2-a. Algorithms for:

- Is the graph a tree?

  For an undirected graph such as the network topology, the algorithm returns true if there is no cycle and the graph is connected i.e. all vertices are reachable from the source (`nodes[0]`).

  `check_spanning_tree(nodes, G):`

  Mark all nodes in the topology as not visited, by initializing their value to `False` in the `visited` dictionary.
  Check if there is a cycle in the topology from the source and mark all nodes visited on the way to `True`.
  If there is a cycle in the topology from the source: return `False`.
  Iteratively check if a node is not visited from source.
  If there exists such a node: return `False`.
  return `True`.

  `isCycle(node, nodes, visited, G, parent):`

  Set the value of node to `True` in the `visited` dictionary.
  For all neighbouring nodes, `adj` of `node`:
  If `adj` is not marked `True` in `visited`:
  Recursively check if there is a cycle in the topology from `adj` and mark all nodes visited on the way to `True`.
  If there is a cycle: return `True`.
  Else if `adj` is not visited and not the `parent` of `node`, then there is a cycle: return `True`.
  No cycle found: return `False`.

- Diameter of graph:

  For an undirected, unweighted graph such as the network topology, the Floyd Warshall Algorithm solves the All Pairs Shortest Path problem in $O(n^3)$ where n is the number of nodes in the topology. The result of this algorithm is iteratively checked to find the maximum shortest path i.e. the diameter. The input representation is modified to the adjacency matrix to use as input to the Floyd Warshall Algorithm.

  `diameter(nodes, G):`

  Represent the topology as an adjacency matrix: `graph = adjMatrix(nodes, G)`.
  Initialize the `distance` matrix as the `graph` matrix.
  For all nodes, `interim` in the `graph`:
  Iteratively pick all nodes as the `source`:
  For each `source`, pick all vertices as the `destination`:
  If node `interim` is on the shortest path from `source` to `destination`, then update the value in `distance[source][destination]` as:

distance[source][destination] = minimum(distance[source][destination], distance[source][interim] + distance[interim][destination])

Iterate through all distance pairs to find the maximum value, where the maximum value is not infinity.

Return the maximum value.

adjMatrix(nodes, G):

For each node, i:

For each node, j:

If j is equal to i: initialize graph[i][j] to 0.

If j is a neighbour to i: initialize graph[i][j] to 1.

Else: initialize graph[i][j] to infinity.

Return graph.

2-b. Final spanning tree with --mdst parameter for PA3-test.py:
Command: sudo python PA3-test.py

```
[openflow.spanning_tree ] *** SPANNING TREE ***
[openflow.spanning_tree ]  1 : 5
[openflow.spanning_tree ]  2 : 5
[openflow.spanning_tree ]  3 : 5
[openflow.spanning_tree ]  4 : 5
[openflow.spanning_tree ]  5 : 1 2 3 4 6
[openflow.spanning_tree ]  6 : 5
[openflow.spanning_tree ] **********************
[openflow.spanning_tree ] Spanning tree updated
```

2-c. ???