Introdução a linguagem Java

Prática integradora

Objetivo

O objetivo deste guia prático é que possamos consolidar e aprofundar os conceitos sobre coleções e estrutura de dados. Para isso, vamos propor um único exercício;

Preparados?

Método de Ordenação RadixSort

Desenvolva o método radixSort que ordena um vetor de inteiros mediante o algoritmo de ordenação: “Radix Sort”.

```
public class RadixSort {

    public static void radixSort(int iArr[])
    {
        //programar aqui
    }

    public static void main(String[] args)
    {
        int iArr[] = { 16223, 898, 13, 906, 235, 23, 9, 1532, 6388, 2511, 8 };

        radixSort(iArr);

        for (int i : iArr)
        {
            System.out.print(i + " ");
        }
    }
}
```

Como este algoritmo é relativamente completo, depois de analisá-lo, vamos dividir a tarefa em problemas bem definidos e mais delimitados, de forma que cada um seja,

por si só, um pequeno desafio.

Entendendo o algoritmo

Radix Sort é um algoritmo projetado para ordenar arrays de cadeias de caracteres, pois sua lógica se baseia em classificar os elementos do array de acordo com o valor da última posição. É importante conhecer este funcionamento, pois, se iremos ordenar um array de inteiros, primeiro temos que convertê-lo em um array de strings.

Naturalmente, podemos usar Radix Sort para ordenar arrays de qualquer tipo de dado, mas para evitar complicações que neste momento seriam irrelevantes, trabalharemos pensando em arrays de inteiros: int[].

Dito isso, o desenvolvimento do algoritmo consiste nas seguintes etapas:

1. Atribuir os valores de int inteirosArray[] para String stringsArray[].

2. Completar stringsArray[] com 0 (zeros) à esquerda (coincidindo com o número de dígitos do maior número de inteirosArray[]).

3. Criar 10 listas inicialmente vazias. L0 (será a lista para colocar os números que terminam com 0 (zero), em L1 colocaremos os números que terminam com 1, e assim por diante até L9.

4. Percorrer stringsArray[] para cada elemento, verificar qual é o seu último dígito e adicionar à lista L0, L1, L2, ... até L9 correspondente (conforme passo 3).

5. Percorrer as listas em ordem: L0, L1, L2, ... até L9 e carregar em uma estrutura de dados do tipo String os elementos de L0, depois os de L1 e assim por diante.

6. Semelhante ao passo 4, agora iremos trabalhar com o penúltimo dígito. Ou seja, L0 será a lista de elementos que possuem 0 em seu penúltimo dígito, L1 terá aqueles elementos que possuem 1 em seu penúltimo, e assim por diante.

7. Em cada iteração, avaliamos e classificamos os elementos de acordo com o seu último, penúltimo, antepenúltimo, (...) e assim por diante. Depois disso, o vetor será ordenado.

Exemplo do desenvolvimento do algoritmo

int a[] = { 3, 673, 106, 45, 2, 23 }

Convertemos e completamos com zeros a esquerda:

String s[] = {"003", "673", "106", "045", "002","023"}

Classificamos em listas de acordo com o último dígito:

L2 = {"002"}

L3 = {"003", "673", "023"}

L5 = {"045"}

L6 = {"106"}

Carregamos em ordem os elementos no vetor String[] s:

String[] s = {"002", "003", "673", "023", "045", "106"}

Agora repetimos considerando o penúltimo dígito.

L0 = {"002", "003", "106"}

L2 = {"023"}

L4 = {"045"}

L7 = {"673"}

Carregamos em ordem os elementos no vetor String[] s:

String[] s = {"002", "003", "106", "023", "045", "673"}

Agora repetimos considerando o antepenúltimo dígito.

L0 = {“002”, “003”, “023”, “045”}

L1 = {“106”}

L6 = {“673”}

Carregamos em ordem os elementos no vetor String[] s:

String[] s = {“002”, “003”, “023”, “045”, “106”, “673”}

Desenvolvimento do algoritmo

Levando em consideração cada uma das etapas do algoritmo, podemos dividir esse problema em vários problemas menores e mais fáceis de resolver.

1. Discuta (em grupo) sobre a(s) estrutura(s) de dados que possa ser usada no algoritmo Radix Sort.
2. Desenvolva o método radixSort() de acordo com a proposta do algoritmo.