# PSA Final Portfolio

## DFS(Depth First Search)

### Arjun Nandkishor Tiwari(002176236)

GitHub Link :- <https://github.com/ArjunNandkishorTiwari/INFO-6205-final-project>

YouTube video link for teaching DFS (by me) :- <https://www.youtube.com/watch?v=wiO8L-9sr7Y&t=2s>

YouTube video link for visualization(by me) :- <https://www.youtube.com/watch?v=izbYVKHJ99E>

**Definition:-**

“Depth-first search refers to an algorithm for navigating or searching through tree or graph data structures (DFS). The algorithm starts at the root node (in the case of a graph, an arbitrary node is chosen to serve as the root node) and investigates each branch as far as it can before turning around. Additional memory, typically a stack, is required to keep track of the nodes found thus far along a particular branch to aid in graph backtracking.”

**Attributes of Depth First Search** :-

DFS time and space analysis has many different applications. Theoretical computer science frequently employs DFS, which traverses a graph in time display style O(|V|+|E|)O(|V| + |E|),where display style |V||V| is the number of vertices and display style |E||E| is the number of edges. The graph size of this is linear. The stack of vertices on the current search path and the set of previously visited vertices are both stored in these applications using the worst-case space display style O(|V|)O(|V|). As a result, in this situation, the time and space limitations are the same as for breadth-first search, and the choice between these two algorithms is therefore more impacted by the variations in their performance than by their relative complexity.

The graph that must be visited for DFS applications in specific domains, such as finding answers in artificial intelligence or web crawling, is frequently either too large to explore completely or infinite (DFS may suffer from non-termination). In these scenarios, search is frequently limited to a certain depth due to limited resources such as memory or disk space. Typically, data structures are not used to keep track of the set of all previously visited vertices.

Although the time required to perform a search to a limited depth is still linear in terms of the number of expanded vertices and edges (even though this number is not equal to the size of the entire graph because some vertices may be searched more than once and others not at all), the space complexity of this variant of DFS is only proportional to the depth limit, and as a result, is significantly less than the space required to perform a search to the same depth using breadth-first search. Additionally, heuristic methods for selecting a branch that seems likely-looking work significantly better for these applications when using DFS. If an adequate depth

limit is not specified beforehand, iterative deepening depth-first search uses DFS repeatedly with a series of expanding boundaries Due to the geometric rise of the number of nodes per

level, iterative deepening in the artificial intelligence mode of analysis only increases the running time by a constant factor over the scenario in which the right depth limit is known.

DFS can also be used to generate a random sample of graph nodes. Incomplete DFS, like incomplete BFS, is biased toward high degree nodes.

**Example with visualization(Refer the video along with the document): -**

Assuming that the left edges in the presented graph are chosen before the right edges and that the search remembers previously visited nodes and won't repeat them, a depth-first search starting at node A will visit the nodes in the following order: A, B, D, F, E, C, G. (because this is a tiny graph). The edges traversed during this search constitute a tree, a structure with important applications in graph theory. Without keeping note of previously visited nodes, the same search will always visit nodes in the following order: A, B, D, F, E, A, B, D, F, E, etc., never reaching C or G.

**PseudoCode :-**

Method DFS(G, v):

label v is visited

for all directed edges in G, adjacent from v to w do

If vertex w is not labeled as visited

call DFS (G, w)

The first neighbor of v visited by the recursive variation is the first in the list of adjacent edges, whereas the first neighbor of v visited by the iterative variation is the last in the list of adjacent edges. These two DFS variations visit the neighbors of each vertex in the opposite order. The nodes in the example graph that will be visited by the recursive implementation are A, B, D, F, E, C, and G. The nodes A, E, F, B, D, C, and G will receive the non-recursive implementation.

In two ways, the non-recursive implementation resembles but differs from breadth-first search

It uses a stack instead of a queue and defers determining whether a vertex has Wait until the vertex is popped from the stack to perform this check rather than doing it before adding it.

The depth-first search algorithm is created by swapping the queue for a stack in the breadth-first search algorithm when G is a tree. For general graphs, a breadth-first search algorithm, albeit a little out of the ordinary, can be produced by substituting a queue for the stack in the iterative depth-first search implementation.

Another potential iterative depth-first search implementation uses a stack of iterators of a node's list of neighbors rather than a stack of nodes. The outcome is the same as with recursive DFS.

**Complexity :-**

John Reif investigated the computational difficulty of the DFS. If a graph has displaystyle G"G, then let displaystyle O=(v 1,"dots,"v n") be the ordering produced by the common recursive DFS algorithm. This technique is known as lexicographic depth-first search ordering. John Reif wondered how hard it would be to compute the lexicographic depth-first search ordering given a graph and a source. The decision version of the problem tests whether some vertex u occurs

before some vertex v in this order, making it P-complete and "a nightmare for parallel processing."A depth-first search ordering can be computed by a randomized parallel algorithm belonging to the RNC complexity class (not always the lexicographic one).

Applications :-

* Find interconnected components
* Sorting using Topological approach
* Finding bridges in a graph
* Planarity testing
* Random Maze generation (Visualization provided in the YouTube video link above
* Finding bi – connectivity in graphs