Algorithm Analysis
Lab 2 : Implementation of Ford-Fulkerson Algorithm to Find Maximum Flow in a S-T Flow Graph

---

AIM : To implement Randomized Quick Sort algorithm and compare its performance with the traditional fixed pivot quick sort.

Max Flow Problem

A Flow Network is a directed graph $G(V, E)$, where capacity of every edge $e \in E$ is defined by weight W_e, a positive integer. The network has two special vertices, Source (S) and Sink (T). As the name suggests, the Source Vertex has no incoming edge and the Sink vertex has no outgoing edge.

An $S-TFlow$ in a flow network is function f, that assigns each edge e, from S to T, a non-negative integer value, namely the flow. The function has to fulfil the following two constraints,

(1) Capacity Constraint : The flow on every edge can not exceed the capacity W_e.

$$f_e \leq W_e$$

(2) Equality Constraint : The inflow and outflow at any vertex, except the source and sink, must be same.

$$\forall v \in V \neq S, T, \sum f_{in} = \sum f_{out}$$

The goal of the maximum flow problem is to find out the maximum flow from S to T, which satisfies the about constraints.

An Example

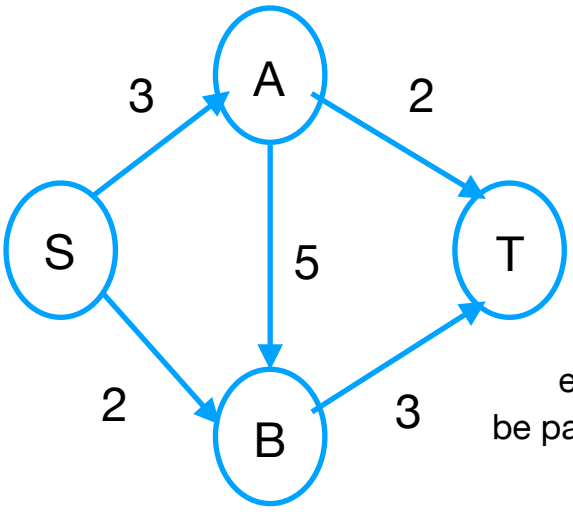

Figure 1 (A) shows a flow network with 4 nodes. S is the starting node, with only outgoing edges; and T is the sink node with all the incoming edges. A & B are the intermediate nodes. The edge weights indicates the integer flow capacity of every edge, e.g., maximum 3 units of flow which can be passed from S to A.

Figure 1 (A) : Flow Network Example

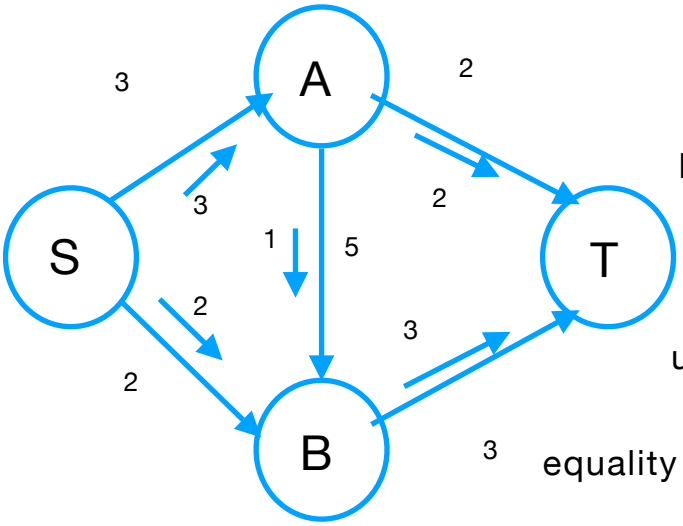

Figure 1 (B) : small arrows along the size of edges show the flow through the edge.

Figure 1 (B) shows the Maximum Flow which can be sent from S to T. It sends the maximum flow along the three paths, 2 units of flow on S-A-T, 1 unit of flow through S-A-B-T and two units of flow through S-B-T.

You may verify that the capacity and equality constraints are satisfied and the maximum flow of 5 units is achieved.

Ford-Fulkerson Algorithm

The idea behind the algorithm can simply be summarised as follows,

Iteratively explore all S-T path and try to send maximum flow along each of the path. The maximum flow which can be sent along an S-T path is defined by the bottleneck capacity of the path. For example, in Figure 1 (A), bottleneck capacity of the path S-A-T is 2.

Once a flow is sent through an S-T path, the capacity of the edges would be modified which will require recalculation of flow along the other paths. The algorithm achieves this by defining residual graph and augmenting path.

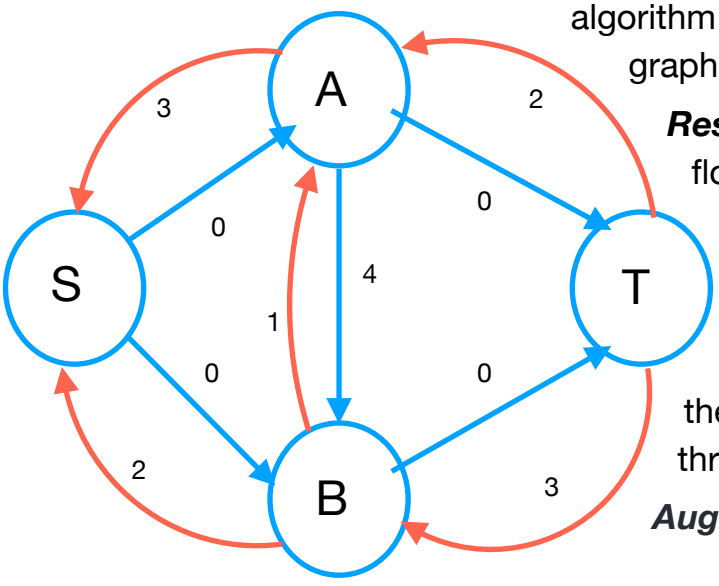

Figure 1 (C) : Residual Graph of the flow graph shown in Figure 1(B)

Residual Graph : Residual Graph of a flow graph contains two edges between every pair of nodes. The forward edge, which is also present in the original flow graph, shows the remaining capacity of the edge; and the backward edge shows the flow sent through the edge.

Augmenting Path : An augmenting path is a path from source to sink which do not include any cycles and contains only positive weighted edges.

The Figure 1(C) shows the residual graph for the figure 1(B). The blue edges are the forward edges, which are also present in the original flow graph. The edge weights of the forward edges are changed to the actual available capacity. The red edges are called the backward edges, these are new edges added in the residual graph. The revers edges show the flow sent through the nodes in reverse direction.

The outline of the algorithm is shown in the algorithm 1.

Algorithm 1 Ford-Fulkerson Algorithm((Graph G, Source S, Sink T)

```
flow = 0
G' = create_residual_graph(G, S, T)
Augmenting path = find_path(G', S, T)
while there is a (augmenting) path, p, from s -> t in residual network G_f:
    residual_capacity(p) = min(capacity(u, v) : for (u, v) in p)
    flow = flow + residual_capacity(p)
    for each edge (u, v) in p:
        if (u, v) is a forward edge:
            capacity(u, v) = capacity(u, v) - residual_capacity(p)
        else:
            capacity(u, v) = capacity(u, v) + residual_capacity(p)
    find_path(G', S, T)
return flow
```

As shown in algorithm 1, The algorithm start with initial flow 0. It constructs the residual graph and finds an augmenting path. It calculates the residual capacity of the augmenting path, which is bottleneck edge capacity, and adds the value to the flow count. It means that the flow = “residual_capacity” is sent through the augmented path.

Next step is to update the residual graph to reflect the flow sent through the chosen path. The flow amount id reduced from the forward edge capacity and added to the backward edge capacity.

Working Example

To demonstrate the working of the algorithm, let's consider the example graph shown in the Figure 2(A). As shown in the algorithm 1, after initialising necessary variables, the algorithm finds s-t paths in the (residual) graph. This requires a call to BFS / DFS (Task 1 : Implement BFS / DFS to find the augmenting path). Let's say the first path returned by the call to find_path, is S-C-D-T.

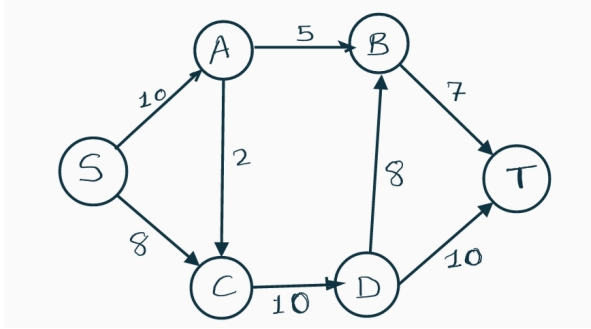

Figure 2 (A) : Input graph instance for Ford-Fulkerson Algorithm

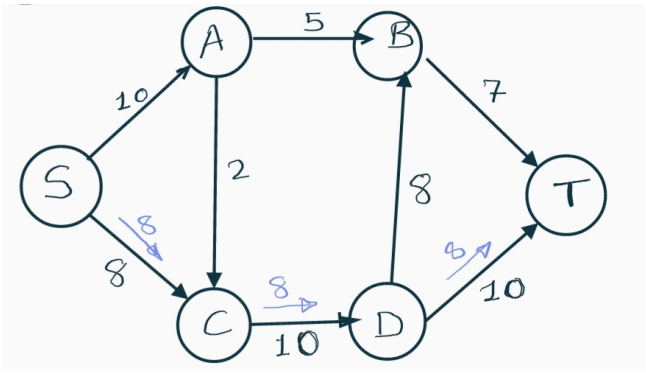

Figure 2(B) : Flow graph with S-C-D-T flow

Inside the while loop the residual capacity of the path S-C-D-T will be calculated, which is 8. The flow value will be incremented by 8 and the forward and backward edges of the residual graph will be updated to subtracting and adding 8, respectively. The resulting residual graph is shown in the Figure 2(C).

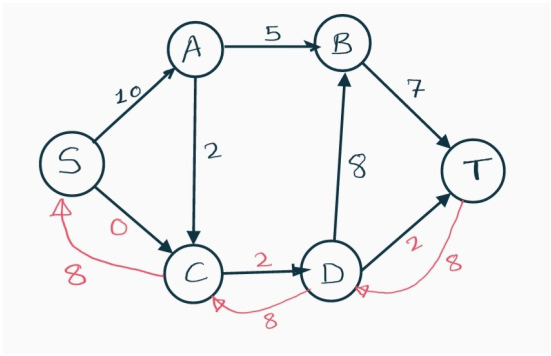

Figure 2 (C) : Residual graph corresponding to the flow graph of Figure 2 (B)

After updating the residual graph as shown in the Figure 2 (C), the algorithm will again find the next augmenting path and repeat the procedure. Let's say the next augmenting path is S-A-B-T, through which an additional flow of 5 units can be sent. The total flow through two paths will be 13. The new flow edges and correspondingly updated residual graph will be as shown in the figure 2(D) and figure 2(E).

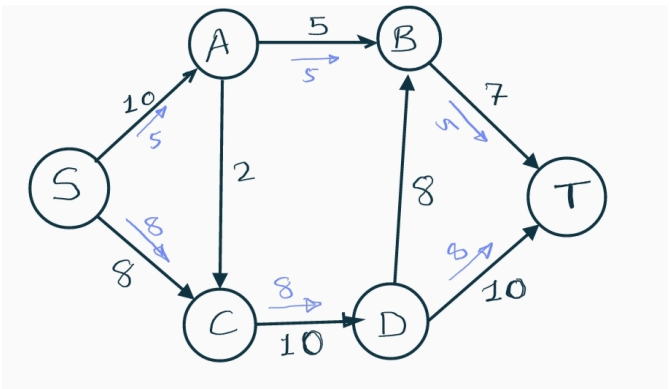

Figure 2(D): Flow graph with S-A-B-T edge selected

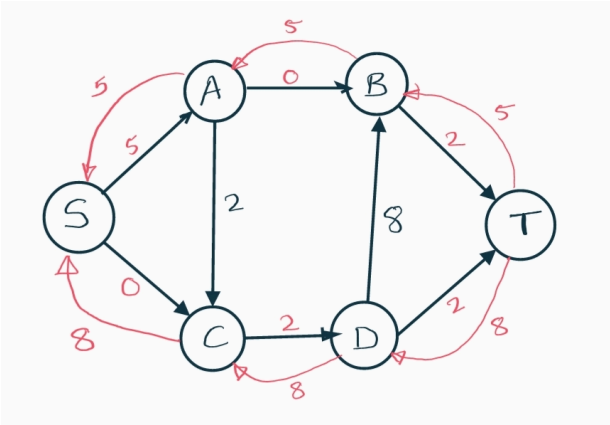

Figure 2(E): Residual Graph corresponding to the Flow graph of Figure 2(D)

figure 2(E). Discarding edges with weight = 0 and avoiding cycle, S-A-C-D-T is the only one augmenting path

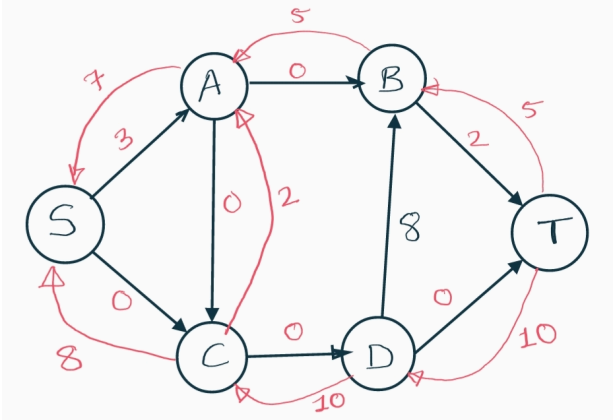

Figure 2(G): Residual graph corresponding to the flow graph of Figure 2(F)

respectively.

As can be seen in the residual graph of figure 2(G), there are now augmenting path available to send flow from s to t. Hence the algorithm stop here and return Max Flow = 15, as the output. The final flow graph corresponding to the residual graph shown in figure 2(G) is shown in the figure 2(H).

The algorithm again finds augmenting path in the residual graph shown in the

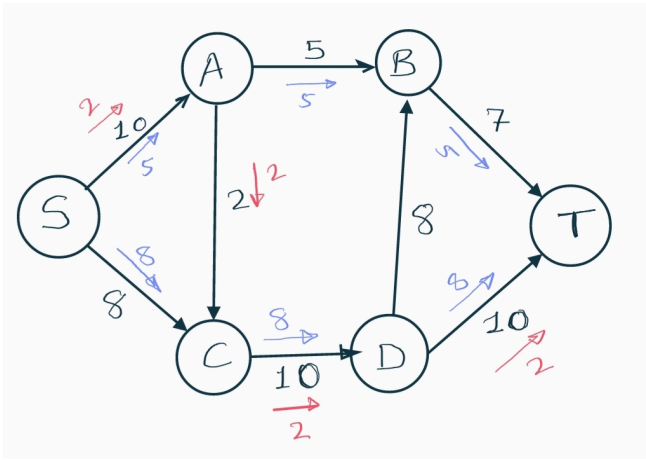

Figure 2(F): Flow graph after selecting edge S-A-C-D-T

left. 2 Units of flow will be sent through that path, which makes the total flow = 15. The chosen path and resulting residual graph are shown in figure 2(F) and 2(G)

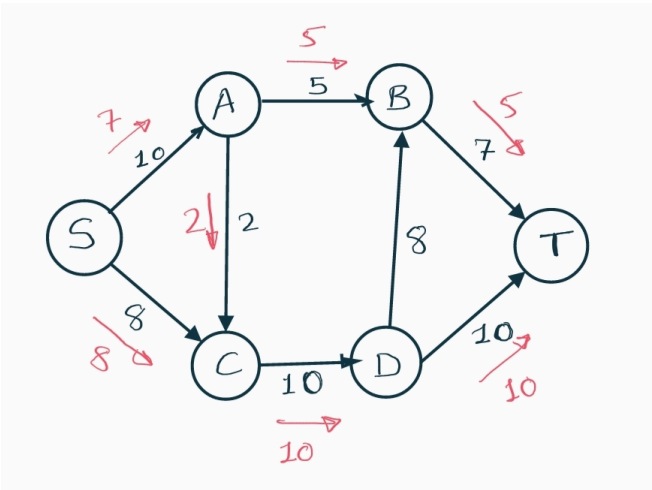

Figure 2(H): Resultant flow graph with Max Flow = 15.

Implementation Notes and Complexity Analysis

Exercise

(A) If we use a simple greedy strategy to select S-T path with maximum bottleneck capacity, what will be the output for the graph 1 (A)?

(B) What is the complexity of Ford-Fulkerson Algorithm? Comment on its running time.

(C) Give an example flow graph for which the Ford-Fulkerson algorithm may give worst case behaviour.