Maximum Flow

Unit 2

Flow Graph

- A common scenario is to use a graph to represent a "flow network" and use it to answer questions about material flows
- Flow is the rate that material moves through the network
- Each directed edge is a conduit for the material with some stated capacity
- Vertices are connection points but do not collect material
  - Flow into a vertex must equal the flow leaving the vertex, flow conservation.

Sample

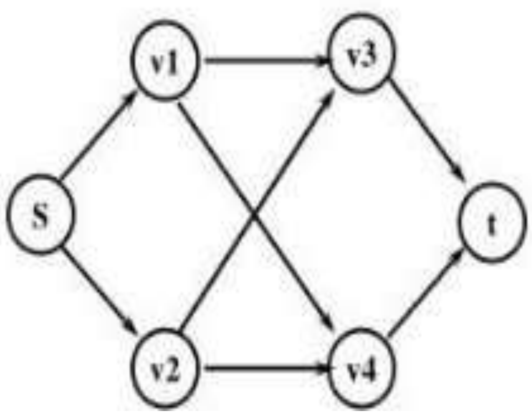
Network	Nodes	Arcs	Flow
communication	telephone exchanges, computers, satellites	cables, fiber optics, microwave relays	voice, video, packets
circuits	gates, registers, processors	wires	current
mechanical	joints	rods, beams, springs	heat, energy
hydraulic	reservoirs, pumping stations, lakes	pipelines	fluid, oil
financial	stocks, companies	transactions	money
transportation	airports, rail yards, street intersections	highways, railbeds, airway routes	freight, vehicles, passengers

Max-Flow Problem

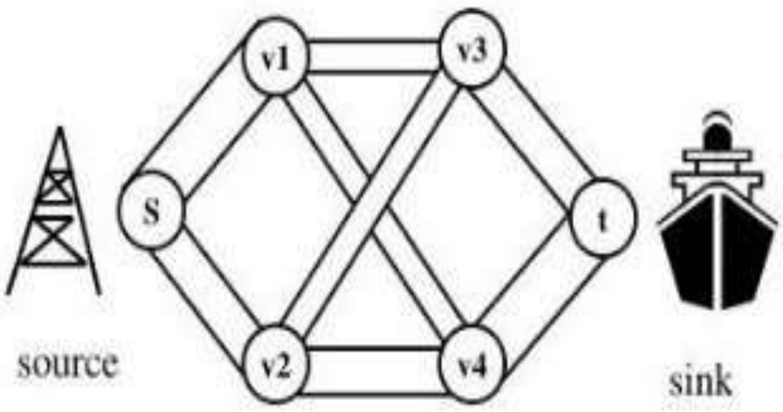
- Informal Definition:
- What is the greatest rate at which material can be shipped from the source to the sink without violating any capacity constraints?

Representation

Flow network: directed graph G=(V,E)

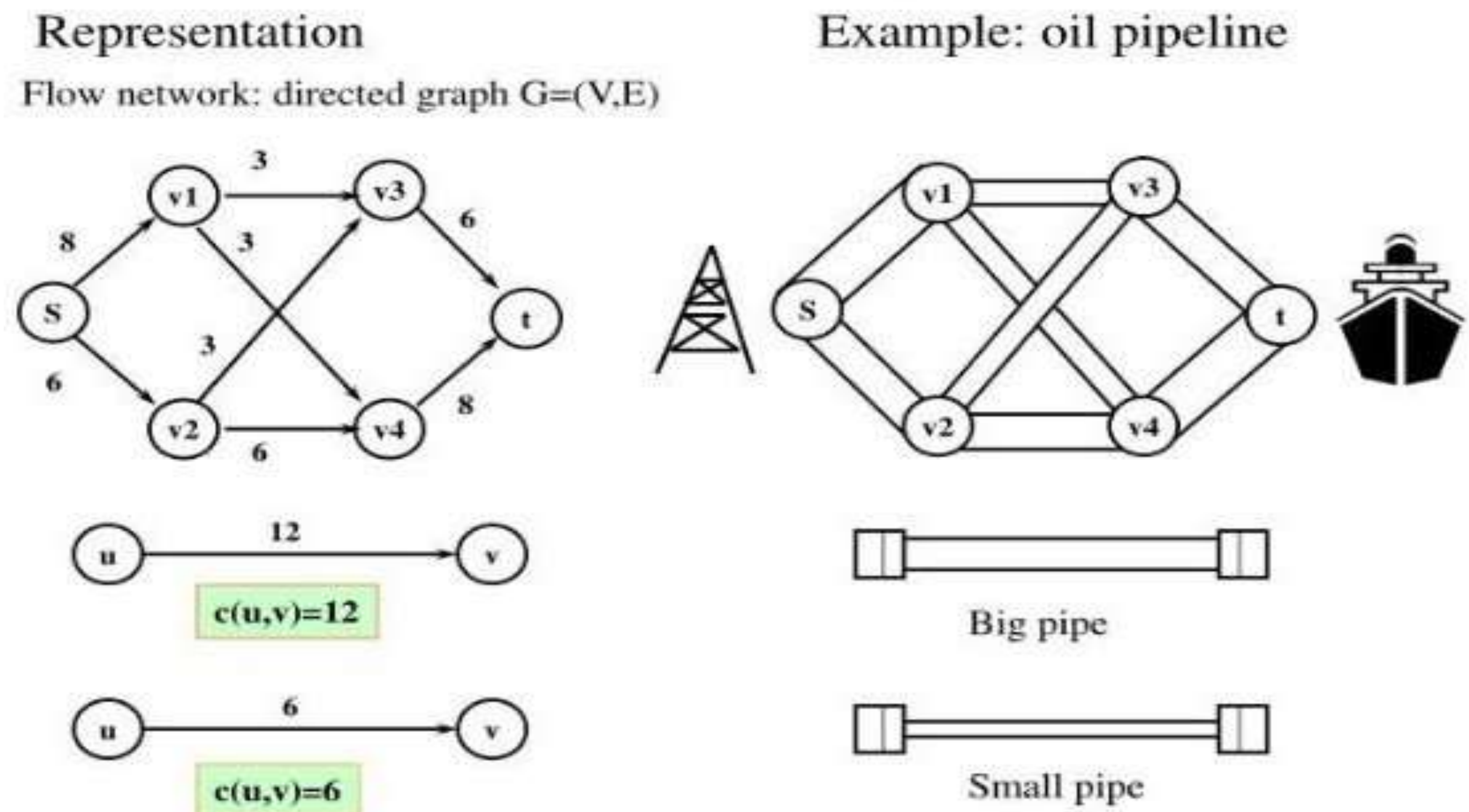


Example: oil pipeline



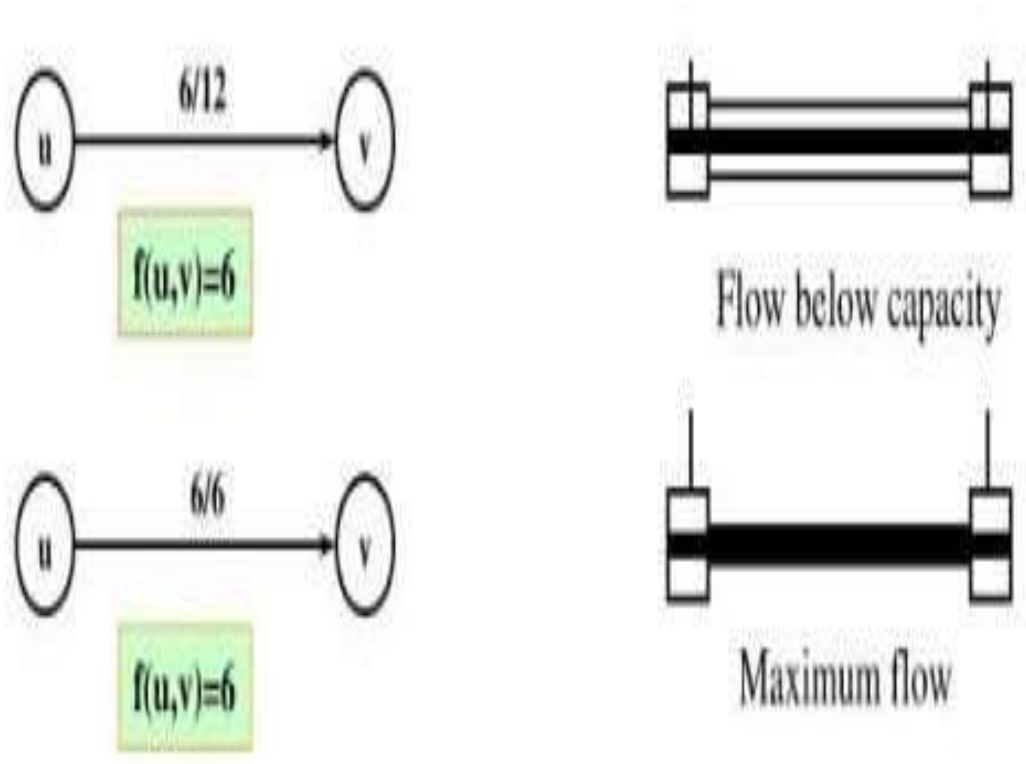
Introduction

- Capacity: Each edge has a certain capacity which can receive a certain



Introduction

- Flow: Is the actual units flowing on an edge. The flow running through an edge must be less than or equal to the capacity.

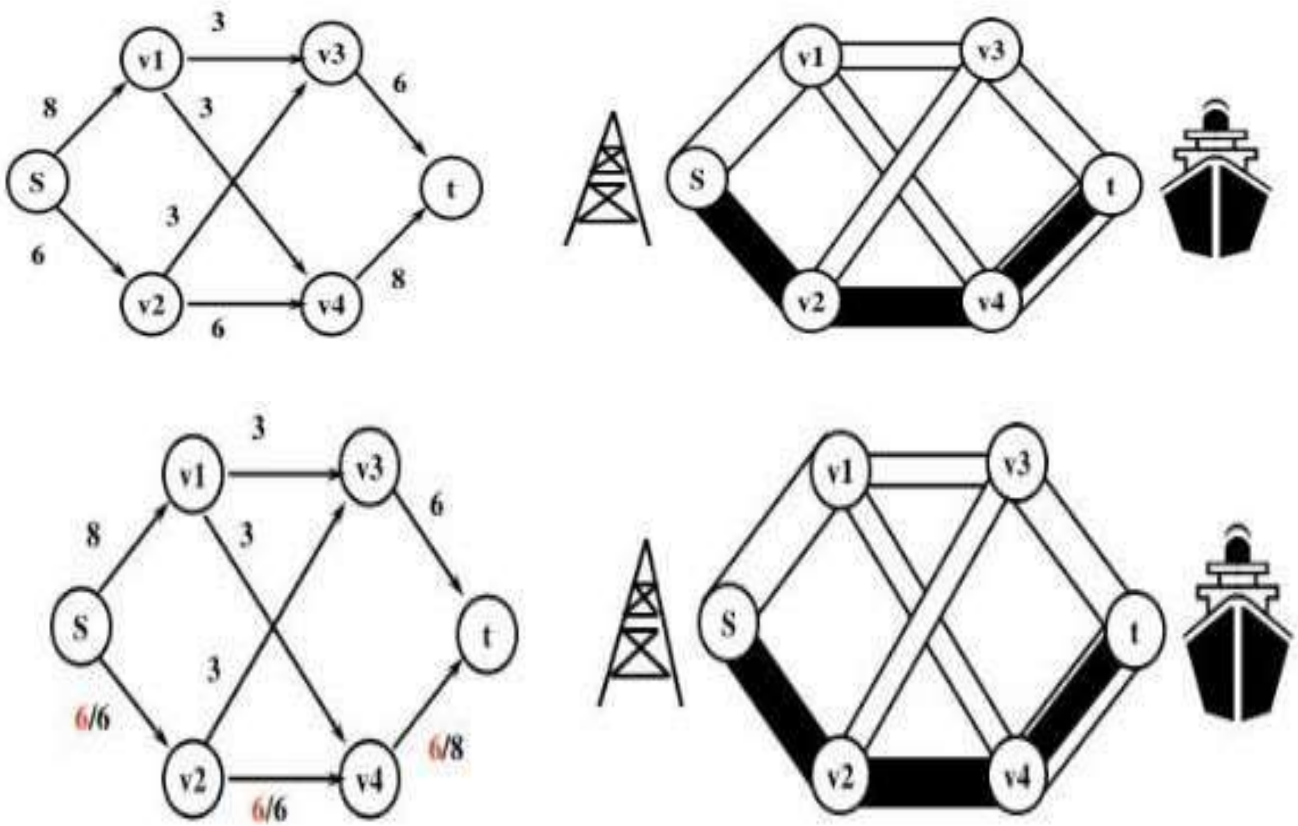


Flow

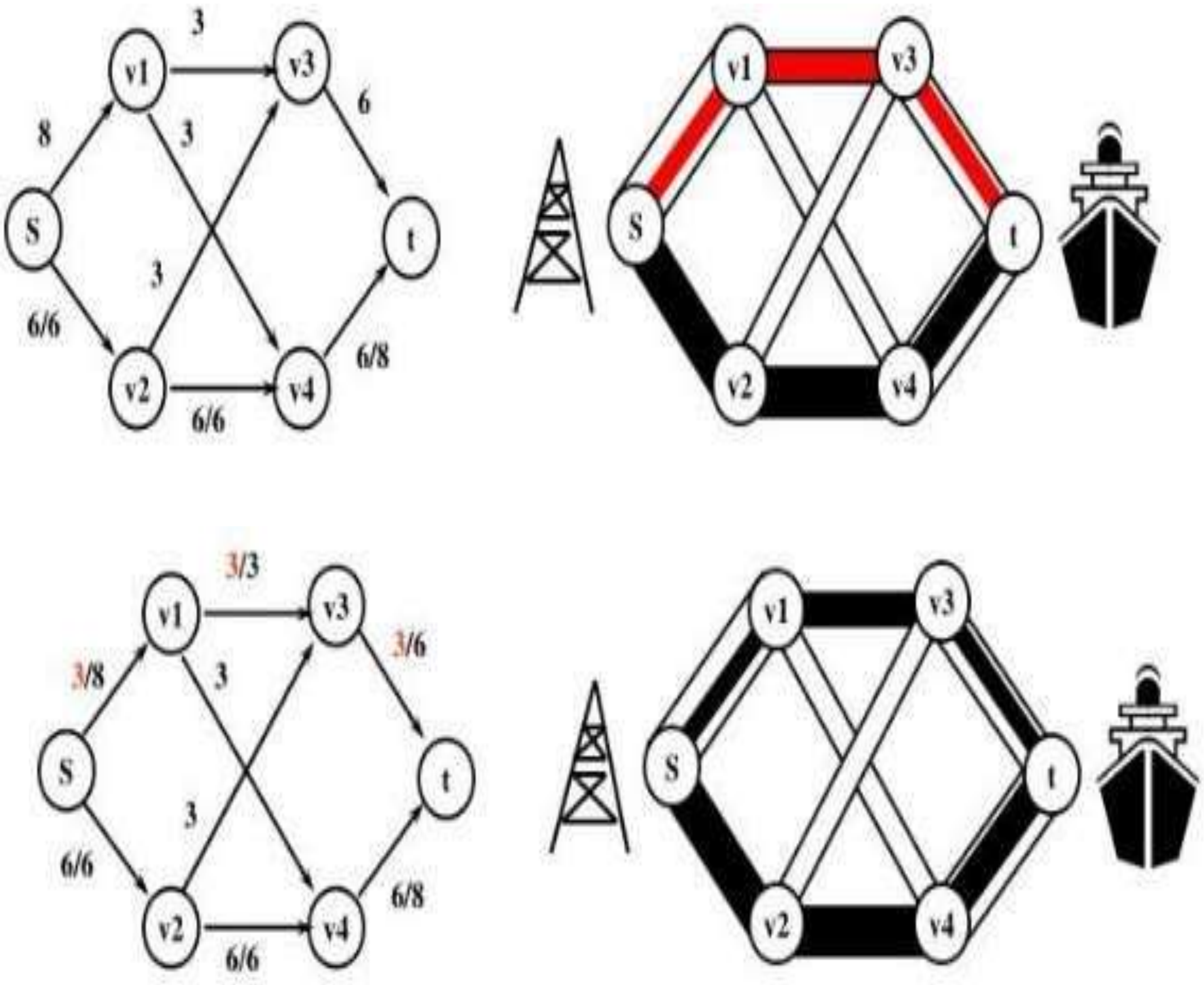
Representation

Example: oil pipeline

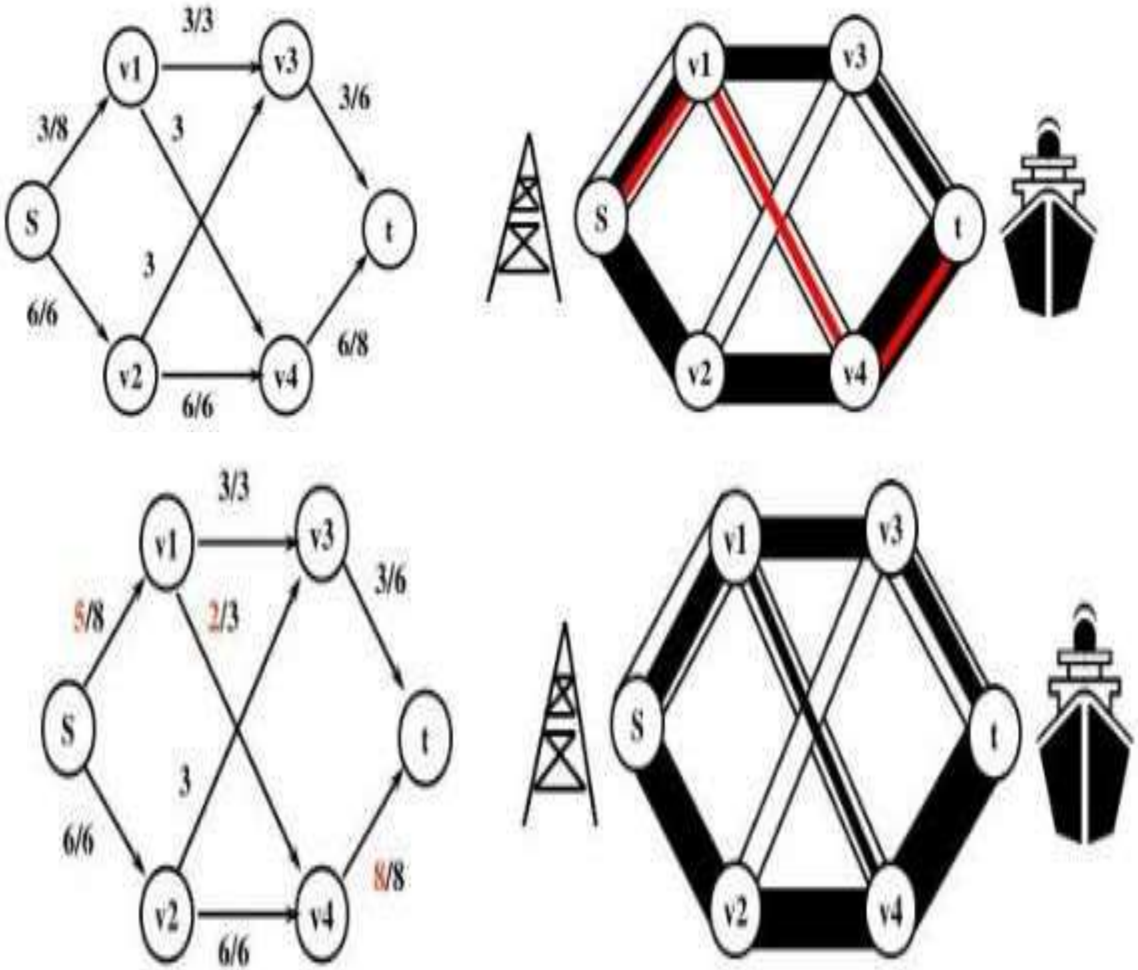
Flow network: directed graph G=(V,E)



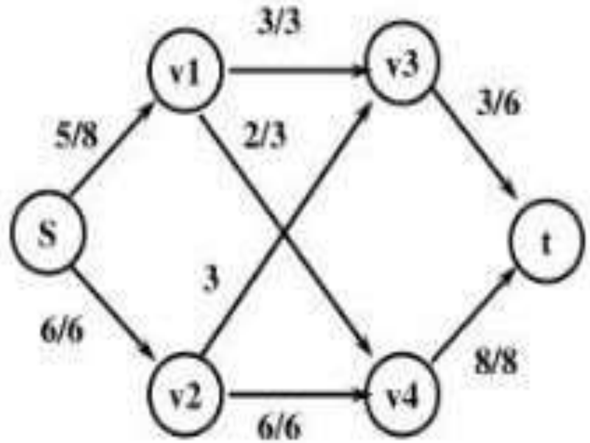
Flow



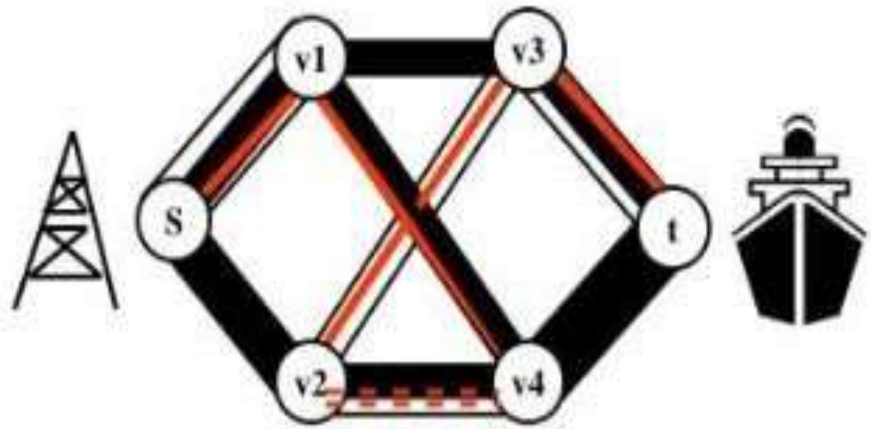
Flo


3/3
v1
v3
3/8
3
3/6
S
t
3
6/6
6/8
v2
v4
6/6
3/3
v1
v3
5/8
2/3
3/6
S
t
3
6/6
8/8
v2
v4
6/6
v1
v3
S
t
v2
v4
v1
v3
S
t
v2
v4

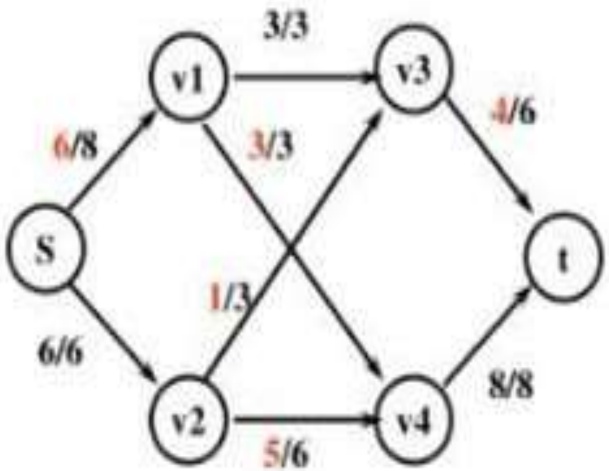
Flo



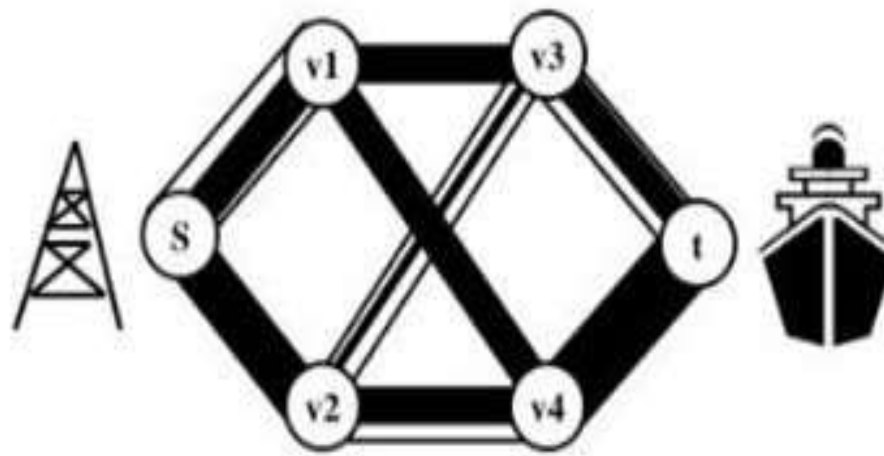
3/3
v1
v3
3/6
5/8
2/3
S
t
3
6/6
8/8
v2
v4
6/6



v1
v3
S
t
v2
v4



3/3
v1
v3
4/6
6/8
3/3
S
t
1/3
6/6
8/8
v2
v4
5/6



v1
v3
S
t
v2
v4

Max-Flow Problem

- With an infinite input source, how much "flow" can we push through the network given that each edge has a certain capacity?

Informal definition of the max-flow problem:

What is the greatest rate at which material can be shipped from the source to the sink without violating any capacity contraints?

Formal definition of the max-flow problem:

The max-flow problem is to find a valid flow for a given weighted directed graph G, that has the maximum value over all valid flows.

Flow Concepts

- Source vertex s
  - where material is produced
- Sink vertex t
  - where material is consumed
- For all other vertices – what goes in must go out
  - Flow conservation
- **Goal: determine maximum rate of material flow from source to sink**

Formal Max Flow Problem

– Graph $G=(V,E)$ – a **flow network**
  - Directed, each edge has **capacity** $c(u,v) \geq 0$
  - Two special vertices: ***source*** *s*, and ***sink*** *t*
  - For any other vertex *v*, there is a path $s \to \ldots \to v \to \ldots \to t$

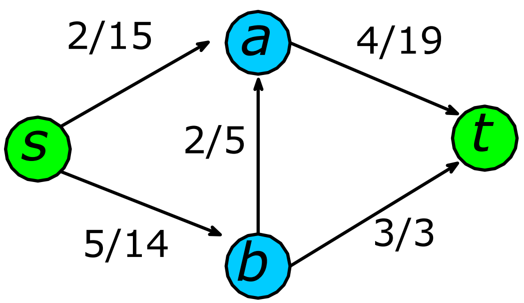
– **Flow** – a function $f : V \times V \to \boldsymbol{R}$

Capacity constraint: For all $u, v \in V$, we require $0 \leq f(u,v) \leq c(u,v)$.

Flow conservation: For all $u \in V - \{s,t\}$, we require

$$\sum_{v \in V} f(v,u) = \sum_{v \in V} f(u,v) .$$

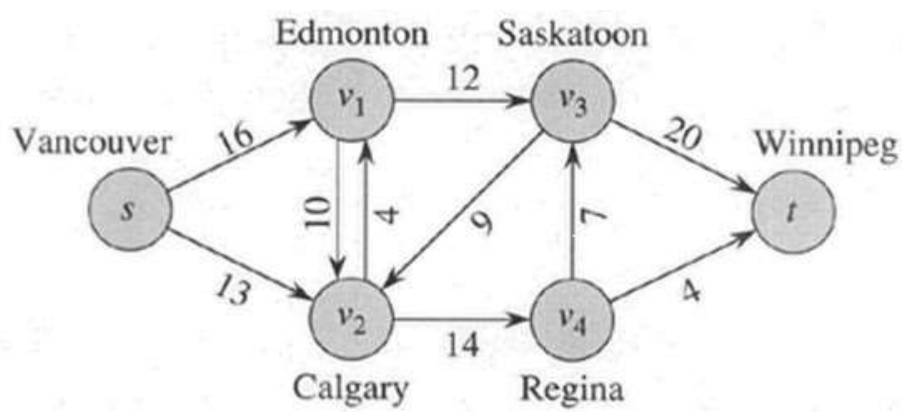
When $(u,v) \notin E$, there can be no flow from u to v, and $f(u,v) = 0$.



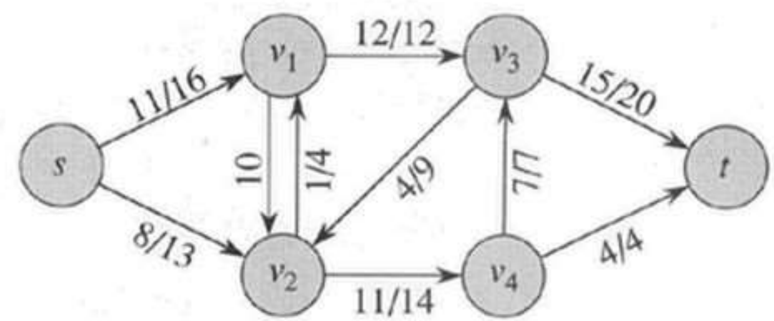
The capacity constraint simply says that the flow from one vertex to another must be nonnegative and must not exceed the given capacity. The flow-conservation property says that the total flow into a vertex other than the source or sink must equal the total flow out of that vertex—informally, "flow in equals flow out."

Max Flow

- We want to find a flow of maximum value from the source to the sink
  - Denoted by |f|



Lucky Puck Distribution Network



Max Flow, |f| = 19 Or is it? Best we can do?

The Ford-Fulkerson method:

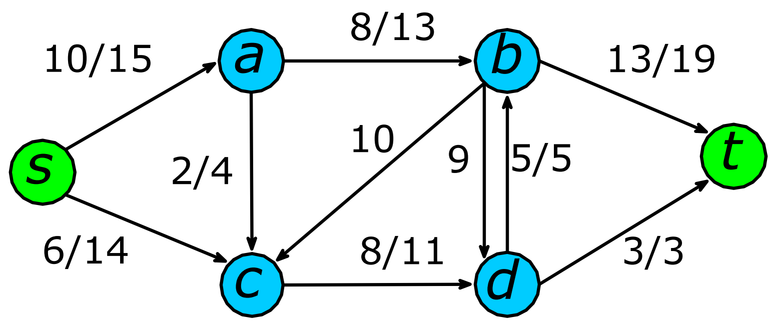
- This section presents the Ford-Fulkerson method for solving the maximum-flow problem. We call it a "method" rather than an "algorithm" because it encompasses several implementations with different running times.
- The Ford-Fulkerson method depends on three important ideas that transcend the method and are relevant to many flow algorithms and problems: residual networks, augmenting paths, and cuts.
- These ideas are essential to the important max-flow min-cut theorem, which characterizes the value of maximum flow in terms of cuts of the flow network.

Ford-Fulkerson method

- To find the maximum flow, the Ford-Fulkerson method repeatedly finds augmenting paths through the residual graph and augments the flow until no more augmenting paths can be found.

```
FORD-FULKERSON-METHOD(G, s, t)
initialize flow f to 0
while there exists an augmenting path p
    do augment flow f along p
return f
```

Augmenting Path?



Continue:

- FORD-FULKERSON-METHOD(G,s,t)
- initialize flow *f* to *0*
- while there exists an *augmenting* path *p*
- do *augment* flow *f* along *p*
- return *f*

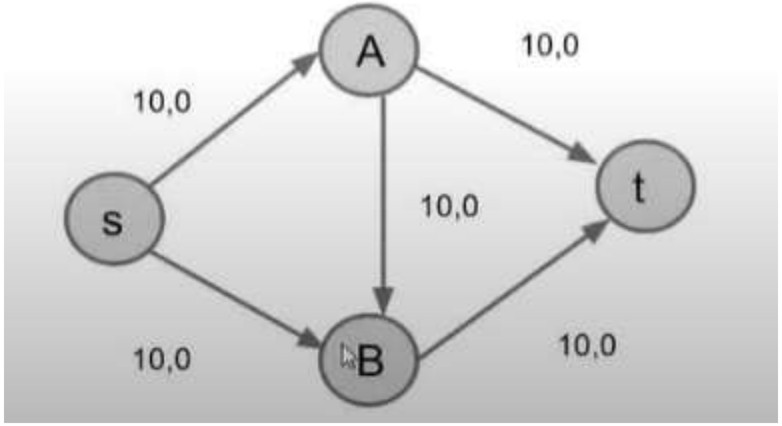
Residual networks:

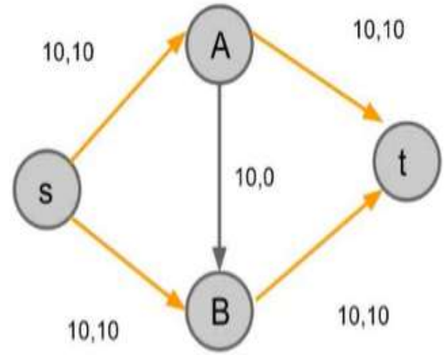
- Given a flow network and a flow, the **residual network** consists of edges that can admit more net flow.
- G=(V,E) --a flow network with source s and sink t
- f: a flow in G.
- The amount of additional net flow from u to v before exceeding the capacity c(u,v) is the residual capacity of (u,v), given by: $c_f(u,v)=c(u,v)-f(u,v)$

  in the other direction: $c_f(v, u)=c(v, u)+f(u, v)$.

Ford-Fulkerson method

- The Ford-Fulkerson : We start with no flow at all, that is, with every edge set equal to 0.
- Then we find what is called an augmenting path from the source to the sink. This is, a path from the source to the sink, that has excess capacity.
- We then figure out how much more we could pipe down that path and add this to the flow we are building.
- Contains several algorithms:
  - Residue networks
  - Augmenting paths
    - Find a path *p* from *s* to *t* (**augmenting path**), such that there is some value *x* > 0, and for each edge *(u,v)* in *p* we can add *x* units of flow
      - $f(u,v) + x \leq c(u,v)$

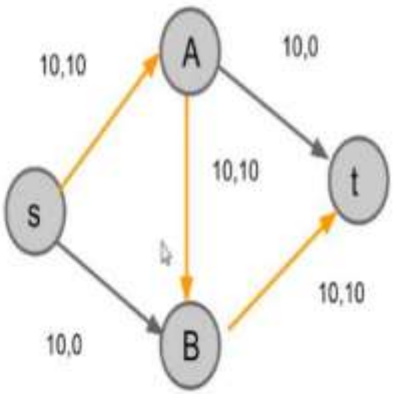




Augmenting Path(s):

1. s->A->t
2. s->B->t

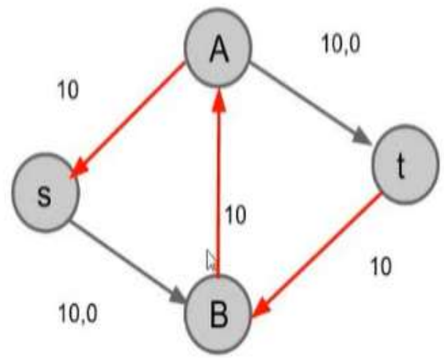
Total Flow = 20



Augmenting Path

1. s->A->B->t (10)

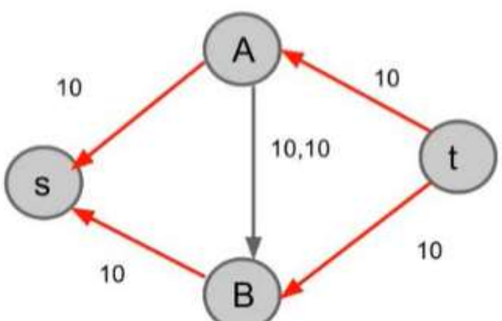
Total Flow = 10



Augmenting Path

1. s->A->B->t (10)

Total Flow = 10



Augmenting Path(s):

1. s->A->B->t (10)
2. s->B->A->t (10)

Total Flow = 20

- ***Front-pointing-edges*** having +ve residual capacity and ***back-pointing-edges*** having +ve flow can only be included in the path.
- In case of ***back-pointing-edge*** the bottle-neck value is subtracted from its flow value.
- The maximum flow a ***back-pointing-edge*** can handle is equal to its current flow.

Augmenting Path : A path from s to t on the Graph, where no vertices are repeated and no edge along this path has its capacity saturated.

Residual Graph

Given a flow network G, and a flow f on G, we define the residual graph $\boldsymbol{G_f}$ of G with respect to flow f as follows.

1. The node set of $\boldsymbol{G_f}$ is the same as that of G
2. Each edge $e = (u, v)$ of $\boldsymbol{G_f}$ is with a capacity of $c_e - f_e$
3. Each edge $e' = (v, u)$ of $\boldsymbol{G_f}$ is with a capacity of f_e

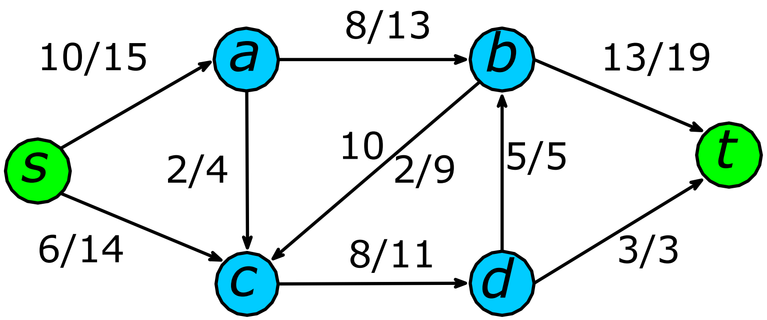
Ford Fulkerson (Max-Flow) Pseudo Code

1. While Exists an Augmenting Path (P)
   a. push maximum possible flow along P (saturating at least one edge on it) , $\boldsymbol{f_p}$
   b. Update the residual Graph (i.e Subtract $\boldsymbol{f_p}$ on the forward edges, add $\boldsymbol{f_p}$ on the reverse edges)
   c. Increase the value of the variable MaxFlow by $\boldsymbol{f_p}$

2. The flow in variable MaxFlow is the maximum flow along the network

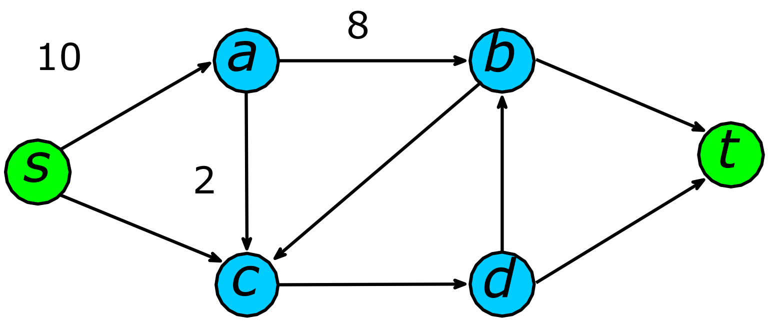
Residual Graph

- Compute the residual graph of the graph with the following flow:



Residual Graph

- Compute the residual graph of the graph with the following flow:



Residual Capacity and Augmenting Path

- An augmenting path is a path of edges in the residual graph with unused capacity greater than zero from the source s to the sink t. It can flow through edges which are not saturated yet.
- We have achieved the max flow when we know that there are no more augmenting paths left to be found.

Residual Capacity and Augmenting Path

- Finding an Augmenting Path
  - Find a path from s to t in the residual graph
  - The r*esidual capacity* of a path p in G_f :

    $c_f(p) = \min\{c_f(u,v): (u,v) \text{ is in } p\}$

    - i.e. find the minimum capacity along p

– Doing augmentation: for all (u,v) in p, we just add this $c_f(p)$ to $f(u,v)$ (and subtract it from $f(v,u)$)

– Resulting flow is a valid flow with a larger value.

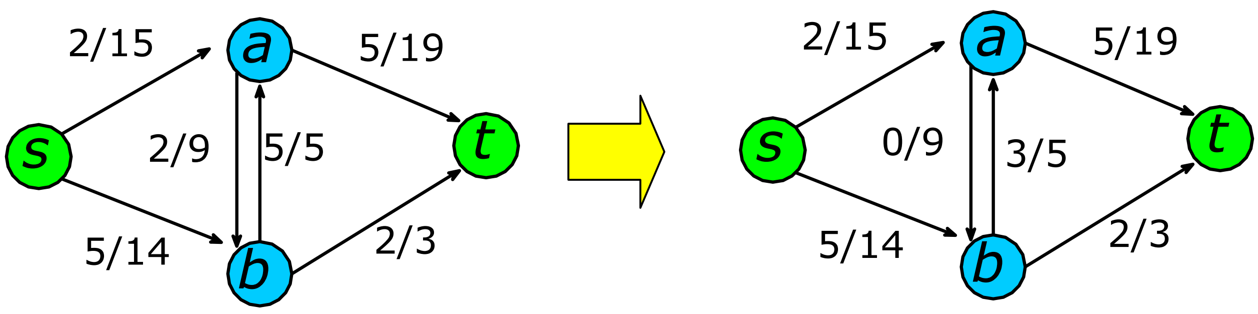
□ The value of a flow is defined as

$$|f| = \sum_{v \in V} f(s,v)$$

□ The total flow from source to any other vertices.

Cancellation of flows

- We would like to avoid two positive flows in opposite directions between the same pair of vertices
  - Such flows *cancel* (maybe partially) each other due to skew symmetry
  - f+f’: the flow in the same direction will be added. the flow in different directions will be cancelled.



Residual network and augmenting path

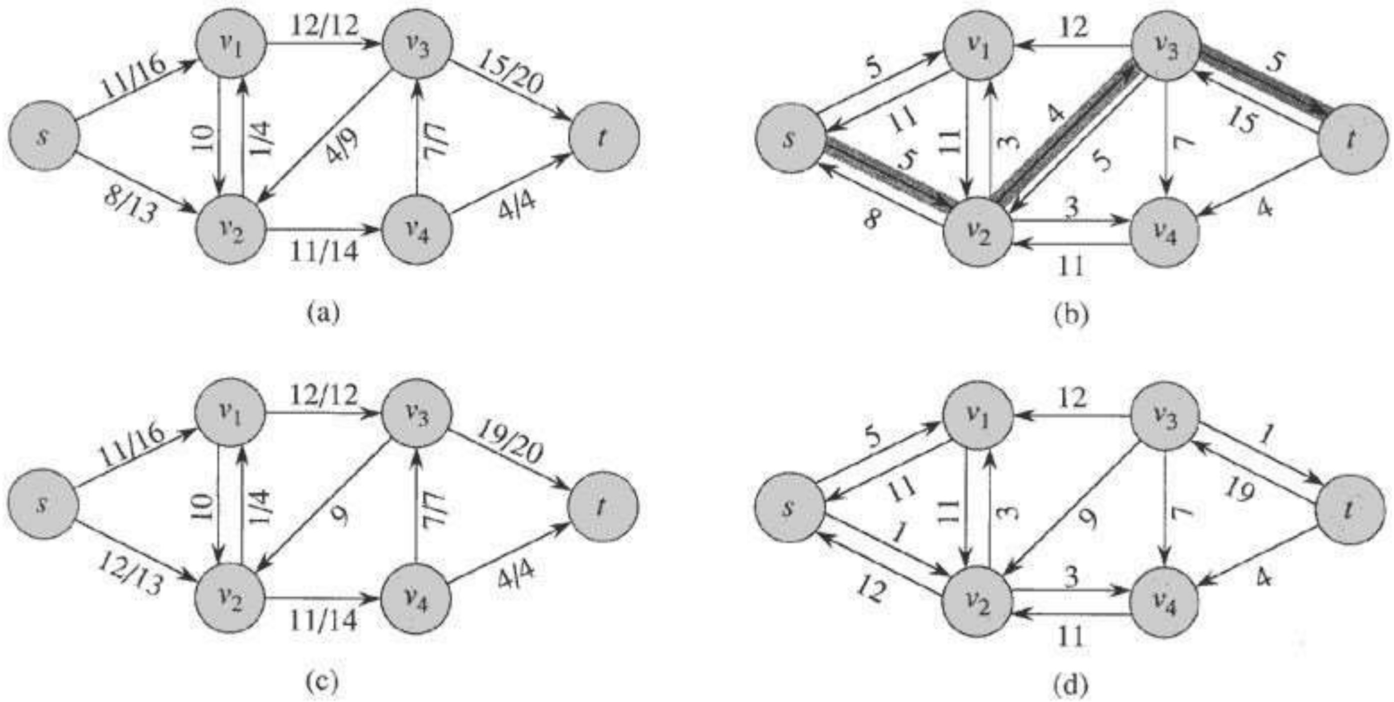


Figure 26.3 **(a)** The flow network G and flow f of Figure 26.1(b). **(b)** The residual network G_f with augmenting path p shaded; its residual capacity is $c_f(p) = c(v_2, v_3) = 4$. **(c)** The flow in G that results from augmenting along path p by its residual capacity 4. **(d)** The residual network induced by the flow in (c).

The Ford-Fulkerson method

```
Ford-Fulkerson(G,s,t)
1 for each edge (u,v) in G.E do
2     f(u,v) ← f(v,u) ← 0
3 while there exists a path p from s to t in residual
  network G_f do
4     c_f = min{c_f(u,v): (u,v) is in p}
5     for each edge (u,v) in p do
6         f(u,v) ← f(u,v) + c_f      //forward edge
7         f(v,u) ← -f(u,v)           //backward edge
8 return f
```

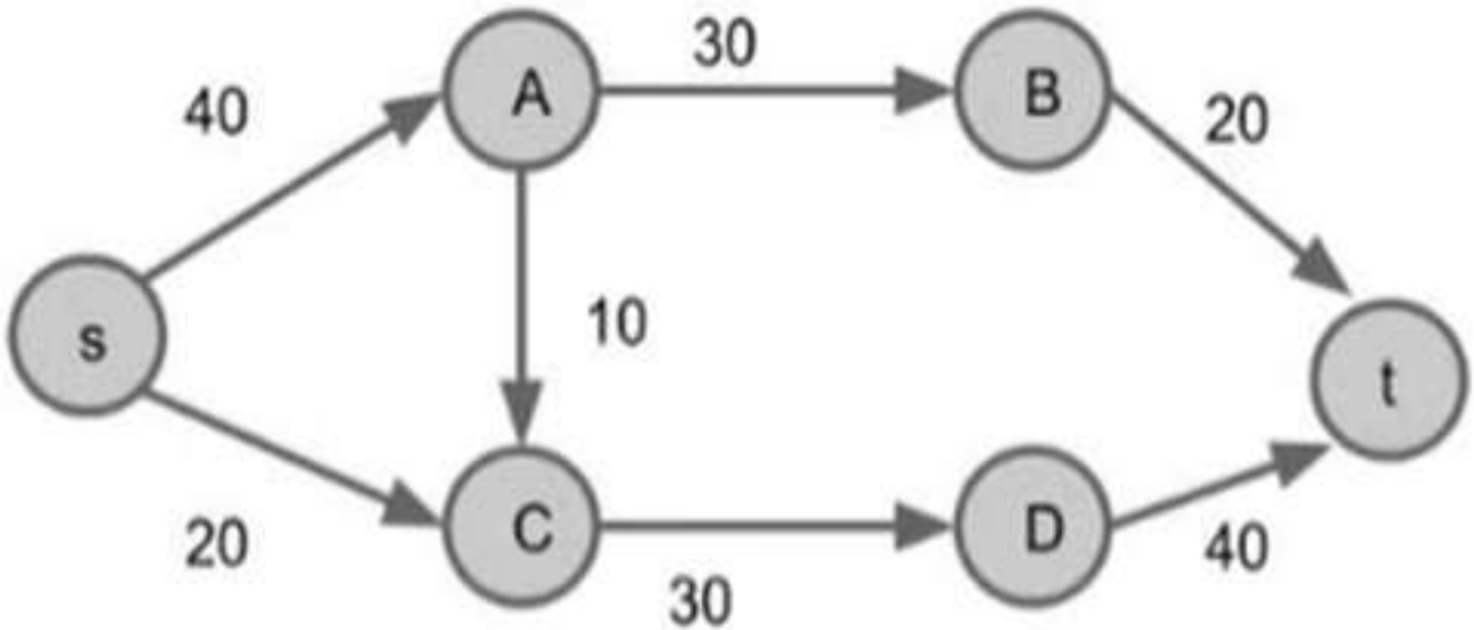
The algorithms based on this method differ in how they choose p in step 3. If chosen poorly the algorithm might not terminate.

Proof of Correctness(Idea):

3 things

1. Every augmenting path is a valid flow (satisfies the capacity constraint, conservation constraint)
2. The algorithm is Finite(each augmenting path increases the flow by at least one)
3. The flow found is Maximum
   (Depends on the MAXFLOW-MINCUT duality theorem)
   a. For any flow f and any s-t cut , f ≤ C(s,t)
   b. Cannot find an augmenting path if the edges in any cut are saturated. This algorithm finds an st cut
   c. Therefore, f <= ANY C(s,t) cut, But f = C(s,t) for some (s,t) cut.

   f = max Flow


s
A
B
C
D
t
40
30
20
10
20
30
40

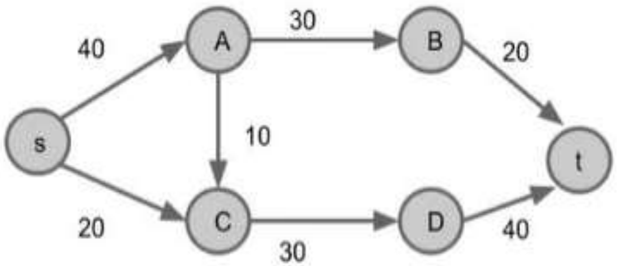
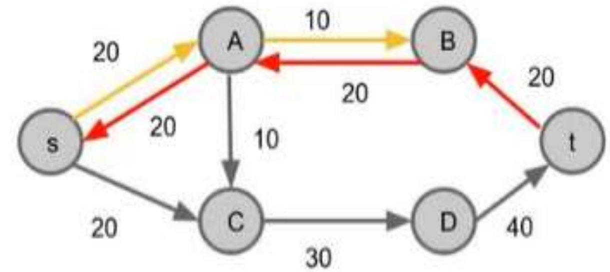
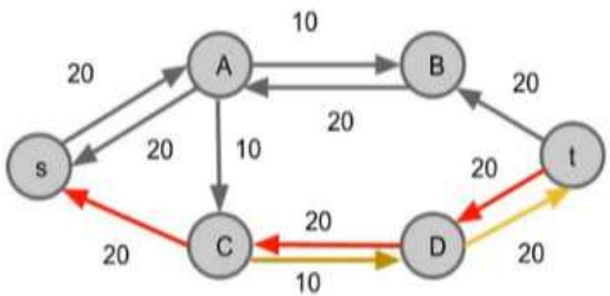


Fig : 1



Augmenting Paths :
s->A->B->t (20)

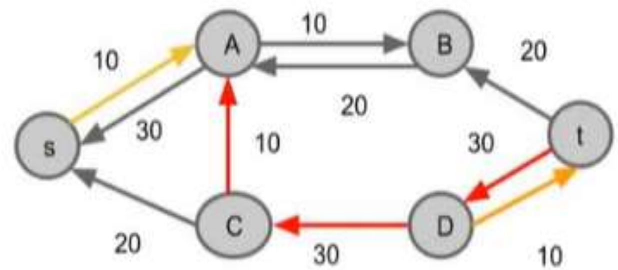
Fig : 2



Augmenting Paths :
1. s > A > B > t (20)
2. s > C > D > t (20)

Total Flow : 40

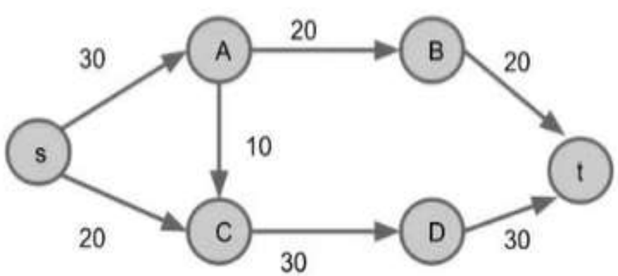
Fig : 3



Augmenting Paths :
1. s > A > B > t (20)
2. s > C > D > t (20)
3. s > A > C > D > t (10)

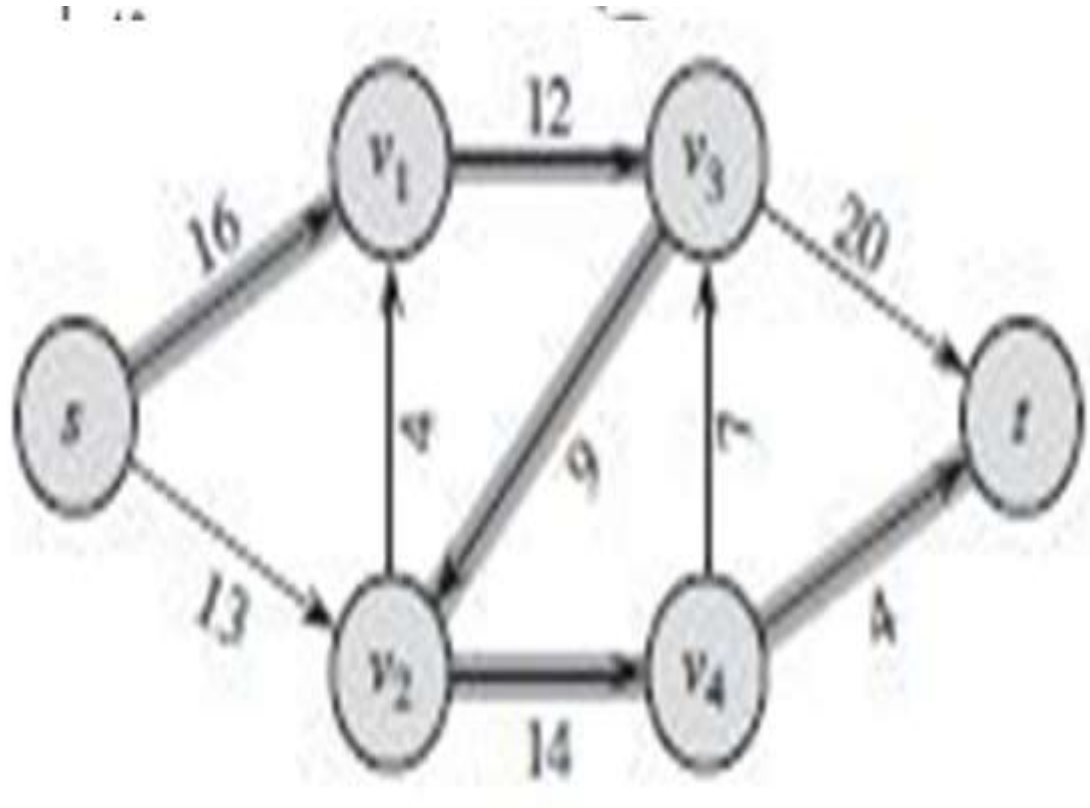
Total Flow : 50

Fig : 4



Max Flow : 50

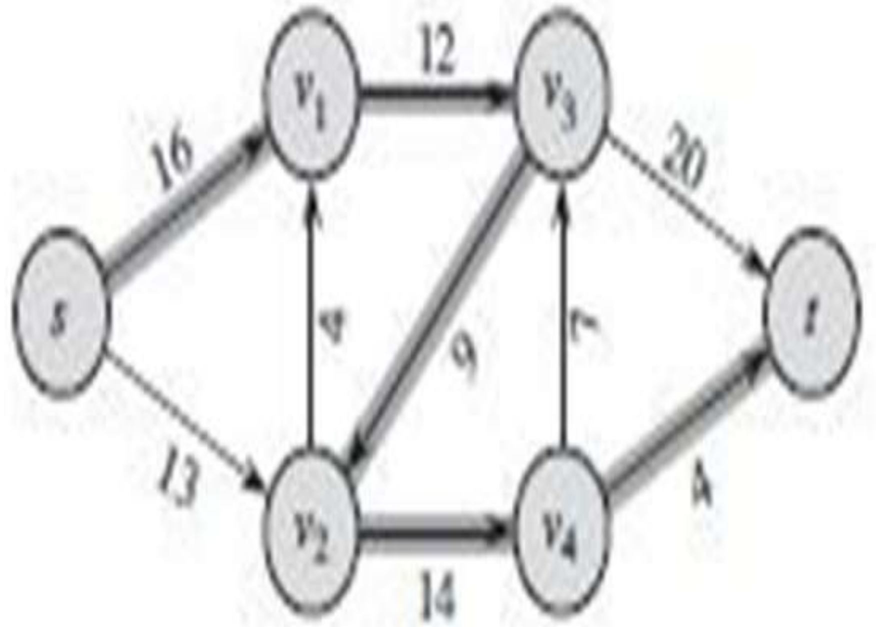
Execution of Ford-Fulkerson (1)

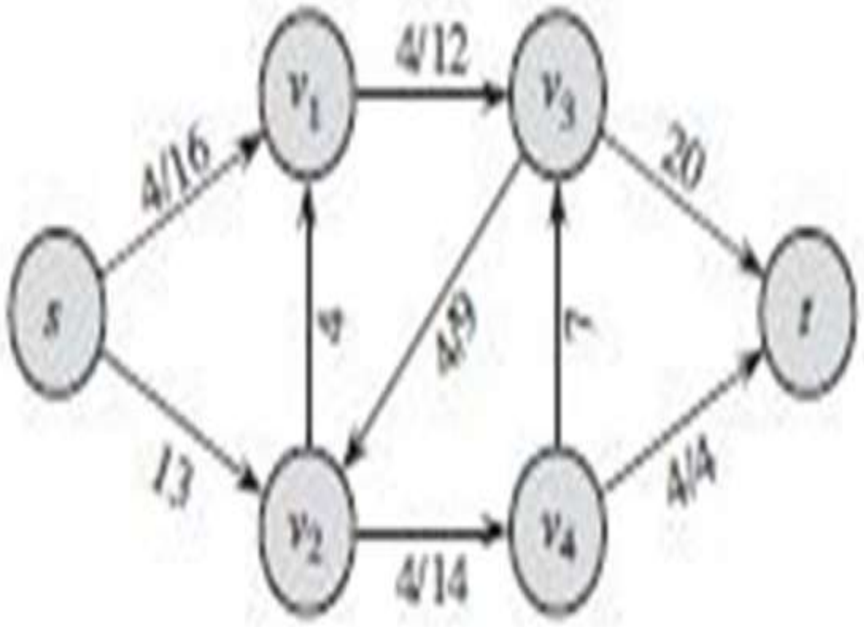


Left Side = Residual Graph

Right Side = Augmented Flow

Execution of Ford-Fulkerson (1)

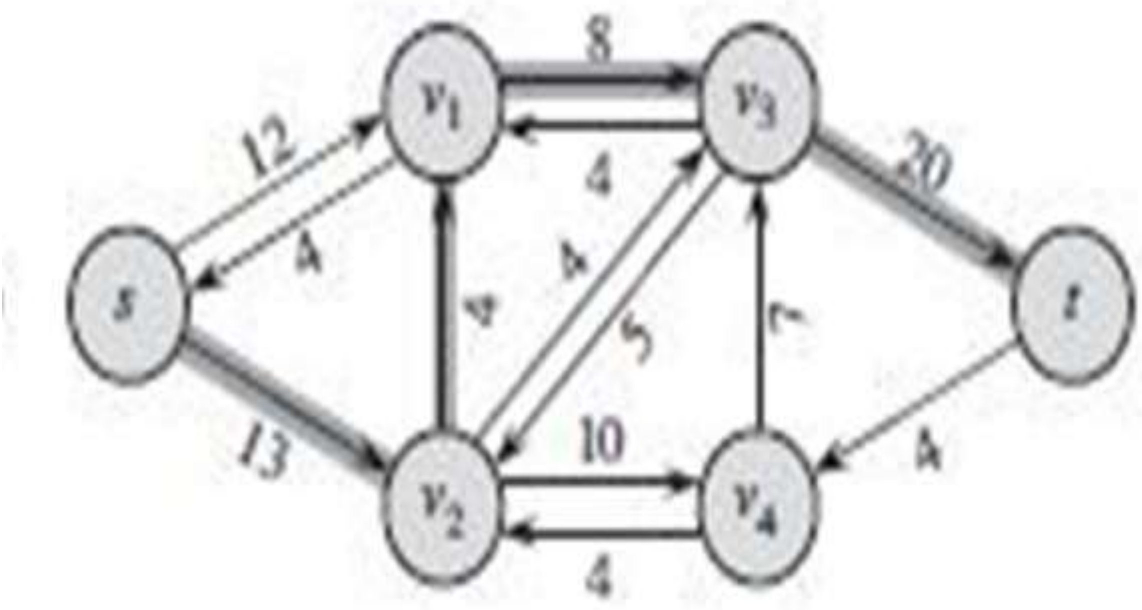




Left Side = Residual Graph

Right Side = Augmented Flow

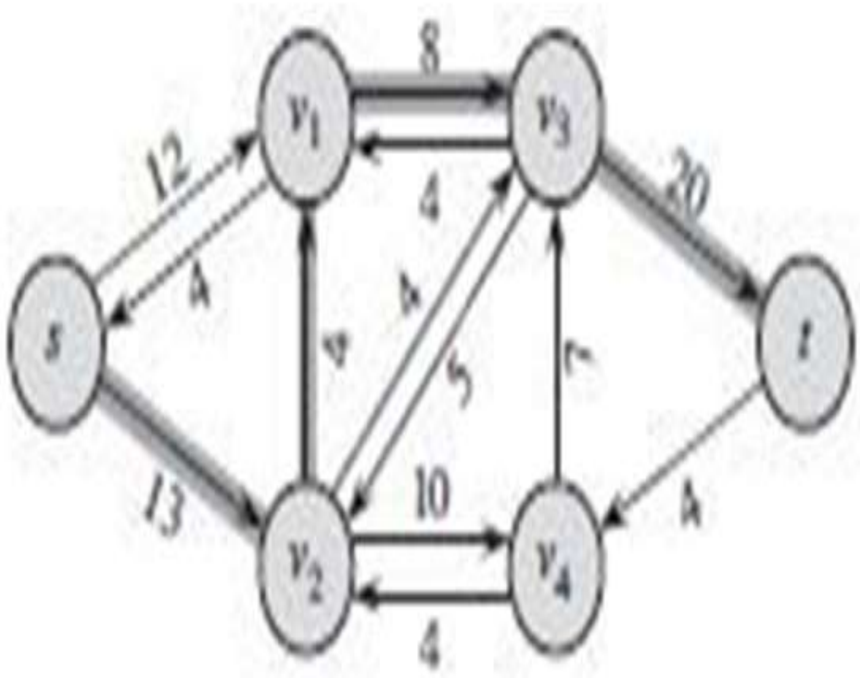
Execution of Ford-Fulkerson (1)

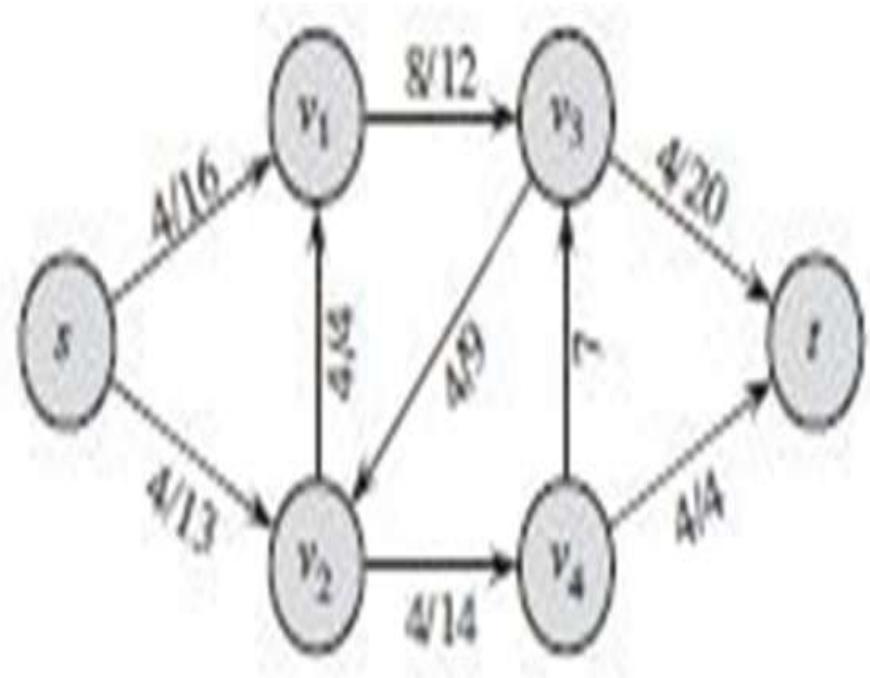


Left Side = Residual Graph

Right Side = Augmented Flow

Execution of Ford-Fulkerson (1)

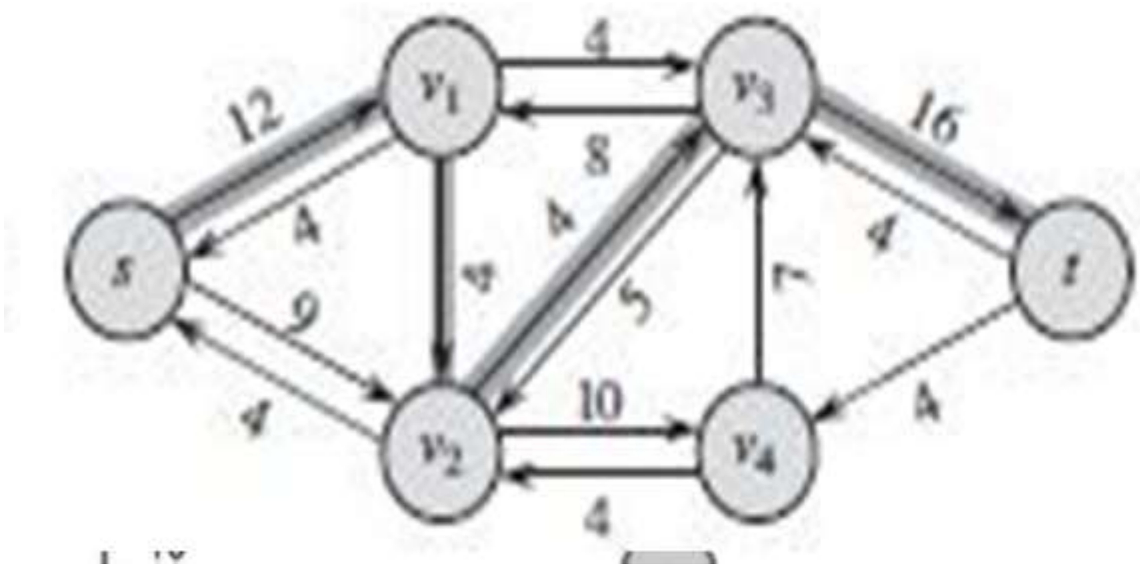




Left Side = Residual Graph

Right Side = Augmented Flow

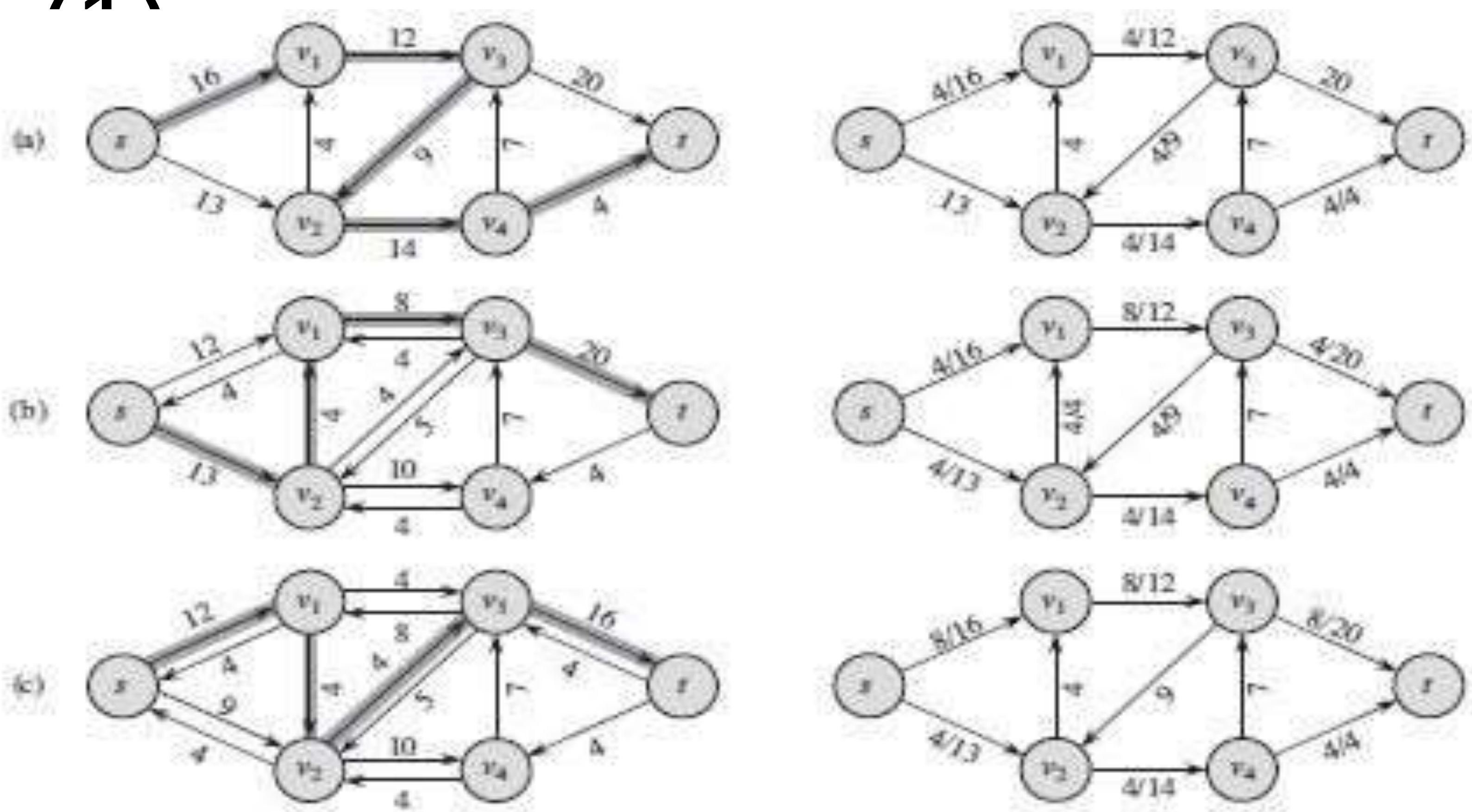
Execution of Ford-Fulkerson (1)



Left Side = Residual Graph

Right Side = Augmented Flow

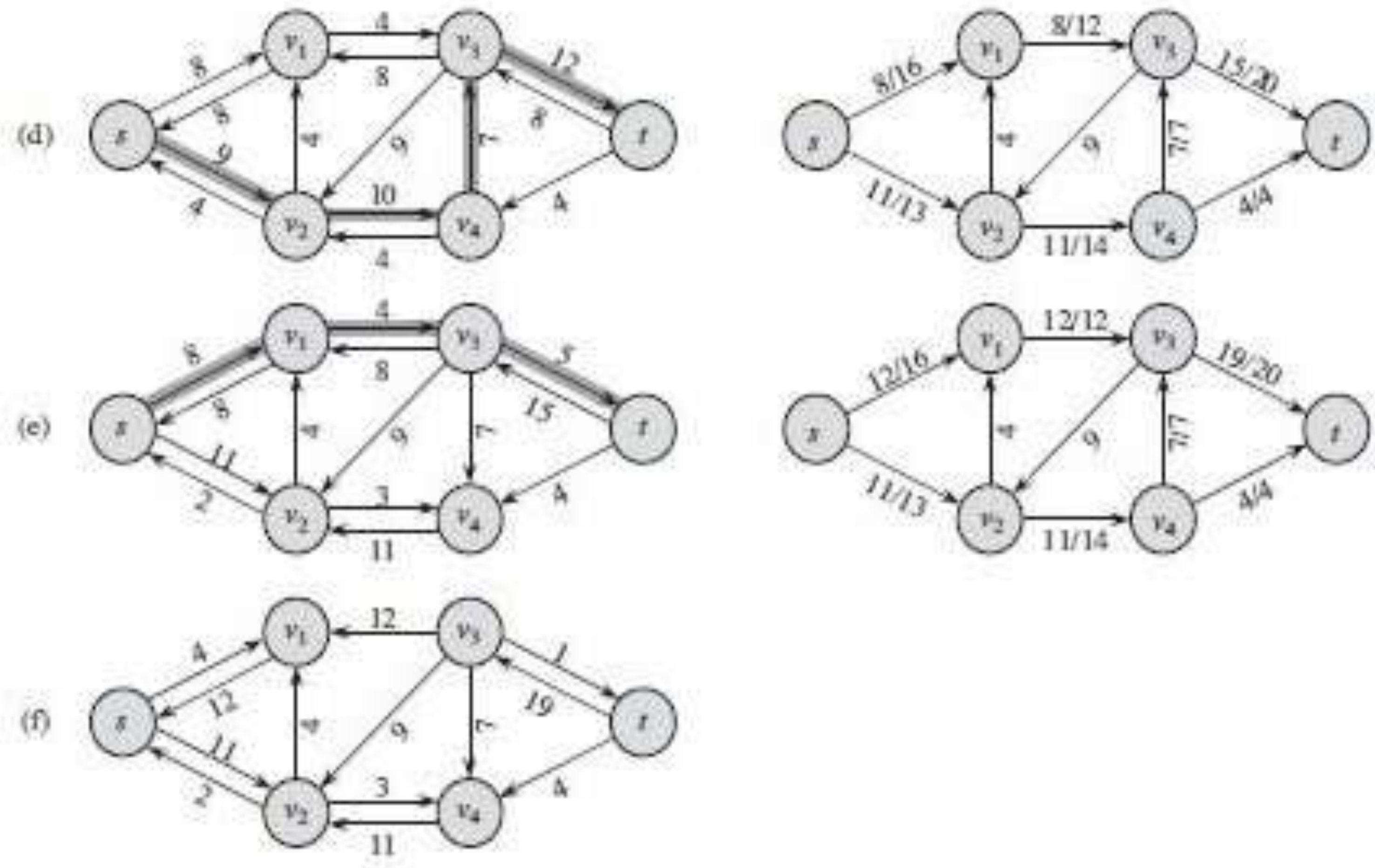
Execution of Ford-Fulkerson (1)



Left Side = Residual Graph

Right Side = Augmented Flow

Execution of Ford-Fulkerson



Left Side = Residual Graph

Right Side = Augmented Flow

Cuts

- A **cut** is a partition of V into S and T = V – S, such that *s* ∈S and *t* ∈T
- The **net flow** (f(S,T)) through the cut is the sum of flows f(u,v), where *s* ∈S and *t* ∈T
  - Includes negative flows back from T to S
- The **capacity** (c(S,T)) of the cut is the sum of capacities c(u,v), where *s* ∈S and *t* ∈T
  - The sum of positive capacities
- **Minimum cut** – a cut with the smallest capacity of all cuts.

|f|= f(S,T) i.e. the value of a max flow is equal to the capacity of a min cut.

Min s-t cut

Definition

s-t cut :

In a flow network, an s-t cut is a set of edges whose removal disconnects s (source) from t (sink).

Or, formally, a cut is a partition of vertex set into A and B, where s is in A and t is in B. (The edges of the cut are then all edges going from A to B)

The capacity of an s-t cut is defined by the sum of capacity of each edge in the cut-set.

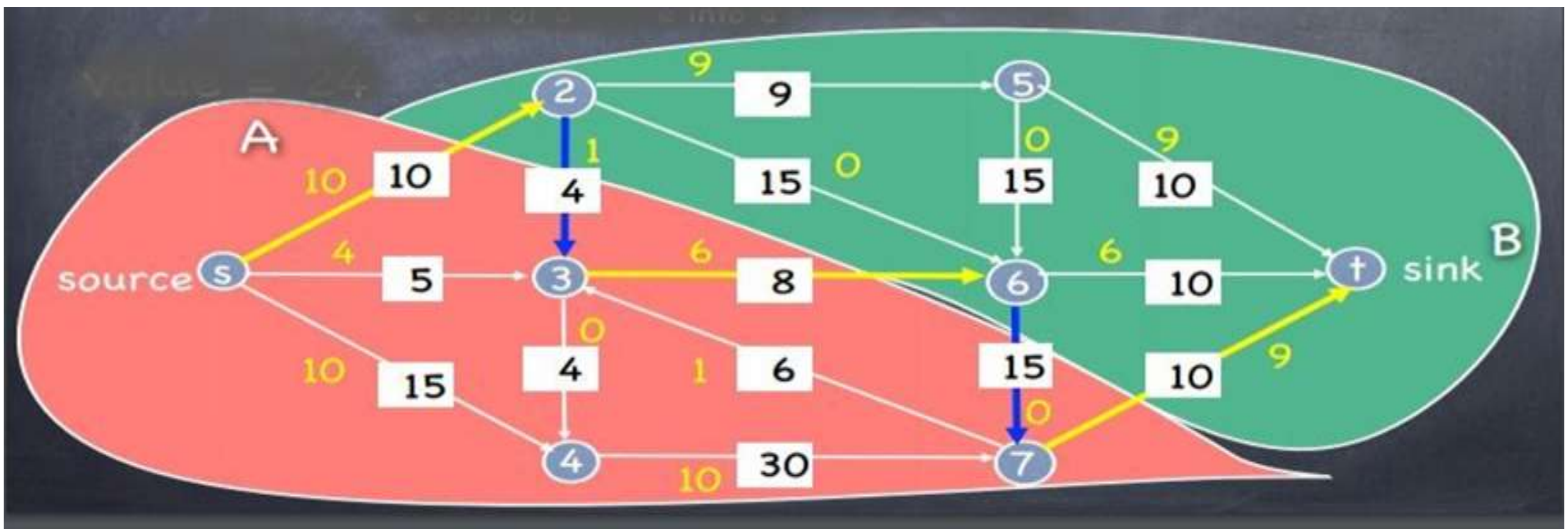
Min s-t cut : Is the minimum of all s-t cuts

Max Flow / Min Cut Theorem

In every network, the maximum flow equals the minimum capacity of a cut.

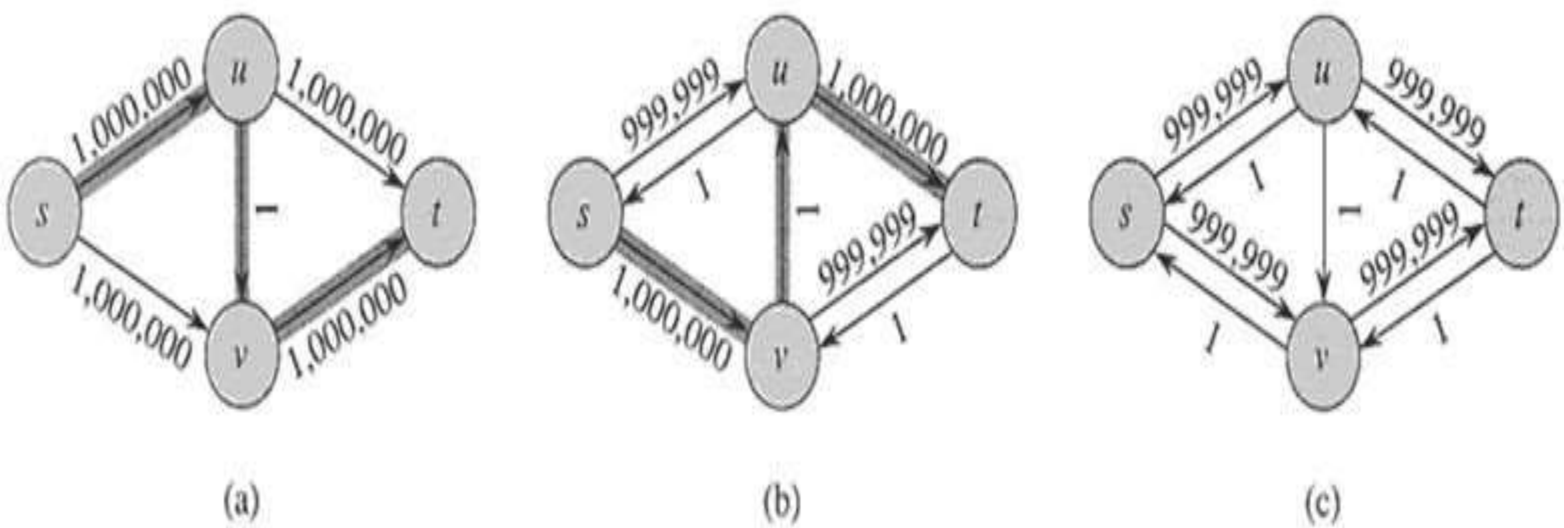
1. Since $|f| \le c(S,T)$ for all cuts of (S,T) then if $|f| = c(S,T)$ then $c(S,T)$ must be the min cut of G
2. This implies that f is a maximum flow of G
3. This implies that the residual network G_f contains no augmenting paths.
   - If there were augmenting paths this would contradict that we found the maximum flow of G

Cut Value=10+6+9= 25



Worst Case Running Time

- Assuming integer flow
- Each augmentation increases the value of the flow by some positive amount.
- Augmentation can be done in O(E).
- Total worst-case running time O(E|f*|), where f* is the max-flow found by the algorithm.
- Example of worst case:



Augmenting path of 1

Resulting Residual Network

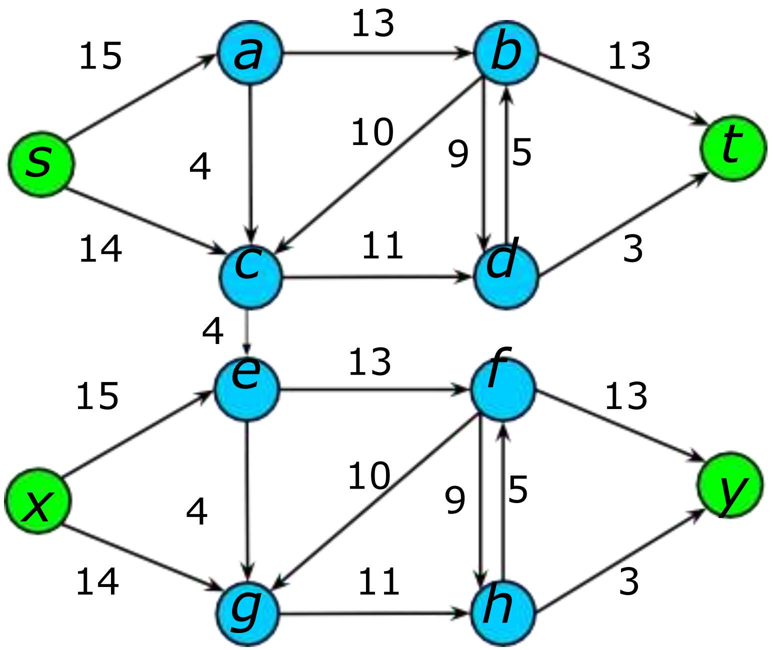
Resulting Residual Network

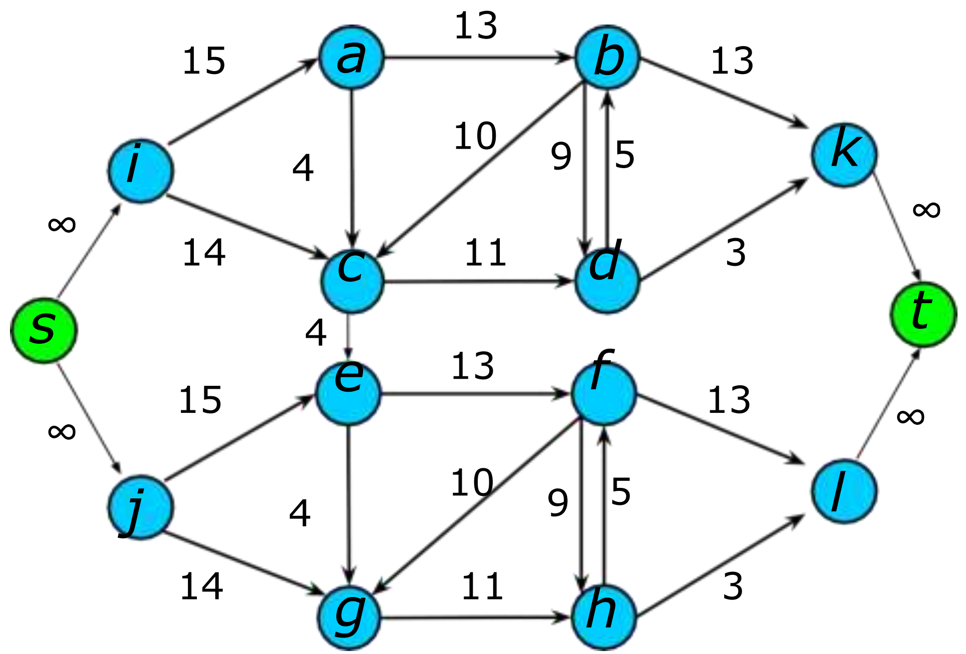
Edmonds Karp

- Take **shortest path** (in terms of number of edges) as an augmenting path – Edmonds-Karp algorithm
  - Augmenting path is found using BFS
  - Running time $O(VE^2)$, because the number of augmentations is $O(VE)$
  - Even better method: push-relabel, $O(V^2E)$ runtime

Multiple Sources or Sinks

- What if you have a problem with more than one source and more than one sink?
- Modify the graph to create a single supersource and supersink



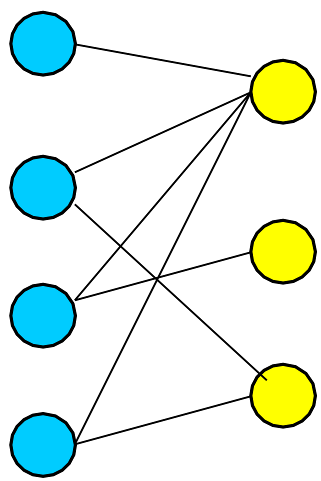


Application – Bipartite Matching

- Example – given a community with n men and m
  women
- Assume we have a way to determine which couples (man/woman) are compatible for marriage
  - E.g. (Joe, Susan) or (Fred, Susan) but not (Frank, Susan)
- Problem: Maximize the number of marriages
  - No polygamy allowed
  - Can solve this problem by creating a flow network out of a bipartite graph

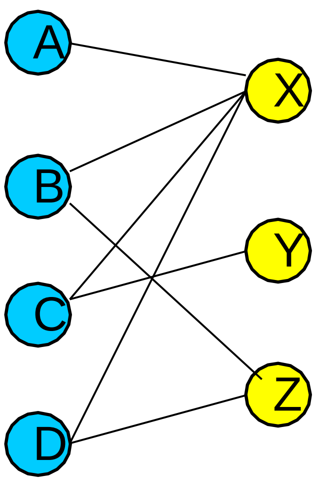
Bipartite Graph

- A bipartite graph is an undirected graph $G=(V,E)$ in which V can be partitioned into two sets V_1 and V_2 such that $(u,v) \in E$ implies either $u \in V_1$ and $v \in V_2$ or vice versa.
- That is, all edges go between the two sets V_1 and V_2 and not within V_1 and V_2.



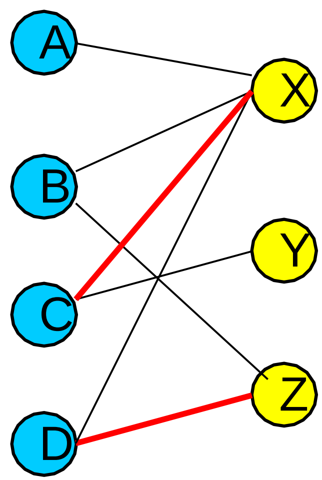
Model for Matching Problem

- Men on leftmost set, women on rightmost set, edges if they are compatible

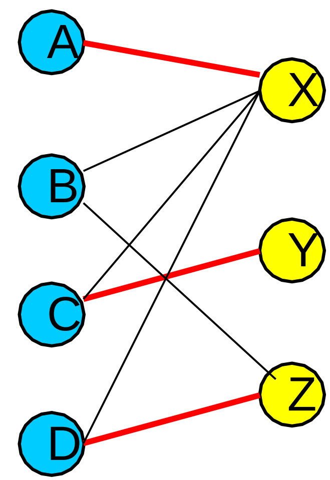


Me
n

Wome
n



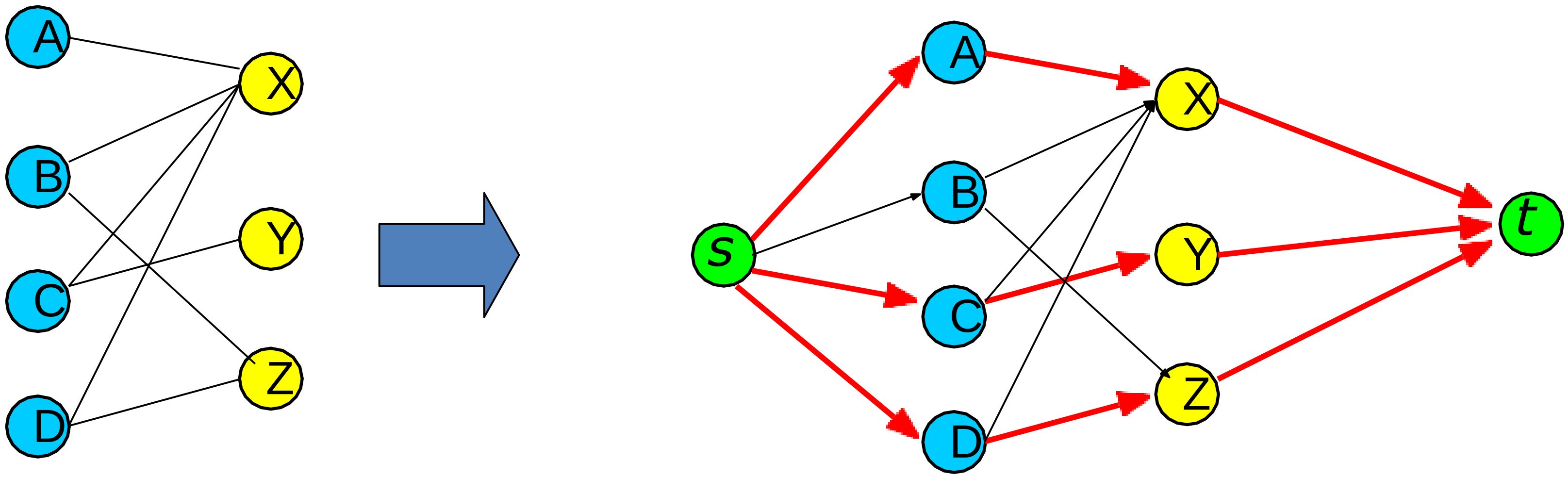
A
matching



Optimal
matching

Solution Using Max Flow

- Add a supersouce, supersink, make each undirected edge directed with a flow of 1



Since the input is 1, flow conservation prevents multiple matchings

- https://www.youtube.com/watch?v=LdOnanfc5TM
- https://www.slideserve.com/dana-mccall/maximum-flow-problems
- https://www.youtube.com/watch?v=_KiOyfrZdTo
- https://www.youtube.com/watch?v=40Iv3ERfS74
- https://www.youtube.com/watch?v=NwenwITjMys
- https://www.youtube.com/watch?v=LdOnanfc5TM
- https://www.youtube.com/watch?v=GiN3jRdgxU4
- https://www.youtube.com/watch?v=6jq52v6Gkts
- Min-Cut: https://www.youtube.com/watch?v=ylxhl1ipWss
- Bipartite: https://www.youtube.com/watch?v=GhjwOiJ4SqU
- Edmonds-Karp: https://www.youtube.com/watch?v=w3Nl2XA0pxA