UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
Campus Guarapuava
Tecnologia em Sistemas para Internet

Resolução de Problemas
Prof. Dr. Eleandro Maschio

Métodos de Busca e de Ordenação

Exercício 1

ORDENA TRÊS – Modele uma classe que tenha três atributos inteiros (recebidos pelo construtor). Implemente um método que ordene esses três valores inteiros. Três estruturas condicionais (todas sem o bloco de **caso-contrário**) são suficientes. Não use métodos pré-definidos de ordenação.

Exercício 2

MAIOR DE TRÊS – Na mesma classe do exercício anterior, implemente um método que encontre o maior dos três valores. Duas estruturas condicionais (todas sem o bloco de **caso-contrário**) são suficientes. Não use método pré-definido para retornar o maior valor.

Exercício 3

BARALHO – Com o uso do baralho, aplique os algoritmos de ordenação *Selection Sort*, *Bubble Sort* e *Insertion Sort* aos seguintes arranjos. Obtenha o número de comparações e o número de trocas feitas em cada um dos casos. Use cartas de 1 a 10 de um mesmo naipe (considere que o ás é o 1).

(a) Primeiro arranjo: 10, 9, 8, 7, 6, 5, 4, 3, 2, 1.

(b) Segundo arranjo: 10, 1, 9, 2, 8, 3, 7, 4, 6, 5.

(c) Terceiro arranjo: 4, 5, 6, 1, 2, 3, 7, 8, 9, 10.

Exercício 4

CRESCENTE – Modele uma classe que tenha como atributo uma uma matriz unidimensional de 20 inteiros (recebida pelo construtor). Implemente um método iterativo que retorne se essa matriz está em ordem crescente. Abandone a verificação tão logo se perceba o contrário.

Exercício 5

BUBBLE MELHORADO – Na mesma classe do exercício anterior, implemente um aperfeiçoamento do *Bubble Sort* em que o método seja interrompido caso não ocorram trocas durante uma iteração do laço mais externo.

Exercício 6

EMBARALHAR – Na mesma classe do Exercício 4, implemente um método iterativo que embaralhe a matriz unidimensional.

Exercício 7

GNOME SORT – A ordenação do gnomo (*Gnome Sort*) consegue ser um algoritmo mais simples do que o próprio método bolha. Baseia-se na técnica ancestral em que os gnomos de jardim ordenam vasos de flores do menor para o maior.

Como os vasos são pesados, os pequenos gnomos não conseguem carregá-los por longas distâncias. Assim, comparam e trocam-nos somente de dois a dois.

Basicamente, o gnomo olha para o vaso de flor que tem na frente e também para o vaso anterior. Se estiverem na ordem correta, o gnomo vai para o próximo vaso da sequência. Caso contrário, o gnomo troca um vaso pelo outro e volta para o vaso anterior.

As condições limites são: (1) se não houver vaso anterior, o gnomo vai para o próximo vaso; e (2) se não houver vaso seguinte, o trabalho acabou.

Pensando computacionalmente, o algoritmo extraído desta lógica também percorre a sequência comparando elementos dois a dois. Assim que encontrar um elemento em posição incorreta, ou seja,

um número maior antes de um menor, troca a posição dos elementos, e volta com o elemento que estava fora de ordem até que encontre o respectivo lugar.

Seguem os passos para ordenar a sequência da primeira linha. Observe que sempre se avança até achar um elemento fora de ordem e, através de trocas sucessivas, traz-se o elemento até a posição correta:

6 2 9 4 7 1 8 5 0

2 6 9 4 7 1 8 5 0

2 6 9 4 7 1 8 5 0

2 6 4 9 7 1 8 5 0

2 4 6 9 7 1 8 5 0

2 4 6 9 7 1 8 5 0

2 4 6 7 9 1 8 5 0

2 4 6 7 9 1 8 5 0

2 4 6 7 1 9 8 5 0

2 4 6 1 7 9 8 5 0

2 4 1 6 7 9 8 5 0

2 1 4 6 7 9 8 5 0

1 2 4 6 7 9 8 5 0

1 2 4 6 7 9 8 5 0

1 2 4 6 7 8 9 5 0

1 2 4 6 7 8 9 5 0

1 2 4 6 7 8 5 9 0

1 2 4 6 7 5 8 9 0

1 2 4 6 5 7 8 9 0

1 2 4 5 6 7 8 9 0

1 2 4 5 6 7 8 9 0

1 2 4 5 6 7 8 0 9

1 2 4 5 6 7 0 8 9

1 2 4 5 6 0 7 8 9

1 2 4 5 0 6 7 8 9

1 2 4 0 5 6 7 8 9

1 2 0 4 5 6 7 8 9

1 0 2 4 5 6 7 8 9

0 1 2 4 5 6 7 8 9

Na mesma classe do Exercício 4, implemente um método que ordene a matriz unidimensional utilizando o *Gnome Sort*.

Exercício 8

SELECTION RECURSIVO – Na mesma classe do Exercício 4, proponha um implementação recursiva para o *Selection Sort*. A cada chamada recursiva, traga o menor elemento da subsequência para a primeira posição desta. Então, aplique a recursão para ordenar os demais elementos.

Exercício 9

EXTREMIDADE MAIS PRÓXIMA – Na mesma classe do Exercício 4, implemente uma busca sequencial que: (1) verifique se o número pesquisado pode estar presente na sequência, ou seja, $a_0 <= x <= a_{n-1}$; (2) inicie a busca pela extremidade mais próxima do número pesquisado; e (3) interrompa a busca tão logo se encontre um elemento que ultrapasse o número pesquisado. Observe que essa busca se aplica a elementos ordenados. Portanto, invoque um método de ordenação antes.

Exercício 10

MEGA SENA – Modele um cartão da Mega Sena, de maneira que uma matriz unidimensional de 6 a 20 números seja recebida pelo construtor. Assuma que serão números válidos, em ordem e sem repetição. Feito isso, implemente um método que receba as seis dezenas premiadas, na ordem do sorteio e retorne o número de acertos. Provenha outro método que retorne, conforme o caso, as mensagens: sena, quina, quadra, não premiado.

Exercício 11

ENIGMA SORT – Considere o código-fonte abaixo na linguagem C. Abstraia e explique a lógica utilizada para a ordenação da sequência.

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#define TAM 10

int main()
{
   srand(time(NULL));
   printf("Enigma Sort\n");
   printf("===========\n\n\n");

   int dados[TAM], i = 0;
   printf("Sequencia Gerada\n");
   printf("----------------\n");
   for (i = 0; i < TAM; i++)
   {
       dados[i] = rand() % 100;
       printf("%2d ", dados[i]);
   }

   int j, aux;
   bool flag;
   do
   {
      for(i = 0; i < TAM; i++)
      {
         j = rand() % TAM;
         if (i != j)
         {
            aux = dados[i];
            dados[i] = dados[j];
            dados[j] = aux;
         }
      }

      flag = true;
      for (i = 1; flag && (i < TAM); i++)
      {
          if (dados[i] < dados[i-1])
             flag = false;
      }
   } while (!flag);

   printf("\n\n");
   printf("Sequencia Ordenada\n");
   printf("------------------\n");
   for (i = 0; i < TAM; i++)
       printf("%2d ", dados[i]);

   return(0);
}
```

Exercício 12

BINGO — Modele e implemente uma classe que gere uma cartela de bingo. A cartela é composta por 25 números de 1 a 50 divididos em cinco colunas. Não deve haver números repetidos. O método `toString()` retorna a cartela em uma saída formatada.

Segue um exemplo de cartela gerada:

B	I	N	G	O
2	11	23	32	44
4	13	25	34	46
6	14	27	36	47
7	18	29	37	48
9	22	30	39	50

Exercício 13

PESQUISA – Há diferença prática entre pesquisar uma chave em uma matriz unidimensional ordenada ou desordenada? Justifique o que pode ser considerado na implementação.

Exercício 14

BUSCA BINÁRIA – Calcule, em uma planilha, o número de comparações para o pior caso de uma busca binária em se tratando de conjuntos de 100, 1.000, 10.000 e 100.000 elementos. Relacione com o ganho percentual com relação à busca sequencial.

Exercício 15

SUDOKU – Sudoku é um quebra-cabeça baseado na colocação lógica de números. O objetivo do jogo é a alocação dos números de 1 a 9 em cada uma das células vazias numa grade de 9x9, constituída por 3x3 subgrades chamadas regiões. O quebra-cabeça contém algumas pistas iniciais. Cada coluna, linha e região só pode ter um número de cada um dos 1 a 9. Resolver o problema requer apenas raciocínio lógico e algum tempo. Os problemas são normalmente classificados de acordo com a dificuldade apresentada. Observe um problema Sudoku fácil e a respectiva solução:

4	3			9	1	8		6
	9				7	4	2	5
6	2							
9	8			1				
7			3		5			9
				2			8	4
							4	8
1	6	8	4				5	
3		4	8	6			9	7

4	3	**5**	**2**	9	1	8	**7**	6
8	9	**1**	**6**	**3**	7	4	2	5
6	2	**7**	**5**	**4**	**8**	**9**	**3**	**1**
9	8	**2**	**7**	1	**4**	**5**	**6**	**3**
7	**4**	**6**	3	**8**	5	**2**	**1**	9
5	**1**	**3**	**9**	2	**6**	**7**	8	4
2	**7**	**9**	**1**	**5**	**3**	**6**	4	8
1	6	8	4	**7**	**9**	**3**	5	**2**
3	**5**	4	8	6	**2**	**1**	9	7

Implemente uma classe cujo construtor receba uma matriz inteira 9x9. Provenha um método que verifique e retorne se ela representa uma solução válida para o Sudoku. Interrompa a verificação tão logo uma restrição seja violada. São mensagens de erro obtidas pelo método `getError()`:

- Linha 1, coluna 3: o número deve estar entre 1 e 9;
- Linha 2, coluna 1: o número 7 já está presente nesta linha;
- Linha 2, coluna 3: o número 8 já está presente nesta coluna;
- Linha 3, coluna 3: o número 9 já está presente nesta região.

Perceba que uma implementação eficiente desconsidera os elementos anteriores a uma célula na verificação dos números repetidos, pois já foram verificados.