

EX.NO:2

DATE:4/9/2024

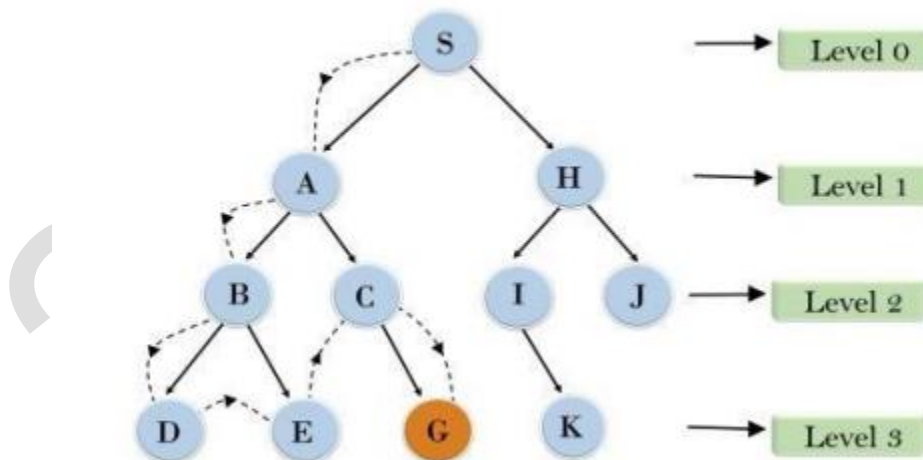
Reg.no:220701026

DEPTH-FIRST SEARCH

AIM: To implement a depth-first search problem using Python

- Depth-first search (DFS) algorithm or searching technique starts with the root node of graph G, and then travel deeper and deeper until we find the goal node or the node which has no children by visiting different node of the tree.
- The algorithm, then backtracks or returns back from the dead end or last node towards the most recent node that is yet to be completely unexplored.
- The data structure (DS) which is being used in DFS Depth-first search is stack. The process is quite similar to the BFS algorithm.
- In DFS, the edges that go to an unvisited node are called discovery edges while the edges that go to an already visited node are called block edges

Depth First Search



CODE:

```
def dfs_recursive(graph, start, visited=None):
```

```

    if visited is None:
        visited = set()
    visited.add(start)
    print(start)

    for neighbor in graph[start]:
        if neighbor not in visited:
            dfs_recursive(graph, neighbor, visited)

graph = {
    'A': ['B', 'C'],
    'B': ['A', 'D', 'E'],
    'C': ['A', 'F'],
    'D': ['B'],
    'E': ['B', 'F'],
    'F': ['C', 'E']
}

print("DFS Recursive:")
dfs_recursive(graph, 'A')

def dfs_iterative(graph, start):
    visited = set()
    stack = [start]

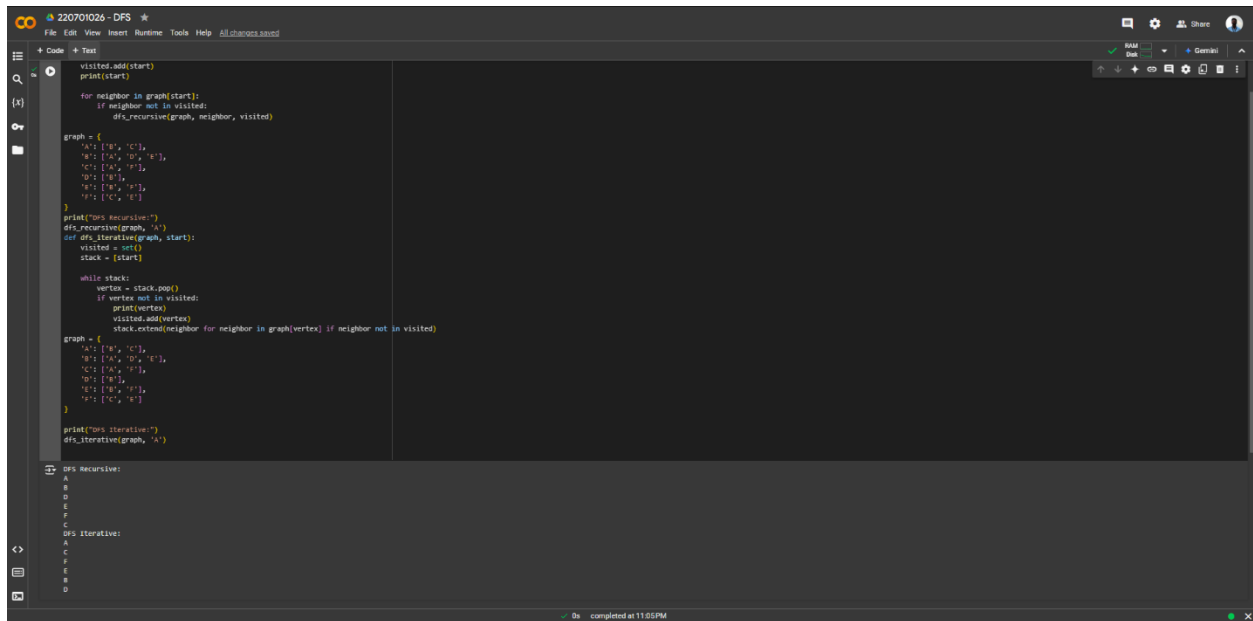
    while stack:
        vertex = stack.pop()
        if vertex not in visited:
            print(vertex)
            visited.add(vertex)
            stack.extend(neighbor for neighbor in graph[vertex] if
neighbor not in visited)

graph = {
    'A': ['B', 'C'],
    'B': ['A', 'D', 'E'],
    'C': ['A', 'F'],
    'D': ['B'],
    'E': ['B', 'F'],
    'F': ['C', 'E']
}

print("DFS Iterative:")
dfs_iterative(graph, 'A')

```

OUTPUT:



The screenshot shows a Jupyter Notebook interface with a dark theme. The top bar indicates the file name '220701026 - DFS'. The notebook contains two code cells. The first cell implements a recursive DFS algorithm, and the second cell implements an iterative DFS algorithm using a stack. Both algorithms operate on a graph with 8 vertices (A-H) and several edges. The recursive algorithm prints the vertices in the order A, B, D, E, C, F, H, G. The iterative algorithm prints the vertices in the order A, C, F, H, B, D, E, G. The bottom status bar shows 'completed at 11:05PM'.

```
visited.add(start)
print(start)

for neighbor in graph[start]:
    if neighbor not in visited:
        dfs_recursive(graph, neighbor, visited)

graph = {
    'A': ['B', 'C'],
    'B': ['A', 'D', 'E'],
    'C': ['A', 'F'],
    'D': ['B'],
    'E': ['B', 'F'],
    'F': ['C', 'E']
}

print("Dfs recursive:")
dfs_recursive(graph, 'A')

def dfs_iterative(graph, start):
    visited = set()
    stack = [start]

    while stack:
        vertex = stack.pop()
        if vertex not in visited:
            print(vertex)
            visited.add(vertex)
            stack.extend(neighbor for neighbor in graph[vertex] if neighbor not in visited)

graph = {
    'A': ['B', 'C'],
    'B': ['A', 'D', 'E'],
    'C': ['A', 'F'],
    'D': ['B'],
    'E': ['B', 'F'],
    'F': ['C', 'E']
}

print("Dfs iterative:")
dfs_iterative(graph, 'A')
```

Dfs recursive:
A
B
D
E
C
F
H
G

Dfs iterative:
A
C
F
H
B
D
E
G

completed at 11:05PM