International Journal of Computer Applications Technology and Research
Volume 5– Issue 1, 26 - 33, 2016, ISSN:- 2319–8656

GRID SEARCHING

Novel way of Searching 2D Array

Rehan Guha
Institute of Engineering & Management
Kolkata, India

Abstract: Linear/Sequential searching is the basic search algorithm used in data structures. Linear search is used to find a particular element in a 2D array. It is not compulsory to arrange an array in any order (Ascending or Descending) as in case of 2D binary search. In this paper, I present a unique searching algorithm named Grid Search, which helps to search an unsorted 2D Array/Matrix with least time complexity and iteration. We also have compared the Grid searching algorithm with Linear Search Algorithm. We used C++ for implementation and analysis of CPU time taken by both the algorithms. Results have shown that Grid Searching Algorithm is working well for all input values and it takes lesser time than Sequential Searching in all aspects.

Keywords: Matrix Searching algorithms; 2D Array Searching algorithms; 2D Array Linear Searching; 2D Array Grid Searching; Complexity Analysis; Algorithms; Searching Algorithms;

1. INTRODUCTION

2D Array or Matrix searching algorithm is a widely used and known searching algorithm. Though it is a primitive and basic searching technique, but it is widely used in different fields even today. Fields include Bio-medical image processing, Rasterization techniques, Electro-optical displays, Pixel tracing and many more.

In this paper I have developed a new searching technique named Grid Searching, based on the existing limitations of the 2D array linear/sequential search algorithm.

In 2D array searching an iterator[1] traverses sequentially from left to right (row major) or top to down (column major). When it encounters an element which is equal to the key[2] the search stops and the index of the element is returned, but if the key is not present in the 2D array the function will return 0, implying that the key is not found. Sequential search or linear search is the basic search algorithm used in data structures for 2D array[1] with unsorted data.

2. RELATED WORK

Linear search is used to find a particular element in an array. It is not compulsory to arrange an array in any order[3] as in the case of binary search. Linear search starts by sequentially scanning the elements in the 2D array (Row major wise or Column major wise) and if the element has been found, it will display the particular element's index value in the 2D array/matrix, else it will display not found[2][3].

If we declare a 2D array of order (n X n) i.e. size of array is n^2 then we initialize the array with the value (either static or dynamic declaration of values)[4]. In that if we mention the particular element to be identified, the linear search will start to search the array from the index 0, 0. If the value is not found, then the iterator will increment the index value by 1 in the column/row index becoming (0, 1) / (1, 0) and then it will check the element in that index again. This process continues till the element has been identified in the 2D Array/Matrix.

[1] It is a variable which acts as an counter variable and acts as an index for 2D array/matrix
[2] The element which is to be searched in the 2D array/matrix

[3] Ascending, Descending or any other sequence/series

Algorithm(s):

- LINEAR_SEARCH (M, NUM, n)

Input:

- 2D array/matrix M
- NUM is the key which is to be searched in M
- n is the order of the 2D array/matrix M (n X n)
- i & j are the iterators and works as index of the 2D array/matrix.

Pseudo Code: where, n= 1,2,3,4,5,6,7 …

LINEAR_SEARCH (M, NUM, n)

```
1.  Flag ← 0
2.  for i ←0 to n-1
3.      for j←0 to n-1
4.          if (M[i][j] == NUM) then
5.              Flag←1
6.              return (i, j)
7.          end if
8.      next j
9.  next i
10. if (Flag == 0) then
11.     return 0
12. end if
```

Flag –Checks whether the element is present or not.

If function LINEAR_SEARCH finds a match with the key then it returns the index of the element(s) (there might be multiple match of elements in the 2D array/matrix). Else it returns 0 which signifies element not found.

3. MODIFIED ALGORITHM (GRID SEARCHING)

If we declare and initialize a 2D array of order (n X n) i.e. size of the array is n^2 and if we mention the particular element to be identified, then the grid search will start searching the array from the index 1, 1. Now considering the index, it will search its boundary like a Grid. If the element is not found in the boundary then it will search the next set of Grids according to the order of matrix. This process continues till the element has been identified.

Algorithm(s):

- GRID_SEARCH(M, n, NUM),
- CHECK (M, n, i, j, NUM),
- RANGE (n)

Input:

- 2D array/matrix M
- NUM is the key which is to be searched in M
- n is the order of the 2D array/matrix M (n X n)
- i & j are the iterators and works as index of the 2D array/matrix.

Pseudo Code: where, n= 1,2,3,4,5,6,7 …

RANGE (n)

```
1. if (n MOD 3 == 1) then
2.     T ←(n+2) / 3
3. else if (n MOD 3 == 2) then
4.     T ←(n+1) / 3
5. else
6.     T ←(n) / 3
7. end if
8. return ( T * T )
```

CHECK (M, n, i, j, NUM)

```
1. if ( ((i>=0) AND (i<n)) AND ((j>=0) AND
   (j<n)) AND M[i][j] == NUM ) then
2.     return 1
3. end if
4. return 0
```

GRID_SEARCH (M, n, NUM)

```
1. Flag ← 0
2. T ←RANGE(n)
3. for i←1 to T
4.     for j←1 to T
5.         if (CHECK(M, n, i-1, j-1,
   NUM)) then
6.             Flag ← 1
7.             return(i-1, j-1)
8.     end if
```

```
9.              if (CHECK(M, n, i-1, j, NUM)) then
10.                 Flag ← 1
11.                 return(i-1, j)
12.             end if
13.             if (CHECK(M, n, i, j-1, NUM)) then
14.                 Flag ← 1
15.                 return(i, j-1)
16.             end if
17.             if (CHECK(M, n, i-1, j+1, NUM)) then
18.                 Flag ← 1
19.                 return 1
20.             end if
21.             if (CHECK(M, n, i, j, NUM)) then
22.                 Flag ← 1
23.                 return(i, j)
24.             end if
25.             if (CHECK(M, n, i, j+1, NUM)) then
26.                 Flag ← 1
27.                 return (i, j+1)
28.             End if
29.             if (CHECK(M, n, i+1, j-1, NUM)) then
30.                 Flag ← 1
31.                 return(i+1, j-1)
32.             End if
33.             if (CHECK(M, n, i+1, j, NUM)) then
34.                 Flag ← 1
35.                 return(i+1, j)
36.             end if
37.             if (CHECK(M, n, i+1, j+1, NUM)) then
38.                 Flag ← 1
39.                 return(i+1, j+1)
40.             end if
41.         next (j+3)
42. next (i+3)
43. if (Flag == 0) then
44.         return 0
45. end if
```

NOTE: Here the order of nested If-Else condition is important and cannot be changed. If it is changed the result of the algorithm may vary.

Example: If array M has 1 element only i.e. order of the array is 1(i.e. 1 X 1) and now if we search the array and the element is found then the total number of searches will equal to 1. But if the order of the nested If-Else condition is changed then the total number searches will be greater than 1 thus in turn increases the complexity of the algorithm.

Flag –Checks whether the element is present or not.

T –Holds the returned value from the function RANGE(n).

If function GRID_SEARCH finds a match with the key then it returns the index of the element(s) (there might be multiple match of elements in the 2D Array/Matrix). Else it returns 0 which signifies element is not found.

4. PROOF OF CORRECTNESS FOR GRID SEARCHING

To prove that an algorithm is correct, we need to show two things: (1) that the algorithm terminates, and (2) that it produces the correct output [5] [6].

4.1 Algorithm Grid Searching terminates after a finite number of steps

The variable T[4] holds a finite number according to the function RANGE(n) (refer Section 3) and the iterator variable starts from 1 and ends at T with increment of 3. If the iteration variable is greater than equal to T then the loop terminates and does not go into an infinite loop.

4.2 Algorithm Grid Searching produces correct output

If the element is found then the function GRID_SEARCH returns the index value of the found element(s) as well as sets the value of Flag[3] to 1 and if the value is NOT FOUND then the function GRID_SEARCH returns 0 indicating that the element is not present in the 2D array/matrix, thus resulting to a correct output.

[4] refer to Grid Searching algorithm under Section 3

5. PERFORMANCE ANALYSIS AND COMPARISON

Both the searching algorithm (Sequential Searching and Grid Searching) were implemented in C++ using GNU G++ Compiler. Both the searching algorithm was executed on machine with:-

32-bit Operating System having Intel® Core™ i3-4005U CPU @ 1.70GHz, 1.70GHz and installed memory (RAM) 4.00GB.

5.1 Analysis of the Function of both the algorithms

Function of Linear/Sequential searching *f(x)*:

$f(x) = x^2 + x + 1$ …..(i)

Function of Grid searching *g(x)*:

$g(x) = x^2/9 + 18x + 5$ …..(ii)

The functions are derived from their respective algorithms (refer Section 2 & 3). The function has taken the **for** loop(s), **if-else** condition(s) into consideration.

The condition below is true for all the values of x greater than equal to 20 (x >= 20) according to the equation (i) and (ii), where x= 1, 2, 3, 4 …

$g(x)>f(x)$ …..(iii)

Table 1: Analysis of Functions

N	Grid Searching (Proposed Algorithm) Function	Sequential Searching Function
0	5	1
3	60	13
6	117	43
9	176	91
12	237	157
15	300	241
18	365	343
21	432	463
24	501	601
27	572	757
30	645	931
33	720	1123
36	797	1333
39	876	1561
42	957	1807
…	…	…

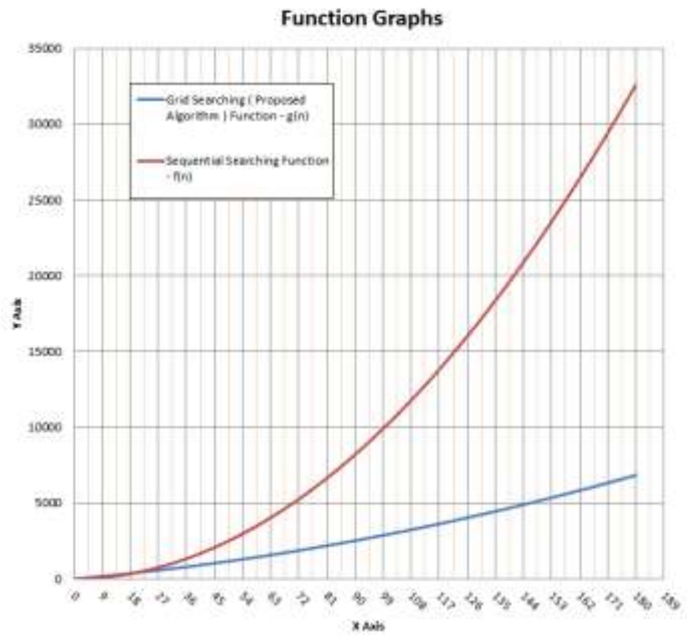

Figure 1

From the above graph (Fig. 1) we may come to a conclusion that Grid Searching function (*g(x)*) is highly effective with respect to Linear/Sequential searching (*f(x)*) when x >= 20 where x=1, 2, 3, 4, ...

5.2 Analysis of Worst-case iteration complexity

Worst-case iteration complexity measures the resources (e.g. running time, memory) that an algorithm requires in the worst-case. It gives an upper bound on the resources required by the algorithm. Thus the worst-case of an algorithm is the total number of iteration(s) required for searching an element which is not present in the 2D array/matrix, thus shows the total no. of iteration required for fulfilling a NOT FOUND condition.

NOT FOUND condition is checked for the order of matrices (n X n) where n = 3, 6, 9, 12…

According to equations (i) and (ii) from Section 5.1

$f(x) = x^2 + x + 1$ …..(i) (refer Section 5.1)

Function of Linear/Sequential searching is *f(x)*

And,

$g(x) = x^2/9 + 18x + 5$ …..(ii) (refer Section 5.1)

Function of Grid searching is *g(x)*

The highest degree of both the equations is as follows:

$f(x) = g(x) = x^2$ ….. (iii)

Thus from equation (iii) we can conclude,

The Orders of Complexity of both the equations are $O(n^2)$

Therefore,

Orders of complexity of both the algorithms are

O(Grid Searching) = O(Sequential Searching) = $O(n^2)$

But,

According to the equation (i) and (ii) considering the highest degree of the equation with constants,

We can say,

$x^2 >> x^2/9$ ….. (iv)

x=1, 2, 3, 4 …

(x^2 is much greater than $x^2/9$)

Precisely, we can also say

O(Grid Searching) = $O(n^2/9)$

O(Sequential Searching) = $O(n^2)$

Finally,

Grid searching has complexity of $O(n^2/9)$

And Sequential searching has a complexity of $O(n^2)$.

The condition below is true for all the values of n where n = 1, 2, 3, 4…

$O(n^2/9) << O(n^2)$

Table 2: Iteration Complexity

n	Order	Grid Searching (Proposed Algorithm)	Sequential Searching
1	3	1	9
2	6	4	36
3	9	9	81
4	12	16	144
5	15	25	225
6	18	36	324
7	21	49	441
8	24	64	576
9	27	81	729
10	30	100	900
11	33	121	1089
12	36	144	1296
…	…	…	…

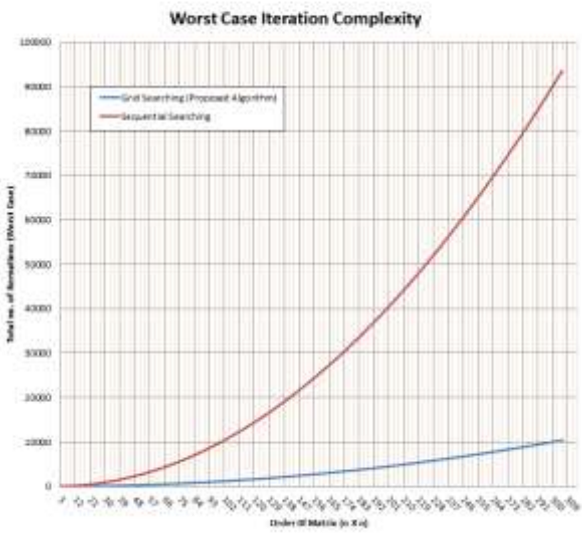

Figure 2

From the above graph (Fig. 2) we may conclude that Grid Searching is highly efficient with an increase in the order of matrix with respect to Linear/Sequential searching.

5.3 Analysis of Worst-case Time complexity

The worst-case time complexity indicates the longest running time performed by an algorithm given *any* input of size *n* (if the algorithm satisfies NOT FOUND condition then that is the longest running time of the algorithm), and thus this guarantees that the algorithm finishes on time and gives us the total worst-case time complexity.

Moreover, the order of growth of the worst-case complexity is used to compare the efficiency of two algorithms.

All the readings shown below are an average of 10 reading of a single NOT FOUND condition. Time is represented in *msec*, and NOT FOUND condition is checked for the order of matrices (n X n) where n = 3, 6, 9, 12…

Table 3: Time Complexity

N	Order	Grid Searching (Proposed Algorithm)	Sequential Searching
1	3	0.31287	0.32967
2	6	0.384615	0.494505
3	9	0.32967	0.384615
4	12	0.32967	0.43956
5	15	0.384615	0.49456
6	18	0.494505	0.512563
7	21	0.43956	0.549451
8	24	0.494505	0.549451
9	27	0.549451	0.494505
10	30	0.494505	0.494505
11	33	0.43956	0.494505
12	36	0.32967	0.549451
…	…	…	…

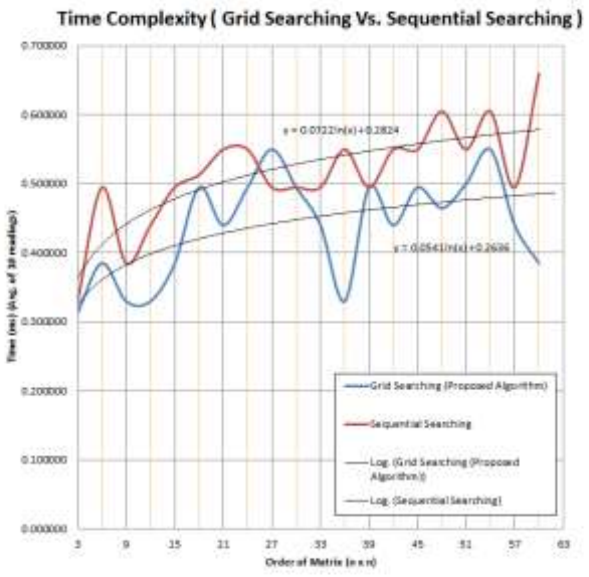

Figure 3

From the graph (Fig. 3) we obtain the logarithm values of the trend-line which is automatically generated by the Graph Analyzer from Table 3.

The logarithm trend-line for linear/sequential searching *f(x)* is as follows:-

$f(x) = 0.0722 \ln(x) + 0.2824$ .....(i)

The logarithm trend-line for Grid searching *g(x)* is as follows:-

$g(x) = 0.0541 \ln(x) + 0.2636$ …..(ii)

The condition below is true for all values of x (1, 2, 3…) according to the equation (i) and (ii),

$g(x)<f(x)$ …..(iii)

Therefore, from equation (iii) we can conclude that Grid Searching is more efficient than Sequential Searching in case of Worst-case time complexity.

5.4 Analysis of Condition checking

Condition checking is the number of conditions required to find a particular element in the matrix irrespective of the presence of the element in the matrix.

All the readings shown below are obtained from a matrix of order 165x165, where all the elements are arranged in an ascending order (but for both the algorithms elements *may not* be arranged in any particular order (Ascending or Descending)) and elements are searched in a cubic function

$f(n) = n^3$ …..(i)

Where,

n = 1, 2, 3, 4… in the matrix of order 165 x 165.

Table 4: Condition Checking

n	Element	Grid Searching (Proposed Algorithm)	Sequential Searching
1	1	1	2
2	8	20	9
3	27	76	28
4	64	190	65
5	125	371	126
6	216	141	217
7	343	25	344
8	512	524	513
9	729	672	730
10	1000	988	1001
11	1331	1472	1332
12	1728	1653	1729
13	2197	2055	2198
14	2744	2687	2745
15	3375	3069	3376
…	…	…	…

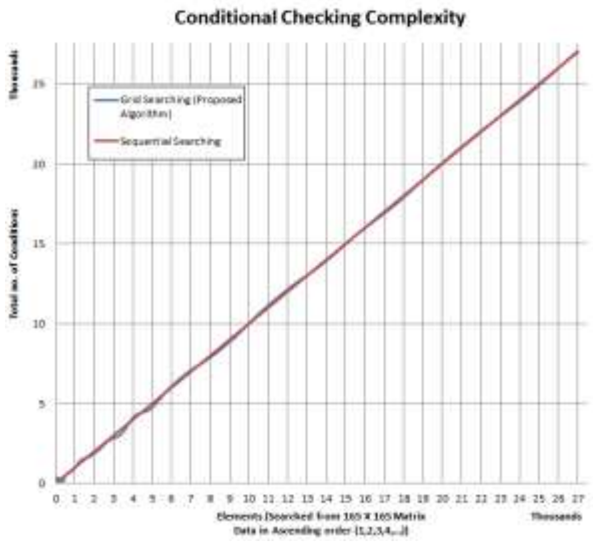

Figure 4

From the above graph (Fig. 4) we can come to a conclusion that there is not much deviation in between the two algorithms.

So, Condition checking complexity of both the algorithms is nearly same and does not affect the total time complexity.

6. CONCLUSIONS

Searching is the technique that is used to find a particular element in an array. In Grid searching and Sequential searching it is not compulsory to arrange an array in any order (Ascending or Descending) as in the case of binary search of 2D array. Result shows that Grid Searching Algorithm is working well for all input values and it is highly efficient than Sequential Searching in all of the following aspects:

1. Analysis of Functions (*f(x)* & *g(x)*) of both the algorithms
2. Worst-case iterations
3. Worst-case Time complexity of both the algorithms
4. Total no. of conditional checking

Grid searching algorithm also satisfies both the proof of correctness.

So for searching unordered set of data in square matrix Grid searching is more efficient than Sequential searching.

7. FUTURE SCOPE

Future scope of Grid searching Algorithm is as follows:-

1. If it can be implemented for any order of matrix (non-square matrix) with unordered set of data present in it.
2. Rasterization algorithm, Electro-optical display algorithm, Pixel tracing algorithm are some examples of algorithms which are derived from sequential searching and are widely used in different fields, but if the same algorithm can be built using Grid searching then the performance might increase.

8. REFERENCES

[1] E. Horowitz and S. Sahni, fundamental of Data Structure Rockville, MD: Computer Science Press, 1982.

[2] Knuth, D.E. The Art of Computer Programming, Vol. 3: Sorting and Searching, Addison Wesley (1997), 3rd ed., 396-408.

[3] Alfred V., Aho J., Horroroft, Jeffrey D.U. (2002) Data Structures and Algorithms.

[4] Frank M.C. (2004) Data Abstraction and Problem Solving with C++. US: Pearson Education, Inc.

[5] Cormen T.H., Leiserson C.E., Rivest R.L. and Stein C. (2003) Introduction to Algorithms MIT Press, Cambridge, MA, 2nd edition.

[6] Seymour Lipschutz (2009) Data Structure with C, Schaum Series, Tata McGraw-Hill Education.

9. ABOUT THE AUTHOR

Rehan Guha is a student of Computer Application. He has already invented offline security software "Data Security- Multi-layer Folder Lock with Hiding"[5]. His other two major works include "Biometric Ticketing System"[6] and "Biometric Voting Machine and System".

[5] Published in Patent journal issue no. 44/2014 on 31/10/14. [6] Issue no. 39/2015 on 25/09/15.

All works are under process for Patent by **Intellectual Property India**.