Jia Feng Yu
20908183
CS341 A2
Question 4

In order to find directed, odd cycles in a directed graph, we can simply reuse the depth first search algorithm and augment it by adding a parent tracking feature similarly to the one used in the notes for BFS.
Every time we visit a neighbor v of a current node u that is not yet visited, we would set visited[v] = true, and bind parent[v] = u. If we find a neighbor v such that visited[v] = true, then by the properties of DFS we would have found a cycle because we would explore down one branch first, and if it is visited then it means that it is a node already on the branch, thus creating a cycle.
Since DFS has runtime $O(n+m)$, ie, $O(|V|+|E|)$ from the course slides and we are only adding the $O(1)$ operation of updating the parent array, the runtime of this algorithm would still be $O(|V|+|E|)$.
To show correctness, assume we have found a scenario as described above, where we claim to have found a cycle. We are currently at a node u and we know that a neighbor v is already visited. By nature of DFS, there must exist a path between u and v through u's ancestors that we have made when we explored the graph. However, because v is a neighbor of u, this means that there is a directed edge linking u to v. Hence, that directed edge combined with the path through u's ancestors create a cycle.
We only need to check if this forms an odd cycle before outputting our result: From u, backtrace its parent through the parent vertex and every time we move to an "older" parent, increment a counter. We do this until we have reached the neighbor v in question, and we simply check the parity of the counter; output yes if the counter is odd and no else wise. Since we have to recurse over the parent vertices, this would take a subset of the total set of vertices, hence this operation is at most $O(|V|)$, so the overall time complexity remains $O(|V|+|E|)$.
Bonus: In order to trace out the cycle, we can take the current node u we have stopped and print its value, look up its parent in the parent array, print out its value, look up its parent and print its value, etc. until we have reached the point where we have found v. We can verify that we have found v by simply doing an equality check with the neighbor of u, which we know the value.