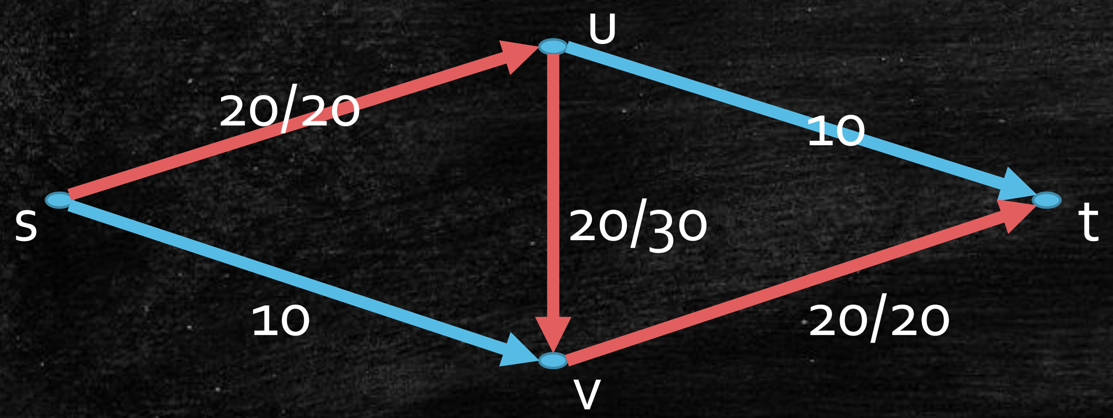
Network Flow: Running Time

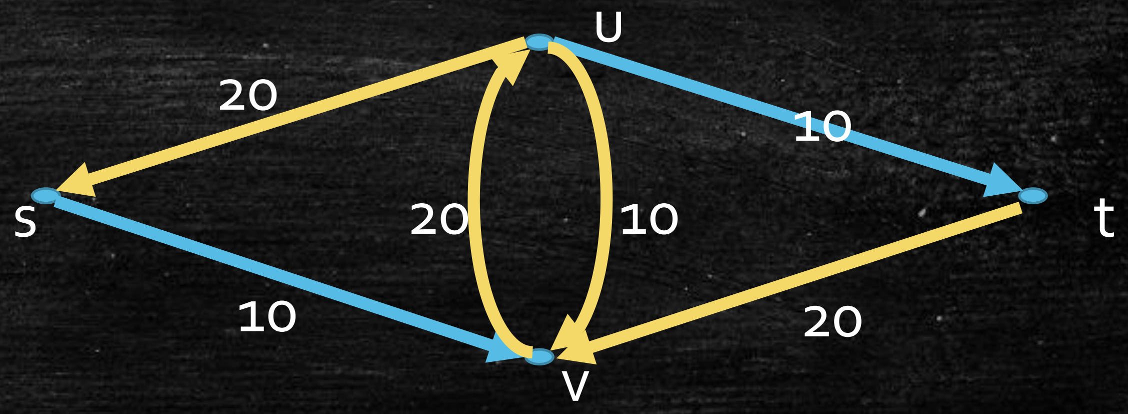
Max-Flow: Edmonds-Karp Algorithm, Dinitz’s Algorithm
Max Bipartite Matching: Hopcroft–Karp–Karzanov algorithm

Residual Network G^f

- Given $G = (V, E)$, c, and a flow f
- $G^f = (V, E^f)$ with capacity c^f
- $(u, v) \in E^f$ if one of the followings holds
  - $(u, v) \in E$ and $\boldsymbol{f(u, v) < c(u, v)}$: $c^f(u, v) = c(u, v) - f(u, v)$
  - $(v, u) \in E$ and $\boldsymbol{f(v, u) > 0}$: $c^f(u, v) = f(v, u)$



G



G^f

Last Lecture – Ford-Fulkerson Method

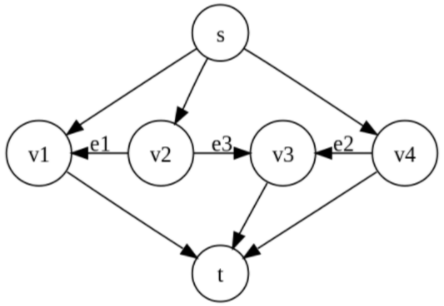
- Always terminates for integer/rational capacities
- Not guaranteed to terminate for irrational capacities
- Time complexity for integer capacities: $O(|E| \cdot v(f_{\max}))$
  - not a polynomial time

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]

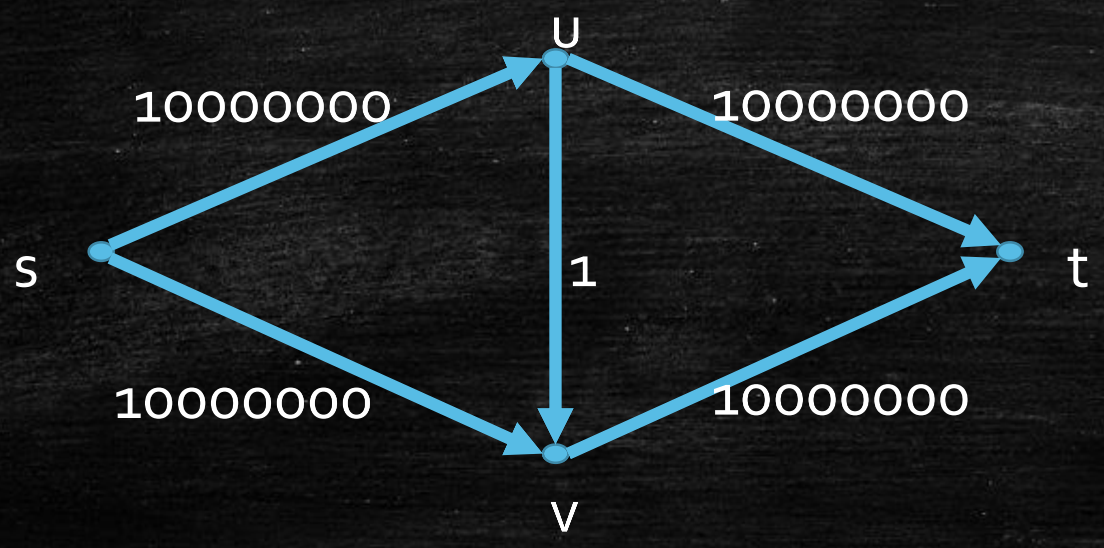
Another non-terminating example based on the Euclidean algorithm is given by Backman & Huynh (2018), where they also show that the worst case running-time of the Ford-Fulkerson algorithm on a network $G(V, E)$ in ordinal numbers is $\omega^{\Theta(|E|)}$.

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 if choices of p are not carefully specified!



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 Preview: Edmonds-Karp Algorithm, which implement Ford-Fulkerson Method with time complexity $O(|V| \cdot |E|^2)$.

Edmonds-Karp Algorithm

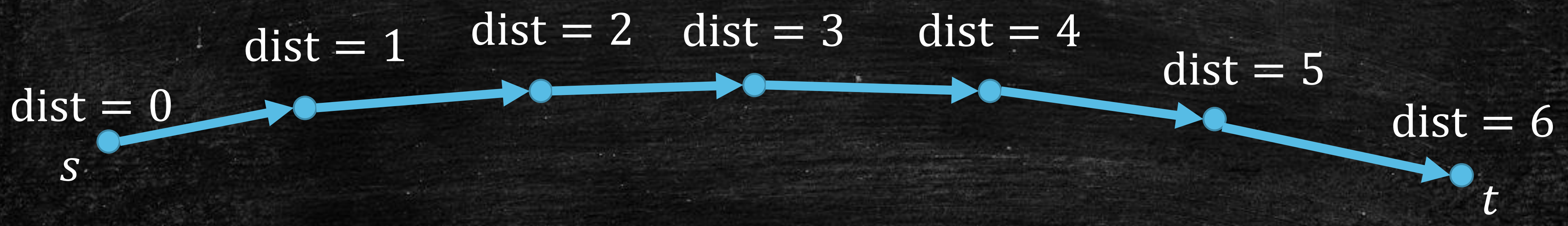
Edmonds-Karp Algorithm

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?

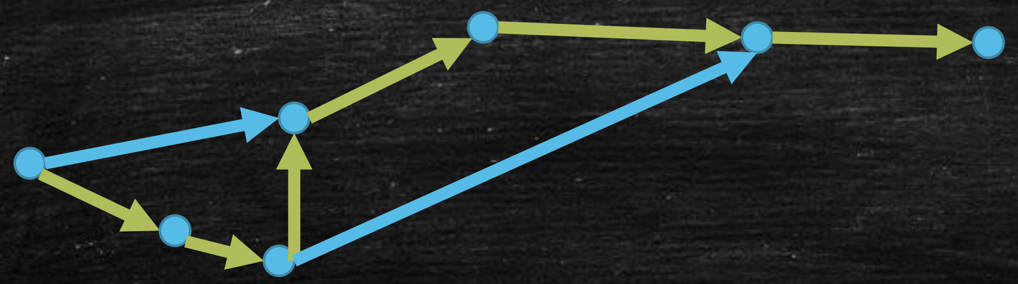
- BFS maintains the distances
  - distance: num of edges, not weighted distance



A path found by an iteration of Edmonds-Karp Algorithm

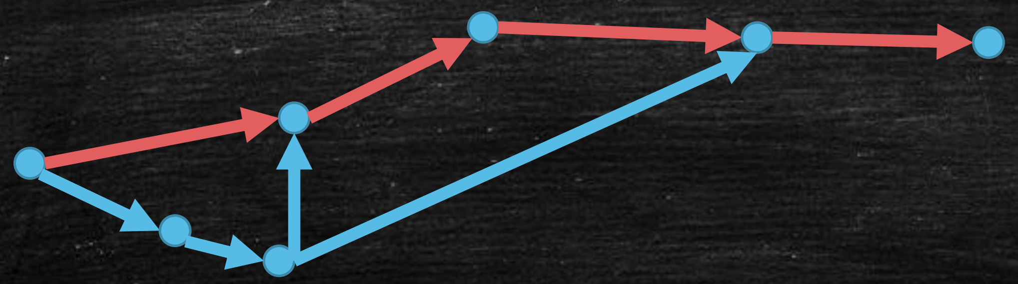
Examples

- Can we choose the green path?



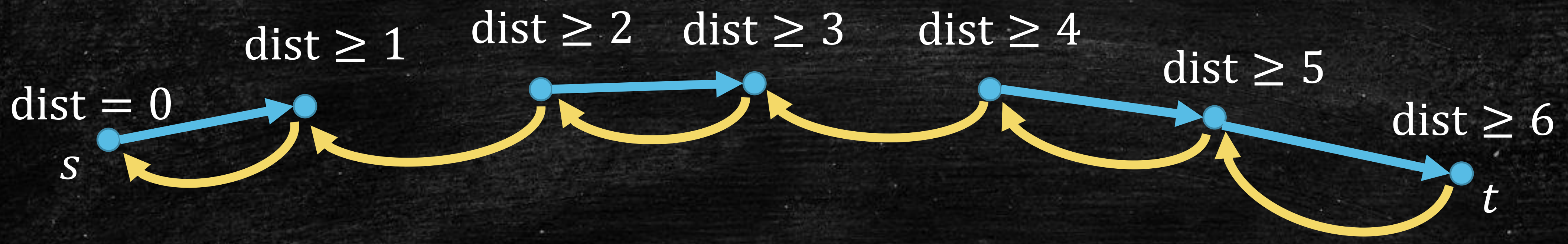
Examples

- We choose the red path!



Why BFS?

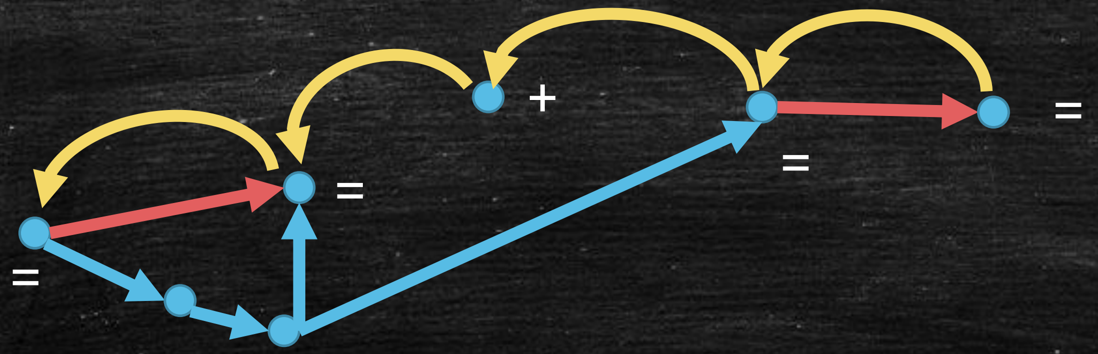
- In the residual network G^f, a **new appeared edge** can only go from a vertex at distance $t+1$ to a vertex at distance t.
- Addition of such edges does not **decrease** the distance between s and u for every $u \in V$.
- **[Key Observation]** $\mathrm{dist}(u)$ in G^f is non-decreasing.



The updates to the edges in G^f

Examples

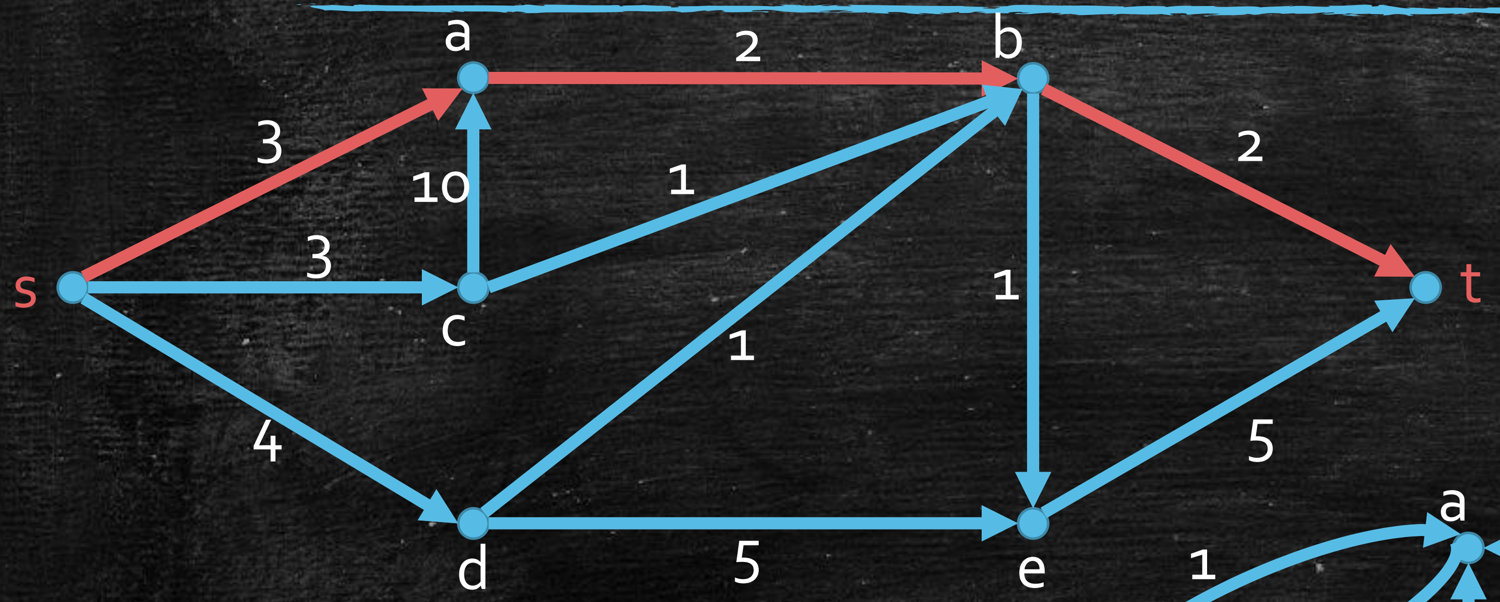
- We choose the red path!

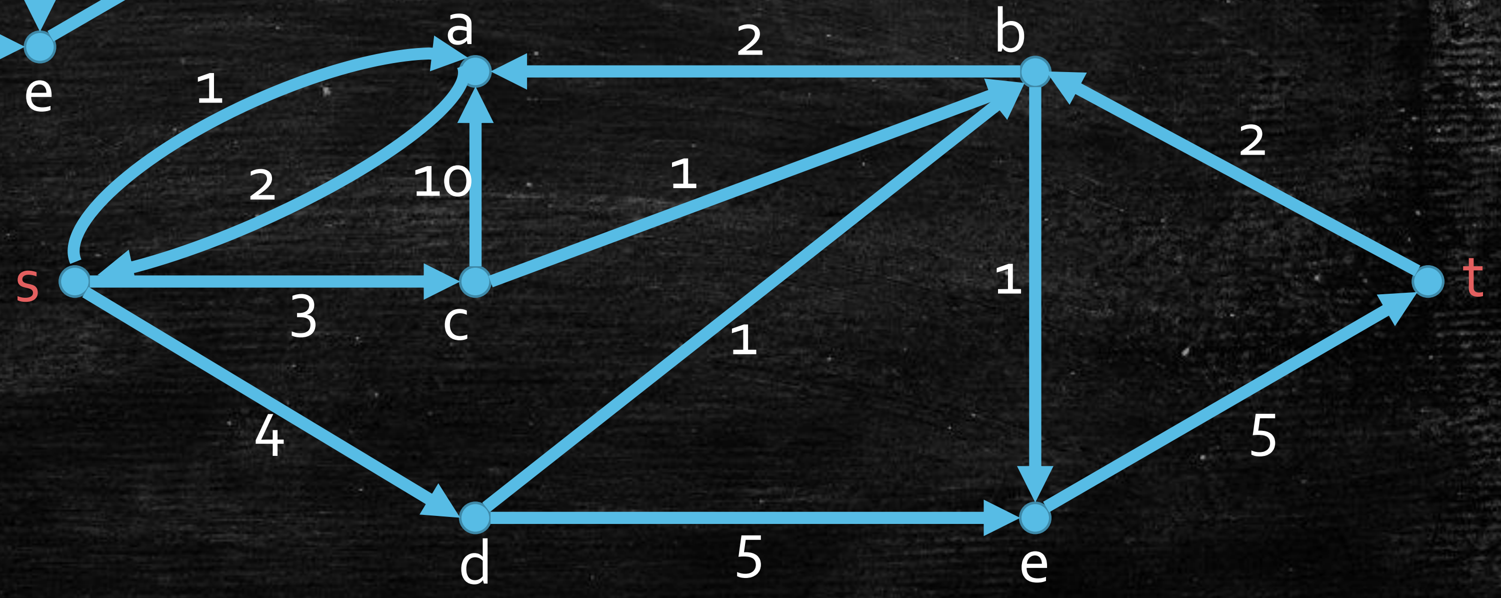


Weak Monotonicity to Strong Monotonicity

- $\mathrm{dist}(u)$ 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: $\mathrm{dist}(u)$ for all vertices remain unchanged after an iteration.





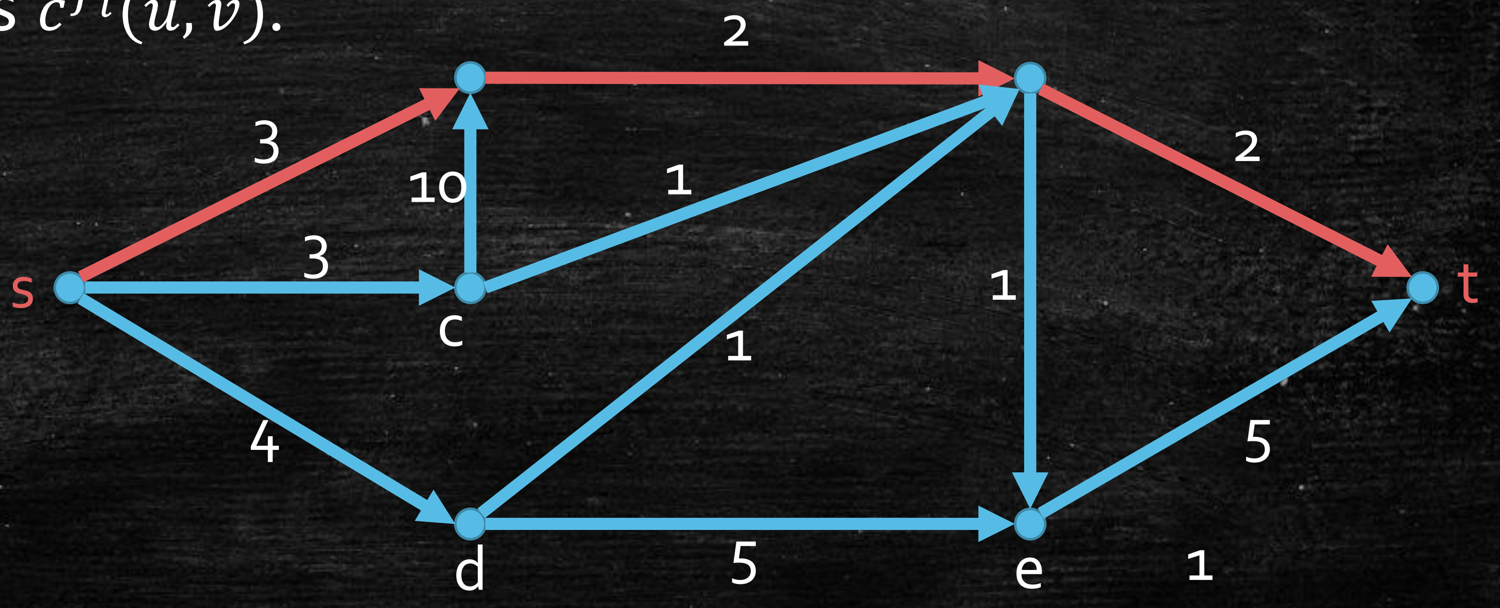
Towards Strong Monotonicity…

- Observation: At least one edge (u, v) on p is saturated, and this edge will be deleted in the next iteration.
- Each iteration will remove 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 $G^{f_{i+1}}$.

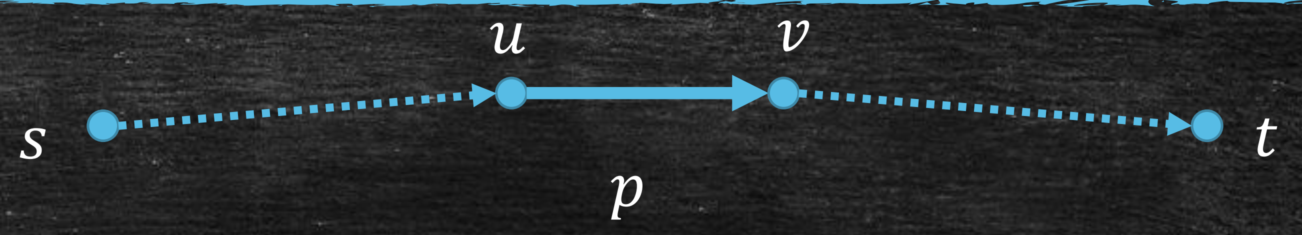


Towards Strong Monotonicity

- 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"

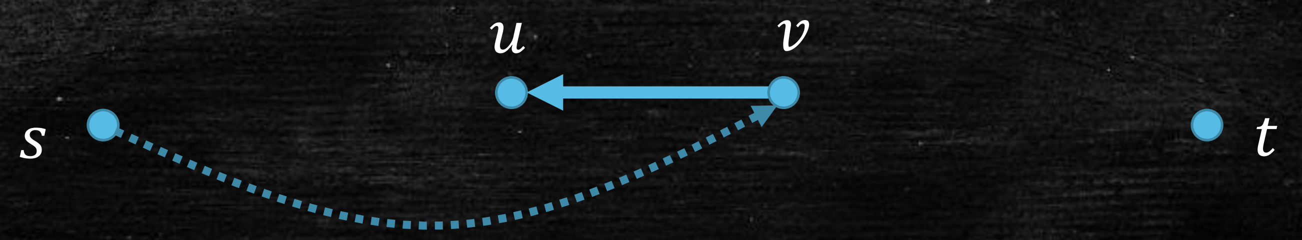
A flow along p in G^{f_i} where (u, v) becomes critical



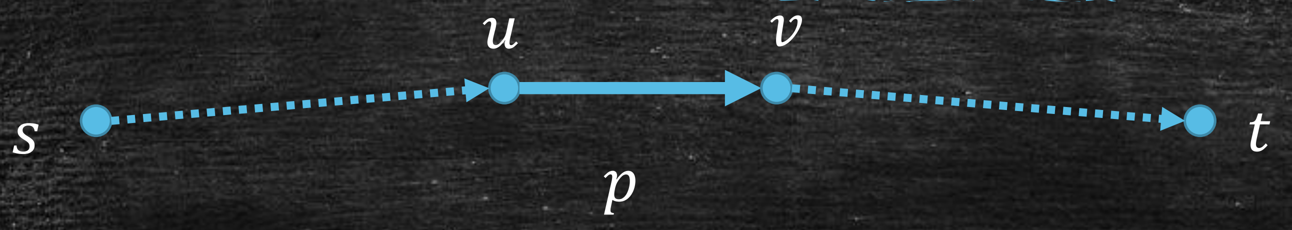
In $G^{f_{i+1}}$, (u, v) disappears, and (v, u) appears.

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

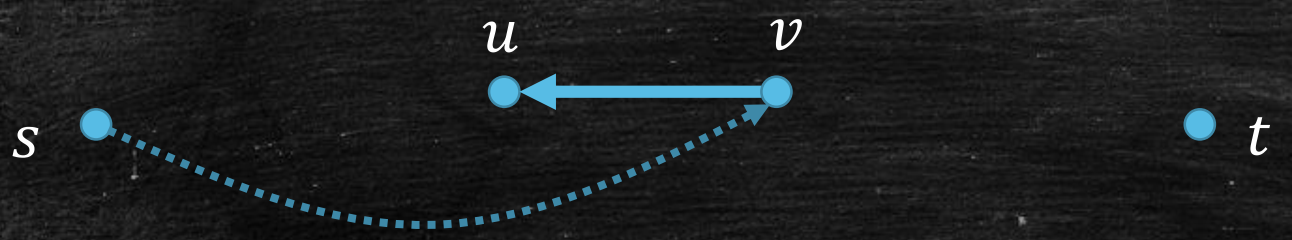
Before (u, v) reappears, the algorithm must have found p going through (v, u).



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.

Putting Together

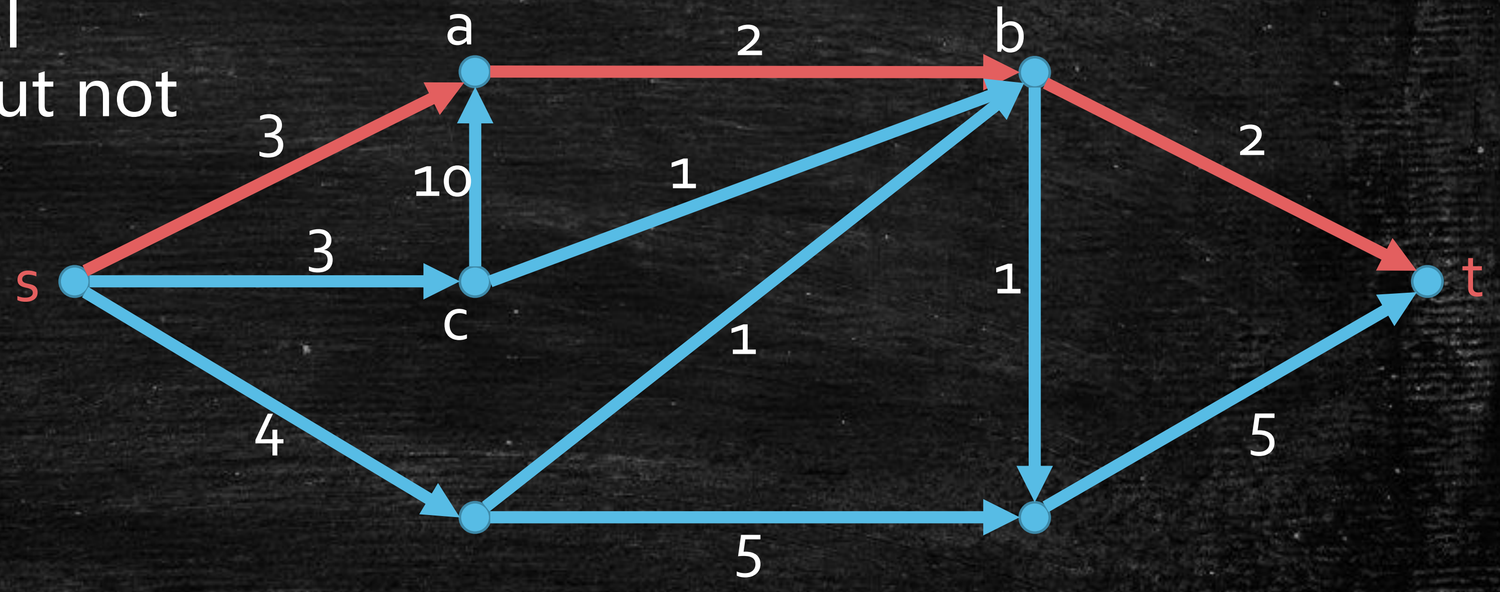
- The distance of u from s increases by 2 between two "critical".
- Distance takes value from $\{0, 1, \ldots, |V|, \infty\}$, and never decrease.
- Thus, each edge can only be critical for $O(|V|)$ times.
- At least 1 edge becomes critical in one iteration.
- Total number of iterations is $O(|V| \cdot |E|)$.
- Each iteration takes $O(|E|)$ time.
- Overall time complexity for Edmonds-Karp: $O(|V| \cdot |E|^2)$.
- It can handle the issue with irrational numbers!

Can we improve?

- Learn from proof.
- $dist[u]$ is non-decreasing but not strictly increasing.
- Can we try to make some $dist$ strictly increasing?
- What if $dist[t]$ is strictly increasing?
  - $O(|V|)$ rounds increasing.

How to make it?

- We want to increase $dist[t]$.
- We should remove all shortest $s \rightarrow t$ path but not only one.

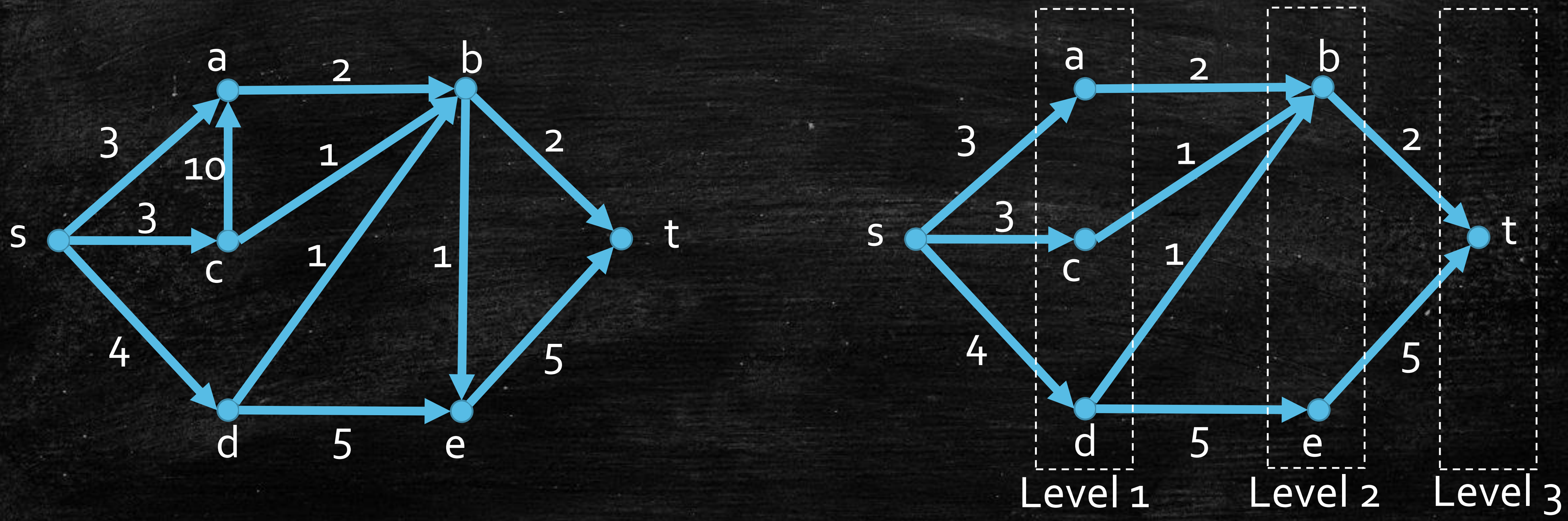


Dinic’s Algorithm (Dinitz’s Algorithm)

- Proposed by Yefim Dinitz (Soviet→ Israeli), in1970.
  - Independent on Edmonds–Karp (1972).
- Updated by Shimon Even (Israeli). and Alon Itai (Israeli).
- Even gave lectures on "Dinic's algorithm", mispronouncing the name of the author while popularizing it.
- Time complexity: $O(|V|^2 \cdot |E|)$
  - Edmonds-Karp: $O(|V| \cdot |E|^2)$

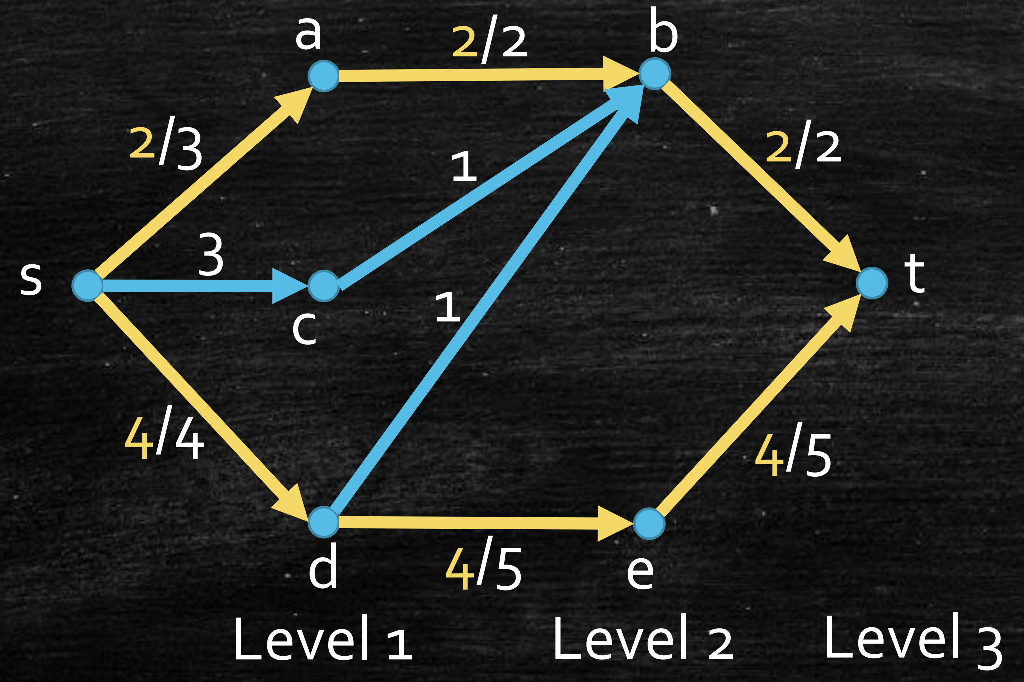
Dinic's Algorithm – high-level ideas

- Build a level graph:
  - Vertices at Level i are at distance i.
  - Only edges go from a level to the next level are kept.
  - Can be done in $O(|E|)$ time using a similar idea to BFS.



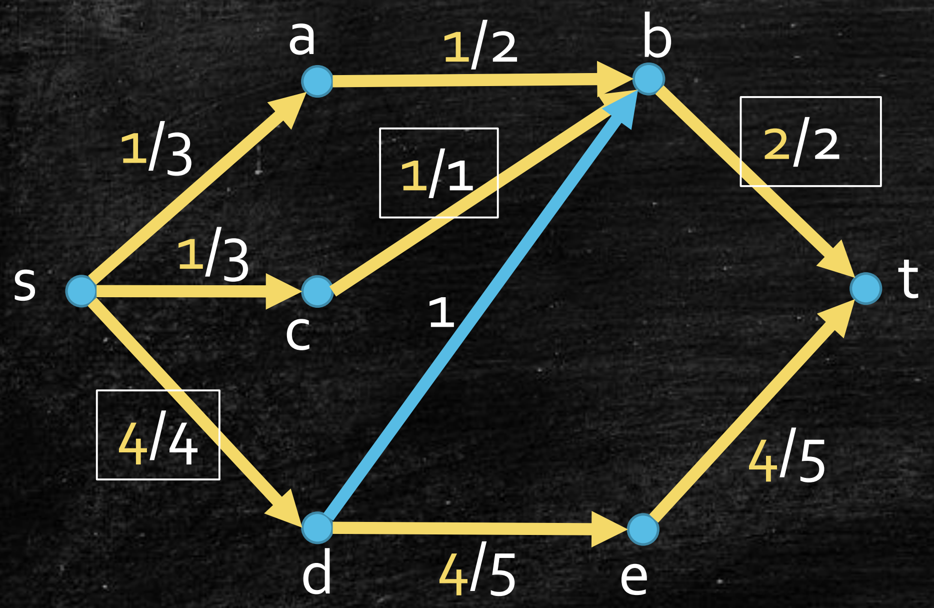
Dinic's Algorithm – high-level ideas

- Find a blocking flow on the level graph:
  - Push flow on multiple s-t paths.
  - Each s-t path must contain a critical edge!

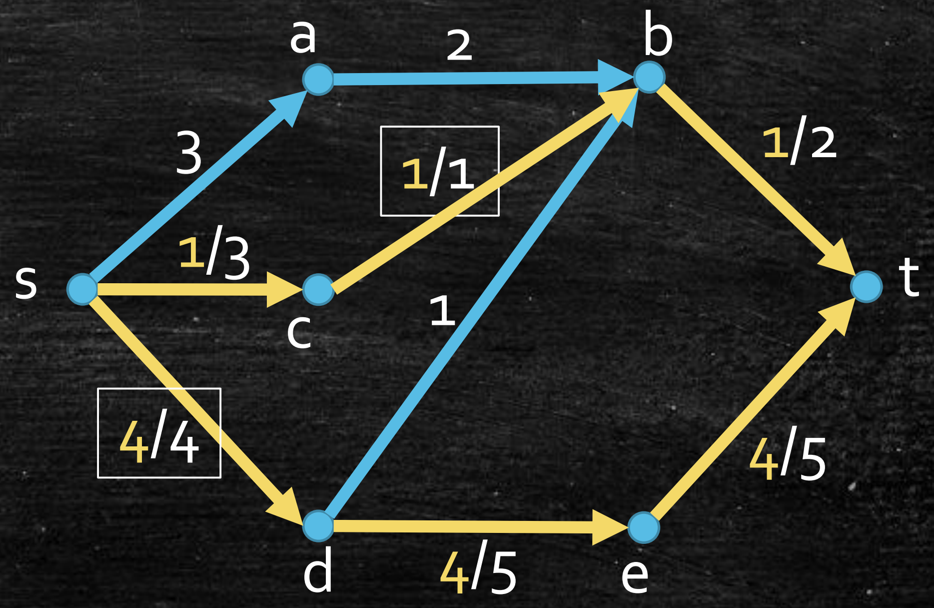


Dinic's Algorithm – high-level ideas

- Find a blocking flow on the level graph:
  - Push flow on multiple s-t paths.
  - Each s-t path must contain a critical edge!



a blocking flow



not a blocking flow: path s-a-b-t contains no critical edge

Dinic's Algorithm – Overview

- Initialize f to be the empty flow and $G^f = G$.
- Iteratively do the followings until $\mathrm{dist}(t) = \infty$:
  - Construct the level graph G_L^f for G^f.
  - Find a blocking flow on G_L^f.
  - Update f and G^f.

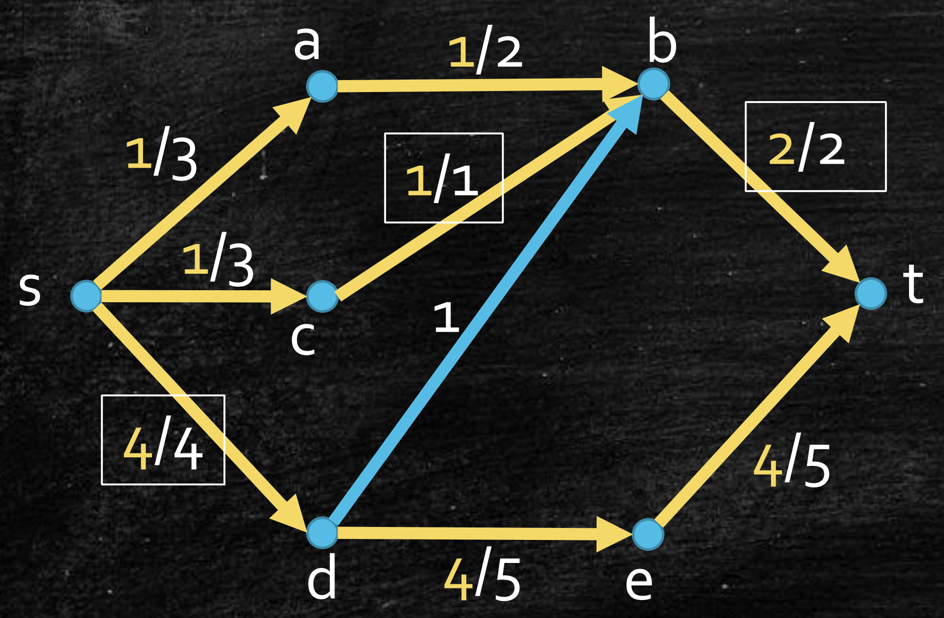
Questions Remain

1. How many iterations do we need before termination?
2. How do we find a blocking flow?

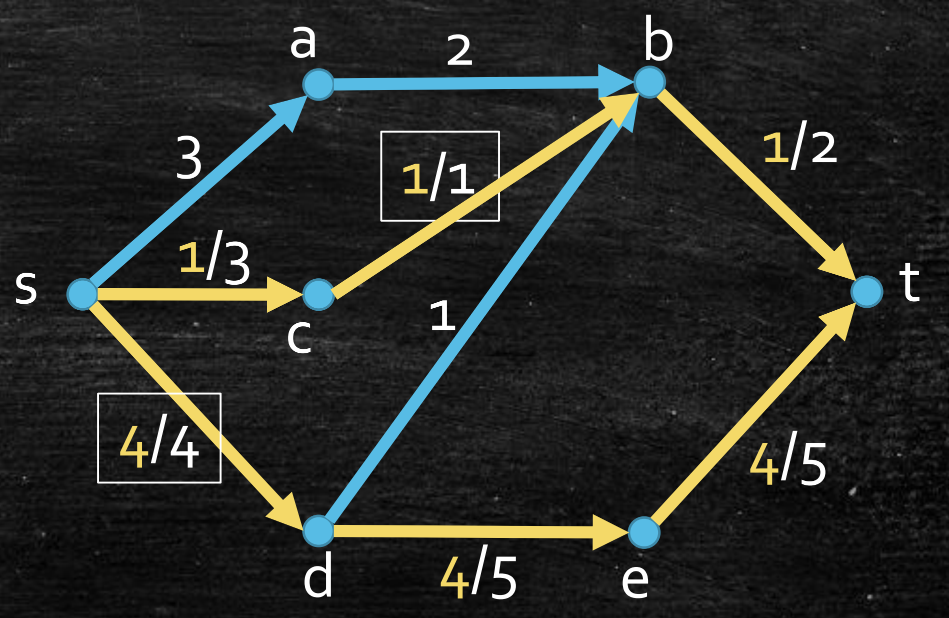
How to find a block flow?

Dinic's Algorithm – high-level ideas

- Find a blocking flow on the level graph:
  - Push flow on multiple s-t paths.
  - Each s-t path must contain a critical edge!



a blocking flow



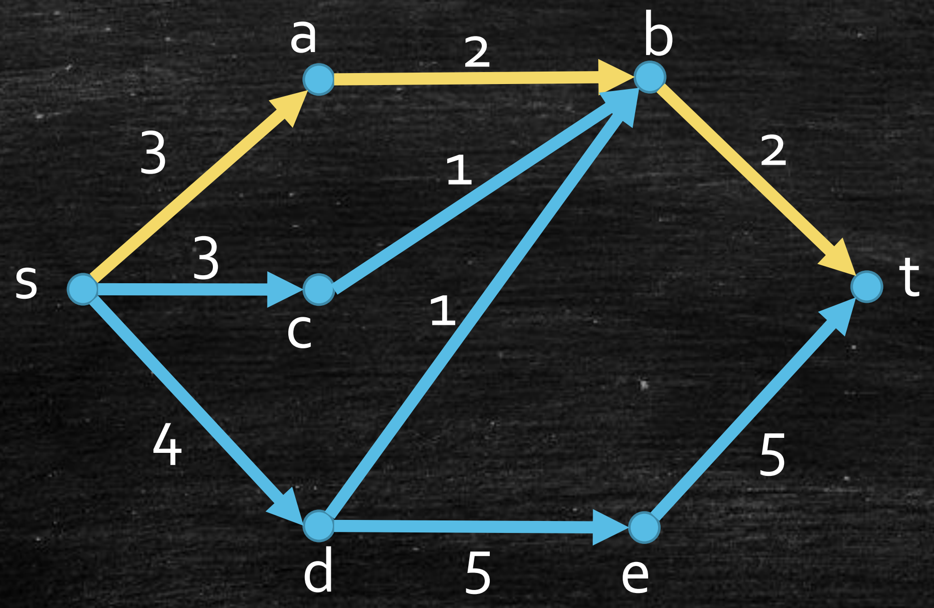
not a blocking flow: path s-a-b-t contains no critical edge

Finding a blocking flow in a level graph…

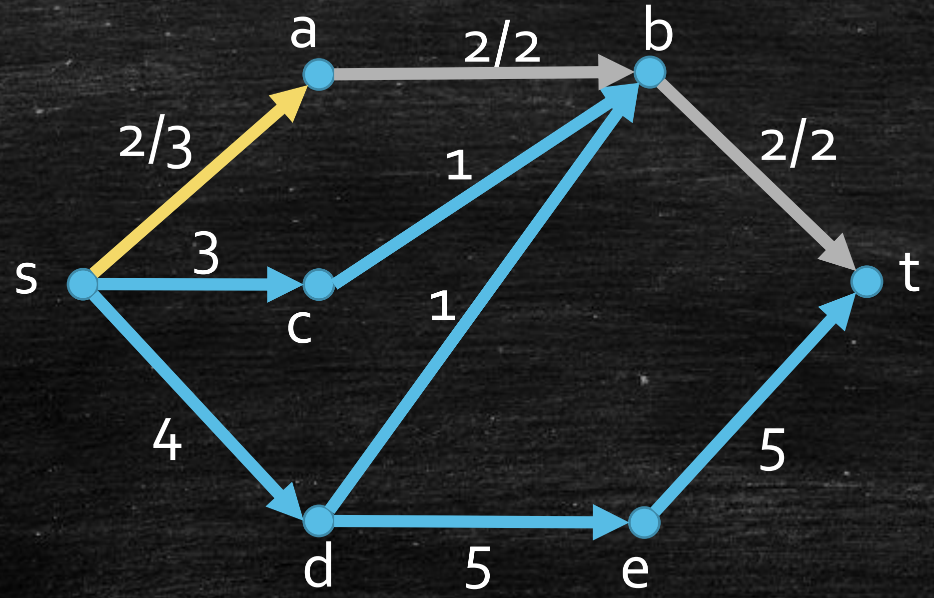
Iteratively do the followings, until no path from s to t:

- Run a frontward DFS.
- Two possibilities:
  - End up at t: in this case, we update f (by pushing flow along the path) and remove the critical edge
  - End up at a dead-end, a vertex v with no out-going edges in G_L^f: in this case, we remove all the incoming edges of v

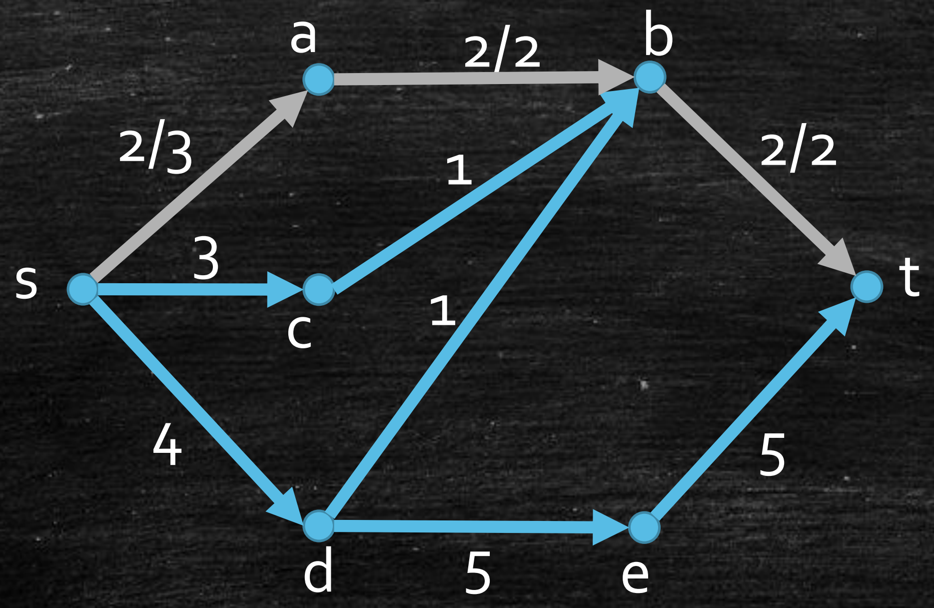
Sample Run.



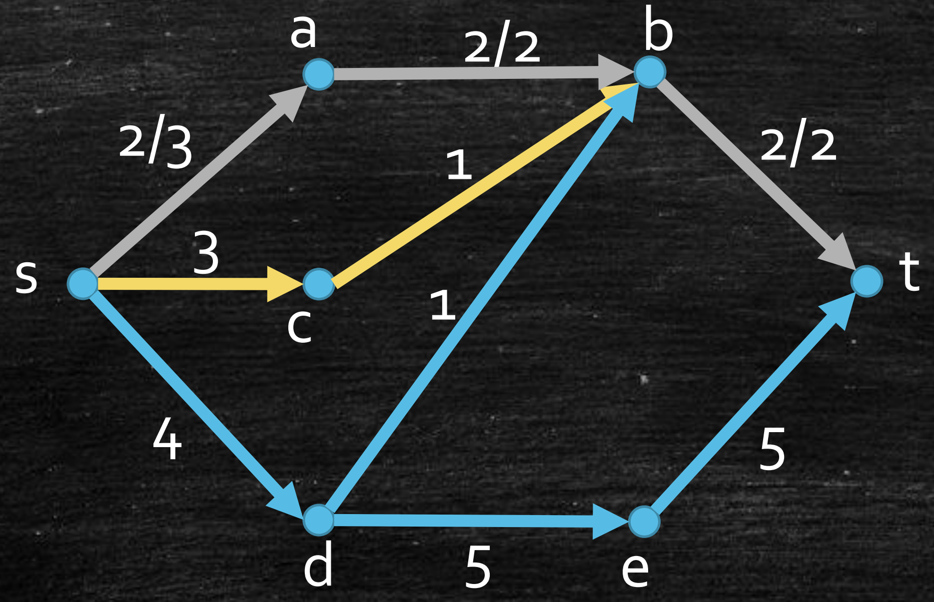
Sample Run.



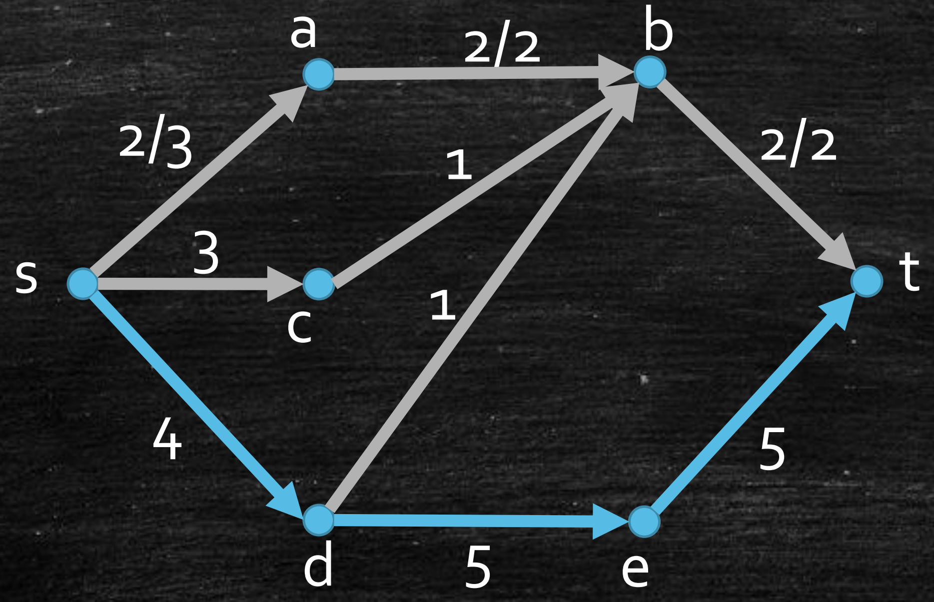
Sample Run.



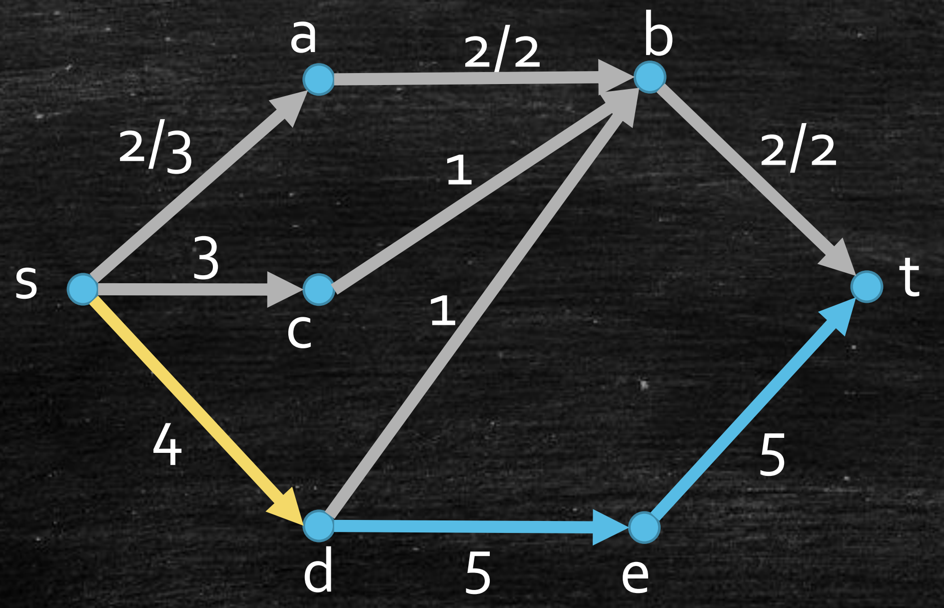
Sample Run.



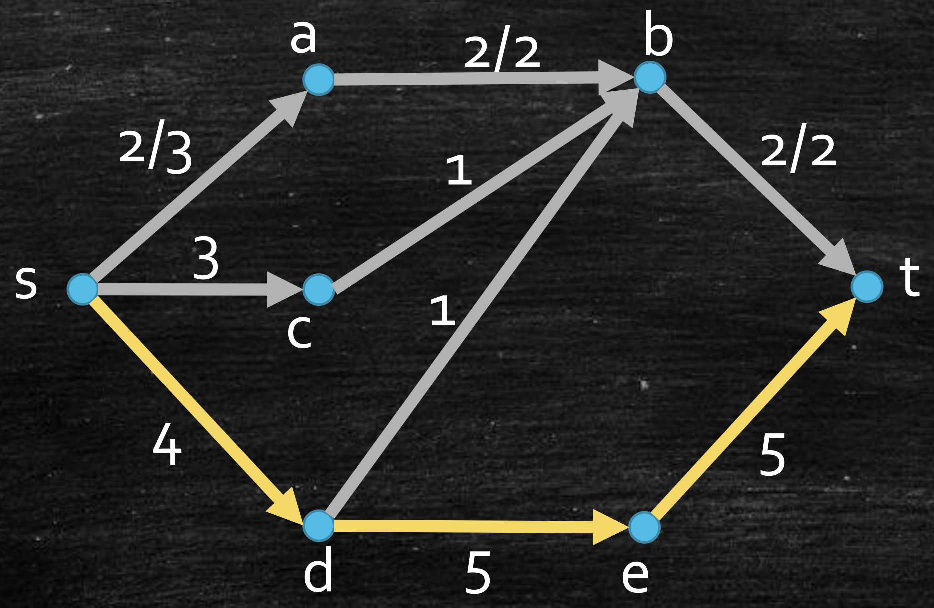
Sample Run.



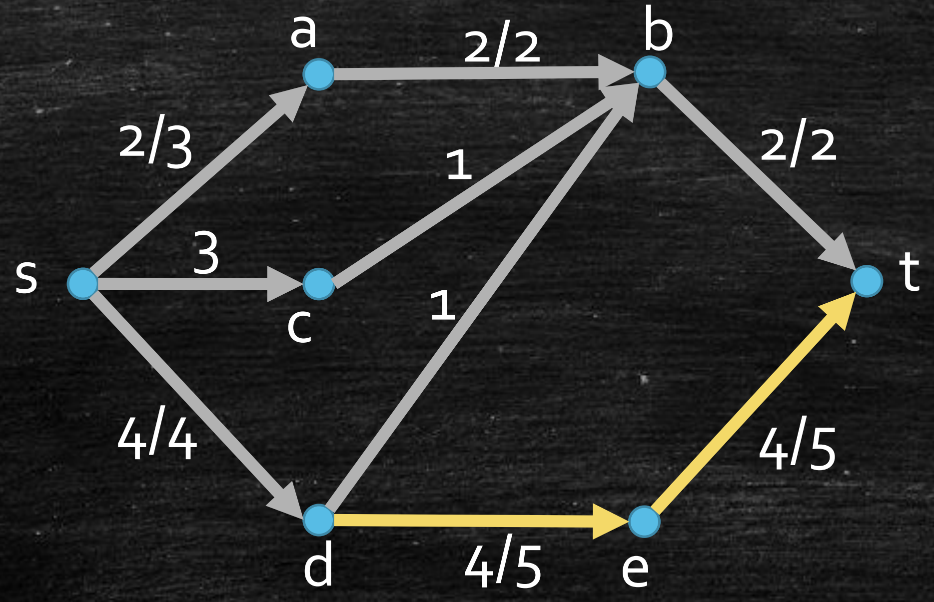
Sample Run.



Sample Run.



Sample Run.



Finding a blocking flow in a level graph…

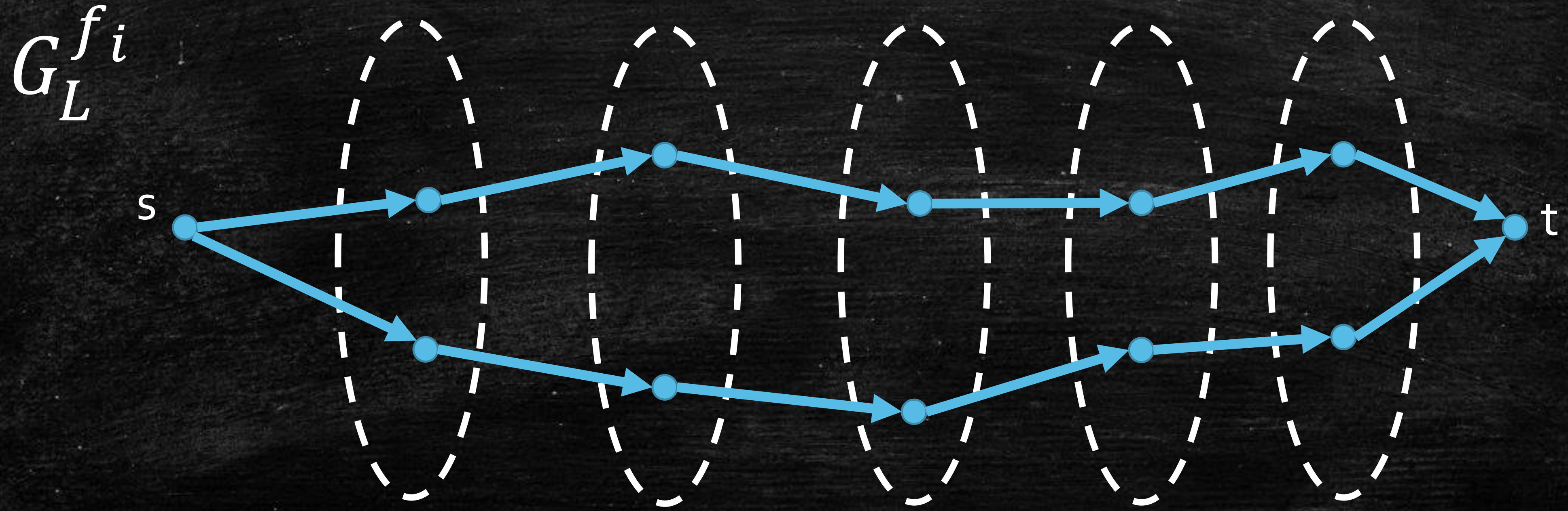
- At least one edge is removed after each search.
  - Total number of searches: $O(|E|)$
- Each search takes at most $|V|$ steps.
- Time complexity for each iteration of Dinic’s algorithm: $O(|V| \cdot |E|)$.

Questions Remain

1. How do we find a blocking flow? ✓
2. How many iterations do we need before termination?

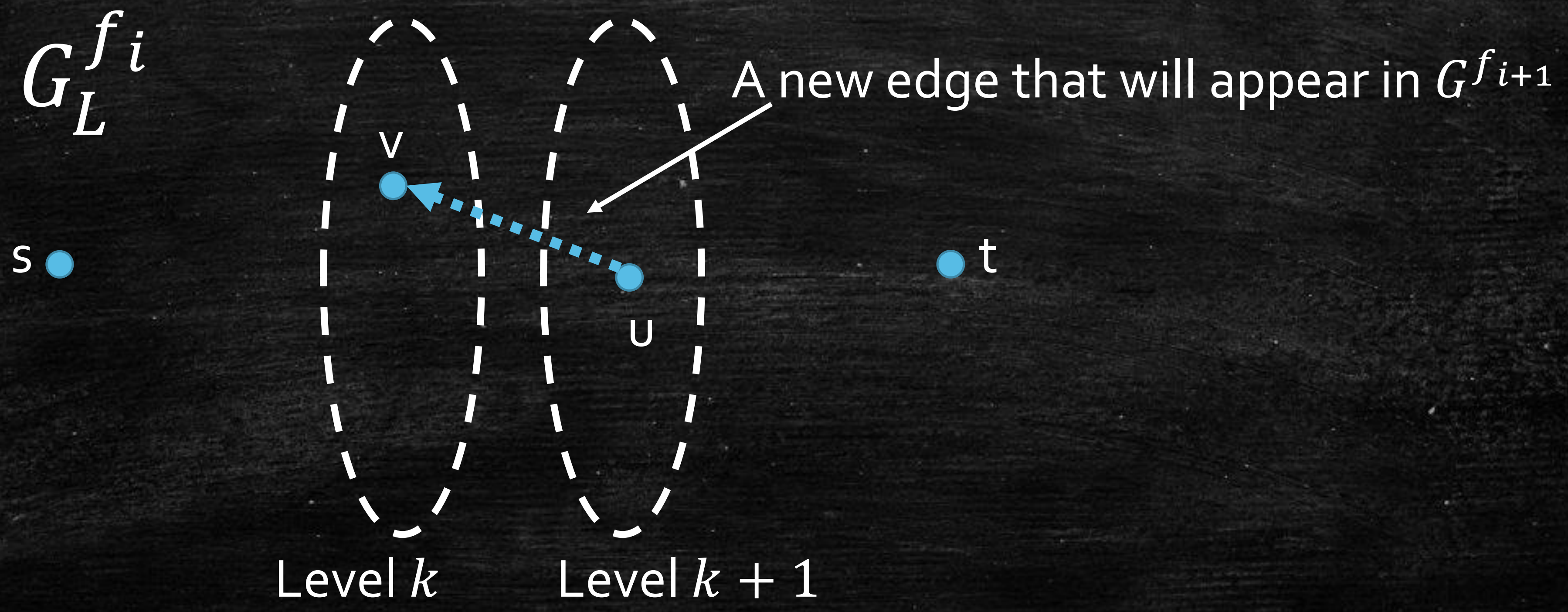
Simple yet important observations

- In the level graph $G_L^{f_i}$ at every iteration i, every s-t path has length $\mathrm{dist}^i(t)$.
- Every shortest s-t path in G^{f_i} also appears in $G_L^{f_i}$.



Distance Monotonicity

- After one iteration, a new edge (u, v) appearing in $G^{f_{i+1}}$ (but not in G^{f_i}) must be "backward": $\mathrm{dist}^i(u) = \mathrm{dist}^i(v) + 1$.



Distance Monotonicity

- After one iteration, a new edge (u, v) appearing in $G^{f_{i+1}}$ (but not in G^{f_i}) must satisfy $\mathrm{dist}^i(u) = \mathrm{dist}^i(v) + 1$.
- Such additions of edges cannot reduce the distance for any vertex!
- We again have that $\mathrm{dist}(u)$ is non-decreasing!
- Can we have strong monotonicity?

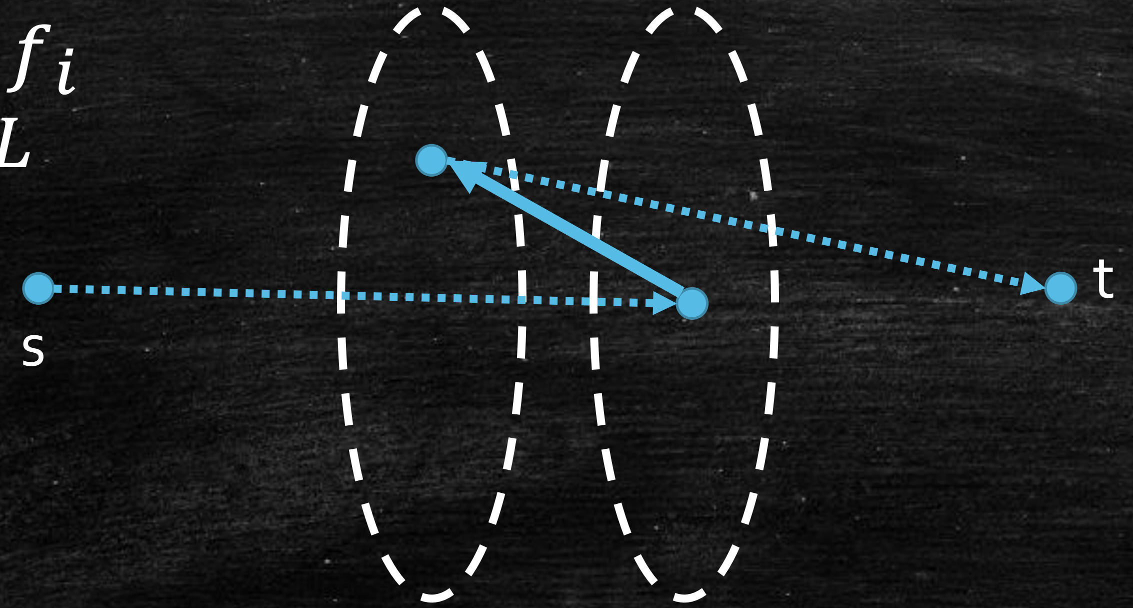
Taking a closer look…

- All the paths in $G_L^{f_i}$ with length $\mathrm{dist}^i(t)$ are “blocked” after the i-th iteration.
- Thus, a path in the $(i+1)$-th iteration must use some edges that are not in $G_L^{f_i}$.

Taking a closer look…

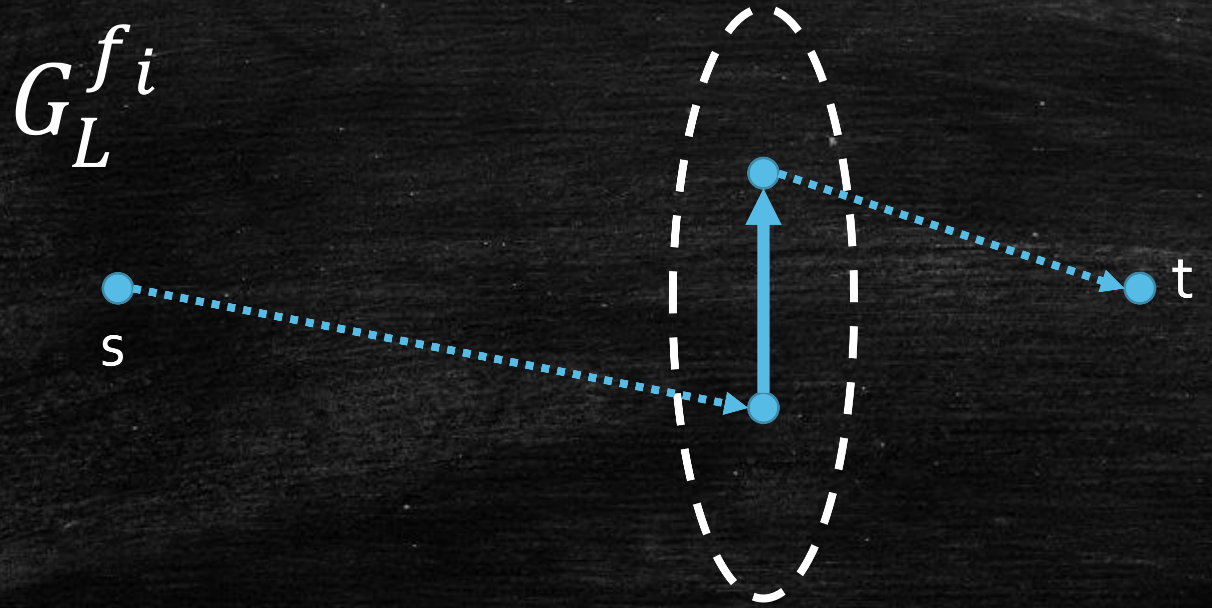
- This new edge may be a “backward” edge whose reverse was a critical edge in the previous iteration.
- In this case, $\mathrm{dist}(t)$ is increased by at least 2.

$G_L^{f_i}$



Taking a closer look…

- Or it may be an edge in G^{f_i}, but not in $G_L^{f_i}$.
- In this case, $\mathrm{dist}(t)$ is increased by at least 1.



Taking a closer look…

- In both cases: $\mathrm{dist}^{i+1}(t) > \mathrm{dist}^{i}(t)$
- Let’s prove it rigorously then…

Proving $\mathrm{dist}^{i+1}(t) > \mathrm{dist}^{i}(t)$

- Consider an arbitrary s-t path p in $G_L^{f_{i+1}}$ with length $\mathrm{dist}^{i+1}(t)$.
- We have $\mathrm{dist}^{i+1}(t) \geq \mathrm{dist}^{i}(t)$ by monotonicity.
- Suppose for the sake of contraction that $\mathrm{dist}^{i+1}(t) = \mathrm{dist}^{i}(t)$.
- Case 1: all edges in p also appear in $G_L^{f_i}$
- Then p is a shortest path containing no critical edges in $G_L^{f_i}$
- Contracting to the definition of blocking flow!

Proving $\mathrm{dist}^{i+1}(t) > \mathrm{dist}^{i}(t)$

- Case 2: p contains an edge (u, v) that is not in $G_L^{f_i}$
- If (u, v) was not in G^{f_i}, then (v, u) was critical in the last iteration. We have $\mathrm{dist}^{i}(u) = \mathrm{dist}^{i}(v) + 1$.
- If (u, v) was in G^{f_i} but not $G_L^{f_i}$, by the definition of level graph, we have $\mathrm{dist}^{i}(u) \geq \mathrm{dist}^{i}(v)$.
- In both cases above, $\mathrm{dist}^{i}(u) \geq \mathrm{dist}^{i}(v)$.
- We have $\mathrm{dist}^{i+1}(u) \geq \mathrm{dist}^{i}(u)$ by monotonicity,
- and we have $\mathrm{dist}^{i+1}(v, t) \geq \mathrm{dist}^{i}(v, t)$. (why?)

Proving $\text{dist}^{i+1}(t) > \text{dist}^{i}(t)$

- Case 2: p contains an edge (u, v) that is not in $G_L^{f_i}$
- Fact i: $\text{dist}^{i}(u) \geq \text{dist}^{i}(v)$.
- Fact ii: $\text{dist}^{i+1}(u) \geq \text{dist}^{i}(u)$.
- Fact iii: $\text{dist}^{i+1}(v, t) \geq \text{dist}^{i}(v, t)$.

Putting together:

$$\begin{aligned}
\text{dist}^{i+1}(t) &= \text{dist}^{i+1}(u) + 1 + \text{dist}^{i+1}(v, t) \\
&\geq \text{dist}^{i}(u) + 1 + \text{dist}^{i}(v, t) && \text{(Fact ii and iii)} \\
&\geq \text{dist}^{i}(v) + 1 + \text{dist}^{i}(v, t) && \text{(Fact i)} \\
&\geq \text{dist}^{i}(t) + 1 && \text{(triangle inequality)}
\end{aligned}$$

Putting Together…

- $\mathrm{dist}(t)$ is increased by at least 1 after each iteration.
- $\mathrm{dist}(t)$ takes value from $\{0, 1, \dots, |V|, \infty\}$, so it can be increased for at most $O(|V|)$ times.
- Total number of iterations is at most $O(|V|)$.

Other Algorithms for Max-Flow

- 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] $\tilde{O}\left(|E|^{\frac{3}{2}-\frac{1}{328}}\log U\right)$

Questions Remain

1. How do we find a blocking flow? ✓
2. How many iterations do we need before termination? ✓

Overall Time Complexity for Dinic's Algorithm

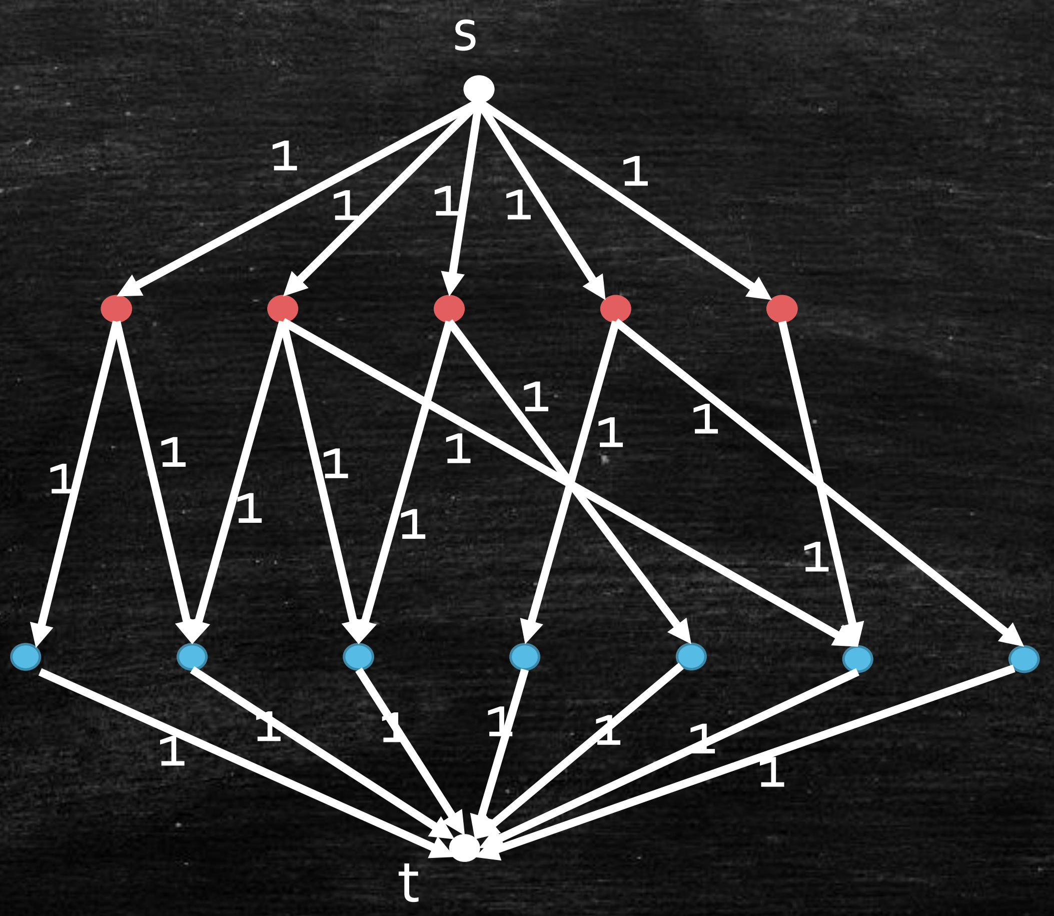
- Each iteration: $O(|V| \cdot |E|)$.
- We need at most $O(|V|)$ iterations.
- Overall time complexity for Dinic's algorithm: $O(|V|^2 \cdot |E|)$.

Hopcroft–Karp–Karzanov algorithm

- Find a maximum bipartite matching in $O\left(|E| \cdot \sqrt{|V|}\right)$ time.
- Proposed independently by Hopcroft-Karp and Karzanov.
- Can be viewed as a special case of Dinic's algorithm.

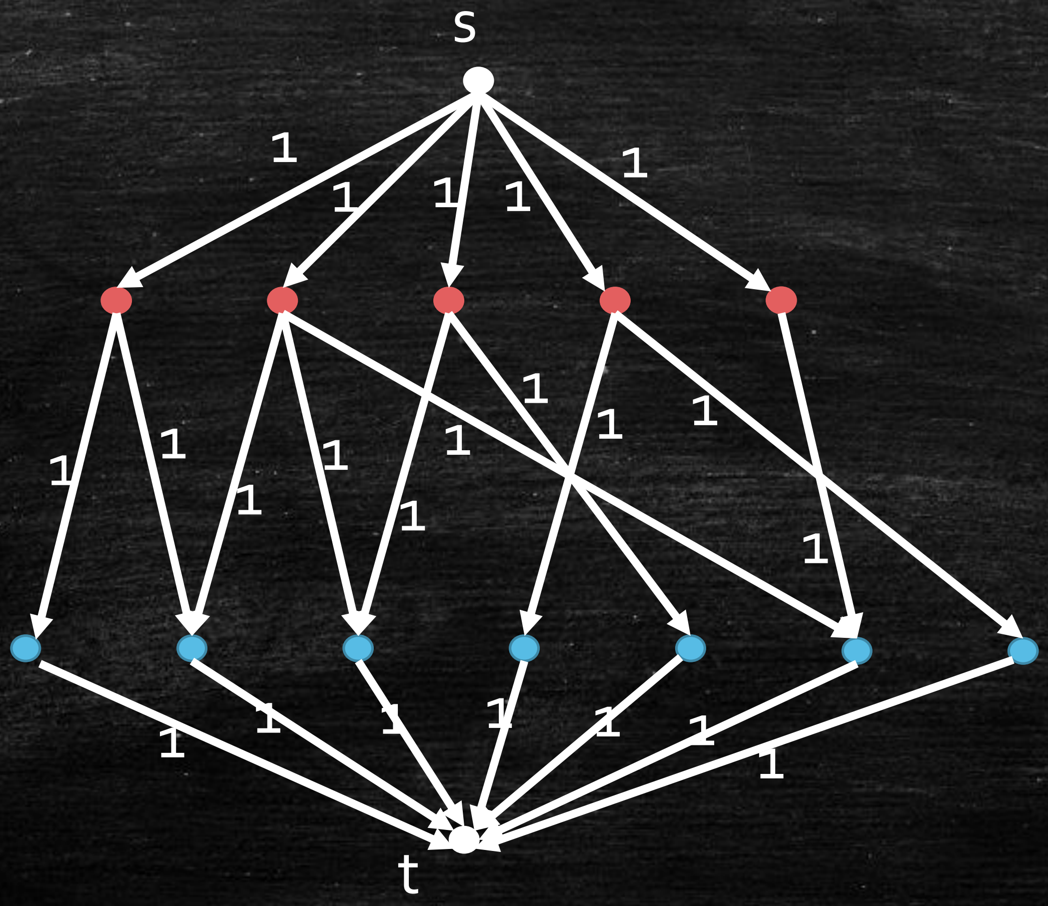
Conversion to Max-Flow Problem

Set the capacity to 1 for all edges.



Conversion to Max-Flow Problem

Dinic's algorithm runs in $O\left(|E| \cdot \sqrt{|V|}\right)$ time for this special case.



Conversion to Max-Flow Problem

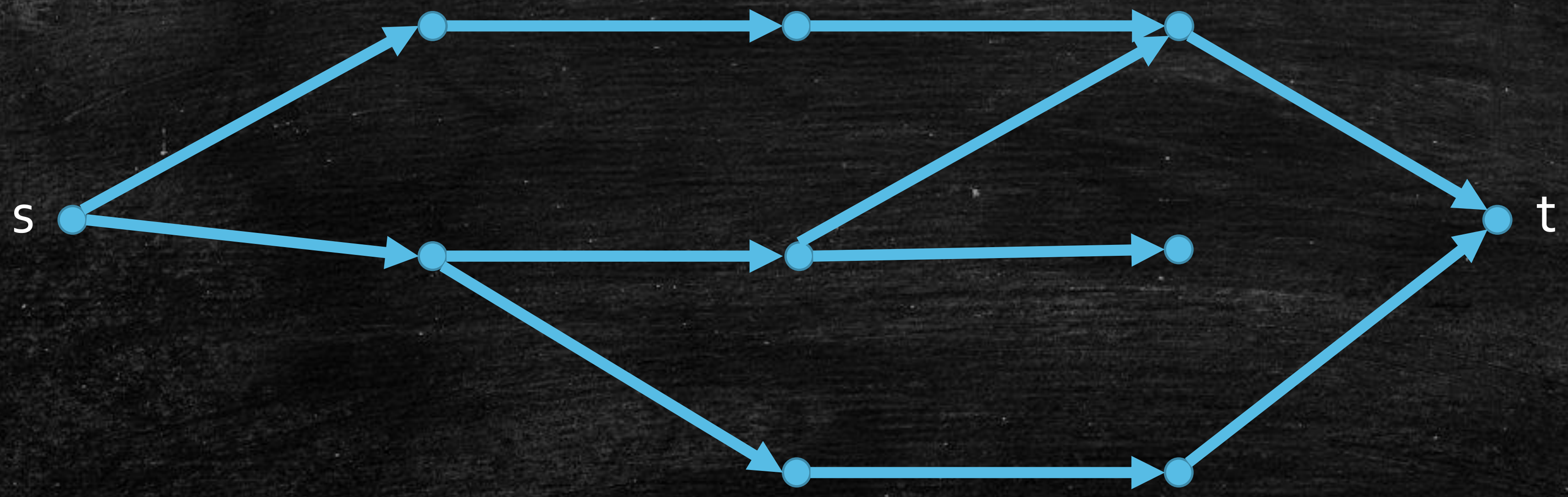
- Integrality theorem also holds for Dinic's algorithm:
  - The flow output by Dinic's algorithm in our case is integral.
- We aim to show Dinic's algorithm runs in $O\left(|E| \cdot \sqrt{|V|}\right)$ time.
- Step 1: Finding a blocking flow in a level graph takes $O(|E|)$ time.
- Step 2: Number of iterations is at most $2\sqrt{|V|}$.

Step 1: Finding a blocking flow in a level graph takes $O(|E|)$ time.

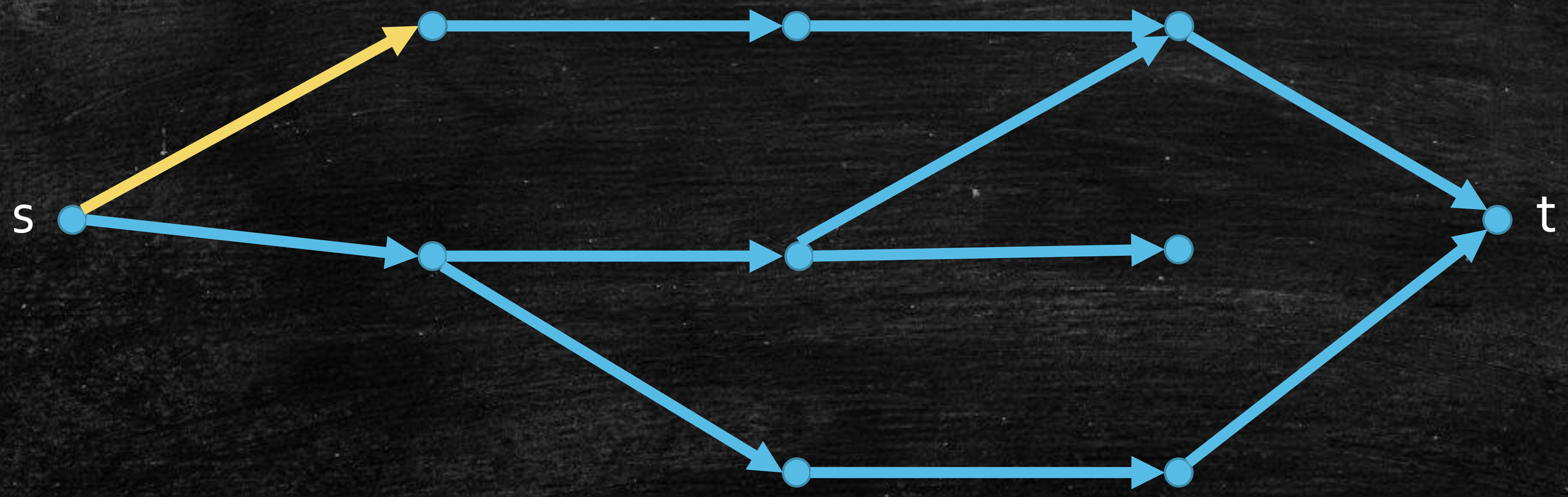
Iteratively do the followings, until no path from s to t:

- Perform DFS from s
- If we reach t, delete all edges on the s-t path (why can we do this?) and start over from s.
- If we ever go backward, delete the edge just travelled. (why can we do this?)

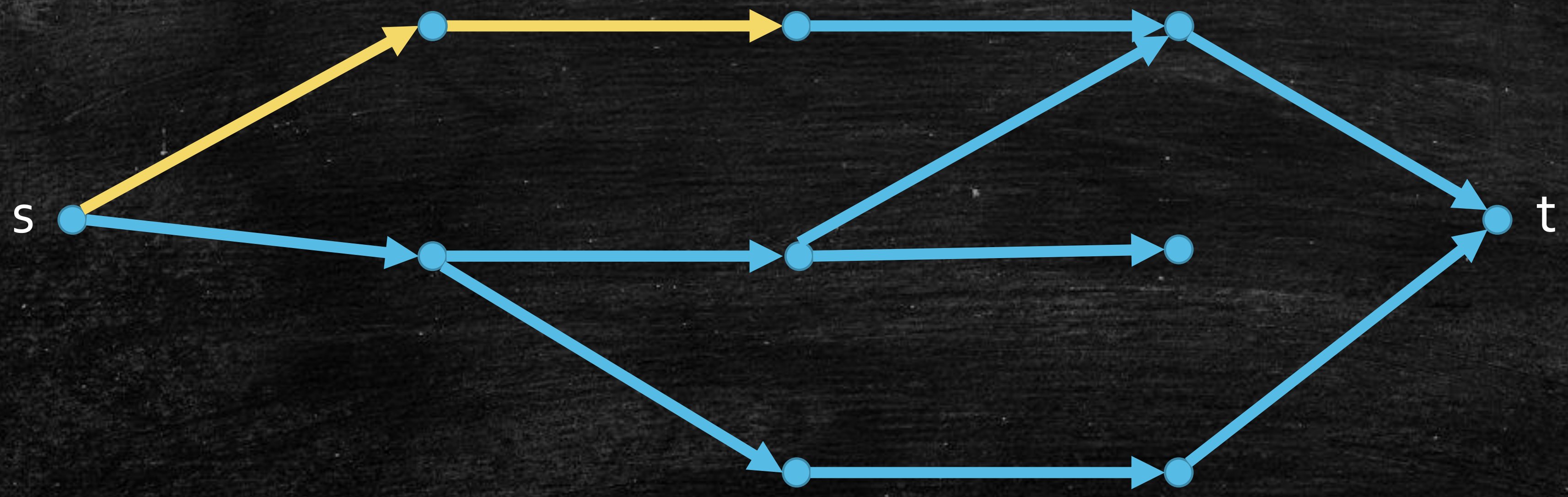
Step 1: Finding a blocking flow in a level graph takes $O(|E|)$ time.



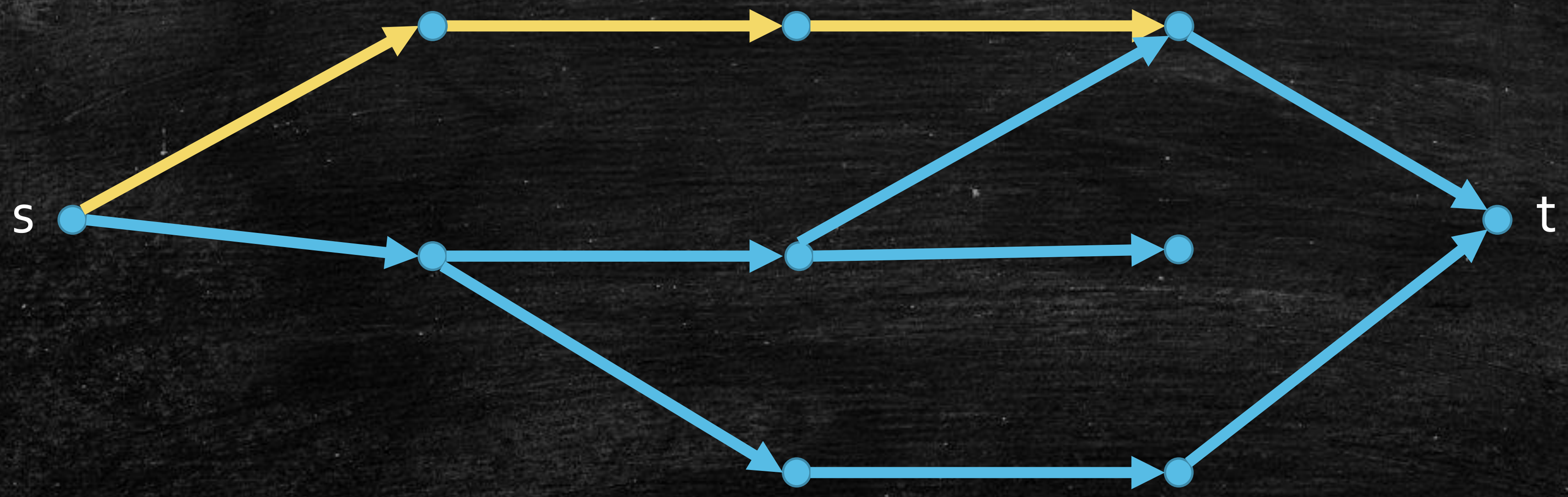
Step 1: Finding a blocking flow in a level graph takes $O(|E|)$ time.



Step 1: Finding a blocking flow in a level graph takes $O(|E|)$ time.

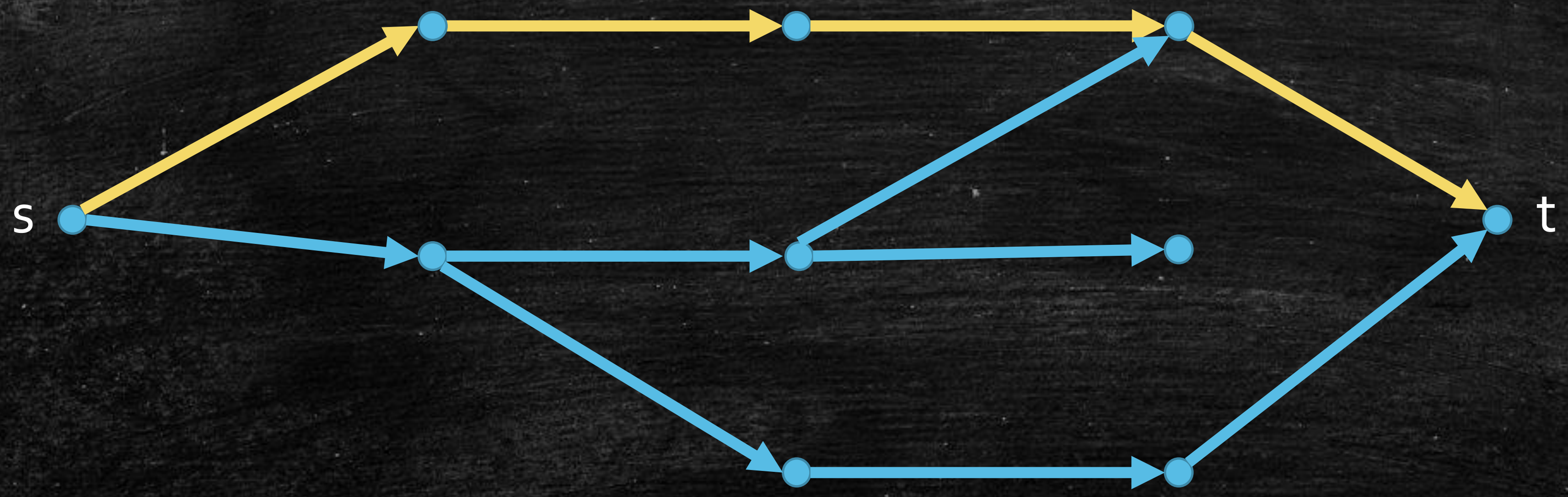


Step 1: Finding a blocking flow in a level graph takes $O(|E|)$ time.



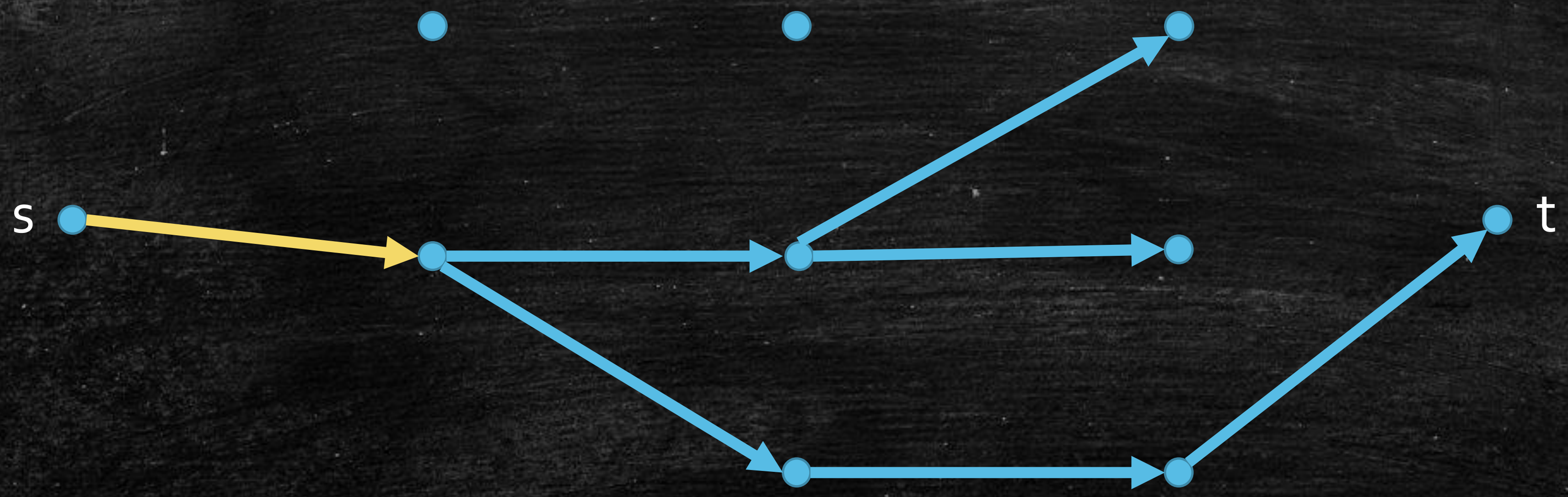
Step 1: Finding a blocking flow in a level graph takes $O(|E|)$ time.

An s-t path is found, remove all edges from the path.

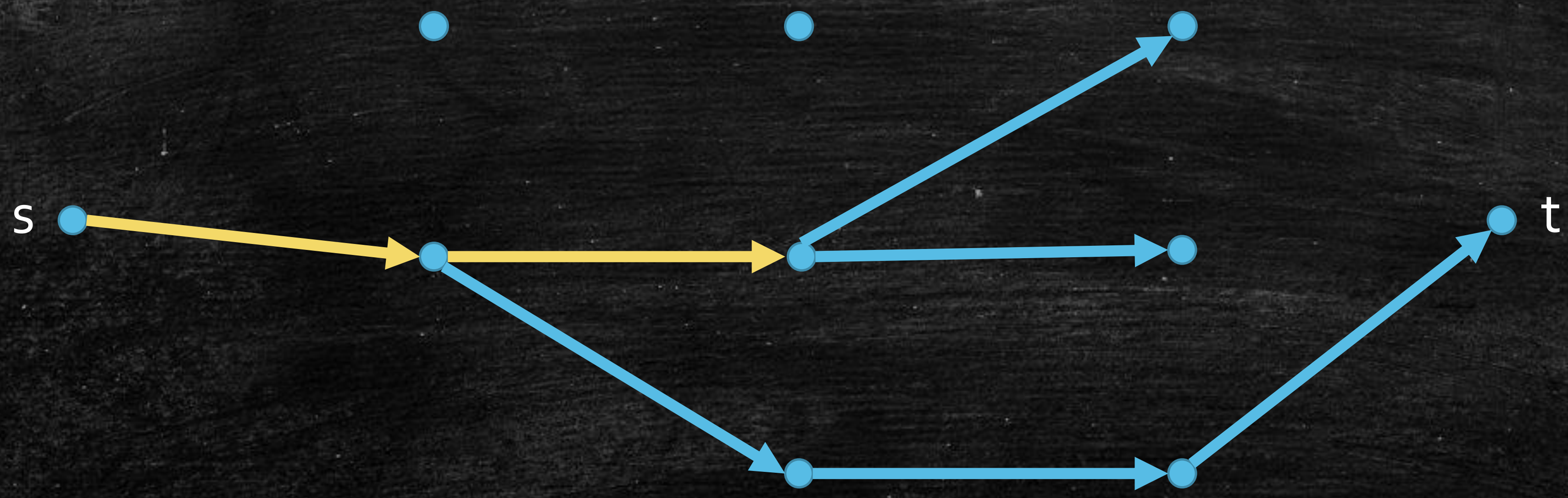


Step 1: Finding a blocking flow in a level graph takes $O(|E|)$ time.

Start over…

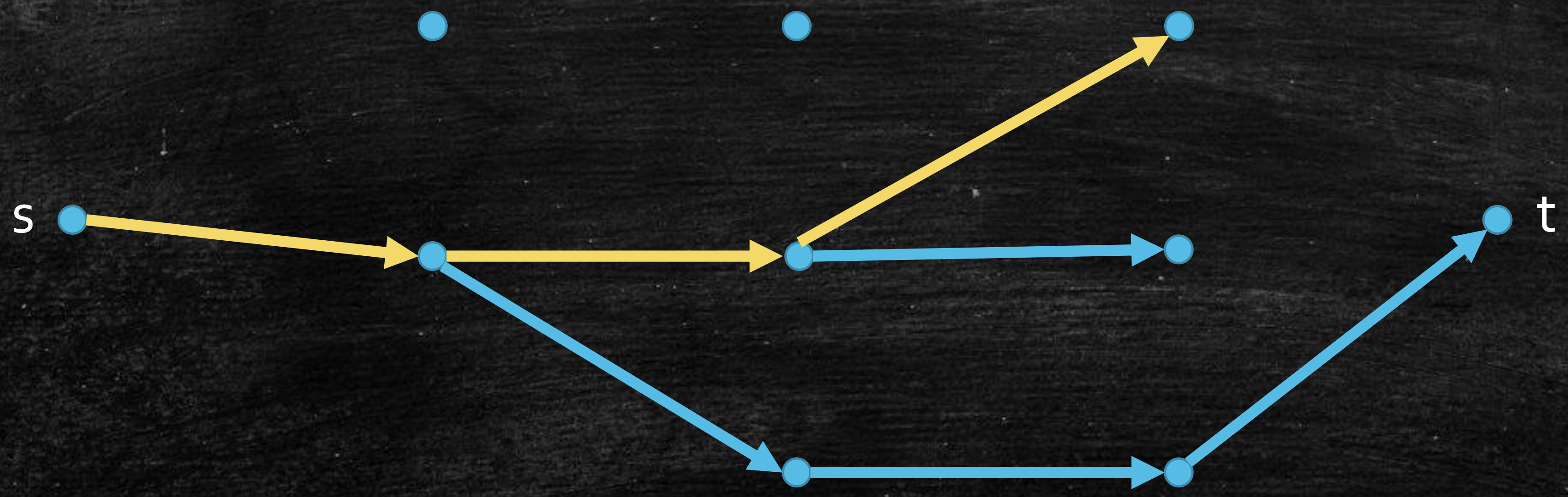


Step 1: Finding a blocking flow in a level graph takes $O(|E|)$ time.

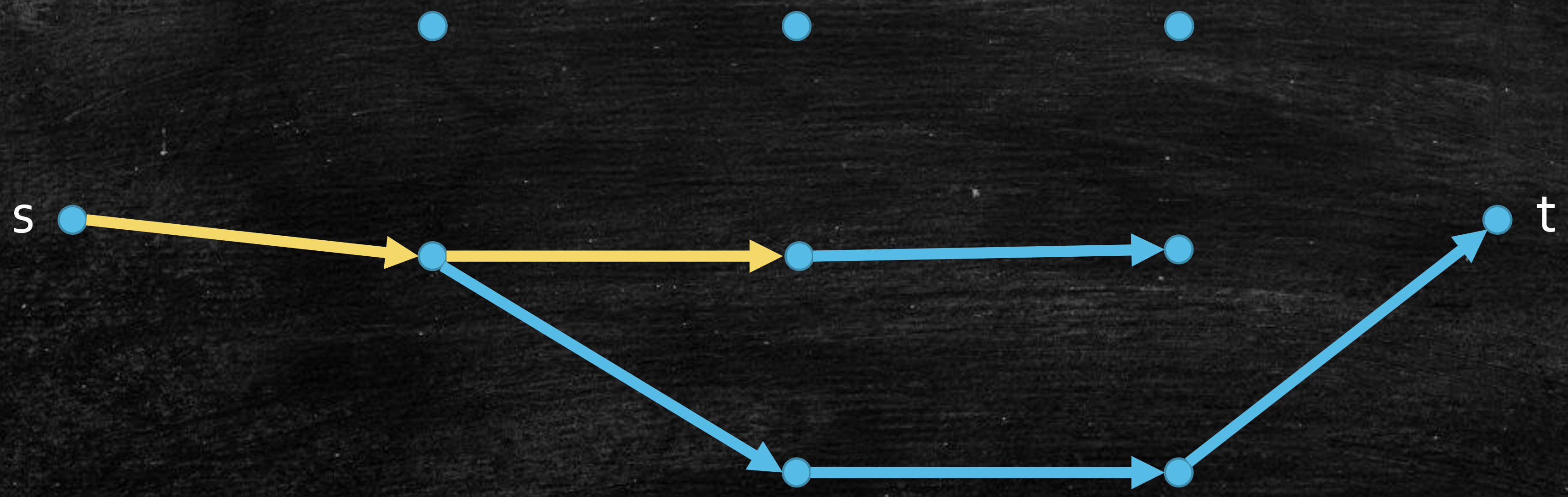


Step 1: Finding a blocking flow in a level graph takes $O(|E|)$ time.

We have to go backward now; delete the edge just travelled.

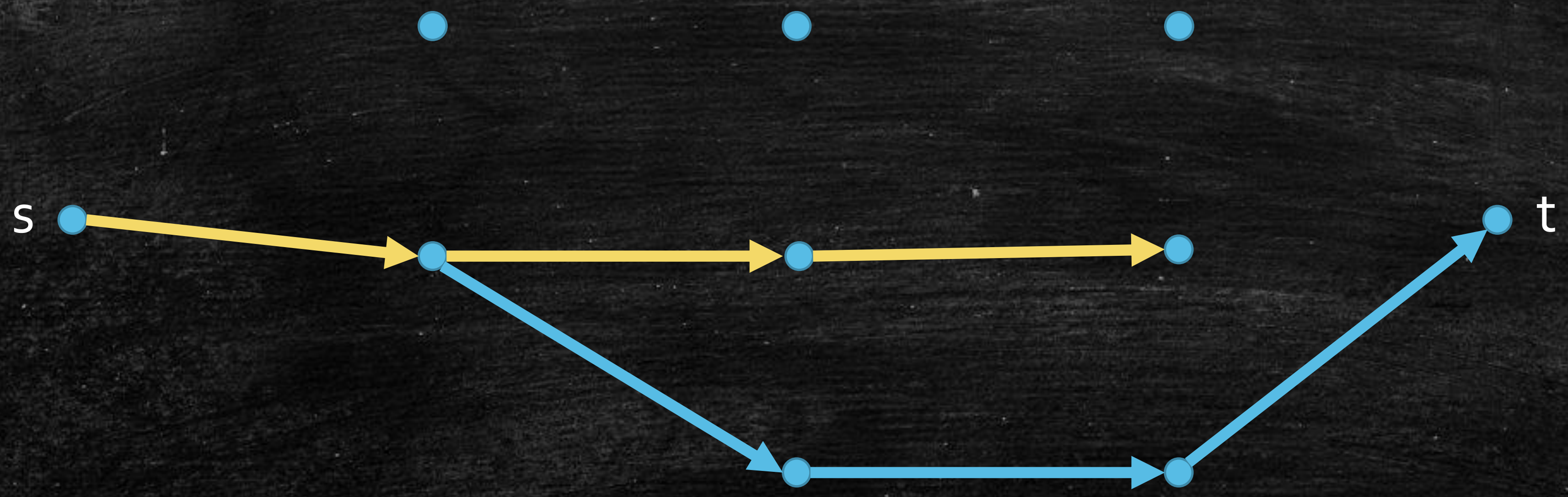


Step 1: Finding a blocking flow in a level graph takes $O(|E|)$ time.



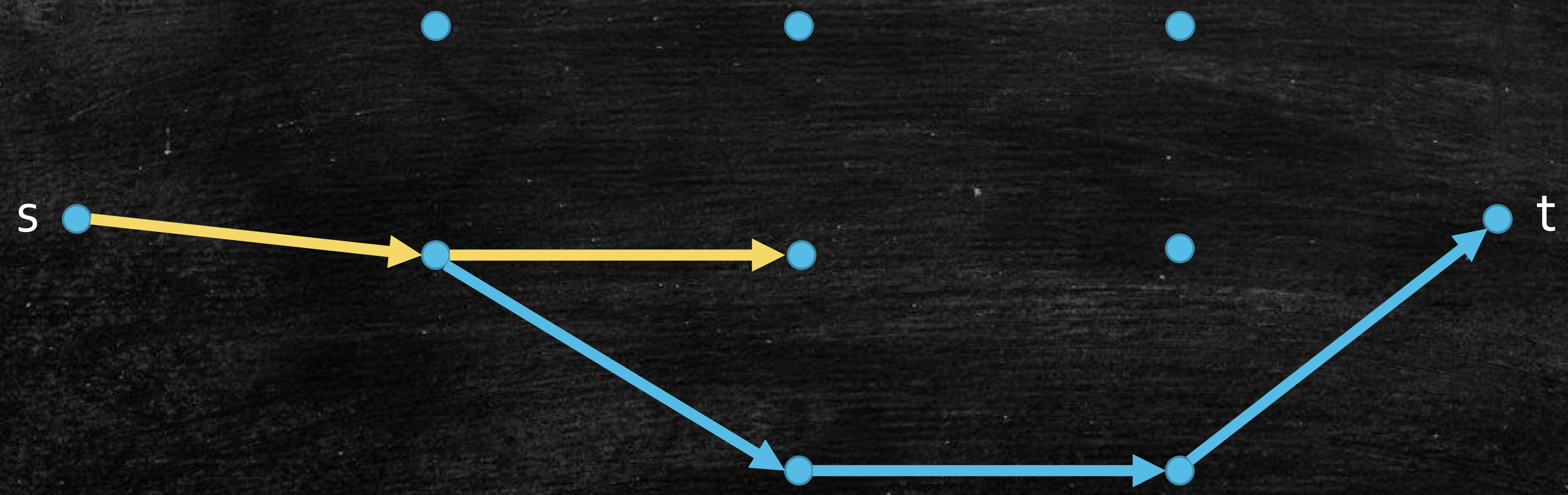
Step 1: Finding a blocking flow in a level graph takes $O(|E|)$ time.

Again, we have to go backward; delete the edge just travelled.



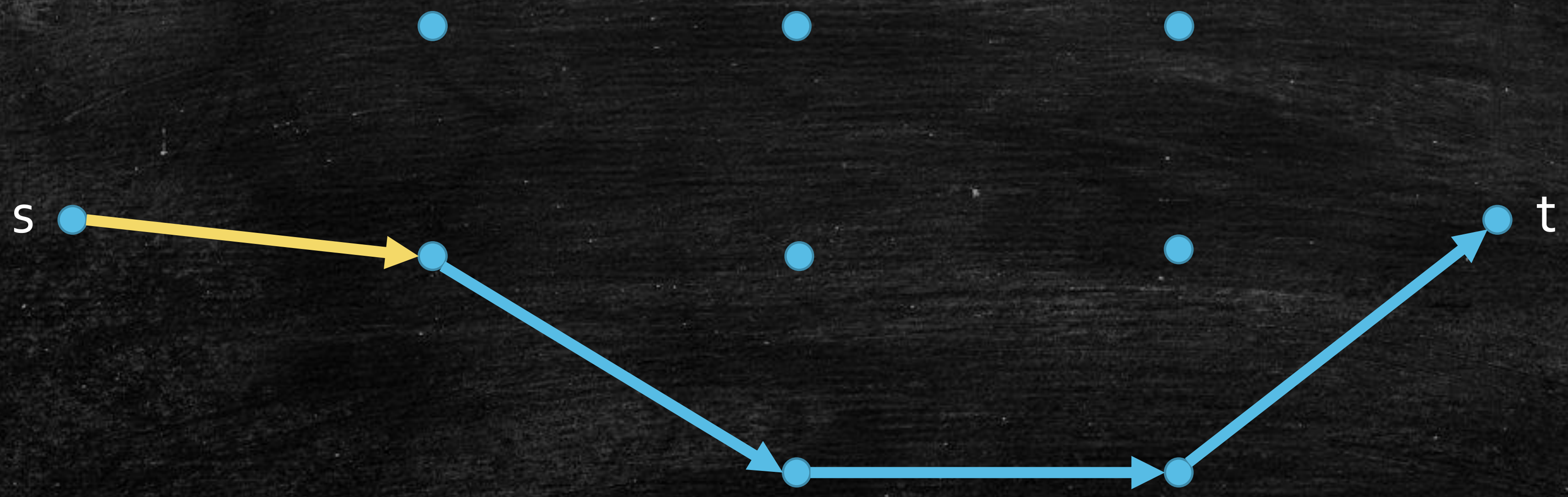
Step 1: Finding a blocking flow in a level graph takes $O(|E|)$ time.

Again, we have to go backward; delete the edge just travelled.

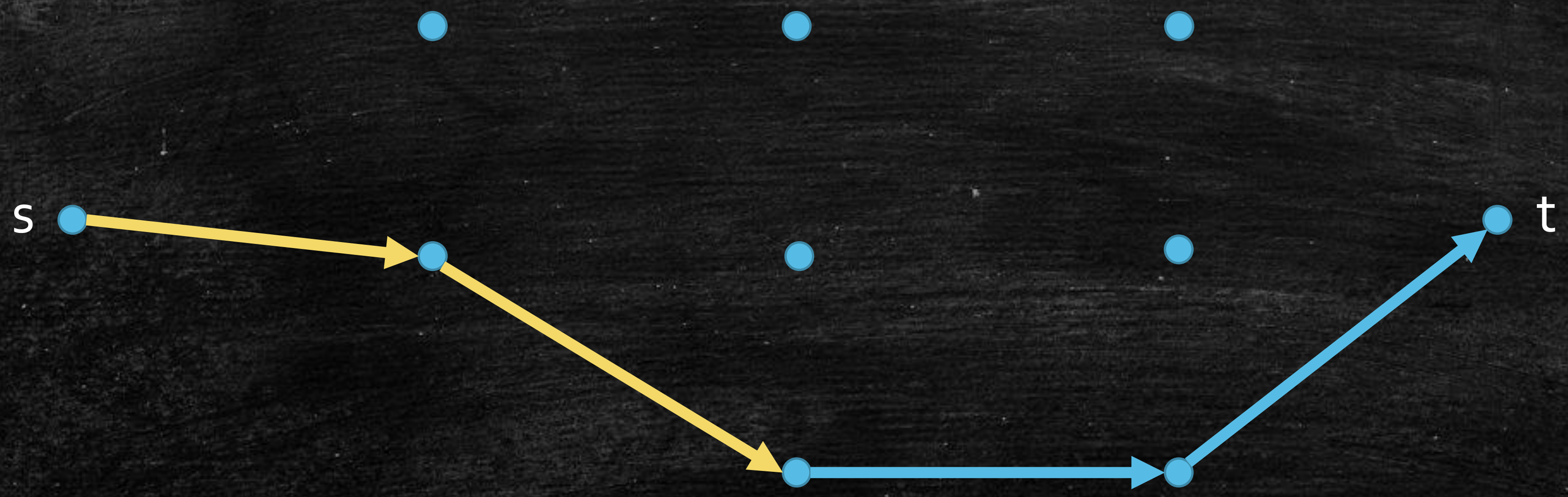


Step 1: Finding a blocking flow in a level graph takes $O(|E|)$ time.

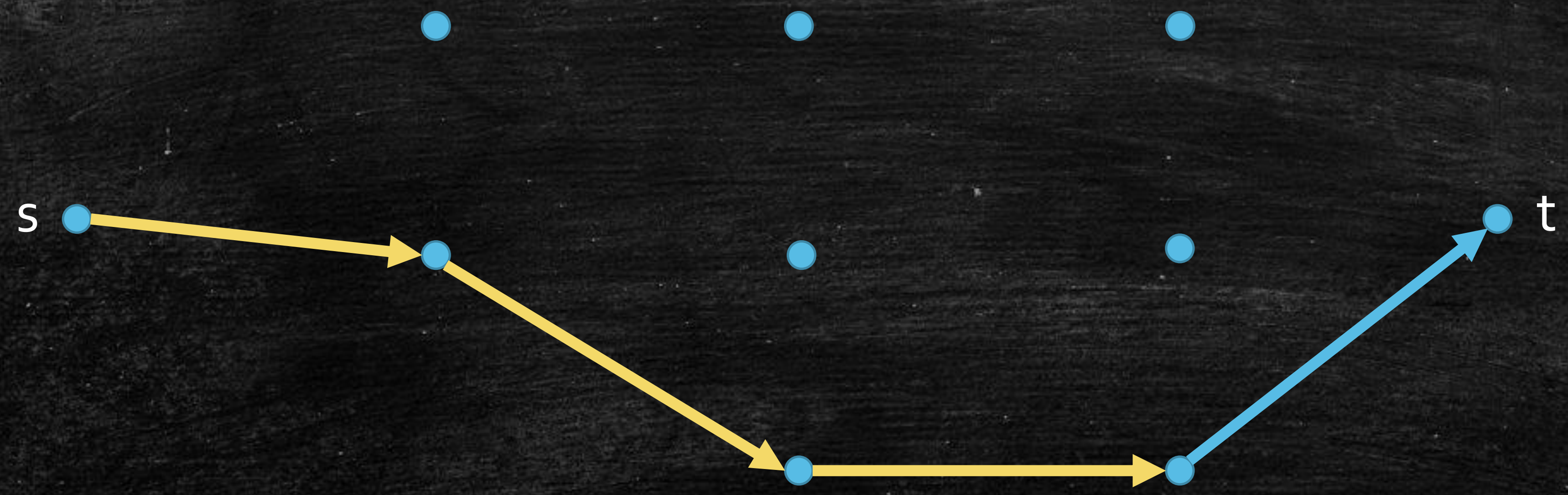
Again, we have to go backward; delete the edge just travelled.



Step 1: Finding a blocking flow in a level graph takes $O(|E|)$ time.

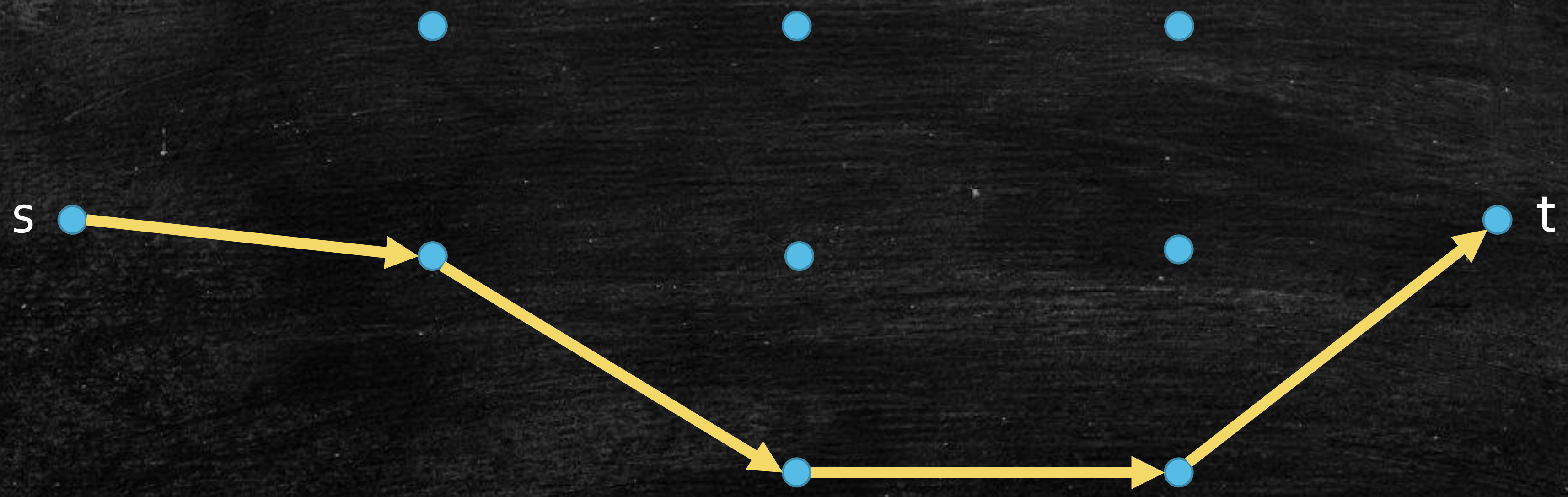


Step 1: Finding a blocking flow in a level graph takes $O(|E|)$ time.



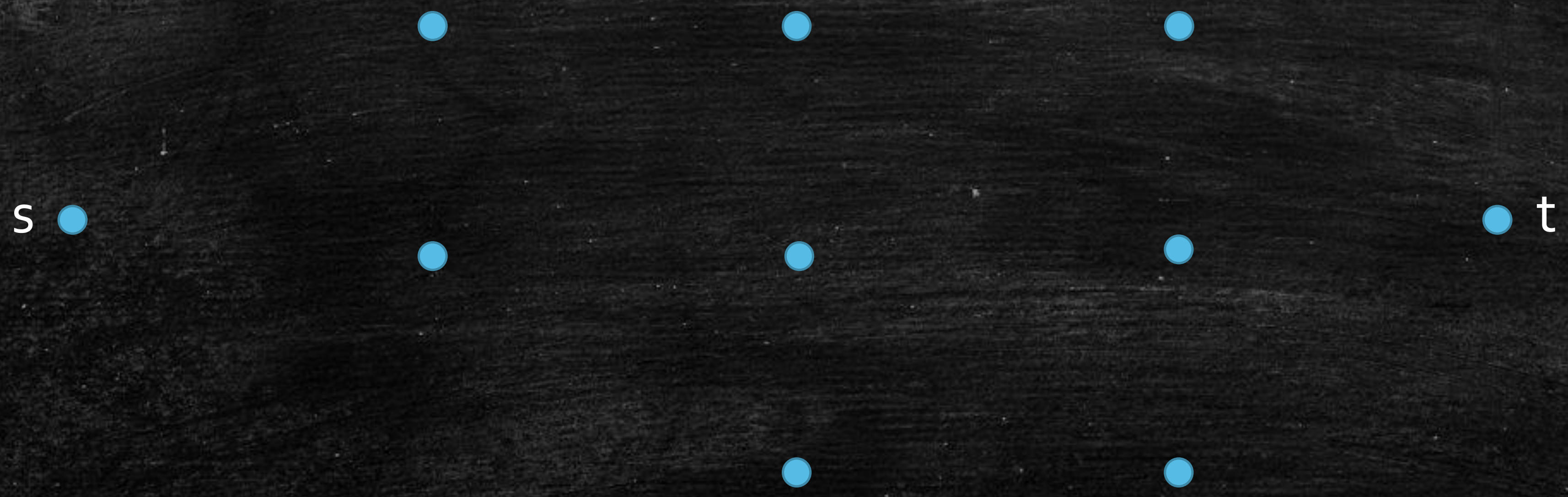
Step 1: Finding a blocking flow in a level graph takes $O(|E|)$ time.

Find another s-t path; delete all edges on the path



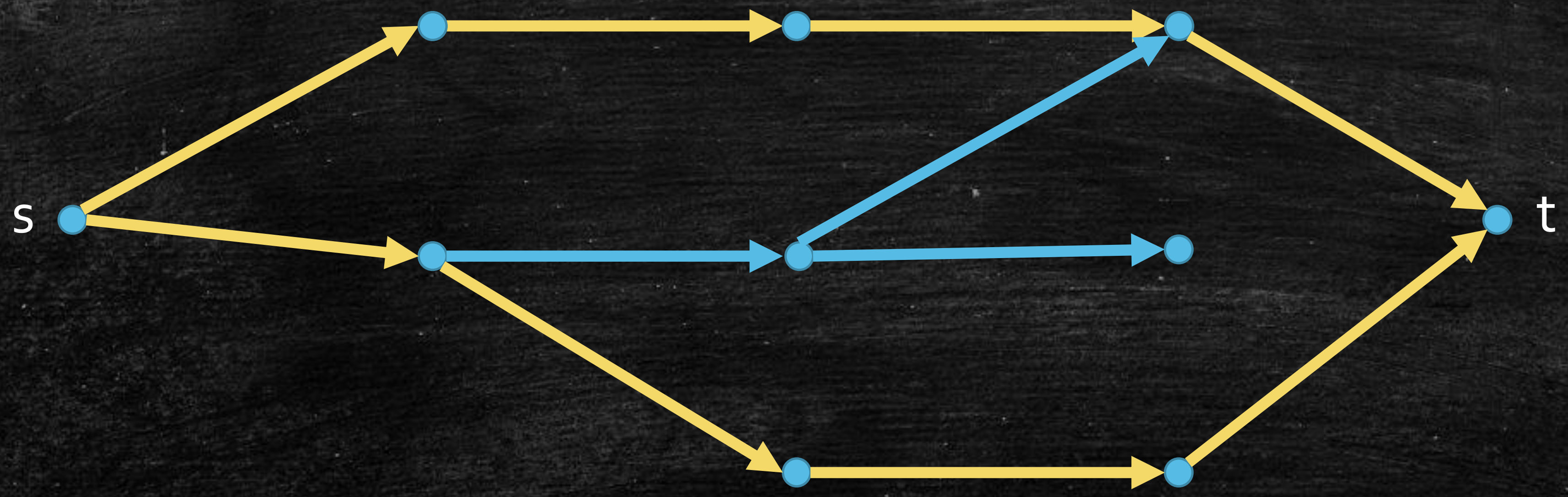
Step 1: Finding a blocking flow in a level graph takes $O(|E|)$ time.

We are done!



Step 1: Finding a blocking flow in a level graph takes $O(|E|)$ time.

We have obtained a blocking flow!



Step 1: Finding a blocking flow in a level graph takes $O(|E|)$ time.

Time complexity: $O(|E|)$

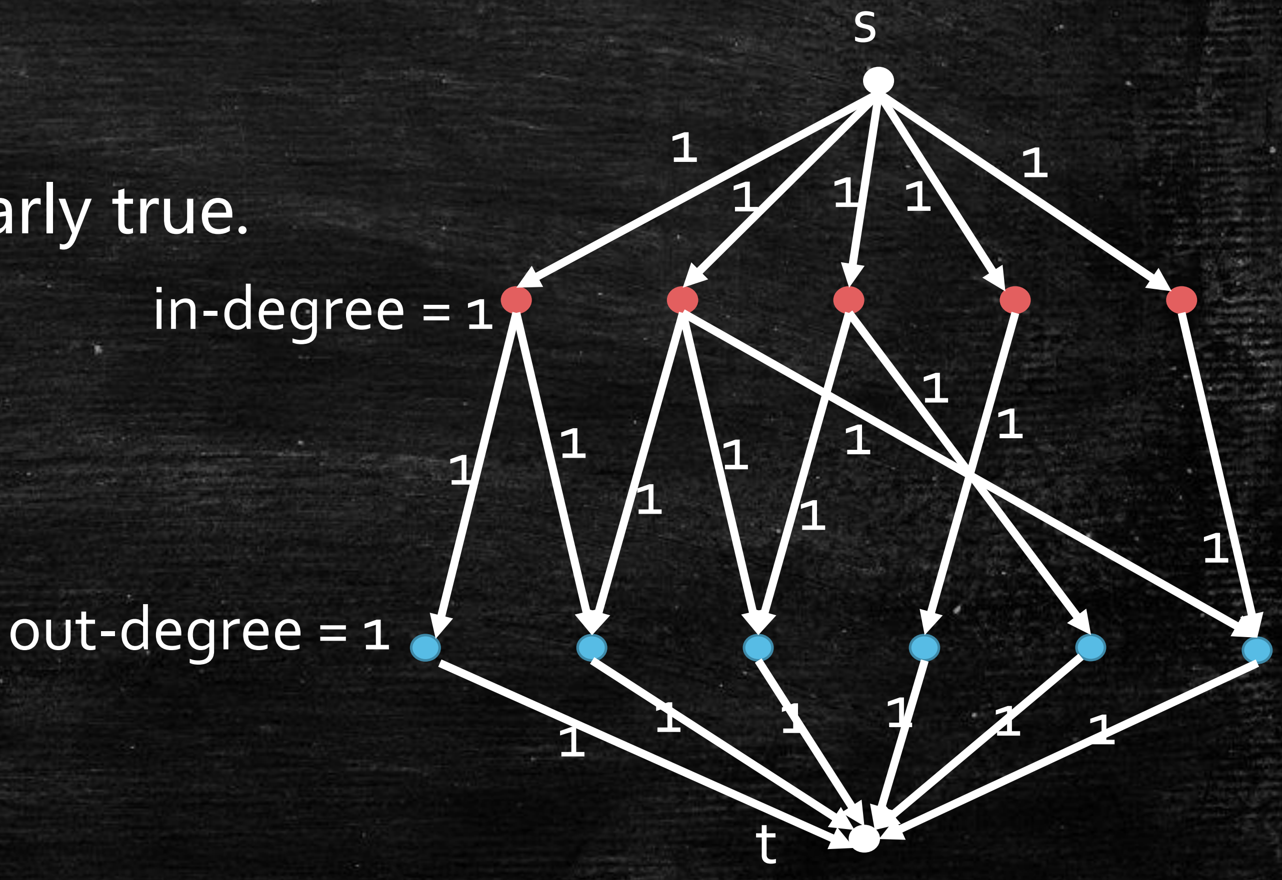
- Each edge is visited at most once.

Step 2: Number of iterations is at most $2\sqrt{|V|}$.

- If the algorithm terminates within $\sqrt{|V|}$ iterations, we are already done!
- Otherwise, let f be the flow after $\sqrt{|V|}$ iterations.
- Claim: the maximum flow in G^f has value at most $\sqrt{|V|}$.
- If the claim is true, we can stop after another $\sqrt{|V|}$ rounds.

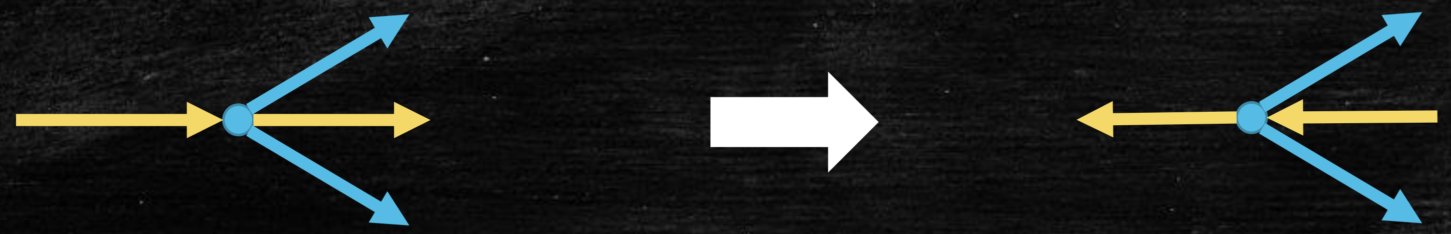
Observation on G^f

- In each iteration, for each $v \in V \setminus \{s, t\}$, either its in-degree is 1, or its out-degree is 1.
- Proof. By Induction…
- At the beginning, this is clearly true.

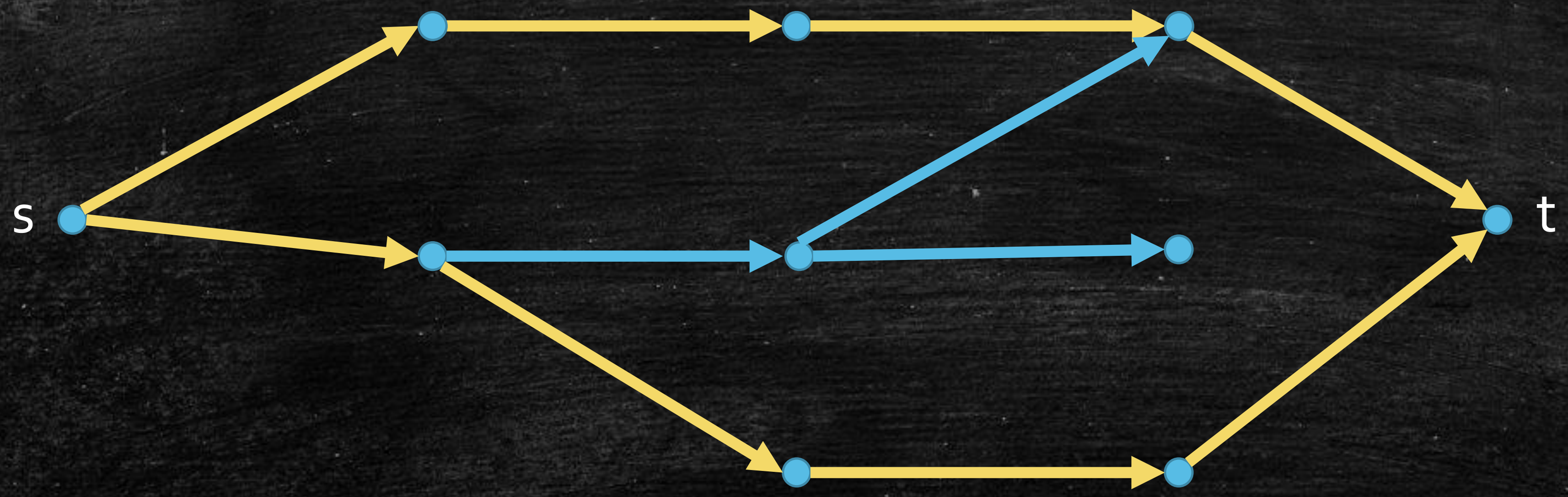


Observation on G^f

- In each iteration, for each $v \in V \setminus \{s, t\}$, either its in-degree is 1, or its out-degree is 1.
- Proof. At the beginning, this is clearly true.
- For each iteration, the amount of flow going through v is either 0 or 1.
- If it is 0, v's in-degree and out-degree are unchanged.
- Otherwise, exactly one in-edge and one out-edge are flipped; the property is still maintained.

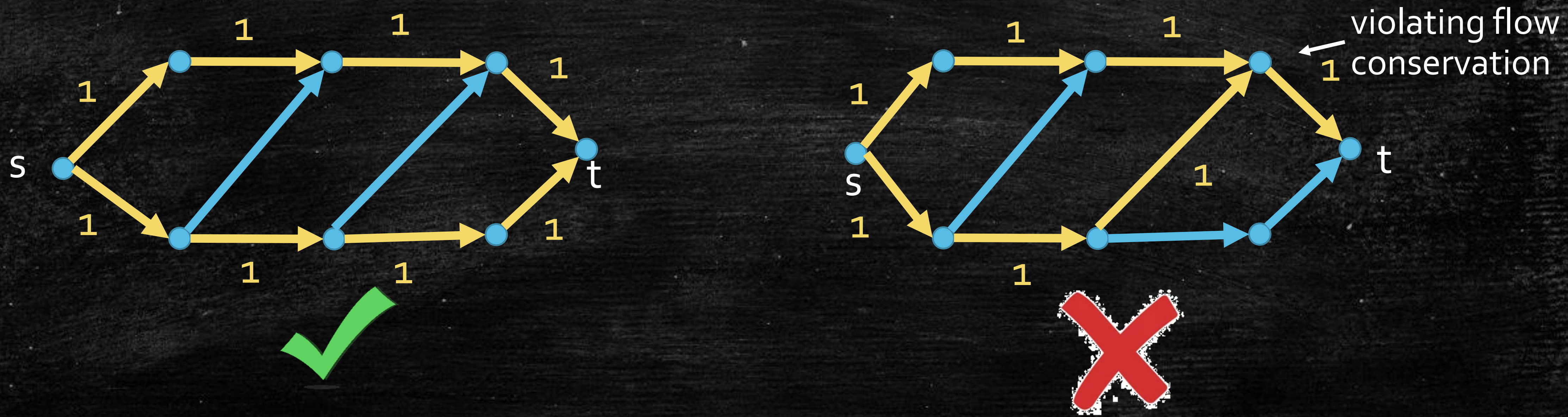


Let us check!



The maximum flow in G^f has value at most $\sqrt{|V|}$

- Integrality Theorem: there exists a maximum integral flow f' in G^f.
- f' consists of vertex-disjoint paths with flow 1!



The maximum flow in G^f has value at most $\sqrt{|V|}$

- Max-flow on G^f, f', is integral and consists of edge-disjoint paths.
- By our analysis to Dinic's algorithm, $\mathrm{dist}^{G^f}(s,t) \geq \sqrt{|V|}$.
- Each path in f' has length at least $\sqrt{|V|}$.
- There are at most $\frac{|V|}{\sqrt{|V|}} = \sqrt{|V|}$ paths in f' by vertex-disjointness.
- $v(f') \leq \sqrt{|V|}$

Step 2: Number of iterations is at most $2\sqrt{|V|}$.

- If the algorithm terminates within $\sqrt{|V|}$ iterations, we are already done!
- Otherwise, let f be the flow after $\sqrt{|V|}$ iterations.
- Claim: the maximum flow in G^f has value at most $\sqrt{|V|}$.
- Each iteration increase the value of flow by at least 1.
- Thus, the algorithm will terminate within at most another $\sqrt{|V|}$ iterations.
- Total number of iterations: $2\sqrt{|V|}$.

Putting Together...

- Step 1: Finding a blocking flow in a level graph takes $O(|E|)$ time.
- Step 2: Number of iterations is at most $2\sqrt{|V|}$.
- Overall time complexity: $O\left(|E| \cdot \sqrt{|V|}\right)$

Today's Lecture

Maximum Flow Problem:

- Edmonds-Karp Algorithm
  - Implement Ford-Fulkerson method by BFS
  - $O(|V| \cdot |E|^2)$
- Dinic's Algorithm
  - Push flow on multiple paths at one iteration
  - Level graph and blocking flow
  - $O(|V|^2 \cdot |E|)$

Maximum Bipartite Matching Problem:

- Hopcroft–Karp–Karzanov algorithm
  - Apply Dinic's algorithm
  - $O\left(|E| \cdot \sqrt{|V|}\right)$