Network Flow

Mohammad Javad Dousti

Some slides are courtesy of Dr. Mahini.

Changelog

- Rev. 1
    - Added the missing reverse flow code (slide 37.)
- Rev. 2
    - Removed the unnecessary `return f` at the bottom of the code (slides 25 and 26.) Thanks to Mr. Sedghian.
    - Fixed a typo (slide 44.) Thanks to Mr. Sedghian.

Network Flow

Overview

- Network Flow Problem
- Ford-Fulkerson Algorithm
- Extensions to the Maximum-Flow Problem
  - Matching
  - Multi source/sink
  - Circulation
- Sample problems

Network Flow Problem

Problem Statement

Input: A directed weighted graph $G = (V, E)$, a source s, and a sink t. Assume $c[e]$ is the capacity (weight) of edge $e \in E$.

Flow: a function $f: E \to \mathbb{R}$.

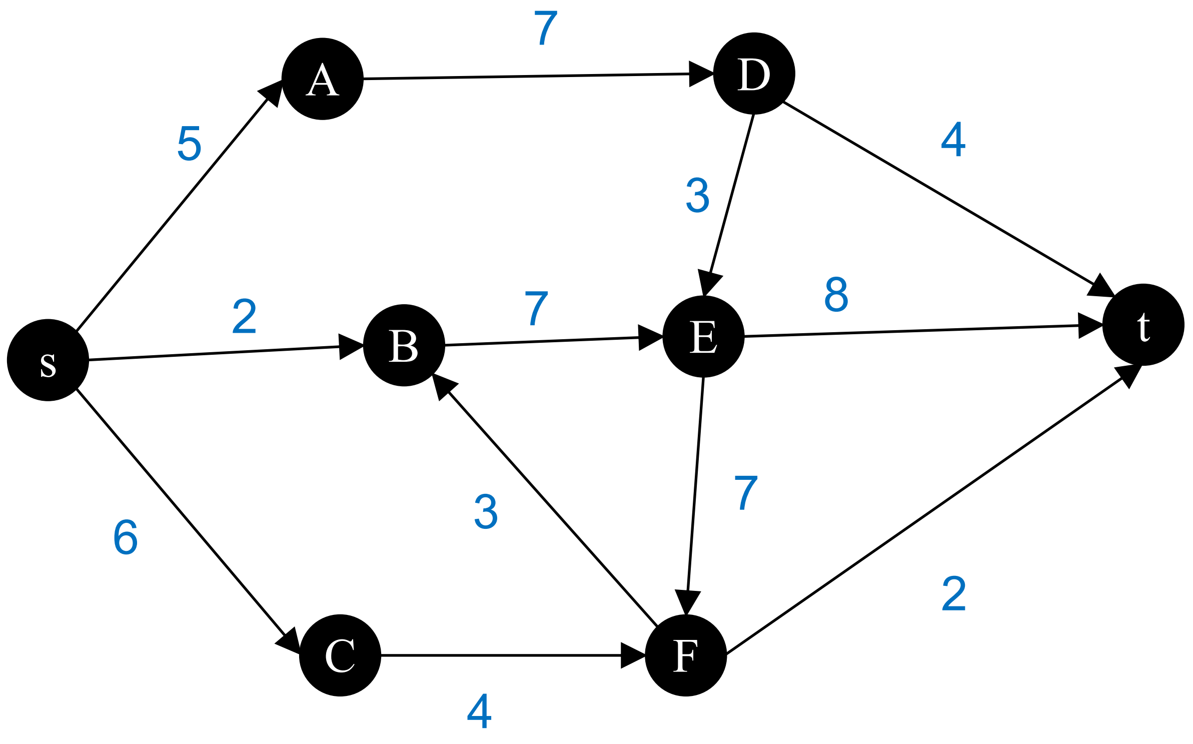
A flow is feasible if:

- ❑ For all edge $e = (u, v)$, we have $f[e] \le c[e]$
- ❑ For all pairs $u, v \in V$, we have $f[u, v] = -f[v, u]$
- ❑ For all node $u \neq s, t$, we have
  - ➢ $\sum_{u \to e} f(e) = \sum_{u \leftarrow e} f(e)$
  - ➢ $\sum_{v \in V} f[u, v] = 0$

Amount of flow is defined as:

- ❑ $|f| = \sum_{s \to e} f(e) - \sum_{s \leftarrow e} f(e) = \sum_{t \leftarrow e} f(e) - \sum_{t \to e} f(e)$
- ❑ $|f| = \sum_{v \in G} f[s, v] = \sum_{v \in G} f[v, t]$

Goal: Find a feasible flow with the maximum possible amount (max flow).



A Feasible Flow (1)

Input: A directed weighted graph $G = (V, E)$, a source s, and a sink t. Assume $c[e]$ is the capacity (weight) of edge $e \in E$.

Flow: a function $f: E \to \mathbb{R}$.

A flow is feasible if:

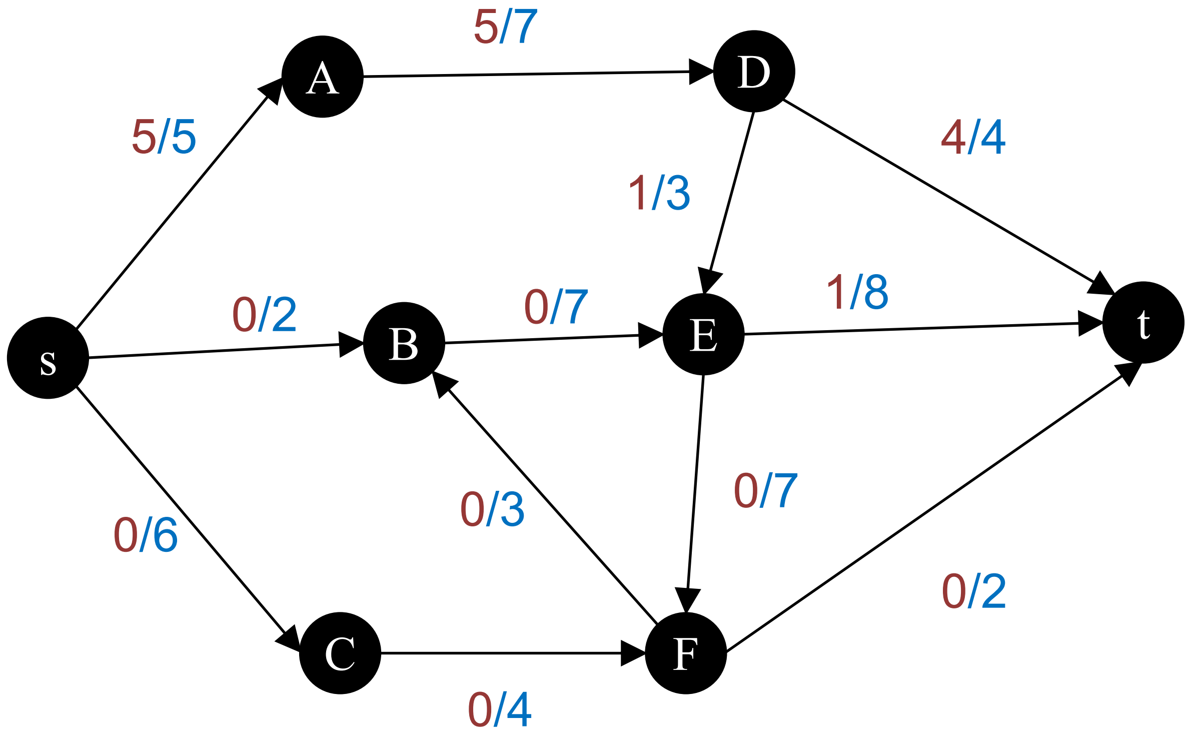
- ❑ For all edge $e = (u, v)$, we have $f[e] \leq c[e]$
- ❑ For all pairs $u, v \in V$, we have $f[u, v] = -f[v, u]$
- ❑ For all node $u \neq s, t$, we have
  - ➢ $\sum_{u \to e} f(e) = \sum_{u \leftarrow e} f(e)$
  - ➢ $\sum_{v \in V} f[u, v] = 0$

Amount of flow is defined as:

- ❑ $|f| = \sum_{s \to e} f(e) - \sum_{s \leftarrow e} f(e) = \sum_{t \leftarrow e} f(e) - \sum_{t \to e} f(e)$
- ❑ $|f| = \sum_{v \in G} f[s, v] = \sum_{v \in G} f[v, t]$

Goal: Find a feasible flow with the maximum possible amount (max flow).

Feasible Flow with $|f| = 5$



A Feasible Flow (2)

Input: A directed weighted graph $G = (V, E)$, a source s, and a sink t. Assume $c[e]$ is the capacity (weight) of edge $e \in E$.

Flow: a function $f: E \to \mathbb{R}$.

A flow is feasible if:

- For all edge $e = (u, v)$, we have $f[e] \leq c[e]$
- For all pairs $u, v \in V$, we have $f[u, v] = -f[v, u]$
- For all node $u \neq s, t$, we have
  - $\sum_{u \to e} f(e) = \sum_{u \leftarrow e} f(e)$
  - $\sum_{v \in V} f[u, v] = 0$

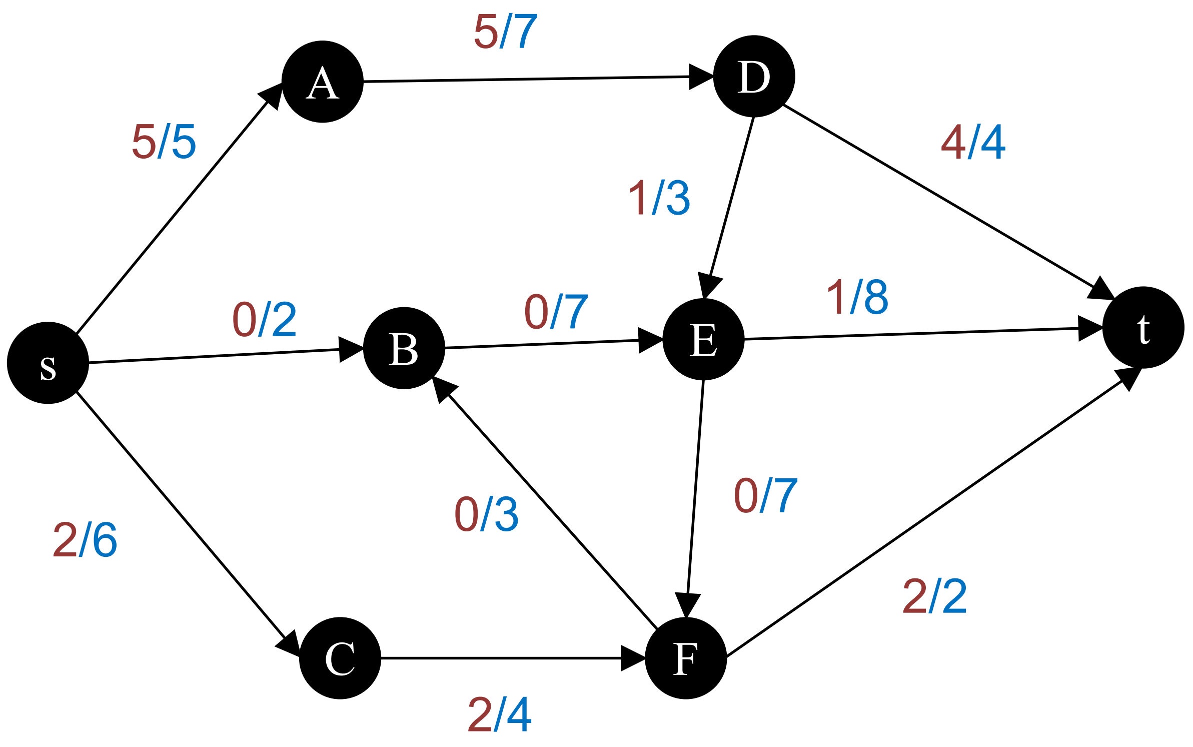
Amount of flow is defined as:

- $|f| = \sum_{s \to e} f(e) - \sum_{s \leftarrow e} f(e) = \sum_{t \leftarrow e} f(e) - \sum_{t \to e} f(e)$
- $|f| = \sum_{v \in G} f[s, v] = \sum_{v \in G} f[v, t]$

Goal: Find a feasible flow with the maximum possible amount (max flow).

Another Feasible Flow with $|f| = 7$

$f[C, F] = 2, \qquad f[F, C] = -2$



A Feasible Flow (3)

Input: A directed weighted graph $G = (V, E)$, a source s, and a sink t. Assume $c[e]$ is the capacity (weight) of edge $e \in E$.

Flow: a function $f: E \to \mathbb{R}$.

A flow is feasible if:

- ❑ For all edge $e = (u, v)$, we have $f[e] \leq c[e]$
- ❑ For all pairs $u, v \in V$, we have $f[u, v] = -f[v, u]$
- ❑ For all node $u \neq s, t$, we have
  - ➢ $\sum_{u \to e} f(e) = \sum_{u \leftarrow e} f(e)$
  - ➢ $\sum_{v \in V} f[u, v] = 0$

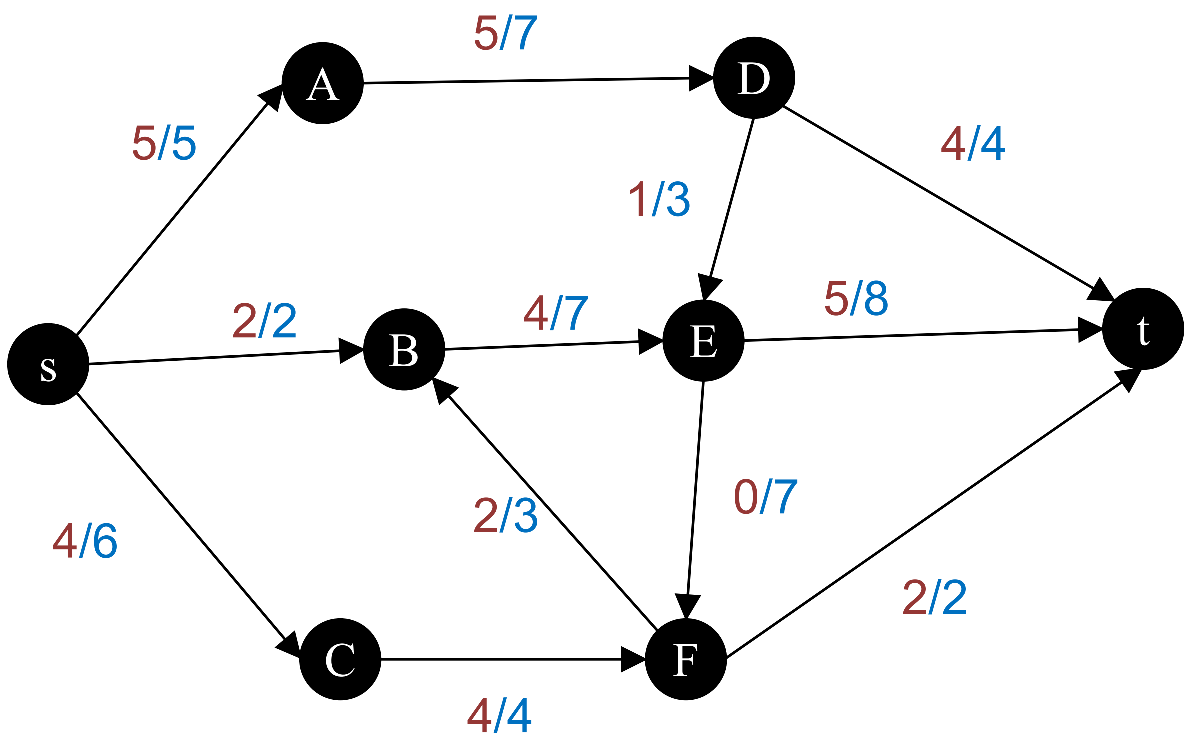
Amount of flow is defined as:

- ❑ $|f| = \sum_{s \to e} f(e) - \sum_{s \leftarrow e} f(e) = \sum_{t \leftarrow e} f(e) - \sum_{t \to e} f(e)$
- ❑ $|f| = \sum_{v \in G} f[s, v] = \sum_{v \in G} f[v, t]$

Goal: Find a feasible flow with the maximum possible amount (max flow).

Another Feasible Flow with $|f| = 11$

Is it a max flow?



A Feasible Flow (4)

Input: A directed weighted graph $G = (V, E)$, a source s, and a sink t. Assume $c[e]$ is the capacity (weight) of edge $e \in E$. Assume capacities are integer.

For simplicity

Flow: a function $f: E \to \mathbb{R}$.

A flow is feasible if:

- For all edge $e = (u, v)$, we have $f[e] \le c[e]$
- For all pairs $u, v \in V$, we have $f[u, v] = -f[v, u]$
- For all node $u \neq s, t$, we have
  - $\sum_{u \to e} f(e) = \sum_{u \leftarrow e} f(e)$
  - $\sum_{v \in V} f[u, v] = 0$

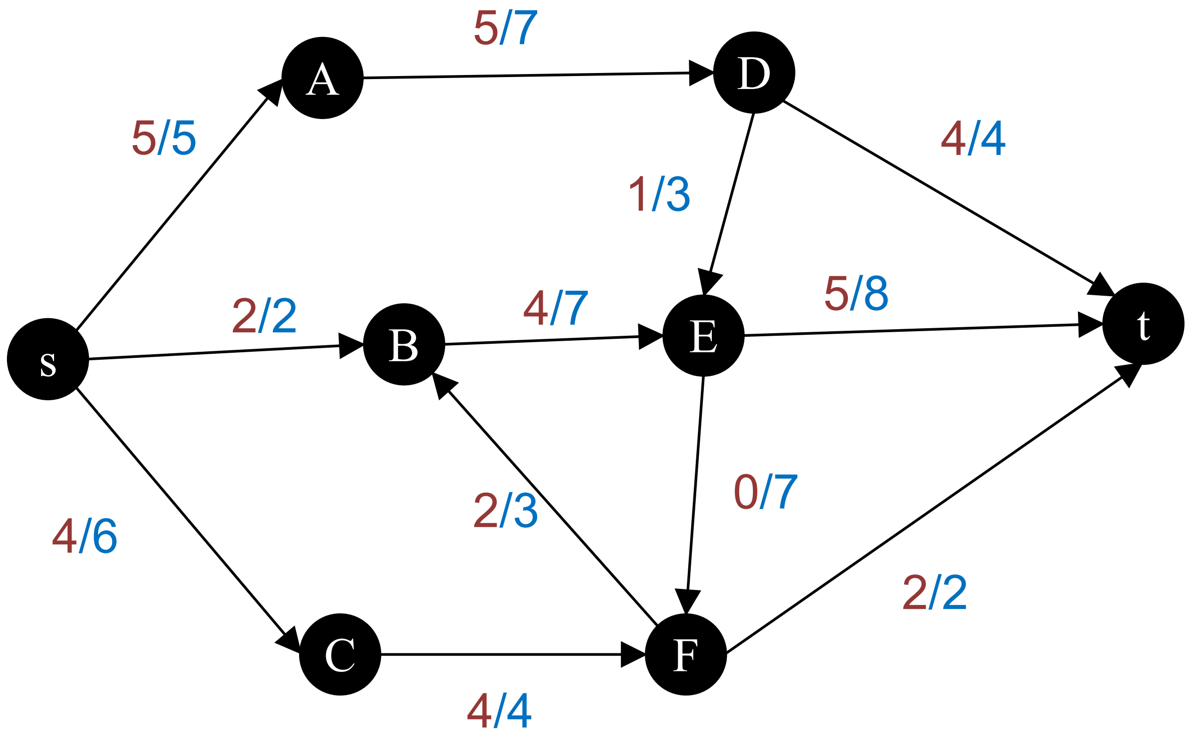
Amount of flow is defined as:

- $|f| = \sum_{s \to e} f(e) - \sum_{s \leftarrow e} f(e) = \sum_{t \leftarrow e} f(e) - \sum_{t \to e} f(e)$
- $|f| = \sum_{v \in G} f[s, v] = \sum_{v \in G} f[v, t]$

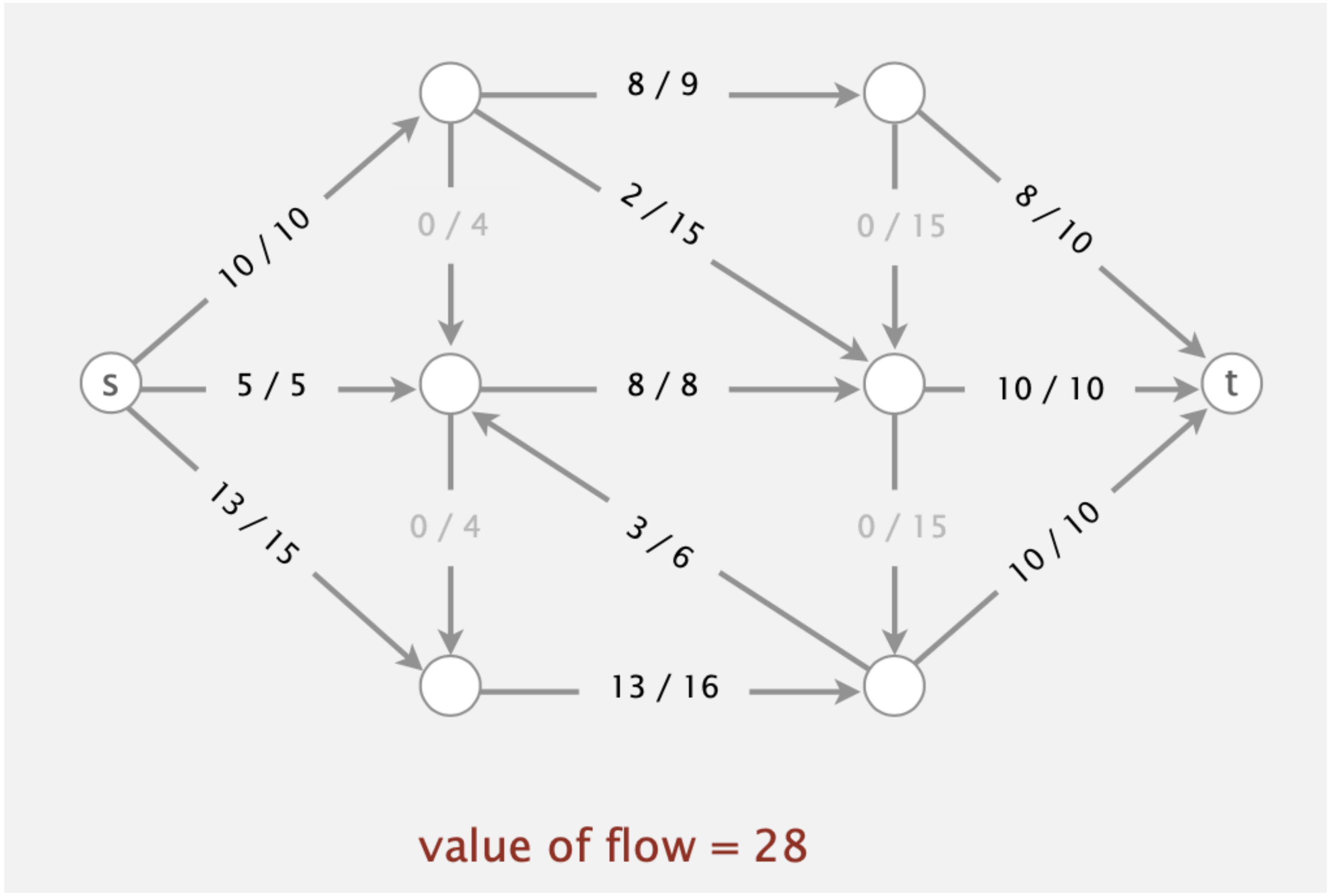
Goal: Find a feasible flow with the maximum possible amount (max flow).

Another Feasible Flow with $|f| = 11$

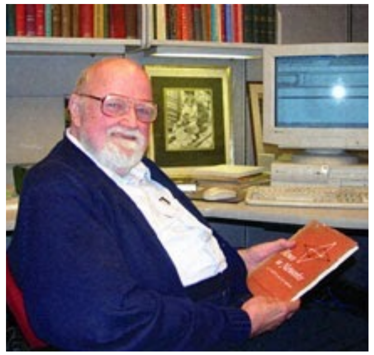
Is it a max flow?



Another Example



Ford-Fulkerson Algorithm


Lester R. Ford Jr.

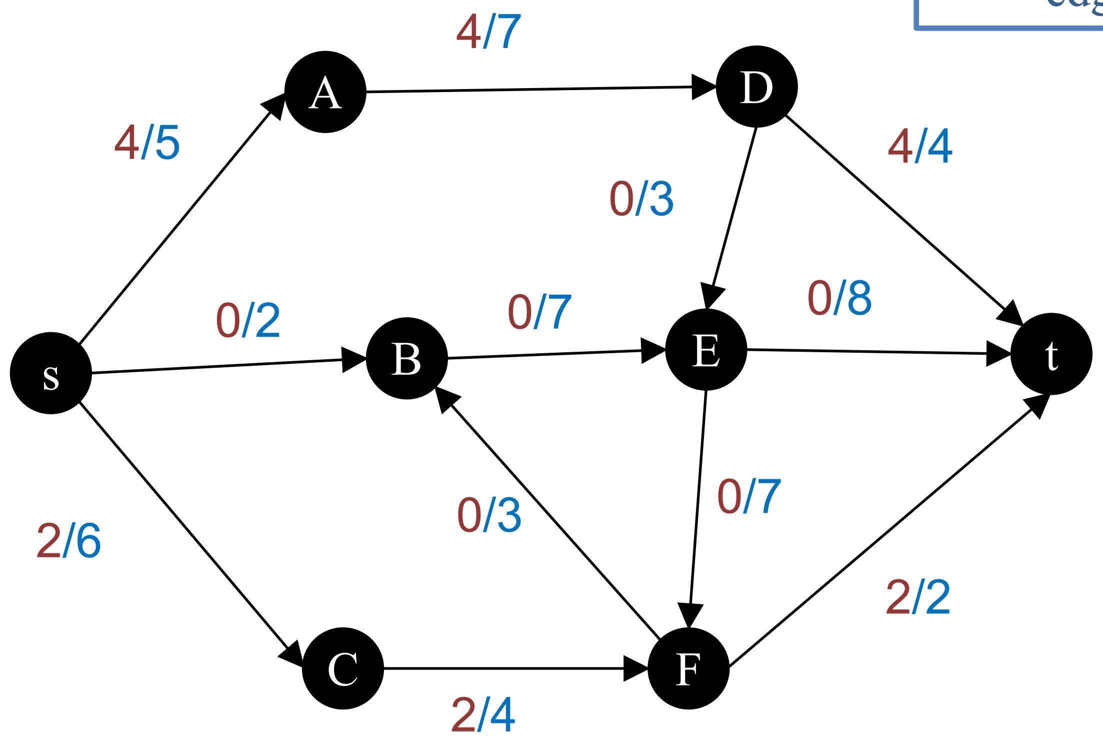
Delbert R. Fulkerson

A flow is feasible if:

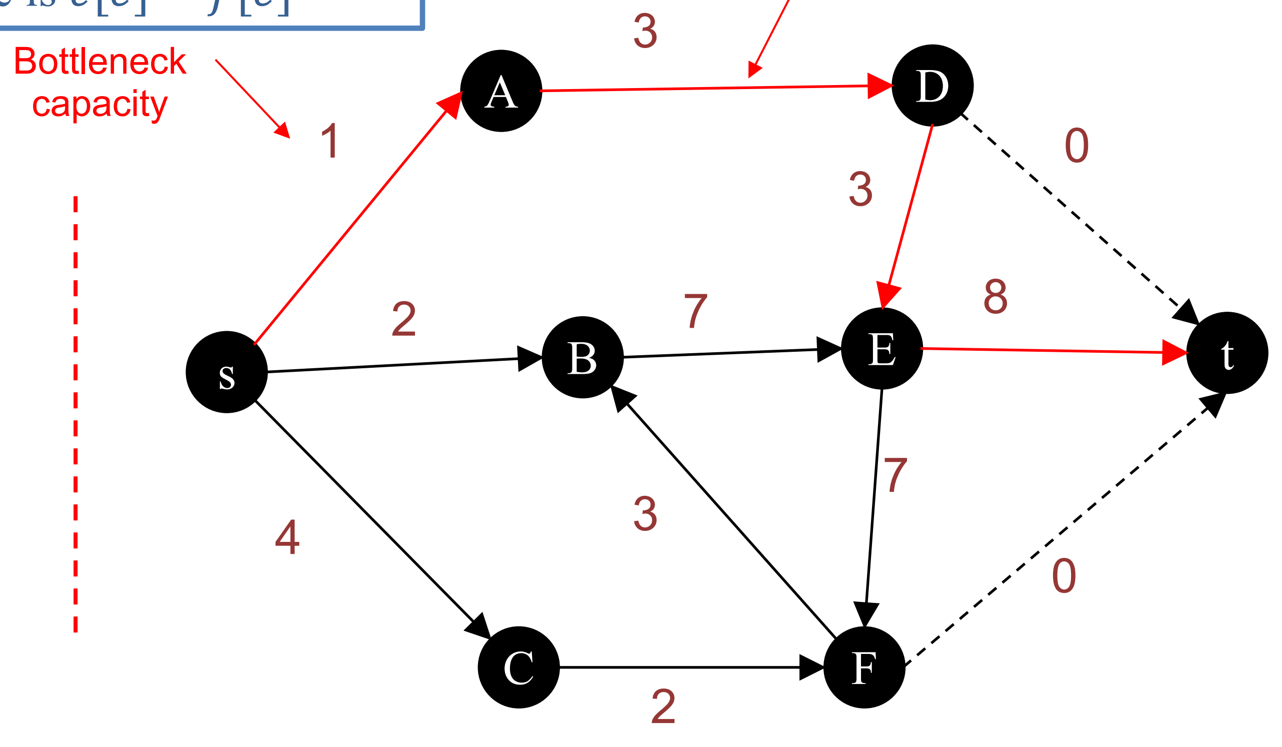
- ❑ For all edge $e = (u, v)$, we have $f[e] \leq c[e]$
- ❑ For all node $u \neq s, t$, we have
  - ➢ $\sum_{u \to e} f(e) = \sum_{u \leftarrow e} f(e)$
  - ➢ $\sum_{v \in V} f[u, v] = 0$

Remaining capacity of each edge is $c[e] - f[e]$

Augmenting path: A simple path from *s* to *t* in residual graph



Network Flow Graph with $|f| = 6$

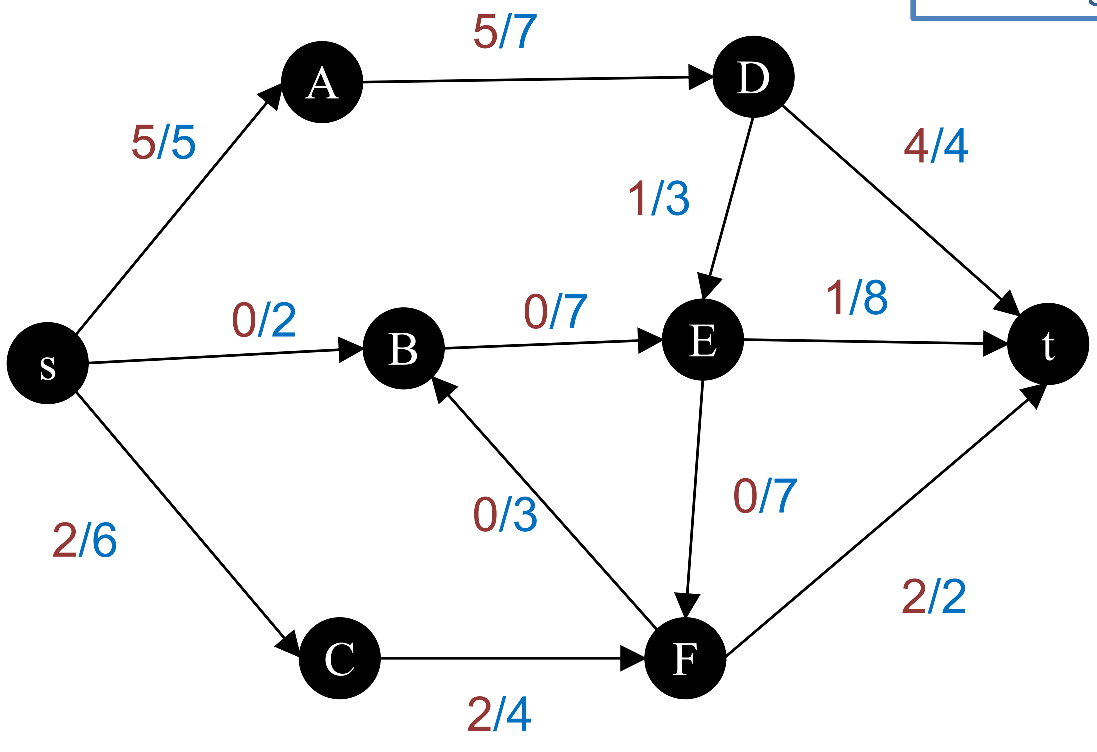


Residual Graph

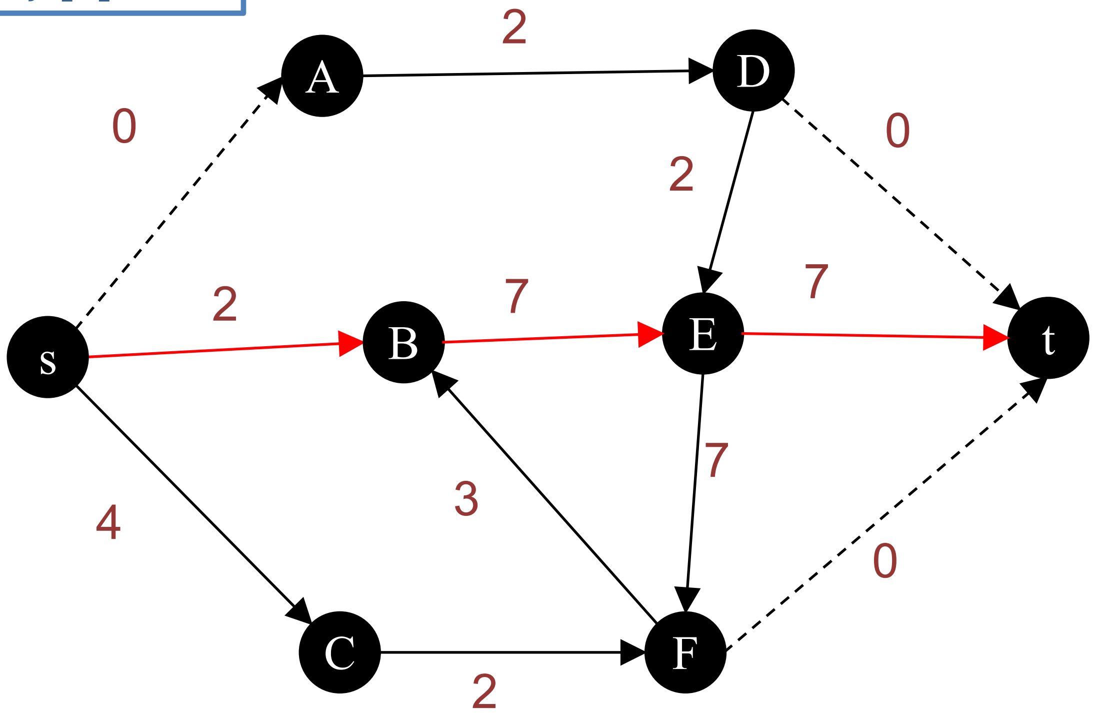
A flow is feasible if:

- For all edge $e = (u, v)$, we have $f[e] \leq c[e]$
- For all node $u \neq s, t$, we have
  - $\sum_{u \to e} f(e) = \sum_{u \leftarrow e} f(e)$
  - $\sum_{v \in V} f[u, v] = 0$

Remaining capacity of each edge is $c[e] - f[e]$



Network Flow Graph with $|f| = 7$

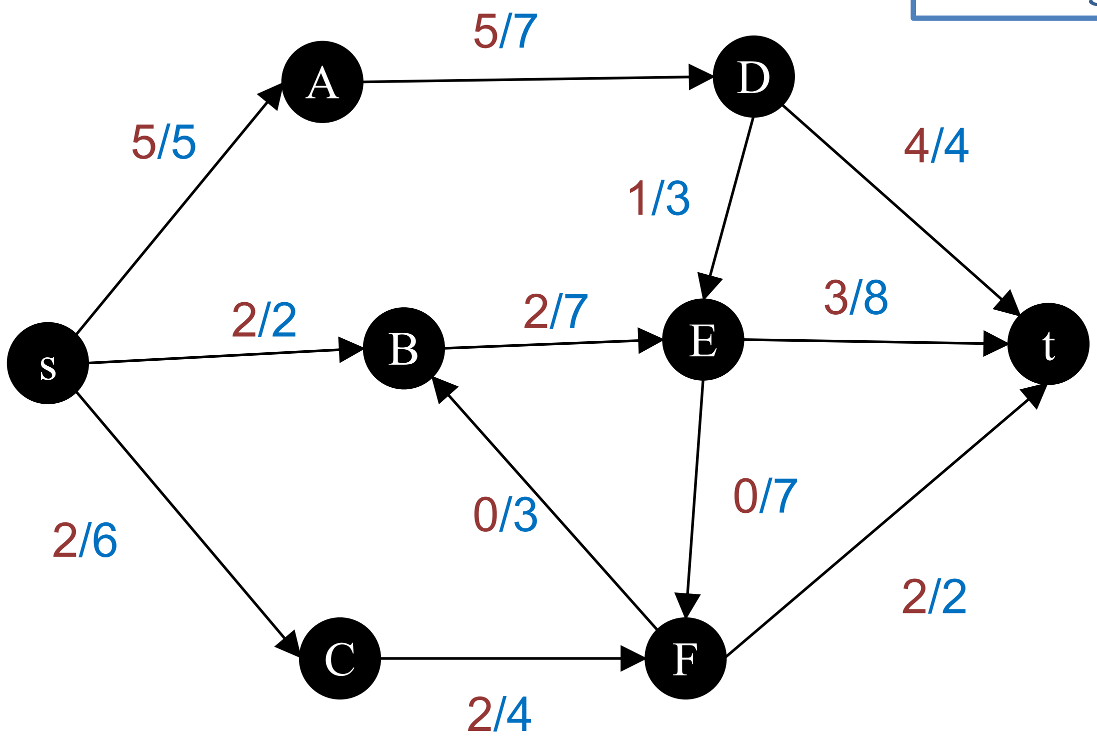


Residual Graph

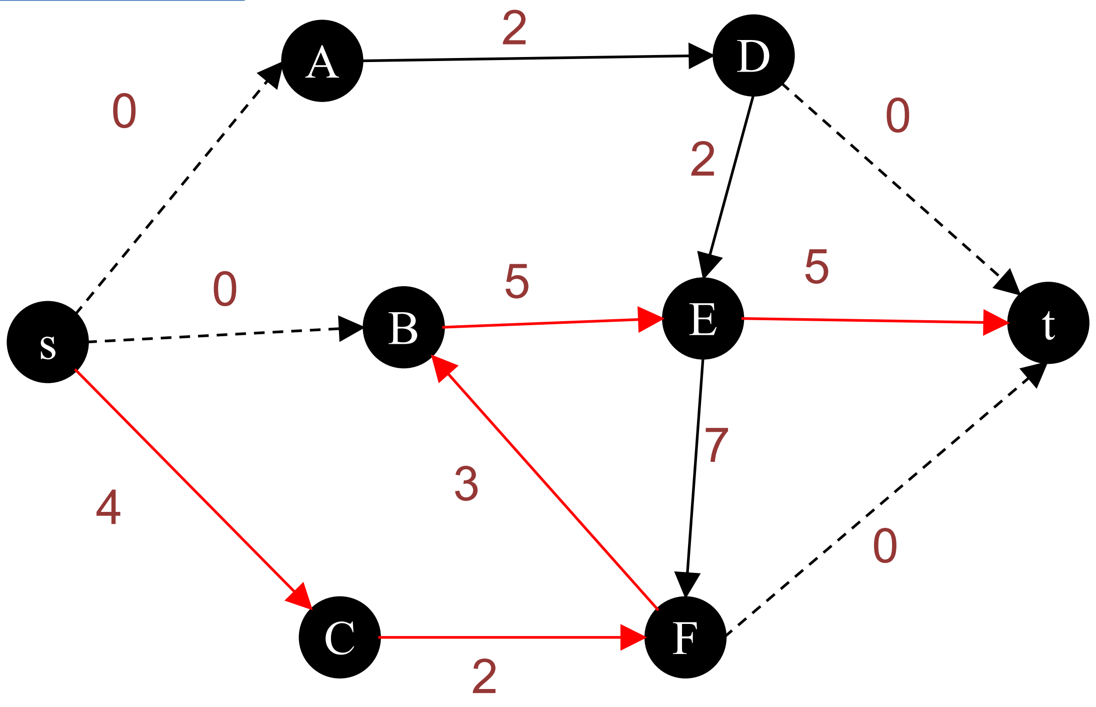
A flow is feasible if:

- For all edge $e = (u, v)$, we have $f[e] \leq c[e]$
- For all node $u \neq s, t$, we have
  - $\sum_{u \to e} f(e) = \sum_{u \leftarrow e} f(e)$
  - $\sum_{v \in V} f[u, v] = 0$

Remaining capacity of each edge is $c[e] - f[e]$



Network Flow Graph with $|f| = 9$

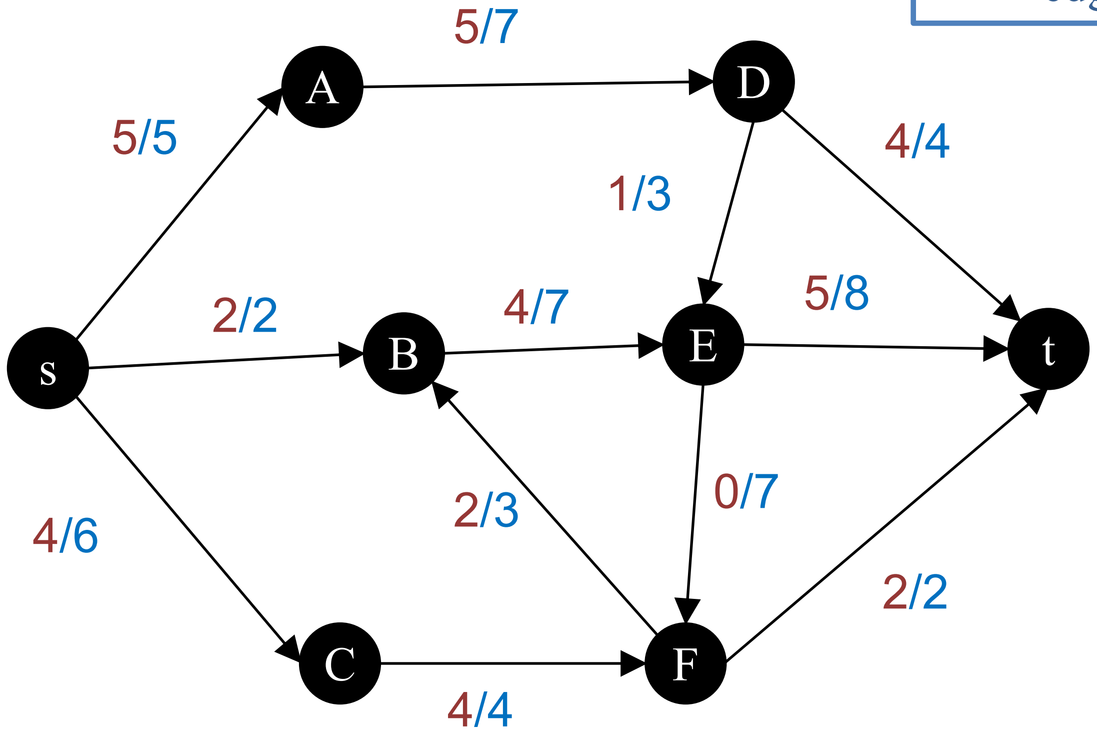


Residual Graph

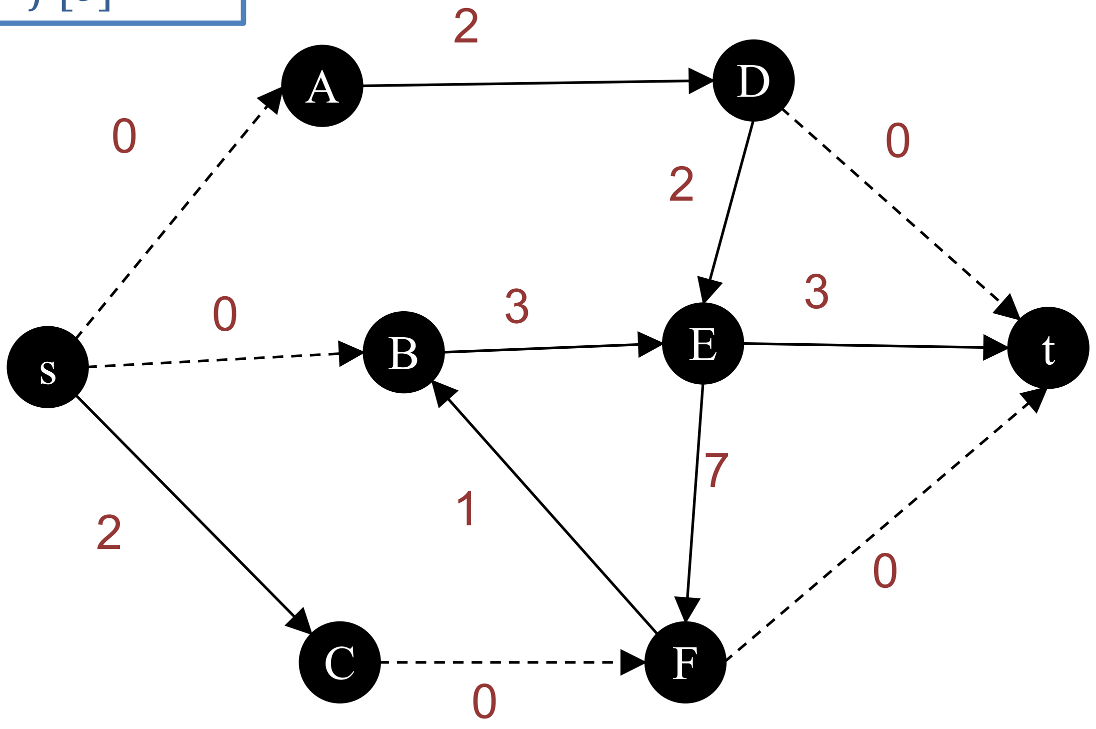
A flow is feasible if:

- For all edge $e = (u, v)$, we have $f[e] \leq c[e]$
- For all node $u \neq s, t$, we have
  - $\sum_{u \to e} f(e) = \sum_{u \leftarrow e} f(e)$
  - $\sum_{v \in V} f[u, v] = 0$

Remaining capacity of each edge is $c[e] - f[e]$



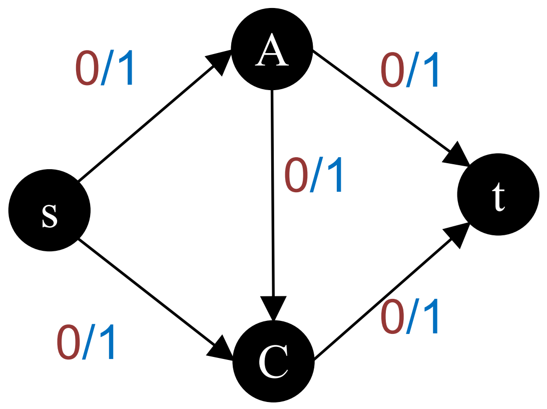
Network Flow Graph with $|f| = 11$

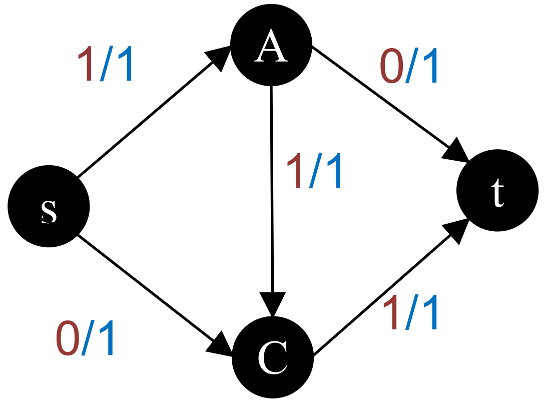


Residual Graph

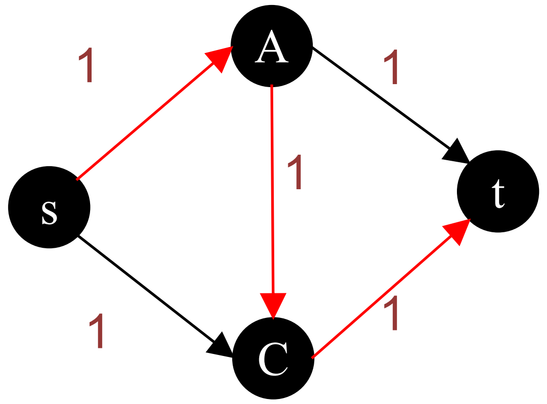
How to prove the correctness of the algorithm?

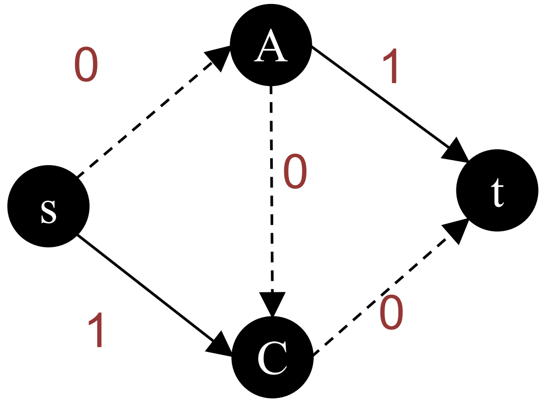
Counter Example





Network Flow Graphs

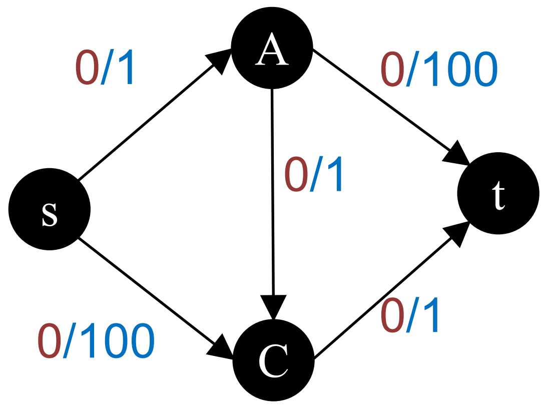


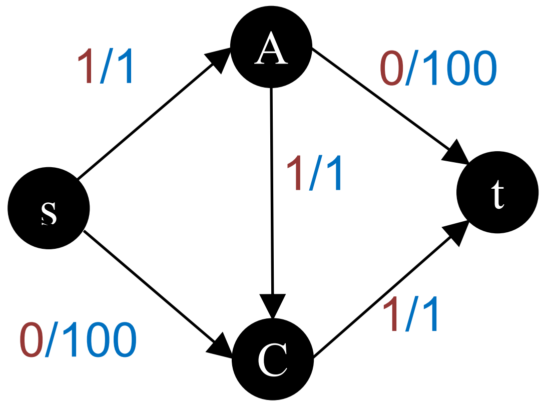


Residual Graphs

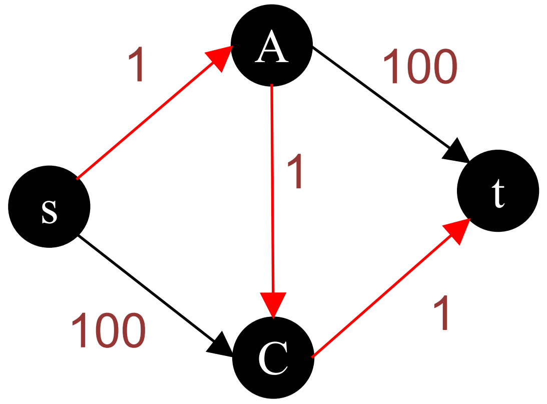
Another Counter Example

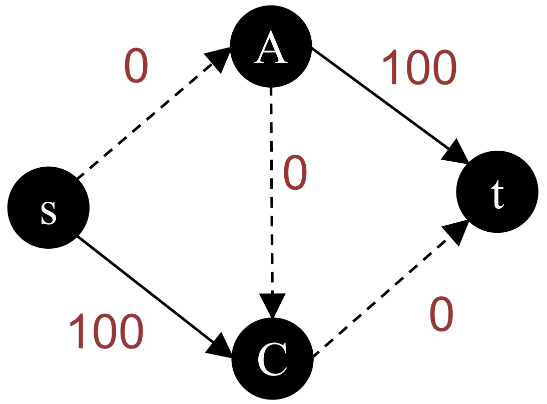
- What if we choose the shortest path from *s* to *t*?





Network Flow Graphs





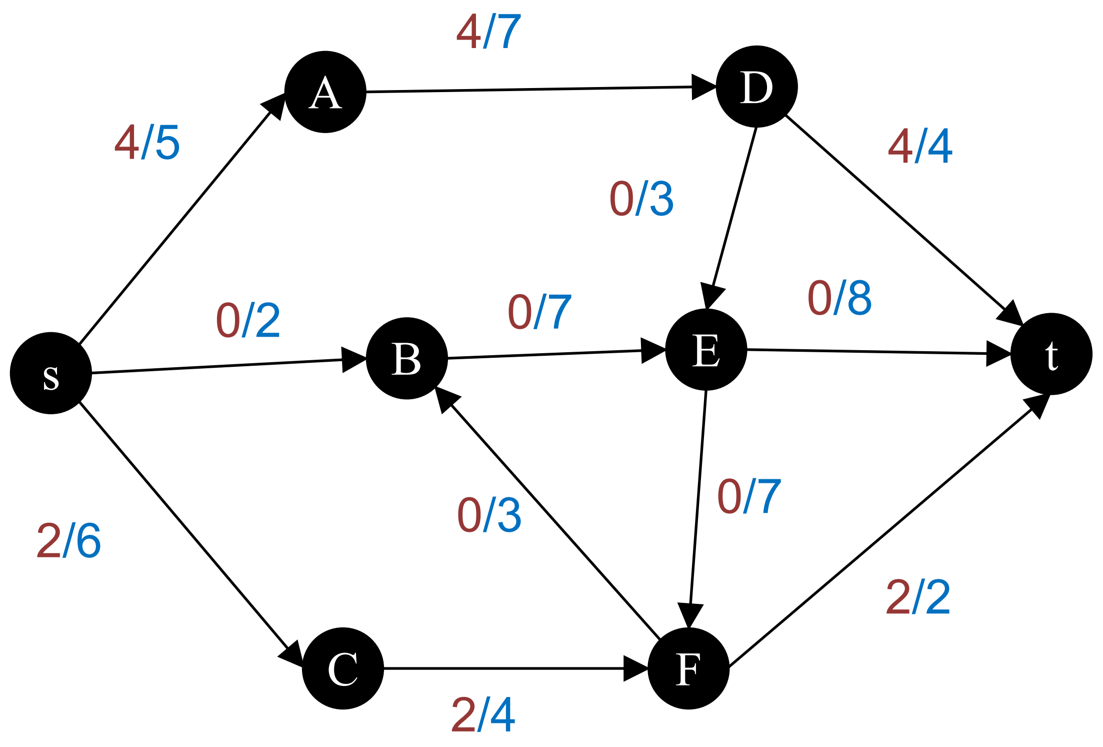
Residual Graphs

How can we resolve the issue?

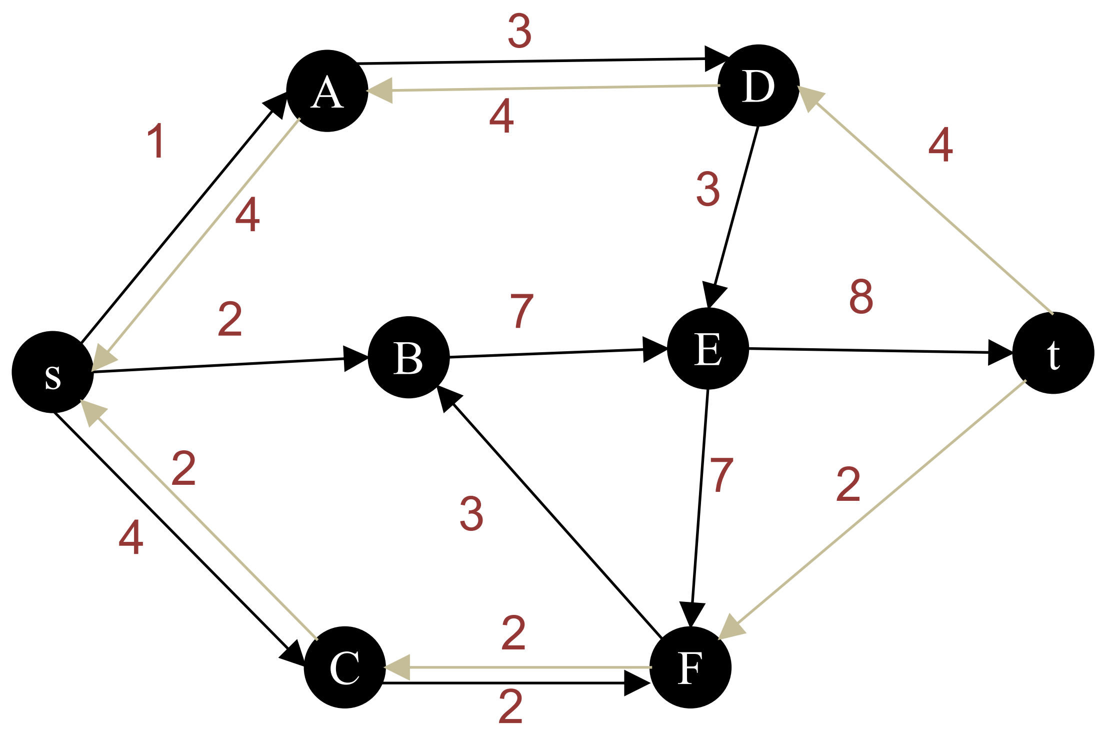
A flow is feasible if:

- ❑ For all edge $e = (u, v)$, we have $f[e] \leq c[e]$
- ❑ For all node $u \neq s, t$, we have
  - ➢ $\sum_{u \to e} f(e) = \sum_{u \leftarrow e} f(e)$
  - ➢ $\sum_{v \in V} f[u, v] = 0$

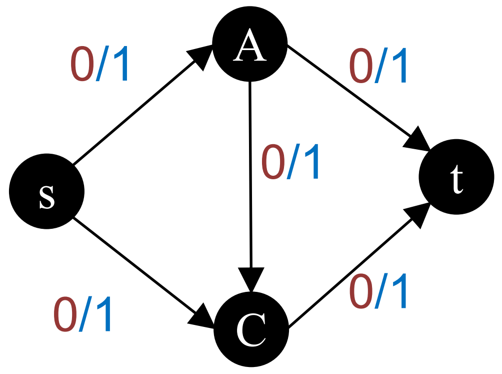
- **Forward edge**: Remaining capacity of each edge is $c[e] - f[e]$
- **Backward edge:** Flow of edge in the backward direction

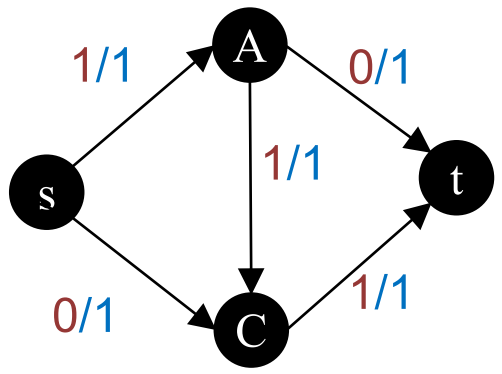


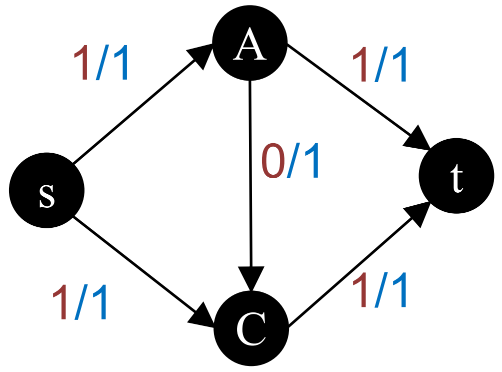
Network Flow Graph with $|f| = 6$

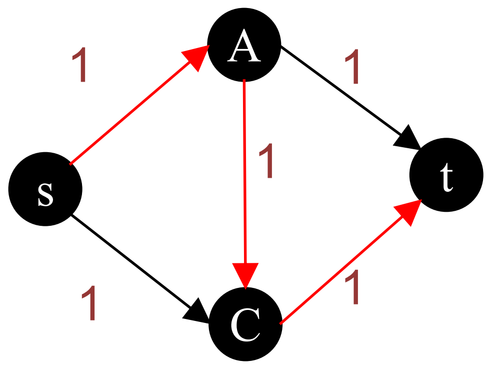


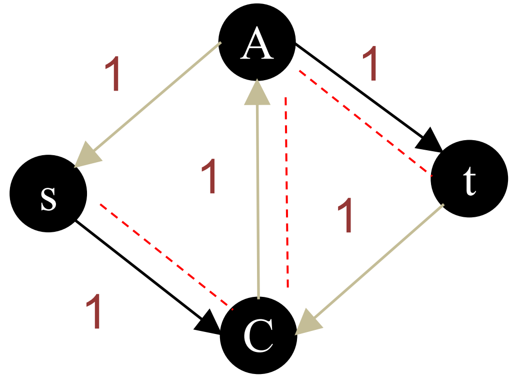
Residual Graph

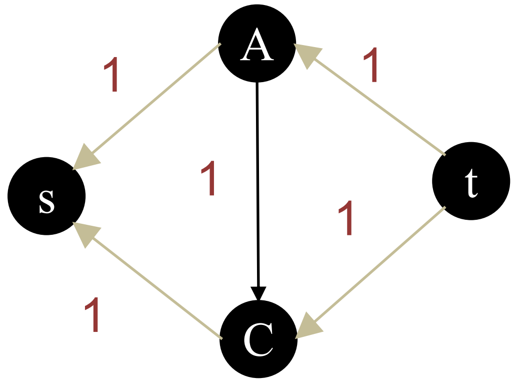

0/1
0/1
0/1
0/1
0/1
A
s
t
C

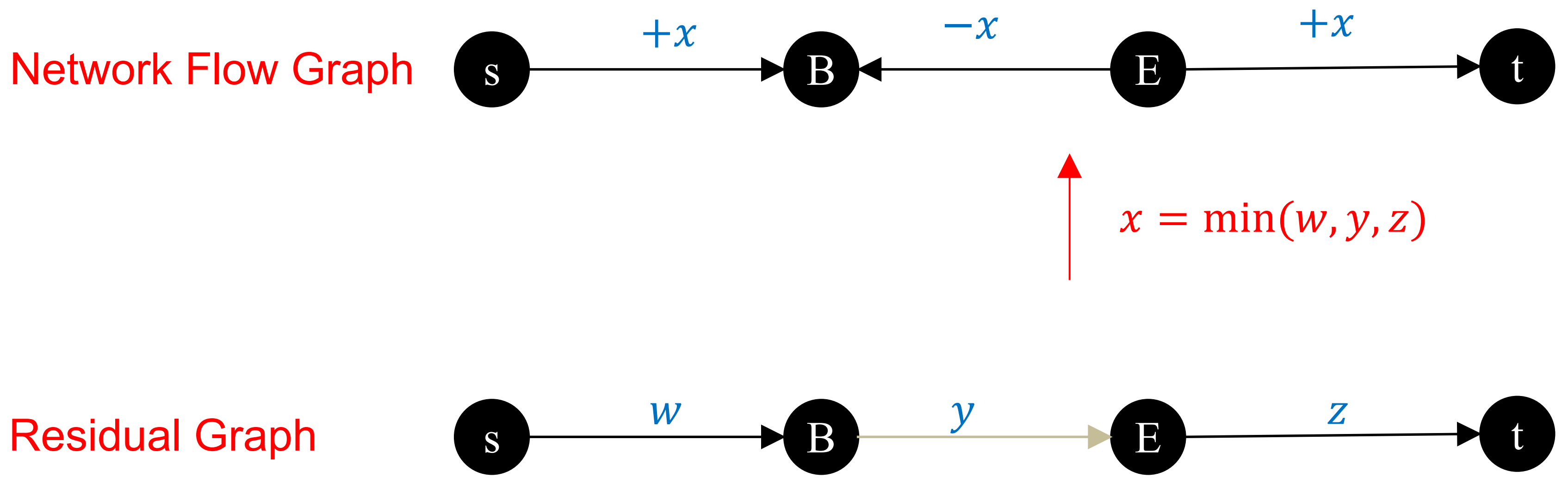

1/1
0/1
1/1
0/1
1/1
A
s
t
C


1/1
1/1
0/1
1/1
1/1
A
s
t
C


1
1
1
1
1
A
s
t
C


1
1
1
1
1
A
s
t
C


1
1
1
1
1
A
s
t
C


Network Flow Graph
s
$+x$
B
$-x$
E
$+x$
t
$x = \min(w, y, z)$
Residual Graph
s
w
B
y
E
z
t

Ford-Fulkerson Algorithm: Idea

```
Ford-Fulkerson(G, s, t) {
    Set all flows to be 0
    while there exists an augmenting path {
        Find an augmenting path
        Compute bottleneck capacity
        Increase flow on that path by bottleneck capacity
    }
}
```

Ford-Fulkerson Algorithm: Details

```
Ford-Fulkerson(G, s, t) {
    for each edge e in G.E {
        f[e] = 0
    }

    while true {
        for each node u in G.V {
            flow[u] = null
        }

        DFS(s, ∞)

        if flow[t] = null {
            return f // max flow is found
        }

        v = t
        x = flow[t]
        while v ≠ s {
            u = parent[v]
            f[u, v] += x
            f[v, u] -= x
            v = u
        }
    }
}
```

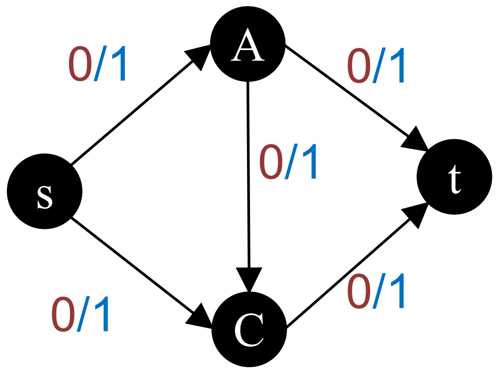
Ford-Fulkerson Algorithm: Details

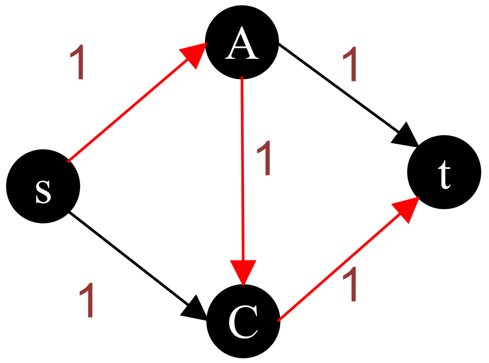
```
Ford-Fulkerson(G, s, t) {
    for each edge e in G.E {
        f[e] = 0
    }

    while true {
        for each node u in G.V {
            flow[u] = null
        }

        DFS(s, ∞)

        if flow[t] = null {
            return f // max flow is found
        }

        v = t
        x = flow[t]
        while v ≠ s {
            u = parent[v]
            f[u, v] += x
            f[v, u] -= x
            v = u
        }
    }
}
```

Updating the current flow

```
DFS(u, x) {
    flow[u] = x
    for each e = (u, v) as outgoing edges of u {
        if flow[v] = null and c[u, v] - f[u, v] > 0 {
            parent[v] = u
            DFS(v, min(x, c[u, v] - f[u, v]))
        }
    }
    for each e = (v, u) as incoming edges of u {
        if flow[v] = null and f[v, u] > 0 {
            parent[v] = u
            DFS(v, min(x, f[v, u]))
        }
    }
}
```

Example 1 (1)

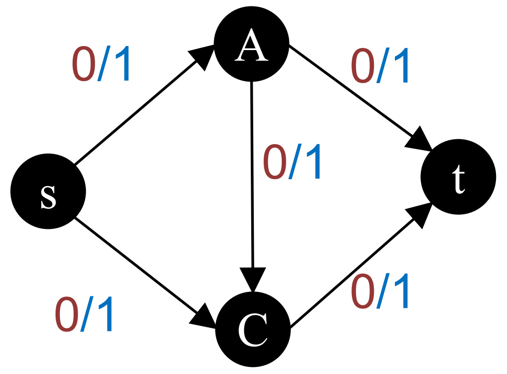


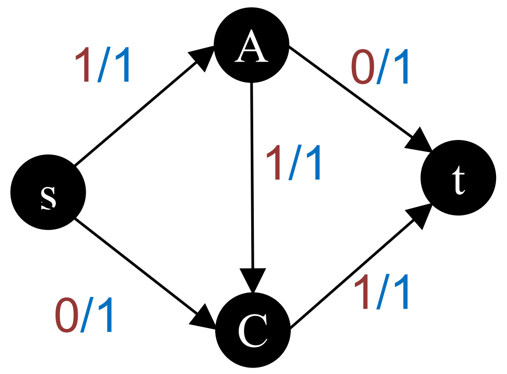


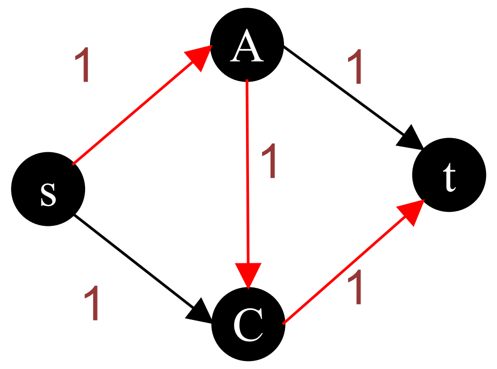
f	(s, A)	(A, s)	(s, C)	(C, s)	(A, C)	(C, A)	(C, t)	(t, C)	(A, t)	(t, A)
	0	0	0	0	0	0	0	0	0	0

	s	A	C	t
flow	∞	1	1	1
parent	-	s	A	C

Example 1 (2)





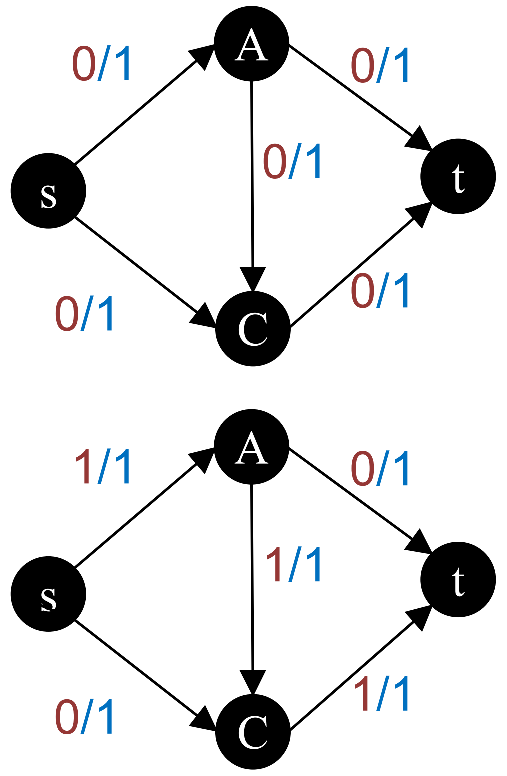


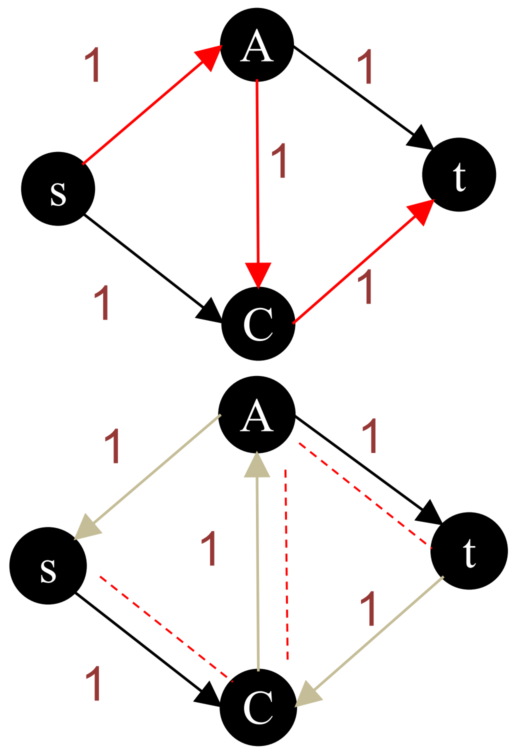
f	(s, A)	(A, s)	(s, C)	(C, s)	(A, C)	(C, A)	(C, t)	(t, C)	(A, t)	(t, A)
	0	0	0	0	0	0	0	0	0	0

	s	A	C	t
flow	∞	1	1	1
parent	-	s	A	C

f	(s, A)	(A, s)	(s, C)	(C, s)	(A, C)	(C, A)	(C, t)	(t, C)	(A, t)	(t, A)
	1	-1	0	0	1	-1	1	-1	0	0

Example 1 (3)

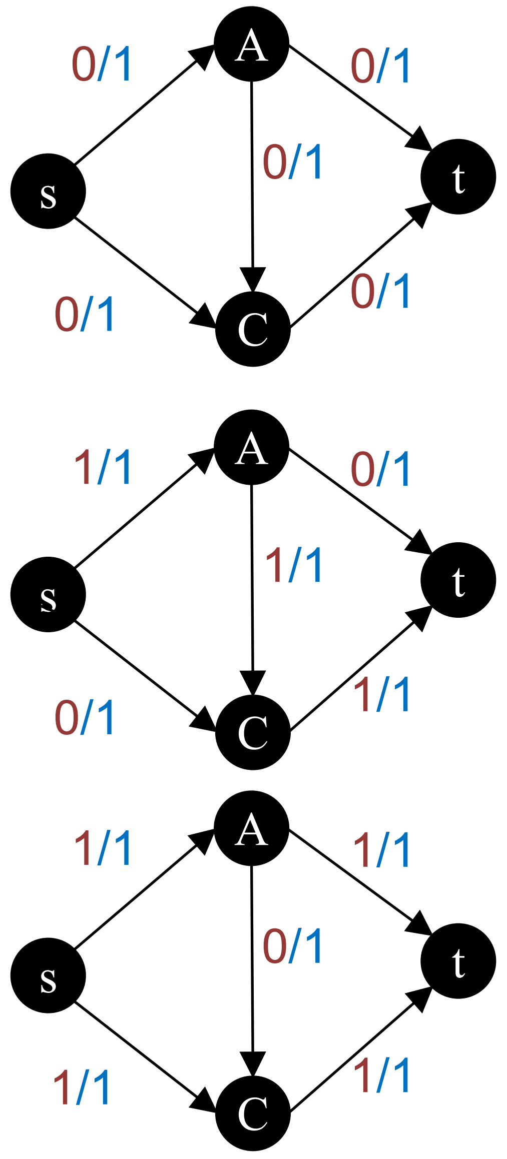


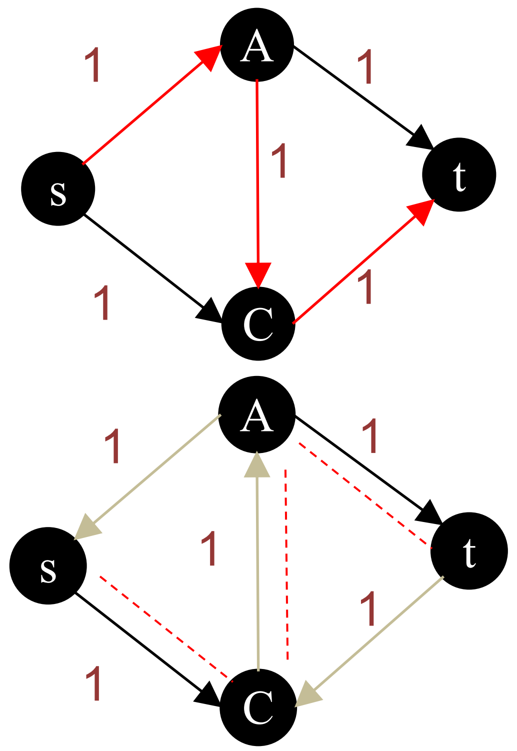


f	(s, A)	(A, s)	(s, C)	(C, s)	(A, C)	(C, A)	(C, t)	(t, C)	(A, t)	(t, A)
	1	-1	0	0	1	-1	1	-1	0	0

	s	A	C	t
flow	∞	1	1	1
parent	-	C	s	A

Example 1 (4)



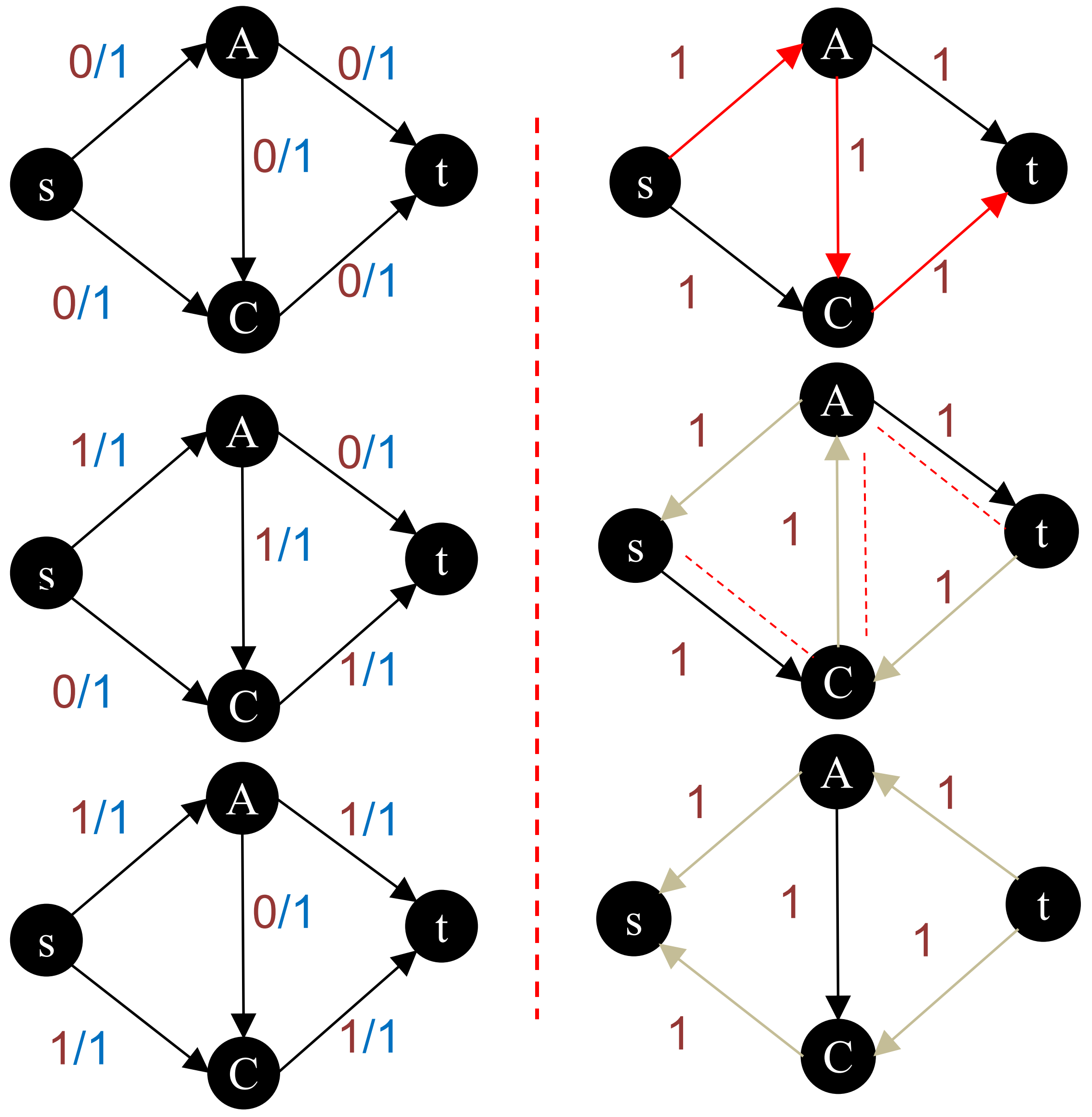


f	(s, A)	(A, s)	(s, C)	(C, s)	(A, C)	(C, A)	(C, t)	(t, C)	(A, t)	(t, A)
	1	-1	0	0	1	-1	1	-1	0	0

	s	A	C	t
flow	∞	1	1	1
parent	-	C	s	A

f	(s, A)	(A, s)	(s, C)	(C, s)	(A, C)	(C, A)	(C, t)	(t, C)	(A, t)	(t, A)
	1	-1	1	-1	0	0	1	-1	1	-1

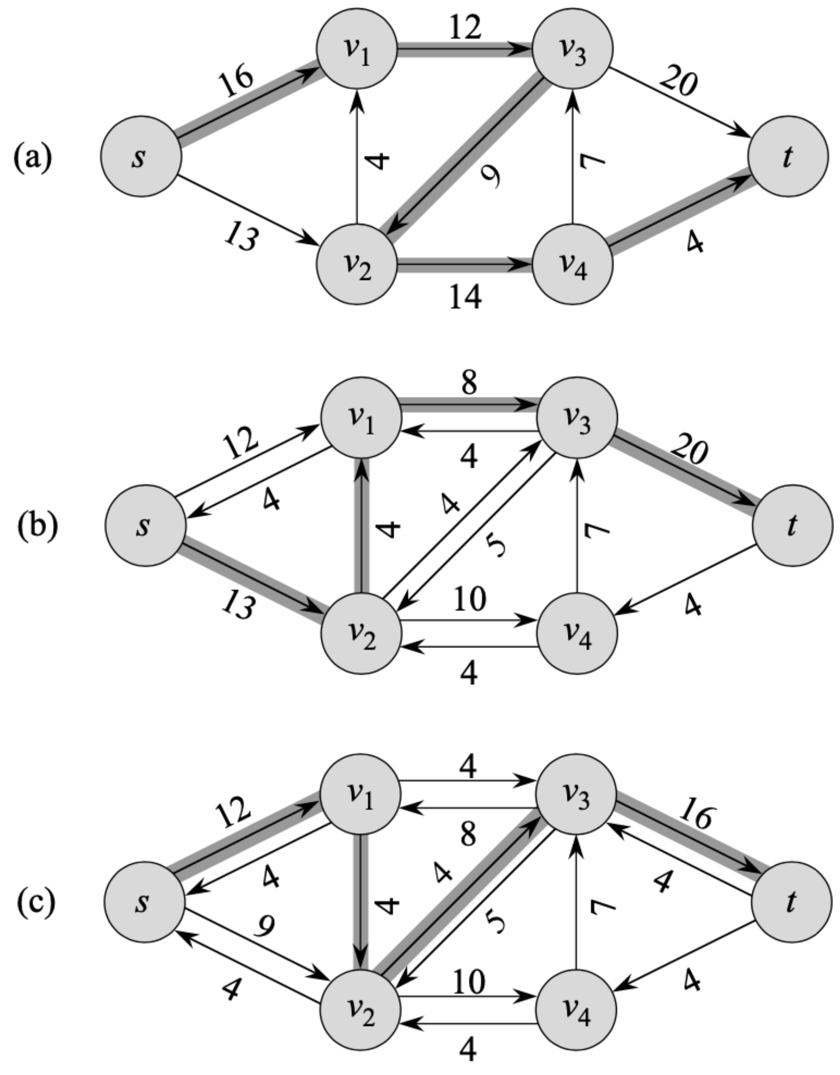
Example 1 (5)

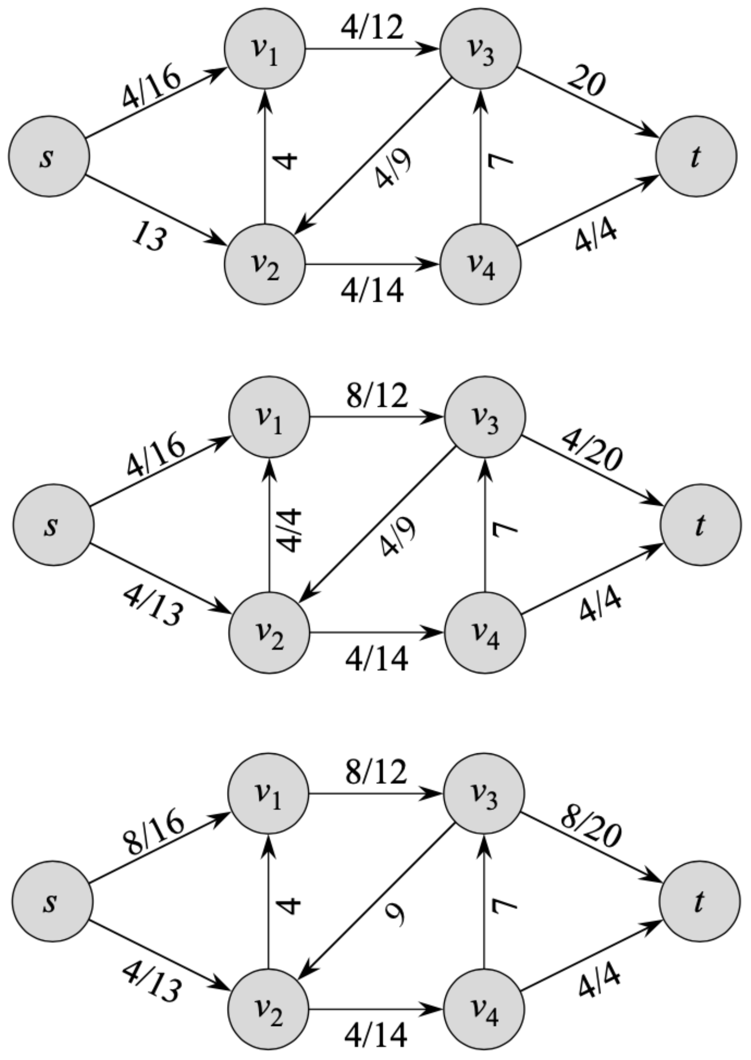


f	(s, A)	(A, s)	(s, C)	(C, s)	(A, C)	(C, A)	(C, t)	(t, C)	(A, t)	(t, A)
	1	-1	1	-1	0	0	1	-1	1	-1

	s	A	C	t
flow	∞	null	null	null
parent	-	-	-	-

Example 2 (1)

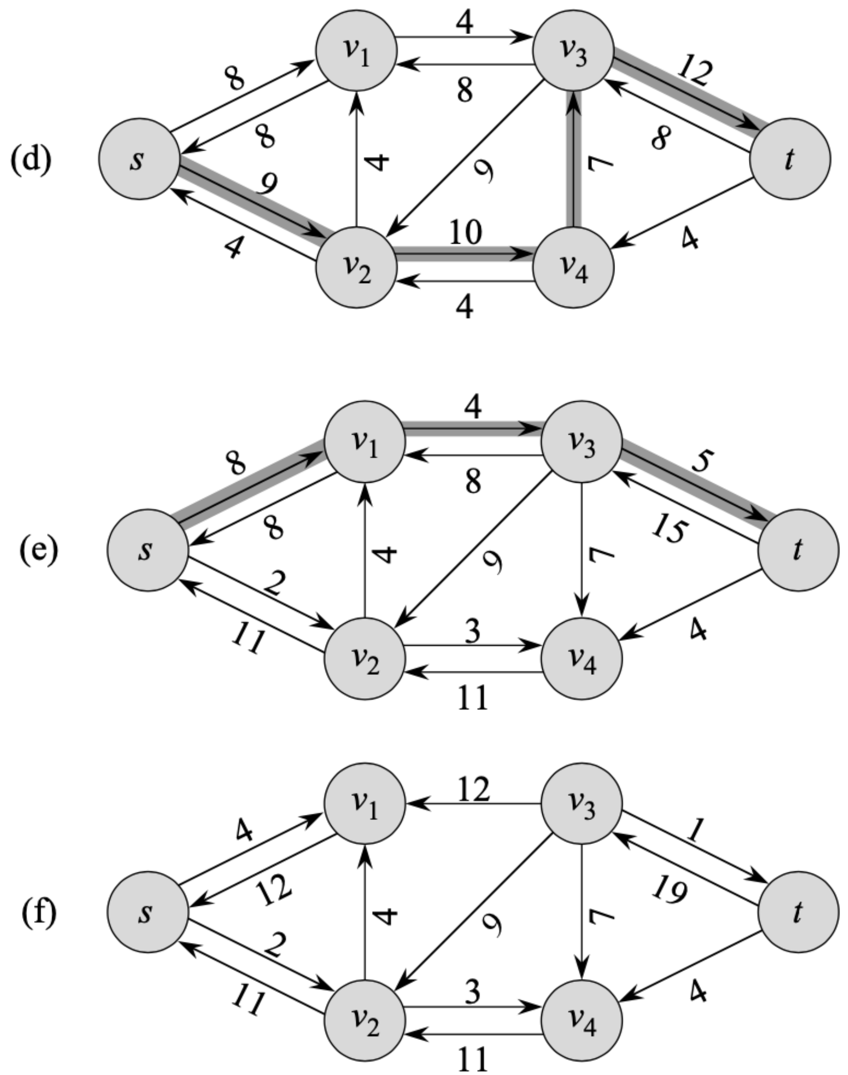




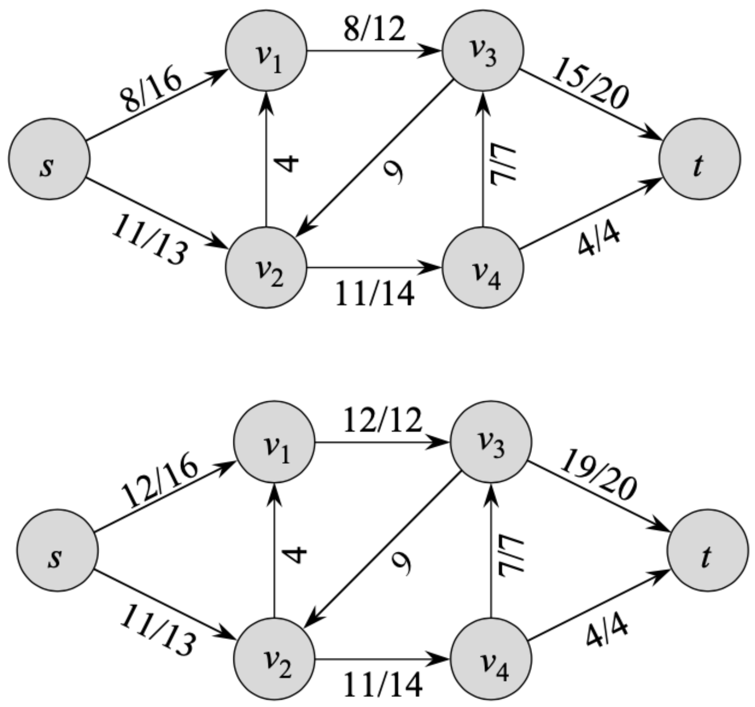
Residual Graphs

Network Flow Graphs

Example 2 (2)



Residual Graphs



Network Flow Graphs

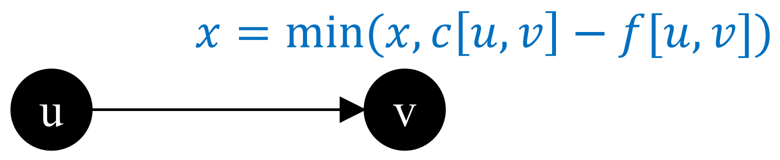
Proof of Correctness

The updated flow is feasible (1)

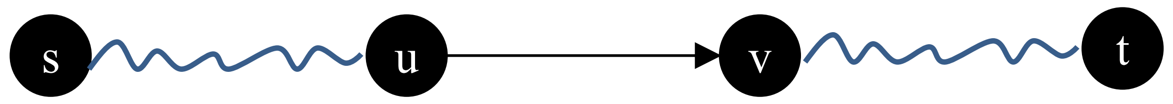
A flow is feasible if:

- ❑ For all edge $e = (u, v)$, we have $f[e] \le c[e]$
- ❑ For all pairs $u, v \in V$, we have $f[u, v] = -f[v, u]$
- ❑ For all node $u \ne s, t$, we have
  - ➢ $\sum_{u \to e} f(e) = \sum_{u \leftarrow e} f(e)$
  - ➢ $\sum_{v \in V} f[u, v] = 0$

```
DFS(u, x) {
    flow[u] = x
    for each e = (u, v) as outgoing edges of v {
        if flow[v] = null and c[u, v] - f[u, v] > 0 {
            parent[v] = u
            DFS(v, min(x, c[u, v] - f[u, v]))
        }
    }
    for each e = (v, u) as incoming edges of v {
        if flow[v] = null and f[v, u] > 0 {
            parent[v] = u
            DFS(v, min(x, f[v, u]))
        }
    }
}
```



Network Flow Graph



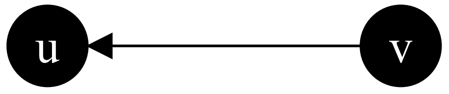
Residual Graph (forward)

The updated flow is feasible (2)

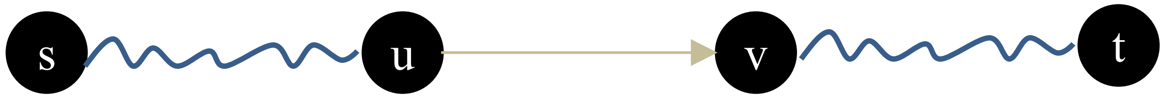
A flow is feasible if:

- ❑ For all edge $e = (u, v)$, we have $f[e] \le c[e]$
- ❑ For all pairs $u, v \in V$, we have $f[u, v] = -f[v, u]$
- ❑ For all node $u \neq s, t$, we have
  - ➢ $\sum_{u \to e} f(e) = \sum_{u \leftarrow e} f(e)$
  - ➢ $\sum_{v \in V} f[u, v] = 0$

$x = \min(x, f[v, u])$



Network Flow Graph



Residual Graph (backward)

```
DFS(u, x) {
    flow[u] = x
    for each e = (u, v) as outgoing edges of v {
        if flow[v] = null and c[u, v] - f[u, v] > 0 {
            parent[v] = u
            DFS(v, min(x, c[u, v] - f[u, v]))
        }
    }
    for each e = (v, u) as incoming edges of v {
        if flow[v] = null and f[v, u] > 0 {
            parent[v] = u
            DFS(v, min(x, f[v, u]))
        }
    }
}
```

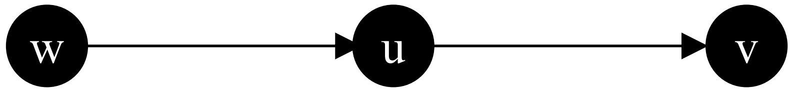
The updated flow is feasible (3)

A flow is feasible if:

- ✓ For all edge $e = (u, v)$, we have $f[e] \leq c[e]$
- ❑ For all pairs $u, v \in V$, we have $f[u, v] = -f[v, u]$
- ❑ For all node $u \neq s, t$, we have
  - ➢ $\sum_{u \to e} f(e) = \sum_{u \leftarrow e} f(e)$
  - ➢ $\sum_{v \in V} f[u, v] = 0$

trivial

Why?



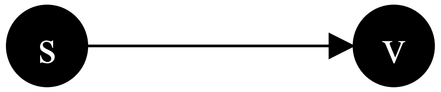
Residual Graph

```
Ford-Fulkerson(G, s, t) {
    for each edge e in G.E {
        f[e] = 0
    }

    while true {
        for each node u in G.V {
            flow[u] = null
        }

        DFS(s, ∞)

        if flow[t] = null {
            break // max flow is found
        }

        v = t
        x = flow[t]
        while v ≠ s {
            u = parent[v]
            f[u, v] += x
            f[v, u] -= x
            v = u
        }
    }
}
```

Flow has been increased

Amount of flow is defined as:

- ❑ $|f| = \sum_{s\to e} f(e) - \sum_{s\leftarrow e} f(e) = \sum_{t\leftarrow e} f(e) - \sum_{t\to e} f(e)$
- ❑ $|f| = \sum_{v\in G} f[s,v] = \sum_{v\in G} f[v,t]$



Residual Graph

```
Ford-Fulkerson(G, s, t) {
    for each edge e in G.E {
        f[e] = 0
    }

    while true {
        for each node u in G.V {
            flow[u] = null
        }

        DFS(s, ∞)

        if flow[t] = null {
            break // max flow is found
        }

        v = t
        x = flow[t]
        while v ≠ s {
            u = parent[v]
            f[u, v] += x
            f[v, u] -= x
            v = u
        }
    }
}
```

Next Step?

Cuts of Flow Networks

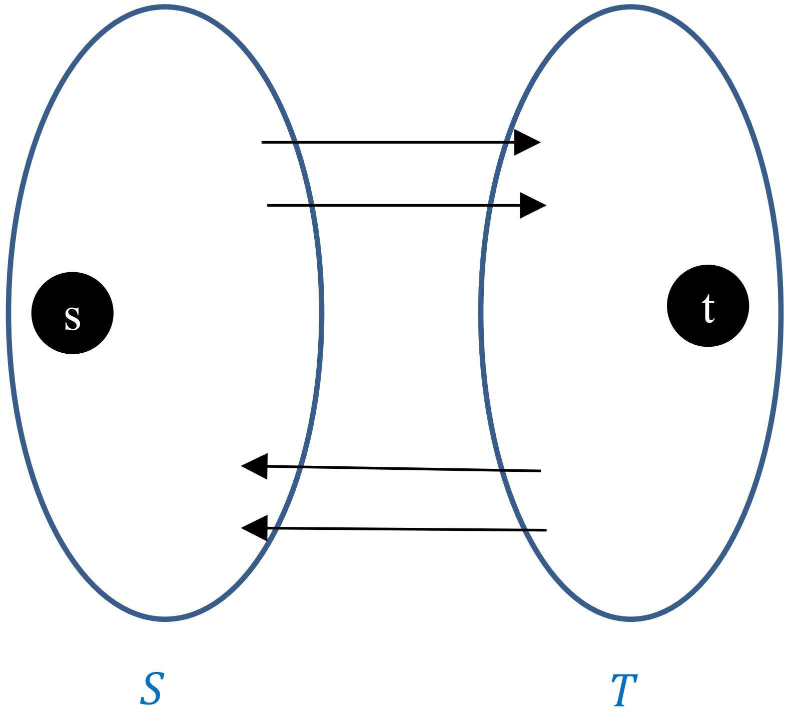
- Definition of cut (S, T): A partition of vertices V into S and T=V–S such that $s \in S$ and $t \in T$.

- Definition of capacity of cut (S,T): Sum of the capacities of edges from S to T

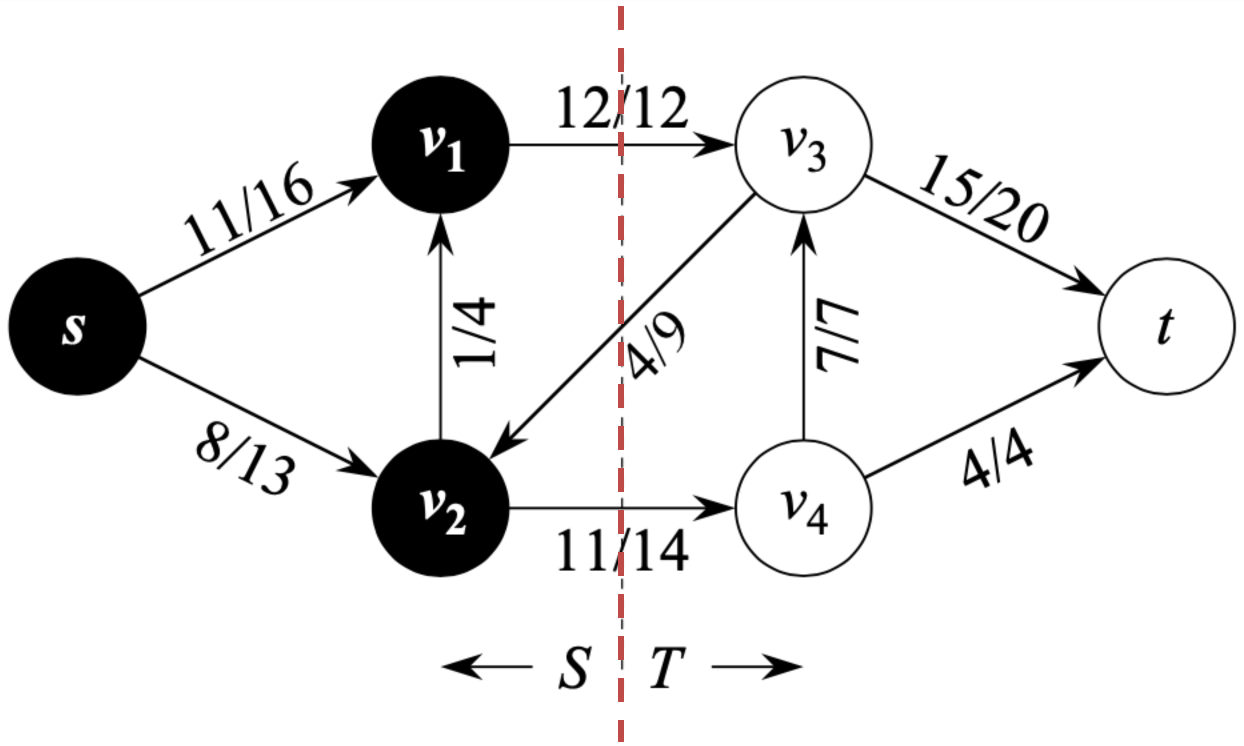
$$c(S,T) = \sum_{u \in S, v \in T} c[u,v]$$

- Definition of net flow $f(S,T)$: Net flow out of S.

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



Example



- Net flow: $f(S,T) = f(v_1, v_3) + f(v_2, v_4) - f(v_3, v_2) = 12 + 11 - 4 = 19$
- Cut capacity: $c(S,T) = c(v_1, v_3) + c(v_2, v_4) = 12 + 14 = 26$

Max-Flow / Min-Cut Theorem (1)

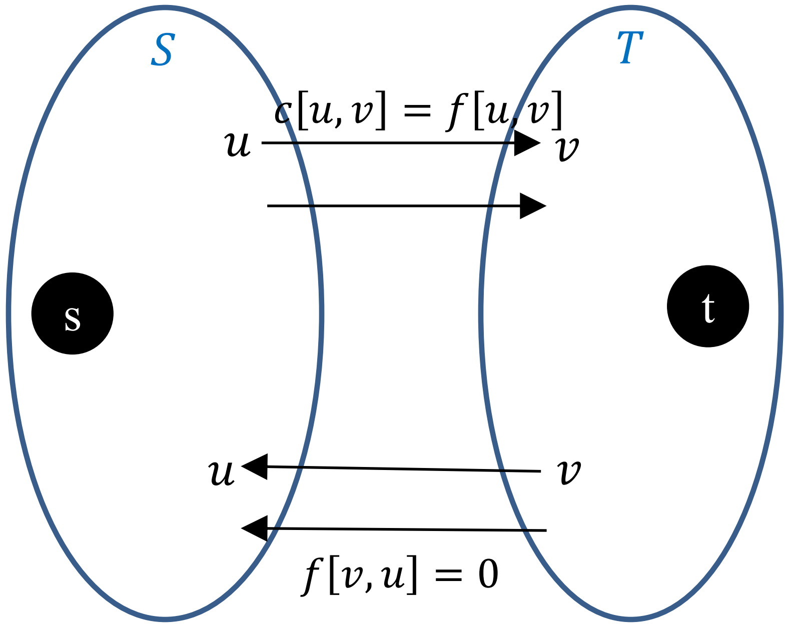
- Theorem: In a flow network with a feasible flow f, the following statements are equivalent:
  1. *i.* f is maximized.
  2. ii. Residual graph has no augmenting paths.
  3. iii. There exists a cut (S,T) such that $c(S,T) = f(S,T)$.
- Ford-Fulkerson Proof: (ii) ⇒ (i)
- Detailed Proof:
  - (i) ⇒ (ii): f is maximized ⇒ Residual graph has no augmenting paths
    - We show by contradiction.
    - If there exists an augmenting path, then we can improve f by sending flow along path.
    - So f is not a maximum flow which is a contradiction.

Max-Flow / Min-Cut Theorem (2)

- (ii) ⇒ (iii): No augmenting paths ⇒ A cut (S, T) exists such that $c(S,T) = f(S,T)$.
  - Create partition S such that all nodes within it are reachable from node s in the residual graph. Other nodes are placed in T.
  - All edges from S to T should be saturated, because otherwise there would be an augmenting path from S to T.
  - All edges from T to S should be empty, because otherwise the residual would have a backward edge and hence there would be an augmenting path from S to T.

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

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



Max-Flow / Min-Cut Theorem (3)

- (iii) ⇒ (i): A cut (S,T) exists such that $c(S,T) = f(S,T) \Rightarrow f$ is maximized
  - For a feasible flow f and a cut (S,T): $|f| = f(S,T)$
    - Proof: see CLRS 26.2.
  - For a feasible flow f and a cut (S,T):$|f| = f(S,T) \leq c(S,T)$
    - The amount of any feasible flow is less than, or equal to the size of any cut.
    - Proof based on definition.
  - Suppose f' is the maximized flow and there exists a cut such that $c(S,T) = f(S,T)$.
    1) $|f| \leq |f'|$
    2) $|f| = f(S,T) = c(S,T)$
    3) $|f'| \leq c(S,T)$

    (1), (2) , (3) $\Rightarrow |f| = |f'|$; in other words, f is also a maximum flow.

Runtime Complexity

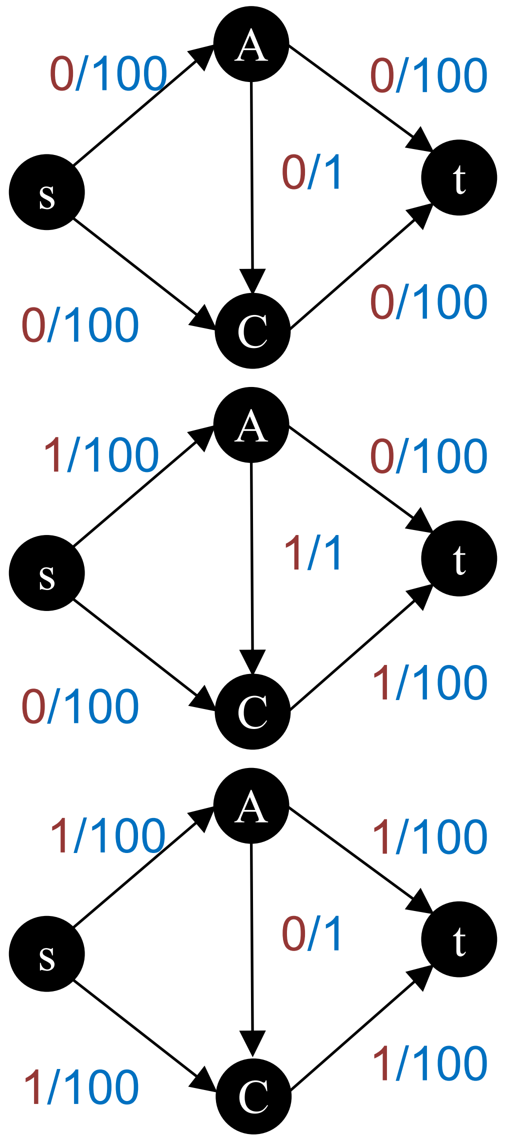
Runtime Complexity

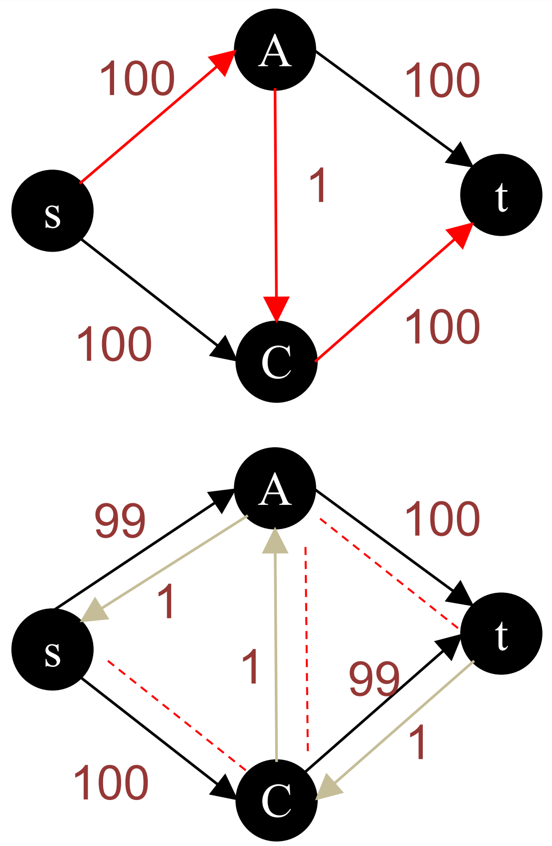
```
Ford-Fulkerson(G, s, t) {
    for each edge e in G.E {
        f[e] = 0
    }

    while true {
        for each node u in G.V {
            flow[u] = null
        }

        DFS(s, ∞)

        if flow[t] = null {
            break // max flow is found
        }

        v = t
        x = flow[t]
        while v ≠ s {
            u = parent[v]
            f[u, v] += x
            v = u
        }
    }
}
```

```
DFS(u, x) {
    flow[u] = x
    for each e = (u, v) as outgoing edges of v {
        if flow[v] = null and c[u, v] - f[u, v] > 0 {
            parent[v] = u
            DFS(v, min(x, c[u, v] - f[u, v]))
        }
    }
    for each e = (v, u) as incoming edges of v {
        if flow[v] = null and f[v, u] > 0 {
            parent[v] = u
            DFS(v, min(x, f[v, u]))
        }
    }
}
```

Runtime Complexity: $O(|f| \times (|V| + |E|))$

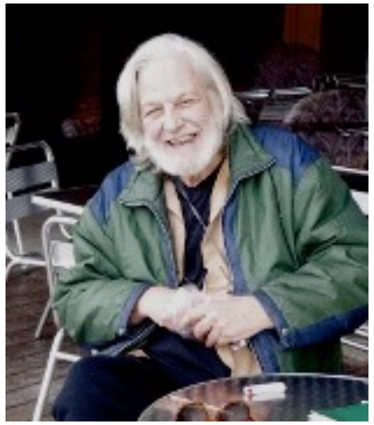
Runtime = $O(|f| \times (|V| + |E|))$





How to make it polynomial?

- Edmonds-Karp algorithm is similar to Ford-Fulkerson method except it finds the shortest path from s to t in the residual graph where each edge has unit distance (weight).
  - To do that, it employs BFS.
- It can be proven that Edmonds-Karp algorithm runs in $O(|V||E|^2)$.
  - Proof: See CLRS 26.2.


Jack R. Edmonds

Richard M. Karp

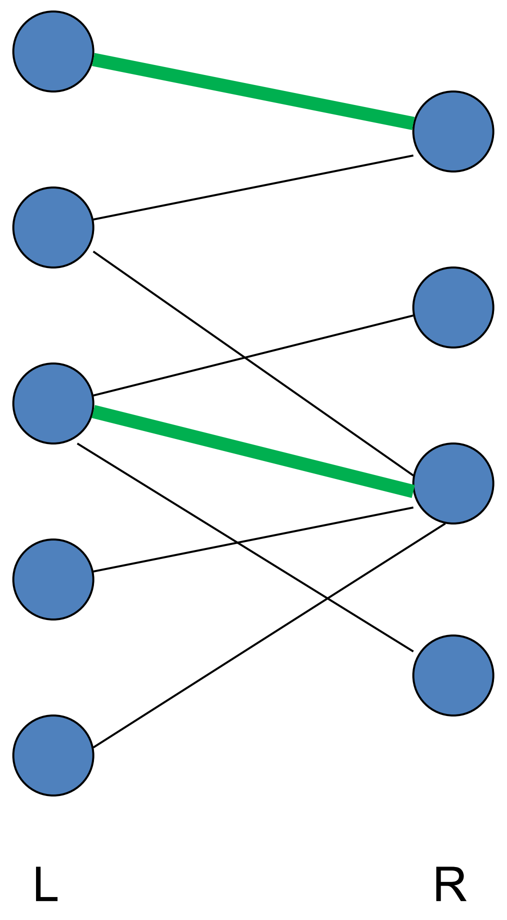
Extensions to the Maximum-Flow Problem

Maximum Bipartite Matching Problem

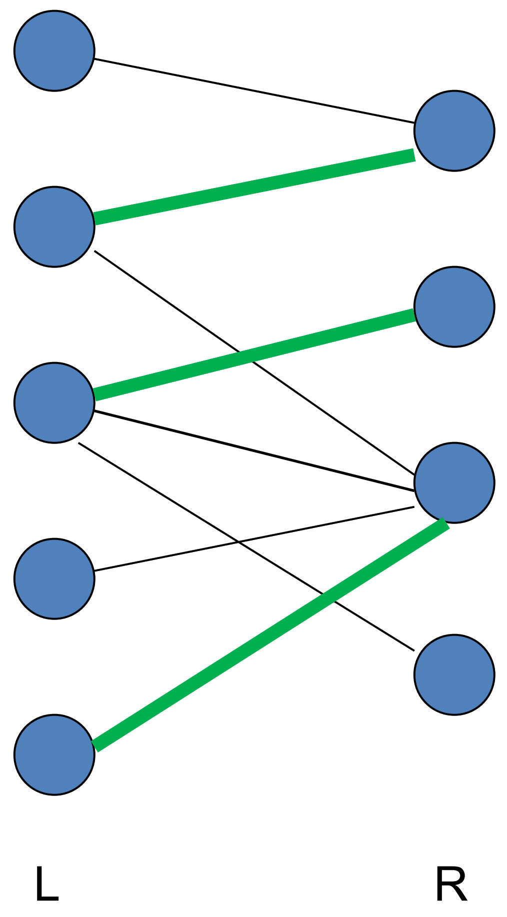
- **Bipartite graph:** A graph (V, E), where $V=L\cup R$, $L\cap R=\emptyset$, and for every $(u, v)\in E$, $u\in L$ and $v\in R$.
- Given an undirected graph $G = (V, E)$, a matching is a subset of edges $M\subseteq E$ such that for all vertices $v\in V$, at most one edge of M is incident on v.
- We say that a vertex $v\in V$ is matched by matching M if some edge in M is incident on v; otherwise, v is unmatched.
- **Maximum matching:** A matching of maximum cardinality, that is, a matching M such that for any matching M', we have

$$|M| \geq |M'|$$

Bipartite Matching Example

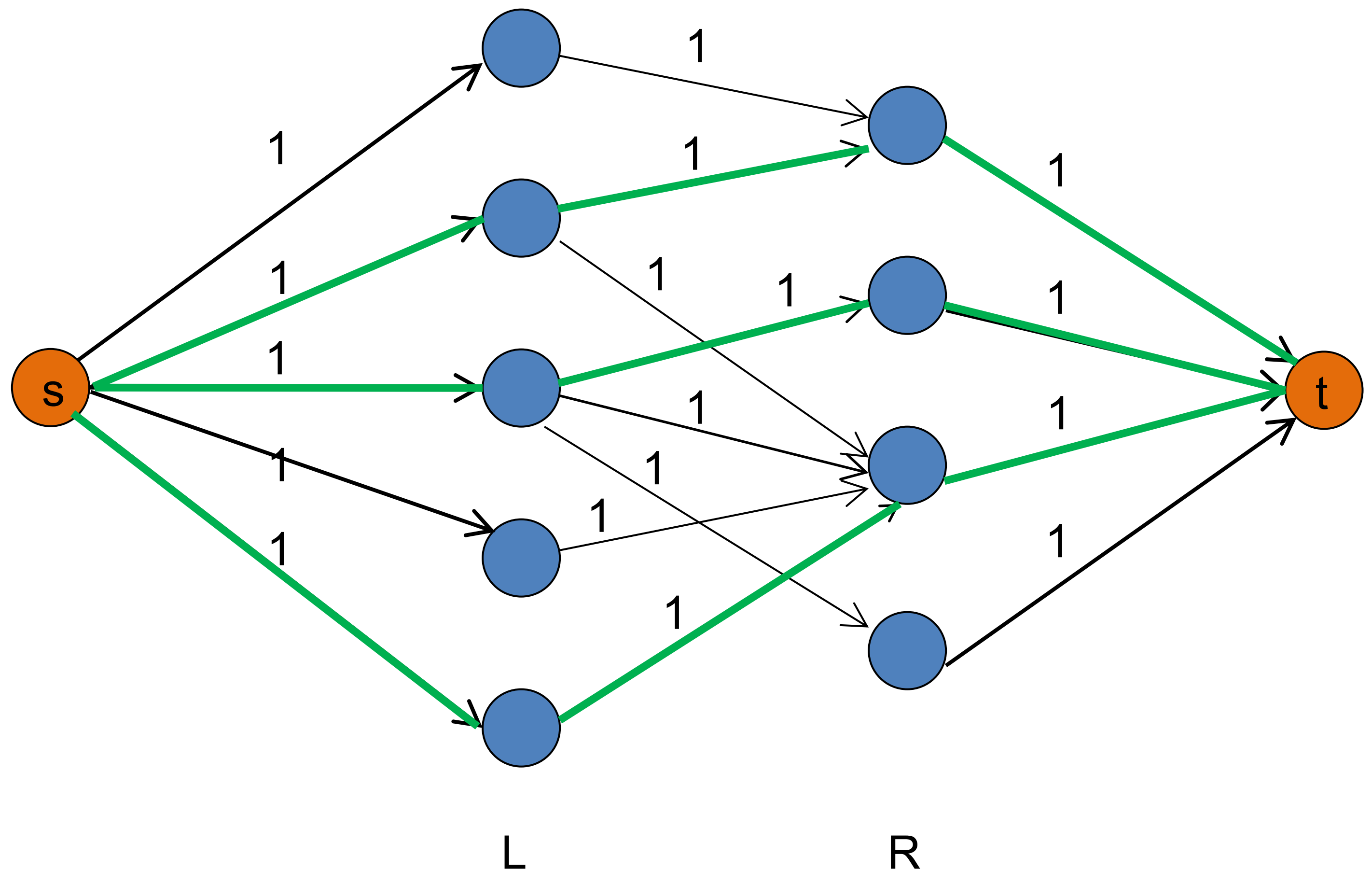


A matching with cardinality 2.



A maximum matching with cardinality 3.

Maximum Bipartite Matching using Max Flow



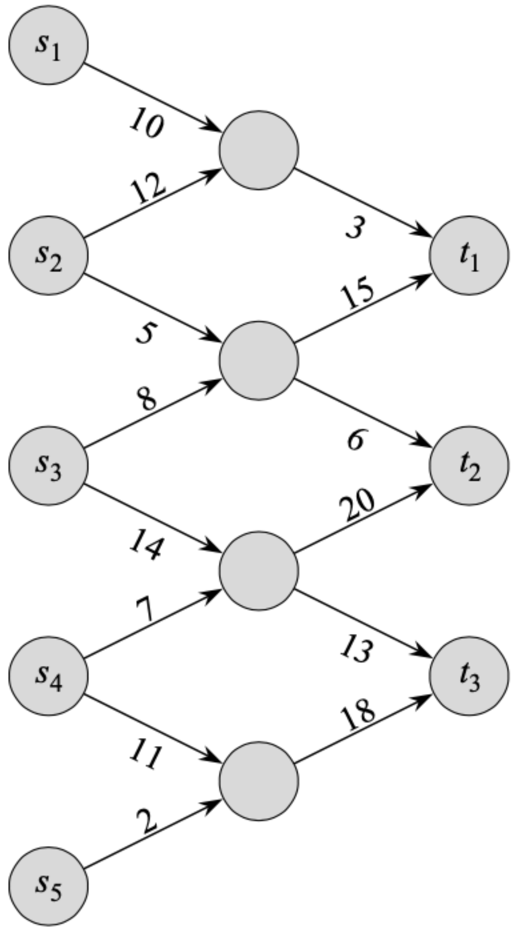
Theorem: A max flow in the *corresponding flow network* graph corresponds to a maximum matching in the original bipartite graph.

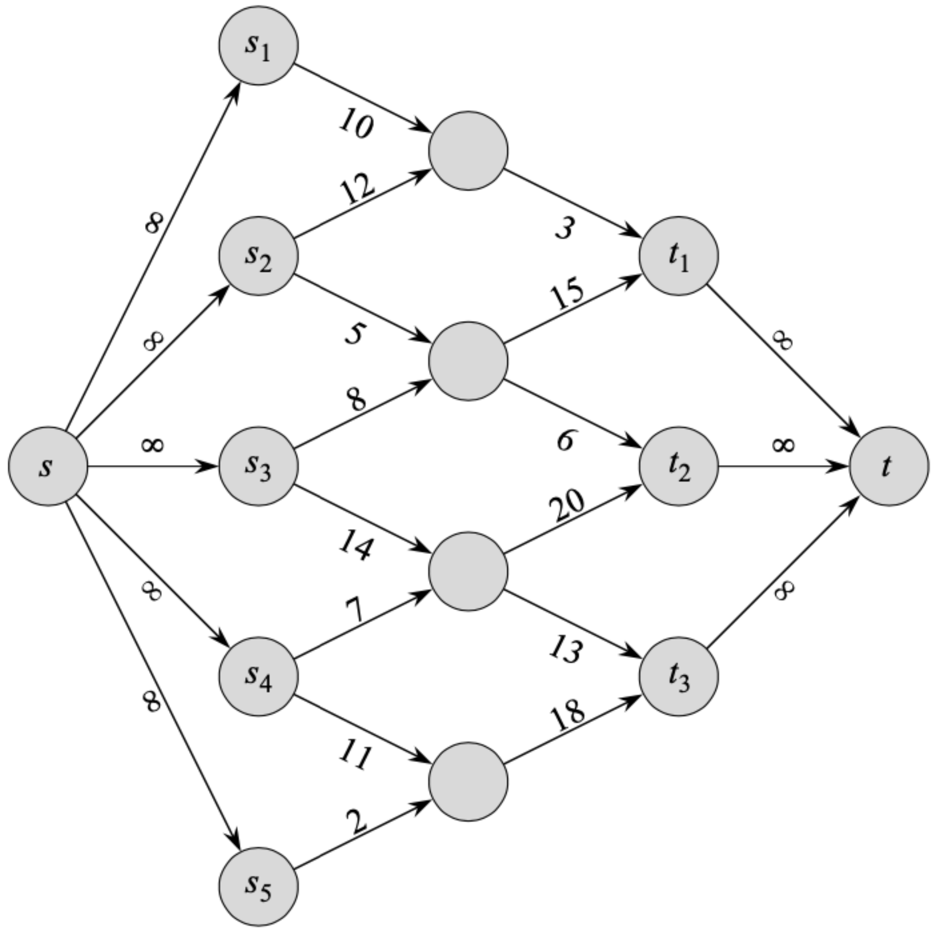
Proof

- **Integrality theorem:** If capacity of all edges are integer, then the max flow f produced by Ford-Fulkerson gives $|f|$ as integer. Also the flow over each edge (u, v) is also an integer.
  - Proof: Induction for each iteration of the algorithm.
- **Lemma 1:** If there is a matching of size $|M|$, then there's an integer-valued flow of size $|M|$.
  - If $(u, v) \in M$, $f(s,u) = f(u,v) = f(v,t) = 1$. For all other edges, $f(u,v) = 0$.
  - The net flow across cut $(\{s\} \cup L, R \cup \{t\})$ is equal to $|M|$. Hence, $|f| = |M|$.
- **Lemma 2:** If there's an integer-valued flow of size $|f|$, then there's a matching of size $|f|$.
  - Flow f corresponds to a valid matching. Why?
  - The net flow across cut $(\{s\} \cup L, R \cup \{t\})$ is equal to $|f|$. Hence, $|f| = |M|$.

Lemmas 1 & 2 ⇒ Max flow = Max matching

Networks w/ Multiple Sources and Sinks

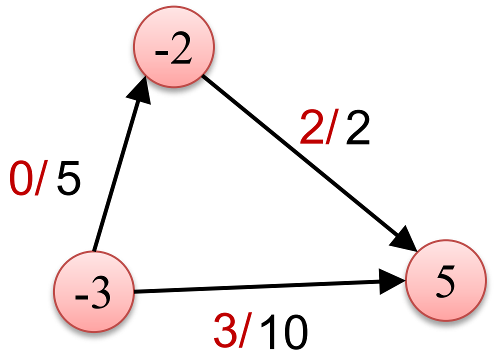


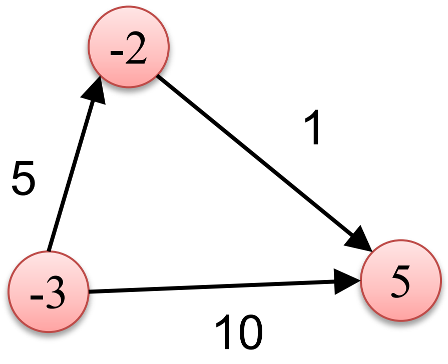


Add a ***supersource*** s and a ***supersink*** t.

Circulation with Supplies and Demands

- Given a digraph $G = (V, E)$ with edge capacities $c(e) \geq 0$ and node demands $d(v)$, a circulation is a function $f(e)$ which satisfies:
  - For each $e \in E$: $0 \leq f(e) \leq c(e)$ (capacity)
  - For each $v \in V$: $\Sigma_{e \to v} f(e) - \Sigma_{e \leftarrow v} f(e) = d(v)$ (flow conservation)
- Circulation problem: Given (V, E, c, d), does there exist a feasible circulation?





No circulation!

Reducing to the Maximum Flow Problem

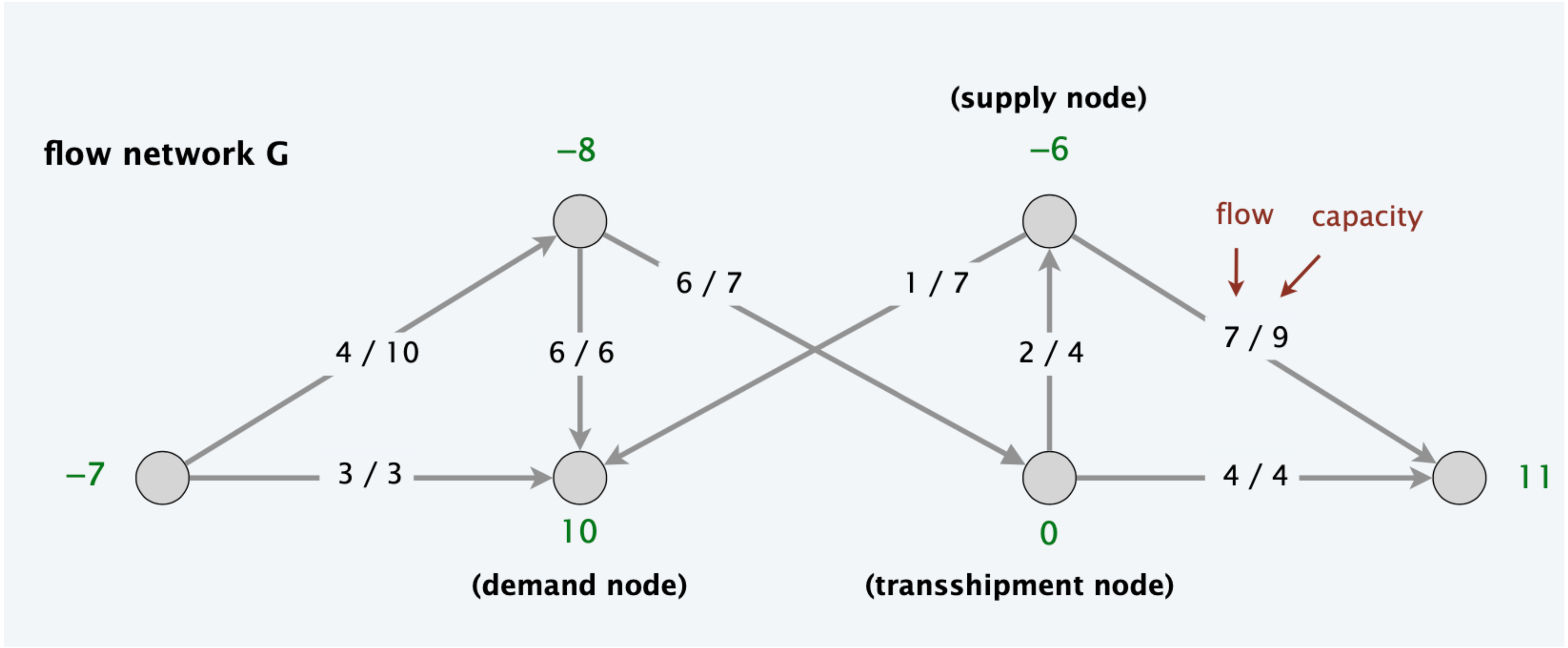
- Add new source s and sink t.
- For each v with $d(v) < 0$, add edge (s, v) with capacity $-d(v)$.
- For each v with $d(v) > 0$, add edge (v, t) with capacity $d(v)$.

- Claim: G has circulation iff G' has max flow of value

saturates all edges leaving s and entering t

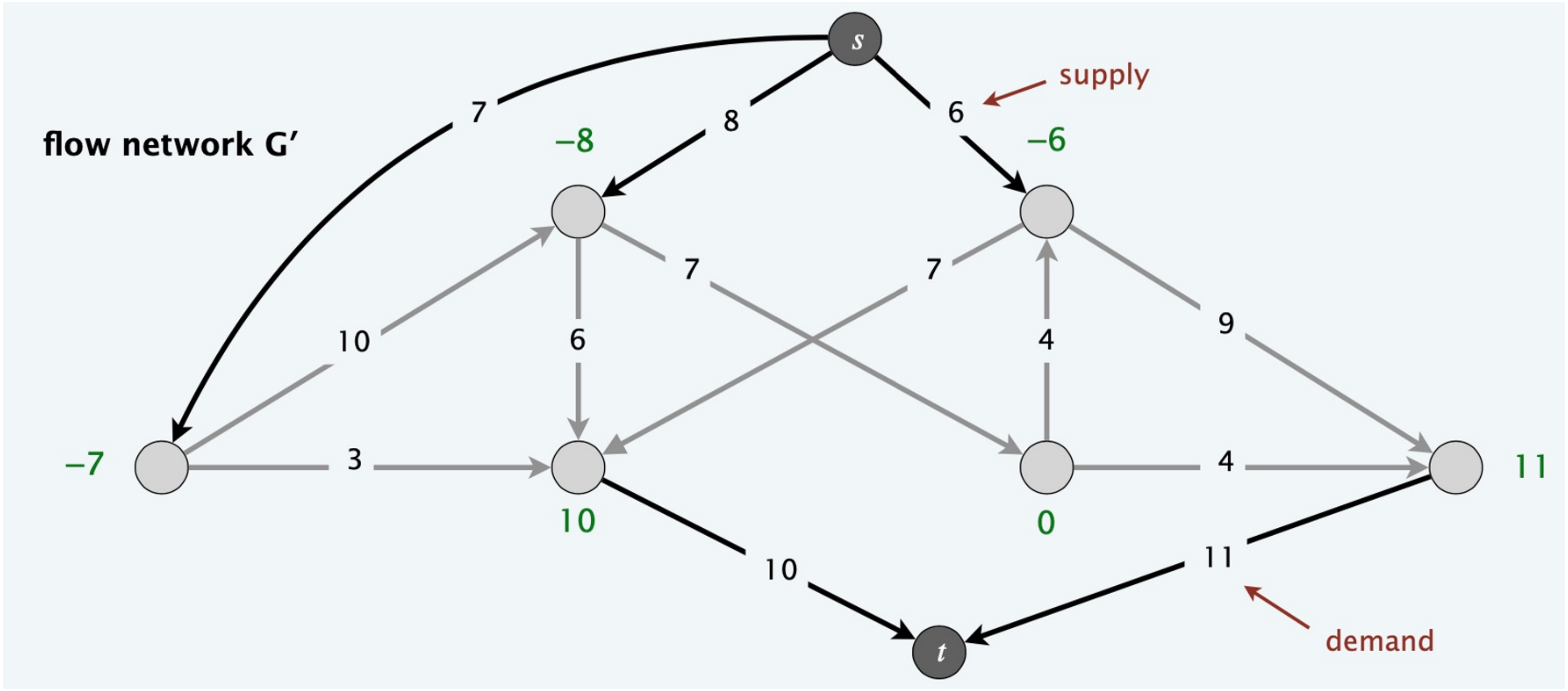
$$D = \Sigma_{v:d(v)>0} d(v) = \Sigma_{v:d(v)<0} - d(v)$$

Example 1



flow network G
(supply node)
–8
–6
flow
capacity
6 / 7
1 / 7
4 / 10
6 / 6
2 / 4
7 / 9
–7
3 / 3
4 / 4
11
10
0
(demand node)
(transshipment node)

Example 2


flow network G′
s
t
7
8
6
supply
−8
−6
7
7
10
6
4
9
−7
3
4
11
10
0
10
11
demand

Proof

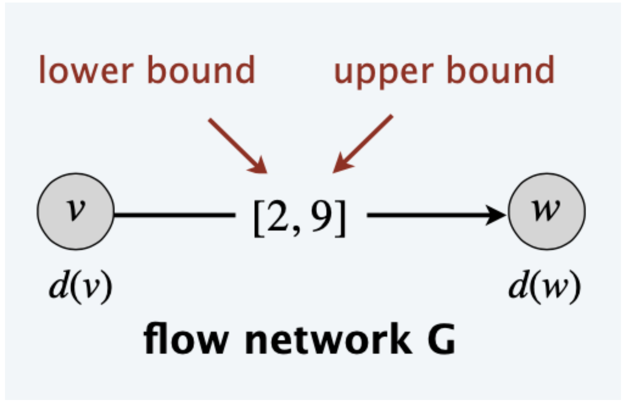
- Lemma 1: If a max flow saturates all outgoing links on *s*, a feasible circulation can be found (constructed.)

- Lemma 2: If a feasible circulation exists, a max flow can be found such that all outgoing links on *s* saturates.

Lemmas 1 & 2 ⇒ Done

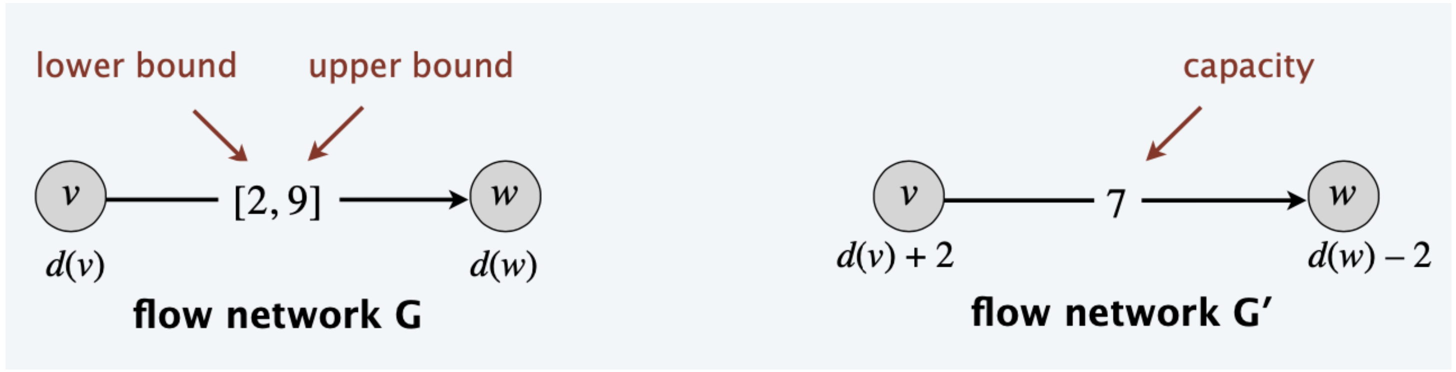
Circulation with Supplies, Demands, and lower bounds

- Given a digraph $G = (V, E)$ with edge capacities $c(e) \geq 0$, lower bounds $l(e) \geq 0$, and node demands $d(v)$, a circulation $f(e)$ is a function that satisfies:
  - For each $e \in E$: $l(e) \leq f(e) \leq c(e)$ (capacity)
  - For each $v \in V$: $\Sigma_{e\to v} f(e) - \Sigma_{e\leftarrow v} f(e) = d(v)$ (flow conservation)
- Circulation problem with lower bounds: Given (V, E, c, l, d), does there exist a feasible circulation?



Solution

- Max-flow formulation: Model lower bounds as circulation with demands.
  - Send $l(e)$ units of flow along edge e.
  - Update demands of both endpoints.

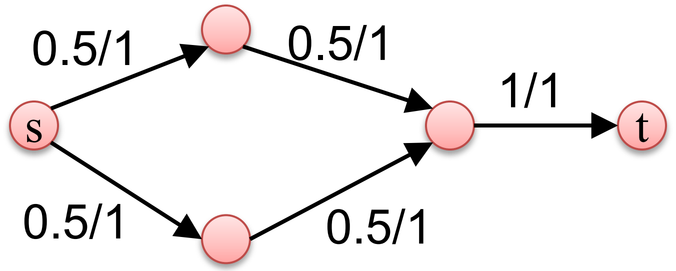


- Theorem: There exists a circulation in G, iff there exists a circulation in G'. Moreover, if all demands, capacities, and lower bounds in G are integers, then there exists a circulation in G that is integer-valued.
  - Proof sketch: $f(e)$ is a circulation in G, iff $f'(e) = f(e) - l(e)$ is a circulation in G'.

Sample Problems

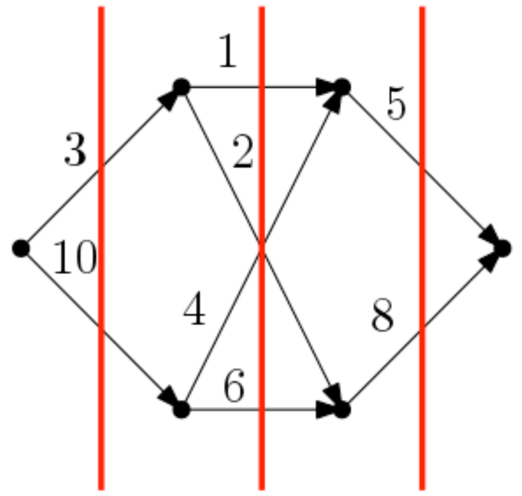
True or False?

- In a network with source s and sink t where each edge capacity is a positive integer, there is always a max s-t flow where the flow assigned to each edge is an integer.
- Maximum value of an s-t flow could be less than the capacity of a given s-t cut in a flow network.
- In a flow network, if we increase the capacity of an edge that happens to be on a minimum cut, this will increase the max flow in the network.
- A network with integer capacity flow may have an edge with non-integer flow in the max flow.



True or False? (cont'd)

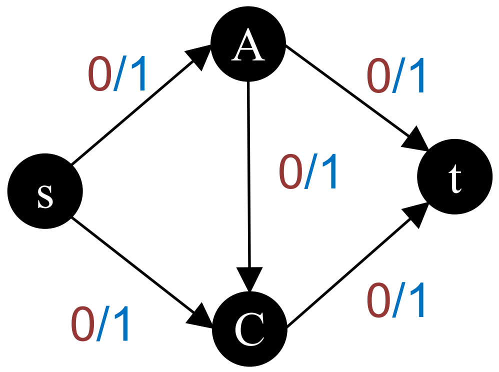
- The best worst-case time complexity to solve the max flow problem is $O(Cm)$ where C is the total capacity of the edges leaving the source and m is the number of edges in the network.
- A flow network with unique edge capacities has a unique min cut.

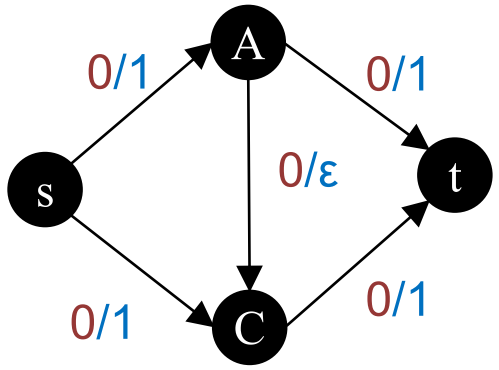


Runtime?

- What is running time of Ford–Fulkerson algorithm to find a max-cardinality matching in a bipartite graph with m edges and $|L| = |R| = n$?
  - *A.* $O(m + n)$
  - *B.* $O(mn)$
  - *C.* $O(mn^2)$
  - *D.* $O(m^2n)$

What happens if capacities are not integer?





What happens if $\varepsilon \to 0$?

Multiple Choice

- The Ford–Fulkerson algorithm is guaranteed to terminate if the edge capacities are ...
  A. Rational numbers.
  B. Real numbers.
  C. Both A and B.
  D. Neither A nor B.

Dinner Party Planning

- At a dinner party, there are n families $\{a_1, a_2, \ldots, a_n\}$ and m tables $\{b_1, b_2, \ldots, b_m\}$.
- The i^{th} family a_i has g_i members and the j^{th} table b_j has h_j seats.
- Everyone is interested in making new friends and the dinner party planner wants to seat people such that no two members of the same family are seated in the same table.
- Design an algorithm that decides if there exists a seating assignment such that everyone is seated and no two members of the same family are seated at the same table.