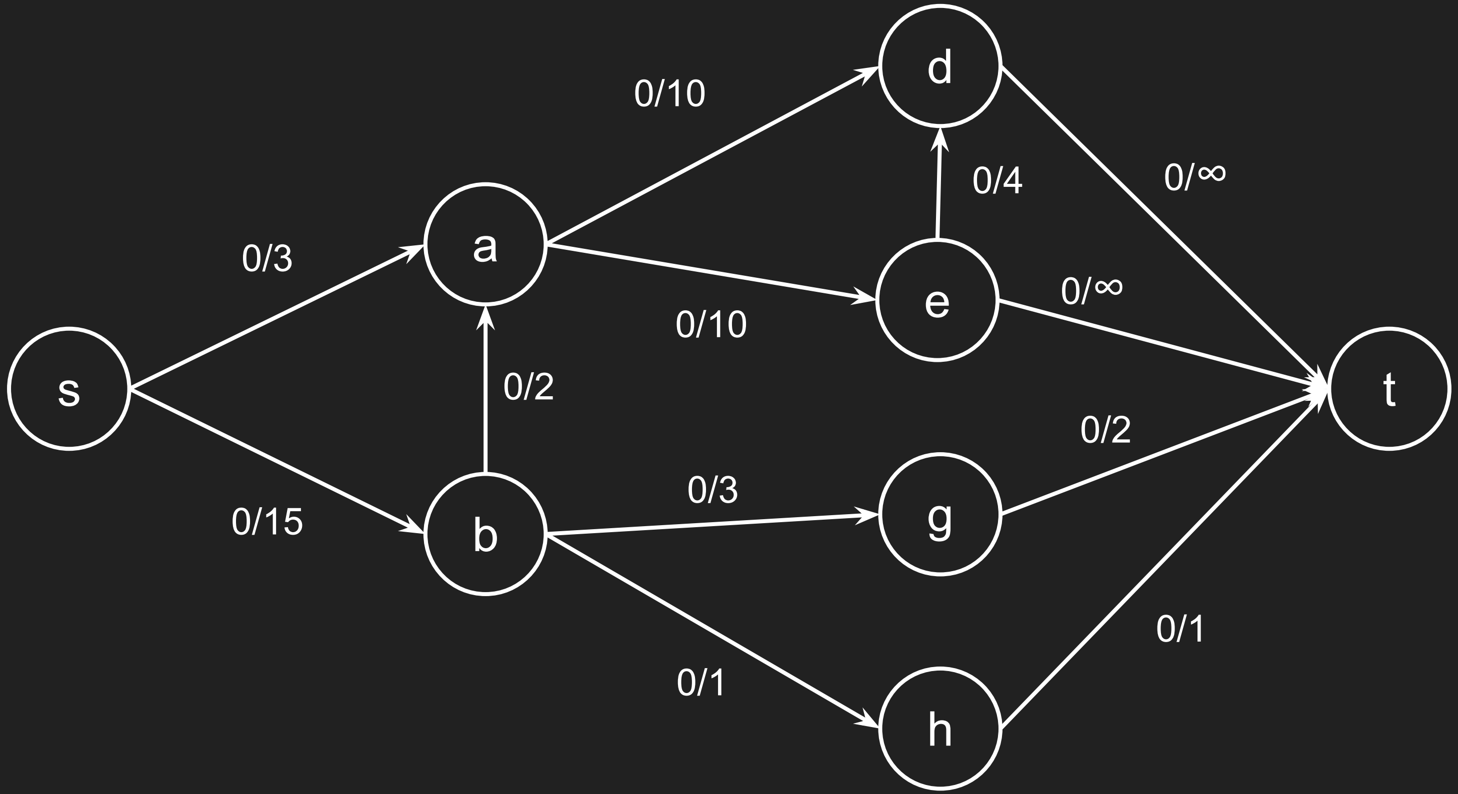
Max Flow / Min Cut

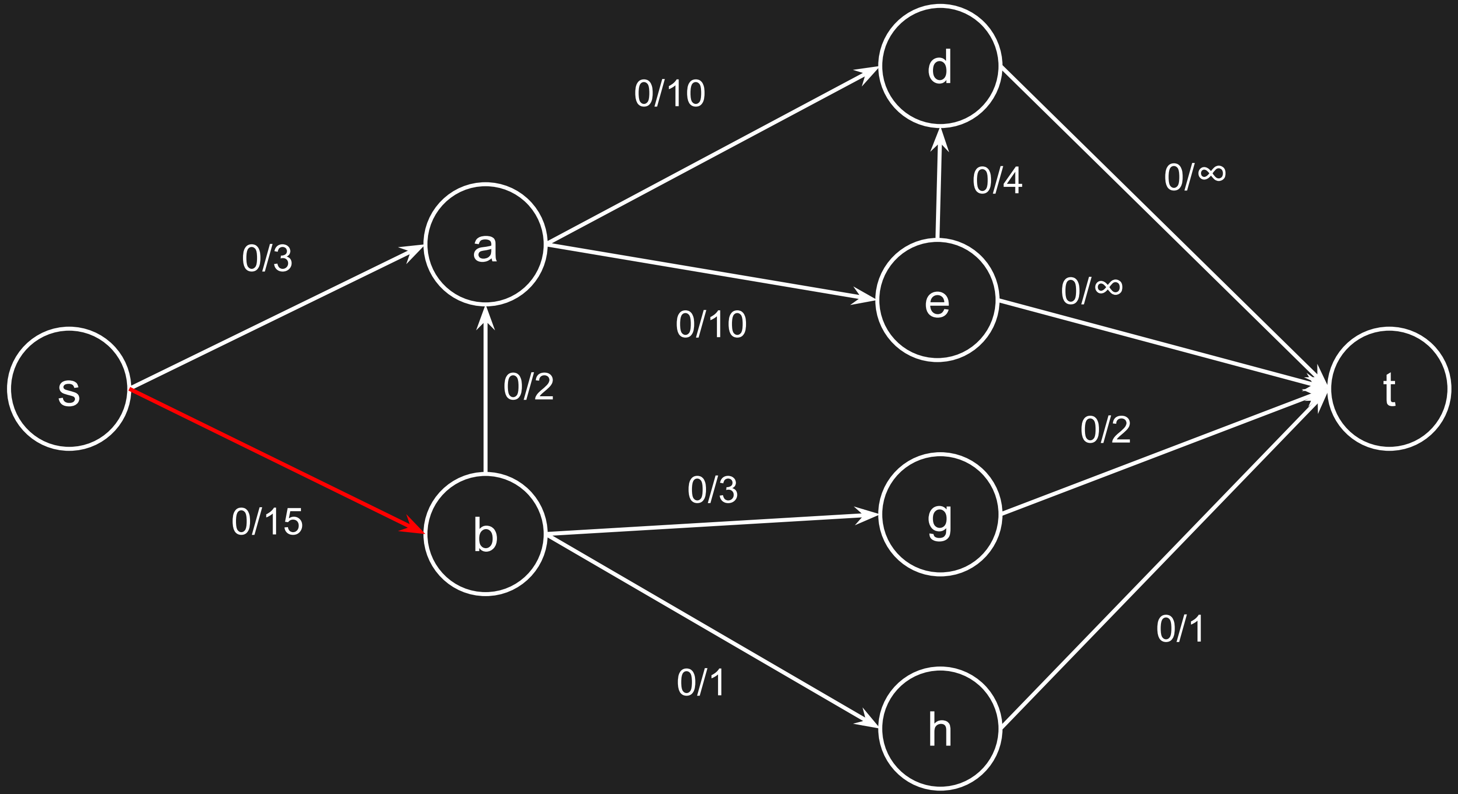
Max Flow

- Given a directed graph *G* with a special source node *s*, a special sink node *t*
  - *s* has no inbound edges, and *t* has no outbound edges
- For each edge *e* in the graph, *c(e)* is the given "capacity" of the edge
  - The capacity must be greater than 0
  - For simplicity, assume the capacity is an integer or ∞
- Define *f(e)* to be the "flow" along an edge
  - The flow must be non-negative
  - The flow must also no greater than the capacity for a given edge
- For any given node, the sum of flows of inbound edges must equal the sum of flows of outbound edges ("conservation of flow")
  - Exceptions: *s* may have any amount of outbound flow, and *t* may have any amount of inbound flow
- Question: what is the maximum amount of flow that can be sent from *s* to *t*?

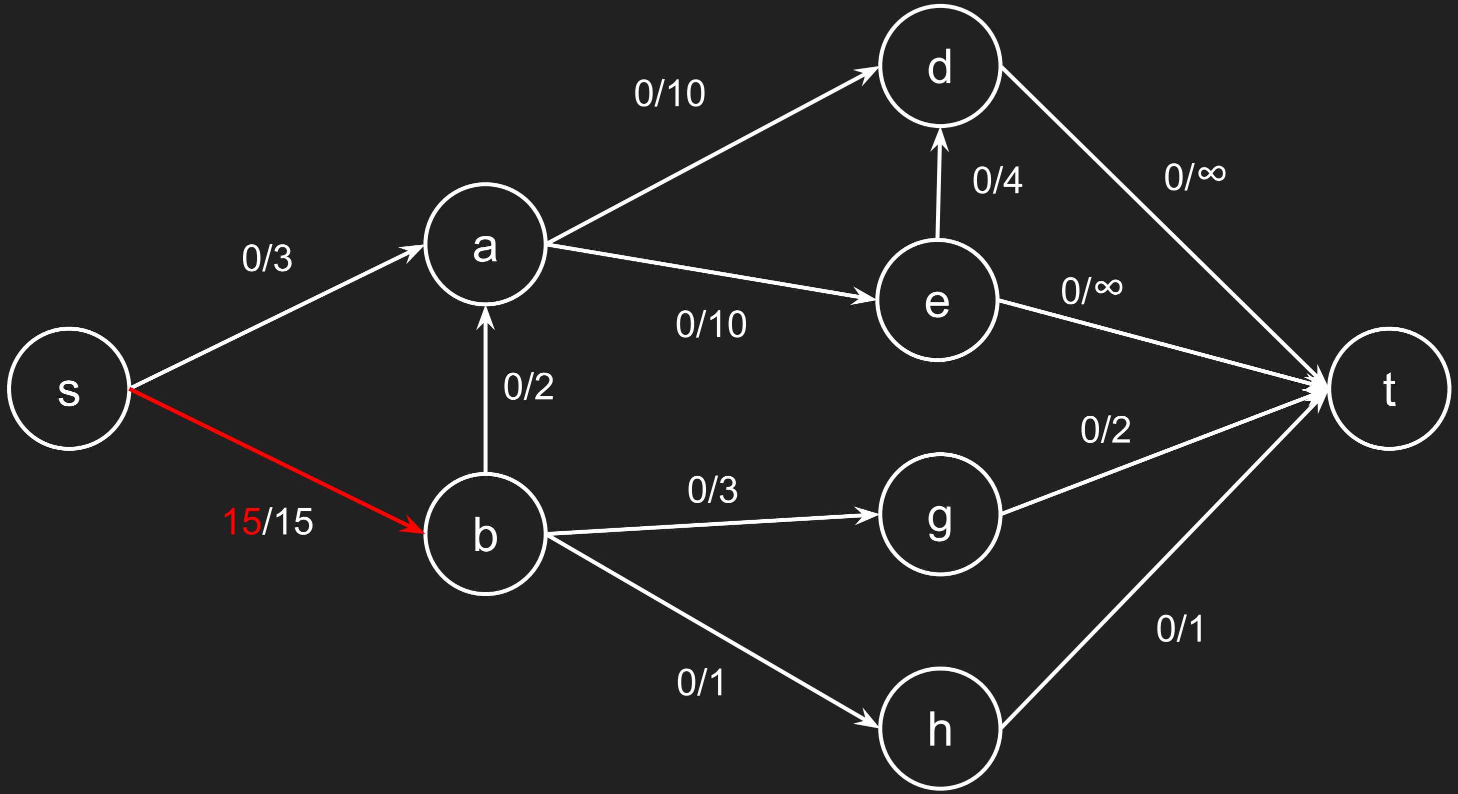
Max Flow



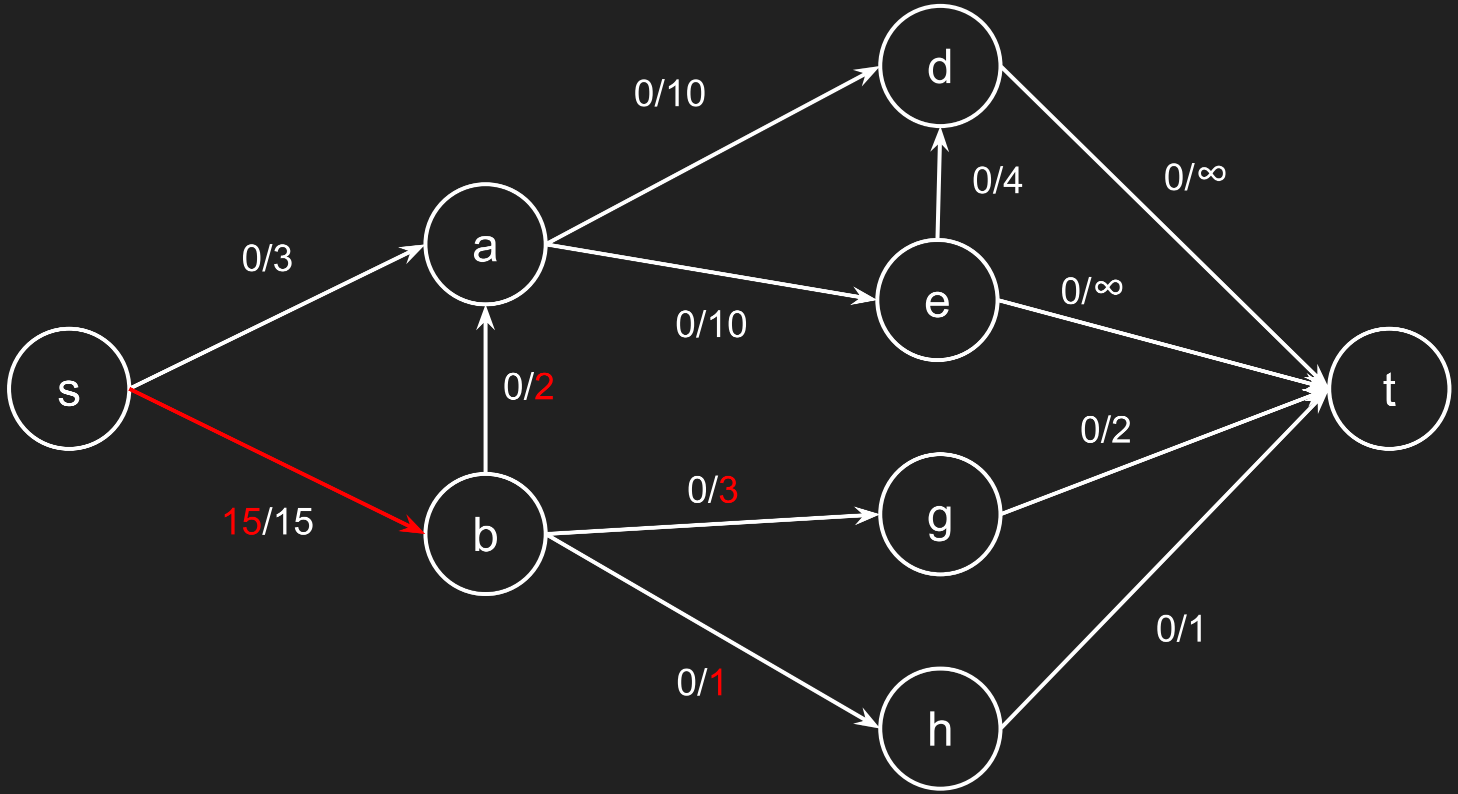
Max Flow



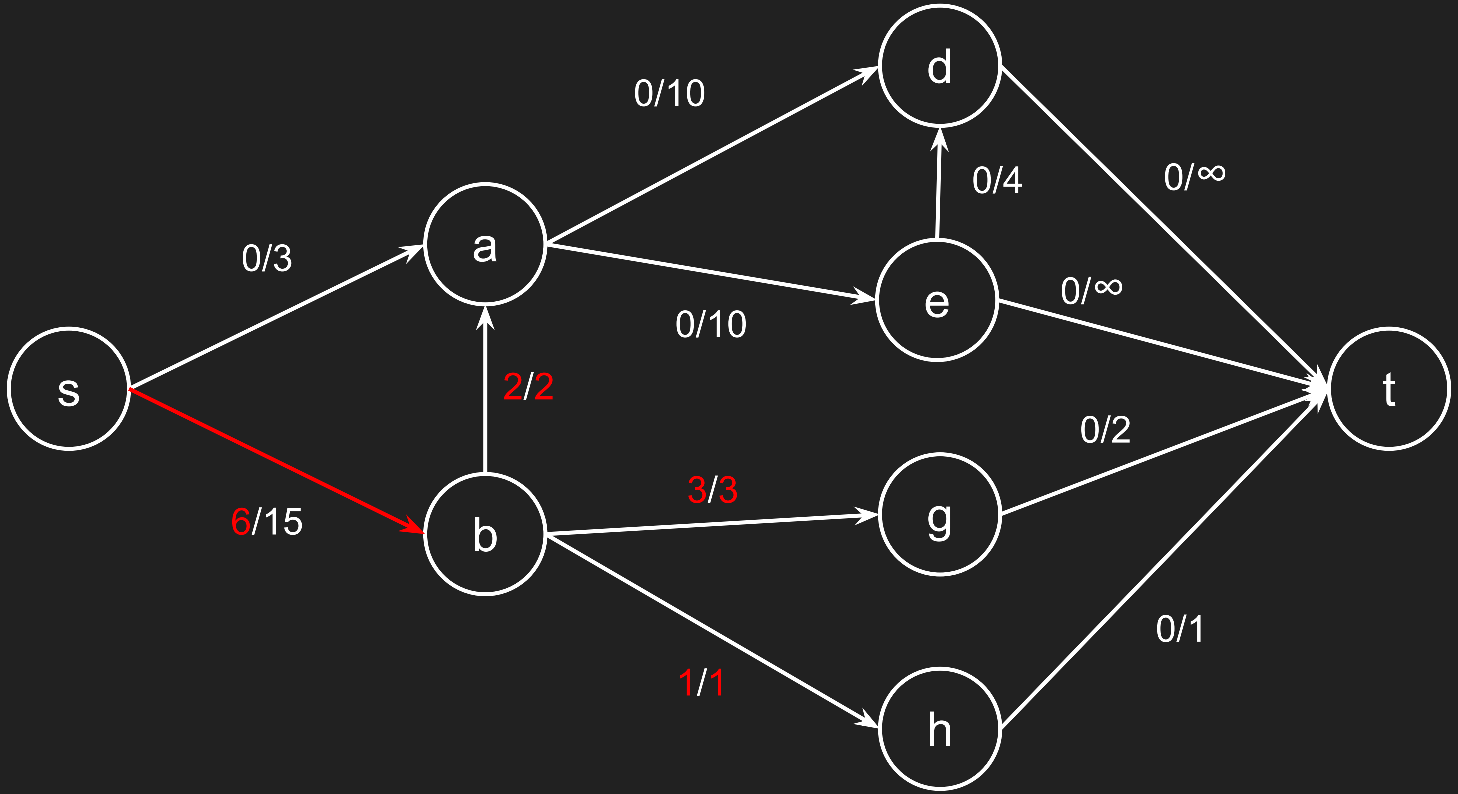
Max Flow



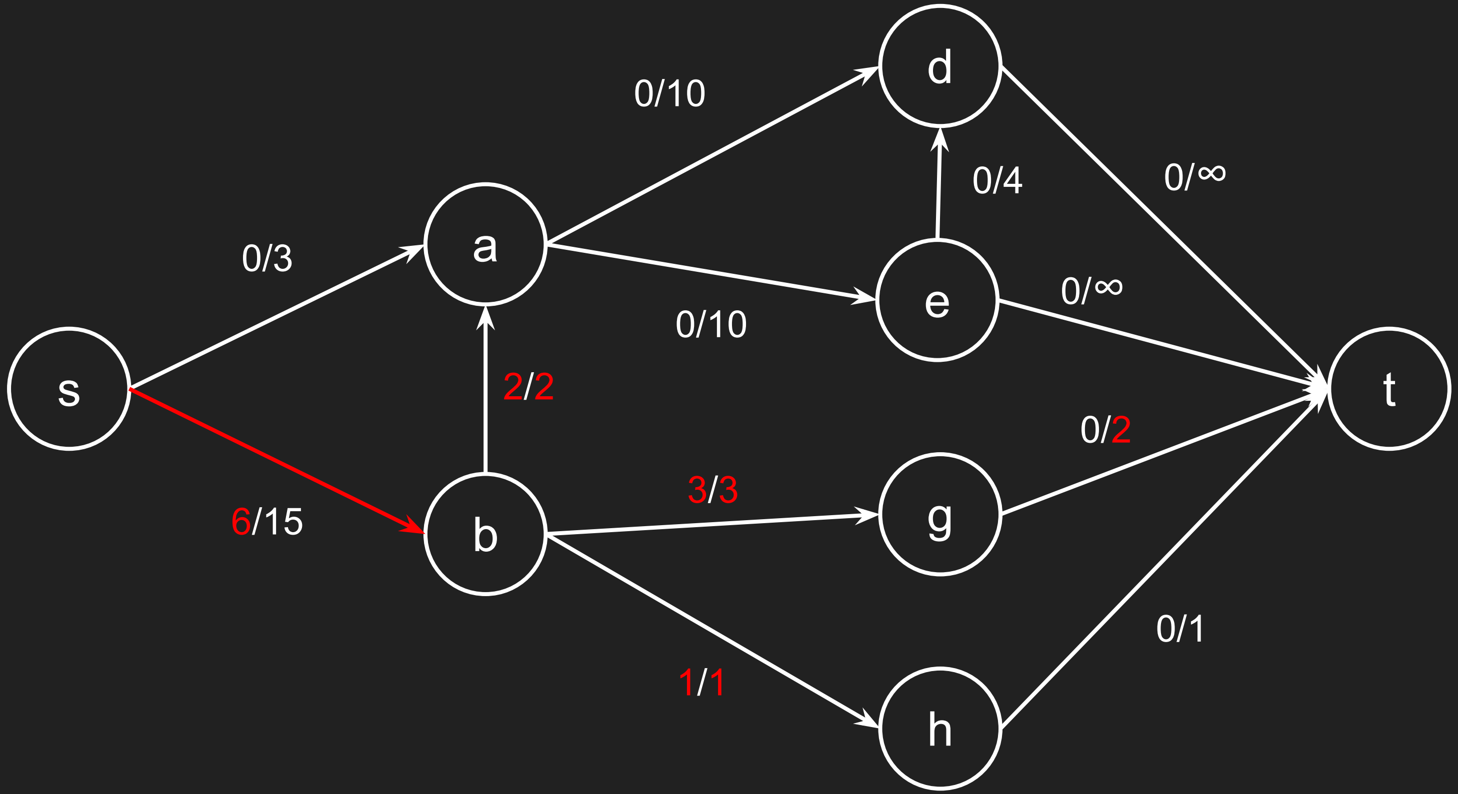
Max Flow



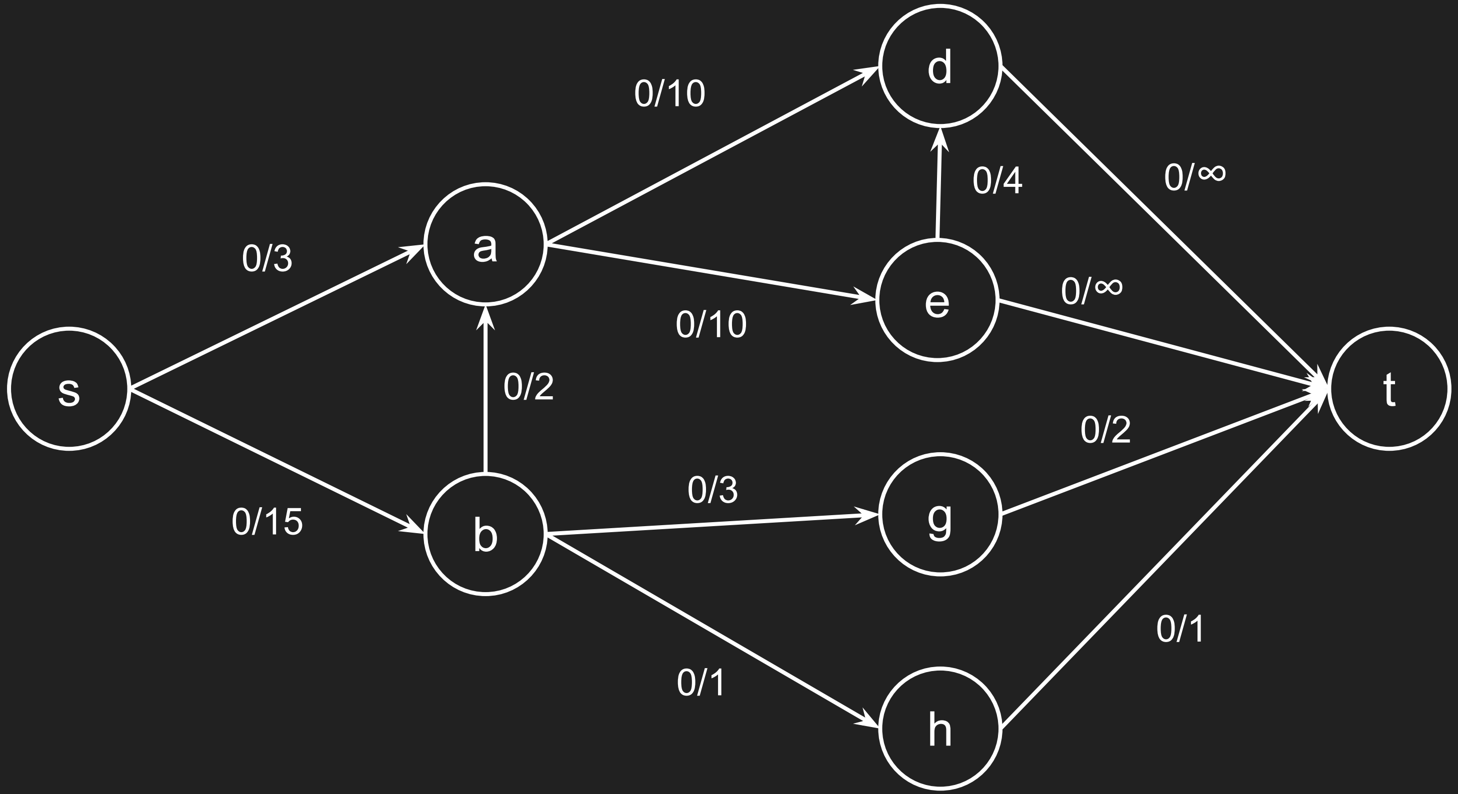
Max Flow



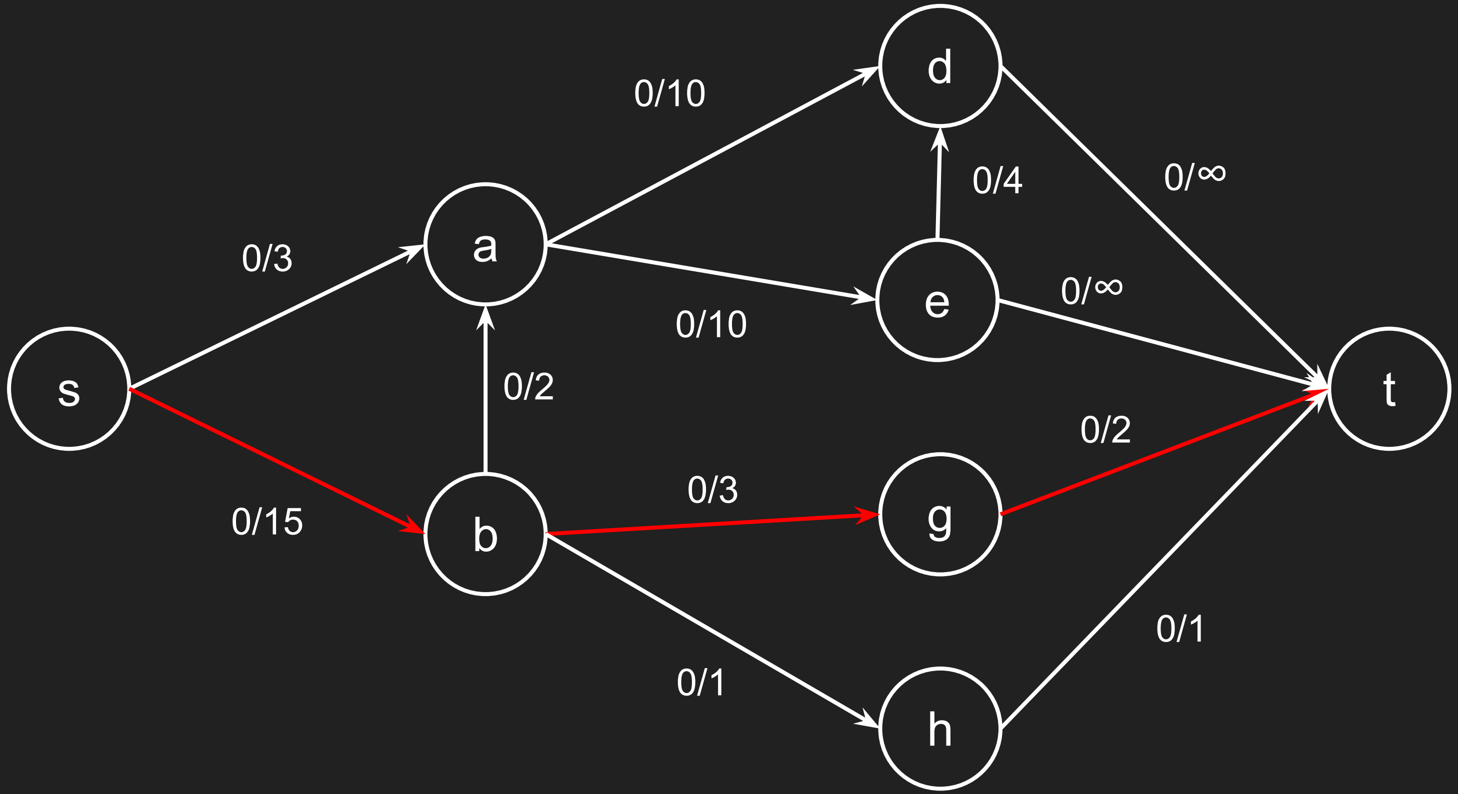
Max Flow



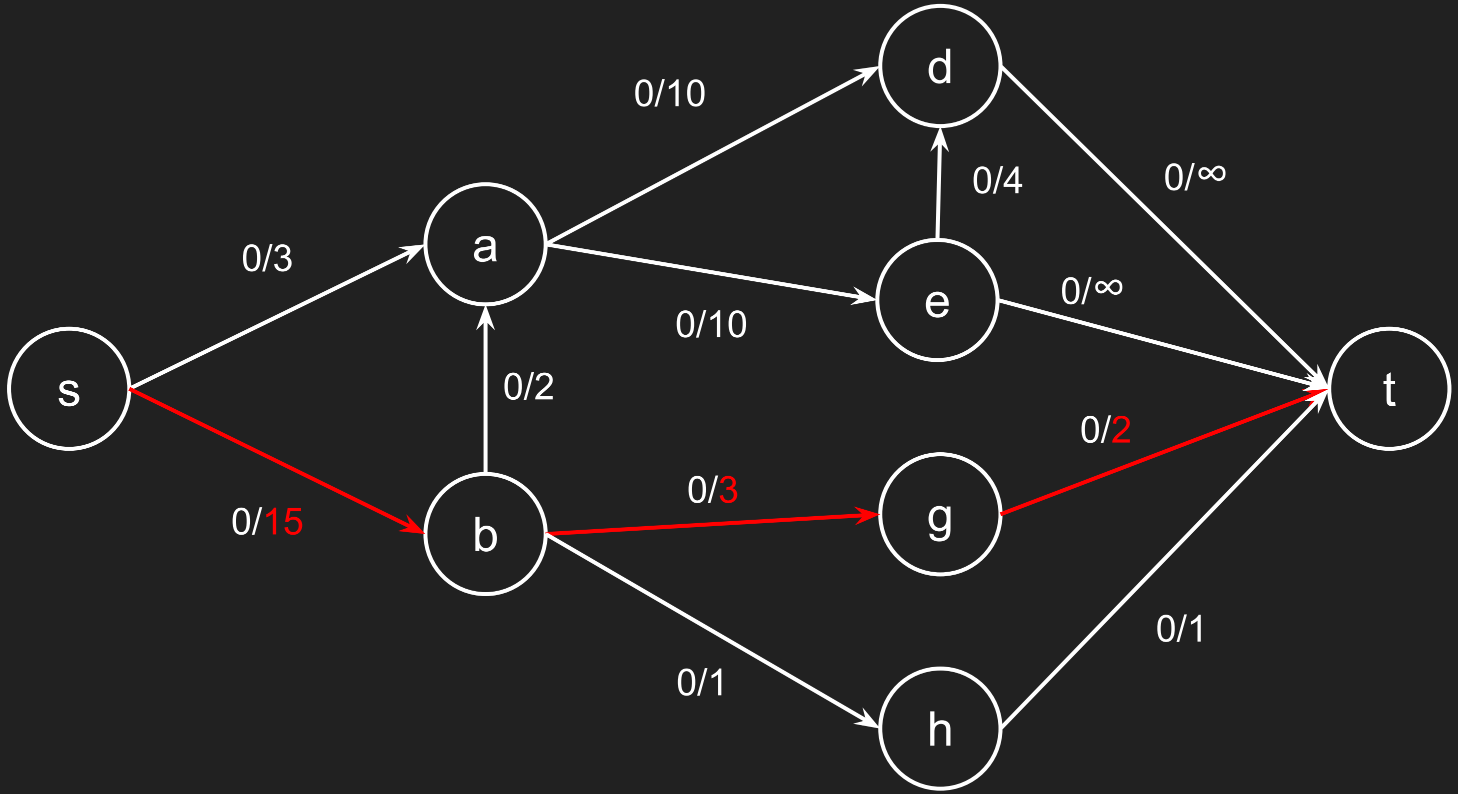
Max Flow



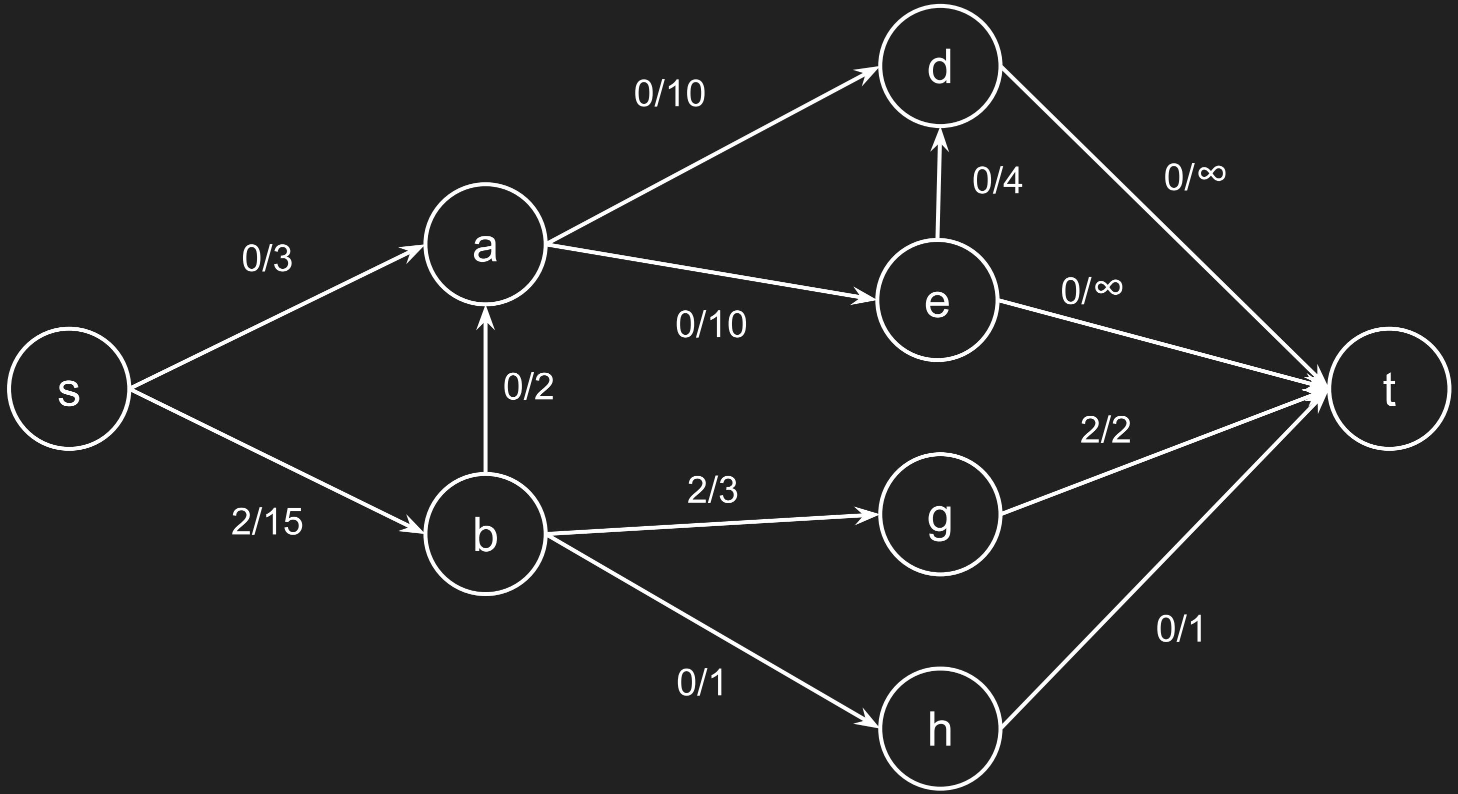
Max Flow



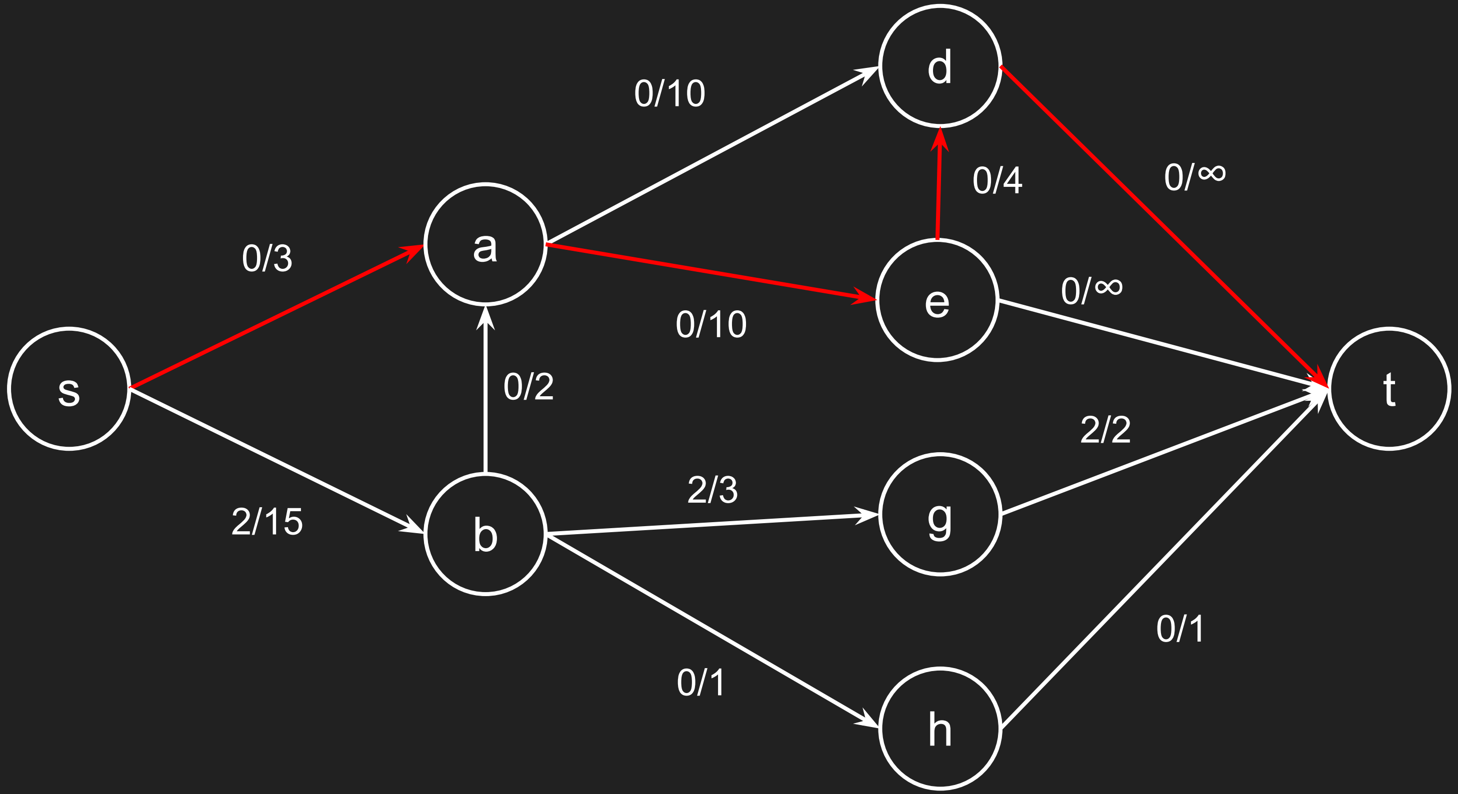
Max Flow



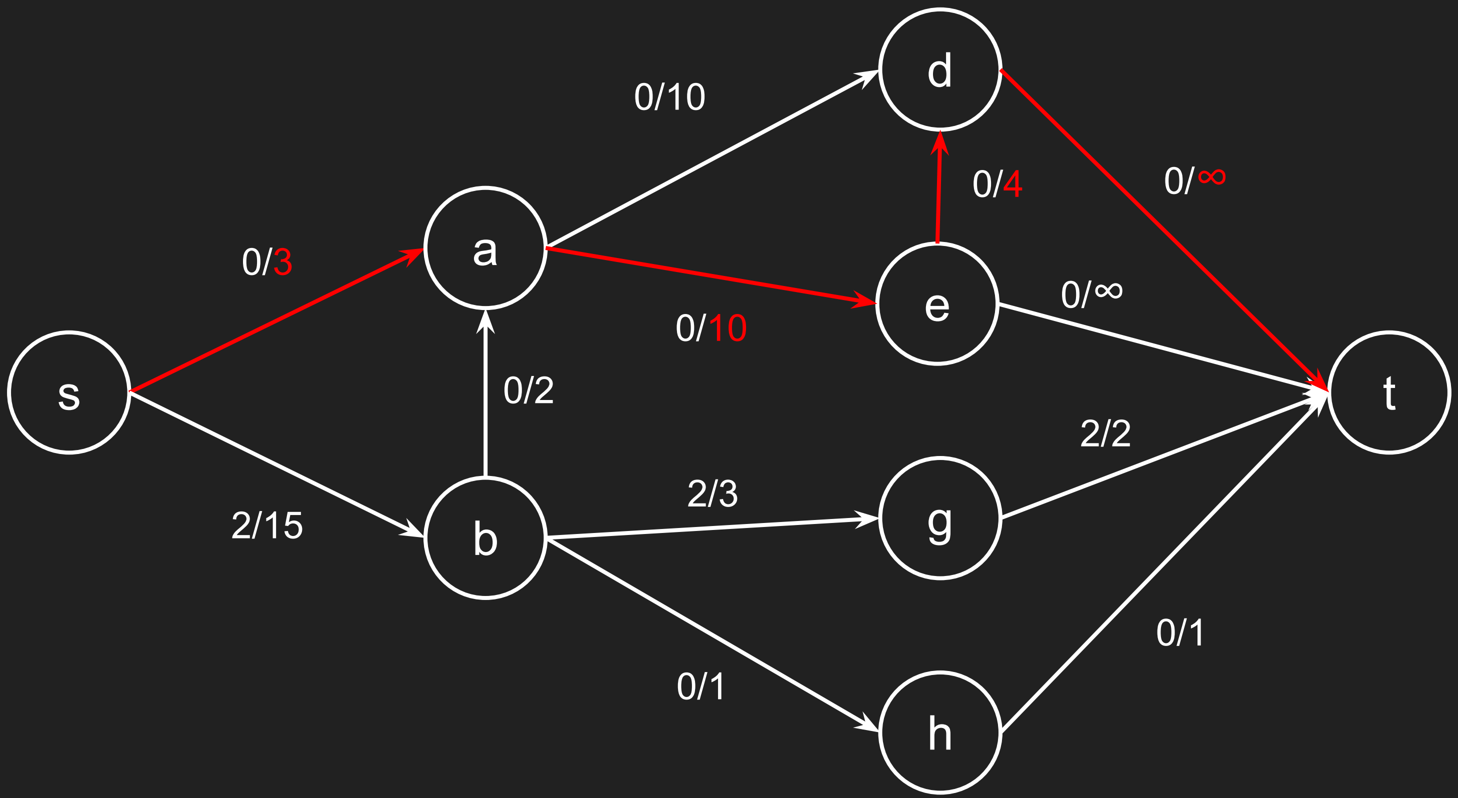
Max Flow



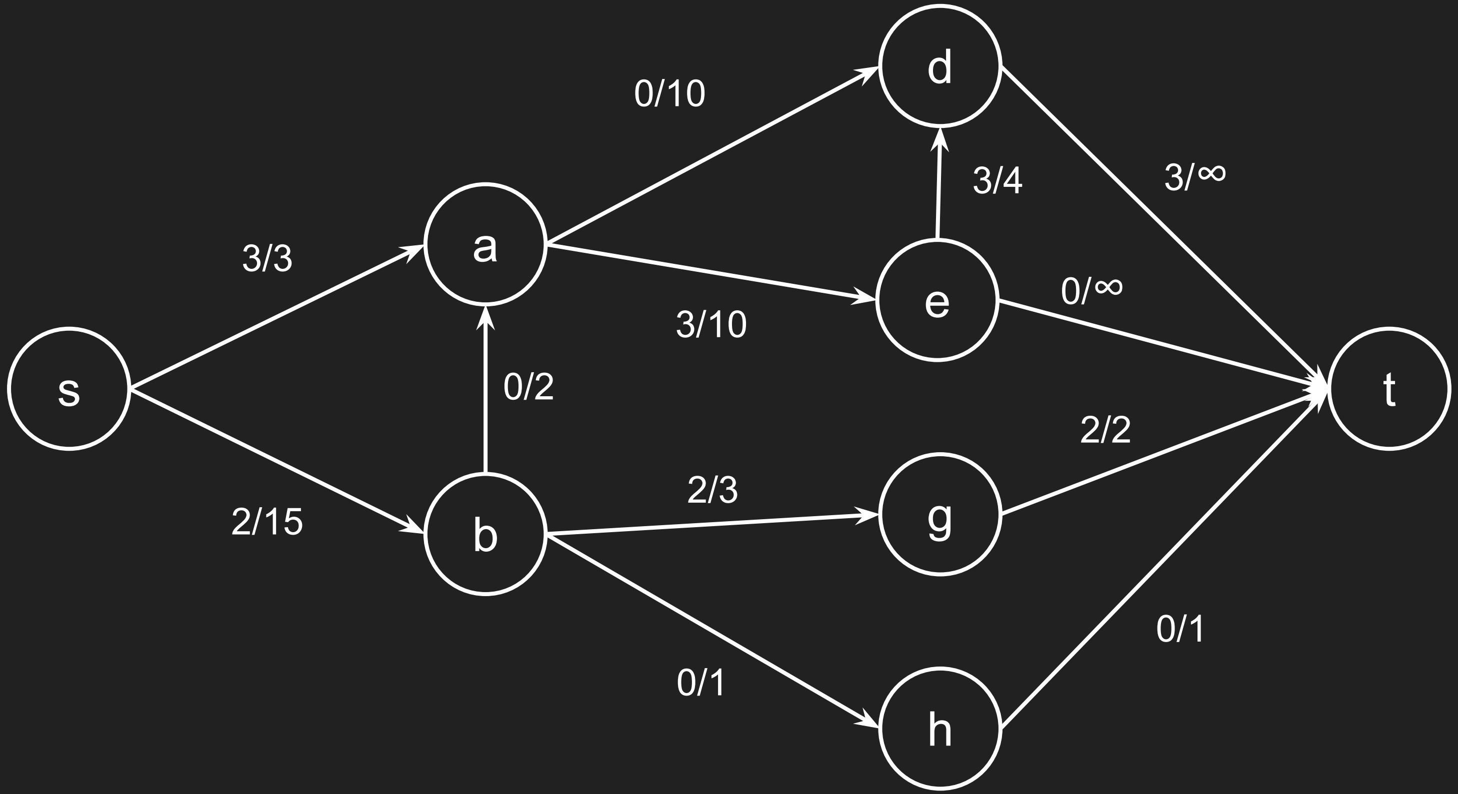
Max Flow



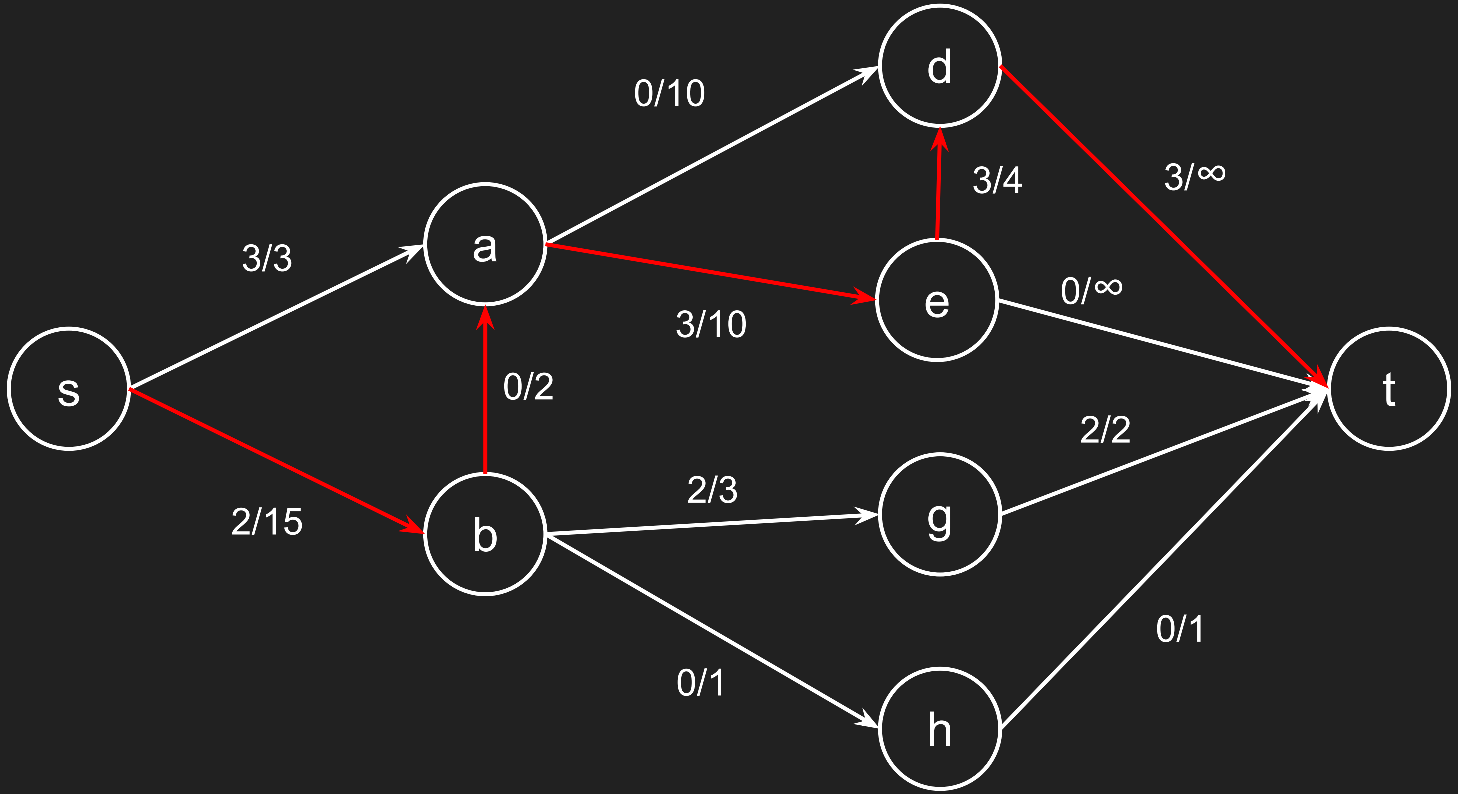
Max Flow



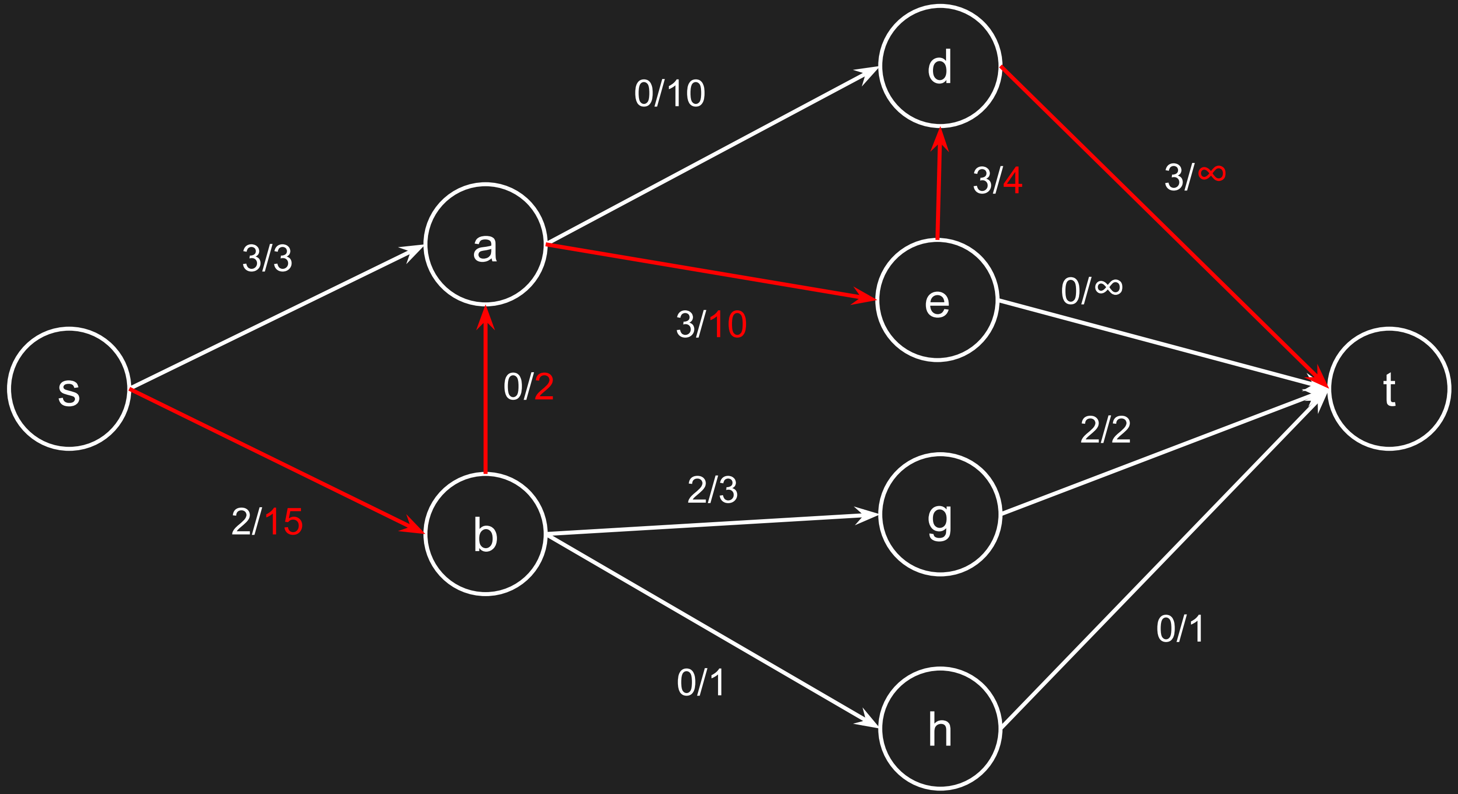
Max Flow



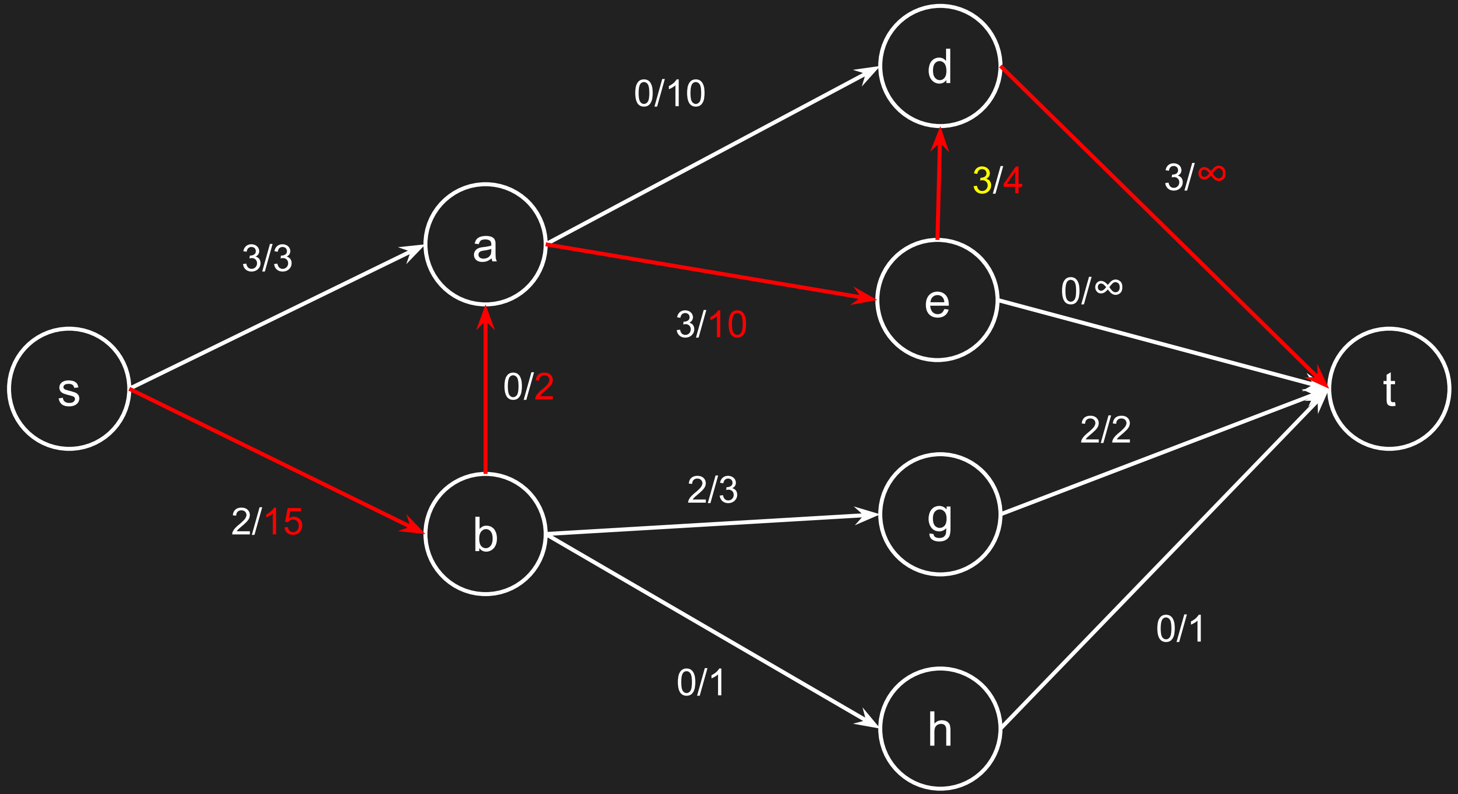
Max Flow



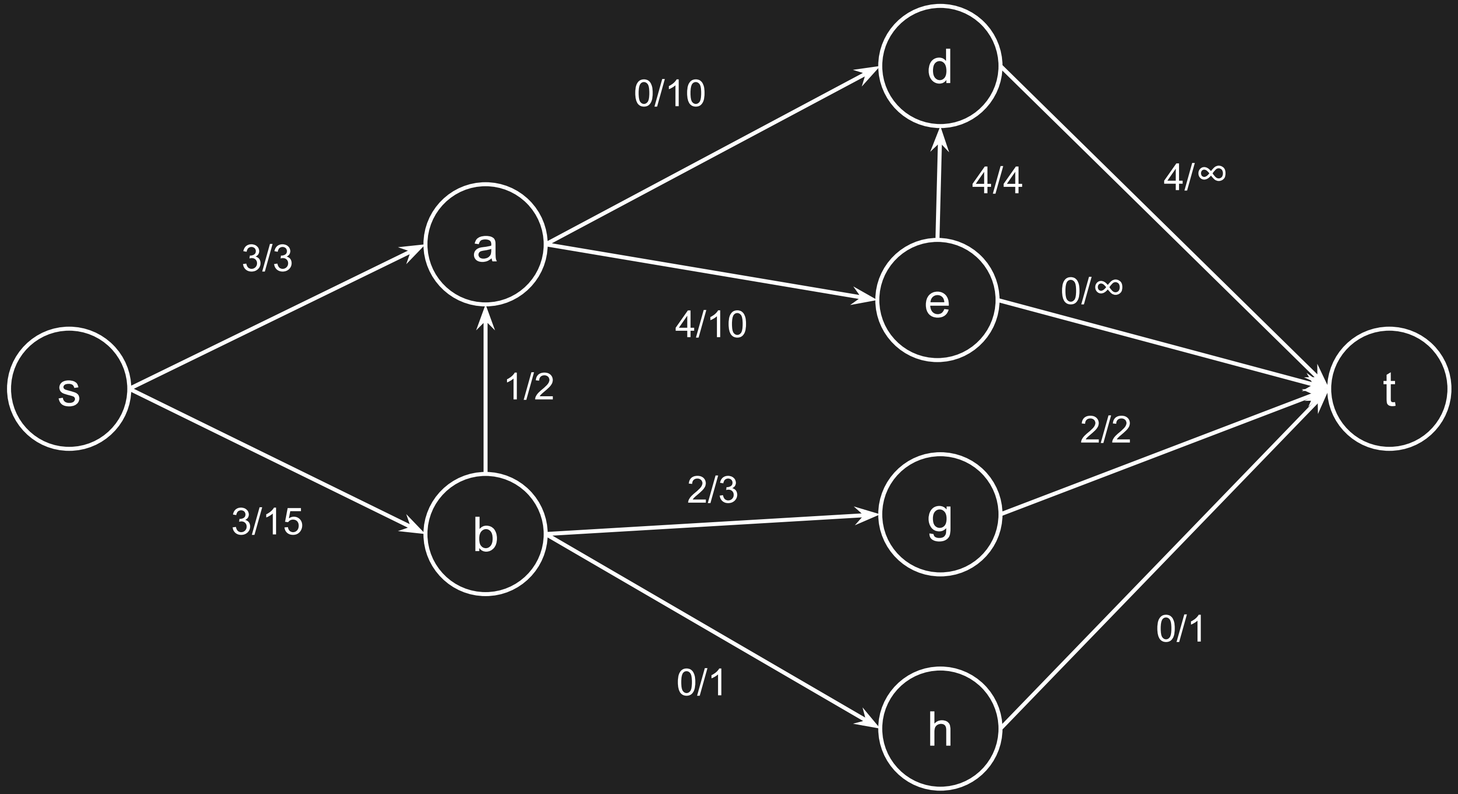
Max Flow



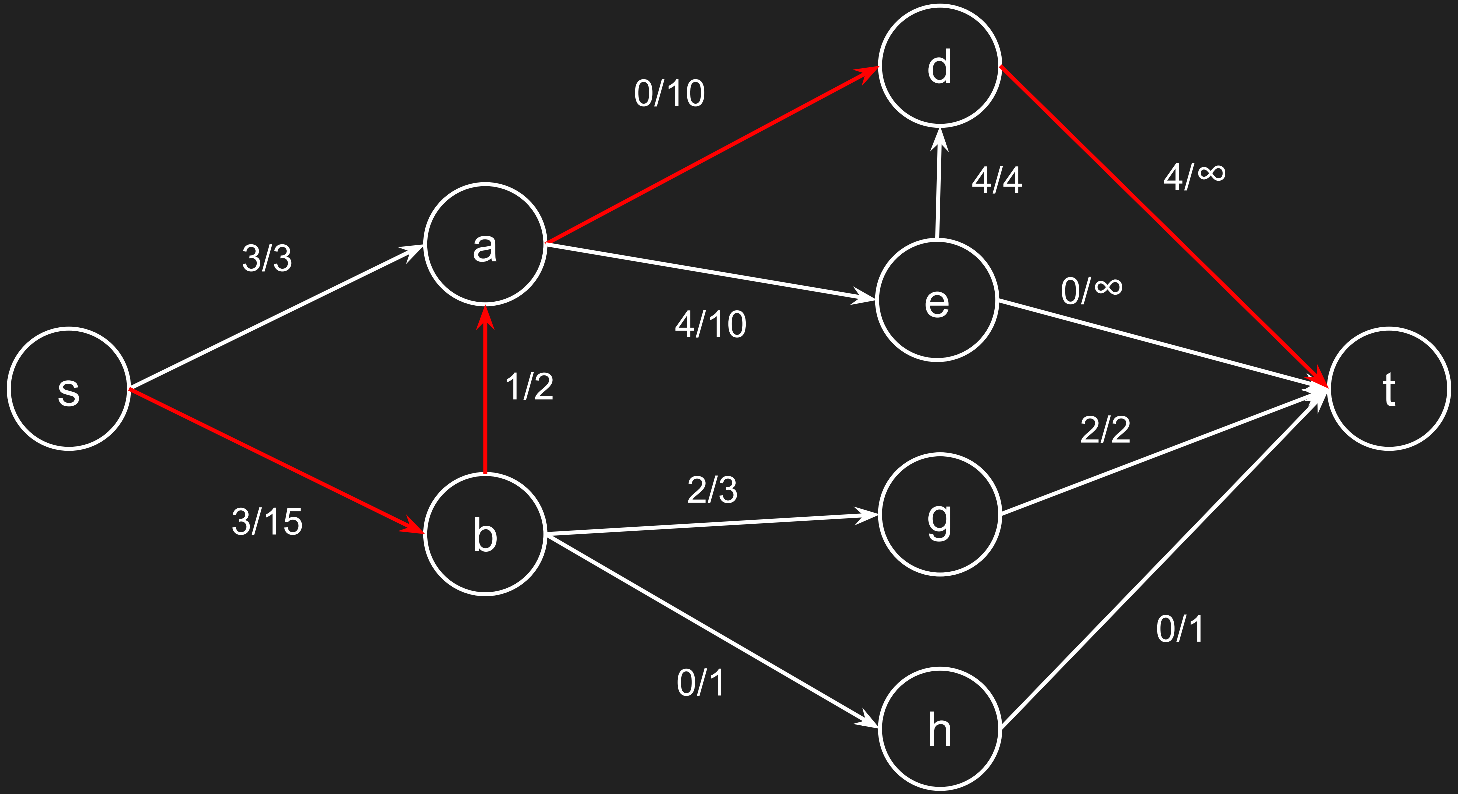
Max Flow



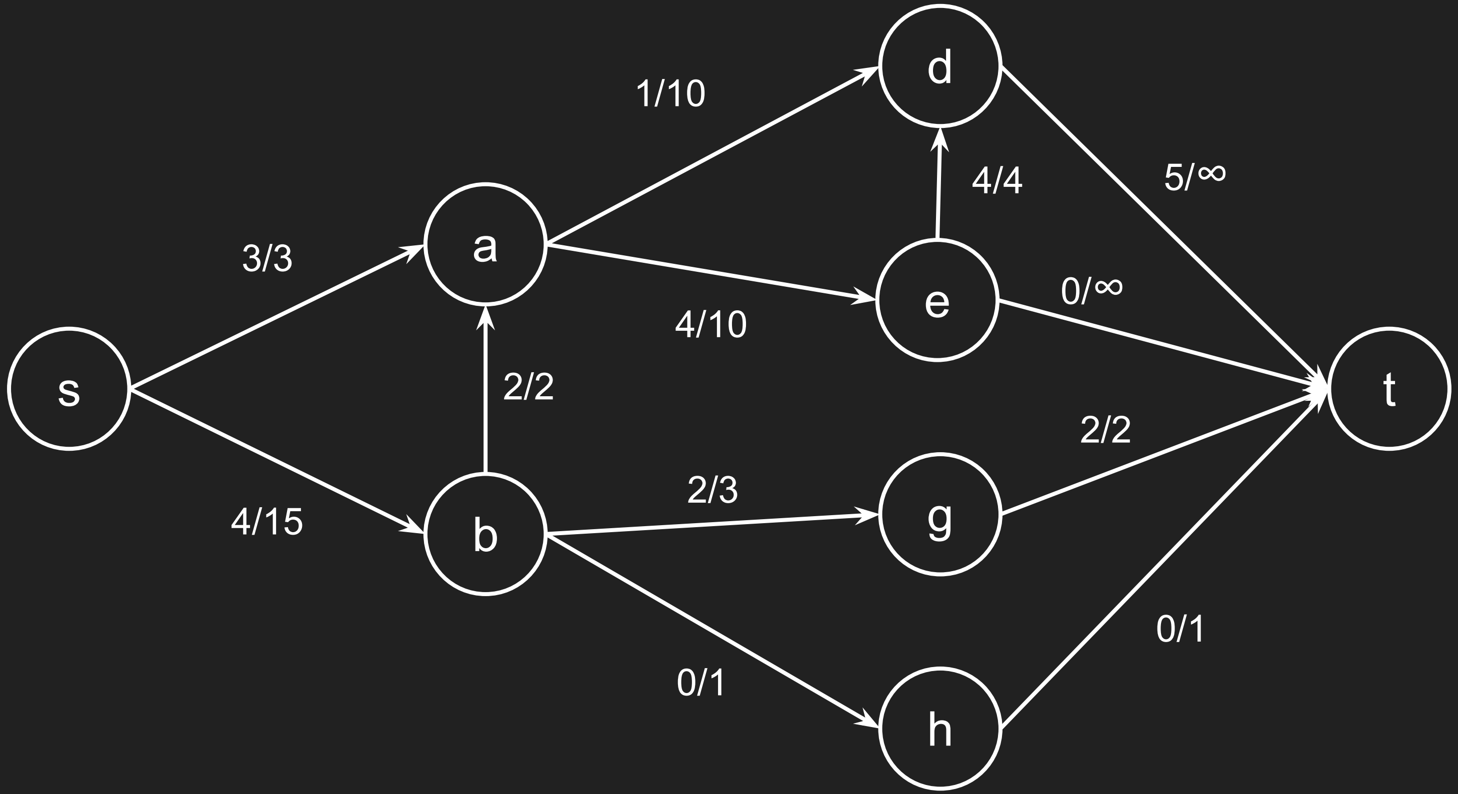
Max Flow



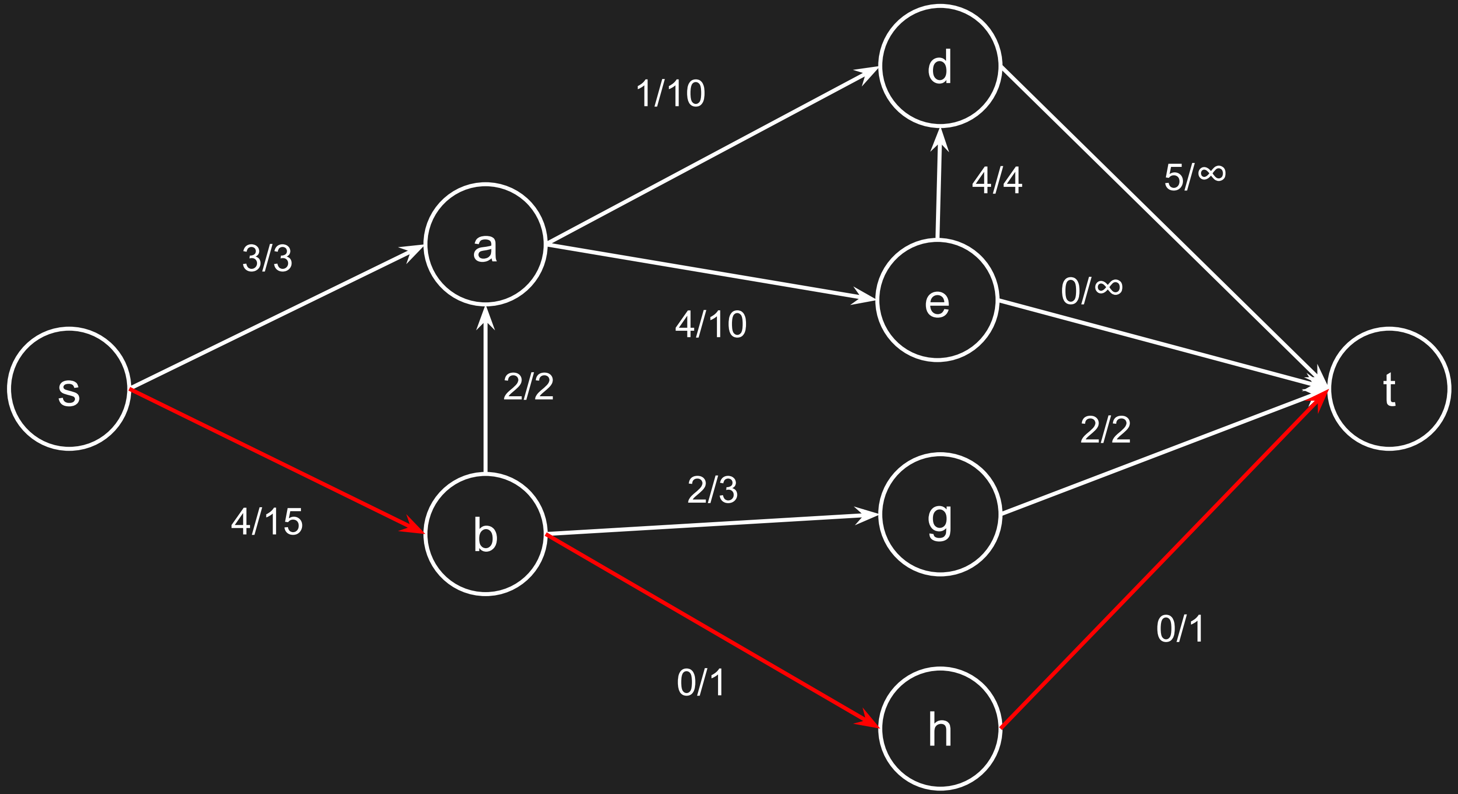
Max Flow



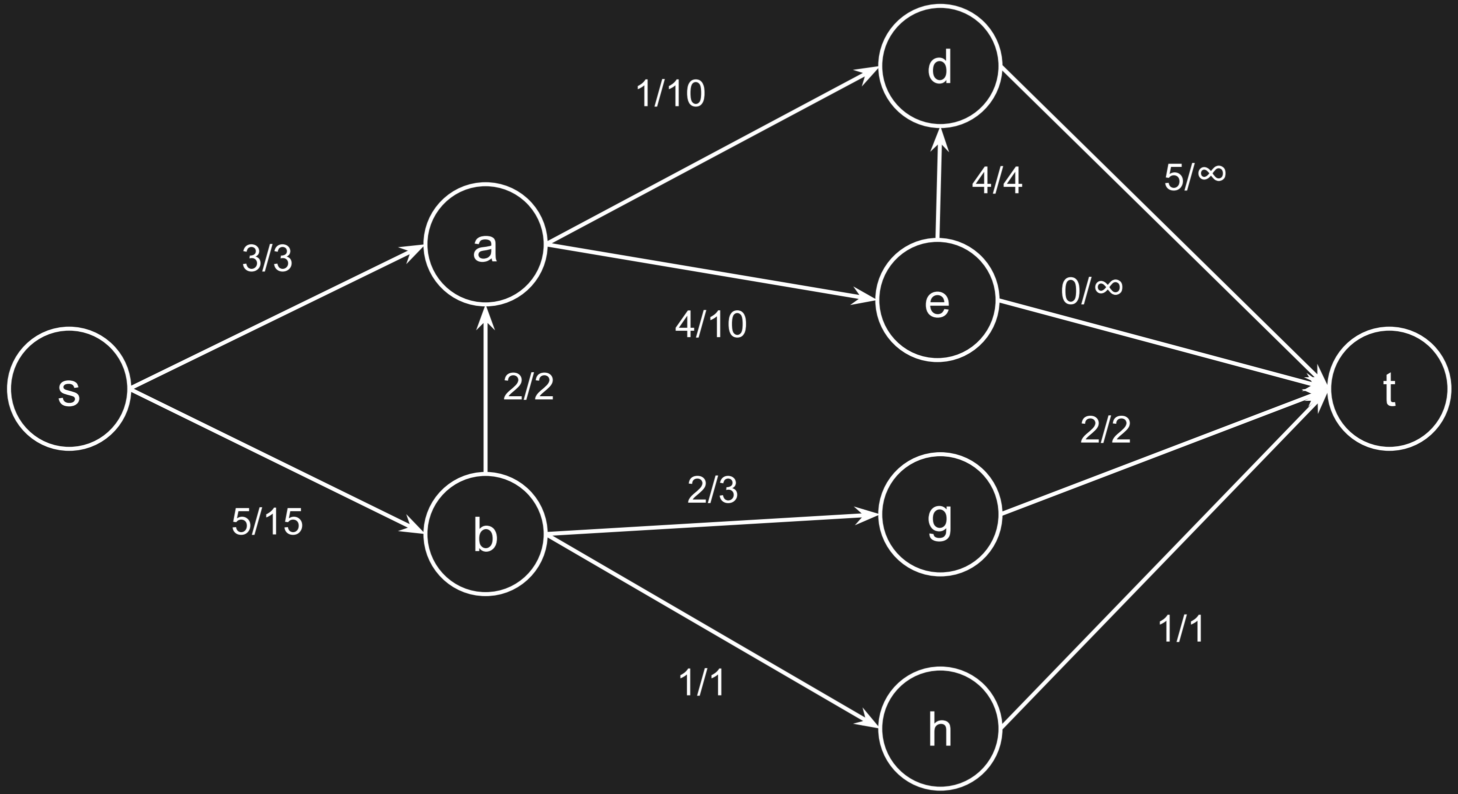
Max Flow



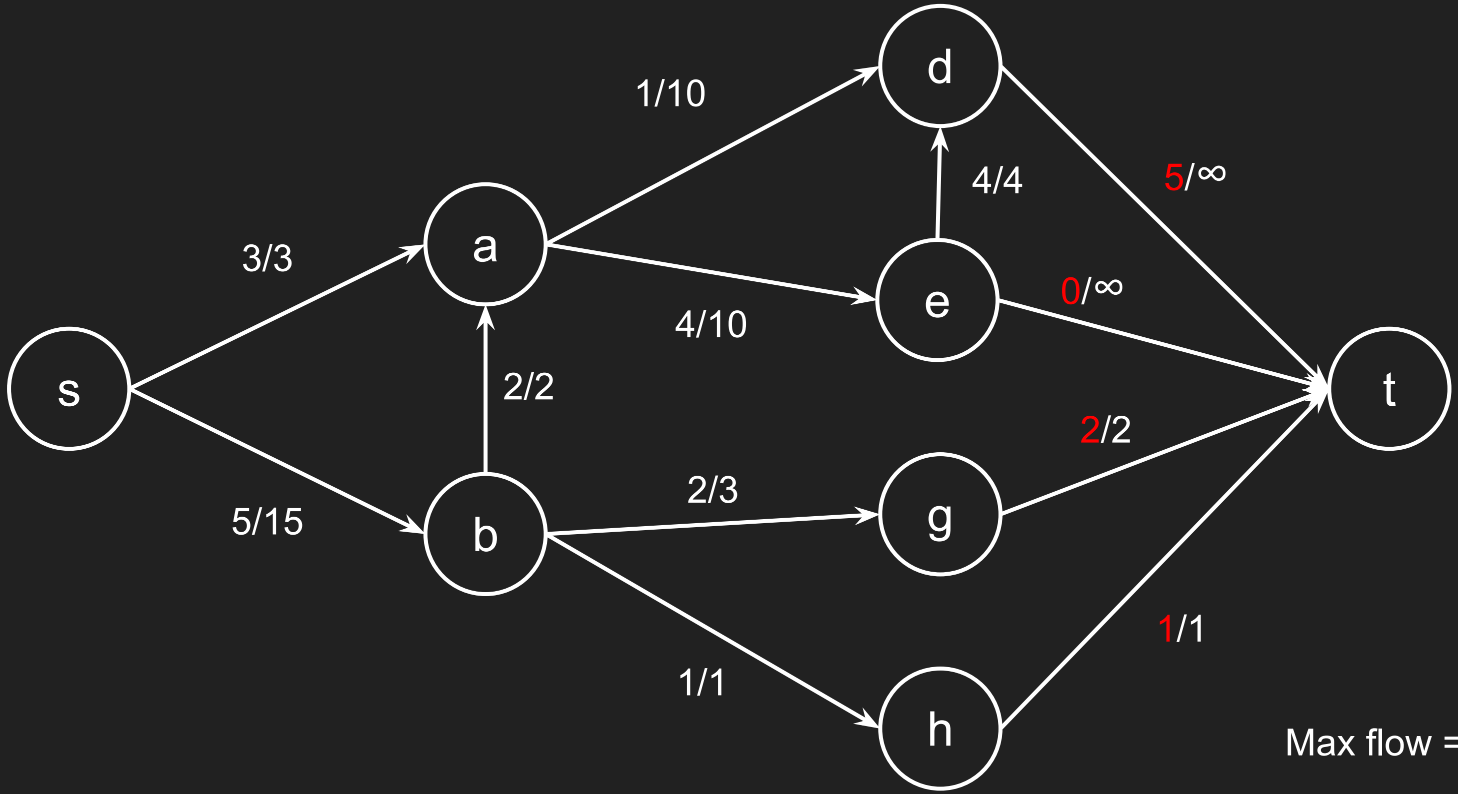
Max Flow



Max Flow

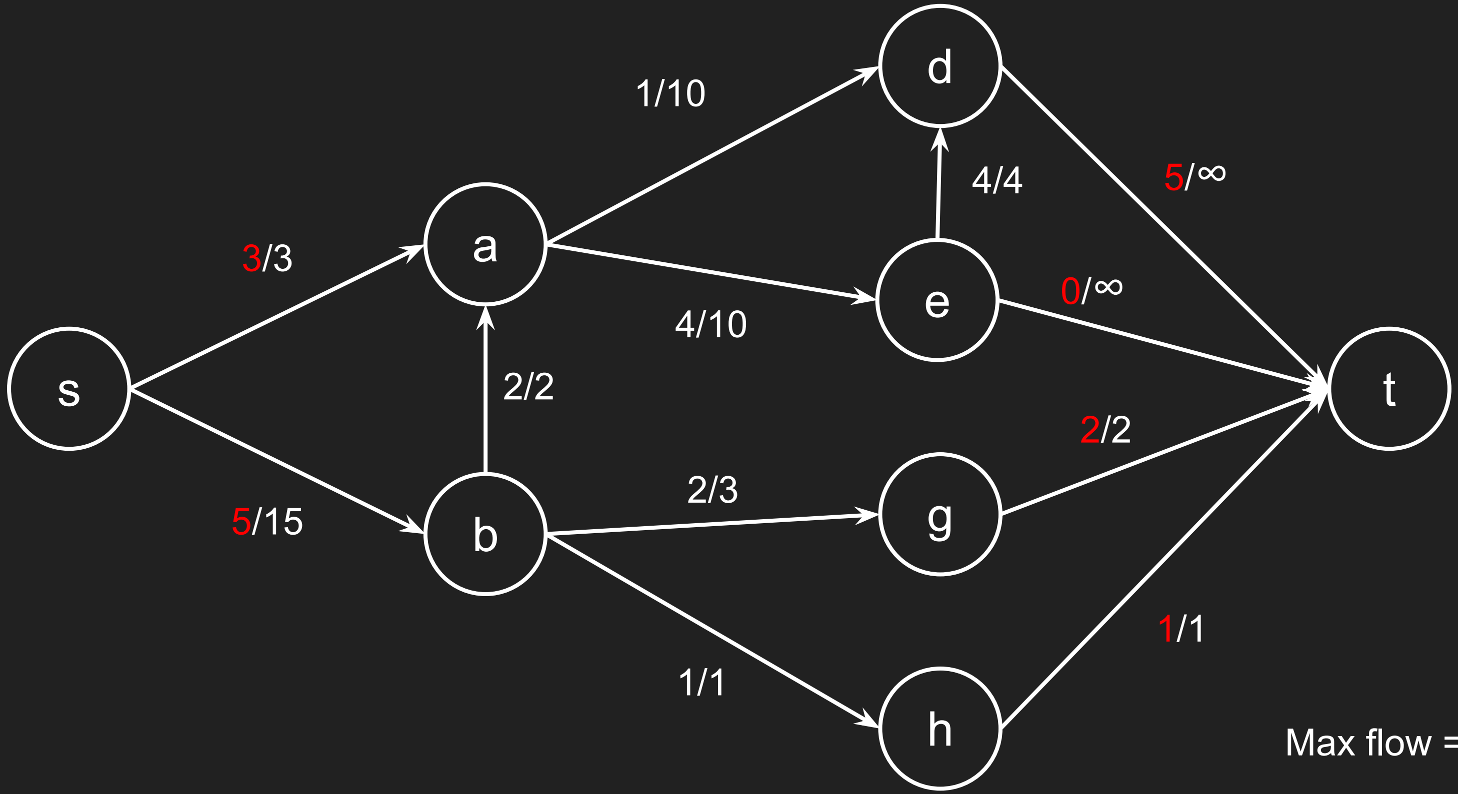


Max Flow



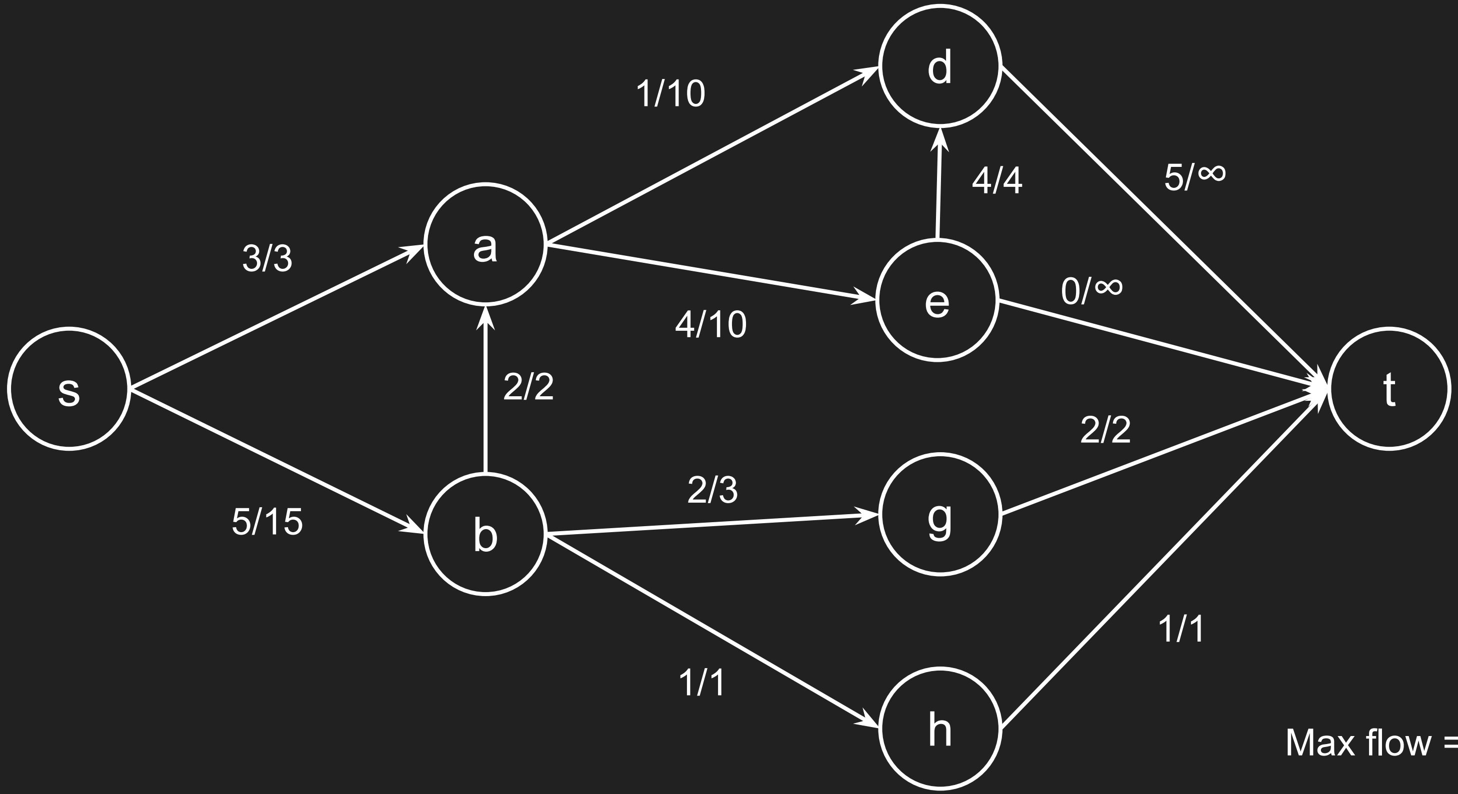
Max flow = 8

Max Flow



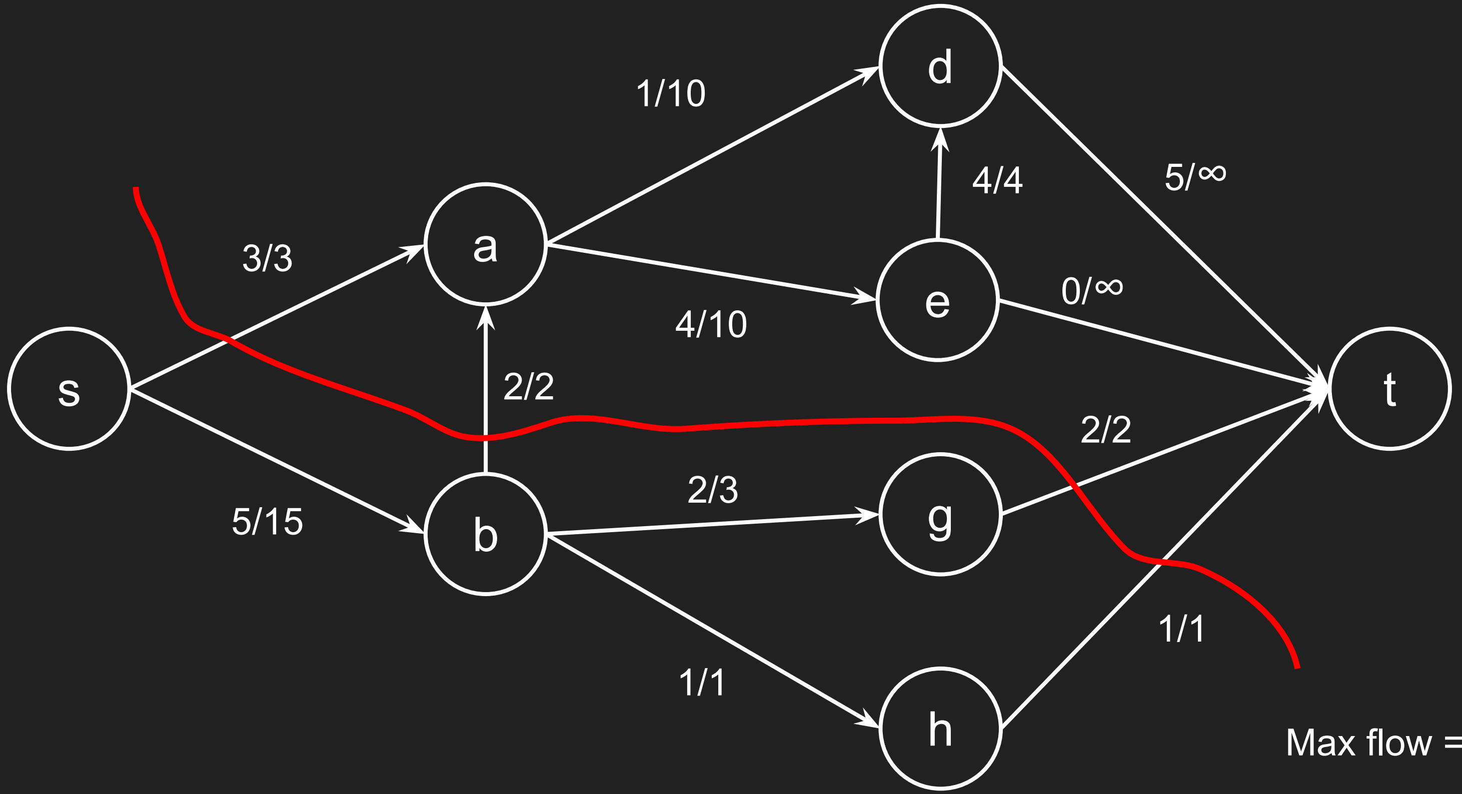
Max flow = 8

Max Flow



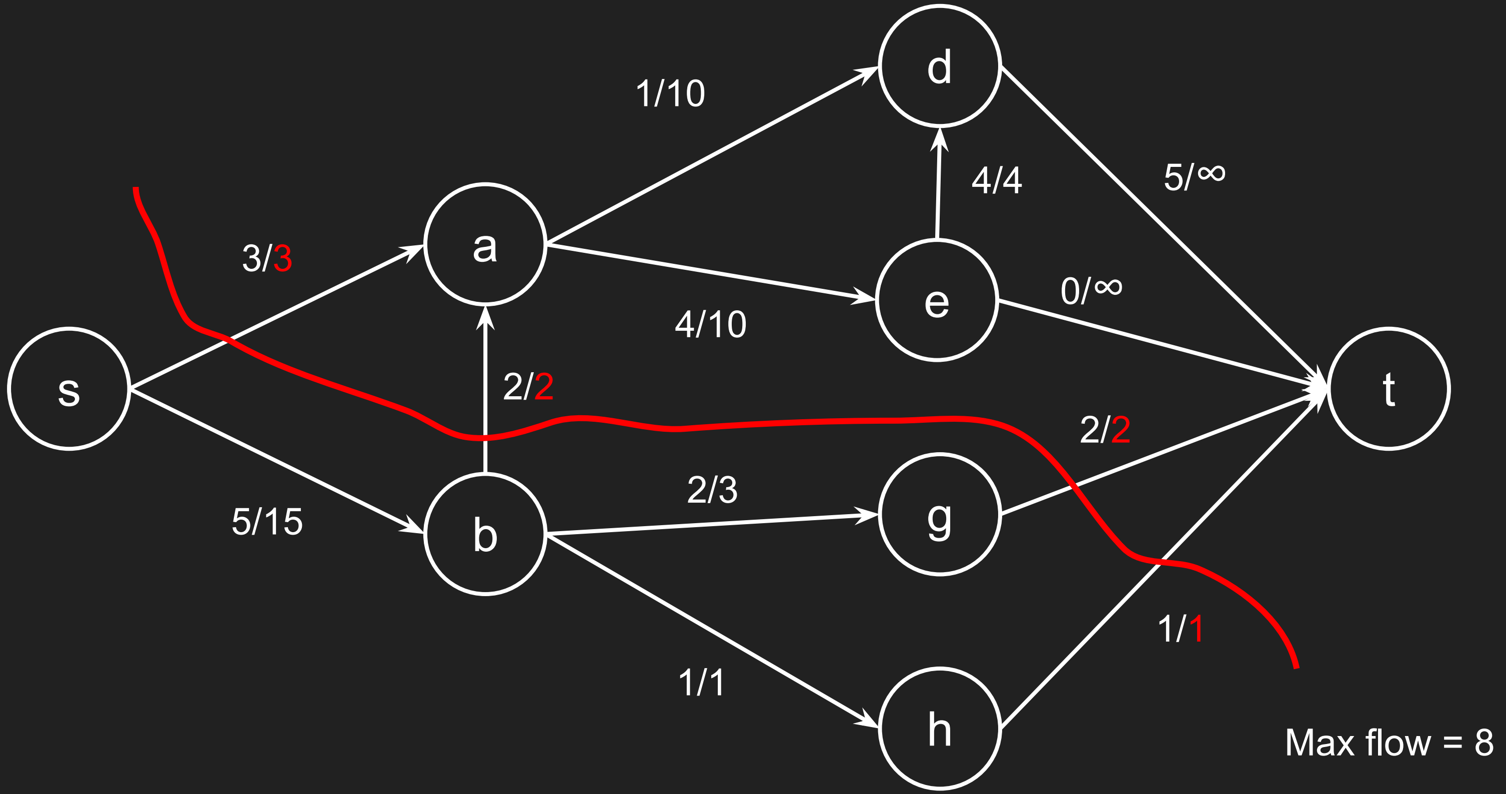
Max flow = 8

Max Flow



Max flow = 8

Max Flow

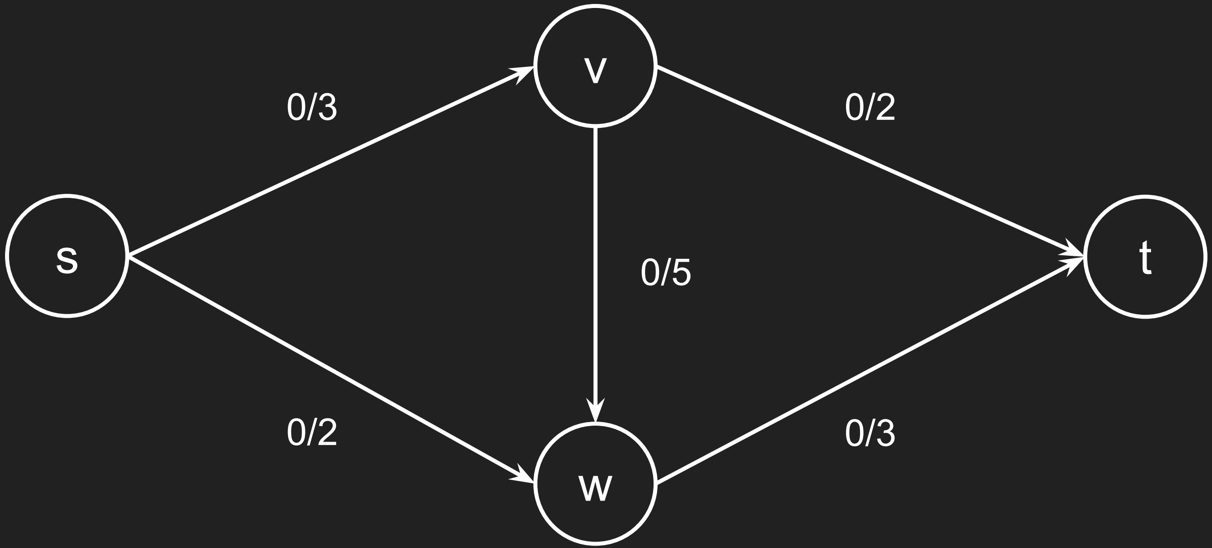


Max Flow / Min Cut

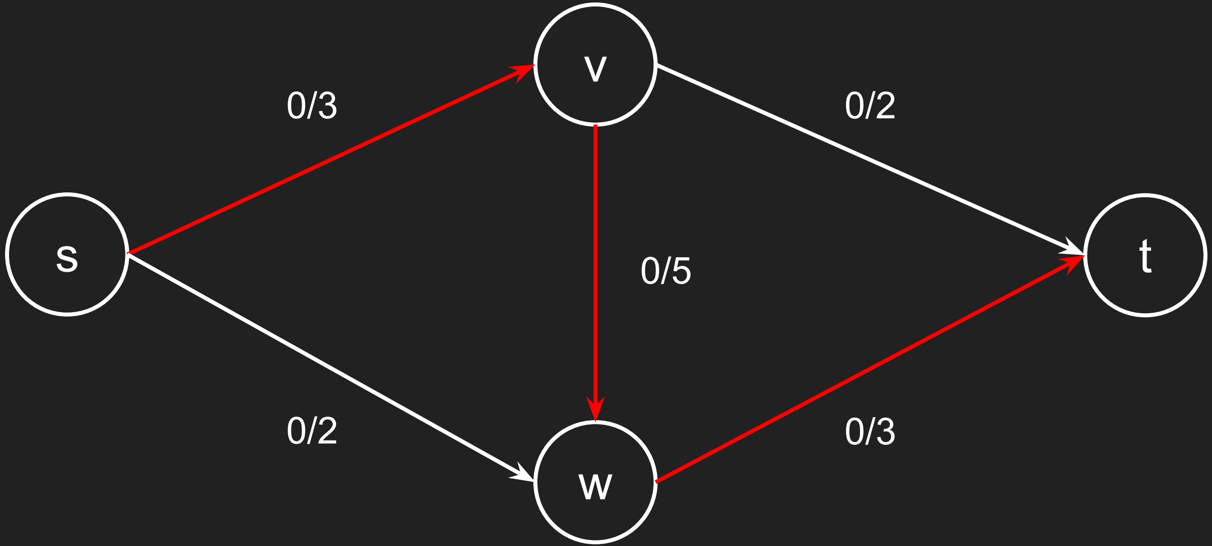
- Define a "cut" to be a partition of the nodes into two sets. An *s*-*t* cut is one where one set contains *s* and the other contains *t*
- Define the capacity of an *s*-*t* cut to be the sum of the capacities of the edges that "cross the partition boundary" from the *s* set to the *t* set
    - In other words, if the nodes are partitioned into sets A and B with $s \in A$ and $t \in B$, the edges from u to v where $u \in A$ and $v \in B$
- The Min Cut problem is to find the *s*-*t* cut with the minimum capacity
- The Max Flow / Min Cut Theorem says the answers are the same!
    - Essentially formalizes the notion of "bottlenecks"

- ...but does our bottleneck finding/saturating algorithm always work?

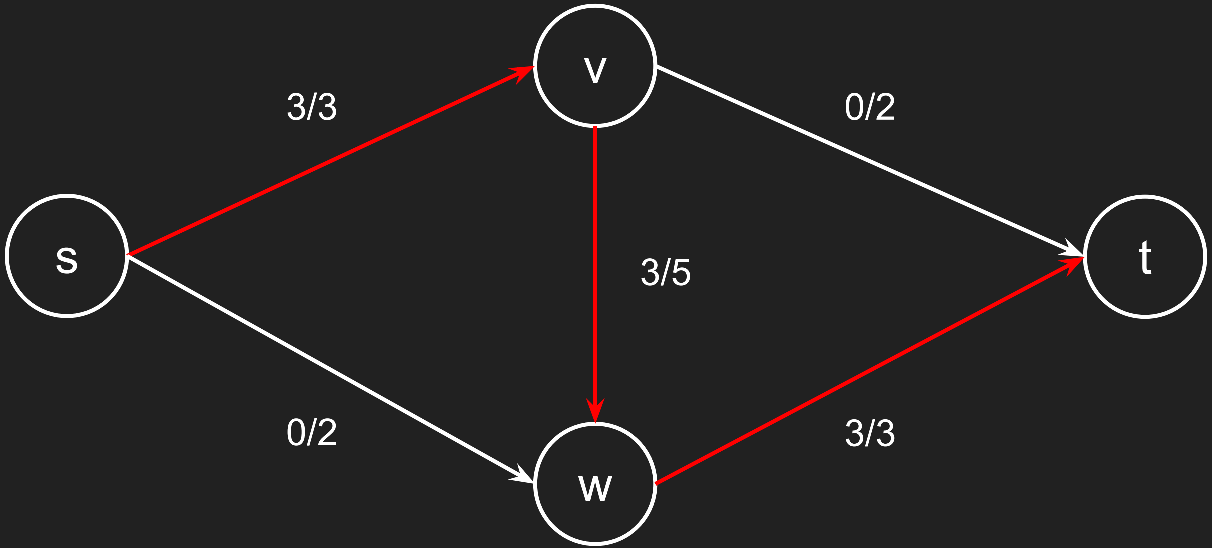
Max Flow



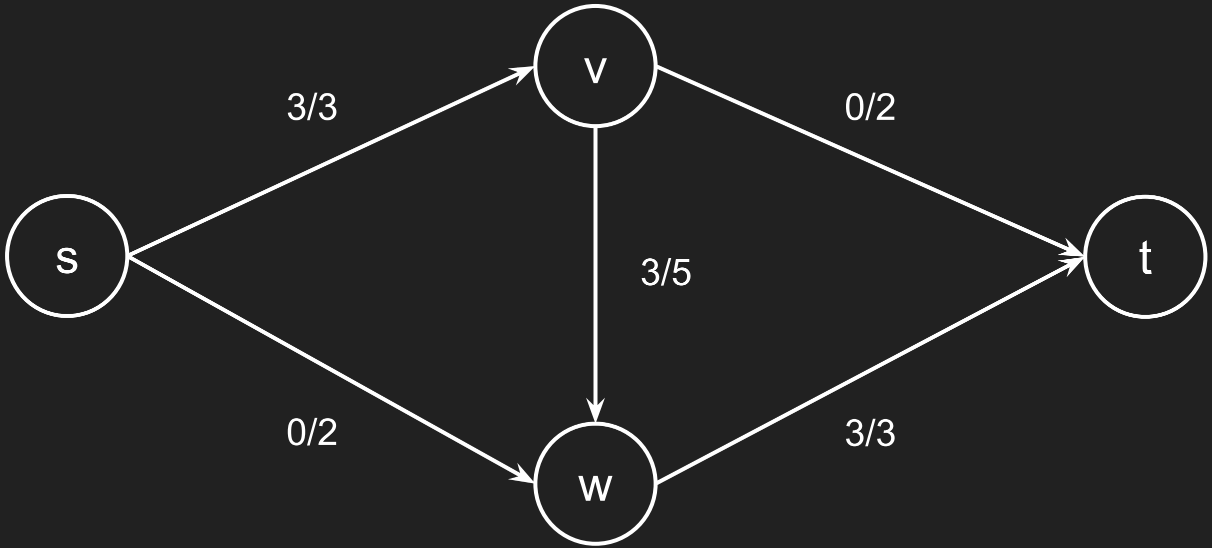
Max Flow



Max Flow

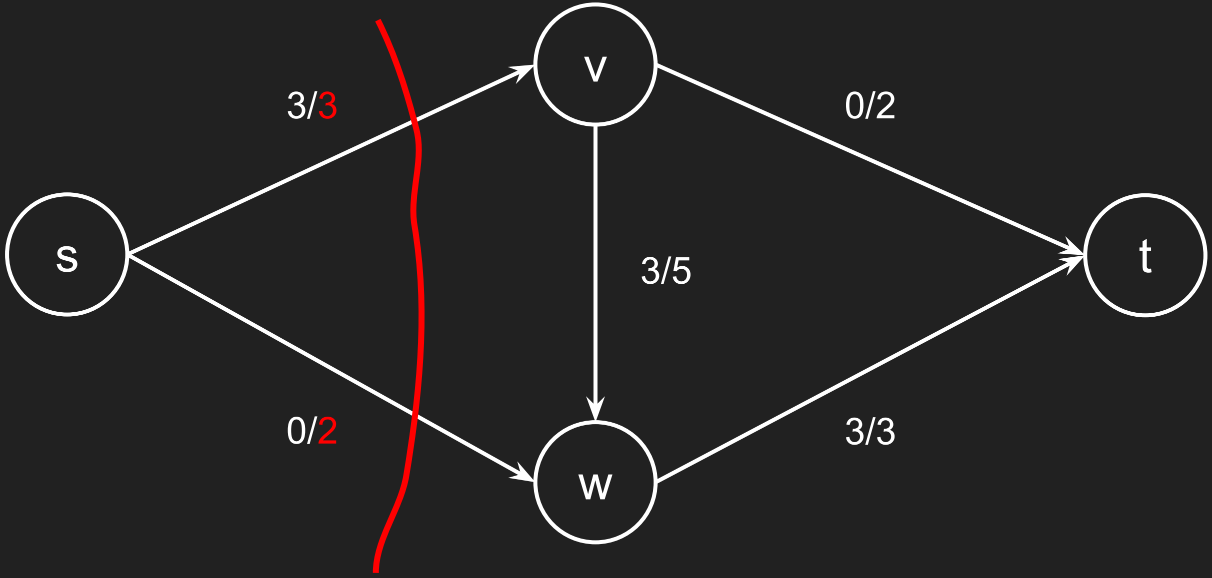


Max Flow



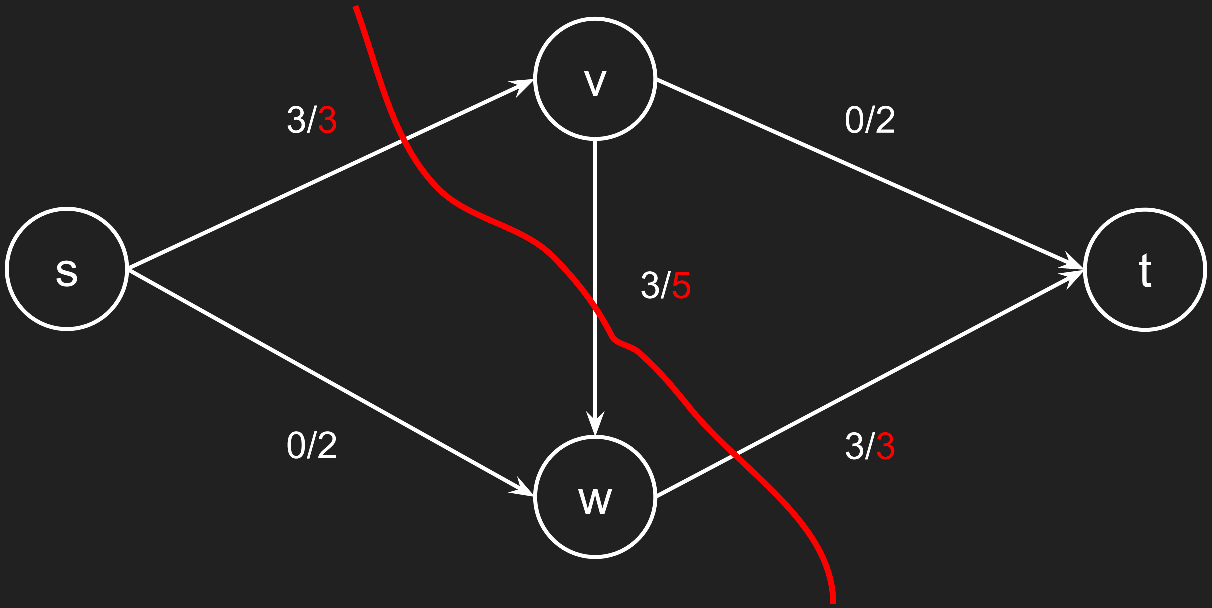
Max flow = 3 ?

Max Flow



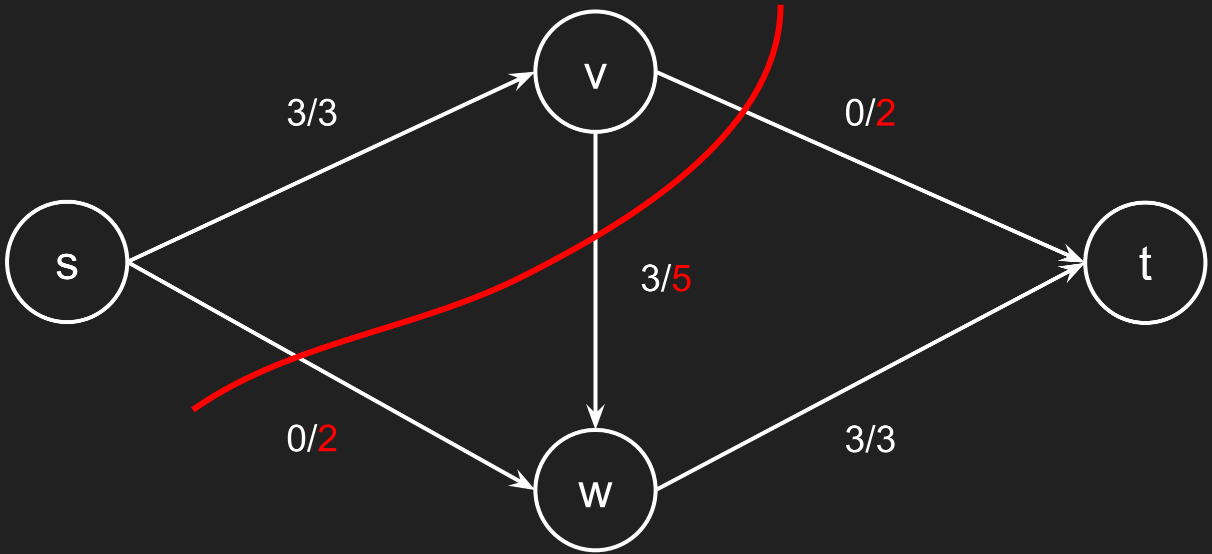
Max flow = 3 ?

Max Flow



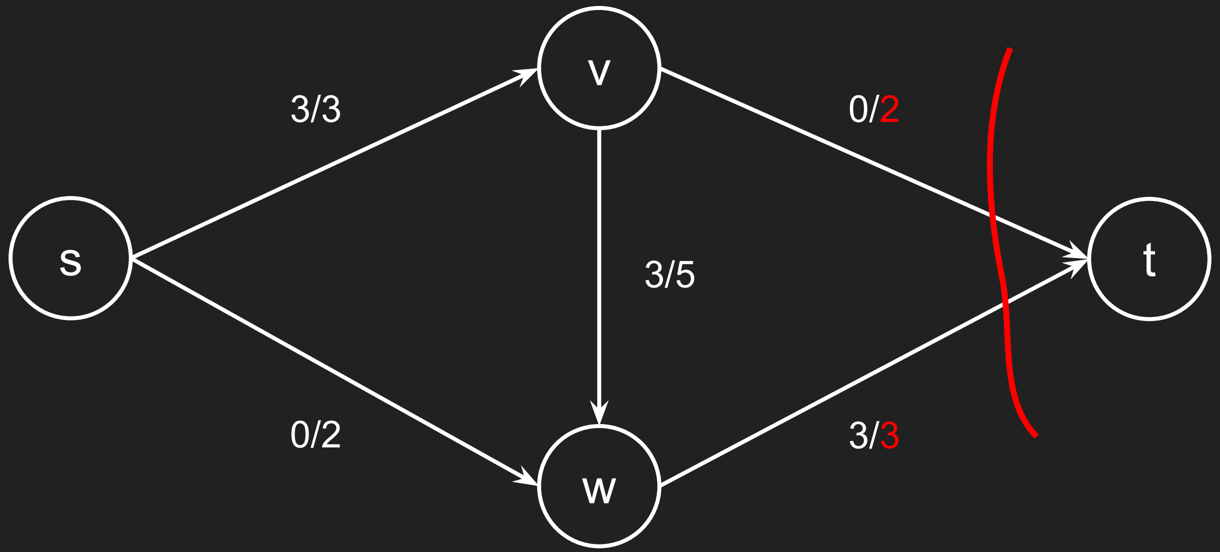
Max flow = 3 ?

Max Flow



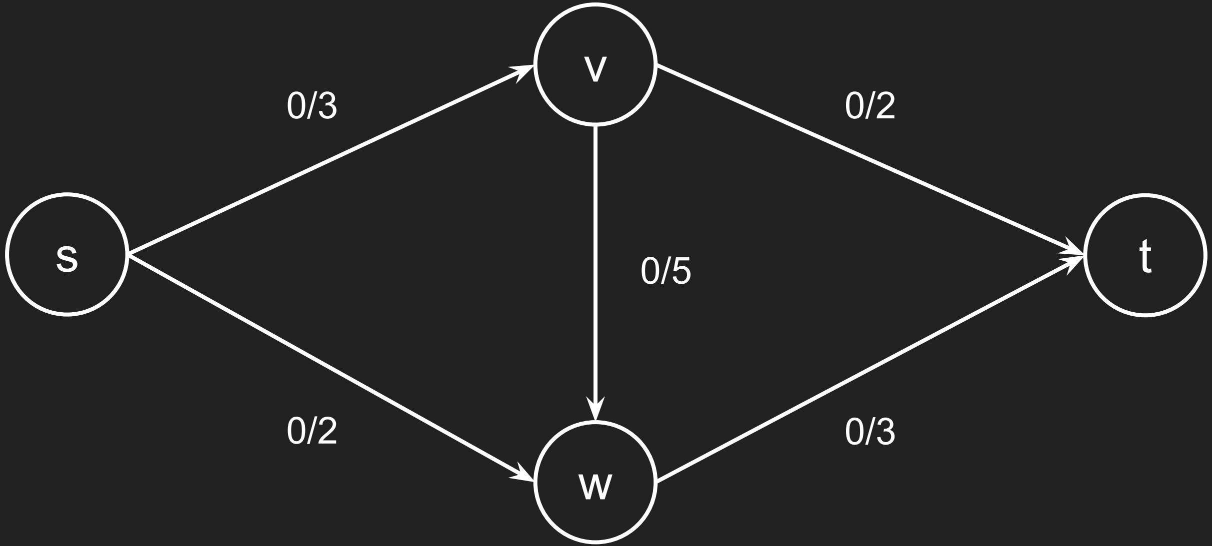
Max flow = 3 ?

Max Flow

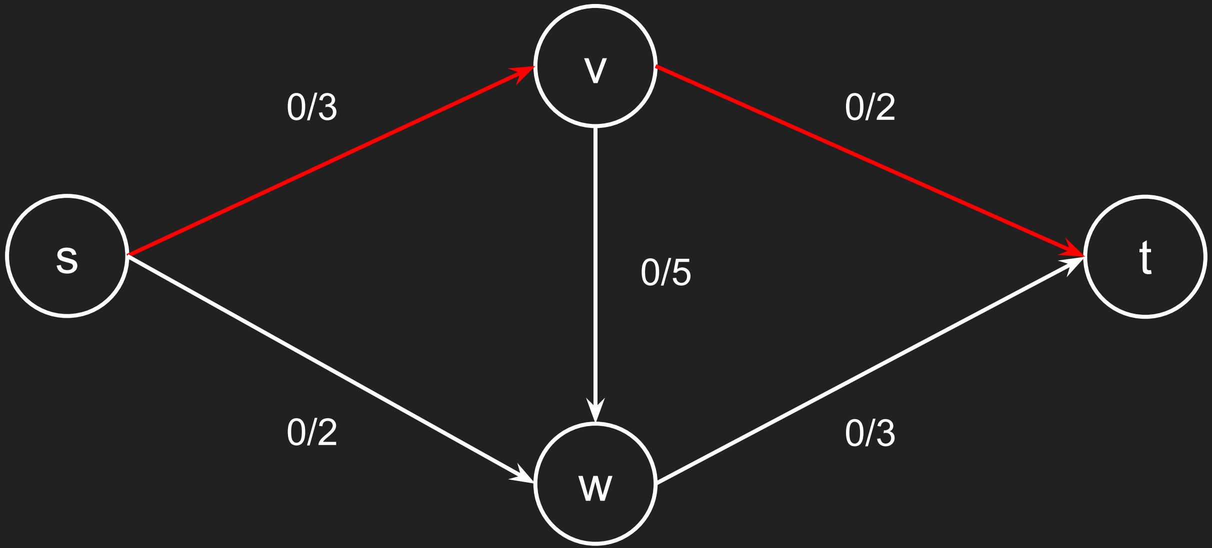


Max flow = 3 ?

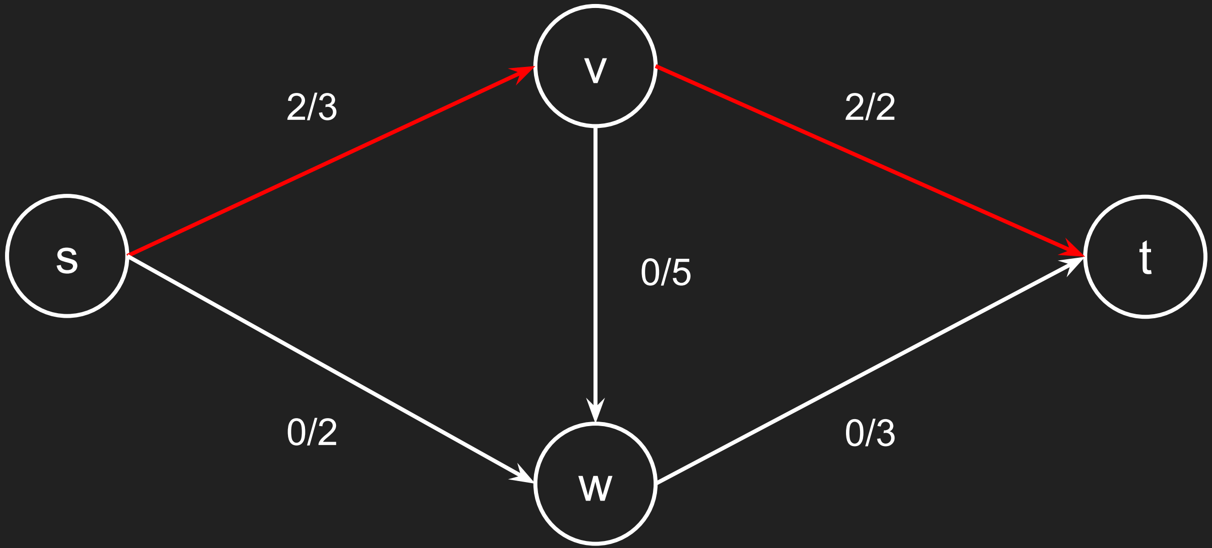
Max Flow



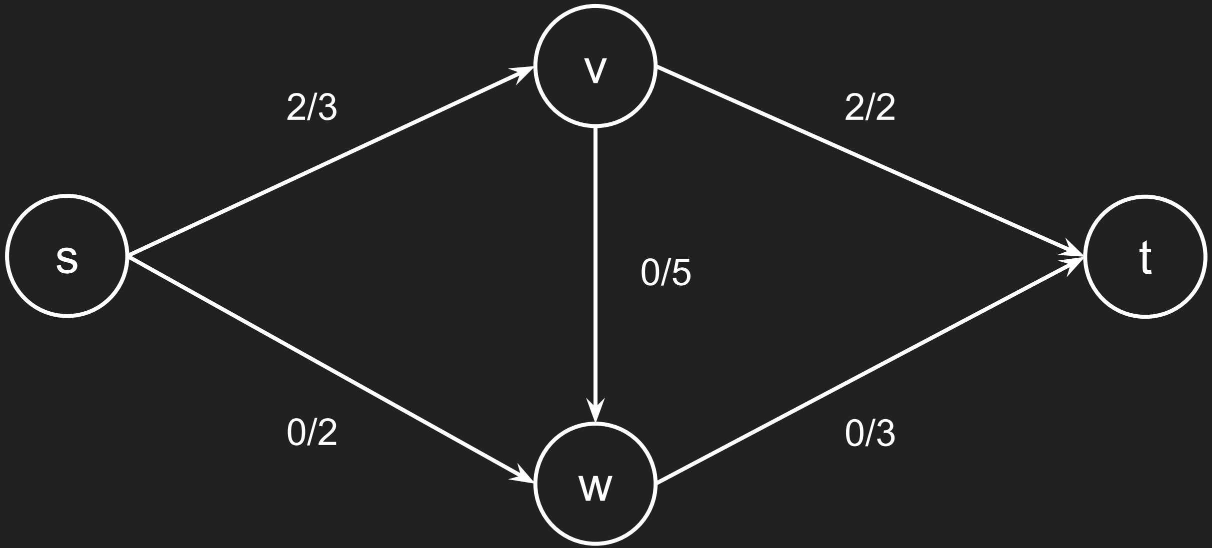
Max Flow



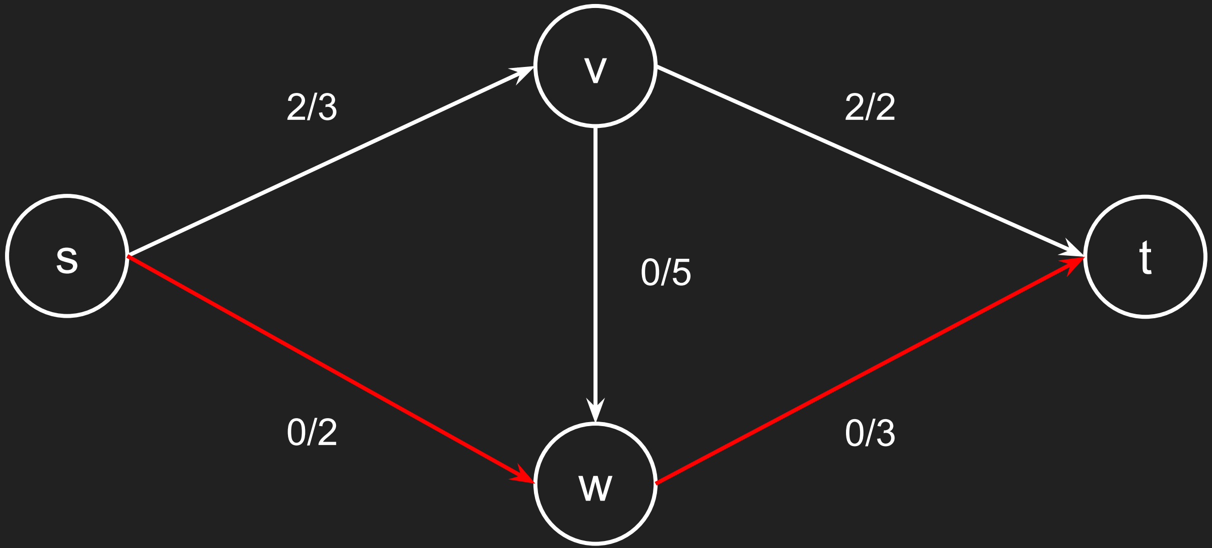
Max Flow



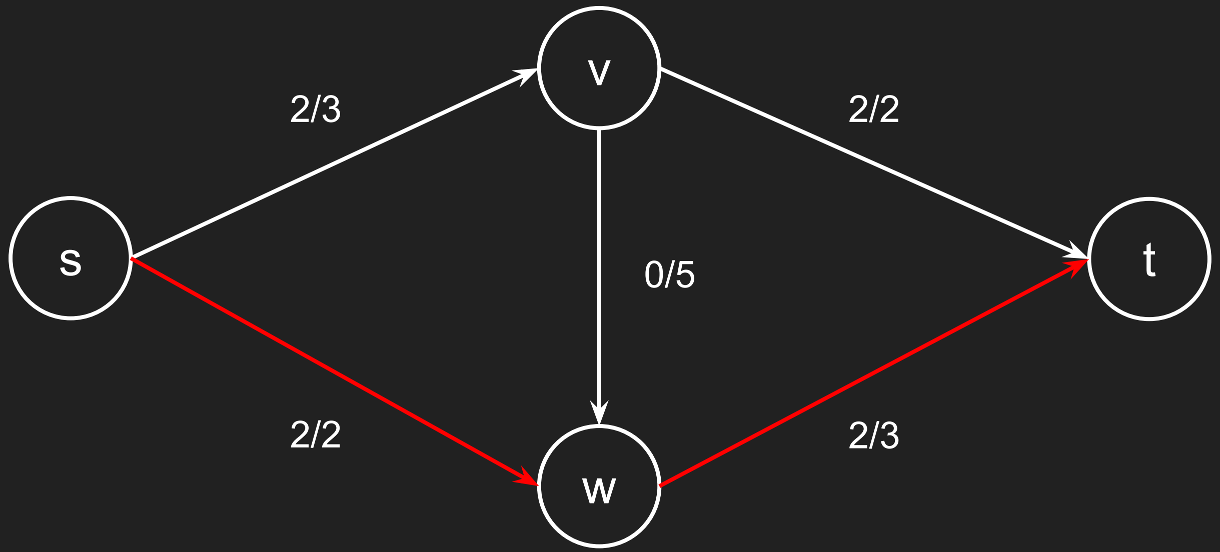
Max Flow



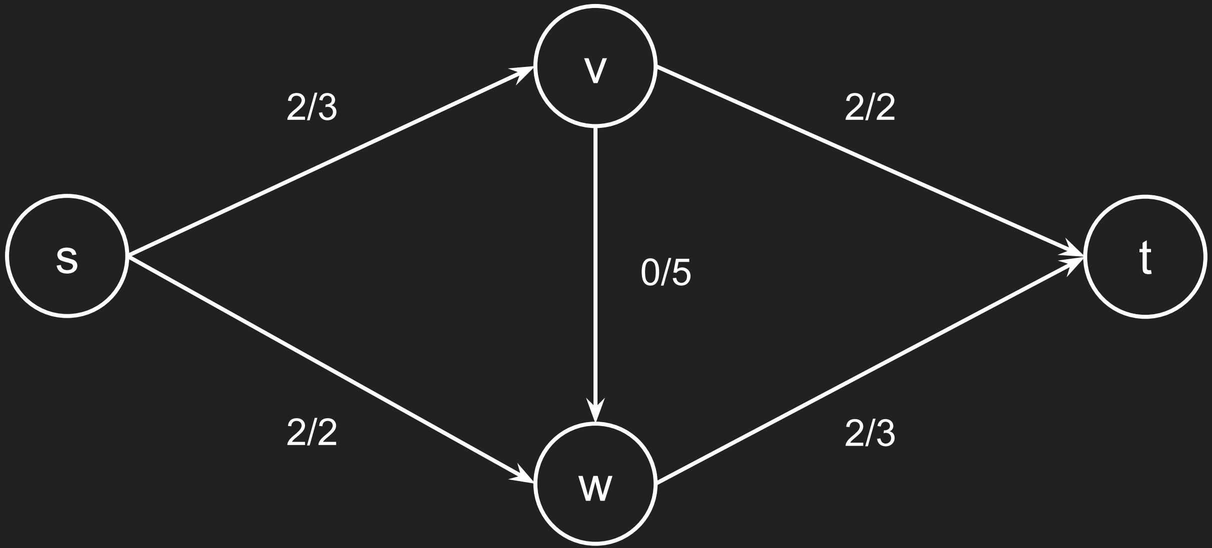
Max Flow



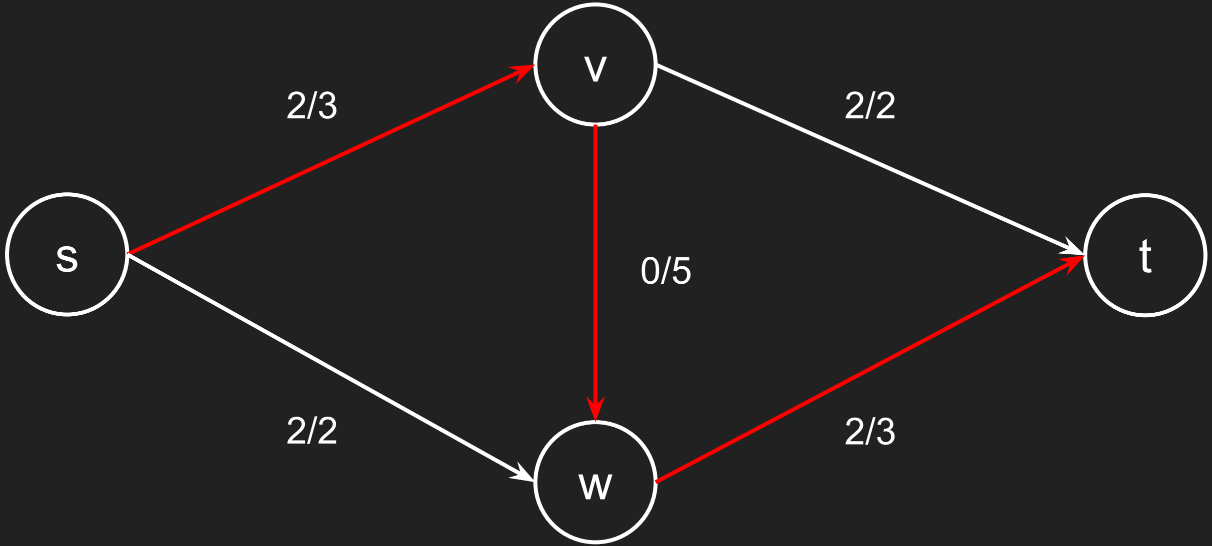
Max Flow



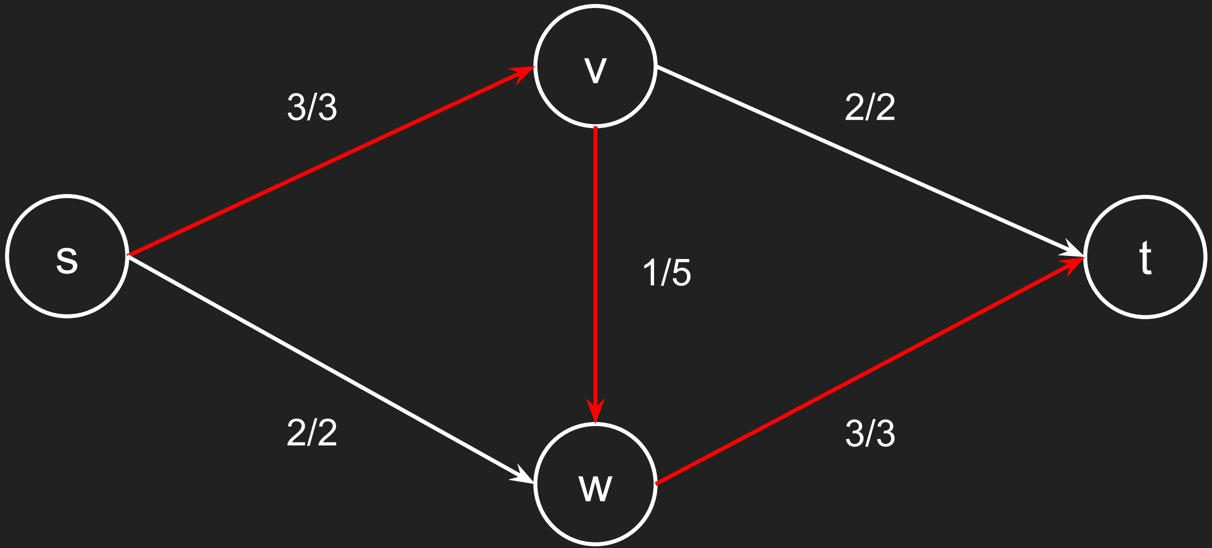
Max Flow



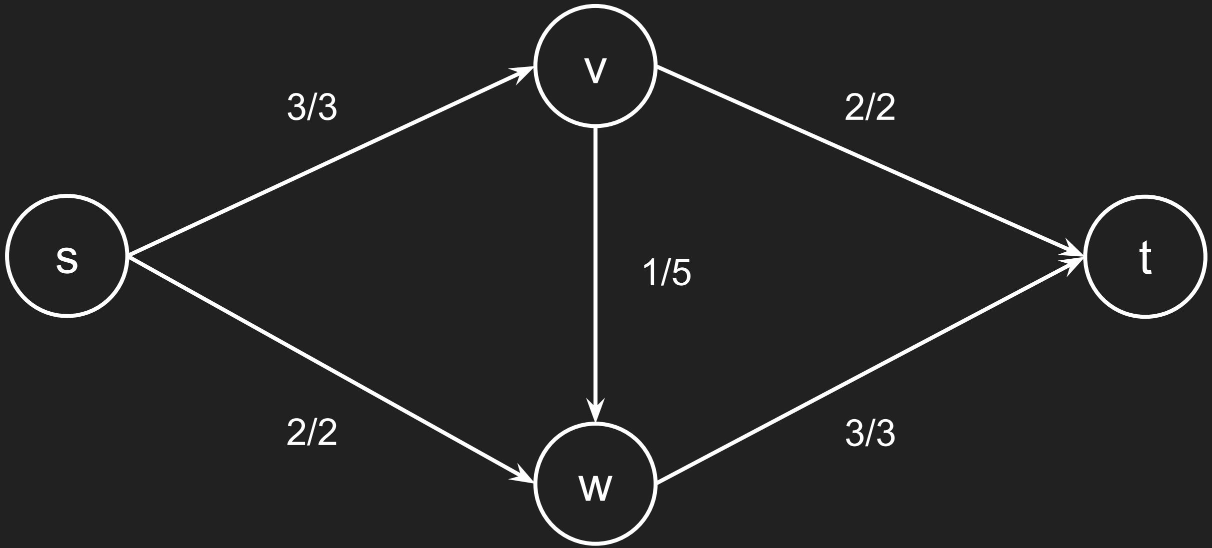
Max Flow



Max Flow



Max Flow

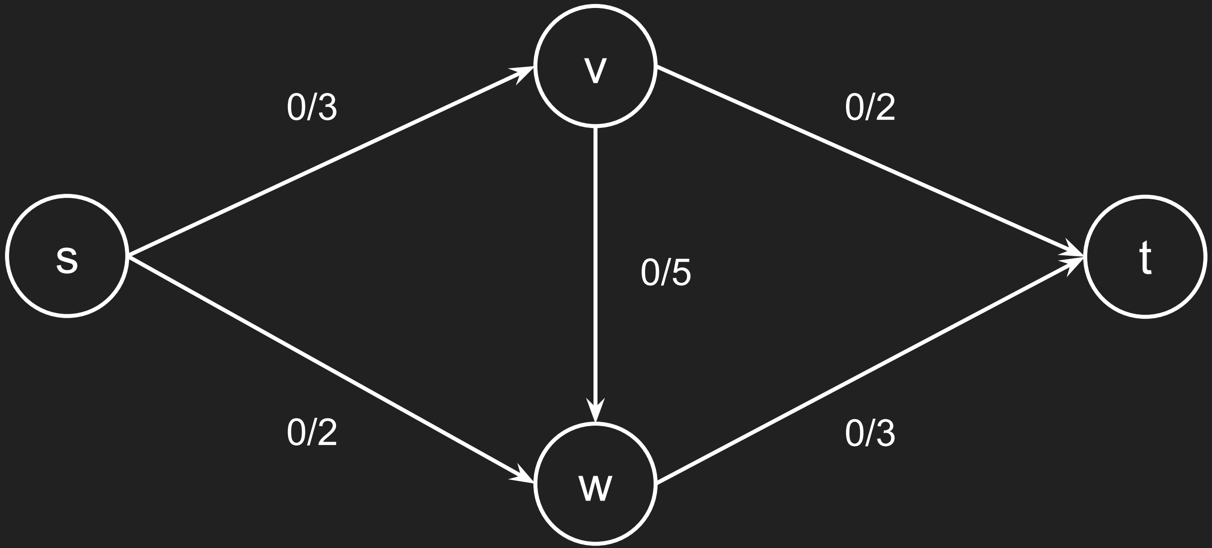


Max flow = 5 !

Max Flow

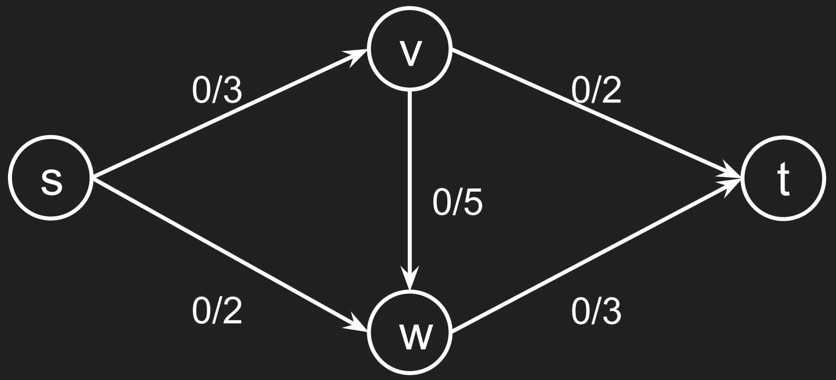
- Being greedy isn't quite good enough
  - At least, the order you pick the paths seems to matter
- Key idea: just because we *can* saturate a path doesn't necessarily mean we *should*
  - We want to be able to "undo" choices if it turns out they boxed us in a corner
- Solution: "residual graphs"
  - Augment the graph with information that allows algorithm to "undo" or "push flow back"

Max Flow

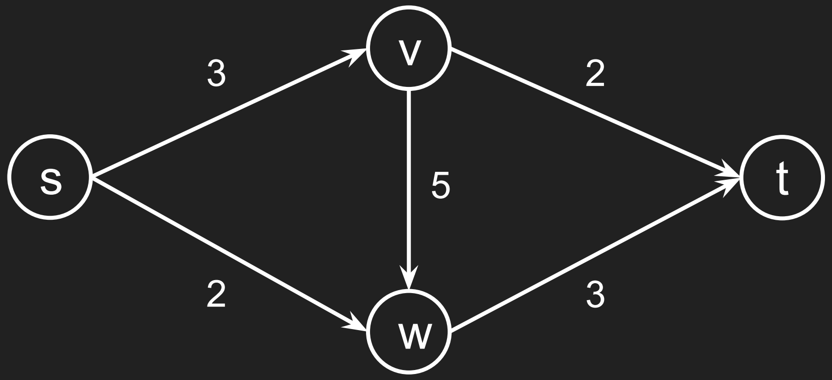


Max Flow

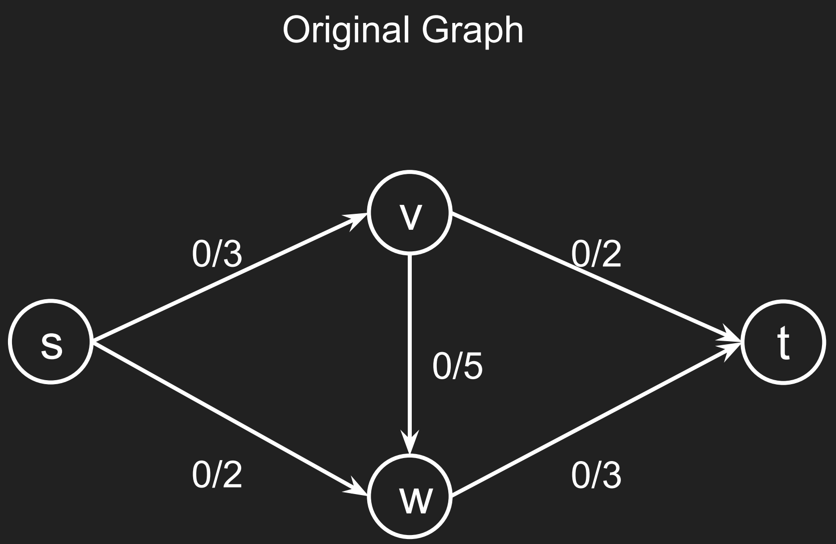
Original Graph

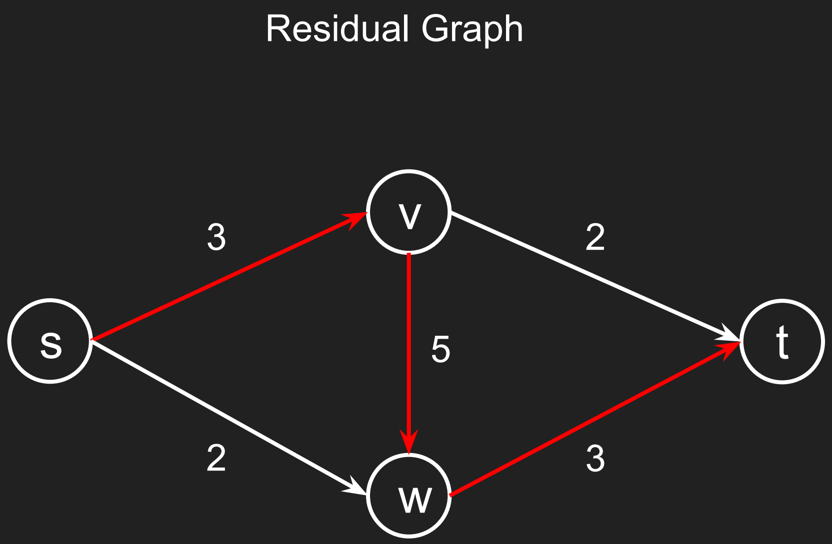


Residual Graph



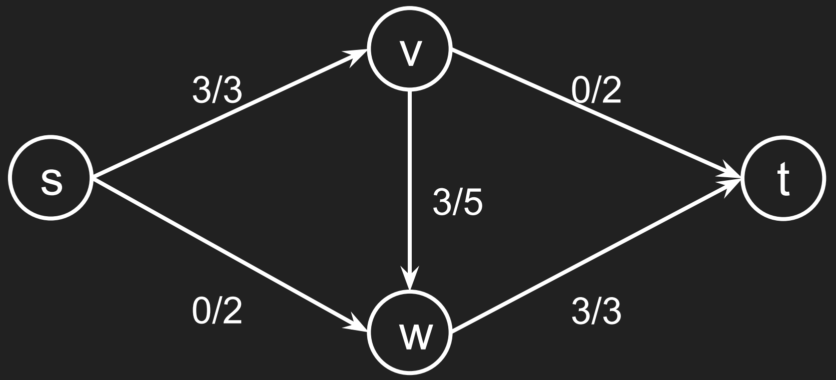
Max Flow



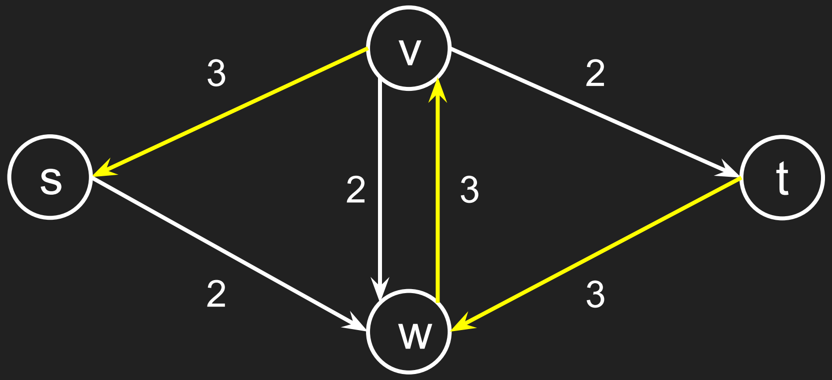


Max Flow

Original Graph

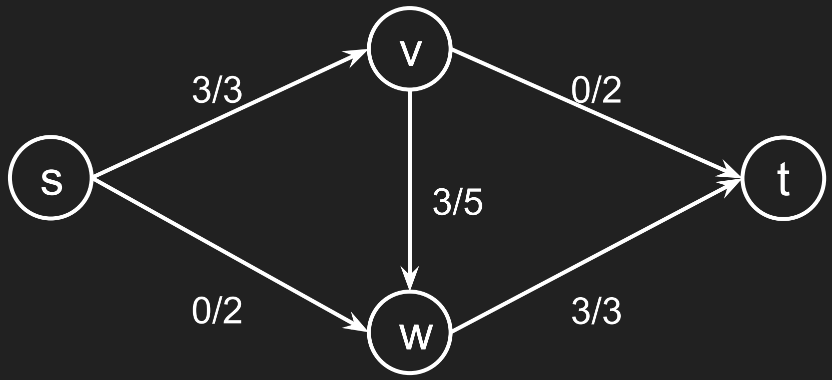


Residual Graph

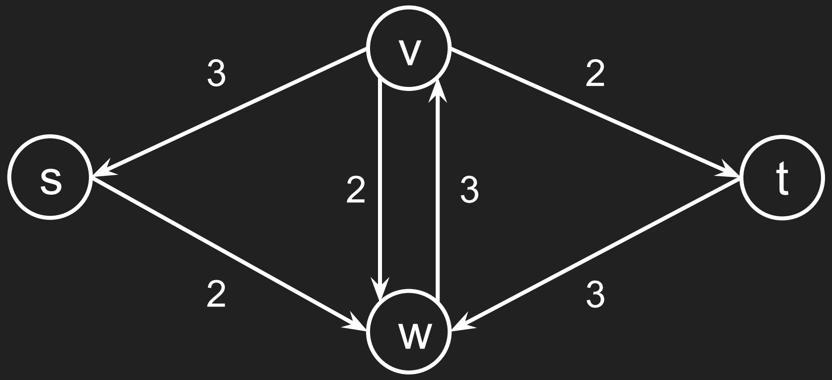


Max Flow

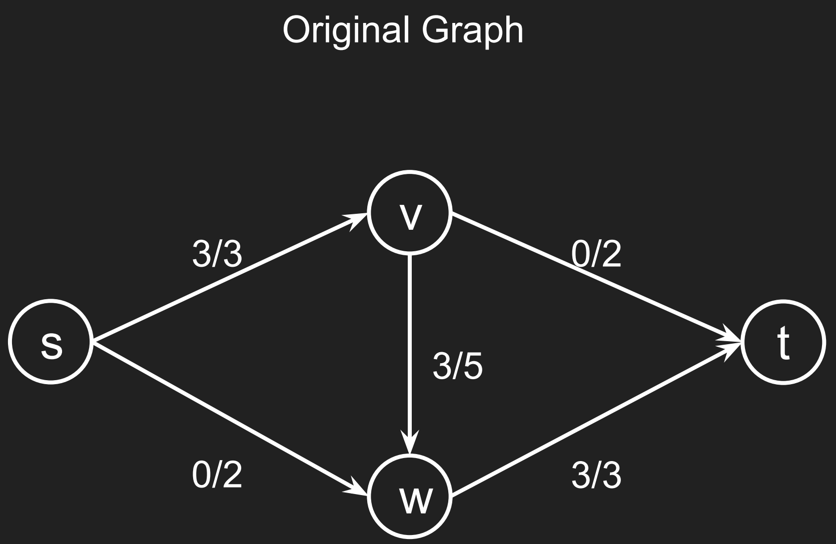
Original Graph

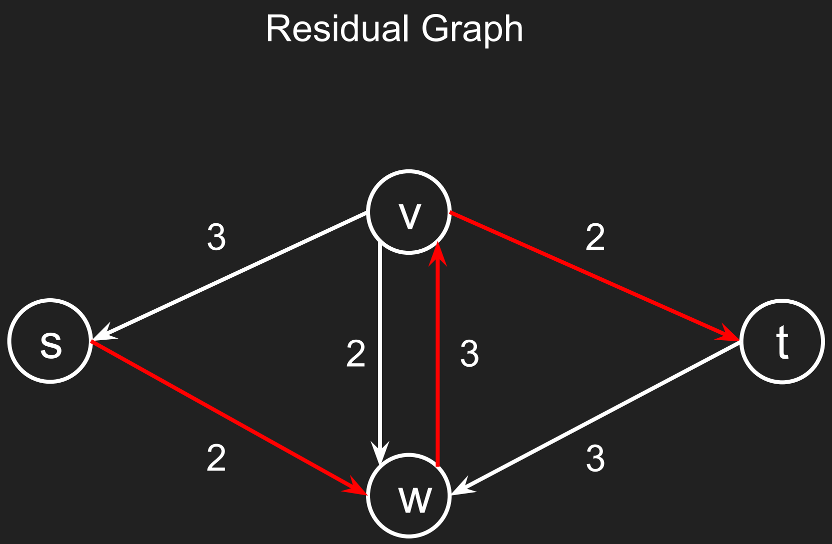


Residual Graph



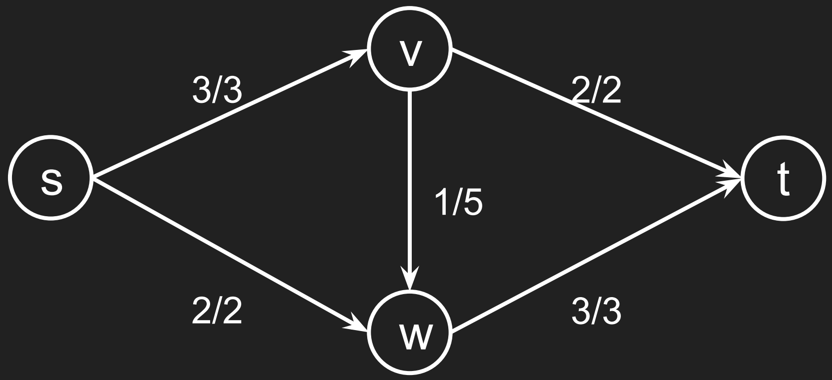
Max Flow


Original Graph
s
v
w
t
3/3
0/2
3/5
0/2
3/3

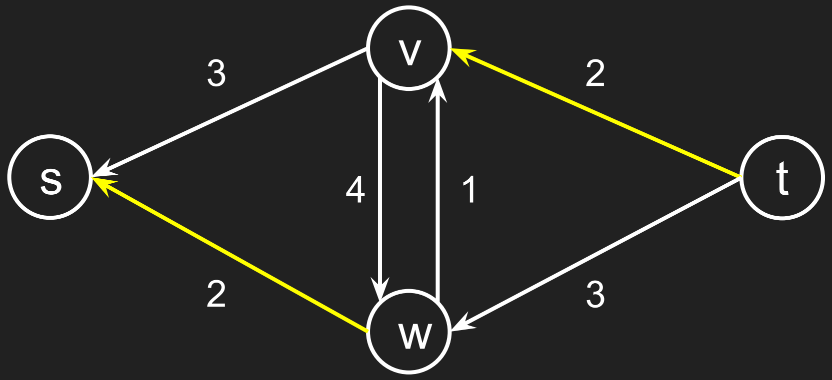

Residual Graph
s
v
w
t
3
2
2
3
2
3

Max Flow

Original Graph

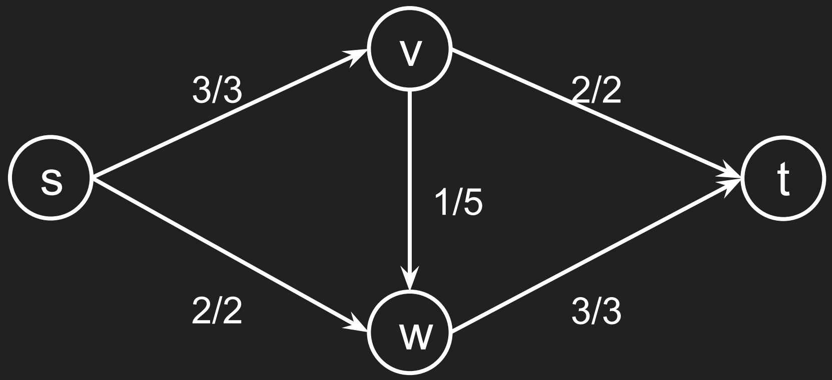


Residual Graph

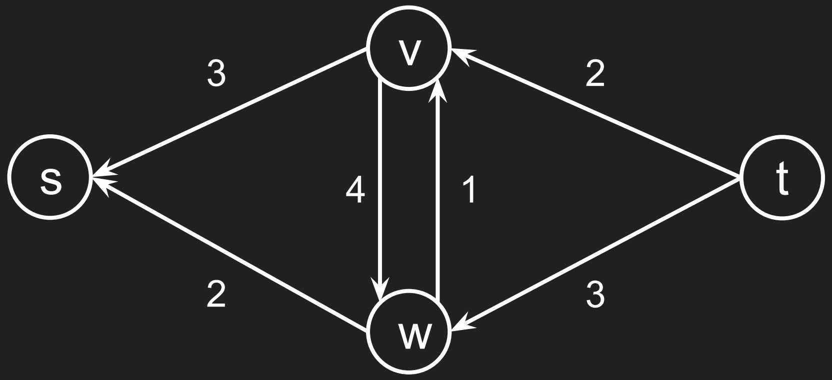


Max Flow

Original Graph



Residual Graph



Max Flow: Ford-Fulkerson Method

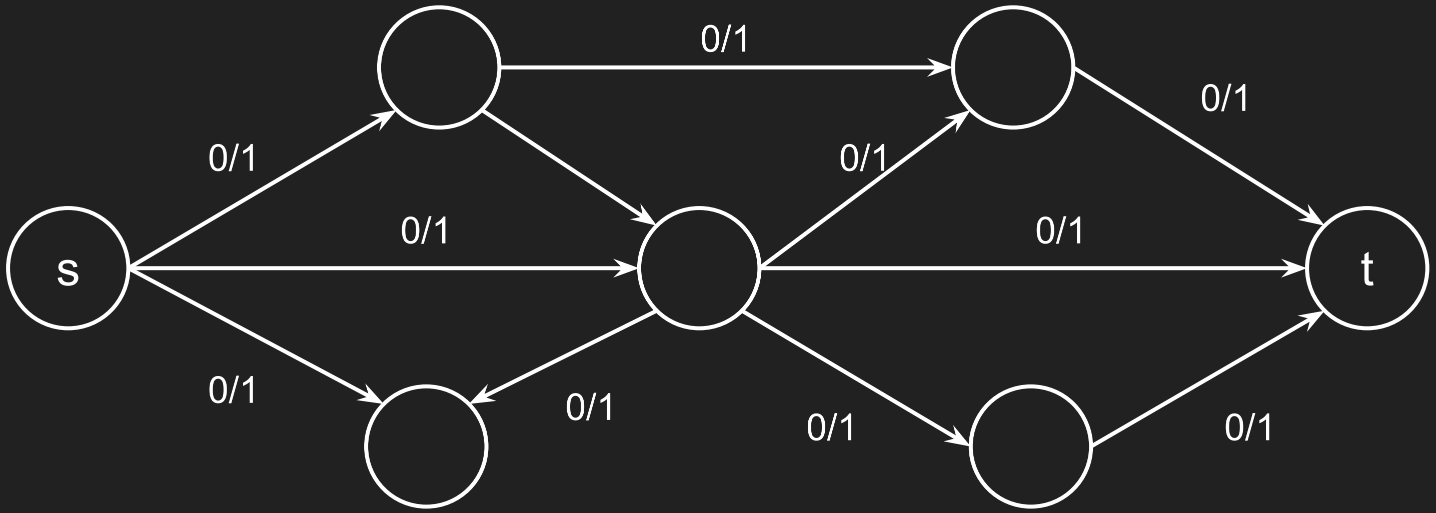
- Given graph G, define G_R (the residual graph) to be the same graph but with only capacity (no flow) and capacities denoted by $c_R(e)$.
- Maintain that for any edge (u, v) in G that $c_R(u, v) + c_R(v, u) = c(u, v)$
  - Called “skew symmetry”
- While there’s a path P from s to t in G_R:
  - Find the minimum capacity c_R among all edges in P, call it m
  - For each edge (u, v) in P:
    - If (u, v) in G, update $f(u, v)$ += m
    - Otherwise, (v, u) is in G so update $f(v, u)$ -= m
    - Update $c_R(u, v)$ and $c_R(v, u)$ accordingly to match remaining capacity in G and maintain skew symmetry
- Called a “method” because the path finding mechanism is not explicitly defined
  - If you use Breadth-First search, it’s called the Edmonds-Karp algorithm

Max Flow: Applications

- Many seemingly unrelated problems map nicely into a network flow equivalent
- Useful fact: If all edge capacities are integers, the max flow will also be an integer (and the flow along any given edge will also be an integer)
  - Known as the Integral Flow Theorem

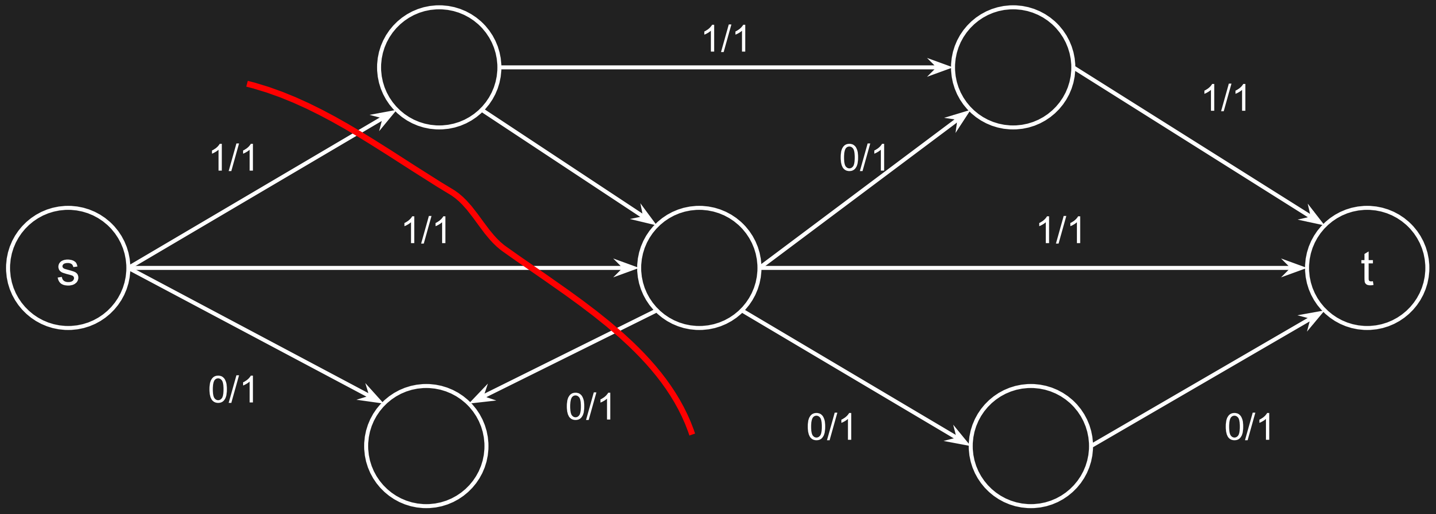
Max Flow: Network Connectivity

- You have two computers that are indirectly connected through a network of other computer systems. How many internal network disconnections is your connection resilient to?



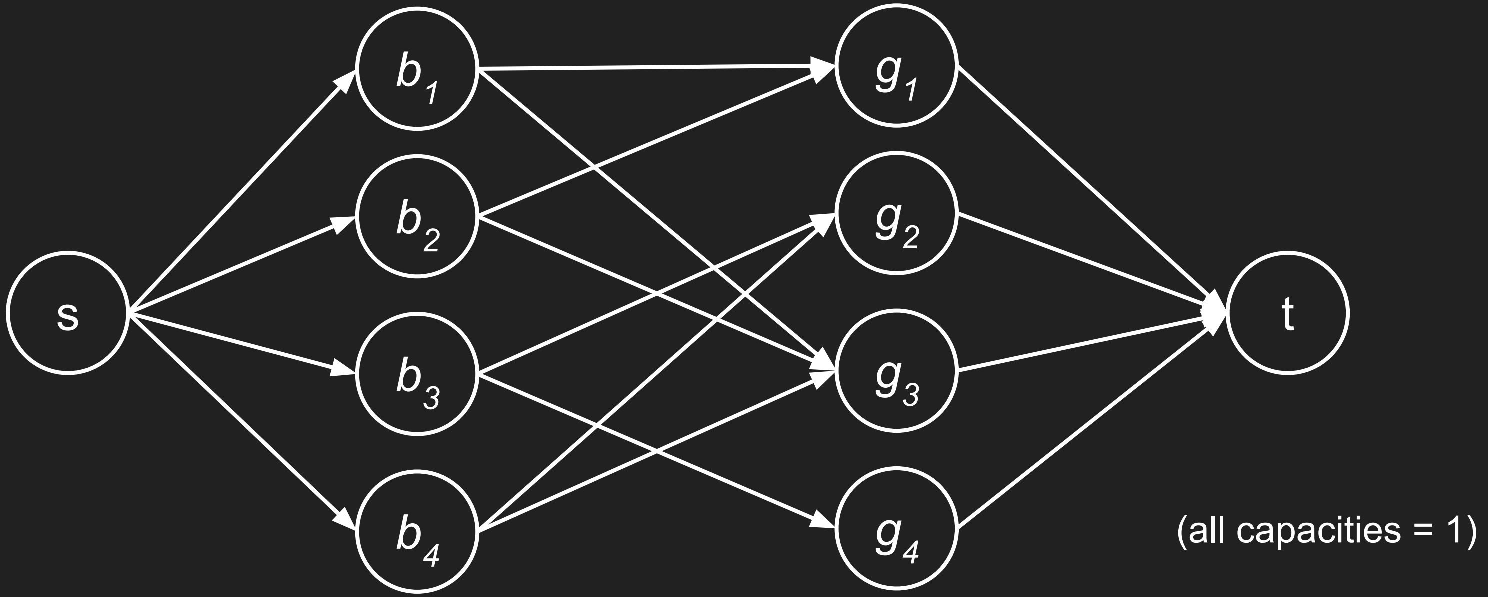
Max Flow: Network Connectivity

- You have two computers that are indirectly connected through a network of other computer systems. How many internal network disconnections is your connection resilient to?



Max Flow: School Dance

- Boys and girls need to be paired up for the school dance, but the kids only want to be paired with someone that they know. Is such a pairing possible? And if so, what’s the pairing?



Min Cut: Project Selection

- You have a set of projects p_i which will each net a revenue of $r(p_i)$. Each project will require purchasing one or more machines q_j each of which costs $c(q_j)$. Machines can be shared by multiple projects. The goal is to maximize profit.

Let P be the set of projects NOT taken, and Q be the set of machines purchased. Then we want:

$$\max \left[\sum_i r(p_i) - \sum_{pi \in P} r(p_i) - \sum_{pj \in Q} c(q_j) \right]$$

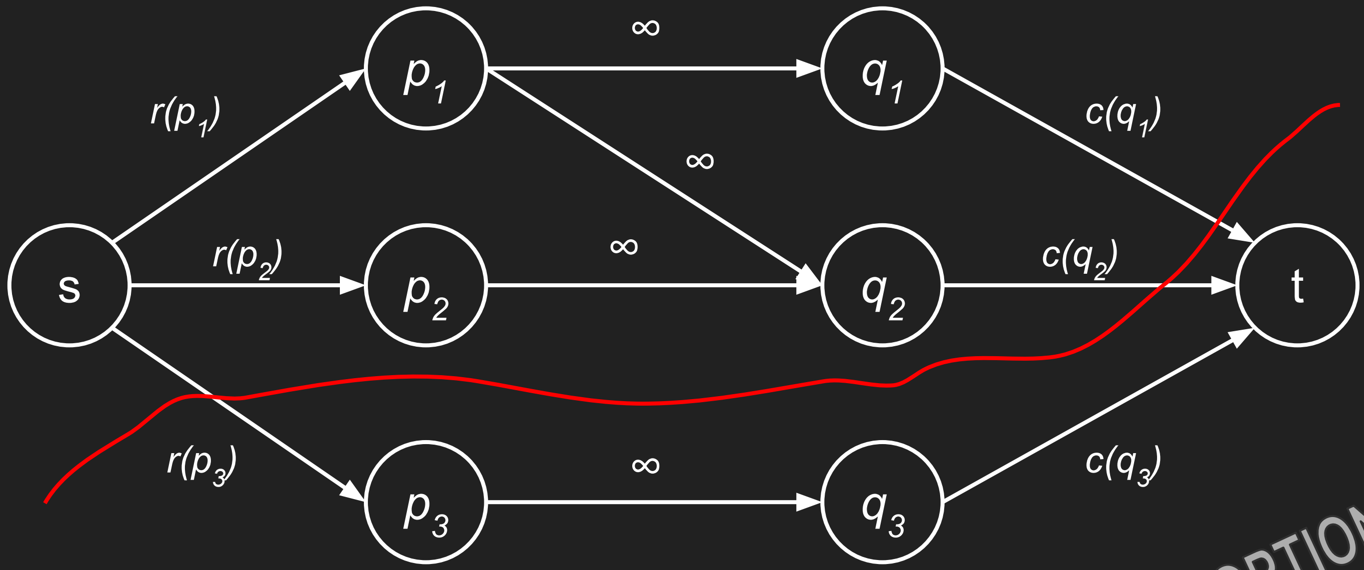
Which can be reformulated as:

$$\sum_i r(p_i) - \min \left[\sum_{pi \in P} r(p_i) + \sum_{pj \in Q} c(q_j) \right]$$

OPTIONAL MATERIAL

Min Cut: Project Selection

$$\sum_i r(p_i) - \min \left[\sum_{p_i \in P} r(p_i) + \sum_{p_j \in Q} c(q_j) \right]$$



OPTIONAL MATERIAL