

Re: Need quick lookup like Hashtable, but don't need to store value

Re: Need quick lookup like Hashtable, but don't need to store value

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.framework/2006-08/msg00656.html>

- *From:* "Carl Daniel [VC++ MVP]" <cpdaniel_remove_this_and_nospam@xxxxxxxxxxxxxxxxxxx>
 - *Date:* Wed, 16 Aug 2006 12:24:59 -0700
-

<illegal.prime@xxxxxxxxxx> wrote in message
news:1155753807.933156.173590@xx

This is a fundamental concept in Computer Science. Searching a sorted list using binary search is $O(\log n)$.
The above log is base 2.

The complexity to get an entry from a hashtable that doesn't have collisions is $O(1)$.

In other words, I can't think of any reason to use a sorted list over a hashtable – regardless of the language (C# or otherwise).

Big-O notation tells you the relative performance in terms of number of operations when the number of operations is large (approaches infinity).

Unfortunately, it's easy to be lead astray by worst-case theoretical performance in real-world cases. There are a couple of factors that play heavily into the equation:

1. The size of your collections.
2. The cost of the individual operations.

In the case of sorted array versus hashtable, you have to consider the cost of comparison to the cost of hashing. The relative cost of those operations is a function of the size and content of the items being compared and hashed.

For example, imagine you have a container with 100, 100-character strings.

For a hashed container, determining if a given 100-character string is in that container will require touching all 100 characters of the target string in order to compute its hash. The hash is then used as an index into a table of "buckets" to find the item. If the collection contains items with the same bucket number (hash collisions), the hashtable generally degenerates to a linear search, doing full-value comparisons of the target

Re: Need quick lookup like Hashtable, but don't need to store value

Re: Need quick lookup like Hashtable, but don't need to store value

string to each item in the bucket. If there are no hash collisions, then this cost can be ignored, and the cost of a lookup is roughly constant and roughly equal to the cost of calculating that one hash value (100 character accesses).

For a sorted array, finding a string using a binary search will take at most 7 comparisons, with each comparison taking between 2 and 200 character accesses, depending on how different the target string is from the ones already in the container. Clearly, in the worst case (nearly identical strings differing only at high index values), the hashtable will vastly outperform the sorted array, but for many real-world applications, the string comparison will drop out after only a handful of comparisons, and the cost of 7 string comparisons will actually be less than the cost of calculating a single hash value.

Bottom line: you have to know your content and how it's accessed to know which is best. Researchers have shown that for small collections, the sorted list is usually faster even though the hashtable has a better Big-O figure. The only way to know for sure in your application is to measure.

-cd

.