

PROBLEM 1

Question. Give an algorithm to detect whether a given undirected graph contains a cycle. If the graph contains a cycle, then your algorithm should output one (it need not output all cycles in the graph, just one of them). The running time of your algorithm should be $O(m+n)$ for a graph with n nodes and m edges.

Solution. Let us consider an undirected, connected graph (or an undirected graph with some connected components).

We can use Depth First Search to detect a cycle in an undirected graph. In order to do this, we need to try and convert our graph into a tree - more specifically, a DFS tree, as we are using DFS for our graph traversal.

If at any point during the creation of the DFS tree from our graph we run into a **back edge (an edge that connects a node to itself or one of its ancestors)**, we can conclude that a graph contains a cycle.

The algorithm to detect a cycle in an undirected graph can be defined as follows:

Algorithm 1 Detect a cycle in an undirected graph

Ensure: The graph G has been created using the given number of edges and vertices

Ensure: The graph G contains connected components

 Perform DFS on the graph G and output the DFS tree T

if All the edges of G are present in T **then**

 Graph does not contain a cycle

else

 Let (u, v) be an edge that is present in G but not in T

 Here, (u, v) is a back edge in T

 In T , traverse the path P from v to u

 The cycle C is equal to $P \cup (u, v)$

 Return the cycle C

end if

As we are using DFS to detect the cycle in our graph, we should have a **running time of $O(m + n)$** for a graph with m edges and n nodes. \square

PROBLEM 2

Question. Suppose that an n -node undirected graph $G = (V, E)$ contains two nodes s and t such that

the distance between s and t is strictly greater than $n/2$. Show that there must exist some node v , not equal to either s or t , such that deleting v from G destroys all s - t paths. (In other words, the graph obtained from G by deleting v contains no path from s to t .) Give an algorithm with running time $O(m + n)$ to find such a node v .

Solution. Let us traverse the graph G using Breadth First Search starting from node s . Let node s be present in layer L_0 , and node t be present in the layer L_d . According to the problem statement, we know that $d \geq n/2$.

Now, let us consider the layers that the nodes s and t are **not** present in. Assume that one of these layers L_1, L_2, \dots, L_{d-1} contains only a single node. The reasoning behind this assumption is that if each of these layers had even just 2 nodes, they would contain at least $2(n/2)$ nodes - which is n .

But we know that G only has n nodes and neither s nor t appear in the aforementioned layers. Therefore, there has to be some layer L_x containing only 1 node v .

Next, let us assume that the removal of this node v would destroy all paths from s to t . Let us consider the set of layers $X = L_0, L_1, \dots, L_{x-1}$, that is, all the layers before L_x . Node s is part of layer L_0 . Node t is not a part of the above set of layers.

Since we have considered all the layers before L_x , by the properties of BFS, any edge that emerges from X must be a part of L_x . **Therefore, each and every path from s to t must emerge from X and by consequence, must contain a node in L_x .**

However, the only node in L_x is v .

In conclusion, we can confidently say that there must exist some node v , not equal to either s or t , such that deleting v from G destroys all s - t paths. \square