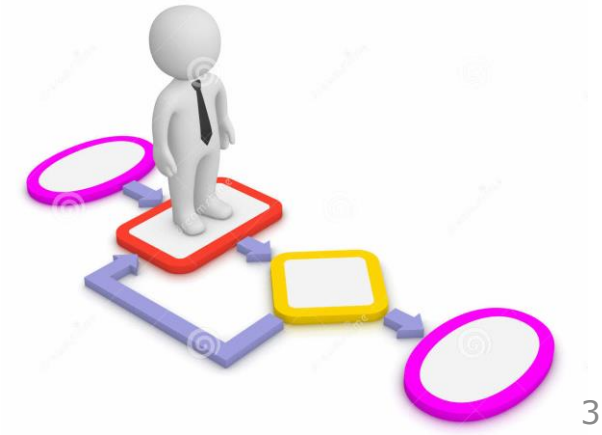
Estruturas de Dados

Tipos Abstratos de Dados (TADs) e Complexidade de Algoritmos

Aula de hoje

- Lógica
- A importância da lógica em Estruturas de Dados
- O que são Estruturas de Dados?
- Tipos Abstrato de Dados
- Complexidade

Lógica



A lógica no dia-a-dia

- O que você deve fazer ao entrar em um cômodo escuro?
- O que você deve fazer quando estiver chovendo?
- O que você deve fazer quando estiver fazendo frio?

O que é lógica? E sua importância nas Estruturas de Dados!

> *"A **Lógica** é a forma de organizar os **pensamentos** e demonstrar o **raciocínio** de maneira **coerente**, permitindo escolher caminhos* para resolver problemas."*

Testando a sua lógica!

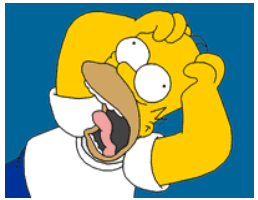
Você está numa cela onde existem **duas portas**, cada uma vigiada por um guarda. Existe uma porta que dá para a **liberdade**, e outra para a **morte.**

Você está livre para **escolher a porta** que quiser e por ela sair.

Poderá fazer apenas **uma pergunta** a um dos dois guardas que vigiam as portas.

Um dos guardas sempre **fala a verdade**, e o outro **sempre mente** e você não sabe quem é o **mentiroso** e quem fala a **verdade**.

Que pergunta você faria?



Testando a sua lógica!

Resposta - Pergunte a qualquer um deles:

Qual a porta que o seu companheiro apontaria como sendo a porta da liberdade?

Explicação: O mentiroso apontaria a porta da morte como sendo a porta que o seu companheiro (o sincero) diria que é a porta da liberdade, já que se trata de uma mentira da afirmação do sincero. E o sincero, sabendo que seu companheiro (o mentiroso) sempre mente, diria que ele apontaria a porta da morte como sendo a porta da liberdade.

Conclusão: Os dois apontariam a porta da morte como sendo a porta que o seu companheiro diria ser a porta da liberdade.

Portanto, é só seguir pela outra porta.

A importância da lógica em Estruturas de Dados

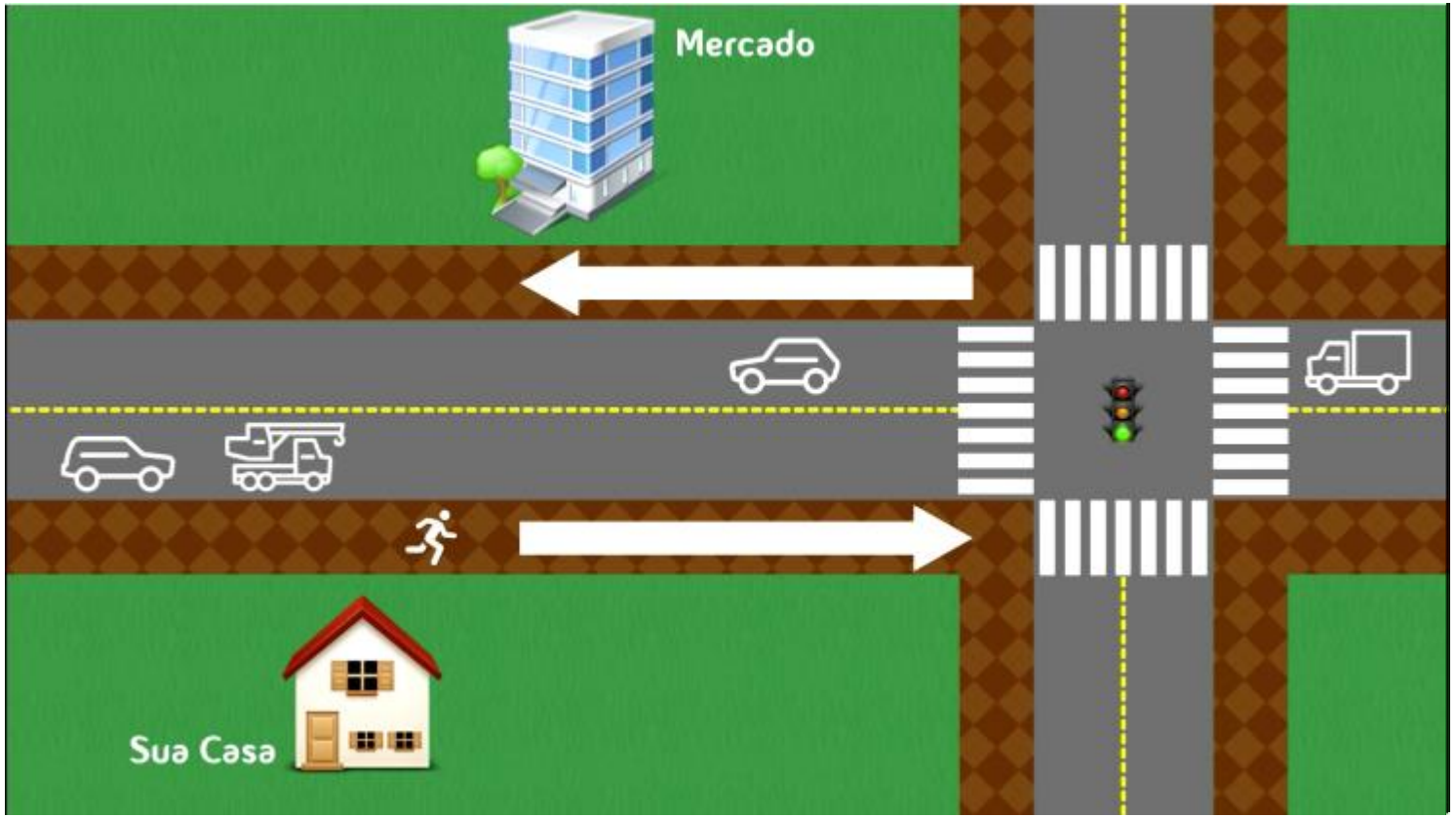
Lógica nas Estruturas de Dados!!!

- Em estruturas de dados, tudo a ser realizado deve está baseado no raciocínio lógico

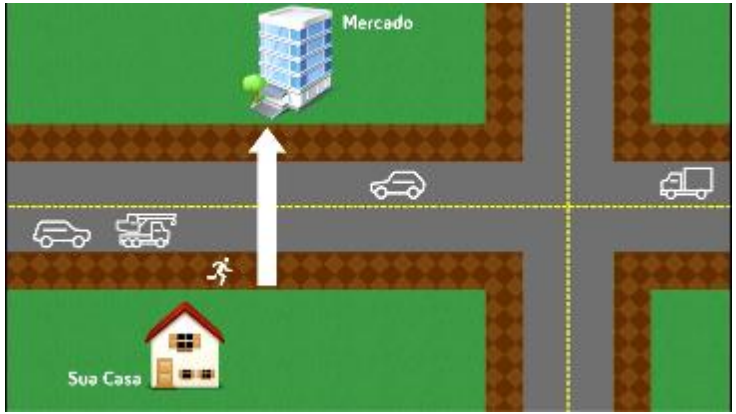
Isso permite que os problemas sejam solucionados da melhor maneira possível de acordo com o contexto em que se encontram

Imagine...

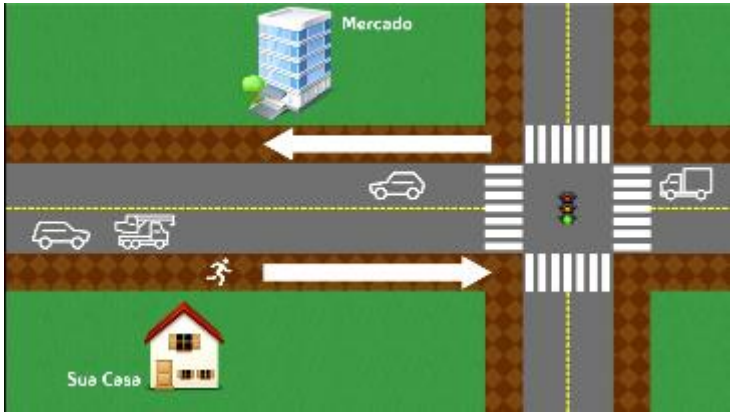
O objetivo foi atingido?



Qual a “maneira ideal” de atingir o objetivo?



Performance

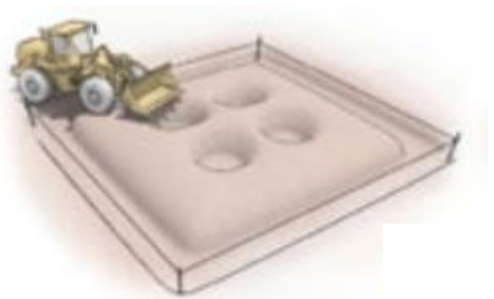


Segurança

Dependendo do problema, as vezes é necessário abrir mão da performance para obter uma maior segurança, e outras vezes, pode-se abrir mão da segurança para melhorar a performance

Lógica em Estruturas de Dados!!!

- Requisito básico para **aprender a programar em Estruturas de Dados**
- Começar a programar sem lógica: É inversão de papéis
  - "Não é impossível", mas o gasto e o tempo perdidos são muito maiores quando a tarefa não é executada na ordem adequada



Estruturas de Dados

Estruturas de Dados

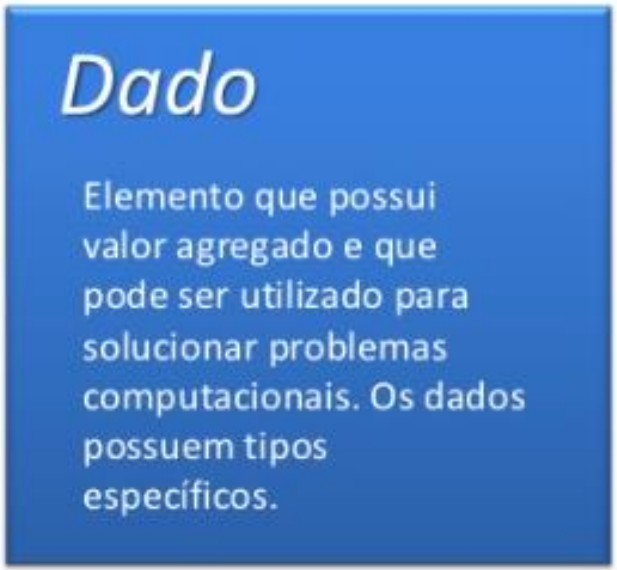
- O que são estruturas de dados?

Maneira de organizar dados para fins de operar sobre eles

As estruturas de dados são formas de distribuir e relacionar os dados disponíveis, de modo a tornar mais eficientes os algoritmos que manipulam estes dados

Estruturas de Dados

- Uma estrutura de dados pode ser dividida em dois pilares fundamentais:



Estruturas de Dados

- Uma estrutura de dados pode ser dividida em dois pilares fundamentais:

Principais tipos de estruturas de dados

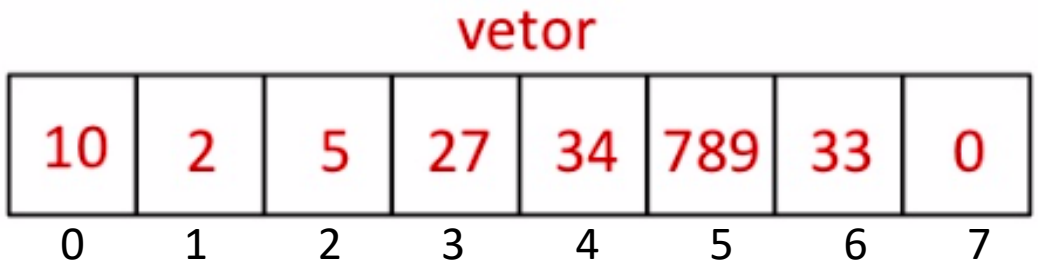
- Vetores:

  É a estrutura de dados mais simples e mais utilizadas dentre todas

- Principais características:
  - Adição e pesquisa de novos elementos de forma aleatória
  - Acesso aos elementos por meio de índices
  - Possuem tamanho finito de elementos
  - Carregam dados de tipos específicos
  - Indexação com o início Zero
  - Unidimencional e Bidimensional

Principais tipos de estruturas de dados

- Vetor unidimensional:



- Vetor[4] = 34
- Vetor[6] = 33
- Qual o tamanho do vetor?

Principais tipos de estruturas de dados

- Como declarar um vetor unidimensional?

```
<tipo> <nome> [];
```

```
int valor[10];
```

- O que o programa faz?

```
int main() {

    int nums[10];

    nums[0] = 100;
    nums[1] = 99;
    nums[2] = 50;
    nums[3] = 25;
    nums[4] = 4;
    nums[5] = 5;
    nums[6] = 60;
    nums[7] = 0;
    nums[8] = -1;
    nums[9] = 9;

    cout << nums[5] + nums[9] << endl;
    cout << nums[0];

    return 0;
}
```

Principais tipos de estruturas de dados

- Vetor bidimensional:
  - Vetorb[0][1] = 2
  - Vetorb[1][1] = 50
  - Qual o tamanho do vetor?

Principais tipos de estruturas de dados

- Como declarar um vetor bidimensional?
- O que o programa faz?

```
int main() {

    int tabela[2][2];
    tabela[0][0] = 10;
    tabela[0][1] = 100;  //10, 100
    tabela[1][0] = 20;   //20, 111
    tabela[1][1] = 111;

    int tabela2[2][2] = { {10, 100},
                          {20, 111} };

    cout << "{{" << tabela2[0][0] << "," << tabela2[0][1] << "}, {" <
<
                    tabela2[1][0] << "," << tabela2[1][1] << "}}" <<
endl;

    return 0;
}
```

Principais tipos de estruturas de dados

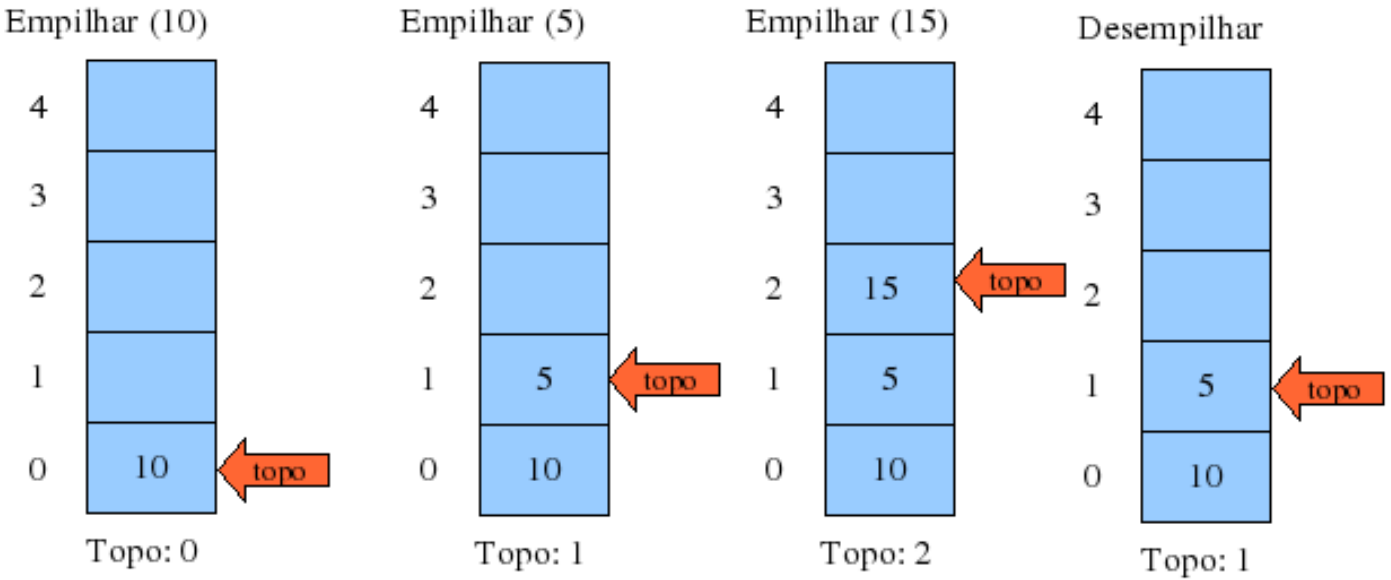
- Pilha:

  É uma estrutura de dados amplamente utilizada e que implementa a ideia de pilha de elementos

- Principais características:
  - LIFO, *Last-In First-out*
  - Permite a adição e remoção de elementos
  - O elemento a ser removido é sempre aquele mais novo
  - Simula a ideia de pilhas de elementos
  - Para que o acesso a um elemento da pilha ocorra, os demais acima devem ser removidos

Principais tipos de estruturas de dados

- Exemplo de funcionamento da Pilha:



- É possível implementar uma Pilha por meio de um vetor?

Principais tipos de estruturas de dados

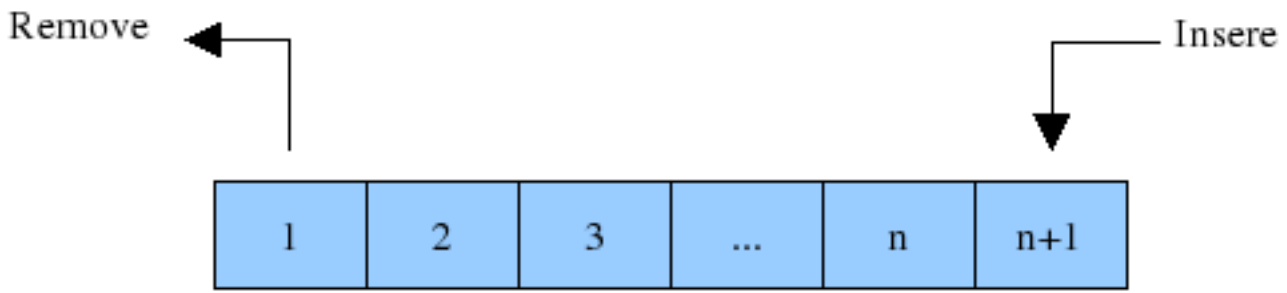
- Fila:

  É uma estrutura de dados amplamente utilizada e que implementa a ideia de lista de elementos

- Principais características:
  - FIFO, *First-In First-out*
  - Permite a adição e remoção de elementos
  - O elemento a ser removido é sempre o primeiro a entrar
  - As operações de entrada e saída sempre ocorrem nas extremidades

Principais tipos de estruturas de dados

- Exemplo de funcionamento da Fila:



- Qual a diferença da **Fila** para um **Vetor**?

Principais tipos de estruturas de dados

- Outras estruturas de dados:
  - Árvores
  - Árvores binárias
  - Grafos
  - Variações da Pilha/Lista dinâmica/encadeada

Exemplos

- **Problema 1**: Manipular um conjunto de fichas em um fichário
- **Solução**: Organizar as fichas em ordem alfabética
- **Métodos possíveis**: Inserir ou retirar uma ficha, procurar uma ficha, procurar uma ficha em determinada posição.
- **Estrutura de dados correspondente**: Lista ordenada (sequência de elementos dispostos em ordem)

Exemplos

- **Problema 2**: Organizar as pessoas que querem ser atendidas em um guichê
- **Solução**: Colocar as pessoas em fila
- **Métodos possíveis**: Sair da fila para ir ao atendimento e Entrar na fila para ser atendido
- **Estrutura de dados correspondente**: FILA – Sequência de elementos dispostos de maneira que o primeiro que chega é o primeiro que sai, FIFO

Exemplos

- **Problema 3**: Visualizar o conjunto de pessoas que trabalham em uma empresa, considerando sua função.
- **Solução**: Construir um organograma da empresa
- **Métodos possíveis**: Inserir ou Retirar certas funções, Localizar pessoas ser atendido
- **Estrutura de dados correspondente**: Árvore - Estrutura de dados que caracte-riza uma relação de hierarquia entre os elementos

Estruturas de dados

- Estruturas de dados são encontradas em praticamente todas as áreas da computação
- Banco de dados:
  - Resultados de consultas (lista de dados)
  - Indexação de arquivos de dados (árvores de busca)
- Sistemas Operacionais:
  - Controle de processos (filas de espera por recursos)
- Computação gráfica:
  - Manipulação de imagens (matrizes)

Tipo Abstrato de Dados - TAD

Projeto e Implementação da TAD

- Para projetar e implementar uma estrutura de dados, é preciso:
  - Uma **modelagem abstrata** dos objetos a serem **manipulados** e dos **métodos** sobre eles
    - Ex. Pilha: Empilhar e Desempilhar
  - Uma **modelagem concreta** do TDA, isto é, **como** armazenar o TDA em memória/disco e que algoritmos devem ser usados para **implementar os métodos**
    - Ex.: **Pilha** armazenada como **lista encadeada** ou **vetor**

Tipos de Dados

Definição

Caracteriza o conjunto de valores a que uma constante pertence, ou que podem ser assumidos por uma variável ou expressão, ou que podem ser gerados por uma função (Wirth, 1976)

Tipos simples: int, float, double, etc

Tipos estruturados: structs

Tipos de Dados Abstratos - TADs

Definição

Pode ser visto como um modelo matemático, acompanha-do dos métodos definidos sobre o modelo (Ziviani, 2003)

É usado para **encapsular** tipos de dados (pensar em termos dos métodos suportados e não como são implementadas)

Não há necessidade de saber a representação interna de um tipo de dado

Modelo Matemático

Um TAD pode ser visto como uma tupla (*v, o*), onde

v é o conjunto de valores

o é o conjunto de métodos aplicados sobre esses valores

Exemplo, tipo REAL

$v = R$

$o = \{+, -, *, /, =, <, >, <=, >=\}$

Implementação de uma TAD

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main(){
4      Tipos Abstratos de Dados ou "TAD":
5      incluem as operações para a
6      manipulação dos dados
7      Ex:
8      - criação da estrutura
9      - inclusão de um elemento
10     - remoção de um elemento
11     - acesso a um elemento
12     - etc
13
14     system("pause");
15     return 0;
16 }
```

Implementação de uma TAD

```
1   #include <iostream>
2   #include <fstream>
3
4   using namespace std; // Define o namespace std como namespace padrão
5
6   int main(){
7       Exemplo de TAD: arquivos em C++
8       fstream file;
9
10      Os dados de "f" só podem ser acessados pelos métodos de manipulação de arquivos, como:
11          - open()
12          - read()
13          - write()
14          - close()
15
16  Ex:
17          - file.open("arquivo.txt")
18          - file.read(VariavelDeDestino, NumeroDeCaracteresASeremLidos)
19          - file.write(VariavelDeOrigem, NumeroDeCaracteresASeremEscritos)
20          - file.close()
21  }
```

Implementação de uma TAD

- TADs podem ser implementados como classes
- Os conceitos de POO estendem os conceitos de TAD
- Vantagens da TAD?
  - **Encapsulamento** e **Segurança**: Usuário não possui acesso direto aos dados
  - **Flexibilidade** e **Reutilização**: Pode-se alterar um TAD sem alterar as aplica-ções que as utilizam

Complexidade

O que é um algoritmo?

Algoritmo é

Uma **sequência** finita de **instruções**

Bem definidas e não ambíguas

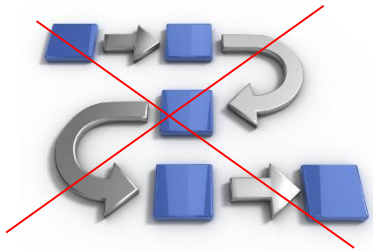
Cada uma das quais devendo ser executadas

Mecânica ou

Eletronicamente

Em um intervalo de tempo finito e

Com uma quantidade de esforço finito



=

Algoritmo

- Um algoritmo corretamente executado **não** resolve uma **tarefa** se:
  - For implementado **incorretamente**
  - **Não** for **apropriado** para a tarefa

- Um algoritmo deve:
  - Funcionar corretamente
  - Executar o mais rápido possível
  - Utilizar a memória da melhor forma possível

Complexidade de um Algoritmo

- Com o intuito de sabermos mais sobre um algoritmo, podemos analisá-lo!
  - Para isso, é preciso estudar as suas especificações e tirar conclusões sobre como a sua implementação (o programa) irá se comportar em geral.

- Entretanto, como podemos analisar um algoritmo?

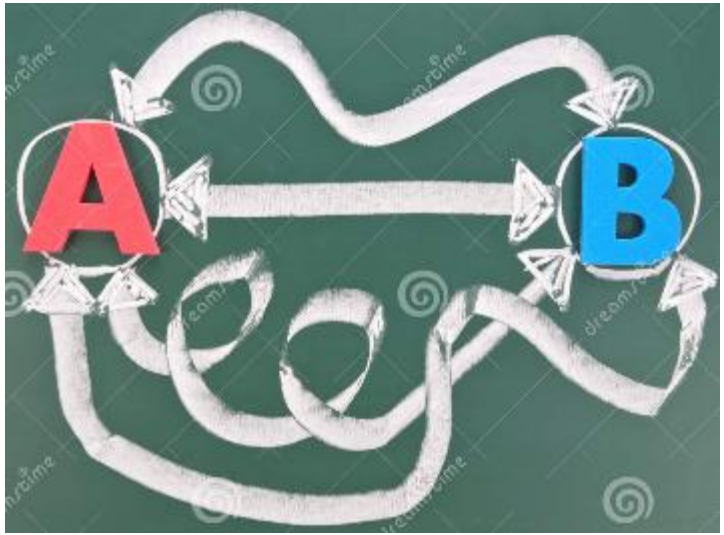
Análise de algoritmos

A **análise de algoritmo** é uma área da ciência da computação que tem como objetivo analisar o comportamento dos algoritmos

- Busca responder a seguinte pergunta:
  - É possível desenvolver um algoritmo mais eficiente?

> “Algoritmos diferentes, mas capazes de resolver o mesmo problema, não necessariamente o fazem com a mesma eficiência.”

  - Diferenças de eficiência entre os algoritmos podem ser: (i) irrelevantes; ou (ii) relevantes, crescem proporcional-mente.



Análise de algoritmos

- Complexidade computacional
  - Medida utilizada para comparar a eficiência entre os algoritmos.
  - Determina uma **função de custo** ao se aplicar um algoritmo.

Custo = Complexidade de **Tempo** + Complexidade de **Espaço**

  - Complexidade de Tempo: Quanto tempo dura a execução do algoritmo.
  - Complexidade Espaço: Quanto de memória o algoritmo utiliza.

- Para determinar se um algoritmo é mais eficiente, pode-se utilizar duas abordagens:
  - **Análise empírica**, comparação entre programas.
  - **Análise matemática**, estudo das propriedades do algoritmo.

Função de custo

Função de custo

- Contar quantas instruções são executadas em um algoritmo.

```
//====Maior valor de um array====
/*1*/  int M = A[0];
/*2*/  for(int i = 0; i < n; i++){
/*3*/      if(A[i] >= M){
/*4*/          M = A[i];
/*5*/  }}
//=========================
=
```

- Quantas **instruções simples** o algoritmo executa?
  - O que é uma instrução simples?
  - Tipos de instruções.

Tipos de instruções:

-Atribuir um valor para uma variável.

-Acessar o valor de uma determinada posição do *array*.

-Comparar dois valores.

-Incrementar um valor.

-Operações aritméticas básicas.

-Inicializar uma variável.

As instruções possuem o mesmo custo de execução.

Função de custo da execução do algoritmo

```
//====Maior valor de um array====
/*1*/ int M = A[0];
/*2*/ for(int i = 0; i < n; i++){
/*3*/     if(A[i] >= M){
/*4*/         M = A[i];
/*5*/ }}
//==========================
=
```

Custo da linha 1 é de **1 instrução**.

Custo de inicializar e realizar no mínimo uma comparação: **2 instruções**.

Custo de incrementar (++) e comparar para a continuação de laço: n instruções + n instruções => **2n instruções**

- Laço vazio, a **função** matemática que representa o **custo** do algoritmo em relação ao tamanho do *array* de entrada:

$$f(n) = 2n + 3$$

Função de custo da execução do algoritmo

- Custo do **if**?

```
//====Maior valor de um array====
/*1*/ int M = A[0];
/*2*/ for(int i = 0; i < n; i++){
/*3*/     if(A[i] >= M){
/*4*/         M = A[i];
/*5*/ }}
//=========================
=
```

$f(n) = 2n + 3$

- Característica do comando de seleção **if**:
  - Array, A1 = [1, 2, 3, 4], sempre verdadeiro.
  - Array, A2 = [4, 3, 2, 1], sempre falso.

Custo da linha 3 no pior/melhor caso é de **n instruções**.

Custo de atribuição no pior caso é **n instruções** e no melhor caso **1 instruções**

- A **função de custo** do algoritmo em relação ao tamanho do *array* de entrada:
  - $f(n) = 2n + 3 + 2n => f(n) = 4n + 3$ (pior caso)

Comportamento assintótico

Pode-se descartar todos os termos que crescem lentamente e manter apenas os que crescem mais rápido à medida que o valor de *n* se torne maior.*

- A **função de custo** do algoritmo:
  - f(*n*) = 4*n* + 3 (pior caso)
- Há dois termos: (i) 4*n*; e (ii) 3.
  - O termo 3 é uma constante de inicialização do algoritmo.
  - Não há alteração à medida que ***n*** aumenta.
- Redução da **função de custo**:
  - f(*n*) = 4*n*
  - f(*n*) = *n*

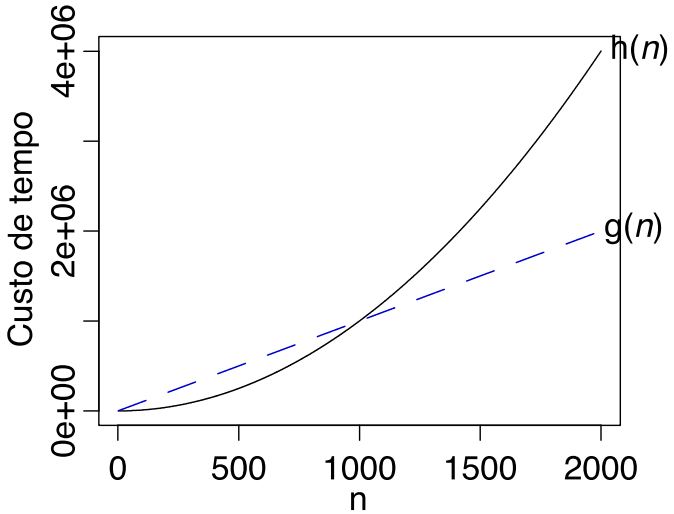
*Manter apenas os termos que nos dizem o que acontece com a função

Comportamento assintótico

- Considere duas funções de custo:
  - g(*n*) = 1000*n* + 500 => ? g(*n*) = *n*
  - h(*n*) = *n*^2 + *n* + 1 => ? h(*n*) = *n*^2

- Apesar da função g(*n*) possuir constantes maiores multiplicando-a, existe um valor de *n* a partir do qual h(*n*) é sempre maior do que g(*n*).
  - Tornando os demais termos e constantes **dispensáveis**.

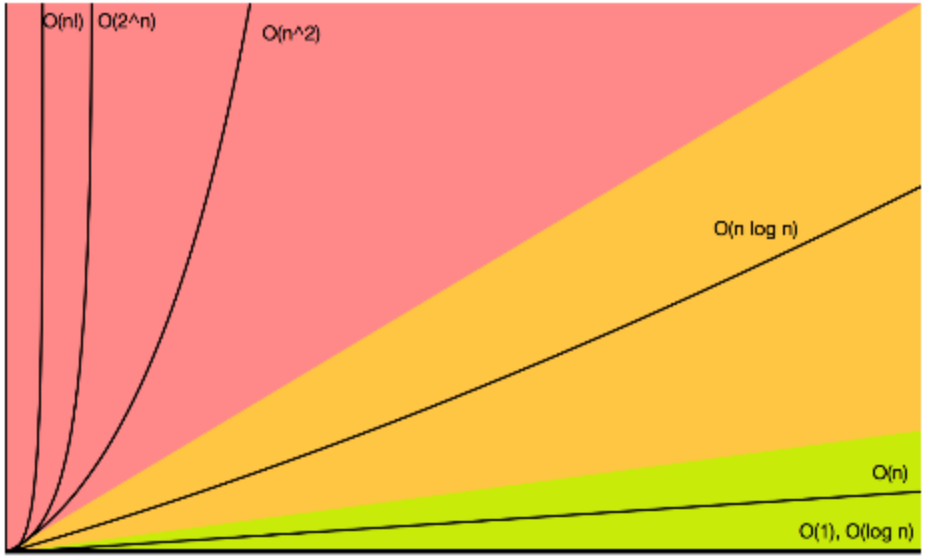
- Para um *n* igual a 2000:



Comportamento assintótico

- Função de custo e comportamento assintótico:

Função de custo	Comportamento assintótico
f(n) = 105	f(n) = 1
f(n) = 15n + 2	f(n) = n
f(n) = n^2 + 5n + 2	f(n) = n^2
f(n) = 5n^3 + 200n^2 + 2	f(n) = n^3
f(n) = log$n$ + 1	f(n) = log$n$
f(n) = $n$log$n$ + n	f(n) = $n$log$n$

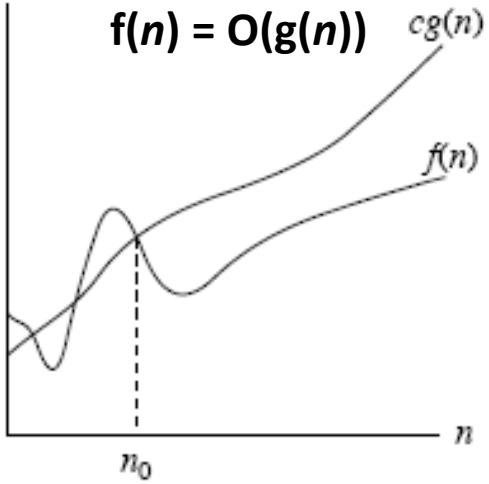


- Tipicamente, pode-se obter a função de custo de um **algoritmos simples** contando os comandos de laços.

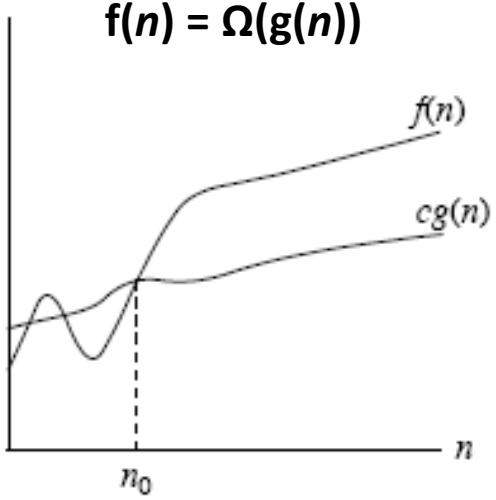
Análise assintótica

- Notação assintótica:
  - Grande-O, O(g(*n*));
  - Grande-Omega, Ω(g(*n*));
  - Grande-Theta, θ(g(*n*));
  - Pequeno-o, o(g(*n*)); e
  - Pequeno-omega, ω(g(*n*)).

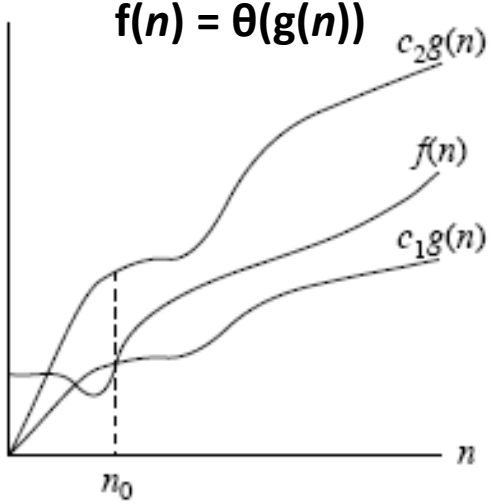
f(*n*) = O(g(*n*))



f(*n*) = Ω(g(*n*))

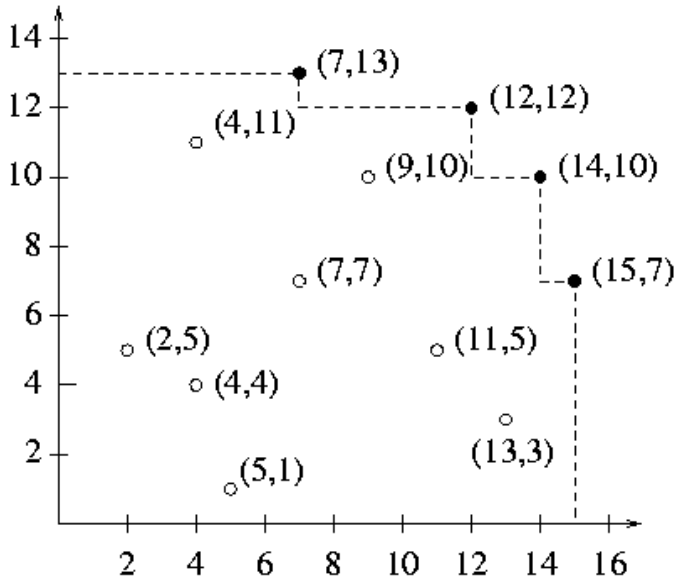


f(*n*) = θ(g(*n*))



Exemplo – Ponto máximo

- Um **ponto máximo** de uma coleção é um que **não é dominado** por nenhum outro (da coleção)
- Diz-se que um ponto (x_1, y_1) domina um ponto (x_2, y_2) se se $x_1 \geq x_2$ e $y_1 \geq y_2$



- Quais são os **pontos máximos** no gráfico acima?
- Determinar a complexidade (assintótica de pior caso) desse algoritmo.

Estruturas de Dados

Tipos Abstratos de Dados (TADs) e Complexidade de Algoritmos