Algoritmos e Estruturas de Dados com Java

Bootcamp: Programador de Software Iniciante

Guilherme Assis

2020

Algoritmos e Estruturas de Dados com Java

Bootcamp: Programador de Software Iniciante

Guilherme Assis

Sumário

Capítulo 1. Estruturas de dados

Neste capítulo, veremos o que são estruturas de dados na computação, assim algumas de suas aplicações. Serão abordadas algumas das estruturas de dados mais utilizadas na programação. São elas: lista, pilha, fila e matriz.

1.1. Introdução às estruturas de dados

Estruturas de dados são formas de organizar as informações dentro de um programa, para que seja mais fácil o processamento. Dependendo do problema e dos dados em questão, uma estrutura pode ser mais indicada que outra para a sua resolução com a melhor performance.

A medida em que o volume de informações a serem processadas por um algoritmo cresce, a importância dessa organização aumenta, pois isso irá impactar diretamente no tempo e na memória gastos com processamento. Podemos dizer que as estruturas de dados facilitam o armazenamento das informações para que elas possam ser processadas de forma eficiente.

Quando um programador vai implementar um algoritmo para resolver determinado problema, ele também precisa pensar em quais estruturas irá utilizar para melhor dispor aquele conjunto de informações, para que seu algoritmo consiga realizar o processamento de forma adequada. O estudo dessas estruturas é importante para que o programador possa criar algoritmos que resolvam de fato o problema, otimizando seu custo de processamento e de armazenamento.

Um dado pode ser de um tipo primitivo, como inteiro, ou pode ser um objeto, instância de uma classe. Algumas operações são comuns à praticamente todas as estruturas de dados, como inserir ou remover um registro na estrutura ou buscar por um determinado registro. De acordo com o propósito da estrutura, também pode ser necessário realizar a ordenação dos elementos, como em ordem alfabética ou numérica.

Um exemplo de utilização de uma estrutura de dados seria para um sistema que precisa organizar uma fila de atendimento aos clientes por ordem de chegada. O primeiro a chegar deve ser o primeiro a ser atendido, e os próximos que forem chegando vão sendo organizados em uma fila. Neste capítulo, veremos algumas das estruturas de dados mais utilizadas na programação: lista, pilha, fila e matriz.

1.2. Lista

A primeira estrutura de dados que veremos é a lista. A lista também é chamada de array, e é amplamente utilizada. Você pode armazenar vários registros em uma lista, o que facilitará quando houver a necessidade de se executar uma mesma ação em todos os registros, pois bastará realizar uma iteração pelos elementos, utilizando para isso alguma estrutura de repetição.

Podemos citar como exemplo de lista uma agenda telefônica. Ela consiste em um conjunto de números de telefones, cada um com os dados desse contato associados. Essa lista pode ou não estar ordenada. No caso da agenda telefônica, faria sentido que os contatos estivessem ordenados por ordem alfabética, facilitando, assim, a localização do registro desejado.

A imagem abaixo ilustra uma lista com 4 elementos: os números 4, 7, 12 e 15. O elemento de índice zero seria o 4, de índice 1 o 7, de índice 2 o 12 e de índice 3 o 4. É sempre importante lembrar que, nas listas, a contagem dos índices se inicia pelo zero. Dessa forma, o último índice será o tamanho da lista menos 1.

Figura 1 – Lista

Nas listas, algumas operações são bem comuns, como inserir ou remover um elemento elemento ou buscar por um registro em específico. No Java, a forma mais comum de se trabalhar com listas é utilizando a classe chamada ArrayList. Nas aulas

gravadas são demonstradas as principais formas de se manipular um ArrayList no Java. Segue abaixo um resumo dos métodos mais utilizados dessa classe:

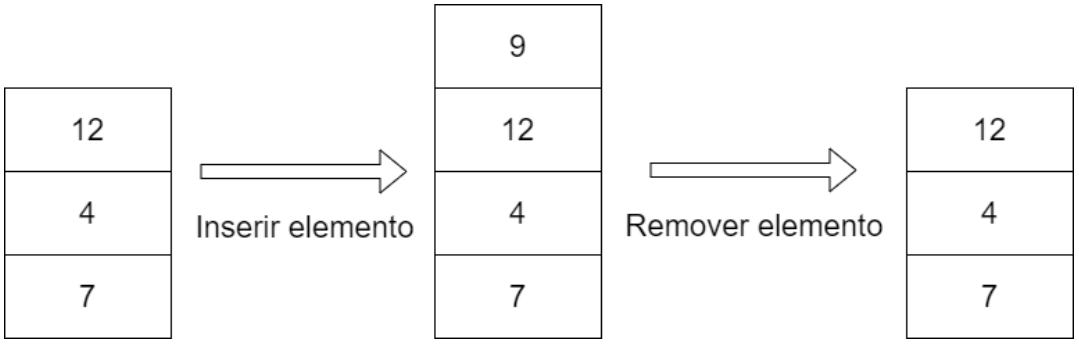
- add(elemento): adiciona um elemento no final da lista.
- add(índice, elemento): insere um elemento na posição indicada.
- remove(índice): remove o elemento da posição indicada.
- clear: remove todos os elementos da lista.
- get(índice): retorna o elemento da lista localizado no índice informado.
- set(índice, elemento): atualiza o elemento localizado no índice indicado, trocando o antigo pelo informado.
- size: retorna o tamanho da lista.
- isEmpty: verifica se a lista está vazia.
- contains(elemento): verifica se o elemento informado existe na lista
- indexOf(elemento): retorna o índice da primeira vez que o elemento informado é encontrado na lista.
- indexOf(elemento): retorna o índice da primeira vez que o elemento informado é encontrado na lista.

1.3. Pilha

Para entendermos a estrutura de dados pilha, podemos fazer uma analogia com uma pilha de livros, pois ela se comporta de forma similar. O primeiro livro inserido fica por baixo, e à medida em que novos livros vão sendo colocados, essa pilha vai aumentando. Ao remover um livro, o primeiro livro a ser retirado, normalmente, será o livro do topo, certo? Dessa forma, o primeiro livro que for colocado na pilha será o último a ser removido, enquanto o último a ser colocado será o primeiro a ser removido.

A estrutura de dados chamada pilha trabalha da mesma forma. Essa forma de trabalhar também é chamada de LIFO (last in, first out), ou seja, o último a entrar será o primeiro a sair. Essa estrutura permite o acesso a apenas um item por vez, sempre o último que foi inserido. A imagem abaixo ilustra essa operação. A pilha inicial continha os elementos 12, 4 e 7, e então um novo elemento foi inserido, o 9, que passou a ocupar o topo. Quando foi solicitado a remoção de um elemento, o próprio 9 foi o removido.

Figura 2 – Pilha



Fonte: Elaborada pelo próprio professor

No Java, a forma mais comum de se trabalhar com pilhas é utilizando a classe Stack. Nas aulas gravadas são demonstradas as principais formas de se manipular uma Stack no Java. Segue abaixo um resumo dos métodos mais utilizados dessa classe:

- push(elemento): insere o elemento no topo da pilha.
- pop: remove o elemento do topo da pilha.
- peek: retorna o elemento do topo da pilha, mas sem removê-lo.
- search: retorna a posição do elemento na pilha, iniciando com 1. Caso o elemento não seja encontrado, retorna -1.

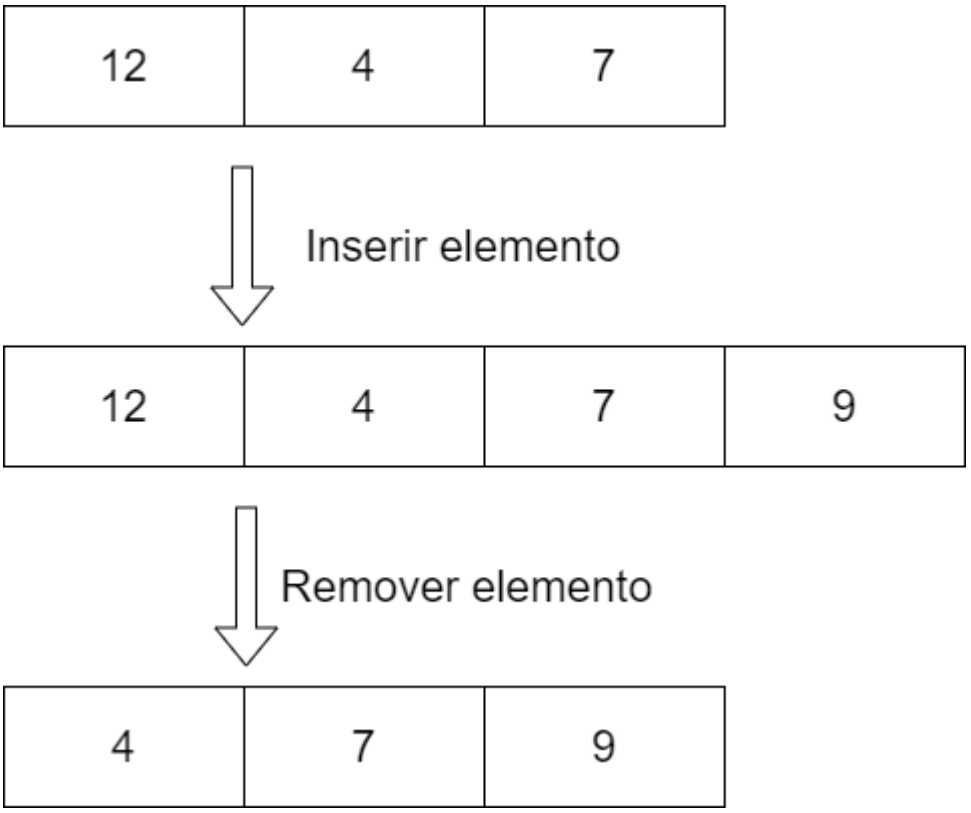
- isEmpty: verifica se a pilha está vazia, retornando true caso esteja e false caso não esteja.

1.4. Fila

Para entendermos a estrutura de dados fila, podemos fazer uma analogia com uma fila de supermercado. O primeiro que entrar na fila será o primeiro a ser atendido, e os clientes que forem chegando entram no fim da fila. A ordem de atendimento segue a ordem de chegada, e último a entrar na fila será o último a ser atendido.

A estrutura de dados fila trabalha da mesma forma. Esta forma também é chamada de FIFO (first in, first out), ou seja, o primeiro a entrar será o primeiro a sair. Essa estrutura permite o acesso a apenas um item por vez, sempre o primeiro que foi inserido. A imagem abaixo ilustra essa operação. A fila inicial continha os elementos 12, 4 e 7, e então um novo elemento foi inserido, o 9, que passou a ocupar o fim da fila. Quando foi solicitada a remoção de um elemento, foi removido o primeiro elemento da fila, o 12.

Figura 3 – Fila



No Java, a forma mais comum de se trabalhar com filas é utilizando a Queue e LinkedList. Nas aulas gravadas são demonstradas as principais formas de se manipular uma Queue no Java. Segue abaixo um resumo dos métodos mais utilizados dessa classe:

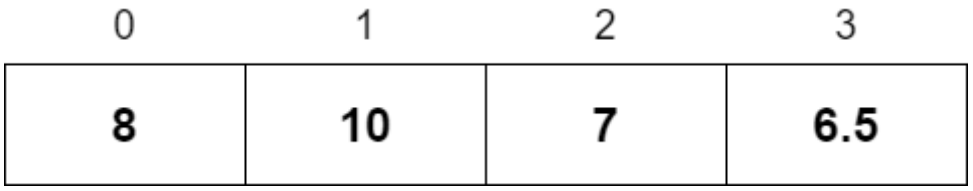
- offer(elemento): insere o elemento na fila.
- poll: remove o primeiro elemento da fila.
- peek: retorna o primeiro elemento da fila, mas sem removê-lo.
- size: retorna o tamanho da fila.
- isEmpty: verifica se a fila está vazia, retornando true caso esteja e false caso não esteja.

1.5. Matriz

As estruturas que vimos até aqui possuem apenas uma dimensão. Em outras palavras, possuem apenas uma linha. Caso haja a necessidade de se representar mais de uma dimensão, você pode começar a trabalhar com matriz. Podemos pensar em uma matriz como se fosse uma tabela, que possui linhas e colunas.

Vamos supor que você precisa de armazenar as notas de um aluno em um determinada disciplina, como matemática. Uma solução seria criar um array e colocar nele as notas de todas as atividades dessa disciplina. A organização ficaria de forma parecida com a imagem abaixo:

Figura 4 – Array



O array da imagem possui as notas desse aluno na disciplina de matemática. A primeira nota foi 8, a segunda 10, a terceira 7 e a quarta 6.5. Os números acima das notas representam o índice do array, iniciado em zero.

Caso fosse necessário armazenar as notas de outras disciplinas, como isso poderia ser feito? Uma solução seria criar um array parecido com o dessa imagem para cada disciplina. No entanto, se você não soubesse quantas disciplinas são, ou se elas pudessem variar de acordo com a entrada do programa, ficaria difícil de organizar essas informações. Uma solução para isso seria utilizar uma matriz como a da imagem abaixo:

Figura 5 – Matriz

	0	1	2	3
0	**8**	**10**	**7**	**6.5**
1	**7**	**9**	**5**	**4**
2	**4**	**8**	**6**	**9**

A imagem acima representa uma matriz. Uma matriz nada mais é do que uma estrutura capaz de armazenar linhas e colunas, como se fosse uma tabela. Podemos implementá-la utilizando um “array de arrays”. Nas aulas gravadas serão demonstrados exemplos que facilitarão o entendimento da parte de código.

Na imagem podemos ver que, além dos números acima da imagem, que representam os índices da coluna, temos agora números à esquerda, que representam os índices das linhas. Com essa estrutura podemos armazenar as notas do aluno em várias disciplinas, fazendo com que cada linha represente as notas de uma disciplina, e com que cada coluna represente as notas em determinada atividade. A primeira linha poderia representar a disciplina de matemática, enquanto a linha 1 de português e a linha 2 de geografia, por exemplo. Se você quisesse verificar a nota do aluno na terceira atividade de português, você poderia consultar nessa matriz a linha de índice 1 e coluna de índice 2.

Capítulo 2. Programação Orientada a Objetos

Neste capítulo faremos uma breve recapitulação do conceito de programação orientada a objetos, que foi visto no módulo anterior. Em seguida, avançaremos um pouco mais nesse tema, abordando os conceitos de herança, interface e polimorfismo.

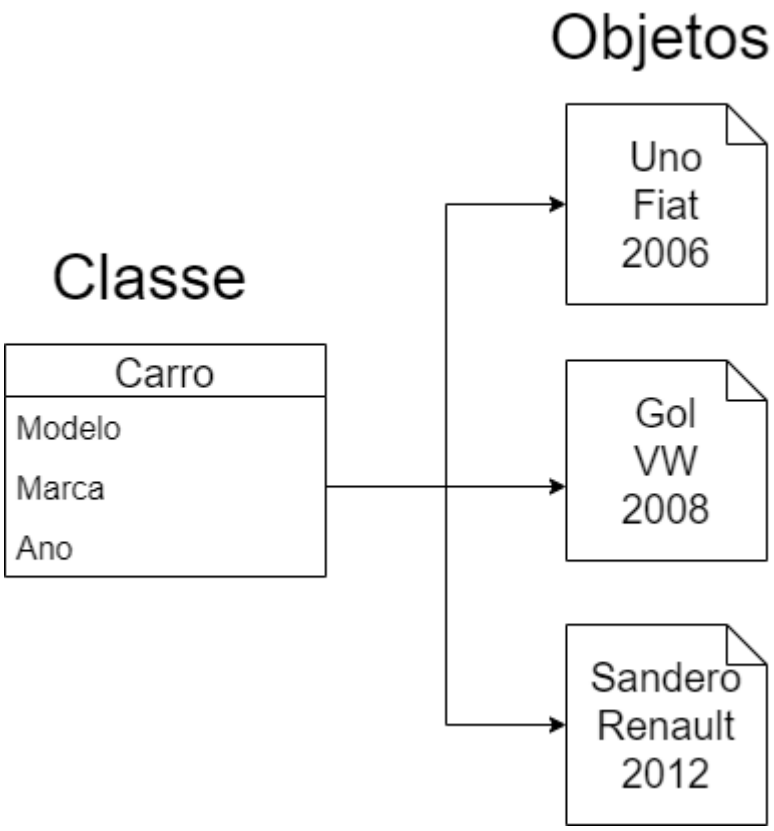
2.1. Recapitulação

A programação orientada a objetos é um paradigma de programação muito utilizado no mercado, que busca abstrair coisas do mundo real a partir de classes e de objetos.

Classe é uma forma de representar um conjunto de informações que dizem respeito a uma mesma entidade. Por exemplo, se criássemos uma classe para representar um carro, ela poderia ter características como modelo, marca e ano. A partir da definição dessa classe, seria possível criar cada carro com suas características específicas. Poderíamos ter uma instância da classe carro com características como modelo Uno, marca Fiat e ano 2006. Essa instância do carro, com suas características específicas, é o que chamamos de objeto.

A imagem abaixo ilustra essa definição de classes e objetos. Temos no lado esquerdo uma representação de uma classe chamada Carro, que possui as propriedades (também chamadas de atributos) modelo, marca e ano. No lado direito, temos três objetos dessa classe, cada um com suas particularidades.

Figura 6 – Classes e objetos



Uma classe também pode ter métodos associados. Um método é como se fosse uma função que pode executar um código definido, podendo acessar e manipular os atributos do objeto do qual ele está sendo acessado. Dessa forma, os métodos podem definir comportamentos para os objetos de uma classe. Por exemplo, se a classe Carro tivesse um método chamado “acelerar”, esse método poderia modificar o atributo “velocidade” dos objetos.

Os atributos de uma classe podem ter modificadores de acesso, como públicos e privados. Quando são públicos, podem ser acessados por objetos até mesmo de outras classes, enquanto quando são privados, só pode ser acessados internamente por seus métodos.

Outro conceito importante é o de encapsulamento. Ele trabalha diretamente com o conceito de atributos privados. O programador pode optar por quais

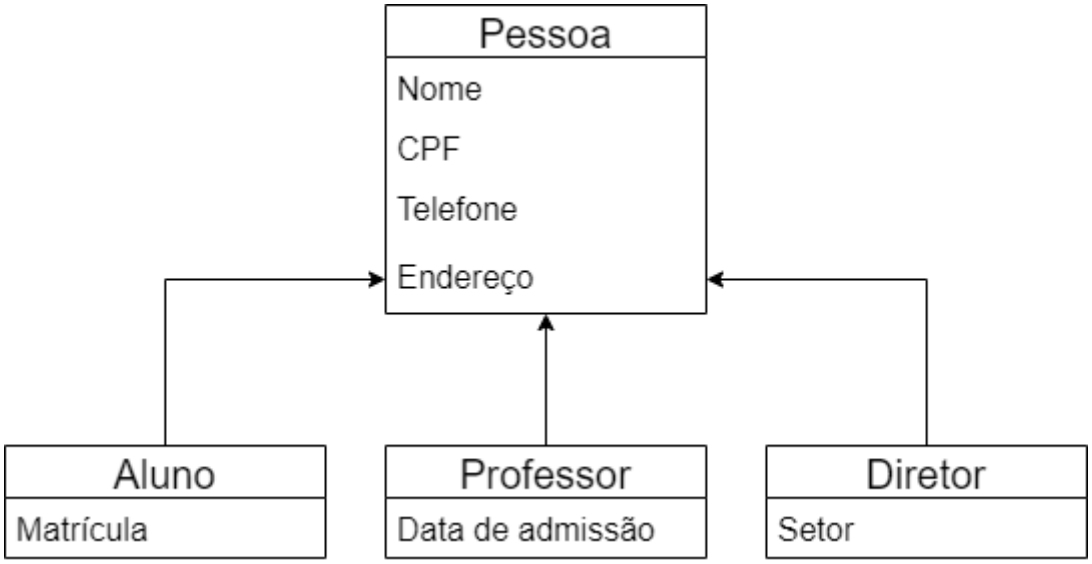
informações ele deseja que sejam possíveis de se acessar fora da classe, e encapsular a alteração das demais a partir de métodos. No exemplo do carro, uma opção seria transformar o atributo “velocidade” em privado, evitando assim que ele seja alterado de forma equivocada, encapsulando sua alteração somente nos métodos de “acelerar” e “desacelerar”. Assim, as alterações estariam sob controle, sendo realizado somente o que foi especificado no momento da definição da classe.

2.2. Herança

Herança é um dos conceitos da programação orientada a objetos. A partir dessa característica, é possível que uma classe herde atributos e métodos de outra. Por exemplo, suponhamos que quiséssemos representar as classes integrantes de uma escola, como professores, diretores e alunos. Ao começar a definir os atributos de cada uma dessas classes, você notará que muitos são comuns entre elas.

Por exemplo, precisamos definir nome, CPF, telefone e endereço para todos os integrantes. Ao invés de replicar esses atributos em todas as três classes, uma outra solução seria criar uma classe chamada Pessoa que tivesse esses atributos em comum. Assim, as demais classes poderiam herdar essas informações, para depois serem definidas suas particularidades. A imagem abaixo ilustra essa situação. As classes Aluno, Professor e Diretor herdam da classe Pessoa suas características, e mesmo assim elas podem implementar suas especificidades sem problemas. No Java, isso é feito com a palavra-chave “extends”. Então, no exemplo abaixo, a criação da classe Aluno seria feita da seguinte forma: class Aluno extends Pessoa.

Figura 7 – Herança



Uma outra característica importante da herança é a chamada classe abstrata. A principal característica de uma classe abstrata é que não é possível instanciar diretamente um de seus objetos. Ela atua como se fosse um rascunho de uma classe, pois você não consegue criar um objeto diretamente a partir dela.

No exemplo acima, poderíamos definir a classe Pessoa como abstrata, já que pode não fazer muito sentido para a solução do problema um objeto Pessoa, e sim um objeto Aluno, Professor ou Diretor. No Java, uma classe abstrata é criada utilizando a palavra-chave “abstract”. Então, a criação da classe Pessoa seria da seguinte forma: abstract class Pessoa. Uma observação importante é que uma classe pode herdar classes abstratas ou não abstratas normalmente.

Em uma classe abstrata, alguns de seus métodos também podem ser “abstract”. Isso quer dizer que eles não precisam fornecer a implementação do método, somente a declaração. A classe que herdar dessa classe abstrata será obrigada a implementar todos os métodos abstratos.

Uma outra possibilidade que existe quando estamos trabalhando com herança é a utilização da palavra-chave “final”. Definir uma classe como final impedirá que outras classes possam herdar dela através do uso do “extends”. Então, no exemplo mencionado anteriormente, caso colocássemos um “final” na declaração da classe Aluno, não seria possível que uma outra classe herdasse dela, pois estamos dizendo que aquelas definições são definitivas, e ninguém pode reaproveita-las através da herança.

Nas aulas gravadas veremos exemplos práticos utilizando esses conceitos de herança.

2.3. Interface

Na programação orientada a objetos, uma interface é vista como um contrato a ser seguido. Ela é composta somente pela declaração de alguns métodos, e as classes que implementarem essa interface são obrigadas a fornecer a implementação desses métodos. Os métodos são não podem ser implementados diretamente na interface, somente nas classes que a implementarem.

Por exemplo, poderíamos definir uma interface chamada Carro, e colocar dentro dela os métodos que todo carro deve ter, como “acelerar” e “frear”. Para caracterizar uma classe como Carro, você poderia implementar essa interface chamada Carro. Então, ao criar uma classe chamada Gol, essa classe poderia ter na sua definição a seguinte expressão: “class Gol implements Carro”. Ao fazer isso, a classe Carro implementaria os métodos “acelerar” e “frear”.

No Java, uma classe pode implementar várias interfaces. Então, a classe Gol poderia implementar a interface Carro e uma outra interface, como MeioLocomocao.

A partir do Java 8, foi criada a possibilidade de se definir implementações default para um método dentro da interface. Dessa forma, caso o desenvolvedor esteja implementando uma interface em alguma classes, ele tem a opção de não fornecer uma implementação para os métodos já definidos como default. Caso ele

opte por fornecer essa implementação, não há problema: ela irá sobrescrever a default.

2.4. Polimorfismo

Polimorfismo é um termo muito utilizado na programação orientada a objetos, e diz respeito à possibilidade de um objeto se comportar de formas diferentes de acordo com a situação. No polimorfismo, duas características estão muito presentes: a sobrescrita e a sobrecarga de métodos.

Quando uma classe herda uma outra, ela herda seus atributos e seus métodos. Nesse momento, essa classe tem a opção de realizar uma sobrescrita de alguns métodos, substituindo o código herdado pelo que o programador definir. Para que isso ocorra, o novo método precisa ter o mesmo nome, tipo de retorno e parâmetros. Dessa forma, a classe filha estaria fornecendo uma nova implementação para o método, e não um novo método.

A sobrecarga ocorre quando a classe filha optar por criar um novo método com o mesmo nome da que foi herdada, variando somente a quantidade de parâmetros ou o tipo de alguns deles. Então, apesar de o método ter o mesmo nome do que estava presente na classe herdada, sua assinatura é diferente, e o método que será executado ao ser chamado dependerá dos parâmetros enviados. Um exemplo de utilização frequente da sobrecarga é nos construtores das classes no Java, pois dessa forma você pode inicializar um objeto de formas diferentes de acordo com os parâmetros fornecidos.

Capítulo 3. Algoritmos

Neste capítulo estudaremos alguns algoritmos clássicos da computação. Iniciaremos vendo alguns algoritmos de ordenação e depois veremos os algoritmos de busca linear e binária, utilizados para encontrar elementos em uma lista.

3.1. Algoritmos de ordenação

Algoritmos de ordenação são um conjunto de algoritmos que tem por objetivo ordenar um conjunto de elementos. Existem vários algoritmos que possuem esse fim. Veremos, aqui, dois deles: o algoritmo da bolha e o algoritmo de seleção.

Apesar da maioria das linguagens de programação já fornecer métodos prontos para se realizar a ordenação de um conjunto, o estudo desses algoritmos é importante para que o programador exercite sua lógica, e entenda o que está sendo feito internamente nessas funções já prontas das linguagens.

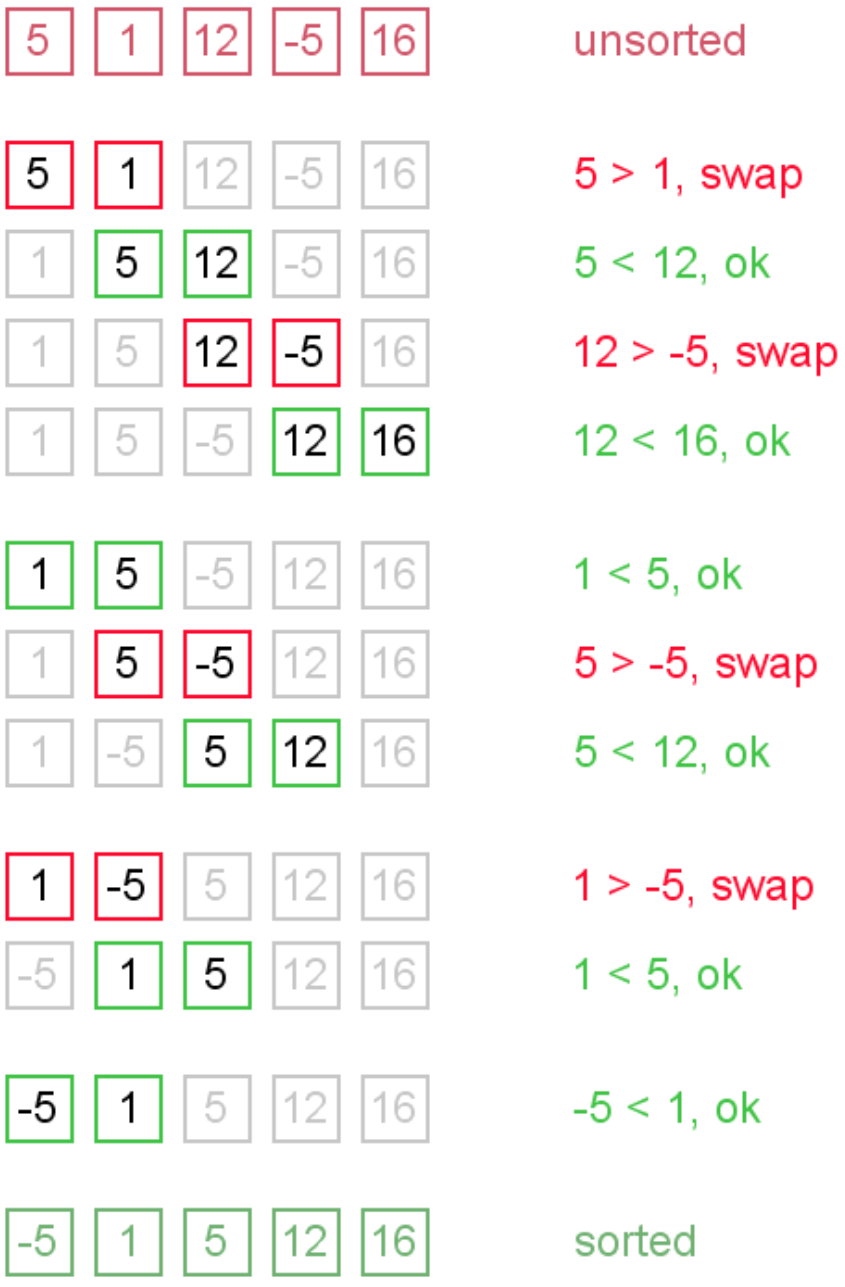
Os elementos de um conjunto que serão ordenados podem ser de qualquer tipo, desde que seja possível realizar uma comparação entre eles. Para elementos como números, fica fácil entender, pois é bem claro quando um número é maior do que outro. Para strings, as linguagens geralmente utilizam os bytes de cada caractere, comparando-os internamente a partir do código Unicode daquele caractere. Já para outros elementos, como um objeto de uma classe, cabe ao programador definir os critérios necessários para que um objeto seja comparado com outro.

3.2. Algoritmo da bolha

O algoritmo da bolha é um dos mais simples que existe. De forma resumida, ele consiste em percorrer o conjunto de elementos, sempre comparando dois elementos de cada vez e verificando se eles estão ordenados entre si. Caso não estejam, é feita a troca de posição entre eles. O algoritmo segue até o fim do conjunto, depois faz o caminho inverso, até que todo o conjunto esteja ordenado.

A imagem abaixo ilustra esse procedimento, iniciando com a lista desordenada e, a cada iteração, fazendo as comparações dos dois elementos em questão.

Figura 8 – Algoritmo da bolha



Fonte: https://www.algolist.net/img/sorts/bubble-sort-1.png

3.3. Ordenação por seleção

O algoritmo de ordenação por seleção adota a abordagem de selecionar o menor elemento do conjunto e trocá-lo de lugar com o que estiver na primeira posição. Depois, ele repete esse mesmo processo com o restante dos elementos, não mexendo mais no elemento que já está ordenado.

Dessa forma, esse algoritmo coloca cada item no seu devido lugar à cada iteração. Inicialmente, o primeiro elemento é colocado em sua posição, depois o segundo, o terceiro, até que se esgotem os elementos e, por consequência, o conjunto esteja ordenado.

A figura abaixo ilustra esse processo, selecionando a cada etapa o elemento de menor valor e colocando-o em sua posição correta.

Figura 9 – Ordenação por seleção

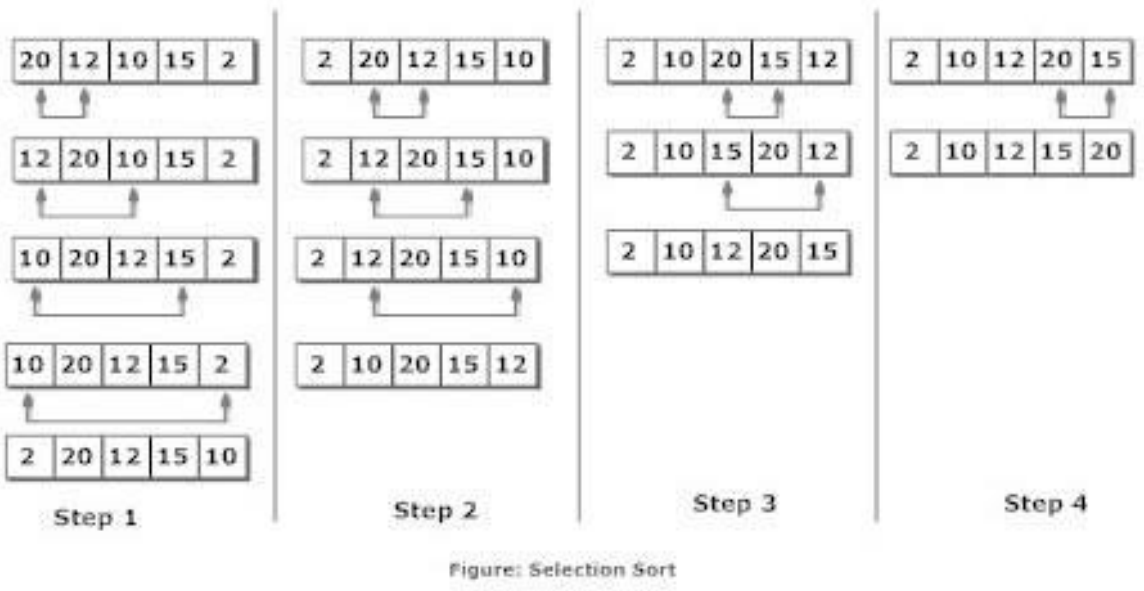


Figure: Selection Sort

Fonte: https://medium.com/@paulsoham/selection-sort-df6c93d9da3d

3.4. Busca linear

A busca linear, também chamada de busca sequencial, é um algoritmo simples, que pode ser utilizado para encontrar determinado elemento em uma lista. O algoritmo realiza um loop pelos elementos da lista e, a cada elemento, verifica se o elemento em questão é igual ao que está sendo procurado. Caso seja, o elemento é retornado e a execução é finalizada.

A medida em que o tamanho da lista aumenta, o custo de processamento desse algoritmo aumenta na mesma medida, já que ele precisará verificar mais elementos. A melhor das hipóteses é quando o elemento buscado está na primeira posição, pois assim ele precisará verificar apenas um elemento. A pior das hipóteses é quando o elemento buscado está na última posição, pois assim o algoritmo precisará passar por todos elementos até encontrar o desejado.

3.5. Busca binária

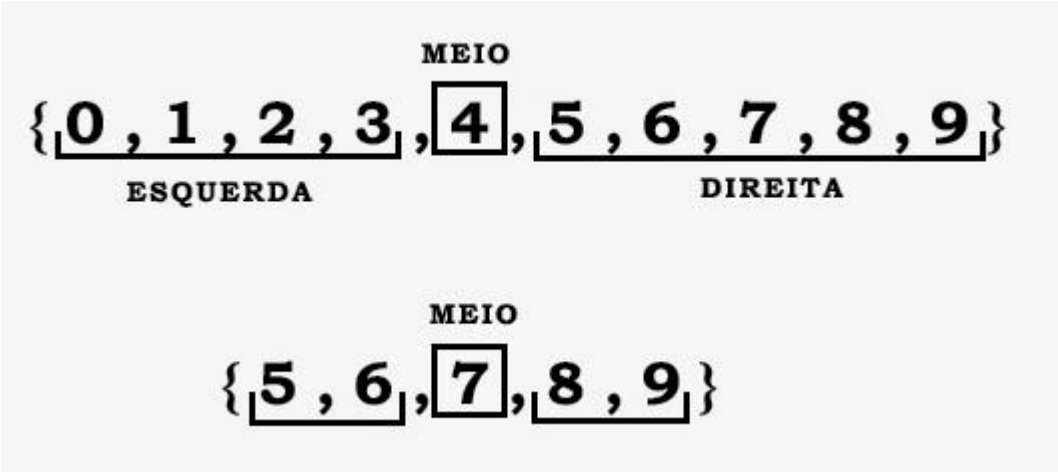
A busca binária, também chamada de pesquisa binária, é um algoritmo de busca em listas que utiliza a ideia da divisão e conquista. Para ele funcionar, é preciso que a lista esteja previamente ordenada.

Ele inicializa verificando se o elemento buscado é maior ou menor que o elemento do meio da lista. Se o elemento do meio for igual ao buscado, o algoritmo é encerrado. Se o elemento do meio for menor que o buscado, o algoritmo realiza o mesmo procedimento novamente porém utilizando somente a primeira metade da lista. Se o elemento buscado for maior que o elemento do meio, o mesmo ocorre só que com a segunda metade da lista. Esse processo é então repetido até que seja encontrado o elemento desejado.

Em uma lista ordenada, esse algoritmo é mais eficiente que a busca linear, pois na maioria dos casos ele possibilita que seja efetuado menos verificações até que seja encontrado o elemento desejado.

A imagem abaixo ilustra a ideia da busca binária, de dividir a lista no meio e verificar em qual parte dela o elemento se encontra, para depois seguir com o mesmo processo somente nesta metade.

Figura 9 – Busca binária



Fonte: http://wurthmann.blogspot.com/2015/06/busca-binaria.html

Referências

Algoritmos Iterativos. Disponível em: https://www.ic.unicamp.br/~lehilton/mc102qr/unidades/07-algortimos-iterativos.html. Acesso em: 04 ago. 2020.

Bynary Vs Linear Search Through Animates Gifs. Disponível em: https://blog.penjee.com/binary-vs-linear-search-animated-gifs/. Acesso em: 04 ago. 2020.

Caelum. Disponível em: https://www.caelum.com.br/apostila-java-orientacao-objetos, Acesso em: 04 ago. 2020.

Entendendo o Algoritmo Bubble Sort em Java. Disponível em: https://www.devmedia.com.br/entendendo-o-algoritmo-bubble-sort-em-java/24812. Acesso em: 04 ago. 2020.

Selection Sort Gif. Disponível em: https://algorithms.tutorialhorizon.com/selection-sort-java-implementation/selection-sort-gif/. Acesso em: 04 ago. 2020.

Uso de Polimorfismo em Java. Disponível em: https://www.devmedia.com.br/uso-de-polimorfismo-em-java/26140. Acesso em: 04 ago. 2020.

WIKIMEDIA. *File: Bubble-sort.gif* Disponível em: https://commons.wikimedia.org/wiki/File:Bubble-sort.gif. Acesso em: 04 ago. 2020.