Last Lecture

- Max-Flow Problem
- Ford-Fulkerson Algorithm (Method)
- Max-Flow-Min-Cut Theorem
  - Correctness of Ford-Fulkerson Algorithm

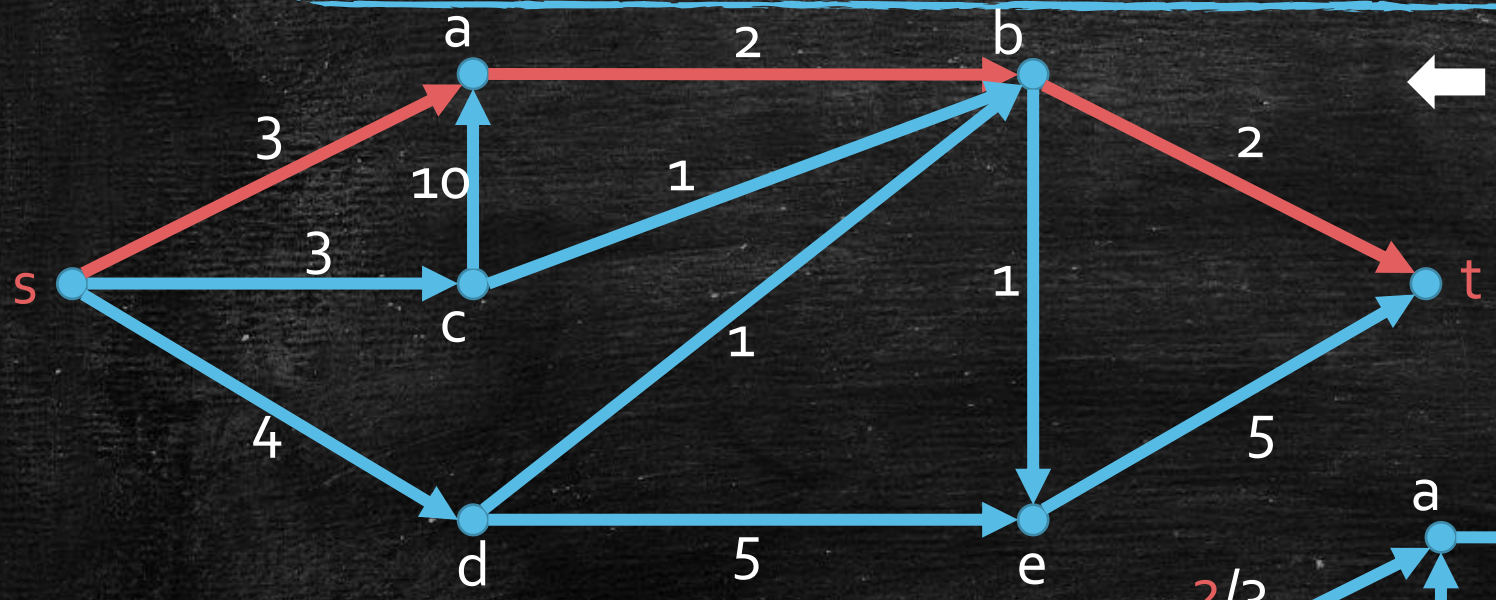
Ford-Fulkerson Algorithm

Ford-Fulkerson Algorithm

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

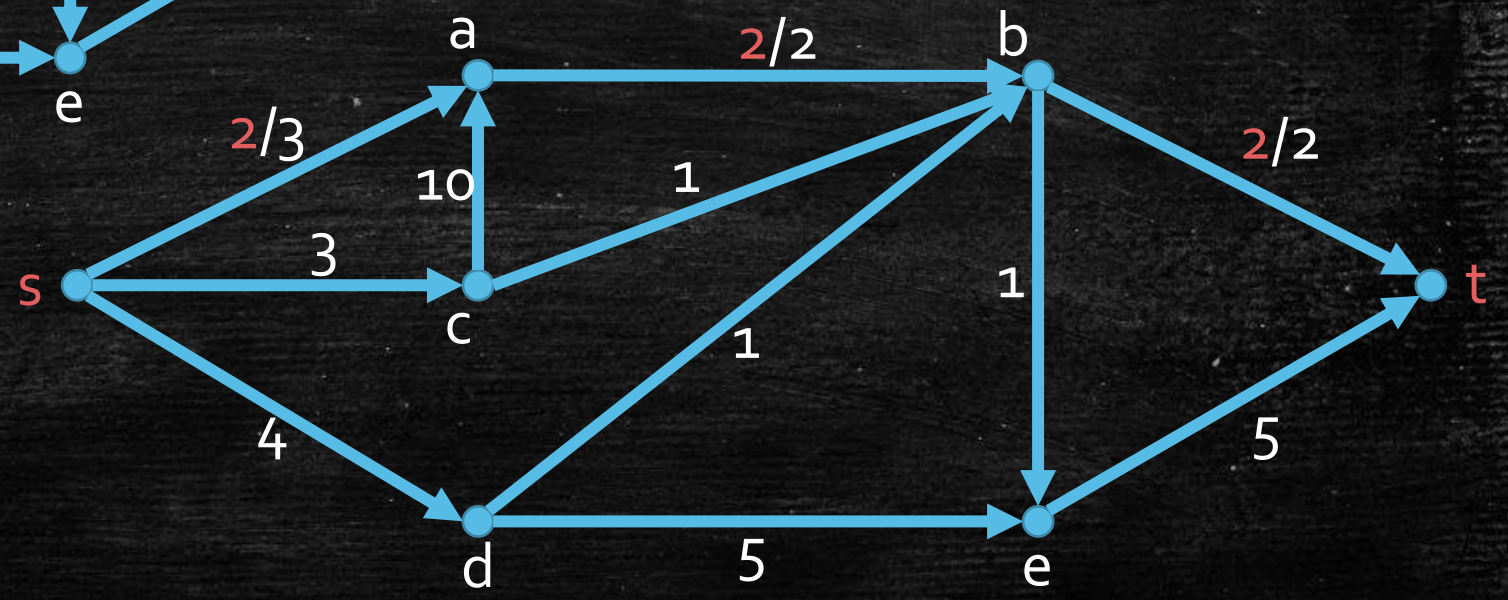
1. initialize f such that $\forall e \in E: f(e) = 0$; initialize $G^f \leftarrow G$;
2. while there is an s-t path on G^f :
3. find an edge $e \in p$ with minimum capacity b;
4. update f that pushes b units of flow along p;
5. update G^f;
6. endwhile
7. return f

Ford-Fulkerson Algorithm

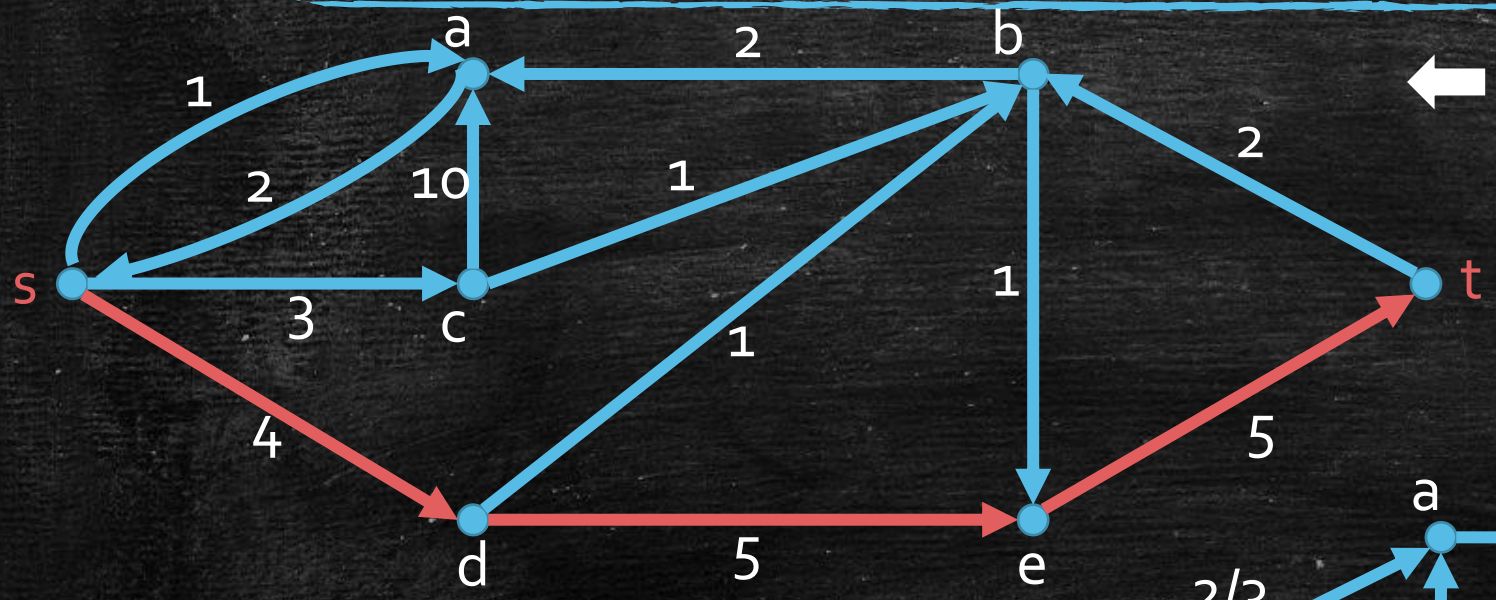


Residual Network

Current Flow

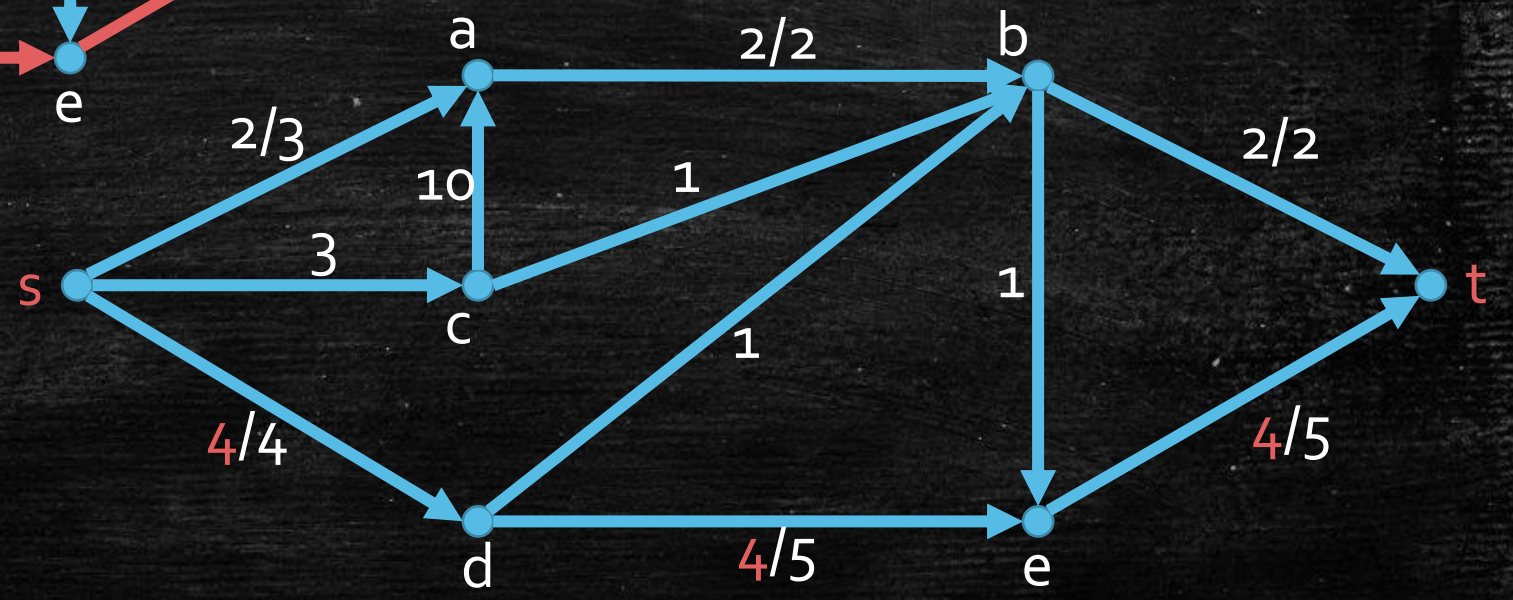


Ford-Fulkerson Algorithm

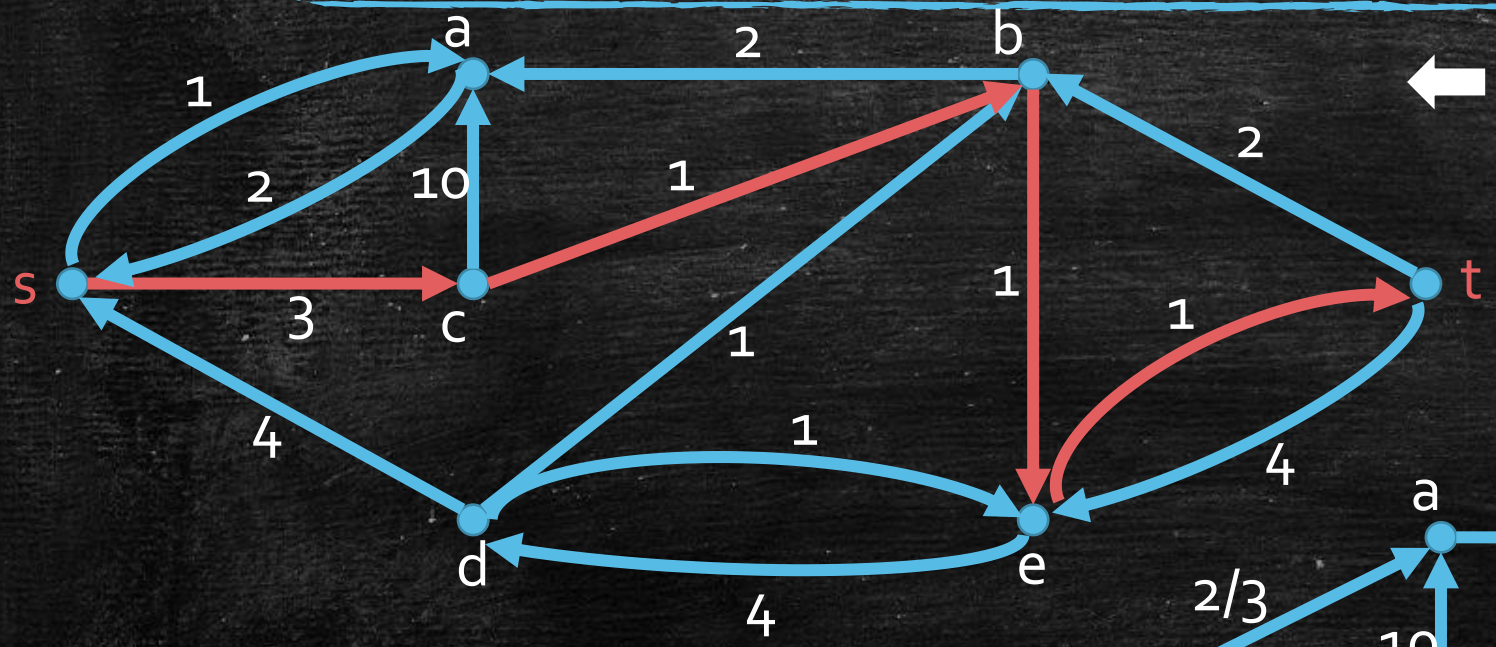


Residual Network

Current Flow

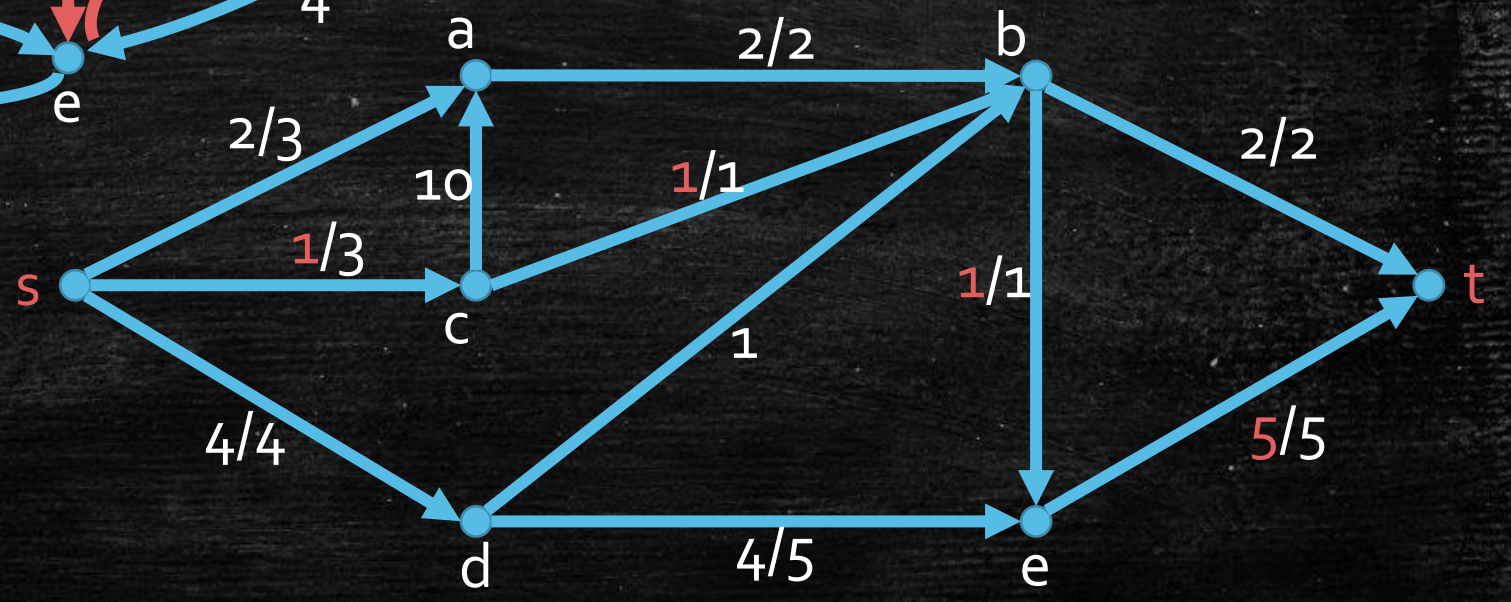


Ford-Fulkerson Algorithm

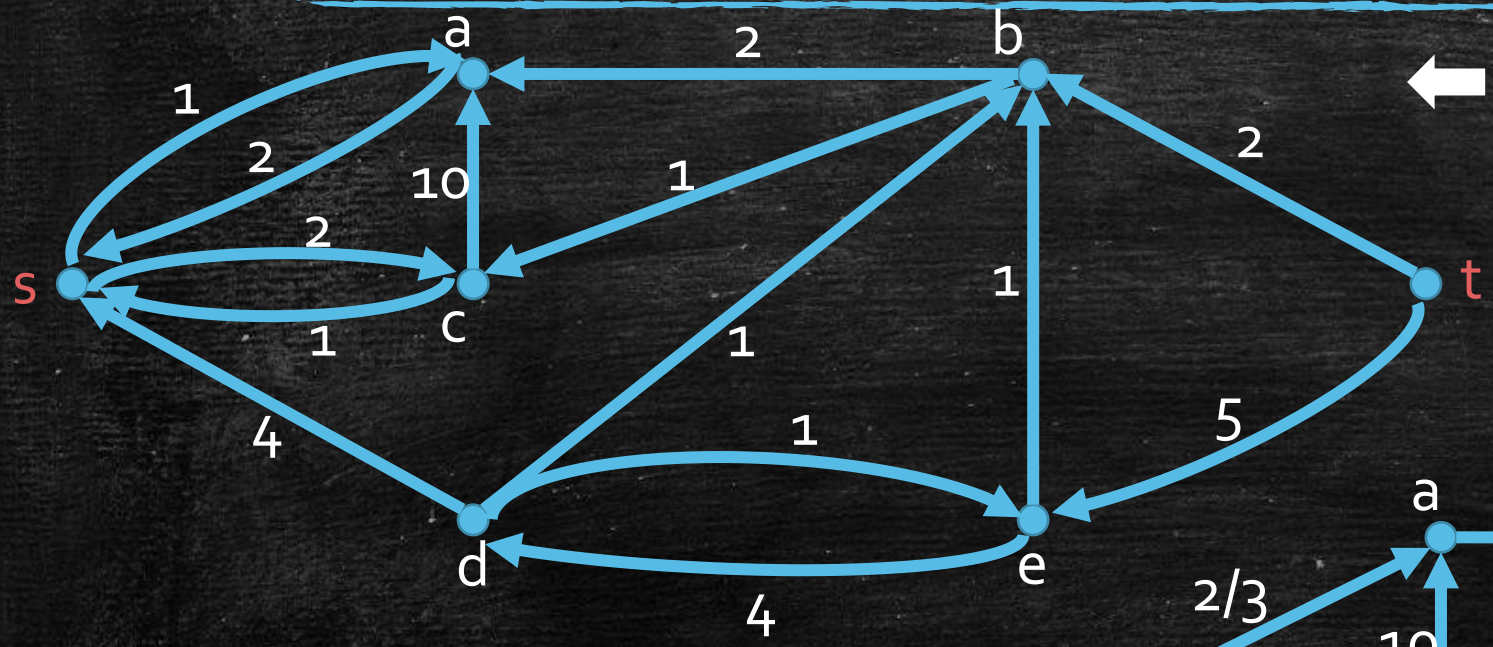


Residual Network

Current Flow

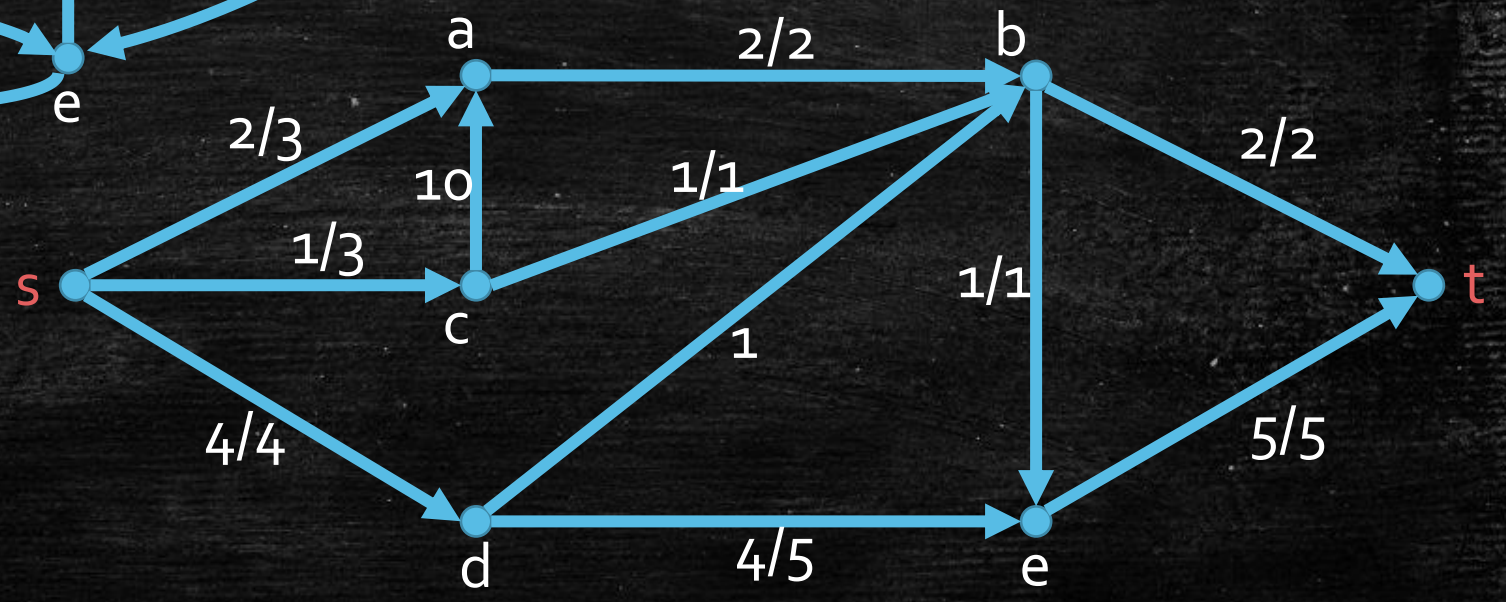


No more s-t path… Algorithm Terminate…



Residual Network

Current Flow



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!

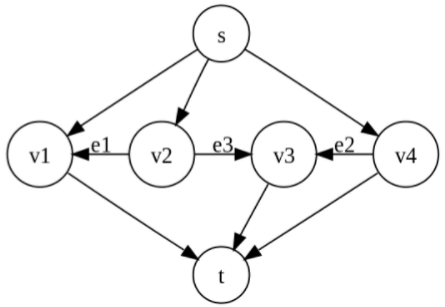
Does the algorithm always halt?

- 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?

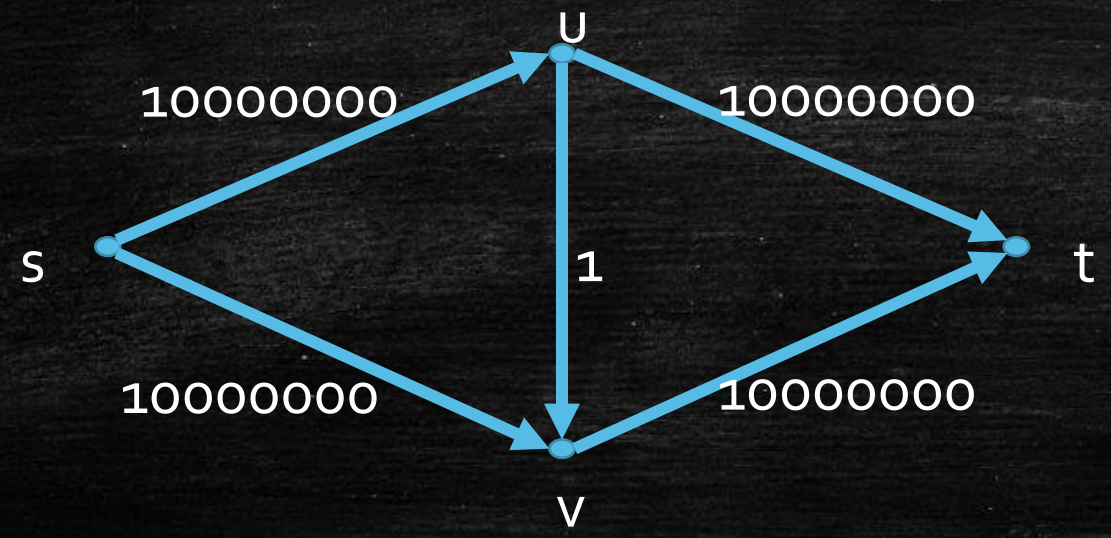
- Assume all capacities are integers, what is the time complexity?
- Each iteration requires $O(|E|)$ time:
  - $O(|E|)$ is sufficient for finding p, updating f and G^f
- There are at most f_{max} iterations.
- Overall: $O(|E| \cdot f_{max})$

Class Activity 1

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

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: Edmonds-Karp Algorithm
  - Careful choices of augmenting paths
  - Polynomial time

Edmonds-Karp Algorithm

A polynomial time implementation of Ford-Fulkerson method

Edmonds-Karp Algorithm

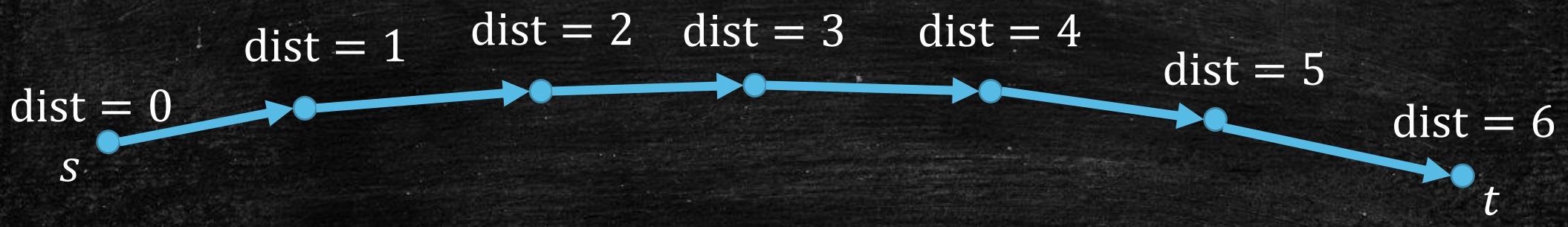
Edmonds-Karp Algorithm

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

1. initialize f such that $\forall e \in E: f(e) = 0$; initialize $G^f \leftarrow G$;
2. while there is an s-t path on G^f:
3. **find such a path p by BFS;**
4. find an edge $e \in p$ with minimum capacity b;
5. update f that pushes b units of flow along p;
6. update G^f;
7. endwhile
8. return f

Why BFS?

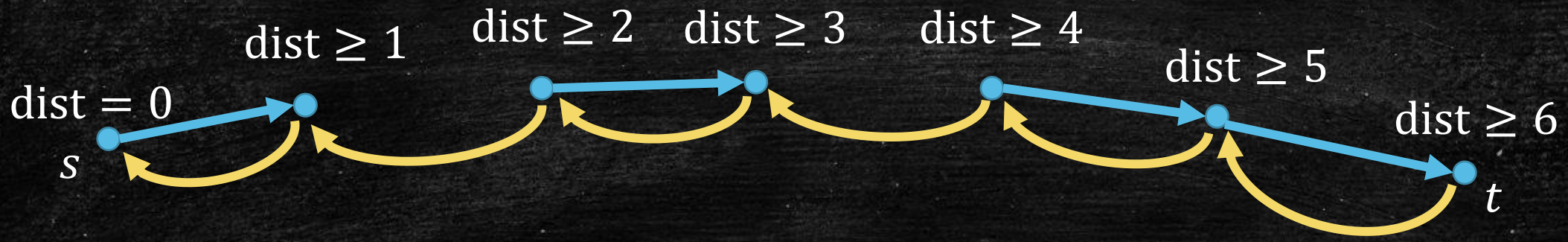
- BFS maintains the distances
  - distance: num of edges, not weighted distance



A path found by an iteration of Edmonds-Karp Algorithm

Why BFS?

- In the residual network G^f, a new appeared edge can only goes from a vertex at distance $t+1$ to a vertex at distance t.
- Addition of such edges does not decrease the distance between s and u for every $u \in V$.
- **[Key Observation]** Thus, $\mathrm{dist}(u)$ is non-decreasing throughout the algorithm for every $u \in V$.

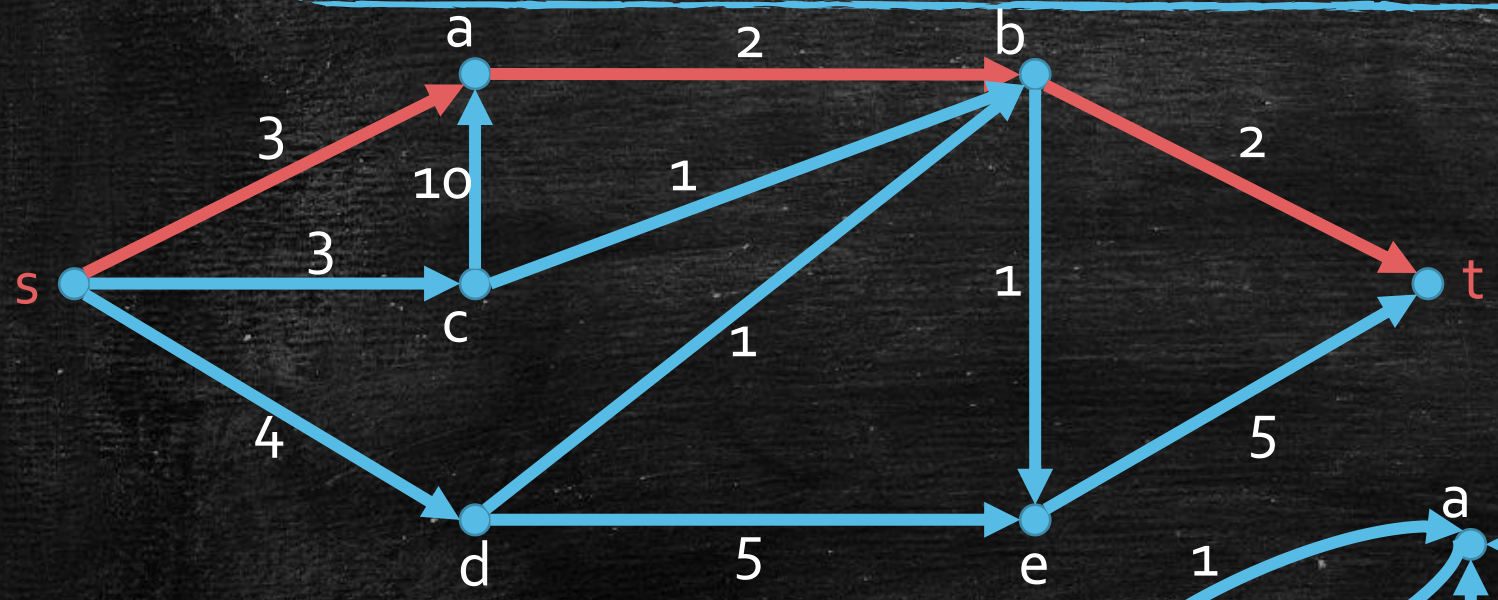


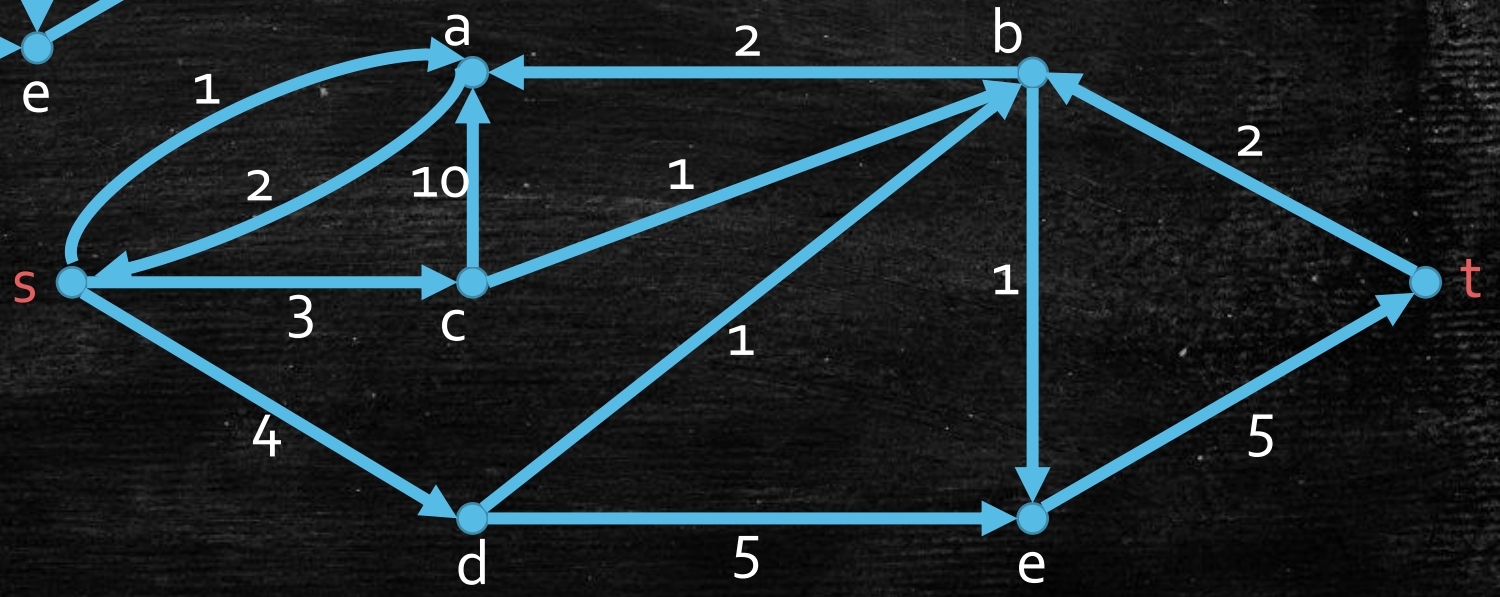
The updates to the edges in G^f

Weak Monotonicity to Strong Monotonicity

- $\mathrm{dist}(u)$ can only be one of $0, 1, 2, \ldots, |V|, \infty$
  - It can only be increased for $|V| + 1$ times!
- It's great that BFS buys us distance monotonicity!
- However, weak monotonicity is not enough.
- To make a progress, we need $\mathrm{dist}(u)$ strictly increases for some $u \in V$, so that we can upper bound the number of iterations.

Counterexample: dist for all vertices remain unchanged after an iteration.





Towards Strong Monotonicity…

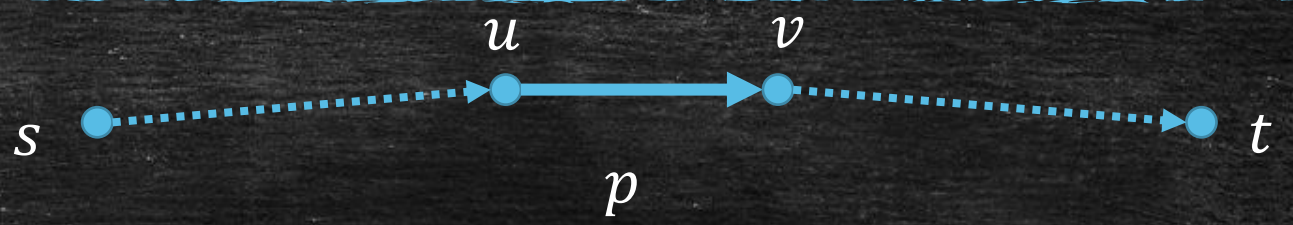
- Observation: At least one edge (u, v) on p is "saturated", and this edge will be deleted in the next iteration.
- Each iteration removes an edge from a vertex at distance i to a vertex at distance $i + 1$.
- Intuitively, we cannot keep removing such edges while keeping the distances of all vertices unchanged.

Towards Strong Monotonicity

- Suppose we are at the $(i+1)$-th iteration. f_i is the current flow, and p is the path found in G^{f_i} at the $(i+1)$-th iteration.
- We say that an edge (u, v) is critical if the amount of flow pushed along p is $c^{f_i}(u, v)$.
- A critical edge disappears in $G^{f_{i+1}}$, but it may reappear in the future…
- We will try to bound the number of times (u, v) becomes critical.

Between two "critical"

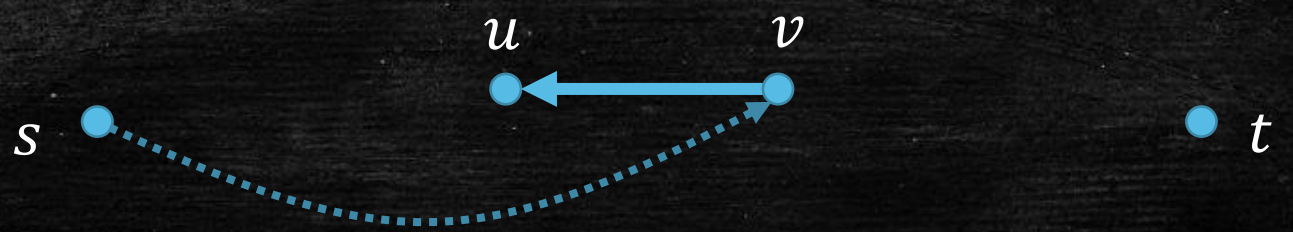
A flow along p in G^{f_i} where (u, v) becomes critical



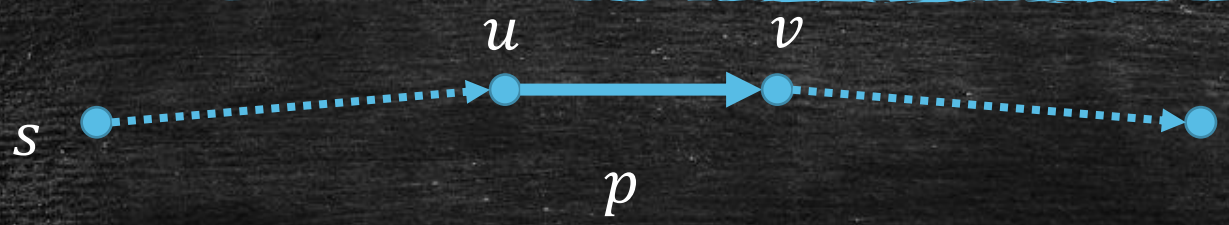
In $G^{f_{i+1}}$, (u, v) disappears, and (v, u) appears.

Before the next time (u, v) becomes critical again, (u, v) must first reappear!

Before (u, v) reappears, the algorithm must have found p going through (v, u).



Between two “critical”



$\mathrm{dist}^i(v) = \mathrm{dist}^i(u) + 1$

$\mathrm{dist}^{i+j}(u) = \mathrm{dist}^{i+j}(v) + 1$

- Distance monotonicity: $\mathrm{dist}^{i+j}(v) \geq \mathrm{dist}^i(v)$.
- Thus, $\mathrm{dist}^{i+j}(u) = \mathrm{dist}^{i+j}(v) + 1 \geq \mathrm{dist}^i(v) + 1 \geq \mathrm{dist}^i(u) + 2$.
- The distance of u from s increases by 2 between two “critical”.

Class Activity 2

- The distance of u from s increases by 2 between two "critical".
- At least 1 edge become critical in one iteration.

Can you figure out the time complexity for Edmonds-Karp?

A. $O(|V| \cdot |E|)$

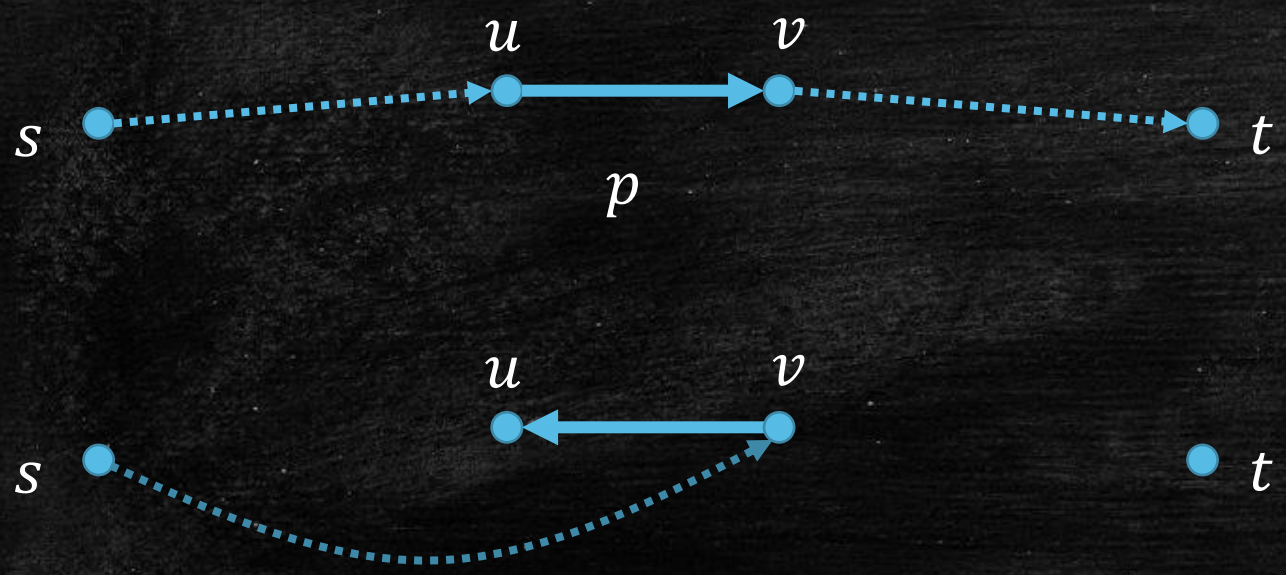
B. $O(|V|^2 \cdot |E|)$

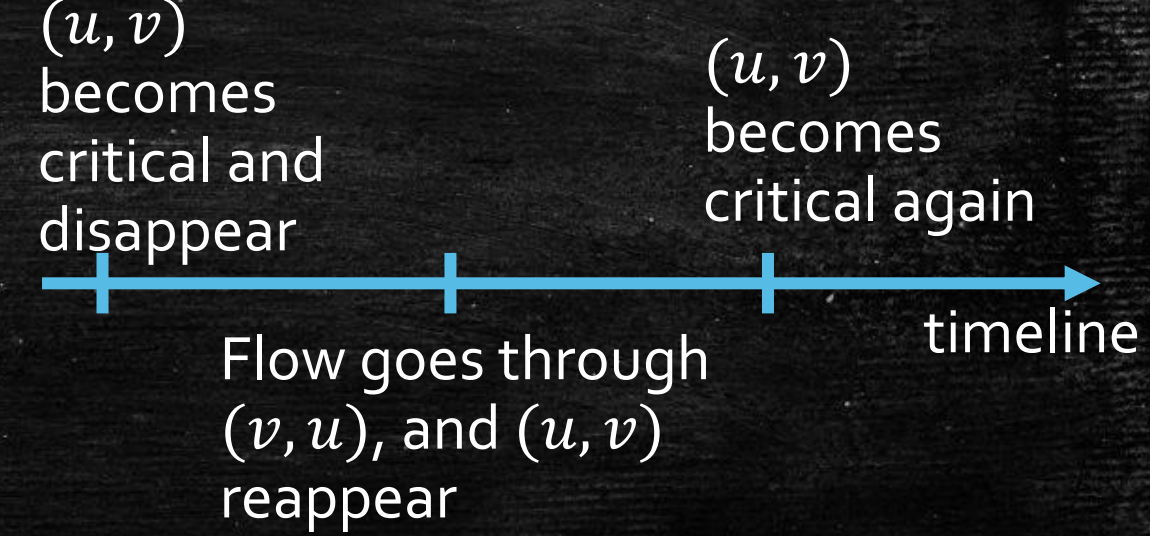
C. $O(|V| \cdot |E|^2)$

D. $O(|V|^2 \cdot |E|^2)$

Recap of Key Points

- BFS ensures (weak) distance monotonicity
- Bound the number of "critical" for each edge
- Distance monotonicity ensures that $\mathrm{dist}(u)$ is increased by 2 between two "critical" of (u, v)





Other Algorithms for Max-Flow

- Dinic's algorithm: $O(|V|^2|E|)$
- Improvements to Dinic's algorithm:
  - [Malhotra, Kumar & Maheshwari, 1978]: $O(|V|^3)$
  - Dynamic tree: $O(|V| \cdot |E| \cdot \log|V|)$
- Push-relabel algorithm [Goldberg & Tarjan, 1988]
  - $O(|V|^2|E|)$, later improved to $O(|V|^3)$, $O\left(|V|^2\sqrt{|E|}\right)$, $O\left(|V||E|\log\frac{|V|^2}{|E|}\right)$
- [King, Rao & Tarjan, 1994] and [Orlin, 2013]: $O(|V| \cdot |E|)$
- Interior-point-method-based algorithms:
  - [Kathuria, Liu & Sidford, 2020] $|E|^{\frac{4}{3}+o(1)}U^{\frac{1}{3}}$
  - [BLNPSSSW, 2020] [BLLSSSW, 2021] $\tilde{O}\left(\left(|E|+|V|^{\frac{3}{2}}\right)\log U\right)$
  - [Gao, Liu & Peng, 2021] $\tilde{O}\left(|E|^{\frac{3}{2}-\frac{1}{328}}\log U\right)$

Applications of Max-Flow

Problems Related to Assignments

Applications of Max-Flow

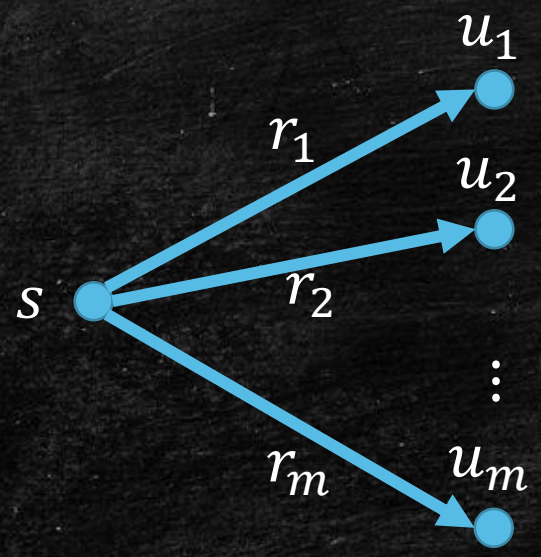
- Many problems related to assignments can be formulated as the max-flow problem.
- Integrality theorem usually plays an important role
  - [Integrality Theorem] Integer capacities imply integral flow.

Dinner Table Assignment

- Students from m different universities participate in a conference.
- Each university i has r_i students.
- n tables, each table i can be shared by at most c_i students.
- Decide if it is possible to make an arrangement such that each table is shared by students from different universities.

Dinner Table Assignment

- Build vertices for universities
- Capacities encode $\{r_i\}$



University

Dinner Table Assignment

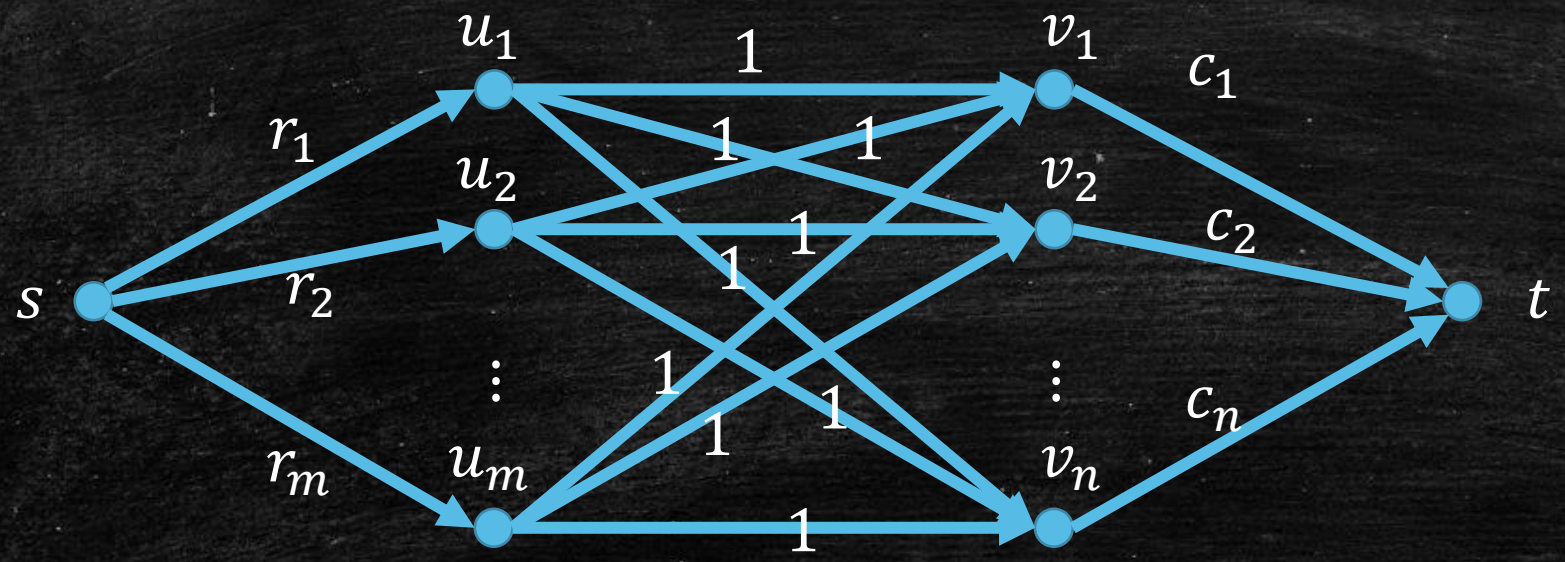
- Build vertices for tables
- Capacities encode $\{c_i\}$

University

Table

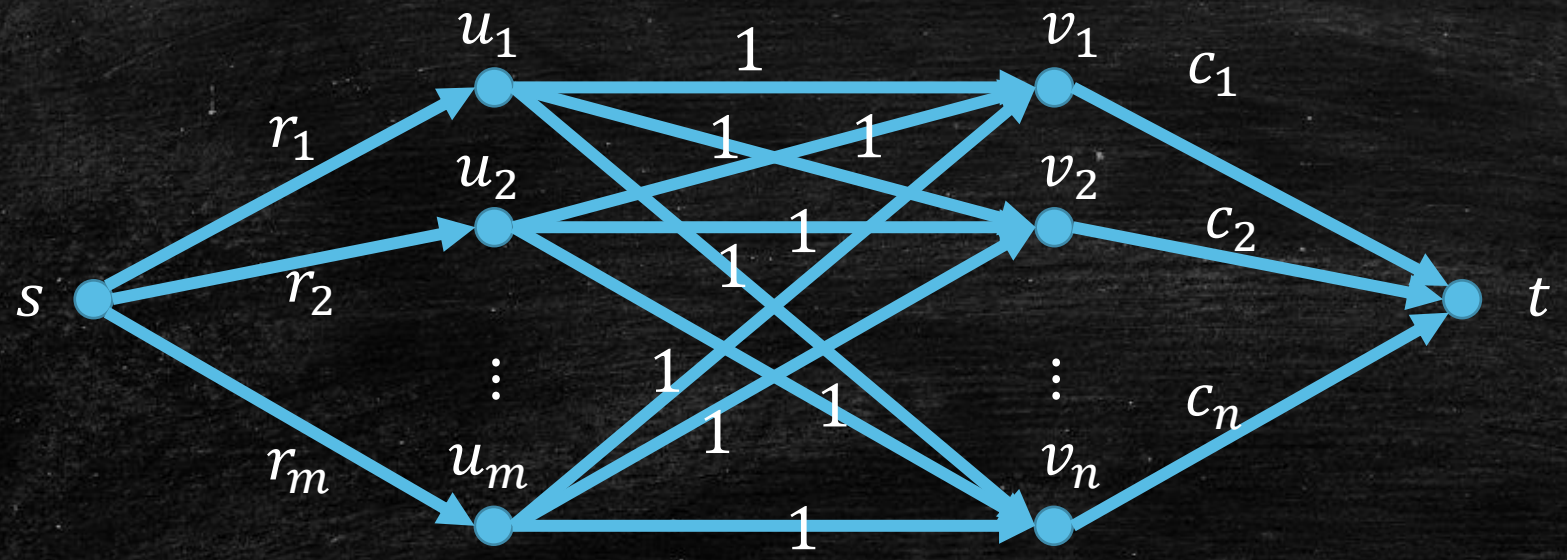
Dinner Table Assignment

- A complete bipartite graph in between
- Each edge has capacity 1: each table can only hold one student from each university.



Dinner Table Assignment

- An integral flow represents an assignment
- Integrality Theorem: integer capacities $\Longrightarrow$ integral flow
- Only need to decide if the max-flow has value $\sum_{i=1}^{m} r_i$

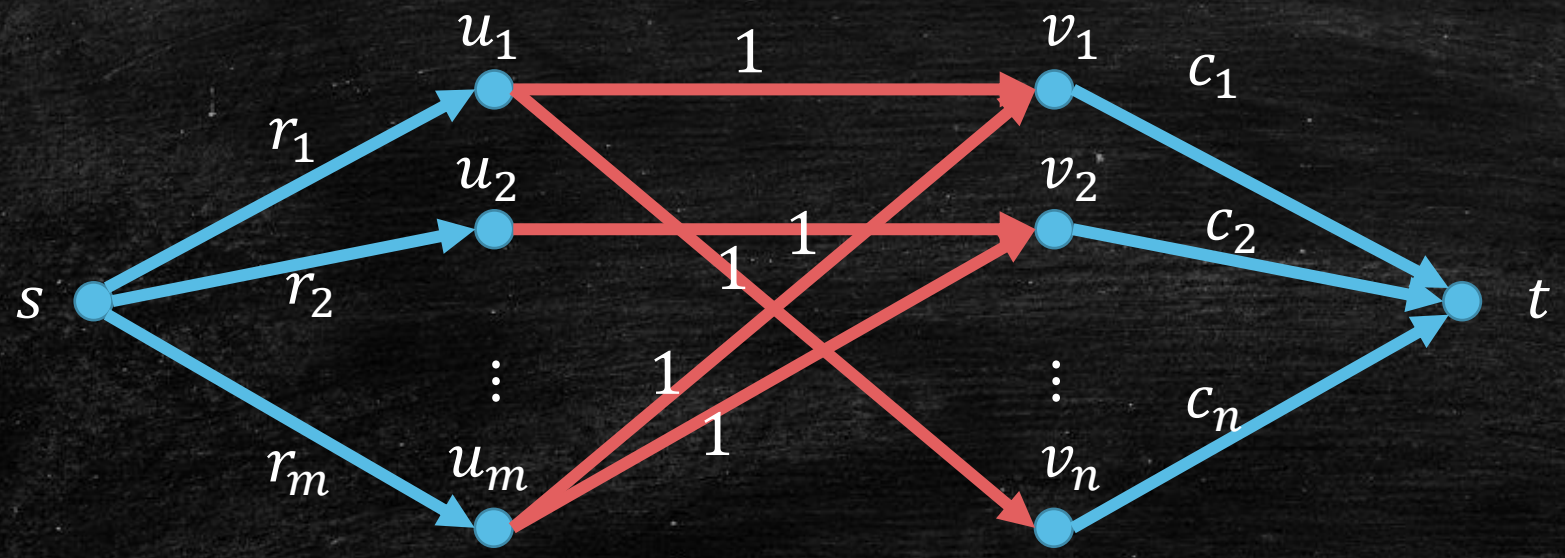


Dinner Table Assignment

- Students from m different universities participate in a conference.
- Each university i has r_i students.
- n tables, each table i can be shared by at most c_i students.
- Decide if it is possible to make an arrangement such that each table is shared by students from different universities.
- What if we add the following extra requirement?
- Each table i is associated with a subset $T_i \in \{1, \dots, m\}$ such that table i can only accept students from universities in T_i

Dinner Table Assignment

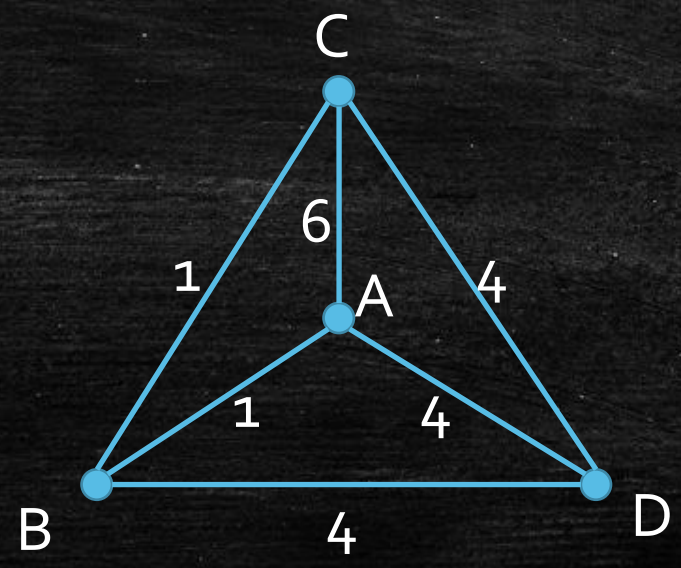
- Now the bipartite graph in the middle is no longer complete
- In-neighbors of v_i correspond to T_i
- Still Max-flow problem



Tournament

- 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?

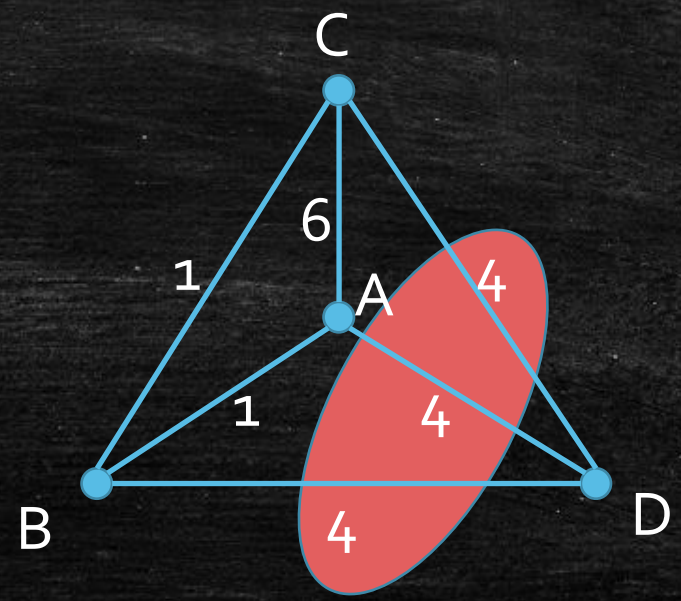
	Wins
A	40
B	38
C	37
D	29



Tournament

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

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



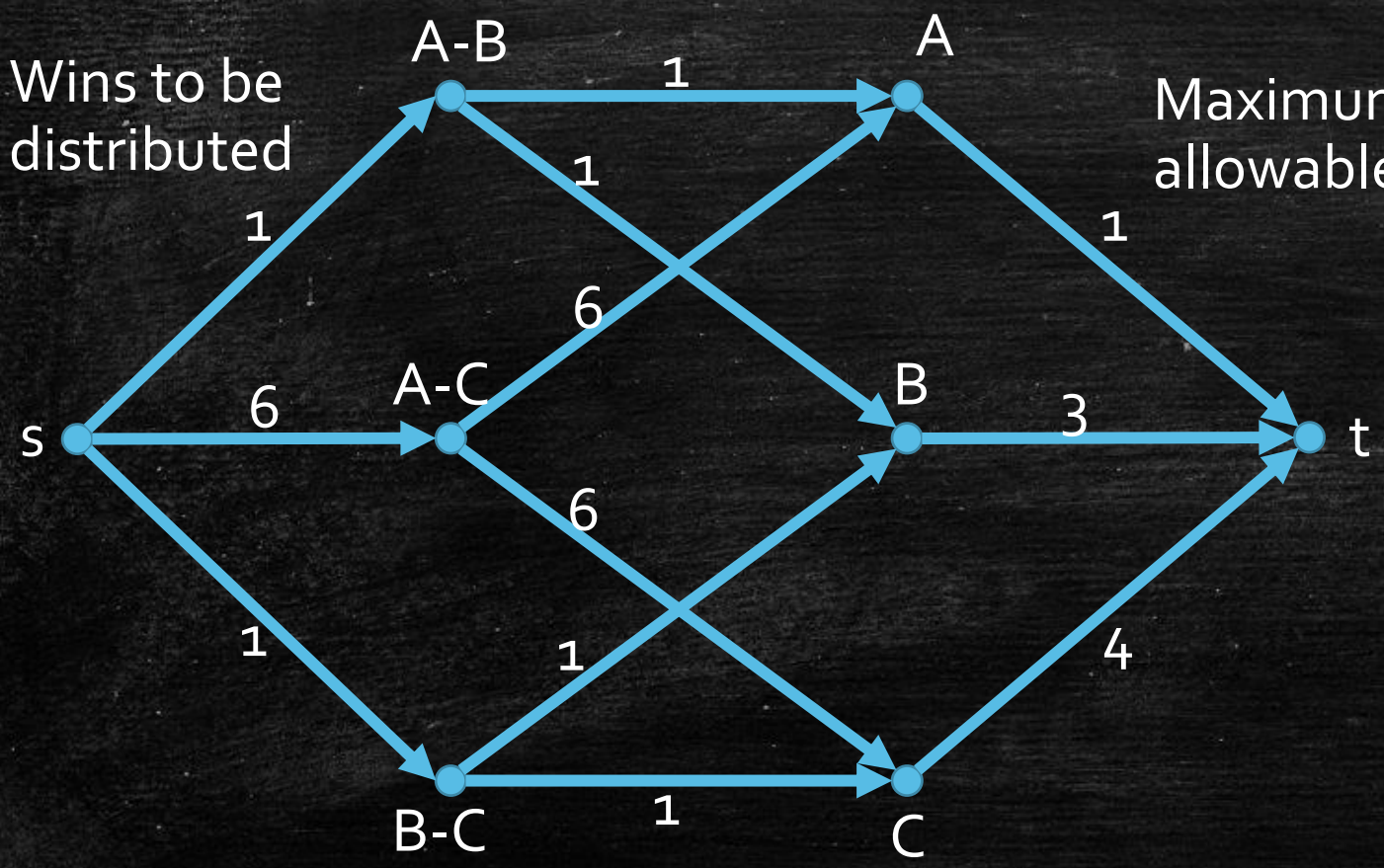
Tournament

- 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

Tournament

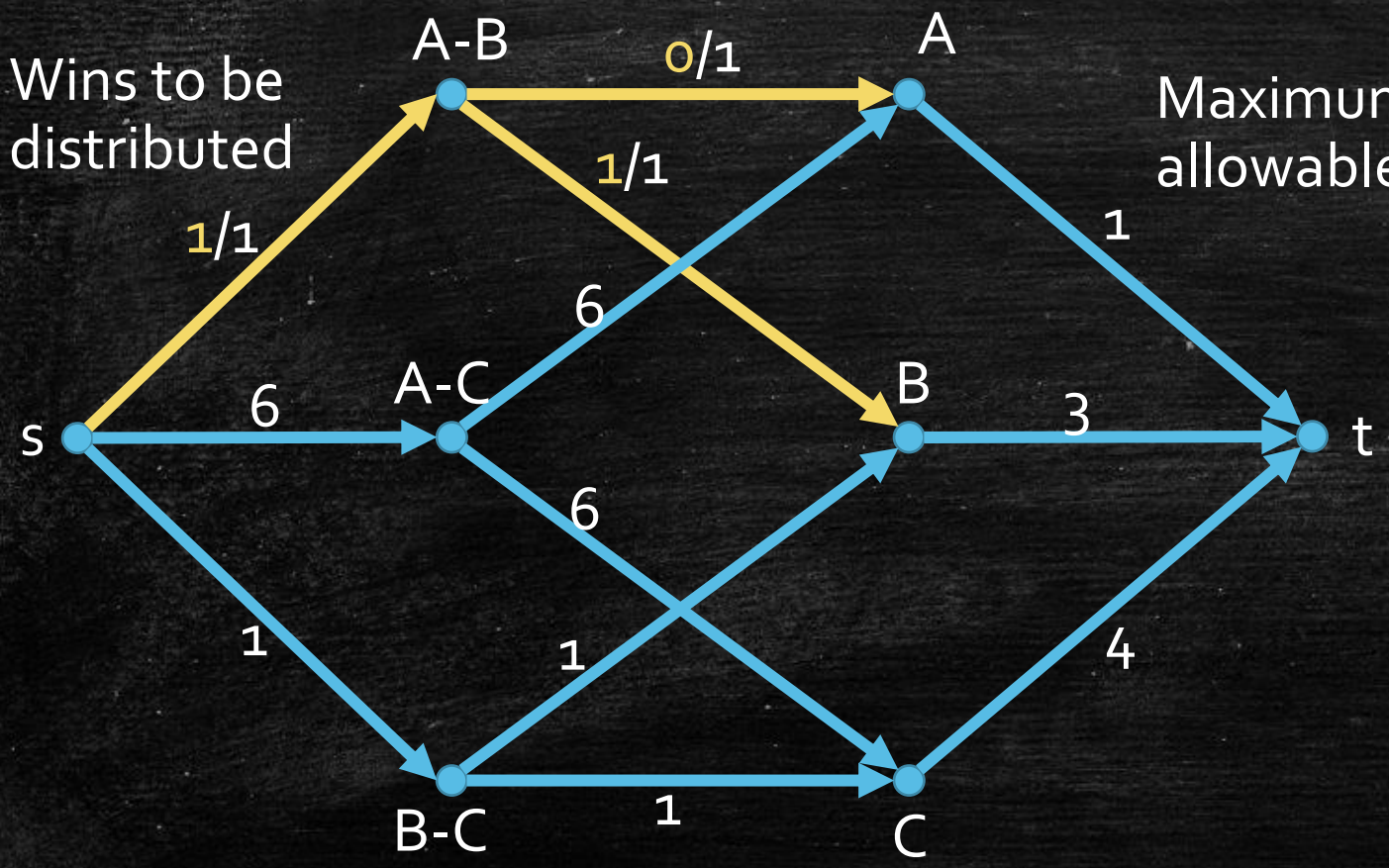
- Model the problem as Max-Flow.



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

Tournament

- Model the problem as Max-Flow.

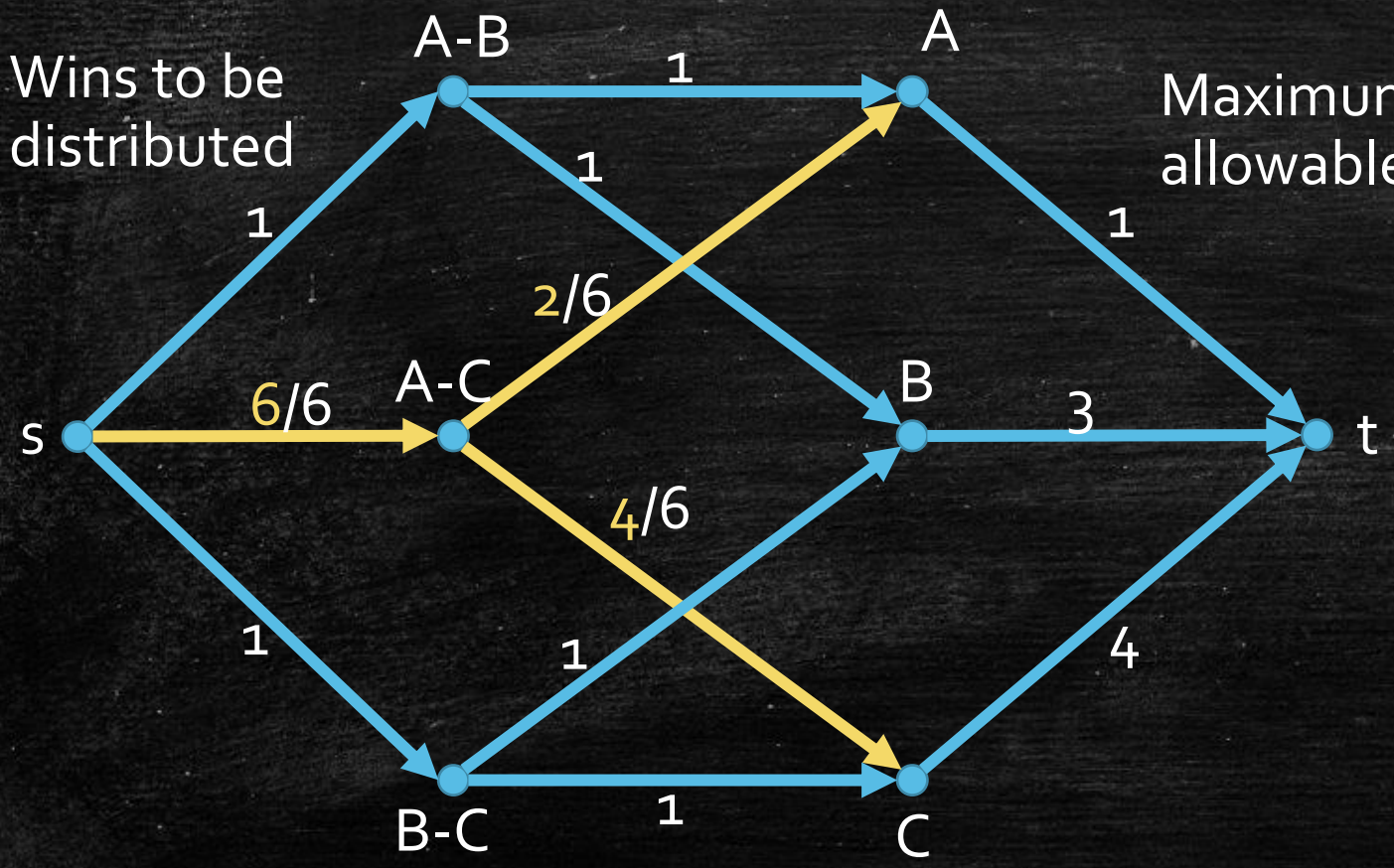


Maximum allowable wins

Example:
In the one match between A and B, B wins.

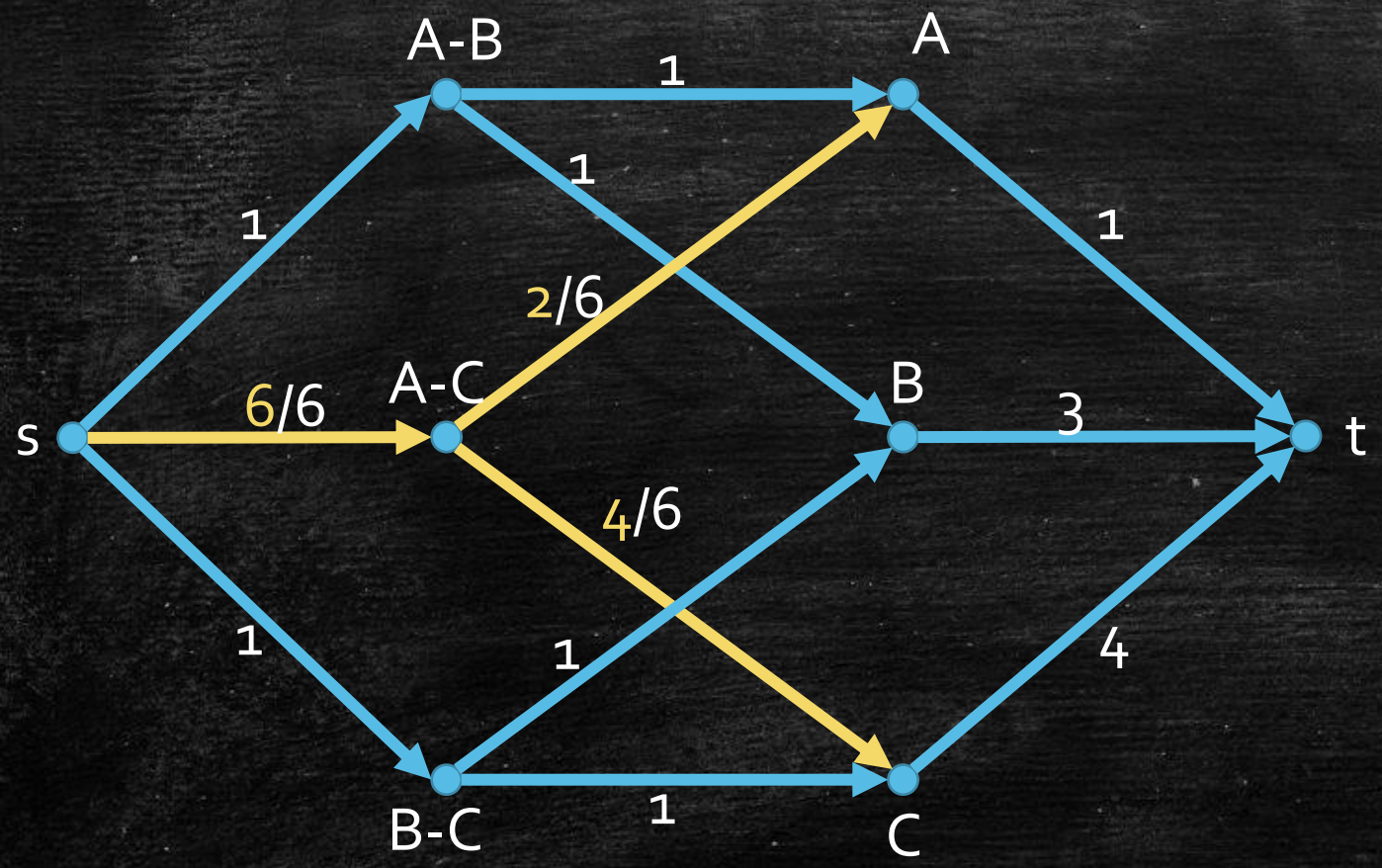
Tournament

- Model the problem as Max-Flow.



Example:
In the 6 matches between A and C, A wins 2 matches and C wins 4 matches.

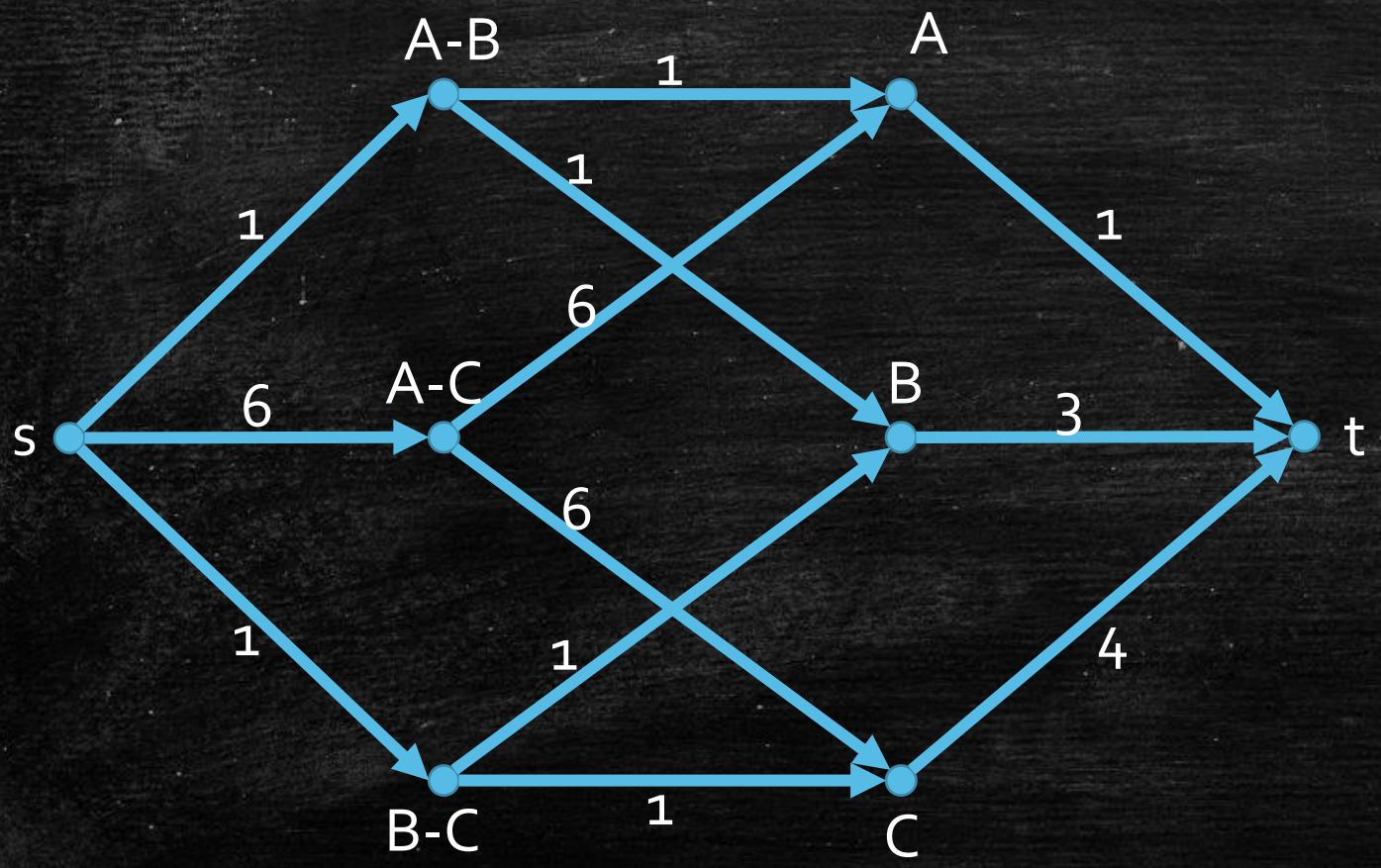
Tournament



- Only integral flow makes sense.
- Integrality Theorem ensures this!

Tournament

- 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	

Today's Lecture

- Ford-Fulkerson Method Time Complexity
  - May not terminate for irrational capacities
  - Not in polynomial time for integer/rational capacities
- Integrality Theorem
  - integer capacity $\Longrightarrow$ integral flow
- Edmonds-Karp Algorithm
  - An implementation of Ford-Fulkerson Method
  - Use BFS to search s-t paths in residual network
- Application of Max-Flow
  - Assignment problems