Introducing FastSearch, a very fast string search for objects and lookups

Welcome to our first article regarding fast in-memory search of a list of objects using **FastSearch**, a .NET class library for fast string-based search brought to you by the team at MILL5.

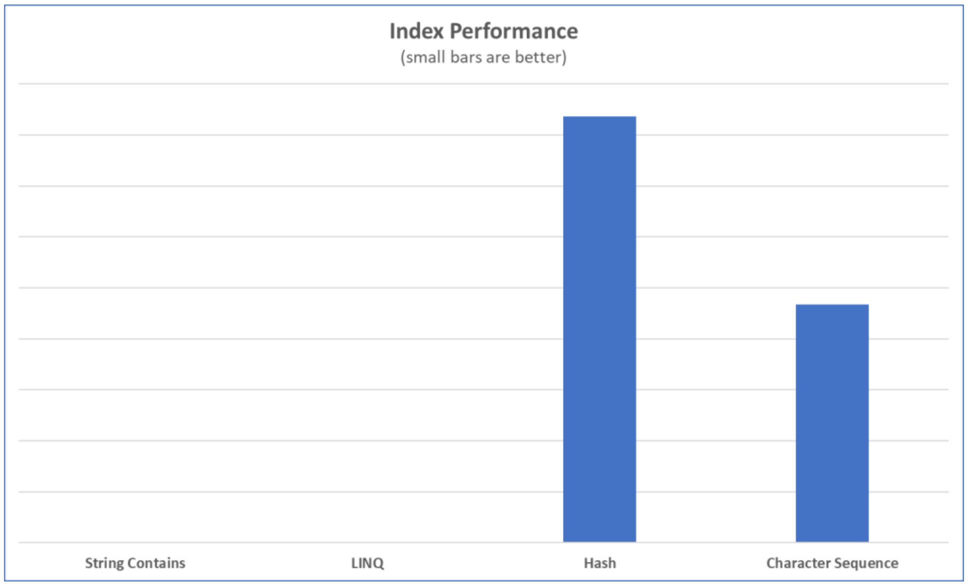
The motivation behind this library is simple, we want very fast search of a large list of objects based on strings so that we can display the results to users. Of course, there are many ways search can be done in .NET by writing very little code. One way is to loop through a list of objects using the String class to search string properties of your objects. Unfortunately, this type of search is not very fast and gets worse the larger the list gets.

A variation on using the String class is to use LINQ. It is extremely easy to write a small amount of LINQ code which queries a list of objects. The amazing part about this approach is how simple this is and how much .NET developers rely on LINQ queries every day. Unfortunately, there is a lot of LINQ code that is the source of performance problems everywhere.

String Contains and LINQ queries are not optimized and require some help to get better search performance. We will see the different optimizations we do per algorithm and compare their performance. Here is the list of algorithms we will be comparing.

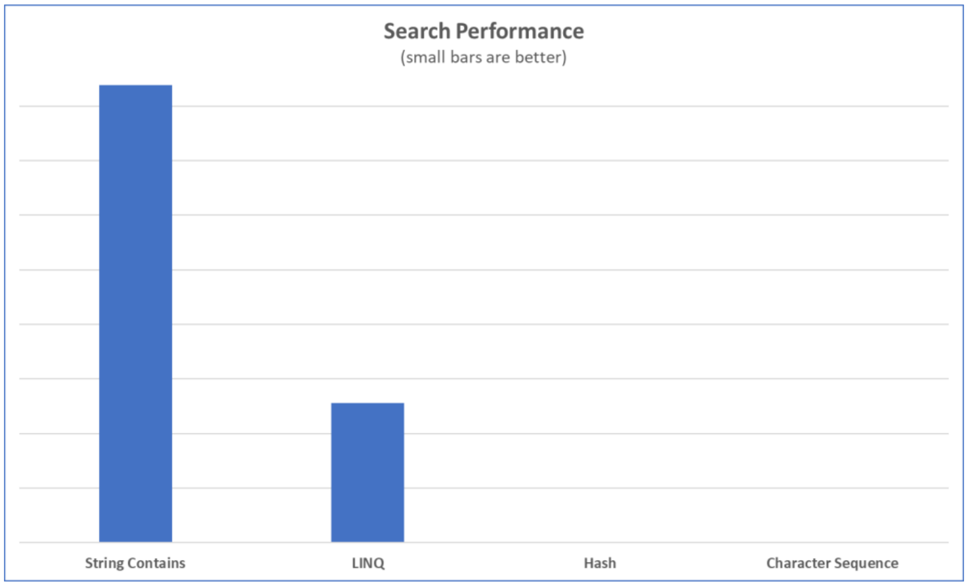
Algorithm	Description
String Contains	Search a list using String.Contains method
LINQ	Search a list using a LINQ query
Hash	Search a list by precomputing hashes for all possible search patterns
Character Sequence	Search a list using a precomputed index structure which maintains a character sequence tree of all possible search pattens

Each of these algorithms will have different performance characteristics for indexing and searching. Let us look at the index performance for each algorithm.



Notice that the Hash and Character Sequence algorithms takes significantly longer to index than the String Contains and LINQ algorithms. That is because very little is done to build an index or precompute values for searching. Fortunately for the Hash and Character Sequence algorithms we assume that building indexes happen once or so infrequently that our users will not be impacted by the overhead of the indexing process. Of course, when we do have to update our indexes, we do so in the background so that the performance impact is not observed by our users. Once the indexes are rebuilt, we swap out the old indexes for the new indexes.

Let us turn our attention to search performance. Notice that the String Contains and LINQ algorithms take a significantly long time to search. That is because we have not done much to improve the performance on these algorithms. Instead, we put the performance optimizations into the Hash and Character Sequence algorithms.



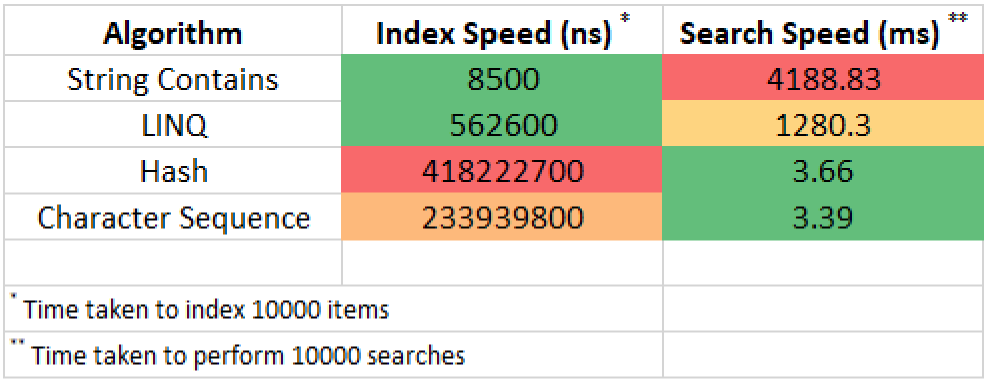
Notice that searching using the Hash and Character Sequence algorithms is very fast compared to String Contains and LINQ. Most of the optimization is due to the data structure used for the index. We do perform other optimizations like precomputing case insensitive strings, binary searching, and parallelism, but these optimizations pale in comparison to the performance gains by using efficient data structures.

	String Contains	LINQ	Hash	Character Sequence
Case Insensitive		X	X	X
Binary Search			X	
Parallel Indexing			X	X
Hashing			X	
Data Structure			X	X
Parallel Search		X	X	

The Hash algorithm uses a map based on precompute hash of all possible search patterns within the list of objects. When a search is performed, the hash of the search pattern is computed and used to find all matching objects within the map.

The Character Sequence algorithm builds a tree of all possible character sequences. When a search is performed, the search pattern is broken into its character sequence. The search is performed by walking the tree to find all matching objects that contain that sequence.

The Hash and Character Sequence algorithms offer significant performance improvements over the String Contains and LINQ algorithms. The clear winner though is the Character Sequence algorithm. This is due to having the fastest search performance and better index performance that the Hash algorithm. The Character Sequence algorithm is also a very compact data structure and is very efficient and uses very little memory.



Algorithm	Index Speed (ns) *	Search Speed (ms) **
String Contains	8500	4188.83
LINQ	562600	1280.3
Hash	418222700	3.66
Character Sequence	233939800	3.39
* Time taken to index 10000 items		
** Time taken to perform 10000 searches		

In a future article, we will go into the data structures used for each of these algorithms. In addition, we will be improving this library over time to offer better and faster searching. If you want to use the **FastSearch** library, add it to your .NET project from NuGet at https://github.com/MILL5/FastSearch.

Enjoy using **FastSearch** for your own needs.

Author(s):
Richard Crane, Founder/CTO

Special Thanks:

Steve Tarmey, Principal Architect

James Pansarasa, Chief Architect

References:

Fast Search – NuGet

https://www.nuget.org/packages/FastSearch/

FastSearch – GitHub

https://github.com/MILL5/FastSearch

Rabin–Karp algorithm

https://en.wikipedia.org/wiki/Rabin%E2%80%93Karp_algorithm

Boyer–Moore–Horspool algorithm

https://en.wikipedia.org/wiki/Boyer%E2%80%93Moore%E2%80%93Horspool_algorithm