DFS PROGRAMMING USING PYTHON:

CODE:

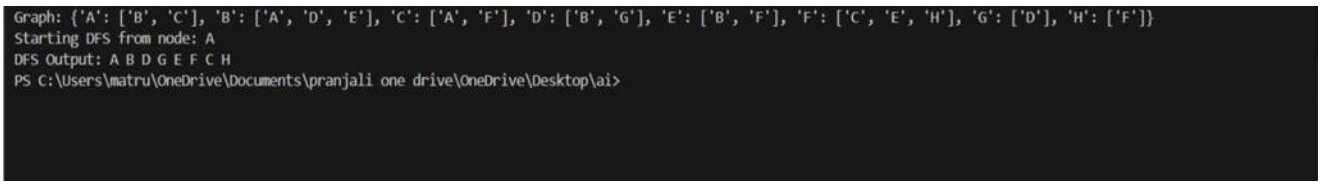
```
# DFS using python def
dfs(graph, start):
   visited = set() # to track visited nodes stack =
   [start] # stack to hold nodes to visit

   print(f"Graph: {graph}") # print the graph before traversal
   print(f"Starting DFS from node: {start}") print("DFS
   Output:", end=" ")

   while stack: node = stack.pop() # get last node in
      the stack if node not in visited:
         print(node, end=" ") # process node
         visited.add(node) # mark node as visited
         # add neighbors to the stack, reverse order to maintain correct traversal
         stack.extend(reversed(graph[node]))

graph = {
   'A': ['B', 'C'],
   'B': ['A', 'D', 'E'],
   'C': ['A', 'F'],
   'D': ['B', 'G'],
   'E': ['B', 'F'],
   'F': ['C', 'E', 'H'],
   'G': ['D'],
   'H':  ['F']  }
dfs(graph, 'A')
```

OUTPUT:



BFS PROGRAMMING USING PYTHON:

CODE:

```
# BFS using python #
BFS using python

from collections import deque def
bfs(graph, start):
   visited = set() # to track visited nodes queue = deque([start])
   # queue to hold nodes to visit print(f"Graph: {graph}") #
   print graph before traversal print(f"Starting BFS from node:
   {start}") print("BFS Output:", end=" ") while queue: node
   = queue.popleft() # get the first node in the queue if node
   not in visited: print(node, end=" ") # process node
   visited.add(node) # mark node as visited
        queue.extend(graph[node]) # add neighbors to the queue graph
= {
   'A': ['B', 'C'],
   'B': ['A', 'D', 'E'],
   'C': ['A', 'F'],
   'D': ['B', 'G'],
   'E': ['B', 'F'],
   'F': ['C', 'E', 'H'],
   'G': ['D'],
   'H':  ['F']  }
bfs(graph,
'B')
```

OUTPUT: