SEMANAS:
INICIO/APRESENTAÇÃO
INTRODUÇÃO AOS ALGORITMOS E ESTRUTURAS DE DADOS
ESTRUTURAS DE CONTROLO DE FLUXO EM C#
CONCEITO DE CLASSE E OBJETO
SEPARAÇÃO ENTRE INTERFACE PÚBLICA E IMPLEMENTAÇÃO PRIVADA
ESTRUTURAS DE DADOS – INTRODUÇÃO
ANÁLISE DE ALGORITMOS E COMPLEXIDADE
ALGORITMOS DE ORDENAÇÃO
ALGORITMOS DE ORDENAÇÃO (PARTE 2)
PROVA DE AVALIAÇÃO
LISTAS E FILAS
TABELAS DE HASH
TRATAMENTO DE COLISÕES
ÁRVORES BINÁRIAS
TIPOS GENÉRICOS E TRATAMENTO DE EXCEPÇÕES
PROVA DE AVALIAÇÃO

Algoritmos e Estruturas de Dados

CTeSP - Tecnologias e Programação de Sistemas de Informação 2020/21

Docente: Ricardo Henriques.

Sumário:

Estruturas de dados - introdução

Nesta e nas próximas aulas iremos ver os fundamentos das principais estruturas de dados, nomeadamente:

- Vetor (*array*)
- Matriz
- Pilha (*stack*)
- Listas e Filas
- Árvore
- Tabela de dispersão (*hash table*)

Vamos começar por (re)apresentar os *arrays*, agora apresentado-os de modo mais formal e a utilizá-los como base para evoluir a nossa aplicação das *contas bancárias*.

Arrays (ou vectores)

Um *array* é uma sequência de valores todos do mesmo tipo. Para além de armazenar os valores é relevante aceder a cada seu valor individual. A *indexação* é o método de acesso a um elemento de um *array*. No caso do C#, um *array* com $\mathbb{N}$ elementos terá os seus valores disponíveis nos índices `0..ℕ-1`.

Para referenciar um qualquer elemento basta usar a notação `a[i]` para se referir ao índice correspondente valor nesse índice. É por isso considerado um *array unidimensional*.

Constituir um *array* em C# (e na generalidade das linguagens de programação) passa por realizar três passos distintos:

1. Declaração do tipo e do nome do *array*;
2. Criar o *array*;
3. Iniciar os valores do *array*.

Para declarar o *array* é necessário especificar o nome e o tipo de dados que irá conter. Para o criar é necessário indicar (reservar memória) com determinado tamanho (máximo número de elementos).

```
double[] numeros;
numeros = new double[20];
for (int i = 0; i < numeros.Length; i++) {
    numeros[i] = 1;
}
```

A forma apresentada mostra a modo de declaração, criação e iniciação de valores mais "longa". As linguagens de programação, normalmente, permitem escrever de forma mais compacta, exemplo:

```
double[] numeros = new double[20];
for (int i = 0; i < numeros.Length; i++) {
    numeros[i] = 1;
}
```

Ou ainda... se souber os valores:

```
double[] numeros2 = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
    14, 15, 16, 17, 18, 19, 20 };
```

A classe `System.Array.Base`

Dado o C# caraterizar-se por ser fortemente OO, cada *array* recebe (herda) grande parte da sua funcionalidade da classe `System.Array.Base`. A utilização destas propriedades/métodos permitem a utilização dos *arrays* com base no modelo de objetos.

Propriedade/Método	Descrição
Clear()	Elimina os valores do *array*, valores e referências.
CopyTo()	Copia dos elementos de um *array* para outro.
GetEnumerator()	Devolve a *interface* que permite iterar sobre o *array* nomeadamente através do contructor `foreach`.
Length	Devolve o número total de índices do *array*, mesmo para os *arrays* multi-dimensionais.
Rank	Devolve o número de dimensões de um *array*.
Reverse()	Ordena de forma inversa os conteúdos de um *array* multi-dimensional.
Sort()	Ordena os conteúdos de um *array* multi-dimensional. Se os elementos não forem de um tipo "primitivo" é no entanto possível definir um critério de comparação, implementando a *interface* `IComparer`.

Matrizes e cubos

Outra forma de *arrays* são os designados de matrizes, ou *arrays* de duas dimensões e no caso de três dimensões serão designados de cubos. A principal caraterística é que são vectores de mais de uma dimensão, mas sendo que as dimensões são `M x N` ou `M x N x W`.

O trecho de código abaixo mostra exemplos da sua utilização.

```
static void Main(string[] args) {
    string[,] nomes_e_moradas = new string[4, 2];
    double[,] quadradosEcubos = { { 2, 4, 8 }, { 3, 9, 27 }, { 4, 16, 64 } };

    nomes_e_moradas[0, 0] = "Júlio"; nomes_e_moradas[0, 1] = "Rua da Bela Vista";
    nomes_e_moradas[1, 0] = "Cristina"; nomes_e_moradas[1, 1] = "Rua das Maças";
    nomes_e_moradas[2, 0] = "Felizardo"; nomes_e_moradas[2, 1] = "Av. Fontes Pereira de Melo";
    nomes_e_moradas[3, 0] = "Machado & Irmão"; nomes_e_moradas[3, 1] = "Rua da Picaria";

    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 2; j++) {
            Console.WriteLine("[{0}, {1}] = {2}", i, j, nomes_e_moradas[i, j]);
        }
    }
    Console.WriteLine("Length = {0}", nomes_e_moradas.Length);
    Console.WriteLine("---------------");
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
                Console.WriteLine("[{0}, {1}] = {2}", i, j, quadradosEcubos[i, j]);
        }
    }
}
```

Matrizes de 4x2 e 3x3

Jagged arrays

O seguinte tipo de array multi-dimensional é designado por *jagged array*. Como o nome implica ele conterá algum número de arrays internos, cada um dos quais poderá ter o seu próprio limite (o que diferencia do conceito de matriz).

O seguinte exemplo mostra a forma como este é declarado e usado. Repare-se em particular que todo a síntaxe de acesso a cada posição do *array* é feita pela utilização de parênteses rectos individuais.

```
#region jagged array
Console.WriteLine(System.Environment.NewLine + "jagged array, exemplo:");
// jagged arrays - exemplo
int[][] jaggedArray = new int[2][];
Console.WriteLine("tamanho = {0}",  jaggedArray.Length);
for (int k = 0; k < jaggedArray.Length; k++) {
    jaggedArray[k] = new int[5 * (k + 1)];
    Console.WriteLine("tamanho de {0} é {1}", k, jaggedArray[k].Length);
    for (int i = 0; i < jaggedArray[k].Length; i++) {
        jaggedArray[k][i] = (1 + k) * (i + 1);
    }
}
Console.WriteLine("Listagem:");
for (int i = 0; i < jaggedArray.Length; i++) {
    for (int j = 0; j < jaggedArray[i].Length; j++)
        Console.Write(jaggedArray[i][j] + "\t"); // tab
    Console.WriteLine();
}
#endregion
```

Jagged array

Implementação da classe `Banco` como um *array*

Descarregue aqui a solução.

MRH, 2020/21