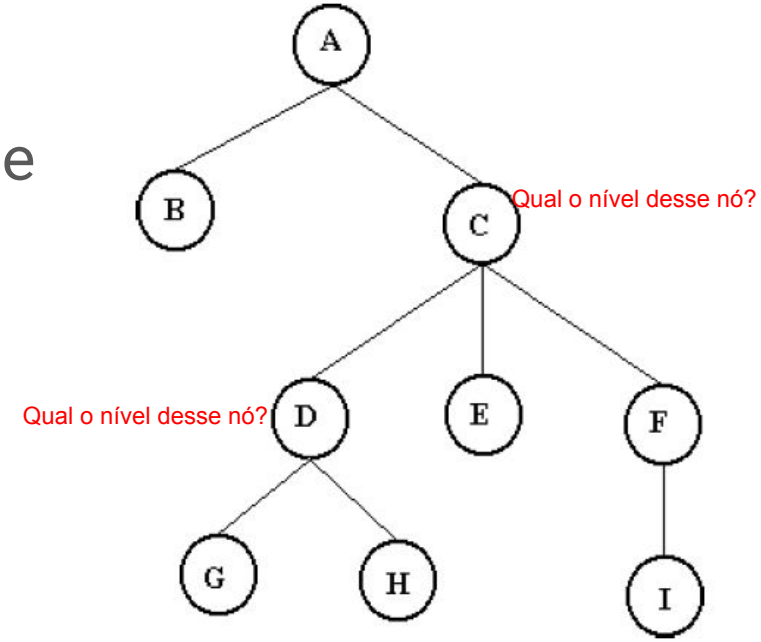
- **Nós filhos, pais, tios, irmãos e avô:** idênticos a nossa genealogia.
- Exemplos:
  - B e C são irmãos
  - A é pai de B
  - D, E, F são filhos de C
  - A é avô de D



Árvores: Conceitos Fundamentais

- **Nível ou Profundidade de um Nó:** O comprimento do caminho (número de arestas) da raiz até esse nó. A raiz está no nível 0.

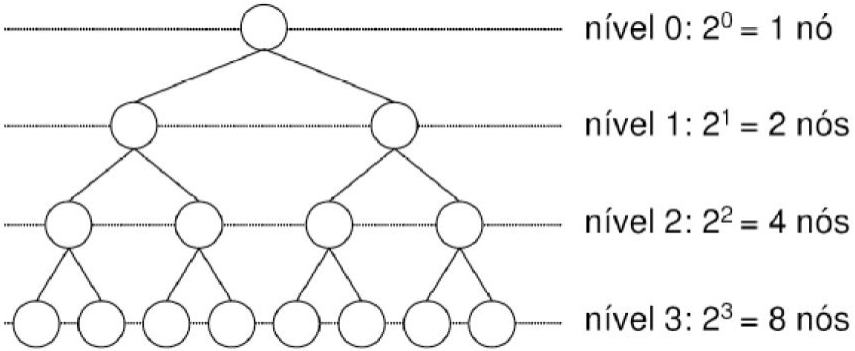


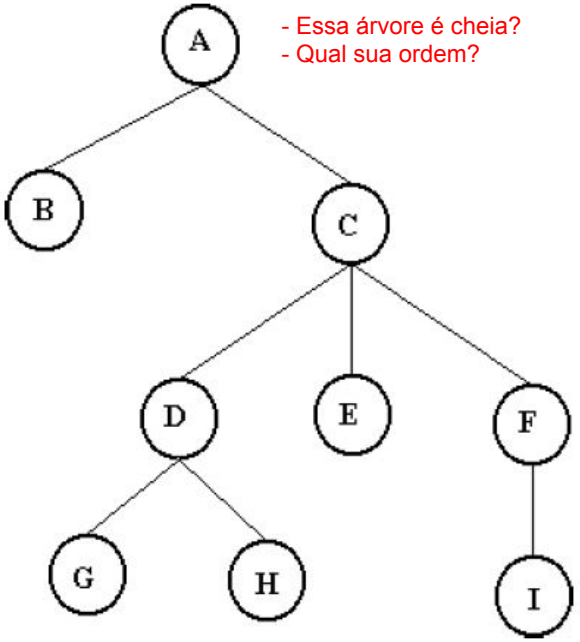


Árvores: Conceitos Fundamentais

- **Árvore Cheia:** Todos os nós tem o número máximo de subárvores.
- **Ordem**: O número máximo de filhos que qualquer nó na árvore pode ter.

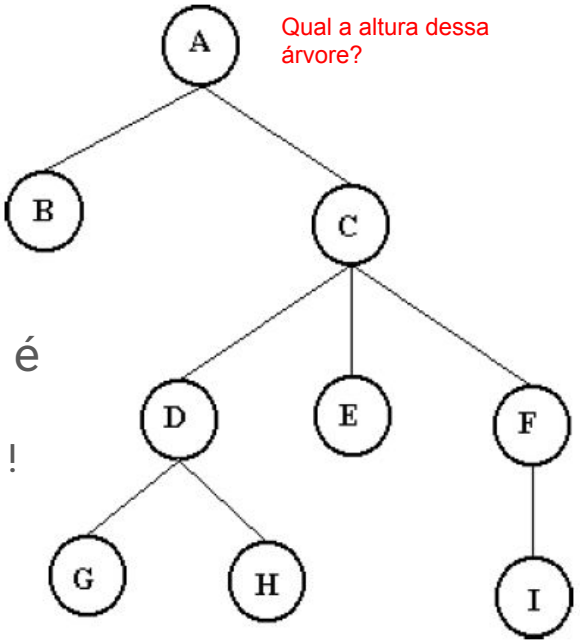
- Exemplo: árvore binária cheia





Árvores: Conceitos Fundamentais

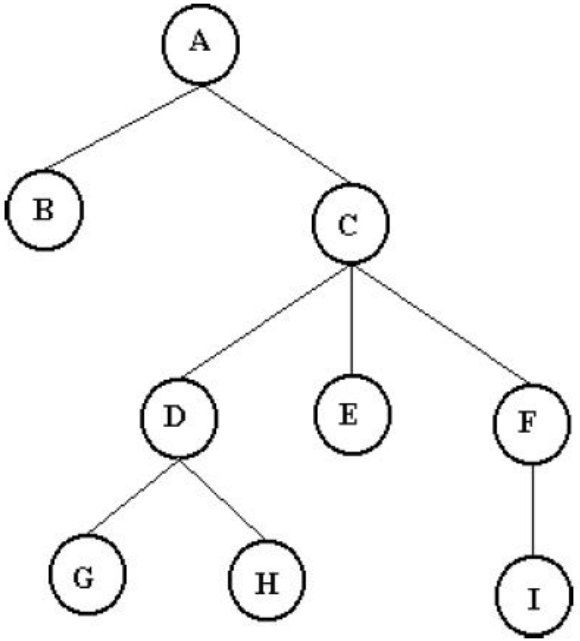
- **Altura:** O comprimento do caminho mais longo da raiz até a folha mais distante.
  - a altura de uma árvore com um único nó (raiz) é zero
  - a altura de uma árvore vazia é -1
- É o nível máximo de qualquer nó na árvore.
- O esforço computacional necessário para alcançar qualquer nó da árvore é proporcional à altura da árvore.
  - Queremos árvores o mais cheias possível!
- **Degenerada**: quando os nós internos têm uma única subárvore associada, virando uma estrutura linear.



Aplicações das Árvores: onde elas crescem na Computação?

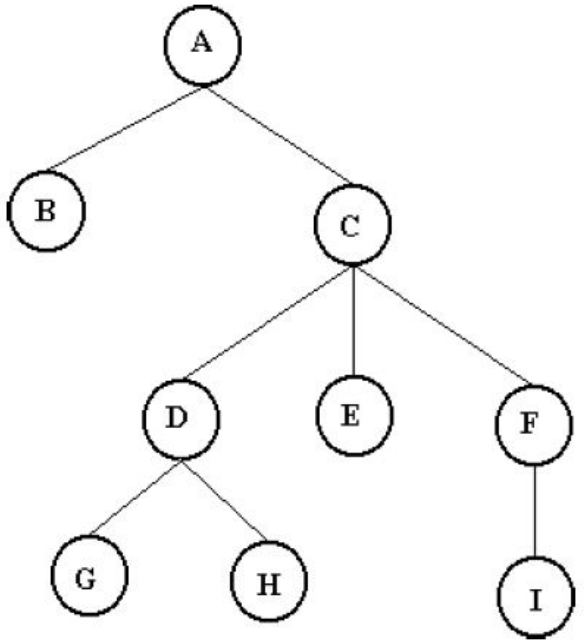
Aplicações das Árvores

- **Sistemas de Arquivos:** A forma como organizamos pastas e arquivos em nossos computadores é uma árvore!
- A pasta raiz (como `C:\` no Windows ou `/` no Linux/macOS) é a raiz da árvore, as pastas são nós internos e os arquivos são geralmente as folhas.



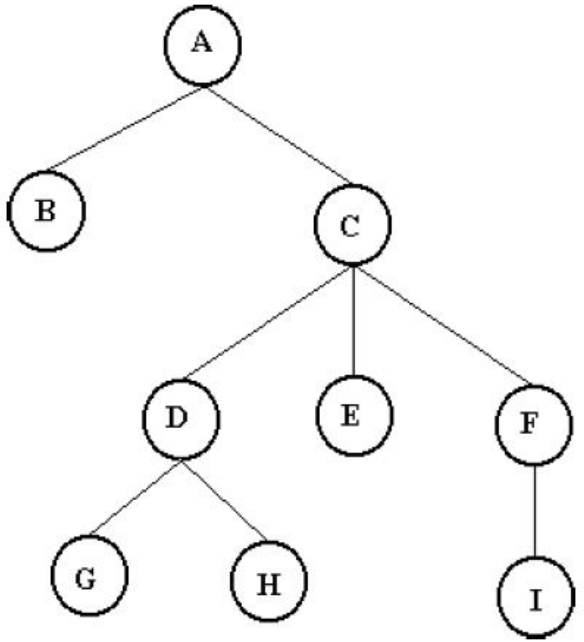
Aplicações das Árvores

- **Estrutura de Sites e Documentos (DOM):** Páginas da web (HTML) e documentos XML são estruturados como árvores.
- A tag <html> é a raiz, e outras tags como <head> , <body> , , <p> são nós que formam a hierarquia do conteúdo. O navegador usa essa estrutura (chamada DOM - Document Object Model) para renderizar a página.



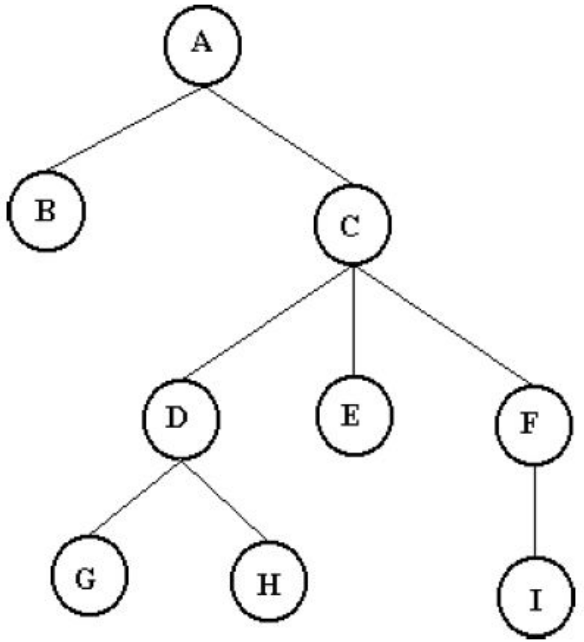
Aplicações das Árvores

- **Estrutura de Sites e Documentos (DOM):** Páginas da web (HTML) e documentos XML são estruturados como árvores.
- A tag <html> é a raiz, e outras tags como <head> , <body> , , <p> são nós que formam a hierarquia do conteúdo. O navegador usa essa estrutura (chamada DOM - Document Object Model) para renderizar a página.



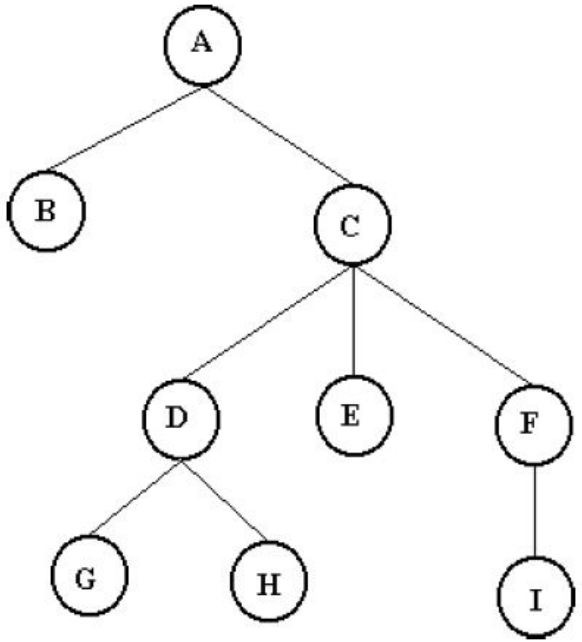
Aplicações das Árvores

- **Organogramas e Hierarquias:** A estrutura de uma empresa, com presidente, diretores, gerentes e funcionários, pode ser representada perfeitamente por uma árvore.



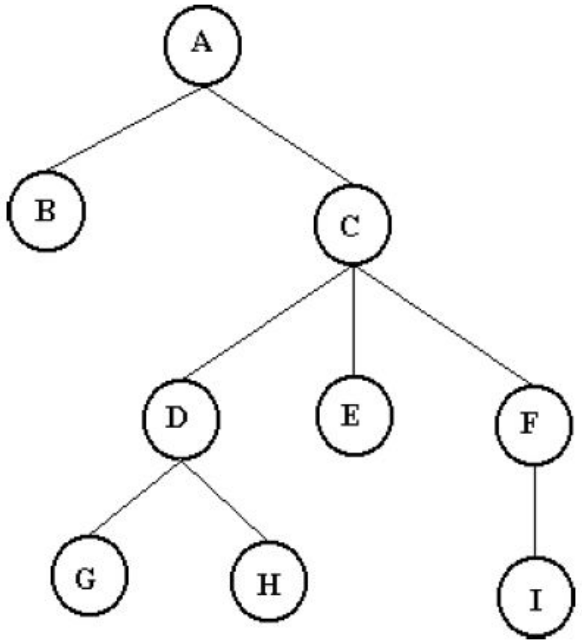
Aplicações das Árvores

- **Árvores Genealógicas:** Representam as relações familiares ao longo das gerações.



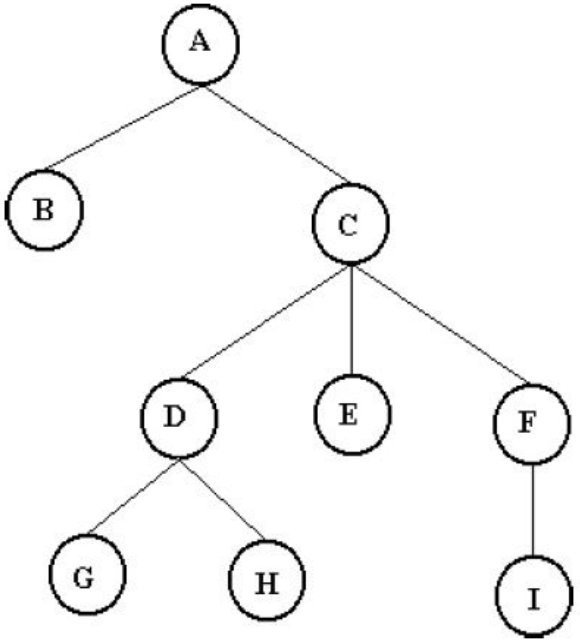
Aplicações das Árvores

- **Árvores Genealógicas:** Representam as relações familiares ao longo das gerações.



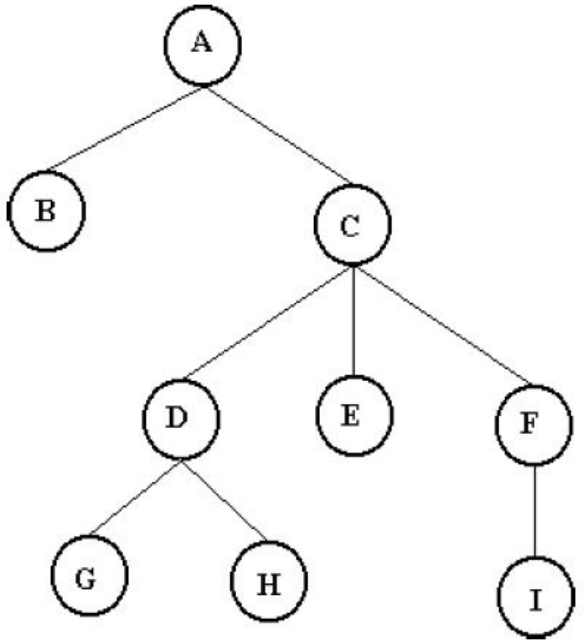
Aplicações das Árvores

- **Tomada de Decisão (Árvores de Decisão):** Em inteligência artificial e análise de dados, árvores são usadas para modelar processos de decisão, onde cada nó representa um teste e cada ramo uma possível resposta, levando a uma conclusão (folha).



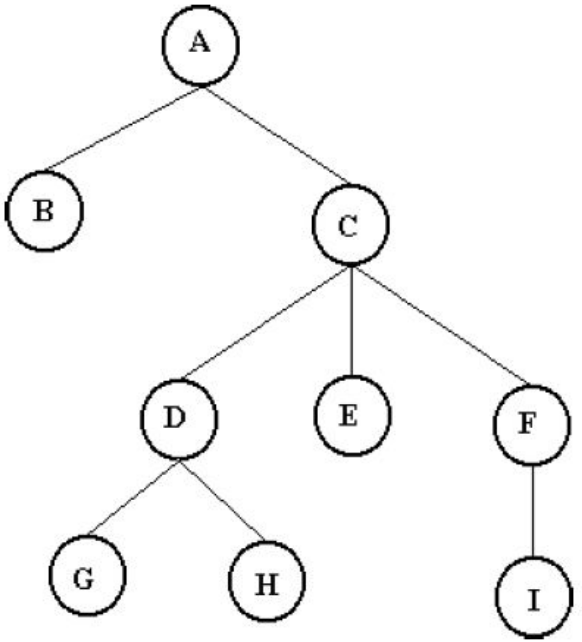
Aplicações das Árvores

- **Bancos de Dados (Indexação):** Tipos especiais de árvores (como Árvores B e B+) são cruciais para indexar grandes volumes de dados em bancos de dados, permitindo buscas muito rápidas.



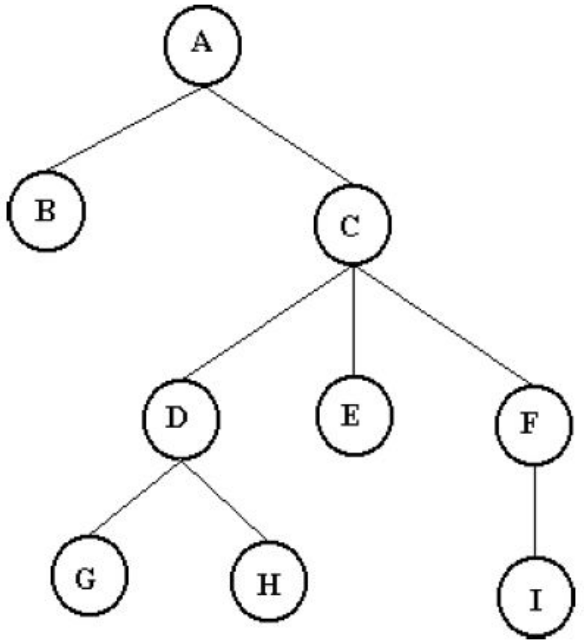
Aplicações das Árvores

- **Redes de Computadores:** A topologia de algumas redes pode ser modelada como árvores. Roteadores podem usar árvores para encontrar os melhores caminhos (árvores de abrangência mínima).



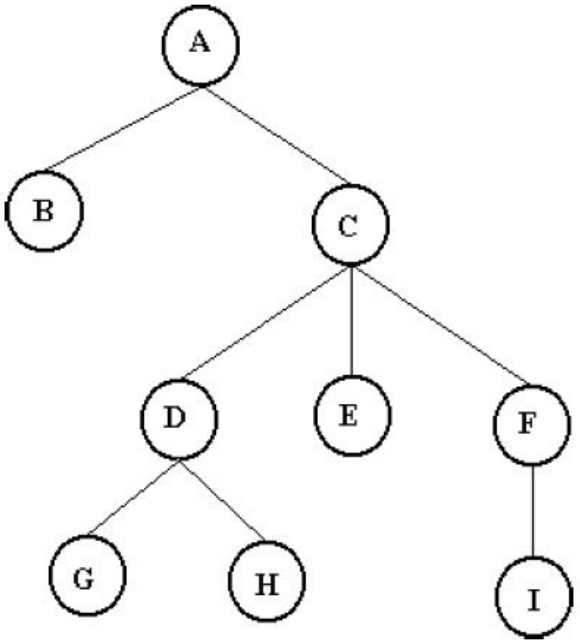
Aplicações das Árvores

- **Compiladores:** Compiladores transformam o código que escrevemos em uma estrutura de árvore (Árvore de Sintaxe Abstrata - AST) para analisar e otimizar o programa antes de gerar o código executável.



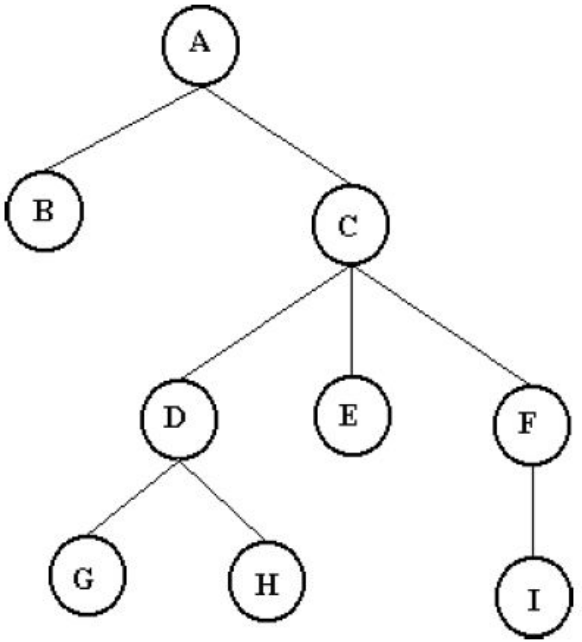
Aplicações das Árvores

- **Algoritmos de Busca e Ordenação:** Árvores de busca (como a Árvore Binária de Busca) são otimizadas para encontrar elementos rapidamente. Heaps (um tipo de árvore) são usados no eficiente algoritmo de ordenação Heapsort.



Aplicações das Árvores

- **Autocompletar e Dicionários (Tries):** Árvores Trie são especializadas para buscar palavras e prefixos, sendo usadas em corretores ortográficos e sugestões de busca.



Tipos Comuns de Árvores: Uma Família Diversificada

Árvore Binária (Binary Tree)

- **Definição:** É uma árvore de ordem 2, o que significa que cada nó pode ter no máximo dois filhos: um filho à esquerda e um filho à direita.
- **Características**: Simples de implementar e entender. É a base para muitas outras estruturas de árvore mais complexas.
- **Uso**: Representação de expressões matemáticas, árvores de decisão simples, base para árvores de busca.

Árvore Binária de Busca (Binary Search Tree - BST)

- **Definição:** É uma árvore binária com uma propriedade adicional importante:
  - para qualquer nó, todos os valores na sua subárvore esquerda são menores
  - que o valor do nó, e todos os valores na sua subárvore direita são maiores que o valor do nó.
- **Características**: Permite buscas, inserções e remoções eficientes (em média). A eficiência depende do balanceamento da árvore.
- **Uso**: Implementação de dicionários e mapas, busca rápida de elementos.

Árvores Balanceadas (AVL e Rubro-Negra)

- **Definição:** São árvores binárias de busca que utilizam regras e operações especiais (rotações) para manter a árvore balanceada, ou seja, evitar que ela fique muito "torta" para um lado. Isso garante que a altura da árvore seja sempre logarítmica em relação ao número de nós.
- Tipos Comuns:
  - **Árvore AVL**: Mantém um fator de balanceamento em cada nó (diferença entre as alturas das subárvores esquerda e direita) dentro de um limite (-1, 0 ou 1).
  - **Árvore Rubro-Negra (Red-Black Tree)**: Usa cores (vermelho ou preto) nos nós e um conjunto de regras para garantir o balanceamento.
- **Características**: Garantem desempenho eficiente (logarítmico) para busca, inserção e remoção, mesmo no pior caso.
- **Uso**: Amplamente usadas em implementações de mapas e conjuntos em bibliotecas padrão de linguagens, bancos de dados, agendadores de sistemas operacionais.

Árvores B e B+

- **Definição:** São árvores de busca multi-vias (não binárias), onde cada nó pode ter muitos filhos. São otimizadas para sistemas que leem e escrevem grandes blocos de dados, como discos rígidos.
- **Características**: Mantêm a árvore sempre balanceada e minimizam o número de acessos a disco necessários para encontrar um dado.
- **Uso**: Cruciais para sistemas de gerenciamento de bancos de dados (SGBDs) e sistemas de arquivos (como NTFS, HFS+).

Árvore Heap (monte)

- **Definição:** De maneira geral, um heap é uma forma eficiente de implementação de uma fila de prioridade.
- É uma árvore binária especial (geralmente completa ou quase completa) que satisfaz a "propriedade do heap":
  - Max Heap: O valor de um nó pai é sempre maior ou igual aos valores de seus filhos (o maior elemento está na raiz).
  - Min Heap: O valor de um nó pai é sempre menor ou igual aos valores de seus filhos (o menor elemento está na raiz).
- **Características**: Permite acesso rápido ao maior (ou menor) elemento e inserção/remoção eficientes.
- **Uso**: Implementação de filas de prioridade, algoritmo de ordenação Heapsort, algoritmos de grafos (como Dijkstra e Prim).

Trie (Árvore de Prefixos)

- **Definição:** Uma árvore especializada onde os nós não armazenam chaves diretamente. Em vez disso, a posição de um nó na árvore define a chave associada a ele. Cada caminho da raiz até um nó representa um prefixo.
- **Características**: Extremamente eficiente para buscar palavras baseadas em prefixos.
- **Uso**: Dicionários, autocompletar em buscas, corretores ortográficos, roteamento IP.

Representação e Implementação em Java

Implementação em Java

- A implementação de árvores pode ser realizada usando vetores ou alocação dinâmica de memória (com TAD)
- Vamos focar em alocação dinâmica de memória!
- Muitos exemplos na Internet utilizam POO, mas como não foi abordados no curso ainda, não utilizaremos POO.
- É muito comum usar funções recursivas!

Implementação em Java

- Exemplo: uma árvore binária de busca que armazena um número.

Implementação em Java

- A classe para uma árvore de ordem 2:

```
class No {

    int valor;
    No esquerda;
    No direita;
}
```

Implementação em Java

- O nó raiz, será declarado no início do programa principal (main):

```
public static void main(String[] args) {
    No raiz = null;
```

Implementação em Java

- Inserir um novo dado (recursivo e ordenado):

```
static No inserirRecursivamente(No raiz, int valor) {
    if (raiz == null) {
        No novo = new No();
        novo.valor = valor;
        return novo;
    }
    if (valor < raiz.valor) {
        raiz.esquerda = inserirRecursivamente(raiz.esquerda, valor);
    } else {
        raiz.direita = inserirRecursivamente(raiz.direita, valor);
    }
    return raiz;
}
```

Implementação em Java

- Inserir um novo dado ordenadamente sem utilizar recursividade:

```
static No inserirSemRecursividade(No raiz, int valor) {
    No novo = new No();
    novo.valor = valor;

    if (raiz == null) {
        return novo;
    }

    No atual = raiz;
    No anterior = null;

    while (atual != null) {
        anterior = atual;

        if (valor < atual.valor) {
            atual = atual.esquerda;
        } else {
            atual = atual.direita;
        }
    }

    if (valor < anterior.valor) {
        anterior.esquerda = novo;
    } else {
        anterior.direita = novo;
    }

    return raiz;
}
```

Implementação em Java

- Busca recursiva em árvore binária ordenada:

```
static boolean buscaRecursiva(No raiz, int valor) {
    if (raiz == null) {
        return false;
    }
    if (valor == raiz.valor) {
        return true;
    } else if (valor < raiz.valor) {
        return buscaRecursiva(raiz.esquerda, valor);
    } else {
        return buscaRecursiva(raiz.direita, valor);
    }
}
```

Implementação em Java

- Busca **não recursiva** em árvore binária ordenada:

```
static boolean buscarSemRecursividade(No raiz, int valor) {
    No atual = raiz;

    while (atual != null) {
        if (valor == atual.valor) {
            return true;
        } else if (valor < atual.valor) {
            atual = atual.esquerda;
        } else {
            atual = atual.direita;
        }
    }

    return false;
}
```

Implementação em Java

- Apresenta toda a árvore em ordem (obrigatório usar recursividade ou pilha):

```
static void emOrdem(No raiz) {
    if (raiz != null) {
        emOrdem(raiz.esquerda);
        System.out.print(raiz.valor + " ");
        emOrdem(raiz.direita);
    }
}
```

Implementação em Java

- Exemplo de programa principal com testes:

```
public static void main(String[] args) {
    No raiz = null;

    raiz = inserirRecursivamente(raiz, 50);
    raiz = inserirSemRecursividade(raiz, 30);
    raiz = inserirSemRecursividade(raiz, 70);
    raiz = inserirSemRecursividade(raiz, 20);
    raiz = inserirSemRecursividade(raiz, 40);
    raiz = inserirSemRecursividade(raiz, 60);
    raiz = inserirSemRecursividade(raiz, 80);

    System.out.print("Busca por 40: ");
    System.out.println(buscarSemRecursividade(raiz, 40)); // true

    System.out.print("Busca por 90: ");
    System.out.println(buscaRecursiva(raiz, 90)); // false

    System.out.print("Valores em ordem: ");
    emOrdem(raiz);  // Deve imprimir em ordem crescente
}
```

Exercícios

Atividade 1

- Utilizando o exemplo apresentado, implemente a função para remover nós: `remover(No raiz, int valor)`
  - Descrição da tarefa:
    - Receber como parâmetros a raiz da árvore e o valor a ser removido.
    - Localizar o nó que contém esse valor (iteração, como nas funções de busca/inserção sem recursão).
    - Tratar os três casos de remoção:
      - Nó folha (zero filhos): basta “desligar” do pai.
      - Nó com um filho: substituir o nó pelo seu único filho.
      - Nó com dois filhos:
        - Encontrar o sucessor (menor valor na subárvore direita).
        - Copiar o valor do sucessor para o nó a remover.
        - Remover o sucessor (que agora é folha ou tem um único filho).

Atividade 2

- Utilizando o exemplo apresentado, implementar:
  - **altura(No raiz)**: retornar a altura máxima da árvore.
  - **ehBalanceada(No raiz)**: dizer se, para todo nó, a diferença entre as alturas das subárvores esquerda e direita é no máximo 1.
- Descrição da tarefa:
  - **altura(Node raiz)**: Iterar de forma não-recursiva (ou recursiva,
    - se já tiverem visto) para medir o comprimento do caminho mais longo da raiz até uma folha.
  - **ehBalanceada(Node raiz)**: Para cada nó da árvore, comparar as alturas das duas subárvores (use seu método de altura).
    - Se alguma diferença > 1, retornar false; caso contrário, true.