3. Graph Algorithm

Given an undirected graph represented in adjacency list and the n nodes in the graph are labeled from 0 to n − 1, design and analyze an algorithm to determine whether the graph is a tree.

According to the tree definition, an undirected graph is tree if there is no cycle in the graph and the graph is connected. To determine if there is a cycle in the adjacency matrix, we have to traverse the graph using adjacency matrix using DFS.

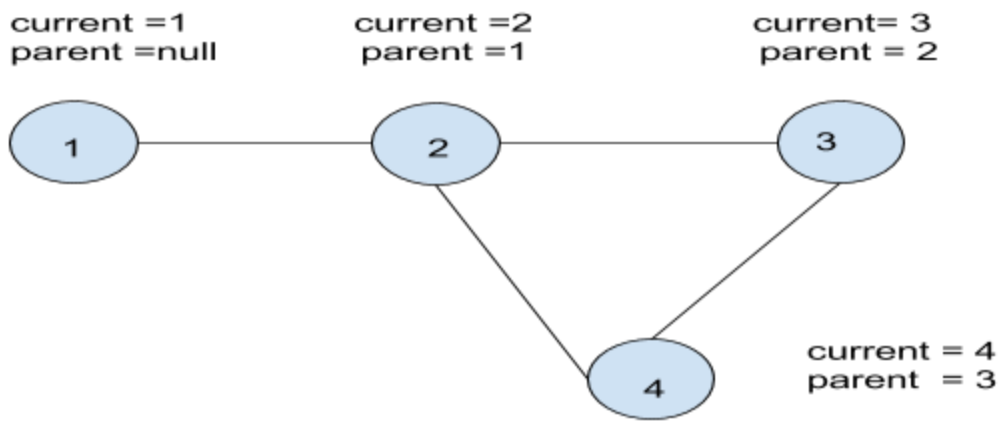
we first construct a new array which keeps track of the visited nodes in the adjacency matrix. The array is initialized to all zeros which means no nodes have been explored yet. The length of this array is equal to the length of the adjacency_list.

Track = [0,0,0,.....]

We first construct a helper function to determine if there is a cycle in the graph, this function takes input of a vertex, and the track array and previous vertex. Previous vertex means the that is visited before visiting the current vertex. It is used to check if a cycle is formed or not.

```
def determine_cycle( vertex_ID, track, previous) {
# set this node to explored status

Track[vertex_ID] = 1

# Now, we want to loop through all the neighbours of the current vertex.

for Vertex adjacent to the current_vertex{
  ## this can be done by tracing adjacency list.
## if the vertex is not visited, we run the algorithm recursively.

If visited[Vertex] == 0{
determine_cycle(Vertex, track, current_vertex)
}
## it is obvious that if the adjacent vertex of the current vertex is already visited, the parent of the current vertex must equal to that vertex. Otherwise loop is formed. (Below is the graph to explain this idea)

Else if  Vertex ! = previous:{
          return  true          ##  a cycle is formed, the graph does not contain trees.
}
Return False                ## If no cycle is detected, return False
}
```

Here is the graph to explain my idea:



Suppose we are at node 4. We start to check every adjacent node of node 4.
It first detects node 3 with the property that 4.parent == 3 is true. (no cycle)
Then it detects node 2 with the property that 4.parent == 2 is false (cycle is detected)

Here is my main function:
determine_tree(adjacency_list){

Initialize track array
Track = [0,0,0,.....]

check cycle using my helper function.(starts from vertex 0)
If determine_cycle(0, track, null) == true{
    return False
}

we have to then check the connectivity by checking all if all vertices are visited.
If track_array does not contain 0{
        Return True
}

Else {
Return False   ## the graph is not fully connected, return false.

}}

Time Complexity:
The time complexity is O (|V|+|E|). The reason is that for each vertex(V), we are going to check if all its neighbours (E) meet the requirement