Problem Set: 3.1

Question: 1

Answer:

Algorithm

(1) cycle_detection ($G = (V, E)$)

for each $u \in V$

u.visited = False

u.parent = None

BFS(s) ⇒ s is starting node

1. Initialize an empty Queue Q. Set s.visited = True.
2. Enqueue s into Q.
3. While Q is not empty:
4. Dequeue node u from Q.
5. For each neighbor v of u:
6. if v is not visited:
7. set v.visited = True
8. set v.parent = u
9. Enqueue v into Q.
10. Else if v ≠ u.parent (if v is visited and not the parent of u):
11. A cycle is detected.

Proof of Correctness:-

- If BFS explores all edges and there is no edge that connects two already visited nodes, the the graph is a Tree, and no cycle exists.

- If in BFS, an edge $e = (u, v)$ is found where both u and v are already visited but not directly parent-child in the BFS tree, It means that the graph contains a cycle. This edge 'e' is refereed as 'back-edge".

Time Complexity:-

- Each vertex is visited exactly once when it is dequeued from the queue. When a vertex is dequeued, all of its adjacent vertices are examined.

- For each vertex u, the algorithm processes the adjacency list. The adjacency list of each vertex contains all edges incident to that vertex.

- over the course of BFS, each edge is examined exactly twice, the total time to process all edges is proportional to number of edges (m).

So, total time complexity is $O(m+n)$

Question: 2

Answer:-

Claim:- Let 'G' be graph of 'n' nodes, where 'n' is an even number. if every node of 'G' has degree at least $n/2$, then G is connected.

Proof:-

To prove the claim is true, we need to show that if every node in the graph has a degree of atleast $n/2$, then the graph must be connected. Let's proceed by contradiction.

Step: 1 Assume Graph 'G' is not connected

This means that G can be split into two or more disjoint components, say G_1 and G_2, where:

- The number of nodes in G_1 is $|V(G_1)| = K$
- The number of nodes in G_2 is $|V(G_2)| = n-K$
- Since G is not connected, there are no edges between the nodes in G_1 and G_2.

Step 2:- Apply the degree condition to G_1 and G_2.

- The degree of each node in G_1 counts the number of nodes it is connected to within G_1;

- Maximum possible number of nodes in G_1 is k, so the degree of any node in G_1 is atmost $k-1$.
- But from assumption, every node has degree atleast $n/2$

Step 3: Contradiction for G_1 and G_2

for node in G_1, it's degree is atmost $k-1$,

$$\therefore k-1 \geq n/2.$$

$$\Rightarrow k \geq \frac{n}{2}+1$$

for G_2 which has $n-k$ nodes, degree of node $= n-k-1$

$$n-k-1 \geq n/2$$

$$n-k \geq \frac{n}{2}+1$$

Step 4:-

Now $k \geq \frac{n}{2}+1$ and $n-k \geq \frac{n}{2}+1$

$$k+(n-k) \geq \frac{n}{2}+1+\frac{n}{2}+1$$

$$n \geq n+2 \text{ (impossible)}$$

Since contradiction is wrong, The original assumption is false. So G must be Connected.

Question:3

Answer : The primary effect of shifting from an adjacency list to an adjacency matrix representation is in how the edges of graph are accessed.

(i) steps (1) and (2) still take $O(n)$ time, as these steps initialize data structures for all 'n' nodes.

(ii) steps (3), (4) and (5) continue to take $O(1)$ time, as they involve initializing and adding the starting node to the queue.

(iii) The while loop in step 6 still runs $O(n)$ times because each node is added to the queue only once. ∴ step (7) also takes $O(n)$ time.

(iv) The for loop in step (8) now takes more time since we must check all nodes to find those adjacent to 'u' This requires $O(n)$ time for each node 'u' because we need to scan the entire row of adjacency matrix. ∴ this step runs $O(n) \times O(n)$ resulting $O(n^2)$ time overall.

(v) steps 9, 10, 11 still take $O(1)$ time, ∴ $S.T O(n^2)$ overall.

So, the complexity increases from $O(m+n)$ to $O(n^2)$

↓ for Adjacency list

↓ Adjacency Matrix.