Ramaneek Gill
1000005754

CSC264 Problem Set 7

ALGORITHM:

```
DFS(G=(V,E)):
        array_of_edges_in_simple_cycle = []
        for i <- 1...n:
        colour[i] <- white

   for i <- 1...n:  //for loop makes sure to go through every node even if it is
                                    //unreachable from an arbitrary point on the graph
        if colour[i] == white:
                array_of_edges_in_simple_cycle = []
                if DFS-VISIT(G,i, i) != []:  //if a cycle was found no need to keep searching
                        return array_of_edges_in_simple_cycle

        return array_of_edges_in_simple_cycle

DFS-VISIT(G=(V,E),u, START):
        colour[i] <- gray
        for k <- 1...n:
                if G.A[i,k] = 1: //if G contains an edge {i,k}
                        if colour[k] == white:
                                array_of_edges_in_simple_cycle.append((i,k))  //add the edge to the list
                        DFS-VISIT(G,v, START)

                   if k == START:  //have reached the start node and made sure it was a simple cycle
                                array_of_edges_in_simple_cycle.append((i,k))  //add the LAST edge to
the list
                        return array_of_edges_in_simple_cycle

        colour[u] <- black
```

NOTE:
I added an extra parameter to the function DFS-VISIT, it is the START node from where the initial DFS search begins.

CORRECTNESS:
The algorithm will traverse through all paths that can be reached from every single node in the matrix. However it will not traverse through an edge if that next node has already been discovered. This prevents a future case of discovering a cycle that is not simple. The algorithm will append to the list array_of_edges_in_simple_cycle the edges it is traversing through, if after an arbitrary amount of traversals it has reached the same node it started the traversal with it will return the list. Because of the previous condition we can be sure that the cycle is a simple one.

The list is reset to an empty array each time a new START is declared so the list will always have edges that are not necessary for creating a simple cycle (ie A->B->C->D->B, A is necessary).

RUNTIME:
This program operates on a matrix of input size n^2. A normal DFS operates at a running time of \Theta(n^2). This algorithm essentially operates the exact same way as a normal DFS since it will always do the exact
same traversals until a simple cycle is detected or until there are no more paths to discover. Therefore the worst case running time of this algorithm is \Theta(n^2).

Part 2:
Input matrix, size is 5x5
  1 2 3 4 5
1 0 1 0 0 1
2 1 0 1 1 1
3 0 1 0 1 0
4 0 1 1 0 1
5 1 1 0 1 0

Cyclic will find the cycle:
1->2->5->1
My algorithm will discover:
1->2->1 since my DFS search relies on the fact that the graph is directed, for undirected graphs my algorithm
will always find that you can move "backwards" and create a simple loop with only 2 nodes for every connection.
This is also related to the fact that each row is a transpose of it's coloumn and vice versa.