

Forum:[Hack Forums](#)

Thread: [Linear search in different collections, Practical runtime analysis\[The Society\]](#)

Mefie - January 7, 2017, 12:09 pm

Hello!

In this guide, I am showing the pseudo-code of linear search in unsorted array, sorted array and linked list. My purpose with this demonstration is to show how different collections accomplish with the simplest search algorithm we know and also showing a bit of practical runtime analysis to you. How is it important for you? In business, time is a decisive factor. Usually, the question is not whether you can accomplish something, but whether you can do it faster than your competition. So, runtime analysis is important and easy examples are the best way to understand the basics of it.

Linear search

Short description: With linear search, we can locate an element with the given attribute (e.g.: value) if there is any.

Linear search in unsorted arrays:

Code Function `linearssearch_unsorted(x – T array, n – integer, P – logical function)`

```
i <- 0
While i <= n ^ !P(x[i]) loop
i <- i + 1
end loop
exist <- (i <= n)
if exist then
return (exist, i)
else
return exist
end if
End function
```

When the collection is unsorted and there is not an element that makes our logical function true, we have to do a checking on the collection elements' from the first to the last, one by one. Certainly, this is the worst case from run-time view. In average case, an element that makes the function true is somewhere in the middle of the collection. In best case, the first element makes the function return true. Taking the average and worst cases into consideration, we can tell this algorithm has a runtime that is in linear relationship with the collection's size.

Linear search in sorted arrays:

Code function `linearssearch_sorted(x -T array, n - integer, value_to_search - T (where two T elements are comparable))`

```
i <- 1
while i <= n ^ x[i] < value_to_search loop
i <- i + 1
```

```

end loop
exist <- (i <= n ^ x[i] = value_to_search)
if exist then
return (exist, i)
else
return exist
end if
end function

```

When our collection is sorted, we can add an additional condition into the while loop from the previous code because when the checked value is greater than the value_to_search then we can be sure that the value_to_search is not in the collection. In best case the first element has the value_to_search or the first element is greater than the value_to_search. We can see if the array is sorted, there are more "best cases" than in the previous example and stopping the loop where there is a greater value than the value_to_search saves us runtime. This especially is helpful when we are searching a relatively small value in a collection of relatively great values. However, it does not change the relationship with the collection's size. So, I can suggest this solution when the collection is sorted already, you are searching a relatively small value and you do not want to understand the code of a more complex search function (e.g.: in scripting, cowboy-coding, your IT test in school...etc).

Linear search in linked list

```

Code function linearsearch_linkedlist(head - Node, F - logical function)
p <- head
while p != null ^ !F(p.content) loop
p <- p.next
end loop
exist <- P != null
if exist then
return (exist, p.content)
else
return exist
end if
end function

```

When implementing linear search in a linked list, we often do not know how the list is linked because a programmer's practice is to use linked list when the need of a frequently changing collection. So, the algorithm is very similar in runtime to the runtime of linear search in an unsorted array, so linear relationship. However, in the while condition we are writing `p != null`, which is a null check condition. A null check condition is usually quicker than comparing two objects so I would say linear search in a linked list is a bit quicker than searching in unsorted arrays.

Summation

Linear search is a basic search algorithm. In general, I can say that there are MUCH quicker collection-specific algorithms for almost any collection type. HOWEVER, linear search is your EASY algorithm that even your little sister can understand and an algorithm that you will remember even after 10 years. So, there are cases when linear search is simply the simplest solution from coding view. An example is writing a script where you want to write it as fast as possible and run it as soon as possible. To sum up our experience in linear search in the mentioned three, different collections I can say that using a linked list than an unsorted array has better runtime while a sorted array has better runtime than an unsorted linked list, with linear search. However, the differences are not that great that would suggest sorting your array just because of it.

Annabelle Lee - January 7, 2017, 7:41 pm

How fast can these sort compared to other algorithms? How fast do they work on semi-sorted/fully unsorted arrays? O time complexity?

Mefie - January 8, 2017, 2:32 am

(Jan 7, 2017 08:41 PM)AnnabelleLee Wrote: [\[CITE\]](#) How fast can these sort compared to other algorithms? How fast do they work on semi-sorted/fully unsorted arrays? O time complexity?

Linear search is usually slower than other algorithms in fully sorted arrays. For fully sorted arrays, I recommend using of logarithmic search, which has a time complexity of $O(\log_2 n)$. But for semi-sorted arrays, we cannot use logarithmic search. We are using linear search because it is easy to understand and has a simple code. It has a $O(n)$ time complexity where n is the size of the array.