International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

# Faster Graph Traversal Algorithm with Degree Level Ordering

**Shyma PV[1], Sanil Shanker K P*[1]**

**Submitted**:13/03/2024 **Revised**: 28/04/2024 **Accepted**: 05/05/2024

**Abstract:** Increasing the efficiency of algorithms for fundamental computations can have a wide-ranging impact on the performance of a large number of computations. A search algorithm is one such primitive task, occurring in many systems for identifying the shortest path from the start to the goal. The paper reports a fundamental searching algorithm for traversing a graph with degree level ordering. The adjacency list representation of the graph is used by the algorithm because it is effective in terms of storage and makes it simple to figure out the degree of each node. Without requiring any additional templates, the approach may be applied to both connected and disconnected graphs. The results highlight an efficient graph traversal algorithm and elucidated how the proposed algorithm can be used to solve Maze problem.

***Keywords:*** *Graph traversal, Degree Level Search algorithm, Ascendant Node First Search algorithm, Descent Node First Search algorithm.*

## 1. Introduction

It is becoming more and more crucial to analyze and comprehend social interaction data, relational data in general, complex engineered systems like the power grid and the Internet, communication data like email and phone networks, and biological systems using graph abstractions. These application fields frequently encounter graph-theoretic issues including identifying and rating significant entities, seeing unusual patterns or rapid changes in networks, locating strongly connected clusters of entities, and others. For issues like discovering spanning trees, shortest paths, biconnected components, matching, and flow-based computations in these graphs, traditional techniques are frequently used as solutions. A traversal is a methodical exploration of each vertex and edge in a graph. The primary objective is to address the fundamental task of searching algorithm by utilising the concept of degree of a node for efficient graph traversal. The proposed algorithm follows a degree level order when traversing the graph, whereas the conventional graph traversal algorithms BFS and DFS traverse the graph in the horizontal and vertical directions, respectively [1, 2, 3]. Thus, with respect to computational efficiency, the Degree Level Search (DLS) algorithm outperforms existing algorithms in terms of execution time. This computational procedure explores two search strategies, namely the Ascendant First Search and the Descent First Search. The algorithm is developed to overcome the constraints of current search algorithms, even though graph traversals such as Breadth First Search and Depth First Search form the basis for numerous advanced graph analysis tools. It is impossible to find a useless or ineffective path as the Breadth First Search checks levels by levels [4].

A stack is used in Depth First Search to hold a group of old vertices with prospective unexplored edges, and its complexity relies on the quantity of pathways [5]. The shortest path cannot be guaranteed by Depth First Search, and it cannot check for duplicate nodes. DFS is used for locating elements deeper in the ground, while BFS is preferred for locating components closer to the root [1]. Instead, we choose the node with the highest or lowest degree while traversing a node to traverse the graph.

We formulate the graph searching algorithm discovery procedure as a single-player game, called Maze solution. At each step of the Maze game, the player selects a path from the entrance to the exit without hitting the wall. The agent can only see cells in the Maze that are nearest to it, making this a hard game. We create a DLS agent to find the path with highest number of entrances. We create a faster graph traversal algorithm with degree level ordering to choose the maximum path with other nodes in order to solve the Maze game. The Degree Level Search Algorithm selects the vertex that has the highest or lowest edge, and then it traverses all of neighboring vertices of the selected vertex. In order to choose the node in the order of degree, the algorithm keeps an additional array called Degree. When the node is visited, the Degree element of the node is set to zero and all of the Degree elements of the neighboring nodes will be reduced by one. Based on the selection of node with the maximum or minimum degree, the degree level search algorithm can be classified as Ascendant Node First Search Algorithm and Descent Node First Search Algorithm (Fig 1).

[1]*Department of Information Technology, Kannur University, Kerala, India*
** Corresponding Author Email: sanil@kannuruniv.ac.in*

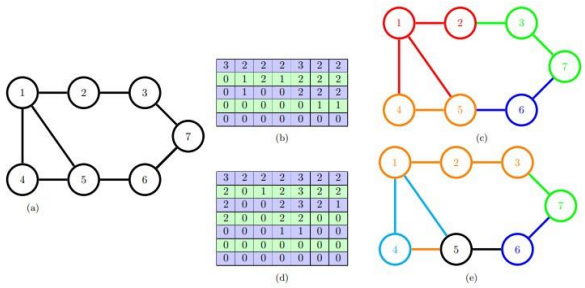


**Fig 1.** Degree Level Search algorithm procedure. a) The initial graph and the visited nodes and adjacent nodes in the Ascendant First Search Algorithm. b) The main constraint of the algorithm- the array Degree and changes the value of elements in the array while visiting the nodes c) The visited nodes and adjacent nodes in the Ascendant First Search algorithm. d) The array Degree and changes the value of elements in the array while visiting the nodes in descending order. e) The visited nodes and adjacent nodes in the Descent First Search algorithm.

## 2. Algorithm for traversal with Degree Level Ordering

The Degree Level Traversal Algorithm implicitly searches all the vertices from a given source vertex of a graph G = (V, E). This computation is achieved by traversing in the order of degree of nodes of the graph.

### 2.1 Ascendant Node First Search algorithm

We cast the problem of finding a method for traversing a graph using adjacency list. The algorithm initializes with the source node, then find out all of the adjacent nodes of the source node and continue with the ascendant node as the current node. In Ascendant Node First Search algorithm (ANFS algorithm), there is an additional array to store the degrees of the vertices and the main constraint in the algorithm is to check whether the array becomes empty.

**Algorithm 1**
**I/P:** G= (V, E), R as source node
**O/P:** $v_1, v_2, v_3, \ldots v_n$ according to the ascending order of the degree
**Data Structure:** Linked List
**Parameters** : Degree, **Min**(Degree)
ANFS(R)

1. for all neighbours W of R in Graph G
2. $V_{max}$:= W ( **Min** (DEGREE))
3. Degree [Next]:= Degree [Next]-1
4. while (Degree[i] = 0 or Adj[]= Visited)
5. for i from 1 to n
6. $V_{max}$ := V ( **Min** (DEGREE))
7. Current := $V_{Min}$
8. Degree [$V_{Min}$] : = 0
9. $V_{Min}$:= W ( **Min** (DEGREE))
10. Next: = Adj ($V_{Min}$)
11. Set $V_{max}$:=Visited
12. Degree [$V_{Min}$] := 0
13. Degree [Next]:= Degree[Next]-1

### 2.2 Descent Node First Search algorithm

In Descent Node First Search algorithm **(**DNFS algorithm), instead of starting from the Ascendant node, it starts from the Descent node.

**Algorithm 2**
**I/P:** G= (V, E), R as source node
**O/P:** $v_1, v_2, v_3, \ldots v_n$ according to the ascending order of the degree
**Data Structure:** Linked List
**Parameters** : Degree, **Max**(Degree)
ANFS(R)

1. for all neighbours W of R in Graph G
2. $V_{max}$:= W ( **Max** (DEGREE))
3. Degree [Next]:= Degree [Next]-1
4. while (Degree[i] = 0 or Adj[]= Visited)
5. for i from 1 to n
6. $V_{max}$ := V ( **Max** (DEGREE))
7. Current := $V_{max}$
8. Degree [$V_{max}$] : = 0
9. $V_{max}$:= W ( **Max** (DEGREE))
10. Next:= Adj ($V_{max}$)
11. Set $V_{max}$:=Visited
12. Degree [$V_{max}$] := 0
13. Degree [Next]:= Degree[Next]-1

## 3. Maze path solution

One of the most frequent issues in daily life is figuring out the best path from one place to another. It affects us when we move around a city, distribute things, clean a house, and in many other situations[6, 7]. Many technological advancements have been made to simplify the solution to this issue. However, when the area being explored is unknown, the destination is not clearly defined, or there are barriers and closed paths in the way, thus the route planning becomes a particularly challenging one[6,8,9]. An agent operates the Maze solving issue and uses exploration strategies to discover a way out. We cast the problem of finding the path or collection of paths, typically from an entry to exit based on the graph analytics techniques. An undirected graph can be used to describe a Maze by having vertices for the beginning, end, dead ends, and spots where more than one path can be selected [10,11]. The vertices are then connected in accordance with the paths in the Maze. Mazes can be created in a range of densities, from extremely sparse to dense. There are not many

walls in a sparse Maze; instead, there are a lot of open spaces, which creates a variety of paths that connect one junction to a distant junction [12].

**Procedure for the Maze Game.**
1. Select the initial cell and mark it as visited.
2. Assign the degree of visited cell to zero
3. Find all cells connected to the initial cell.
   a) Select the cell with maximum number of entrances and mark it as visited.
   b) Decrease the degree of the neighbouring cell by one.
   c) while there is no other movement
      i) Select the next neighbouring cell with strongly connected.
4. For each neighbouring fields repeat step 2 and 3.
5. If the last cell is the destination then
6. End of the algorithm /* there is a way out of the maze*/.
7. end if

## 4. Method

The following instructions outline the steps for engaging in the Maze game: In the Degree Level Search algorithm, the cell located at [0, 0] is analogous to position S in the game.

At every iteration of the game, the agent modifies the adjacent cells of the visited location and the count of available pathways of the neighboring cells. As demonstrated in Algorithms 1 and 2, the DLS algorithm employs an adjacency list data structure to maintain the graph's information, and it initiates its traversal from the node with the maximum or minimum degree. The array "Degree" can be utilized to store the degree of nodes. This array is a constituent of the graph data structure recommended by Algorithms 1 and 2, along with the vertex list, visited list, and edge list. The nodes in the Maze graph will be represented with array of two elements. When the agent at node (i, j), neighbors are: (i−1, j), (i+1, j), (i, j−1), (i, j+1). Every time the agent must select between the available directions, the value of (i, j) is updated with the value of the relevant Maze cell. Setting the degree of the visited cells to zero and decrementing the degree of surrounding cells by one will avoid backtracking as the agent moves through the Maze cells. The algorithm will keep running till the goal is met (Fig 2).

**Fig 2.** Depicts the output of the Maze solution using the DLS algorithm

Theorem **1.**

Consider the graph G = (V, E), where V represents the set of all vertices $\{V_1,V_2,\ldots, V_n\}$, and E represents the set of all edges $\{E_1,E_2, ..,.E_m\}$. Let $V_i$ denote a set of vertices connected by an edge to $V_j$, where i=j. Then, the degree of a given vertex $V_i$, denoted as $\deg(V_i)$, is equal to the cardinality of set $V_i$, i.e.,$\deg(V_i)=|V_i|$.

Proof.

Let $V_i$ be a set of vertices that are connected by an edge to $V_j$. By the definition of vertex degree, each edge contributes precisely one unit of degree to the corresponding vertex. Let $V_j$ be an element of the set $V_i$. It follows that each element in the set $V_i$ contributes one degree to the vertex $V_i$. Let $V_i$ be a vertex in a graph. We can define the set $V_i$ as the set of all elements adjacent to $V_j$. Therefore, the total number of elements in the set $V_i$ provides the degree of the vertex $V_i$.

Furthermore, we can extend this concept to characterize the degree of $V_i$ within the context of the entire graph G. The degree of a vertex $V_i$ is influenced by the connectivity of its adjacent vertices, as each connection contributes to the degree of Vi. This relationship elucidates the significance of vertex degree in understanding the structural properties of the graph and its implications for various graph algorithms and analyses.

Corollary.

Let $|V_1| = n_1$, $| V_2| = n_2$,…… $|V_i| = n_i$……..$|V_n| = n_n$ if $n_i$ is greater than all other $n_j$, where j = 1,2,…n and $i \neq j$ then $V_i$ is the vertex with maximum degree.

Theorem **2**.

Let G = (V, E) be the graph where $V = \{V_1, V_2,.., V_n\}$ be the set of all vertices and $E = \{E_1, E_2, .., E_m\}$ be the set of all edges. A sequence of vertices $v_1, v_2, v_3, \ldots v_n$ traversed with respect to the order of degree of vertices with computational time O(m + n logn).

Proof.

Given a graph G = (V, E), where V represents the set of all vertices and E represents the set of all edges. Let G be a graph with m edges. In the first step of the algorithm, we partition G into subsets based on the number of edges. This partitioning can be done in $O(m^2)$ . Theorem 1 says that the time complexity for determining the degree is O(n). By virtue of the corollary, it follows that the selection of the vertex with the highest degree can be accomplished in O(n). Based on the analysis, the time complexity of the searching the nodes in the adjacency list is asymptotically equal to O(n logn). Hence the total computational time is equivalent to O(m+ nlogn).

### 4.1 Discovery of Graph Searching Algorithm

The algorithm for searching the degree level is discovered by representing the graph as an adjacency list and processing it in the order of the node's degree. The determination of vertex degrees can be readily achieved through the utilization of the adjacency list representation. The Degree Level Search algorithm employs a strategy of selecting vertices with the highest or lowest edge and subsequently traversing all vertices in their proximity. The analysis will involve determining the vertex degree of every adjacency node connected to the source node, followed by the processing of each of these adjacency nodes. The degree of a given node that has been visited is reduced to zero, while the degrees of its neighboring nodes are decreased by one. The algorithm iteratively examines the degree of each node in a given list, allowing for its direct application to disconnected graphs.

### 4.2 Computational complexity of the Degree Level Search algorithm

The first stage of the methodology involves partitioning the complete graph based on the solvability of its edges, which necessitates a time complexity of O(m). The algorithm selects the vertex with the maximum degree in O(n). At the conclusion of the procedure, the entirety of the graph is partitioned into distinct sections, and solely

those sections are subjected to further processing. The Degree Level Searching algorithm possesses an average time complexity of O(m+ nlogn).

## 5. Experiment

The algorithm implemented in the Maze domain, where an agent needs to navigate from top left to bottom right through a maze made up of barriers and open areas. We implemented Maze solver in Python 3.9.0 (tags/v3.9.0:9cf6752) [MSC v.1927 64 bit (AMD64)] on win32. The experiments were run on Microsoft Windows 10 Pro with Processor Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz, 1800 MHz, 4 Core(s), 8 Logical Processor(s) and 6 MB L2 and L3 cache, respectively.

### 5.1 Comparison of Degree Level Search (DLS) algorithm with BFS and DFS

The Breadth First Search algorithm is a widely used method for analysing graphs. It involves traversing all vertices in a graph, starting from a specified source vertex, and exploring all levels of the graph. The Degree Level Search algorithm initiates its search by identifying the vertex with the highest number of edges, rather than commencing from the root node. It then proceeds to explore all adjacent vertices of the selected vertex. In the context of graph theory, both the efficiency of storage is high when using an adjacency list due to the requirement of storing edge values. Within the context of Degree Level Search algorithm, an adjacency list is utilized as a data structure to contain the values pertaining to the edges. In the context of Breadth First Search, it is necessary to utilize a queue data structure to maintain a record of the child nodes that have been examined but not yet traversed. DFS algorithm employs a stack data structure to maintain a set of traversed vertices that may contain unexplored edges. Depth First Search and Breadth First Search algorithms are incapable of identifying the location of a node within a disconnected graph. The DLS algorithm utilises an auxiliary array to store and compare the degrees of the nodes in the main array for node selection. This feature enables its applicability to disconnected graphs. The data presented in Fig 3 illustrates the distinctions in the duration of execution for implementing randomly generated graph with 1000 nodes.

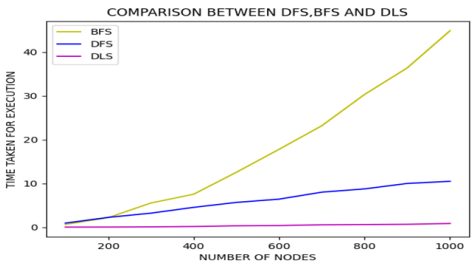


**Fig 3** Displays a comparison of the execution time for Depth-First Search (DFS), Breadth-First Search (BFS) and Degree Level Search (DLS) algorithms

## 6. Conclusion and future work

In this paper we proposed the graph traversal algorithm with degree level ordering. The result explores the efficiency of the Degree Level Search algorithm by solving Maze problem. The method that developed is being compared to the Depth First Search and Breadth First Search algorithms. In future work, the Degree Level Searching algorithm's ability to expedite path discovery apply on a variety of problems including autonomous driving, warehouse robots, and biological systems using graph abstractions and in other graph analytical applications.

## Author contributions

**Shyma PV:** Investigation, Methodology, Literature review, Writing- Original draft preparation.

**Sanil Shanker KP:** Investigation, Methodology, Writing-Reviewing.

## Conflicts of interest

The authors declare no conflicts of interest.

## References

[1] Maharshi J. Pathak, Ronit L. Patel, Sonal P. Rami,2018, "Comparative Analysis of Search Algorithms", International Journal of Computer Applications (0975 – 8887) Volume 179 – No.50.

[2] Samanta, D, 2004, Classic Data Structures, 9788120318748, Prentice Hall India Pvt . Limited, Second edition.

[3] Cheng, Y., Park, J., & Sandhu, R.2016, An Access Control Model for Online Social Networks Using User-to-User Relationships. IEEE Transactions on Dependable and Secure Computing, 13(4), 424–436 .

[4] Aho. A.V. and Hopcroft J.E. and Ullman J.D ,1974, The Design and Analysis of Computer Algorithms, 9780201000290, 74003995, Addison-Wesley series in computer science and information processing, Addison-Wesley Publishing Company, Fourth Edition .

[5] Galimberti, E., Bonchi, F., Gullo, F., & Lanciano, T, 2020, Core Decomposition in Multilayer Networks. ACM Transactions on Knowledge Discovery from Data, 14(1), 1–40 .

[6] Camilo Alaguna and Jonatan Gomez. 2018, Maze Benchmark for Testing Evolutionary Algorithms. In GECCO '18 Companion: Genetic and Evolutionary Computation Conference Companion, July 15–19, Kyoto, Japan. ACM, New York, NY, USA, 8 pages.

[7] G. Klančar, S. Blažič, and A. Zdešar.2017, C2-continuous path planning by combining bernstein-

bézier curves. ICINCO - Proceedings of the 14th International Conference on Informatics in Control, Automation and Robotics 2.

[8] Paul Hyunjin Kim, Jacob Grove, Skylar Wurster, and Roger Crawfis August 26–30, 2019,Design-Centric Maze Generation. In The Fourteenth International Conference on the Foundations of Digital Games (FDG'19), San Luis Obispo, CA, USA. ACM, New York, NY, USA, 9 pages.

[9] Chengshan Qian, Xinfeng Shen, Yonghong Zhang, Qing Yang, Jifeng Shen, and Haiwei Zhu. 2017, Building and Climbing based Visual Navigation Framework for Self-Driving Cars. Mobile Networks and Applications 1–1

[10] John R. Koza.1992, Genetic Programming On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge .

[11] Alexander Schrijver,2010, On the History of the Shortest Path Problem, Mathematics Subject Classification: 01A60, 05-03, 05C38, 05C85, 90C27.

[12] Omidshafiei, S., Tuyls, K., Czarnecki, W.M. et al. 2020, Navigating the landscape of multiplayer games. Nature Commun **11**,5603.