

**EXERCICIOS DE ESTRUTURAS DE DADOS**

Prof. Fabio Pereira da Silva

Ana Beatriz Barbosa Alves RA: 1110482113012

Faculdade de Tecnologia da Zona Leste

Tecnologia em Análise e Desenvolvimento de Sistemas – Noite

<https://github.com/anabalves/estruturas-de-dados-20211>

**Lista 1 – Vetores, Matrizes e recursividade**

**Atividades iguais serão zeradas.**

**Exercícios que envolvam codificações iguais serão zerados.**

1. Carregar um vetor [100] real de valores de saldos. Calcular e exibir:

* Média dos saldos positivos entre 100 à 1000
* Média geral dos saldos
* Soma dos saldos negativos

public class Exercicio\_L1\_1 {

static double[] *valoresSaldos* = new double[100];

static double *mediaSaldosPositivos* = 0;

static double *mediaGeralSaldos* = 0;

static double *somaSaldosNegativos* = 0;

static double *qtd100a1000* = 0;

public Double getRandomNumber() {

return ((Math.*random*() \* (1000 - (-1000)) + (-1000)));

}

public void calculaSaldos(double[] valoresSaldos, double mediaSaldosPositivos, double mediaGeralSaldos, double somaSaldosNegativos, double qtd100a1000) {

System.***out***.println("Valores: \n");

for (int i = 0; i < 100; i++) {

valoresSaldos[i] = getRandomNumber();

System.***out***.println(valoresSaldos[i]);

mediaGeralSaldos = mediaGeralSaldos + valoresSaldos[i];

if (valoresSaldos[i] > 100 & valoresSaldos[i] < 1000) {

mediaSaldosPositivos = mediaSaldosPositivos + valoresSaldos[i];

qtd100a1000++;

} else if (valoresSaldos[i] < 0) {

somaSaldosNegativos = somaSaldosNegativos + valoresSaldos[i];

}

}

System.***out***.println("----------------------------");

mediaGeralSaldos = mediaGeralSaldos / 100;

mediaSaldosPositivos = mediaSaldosPositivos / qtd100a1000;

System.***out***.println("A média dos saldos positivos entre 100 à 1000 é igual a: " + mediaSaldosPositivos);

System.***out***.println("A média geral dos saldos é igual a: " + mediaGeralSaldos);

System.***out***.println("A soma dos saldos negativos é igual a: " + somaSaldosNegativos);

}

public static void main(String[] args) {

Exercicio\_L1\_1 t = new Exercicio\_L1\_1();

t.calculaSaldos(*valoresSaldos*, *mediaSaldosPositivos*, *mediaGeralSaldos*, *somaSaldosNegativos*, *qtd100a1000*);

}

}

1. Carregar um vetor [5] inteiro. Enviar cada elemento para uma função e esta irá retornar o seu fatorial que será armazenado em um outro vetor. Exibir os dados dos vetores.

import javax.swing.JOptionPane;

public class Exercicio\_L1\_2 {

public static void main(String[] args) {

int[] valores = new int[5];

int[] valoresFatorial = new int[5];

for (int i = 0; i < 5; i++) {

valores[i] = Integer.*parseInt*(JOptionPane.*showInputDialog*("Digite um número inteiro positivo para calcular seu fatorial"));

}

for (int i = 0; i < 5; i++) {

valoresFatorial[i] = *calculaFatorial*(valores[i]);

}

for (int i = 0; i < 5; i++) {

System.***out***.println("O fatorial do número " + valores[i] + " é igual a: " + valoresFatorial[i]);

}

}

public static int calculaFatorial(int num) {

int fatorial = 1;

for (int i = 2; i <= num; i++) {

fatorial = fatorial \* i;

}

return fatorial;

}

}

1. Carregar um vetor [100] inteiros positivos ou negativos. Classificar este vetor em ordem crescente e apresentar os valores.

public class Exercicio\_L1\_3 {

public static void main(String[] args) {

int[] valores = new int[100];

for (int i = 0; i < 100; i++) {

valores[i] = *getRandomNumber*();

}

for (int i = 0; i < 100; i++) {

for (int j = 0; j < 99; j++) {

if (valores[j] > valores[j + 1]) {

int auxiliar = valores[j];

valores[j] = valores[j + 1];

valores[j + 1] = auxiliar;

}

}

}

System.***out***.println("O vetor classificado em ordem crescente: ");

for (int i = 0; i < 100; i++) {

System.***out***.println(valores[i]);

}

}

public static int getRandomNumber() {

return (int) ((Math.*random*() \* (50 - (-50)) + (-50)));

}

}

1. Carregar uma matriz [4 x 4 inteiro]. Apresentar:

* Soma dos valores no intervalo de 1 a 100
* Quantidade de números impares entre 30 a 50
* Quantidade de números divisíveis por 8
* Quantidade de números ímpares divisíveis por 3
* Fatorial do maior número informado na matriz

public class Exercicio\_L1\_4 {

public static void main(String[] args) {

double matriz[][] = new double[4][4];

double soma1a100 = 0;

double qtdImpares30a50 = 0;

double qtdDivisiveis8 = 0;

double qtdImparesDivisiveis3 = 0;

double fatorialMaiorNumero = 1;

double maiorNumeroMatriz = 0;

for (int i = 0; i < 4; i++) {

for (int j = 0; j < 4; j++) {

matriz[i][j] = *getRandomNumber*();

System.***out***.println(matriz[i][j]);

if (matriz[i][j] > 1 & matriz[i][j] < 100) {

soma1a100 = soma1a100 + matriz[i][j];

}

if (matriz[i][j] > 30 & matriz[i][j] < 50 & matriz[i][j] % 2 == 1) {

qtdImpares30a50 = qtdImpares30a50 + 1;

}

if (matriz[i][j] % 8 == 0) {

qtdDivisiveis8 = qtdDivisiveis8 + 1;

}

if (matriz[i][j] % 2 == 1 & matriz[i][j] % 3 == 0) {

qtdImparesDivisiveis3 = qtdImparesDivisiveis3 + 1;

}

if (matriz[i][j] > maiorNumeroMatriz || i == 0) {

maiorNumeroMatriz = matriz[i][j];

}

}

}

for(int i = 1; i <= maiorNumeroMatriz; i++) {

fatorialMaiorNumero = fatorialMaiorNumero \* i;

}

System.***out***.println("A soma dos valores no intervalo de 1 a 100 é igual a " + soma1a100);

System.***out***.println("A quantidade de números ímpares entre 30 a 50 é igual a " + qtdImpares30a50);

System.***out***.println("A quantidade de números divisíveis por 8 é igual a " + qtdDivisiveis8);

System.***out***.println("A quantidade de números ímpares divisíveis por 3 é igual a " + qtdImparesDivisiveis3);

System.***out***.println("O fatorial do maior número informado na matriz é igual a " + fatorialMaiorNumero);

}

public static int getRandomNumber() {

return (int) ((Math.*random*() \* (100 - 1) + 1));

}

}

1. Criar e carregar uma matriz [4 x 4] inteiro, onde os valores da diagonal principal serão carregados pela aplicação conforme o gráfico e os demais dados serão digitados pelo usuário

import javax.swing.JOptionPane;

public class Exercicio\_L1\_5 {

public static void main(String[] args) {

int matriz[][] = new int[4][4];

for (int i = 0; i < 4; i++) {

for (int j = 0; j < 4; j++) {

if (i == j) {

matriz[i][j] = (int) Math.*pow*(3, i);

} else if (i != j) {

matriz[i][j] = Integer.*parseInt*(JOptionPane.*showInputDialog*("Digite números inteiros positivos para visualizar graficamente a matriz"));

}

}

}

String visualizarGraficaMatriz = "";

for (int linha = 0; linha < 4; linha++) {

for (int coluna = 0; coluna < 4; coluna++) {

visualizarGraficaMatriz = visualizarGraficaMatriz + matriz[linha][coluna] + " ";

}

visualizarGraficaMatriz = visualizarGraficaMatriz + "\n";

}

System.***out***.println(visualizarGraficaMatriz);

}

}

1. Receba um número inteiro. Calcule e mostre o seu fatorial **recursivamente.**

import javax.swing.JOptionPane;

public class Exercicio\_L1\_6 {

public static void main(String[] args) {

int num;

do {

num = Integer.*parseInt*(JOptionPane.*showInputDialog*("Digite um número inteiro positivo para calcular seu fatorial"));

} while (num < 0);

System.***out***.print("O fatorial de " + num + " é " + *calculaFatorial*(num));

}

private static int calculaFatorial(int n) {

if (n == 0)

return 1;

else

return n \* *calculaFatorial*(n - 1);

}

}

1. Realize **recursivamente** a soma de todos os números de 1 a 100.

public class Exercicio\_L1\_7 {

public static void main(String[] args) {

int num = 1;

System.***out***.print("A soma de todos os números de 1 a 100 é igual a " + *somatoria*(num));

}

private static int somatoria(int n) {

if (n == 100) {

return 100;

} else {

int soma = n + *somatoria*(n + 1);

return soma;

}

}

}

1. Realize **recursivamente** a soma de todos os números pares de 1 a 200.

public class Exercicio\_L1\_8 {

public static void main(String[] args) {

int soma = 200;

System.***out***.print("A soma de todos os números pares de 1 a 200 é igual a " + *somaPares*(soma));

}

private static int somaPares(int num) {

if (num == 0)

return 0;

if (num % 2 == 0)

return num + *somaPares*(num - 1);

return *somaPares*(num - 1);

}

}

1. Realize **recursivamente** a soma de todos os números ímpares de 1 a 300.

public class Exercicio\_L1\_9 {

public static void main(String[] args) {

int soma = 300;

System.***out***.print("A soma de todos os números ímpares de 1 a 300 é igual a " + *somaImpares*(soma));

}

private static int somaImpares(int num) {

if (num == 0)

return 0;

if (num % 2 != 0)

return num + *somaImpares*(num - 1);

return *somaImpares*(num - 1);

}

}

1. Receba um número inteiro. Calcule e mostre a série de Fibonacci **recursivamente** até o número recebido.

import javax.swing.JOptionPane;

public class Exercicio\_L1\_10 {

public static void main(String[] args) {

int num;

do {

num = Integer.*parseInt*(JOptionPane.*showInputDialog*("Digite um número inteiro positivo para calcular a série de Fibonacci"));

} while (num <= 0);

System.***out***.print("A série de Fibonacci até o número " + num + " é ");

for (int i = 0; i < num; i++) {

System.***out***.print(*calculaSerieFibonacci*(i) + " ");

}

}

private static int calculaSerieFibonacci(int fibo) {

if (fibo < 2)

return fibo;

else

return *calculaSerieFibonacci*(fibo - 1) + *calculaSerieFibonacci*(fibo - 2);

}

}

1. Receba um número. Calcule e mostre a série 1 + 1/2 + 1/3 + ... + 1/N **recursivamente**.

import javax.swing.JOptionPane;

public class Exercicio\_L1\_11 {

public static void main(String[] args) {

int num;

do {

num = Integer.*parseInt*(JOptionPane.*showInputDialog*("Digite um número inteiro positivo para calcular a série"));

} while (num <= 0);

System.***out***.print("O cálculo da série: \n 1 ");

for (int i = 1; i < num; i++) {

System.***out***.print(" + 1/" + (i + 1));

}

System.***out***.println("\n é igual a " + (0 + *CalcularSerie*(num)));

}

public static double CalcularSerie(int num) {

if (num == 1)

return 1.0;

else

return 1.0 / num + *CalcularSerie*(num - 1);

}

}

1. Elabore um resumo do artigo “A Study on Performance Analysis of Data Structures” disponível no link abaixo:

<https://www.academia.edu/37436288/A_Study_on_Performance_Analysis_of_Data_Structures?auto=download>

As estruturas de dados são utilizadas nas situações em que é necessária uma relação lógica entre os elementos de dados a fim de armazenar os dados. O modelo lógico ou matemático de uma determinada organização de dados é designado como estrutura de dados . Alguns métodos formais de concepção e linguagens de programação enfatizam estruturas de dados, em vez de algoritmos. Ao selecionarmos uma estrutura de dados, precisamos primeiro analisar o problema para determinar as restrições de recursos que uma solução deve satisfazer, e depois determinar as operações básicas que devem ser suportadas. O artigo dá uma descrição clara sobre estruturas de dados, análise da complexidade temporal e as suas aplicações. A Seção II apresenta o trabalho relacionado que é realizado para analisar a complexidade temporal das estruturas de dados e também a classificação baseada na sua complexidade temporal. As estruturas de dados são classificadas em sete categorias que as agrupam de acordo com a sua complexidade temporal. Explica como o tempo de execução variará para diferentes tamanhos de dados para a realização de cada operação. É descrito claramente como o tempo de execução irá mudar quando o tamanho dos dados introduzidos aumentar e também indica qual a estrutura de dados mais adequada para a realização de operações específicas. Se o algoritmo envolver a adição de muitos dados, podem ser usados montes de dados e é mais adequado se for necessário um grande número de inserções e apagamentos. Independentemente do tamanho dos dados introduzidos, o tempo de execução para as estruturas de dados que pertencem à categoria 4 para realizar operações de inserção, apagamento e pesquisa é muito menor. A fila prioritária pode ser utilizada para ordenar uma lista por algum tipo de importância. O tempo de execução para a fila é constante para as operações de inserção e eliminação, mas à medida que o tamanho dos dados de entrada aumenta o tempo de execução aumenta rapidamente. As pilhas podem ser utilizadas para converter um número decimal num número binário, problema das Torres de Hanói, parsing, e na gestão da memória em tempo de execução. As filas podem ser utilizadas para Simulação, Pedidos ordenados e Pesquisas. Os temas de uma lista podem ser classificados para fins de pesquisa offast. Listas ligadas são utilizadas para implementar vários outros tipos de dados abstratos comuns, incluindo pilhas, filas, matrizes associativas, e expressões simbólicas. Binomial Heaps são utilizados para a simulação de eventos indiscretos e filas de espera prioritárias. As árvores B têm uma vasta gama de aplicações em Base de Dados, Dicionários, pesquisa de gama 1-D. As árvores são utilizadas como infraestruturas de dados em memória para que o usuário possa memorizar os passos do programa para avaliar e otimizar uma implementação. O grau de rapidez na prática dependerá das máquinas em que forem implementadas. Durante o levantamento, foram encontrados alguns pontos que podem ser mais explorados no futuro, tais como a concepção de algoritmos e estruturas de dados, a fim de minimizar o tempo de execução mesmo para tamanhos de dados de entrada maiores e tentar explorar mais profundamente nesta área de investigação.