Ashwin Ramesh – Comp2007 Assignment 2

Algorithm Explanation:

To calculate the disconnecting power of each node, all cut vertices must be established. At the same time, the number of nodes that make up each sub tree in the graph must also be calculated.

Once these two factors are known, to calculate the disconnecting power, all that must be done is to iterate through the nodes and establish the various new forests created and use the subtree sizes to calculate the eventuating D.P.

For example, if a cut vertex is removed and all that remains is 3 separate forests of size: 10, 2, 5. The D.P = 10x2 + 10x5 + 2x5 = 80

Detailed Algorithm:

Assume the graph and nodes has the following properties:

```
class graph:
    size,
    root

#this is the node object
class node:
    element,
    parent,
    children, #iterative
    size, # size of tree from below node
    discovery, # discovery time
    finish, # finish time
    downUp, # down&up value
    cutVertex, # true/false if cut vertex
    breakableNode, # true/false if a cut vertex will immediately make this a seperate forest
    visited,
    disconnectingPower
```

Depth First Search Algorithm:

```
def DFS(G):
    for vertex u in G:
        u.visted = False
        u.parent = None
    time = 0
    for vertex u in G:
        if u.visted == False:
            u.size = u.size + DFS_VISIT(u)
    return root

def DFS_VISIT(u):
    time = time + 1
    u.discovery() = time
    u.visited() = True
    for children v in u:
        if v.visted() == False:
            v.parent() = u
            u.size = u.size + DFS_VISIT(v)
    u.size = u.size + 1 #include current node
    time = time + 1
    u.finish = time
    return u.size
```

Cut Vertices Algorithm:

```
def getCutVertex(G):
    root = DFS(G) # get the root of the graph
    if root.children.count > 1:
        root.cutVertex = True # root is a C.V if it has more than 1 child
        for child in root.children:
            child.breakableNode = True # each child in the root is a seperate forest
    for v internal node of T:
        for each child v of u in T:
            if v.downUp <= u.discovery: # if there are no back edges
                v.cutVertex = True
                v.breakableNode = True
            else:
                v.breakableNode = False
    return G
```

Disconnecting Power Algorithm:

```
def DisconnectingPower(G):
    G = getCutVertex(G)
    for vertex in G:
        if vertex.cutVertex == False:
            vertex.disconnectingPower = 0
        else:
            tempCount = G.size - 1 # total number of nodes remaining in the graph
            tempArray = []
            for child in vertex.children:
                if child.breakableNode = True:
                    tempArray.append(child.size) # append size to array
                    tempCount = tempCount - child.size # reduce the size of main forest
                vertex.disconnectingPower = sum(tempCount * tempArray elements)
                # multiply main forest by all sub forests
    return G
```

Proof of Correctness:

The main point to note is that if we remove a cut vertex, all nodes above and to the left and right are still connected, making a “major” forest.

This means that the only calculation required is to determine the size of all the smaller forests below and to take into account any back edges that will further increase the size of the “major” forest”

Once the size of the major forest is calculated and all the sizes of the smaller forests are also determined, it is only a matter of multiplying and summing up the results to get the final Disconnecting Power.

The root node is only a C.V. if it has more than 1 child. This is obvious because two or more sections/forests will be created.

Leaf nodes have no children thus they are never C.V.

Inner nodes, as mentioned above can be cut vertices only if there is a back node attributing from all children of the C.V. back above the C.V.

As long as these conditions are adhered by, the final result will always be correct.

Time Complexity:

Depth First Search: run in O(m) time as the graph is connected. (m = n i.e. O(2m) = O(m))

Cut Verticies: Runs in $O(n^2)$ time as the two for loops will iterate a maximum of n times. The DFS is added on, but O(m) < $O(n^2)$, thus meaning $O(n^2)$ is the complexity

Disconnecting Power: Runs in $O(n^2)$ also as the first for loop runs in O(n) time and the second will sum up to O(n) time. Adding the C.V. algorithm still ensures that this will run in $O(n^2)$