PROGRAM TITLE 08

DEPTH – FIRST SEARCH

AIM:

To Write the python program to implement DFS.

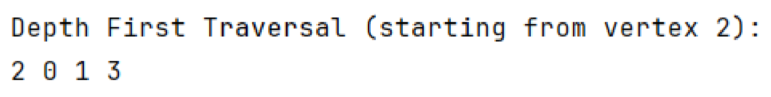
PROCEDURE:

1. Initialize Graph Class: Define a class called Graph to represent a graph. This class will have a dictionary to store the adjacency list representation of the graph.
2. Add Edges: Implement a method add_edge in the Graph class to add edges to the graph. If a vertex is not present in the graph, create a new list to store its neighbors and append the neighbor to the list.
3. DFS Utility Function: Define a utility function dfs_util within the Graph class to perform the actual depth-first traversal recursively. This function will print the visited vertices and mark them as visited to avoid revisiting.
4. Depth-First Search: Implement a method dfs in the Graph class to initiate the depth-first search traversal. This method initializes a set to store visited vertices and calls the dfs_util function with the starting vertex.
5. Example Usage: In the main section of the code, create an instance of the Graph class. Add edges to the graph using the add_edge method. Then, call the dfs method with the starting vertex to perform the depth-first traversal. Finally, print the result of the traversal.

CODING:

```
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:



```
Depth First Traversal (starting from vertex 2):
2 0 1 3
```

RESULT:

Hence the program been successfully executed and verified.