ASSIGNMENT 5:

Design Analysis of Algorithm

Animesh Patni
A20403240

Question 1:→

In this problem, we are going to modify the max-flow algorithm, to get feasible flow f'. The step to achieve this are as follows:-

① As given in the problem, we have a flow f which has f(e) = 1.

② Now, run the DFS algorithm on the input graph G_f.

③ If DFS detects a cycle, in the graph G_f, we subtract 1 from each edge of the cycle. By subtracting 1 f'(e) becomes 0 and the flow f = f'. update the residual graph $G_{f'}$ according to changes made in G_f. The cycle in this step should have an edge e pointing to the

source, then only subtract 1.

(4) Now run the algorithm to find the augmenting path in G_f (residual graph) for any augmenting paths, if some augmenting path exists, check whether that path is forming a cycle in G (original graph) or not, using DFS like we did in step 3. If cycle exists (including the edge e) disregard that path else add the value to f', which would eventually takes $f' > f$.

(5) If no augmenting path exists in residual graph G_f, break the algorithm return f' which would be either greater than or equal to f.

As, we are using DFS after each iteration to detect the cycle as well as the augmenting path, the running time would be $O(V+E)$.

Using the above algorithm we will have $f'(e) = 0$ and $|f| \leq |f'|$.

The proof of correctness for the above algorithm, specially the main step 6 goes as follows:→

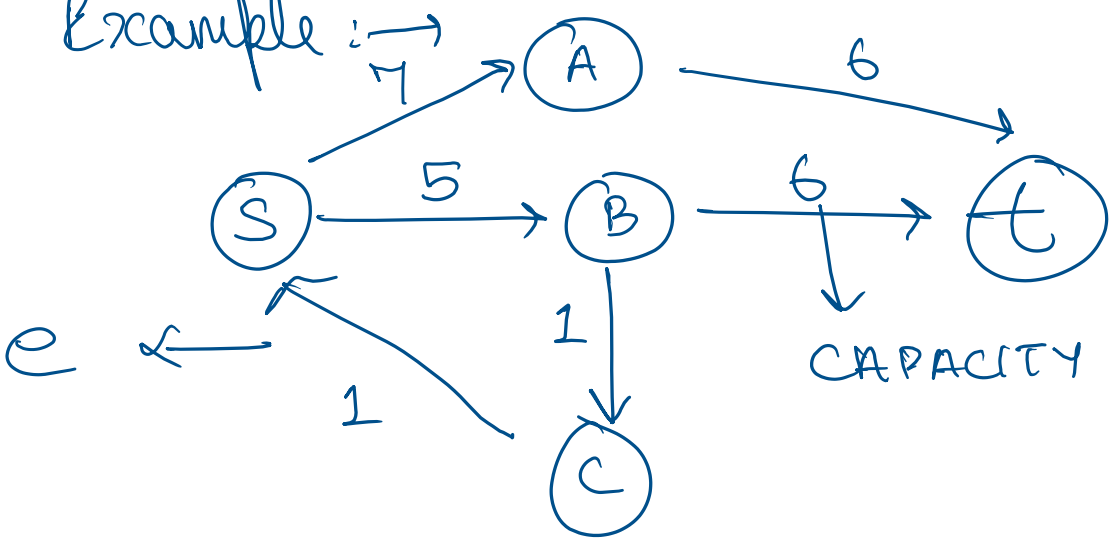
As $f(e) = 1$, e being the edge entering into the source s, such that $B \longrightarrow s = 1$ ie B is flow connected to s through the path p. Suppose path p' is a cycle that has positive flow on each edge i.e $s \rightsquigarrow B \longrightarrow s$. Now if

we subtract 1 from each edge of the cycle p' and obtain a flow f', then f' satisfies the property of flow ie $f'(e) < c(e)$ and flow into B will be equal to flow out of B. The edge $f'(B \to S) = f(B \to S) - 1$

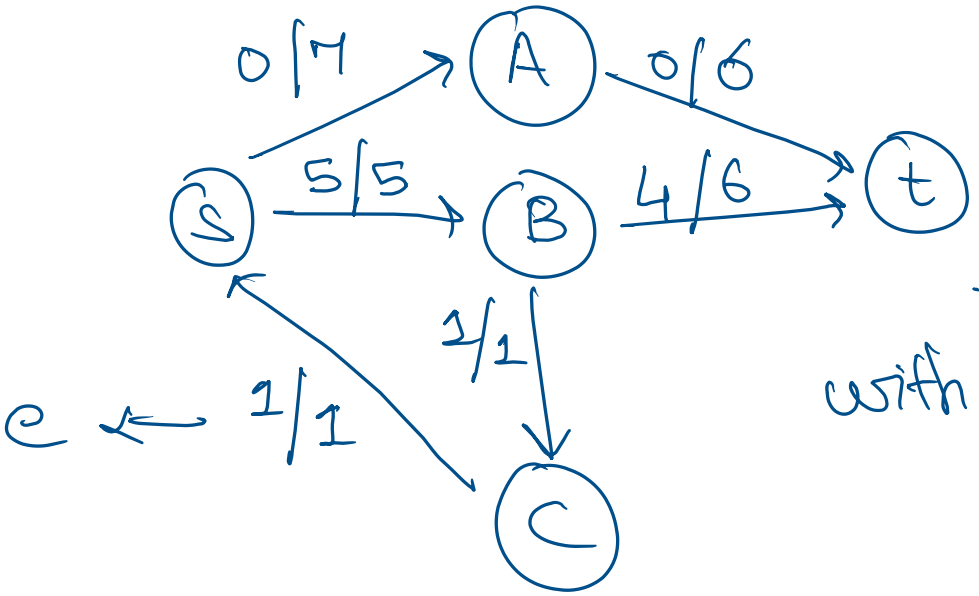
$= 0$.

There can be a possibility where f' becomes greater than f as shown in step 4. Hence proved.

Example :



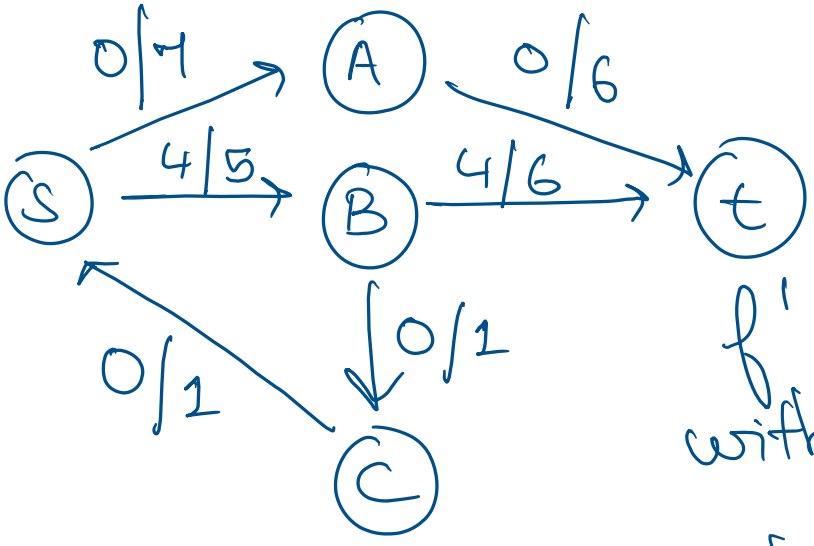
flow network provided :→



$f = 4$

with $f(e) = 1$

now, running step 3 on the above graph we have :→



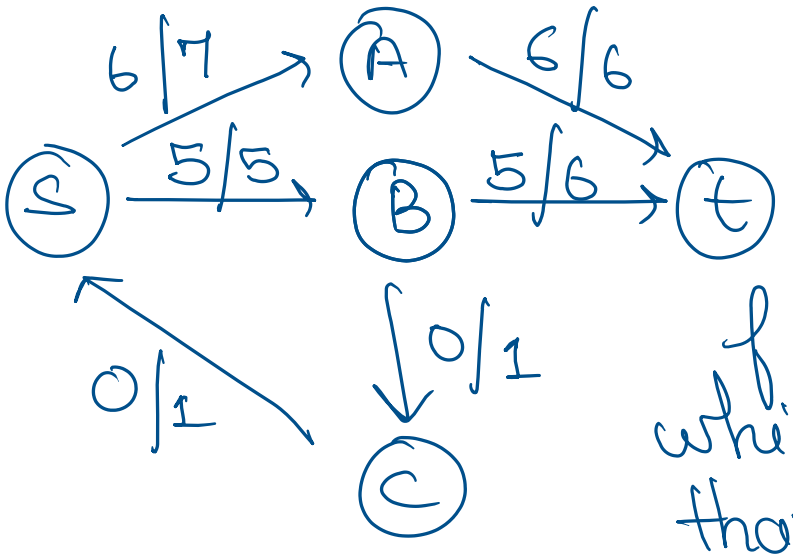
$f' = f$

$f' = 4$

with $f'(e) = 0$.

We can still maximize the flow using step 4, so the original graph G

after running step 4 (all iterations) we have:→



$f' > f$

$f' = 11$ which is greater than f and $f(e) = 0$.

Both the conditions are satisfied.

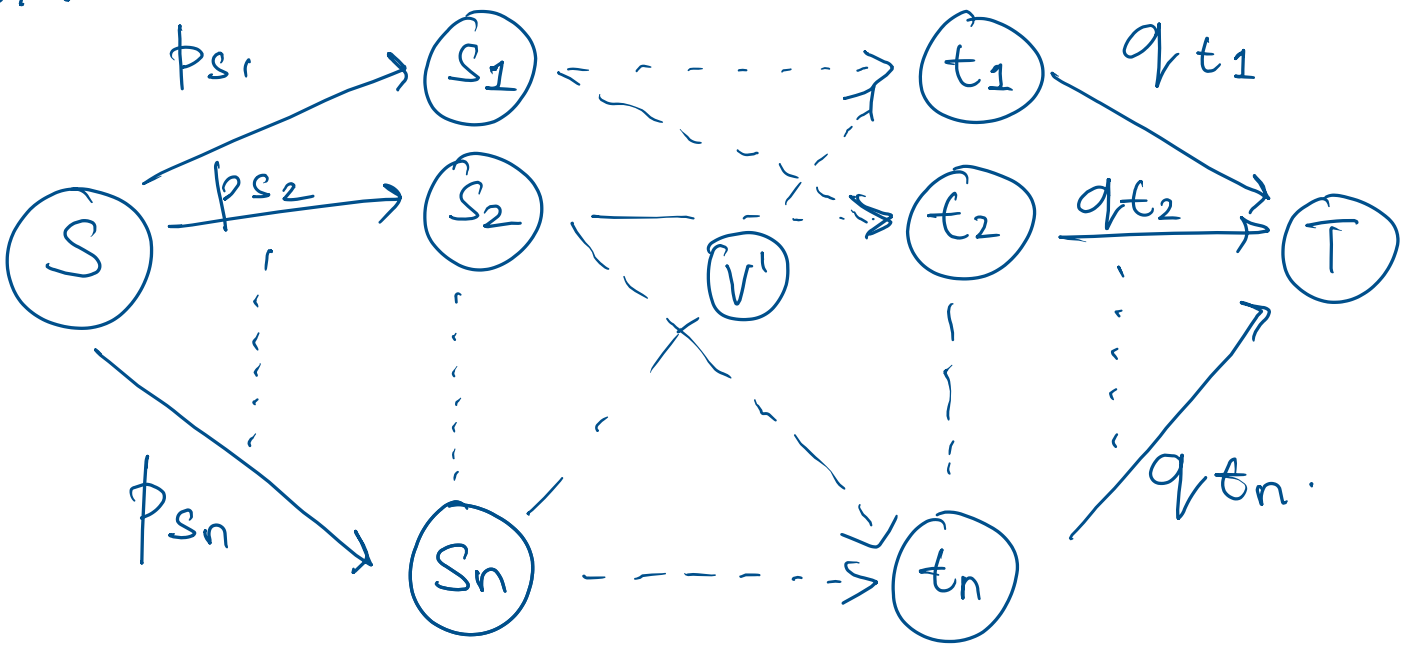
P.T.O→

Question 2:→

In this problem to solve the multiple source-sink problem, I am going to use the concept of super-source & sink method as explained in the section 26.1. As we are given p_{si} for every source and q_{ti} for every sink, we assign supersource to s_1, the capacity as p_{s1}, we would do this for all the sources connected to supersource. Similarly for t_1 to supersink we assign capacity as q_{t1}, we would do this for all the sinks connected to supersink. After this there will be a new graph G'. The graph below clarifies the concept of G':→

P.T.O.→

G' :→



Algorithm for this problem is :→

(1) $\Sigma p_{si} != \Sigma q_{ti}$, we return NO else go to the next step.

(2) Run the max-flow algorithm :→

After running the algorithm, we need to check if $(\sum_{i=1}^{n} f(p_{si}) == \sum_{i=1}^{n} f(q_{ti})$ and max-flow $== \sum_{i=}^{n} (q_{ti}))$ if yes then check for each edge value q_t in G as well as of G'.

a.) We check for each q_{ti} in G and q_{ti} in G', if they are equal we return YES.

$q_{ti} = \{ q_{t1}, q_{t2} \cdots q_{tn} \}$ in G

$q_{ti} = \{ q_{t1}, q_{t2} \cdots q_{tn} \}$ in G'

b.) If it is not equal for any single edge we return NO.

Running Time Analysis for the algorithm:→

As, we have the major part in Ford-Fulkerson's algorithm, the running time would be $O(E \times f)$, E being the number of edges and f being the maximum flow.

NO → If means the flow is not valid

YES → If means the flow is valid.

Proof of Correctness:→

So, to prove the algorithm we assume that there is a feasible flow (f) in G. The net flow coming out of all the sources is equal to total

demand (all sinks). Now lets go for G', we saturate all the edges coming out of supersource and all the edges coming into supersink. We compute f' in G'. Now if f' satisfies all the flow condition, we prove that f' is a valid flow in G' for this we have that capacity of each edge e in G' is greater than or equal to flow on each edge in G'. Each vertex leaving from supersource have flow in = flow out of vertex, similarly for supersink, each vertex entering into supersink obeys flow in = flow out. As the edges leaving the supersource and entering supersink

are saturated, and all the source vertices and sink vertices have reached there desired flow and all the other vertices v in G_f' satisfy the flow conservation as flow into → v = flow out of v. Our main intuition is that G_f is same as G_f'. So, if the flow exists in G_f, it should exists in G_f' and visa-versa. Hence Proved.

Question 3:→

For any s and t in a graph G, As graph in the problem is undirected, we create a directed version of that graph called G, with each edge capacity equals to 1. In this case we can have anti-parallel edges as it is a multi-graph; we can normalize it by using the approach in section 26.1 in book, which creates a dummy node v' between one edge of the parallel edge $u \rightarrow v$. Such that capacity $u \rightarrow v' = v' \rightarrow v =$ capacity of $u \rightarrow v$. As the graph is undirected so, we have $O(V)$ vertices & $2 \times E$ edges from $A \rightarrow B$ & $B \rightarrow A$. So, it satisfies $O(E)$ edges as well. The capacity is kept as 1, so that the number of edges in G crossing a

cut equals to the capacity of the cut $G_{s \to t}$.

f be the flow in $G_{s \to t}$.

The algorithm goes as follows:→

```
1) DISCONNECT(G, S, t):
2)       K → 99999999
3)   LOOP THROUGH ALL VERTEX EXCEPT S:
4)     GET THE FLOW G s→t
5)     GET THE MAX-FLOW f
6)           K → minimum of( K , f)
7)   PRINT( K ).
```

s→ can be any vertex in G, which can be the source. Since our main goal is to remove edges so that graph becomes disconnected,

so we have the max-flow, the number of edges needed to be removed so that s and t are in different components. That is the reason we consider all n-1 flows. We know that after removing smallest number of edges in G, s will be in different component but to get that smallest number we need to have a proper flow through t. and t be that sink vertex. So, we prove that

$$K = \min_{v \in V - \{s\}} \{f_{s \to t}\} \quad -(1)$$

As the capacity of each edge is 1, so the capacity of each cut through max-flow min-cut theorm is the

number of edges crossing it. I prove the above equation (1) in two separate cases :→

case 1:→ $K <= \min_{v \in V - \{s\}} \{f_{s \to t}\}$

We have a vertex t, with the minimum f. There is a cut of capacity of f that disconnects s and t. ∴ As said above exactly f edges cross this cut. If all these edges are removed the graph is disconnected.

case 2:→ $K >= \min_{v = V - \{s\}} \{f_{s \to t}\}$

In the worst case there might come a situation that after removing n - 1 edges with minimum capacity, still

the vertex $s \rightarrow t$ are connected, this means we have to remove n-edges to completely disconnect s & t. This proves the above expression

Thus the claim that

$$K = \min_{v \in V - \{s\}} \{f_{s \rightarrow t}\} \text{ holds.}$$

Explaining, the above algorithm, we loop through all the vertices except the source, find the minimum edges that are required to disconnect the graph and return that value. As we are looping through all the vertices we use $|V|$ flow networks for the same. We keep on changing the source till we find the minimum number

of edges / capacity to disconnect the graph.