Module 2 Graphs Written Assignment

Tyler Kim
tkj9ep

October 05, 2022

Responses

1. Let the adjacency matrix, Adj, for G = (V, E), represent the edges of G such that Adj[u][v] denotes an edge from node u to node v. Let an array, A, of size 1 x V denote whether a particular node could possibly be a sink node and be initialized to all 1's. In other words, if the value of A[x] is 1, then node x could possibly be a sink node, otherwise node x is not. Let a pointer, ptr, point to the first element of A.

The algorithm would start at Adj[u][v] where u, v = 0. If the algorithm finds the value of 0 at Adj[u][v] and u != v, then A[v] will be set to 0 and will move one index to the right (v = v + 1). Otherwise, if the algorithm finds Adj[u][v] = 1, then A[u] is set to 0 and the algorithm will look one index down (u = u + 1). If ptr is not pointing to a value of 1 in A, ptr will traverse through A until it reaches an index with the value of 1. Once the algorithm reaches the end of Adj, it will check if the last node is the sink node by seeing if its row is all 0's and its column is all 1's except where the edge is to itself.

The runtime of the algorithm is O(V). Forming A initially is linear. Let s = u + v. Everytime the algorithm moves to the next index in Adj, s will increase by 1 because only u or v will increment after each evaluation. This means $s \leq 2V$. Therefore, the runtime of this algorithm must be O(V).

2.

```
boolean hasFoundT = False
visited = {}

DFS(G, s, t) // s is the start node and t is the target node
    for each vertex u in G.V
        u.color = WHITE
    if hasFoundT == True
        return visited
    else
        return {}

DFS-VISIT(G, s, t)
    u.color = GRAY
    visited.prepend(s)
    if s == t // base cases
        hasFoundT == True
    else  // recursive cases
        for each v in G.Adj[s]
            if v.color == WHITE
                DFS-VISIT(G, v, t)
        s.color = BLACK
    return
```

3. Let G = (n, E) be an undirected graph with n nodes and that n is an even number.

Claim: If every node of G has a degree of at least $\frac{n}{2}$, then G must be connected.

Proof. Prove by contradiction by assuming that G is not connected when the degree of every node $\geq \frac{n}{2}$. □

Base: When $n = 2$, that is, only node u and node v are in the graph, then each node must have a degree of at least 1 by the claim. Since an edge must connect two nodes, then node u and node v must be connected to each other.

Inductive: Let all the nodes of graph G have a degree of at least $\frac{n}{2}$ such that G is connected. Let node v be an arbritrary node that is added to G such that G is not a connected graph. This must mean that v cannot be reachable from all other nodes. Since v must have a degree of at least $\frac{n}{2}$ and cannot point to itself, v must point to another node, u, in the graph. This contradicts since all the nodes previously in the graph could reach each other node and each edge is bidirectional. Thus, if node v had an edge with u and u is reachable from other nodes by the definition of a connected directed graph, then v must also be reachable by all other nodes either through u or another path.

$\therefore$ If every node of the undirected graph G has a degree that is at least $\frac{n}{2}$, then G must be connected.

4.

1) Let G = (V, E) be an undirected and unweighted graph where each vertex V represents a state where each state represents the possible positions of the robots. Let each edge E in G represent one single instruction to get from one state to another state. An edge E exists between vertex/state u and vertex/state v if and only if the position of the two robots are not adjacent nodes. If i represents the node that first robot is on and j represents the node that the second robot is on, then the algorithm can use the adjacency matrix to see if node i and node j are adjacent. Let array A be an empty list that represents the optimal path.

Finally, the algorithm would run BFS on the graph starting at (s1, s2) and record the path taken into array A until it reaches vertex (d1, d2).

2) Making graph G would take $\Theta(E)$ because each edge will only be check once to see if a valid state could be found. The next operation is BFS which dominates the runtime in the entire algorithm. Therefore, the runtime of the entire algorithm would be $\Theta(V + E)$

3) The runtime would grow and the number of robots grow because more edges will need to be checked to account for the other robots. Eventually, there will be enough robots such that the runtime for making the graph G will dominate.