Analysis of Algorithms

(5CS4-05/5IT4-05)

Unit 4

Flow Network Problems

Ramakant Soni

Assistant Professor, Computer Science Department,

B K Birla Institute of Engineering & Technology, Pilani,

Rajasthan

Contents

- Randomized Algorithms
- Maximum Flow using Ford Fulkerson Algorithm
- Multi Commodity Flow Problem
- Flow Shop Scheduling

Randomized Algorithm

Randomized algorithms :- Algorithms that make use of randomness in their computation.

These algorithms use uniformly random bits as an auxiliary input to guide its behavior, in the hope of achieving good performance in the 'average case' over all possible choices of random bits.

Type of randomized algorithms :-

1. Las Vegas algorithms
2. Monte Carlo algorithms

Randomized Algorithm

Las vegas Algorithms :- The solution will always be correct and the time is not bounded. When run for specific time may not give solution.

Monte Carlo Algorithms :- These are probabilistic algorithm. The runtime of such algorithm is bounded. Has a slight probability of producing an incorrect result or may also fail to give any result.

Randomized Algorithm

Randomized Algorithm for Min-Cut Problem. ← imp

Min Cut :- Given a graph G(V, E), min cut partitions the vertices into two set L and R, with the minimum number of edges between L and R.

$R = V - L$

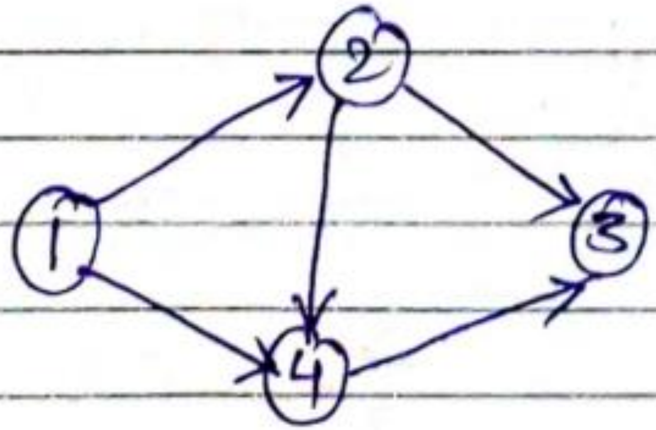
Here, we use Karger's algorithm ←

```
for i = 1 to m
    repeat until 2 nodes remaining
        Take a random edge (u, v) ∈ E and
        form a contraction u'
    output the minimum cut
end.
```

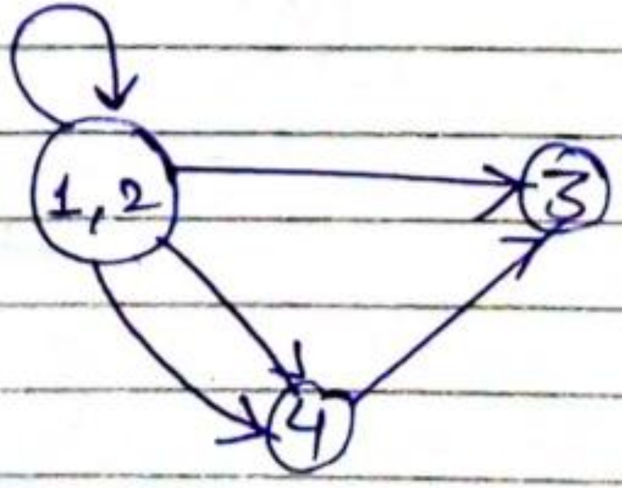
Randomized Algorithm

Edge contraction

Eg :-



1. Select {1, 2} and contract

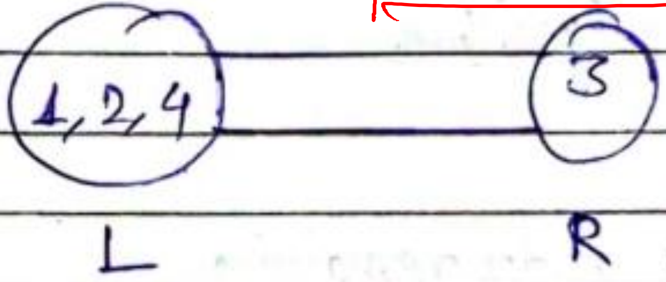


Self loop will be removed.

Randomized Algorithm

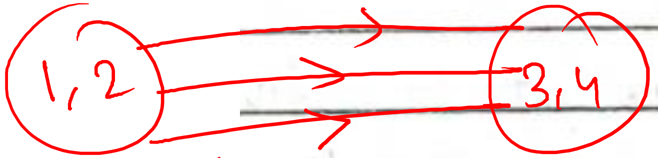
2. Select {2, 4} and contract

Removing the self loop



Between L and R there are 2 edges so Min-Cut = 2.

If we had contracted {3, 4} instead we would have got



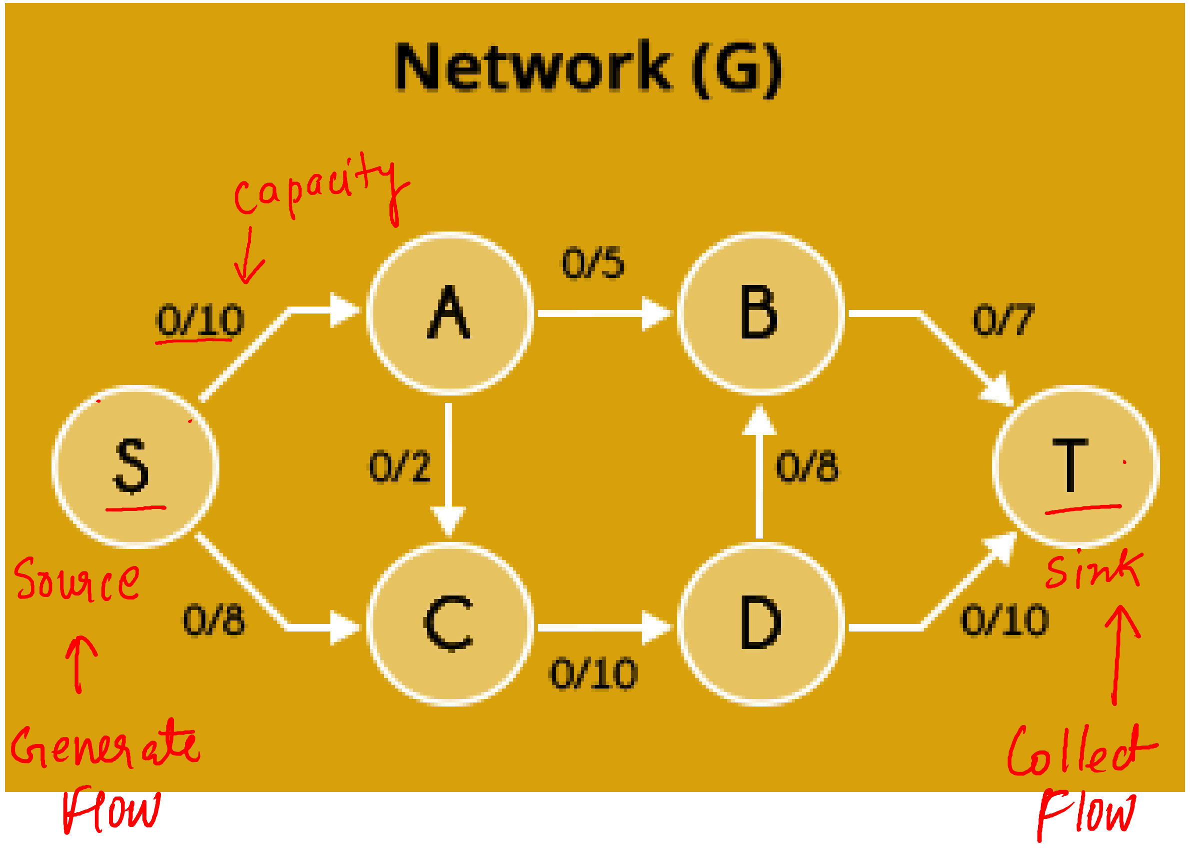
Here Min-cut = 3

So its a Monte-Carlo algo not ensuring optimal result always

Maximum Flow in Network

In graph theory, a **flow network** is defined as a directed graph involving a source (**S**) and a sink (**T**) and several other nodes connected with edges.

Each edge has an individual capacity which is the maximum limit of flow that edge could allow.



$0/10 \rightarrow f/C$

flow value / capacity

Maximum Flow in Network

Flow in the network should follow the following conditions:

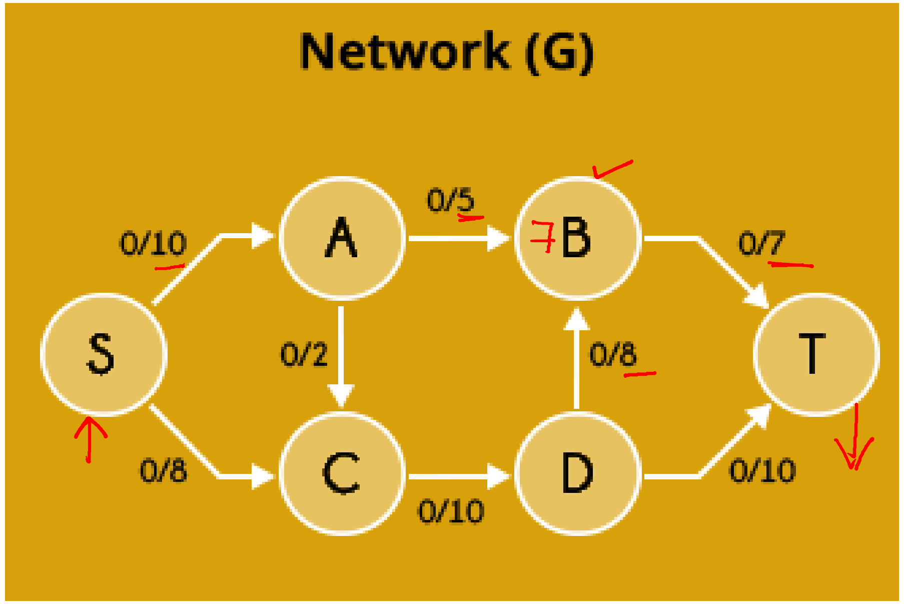
- For any non-source and non-sink node, the input flow is equal to output flow.

- For any edge **(E_i)** in the network,

$0 \le$ flow (E_i) $\le$ Capacity (E_i).

- Total flow out of the source node is equal to total flow in to the sink node.

- Net flow in the edges follows skew symmetry **i.e. F(u,v)= – F(v,u) where F(u,v)** is flow from node u to node v. This leads to a conclusion where you have to sum up all the flows between two nodes (either directions) to find net flow between the nodes initially.



f / c

$S \xrightarrow{f=5} A$

flow(S,A) = 5

flow(A,S) = -5

Maximum Flow in Network

Maximum Flow:

It is defined as the maximum amount of flow that the network would allow to flow from source to sink.

Multiple algorithms exist in solving the maximum flow problem.

Two major algorithms to solve these kind of problems are :

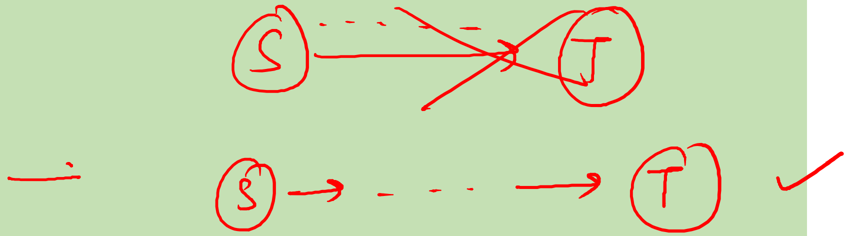
- Ford-Fulkerson Algorithm
- Dinic's Algorithm.

Ford-Fulkerson Algorithm: Maximum Flow

It was developed by **L. R. Ford Jr.** and **D. R. Fulkerson** in **1956**.

A pseudocode for the algorithm is given below, Inputs required are network graph **G** , source node **S** and sink node **T**.

```
function: FordFulkerson(Graph G,Node S,Node T):
{
    Initialize flow in all edges to 0
    while (there exists an augmenting path(P) between S and T in residual network graph):
        {
         Augment flow between S to T along the path P
         Update residual network graph
        }
    return
}
```



Augmenting Path

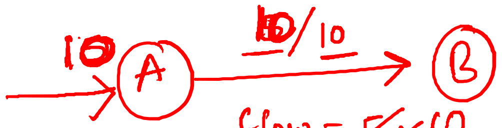
An augmenting path is a simple path from source to sink which do not include any cycles and that pass only through positive weighted edges.

A residual network graph indicates how much more flow is allowed in each edge in the network graph.

If there are no augmenting paths possible from **S** to **T**, then the flow is maximum.

The result i.e. the maximum flow will be the total flow out of source node (**S**) which is also equal to total flow in to the sink node (**T**).

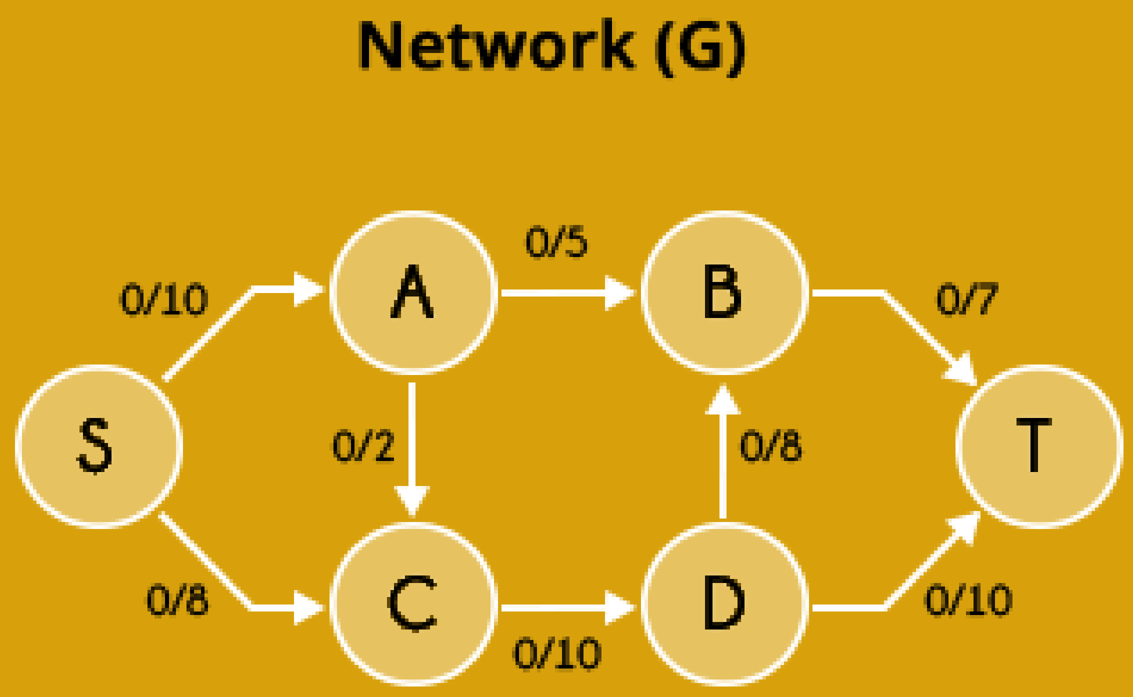
residual n/w graph / residual capacity graph.

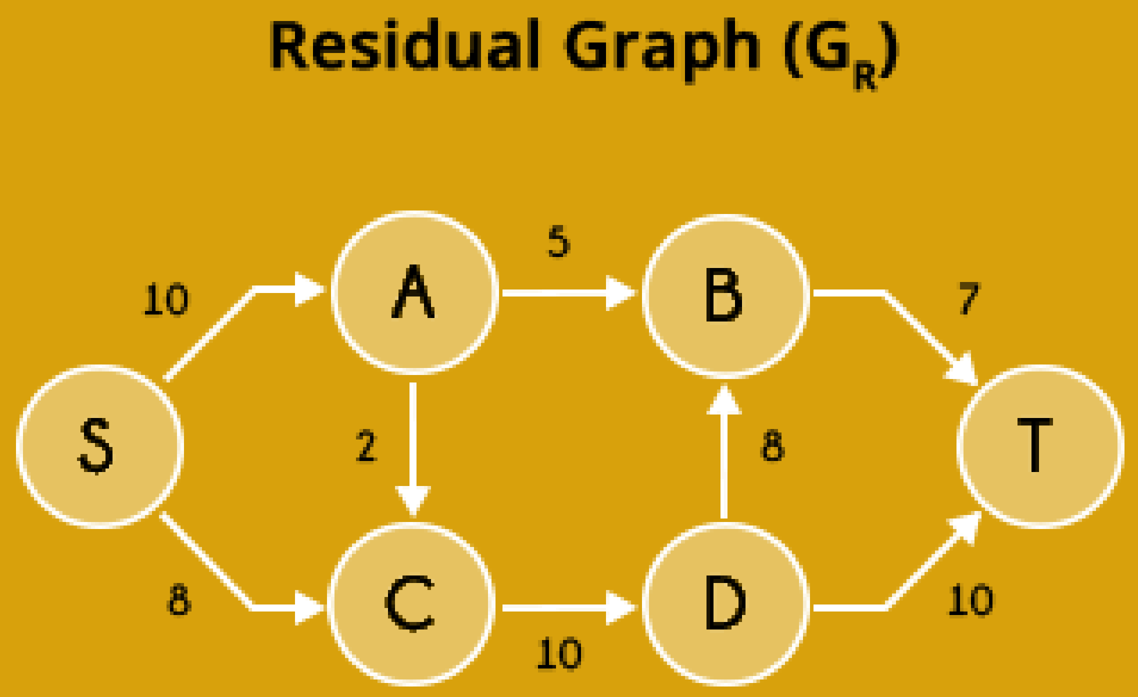


flow = ~~5~~ v 10

RC = 10-5 = 5   10 - 10 = 0

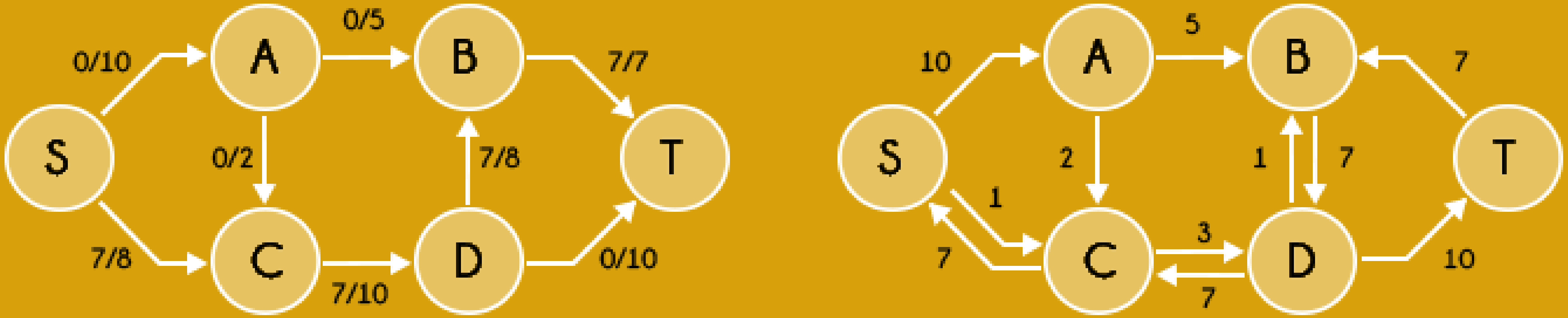
Exercise: Computing Maximum Flow

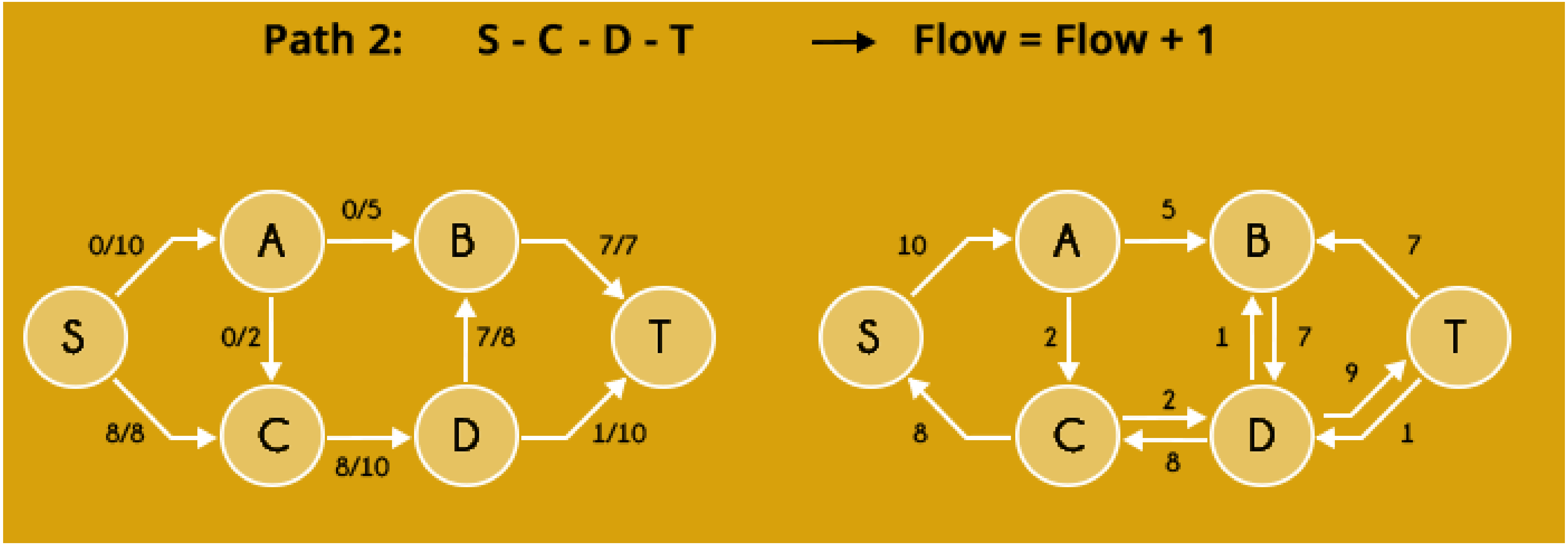


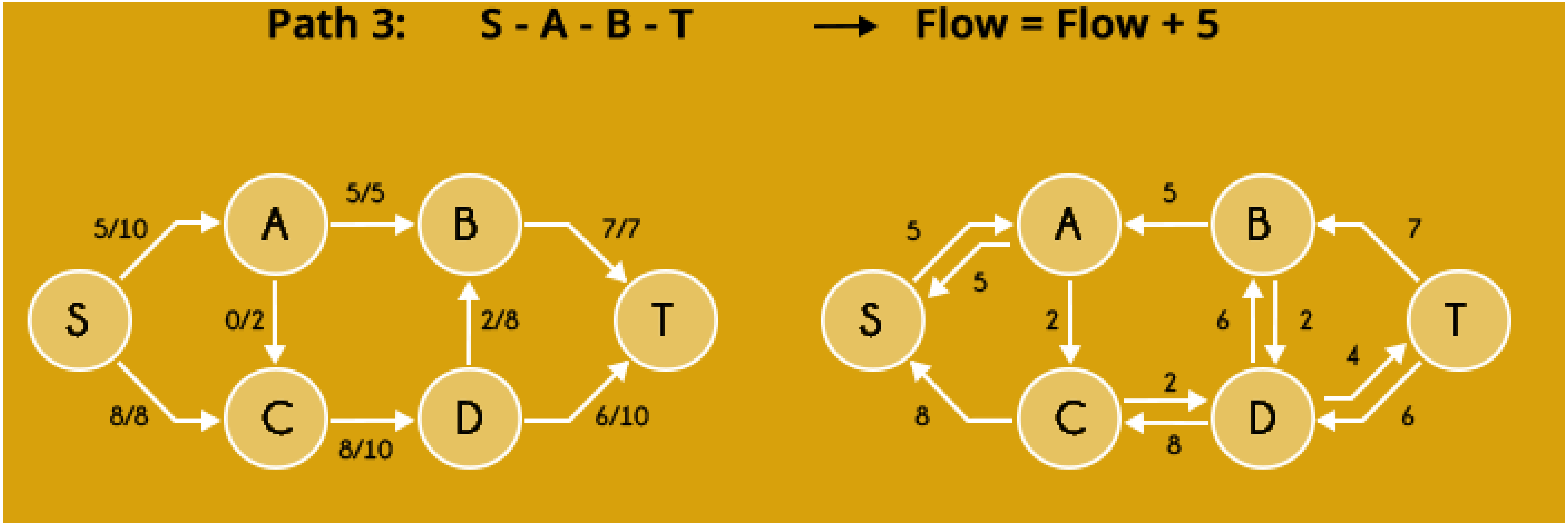


Flow = 0

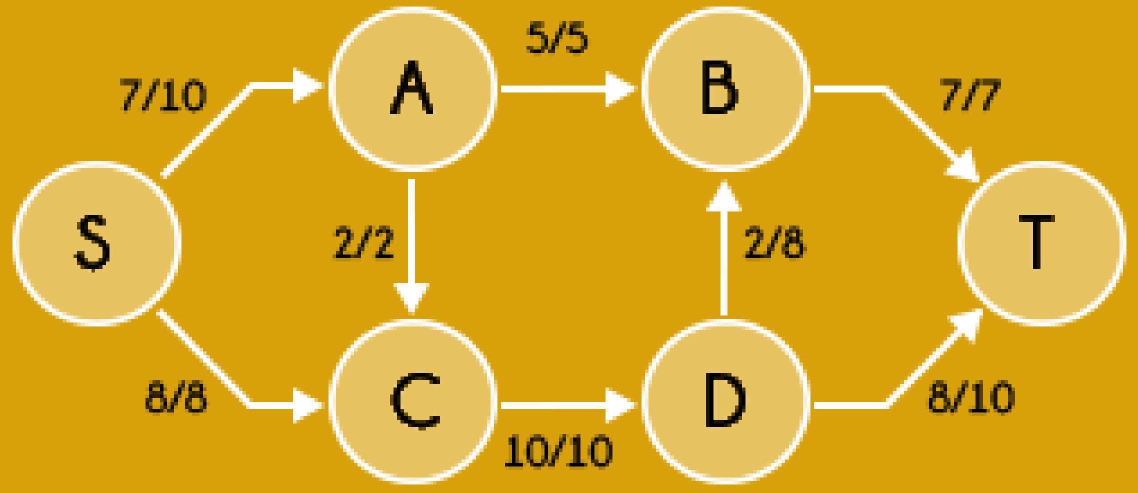
Path 1: S - C - D - B - T → Flow = Flow + 7

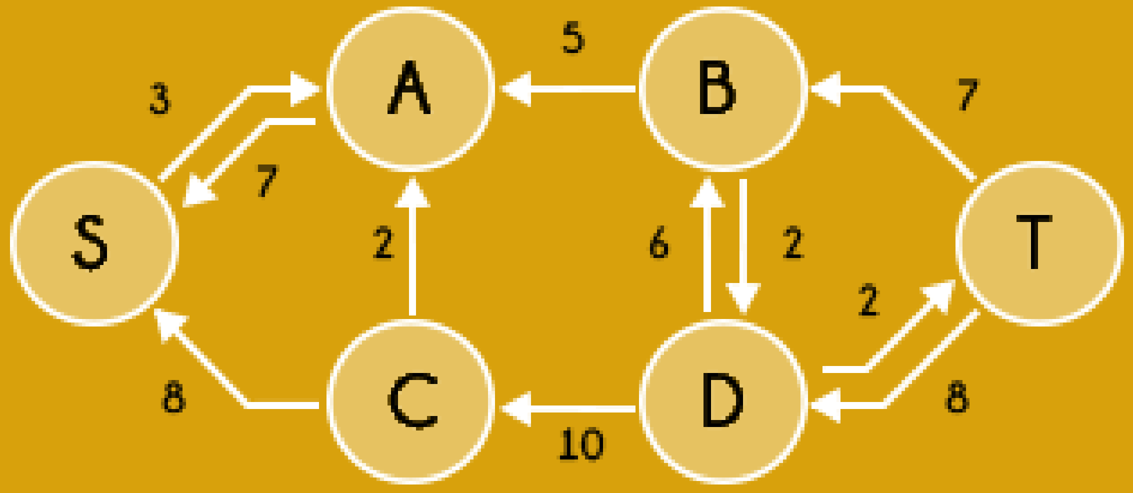



Path 2: S - C - D - T → Flow = Flow + 1
0/10
A
0/5
B
7/7
S
0/2
7/8
T
8/8
C
D
1/10
8/10
10
A
5
B
7
S
2
1
7
T
9
2
8
C
D
1
8


Path 3: S - A - B - T → Flow = Flow + 5
S
A
B
C
D
T
5/10
5/5
7/7
0/2
2/8
8/8
8/10
6/10
S
A
B
C
D
T
5
5
5
7
2
6
2
4
2
8
8
6

Path 4: S - A - C - D - T ⟶ Flow = Flow + 2



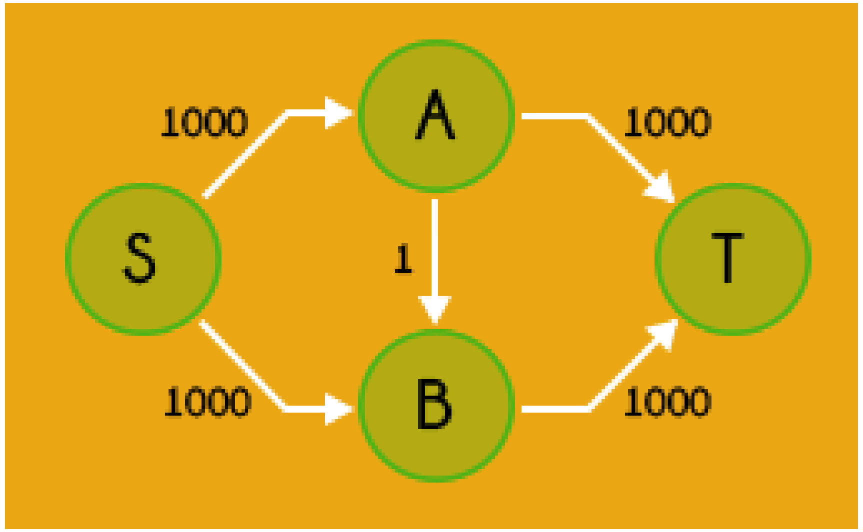


No More Paths Left

Max Flow = 15

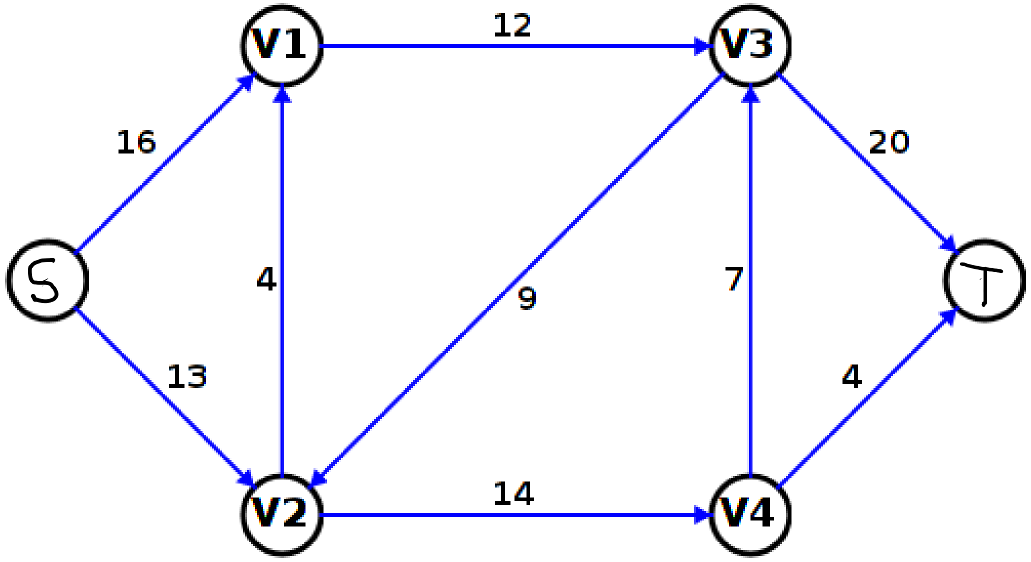
Implementation

- An augmenting path in residual graph can be found using DFS or BFS.
- Updating residual graph includes following steps:
  - For every edge in the augmenting path, a value of minimum capacity in the path is subtracted from all the edges of that path.
  - An edge of equal amount is added to edges in reverse direction for every successive nodes in the augmenting path.



The complexity of Ford-Fulkerson algorithm cannot be accurately computed as it all depends on the path from source to sink. For example, considering the network shown below, if each time, the path chosen are **S−A−B−T** and **S−B−A−T** alternatively, then it can take a very long time. Instead, if path chosen are only **S−A−T** and **S−B−T**, would also generate the maximum flow.

Exercise: Find Maximum Flow in given Network



Multi Commodity Flow Problem

The multi-commodity flow problem is a network flow problem with multiple commodities (flow demands) between different source and sink nodes.

Given a flow network $G(V,E)$, where edge $(u,v) \in E$ has capacity $c(u,v)$. There are k commodities $K_1, K_2, \ldots K_k$ defined by $K_i = (s_i, t_i, d_i)$ where s_i is source, t_i is sink of commodity i and demand d_i, variable $f_i(u,v)$ defines the fraction of flow i along edge (u,v), where $f_i(u,v) \in [0,1]$.

In this problem, we have to find an assignment of all flow variables which satisfies the following four constraints

1) Link Capacity :- The sum of all flows routed over a link does not exceed its capacity

$$\forall (u,v) \in E: \sum_{i=1}^{k} f_i(u,v) \cdot d_i \leq c(u,v)$$

Multi Commodity Flow Problem

(2) Flow conservation on transit nodes :- The amount of a flow entering an intermediate node u is the same that exits the node.

$$\sum_{w \in V} f_i(u,w) - \sum_{w \in V} f_i(w,u) = 0 \quad \text{when } u \neq s_i, t_i$$

(3) Flow conservation at the source :- A flow must exit its source node completely

$$\sum_{w \in V} f_i(s_i,w) - \sum_{w \in V} f_i(w,s_i) = 1$$

(4) Flow conservation at the destination :- A flow must enter its sink node completely

$$\sum_{w \in V} f_i(w,t_i) - \sum_{w \in V} f_i(t_i,w) = 1$$

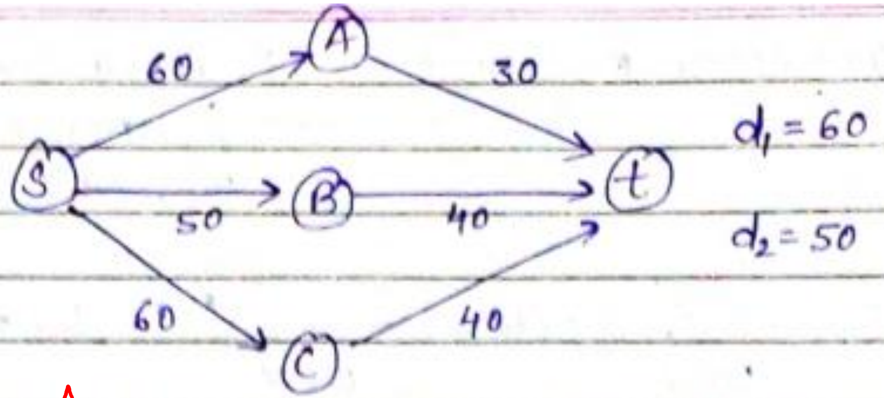
Multi Commodity Flow Problem

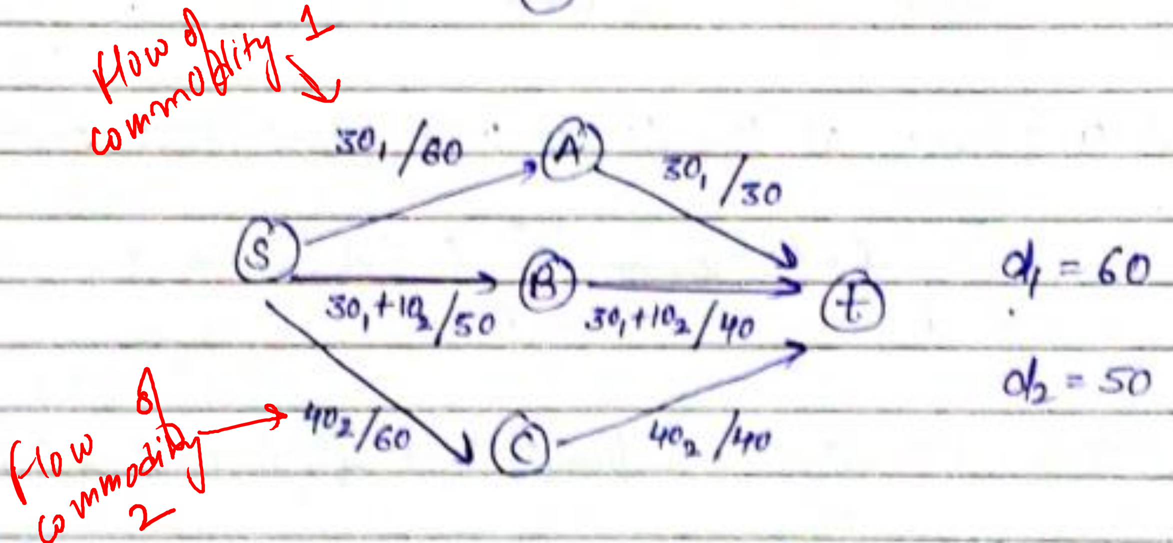
Optimization Problem :- Maximum multi-commodity flow problem

The demand of each commodity is not fixed and the total throughput is maximized by maximizing the sum of all demands $\sum_{i=1}^{K} d_i$

Multi Commodity Flow Problem

Example :-





demand d_1 for commodity $K=1$ and d_2 for Commodity $K=2$ is met completely through the network.

Flow Shop Scheduling

Jobs

Machine

Processing time		J_1	J_2	J_3	J_4	J_5
M_1	P_{1i}	9	3	5	8	10
M_2	P_{2j}	6	8	9	8	7

Flow Shop Scheduling

Flow shop scheduling is a scheduling problem in which flow control shall enable an appropriate sequencing for each job and for processing on a set of machines to minimize the completion time (makespan).

Computation of Minimum Makespan using Johnson's algorithm

Johnson's Algorithm

Step 1 :- Form two groups of jobs S_1 and S_2 such that

S_1 is set of job J_i | $P_{1i} \le P_{2i}$, means processing time of Job i on machine M_1 is less than or equal to on M_2, and

S_2 is set of job J_i | $P_{1i} > P_{2i}$, means processing time of job i on Machine M_1 is greater than on M_2.

Flow Shop Scheduling

Step 2 :- Sort S_1 according to Shortest Processing time on M_1 and

Sort S_2 according to Longest Processing time on M_2

Step 3 :- Schedule S_2 after S_1 i.e. → (S_1, S_2)

So, here applying the Johnson's algo on given data, we form

$S_1 = J_2, J_3, J_4$ and $S_2 = J_1, J_5$

Sorting S_1 acc. to SPT on M_1 and S_2 acc. to LPT on M_2, we get

$S_1 = J_2, J_3, J_4$ and $S_2 = J_5, J_1$

Final sequence $= (S_1, S_2) = (J_2, J_3, J_4, J_5, J_1)$

Flow Shop Scheduling

Processing time		J_1	J_2	J_3	J_4	J_5
M_1	P_{1i}	9	3	5	8	10
M_2	P_{2i}	6	8	9	8	7

Makespan ⇒

	M_1	M_2
J_2	3	3+8 = 11
J_3	3+5=8	11+9=20
J_4	8+8=16	20+8=28
J_5	16+10=26	28+7=35
J_1	26+9=35	35+6=41

⇒ Makespan = 41

← This value indicate the minimum makespan