**Design and Analysis of Algorithm**

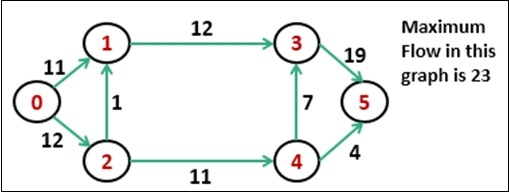
**Experiment No. : 11**

**Write a program to Implement Ford-Fulkerson algorithm for maximum Flow Problem**

Experiment No. 11

1. **Aim:** Write a program to Implement Ford-Fulkerson algorithm for maximum Flow Problem.
2. **Algorithm**

The Ford-Fulkerson algorithm is used to detect maximum flow from start vertex to sink vertex in a given graph. In this graph, every edge has the capacity. Two vertices are provided named Source and Sink. The source vertex has all outward edge, no inward edge, and the sink will have all inward edge no outward edge.



There are some constraints:

Flow on an edge doesn’t exceed the given capacity of that graph.

Incoming flow and outgoing flow will also equal for every edge, except the source and

the sink.

**Algorithm**

**bfs(vert, start, sink)**

**Input:** The vertices list, the start node, and the sink node.

**Output** − True when the sink is visited.

Begin

initially mark all nodes as unvisited

state of start as visited

predecessor of start node is φ

insert start into the queue qu

while qu is not empty, do

delete element from queue and set to vertex u

for all vertices i, in the residual graph, do

if u and i are connected, and i is unvisited, then

add vertex i into the queue

predecessor of i is u

mark i as visited

done

done

return true if state of sink vertex is visited

End

**fordFulkerson(vert, source, sink)**

**Input:** The vertices list, the source vertex, and the sink vertex.

**Output −** The maximum flow from start to sink.

Begin

create a residual graph and copy given graph into it

while bfs(vert, source, sink) is true, do

pathFlow := ∞

v := sink vertex

while v ≠ start vertex, do

u := predecessor of v

pathFlow := minimum of pathFlow and residualGraph[u, v]

v := predecessor of v

done

v := sink vertex

while v ≠ start vertex, do

u := predecessor of v

residualGraph[u,v] := residualGraph[u,v] – pathFlow

residualGraph[v,u] := residualGraph[v,u] – pathFlow

v := predecessor of v

done

maFlow := maxFlow + pathFlow

done

return maxFlow

End

**Conclusion and Discussion:** The analysis of Ford-Fulkerson depends heavily on how the augmenting paths are found. The typical method is to use breadth-first search to find the path. If this method is used, Ford-Fulkerson runs in polynomial time.