COMP 446 / 546 ALGORITHM DESIGN AND ANALYSIS

LECTURE 12 MAXIMUM FLOW MINIMUM CUT

ALPTEKİN KÜPÇÜ

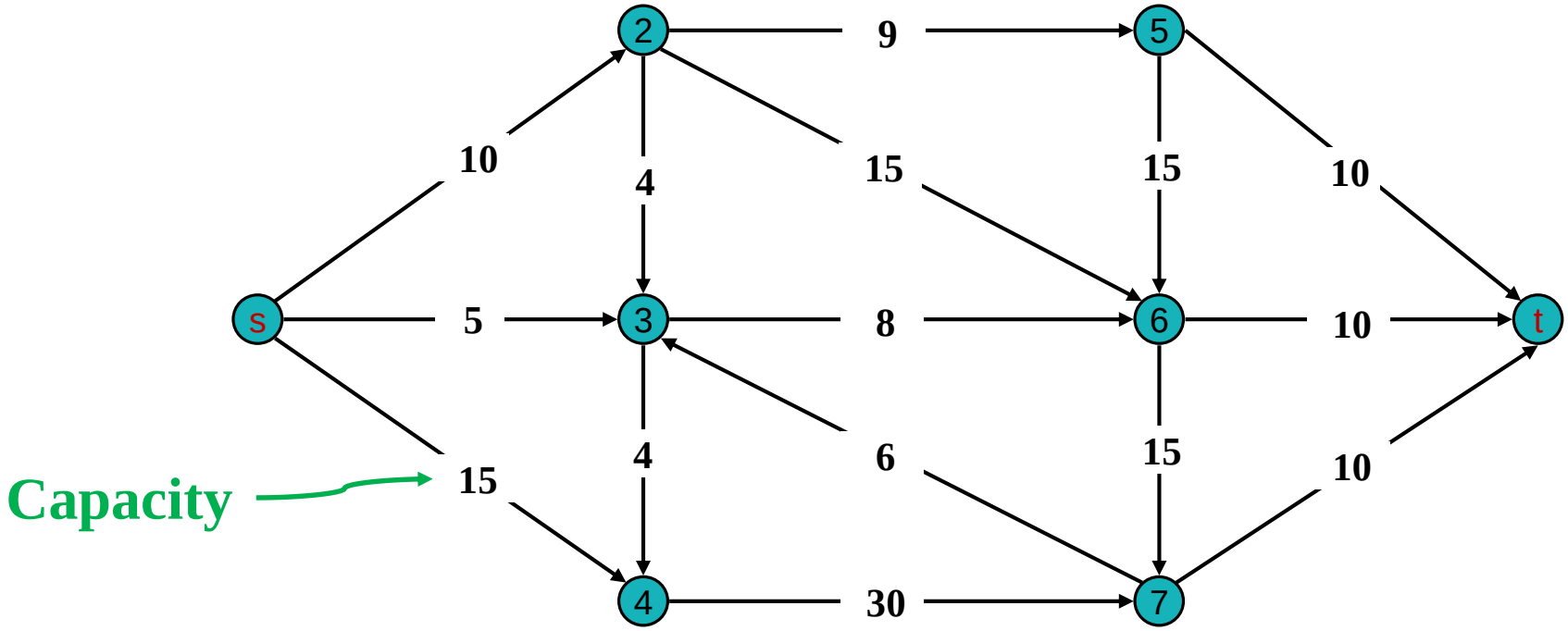
Based on slides of Kevin Wayne

MAXIMUM FLOW AND MINIMUM CUT

- **Max flow and min cut**
  - Two very rich algorithmic problems for combinatorial optimization.
  - Mathematical duality (and introduction to linear programming)
- **Nontrivial applications / reductions**
  - Network connectivity.
  - Bipartite matching.
  - Data mining.
  - Open-pit mining.
  - Airline scheduling.
  - Image processing.
  - Project selection.
  - Baseball elimination.
  - Network reliability.
  - Security of statistical data.
  - Distributed computing.
  - Egalitarian stable matching.
  - Distributed computing.
  - Many more . . .

MAX FLOW NETWORK

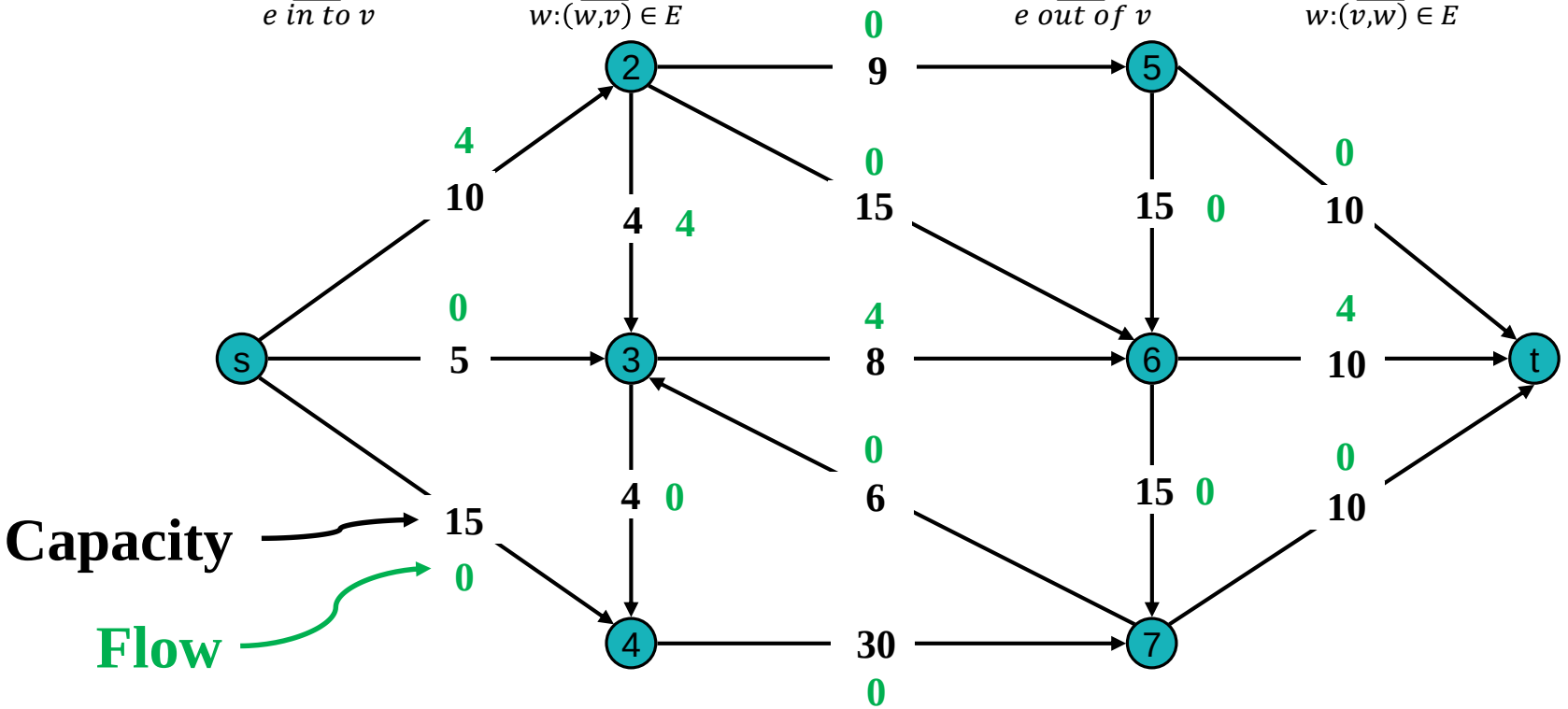
- **Max flow network: G = (V, E, s, t, u)**
  - (V, E): directed graph, no parallel arcs (not a multi-graph).
  - Two distinguished nodes: s: source, t: target (sink).
  - u(e): capacity of edge e.



FLOW

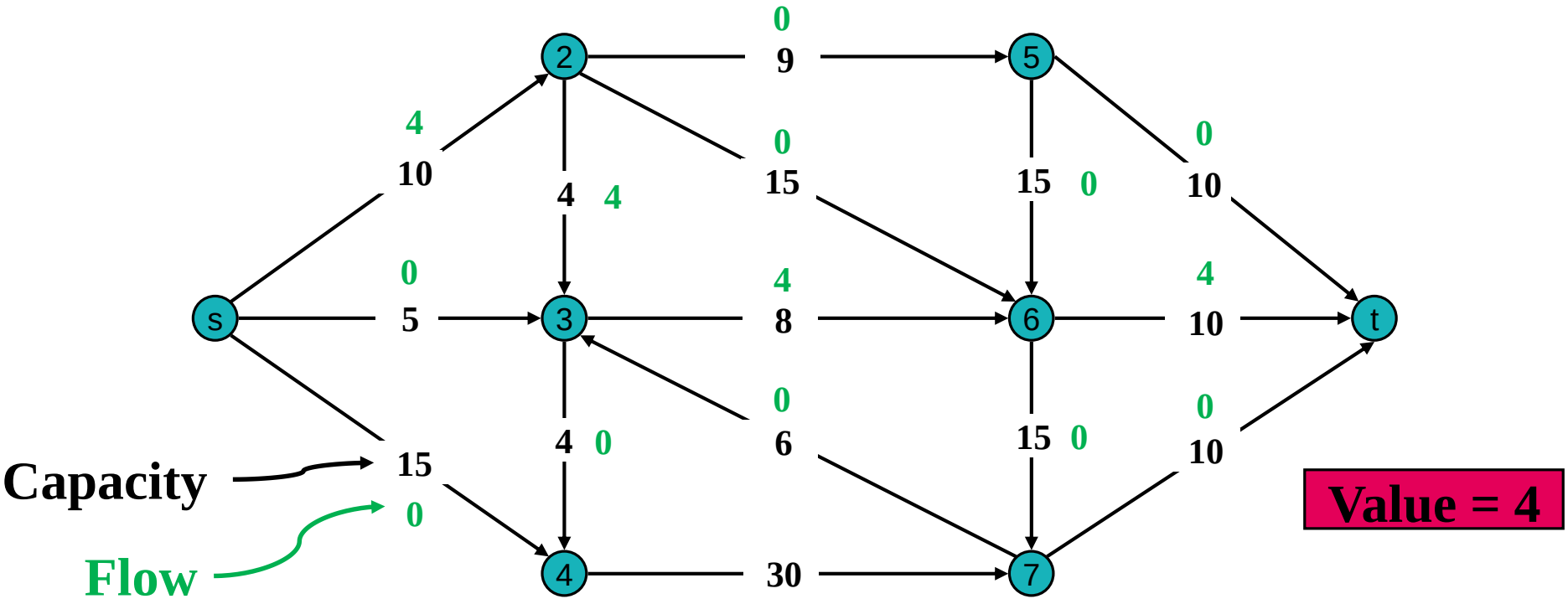
- **An s-t flow is a function f: E → ℜ that satisfies:**
  - For each e ∈ E: $0 \leq f(e) \leq u(e)$ (capacity)
  - For each v ∈ V – {s, t}: $\sum_{e\ in\ to\ v} f(e) = \sum_{e\ out\ of\ v} f(e)$ (conservation)

$$\sum_{e\ in\ to\ v} f(e) = \sum_{w:(w,v)\in E} f(w,v) \qquad \sum_{e\ out\ of\ v} f(e) = \sum_{w:(v,w)\in E} f(v,w)$$



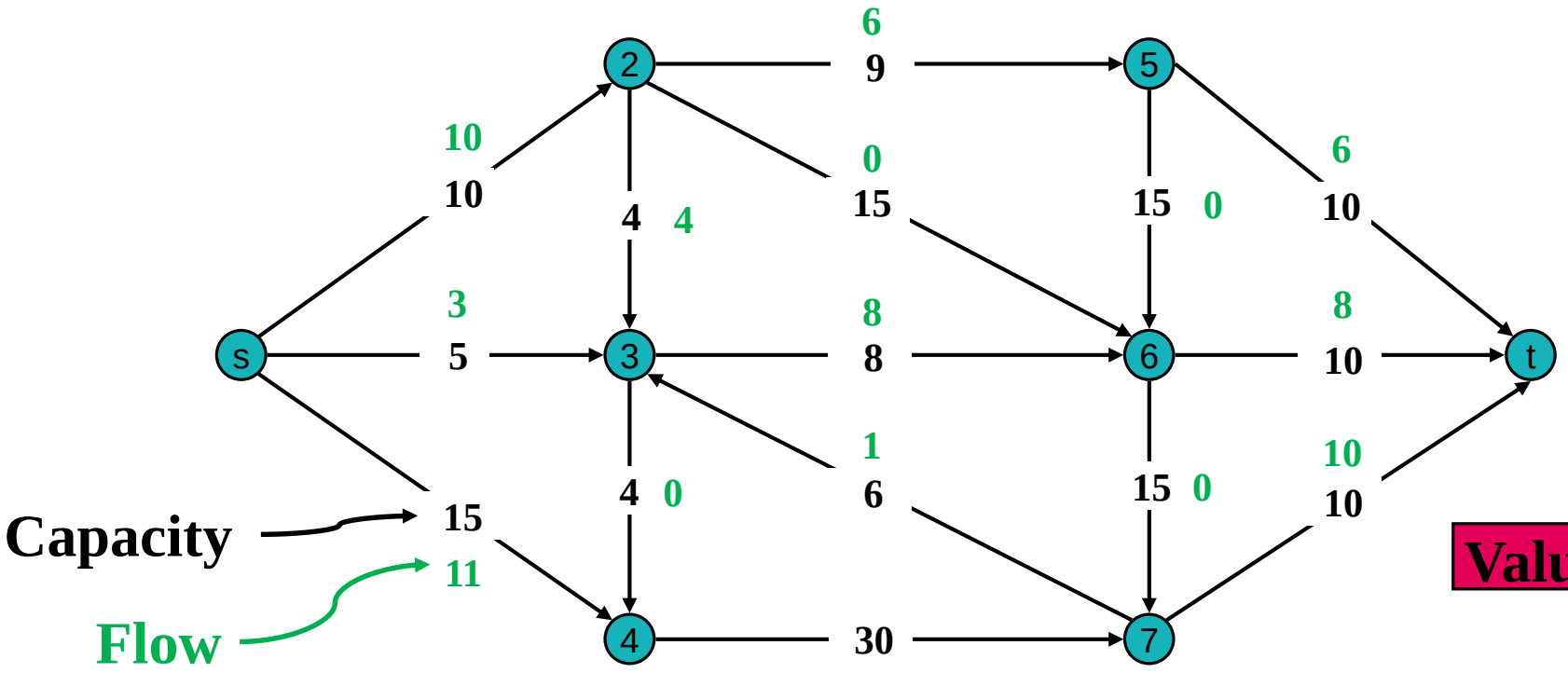
FLOW

- **An s-t flow is a function f: E → ℜ that satisfies:**
  - For each e ∈ E: $0 \leq f(e) \leq u(e)$ (capacity)
  - For each v ∈ V – {s, t}: $\sum_{e\ in\ to\ v} f(e) = \sum_{e\ out\ of\ v} f(e)$ (conservation)
- **Max Flow: s-t flow that maximizes flow out of s** $|f| = \sum_{e\ out\ of\ s} f(e)$



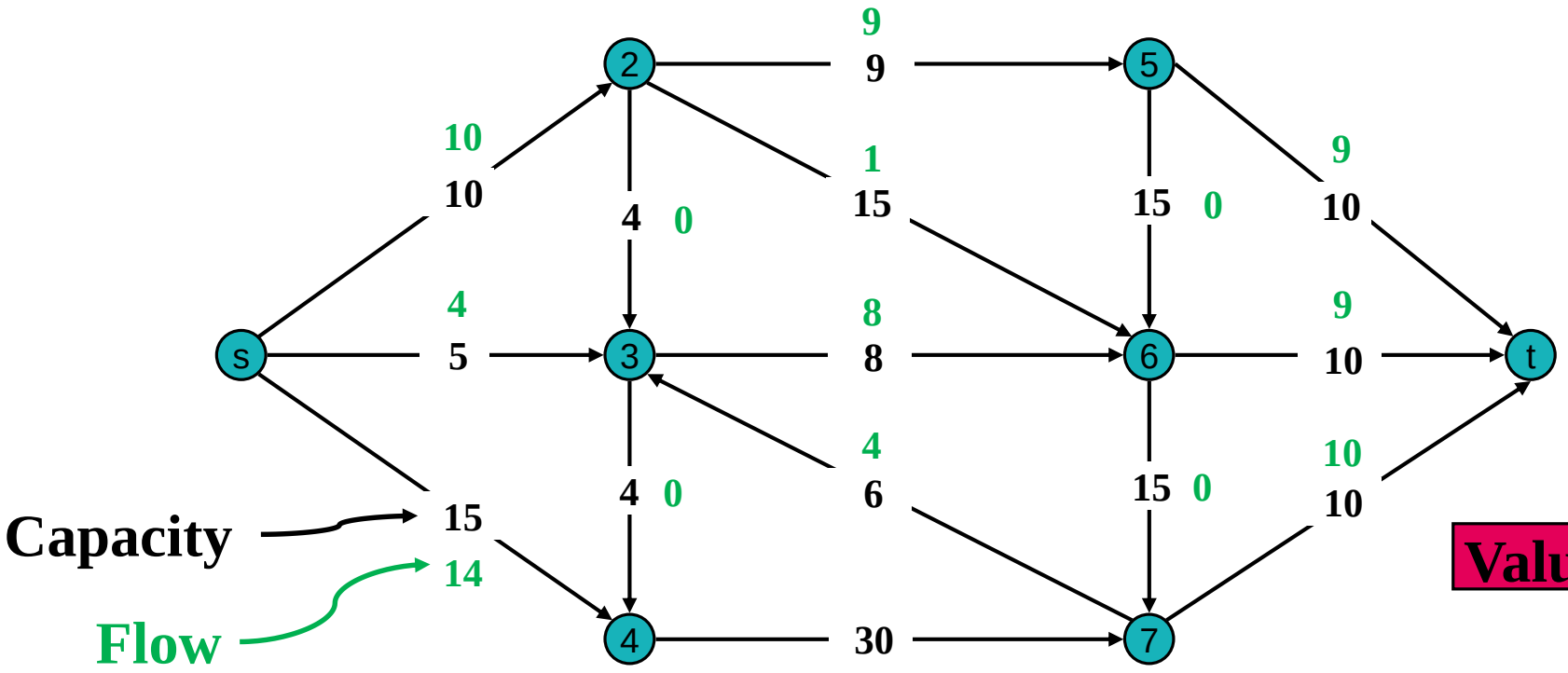
FLOW

- **An s-t flow is a function f: E → ℜ that satisfies:**
  - For each e ∈ E: $0 \leq f(e) \leq u(e)$ (capacity)
  - For each v ∈ V – {s, t}: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ (conservation)
- **Max Flow: s-t flow that maximizes flow out of s** $|f| = \sum_{e \text{ out of } s} f(e)$



FLOW

- **An s-t flow is a function f: E → ℜ that satisfies:**
  - For each e ∈ E: $0 \leq f(e) \leq u(e)$ (capacity)
  - For each v ∈ V – {s, t}: $\sum_{e\ in\ to\ v} f(e) = \sum_{e\ out\ of\ v} f(e)$ (conservation)
- **Max Flow: s-t flow that maximizes flow out of s** $|f| = \sum_{e\ out\ of\ s} f(e)$

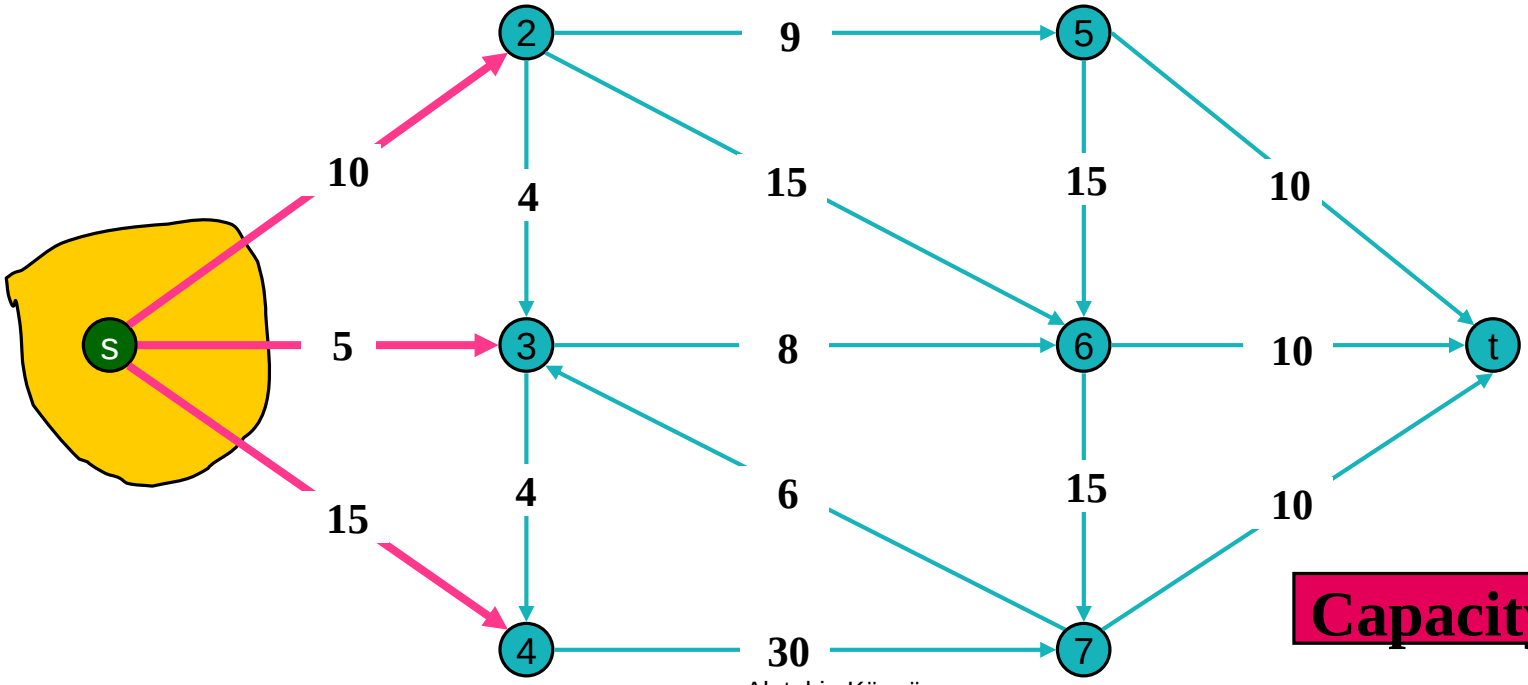


SAMPLE NETWORKS

Network	Nodes (Vertices)	Arcs (Edges)	Flow
Communication	telephone exchanges, computers, satellites	cables, fiber optics, microwave relays	voice, video, packets
Circuits	gates, registers, processors	wires	current
Mechanics	joints	rods, beams, springs	heat, energy
Hydraulic	reservoirs, pumping stations, lakes	pipelines	fluid, oil
Finance	stocks, currency	transactions	money
Transportation	airports, rail yards, street intersections	highways, railbeds, airway routes	freight, vehicles, passengers
Chemistry	elements	bonds	energy

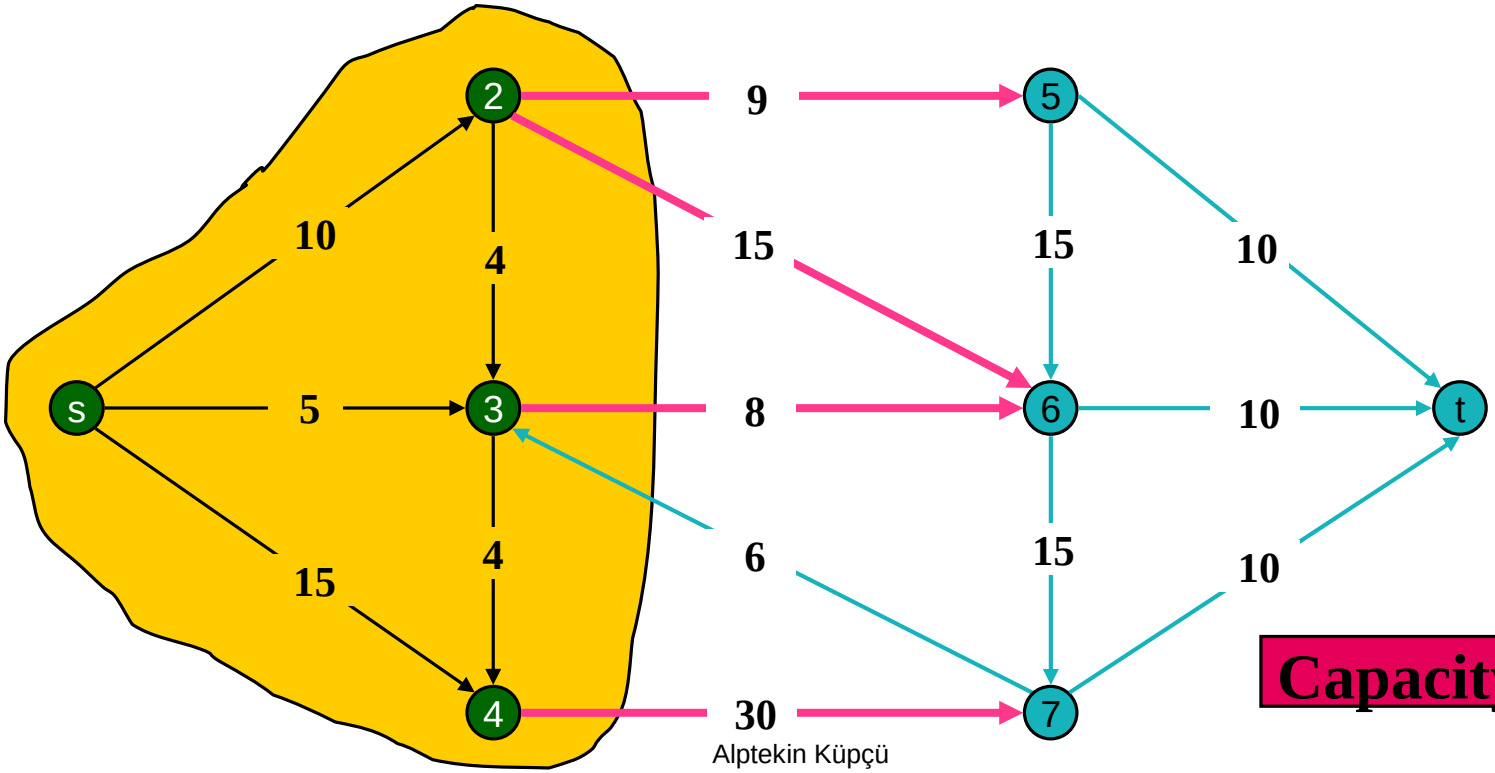
CUT

- **An s-t cut is a node partition (S, T) such that s ∈ S, t ∈ T.**
  - The capacity of an s-t cut (S, T) is: $\sum_{e \text{ out of } S} u(e) = \sum_{\substack{(v,w) \in E \\ v \in S,\, w \in T}} u(v,w)$
- **Min Cut: Find s-t cut of minimum capacity.**



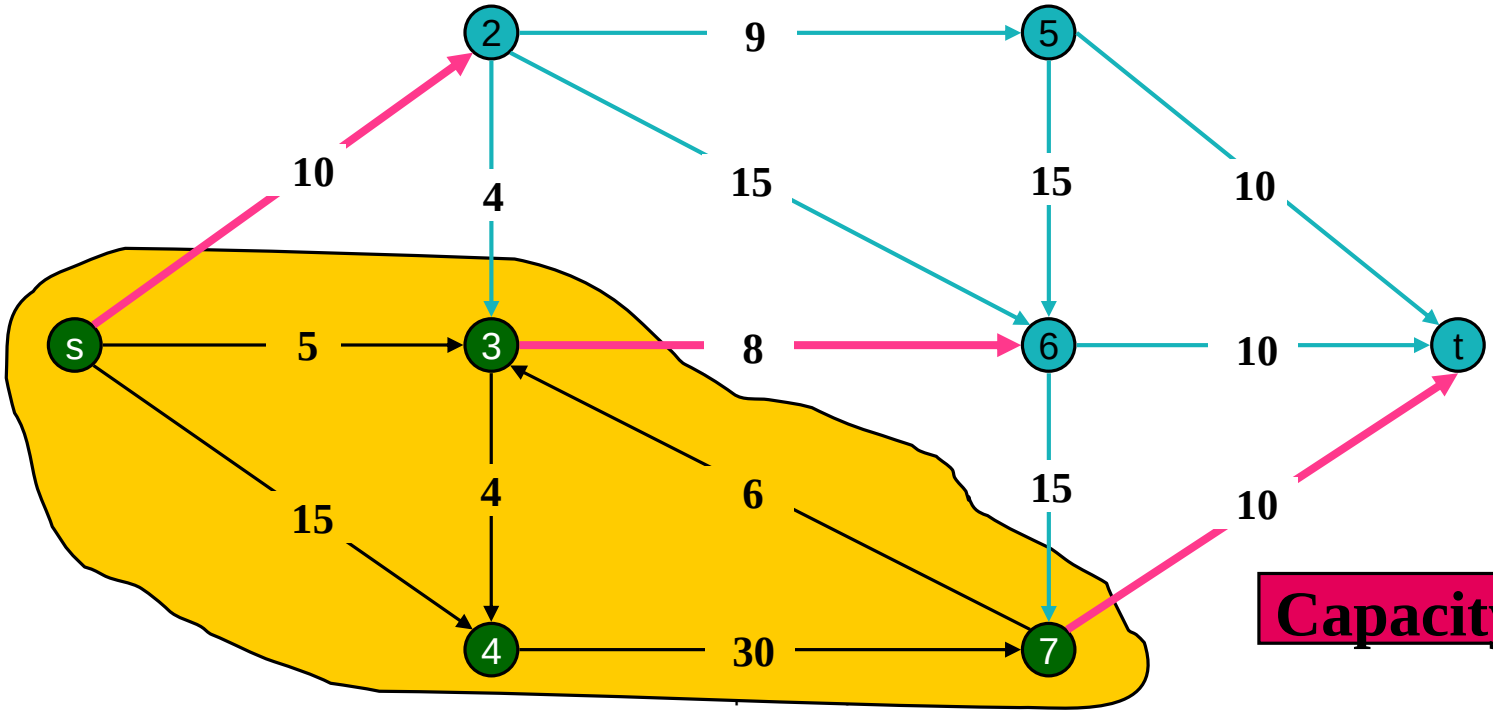
CUT

- **An s-t cut is a node partition (S, T) such that s ∈ S, t ∈ T.**
  - The capacity of an s-t cut (S, T) is: $\sum_{e \text{ out of } S} u(e) = \sum_{\substack{(v,w) \in E \\ v \in S,\, w \in T}} u(v,w)$
- **Min Cut: Find s-t cut of minimum capacity.**



CUT

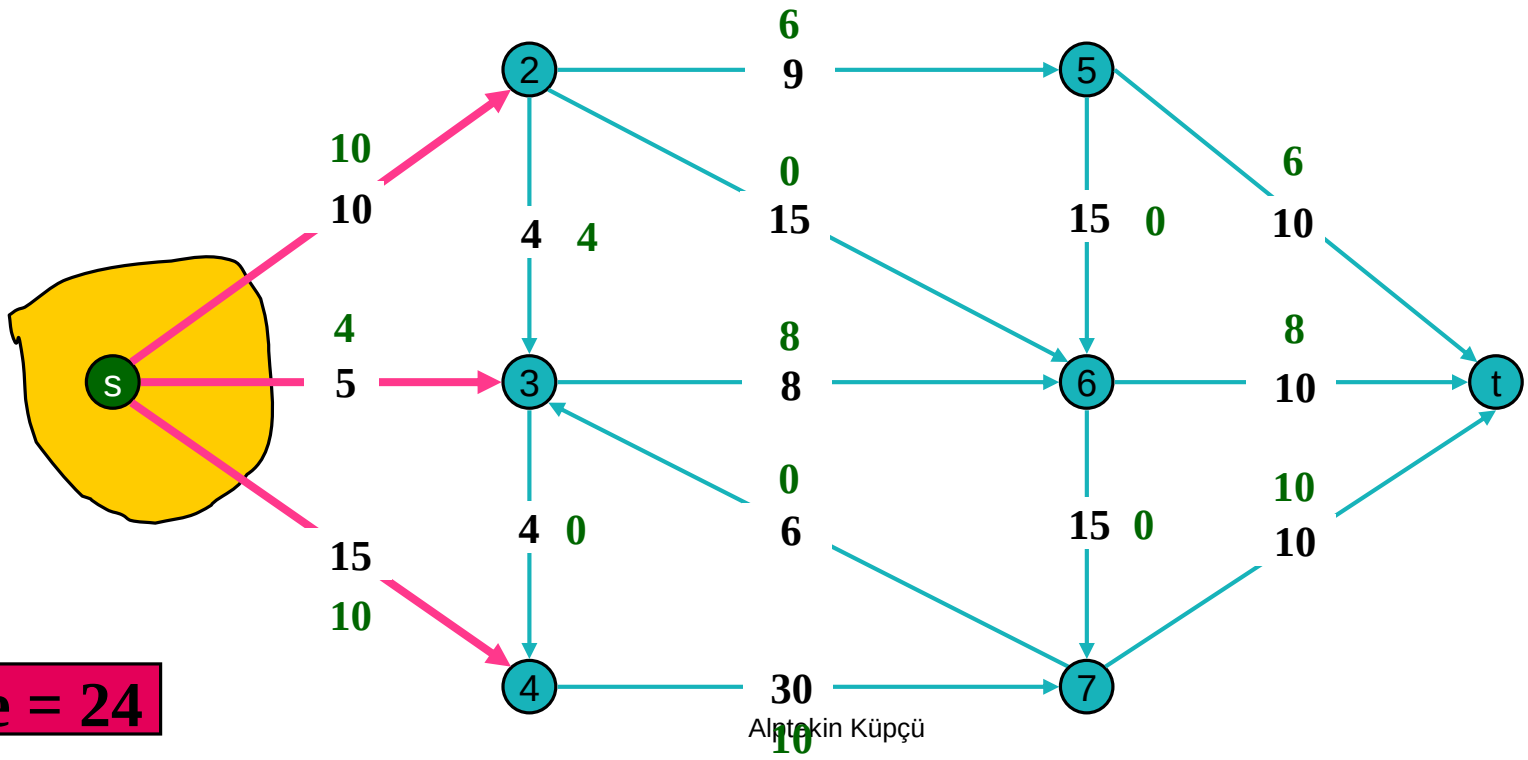
- **An s-t cut is a node partition (S, T) such that s ∈ S, t ∈ T.**
  - The capacity of an s-t cut (S, T) is: $\sum_{e \text{ out of } S} u(e) = \sum_{\substack{(v,w) \in E \\ v \in S,\, w \in T}} u(v,w)$
- **Min Cut: Find s-t cut of minimum capacity.**



FLOWS AND CUTS

- **Lemma 1:** Let ***f*** be a flow, and let **(S, T)** be a cut. Then, the net flow sent across the cut is equal to the amount reaching **t**.

$$\sum_{e \text{ out of } S} f(e) - \sum_{e \text{ in to } S} f(e) = \sum_{e \text{ out of } s} f(e) = |f|$$

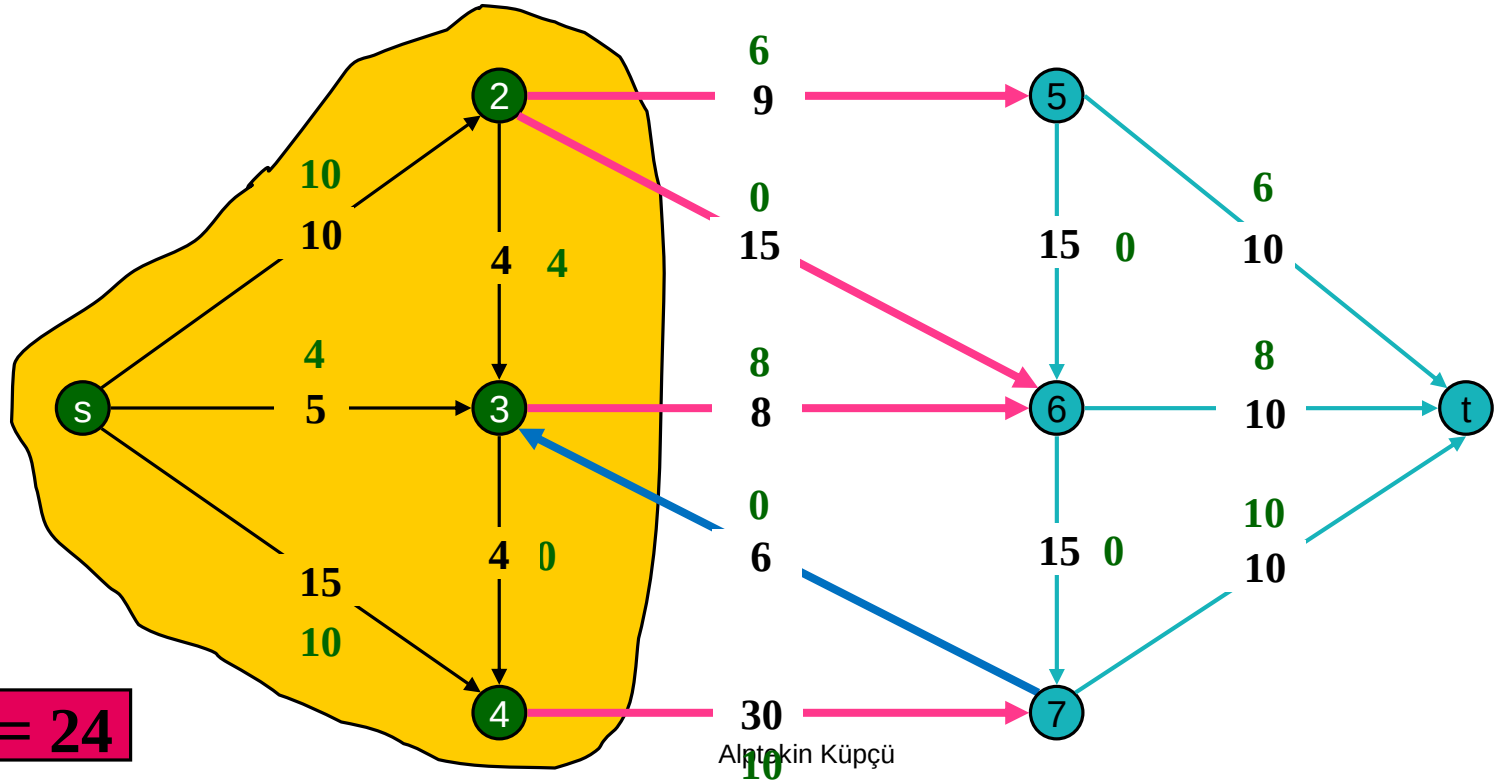


Value = 24

FLOWS AND CUTS

- **Lemma 1:** Let ***f*** be a flow, and let **(S, T)** be a cut. Then, the net flow sent across the cut is equal to the amount reaching **t**.

$$\sum_{e \text{ out of } S} f(e) - \sum_{e \text{ in to } S} f(e) = \sum_{e \text{ out of } s} f(e) = |f|$$

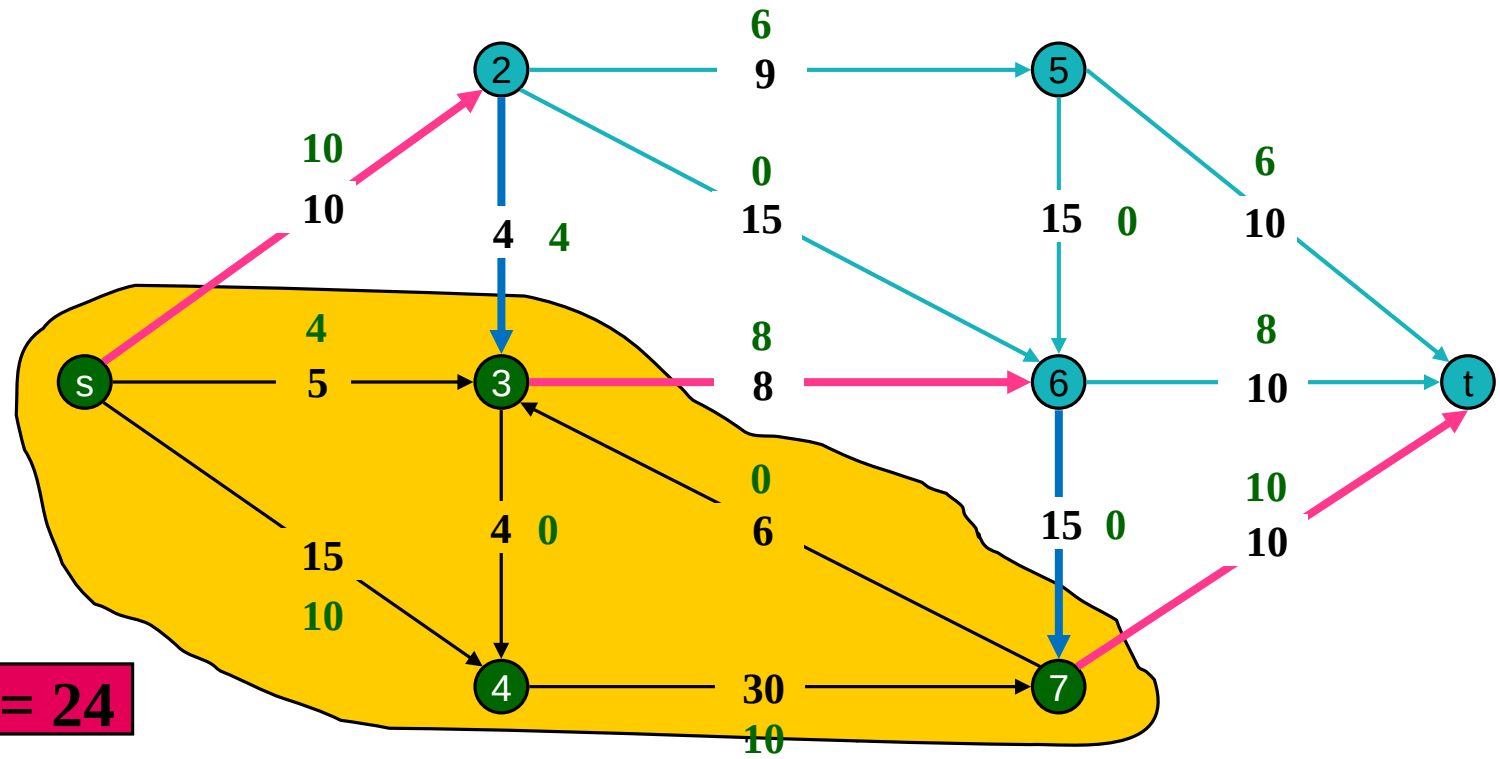


Value = 24

FLOWS AND CUTS

- **Lemma 1:** Let ***f*** be a flow, and let **(S, T)** be a cut. Then, the net flow sent across the cut is equal to the amount reaching **t**.

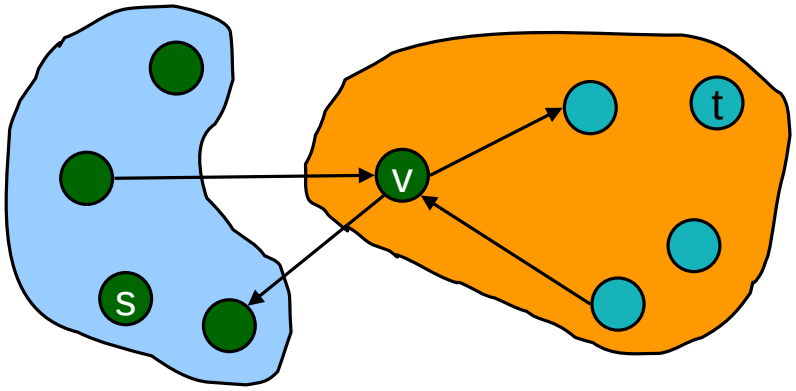
$$\sum_{e \text{ out of } S} f(e) - \sum_{e \text{ in to } S} f(e) = \sum_{e \text{ out of } s} f(e) = |f|$$



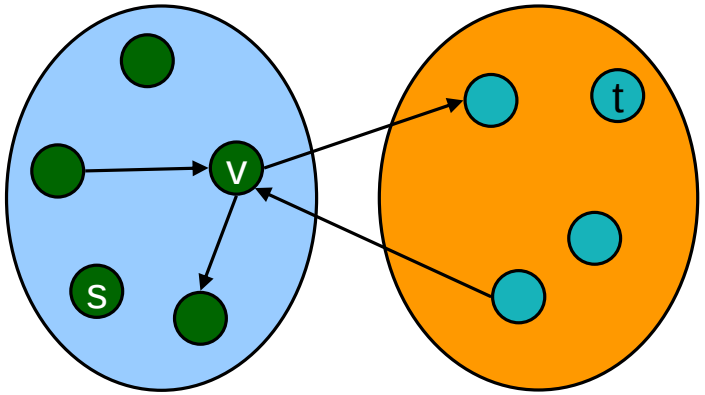
Value = 24

FLOWS AND CUTS

- **Let *f* be a flow, and let (S, T) be a cut. Then,** $\sum_{e \text{ out of } S} f(e) - \sum_{e \text{ in to } S} f(e) = |f|$
- **Proof: by induction on |S|**
  - Base case: S = { s }, holds by definition (no in-flow).
  - Inductive hypothesis: Assume true for all S' with |S'| < k.
    - Consider cut (S, T) with |S| = k
    - Then S = S' ∪ { v } for some S' and v (v ≠ s and v ≠ t)
      - cap(S', T') = | *f* | (by inductive hypothesis)
    - adding v to S' increase cut capacity by $\sum_{e \text{ out of } v} f(e) - \sum_{e \text{ in to } v} f(e) = 0$



S' **Before**

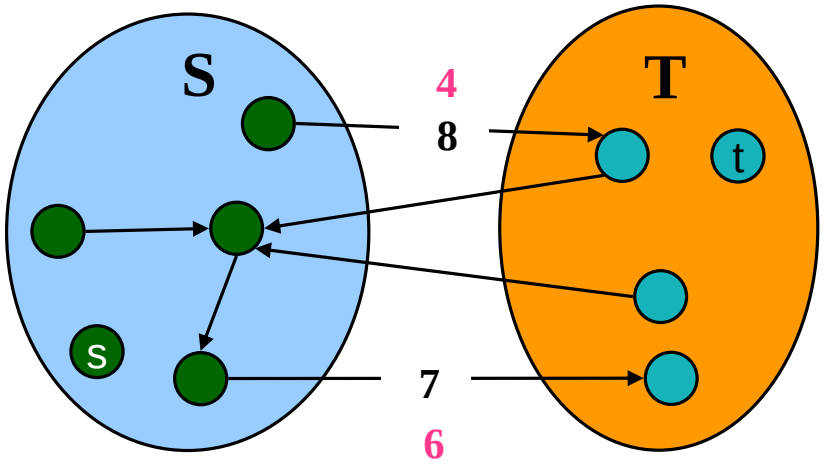


S **After**

FLOWS AND CUTS

- **Lemma 2:** Let ***f*** be a flow, and let (S, T) be a cut. Then, $|f| \leq cap(S,T)$
- **Proof:**

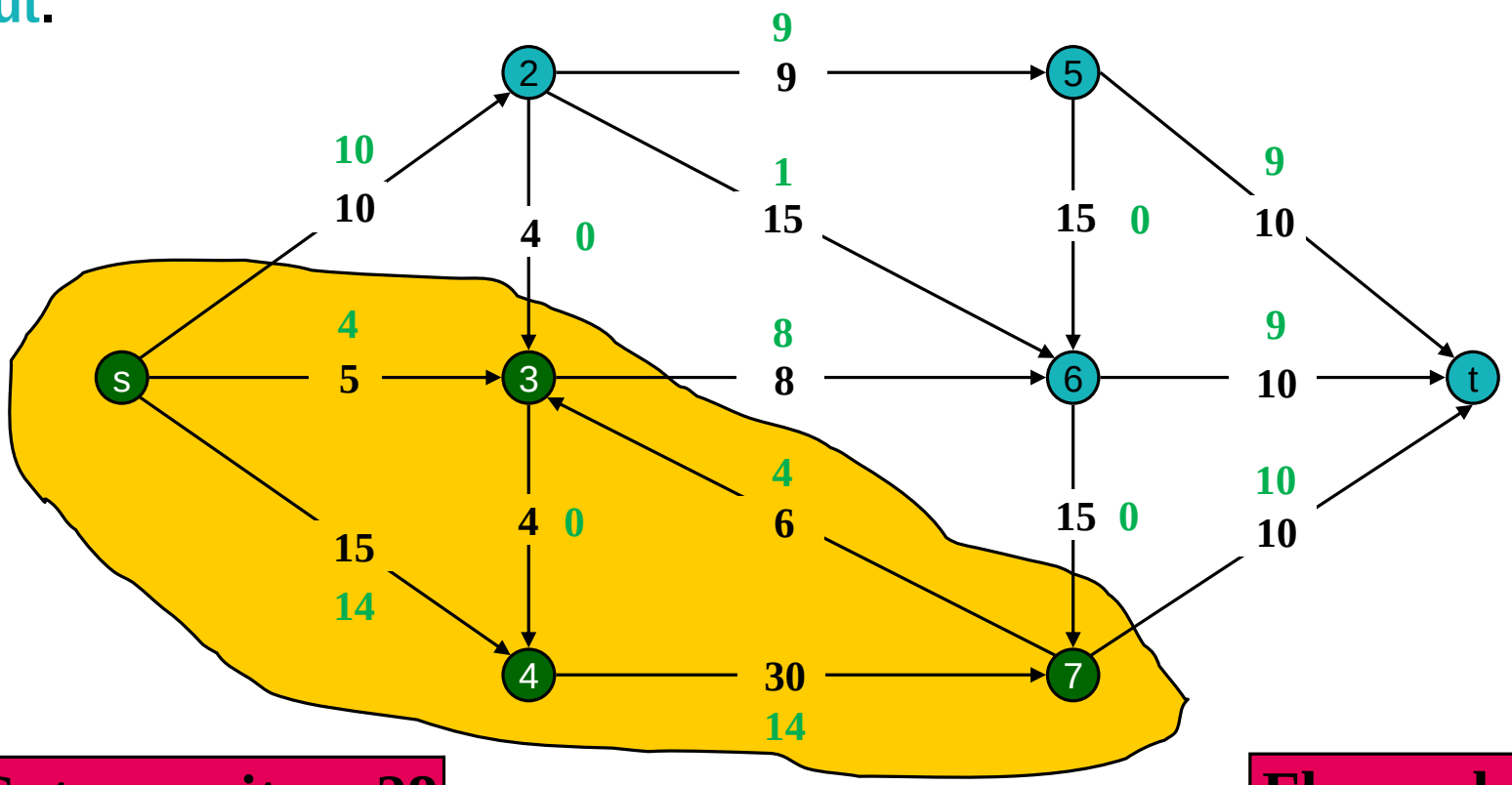
$$|f| = \sum_{e \text{ out of } S} f(e) - \sum_{e \text{ in to } S} f(e)$$
$$\leq \sum_{e \text{ out of } S} f(e)$$
$$\leq \sum_{e \text{ out of } S} u(e)$$
$$= cap(S, T)$$



- **Corollary:** Let ***f*** be a flow, and let (S, T) be a cut. If $|f| = cap(S, T)$, then ***f*** is a max flow and (S, T) is a min cut.

MAX-FLOW MIN-CUT THEOREM

- **MAX-FLOW MIN-CUT THEOREM (Ford-Fulkerson, 1956): In any network, the value of the max flow is equal to the value of the min cut.**

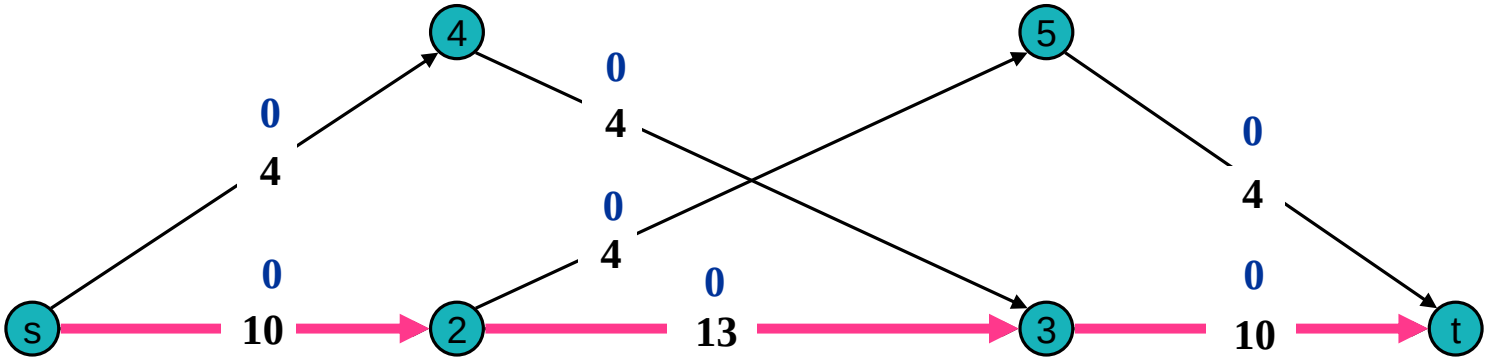


Cut capacity = 28

Flow value = 28

TOWARDS AN ALGORITHM

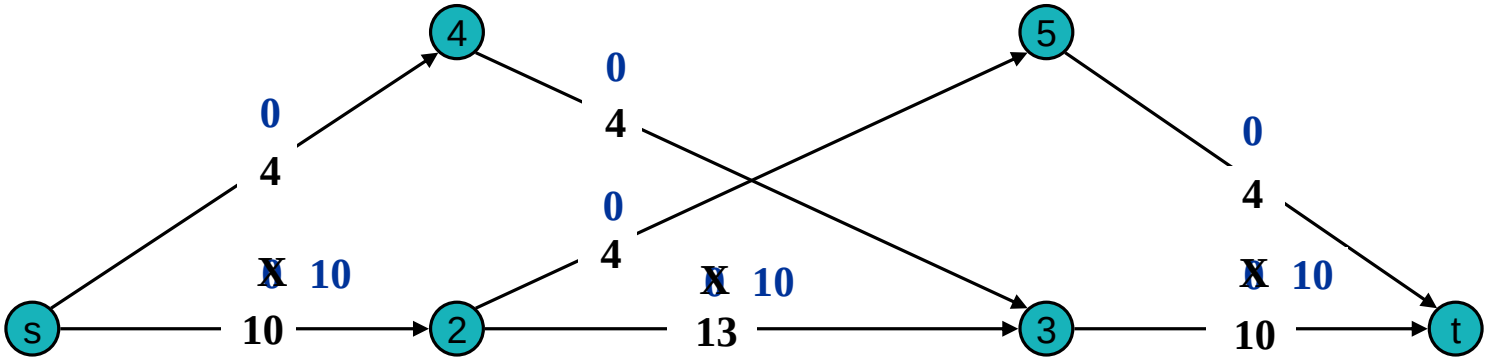
- **Find an s-t path where each edge has u(e) > f(e) and "augment" flow along the path.**



Flow value = 0

TOWARDS AN ALGORITHM

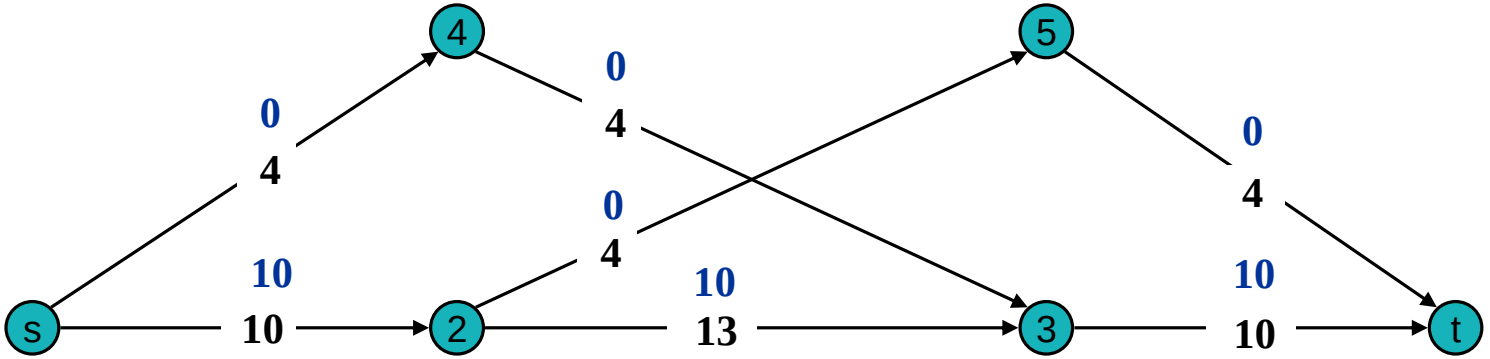
- **Find an s-t path where each edge has u(e) > f(e) and "augment" flow along the path.**
  - Repeat until you get stuck.



Flow value = 10

TOWARDS AN ALGORITHM

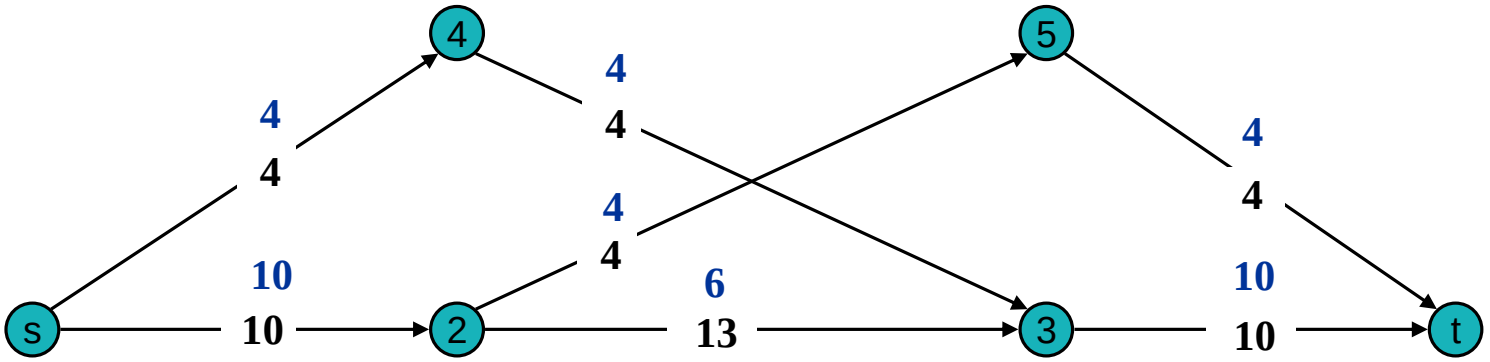
- **Find an s-t path where each edge has u(e) > f(e) and "augment" flow along the path.**
  - Stuck.



Flow value = 10

FAILED APPROACH

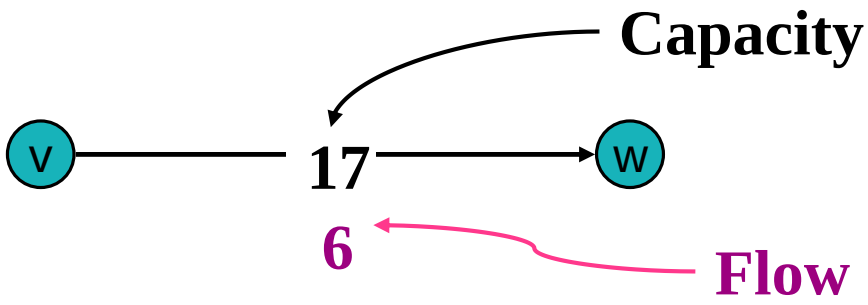
- **Actual max-flow value is 14.**



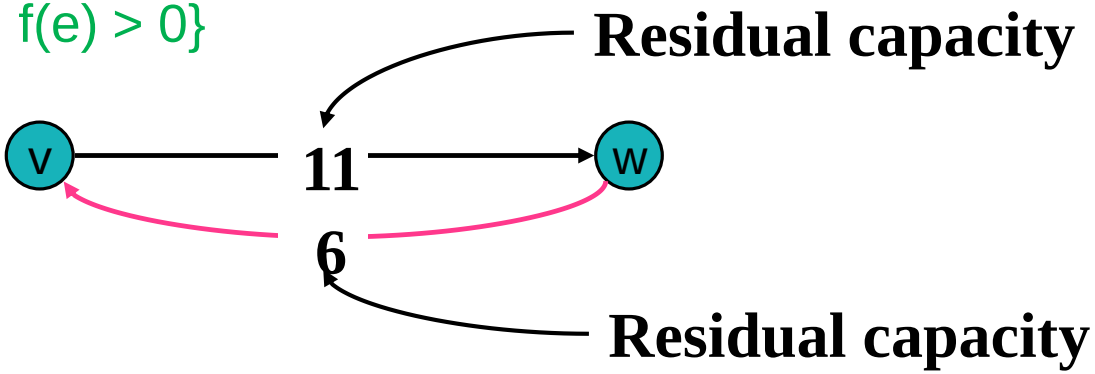
Flow value = 14

RESIDUAL ARCS

- **Original graph G = (V, E)**



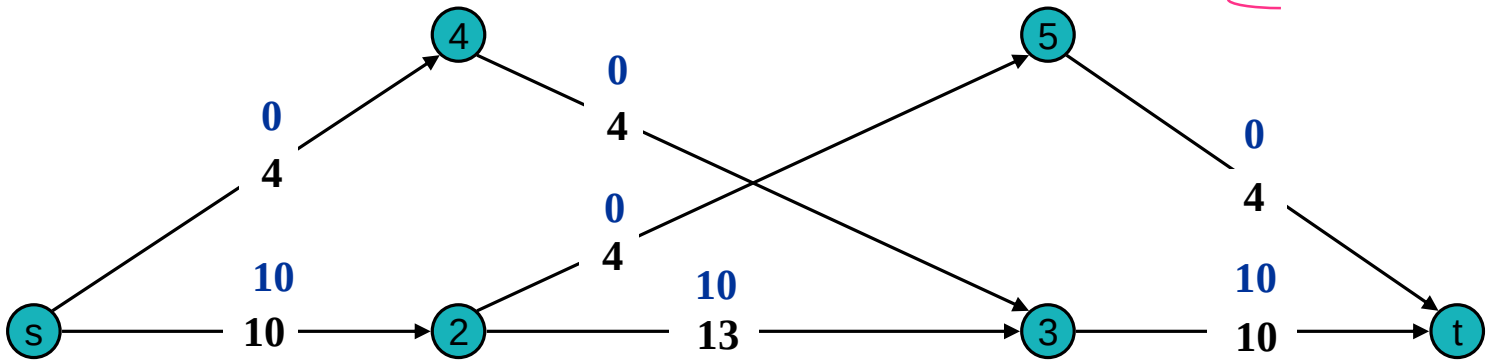
- **Residual graph: $G_f = (V, E_f)$**
  - Residual arcs e = (v, w) and e^R = (w, v)
    - "Undo" flow sent
  - $E_f = \{e: f(e) < u(e)\} \cup \{e^R: f(e) > 0\}$



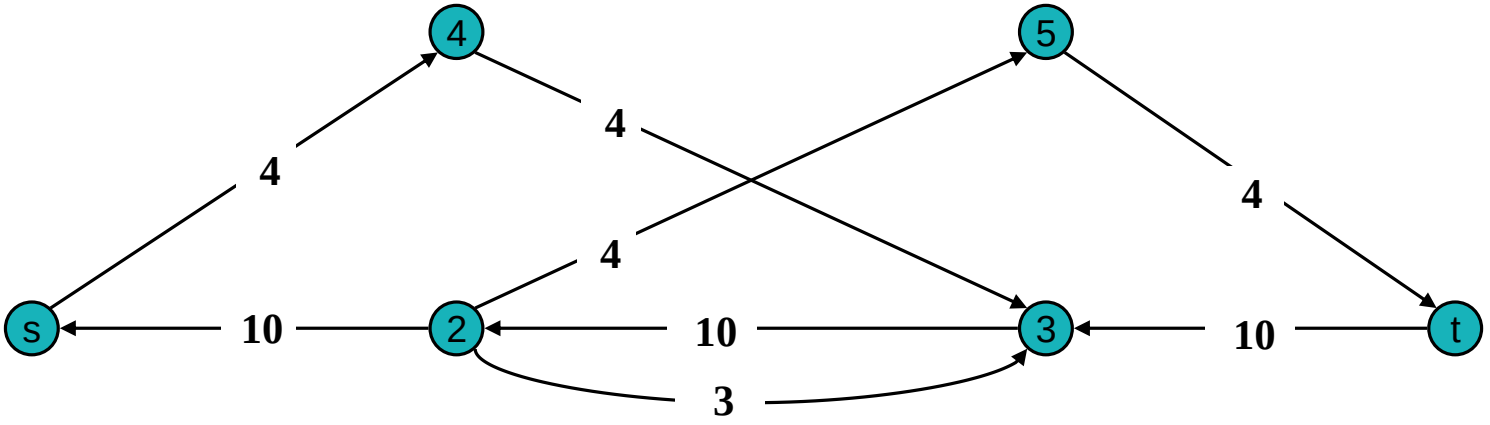
RESIDUAL GRAPH AND AUGMENTING PATHS

$$u_f(e) = \begin{cases} u(e) - f(e) & \text{if } e \in E \\ f(e) & \text{if } e^R \in E \end{cases}$$

G

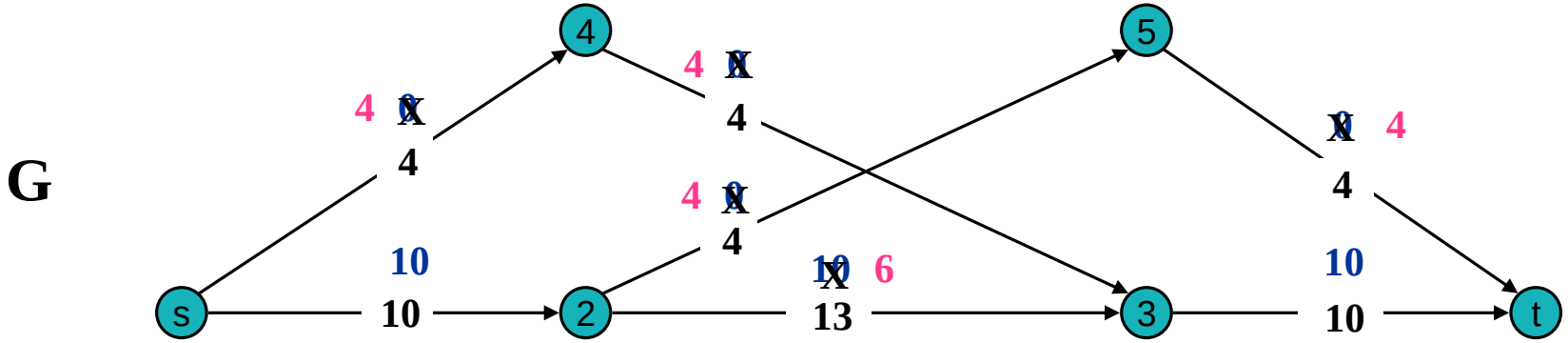


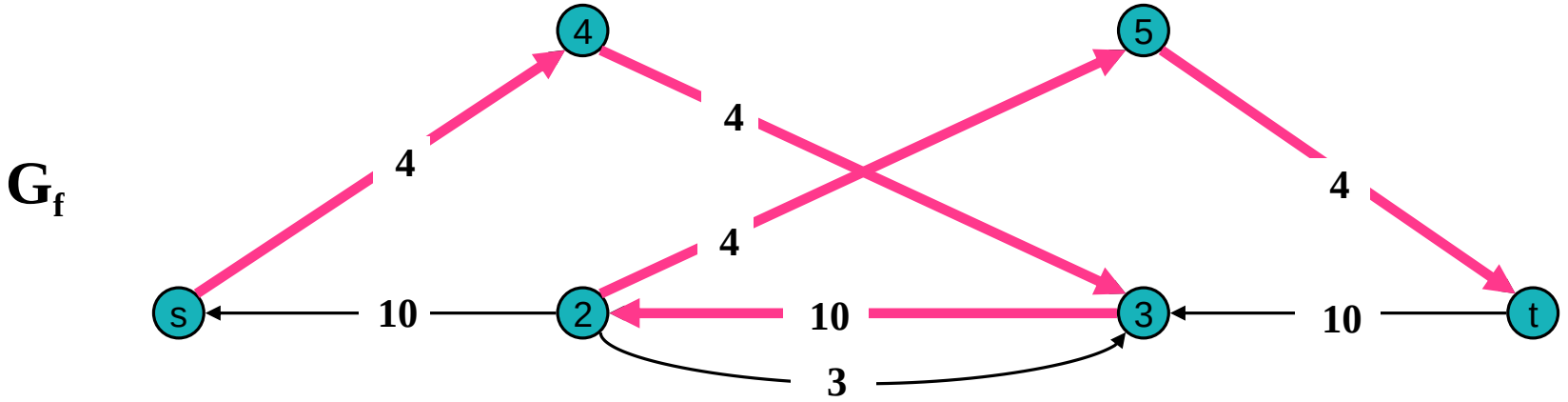
G_f



AUGMENTING PATH

- **Augmenting path = path in residual graph.**

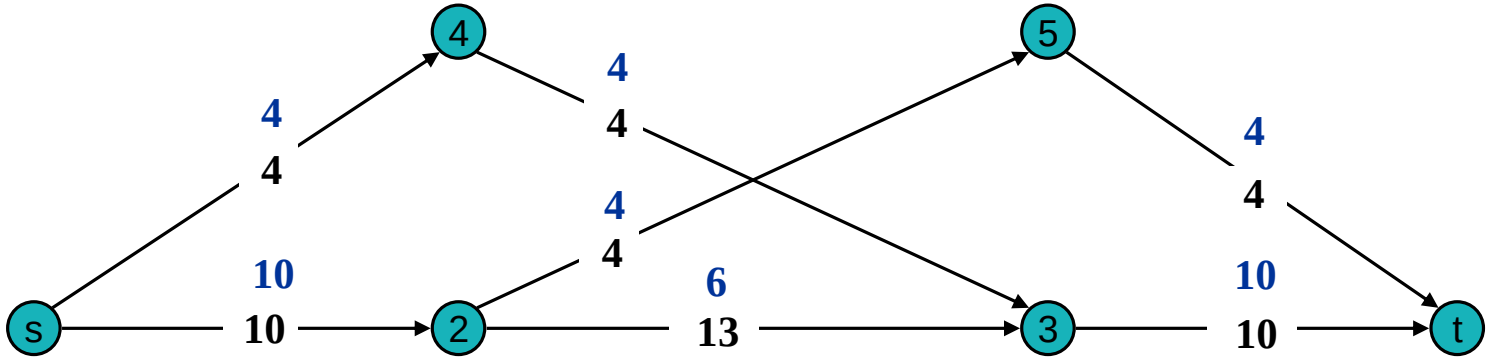




AUGMENTING PATH

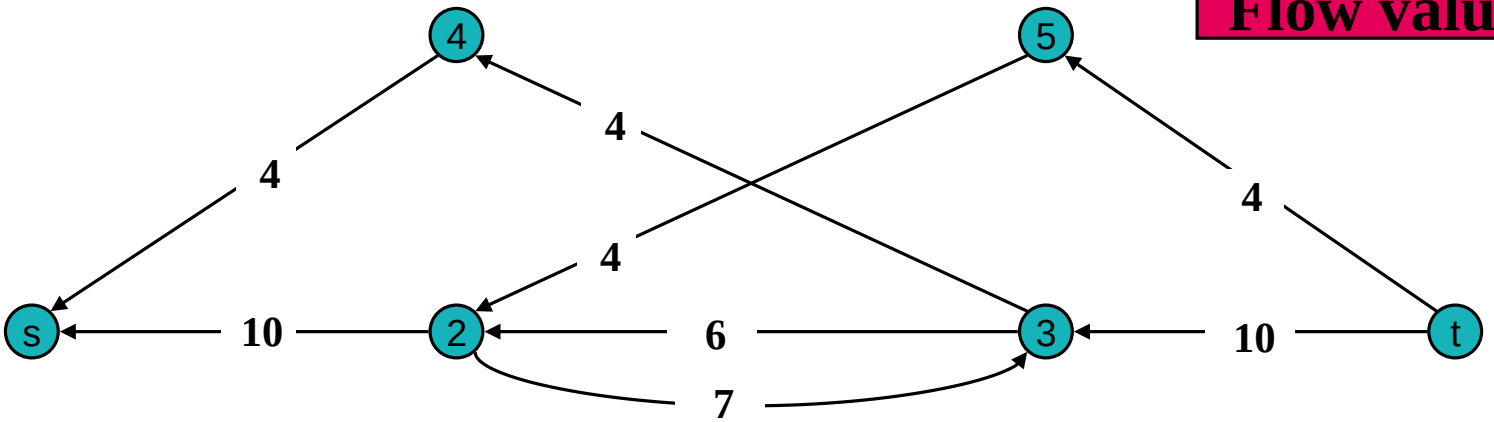
- **Max flow = no remaining augmenting paths from s to t.**

G



Flow value = 14

G_f



MAX-FLOW MIN-CUT THEOREM

- **Augmenting path theorem (Ford-Fulkerson, 1956): A flow f is a max flow if and only if there are no augmenting paths.**

- **MAX-FLOW MIN-CUT THEOREM (Ford-Fulkerson, 1956): The value of the max flow is equal to the value of the min cut.**

- **Proof: We prove both simultaneously by showing the following are equivalent:**

  (i) f is a max flow.

  (ii) There is no augmenting path relative to f.

  (iii) There exists a cut (S, T) such that |f| = cap(S, T).

PROOF OF MAX-FLOW MIN-CUT THEOREM

(i) f is a max flow.
(ii) There is no augmenting path relative to f.
(iii) There exists a cut (S, T) such that $|f| = cap(S, T)$.

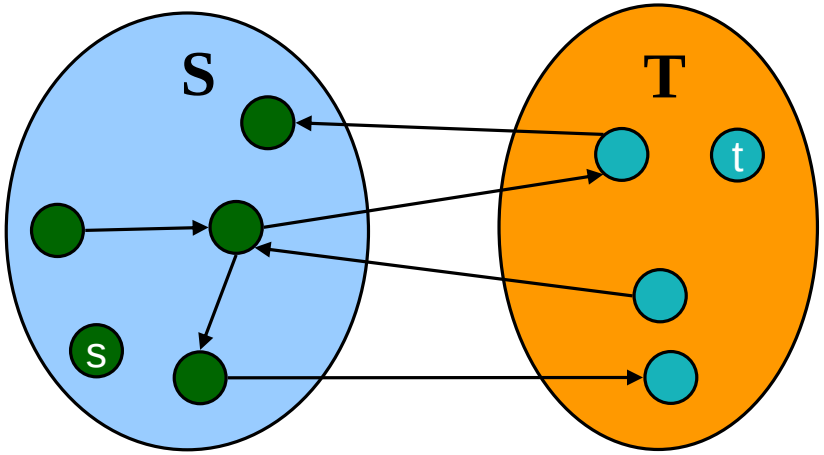
- **(i) ⇒ (ii)**
  - Proof by contraposition.
  - Let f be a max flow. If there exists an augmenting path, then we can improve f by sending flow along path.
- **(iii) ⇒ (i)**
  - This was the Corollary to Lemma 2.
- **(ii) ⇒ (iii)**
  - Next slide.

PROOF OF MAX-FLOW MIN-CUT THEOREM

(ii) There is no augmenting path relative to f.

(iii) There exists a cut (S, T) such that |f| = cap(S, T).

- **(ii) ⇒ (iii)**
  - Let f be a flow with no augmenting path.
  - Let S be set of vertices reachable from s in the residual graph.
    - clearly $s \in S$, and $t \notin S$ by definition of f (no augmenting path)



Original Network

$$|f| = \sum_{e \text{ out of } S} f(e) - \sum_{e \text{ in to } S} f(e)$$

$$= \sum_{e \text{ out of } S} u(e)$$

$$= \textbf{cap(S,T)}$$

FORD-FULKERSON ALGORITHM

FordFulkerson (V, E, s, t)

```
FOREACH e ∈ E
    f(e) ← 0
G_f ← residual graph
WHILE (there exists augmenting path P)
    f ← augment(f, P)
    update G_f
RETURN f
```

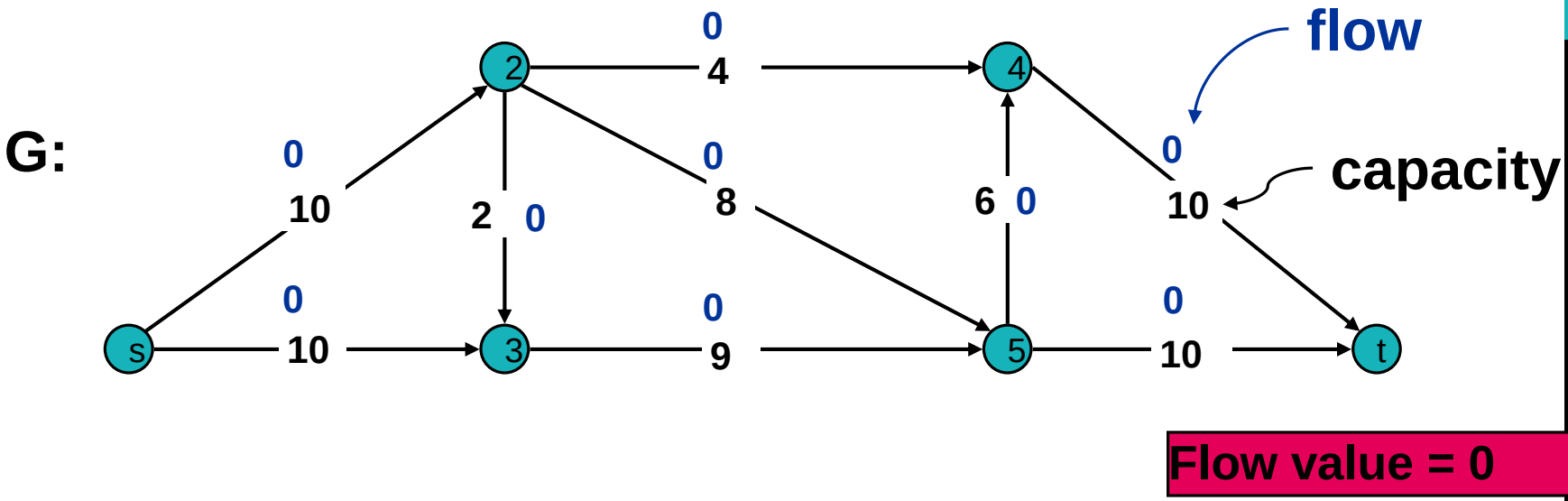
Augment (f, P)

```
b ← min{u(e): e ∈ P}
FOREACH e ∈ P
    IF (e ∈ E) // forward arc
        f(e) ← f(e) + b
    ELSE       // backwards arc
        f(e^R) ← f(e) - b
RETURN f
```

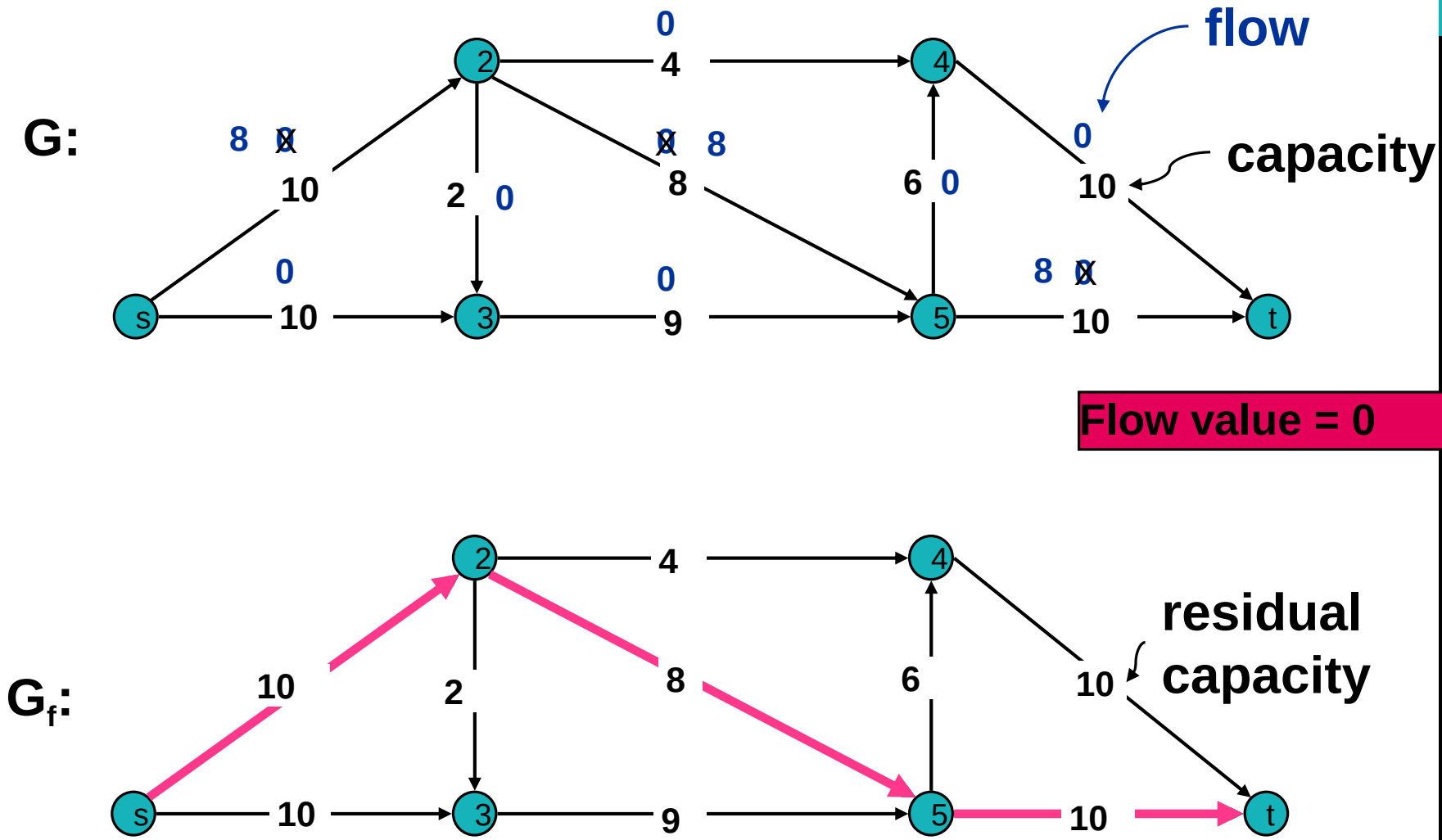
RUNNING TIME

- **Assumption: All capacities are integers between 0 and U (max capacity).**
- **Integrality theorem: If all edge capacities are integers, then there exists a max flow f for which every flow value f(e) is an integer.**
  - Algorithm may not terminate on instances with non-integer capacities (e.g., irrational capacities), or the flow value may not converge to correct answer.
- **Invariant: Every flow value f(e) and every residual capacity $u_f(e)$ remains an integer throughout the algorithm.**
- **Theorem: The algorithm terminates in at most |f*| iterations, where |f*| denotes the max flow, since at each iteration flow increases by at least 1.**
- **Each iteration takes at most O(m) time. (m = #edges)**
- **Corollary: Algorithm runs in O(|f*| m) time.**

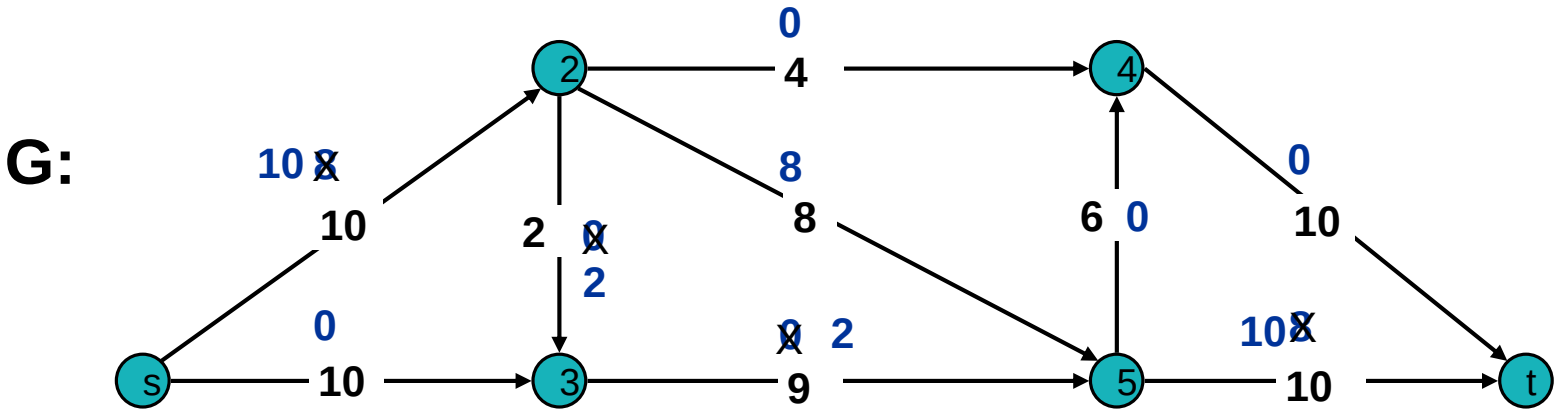
AUGMENTING PATH ALGORITHM



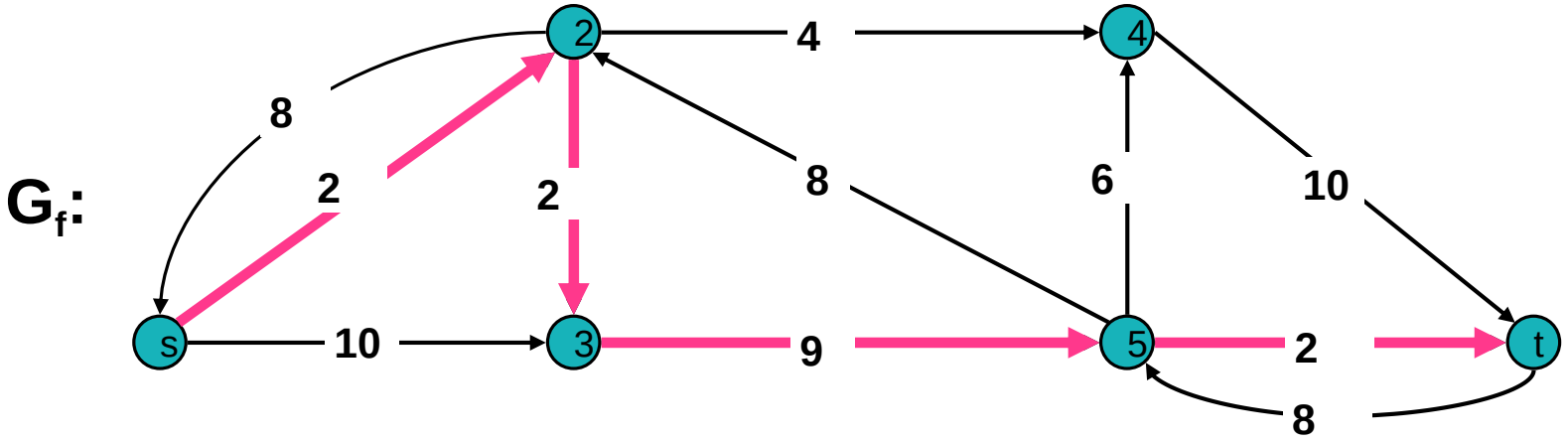
AUGMENTING PATH ALGORITHM



AUGMENTING PATH ALGORITHM

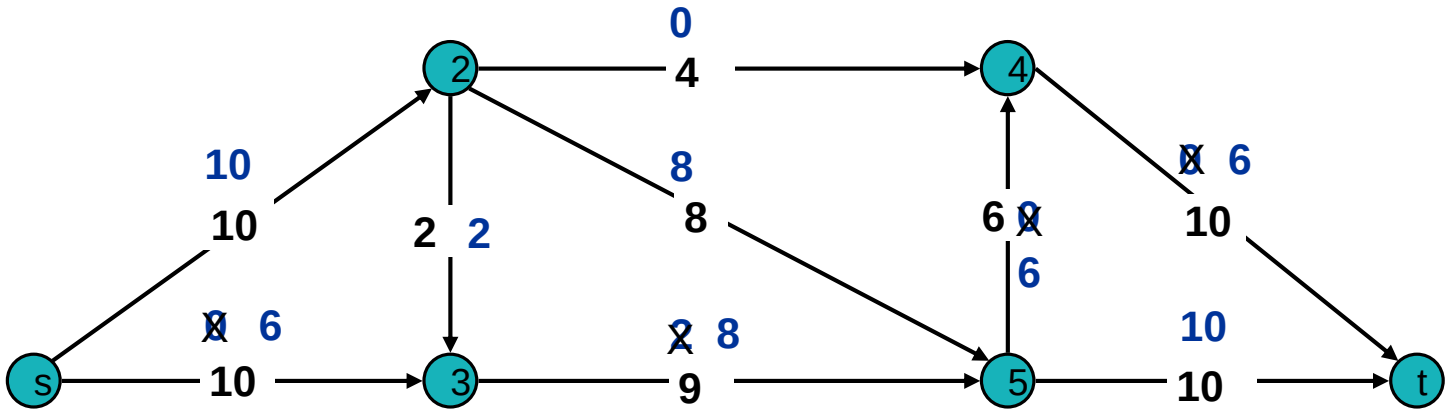


Flow value = 8



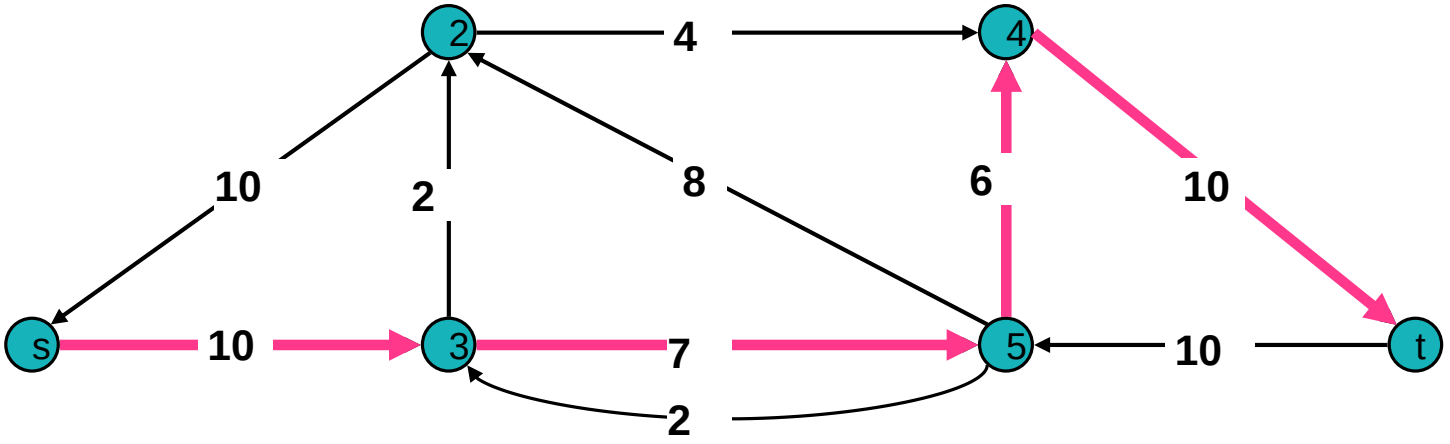
AUGMENTING PATH ALGORITHM

G:

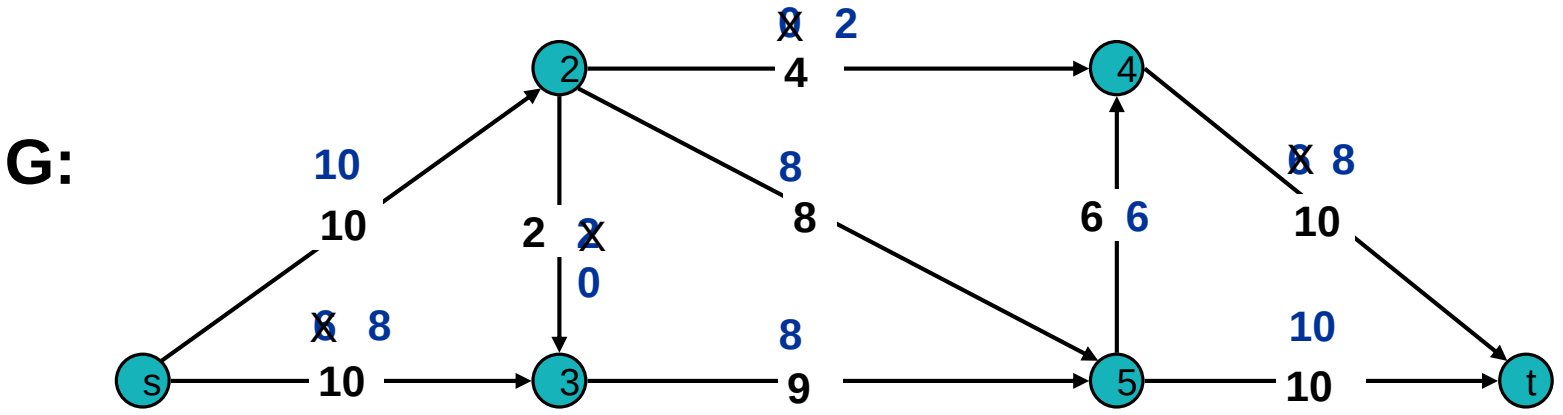


Flow value = 10

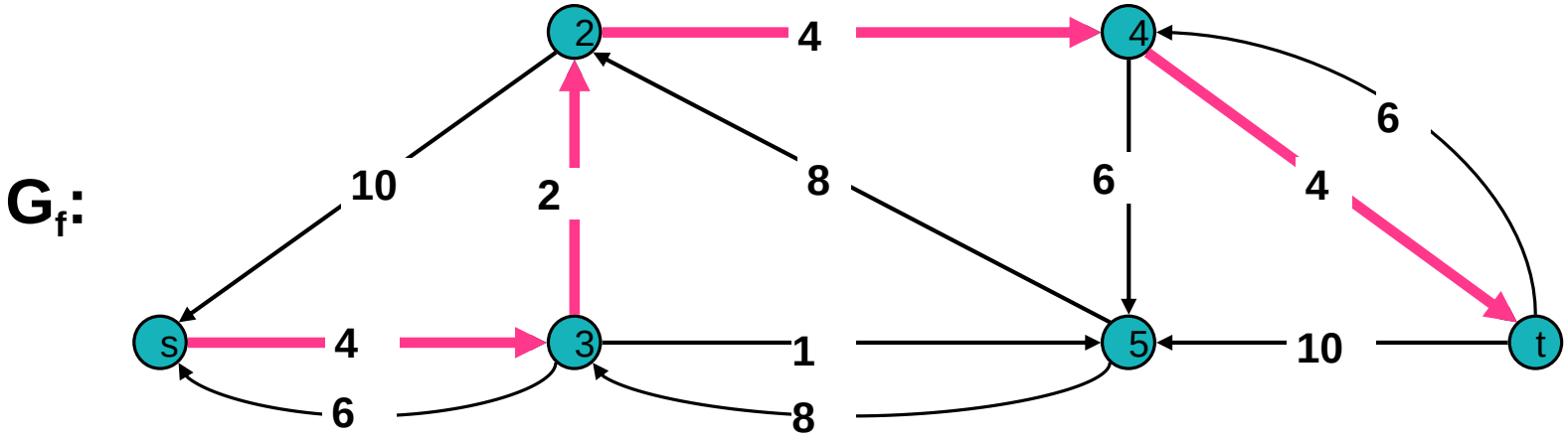
G_f:



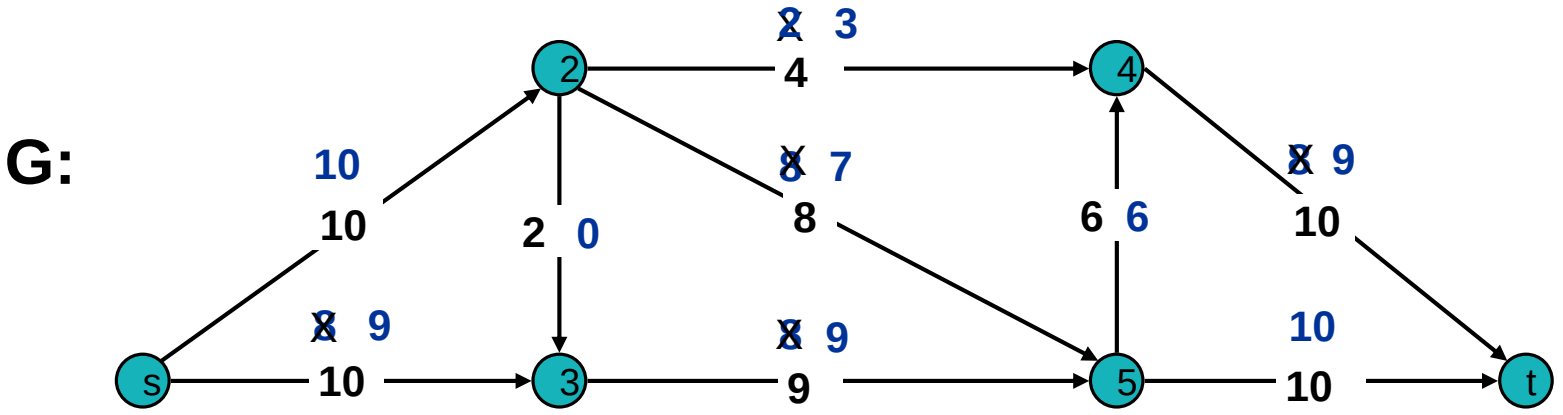
AUGMENTING PATH ALGORITHM



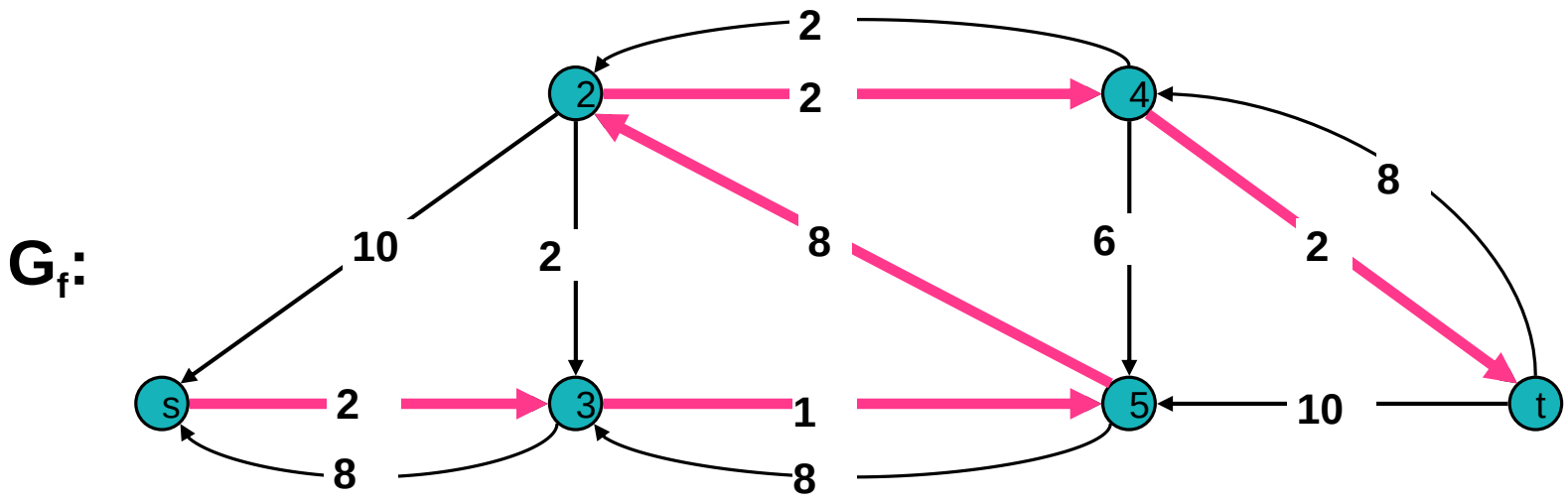
Flow value = 16



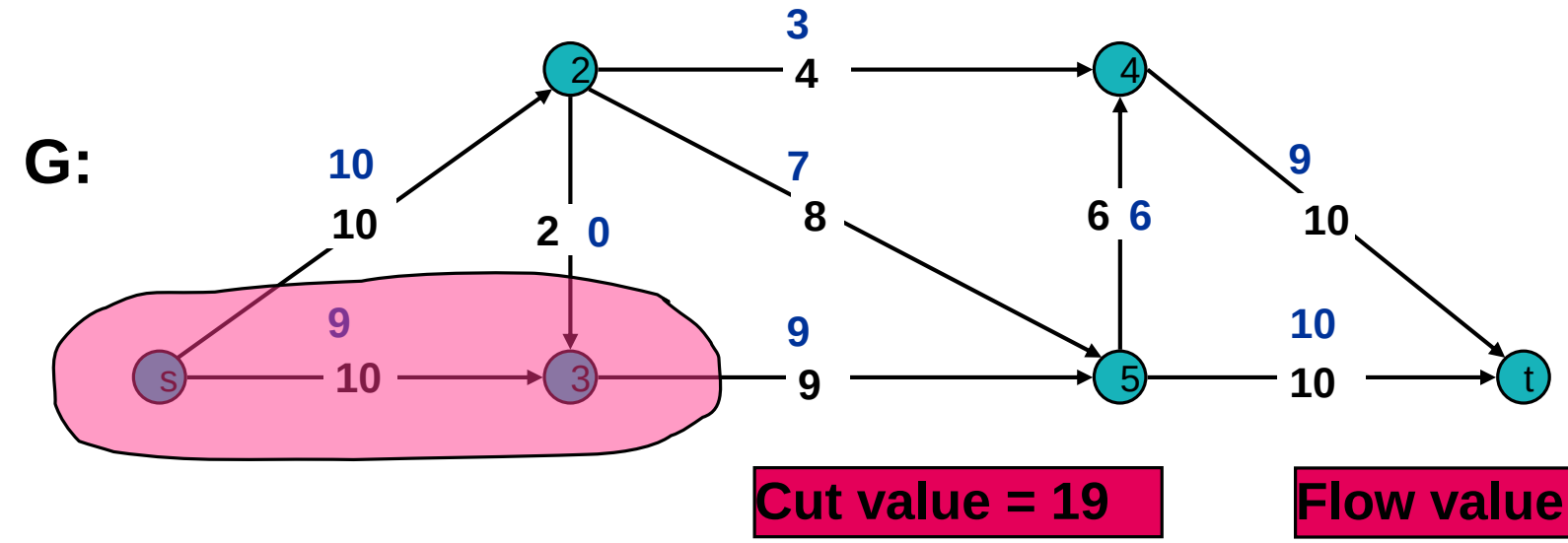
AUGMENTING PATH ALGORITHM

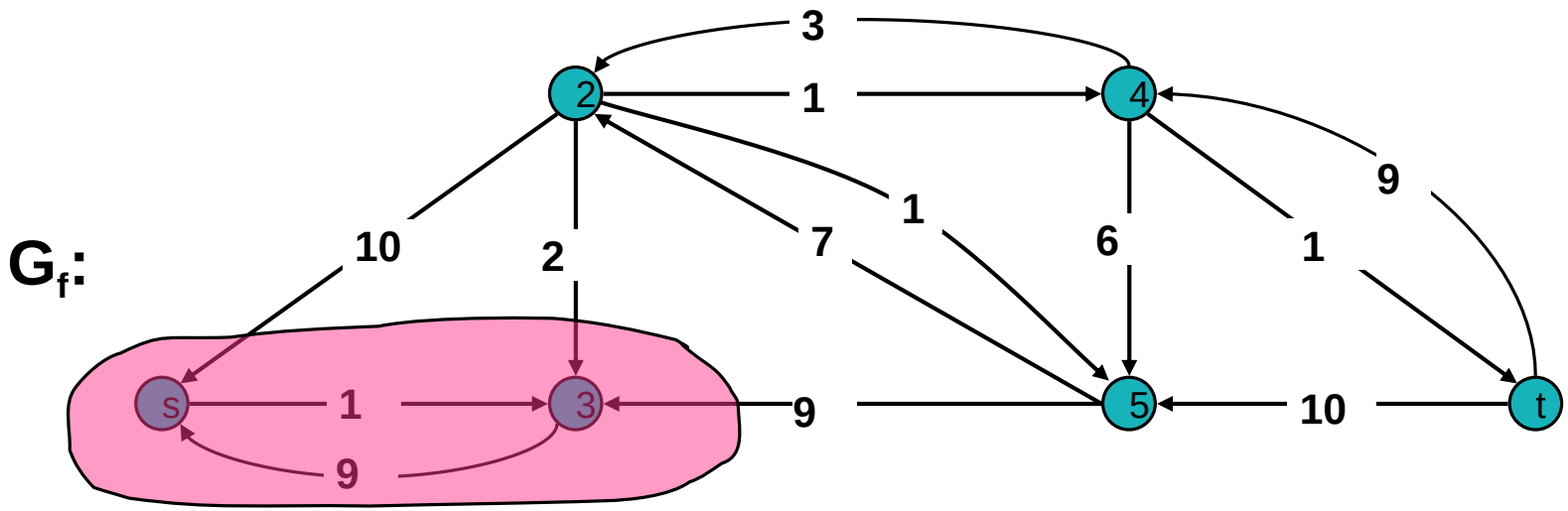


Flow value = 18



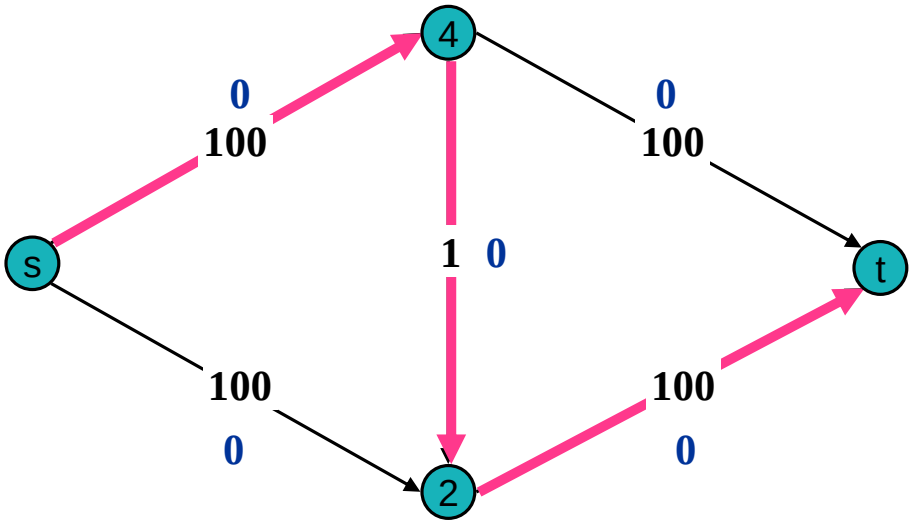
AUGMENTING PATH ALGORITHM





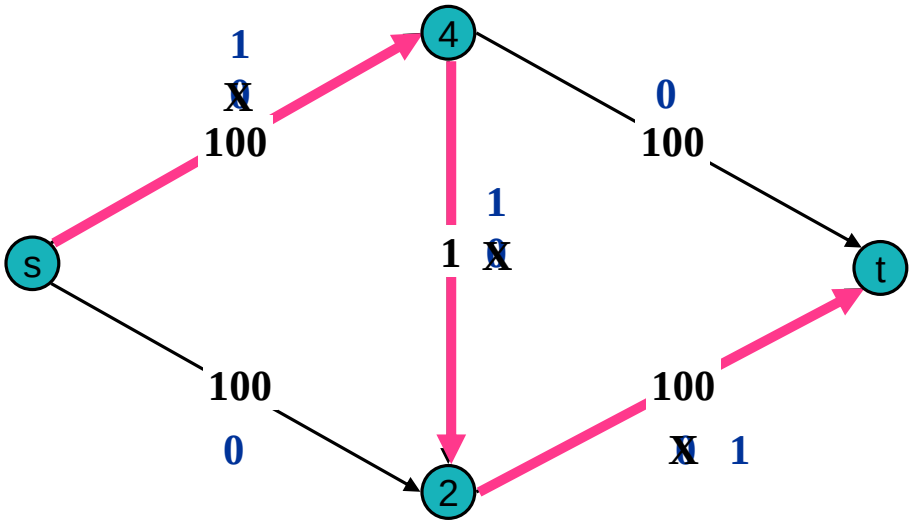
CHOOSING GOOD AUGMENTING PATHS

- **Selecting augmenting paths:**



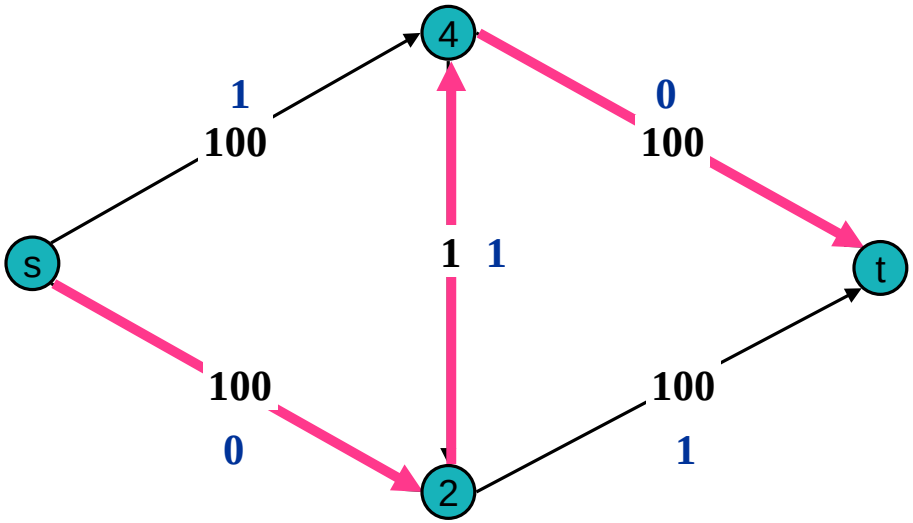
CHOOSING GOOD AUGMENTING PATHS

- **Selecting augmenting paths:**



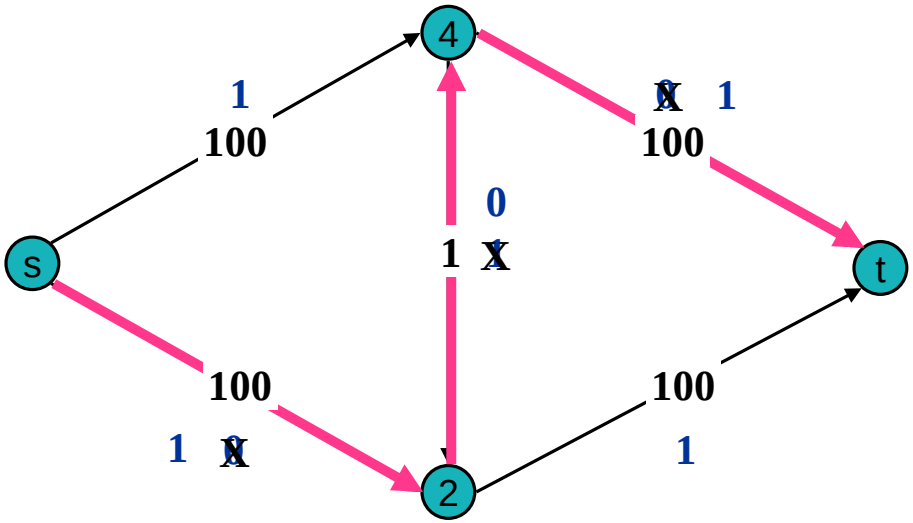
CHOOSING GOOD AUGMENTING PATHS

- **Selecting augmenting paths:**



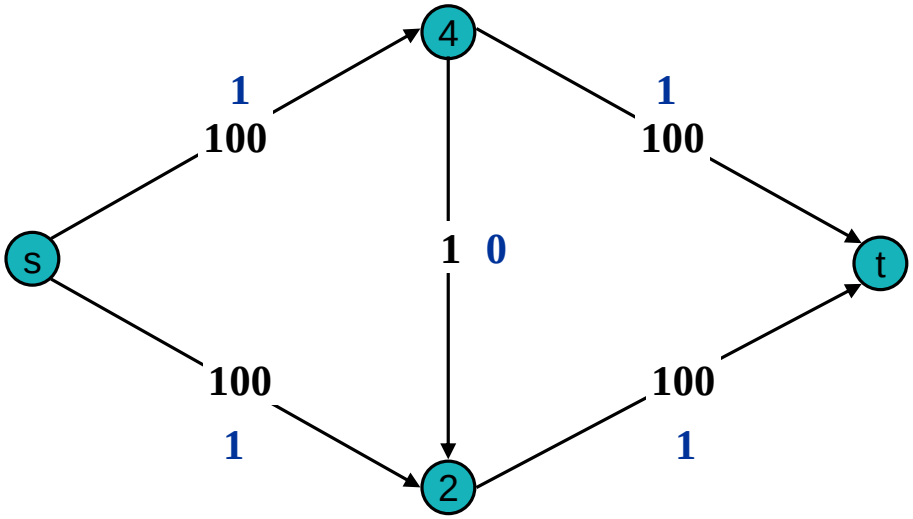
CHOOSING GOOD AUGMENTING PATHS

- **Selecting augmenting paths:**



CHOOSING GOOD AUGMENTING PATHS

- **Selecting augmenting paths:**



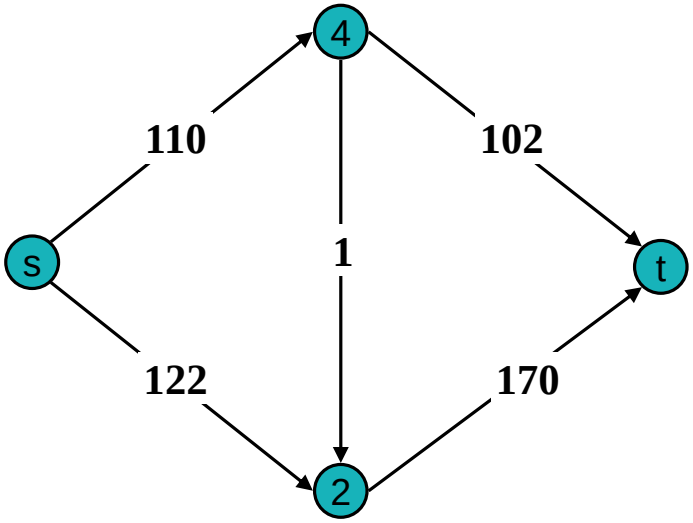
200 iterations possible !!!

CHOOSING GOOD AUGMENTING PATHS

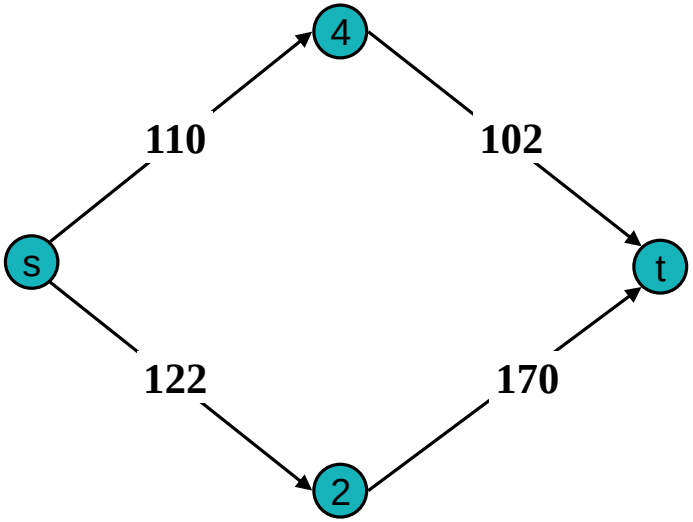
- **Carefully select augmenting paths.**
  - Some choices lead to exponential algorithms. ***(Exponential? How?)***
  - Clever choices lead to polynomial algorithms.
- **Goal: Choose augmenting paths so that**
  - Can find augmenting paths efficiently
  - Few iterations
- **Edmonds-Karp (1972): choose augmenting path with**
  - Max bottleneck capacity (fat path)
  - Sufficiently large capacity (capacity-scaling)
  - Fewest number of edges (shortest path)

CAPACITY SCALING

- **Intuition: Choose the path with highest bottleneck capacity.**
  - Increases flow by max possible amount.
  - Do not worry about finding the exact highest bottleneck path.
  - Maintain scaling parameter Δ.
  - Let $G_f(\Delta)$ be the sub-graph of the residual graph consisting of only edges with capacity at least Δ.



G_f



$G_f(100)$

CAPACITY SCALING

- The algorithm runs in $O(m^2 \log U)$ time.

ScalingMaxFlow(V, E, s, t)

```
FOREACH e ∈ E
    f(e) ← 0
Δ ← largest power of 2 that is <= U
WHILE (Δ ≥ 1)
    G_f(Δ) ← Δ-residual graph
    WHILE(there exists augmenting path P in G_f(Δ))
        f ← augment(f, P)
        update G_f(Δ)
    Δ ← Δ / 2
RETURN f
```

HISTORY

Year	Discoverer	Method	Big-Oh
1951	Dantzig	Simplex	mn^2U
1955	Ford, Fulkerson	Augmenting path	mnU
1970	Edmonds-Karp	Shortest path	m^2n
1970	Dinitz	Shortest path	mn^2
1972	Edmonds-Karp, Dinitz	Capacity scaling	$m^2 \log U$
1973	Dinitz-Gabow	Capacity scaling	$mn \log U$
1974	Karzanov	Preflow-push	n^3
1983	Sleator-Tarjan	Dynamic trees	$mn \log n$
1986	Goldberg-Tarjan	FIFO preflow-push	$mn \log (n^2 / m)$
. . .	. . .	. . .	. . .
1997	Goldberg-Rao	Length function	$m^{3/2} \log (n^2 / m) \log U$ $mn^{2/3} \log (n^2 / m) \log U$