DEPTH-FIRST SEARCH ALGORITHM

- The depth-first search algorithm progresses by expanding the starting node of G and then going deeper and deeper until the goal node is found, or until a node that has no children is encountered. When a dead-end is reached, the algorithm backtracks, returning to the most recent node that has not been completely explored.
- In other words, depth-first search begins at a starting node A which becomes the current node. Then, it examines each node N along a path P which begins at A. That is, we process a neighbour of A, then a neighbour of neighbour of A, and so on. During the execution of the algorithm, if we reach a path that has a node N that has already been processed, then we backtrack to the current node. Otherwise, the unvisited (unprocessed) node becomes the current node.
- The algorithm proceeds like this until we reach a dead-end (end of path P). On reaching the dead- end, we backtrack to find another path P¢. The algorithm terminates when backtracking leads back to the starting node A. In this algorithm, edges that lead to a new vertex are called discovery edges and edges that lead to an already visited vertex are called back edges.

STEPS

Step 1: SET STATUS = 1 (ready state) for each node in G

Step 2: Push the starting node A on the stack and set its STATUS = 2 (waiting state)

Step 3: Repeat Steps 4 and 5 until STACK is empty

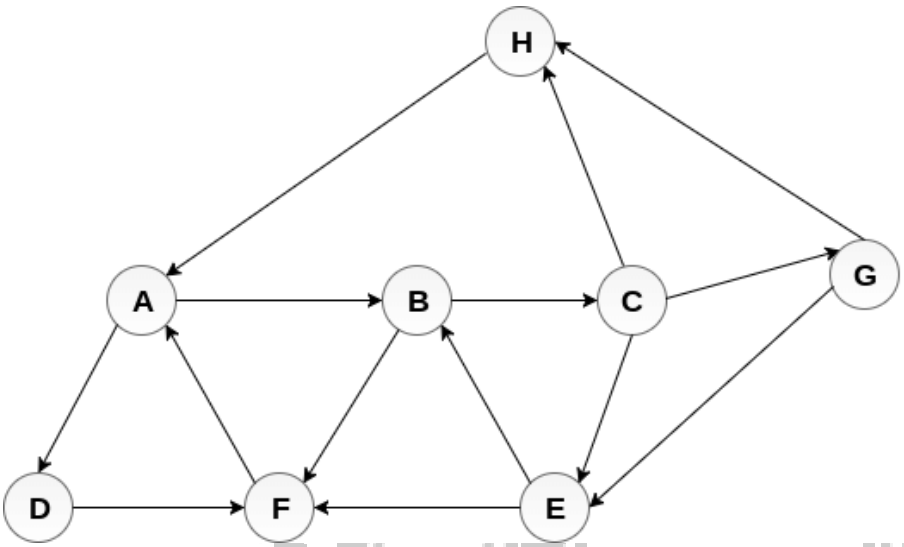
Step 4: Pop the top node N. Process it and set its STATUS = 3 (processed state)

Step 5: Push on the stack all the neighbours of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2 (waiting state) [END OF LOOP]

Step 6: EXIT

EXAMPLE

Consider the graph G along with its adjacency list, given in the figure below. Calculate the order to print all the nodes of the graph starting from node H, by using depth first search **(DFS) algorithm.**



Adjacency Lists

A : B, D
B : C, F
C : E, G, H
G : E, H
E : B, F
F : A
D : F
H : A

Solution :

Push H onto the stack

STACK : H

POP the top element of the stack i.e. H, print it and push all the neighbours of H onto the stack that are is ready state.

Print H

STACK : A

Pop the top element of the stack i.e. A, print it and push all the neighbours of A onto the stack that are in ready state.

Print A

Stack : B, D

Pop the top element of the stack i.e. D, print it and push all the neighbours of D onto the stack that are in ready state.

Print D

Stack : B, F

Pop the top element of the stack i.e. F, print it and push all the neighbours of F onto the stack that are in ready state.

Print F

Stack : B

Pop the top of the stack i.e. B and push all the neighbours

Print B

Stack : C

Pop the top of the stack i.e. C and push all the neighbours.

Print C

Stack : E, G

Pop the top of the stack i.e. G and push all its neighbours.

Print G

Stack : E

Pop the top of the stack i.e. E and push all its neighbours.

Print E

Stack :

Hence, the stack now becomes empty and all the nodes of the graph have been traversed.

The printing sequence of the graph will be :

H → A → D → F → B → C → G → E

Applications of Depth-First Search Algorithm

Depth-first search is useful for:

- Finding a path between two specified nodes, u and v, of an unweighted graph.

- Finding a path between two specified nodes, u and v, of a weighted graph.
- Finding whether a graph is connected or not.
- Computing the spanning tree of a connected graph.