Experiência de programador

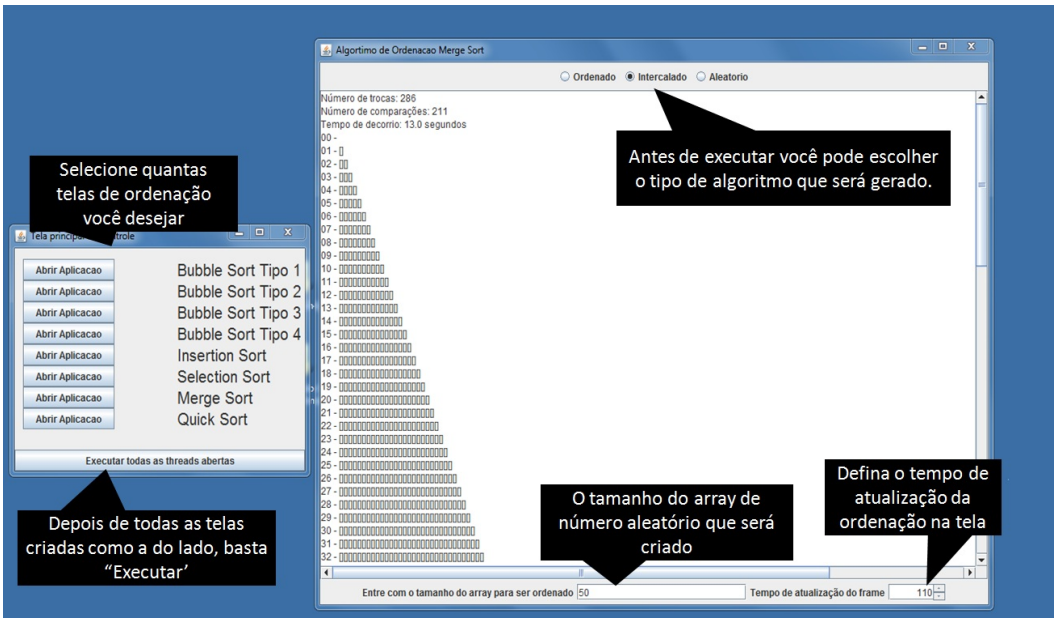
Joao Pedro Schmitt

Algoritmos de Ordenação em Java

 schmittjoaopedro . Java 14 14+00:00 Maio 14+00:00 20134 04+00:00 Novembro 04+00:00 2017 4 Minutes

Atualmente com a acensão da linguagem Java, muitos algoritmos feitos em linguagens legadas tem deixado de ser usados e essas tecnologias tem sido atualizadas para sistemas orientado a objetos. Neste tópico vou buscar apresentar os principais tipos de ordenação de dados e apresentar seu respectivos algoritmos em Java, para quem preferir testar esses algoritmos estou disponibilizando o link para download da aplicação que faz a ordenação de algoritmos em java em tempo real usando threads, aonde você pode tirar informações relevantes como número de trocas, comparações e avaliar a sistemática de ordenação para cada tipo de dado. O código fonte está disponível no seguinte repositório do GitHub (https://github.com/schmittjoaopedro/algoritmos-ordenacao-java).

Segue abaixo uma imagem de como utilizar a aplicação, desse modo facilitando o entendimento, esse programa gera valores aleatórios para ordenação. Esse programa é somente demonstrativo em que o objetivo é auxiliar no aprendizado dessas estruturas.



(https://joaoschmitt.files.wordpress.com/2013/05/programaordenador.jpg)

Alguns algoritmos de ordenação em java, esses algoritmos se encontram na integra da mesma forma que foram utilizados no programa citado acima, neste modelo somente estarei passando o método que executa a ordenação, em todos os algoritmos o getOrdenaveis() retorna o array que está a ser ordenado.

Bubble Sort

O método bubble sort trabalha comparando todos os valores dos vetores, por exemplo o primeiro com o segundo depois com o terceiro, e assim por diante até o último valor, depois ele inicia o mesmo processo só que uma posição a mais comparando o segundo com o primeiro, depois com o terceiro. Esse processo de comparação passa por uma condição que se um valor for menor que o outro eles trocam de posição. O algoritmo abaixo descreve esse processo.

```
public synchronized void ordenar() {
int troca = 1;
int n = 0;
setTempoInicial();
while (n < getOrdenaveis().length && troca == 1) {
troca = 0;
MostraStatusLoop();
for (int i = 0; i < getOrdenaveis().length – 1; i++) {
setNumeroComparacoes();
if (getOrdenaveis()[i] > getOrdenaveis()[i + 1]) {
setNumeroTrocas();
troca = 1;
trocarPosicaoVetores(i, (i + 1));
}
}
atualizarPainel();
n++;
try{
sleep(getTimeSleepValue());
}catch(InterruptedException er){
JOptionPane.showMessageDialog(null, “O Aplicativo parou inesperadamente”);
}
setTempoFinal();
}
}
```

Insertion Sort

Esse algoritmo se baseia no processe de inserção, dessa forma quando ele recebe um novo valor ele sempre vai o alocar na posição que corresponde a sua faixa de ordenação.

```
public synchronized void ordenar() {
setTempoInicial();
for (posicao = 1; posicao < getOrdenaveis().length; posicao++) {
MostraStatusLoop();
eleito = getOrdenaveis()[posicao];
aux = posicao – 1;
```

```
setNumeroComparacoes();
while (aux >= 0 && getOrdenaveis()[aux] > eleito) {
setNumeroTrocas();
getOrdenaveis()[aux + 1] = getOrdenaveis()[aux];
aux–;
}
getOrdenaveis()[aux + 1] = eleito;
atualizarPainel();
setTempoFinal();
try{
sleep(getTimeSleepValue());
}catch(InterruptedException e){
JOptionPane.showMessageDialog(null, “Aplicativo encerrou inesperadamente”);
}
}
}
```

Merge Sort

O vetor vai sendo dividido em blocos com a metade do inicial, no final quando sobra um único valor ou dois valores esses valores são comparados pra todos os blocos que foram divididos, depois de feitos esse processo os blocos são intercalados de forma ordenada.

```
public void merge(int vetor[], int inicio, int fim, int cont){
int meio;
cont++;
if(inicio < fim){
meio = (inicio + fim) /2;
merge(vetor, inicio, meio,cont);
merge(vetor, meio + 1, fim,cont);
intercalar(vetor, inicio, meio, fim);
}
MostraStatusLoop();
atualizarPainel();
try{
sleep(getTimeSleepValue());
}catch(InterruptedException er){
JOptionPane.showMessageDialog(null, “O Aplicativo Parou Inesperadamente”);
}
}
```

```
public void intercalar(int vetor[], int inicio, int meio, int fim){
int posLivre = inicio;
int inicioVetor1= inicio;
int inicioVetor2 = meio + 1;
int aux[] = new int[vetor.length];


while(inicioVetor1 <= meio && inicioVetor2 <= fim){
setNumeroComparacoes();
if(vetor[inicioVetor1] <= vetor[inicioVetor2]){
setNumeroTrocas();
aux[posLivre] = vetor[inicioVetor1];
inicioVetor1++;
}else{
setNumeroTrocas();
aux[posLivre] = vetor[inicioVetor2];
inicioVetor2++;
}
posLivre++;
}
for (int i = inicioVetor1; i <= meio; i++) {
setNumeroTrocas();
aux[posLivre] = vetor[i];
posLivre++;
}
for (int i = inicioVetor2; i <= fim; i++) {
setNumeroTrocas();
aux[posLivre] = vetor[i];
posLivre++;
}
for (int i = inicio; i <= fim; i++) {
vetor[i] = aux[i];
}
}
```

Quick Sort (o mais eficiente)

Faz um processo similar ao merge sort, aonde a medida que o vetor vai sendo ordenado vão sendo realizadas partições recursivas em torno de ganhar desempenho.

```
public synchronized void quickSort(int vet[], int ini, int fim, int cont) {
int meio;
```

```
if (ini < fim) {
meio = partition(vet, ini, fim);
quickSort(vet, ini, meio, cont);
quickSort(vet, meio + 1, fim, cont);
}


MostraStatusLoop();
atualizarPainel();
try{
sleep(getTimeSleepValue());
}catch(InterruptedException er){
JOptionPane.showMessageDialog(null, “O Aplicativo Parou Inesperadamente”);
}
}


public int partition(int vet[], int ini, int fim) {
int pivo, topo, i;
pivo = vet[ini];
topo = ini;


for (i = ini + 1; i <= fim; i++) {
setNumeroComparacoes();
if (vet[i] < pivo) {
setNumeroTrocas();
vet[topo] = vet[i];
vet[i] = vet[topo + 1];
topo++;
}
}
vet[topo] = pivo;
return topo;
}
```

Selection Sort

Neste método o vetor seleciona sequencialmente os vetores da ordenação e localiza um próximo que seja o menor ou maior (conforme o tipo de ordenação crescente ou decrescente) então esse valor é comparado com o valor selecionado no inicio, se a condição é verdadeira a posição dos valores são trocadas ordenando o algoritmo.

```
public synchronized void ordenar() {
setTempoInicial();
for (posicao = 0; posicao < getOrdenaveis().length – 1; posicao++) {


MostraStatusLoop();


eleito = getOrdenaveis()[posicao];
menor = getOrdenaveis()[posicao + 1];
pos = posicao + 1;
for (int j = posicao + 2; j < getOrdenaveis().length; j++) {
setNumeroComparacoes();
if (getOrdenaveis()[j] < menor) {
menor = getOrdenaveis()[j];
pos = j;
}
}
if (menor < eleito) {
setNumeroTrocas();
getOrdenaveis()[posicao] = getOrdenaveis()[pos];
getOrdenaveis()[pos] = eleito;
}


atualizarPainel();
try {
sleep(getTimeSleepValue());
} catch (InterruptedException er) {
JOptionPane.showMessageDialog(null, “Aplicativo encerrado inesperadamente”);
}
}
setTempoFinal();
}
```

Autor: João Pedro Schmitt – 05/2013

Com as etiquetas :
algoritmos,
codigo,
dados,
estrutura,
java,

Ordenação,
ordenador,
ordenar,
programa,
script,
vetores

Publicado por schmittjoaopedro

Graduado como bacharel em Sistemas de Informação pelo Centro Universitário Católica de Santa Catarina campus Jaraguá do Sul. Formado no Ensino Médio pelo Senai com Técnico em Redes de Computadores Articulado. Atualmente desenvolvedor JEE/Web em Sistemas de Engenharia na WEG. Pesquisador no período de faculdade em Informática pela Católica de Santa Catarina. Contato 47 - 99615 2305 E-mail: schmittjoaopedro@gmail.com Web page: https://joaoschmitt.wordpress.com/ Linkedin: https://www.linkedin.com/in/joao-pedro-schmitt-60847470/ Curriculum lattes: http://lattes.cnpq.br/9304236291664423 Twitter: @JooPedroSchmitt Ver todos os artigos de schmittjoaopedro

3 thoughts on “Algoritmos de Ordenação em Java”

heuheuehue diz:
11 11+00:00 Novembro 11+00:00 2013 às 21:03 | Editar
BRBRBRBRBR HUEHUEHEUHEUHE

 Responder
joão Victor diz:
4 04+00:00 Novembro 04+00:00 2017 às 0:24 | Editar
a aplicação não esta mais disponível para download, sera que não poderia disponibiliza-la novamente?

 Responder
schmittjoaopedro diz:
4 04+00:00 Novembro 04+00:00 2017 às 11:50 | Editar
Segue link: https://github.com/schmittjoaopedro/algoritmos-ordenacao-java

 Responder