Elias Flávio de paiva

Algoritimos e Estrutura de Dados: Algorítmos de ordenação

* Implementação

Nesta parte foram utilizados vetores pequenos gerados aleatoriamente com tamanhos definidos para validar os algoritmos.

Código da Classe principal que Chama os algoritmos

static void Main(string[] args)

{

Console.Write("Digite o tamanho do vetor desejado: "); //Solicito o tamanho do vetor que eu desejo

int tamanho = int.Parse(Console.ReadLine()) + 1; //Somo 1 ao tamanho informado, pois farei uso de sentinela

int[] vet = Funcoes.criaVetor(tamanho); //Crio um vetor do tamanho solicitado + 1, com valores aleatórios entre 0 e 10 vezes o tamanho informado

//Copio o vetor original para outros para garantir que cada um faça referência a um endereço de memória particular

int[] vet1 = Funcoes.copiaVetor(vet);

int[] vet2 = Funcoes.copiaVetor(vet);

int[] vet3 = Funcoes.copiaVetor(vet);

int[] vet4 = Funcoes.copiaVetor(vet);

int[] vet5 = Funcoes.copiaVetor(vet);

int[] vet6 = Funcoes.copiaVetor(vet);

Console.WriteLine("\nVetor Original: ");

Funcoes.imprimeVetor(vet); //Imprimo o vetor original para visualização

Console.WriteLine("\n\nVetor 1 antes da ordenação por inserção: ");

Funcoes.imprimeVetor(vet1);

Console.WriteLine("\nVetor Ordenado por inserção: ");

Funcoes.imprimeVetor(Algoritmos.insertSort(vet1));

Console.WriteLine("\n\nVetor 2 antes da ordenação por seleção: ");

Funcoes.imprimeVetor(vet2);

Console.WriteLine("\nVetor Ordenado por seleção: ");

Funcoes.imprimeVetor(Algoritmos.selectSort(vet2));

Console.WriteLine("\n\nVetor 3 antes da ordenação por bolha: ");

Funcoes.imprimeVetor(vet3);

Console.WriteLine("\nVetor Ordenado por bolha: ");

Funcoes.imprimeVetor(Algoritmos.bubbleSort(vet3));

Console.WriteLine("\n\nVetor 4 antes da ordenação por merge: ");

Funcoes.imprimeVetor(vet4);

Console.WriteLine("\nVetor Ordenado por merge: ");

Funcoes.imprimeVetor(Algoritmos.mergeSort(vet4));

Console.WriteLine("\n\nVetor 5 antes da ordenação por quick: ");

Funcoes.imprimeVetor(vet5);

Console.WriteLine("\nVetor Ordenado por quick: ");

Funcoes.imprimeVetor(Algoritmos.quickSort(vet5));

Console.WriteLine("\n\nVetor 6 antes da ordenação por SelectBubbleSort: ");

Funcoes.imprimeVetor(vet6);

Console.WriteLine("\nVetor Ordenado por SelectBubbleSort: ");

Funcoes.imprimeVetor(Algoritmos.bubbleSelectSort(vet6));

Console.ReadKey();

}

Códigos de funções auxiliares

class Funcoes

{

public static int[] criaVetor(int tamanho)

//Cria um vetor com valores aleatórios a partir de um tamanho informado

{

Random random = new Random();

int[] vet = new int[tamanho]; //Instância do vetor

//Preenchimento do vetor

for (int i = 1; i < tamanho; i++)

vet[i] = random.Next(0, 10 \* tamanho);

//Retorna o vetor criado

return vet;

}

public static void imprimeVetor(int[] vet)

//Imprime um vetor passado por parâmetro

{

//Percorre o vetor imprimindo todos os seus elementos na tela

for (int i = 1; i < vet.Length; i++)

Console.Write(vet[i] + " | ");

}

public static int[] copiaVetor(int[] vet)

//Copia um vetor em outro para criar referências diferentes na memória

{

int tamanho = vet.Length;

int[] vet2 = new int[tamanho];

for (int i = 0; i < tamanho; i++)

vet2[i] = vet[i];

return vet2;

}

public static int[] criaVetorPeloArquivo(int tamanho)

{

int[] vet = new int[tamanho];

string linha;

string[] registro;

int i = 0;

FileStream arq = new FileStream("dados\_airbnb.txt", FileMode.Open);

StreamReader leitor = new StreamReader(arq);

do

{

linha = leitor.ReadLine();

if (linha != null && i != 0)

{

registro = linha.Split('\t');

vet[i] = int.Parse(registro[0]);

}

i++;

} while (linha != null && i < tamanho);

leitor.Close();

return vet;

}

}

Códigos dos algoritmos de ordenação

public static class Algoritmos

{

/\*

\* Principal fonte para os códigos

\* Slides utiizados em aula pelo professor

\*/

public static int[] insertSort(int[] vet)

{

int i, j, x, tamanho = vet.Length;

//Percorre o vetor a partir da segunda posição até a útima

for (i = 2; i < tamanho; i++)

{

//x recebe o valor que está sendo comparado

x = vet[i];

//j recebe o valor que indica a posição imediatamente à esquerda do valor que está sendo comparado

j = i - 1;

//Sentinela, indica fim do vetor, pois tem valor igual ao do item que está sendo comparado

vet[0] = x;

//Ao fazer esta comparação este laço percore o vetor até chegar ao sentinela, procurando a posição que o valor comparado deve ficar

while (x < vet[j])

{

//Toda vez que o valor for de vet[j] for maior que x ele deve ser delocado para a direita deixando espaço livre até chegar á posição em que x deve ficar

vet[j + 1] = vet[j];

//É decrementado para percorrer o vetor da direita para a esquerda

j--;

}

//Faz a atribuição do valor de x à posição em que deve ficar

vet[j + 1] = x;

}

//Retorna o vetor ordenado

return vet;

}

public static int[] selectSort(int[] vet)

{

int i, j, min, temp, tamanho = vet.Length;

//Percorre todo o vetor a partir da posição de partida

for (i = 1; i < tamanho; i++)

{

//Atribuição do indice da posição de partida para as comparações

min = i;

//Percorre todo o vetor a partir da posição imediatamente à direita da posição de partida, em busca de valores menores

for (j = i + 1; j < tamanho; j++)

{

//Caso seja encontrado um valor menor do que o que estava na posição min, min recebe o valor do índice da posição que o contém

if (vet[j] < vet[min])

min = j;

}

//Finalmente é realizada a troca, onde a variável temporária, recebe o menor valor encontrado na busca, que é o que está na posição de índice min

temp = vet[min];

//A posição que continha o menor valor recebe o valor da posição de partida

vet[min] = vet[i];

//A posição de partida recebe o menor valor encontrado, que estava armazenado na variavel temporária

vet[i] = temp;

}

//Retorna o vetor ordenado

return vet;

}

public static int[] bubbleSort(int[] vet)

{

int tamanho = vet.Length;

//Percorre todo o vetor a partir da posição de partida

for (int i = 1; i < tamanho - 1; i++)

{

//Percorre todo o vetor a partir da posição de partida, comparando elementos vizinhos 2 a 2

for (int j = 1; j < tamanho - 1; j++)

{

//Caso seja encontrado um par em que o valor da direita seja menor que o da esquerda, estes dois são trocados de lugar

//Fazendo essa troca por todo o vetor, o elemento de maior vai sendo "borbulhado para o fim do vetor"

if (vet[j] > vet[j + 1])

{

int temp = vet[j];

vet[j] = vet[j + 1];

vet[j + 1] = temp;

}

}

}

//Retorna o vetor ordenado

return vet;

}

public static int[] mergeSort(int[] vet)

{

//Para possibilitar a chamada do algorítmo usando apenas o vetor

//Preferi implementar desta forma, com este "Chamador" antes

merger(vet, 1, vet.Length - 1);

return vet;

}

public static void merger(int[] vet, int inicio, int fim)

{

//Aqui acontece o MergeSort propriamente dito.

//Enquanto o tamanho for maior que 1 continua a quebrar o vetor em vetor menor

if (inicio < fim)

{

int meio = (inicio + fim) / 2;

//Chama novamente este método duas vezes passando a primeira metade

//Para o primeiro e a segunda metade do vetor para o segundo

merger(vet, inicio, meio);

merger(vet, meio + 1, fim);

//Reúne os vetores de forma ordenada inicio, meio e fim

merge(vet, inicio, meio, fim);

}

}

public static void merge(int[] vet, int inicio, int meio, int fim)

{

//Este método mistura os vetores resultantes do método merge, para formar um vetor ordenado

int n1 = meio - inicio + 1, n2 = fim - meio, i, j, k;

//São instanciados 2 vetores para receberem os "vetores metades"

int[] vet1 = new int[n1 + 1];

int[] vet2 = new int[n2 + 1];

//Atribui-se os valores contidos nos vetores aos "vetores metades"

for (i = 0; i < n1; i++)

vet1[i] = vet[inicio + i];

for (j = 0; j < n2; j++)

vet2[j] = vet[meio + j + 1];

//Atribui-se um valor alto à posição final dos vetores metades

vet1[i] = 9999;

vet2[j] = 9999;

//Reinicía-se os valores das variáveis i e j

i = 0;

j = 0;

//Percorre-se o vetor final guardando de forma ordenada os valores

//Contidos nos vetores metade ao vetor final

for (k = inicio; k <= fim; k++)

{

if (vet1[i] <= vet2[j])

vet[k] = vet1[i++];

else

vet[k] = vet2[j++];

}

}

public static int[] quickSort(int[] vet)

{

//Para possibilitar a chamada do algorítmo usando apenas o vetor

//Preferi implementar desta forma, com este "Chamador" antes

quick(vet, 1, vet.Length - 1);

return vet;

}

public static void quick(int[] vet, int esquerda, int direita)

{

//Atribuo a i e j os valores dos indices extremos do vetor

//Escolo o pivô como sendo o elemento central

int i = esquerda, j = direita, temp, pivo = vet[(direita + esquerda) / 2];

//Percorro todo o vetor até que i ultrapasse o valor de j

//Eles são iterados em sentidos opostos

while (i <= j)

{

//Enquanto i não chegar à outra extremidade e não for igual ao pivô

//Atualizo o valor incrementando 1

while (vet[i] < pivo && i < direita) i++;

//Enquanto j não chegar à outra extremidade e não for igual ao pivô

//Atualizo o valor decrementando 1

while (vet[j] > pivo && j > esquerda) j--;

//Após a seleção de i e de j testo para saber se devem ser trocados

//Se o teste for positivo troco os falores contidos nestas posições

if (i <= j)

{

temp = vet[i];

vet[i] = vet[j];

vet[j] = temp;

i++;

j--;

}

}

//Caso j ainda não tenha alcançado o outro etremo

//Chamo o algoritmo novamente passando uma parte do vetor

if (j > esquerda)

quick(vet, esquerda, j);

//Caso i ainda não tenha alcançado o outro etremo

//Chamo o algoritmo novamente passando uma parte do vetor

if (i < direita)

quick(vet, i, direita);

}

public static int[] bubbleSelectSort(int[] vet)

{

int tamanho = vet.Length;

//Percorro todo o vetor a partir da primeira posição

//Fixando a cada passada o elemento com indice igual ao número da passada

for(int i = 1; i < tamanho; i++)

{

//Percorro todo o vetor a partit da posição imediatamente à direita da

//Que tem índice com o número da passada buscando elementos menores

for (int j = i+1; j < tamanho; j++)

{

//Toda vez que encontro um elemento menor efetuo a troca

if (vet[i] > vet[j])

{

int temp = vet[i];

vet[i] = vet[j];

vet[j] = temp;

}

}

}

//Retorno o vetor ordenado.

return vet;

}

}

* Análise de Desempenho Experimental

Códigos utilizados para realizar as medições

static void Main(string[] args)

{

FileStream arquivo = new FileStream("Dados da ordenação.txt", FileMode.OpenOrCreate);

StreamWriter registrador = new StreamWriter(arquivo);

{

int tamanho = 2000;

for (int i = 1; i < 128; i \*= 2)

{

int[] vet = Funcoes.criaVetorPeloArquivo(tamanho);

registrador.WriteLine("Ordenação com vetor de " + tamanho + " elementos");

tamanho = i \* 2000;

registrador.Write("\nINSERÇÃO");

for (int j = 1; j <= 5; j++)

{

var watch = System.Diagnostics.Stopwatch.StartNew();

Algoritmos.insertSort(Funcoes.copiaVetor(vet));

watch.Stop();

var elapsedMs = watch.ElapsedMilliseconds / 1000.0;

registrador.Write("|" + elapsedMs + "|");

var ramUsage = System.Diagnostics.Process.GetCurrentProcess().PeakWorkingSet64;

var allocationInMB = ramUsage / (1024 \* 1024);

registrador.Write("|" + allocationInMB);

}

registrador.Write("\nSELEÇÃO");

for (int j = 1; j <= 5; j++)

{

var watch = System.Diagnostics.Stopwatch.StartNew();

Algoritmos.selectSort(Funcoes.copiaVetor(vet));

watch.Stop();

var elapsedMs = watch.ElapsedMilliseconds / 1000.0;

registrador.Write("|" + elapsedMs + "|");

var ramUsage = System.Diagnostics.Process.GetCurrentProcess().PeakWorkingSet64;

var allocationInMB = ramUsage / (1024 \* 1024);

registrador.Write("|" + allocationInMB);

}

registrador.Write("\nBOLHA");

for (int j = 1; j <= 5; j++)

{

var watch = System.Diagnostics.Stopwatch.StartNew();

Algoritmos.bubbleSort(Funcoes.copiaVetor(vet));

watch.Stop();

var elapsedMs = watch.ElapsedMilliseconds / 1000.0;

registrador.Write("|" + elapsedMs + "|");

var ramUsage = System.Diagnostics.Process.GetCurrentProcess().PeakWorkingSet64;

var allocationInMB = ramUsage / (1024 \* 1024);

registrador.Write("|" + allocationInMB);

}

/\*registrador.Write("\nMERGESORT");

for (int j = 1; j <= 5; j++)

{

var watch = System.Diagnostics.Stopwatch.StartNew();

Algoritmos.mergeSort(Funcoes.copiaVetor(vet));

watch.Stop();

var elapsedMs = watch.ElapsedMilliseconds / 1000.0;

registrador.Write("|" + elapsedMs + "|");

var ramUsage = System.Diagnostics.Process.GetCurrentProcess().PeakWorkingSet64;

var allocationInMB = ramUsage / (1024 \* 1024);

registrador.Write("|" + allocationInMB);

}\*/

//Não consegui fazer o MergeSort funcionar com o vetor gerado pelo arquivo

//Mas, funciona perfeitamente com um vetor aleatório.

registrador.Write("\nQUICKSORT");

for (int j = 1; j <= 5; j++)

{

var watch = System.Diagnostics.Stopwatch.StartNew();

Algoritmos.quickSort(Funcoes.copiaVetor(vet));

watch.Stop();

var elapsedMs = watch.ElapsedMilliseconds / 1000.0;

registrador.Write("|" + elapsedMs + "|");

var ramUsage = System.Diagnostics.Process.GetCurrentProcess().PeakWorkingSet64;

var allocationInMB = ramUsage / (1024 \* 1024);

registrador.Write("|" + allocationInMB);

}

registrador.Write("\nBOLHA COM SELEÇÃO");

for (int j = 1; j <= 5; j++)

{

var watch = System.Diagnostics.Stopwatch.StartNew();

Algoritmos.bubbleSelectSort(Funcoes.copiaVetor(vet));

watch.Stop();

var elapsedMs = watch.ElapsedMilliseconds / 1000.0;

registrador.Write("|" + elapsedMs + "|");

var ramUsage = System.Diagnostics.Process.GetCurrentProcess().PeakWorkingSet64;

var allocationInMB = ramUsage / (1024 \* 1024);

registrador.Write("|" + allocationInMB);

}

registrador.WriteLine("\n");

}

}

registrador.Close();

Console.WriteLine("Ordenação Concluida");

Console.ReadKey();

}

* Bibliografia

LEAL, Gislaine Camila Lapasini, Algoritmos e Lógica de Programação II, Núcleo de Educação a Distância Unicesumar – Maringá-Pr, 2016.

SOUZA, Kleber Jacques F. de, Aula 06 – Ordenação Interna – Bolha, Seleção e Inserção.

SOUZA, Kleber Jacques F. de, Aula 07 – Ordenação Interna – MergeSort e QuickSort.