Universidade Cruzeiro do Sul

Estruturas de Dados I

Conteúdos

Exercícios sobre:

- Vetores.
- Aleatorização de vetores.
- Busca sequencial. Busca binária.
- Métodos recursivos.
- Métodos de ordenação.

Elaboração

Prof. Manuel F. Paradela Ledón.

Exercício 1 - para entregar

para entregar

Na Aula 03 estudamos o algoritmo de Fisher-Yates para "embaralhar" aleatoriamente os elementos de um vetor, utilizando a classe pronta ArrayList:

public void **aleatorizar** (ArrayList lista)

Utilizando a mesma lógica deste algoritmo, crie um método que permita "aleatorizar" um vetor comum (utilizando somente vetores), com o cabeçalho:

public void **aleatorizarVetor** (double vet[])

Como sugestão, para eliminar um elemento do vetor original, poderia utilizar o método **deslocaEsquerda**, também estudado na Aula 03:

```
void  deslocaEsquerda (double vet[], int de, int ate) {
            if(de>ate)return;
            if(de<=0)de=1;
            if(ate > vet.length-1)ate=vet.length-1;
            // elimina o item na posição (d-1) e desloca os restantes
            for (int i = (de - 1); i < ate; i++) vet[i] = vet[i+1];
            vet[ate] = 0;  // só para marcar o item final
}
```

A resposta estará disponível em Ex_random_mais_completo.java

Veja também uma solução, sem utilizar o algoritmo de Fisher-Yates, no projeto NetBeans em Embaralhar_outra_solução.

Exercício 2

Precisamos implementar algoritmos para efetuar uma **busca** de um país dentro de um vetor de Strings. Analise as características dos vetores mostrados e responda V ou F (verdadeiro ou falso) para cada uma das assertivas a seguir.

```
String paisesA [] = { "Ucrânia", "Turquia", "Suíça", "México", "França", "Espanha",
                      "Chile", "Brasil , "Argentina" };
String paisesB [] = { "Argentina", "Brasil", "Chile", "Dinamarca", "Espanha", "França",
                      "Inglaterra", "Turquia", "Uruguai" };
String paisesC [] = { "Canadá", "Áustria", "Chile", "Itália", "Portugal", "Grécia",
                      "Angola", "Moçambique", "Rússia" };
```

1.___ Para buscar um país em paisesA[] o método mais eficiente seria o da busca binária.
2.___ Para buscar um país em paisesB[] o método mais eficiente seria o da busca binária.
3.___ Para buscar um país em paisesC[] seria adequado o método de busca binária.
4.___ Para buscar um país em paisesC[] seria adequado o método de busca sequencial.
5.___ Seria possível buscar um país em paisesB[] com o método de busca sequencial.
6.___ Com o método de busca sequencial, no pior caso, encontrar um país tem complexidade O(n).
7.___ Com o método de busca binária, no pior caso, encontrar um país tem complexidade O(log n).
8.___ Com o método de busca sequencial, no pior caso, encontrar um país tem complexidade $O(n^2)$.

Exercício 2 - respostas

Precisamos implementar algoritmos para efetuar uma busca de um país dentro de um vetor de Strings. Analise as características dos vetores mostrados e responda V ou F (verdadeiro ou falso) para cada uma das assertivas a seguir.

```
String paisesA [] = { "Ucrânia", "Turquia", "Suíça", "México", "França", "Espanha",
                      "Chile", "Brasil , "Argentina" };
String paisesB [] = { "Argentina", "Brasil", "Chile", "Dinamarca", "Espanha", "França",
                      "Inglaterra", "Turquia", "Uruguai" };
String paisesC [] = { "Canadá", "Áustria", "Chile", "Itália", "Portugal", "Grécia",
                      "Angola", "Moçambique", "Rússia" };
```

1. _V_ Para buscar um país em paisesA[] o método mais eficiente seria o da busca binária.
2. _V_ Para buscar um país em paisesB[] o método mais eficiente seria o da busca binária.
3. _F_ Para buscar um país em paisesC[] seria adequado o método de busca binária.
4. _V_ Para buscar um país em paisesC[] seria adequado o método de busca sequencial.
5. _V_ Seria possível buscar um país em paisesB[] com o método de busca sequencial.
6. _V_ Com o método de busca sequencial, no pior caso, encontrar um país tem complexidade O(n).
7. _V_ Com o método de busca binária, no pior caso, encontrar um país tem complexidade O(log n).
8. _F_ Com o método de busca sequencial, no pior caso, encontrar um país tem complexidade $O(n^2)$.

Exercício 3 - para entregar

para entregar

Implemente (ou adapte os algoritmos estudados) os métodos de **busca sequencial** e **busca binária**, para efetuar a busca de um país em exemplos de vetores como mostrados a seguir. Teste os métodos com os vetores exemplos mostrados a seguir.

Efetue chamadas para testar seus métodos, por exemplo, com os vetores:

```
String paisesA [] = { "Ucrânia", "Turquia", "Suíça", "México", "França", "Espanha",
                      "Chile", "Brasil , "Argentina" };
String paisesB [] = { "Argentina", "Brasil", "Chile", "Dinamarca", "Espanha", "França",
                      "Inglaterra", "Turquia", "Uruguai" };
String paisesC [] = { "Canadá", "Áustria", "Chile", "Itália", "Portugal", "Grécia",
                      "Angola", "Moçambique", "Rússia" };
```

Tarefas:

- Criar um método: int **buscaSequencial**(String vet[], String buscado).
- Criar um método: int **buscaBinariaEmListaCrescente** (String vet[], String buscado).
- Criar um método: int **buscaBinariaEmListaDecrescente** (String vet[], String buscado).
- Testar os métodos anteriores. Entregar um projeto NetBeans com todos os métodos e testes dentro.

Exercício 4 (veja antes o próximo slide)

O método Merge Sort utiliza uma lógica para fundir (fusionar, misturar) os elementos de duas partes de um vetor para que o trecho total fique ordenado.

Utilizando uma lógica de fusão semelhante, construir um método simples:

mergePaises (String a[], String b[], String res[])

que recebe dois vetores de Strings **a** e **b**, ordenados em ordem crescente, e retorne o vetor **res** com todos os países (dos dois vetores) também em ordem crescente. Considere que **a** e **b** poderiam ter países repetidos (repita os mesmos na cópia).

Teste seu método, por exemplo, fusionando os vetores:

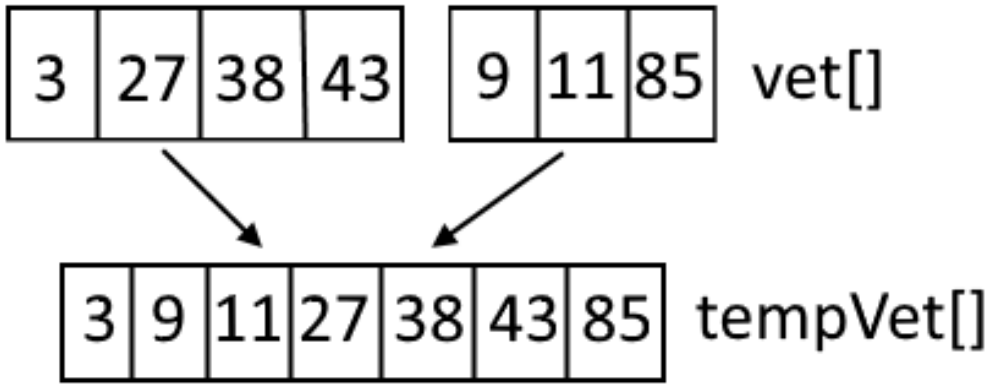
```
String veta[] = { "Angola", "Chile", "Grécia", "Itália", "Moçambique", "Portugal",
                  "Rússia", "Suécia" };
String vetb[] = { "Argentina", "Brasil", "Chile", "Dinamarca", "Espanha", "França",
                  "Inglaterra", "Turquia", "Uruguai" };
```

A resposta estará depois disponível em MergePaises.java

Exercício 4 (lembrando)

O método Merge Sort utiliza uma lógica para fundir (fusionar, misturar) os elementos de dois trechos de um vetor, para que os trechos fundidos fiquem finalmente ordenados dentro destas mesmas posições de um vetor auxiliar.

...



Nesta figura mostramos o exemplo de uma etapa de ordenação pelo método Merge Sort.

Conquista: as soluções dos subproblemas são unidas em uma única solução (também em etapas para cada tamanho de partição) até chegar no vetor final ordenado.

Exercício 5

Adapte o método **Quick Sort** estudado, considerando as alterações:

- Seleção do pivô utilizando o algoritmo de **mediana de três valores**.
- Ordenar valores **inteiros** em **ordem decrescente**.

Teste seu algoritmo com algum vetor de valores inteiros, por exemplo:

```
int x[] = {3, 5, -12, 34, 91, 81, 91, 2, 0, 180, 21, 76, 22, 20, 19, 43, -15, 1, 65};
```

A resposta estará disponível no projeto NetBeans OrdenacaoQuickSortMediana3

Bibliografia (oficial) da disciplina

BIBLIOGRAFIA BÁSICA	BIBLIOGRAFIA COMPLEMENTAR
CORMEN, Thomas H.; LEISERSON, Charles E.; RIVEST, Ronald L.; STEIN, Clifford. Algoritmos: teoria e prática. Rio de Janeiro: Elsevier, Campus, 2002. GOODRICH, Michael T.; TAMASSIA, Roberto. Estruturas de dados e algoritmos em Java. 2. ed. Porto Alegre, São Paulo: Bookman, 2002. SZWARCFITER, Jayme Luiz. Estruturas de dados e seus algoritmos. 3. Rio de Janeiro: LTC, 2010, recurso online, ISBN 978-85-216-2995-5.	ASCENCIO, A. F. G.; ARAÚJO, G. S. Estruturas de Dados. São Paulo: Pearson, 2011. [eBook] EDELWEISS, N.; GALANTE, T. Estruturas de Dados. Porto Alegre: Bookman, 2009. [eBook] MORIN, P. Open Data Structures (in Java) Creative Commons, 2011. Disponível em http://opendatastructures.org/ods-java.pdf [eBook] PUGA, S.; RISSETTI, G. Estruturas de Dados com aplicações em Java, 2a ed. São Paulo: Pearson, 2008. [eBook] SHAFFER, C. A.; Data Structures and Algorithm Analysis. Virginia Tech, 2012. Disponível em