ALGORITHM bfs\_detect\_cycle(M, V)

//Breadth-First Search traversal of matrix

//Input: List vertices = [vertex], matrix Matrix = {vertices, edges}

//Output: List Cycles filled with lists of vertices Cycle in order of each cycle

//For asymptotic analysis comments, will refer to LENGTH(vertices) as V, and LENGTH(edges) as E

num\_vertices ← LENGTH(vertices)

visited ← [FALSE \* num\_vertices]

queue ← [EMPTY QUEUE]

cycles ← [EMPTY LIST]

FOR start IN range(num\_vertices) DO // O(V)

IF visited[start] == FALSE DO

path ← [EMPTY LIST]

APPEND (start, [start]) to queue

WHILE queue is not empty DO

current, path == POP[queue]

visited[current] = TRUE

FOR neighbor IN range(num\_vertices) DO // O(V)

IF matrix[current][neighbor] exists DO

IF neighbor IN path DO //O(E)

cycle ← path[index of neighbor in path to end] + [neighbor] //O(LENGTH(cycle))

APPEND cycle TO cycles //since LENGTH(cycle) == V in

ELSE IF visited[neighbor] == FALSE DO //worst case, O(V)

APPEND (neighbor, path + [neighbor]) TO queue

IF LENGTH(cycles) > 0 DO

FOR cycle in cycles DO

PRINT cycle

TIME COMPLEXITY:

Traversal of Matrix is O(V2) (nested loop)

(Worst case) Cycle detection is O(E ∙ V)

Total Complexity is O(V2 + E ∙ V)