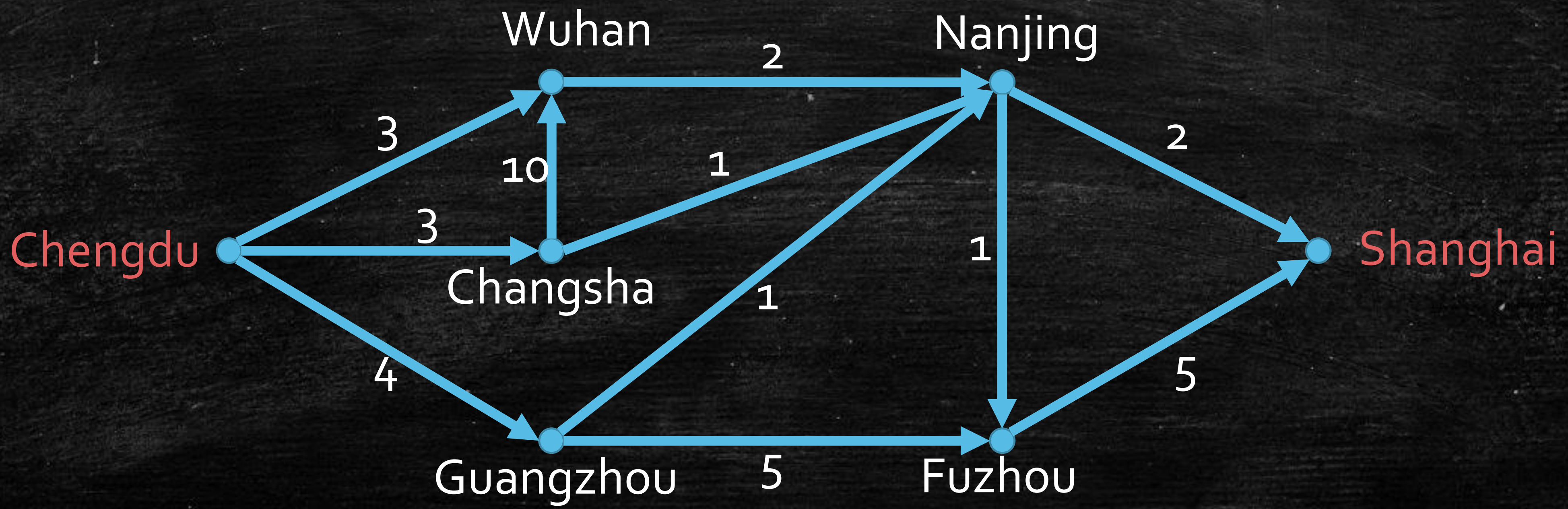
Network Flow

Maximum Flow Problem
Ford-Fulkerson Algorithm
Max-Flow-Min-Cut Theorem
Flow Integrality Theorem

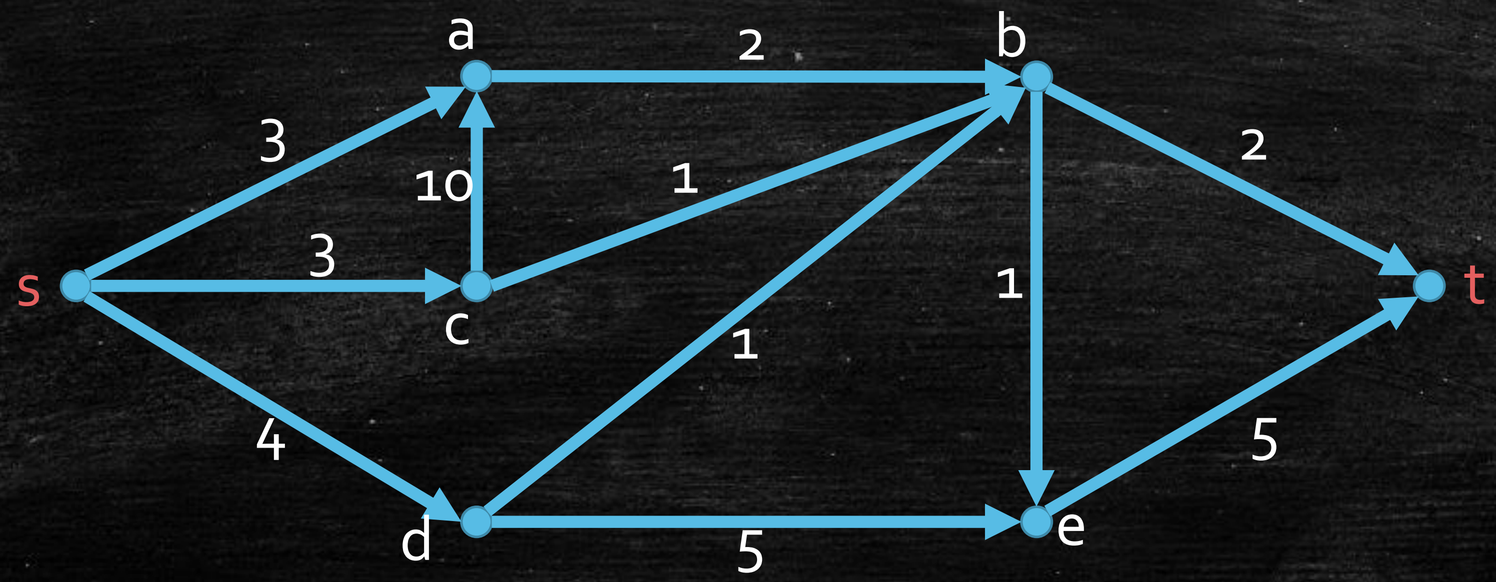
Maximum Flow Problem

- Railway system has a network of city-to-city routes.
- Each route labeled with maximum number of passengers per train.
- Question: How many passengers can we send from Chengdu to Shanghai?



Maximum Flow Problem

- We want to build a data transmission channel from s to t.
- We can use intermediate routers a, b, c, d, e.
- Each edge has a bandwidth, limiting the maximum rate of data transmission.
- What is the maximum rate of data that can be transferred?



Flow – Formal Definition

- Given a directed graph $G = (V, E)$ with a source $s \in V$ and a sink $t \in V$, and a capacity assigned to each edge $c: E \to \mathbb{R}^+$, a flow is a map $f: E \to \mathbb{R}_{\geq 0}$ satisfying the followings:
  - **Capacity Constraint**: for each $e \in E$, $f(e) \leq c(e)$, and
  - **Flow Conservation**: for each $u \in V \setminus \{s, t\}$,

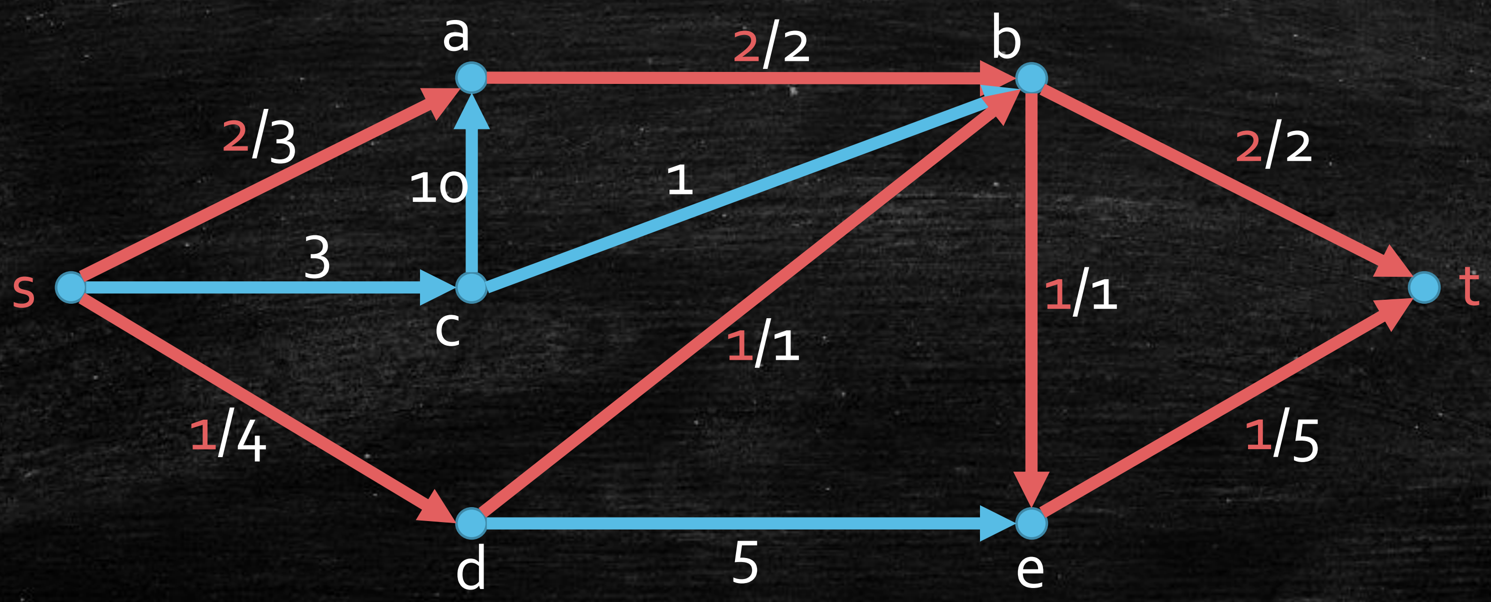
$$\sum_{v:(v,u)\in E} f(v, u) = \sum_{w:(u,w)\in E} f(u, w).$$

The value of the flow is defined as

$$v(f) = \sum_{v:(s,v)\in E} f(s, v).$$

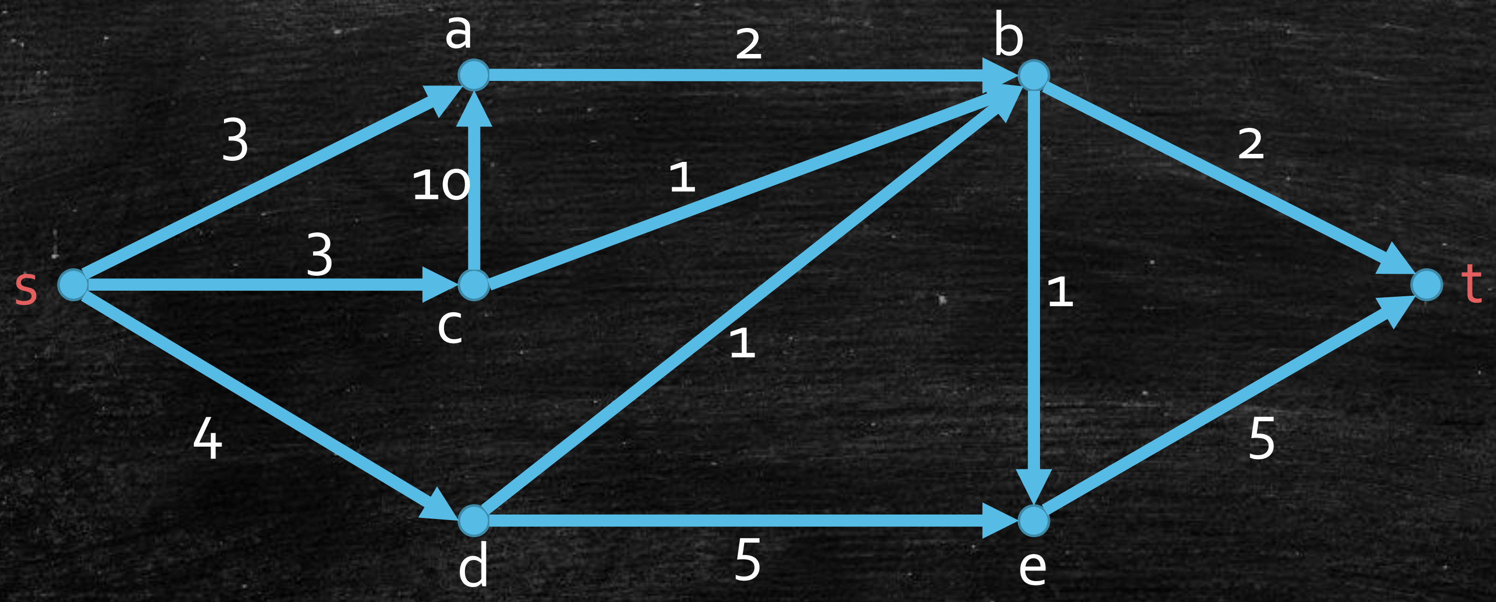
Example

- Red number: amount of flow $f(e)$ on the edge e
- Is this a valid flow?
- What is the value of this flow?
- Is it maximum?



Class Activity 1

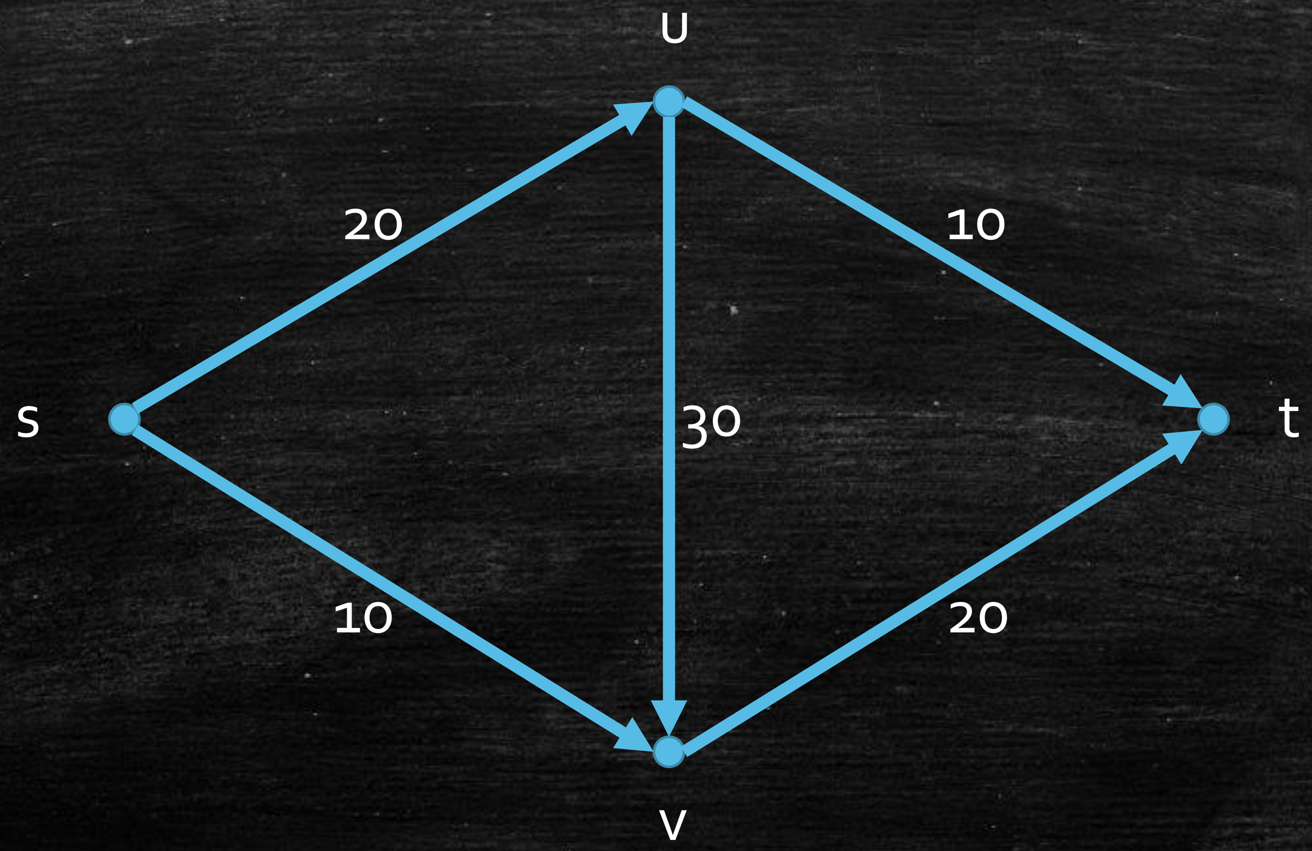
- What is the value of the maximum flow?



Ford-Fulkerson Algorithm

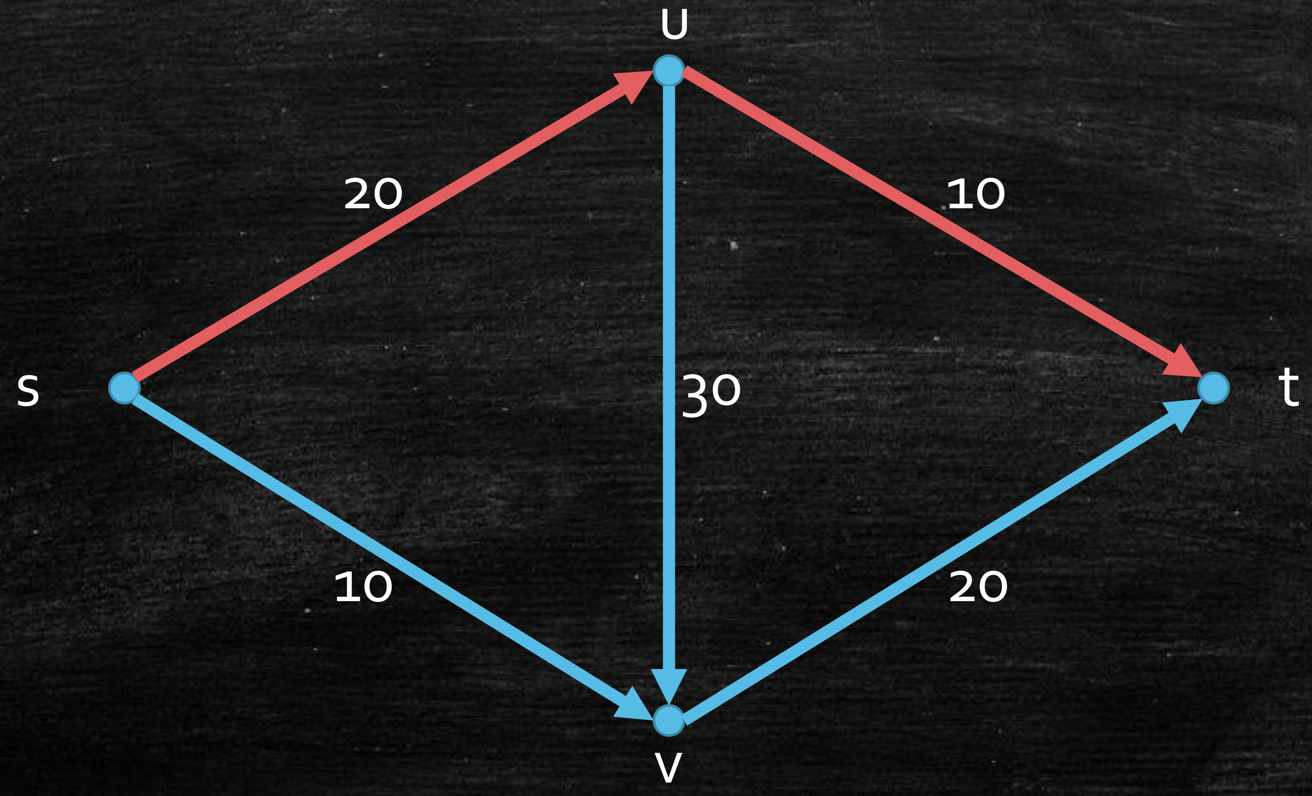
A Greedy Attempt

- Iteratively find an s-t path and push as much flow as possible along it.



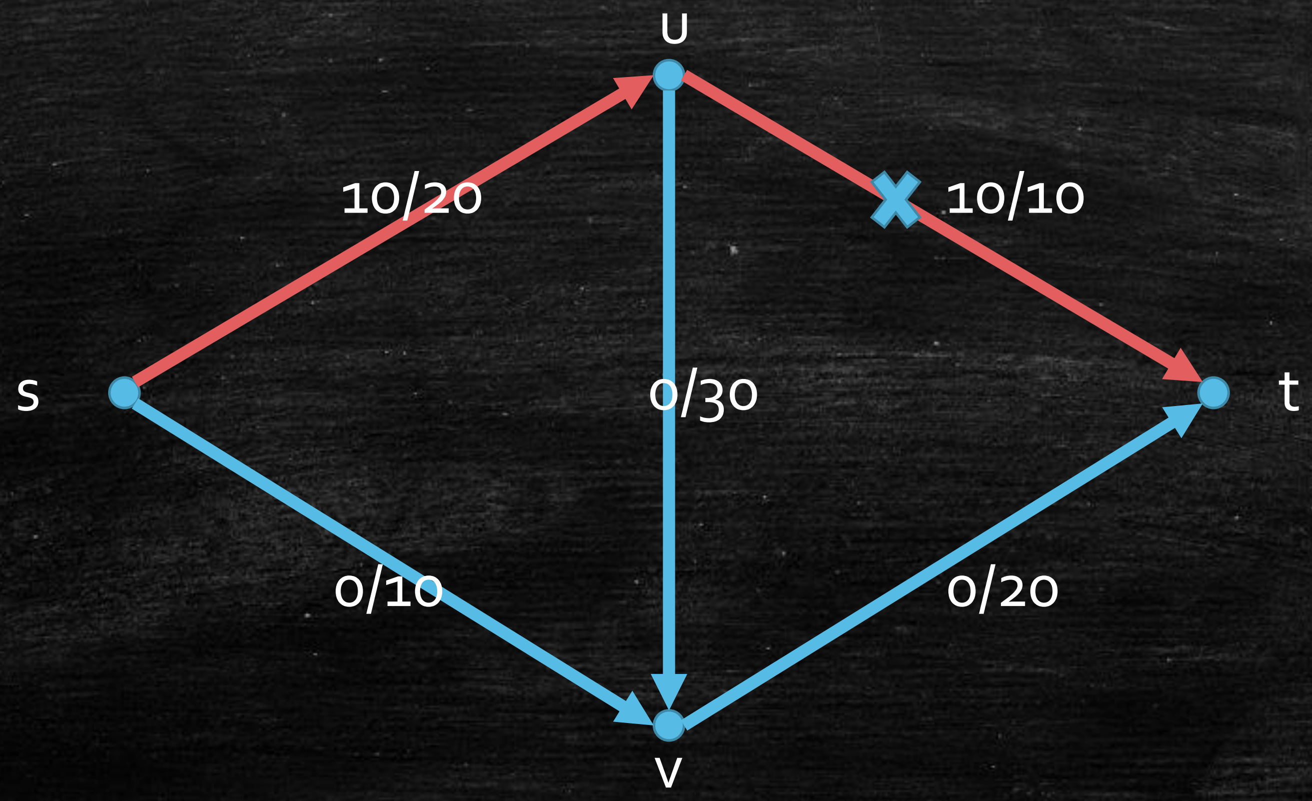
A Greedy Attempt

- Iteratively find an s-t path and push as much flow as possible along it.
  - s-u-t



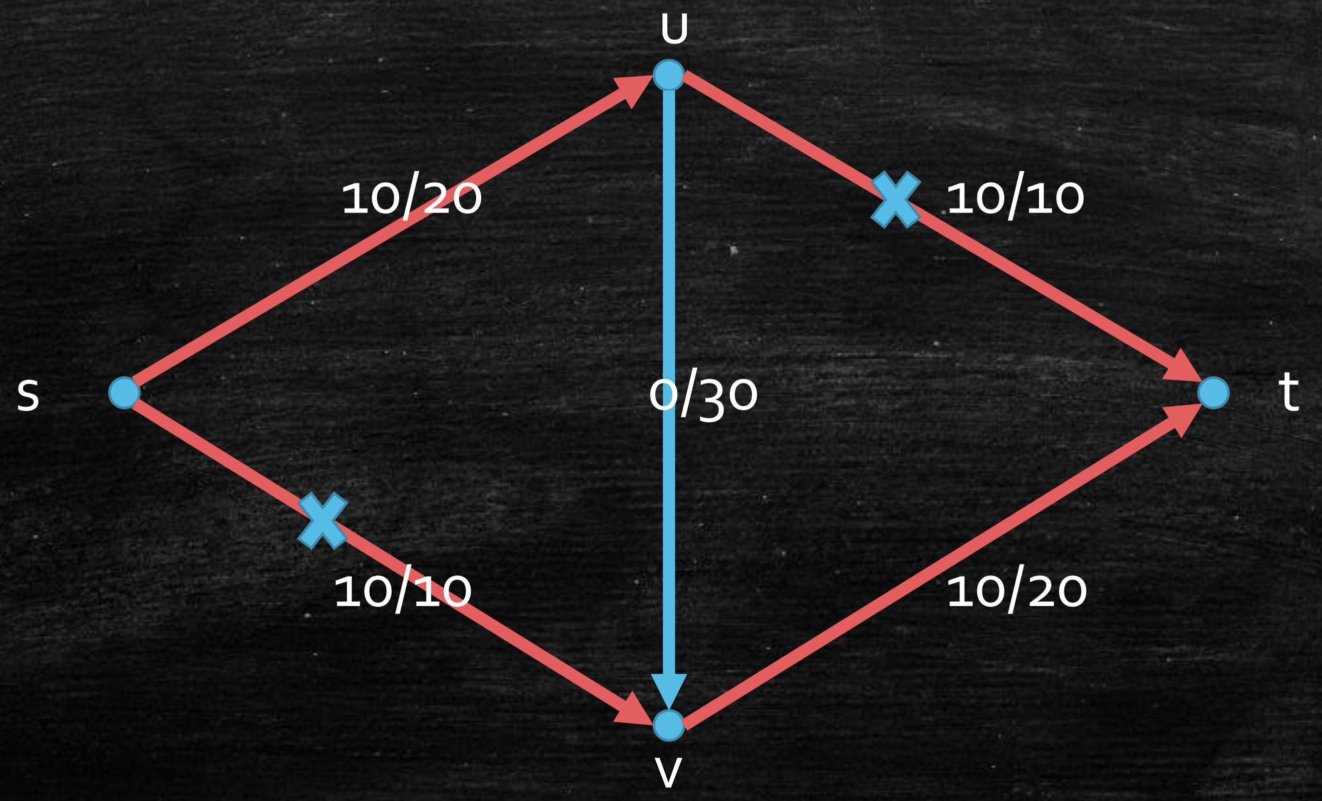
A Greedy Attempt

- Iteratively find an s-t path and push as much flow as possible along it.
  - s-u-t



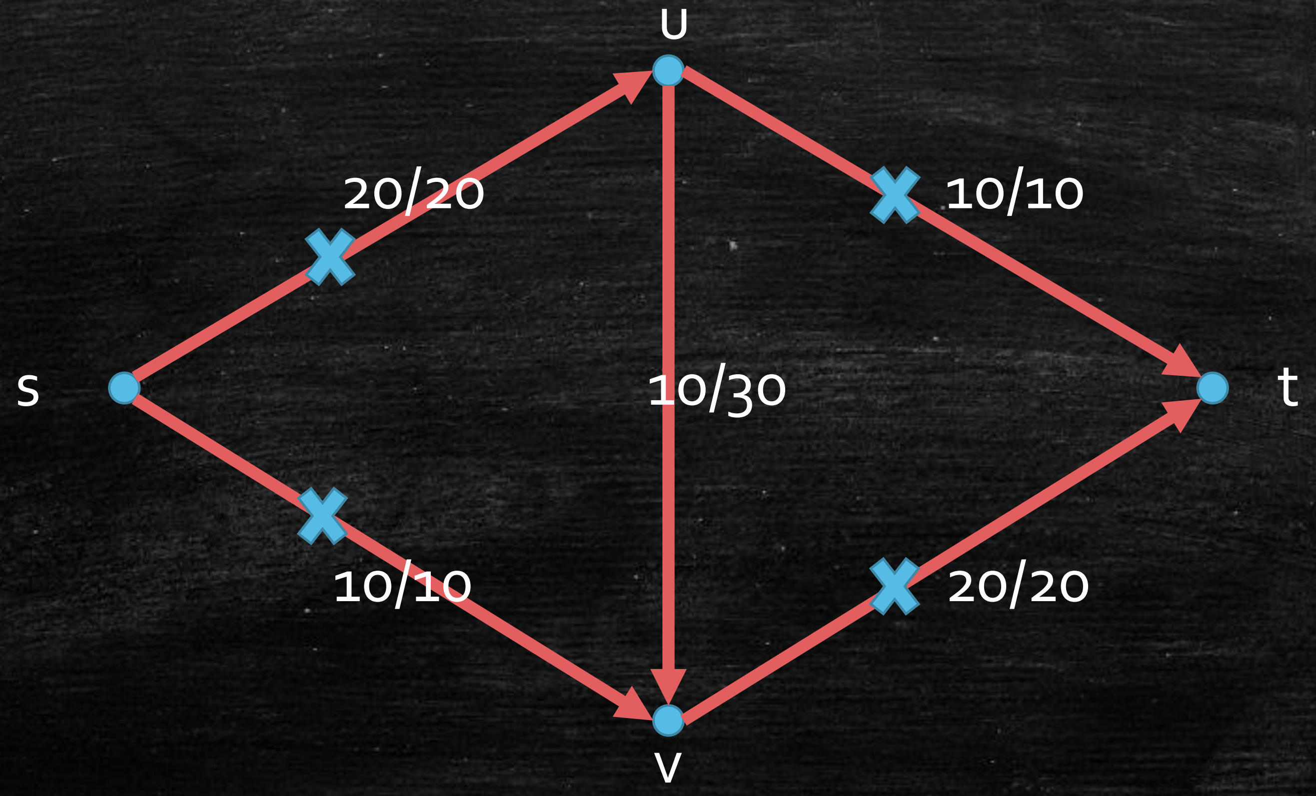
A Greedy Attempt

- Iteratively find an s-t path and push as much flow as possible along it.
  - s-u-t, s-v-t



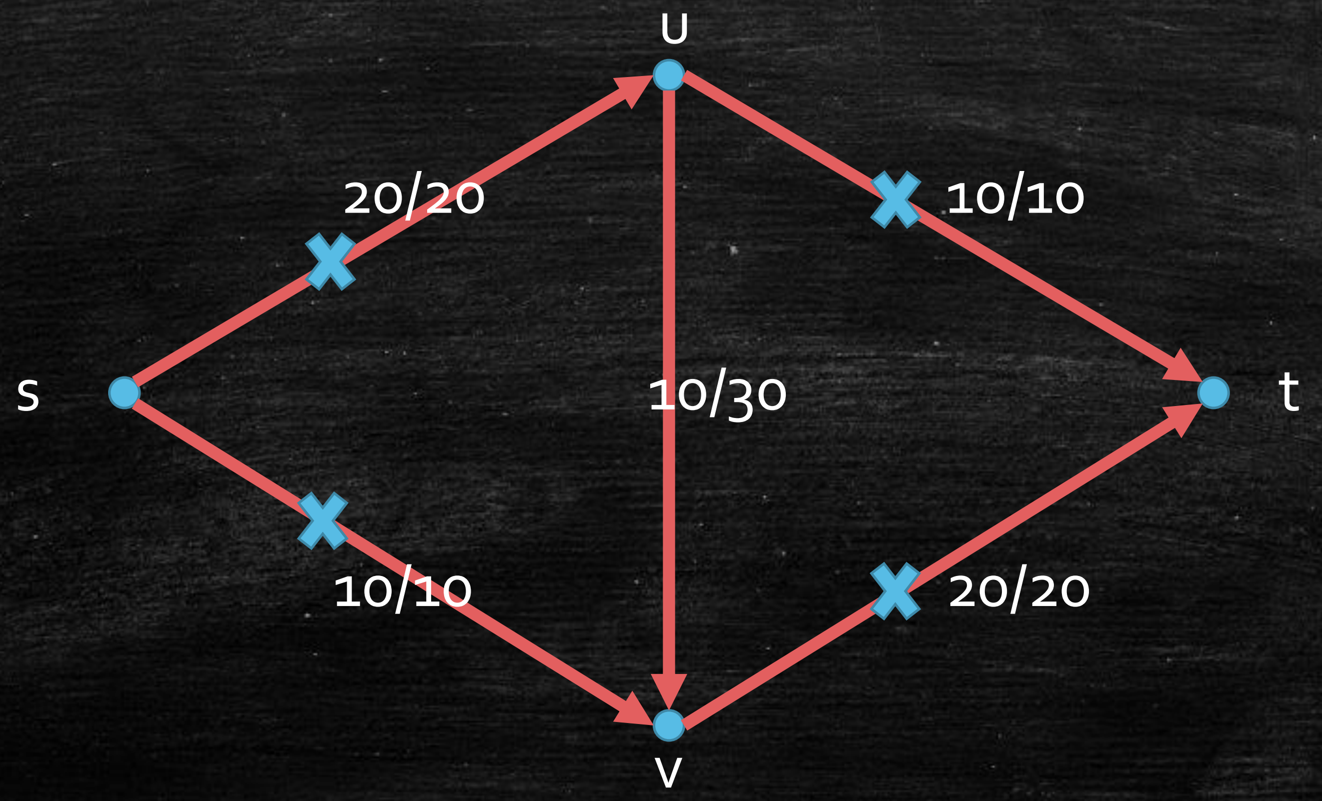
A Greedy Attempt

- Iteratively find an s-t path and push as much flow as possible along it.
  - s-u-t, s-v-t, s-u-v-t



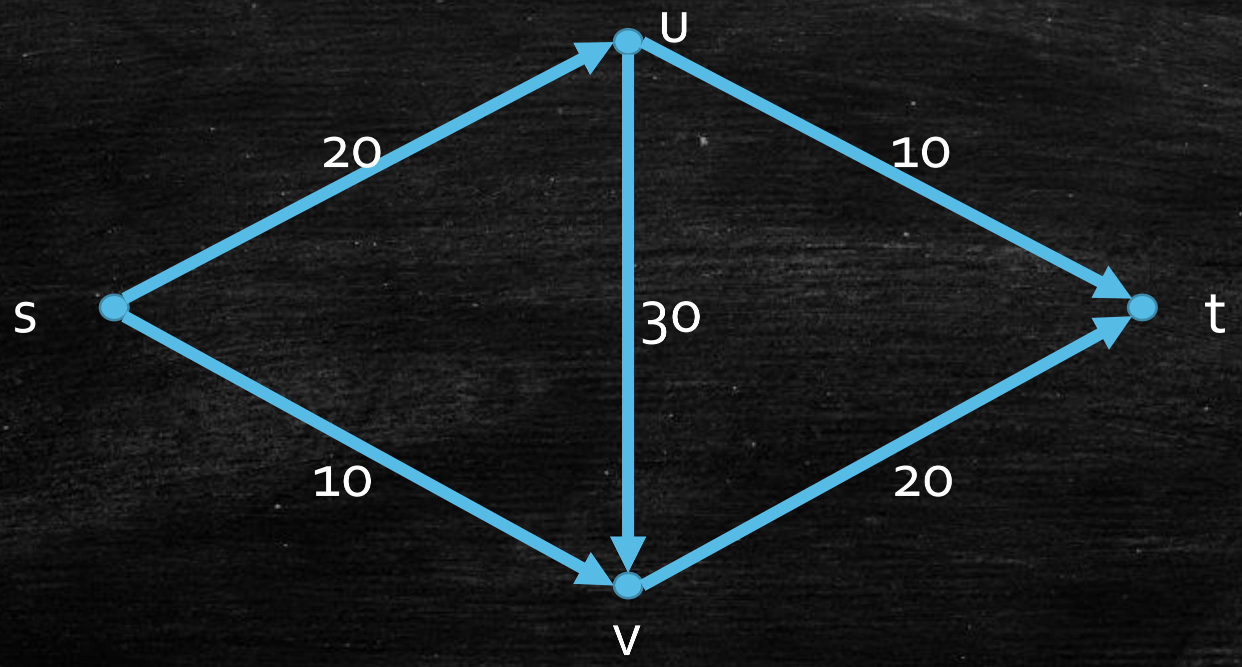
A Greedy Attempt

- We have a flow of size 30, and it is optimal.
- Is this algorithm always optimal?



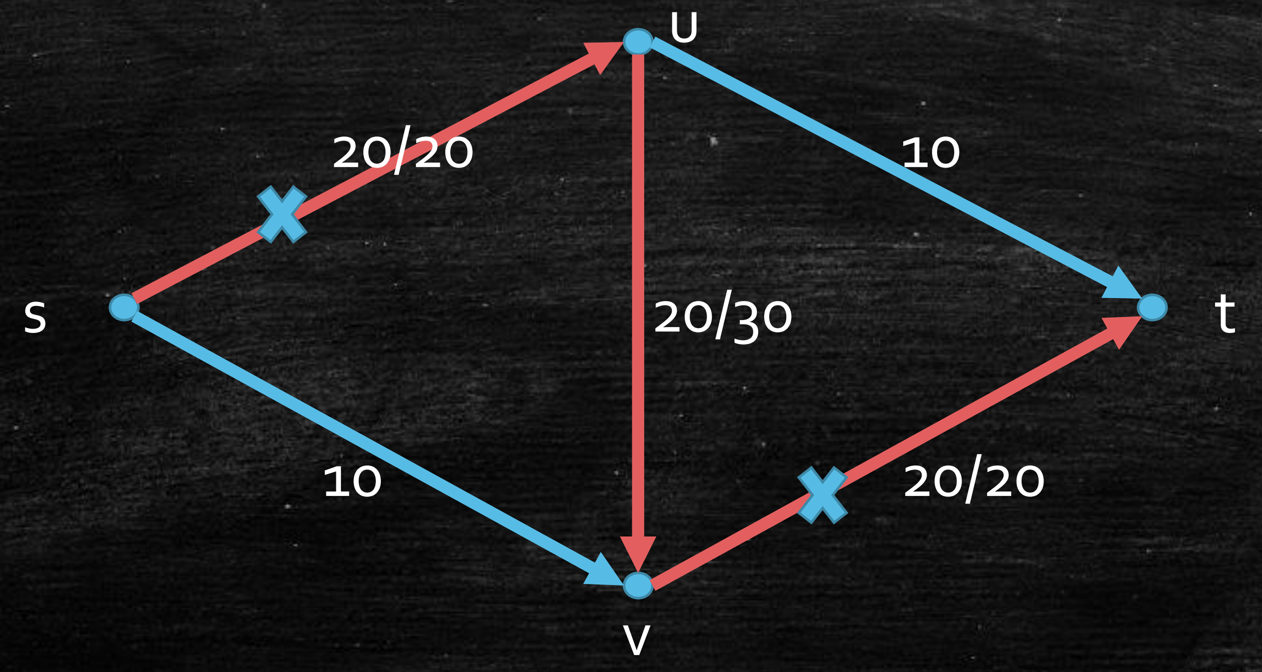
A Greedy Attempt

- Iteratively find an s-t path and push as much flow as possible along it.
- What if our first choice is s-u-v-t?



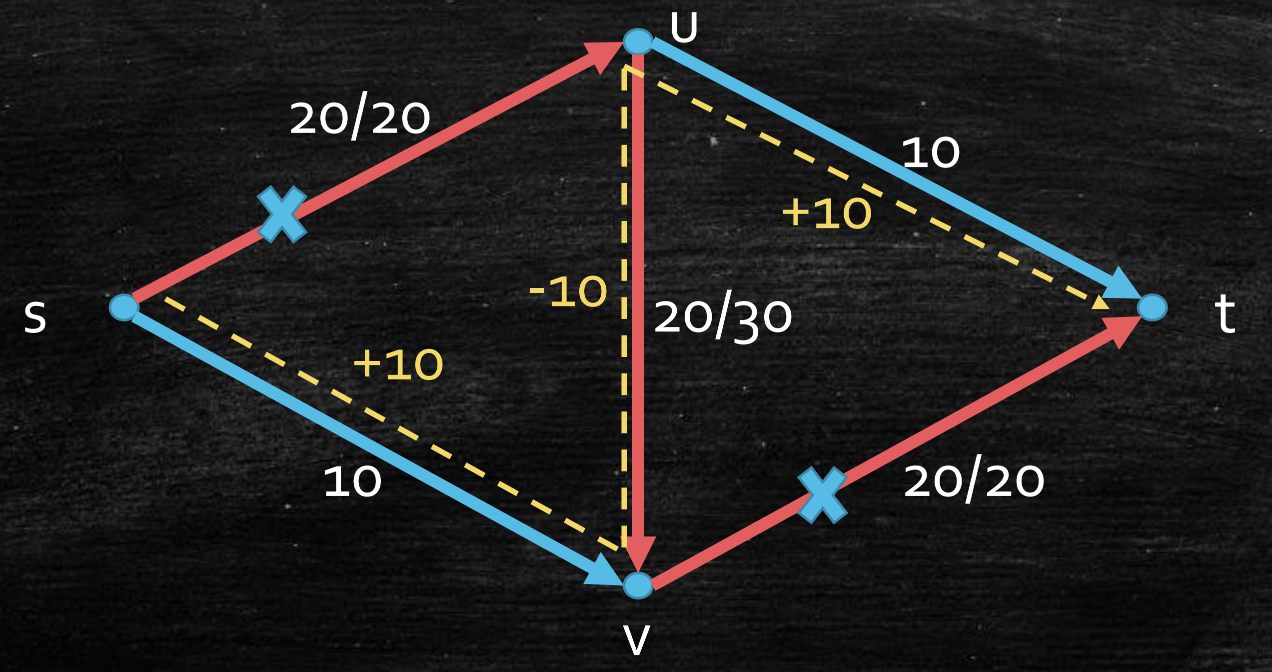
A Greedy Attempt

- Iteratively find an s-t path and push as much flow as possible along it.
- What if our first choice is s-u-v-t?



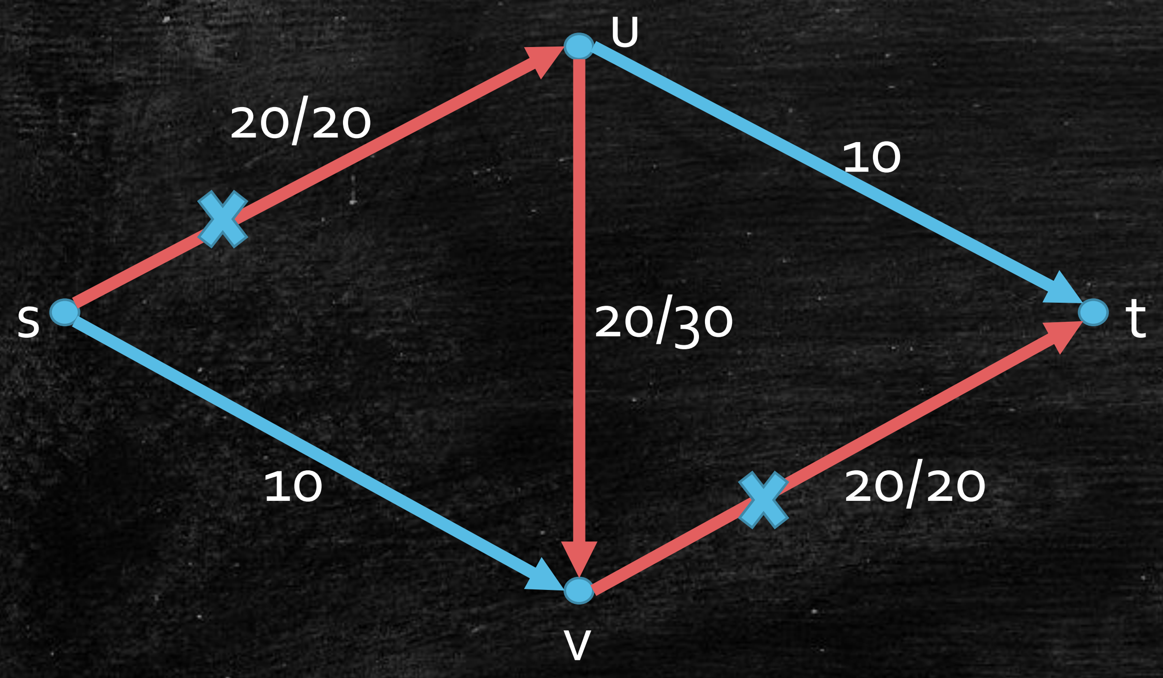
Flow “Cancellation”

- What if our first choice is s-u-v-t?
- We need to be able to “cancel” flow on an edge!

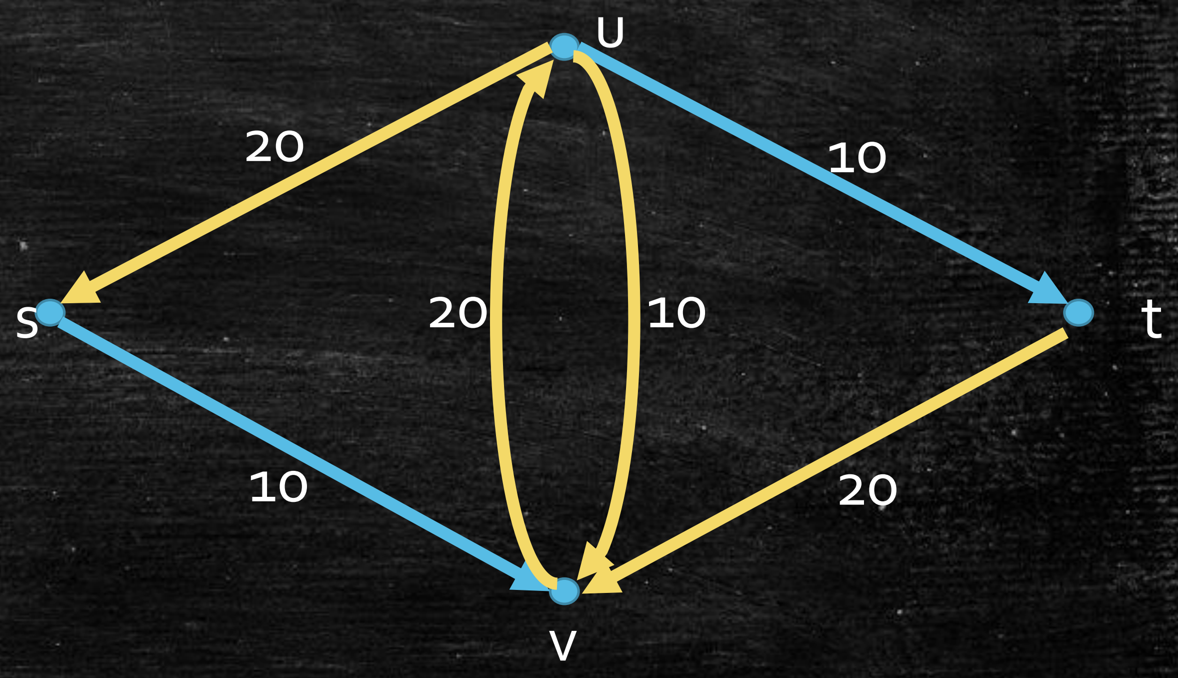


Residual Network

- Residual Network G^f with respect to a flow f.



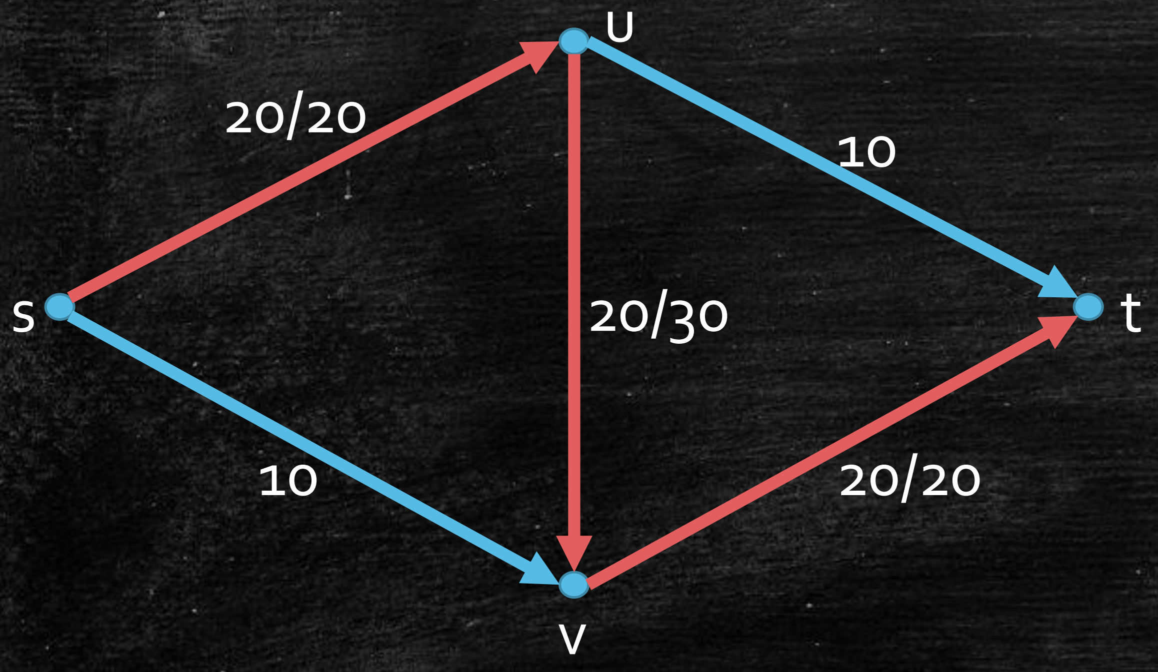
G



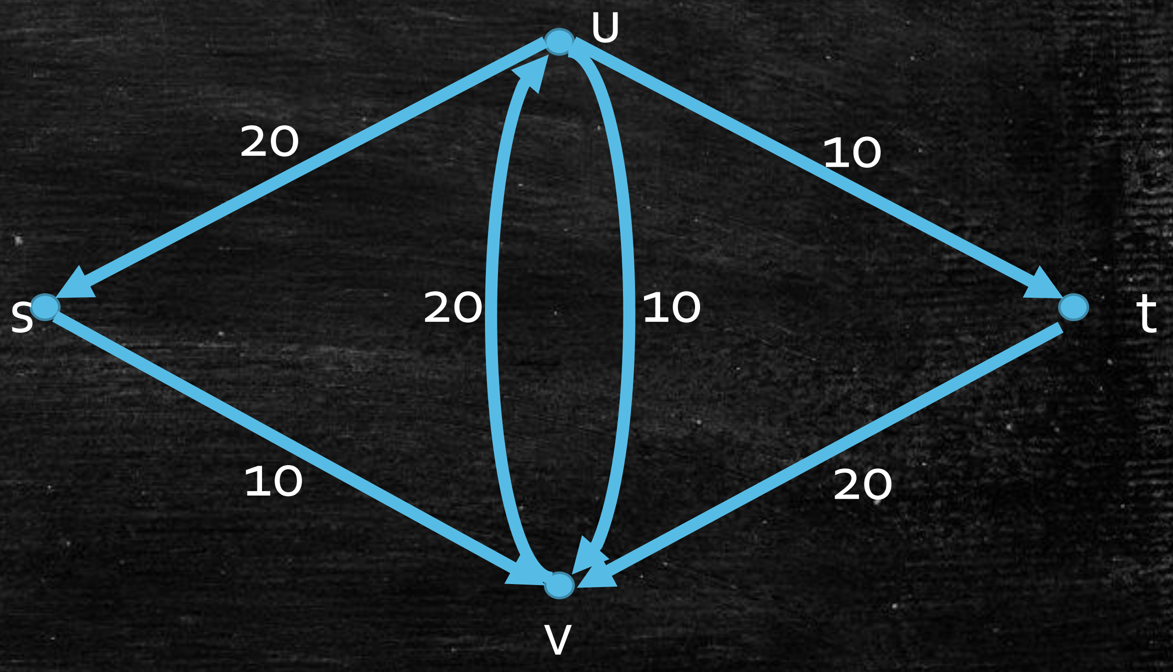
G^f

Residual Network

- Now we are able to continue!
- There is a path on G^f: s-v-u-t



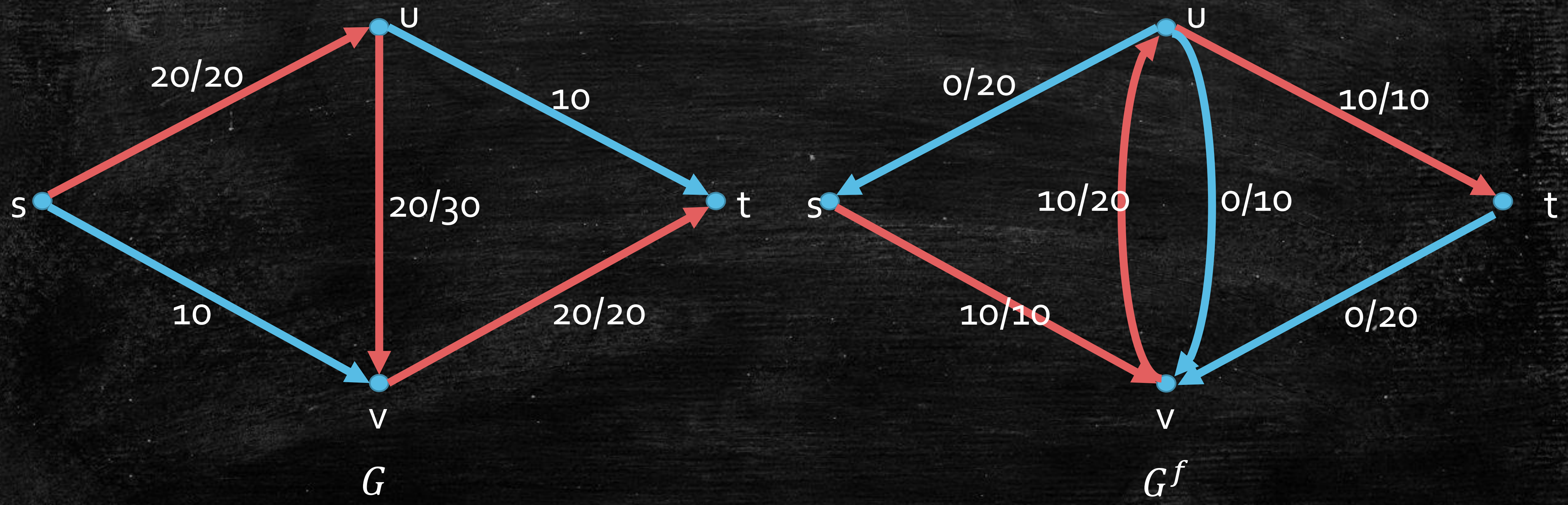
G



G^f

Residual Network

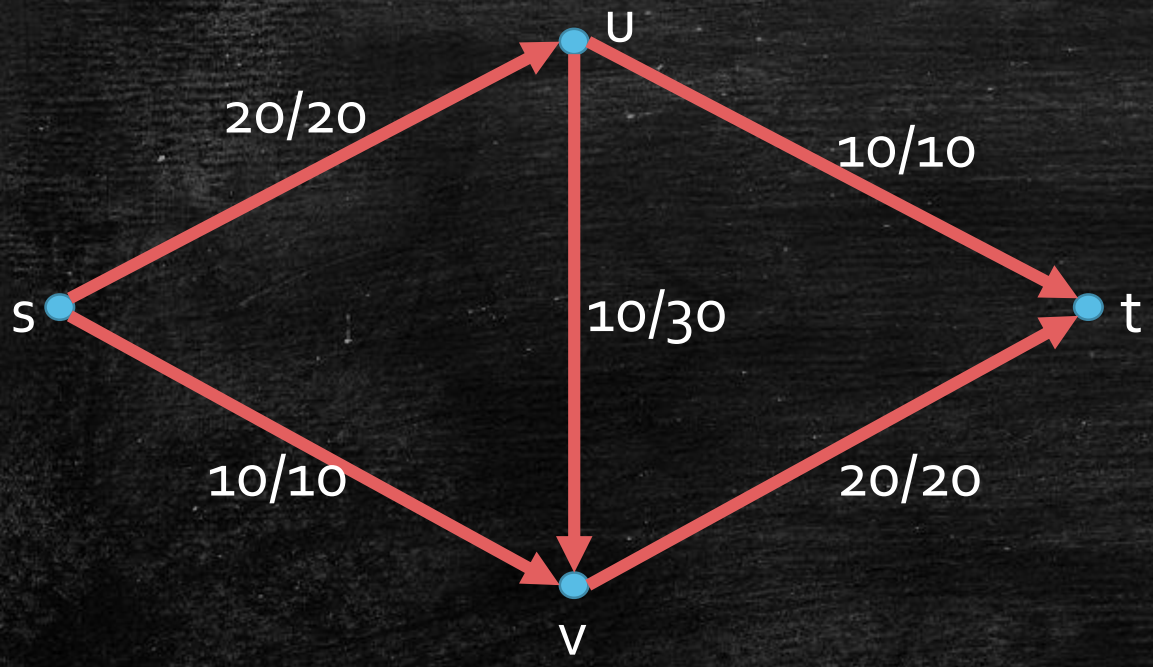
- Now we are able to continue!
- We can push 10 unit of flow on s-v-u-t



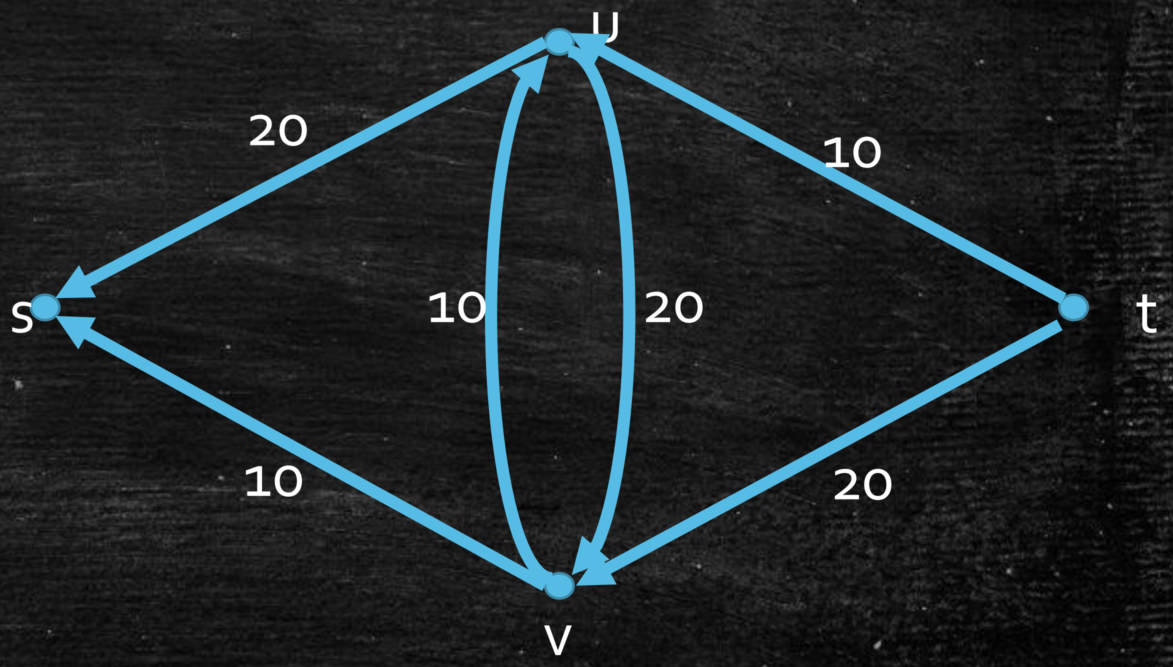
Residual Network

original graph G.

residual graph G^f.



G



G^f

Now it is clear to us that no more flow can be pushed from s to t!

Update Residual Network G^f

Given $G = (V, E)$, c, and a flow f

$G^f = (V^f, E^f)$ and the associated capacity $c^f: E^f \to \mathbb{R}^+$ are defined as follows:

- $V^f = V$
- $(u, v) \in E^f$ if one of the followings holds
  - $(u, v) \in E$ and $f(u, v) < c(u, v)$: in this case, $c^f(u, v) = c(u, v) - f(u, v)$
  - $(v, u) \in E$ and $f(v, u) > 0$: in this case, $c^f(u, v) = f(v, u)$

Putting Together

- Initialize an empty flow f and the corresponding residual flow G^f.
- Iteratively
  - find a path on G^f,
  - push maximum amount of flow on G^f, and
  - update f and G^f,
- until there is no s-t path on G^f.

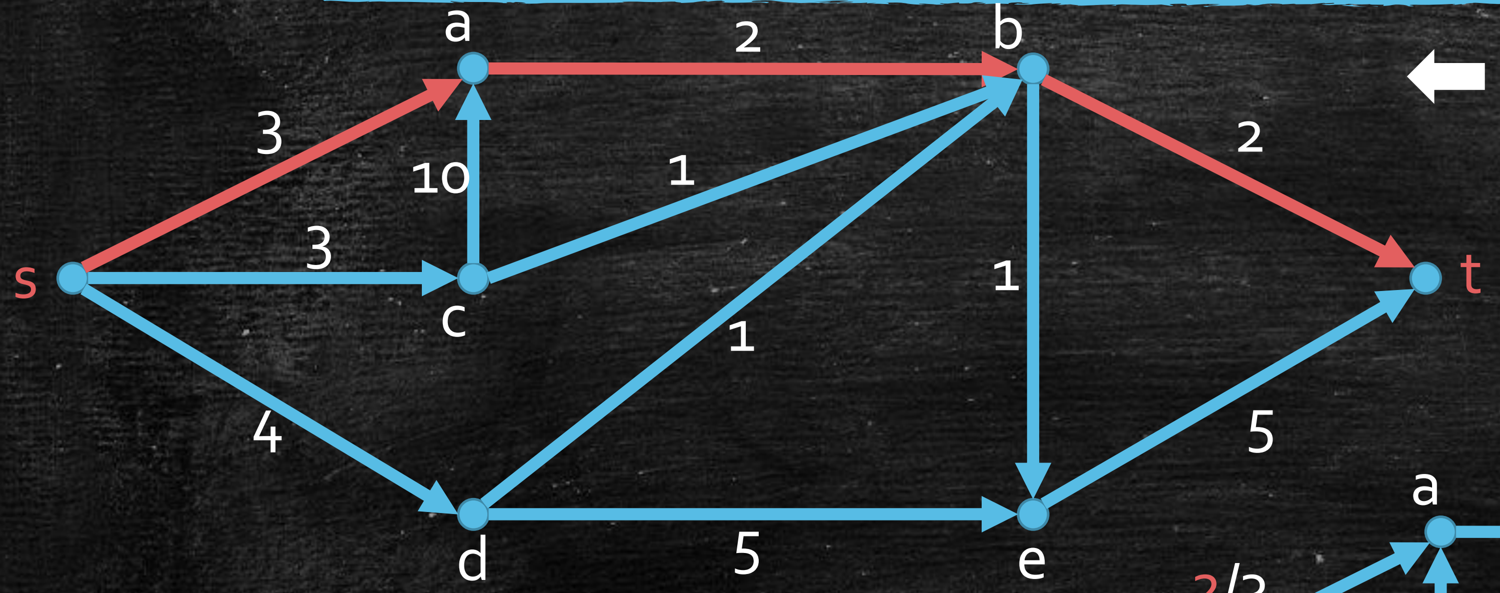
This is exactly Ford-Fulkerson Algorithm!

Ford-Fulkerson Algorithm

FordFulkerson($G = (V, E), s, t, c$):

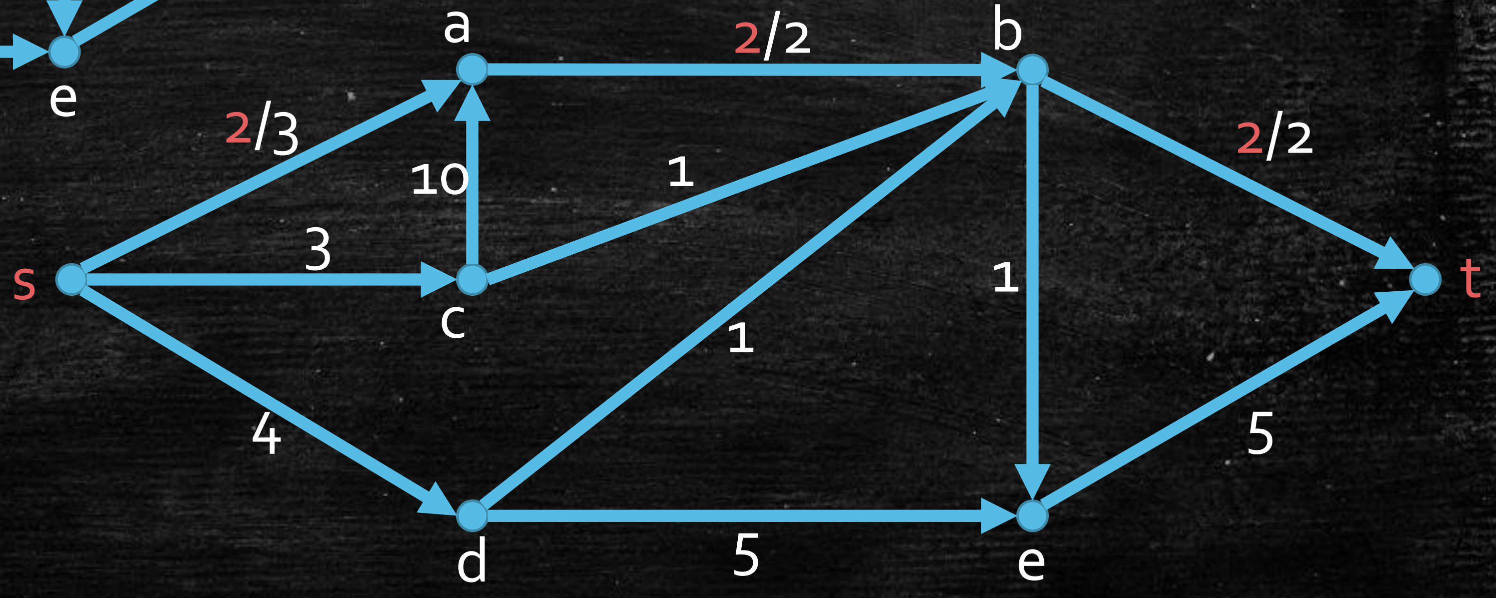
initialize f such that $\forall e \in E: f(e) = 0$; initialize $G^f \leftarrow G$;
while there is an s-t path p on G^f:
    find an edge $e \in p$ with minimum capacity b;
    for each $e = (u, v) \in p$:
        if $(u, v) \in E$: update $f(e) \leftarrow f(e) + b$;
        if $(v, u) \in E$: update $f(e) \leftarrow f(e) - b$;
    endfor
    update G^f;
endwhile
return f

Ford-Fulkerson Algorithm

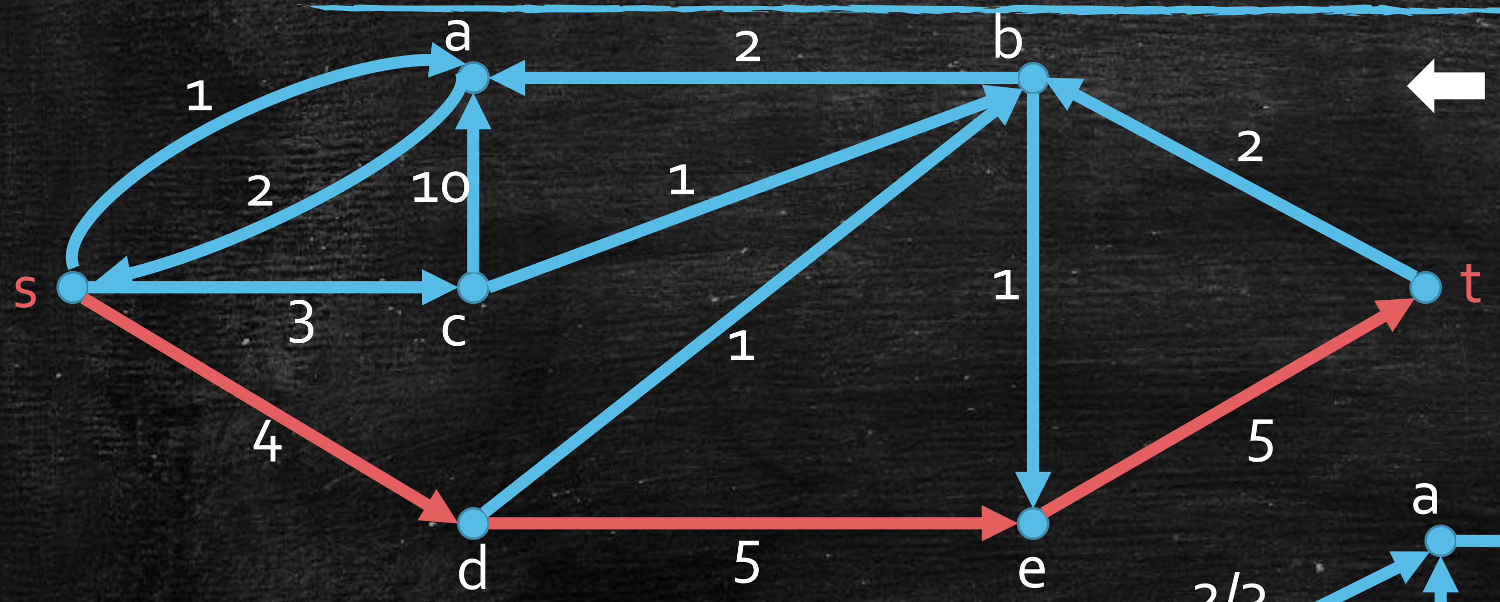


Residual Network

Current Flow

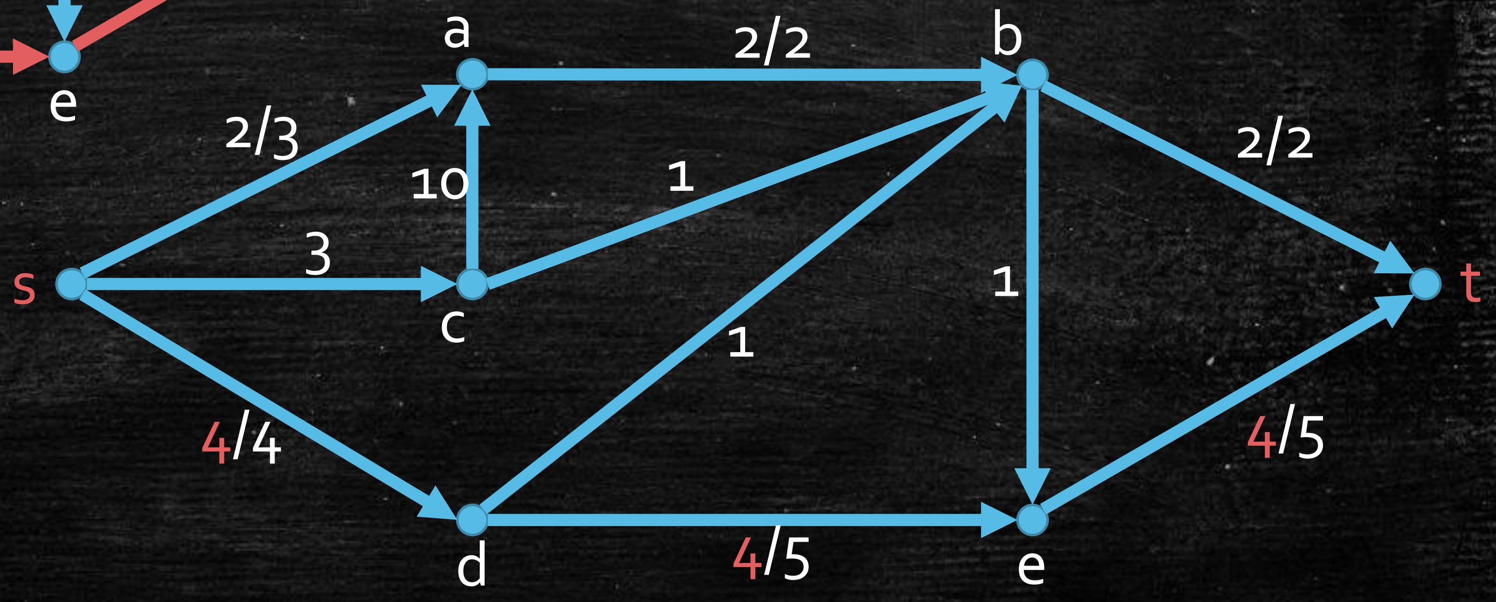


Ford-Fulkerson Algorithm

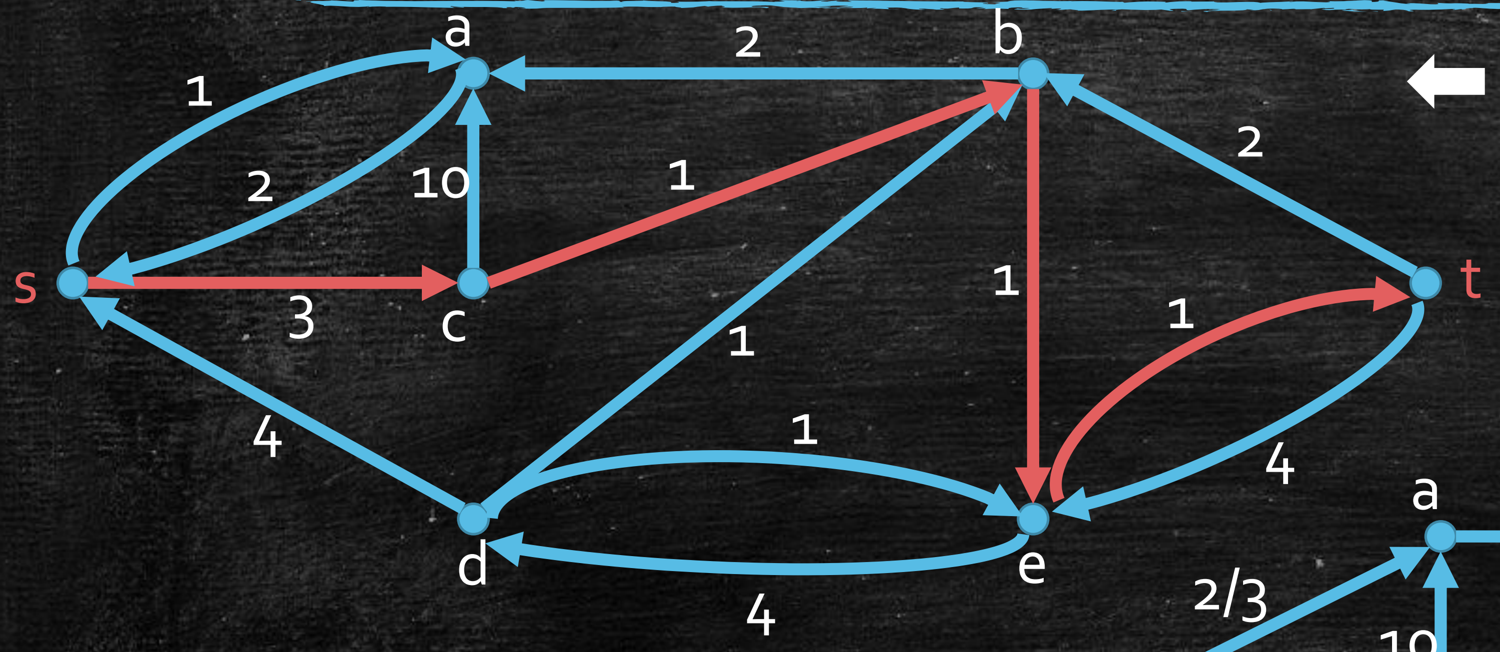


⬅ Residual Network

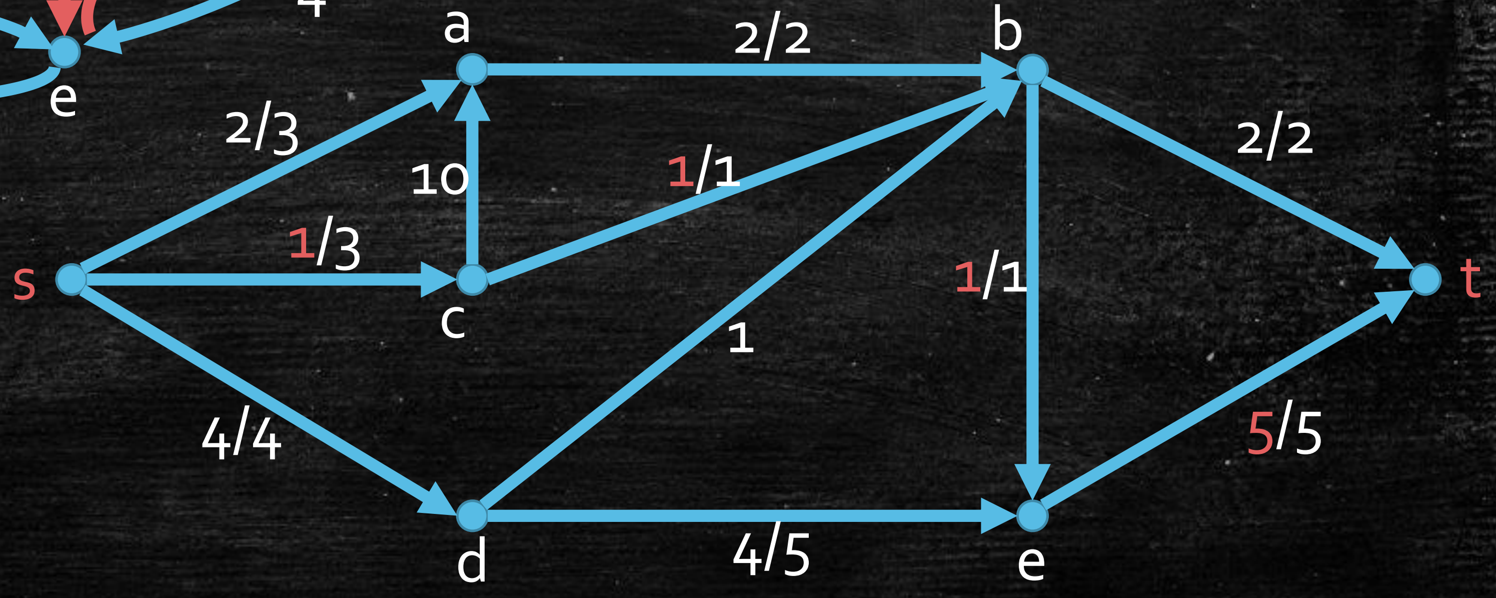
Current Flow ➡



Ford-Fulkerson Algorithm

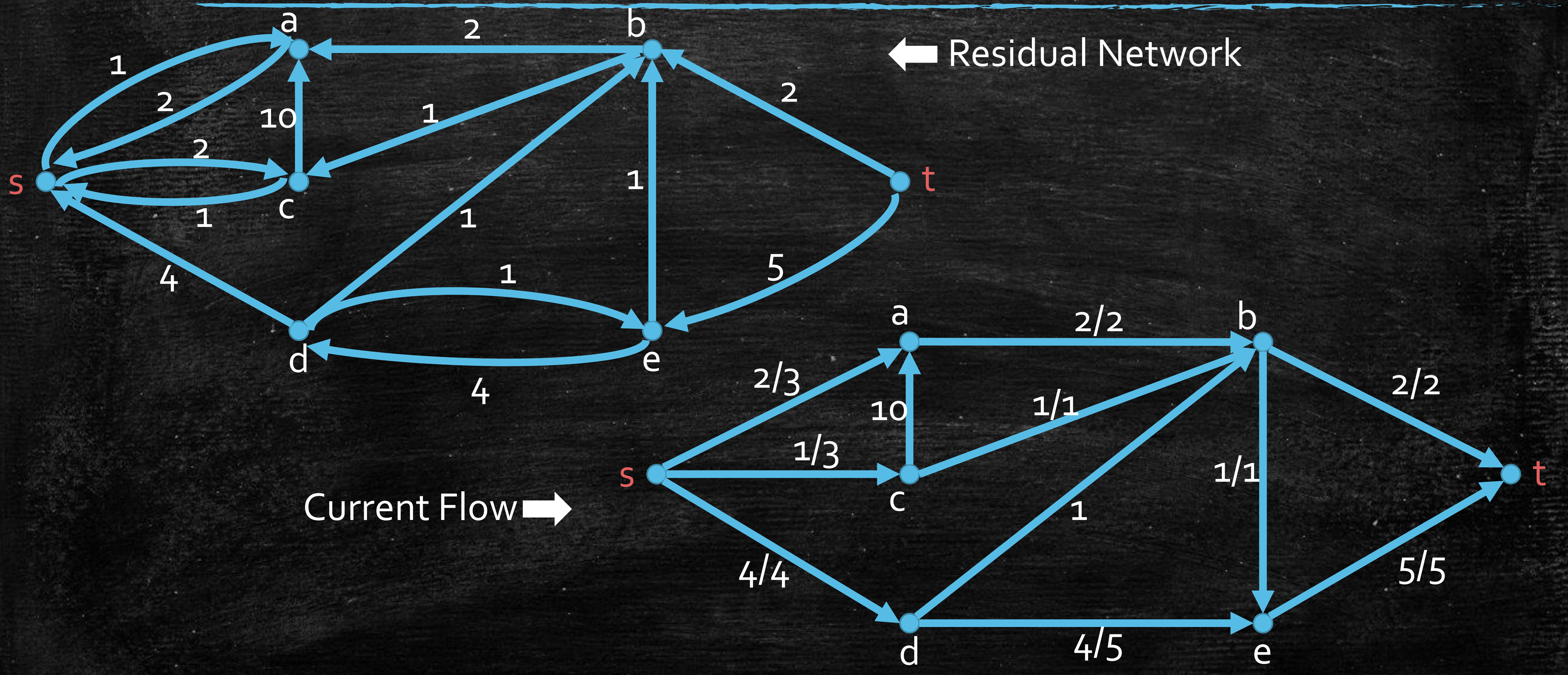


Residual Network



Current Flow

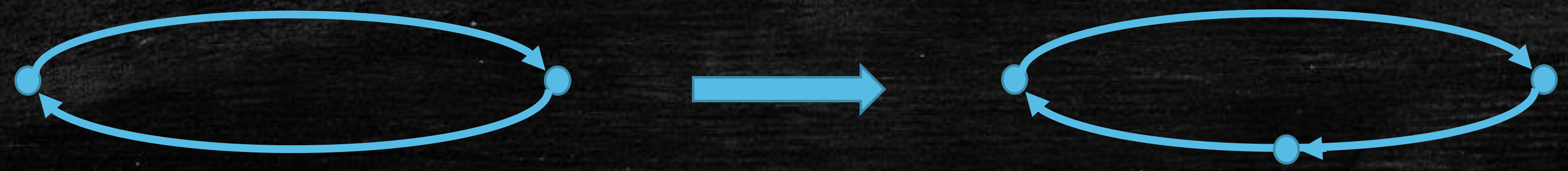
No more s-t path... Algorithm Terminate...



A Small Bug…

```
4.    for each e = (u, v) ∈ p:
5.        if (u, v) ∈ E: update f(e) ← f(e) + b;
6.        if (v, u) ∈ E: update f(e) ← f(e) − b;
7.    endfor
```

- What if we have both $(u, v) \in E$ and $(v, u) \in E$?
- We need to do either 5 or 6, but not both!
- Fix: modify the graph so that no anti-parallel edge exists.



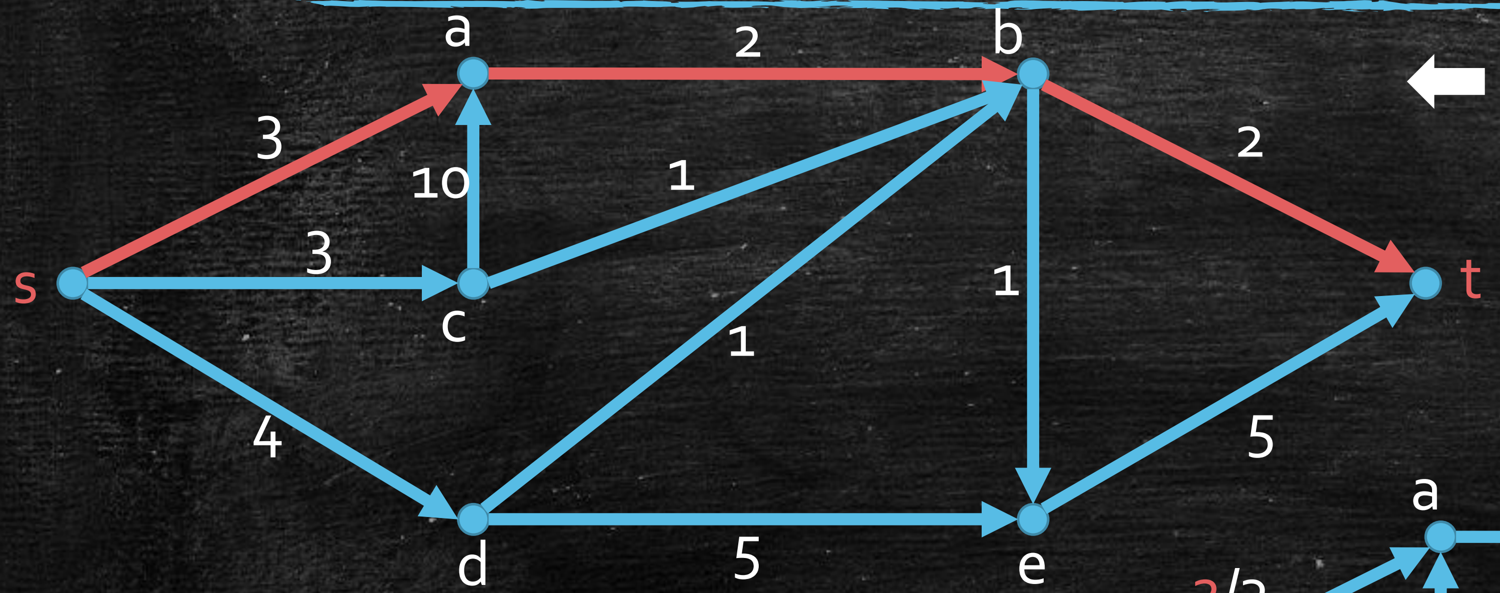
Correctness? Time Complexity?

- Correctness: Max-Flow-Min-Cut Theorem
- Time Complexity:
  - Question 1: Does the algorithm always halt?
  - Question 2: If so, what is the time complexity?

Max-Flow-Min-Cut Theorem

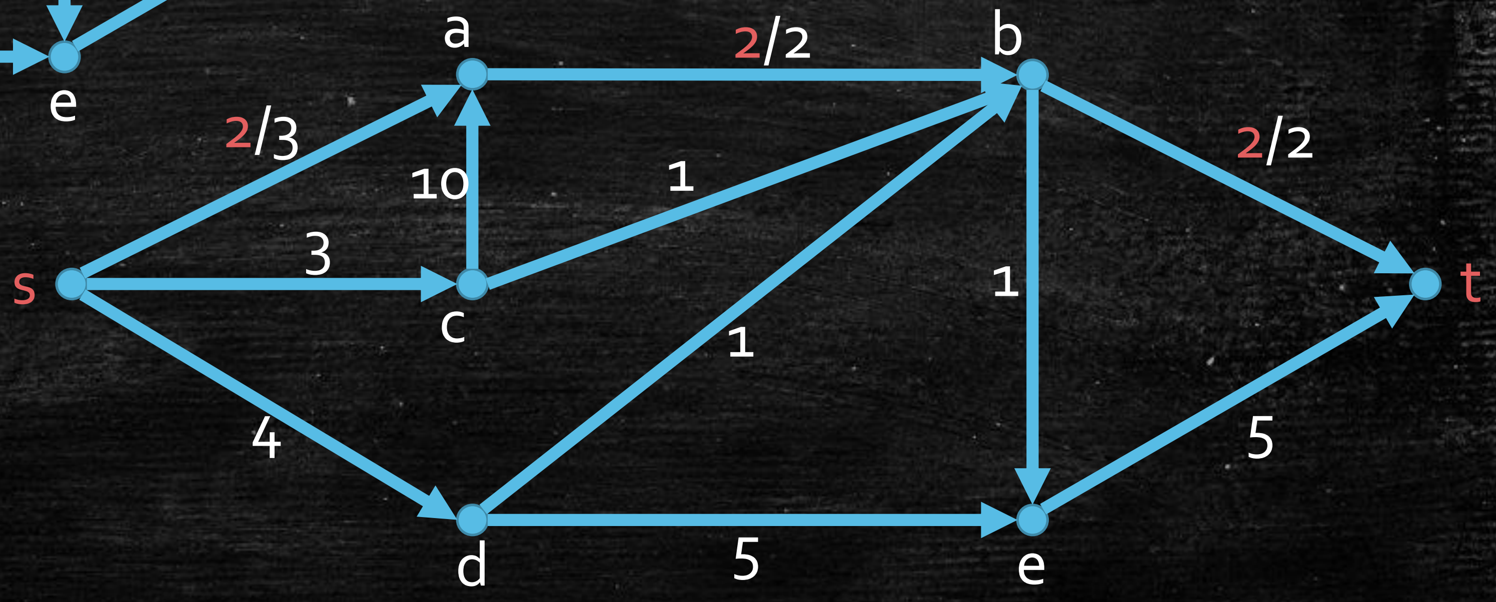
Correctness of Ford-Fulkerson Algorithm

Ford-Fulkerson Algorithm

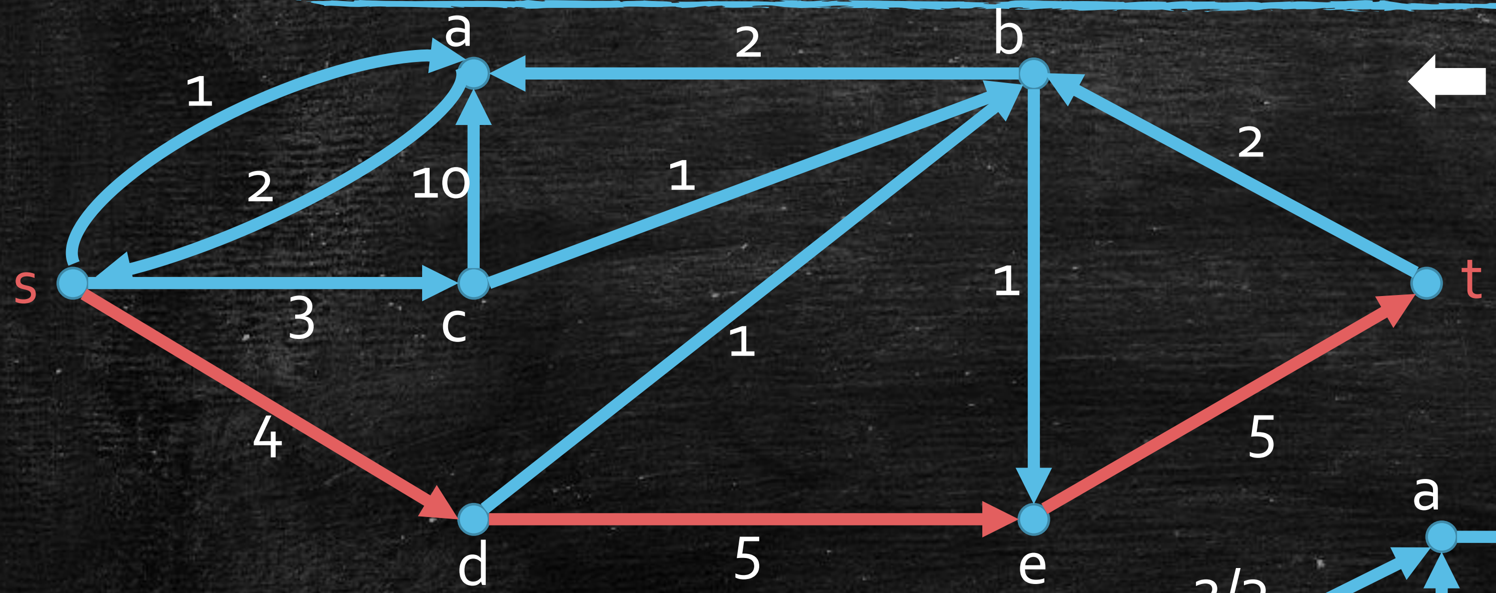


Residual Network

Current Flow

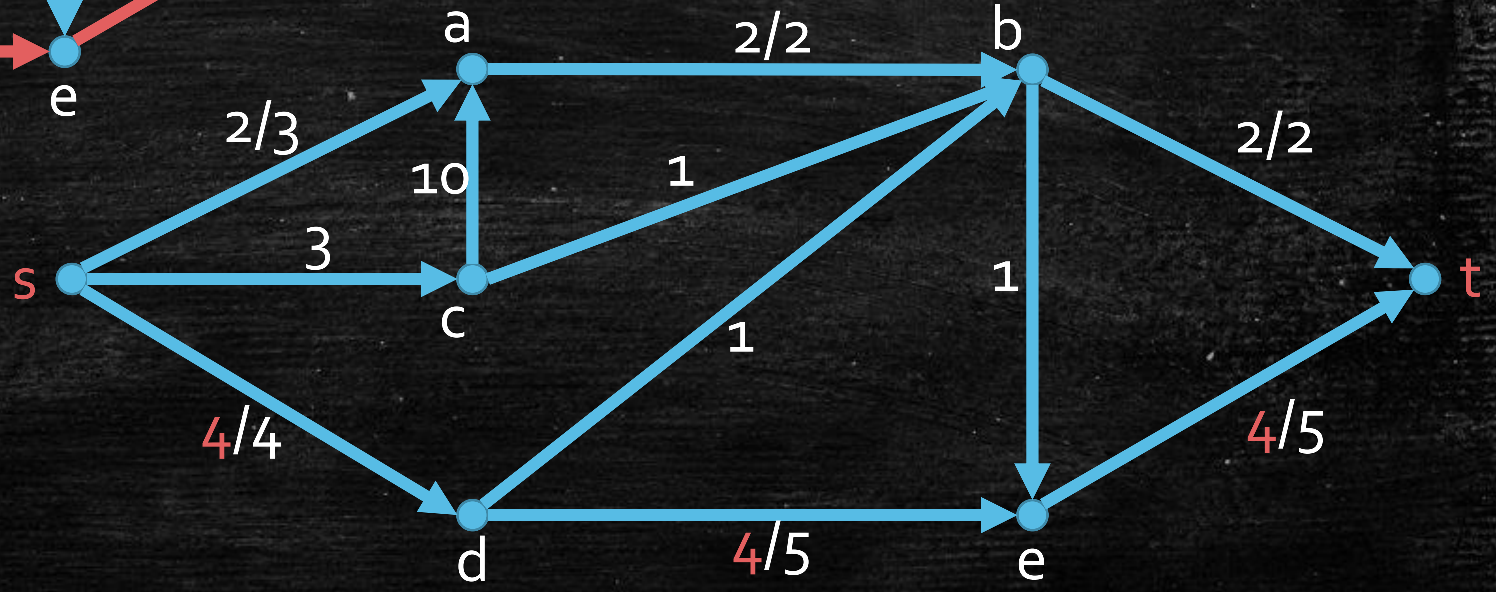


Ford-Fulkerson Algorithm

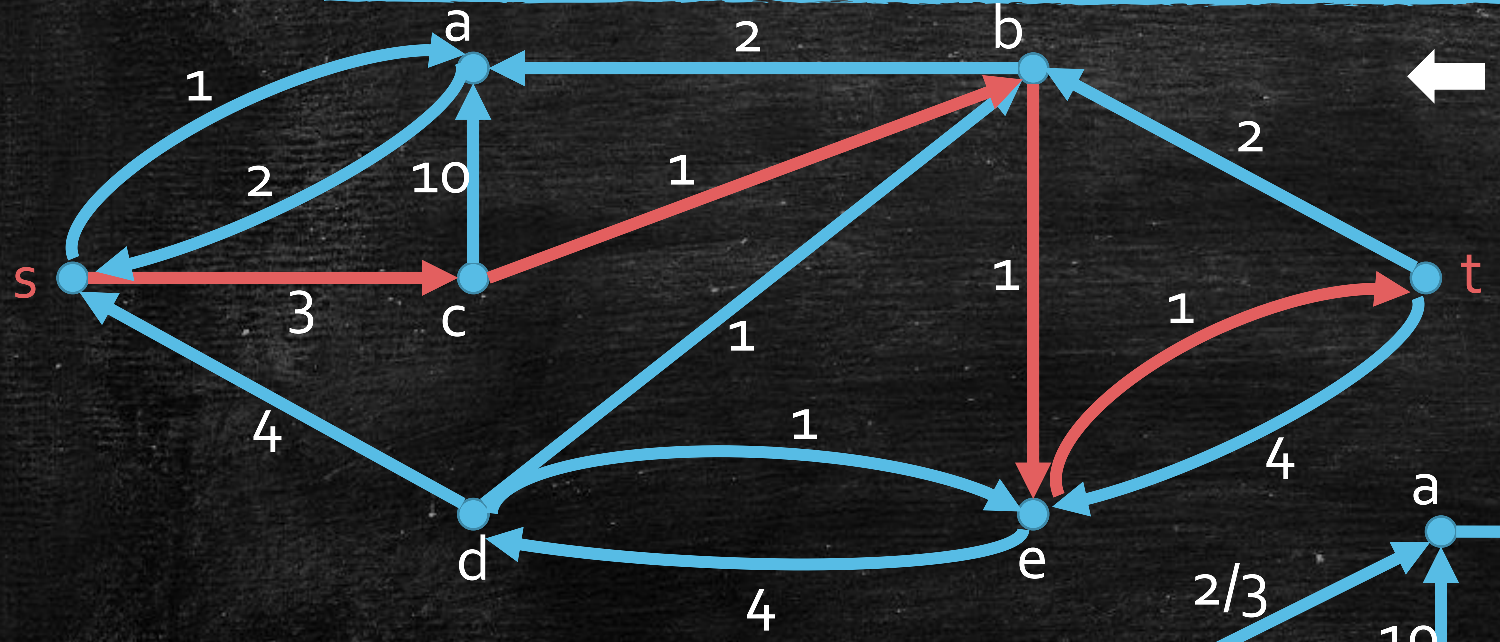


Residual Network

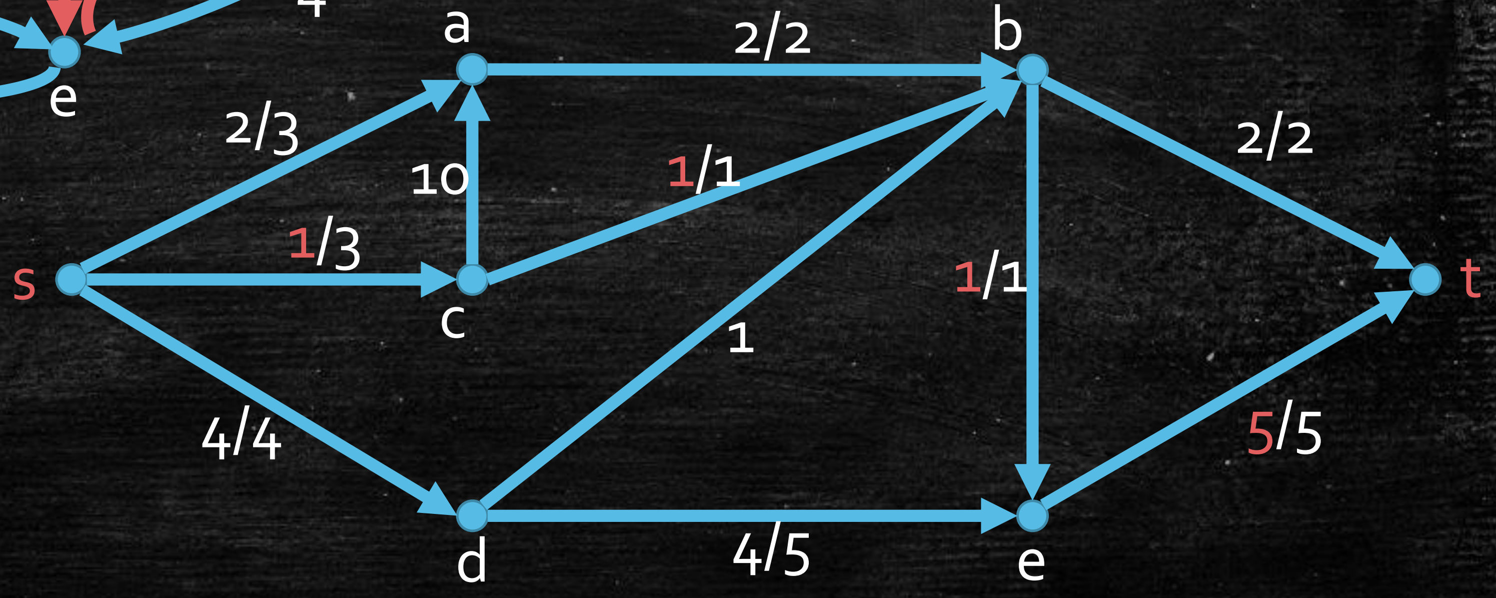
Current Flow



Ford-Fulkerson Algorithm

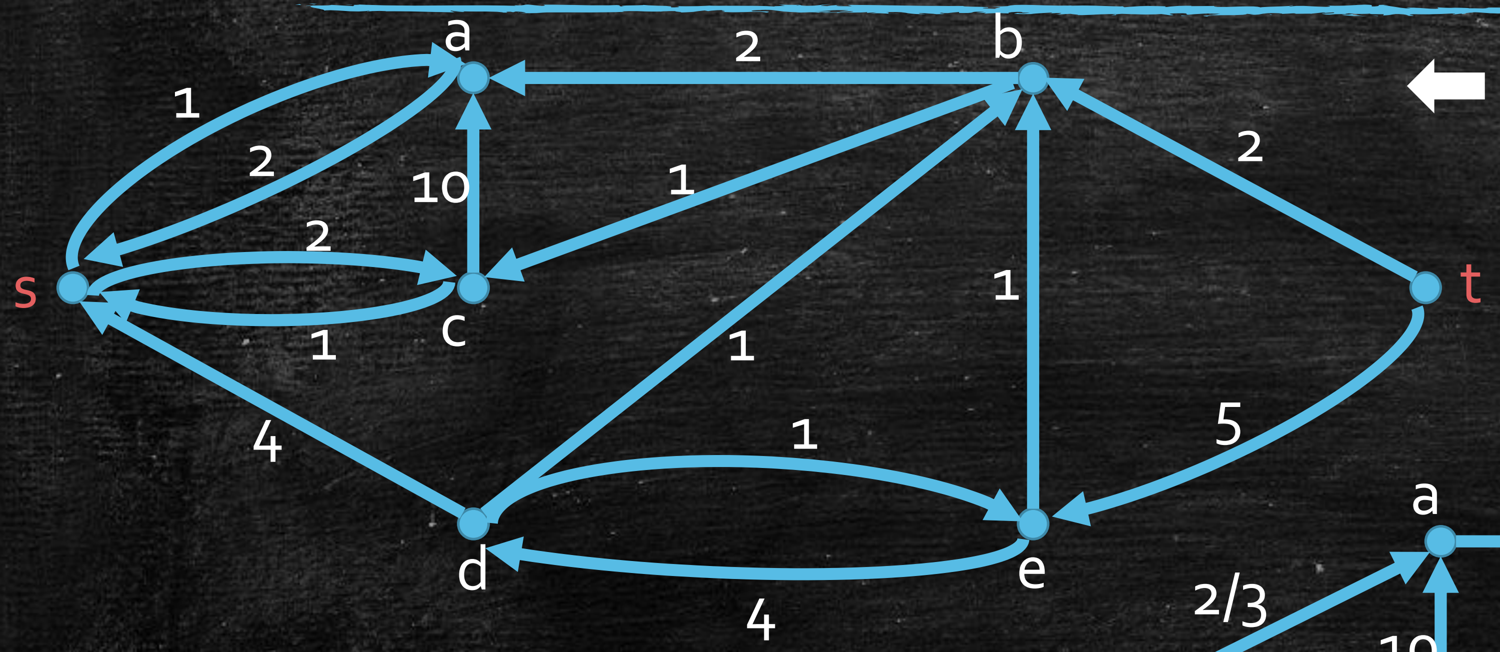


Residual Network



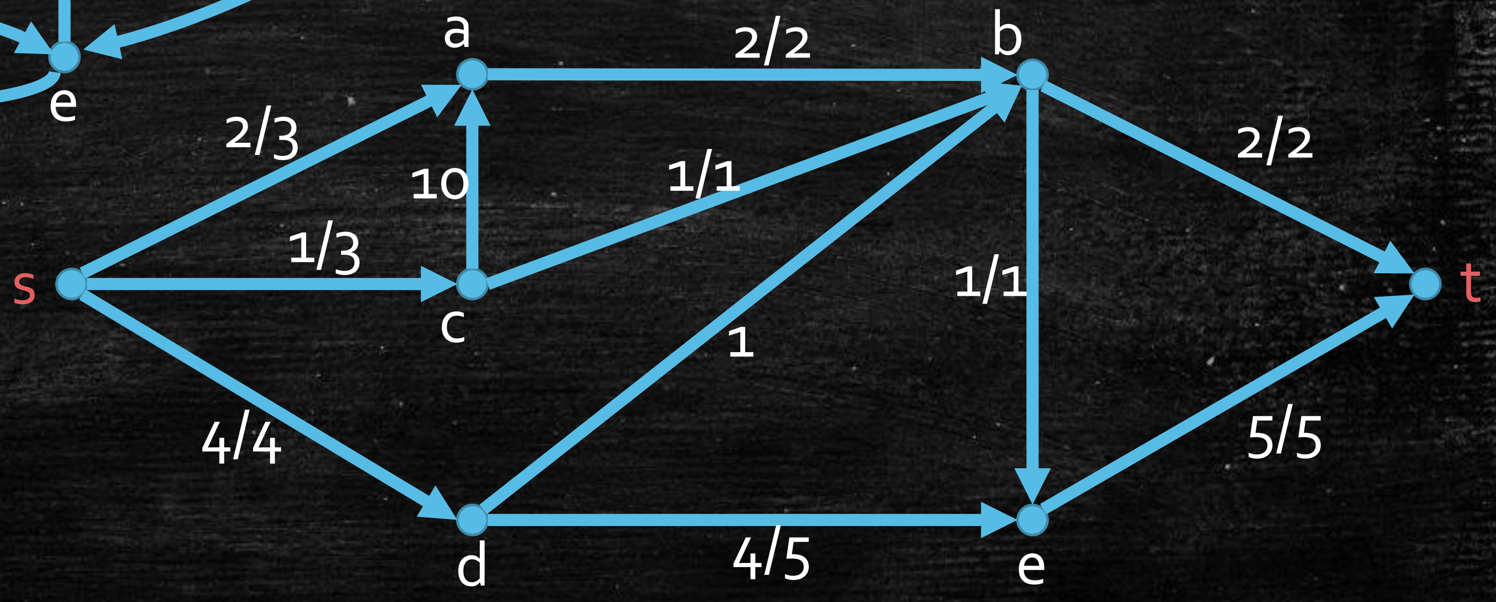
Current Flow

Is $v(f) = 7$ optimal?
Correctness of Ford-Fulkerson algorithm?

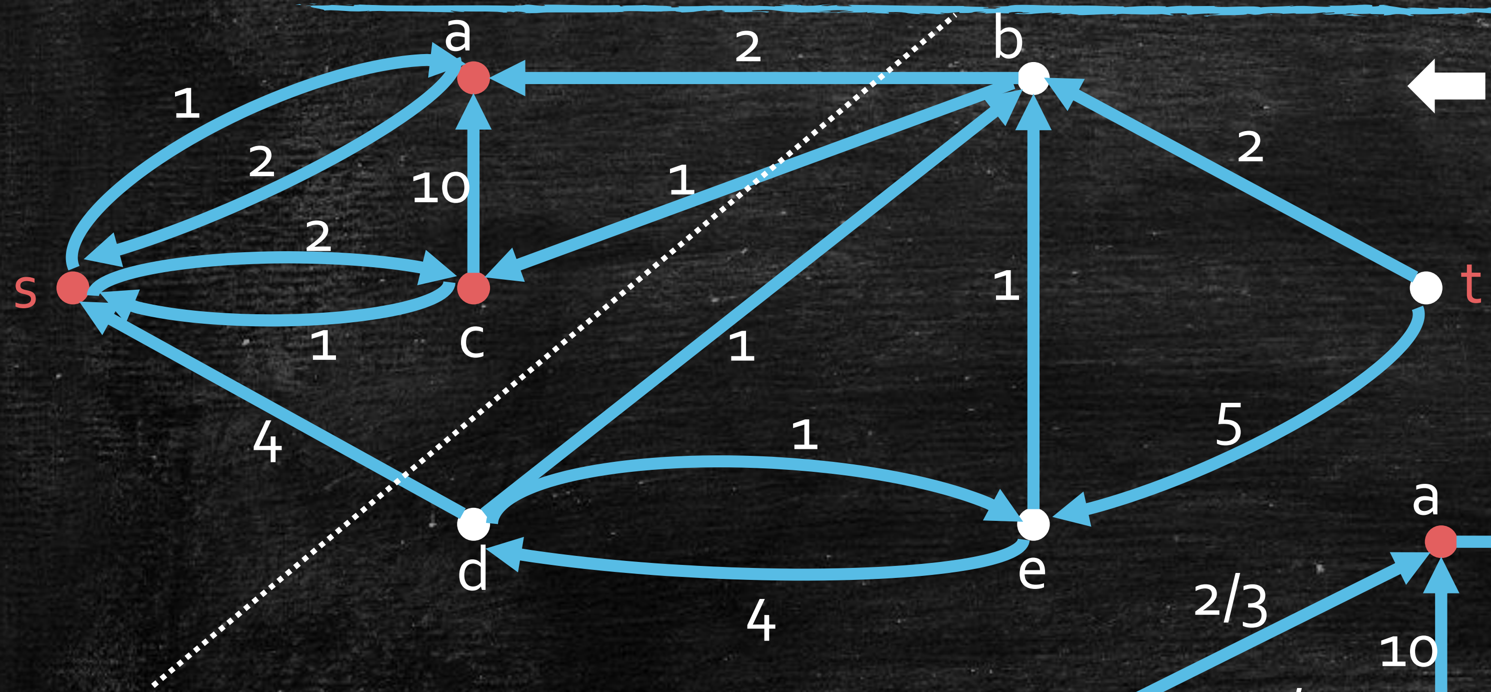


Residual Network

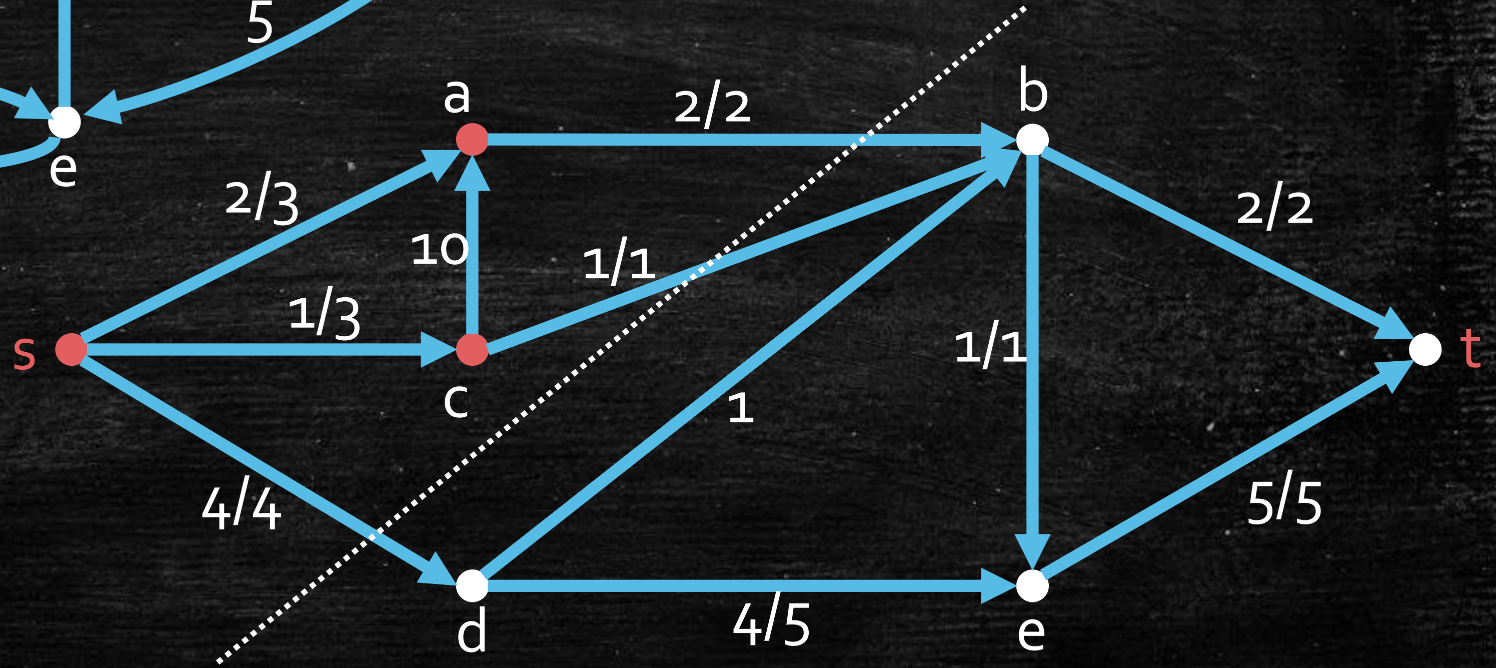
Current Flow



Consider the following partition of vertices…

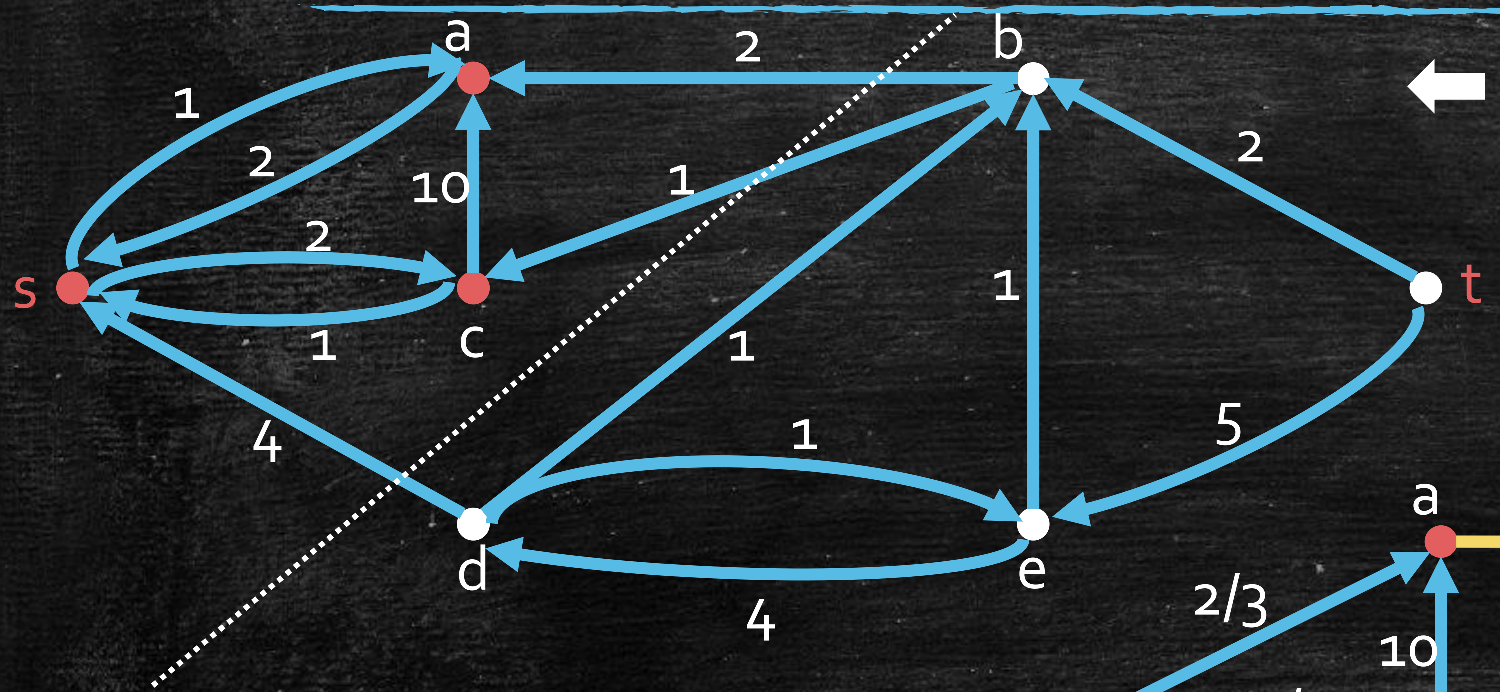


Residual Network



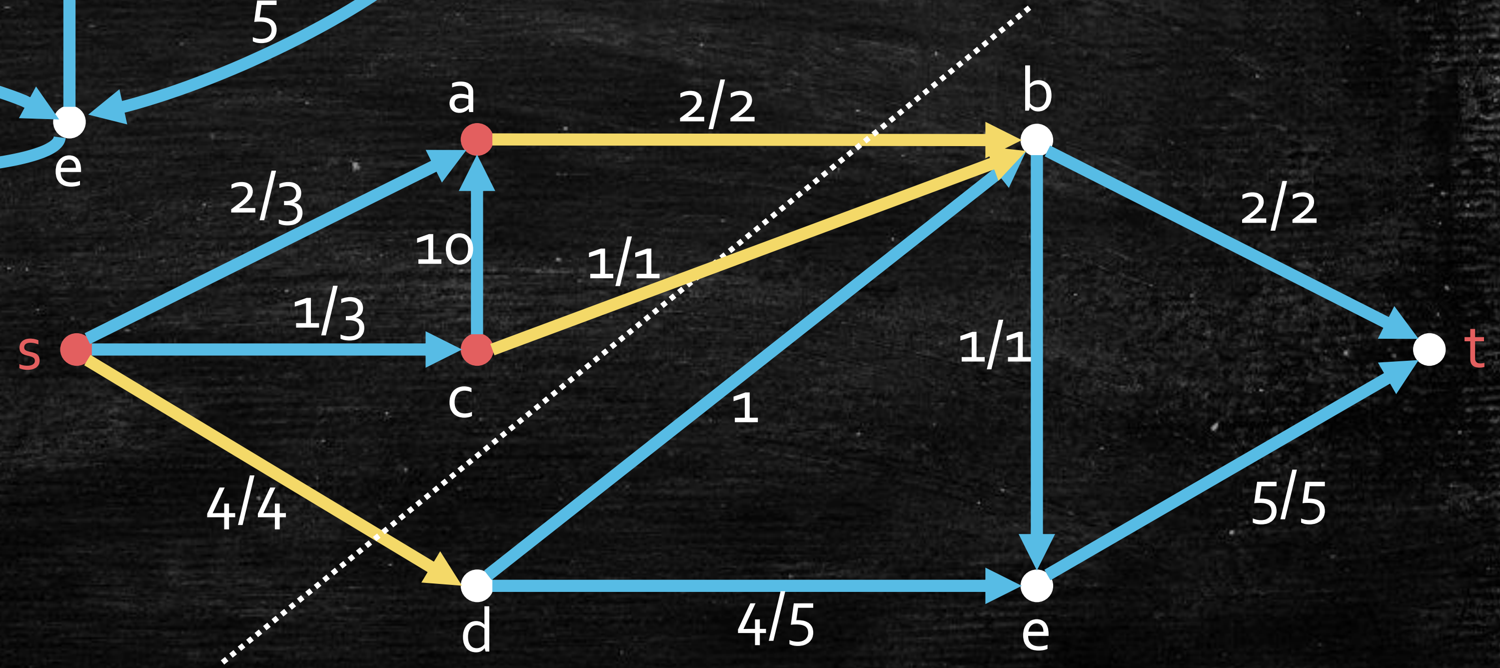
Current Flow

No more additional flow can be sent along the yellow edges crossing the border!

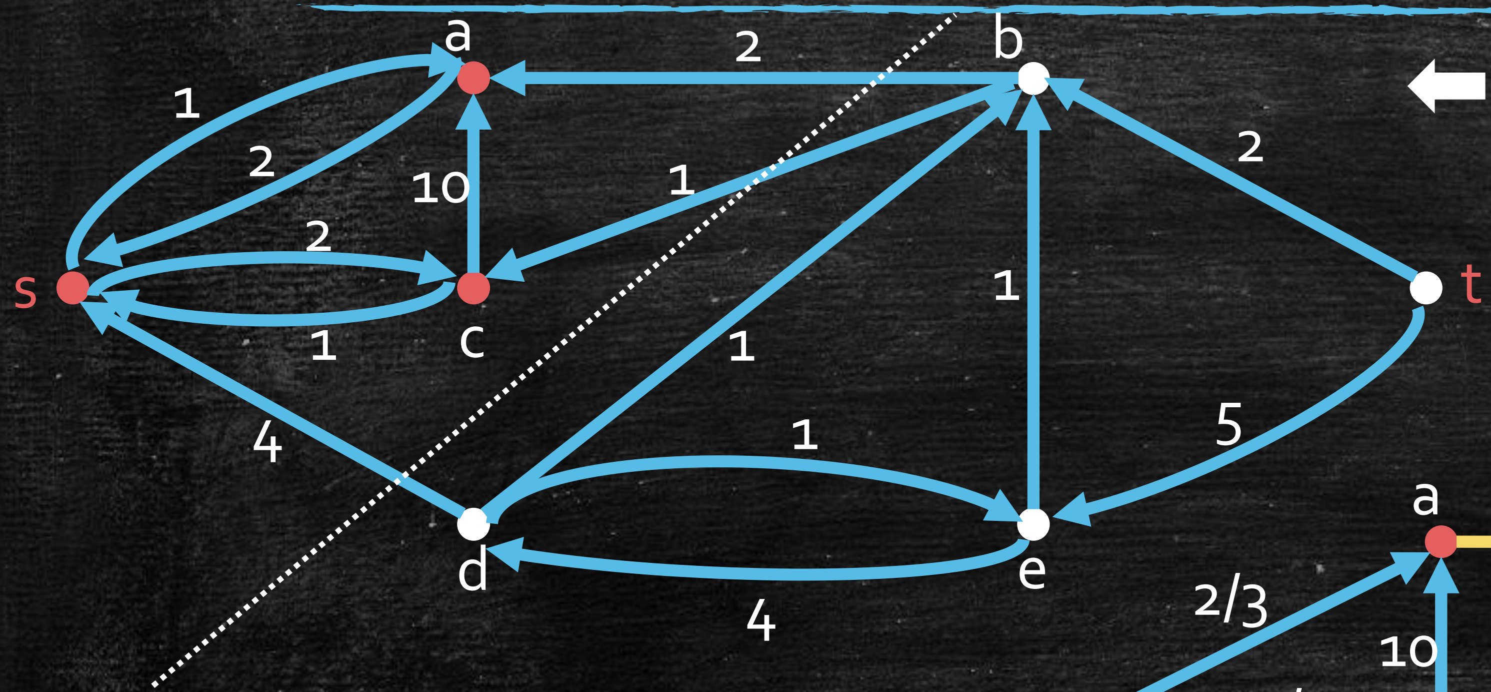


Residual Network

Current Flow

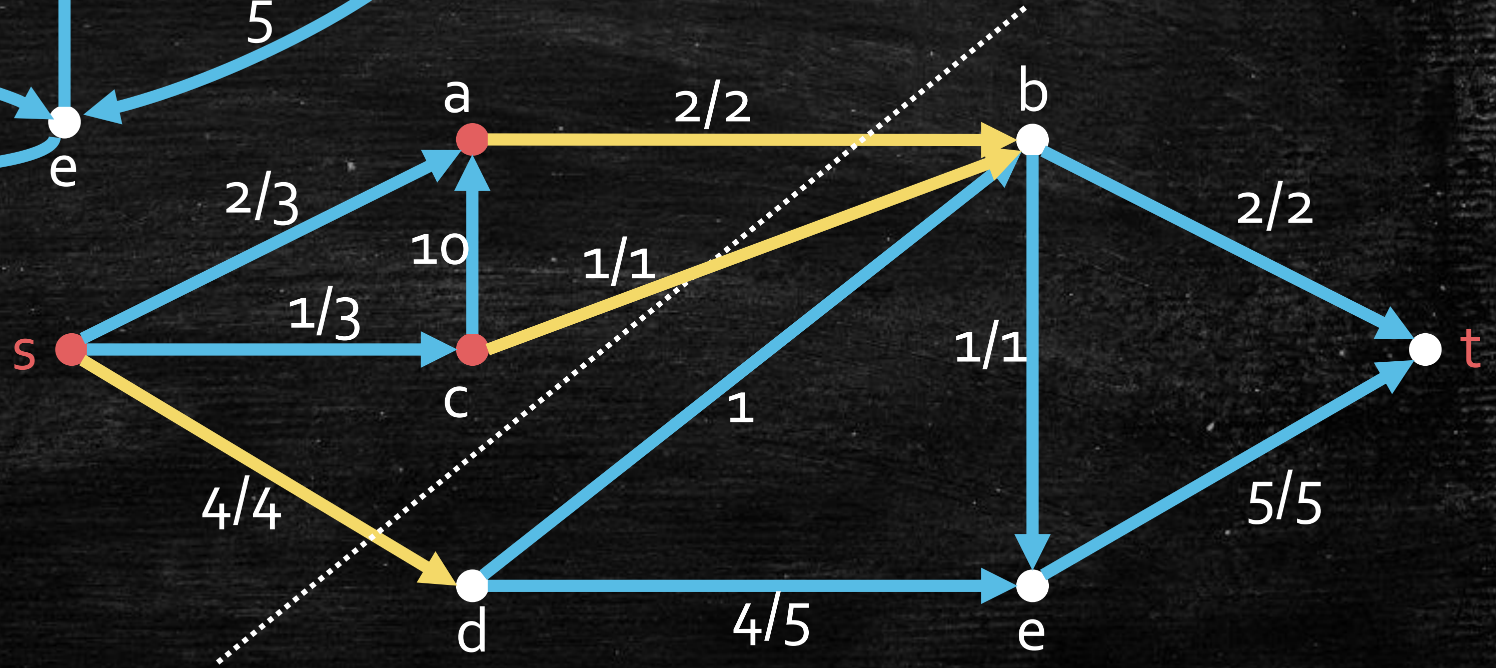


We have $v(f) \leq 7$, since we can send at most 7 units of flow across the border.

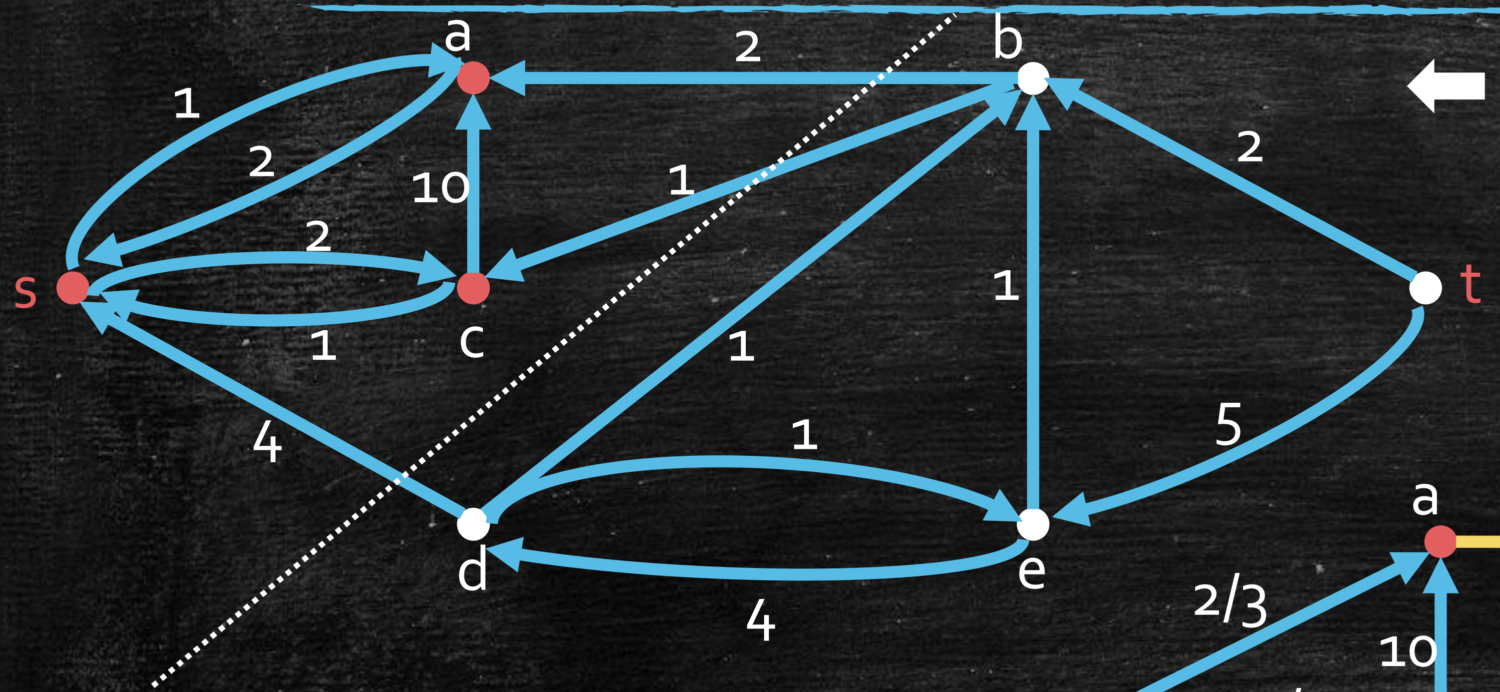


Residual Network

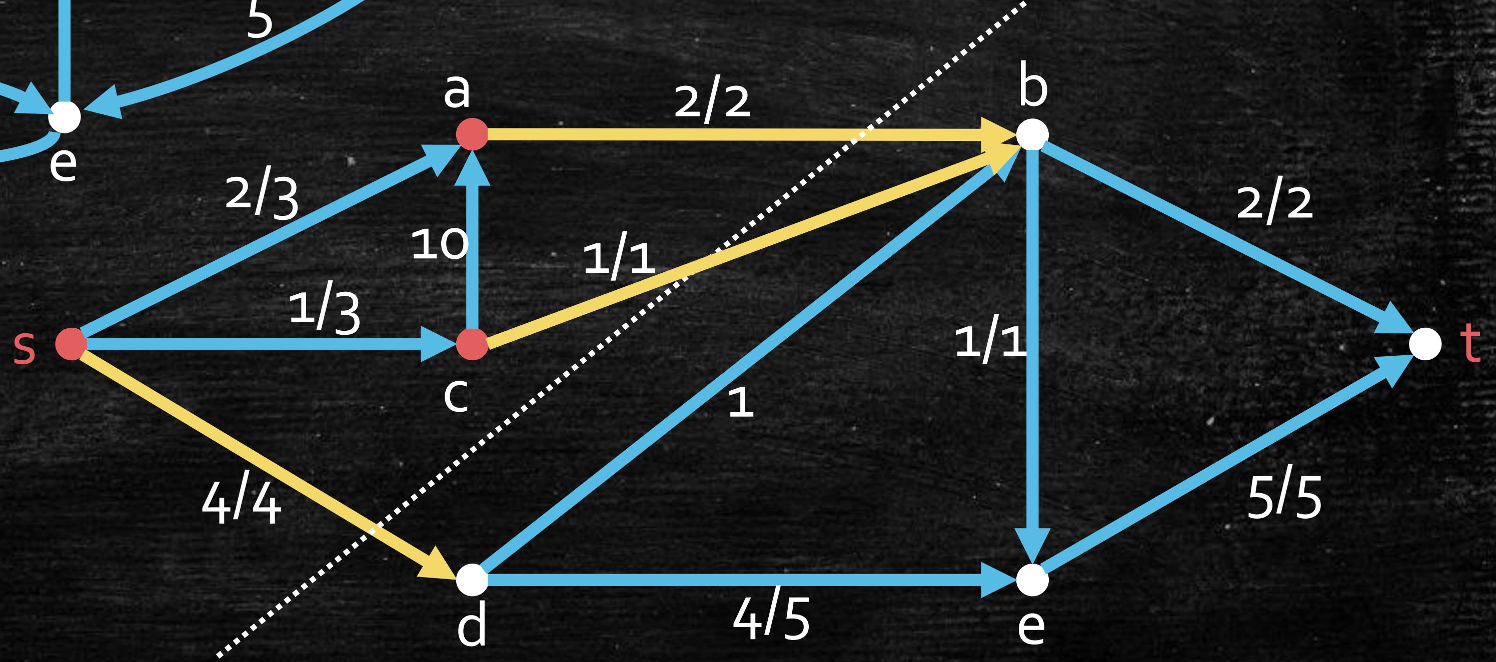
Current Flow



Thus, $v(f) = 7$ is optimal!

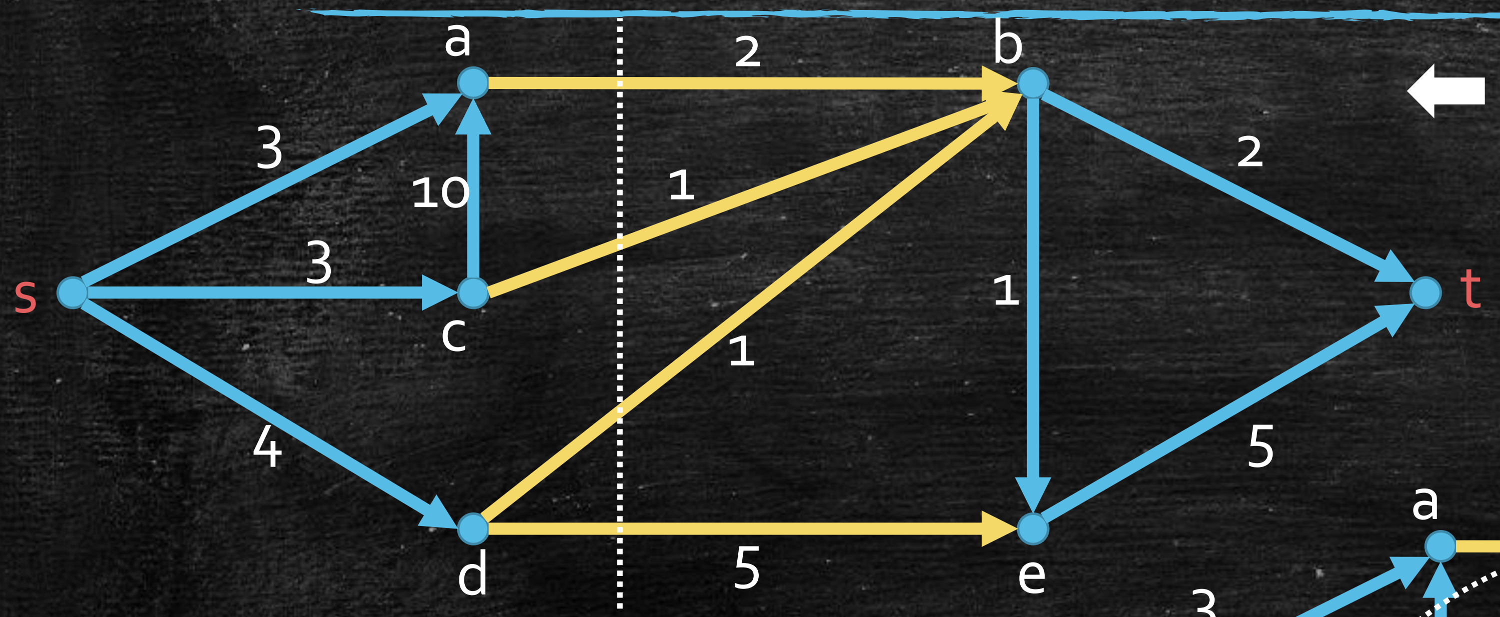


⬅ Residual Network

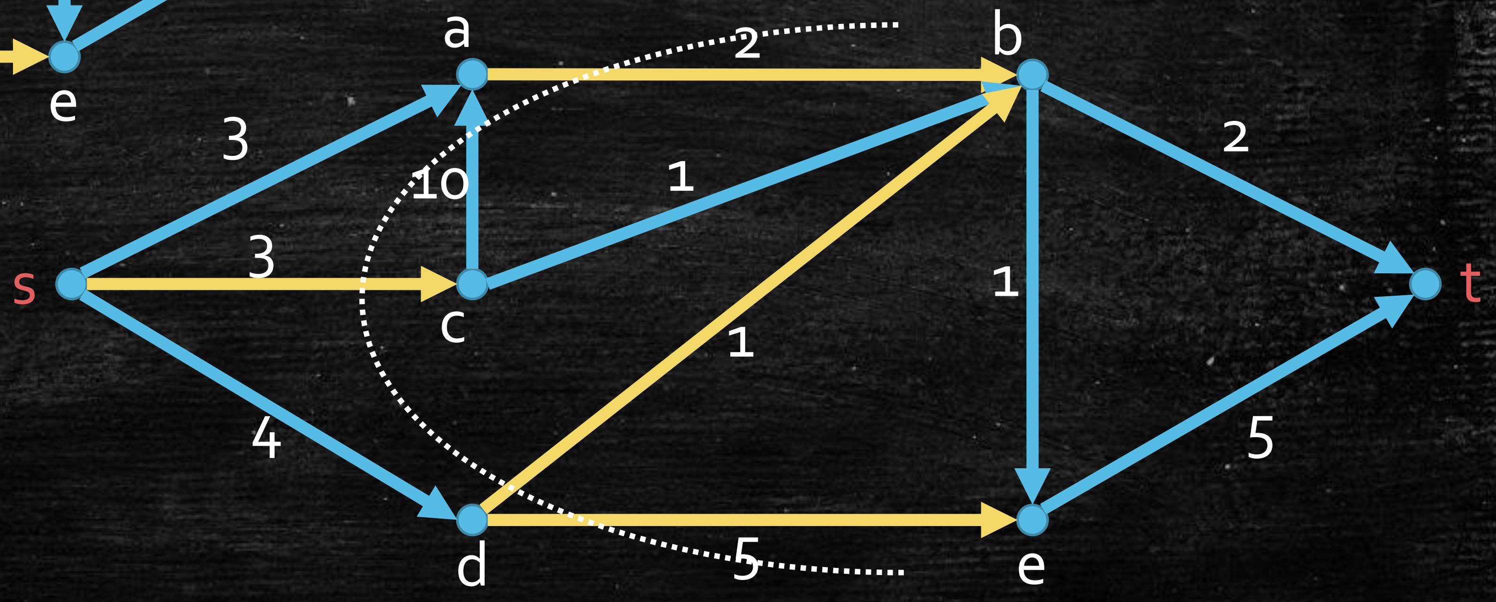


Current Flow ➡

In fact, every "cut" gives an upper-bound to $v(f)$.



$v(f) \leq 2 + 1 + 1 + 5 = 9$



$v(f) \leq 3 + 2 + 5 + 1 = 11$

The Minimum Cut Problem

- We want to find a tightest upper-bound to $v(f)$ by a carefully chosen cut.
- Given weighted graph $G = (V, E, w)$ and $s, t \in V$, an s-t cut is a partition of V to L, R such that $s \in L$ and $t \in R$.
- The value of the cut is defined by

$$c(L, R) = \sum_{(u,v)\in E, u\in L, v\in R} w(u, v)$$

- **Min-Cut Problem**: Given $G = (V, E, w)$ and $s, t \in V$, find the s-t cut with the minimum value.

Max-Flow-Min-Cut Theorem

- View the capacity $c(u, v)$ as the weight $w(u, v)$
- The value of every s-t cut is an upper-bound to $v(f)$.

Max-Flow-Min-Cut Theorem. The value of the maximum flow is exactly the value of the minimum cut:

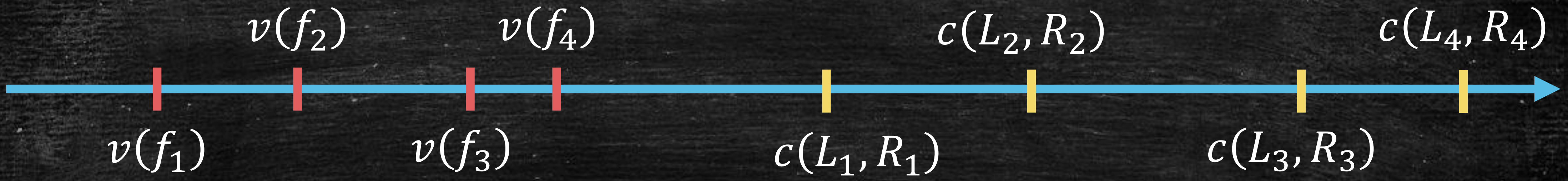
$$\max_f v(f) = \min_{L,R} c(L, R)$$

Proving Max-Flow-Min-Cut Theorem

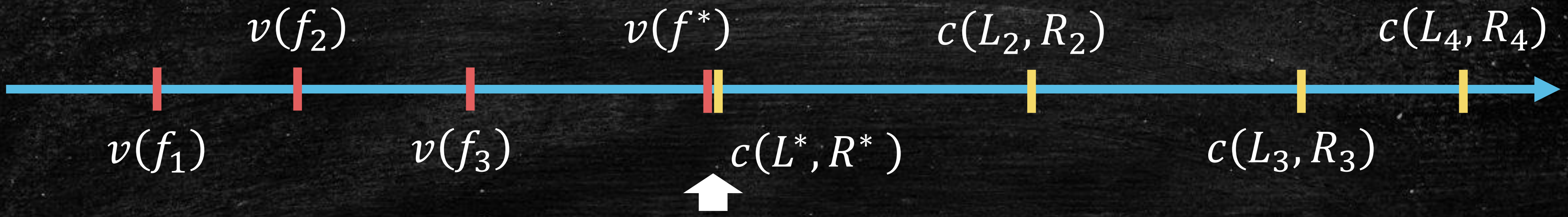
- **Lemma 1**. For any flow f and any cut $\{L, R\}$, we have $v(f) \leq c(L, R)$.
  - Formalize the idea that the value of any cut is an upper-bound to the value of any flow.
- **Lemma 2**. There exists a cut $\{L, R\}$ such that the flow f output by Ford-Fulkerson Algorithm satisfies $v(f) = c(L, R)$.
  - Concludes Max-Flow-Min-Cut Theorem.
  - Proves the correctness of Ford-Fulkerson Algorithm.

Proof of Max-Flow-Min-Cut Theorem

Lemma 1. For any flow f and any cut $\{L, R\}$, we have $v(f) \leq c(L, R)$.



Lemma 2. There exists a cut $\{L, R\}$ such that the flow f output by Ford-Fulkerson Algorithm satisfies $v(f) = c(L, R)$.



Max-Flow = Min-Cut

Proof of Lemma 1

Lemma 1. For any flow f and any cut $\{L, R\}$, we have $v(f) \leq c(L, R)$.

- Let $f(L, R)$ be the amount of flow going from L to R:

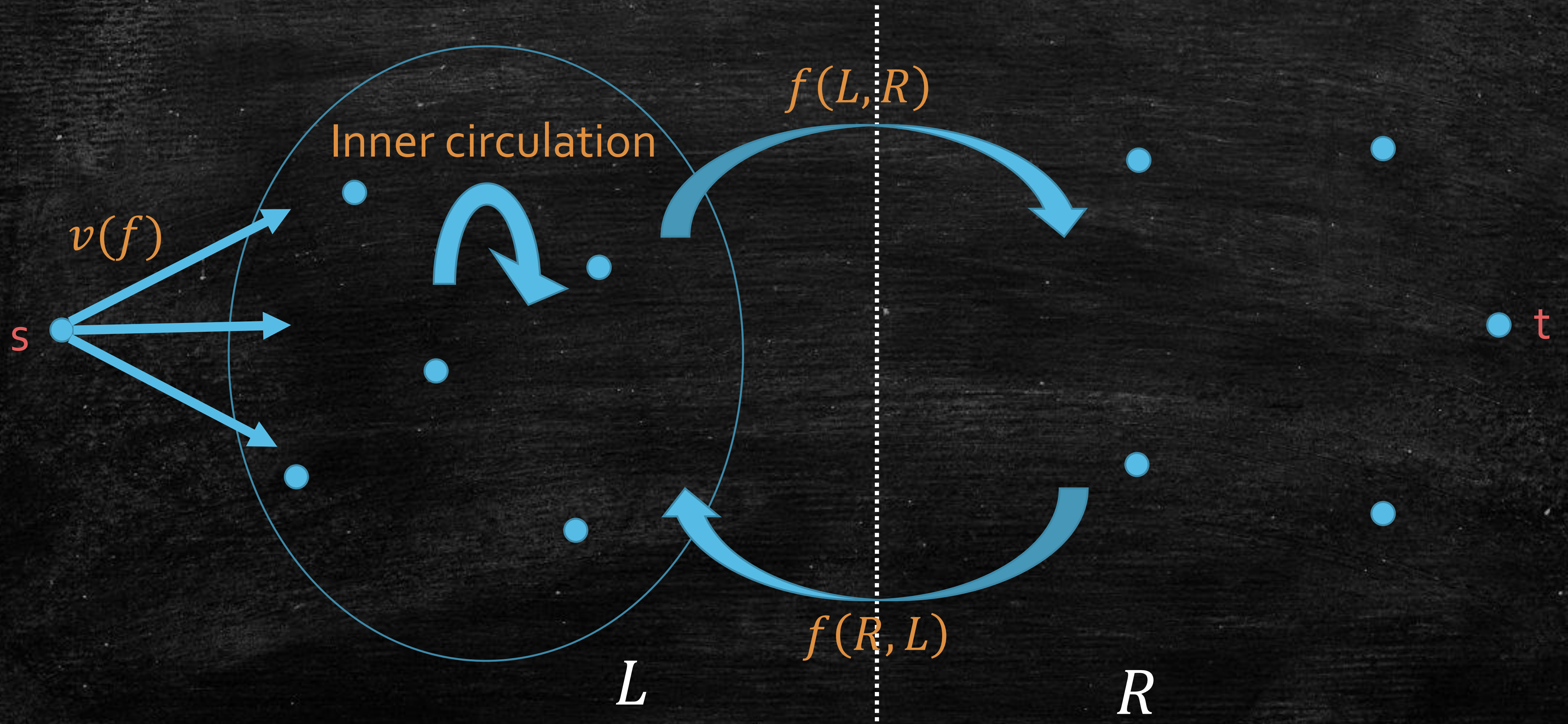
$$f(L, R) = \sum_{(u,v)\in E, u\in L, v\in R} f(u, v)$$

- Define $f(R, L)$ similarly.
- Claim: $v(f) = f(L, R) - f(R, L)$
  - Generalization of flow conservation.
- If the claim holds, Lemma 1 is proved:

$$v(f) \leq f(L, R) = \sum_{(u,v)\in E, u\in L, v\in R} f(u, v) \leq \sum_{(u,v)\in E, u\in L, v\in R} c(u, v) = c(L, R)$$

Proving generalized flow conservation

Claim: $v(f) = f(L,R) - f(R,L)$



Proving generalized flow conservation

Claim: $v(f) = f(L,R) - f(R,L)$

- Let $f^{\text{out}}(u) = \sum_{w:(u,w)\in E} f(u,w)$ be the amount of flow leaving u.
- Let $f^{\text{in}}(u) = \sum_{w:(w,u)\in E} f(w,u)$ be the amount of flow entering u.
- Flow conservation:
  - $f^{\text{out}}(u) = f^{\text{in}}(u)$ for $u \in V \setminus \{s,t\}$
  - $f^{\text{out}}(s) = v(f)$, $f^{\text{in}}(s) = 0$
- Summing up vertices in L:

$$\sum_{u\in L}\left(f^{\text{out}}(u) - f^{\text{in}}(u)\right) = f^{\text{out}}(s) + \sum_{u\in L\setminus\{s\}} 0 = v(f).$$

Proving generalized flow conservation

Claim: $v(f) = f(L,R) - f(R,L)$

- Summing up vertices in L:

$$\sum_{u \in L} \left(f^{\text{out}}(u) - f^{\text{in}}(u) \right) = f^{\text{out}}(s) + \sum_{u \in L \setminus \{s\}} 0 = v(f).$$

- Look at the summation again. Can you see the following?

$$\sum_{u \in L} \left(f^{\text{out}}(u) - f^{\text{in}}(u) \right) = \sum_{(u,v) \in E, u \in L, v \in R} f(u,v) - \sum_{(u,v) \in E, u \in R, v \in L} f(u,v)$$

- For each $f(u,v)$ with $u, v \in L$, it contributes $+f(u,v)$ to the summation by $f^{\text{out}}(u)$ and contributes $-f(u,v)$ by $f^{\text{in}}(v)$. Cancelled!
- For each $f(u,v)$ with $u \in L, v \in R$, it contributes $+f(u,v)$ to the summation.
- For each $f(u,v)$ with $u \in R, v \in L$, it contributes $-f(u,v)$ to the summation.

Proving generalized flow conservation

Claim: $v(f) = f(L,R) - f(R,L)$

- We have

$$\sum_{u \in L} \left(f^{\text{out}}(u) - f^{\text{in}}(u) \right) = f^{\text{out}}(s) + \sum_{u \in L \setminus \{s\}} 0 = v(f)$$

- and

$$\sum_{u \in L} \left(f^{\text{out}}(u) - f^{\text{in}}(u) \right) = \sum_{(u,v) \in E, u \in L, v \in R} f(u,v) - \sum_{(u,v) \in E, u \in R, v \in L} f(u,v)$$

- Putting together:

$$v(f) = \sum_{(u,v) \in E, u \in L, v \in R} f(u,v) - \sum_{(u,v) \in E, u \in R, v \in L} f(u,v) = f(L,R) - f(R,L)$$

Proof of Lemma 1

Lemma 1. For any flow f and any cut $\{L, R\}$, we have $v(f) \leq c(L, R)$.

- Claim: $v(f) = f(L, R) - f(R, L)$
  - Generalization of flow conservation.
- Proof of Lemma 1:

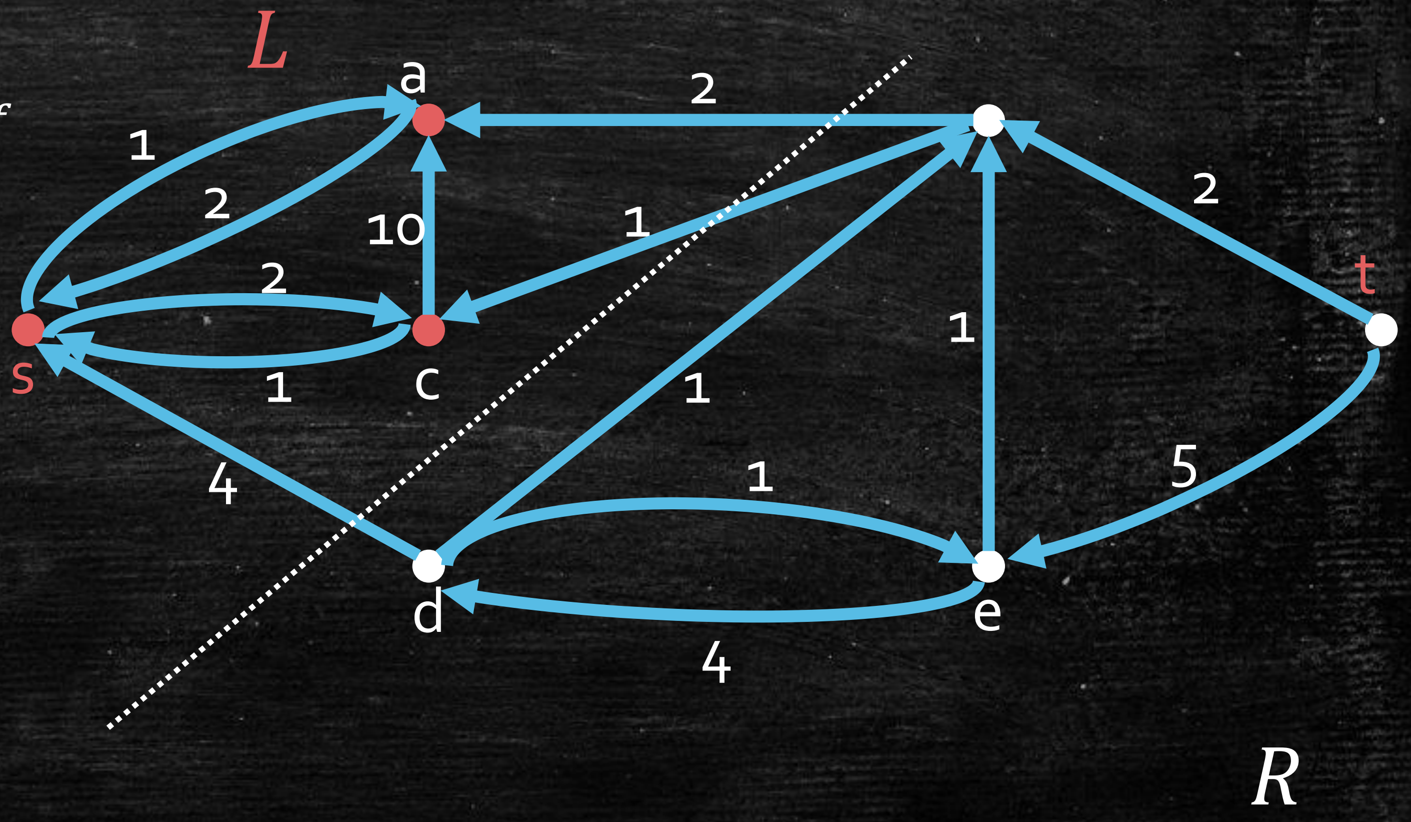
$$v(f) \leq f(L, R) = \sum_{(u,v)\in E, u\in L, v\in R} f(u, v) \leq \sum_{(u,v)\in E, u\in L, v\in R} c(u, v) = c(L, R)$$

Proof of Lemma 2

Lemma 2. There exists a cut $\{L, R\}$ such that the flow f output by Ford-Fulkerson Algorithm satisfies $v(f) = c(L, R)$.

- f: output of Ford-Fulkerson
- L: vertices reachable from s in G^f
- $R = V \setminus L$
- Claim A: $f(L, R) = c(L, R)$
- Claim B: $f(R, L) = 0$
- $v(f) = f(L, R) - f(R, L) = c(L, R)$

Residual Network G^f



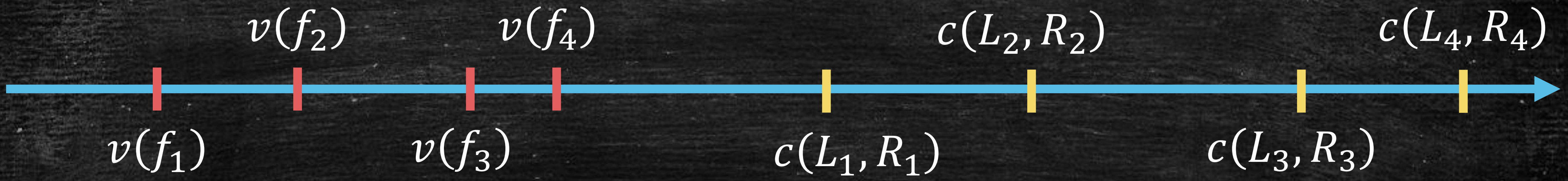
Proof of Lemma 2

Lemma 2. There exists a cut $\{L, R\}$ such that the flow f output by Ford-Fulkerson Algorithm satisfies $v(f) = c(L, R)$.

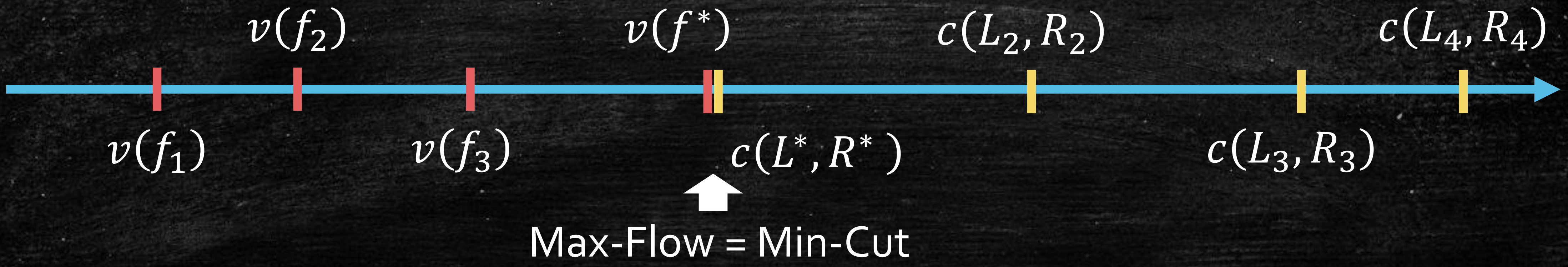
- Claim A: $f(L, R) = c(L, R)$
  - Otherwise, exist (u, v) with $u \in L, v \in R$ such that $f(u, v) < c(u, v)$
  - Thus, (u, v) is in G^f and v is reachable from s
  - Contradict to $v \in R$ by our definition of L
- Claim B: $f(R, L) = 0$
  - Otherwise, exist (v, u) with $u \in L, v \in R$ such that $f(v, u) > 0$
  - Thus, (u, v) is in G^f and v is reachable from s
  - Contradict to $v \in R$ by our definition of L

Proof of Max-Flow-Min-Cut Theorem

Lemma 1. For any flow f and any cut $\{L, R\}$, we have $v(f) \leq c(L, R)$.



Lemma 2. There exists a cut $\{L, R\}$ such that the flow f output by Ford-Fulkerson Algorithm satisfies $v(f) = c(L, R)$.



Algorithm for finding a minimum cut

Min-Cut Problem: Given $G = (V, E, w)$ and $s, t \in V$, find the s-t cut with the minimum value.

- Solve the max-flow problem with $\forall(u, v) \in E$: $c(u, v) = w(u, v)$
- Let f be the maximum flow and construct G^f
- L: vertices reachable from s in G^f
- $R = V \setminus L$
- Return $\{L, R\}$

Recap

- Question: Ford-Fulkerson Correctness?
- Any cut provides an upper bound for the max-flow (Lemma 1)
- Ford-Fulkerson can find a flow whose value equals to the value of the min-cut (Lemma 2)
- This proves max-flow-min-cut theorem
- This also proves the correctness of Ford-Fulkerson

Time Complexity?

- Correctness: Max-Flow-Min-Cut Theorem ✓
- Time Complexity:
  - Question 1: Does the algorithm always halt?
  - Question 2: If so, what is the time complexity?

Ford-Fulkerson Algorithm: Time Complexity?

Does the algorithm always halt?

- Let's start from simplest case: all the capacities are integers.
- Each while-loop iteration increases the value of f by at least 1.
- Thus, the algorithm will halt within f_{max} iterations.

Integrality Theorem

- **Theorem**. If each $c(e)$ is an integer, then there exists a maximum flow f such that $f(e)$ is an integer for each e.
- *Proof*. For each edge e, the value of $f(e)$ is always an integer throughout Ford-Fulkerson Algorithm.

Does the algorithm always halt?

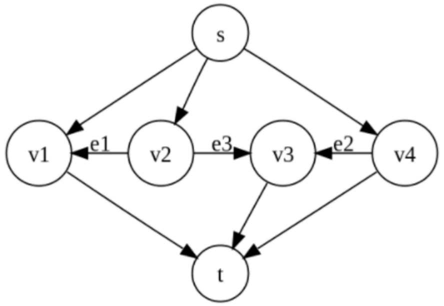
- How about rational capacities?
- Rescale capacities to make them integers.
- Yes, the algorithm will halt!

Does the algorithm always halt?

- How about possibly irrational capacities?
- No, the algorithm does not always halt!

Non-terminating example [edit]

Consider the flow network shown on the right, with source s, sink t, capacities of edges e_1, e_2 and e_3 respectively 1, $r = (\sqrt{5}-1)/2$ and 1 and the capacity of all other edges some integer $M \geq 2$. The constant r was chosen so, that $r^2 = 1 - r$. We use augmenting paths according to the following table, where $p_1 = \{s, v_4, v_3, v_2, v_1, t\}$, $p_2 = \{s, v_2, v_3, v_4, t\}$ and $p_3 = \{s, v_1, v_2, v_3, t\}$.



Step	Augmenting path	Sent flow	Residual capacities		
			e_1	e_2	e_3
0			$r^0 = 1$	r	1
1	$\{s, v_2, v_3, t\}$	1	r^0	r^1	0
2	p_1	r^1	r^2	0	r^1
3	p_2	r^1	r^2	r^1	0
4	p_1	r^2	0	r^3	r^2
5	p_3	r^2	r^2	r^3	0

Note that after step 1 as well as after step 5, the residual capacities of edges e_1, e_2 and e_3 are in the form r^n, r^{n+1} and 0, respectively, for some $n \in \mathbb{N}$. This means that we can use augmenting paths p_1, p_2, p_1 and p_3 infinitely many times and residual capacities of these edges will always be in the same form. Total flow in the network after step 5 is $1 + 2(r^1 + r^2)$. If we continue to use augmenting paths as above, the total flow converges to $1 + 2\sum_{i=1}^{\infty} r^i = 3 + 2r$. However, note that there is a flow of value $2M + 1$, by sending M units of flow along sv_1t, 1 unit of flow along sv_2v_3t, and M units of flow along sv_4t. Therefore, the algorithm never terminates and the flow does not even converge to the maximum flow.[4]

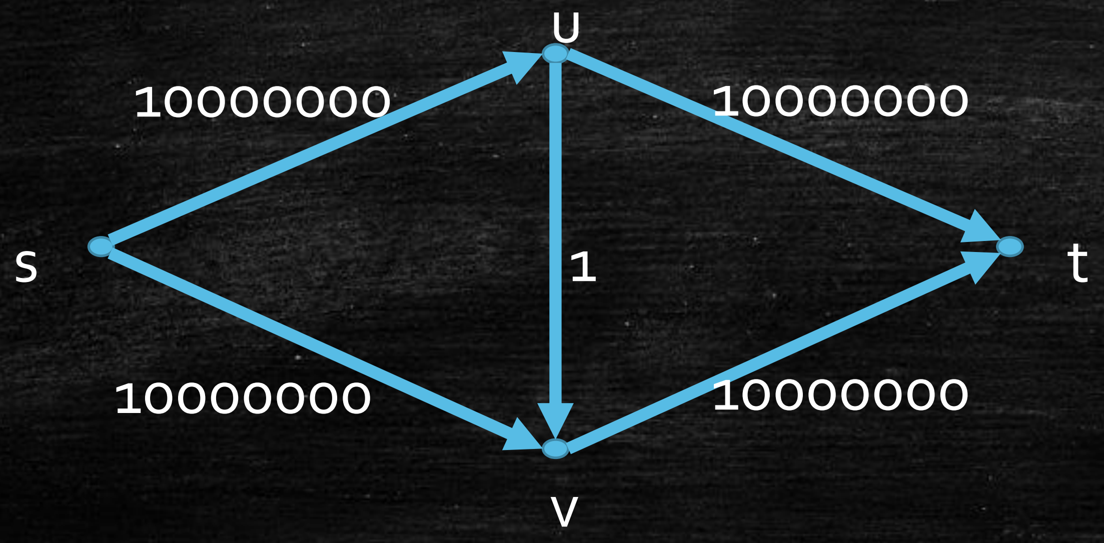
Another non-terminating example based on the Euclidean algorithm is given by Backman & Huynh (2018), where they also show that the worst case running-time of the Ford-Fulkerson algorithm on a network $G(V, E)$ in ordinal numbers is $\omega^{\Theta(|E|)}$.

Time Complexity?

- Assume all capacities are integers, what is the time complexity?
- Each iteration requires $O(|E|)$ time:
  - $O(|E|)$ is sufficient for finding p, updating f and G^f
- There are at most f_{max} iterations.
- Overall: $O(|E| \cdot f_{max})$
- Can we analyze it better?

Time Complexity?

- Can we analyze it better?
- It depends on how you choose p in each iteration!
- The complexity bound $O(|E| \cdot f_{max})$ is tight for arbitrary choices!



Class Activity 2

- For integer capacities, the time complexity for Ford-Fulkerson algorithm is $O(|E| \cdot f_{max})$.
- Does Ford-Fulkerson algorithm run in polynomial time?

Method vs Algorithm

- Different choices of augmenting paths p give different implementation of Ford-Fulkerson.
- The description of Ford-Fulkerson Algorithm is incomplete.
- For this reason, it is sometimes called Ford-Fulkerson Method.
- Next Lecture: Edmonds-Karp Algorithm
  - Careful choices of augmenting paths
  - Polynomial time

This Lecture

- Max-Flow Problem
- Ford-Fulkerson Algorithm (Method)
- Max-Flow-Min-Cut Theorem
  - Correctness of Ford-Fulkerson Algorithm
- Flow Integrality Theorem
  - Integral Capacities imply integral flow