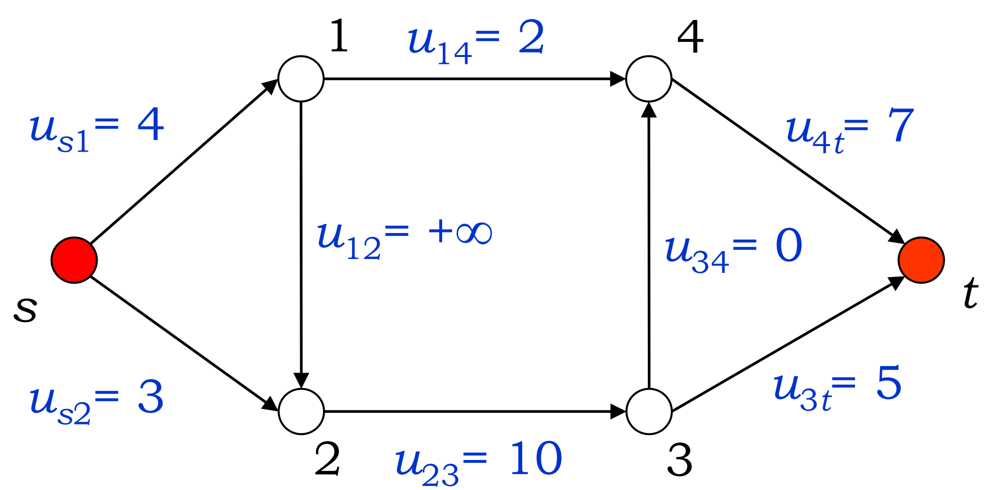
# Network Design (part 1)

This is an unofficial translation of the course material made by previous students

For any questions please contact the teacher

# Notation

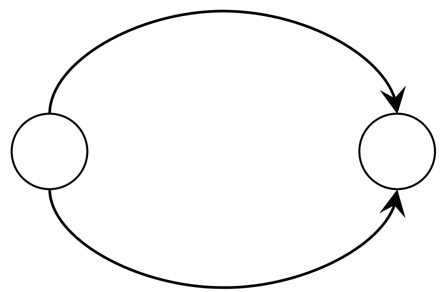
Directed graph $G = (N, A)$, with two “special” nodes : node s [the source] and node t [the sink]




$u_{ij} \in [0, +\infty)$, is the (integer) capacity of arc $(i, j)$

# Technical hypothesis

1. The graph does not contain a directed path from node *s* to node *t* composed only of infinite capacity arcs.
2. The graph is “simple”, i.e., does NOT contain "parallel“ arcs



3. Arcs with zero capacity can be added to the graph

# Problem

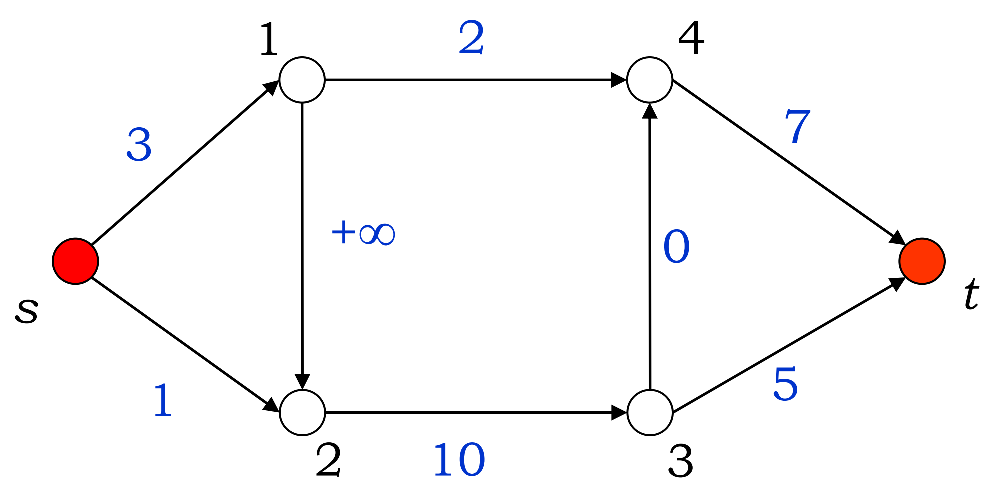
Path packing

Given a directed graph $G = (N, A)$ and a capacity vector $u \in Z^{|A|}$, find a family of directed paths (simple) $P = (P_1, P_2, \ldots, P_k)$, not necessarily distinct, such that:

1. Each arc $(i, j) \in A$ is an arc of at most $u_{ij}$ dipaths
2. $k$ is maximized

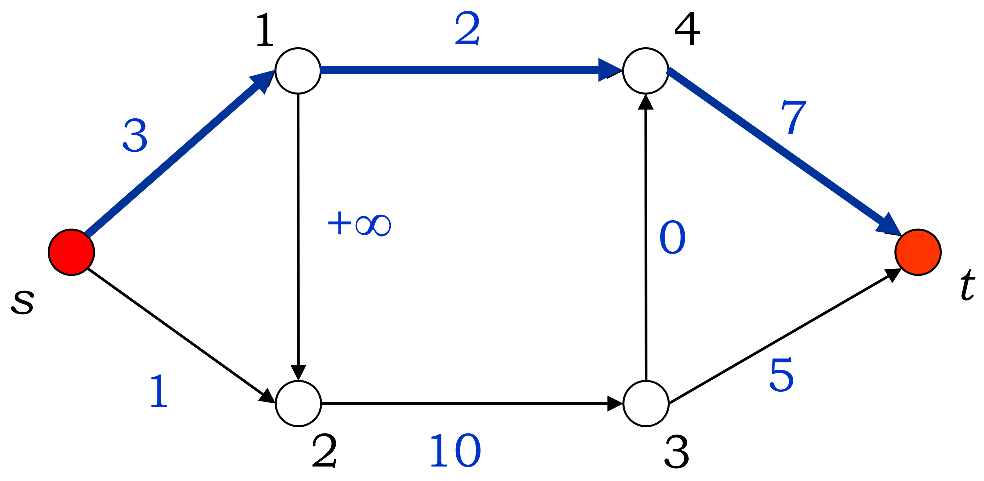
# Example

Consider the following graph
(numbers on the arcs represent capacities)



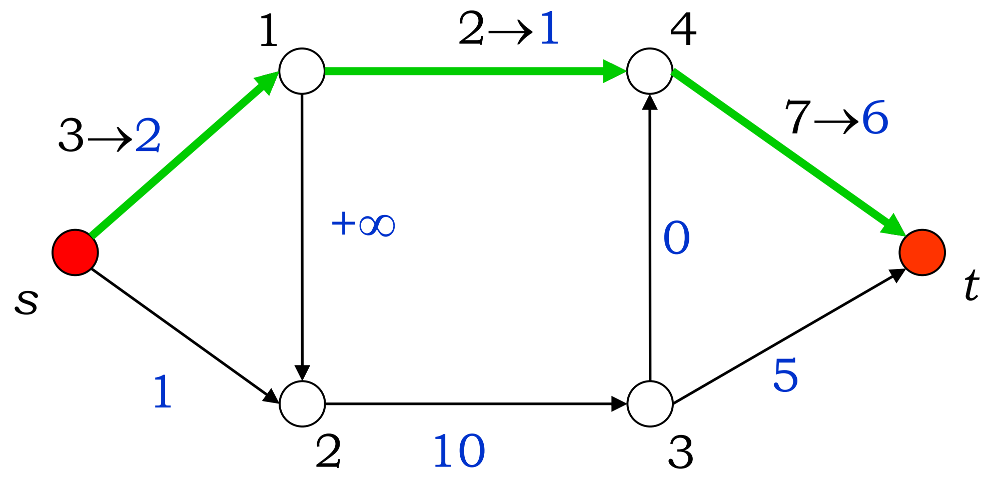
# Example

The first $(s,t)$ path is the path $P_1 = \{s, 1, 4, t\}$



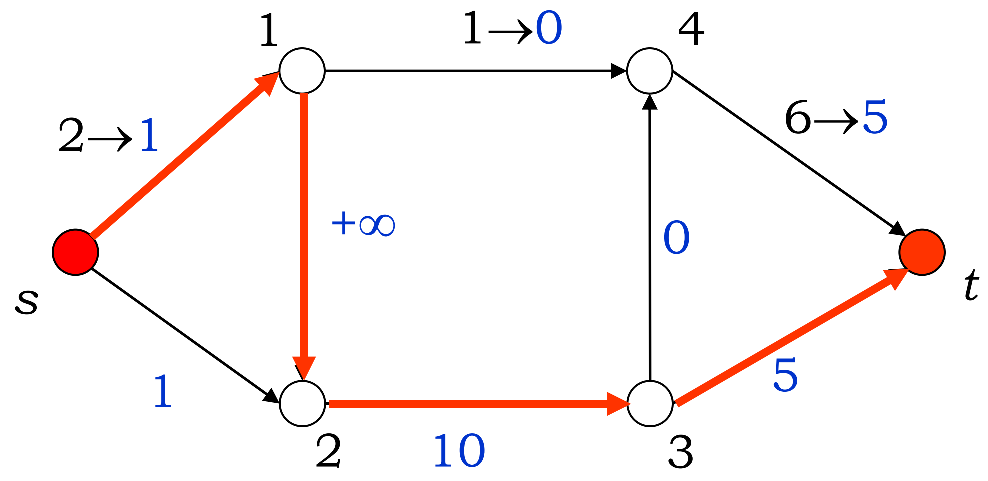
# Example

The second path uses the same arcs of $P_1$:
$P_2 = \{s, 1, 4, t\}$



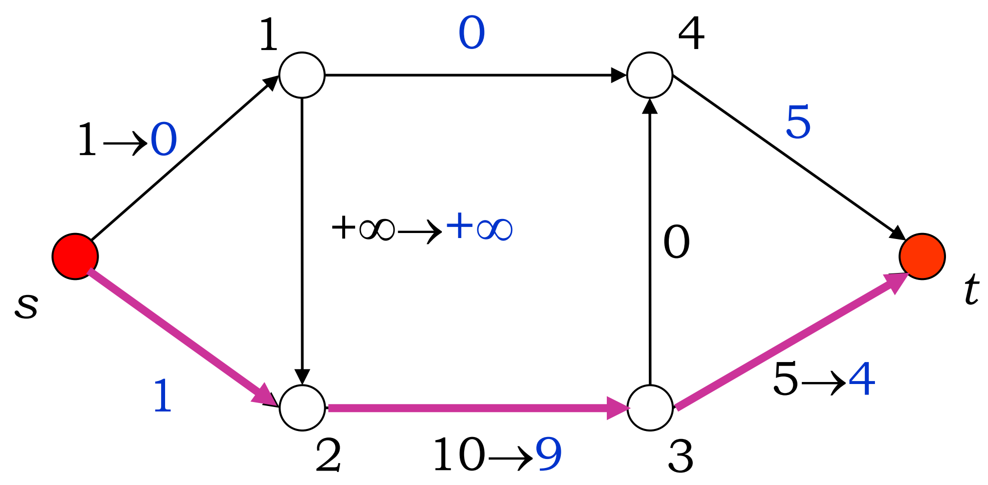
# Example

The residual capacity of the arc $(1, 4)$ is zero, so, we cannot choose a path that uses the same arcs of $P_1$ and $P_{2.}$
Another possible path is $P_3 = \{s, 1, 2, 3, t\}$



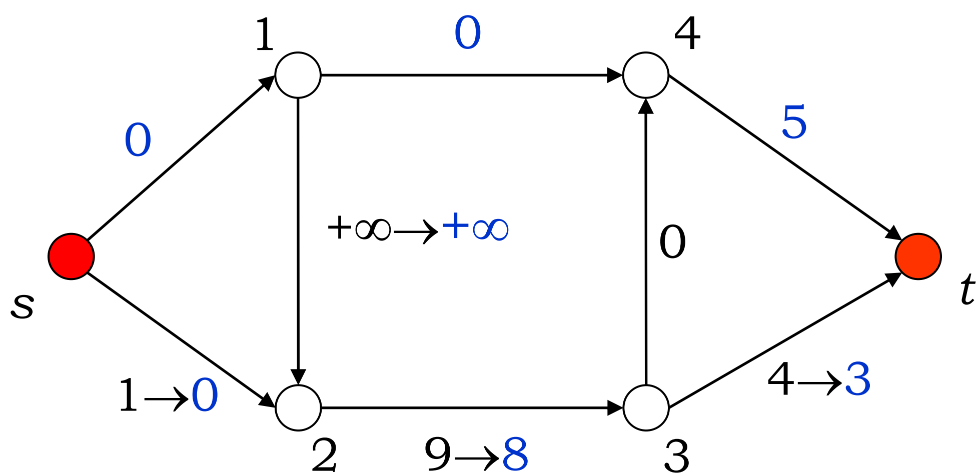
# Example

Now, the residual capacity of the arc $(s, 1)$ is also zero.
Therefore we choose the path $P_4 = \{s, 2, 3, t\}$ that uses arc $(s, 2)$



# Example

Now it is not possible to add a path from $s$ without violating the capacity constraints on the arc $(s,1)$ and $(s, 2)$.
The family $\{P_1, P_2, P_3, P_4\}$, with $k$ =4 is a feasible solution to the "path packing" problem.




$P_1 = \{s, 1, 4, t\}$
$P_2 = \{s, 1, 4, t\}$
$P_3 = \{s, 1, 2, 3, t\}$
$P_4 = \{s, 2, 3, t\}$

# Questions

1. It is possible to certify the optimality of the solution previously found?

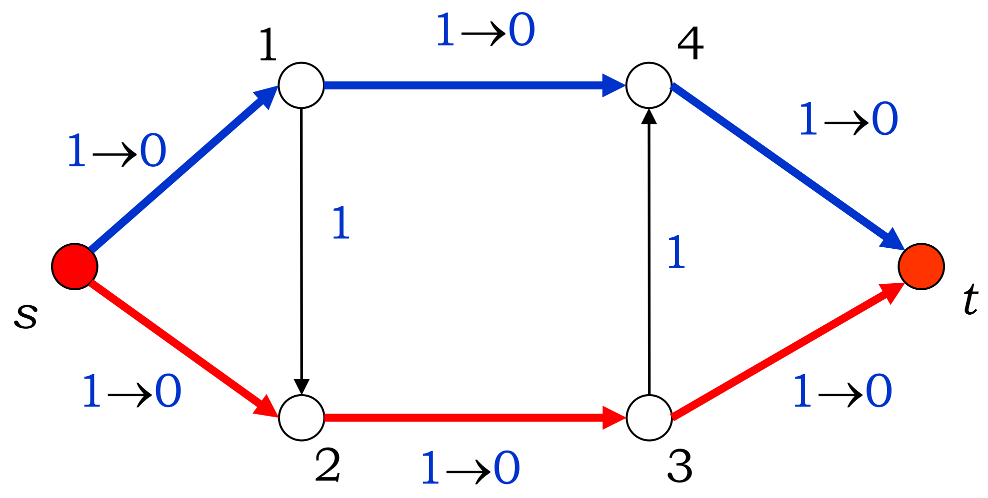
2. There exists an Integer Linear Programming formulation of the path packing problem?

3. There exists a polynomial time algorithm for the path packing problem?

# Answer to question 1

In general, it is not guaranteed that an optimal solution is found by choosing paths in a greedy way.

The following graph contains two (s, t) paths:

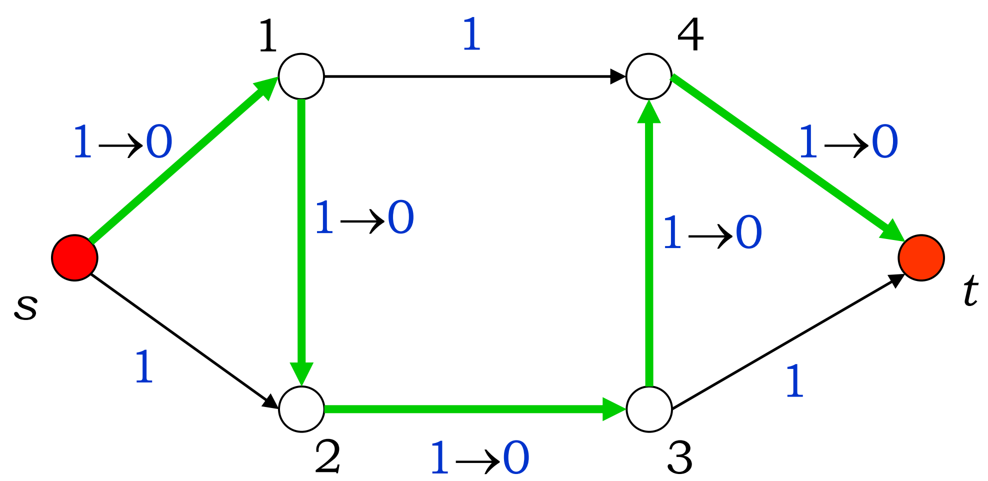


# Observation 1

but, if I choose as the first path the path

$P = \{s, 1, 2, 3, 4, t\}$

I cannot find more paths from $s$ to $t$ ….



# Question 2: formulation

Associate with each arc $(i, j)$ an INTEGER variabile $x_{ij}$ with the following meaning:

$x_{ij}$ = number of times that the arc $(i,j)$ is used by paths in $\mathcal{P}$

Constraints

Observation

For each node $v \neq s, t$ we have that every path $P_i$ enters $v$ and leaves from $v$ exactly the same number of times

# Formulation

## Flow balance constraints

$$\sum_{j:(i,j)\in A} x_{ij} - \sum_{j:(j,i)\in A} x_{ji} = 0 \qquad \forall i \in N \setminus \{s,t\}$$

## Capacity constraints

$$0 \le x_{ij} \le u_{ij}, \forall (i,j) \in A$$

## Integrality requirement

$$x_{ij} \text{ Integer}, \quad \forall (i,j) \in A$$

# Flow

For the source node $s$ we have, instead:

$$k = \sum_{j:(s,j)\in A} x_{sj} - \sum_{j:(j,s)\in A} x_{js}$$

A vector $x \in \mathcal{Z}_+^{|A|}$ that satisfies all the balance constraints is called $(s,t)$-flow , or simply flow.

A feasible flow is a flow $x$ that also satisfies the capacity constraints.

The term

$$f_x(v) = \sum_{j:(v,j)\in A} x_{vj} - \sum_{j:(j,v)\in A} x_{jv}$$

is called net flow in $v$.

$f_x(s)$ is the value of the flow $x$ in $G$

# Decomposition theorem

Given a family of paths $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$ is always possible to construct a feasible vector flow $x$.

The following result shows that also the reverse is true.

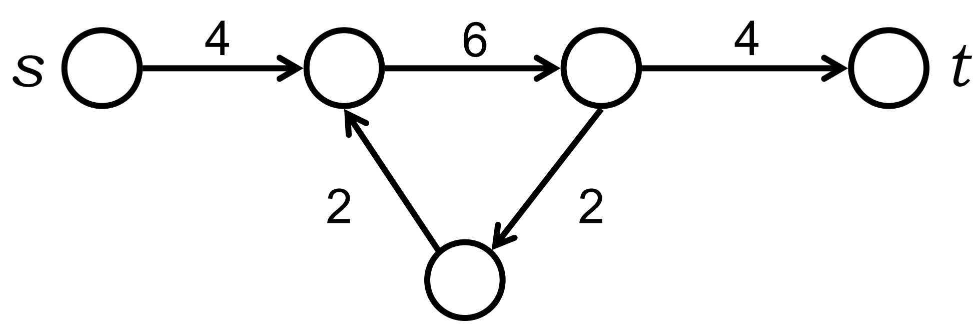
Decomposition theorem

Given a graph $G=(N,A)$, there exists a family $\mathcal{P} = \{P_1, P_2, \ldots, P_k\}$ of $k$ $(s,t)$-paths such that each arc of the graph is used at most $u_{ij}$ times by the paths in $\mathcal{P}$, if and only if there exists an integral feasible $(s,t)$-flow of value $k$

# Proof

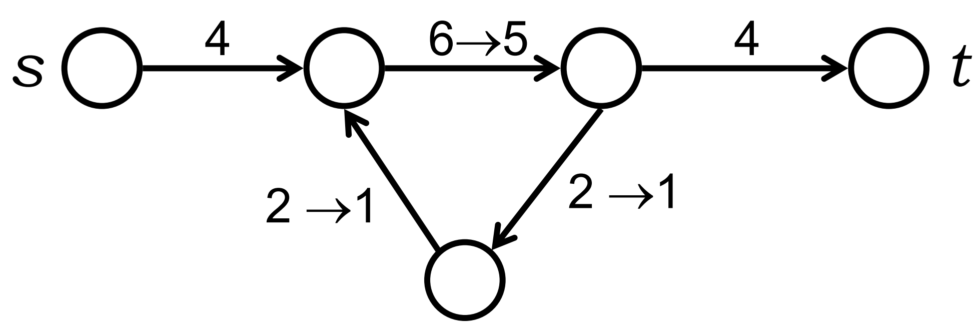
Let be an "acyclic"flow, i.e. a flow that does not contain an oriented cycle $C$ with $x_{ij} > 0$ for all arcs $(i,j) \in C$.

Note that, if $x$ contains an oriented cycle $C$ with this property, an acyclic flow can always be obtained from $x$ just by decreasing $x_{ij}$ for all arcs $(i,j) \in C$.



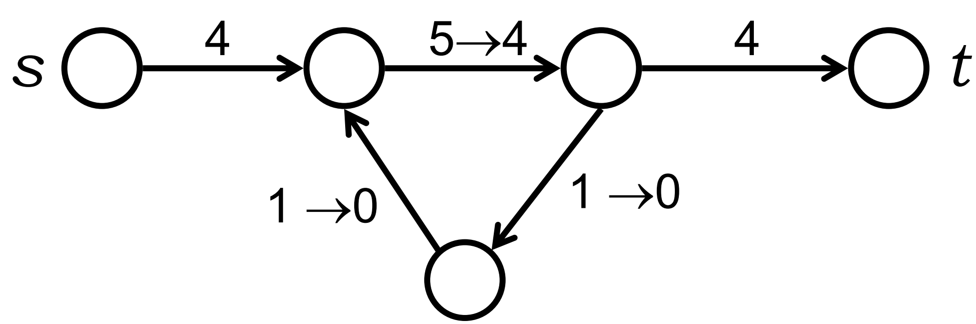
# Proof

In particular, $x_{ij}$ is decreased by one unit for all arcs $(i,j) \in C$ until the cycle $C$ disappears.



# Proof

Note that this simple procedure does not alter the value $k$ of the flow from $s$



# Proof

Now, if $k \geq 1$ then there is an arc $(v, t)$ with $x_{vt} \geq 1$.

If $v \neq s$, the balance constraint implies that there exists at least one arc with $x_{wv} \geq 1$.

If $w \neq s$, the argument can be repeated returning an arc $(p, w)$ with $x_{pw} \geq 1$. Since $x$ is acyclic, this procedure produces distinct nodes, until we eventually get node $s$ and $t$. In such a case we found an $(s,t)$ simple path made of arcs $(i, j)$ with $x_{ij} \geq 1$.

It is therefore sufficient to decrease by one unit each component of the vector $x$ corresponding to the arcs in the $(s,t)$ path to obtain a new feasible (and integer) flow of value $k$-1.

By repeating this procedure until $k = 0$, one obtains the $k$ paths associated with $\mathcal{P}$ ■

# ILP formulation

$$\max \quad f_x(s)$$

s.t.

$$\sum_{j:(i,j)\in A} x_{ij} - \sum_{j:(j,i)\in A} x_{ji} = 0 \qquad \forall i \in N \setminus \{s,t\}$$

$$0 \le x_{ij} \le u_{ij}, \forall (i,j) \in A$$

$$x_{ij} \quad \cancel{\text{integral}}, \quad \forall (i,j) \in A$$

# Cut of a graph

Definition

Given a graph $G=(N, A)$, a cut is a set $\delta(R)=\{vw$: $(v,w) \in A,\ v\in R,\ w \notin R\}$ for some $R \subseteq V$

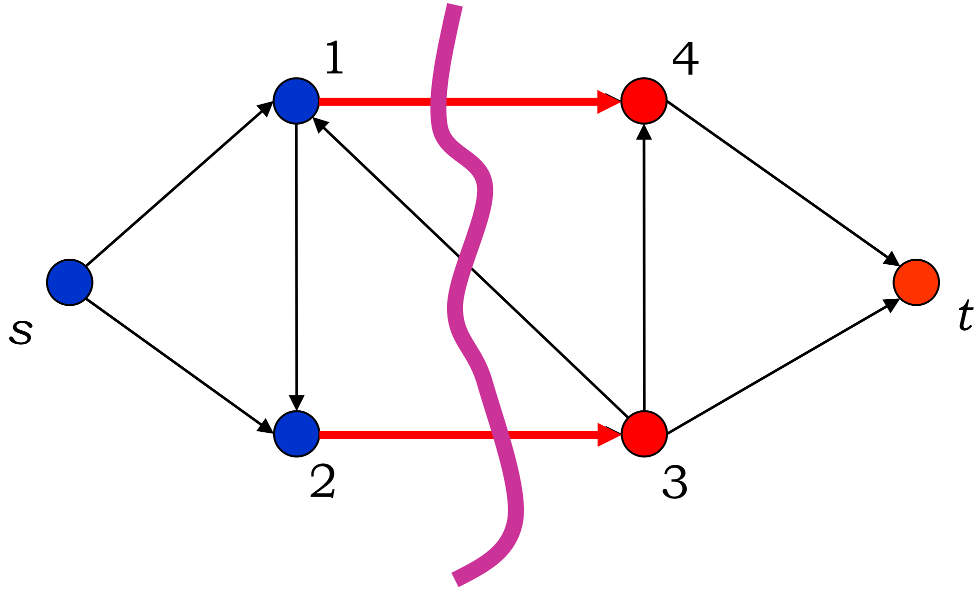
An $(s,t)$-cut is a cut such that $s\in R,\ t \notin R$

The capacity of an $(s,t)$-cut is the quantity

$\sum_{i\in R,\ j\notin R} u_{ij} = u\,(\delta(R))$

# Example

$R = \{s, 1, 2\}$
$\bar{R} = \{3, 4, t\}$

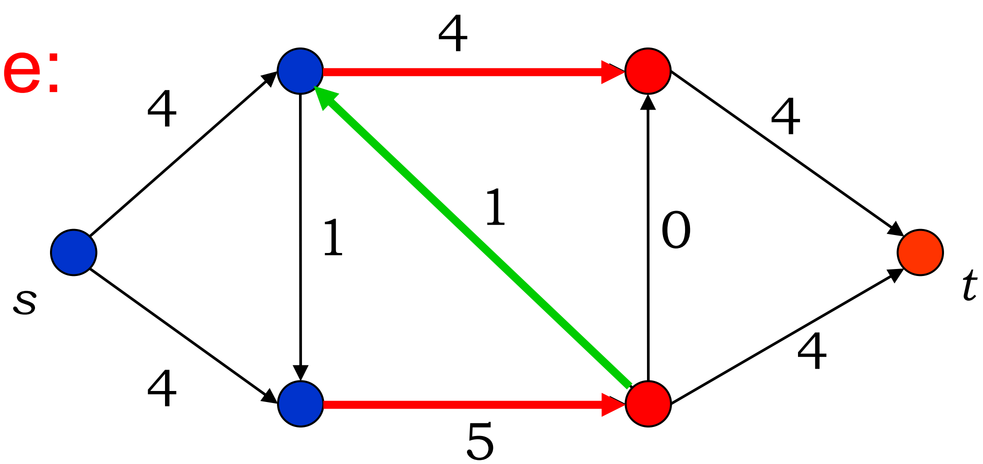


# Theorem 1

For any $(s,t)$-cut $\delta(R)$ and any feasible $(s,t)$-flow $x$, we have

$$x(\delta(R)) - x(\delta(\overline{R})) = f_x(s) \qquad \blacksquare$$

Example:



# Proof

Let $R$ be an (s,t) cut and, for all nodes $v \in R$, $v \neq s$, add all balance constraints.

The resulting equation has the form

LHS = $0$

The term LHS has the following form:

1. For any arc $(v, w)$ such that $v, w \in R$, $v \neq s$ the variable $x_{vw}$ is NOT contained in the LHS (coefficient equal to zero in the sum).
2. For any arc $(v, w)$ such that $v, w \notin R$, the variable $x_{vw}$ is NOT contained in LHS (arc extremes do not belong to $R$).
3. For any arc $(v, w)$ such that $v \in R$, $w \notin R$ the variable $x_{vw}$ appears in the LHS with coefficient +1

# Proof

4. For any arc $(v, w)$ such that $v \notin R$, $w \in R$ the variable $x_{vw}$ appears in the LHS with coefficient -1.
5. For any arc $(s, v)$ such that $v \in R$ the variable $x_{sv}$ appears in the LHS variable with coefficient -1
6. For any arc $(v, s)$ such that $v \in R$ the variable $x_{vs}$ the variable appears in the LHS with coefficient 1.

By grouping the variables that satisfy conditions 3 and 4 one obtains:

$$x(\delta(R)) - x(\delta(\overline{R}))$$

The variables that satisfy the conditions 5 and 6 sum up to $-f_x(s)$.

Therefore,

$$\text{LHS} = x(\delta(R)) - x(\delta(\overline{R})) - f_x(s)$$

■

# Corollary (weak duality)

For any $(s,t)$-cut $\delta(R)$ and for any $(s,t)$-flow $x$, we have:

$$f_x(s) \leq u(\delta(R))$$

Proof

From Theorem 1 we have that:

$$x(\delta(R)) - x(\delta(\bar{R})) = f_x(s)$$

Now, by definition $x(\delta(R)) \leq u(\delta(R))$. Moreover, $x(\delta(\bar{R})) \geq 0$

Therefore, $f_x(s) \leq u(\delta(R))$.

■

# Consequence

The weak duality provides a bound for the value of the maximum flow.

Therefore, if we identify a flow $x$ in $G$ with value equal to the capacity $u$ of a cut $R$, we proved that $x$ is the optimal solution to maximum flow problem.

The Max-Flow Min-Cut theorem says that this possibility occurs for all graphs $G$ that admits a finite maximum flow.

# Max-flow Min-cut theorem

If $G=(N,A)$ admits an $(s,t)$ maximum flow, then

$$\max\left\{f_x(s) : x \text{ is a feasible } (s,t)\text{-flow}\right\} =$$
$$= \min\left\{u(\delta(R)) : \delta(R) \text{ is an } (s,t)\text{-cut}\right\}$$

■

[Ford e Fulkerson, Kotzig 1956]

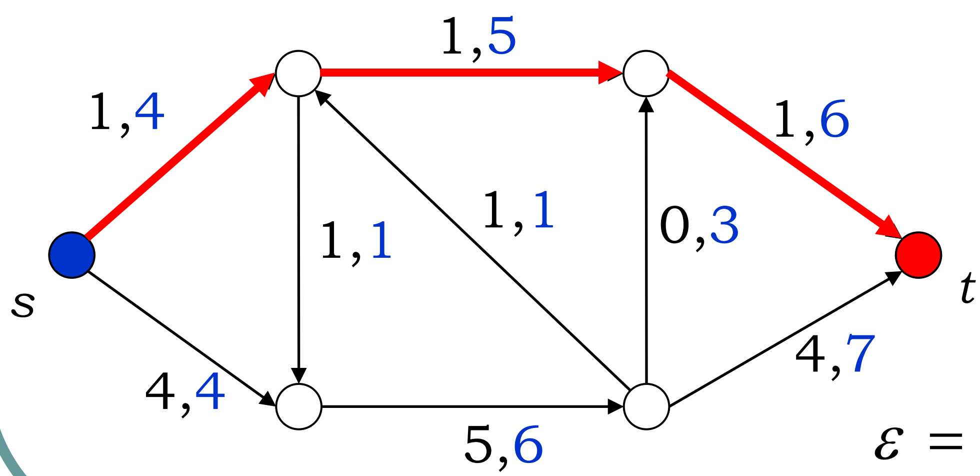
To prove this theorem we introduce the concept of augmenting path

# Augmenting Path

Observation

Given a graph $G = (N, A)$ and a flow $x$, if there is an $(s, t)$-path $P$ such that $x_{ij} < u_{ij}$ for all arcs $(i, j) \in P$, then the flow can be increased by the following value

$$\varepsilon = \min \left\{ u_{ij} - x_{ij}, (i, j) \in P \right\}$$

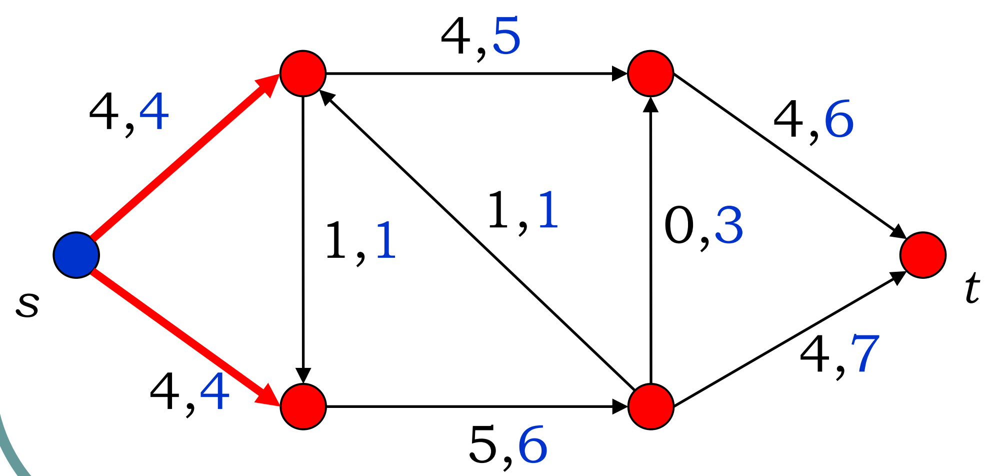



$$\varepsilon = \min \left\{ 4 - 1, 5 - 1, 6 - 1 \right\} = 3$$

# Augmenting Path

## Observation (cont.)

The new flow is optimal. In fact, there is an $(s,t)$-cut (identified by the blue and red nodes) of capacity 8, equal to the value of the maximum flow.



# A possible algorithm

**inizialization:** $x = 0$;

**do** {

**search in** $G$ **an** $(s, t)$**-path** $P$ **such that** $x_{ij} < u_{ij}$

**for each arc** $(i, j) \in P$ ;

**increase the flow** $x$ **along** $P$ **by the value**

$\varepsilon = \min \left\{ u_{ij} - x_{ij}, (i, j) \in P \right\}$

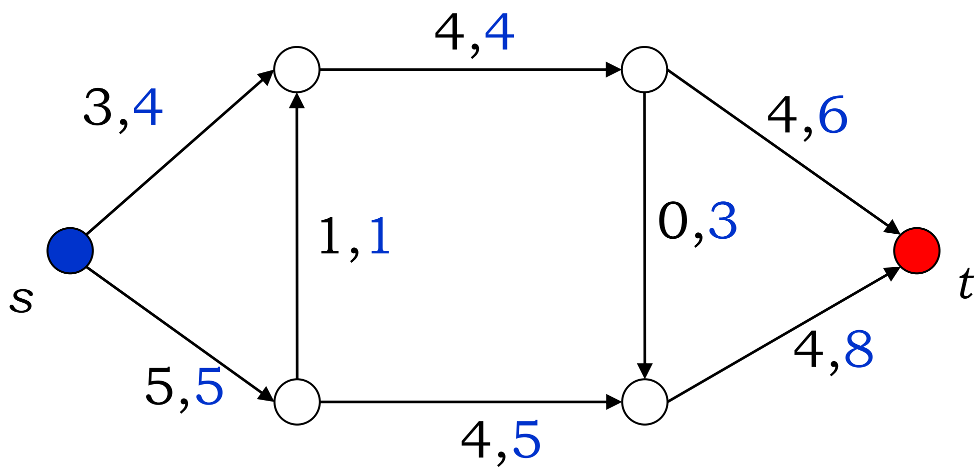
} **while** $(P \neq \varnothing)$;

The following questions arise:

1. Does the algorithm terminate correctly?
2. Does the algorithm return the optimal solution?
3. What is its complexity?

# Augmenting Path

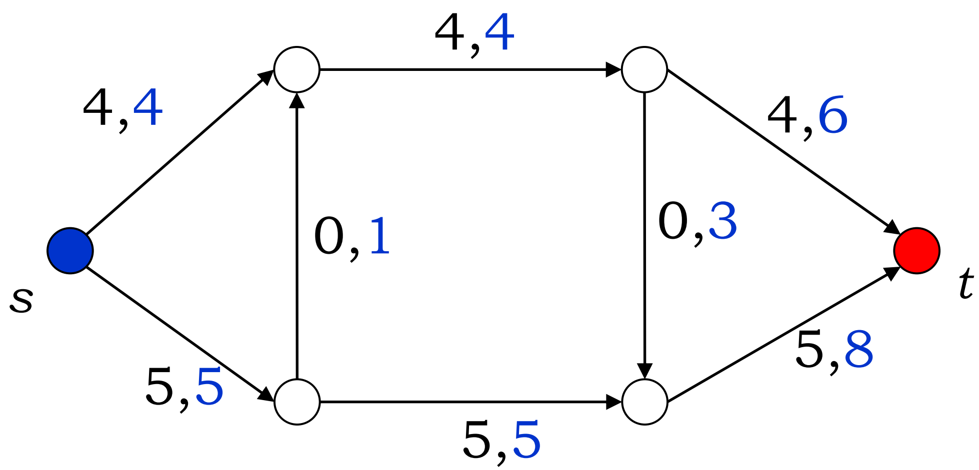
Consider the graph:




There is no $(s, t)$ path $P$ such that $x_{ij} < u_{ij}$ for all arc $(i, j) \in P$, but the flow is not optimal.

# Augmenting Path

This flow is optimal! [why?]




How can I get it from the previous flow?

# Augmenting Path

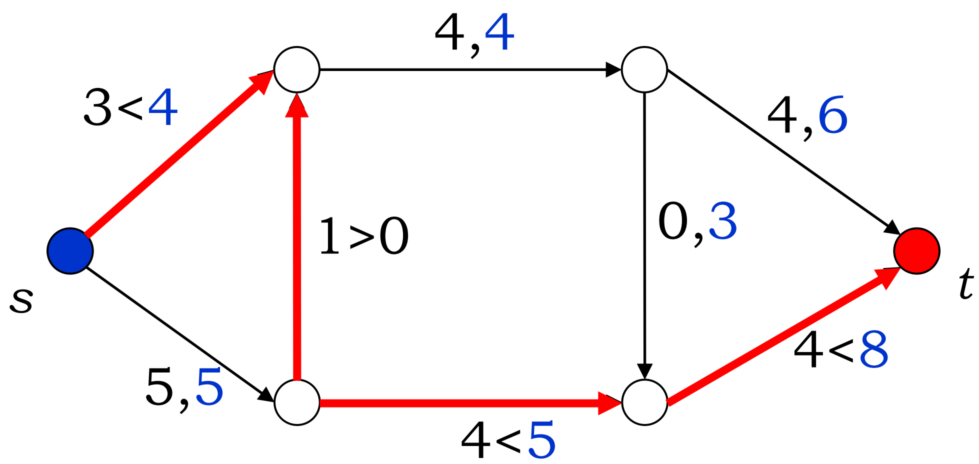
Let $P$ be a path (NOT well oriented) from $s$ to $t$. An arc $P$ is called forward if it has direction $s \to t$ and reverse viceversa.

Definition

An $(s, t)$-path $P$ such that every forward arc $(i, j)$ has $x_{ij} < u_{ij}$ and every reverse arc $(i, j)$ has $x_{ji} > 0$ is said to be an augmenting path.

# Augmenting Path

The path in the figure is an augmenting path:



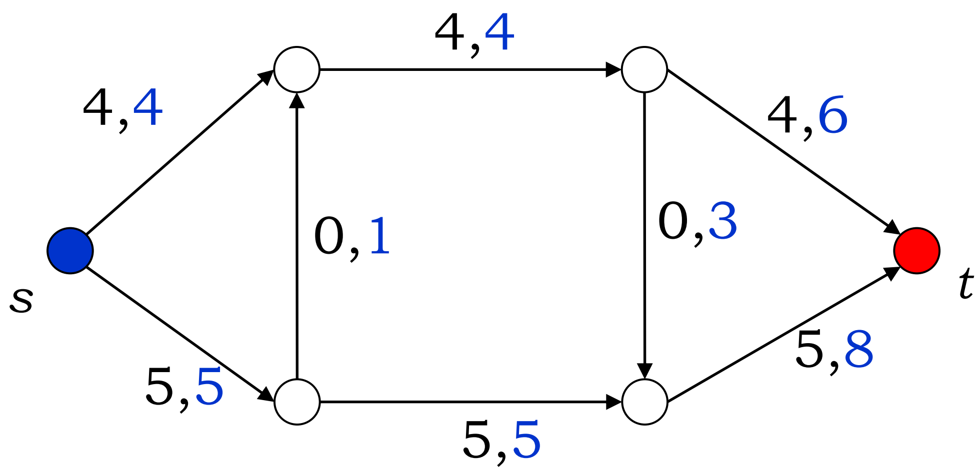

On this path we can increase the flow by the quantity $\min\{\varepsilon_1, \varepsilon_2\}$ where

$\varepsilon_1=\min\{u_{ij}-x_{ij}: (i,j)\in P$ e $(i,j)$ is a forward arc$\}$

$\varepsilon_2=\min\{x_{ij}: (i,j)\in P$ e $(i,j)$ is a reverse arc$\}$

# Augmenting Path

By increasing the flow along $P$ we obtain:




Definition

An $(s, v)$-path $P$ such that $v \neq t$, $x_{ij} < u_{ij}$ for every forward arc $(i, j)$ and $x_{ji} > 0$ for every reverse arc $(i, j)$, is called incrementing path.

# Max-Flow Min-Cut theorem: proof

From weak duality we know that it is sufficient to show that there exists in $G$ a flow $x$ and a cut $\delta(R)$ such that $f_x(s) = u(\delta(R))$.

Let $x$ be a flow with maximum value. Build the cut $\delta(R)$ by defining $R$ as follows:

$R = \{v \in N$: there exists a incrementing path $(s, v)$ $\}$.

By definition, $t \notin R$. In fact, if $t$ belongs to $R$ the path would be augmenting, contradicting the maximality of $x$.

$x_{ij} = u_{ij}$ for each arc $(i, j) \in \delta(R)$. In fact, if $x_{ij} < u_{ij}$, then $j \in R$. Hence, $x(\delta(R)) = u(\delta(R))$.

$x_{ij} = 0$ for each arc $(i, j) \in \delta(R)$. In fact, if $x_{ij} > 0$ then $j \in R$. Hence, $x(\delta(R)) = 0$

Therefore, from Theorem 1 we have:

$$f_x(s) = x(\delta(R)) - x(\delta(\overline{R})) = u(\delta(R))$$

■

# Consequences of the MFMC theorem

**Theorem 2**

A feasible flow $x$ is optimal if and only if there is no augmenting path in $G$.

**Proof**

$x$ maximum $\Rightarrow$ there is no augmenting path (trivial)

there is no augmenting path $\Rightarrow$ $x$ maximum

If there is no augmenting path, then by using the construction of the MFMC theorem, we can build a cut $\delta(R)$ with the property $f_x(s) = u(\delta(R))$.

From the weak duality property it follows that $x$ is maximum. ■

**Corollary 1**

If $x$ is an $(s, t)$ feasible flow and $\delta(R)$ is a $(s,t)$-cut, then $x$ is maximum and $\delta(R)$ is minimum if only if $x_{ij}=u_{ij}$ for all $(i, j) \in \delta(R)$ e $x_{ij}=0$ for all $(i, j) \in \delta(R)^-$ ■

# Augmenting Path algorithm

**inizialization:** $x = 0$;

**do** {
  **search in** $G$ **an** $(s, t)$**-path augmenting** $P$;
  **increases along** $P$ **the flow of value** $x \min \{\varepsilon_1, \varepsilon_2\}$ ;

} **while** $(P \neq \emptyset)$;  [Ford and Fulkerson's algorithm]

This algorithm

1. Terminates?
2. Is the solution optimal?

 YES: if the algorithm terminates, theorem 2 we know that the flow is excellent. è optimal.

3. What is its complexity?

# Data structure for augmenting paths

To check the paths we need an appropriate data structure.

Starting from $G$, define an auxiliary graph $G(x)$ with the following characteristics:

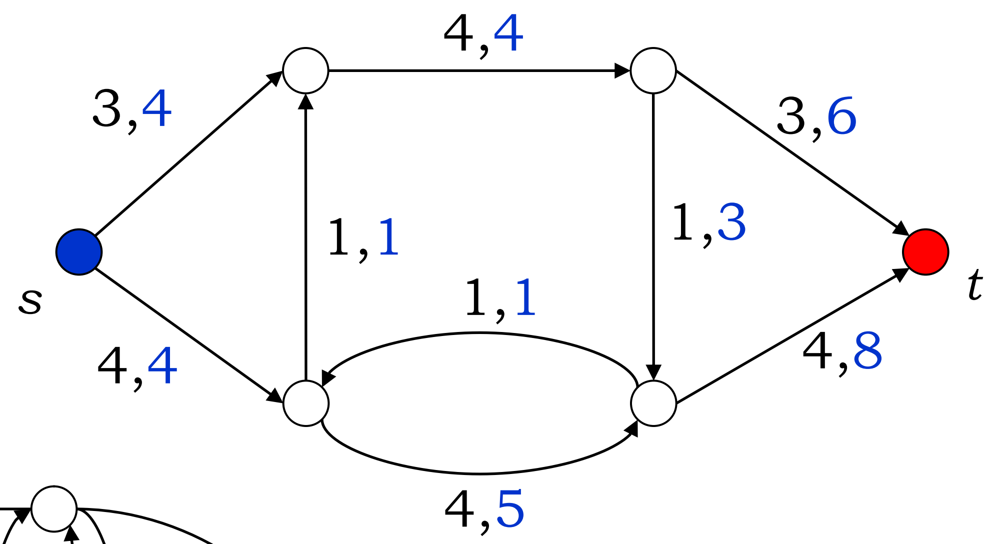
- $N(G(x)) = N$
- The arc $(i,j)$ belongs to $A(G(x))$ if and only if the arc $(i,j)$ belongs to $A$ and $x_{ij} < u_{ij}$ or $(j, i)$ belongs to $A$ and $x_{ji} > 0$.

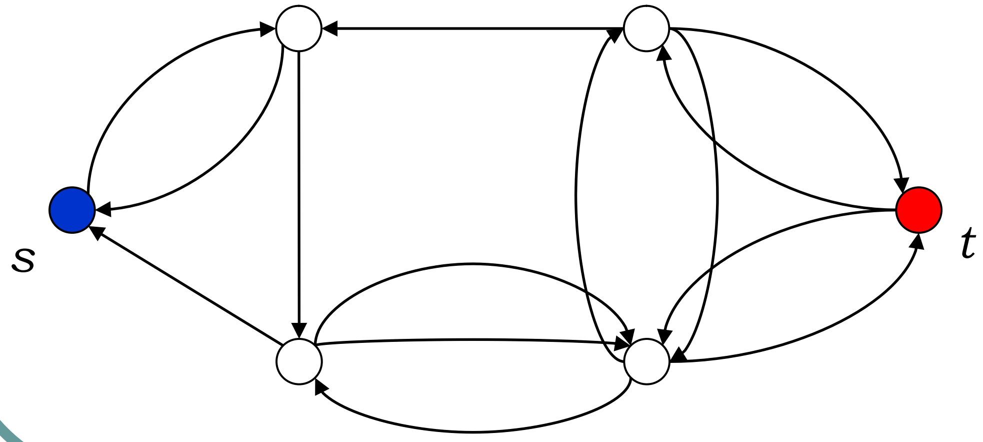
Observation

The graph $G(x)$ is not a simple graph.

# Data structure for augmenting paths

## Example




$G(x)$

# Termination and Complexity

An $(s,t)$-path in $G(x)$ corresponds to an augmenting path in $G$. Therefore, an augmenting path in $G$ can be determined in $O(m)$, by "JUST" visiting $G(x)$.

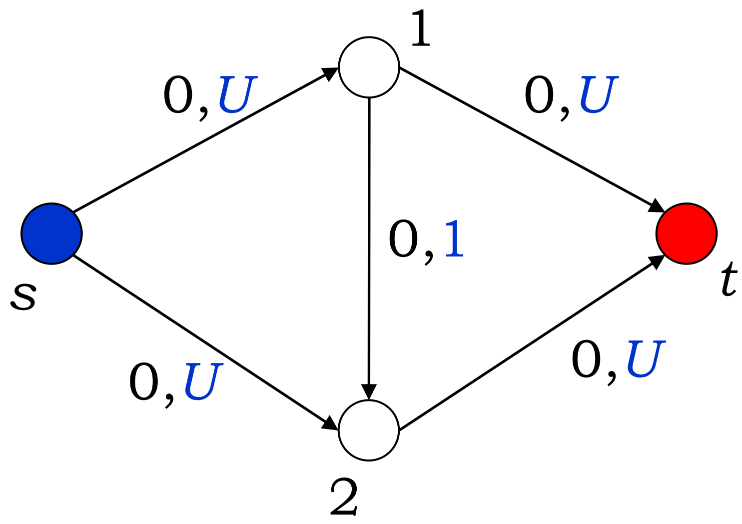
If $u$ is integer and the graph admits a finite flow, then a bound on the number of iterations of the algorithm is given by $k$, where $k$ is the value of maximum flow.

A trivial bound for k is obtained by considering a cut with $R = (s)$. Denoting by $U$ the maximum capacity of the arcs, $u(\delta(R)) \leq nU$..
Therefore, the algorithm based on the augmenting paths terminates in at most $O(nmU)$ iterations if $G$ admits a flow different from $\infty$.

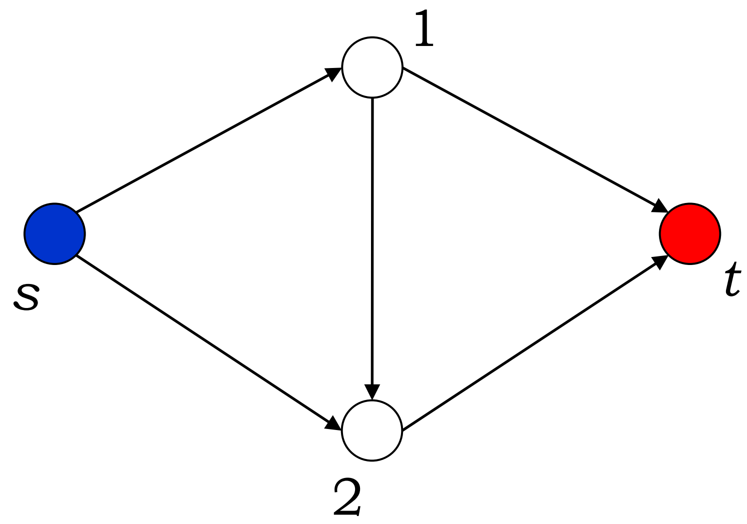
Even if $u$ is rational, one can construct an appropriate problem "scale" and prove the convergence in a finite number of steps.

# Example

The practical complexity depends on the choice of augmenting paths. Consider the following graph:



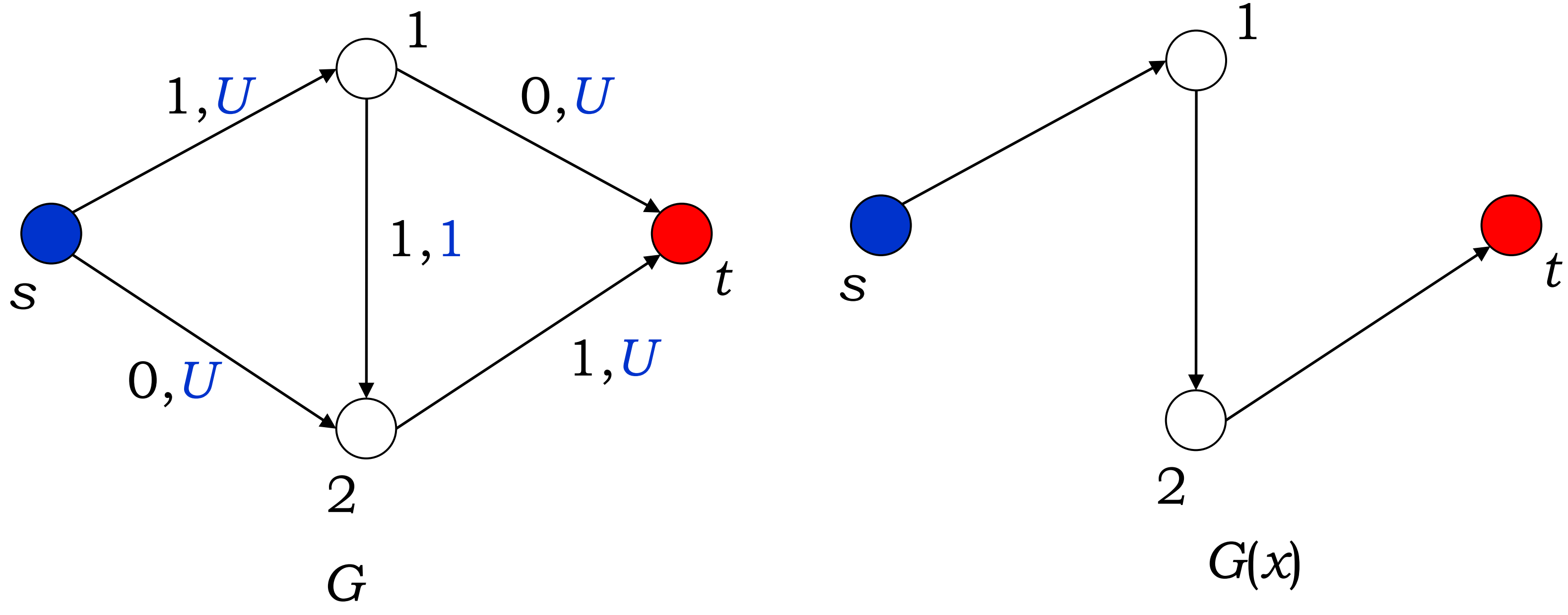

$G$




$G(x)$

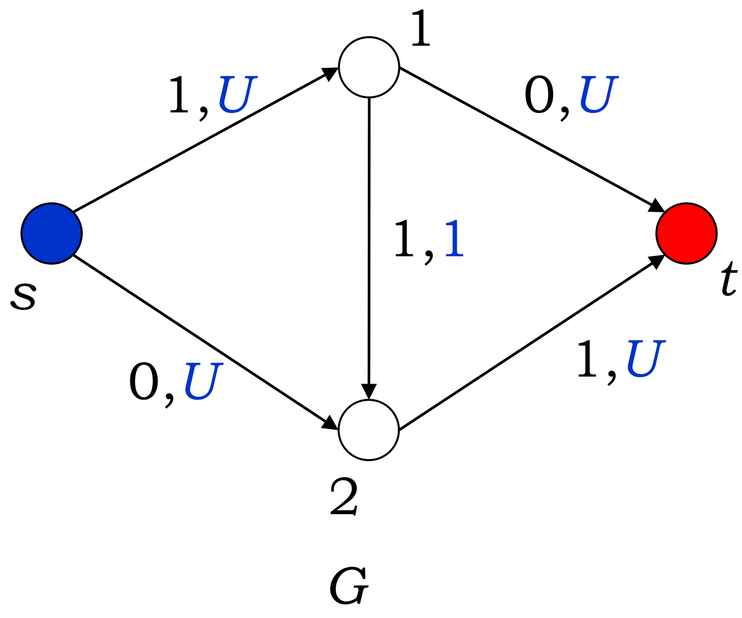
# Example

Augmenting Path in $G$: $\{s, 1, 2, t\}$. $\varepsilon = 1$

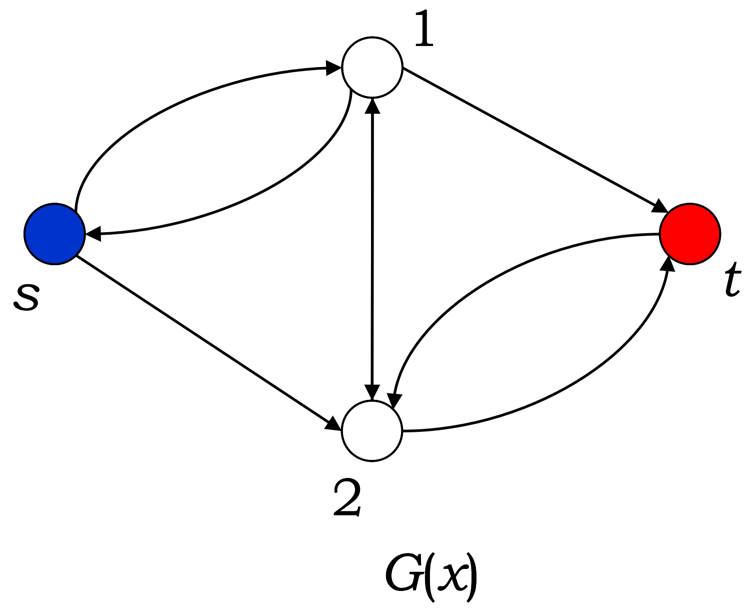


# Example

flow of value 1

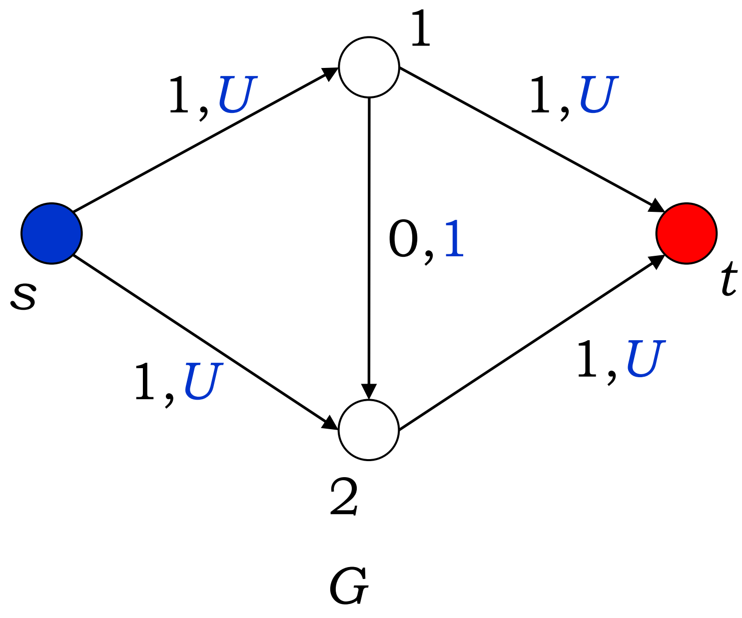


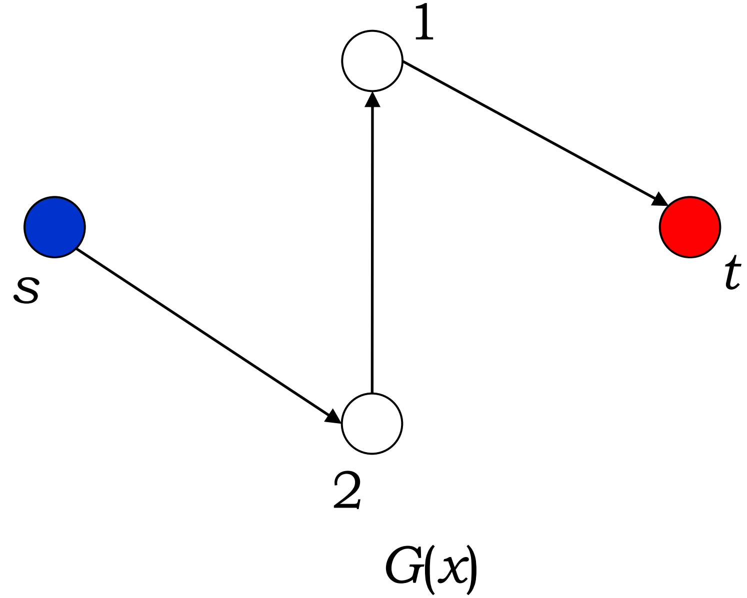

Auxiliary graph



# Example

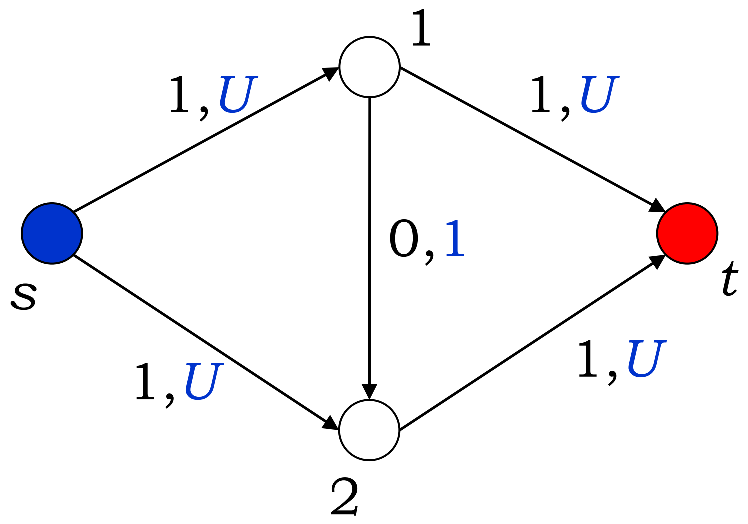
Augmenting path $G$: $\{s, 2, 1, t\}$, $\varepsilon = 1$



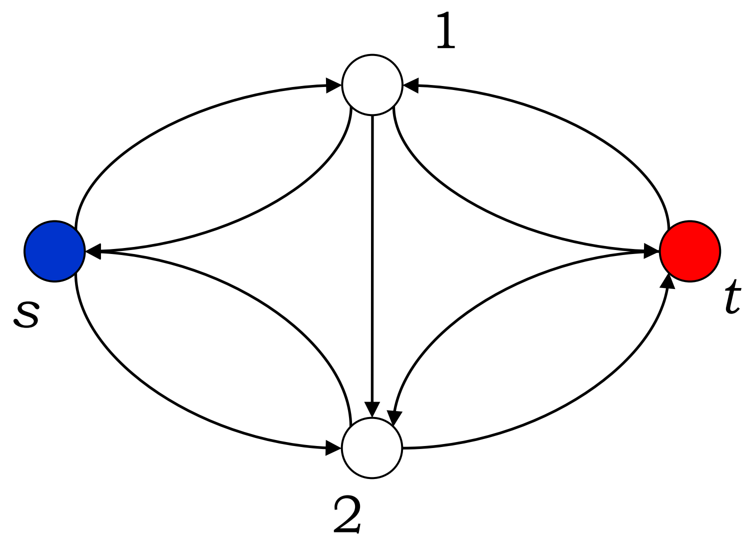
# Example

Value of flow 2




$G$

auxiliary graph

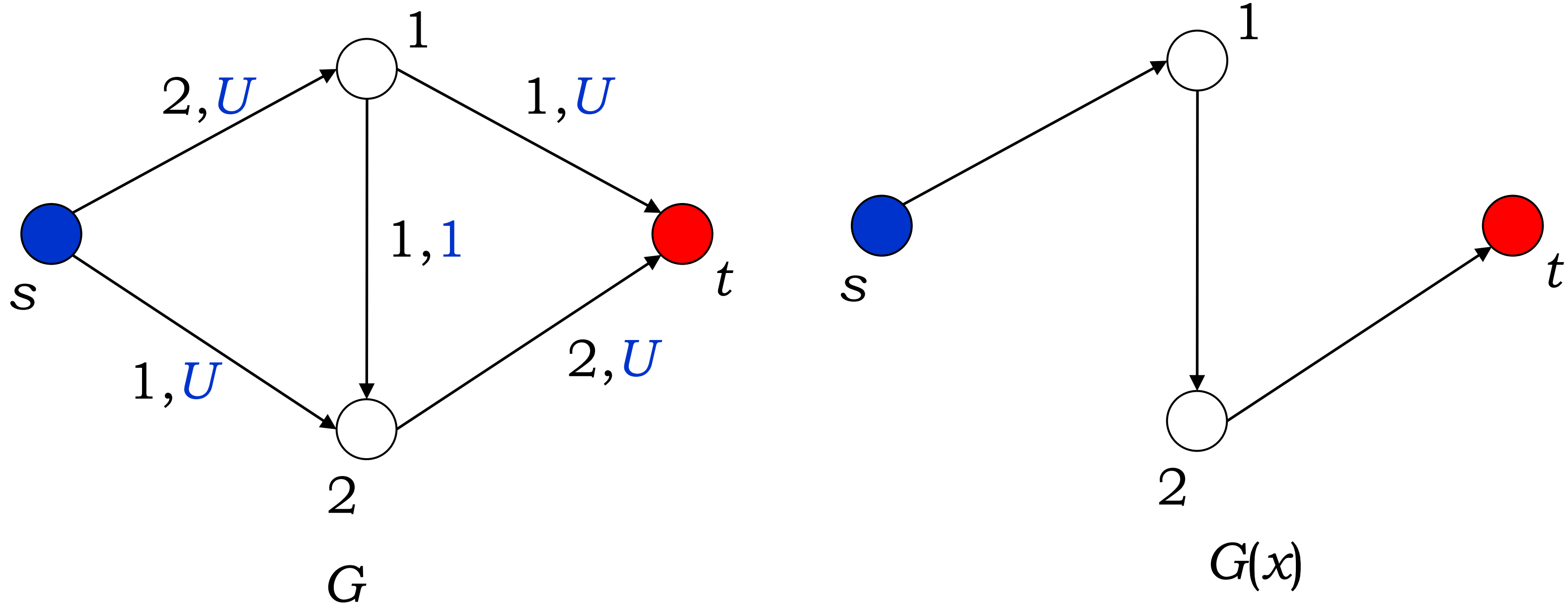



$G(x)$

# Example

Augmenting path on $G$: $\{s, 1, 2, t\}$. $\varepsilon = 1$

Flow value: 3



# Example

By repeating this choice of augmeting paths, the algorithm terminates after exactly 2U iterations!

Observation

If the value of U is not integer, the algorithm could not terminate!

A simple criterion for choosing the augmenting path at each iteration removes this difficulty and leads to a polynomial version of Ford and Fulkerson algorithm [ie, not dependent on $U$]. We will show that it is sufficient to choose at each iteration the path in G (x) that minimizes the number of used arcs (in other words, the "shortest augmenting path").[Edmonds e Karp]

# Shortest augmenting path

At a generic iteration of the F&F algorithm, the flow $x'$ is obtained from flow $x$ through an increasing path $P = \{v_0, v_1, \ldots, v_k\}$.

Let $d_x(v,w)$ the length of the path from $v$ to $w$ with the minimum number of arcs.

If $P$ is the shortest augmenting path, the following properties hold:

1. $d_x(s, v_i) = i$
2. $d_x(v_i, t) = k - i$

In addition, if an arc $(v,w)$ $G(x')$ does not belong to $G(x)$, then there exists an index $i$ such that $v=v_i$, $w=v_i-1$.

In other words, the arc $(w, v)$ was an arc of the shortest augmenting path on $G(x)$.

# Shortest augmenting path

Lemma 1

For each $v \in N$, $d_{x'}(s, v) \geq d_x(s,v)$ and $d_{x'}(v, t) \geq d_x(v,t)$

Demonstration

Suppose that there exists a node $v$ such that $d_{x'}(s, v) < d_x(s,v)$ and choose $v$ so that $d_{x'}(s, v)$ is as small as possible.

Since $v \neq s$, $d_{x'}(s, v) > 0$.

Let $P'$ be an $(s,v)$ path in $G(x')$ and $w$ the next-to-last node of $P'$. One has:

$$d_x(s,v) > d_{x'}(s, v) = d_{x'}(s, w) + 1 \geq d_x(s, w) + 1$$

# Proof

If $d_x(s,v) > d_x(s, w) + 1$, then the arc $(w,v)$ does not belong to $G(x)$ (otherwise $d_x(s,v) = d_x(s, w) + 1$).

If the arc $(w,v)$ (which belongs to $G(x')$) does not belong to $G(x)$ then there exists an index $i$ such that $w = v_i$ e $v = v_i$-1.

Therefore, $d_x(s,v) = i$ -1 $> d_x(s, w) + 1 = i + 1$, and there is a contradiction.

With a similar argument one can prove the second part of the Lemma ■

# Shortest augmenting path

## Lemma 2

During the execution of the Edmonds and Karp algorithm an arc $(i,j)$ disappears (and appears) in $G(x)$ at most $n/2$ times.

## Proof

If an arc $(i,j)$ "disappear" from the auxiliary network, then it is on a augmenting path. The corresponding arc in $G$ is either saturated or is empty. Therefore, in the next auxiliary network the arc $(j,i)$ appears. Let $x_f$ be the flow when the arc disappears. Suppose that at any successive iteration the arc $(i,j)$ re-appear in $G(x_h)$. This means that the augmenting path that has generated $x_h$ contains the arc $(j,i)$.

# Shortest augmenting path

So, if $x_g$ is the flow from which $x_h$ has been generated , one has [by Lemma 1]

$$d_g(s, i) = d_g(s, j) + 1 \geq d_f(s,j) + 1 = d_f(s,i) + 2$$

Therefore, in moving from flow $x_f$ to flow $x_h$, $d(s,u)$ has been increased by at least 2 units. Since the maximum value that can assume $d(s,u)$ is $n$ , an arc can disappear and reappear at most $n/2$ times. ■

# Shortest augmenting path

Lemma 3

The complexity of the Edmonds and Karp's algorithm is $O(nm^2)$.

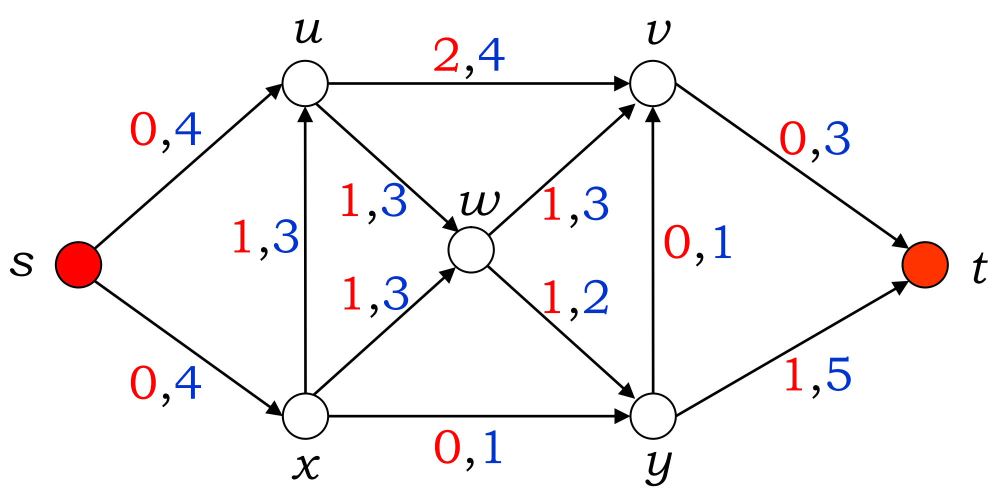
Demonstration

Each arc can "disappear" at most $n/2$ times during the execution of the algorithm [Lemma 2]. Every time the flow is increased at least one arc disappears. Therefore, during the execution, there are at most $mn/2$ "disappearances". Each augmenting operation requires $O(m)$ and, therefore, the complexity of the algorithm is $O(nm^2)$.

■

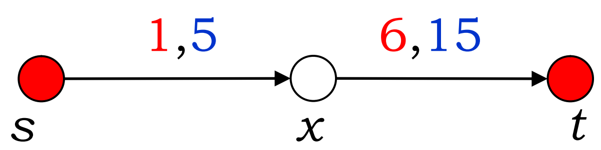
# Flows with lower bounds

The red labels on the following network represent a minimal amount of flow that must be transported by arc $(i,j)$ (Notation: $(l_{ij},\ u_{ij})$ lower bound, capacity)



# Observation

A flow problem with a positive minimum requirement on the arcs may be infeasible.



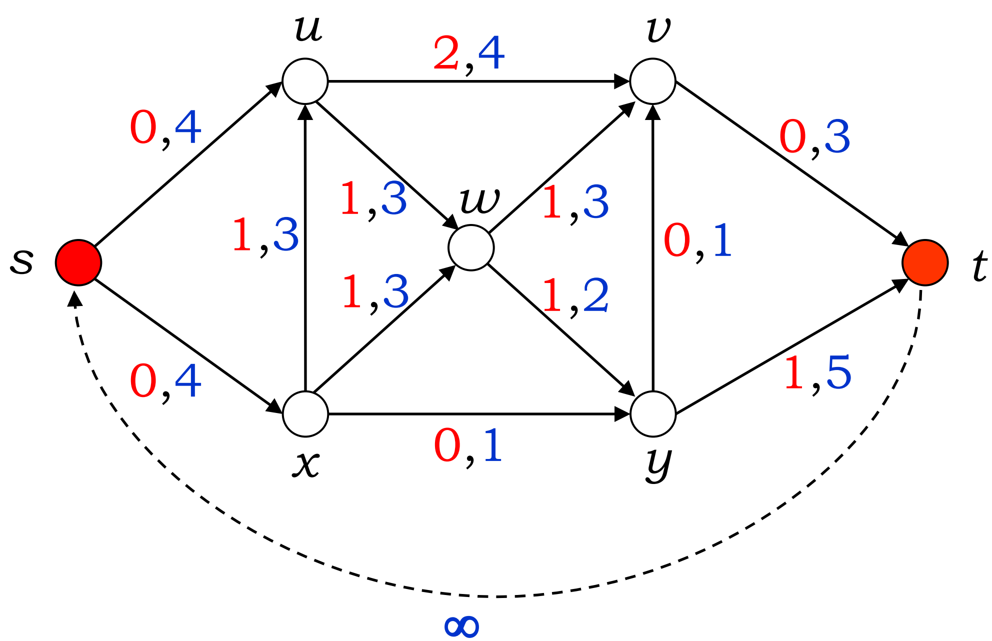

Therefore, before calculating the maximum flow on the network, we have to check if the problem is feasible.

Procedure:

1. Transform the flow problem into a circulation problem, adding an arc $(t, s)$ of infinite capacity.

# 1. Feasibility

The original problem admits a feasible solution if and only if the circulation problem is feasible



# 1. Feasibility

In the circulation problem the incoming flow in each node is equal to the outgoing flow: :

$s$: $x_{su} + x_{sx} - x_{ts} = 0$
$u$: $x_{uv} + x_{uw} - x_{su} - x_{xu} = 0$
$v$: $x_{vt} - x_{uv} - x_{wv} - x_{yv} = 0$
$w$: $x_{wv} + x_{wy} - x_{uw} - x_{xw} = 0$
$x$: $x_{xu} + x_{xw} + x_{xy} - x_{sx} = 0$
$y$: $x_{yv} + x_{yt} - x_{wy} - x_{xy} = 0$
$t$: $x_{ts} - x_{vt} - x_{yt} = 0$
$l_{ij} \leq x_{ij} \leq u_{ij} \quad \forall$ arco $(i,j)$

Variable substitution: $x_{ij} = x'_{ij} + l_{ij}$

# 1. Feasibility

$s$: $x'_{su} + x'_{sx} - x'_{ts} = b(s) = 0$
$u$: $x'_{uv} + x'_{uw} - x'_{su} - x'_{xu} = b(u) = -2$
$v$: $x'_{vt} - x'_{uv} - x'_{wv} - x'_{yv} = b(v) = 3$
$w$: $x'_{wv} + x'_{wy} - x'_{uw} - x'_{xw} = b(w) = 0$
$x$: $x'_{xu} + x'_{xw} + x'_{xy} - x'_{sx} = b(x) = -2$
$y$: $x'_{yv} + x'_{yt} - x'_{wy} - x'_{xy} = b(y) = 0$
$t$: $x'_{ts} - x'_{vt} - x'_{yt} = b(t) = 1$

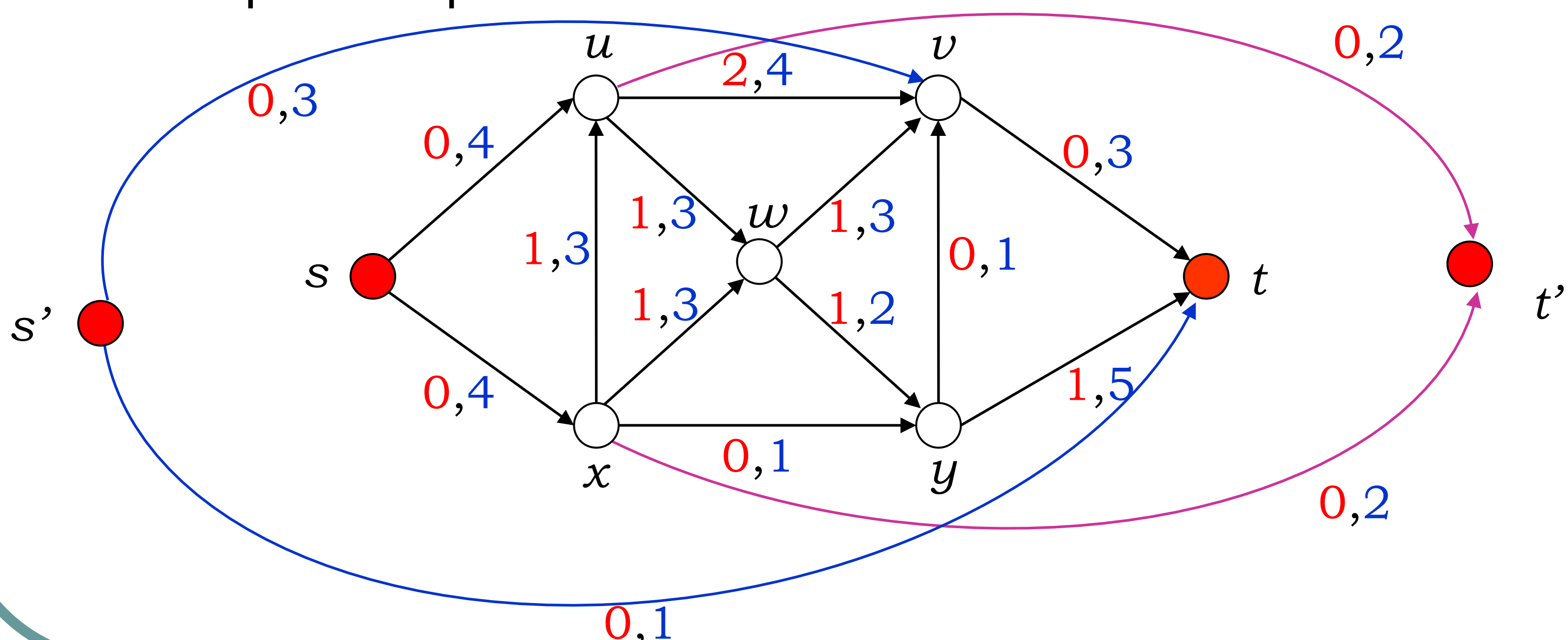
$0 \leq x'_{ij} \leq u_{ij} - l_{ij} \;\; \forall$ arc $(i,j)$

This is a circulation problem with "special" balance constraints (RHS not all equal to zero).

# 1. Feasibility

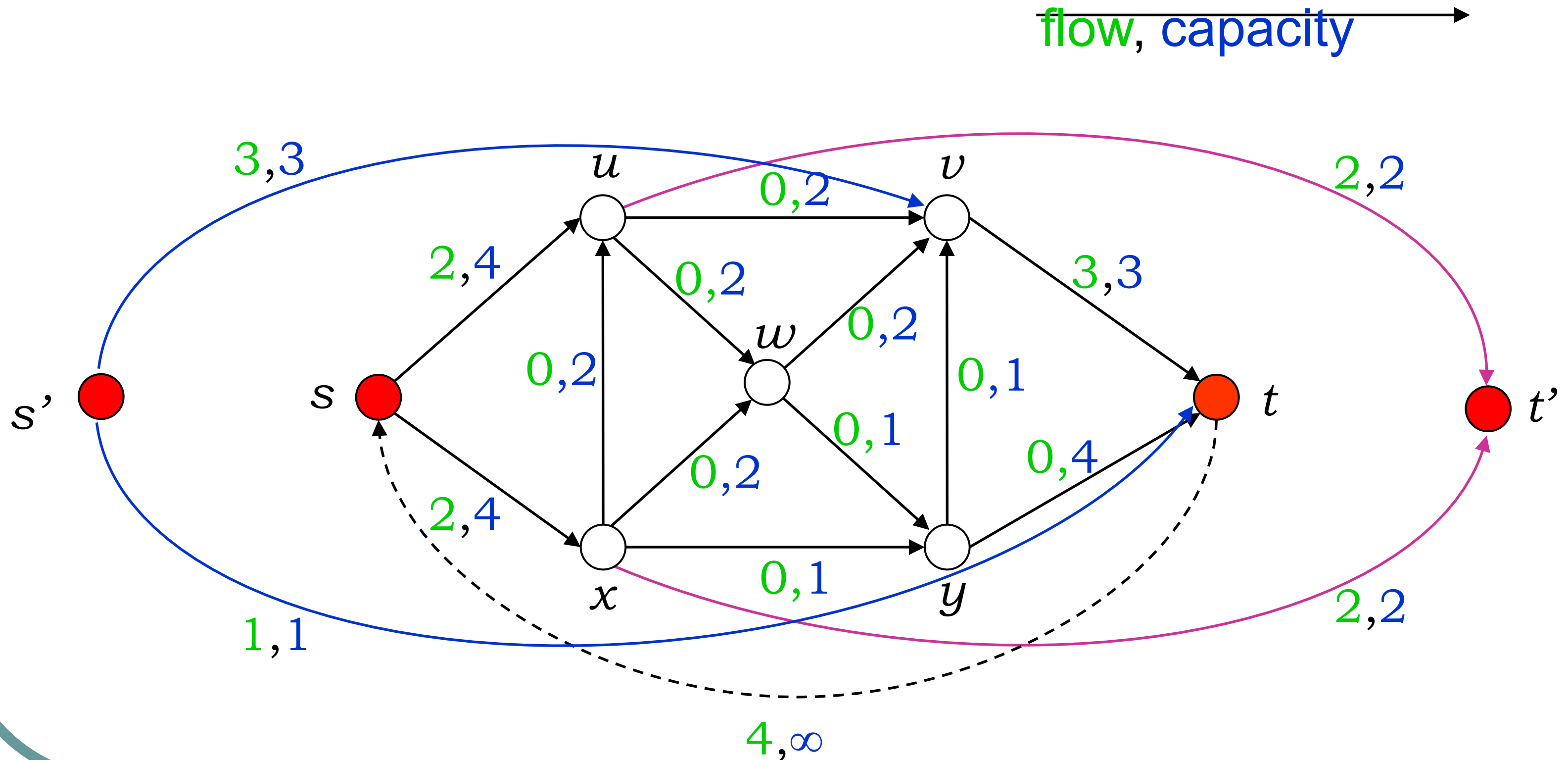
Now, introduce two nodes $s'$ and $t'$.

1. For each node $i$ with $b(i) > 0$ add an arc $(s', i)$ with capacity $b(i)$ and request $l$ equal to 0
2. For each node $i$ with $b(i) < 0$ add an arc $(i, t')$ with capacity $-b(i)$ and request $l$ equal to 0



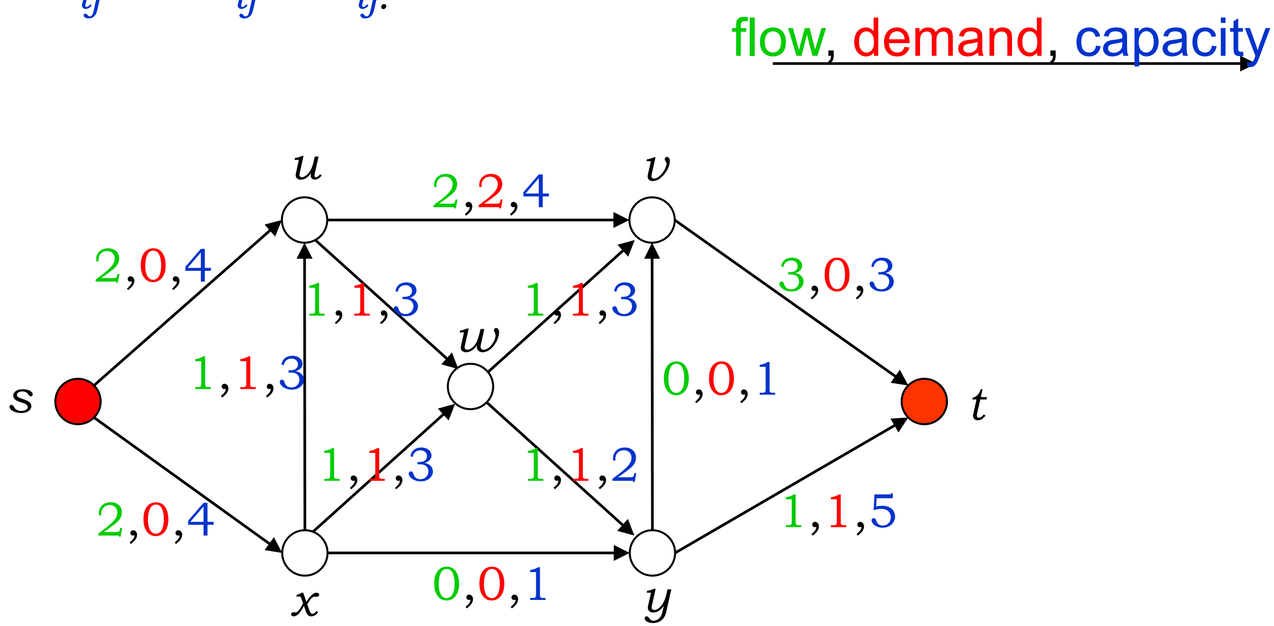
# 1. Feasibility

If the maximum flow saturates the arcs $(s', i)$ or, equivalently, the arcs $(i, t')$ then the initial problem is feasible, otherwise it is infeasible.



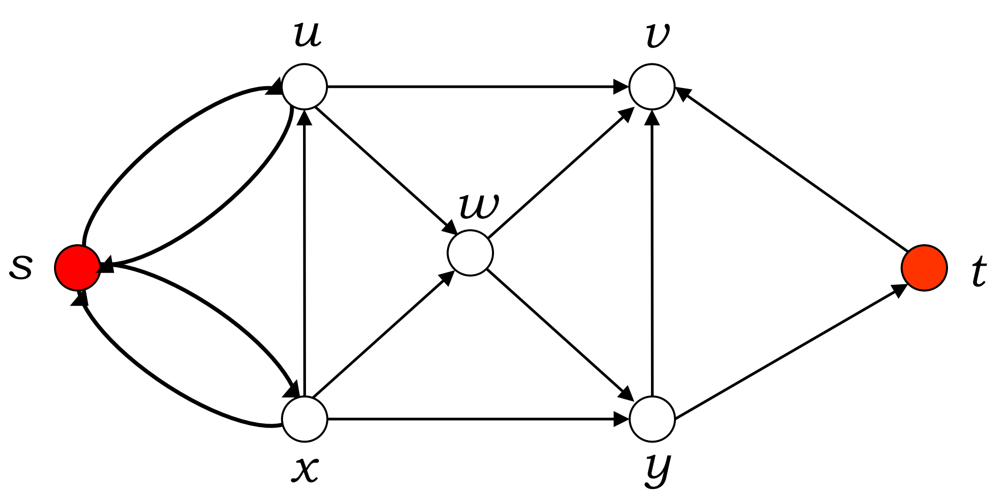
# 1. Feasibility

To obtain a feasible solution to the original problem one has to delete the added arcs and restore the original variables $x_{ij} = x'_{ij} + l_{ij}$.



# 2. Optimality

Starting from the feasible flow found we determine the maximum flow.
Construct the auxiliary graph by putting an arc "forward" if $x_{ij} < u_{ij}$ and an arc "reverse" if $x_{ij} > l_{ij}$ and look for an augmenting path.

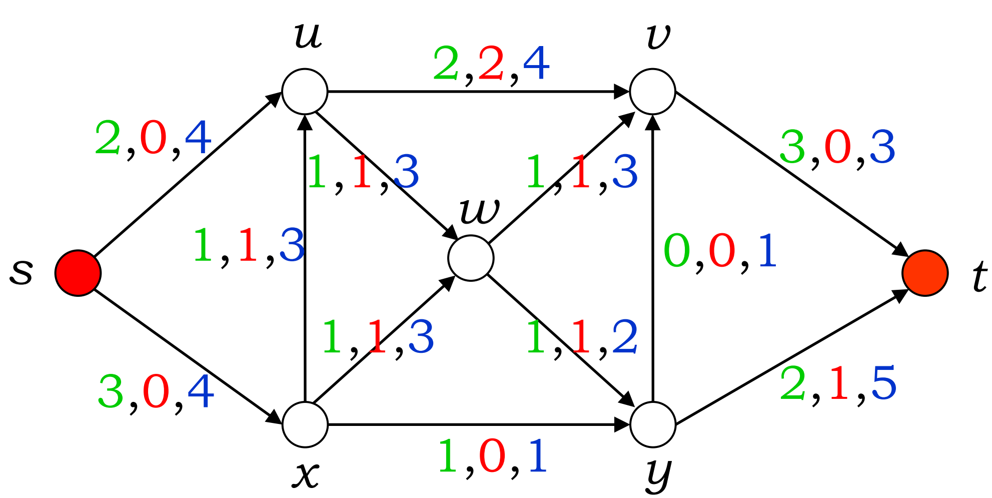


# Optimality

For instance, the augmenting path P: $\{s, x, y, t\}$.

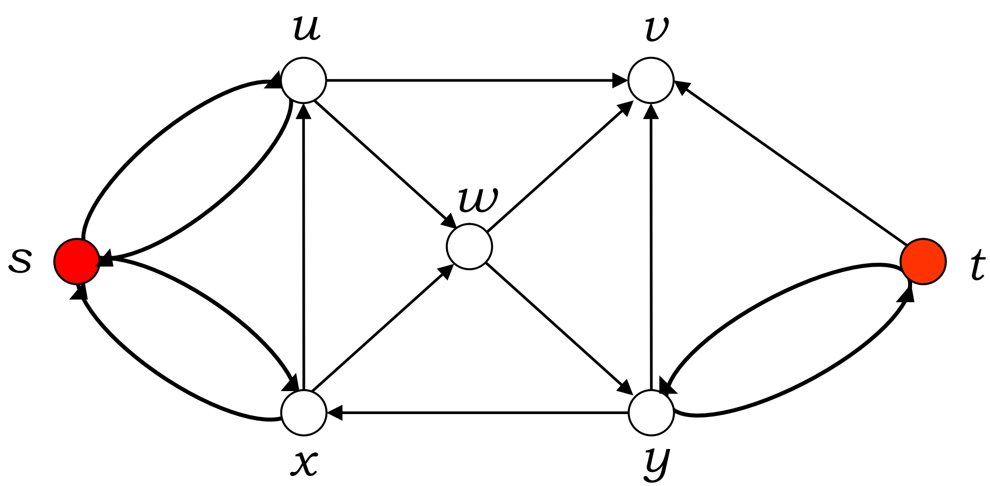
The maximum flow that can be sent on that path is the minimum value of $(u_{ij} - x_{ij})$ for each "forward" arc and $(x_{ij} - l_{ij})$ for each "reverse" arc

In this case: $\min\{2, 1, 4\} = 1.$



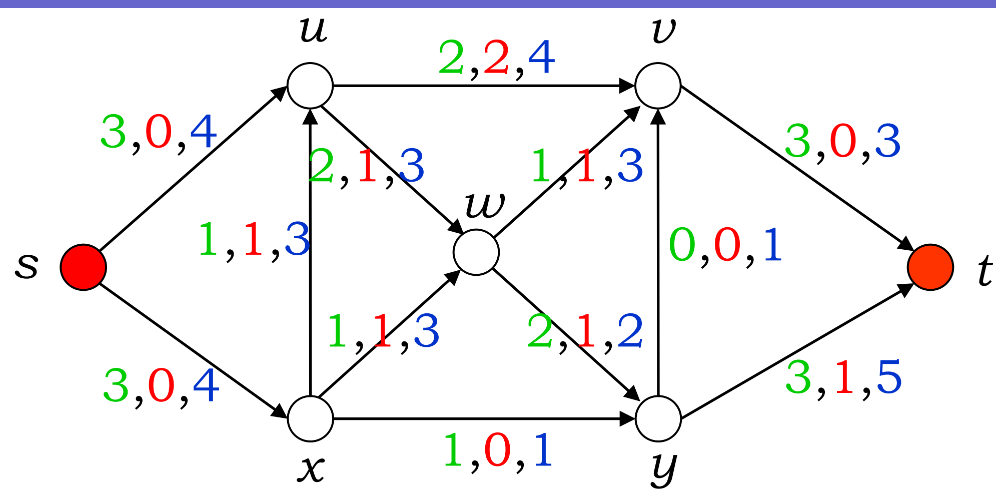
# Optimality

The new auxiliary graph is:

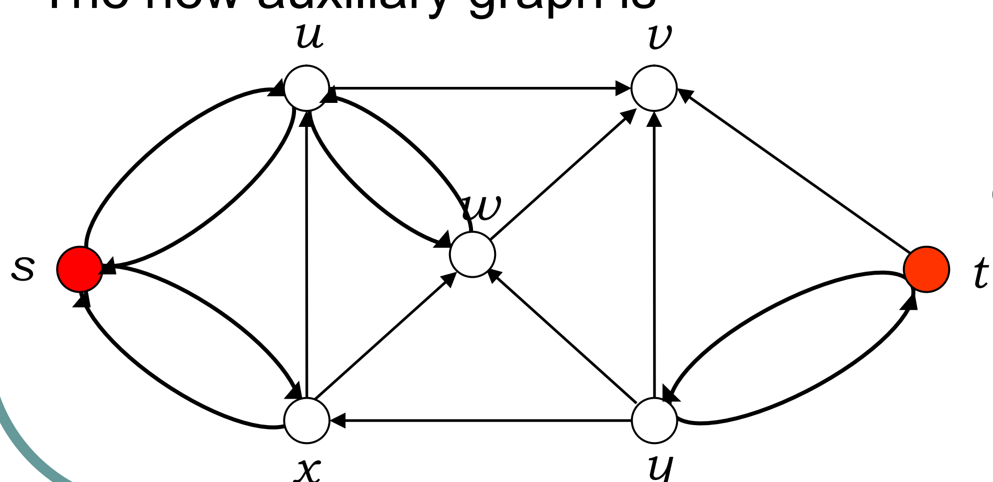



An augmenting path is P: $\{s,\ u,\ w,\ y,\ t\}$, again of value 1. We update the flow:

# Optimality



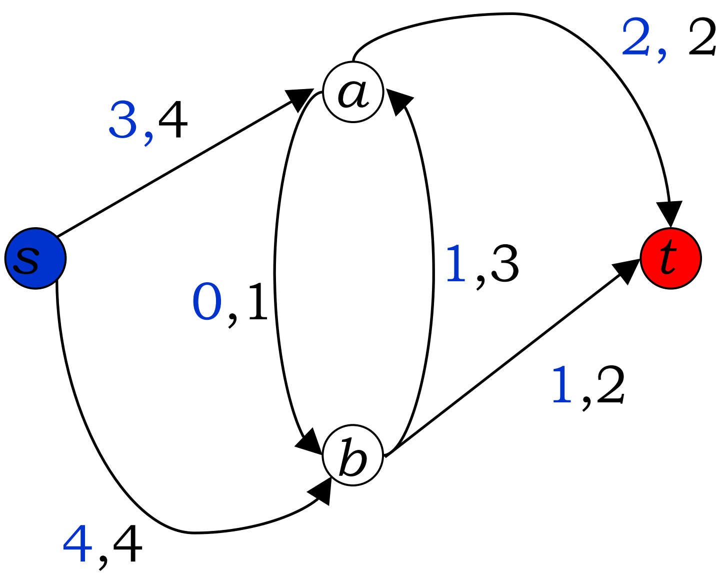

The new auxiliary graph is




On the last auxiliary graph does not exist at augmenting path, so the flow is the maximum

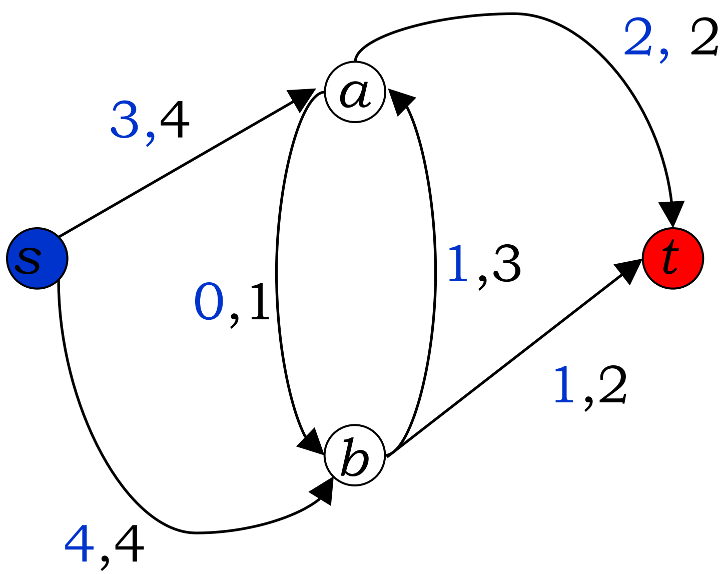
# A new algorithm

Consider the following graph and the following flow distribution : (in black capacity, flow blue)



# A new algorithm

The flow is not feasible because the balance constraints are not satisfied at the nodes (inflow > outflow)




$e_x(a) = 3 + 1 - 2 - 0 = 2 > 0$

$e_x(b) = 4 + 0 - 1 - 1 = 2 > 0$

However, capacity constraints are satisfied on the arcs.

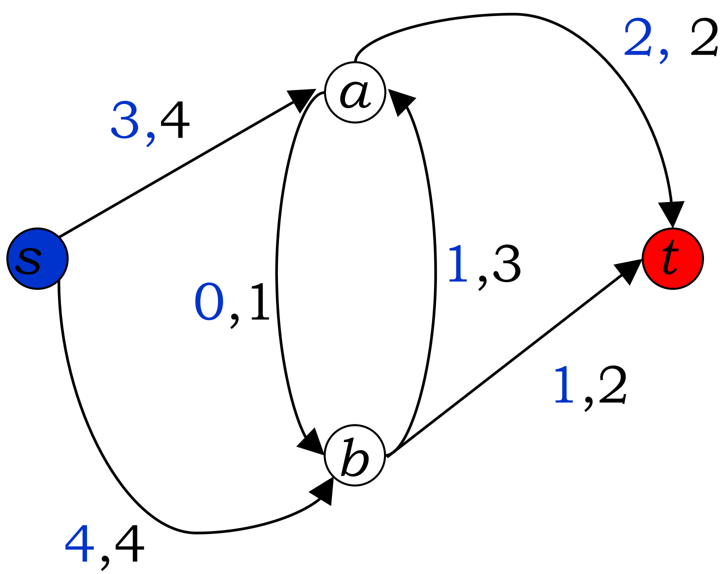
$e_x(i)$= "excess" of flow in the node $i$

# Preflow

A vector $x \in \mathcal{Z}_+^{|A|}$ such that:

1. $e_x(n) \geq 0$ for each node $n \in N \setminus \{s, t\}$
2. $0 \leq x_{ij} \leq u_{ij}$ for each arc $(i, j) \in A$

Is called preflow.

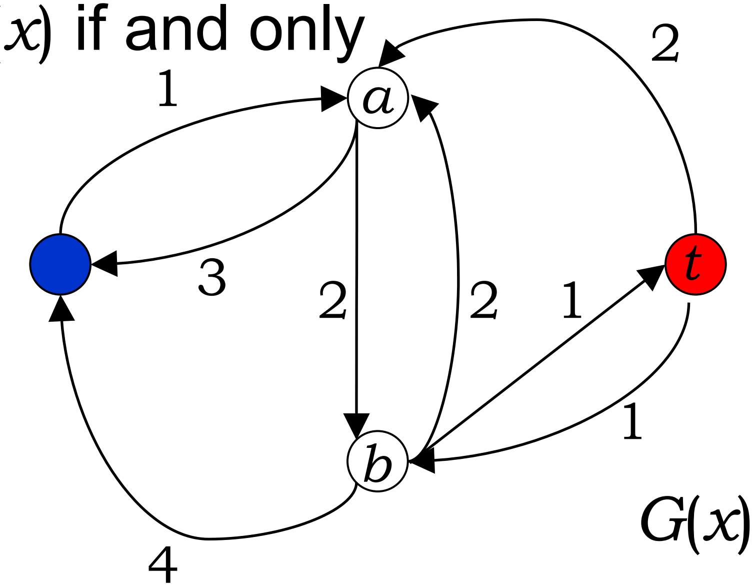


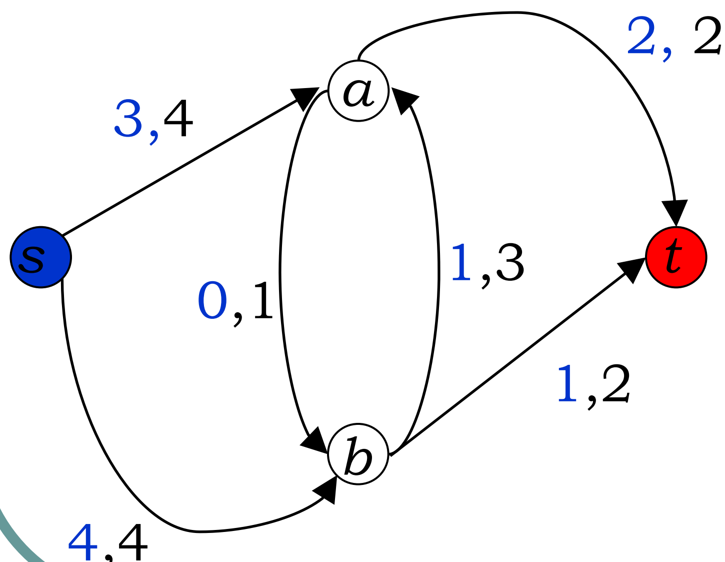

Any preflow can be associated with a residual network $G(x)$ with the following characteristics:

# Residual network

1. There is an arc $(i,j)$ in $G(x)$ if and only if $x_{ji} > 0$ or $x_{ij} < u_{ij}$

(eliminate any parallel arc)

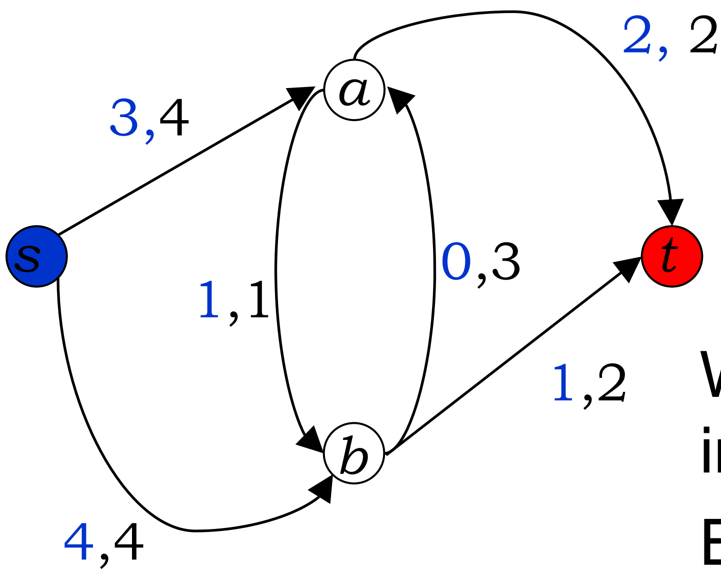



2. Relabel the arc $(i,j)$ in $G(x)$ with $u'_{ij} = u_{ij} - x_{ij} + x_{ji}$

# The operation "push"

The residual network tells us that we can "push" 2 units of flow from $a$ to $b$, obtaining:




What happens to excess flow in $a$ and $b$?

Before: $e_x(a) = 2$; $e_x(b) = 2$

After: $e_x(a) = 0$; $e_x(b) = 4$

# The "push" operation

The push operation has modified the excess flow at the nodes $a$ and $b$, but did not cause the violation of capacity constraints on arcs.

In particular, we moved from a feasible preflow to a new feasible preflow (i.e., such that $e_x(n) \geq 0$ for each node $n \in N \setminus \{s, t\}$)

Moreover, the excess flow $a$ is now equal to zero.

# Observation

If all the nodes of a preflow (but $s$ and $t$) have $e_x(i) = 0$, then the preflow is a feasible flow.

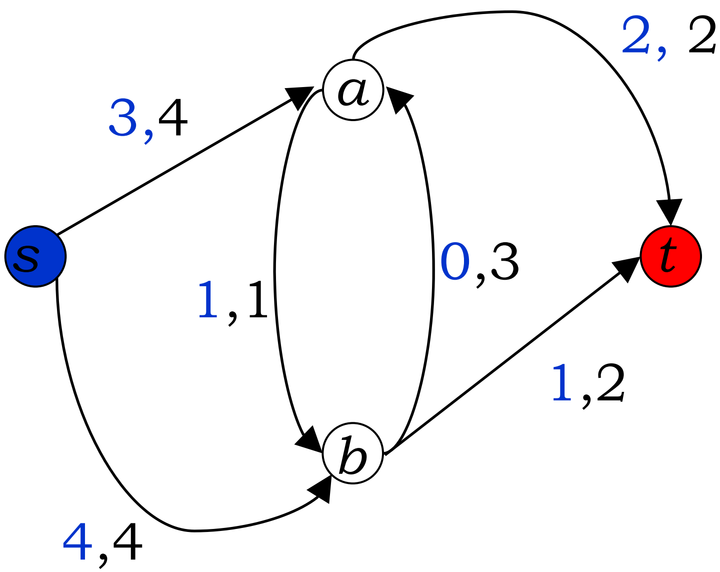
Given a feasible flow on $G$, $s$ has a negative excess of flow and $t$ has a positive excess of flow.

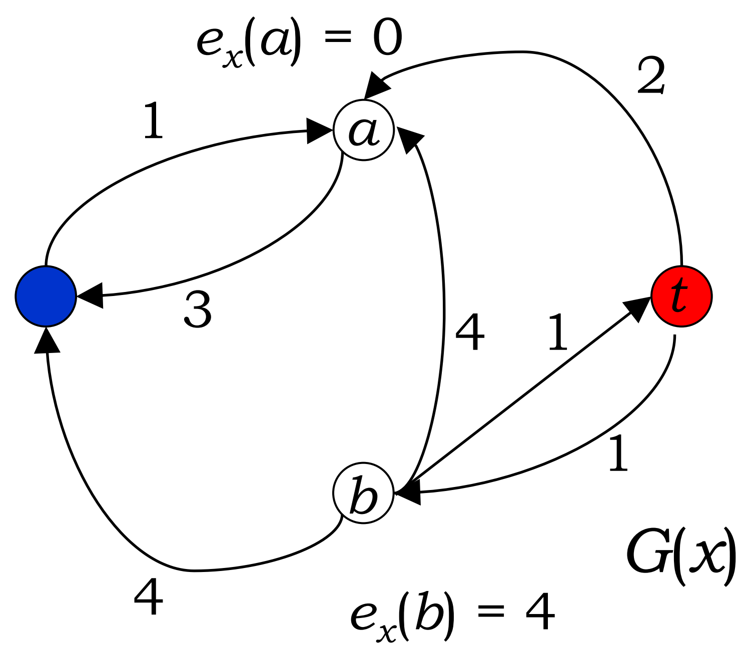
To mantain the preflow feasibility, you must perform a push on an arc $(i, j)$ with a flow value equal to $\min \{u'_{ij}, e_x(i)\}$

To decrease the excess flow one can try to push the flow towards the sink until it is possible. Otherwise, one can send flow back to the source.

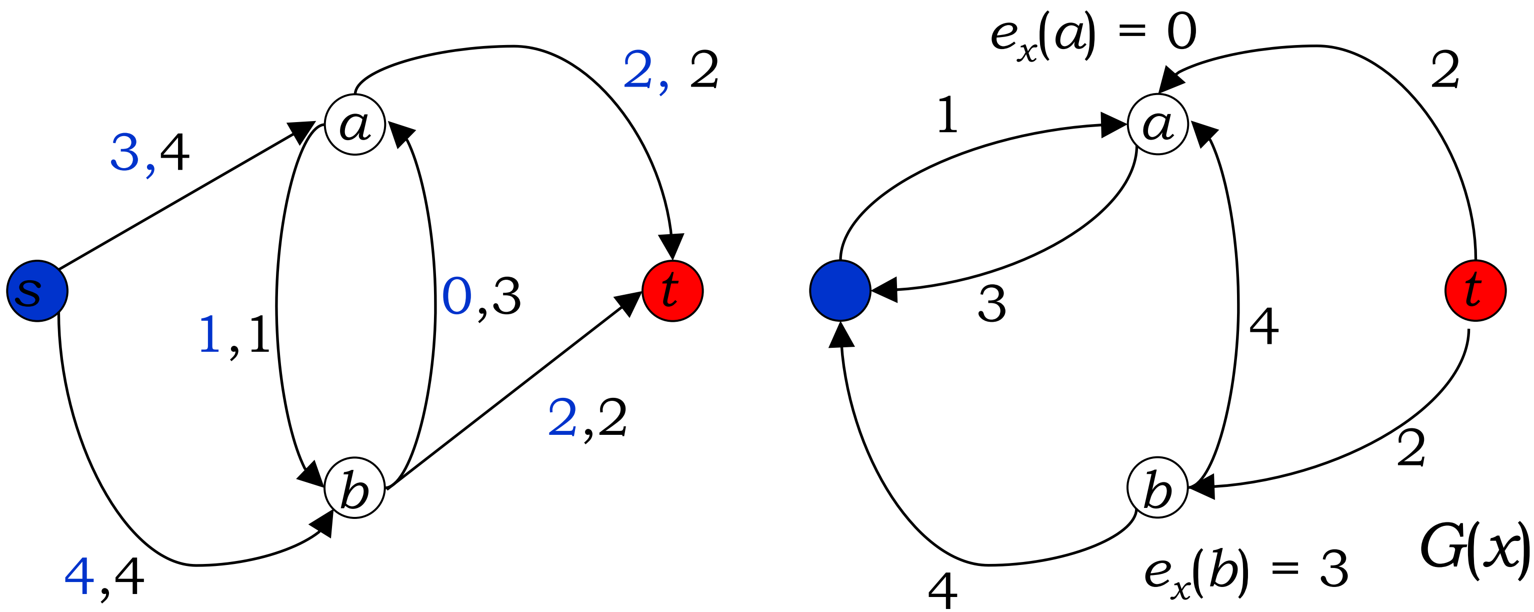
# The “push” operation

In this case we choose the arc $(b, t)$ $(1 = \min \{e_x(b), u'_{bt}\})$



# The "push" operation

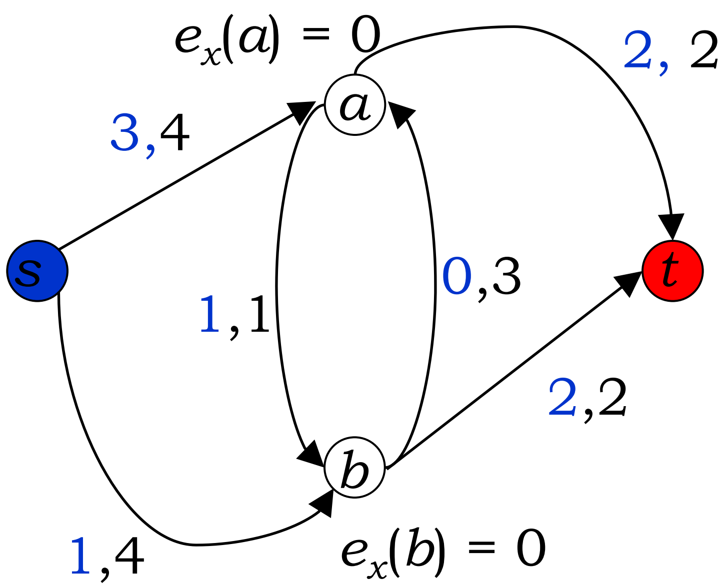



Now we have only two choices to decrease $e_x(b)$: the arc $(b,a)$ and arc $(b,s)$. However, if we choose the arc $(b,a)$ we increase $e_x(a)$.

Therefore, we choose $(b, s)$ $(3 = \min \{e_x(b),\ u'_{bs}\})$

# The "push" operation

We got a feasible flow.




Is it Optimal?

How can we formalize the previous choices?

# Labelling

Definitions

- A node $i$ *of* $G$ is called active if $e_x(i) > 0$
- A vector $\boldsymbol{d} \in (Z_+ \cup \{+\infty\})^{|A|}$ is a valid labelling for a preflow $x$ if:
  1. $d(s) = n$, $d(t) = 0$
  2. for each arc $(i,j)$ di $G(x)$, $d(i) \le d(j) + 1$

Observation

Denote by $d_x(i,t)$ the shortest path (in terms of number of arcs) from $i$ to $t$ on $G(x)$.

If we choose as labels $d(i) = d_x(i,t)$, all the above conditions except $d(s) = n$, are satisfied.

# Initialization

Given a graph $G = (N, A)$ it is always possible to identify a preflow and a valid labelling:

(Initialization)

1. $x_{si} = u_{si}$, for each arc $(s,i)$ outgoing from $s$
2. $x_{ij} = 0$ for all other arcs of $A$
3. $d(s) = n$, $d(i) = 0$ for all other nodes of $N$

Theorem

If $x$ is a feasible preflow and labelling $\boldsymbol{d}$ is valid for $x$, then their exists an $(s,t)$-cut $\delta(R)$ such that $x_{ij}=u_{ij}$ for each $(i,j) \in \delta(R)$ and $x_{ij}=0$ for each $(i,j) \in \delta(R)$

# Push-relabel

Consequence

If $x$ is a feasible flow and admits a valid labeling, then $x$ is a maximum flow.

Problem

How to build a flow and a valid labeling from a feasible preflow and a valid labeling?

Idea

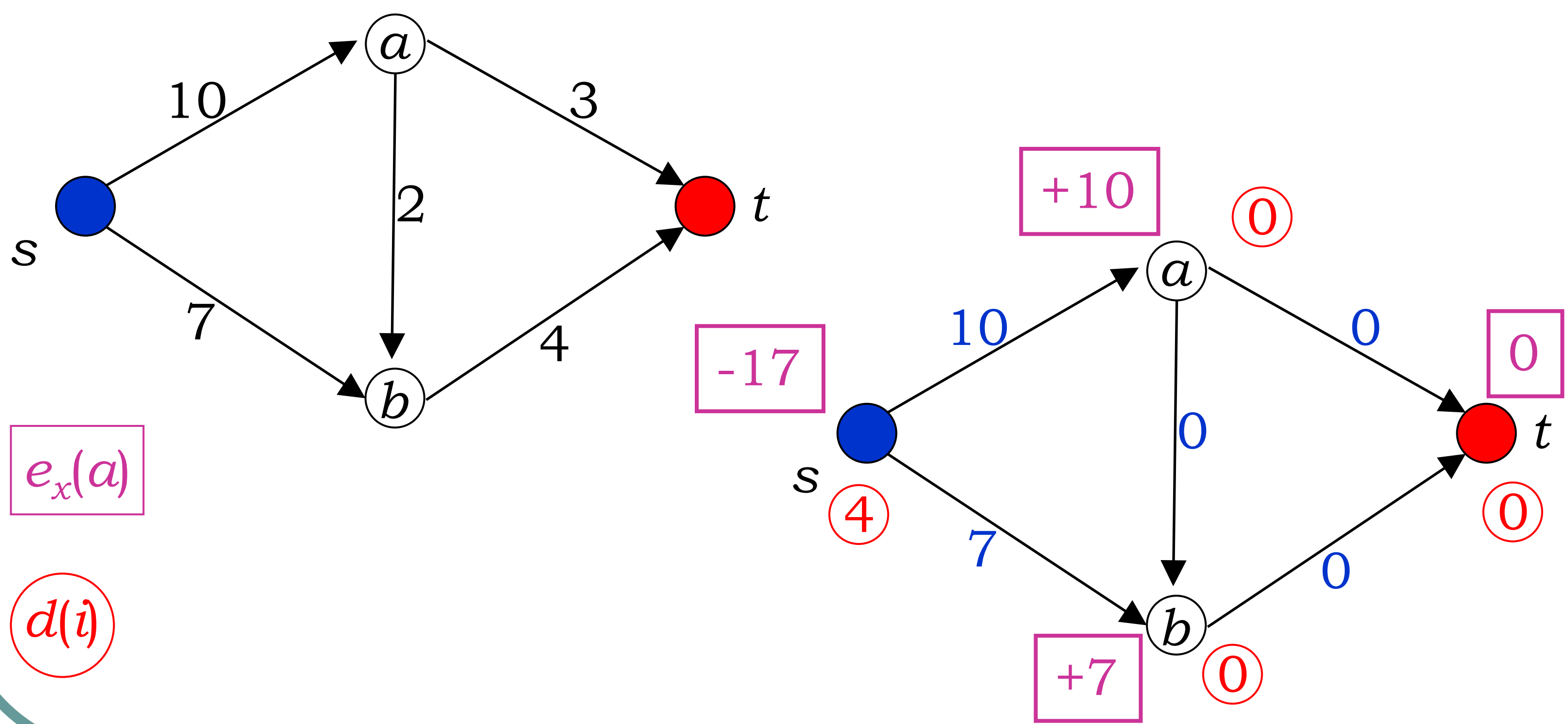
Push the flow from an active node on an arc $(i,j)$ with the property that $d(i) = d(j) + 1$

An arc $(i,j)$ with this property is said to be admissibile

# Push-relabel

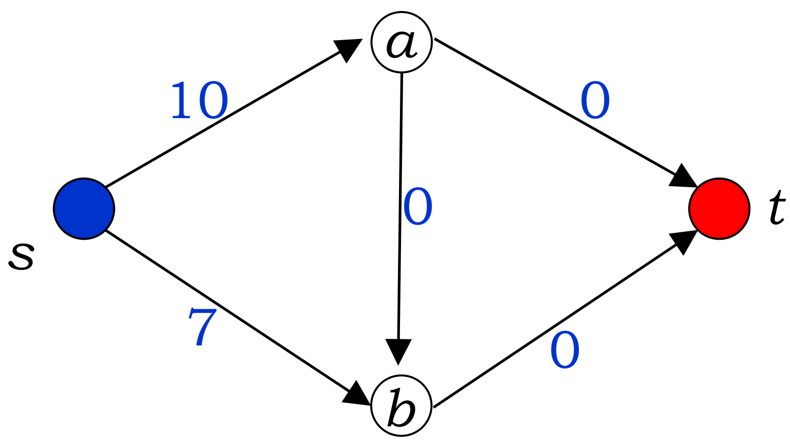
## Observation

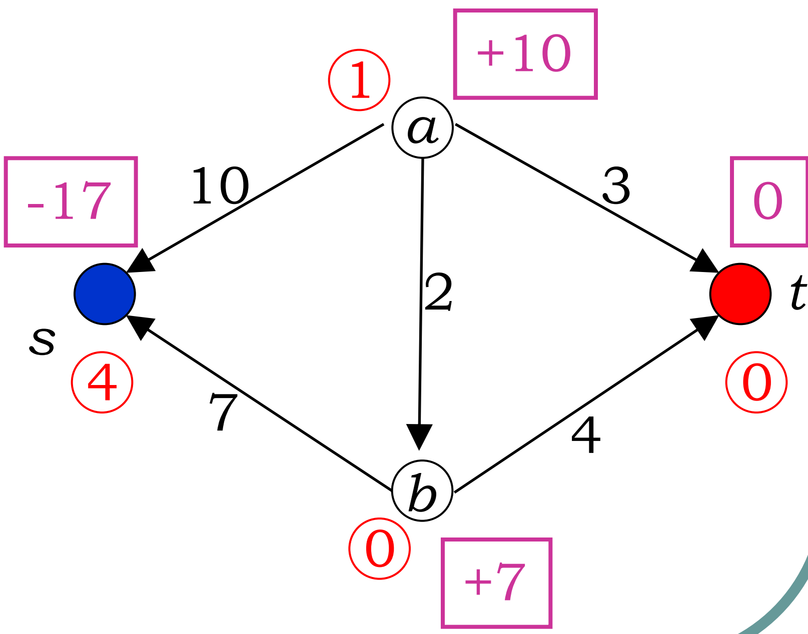
Admissible arcs may not exist



# Push-relabel

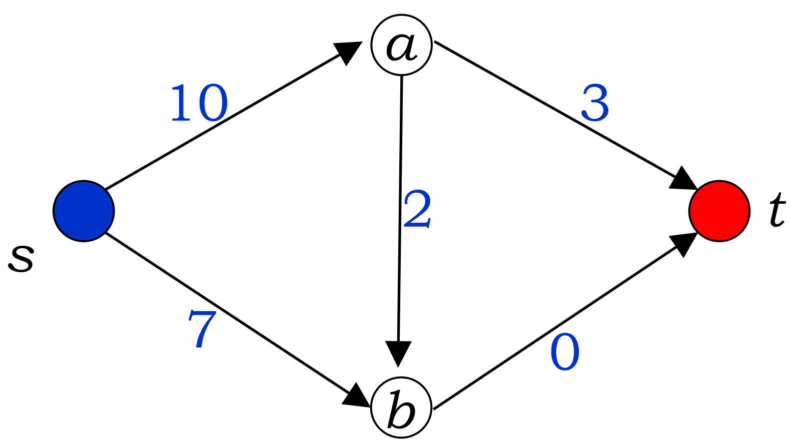
However, by selecting an active node $i$ and taking $d(i) = \min \{d(j) + 1\}$, for $(i,j)$ arc of di $G(x)$, I get a new valid labeling and at least one admissible arc (relabel)

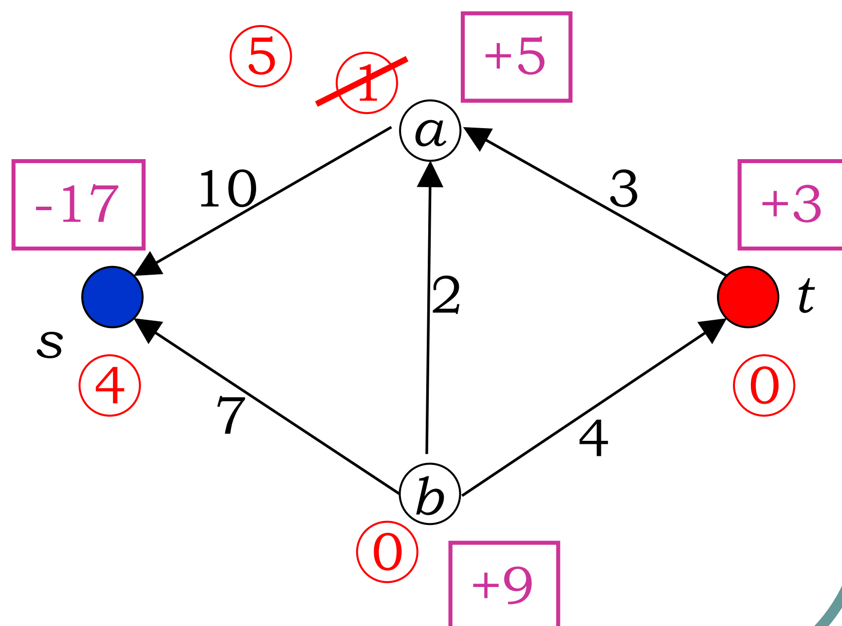


# Push-relabel

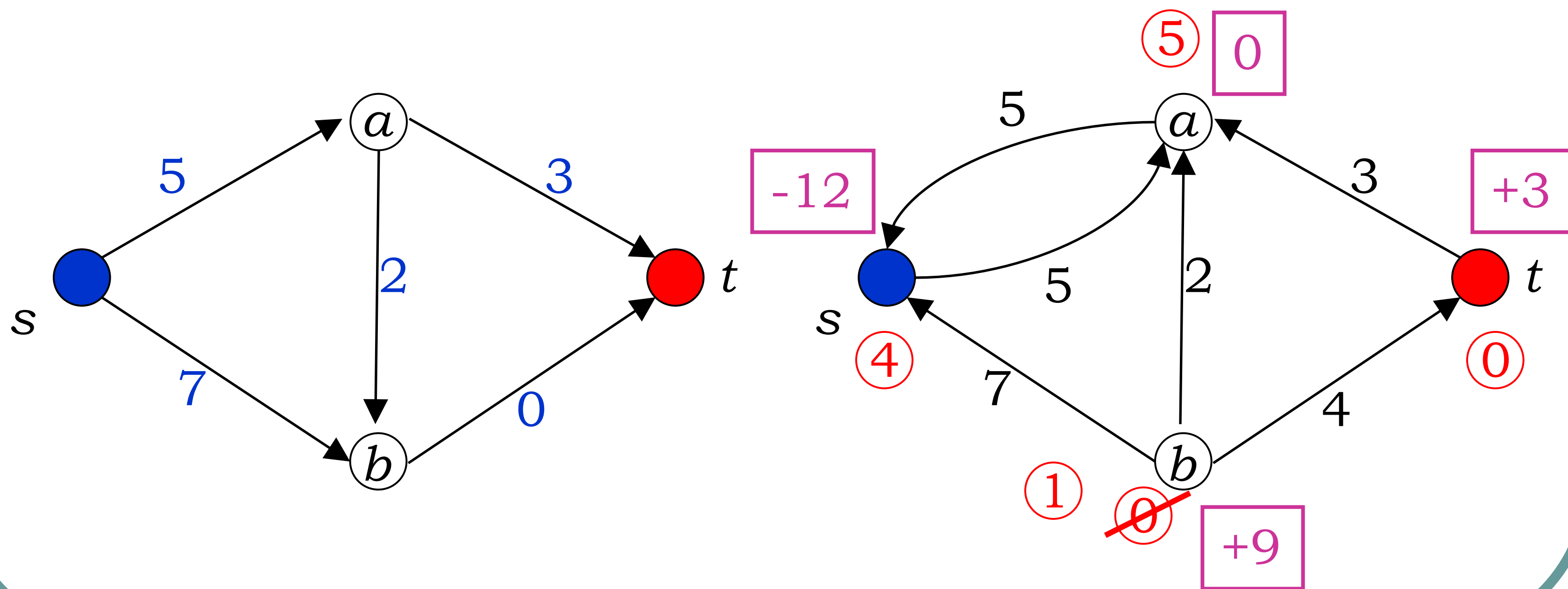
At this point I can make 2 push from node a. The first has value 3 on the arc $(a,t)$ and the second has value 2 on the arc $(a,b)$. $a$ is still active $\Rightarrow$ relabel



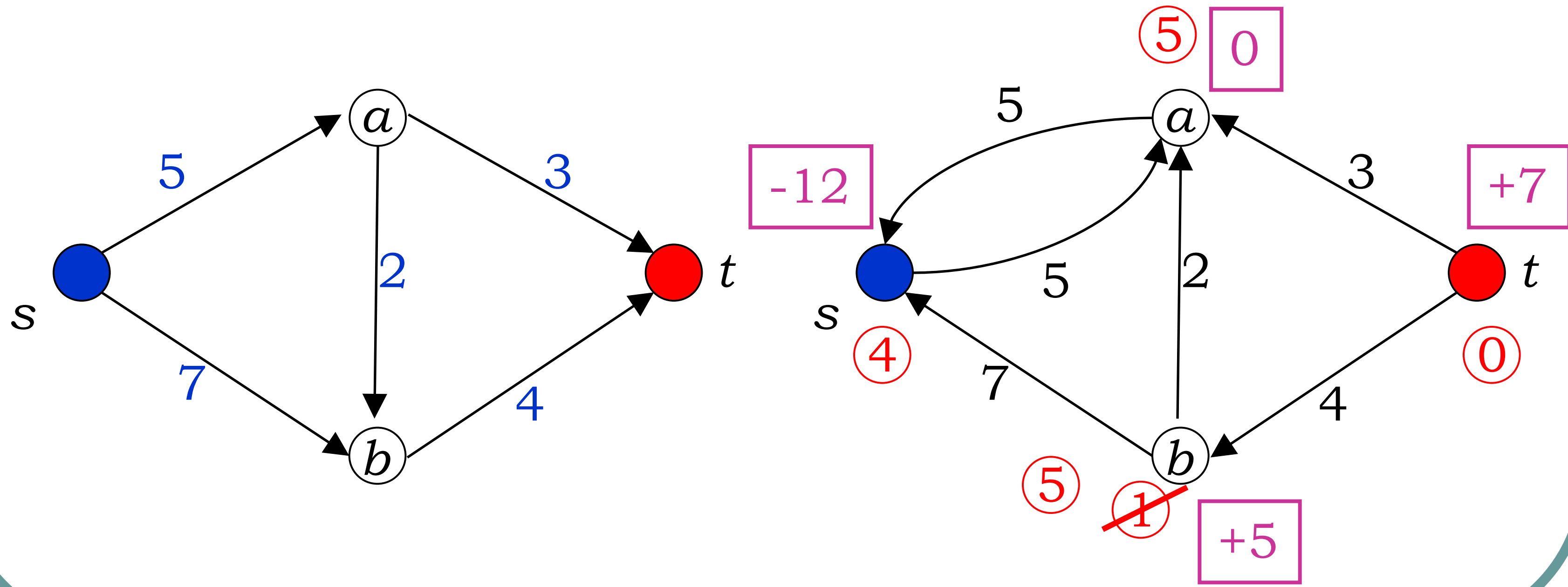
# Push-relabel

After a push of value 5 on the arc $(a,\ s)$ we remain with only one active node $b \Rightarrow$ relabel. After the relabeling one can make a push of value 4 on $(b, t)$



# Push-relabel

Now $b$ is still active but there are not active arcs $\Rightarrow$ relabel. Finally, we can make a push of value 5 on $(b,s)$.

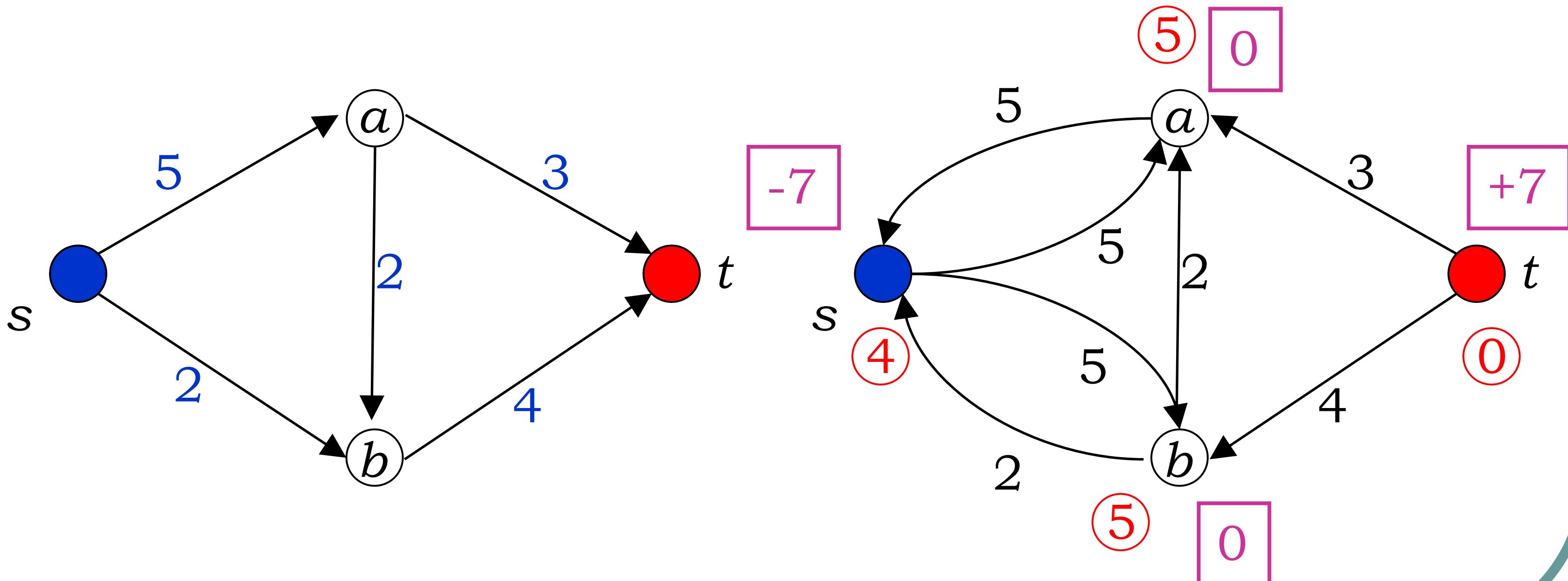


# Push-relabel

Now: 1) there are no active nodes

2) the labelling is valid

Then the solution is optimal!



# Push-relabel

Algorithm push-relabel

$x$ preflow, $d$ vector of labels

Initialize $x$ and $d$;

```
while (x is not a flow)
        Choose an active node i on G(x);
        while (there exists an admissible arc (i,j))
                push (i, j)
        if (i is active)
                relabel i
```

# Problem

A complex calculation program, consisting of 3 modules, must be run on a computer with 2 processors.

The table shows the cost of allocating modules to processors *P*1 and *P*2:

| | $M_1$ | $M_2$ | $M_3$ |
|---|---|---|---|
| $C_{P1}$ | 20 | 23 | 8 |
| $C_{P2}$ | 15 | 14 | 19 |

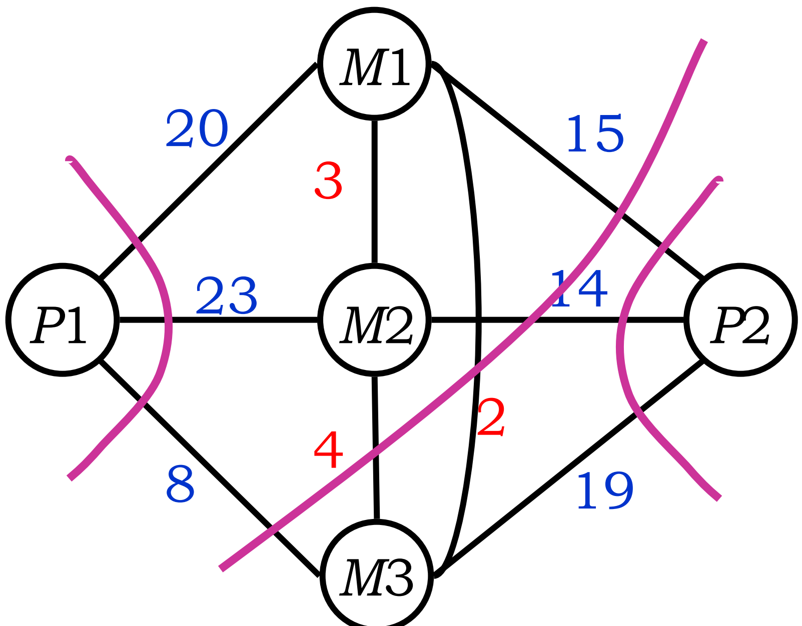
# Problem

In table are reported the costs $c_{ij}$ of intercommunication between processors, when two modules are assigned to two different processors:

| | $M_1$ | $M_2$ | $M_3$ |
|---|---|---|---|
| $M_1$ | 0 | 3 | 2 |
| $M_2$ | 3 | 0 | 4 |
| $M_3$ | 2 | 4 | 0 |

# Problem

Consider the following graph:




an $(s,t)$-cut on $G$ corresponds to an assignment of modules to processors and its cost is equal to the allocation cost plus the intercommunication cost. How we can determine an $(s,t)$ -minimum cut if $G$ is symmetric? And the “global" minimum cut of $G$?

# Minimum cut on symmetric graphs

## Minimum cut problem

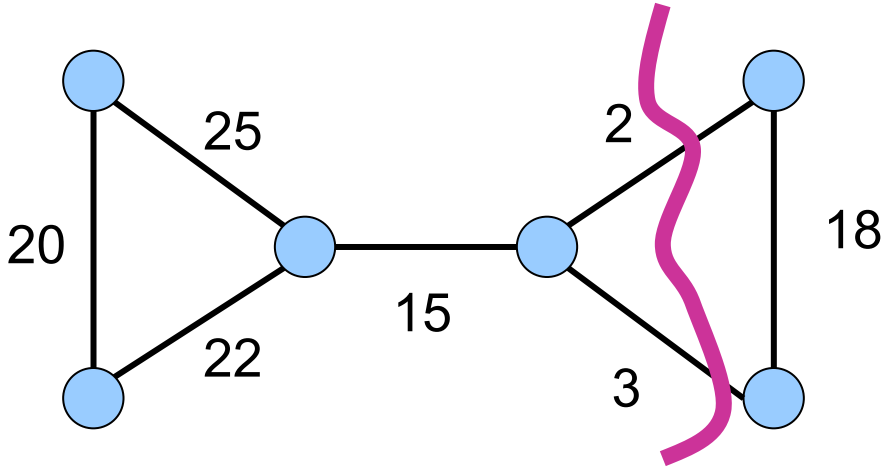
**Given**

$G=(V, E)$ symmetric and connected graph,
vector capacity $\boldsymbol{u} \in \mathcal{R}_{+}^{|E|}$

**Find**

A set of vertices $\emptyset \subset S \subset V$, such that $u(\delta(S))$ is minimum

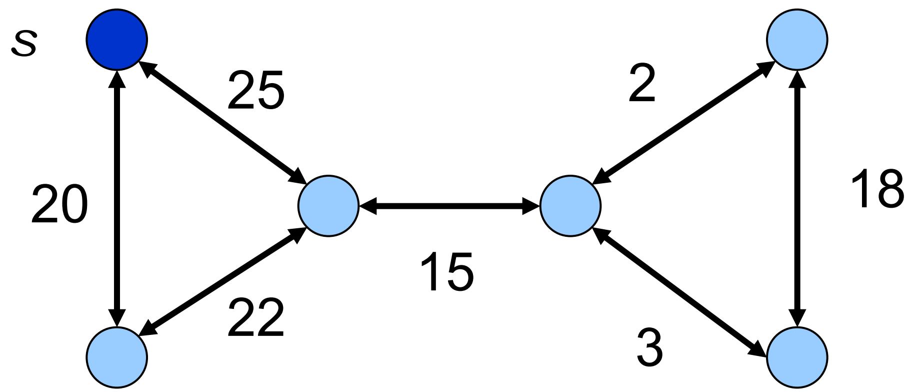
# Example




Observation

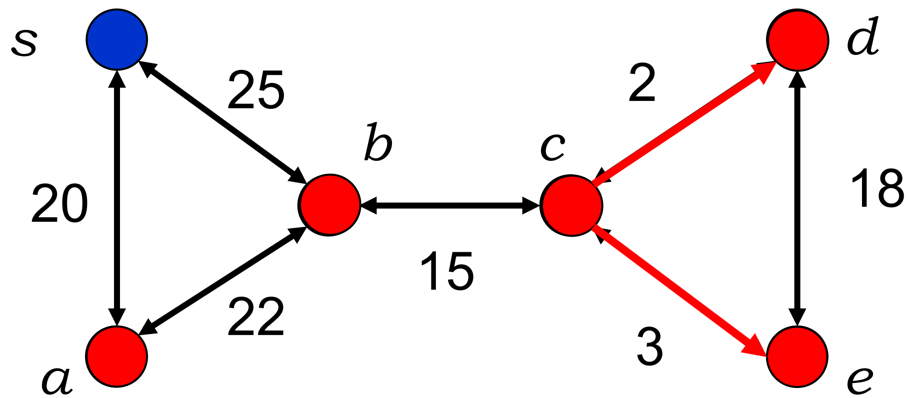
Can we solve this problem by using the minimum cut algorithm of Ford and Fulkerson?

# Example




1. Replace each arc with a pair of arcs with the same capacity of the original arc
2. Choose a node of $G$, for instance, the node $s$.
3. Solve $(n\text{-}1)$ instances of the maximum $(s,v)$-flow problem, where $v$ is a node of $G \neq S$.
4. The minimum cut among the $n$-1 cuts is the globally minimum cut. [Complexity: $O(nk)$, where k is the complexity of an algorithm for the max-flow. If $k = nm^2$, the complexity is $O(n^2m^2)$]

# Example




($s$, $a$): flow of value 42

($s$, $b$): flow of value 45

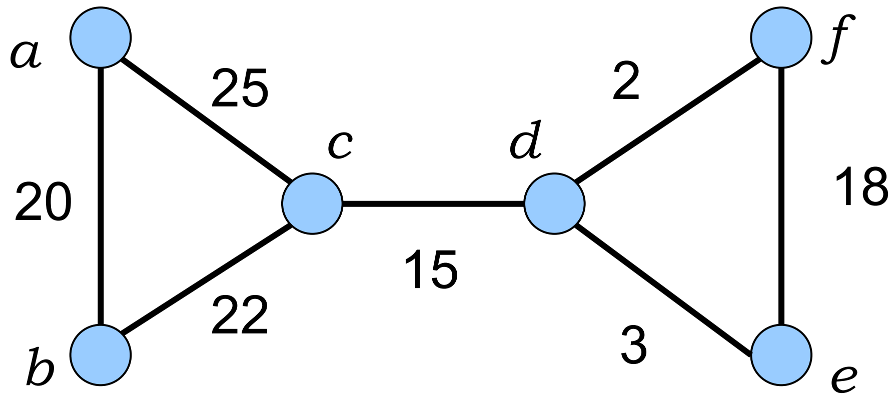
($s$, $c$): flow of value 15

($s$, $d$): flow of value 5

($s$, $e$): flow of value 5

Cut corresponding to the minimum flow ($s$, $e$): $S = \{s, a, b, c\}$

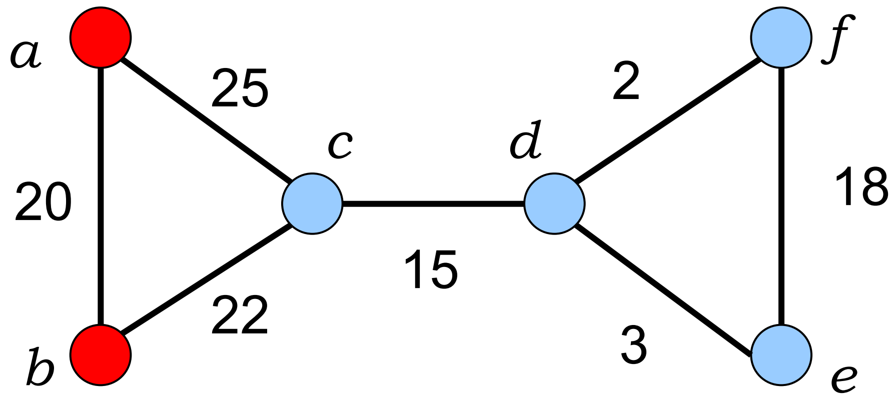
# Node identification



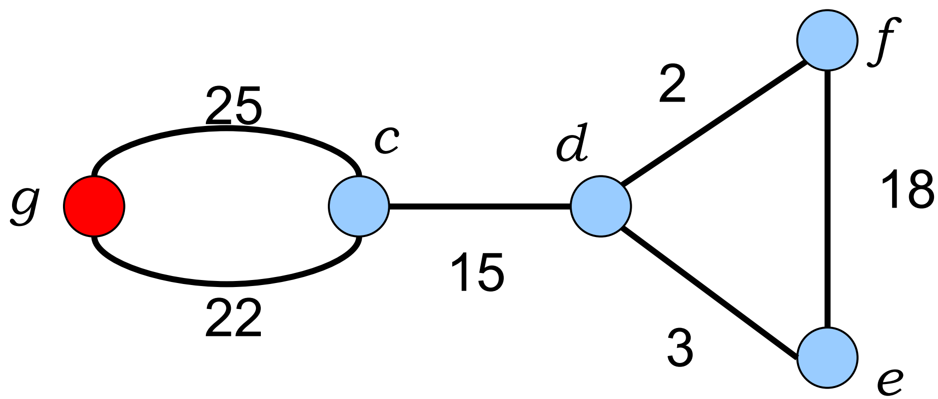

Pick two distinct vertices $u$ and $v$ and let $G_{uv}$ *be* is the graph that is obtained by:

1. $V(G_{uv}) = V \setminus \{u, v\} \cup \{x\}$
2. An arc $ij$ of $G$ remains in $G_{uv}$ if both $i$ and $j$ are distinct from $u$ and from $v$. If $j = u$ (or $j = v$), the arc $iu$ (or $iv$) becomes $ix$, where $i = u$ (or $i = v$), the arc $uj$ (or $vj$) becomes the arc $xj$.

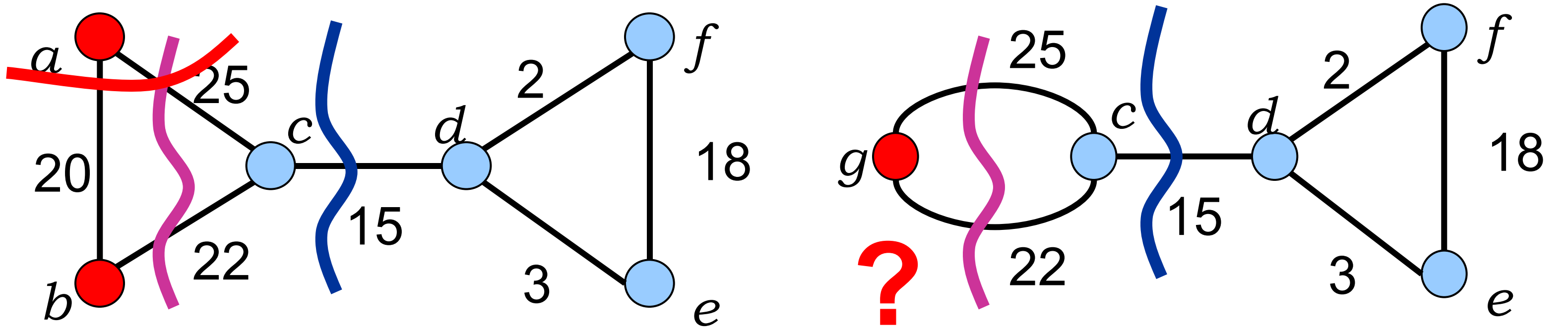
# Example




By identifying $a$ and $b$ a new vertex $g$ is generated:



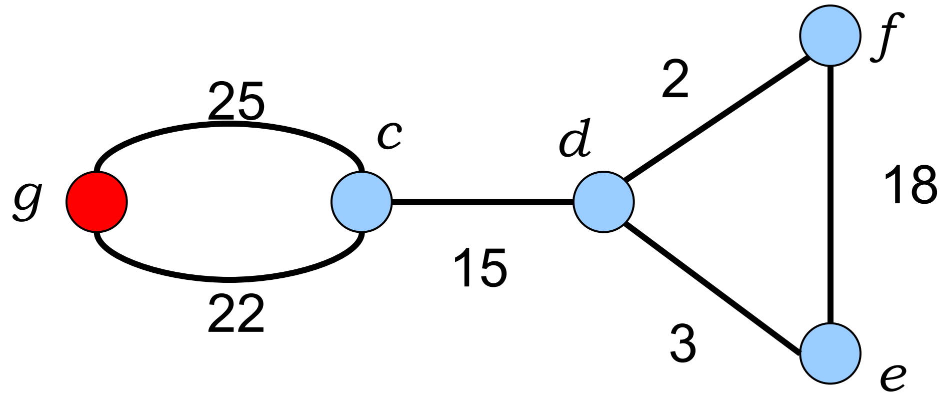
# Example




Observation

1. $G_{ab}$ is not a simple graph
2. A cut of $G_{ab}$ is a cut in $G$.
3. A cut of $G$ which does NOT separate $a$ and $b$, is a cut of $G_{ab}$

# Example

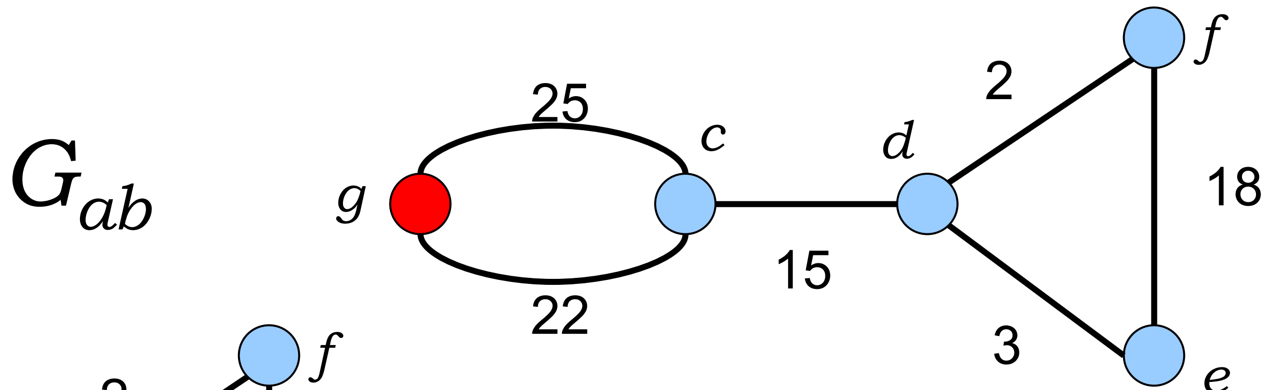


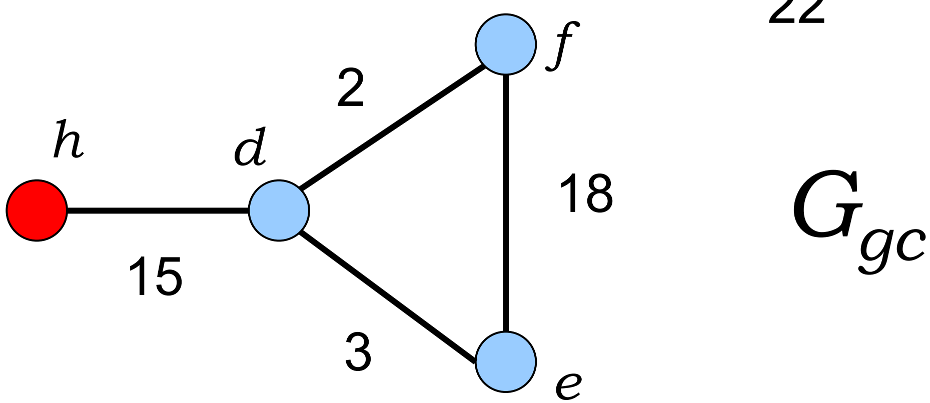

Consequence of oss. 2 and 3:

Let $\lambda(G)$ be the minimum cut in $G$, and $\lambda(G, a, b)$ an $(a,b)$-cut minimum (or, the minimum cut which separates $a$ and $b$). We have:

$$\lambda(G) = \min \{\lambda(G_{ab}), \lambda(G, a, b)\} = \min \{42, \lambda(G_{ab})\}$$

# Example

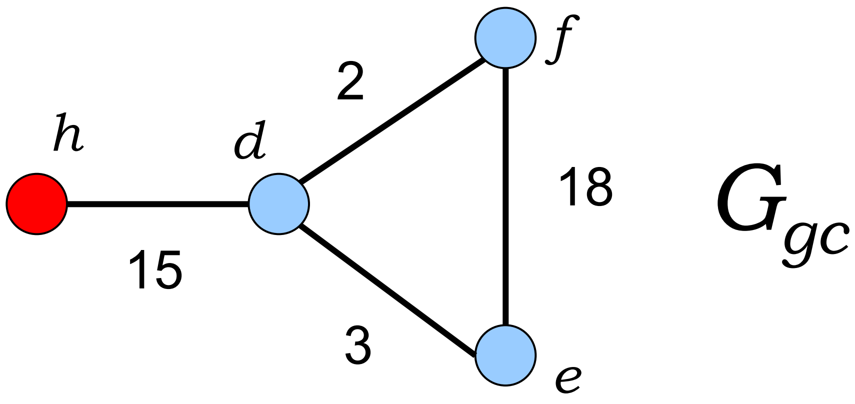


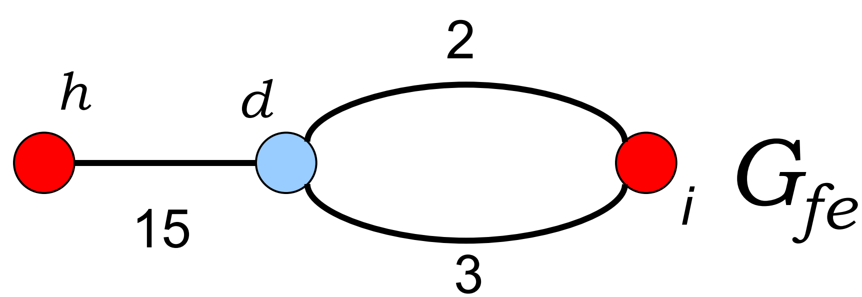




By identifying $g$ and $c$, we obtain $h$ and one has:

$\lambda(G) = \min\{42, \lambda(G_{ab})\} = \min\{42, \min\{\lambda(G_{gc}), \lambda(G, g, c)\}\} = \min\{42, 47, \lambda(G_{gc})\}$

# Example

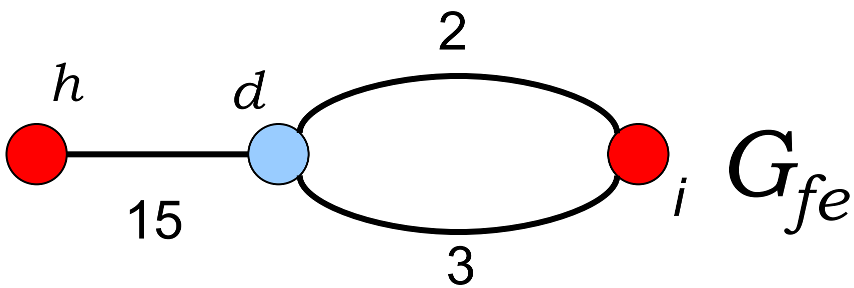


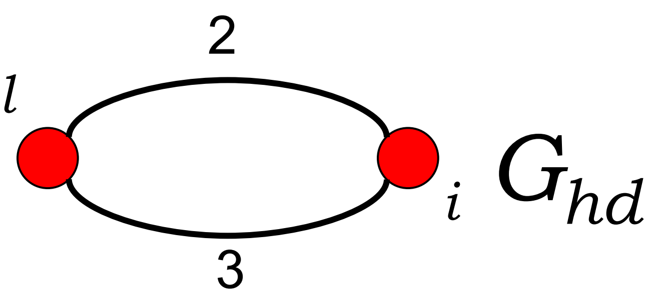




By identifying $f$ and $e$, we get $i$ and one has:

$\lambda(G) = \min \{42, 47, \lambda(G_{gc})\} = \min\{42, 47, 20, \lambda(G_{fe})\}$

# Example




Finally, by identifying $h$ and $d$, we get $l$ and we have:

$\lambda(G)$= min {42, 47, 20, 15, $\lambda(G_{hd})$} =

min{42, 47, 20, 15, 5}= 5.

# Example

Taking into account the cuts "lost" during the $n-1$ identifications, we developed an algorithm that has complexity $O(nk)$, where $k$ is the complexity of an algorithm for the max-flow.

So far, the only advantage of this approach is to solve max-flow decreasing in size.

Idea

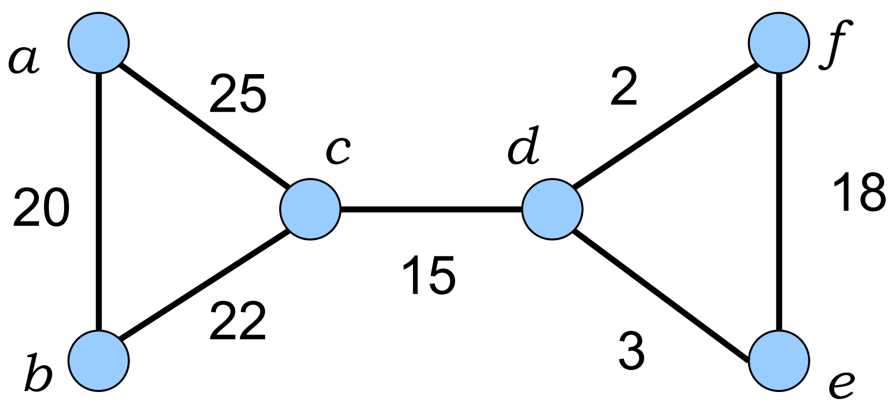
Choose the vertices to be identified so that it is easy to identify the minimum cut that separates them.

# Legal ordering

Let $v_1, v_2, \ldots, v_n$ an ordering of the vertices of $G$ and let $V_i = \{v_1, v_2, \ldots, v_i\}$. If

$$u(\delta(V_{i-1}) \cap \delta(v_i)) \geq u(\delta(V_{i-1}) \cap \delta(v_j)) \text{ per } 2 \leq i \leq j \leq n$$

we say that $v_1, v_2, \ldots, v_n$ is a “legal ordering”.




Example:
$a, c, b, d, e, f$

# Finding a "legal ordering"

**Inizialization**

Assign a label $e_i = 0$, for each $i \in V$.
Choose a node $u$ of $G$ and set $v_1 = u$, $V^{\text{ORD}} = \{v_1\}$, $k = 1$

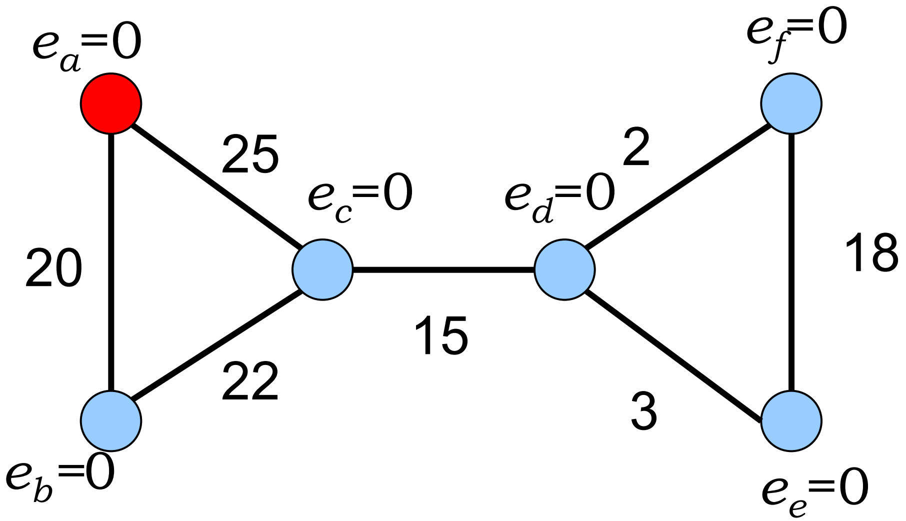
**Step $k$**

Updates the labels of the nodes adjacent to $v_k$,
by letting $e_i = e_i + u\,(iv_k)$, for each $i$ adjacent to $v_k$.
Select the node with maximum label among nodes belonging to $V^{\text{ORD}}$, say the node $v$.
Let $v_{k+1} = v$ and $V^{\text{ORD}} = V^{\text{ORD}} \cup \{v_{k+1}\}$, $k = k + 1$.

**Repeat until $k < n$**

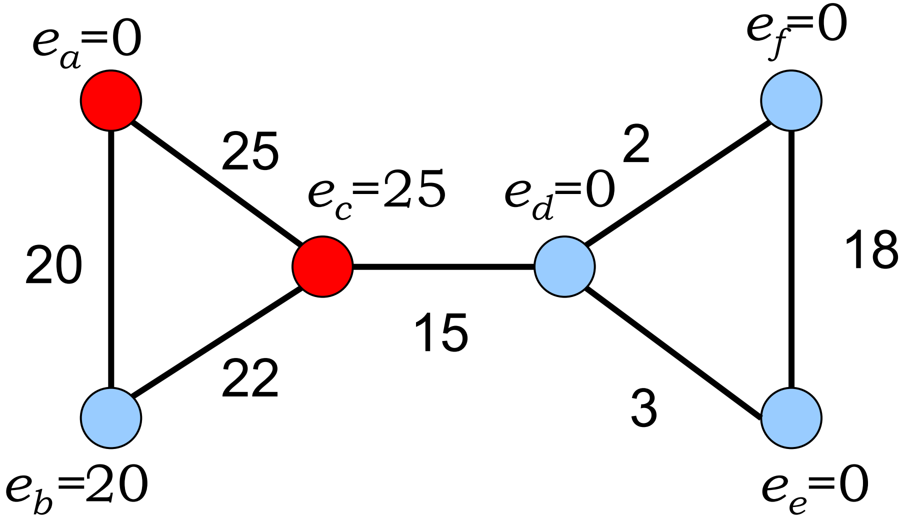
# Example

$V^{\mathrm{ORD}}=\{a\}$



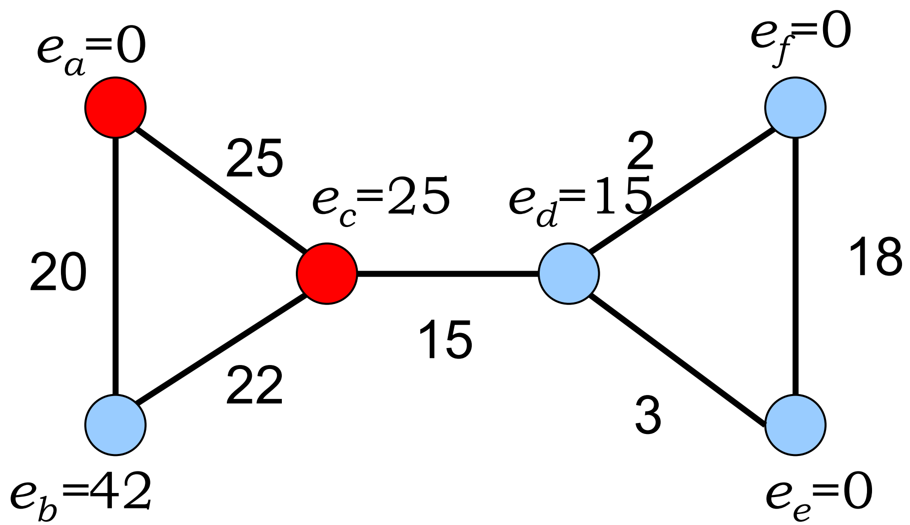
# Example

$V^{\mathrm{ORD}}=\{a\}$, $v_1=\{c\}$



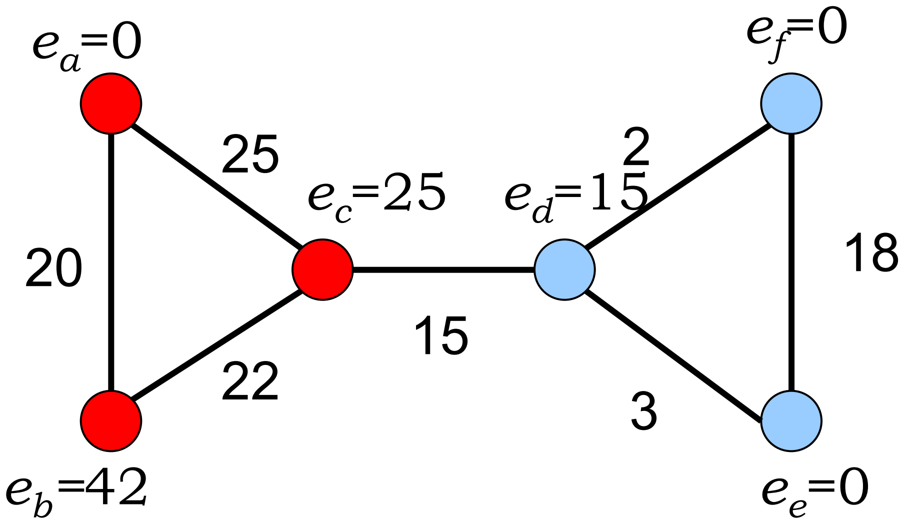
# Example

$V^{\mathrm{ORD}}=\{a, c\}$, $v_2=\{b\}$



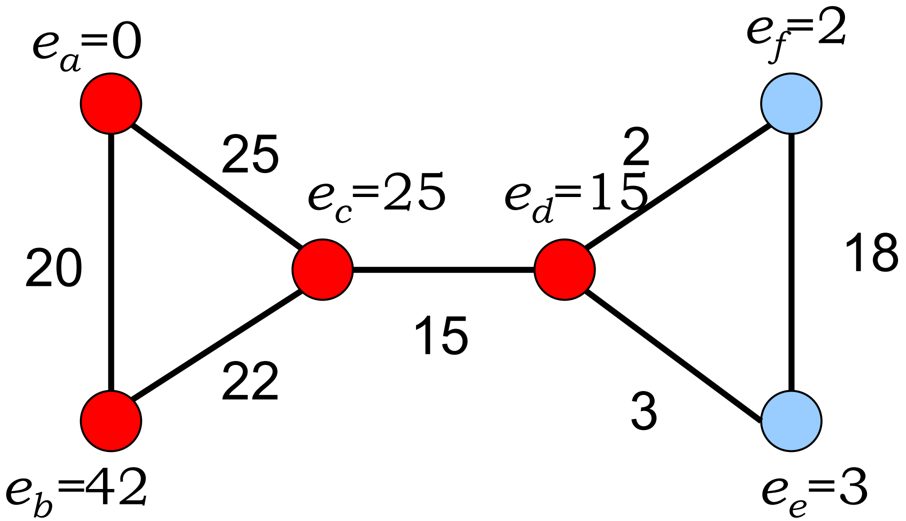
# Example

$V^{\mathrm{ORD}}=\{a, c, b\}$, $v_3=\{d\}$



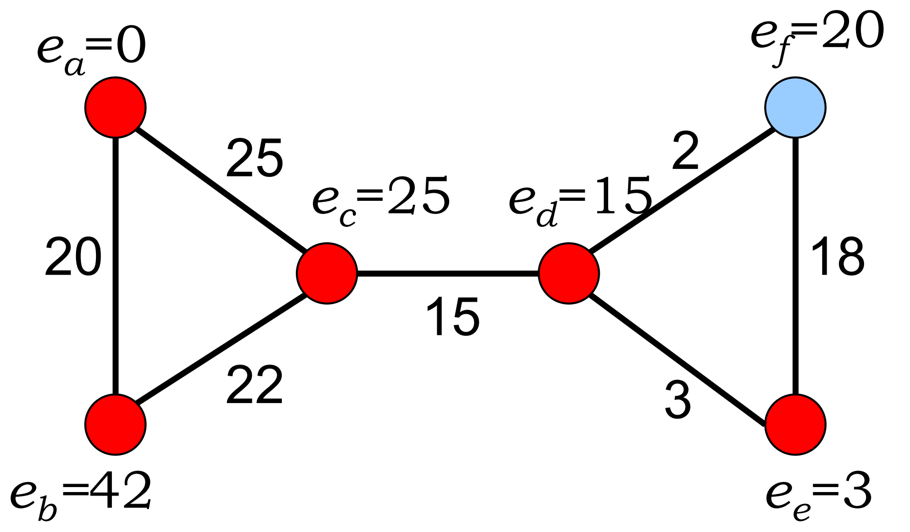
# Example

$V^{ORD}=\{a, c, b, d\}$, $v_3=\{e\}$



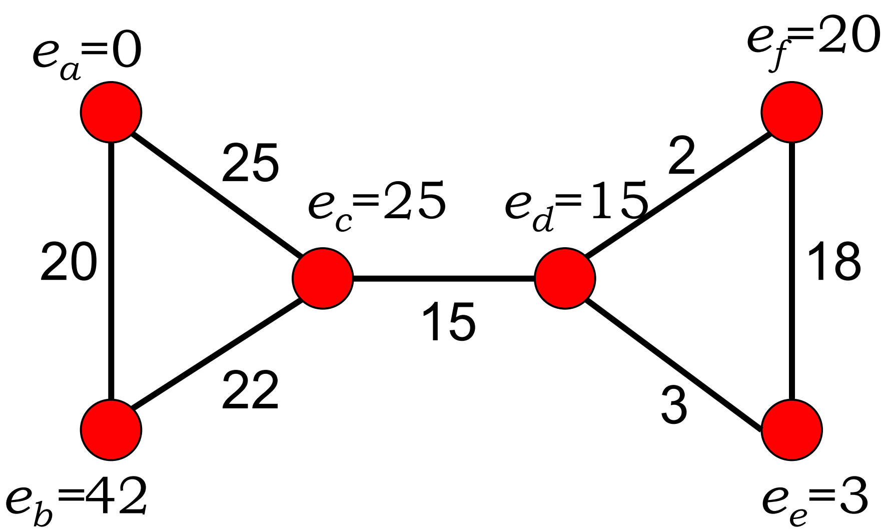
# Example

$V^{\mathrm{ORD}}=\{a, c, b, d, e\}$, $v_3=\{f\}$



# Example

$V^{\text{ORD}}=\{a, c, b, d, e, f\}$



# Properties

1. The algorithm described find a legal ordering in $O(n^2)$.
2. If $V^{\mathrm{ORD}}$ is a legal ordering, then $\delta(v_n)$ is the $(v_{n\text{-}1}, v_n)$-minimum cut of $G$

$$\lambda(G) = \min\left\{\lambda(G_{v_{n-1}v_n}), \lambda(G, v_{n-1}, v_n)\right\}$$

Recall that:

$$\lambda(G) = \min\left\{\lambda(G_{v_{n-1}v_n}), \delta(v_n)\right\}$$

Then, the following algorithm ends with a cut of minimum size of $G$ :

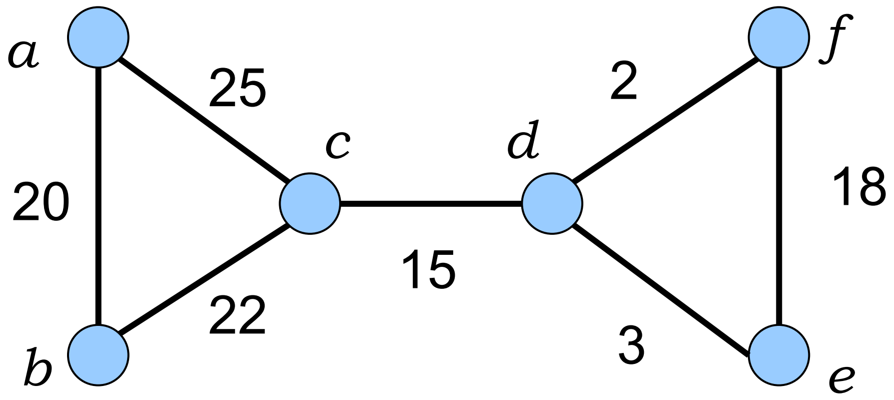
# Min Cut Algorithm

**initialization:** $M = +\infty$, $A = \varnothing$

**while** $G$ **has more than 2 nodes** $\{$
  **find a legal ordering of**
  $G : \{ v_1, v_2, \ldots, v_n \}$
  **if** $u(\delta(v_n)) < M$ **then** $M = u(\delta(v_n))$ **e** $A = \delta(v_n)$;
  **identifies** $v_{n\text{-}1}$ **e** $v_n$;
  **let** $G = G_{v_{n-1} v_n}$ **;**
$\}$ **endwhile;**

# Example (continued)

$V^{\mathrm{ORD}} = \{a, c, b, d, e, f\} \Rightarrow u(\delta(f)) = 20$
$M = 20$; $A = \{df, ef\}$

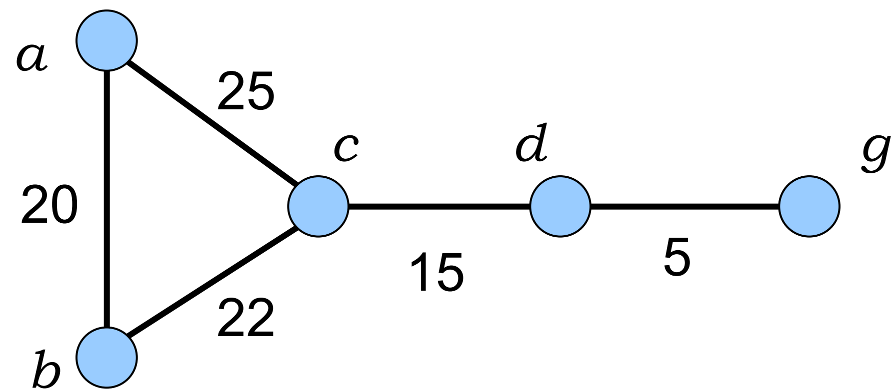


# Example (continued)

After identifying $f$ and $e$, we get $G_{ef}$ that admits the legal ordering:

$V^{\mathrm{ORD}}=\{g, d, c, a, b\} \Rightarrow u(\delta(b)) = 44$, So we don't update $M$ and $A$.

$M = 20;\ A = \{df, ef\}$

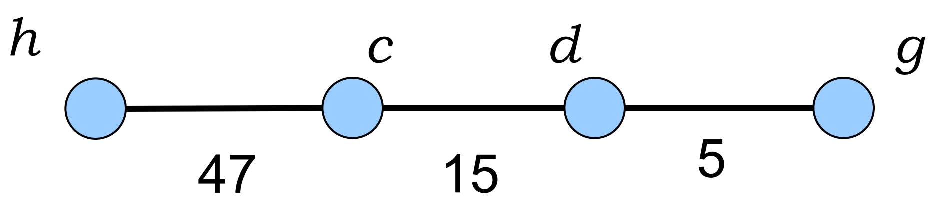


# Example (continued)

After identifying $a$ and $b$, we get $G_{ab}$ that admits the legal ordering:

$V^{\mathrm{ORD}}=\{h, c, d, g\} \Rightarrow u(\delta(g)) = 5$. Hence,

$M = 5$; $A = \{dg\} = \{df, de\}$




At this point, I can identify $g$ and $d$ and I can repeat the main step of the algorithm …

Observation

Min-cut algorithm has complexity $O(n^3)$.

# Theorem

**Theorem**

If $V^{\mathrm{ORD}}$ is a legal ordering, then $\delta(v_n)$ is the $(v_{n-1}, v_n)$-minimum cut of G.

**Lemma**

If $i$, $j$, $h$ are nodes of $V$, then

$\lambda(G, i, j) \geq \min \{\lambda(G, j, h), \lambda(G, i, h)\}$

**Proof**

Consider a min $(i,j)$-cut $\delta(S)$ and suppose $i \in S$. If $h \in S$, then $\delta(S)$ is also a $(j,h)$-cut and $u(\delta(S)) \geq \lambda(G, j, h)$.Otherwise, $\delta(S)$ is a $(i,h)$-cut e $u(\delta(S)) \geq \lambda(G, i, h)$ ■

# Proof

$\delta(v_n)$ is a $(v_{n-1}, v_n)$-cut.
We must show that it is minimum (i.e., that $u(\delta(v_n) \leq \lambda(G, v_{n-1}, v_n))$.

Proof by induction: if $n = 2$ the theorem is true.
Suppose that there exists the edge $e = v_{n-1}v_n$ in $G$ and let $G' = G \backslash e$.

The legal ordering $v_1, v_2, \ldots, v_n$ of $G$ s also a legal ordering of $G'$.
Then:
$u(\delta(v_n)) = u(\delta'(v_n)) + u_e$ and by the induction hypothesis,

$$\lambda(G', v_{n-1}, v_n) + u_e = \lambda(G, v_{n-1}, v_n).$$

# Proof

On the other hand, if $v_{n\text{-}1}$ and $v_n$ are not adjacent in $G$, we consider the node $v_{n\text{-}2}$ and we prove that:

1. $u(\delta(v_n)) \le \lambda(G, v_{n\text{-}2}, v_n)$
2. $u(\delta(v_n)) \le \lambda(G, v_{n\text{-}2}, v_{n\text{-}1})$.

Since from the previous lemma we have that

$$\lambda(G, v_{n\text{-}1}, v_n) \ge \min \{\lambda(G, v_{n\text{-}2}, v_n), \lambda(G, v_{n\text{-}2}, v_{n\text{-}1})\}$$
$$\ge u(\delta(v_n)),$$

If points 1 and 2 are true, the theorem is proved.

# Proof

Case 1

Consider $G' = G \backslash v_{n-1}$

The sequence $v_1, v_2, \ldots, v_{n-2}, v_n$ is a legal ordering of $G'$.

Now $u(\delta(v_n)) = u(\delta'(v_n))$ and by induction hypothesis

$u(\delta'(v_n)) = \lambda(G', v_{n-2}, v_n) \leq \lambda(G, v_{n-2}, v_n)$,

or $u(\delta(v_n)) \leq \lambda(G, v_{n-2}, v_n)$.

Case 2

Consider $G' = G \backslash v_n$

The sequence $v_1, v_2, \ldots, v_{n-1}$ is a legal ordering of $G'$.

By definition of legal ordering $u(\delta(v_n)) \leq u(\delta(v_{n-1}))$, but

$u(\delta(v_{n-1})) = u(\delta'(v_{n-1}))$ and from the induction hypothesis

$u(\delta'(v_{n-1})) = \lambda(G', v_{n-2}, v_{n-1}) \leq \lambda(G, v_{n-2}, v_{n-1})$,

i.e., $u(\delta(v_n)) \leq \lambda(G, v_{n-2}, v_{n-1})$. ■

# A probabilistic algorithm

```
while G has more than 2 nodes {
    choose an arc ij of G with probability
    u_ij / u(E);
    G = G_ij
}
```

The result of the algorithm is a cut of $G$.

# Theorem

Let $A$ be the minimum cut of $G$.. The algorithm of random contraction returns $A$ with probability $2/n\,(n\text{-}1)$

Proof

If the arcs of $A$ are not chosen during execution, then the algorithm returns exactly $A$.

Suppose you performed $i$ steps of the algorithm, and you contracted I arcs, none of which belongs to $A$. Let $G'=(V', E')$ the current graph. Obviously, $|V'| = n\text{-}i$.. Since A minimum cut of G, it is also the minimum cut of $G$ ‘.

The value of the minimum cut is at most equal to the average of the capacity cuts of type $\delta'(v)$, namely:

$$u(A) \le \sum_{v \in V'} u(\delta'(v)) /(n-i) = 2u(E')/(n-i)$$

# Demonstration

Therefore, the probability $p$ that an arc of A is chosen in step i +1 is:

$$\frac{u(A)}{u(E')} \leq \frac{2u(E')}{(n-i)u(E')} = \frac{2}{n-i}$$

The complementary probability (ie, that NO arc of A is chosen in step $i+1$) holds:

$$1 - \frac{2}{n-i} = \frac{(n-i-2)}{(n-i)}$$

# Demonstration

Therefore, the probability that during the performance of the algorithm is not chosen any arc of $A$ holds:

$$\frac{n-2}{n}\cdot\frac{n-3}{n-1}\cdot\frac{n-4}{n-2}\cdots\frac{3}{5}\cdot\frac{2}{4}=\frac{2}{n(n-1)}$$

■

# Corollario

Let $A$ be a minimum cut of $G$ and $k$ a positive integer. The probability that the algorithm of random contraction "are not returned in one of $kn^2$ executions is at most $e^{-2k}$.

Demonstration

$$\left(1-\frac{2}{n(n-1)}\right)^{kn^2} \leq \left(1-\frac{2}{n^2}\right)^{kn^2} \leq \left(e^{-\frac{2}{n^2}}\right)^{kn^2} = e^{-2k}$$

$$1-x \leq e^{-x}$$

■