Algorithms to find Max-Flow in a network.

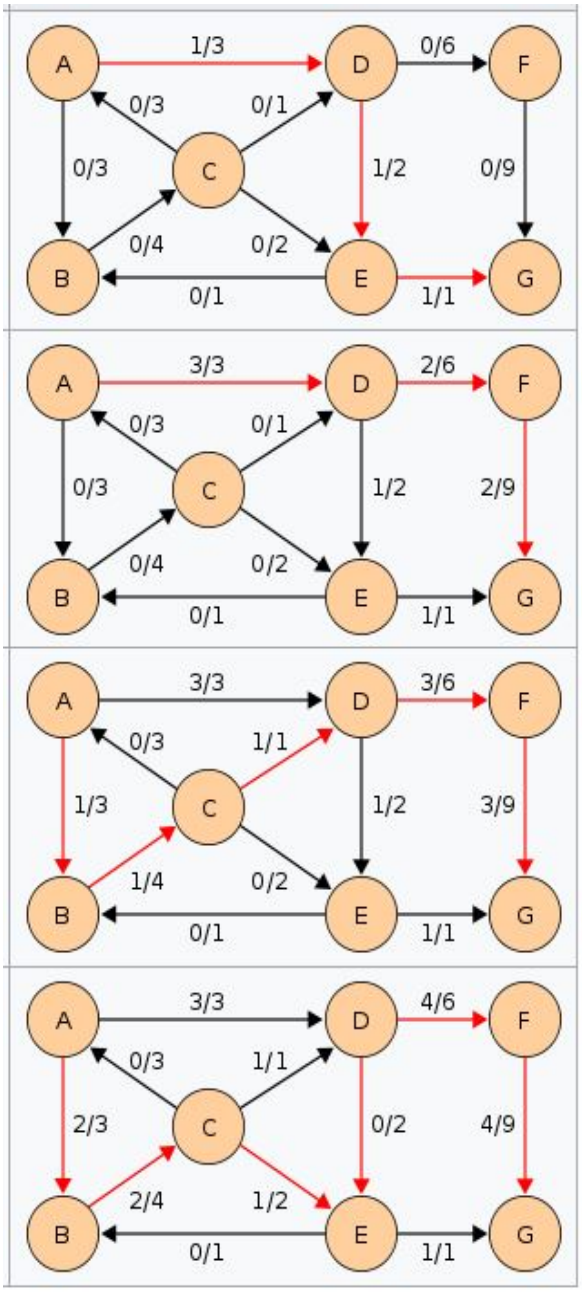
We will discuss two algorithms to find the maximum flow in the network along with their examples and complexities.

1. Edmonds Karp Algorithm

First, we set the flow of each edge to zero. Then we look for an augmenting path from s to t. An augmenting path is a simple path in the residual graph, i.e. along the edges whose residual capacity is positive. If such a path is found, then we can increase the flow along these edges. We keep on searching for augmenting paths and increasing the flow. Once there doesn't exist an augmenting path anymore, the flow is maximal.

Let us specify in more detail what increasing the flow along with an augmenting path means. Let C be the smallest residual capacity of the edges in the path. Then we increase the flow in the following way: we update f((u,v)) += C and f((v,u)) -= C for every edge(u,v) in the path.

Here is an example to demonstrate the method. We try to find the maxFlow between A and G.



We start with 0 flow on each edge. And in every stage, we find an augmenting path (shown by red colour) such that the residual capacity of this path is > 0. We now observe that there is no augmenting path left in the graph. Hence, we report the current flow in the network as maximal.

Here is the implementation of the above idea.

```
/*
        Function to find an augmenting path between source s
        and sink t using breadth-first search
```

```
*/

function bfs(s, t) {

	//   empty queue for bfs to store the node and the flow till that node
	q = queue()
	q.push({s, INF})
	vis = array[n]
	while (!q.empty) {
		cur, flow = q.front()
		q.pop()
		for (edge : adj[cur]) {
			if (!vis[edge.next] && edge.capacity) {
				vis[next] = 1
				//   update the value of minimum capacity till here
				new_flow = min(flow, edge.capacity)
				//   if sink is reached return
				if (edge.next == t)
					return new_flow
				q.push({edge.next, new_flow})
			}
		}

	//   return 0 if no augmenting path is found
	return 0
}



/*
	Function to find the max flow between source s
	and sink t using Edmonds Karp max flow algorithm
*/
function EdmondsKarp(s, t)
```

```
        flow = 0
        // while there exists an augmenting path
        while (new_flow = bfs(s, t)) {
                flow += new_flow
                // update capacity of each edge in the path
                for each edge in augmenting_path
                        edge.flow += new_flow
                        reverse(edge).capacity -= new_flow
        }
        // return maxflow
        return flow
}
```

Time Complexity: The algorithm runs in $O(VE^2)$ time, even for irrational capacities. The intuition is, that every time we find an augmenting path one of the edges becomes saturated, and the distance from the edge to s will be longer if it appears later again in an augmenting path. And the length of simple paths is bounded by V.

Space Complexity: Since we are using adjacency list representation, the space complexity of the above algorithm is the same as that of bfs which is O(V+E).