Lecture 27.
Maximum Flow / P or NP

Introduction to Algorithms

Sungkyunkwan University

Hyunseung Choo

choo@skku.edu

Max Flow Problem

- $\boldsymbol{G} = G(V, E)$
- $\boldsymbol{x}_{ij}$ = flow on arc (i, j)
- $\boldsymbol{u}_{ij}$ = capacity of flow in arc (i, j)
- $\boldsymbol{s}$ = source node
- $\boldsymbol{t}$ = sink node
- **Maximize** $\boldsymbol{v}$

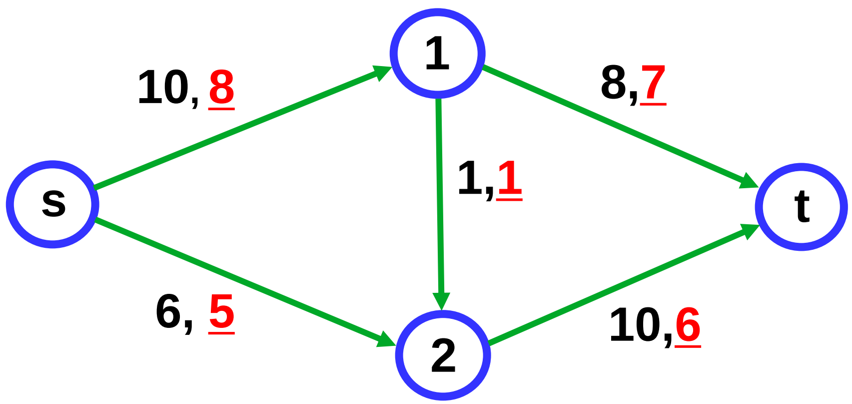
Subject to: $\sum_j \boldsymbol{x}_{ij} - \sum_k \boldsymbol{x}_{ki} = \mathbf{0}$ **for each** $\boldsymbol{i} \neq \boldsymbol{s}, \boldsymbol{t}$

$\sum_j \boldsymbol{x}_{sj} = \boldsymbol{v}$

$\mathbf{0} \leq \boldsymbol{x}_{ij} \leq \boldsymbol{u}_{ij}$ **for all** $(\boldsymbol{i}, \boldsymbol{j}) \in \boldsymbol{E}$

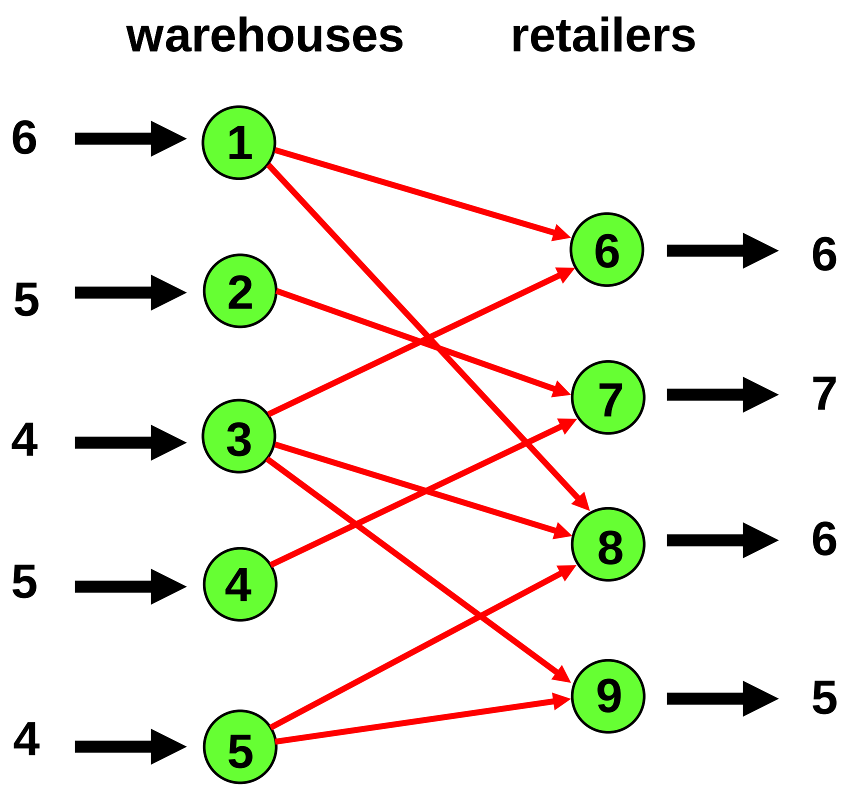
Maximum Flows

- We refer to a flow x as *maximum* if it is feasible and maximizes v
- Our objective in the max flow problem is to find a maximum flow



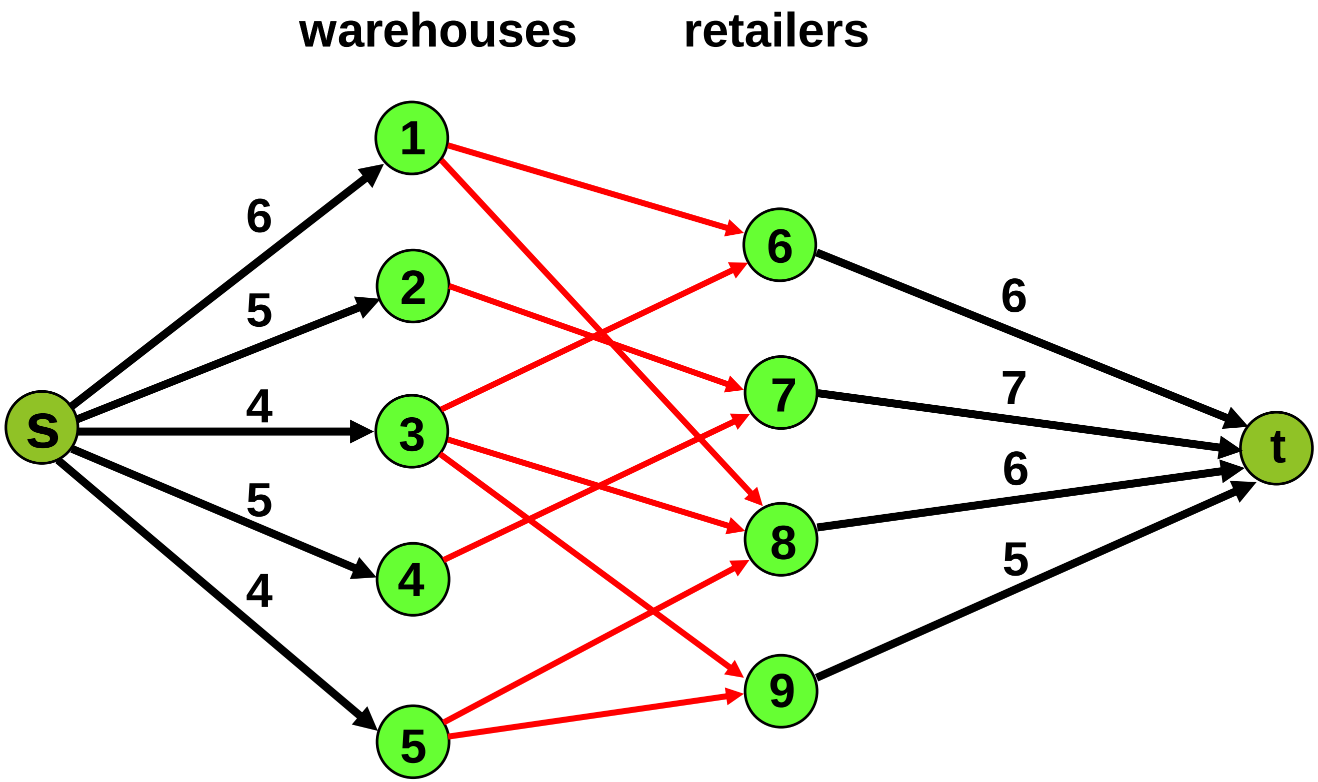
A max flow problem
Capacities and a non-optimum flow

Feasibility Problem: Find a Feasible Flow



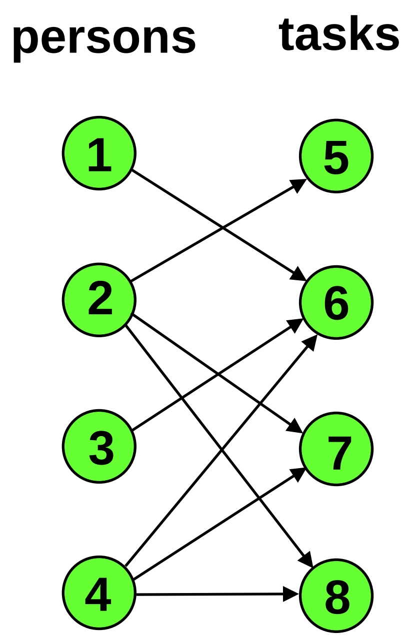
Is there a way of shipping from the warehouses to the retailers to satisfy demand?

Transformation to a Max Flow Problem



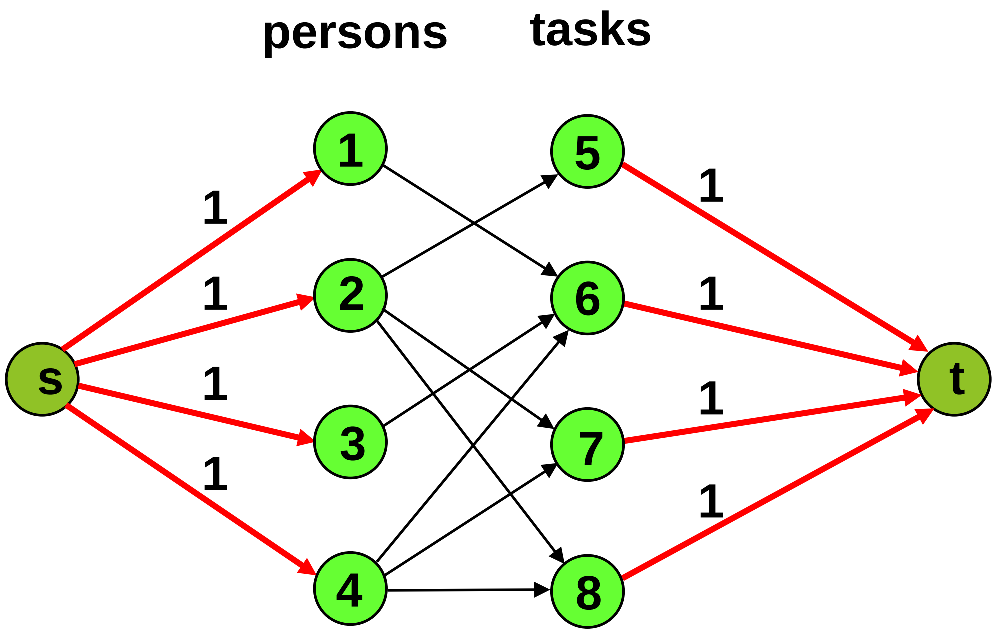
There is a 1-1 correspondence with flows from s to t with 24 units (why 24?) and feasible flows for the transportation problem

Feasibility Problem: Find a Matching



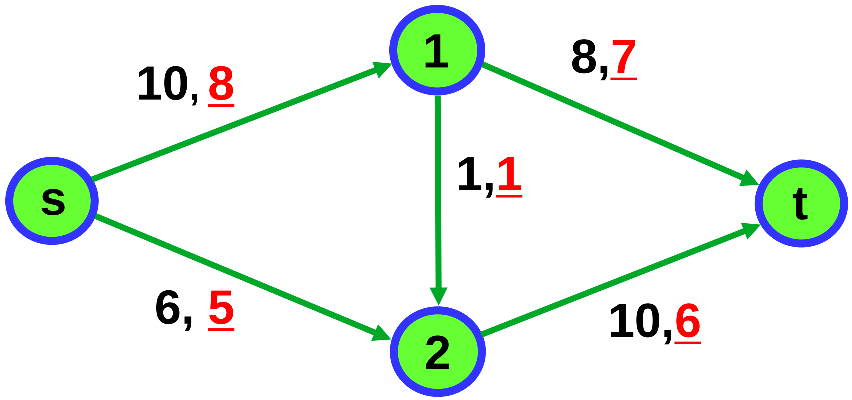
Is there a way of assigning persons to tasks so that each person is assigned a task, and each task has a person assigned to it?

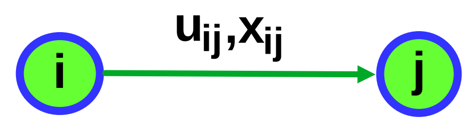
Transformation to a Max Flow Problem

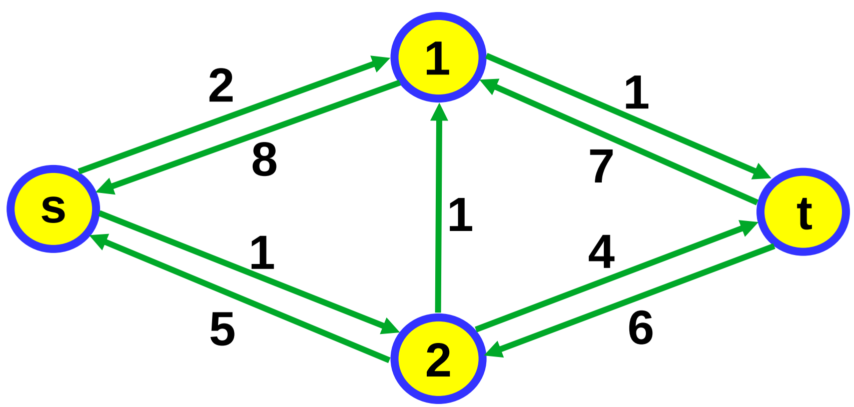


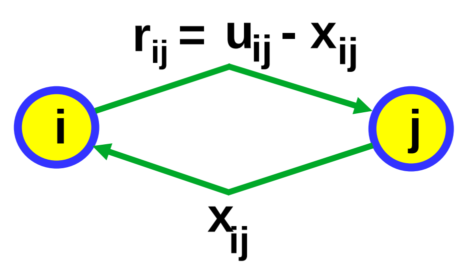
Does the maximum flow from s to t have 4 units?

Residual Networks







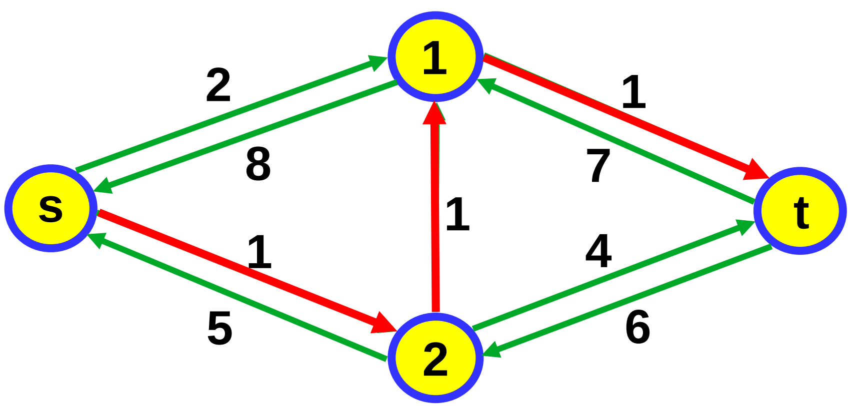


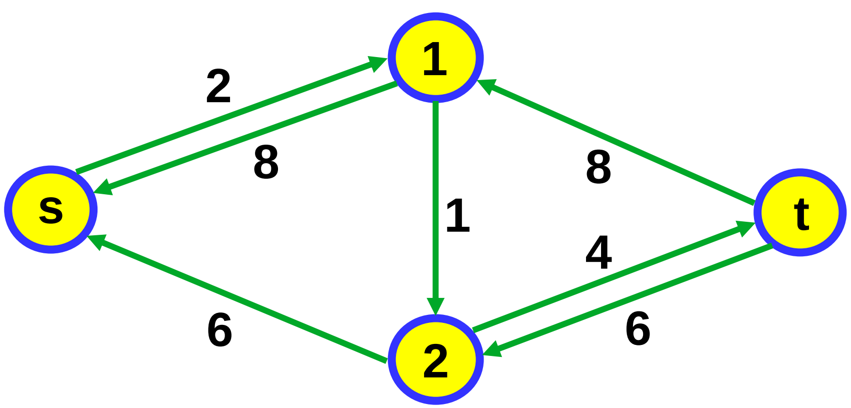
The *Residual Network* G(x)

Let r_{ij} denote the *residual capacity* of arc (i,j)

Augmenting Paths

- An *augmenting path* is a path from s to t in the residual network
- The *residual capacity* of the augmenting path P
  - $\delta(P) = \min\{r_{ij} : (i,j) \in P\}$
- To *augment along P*, send $\delta(P)$ units of flow along each arc of the path, then, modify x and the residual capacities appropriately
- $r_{ij} := r_{ij} - \delta(P)$ and $r_{ji} := r_{ji} + \delta(P)$ for $(i,j) \in P$

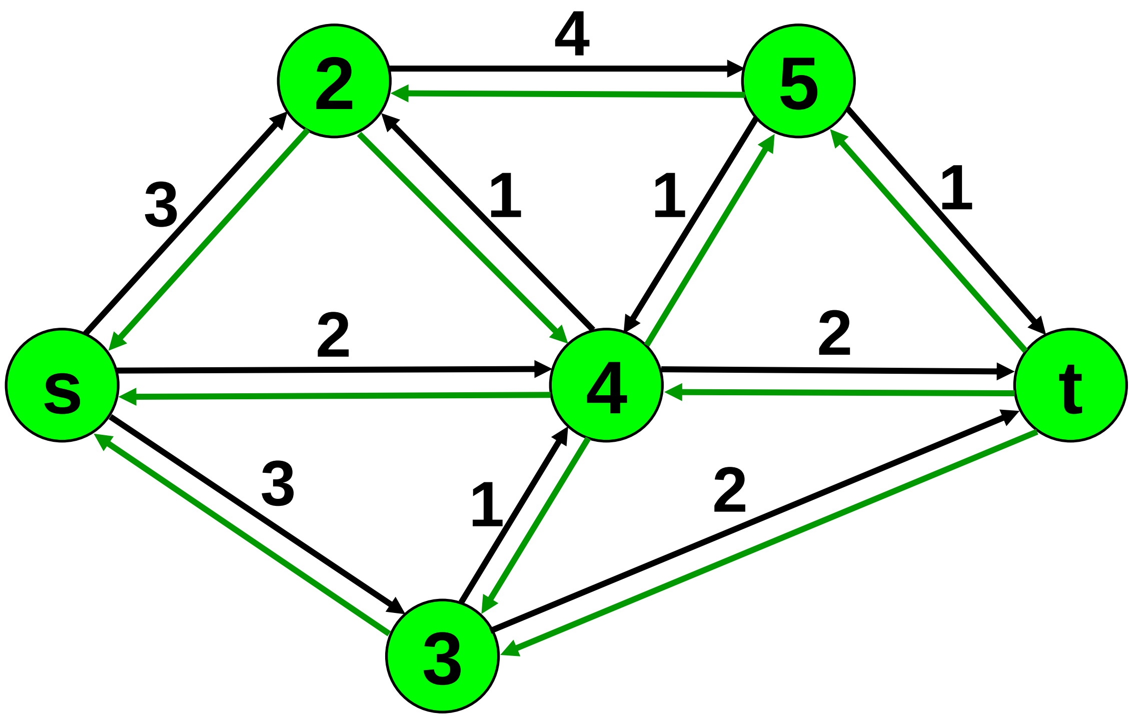




Ford-Fulkerson Max-Flow (1/16)

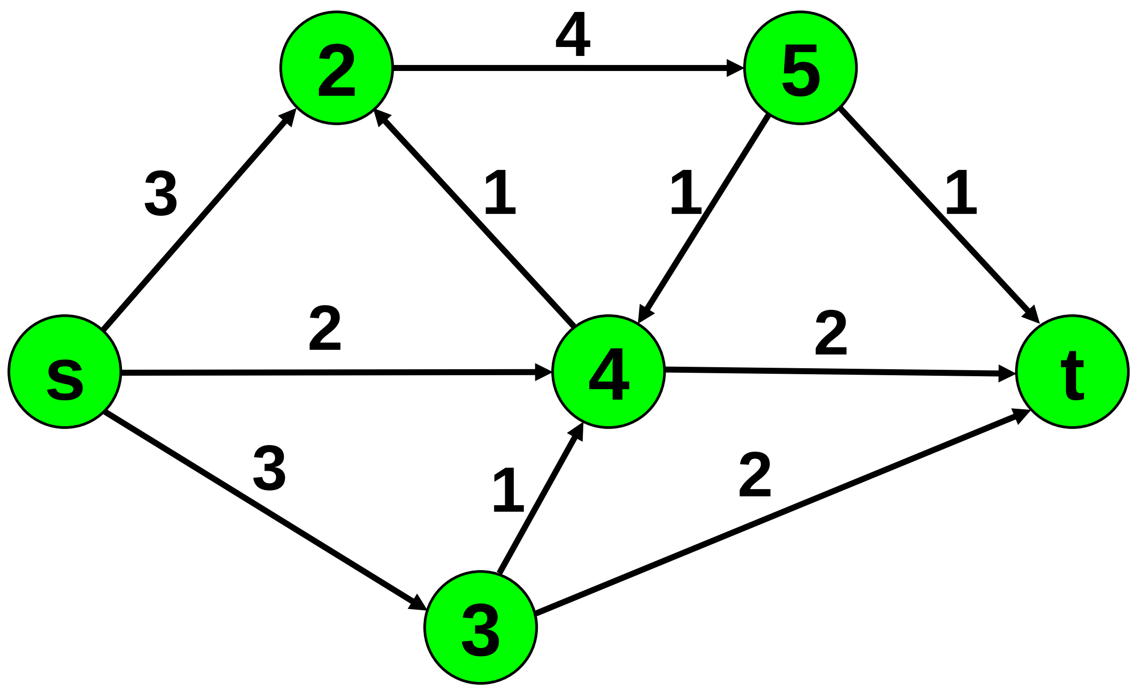
- Begin
  - x := 0
  - create the residual network G(x)
  - while there is some directed path from s to t in G(x) do
  - begin
    - let P be a path from s to t in G(x)
    - $\Delta := \delta(P)$
    - send Δ units of flow along P
    - update the r's
  - end
- End {the flow x is now maximum}

Ford-Fulkerson Max-Flow (2/16)



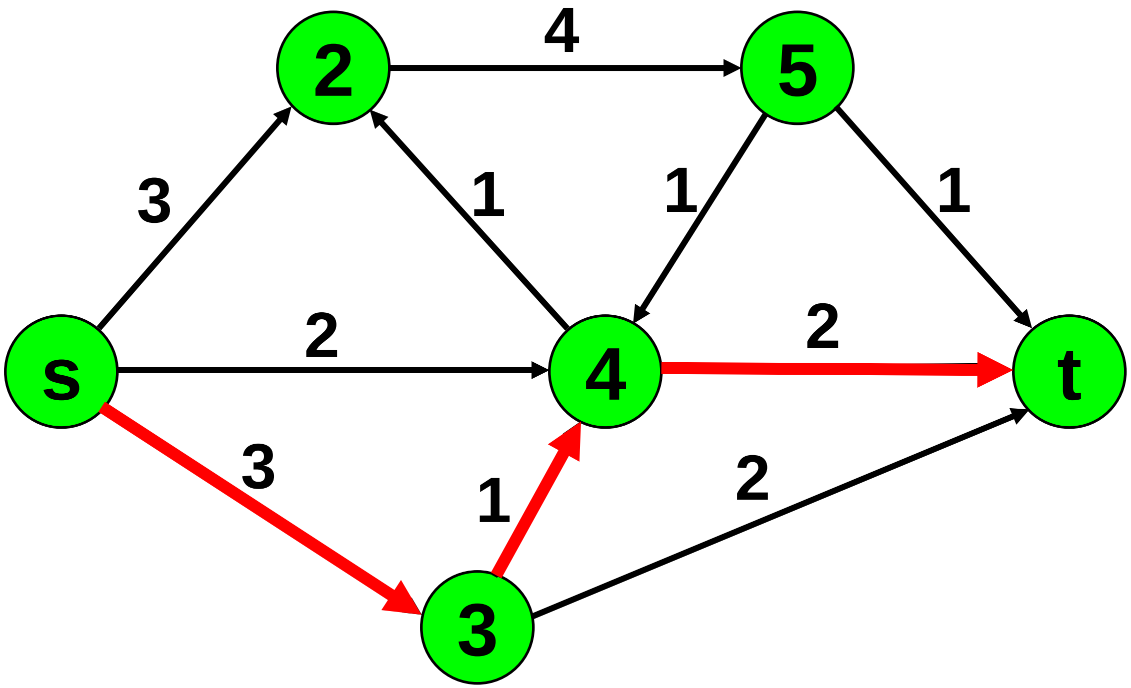
This is the original network, plus reversals of the arcs

Ford-Fulkerson Max-Flow (3/16)



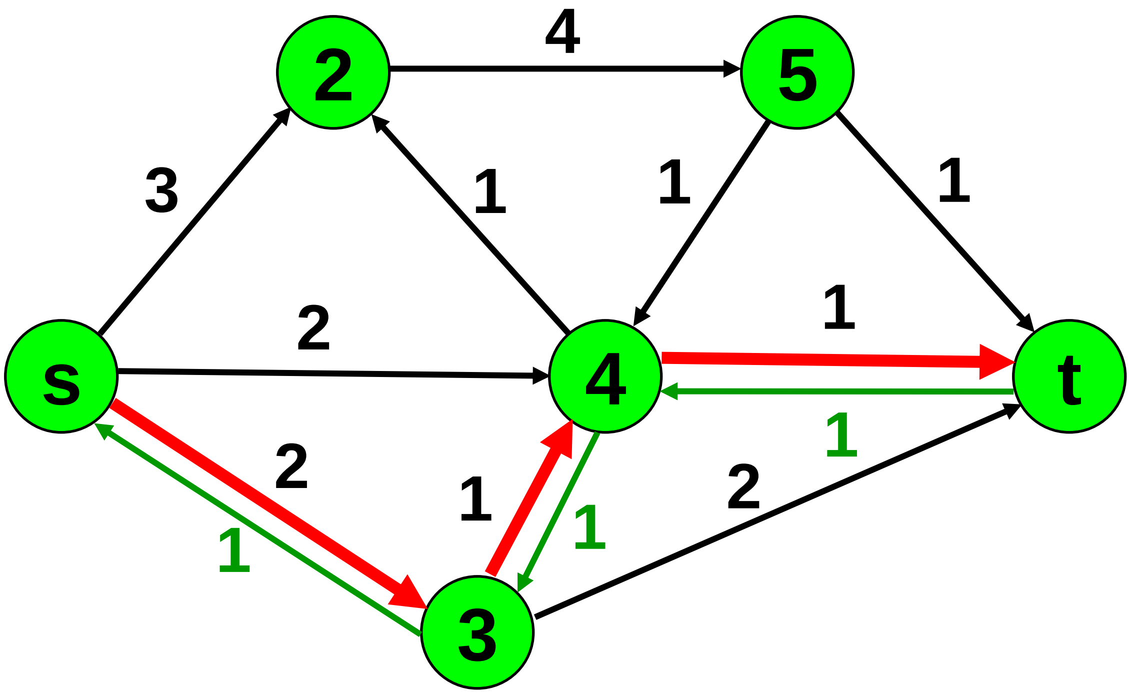
This is the original network, and the original residual network

Ford-Fulkerson Max-Flow (4/16)



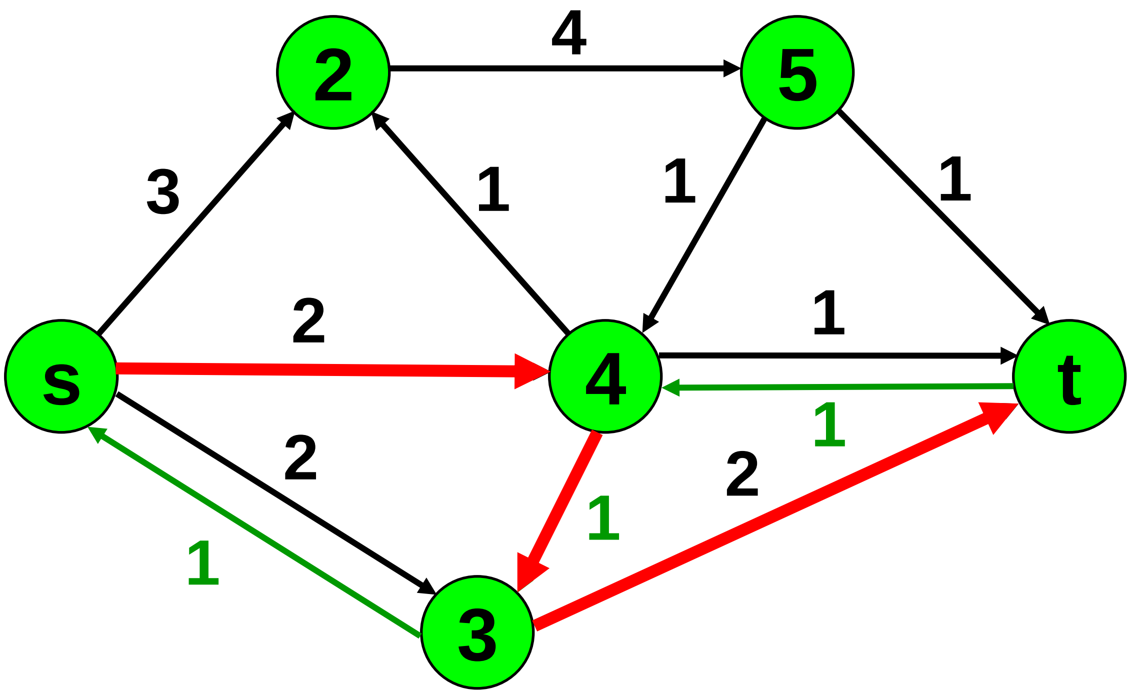
Find any s-t path in G(x)

Ford-Fulkerson Max-Flow (5/16)



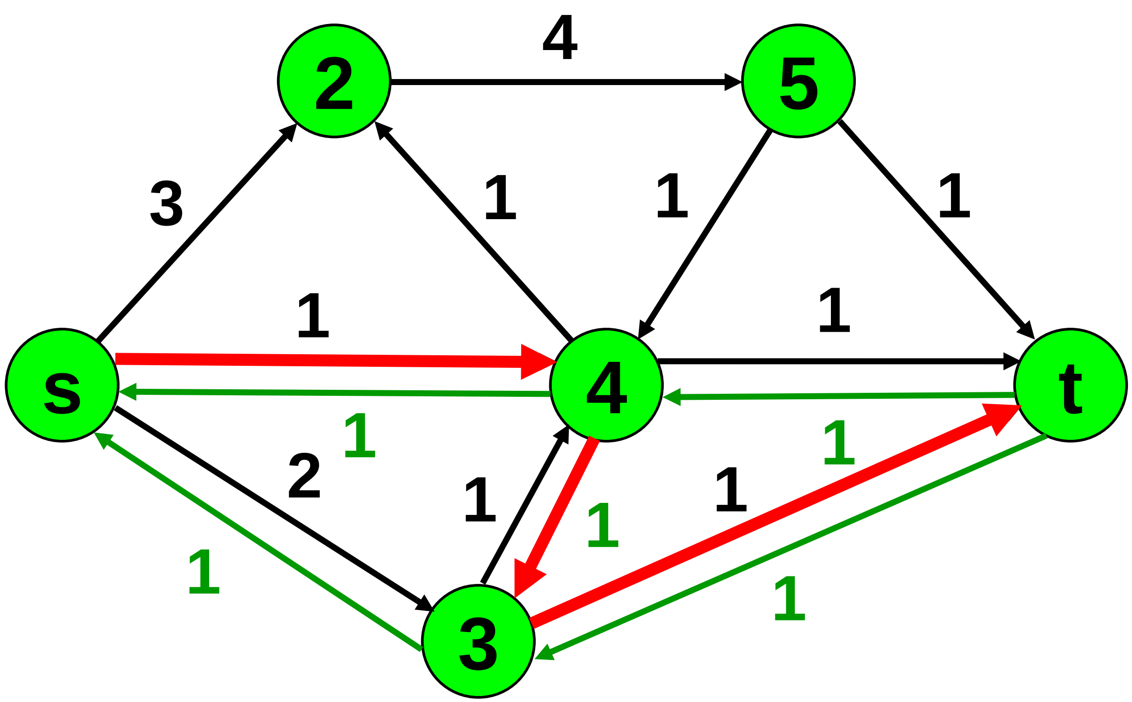
Determine residual capacity Δ of the path
Send Δ units of flow in the path
Update residual capacities

Ford-Fulkerson Max-Flow (6/16)



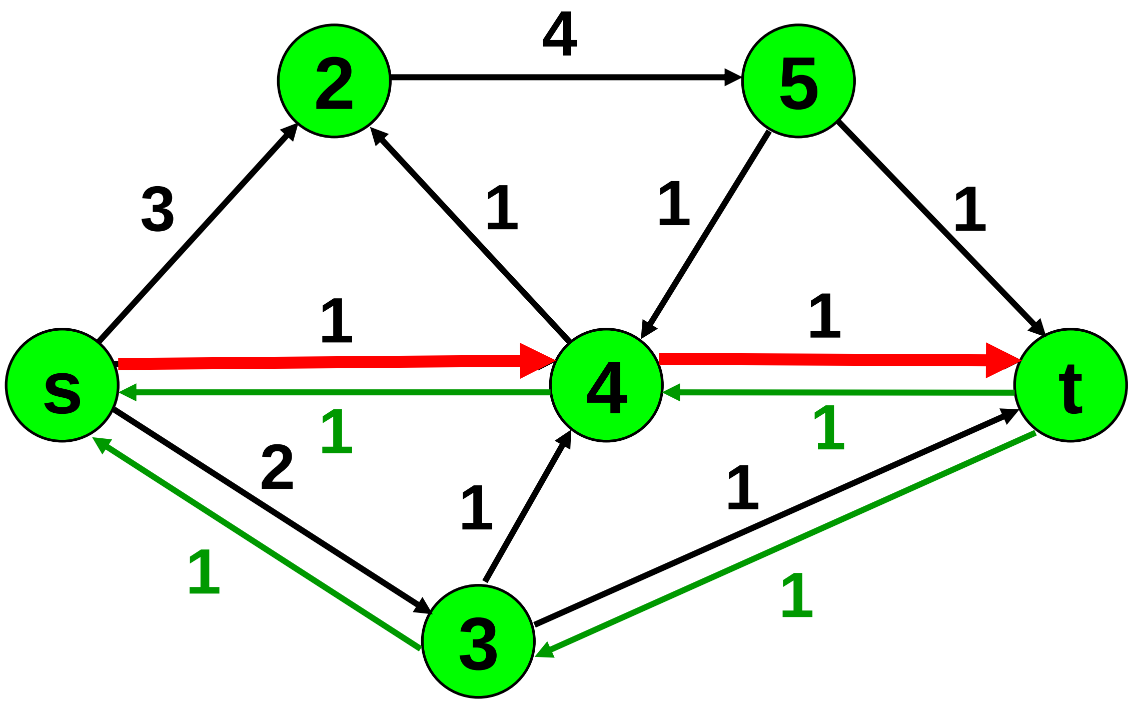
Find any s-t path

Ford-Fulkerson Max-Flow (7/16)



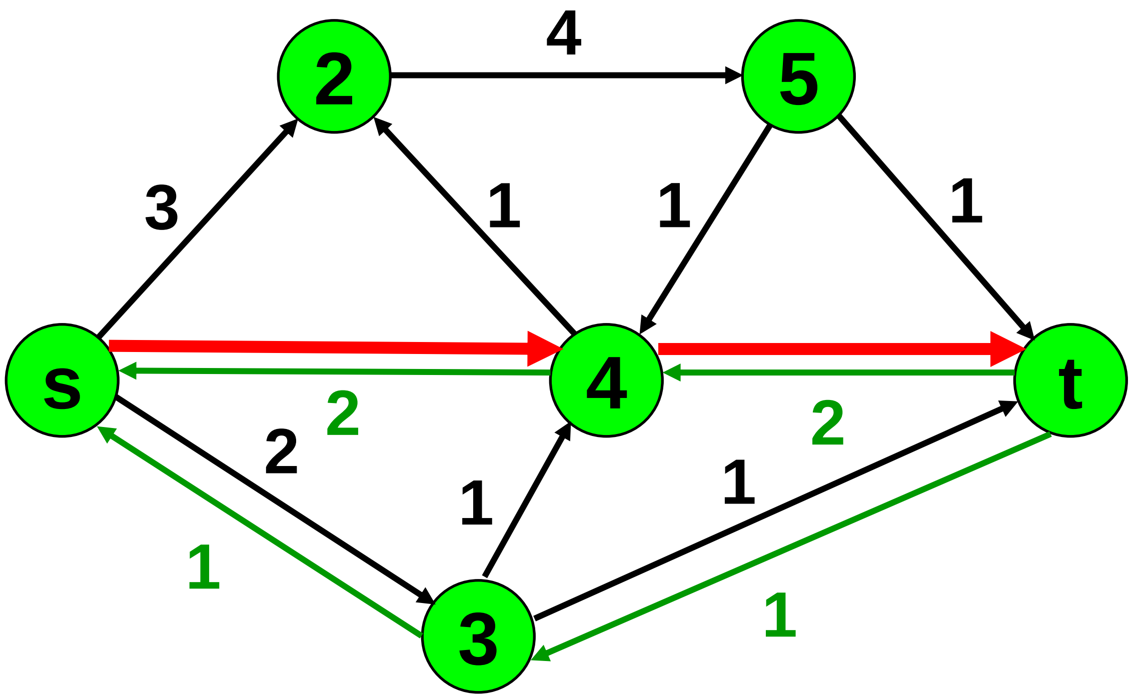
Determine the residual capacity Δ of the path
Send Δ units of flow in the path
Update residual capacities

Ford-Fulkerson Max-Flow (8/16)



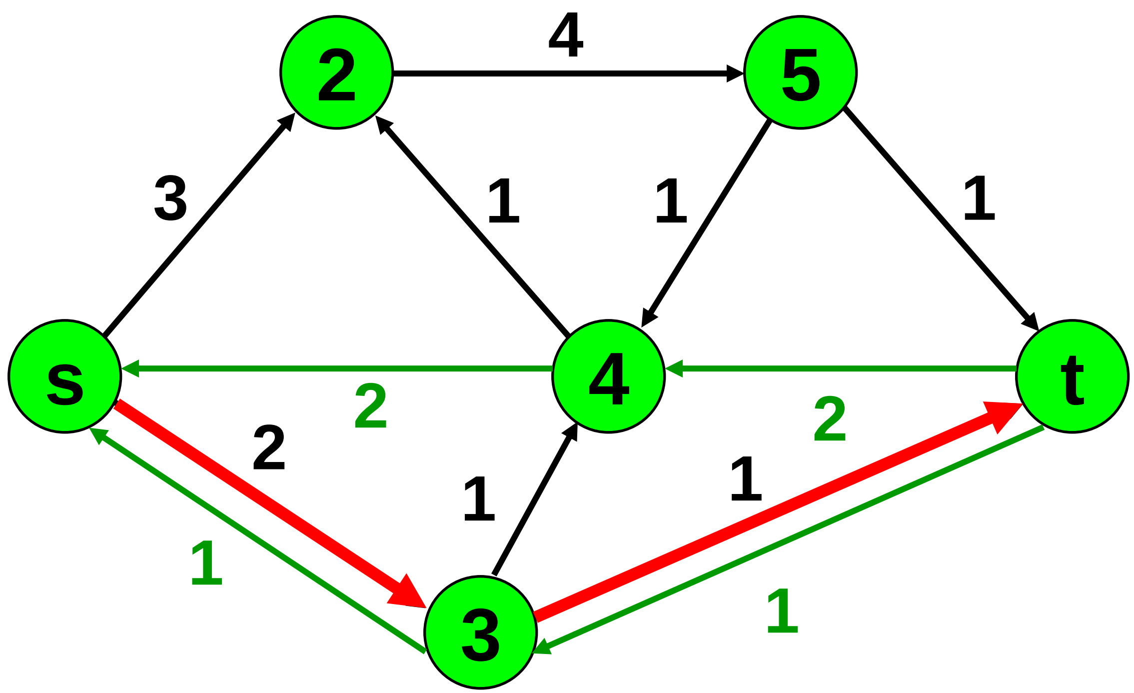
Find any s-t path

Ford-Fulkerson Max-Flow (9/16)



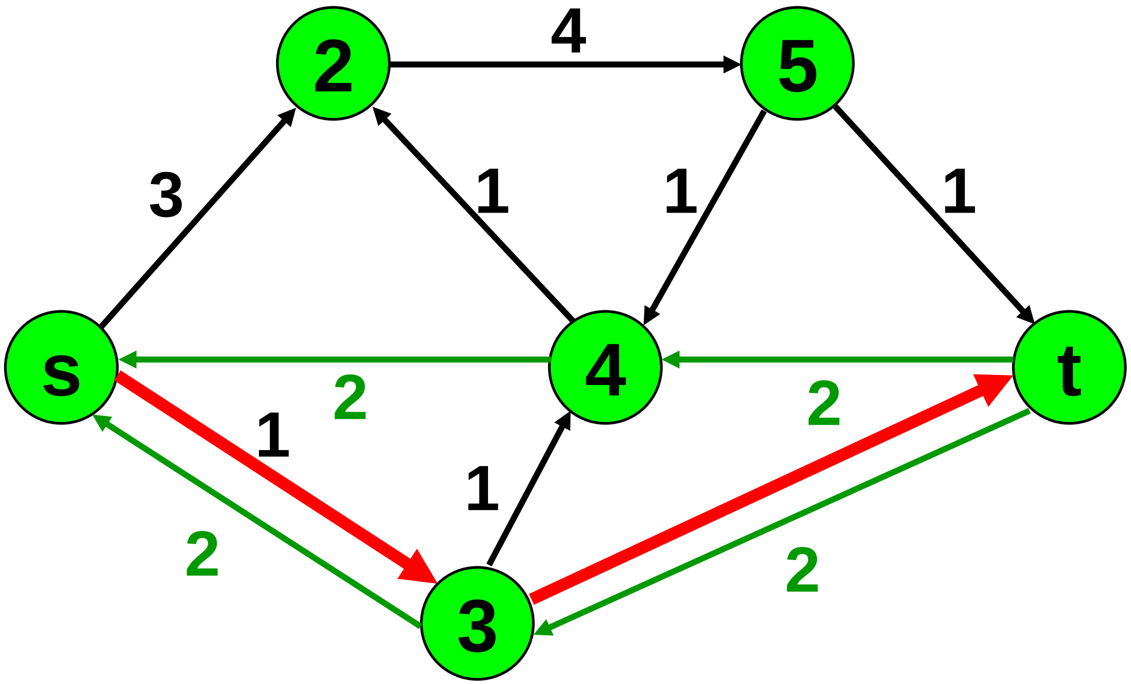
Determine the residual capacity Δ of the path
Send Δ units of flow in the path
Update residual capacities

Ford-Fulkerson Max-Flow (10/16)



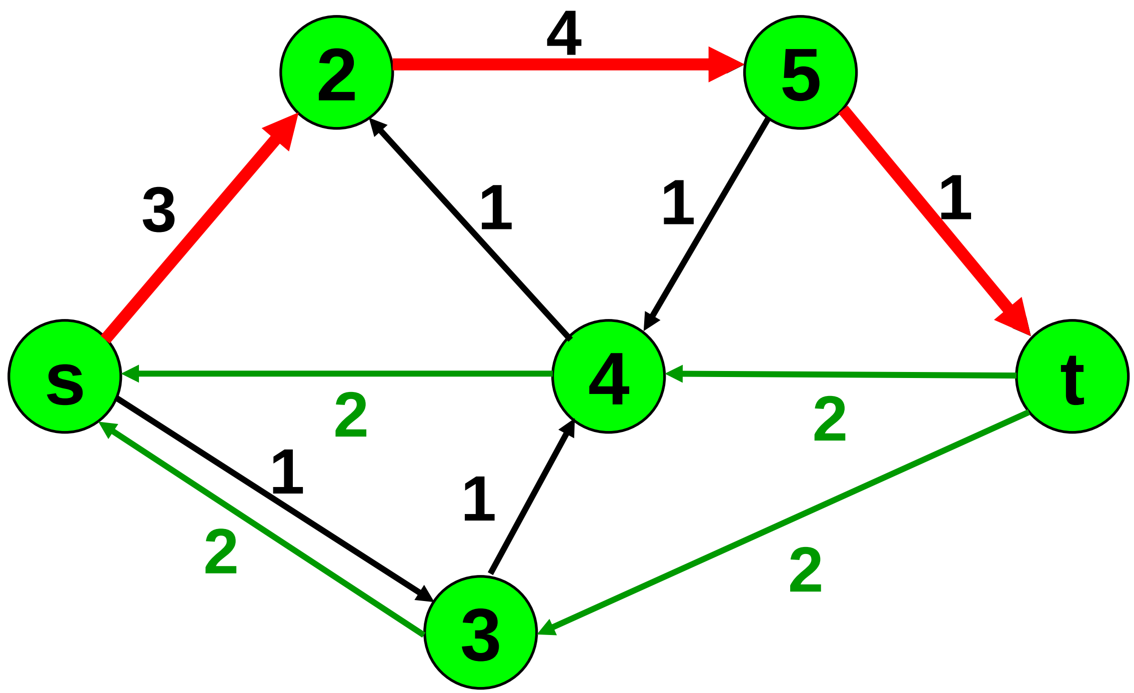
Find any s-t path

Ford-Fulkerson Max-Flow (11/16)



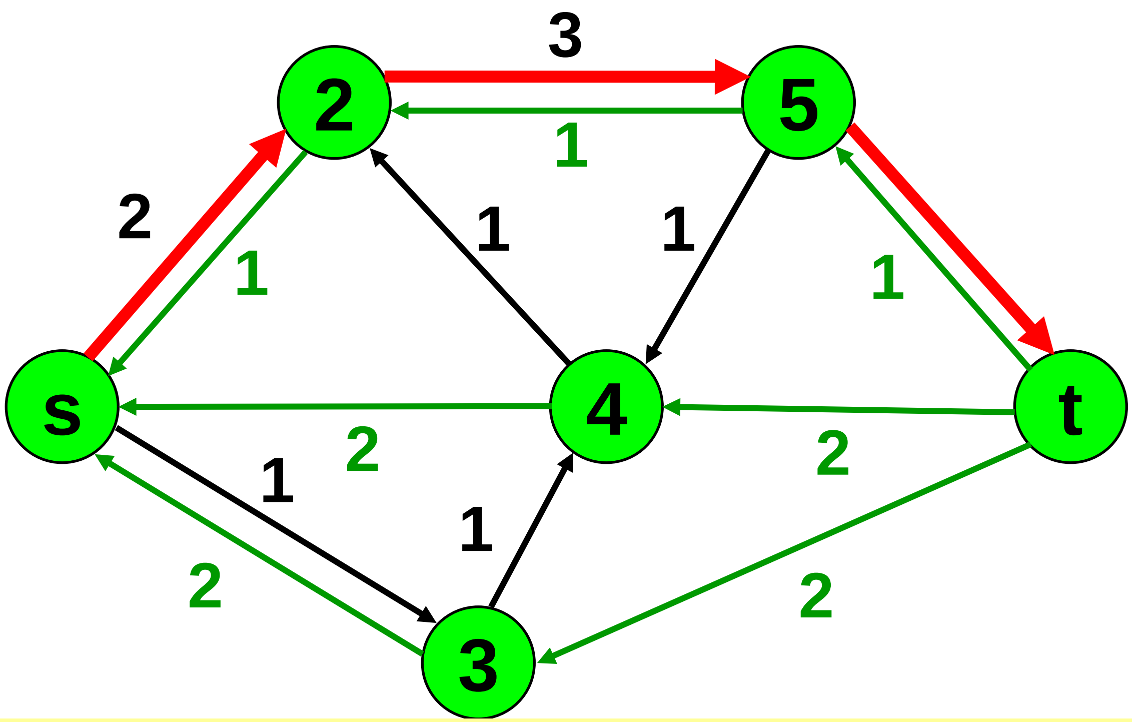
Determine the residual capacity Δ of the path
Send Δ units of flow in the path
Update residual capacities

Ford-Fulkerson Max-Flow (12/16)



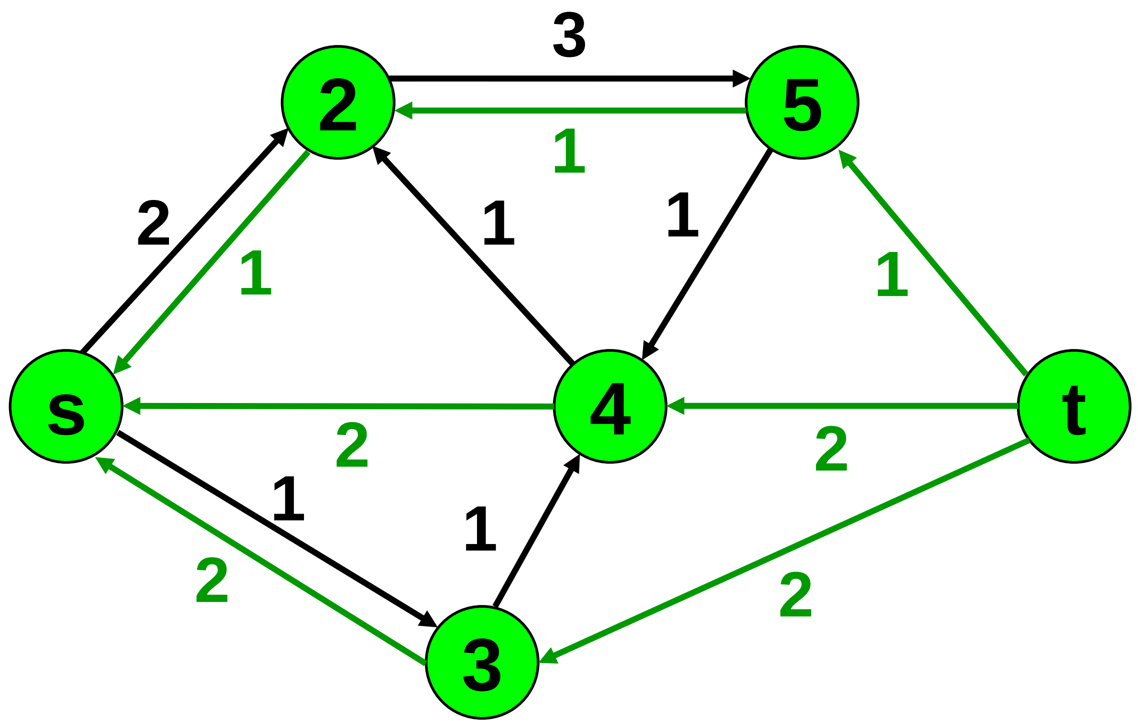
Find any s-t path

Ford-Fulkerson Max-Flow (13/16)



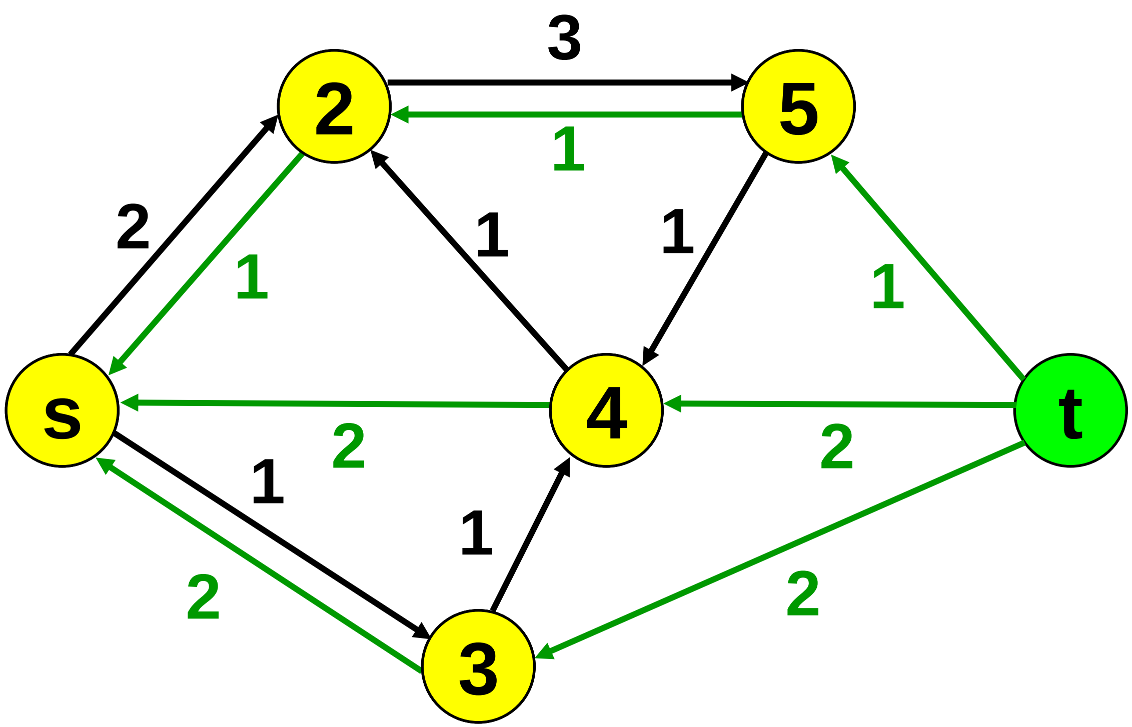
Determine the residual capacity Δ of the path
Send Δ units of flow in the path
Update residual capacities

Ford-Fulkerson Max-Flow (14/16)



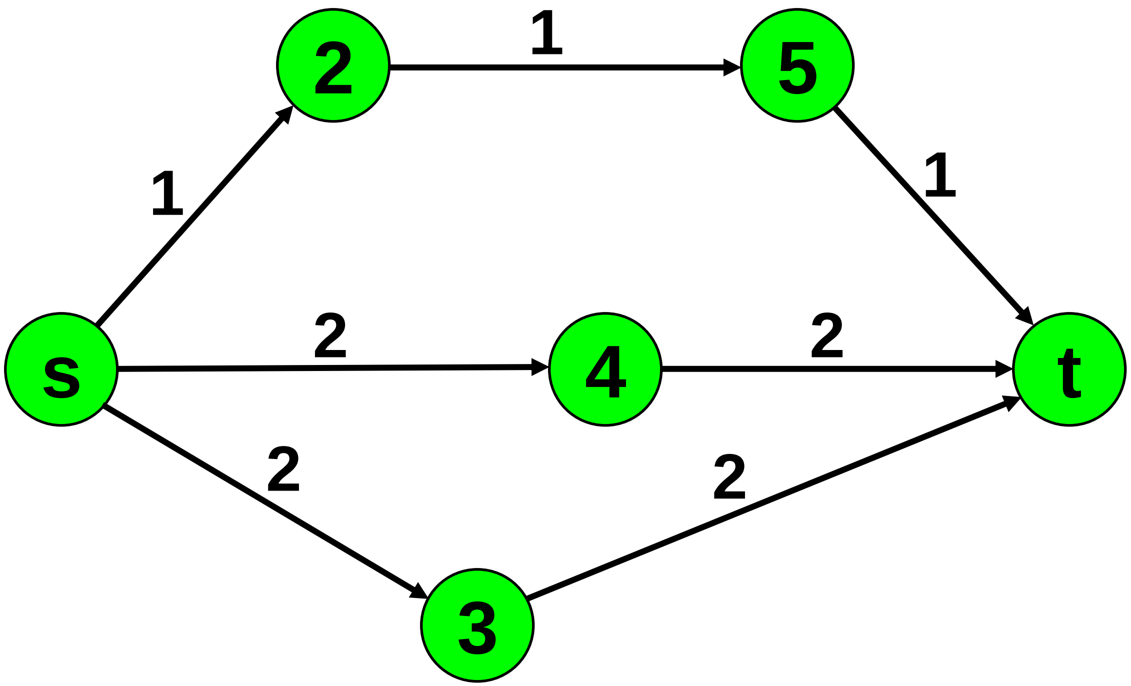
There is no s-t path in the residual network
This flow is optimal

Ford-Fulkerson Max-Flow (15/16)



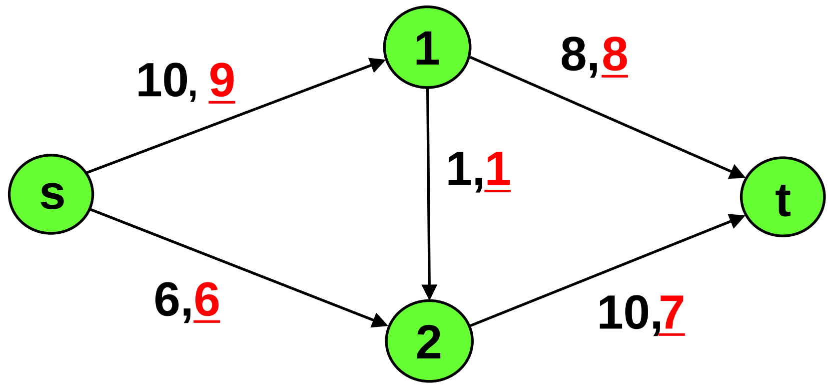
These are the nodes that are reachable from node s

Ford-Fulkerson Max-Flow (16/16)

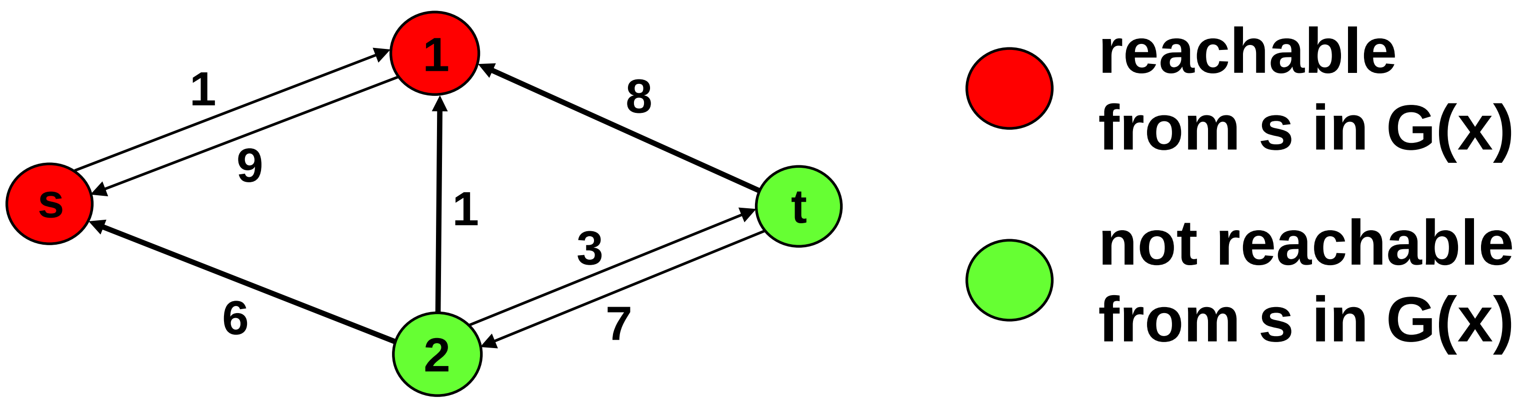


Here is the optimal flow

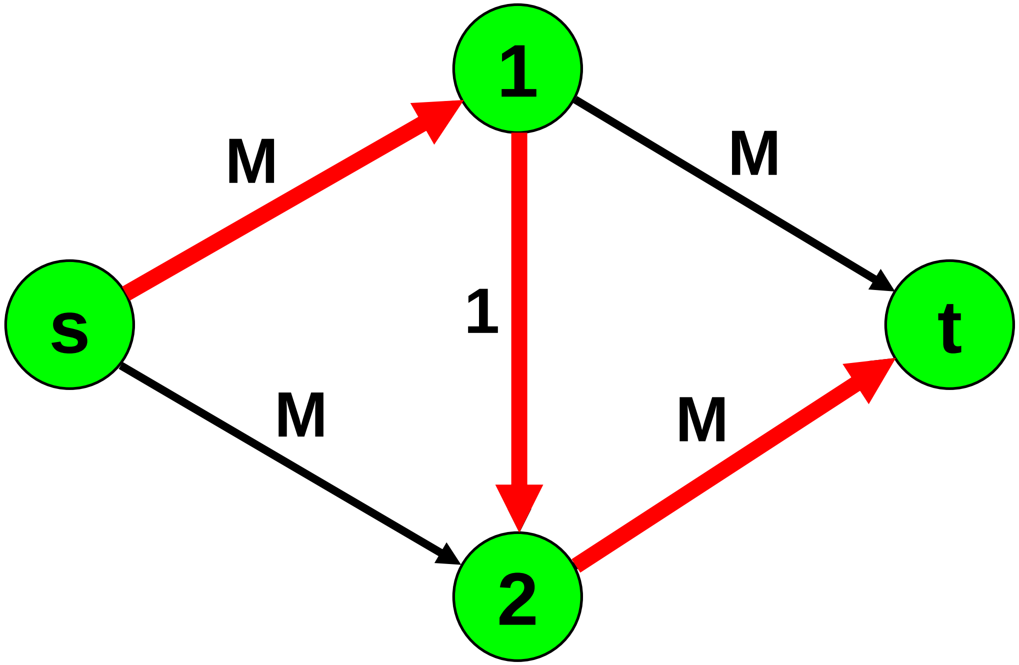
How Do We Know When a Flow Is Optimal?



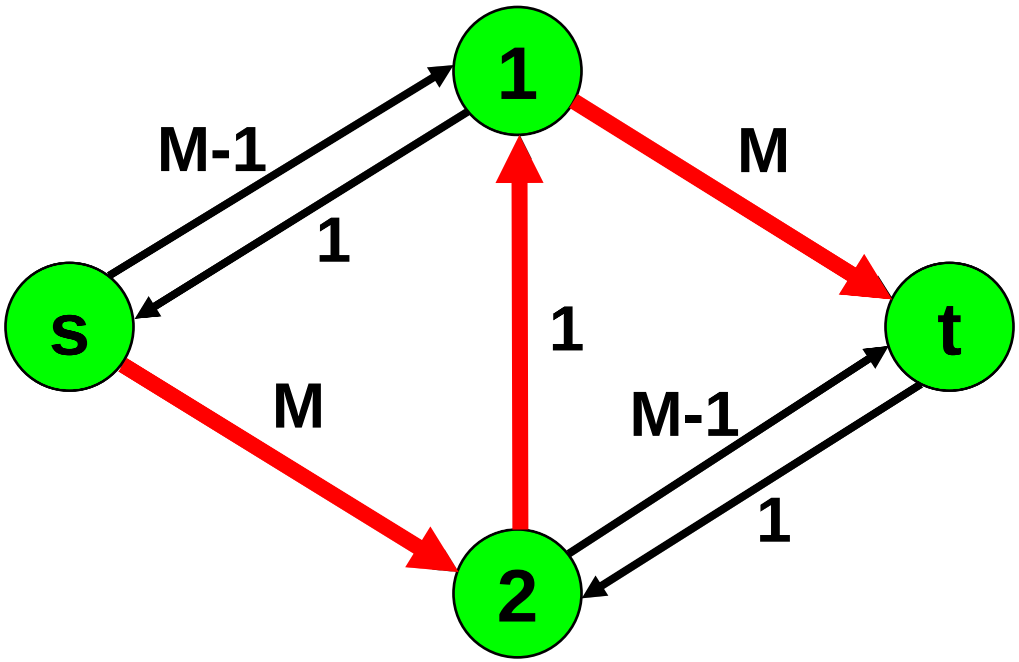
- METHOD
  - There is no augmenting path in the residual network



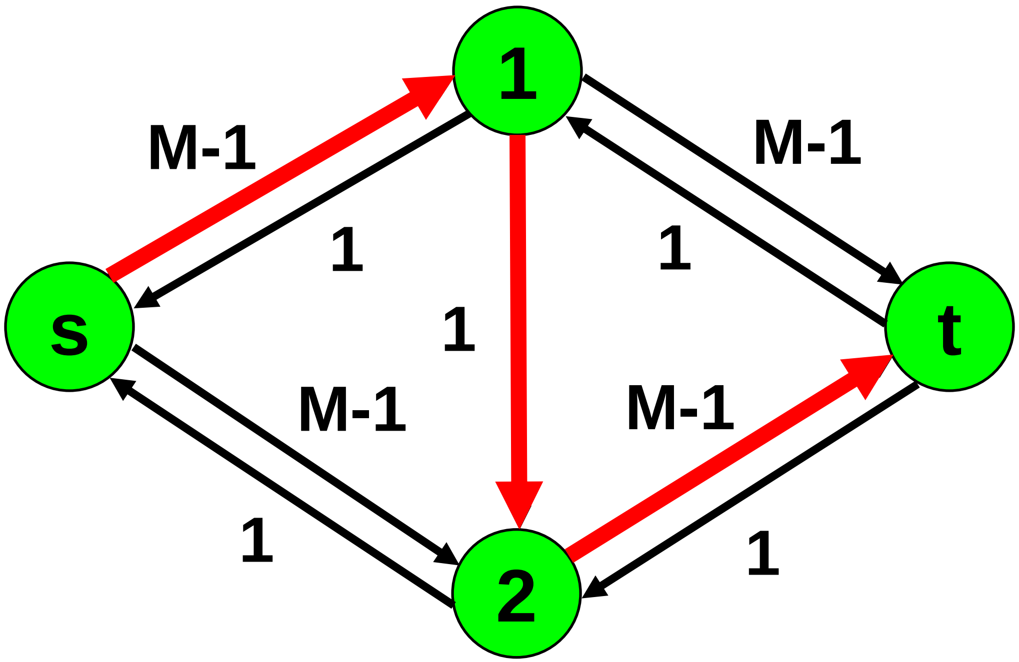
A Simple and Very Bad Example



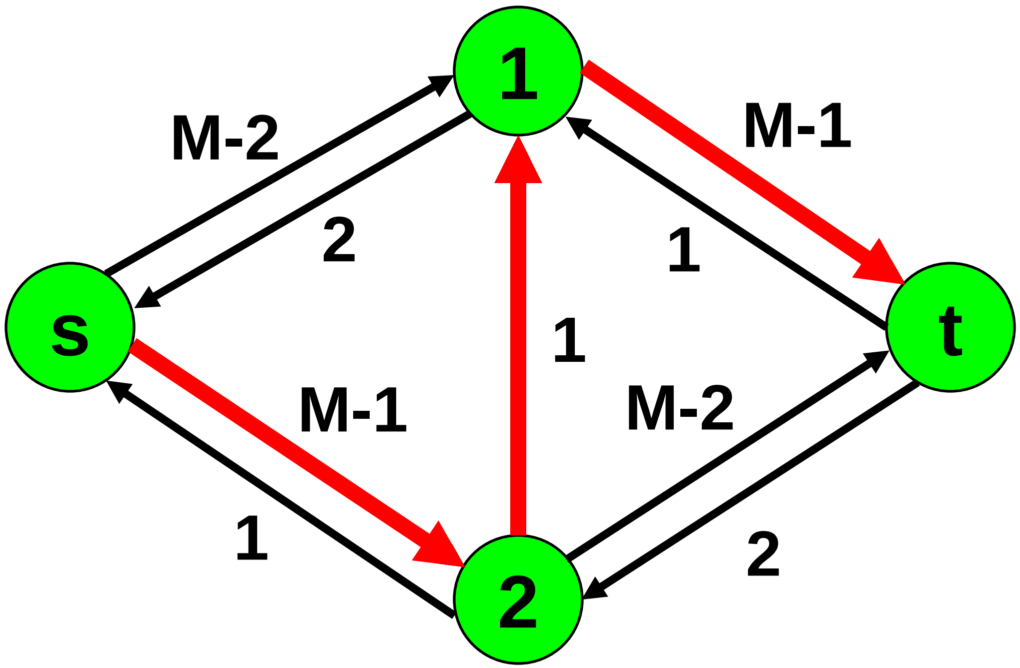
After 1 Augmentation



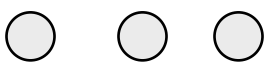
After Two Augmentations



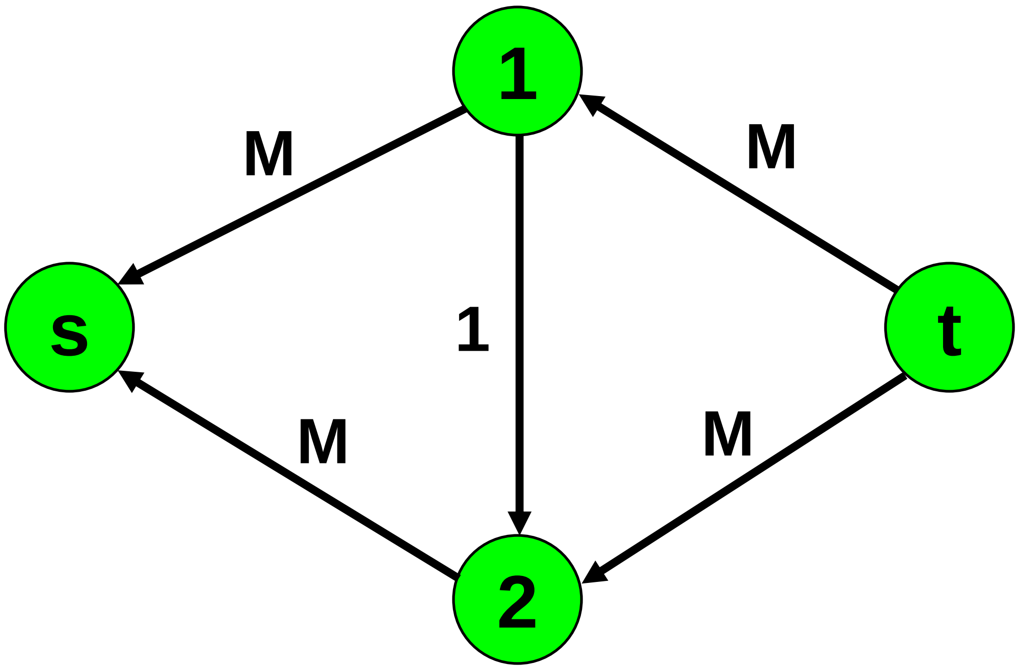
After Three Augmentations



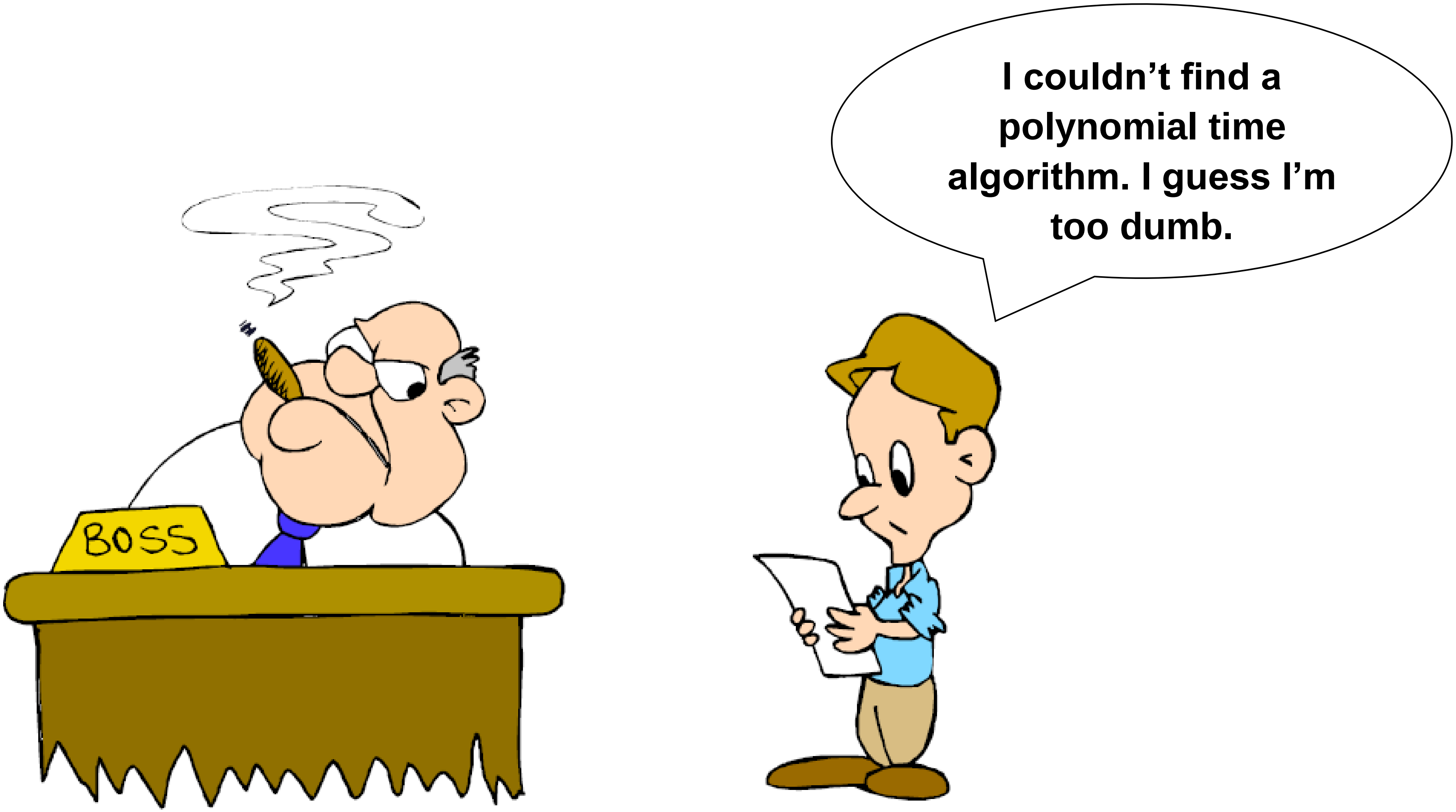
And so on



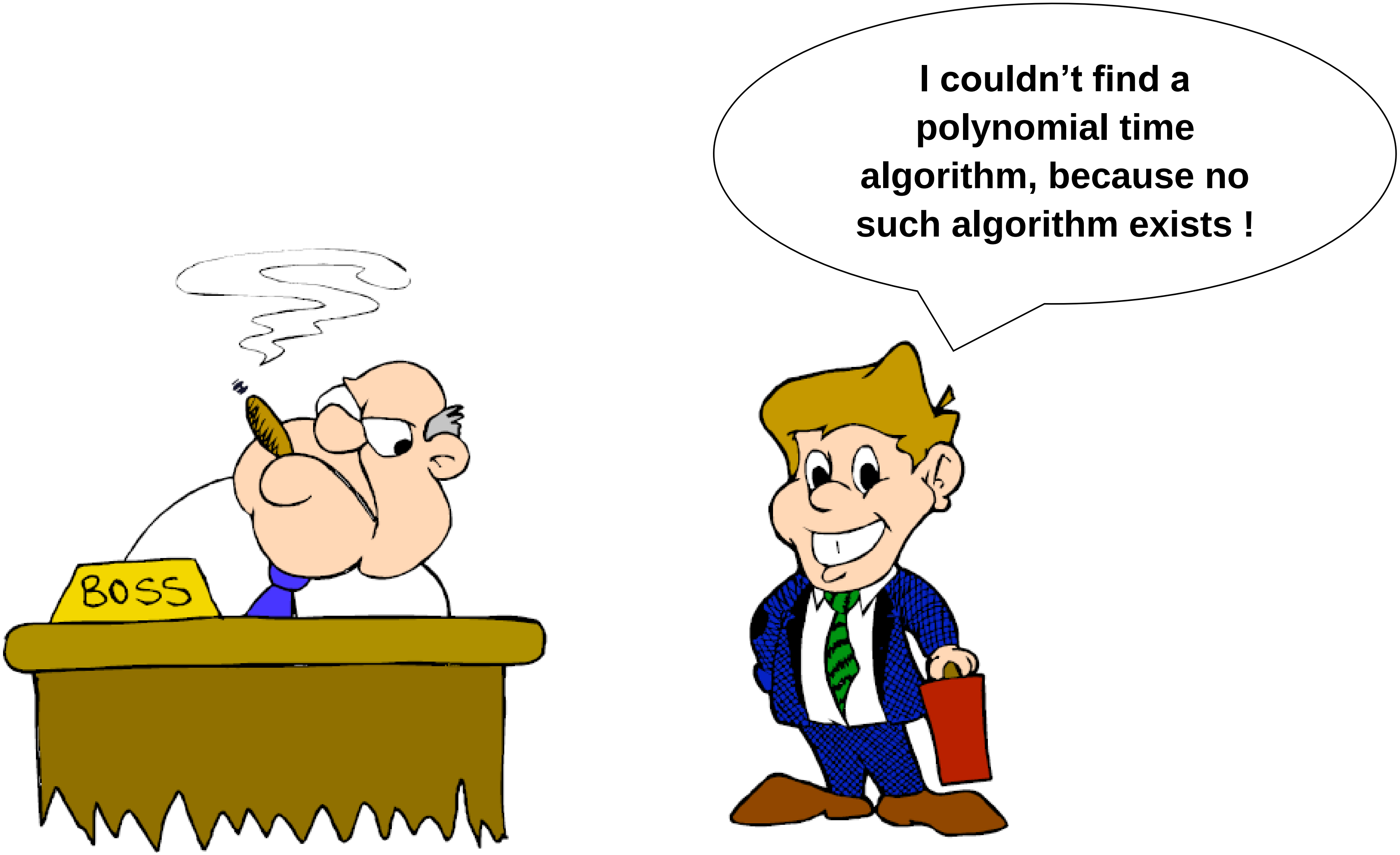
After 2M Augmentations



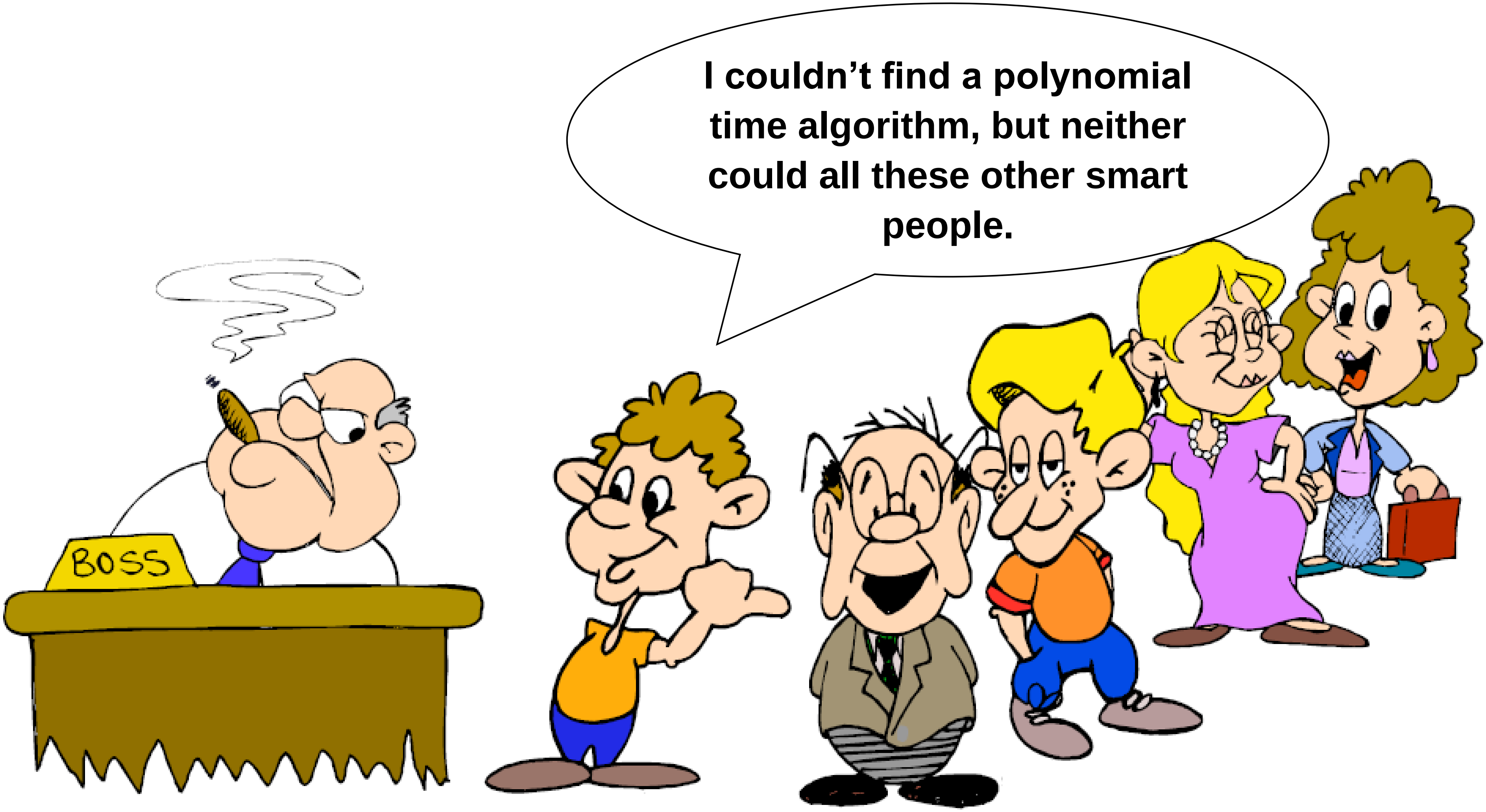
P or NP


BOSS
I couldn't find a polynomial time algorithm. I guess I'm too dumb.

P or NP


I couldn't find a polynomial time algorithm, because no such algorithm exists !
BOSS

P or NP


I couldn't find a polynomial time algorithm, but neither could all these other smart people.
BOSS

P or NP

NP-Completeness

- Some problems are *intractable*
  - They grow large, we are unable to solve them in reasonable time
- What constitutes reasonable time?
  Standard working definition: *polynomial time*
  - For an input of size n, the worst-case running time is $O(n^k)$ for some constant k
  - Polynomial time: $O(1)$, $O(n \lg n)$, $O(n^2)$, $O(n^3)$
  - Not in polynomial time: $O(2^n)$, $O(n^n)$, $O(n!)$

Polynomial-Time Algorithms

- Are some problems solvable in polynomial time?
  - Every algorithm we've studied provides polynomial-time solution to some problems
  - We define **P** to be the class of problems solvable in polynomial time
- Are all problems solvable in polynomial time?
  - No: Turing's "Halting Problem" is not solvable by any computer, no matter how much time is given
  - Such problems are clearly intractable, not in **P**

NP-Complete Problems

- Definition
  - The class P (or NP) denotes the family of all problems that can be solved by deterministic (nondeterministic) polynomial time algorithms
    All decision problems only answer YES or NO
- Nondeterministic polynomial time algorithm exist?

NP-Complete Problems

- The *NP-Complete* problems are an interesting class of problems whose status is unknown
  - No polynomial-time algorithm has been discovered for an NP-Complete problem
- We call this the *P* = *NP?*
  - The biggest open problem in CS
- Examples of NP-Complete problems:
  - Vertex cover problem
  - Traveling salesman problem
  - Job scheduling problem
  - …

Thanks to contributors

Mr. Pham Van Nguyen (2022)

Dr. Thien-Binh Dang (2017 – 2022)

Prof. Hyunseung Choo (2001 – 2022)