Journal of Network Communications and Emerging Technologies (JNCET) www.jncet.org
Volume 5, Special Issue 2, December (2015)

Layer Search Algorithm for 2-D Array

Tushar Sharma
Department of Computer Science Engineering, BS Anangpuria Institute of Tech. & Management, Haryana, India

Rohit Raghav
Department of Computer Science Engineering, BS Anangpuria Institute of Tech. & Management, Haryana, India

Abstract – Searching algorithms are considered as the most required techniques in "Data Structures and Algorithm". This research paper is presenting a new searching algorithm named as Layer Search for searching elements in 2-D Array. Layer Search is an algorithm which can particularly be implemented on square matrix. In this paper wholesome description, pseudo code, implementation and efficiency with conclusion is mentioned for the Layer Search which works faster than Linear Search.

Index Terms – Searching Techniques, Linear Search, Layer Search, Time Complexity, Chain Reaction.

This paper is presented at International Conference on Recent Trends in Computer and information Technology Research on 25th & 26th September (2015) conducted by B. S. Anangpuria Institute of Technology & Management, Village-Alampur, Ballabgarh-Sohna Road, Faridabad.

1. INTRODUCTION

The searching algorithms are known as the backbone of a program and an application. In this era we are using many of the applications and web applications in our PC as well as in our desktop and these apps too have search command option to search a particular value in their database. By analysing a game named "Chain Reaction" which works on matrix concept where an element gets split into all possible directions we have designed a new searching technique named as Layer Search. In many extents Layer Search is better and faster than Linear Search.

Linear Search: This is the traditional algorithm for searching an element in the array. In this searching technique the compiler checks the existence of a particular value one by one checking method. This means it first move in 1st row and compare the entered value with all elements in that row. If compiler found that element, it shows its index otherwise follow the same search procedure with the 2nd row and so on.
Ex. If an array is of size [3][3] then 1st it will search [0][0], [0]1], [0][2]. Then [1][0], [1][1], [1][2] and so on the operation will be followed.

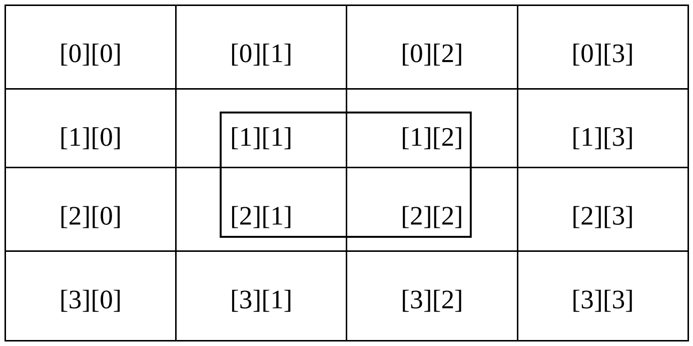
Layer Search: This is a new algorithm designed by us which works on the concept of layer by layer searching technique. In this technique the compiler distributes the whole square matrix into different layers in square formation. This means it will consider the middle most elements of the array forming a square as 1st layer, the elements present at the very 1st outer indexs of the elements of 1st layer are considered as 2nd layer and so on the distribution of the layers take place.

2. CONCEPT BEHIND SEARCHING

Each searching algorithm has its own process to search the existence of an element in 2-D array. This process is specified at the time of designing that particular algorithm by the programmer.

In this sub-topic we will discuss about the process of searching process of Layer Search in a square matrix.

Ex. Let we have a 2-D array or square matrix of size [4][4].



[0][0]	[0][1]	[0][2]	[0][3]
[1][0]	[1][1]	[1][2]	[1][3]
[2][0]	[2][1]	[2][2]	[2][3]
[3][0]	[3][1]	[3][2]	[3][3]

Fig.1: 1st Layer formation

[0][0]	[0][1]	[0][2]	[0][3]
[1][0]	[1][1]	[1][2]	[1][3]
[2][0]	[2][1]	[2][2]	[2][3]
[3][0]	[3][1]	[3][2]	[3][3]

Fig.2: 2nd Layer formation

As we can see in above example we are having a square matrix of size 4X4. By the law proposed for distribution of layers, compiler will break the whole matrix into various layers. In the above example, there will be the formation of only 2 layers. 1st layer will be containing the indexes [1][1], [1][2], [2][1 and [2][2] and 2nd layer will be containing the indexes [0][0], [0][1], [0][2], [0][3], [1][0], [1][3], [2][0], [2][3], [3][0], [3][1], [3][2] and [3][3] The compiler will check the existence of the value to searched in the 1st layer. If it is found, it will show the message "Found" with the respective index otherwise will continue its search till the last layer.

3. PSEUDO CODE

Let us consider an array, list of size n*n. Let m1, m2 be the middle indexes of the array, i and j be the loop variables and variable named "value" to store the value of element to be searched. The step-by-step procedure for implementing the Layer Search is as follows:-

```
Step 1: Enter the size of n*n square matrix.

Step 2: Enter the value to be searched.

Step 3:  m2=n/2.

Step 4:  m1=m2-1.

Step5: If n is even proceed to function call LayerSearch(list,value,m1,m2) and skip to step 5.1.1 otherwise to step 5.2.

Step 5.1.1: If (m1<0 OR m2>n-1) then print "Value not found" and go to step 6 otherwise to step 5.1.2. /* m1 and m2 in layersearch() contains value passed in it. Not the actual values at main(). */

Step 5.1.2: Loop begin where i=m1 and j=m1, will execute till j<=m2 and increment of j by 1.

Step 5.1.2.1: If value is equal to list[i][j], print "Value found at [i][j] index and proceed to step 6 otherwise to step 5.1.2.2.

Step 5.1.2.2: If value is equal to list[j][i], print "Value found at [j][i] index and proceed to step 6 otherwise to step 5.1.3. Loop ended.

Step 5.1.3: Loop begin where i=m2 and j=m1, will execute till j<=m2 and increment of j by 1.

Step 5.1.3.1: If value is equal to list[i][j], print "Value found at [i][j] index and proceed to step 6 otherwise to step 5.1.3.2.

Step 5.1.3.2: If value is equal to list[j][i], print "Value found at [j][i] index and proceed to step 6 otherwise to step 5.1.4. Loop ended.

Step 5.1.4: Recursive function call LayerSearch(list,value,m1-1,m2+1) and proceed to step 5.1.

Step 5.2: If value is equal to list[m2][m2], print"Value found at [m2][m2] index and goto step 6 otherwise function call LayerSearch(list,value,m2-1,m2+1) and goto to step 5.1.1. /* Here (m2-1) is m1 and (m2+1) is m2 for LayerSearch() as in step 5.1 for the case of even. */

Step 6: End.
```

4. IMPLEMENTING LAYER SEARCH IN C

Let us make a program of Layer Search in C to check the working procedure.

```
/*Layer Search in 2-D Array or Square Matrix"

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#include<time.h>

#define n 8

void layersearch(int list[n][n],int value, int m1, int m2);

void main()
```

```
{
int list[n][n]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19
,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39
,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59
,60,61,62,63,64};
int value,m1,m2,i,j;
clrscr();
        for ( i=0 ; i< n ; i++)
        {
                for ( j=0; j< n; j++)
                {
                printf("[%d][%d]=%d\t",i,j,list[i][j]);
                }
        }
        printf("\n\nEnter the value to be searched : ");
        scanf("%d",&value);

        m2=n/2;
        m1=m2-1;
        layersearch(list,value,m1,m2);
}

void layersearch(int list[n][n],int value, int m1, int m2)
{
        int i,j;
        if (m1<0 || m2>n-1)
        {
                printf("%d Not found",value);
                getch();
                exit(0);
        }
        for ( i=m1 , j=m1 ; j<=m2 ; j++)
        {

                if (list[i][j]==value)
                {

        printf("%d is present at [%d][%d] index",value,i,j);
                        getch();
                        exit(0);
                }

                else
                {
                        if (list[j][i]==value)
                        {
                printf("%d is present at
                [%d][%d] index",value,j,i);
                                getch();
                                exit(0);
                        }
                }
        }

        for ( i=m2 , j=m1 ; j<=m2 ; j++)
        {
                if (list[i][j]==value)
                {
printf("%d is present at [%d][%d]  index",value,i,j);
```

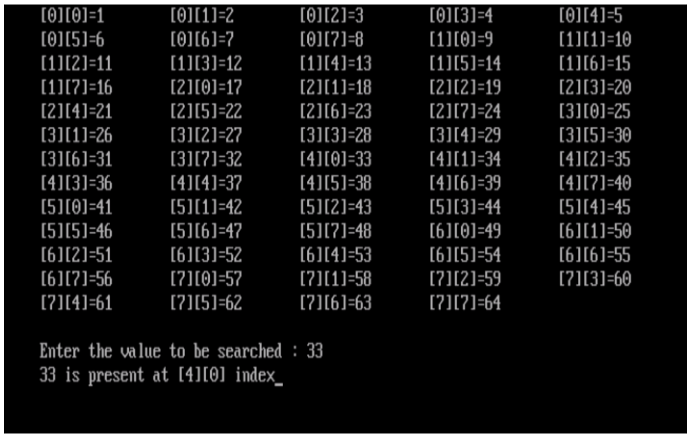
```
                                        getch();

                                        exit(0);

                              }

                              else

                              {

                                        if (list[j][i]==value)

                                        {

printf("%d is present at [%d][%d] index",value,j,i);

                                                  getch();

                                                  exit(0);

                                        }

                              }

                    }

                    layersearch(list,value,m1-1,m2+1);

}
```

This is the code of implementation of Layer Search in C Technology. We can also implement the same logic in other languages too.

5. OUTPUT OF LAYER SEARCH

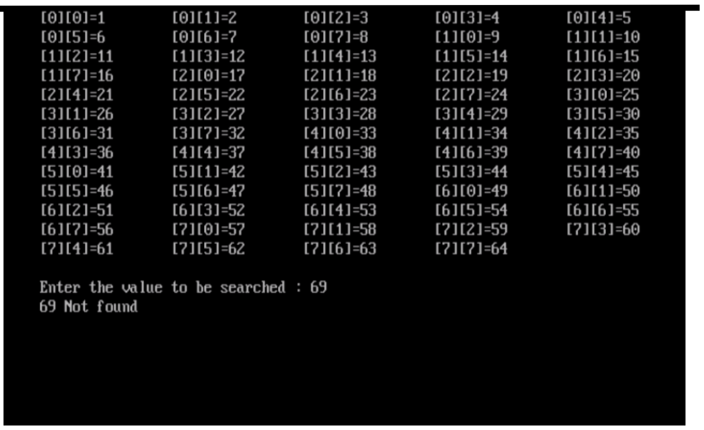
Here are the few of the outputs of the layer search.

These are the outputs for the values present in the array.



```
[0][0]=1    [0][1]=2    [0][2]=3    [0][3]=4    [0][4]=5
[0][5]=6    [0][6]=7    [0][7]=8    [1][0]=9    [1][1]=10
[1][2]=11   [1][3]=12   [1][4]=13   [1][5]=14   [1][6]=15
[1][7]=16   [2][0]=17   [2][1]=18   [2][2]=19   [2][3]=20
[2][4]=21   [2][5]=22   [2][6]=23   [2][7]=24   [3][0]=25
[3][1]=26   [3][2]=27   [3][3]=28   [3][4]=29   [3][5]=30
[3][6]=31   [3][7]=32   [4][0]=33   [4][1]=34   [4][2]=35
[4][3]=36   [4][4]=37   [4][5]=38   [4][6]=39   [4][7]=40
[5][0]=41   [5][1]=42   [5][2]=43   [5][3]=44   [5][4]=45
[5][5]=46   [5][6]=47   [5][7]=48   [6][0]=49   [6][1]=50
[6][2]=51   [6][3]=52   [6][4]=53   [6][5]=54   [6][6]=55
[6][7]=56   [7][0]=57   [7][1]=58   [7][2]=59   [7][3]=60
[7][4]=61   [7][5]=62   [7][6]=63   [7][7]=64

Enter the value to be searched : 33
33 is present at [4][0] index_
```

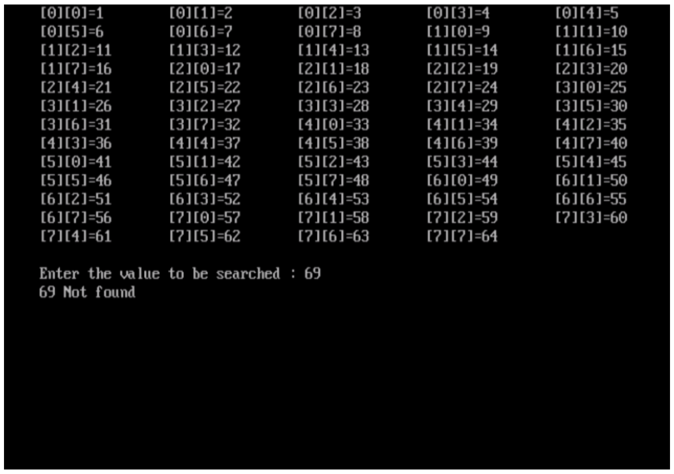
Fig.3: Successful Search of 33



```
[0][0]=1    [0][1]=2    [0][2]=3    [0][3]=4    [0][4]=5
[0][5]=6    [0][6]=7    [0][7]=8    [1][0]=9    [1][1]=10
[1][2]=11   [1][3]=12   [1][4]=13   [1][5]=14   [1][6]=15
[1][7]=16   [2][0]=17   [2][1]=18   [2][2]=19   [2][3]=20
[2][4]=21   [2][5]=22   [2][6]=23   [2][7]=24   [3][0]=25
[3][1]=26   [3][2]=27   [3][3]=28   [3][4]=29   [3][5]=30
[3][6]=31   [3][7]=32   [4][0]=33   [4][1]=34   [4][2]=35
[4][3]=36   [4][4]=37   [4][5]=38   [4][6]=39   [4][7]=40
[5][0]=41   [5][1]=42   [5][2]=43   [5][3]=44   [5][4]=45
[5][5]=46   [5][6]=47   [5][7]=48   [6][0]=49   [6][1]=50
[6][2]=51   [6][3]=52   [6][4]=53   [6][5]=54   [6][6]=55
[6][7]=56   [7][0]=57   [7][1]=58   [7][2]=59   [7][3]=60
[7][4]=61   [7][5]=62   [7][6]=63   [7][7]=64

Enter the value to be searched : 69
69 Not found
```

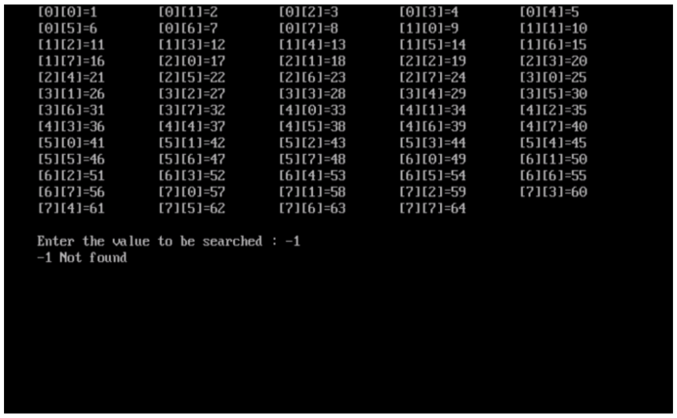
Fig.4: Successful Search of 55

These are the outputs for the values not present in the array.



```
[0][0]=1    [0][1]=2    [0][2]=3    [0][3]=4    [0][4]=5
[0][5]=6    [0][6]=7    [0][7]=8    [1][0]=9    [1][1]=10
[1][2]=11   [1][3]=12   [1][4]=13   [1][5]=14   [1][6]=15
[1][7]=16   [2][0]=17   [2][1]=18   [2][2]=19   [2][3]=20
[2][4]=21   [2][5]=22   [2][6]=23   [2][7]=24   [3][0]=25
[3][1]=26   [3][2]=27   [3][3]=28   [3][4]=29   [3][5]=30
[3][6]=31   [3][7]=32   [4][0]=33   [4][1]=34   [4][2]=35
[4][3]=36   [4][4]=37   [4][5]=38   [4][6]=39   [4][7]=40
[5][0]=41   [5][1]=42   [5][2]=43   [5][3]=44   [5][4]=45
[5][5]=46   [5][6]=47   [5][7]=48   [6][0]=49   [6][1]=50
[6][2]=51   [6][3]=52   [6][4]=53   [6][5]=54   [6][6]=55
[6][7]=56   [7][0]=57   [7][1]=58   [7][2]=59   [7][3]=60
[7][4]=61   [7][5]=62   [7][6]=63   [7][7]=64

Enter the value to be searched : 69
69 Not found
```

Fig.4: Unsuccessful Search of 69



```
[0][0]=1    [0][1]=2    [0][2]=3    [0][3]=4    [0][4]=5
[0][5]=6    [0][6]=7    [0][7]=8    [1][0]=9    [1][1]=10
[1][2]=11   [1][3]=12   [1][4]=13   [1][5]=14   [1][6]=15
[1][7]=16   [2][0]=17   [2][1]=18   [2][2]=19   [2][3]=20
[2][4]=21   [2][5]=22   [2][6]=23   [2][7]=24   [3][0]=25
[3][1]=26   [3][2]=27   [3][3]=28   [3][4]=29   [3][5]=30
[3][6]=31   [3][7]=32   [4][0]=33   [4][1]=34   [4][2]=35
[4][3]=36   [4][4]=37   [4][5]=38   [4][6]=39   [4][7]=40
[5][0]=41   [5][1]=42   [5][2]=43   [5][3]=44   [5][4]=45
[5][5]=46   [5][6]=47   [5][7]=48   [6][0]=49   [6][1]=50
[6][2]=51   [6][3]=52   [6][4]=53   [6][5]=54   [6][6]=55
[6][7]=56   [7][0]=57   [7][1]=58   [7][2]=59   [7][3]=60
[7][4]=61   [7][5]=62   [7][6]=63   [7][7]=64

Enter the value to be searched : -1
-1 Not found
```

Fig.5: Unsuccessful Search of -1

6. CONCLUSION

In this paper we proposed a new searching algorithm for 2-D array of order n*n or square matrix. The working of this algorithm is totally based on the formation of different layers of elements on the basis of their indexes and then searching a particular element layer by layer. The complexity of linear search is n^2 because it uses straight forward method. The complexity of the Layer Search is very less for inner layers and little high for outer layers.
The best case in the algorithm is for the elements present at the inner layer.
The worst case in the algorithm is for the elements present at the outer most layers.

REFERENCES

[1] Jyotirmayee Rautaray, Hashed Based Searching Algorithm, Vol. 2, Issue 2, Februrary 2013.
[2] Nitin Arora, Two Way Linear Search Algorithm, Vol. 107-No. 21, December 2014.
[3] Debadrita Roy, A comparative Analysis of Three Different Types of Searching Algorithms in Data Structures, Vol. 3, Issue 5, May 2014.
[4] A. K. Sharma, Data Structures using C, 2nd Edition, Pearson Publications.
[5] E Balagurusamy, Object Oriented Programming with C++, 4th Edition, The McGraw-Hill Compaines.
[6] S.K. Srivastava, Data Structures Through C in Depth, 1st Edition, BPB Publications.