

Algorithms

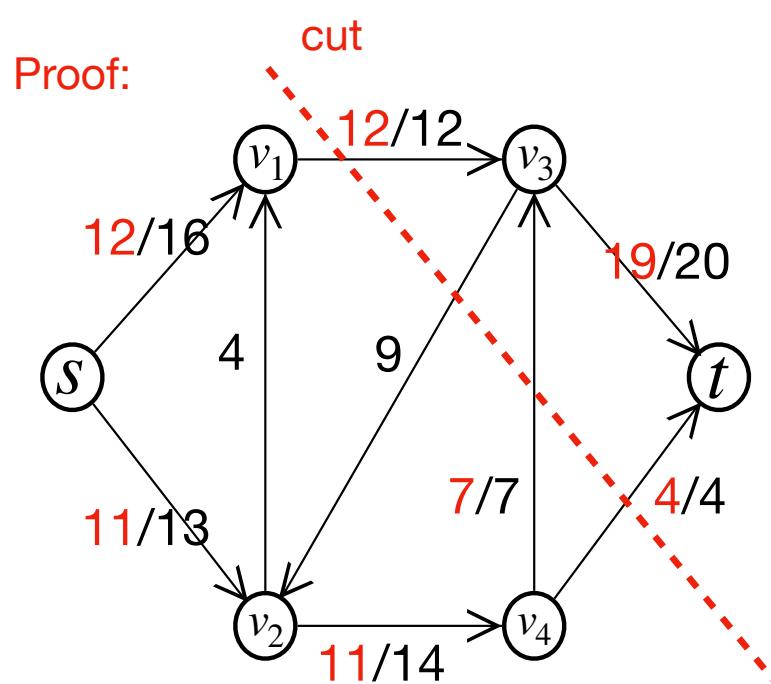
Lecture 22: Maximum Flow (Part 2)

Anxiao (Andrew) Jiang

CH 26. Maximum Flow

Ford-Fulkerson Method

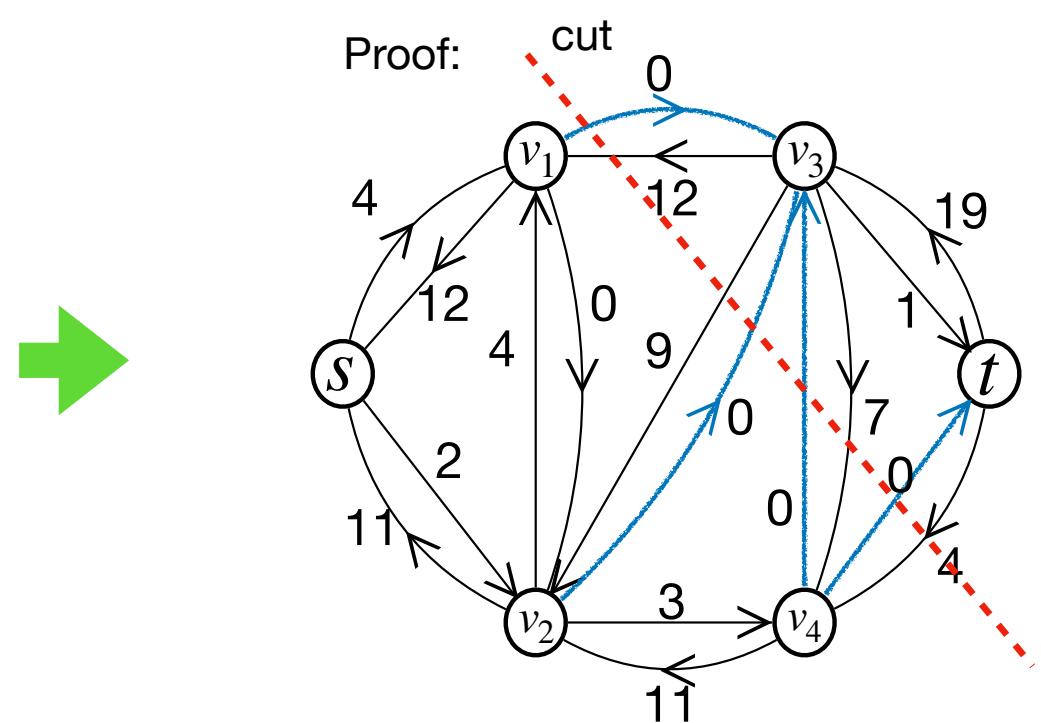
Original network



Size of flow: $12+11=19+4=23$

Is it optimal? YES.

Residual network



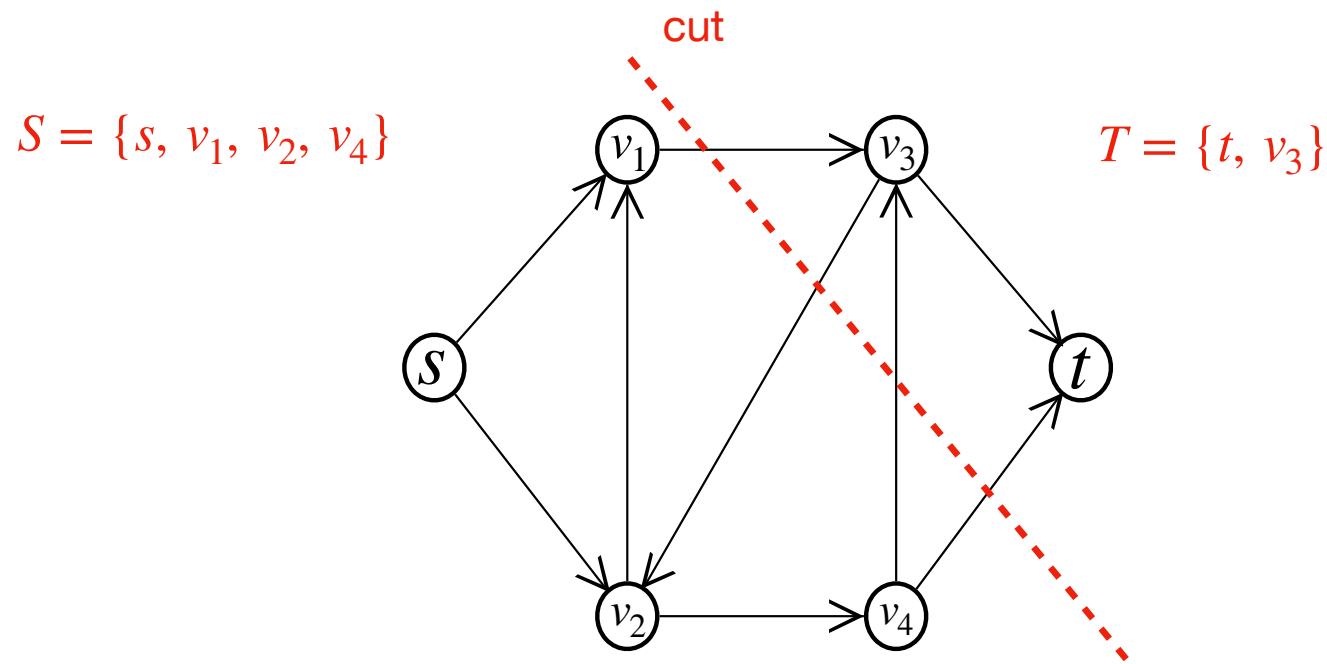
Is there still a path from s to t of residual capacity > 0 ?

NO

Theorem: The Ford-Fulkerson Method finds a maximum flow.

Theorem: The Ford-Fulkerson Method finds a maximum flow.

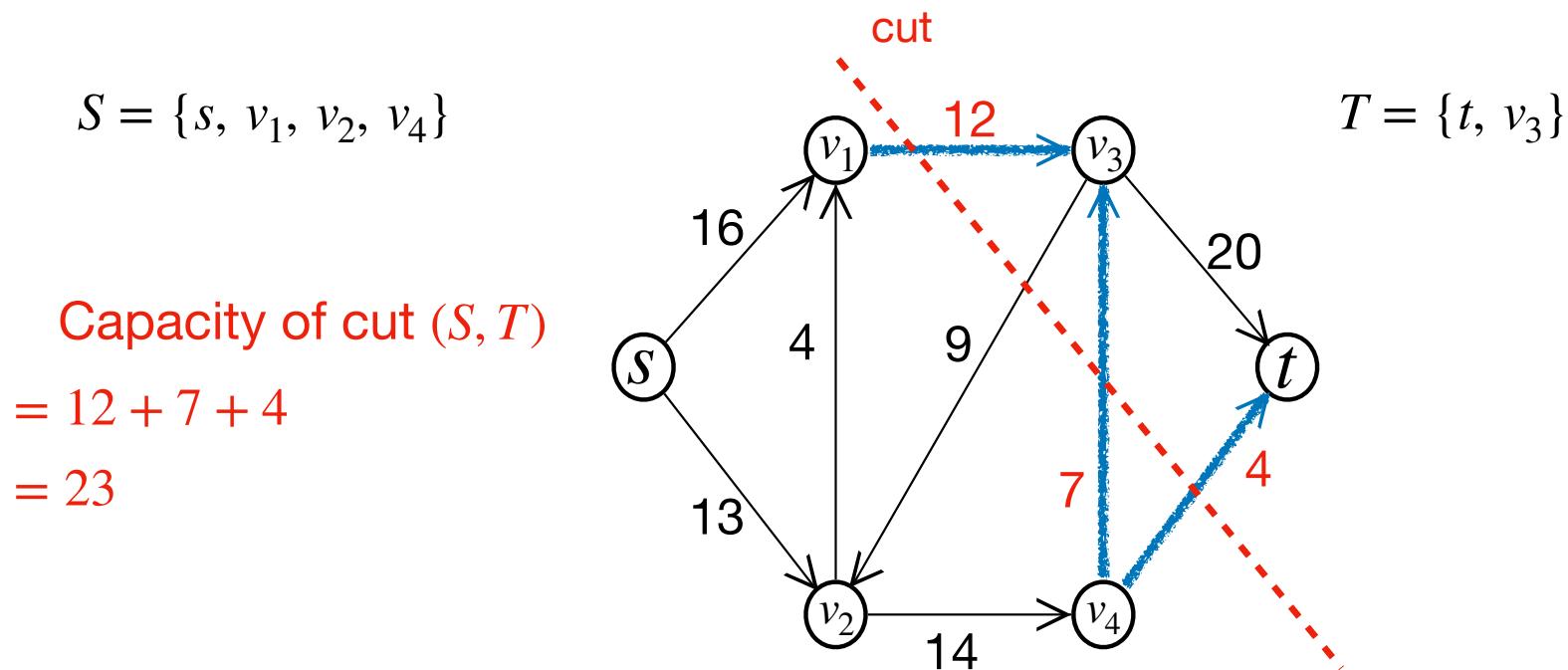
Cut: A partition of the nodes in the network $G=(V,E)$ into two sets S and T , such that $s \in S$ and $t \in T$.



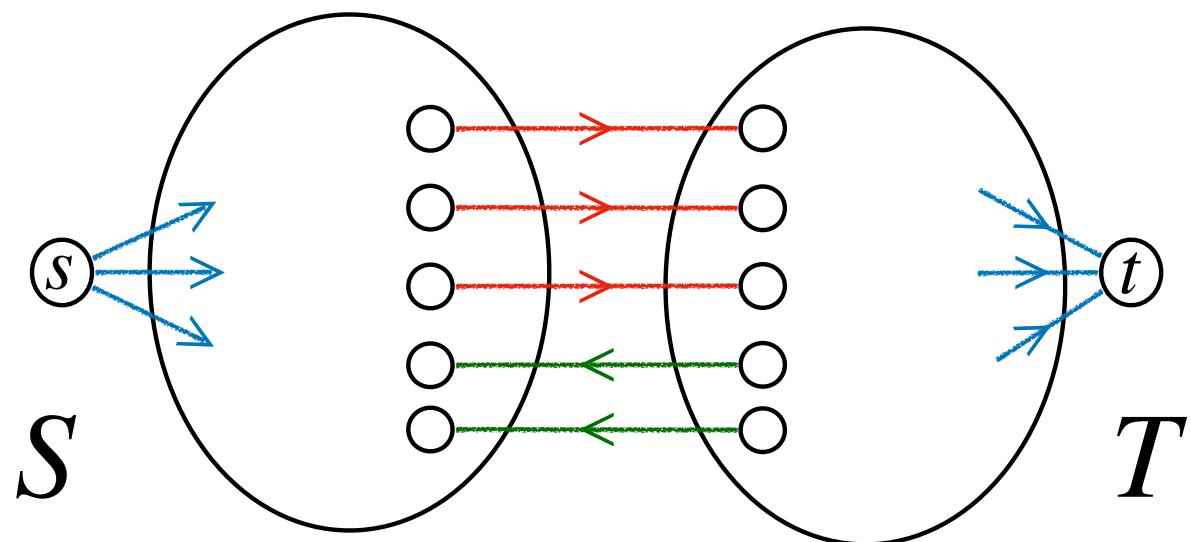
Theorem: The Ford-Fulkerson Method finds a maximum flow.

Cut: A partition of the nodes in the network $G=(V,E)$ into two sets S and T , such that $s \in S$ and $t \in T$.

Capacity of Cut (S,T) : the total capacity of the edges from S to T .



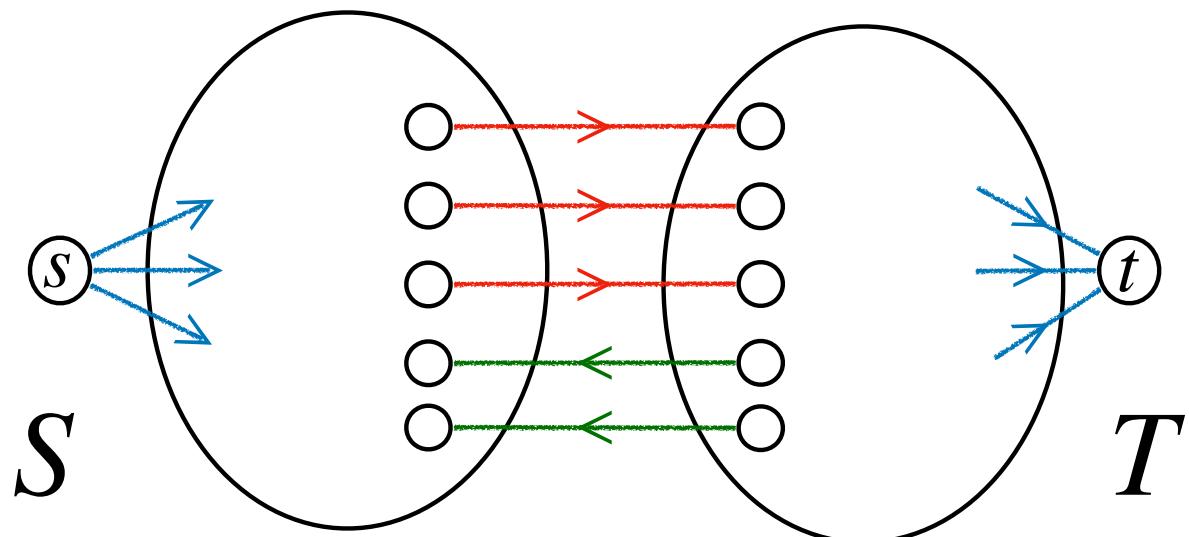
Lemma: The size of any flow is **less than or equal to** the capacity of any cut.



Lemma: The size of any flow is **less than or equal to** the capacity of any cut.

Proof: Size of flow (leaving source s or entering sink t)

$$= \text{flow from } S \text{ to } T - \text{flow from } T \text{ to } S$$

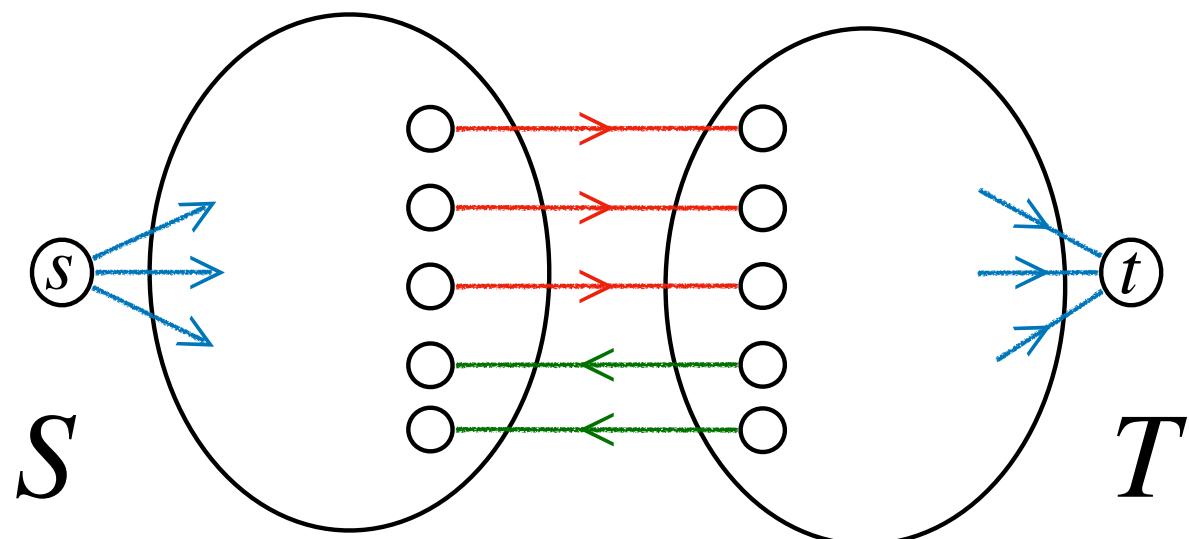


Lemma: The size of any flow is **less than or equal to** the capacity of any cut.

Proof: Size of flow (leaving source s or entering sink t)

$$= \text{flow from } S \text{ to } T - \text{flow from } T \text{ to } S$$

$$\leq \text{flow from } S \text{ to } T$$



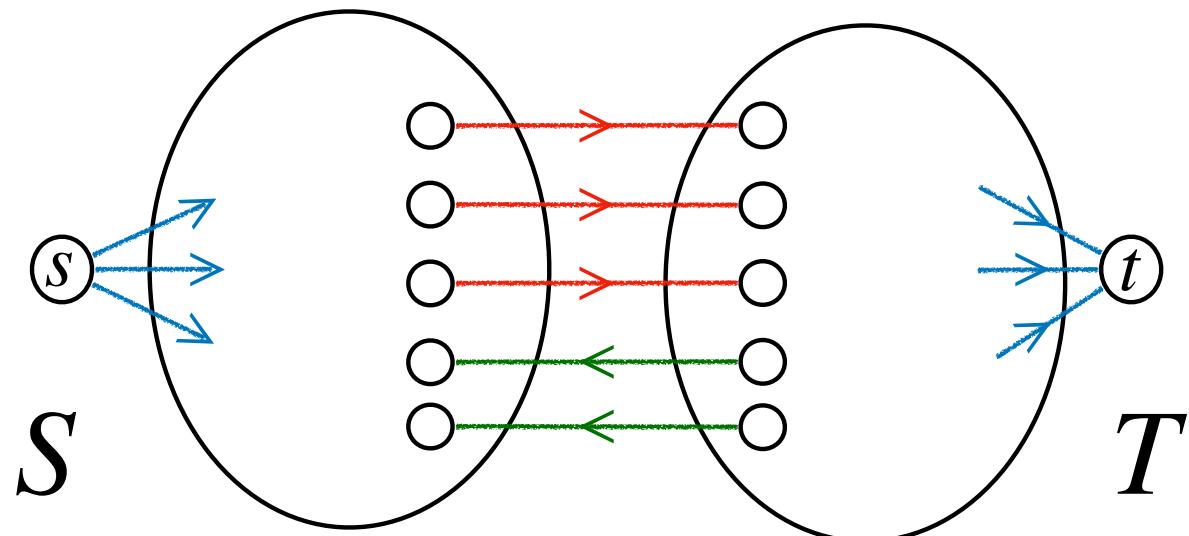
Lemma: The size of any flow is **less than or equal to** the capacity of any cut.

Proof: Size of flow (leaving source s or entering sink t)

$$= \text{flow from } S \text{ to } T - \text{flow from } T \text{ to } S$$

$$\leq \text{flow from } S \text{ to } T$$

$$\leq \text{total capacity of the edges from } S \text{ to } T$$



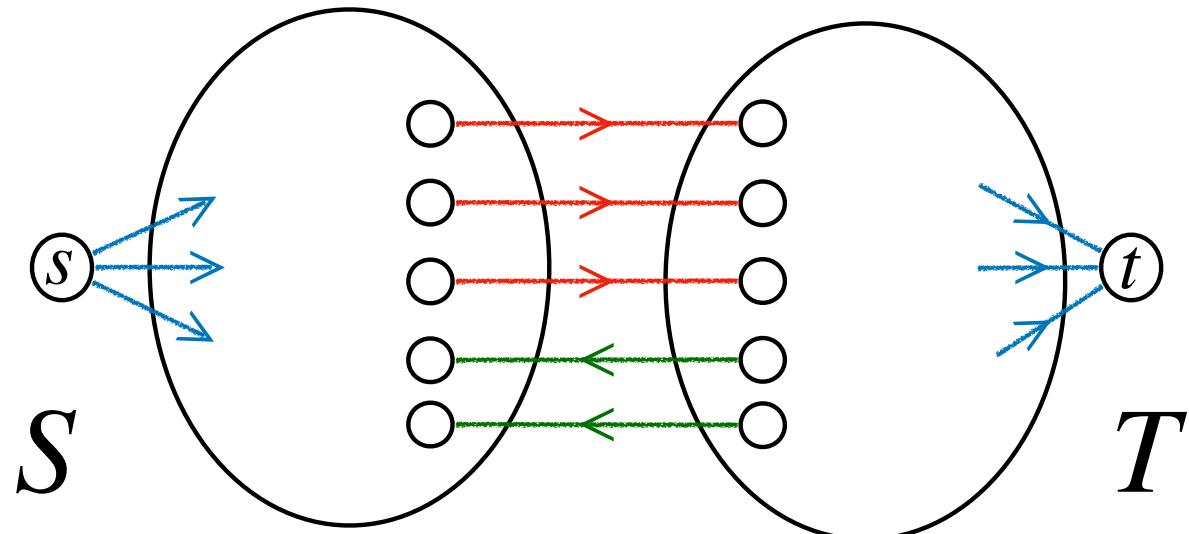
Lemma: The size of any flow is **less than or equal to** the capacity of any cut.

Proof: Size of flow (leaving source s or entering sink t)

$$= \text{flow from } S \text{ to } T - \text{flow from } T \text{ to } S$$

$$\leq \text{flow from } S \text{ to } T$$

$$\leq \text{total capacity of the edges from } S \text{ to } T = \text{capacity of the cut } (S, T)$$



Lemma: The size of **any flow** is less than or equal to the capacity of **any cut**.

Maximum flow: a flow from s to t of maximum size.

Minimum cut: a cut (S,T) of minimum capacity.

$$\text{size of maximum flow} \leq \text{capacity of minimum cut}$$

Question: is it true that

$$\text{size of maximum flow} = \text{capacity of minimum cut}$$

Answer: Yes. We will prove that.

Theorem: Let f be a flow in a network $G = (V, E)$ with source s and sink t .

Then these three conditions are equivalent:

1. f is a maximum flow in G .
2. The residual network G_f has no path from s to t with positive residual capacity.
3. The size of the flow f equals the capacity of some cut (S, T) .

Theorem: Let f be a flow in a network $G = (V, E)$ with source s and sink t .

Then these three conditions are equivalent:

1. f is a maximum flow in G .
2. The residual network G_f has no path from s to t with positive residual capacity.
3. The size of the flow f equals the capacity of some cut (S, T) .

Proof: $1 \rightarrow 2$: By contradiction.

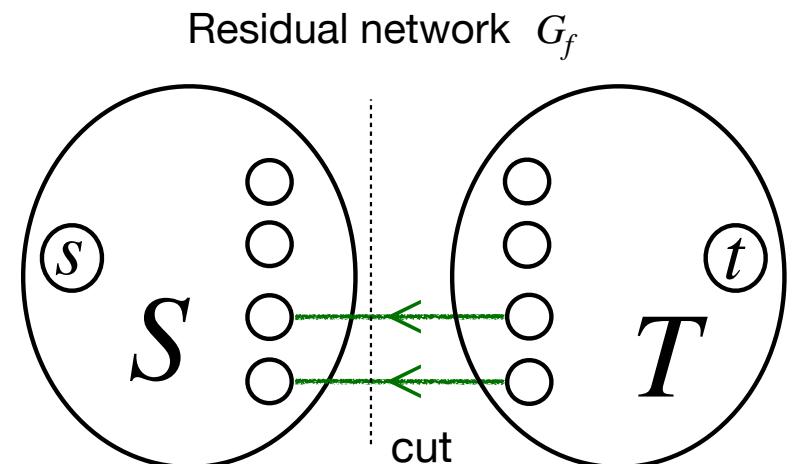
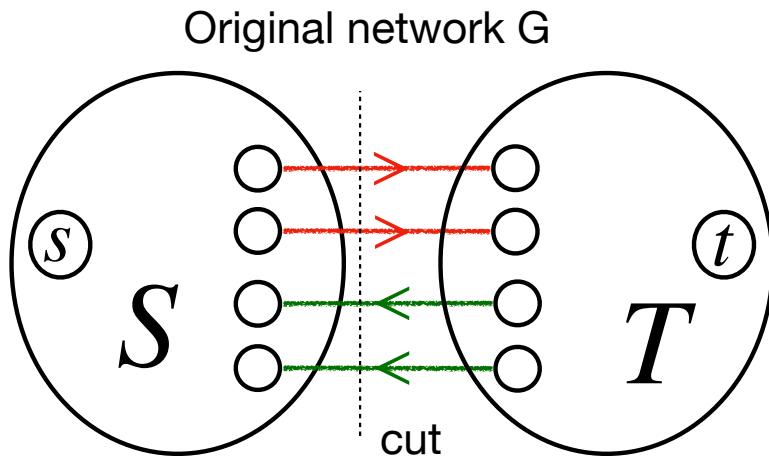
Theorem: Let f be a flow in a network $G = (V, E)$ with source s and sink t .

Then these three conditions are equivalent:

1. f is a maximum flow in G .
2. The residual network G_f has no path from s to t with positive residual capacity.
3. The size of the flow f equals the capacity of some cut (S, T) .

Proof: $1 \rightarrow 2$: By contradiction.

$2 \rightarrow 3$: S : the set of nodes that the source s can reach in the residual network G_f
 T : the remaining nodes

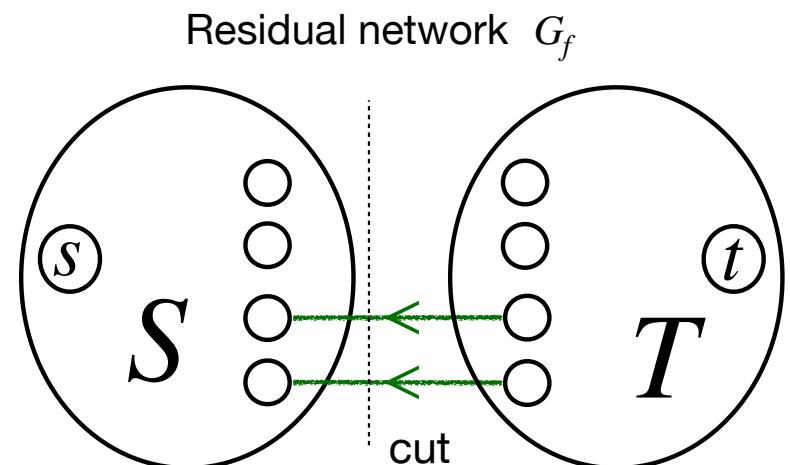
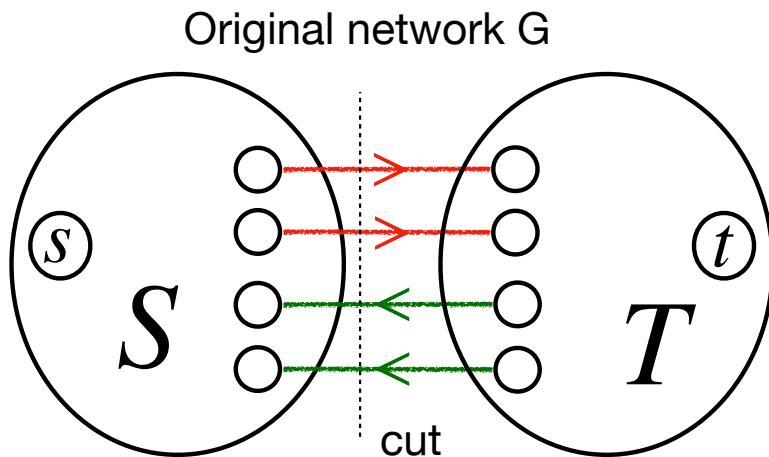


In the original network G:

“Forward” edges from S to T are saturated by flow

“Backward” edges from T to S have no flow at all

Therefore, the size of the flow f equals the capacity of this cut (S, T) .



Theorem: Let f be a flow in a network $G = (V, E)$ with source s and sink t .

Then these three conditions are equivalent:

1. f is a maximum flow in G .
2. The residual network G_f has no path from s to t with positive residual capacity.
3. The size of the flow f equals the capacity of some cut (S, T) .

Proof: $1 \rightarrow 2$: By contradiction.

$2 \rightarrow 3$: Done.

$3 \rightarrow 1$: Because we already know

$$\text{size of any flow} \leq \text{capacity of any cut}$$

Theorem: Let f be a flow in a network $G = (V, E)$ with source s and sink t .

Then these three conditions are equivalent:

1. f is a maximum flow in G .
2. The residual network G_f has no path from s to t with positive residual capacity.
3. The size of the flow f equals the capacity of some cut (S, T) .

Proof: $1 \rightarrow 2$: By contradiction.

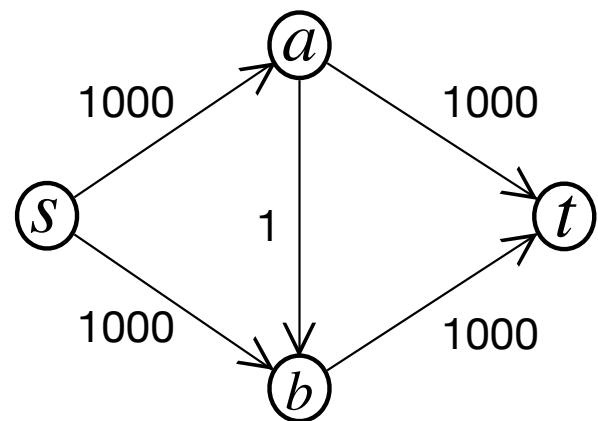
$2 \rightarrow 3$: Done.

$3 \rightarrow 1$: Because we already know

$$\text{size of any flow} \leq \text{capacity of any cut}$$

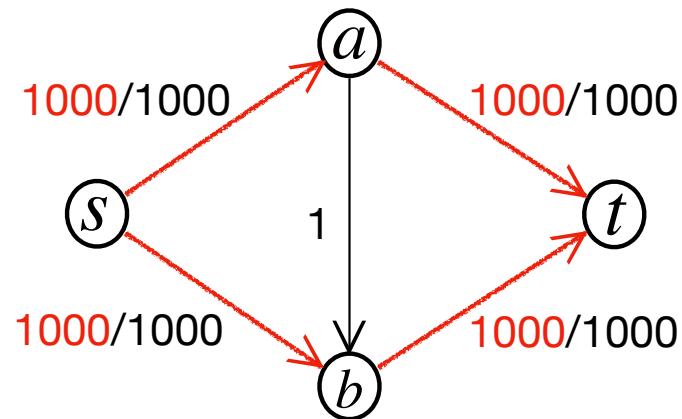
Theorem: The Ford-Fulkerson Method finds a maximum flow.

Time complexity of the Ford-Fulkerson Method



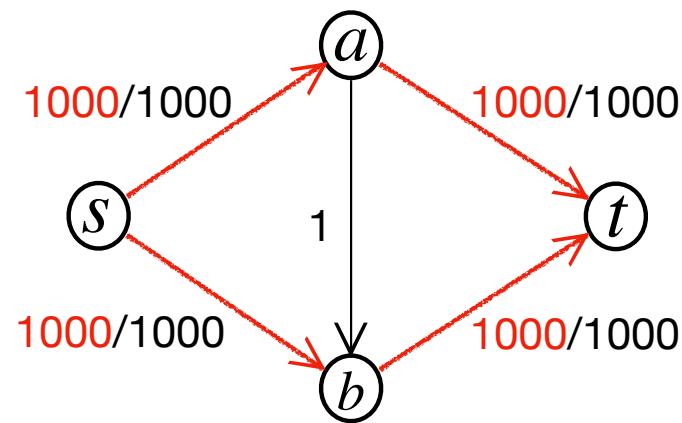
Time complexity of the Ford-Fulkerson Method

Maximum Flow: 2000

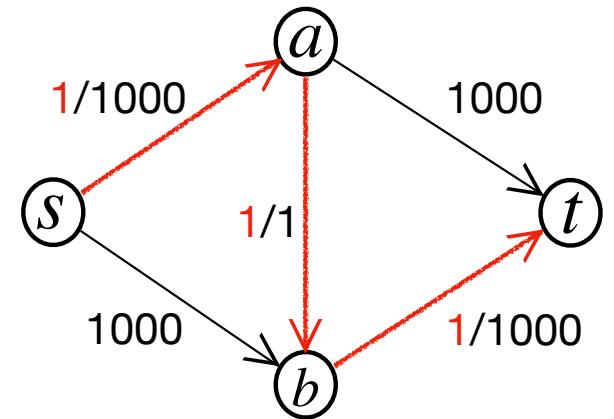


Time complexity of the Ford-Fulkerson Method

Maximum Flow: 2000



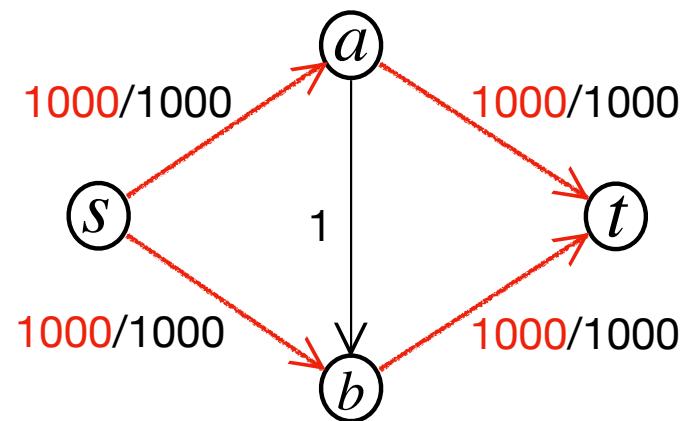
Naive way to run Ford-Fulkerson Method:



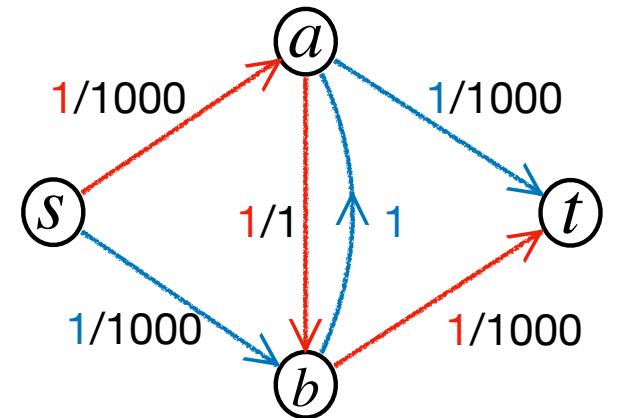
Current flow size = 1

Time complexity of the Ford-Fulkerson Method

Maximum Flow: 2000



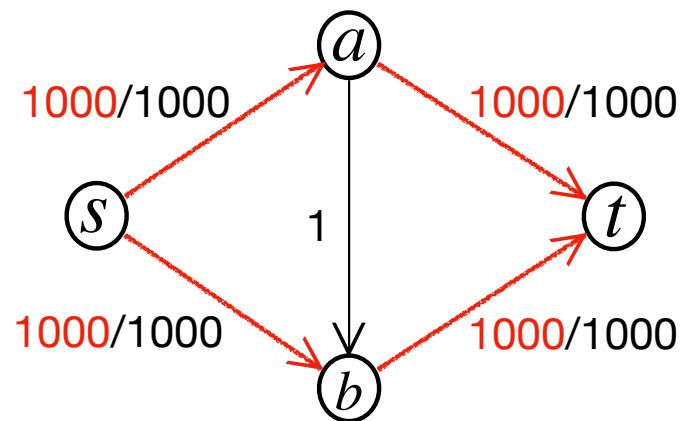
Naive way to run Ford-Fulkerson Method:



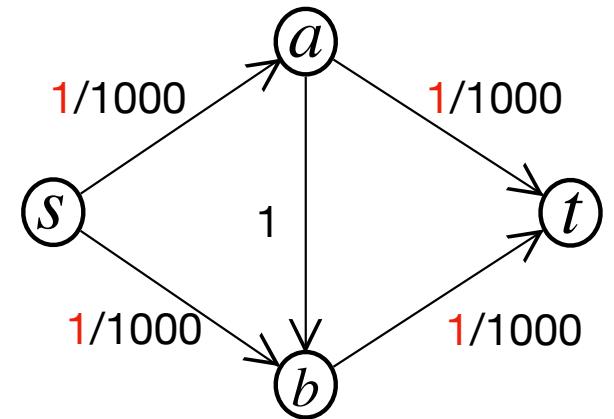
Current flow size = 2

Time complexity of the Ford-Fulkerson Method

Maximum Flow: 2000



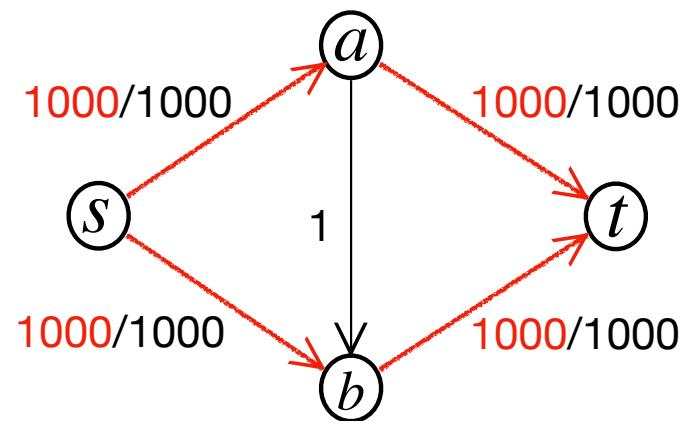
Naive way to run Ford-Fulkerson Method:



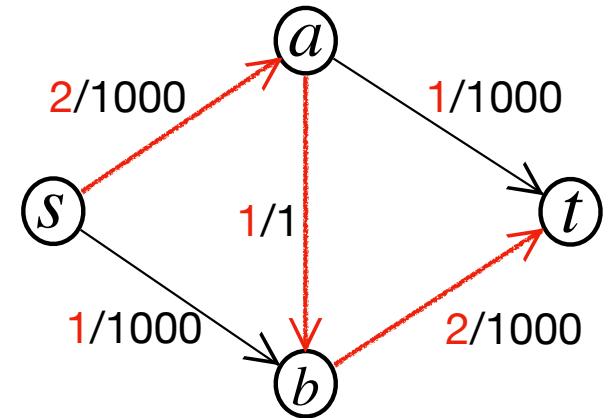
Current flow size = 2

Time complexity of the Ford-Fulkerson Method

Maximum Flow: 2000



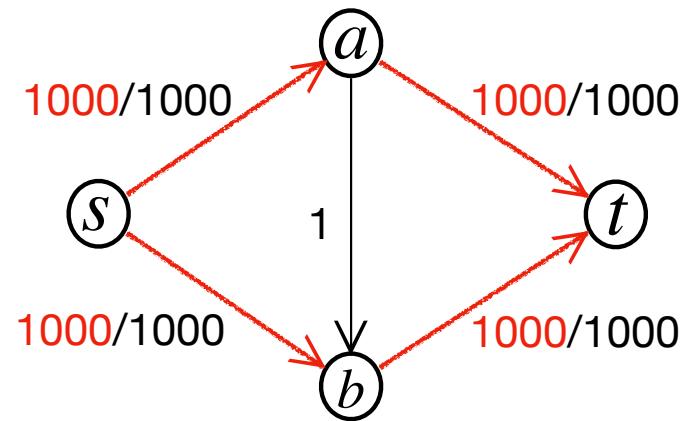
Naive way to run Ford-Fulkerson Method:



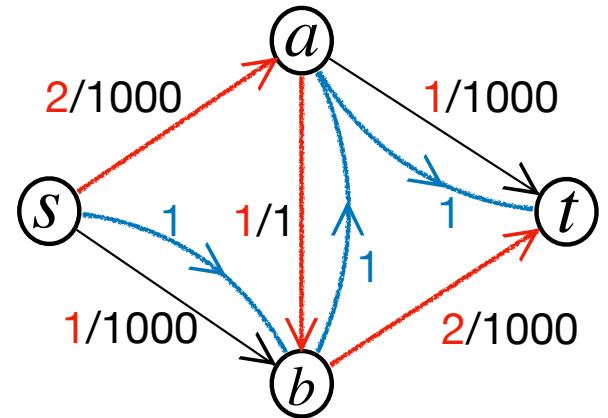
Current flow size = 3

Time complexity of the Ford-Fulkerson Method

Maximum Flow: 2000



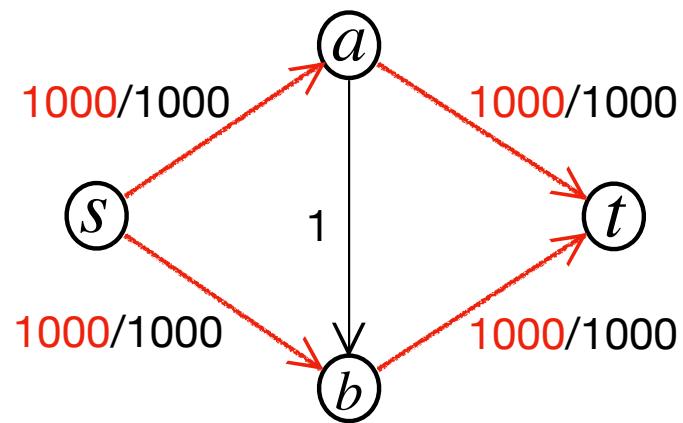
Naive way to run Ford-Fulkerson Method:



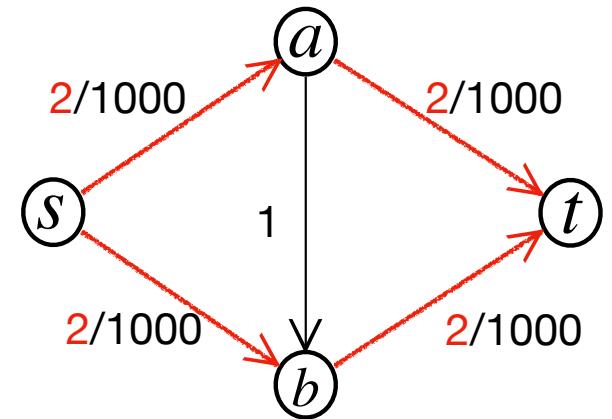
Current flow size = 4

Time complexity of the Ford-Fulkerson Method

Maximum Flow: 2000



Naive way to run Ford-Fulkerson Method:

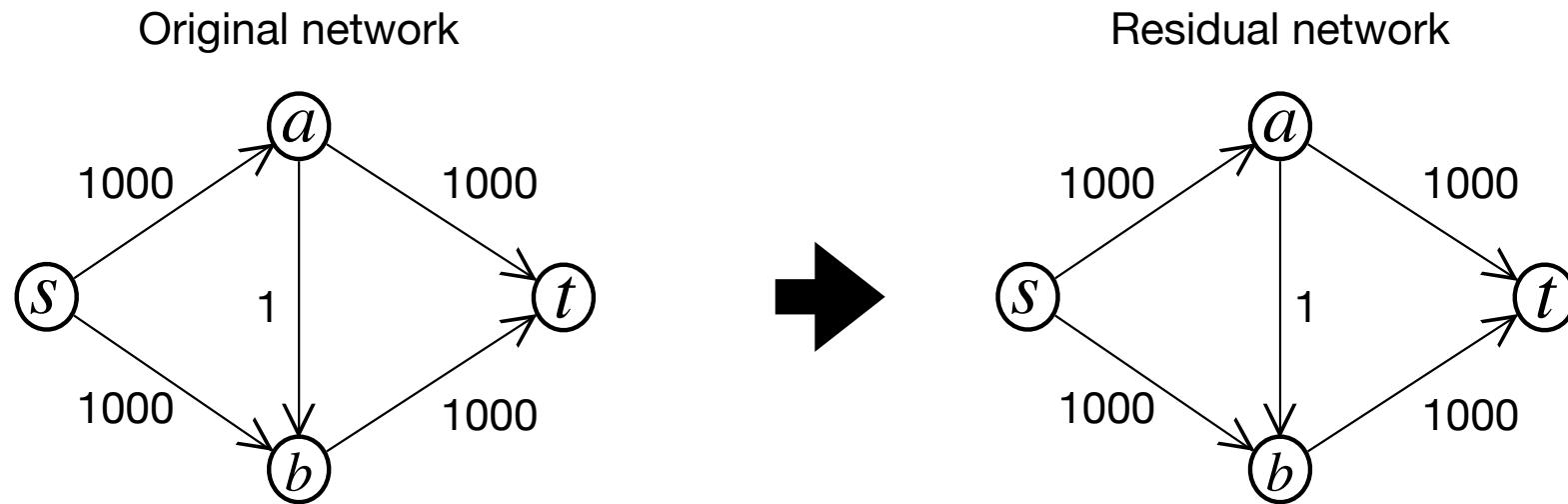


Current flow size = 4

Time complexity: too high

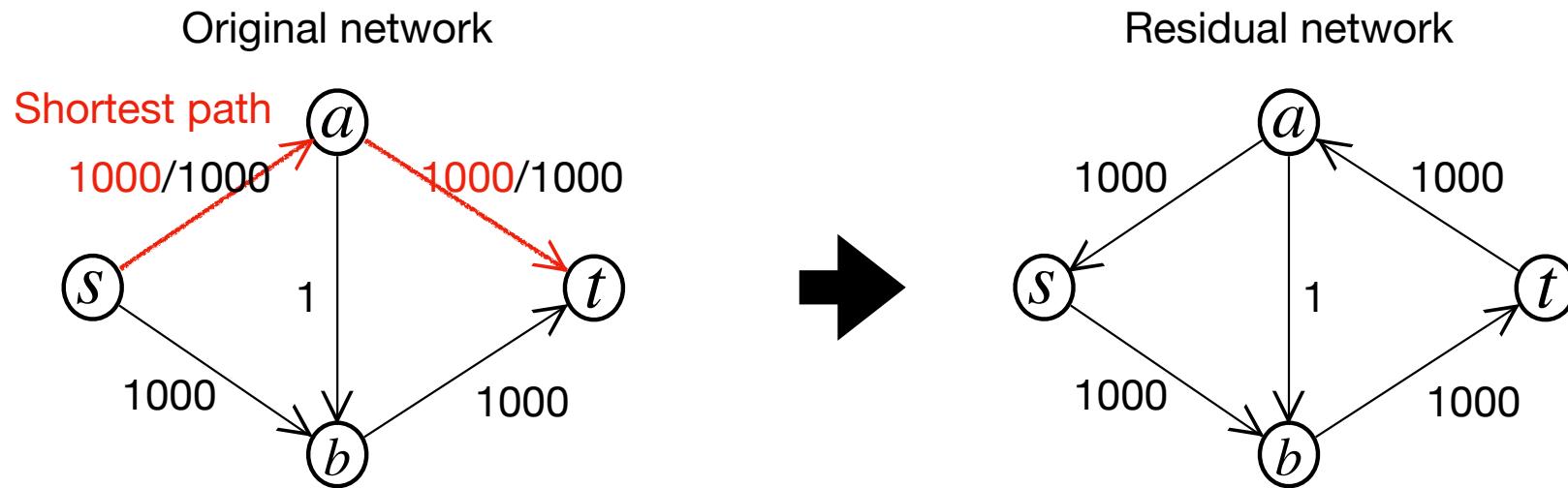
Edmonds-Karp Algorithm (an efficient way to implement the Ford-Fulkerson Method):

Each time, pick the **shortest path** from s to t in the residual network,
and augment the flow along that path.



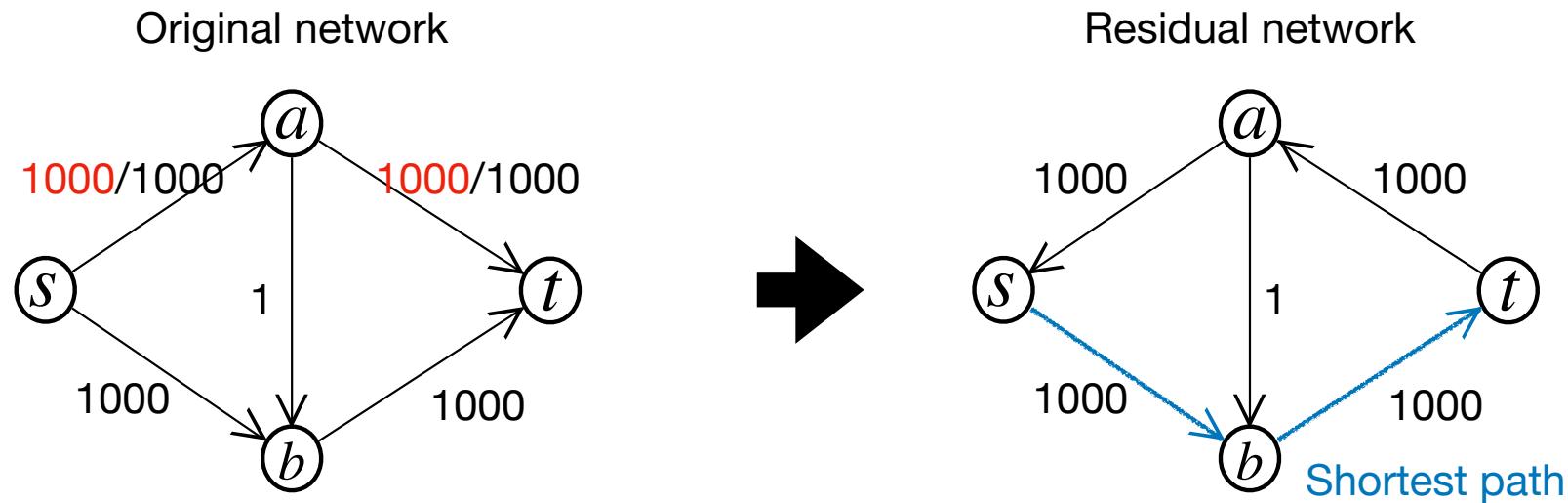
Edmonds-Karp Algorithm (an efficient way to implement the Ford-Fulkerson Method):

Each time, pick the **shortest path** from s to t in the residual network,
and augment the flow along that path.



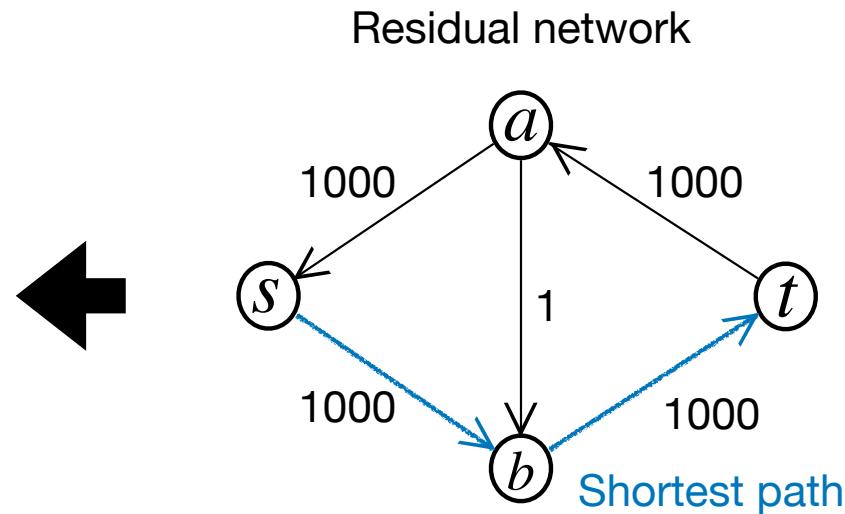
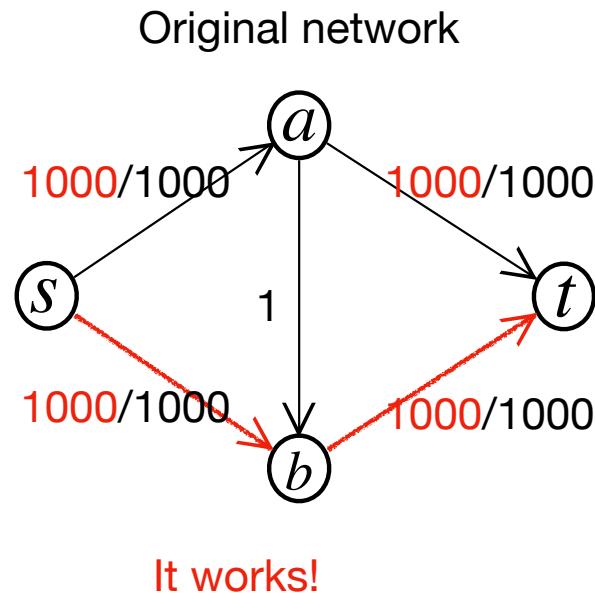
Edmonds-Karp Algorithm (an efficient way to implement the Ford-Fulkerson Method):

Each time, pick the **shortest path** from s to t in the residual network,
and augment the flow along that path.



Edmonds-Karp Algorithm (an efficient way to implement the Ford-Fulkerson Method):

Each time, pick the **shortest path** from s to t in the residual network,
and augment the flow along that path.



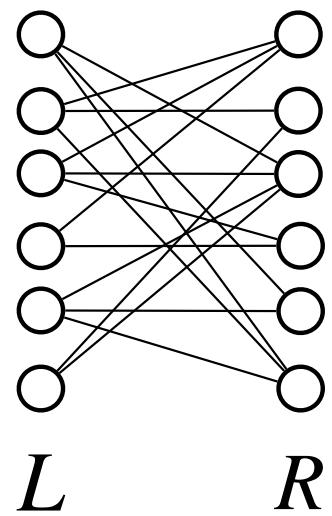
Edmonds-Karp Algorithm (an efficient way to implement the Ford-Fulkerson Method):

Each time, pick the shortest path from s to t in the residual network,
and augment the flow along that path.

Time Complexity: $O(VE^2)$

Application: Maximum Bipartite Matching

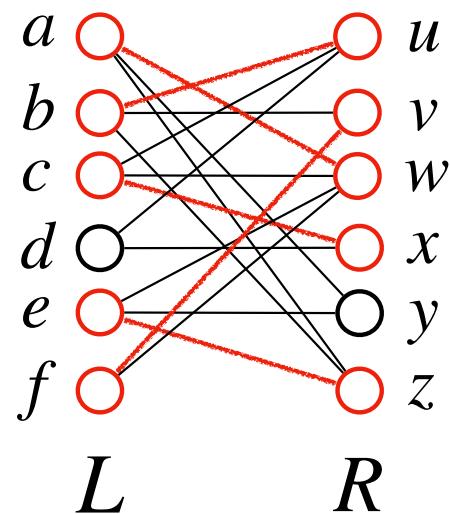
Bipartite Graph: A graph $G = (V, E)$ is bipartite if we can partition the nodes into L and R , such that every edge is between a node in L and a node in R .



Application: Maximum Bipartite Matching

Bipartite Graph: A graph $G = (V, E)$ is bipartite if we can partition the nodes into L and R , such that every edge is between a node in L and a node in R .

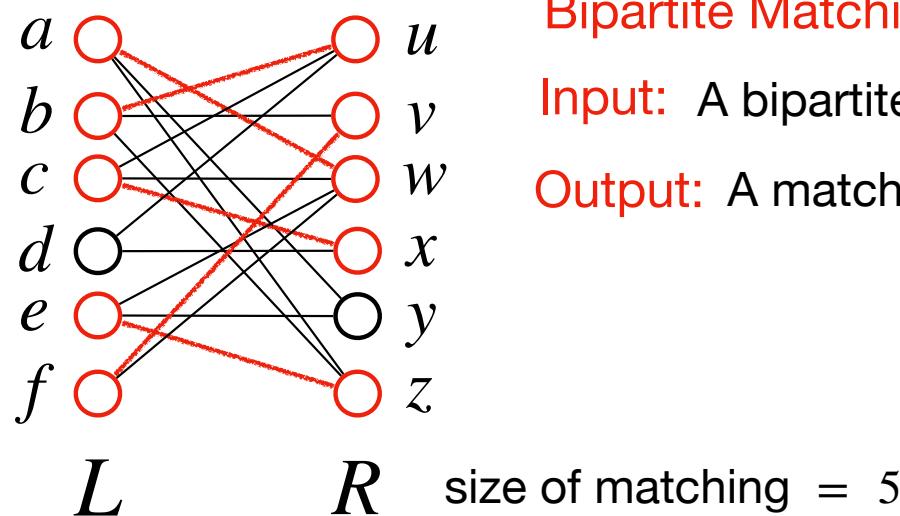
Matching: Given an undirected graph $G = (V, E)$, a matching is a subset of edges $S \subseteq E$ such that every node $v \in V$ is an endpoint of at most one edge in S .



Application: Maximum Bipartite Matching

Bipartite Graph: A graph $G = (V, E)$ is bipartite if we can partition the nodes into L and R , such that every edge is between a node in L and a node in R .

Matching: Given an undirected graph $G = (V, E)$, a matching is a subset of edges $S \subseteq E$ such that every node $v \in V$ is an endpoint of at most one edge in S .



Bipartite Matching Problem:

Input: A bipartite graph $G = (L \cup R, E)$.

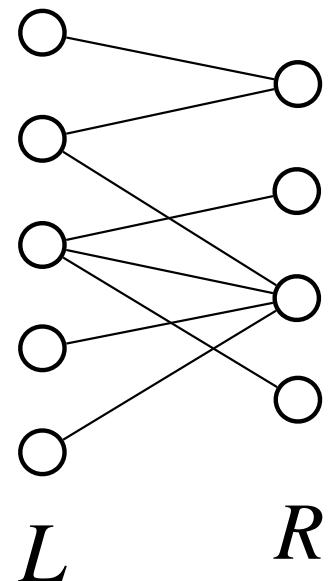
Output: A matching of maximum size.

Bipartite Matching Problem:

Input: A bipartite graph $G = (L \cup R, E)$.

Output: A matching of maximum size.

Example instance:

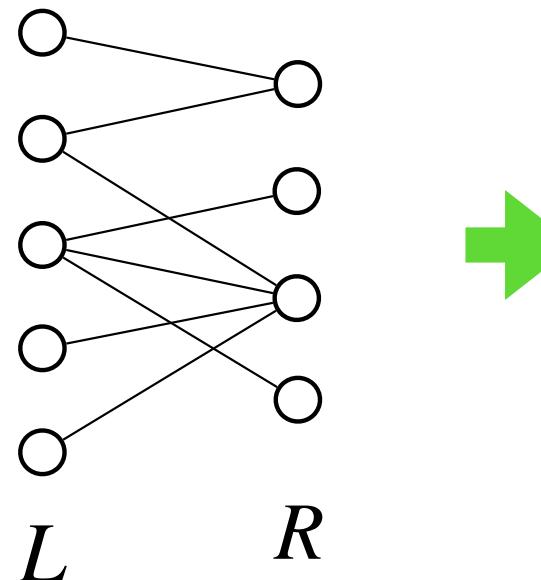


Bipartite Matching Problem:

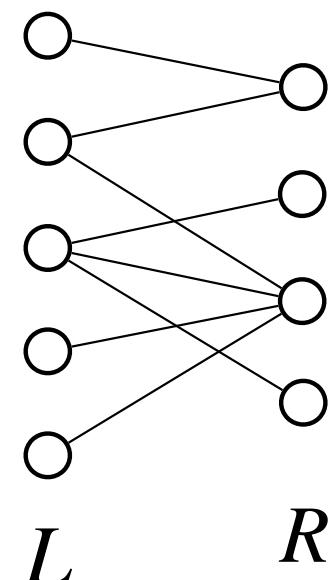
Input: A bipartite graph $G = (L \cup R, E)$.

Output: A matching of maximum size.

Example instance:



Maximum Flow Problem:

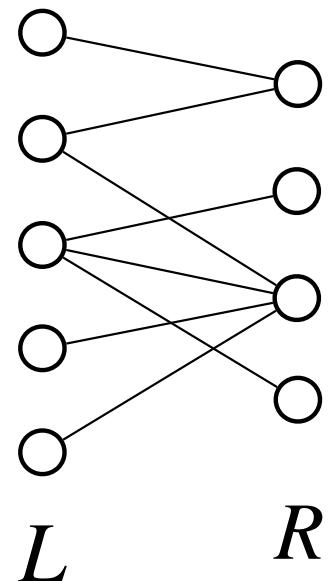


Bipartite Matching Problem:

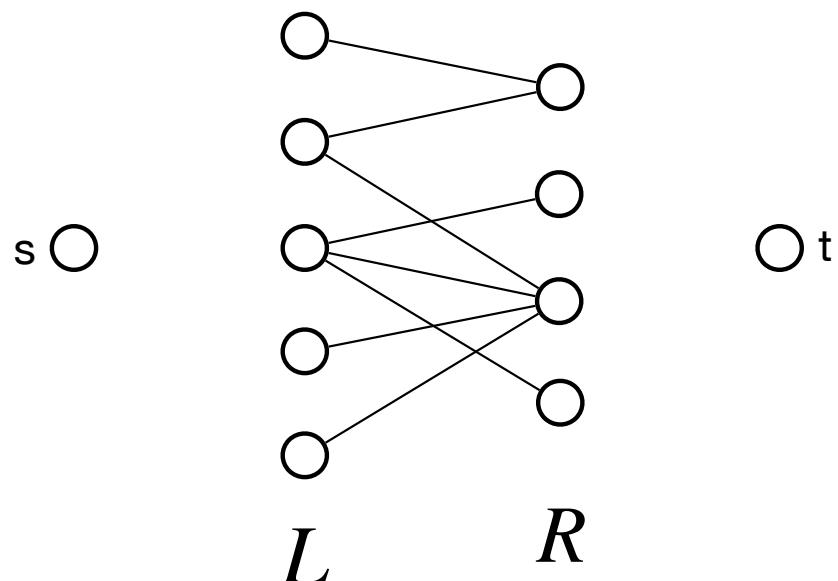
Input: A bipartite graph $G = (L \cup R, E)$.

Output: A matching of maximum size.

Example instance:



Maximum Flow Problem:

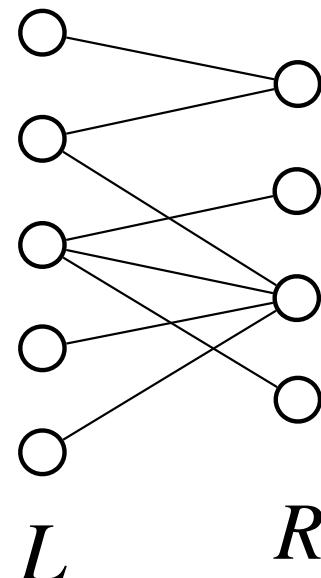


Bipartite Matching Problem:

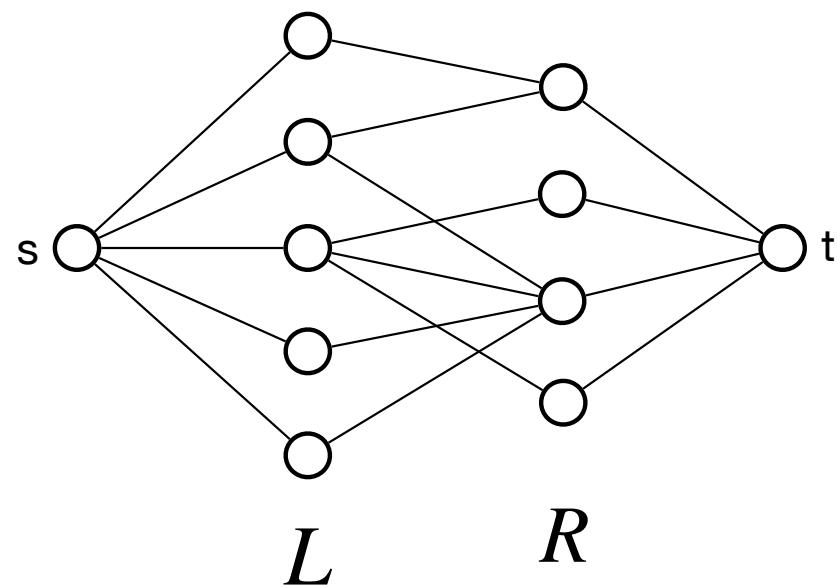
Input: A bipartite graph $G = (L \cup R, E)$.

Output: A matching of maximum size.

Example instance:



Maximum Flow Problem:

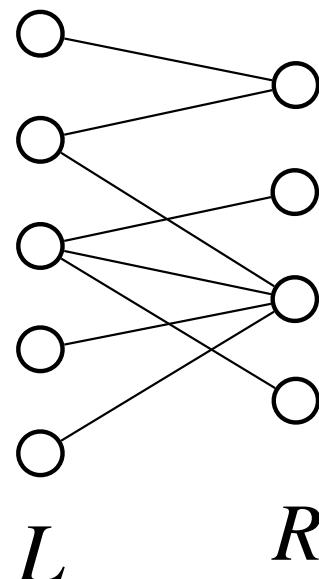


Bipartite Matching Problem:

Input: A bipartite graph $G = (L \cup R, E)$.

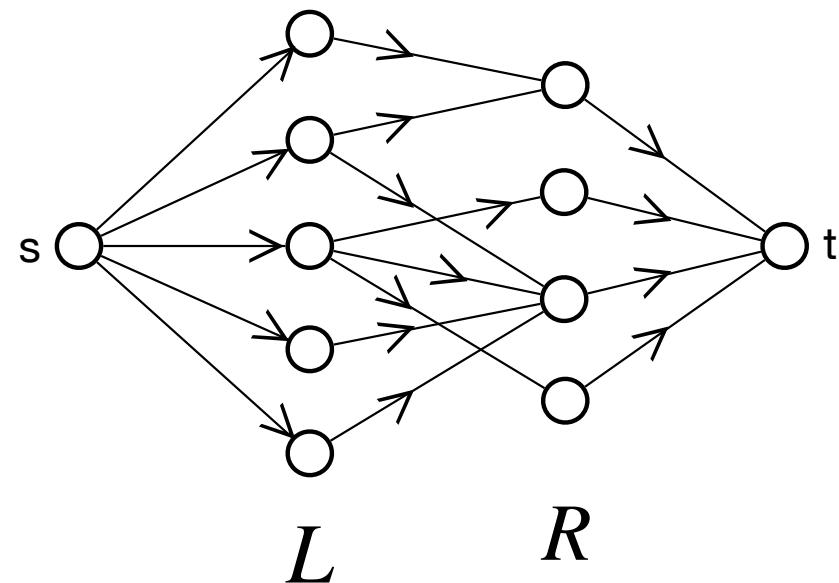
Output: A matching of maximum size.

Example instance:



Maximum Flow Problem:

All edges have capacity 1

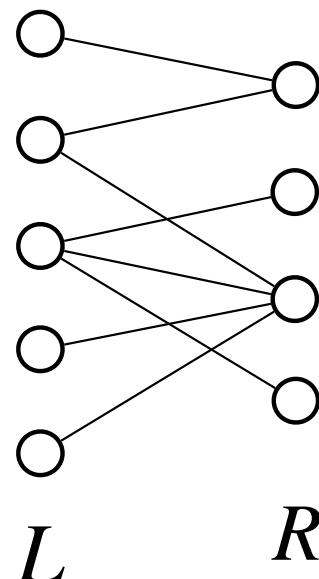


Bipartite Matching Problem:

Input: A bipartite graph $G = (L \cup R, E)$.

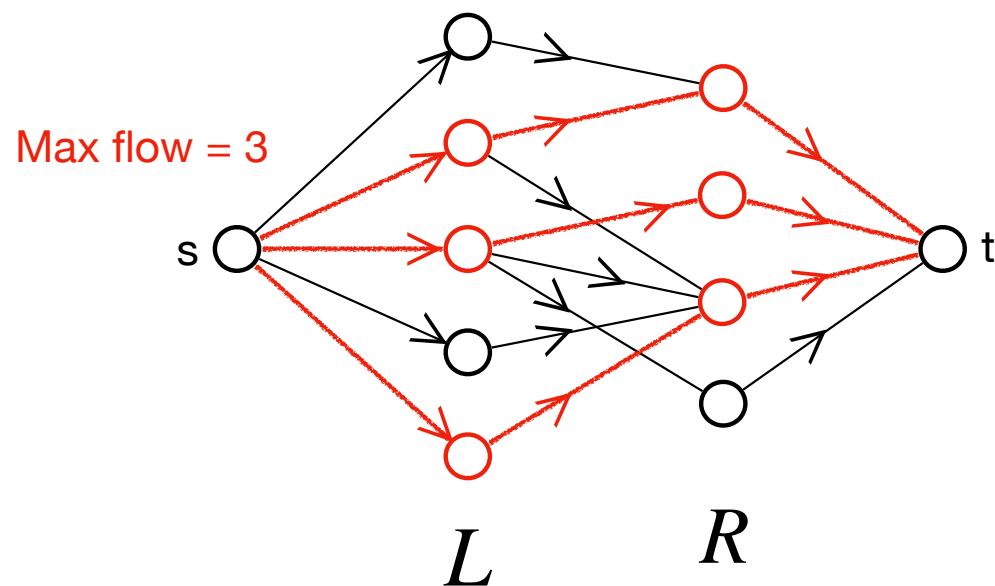
Output: A matching of maximum size.

Example instance:



Maximum Flow Problem:

All edges have capacity 1



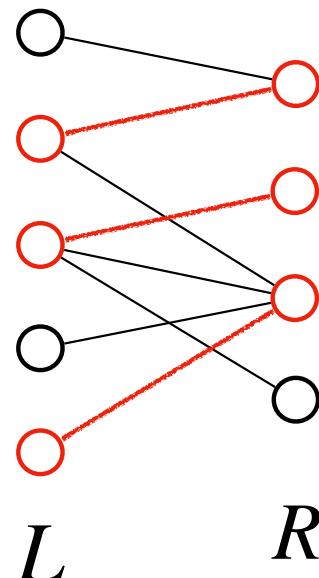
Bipartite Matching Problem:

Input: A bipartite graph $G = (L \cup R, E)$.

Output: A matching of maximum size.

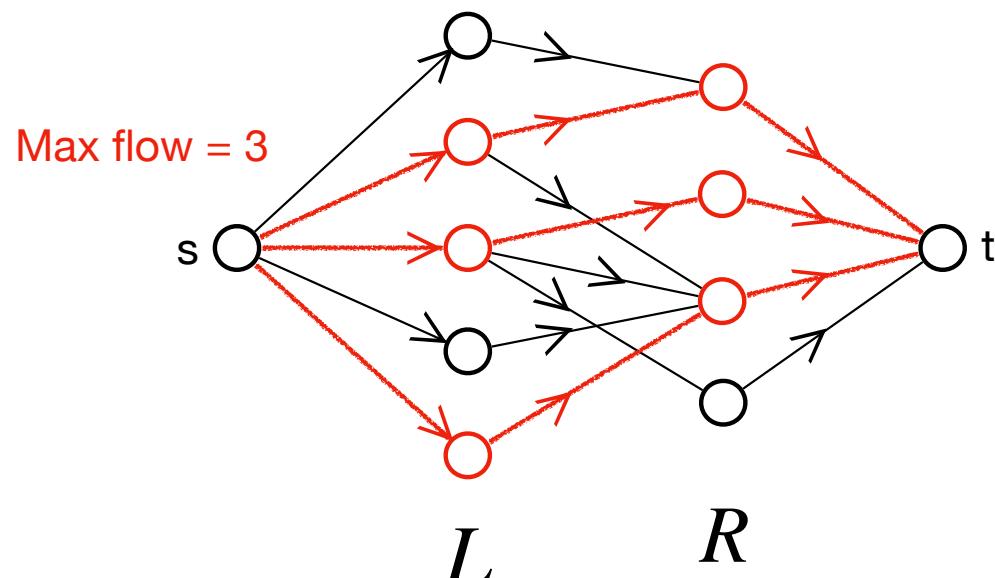
Example instance:

Max matching = 3



Maximum Flow Problem:

All edges have capacity 1



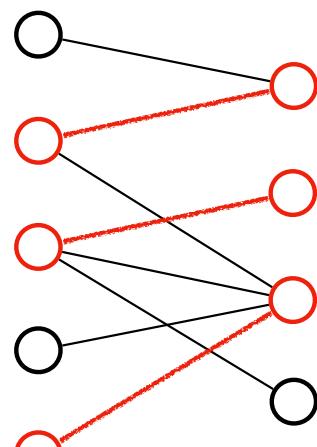
Bipartite Matching Problem:

Input: A bipartite graph $G = (L \cup R, E)$.

Output: A matching of maximum size.

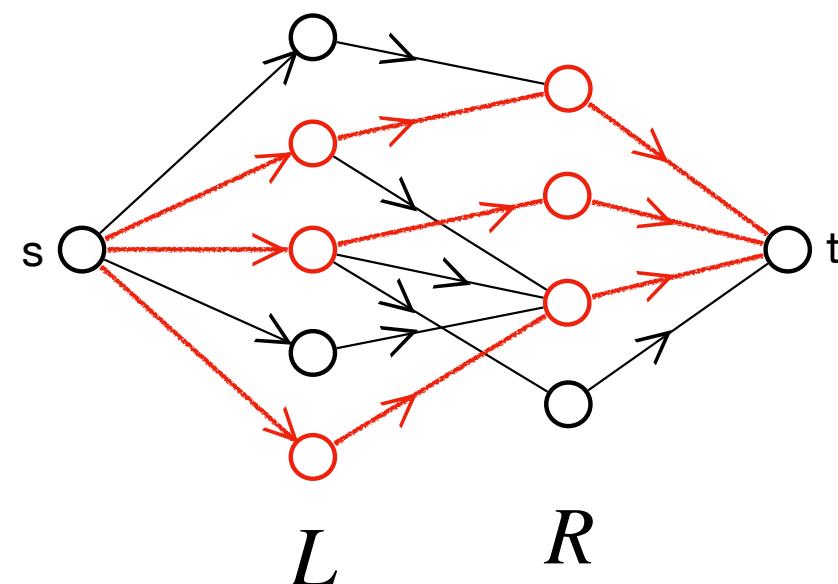
Theorem: G has a matching of size k
if and only if
 G' has a flow of size k .

$$G = (L \cup R, E)$$



$$G'$$

L R



Bipartite Matching Problem:

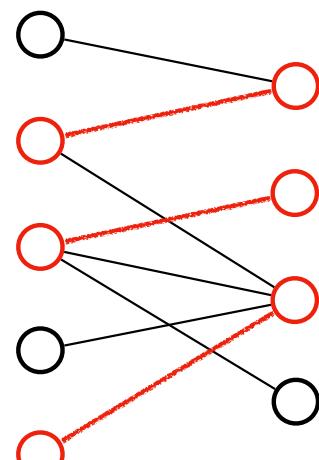
Input: A bipartite graph $G = (L \cup R, E)$.

Output: A matching of maximum size.

Theorem: G has a matching of size k
if and only if
 G' has a flow of size k .

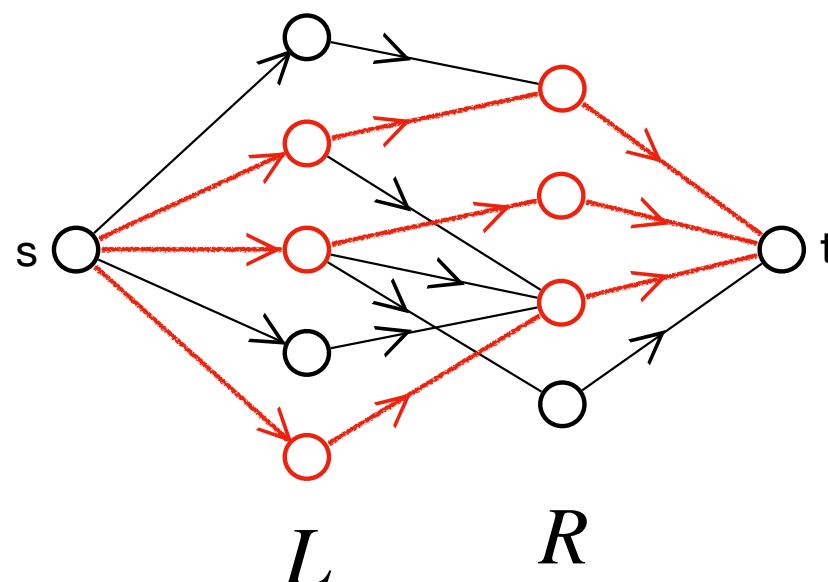
Proof:  Construct flow from matching.

$$G = (L \cup R, E)$$



L R

$$G'$$



Bipartite Matching Problem:

Input: A bipartite graph $G = (L \cup R, E)$.

Output: A matching of maximum size.

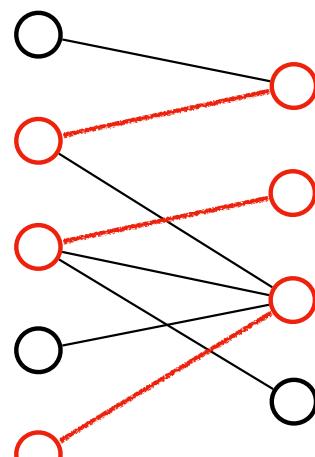
Theorem: G has a matching of size k
if and only if
 G' has a flow of size k .

Proof:

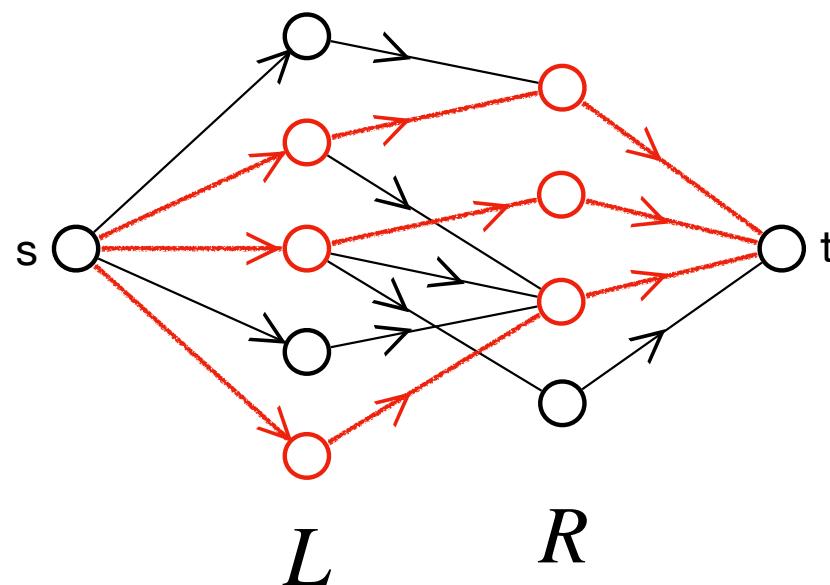
→ Construct flow from matching.

← G' has an integer flow of size k .

$$G = (L \cup R, E)$$



$$G'$$

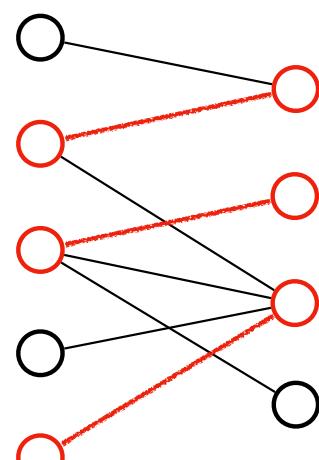


Bipartite Matching Problem:

Input: A bipartite graph $G = (L \cup R, E)$.

Output: A matching of maximum size.

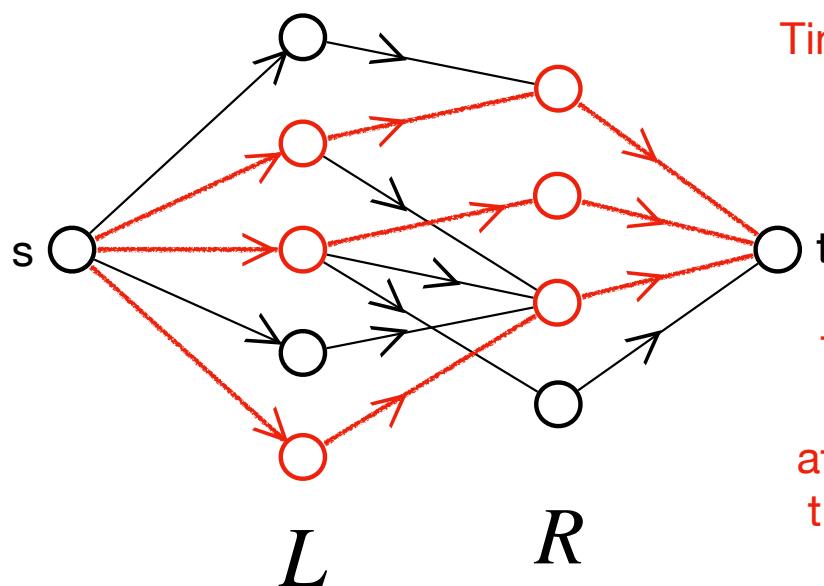
$$G = (L \cup R, E)$$



$$L \quad R$$



$$G'$$



Theorem: G has a matching of size k
if and only if
 G' has a flow of size k .

Proof:

→ Construct flow from matching.

← G' has an integer flow of size k .

Time complexity: $O(VE)$

$$|f| \leq |L| \leq |V|$$

The Edmonds-Karp algorithm needs to augment the flow at most $|V|$ times, and each time the BFS algorithm uses $O(V+E)$ time to find a shortest path.

Formulate Maximum Flow Problem as a Linear Program

Input: A directed graph $G = (V, E)$, where every edge $(u, v) \in E$ has a positive capacity $c(u, v)$.
Let $s \in V$ be a source node, and let $t \in V$ be a sink node.

Output: A maximum flow from s to t .

Formulate Maximum Flow Problem as a Linear Program

Input: A directed graph $G = (V, E)$, where every edge $(u, v) \in E$ has a positive capacity $c(u, v)$.

Let $s \in V$ be a source node, and let $t \in V$ be a sink node.

Output: A maximum flow from s to t .

Linear Program: 1) variables. 2) constraints. 3) objective function.

Formulate Maximum Flow Problem as a Linear Program

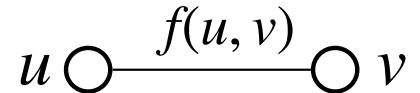
Input: A directed graph $G = (V, E)$, where every edge $(u, v) \in E$ has a positive capacity $c(u, v)$.

Let $s \in V$ be a source node, and let $t \in V$ be a sink node.

Output: A maximum flow from s to t .

Linear Program: 1) variables. 2) constraints. 3) objective function.

1) variables: \forall edge $(u, v) \in E$, the flow $f(u, v)$ on it is a variable.



Formulate Maximum Flow Problem as a Linear Program

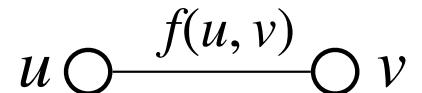
Input: A directed graph $G = (V, E)$, where every edge $(u, v) \in E$ has a positive capacity $c(u, v)$.

Let $s \in V$ be a source node, and let $t \in V$ be a sink node.

Output: A maximum flow from s to t .

Linear Program: 1) variables. 2) constraints. 3) objective function.

1) variables: \forall edge $(u, v) \in E$, the flow $f(u, v)$ on it is a variable.



Objective function and constraints:

$$\text{maximize} \quad \sum_{(s,v) \in E} f(s, v)$$

$$\text{subject to: } 0 \leq f(u, v) \leq c(u, v), \quad \forall (u, v) \in E$$

$$\sum_{(u,v) \in E} f(u, v) = \sum_{(v,u) \in E} f(v, u), \quad \forall v \in V - \{s, t\}$$

Formulate Maximum Flow Problem as a Linear Program

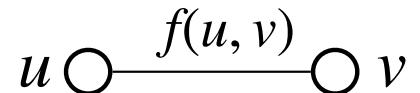
Input: A directed graph $G = (V, E)$, where every edge $(u, v) \in E$ has a positive capacity $c(u, v)$.

Let $s \in V$ be a source node, and let $t \in V$ be a sink node.

Output: A maximum flow from s to t .

Linear Program: 1) variables. 2) constraints. 3) objective function.

1) variables: \forall edge $(u, v) \in E$, the flow $f(u, v)$ on it is a variable.



Objective function and constraints:

$$\text{maximize} \quad \sum_{(s,v) \in E} f(s, v)$$

$$\text{subject to: } 0 \leq f(u, v) \leq c(u, v), \quad \forall (u, v) \in E$$

$$\sum_{(u,v) \in E} f(u, v) = \sum_{(v,u) \in E} f(v, u), \quad \forall v \in V - \{s, t\}$$

Why do we still need
Edmonds-Karp Algorithm?
Lower time complexity.