AFP - Assignment 2 - KColor

Maximo Garcia Martinez

1. Graph implementation

My implementation of the graph is a tuple of two list as follows (same as Assignment 1): `[[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 - 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 and a graph, it returns a list of neighbors of that vertex.
```

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. Testing graph implementation

The test function is using QuickCheck for testing different properties:

- Normal creation of a graph.
- Not add same vertex twice.
- Create a graph with edges
- Not add same edge twice.
- Get the neighbors of a vertex.

3. Algorithm

1. Convert the structure given as an argument to the graph implementation explained before.
2. Calculate all the possibilities for coloring the graph using n as maximum number of colors and for each possibility:

   2.1. Check if the coloring is correct; for every edge in the graph, check that the colors of the vertices in the edge vertices are different. If it is correct, return the colors distribution, otherwise, check next possibility.
3. If we don't find a way of coloring the graph with n, then return Nothing.

4. Monads and laziness

4.1. Use of Monads

The implementation of my code uses Maybe for returning the list of colors.
Also, the function asum() is being used, which is imported from Data.Foldable which uses monads as well.

asum() takes the head of a list and applies a function, if that function returns Nothing, then it checks the next element of the list, otherwise it returns the value returned by the function. If there are no elements left in the list, then asum() will return Nothing.

4.2. Use of laziness

The combinations of the different coloring possibilities are not evaluated at once. First, the algorithm takes the first possibility, and check if it is correct which in that case it will return that possibility and the algorithm will stop. If the check fails, then the next possibilities will be evaluated and checked until one option has been found or there are no more options which it will return Nothing.