Exercise 3

```
During the course, we have seen a recursive implementation for the DFS graph visit. There
is also a simple equivalent iterative implementation that uses a stack as an auxiliary data
structure (it runs just
like the BFS algorithm, except that we use the stack in place of the queue). However, there
is an algorithm that iteratively implements the DFS visit and does not use any auxiliary
data structure.
Design this algorithm and implement it.
```

The main objective of the exercise was to project and implemement an iterative version of the DFS algorithm without any support from external data structures.

We were allowed to edit the graph structure and so we did. We added the possibility to mark the edges by setting a internal variable.

We also added a new Edge type, which is called .

The algorithm explores the first undiscovered edge leading to a not-visited vertex, until a 1-degree vertex is found. That vertex (which we called in the algorithm) has only 1 edge, which is the used to set said vertex discovered.

When a 1-degree vertex is found, we need to go back in the graph through an edge already set as discovery or a back edge. Once this edges are used to find another undiscovered edge they are set as . The whole point of this is to avoid going back twice on the same edges. This would cause infinite loops.

Time complexity

The algorithm, in the worst case, is forced to go through each vertex in a connected component and for each vertex it has to run across each edge not once but twice, in the said case. (We ignored the computation needed to set all edges and vertices to unexplored in the time complexity calculation)

$$O(n(m+m)) => O(nm)$$