Algorithm Assignment -6.1

Q1>

In order to identify all the critical bridges we can employ **brute force approach** where we consider each bridge between the islands one by one until all the bridges connecting the islands are covered. Check if removal of the edge from the graph disconnects the islands.

Steps involved:

1. Loop through each bridge that connect the island.
2. Remove the edge that is considered from the graph
3. Perform depth first search on the islands by considering any island as root node. During traversal add all the unvisited islands on to a array only if it is unvisited and mark them as visited
4. After traversal count the number of islands in the array. If count is not equal to the total number of islands Micronesia then it's a critical edge.
   Add this edge to an array which keeps track of all the critical bridge.
5. Once all the bridges connecting Micronesia are covered, we have a final array which has all critical bridges.

This approach takes B *(I + B) time, where B is the number of bridges, I is the number of islands in Micronesia. **Space complexity = l** , as this involves holding all the islands during DFS traversal

Efficient Approach: Let i1, i2, i3,…in be the islands in Micronesia. Let b1, b2, b3, b4…...bm be the bridges connecting the islands. This forms a graph representation with islands as nodes and bridges as edges of the Graph.

Finding all the bridges that are not part of the strongly connected component(SCC) of the graph will give us the critical edges, as their removal will disconnect the islands. We can find these bridges by using depth first search as depth first search creates sub forest on traversal for all strongly connected components.

In order to accomplish this, we assign each island a smallest index number which is unassigned and keep track of the node with the lowest index which is reachable from the given node. This allows us to keep track of a back bridge to an already visited node.

For every island we recur on their neighbouring island until we have covered all the islands. During this we update the lowest index island reachable from the current island. Once we have covered all the neighboring islands we check if currently considered island in the first island visited amongst the islands in strongly connected component

Steps involved:

1. Each island has an index which is lowest possible number that can be assigned. It also has a field which keeps track of lowest index island that is reachable from any island using one or more brides or an back edge which connects the island to an already visited node. Each island has an Boolean onStack field which helps us track all the islands that are part of the current strongly connected component.
2. Let resStack be a stack which is used to hold all critical edges.

3. Mark all islands as unvisited initially, all nodes have their onStack field set to false
4. Create an empty stack that can hold all the
5. Loop through all the unvisited islands to perform depth first traversal
   5.1 We assign the island the smallest index possible
   5.2 Mark the current island as visited and set the lowest index island Reachable island as itself initially.
   5.3 Add this island on to the stack
   5.4 Loop through all the neighboring islands of the currently considered Island
      5.4.1 If island is not visited then recur on this island and update the lowest index island reachable from the current node
      5.4.2 If island is already visited and is on stack which holds all the Islands which are part of strongly connected component then just update the lowest index island reachable.
      5.4.2 If island is already visited but is not on stack then the bridges connecting the two island forms the critical bridge. Push this bridge on to resStack
   5.5 If the current island index is equal to the lowest reachable index then All the islands on the stack form a Strongly connected component. pop all these islands from stack and set onStack to false for all these islands to discover other strongly connected components.

6. After all the islands are visited, the resultant array we get will have all the critical bridges amongst the bridges connecting islands.
7. Remove all duplicate entries from resStack and return resStack

Pseudocode:

We assume the island provided is of type struct Island as described below

```
// structure of the island
struct Island
{
        Integer Indx;
        Boolean onStack;
        Integer low;
}
// global variable
Set index =0;

Input: Islands in Micronesia and Bridges which keeps track of all the islands
        connected to a particular island by a bridge
Output: Stack containing all the critical bridges.
FetchAllCriticalBridges(Islands, Bridges)
{

      // Marking all the islands unvisited ,initialize lowest index and stack
```

```
for( isl in islands)
{
      Set isl.Indx = -1
      Set isl.OnStack = false
      Set isl.low = -1
}

Let searchStack  be used to keep track of all the islands of strongly connected
component visited during the search.

Let resStack be used to keep track of all critical bridges

 for ( isl in Islands)
{
            if(  isl.Indx  == -1 )
            {
                  FetchBridgesHelper( isl, searchStack, resStack);
            }
 }

 Remove all the duplicate entries from resStack

// resStack has all critical bridges
return resStack


}

FetchBridgesHelper( isl1, searchStack, resStack, Bridges[])
{

      // set the index of the island to lowest possible value
      Set isl1.Indx = index
      Set isl1.low = index
      index++

      // adding island on to the stack which holds all strongly connected
      // islands
      searchStack(    isl1)
      Set Isl1.onStack = true

      // For all neighbours of isl1
      for  ( isl1, isl2) in Bridges
      {
            // If island is not yet visited then recur on it
            if isl2.indx  ==  -1
            {
                  // Recurse on the neighbours of isl1 if not visited yet
                  FetchBridgesHelper(isl2, searchStack, resStack,Bridges);
                  isl1.low = min(isl2.low, isl1.low)
```

```
            }

            //  If the island isl2 is already visited and is on stack then bridge
            // connecting isl1 and isl2  is a back bridge
            else if( isl2.onStack )
                    isl1.low = min(isl1.low, isl2.indx)

            //  If the island isl2 is visited but not on stack then the bridge
            //  connecting isl1 and isl2 is a critical edge, so push it on result
            //  stack which holds all critical edges
            else
                    resStack.push(is1, isl2)
    }

    // pop out all the islands from the current strongly connected //component, to discover
    another strongly connected component
    If( isl1. low == isl1.indx )
    {
            islRes= searchStack.pop();
            While ( islRes != isl1 )
            {
                    islRes.OnStack = false
            }
            islRes =islsearchStack.pop()
            islRes.OnStack = false
    }

}
```

Analysis of time complexity:

This involves visiting each island and each bridges at most twice. If I is the number of islands and B is the number of bridges then, the time complexity
T(n) = (I + B)

Space complexity = l, as it involves two stack resStack and searchStack to hold all the critical bridges and all the islands of the current strongly connected component traversed, Whose size can be I in the worst case. The other variables used take constant space hence ignored

'I' stands for number of islands in Micronesia and **'B'** stands for number of bridges.