Lab1: DirectedGraph

Graph with 5 vertices and 6 edges:
5 6

0 0 1

0 1 7

1 2 2

2 1 -1

1 3 8

2 3 5

The implementation uses an auxiliary class for the edges:

```
class Edge:
    def __init__(self, start: int, end: int, cost: int):
        self.__start = start
        self.__end = end
        self.__cost = cost
```

The graph is kept in memory as follows:

```
class DirectedGraph:
    def __init__(self):
        # Stores the inbound edges of each vertex
        self.__in = {}
        # Stores the outbound edges of each vertex
        self.__out = {}
        # Stores the vertices
        self.__vertices = {}
        # Stores the edges
        self.__edges = {}
```

__edges: dictionary – contains the edges, stored with their starting and ending vertices as key

{

(0, 0): Edge(0, 0, 1),

(0, 1): Edge(0, 1, 7),

(1, 2): Edge(1, 2, 2),

(2, 1): Edge(2, 1, -1),

(1, 3): Edge(1, 3, 8),

(2, 3): Edge(2, 3, 5),

}

__vertices: dictionary – contains the vertices, stored with their identifier as key

{

0: 0,

1: 1,

2: 2,

3: 3,

4: 4,

}

__in: dictionary – contains the inbound edges of each vertex, stored in lists at each vertex key

{

0: [__edges[(0, 0)]],

1: [__edges[(0, 1)], __edges[(2, 1)]],

2: [__edges[(1, 2)]],

3: [__edges[(1, 3)], __edges[(2, 3)]],

}

__out: dictionary – contains the outbound edges of each vertex, stored in lists at each vertex key

{

0: [__edges[(0, 0)], __edges[(0, 1)]],

1: [__edges[(1, 2)], __edges[(1, 3)]],

2: [__edges[(2, 1)], __edges[(2, 2)]],

}