Lecture 18

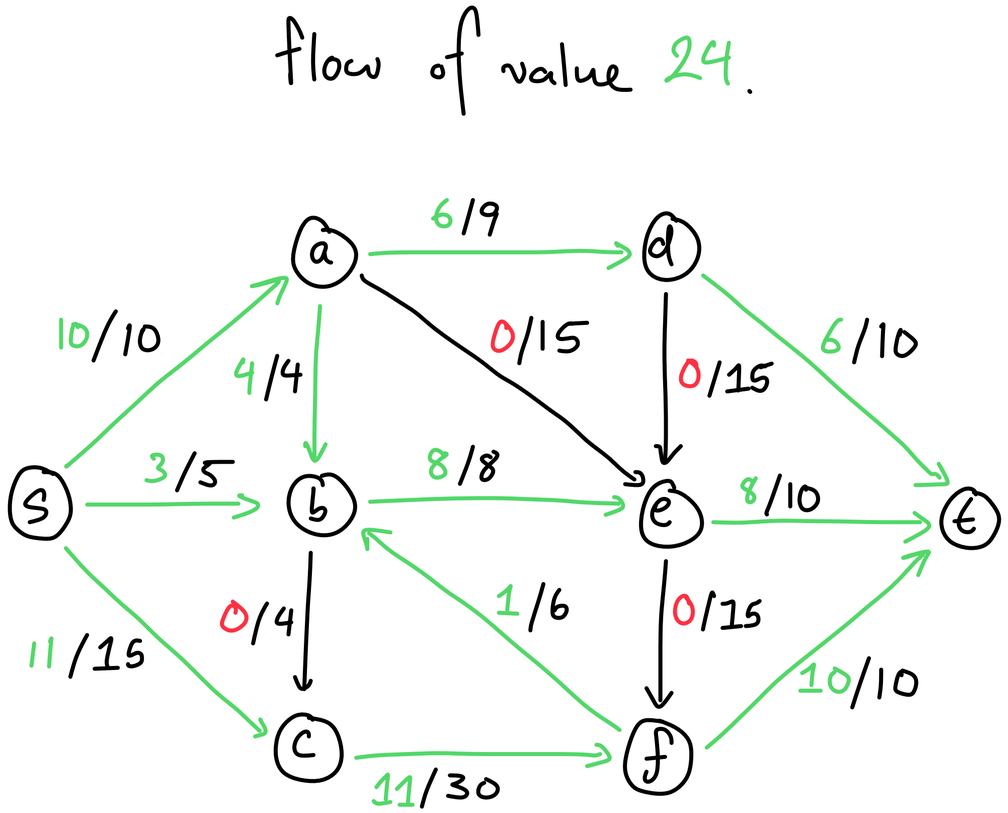
Efficient maximum flow and applications

Chinmay Nirkhe | CSE 421 Winter 2026

Previously in CSE 421…

The maximum flow problem

- **Input:** a flow network (G, c, s, t)
- **Output:** a s-t flow of maximum value



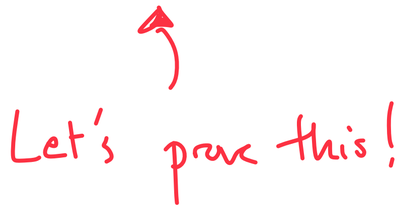
Ford-Fulkerson algorithm

- Initialize a flow of $f(e) \leftarrow 0$ for all edges. Set residual network $G_f \leftarrow G$
- While there is a simple path $p : s \rightsquigarrow t$ in G_f
  - Let f_{aug} be the flow along p of weight $\min_{e \in p} c_{G_f}(e)$
  - Augment $f \leftarrow f + f_{\text{aug}}$
  - Update G_f along the edges of p

Today

Ford-Fulkerson always finds a max flow

- **Theorem:** When capacities are positive integers, Ford-Fulkerson always terminates and outputs a max-flow.
- **Observation:** Ford-Fulkerson only terminates if there is no path $s \rightsquigarrow t$ in the residual graph G_f.
- Therefore, it suffices to show that a flow f is maximal iff there is no path $s \rightsquigarrow t$ in the residual graph G_f.



The max flow/min cut theorem

- **Max flow/min cut theorem:** Let f be a flow in a network (G, s, t, c). The following statements are equivalent!
  - (1) There exists a s-t cut (S, T) such that $v(f) = c(S, T)$.
  - (2) f is a max flow.
  - (3) There is no augmentation path $s \rightsquigarrow t$ in G_f.
- We will prove that (1) $\Longrightarrow$ (2), (2) $\Longrightarrow$ (3), and (3) $\Longrightarrow$ (1).

The max flow/min cut theorem

(1) $\Longrightarrow$ (2)

- (1) There exists a s-t cut (S, T) such that $v(f) = c(S, T)$.
- (2) f is a max flow.
- **Proof**:
  - We know that $v(f) \leq c(S, T)$ for any s-t cut [Weak duality].
  - So if $v(f) = c(S, T)$, then there cannot be any flow f' s.t. $v(f') > v(f)$.
  - So f must be maximal.

The max flow/min cut theorem

(2) $\Longrightarrow$ (3)

- (2) f is a max flow.
- (3) There is no augmentation path $s \rightsquigarrow t$ in G_f.

- **Proof:** By contrapositive.

The max flow/min cut theorem

¬ (3) $\Longrightarrow$ ¬ (2)

- ¬ (2) f is **not** a max flow.
- ¬ (3) There **is** a augmentation path $s \rightsquigarrow t$ in G_f.

- **Proof**:
  - Let f_{aug} be the augmentation path.
  - We saw last lecture that $f + f_{\text{aug}}$ is a flow in G. And $v(f + f_{\text{aug}}) > v(f)$.

The max flow/min cut theorem

(3) $\Longrightarrow$ (1)

- (3) There is no augmentation path $s \rightsquigarrow t$ in G_f.
- (1) There exists a s-t cut (S, T) such that $v(f) = c(S, T)$.

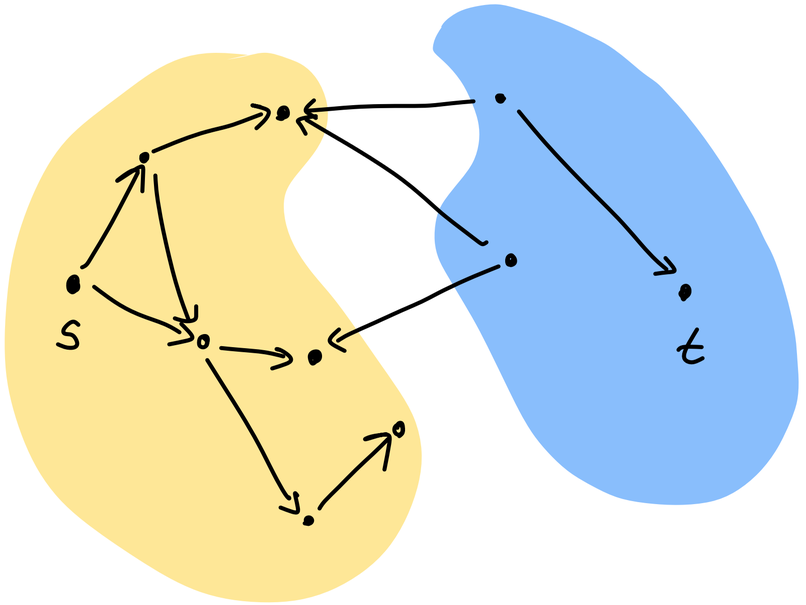
- **Proof:** This is a lengthy proof! It will take us a few slides. Key ideas:
  - We will need to find the s-t cut (S, T). It should be found using the flow f.
  - Then we will use that $v(f) = f^{\text{out}}(S) - f^{\text{in}}(S)$ to prove that $v(f) = c(S, T)$.

The max flow/min cut theorem

(3) $\Longrightarrow$ (1)

- **Proof**:
  - Let f be a flow such that there are no augmenting paths in G_f.
  - Let S be the set of vertices reachable from s.
    - Since there are no paths, $t \notin S$.
    - Let $T = V \backslash S$ and this defines a s-t cut.

residual graph G_f.

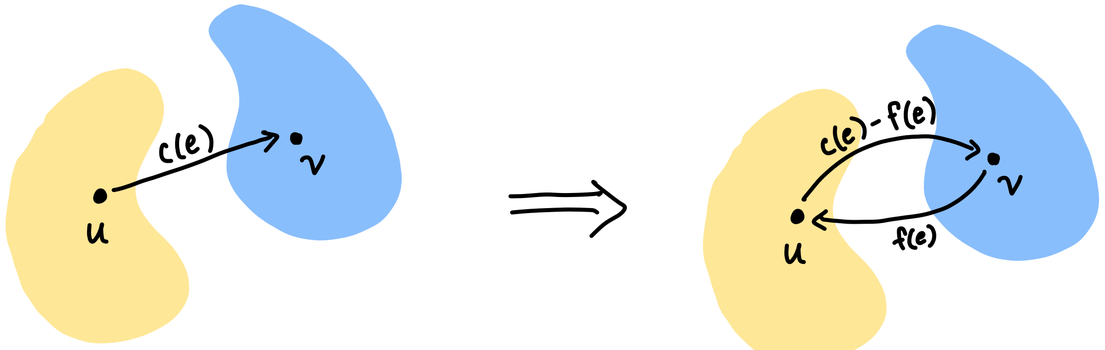


there are no edges S to T in the residual graph G_f.

The max flow/min cut theorem

(3) $\Longrightarrow$ (1)

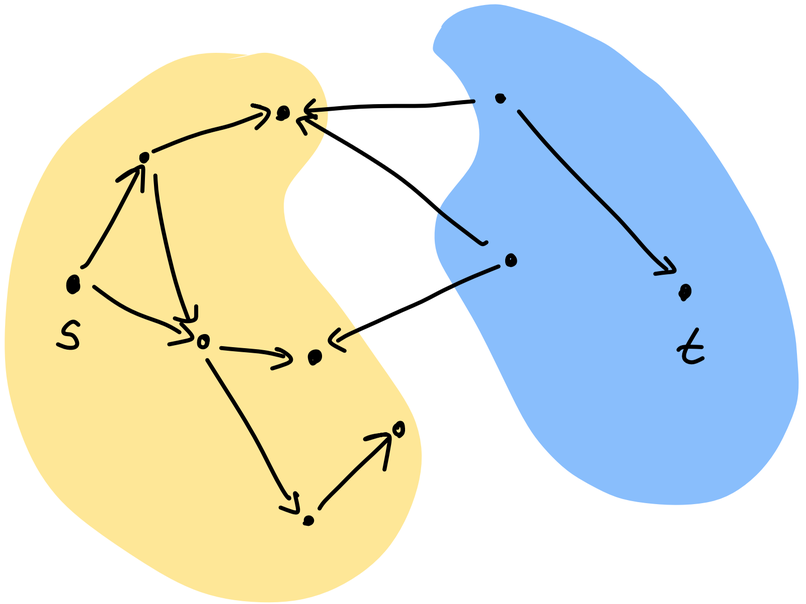
- **Proof**:
  - What does it mean for there to be no edges S to T in the residual graph G_f?
  - For any edge $e = (u \to v) \in G$ from S to T,



edge in the graph G

edges in residual graph G_f

residual graph G_f.

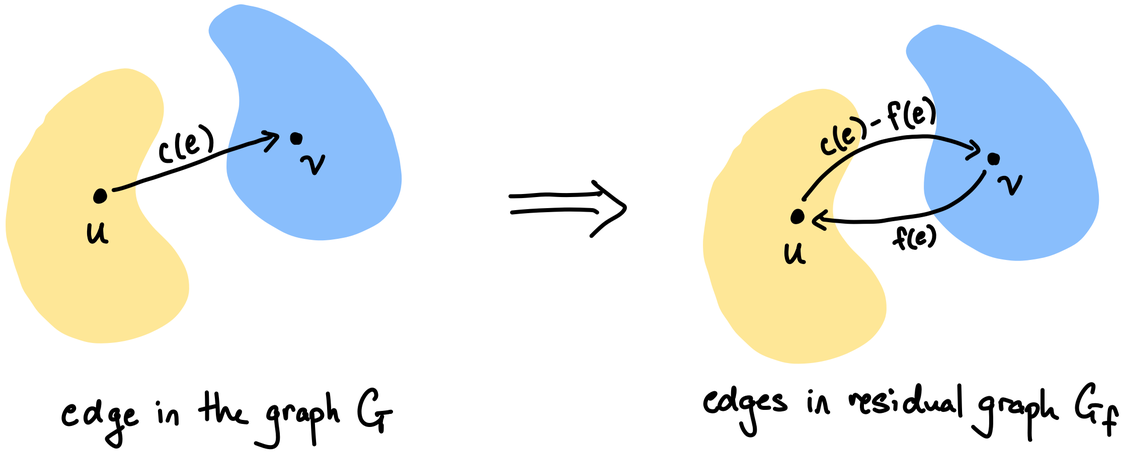


there are no edges S to T in the residual graph G_f.

The max flow/min cut theorem

(3) $\Longrightarrow$ (1)

- **Proof**:
  - What does it mean for there to be no edges S to T in the residual graph G_f?
  - For any edge $e = (u \to v) \in G$ from S to T,



edge in the graph G

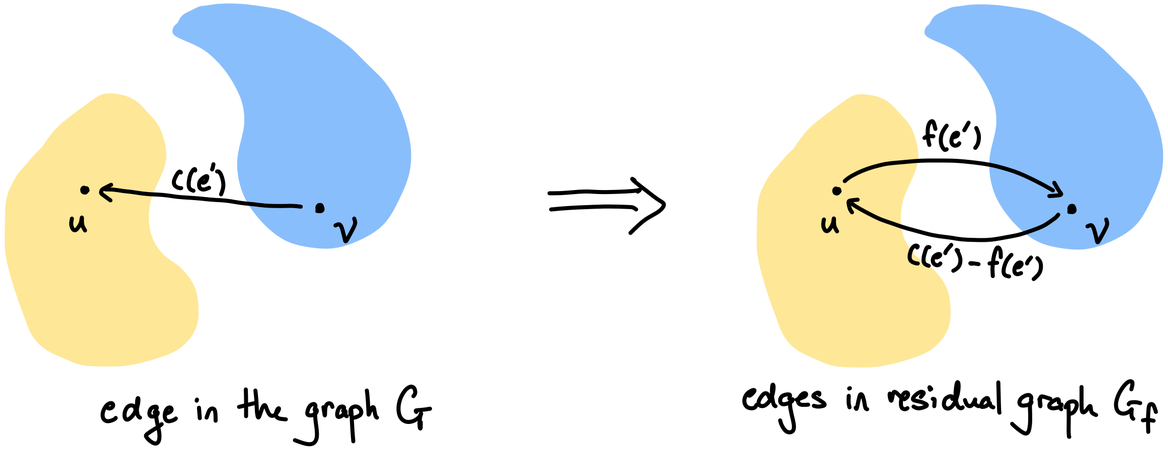
edges in residual graph G_f

Therefore, $c(e) = f(e)$ for all edges $u \to v$ from S to T.

The max flow/min cut theorem

(3) $\Longrightarrow$ (1)

- **Proof**:
  - What does it mean for there to be no edges S to T in the residual graph G_f?
  - For any edge $e' = (v \to u) \in G$ from T to S,



edge in the graph G

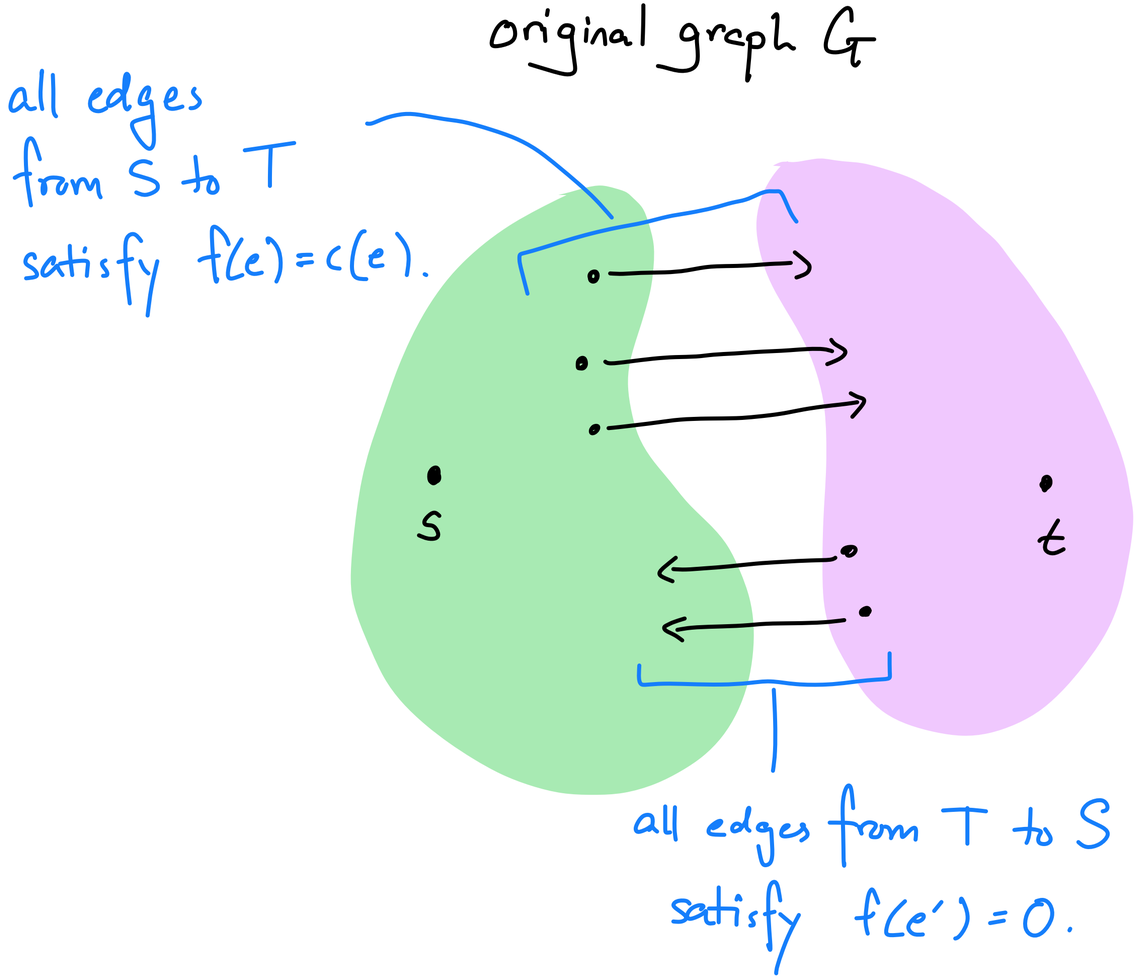
edges in residual graph G_f

Therefore, $f(e') = 0$ for all edges $v \to u$ from T to S.

The max flow/min cut theorem

(3) $\Longrightarrow$ (1)

- **Proof**:
  - Edges from S to T are *saturated* with flow.
  - Edges from T to S have no flow.
  - $v(f) = f^{\text{out}}(S) - f^{\text{in}}(S)$
  - $$= \sum_{e \text{ from } S \text{ to } T} f(e) - \sum_{e' \text{ from } T \text{ to } S} f(e')$$
  - $$= \underbrace{\sum_{e \text{ from } S \text{ to } T} c(e)}_{C(S,T)} - \underbrace{\sum_{e' \text{ from } T \text{ to } S} 0}_{0}$$
  - $$= C(S,T).$$



The max flow/min cut theorem

(3) $\Longrightarrow$ (1)

- (3) There is no augmentation path $s \rightsquigarrow t$ in G_f.
- (1) There exists a s-t cut (S, T) such that $v(f) = c(S, T)$.

- **Proof:** This is a lengthy proof! It will take us a few slides. Key ideas:
  - We will need to find the s-t cut (S, T). It should be based on the aug. path.
  - Then we will use that $v(f) = f^{\text{out}}(S) - f^{\text{in}}(S)$ to prove that $v(f) = c(S, T)$.

The max flow/min cut theorem

- **Max flow/min cut theorem:** Let f be a flow in a network (G, c, s, t). The following statements are equivalent!
  - (1) There exists a s-t cut (S, T) such that $v(f) = c(S, T)$.
  - (2) f is a max flow.
  - (3) There is no augmentation path $s \rightsquigarrow t$ in G_f.
- **Corollary:** The value of the max flow equals the value of the min cut!

Returning to Ford-Fulkerson

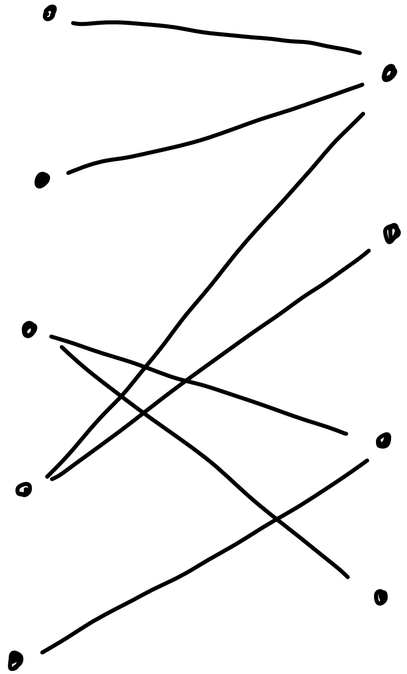
- Ford-Fulkerson is a greedy algorithm which calculates the max flow by incrementally increasing the flow.
- Max flow/min cut theorem proves that Ford-Fulkerson only terminates when the max flow is achieved.
- If the capacities are integer, Ford-Fulkerson will increase the flow by at least 1 per iteration.
- Yields a runtime of $O(mC)$ where C is the sum of capacities of edges leaving s.
- Runtime can be exponential time in input length for large C as capacities are expressed in binary in the input.
- But when $C = \text{poly}(n)$, then algorithm can be very efficient.

Integral max flow

- **Theorem**: Consider a graph network (G, s, t, c) where $c : E \to \mathbb{Z}_{\geq 0}$. Then, there exists a max flow which assigns an integer flow to every edge.
- **Proof**:
  - Ford-Fulkerson will calculate the max flow.
  - Ford-Fulkerson only increases the flow by integer quantities starting from 0.
  - Therefore, there exists a max flow that has integer flow.

Application: Bipartite matching

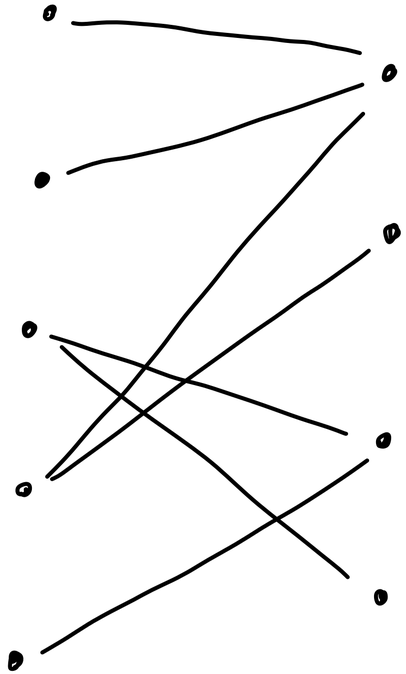
- **Input**: A bipartite graph $(V = L \sqcup R, E)$
- **Output**: A maximal collection of edges that don't share any vertices.



- We will see that there is an algorithm based on Ford-Fulkerson.

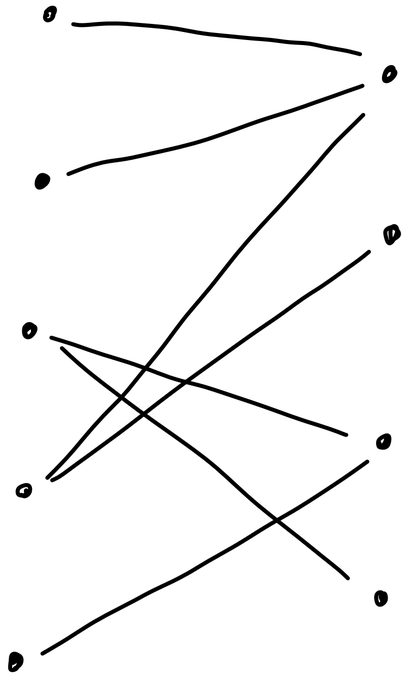
Application: Bipartite matching

- **Input**: A bipartite graph $(V = L \sqcup R, E)$
- **Output**: A maximal collection of edges that don't share any vertices.

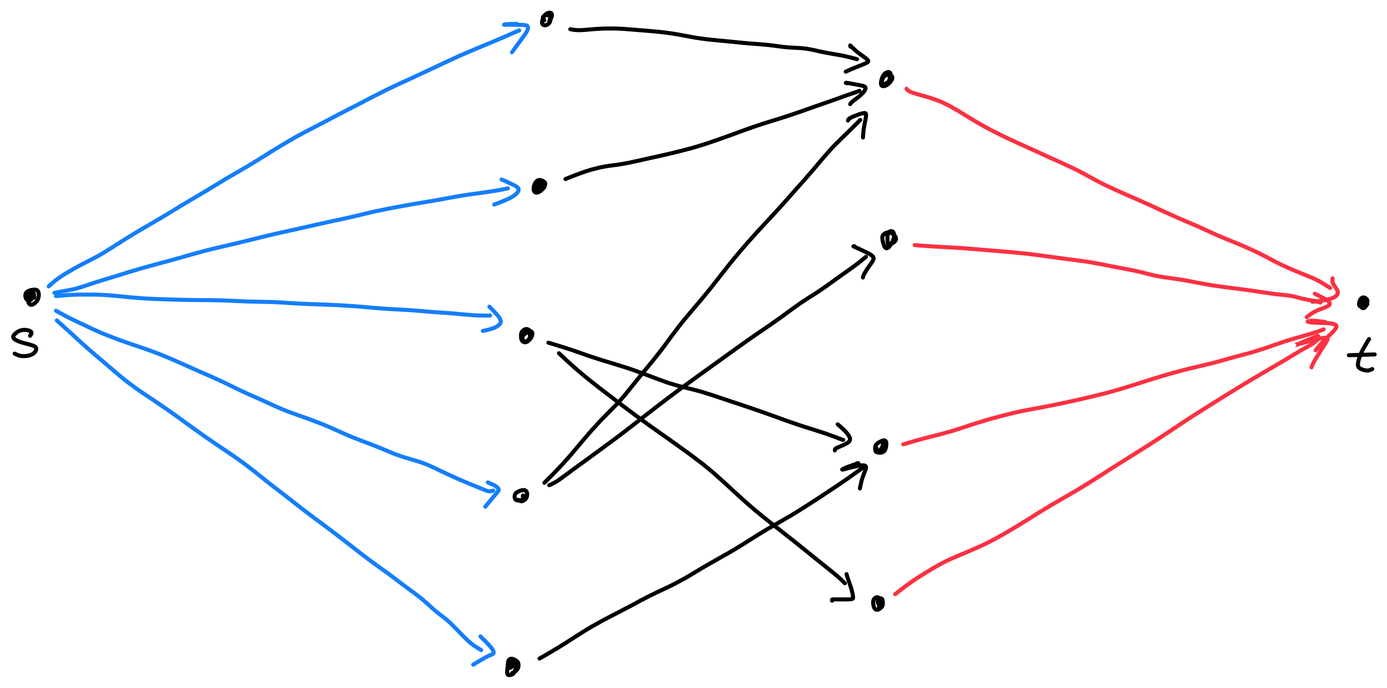


- To solve with Ford-Fulkerson, we need to create a directed graph and identify a source s and sink t.

Application: Bipartite matching

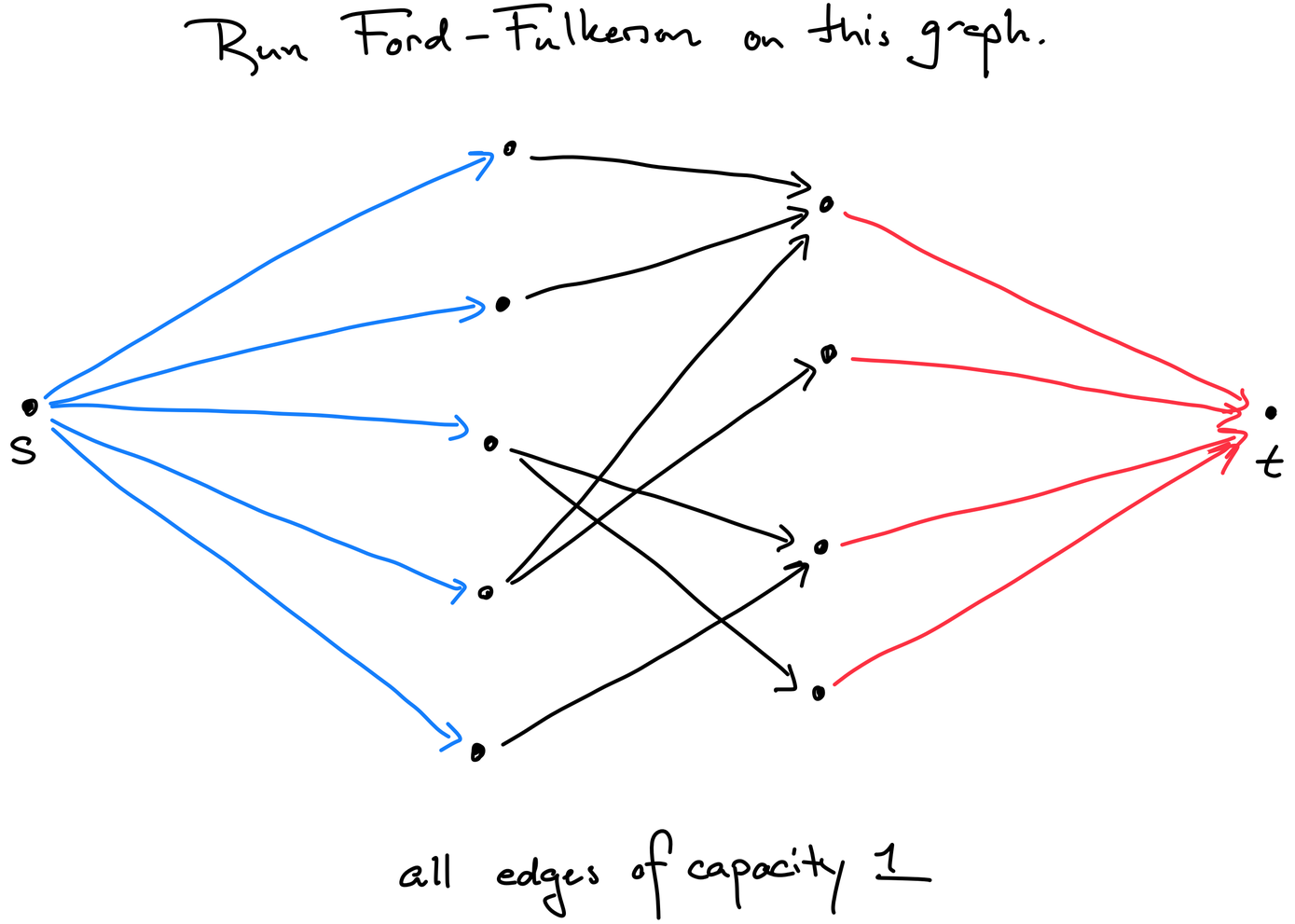


Application: Bipartite matching

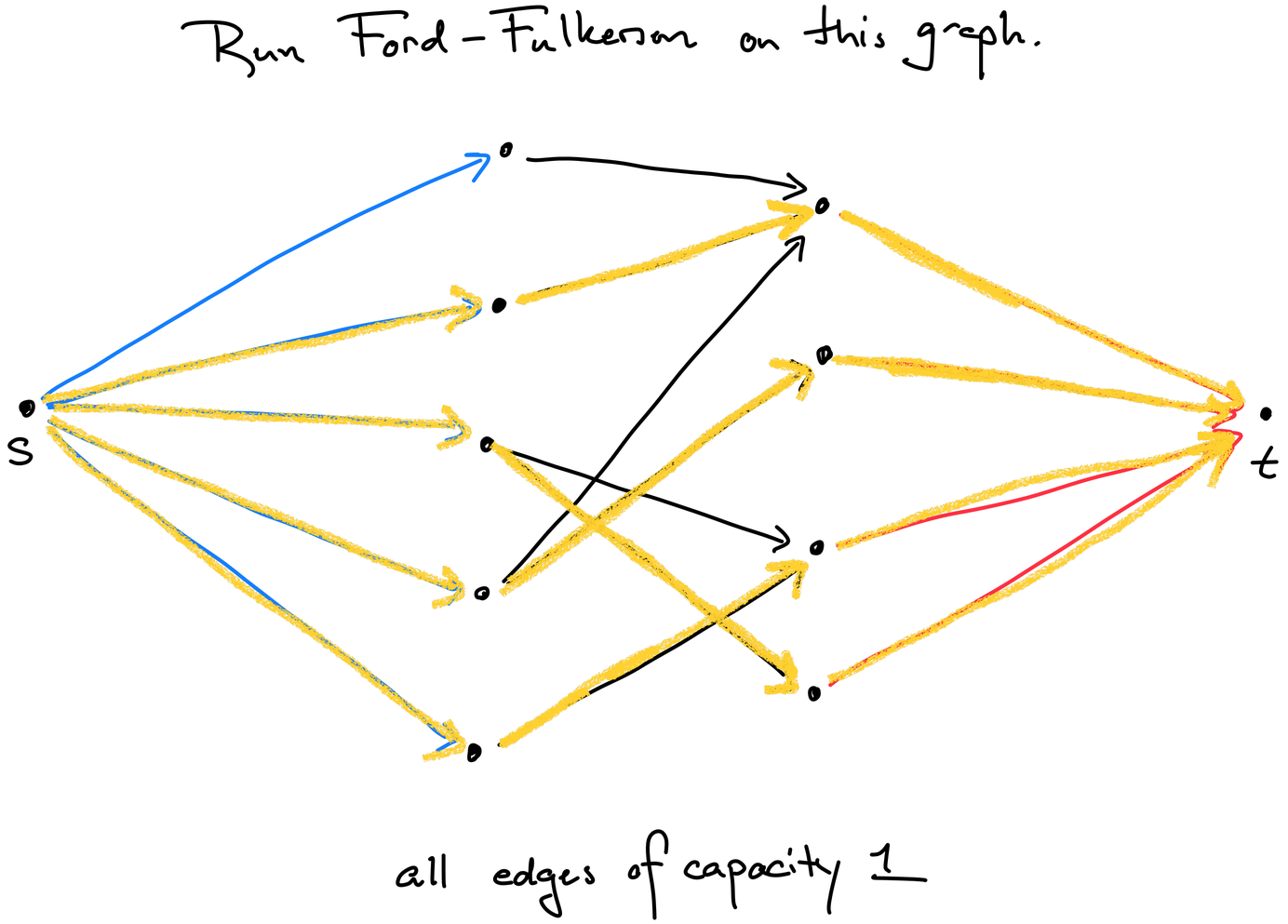


all edges of capacity 1

Application: Bipartite matching



Application: Bipartite matching



Application: Bipartite matching

- **Claim**: The edges of flow 1 in the max flow form a maximal bipartite matching.
- **Proof**:
  - Integer flow and bipartite matching *bijection*:
    - Since FF only outputs integer flow, and each edge capacity is 1, at most 1 edge leaving a $v \in L$ can be selected. So a integer flow yields a matching of equal size.
    - For every edge $u \to v$ from L to R in the bipartite matching add the flow $s \to u \to v \to t$. All flows will be compatible. So a bipartite matching yields a flow of equal size.
  - By bijection, max flow will yield a max bipartite matching.

Application: Bipartite matching

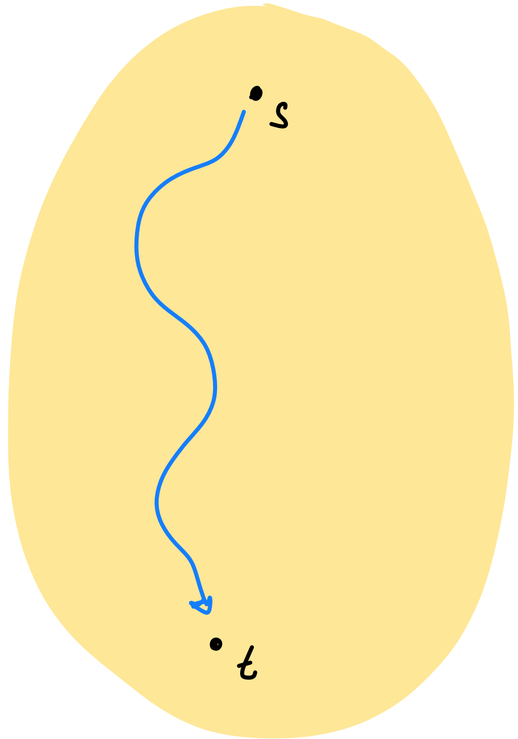
- **Runtime:** Each edge has capacity 1, root node has total output capacity n.
  - $C = n$, number of edges in network is $m + 2n$.
  - Total runtime after reduction, $O((2n + m)n) = O(n^2 + mn)$.
- **Aside:** Best known algorithm is called Hopcroft-Karp
  - It is also a flow-based algorithm but uses tricks to add $\Omega(\sqrt{n})$ flows simultaneously
  - Gives a runtime of $O(m\sqrt{n})$.

Ford-Fulkerson can be slow

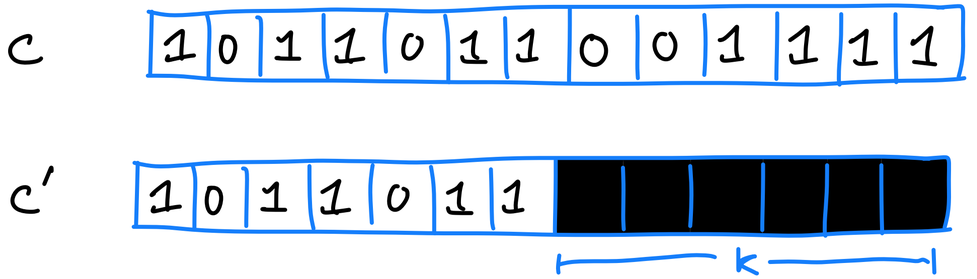
- **Input**: The input is a flow network (G, s, t, c)
  - Formally, $c = \{c(e_1), c(e_2), \ldots, c(e_m)\}$ for each edge e_j with $c(e_j)$ being a number expressed in binary.
  - Then $C = C(\{s\}, V \backslash \{s\})$ is an exponential number in the size of the input.
  - Ford-Fulkerson can be slow! Runtime of $O(mC)$.
    - Because each update only guarantees flow increase by 1.
    - Is there a fast way to find bigger increases in flow?

Finding an augmenting path

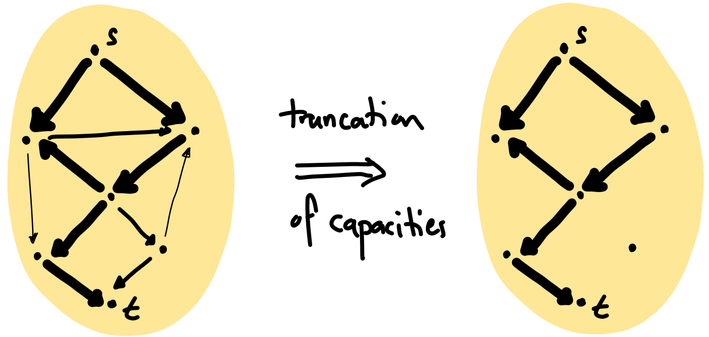
- We previously chose an augmenting path $s \rightsquigarrow t$ in G_f by running a graph traversal from s to t and picking a path
- This will find an augmenting path but may fail to find the augmenting path of largest bottleneck capacity
- **Idea:** If there exists some augmenting path of bottleneck capacity $\geq 2^k$, can we construct an algorithm that finds an augmenting path of bottleneck capacity at least 2^k?



Finding a pretty big augmenting path



- **Fast (Scaling) Augment:** Starting with $k \leftarrow \lfloor \log C \rfloor$,
  - Find an augmenting path of size 2^k:
    - Run regular augmenting path search on G_f except with capacities $c' = \lfloor c/2^k \rfloor$.
    - If a path exists of bottleneck $\geq 2^k$, it still exists in adjusted graph.
  - If yes, add this augmenting path and restart.
  - If not, decrease $k \leftarrow k - 1$, and repeat.



- **Theorem:** If the max bottleneck capacity of any augmenting path is v, the fast augment subroutine finds an augment of size $\geq v/2$ in time $O(m \log C)$.

Scaling Ford-Fulkerson

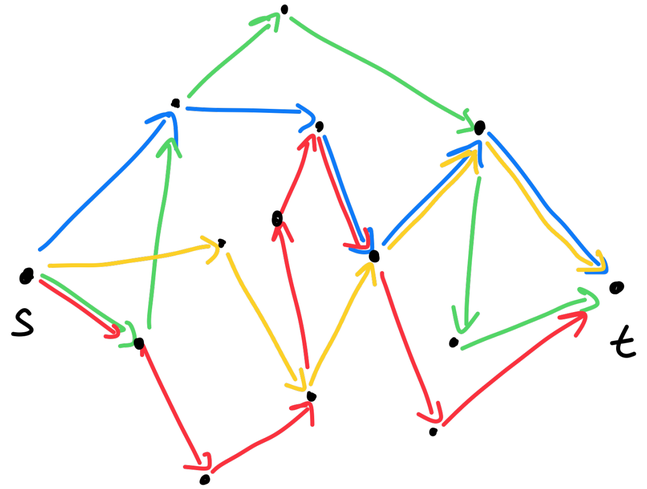
- **Algorithm:** Start with flow $f \leftarrow 0$ and $G_f \leftarrow G$.
  - While the fast augment subroutine can find an augmenting path p
    - Augment f by f_{aug} along path and update G_f

- **Theorem:** The scaling version of Ford-Fulkerson runs in time $O(m^2 \log C)$.

Scaling Ford-Fulkerson runtime

- To prove the runtime of $O(m^2 \log C)$, we need to prove a few lemmas.

- **Lemma**: Every flow f can be expressed as the sum of $\leq m$ flows along paths.
- **Proof**:
  - While there exists a path $p : s \rightsquigarrow t$ in the flow,
    - Remove flow along p of the bottleneck capacity of p.
    - The resulting flow is 0 along some edge.
  - This can be repeated $\leq m$ times.



Scaling Ford-Fulkerson runtime

- To prove the runtime of $O(m^2 \log C)$, we need to prove a few lemmas.

- **Lemma**: Every flow f can be expressed as the sum of $\leq m$ flows along paths.
- **Corollary**: There exists a path within the flow of bottleneck capacity $\geq \text{maxflow}(G)/m$.
- **Proof**:
  - Run the lemma on the max flow.
  - By pigeon-hole principle, one of the paths must have large flow.

Scaling Ford-Fulkerson runtime

- To prove the runtime of $O(m^2 \log C)$, we need to prove a few lemmas.

- **Lemma**: Every flow f can be expressed as the sum of $\leq m$ flows along paths.
- **Corollary**: There exists a path within the flow of bottleneck capacity $\geq \text{maxflow}(G)/m$.
- **Corollary**: Fast-Augment will find an augmenting path in G_f of bottleneck capacity $\geq \text{maxflow}(G_f)/(2m)$.

Scaling Ford-Fulkerson runtime

- **Corollary:** Fast-Augment will find an augmenting path in G_f of bottleneck capacity $\geq \text{maxflow}(G_f)/(2m)$.

- Each iteration of Fast-Augment will decrease by a mult. factor of $1 - 1/(2m)$
- # of iterations $\leq \log_{(1-1/(2m))^{-1}}(C) = \dfrac{\log C}{-\log(1 - 1/(2m))} \leq \dfrac{\log C}{1/(2m)} = 2m \log C.$
- Total runtime is $O(m) \cdot 2m \log C = O(m^2 \log C)$.

Flow independent of capacity

- So far, for integer capacities:
  - **Vanilla Ford-Fulkerson:** Runtime $O(mC)$
    - Pick any augmenting path
  - **Scaling Ford-Fulkerson:** Runtime $O(m^2 \log C)$
    - Pick the largest augmenting paths
  - **Edmonds-Karp (next):** Runtime $O(m^2 n)$
    - Pick the shortest augmenting path (in terms of # of edges)