CHANDIGARH UNIVERSITY
Discover. Learn. Empower.

NAAC GRADE A+
Accredited University

Experiment 2

Student Name: Rahul Saxena

UID: 24MCI10204

Branch: MCA (AI-ML)

Section/Group: MAM - 3(B)

Semester: II

Subject Name: Artificial Intelligence Lab [24CAP-674]

Aim: Implement Depth-First Search (DFS) Algorithm.

Steps:

- Represent the graph using an adjacency list or matrix.
- Create a Boolean array (or similar) to track visited nodes.
- Start from a source node and mark it as visited.
- Recursively visit all unvisited adjacent nodes.
- Backtrack when no unvisited adjacent nodes are left.

Algorithm:

Input:

- A graph G(V,E)G(V, E)G(V,E), where VVV is the set of vertices and EEE is the set of edges.
- A starting node S.

Output:

- A traversal of all reachable nodes from S.

Algorithm:

1. Initialize a stack (for iterative implementation) or use recursion (for recursive implementation).
2. Mark the starting node S as visited.
3. If using recursion:
    - Recursively visit all unvisited adjacent nodes.
4. If using iteration:
    - Push the starting node S onto the stack.
    - While the stack is not empty:
        - Pop the top node, mark it as visited, and push its unvisited neighbours onto the stack.
5. Continue until all reachable nodes are visited.

Source Code (Stack):

```
from collections import deque
def dfs(graph, start):
visited = set()
stack = [start]
while stack:
  node = stack.pop()
  if node not in visited:
    print(node, end=" ")
    visited.add(node)
```

```
      for meighbor in reversed(graph[node]):
        if meighbor not in visited:
          stack.append(meighbor)
if __name__ == "__main__":
  graph = {
    "A" : ["B", "D"],
    "B" : ["C", "F"],
    "C" : ["G", "E", "H"],
    "D" : ["F"],
    "E" : ["B"],
    "F" : ["A"],
    "G" : ["H", "E"],
    "H" : ["A"]
  }
start_node = 'A'
print("DFS traversal starting from node ", start_node, " : ", end="")
dfs(graph, start_node)
import networkx as nx
import matplotlib.pyplot as plt
G = nx.DiGraph()
nodes = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']
G.add_nodes_from(nodes)
edges = [
    ('H', 'A'),
    ('A', 'B'), ('A', 'D'),
    ('B', 'C'), ('B', 'F'),
    ('C', 'G'), ('C', 'E'), ('C', 'H'),
    ('E', 'F'), ('E', 'B'),
    ('D', 'F'),
    ('G', 'E'), ('G', 'H')
]
G.add_edges_from(edges)
pos = nx.spring_layout(G)
nx.draw(G, pos, with_labels=True, node_color='skyblue', edge_color='brown', node_size=2000, font_size=12, font_weight='bold')
plt.show()
```

Source Code (Recursion):

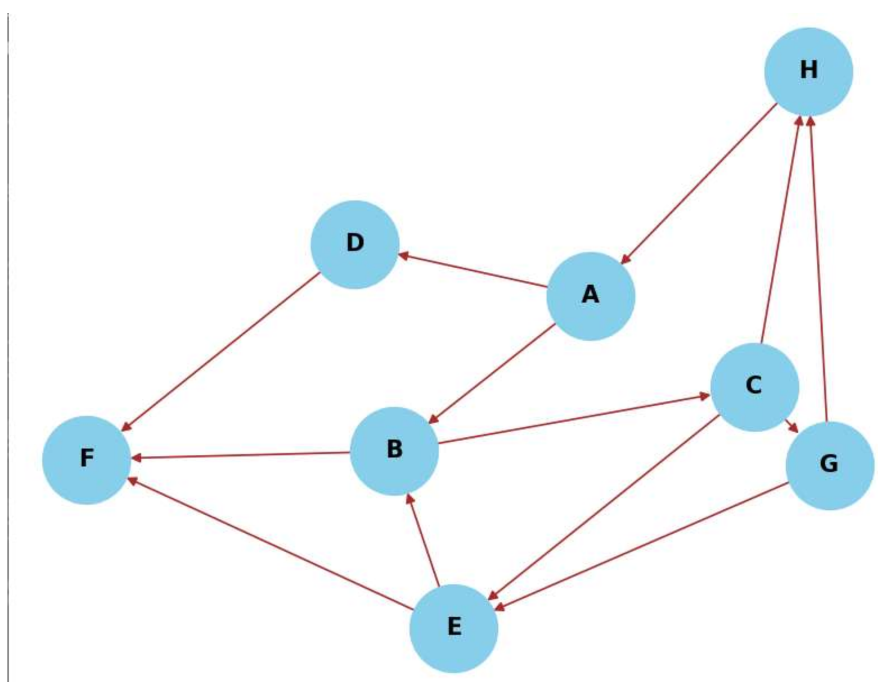
```
def dfs(graph, start, visited=None):
  if visited is None:
    visited = set()
  visited.add(start)
  print(start, end=" ")
  for neighbor in graph[start]:
    if neighbor not in visited:
      dfs(graph, neighbor, visited)
  return visited

if __name__ == "__main__":
  graph = {
    "A" : ["B", "D"],
    "B" : ["C", "F"],
    "C" : ["G", "E", "H"],
    "D" : ["F"],
    "E" : ["B"],
```

```
    "F" : ["A"],
    "G" : ["H", "E"],
    "H" : ["A"]
  }

start_node = 'A'
print("DFS traversal starting from node ", start_node, " : ", end="")
dfs(graph, start_node)
print()
import networkx as nx
import matplotlib.pyplot as plt
G = nx.DiGraph()
nodes = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']
G.add_nodes_from(nodes)
edges = [
    ('H', 'A'),
    ('A', 'B'), ('A', 'D'),
    ('B', 'C'), ('B', 'F'),
    ('C', 'G'), ('C', 'E'), ('C', 'H'),
    ('E', 'F'), ('E', 'B'),
    ('D', 'F'),
    ('G', 'E'), ('G', 'H')
]
G.add_edges_from(edges)
pos = nx.spring_layout(G)
nx.draw(G, pos, with_labels=True, node_color='skyblue', edge_color='brown', node_size=2000, font_size=12, font_weight='bold')
plt.show()
```

Graph:



Output Using “Stack”:

```
PS D:\MCA\Semester 2\AI Practice> & C:/Users/saxen/AppData/Local/Pro
S.py"
DFS traversal starting from node  A  : A B C G H E F D
PS D:\MCA\Semester 2\AI Practice>
```

Output Using “Recursion”:

```
PS D:\MCA\Semester 2\AI Practice> & C:/Users/saxen/AppData/Local/Pr
 2/AI Practice/DFS.py"
DFS traversal starting from node  A  : A B C G H E F D
```

Learning Outcome:

1. **Understanding Graph Representation**: You learn how to represent a graph using adjacency lists or matrices.
2. **Algorithm Design**: You understand the recursive nature of DFS and how to implement it iteratively using stacks.
3. **Traversal Techniques**: You learn how DFS explores all reachable nodes in a depth-oriented manner.
4. **Applications of DFS**:
    a. Cycle detection in graphs.
    b. Topological sorting.
    c. Connected components in a graph.
    d. Solving mazes and puzzles.
5. **Comparison with BFS**: You gain insight into how DFS differs from Breadth-First Search (BFS) in terms of traversal and memory usage.