Network Flows

UPCOPENCOURSEWARE number 34414

Topic 4: Max Flow Problems

F.-Javier Heredia

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Departament d'Estadística
i Investigació Operativa

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/3.0/ or send a letter to Creative Commons, 444 Castro Street, Suite 900, Mountain View, California, 94041, USA.

4.- Maximum flow algorithms (MF)
(Chap. 6 & 7 Ahuja,Magnanti,Orlin)

- Notation and hypothesis.
- Classification of the MF algorithms.
- Cuts and Flows.
- Pseudopolynomial MF augmenting paths algorithms.
  - Generic augmenting path algorithm.
  - Labeling algorithm (Ford-Fulkerson).
    - Convergence.
    - Complexity.
- Polynomial MF augmenting path algorithms.
  - Improvements of the Ford-Fulkerson algorithm
  - Shortest Augmenting Path Algorithm.
    - Convergence and complexity.
- Exercices.

BY-NC-ND

Notation and Assumptions.

- **MFP**: considering a directed network $G=(N,A)$ with a nonnegative capacity u, the maximum flow problem finds the maximum flow from the source node s to the sink node t.

$$\begin{cases} \max & z = v & \text{(5.1a)} \\ \text{s.a.:} & & \\ & \sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = \begin{cases} v & i = s \\ 0 & \forall i \in N - \{s,t\} \\ -v & i = t \end{cases} & \text{(5.1b)} \\ & 0 \le x_{ij} \le u_{ij} \quad \forall (i,j) \in A & \text{(5.1c)} \end{cases}$$

- **Assumptions:**
  - *i. The network is directed.*
  - *ii. The capacities u_{ij} are nonnegative integers.*
  - *iii. G does not contain any directed path s-t with infinite capacity arcs.*
  - *iv. If $(i,j)\in A$, then $(j,i)\in A$ (probably with $u_{ji}=0$).*
  - *v. G does not contain parallel arcs.*

BY-NC-ND

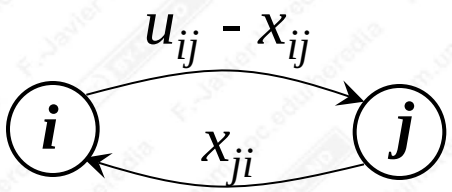
MFP algorithm classification

- **MFP algorithms:**
  - ***Augmenting-Path algorithms***: maintain mass balance constraints at every node of the network other that the source (s) and sink (t) nodes. This algorithms incrementally augment flow along paths from the source node to the sink node.
    - Algorithmic complexity: $O(n^2m)$ the *successive shortest path algorithm* is the most efficient augmenting-path algorithm.
    - With uncapacitated arcs, simple implementations may not converge to an optimal solution.
  - ***Pre-flow-Push algorithms***: flood the network so that some nodes have excess of flow. These algorithms incrementally relieve flow from nodes with excess by sending flow to the forward toward the sink node (t) or backward to the source node (s).
    - Algorithmic complexity: $O(n^2\sqrt{m})$ *Highest-label pre-flow-push* algorithm is most efficient maximum flow algorithm.

BY-NC-ND

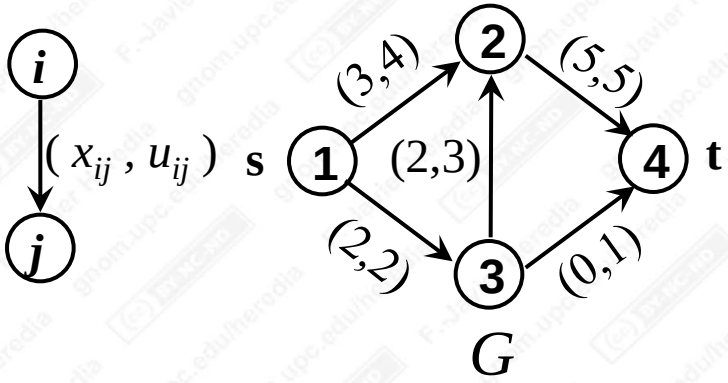
Flows and Cuts (1/3)

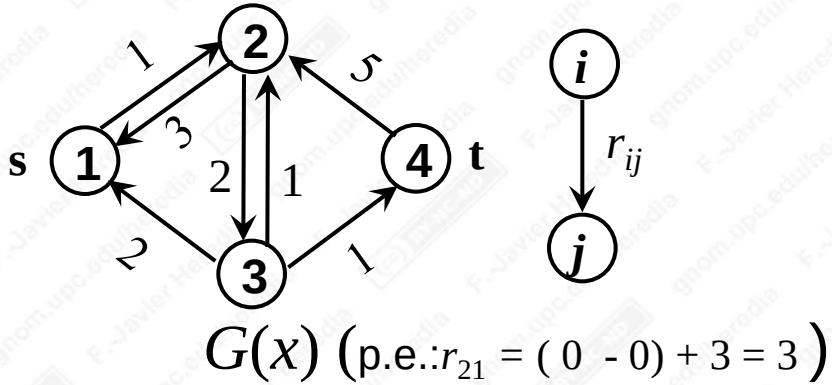
- **Residual network**:
  - **Residual capacity r_{ij} of arc $(i,j) \in A$, w.r.t the flux x_{ij}** : max flow that can be sent from node i to node j through arcs (i,j) and (j,i).



$$r_{ij} = (u_{ij} - x_{ij}) + x_{ji}$$

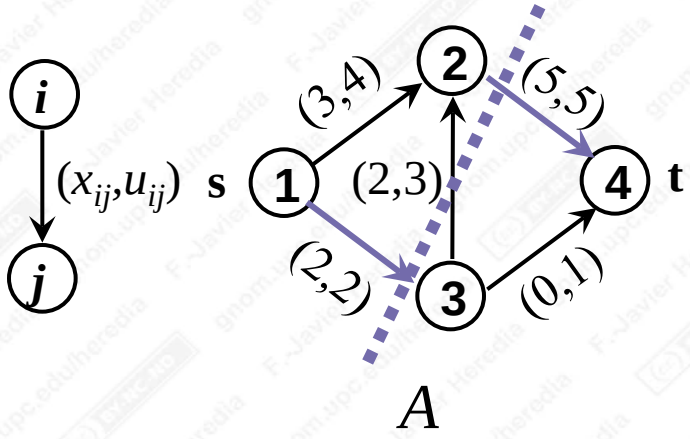
  - **Residual network $G(x)$** : formed by the arcs with positive residual capacity w.r.t. flow x:



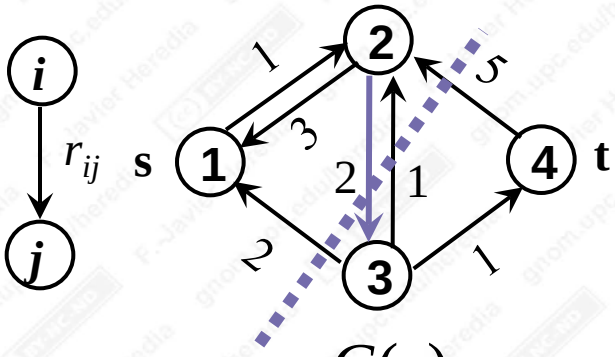


BY-NC-ND

Flows and Cuts (2/3)



A



$G(x)$

- ***s-t* cut capacity, $u[S,\underline{S}]$:** sum of capacities of the foward arcs $(i,j)\in(S,\underline{S})$ in the *s-t* cut.
  - $S=\{1,2\}$, $\underline{S}=\{3,4\}$, $(i,j)\in(S,\underline{S}) = \{(1,3),(2,4)\}$
  - $u[S,\underline{S}] = 5+2 = 7$
- **Minimum cut :** *s-t* cut with the minimum capacity among all cuts *s-t*:
  - Example, $\min\{ 6, 7, 8, 6\} = 6$
- **Residual capacity of the *s-t* cut, $r[S,\underline{S}]$:** sum of the **residual capacities** of the forward arcs in the cut $(S,\underline{S})$:
  - $S=\{1,2\}$, $\underline{S}=\{3,4\}$, $(i,j)\in(S,\underline{S}) = \{(2,3)\}$
  - $r[S,\underline{S}] = 2$

- **Flow across an *s-t* cut :**

$$v = \sum_{i\in S}\left[\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji}\right] = \sum_{(i,j)\in(S,\underline{S})} x_{ij} - \sum_{(i,j)\in(\underline{S},S)} x_{ij} \qquad (5.2)$$

BY-NC-ND

Flows and Cuts (3/3)

- **Property 4.1:**
  "The value of any flow x *is less than or equal to the capacity of any* s-t *cut in the network*"
  - **Proof:** substuting $x_{ij} \leq u_{ij}$ in the first expression of (5.2) and $x_{ij} \geq 0$ in the second expression follows that:

$$v \leq \sum_{(i,j)\in(S,\underline{S})} u_{ij} = u[S,\underline{S}]$$

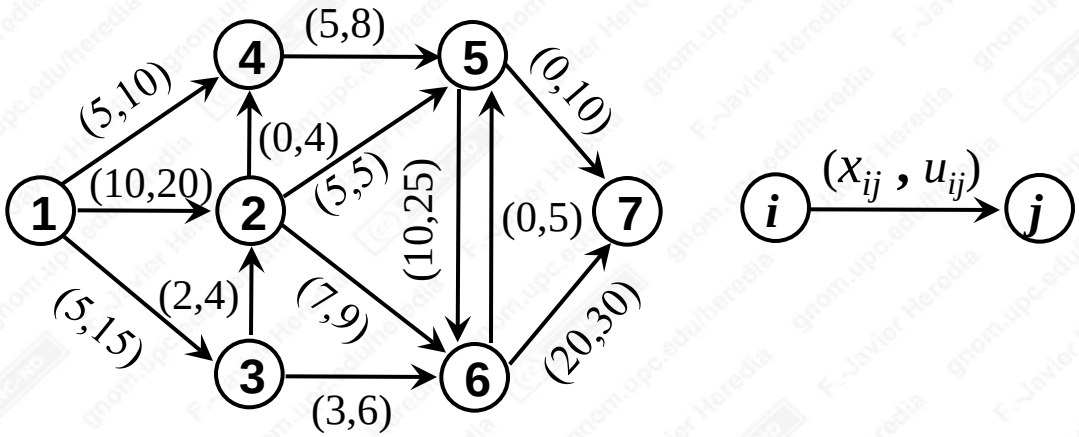
  - **Results:** if we find a flow x equal to the value of any cut s-t, this flow is the optimum. It has been demonstrated that always exist a flow of this type.

- **Property 4.2:**
  "*For any flow* x *of value* v *in the network* A*, the additional flow that can be sent from the source* s *to the sink node* t *is less than or equal to the residual capacity of any* s-t *cut.*"

BY-NC-ND

Example (prob. 6.12 A-M-O)

- Considering the following network:



Find:

- Four s-t cuts, each one with 4 forward arcs, indicating their capacity, residual capacity and the flow across the cut.
- Represent the residual graph $G(x)$ and indicating two augmenting paths s-t (i.e. any directed path s-t over $G(x)$).

BY-NC-ND

Generic Augmenting Path Algorithm (GAP).

GAP Algorithm : given a directed graph $G=(N,A)$ with **integer** capacities finds the maximum flow between nodes s and t.

begin

$x:=0$;

find $G(x)$;

while $G(x)$ contains a directed path from node $s \rightarrow t$ **do**

identify an augmenting path P $s \rightarrow t$;

set δ := min{ r_{ij} : $(i,j) \in P$ };

augment δ units of flow along P;

update $G(x)$;

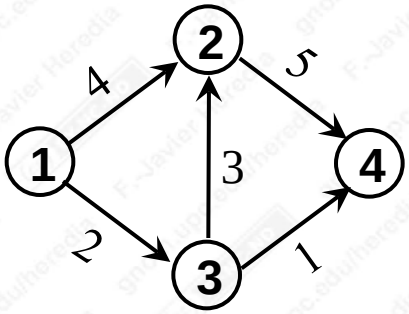
end;

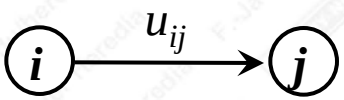
end

BY-NC-ND

Generic Augmenting Path : example.

- Solve the following MFP with the GAP algorithm:

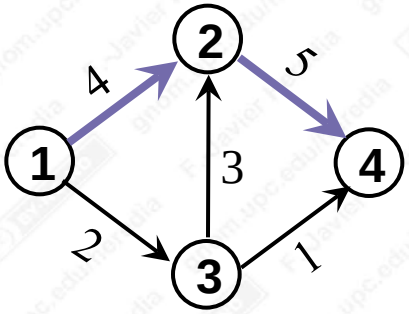


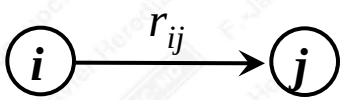


BY-NC-ND

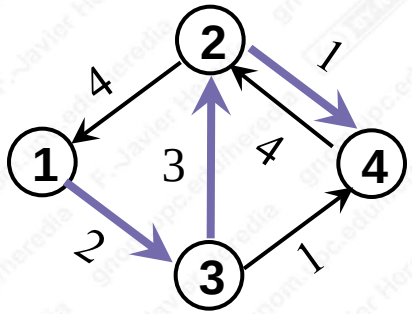
GAP Algorithm solution example

- **Begin**: $x^0 := 0$
- $G(x^0)$; $P = \{(1,2),(2,4)\}$; $\delta = 4$

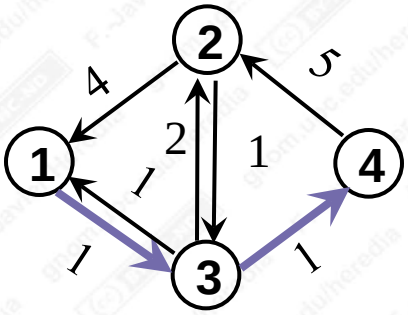




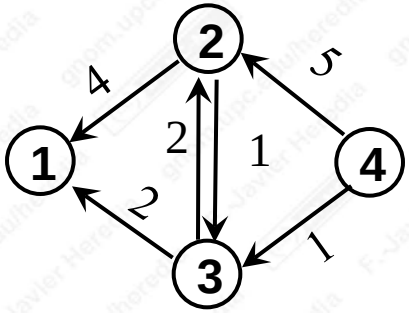
- $G(x^1)$; $P = \{(1,3),(3,2),(2,4)\}$; $\delta = 1$



- $G(x^2)$; $P = \{(1,3),(3,4)\}$; $\delta = 1$



- $G(x^3)$; $P = \varnothing$; STOP



BY-NC-ND

Ford-Fulkerson Algorithm: description.

Ford-Fulkerson Algorithm : let the directed path $G=(N,A)$ with integer capacities, find the maximum flow between s and t.

begin

node t *is* labeled;

if t is labeled **do**

unlabel all nodes;

set $pred(j) := 0$ for each $j \in N$;

label node s and set LIST:={ s };

while LIST $\neq \varnothing$ **and** t is unlabeled **do**

remove a node i from LIST;

for each arc (i, j) in the residual network emanating from node i **do**

if node j is unlabeled **then**

set $pred(j) := i$; label node j and add j to LIST;

end;

if t is labeled **then** **`augment(r,pred)`**;

end;

end if;

end

Procedure `augment(r,pred);`

begin

find an augmenting path P from s to t using $pred(\,)$;

set $\delta := \min\{ r_{ij} : (i,j) \in P\}$;

augment δ units of flow along P;

update the residual capacities;

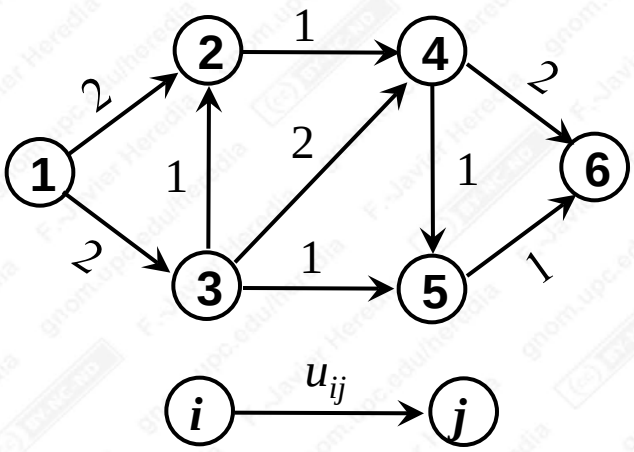
end;

BY-NC-ND

Ford-Fulkerson algorithm : example.

- See the Applet for main idea of the algorithm:
  - http://www-b2.is.tokushima-u.ac.jp/~ikeda/suuri/maxflow/MaxflowApp.shtml?demo1

- Solve now the following MFP with the Ford-Fulkerson algorithm:



BY-NC-ND

Ford-Fulkerson Algorithm: convergence (I).

- **Convergence of the algorithm**:
  Theorem 4.1: "*At termination of the Ford-Fulkerson algorithm, the flows* x *are the solution of the MFP.*"
- **Proof:**
  - Let S be the set of labeled nodes at the last iteration. Thus: $s \in S$ and $t \in \underline{S} = N - S$
  - a) $r_{ij} = 0 \ \forall (i,j) \in (S,\underline{S})$ (there is not any augmenting paths between S and $\underline{S}$.)
  - b) $r_{ij} = (u_{ij} - x_{ij}) + x_{ji} \quad ; \quad x_{ij} \le u_{ij} \quad ; \quad x_{ji} \ge 0$
  - a) and b) $\Rightarrow x_{ij} = u_{ij} \ \forall (i,j) \in (S,\underline{S})$; $x_{ij} = 0 \ \forall (i,j) \in (\underline{S},S)$. Substituting in (4.2) we get:

$$v = \sum_{(i,j) \in (S,\underline{S})} x_{ij} - \sum_{(i,j) \in (\underline{S},S)} x_{ij} = \sum_{(i,j) \in (S,\underline{S})} u_{ij} = u\left[S,\underline{S}\right]$$

  - Therefore, according to **property (4.1)**, the flows x are the maximum flows and $[S,\underline{S}]$ cut is a minimum cut.

BY-NC-ND

Ford-Fulkerson Algorithm: convergence (II).

- **Max-Flow Min-Cut Theorem**:
  "The maximum flow between the source node s *and the sink node* t *in a capacited network is equal to the minimum capacity among all* s-t *cuts.*"
- **Augmenting Path Theorem:**
  "A flow x^* *is a maximun flow if and only if the residual network* $G(x^*)$ *contains no augmenting path.*"
- **Integrality Theorem:**
  "*If all arc capacities are integer, the maximum flow problem has an integer maximum flow.*"

BY-NC-ND

Ford-Fulkerson Algorithm: complexity.

- ***Worst-case complexity***: we compute an upper bound of the number of iterations of Ford-Fulkerson:
  - **Calculation of the augmenting path:** each augmentation visits each arc at most once, and therefore the cost is $O(m)$.
  - **Number of augmenting paths:**
    - If all capacities are integer and bounded by U, then the capacity of the cut $(s,N\text{-}\{s\})$ is $\leq nU$, and the maximum flow is $\leq nU$.
    - The algorithm increase the flow at least one unit in any augmentation, i.e., then the number of augmentations is $\leq nU$.
- **Theorem 4.2:**

  "*The labeling algorithm solves the MFP in $O(nmU)$ time*".

BY-NC-ND

Improved Generic Path Algorithms

- Drawbacks of the Ford-Fulkerson labeling algorithm:
  1. Its pseudopolynomial complexity $O(nmU)$ is unattractive for problems with large capacities.
  2. If the capacities are irrational, the algorithm migth not converge.
- Improvements:
  - **Largest augmenting path algorithm**: let P be a path from s to t in $G(x)$ such that δ^* is maximum ($O(m \cdot logU)$)
  - **Shortest augmenting path algorithm**: let P be a path from s to t in $G(x)$ with the fewest number of arcs ($O(m \cdot n^2)$).

BY-NC-ND

Shortest augmenting path algorithm (1)

Distance Labels

- ***A distance label*** is a function $d: N \rightarrow Z^+$. A distance label is said to be ***valid*** if it satisfies the following:
  - $d(t) = 0$.
  - $d(i) \leq d(j) + 1$ for each $(i,j) \in G(x)$.
- An arc $(i,j) \in G(x)$ is ***admissible*** if $d(i) = d(j) + 1$.

(1) Adapted from: Orlin, James. *15.082J Network Optimization,Fall 2010*. (Massachusetts Institute of Technology: MIT OpenCourseWare), http://ocw.mit.edu (Accessed 28 Mar, 2013). License: Creative Commons BY-NC-SA

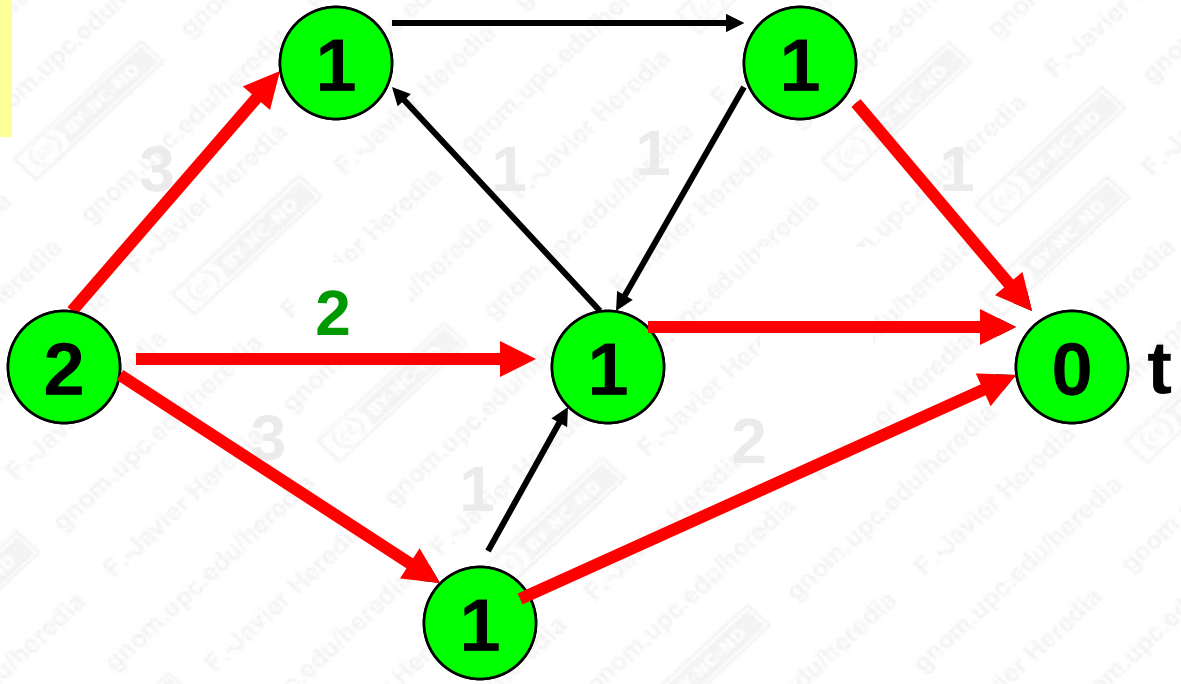
BY-NC-ND

An example of valid distance labels

The distance labels are on the nodes.

All arcs are in the residual network.

The admissible arcs are thick and red.



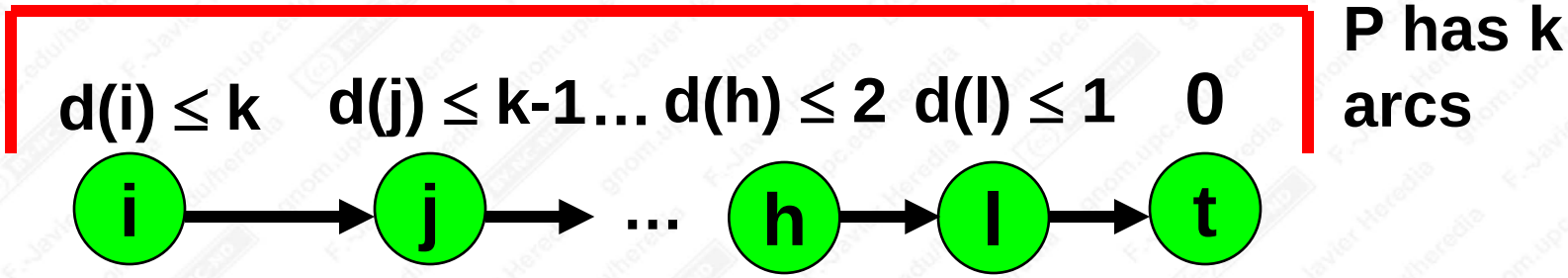
The labels would not be valid if there were an arc from “2” to “0”.

BY-NC-ND

More on valid distance labels

- ***Property 4.3:*** *"Let d( ) be a valid distance label. Then d(i) is a lower bound on the shortest path from i to t in the residual network. (The distance is measured in terms of the number of arcs.)"*

Proof. Let P be the shortest path from I to t. If P contains k arcs then:



BY-NC-ND

On Finding Paths shortest s-t paths

- ***Property 4.3:*** *"Let d( ) be a valid distance label. Then d(i) is a lower bound on the shortest path from i to t in the residual network. (The distance is measured in terms of the number of arcs.)"*
- ***Property 4.4:*** *"If d(s)≥n, the residual network contains no directed path from the source node to the sink node".*
- ***Property 4.5:*** *"An admissible path is a shortest augmenting path from the source to the sink".*

BY-NC-ND

The shortest augmenting path algorithm

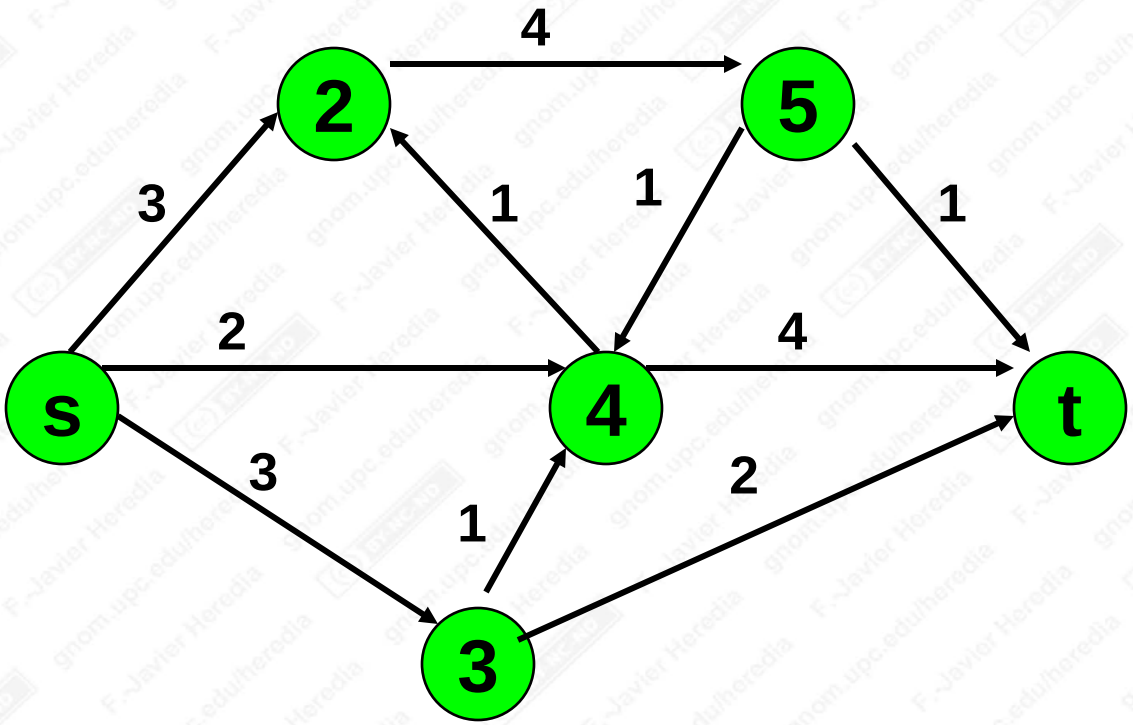
```
begin
    x:=0 ; Obtain exact distance labels d(i); i:= s;
    while d(s) < n do
    begin
        If i has an admissible arc then
            advance(i);
            if i = t  then augment  and set i = s ;
        else
            retreat(i);
        endif
    end
end
```

```
Procedure advance(i)
begin
        Let ( i, j) be an admissible arc in A( i); pred( j):= i and i:= j;
end
```

```
Procedure retreat(i)
begin
        d(i) := 1 + min { d(j) : r_ij > 0};
        if i ≠ s then i:= pred(i);
end
```


BY-NC-ND

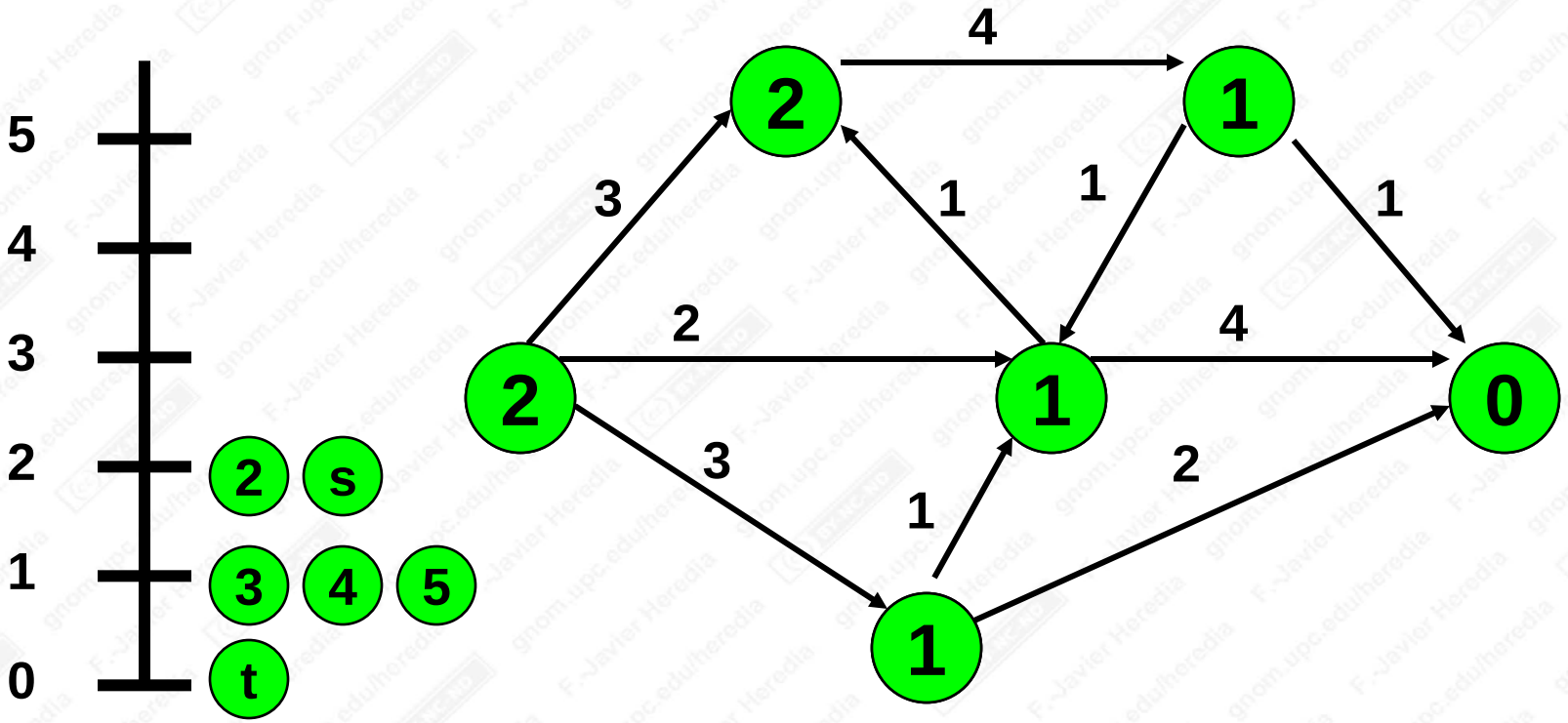
Shortest Augmenting Path



This is the original network, and the original residual network.

BY-NC-ND

Initialize Distances

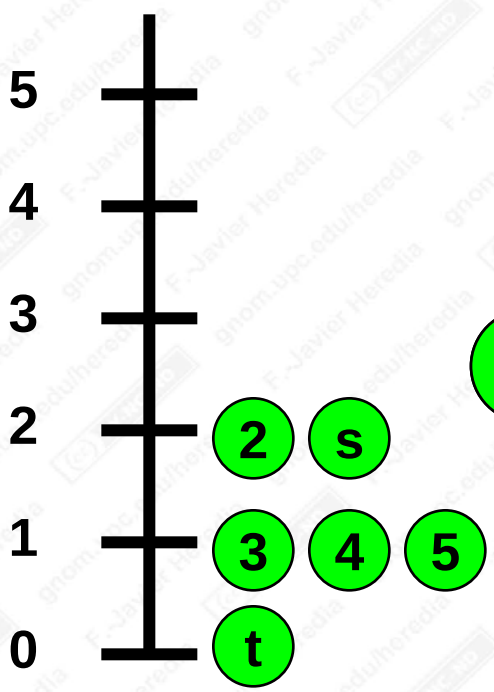


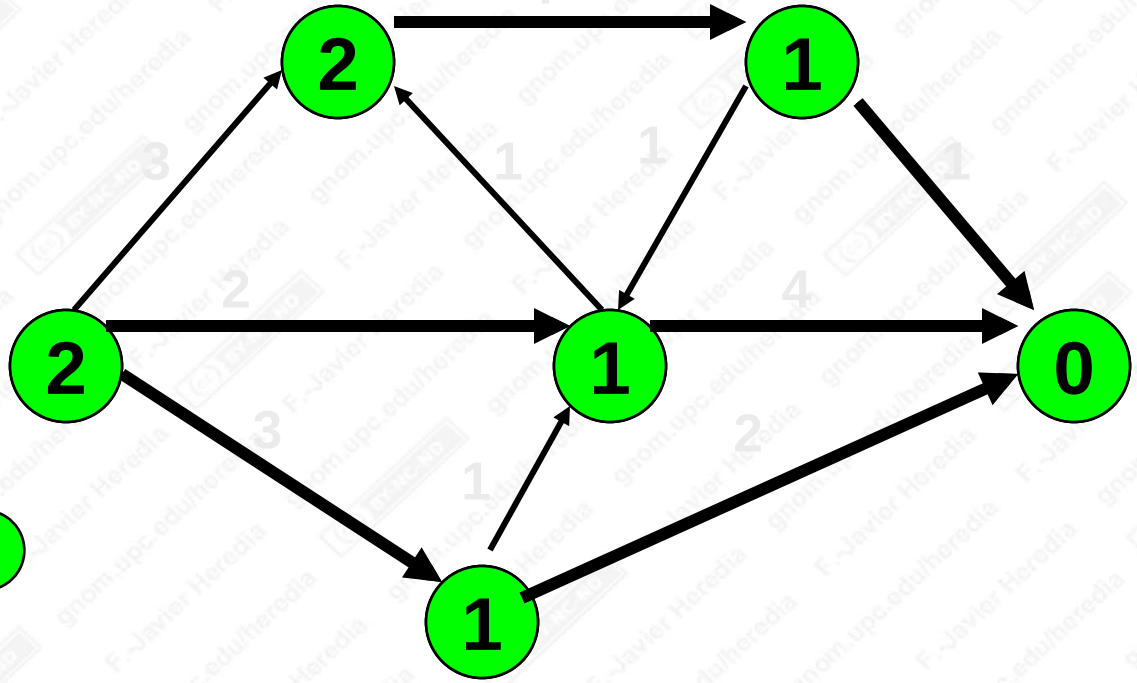
The node label henceforth will be the distance label.

d(j) is at most the distance of j to t in G(x)

BY-NC-ND

Representation of admissible arcs





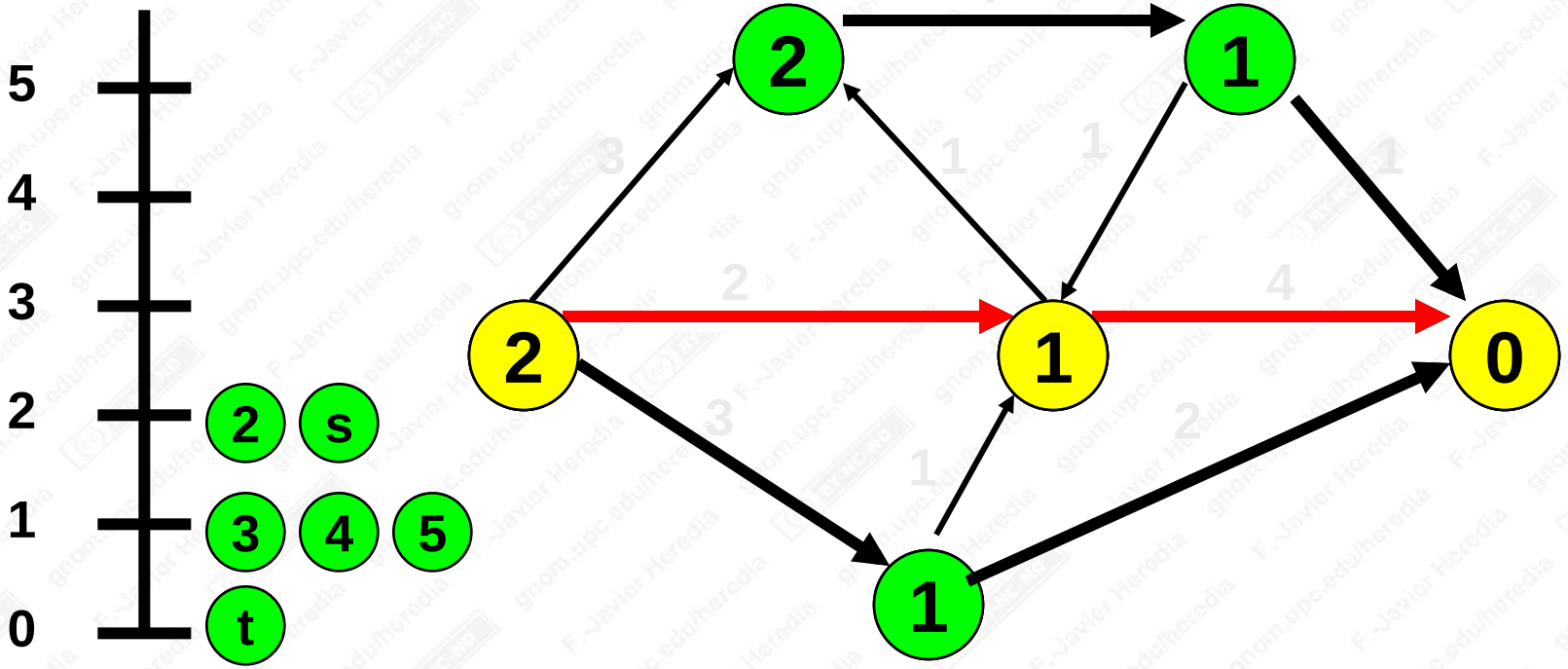
An arc (i,j) is *admissible* if d(i) = d(j) + 1.

An s-t path of admissible arcs is a shortest path

Admissible arcs will be represented with thick lines

BY-NC-ND

Look for a shortest s-t path

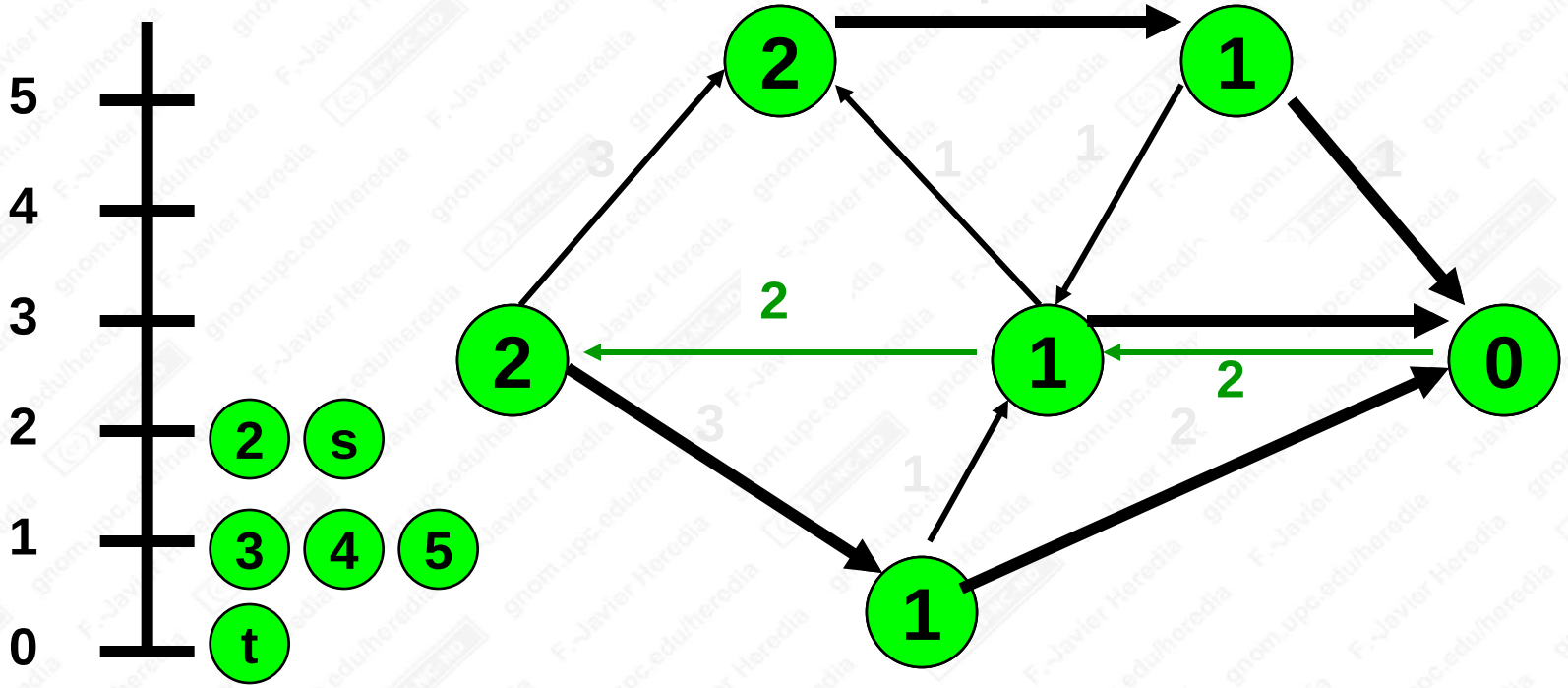


Start with s and do a depth first search using admissible arcs.

Next. Send flow, and update the residual capacities.

BY-NC-ND

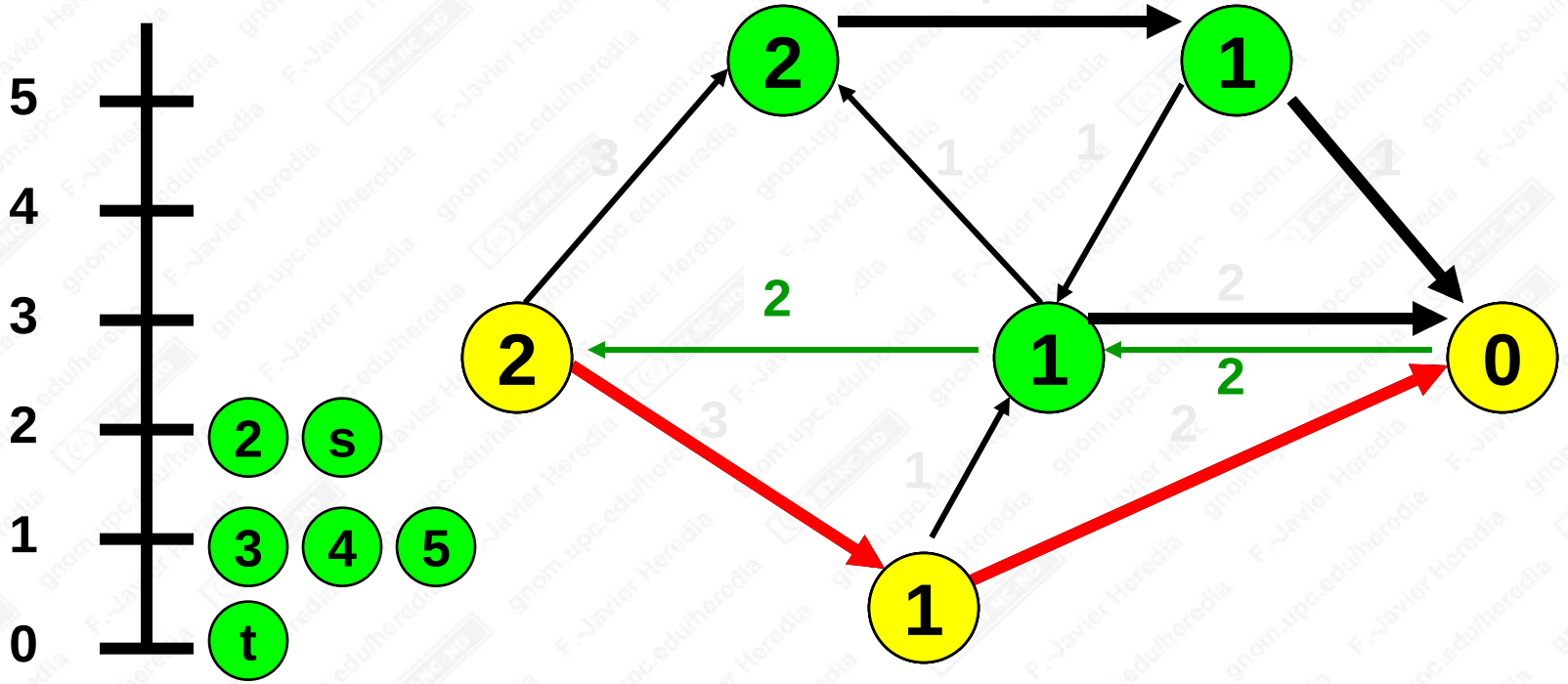
Update residual capacities



Here are the updated residual capacities.

We will update distance labels later, as needed.

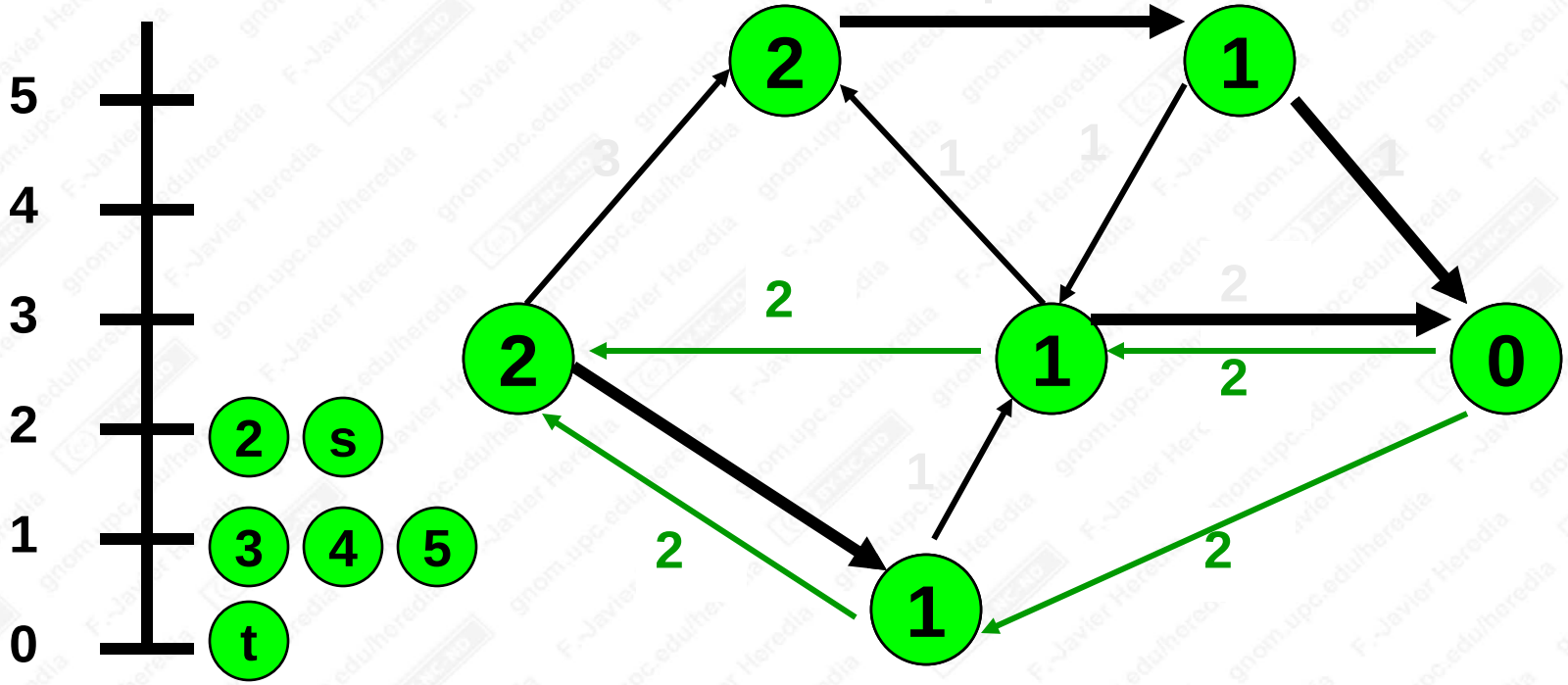
Look for a shortest s-t path



Start with s and do a depth first search using admissible arcs.

Next. Send flow, and update the residual capacities.

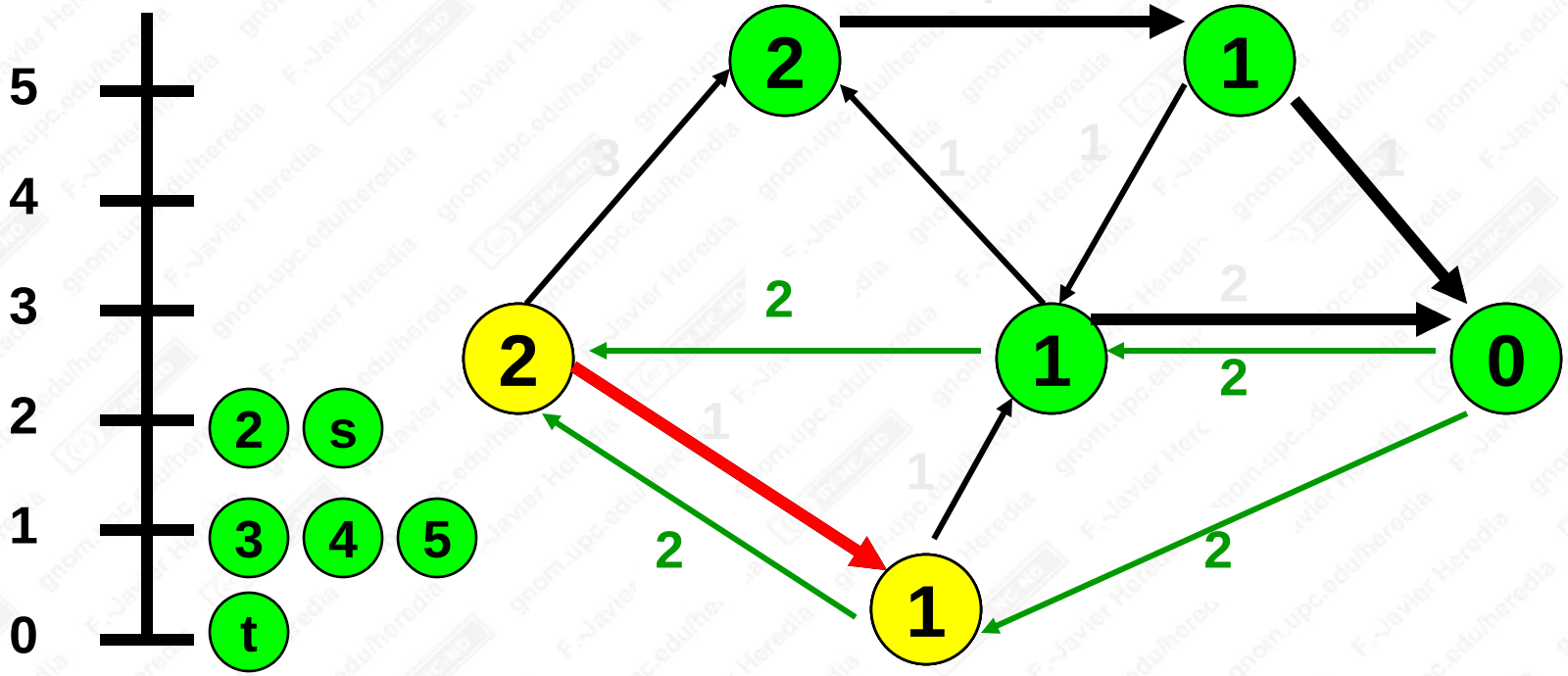
Update residual capacities



Here are the updated residual capacities.

We will update distance labels later, as needed.

Search for a shortest s-t path

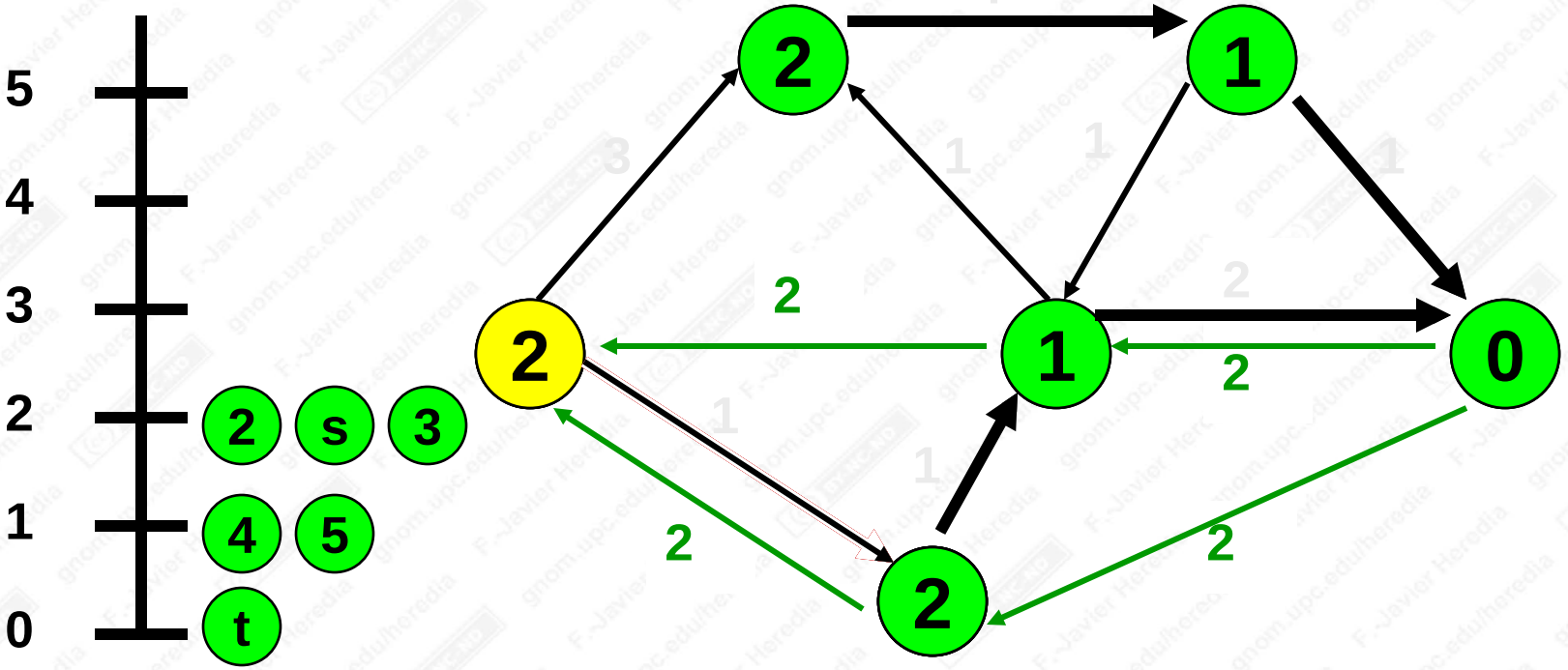


Start with s and do a depth first search using admissible arcs.

If there are no admissible arcs from i, then relabel(i) and reverse along the path leading to i.

BY-NC-ND

Update distances and path

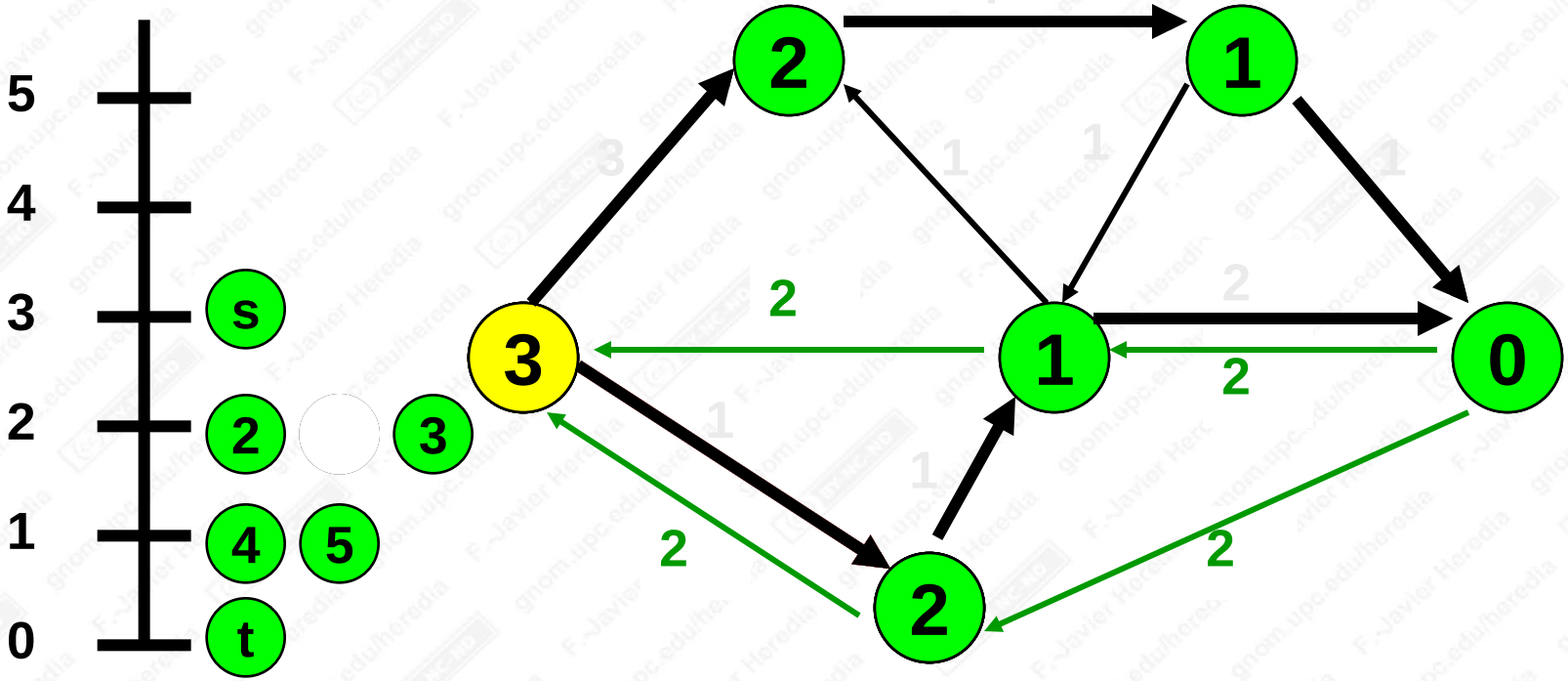


Start with s and do a depth first search using admissible arcs.

If there are no admissible arcs from i, then relabel(i) and reverse one arc along the path leading from s .

BY-NC-ND

Update distances and path

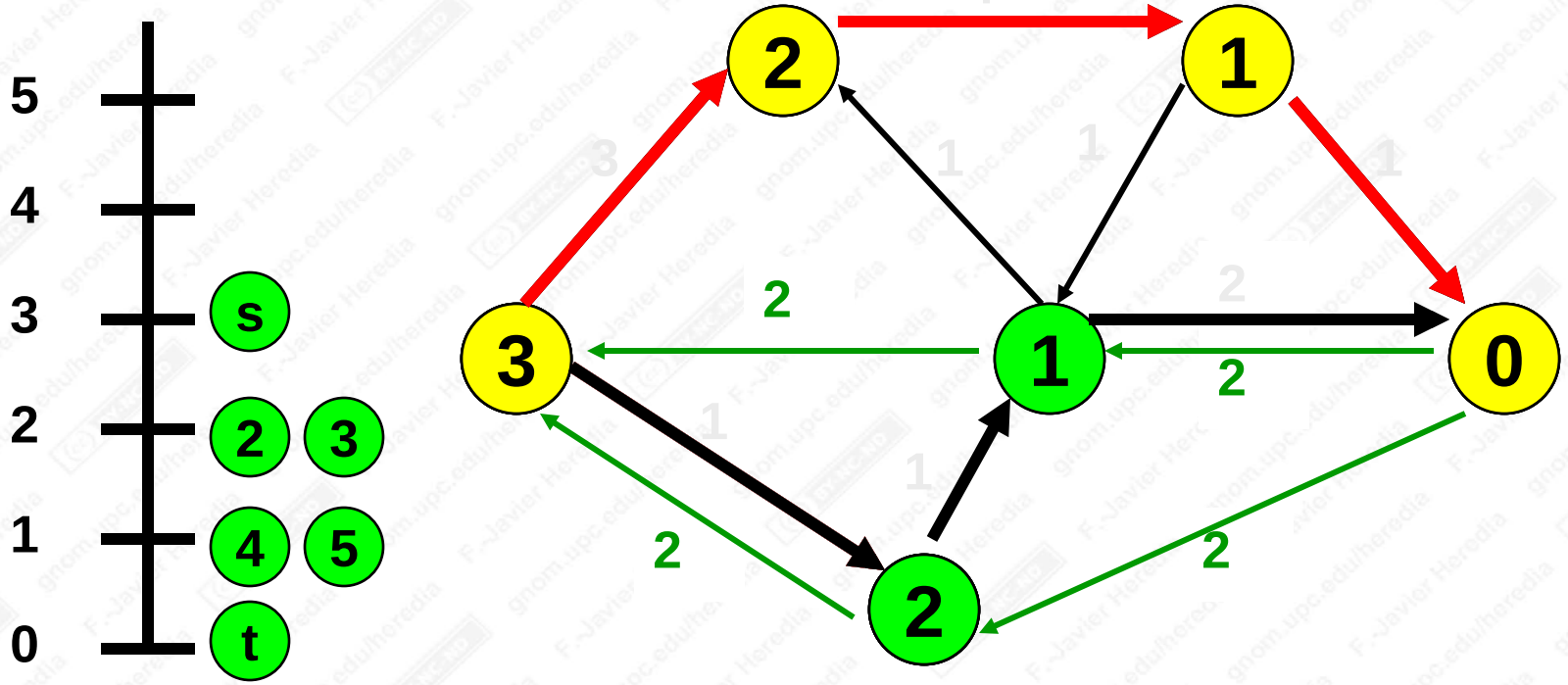


Start with s and do a depth first search using admissible arcs.

If there are no admissible arcs from i, then relabel(i) and reverse one arc along the path leading from s.

BY-NC-ND

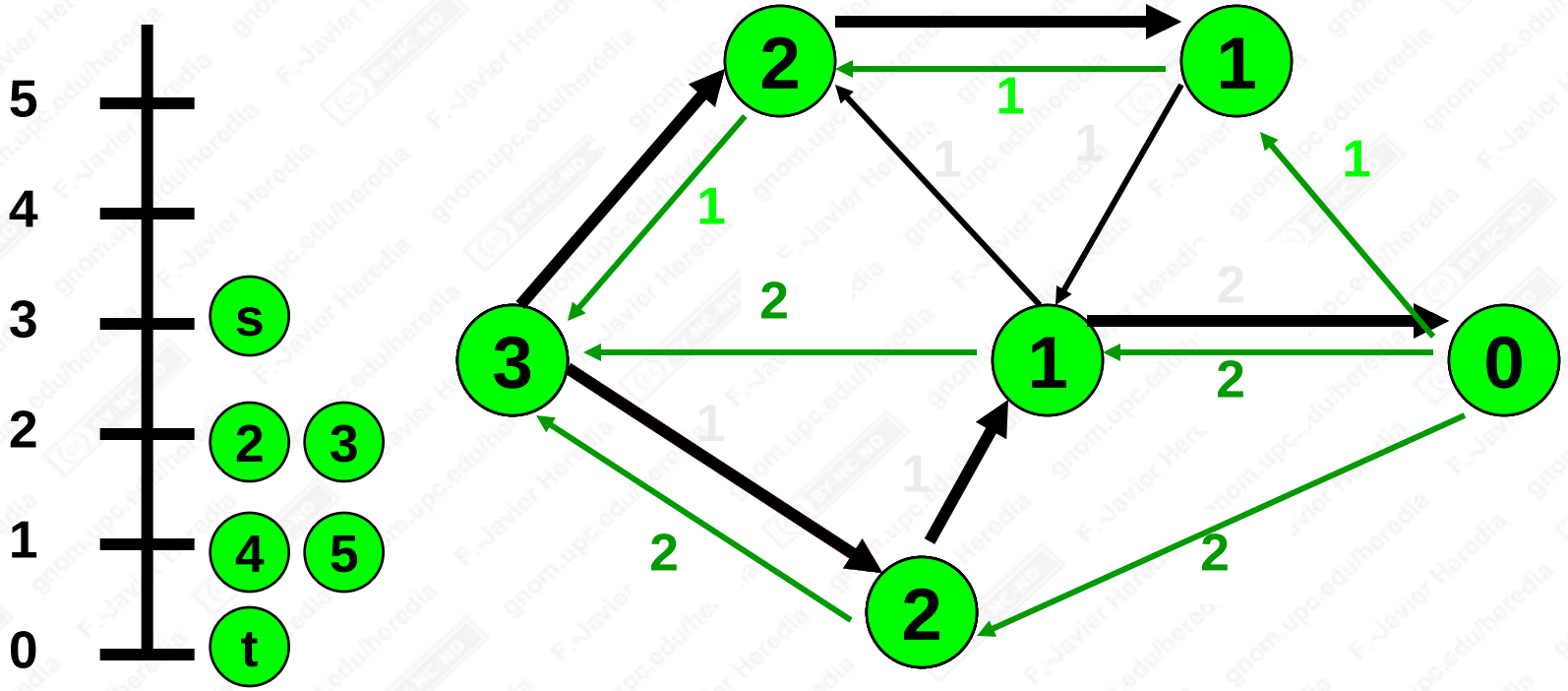
Look for a shortest s-t path



Continue the path from where it left off.

If the path reaches t, then send flow and update residual capacities.

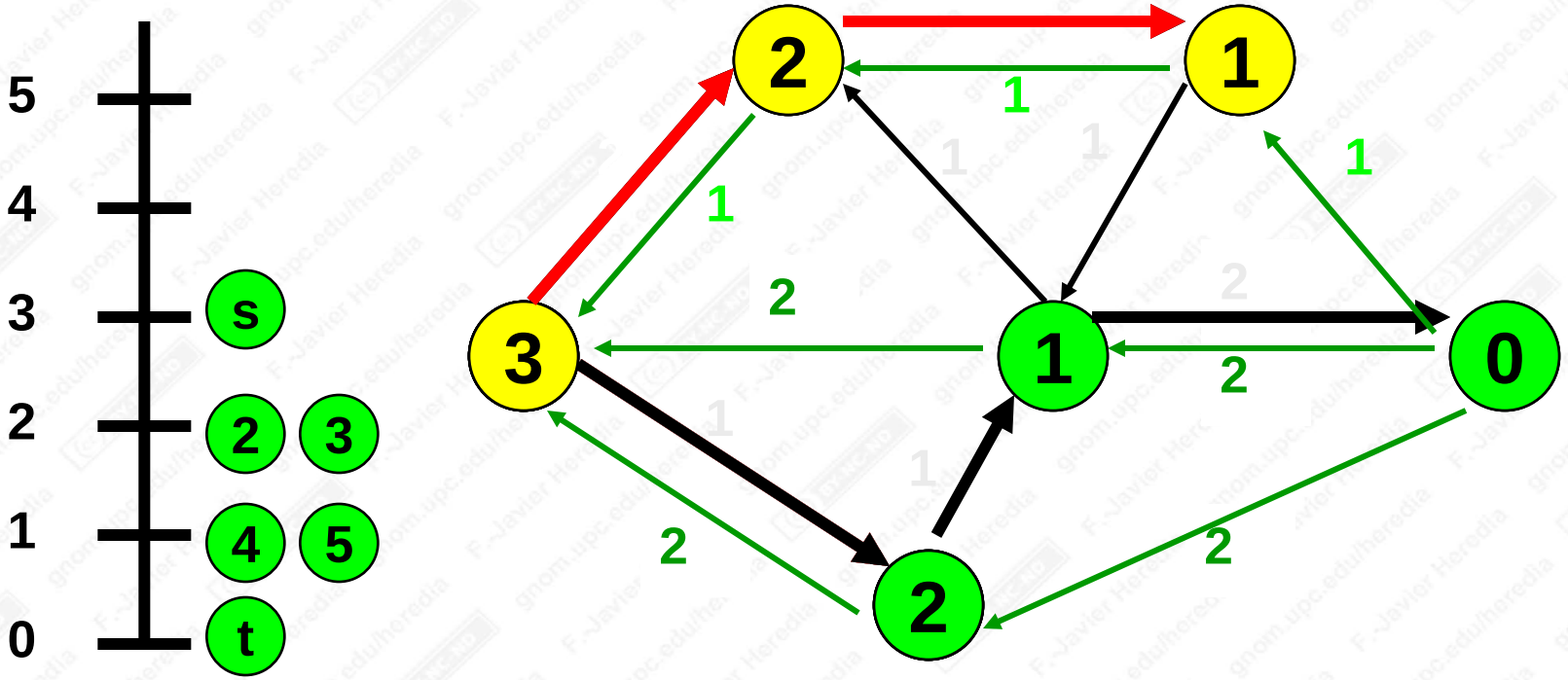
Update residual capacities



Here are the updated residual capacities.

BY-NC-ND

Search for a shortest s-t path

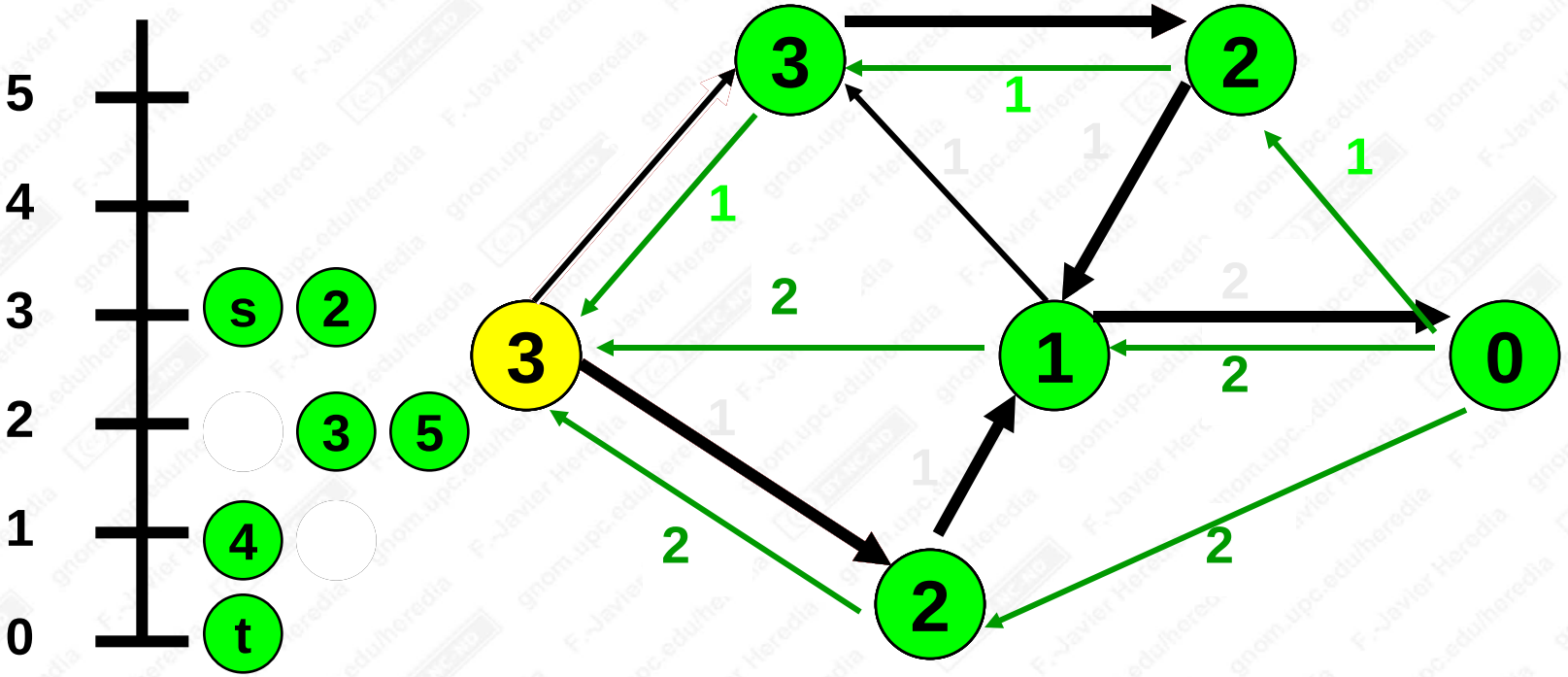


Search for a shortest s-t path starting from s

If there are no admissible arcs from i, then relabel(i) and reverse one arc along the path leading from s.

BY-NC-ND

Search for a shortest s-t path

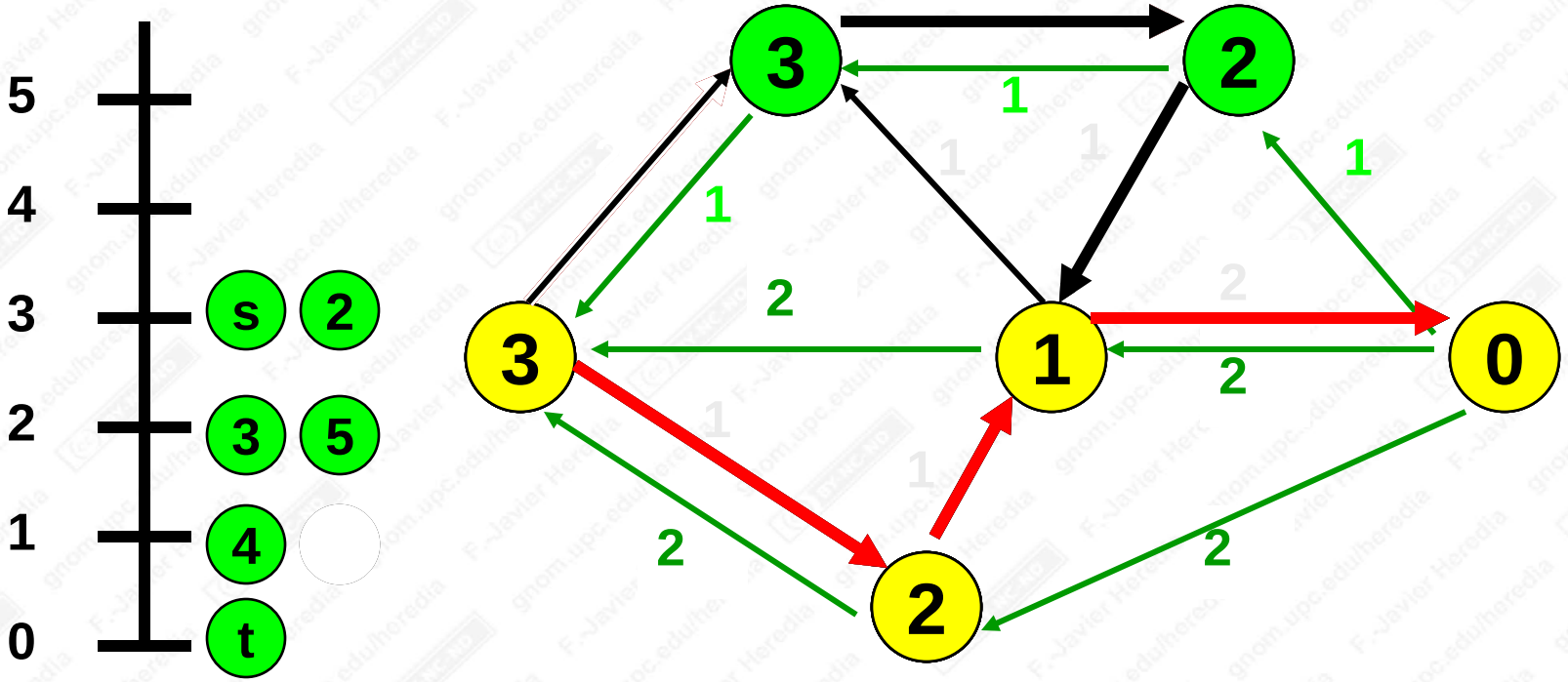


Search for a shortest s-t path starting from s

If there are no admissible arcs from i, then relabel(i) and reverse one arc along the path leading from s.

BY-NC-ND

Search for a shortest s-t path

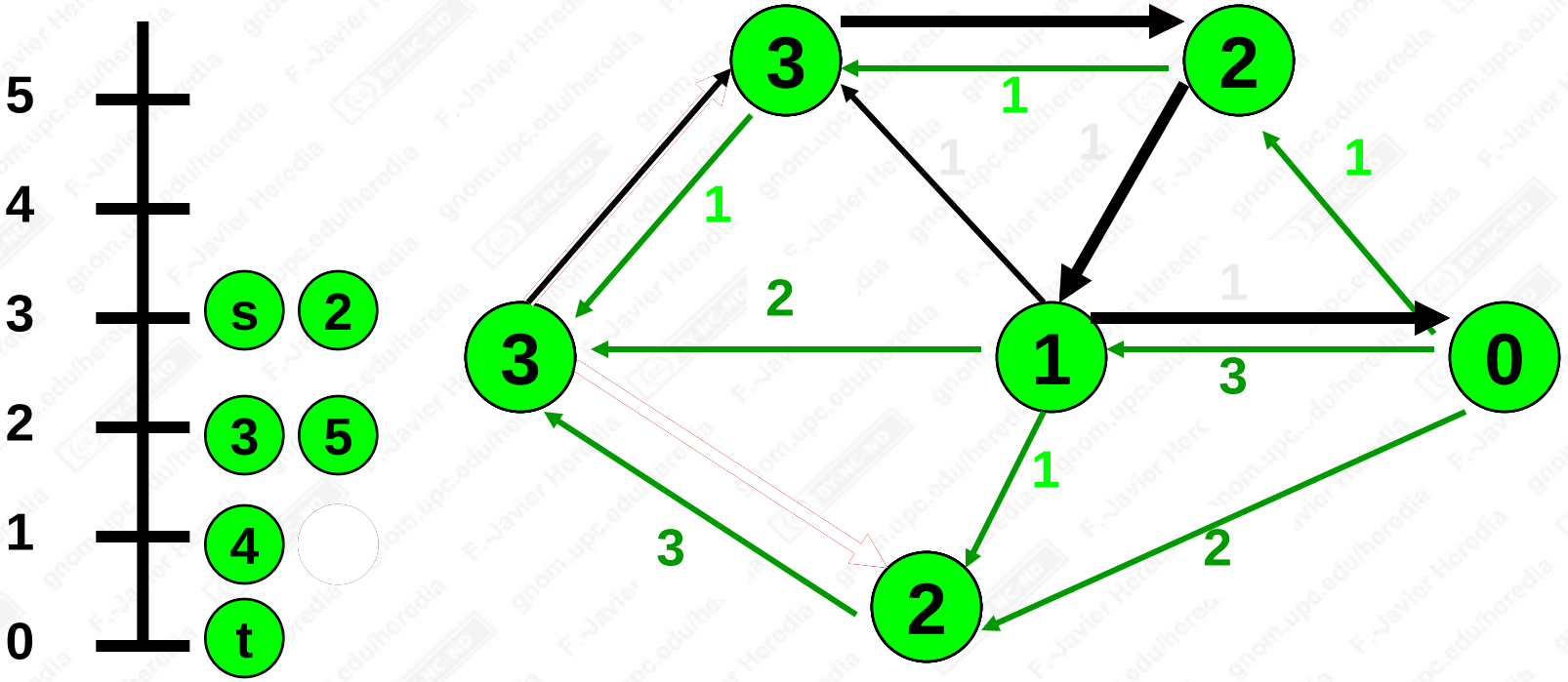


Search for a shortest s-t path starting from s

If the path reaches t, then send flow and update residual capacities.

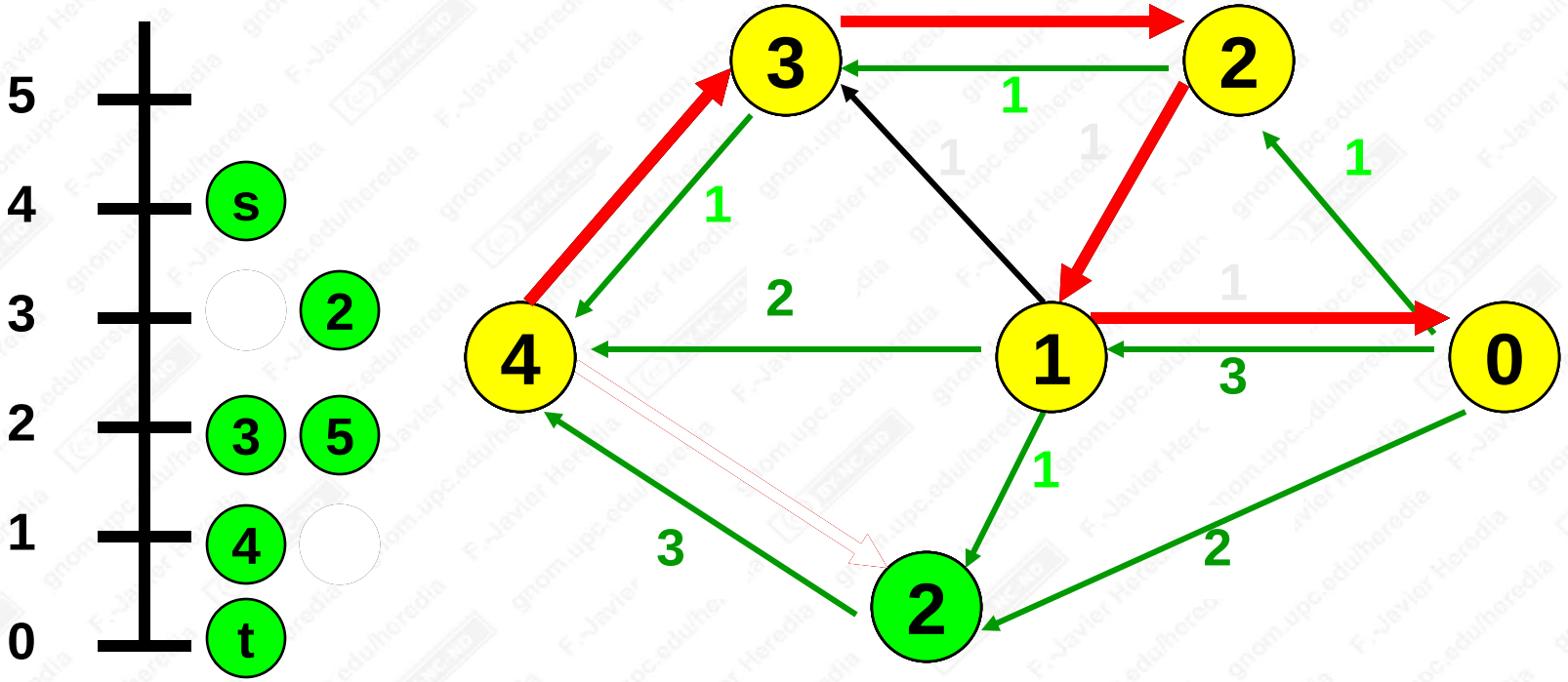
BY-NC-ND

update the residual capacities



Here are the updated residual capacities

Search for a shortest s-t path

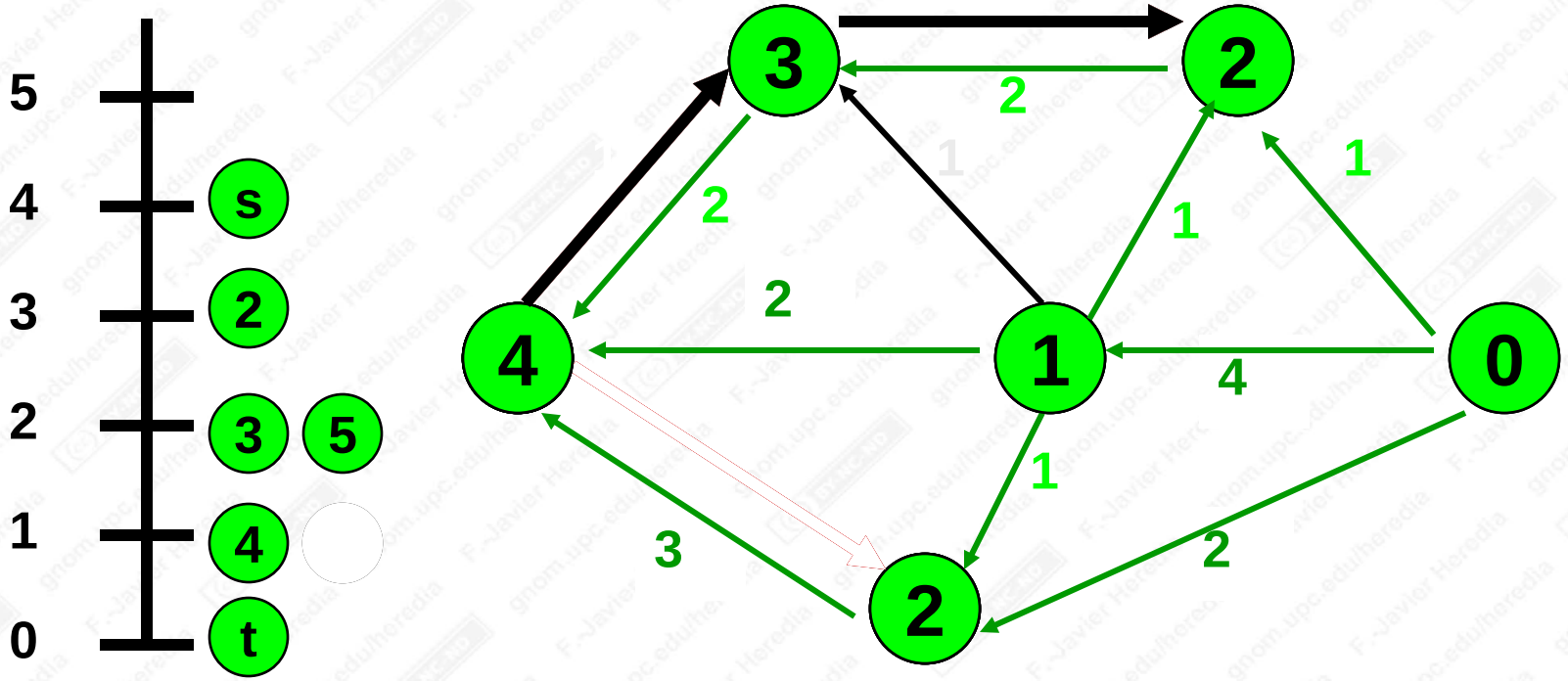


Search for a shortest s-t path

Next: update the residual capacities

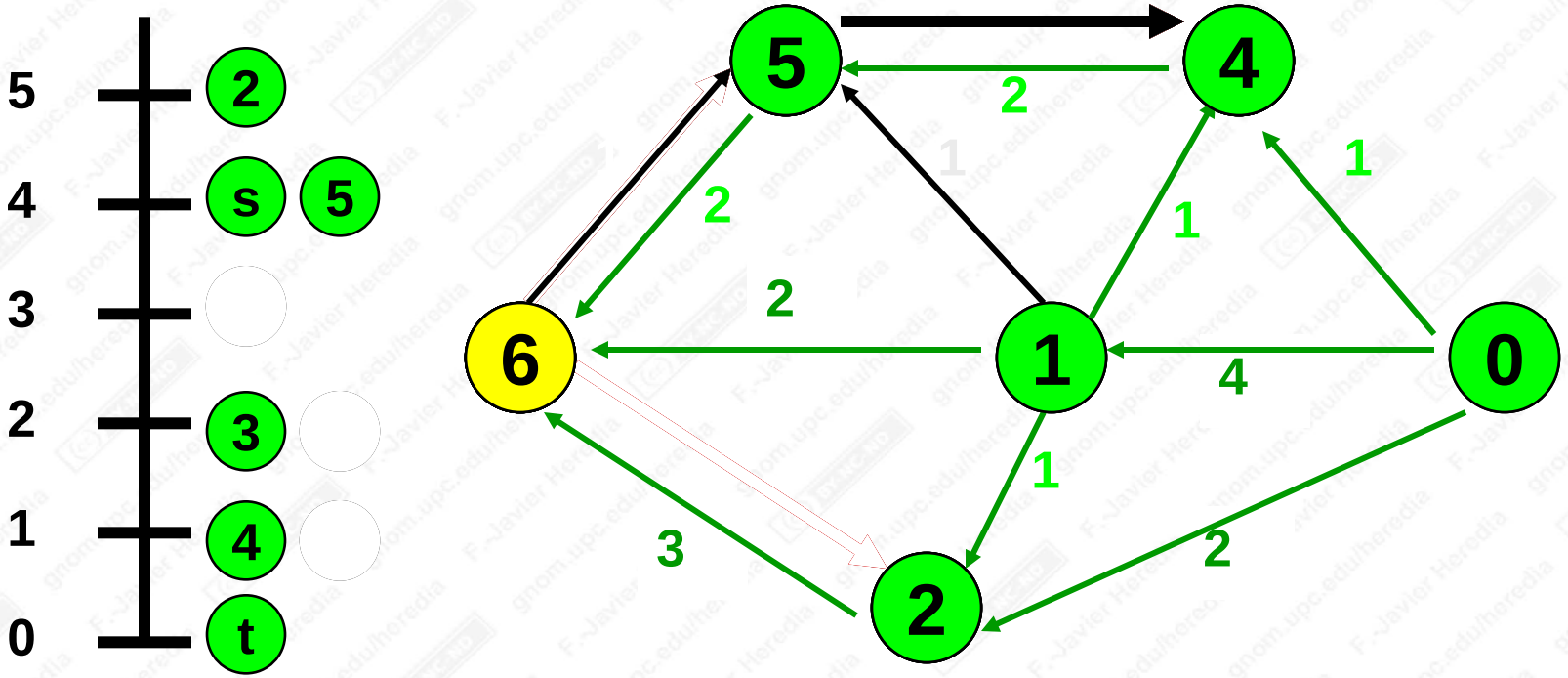
BY-NC-ND

Update the residual capacities



Here are the updated residual capacities

Look for a shortest s-t path

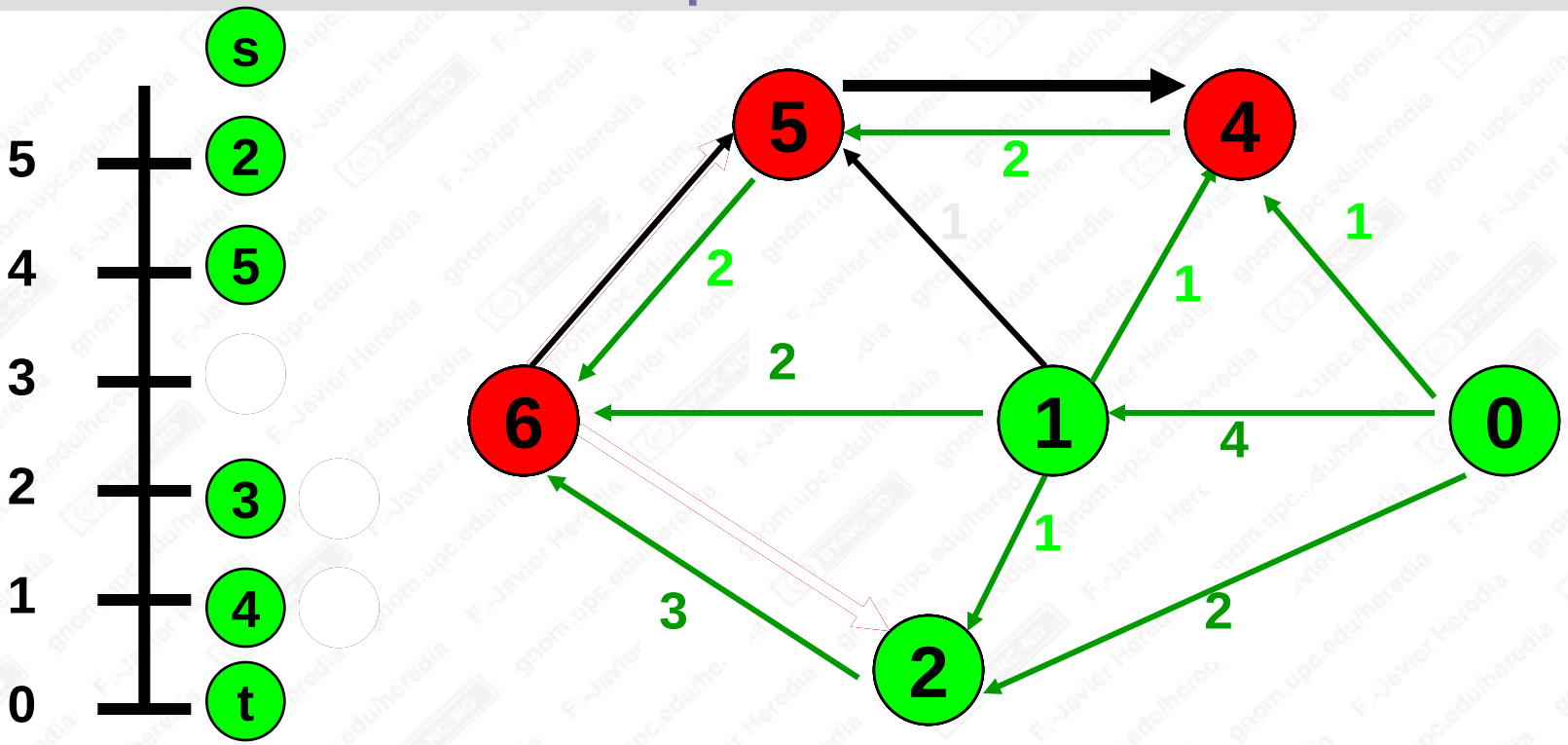


update distance labels and path

If d(s) > n-1, then there is no path from s to t

BY-NC-ND

These are the residual capacities for the optimum flow



There is no s-t path in the residual network

A min cut has S = {s, 2, 5}.

BY-NC-ND

Shortest Augmenting Path Algorithm
Convergence and Complexity (AMO-§7.4)

- **Convergence**:
  - **Lemma 4.1:** *"The SAPA maintains valid distance labels at each step. Moreover, each relabel operation strictly increases the distance label of a node"*
  - **Theorem 4.3:** "*The SAPA correctly computes a maximum flow*"

    Rationale: The SAPA terminates when d(s)>n+1 $\Rightarrow$ the network contains no augmenting path s$\rightarrow$t (Prop. 7.2). Consequently, the flow x is a max. flow.

- **Complexity** :
  - We can determine each augmentation in O(n).
  - The total time to maintain and update all distance labels is O(nm).
  - The total number of augmentations is O(nm).
  - **Conclusion**. The total running time is **$O(n^2m)$**.

BY-NC-ND

Ford-Fulkerson Algorithm : exercises (II).

- The following exercises will be assigned to:
  - 6.1(), 6.2 (), 6.3 (), 6.5 (), 6.16 (), 6.32 ().
- Each one represent an application of the MFP. Tasks to complete:
  - MFP formulation.
  - If the statement does not define the MFP, formulate the problem with a topology of the network and its respective costs.
  - Solve numerically with any software the MFP formulated.

BY-NC-ND