Shaival Vora(G37099269), Devershi Patel(G35175105), Vigna(G26660369), Naveena(G24275981)

Algorithms Project 3

1 Problem Analysis

Question Biconnectivity: Given a graph G, check if the graph is biconnected or not. If it is not, identify all the articulation points. The algorithm should run in linear time. Use the given input sets as tests.
We were tasked with developing a graph biconnectivity algorithm that employs recursive depth-first search to find articulation points. Execution time depends on graph size and connectivity, with larger or denser graphs requiring more analysis time. We compare theoretical and experimental execution times, adjusting theoretical values based on code runtime. Biconnectivity is achieved when any vertex can be reached from any other through a simple path, and removing any vertex does not disconnect the graph. Such nodes causing disconnection are termed articulation points. To determine biconnectivity, we calculate Depth First Number (DFN), the Lowest DFN reachable from a node, and identify articulation points based on conditions, mainly ensuring L[v] >= L[u], except for the root which becomes an articulation point when it has multiple children. The code serves various applications, such as network analysis and critical point identification in graphs.

2 Pseudocode

```
Function generate_graph(num_nodes, num_edges_per_Node):
    G = create an empty graph
    nodes = create a list of nodes from 1 to num_nodes
    shuffle the list of nodes to randomize node selection
    while the list of nodes has more than one node:
        node = pop the first node from the shuffled list
        # Randomly select other nodes to connect to, limited by num_edges_perNode
         edges = randomly select a sample of nodes from the remaining nodes, up to  num_edges_perNode
         connect node to the selected edges in the graph G
    return G

Function Biconnectivity(input: graph G)
   Stack T;
   Integer num = 1;
   Integer DFN[l:n], L[l:n], Parent[l:n]
   Node s = an unvisited node
   L[s] = DFN[s] = num++
   mark s as visited
   T.push(s)
   While (stack T is not empty) do
     Node x = top(T)
     if (x has an unvisited neighbour y) then
mark y as visited T.push(y)
           DFN[y] = num++
Parent[y] = x
L[y] = DFN [y]
     else pop(T)
     for (every neighbor y of x) do
           if (y != parent[x] and DFN[y] < DFN[x]) then 1* y is an ancestor of x, and (x,Y) is a back edge*1
          L[x] = min(L[x] ,DFN[y])
else if (x = Parent[y]) then
L[x] = min(L[x] ,L[y])
```

```
if (L[y] ~ DFN[x] and xis not root) then
        return x as an articulation point if (s has more than one child) then
return s as an articulation point return true II Graph G is biconnected.
```

3 Time Complexity

The time complexity for a brute force approach to finding if a given undirected graph is biconnected is typically **O(V * (V + E))**, where V is the number of vertices, and E is the number of edges in the graph. This is a relatively inefficient approach, especially for large graphs, as it involves multiple DFS traversals and can be computationally expensive.

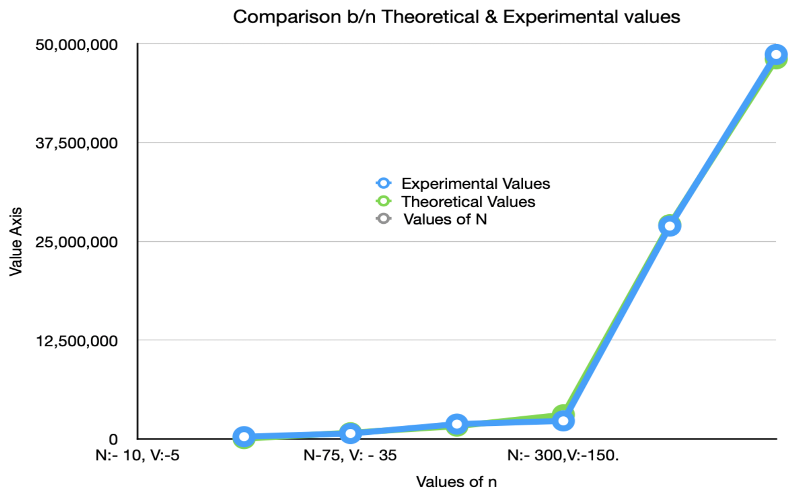
The primary part of the algorithm involves a depth-first search traversal of the graph, where each node is visited once. This part has a time complexity of O(V + E), where V is the number of vertices, and E is the number of edges.Marking nodes as visited, pushing them onto the stack, and updating various arrays like DFN, L, and Parent all take constant time for each node. The loop over neighbors of a node involves a constant amount of work per neighbor. The dominant part of the algorithm is the DFS traversal, which is O(V + E).So, the overall time complexity of the biconnectivity algorithm is O(V + E), where V is the number of vertices, and E is the number of edges

4 Time Complexity

N	Experimental Values, ns	Theoretical Result	Scaling Constant	Adjusted theoretical values
N:- 10, V:-5	279620	60		35900.0176479311
N:-50,V:-25.	679380	1300		777833.715705175
N-75, V: - 35	1868309	2700		1615500.7941569
N:- 100,V:-50.	2276560	5100		3051501.50007415
N:- 300,V:-150.	26932618	45300		27104513.324188
N:- 400,V:- 200.	48654786	80400		48106023.6482277
	13448545.5	22476.6666666667	598.333627465519	

5 Graph

.



6 Github Link

https://github.com/shaival-vora/CSCI_6212_Algorithms