```python

# HSLU / ICS/AIML : Modul ADS : Algorithmen & Datenstrukturen
# Path   : uebung11/al/aufgabe02
# Version: Sun May  4 17:01:35 CEST 2025

from uebung11.graphs.graph import Graph
from uebung11.graphs.graph_adv import GraphADV
from uebung11.al.aufgabe02.depth_first_search import DepthFirstSearch


if __name__ == '__main__':

  graph = Graph() # without ADV
  #graph = GraphADV() # with ADV

  v_a = graph.insert_vertex('A')
  v_b = graph.insert_vertex('B')
  v_c = graph.insert_vertex('C')
  v_d = graph.insert_vertex('D')
  v_e = graph.insert_vertex('E')

  graph.insert_edge(v_a, v_b)
  graph.insert_edge(v_a, v_c)
  graph.insert_edge(v_a, v_d)
  graph.insert_edge(v_a, v_e)
  graph.insert_edge(v_b, v_c)
  graph.insert_edge(v_c, v_d)
  graph.insert_edge(v_c, v_e)

  dfs = DepthFirstSearch()
  dfs.search(graph)

  dfs.print_maps()


""" Session-Log:

DepthFirstSearch._search(): v = A
  e = A-B
    w = B
DepthFirstSearch._search(): v = B
  e = A-B
  e = B-C
    w = C
DepthFirstSearch._search(): v = C
  e = A-C
    w = A
  e = B-C
  e = C-D
    w = D
DepthFirstSearch._search(): v = D
  e = A-D
    w = A
  e = C-D
  e = C-E
    w = E
DepthFirstSearch._search(): v = E
  e = A-E
    w = A
  e = C-E
  e = A-C
  e = A-D
  e = A-E

DepthFirstSearch.print_maps():
Vertex-Map : {A=VISITED, B=VISITED, C=VISITED, D=VISITED, E=VISITED}
Edge-Map   : {A-B=DISCOVERY, A-C=BACK, A-D=BACK, A-E=BACK, B-C=DISCOVERY, C-D=DISCOVER
Y, C-E=DISCOVERY}

"""

```

```python

# HSLU / ICS/AIML : Modul ADS : Algorithmen & Datenstrukturen
# Path   : uebung11/al/aufgabe02
# Version: Sun May  4 17:01:35 CEST 2025

import enum


class DepthFirstSearch:

  class _VertexLabelDFS(enum.Enum):
    UNEXPLORED = enum.auto()
    VISITED = enum.auto()

  class _EdgeLabelDFS(enum.Enum):
    UNEXPLORED = enum.auto()
    DISCOVERY = enum.auto()
    BACK = enum.auto()

  def __init__(self):
    self._vertex_map = dict()
    self._edge_map = dict()
    self._graph = None

  def search(self, graph):
    self._graph = graph
    self._vertex_map = graph.get_dfs_vertex_map()
    self._edge_map = graph.get_dfs_edge_map()

    for u in graph.vertices():
      self._vertex_map[u] = DepthFirstSearch._VertexLabelDFS.UNEXPLORED
    for e in graph.edges():
      self._edge_map[e] = DepthFirstSearch._EdgeLabelDFS.UNEXPLORED
    for v in graph.vertices():
      if self._vertex_map.get(v) is DepthFirstSearch._VertexLabelDFS.UNEXPLORED:
        self._search(graph, v)

  def _search(self, graph, v):
    print("DepthFirstSearch._search(): v = " + str(v))
    self._vertex_map[v] = DepthFirstSearch._VertexLabelDFS.VISITED

    # TODO: Implement here ...


  def print_maps(self):
    self._graph.printing_maps(True)
    print("\nDepthFirstSearch.print_maps():")
    print("Vertex-Map : {", end = "")
    mappings = list()
    for v in self._vertex_map:
      mappings.append(v.__str__() + "=" + self._get_enum_name(self._vertex_map[v]))
    print(", ".join(mappings), end = "")
    print("}")
    print("Edge-Map   : {", end = "")
    mappings = list()
    for e in self._edge_map:
      mappings.append(e.__str__() + "=" + self._get_enum_name(self._edge_map[e]))
    mappings.sort()
    print(", ".join(mappings), end = "")
    print("}")
    self._graph.printing_maps(False)

  def _get_enum_name(self, enum_value):
    return enum_value.__str__().split(".")[1]


```