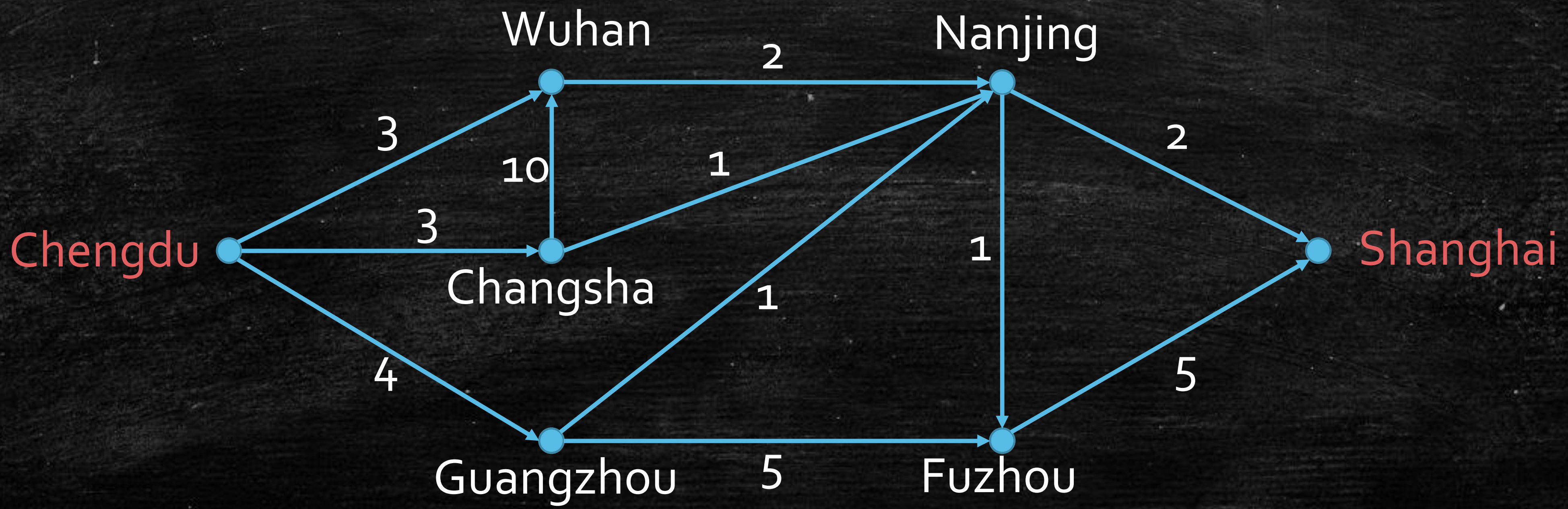
Network Flow

Maximum Flow Problem, Ford-Fulkerson Algorithm, Max-Matching on Bipartite Graphs

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?



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 $u \in V \setminus \{s, t\}$,

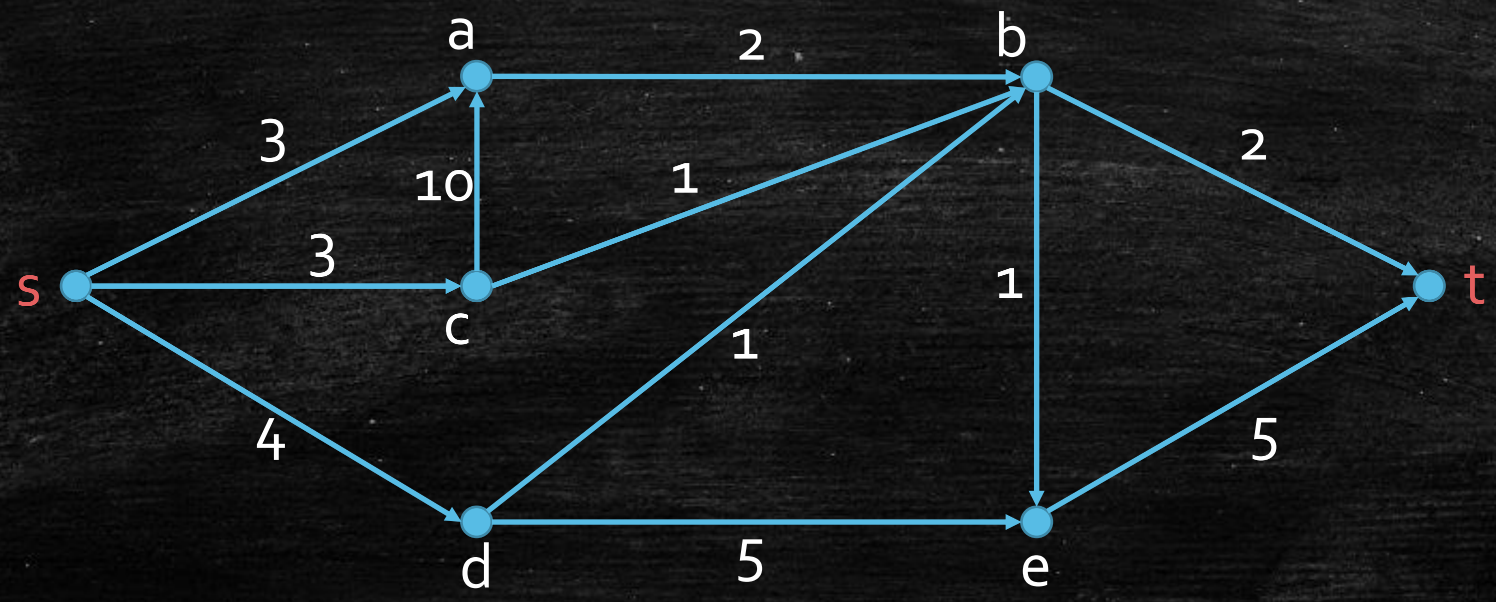
$$\sum_{v:(u,v)\in E} f(v, u) = \sum_{w:(u,w)\in E} f(u, w).$$

The value of the flow is defined as

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

More Applications

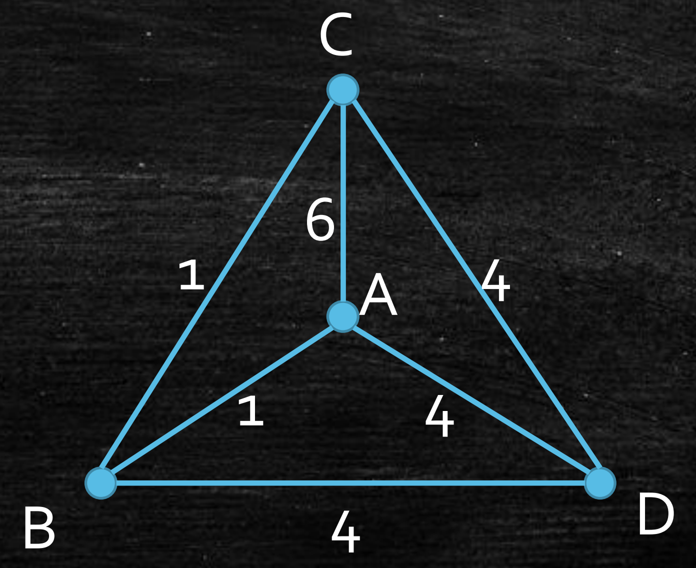
- We want to build a data transmission channel from s to t.
- We can use intermediate routers a, b, c, d, e.
- Each edge has a bandwidth, limiting the maximum rate of data transmission.
- What is the maximum rate of data that can be transferred?



More Applications

- Table describes number of matches each team has won.
- Number on each edge represents number of remaining matches.
- Does Team D have a chance for the champion?

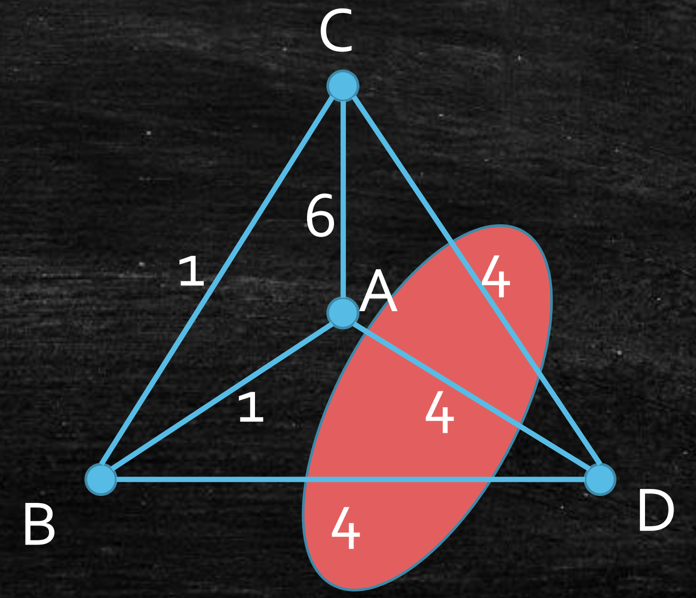
	Wins
A	40
B	38
C	37
D	29



More Applications

- Let us first assume Team D wins all the 12 remaining matches.

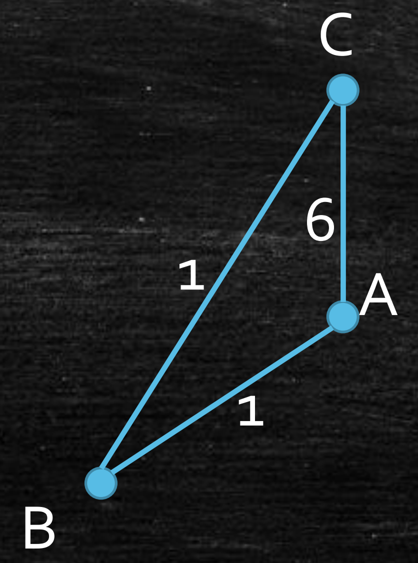
	Wins
A	40
B	38
C	37
D	29+12=41



More Applications

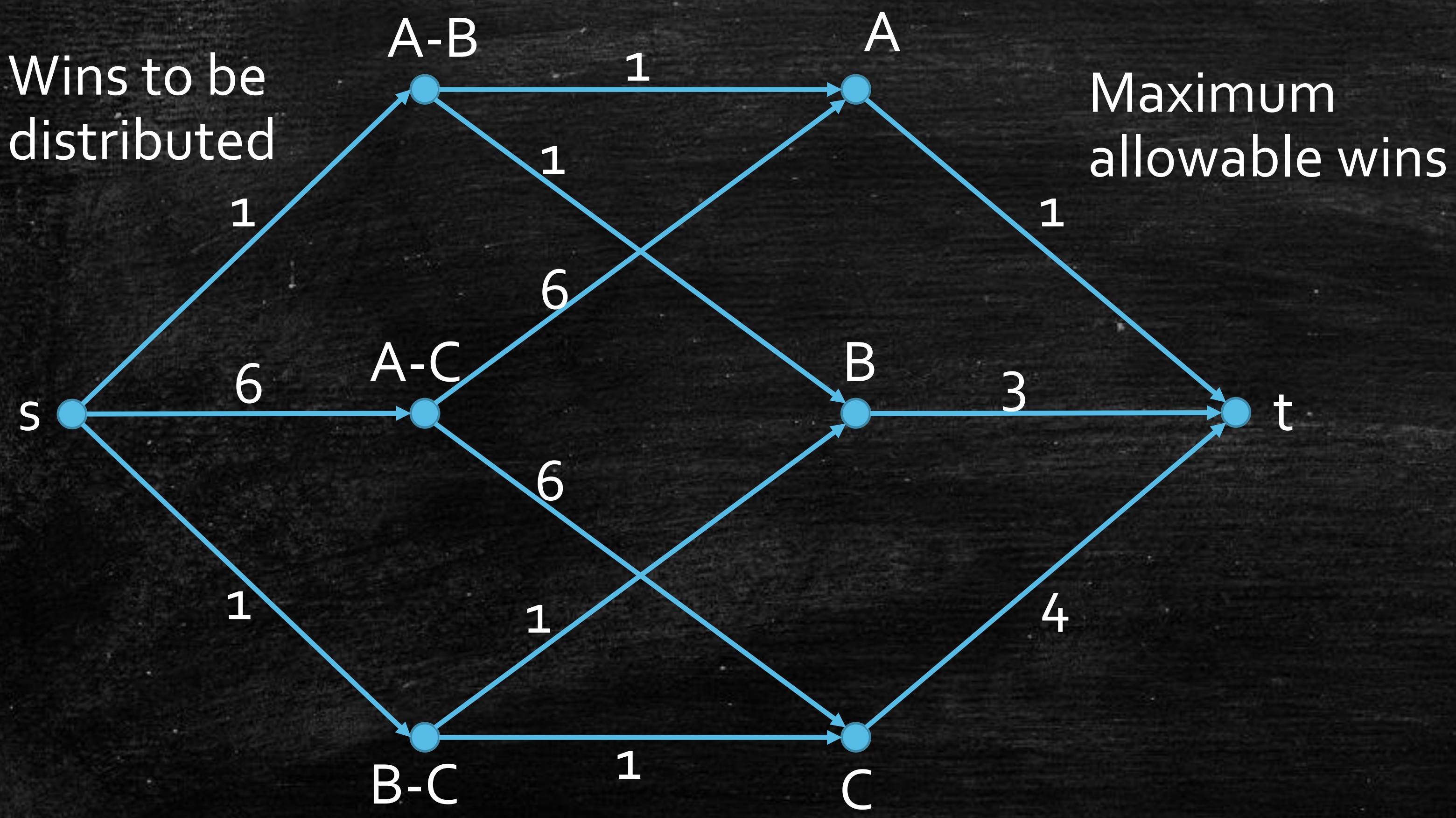
- Team A must win at most 1
- Team B must win at most 3
- Team C must win at most 4

	Wins
A	40
B	38
C	37
D	41

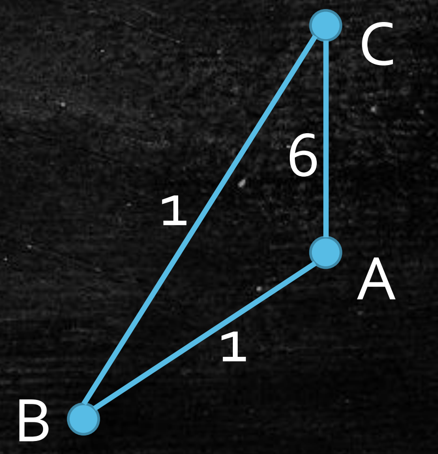


More Applications

- Model the problem as Max-Flow.

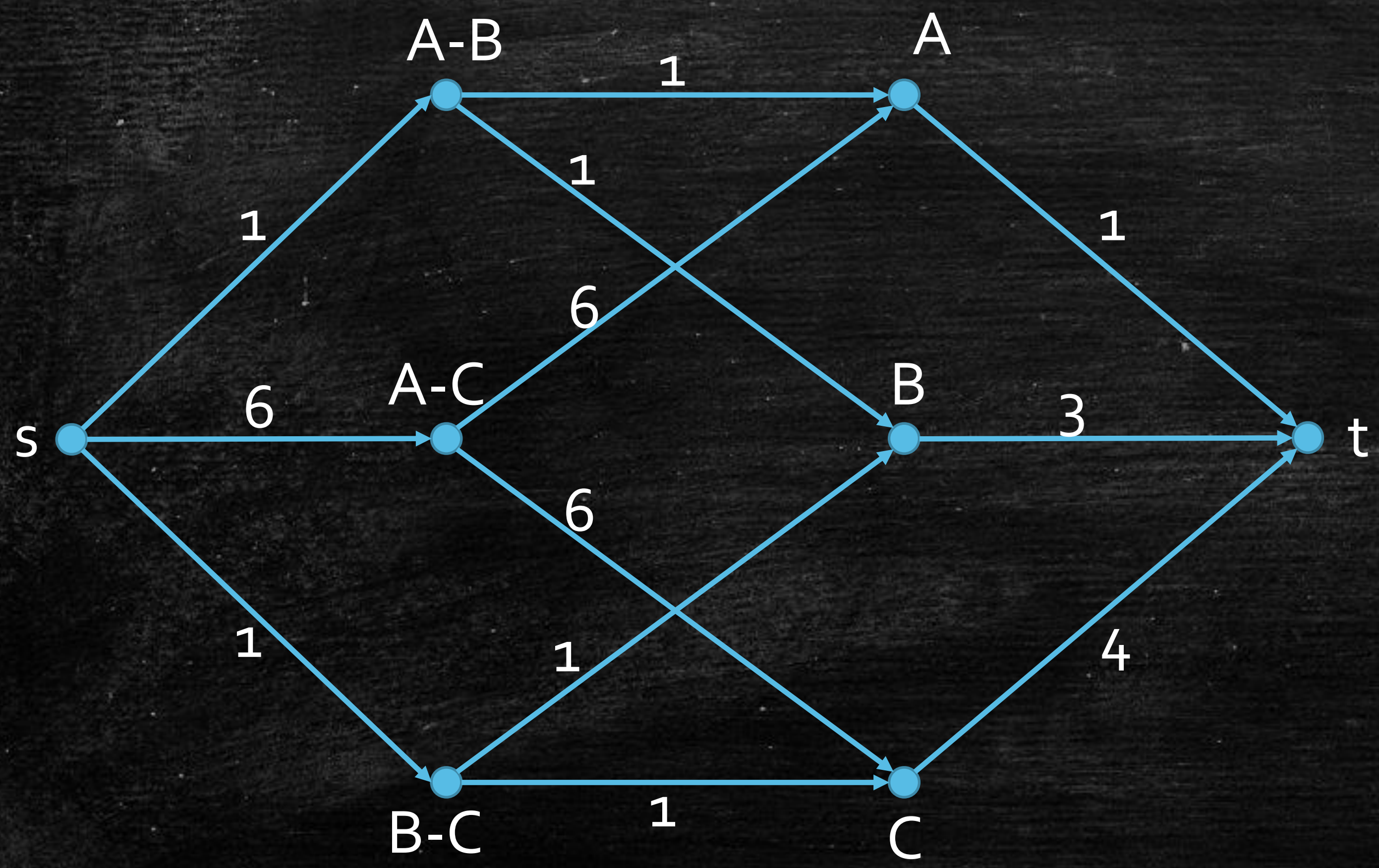


	Wins	Max Num of Additional Wins
A	40	1
B	38	3
C	37	4
D	41	



More Applications

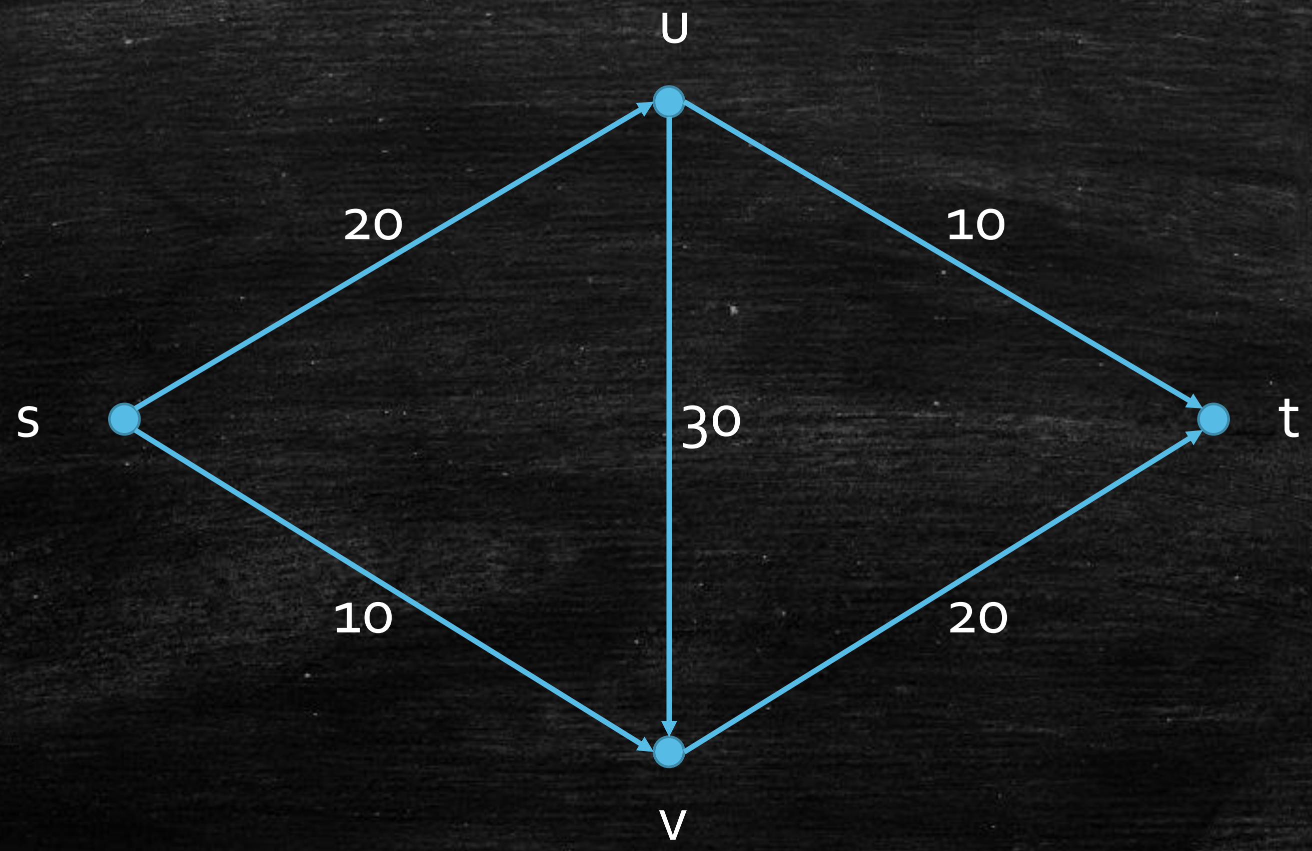
- If Team D has a chance for championship, the maximum flow should be 1+6+1=8.



	Wins	Max Num of Additional Wins
A	40	1
B	38	3
C	37	4
D	41	

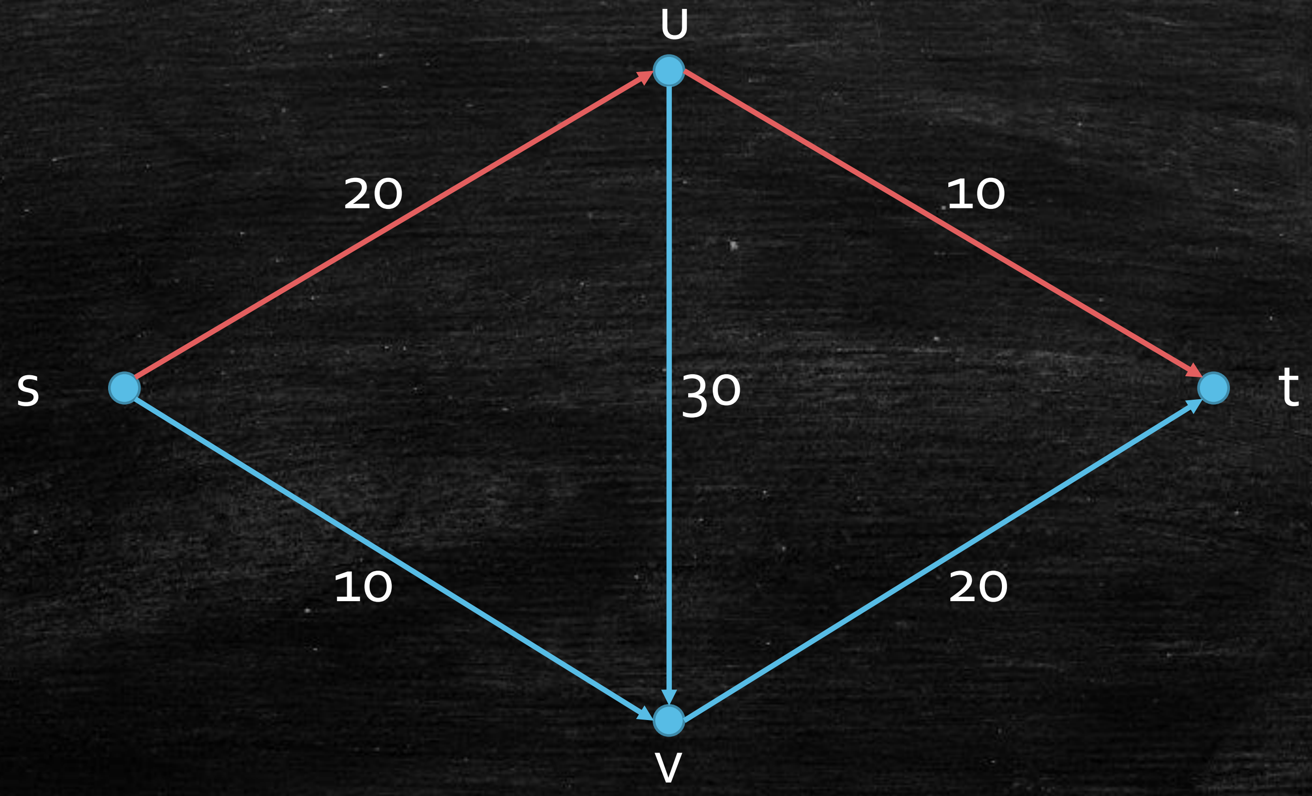
A Greedy Attempt

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



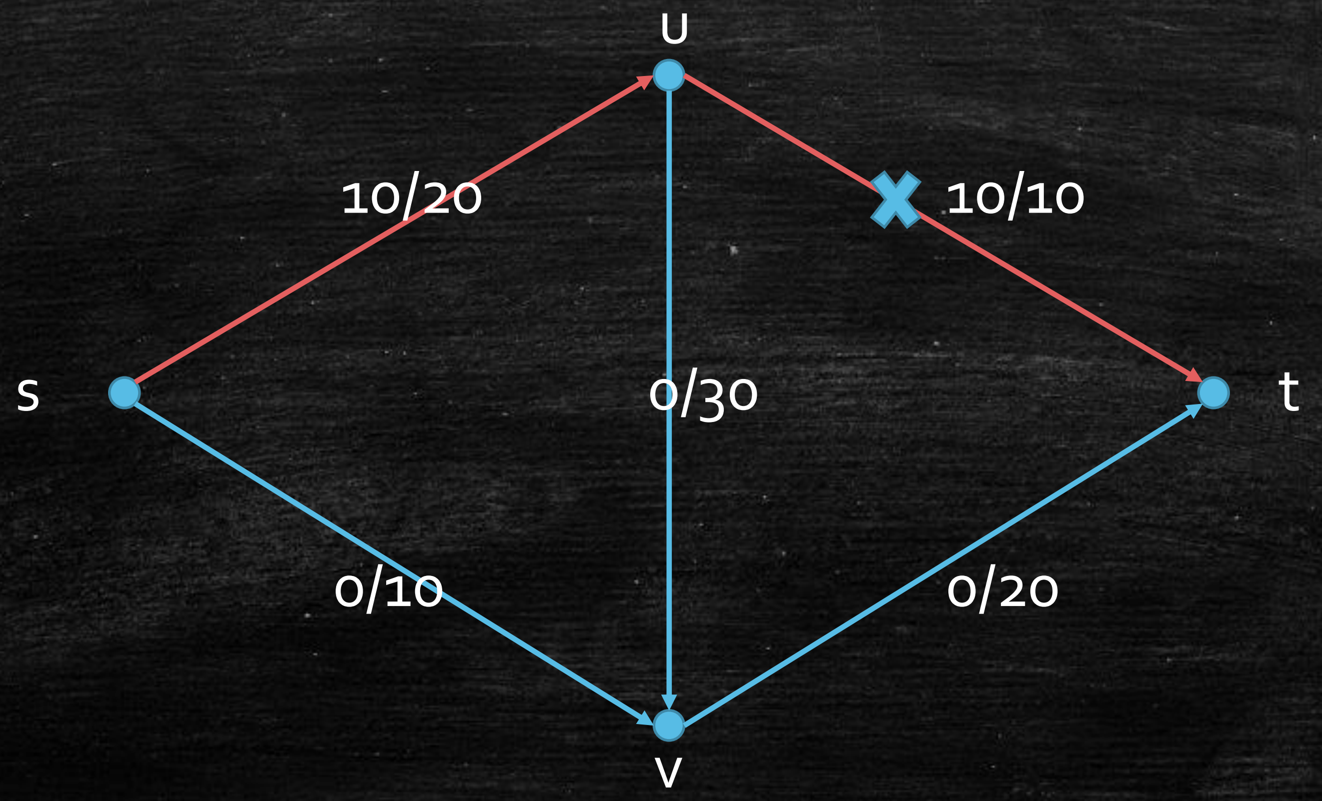
A Greedy Attempt

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



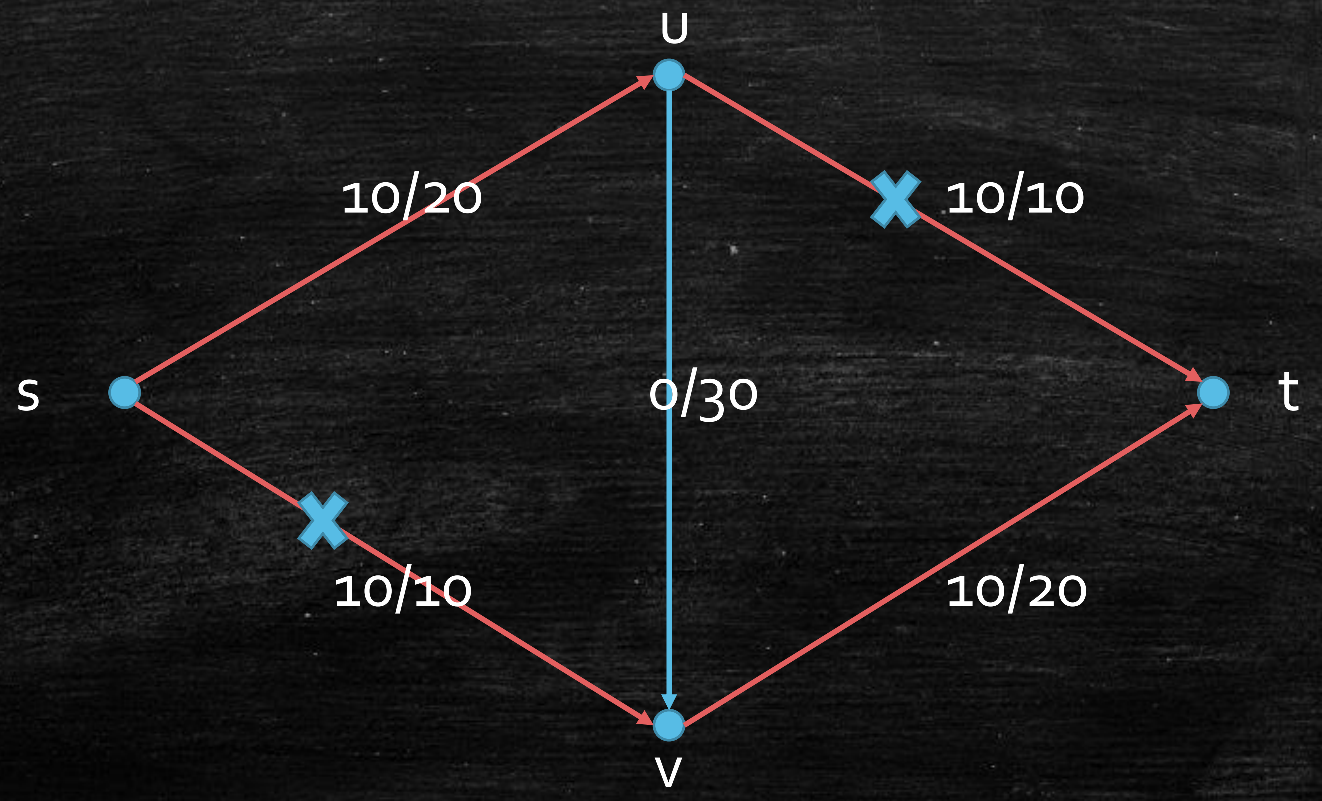
A Greedy Attempt

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



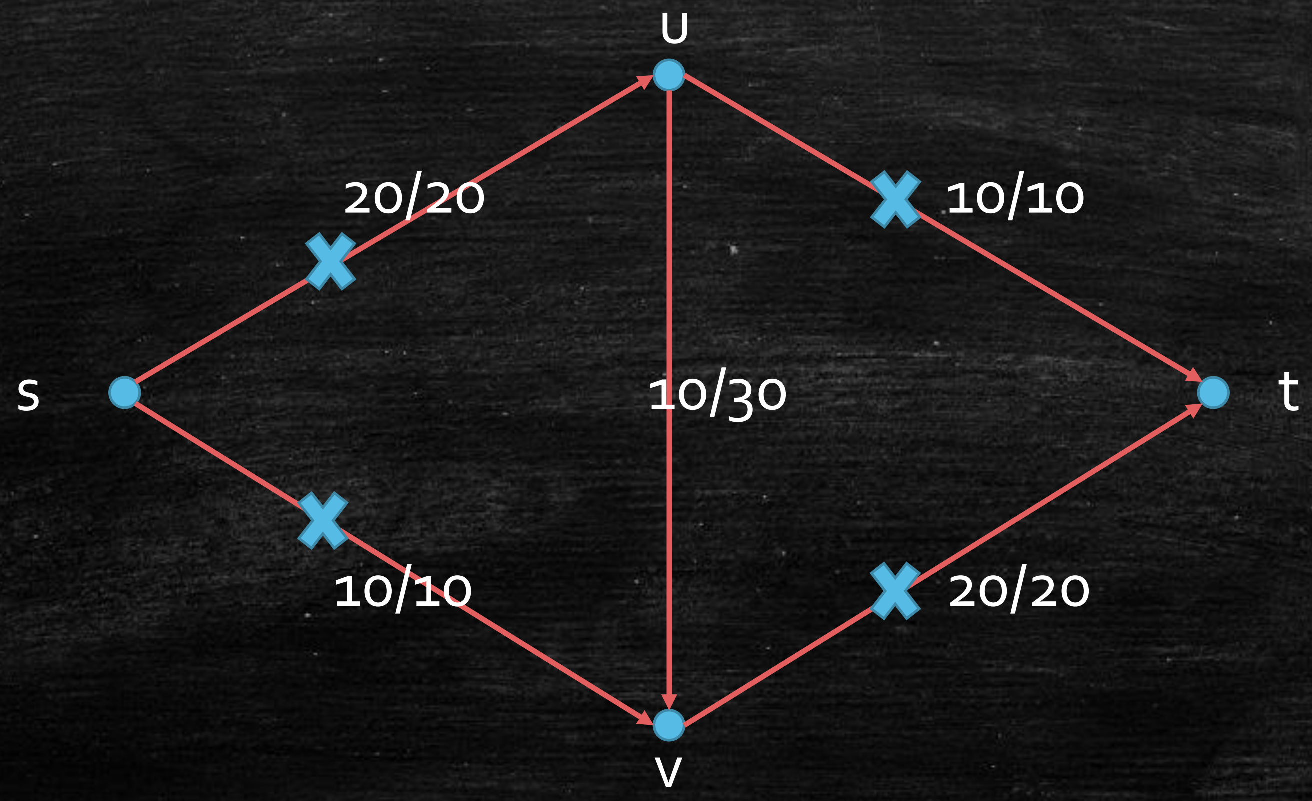
A Greedy Attempt

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



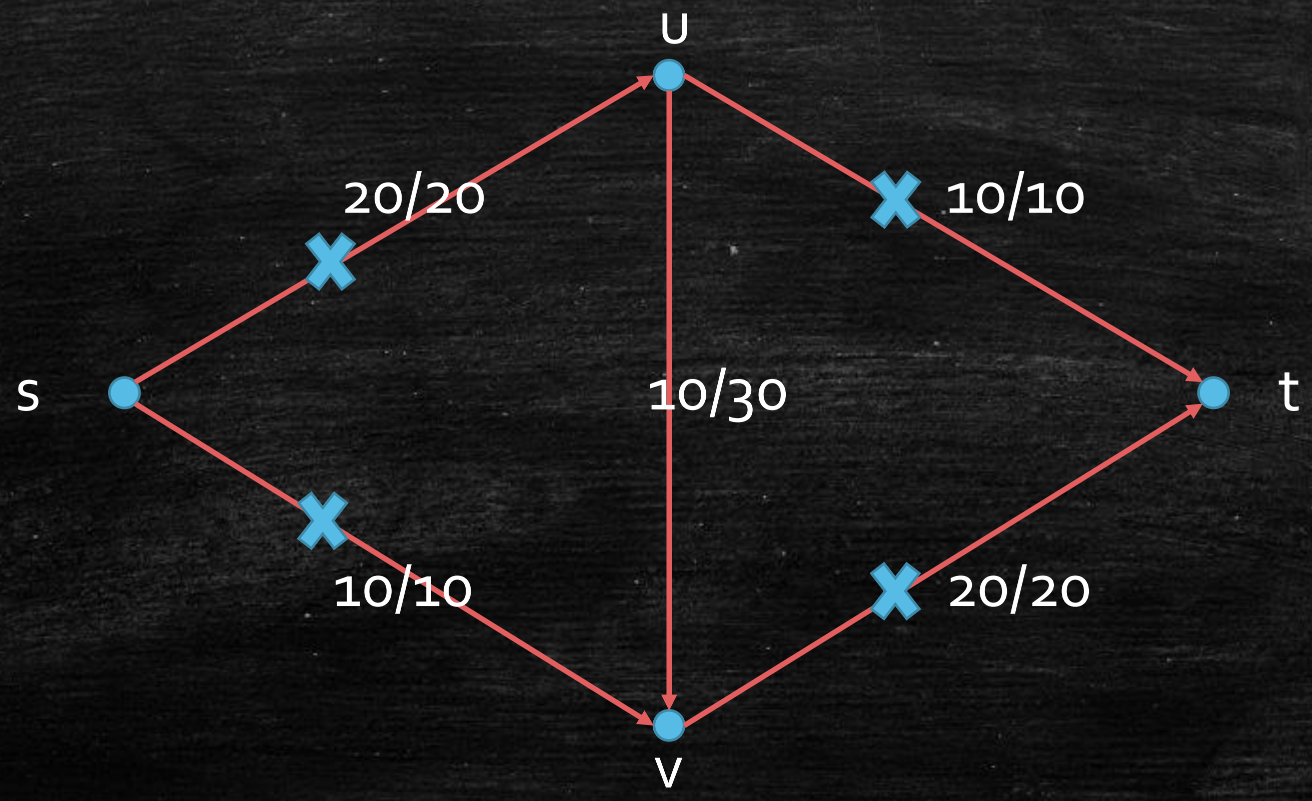
A Greedy Attempt

- 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



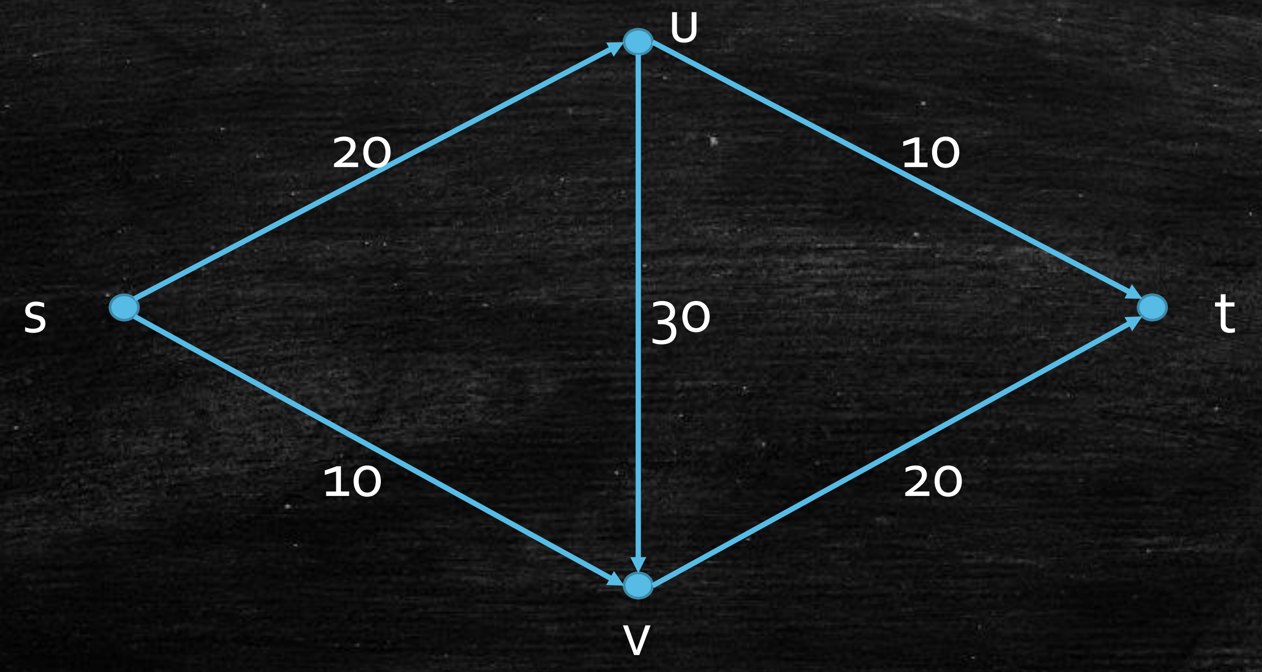
A Greedy Attempt

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



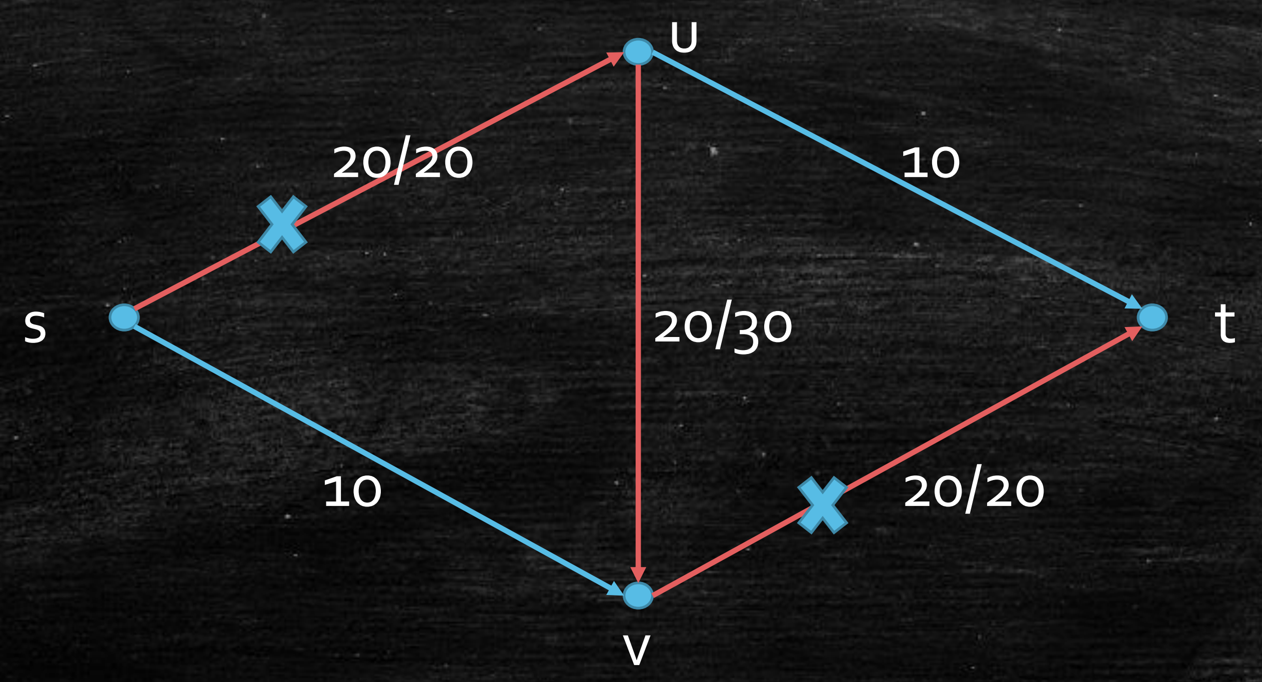
A Greedy Attempt

- 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?



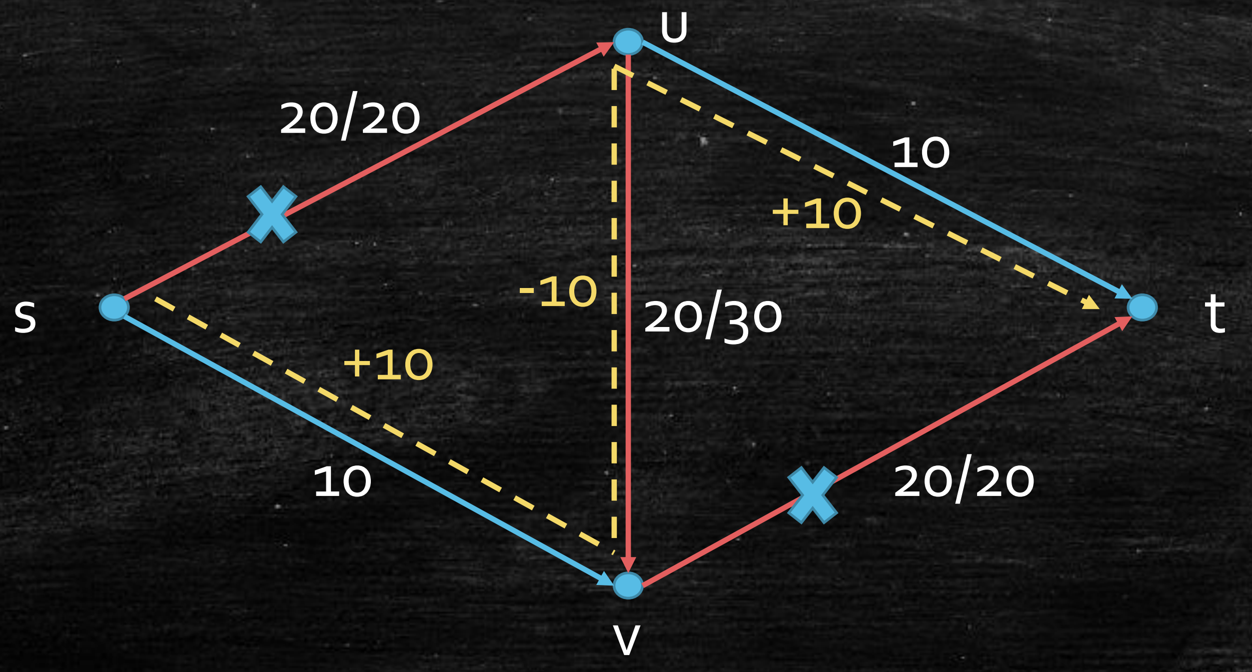
A Greedy Attempt

- 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?



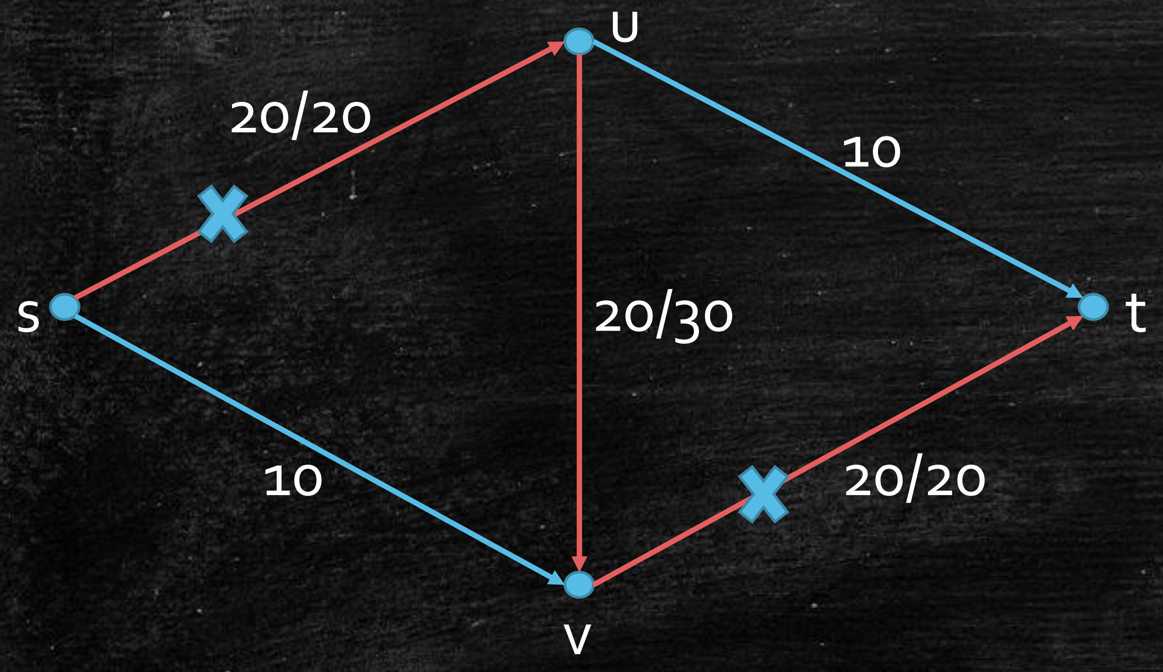
Flow “Cancellation”

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

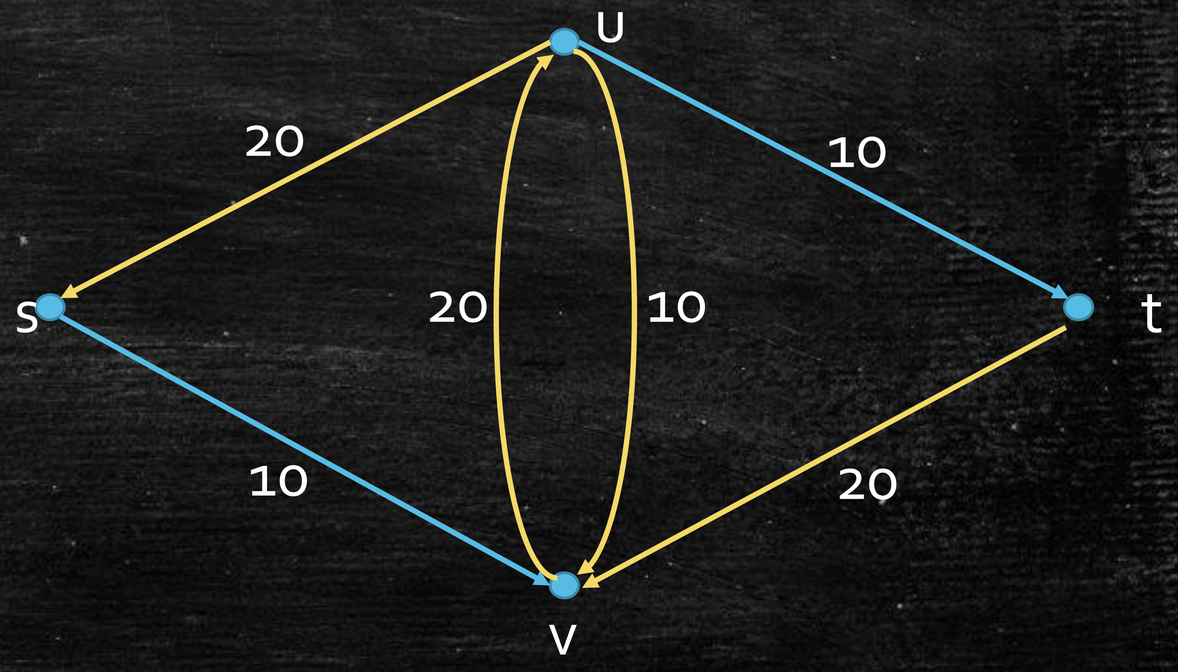


Residual Network

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



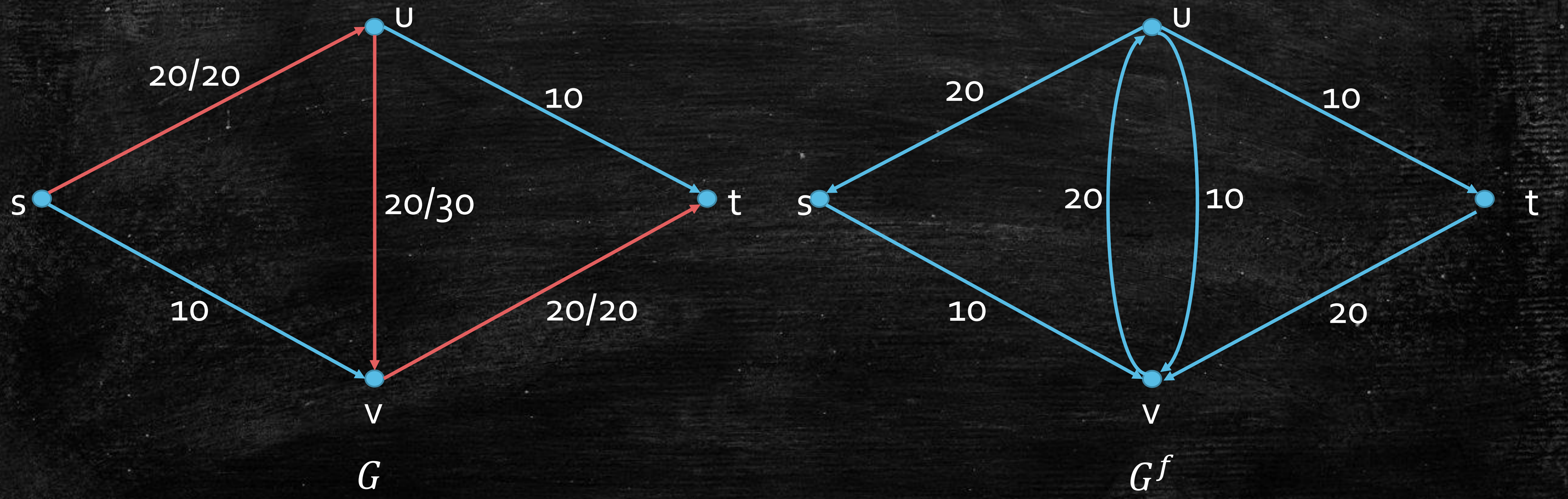
G



G^f

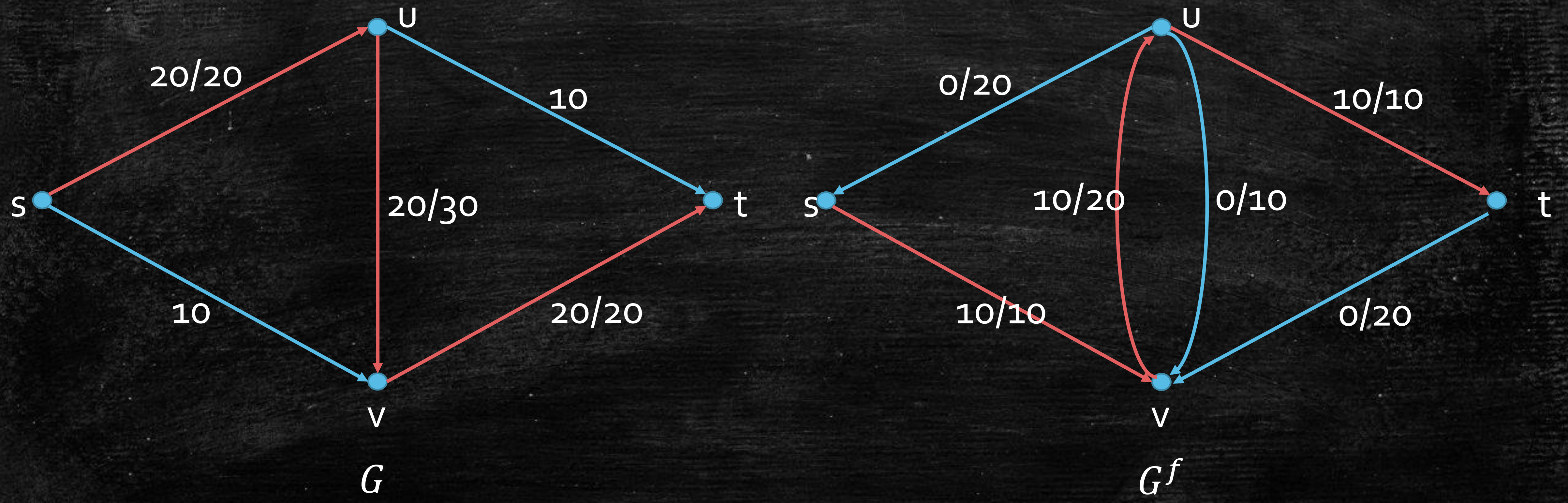
Residual Network

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



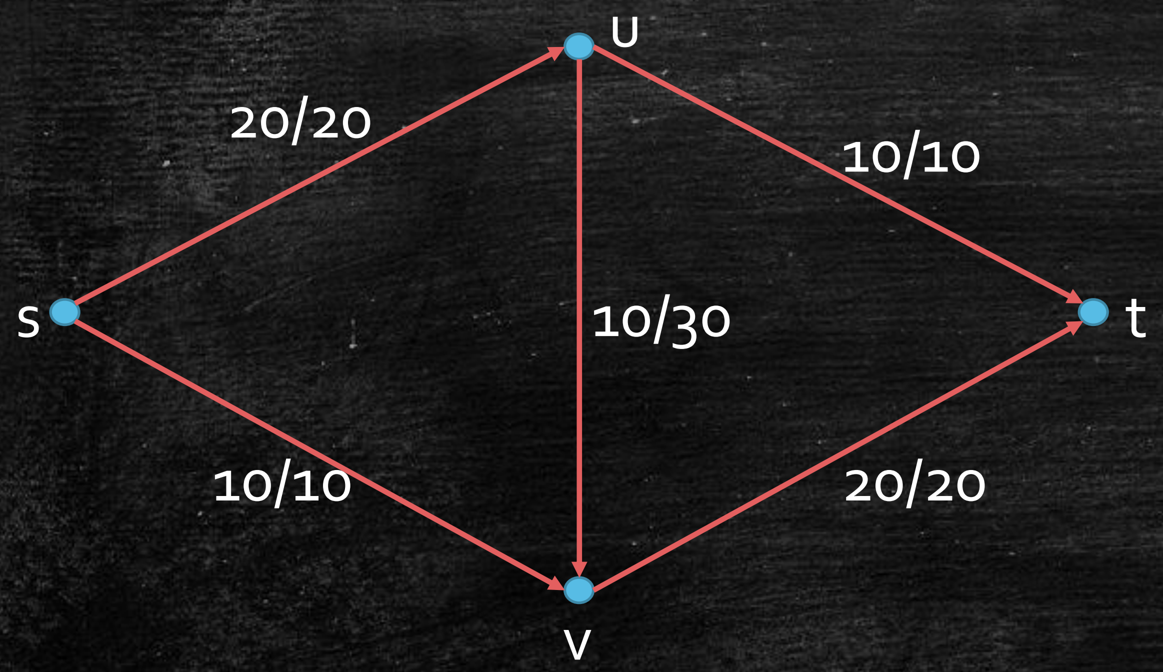
Residual Network

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



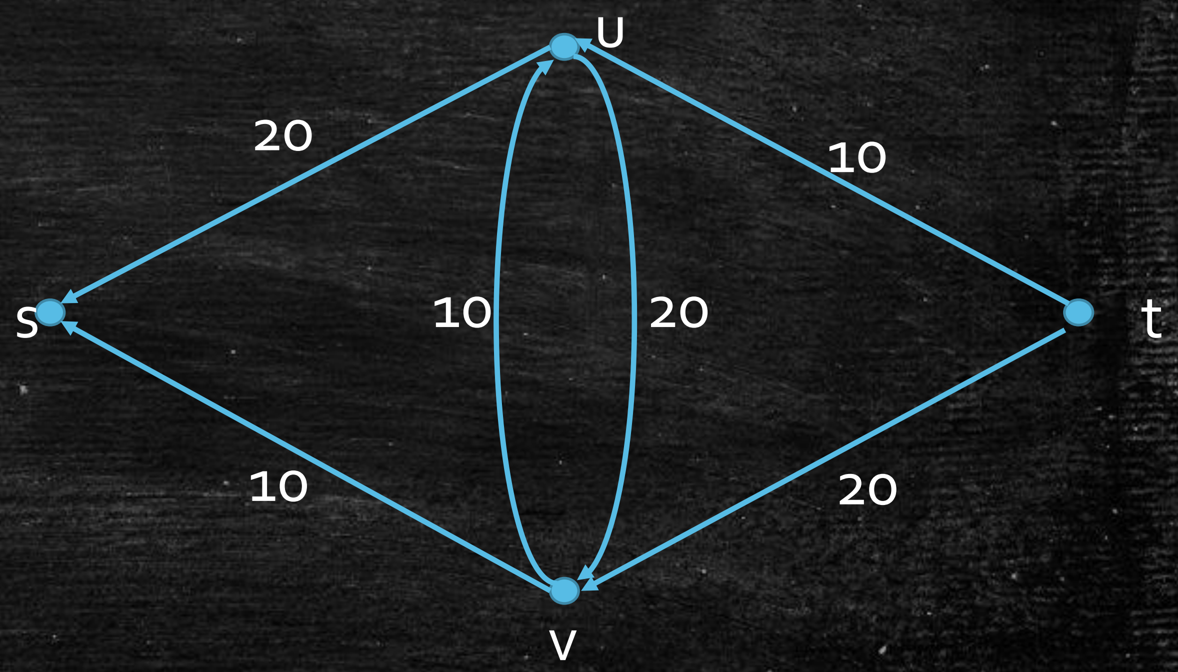
Residual Network

original graph G.



G

residual graph G^f.



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

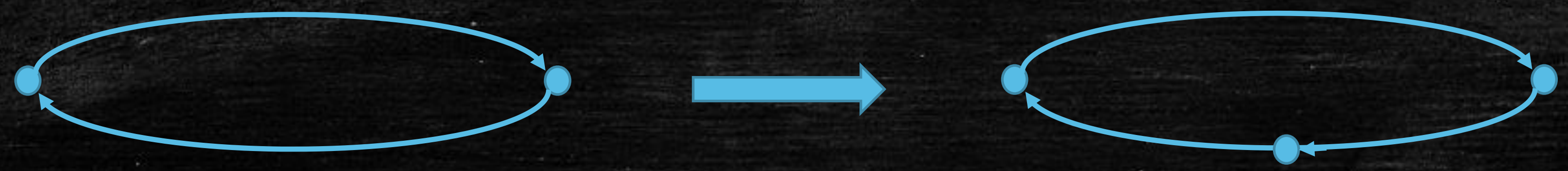
FordFulkerson$(G = (V, E), s, t, c)$:

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

A Small Bug…

4. **for each** $e = (u, v) \in p$:
5.   if $(u, v) \in E$: update $f(e) \leftarrow f(e) + b$;
6.   if $(v, u) \in E$: update $f(e) \leftarrow f(e) - b$;
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?

Does the algorithm always halt?

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

- **Theorem**. If each $c(e)$ is an integer, then the value of the maximum flow f is an integer.
- *Proof.* The value of f is always an integer throughout Ford-Fulkerson Algorithm.

Does the algorithm always halt?

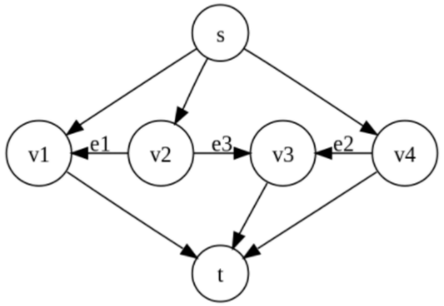
- How about rational capacities?
- Rescale capacities to make them integers.
- Yes, the algorithm will halt!

Does the algorithm always halt?

- How about possibly irrational capacities?
- No, the algorithm do 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]

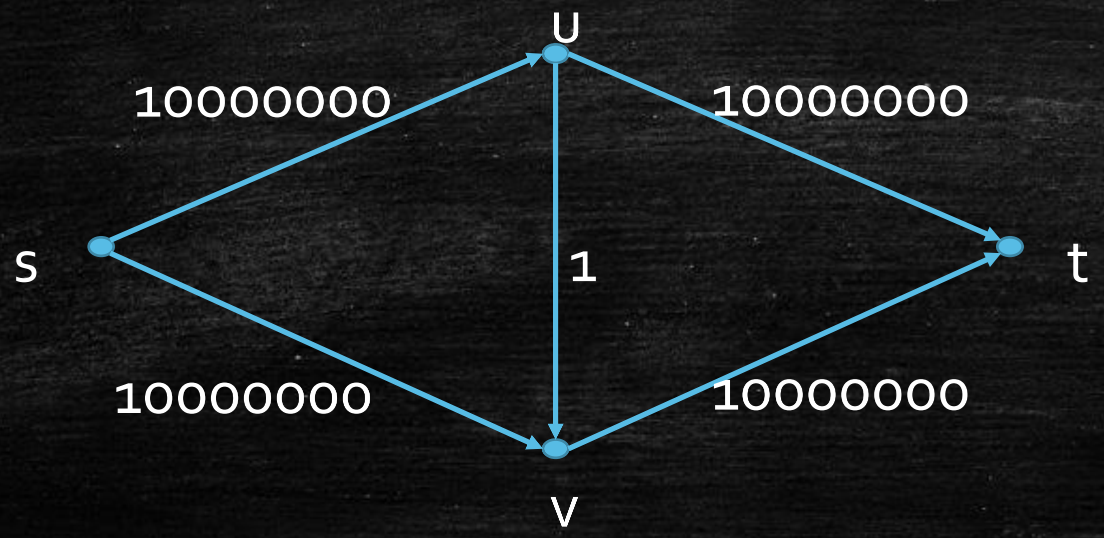
Another non-terminating example based on the Euclidean algorithm is given by Backman & Huynh (2018), where they also show that the worst case running-time of the Ford-Fulkerson algorithm on a network $G(V, E)$ in ordinal numbers is $\omega^{\Theta(|E|)}$.

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!



Method vs Algorithm

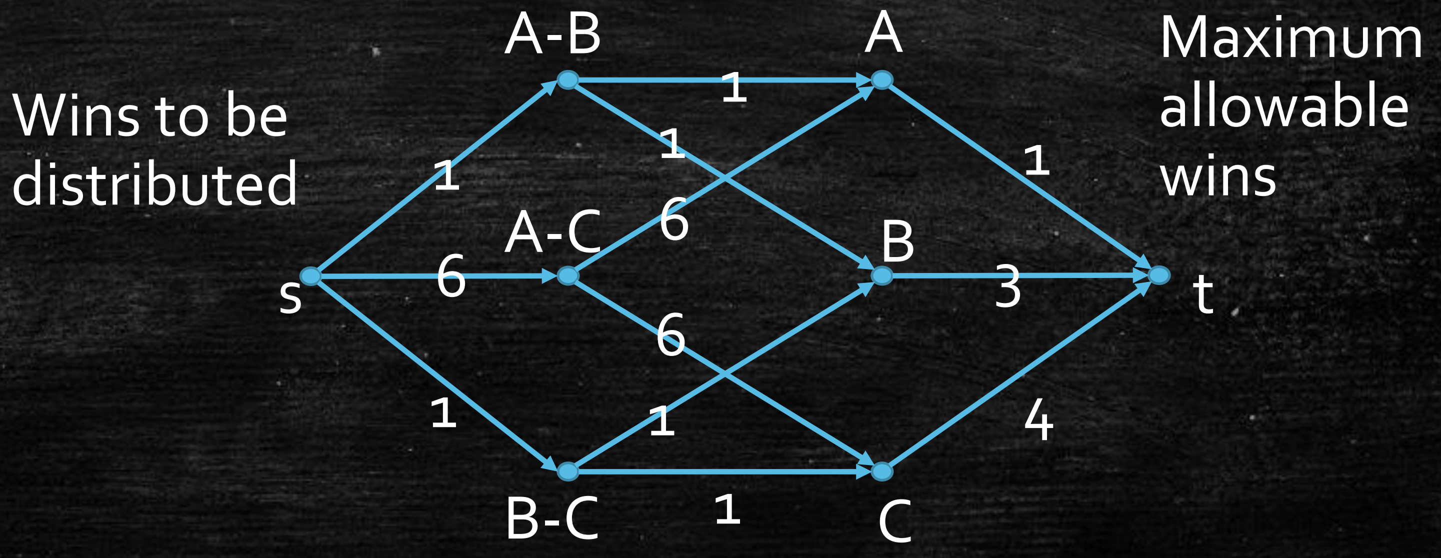
- Different choices of augmenting paths p give different implementation of Ford-Fulkerson.
- The description of Ford-Fulkerson Algorithm is incomplete.
- For this reason, it is sometimes called Ford-Fulkerson Method.

Next Lecture…

- Max-Flow-Min-Cut Theorem
  - Correctness of Ford-Fulkerson Method
  - Many theorem applications
- Edmonds-Karp Algorithm
  - An implementation of Ford-Fulkerson Method with complexity $O(|V| \cdot |E|^2)$.

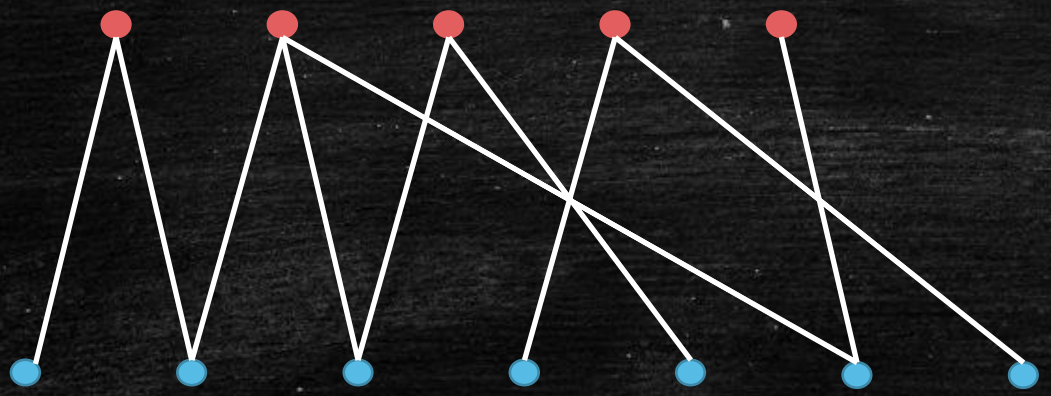
Applications of Integrality Theorem

- **Theorem**. If each $c(e)$ is an integer, then the value of the maximum flow f is an integer.
- Application 1: Tournament example you have seen earlier.
- The max-flow f must satisfy $\forall e$: $f(e) \in \mathbb{Z}$.



Application 2: Maximum Bipartite Matching

- Top vertices are girls, bottom vertices are boys.
- An edge represent a possible match for a boy and a girl.
- Problem: find a maximum matching for boys and girls.

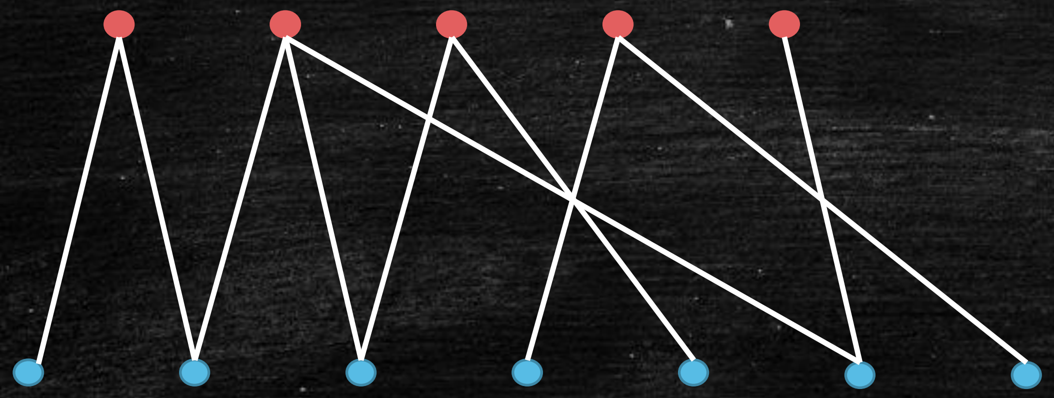


Maximum Bipartite Matching - Formal

- Given a graph $G = (V, E)$, a matching M is a subset of edges that do not share vertices in common.
- The size of a matching is the number of edges in it.
- Problem: Given a bipartite graph $G = (A, B, E)$ find a matching with the maximum size.

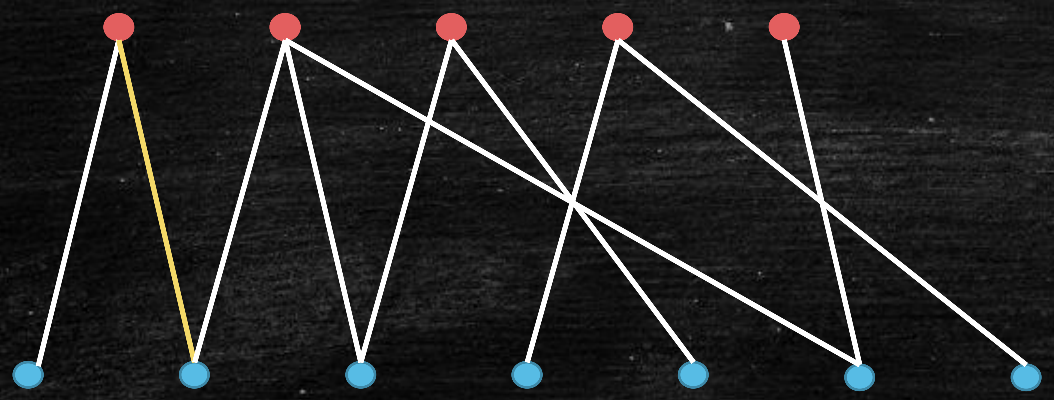
Application 2: Maximum Bipartite Matching

- Greedy doesn't work!



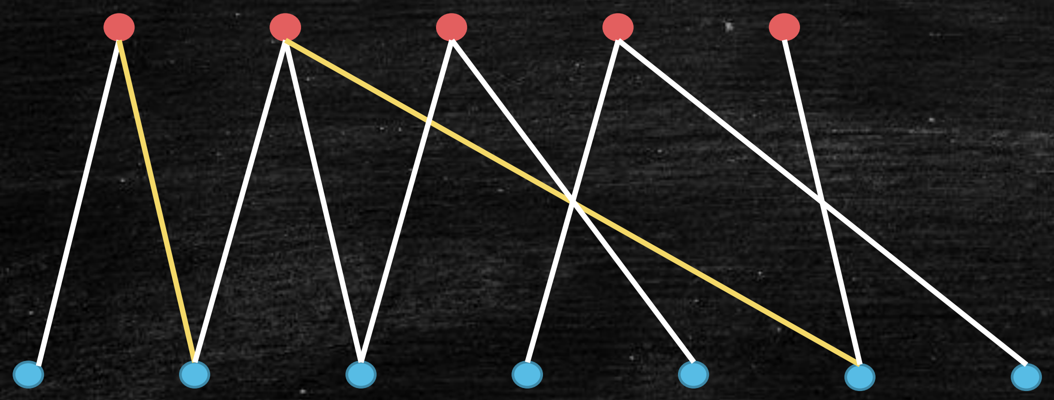
Application 2: Maximum Bipartite Matching

- Greedy doesn't work!



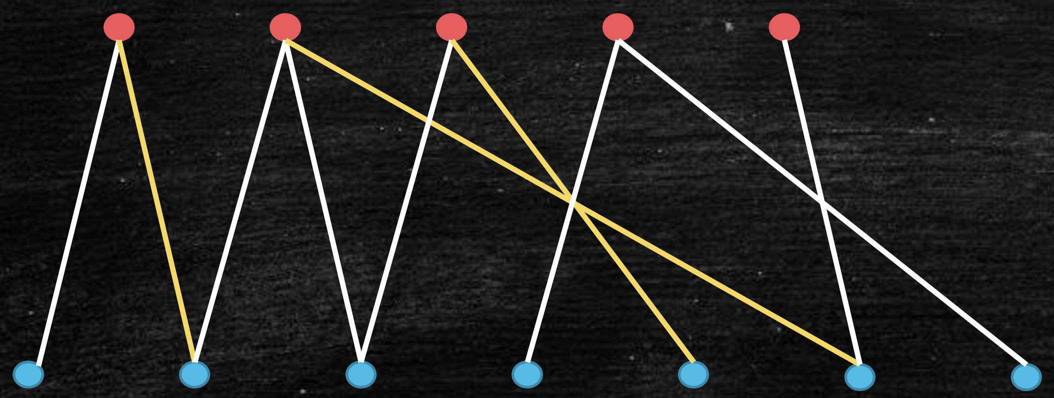
Application 2: Maximum Bipartite Matching

- Naïve greedy doesn't work!



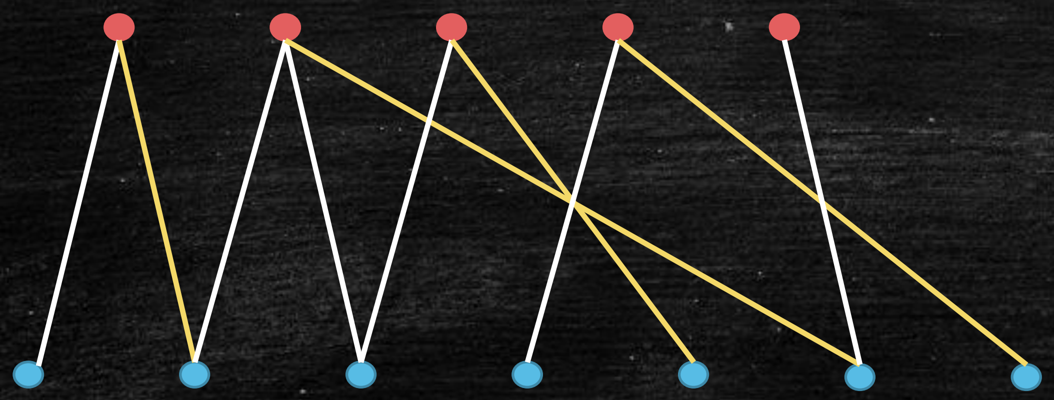
Application 2: Maximum Bipartite Matching

- Naïve greedy doesn’t work!



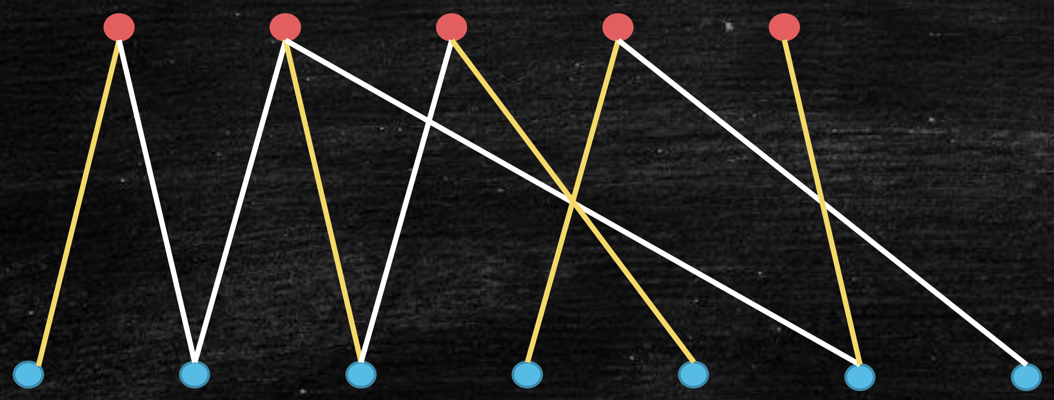
Application 2: Maximum Bipartite Matching

- Naïve greedy doesn't work!
- A total of 4 matches…



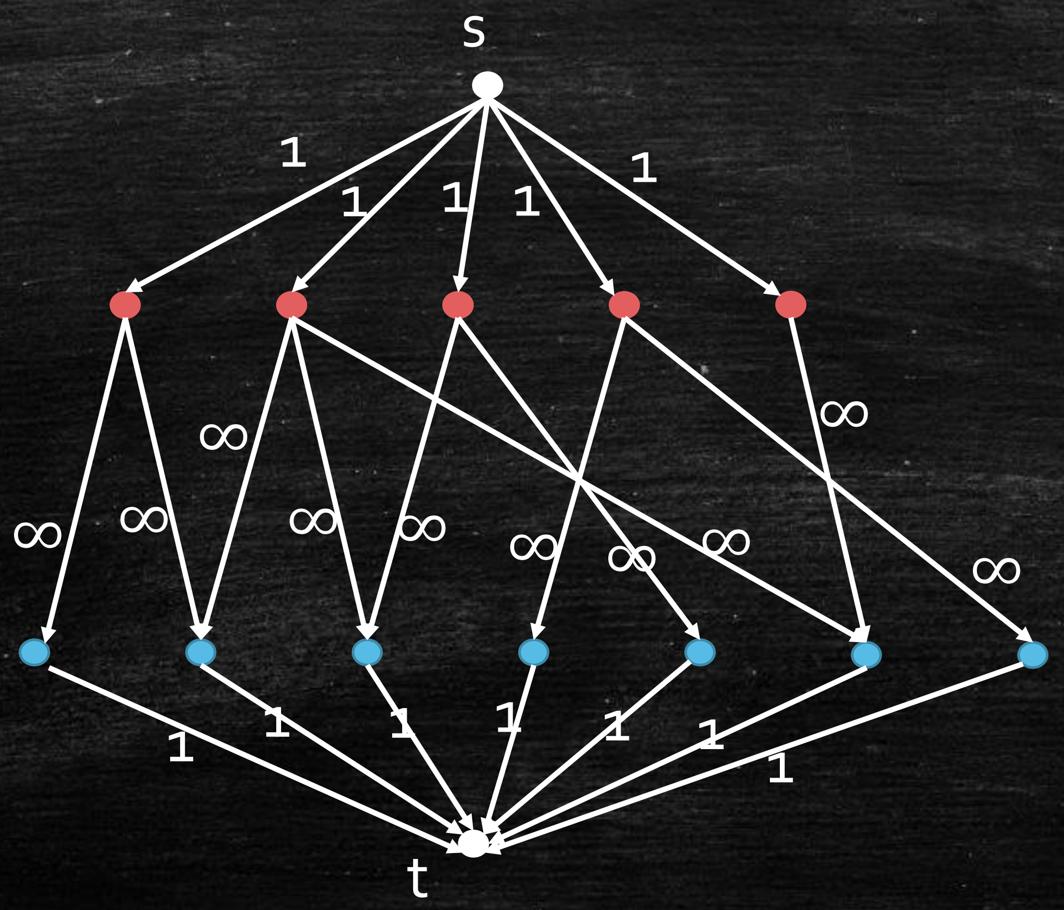
Application 2: Maximum Bipartite Matching

- Greedy doesn't work!
- A better solution…



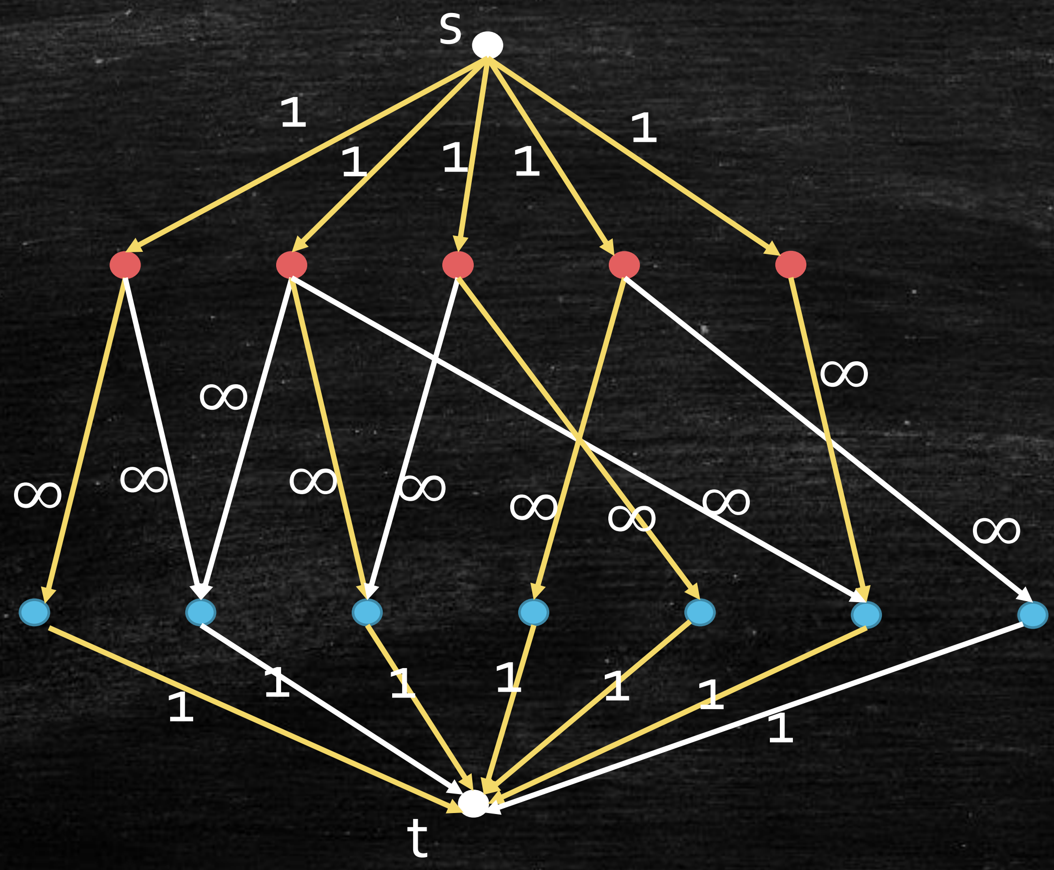
Application 2: Maximum Bipartite Matching

- Applying maximum flow and Ford-Fulkerson Method.



Application 2: Maximum Bipartite Matching

- An integral flow corresponds to a matching.
- Integrality theorem ensures the maximum flow can be integral.



Dessert

- A graph is regular if all the vertices have the same degree.
- A matching is perfect if all the vertices are matched.
- Prove that a regular bipartite graph always has a perfect matching.