Seřazené a neseřazené datové struktury

Seřazené

- SortedSet
- SortedDictionary – pouze key
- SortedList – pouze key

Objekty, které jsou ukládány do SortedSetu nebo jako key do SortedDictionary a SortedListu musí implementovat rozhraní IComparable a metodu compareTo.

```
class Nevim : IComparable<Nevim>
{
    public int Cislo;
    Počet odkazů: 5
    public Nevim(int cislo)
    {
        this.Cislo = cislo;
    }
    Počet odkazů: 0
    public int CompareTo(Nevim other)
    {
        Nevim d = (Nevim)other;
        return this.Cislo.CompareTo(d.Cislo);
    }
}
```

Nebo mít vytvořený IComparer pro tyto objekty a ten dát dané kolekci.

```
class NevimComparer : IComparer<Nevim>
{
    Počet odkazů: 0
    public int Compare(Nevim x, Nevim y)
    {
        if (x.Cislo > y.Cislo)
            return 1;
        if (x.Cislo < y.Cislo)
            return -1;
        else
            return 0;
    }
}
```

```
SortedSet<Nevim> ss = new SortedSet<Nevim>(new NevimComparer());
```

SortedSet

Generický

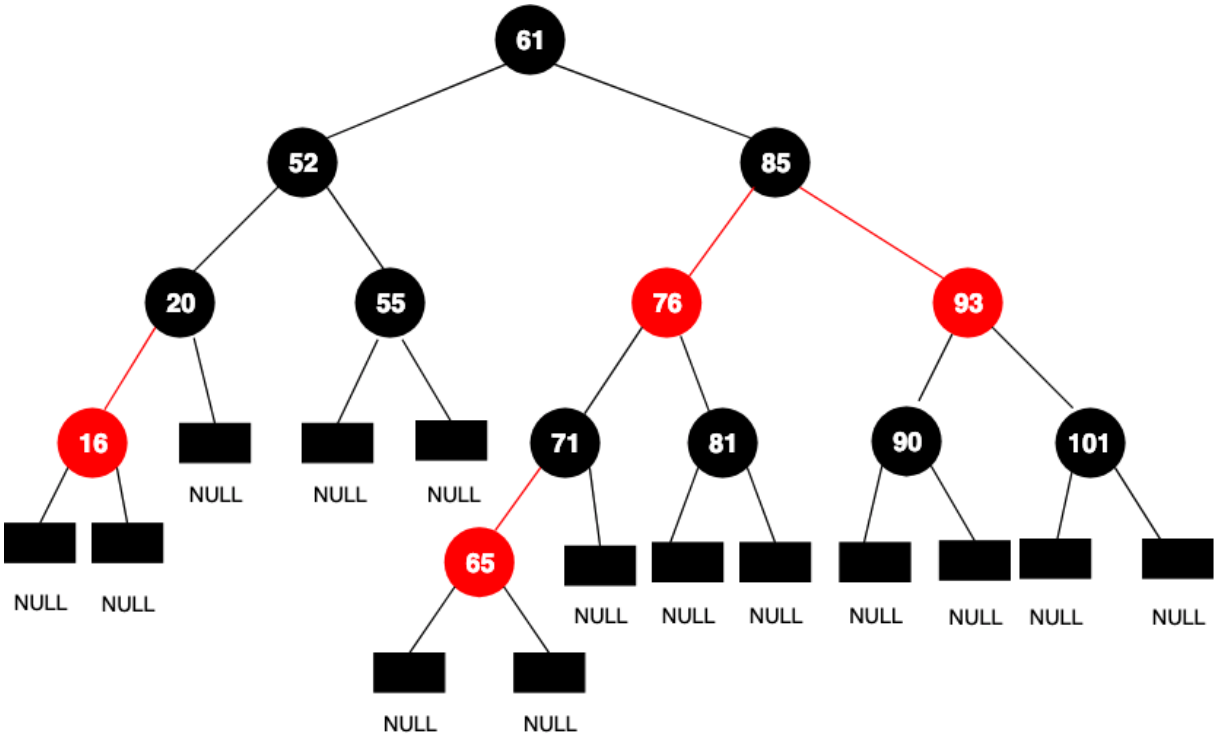
Struktura: Red-Black tree

Je podobný Balanced Binary Search Tree, takže dělá rotace (root nezůstává stejný)

Red-Black tree - je binární strom s jedním dvouhodnotovým příznakem v uzlu navíc. Tento příznak představuje barvu uzlu, která může být červená nebo černá. Zajišťuje, že žádná cesta z kořene do libovolného listu stromu nebude dvakrát delší než kterákoli jiná, to znamená, že strom je přibližně vyvážený.

Musí splňovat

- Každý uzel je buď černý nebo červený.
- Každý list (NULL) je černý.
- Jestliže je daný uzel červený, pak jeho potomci jsou černí.
- Každá cesta z libovolného uzlu do listu obsahuje stejný počet černých uzlů.



Tento strom nepodporuje duplicitní prvky. Ale nevyhodí exception při pokusu vložit prvek, který je ve stromu.

Add() – O(log n)

Remove() - O(log n)

$\log_2$ (n) protože při přidávání, hledání neprojde celou kolekci, hled na rootu jednu větev, tím, že se vydá buď do prava, nebo do leva a tak se to opakuje

Nemá kapacitní omezení ani se jeho struktura nezvětšuje, protože prvky podobně jako linkedlist na sebe odkazují.

SortedDictionary

Generická

Skoro stejný jako SortedSet, jen má seřazený key a value ne.

Je to jako SortedSet pro key, kterému vadí pokus o přidání key, který je v kolekci a vyhodí exception.

Key nemůže být null. Value může být null pokud jde o referenční datový typ.

Rozdíl oproti SortedSetu je, že prvky v kolekci obsahují atribut value.

Jinak je stejný jako SortedSet, časové složitosti, struktura.

SortedList a SortedList<Key,Value> negenerický a generický typ

Hlavní rozdíl: Negenerický umožňuje získání prvků pomocí indexů

```
SortedList sl = new SortedList();
SortedList<int, string> sl2 = new SortedList<int, string>();
sl.Add(51,"nongeneric");
sl2.Add(12,"generic");
Console.WriteLine(sl.GetByIndex(0));
Console.WriteLine(sl2[12]);
```

Struktura: 2 dynamická pole, jedno pro klíče, druhé pro hodnoty

Zachované pořadí: ne

Označení místa uložení: key

Dvě stejné hodnoty: klíč musí být unikát, value může

Add() – O(n), O(log n) přidán na konec listu

Remove() - O(n)

Dictionary[key] - O(log n)

ConstainValue() – O(n)

ConstainKey() – O(log n)

SorterList

Negenerický

Výhoda je následující: key musí zůstat stejného datového typu, ale value nemusí

Byl vytvořen ve verzi 1.1 a je tu pořád kvůli zpětné kompatibilitě

Výhoda je možnost získávání hodnot pomocí indexu

GetByIndex – O(1)

Metody: Clear(), Clone(),GetByIndex(int), IndexOfValue(int), IndexOfKey(int), RemoveAt(int), SetByIndex(int, value)

Podtržený jsou metody pracující s indexem

Vlastnosti: Capacity - Získá nebo nastaví kapacitu SortedList objektu.

SortedList<Key,Value>

Generický

SortedList<Key,Value> používá méně paměti než SortedDictionary<Key,Value> a je rychlejší při procházení všech prvků

Metody: Add(key,value), Clear(), ContainsKey(), IndexOfKey(key), IndexOfValue(value), RemoveAt(int)

Neseřazené

Ostatní… některé z nich se mohou stát seřazenými pomocí implementovaných metod .Sort() jako jsou kolekce založené na poli. Může být napsán i algoritmus pro třídění kolekcí jako LinkedList.