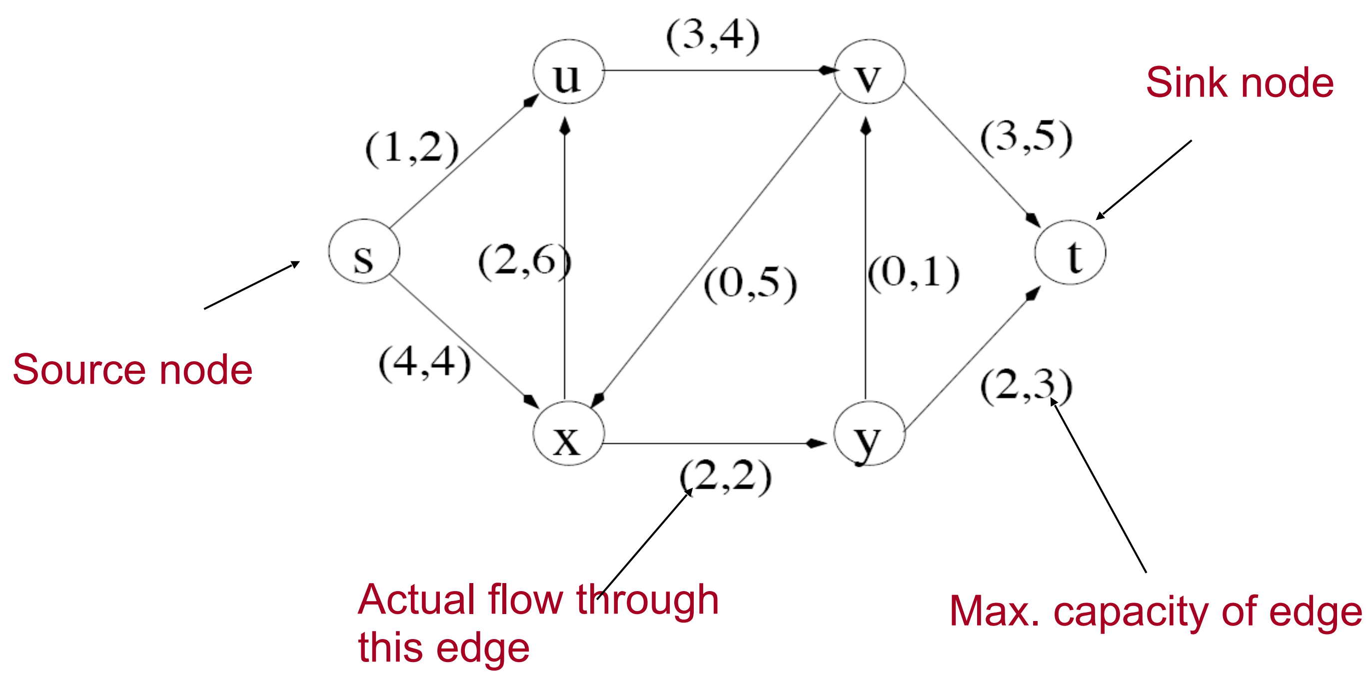
# NETWORK FLOW

# Usage Places

| Network | Vertex | Edges | 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, currency | transactions | money |
| transportation | airports, rail yards, street intersections | highways, railways, airway routes | freight, vehicles, passengers |
| chemical | sites | bonds | energy |

# Example Graph



# Simplified Model

The network is modeled simply as

a) a directed graph G = (V,E) with
b) non-negative capacity on each edge,
c) a single source node, *s*, and
d) a single sink node, *t*

# Some assumptions…

We will simplify our discussion by assuming the following:

(i) No edge enters *s*, the source

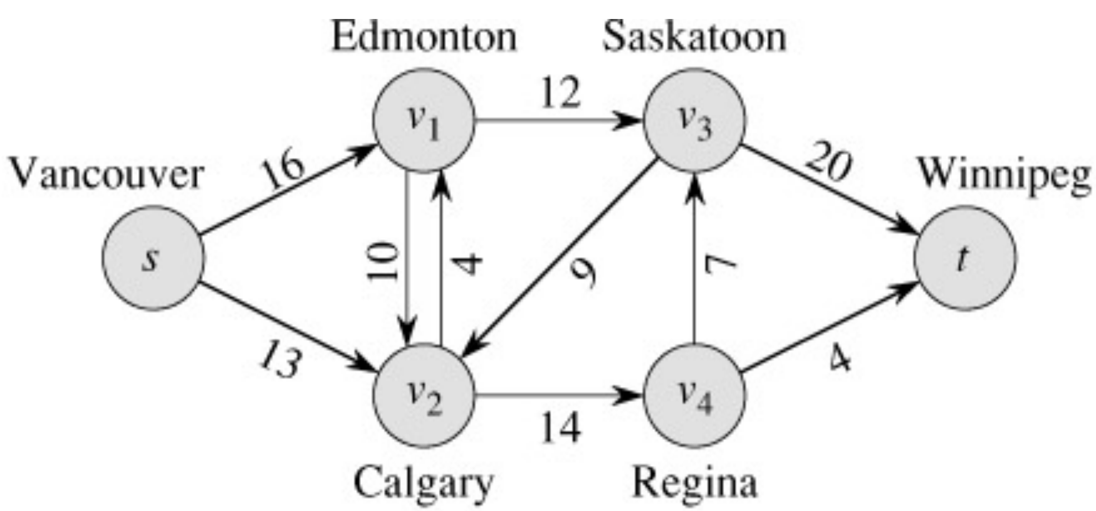
(ii) No edge leaves *t*, the sink

(iii) At least one edge is incident to each node

(iv) All capacities are integers

# What is Network Flow?

- Each edge (u,v) has a nonnegative capacity c(u,v).
- If (u,v) is not in E, assume c(u,v)=0.
- We have a source s, and a sink t.
- Assume that every vertex v in V is on some path from s to t.
- $c(s,v_1)=16$; $c(v_1,s)=0$; $c(v_2,v_3)=0$



# Formalization

# Formalization

**Flow network** – $G=(V,E)$

# Formalization

**Flow network** – $G=(V,E)$

- Directed, each edge has **capacity** $c(u,v) \geq 0$

# Formalization

**Flow network** – $G=(V,E)$

- Directed, each edge has **capacity** $c(u,v) \geq 0$
- Two special vertices: ***source*** *s,* and ***sink*** *t*

# Formalization

**Flow network** – $G=(V,E)$

- 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 $s \rightarrow \ldots \rightarrow v \rightarrow \ldots \rightarrow t$

# Formalization

**Flow network** – $G=(V,E)$

- 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 $s \rightarrow \ldots \rightarrow v \rightarrow \ldots \rightarrow t$

**Flow** – $f : V \times V \rightarrow \boldsymbol{R}$

# Formalization

**Flow network** – $G=(V,E)$
- 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** – $f : V \times V \to \boldsymbol{R}$
- *Capacity constraint*: $\forall u,v \in V$: $f(u,v) \leq c(u,v)$

# Formalization

**Flow network** – $G=(V,E)$

- 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** – $f : V \times V \to \mathbf{R}$

- *Capacity constraint*: $\forall u,v \in V$: $f(u,v) \leq c(u,v)$
- *Skew symmetry*: $\forall u,v \in V$: $f(u,v) = -f(v,u)$

# Formalization

**Flow network** – $G=(V,E)$

- 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** – $f : V \times V \to \boldsymbol{R}$

- *Capacity constraint*: $\forall u,v \in V$: $f(u,v) \leq c(u,v)$
- *Skew symmetry*: $\forall u,v \in V$: $f(u,v) = -f(v,u)$
- *Flow conservation:* $\forall u \in V - \{s, t\}$:

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

# Flow in a Flow Network

– A flow in the network is an integer-valued function f defined on the edges of G satisfying $0 \le f(i,j) \le c(i,j)$ for every edge (i,j) in E.

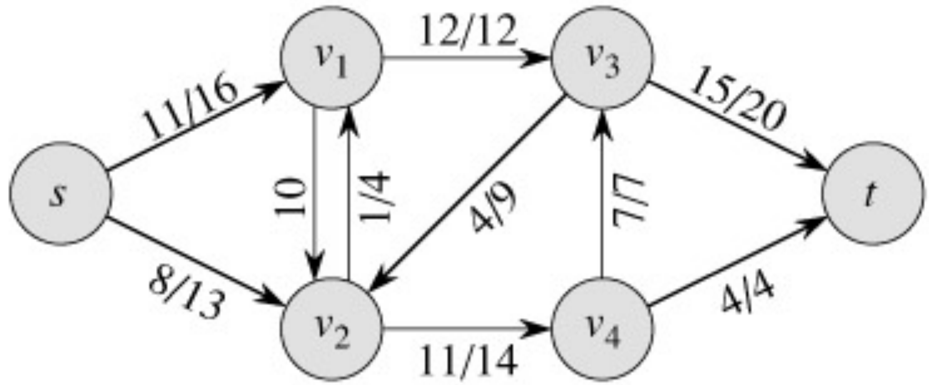
# The Value of a Flow

- The value of a flow is given by

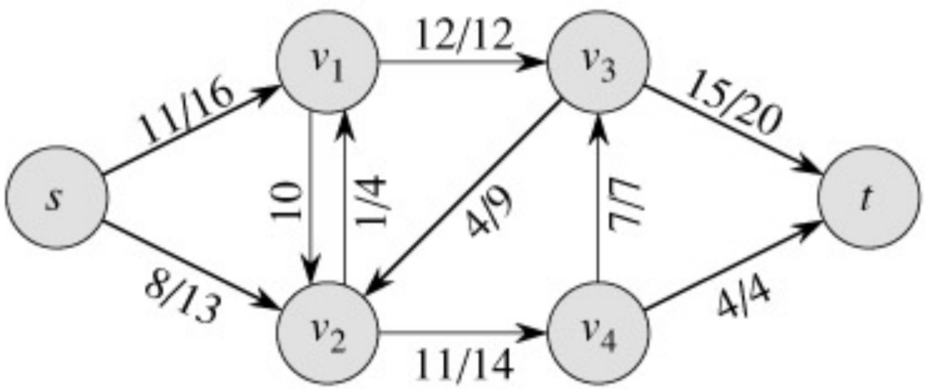
$$|f| = \sum_{v \in V} f(s,v) = \sum_{v \in V} f(v,t)$$

- This it the total flow leaving $s$ = the total flow arriving in $t$

# Example:

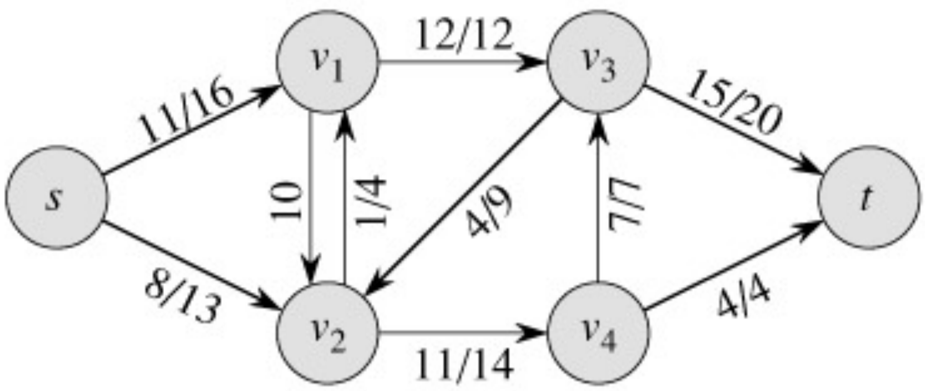


# Example:




$|f| = f(s, v_1) + f(s, v_2) + f(s, v_3) + f(s, v_4) + f(s, t) =$

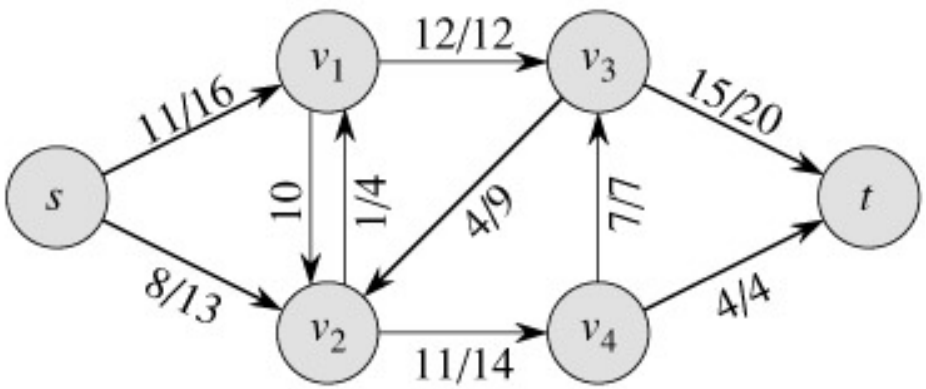
# Example:




$|f| = f(s, v_1) + f(s, v_2) + f(s, v_3) + f(s, v_4) + f(s, t) =$

$11 + 8 + 0 + 0 + 0 = 19$

# Example:

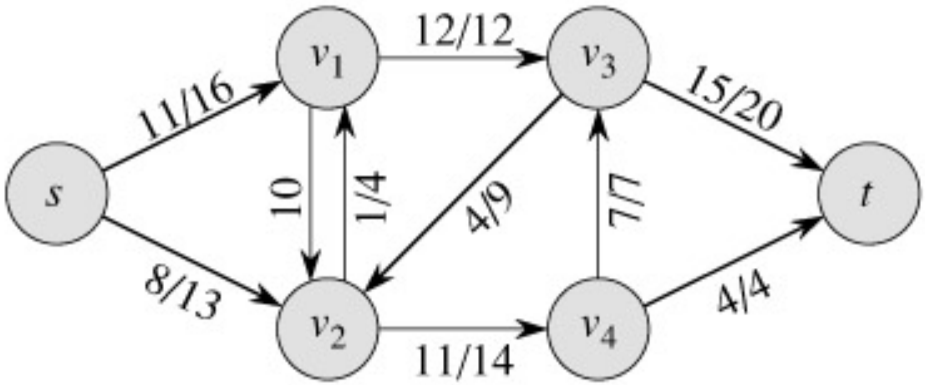



$|f| = f(s, v_1) + f(s, v_2) + f(s, v_3) + f(s, v_4) + f(s, t) =$

$11 + 8 + 0 + 0 + 0 = 19$

$|f| = f(s, t) + f(v_1, t) + f(v_2, t) + f(v_3, t) + f(v_4, t) =$

# Example:



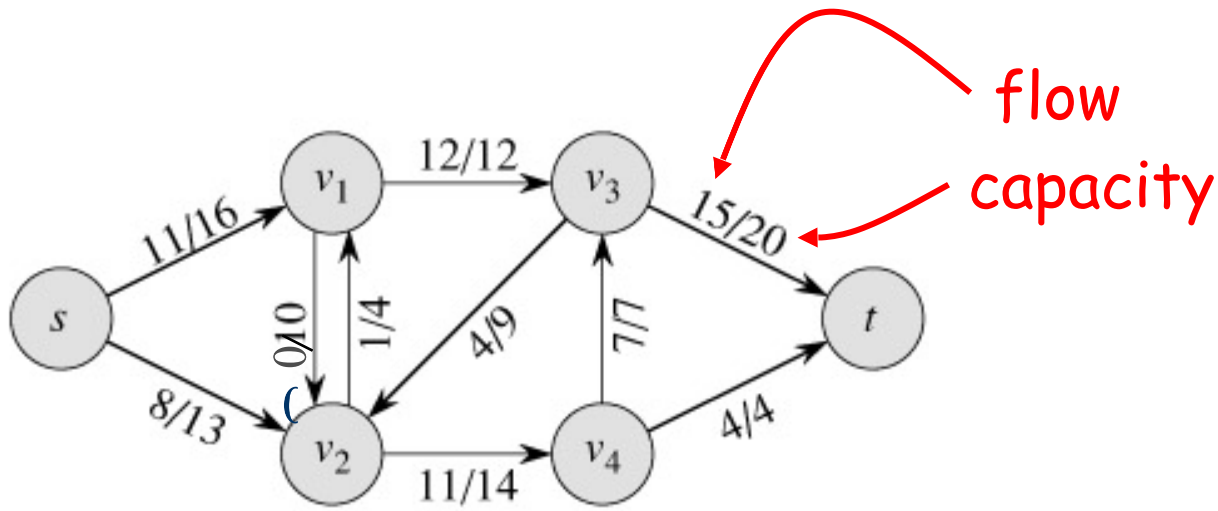

$|f| = f(s, v_1) + f(s, v_2) + f(s, v_3) + f(s, v_4) + f(s, t) =$

$11 + 8 + 0 + 0 + 0 = 19$

$|f| = f(s, t) + f(v_1, t) + f(v_2, t) + f(v_3, t) + f(v_4, t) =$

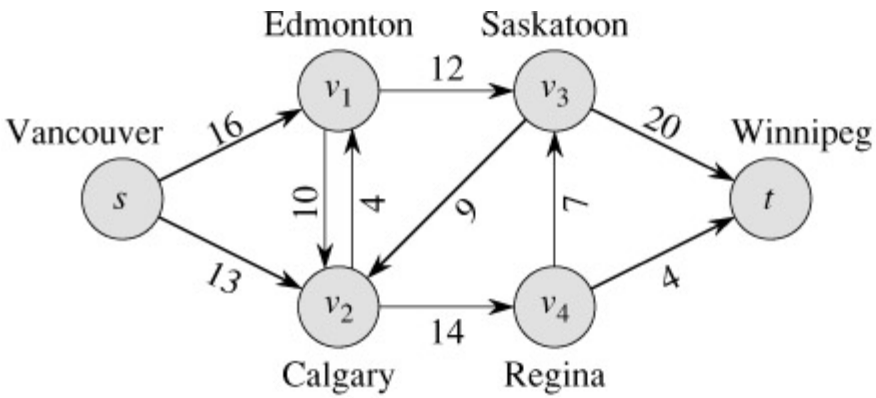
$0 + 0 + 0 + 15 + 4 = 19$

# Example of a Flow



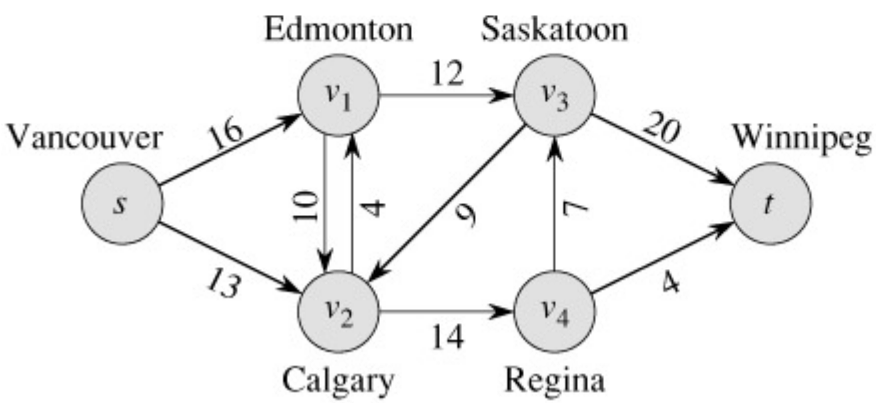

- $f(v_2, v_1) = 1$, $c(v_2, v_1) = 4$.
- $f(v_1, v_2) = -1$, $c(v_1, v_2) = 10$.
- f(v3, s) + f(v3, v1) + f(v3, v2) + f(v3, v4) + f(v3, t) =
- 0 + (-12) + 4 + (-7) + 15 = 0

# A flow in a network



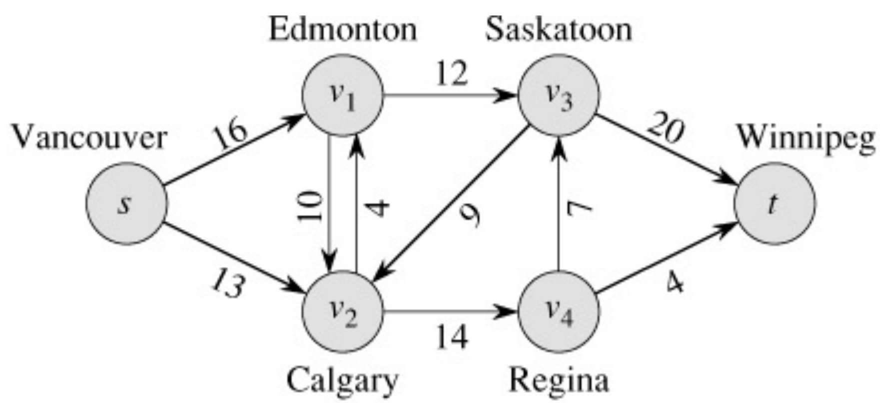
# A flow in a network

- We assume that there is only flow in one direction at a time.



# A flow in a network

- We assume that there is only flow in one direction at a time.




- Sending 7 trucks from Edmonton to Calgary and 3 trucks from Calgary to Edmonton has the same net effect as sending 4 trucks from Edmonton to Calgary.

# Maximum flow

# Maximum flow

- *What do we want to maximize*?

# Maximum flow

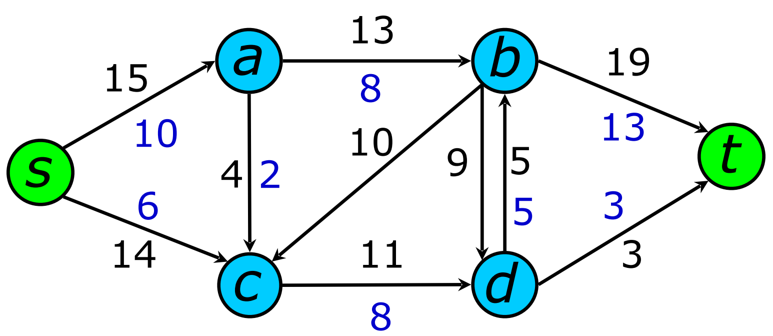
- *What do we want to maximize*?
  – ***Value*** of the flow *f*:

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

# Maximum flow

- *What do we want to maximize?*
  - ***Value*** of the flow *f*:

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

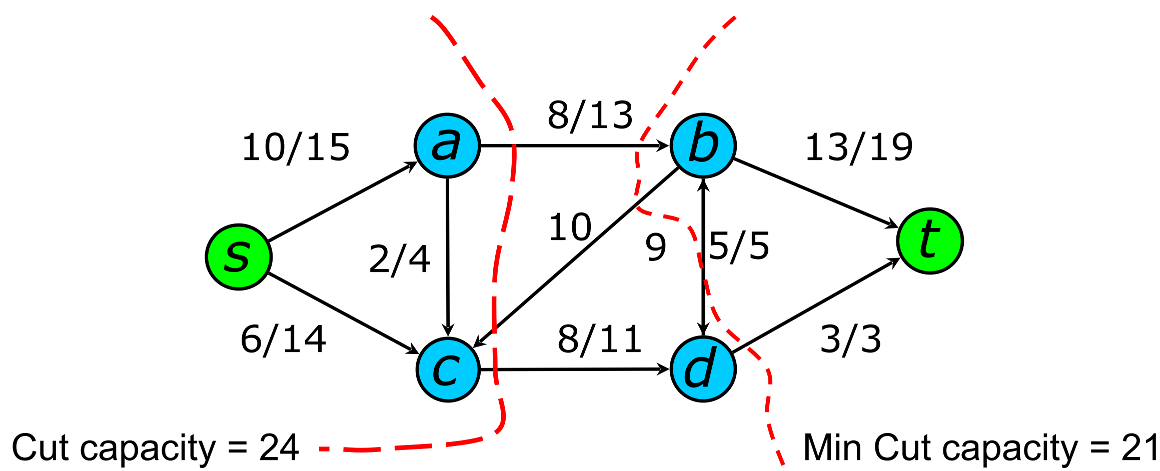


# Some Lemmas:

- Prove that, $f(s,V)=f(V,t)$
- [Lemma26.2] Prove that, $|f+f'|=|f|+|f'|$
- *Lemma: 26.3*
- *Lemma 26.4* $|f'| = |f| + |f_p| \geq |f|$
- *Lemma: 26.5* $|f| = f(S,T)$
- *Lemma: 26.6* $|f| \leq c(S,T)$

# Cuts

- A **cut** is a partition of V into S and T = V – S, such that $s \in$ S and $t \in$ T
- The **net flow** (f(S,T)) through the cut is the sum of flows f(u,v), where $s \in$ S and $t \in$ 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 $s \in$ S and $t \in$ 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.

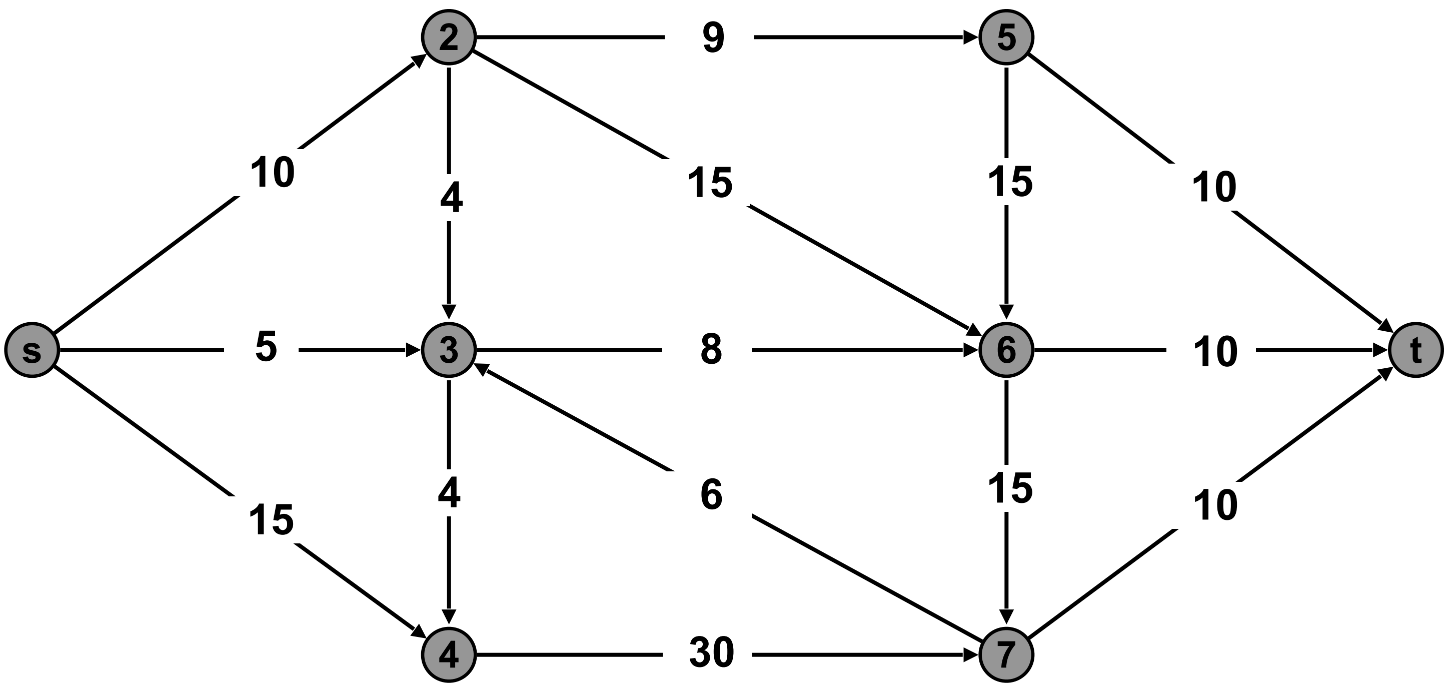


# Max Flow Network

_G = (V, E, s, t, u) _(V, E) = directed graph, no parallel arcs.

_Two distinguished nodes:  s = source, t = sink.

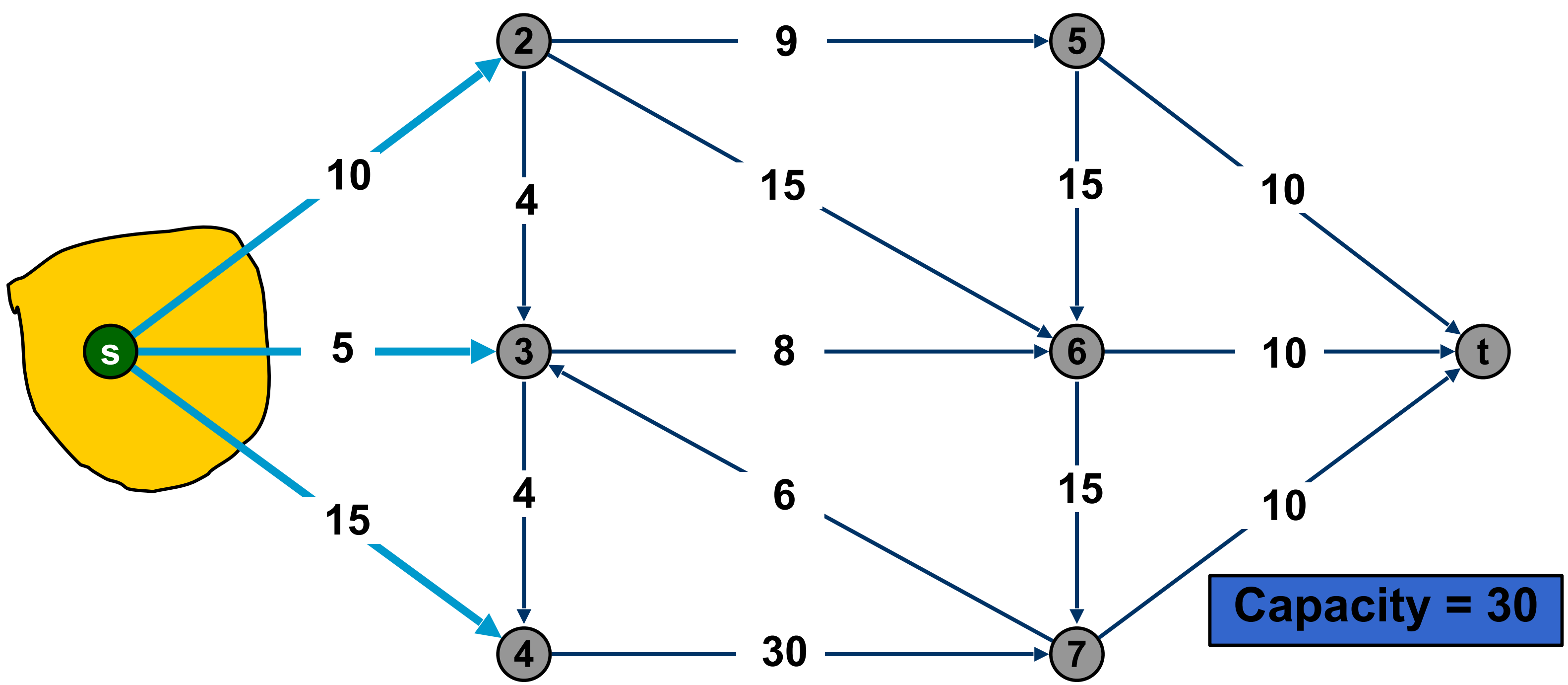
_u(e) = capacity of arc e.




**MAX FLOW:  find s-t flow that maximizes net flow out of the source.**

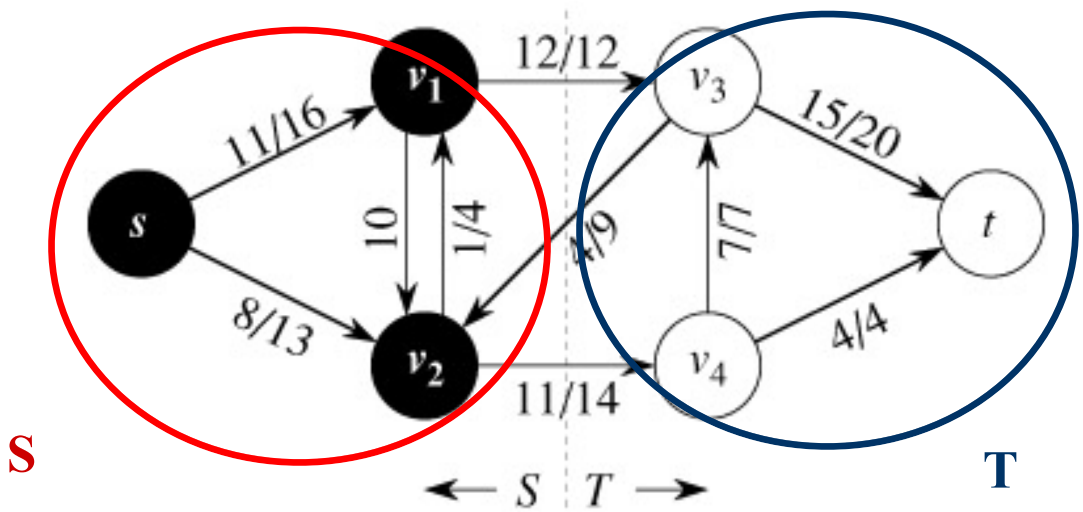
# Cuts of Flow Networks

- A cut in a network is a partition of V into S and T=V-S so that s is in S and t is in T.



# The Net Flow Through a Cut(S,T)

$$f(S,T) = \sum_{u \in S, v \in T} f(u,v) - \sum_{v \in T, u \in S} f(v,u)$$

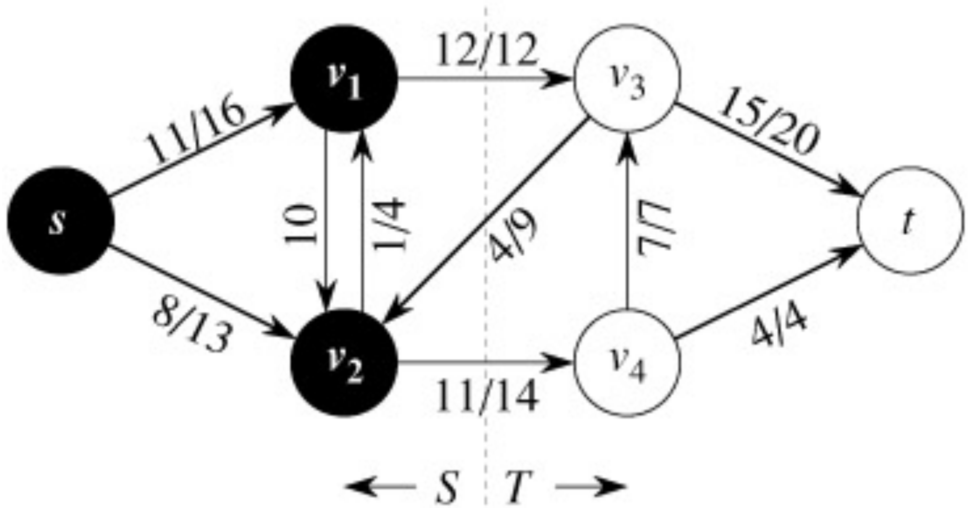



$f(S,T) = 12 – 4 + 11 = 19$

**The value of any flow f in a flow network G is bounded from above by the capacity of any cut of G.**

# The Capacity of Cut(S,T)

$$\boxed{c(S,T) = \sum_{u \in S, v \in T} c(u,v)}$$




c(S,T)= 12+ 14 = 26

# The Ford-Fulkerson Method

# The Ford-Fulkerson Method

- Try to improve the flow, until we reach the maximum.

# The Ford-Fulkerson Method

- Try to improve the flow, until we reach the maximum.
- The residual capacity of the network with a flow f is given by:

# The Ford-Fulkerson Method

- Try to improve the flow, until we reach the maximum.
- The residual capacity of the network with a flow f is given by:

$$c_f(u,v) = c(u,v) - f(u,v)$$

# The Ford-Fulkerson Method

- Try to improve the flow, until we reach the maximum.
- The residual capacity of the network with a flow f is given by:

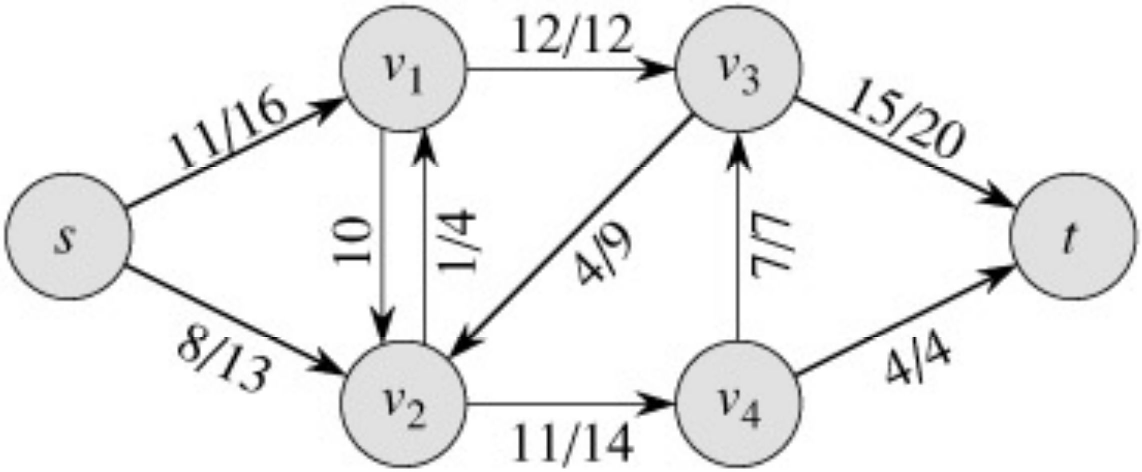
$$c_f(u,v) = c(u,v) - f(u,v)$$

Always nonnegative (why?)

# Example of residual capacities

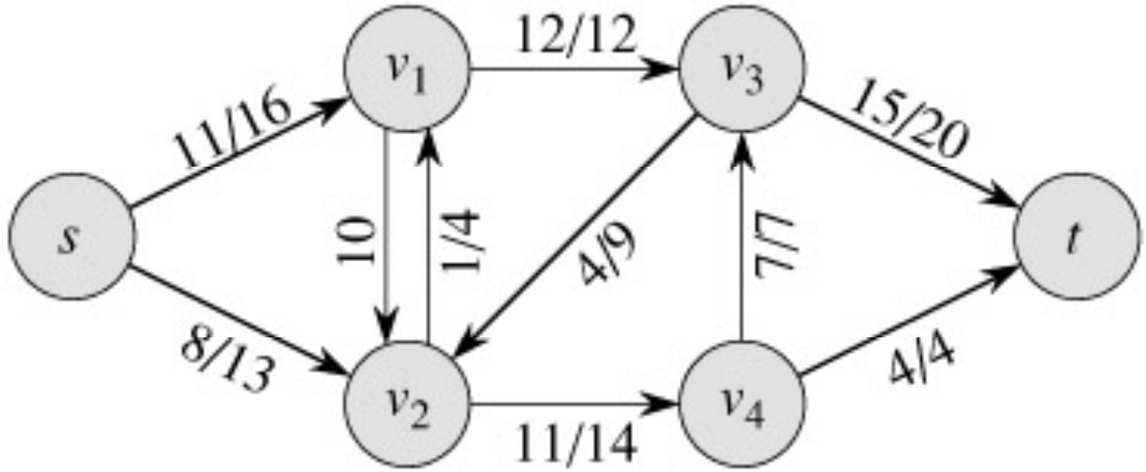
# Example of residual capacities

Network:

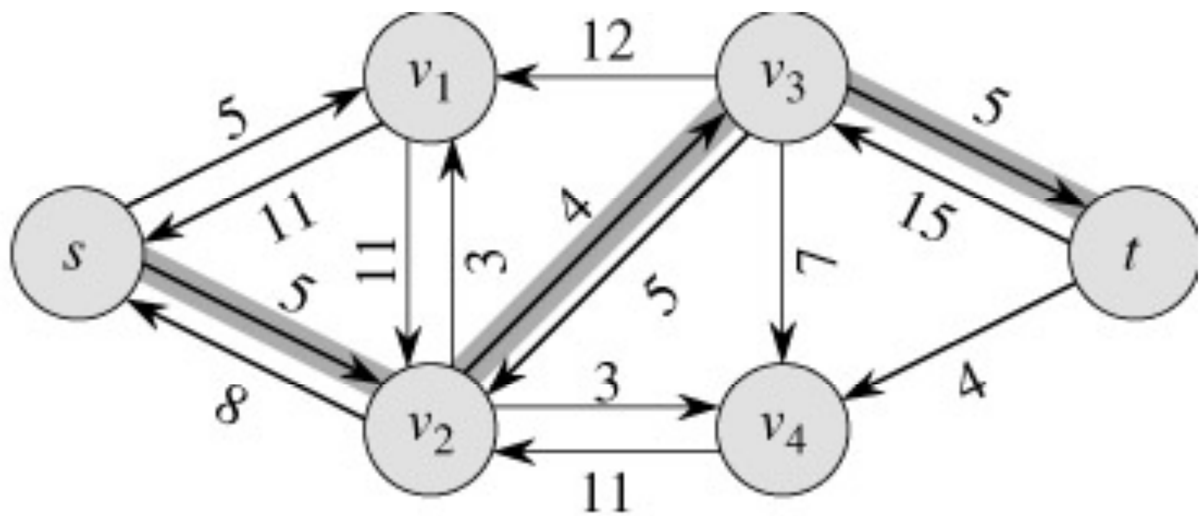


# Example of residual capacities

Network:

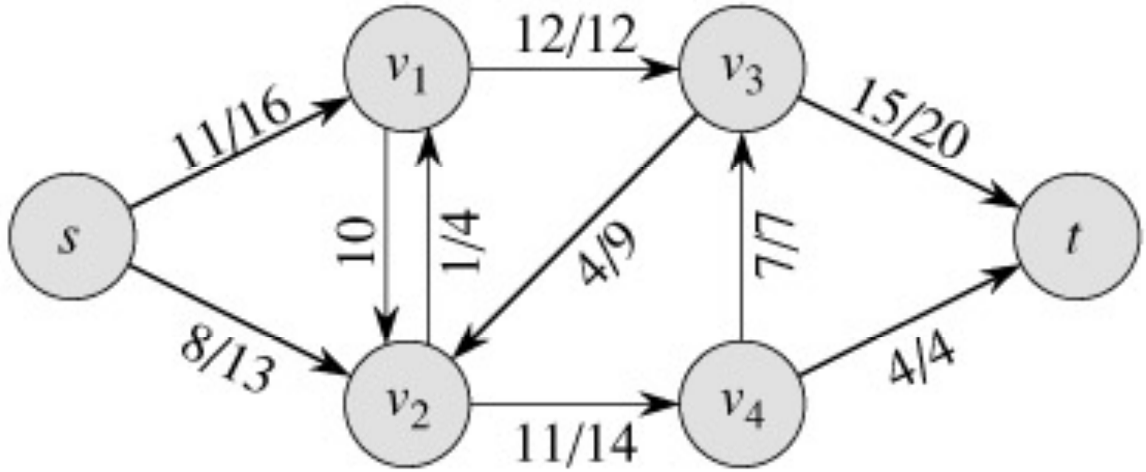



Residual Network:

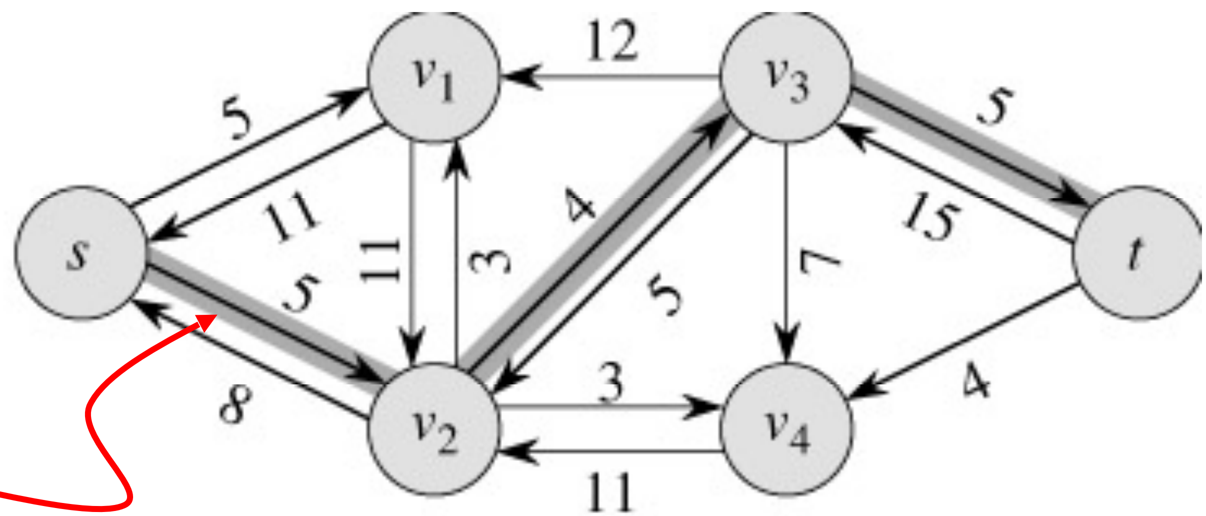


# Example of residual capacities

Network:



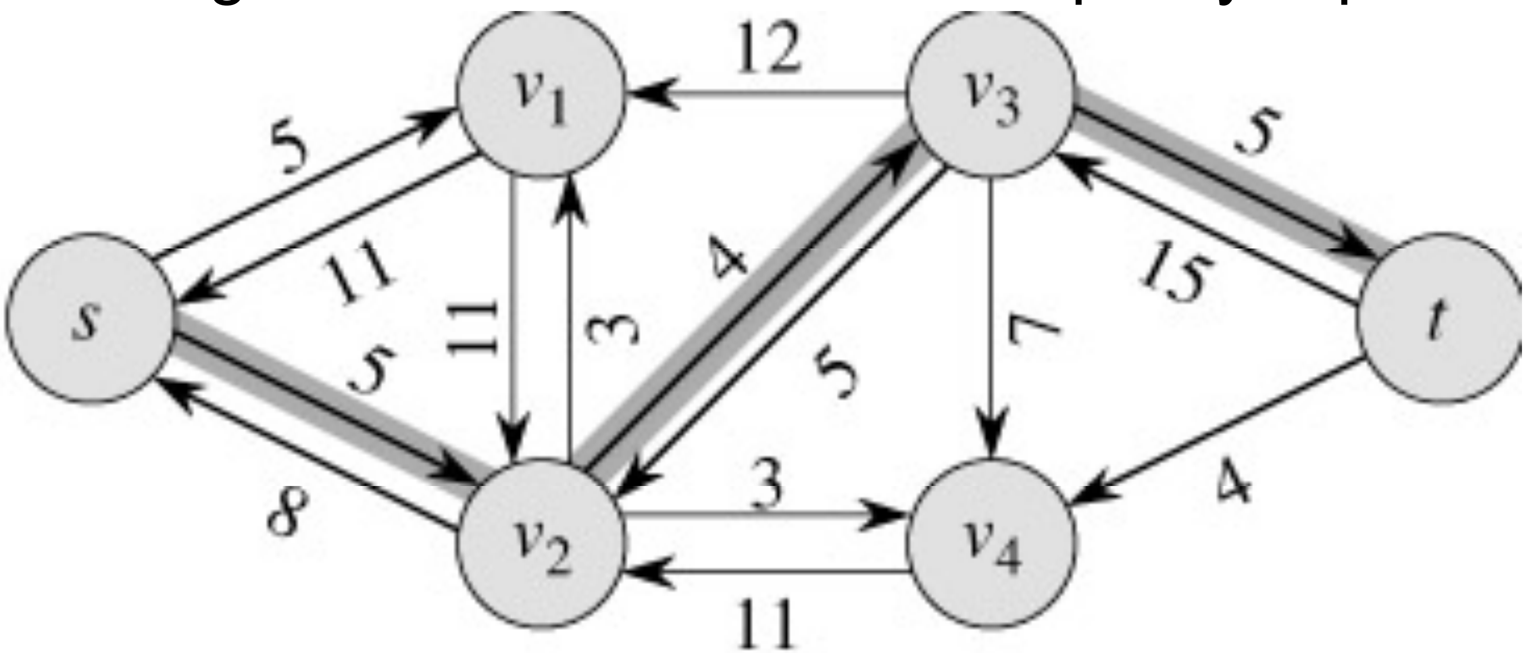

Residual Network:




Augmenting path

# The residual network

- The edges of the residual network are the edges on which the residual capacity is positive.



# Why do we need residual networks?

# Why do we need residual networks?

- Residual networks allow us to reverse flows if necessary.

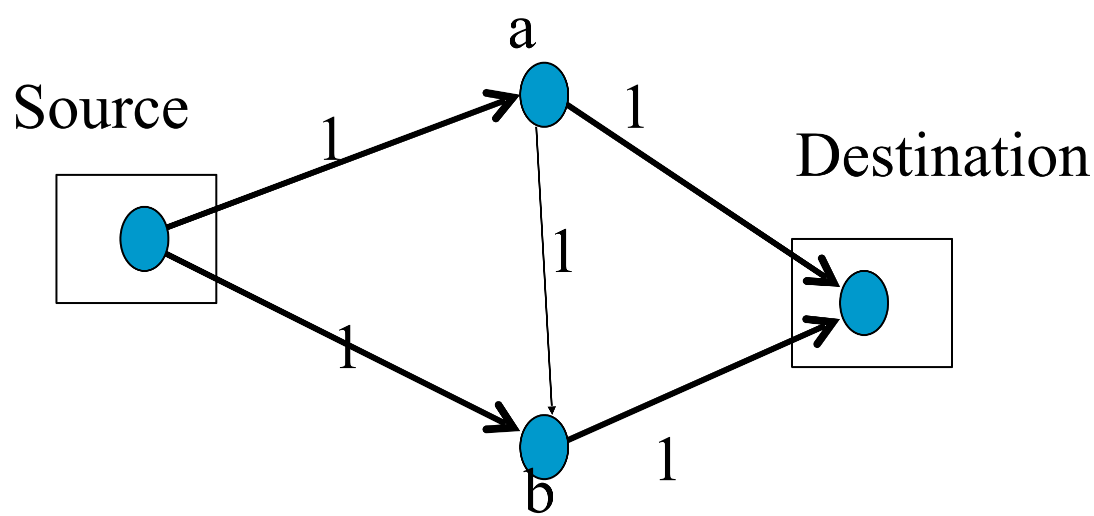
# Why do we need residual networks?

- Residual networks allow us to reverse flows if necessary.
- If we have taken a bad path then residual networks allow one to detect the condition and reverse the flow.

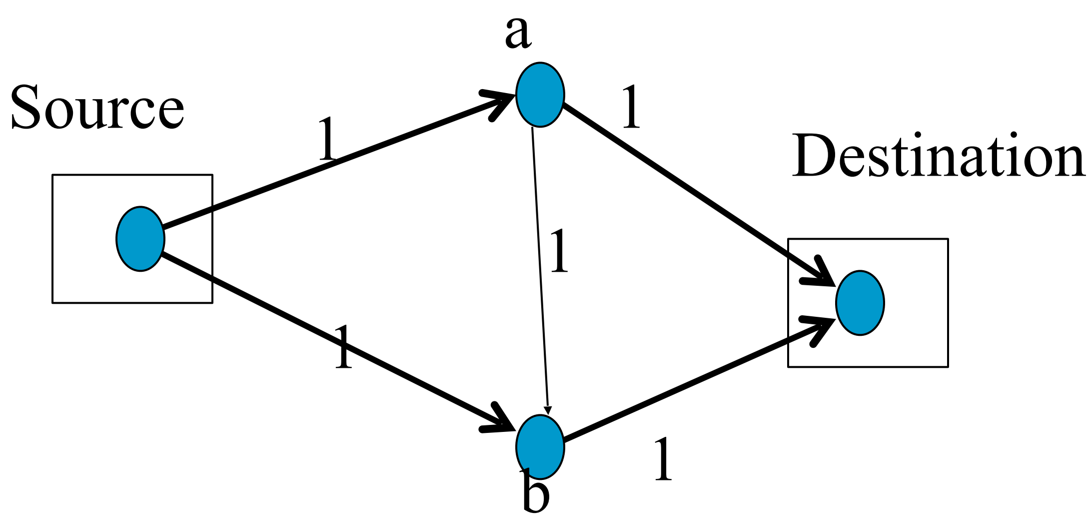
# Why do we need residual networks?

- Residual networks allow us to reverse flows if necessary.
- If we have taken a bad path then residual networks allow one to detect the condition and reverse the flow.
- A bad path is one which overlaps with too many other paths.

# Example

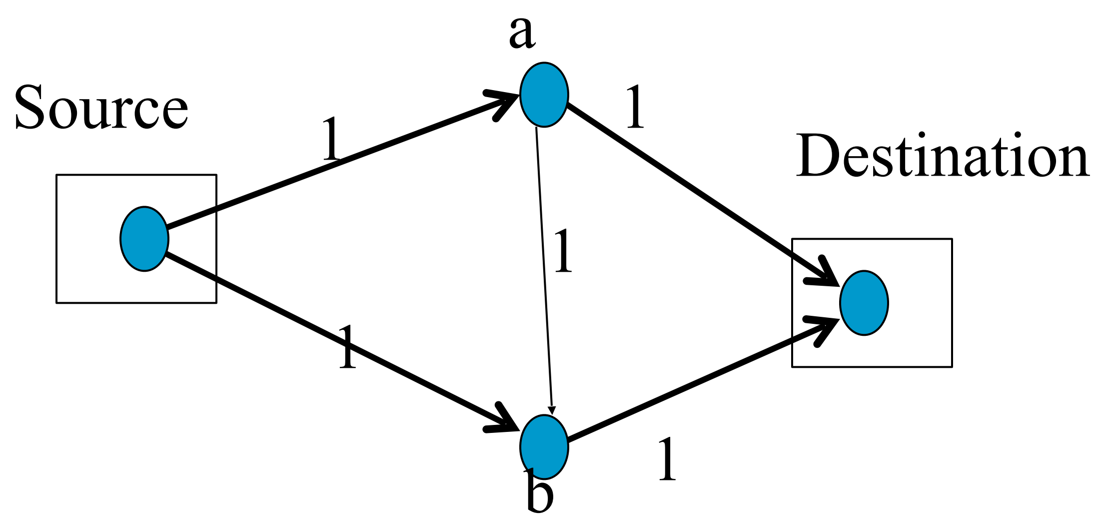


# Example




Paths source, a, destinations and source, b destination gives a flow of 2 units.

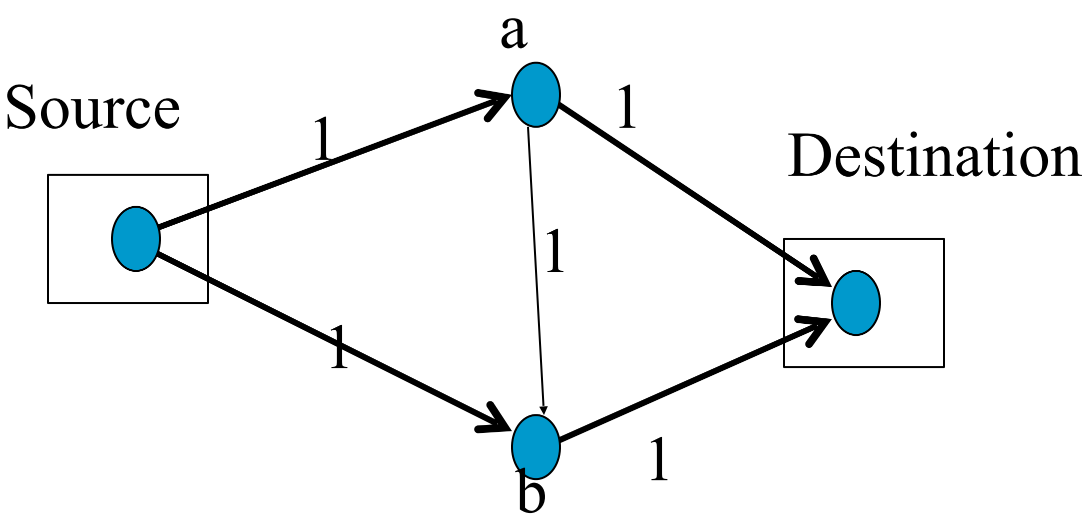
# Example




Paths source, a, destinations and source, b destination gives a flow of 2 units.

Path source, a, b, destination overlaps with both the optimal paths.

# Example

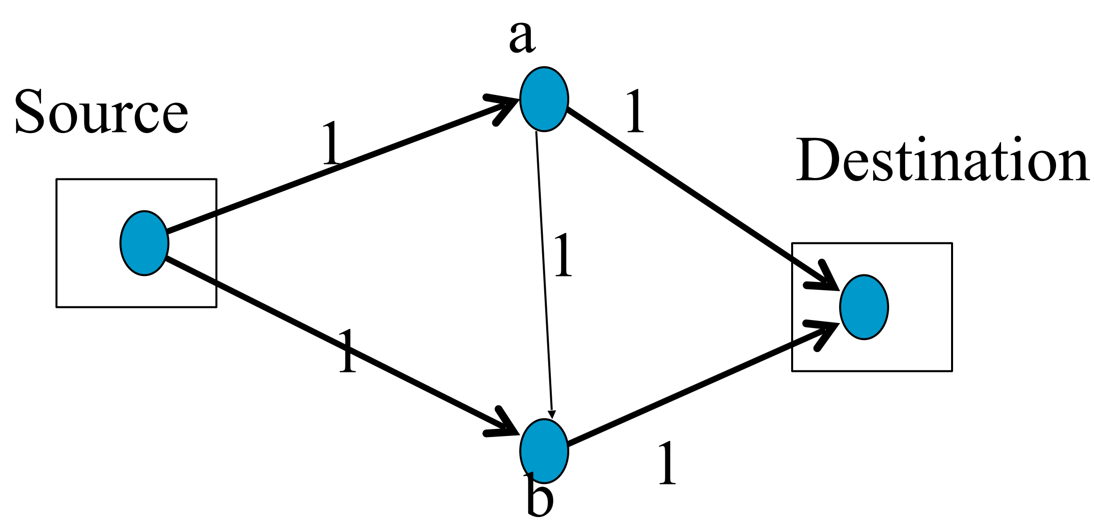



Paths source, a, destinations and source, b destination gives a flow of 2 units.

Path source, a, b, destination overlaps with both the optimal paths.

If we initially choose source, a, b, destination as our path, then no greedy strategy will be able to augment the network flow any further (unless we use residual edges which allows recovery)

# Example




Paths source, a, destinations and source, b destination gives a flow of 2 units.

Path source, a, b, destination overlaps with both the optimal paths.

If we initially choose source, a, b, destination as our path, then no greedy strategy will be able to augment the network flow any further (unless we use residual edges which allows recovery)

Verify how we recover in spite of the initial bad choice, if we use the residual network to augment flows.

# Augmenting Paths

# Augmenting Paths

- An augmenting path p is a simple path from s to t on the residual network.

# Augmenting Paths

- An augmenting path p is a simple path from s to t on the residual network.
- We can put more flow from s to t through p.

# Augmenting Paths

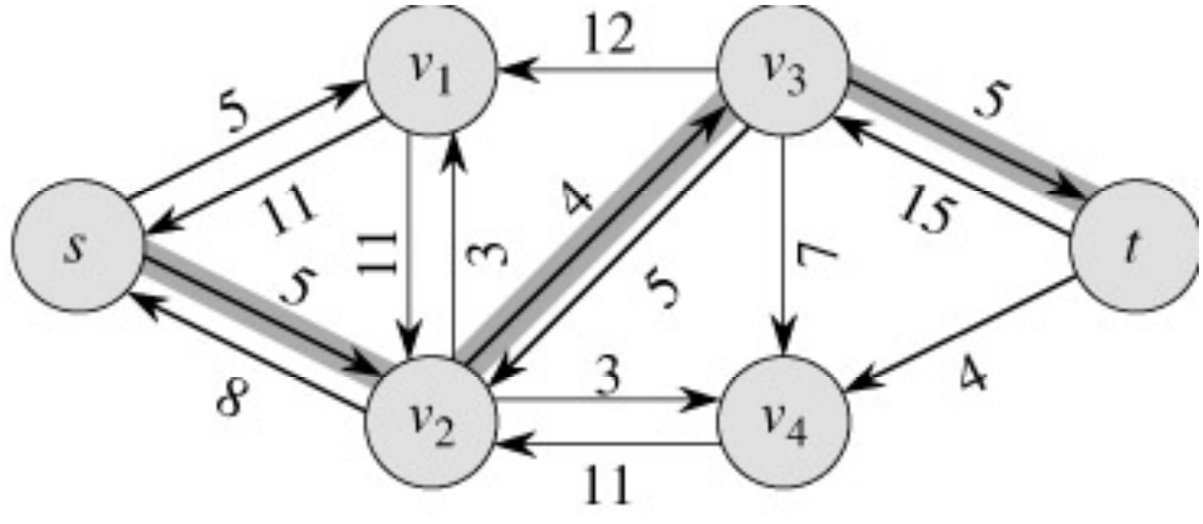
- An augmenting path p is a simple path from s to t on the residual network.
- We can put more flow from s to t through p.
- We call the maximum capacity by which we can increase the flow on p the residual capacity of p.

# Augmenting Paths

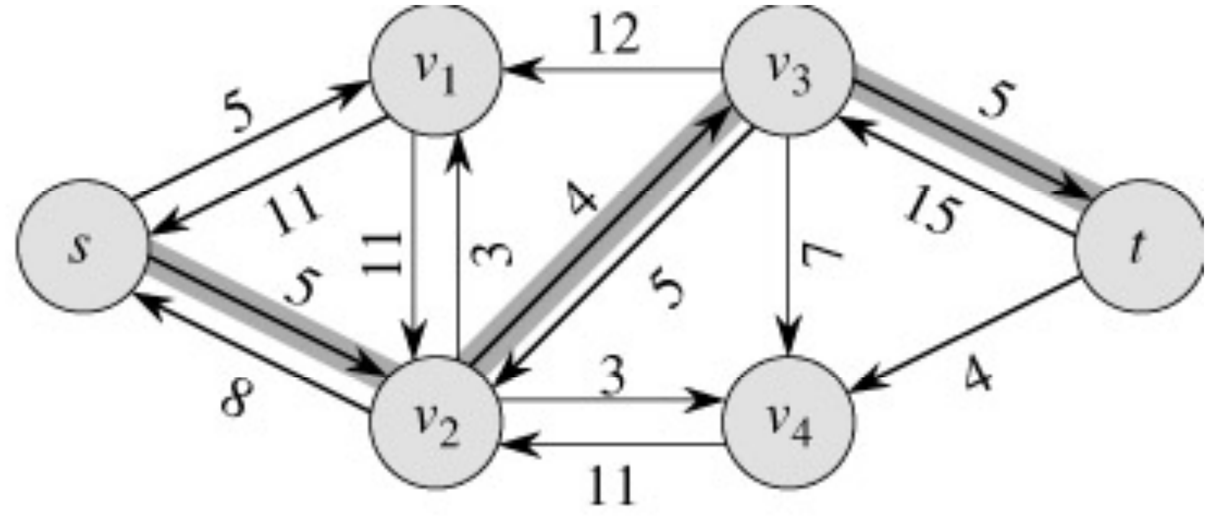
- An augmenting path p is a simple path from s to t on the residual network.
- We can put more flow from s to t through p.
- We call the maximum capacity by which we can increase the flow on p the residual capacity of p.

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

# Augmenting Paths - example

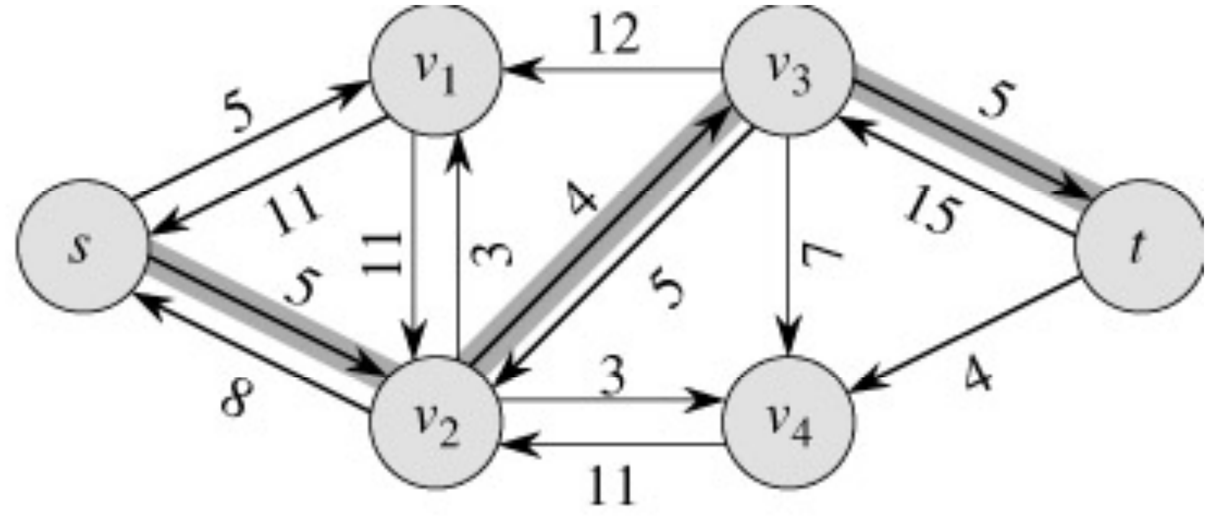


# Augmenting Paths - example




- The residual capacity of p = 4.

# Augmenting Paths - example




- The residual capacity of p = 4.
- Can improve the flow along p by 4.

# Maxflow-Mincut Theorem

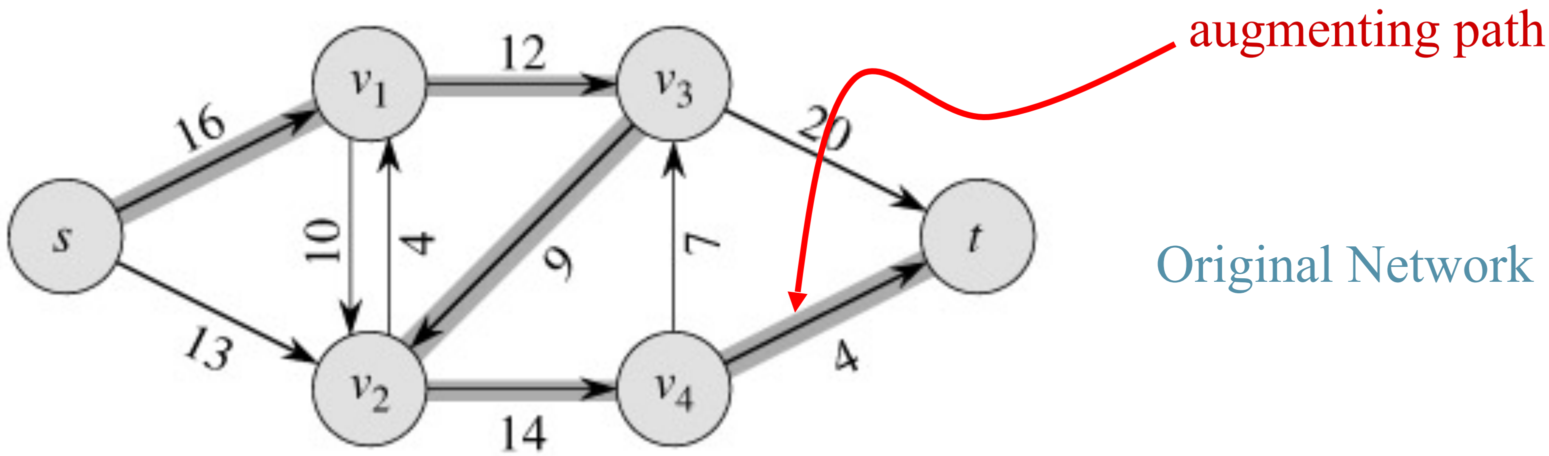
- *Max-flow min-cut theorem*:
  - If $f$ is the flow in G, the following conditions are equivalent:
    - 1. $f$ is a maximum flow in $G$
    - 2. The residual network $G_f$ contains no augmenting paths
    - 3. $|f| = c(S,T)$ for some cut $(S,T)$ of $G$

# Ford-Fulkerson Method

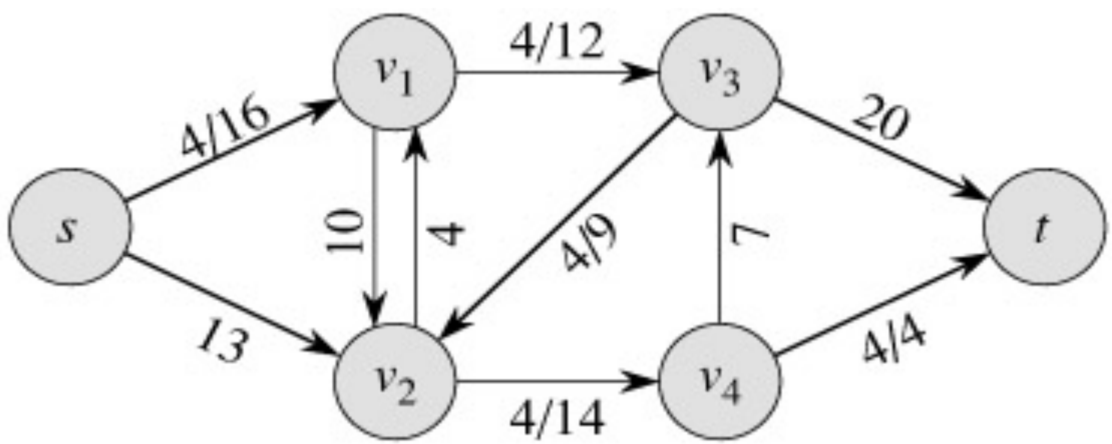
FORD-FULKERSON-METHOD$(G, s, t)$

initialize flow $f$ to 0<br>
**while** there exists an augmenting path $p$<br>
    **do** augment flow $f$ along $p$<br>
**return** $f$

# Example

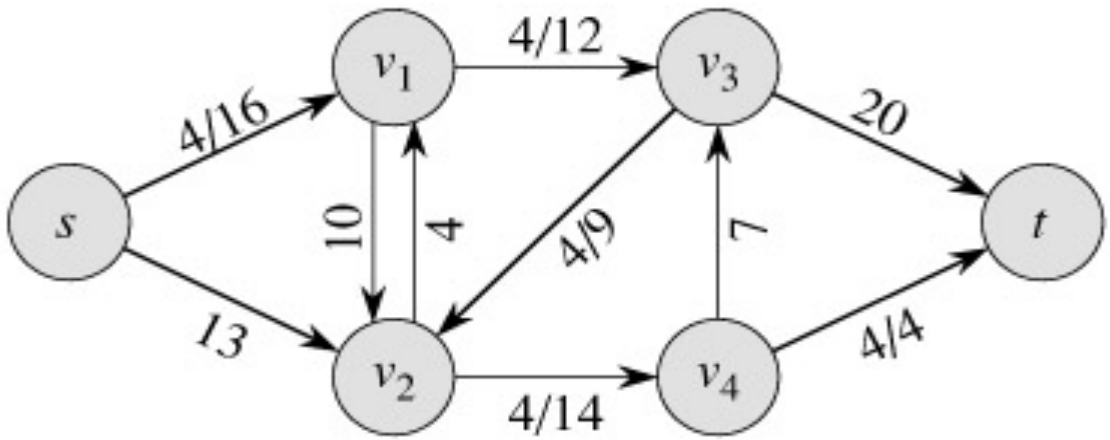



Original Network



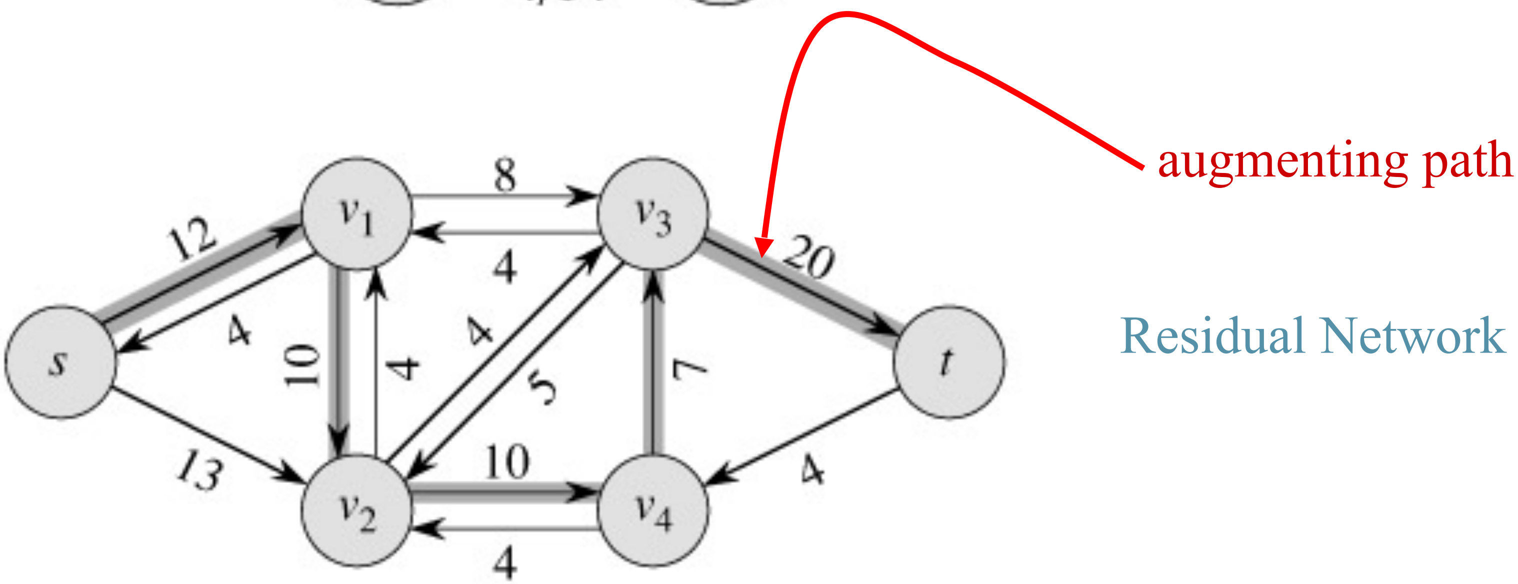

Resulting Flow =4

# Example



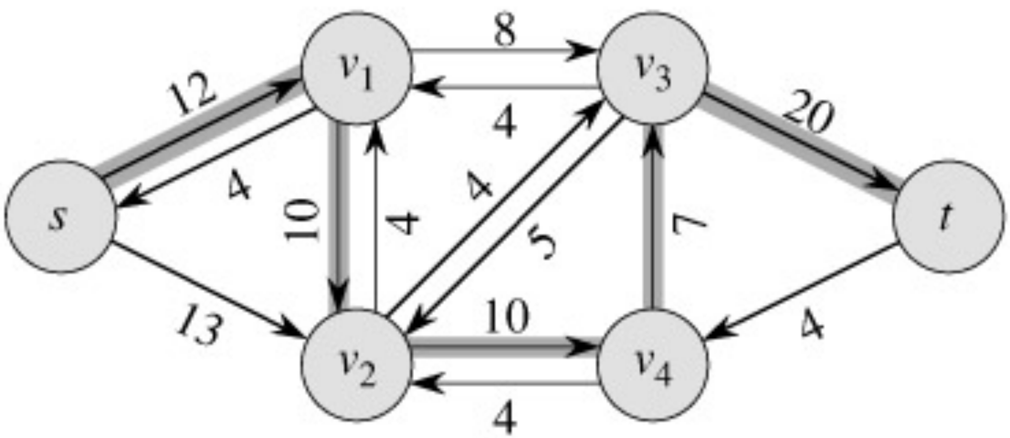

Resulting

Flow = 4

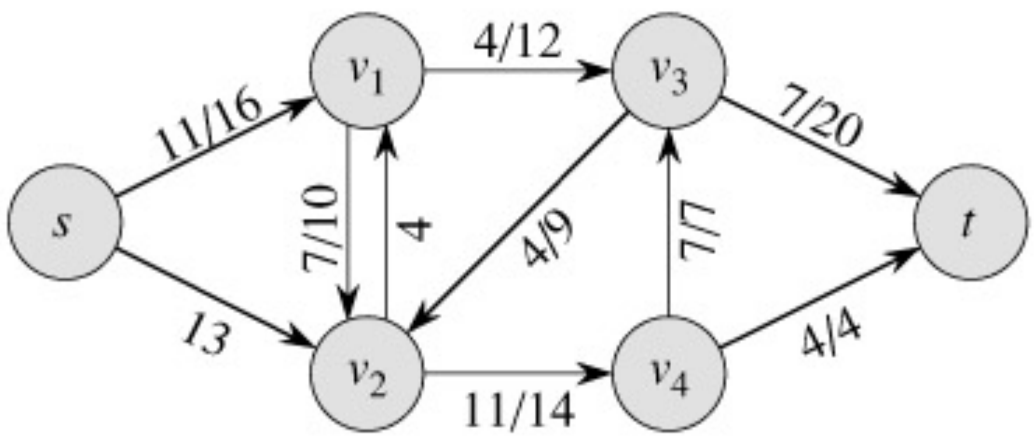



Residual Network

# Example

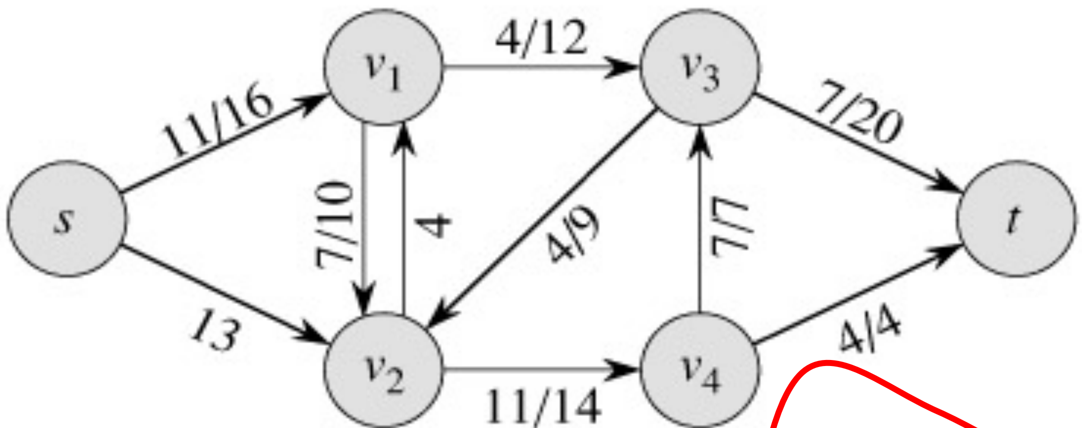



Residual Network




Resulting Flow =11

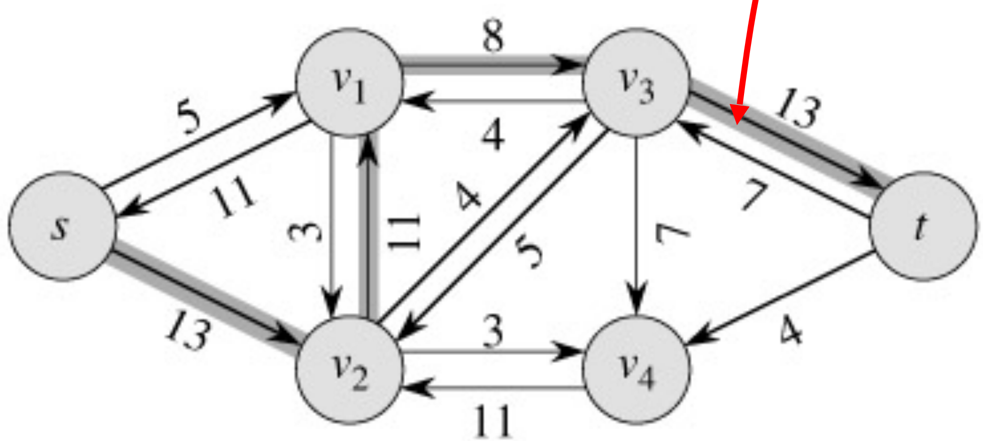
# Example




Resulting

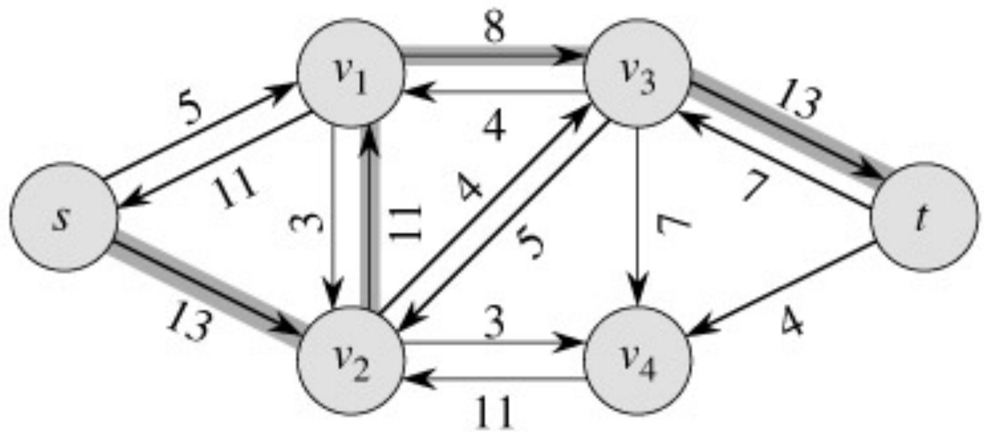
Flow =11

augmenting path

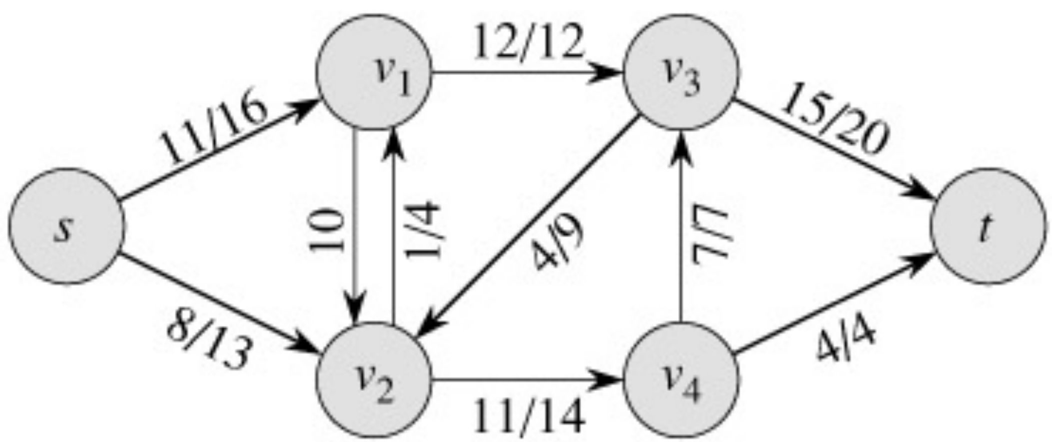



Residual Network

# Example



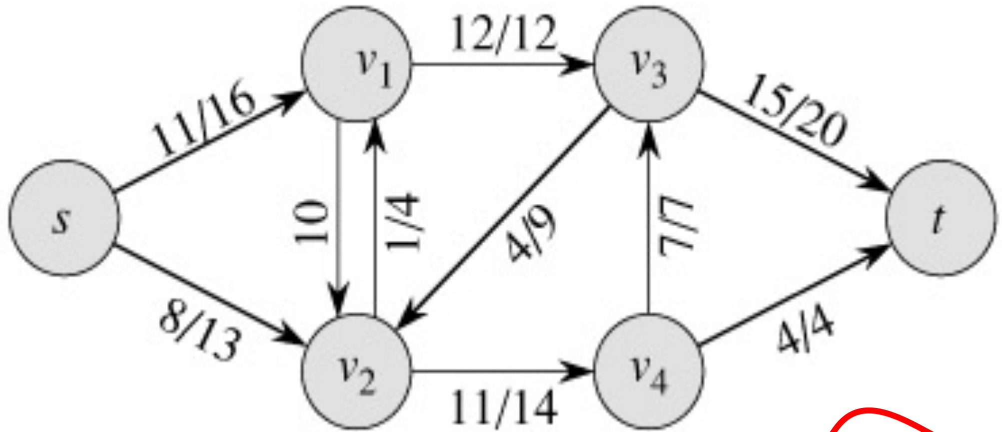

Residual Network




Resulting

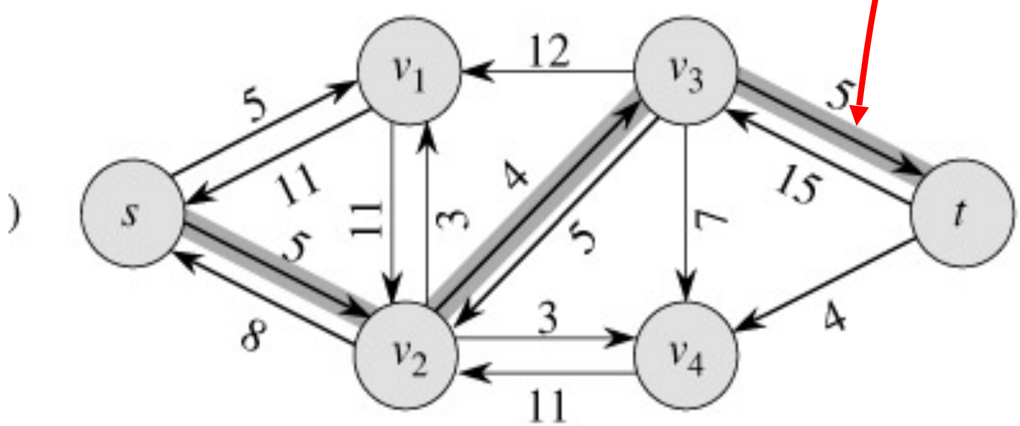
Flow =19

# Example




Resulting

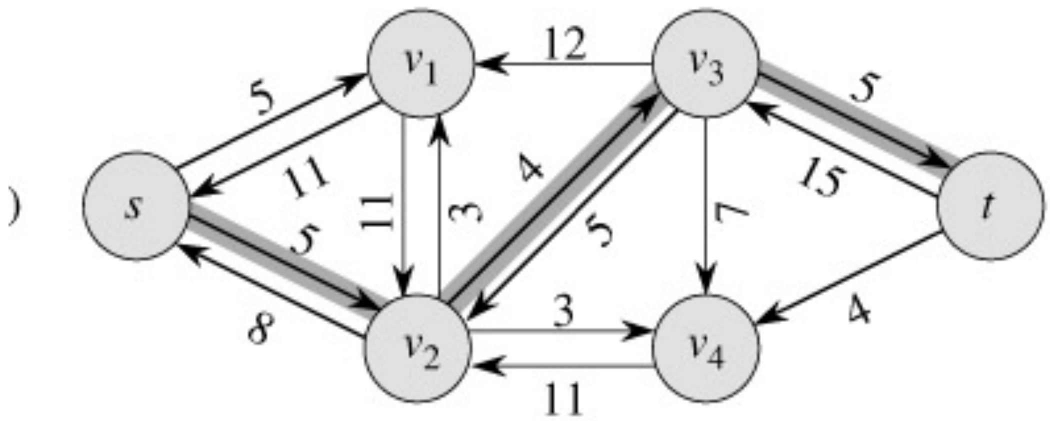
Flow = 19



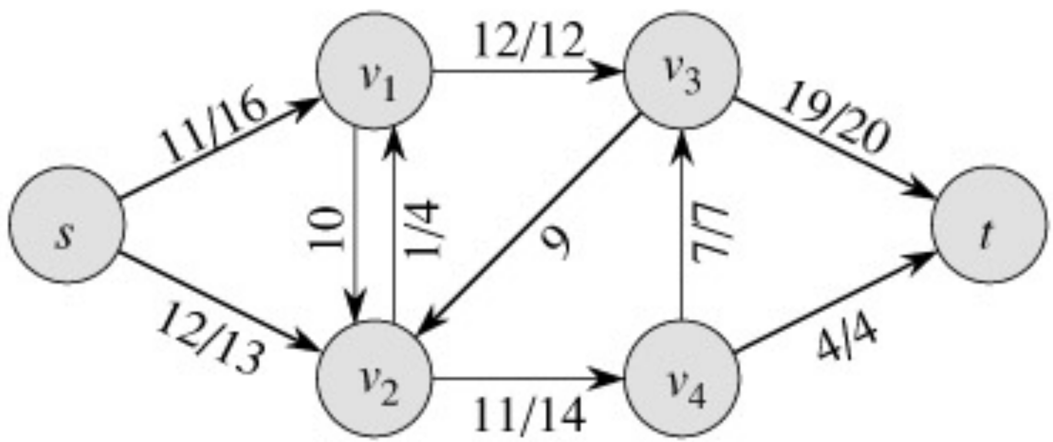

augmenting path

Residual Network

# Example



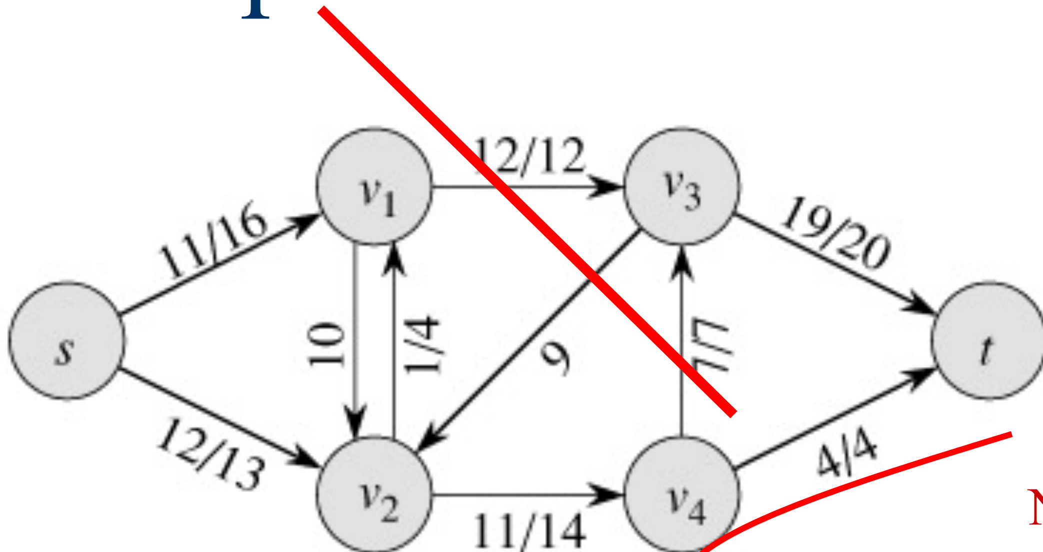

Residual Network




Resulting

Flow = 23

# Example



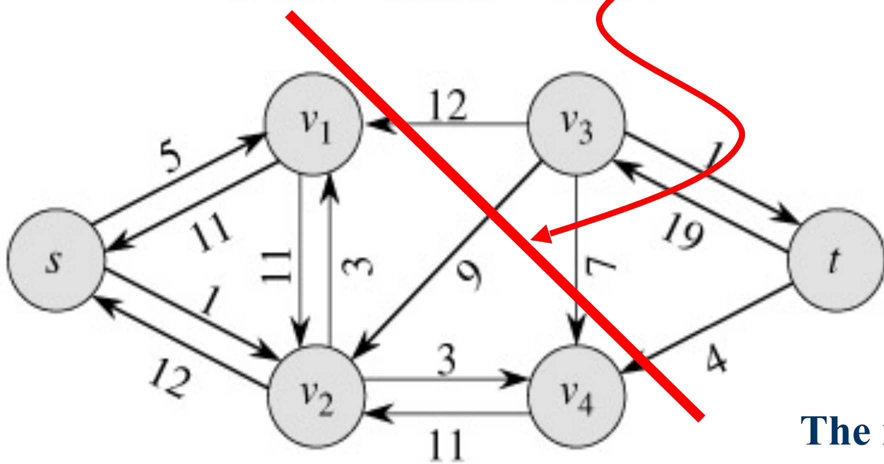

Resulting

Flow = 23

No augmenting path:

Maxflow=23

(e)




Residual Network

**The residual network Gf contains no augmented paths. So f is a maximum flow in G.**

# Ford-Fulkerson method, with details

```
Ford-Fulkerson(G,s,t)
for each edge (u,v)∈G.E do
   f(u,v) = f(v,u) = 0
while ∃ path p from s to t in residual network G_f do
   c_f = min{c_f(u,v): (u,v)∈p}
   for each edge (u,v) in p do
      f(u,v) = f(u,v) + c_f
      f(v,u) = -f(u,v)
return f
```

# Ford-Fulkerson method, with details

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

The algorithms based on this method differ in how they choose *p* in step 3.

# Time Analysis I

- A complete analysis establishing which specific method is best is a complex task, however, because their running times depend on
  - The number of augmenting paths needed to find a maxflow
  - The time needed to find each augmenting path

# Analysis

FORD-FULKERSON$(G, s, t)$

```
for each edge (u, v) ∈ E[G]
        do f[u, v] ← 0
           f[v, u] ← 0
while there exists a path p from s to t in the residual network G_f
        do c_f(p) ← min {c_f(u, v) : (u, v) is in p}
           for each edge (u, v) in p
                  do f[u, v] ← f[u, v] + c_f(p)
                     f[v, u] ← −f[u, v]
```

# Analysis

```
FORD-FULKERSON(G, s, t)
for each edge (u, v) ∈ E[G]
        do f[u, v] ← 0
           f[v, u] ← 0
while there exists a path p from s to t in the residual network G_f
        do c_f(p) ← min {c_f(u, v) : (u, v) is in p}
           for each edge (u, v) in p
                do f[u, v] ← f[u, v] + c_f(p)
                   f[v, u] ← −f[u, v]
```

O(E)

# Analysis

```
FORD-FULKERSON(G, s, t)
for each edge (u, v) ∈ E[G]
     do f[u, v] ← 0                                   O(E)
        f[v, u] ← 0
while there exists a path p from s to t in the residual network G_f
     do c_f(p) ← min {c_f(u, v) : (u, v) is in p}
        for each edge (u, v) in p
            do f[u, v] ← f[u, v] + c_f(p)                    O(E)
               f[v, u] ← −f[u, v]
```

# Analysis

# Analysis

- If capacities are all integer, then each augmenting path raises $|f|$ by $\geq 1$.

# Analysis

- If capacities are all integer, then each augmenting path raises $|f|$ by $\geq 1$.
- If max flow is $f^*$, then need $\leq |f^*|$ iterations
  - **So time is $O(E|f^*|)$.**

# Analysis

- If capacities are all integer, then each augmenting path raises $|f|$ by $\geq 1$.
- If max flow is $f^*$, then need $\leq |f^*|$ iterations
  - **So time is $O(E|f^*|)$.**
- Note that this running time is not polynomial in input size. It depends on $|f^*|$, which is not a function of $|V|$ or $|E|$.

# Analysis

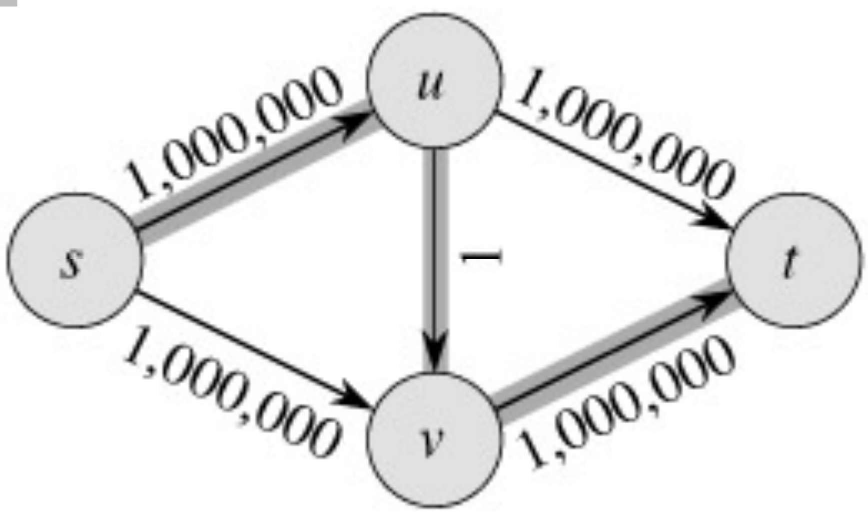
- If capacities are all integer, then each augmenting path raises $|f|$ by $\geq 1$.
- If max flow is $f^*$, then need $\leq |f^*|$ iterations
  - **So time is $O(E|f^*|)$.**
- Note that this running time is not polynomial in input size. It depends on $|f^*|$, which is not a function of $|V|$ or $|E|$.
- If capacities are rational, can scale them to integers.

# Analysis

- If capacities are all integer, then each augmenting path raises $|f|$ by $\geq 1$.
- If max flow is $f^*$, then need $\leq |f^*|$ iterations
  - **So time is $O(E|f^*|)$.**
- Note that this running time is not polynomial in input size. It depends on $|f^*|$, which is not a function of $|V|$ or $|E|$.
- If capacities are rational, can scale them to integers.
- If irrational, FORD-FULKERSON might never terminate!

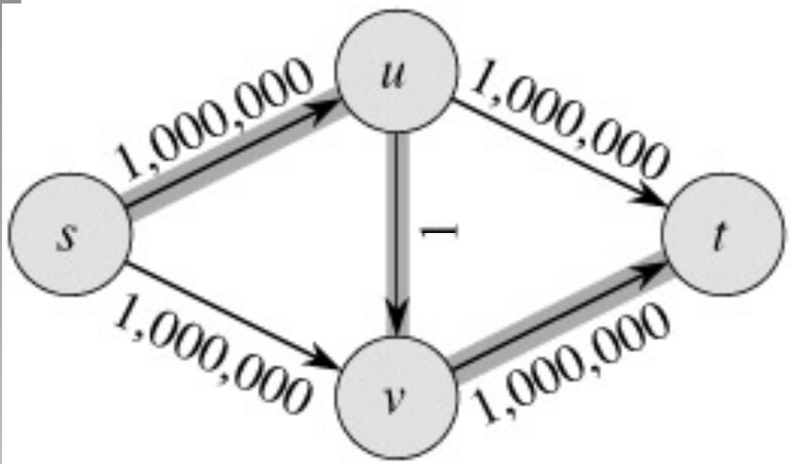
# The basic Ford-Fulkerson Algorithm

- With time O ( E |f*|), the algorithm is not polynomial.
- This problem is real: Ford-Fulkerson may perform very badly if we are unlucky:

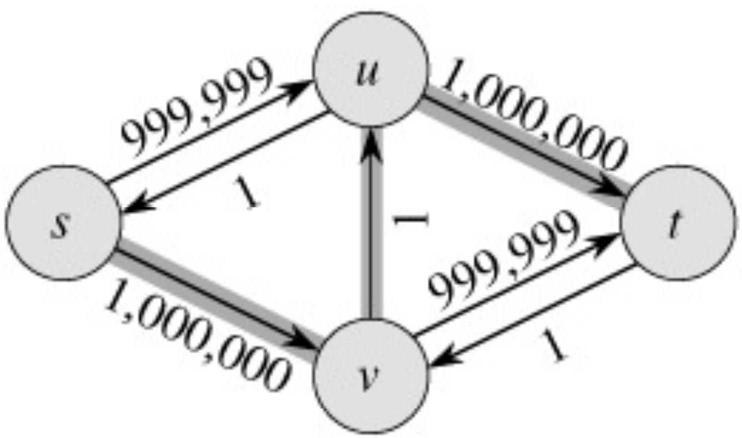



|f*|=2,000,000

# Run Ford-Fulkerson on this example

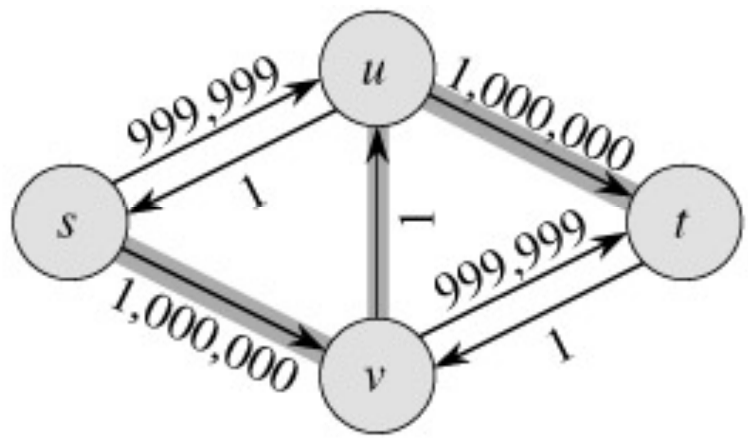



Augmenting Path

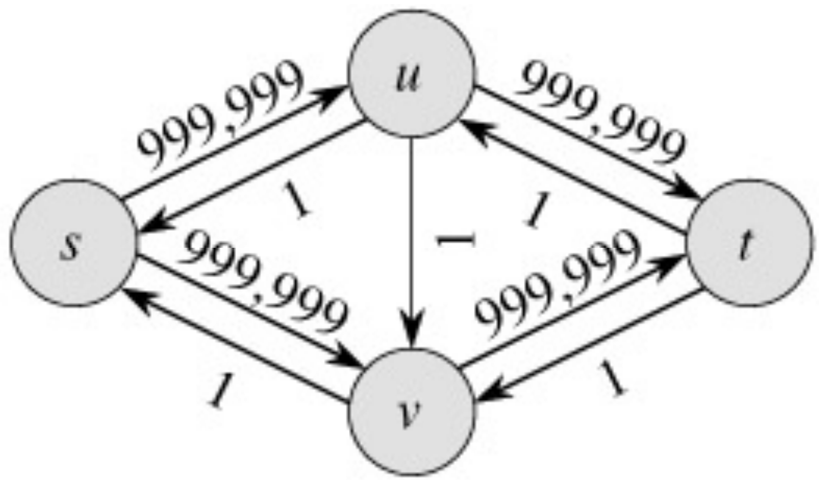



Residual Network

# Run Ford-Fulkerson on this example

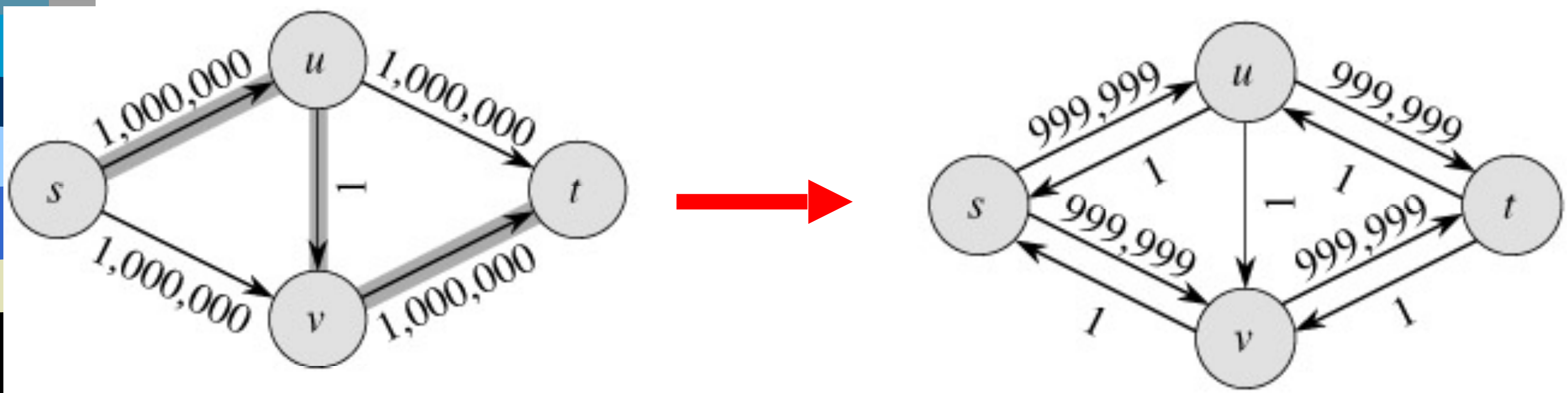



Augmenting Path




Residual Network

# Run Ford-Fulkerson on this example




- Repeat 999,999 more times…

# The Edmonds-Karp Algorithm

FORD-FULKERSON$(G, s, t)$

**for** each edge $(u, v) \in E[G]$
  **do** $f[u, v] \leftarrow 0$
    $f[v, u] \leftarrow 0$
**while** there exists a path $p$ from $s$ to $t$ in the residual network $G_f$
  **do** $c_f(p) \leftarrow \min\{c_f(u, v) : (u, v) \text{ is in } p\}$
    **for** each edge $(u, v)$ in $p$
      **do** $f[u, v] \leftarrow f[u, v] + c_f(p)$
        $f[v, u] \leftarrow -f[u, v]$

# The Edmonds-Karp Algorithm

- A small fix to the Ford-Fulkerson algorithm makes it work in polynomial time.

FORD-FULKERSON$(G, s, t)$

**for** each edge $(u, v) \in E[G]$
    **do** $f[u, v] \leftarrow 0$
        $f[v, u] \leftarrow 0$
**while** there exists a path $p$ from $s$ to $t$ in the residual network $G_f$
    **do** $c_f(p) \leftarrow \min\{c_f(u, v) : (u, v) \text{ is in } p\}$
        **for** each edge $(u, v)$ in $p$
            **do** $f[u, v] \leftarrow f[u, v] + c_f(p)$
                $f[v, u] \leftarrow -f[u, v]$

# The Edmonds-Karp Algorithm

- A small fix to the Ford-Fulkerson algorithm makes it work in polynomial time.
- Specify how to compute the path in line 4.

FORD-FULKERSON$(G, s, t)$

1 **for** each edge $(u, v) \in E[G]$
2     **do** $f[u, v] \leftarrow 0$
3         $f[v, u] \leftarrow 0$
4 **while** there exists a path $p$ from $s$ to $t$ in the residual network $G_f$
5     **do** $c_f(p) \leftarrow \min\{c_f(u, v) : (u, v) \text{ is in } p\}$
6         **for** each edge $(u, v)$ in $p$
7             **do** $f[u, v] \leftarrow f[u, v] + c_f(p)$
8                 $f[v, u] \leftarrow -f[u, v]$

# The Edmonds-Karp Algorithm

# The Edmonds-Karp Algorithm

- Compute the path in line 4 using breadth-first search on residual network.

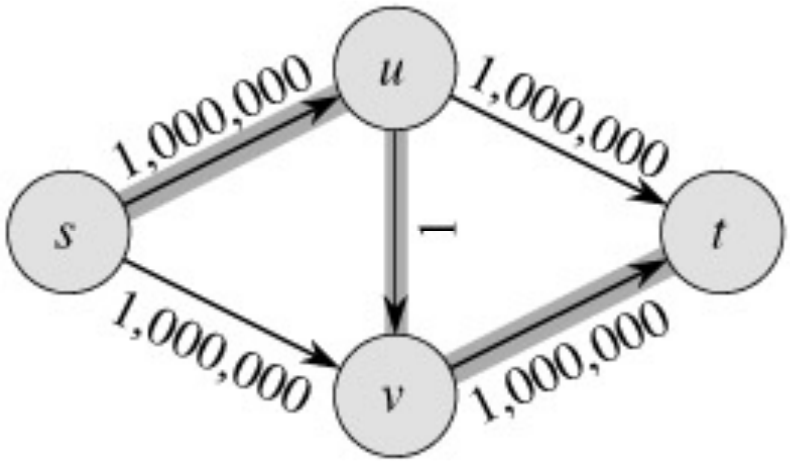
# The Edmonds-Karp Algorithm

- Compute the path in line 4 using breadth-first search on residual network.
- The augmenting path p is the shortest path from s to t in the residual network (treating all edge weights as 1).

# The Edmonds-Karp Algorithm

- Compute the path in line 4 using breadth-first search on residual network.
- The augmenting path p is the shortest path from s to t in the residual network (treating all edge weights as 1).
- Runs in time $O(V E^2)$.

# The Edmonds-Karp Algorithm - example




- Edmonds-Karp's algorithm runs only 2 iterations on this graph.

# Time Complexity

```
FORD-FULKERSON(G, s, t)
for each edge (u, v) ∈ E[G]
    do f[u, v] ← 0
       f[v, u] ← 0
while there exists a path p from s to t in the residual network G_f
    do c_f(p) ← min {c_f(u, v) : (u, v) is in p}
       for each edge (u, v) in p
           do f[u, v] ← f[u, v] + c_f(p)
              f[v, u] ← −f[u, v]
```

# Time Complexity

FORD-FULKERSON$(G, s, t)$

**for** each edge $(u, v) \in E[G]$
    **do** $f[u, v] \leftarrow 0$
        $f[v, u] \leftarrow 0$
**while** there exists a path $p$ from $s$ to $t$ in the residual network $G_f$
    **do** $c_f(p) \leftarrow \min\{c_f(u, v) : (u, v) \text{ is in } p\}$
        **for** each edge $(u, v)$ in $p$
            **do** $f[u, v] \leftarrow f[u, v] + c_f(p)$
                $f[v, u] \leftarrow -f[u, v]$

- Let, total number of flow augmentations performed by Edmonds-Karp algorithm is O(VE)

# Time Complexity

FORD-FULKERSON$(G, s, t)$

**for** each edge $(u, v) \in E[G]$
    **do** $f[u, v] \leftarrow 0$
        $f[v, u] \leftarrow 0$
**while** there exists a path $p$ from $s$ to $t$ in the residual network $G_f$
    **do** $c_f(p) \leftarrow \min\{c_f(u, v) : (u, v) \text{ is in } p\}$
        **for** each edge $(u, v)$ in $p$
            **do** $f[u, v] \leftarrow f[u, v] + c_f(p)$
                $f[v, u] \leftarrow -f[u, v]$

- Let, total number of flow augmentations performed by Edmonds-Karp algorithm is O(VE)
- BFS to find the augmented path – O(E)

# Time Complexity

```
FORD-FULKERSON(G, s, t)
for each edge (u, v) ∈ E[G]
    do f[u, v] ← 0
       f[v, u] ← 0
while there exists a path p from s to t in the residual network G_f
    do c_f(p) ← min {c_f(u, v) : (u, v) is in p}
       for each edge (u, v) in p
           do f[u, v] ← f[u, v] + c_f(p)
              f[v, u] ← −f[u, v]
```

- Let, total number of flow augmentations performed by Edmonds-Karp algorithm is O(VE)
- BFS to find the augmented path – O(E)
- So, Total running time is $O(VE^2)$

# Conditions

If f is a flow in a flow network G=(V,E), with source s and sink t, then the following conditions are equivalent:

1. f is a maximum flow in G.
2. The residual network $G_f$ contains no augmented paths.
3. $|f| = c(S,T)$ for some cut (S,T) (a min-cut).

It is a flow since there is no augmented paths It is maximum since the sink is not reachable from the source