The Hungry Dog: An Efficient Algorithm for Searching from 2-D Plot.

Mohammad Shabaz*, Amit Sharma[1], Gourav Ajmera[2]

*Apex Institute of Technology, Chandigarh University, Mohali.

[1]Apex Institute of Technology, Chandigarh University, Mohali.

[2]Apex Institute of Technology, Chandigarh University, Mohali.

*mohdshabaze6819@cumail.in ,[1] khandelwal.amit.010@gmail.com ,
[2]goravajmera@gmail.com

Abstract

Searching is one of the required operation while implementing any data-structure or algorithm especially when we are talking about Sorting and Merging, without searching these operations are not possible. There are different searching algorithms such as linear search, binary search, sequential search, Grover's Search etc. which are used to search element from 1-D Plot. But when we are talking about 2-D Plot, Searching becomes quite complex as it takes at least O(n^2) of Time complexity. The Hungry Dog is basically a problem in which a dog is standing inside any cell of 2-D Plot and its food lies inside some other cell but the dog has a capability to smell to its adjacent cells and has to reach to its food by jumping to other cells. But there are certain conditions that the dog cannot jump to already traversed cell and the dog has to take minimum number of jumps. In this paper we have designed an algorithms associated with a formula which enables the dog to reach to its food by taking minimum number of jumps. This Algorithm also helps us to solve the travelling salesman problem by reducing its moves.

The Hungry Dog results to the discovery of novel searching algorithm for 2-D Plot.

Keywords: Linear, Binary, Sequential, Breadth First Search, Complexity Analysis, Hungry Dog.

Introduction

The Hungry Dog Algorithm finds the minimal path to reach to the destination. The Source and destination has already been assigned to it. Let us look at some of the previous searching techniques which are used to search required item from 1-D and 2-D Plot.

Linear Search: Linear search is the most basic technique to search a particular item in 1-D Plot. This technique is mainly used when the data is unsorted and the data-set is small. In this technique, we firstly set the item which we want to search and compare this item linearly with all the other items which are lying in the plot. When the set item matches with any of the lying item, the item is found in the plot [1]

Algorithm 1 shows linear search.

Algorithm 1: Linear search

```
1. int a[1...n], search;
2. for i:=1 to n do{
3. if(a[i]= = search){
4. search found;
5. }
6. End if
7. }
8. End for
```

The time complexity of Linear search in worst and average case would be O(n) while in best case it is O(1).

Binary Search: This algorithm works on the list in which the records of the list are sorted. Getting the middle element of the list and comparing it with the searched element to find the position of the searched element. If the searched element is found, write its position and if not found then return nothing. While performing this complete procedure, every time the list is divided into two parts.

i.e. middle=(First + Last) / 2.

Where, First=0, Last=size of list -1.

Algorithm 2 shows Binary Search.

Algorithm 2: Binary Search

```
1. Assuming First = 0, Last = Size-1,
   Searched element.
2. If Last < First, then return.
3. Calculate middle.
4. If middle element equals searched
   element, then stop.
5. If the searched element is low, then
6. First = searched element + 1.
7. else
8. Last = searched element - 1.
9. Go back to step 2.
```

The time complexity of Binary Search in best, worst and average case would be O(log n).

Sequential Search: Sequential Search algorithm can be implemented on an unsorted list. Starting from the beginning of the list till end, the aim is find the desired search element. A flag variable initialized to 0 is declared initially and whenever the searched element is found anywhere in the list, this flag returns the value 1.
Algorithm 3 shows Sequential Search.

Algorithm 3: Sequential Search.

```
1. Put flag to 0
2. Put First to 0
3. While (First<=N) and (flag = = 0){
4. If (List [First] = =Searched Element)
5. flag = true
6. else
7. First=First+1
8. }
9. If (flag = =0)
10. Searched Element is not present in
    List.
```

The time complexity of Sequential Search in best case is O(1), but in worst and average case would be O(n).

Breadth First Search: Initialize all the vertex of the tree or graph to 0. Starting from the root or first vertex of a tree or graph and comparing its value with the searched element, if the searched element is found, then return the value, else similarly traverse till the last vertex and mark the entire traversed vertex with 1, which shows that the vertex has already tackled or traversed [2].
Algorithm 4 shows Breadth First Search.

Algorithm 4: Breadth First Search.

```
    1. Let the starting vertex be X, at
       Level 0.
```

2. Find all vertices that are reachable from X, i.e., only those which are one edge away from X.
3. Mark these reachable vertices to be at Level 1.
4. Those vertices which do not have their assigned Level T set some value to them.
5. Mark the root vertex of the currently targeted vertex.
6. Find all those vertices which are a one edge away from those vertices which are at Level 1. These new class of vertices will be at Level 2.
7. Repeat the process till all the vertices are targeted.

The time complexity of Breadth First Search is O(V + E) where V denotes vertex and E denotes Edge. In worst case analysis E = O(V^2), then the complexity becomes O(V^2).

Depth First Search: Avoiding cycle in a graph, this searching algorithm traverse in the whole graph and mark the visited vertex until all the vertices are tackled.
Algorithm 5 shows Depth First Search.

Algorithm 5: Depth First Search

1. Initialize by setting any one of the vertices of graph/tree onto stack.
2. Add the top element of the stack to already visited list.
3. Establish the list of that vertex which are one edge away from visited vertex and add them to the visited list onto stack.
4. Repeat the steps 2 and 3 until the stack underflow occurs.

The time complexity of Depth First Search is O(V + E) where V denotes Vertex and E denotes Edge.

Literature Review and Related Work

Akanmu T. A., Olabiyisi S. O., Omidiora E. O. ,Oyeleye C. A., Mabayoje M.A. and Babatunde A. O. [3]: In this paper the authors finds the software complexity measurement after performing a detail study on breadth first search and depth-first search algorithms.
Both these algorithms were measured on Program Volume (V), the Program Effort (E), the Program Difficulty (D) and the Cyclomatic Number V(G) parameters using different programming languages.
Finding the objectivity between execution and performance of these algorithms, helps in quick and reliable decision making.
Muhammad Usman , Zaman Bajwa and Mudassar Afzal [4]: In this paper the authors have focus on the performance of different searching algorithms using C# programming language such as linear search, Binary search and brute force search and measured them in term of time complexity i.e. execution time of searching algorithm. The authors found that linear search is better in time complexity and brute force search is better in finding all search patterns.
Najma Sultana, Smita Paira, Sourabh Chandra and Sk Safikul Alam [5]: The authors evaluate the better searching algorithm after implementing them on sorted and unsorted list or record. The conclusion is that the particular searching algorithms works on particular nature of data, i.e. for sorted data Binary search, Jump search and interpolation search works better where as for unsorted nature of data, Linear Search performs better.

Er. Mohammad Shabaz and Er. Neha Kumari [6]: Searching some pattern from string is another application of Searching. The authors, in this paper compare different searching algorithms and create a new one which has better execution time as compared to others.

Implementation and Methodology of "The Hungry Dog"

Algorithm 6 Shows "The Hungry Dog"

Algorithm 6: The Hungry Dog

1. Let us take a 2-D matrix of size n*n.
2. Assume that the dog is at corner of matrix which is starting point of dog.
3. On every position in matrix, dog will check every adjacent cell from its position and if Bone found, return and give result.
4. Dog moves diagonally in given matrix from one end to opposite end in which every jump is from one diagonal position to adjacent diagonal position.
5. Two Ends of the Matrix are formed, High End and the Low End.
6. For High End:
   Row High Jump- make a jump in row skipping two columns from end position and start traversing diagonally in (row*row)matrix from one end to opposite end.
   Column High Jump-make a jump in column skipping two row from end position and start traversing diagonally in (1-column)*(1-column) matrix from one end to opposite end.
7. For Low End:
   Column High Jump- make a jump in column skipping two row from end position and start traversing diagonally in (1-column)*(1-column) matrix from one end to opposite end.
   Row High Jump-- make a jump in row skipping two columns from end position and start traversing diagonally in (row*row)matrix from one end to opposite end.

In case of n%4= =2 then check another end diagonal elements.

Results

The results showed in Table 1 shows the comparison of Hungry Dog Algorithm with Linear Search, Sequential Search and Breadth First Search.

MAT 8*8	linear	Sequential	Bfs	Hungry
00	2	1	7	18
01	3	2	10	15
02	4	3	9	15
12	13	11	8	14
23	23	20	8	5
24	24	21	8	5
33	32	28	7	3
77	72	64	7	11
61	50	50	2	1
07	8	8	21	7

Table 1: Shows the comparison among different searching algorithm on 8*8 Matrix on basis of their number of iterations.

Conclusion

The Hungry Dog is basically a problem in which a dog is standing inside any cell of 2-D Plot and its food lies inside some other cell but the dog has a capability to smell to its adjacent cells and has to reach to its food by jumping to other cells. The Hungry Dog algorithm is compared to different searching algorithms and we come to a conclusion that this algorithm performs better as compared to Linear Search, Sequential Search and Breadth First Search in average and worst case Scenario.

Acknowledgment
We thereby whole-heartedly acknowledge the people who have been connected to us especially Mr. Mohd. Rafiq, Mr. Vinayak Khajuria, Mr. Naresh Khandelwal, Mr. Omprakash Sharma, Mr. Vinod Ajmera.

References

[1] Cormen, T. H., Leiserson, C. E., and Rivest, R. L. (1990). Introduction to algorithms. Massachussets: MIT Press; New York: McGraw-Hill.

[2] Ellis Horowitz and Sartaj Sahni, “Fundamentals of Computer Algorithms”, Computer Science press, 1978.

[3] Akanmu T. A., Olabiyisi S. O., and et. al, “Comparative Study of Complexities of Breadth-First Search and Depth-First Search Algorithms Using Software Complexity Measures”, Proceedings of the World Congress on Engineering (WCE’10), London, 30 June-2 July 2010, 203-208.

[4] Muhammad usman , Zaman bajwa and Mudassar afzal, “Performance Analysis of Searching Algorithms in C# ”, International Journal for Research in Applied Science & Engineering Technology (IJRASET), Volume 2 Issue XII, December 2014.

[5] Najma Sultana, Smita Paira, Sourabh Chandra and Sk Safikul Alam, “A brief study and analysis of different searching algorithms”, IEEE International Conference on Electrical, Computer and Communication Technologies, 2017.

[6] Er. Mohammad Shabaz and Er. Neha Kumari, “Advance-Rabin Karp Algorithm for String Matching”, International Journal of Current Research Vol. 9, Issue, 09, pp.57572-57574, September, 2017.