06

SortedDictionary

Transcrição

Neste vídeo, trabalharemos com um novo projeto *console application*.

Para começar, faremos uma cópia do nosso projeto que tratou da `SortedList`.

```
IDictionary<string, Aluno> sorted
            = new SortedList<string, Aluno>();

            sorted.Add("VT", new Aluno("Vanessa", 34672));
            sorted.Add("AL", new Aluno("Ana", 5617));
            sorted.Add("RN", new Aluno("Rafael", 17645));
            sorted.Add("WM", new Aluno("Wanderson", 11287));

            foreach (var item in sorted)
            {
                    Console.WriteLine(item);
            }
```

Faremos uso do mesmo código, utilizando a classe `Aluno` e a implementação do `SortedList`.

Pressionando as teclas "Ctrl + F5", executaremos o programa.

Vemos que ele continua funcionando, da mesma forma como anteriormente, inclusive respeitando a ordem alfabética.

Faremos a seguir um novo tipo de coleção, que também será um dicionário ordenado. Ela se chamará `SortedDictionary`.

Portanto, substituiremos pelo novo nome.

```
IDictionary<string, Aluno> sorted
            = new SortedList<string, Aluno>();
```

Com isso, executaremos o programa novamente, utilizando o atalho "Ctrl + F5".

Percebemos que, somente com a alteração do nome, o resultado é idêntico ao que obtivemos anteriormente.

Estes tipos de listas são diferentes porque sua implementação interna não é igual.

Uma `SortedList` tem, internamente, duas listas. Uma de chaves, e outra de valores, que estarão sempre ordenadas. Neste sistemas, conforme formos inserindo novos elementos na lista, eles serão automaticamente ordenados.

Abriremos o `SortedDictionary`, para comparação. Com o cursor sobre o seu nome, utilizaremos o atalho "Alt + F12".

Por ser uma classe genérica, temos o tipo da chave e o tipo do valor.

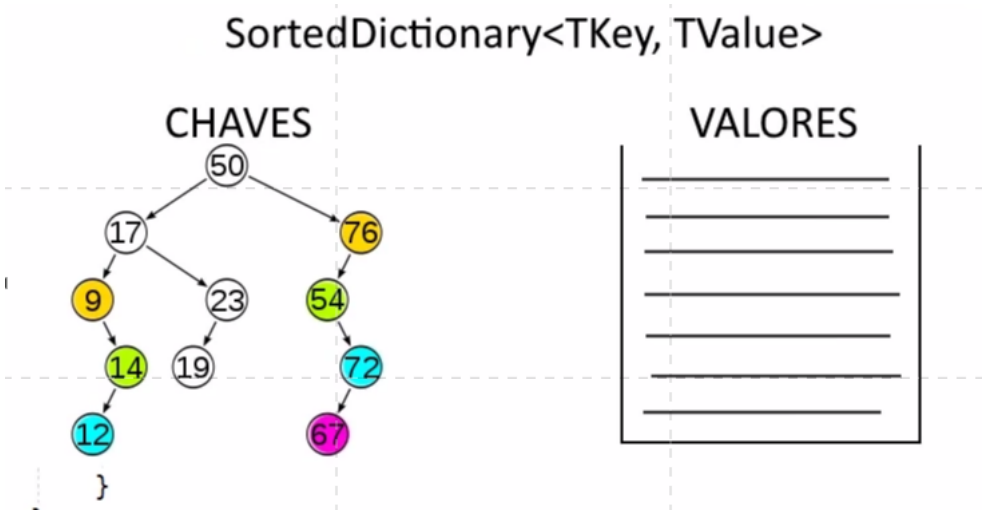
```
class SortedDictionary<TKey, TValue>
```

Ele também implementa um `IDictionary`, e terá sua coleção de chaves e de valores, como todo dicionário.

A declaração das chaves é feita com a propriedade `Key`, só que aqui ela será uma `KeyCollection`, diferente do que aconteceu na `SortedList`.

Já para a propriedade de valores, utilizaremos uma `ValueCollection`, que também difere do `SortedList`, onde tínhamos um `IList`.

Na prática, isto funcionará da seguinte forma:



Temos duas partes, uma em que armazenamos as chaves, e outra onde serão armazenados os valores.

Na hipótese de um `SortedDictionary` onde a chave é de um tipo inteiro, temos uma estrutura de árvore.

Cada um dos nós aponta para um ou dois valores, ou, ainda, para nenhum valor.

Cada elemento tem, no máximo, duas setas. Sendo que, quando aponta para esquerda, está indicando o elemento que veio antes dele. Com isso, é possível estabelecer uma ordem.

O indicador direito mostra o elemento posterior.

No diagrama acima, temos a raiz 50, indicando à direita o número 76, ou seja, posterior a ele.

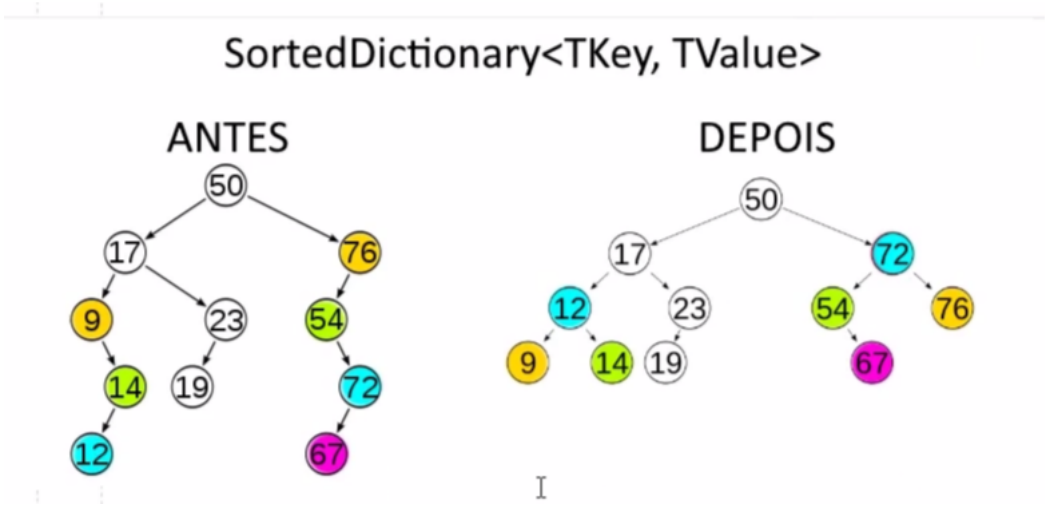
O 76, por sua vez, aponta com a seta esquerda para o número 54, indicando que ele vem antes.

Isto estabelece a ordem e, é dessa maneira, que o `SortedDictionary` mantem seus elementos organizados.

Por outro lado, as chaves têm uma correspondência em outra coleção, que é a de valores.

Este chaveamento é chamado de "Árvore Binária", porque ela sempre conta com dois ponteiros, um para o anterior e outro para o posterior.

Esta árvore conta com um algorítimo que a balanceará, principalmente quando contar com muitos itens, vide o diagrama a seguir:



Na primeira árvore binária, os elementos foram adicionados e ela foi adotando um formato comprido, ao invés de largo.

Dessa forma, se quisermos buscar, por exemplo, o número 67, a busca começará pelo 50, já que sempre começa pela raiz. Assim, serão muitos passos até chegarmos no resultado desejado.

Com o tempo, a árvore será reajustada para que fique mais larga do que comprida.

Como podemos ver no "Depois", o número 67 já subiu de nível. Isso faz com que a navegação seja mais curta.

As estruturas de `SortedList` e `SortedDictionary` são bastante diferentes, mas servem a um mesmo objetivo.

A diferença é que, o método do dicionário é mais ágil caso seja necessário inserir ou remover elementos com muita frequência. Isto se deve ao seu formato de árvore.

Na `SortedList`, que trabalha com um `IList`, ao removermos um elemento, ela precisa se reajustar, para poder ocupar o vazio deixado. Da mesma forma, ao adicionarmos um elemento, ela aumentará de tamanho e realocará seus itens.

Por isso, uma inserção ou remoção muito frequente de elementos, neste tipo de lista, é menos eficiente do que num `SortedDictionary`.

Entretanto, caso o objetivo seja criar uma coleção em que todos os elementos iniciais já estão definidos, o mais indicado seria a `SortedList`.

Estas diferenças são relevantes apenas para coleções muito grandes e, como hoje trabalhamos com altas capacidades de processamento, elas acabam não sendo tão gritantes.

Em caso de dúvida, recomenda-se iniciar com um `SortedDictionary`.