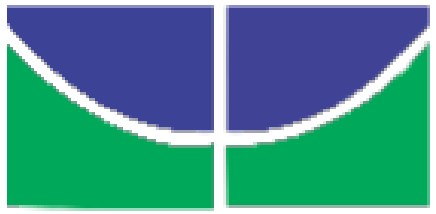


Algoritmos e estrutura de dados

Prof. Wesin Ribeiro

Aula de hoje

- Lógica
- Estrutura de dados
- Tipos abstrato de dados
- Algoritmos
- Complexidade computacional

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?

Qual importância da lógica na construção de algoritmos e estrutura 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 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.
- Qual pergunta você faria?

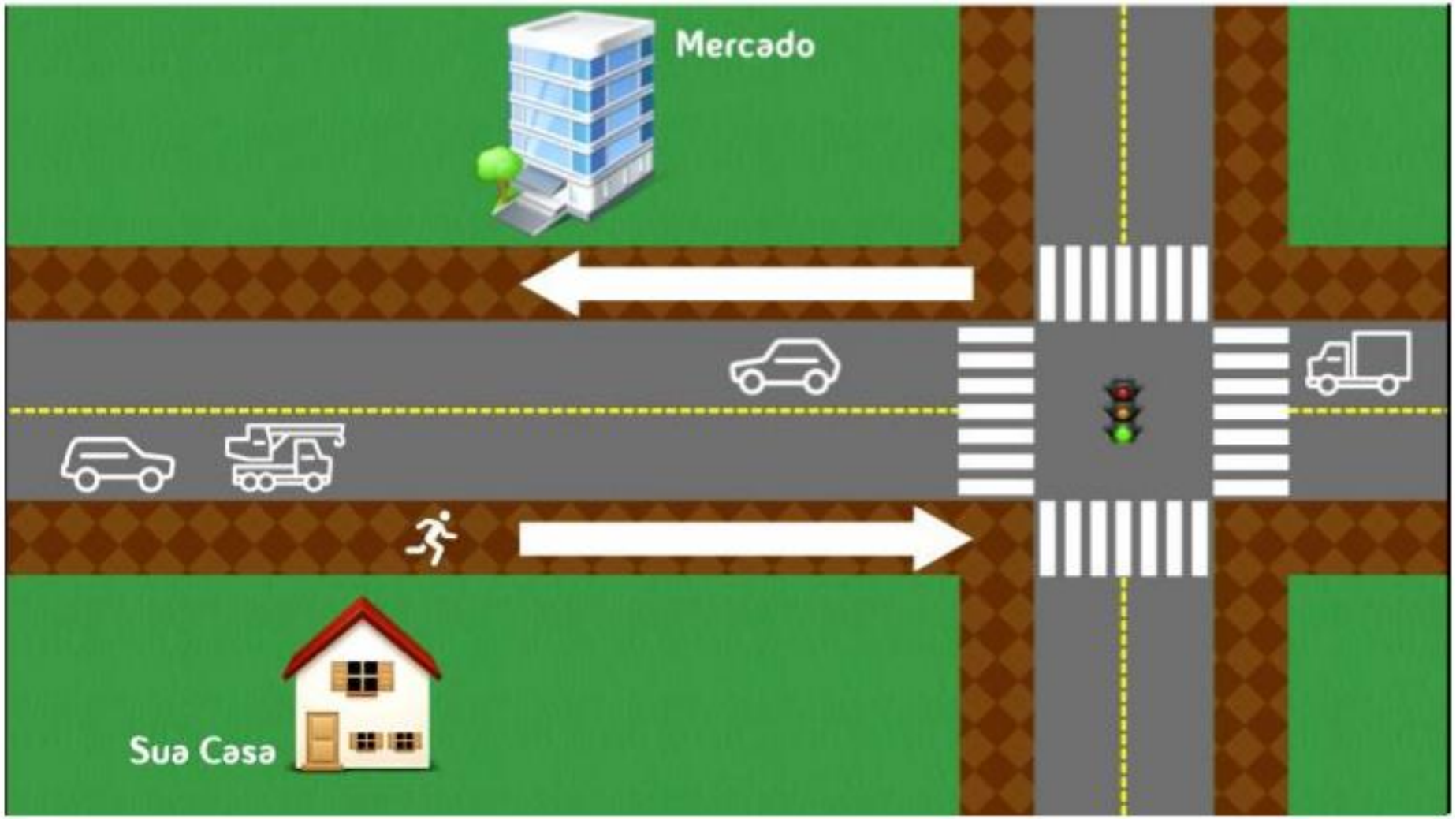
Resposta do teste

- 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

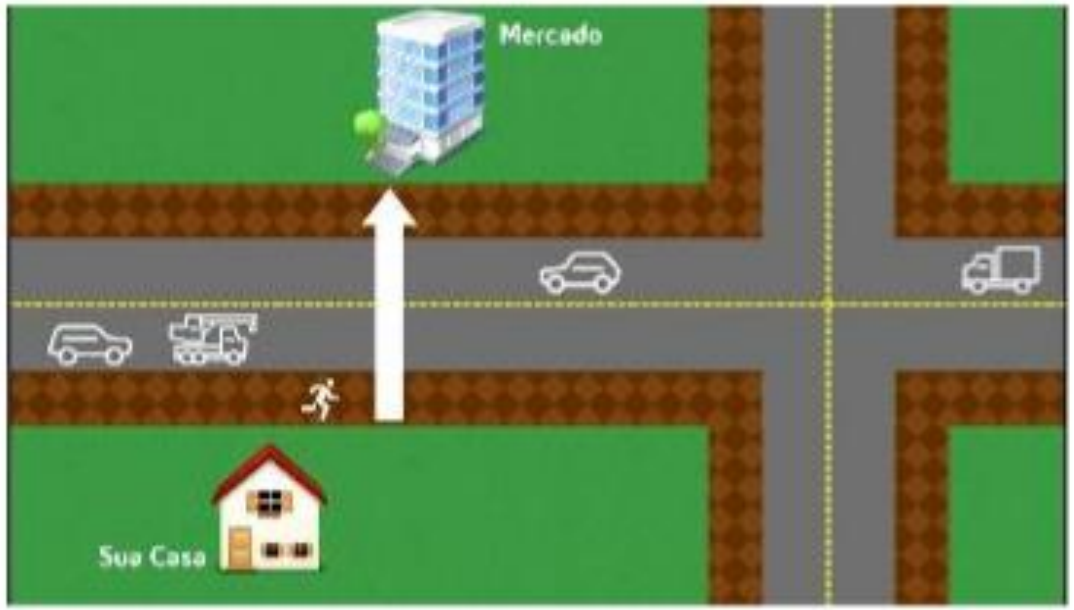
Em algoritmos e estrutura de dados

- 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

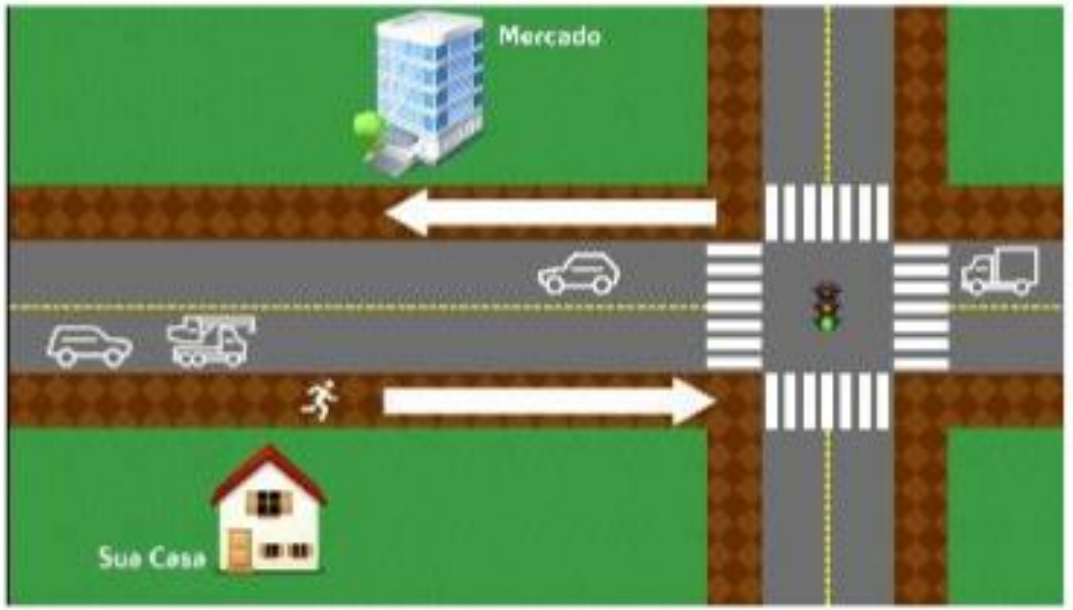
O objetivo foi atingido?



Qual a melhor maneira de atingir o objetivo



Performance

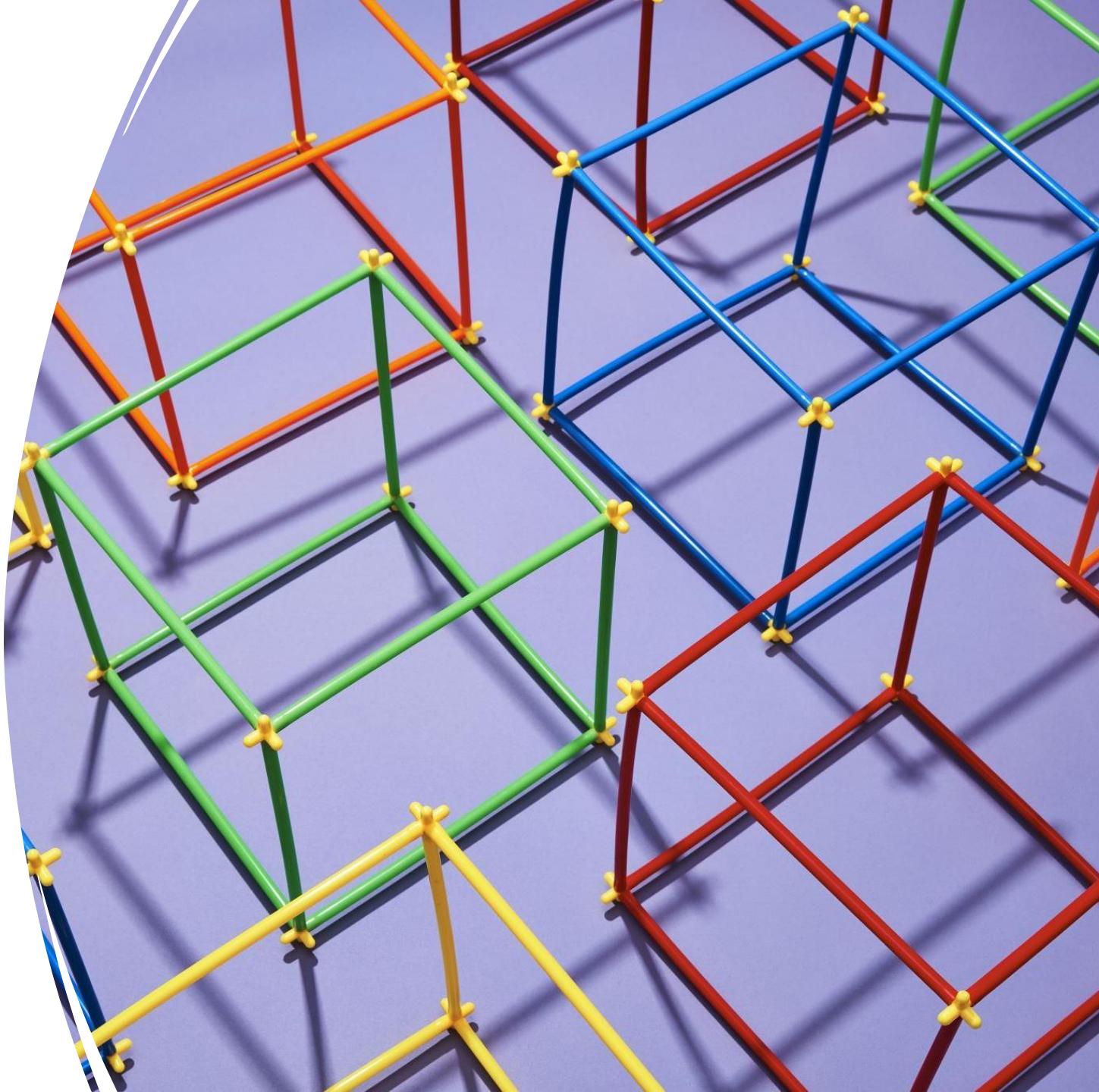


Segurança

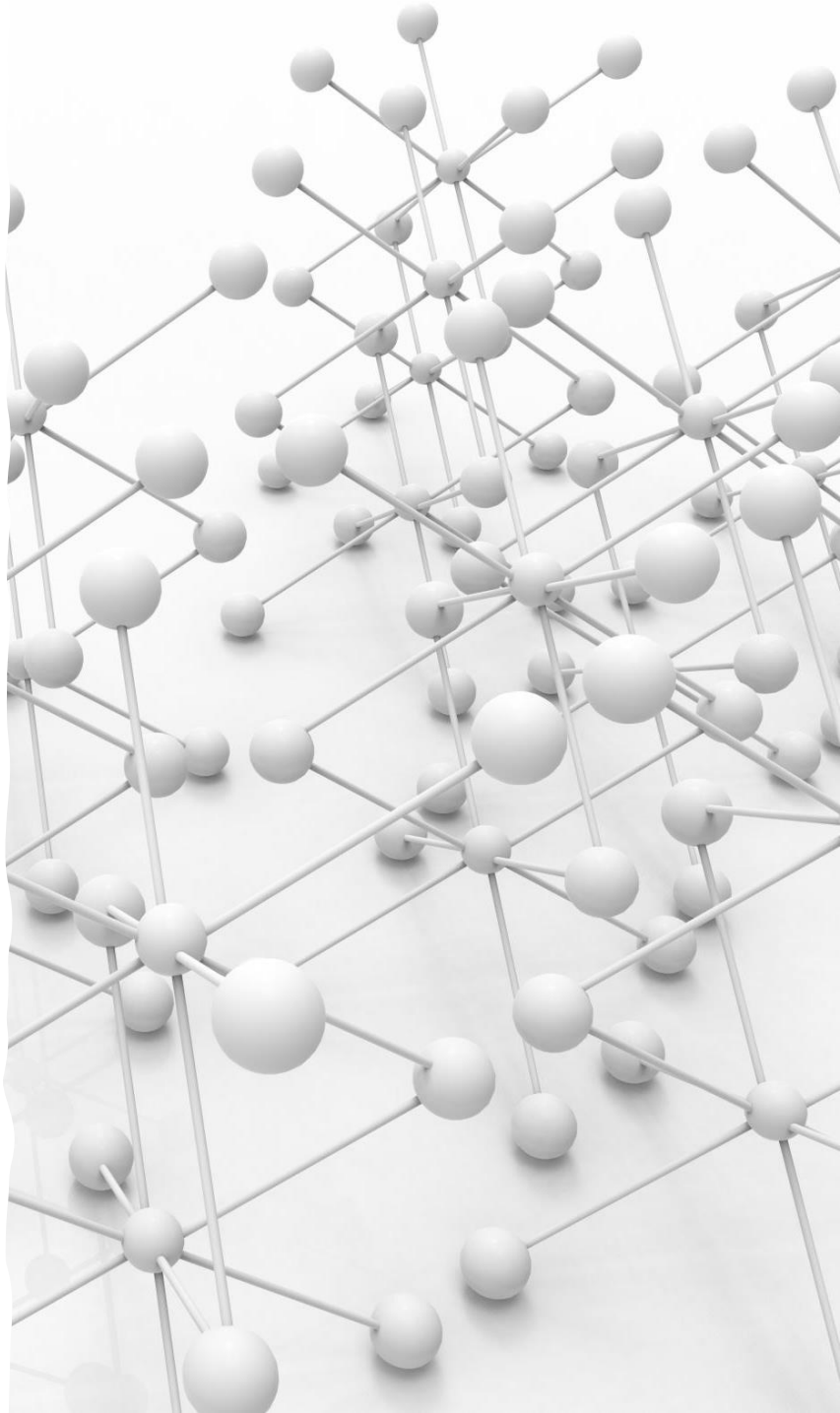
Dependendo do problema, as vezes é necessário abrir mão da performance para obter uma maior segurança, e outras vezes, pode-se abrir mão da segurança para melhorar a performance.

O que são estrutura de dados?

- Diferentes maneiras de organizar dados a fim de solucionar um algoritmo/problema.
- 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.



Principais tipos de estrutura de dados

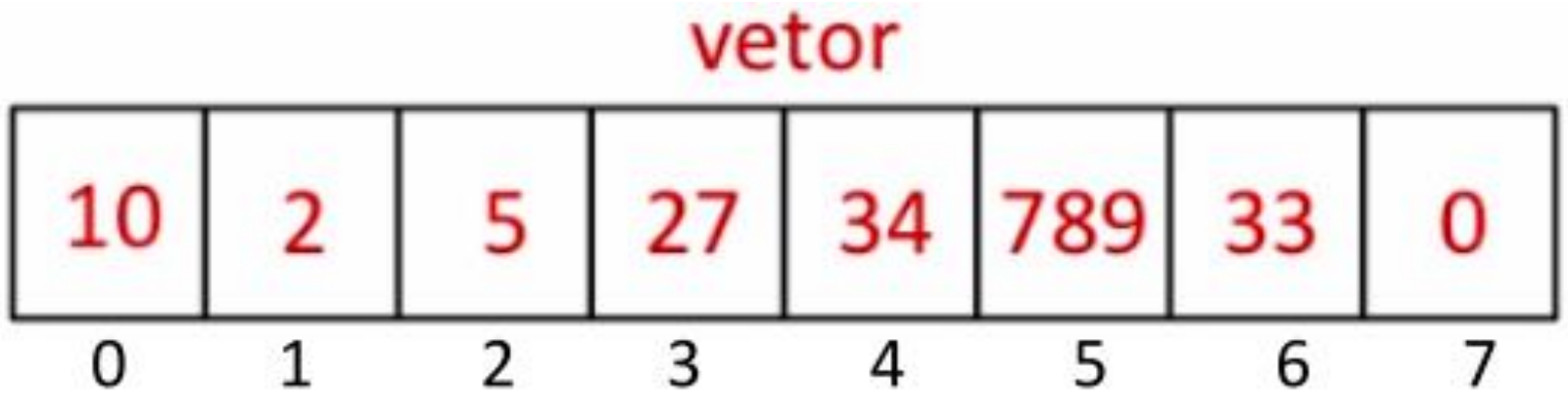


Vetores

- 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

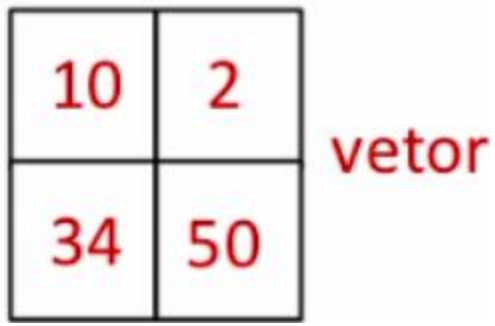
Vetor unidimensional

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



Vetor bidimensional

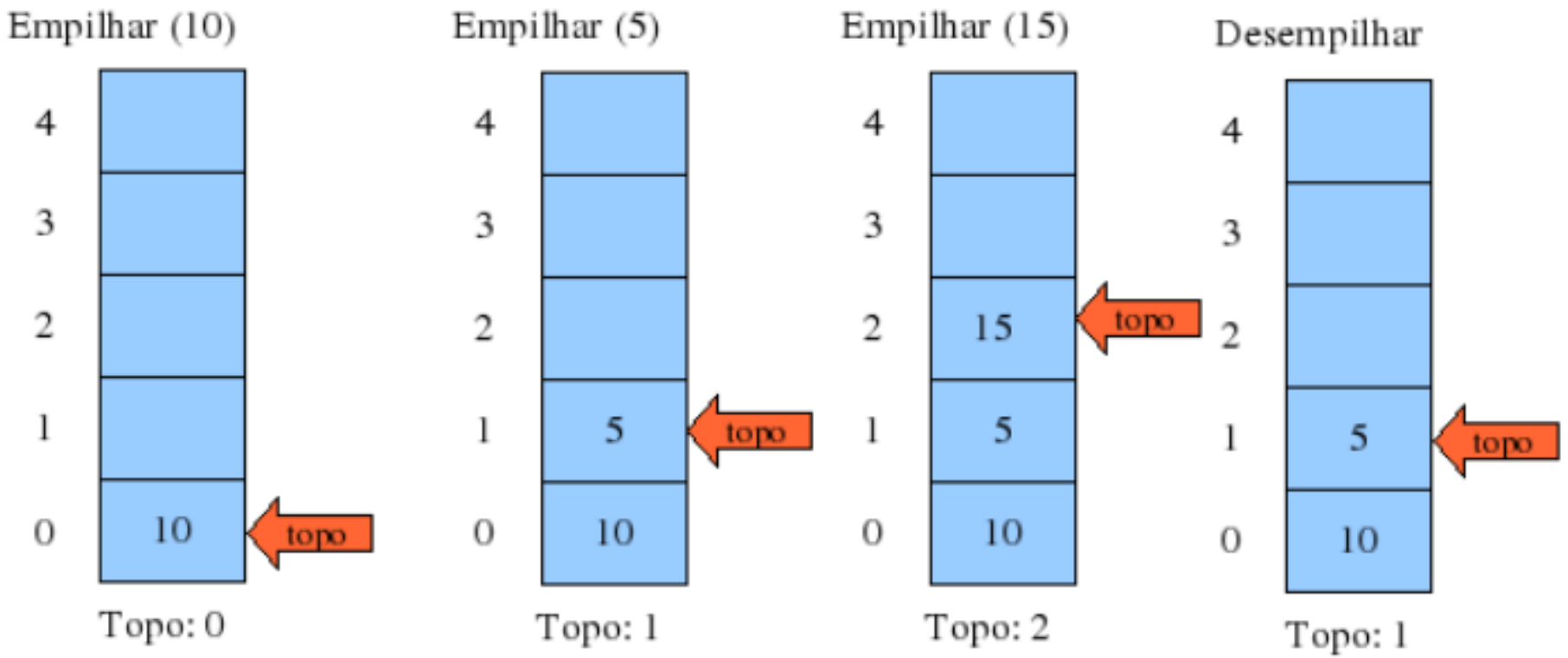
- vetor[0][1] = 2
- Vetor[1][1] = 50
- Qual o tamanho do vetor?



Pilha (stack)

- 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 que está no topo
  - Simula a ideia de pilhas de elementos
  - Para que o acesso a um elemento da pilha ocorra, os demais acima devem ser removidos

Exemplo de funcionamento da Pilha

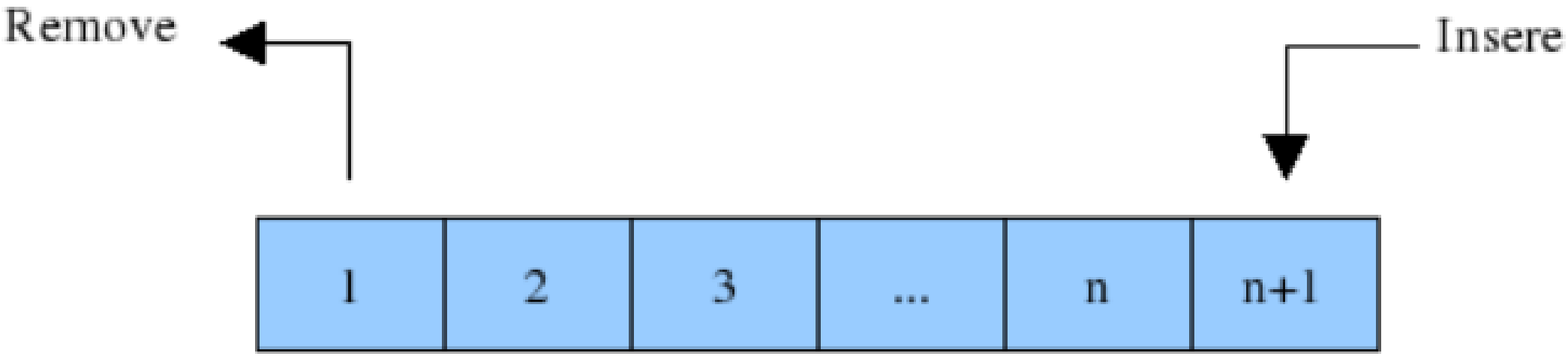


- É possível implementar uma pilha por meio de um vetor?

Fila (Queue)

- É uma estrutura de dados amplamente utilizada e que implementa a ideia de lista de elementos
- Principais caracteristicas:
  - 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

Exemplo de funcionamento da fila



- Qual a diferença entre fila e um vetor?

Outros tipos de estrutura de dados

- LISTAS ENCADEADAS
- ÁRVORES
- GRAFOS
- TABELAS HASH

Exemplo de utilização de estrutura de dados

- **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)

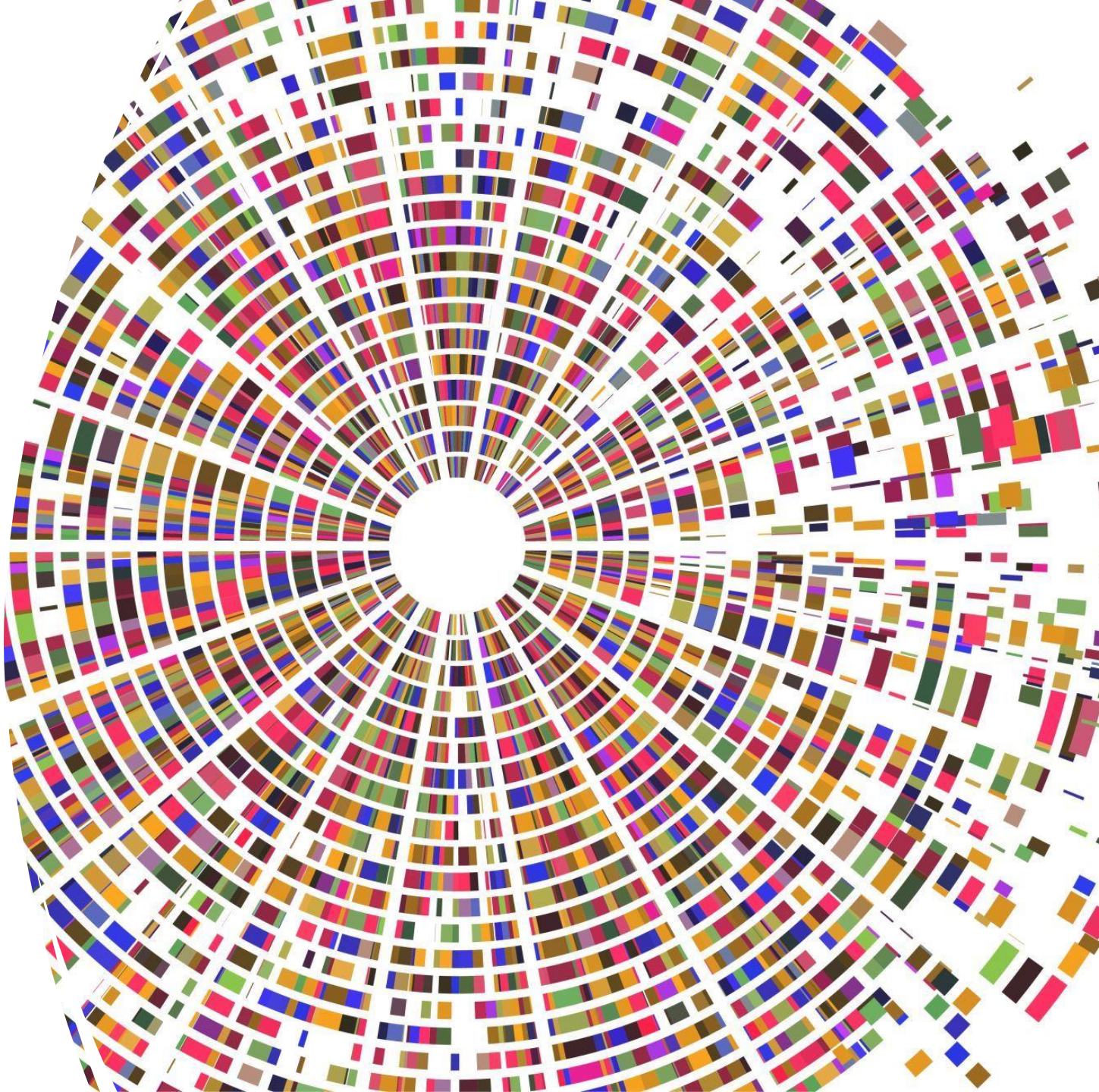
Exemplo de utilização de estrutura de dados

- **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.

Exemplo de utilização de estrutura de dados

- **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
- **Estrutura de dados correspondente**: Árvore - Estrutura de dados que caracteriza uma relação de hierarquia entre os elementos

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 (Pilhas – 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 (Pilhas – armazenada com lista encadeada ou vetor

Tipos de dados

- 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, boolean, etc.
- Tipos estruturados: structs

Tipos abstrato de dados

- 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

- 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

Algoritmos

Qual o papel dos algoritmos na computação?

- Antes de existirem computadores já havia algoritmos
- Um algoritmo é qualquer procedimento computacional bem definido que toma algum valor ou conjunto de valores como entrada e produz algum valor ou conjunto de valores como saída.
- Portanto, um algoritmo é uma sequência de etapas computacionais que transformam a entrada na saída. (Cormen, 2012)

Que tipos de problemas são resolvidos por algoritmos?

O Projeto Genoma Humano - identificar todos os 100.000 genes do DNA humano, determinar as sequências dos três bilhões de pares de bases químicas que constituem o DNA humano, armazenar essas informações em bancos de dados e desenvolver ferramentas para análise de dados.

A Internet - determinação de boas rotas para a transmissão de dados e a utilização de um mecanismo de busca para encontrar rapidamente páginas em que estão determinadas informações.

Comércio eletrônico - Entre as principais tecnologias utilizadas no comércio eletrônico estão a criptografia de chave pública e as assinaturas digitais, ambas baseadas em algoritmos numéricos e na teoria dos números.

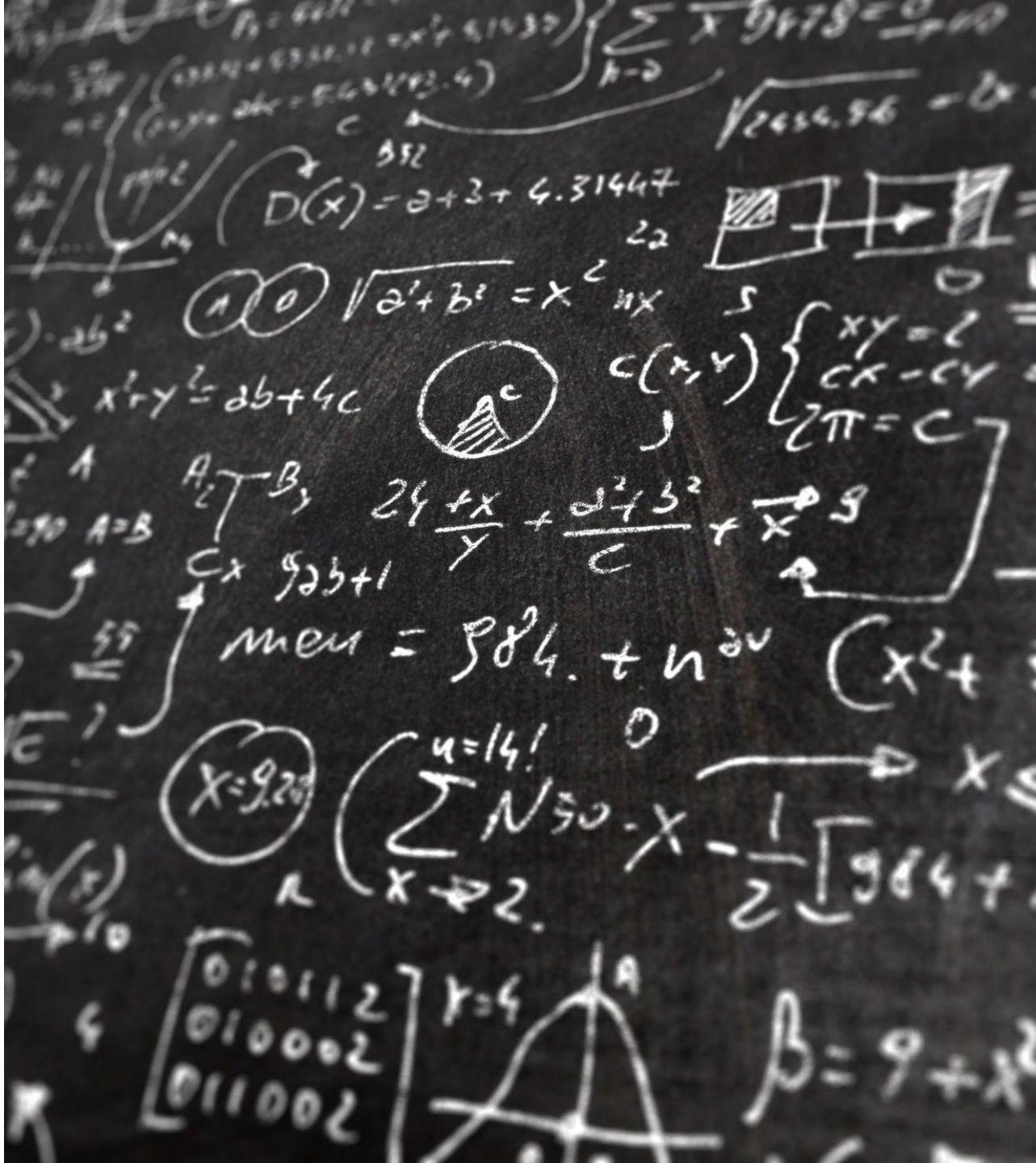
Da maneira errada não adianta fazer o certo

- 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 computacional

Qual a 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

- 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 proporcionalmente.

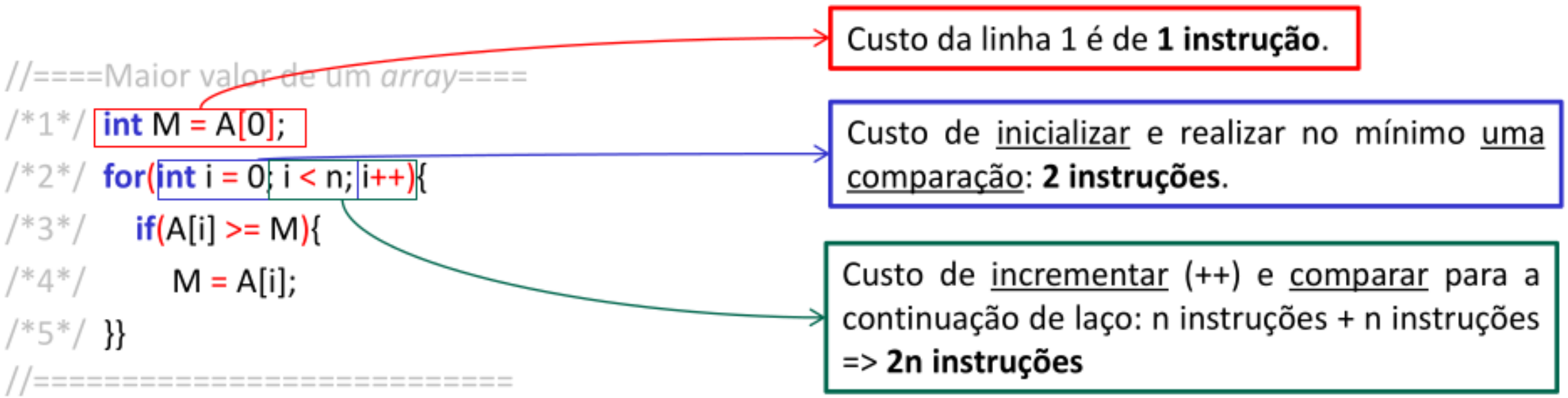
O que é complexidade computacional?

- 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 custo

- Contar quantas instruções são executadas em um algoritmo
- Quantas instruções simples o algoritmo executa?
- O que é uma instrução simples?
  - 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.

Exemplo de função custo



- a função matemática que representa o custo do algoritmo em relação ao tamanho do array de entrada:
- f(n) = 2n + 3

Custo dentro 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**

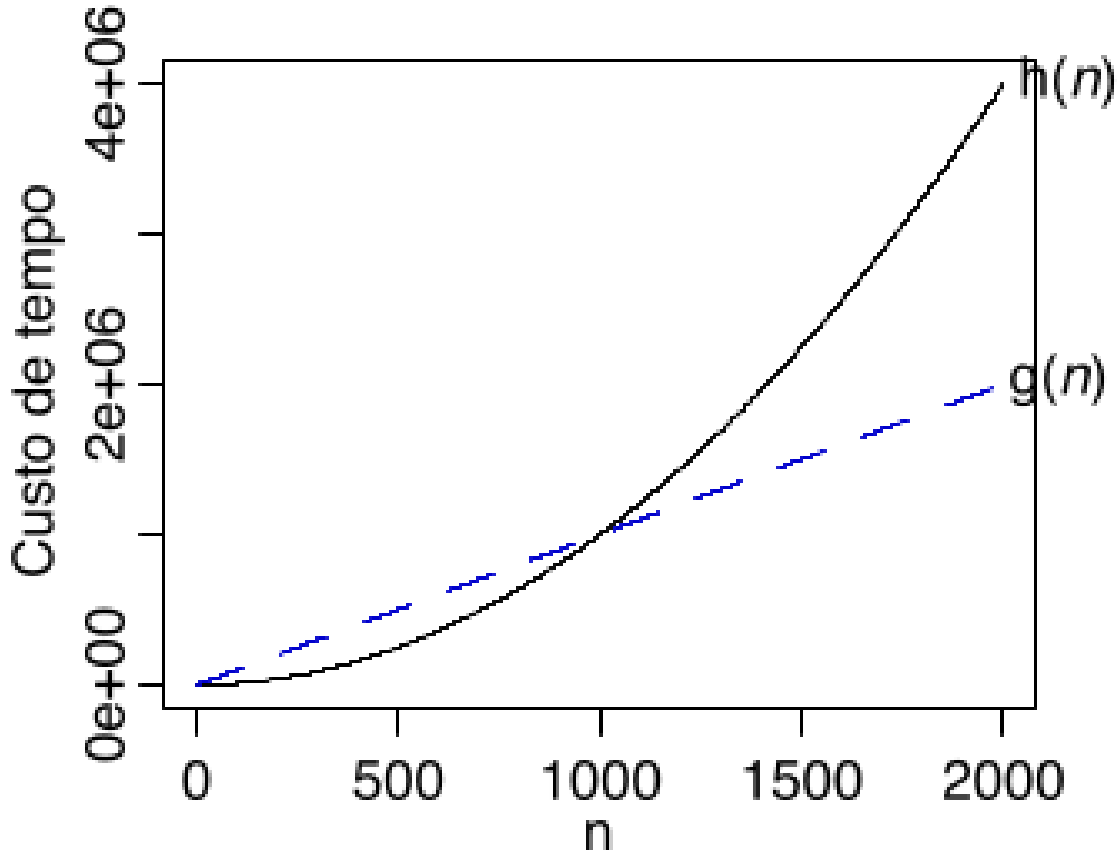
- f(n) = 2n + 3 + 2n => f(n) = 4n + 3

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) = 4n + 3 (pior caso)
- Há dois termos: (i) 4n; (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) = 4n
  - f(n) = n

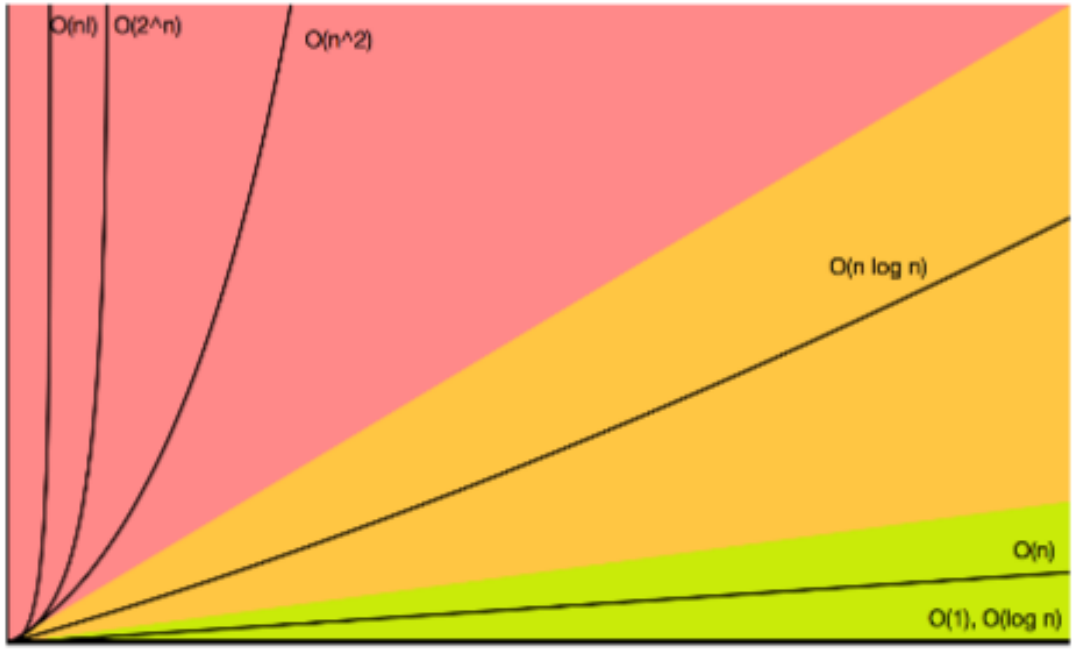
Comportamento assintótico

- Considere duas funções de custo:
  - g(n) = 1000n + 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



Função de custo e comportamento assintótico

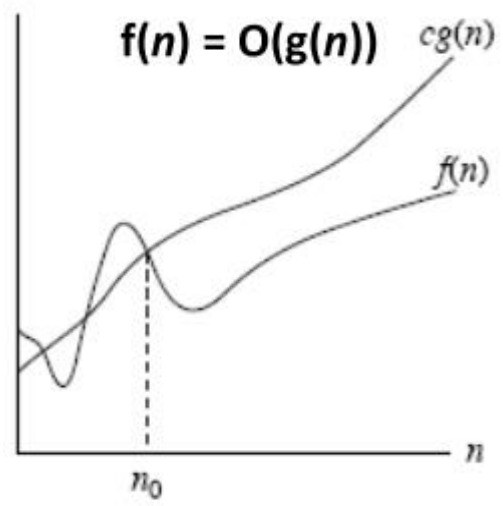
Função de custo	Comportamento assintótico
f(*n*) = 105	f(*n*) = 1
f(*n*) = 15*n* + 2	f(*n*) = *n*
f(*n*) = *n*^2 + 5*n* + 2	f(*n*) = *n*^2
f(*n*) = 5*n*^3 + 200*n*^2 + 2	f(*n*) = *n*^3
f(*n*) = log*n* + 1	f(*n*) = log*n*
f(*n*) = *n*log*n* + *n*	f(*n*) = *n*log*n*

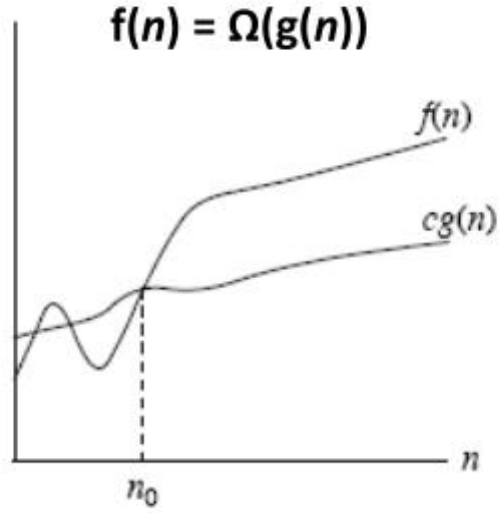


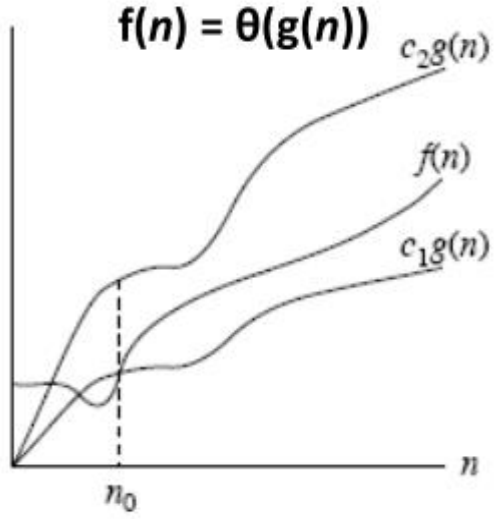
- Tipicamente, pode-se obter a função de custo de um algoritmos simples contando os comandos de laços.

Notação assintótica

- Grande O (limite superior – pior caso)
- Grande omega (limite inferior – melhor caso)
- Grande theta (limite médio - na maioria dos casos)







Exercício – Ponto máximo

- Um ponto máximo de uma coleção é um que não é dominado por nenhum outro (da coleção)
- Diz-se que um ponto (x1, y1) domina um ponto (x2, y2) se se x1 >= x2 e y1>= y2
- Quais são os pontos máximos no gráfico ao lado?
- Determinar a complexidade (assintótica de pior caso) desse algoritmo.