FORD FULKERSON ALGORITHM:

(for maximum flow prob).

PROBLEM:

Given a graph which represents a flow n/w where every edge has a capacity. Also, given two vertices source 's' and sink 't' in the graph. Find out the maximum possible flow from 's' to 't' with following constraints.

a) Flow on an edge does not exceed given capacity of the edge.

b) In-flow is equal to out-flow for every vertex except 's' and 't'.

ALGORITHM:

```
FORD FULKERSON ALGORITHM (G, s, t)
    initialize flow f to 0
    while there exists an augumenting path p in the
    residual n/w Gf:
        augument flow f along p.
        update residual network
    return f.
```

TERMINOLOGY:

Residual graph:

It's a graph which indicates additional possible flow. If there is such path from source to sink, then there is possibility to add flow.

Residual Capacity:

It's mathematically defined as:

residual capacity = original capacity − current flow

Minimum Cut:

* Also known as bottleneck capacity.
* It is the minimum capacity of any edge on the graph

Augumenting path:

* A path from 's' to 't', no cycle & only +ve weights

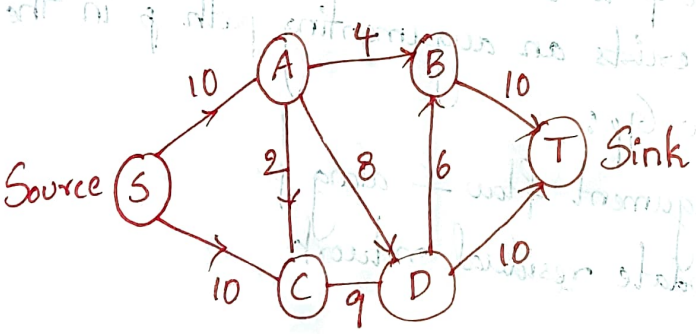
Two ways:

1. Non-full forward edges
2. Non-empty backward edges

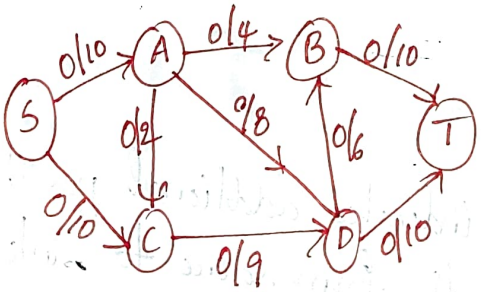
Time Complexity

$O(E|f^*|)$

EXAMPLE:

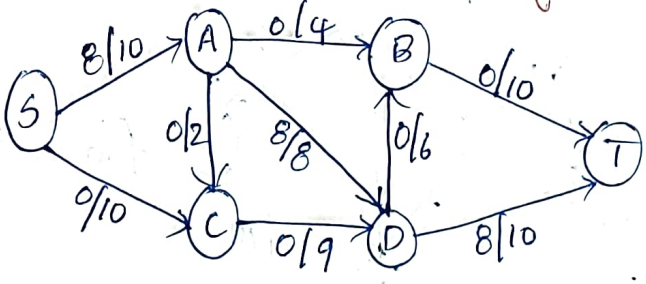


Step 1: Initialize flow as zero at all edges.



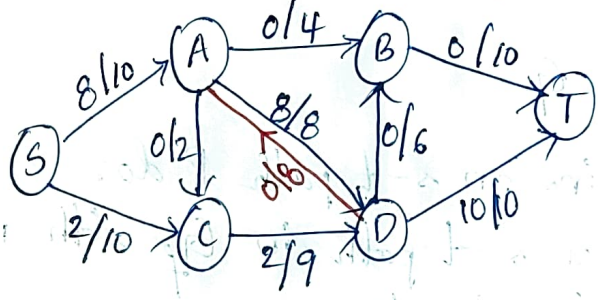
The process is to select a path, calculate a bottleneck, assign it to each edge of the path, calculate residual capacity & iterate through entire process till all edges are exhausted & maximum flow is attained

PATH1 → S → A → D → T, Bottleneck-8, Assign '8' along path.



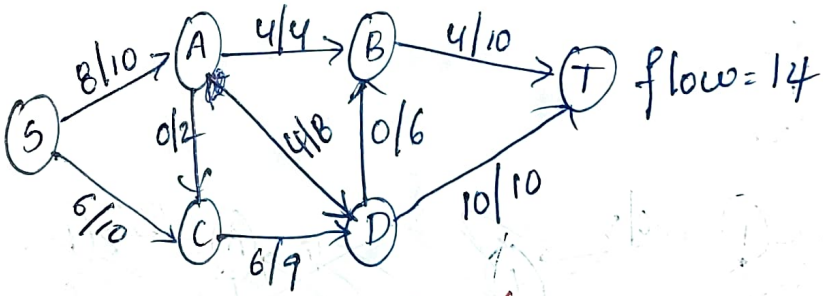
Path	Bottleneck
S → A → D → T	8
S → C → D → T	2
S → C → D → A → B → T	4
S → C → D → B → T	3
S → A → D → B → T	2

Path → S → C → D → T, Bottleneck-2, Assign '2' along path.

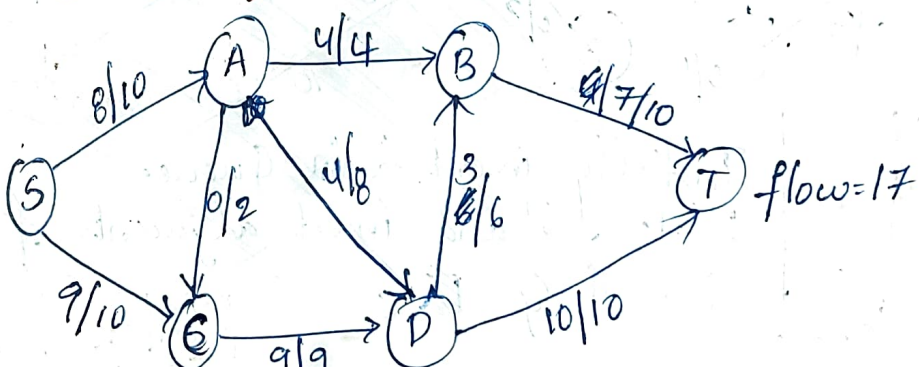


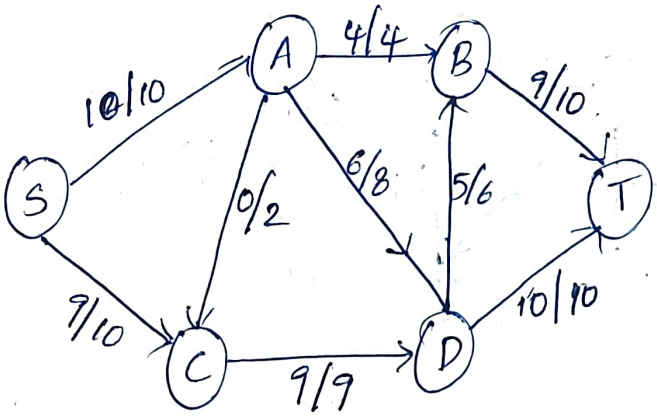
Path → S → C → D → A → B → T, [DA → Non-empty backward edge ⇒ Taking reverse edge]

Bottleneck → 4



Path → S → C → D → B → T, Bottleneck → 3.





EDMOND- KARP ALGORITHM:

An implementation of Ford Fulkerson that specifically mentions the use of BFS to find maximum flow.

ALGORITHM:

$f \leftarrow 0$; $G_f \leftarrow G$

while G_f contains an $s-t$ path P do:

Let P be an $s-t$ path in G_f with min no. of edges

Augument f using P.

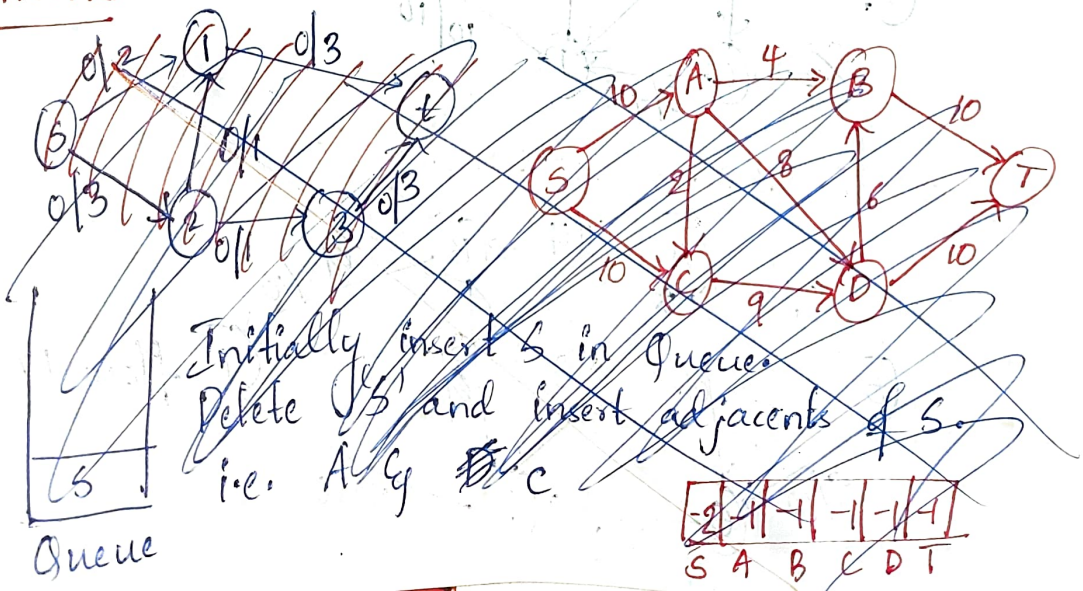
Update G_f

end while

return f.

PROBLEM:

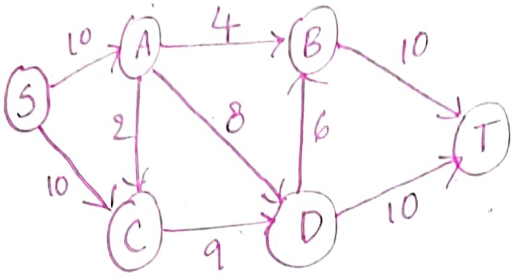
PATH:



IMPLEMENTATION:

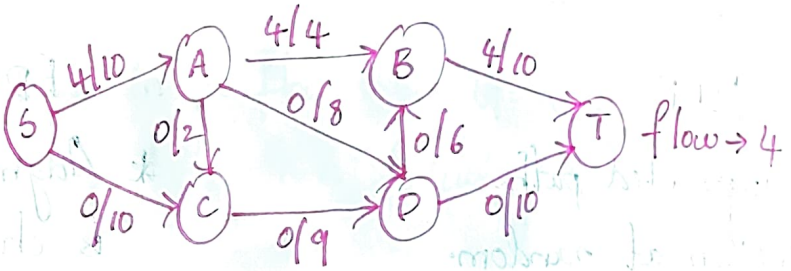
① Find the shortest path.
② Increment the flow monotonically.

} DONE USING BFS.

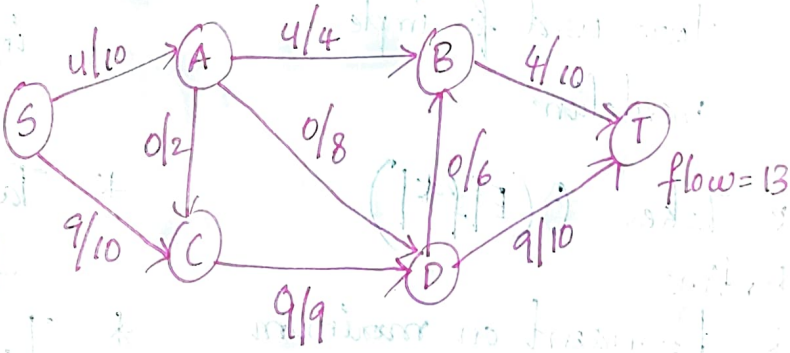


In the above graph, we have two shortest paths:
$S \to A \to B \to T$ & $S \to C \to D \to T$.

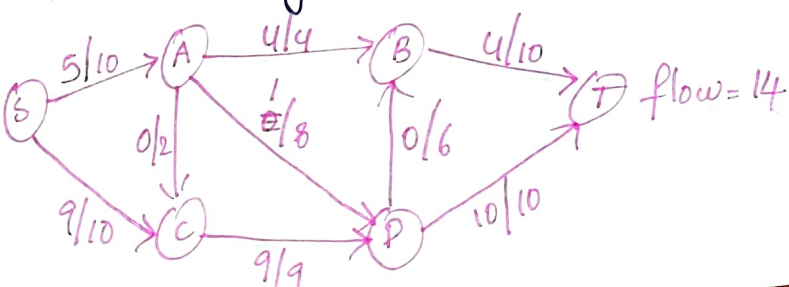
We consider $S \to A \to B \to T$. Bottleneck → 4. Increment flow by 4.



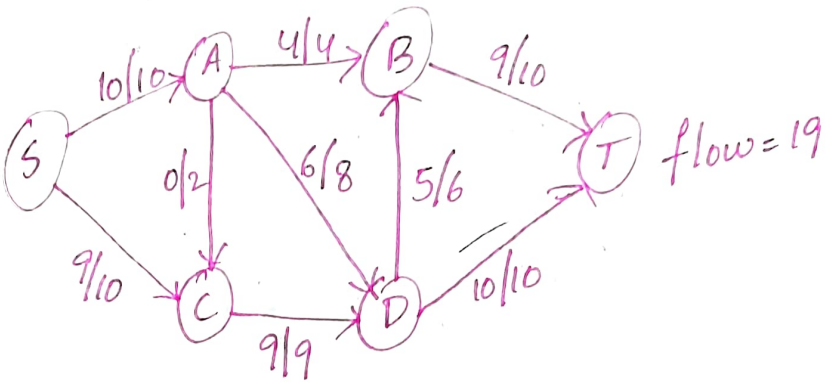
Now, we consider $S \to C \to D \to T$. Bottleneck → 9. Increment flow by 9.



The next shortest path is: $S \to A \to D \to T$. Bottleneck is 1. So increment flow by 1.



The next shortest path is $S \to A \to D \to B \to T$.
Bottleneck is 5. Increment flow by 5.



As the paths are now exhausted, we attain maximum flow = 19

MAIN DIFFERENCES BETWEEN FORD & EDMOND

FORD	EDMOND
* Augmented paths choosen at random.	* Augmented path is choosen in an ordered manner.
* No special datastructure used for implementation.	* Uses BFS for implementation
* Takes $O(E\|f^*\|)$ Runtime	* Takes $O(VE^2)$ time
* Dependent on maximum flow of network, f^*.	* Independent of maximum flow of nfw, f^*

Describe Max-Flow Min-Cut Theorem And It's Applications:

Max-Flow Min-Cut:

Max flow Min Cut theorem states that, in a flow n/w, the max amount of flow passing through the source to the sink is equal to total weight of edges in a minimum cut (smallest total weight of the edges which if removed would disconnect source from the sink)

Max-flow Min-Cut Conditions:

If f is a flow in a flow n/w $G_1 = (V, E)$ with source 's' and sink 't', then following conditions are equivalent:

1. f is a maximum flow in G_1.
2. The residual n/w G_f contains no augumenting paths.
3. If $| = c(S,T)$ for some cut (S,T) of G_1.

Min-Cut:

Set of edges whose removal divides n/w into 2-halves X & Y such that

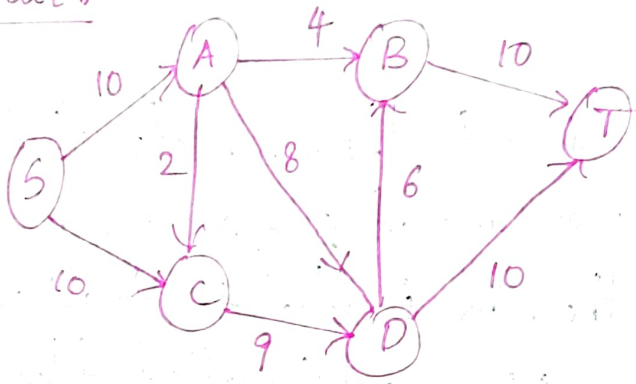
's' (source) $\in$ X

't' (sink) $\in$ Y

Example Illustration:

(For Max-flow refer to ford fulkerson)

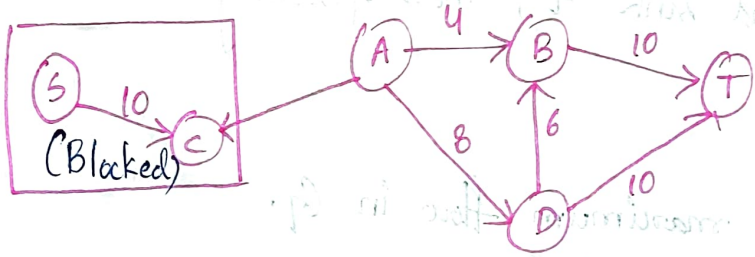
For Min-$\not{s}$ cut:



When disconnecting the graph, it should be ensured that sum of weights of removed edges = max flow

Max-flow = 19.

If we remove $S \to A$ and $C \to D$



$S \to A = 10$

$C \to D = 9$

Min-cut weight $= 19 =$ max-flow.

APPLICATIONS:

1. Nlw reliability, availability & connectivity
2. Matching in graphs (bipartite matching)
3. airlines use this to decide when to allow planes to leave airports to maximize flow of flights.

Define Divide And Conquer Method And Mention Adv & Disadv.

* Many Algorithms are recursive in nature
* These Algorithms typically follow a divide-and-conquer approach

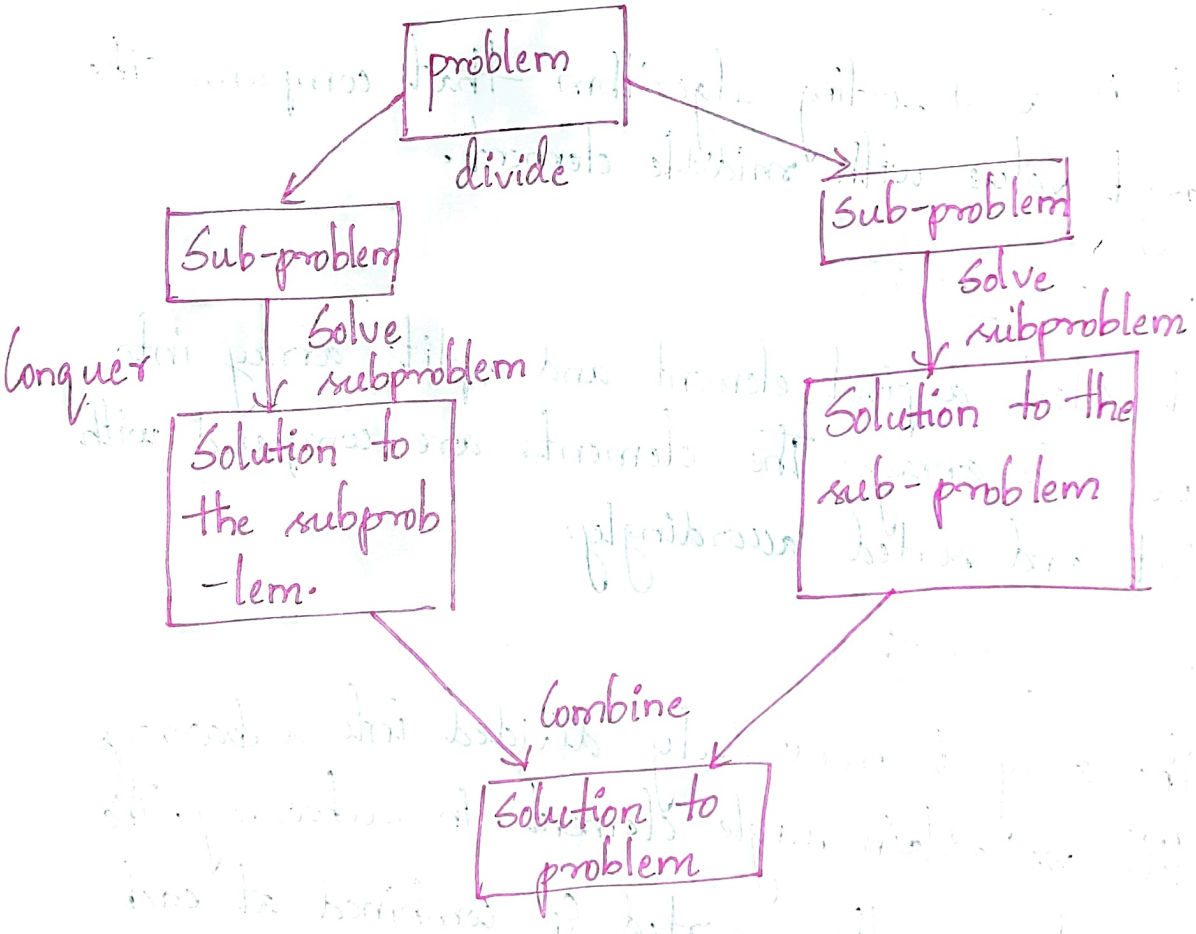
There are three steps at each level of recursion:

Divide & Conquer: Steps:

Divide: the problem is divided into subproblems

Conquer: The subproblems are solved recursively.

Combine: Merge the solutions to the subproblems into solution for original problem.



There are two fundamentals to Divide & Conquer Strategy:

1. Relational formula:

Formula that is developed and used to perform D & C strategy.

2. Stopping Condition:

Condition that specifies where the D & C strategy is to be stopped.

APPLICATIONS:

1. Binary Search:

It is a sorting algorithm that compares the target value with middle element.

2. Quick Sort:

It selects a pivot element and splits array into two sub-arrays. The elements are compared with pivot and sorted accordingly.

3. Merge Sort:

The array is recursively divided into subarrays till we attain single element in subarray. The elements are then sorted & combined at each level to attain sorted array.

4. Strassen's Algorithm:

An algorithm for matrix multiplication $\mathcal{G}$ is faster than traditional Algorithm.

Advantages:

1. Used to solve complicated problems such as towers of Hanoi.
2. Efficient use of cache memory
3. proficient than brute force technique
4. System can easily handle D & C by parallel processing.

Disadvantages:

1. Requires high memory management
2. Explicit stack is required resulting in space overuse
3. System may crash due to rigorous recursion.

EXPLAIN STRASSEN'S MATRIX MULTIPLICATION ALGORITHM

Given 2 square matrices A and B of size n x n each, find multiplication matrix.

NAIVE METHOD:

```
void multiply (int A[][N], int B[][N], int C[][N])
{
    for (int i = 0; i<N; i++)
    {
        int (j=0, j<N; j++)
```

```
{
    C[i][j] = 0;
    for (int k=0; k<N; k++)
    {
        C[i][j] += A[i][k] * B[k][j];
    }
}
}
}
```

Time Complexity is $O(N^3)$

DIVIDE AND CONQUER:

* Matrices are divided into sub-matrices of size $N/2$ & $N/2$.

$$\underbrace{\begin{bmatrix} a & b \\ c & d \end{bmatrix}}_{A} \times \underbrace{\begin{bmatrix} e & f \\ g & h \end{bmatrix}}_{B} = \underbrace{\begin{bmatrix} aet+bg & af+bh \\ cet+dg & cf+dh \end{bmatrix}}_{C}$$

$A, B, C \rightarrow N \times N$ matrices

$a, b, c, d \rightarrow$ submatrices of A of size $N/2 \times N/2$

$e, f, g, h \rightarrow$ " " B " " $N/2 \times N/2$

Time Complexity is $O(N^3)$

A better approach is to use strassen's matrix multiplication that reduces recursive calls from 8 to 7.

STRASSEN'S METHOD:

* Uses divide and conquer.
* Uses 7 calls and not 8.

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$M_1 = (A_{11} + A_{22})(B_{11} + B_{22})$$
$$M_2 = (A_{21} + A_{22}) B_{11}$$
$$M_3 = (A_{11})(B_{12} - B_{22})$$
$$M_4 = A_{22}(B_{21} - B_{11})$$
$$M_5 = (A_{11} + A_{12}) B_{22}$$
$$M_6 = (A_{21} - A_{11})(B_{11} + B_{22})$$
$$M_7 = (A_{12} - A_{22})(B_{21} + B_{22})$$
$$C_{11} = M_1 + M_4 - M_5 + M_7$$
$$C_{12} = M_3 + M_5$$
$$C_{21} = M_2 + M_4$$
$$C_{22} = M_1 - M_2 + M_3 + M_6$$

Time Complexity: $O(n^{\log 7})$