DCC
DEPARTAMENTO DE
CIÊNCIA DA COMPUTAÇÃO

UFMG
UNIVERSIDADE FEDERAL
DE MINAS GERAIS

Estruturas de Dados

Análise de Complexidade

Professores: Anisio Lacerda
Lucas Ferreira
Wagner Meira Jr.
Washington Cunha

Projeto de Algoritmos

- Projeto de algoritmos
  - Análise do problema
  - Decisões de projeto
    - Tipos Abstratos de Dados
    - Algoritmo a ser utilizado

- Principais Perguntas:
  - O Algoritmo funciona?
  - **O Algoritmo é eficiente?**

Projeto de Algoritmos

- A eficiência de um algoritmo pode ser medida com várias métricas. Por exemplo:
  - tempo de execução
  - espaço ocupado
  - …

- Esse tipo de estudo é chamado:
  Análise de Algoritmos

Medida do Custo por meio de um Modelo Matemático

- Deve ser especificado o conjunto de operações e seus custos de execuções.
  - É mais usual ignorar o custo de algumas das operações e considerar apenas as operações mais significativas.
- Ex.: algoritmos de ordenação: Consideramos o número de comparações entre os elementos do conjunto a ser ordenado e ignoramos as operações aritméticas, de atribuição e manipulação de índices, caso existam.

Custo de um Algoritmo

- Determinando o menor custo possível para resolver problemas de uma dada classe, temos a medida da dificuldade inerente para resolver o problema.
- **Algoritmo Ótimo:** Quando o custo de um algoritmo é igual ao menor custo possível, o algoritmo é **ótimo** para a medida de custo considerada.

Função de Complexidade

- O custo de execução de um algoritmo é dado por uma função de custo ou **função de complexidade *f***.

- *f(n)* é a medida do custo necessário para executar um algoritmo para um **problema de tamanho *n***.

Função de Complexidade

- Função de **complexidade de tempo**:
  - *f(n)* mede o custo em **número de operações** para executar um algoritmo em um problema de tamanho n

- Função de **complexidade de espaço:**
  - *f(n)* mede a **memória** necessária para executar um algoritmo em um problema de tamanho n

Comparação entre os Algoritmos

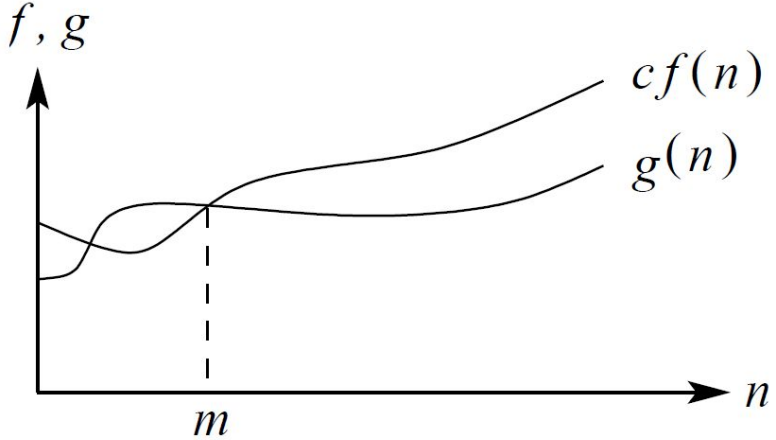
- Comparação entre os algoritmos dos programas MaxMin1, MaxMin2 e MaxMin3, considerando o número de comparações como medida de complexidade.

Os três algoritmos	$f(n)$		
	Melhor caso	Pior caso	Caso médio
MaxMin1	$2(n-1)$	$2(n-1)$	$2(n-1)$
MaxMin2	$n-1$	$2(n-1)$	$3n/2-3/2$
MaxMin3	$3n/2-2$	$3n/2-2$	$3n/2-2$

Notação *O*

Dada uma função *f(n)* definimos formalmente o conjunto *O(f(n))* como:

$$O(f(n)) := \{g(n) : \exists c, m > 0 \text{ t.q. } g(n) \leq cf(n), \forall n \geq m\}$$



Notação O

$$O(f(n)) := \{g(n) : \exists c, m > 0 \text{ t.q. } g(n) \leq cf(n), \forall n \geq m\}$$

Exemplo 2:

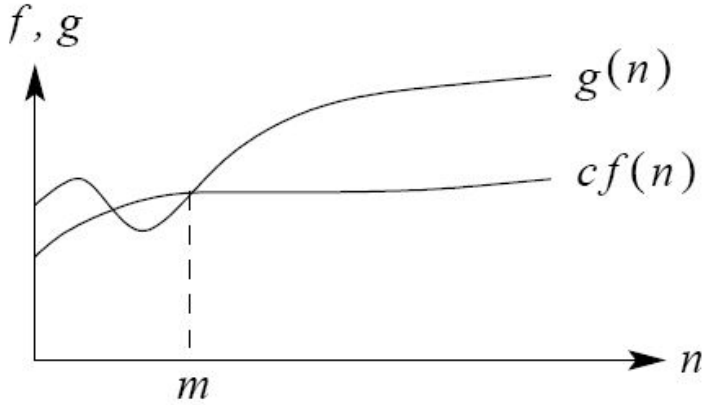
Mostre que $f(n) = n$ pertence ao conjunto $O(n^2)$.

- Tome $c = 1$ e $m = 1$.
- $n \leq n^2$ para todo $n \geq 1$.

Notação Ω

Dada uma função *f(n)* definimos formalmente o conjunto *Ω(f(n))* como:

$$\Omega(f(n)) := \{g(n) : \exists c, m > 0 \text{ t.q. } g(n) \geq cf(n), \forall n \geq m\}$$



Notação Ω

$$\Omega(f(n)) := \{g(n) : \exists c, m > 0 \text{ t.q. } g(n) \geq cf(n), \forall n \geq m\}$$

Exemplo:

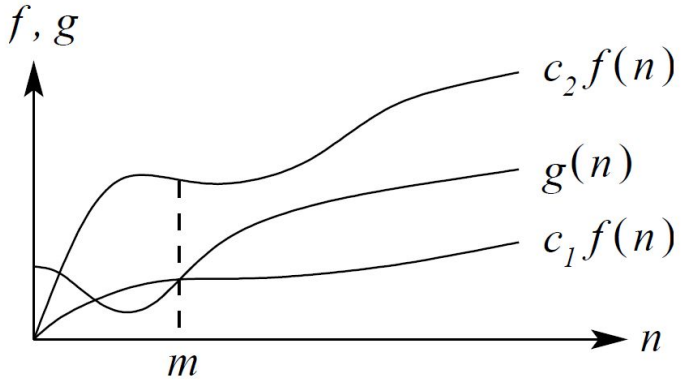
Mostre que $f(n) = 3n^3 + n^2$ pertence ao conjunto $\Omega(n^3)$:

- Tome $c = 1$ e $m = 1$.
- Logo $3n^3 + n^2 \geq n^3$, para todo $n \geq 1$.
- Subtraindo n^3 dos dois lados.
- Temos $2n^3 + n^2 \geq 0$, para todo $n \geq 1$.

Notação θ

Dada uma função *f(n)* definimos formalmente o conjunto $\Theta(f(n))$ como:

$$\Theta(f(n)) := \{g(n) : \exists c_1, c_2, m > 0 \text{ t.q.}$$
$$c_1 f(n) \leq g(n) \leq c_2 f(n), \forall n \geq m\}$$



Notação θ

$$\Theta(f(n)) := \{g(n) : \exists c_1, c_2, m > 0 \text{ t.q.}$$
$$c_1 f(n) \leq g(n) \leq c_2 f(n), \forall n \geq m\}$$

Exemplo:

Mostre que $f(n) = n^2 + 400n$, pertence a $\theta(n^2)$

- Tome $c_1 = 1$ e $m_1 = 1$.
- Temos $n^2 \leq n^2 + 400n$, para todo $n \geq 1$.
- Subtraia n^2 dos dois lados.
- Temos então $0 \leq 400n$, para todo $n \geq 1$.

Notação θ

$$\Theta(f(n)) := \{g(n) : \exists c_1, c_2, m > 0 \text{ t.q.}$$
$$c_1 f(n) \leq g(n) \leq c_2 f(n), \forall n \geq m\}$$

Exemplo:

Mostre que $f(n) = n^2 + 400n$, pertence a $\theta(n^2)$

- Tome $c_2 = 2$ e $m_2 = 400$.
- Temos $n^2 + 400n \leq 2n^2$, para todo $n \geq 400$.
- Subtraia n^2 dos dois lados.
- Temos então $400n \leq n^2$, para todo $n \geq 400$.

Notação θ

$$\Theta(f(n)) := \{g(n) : \exists c_1, c_2, m > 0 \text{ t.q.}$$
$$c_1 f(n) \leq g(n) \leq c_2 f(n), \forall n \geq m\}$$

Exemplo:

Mostre que $f(n) = n^2 + 400n$, pertence a $\theta(n^2)$

- Queremos que as duas afirmações valham.
- Então tome $m = max(m_1, m_2)$.
- Concluindo: tome $c_1 = 1$, $c_2 = 2$ e $m = 400$.
- Temos $n^2 \leq n^2 + 400n \leq 2n^2$, para todo $n \geq 400$.

Notação θ

- Note que a definição da notação θ soa como se estivéssemos utilizando simultaneamente as notações O e Ω.

Teorema: Sejam $f(n)$ e $g(n)$ funções. Temos que $f(n) = \theta(g(n))$ se e somente se $f(n) = O(g(n))$ e $f(n) = \Omega(g(n))$.

- A ideia por trás deste teorema é intuitiva e é um bom exercício de fixação demonstrá-lo!

Relações de recorrência

- Agora vamos pensar no custo do caso recursivo.
- Existe um custo constante *d* do produto.
- Se *T(n)* é o tempo gasto para calcular fatorial de *n*, então o tempo necessário para calcular fatorial de *n-1* é *T(n-1)*.
- Concluindo então o nosso caso recursivo.

```
int Fatorial(int n) {
    if(n <= 0)
        return 1;
    else
        return n * Fatorial(n-1);
}
```

$$T(n) := \begin{cases} c, & \text{se } n <= 0 \\ d + T(n-1), & \text{se } n > 0 \end{cases}$$

Relações de recorrência

- Como calcular a ordem de complexidade de uma relação de recorrência?

Relações de recorrência

- Como calcular a ordem de complexidade de uma relação de recorrência?
- A ideia gira em torno de expandir os termos do caso recursivo e concluir algo a respeito disso.

$$T(n) := \begin{cases} c, & \text{se } n <= 0 \\ d + T(n-1), & \text{se } n > 0 \end{cases}$$

Relações de recorrência

- Como calcular a ordem de complexidade de uma relação de recorrência?
- A ideia gira em torno de expandir os termos do caso recursivo e concluir algo a respeito disso.

$$T(n) := \begin{cases} c, & \text{se } n <= 0 \\ d + T(n-1), & \text{se } n > 0 \end{cases}$$

$$T(n) = d + T(n-1)$$
$$T(n-1) = d + T(n-2)$$
$$T(n-2) = d + T(n-3)$$
$$\dots$$
$$T(1) = d + T(0)$$

Relações de recorrência

- Substituindo estes termos, vamos reescrever *T(n)*.

$$T(n) := \begin{cases} c, & \text{se } n <= 0 \\ d + T(n-1), & \text{se } n > 0 \end{cases}$$

$$\begin{aligned} T(n) &= d + T(n-1) \\ T(n-1) &= d + T(n-2) \\ T(n-2) &= d + T(n-3) \\ &\dots \\ T(1) &= d + T(0) \end{aligned}$$

Relações de recorrência

- Substituindo estes termos, vamos reescrever *T(n)*.

$$T(n) = d + d + d + \cdots + d + c$$

$$T(n) = d + T(n-1)$$
$$T(n-1) = d + T(n-2)$$
$$T(n-2) = d + T(n-3)$$
$$\cdots$$
$$T(1) = d + T(0)$$

Relações de recorrência

- Substituindo estes termos, vamos reescrever *T(n)*.

$$T(n) = d \underbrace{+ d + d + \cdots + d}_{n \text{ vezes}} + c$$

$$T(n) = d + T(n-1)$$
$$T(n-1) = d + T(n-2)$$
$$T(n-2) = d + T(n-3)$$
$$\cdots$$
$$T(1) = d + T(0)$$

Relações de recorrência

- Substituindo estes termos, vamos reescrever *T(n)*.

$$T(n) = d \underbrace{+ d + d + \cdots + d}_{n \text{ vezes}} + c$$

$$T(n) = dn + c$$

$$T(n) = d + T(n-1)$$
$$T(n-1) = d + T(n-2)$$
$$T(n-2) = d + T(n-3)$$
$$\cdots$$
$$T(1) = d + T(0)$$

Relações de recorrência

- Substituindo estes termos, vamos reescrever *T(n)*.

$$T(n) = d \underbrace{+ d + d + \cdots + d}_{n \text{ vezes}} + c$$

$$T(n) = dn + c$$

$$T(n) = O(n)$$

$$T(n) = d + T(n-1)$$
$$T(n-1) = d + T(n-2)$$
$$T(n-2) = d + T(n-3)$$
$$\cdots$$
$$T(1) = d + T(0)$$

Relações de recorrência

- Note que agora é um pouco mais complicado utilizar o mesmo procedimento que fizemos com fatorial.
- Iremos utilizar uma outra abstração para fazer este cálculo, chamada **árvore de recursão**.

$$T(n) := \begin{cases} c, & \text{se } n <= 1 \\ d + T(\frac{n}{2}), & \text{se } n > 0 \end{cases}$$

Relações de recorrência

- Agora conseguimos finalizar a expansão dos termos da relação de recorrência.

$$T(n) := \begin{cases} c, & \text{se } n <= 1 \\ d + T(\frac{n}{2}), & \text{se } n > 0 \end{cases}$$

$$T(n) = d + \underbrace{d + d + \cdots + d}_{log\ n\ \text{vezes}} + c$$

$$T(n) = d(logn) + c$$

$$T(n) = O(logn)$$

DCC
DEPARTAMENTO DE
CIÊNCIA DA COMPUTAÇÃO

UFMG
UNIVERSIDADE FEDERAL
DE MINAS GERAIS

Estruturas de Dados

Teorema Mestre

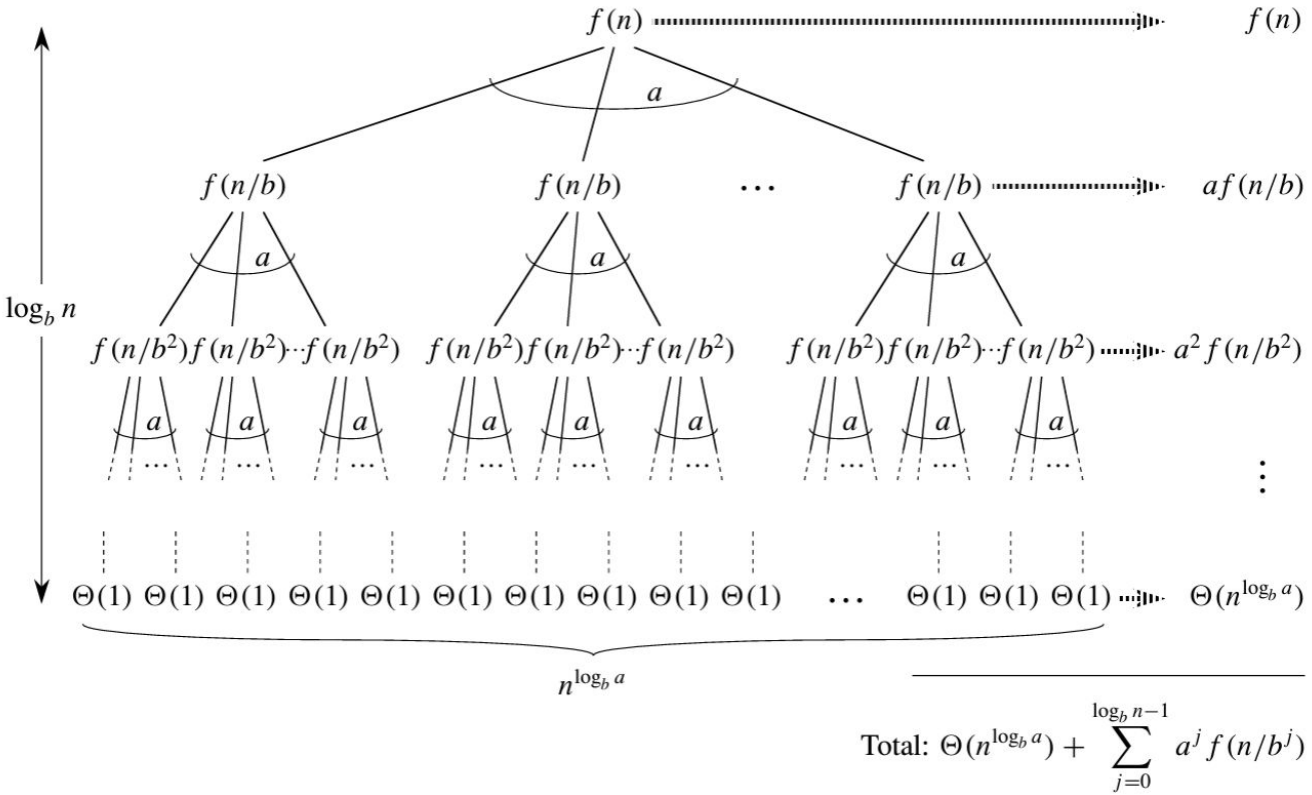
Professores: Anisio Lacerda
Lucas Ferreira
Wagner Meira Jr.
Washington Cunha

Divisão e Conquista

- Vimos na última aula que nem sempre é trivial resolver uma relação de recorrência
- Alguns algoritmos exibem comportamentos semelhantes, como por exemplo:
  - Fazer *a* chamadas recursivas para instâncias de tamanho *n/b*.
  - faz um processamento de custo *f(n)* com as respostas obtidas e retorna uma solução.
- Nesses casos a relação de recorrência será:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

Teorema Mestre - Intuição



Teorema Mestre - Enunciado

- Sejam $a \geq 1$ e $b > 1$ constantes, $f(n)$ uma função assintoticamente positiva e $T(n)$ uma relação de recorrência definida da forma:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

Caso 1: Se $f(n) = O(n^{log_b(a-\epsilon)}), \epsilon > 0 \implies T(n) = \Theta(n^{log_b(a)})$.

Caso 2: Se $f(n) = \Theta(n^{log_b(a)}) \implies T(n) = \Theta(n^{log_b(a)} log(n))$.

Caso 3: Se $\begin{cases} f(n) = \Omega(n^{log_b(a+\epsilon)}), \epsilon > 0 \\ af(n/b) \leq cf(n), \forall n \geq m, c < 1 \end{cases} \implies T(n) = \Theta(f(n))$.

DCC
DEPARTAMENTO DE
CIÊNCIA DA COMPUTAÇÃO

UFMG
UNIVERSIDADE FEDERAL
DE MINAS GERAIS

Estruturas de Dados

Ordenação: Introdução

Professores: Anisio Lacerda
Lucas Ferreira
Wagner Meira Jr.
Washington Cunha

Ordenação

- Objetivo:
  - Rearranjar os itens de um vetor ou lista de modo que suas chaves estejam ordenadas de acordo com alguma regra.
- Estrutura:
  - um vetor v vai ser um `Item *v` ou `Item v[max]`

```
typedef int TipoChave;
typedef struct {
 ChaveTipo chave;
 /* outros componentes */
} Item;
```

Critérios de Classificação

- Localização dos dados
- Estabilidade
- Adaptabilidade
- Uso da memória
- Movimentação dos dados
- Estratégia de ordenação: Comparação de Chaves x Outros

Métodos de Ordenação

- Métodos simples:
  - Bolha
  - Seleção
  - Inserção
- Métodos eficientes:
  - Quicksort
  - Mergesort
  - Heapsort
- Métodos lineares:
  - Bucketsort
  - Radixsort

DCC
DEPARTAMENTO DE
CIÊNCIA DA COMPUTAÇÃO

UFMG
UNIVERSIDADE FEDERAL
DE MINAS GERAIS

Estruturas de Dados

TADs: Listas, Filas, Pilhas

Professores: Anisio Lacerda
Lucas Ferreira
Wagner Meira Jr.
Washington Cunha

Módulo 3 - Sumário

- Introdução
  - Tipos Abstratos de Dados
- Listas Lineares
  - Implementação por arranjos (sequencial)
  - Implementação por apontadores (encadeada)
- Filas, Pilhas
  - Implementação por arranjos (sequencial)
  - Implementação por apontadores (encadeada)

Tipos Abstratos de Dados (TADs)

- **Construções que agrupam a estrutura de dados juntamente com as operações que podem ser feitas sobre esses dados**

- O TAD **encapsula** a estrutura de dados, fornecendo acesso apenas através de uma “interface” (conjunto de funções públicas)
  - Usuário do TAD só “enxerga” a interface, não a implementação específica

TAD: Pilha

- Duas Implementações:
  - Sequencial (uso de arranjos, alocação estática)
  - Encadeada (uso de apontadores, alocação dinâmica)
- Operações:
  - Criar uma nova pilha (construtor)
  - Testar se a pilha está *vazia*
  - **Empilhar** um item
  - **Desempilhar** um item
  - Limpar a pilha

Disclaimer: os códigos que serão apresentados devem ser considerados como exemplos. Eles não são, necessariamente, os mais modulares ou eficientes…

TAD: Fila

- Duas Implementações:
  - Sequencial (uso de arranjos, alocação estática)
  - Encadeada (uso de apontadores, alocação dinâmica)
- Operações:
  - Criar uma nova fila (construtor)
  - Testar se a fila está *vazia*
  - **Enfileirar** um item: colocar um item no final da fila
  - **Desenfileirar** um item: retirar um item do início da fila
  - Limpar a fila

Disclaimer: os códigos que serão apresentados devem ser considerados como exemplos. Eles não são, necessariamente, os mais modulares ou eficientes…

TAD: Lista

- Duas Implementações:
  - Sequencial (uso de arranjos, alocação estática)
  - Encadeada (uso de apontadores, alocação dinâmica)
- Operações:
  - Criar uma nova lista (construtor)
  - Métodos de Acesso (Get, Set)
  - Testar se é uma lista *vazia*
  - Inserção: no início, no final, em uma posição *p*
  - Remoção: do início, do final, de uma posição *p*
  - Pesquisar por uma chave
  - Imprimir a Lista
  - Limpar a Lista

Disclaimer:
os códigos que serão apresentados devem ser considerados como exemplos. Eles não são, necessariamente, os mais modulares ou eficientes…

DCC
DEPARTAMENTO DE
CIÊNCIA DA COMPUTAÇÃO

UFMG
UNIVERSIDADE FEDERAL
DE MINAS GERAIS

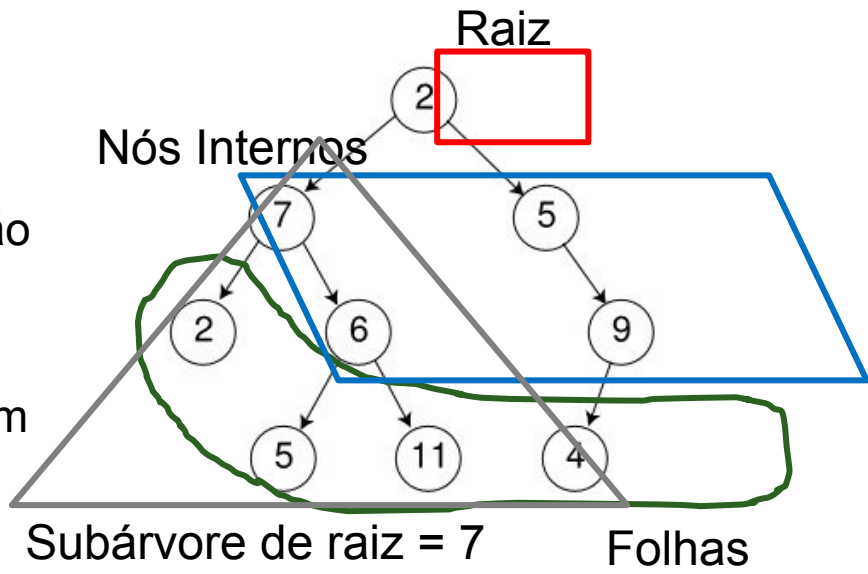
Estruturas de Dados

Árvores

Professores: Anisio Lacerda
Lucas Ferreira
Wagner Meira Jr.
Washington Cunha

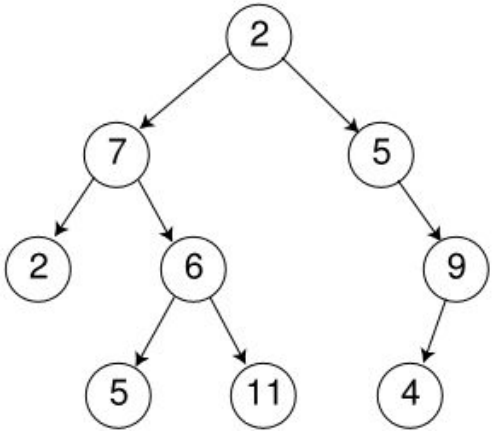
Conceitos básicos

- Árvores organizam os dados em uma estrutura hierárquica
  - *Pais* e *Filhos*
  - *Antecessores* e *Sucessores*
- Cada elemento é chamado de **nó**, e nós são ligados por **arestas**
- O primeiro nó da árvore é a **raiz**
- Os nós **folha** são aqueles que não possuem "filhos"
- Os outros nós são chamados de **nós internos**
- **Recursividade**: o filho de um nó é a raiz de uma outra subárvore



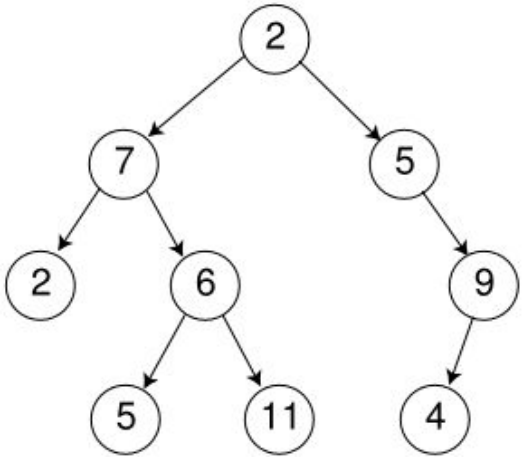
Mais Conceitos

- Níveis
  - A raiz da árvore está no nível 0
  - Se um nó está no nível i, os seus filhos estão no nível $i+1$
- Caminho
  - É a sequência de nós percorrida entre quaisquer 2 nós
  - Em uma árvore, só existe um único caminho entre quaisquer 2 nós
  - Tamanho ou Comprimento do caminho é igual ao número de arestas percorridas (que é igual ao número de nós -1)



Altura / Profundidade

- A **profundidade** de um nó é o comprimento do caminho entre a raiz e aquele nó
- A **altura** de um nó é o comprimento do caminho mais longo desse nó até uma folha
- **Altura da árvore** é igual a altura da raiz que é igual à sua maior profundidade

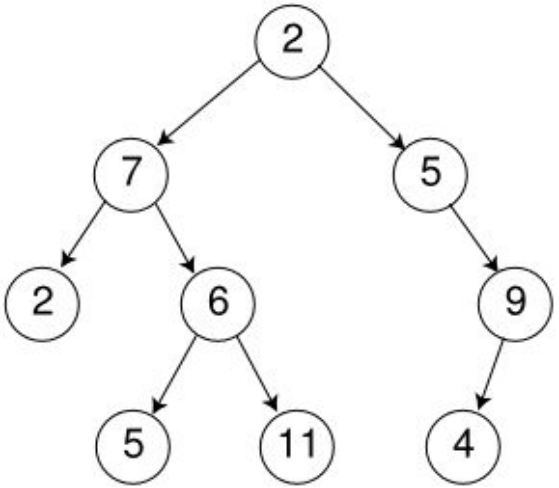


Profundidade do nó 7: **1**
Altura do nó 7: **2**
Altura da Árvore: **3**

Árvores Binárias

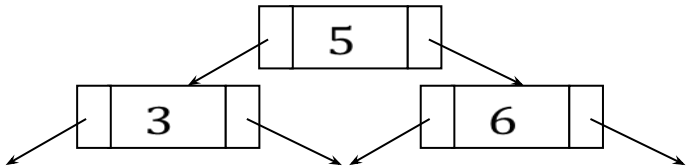
- Em uma árvore binária, cada nó pode ter no máximo 2 filhos (subárvores da esquerda e da direita)

- Algumas aplicações impõem restrições na organização desses nós
  - Heap
  - Árvore Binária de Pesquisa
  - …



TAD Árvore Binária

- Implementação usando apontadores
  - Cada nó vai armazenar um item e apontadores para os filhos da esquerda e direita
- Operações Comuns
  - Criar uma árvore
  - Inserir Itens
  - Remover Itens
  - Pesquisar por um item
  - "Percorrer" ou "Caminhar" na árvore
    - Para imprimir todos os itens, por exemplo



Essas operações dependem da organização desejada e serão estudadas com detalhes no contexto dos algoritmos de ordenação e pesquisa

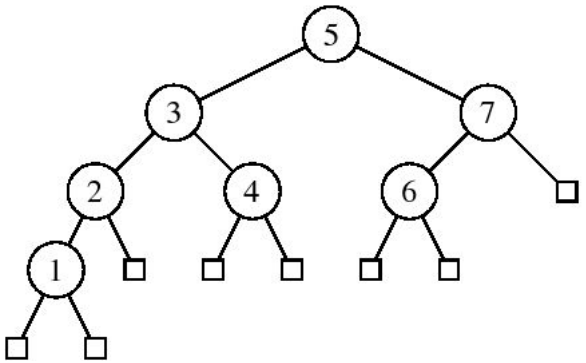
Caminhamento em Árvores

- Pré-ordem (Pré-fixada)
  - Visita o nó e depois os filhos da esquerda e da direita
- In-ordem (Central ou Infixada)
  - Visita o filho da esquerda, o nó, e depois o filho da direita
- Pós-ordem (Pós-fixada)
  - Visita os filhos da esquerda e da direita e depois o nó
- Caminhamento por nível
  - Visita os nodos de cada nível em sequência

Pré-Ordem

- Imprime o item, e depois visita recursivamente as árvores da esquerda e da direita

```
void ArvoreBinaria::PreOrdem(TipoNo *p){
    if(p!=NULL){
        p->item.Imprime();
        PreOrdem(p->esq);
        PreOrdem(p->dir);
    }
}
```



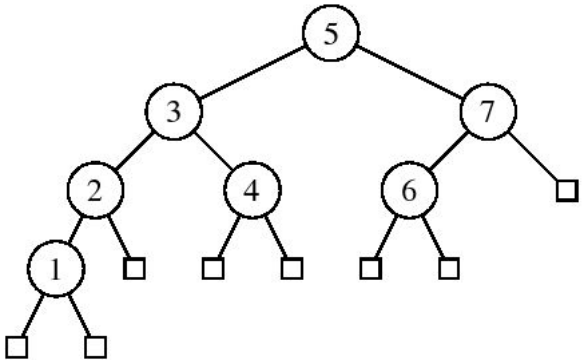
5 3 2 1 4 7 6

In-Ordem (ou central)

- Visita recursivamente a árvore da esquerda, imprime o item, e depois visita a subárvore da direita

```
void ArvoreBinaria::InOrdem(TipoNo *p){
    if(p!=NULL){
        InOrdem(p->esq);
        p->item.Imprime();
        InOrdem(p->dir);
    }
}
```

1 2 3 4 5 6 7

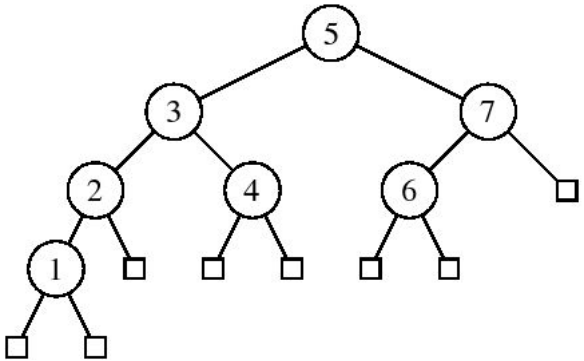


Pós-Ordem

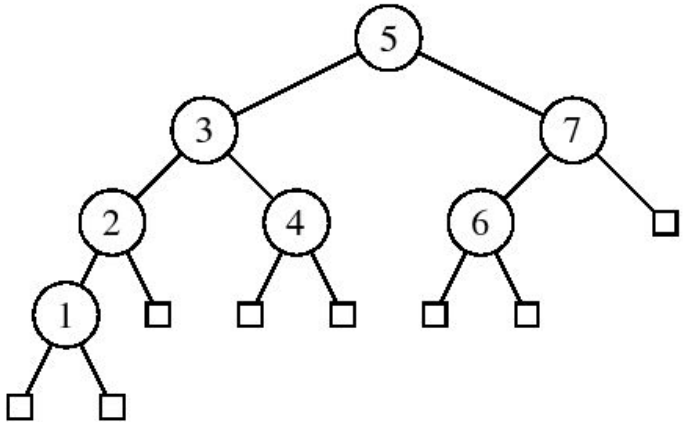
- visita recursivamente as árvores da esquerda e da direita e depois imprime o item

```
void ArvoreBinaria::PosOrdem(TipoNo *p){
    if(p!=NULL){
        PosOrdem(p->esq);
        PosOrdem(p->dir);
        p->item.Imprime();
    }
}
```

1 2 4 3 6 7 5



Exemplo de Caminhamento



- Pré-Ordem: 5, 3, 2, 1, 4, 7, 6
- Central: 1, 2, 3, 4, 5, 6 , 7
- Pós-Ordem: 1, 2, 4, 3, 6, 7, 5
- Por nível: 5, 3, 7, 2, 4, 6, 1