PROGRAM TITLE 08

DEPTH – FIRST SEARCH

AIM:

To Write the python program to implement DFS.

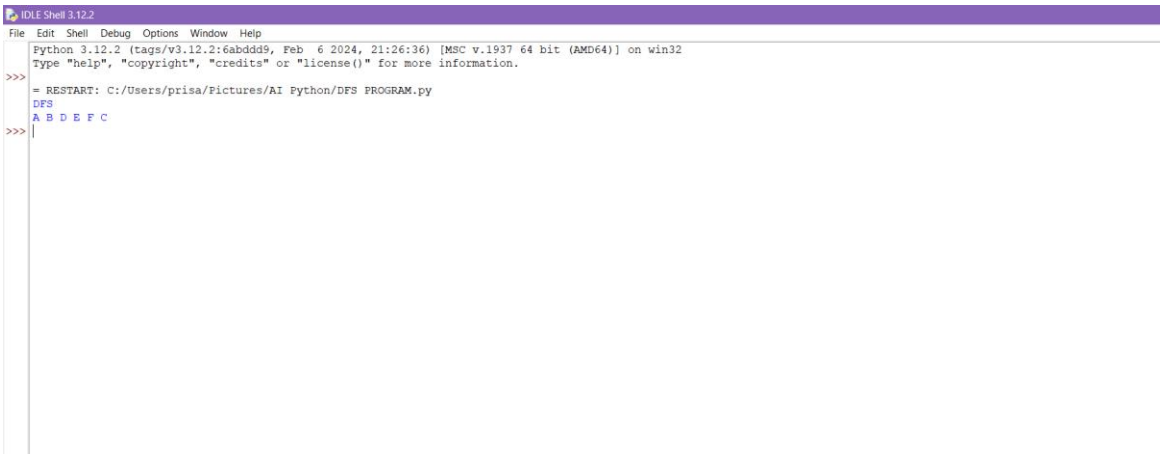
PROCEDURE:

1. **Initialize Data Structures:**
    - Create a stack (or use the call stack in recursive implementation) to keep track of nodes to visit.
    - Create a set or array to keep track of visited nodes.
2. **Start at the Initial Node:**
    - Push the initial node onto the stack (or call the recursive function with the initial node).
3. **Explore Neighbours:**
    - While the stack is not empty (or in the case of recursion, until all branches are explored):
        - Pop a node from the stack (or use the function call stack).
        - If the node has not been visited:
            - Mark the node as visited.
            - Process the node (e.g., print its value or perform some operation).
            - Push its unvisited neighbours onto the stack (or make recursive calls).
4. **Termination:**
    - The algorithm terminates when the stack is empty (or all branches are explored in the case of recursion).

```
class Graph:     def
__init__(self):
self.graph = {}

   def add_edge(self, u, v):
if u not in self.graph:
self.graph[u] = []
self.graph[u].append(v)

   def dfs_util(self, vertex, visited):
      visited.add(vertex)
print(vertex, end=" ")

      if vertex in self.graph:            for
neighbor in self.graph[vertex]:
if neighbor not in visited:
                  self.dfs_util(neighbor, visited)

   def dfs(self, start):
visited = set()
self.dfs_util(start, visited)

if __name__ == "__main__":
   g = Graph()
   g.add_edge(0, 1)
   g.add_edge(0, 2)
   g.add_edge(1, 2)
   g.add_edge(2, 0)
```

```
g.add_edge(2, 3)
g.add_edge(3, 3)

print("Depth First Traversal (starting from vertex 2):")
g.dfs(2)
```

OUTPUT:



RESULT:

Hence the program been successfully executed and verified.