Advanced Data Structures and Algorithms

The Maximum Network Flow Problem

Network Flow

Types of Networks

- Internet
- Telephone
- Cell
- Highways
- Rail
- Electrical Power
- Water
- Sewer
- Gas
- …

Maximum Flow Problem

- How can we maximize the flow in a network
  - from a source or set of sources
  - to a destination or set of destinations?

Network Flow

- Instance:
  - A Network is a directed graph G

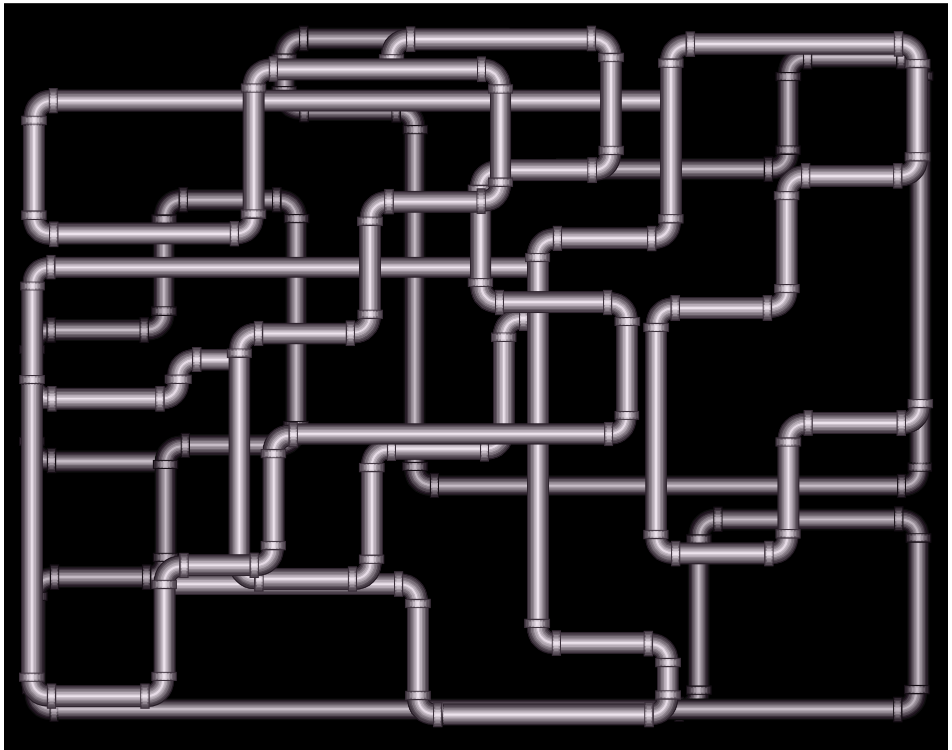


Figure courtesy of J. Edmonds

Network Flow

- Instance:
  - A Network is a directed graph G
  - Edges represent pipes that carry flow

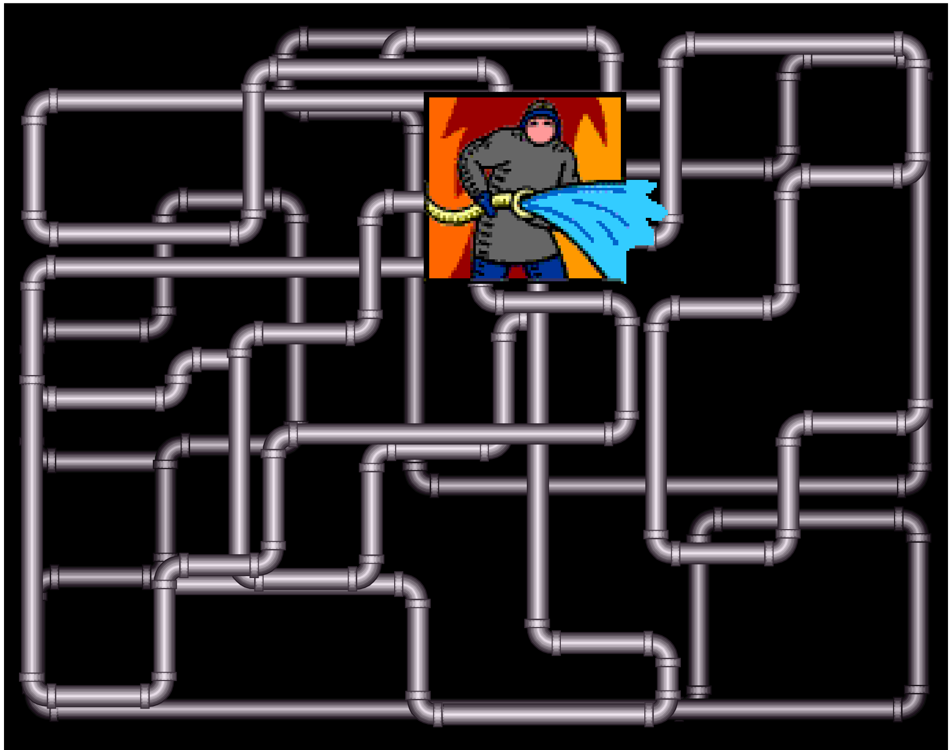


Figure courtesy of J. Edmonds

Network Flow

- Instance:
  - A Network is a directed graph G
  - Edges represent pipes that carry flow
  - Each edge (u,v) has a maximum capacity $c(u,v)$

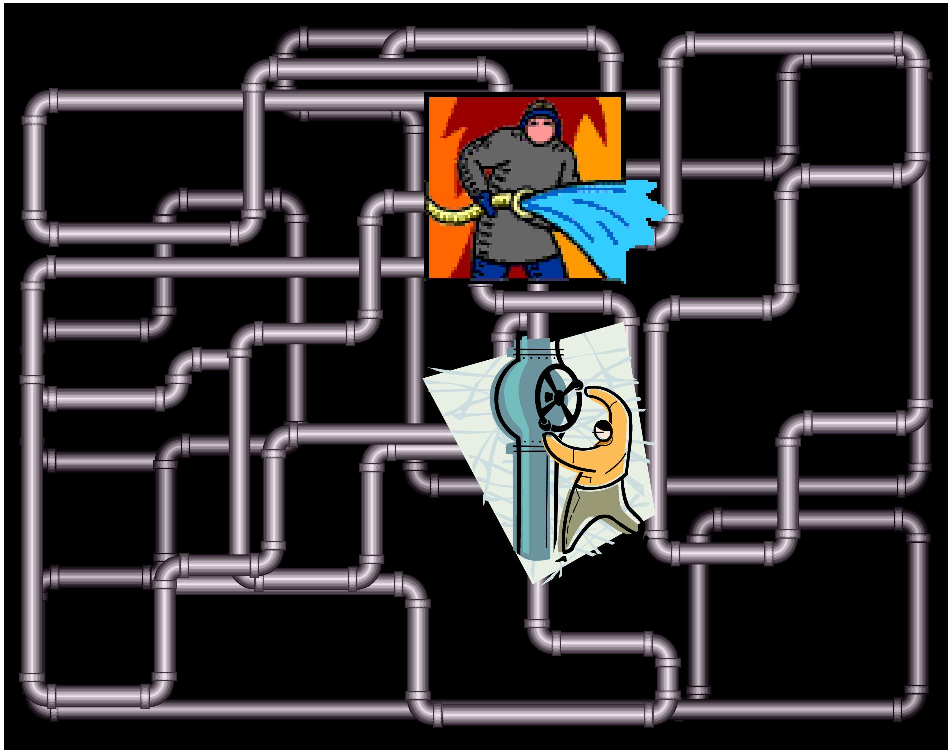


Figure courtesy of J. Edmonds

Network Flow

- Instance:
  - A Network is a directed graph G
  - Edges represent pipes that carry flow
  - Each edge (u,v) has a maximum capacity $c(u,v)$
  - A source node s in which flow arrives

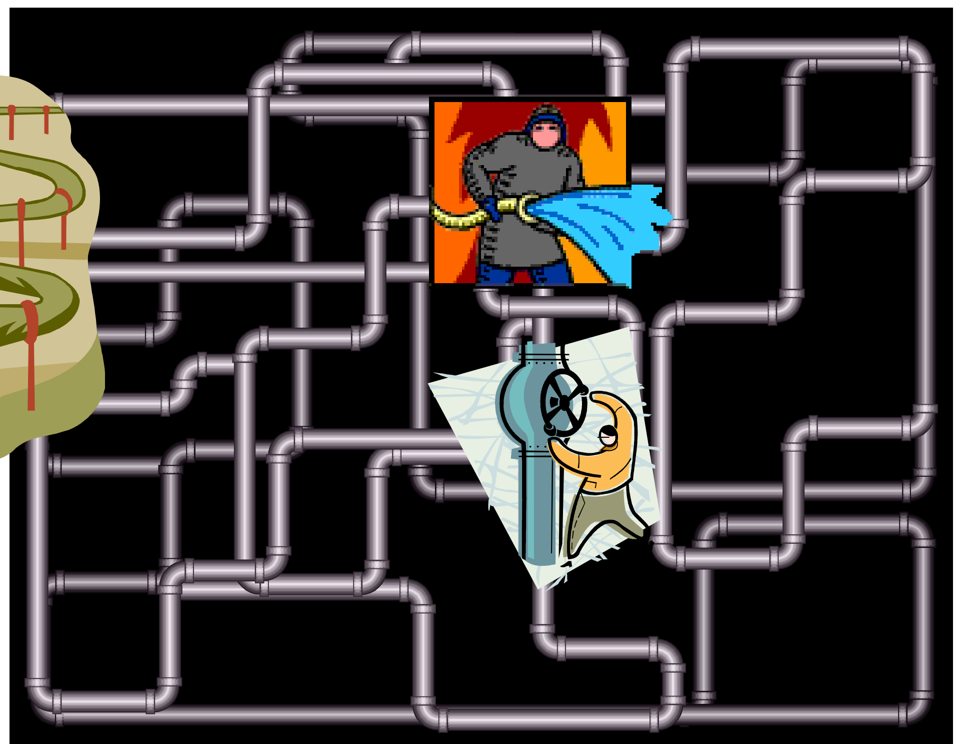


Figure courtesy of J. Edmonds

Network Flow

- Instance:
  - A Network is a directed graph G
  - Edges represent pipes that carry flow
  - Each edge (u,v) has a maximum capacity $c(u,v)$
  - A source node s in which flow arrives
  - A sink node t out which flow leaves

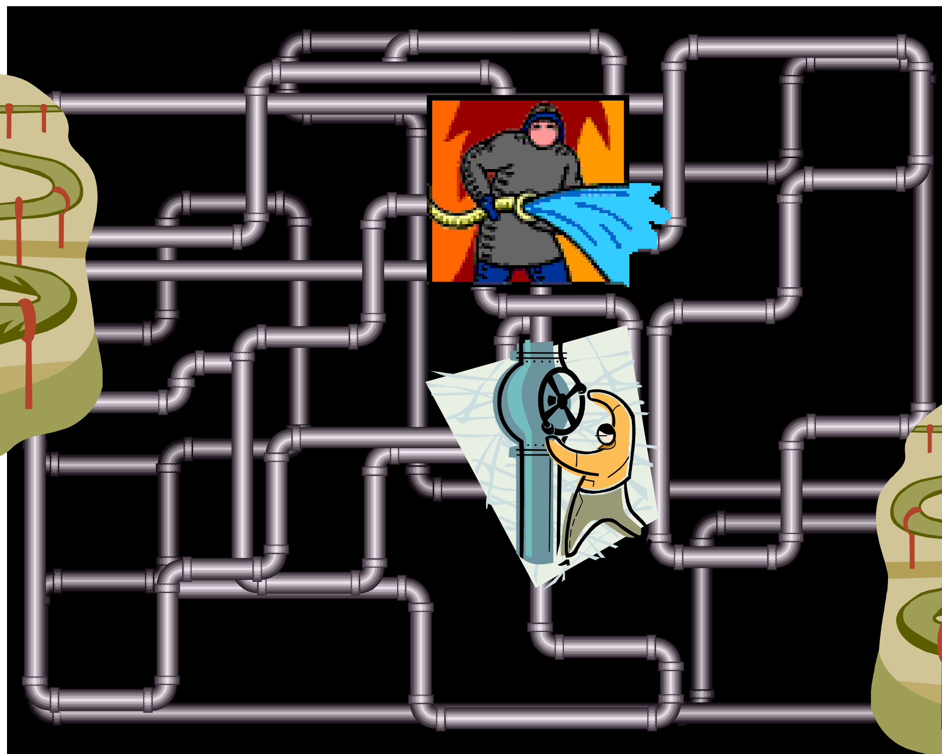




Figure courtesy of J. Edmonds

Network Flow

- Instance:
  - A Network is a directed graph G
  - Edges represent pipes that carry flow
  - Each edge (u,v) has a maximum capacity $c(u,v)$
  - A source node s in which flow arrives
  - A sink node t out which flow leaves

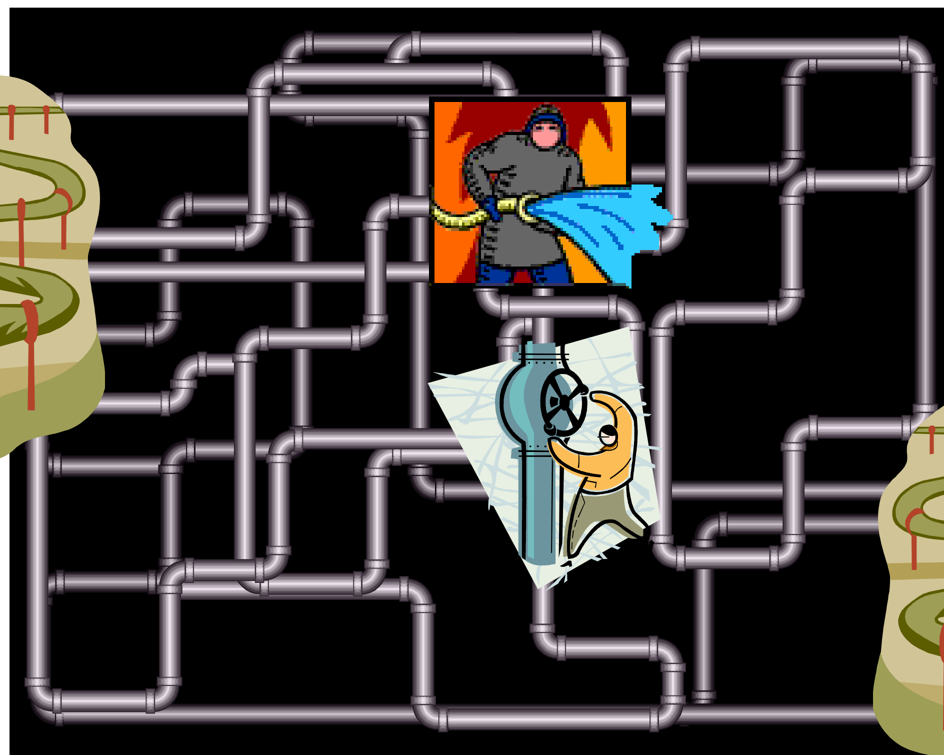
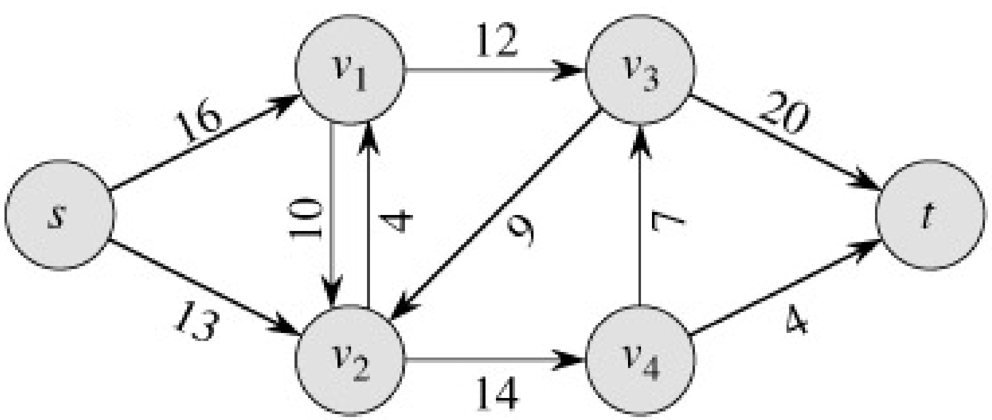


Figure courtesy of J. Edmonds

Goal:
Max Flow

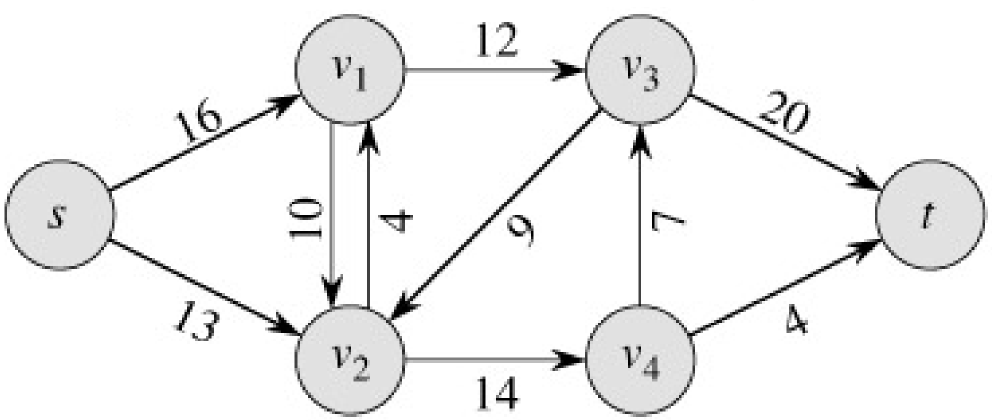
The Problem

- Use a graph to model material that flows through conduits.
- Each edge represents one conduit, and has a capacity, which is an upper bound on the flow rate, in units/time.
- Can think of edges as pipes of different sizes.
- Want to compute max rate that we can ship material from a designated source to a designated sink.



What is a Flow Network?

- 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.
- e.g., $c(s,v_1)=16$; $c(v_1,s)=0$; $c(v_2,v_3)=0$



What is a Flow Network?

- For each edge (u,v), the flow f(u,v) is a real-valued function that must satisfy 3 conditions:

What is a Flow Network?

- For each edge (u,v), the flow f(u,v) is a real-valued function that must satisfy 3 conditions:

Capacity constraint: $\forall u,v \in V,\ f(u,v) \leq c(u,v)$

What is a Flow Network?

- For each edge (u,v), the flow f(u,v) is a real-valued function that must satisfy 3 conditions:

Capacity constraint: $\forall u,v \in V,\ f(u,v) \le c(u,v)$

Skew symmetry: $\forall u,v \in V,\ f(u,v) = -f(v,u)$

What is a Flow Network?

- For each edge (u,v), the flow f(u,v) is a real-valued function that must satisfy 3 conditions:

Capacity constraint: $\forall u,v \in V,\ f(u,v) \le c(u,v)$

Skew symmetry: $\forall u,v \in V,\ f(u,v) = -f(v,u)$

Flow conservation: $\forall u \in V - \{s,t\},\ \sum_{v \in V} f(u,v) = 0$

What is a Flow Network?

- For each edge (u,v), the flow f(u,v) is a real-valued function that must satisfy 3 conditions:

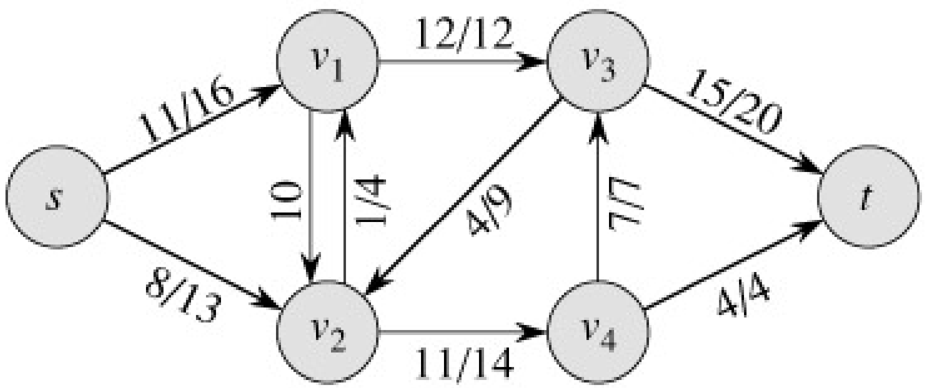
Capacity constraint: $\forall u,v \in V,\ f(u,v) \le c(u,v)$

Skew symmetry: $\forall u,v \in V,\ f(u,v) = -f(v,u)$

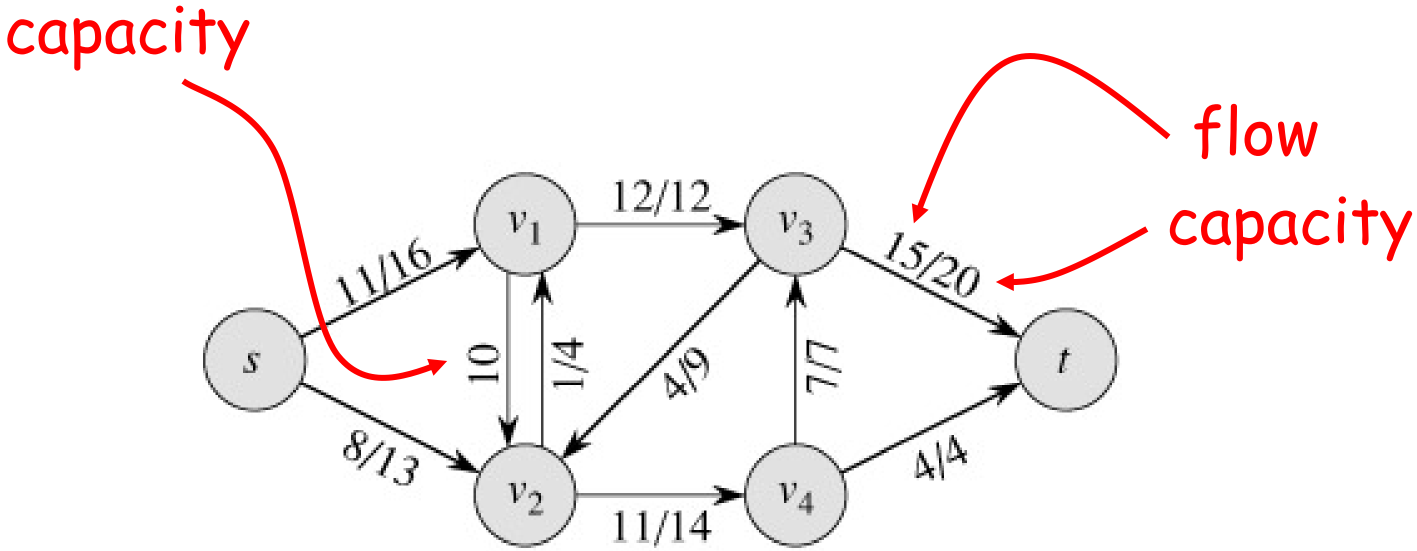
Flow conservation: $\forall u \in V - \{s,t\},\ \sum_{v \in V} f(u,v) = 0$

- Notes:
  - The skew symmetry condition implies that f(u,u)=0.
  - We show only the *positive* flows in the flow network.

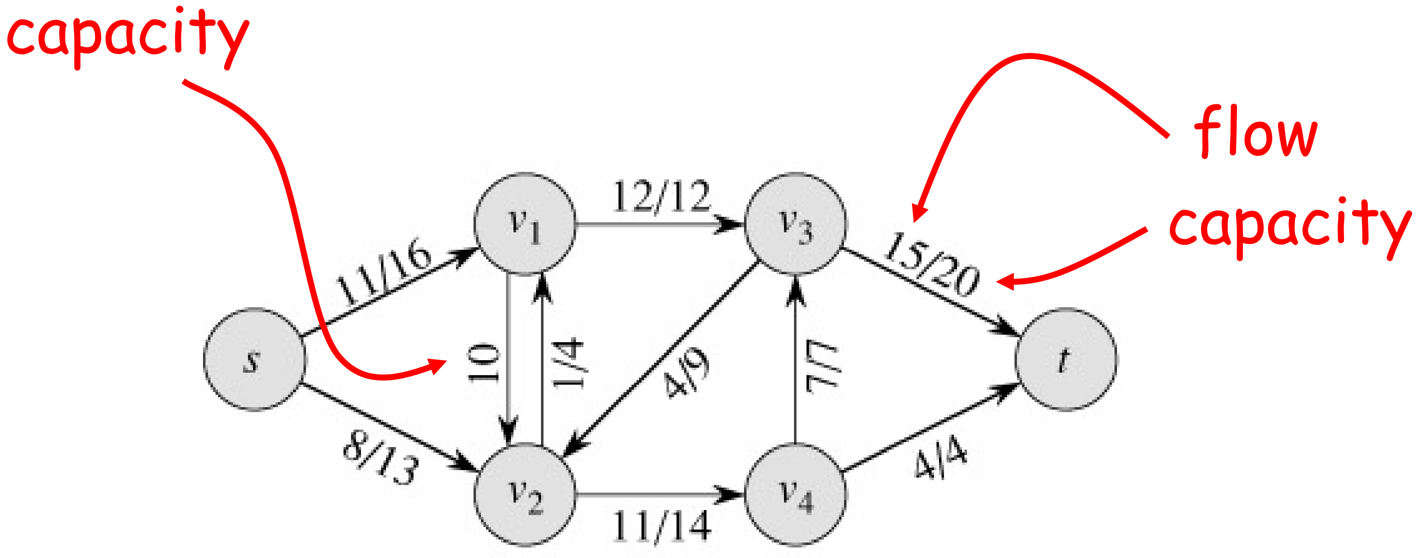
Example of a Flow:



Example of a Flow:

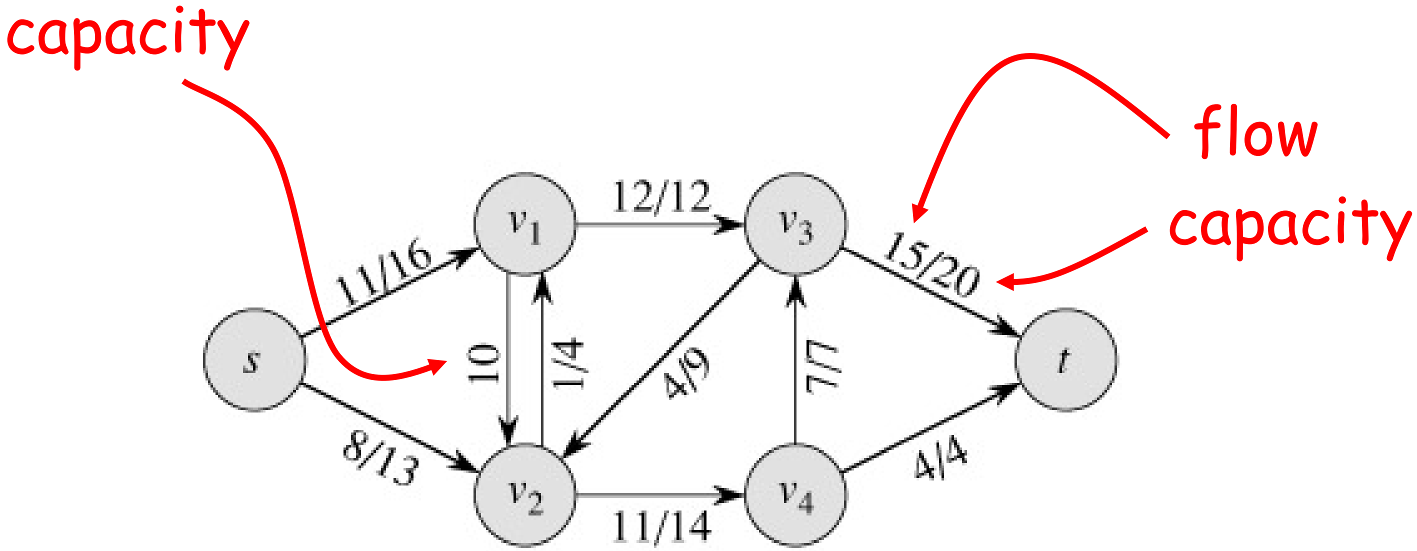


Example of a Flow:



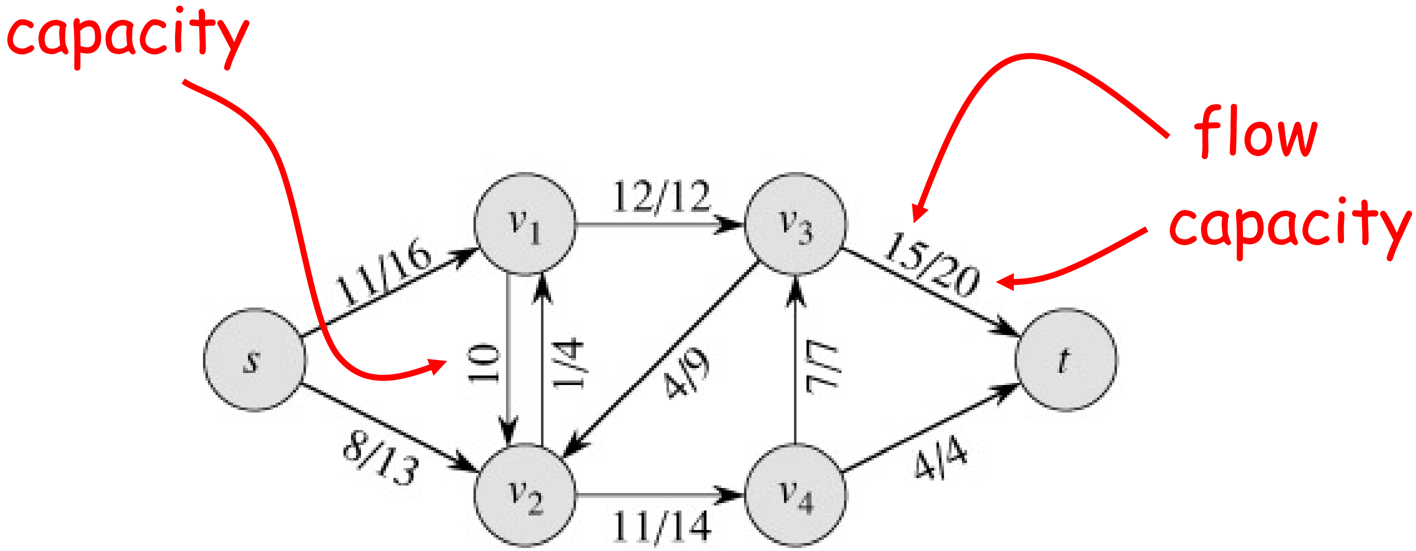
- $f(v_2, v_1) = 1$, $c(v_2, v_1) = 4$.

Example of a Flow:



- $f(v_2, v_1) = 1$, $c(v_2, v_1) = 4$.
- $f(v_1, v_2) = -1$, $c(v_1, v_2) = 10$.

Example of a Flow:



- $f(v_2, v_1) = 1$, $c(v_2, v_1) = 4$.
- $f(v_1, v_2) = -1$, $c(v_1, v_2) = 10$.
- $f(v_3, s) + f(v_3, v_1) + f(v_3, v_2) + f(v_3, v_4) + f(v_3, t) =$

  $0 \quad + \quad (-12) \quad + \quad 4 \quad + \quad (-7) \quad + \quad 15 \quad = \quad 0$

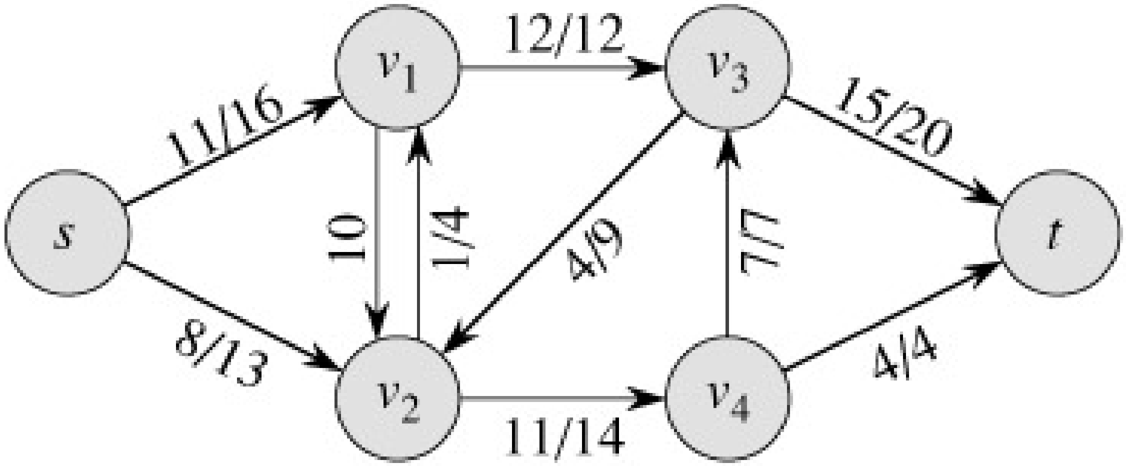
The Value of a flow

- The value of a flow is given by

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

- This is the total flow leaving s = the total flow arriving in t.

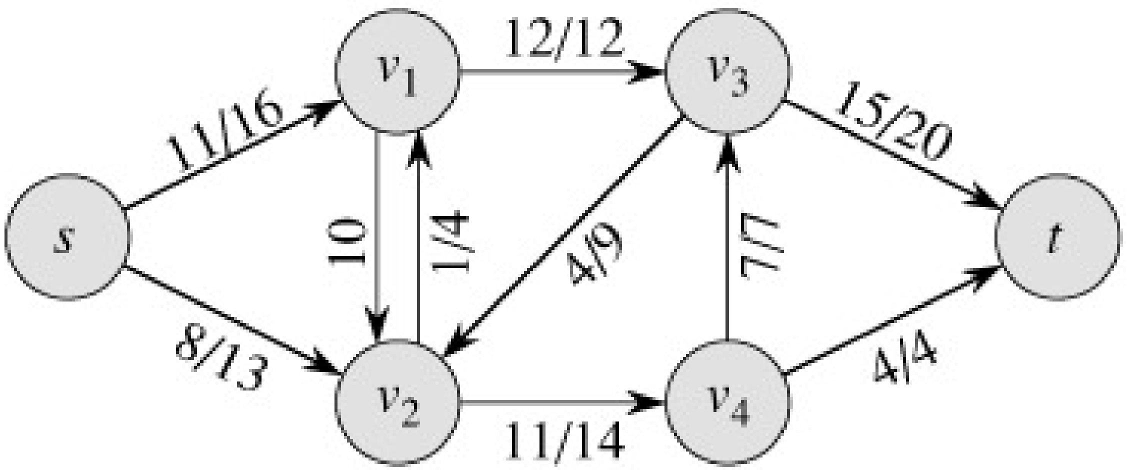
Example:



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

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

Example:

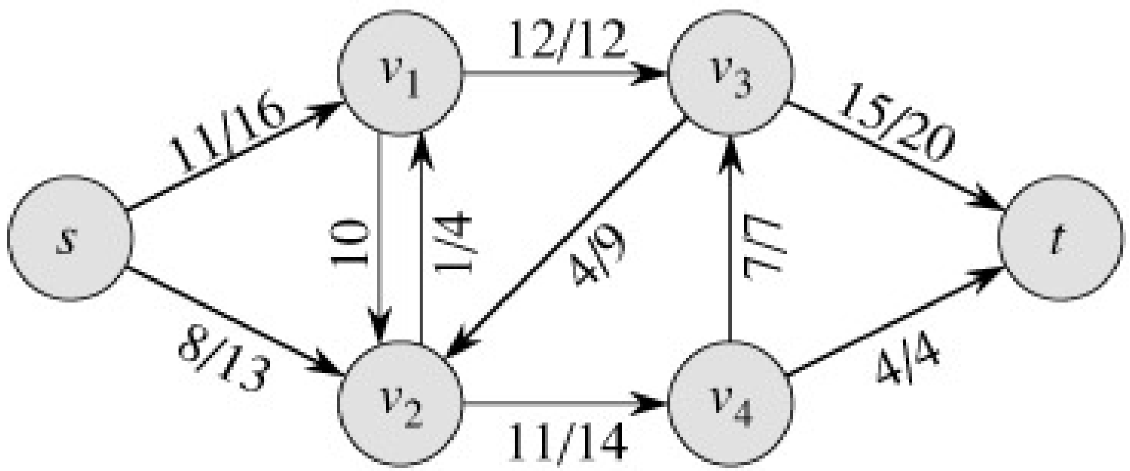


$$|f| = f(s, v_1) + f(s, v_2) + f(s, v_3) + f(s, v_4) + f(s, t) =$$
$$11 + 8 + 0 + 0 + 0 = 19$$

$$|f| = f(s, t) + f(v_1, t) + f(v_2, t) + f(v_3, t) + f(v_4, t) =$$
$$0 + 0 + 0 + 15 + 4 = 19$$

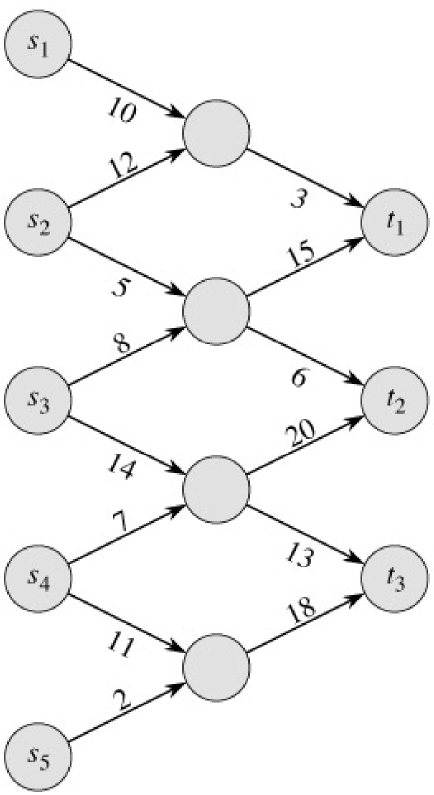
A flow in a network

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



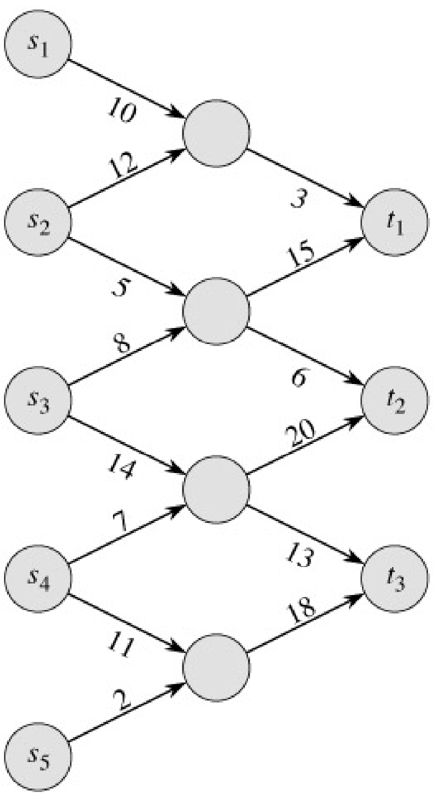
Multiple Sources Network

- We have several sources and several targets.
- Want to maximize the total flow from all sources to all targets.



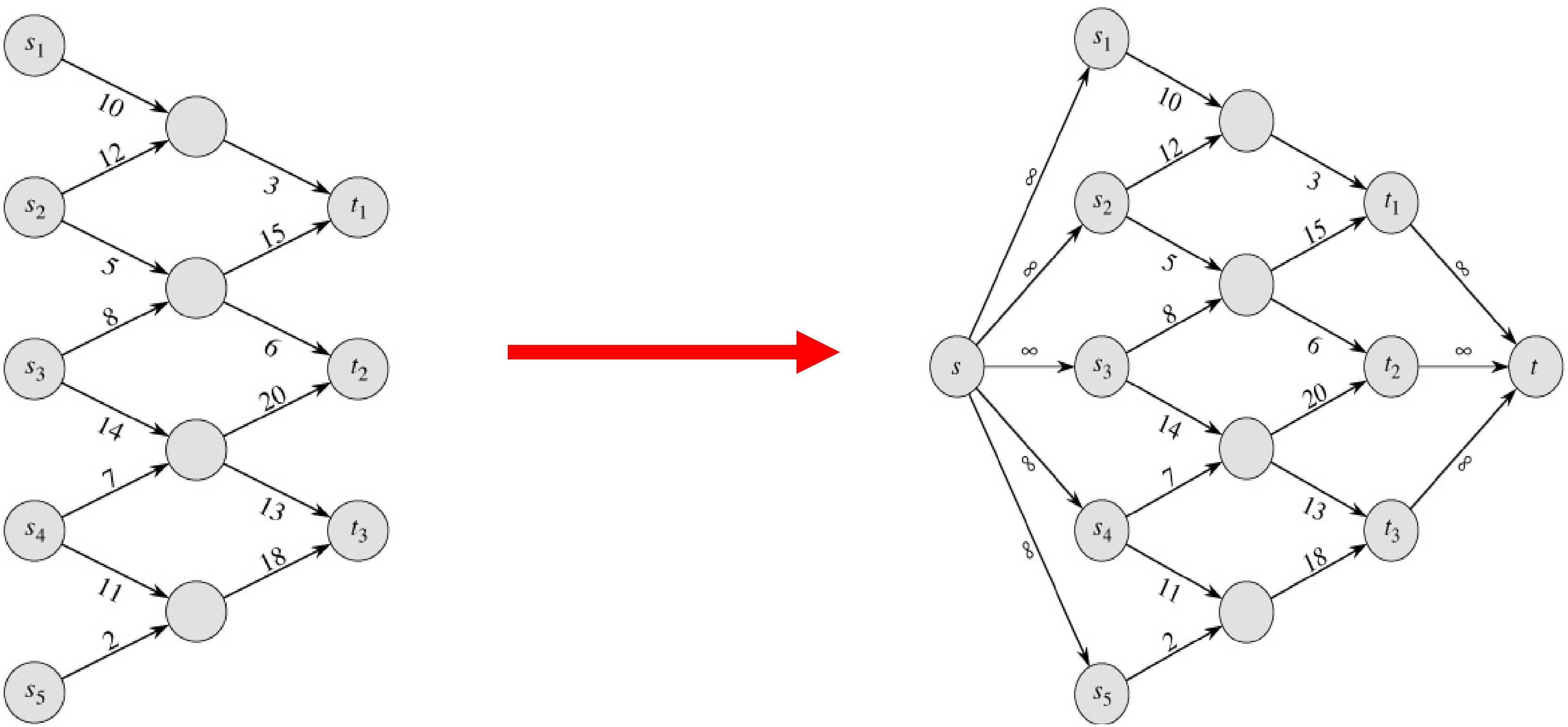
Multiple Sources Network

- We have several sources and several targets.
- Want to maximize the total flow from all sources to all targets.
- Reduce to max-flow by creating a supersource and a supersink:



Multiple Sources Network

- We have several sources and several targets.
- Want to maximize the total flow from all sources to all targets.
- Reduce to max-flow by creating a supersource and a supersink:



Residual Networks

- The residual capacity of an edge *(u,v)* in a network with a flow f is given by:

$$\boxed{c_f(u,v) = c(u,v) - f(u,v)}$$

Residual Networks

- The residual capacity of an edge *(u,v)* in a network with a flow f is given by:

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

- The residual network of a graph G induced by a flow f is the graph including only the edges with positive residual capacity, i.e.,

Residual Networks

- The residual capacity of an edge (u,v) in a network with a flow f is given by:

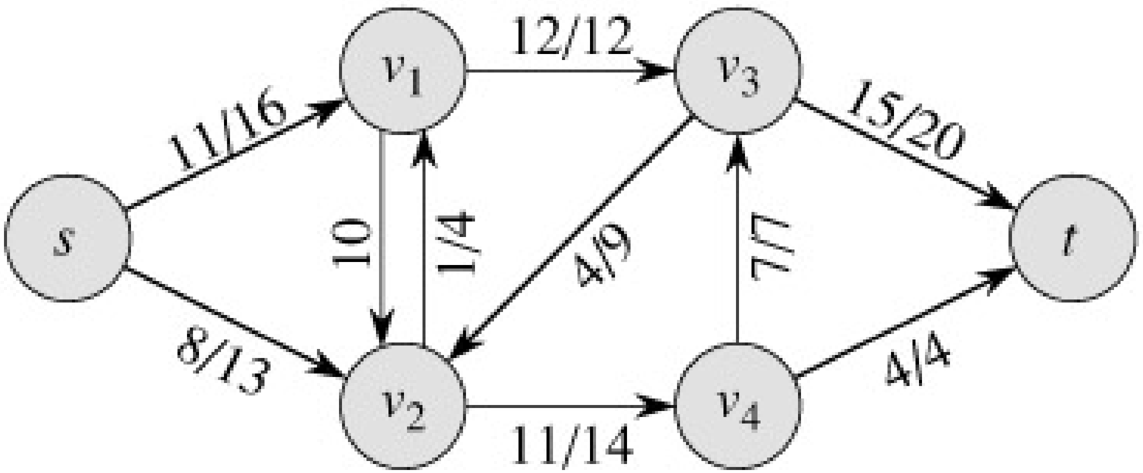
$$\boxed{c_f(u,v) = c(u,v) - f(u,v)}$$

- The residual network of a graph G induced by a flow f is the graph including only the edges with positive residual capacity, i.e.,

$$G_f = (V, E_f), \text{ where } E_f = \{(u,v) \in V \times V : c_f(u,v) > 0\}$$

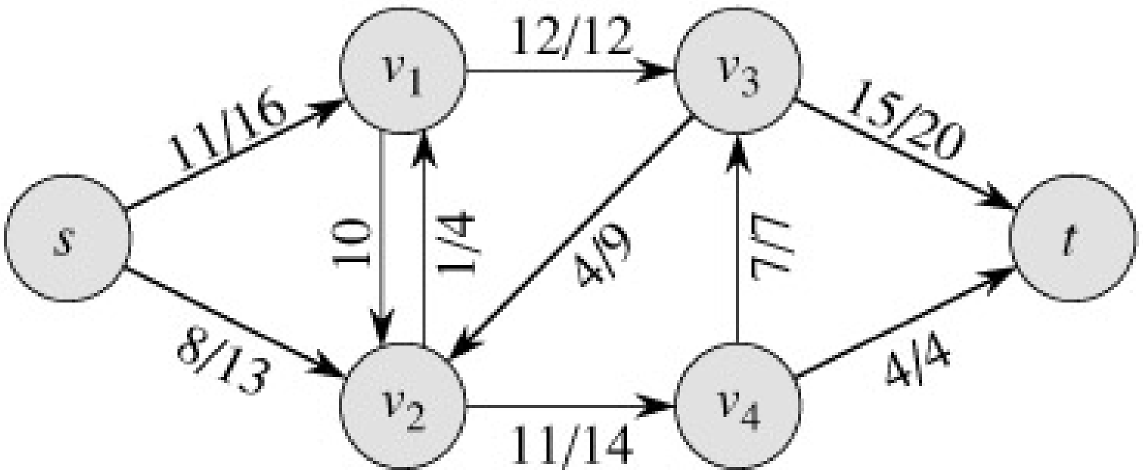
Example of Residual Network

Flow Network:

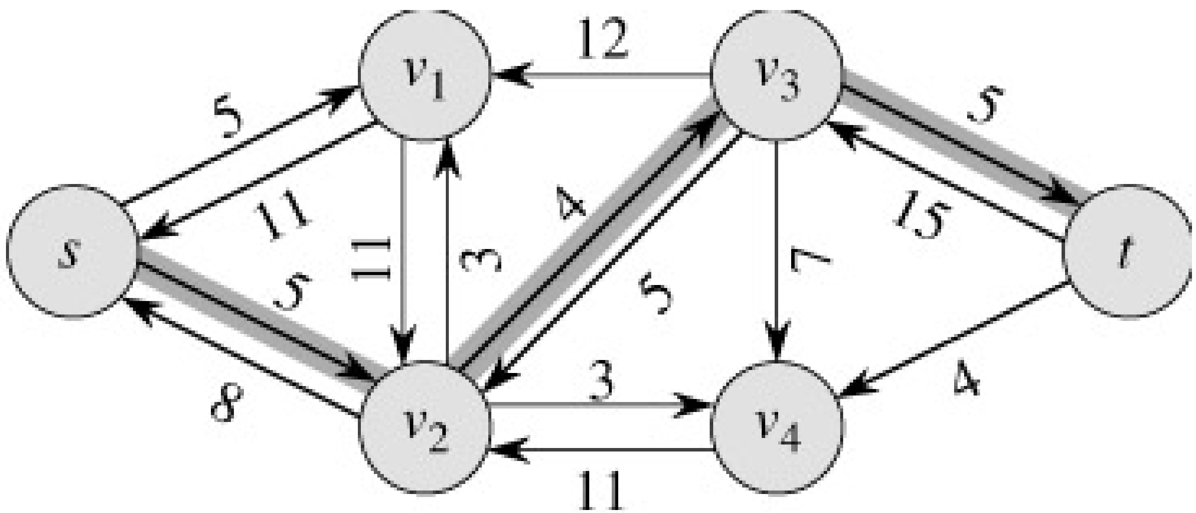


Example of Residual Network

Flow Network:



Residual Network:



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.

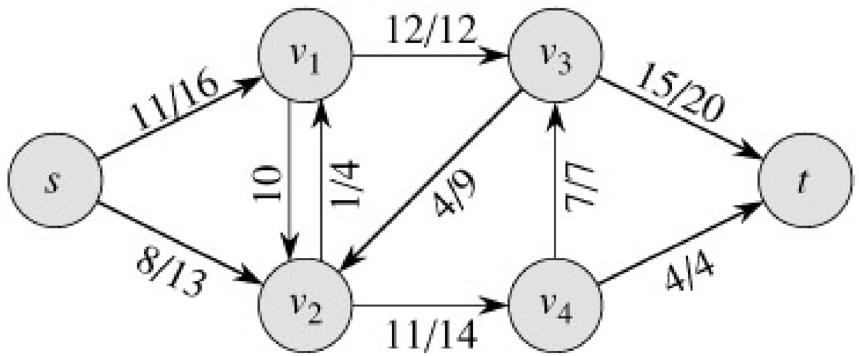
Augmenting Paths

- 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.

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

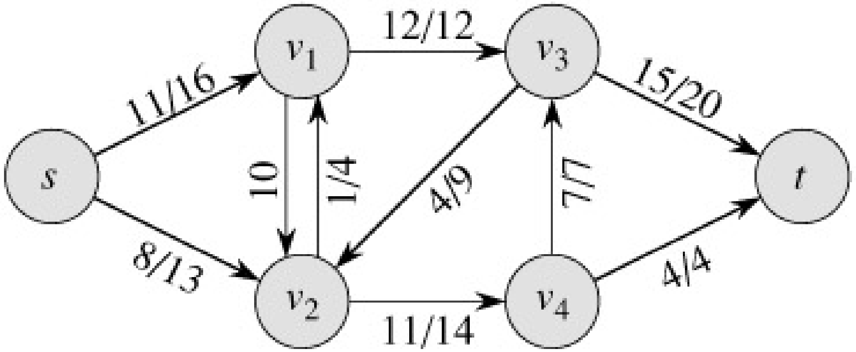
Augmenting Paths

Network:

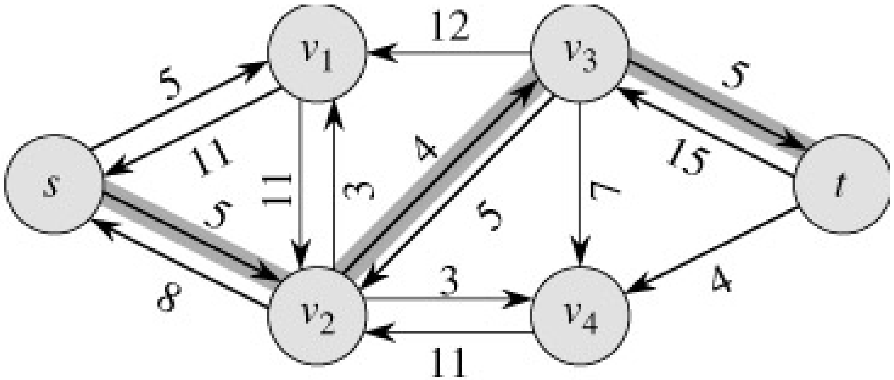


Augmenting Paths

Network:

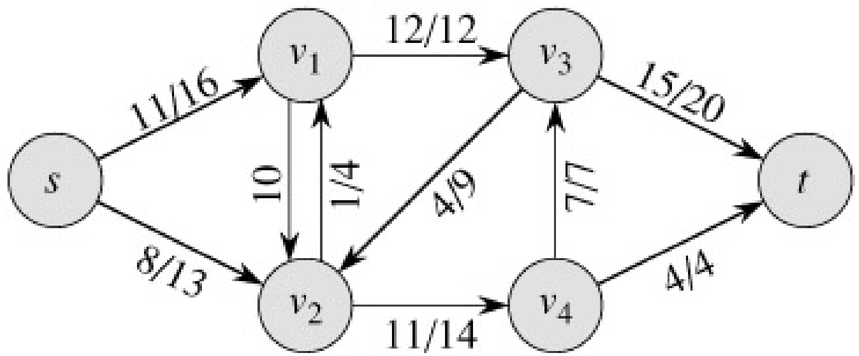


Residual Network:

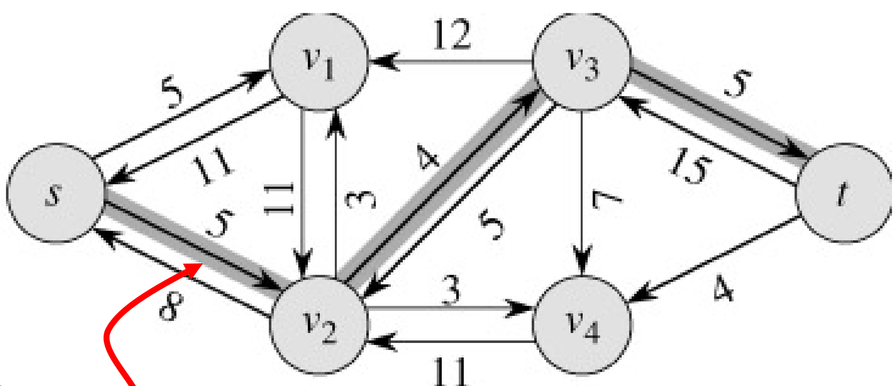


Augmenting Paths

Network:



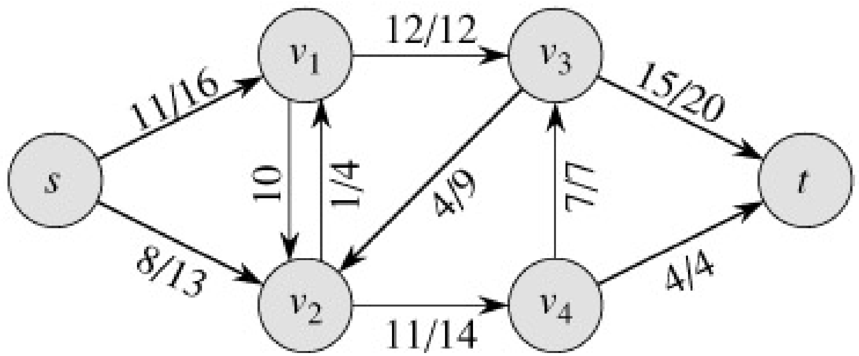
Residual Network:



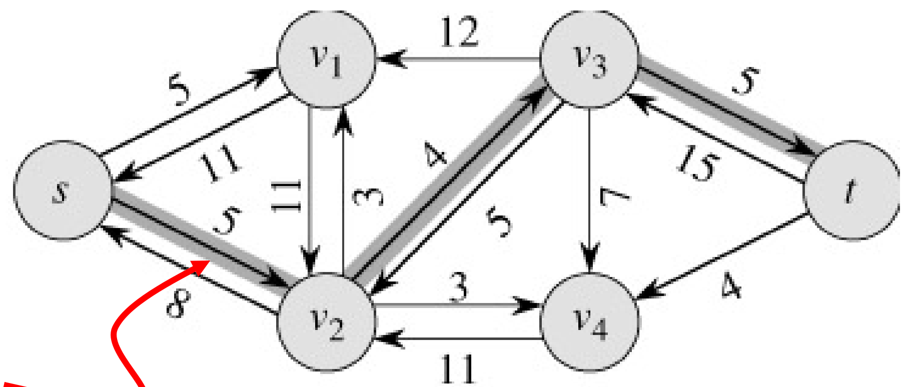
Augmenting path

Augmenting Paths

Network:



Residual Network:



Augmenting path

The residual capacity of this augmenting path is 4.

Computing Max Flow

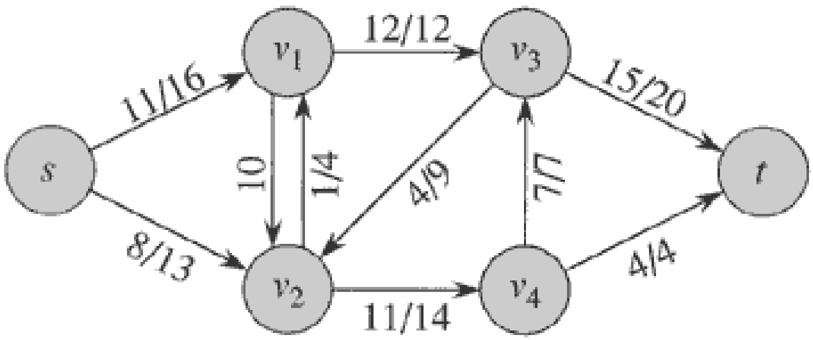
- Classic Method:
  - Identify augmenting path
  - Increase flow along that path
  - Repeat

Ford-Fulkerson Method

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

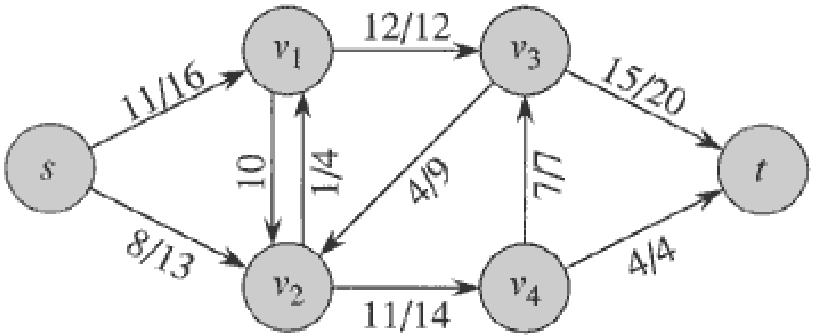
initialize flow f to 0
while there exists an augmenting path p
    do augment flow f along p
return f

Example

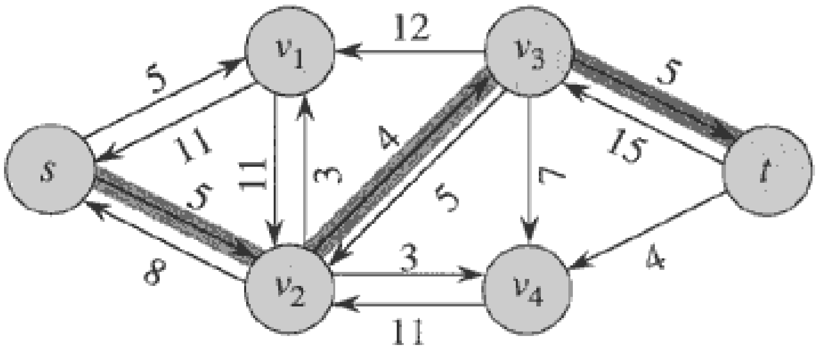


Flow(1)

Example

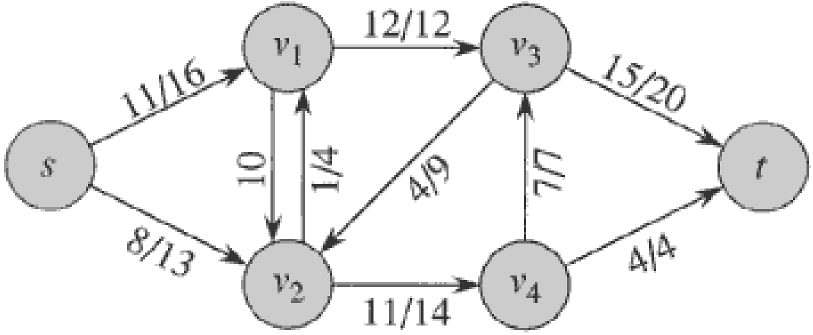


Flow(1)

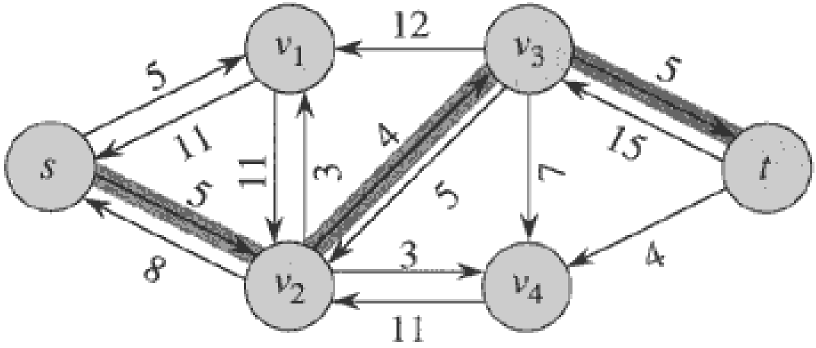


Residual(1)

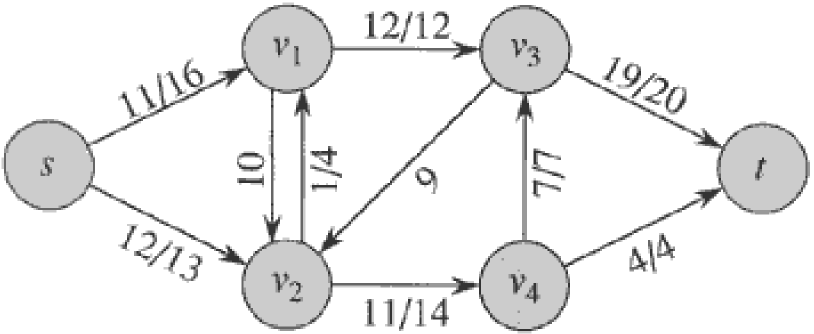
Example



Flow(1)

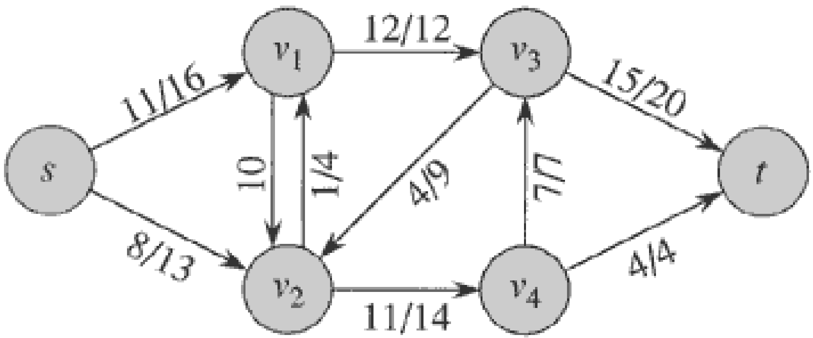


Residual(1)

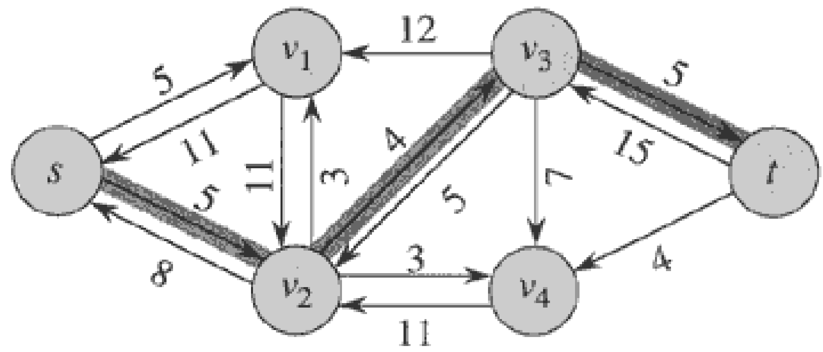


Flow(2)

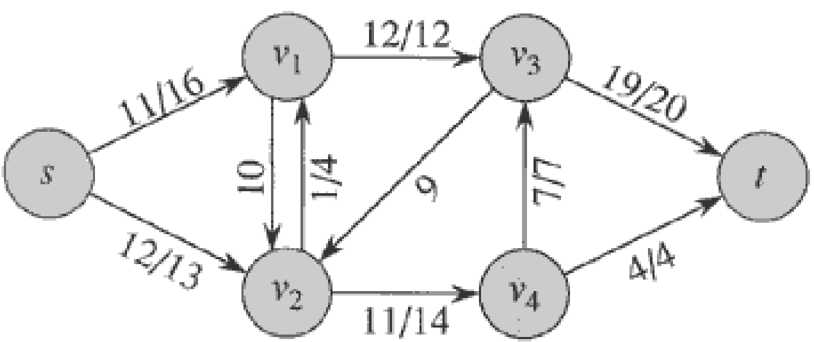
Example



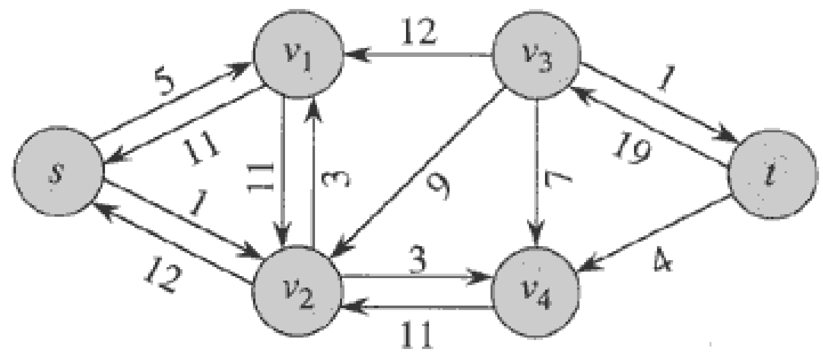
Flow(1)



Residual(1)

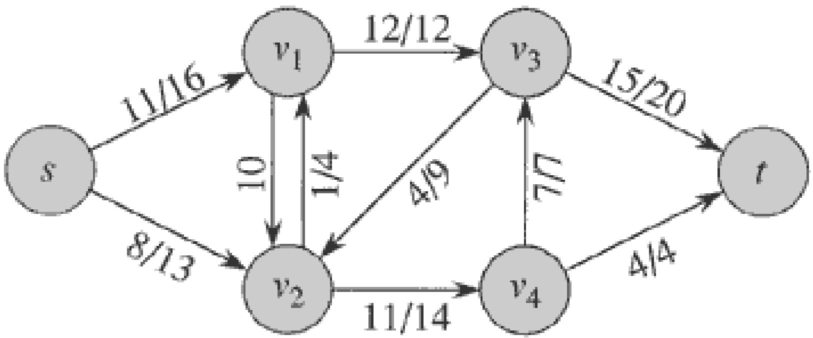


Flow(2)

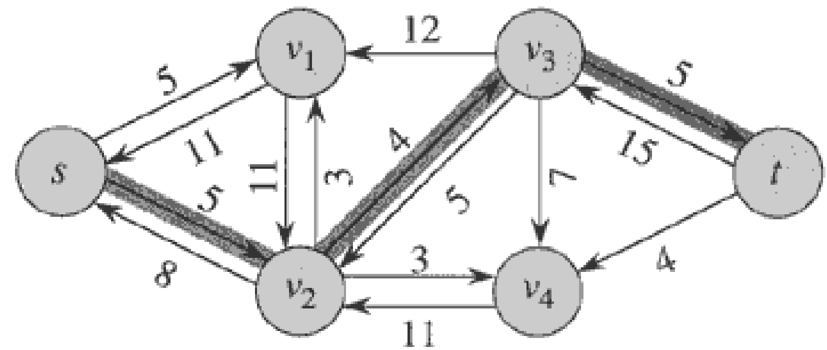


Residual(2)

Example

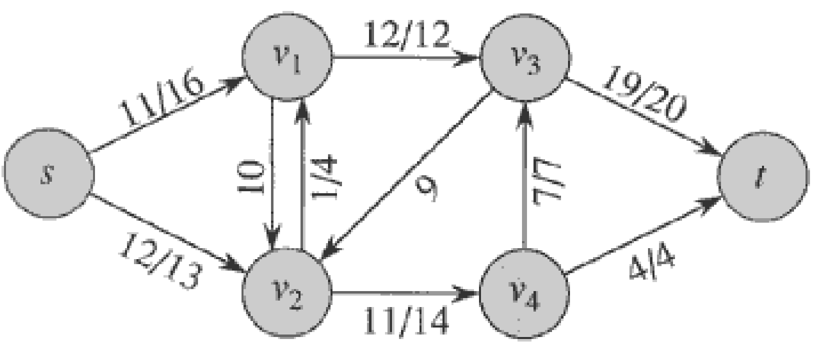


Flow(1)

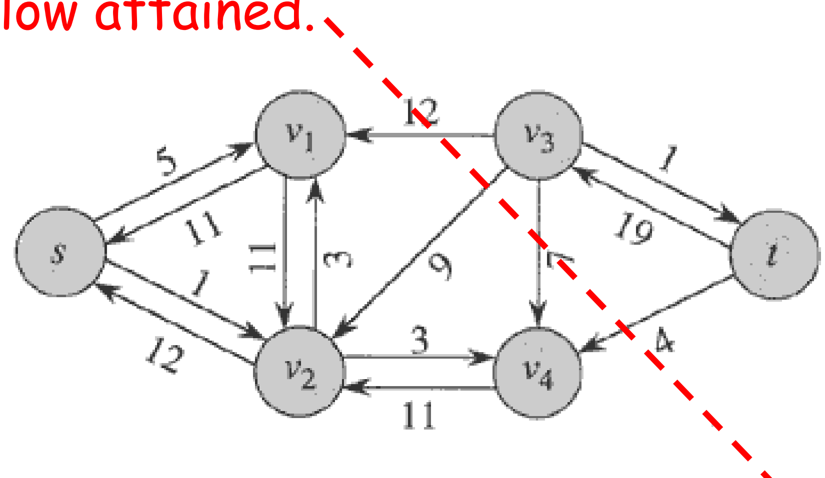


Residual(1)

No more augmenting paths → max flow attained.

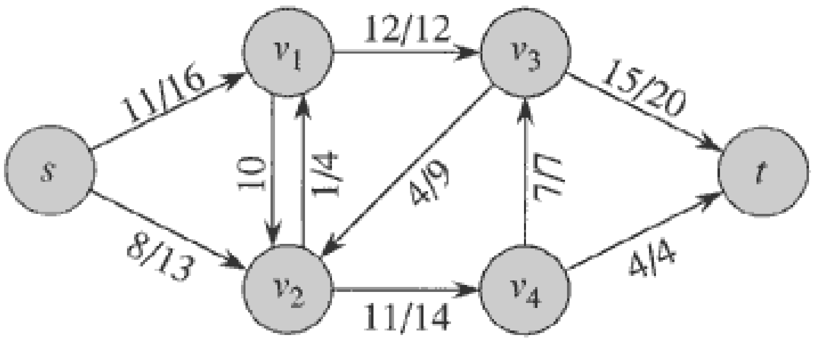


Flow(2)

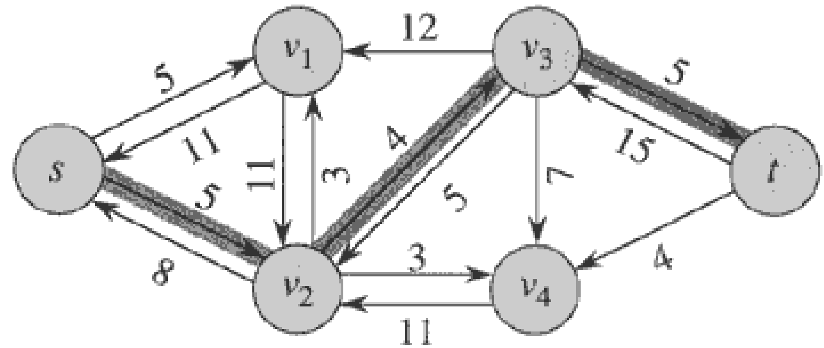


Residual(2)

Example

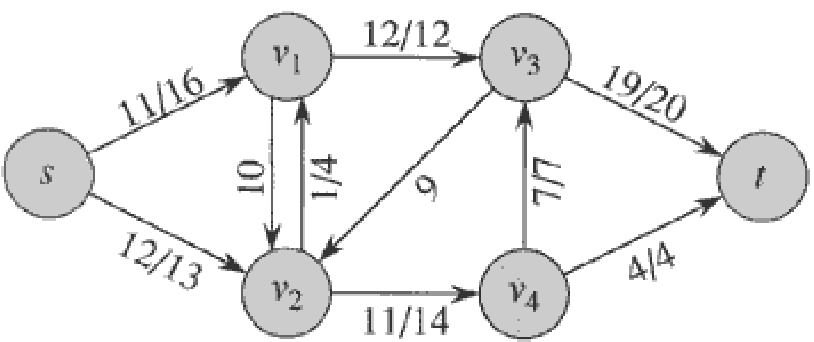


Flow(1)

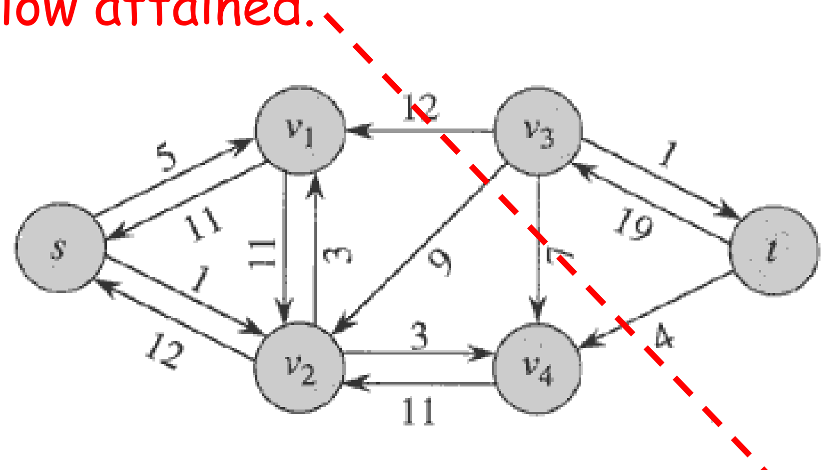


Residual(1)

No more augmenting paths → max flow attained.



Flow(2)

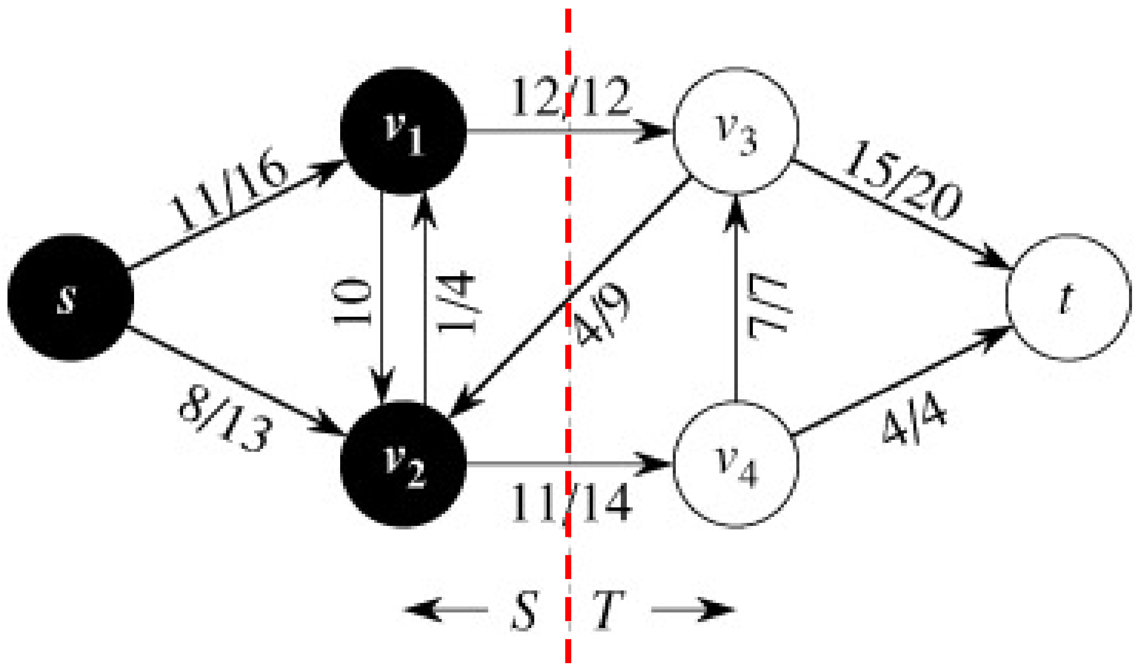


Residual(2)

Cut

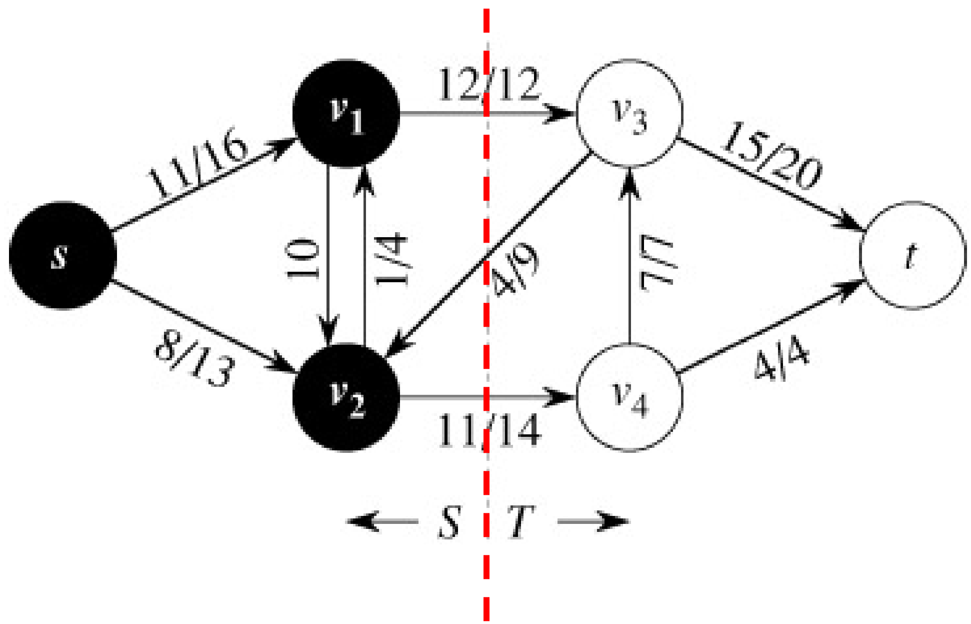
Cuts of Flow Networks

A cut (S,T) of a flow network is a partition of V into S and $T = V - S$ such that $s \in S$ and $t \in T$.



The Net Flow through a Cut (S,T)

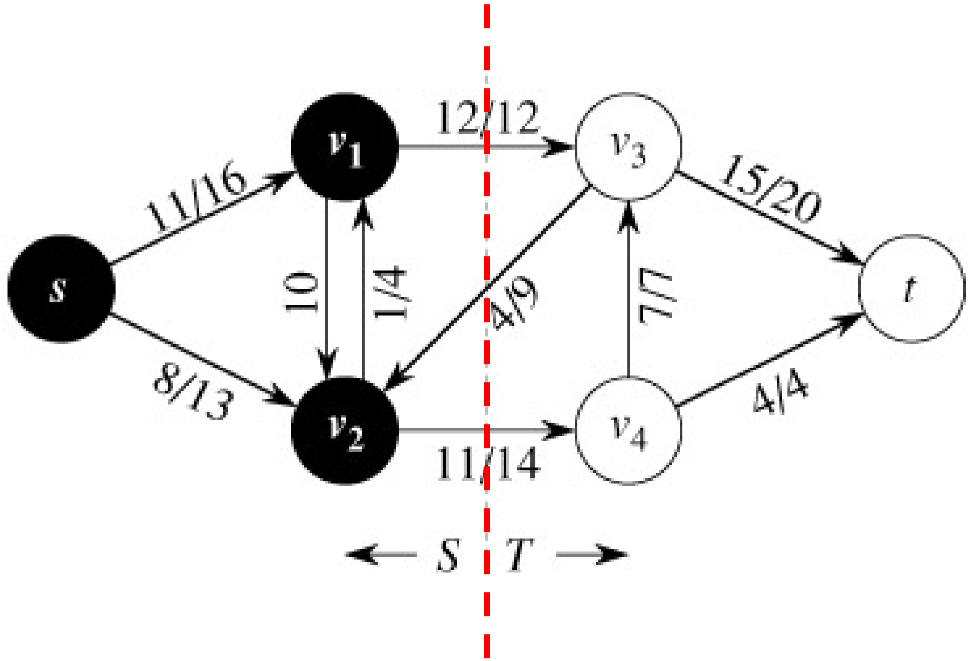
$$f(S,T) = \sum_{u \in S, v \in T} f(u,v)$$



- f(S,T) = 12 – 4 + 11 = 19

The Capacity of a Cut (S,T)

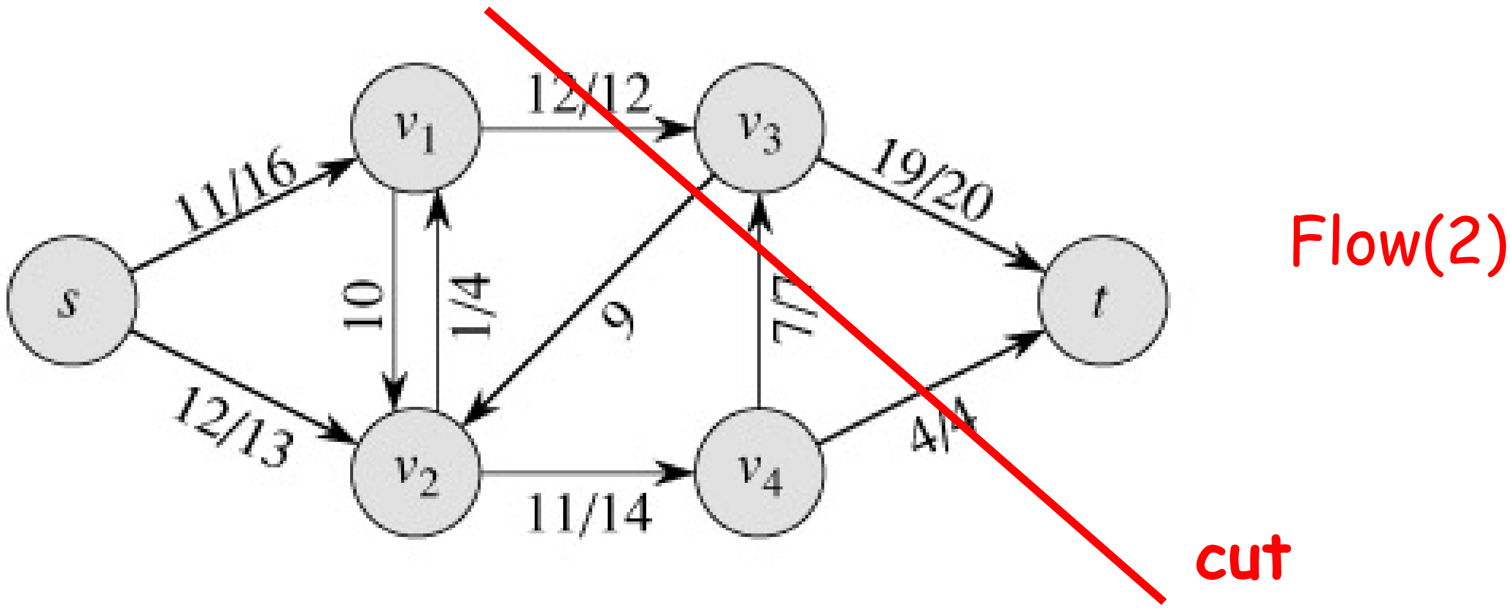
$$c(S,T) = \sum_{u \in S, v \in T} c(u,v)$$



- c(S,T)= 12+ 0 + 14 = 26

Augmenting Paths – example

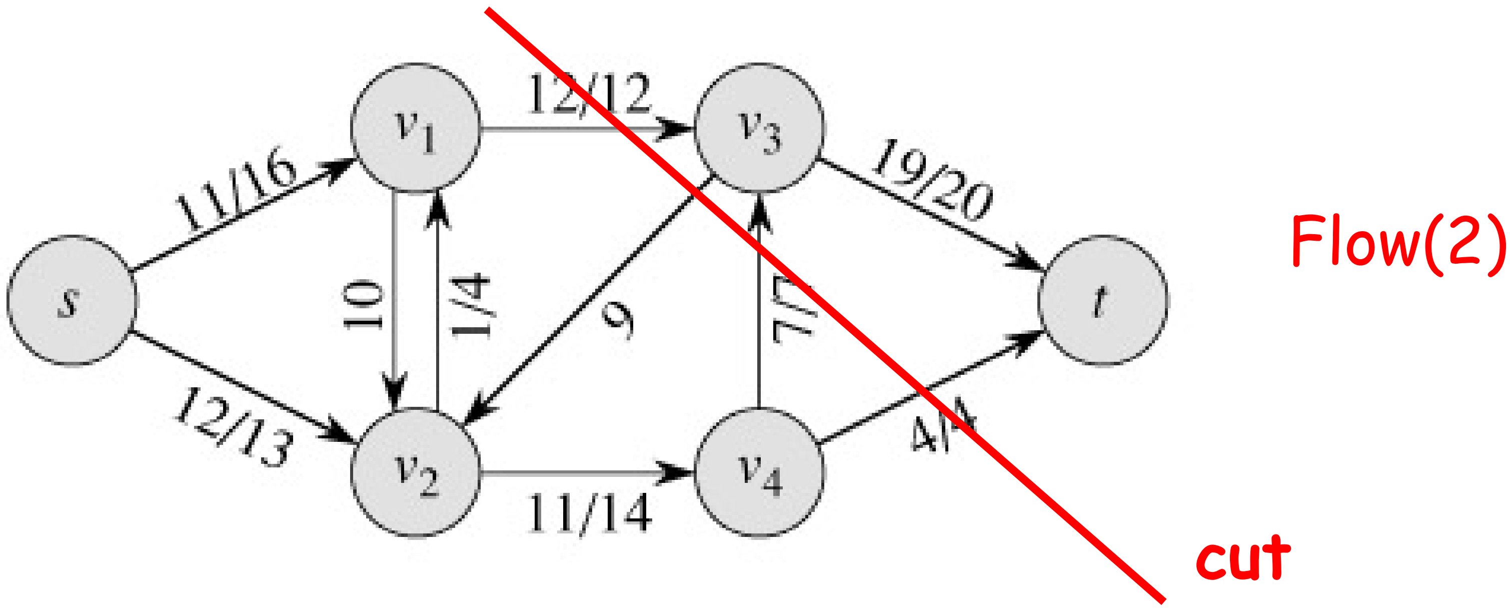
- **Capacity of the cut**
  = maximum possible flow through the cut
  = 12 + 7 + 4 = 23



- The network has a capacity of **at most** 23.

Augmenting Paths – example

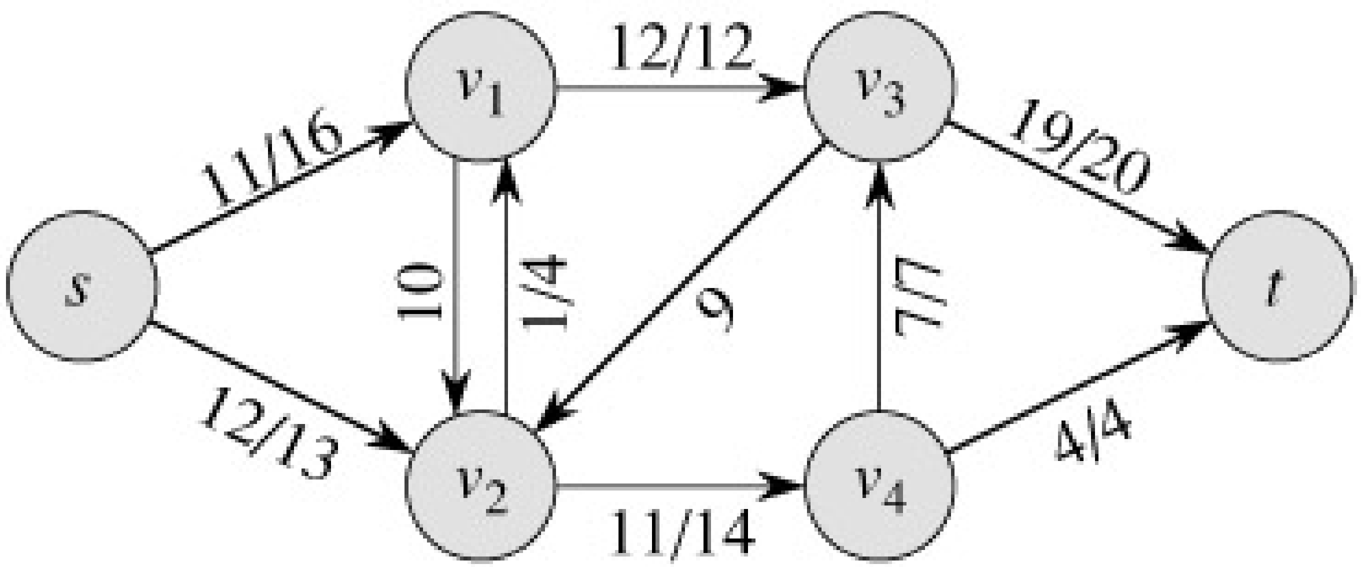
- **Capacity of the cut**
  = maximum possible flow through the cut
  = 12 + 7 + 4 = 23



- The network has a capacity of **at most** 23.
- In this case, the network **does** have a capacity of 23, because this is a **minimum cut**.

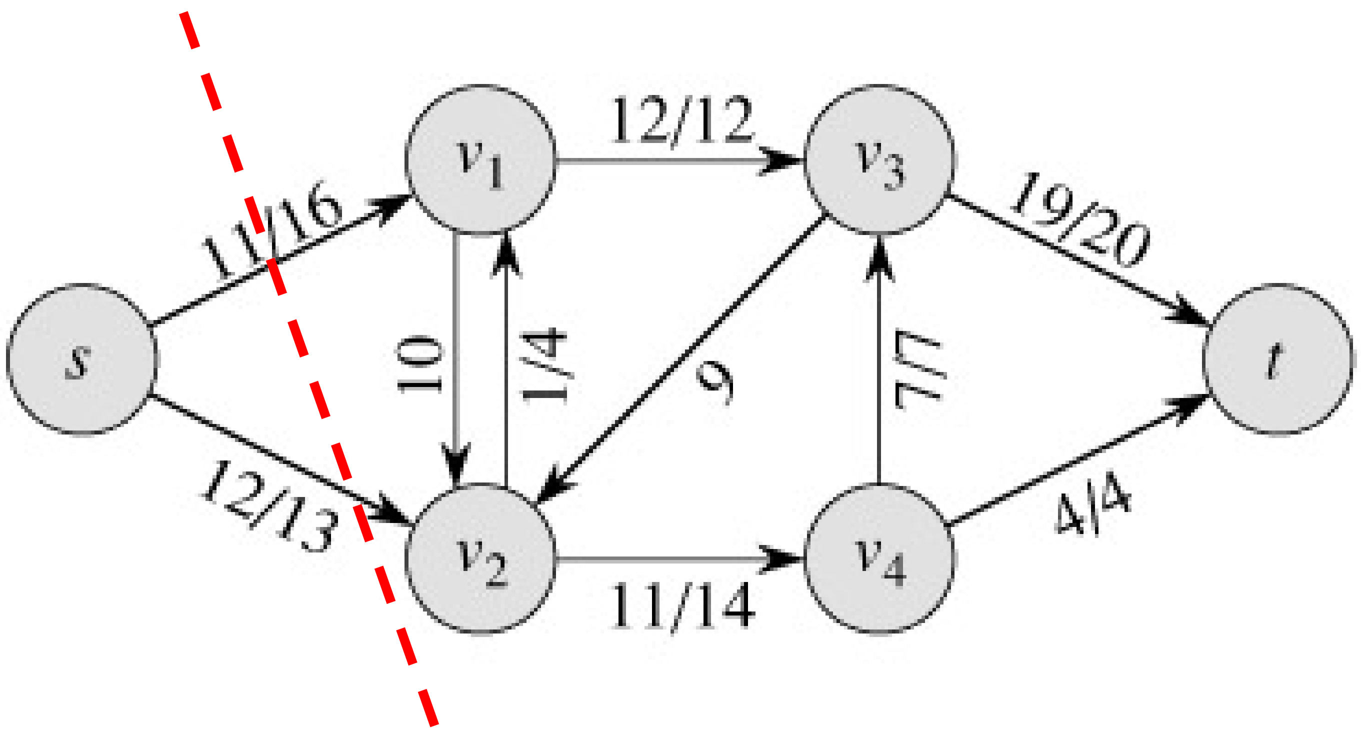
Net Flow of a Network

- The net flow across any cut is the same and equal to the flow of the network |f|.



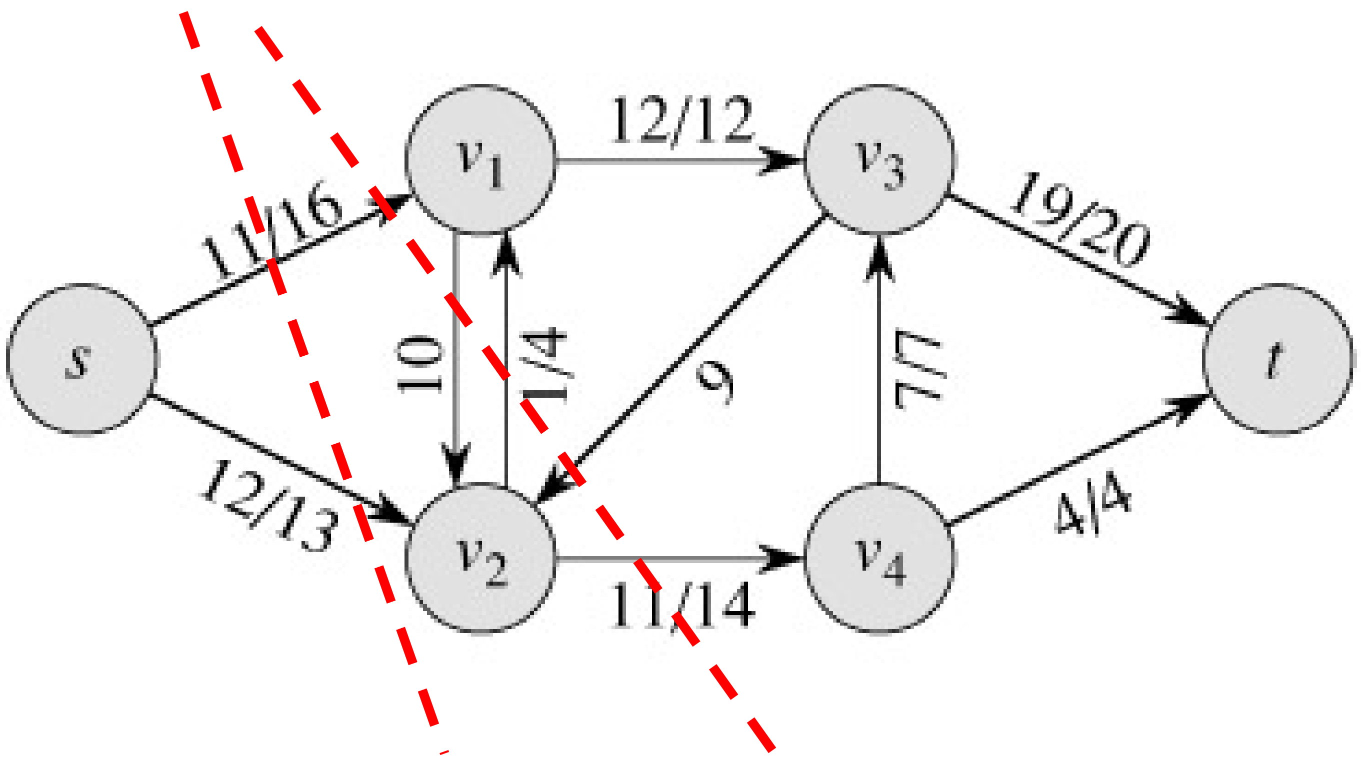
Net Flow of a Network

- The net flow across any cut is the same and equal to the flow of the network |f|.



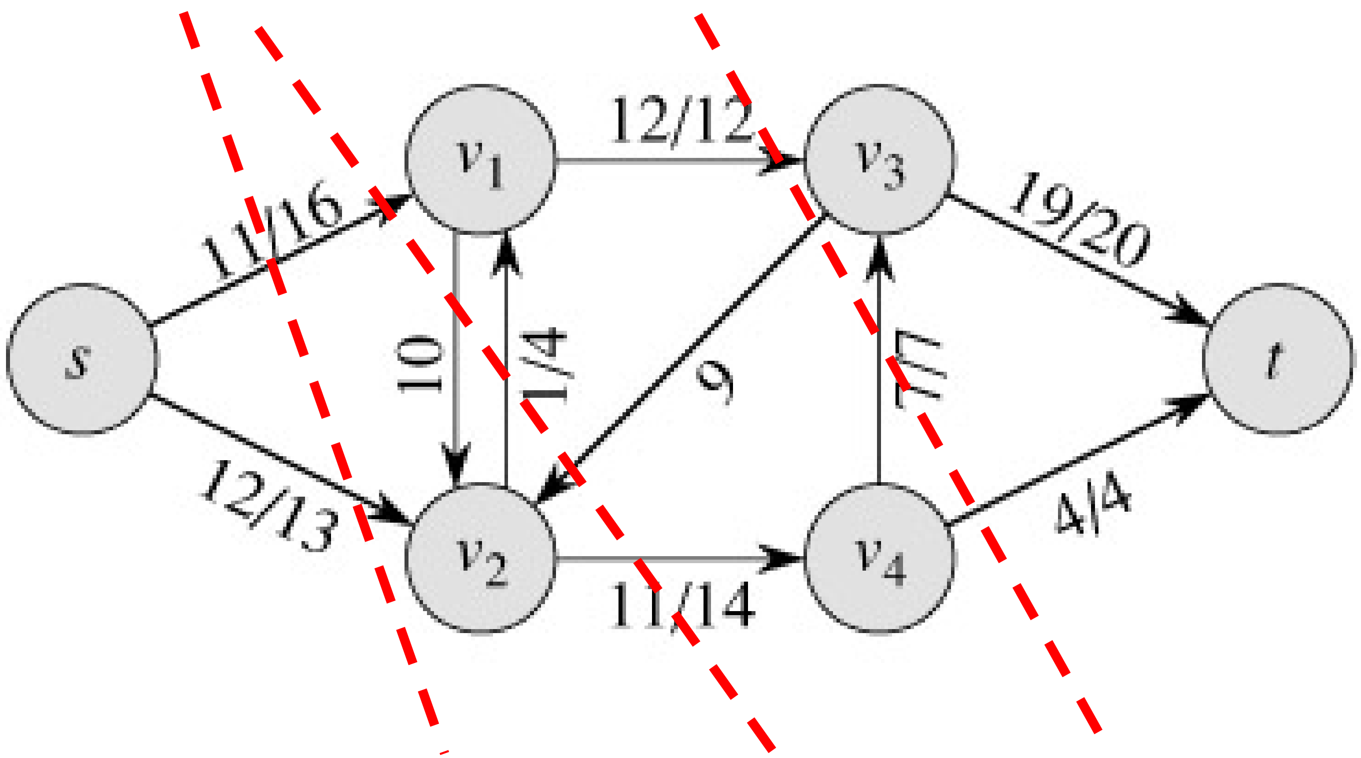
Net Flow of a Network

- The net flow across any cut is the same and equal to the flow of the network |f|.



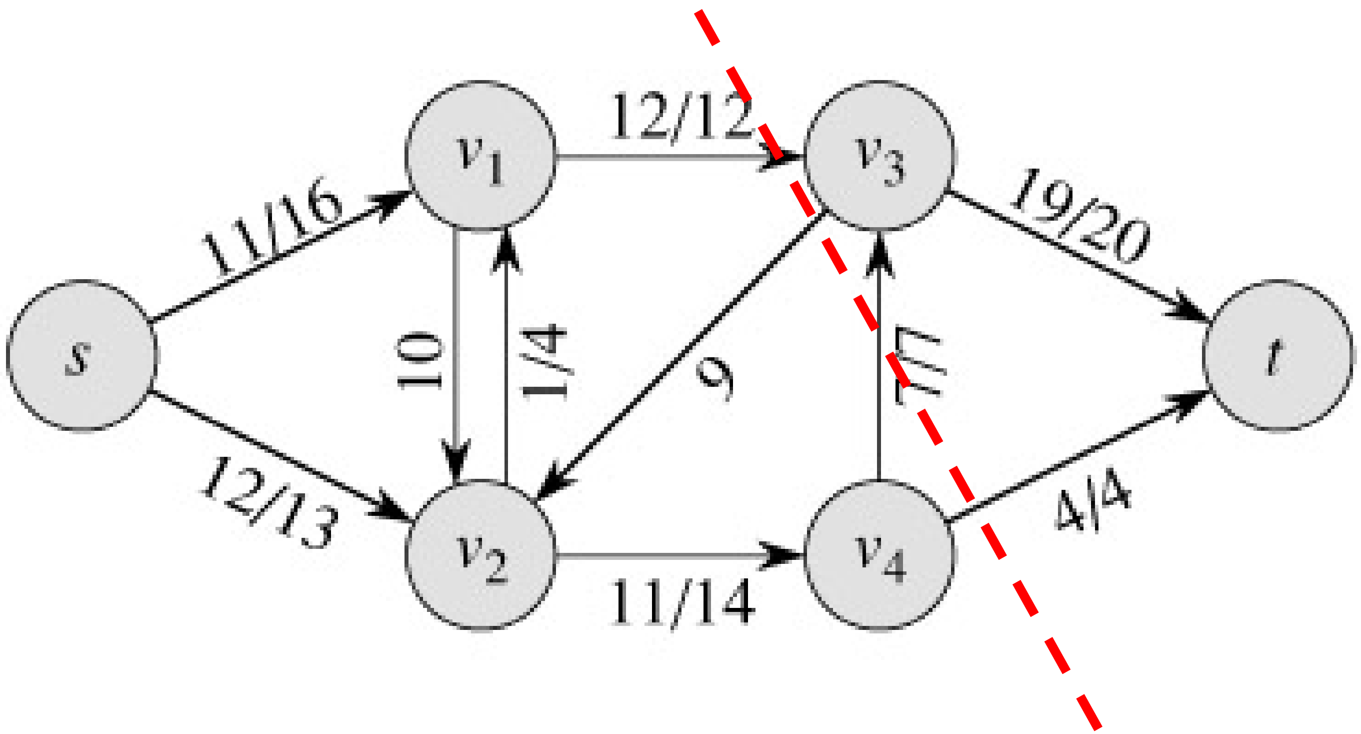
Net Flow of a Network

- The net flow across any cut is the same and equal to the flow of the network |f|.



Bounding the Network Flow

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



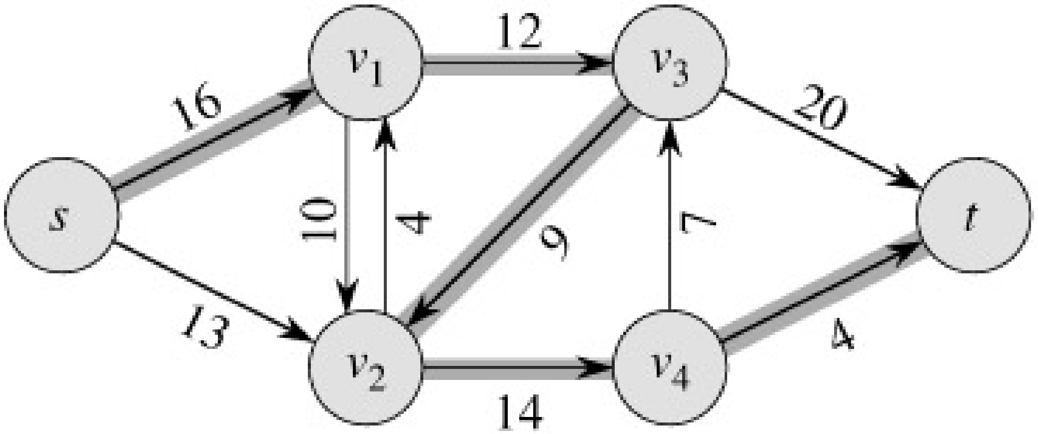
Max-Flow Min-Cut Theorem

- 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).

The Basic Ford-Fulkerson Algorithm

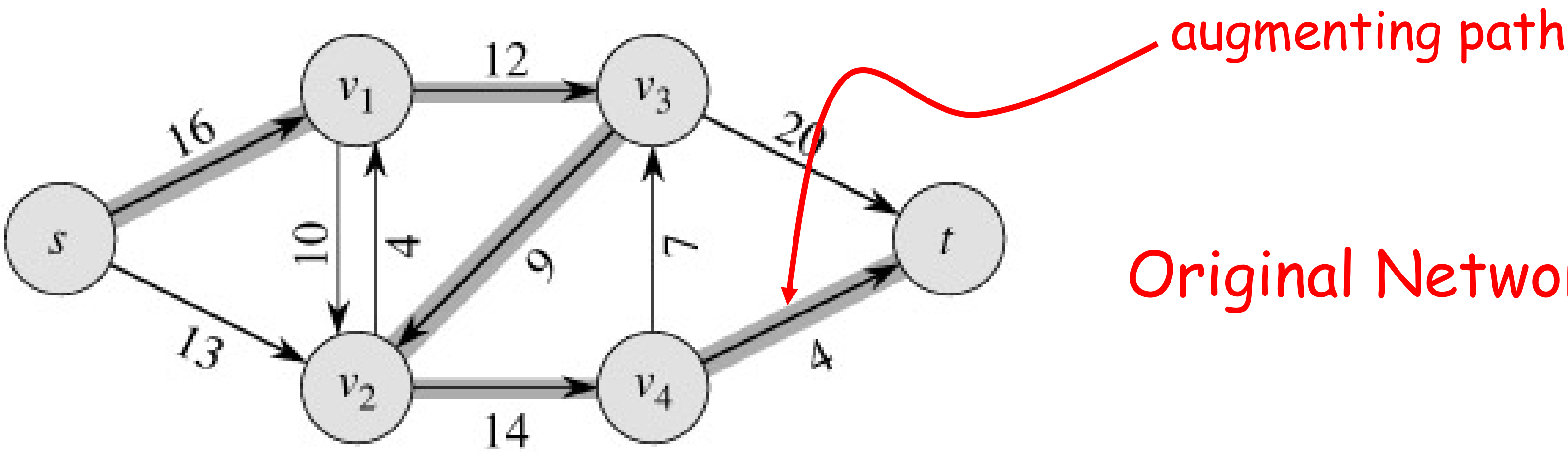
```
FORD-FULKERSON(G, s, t)
for each edge (u, v) ∈ E[G]
     do f[u, v] ← 0
        f[v, u] ← 0
while there exists a path p from s to t in the residual network G_f
     do c_f(p) ← min {c_f(u, v) : (u, v) is in p}
        for each edge (u, v) in p
            do f[u, v] ← f[u, v] + c_f(p)
          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

Example



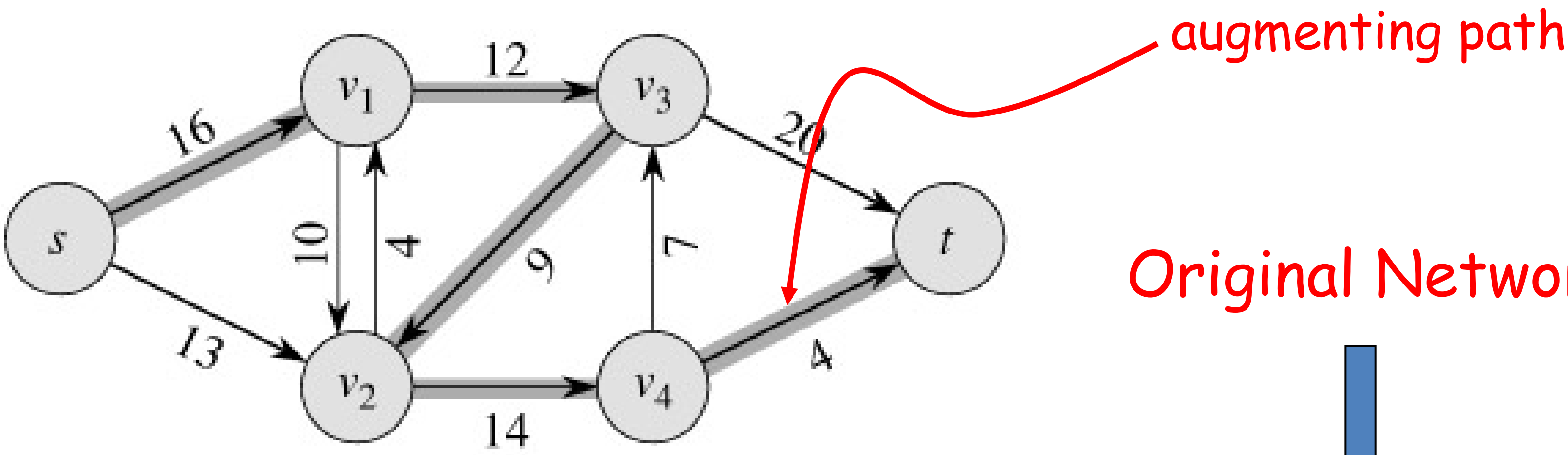
Original Network

Example

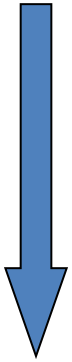


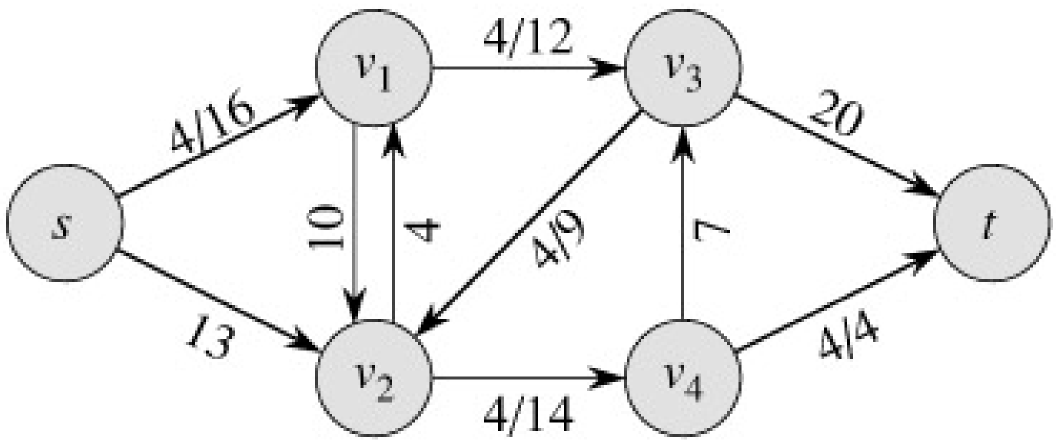
Original Network

Example



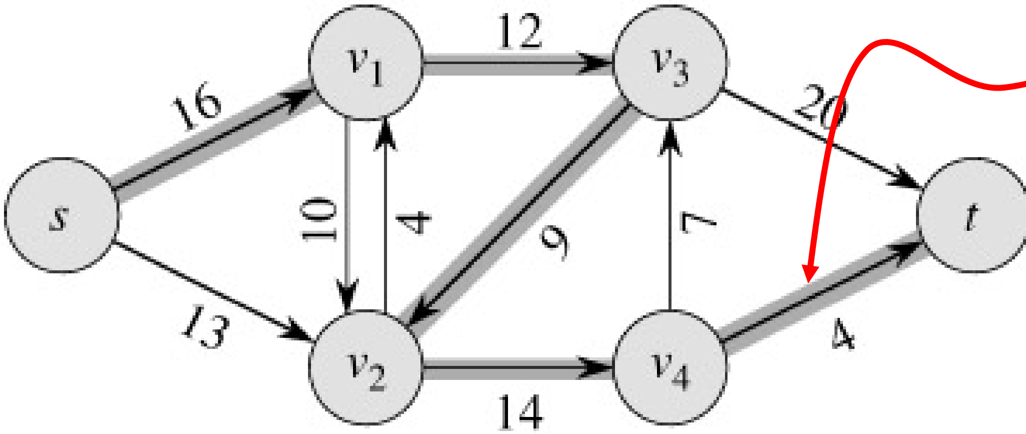
Original Network





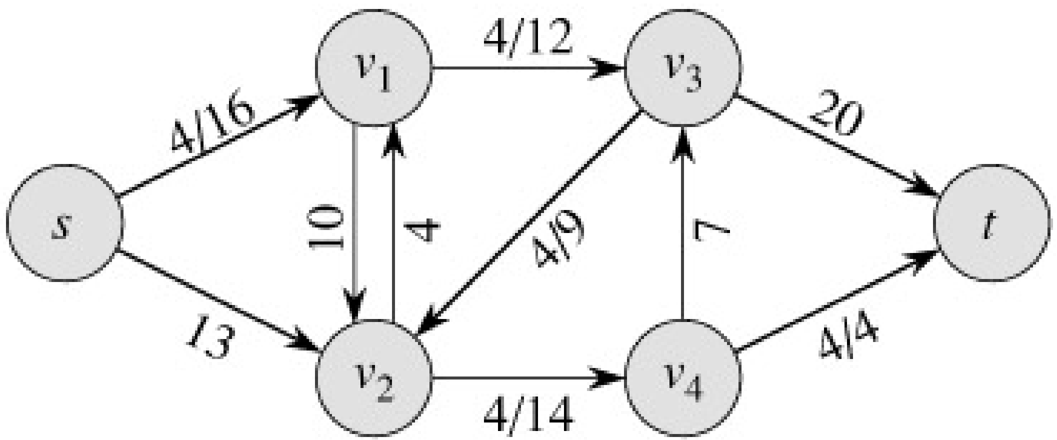
Flow Network

Example



augmenting path

Original Network

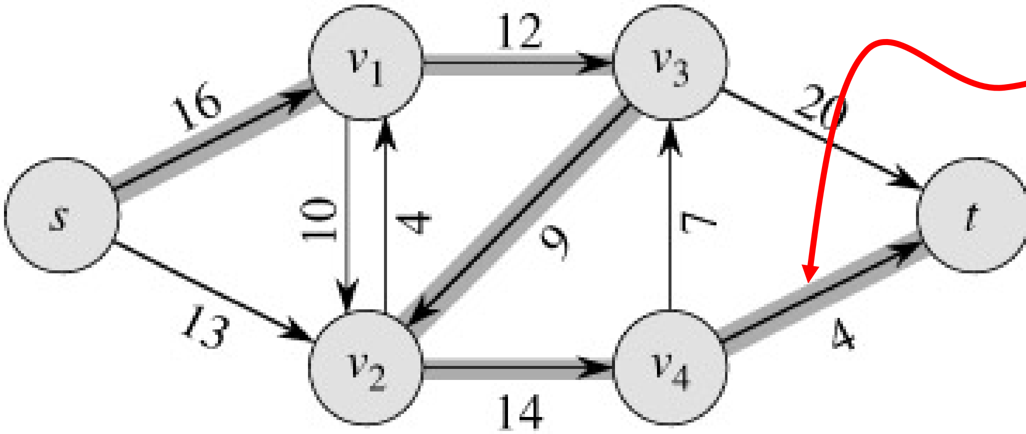


Flow Network

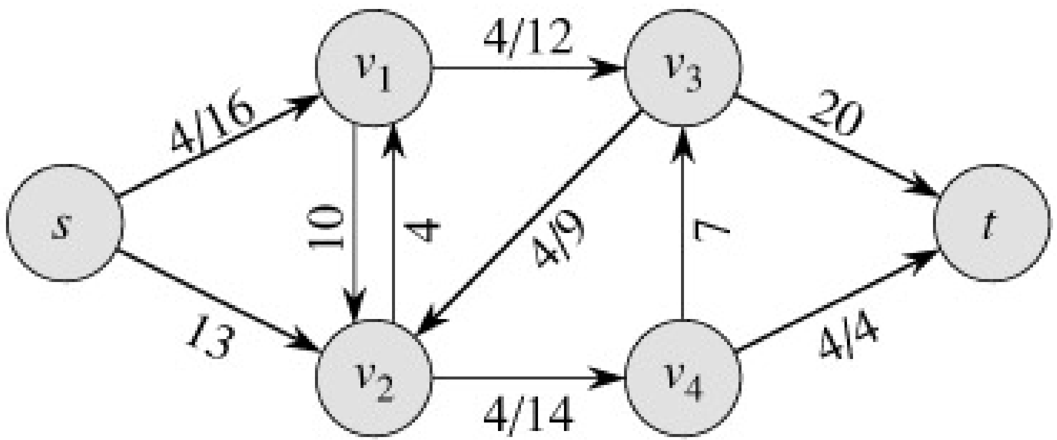
Resulting Flow = ?

Example

augmenting path



Original Network

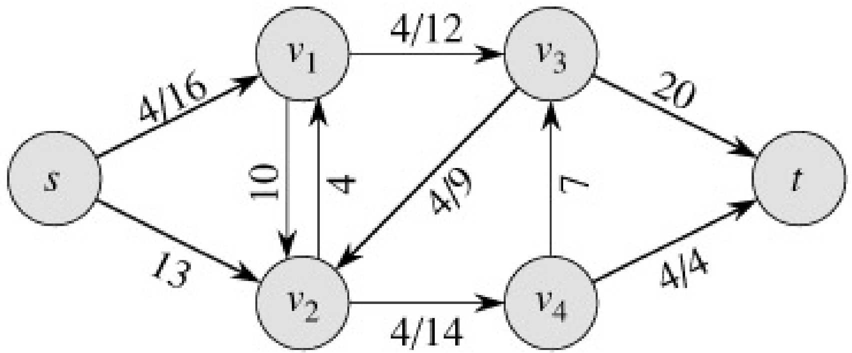


Flow Network

Resulting Flow = 4

Example

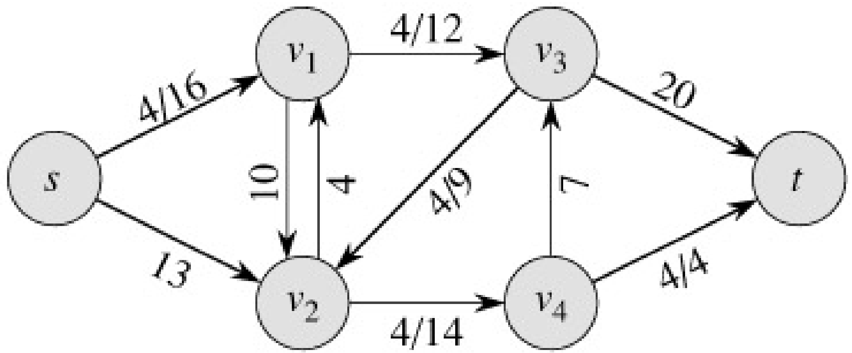
Flow Network



Resulting Flow = 4

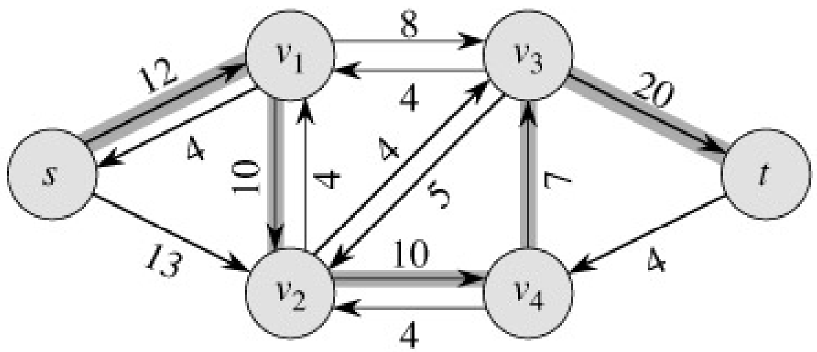
Example

Flow Network



Resulting Flow = 4

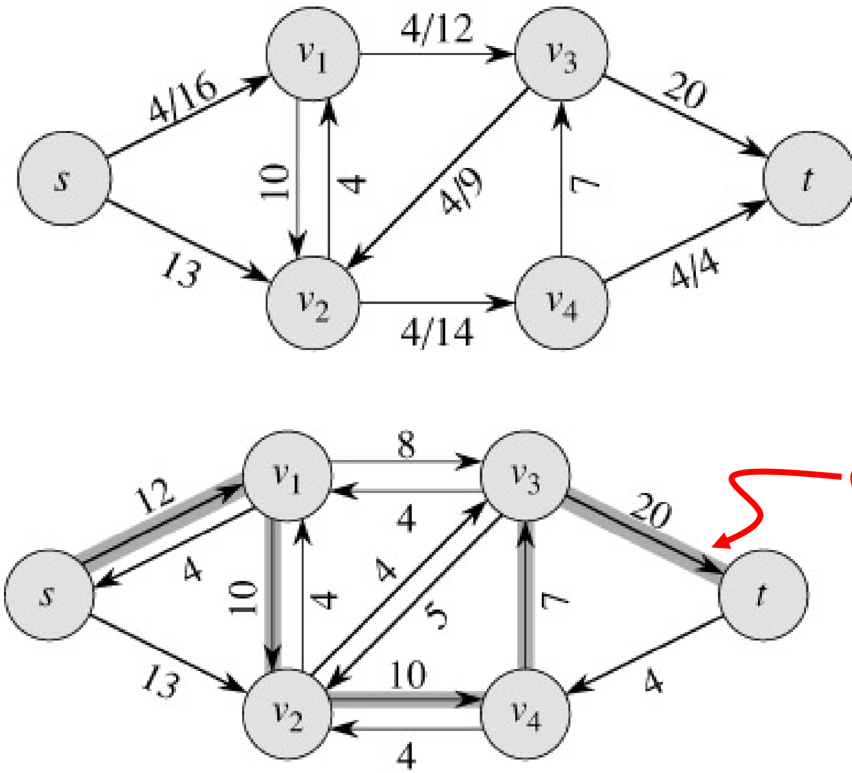
Residual Network



Example

Flow Network

Residual Network

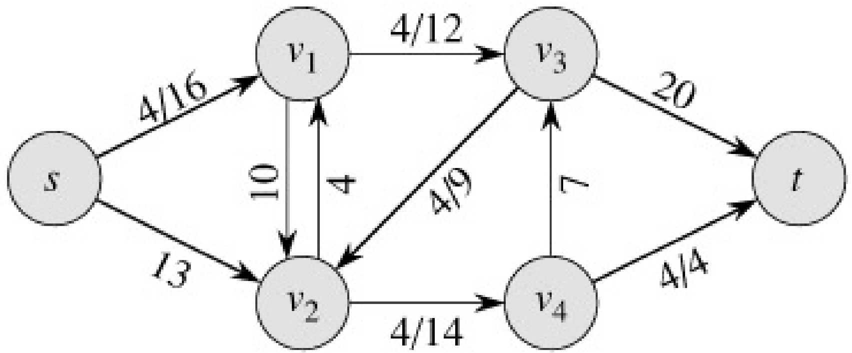


Resulting Flow = 4

augmenting path

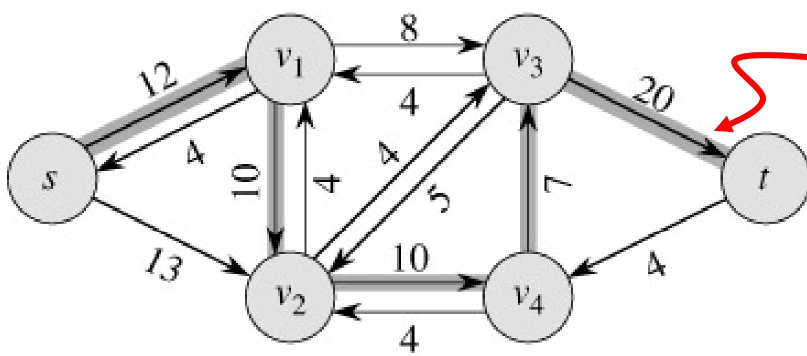
Example

Flow Network



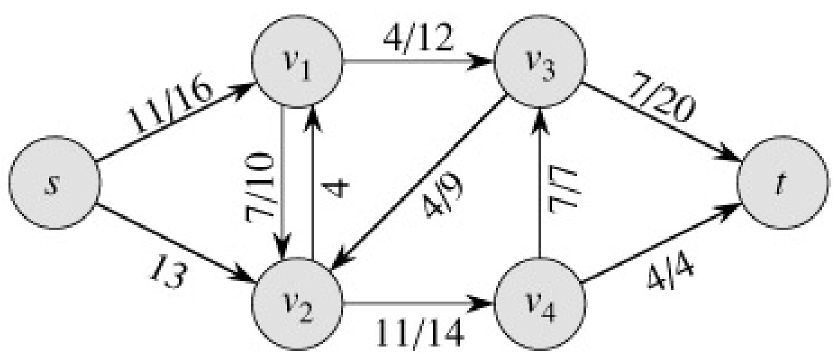
Resulting Flow = 4

Residual Network



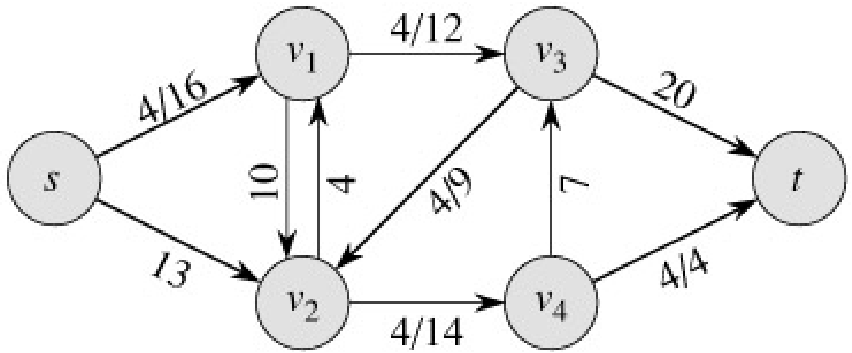
augmenting path

Flow Network



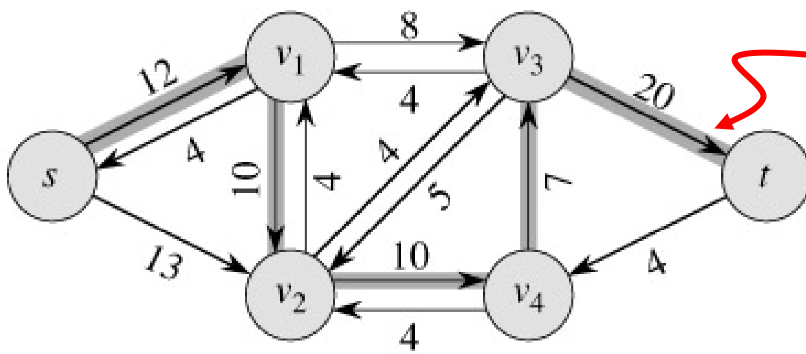
Example

Flow Network



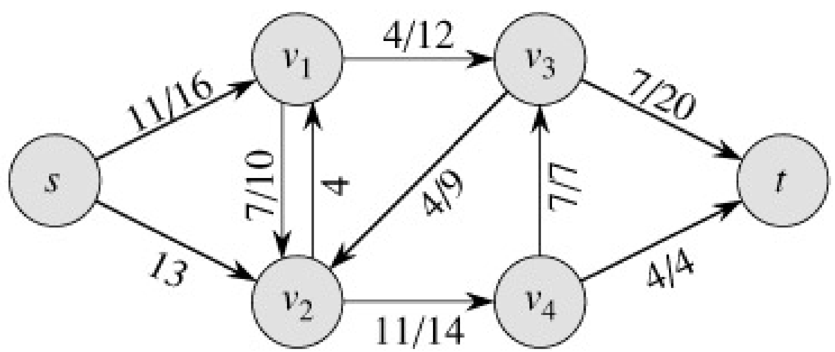
Resulting Flow = 4

Residual Network



augmenting path

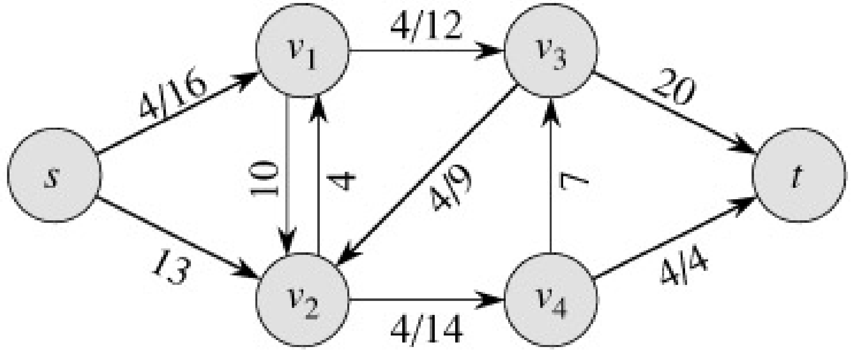
Flow Network



Resulting Flow = ?

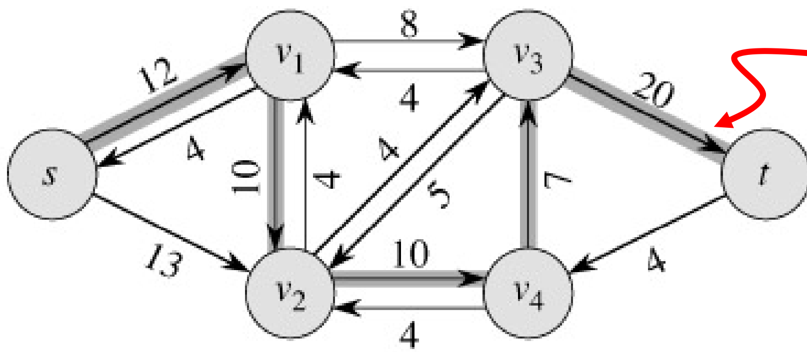
Example

Flow Network



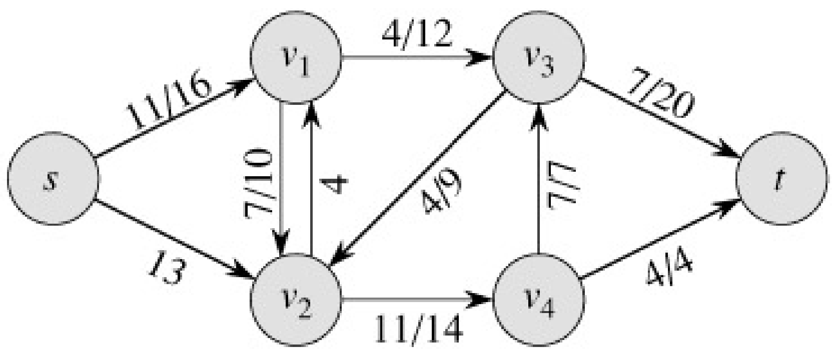
Resulting Flow = 4

Residual Network



augmenting path

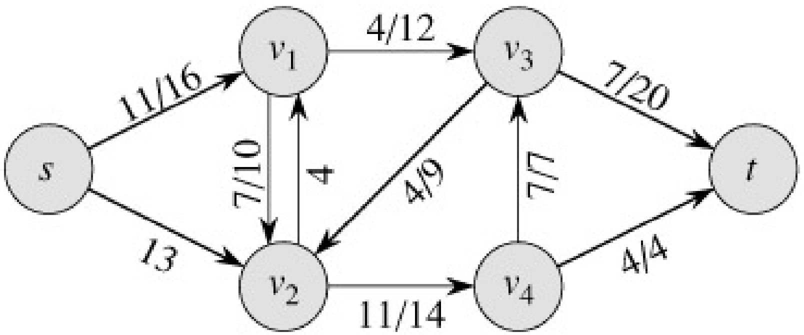
Flow Network



Resulting Flow = 11

Example

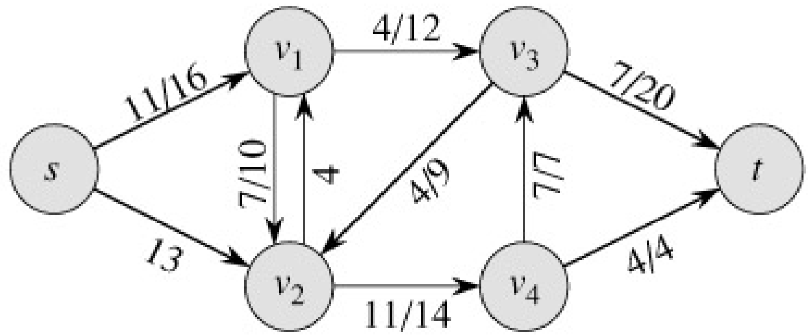
Flow Network



Resulting Flow = 11

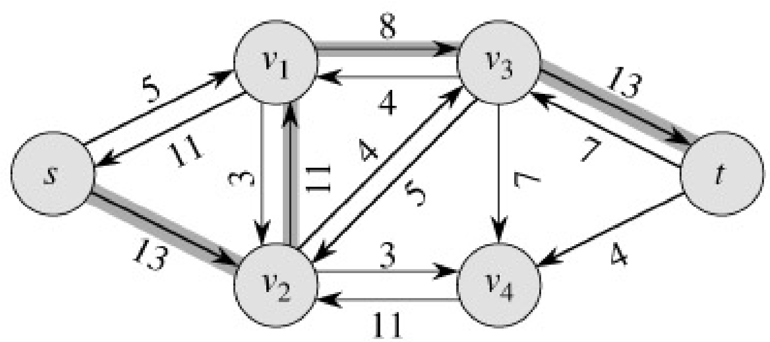
Example

Flow Network



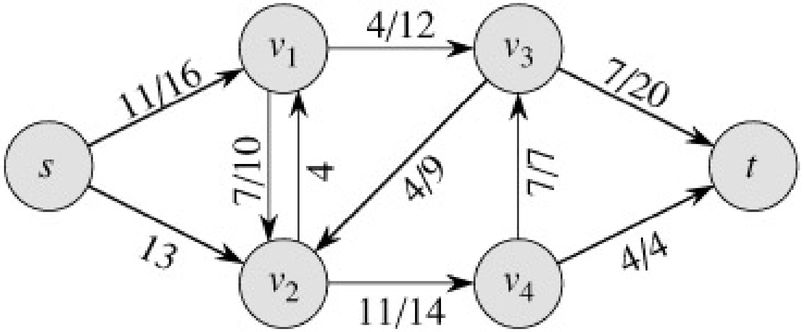
Resulting Flow = 11

Residual Network



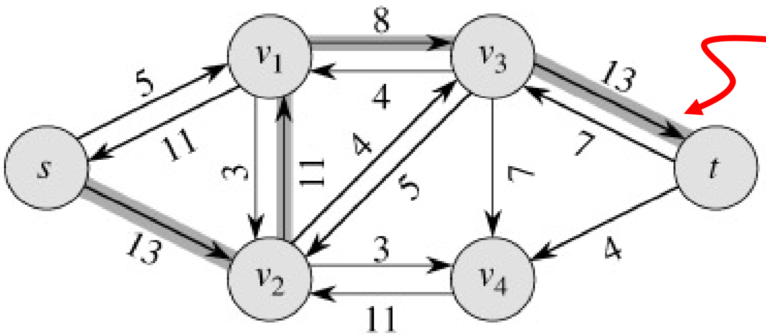
Example

Flow Network



Resulting Flow = 11

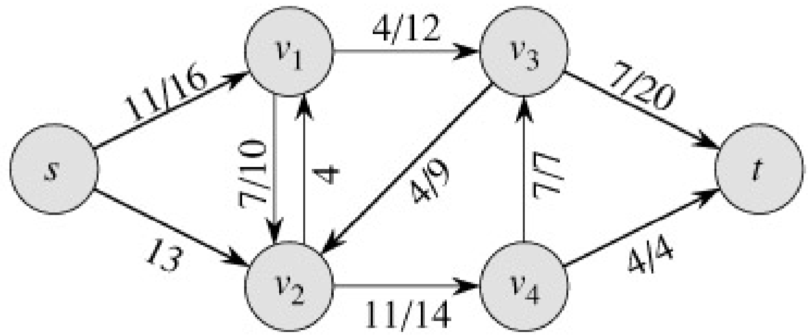
Residual Network



augmenting path

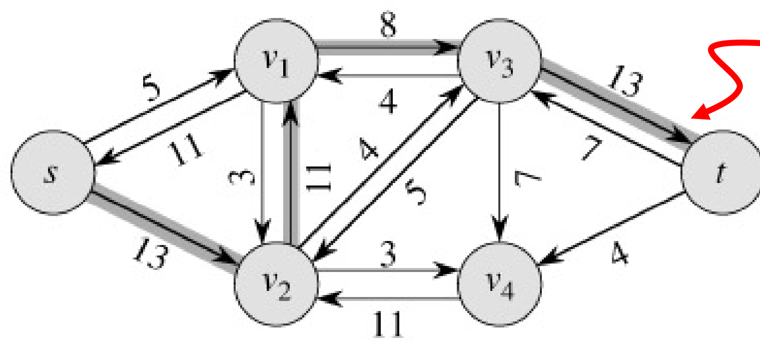
Example

Flow Network



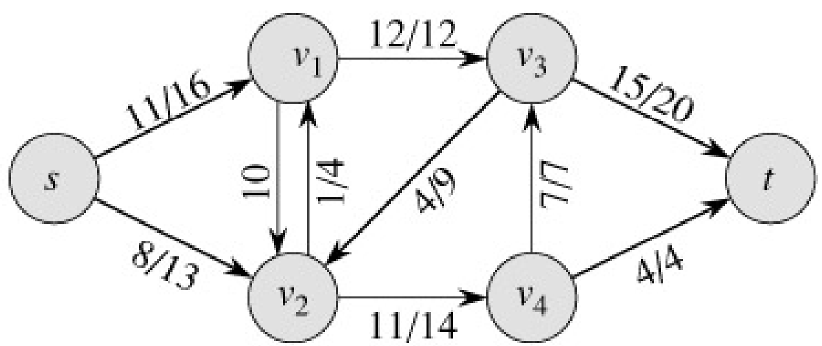
Resulting Flow = 11

Residual Network



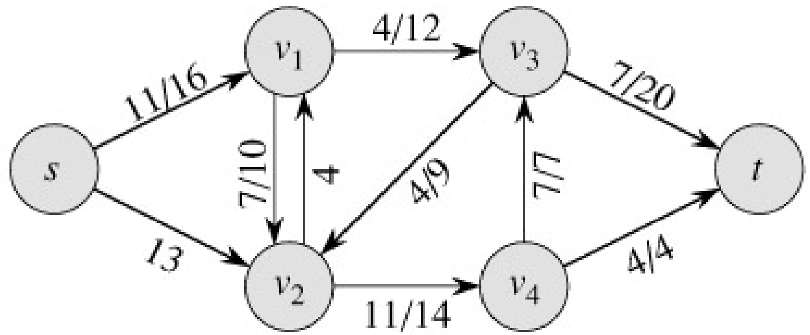
augmenting path

Flow Network



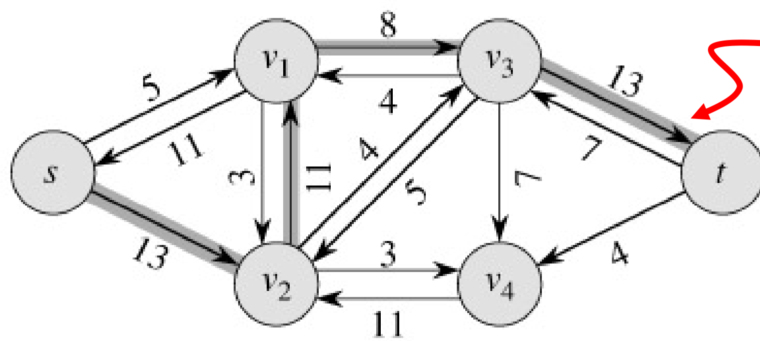
Example

Flow Network



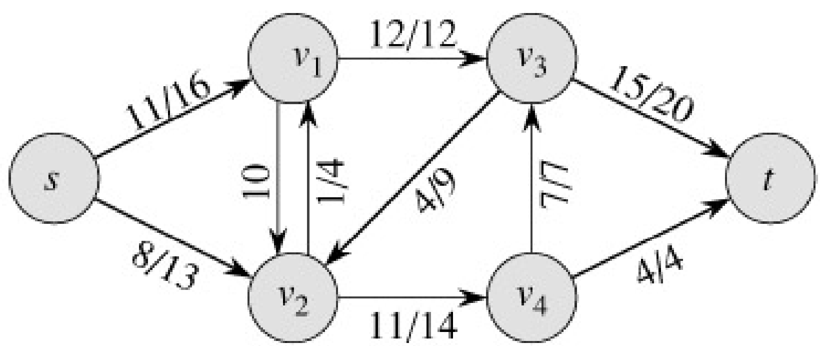
Resulting Flow = 11

Residual Network



augmenting path

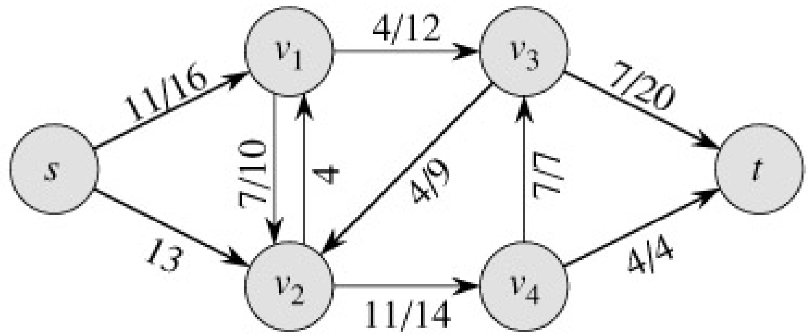
Flow Network



Resulting Flow = ?

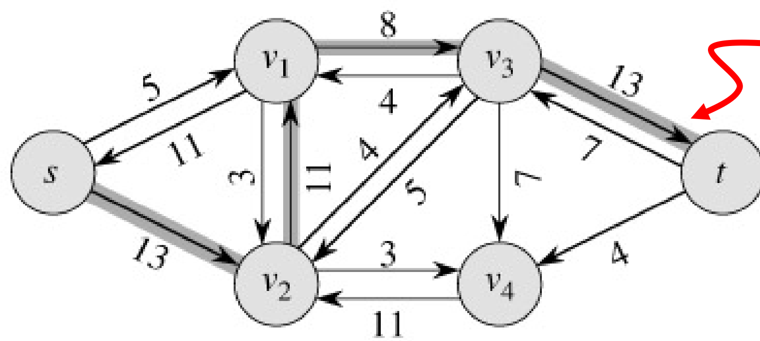
Example

Flow Network



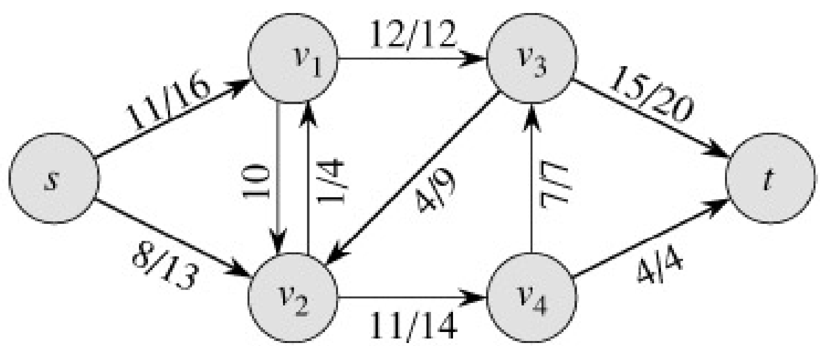
Resulting Flow = 11

Residual Network



augmenting path

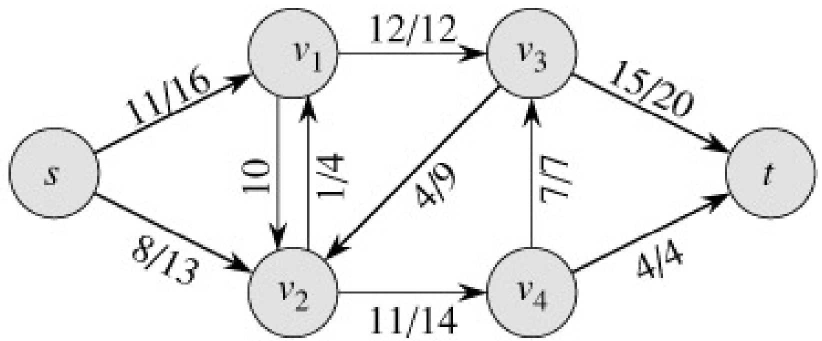
Flow Network



Resulting Flow = 19

Example

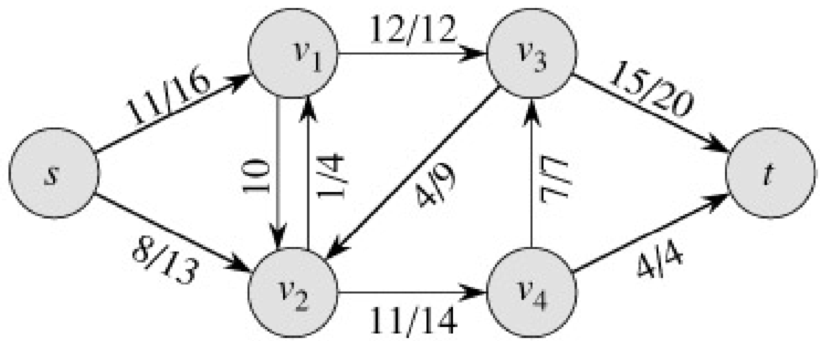
Flow Network



Resulting Flow = 19

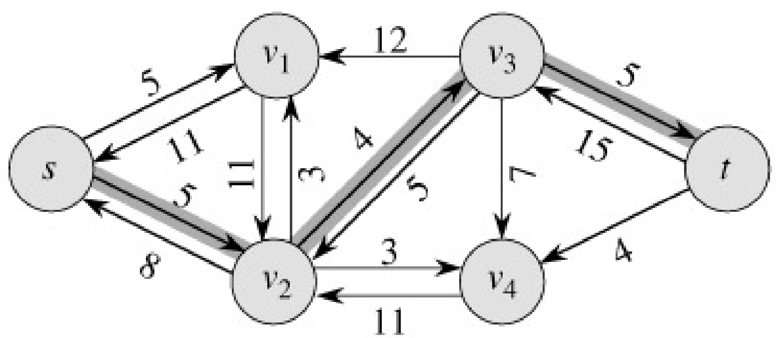
Example

Flow Network



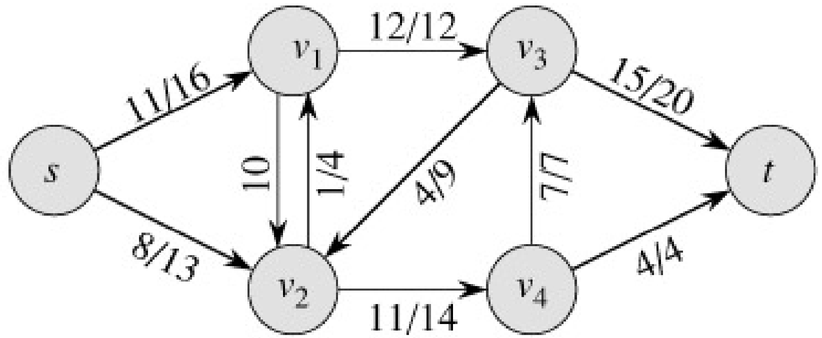
Resulting Flow = 19

Residual Network



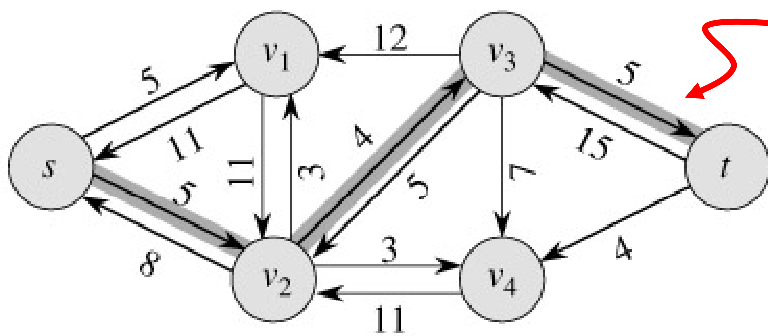
Example

Flow Network



Resulting Flow = 19

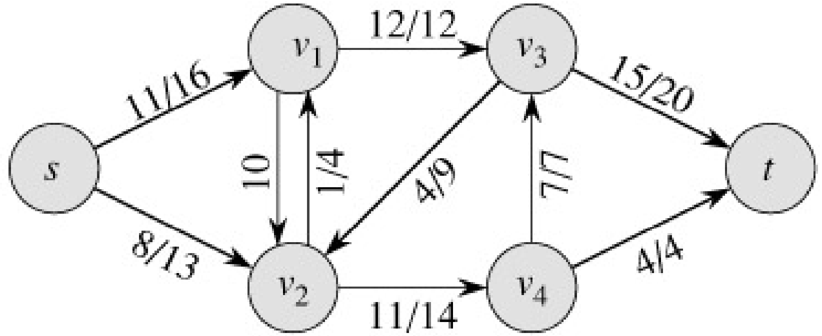
Residual Network



augmenting path

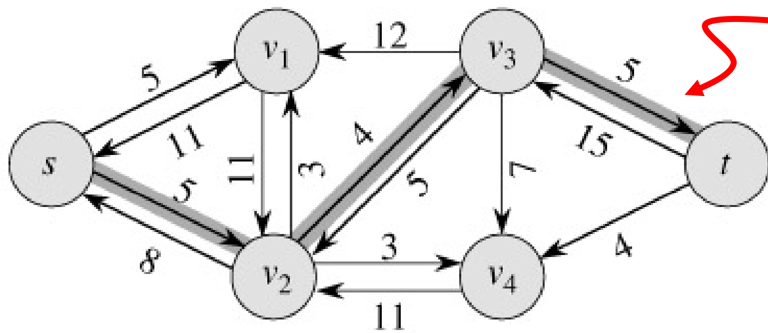
Example

Flow Network



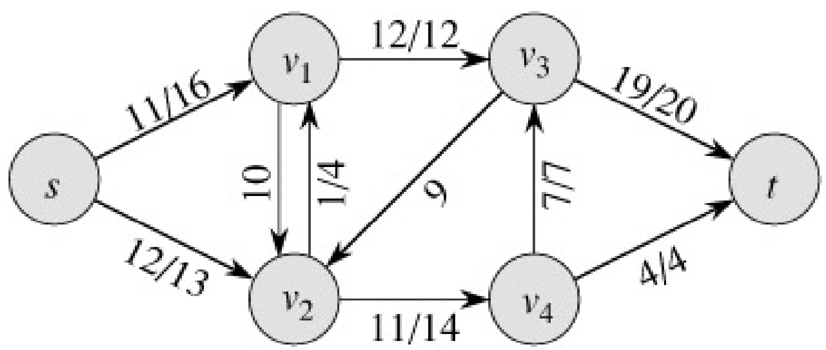
Resulting Flow = 19

Residual Network



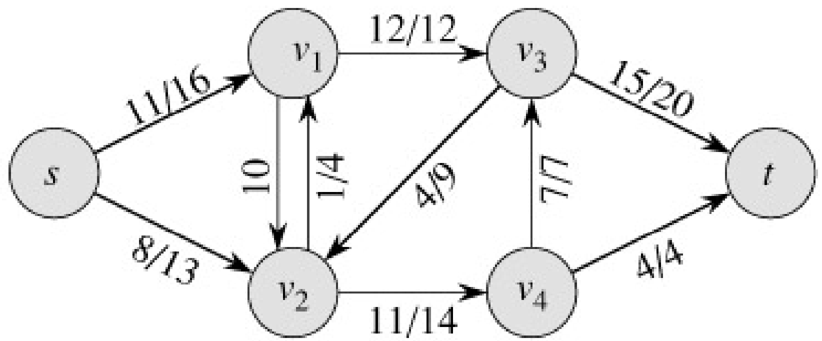
augmenting path

Flow Network



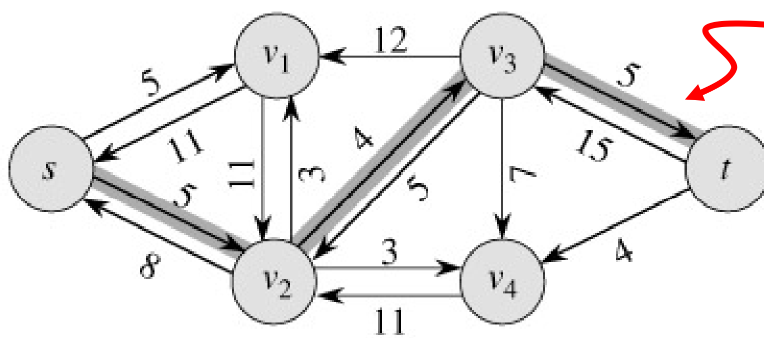
Example

Flow Network



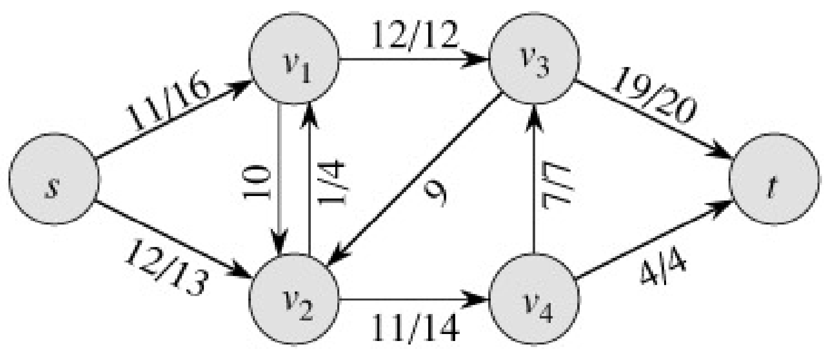
Resulting Flow = 19

Residual Network



augmenting path

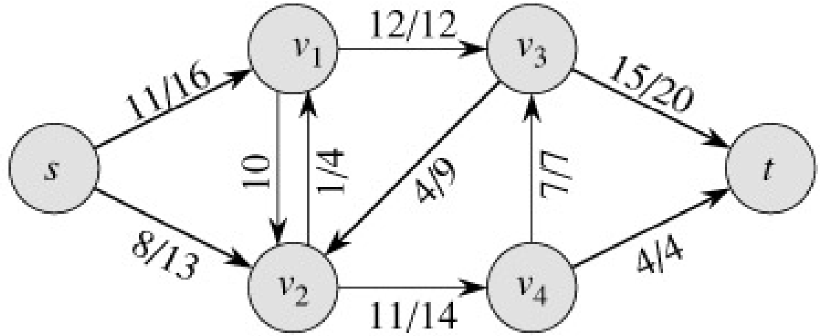
Flow Network



Resulting Flow = ?

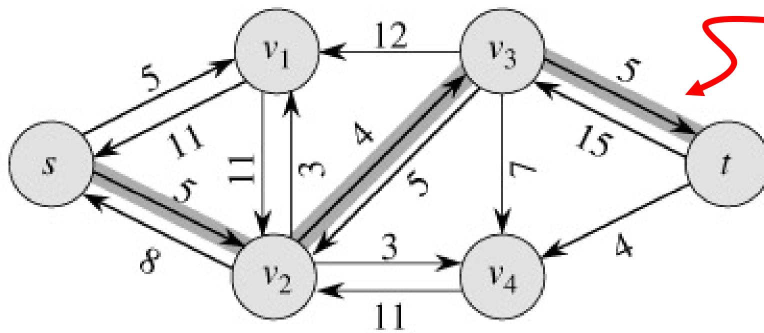
Example

Flow Network



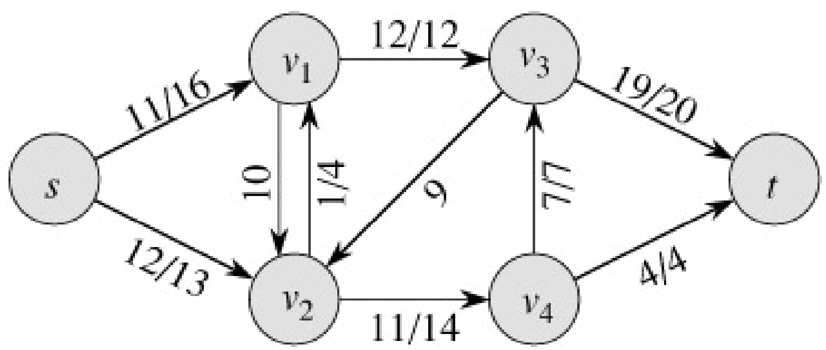
Resulting Flow = 19

Residual Network



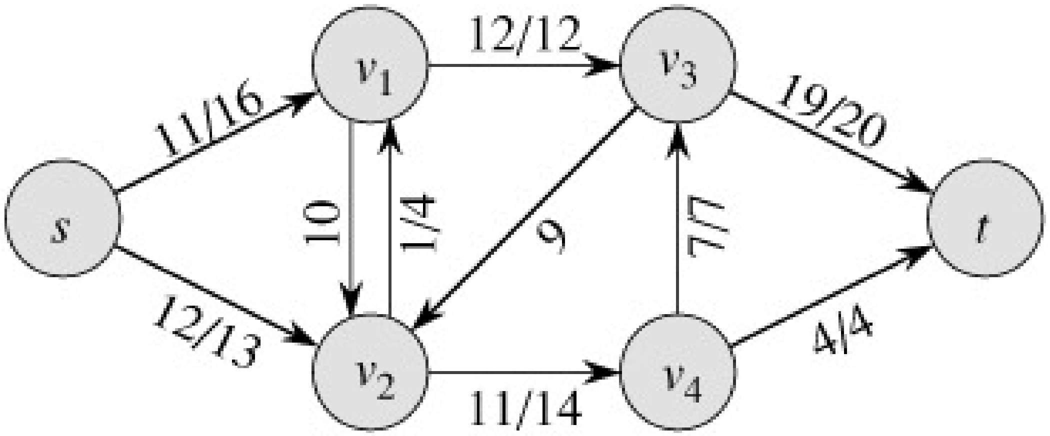
augmenting path

Flow Network



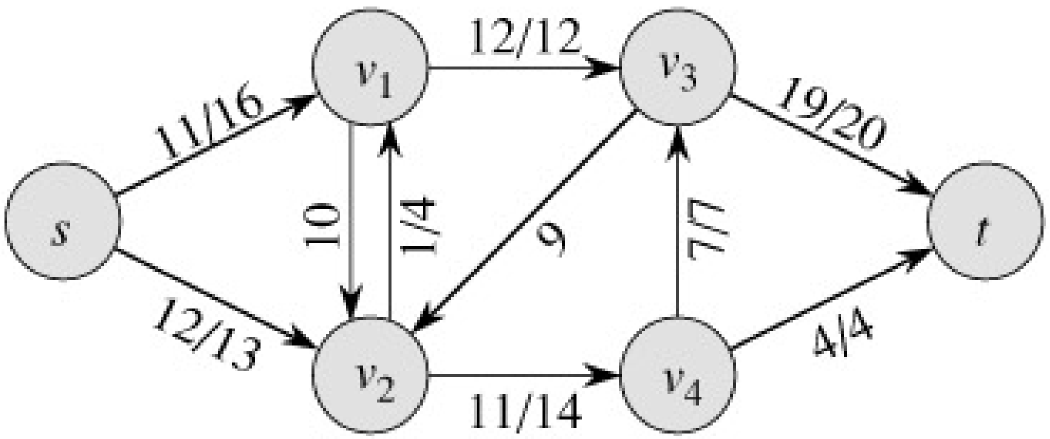
Resulting Flow = 23

Example

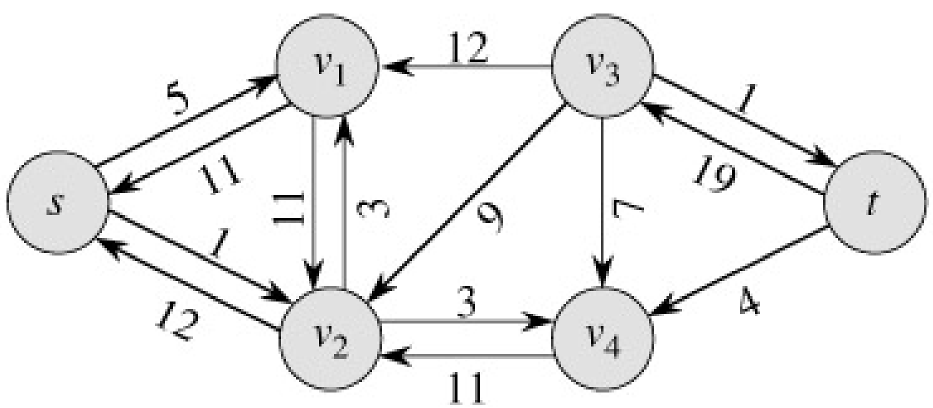


Resulting Flow = 23

Example

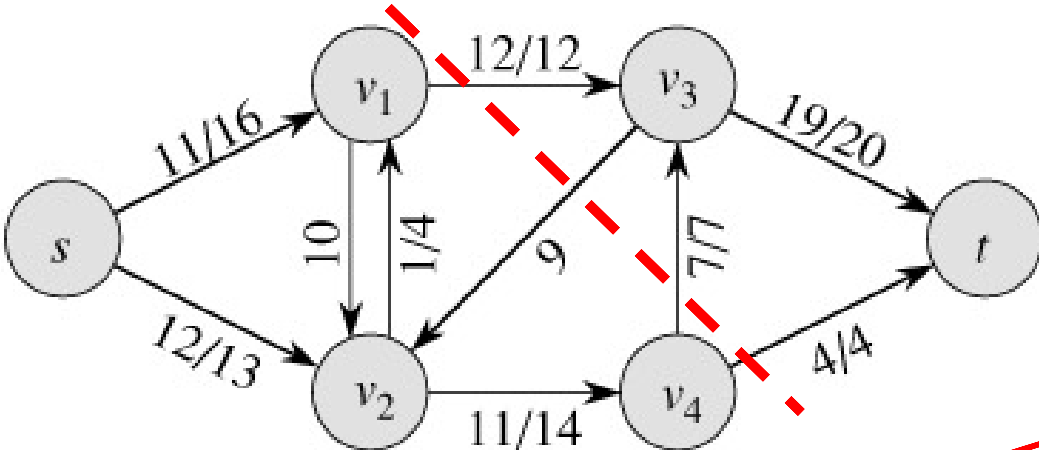


Resulting
Flow = 23



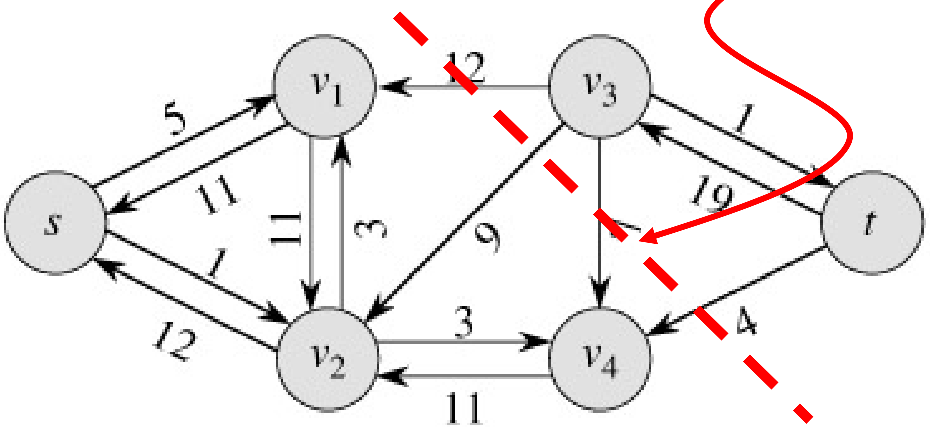
Residual Network

Example



Resulting Flow = 23

. .



No augmenting path:

Maxflow=23

Residual Network

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                                   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)
               f[v, u] ← −f[u, v]
```

Analysis

FORD-FULKERSON(G, s, t)

for each edge $(u, v) \in E[G]$
  do $f[u, v] \leftarrow 0$
    $f[v, u] \leftarrow 0$

O(E)

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 $f[u, v] \leftarrow f[u, v] + c_f(p)$
        $f[v, u] \leftarrow -f[u, v]$

O(E)

Analysis

FORD-FULKERSON(G, s, t)

for each edge $(u, v) \in E[G]$
    do $f[u, v] \leftarrow 0$
        $f[v, u] \leftarrow 0$

O(E)

?

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 $f[u, v] \leftarrow f[u, v] + c_f(p)$
                $f[v, u] \leftarrow -f[u, v]$

O(E)

Analysis

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

Analysis

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

- If max flow is f^*, then need $\leq |f^*|$ iterations $\rightarrow$ time is $O(E|f^*|)$.

Analysis

- If capacities are all integer, then each augmenting path raises $|f|$ by ≥ 1.
- If max flow is f^*, then need $\leq |f^*|$ iterations → 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 ≥ 1.

- If max flow is f^*, then need $\leq |f^*|$ iterations → time is $O(E|f^*|)$.

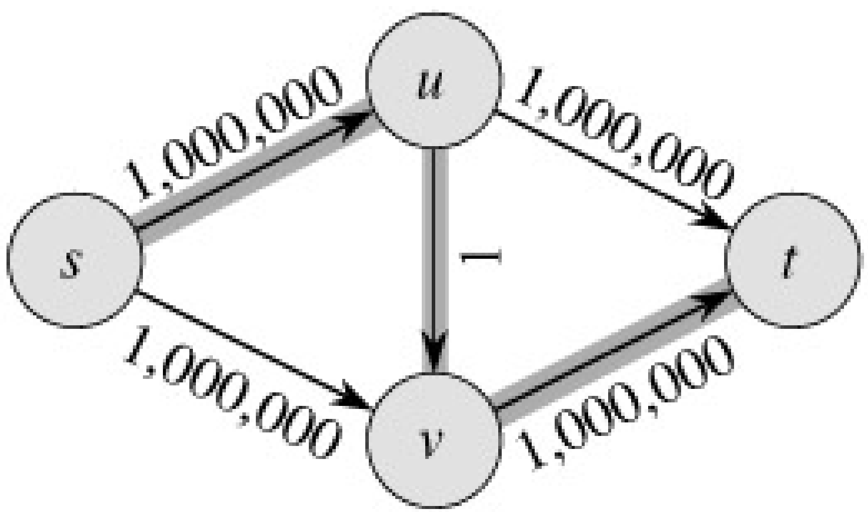
- Note that this running time is **not polynomial** in input size. It depends on $|f^*|$, which is not a function of $|V|$ or $|E|$.

- If capacities are rational, can scale them to integers.

- If irrational, FORD-FULKERSON might never terminate!

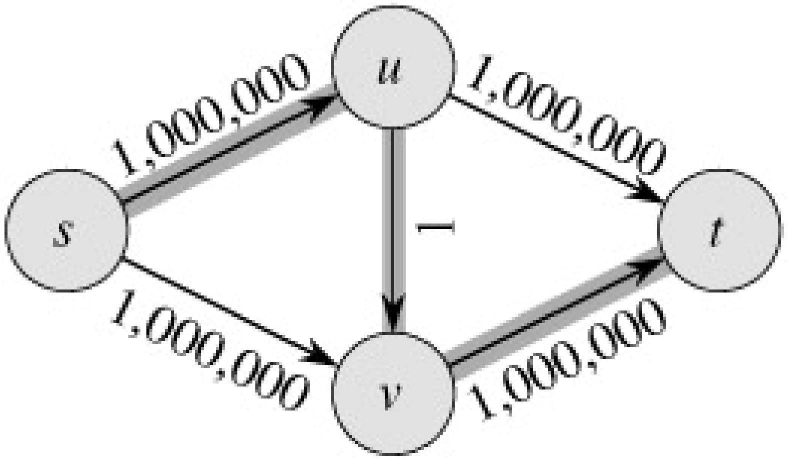
The Basic Ford-Fulkerson Algorithm

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

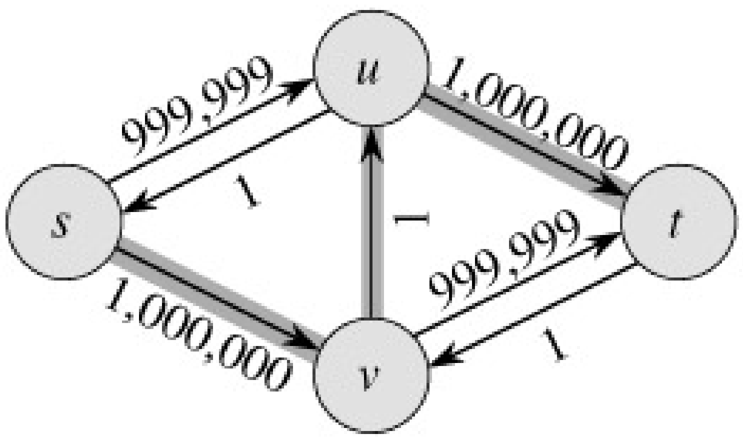


|f*|=2,000,000

Run Ford-Fulkerson on this example

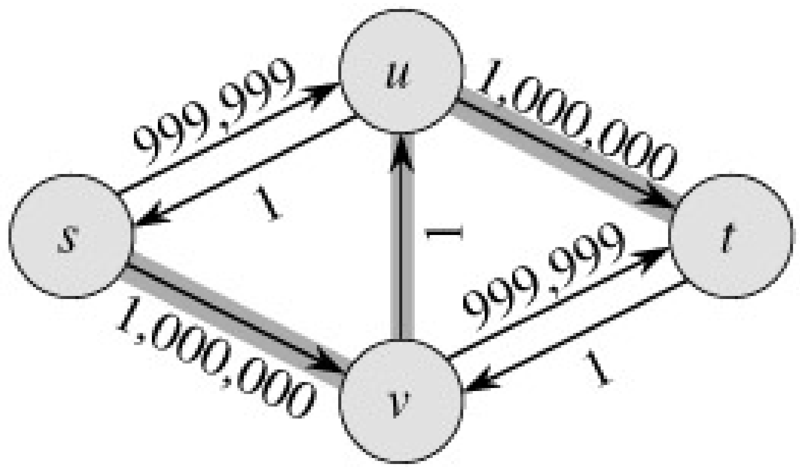


Augmenting Path

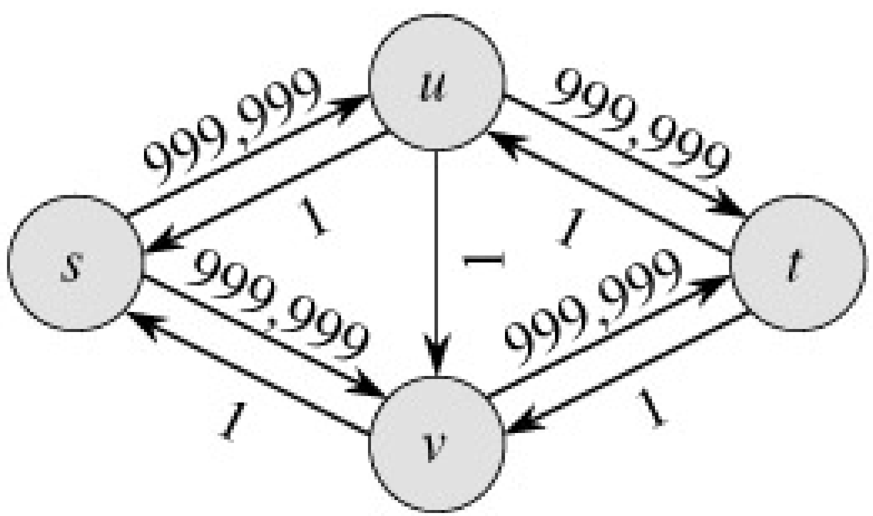


Residual Network

Run Ford-Fulkerson on this example

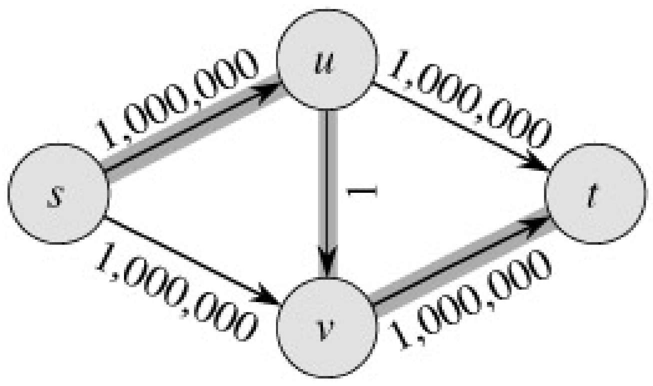


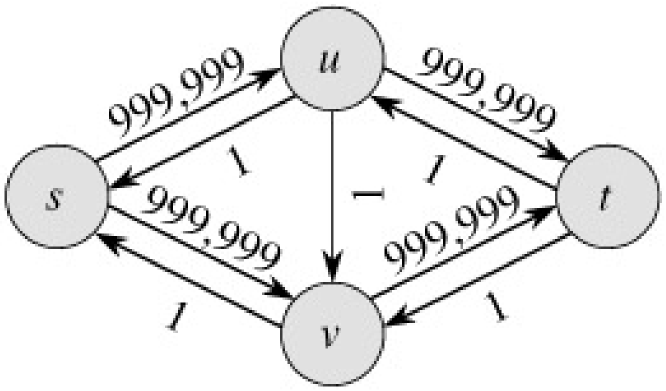
Augmenting Path



Residual Network

Run Ford-Fulkerson on this example





- Repeat 999,999 more times…
- Can we do better than this?

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.
- Select the augmenting path using **breadth-first search** on residual network.

```
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]
```

The Edmonds-Karp Algorithm

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

FORD-FULKERSON(G, s, t)

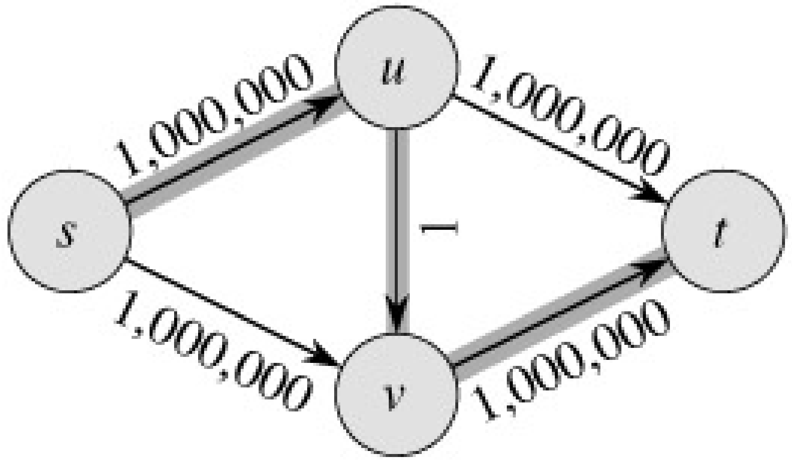
for each edge $(u, v) \in E[G]$
    do $f[u, v] \leftarrow 0$
        $f[v, u] \leftarrow 0$
while there exists a path p from s to t in the residual network G_f
    do $c_f(p) \leftarrow \min\{c_f(u, v) : (u, v) \text{ is in } p\}$
        for each edge (u, v) in p
            do $f[u, v] \leftarrow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

The Edmonds-Karp Algorithm

- A small fix to the Ford-Fulkerson algorithm makes it work in polynomial time.
- Select the augmenting path 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 $O(V E^2)$.

```
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]
```

The Edmonds-Karp Algorithm - example



- The Edmonds-Karp algorithm halts in only 2 iterations on this graph.

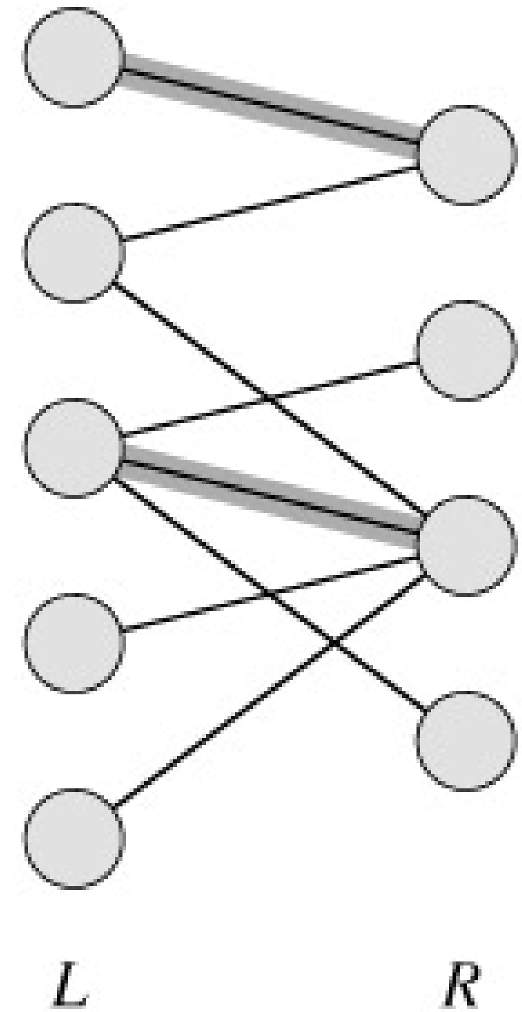
Further Improvements

- Push-relabel algorithm – $O(V^2 E)$.
- The relabel-to-front algorithm – $O(V^3)$.
- (We will not cover these)

An Application of Max Flow:

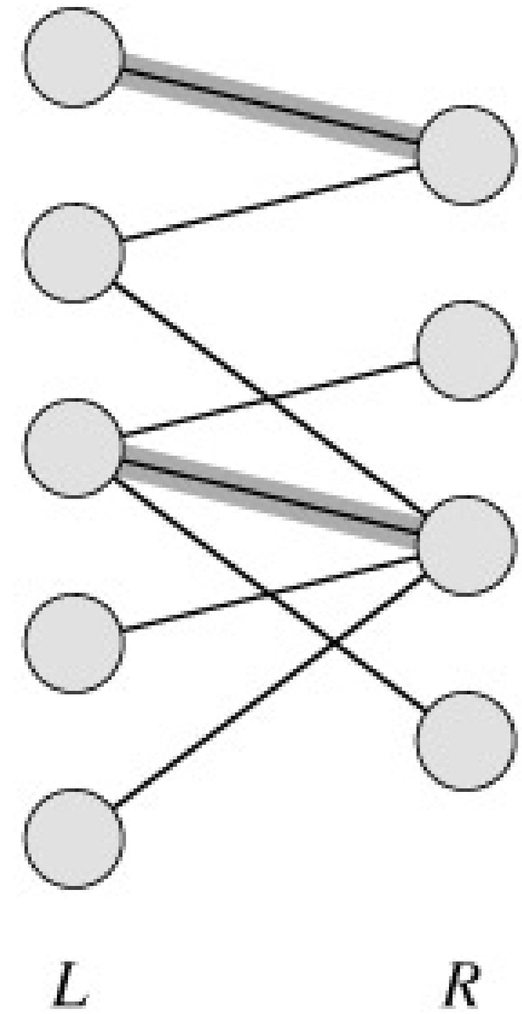
Maximum Bipartite Matching

Maximum Bipartite Matching



Maximum Bipartite Matching

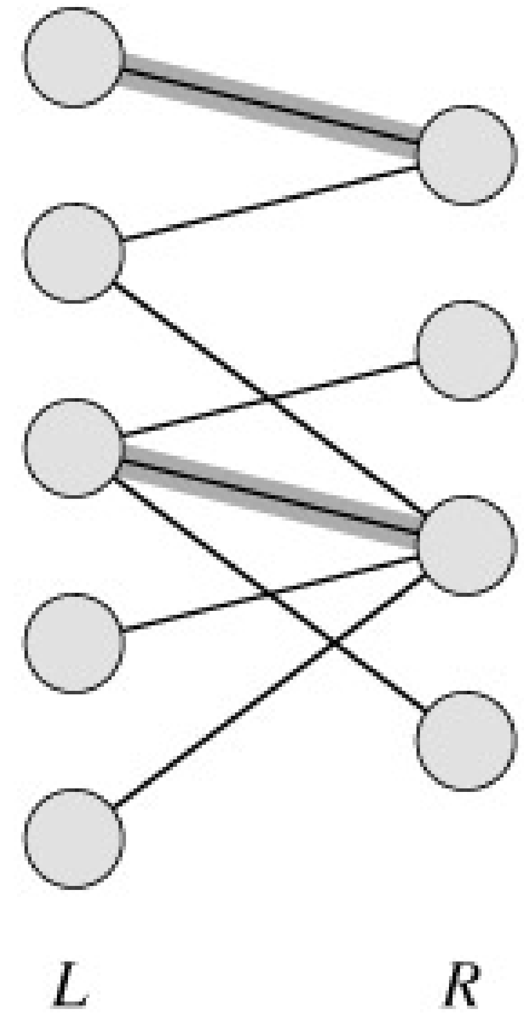
- A bipartite graph is a graph $G=(V,E)$ in which V can be divided into two parts L and R such that every edge in E is between a vertex in L and a vertex in R.



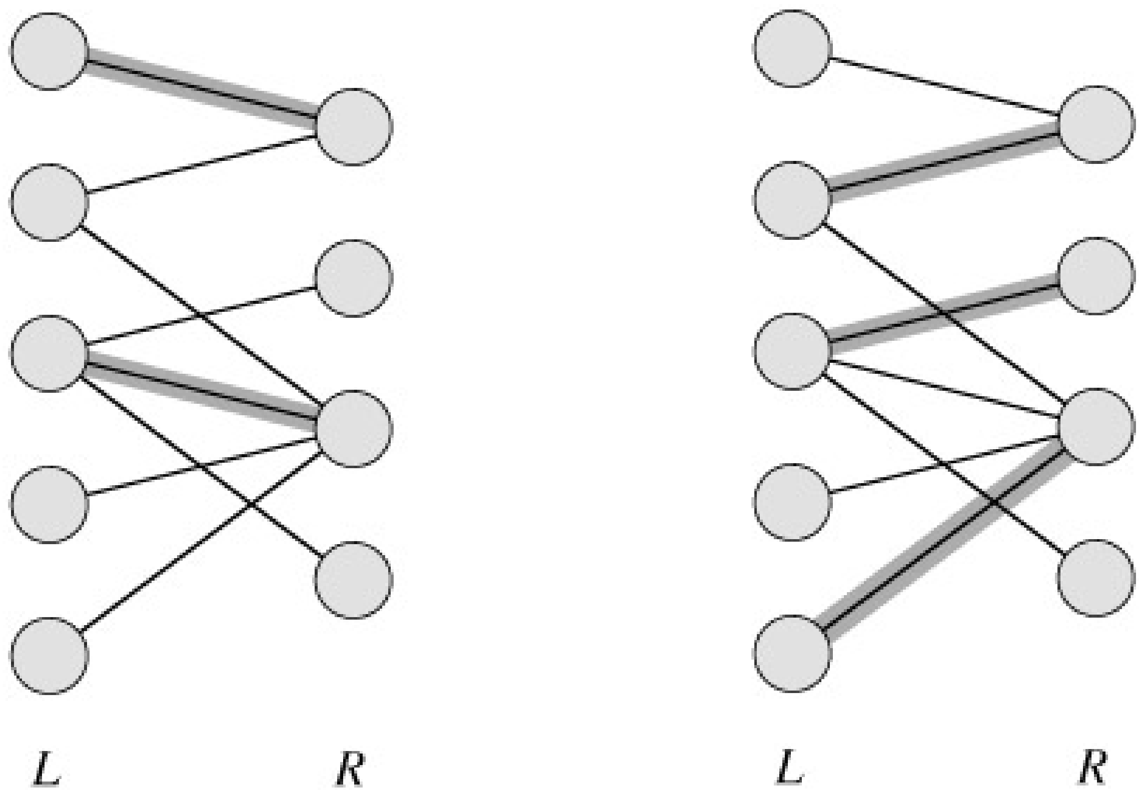
Maximum Bipartite Matching

- A bipartite graph is a graph $G=(V,E)$ in which V can be divided into two parts L and R such that every edge in E is between a vertex in L and a vertex in R.

- e.g. vertices in L represent skilled workers and vertices in R represent jobs. An edge connects workers to jobs they can perform.

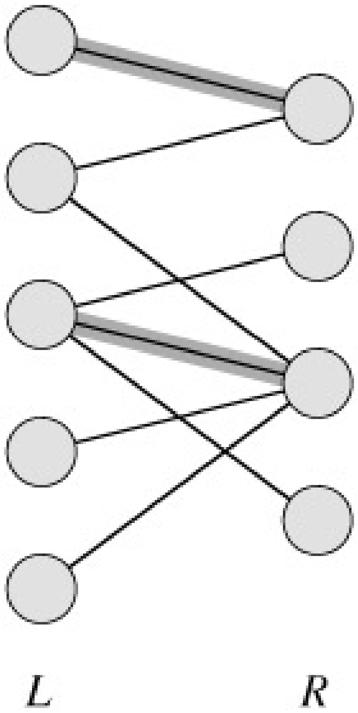


- A matching in a graph is a subset M of E, such that for all vertices v in V, at most one edge of M is incident on v.

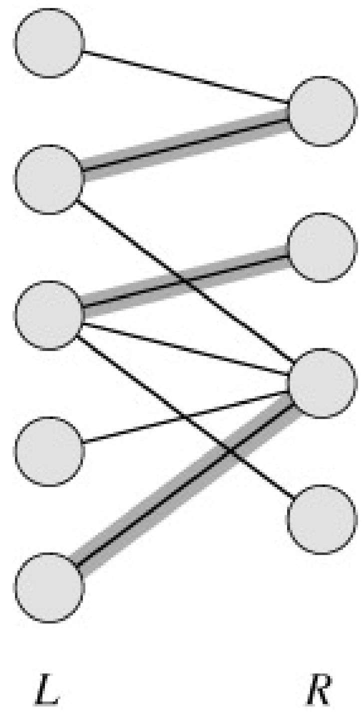


- A **maximum matching** is a matching of maximum cardinality (maximum number of edges).

not maximum

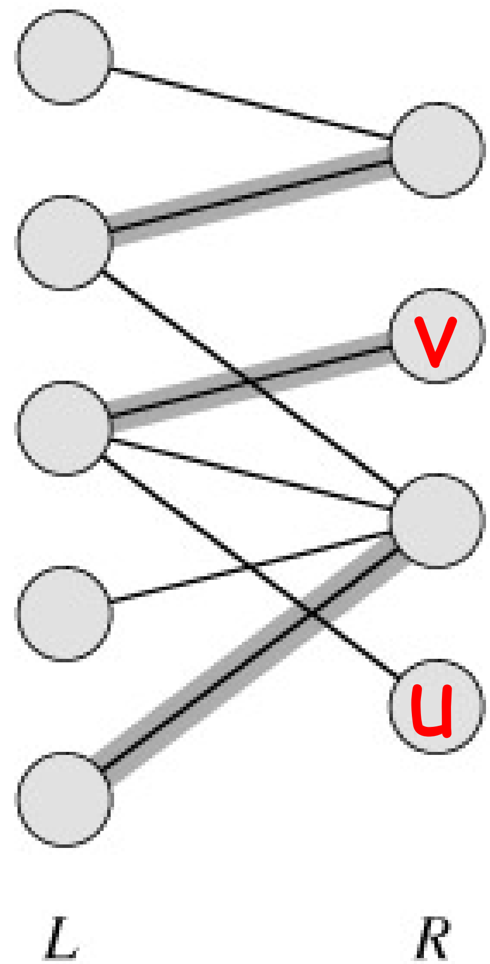


maximum



A Maximum Matching

- No matching of cardinality 4, because only one of v and u can be matched.
- In the workers-jobs example a max-matching provides work for as many people as possible.



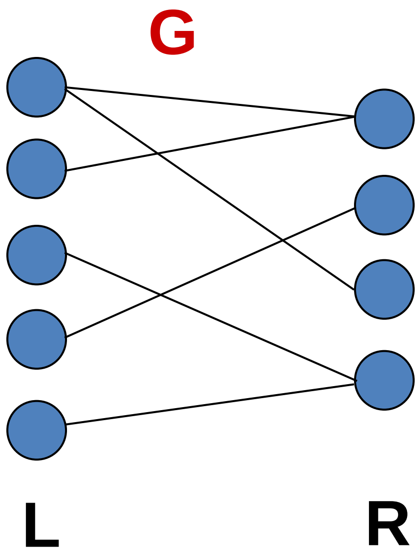
Solving the Maximum Bipartite Matching Problem

- Reduce the maximum bipartite matching problem on graph **G** to the max-flow problem on a corresponding flow network **G'**.

- Solve using Ford-Fulkerson method.

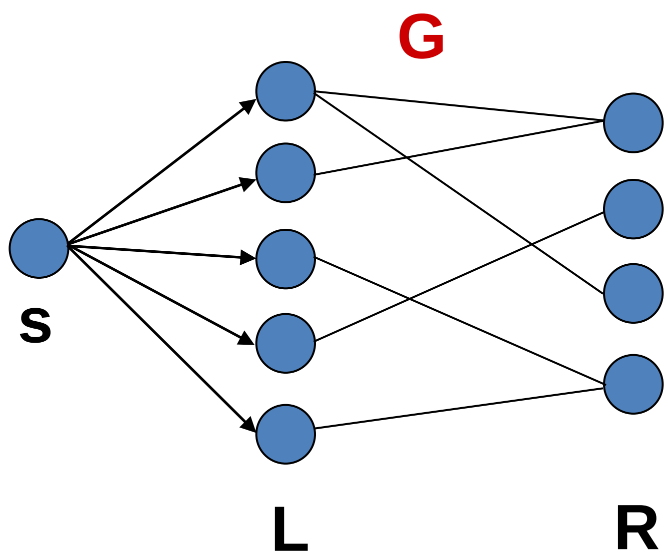
Corresponding Flow Network

- To form the corresponding flow network **G'** of the bipartite graph **G**:



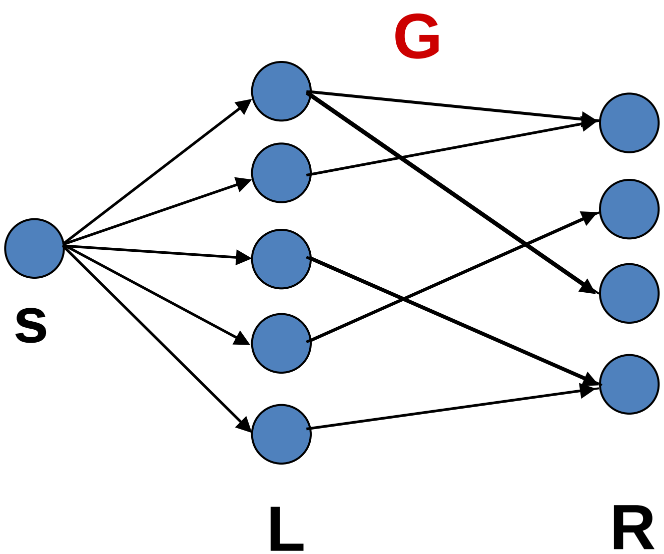
Corresponding Flow Network

- To form the corresponding flow network **G'** of the bipartite graph **G**:
  - Add a source vertex s and edges from s to L.



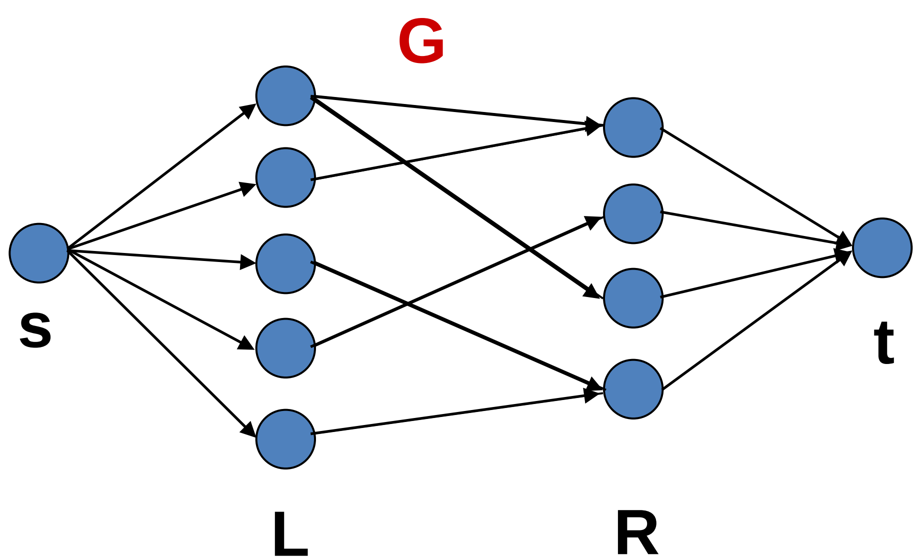
Corresponding Flow Network

- To form the corresponding flow network **G'** of the bipartite graph **G**:
  - Add a source vertex s and edges from s to L.
  - Direct the edges in E from L to R.



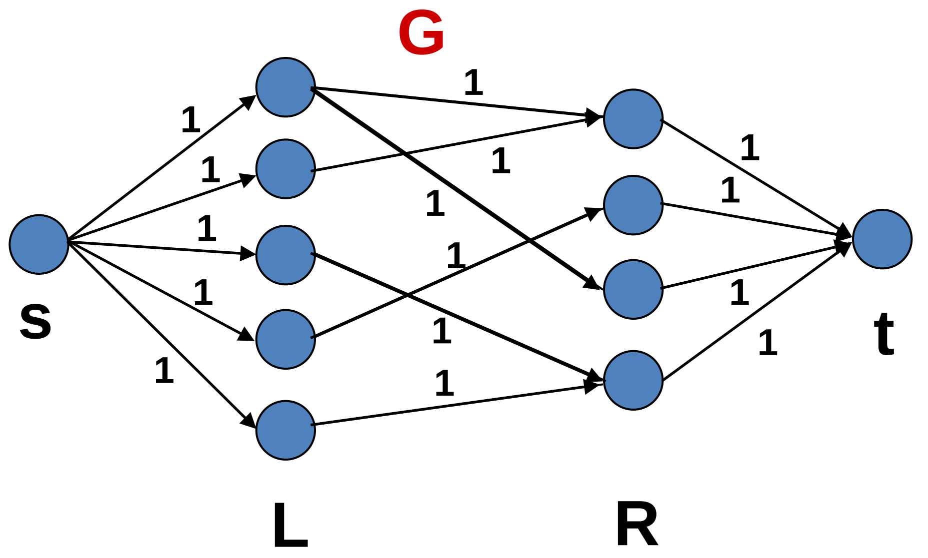
Corresponding Flow Network

- To form the corresponding flow network **G'** of the bipartite graph **G**:
  - Add a source vertex s and edges from s to L.
  - Direct the edges in E from L to R.
  - Add a sink vertex t and edges from R to t.



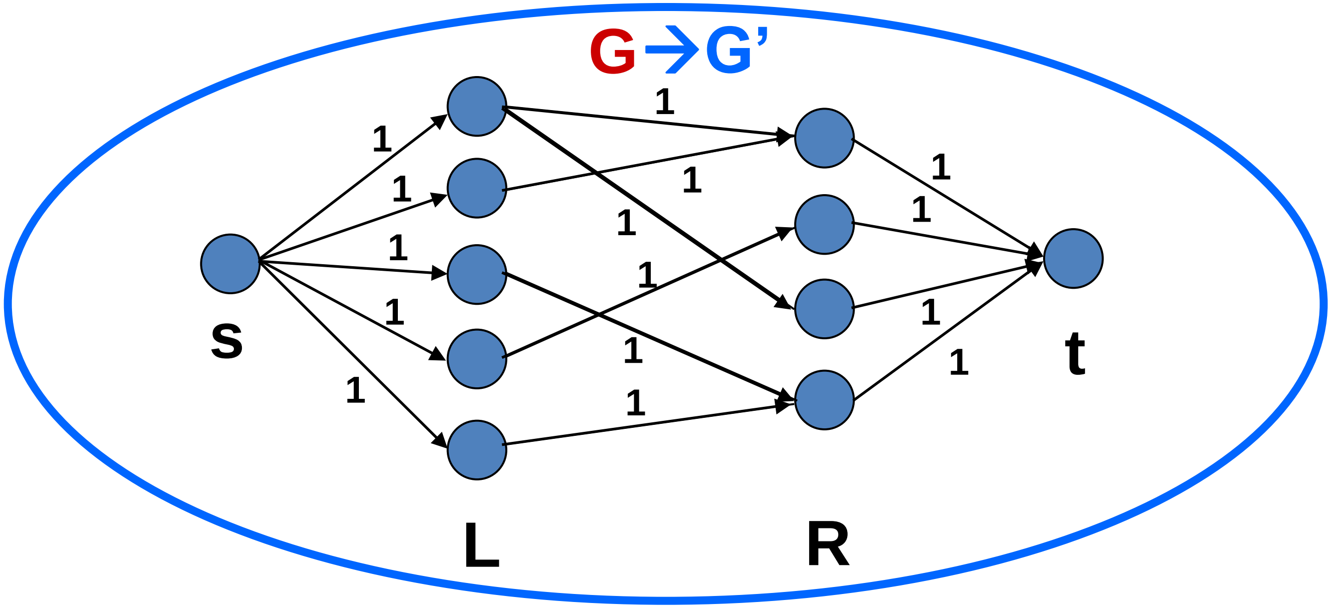
Corresponding Flow Network

- To form the corresponding flow network **G'** of the bipartite graph **G**:
  - Add a source vertex s and edges from s to L.
  - Direct the edges in E from L to R.
  - Add a sink vertex t and edges from R to t.
  - Assign a capacity of 1 to all edges.



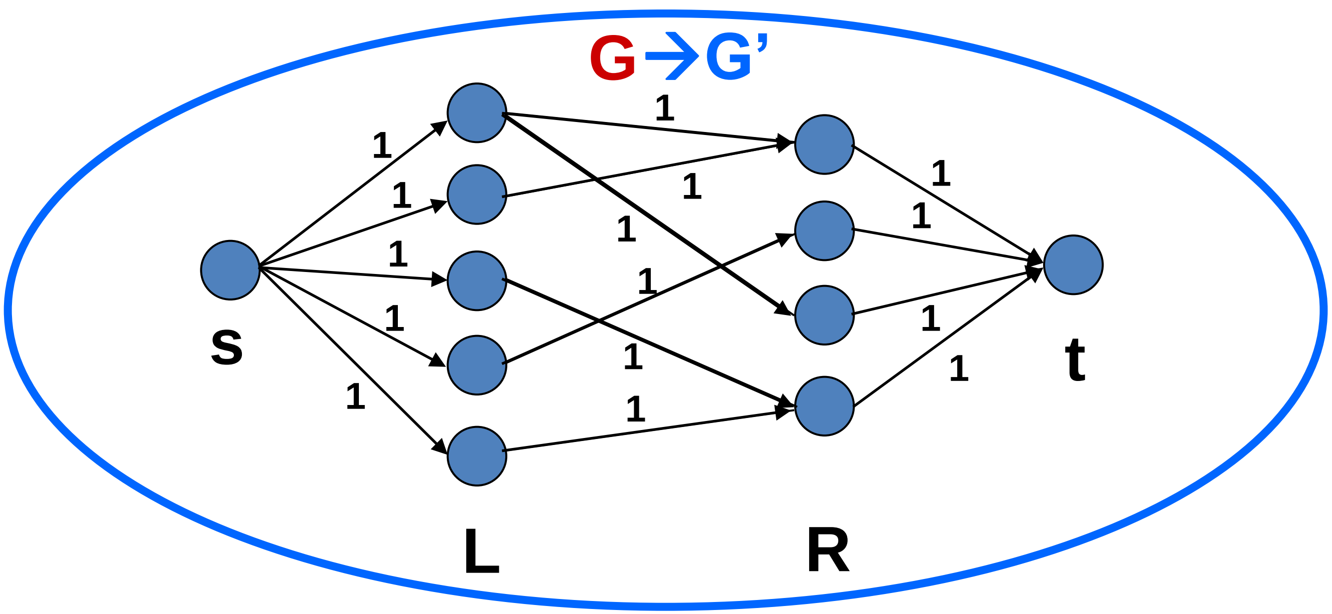
Corresponding Flow Network

- To form the corresponding flow network **G'** of the bipartite graph **G**:
  - Add a source vertex s and edges from s to L.
  - Direct the edges in E from L to R.
  - Add a sink vertex t and edges from R to t.
  - Assign a capacity of 1 to all edges.



Corresponding Flow Network

- To form the corresponding flow network **G'** of the bipartite graph **G**:
  - Add a source vertex s and edges from s to L.
  - Direct the edges in E from L to R.
  - Add a sink vertex t and edges from R to t.
  - Assign a capacity of 1 to all edges.
- Claim: max-flow in **G'** corresponds to a max-bipartite-matching on **G**.



Solving Bipartite Matching as Max Flow

Let $G = (V, E)$ be a bipartite graph with vertex partition $V = L \cup R$.

Let $G' = (V', E')$ be its corresponding flow network.

Solving Bipartite Matching as Max Flow

Let $G = (V, E)$ be a bipartite graph with vertex partition $V = L \cup R$.

Let $G' = (V', E')$ be its corresponding flow network.

If M is a matching in G,

then there is an integer-valued flow f in G' with value $|f| = |M|$.

Solving Bipartite Matching as Max Flow

Let $G=(V,E)$ be a bipartite graph with vertex partition $V=L\cup R$.

Let $G'=(V',E')$ be its corresponding flow network.

If M is a matching in G,

then there is an integer-valued flow f in G' with value $|f|=|M|$.

Conversely if f is an integer-valued flow in G',

then there is a matching M in G with cardinality $|M|=|f|$.

Solving Bipartite Matching as Max Flow

Let $G=(V,E)$ be a bipartite graph with vertex partition $V=L\cup R$.

Let $G'=(V',E')$ be its corresponding flow network.

If M is a matching in G,

then there is an integer-valued flow f in G' with value $|f|=|M|$.

Conversely if f is an integer-valued flow in G',

then there is a matching M in G with cardinality $|M|=|f|$.

Thus $\max|M|=\max(\text{integer } |f|)$

Does this mean that max $|f|$ = max $|M|$?

Does this mean that max |f| = max |M|?

- Problem: we haven't shown that the max flow f(u,v) is necessarily integer-valued.

Integrality Theorem

- If the capacity function c takes on only integral values, then:

Integrality Theorem

- If the capacity function c takes on only integral values, then:
    1. The maximum flow f produced by the Ford-Fulkerson method has the property that |f| is integer-valued.

Integrality Theorem

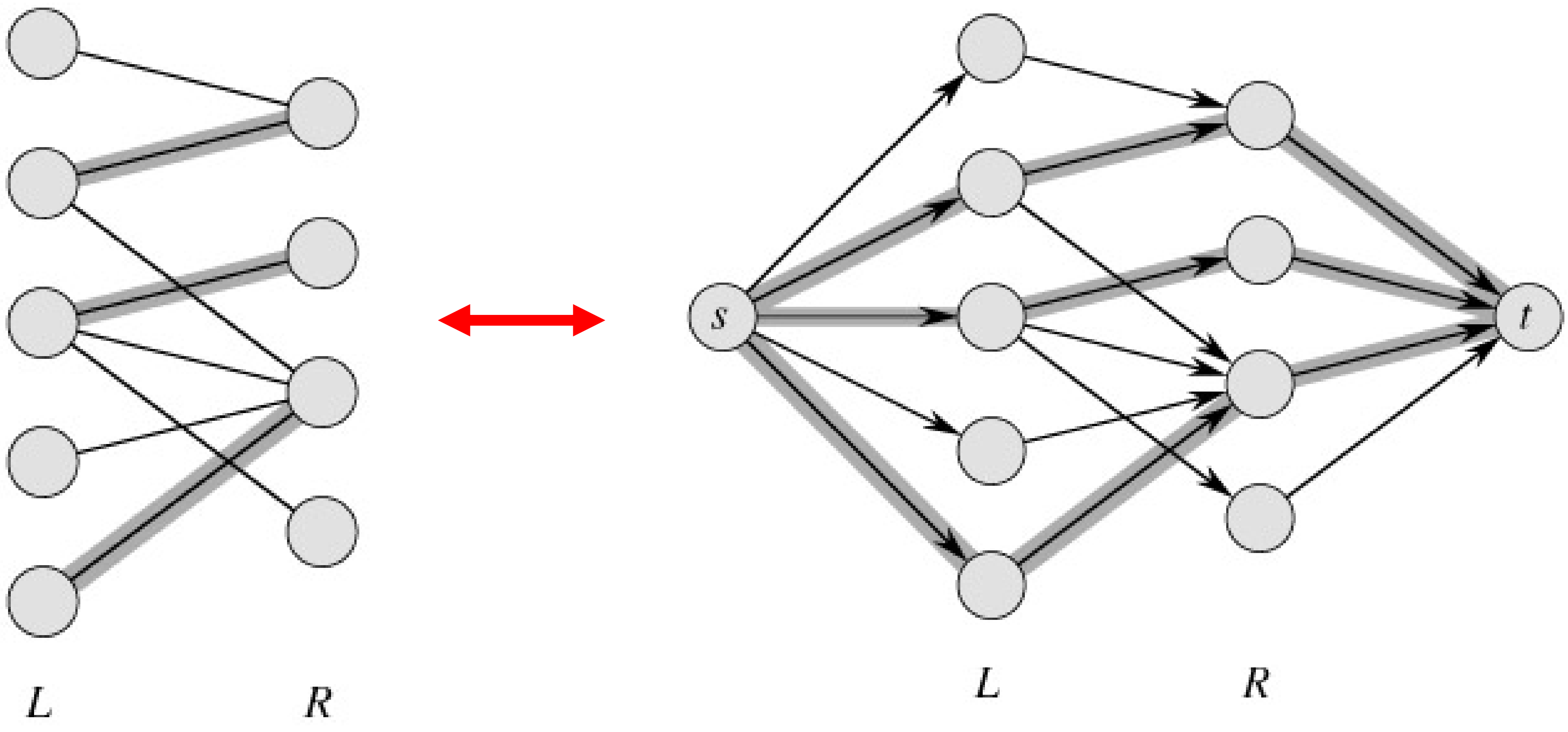
- If the capacity function c takes on only integral values, then:
    1. The maximum flow f produced by the Ford-Fulkerson method has the property that |f| is integer-valued.
    2. For all vertices u and v the value f(u,v) of the flow is an integer.

Integrality Theorem

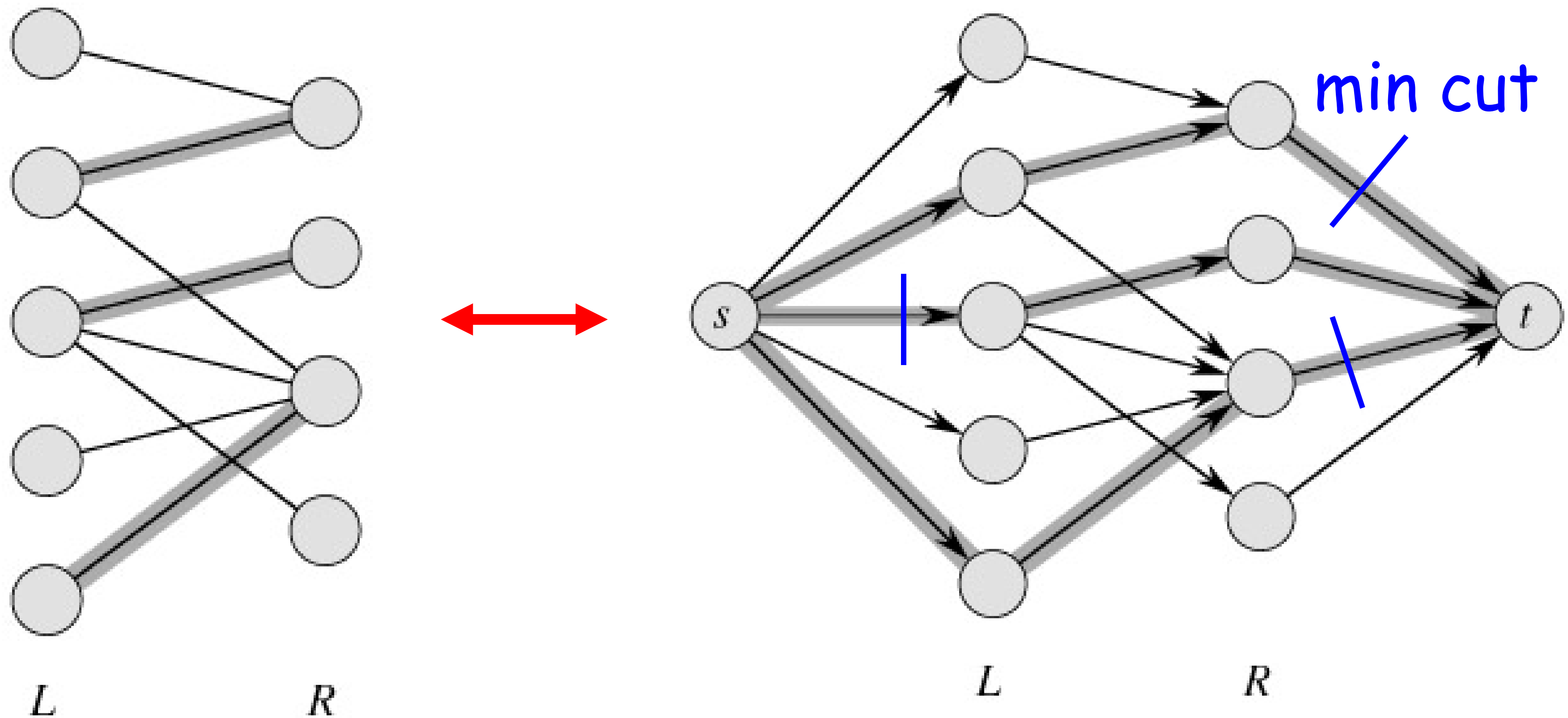
- If the capacity function c takes on only integral values, then:
  1. The maximum flow f produced by the Ford-Fulkerson method has the property that $|f|$ is integer-valued.
  2. For all vertices u and v the value f(u,v) of the flow is an integer.

- So $\max|M| = \max |f|$

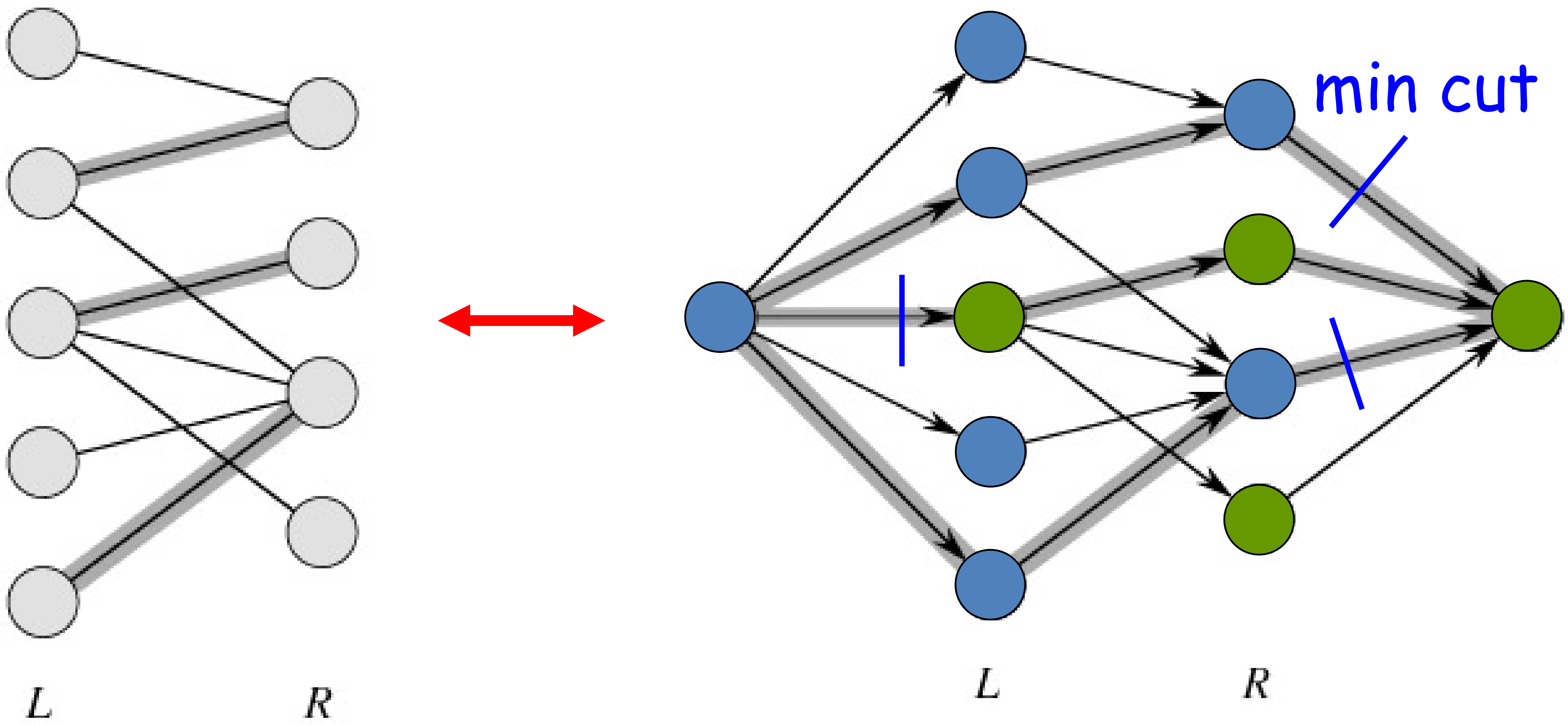
Example



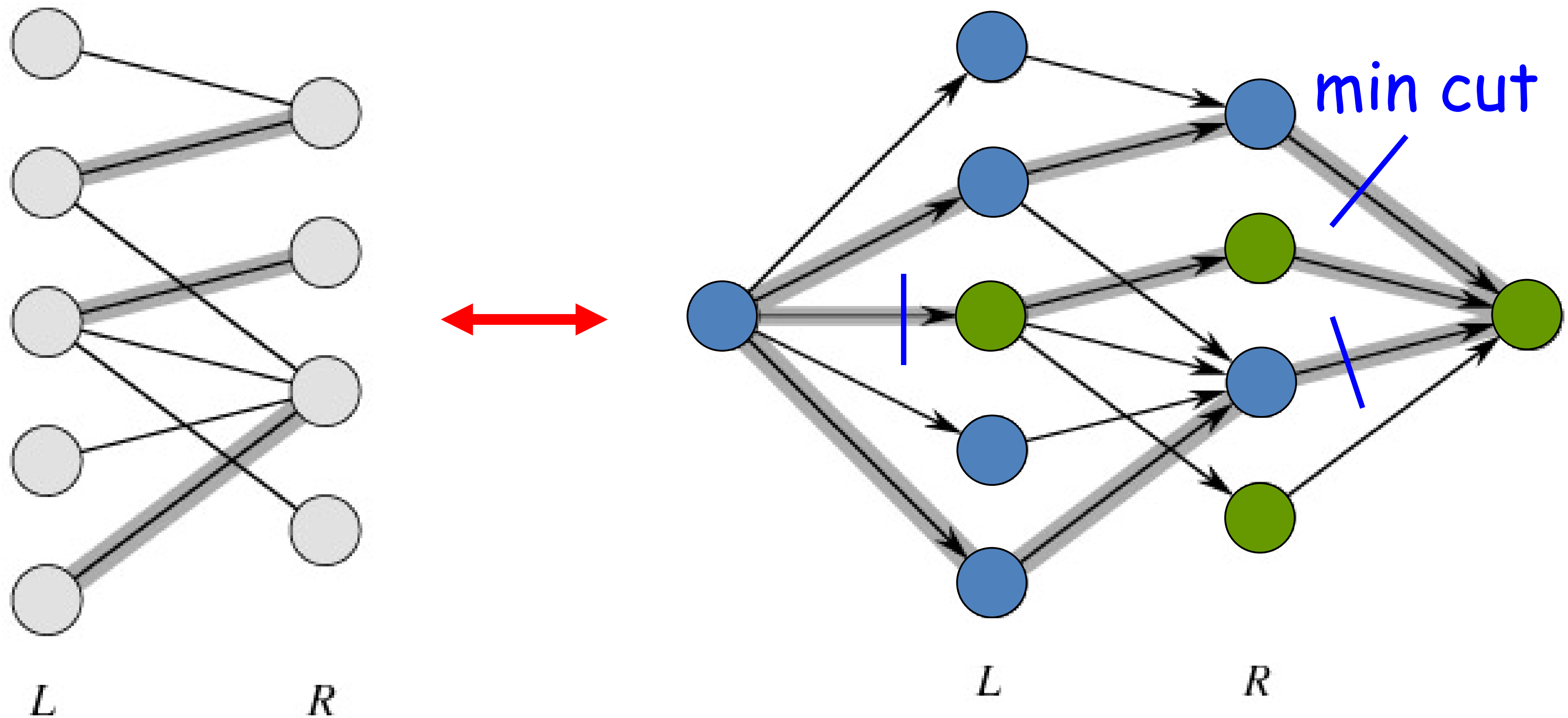
Example



Example



Example



$|M| = 3 \quad \leftrightarrow \quad \text{max flow} = |f| = 3$

Conclusion

- Network flow algorithms allow us to find the maximum bipartite matching fairly easily.
- Similar techniques are applicable in other combinatorial design problems.

Example

- In a department there are n courses and m instructors.
- Every instructor has a list of courses he or she can teach.
- Every instructor can teach at most 3 courses during a year.

- The goal: find an allocation of courses to the instructors subject to these constraints.