International Journal on Emerging Technologies (Special Issue NCETST-2017) **8**(1): 527-530(2017)
(Published by Research Trend, Website: www.researchtrend.net*)*

ISSN No. (Print) : 0975-8364
ISSN No. (Online) : 2249-3255

Simple Search Technique with enhanced feature

Deep Chandra Andola
Assistant Professor, Amrapali Institute of Technology & Sciences, Haldwani, (U.K.), India

ABSTRACT: Searching is the method of finding any number or element in the given set of data or list. There are many searching algorithms – Linear Search, Binary Search, DN Search etc. Linear search is the simplest and easiest technique for finding the required element in the given data. In a Linear Search method, searching is continued until the desired element is found or the list gets exhaust. Time taken by this searching technique is equal to T (n). The disadvantage of this technique is that, once the search is successful it stops. In this paper, the algorithm is discussed takes equal time with respect to linear search, but it continues, after the search is successful.

Keywords : searching, linear, binary search, DN search

I. INTRODUCTION TO ALGORITHMS

In mathematics, an algorithm is an self-governing step-by-step set of operations to be performed. An algorithm is a well-organized and successful method that can be expressed within a restricted quantity of space and time and in a well defined formal language for estimating a function [1] [2].

In computer world, an algorithm is basically an instance of logic written in software by software developers to be effective for the intended "target" computer(s) for the target machines to produce output from given input (perhaps null). In addition every algorithm must satisfy the following criteria:

- **Input:** there are one or more quantities which are externally supplied.
- **Output:** At least one quantity is produces.
- **Definiteness:** Each instruction must be clear and unambiguous.
- **Finiteness:** If we trace out the instructions of an algorithm, then for all cases the algorithm will terminate after the finite number of steps.
- **Effectiveness:** Every instruction must be sufficiently basic that it can in principle be carried out by a person using only pencil and paper. It is not enough that each operation be definite, but it is also be feasible.
-

II. UNDERSTANDING COMMUNCATION OF TIME & SPACE TRADE-OFF FOR ALGORITHM

A problem may have numerous solutions. In order to choose the best algorithm for a particular task, you need to be able to judge how long a particular solution will take to run. Or, more accurately, you need to be able to judge how long two solutions will take to run, and choose the better of the two. You don't need to know how many minutes and seconds they will take, but you do need some way to compare algorithms against one another [1] [4].

Asymptotic complexity is a way of expressing the main component of the cost of an algorithm, using idealized (not comparable) units of computational work. Consider, for example, the algorithm for sorting a deck of cards, which proceeds by repeatedly searching through the deck for the lowest card. The asymptotic complexity of this algorithm is the square of the number of cards in the deck. This quadratic behavior is the main term in the complexity formula, it says, e.g., if you double the size of the deck, then the work is roughly quadrupled.

Now let us consider how we would go about comparing the complexity of two algorithms. Let f (n) be the cost, in the worst case, of one algorithm, expressed as a function of the input size n, and g (n) be the cost function for the other algorithm. E.g., for sorting algorithms, f (10) and g (10) would be the maximum number of steps that the algorithms would take on a list of 10 items. If, for all values of n >= 0, f

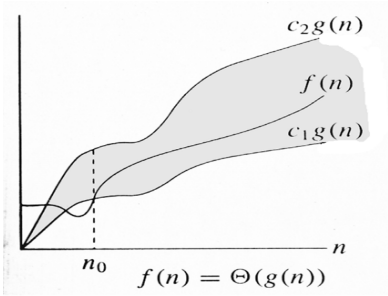
(n) is less than or equal to g(n), then the algorithm with complexity function f is strictly faster. But, generally speaking, our concern for computational cost is for the cases with large inputs; so the comparison of f (n) and g(n) for small values of n is less significant than the "long term" comparison of f(n) and g(n), for n larger than some threshold.

The following 3 asymptotic notations are mostly used to represent time complexity of algorithms.

1. θ Notation: The theta notation bounds functions from above and below, so it defines exact asymptotic behavior.

A simple way to get Theta notation of an expression is to drop low order terms and ignore leading constants.

For a given function g (n), we denote **θ** (g (n)) is following set of functions.

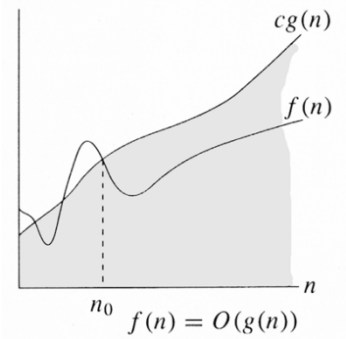


θ (g(n)) = {f (n): there exist positive constant c1, c2 and n_0 such that

0 <= c1*g (n) <= f (n) <= c2*g(n) for all n >= n_0

2) Big 'Oh' (O) Notation: The Big *O* notation defines an upper bound of an algorithm; it bounds a function only from above. For example, consider the case of Insertion Sort. It takes linear time in best case and quadratic time in worst case. We can safely say that the time complexity of Insertion sort is O (n^2). Note that O (n^2) also covers linear time.

The Big O notation is useful when we only have upper bound on time complexity of an algorithm.

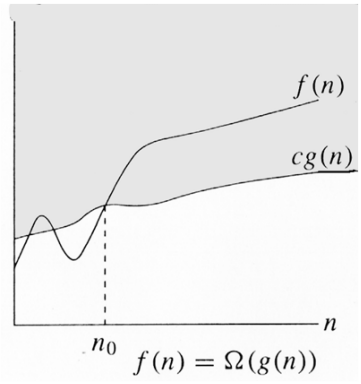


O (g(n)) = {f (n): there exist positive constant c and n_0 such that

0 <= f (n) <= c*g(n) for all n >= n_0

3) Ω Notation: Just as Big *O* notation provides an asymptotic upper bound on a function, **Ω** notation provides an asymptotic lower bound.

Ω Notation can be useful when we have lower bound on time complexity of an algorithm.



Ω (g (n)) = {f (n): there exist positive constant c and n_0 such that

0 <= c*g (n) <= f (n) for all n >= n_0

III. LINEAR SEARCH

In computer science, linear search or sequential search is a method for finding a particular value in a list that checks each element in sequence until the desired element is found or the list is exhausted. The list need not be ordered. Linear search is the simplest search algorithm; it is a special case of brute-force search. Its worst case cost is proportional to the number of elements in the list. Its expected cost is also proportional to the number of elements if all elements are searched equally.

Algorithm for Linear Search

Here '***A***' is the linear array with '***n***' elements and '***item***' is the given item of information. This algorithm find the location '***loc***' of item in *A*, or set loc = 0 if the search is unsuccessful. [3].

linear_search (A, item)

1. [insert item at the end of data, i.e. 'A']
   Set A [n+1] = item
2. [initialize counter]
   set loc = 1
3. [search for item]
   Repeat while A [loc] =! Item
   set loc = loc + 1
4. [successful]
   if loc = n + 1,
   else set loc = 0
5. Exit

Example:
Let us consider an array of ten numbers:
A = 12, 54, 23, 43, 12, 65, 23, 16, 87, 23
Now using linear search algorithm, discussed above, let us find the number, *num = 23*, in the given array. In the first step, the num (=23) will be inserted at the (*n+1*) location of the array *A*, i.e. at eleventh position. Thus, the new array will be:
A = 12, 54, 23, 43, 12, 65, 23, 16, 87, 23, **23**
After insertion of *num* into a given array, searching starts. In the very first step, when loc = 1, A[1] will be compare to A[n+1], i.e. A[11].
A = **12**, 54, 23, 43, 12, 65, 23, 16, 87, 23, **23**
In this step, it is an unsuccessful search. Similarly for loc = 2, it will be an unsuccessful search.
When the value of loc = 3, then the search will be successful.
A = 12, 54, **23**, 43, 12, 65, 23, 16, 87, 23, **23**
As soon as search is successful, it will automatically discontinue the loop in step 3 of algorithm and print successful and will exit.
In linear search algorithm, it provides the information about the existence of the searched number but it does not provide any information about the occurrence of the number.

Analysis:
For a list with n items, the best case is when the value is equal to the first element of the list, in which case only one comparison is needed. The worst case is when the value is not in the list (or occurs only once at the end of the list), in which case n comparisons are needed. Thus worst cost of Linear Search is **θ (n)**.

IV. MODIFIED LINEAR SEARCH TECHNIQUE

As we have seen in the above section that linear search technique just provide the information about the existence of the searched number, but it does not tells about the occurrence of it. To overcome this disadvantage we will design *Modified linear search technique*. In modified linear search technique, we will get information about the existence of the searched number but in addition it also provides the information about the total occurrence of it.
Working of this algorithm is simple. During the searching of an element in the given array modified linear search algorithm took one variable for counting the number of occurrences and one temporary array to store the location, every time search is successful.

Algorithm for Modified Linear Search Technique
Here '***A***' is the linear array with '***n***' elements and '***item***' is the given item of information. This algorithm provides information about the *item*, that it is present in the given array or not, also notifies the total number of occurrence of the item.

modified_linear_search (A, item)

1. set c = 0, j=1
2. set for i = 1 to n [initialize loop]
   [repeat step 3 to 6]
3. if (A[i] == item)
   [repeat step 4 to 6]
4. set c = c + 1
5. set temp[j] = i
6. set j++
   [end of if condition and for loop]
7. if c == 0
   then print "UNSUCCESSFUL"
   else
   print "SUCCESSFUL" and
   print "ITEM is present 'c' times at 'temp[j]' location "

Example:
Let us consider an array of ten numbers:
A = 12, 54, 23, 43, 12, 65, 23, 16, 87, 23
Now using modified linear search algorithm, discussed above, let us find the number, *num = 23*, in the given array.
In the first step, value of *c* will be set to *0* and loop will initialize.
For i = 1, item is not equal to A[1].
12, 54, 23, 43, 12, 65, 23, 16, 87, 23 item= **23**
So, value of c = 0.
For i = 2, item is not equal to A[2].
12, **54**, 23, 43, 12, 65, 23, 16, 87, 23 item= **23**
So, value of c = 0.
For i = 3, item is equal to A[3].
12, 54, **23**, 43, 12, 65, 23, 16, 87, 23 item= **23**
So, value of **c = 1** and **temp[1] = 3**.
For i = 4, item is not equal to A[4].
12, 54, 23, **43**, 12, 65, 23, 16, 87, 23 item= **23**
So, value of c = 1.
For i = 5, item is not equal to A[5].
12, 54, 23, 43, **12**, 65, 23, 16, 87, 23 item= **23**
So, value of c = 1.
For i = 6, item is not equal to A[6].
12, 54, 23, 43, 12, **65**, 23, 16, 87, 23 item= **23**
So, value of c = 1.
For i = 7, item is equal to A[7].
12, 54, 23, 43, 12, 65, **23**, 16, 87, 23 item= **23**
So, value of **c = 2** and **temp[2] = 7**.
For i = 8, item is not equal to A[8].
12, 54, 23, 43, 12, 65, 23, **16**, 87, 23 item= **23**
So, value of c = 2.
For i = 9, item is not equal to A[9].
12, 54, 23, 43, 12, 65, 23, 16, **87**, 23 item= **23**

So, value of c = 2.
For i = 10, item is equal to A[10].
12, 54, 23, 43, 12, 65, 23, 16, 87, **23** item= **23**
So, value of **c = 3** and **temp[3] = 10**.

Thus after completion of *for* loop, value of c =3 and temp[3] = {3, 7, 10}. This implies *item = 23* is present in the given array three time (c = 3) and at 3rd, 7th and 10th position.

Analysis:
In the above algorithm, we use only one single '*for*' loop. This implies that the complexity for the above algorithm will be T(n).
Best case: Best case is one where the item is searched at first position. In modified linear search algorithm, best case is when all the elements in the given array are equal to the item searched. In this case, *for* loop will execute for *n* times, so best case complexity will be **T(n)**.
Worst Case: In worst case, search will never be successful. In modified linear search algorithm, this condition appears when item will never equal to the elements of the given array. But *for* loop will execute *n* times. Thus, worst case complexity for modified linear search will be **T(n)**.

V. RESULT

As linear search and modified linear search are discussed above and their average time can be given as:

Table 1: Comparison between Linear Search & Modified Linear Search.

SNo	Technique	Average Time
1.	Linear Search	T (n) = **θ (n)**
2.	Modified Linear Search	T (n) = **θ (n)**

VI. CONCLUSION

In this paper, Linear Search and Modified Linear Search techniques were discussed with their average time (in Table1). Linear search gives information about the existence of element in the given array but modified linear search provide same information with some added features such as total number of occurrences in the given array and location of each occurrence. Since, both the searching technique took same amount of time, but with added features, modified linear search can be proved as more powerful technique than simple linear search.

REFERENCES

[1] Thomas H Cormen, Charles H Leiserson, Ronald L Rivest, Clifford Stein, "Introduction to Algorithm", 3rd Edition, Prentice Hall of India Pvt Ltd., Chapter 1 and Chapter 3.
[2] Ellis Horowitz, Suraj Sahni, Sanguthevar Rajasekaran, "Fundament of Computer Algorithms", Galgotia Publication, Chapter 1.3.
[3] Seymour Lipshutz, Schaum's Outline "Data Structures", Tata McGraw Hill, Chapter 4.7 and Chapter 4.8.
[4] Seymour Lipshutz, Marc Lars Lipson, Varsha H Patil, Schaum's Outline "Discrete Structure", Third Edition, Tata McGraw Hill, Chapter 3.9.