Cycle Detection

Here is a video walkthrough of the solutions.

Given an undirected graph, provide an algorithm that returns true if a cycle exists in the graph, and false otherwise. Also, provide a Θ bound for the worst case runtime of your algorithm. You may use either an adjacency list or an adjacency matrix to represent your graph. (We are looking for an answer in plain English, not code).

We do a depth first search traversal through the graph. While we recurse, if we visit a node that we visited already, then we've found a cycle. Assuming integer labels, we can use something like a `visited` boolean array to keep track of the elements that we've seen, and while looking through a node's neighbors, if `visited` gives true, then that indicates a cycle.

However, since the graph is undirected, if an edge connects vertices `u` and `v`, then `u` is a neighbor of `v`, and v is a neighbor of `u`. As such, if we visit `v` after `u`, our algorithm will claim that there is a cycle since `u` is a visited neighbor of `v`. To address this case, when we visit the neighbors of `v`, we should ignore `u`. To implement this in code, one idea is using a `Map` to map each node to the node we took to get there, e.g. we would map `v` to `u` in the example described above.

In the worst case, we have to explore at most V edges before finding a cycle (number of edges doesn't matter). So, this runs in $\Theta(V)$.