AFP - Assignment 1 - KColor

Maximo Garcia Martinez

1. Graph implementation

My implementation of the graph is a tuple of two list as follows: `{[1,2,3], [{1, 2}]}`. The first element is a list of the vertexes in the graph. The second list is a list of tuples representing the edges. For the purpose of the task, reflexive relations are not allowed and edges between two vertex only can exist once. Edges are unweighted and undirected (A to B is the same as B to A). I have implemented the following functions:

```
empty_graph - Return an empty graph
add_vertex - Add a new vertex if it is not already in the graph.
add_edge - Add a new edge if it is not already in the graph.
neighbors - Given a vertex, it returns a list of neighbors of that vertex.
add_vertexes - Given a list of vertexes, it adds all of them to the graph.
add_edges - Given a list of edges, it adds all of them to the graph.
build_graph - Given a list of vertexes and a list of edges, it adds them to the graph.
```

With this structure, we have an easy way to get the neighbors of a vertex, which we will need for the coloring algorithm as well as an easy way to count the number of vertexes.

2. Algorithm

1. Convert the structure given as an argument to the graph implementation explained before.
2. Create a list for the colors (C), and append {1, 1} (which represents the color of the first vertex).
3. For each of the vertex (V) in the vertexes list:
   - 3.1. Take the list of colors of the neighbors (N) of V.
     - 3.1.1. If N is empty, append {1, V} to C and evaluate the next vertex.
     - 3.1.2. Otherwise, check if N contains all the numbers in the sequence between 1 and max(N).
       - 3.1.2.1. If there is one value missing (X), append {X, V} to C and evaluate the next vertex.
       - 3.1.2.2. Otherwise, append {max(N)+1,V} to C and evaluate the next vertex.
4. If the number of colors used in the algorithm is lower or equal to the expected value return C, otherwise return false.

3. Testing

I have created the following files with different tests (this files can be found in my submission):

```
test_cases_graph_impl - Test basic implementation of the graph. Add vertexes and edges.
Check that same vertexes and edges are only added once.

test_cases_graph_neighbors - Test that neighbors function works good.

test_cases_graph_parse - Test the parse of the graphs.
From the structure given to the structure developed and explained before.

test_cases_kcolor - Test the coloring algorithm.
```

The test function reads all the lines from this files and iterate for each case. The test function is shown below:

```
main_test() ->
    {ok, GraphCases} =
        file:consult("test_cases_graph_impl.txt"),
    {ok, ParseCases} =
        file:consult("test_cases_graph_parse.txt"),
    {ok, NeighborsCases} =
        file:consult("test_cases_graph_neighbors.txt"),
    {ok, KcolorCases} =
        file:consult("test_cases_kcolor.txt"),
    lists:map(fun ({V, E, OK1}) -> OK1 = build_graph(V, E)
              end,
              GraphCases),
    lists:map(fun ({Input, OK2}) -> OK2 = parse_graph(Input)
              end,
              ParseCases),
    lists:map(fun ({G, V, OK3}) -> OK3 = neighbors(G, V)
              end,
              NeighborsCases),
    lists:map(fun ({G, N, OK4}) -> OK4 = kcolor(G, N) end,
              KcolorCases).
```