Shreya Patel

Assignment 4

1.
   a. The time complexity of an undirected graph adjacency matrix representation is O(|V|^2). Since the adjacency matrix is a 2D matrix of size V*V, it would take V^2 amount of time for the matrix to be executed. The number of edges don't affect this because the edges connected by each of the number is represented by a 0, if there is no edge between the numbers, and a 1, if there is an edge, in the matrix.
   b. The time complexity of an undirected graph adjacency list representation is O(|V| + |E|). The adjacency list shows each individual vertex and all the edges that they are connected to. So the time complexity would be the sum of the number of edges and the number of vertices of the graph. This is why the time complexity would be the sum of those two things.
   c. The time complexity of a directed graph adjacency matrix representation is O(|V|^2). The adjacency matrix of a directed graph is also a 2D matrix of size V*V and the number of edges are represented by a 1 or a 0, if there exists an edge or not, respectively. Even if directed graphs have a direction to their edges the time complexity remains the same as the undirected graph.
   d. Time complexity of a directed graph adjacency list representation is O(|V| + |E|). Adjacency list has shows each vertex connected to other vertices depending on the direction of the edge. Even though directed graphs are different from undirected graphs, the time complexity is the same for both.
2.
   a. Depth-First Search(Undirected Graph)

Vertex	0	1	2	3	4	5
Marked []	T	T	T	T	T	T
edgeTo []	-	4	0	2	2	1

   b. Breadth-First Search(Undirected Graph)

Vertex	0	1	2	3	4	5
edgeTo []	-	4	0	2	0	0
distTo []	0	2	1	2	1	1

3.
   a. Depth-First Search(Directed Graphs)

Vertex	0	1	2	3	4	5
Marked []	T	F	T	T	T	T
edgeTo []	-	-	0	2	0	4

   b. Breadth-First Search(Directed Graphs)

Vertex	0	1	2	3	4	5
edgeTo []	-	-	0	2	0	0
distTo []	0	-	1	2	1	1

4. Pseudocode for DFS and BFS

a. Iterative DFS:
    1. Start with the source vertex
    2. Check if the vertex has an edge
        a. If it does follow the edge to the next vertex
        b. If it doesn't then the search has ended.
    3. Repeat steps 1-2
        a. If the next vertex has an edge connected to a vertex already visited
            i. Keep going back to the previous vertex until the previous has an edge connected to a vertex that hasn't been visited.
        b. If repeating step i. leads you to the source vertex then the search has ended.

b. Recursive BFS
    1. Start with the source vertex
    2. Check if the vertex has an edge
        a. If it does, then follow the edge to the next vertex
        b. If it doesn't then the search
    3. Go back to the previous vertex to check it it is connected to a different edge.
        a. If it does, then repeat step 2a.
        b. If it doesn't go to the first vertex visited from the source vertex and repeat steps 2-3.
    4. When all the vertices have been visited the search has ended

The big Oh notation for DFS is O(|V| + |E|) because its iterative and it searches the entire graph which encompasses checking all the vertices and the edges. The big Oh notation for BFS is O(|V| + |E|) in the worst case because it is recursive and in the worst case every single vertex and edge will be explored. In the best case all the vertices would be connected to the source vertex so it would only check the source vertex and the edges: O(1 + |E|).

5. Graph-Processing Challenge 2
    1. Start with the source vertex
    2. Assign the source vertex with the color red
    3. Make all the neighboring vertices the color white
    4. Make all the neighboring vertices of that vertex the color red and connect them
        a. If the neighboring vertex and this vertex are both the same color then the bipartite has ended.
        b. If they are different colors repeat step 4 but make sure that the vertex and it neighbors are alternative colors
    5. When all the vertices in the graph are connected with the previous conditions the graph is bipartite and the division has ended.

Vertex	0	1	2	3	4	5	6
edgeTo []	-	0	0	1	5	0	0
distTo []	0	1	1	2	2	1	1
Color	red	white	white	red	red	white	white

6. Graph-Processing Challenge 3
    1. Start with the source vertex
    2. If there is an edge connecting the source vertex to another vertex, then go to that vertex.
    3. Go from that vertex to the next vertex from the edge.
    4. Repeat step 3 until
        a. If the edge connects back to the source vertex there is a cycle present in the graph and execution ends.
        b. If the edge connects back to a vertex already visited
            i. Go back to the previous vertex to check if it is having another edge. If there is no other edge there is no cycle in the graph and execution ends.
            ii. If there is another edge from that vertex follow that edge and execute step 4.
        c. If there is no edge connecting the vertex, then there there is no cycle present in the graph and execution ends.

Vertex	0	1	2	3	4	5	6
Marked []	T	F	F	F	T	T	T
edgeTo []	-	-	-	-	5	0	4

7. Cycle (Vertex v)
    1. v is the source vertex
    2. Push(v)
    3. For the unvisited vertex
        a. For each neighbor, x of v
            i. If x is unvisited
                1. Set the last vertex as v
                2. Push(u)
                3. Check the cycle with the vertex x
            ii. Else if the last vertex is x
                1. Check the next neighbor
            iii. Else
                1. Set cycle to true
                2. Output “Set found!”
                3. New line output would be x + “ “
                4. Break
        b. If cycle is true
            i. Output v + “ “
        c. New queue
        d. Move to next unvisited vertex.

The linear time algorithm is $O(|V| + |E|)$ because the recursive function visits each vertex once and then each edge traverses at least once which means that its actually $V + t$. Where $E <= t <= 2E$. So in the big Oh notation is $O(|V| + |E|)$.