C# Radix Sorted Collection

A radix sorted collection, first and foremost, requires that elements be segmental. That is, the elements inserted into the collection will have their structure defined such as to efficiently store them in the collection.

Generic Type Information

It would be the best to require that the user specify only the type of the elements intended to be stored in the collection and no other. If this type is required by the implementation and class definition to be an IEnumerable the type information of the pieces that the elements break down into should be available. It might be easy to specify an out-type parameter.

Class Syntax

public class RadixSortedCollection<T, out TPiece> : ICollection<T>,

IEnumerable<T>, IEnumerable, ICollection,

IReadOnlyCollection<T>

where TPiece : IComparable

The class is first and foremost an ICollection so we ensure this is the first inheritance. Next, we want to ensure this collection is iterable, so we explicitly inherit from IEnumerable. The IReadOnlyCollection inheritance enables the collection to be cast into an immutable collection, which is handy for permanent lookup tables after construction.

Lastly, the out TPiece type parameter is set to be IComparable for inherent structure of the radix tree. Note that the comparability of the TPieces can be overridden by an IComparer<TPieces> passed to the constructor.

Constructor Options

We should have useful and standard collection constructor overloads.

* ()
* (IEnumerable<T> otherCollection)
* (ICollection<T> otherCollection)
* (IComparer<TPiece> comparer)
* (IEnumerable<T> otherCollection, IComparer<TPiece> comparer)
* (ICollection<T> otherCollection, IComparer<TPiece> comparer)
* (RadixSortedCollection<T> other)

The last copy constructor is important because of the simplicity of copying over trees and efficiency of copying over the structure.