Acadêmico: João Victor Surdi de Melo

**CSharp data structures and algorithms**

**Capítulo II**

**Arrays:**

* A estrutura de array pode ser usada para guardar uma ou mais variáveis do mesmo tipo primitivo, essas como INT ( arrayInt = [1, 2, 3]) ou então STRING (arrayString = [‘a’, ‘b’, ‘c’]) em uma classe definida pelo usuário...
* No C#, tem-se o benefício de uma quantidade vasta de variantes de array as apresentadas no livro são exemplos de unidimensionais(a), multidimensionais (b) e irregulares (c) todas foram apresentadas no diagrama do texto pág. 34...
* Um fato importante sobre Arrays é: após uma array ser declarada, ela não pode ser alterada, pelo menos não facilmente como uma lista genérica... Para uma aplicação em que precise ser alterado, array não é uma opção tão boa quanto lista.

Arrays Unidimensionais:

* Por ser uma array, pode acumular vários itens de um mesmo tipo, mas, as unidimensionais trabalham com Index, ou seja, o vetor ou “casa” onde se encontram, importante lembrar que esses índices são conhecidos como “Zero-based”, entretanto o primeiro vetor sempre vai ser ‘0’ (Exemplo = [0, 1, 2, 3, 4, 5] -> Array com 6 itens) e o útimo sempre será o tamanho da array –1...
* Para usar uma array unidimensional, você precisa declará-la, a declaração é bem simples, primeiro se identifica o tipo do elemento e o nome:

Tipo[] nome;

Int[] numbers;

* A declaração é necessária para que se possa inicializar uma array, a inicialização também é bem simples:

numbers = new int [5];

* É possível também declarar e inicializar a array na mesma linha:

Int[] numbers = new int[5];

* Assim que declarar uma array, ela terá um valor padrão, esse é o número 0 no caso de uma array com int, caso precise alterar um valor, terá de usar os colchetes [] para indicar a index do elemento:

numbers [0] = 9;

numbers [1] = -11;

numbers [4] = 1;

Resultado: numbers -> [9, -11, 0, 0, 1, 0]

* Você também consegue declarar e iniciar uma array, já definindo os elementos também, na mesma linha:

Int [] numbers = new int [] {9, -11, 6, -12, 1};

int [] numbers = {9, -11, 6, -12, 1};

* Quando se tem uma array com valores já definidos, você pode consultar esses valores usando a seguinte linha de código:

Int middle = numbers[2];

* Nesse caso, eu estou consultando o valor que está na terceira posição, index 2 ele deve me retornar o número ‘6’.
* Exemplo “Month Names” feito no vscode e estará junto com esse resumo no repositório

Arrays Multidimensionais:

* As arrays no C# não se limitam a apenas uma dimensão, também é possível criar arrays com duas ou três dimensões, para criar uma dimensão a mais na array, é necessário adicionar um elemento a mais na inicialização da array:

Int [, ] numbers = new int [5, 2]; -> 2 dimensões

Int [, , ] numbers = new int [5, 4, 3]; -> 3 dimensões

Claro, você pode também combinar a declaração com a inicialização da array:

Int [,] numbers = new int [,] =

{

{ 9, 5, -9},

{ -11, 4, 0},

{ 6, 115, 3},

{ -12, -9, 71},

{ 1, -6, -1}

}

* Para consultar os elementos de uma array multidimensional, é preciso consultar com colchetes a dimensão e o índice do elemento:

Int number = numbers [2] [1]; -> Resposta: 115

* Para imputar um valor, seguimos a mesma lógica:

Numbers [1] [0] = 11;

* Exemplos “Multiplication Table” e “Game map” feitos no vsode e estarão disponíveis no repositório GitHub da disciplina...

Arrays Irregulares:

* A última variante de Array descrita no livro, é a array irregular, ela é chamada popularmente de “Array de arrays”, soa complicado, mas é bem simples, ela é como uma array unidimensional, pode ser entendida com a mesma lógica, essa é uma array onde cada elemento também é uma array, porém, as arrays “Contidas” não podem ter um tamanho próprio, somente a cadeia de arrays...
* A declaração e início de uma array irregular é diferente das descritas acima:

Int [] [] numbers = new int [4] [];

Numbers [0] = new int[] {9, 5, -9};

Numbers [1] = new int[] {0, -3, 12, 51, -3};

Numbers [3] = new int[] {54};

* Se for notar, a escrita é parecida com a unidimensional com 4 elementos, porém, os elementos são “Convertidos” em dimensões, e cada uma com suas próprias arrays, menos a terceira, que não tem valor nenhum atribuído, ou seja, será 0 ou Nulo...
* Também podemos escrever dessa maneira:

Int [] [] numbers =

{

New int[] {9, 5, -9},

New int[] {0, -3, 12, 51, -3}.

NULL,

New int[] {54}

};

* Método de consulta/alteração continua basicamente igual:

Int number = numbers [1] [2]; -> Reposta: 12

Number [1] [3] = 50; -> 51 irá mudar para o número 50

* Exemplo “Yearly transport plan” feito no vscode e estará disponível no repositório GitHub da matéria

**Ordenação de Algoritmos:**

* Vários algoritmos trabalham com arrays e manipulam elas, uma das tarefas mais utilizadas quando se fala em array é a ordenação, manipulação dos elementos da array...
* Temos alguns exemplos de ordenação no livro, como a ordenação por seleção, por inserção, por “bolhas” e a rápida...

“Selection Sort”:

O Selection Sort é um algoritmo de ordenação simples e intuitivo que funciona selecionando repetidamente o menor elemento da lista não ordenada e movendo-o para o início da lista ordenada. Esse processo é repetido até que todos os elementos estejam na posição correta

- Divide a lista em duas partes: uma parte ordenada e outra não ordenada.

- Inicialmente, a parte ordenada está vazia.

- Em cada passo, encontra o elemento mínimo na parte não ordenada.

- Move o elemento mínimo para a última posição da parte ordenada, expandindo assim a parte ordenada.

- Repete os passos 3 e 4 até que toda a lista esteja ordenada

“Bubble Sort”:

O Bubble Sort é um algoritmo de ordenação simples que funciona comparando pares de elementos adjacentes em uma lista e trocando-os se estiverem fora de ordem. Esse processo é repetido várias vezes até que toda a lista esteja ordenada.

- Começa comparando o primeiro elemento com o segundo e, se estiverem fora de ordem, os troca.

- Em seguida, compara o segundo elemento com o terceiro e repete o processo até o final da lista.

- Esse processo é uma passagem completa pela lista, durante a qual o maior elemento "borbulha" para a posição final.

- Após a primeira passagem, o maior elemento está na posição correta na lista ordenada.

- Repete os passos 1 a 4 para o restante da lista não ordenada, excluindo o elemento já colocado na posição correta

- Continua esse processo até que todos os elementos estejam na posição correta na lista ordenada.

“QuickSort”:

O Quicksort é um dos algoritmos de ordenação mais eficientes e amplamente usados. Ele pertence à categoria de algoritmos de "dividir para conquistar" e é conhecido por sua eficiência média, com um desempenho médio de tempo de execução

- Escolhe um elemento na lista como o "pivô". O pivô desempenha um papel crucial no algoritmo, pois os elementos são rearranjados em relação ao pivô.

- Divide a lista em duas partes: uma parte com elementos menores que o pivô e outra com elementos maiores que o pivô.

- Recursivamente aplica o Quicksort a ambas as partes da lista.

- Combina as partes ordenadas, juntando o pivô e as duas partes ordenadas (menores e maiores).

- Repete essas etapas até que toda a lista esteja ordenada.

Listas Simples:

Listas são basicamente uma Array em que pode-se alterar o tamanho e valores conforme a necessidade, apesar de arrays serem úteis, listas podem entrar nesses casos onde elas se tornam complicadas de se trabalhar

Array List:

- Um ArrayList é uma estrutura de dados em linguagens de programação como Java, C#, Python (por meio da biblioteca NumPy) e outras. Ele é uma implementação de uma lista dinâmica, que permite armazenar e manipular uma coleção de elementos de maneira flexível.

Generic list:

- Uma lista genérica, frequentemente chamada de "Generic List", é uma estrutura de dados que permite armazenar elementos de um tipo específico de maneira flexível, enquanto também fornece segurança de tipo durante a compilação. As listas genéricas são comuns em linguagens de programação modernas, como C# e C++, e têm várias vantagens sobre estruturas de dados não genéricas.

Sorted List:

- Uma Sorted List, em geral, refere-se a uma lista de elementos que é automaticamente mantida em ordem crescente ou decrescente com base em algum critério de classificação definido. A ordenação pode ser feita em relação aos valores dos elementos ou com base em uma chave associada a cada elemento. No entanto, a implementação específica e o comportamento de uma Sorted List podem variar dependendo da linguagem de programação ou da estrutura de dados usada.

Linked Lists:

- Uma linked list (ou lista encadeada) é uma estrutura de dados linear que consiste em uma sequência de elementos, chamados de nós (nodes), onde cada nó armazena um valor (ou dado) e uma referência (ou ponteiro) para o próximo nó na sequência. As linked lists são uma das estruturas de dados fundamentais em ciência da computação e são usadas para representar coleções de elementos de forma dinâmica.

Circular linked lists:

- Uma Circular Linked List (ou lista encadeada circular) é uma variação da lista encadeada na qual o último nó da lista não aponta para null, como acontece nas listas encadeadas simples, mas sim para o primeiro nó da lista. Isso cria um ciclo, formando uma estrutura de loop, onde a sequência de nós forma um ciclo fechado.