***Faculty of Mathematics and Information Science***

**Functional Documentation : EDGE CONNECTIVITY**

**FONTAINE Victor**

**BONAVENTURE Ashmitha**

**• Problem description :**

Given an undirected graph G(V,E), we have to find the edge connectivity of the graph.

***V*** - Vertices, ***E*** - Edges.

***Edge*** ***Connectivity*** - the minimum number of edges, whose deletion from the input graph disconnects it.

**• Input :**

The input will be a text file (input.txt). The first line indicates the number of vertices(N) and the lines that follow it represents the edges present in the graph.

The format from the second line will include two integers, say x y separated by a space. x y pair indicates the edges present in the graph.

4

1 2

2 3

3 4

1 4

1 3

Where the first line represents the number of vertices(N) and the next lines represent the edges present in the graph.

**• Output :**

The output will be an integer which represents the edge connectivity. The output for the graph mentioned in input will be 2.

**• Solution description:**

The global edge connectivity of an undirected graph is the minimum number of edges that must be removed to disconnect the graph. The edge connectivity of an undirected graph G=(V,E) can be determined by running the maximum-flow algorithm times |V|-1 times, each on a flow network with O(|V|) vertices and

O(|E|) edges.

***Solution:***

Construct a directed graph G’ from by replacing each edge {u,v} in G by two directed edges (u,v) and (v,u) in G’. Let g(u,v) be the maximum flow value from u to v through G’ with all edge capacities equal to one. Pick an arbitrary node u and compute g(u,v) for all v!=u. We claim that the edge connectivity f\* equals minv!=u g(u,v). Therefore the edge connectivity of G can be computed by running the

maximum-flow algorithm |V| -1 times on flow networks each having |V| vertices and 2|E| edges. Suppose k is the edge connectivity of the graph and Q is the set of k edges such that removal of Q will disconnect the graph in two non-empty subgraphs G1 and G2. Without loss of generality assume the node u belongs to G1. Let w be a node in G2. Since u!=w the value g(u,w) will be computed by the algorithm. By the max-flow min-cut theorem, g(u,w) equals the min cut size between the pair (u,w) which is at most k since Q disconnects u and w. Therefore, we have

**f\* <= g(u,w) <= k**

But f\* cannot be smaller than k since that would imply a cut set of size smaller than k, contradicting the fact that k is the edge connectivity. Therefore f\*=k and the algorithm returns the edge connectivity of the graph correctly.

I – Read the input file F and create a graph G with an adjacency matrix to describe it:

1 – n ← F.ReadLine()

3 – while (F.nextLine() != EndOfFile)

2 – G.addEdge (F.nextInt(), F.nextInt())

II – Find the edge connectivity using Max-Flow algorithm:

int Edge-Connectivity(Graph G)

{

int min = infinite;

for (Vertex u: G.V){

for (Vertex v: G.V){

if (u != v){

/\*create directed graph G(u,v) (a graph with directed edges and source u and sink v) \*/

/\*run Ford-Fulkerson algorithm to find the maximum flow |f\*|

**Inputs** Graph G with flow capacity c, a source node s, and a sink node t

**Output** A flow f from s to t which is a maximum

1. f(u,v) <— 0 for all edges (u,v)

2. While there is a path p from s to t in Gf such that cf(u,v)>0 for all edges (u,v) belongs to p.

1. Find cf (p) = min {cf(u,v) : (u,v) belongs to p }

2. For each edge (u,v) belongs to p

1.f(v,u) <— f(v,u) - cf(p)the flow might be returned later

\*/

if (min > |f\*|)

min = |f\*|;

}}}

return min;

}

**• Correctness of the algorithm:**

The flow returned by the algorithm is the maximum flow.

Proof.

1.For any flow f and s-t cut(A,B),v(f) ≤ c(A,B)

2.For flow f∗ returned by algorithm, v(f∗)=c(A∗,B∗)for some s-T cut (A∗, B∗) Hence, f ∗ is maximum

**• Time complexity :**

Finding an augmenting path requires a depth-first search of the graph, which takes O(|E|) time. We have to find a new augmenting path each time the algorithm does another iteration. Since we can do at most f iterations, and each iteration takes O(|E|+|V|) time. The algorithm for edge-connectivity would take O(|V| ^ 2) number of flow networks.