Project 3: Graphing Algorithm "Apollo and Diana"
My Nguyen, U71337744

Before you write a program to solve this problem, you will first write a report describing (in English and pseudocode) how you will solve this problem. This report should answer two basic questions:

1. **What type of graph would you use to model the problem input (detailed in the Section 3.1), and how would you construct this graph? (I.e., what do the vertices, edges, etc., correspond to?) Be specific here; we discussed a number of different types of graphs in class.**

   The model of this graph can be seen below in the table. Firstly, to allow for better computation, the maximum number of vertices were set to 500 by 500 (N x M), which is defined linearly. Then, to store know what information to store from the values from input.txt file, specifiers were defined for colors (NOCOLOR, R, B) and directions (NONE, N, S, E, W, NE, NW, SE, SW) with the case of NOCOLOR and NONE for the bullseye. From there, a struct was created to hold the color, direction, number of neighbors, and an integer array of neighbors.

   The vertices within this graph is labeled as index 0 for the start position to index 63, which is the exit. The graph contains the number of vertices linearly and when the vertices are visited, it is also visited linearly. When the input.txt file is opened, each vertex is read one by one and put inside the graph and labeled according to their color and direction. Initially, none of the vertices have been visited.

   Thus, the starting position has the vertex 0, with N = 0 which is row 0 and M = 0 which is row 1. The end position or solution (bullseye) is at vertex 63, which is (N x M – 1) which is row times column – 1 for the offset with indexing starting at 0.

	Col 0	**Col 1**	**Col 2**	**Col 3**	**Col 4**	**Col 5**	**Col 6**	**Col 7**
Row 0	0	1	2	3	4	5	6	7
Row 1	8	9	10	11	12	13	14	15
Row 2	16	17	18	19	20	21	22	23
Row 3	24	25	26	27	28	29	30	31
Row 4	32	33	34	35	36	37	38	39
Row 5	40	41	42	43	44	45	46	47
Row 6	48	49	50	51	52	53	54	55
Row 7	56	57	58	59	60	61	62	63

   The model is using a directed graph. The vertices are the vertexes (arrows) and the edges connect two vertexes (arrows) of opposite colors together. The graph is built by connecting two arrows if they are different colors to create edges.

2. **What algorithm will you use to solve the problem? You should describe your algorithm in pseudocode. Be sure to describe not just the general algorithm you will use, but how you will identify the sequence of moves Apollo must take in order to reach the goal. Your algorithm must be correct, and it must have the minimum possible complexity.**

   The algorithm used is a depth-first search (DFS), traversing the graph linearly. After constructing the struct called Vertex to hold all the attributes such as color, direction, etc. A graph was created to hold the number of vertices and a Boolean called Visited was created to hold the number of vertices, both linearly. A stack called Solution was created to hold the steps that were taken to get to the exit. A Boolean flag was created to keep track of whether a solution was found or not.

   The graph is based on two for loops (the outer loop for rows and inner loop for columns). If the first position is not the bullseye, then enter the graph and search for the bullseye. Set the current vertex as the first position based on the formula i*M + j. From there, count the number of neighbors next to the current vertex within the graph. Depending on the current vertex, the potential neighbors can be in all eight directions. However, the neighbors must be the opposite color of the current vertex's color. If the colors are different, then add the potential neighbor to the graph and increment the number of neighbors. The directions are described as below:

```
if arrow points north,
        Decrease row
else if arrow points northeast
        Decrease row
        Increase column
else if arrow points east
        Increase column
else if arrow points southeast
        Increase row
        Increase column
else if arrow points south
        Increase row
else if arrow points southwest
        Increase row
        Decrease column
else if arrow points west
        Decrease column
else if arrow points northwest
        Decrease row
        Decrease column
```

   This DFS algorithm marks the first vertex as visited. Then, it checks to see if the vertex is equal to the solution by subtracting the offset of 1 from the number of vertices in the graph (N x M – 1). If the solution is found, then it marks the solution as found and push the vertex onto the bottom of the stack. If the solution is not found, then it traverses through the graph to find the solution and push the vertexes leading to finding the solution onto the stack. DFS will either run until it has found the solution and return or continue running until all possible paths have been found to the solution. The DFS algorithm can be found below:

```
mark the visited vertex as true;
if the vertex = position of the exit (solution){
    set solutionFound = true;
    push the vertex onto bottom of stack;
} else
    for (int i = 0; i < Graph[vertex].NumberOfNeighbors; i++){
        if the neighbors have not been visited
            use DFS to find the neighbors to the solution;
        if the solution has been found
            push the solution onto the stack;
            return;
    }
}
```

To print the solution, the algorithm pop and print the vertexes on the stack. If the solution has already been found, it will be at the bottom of the stack and be printed last. To calculate the steps between the current and next position, the following pseudocode can be found below:

```
int steps;
if ((current position / column) != (next position / column))
    steps = (current position / column) – (next position / column);
else
    steps = current – next;
if (steps < 0)
    steps = -steps;
```