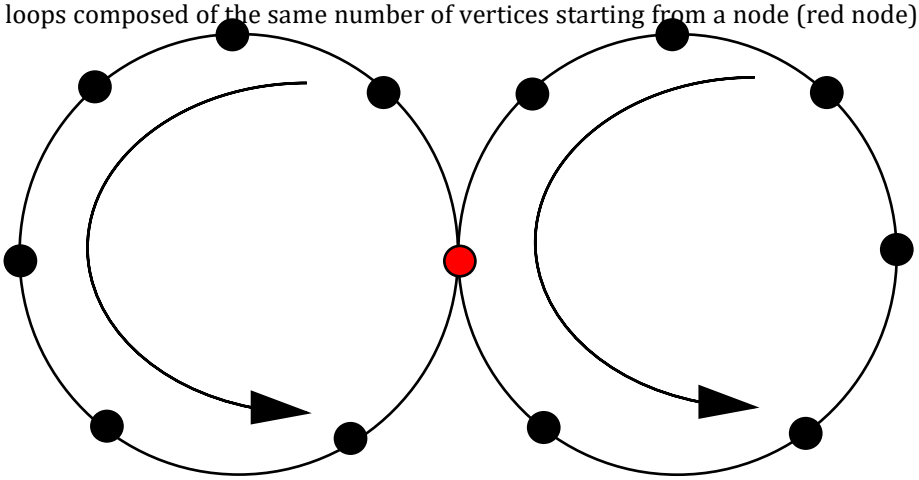
John Ezekiel Juliano 30000523

Questions

1. Describe the algorithm based on the graph traversal to find if there is a loop. Which traversal technique is more suitable? What is the worst-case complexity of this algorithm?
   A loop or a cycle is when a path starts and ends at the same node. A worst-case scenario can be two circular loops composed of the same number of vertices starting from a node (red node) and ending at the same node



   (shown above).
   Using BFS to detect the cycle in this situation would take twice as much time than DFS. Because if we implement BFS using a queue, a visited array and keeping track of the ancestor node(s), this will cause the recursion queue to move back and forth between the two loops and it will take **O(V+E)** as the time complexity of this algorithm. Even though the time complexity of a DFS algorithm would result the same way as BFS (ie: **O(V+E)**) in reality, it will only take half as much. This is because DFS will focus on one path until the search is done, no more edges are present or, there's a loop in the path.
   Using a DFS traversal we can keep track of any back edges. A back edge is an edge that loops from a node to itself or one of its ancestors. Additionally, we need to implement a stack to store the nodes in recursion and a visited array to store visited nodes in the graph. If at any point in time that the adjacent vertex/vertices from the current vertex has/have already been visited, we can check the stack if these vertex/vertices is there. If it is, then we have a loop/cycle.
   The **time complexity of the DFS implementation** is **O(V+E)** where V is the number of vertices and E is the number of edges. This is similar to the regular DFS traversal because we are simply traversing through the graph and having a couple arrays to track whether is node has been visited or is in the recursion stack.

2. For a given query file, which traversal technique is more efficient in determining if the path exists between the given nodes? Use the provided input file and query file to help answer this question.
   Based on the output files generated by the program, it can be seen that **DFS traversal performs much better than BFS**. The paths generated by DFS are much shorter than BFS, this correlates to the time it took to find the sink from the source. Consequently, this also shows how much space the algorithm used to perform the given task. BFS uses more space than DFS because it need to visit more vertices to get to the destination. This is because it has to check all neighbouring vertices before moving on to the next "level."
   DFS is much more efficient because of its recursive nature. It requires less memory space both, to call the method and to store its recursion stack. It also performs much faster since it visits less vertices to get to the destination.