Operational Research aspects in Routing: Network Flows

Paola Festa

Department of Mathematics and Applications "R. Caccioppoli"

University of Napoli FEDERICO II

http://www.dma.unina.it/∼festa/

E-mail: paola.festa@unina.it

Preliminaries: notation

Network flow problems are defined on special directed graphs.

Let $G = (V, A)$ be a **directed graph**, where

- ★ $V = \{1, 2, \ldots, n\}$ is a set of n nodes;
- ★ $A = \{(i, j) | \, i, j \in V\}$ is a set of m arcs;
- ★ for each node $i \in V$, let

$$FS(i) = \{j \in V \mid (i, j) \in A\}$$

  be the *forward star* of node i;
- ★ for each node $i \in V$, let

$$BS(i) = \{j \in V \mid (j, i) \in A\}$$

  be the *backward star* of node i.

Network Flow Problems

Network flow problems are important **combinatorial optimization problems** arising in any real-wold scenarios, whenever it is needed to organize and coordinate **distribution systems of one or more commodities/materials**

- ✔ gas, water;
- ✔ phone calls;
- ✔ e-mails, electronic information;
- ✔ ...

from **one or more source/distribution locations** to **one or more destinations/request locations**.

Network flow problems are **special *Linear Programming* problems**.
Therefore, they could be solved by **any linear programming method**, e.g. the **simplex method**.

Network flow problems are **special *Linear Programming* problems**.
Therefore, they could be solved by **any linear programming method**, e.g. the **simplex method**.

Nevertheless, they have **peculiar characteristics and properties** such that

- ☞ the methods for Linear Programming problems becomes "easier" when specialized to deal with them, but also
- ☞ those characteristics justify the design of efficient ad-hoc techniques.

A general network flow problem, ①

Network flow problems can be stated on **special di-graphs**, called **flow networks** and **whose elements** (nodes and arcs) have **numerical info** associated with.

Definition. A **network flow** is a **di-graph** $N = (V, A)$ whose arcs $(i, j) \in A$ are associated with the **following quantities**:

- a **cost** c_{ij} representing the **cost per unit of flow sent along arc** (i, j);
- a **capacity** $k_{ij} \geq 0$ representing an **upper bound on the quantity of flow** that can be **sent along arc** (i, j).

A general network flow problem, ①

Network flow problems can be stated on **special di-graphs**, called **flow networks** and **whose elements** (nodes and arcs) have **numerical info** associated with.

Definition. A **network flow** is a **di-graph** $N = (V, A)$ whose arcs $(i, j) \in A$ are associated with the **following quantities**:

- a **cost** c_{ij} representing the **cost per unit of flow sent along arc** (i, j);
- a **capacity** $k_{ij} \geq 0$ representing an **upper bound on the quantity of flow** that can be **sent along arc** (i, j).

Note:

If $k_{ij} = +\infty$, **along arc** (i, j) an **arbitrary quantity of flow** can be sent.

A general network flow problem, ②

Definition. A **network flow** is a **di-graph** $N = (V, A)$ whose arcs $(i, j) \in A$ are associated with the **following quantities**:

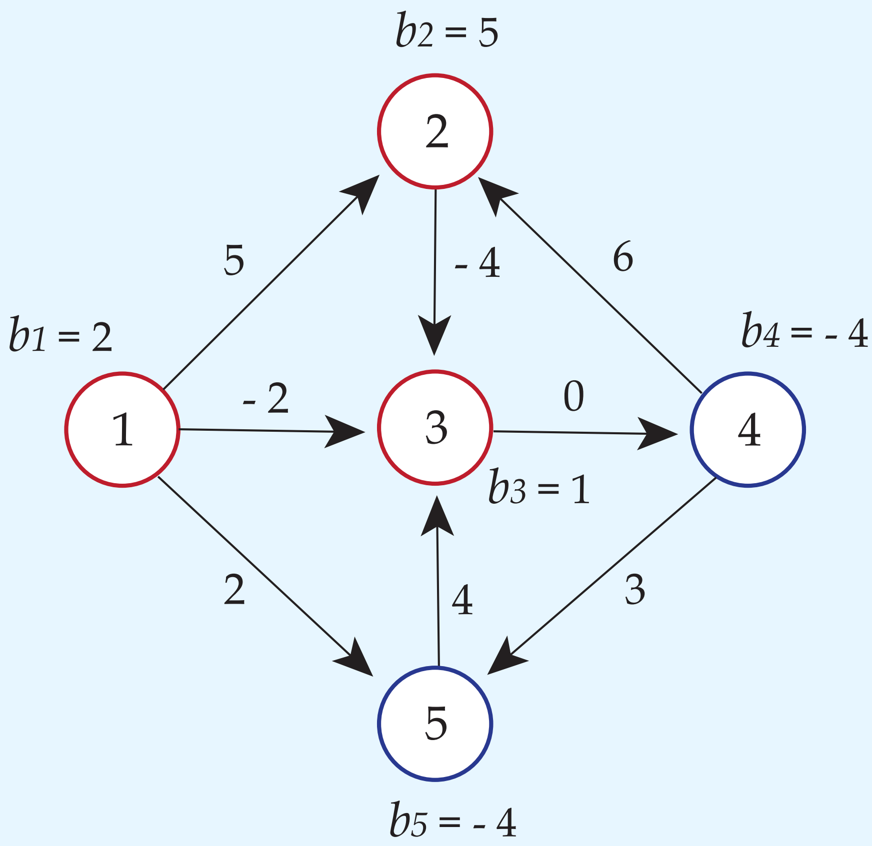
- a **cost** c_{ij} representing the **cost per unit of flow sent along arc** (i, j);
- a capacity $k_{ij} \geq 0$ representing an **upper bound on the quantity of flow** that can be **sent along arc** (i, j).

A **quantity** b_i can be associated with **each node** $i \in V$:

- if $b_i > 0$, b_i represents the **quantity of material entering** i from outside the network.
  b_i is called **supply** and i is called **source node**;
- if $b_i < 0$, $|b_i|$ represents the **quantity of material requested by** i.
  $|b_i|$ is called **demand** and i is called **sink node**;
- if $b_i = 0$, i is called **transit node**.

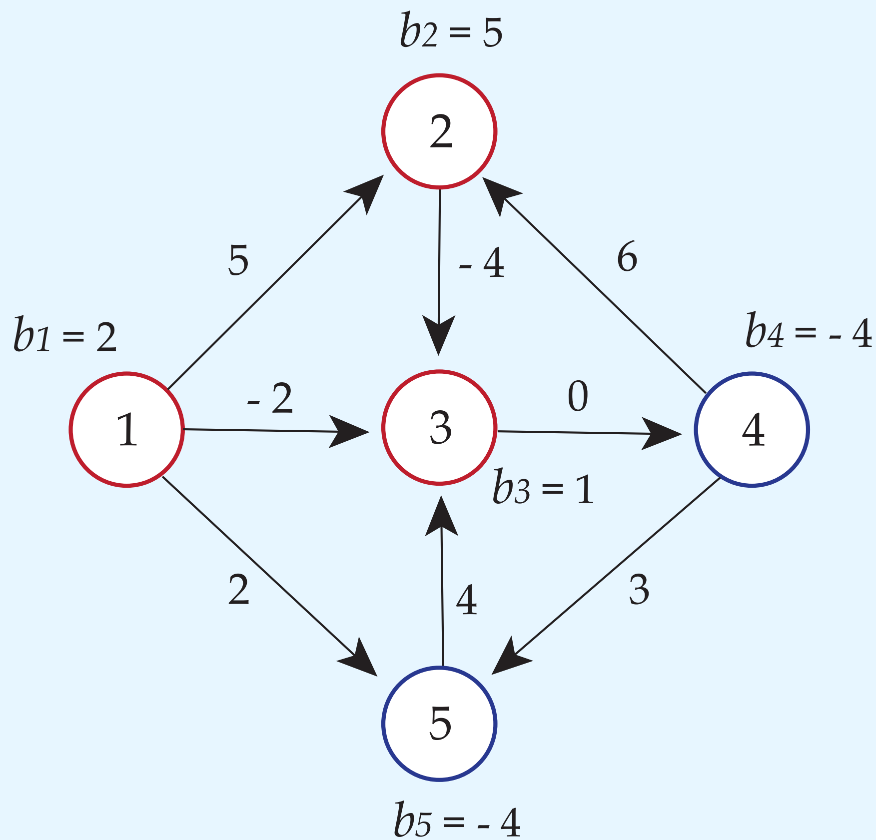
A general network flow problem, ③

Example. $V = \{1, 2, 3, 4, 5\}$; $|A| = 8$; $\quad k_{ij} = +\infty, \forall\, (i, j) \in A$:



A general network flow problem, ③

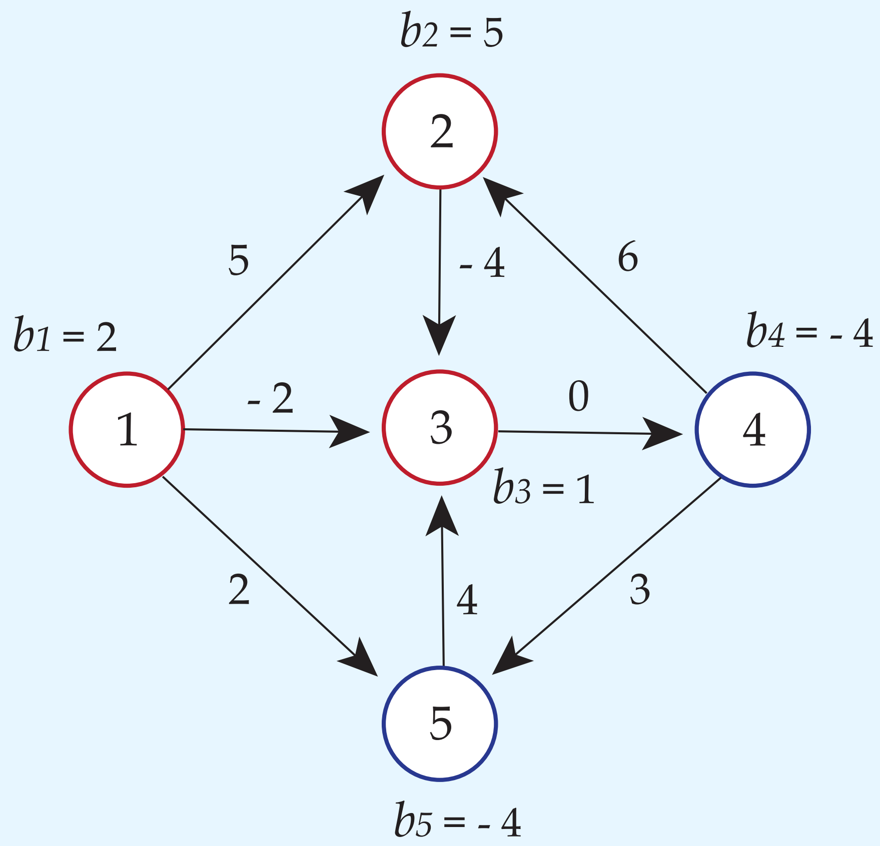
Example. $V = \{1,2,3,4,5\}$; $|A| = 8$; $\quad k_{ij} = +\infty, \forall\,(i,j) \in A$:



Source nodes: 1, 2, and 3;

A general network flow problem, ③

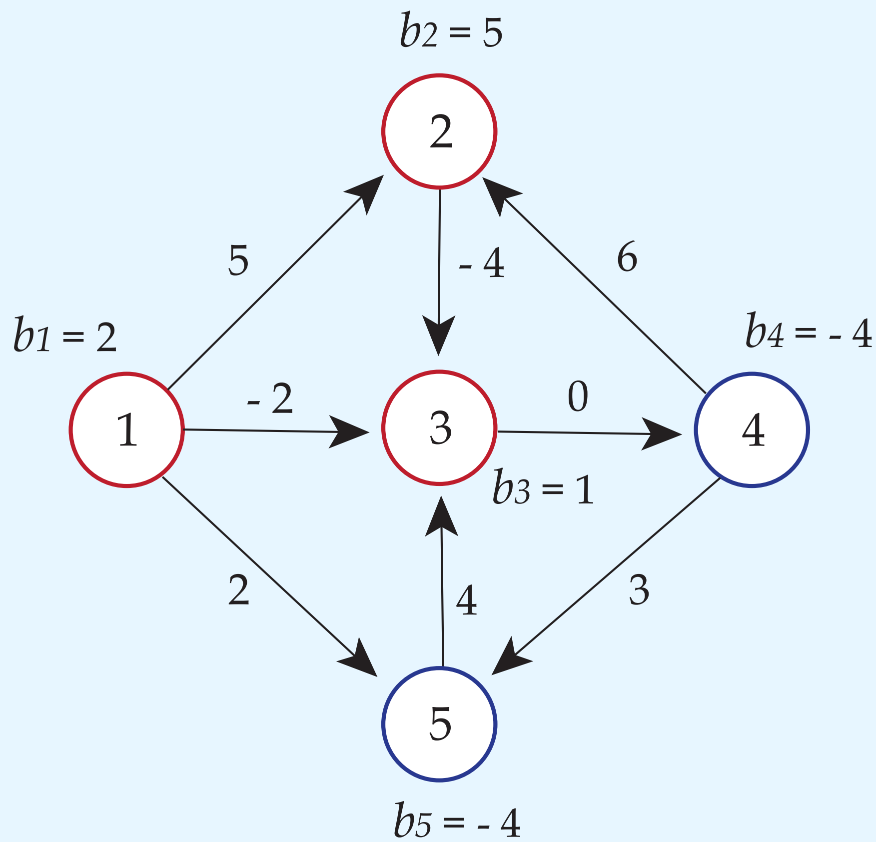
Example. $V = \{1, 2, 3, 4, 5\}$; $|A| = 8$; $k_{ij} = +\infty, \forall\,(i, j) \in A$:



Source nodes: 1, 2, and 3; **Sink nodes**: 4 and 5.

A general network flow problem, ③

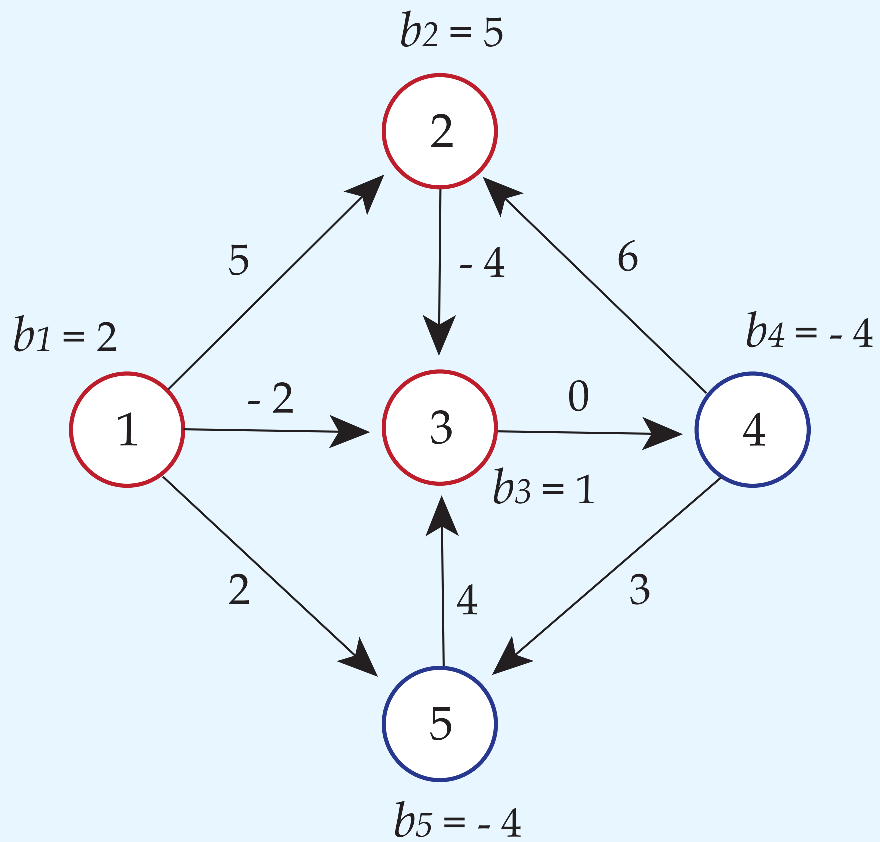
Example. $V = \{1, 2, 3, 4, 5\}$; $|A| = 8$; $k_{ij} = +\infty, \forall\, (i, j) \in A$:



Source nodes: 1, 2, and 3; **Sink nodes**: 4 and 5. **No transit nodes**.

A general network flow problem, ③

Example. $V = \{1, 2, 3, 4, 5\}$; $|A| = 8$; $\quad k_{ij} = +\infty, \forall\, (i, j) \in A$:



Source nodes: 1, 2, and 3; **Sink nodes**: 4 and 5. **No transit nodes**.

Note: $\sum_{i \in V} b_i = 0$.

A general network flow problem, ④

Definition. Given a network flow $N = (V, A)$, a **feasible flow** is an **vector of flow variables**

$$\{x_{ij}\}_{(i,j)\in A}, \quad x_{ij} \in \mathbb{R}, \ \forall\,(i,j) \in A$$

such that

① $l_{ij}(=0) \leq x_{ij} \leq k_{ij}, \forall\,(i,j) \in A$;

② $b_i + \sum_{(j,i)\in A} x_{ji} = \sum_{(i,j)\in A} x_{ij}, \forall\, i \in V.$

.

A general network flow problem, ④

Definition. Given a network flow $N = (V, A)$, a **feasible flow** is an **vector of flow variables**

$$\{x_{ij}\}_{(i,j)\in A}, \quad x_{ij} \in \mathbb{R}, \ \forall\,(i,j) \in A$$

such that

① $l_{ij}(=0) \leq x_{ij} \leq k_{ij}, \forall\,(i,j) \in A;$

② $b_i + \sum_{(j,i)\in A} x_{ji} = \sum_{(i,j)\in A} x_{ij}, \forall\, i \in V.$

Conditions ① are easy to understand.

A general network flow problem, ④

Definition. Given a network flow $N = (V, A)$, a **feasible flow** is an **vector of flow variables**

$$\{x_{ij}\}_{(i,j)\in A}, \quad x_{ij} \in \mathbb{R}, \ \forall\,(i,j) \in A$$

such that

① $l_{ij}(=0) \leq x_{ij} \leq k_{ij}, \forall\,(i,j) \in A;$

② $b_i + \sum_{(j,i)\in A} x_{ji} = \sum_{(i,j)\in A} x_{ij}, \forall\, i \in V.$

Conditions ① are easy to understand.

Conditions ② are known as **equation or flow conservation law**.

A general network flow problem, ④

Definition. Given a network flow $N = (V, A)$, a **feasible flow** is an **vector of flow variables**

$$\{x_{ij}\}_{(i,j)\in A}, \quad x_{ij} \in \mathbb{R}, \ \forall\,(i,j) \in A$$

such that

① $l_{ij}(= 0) \leq x_{ij} \leq k_{ij}, \forall\,(i,j) \in A$;

② $b_i + \sum_{(j,i)\in A} x_{ji} = \sum_{(i,j)\in A} x_{ij}, \forall\, i \in V$.

Note: adding all over nodes $i \in V$ both sides of ②, it results that

$$\sum_{i\in V} b_i = 0. \qquad \text{(Principle of the total divergence)}$$

The vector $\{b_i\}_{i\in V}$ is called divergence vector.

Minimum Cost Flow Problem, ①

Definition. The general **minimum cost network flow problem** consists in **minimizing a linear cost function of the form**

$$\min \sum_{(i,j)\in A} c_{ij}x_{ij},$$

where the vector $\{x_{ij}\}_{(i,j)\in A}$ is a **feasible flow**.

Paola Festa - DMA, University of Napoli FEDERICO II
E-mail: paola.festa@unina.it - Web: http://www.dma.unina.it/∼festa

Minimum Cost Flow Problem, ①

Definition. The general **minimum cost network flow problem** consists in **minimizing a linear cost function of the form**

$$\min \sum_{(i,j)\in A} c_{ij}x_{ij},$$

where the vector $\{x_{ij}\}_{(i,j)\in A}$ is a **feasible flow**.

$$\text{(MF)} \quad \min \sum_{(i,j)\in A} c_{ij}x_{ij}$$

s.t.

$$\text{(a)} \quad \sum_{(i,j)\in A} x_{ij} - \left(b_i + \sum_{(j,i)\in A} x_{ji} \right) = 0, \quad \forall\, i \in V$$

$$\text{(b)} \quad l_{ij} \le x_{ij} \le k_{ij},\ \forall (i,j) \in A.$$

Paola Festa - DMA, University of Napoli FEDERICO II
E-mail: paola.festa@unina.it - Web: http://www.dma.unina.it/∼festa

Minimum Cost Flow Problem, ②

The general **minimum cost network flow problem**:

$$\text{(MF)} \quad \min \sum_{(i,j)\in A} c_{ij}x_{ij}$$

s.t.

$$\text{(a)} \quad \sum_{(i,j)\in A} x_{ij} - \left(b_i + \sum_{(j,i)\in A} x_{ji} \right) = 0, \quad \forall\, i \in V$$

$$\text{(b)} \quad l_{ij} \le x_{ij} \le k_{ij}, \ \forall (i,j) \in A.$$

It is evident that it is a **linear programming problem**.

Minimum Cost Flow Problem, ②

The general **minimum cost network flow problem**:

$$(MF) \quad \min \sum_{(i,j)\in A} c_{ij}x_{ij}$$

s.t.

$$(a) \quad \sum_{(i,j)\in A} x_{ij} - \left(b_i + \sum_{(j,i)\in A} x_{ji} \right) = 0, \quad \forall\, i \in V$$

$$(b) \quad l_{ij} \le x_{ij} \le k_{ij}, \ \forall (i,j) \in A.$$

It is evident that it is a **linear programming problem**.

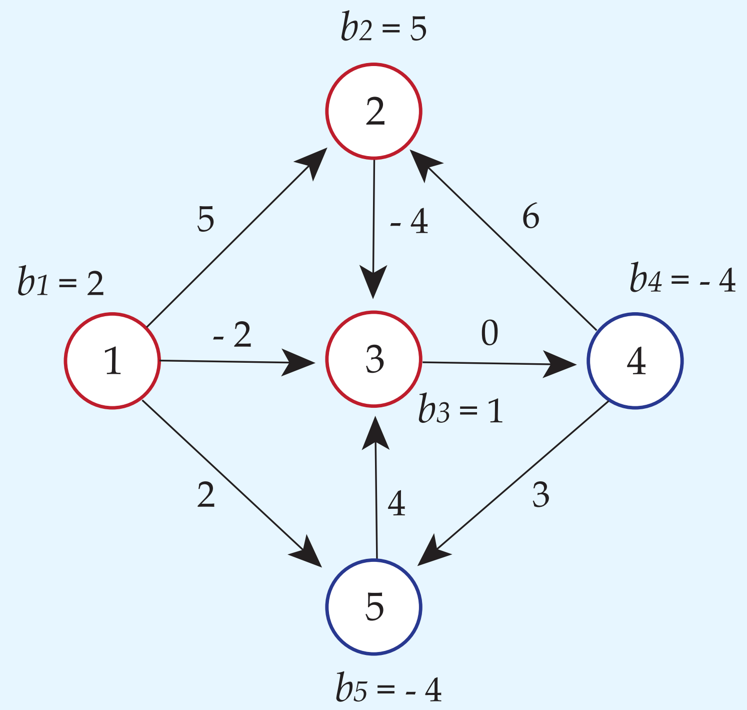
If

$$k_{ij} = +\infty, \qquad \forall\, (i,j) \in A,$$

the problem is said **uncapacitated** and it is **in standard form**.

Minimum Cost Flow Problem, ②

Example: $N = \{1,2,3,4,5\}; |A| = 8; \quad k_{ij} = +\infty, \forall\,(i,j) \in A$:



$$\begin{aligned}
\min \quad & 5x_{12} - 4x_{23} + 6x_{42} - 2x_{13} + 0x_{34} + 2x_{15} + 4x_{53} + 3x_{45} \\
& x_{12} + x_{13} + x_{15} = 2, \quad x_{23} - x_{12} - x_{42} = 5, \\
& x_{34} - x_{13} - x_{23} - x_{53} = 1, \quad x_{42} + x_{45} - x_{34} = -4 \\
& x_{53} - x_{15} - x_{45} = -4, \quad \bar{x} \geq 0.
\end{aligned}$$

Minimum Cost Flow: alternative form., ①

Alternative and concise math formulation: by using the more economical **matrix-vector notation** of the network flow $N = (V, A)$.

Minimum Cost Flow: alternative form., ①

Alternative and concise math formulation: by using the more economical **matrix-vector notation** of the network flow $N = (V, A)$.

Let $N = (V, A)$ be a network flow, where

- ✔ $V = \{1, \ldots, n\}$ and
- ✔ $|A| = m$,

and let $D \in \{-1, 0, 1\}^{n \times m}$ be the associated **node-arc incidence matrix** s.t.

$$d_{ik} = \begin{cases} -1, & \text{if } i \text{ is the tail (start node) of the } k\text{.th arc;} \\ 1, & \text{if } i \text{ is the head (end node) of the } k\text{.th arc;} \\ 0, & \text{otherwise.} \end{cases}$$

Minimum Cost Flow: alternative form., ②

In matrix form:

$$D = \begin{array}{rc} & \begin{array}{ccccc} \overbrace{}^{1} & & \overbrace{}^{k} & & \overbrace{}^{m} \end{array} \\ \begin{array}{r} \\ \text{row } i \rightarrow \\ \\ \\ \text{row } n \rightarrow \end{array} & \left(\begin{array}{ccccc} \vdots & \cdots & \vdots & \cdots & \vdots \\ 1 & \cdots & -1 & \cdots & \vdots \\ \vdots & \cdots & \cdots & \cdots & -1 \\ \vdots & \cdots & 1 & \cdots & 1 \\ -1 & \cdots & \cdots & \cdots & \vdots \end{array}\right) \end{array}$$

Minimum Cost Flow: alternative form., ②

In matrix form:

$$D = \begin{array}{r} \\ \\ \text{row } i \rightarrow \\ \\ \\ \text{row } n \rightarrow \end{array} \left(\begin{array}{ccccc} \overbrace{}^{1} & & \overbrace{}^{k} & & \overbrace{}^{m} \\ \vdots & \cdots & \vdots & \cdots & \vdots \\ 1 & \cdots & -1 & \cdots & \vdots \\ \vdots & \cdots & \cdots & \cdots & -1 \\ \vdots & \cdots & 1 & \cdots & 1 \\ -1 & \cdots & \cdots & \cdots & \vdots \end{array} \right)$$

Notes: **for each row d_i', $i = 1, \ldots, n$,**

① the **nr of "1" in d_i'** is $|\{(i,j) \in A\}| = |FS(i)|$;

Minimum Cost Flow: alternative form., ②

In matrix form:

$$D = \begin{array}{r} \\ \\ \text{row } i \to \\ \\ \\ \text{row } n \to \end{array} \begin{array}{c} \begin{array}{ccccc} \overbrace{}^{1} & & \overbrace{}^{k} & & \overbrace{}^{m} \end{array} \\ \begin{pmatrix} \vdots & \cdots & \vdots & \cdots & \vdots \\ 1 & \cdots & -1 & \cdots & \vdots \\ \vdots & \cdots & \cdots & \cdots & -1 \\ \vdots & \cdots & 1 & \cdots & 1 \\ -1 & \cdots & \cdots & \cdots & \vdots \end{pmatrix} \end{array}$$

Notes: **for each row** d_i', $i = 1, \ldots, n$,

② the **nr of "-1" in** d_i' is $|\{(j,i) \in A\}| = |BS(i)|$;

Minimum Cost Flow: alternative form., ②

In matrix form:

$$D = \begin{array}{r} \\ \\ \text{row } i \rightarrow \\ \\ \\ \text{row } n \rightarrow \end{array} \left(\begin{array}{ccccc} \overbrace{}^{1} & & \overbrace{}^{k} & & \overbrace{}^{m} \\ \vdots & \cdots & \vdots & \cdots & \vdots \\ 1 & \cdots & -1 & \cdots & \vdots \\ \vdots & \cdots & \cdots & \cdots & -1 \\ \vdots & \cdots & 1 & \cdots & 1 \\ -1 & \cdots & \cdots & \cdots & \vdots \end{array} \right)$$

Notes: **for each row d_i', $i = 1, \ldots, n$,**

③ $$\sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = d_i' x;$$

Minimum Cost Flow: alternative form., ②

In matrix form:

$$D = \begin{array}{r@{}l} & \begin{array}{ccccc} \overbrace{\quad}^{1} & & \overbrace{\quad}^{k} & & \overbrace{\quad}^{m} \end{array} \\ \begin{array}{r} \\ \text{row } i \rightarrow \\ \\ \\ \text{row } n \rightarrow \end{array} & \begin{pmatrix} \vdots & \cdots & \vdots & \cdots & \vdots \\ 1 & \cdots & -1 & \cdots & \vdots \\ \vdots & \cdots & \cdots & \cdots & -1 \\ \vdots & \cdots & 1 & \cdots & 1 \\ -1 & \cdots & \cdots & \cdots & \vdots \end{pmatrix} \end{array}$$

Notes: **for each row** d'_i, $i = 1, \ldots, n$

④ the **flow conservation law** can be **rewritten as** $d'_i x = b_i, \quad \forall\, i \in V$ or, in **more compact form**

$$Dx = b, \qquad b \text{ divergence vector.}$$

Minimum Cost Flow: alternative form., ②

In matrix form:

$$D = \begin{array}{r} \\ \\ \text{row } i \rightarrow \\ \\ \\ \text{row } n \rightarrow \end{array} \left(\begin{array}{ccccc} \overbrace{}^{1} & & \overbrace{}^{k} & & \overbrace{}^{m} \\ \vdots & \cdots & \vdots & \cdots & \vdots \\ 1 & \cdots & -1 & \cdots & \vdots \\ \vdots & \cdots & \cdots & \cdots & -1 \\ \vdots & \cdots & 1 & \cdots & 1 \\ -1 & \cdots & \cdots & \cdots & \vdots \end{array} \right)$$

Notes: **for each row** d'_i, $i = 1, \ldots, n$

⑤ adding all rows d'_i results $\bar{0}$, i.e., the rows of D are l.d.
Nevertheless, it is always **possible to remove the reduntant constraints without changing the feasible region of the problem**.

Minimum Cost Flow: alternative form., ③

Notes $\Longrightarrow$ **alternative and concise math formulation**:

$$
\begin{array}{lll}
\text{(MF')} & \min & \displaystyle\sum_{(i,j)\in A} c_{ij}x_{ij} \\
& \text{s.t.} & \\
& \text{(a')} & Dx = b \\
& \text{(b)} & l_{ij} \le x_{ij} \le k_{ij}, \quad \forall (i,j) \in A.
\end{array}
$$

Minimum Cost Flow: alternative form., ③

Notes $\Longrightarrow$ **alternative and concise math formulation**:

$$
\begin{array}{lll}
\text{(MF')} & \min & \displaystyle\sum_{(i,j)\in A} c_{ij}x_{ij} \\
& \text{s.t.} & \\
& \text{(a')} & Dx = b \\
& \text{(b)} & l_{ij} \le x_{ij} \le k_{ij}, \quad \forall (i,j) \in A.
\end{array}
$$

Further notes:

❶ each network flow problem is a linear programming problem;

❷ the constraint matrix D of the math formulation is unimodular.

Then, **it has an integer optimal solution if**

$$l_{ij},\, k_{ij} \in \mathbb{Z}^+ \cup \{0\},\ \forall\, (i,j) \in A.$$

Circulation

Definition. **Any flow vector** (feasible or infeasible) that **satisfies**

$$Dx = 0$$

is called **circulation** (in this case, it results that $b = 0$).

Definition. **Any flow vector** (feasible or infeasible) that **satisfies**

$$Dx = 0$$

is called **circulation** (in this case, it results that $b = 0$).

The **flow conservation law** (also known as **Kirchkoff's equation**) is **imposed only within the network**, without external supply or demand.

In other words, the **flow "circulates" only inside the network**.

Network Flows: Variants

There are **several variants of network flow problems**, **all of which** can be shown to be **equivalent to each other**.

We will discuss now **some examples**.

Variant ①:

Every network flow problem can be reduced to **one with exactly one source s and exactly one sink node d**, $s, d \in V$, $s \neq d$.

Variant ①:
Every network flow problem can be reduced to **one with exactly one source s and exactly one sink node d**, $s, d \in V$, $s \neq d$.

Let us suppose that $N = (V, A)$ has k source nodes $s_1, \ldots, s_k$ and h sink nodes $d_1, \ldots, d_h$.

Variant ①:
Every network flow problem can be reduced to **one with exactly one source s and exactly one sink node d**, $s, d \in V$, $s \neq d$.

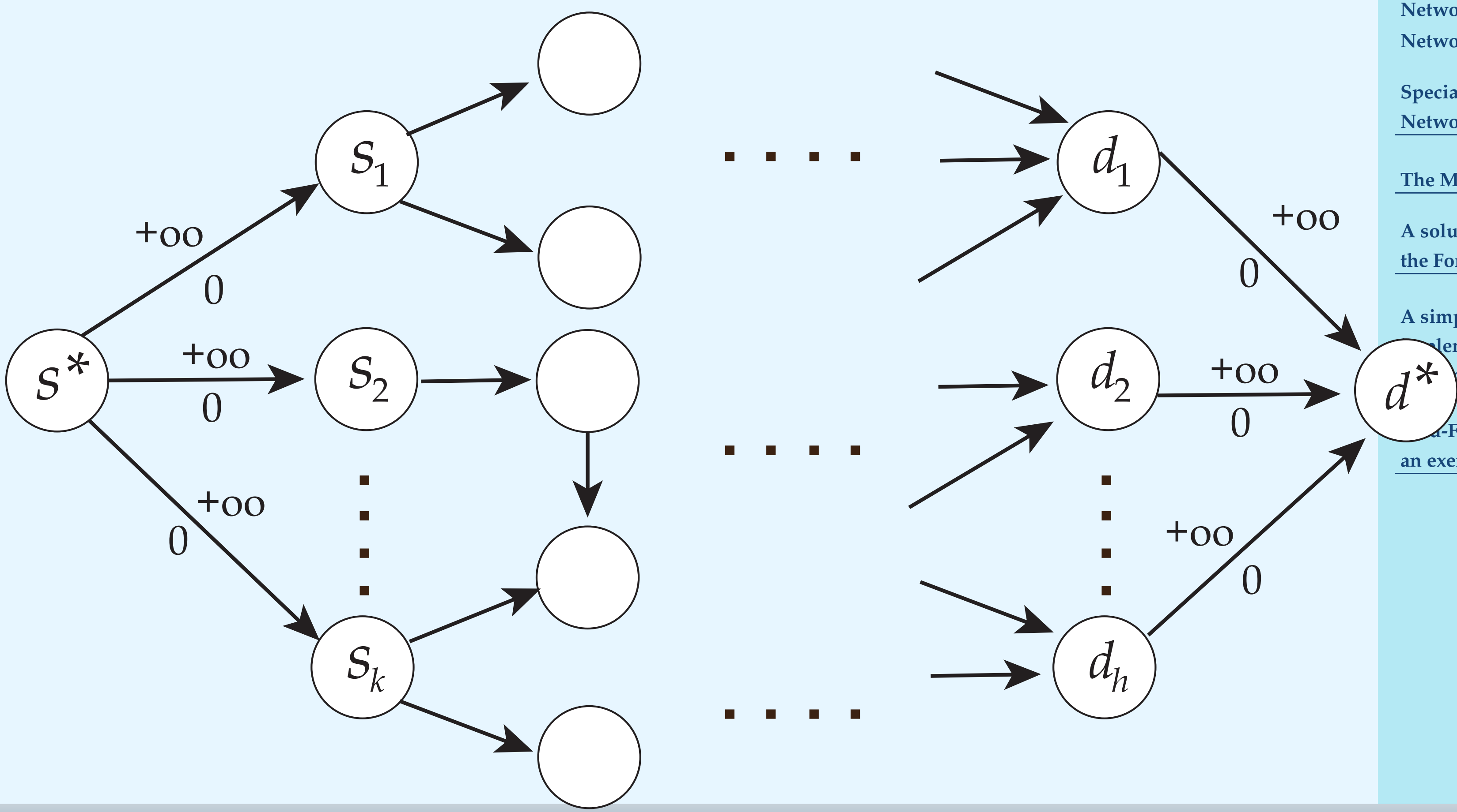
Let us suppose that $N = (V, A)$ has k source nodes $s_1, \ldots, s_k$ and h sink nodes $d_1, \ldots, d_h$.

It is **always possible to define**

- ✍ a **dummy source** s^* and a **dummy sink** d^*;
- ✍ k **dummy arcs** (s^*, s_q) $(q = 1, \ldots, k)$;
- ✍ h **dummy arcs** (d_l, d^*) $(l = 1, \ldots, h)$ s.t.

$$\forall\, q = 1, \ldots, k, \begin{cases} k_{s^* s_q} = +\infty; \\ c_{s^* s_q} = 0. \end{cases} \quad \forall\, l = 1, \ldots, h, \begin{cases} k_{d_l d^*} = +\infty; \\ c_{d_l d^*} = 0. \end{cases}$$

Variant ①:
Every network flow problem can be reduced to **one with exactly one source s and exactly one sink node d**, $s, d \in V$, $s \neq d$.



Variant ②:
Every network flow problem can be reduced to **one without sources or sinks** ($b_i = 0, \forall\, i \in V$, **circulation problems**).

Variant ②:
Every network flow problem can be reduced to **one without sources or sinks** ($b_i = 0, \forall\, i \in V$, **circulation problems**).

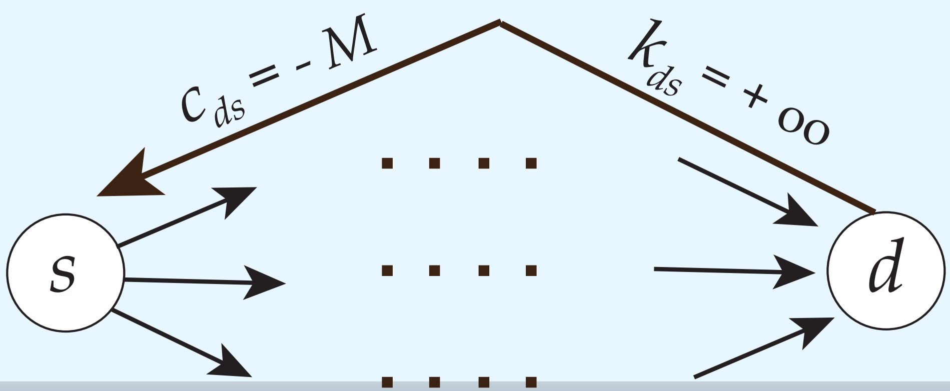
Without loss of generality, consider a network $N = (V, A)$ with **a single source node** s and **a single sink node** d.

Variant ②:
Every network flow problem can be reduced to **one without sources or sinks** ($b_i = 0, \forall\, i \in V$, **circulation problems**).

Without loss of generality, consider a network $N = (V, A)$ with **a single source node** s and **a single sink node** d.

It is **always possible to define** a **dummy arc** (d, s) s.t.

$k_{ds} = +\infty; \quad c_{ds} = -M, \quad$ where M is a sufficiently large number.

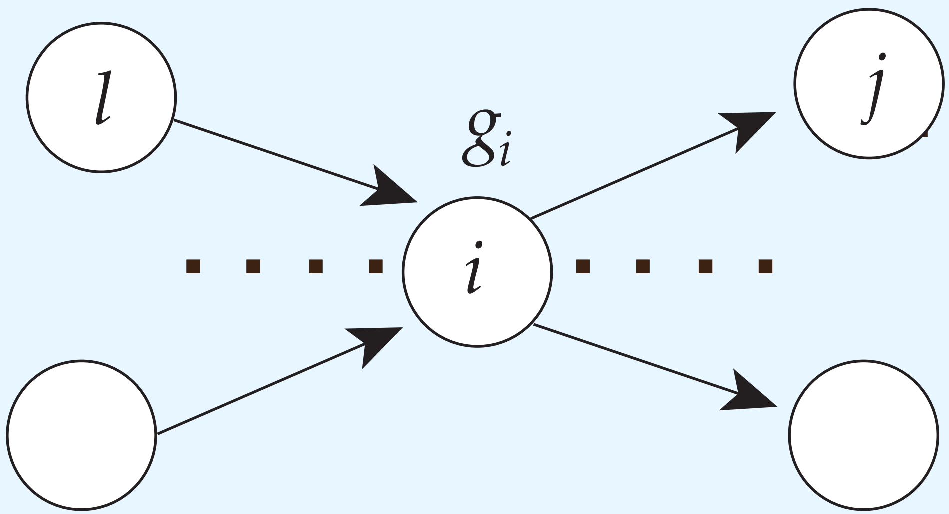


Variant ③:
Every network flow problem with *node capacities*
(**upper bound on the flow that can enter in each node**)
can be reduced to **one with only arc capacities**.

Network Flows: Variants

Variant ③:
Every network flow problem with *node capacities* (**upper bound on the flow that can enter in each node**) can be reduced to **one with only arc capacities**.

Suppose that in $N = (V, A)$ there is a **source node** i with **supply** b_i and **capacity** g_i, i.e., $b_i + \sum_{(l,i) \in A} x_{li} \le g_i$.

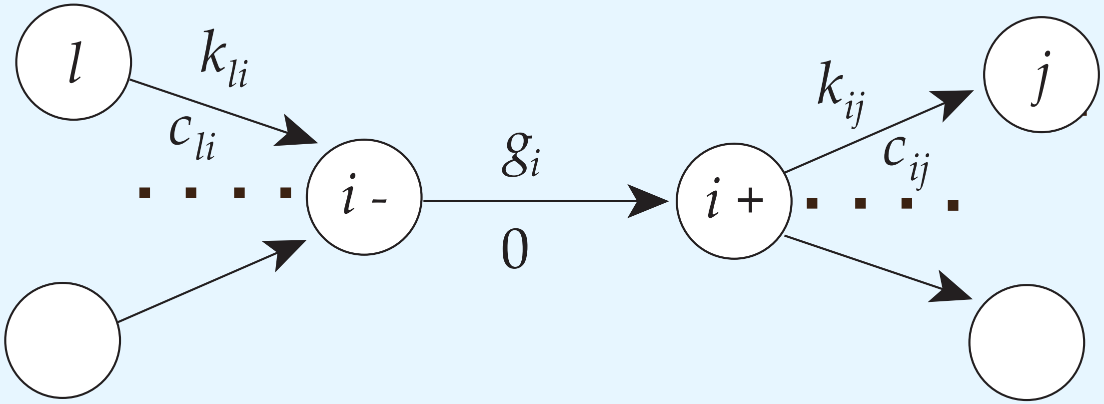


Variant ③: (cont'd)
In this case, the following **3 simple operations** can be performed:

❶ split **node** i into **2 nodes** i^- **and** i^+;

❷ substitute **each arc** $(l, i) \in A$ with the **arc** (l, i^-) with capacity k_{li} and **cost** c_{li} and **each arc** $(i, j) \in A$ with the **arc** (i^+, j) with capacity k_{ij} and **cost** c_{ij};

❸ define an **arc** (i^-, i^+) with **capacity** g_i and **null cost**.



Special cases of the Network Flow Problem

Special Network Flows, ①

Special cases of the **network flow problem** are **important** and **classical optimization problems**, among them

- ✔ the **transportation problem**;
- ✔ the **assignment problem**;
- ✔ the **shortest path problems** (under a certain assumption on the arc costs);
- ✔ the **maximum flow problem**;
- ✔ ...

Special Network Flows, ②

The **shortest path problem** under the **assumption** that $c_{ij} \geq 0, \forall\,(i,j) \in A$, or that **there are no negative cycles**.

Special Network Flows, ②

The **shortest path problem** under the **assumption** that $c_{ij} \geq 0, \forall\,(i,j) \in A$, or that **there are no negative cycles**.

$$(\text{SP}) \min \sum_{(i,j)\in A} c_{ij} x_{ij}$$

s.t.

$$\forall\, i \in V, \quad \sum_{j \in FS(i)} x_{ij} - \sum_{j \in BS(i)} x_{ji} = \begin{cases} 1, & \text{if } i = s; \\ -1, & \text{if } i = d; \\ 0, & \text{otherwise.} \end{cases}$$

$$x_{ij} \in \{0,1\},\ \forall\,(i,j) \in A.$$

Special Network Flows, ②

The **shortest path problem** under the **assumption** that $c_{ij} \geq 0, \forall\,(i,j) \in A$, or that **there are no negative cycles**.

$$\text{(SP)} \min \sum_{(i,j)\in A} c_{ij} x_{ij}$$

s.t.

$$\forall\, i \in V, \quad \sum_{j \in FS(i)} x_{ij} - \sum_{j \in BS(i)} x_{ji} = \begin{cases} 1, & \text{if } i = s; \\ -1, & \text{if } i = d; \\ 0, & \text{otherwise.} \end{cases}$$

$$x_{ij} \in \{0,1\}, \ \forall\,(i,j) \in A.$$

SP can be viewed as a **minimum cost flow problem**, where a **single unit of flow** has to be sent from a **single source node** s to a **single sink node** d.

All nodes $i \in V \setminus \{s,d\}$ are **transit nodes**.

Special Network Flows, ③

The **shortest path tree problem** under the **assumption** that $c_{ij} \geq 0, \forall\,(i,j) \in A$, or that **there are no negative cycles**.

$$\text{(SPT)} \min \sum_{(i,j)\in A} c_{ij}x_{ij}$$

s.t.

$$\forall\, i \in V, \quad \sum_{j\in FS(i)} x_{ij} - \sum_{j\in BS(i)} x_{ji} = \begin{cases} n-1, & \text{if } i = s; \\ -1, & \text{otherwise.} \end{cases}$$

$$x_{ij} \in \{0,1\},\ \forall\,(i,j) \in A.$$

Special Network Flows, ③

The **shortest path tree problem** under the **assumption** that $c_{ij} \geq 0, \forall\,(i,j) \in A$, or that **there are no negative cycles**.

$$\text{(SPT) } \min \sum_{(i,j)\in A} c_{ij}x_{ij}$$

s.t.

$$\forall\, i \in V, \quad \sum_{j\in FS(i)} x_{ij} - \sum_{j\in BS(i)} x_{ji} = \begin{cases} n-1, & \text{if } i = s; \\ -1, & \text{otherwise.} \end{cases}$$

$$x_{ij} \in \{0,1\},\ \forall\,(i,j) \in A.$$

SPT can be viewed as a **minimum cost flow problem**, where $n-1$ **units of flow** have to be sent from a **single source node** s to **any other node** $i \in V \setminus \{s\}$.

All nodes $i \in V \setminus \{s\}$ are **sink nodes**; there are **no transit nodes**.

Paola Festa - DMA, University of Napoli FEDERICO II
E-mail: paola.festa@unina.it - Web: http://www.dma.unina.it/∼festa

The Maximum Flow Problem

Definition

The **Maximum Flow Problem** (**MF**) is a **special flow problem** defined on a **network flow** $N = (V, A)$ s.t.

- $\forall\ (i, j) \in A$
  - c_{ij} represents the cost per unit of flow sent along arc (i, j);
  - $k_{ij} \geq 0$ represents an upper bound on the quantity of flow that can be sent along arc (i, j);
- $s,\ d \in V$, $s \neq d$, are the source and the sink nodes in N, respectively.

The **Maximum Flow Problem** (**MF**) is a **special flow problem** defined on a **network flow** $N = (V, A)$ s.t.

- $\forall\ (i, j) \in A$
  - c_{ij} represents the cost per unit of flow sent along arc (i, j);
  - $k_{ij} \geq 0$ represents an upper bound on the quantity of flow that can be sent along arc (i, j);
- $s,\ d \in V$, $s \neq d$, are the source and the sink nodes in N, respectively.

Objective: sent the **max amount of flow from** the source node s **to** the sink node d.

The **Maximum Flow Problem** (**MF**) is a **special flow problem** defined on a **network flow** $N = (V, A)$ s.t.

- $\forall\ (i, j) \in A$
  - c_{ij} represents the cost per unit of flow sent along arc (i, j);
  - $k_{ij} \geq 0$ represents an upper bound on the quantity of flow that can be sent along arc (i, j);
- $s,\ d \in V$, $s \neq d$, are the source and the sink nodes in N, respectively.

Objective: sent the **max amount of flow from** the source node s **to** the sink node d.

Note ①: we know that the **hypothesis** that s and d are the **only source and sink nodes is not restrictive**.

The **Maximum Flow Problem** (**MF**) is a **special flow problem** defined on a **network flow** $N = (V, A)$ s.t.

- $\forall\ (i, j) \in A$
  - c_{ij} represents the cost per unit of flow sent along arc (i, j);
  - $k_{ij} \geq 0$ represents an upper bound on the quantity of flow that can be sent along arc (i, j);
- $s,\ d \in V,\ s \neq d$, are the source and the sink nodes in N, respectively.

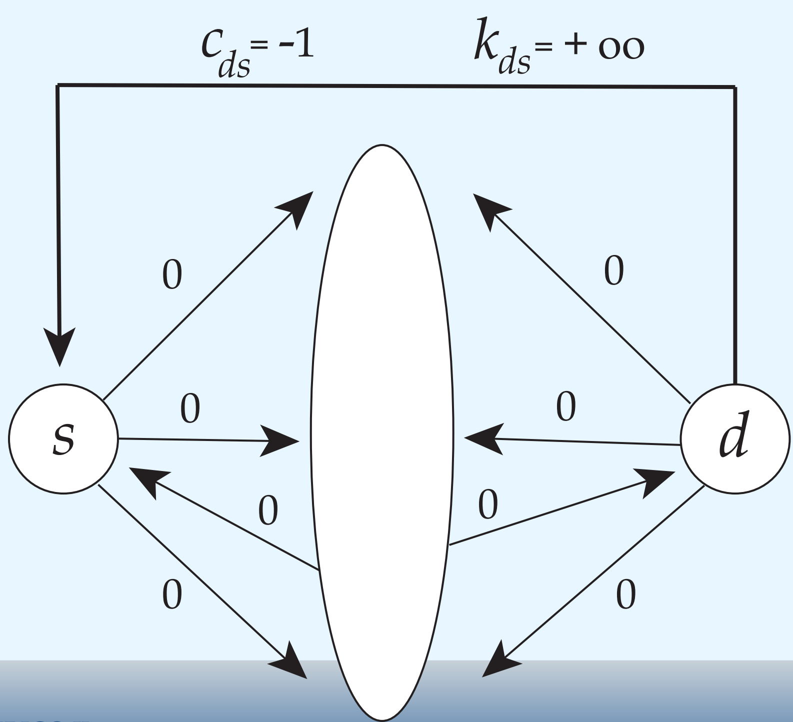
Objective: sent the **max amount of flow from** the source node s **to** the sink node d.

Note ①: we know that the **hypothesis** that s and d are the **only source and sink nodes is not restrictive**.

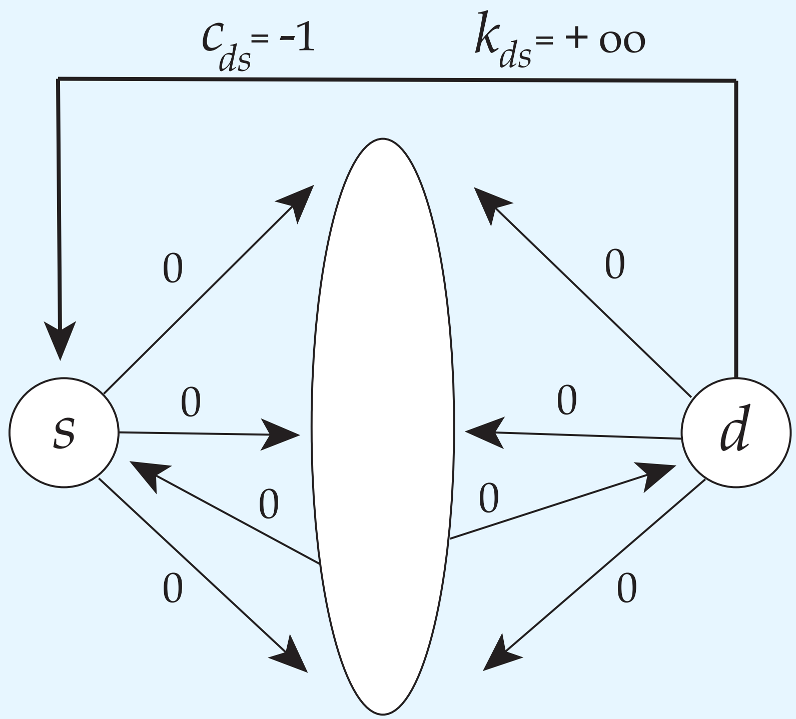
Note ②:
Max Flow Problem $\Leftrightarrow$ **Min Cost Flow Problem** (**MCF**).

Any instance $N = (V, A)$ of MF, where each arc $(i, j) \in A$ is associated with a **capacity** k_{ij}, can be reduced to an **instance** $\overline{N} = (\overline{V}, \overline{A})$ **of MCF** s.t.

- ✔ $\overline{V} = V; \quad \overline{A} = A \cup \{(d, s)\};$
- ✔ $\forall\,(i, j) \in \overline{A} \setminus \{(d, s)\}$, $c_{ij} = 0$ and k_{ij} is unchanged;
- ✔ $c_{ds} = -1$ and $k_{ds} = +\infty$.



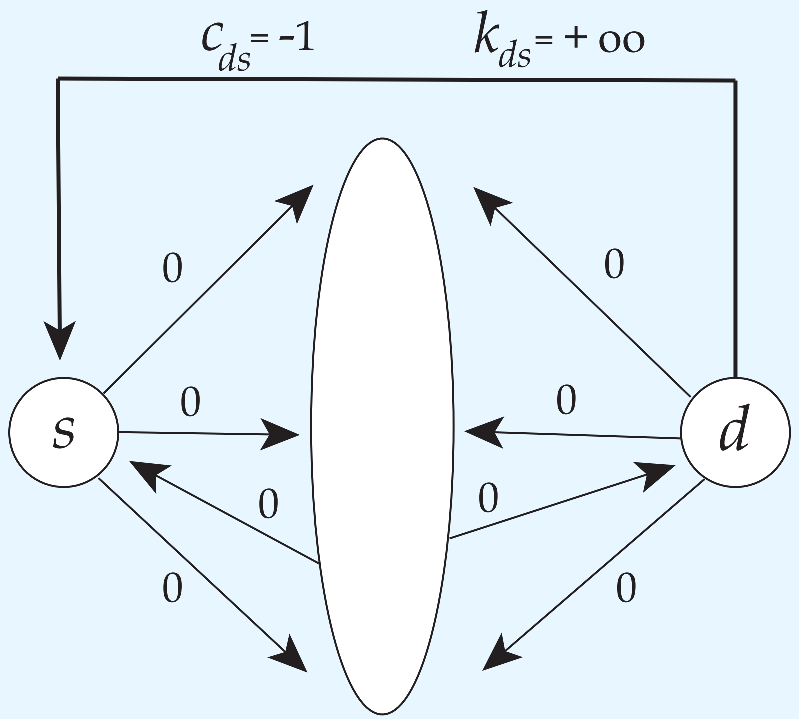
MF as MCF, ②



To find a **max flow from s to d in $N = (V, A)$** is equivalent to finding a **min cost flow in $\overline{N} = (\overline{V}, \overline{A})$**:

$$\boxed{\max x_{ds} \Longleftrightarrow \min \sum_{(i,j)\in A} c_{ij}x_{ij}}$$

MF as MCF, ②



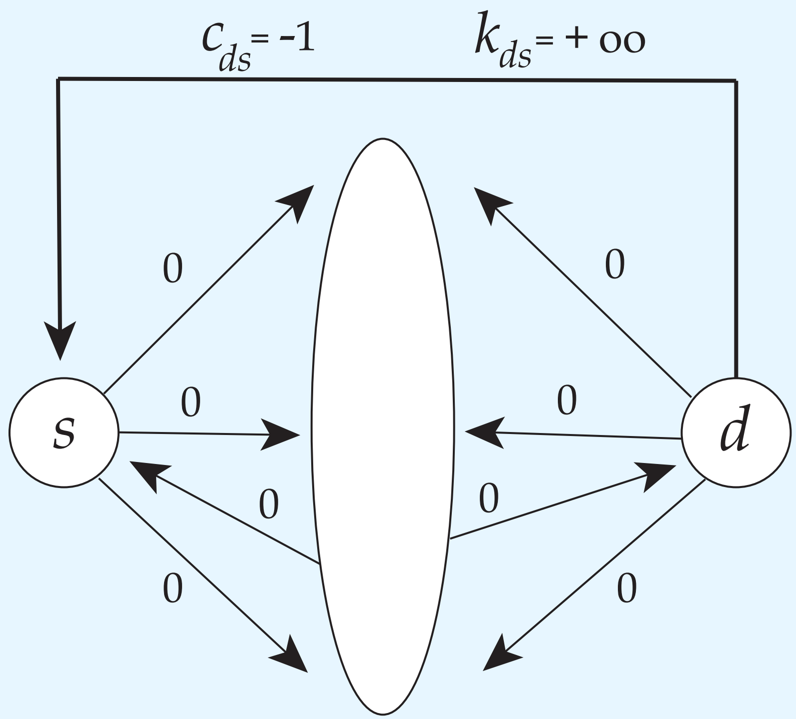
To find a **max flow from** s **to** d **in** $N = (V, A)$ is equivalent to finding a **min cost flow in** $\overline{N} = (\overline{V}, \overline{A})$:

$$\boxed{\max x_{ds} \iff \min \sum_{(i,j)\in A} c_{ij}x_{ij}}$$

and

$$\boxed{\max x_{ds} \iff \max x_{sd}}$$

MF as MCF, ②



To find a **max flow from s to d in $N = (V, A)$** is equivalent to finding a **min cost flow in $\overline{N} = (\overline{V}, \overline{A})$**:

$$\boxed{\max x_{ds} \iff \min \sum_{(i,j)\in A} c_{ij}x_{ij}} \quad \text{and} \quad \boxed{\max x_{ds} \iff \max x_{sd}}$$

Nevertheless, typically the **math formulation** reflects the **special peculiarity of the problem**.

For each $(i,j) \in A$ let

$$x_{ij} \text{ be } \textbf{the amount of flow sent along } (i,j),$$

$$\text{(MF)} \quad \max \varphi_0 = \sum_{j \in FS(s)} x_{sj} - \sum_{j \in BS(s)} x_{js}$$

s.t.

$$\text{(a)} \quad \sum_{j \in FS(i)} x_{ij} - \sum_{j \in BS(i)} x_{ji} = 0, \qquad \forall\, i \in V \setminus \{s,d\}$$

$$\text{(b)} \quad 0 \le x_{ij} \le k_{ij},\ \forall (i,j) \in A.$$

For each $(i,j) \in A$ let

x_{ij} be **the amount of flow sent along** (i,j),

$$\text{(MF)} \quad \max \varphi_0 = \sum_{j \in FS(s)} x_{sj} - \sum_{j \in BS(s)} x_{js}$$

s.t.

$$\text{(a)} \quad \sum_{j \in FS(i)} x_{ij} - \sum_{j \in BS(i)} x_{ji} = 0, \qquad \forall\, i \in V \setminus \{s, d\}$$

$$\text{(b)} \quad 0 \le x_{ij} \le k_{ij},\ \forall (i,j) \in A.$$

Note ①: if $BS(s) = \emptyset$, the o.f. reduces to

$$\max \varphi_0 = \sum_{j \in FS(s)} x_{sj}.$$

Math formulation, ②

For each $(i,j) \in A$ let

x_{ij} be **the amount of flow sent along** (i,j),

$$\text{(MF)} \quad \max \varphi_0 = \sum_{j \in FS(s)} x_{sj} - \sum_{j \in BS(s)} x_{js}$$

s.t.

$$\text{(a)} \quad \sum_{j \in FS(i)} x_{ij} - \sum_{j \in BS(i)} x_{ji} = 0, \qquad \forall\, i \in V \setminus \{s,d\}$$

$$\text{(b)} \quad 0 \le x_{ij} \le k_{ij},\ \forall (i,j) \in A.$$

Note ②: **each feasible solution (feasible flow) for (MF)** is a **circulation**.

Math formulation, ③

For each $(i, j) \in A$ let

x_{ij} be **the amount of flow sent along** (i, j),

$$\text{(MF)} \quad \max \varphi_0 = \sum_{j \in FS(s)} x_{sj} - \sum_{j \in BS(s)} x_{js}$$

s.t.

$$\text{(a)} \quad \sum_{j \in FS(i)} x_{ij} - \sum_{j \in BS(i)} x_{ji} = 0, \qquad \forall\, i \in V \setminus \{s, d\}$$

$$\text{(b)} \quad 0 \le x_{ij} \le k_{ij},\ \forall (i, j) \in A.$$

Note ③: constraints (a) become

for $i = s$**:** $$\sum_{j \in FS(s)} x_{sj} - \sum_{j \in BS(s)} x_{js} = \varphi_0;$$

for $i = d$**:** $$\sum_{j \in FS(d)} x_{dj} - \sum_{j \in BS(d)} x_{jd} = -\varphi_0.$$

Paola Festa - DMA, University of Napoli FEDERICO II
E-mail: paola.festa@unina.it - Web: http://www.dma.unina.it/∼festa

Math formulation, ④

For each $(i, j) \in A$ let

$$x_{ij} \text{ be } \textbf{the amount of flow sent along } (i, j),$$

$$\text{(MF)} \quad \max \varphi_0 = \sum_{j \in FS(s)} x_{sj} - \sum_{j \in BS(s)} x_{js}$$

s.t.

$$\text{(a)} \quad \sum_{j \in FS(i)} x_{ij} - \sum_{j \in BS(i)} x_{ji} = 0, \qquad \forall\, i \in V \setminus \{s, d\}$$

$$\text{(b)} \quad 0 \le x_{ij} \le k_{ij},\ \forall (i, j) \in A.$$

Note ④: since constraints matrix is unimodular (node-arc incidence matrix), **MF has an integer optimal solution if**

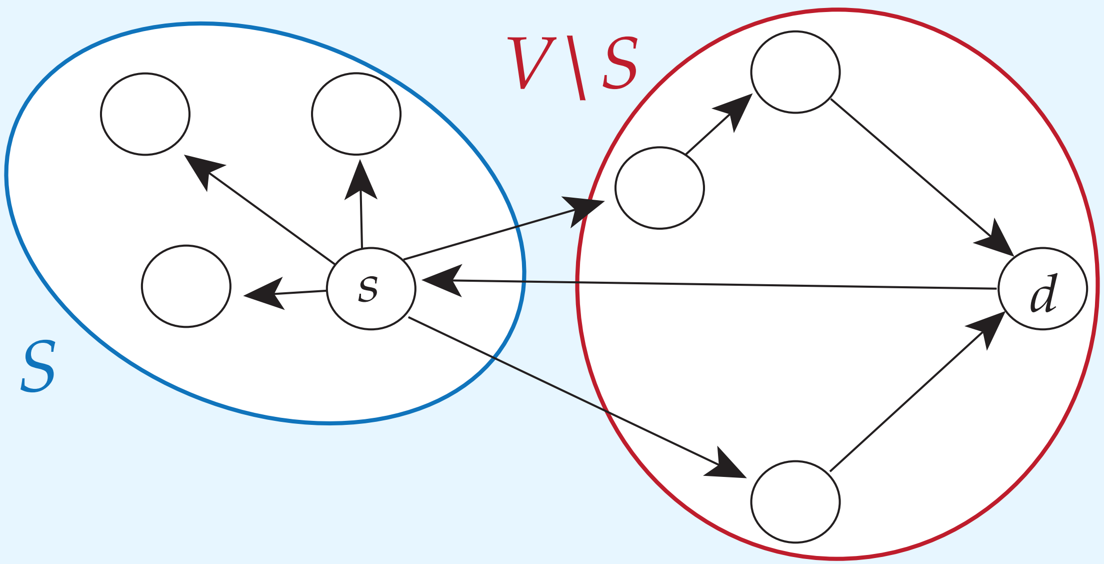
$$k_{ij} \in \mathbb{Z}^+ \cup \{0\}, \forall\, (i, j) \in A$$

and it could be solved by the **Simplex Method**!

Properties, ①

Definition. Given a network $N = (V, A)$, a **cut** is a **partition** $(S, V \setminus S)$ of V s.t.

$$s \in S \quad \text{and} \quad d \in V \setminus S.$$



Properties, ②

Let be

$\delta_N^+(S) := \{(i,j) \in A \mid i \in S,\ j \in V \setminus S)\};$

$\delta_N^-(S) := \{(i,j) \in A \mid i \in V \setminus S,\ j \in S)\}.$

Definition. Given a feasible flow vector x, the **amount of flow crossing a cut** $(S, V \setminus S)$ is given by:

$$\varphi(S) = \sum_{(i,j)\in\delta_N^+(S)} x_{ij} - \sum_{(j,i)\in\delta_N^-(S)} x_{ji},$$

i.e., the difference between the total flow "leaving" from S and that "entering" S.

Let be

$\delta_N^+(S) := \{(i,j) \in A \mid i \in S,\ j \in V \setminus S)\};$

$\delta_N^-(S) := \{(i,j) \in A \mid i \in V \setminus S,\ j \in S)\}.$

Definition. Given a feasible flow vector x, the **amount of flow crossing a cut** $(S, V \setminus S)$ is given by:

$$\varphi(S) = \sum_{(i,j)\in\delta_N^+(S)} x_{ij} - \sum_{(j,i)\in\delta_N^-(S)} x_{ji},$$

i.e., the difference between the total flow “leaving” from S and that “entering” S.

The quantity

$$\varphi_0 = \varphi(\{s\}) \stackrel{\text{def.}}{=} \sum_{(s,j)\in\delta_N^+(\{s\})} x_{sj} - \sum_{(j,s)\in\delta_N^-(\{s\})} x_{js}$$

is called **value of the flow** x.

Definition. Given a cut $(S, V \setminus S)$ in a network flow $N = (V, A)$, the **capacity of the cut** $(S, V \setminus S)$ is given by:

$$K(S) = \sum_{(i,j)\in\delta_N^+(S)} k_{ij}.$$

Note: $K(S)$ does not take into account the capacities of the arcs "entering" S, i.e. those belonging to $\delta_N^-(S)$.

Properties, ④

Theorem. Given a feasible flow vector x in a network flow $N = (V, A)$, **for every cut** $(S, V \setminus S)$ in N it results that

$$\varphi(S) = \varphi_0 = \varphi(\{s\}).$$

Properties, ④

Theorem. Given a feasible flow vector x in a network flow $N = (V, A)$, **for every cut** $(S, V \setminus S)$ in N it results that

$$\varphi(S) = \varphi_0 = \varphi(\{s\}).$$

Note: the theorem claims that, **for every cut** $(S, V \setminus S)$, **the amount of flow that crosses it** is always equal to φ_0, i.e. it is **constant**.

Theorem. Given a feasible flow vector x in a network flow $N = (V, A)$, **for every cut** $(S, V \setminus S)$ in N it results that

$$\varphi(S) = \varphi_0 = \varphi(\{s\}).$$

Proof:
from the **flow conservation law** applied **to every node** $i \in S \setminus \{s\}$.

Theorem. Given a feasible flow vector x in a network flow $N = (V, A)$, **for every cut** $(S, V \setminus S)$ in N it results that

$$\varphi(S) = \varphi_0 = \varphi(\{s\}).$$

Proof:

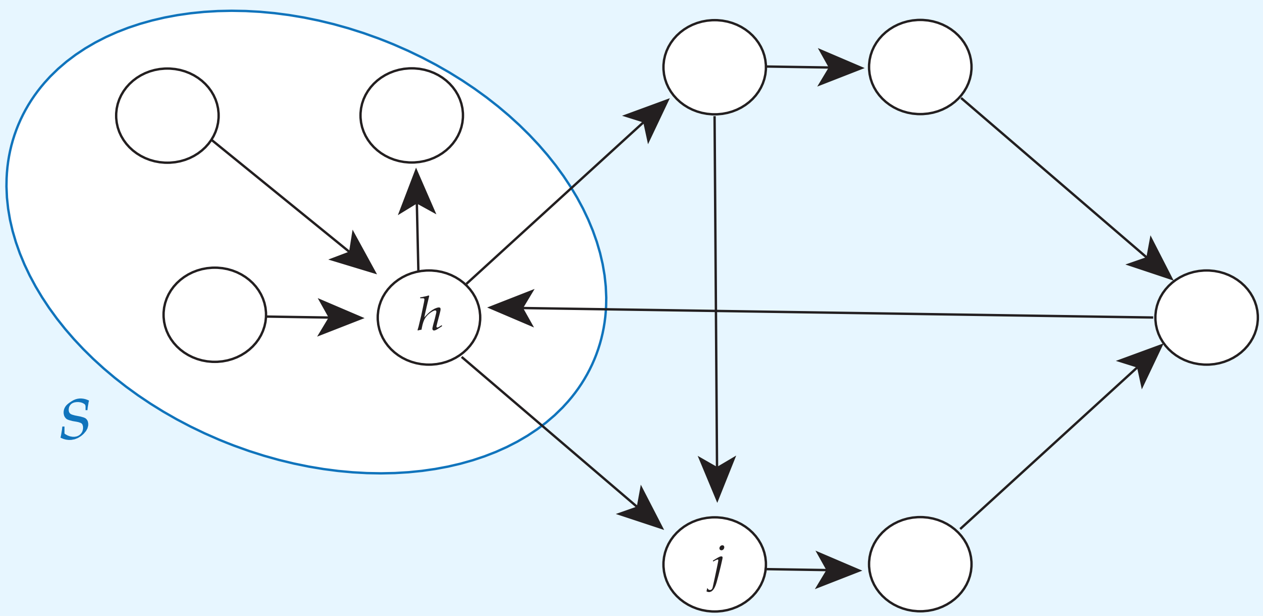
$$\begin{aligned}
\varphi_0 &= \varphi(\{s\}) = \sum_{(s,j)\in\delta_N^+(\{s\})} x_{sj} - \sum_{(i,s)\in\delta_N^-(\{s\})} x_{is} \\
&= \sum_{h\in S}\left[\underbrace{\sum_{(h,j)\in\delta_N^+(\{h\})} x_{hj} - \sum_{(j,h)\in\delta_N^-(\{h\})} x_{jh}}_{=0,\,\forall\, h\neq s}\right] \\
&= \sum_{h\in S}\sum_{(h,j)\in\delta_N^+(\{h\})} x_{hj} - \sum_{h\in S}\sum_{(j,h)\in\delta_N^-(\{h\})} x_{jh}
\end{aligned}$$

Theorem. Given a feasible flow vector x in a network flow $N = (V, A)$, **for every cut** $(S, V \setminus S)$ in N it results that

$$\varphi(S) = \varphi_0 = \varphi(\{s\}).$$

Proof: Note that,

$$\sum_{h \in S} \sum_{(h,j) \in \delta_N^+(\{h\})} x_{hj} = \sum_{h \in S} \sum_{j \in FS(h)} x_{hj} = \sum_{(i,j) \in A(S)} x_{ij} + \sum_{(i,j) \in \delta_N^+(S)} x_{ij}$$

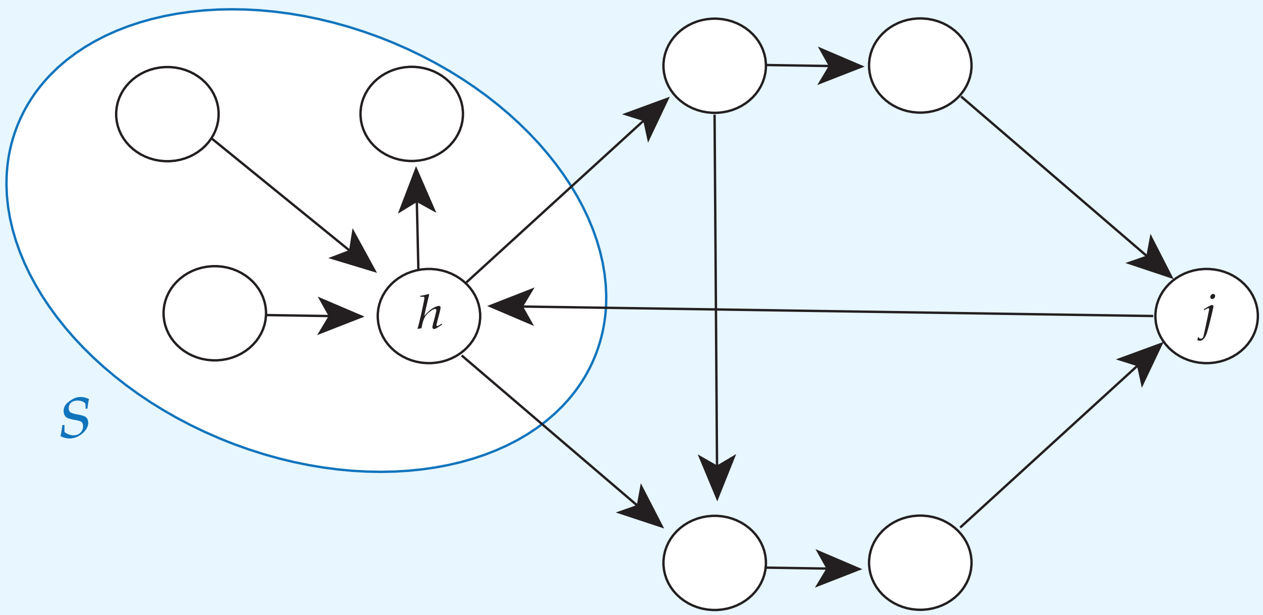


Theorem. Given a feasible flow vector x in a network flow $N = (V, A)$, **for every cut** $(S, V \setminus S)$ in N it results that

$$\varphi(S) = \varphi_0 = \varphi(\{s\}).$$

Proof: Similarly,

$$\sum_{h \in S} \sum_{(j,h) \in \delta_N^-(\{h\})} x_{jh} = \sum_{h \in S} \sum_{j \in BS(h)} x_{jh} = \sum_{(i,j) \in A(S)} x_{ij} + \sum_{(i,j) \in \delta_N^-(S)} x_{ij}$$



Paola Festa - DMA, University of Napoli FEDERICO II
E-mail: paola.festa@unina.it - Web: http://www.dma.unina.it/∼festa

Properties, ④

Theorem. Given a feasible flow vector x in a network flow $N = (V, A)$, **for every cut** $(S, V \setminus S)$ in N it results that

$$\varphi(S) = \varphi_0 = \varphi(\{s\}).$$

Proof: Therefore,

$$\begin{aligned}
\varphi_0 &= \cdots \\
&= \sum_{h\in S} \sum_{(h,j)\in\delta_N^+(\{h\})} x_{hj} - \sum_{h\in S} \sum_{(j,h)\in\delta_N^-(\{h\})} x_{jh} \\
&= \left[\underbrace{\sum_{(i,j)\in A(S)} x_{ij}}_{=} + \sum_{(i,j)\in\delta_N^+(S)} x_{ij} \right] - \left[\underbrace{\sum_{(i,j)\in A(S)} x_{ij}}_{=} + \sum_{(i,j)\in\delta_N^-(S)} x_{ij} \right] \\
&= \varphi(S). \square
\end{aligned}$$

Theorem. **For every feasible flow** x in a network flow $N = (V, A)$ and **for every cut** $(S, V \setminus S)$ in N, it results that

$$\varphi(S) \leq K(S).$$

Theorem. **For every feasible flow** x in a network flow $N = (V, A)$ and **for every cut** $(S, V \setminus S)$ in N, it results that

$$\varphi(S) \leq K(S).$$

Proof:

$$\varphi(S) \stackrel{\text{def.}}{=} \sum_{(i,j)\in\delta_N^+(S)} x_{ij} - \sum_{(j,i)\in\delta_N^-(S)} x_{ji}$$

$$\overset{\sum_{(j,i)\in\delta_N^-(S)} x_{ji} \geq 0}{\leq} \sum_{(i,j)\in\delta_N^+(S)} x_{ij}$$

$$\overset{x_{ij}\leq k_{ij}\ \forall (i,j)\in A}{\leq} \sum_{(i,j)\in\delta_N^+(S)} k_{ij}$$

$$\stackrel{\text{def.}}{=} K(S). \qquad \square$$

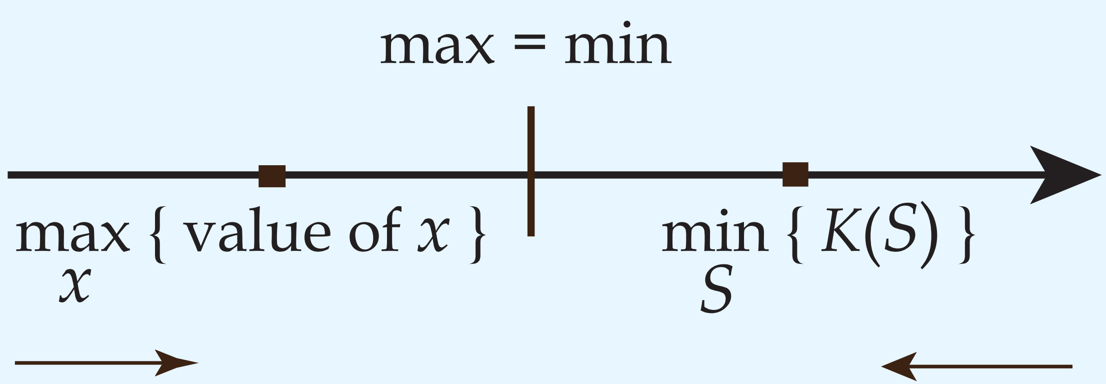
Theorem [**Maximum Flow / Minimum Cut**].

Max Flow / Min Cut, ①

Theorem [Maximum Flow / Minimum Cut].

A **feasible flow** x in a network flow $N = (V, A)$ is a **maximum flow** <u>**iff**</u> there exists a **cut** $(S^*, V \setminus S^*)$ **in** N **with minimum capacity** such that

$$\varphi(S^*) = \varphi_0^* = K(S^*).$$



Theorem [**Maximum Flow / Minimum Cut**].

A **feasible flow** x in a network flow $N = (V, A)$ is a **maximum flow** **iff** there exists a **cut** $(S^*, V \setminus S^*)$ **in** N **with minimum capacity** such that

$$\varphi(S^*) = \varphi_0^* = K(S^*).$$

Proof: *constructive*.

It is needed to define

- **saturated and unloading arcs** and
- **residual network** $\overline{N}$ that can be built from N, once available a feasible flow x.

$\overline{N}$ is an **auxiliary network** built to **keep track of the amount of flow that can be pushed** along the arcs of the **original network** N.

Paola Festa - DMA, University of Napoli FEDERICO II
E-mail: paola.festa@unina.it - Web: http://www.dma.unina.it/∼festa

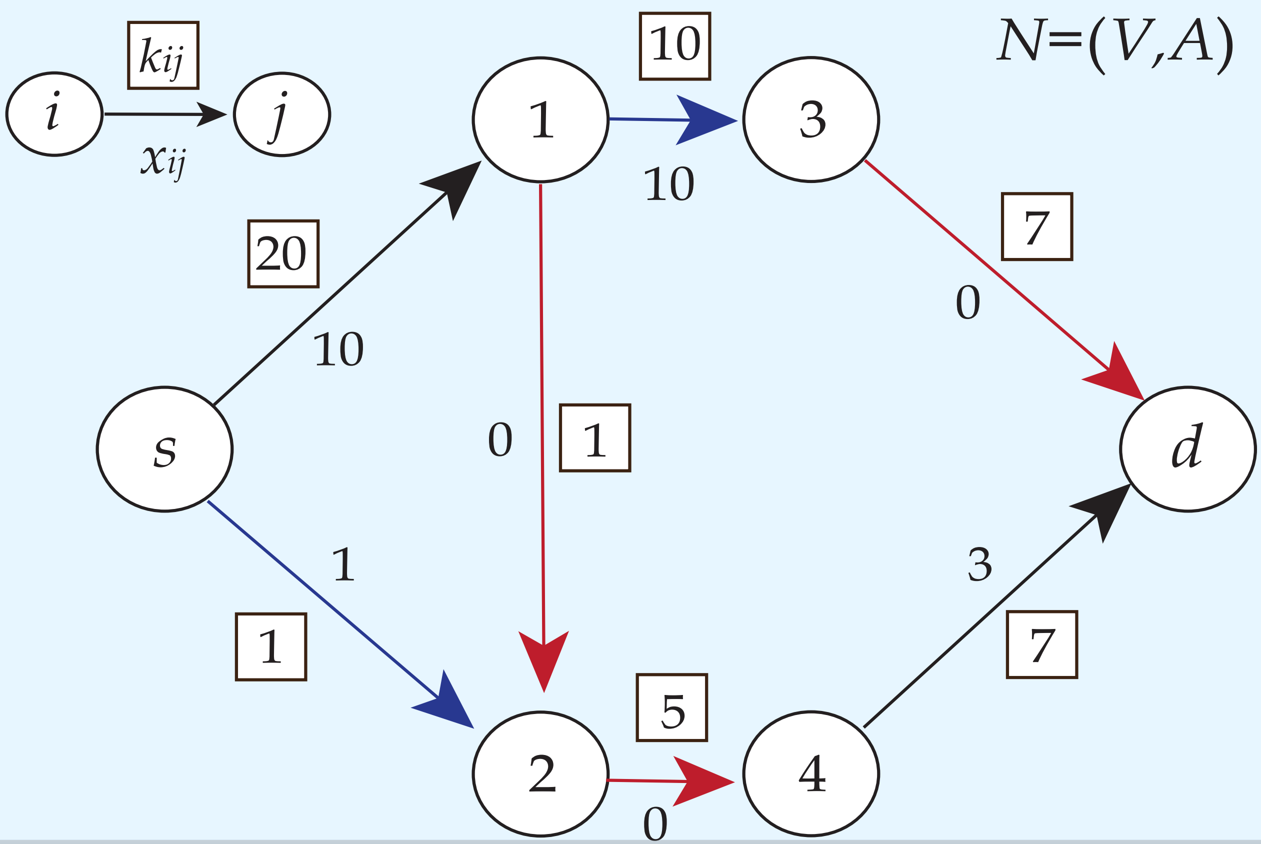
Definition. **Given a feasible flow** x crossing a network flow $N = (V, A)$, **an arc** $(i, j) \in A$ is said

- ✔ **saturated**, if $k_{ij} - x_{ij} = 0$;
- ✔ **unloading**, if $x_{ij} = 0$.

Satured and unloading arcs

Definition. **Given a feasible flow** x crossing a network flow $N = (V, A)$, **an arc** $(i, j) \in A$ is said

- ✔ **saturated**, if $k_{ij} - x_{ij} = 0$;
- ✔ **unloading**, if $x_{ij} = 0$.



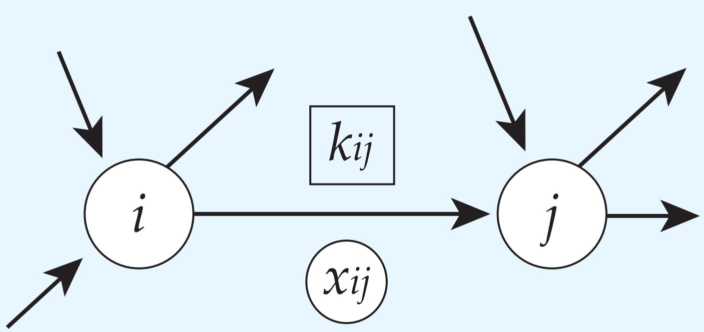
Residual network, ①

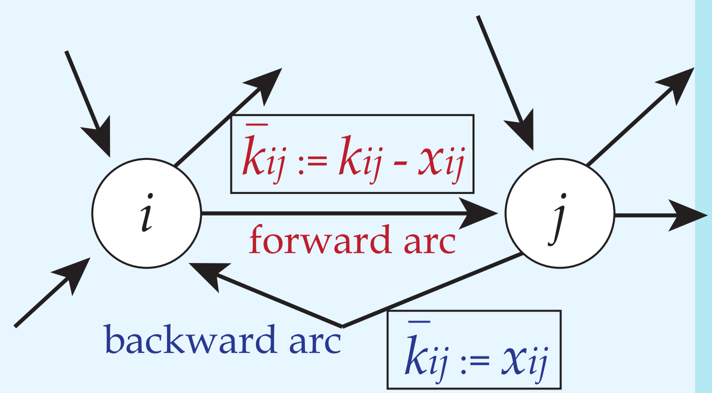
Definition. Given a feasible flow x crossing a network flow $N = (V, A)$, the **residual network** $\overline{N} = (V, \overline{A})$ **associated with** N is built performing the following **2 operations**:

Residual network, ①

Definition. Given a feasible flow x crossing a network flow $N = (V, A)$, the **residual network** $\overline{N} = (V, \overline{A})$ **associated with** N is built performing the following **2 operations**:

① substitute each arc $(i, j) \in A$ with
- a **forward arc** (i, j) with *residual capacity* $\overline{k}_{ij} = k_{ij} - x_{ij} \geq 0$ (by def. of k_{ij});
- a **backward arc** (j, i) with *residual capacity* $\overline{k}_{ji} = x_{ij} \geq 0$ (by def. of feasible flow x);





Residual network, ①

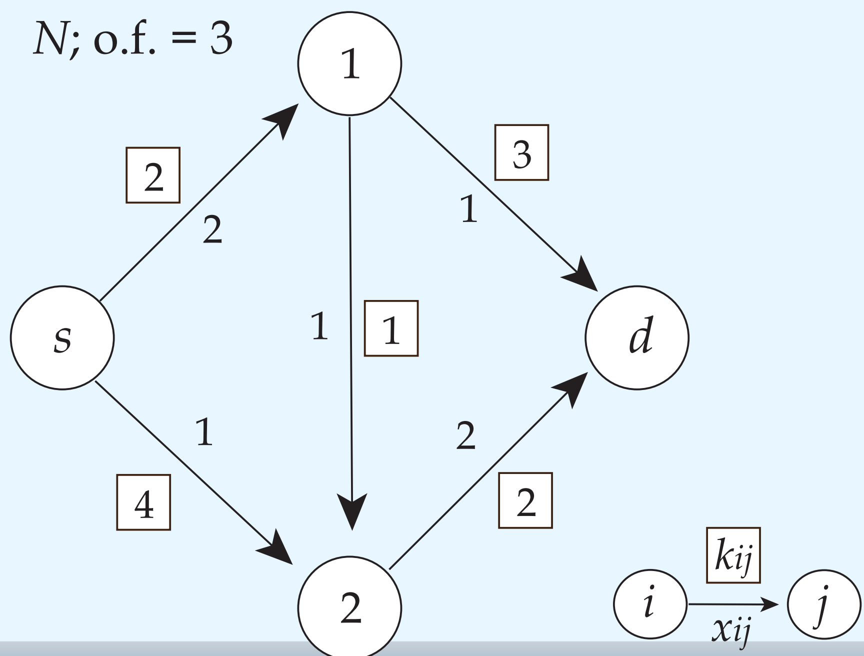
Definition. Given a feasible flow x crossing a network flow $N = (V, A)$, the **residual network $\overline{N} = (V, \overline{A})$ associated with N** is built performing the following **2 operations**:

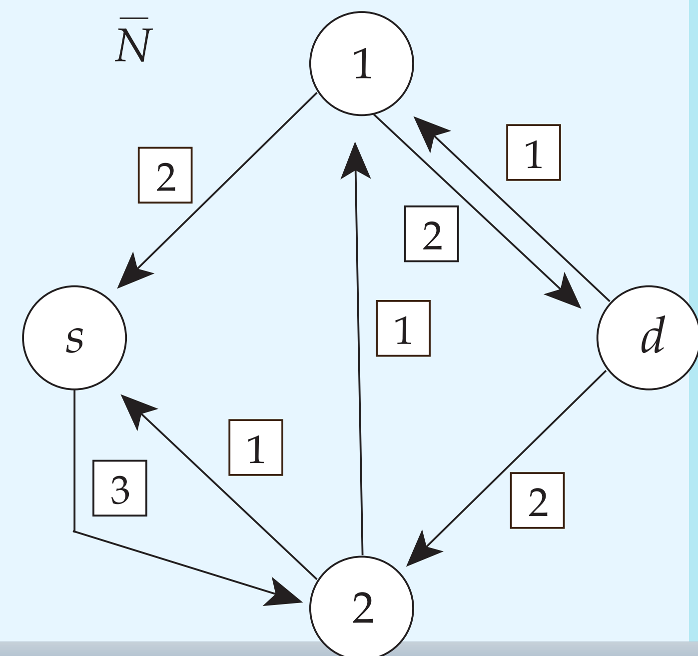
② remove all **arcs with null residual capacity** ((i, j) with $\overline{k}_{ij} = k_{ij} - x_{ij} = 0$ and (j, i) with $\overline{k}_{ji} = x_{ij} = 0$).

Residual network, ①

Definition. Given a feasible flow x crossing a network flow $N = (V, A)$, the **residual network** $\overline{N} = (V, \overline{A})$ **associated with** N is built performing the following **2 operations**:

② remove all **arcs with null residual capacity** ((i, j) with $\overline{k}_{ij} = k_{ij} - x_{ij} = 0$ and (j, i) with $\overline{k}_{ji} = x_{ij} = 0$).





Residual network, ②

The **residual network** $\overline{N}$ allows to find a **flow** $\overline{x}$ **from** $s \in V$ **to** $d \in V$ **higher than** x,

- ✓ **increasing the flow along non saturated arcs** (i.e., those arcs (i, j) with $\overline{k}_{ij} = k_{ij} - x_{ij} \neq 0$) and
- ✓ **decreasing it along non unloading arcs** (i.e., those arcs (j, i) with $\overline{k}_{ji} = x_{ij} \neq 0$).

The **residual network** $\overline{N}$ allows to find a **flow** $\overline{x}$ **from** $s \in V$ **to** $d \in V$ **higher than** x,

- ✓ **increasing the flow along non saturated arcs** (i.e., those arcs (i, j) with $\overline{k}_{ij} = k_{ij} - x_{ij} \neq 0$) and
- ✓ **decreasing it along non unloading arcs** (i.e., those arcs (j, i) with $\overline{k}_{ji} = x_{ij} \neq 0$).

If $\exists\ P$ **from** s **to** d **in** $\overline{N} \Longrightarrow \exists$ **in** N $\overline{x}$ **higher than** x!

Residual network, ②

The **residual network** $\overline{N}$ allows to find a **flow** $\overline{x}$ **from** $s \in V$ **to** $d \in V$ **higher than** x,

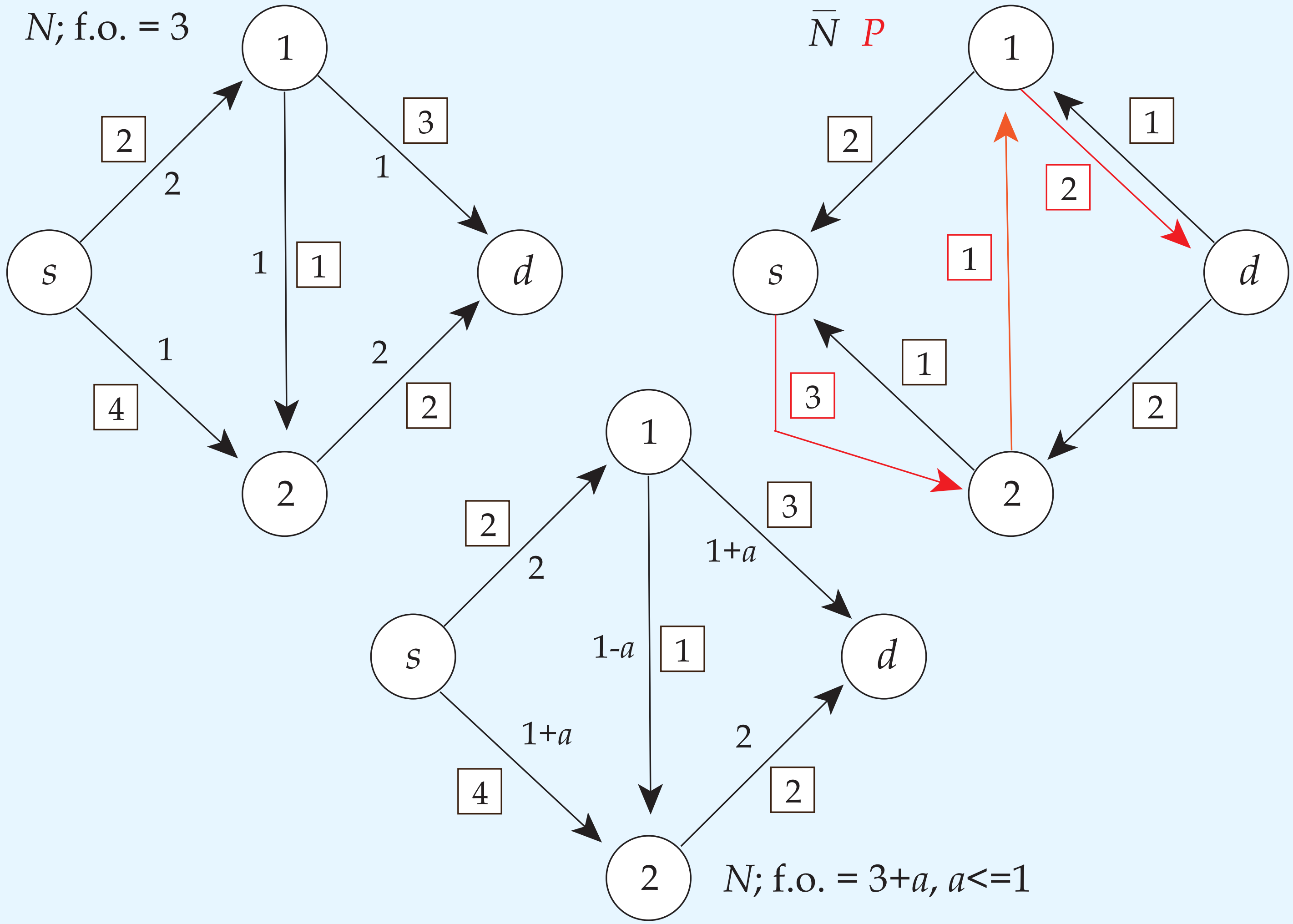
- ✓ **increasing the flow along non saturated arcs** (i.e., those arcs (i, j) with $\overline{k}_{ij} = k_{ij} - x_{ij} \neq 0$) and
- ✓ **decreasing it along non unloading arcs** (i.e., those arcs (j, i) with $\overline{k}_{ji} = x_{ij} \neq 0$).

If $\exists$ P **from** s **to** d **in** $\overline{N} \Longrightarrow \exists$ **in** N $\overline{x}$ **higher than** x!

P **from** s **to** d is said **augmenting path** and **in** N **it corresponds to a sequence**

- ✍ of **forward arcs along which to increase the current flow** and
- ✍ of **backward arcs along which to decrease the current flow**.

Residual network, ②



P in *N* **is a sequence** of **forward arcs along which to increase the current flow** and **backward arcs along which to decrease the current flow**.

Augmenting path

Definition. Given a feasible flow x in a network flow $N = (V, A)$, an **augmenting path** is a path P **from the source** s **to the sink** d **in** N s.t., denoting by

- F the **set of forward arcs in** P and
- B the **set of backward arcs in** P,

it results that

- $x_{ij} < k_{ij}, \forall\, (i, j) \in F \Longrightarrow$
  it is possible to increase the current flow along (i, j)!

Paola Festa - DMA, University of Napoli FEDERICO II
E-mail: paola.festa@unina.it - Web: http://www.dma.unina.it/∼festa

Augmenting path

Definition. Given a feasible flow x in a network flow $N = (V, A)$, an **augmenting path** is a path P **from the source** s **to the sink** d **in** N s.t., denoting by

- F the **set of forward arcs in** P and
- B the **set of backward arcs in** P,

it results that

- $x_{ij} < k_{ij}, \forall\, (i, j) \in F \Longrightarrow$ it is possible to increase the current flow along (i, j)!
- $x_{ij} > 0, \forall\, (i, j) \in B \Longrightarrow$ it is possible to decrease the current flow along (i, j)!

Theorem. A **feasible flow** x in a network flow $N = (V, A)$ is **optimal for MF iff** node d **can not be reached** from node s **in the residual network** $\overline{N}$.

Paola Festa - DMA, University of Napoli FEDERICO II
E-mail: paola.festa@unina.it - Web: http://www.dma.unina.it/∼festa

Optimality conditions, ①

Theorem. A **feasible flow** x in a network flow $N = (V, A)$ is **optimal for MF iff** node d **can not be reached** from node s **in the residual network** $\overline{N}$.

Proof. Let φ_0 be the value of flow x.

Scenario ①: d is reachable from s in $\overline{N}$.

Optimality conditions, ①

Theorem. A **feasible flow** x in a network flow $N = (V, A)$ is **optimal for MF iff** node d **can not be reached** from node s **in the residual network** $\overline{N}$.

Proof. Let φ_0 be the value of flow x.

Scenario ①: d is reachable from s in $\overline{N}$.

Then, $\exists\ P$ augmenting from s to d in $\overline{N}$.

Let be $\delta = \min\{\overline{k}_{ij} \mid (i,j) \in P\}(> 0!)$ and $\forall\ (i,j) \in P$

$$\overline{x}_{ij} := \begin{cases} x_{ij} + \delta, & \text{if } (i,j) \in F; \\ x_{ij} - \delta, & \text{if } (i,j) \in B, \end{cases}$$

$\overline{x}$ is a **feasible flow in** N with value $\varphi_0 + \delta > \varphi_0 \Longrightarrow$
$\Longrightarrow x$ **is not optimal**. □

Optimality conditions, ②

Theorem. A **feasible flow** x in a network flow $N = (V, A)$ is **optimal for MF iff** node d **can not be reached** from node s **in the residual network** $\overline{N}$.

Proof. Let φ_0 be the value of flow x.

Scenario ②: d is not reachable from s in $\overline{N}$.

Then, $\exists\ (S^*, V \setminus S^*)$ in $\overline{N}$ s.t. $\delta^+_{\overline{N}}(S^*) = \emptyset$.

Paola Festa - DMA, University of Napoli FEDERICO II
E-mail: paola.festa@unina.it - Web: http://www.dma.unina.it/∼festa

Theorem. A **feasible flow** x in a network flow $N = (V, A)$ is **optimal for MF iff** node d **can not be reached** from node s **in the residual network** $\overline{N}$.

Proof. Let φ_0 be the value of flow x.

Scenario ②: d is not reachable from s in $\overline{N}$.
By definition of $\overline{N}$, in N it results that

- ✔ $\forall\, (i, j) \in \delta^+_N(S^*)$, $x_{ij} = k_{ij}$ (saturated);
- ✔ $\forall\, (i, j) \in \delta^-_N(S^*)$, $x_{ij} = 0$ (unloading). **Therefore,**

$$\varphi_0 \stackrel{\text{Th.}}{=} \varphi(S^*) \stackrel{\text{def}}{=} \overbrace{\sum_{(i,j)\in\delta^+_N(S^*)} x_{ij}}^{\text{saturated}} - \overbrace{\sum_{(i,j)\in\delta^-_N(S^*)} x_{ij}}^{\text{unloading}}$$

$$= \sum_{(i,j)\in\delta^+_N(S^*)} k_{ij} \stackrel{\text{def}}{=} K(S^*).$$

Theorem. A **feasible flow** x in a network flow $N = (V, A)$ is **optimal for MF iff** node d **can not be reached** from node s **in the residual network** $\overline{N}$.

Proof. Let φ_0 be the value of flow x.

Scenario ②: d is not reachable from s in $\overline{N}$.

Summarizing, $\exists\ (S^*, V \setminus S^*)$ in $\overline{N}$ s.t. $\delta^+_{\overline{N}}(S^*) = \emptyset$ and

$$\varphi_0 \overset{\text{Th.}}{=} \varphi(S^*) = K(S^*). \square$$

Remember: $\varphi(S) \leq K(S), \forall\ (S, V \setminus S)$.

A solution method: the Ford-Fulkerson algorithm

Ford-Fulkerson algorithm, ①

Given a network flow $N = (V, A)$ and a feasible flow x from s to d, **it iteratively searches in N an augmenting path P.**

Ford-Fulkerson algorithm, ①

Given a network flow $N = (V, A)$ and a feasible flow x from s to d, **it iteratively searches in N an augmenting path P**.

If such a path P exists, then it is **possible to increase the current flow along the arcs in F** and **to decrease that sent along those in B**.

In order to **not violate the capacities constraints**, the **current flow** along the arcs **must be changed** of the **same quantity $\delta(P)$ properly chosen**...

Ford-Fulkerson algorithm, ①

Given a network flow $N = (V, A)$ and a feasible flow x from s to d, **it iteratively searches in N an augmenting path P.**

If such a path P exists, then it is **possible to increase the current flow along the arcs in F** and **to decrease that sent along those in B.**

In order to **not violate the capacities constraints**, the **current flow** along the arcs **must be changed** of the **same quantity $\delta(P)$ properly chosen**... but

$$\boxed{\delta(P) \quad = \quad ?}$$

Ford-Fulkerson algorithm, ①

Given a network flow $N = (V, A)$ and a feasible flow x from s to d, **it iteratively searches in N an augmenting path P**.

If such a path P exists, then it is **possible to increase the current flow along the arcs in F** and **to decrease that sent along those in B**.

In order to **not violate the capacities constraints**, the **current flow** along the arcs **must be changed** of the **same quantity $\delta(P)$ properly chosen**... but

$$\boxed{\delta(P) \quad = \quad ?}$$

$$\delta(P) = \min\left\{ \min_{(i,j)\in F}\{k_{ij} - x_{ij}\}, \min_{(i,j)\in B} x_{ij} \right\}.$$

Note: $P \equiv F\ (B = \emptyset)$, $k_{ij} = +\infty, \forall\ (i,j) \in P \Longrightarrow \delta(P) = +\infty$ e $\varphi_0 = +\infty$.

Paola Festa - DMA, University of Napoli FEDERICO II
E-mail: paola.festa@unina.it - Web: http://www.dma.unina.it/∼festa

Ford-Fulkerson algorithm, ②

```
algorithm ford-fulkerson(V, A, x, s, d)
1  (P, δ(P)) := search-augmenting-path(V, A, x, s, d);
2  if (P = ∅) then return(x, φ({s}));
3  do
       if (δ(P) < +∞) then
         for each (i, j) ∈ F do /* F: forward arcs in P */
           x_ij := x_ij + δ(P);
           k_ij := k_ij − δ(P);
         endfor
         for each (i, j) ∈ B do /* B: backward arcs in P */
           x_ij := x_ij − δ(P);
           k_ij := k_ij + δ(P);
         endfor
       else return(x, +∞); /* case δ(P) = +∞: unlim.max flow */
       endif
       (P, δ(P)) := search-augmenting-path(V, A, x, s, d);
4  while (P ≠ ∅)
5  return (x, φ({s}));
end ford-fulkerson
```

Ford-Fulkerson algorithm, ③

Function `search-augmenting-path`.

Input: V, A, x, s, and d.

Output: P, $\delta(P)$.

Ford-Fulkerson algorithm, ③

Function `search-augmenting-path`.

Input: V, A, x, s, and d.

Output: $P, \delta(P)$.

$$P = \emptyset \Longrightarrow x \text{ optimal;}$$

Ford-Fulkerson algorithm, ③

Function `search-augmenting-path`.

Input: V, A, x, s, and d.

Output: $P, \delta(P)$.

$$P = \emptyset \Longrightarrow x \text{ optimal;}$$

$$P \neq \emptyset, \delta(P) = +\infty \Longrightarrow \varphi_0 = +\infty;$$

Function `search-augmenting-path`.

Input: V, A, x, s, and d.

Output: $P, \delta(P)$.

$P = \emptyset \Longrightarrow x$ optimal;

$P \neq \emptyset, \delta(P) = +\infty \Longrightarrow \varphi_0 = +\infty$;

$P \neq \emptyset, \delta(P) < +\infty \Longrightarrow$ increase of the current flow.

Ford-Fulkerson algorithm, ③

Function `search-augmenting-path`.

Input: V, A, x, s, and d.

Output: P, $\delta(P)$.

$P = \emptyset \Longrightarrow x$ optimal;

$P \neq \emptyset, \delta(P) = +\infty \Longrightarrow \varphi_0 = +\infty$;

$P \neq \emptyset, \delta(P) < +\infty \Longrightarrow$ increase of the current flow.

But **how to look for an augmenting path** P?

Ford-Fulkerson algorithm, ③

Function `search-augmenting-path`.

Input: V, A, x, s, and d.

Output: P, $\delta(P)$.

$$P = \emptyset \Longrightarrow x \text{ optimal;}$$

$$P \neq \emptyset, \delta(P) = +\infty \Longrightarrow \varphi_0 = +\infty;$$

$$P \neq \emptyset, \delta(P) < +\infty \Longrightarrow \text{increase of the current flow.}$$

But **how to look for an augmenting path** P?

It is used a *labeling algorithm*, that is a **variant of a visit of the flow network** N **starting from the source** s.

Ford-Fulkerson algorithm, ③

Let x be a feasible flow and let P be a partial path P from s to a node $l \in V$ such that

- ✔ $p(\cdot)$ predecessor array in P ($p(s) = s$);
- ✔ $\forall\ (i, j) \in F$ (forward arc in P), $x_{ij} < k_{ij}$ and $p(j) = i$;
- ✔ $\forall\ (i, j) \in B$ (backward arc in B), $x_{ij} > 0$ and $p(i) = j$.

In this case, P is said **partial augmenting path** from s to l and **visiting labeled nodes**.

Ford-Fulkerson algorithm, ③

Let x be a feasible flow and let P be a partial path P from s to a node $l \in V$ such that

- ✔ $p(\cdot)$ predecessor array in P ($p(s) = s$);
- ✔ $\forall\ (i,j) \in F$ (forward arc in P), $x_{ij} < k_{ij}$ and $p(j) = i$;
- ✔ $\forall\ (i,j) \in B$ (backward arc in B), $x_{ij} > 0$ and $p(i) = j$.

In this case, P is said **partial augmenting path** from s to l and **visiting labeled nodes**.

Starting from the labeled node l, **2 cases** can occur:

Ford-Fulkerson algorithm, ③

Let x be a feasible flow and let P be a partial path P from s to a node $l \in V$ such that

- ✔ $p(\cdot)$ predecessor array in P ($p(s) = s$);
- ✔ $\forall\ (i, j) \in F$ (forward arc in P), $x_{ij} < k_{ij}$ and $p(j) = i$;
- ✔ $\forall\ (i, j) \in B$ (backward arc in B), $x_{ij} > 0$ and $p(i) = j$.

In this case, P is said **partial augmenting path** from s to l and **visiting labeled nodes**.

Starting from the labeled node l, **2 cases** can occur:

① $\exists$ **an arc** $(l, j) \in A$ **s.t.** $x_{lj} < k_{lj}$:
it is possible to **extend** path P **to include the forward arc** (l, j), so obtaining a **partial augmenting path from** s **to node** j, that becomes **labeled** and $p(j) := l$;

Ford-Fulkerson algorithm, ③

Let x be a feasible flow and let P be a partial path P from s to a node $l \in V$ such that

- ✔ $p(\cdot)$ predecessor array in P ($p(s) = s$);
- ✔ $\forall\,(i,j) \in F$ (forward arc in P), $x_{ij} < k_{ij}$ and $p(j) = i$;
- ✔ $\forall\,(i,j) \in B$ (backward arc in B), $x_{ij} > 0$ and $p(i) = j$.

In this case, P is said **partial augmenting path** from s to l and **visiting labeled nodes**.

Starting from the labeled node l, **2 cases** can occur:

② $\exists$ **an arc** $(i,l) \in A$ **s.t.** $x_{il} > 0$:
it is possible to **extend** path P **to include the backward arc** (i,l), so obtaining a **partial augmenting path from s to node i**, that becomes **labeled** and $p(i) := l$.

`search-augmenting-path` **stops when**

- ✗ **node d is labeled**:

  in this case, an **augmenting path P from the source s to the sink d has been found** and **it can be built by means of the predecessor array p**;

or

- ✗ all labeled nodes are processed and the **sink node d results not labeled**:

  in this case, $\nexists$ **an augmenting path P from the source s to the sink d**.

Ford-Fulkerson algorithm, ④

```
function search-augmenting-path (V, A, x, s, d)
1    P := ∅;  δ(P) := +∞;
2    for each  i ∈ V \ {s}  do  p(i) := 0;  label[i] := 0;  endfor
3    p(s) := s;  label[s] := 1;  Q := {s};
4    while  (Q ≠ ∅)  do
        l :=remove(Q);  /* breadth, depth, l := min_{h∈Q} h, ...*/
        for each  j ∈ FS(l)  do
          if  (label[j] = 0  and  x_{lj} < k_{lj})  then
            label[j] := 1;  p(j) := l;  Q := Q ∪ {j};
          endif
        endfor
        for each  i ∈ BS(l)  do
          if  (label[i] = 0  and  x_{il} > 0)  then
            label[i] := 1;  p(i) := l;  Q := Q ∪ {i};
          endif
        endfor
        if  (label[d] = 1)  then break;
5    endwhile
6    ⋮      ⋮      ⋮;
```

Ford-Fulkerson algorithm, ④

```
function search-augmenting-path (V, A, x, s, d)
1    ⋮      ⋮      ⋮;
...     ...     ...
6    if (label[d] = 0) then return(∅, +∞);
7    j := d;
8    while (j ≠ s) do
        i := p(j);
        if ((i, j) ∈ A) then
          P := P ∪ {(i, j)};
          if (δ(P) > k_ij − x_ij) then δ(P) := k_ij − x_ij;
        else /* case (j, i) ∈ A */
          P := P ∪ {(j, i)};
          if (δ(P) > x_ji) then δ(P) := x_ji;
        endif
        j := i;
9    endwhile
10   return(P, δ(P));
end search-augmenting-path
```


Paola Festa - DMA, University of Napoli FEDERICO II

E-mail: paola.festa@unina.it - Web: http://www.dma.unina.it/∼festa

Ford-Fulkerson algorithm, ④

```
subroutine search-augmenting-path (V, A, x, s, d)
    ⋮      ⋮      ⋮;
...     ...     ...
    if (label[d] = 0) then return(∅, +∞);
    j := d;
    while (j ≠ s) do
        i := p(j);
        if ((i, j) ∈ A) then
            P := P ∪ {(i, j)};
            if (δ(P) > k_ij − x_ij) then δ(P) := k_ij − x_ij;
        else /* case (j, i) ∈ A */
            P := P ∪ {(j, i)};
            if (δ(P) > x_ji) then δ(P) := x_ji;
        endif
        j := i;
    endwhile            /* O(|A|) */
    return(P, δ(P));
end search-augmenting-path
```

Ford-Fulkerson: complexity, ①

algorithm `ford-fulkerson`(V, A, x, s, d)
1 $(P, \delta(P)) :=$ `search-augmenting-path`(V, A, x, s, d); /*$O(|A|)$*/
2 **if** $(P = \emptyset)$ **then return**$(x, \varphi(\{s\}))$;
3 **do**
  if $(\delta(P) < +\infty)$ **then**
    for each $(i, j) \in F$ **do** /* F archi forward di P */
      $x_{ij} := x_{ij} + \delta(P)$;
      $k_{ij} := k_{ij} - \delta(P)$;
    endfor
    for each $(i, j) \in B$ **do** /* B archi backward di P */
      $x_{ij} := x_{ij} - \delta(P)$;
      $k_{ij} := k_{ij} + \delta(P)$;
    endfor
  else return$(x, +\infty)$; /* caso $\delta(P) = +\infty$: max flusso ill. */
  endif
  $(P, \delta(P)) :=$ `search-augmenting-path`(V, A, x, s, d); /*$O(|A|)$*/
4 **while** $(P \neq \emptyset)$ /*How many times?*/
5 **return** $(x, \varphi(\{s\}))$;
end `ford-fulkerson`

Ford-Fulkerson: complexity, ②

`ford-fulkerson`: **nr of iterations in the worst case**.

Ford-Fulkerson: complexity, ②

`ford-fulkerson`: **nr of iterations in the worst case**.

At each iteration, $\varphi_0 = \varphi(\{s\})$ is augmented in a strictly monotone manner and the increment is $\delta(P) > 0$.

Ford-Fulkerson: complexity, ②

`ford-fulkerson`: **nr of iterations in the worst case**.

At each iteration, $\varphi_0 = \varphi(\{s\})$ is augmented in a strictly monotone manner and the increment is $\delta(P) > 0$.

W.l.g., we suppose that

$$\forall\,(i,j) \in A,\ k_{ij} \in \mathbb{Z}^+ \cup \{0\} \quad \Longrightarrow \quad \delta(P) \in \mathbb{Z}^+.$$

Therefore, the max nr of iterations is given by φ_0^*.

Ford-Fulkerson: complexity, ②

`ford-fulkerson`: **nr of iterations in the worst case**.

At each iteration, $\varphi_0 = \varphi(\{s\})$ is augmented in a strictly monotone manner and the increment is $\delta(P) > 0$.

W.l.g., we suppose that

$$\forall\,(i,j) \in A,\ k_{ij} \in \mathbb{Z}^+ \cup \{0\} \quad \Longrightarrow \quad \delta(P) \in \mathbb{Z}^+.$$

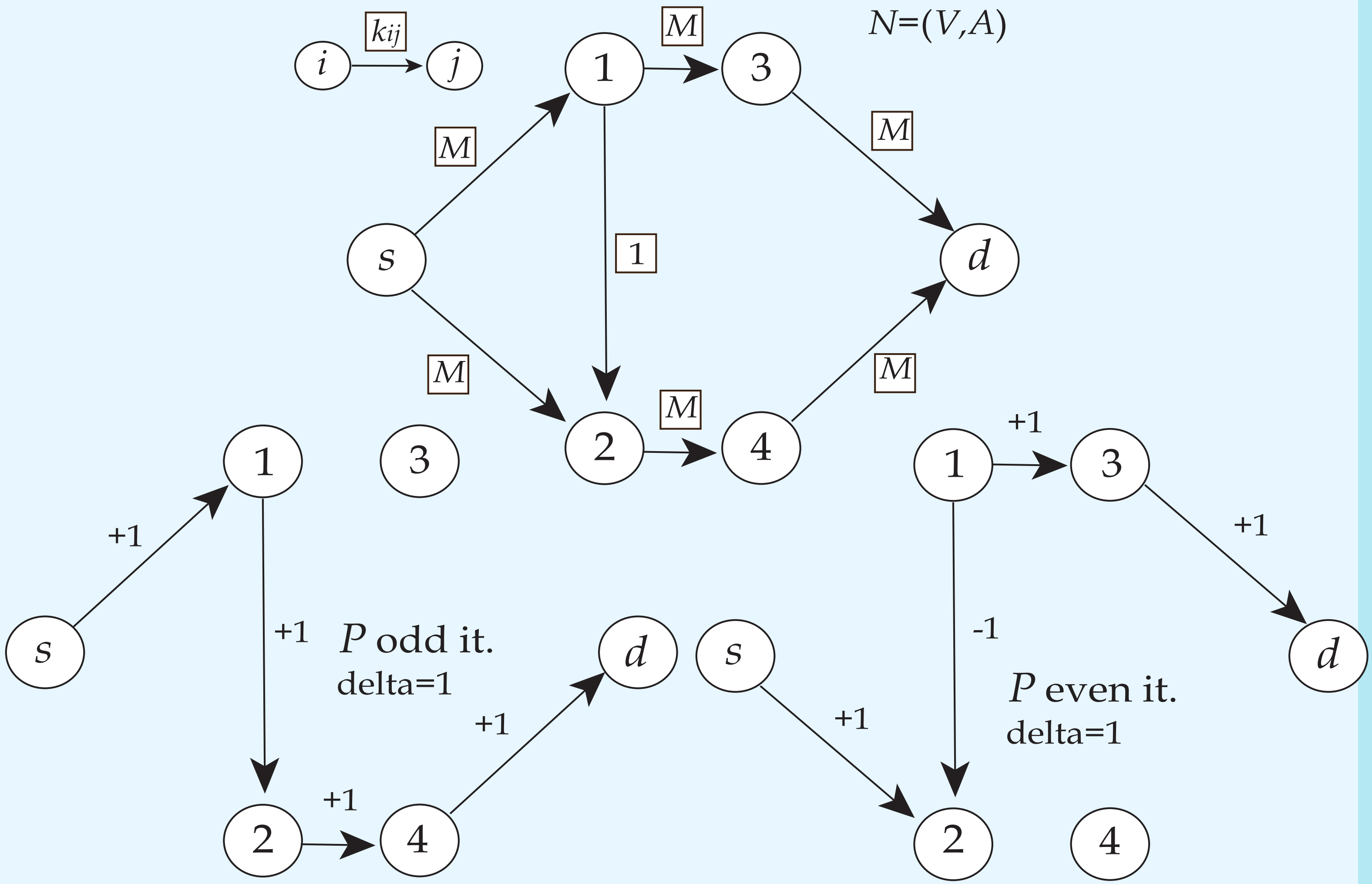
Therefore, the max nr of iterations is given by φ_0^*. Moreover,

$$\varphi_0^* = \varphi^*(\{s\}) \overset{\text{Th}}{\leq} K(\{s\}) = O(|A| \cdot k_{\max}),$$

where $k_{\max} = \max_{(i,j)\in A} k_{ij}$.

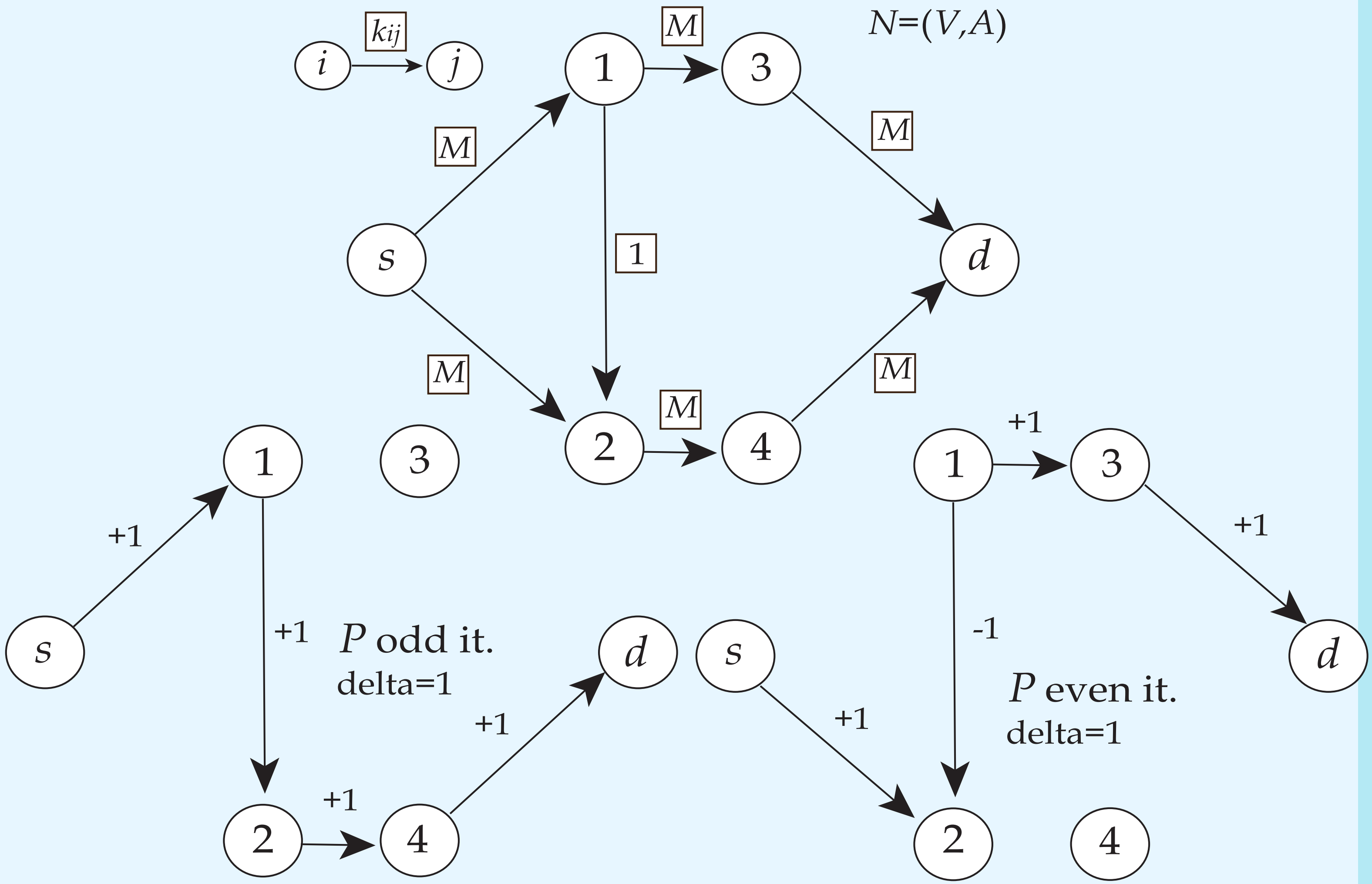
Summarizing, the **worst case computational complexity** of the algorithm is $O(|A|^2 \cdot k_{\max})$.

Instance where `ford-fulkerson` is not efficient:



Ford-Fulkerson: complexity, ③

Instance where `ford-fulkerson` is not efficient:



Nr of iterations: $2M$.

A simpler (not more efficient) implementation of Ford-Fulkerson algorithm

Paola Festa - DMA, University of Napoli FEDERICO II

E-mail: paola.festa@unina.it - Web: http://www.dma.unina.it/∼festa

At each main iteration (**do while loop** lines 3-4), **each node** $j \in V$ is associated with a **label with 2 entries**

$$[\pm p(j), \epsilon_j],$$

where

- ➜ $p(j)$ is the **predecessor node of** j **along the augmenting path** P from s to j;
- ➜ ϵ_j is the **increment of flow at node** j **along** P.

Ford-Fulkerson 2, ①

At each main iteration (**do while loop** lines 3-4), **each node $j \in V$** is associated with a **label with 2 entries**

$$[\pm p(j), \epsilon_j],$$

where

- ➜ $p(j)$ is the **predecessor node of j along the augmenting path** P from s to j;
- ➜ ϵ_j is the **increment of flow at node j along** P.

- ✍ "+" $\Longrightarrow$ j is reached traversing the forward arc $(p(j), j) \in A$;
- ✍ "−" $\Longrightarrow$ j is reached traversing the backward arc $(j, p(j)) \in A$.

Ford-Fulkerson 2, ①

At each main iteration (**do while loop** lines 3-4), **each node** $j \in V$ is associated with a **label with 2 entries**

$$[\pm p(j), \epsilon_j],$$

where

- ➜ $p(j)$ is the **predecessor node of** j **along the augmenting path** P from s to j;
- ➜ ϵ_j is the **increment of flow at node** j **along** P.

- ✍ "+"$\Longrightarrow$ j is reached traversing the forward arc $(p(j), j) \in A$;
- ✍ "−"$\Longrightarrow$ j is reached traversing the backward arc $(j, p(j)) \in A$.

Note: **initially**, also a trivial flow vector $x_{ij} = 0$, $\forall\ (i,j) \in A$ can be used.

Ford-Fulkerson 2, ②

algorithm `ford-fulkerson2`(V, A, s, d)
1 **for each** $(i,j) \in A$ **do** $x_{ij} := 0$;
2 $\varphi_0 := 0$;
3 **do** /* main loop */
  for each $j \in V$ **do** $p(j) := 0$;
  $[p(s), \epsilon_s] := [+s, +\infty]$; $Q := \{s\}$;
4 **while** $(Q \neq \emptyset$ and $p(d) = 0)$ **do**
   $h :=$ `remove`(Q);
   for each $j \in FS(h)$, $x_{hj} < k_{hj}$ **do** /* non saturated forw arcs */
    if $(p(j) = 0)$ **then**
     $[p(j), \epsilon_j] := [+h, \min\{\epsilon_h, k_{hj} - x_{hj}\}]$; $Q := Q \cup \{j\}$;
    endif
   endfor
   for each $i \in BS(h)$, $x_{ih} > 0$ **do** /* non unloading backw arcs */
    if $(p(i) = 0)$ **then**
     $[p(i), \epsilon_i] := [-h, \min\{\epsilon_h, x_{ih}\}]$; $Q := Q \cup \{i\}$;
    endif
   endfor
5 **endwhile**
... ...

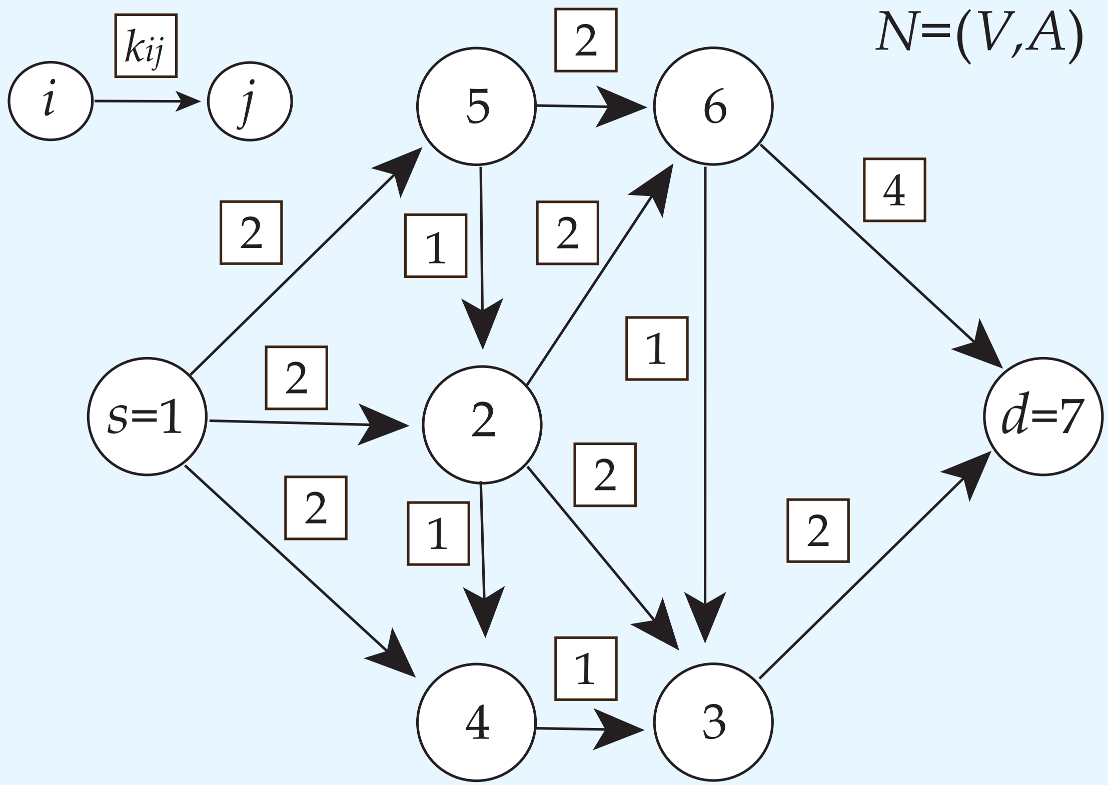
Ford-Fulkerson 2, ②

```
algorithm ford-fulkerson2(V, A, s, d)
...    ...    ...    ...    ...
      if (p(d) ≠ 0) then /* augmenting path P from s to d found */
        δ(P) := ε_d; φ_0 := φ_0 + δ(P); j := d;
        while (j ≠ s) do /* building P */
          i := p(j);
          if (i > 0) then x_ij := x_ij + δ(P);
          else x_j|i| := x_j|i| − δ(P);
          j := |i|;
        endwhile
      endif
   until (p(d) = 0)
   print ''Current flow x is optimal and the cut
          (S*, V \ S*) has minimum capacity,
          where S* = {j ∈ V | p_j ≠ 0}'';
   return(x, φ_0, S*);
end ford-fulkerson2
```

Ford-Fulkerson algorithm: an exercise

Exercise, ①

Exercise. Find the **maximum flow** from the **source node** $s = 1$ to the **sink node** $d = 7$ in the following network flow $N = (V, A)$.



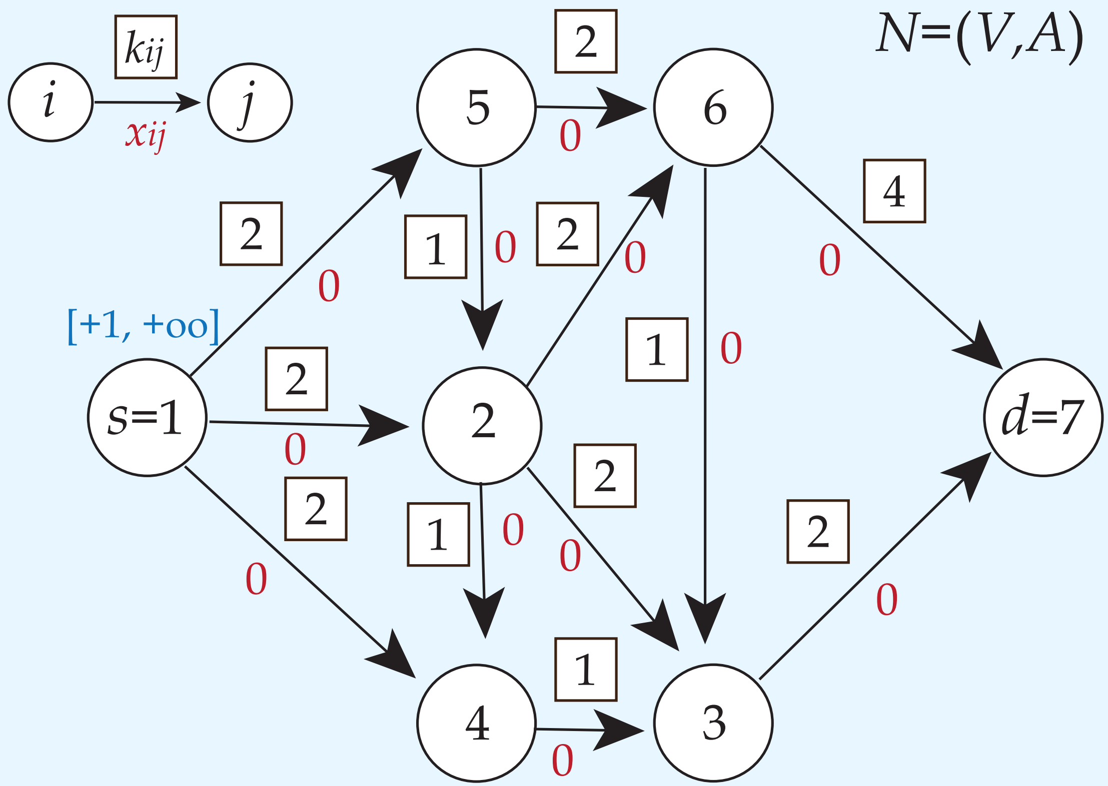
Inizialization:

$p(i) = 0, \forall\, i \in V \setminus \{s\};$ $x_{ij} = 0, \forall\, (i,j) \in A.$

$\varphi_0 = 0.$

$[+s, +\infty] = [+1, +\infty].$

$Q = \{1\}.$



Iteration I: (breadth search)

h :=remove(Q);

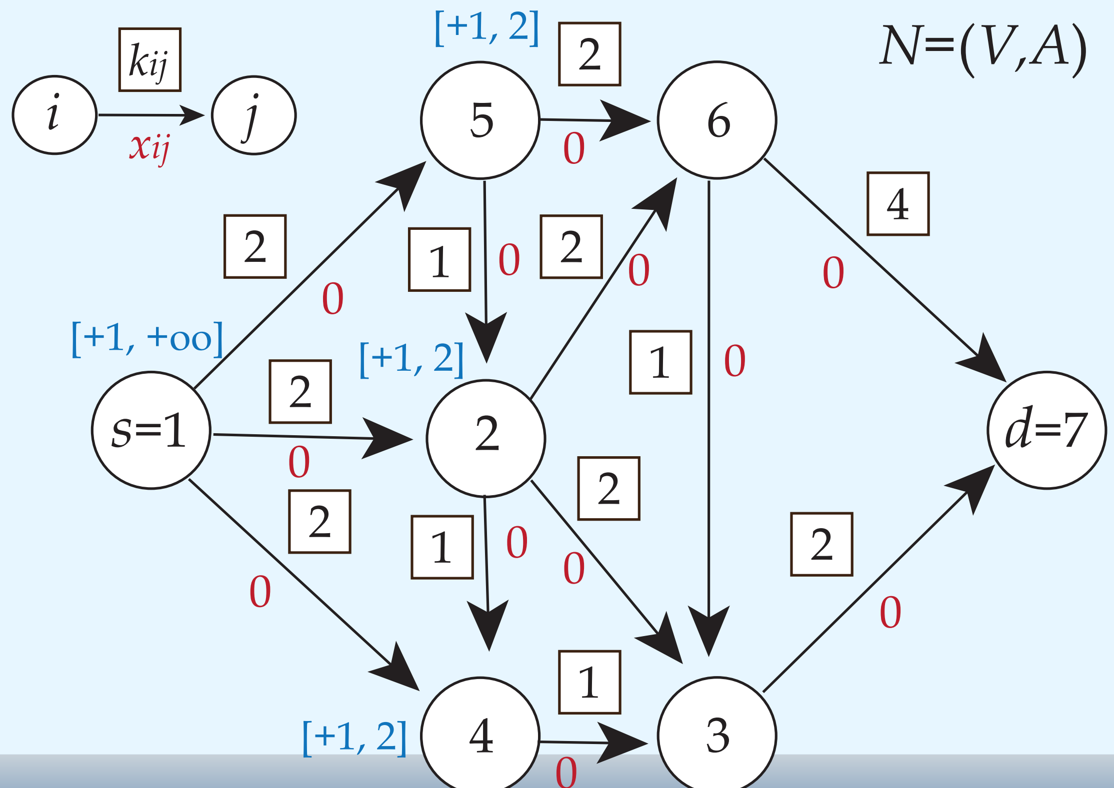
for each $j \in FS(h)$, $x_{hj} < k_{hj}$ **do** /* non saturated forward arcs */

if ($p(j) = 0$) **then** {

$[p(j), \epsilon_j] := [+h, \min\{\epsilon_h, k_{hj} - x_{hj}\}]; Q := Q \cup \{j\};$}

for each $i \in BS(h)$, $x_{ih} > 0$ **do** /* non unloading backward arcs */

if ($p(i) = 0$) **then** { $[p(i), \epsilon_i] := [-h, \min\{\epsilon_h, x_{ih}\}]; Q := Q \cup \{i\};$ }



Iteration I: (breadth search)

h :=remove(Q);

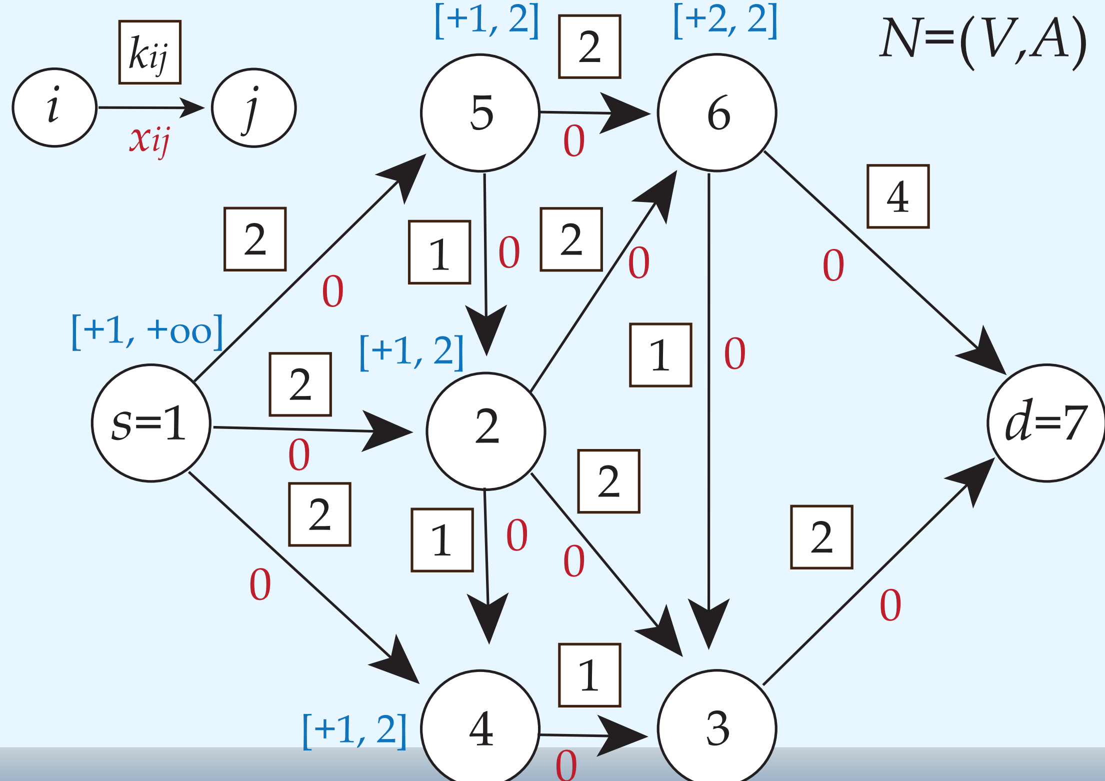
for each $j \in FS(h)$, $x_{hj} < k_{hj}$ **do** /* non saturated forward arcs */

  if $(p(j) = 0)$ **then** {

    $[p(j), \epsilon_j] := [+h, \min\{\epsilon_h, k_{hj} - x_{hj}\}]; Q := Q \cup \{j\};$}

for each $i \in BS(h)$, $x_{ih} > 0$ **do** /* non unloading backward arcs */

  if $(p(i) = 0)$ **then** { $[p(i), \epsilon_i] := [-h, \min\{\epsilon_h, x_{ih}\}]; Q := Q \cup \{i\};$ }



Exercise, ③

Iteration I: (breadth search)

h :=remove(Q);

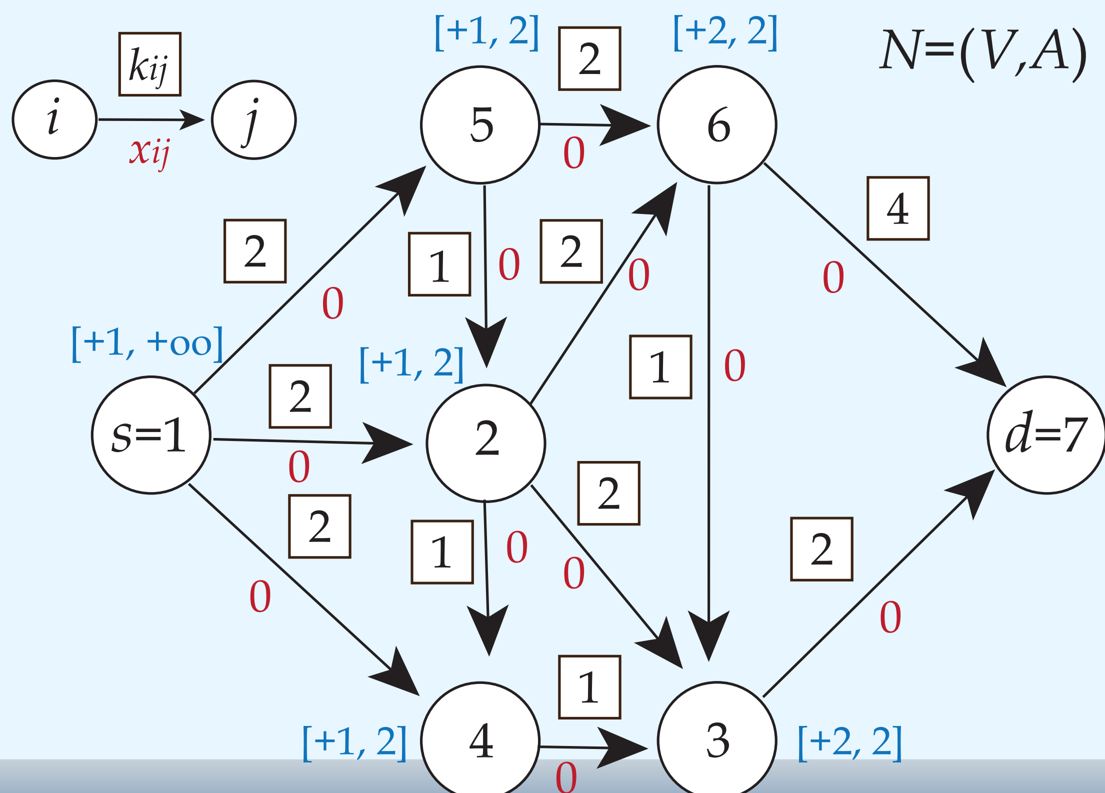
for each $j \in FS(h), x_{hj} < k_{hj}$ **do** /* non saturated forward arcs */

  if $(p(j) = 0)$ **then** {

    $[p(j), \epsilon_j] := [+h, \min\{\epsilon_h, k_{hj} - x_{hj}\}]; Q := Q \cup \{j\};$}

for each $i \in BS(h), x_{ih} > 0$ **do** /* non unloading backward arcs */

  if $(p(i) = 0)$ **then** { $[p(i), \epsilon_i] := [-h, \min\{\epsilon_h, x_{ih}\}]; Q := Q \cup \{i\};$ }



Exercise, ③

Iteration I: (breadth search)

h :=remove(Q);

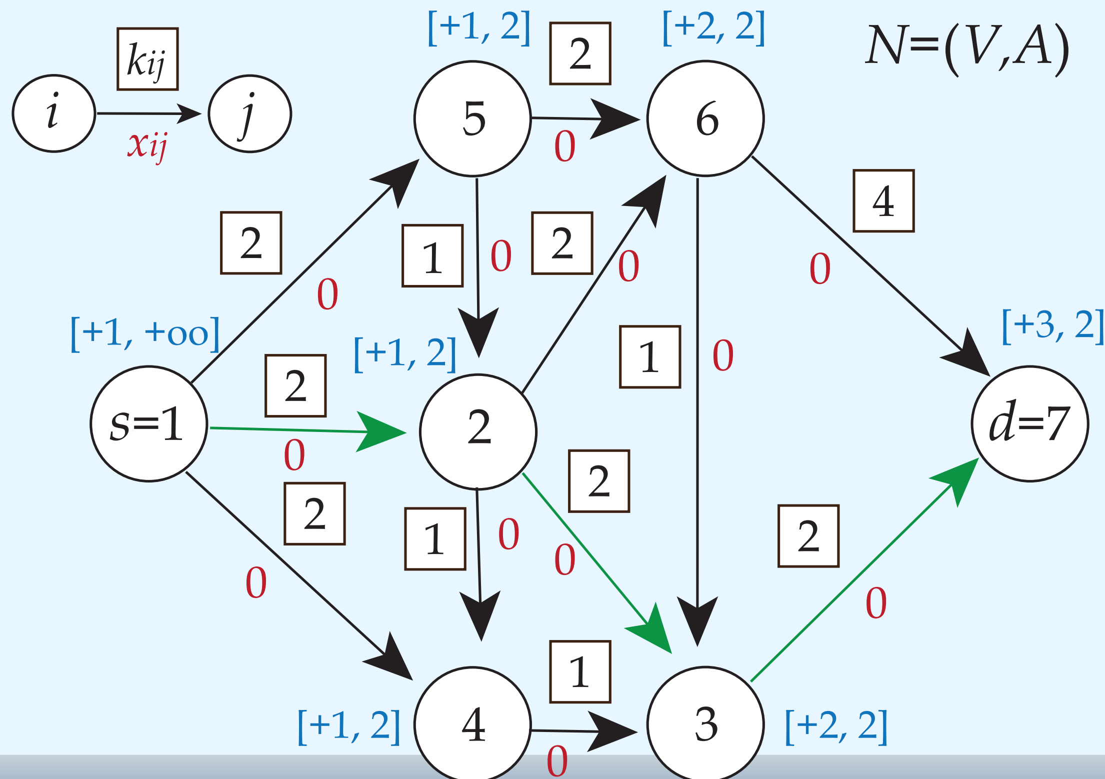
for each $j \in FS(h), x_{hj} < k_{hj}$ **do** /* non saturated forward arcs */

if $(p(j) = 0)$ **then** {

$[p(j), \epsilon_j] := [+h, \min\{\epsilon_h, k_{hj} - x_{hj}\}]; Q := Q \cup \{j\};$}

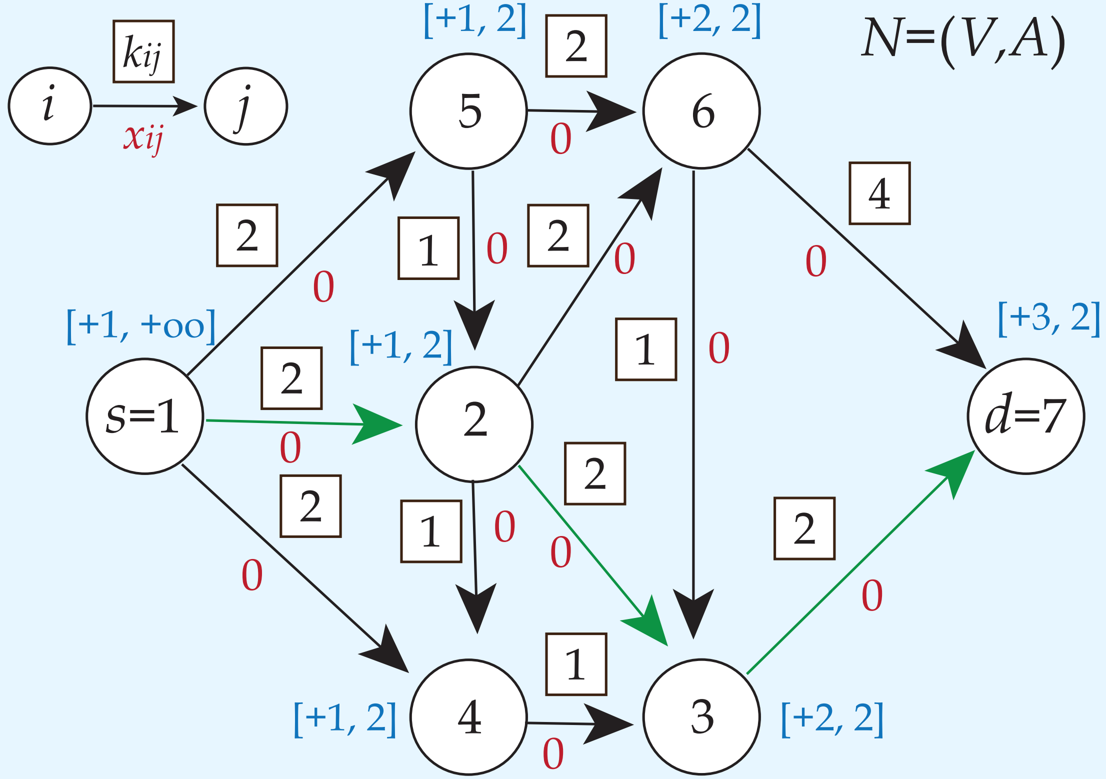
for each $i \in BS(h), x_{ih} > 0$ **do** /* non unloading backward arcs */

if $(p(i) = 0)$ **then** { $[p(i), \epsilon_i] := [-h, \min\{\epsilon_h, x_{ih}\}]; Q := Q \cup \{i\};$ }



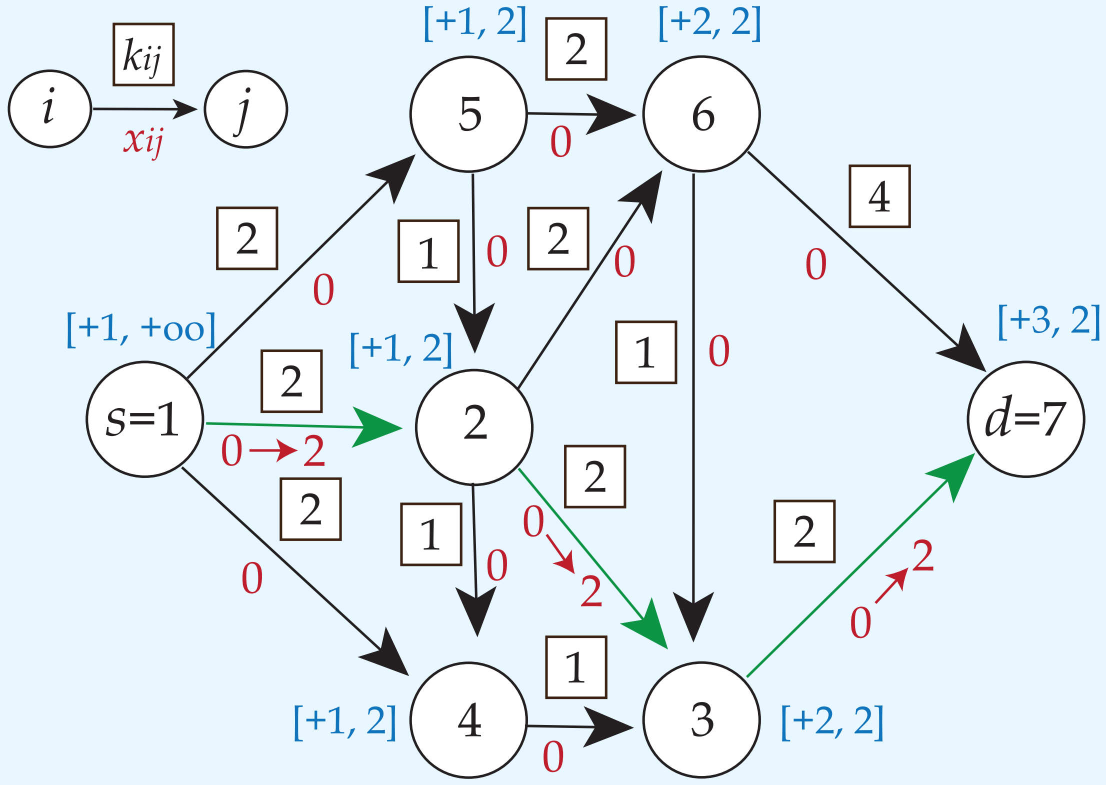
Iteration I:

Augmenting path $P = \{(1,2),(2,3),(3,7)\}$.
$\delta(P) = \epsilon_d = 2$.



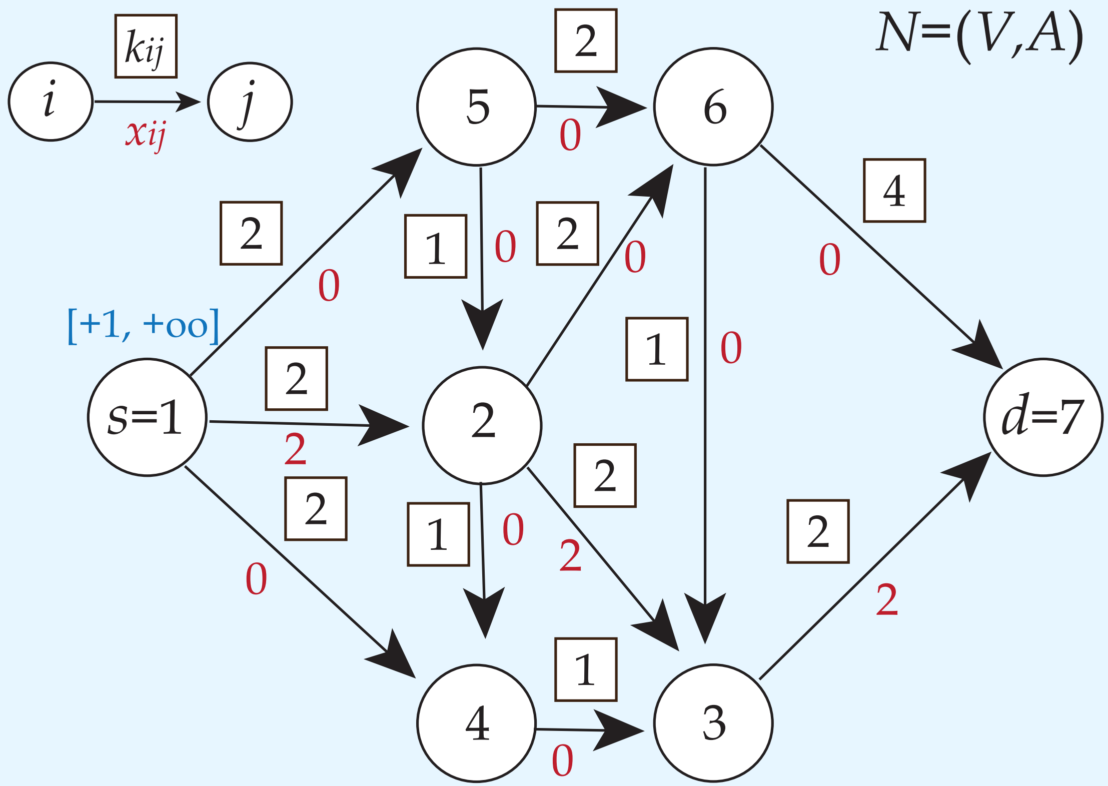
Iteration I:

Increasing/decreasing of $\delta(P) = \epsilon_d = 2$ along the augmenting path $P = \{(1,2),(2,3),(3,7)\}$.



End of Iteration I:

$\varphi_0 = 2.$



Iteration II: (breadth search)

h :=remove(Q);

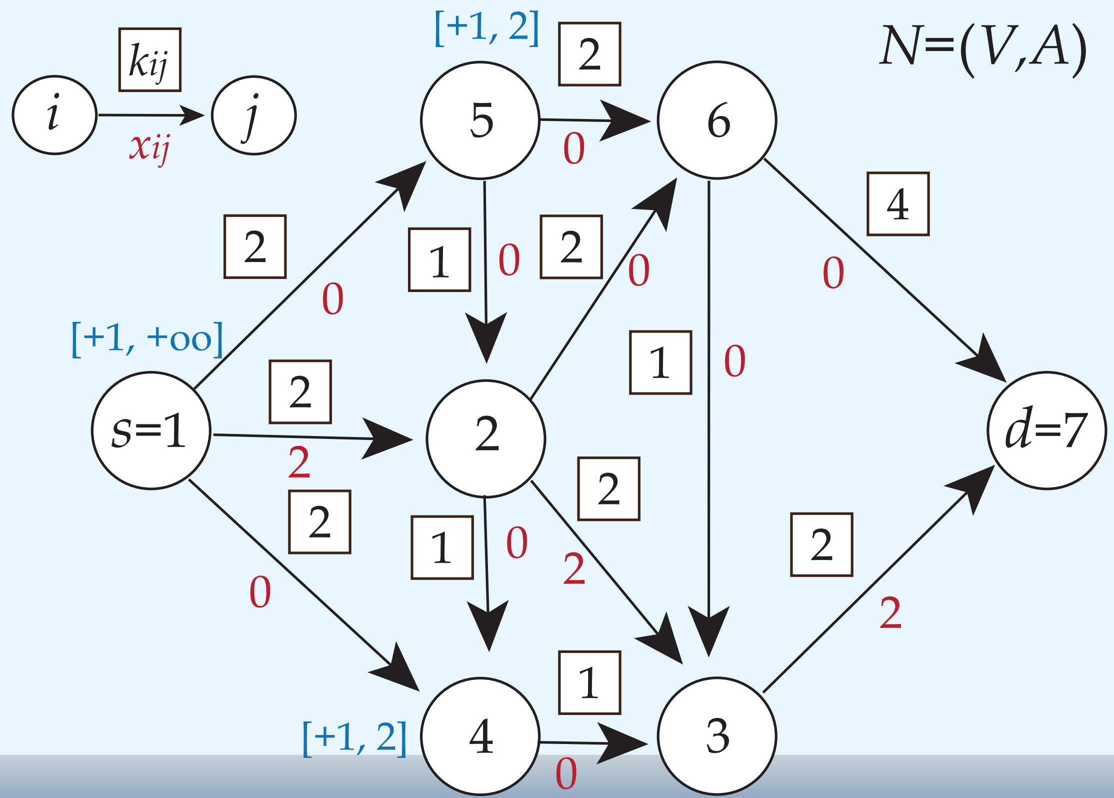
for each $j \in FS(h)$, $x_{hj} < k_{hj}$ **do** /* non saturated forward arcs */

  if $(p(j) = 0)$ **then** {

    $[p(j), \epsilon_j] := [+h, \min\{\epsilon_h, k_{hj} - x_{hj}\}]; Q := Q \cup \{j\};$}

for each $i \in BS(h)$, $x_{ih} > 0$ **do** /* non unloading backward arcs */

  if $(p(i) = 0)$ **then** { $[p(i), \epsilon_i] := [-h, \min\{\epsilon_h, x_{ih}\}]; Q := Q \cup \{i\};$ }



Exercise, ④

Iteration II: (breadth search)

h :=remove(Q);

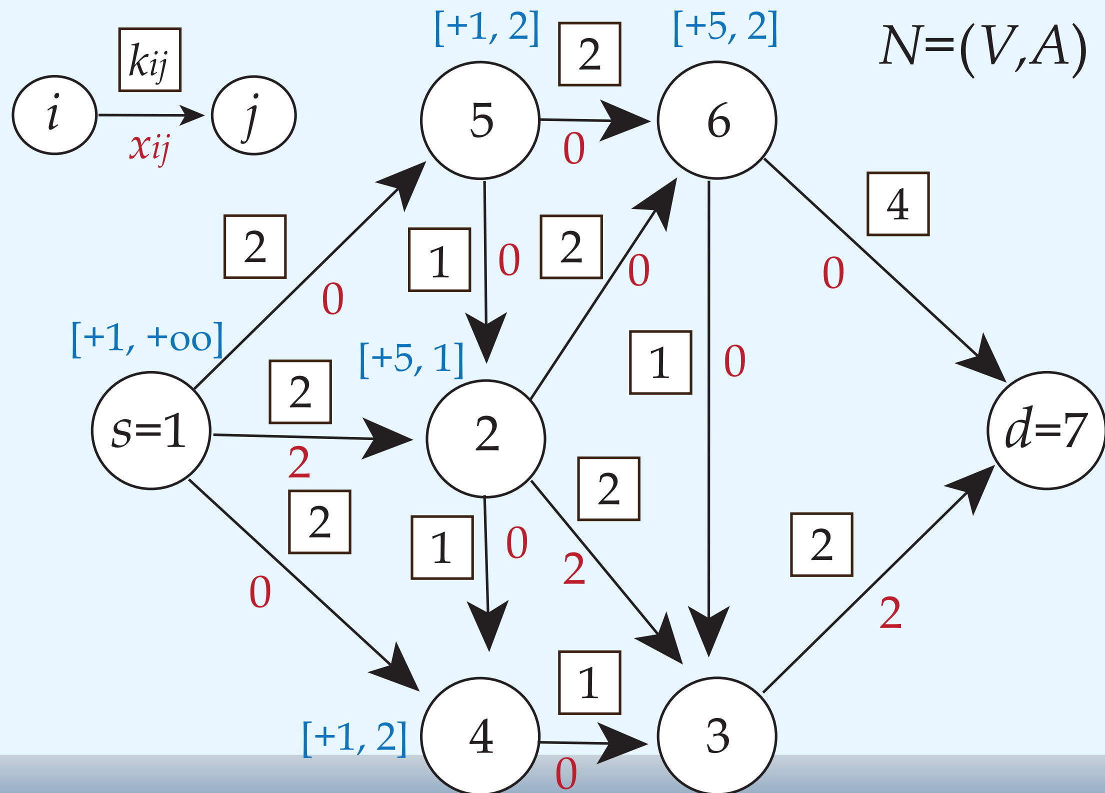
for each $j \in FS(h)$, $x_{hj} < k_{hj}$ **do** /* non saturated forward arcs */

  if $(p(j) = 0)$ **then** {

    $[p(j), \epsilon_j] := [+h, \min\{\epsilon_h, k_{hj} - x_{hj}\}]; Q := Q \cup \{j\};$}

for each $i \in BS(h)$, $x_{ih} > 0$ **do** /* non unloading backward arcs */

  if $(p(i) = 0)$ **then** { $[p(i), \epsilon_i] := [-h, \min\{\epsilon_h, x_{ih}\}]; Q := Q \cup \{i\};$ }



Iteration II: (breadth search)

h :=remove(Q);

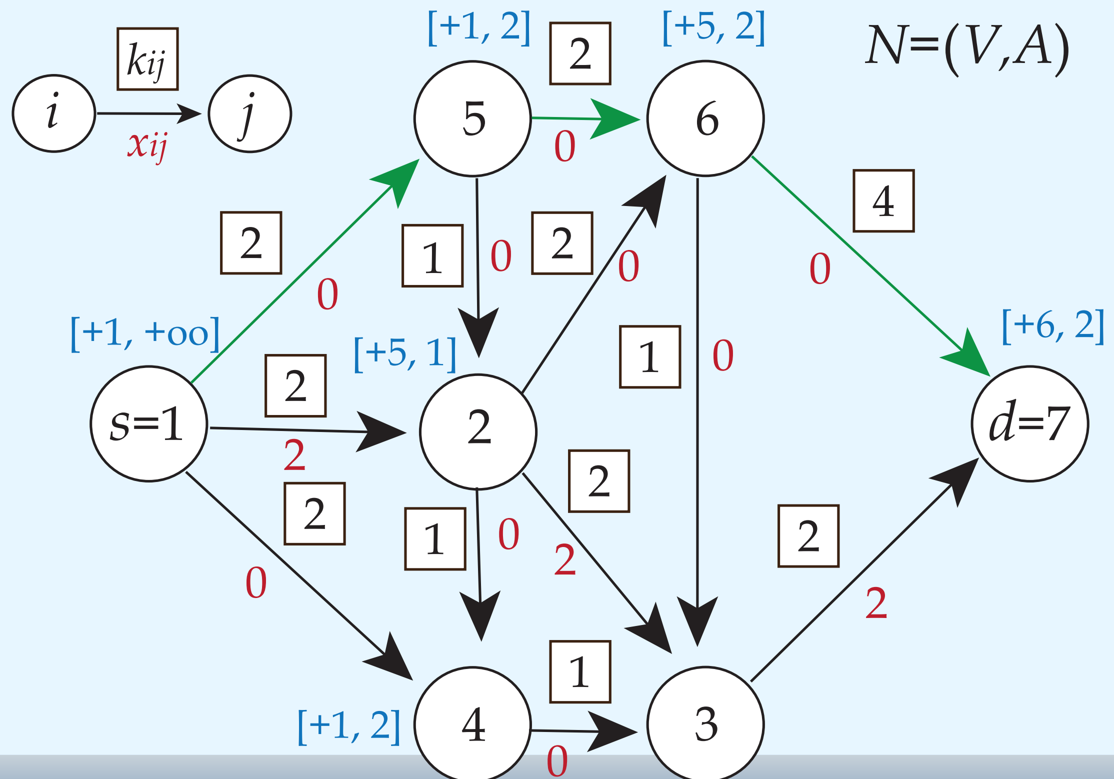
for each $j \in FS(h)$, $x_{hj} < k_{hj}$ **do** /* non saturated forward arcs */

if $(p(j) = 0)$ **then** {

$[p(j), \epsilon_j] := [+h, \min\{\epsilon_h, k_{hj} - x_{hj}\}]; Q := Q \cup \{j\};$}

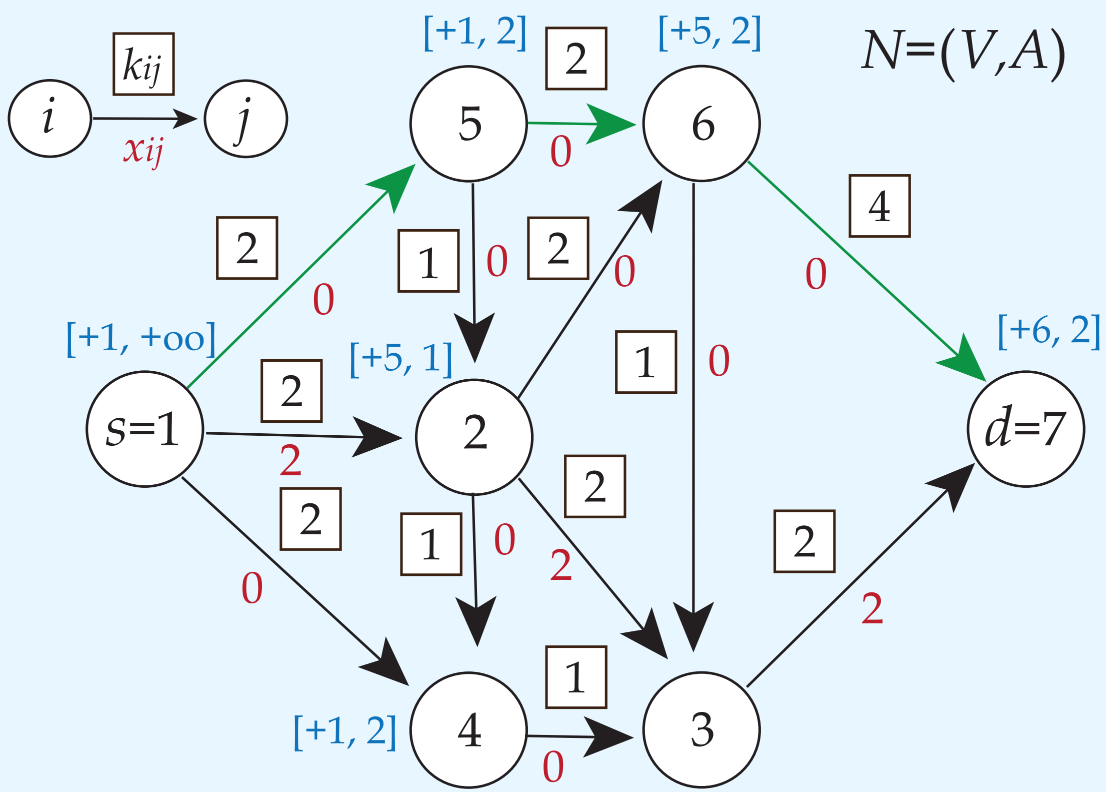
for each $i \in BS(h)$, $x_{ih} > 0$ **do** /* non unloading backward arcs */

if $(p(i) = 0)$ **then** { $[p(i), \epsilon_i] := [-h, \min\{\epsilon_h, x_{ih}\}]; Q := Q \cup \{i\};$ }



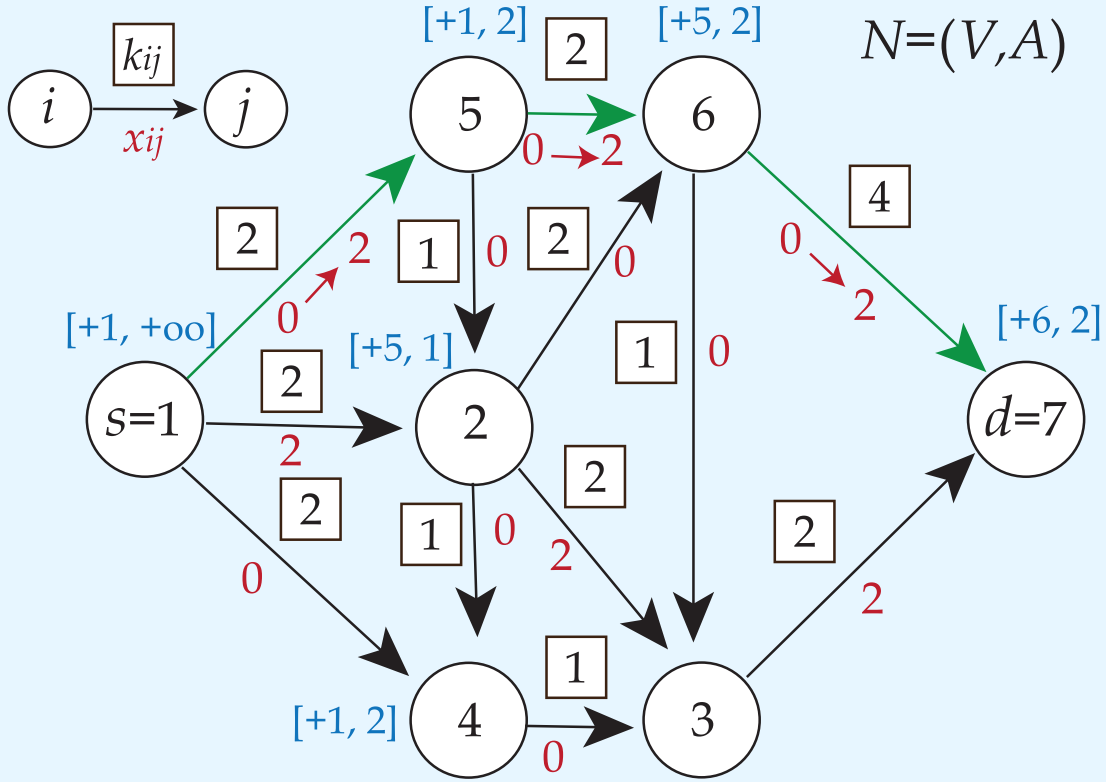
Iteration II:

Augmenting path $P = \{(1,5),(5,6),(6,7)\}$.
$\delta(P) = \epsilon_d = 2$.



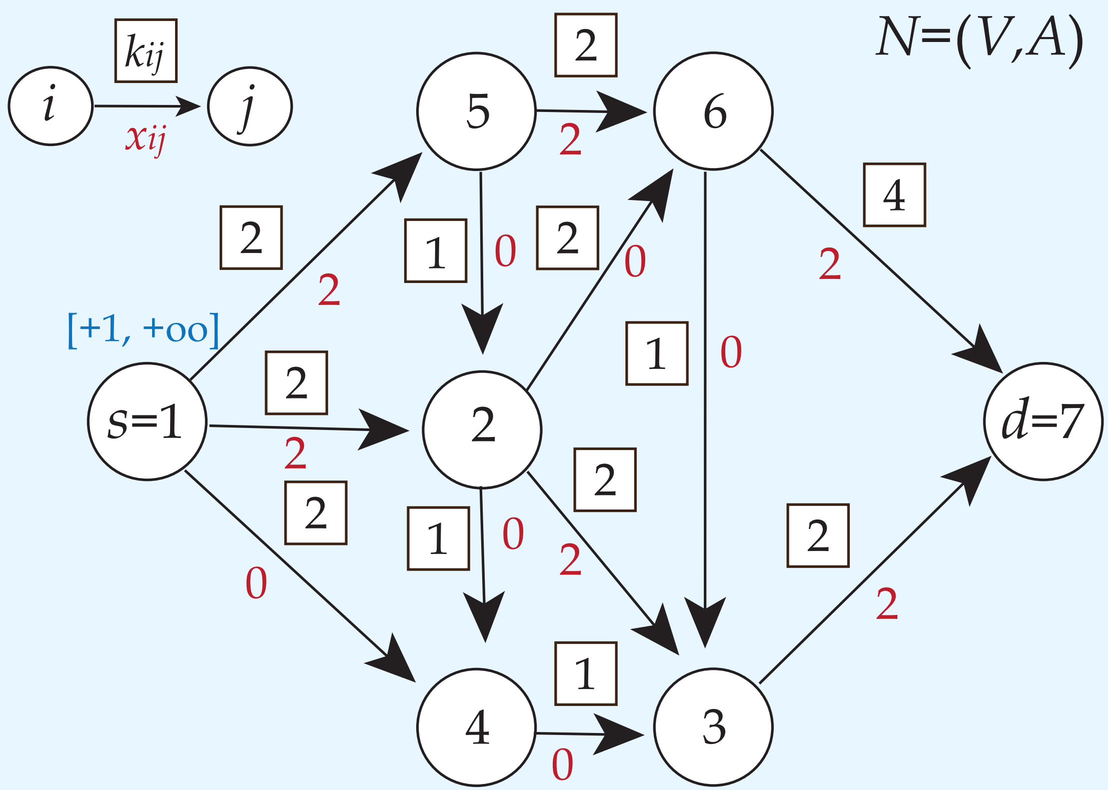
Iteration II:

Increasing/decreasing of $\delta(P) = \epsilon_d = 2$ along the augmenting path $P = \{(1,5),(5,6),(6,7)\}$.



End of Iteration II:

$\varphi_0 = 4.$



Iteration III: (breadth search)

h :=remove(Q);

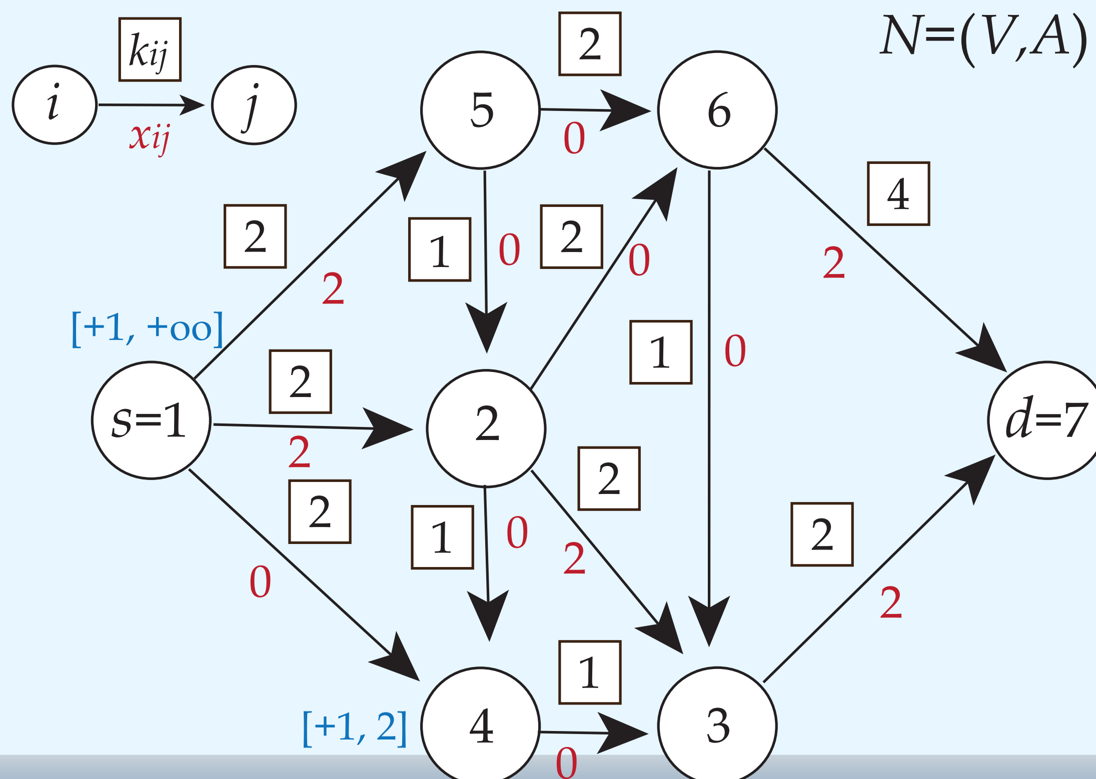
for each $j \in FS(h)$, $x_{hj} < k_{hj}$ **do** /* non saturated forward arcs */

  if $(p(j) = 0)$ **then** {

    $[p(j), \epsilon_j] := [+h, \min\{\epsilon_h, k_{hj} - x_{hj}\}]; Q := Q \cup \{j\};$}

for each $i \in BS(h)$, $x_{ih} > 0$ **do** /* non unloading backward arcs */

  if $(p(i) = 0)$ **then** { $[p(i), \epsilon_i] := [-h, \min\{\epsilon_h, x_{ih}\}]; Q := Q \cup \{i\};$ }



Exercise, ④

Iteration III: (breadth search)

h :=remove(Q);

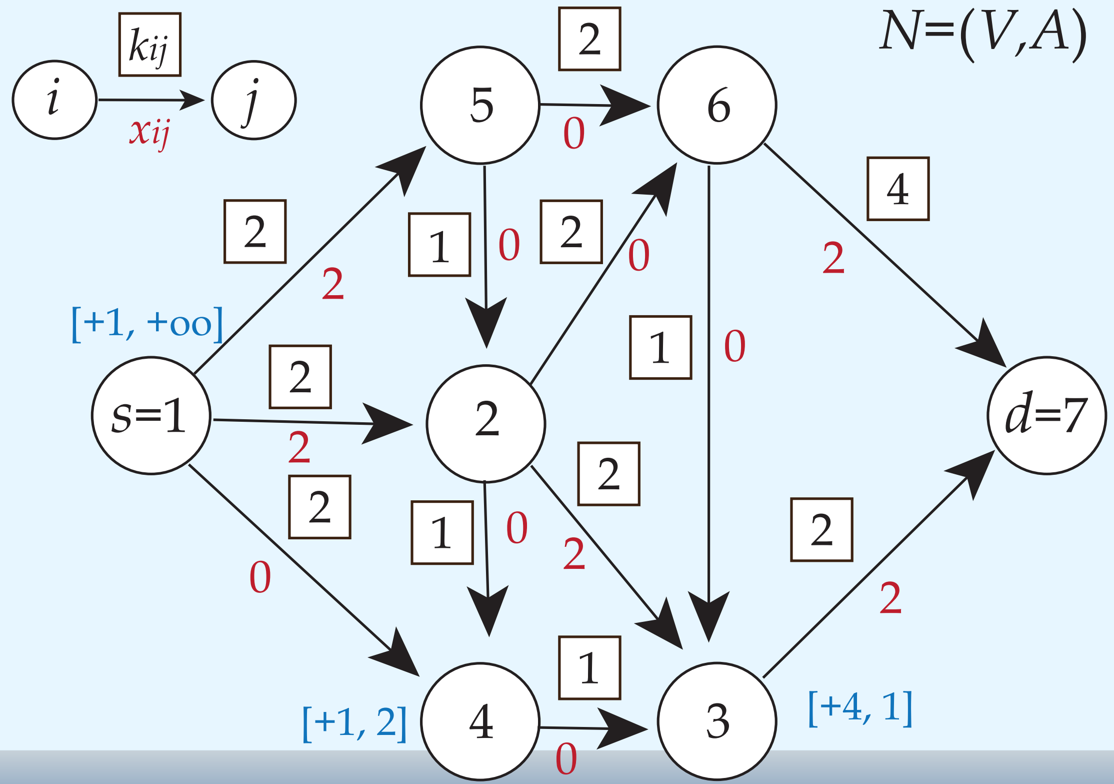
for each $j \in FS(h), x_{hj} < k_{hj}$ **do** /* non saturated forward arcs */

  if $(p(j) = 0)$ **then** $\{$

    $[p(j), \epsilon_j] := [+h, \min\{\epsilon_h, k_{hj} - x_{hj}\}]; Q := Q \cup \{j\};\}$

for each $i \in BS(h), x_{ih} > 0$ **do** /* non unloading backward arcs */

  if $(p(i) = 0)$ **then** $\{ [p(i), \epsilon_i] := [-h, \min\{\epsilon_h, x_{ih}\}]; Q := Q \cup \{i\}; \}$



Iteration III: (breadth search)

h :=remove(Q);

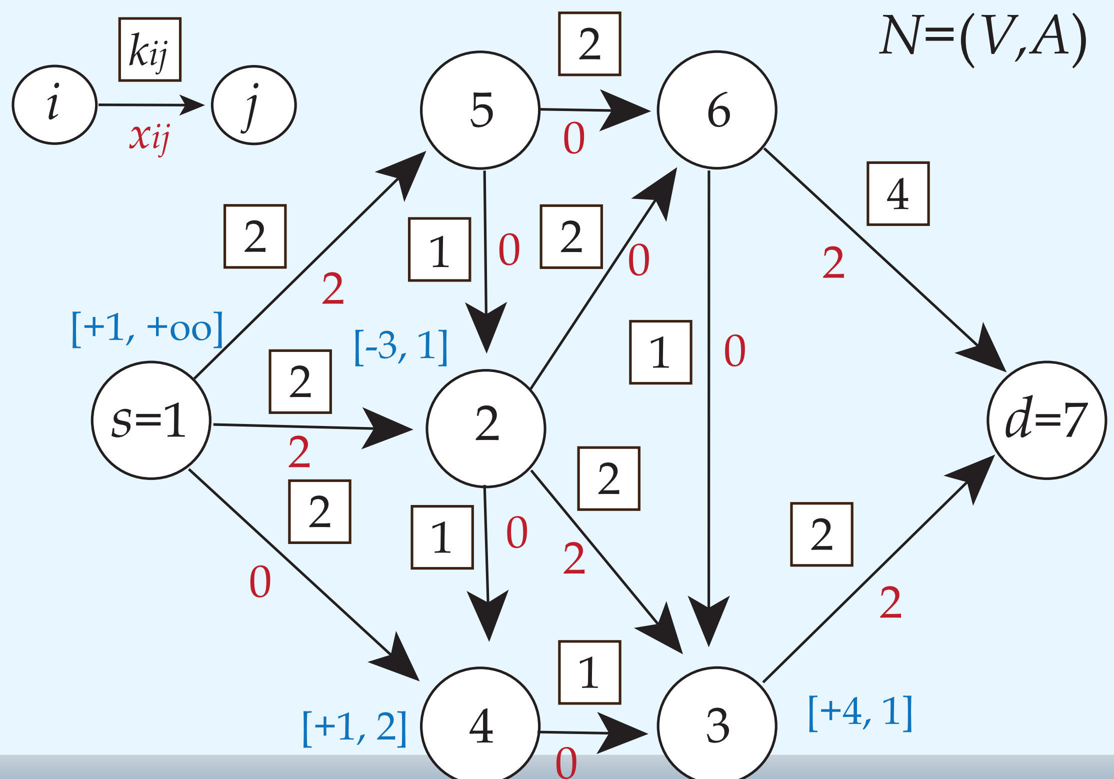
for each $j \in FS(h), x_{hj} < k_{hj}$ **do** /* non saturated forward arcs */

if $(p(j) = 0)$ **then** {

$[p(j), \epsilon_j] := [+h, \min\{\epsilon_h, k_{hj} - x_{hj}\}]; Q := Q \cup \{j\};$}

for each $i \in BS(h), x_{ih} > 0$ **do** /* non unloading backward arcs */

if $(p(i) = 0)$ **then** { $[p(i), \epsilon_i] := [-h, \min\{\epsilon_h, x_{ih}\}]; Q := Q \cup \{i\};$ }



Iteration III: (breadth search)

h :=remove(Q);

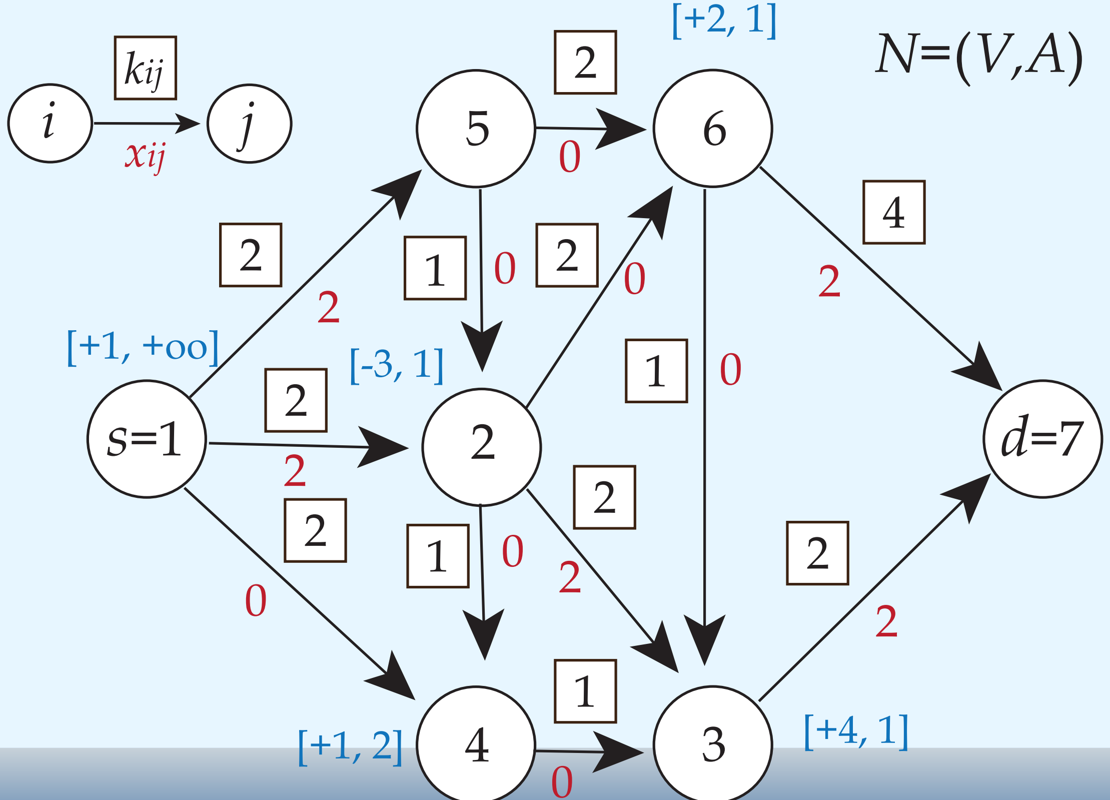
for each $j \in FS(h), x_{hj} < k_{hj}$ **do** /* non saturated forward arcs */

if $(p(j) = 0)$ **then** {

$[p(j), \epsilon_j] := [+h, \min\{\epsilon_h, k_{hj} - x_{hj}\}]; Q := Q \cup \{j\};$}

for each $i \in BS(h), x_{ih} > 0$ **do** /* non unloading backward arcs */

if $(p(i) = 0)$ **then** { $[p(i), \epsilon_i] := [-h, \min\{\epsilon_h, x_{ih}\}]; Q := Q \cup \{i\};$ }



Iteration III: (breadth search)

h :=remove(Q);

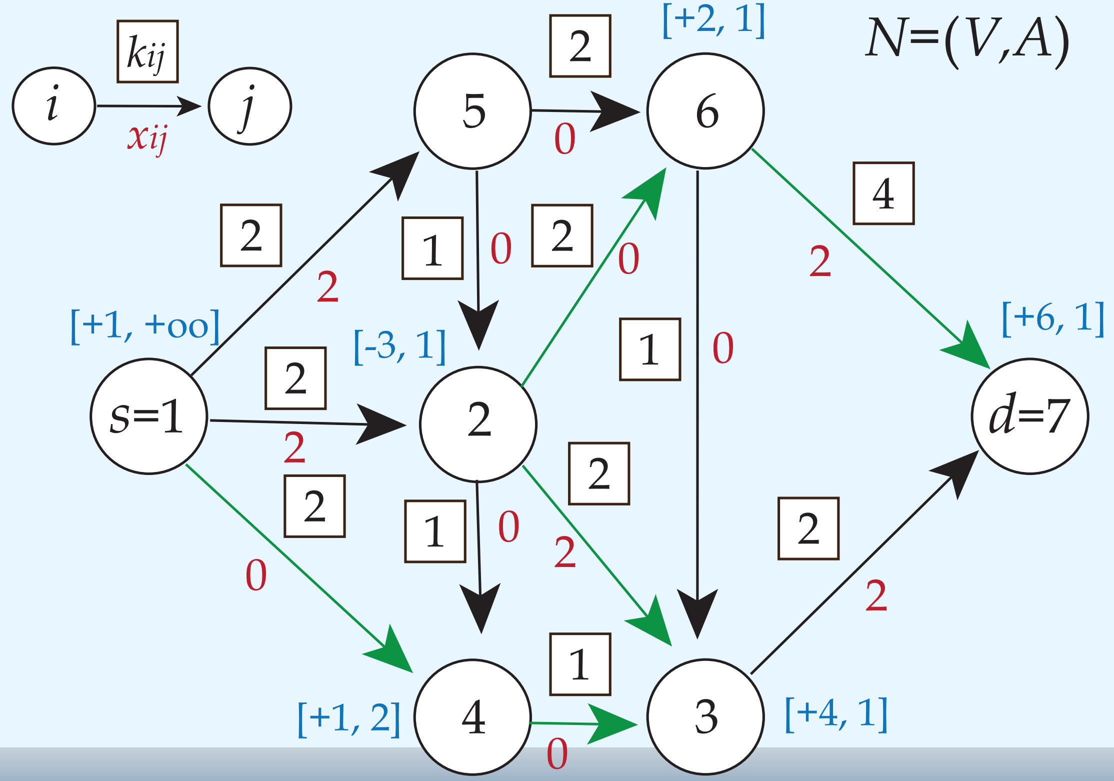
for each $j \in FS(h)$, $x_{hj} < k_{hj}$ **do** /* non saturated forward arcs */

  if ($p(j) = 0$) **then** {

    $[p(j), \epsilon_j] := [+h, \min\{\epsilon_h, k_{hj} - x_{hj}\}]; Q := Q \cup \{j\};$}

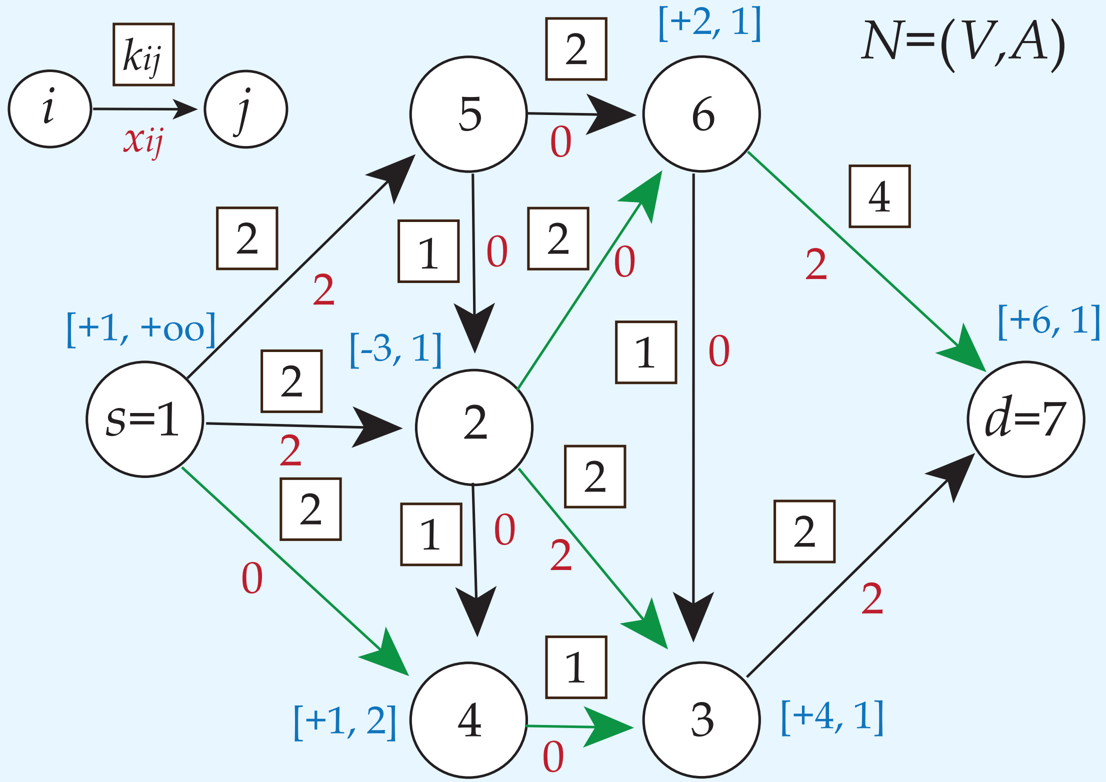
for each $i \in BS(h)$, $x_{ih} > 0$ **do** /* non unloading backward arcs */

  if ($p(i) = 0$) **then** { $[p(i), \epsilon_i] := [-h, \min\{\epsilon_h, x_{ih}\}]; Q := Q \cup \{i\};$ }



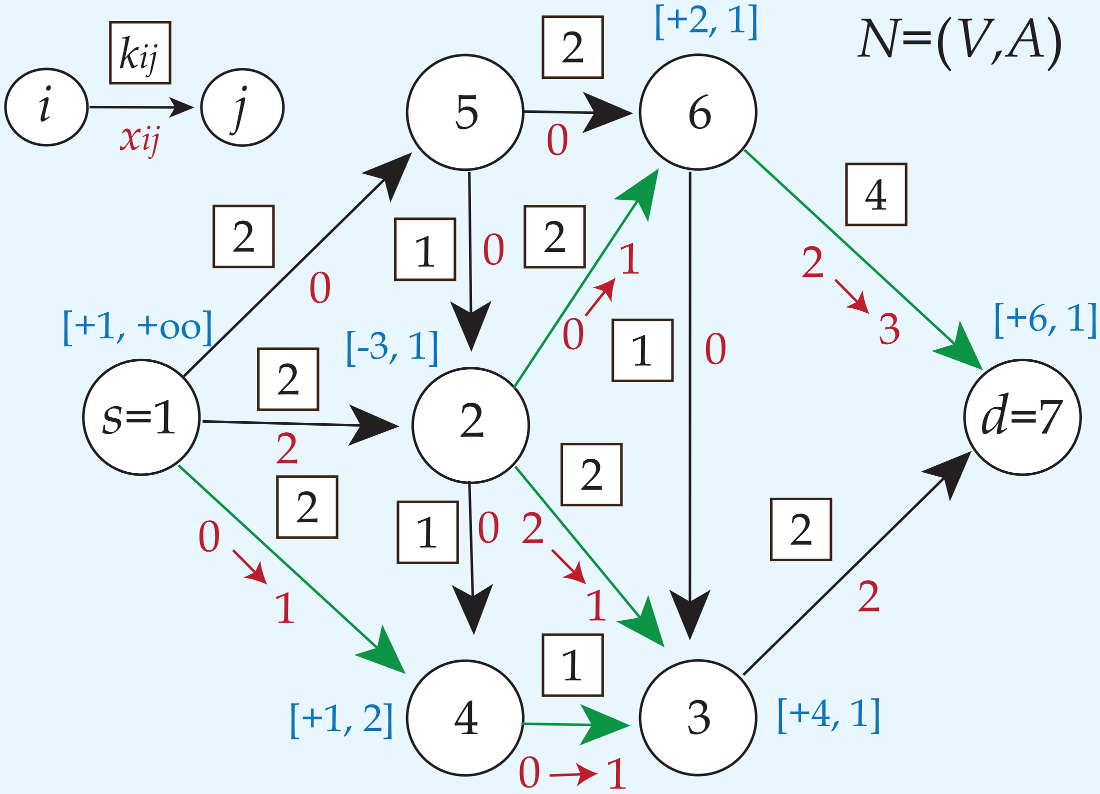
Iteration III:

Augmenting path $P = \{(1,4),(4,3),(2,3),(2,6),(6,7)\}$.
$\delta(P) = \epsilon_d = 1$.



Iteration III:

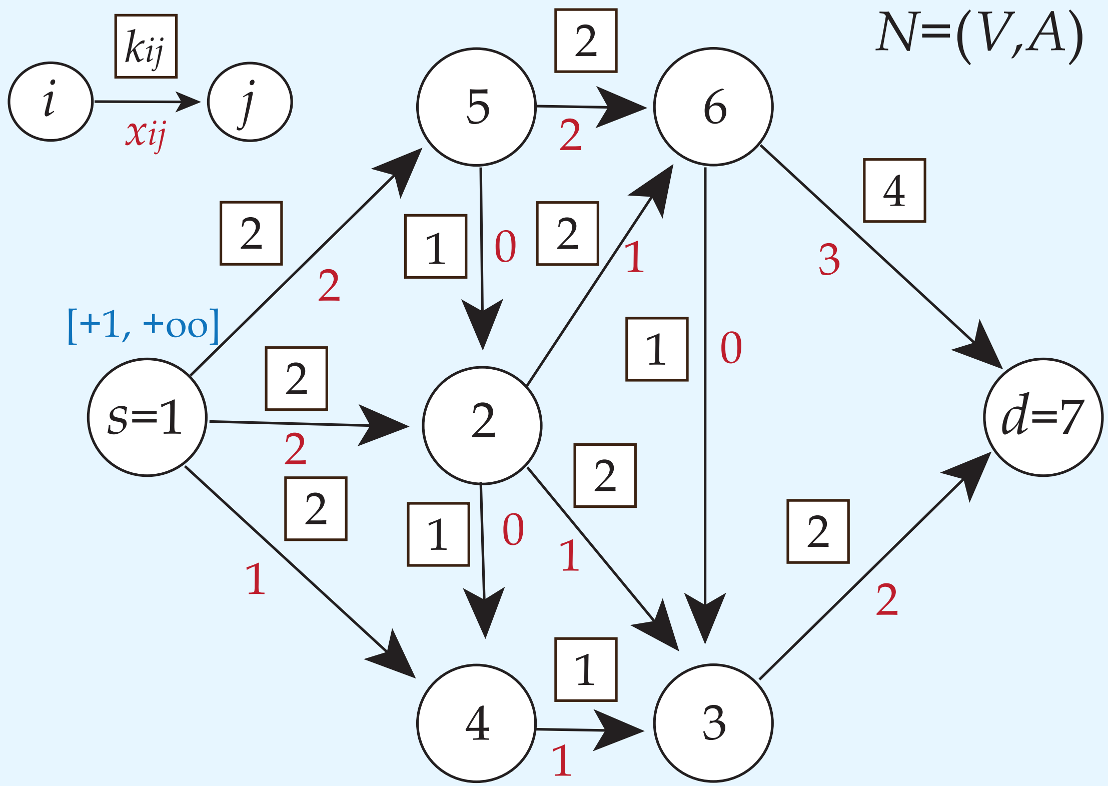
Increasing/decreasing of $\delta(P) = \epsilon_d = 1$ along the augmenting path $P = \{(1,4),(4,3),(2,3),(2,6),(6,7)\}$.



Exercise, ⑤

End of Iteration III:

$\varphi_0 = 5.$

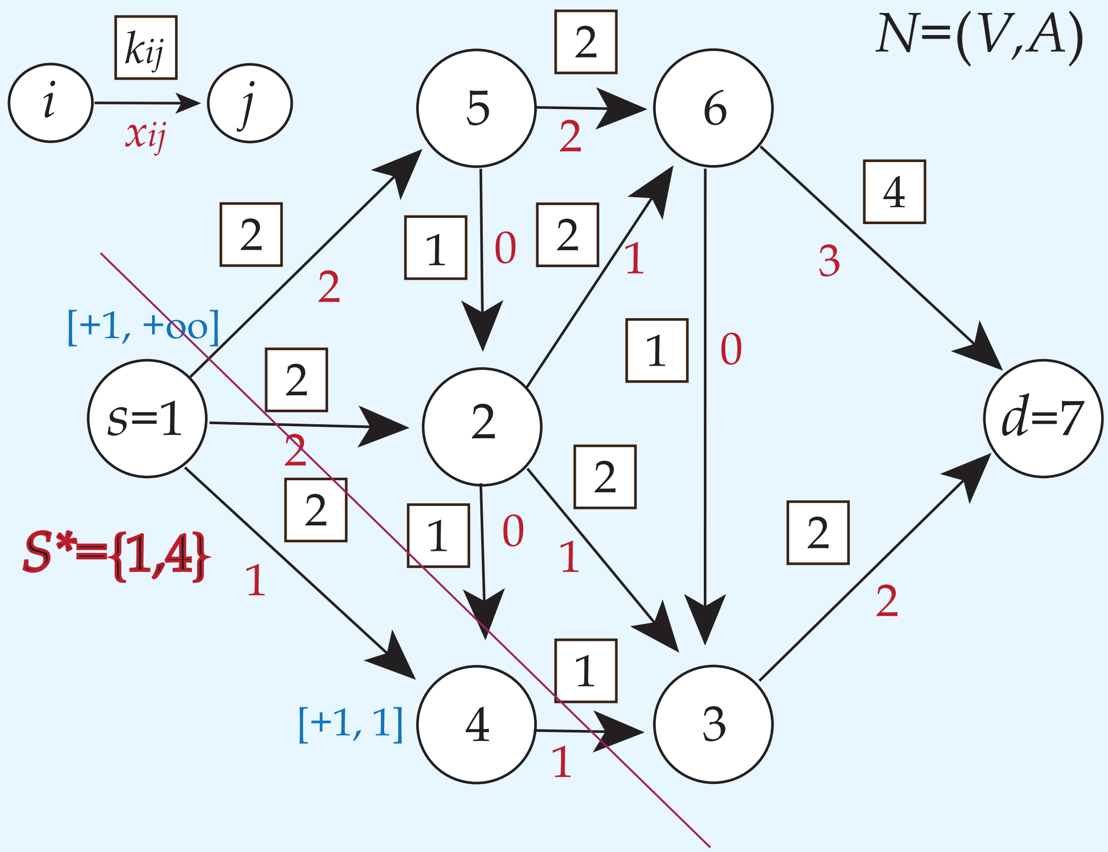


Iteration IV:

d can not be reached from s.

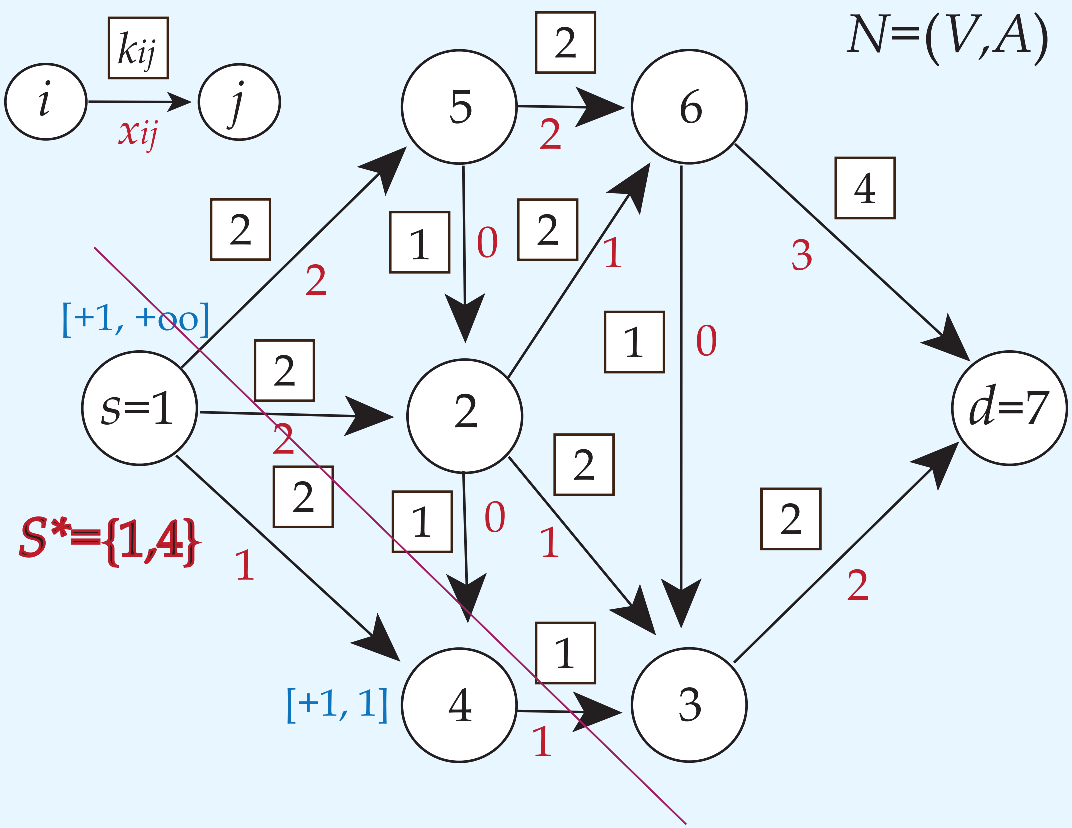
$(S^*, V \setminus S^*) = (\{1,4\}, \{2,3,5,6,7\})$ is the minimum cut.

$\varphi_0^* = 5.$



Exercise, ⑦

$(S^*, V \setminus S^*) = (\{1,4\}, \{2,3,5,6,7\}); \varphi_0^* = \varphi(S^*) = K(S^*) = 5.$



Notes:

- ✔ every arc $(i,j) \in \delta_N^+(S^*)$ is saturated;
- ✔ every arc $(i,j) \in \delta_N^-(S^*)$ is unloading.