```
1
2  # HSLU / ICS/AIML : Modul ADS : Algorithmen & Datenstrukturen
3  # Path   : uebung10/al/aufgabe02
4  # Version: Fri Nov 17 16:52:43 CET 2023
5
6  import sys
7  from uebung10.al.aufgabe02.graph import Graph
8
9
10 if __name__ == '__main__':
11
12   graph = Graph()
13   u = graph.insert_vertex("U")
14   v = graph.insert_vertex("V")
15   w = graph.insert_vertex("W")
16   a = graph.insert_edge(u, v, "a")
17   b = graph.insert_edge(v, w, "b")
18   graph.print_graph()
19
20   if graph.opposite(u, a) is not v:
21     print("ERROR: v is not opposite of u!")
22     sys.exit(11)
23   if not graph.are_adjacent(v, w):
24     print("ERROR: v is not adjacent of w!")
25     sys.exit(22)
26
27
28 """ Session-Log:
29
30 Graph:
31 U -> (V,a)
32 V -> (U,a)(W,b)
33 W -> (V,b)
34
35 """
```



```
1
2  # HSLU / ICS/AIML : Modul ADS : Algorithmen & Datenstrukturen
3  # Path   : uebung10/al/aufgabe02
4  # Version: Fri Nov 17 16:52:43 CET 2023
5
6  from uebung10.graphs.graph_impl import GraphImpl
7
8  class Graph:
9
10   def __init__(self):
11     self._graph = GraphImpl()
12
13   def num_vertices(self):
14     return self._graph.vertex_count()
15
16   def num_edges(self):
17     return self._graph.edge_count()
18
19   def vertices(self):
20     return self._graph.vertices()
21
22   def edges(self):
23     return self._graph.edges()
24
25   def replace_in_vertex(self, v, element):
26     self._validate_vertex(v)
27     return v.replace(element)
28
29   def replace_in_edge(self, e, element):
30     self._validate_edge(e)
31     return e.replace(element)
32
33   def incident_edges(self, v):
34     self._validate_vertex(v)
35     return self._graph.incident_edges(v)
36
37   def end_vertices(self, e):
38     self._validate_edge(e)
39     end_vertices = e.endpoints()
40     return (end_vertices[0], end_vertices[1])
41
42   def opposite(self, v, e):
43     """
44     Raises a ValueError if edge is not incident to vertex.
45     """
46     self._validate_vertex(v)
47     self._validate_edge(e)
48
49     # TODO: Implement here ...
50
51     return None
52
53   def are_adjacent(self, v, w):
54     self._validate_vertex(v)
55     self._validate_vertex(w)
56     inc_v = self.incident_edges(v)
57     inc_w = self.incident_edges(w)
58
59     # TODO: Implement here ...
60
61     return None
62
63   def insert_vertex(self, element):
64     v = self._graph.insert_vertex(element)
65     return v
66
67   def insert_edge(self, v, w, element):
68     self._validate_vertex(v)
69     self._validate_vertex(w)
70     e = self._graph.insert_edge(v, w, element)
71     return e
```

```
72
73    def remove_vertex(self, v):
74      edges = list(self.incident_edges(v))
75      for e in edges:
76        self.remove_edge(e)
77      del self._graph._outgoing[v]
78      return v.element()
79
80    def remove_edge(self, e):
81      self._validate_edge(e)
82      end_vertices = self.end_vertices(e)
83      del self._graph._outgoing[end_vertices[0]][end_vertices[1]]
84      del self._graph._outgoing[end_vertices[1]][end_vertices[0]]
85      return e.element()
86
87    def _validate_vertex(self, v):
88      return self._graph._validate_vertex(v)
89
90    def _validate_edge(self, e):
91      return self._graph._validate_edge(e)
92
93    def print_graph(self):
94      print("Graph:")
95      for v in self._graph.vertices():
96        print(str(v), end = " -> ")
97        for e in self.incident_edges(v):
98          w = self.opposite(v, e)
99          print("(" + str(w) + ",", end = "")
100         print(str(e.element()) + ")", end = "")
101       print("")
102     print("")
103
104
```