Vallero Marzio s249920

Report for the Algorithms and Programming Exam of 31st January 2019

I wanted to solve the vertex coloring problem with a general approach that didn't involve a brute force approach, but rather to find an algorithm that builds up the solution, thus allowing for optimized solution times, that didn't completely go out of control as the problem grew in complexity as a brute force approach would do.

I have used the *graph.h* library provided by Professor Quer during his course, with some slight modifications, in order to allow me to use strings as vertex names, instead of integers. In particular, within *graph.h*, struct vertex_s has been modified in order to accomodate for an additional element: char* name. That pointer allows me to save the name of the vertex through *strdup*, and is used with the same purposes for which, in the original library, int id was used. I kept vertex->id in the struct to still be able to use it.

Also, in order to use the library, I have to point out that the source files that the program reads, have to be structured in such a way that the first line of the file contains two numbers. The first is the number of vertexes of the graph, the second is a "0", that in the graph.h library indicates an undirected graph. If this first line is missing or the parameters are not correct, the program will not work.

Of course, also the functions *graph_load*, *graph_find* and *new_node* had to be slightly modified in order to accomodate for the addition of the new parameter char* name, though this doesn't interfere with how the algorithm that I wrote for the solution works. I kept a copy of graph_find that works with id, called *graph_find_id*, in order to use it in the external function qeue_next.

During vertex processing, I have used vertex_t * vertex->**color** to indicate whether a vertex had already been processed (vertex->color == BLACK) or not (vertex->color == WHITE), as we did with the professor during the semester. I have instead used the parameter vertex_t * vertex->**dist** to indicate the actual color assigned to the vertex to solve the vertex coloring problem. *I hope this does not cause confusion during the correction*. Also, the colors for the vertex coloring problem are actually just numerical values.

The program is composed of a main function and three external functions:

- The *main* function contains checks for input arguments, calls to the external functions that read the file, initialize and find the solution for the graph, the printing on stdout of the solution and at last, calls to functions that free the memory previously allocated for the graph.
- *vertex_color* is a BFS function, that starting from the first vertex of the first edge listed in the source file, performs a Breadth First Search. It recieves as parameters the pointer to a graph (to be processed), a source vertex to start from and an integer, that counts the number of different colors minus one used to solve the graph. At first, the vertex is checked, giving it vertex->**dist** == 0. Then the function checks all vertexes adjacent to the vertex we are processing: if a new vertex is discovered (i.e. vertex->**color** == WHITE), it is added to a FIFO queue through the function *add_queue*; otherwise, the function checks whether vertex->**dist** of the node we are processing is different from the one that's adjacent to it. If they have a different vertex->**dist**, the function proceeds to check the next adjacent vertex. If, on the other hand, the two vertexes have the same vertex->**dist**, the **dist** of the vertex we are processing is increased by 1, then all its adjacent vertexes are checked again from the start in order to avoid fringe cases of failure (i. e. by increasing by 1 the **dist** of the vertex, it becomes equal to the one of a vertex that has been checked before). Once all adjacent vertexes have been checked, the vertex is considered fully processed and its vertex->**color** is changed to BLACK. Then the function looks for the first vertex present in the queue, if any, in order to process it. If there is no vertex to process in the queue, this means that all vertexes originally connected to the first node have been processed. In the case of a disconnected graph, in order to process also the "external" nodes, a further check is done in *main*, in order to be sure that all vertexes have been processed (thus having vertex->color == BLACK). If not, a new call to the function *vertex_color* is done, starting from the not yet processed vertex just found.
- *add_queue* is a function that recieves as parameters: a vertex to add to the queue and the head vertex of the queue. If the queue is empty, the function assigns a new node to the head pointer, that stores in it the vertex->id of the node that has been put in the queue. On the other hand, if the queue is not empty, the newly discovered vertex is added at the end of the queue with a tail insertion. In order to create the queue, I have repurposed the pointers used to link the list of vertexes as they are initialized for a graph in *graph.h*. To create the new vertex for the queue I have made a small modification to the *graph.h* library function new_node, in order to initialize new nodes with ptr->name="nan".
  This allows me to distinguish vertexes which have already been assigned a name from the ones who did not.

- *queue_next* is a function that recieves the head pointer to the queue and the graph we are processing, in order to extract the pointer of the head element of the queue from the graph itself, through the enqueued head vertex's id. This allows me to not have to copy every parameter when I enqueue a vertex from the graph, but rather only the vertex's id, in the new vertex created for the list. I did this in order to be sure that manipulation of the list did not also create modifications in the actual graph. Once the pointer for the head vertex is extracted, the head is moved to the element next to the head, and the pointer of the previous head is freed.

Here is a list of the modifications I have made with respect to the original written exam:

- In *main*, rearranged the initializations of some variables like int sol and graph_t *graph and rearranged in signle consecutive lines the commands that I had previously written on the same line in order to improve readability.
- In *main*, fixed the condition for the *while* at line 36 in order to work. In the written exam I wrote (vertex->next!=NULL), by meaning a check that allowed me to go through the list as long as there were elements in it. I forgot to use the temporary pointer that I had previously initialized to do it.
- In *vertex_color*, added one small vertex->**color** change from WHITE (undiscovered) to GREY (discovered, but yet to be processed) at line 54 at the time of discovering a new vertex. This doesn't change how the algorithm works, since it actually works even without this line, but improves performance, by avoiding possible duplications of one same vertex in the queue caused by line 62, which has been introduced in order to avoid fringe cases of failure in the algorithm (i. e. by increasing by 1 the color of the vertex, it becomes equal to the one of a vertex that has been checked before), but mainly to improve performance. In order to accomodate for the "edge check reset" performed on line 62, I had to add another *else* check at at line 63 in order to avoid missing a check on the first adjacent vertex in the list of adjacent vertexes of the processed vertex.
- In *add_queue* and *queue_next*, the argument *vertex_t *head* has been modified in order to be able to manipulate it as I intended to in the two external functions, by recieving it as a double pointer. According to this change, also "*" characters have been added to allow the correct manipulation of the list in the two functions.
- In *add_queue* the name of the argument *new* has been changed to *new_v*, in order to avoid conflicts with C syntax. I have introduced the use of the graph.h library function *new_node* in order to create an actual list, instead of just relinking the vertex pointers of the original graph, since that is what I actually intended to to in the program. The general enqueue algorithm has not been modified though.

Here is a graphical representation of a complex graph created by mysef to check the correctness of the algorithm, which is saved inside of *source1.txt*. The numbers on the rightmost corner of each vertex represent their color in the solution for the vertex coloring porblem. At the side, the command prompt window I got when running the program on the graph shown. The original example files of the Exam are saved in *source2.txt* and *source3.txt*.