COMPUTER SCIENCE

LINEAR SEARCH ALGORITHM TO BE SOLVED BY PARAMETER LOOP

Dashdemberel J., *Master, Teacher, Mongolian National University of Education, School of Mathematics and Natural Sciences, Ulaanbaatar, Mongolia*
Buren-Arvijikh D., *Master, Teacher, Khovd University, School of Natural Sciences and Technology, Department of Information, Communication and Technology, Khovd, Mongolia*

DOI: https://doi.org/10.31435/rsglobal_conf/30122020/7350

Abstract. *There are common occasions for finding out required optional data or element from the given algebra of sets and any kind of systems when we are solving practical and informatics tasks. In order to solve these types of tasks or issues, students or researchers have to know about the methodologies for basic understanding which are called tasks for searching. This process is very important as for finding out basic understanding and methodologies, having detailed knowledge of algorithms and searching methods, although there are modern specific technologies and automatic programming systems.*
When searching methods are programmed for informatics tasks, necessary abilities such as seeking essential or import of program language operators and making analysis in difference among characterful solving should be owned by students or specialists. Therefore, I set my purpose of thesis of organizing linear searching algorithms by the parameter loop of the programming language C during the discovering process of identifying differences, patterns, and internal core of linear searching methodologies.
Keywords: *searching algorithm, linear searching, linear searching with potential barrier*

In the framework of proposed goal:
- To research linear searching algorithms and discover their patterns/ stereotypes
- To solve linear searching algorithm by the way of using a parameter loop (using C language)

We consider that where is the data or element among given system? And that is why this kind of task is called searching task and it is often transferred into the task which directed to seek given data from massive elements. During the seeking process of solving, if there isn't any definition or information on necessary data, linear searching method is urgent and effective. There are two types of algorithms which are differed from their comparing numbers:

A task: to find out x-element from a-linear massive with given n-elements.

1. Linear search: Main idea of this task is to select every first element of massive (from very final element). After that selected element should be compared with data or given element. If the value of the selected element fixes in the value of giving element, the number of selected elements from massive should be saved at additional variable. In the end value of additional variable should be checked. [1.89]

Version 1.1:
```
for (i=1; i<=n; i++)  if (a[ i ]==x)   k=i; /*to select from initial element, to save number of element which was found out*/
if (k<=n) printf("seeking element on the position of  %d",k); //to print result
else printf("Did not catch seeking/target element");
```

Weakness of this method:
- It continues searching process until it finishes to seek every element of massive; although given element or data is found out.
- If there are several given elements, it finds out the number of final (initial) elements. In order to stop the weakness or disadvantage of this, below condition is appeared after selected element or data was found out.

2. Linear searches with potential barrier: Main idea of this algorithm is to set out X element called as a barrier at 0 (n+1) position of massive. The main goal of this action is to simplify double condition which was written on repetition top slowing down the below execution of algorithm.

Otherwise, it is an opportunity to leave the only condition of (a[i] != x), when the loop condition continues again from the initial (final) point of the massive as for placing the x element, which is being selected among the a massive, for the position of (n+1). During this searching process, 0 (n+1) value of I –index will take the opposite value then searching process will be stopped, if x- element was not found out from a - massive. Because ending condition of the loop is called “barrier” element, there is no need to control over an issue on whether it finishes checking all elements. After the loops checked the result of the value of i -index will be printed based on whether it’s value differs from an index value of “barrier” element. [1.90]

```
Version 1.2:
i=1; //to select the first element of the massive
while (i<=n && a[i]!=x) /*all element did not checked and  if seeking element did not found out , to continue the searching process*/
i++; /*if seeking element was not selected, to change index due to select next element*/
if (i<=n) printf(“Seeking element on the position of %d ”,i); //print result
          else printf(“seeking or target element/data was not caught”);
```

```
Version 2:
a[0]=x; //to establish “Barrier” by the way of placing x-element at the position of 0
i=n; /*because ”barrier” element is placed on the initial point of massive элемент, searching process can start from the final index of element due to searching direction from the beginning point of massive to the ending index of element */
while (a[i] != x) // loop for keep searching
i --;  //if seeking element or data was not found out, index should be changed for the purpose of checking next element.
if (i != 0) printf(“target/seeking element on the position of %d”, i); //print result;
else printf(“target/seeking element was not found out”);
```

From this situation, following two patterns of loop can be observed:

Linear searching:	Linear search with barrier:
1. Useful for any kind of massive	1. Useful for any kind of massive
2. Loop condition is combined condition	2. Loop condition is simple
3. Execution of loop is SLOW	3. Execution of loop is FAST
4. There is NO NEED for extra development.	4. Extra development is NEEDED for placing “Barrier” element

There are several differences between two methods. Searching process needs loop with initial condition but these differences still exist upon both of them. But it is interesting that how to solve these algorithms by the way of using parameter loop.

Because loop in C language of programming contains itself a loop pattern with condition of beginning, there is full opportunity to solve these algorithms by the way of using parameter loop.

Version 1.1 When code is solved by the use of the parameter loop / cycle, it is allowed to be written by following form, in order to be improved. Therefore, let’s consider an operator.

- *Break operator: Break operator is used as an operator to check the condition or cause, and it is used in the corpus of implementing operator or used in the duplicate and loop operator. Break operator’s main duty is to separate execution of these operators and to transfer program coach into the next operator.*

Using break operator, two below weaknesses mentioned on version 1.1 will be disappearing and extra variable is not needed here as well.

```
Version 3:
for (i=1; i<=n; i++)  if (a[ i ]==x)   break; /*to select from the first element of massive, to stop loop when seeking element was founded out*/
if (i<=n) printf(“target element is on the position of %d ”,i); //print result
                    else printf(“target element was not found out”);
```

Version 1.2 can be written by parameter loop.

Version 4:
i=1; //to select the first element of the massive
for (;i<=n && a[i]!=x;) /*all element of massive was not checked and if target element is not found out, searching will be continued*/
i++;/*if seeking element is not found out, index of the element will be changed for selection of next element*/
if (i<=n) printf("seeking/target element is on the position of %d" ,i); //print results
else printf("selecting/target element was not found");

Pattern of parameter loop is used for this time but break operator-if (a[i]==x), which is mentioned before on version 3, is combined into the condition of parameter loop indeed.

Version 4 can be written down as follows and main point to focus is that parameter loop has potential to implement again only one operator behind itself and this one operator should be emptied.

Version 5:
for (i=1;i<=n && a[i]!=x; i++) /*If (from the first element till the last one) –all of the elements is not checked and target data/element is not found out, searching process will be continued*/
; //empty operator
if (i<=n) printf("target element is on the position of %d",i); //print result
else printf("target/seeking element was not found out");

- *Empty operator: do not do any action and sets for only (;).*

Parameter loop has a duty of executing an empty operator and ending loop process based on loop condition. Therefore, version 2 or linear searching with barrier can be written down as follows.

Version 6:
a[0]=x; //to establish x element as a "barrier" on the position of the 0
i=n; /* because "barrier" element is placed on the initial point of massive элемент, searching process can start from the final index of element due to searching direction from the beginning point of massive to the ending index of element */
for(;a[i] != x;) // loop for keep searching
i --; //to change index of the element if target element is not found out
if (i != 0) printf("seeking/target element on the position of %d", i); //print result
else printf("seeking/target element was not found out");

If the first element selection and change loop for the index are involved in this algorithm.

```
a[0]=x; for (i=n; a[i]!=x; i--);
if (i != 0) printf("target/seeking element on the position of %d", i);
            else printf ("target/seeking element was not found out");
```

Or

```
for (a[0]=x, i=n; a[i--]!=x; );
if (i >= 0) printf("target/seeking element on the position of %d", i);
            else printf ("target/seeking element was not found out");
```

This kind of typing can be brief comparing to other types of algorithms.

Conclusions. Researching the linear search algorithm enable opportunities to observe the differences between them, to solve the task fully by the way of using parameter loop, and to extend own abilities or understanding as for using many variables and different methods as well as these all opportunities can provide accurate task solution and deep understanding of using an algorithm.

REFERENCES

1. Д.Цэдэвсүрэн, Ц.Хатанхангай, Л.Чойжооваичиг "Мэдээлэл зүйн практикум Turbo Pascal 7.0". – УБ,1999
2. Р.Минжирмаа. "Параметрт давталтын харьцуулалт", "Баруун монгол, түүний хил залгаа нутгийн ард түмний соёл, байгалийн нөөц" олон улсын X хурал. Ховд., 2011