In order to determine whether a given undirected graph is a tree, we must ensure that the two properties of a tree (as mentioned in the problem) are present: acyclicity and connectivity. To detect if there are any cycles in the graph, we can perform a breadth-first search (BFS) or depth-first search (DFS) to iterate over the entire graph. For every visited vertex, we must check all adjacent vertices for two conditions: whether the adjacent vertex has already been visited **and** if the adjacent vertex is **not** the parent of the vertex we are visiting. If both of these conditions are true, then there is a cycle present in the graph and, thus, is not a tree. The above search will also determine the connectivity of the graph, since any vertex that is not visited by a BFS or DFS is not connected to the graph. The pseudo code below performs this determination using a depth-first search. As long as it is implemented using an adjacency list, it will run in **$O(n + m)$** time where n is the number of vertices and m is the number of edges. Were it implemented using an adjacency matrix, it would run in **$O(n^2)$** time.

Input: *v* - vertex
Output: *true* if graph G is a tree, *false* otherwise

```
Algorithm: DetectCycleDFS
Mark v as visited
For each vertex w adjacent to v
   If w has not yet been visited
      If DetectCycleDFS(w) returns true
         Return true
      End
   End
   Else if w has been visited and is not the parent v
      Return true
   End
End
Return false
```

Input: *G* = (*V*, *E*) - undirected graph to check
Output: *true* if graph *G* is a tree, *false* otherwise

```
Algorithm: Tree?
Create new array of booleans of size V
Initialize each bool to false
For each unvisited vertex v in V
   If DetectCycleDFS(v) returns true
      Return true
   End
End
Return false
```