**American International University-Bangladesh (AIUB)**

FACULTY OF SCIENCE & INFORMATION TECHNOLOGY

***Department of Computer Science***



**CSC2211 Algorithms |**

**LAB MANUAL 07**

**TITLE|** Application of DFS

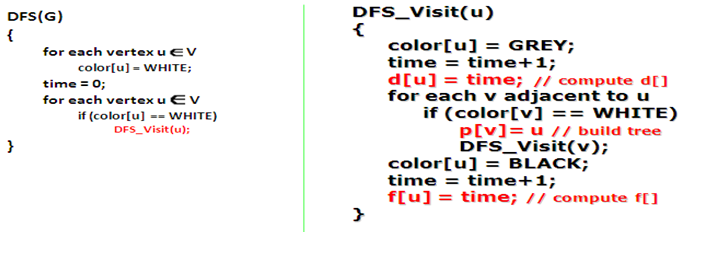
**PREREQUISITE |** Knowledge of Graph Representation| DFS | Linked List

**OBJECTIVE |** Learn and Implement Topological Sort & Strongly Connected Components

**THEORY |** DFS|Topological Sort| Strongly Connected Components

**DEPTH FIRST SEARCH (DFS)**

**Depth-first search**, or **DFS**, is a way to traverse the graph. Initially it allows visiting vertices of the graph only, but there are hundreds of algorithms for graphs, which are based on DFS. Therefore, understanding the principles of depth-first search is quite important to move ahead into the graph theory. The principle of the algorithm is quite simple: to go forward (in depth) while there is such possibility, otherwise to backtrack. The pseudocode for Depth-first search (DFS) is given below-



**APPLICATION OF DFS:**

Depth-first search (DFS) is an algorithm (or technique) for traversing a graph. Following are the problems that use DFS as a building block.

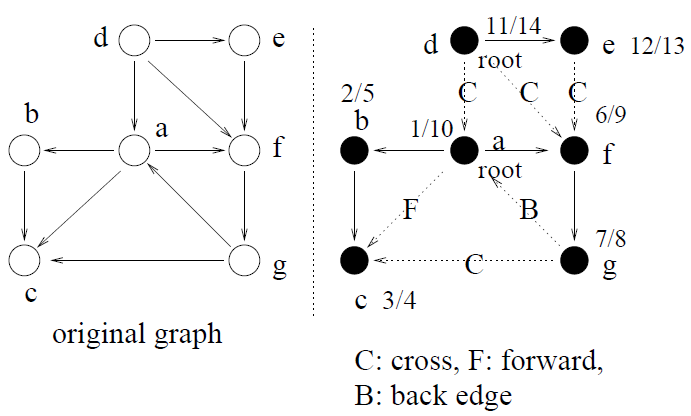
* Detecting cycle in a graph
* Path Finding
* Topological Sorting
* Finding Strongly Connected Components of a graph
* To test if a graph is bipartite
* Minimum Spanning Tree
* Solving puzzles with only one solution

**Cycle detection:** *Becomes back edge detection*

**Idea of Cycle detection:** A digraph is Cyclic if and only if any DFS forest yields back edges.

***Problem:*** Modify the DFS algorithm slightly to give an algorithm for cycle detection.

***Solution:*** This can always be done by first running the algorithm and assigning the ***d[u]*** and ***f[u]*** values and then running through all of the edges one more time, seeing if any of them are back edges.



**Topological sorting**

**Idea of Topological Sorting:** Run the DFS on the DAG and output the vertices in reverse order of finishing time.

**Input Description:** A directed, acyclic graph *G= (V, E)* (also known as a partial order).

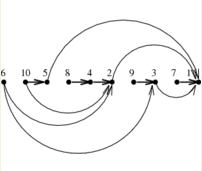
**Problem:** Find a linear ordering of the vertices of *V* such that for each edge *(i, j)* in *E*, vertex *i* is to the left of vertex *j*.

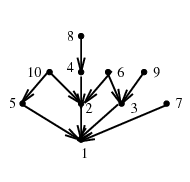
**The Algorithm:**

1. Run DFS (G), computing finish time ***f[u]*** for each vertex;

2. As each vertex is finished, insert it onto the front of a list;

3. Output the list.





**INPUT**

**OUTPUT**