COL 351 : Analysis & Design of Algorithms

LECTURE 32

NETWORK FLOW IV :
EDMONDS - KARP ALGORITHM & APPLICATIONS

OCT 27, 2026 | ROHIT VAISH

STORY SO FAR

* Max flow $\iff$ tight cut exists $\iff$ s,t disconnected in G_f

* Min cut reduces to maximum flow

* Ford-Fulkerson: Correct but only a finite-time bound

* Today: (1) Edmonds-Karp: Ford-Fulkerson + shortest (s,t) paths

  (2) Applications of max flow

FORD - FULKERSON ALGORITHM

① initialize $f_e = 0 \quad \forall e$

② repeat:

* search for an $s-t$ path P in G_f
  such that every edge in P has positive residual capacity

// possible in $O(|E|)$ time via BFS / DFS

* if no such path, return current flow $\{f_e\}_{e \in E}$
* else, let $\Delta := \min_{e \in P}$ e's residual capacity in G_f

  for all $e \in G$ with forward edges in P, add Δ to f_e.
  for all $e \in G$ with reverse edges in P, subtract Δ from f_e.

TERMINATION OF FORD-FULKERSON

For integral capacities,

Ford-Fulkerson terminates in finite time with a valid flow.

TERMINATION OF FORD-FULKERSON

For integral capacities,

Ford-Fulkerson terminates in finite time with a valid flow.

Does the algorithm return a max flow?

TERMINATION OF FORD-FULKERSON

For integral capacities,

Ford-Fulkerson terminates in finite time with a valid flow.

Does the algorithm return a max flow?

🤔 HOW DO WE KNOW WHEN WE ARE DONE?

OPTIMALITY CONDITIONS FOR MAX FLOW

Theorem: For a flow f in a graph G, the following are equivalent:

(a) f is a maximum flow of G

(b) there exists an (s,t) - cut (A,B) such that

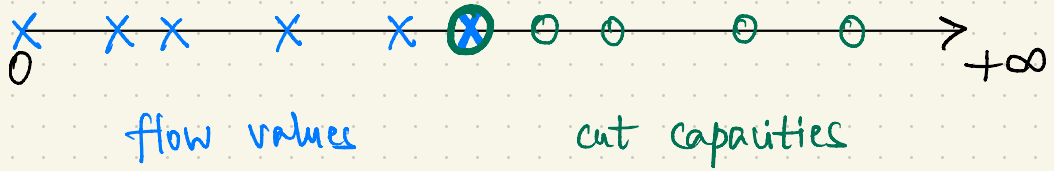
value of f = capacity of (A,B).

(c) there is no (s,t) path in the residual graph G_f.

MAX-FLOW MIN-CUT THEOREM

Corollary : For every network,

maximum value of a flow = minimum capacity of an (s,t)-cut



BACK TO FORD-FULKERSON

BACK TO FORD-FULKERSON

* Recall : f is max flow $\Leftrightarrow$ no (s,t)-path in G_f

$\Rightarrow$ Ford-Fulkerson is correct

BACK TO FORD-FULKERSON

* Recall: f is max flow $\iff$ no (s,t)-path in G_f

$\Rightarrow$ Ford-Fulkerson is correct

* Running time of Ford-Fulkerson: Pseudopolynomial (Tutorial 11)

runtime depends on U_e (not $\log U_e$)

BACK TO FORD-FULKERSON

* Recall : f is max flow $\iff$ no (s,t)-path in G_f

$\Rightarrow$ Ford-Fulkerson is correct

* Running time of Ford-Fulkerson : Pseudopolynomial (Tutorial 11)

runtime depends on U_e (not $\log U_e$)

Keeps choosing "essentially useless" paths over and over

BACK TO FORD-FULKERSON

* Recall : f is max flow $\iff$ no (s,t)-path in G_f

$\Rightarrow$ Ford-Fulkerson is correct

* Running time of Ford-Fulkerson : Pseudopolynomial (Tutorial 11)

runtime depends on U_e (not $\log U_e$)

Keeps choosing "essentially useless" paths over and over

🤔 Choose $s-t$ paths intelligently

BACK TO FORD-FULKERSON

* Recall : f is max flow $\iff$ no (s,t)-path in G_f

$\Rightarrow$ Ford-Fulkerson is correct

* Running time of Ford-Fulkerson : Pseudopolynomial (Tutorial 11)

runtime depends on U_e (not $\log U_e$)

* Fix : Edmonds-Karp algorithm

EDMONDS-KARP ALGORITHM

EDMONDS-KARP ALGORITHM

Refinement of Ford-Fulkerson

In G_f, pick an s-t path with fewest number of edges.

EDMONDS-KARP ALGORITHM

Ford-Fulkerson

① initialize $f_e = 0 \quad \forall e$

② repeat:

* search for an $s-t$ path P in G_f
  such that every edge in P has positive residual capacity

// possible in $O(|E|)$ time via BFS / DFS

* if no such path, return current flow $\{f_e\}_{e \in E}$
* else, let $\Delta := \min_{e \in P}$ e's residual capacity in G_f
  - for all $e \in G$ with forward edges in P, add Δ to f_e.
  - for all $e \in G$ with reverse edges in P, subtract Δ from f_e.

EDMONDS-KARP ALGORITHM

① initialize $f_e = 0 \quad \forall e$

② repeat:

* search for an s–t path P in G_f such that every edge in P has positive residual capacity

// possible in $O(|E|)$ time via BFS / DFS

* if no such path, return current flow $\{f_e\}_{e \in E}$
* else, let $\Delta := \min_{e \in P}$ e's residual capacity in G_f

  for all $e \in G$ with forward edges in P, add Δ to f_e.

  for all $e \in G$ with reverse edges in P, subtract Δ from f_e.

EDMONDS-KARP ALGORITHM

① initialize $f_e = 0 \quad \forall e$

② repeat:

* search for an $s-t$ path P in G_f with the fewest number of edges such that every edge in P has positive residual capacity

// possible in $O(|E|)$ time via BFS / DFS

* if no such path, return current flow $\{f_e\}_{e \in E}$
* else, let $\Delta := \min_{e \in P}$ e's residual capacity in G_f

  for all $e \in G$ with forward edges in P, add Δ to f_e.

  for all $e \in G$ with reverse edges in P, subtract Δ from f_e.

EDMONDS-KARP ALGORITHM

① initialize $f_e = 0 \quad \forall e$

② repeat:

* search for an $s-t$ path P in G_f with fewest no. of edges such that every edge in P has positive residual capacity

// possible in $O(|E|)$ time via BFS / DFS

* if no such path, return current flow $\{f_e\}_{e \in E}$
* else, let $\Delta := \min_{e \in P}$ e's residual capacity in G_f

  for all $e \in G$ with forward edges in P, add Δ to f_e.

  for all $e \in G$ with reverse edges in P, subtract Δ from f_e.

EDMONDS-KARP ALGORITHM

* Correct (special case of Ford-Fulkerson)

EDMONDS-KARP ALGORITHM

* Correct (special case of Ford-Fulkurson)

Theorem: The Edmonds-Karp algorithm runs in $O(m^2 n)$ time, where $m = |E|$ and $n = |V|$.

EDMONDS-KARP ALGORITHM

* Correct (special case of Ford-Fulkurson)

Theorem: The Edmonds-Karp algorithm runs in $O(m^2 n)$ time, where $m = |E|$ and $n = |V|$.

* Polynomial-time for any input capacities $\{u_e\}$.

EDMONDS-KARP ALGORITHM

* Correct (special case of Ford-Fulkerson)

Theorem: The Edmonds-Karp algorithm runs in $O(m^2 n)$ time, where $m = |E|$ and $n = |V|$.

* Polynomial-time for any input capacities $\{u_e\}$.

* $O(n^3)$ for sparse graphs, $O(n^5)$ for dense graphs

Running Time Analysis of Edmonds-Karp

Running Time Analysis of Edmonds-Karp

Recall:

f is max flow $\iff$ no (s,t) path in G_f

Running Time Analysis of Edmonds-Karp

Recall:

f is max flow $\iff$ no (s,t) path in G_f

want to disconnect s and t in G_f

Running Time Analysis of Edmonds-Karp

Recall:

f is max flow $\iff$ no (s,t) path in G_f

want to disconnect s and t in G_f



Measure progress in terms of how far s and t are in G_f

Running Time Analysis of Edmonds-Karp

Recall:

f is max flow $\iff$ no (s,t) path in G_f

want to disconnect s and t in G_f



Measure progress in terms of how far s and t are in G_f

number of edges on the minimum-edge path

RUNNING TIME ANALYSIS OF EDMONDS-KARP

Recall:

f is max flow $\iff$ no (s,t) path in G_f

want to disconnect s and t in G_f



Measure progress in terms of how far s and t are in G_f

NOTE: An (s,t) path can have at most $(n-1)$ edges.

Running Time Analysis of Edmonds-Karp

Key lemma : Fix a network G.

Running Time Analysis of Edmonds-Karp

Key lemma: Fix a network G. For a flow f, define:

$$d(f) = \left[\right.$$

Running Time Analysis of Edmonds-Karp

Key lemma: Fix a network G. For a flow f, define:

$$d(f) = \begin{cases} \text{the number of edges in a shortest } s\text{-}t \text{ path in } G_f \\ \text{with positive residual capacity} \end{cases}$$

Running Time Analysis of Edmonds-Karp

Key lemma: Fix a network G. For a flow f, define:

$$d(f) = \begin{cases} \text{the number of edges in a shortest } s\text{-}t \text{ path in } G_f \\ \text{with positive residual capacity} \\ +\infty \quad \text{if no such path exists.} \end{cases}$$

RUNNING TIME ANALYSIS OF EDMONDS-KARP

Key lemma: Fix a network G. For a flow f, define:

$$d(f) = \begin{cases} \text{the number of edges in a shortest } s\text{-}t \text{ path in } G_f \\ \text{with positive residual capacity} \\ +\infty \quad \text{if no such path exists.} \end{cases}$$

Then,

(a) $d(f)$ never decreases during the execution of the algorithm

RUNNING TIME ANALYSIS OF EDMONDS-KARP

Key lemma: Fix a network G. For a flow f, define:

$$d(f) = \begin{cases} \text{the number of edges in a shortest } s\text{-}t \text{ path in } G_f \\ \text{with positive residual capacity} \\ +\infty \quad \text{if no such path exists.} \end{cases}$$

Then,

(a) $d(f)$ never decreases during the execution of the algorithm

(b) $d(f)$ increases at least once per m iterations.

Running Time Analysis of Edmonds-Karp

Key lemma: Fix a network G. For a flow f, define:

$$d(f) = \begin{cases} \text{the number of edges in a shortest } s\text{-}t \text{ path in } G_f \\ \text{with positive residual capacity} \\ +\infty \quad \text{if no such path exists.} \end{cases}$$

Then,

(a) $d(f)$ never decreases during the execution of the algorithm

(b) $d(f)$ increases at least once per m iterations.

G fixed, f changes $\Rightarrow$ G_f changes

Running Time Analysis of Edmonds-Karp

Key lemma: Fix a network G. For a flow f, define:

$$d(f) = \begin{cases} \text{the number of edges in a shortest } s\text{-}t \text{ path in } G_f \\ \text{with positive residual capacity} \\ +\infty \quad \text{if no such path exists.} \end{cases}$$

Then,

(a) $d(f)$ never decreases during the execution of the algorithm

(b) $d(f)$ increases at least once per m iterations.

$d(f) \in \{0, 1, \cdots, n-1, +\infty\}$

Running Time Analysis of Edmonds-Karp

Key lemma: Fix a network G. For a flow f, define:

$$d(f) = \begin{cases} \text{the number of edges in a shortest } s\text{-}t \text{ path in } G_f \\ \text{with positive residual capacity} \\ \\ +\infty \quad \text{if no such path exists.} \end{cases}$$

Then,

(a) $d(f)$ never decreases during the execution of the algorithm

(b) $d(f)$ increases at least once per m iterations.

$d(f) \in \{0, 1, \cdots, n-1, +\infty\}$

Once $d(f) \geq n \Rightarrow d(f) = +\infty$

Running Time Analysis of Edmonds-Karp

Key lemma: Fix a network G. For a flow f, define:

$$d(f) = \begin{cases} \text{the number of edges in a shortest } s\text{-}t \text{ path in } G_f \\ \text{with positive residual capacity} \\ +\infty \quad \text{if no such path exists.} \end{cases}$$

Then,

(a) $d(f)$ never decreases during the execution of the algorithm

(b) $d(f)$ increases at least once per m iterations.

$d(f) \in \{0, 1, \cdots, n-1, +\infty\}$

Once $d(f) \geq n \Rightarrow d(f) = +\infty$ done!

Running Time Analysis of Edmonds-Karp

Key lemma: Fix a network G. For a flow f, define:

$$d(f) = \begin{cases} \text{the number of edges in a shortest } s\text{-}t \text{ path in } G_f \\ \text{with positive residual capacity} \\ +\infty \quad \text{if no such path exists.} \end{cases}$$

Then,

(a) $d(f)$ never decreases during the execution of the algorithm

(b) $d(f)$ increases at least once per m iterations.

$d(f) \in \{0, 1, \cdots, n-1, +\infty\}$

Once $d(f) \geq n \Rightarrow d(f) = +\infty$ done!

$O(mn)$ iterations

Running Time Analysis of Edmonds-Karp

Key lemma: Fix a network G. For a flow f, define:

$$d(f) = \begin{cases} \text{the number of edges in a shortest } s\text{-}t \text{ path in } G_f \\ \text{with positive residual capacity} \\ +\infty \quad \text{if no such path exists.} \end{cases}$$

Then,

(a) $d(f)$ never decreases during the execution of the algorithm

(b) $d(f)$ increases at least once per m iterations.

$d(f) \in \{0, 1, \dots, n-1, +\infty\}$

Once $d(f) \geq n \Rightarrow d(f) = +\infty$ done!

$O(mn)$ iterations $\xrightarrow{\text{BFS per iteration}}$ $O(m^2 n)$ time.

Proof of key lemma

PROOF OF KEY LEMMA

Imagine running BFS in G_f starting from source s.

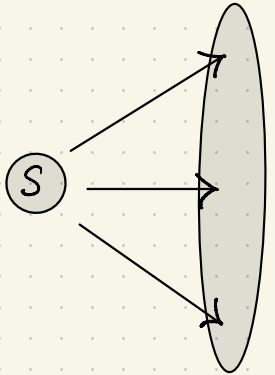
PROOF OF KEY LEMMA

Imagine running BFS in G_f starting from source s.



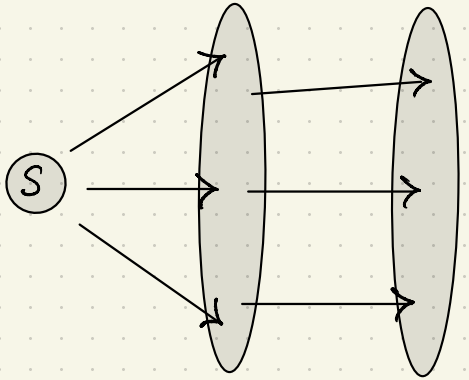
PROOF OF KEY LEMMA

Imagine running BFS in G_f starting from source s.



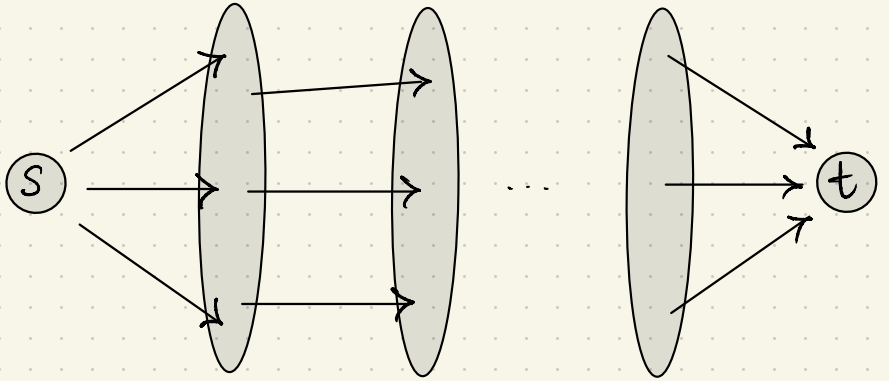
PROOF OF KEY LEMMA

Imagine running BFS in G_f starting from source s.



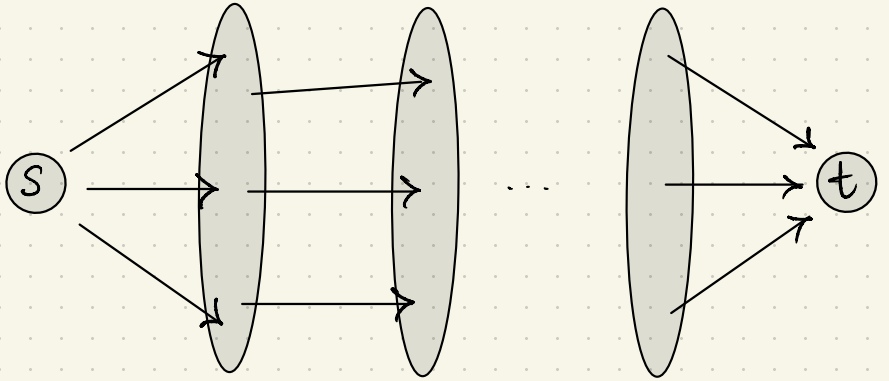
Proof of Key Lemma

Imagine running BFS in G_f starting from source s.



PROOF OF KEY LEMMA

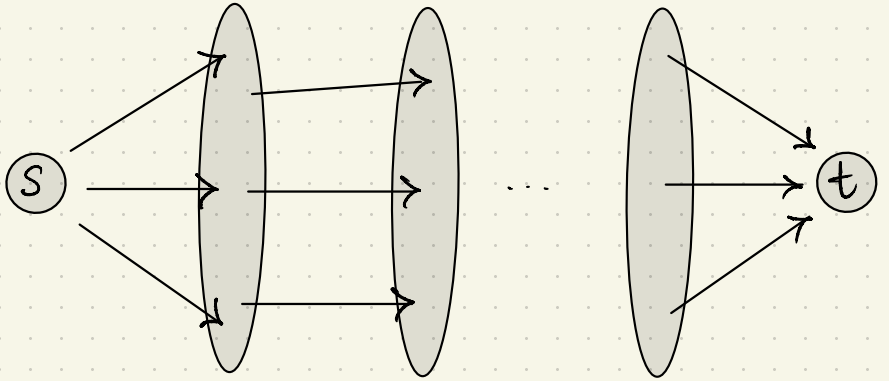
Imagine running BFS in G_f starting from source s.



Layer: 0 1 2 $d(f)-1$ $d(f)$

PROOF OF KEY LEMMA

Imagine running BFS in G_f starting from source s.

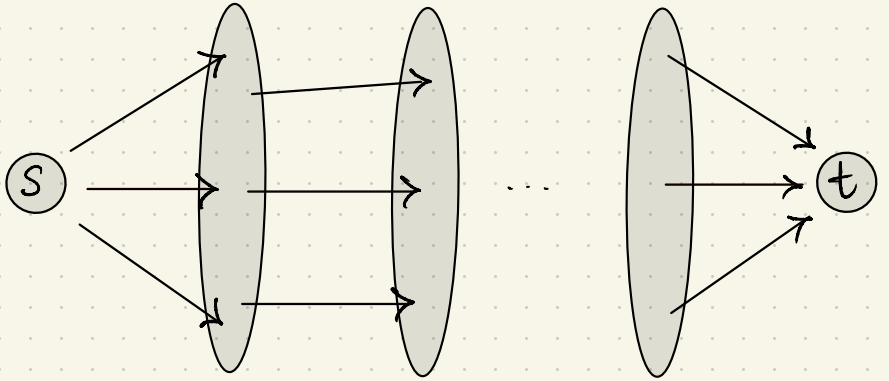


Layer : 0 1 2 $d(f)-1$ $d(f)$

recall that BFS finds shortest paths in unweighted graphs

PROOF OF KEY LEMMA

Imagine running BFS in G_f starting from source s.



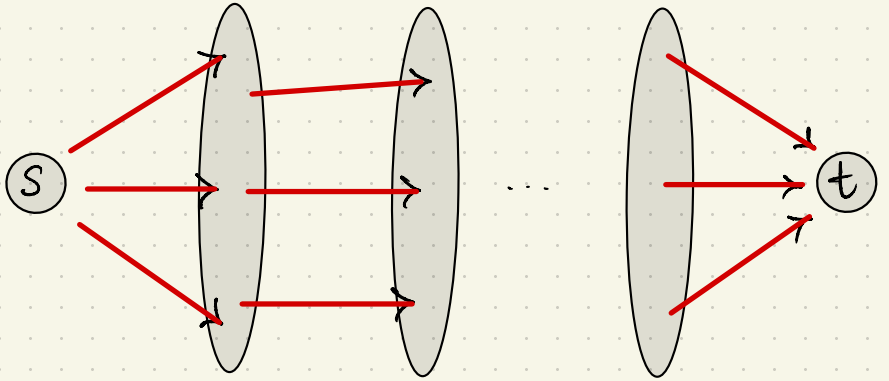
Layer: 0 1 2 $d(f)-1$ $d(f)$

Three types of edges in G_f:

* forward:
* sideways
* backward

PROOF OF KEY LEMMA

Imagine running BFS in G_f starting from source s.



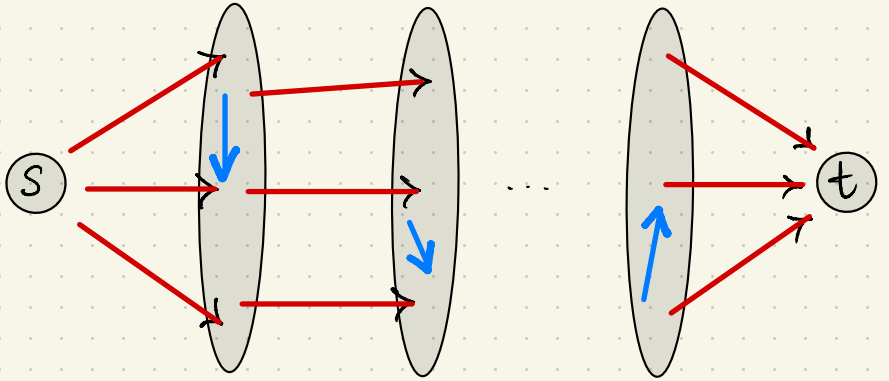
Layer: 0 1 2 $d(f)-1$ $d(f)$

Three types of edges in G_f:

* forward: going from layer i to layer $i+1$ No shortcuts!
* sideways
* backward

PROOF OF KEY LEMMA

Imagine running BFS in G_f starting from source s.



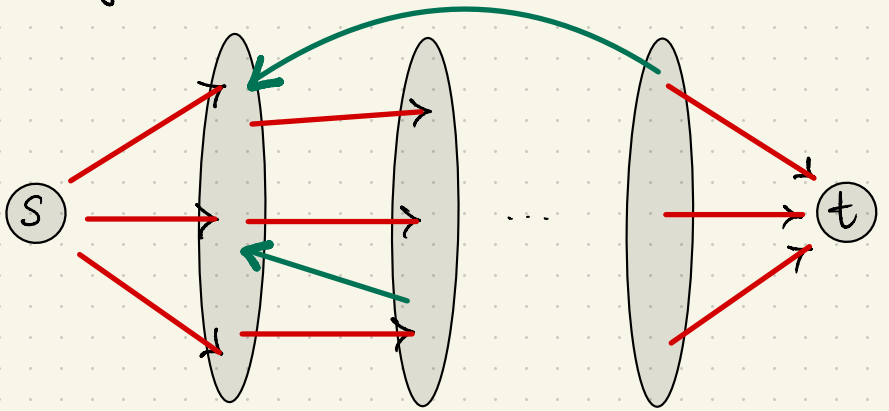
Layer: 0 1 2 $d(f)-1$ $d(f)$

Three types of edges in G_f:

* forward: going from layer i to layer $i+1$
* sideways: both endpoints in the same layer
* backward

PROOF OF KEY LEMMA

Imagine running BFS in G_f starting from source s.



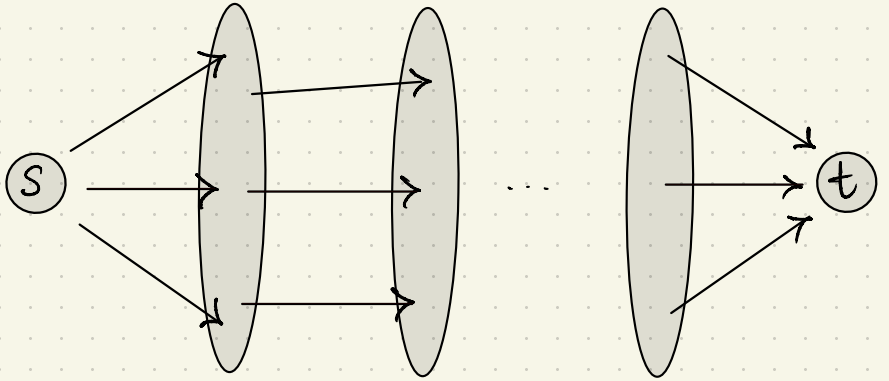
Layer: 0 1 2 $d(f)-1$ $d(f)$

Three types of edges in G_f:

* forward: going from layer i to layer $i+1$
* sideways: both endpoints in the same layer
* backward: going from layer i to layer $j < i$

PROOF OF KEY LEMMA

Imagine running BFS in G_f starting from source s.

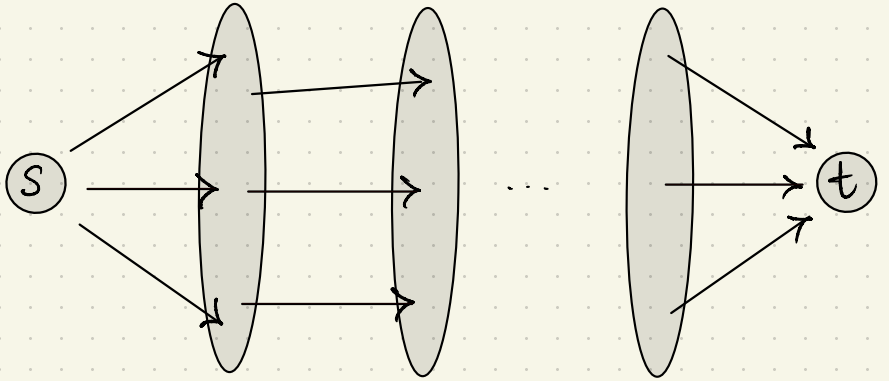


Layer : 0 1 2 $d(f)-1$ $d(f)$

Retain only forward edges in G_f.

PROOF OF KEY LEMMA

Imagine running BFS in G_f starting from source s.



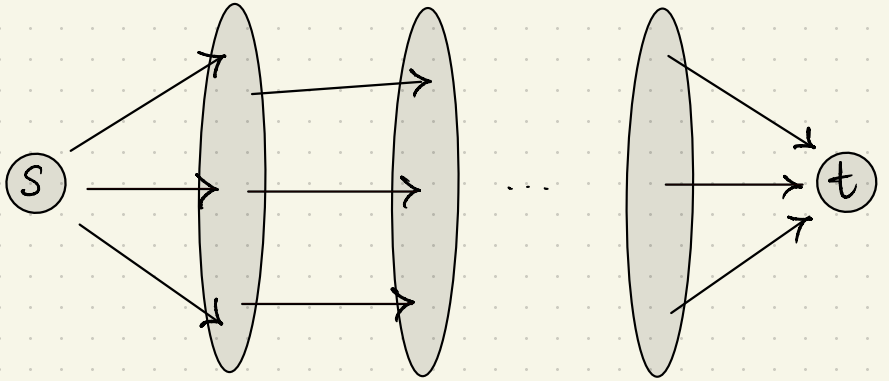
Layer : 0 1 2 $d(f)-1$ $d(f)$

Retain only forward edges in G_f.

Call the subgraph L_f ("layered" graph)

PROOF OF KEY LEMMA

Imagine running BFS in G_f starting from source s.

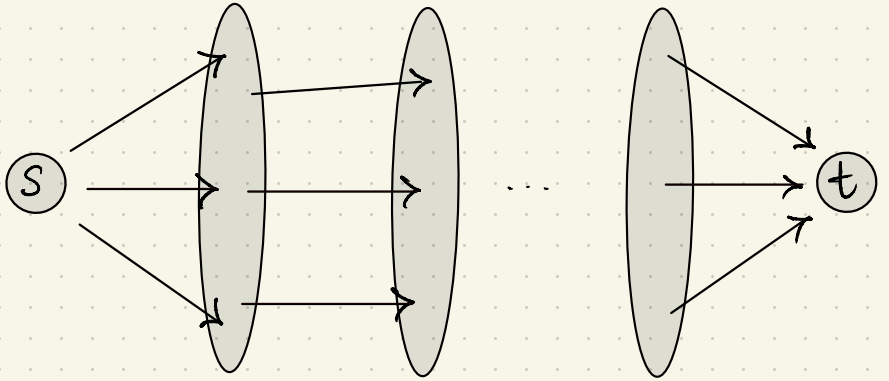


Layer : 0 1 2 $d(f)-1$ $d(f)$

NOTE : $s-t$ path in L_f $\Longleftrightarrow$ shortest $s-t$ path in G_f

PROOF OF KEY LEMMA

Imagine running BFS in G_f starting from source s.



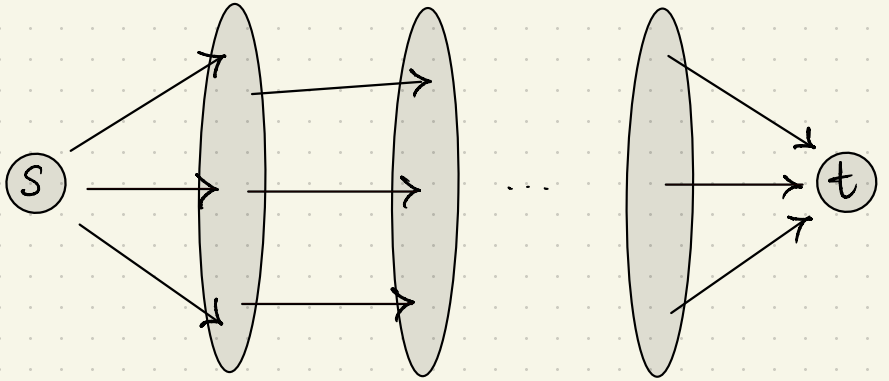
NOTE: s-t path in L_f $\iff$ shortest s-t path in G_f

($\Rightarrow$) any s-t path in L_f uses $d(f)$ edges.

By definition of $d(f)$, it is the shortest path in G_f.

PROOF OF KEY LEMMA

Imagine running BFS in G_f starting from source s.

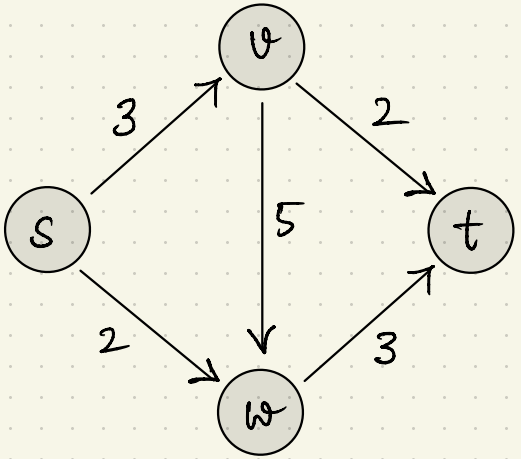


Layer: 0 1 2 $d(f)-1$ $d(f)$

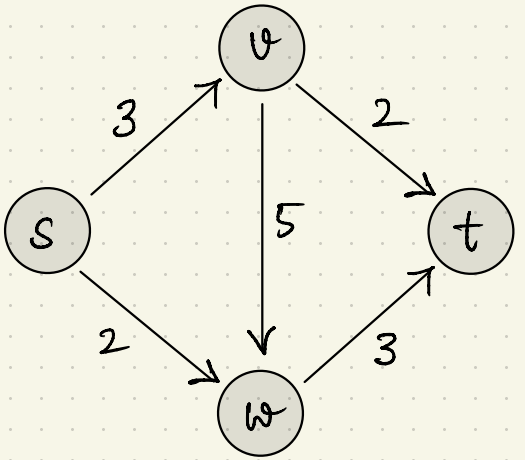
NOTE: s-t path in L_f $\iff$ shortest s-t path in G_f

($\Leftarrow$) an s-t path in G_f that is not in L_f must use sideways or backward edges $\Rightarrow$ # edges $> d(f)$ $\Rightarrow$ NOT shortest path.

Proof of key lemma

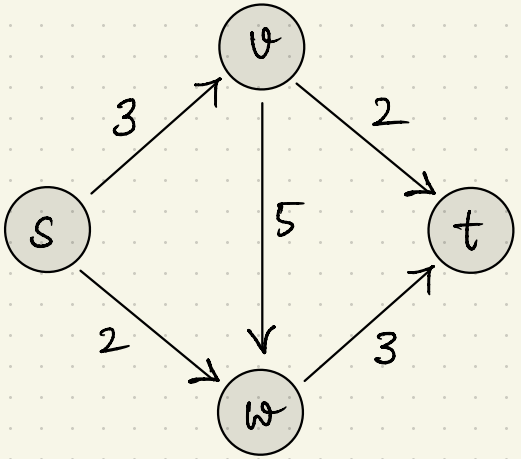


Proof of key lemma



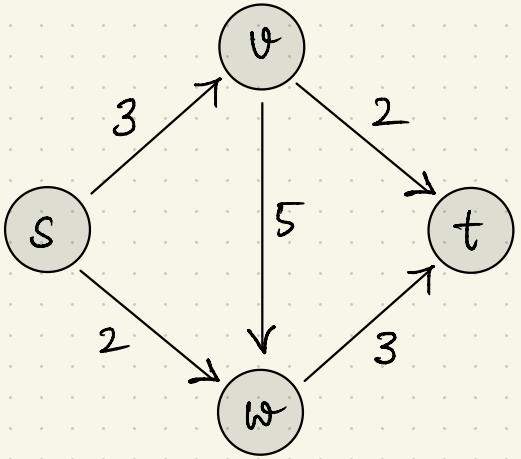
Initially zero flow $\Rightarrow$ $G_f = G$

Proof of key lemma



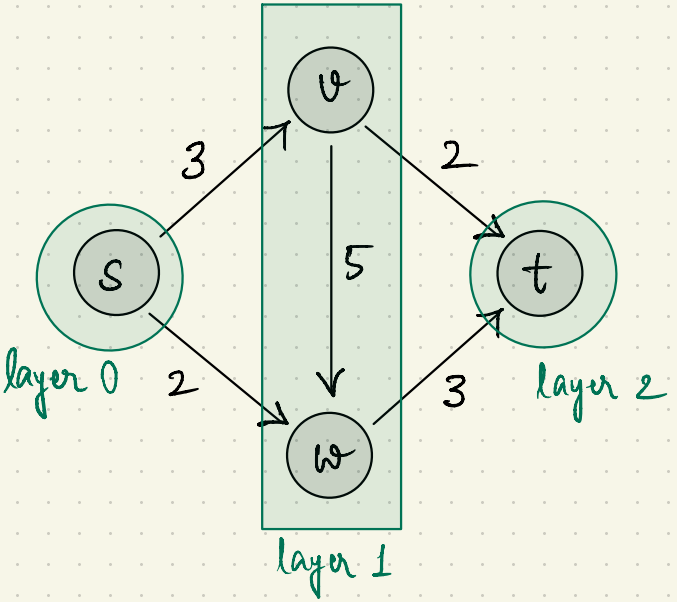
We will only draw G_f.

Proof of key lemma



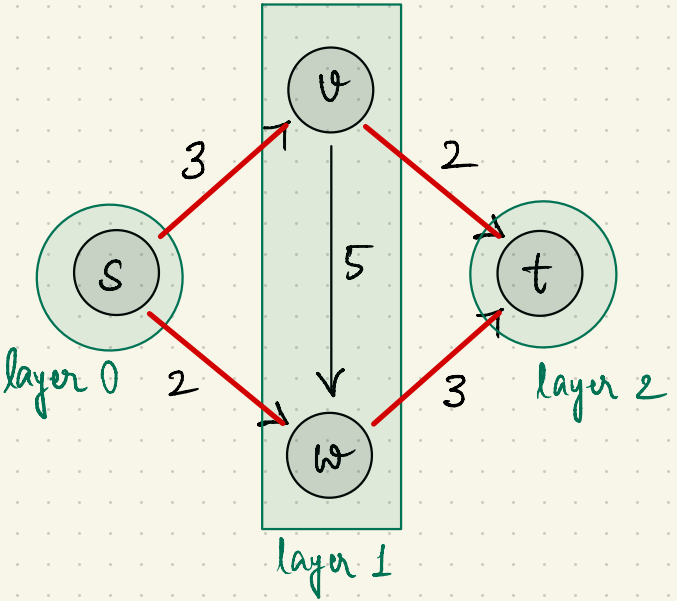
BFS from s

Proof of key lemma



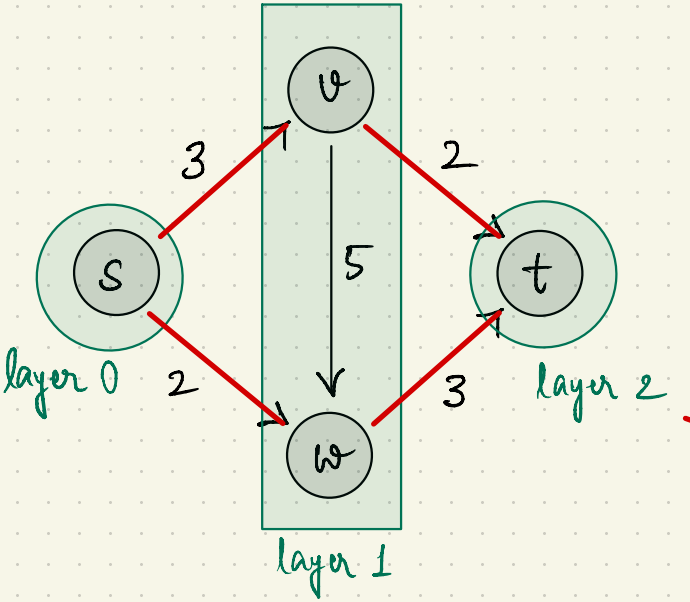
BFS from s

Proof of key lemma



Layered graph

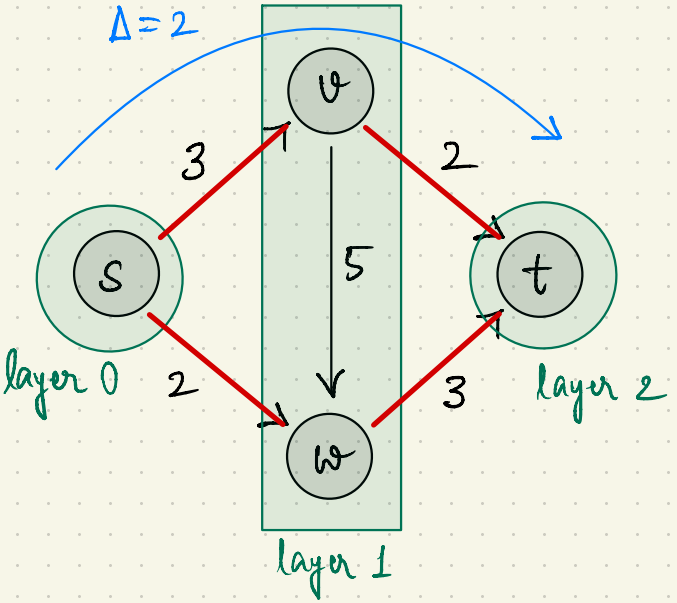
PROOF OF KEY LEMMA



d(f) = 2

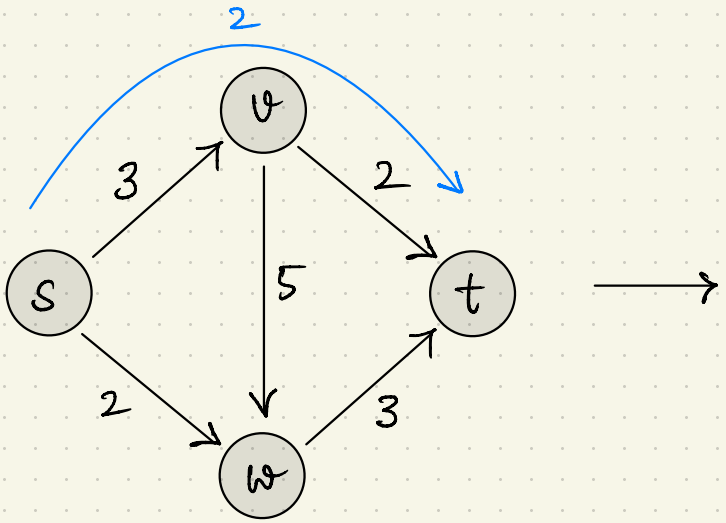
Layered graph

Proof of key lemma

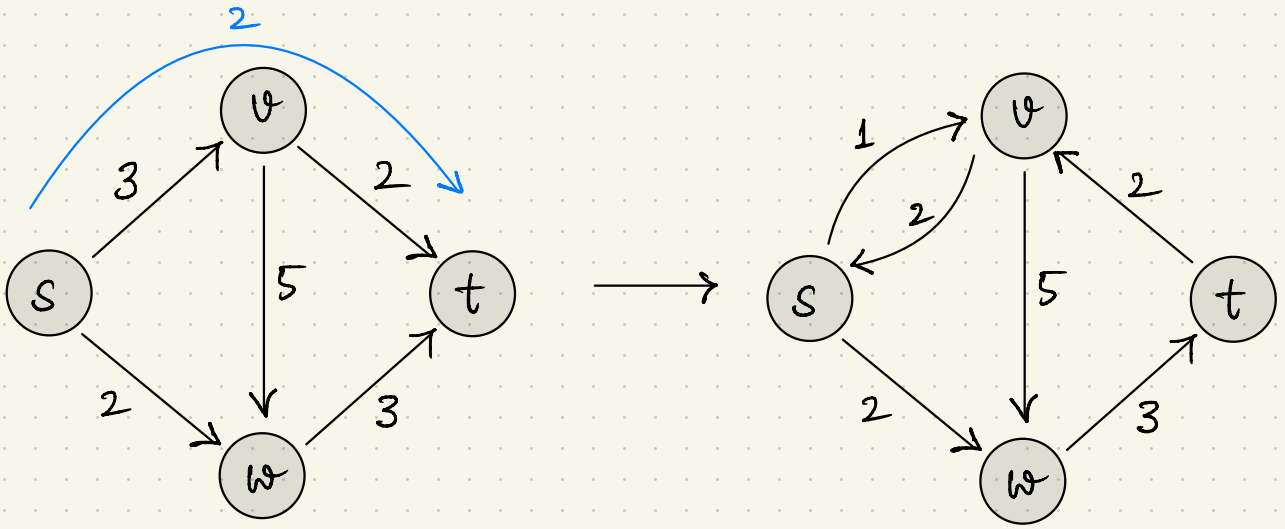


Suppose Edmonds-Karp augments along the top path

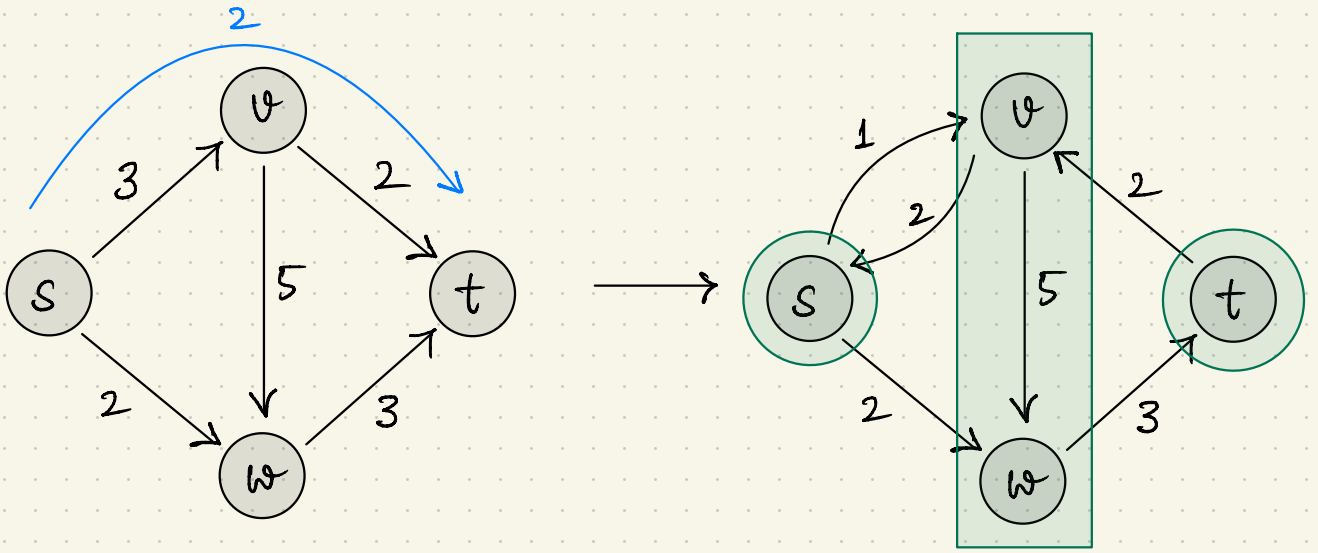
Proof of key lemma



Proof of key lemma

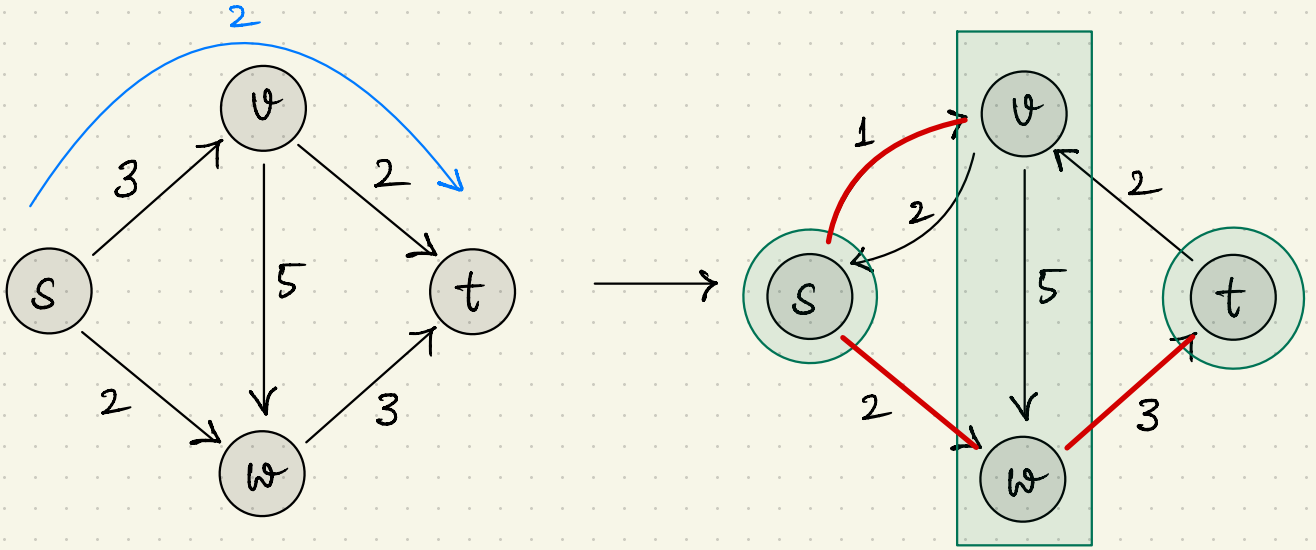


Proof of key lemma

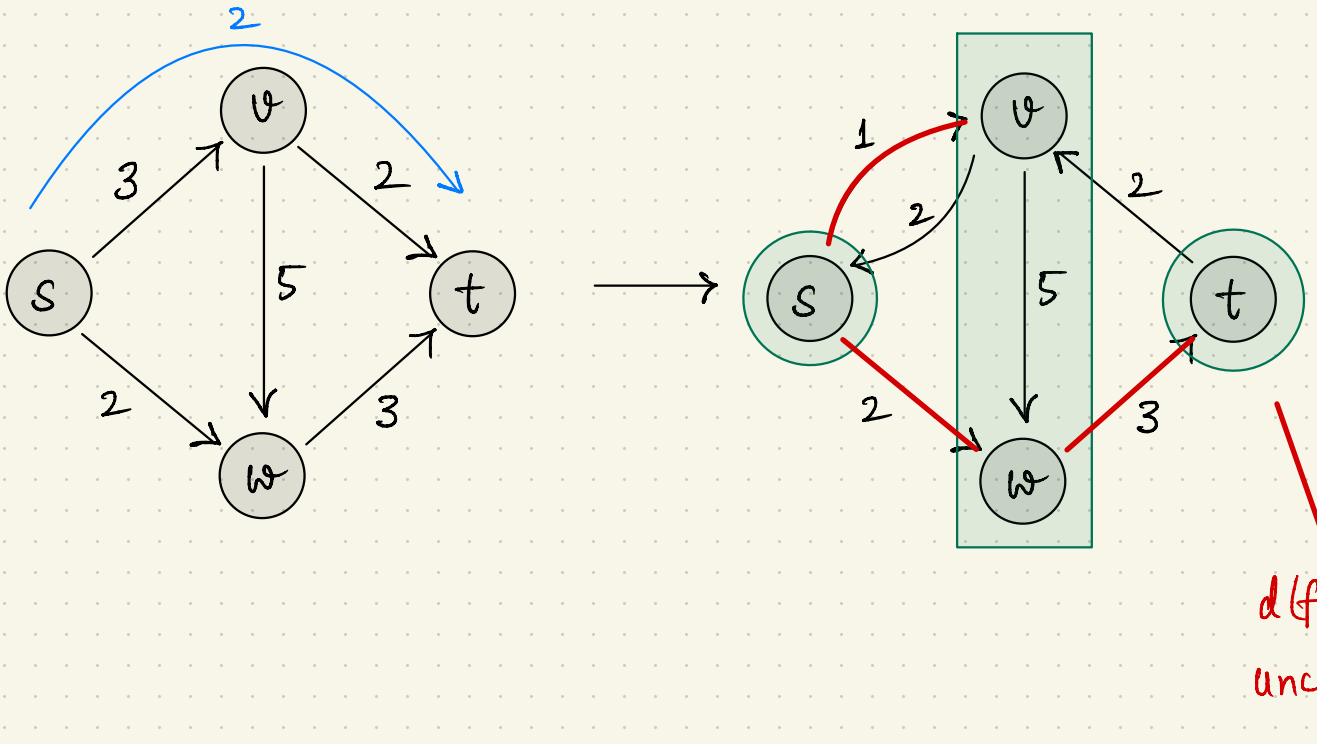


BFS from s in new G_f

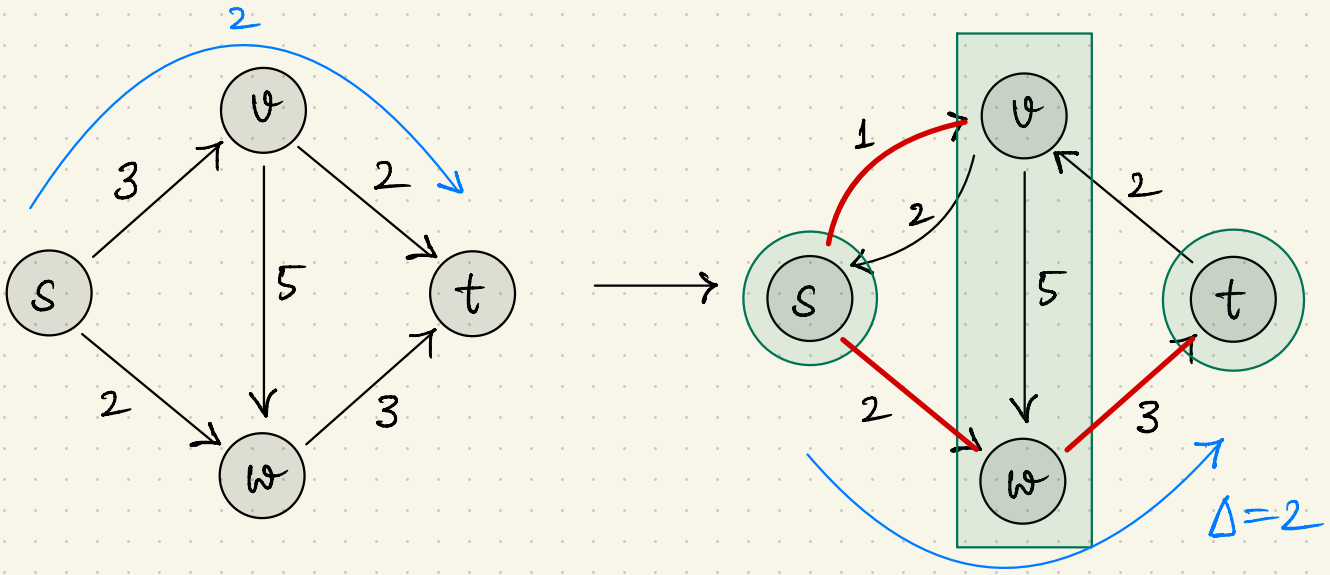
Proof of key lemma



Proof of Key Lemma

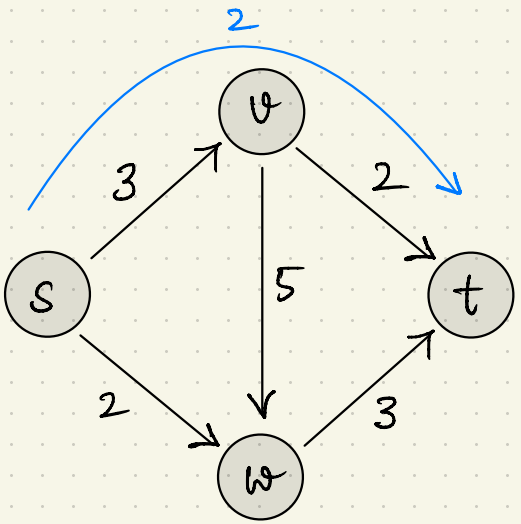


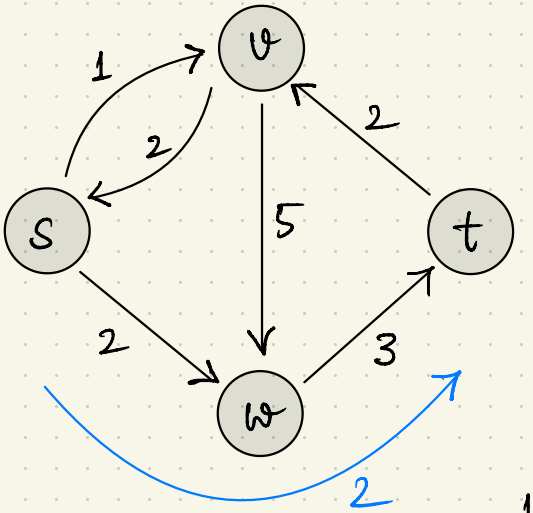
Proof of key lemma

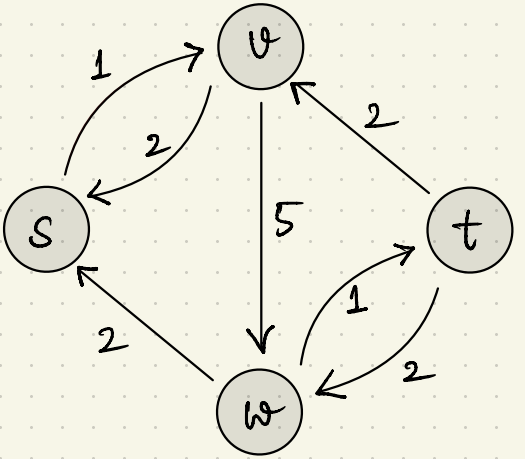


Edmonds-Karp augments along the bottom path

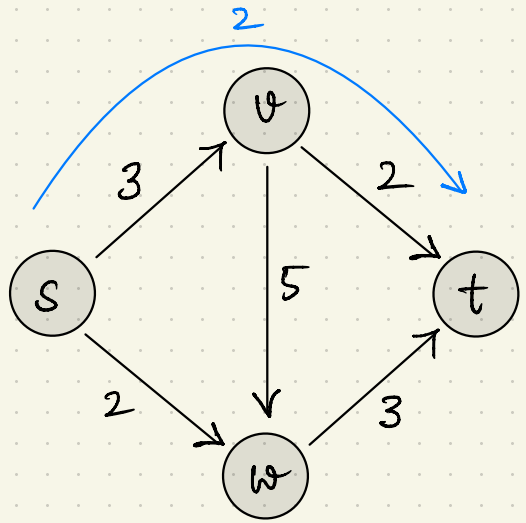
PROOF OF KEY LEMMA

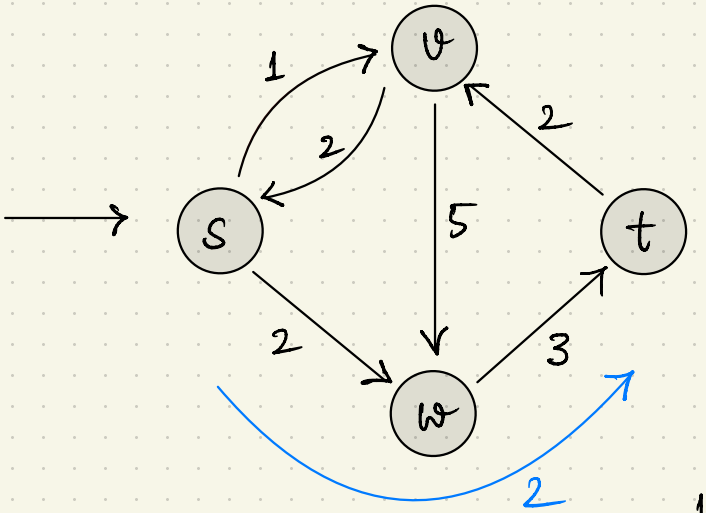


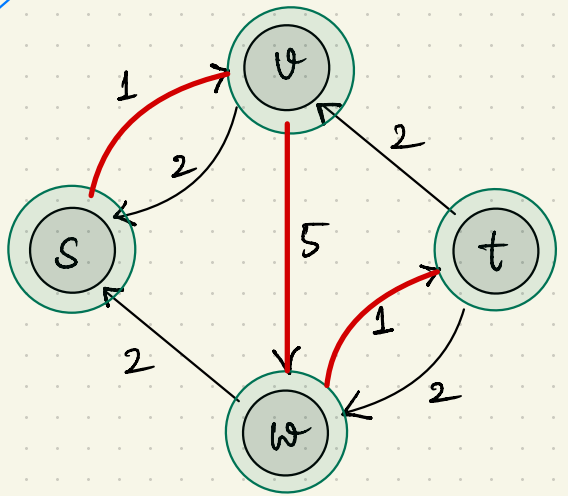




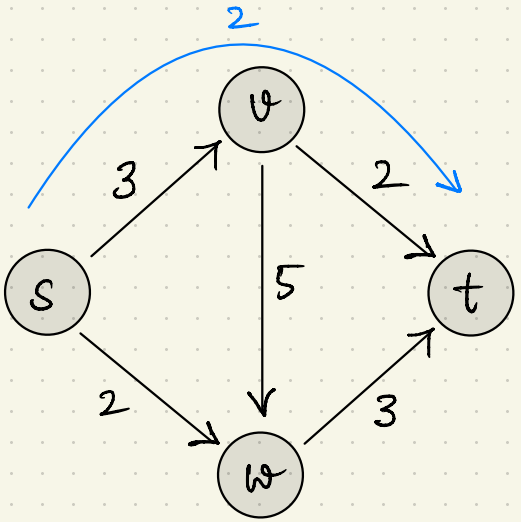
Proof of key lemma

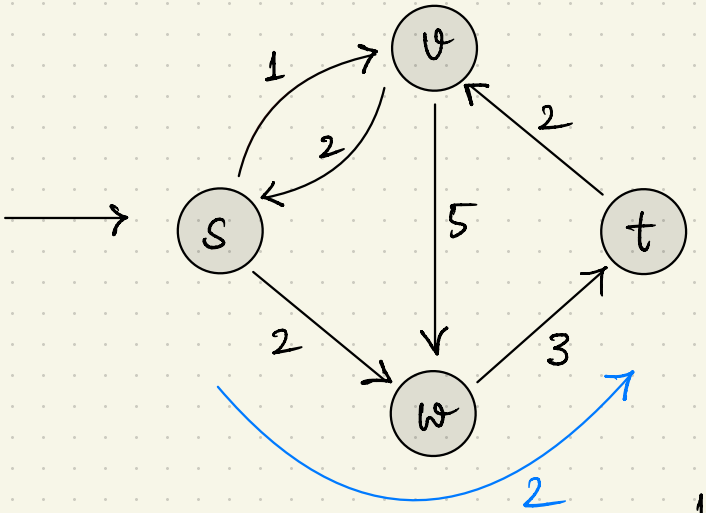


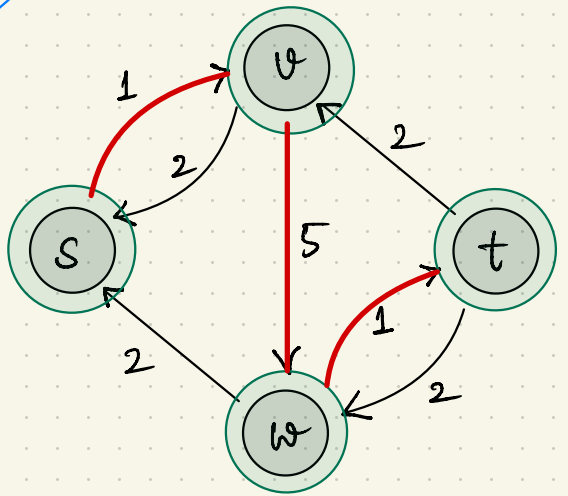




PROOF OF KEY LEMMA

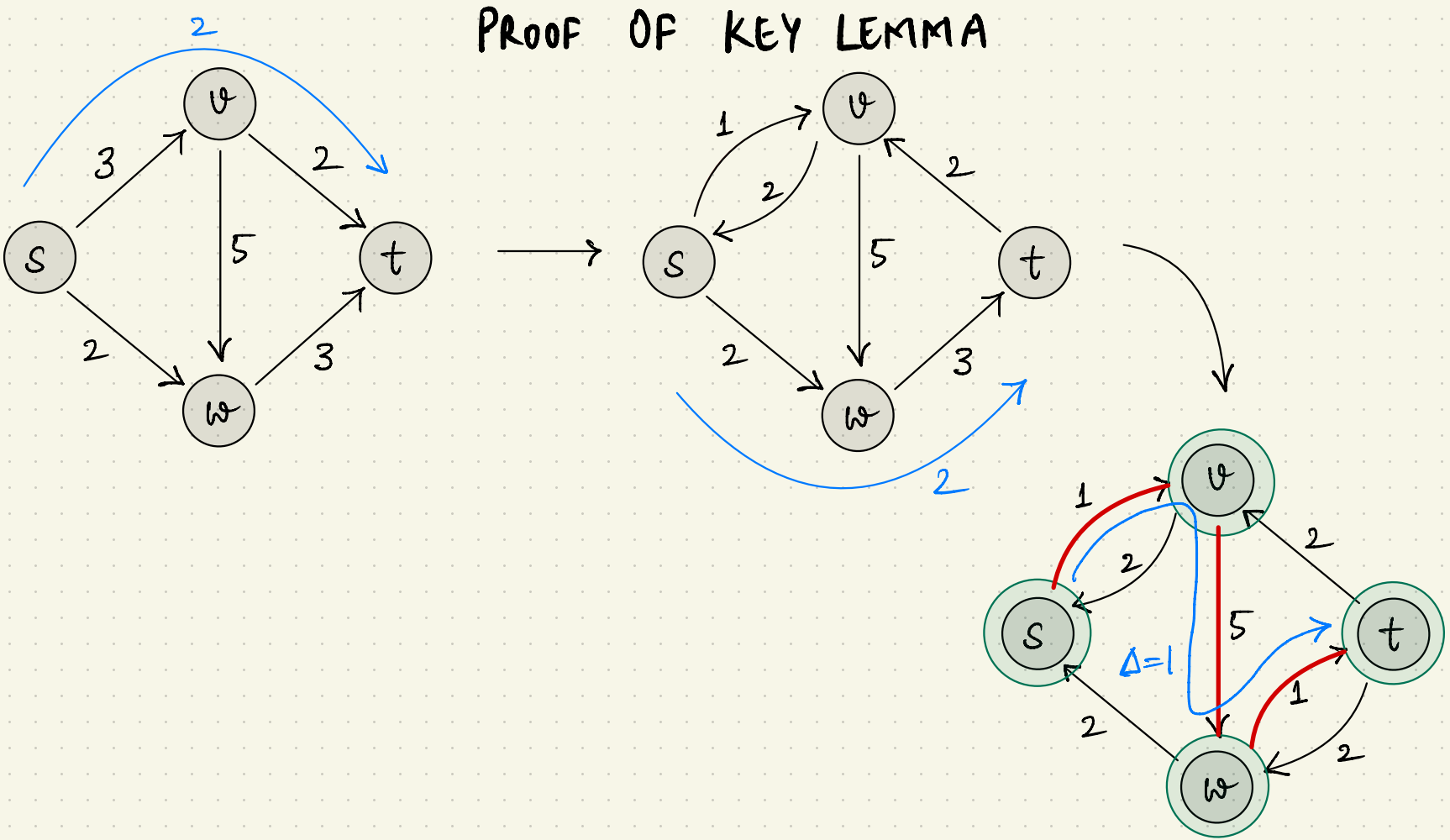




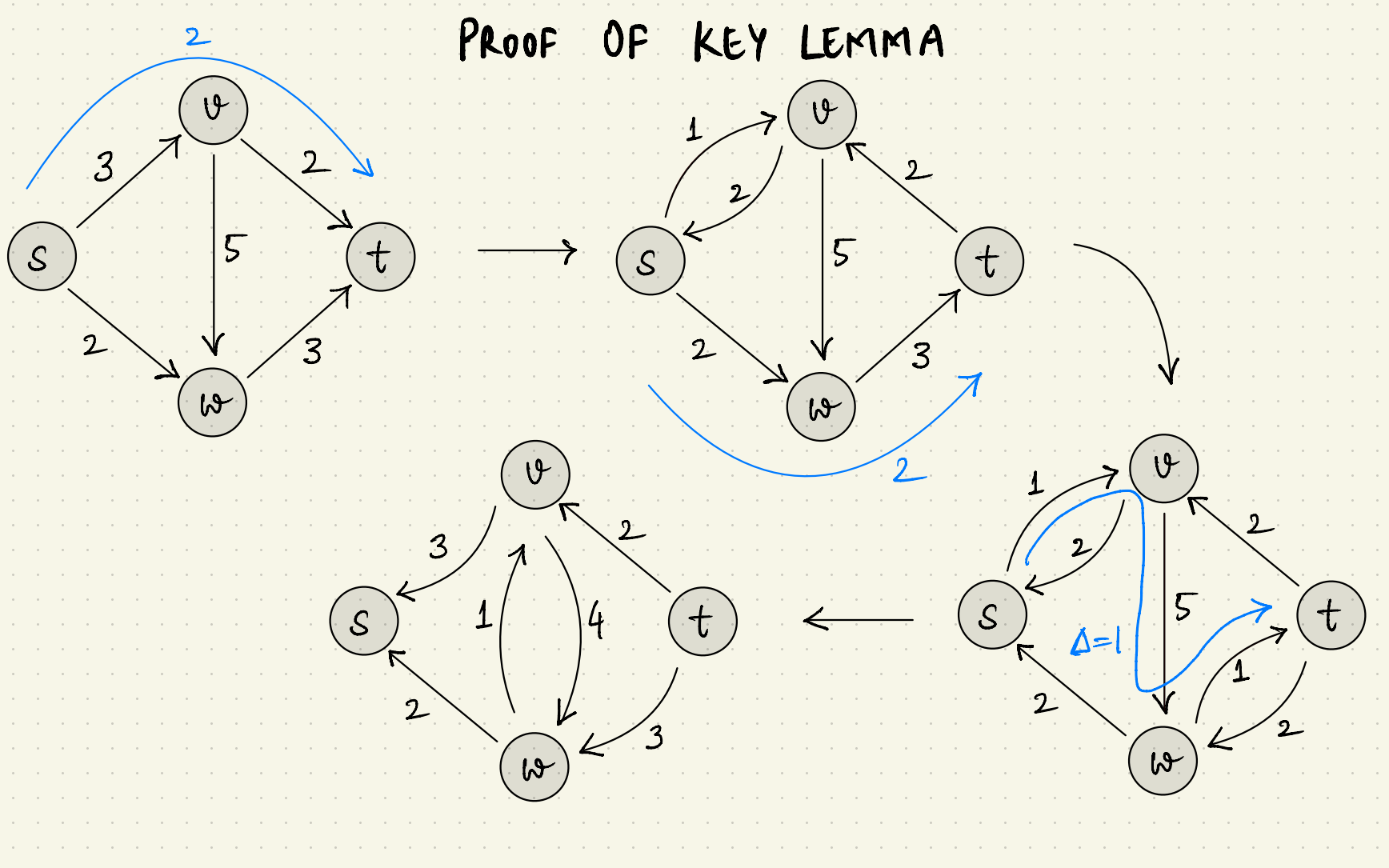


d(f) increases to 3

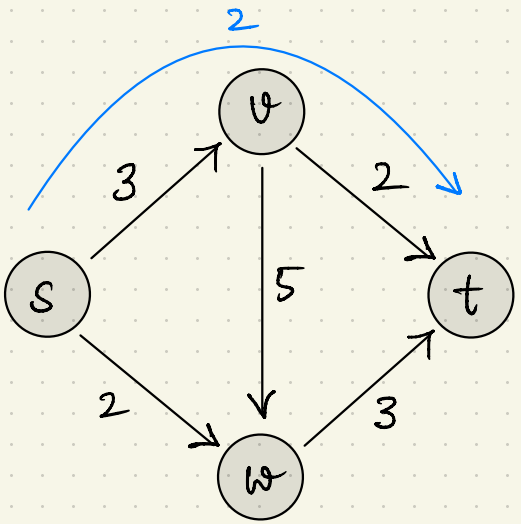
Proof of key lemma

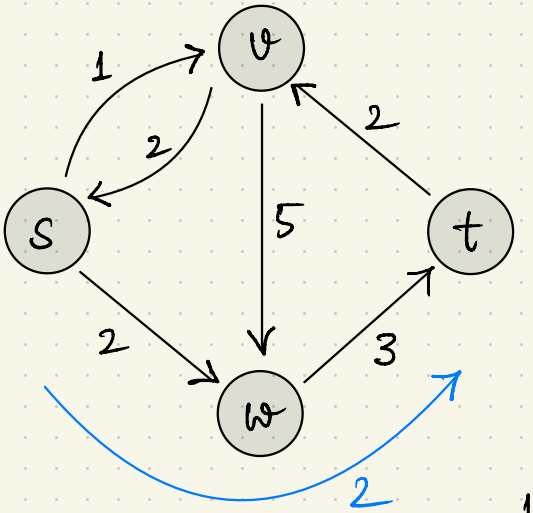


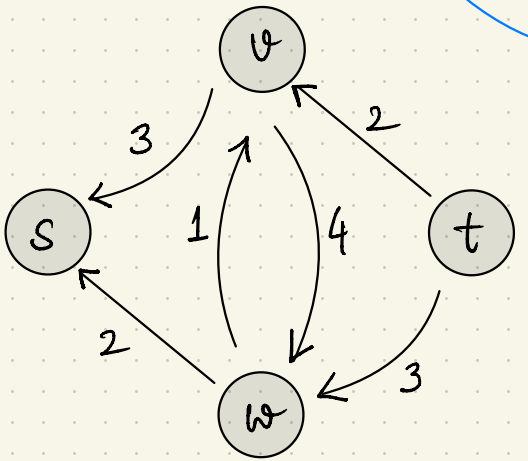
Proof of key lemma

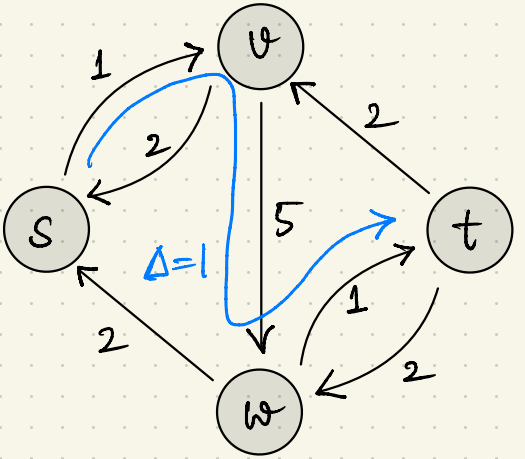


Proof of key lemma





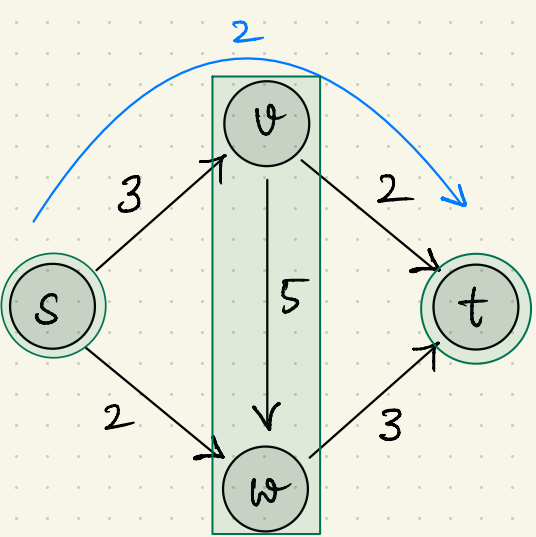


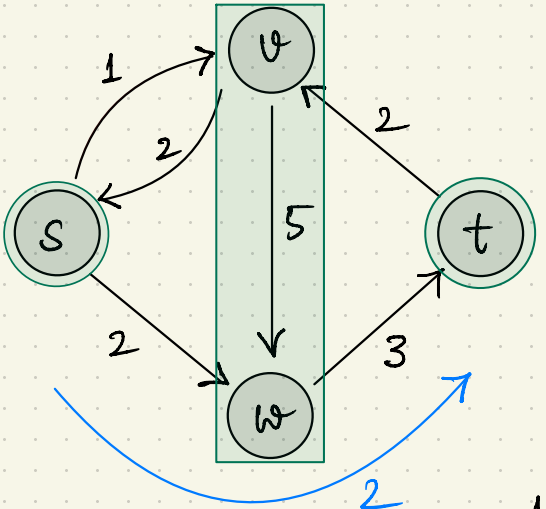


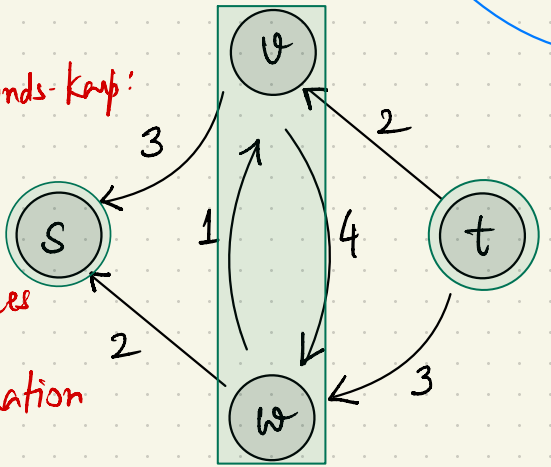
No s-t path

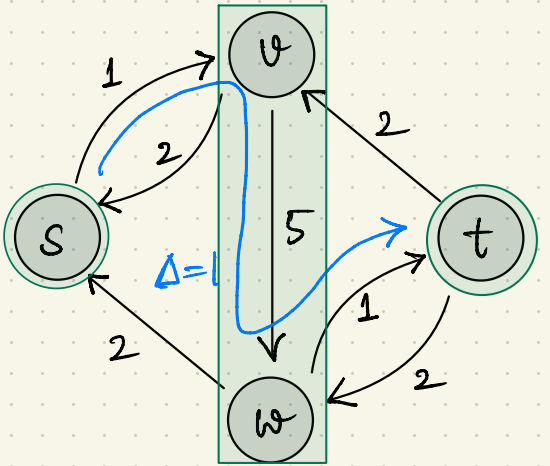
$\Rightarrow$ done!

PROOF OF KEY LEMMA









For analysis of Edmonds-Karp:

Use the same "grouping" of vertices as in the first iteration

PROOF OF KEY LEMMA

Key lemma: Fix a network G. For a flow f, define:

$$d(f) = \begin{cases} \text{the number of edges in a shortest } s\text{-}t \text{ path in } G_f \\ \text{with positive residual capacity} \\ +\infty \quad \text{if no such path exists.} \end{cases}$$

Then,

(a) $d(f)$ never decreases during the execution of the algorithm

(b) $d(f)$ increases at least once per m iterations.

PROOF OF KEY LEMMA

Key lemma: Fix a network G. For a flow f, define:

$$d(f) = \begin{cases} \text{the number of edges in a shortest } s\text{-}t \text{ path in } G_f \\ \text{with positive residual capacity} \\ +\infty \quad \text{if no such path exists.} \end{cases}$$

Then,

(a) $d(f)$ never decreases during the execution of the algorithm

(b) $d(f)$ increases at least once per m iterations.

PROOF OF KEY LEMMA

(a) $d(f)$ never decreases during the execution of the algorithm

PROOF OF KEY LEMMA

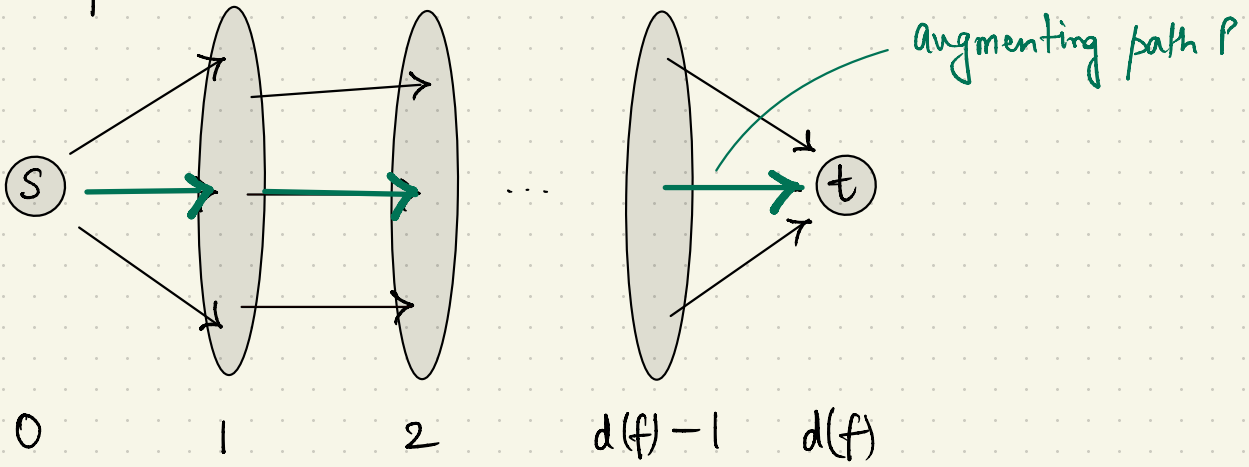
(a) $d(f)$ never decreases during the execution of the algorithm

Suppose, in iteration k, the algorithm augments along path P in G_f. Call the new flow f'.

PROOF OF KEY LEMMA

(a) $d(f)$ never decreases during the execution of the algorithm

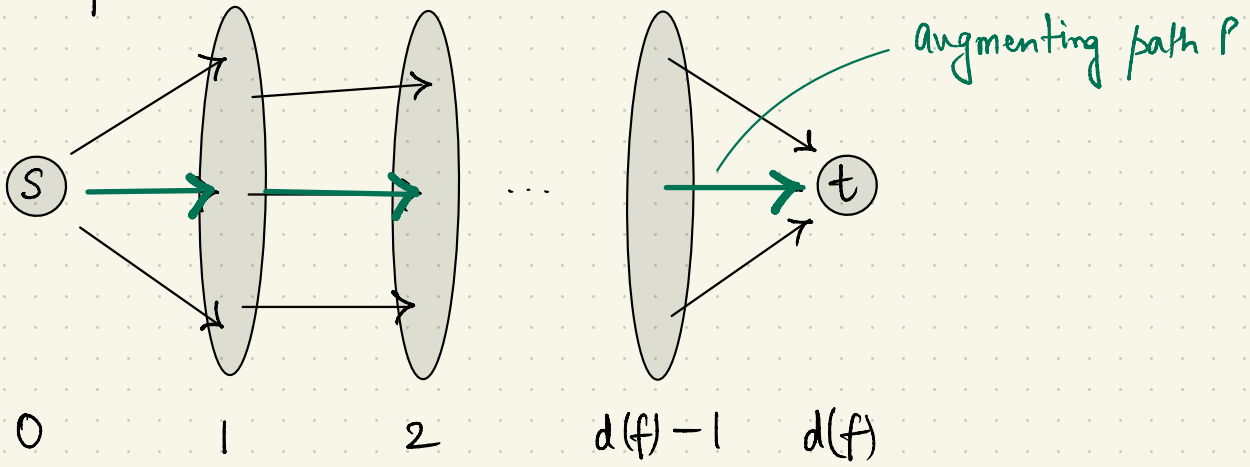
Suppose, in iteration k, the algorithm augments along path P in G_f. Call the new flow f'.



PROOF OF KEY LEMMA

(a) d(f) never decreases during the execution of the algorithm

Suppose, in iteration k, the algorithm augments along path P in G_f. Call the new flow f'.



It suffices to show that at the start of iteration k+1, any shortest s-t path in the updated residual graph $G_{f'}$ has length at least d(f).

PROOF OF KEY LEMMA

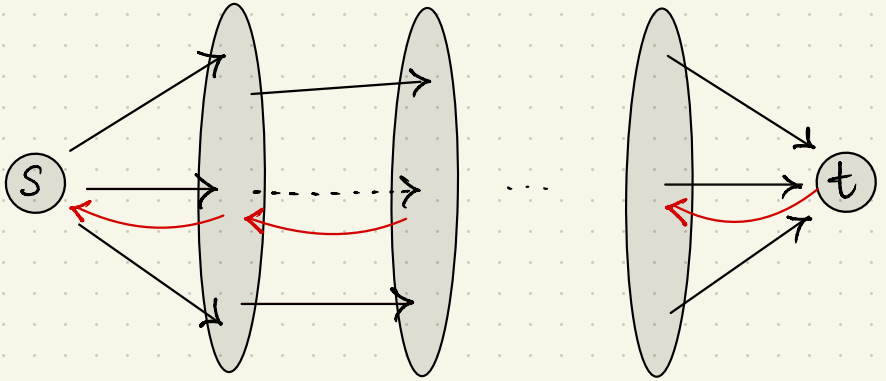
(a) $d(f)$ never decreases during the execution of the algorithm

How does the edge structure change upon augmentation along P?

PROOF OF KEY LEMMA

(a) d(f) never decreases during the execution of the algorithm

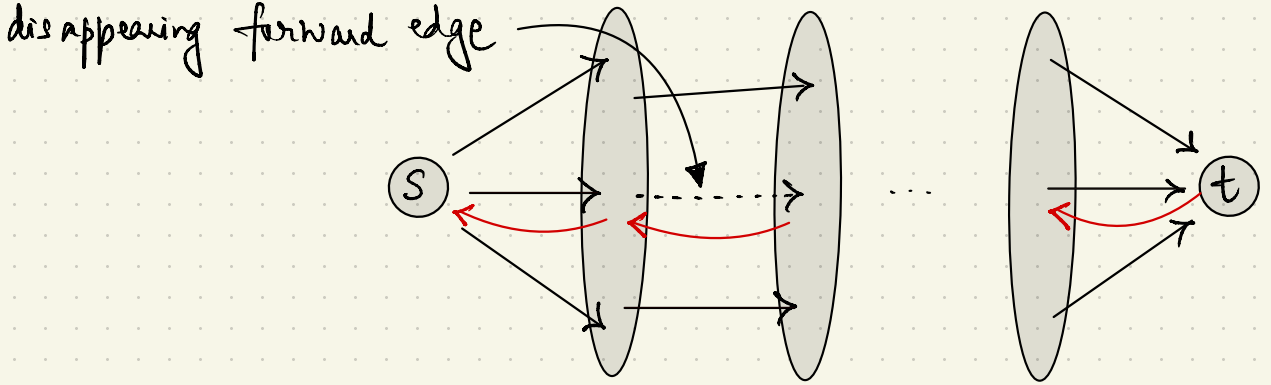
How does the edge structure change upon augmentation along P?



PROOF OF KEY LEMMA

(a) d(f) never decreases during the execution of the algorithm

How does the edge structure change upon augmentation along P?



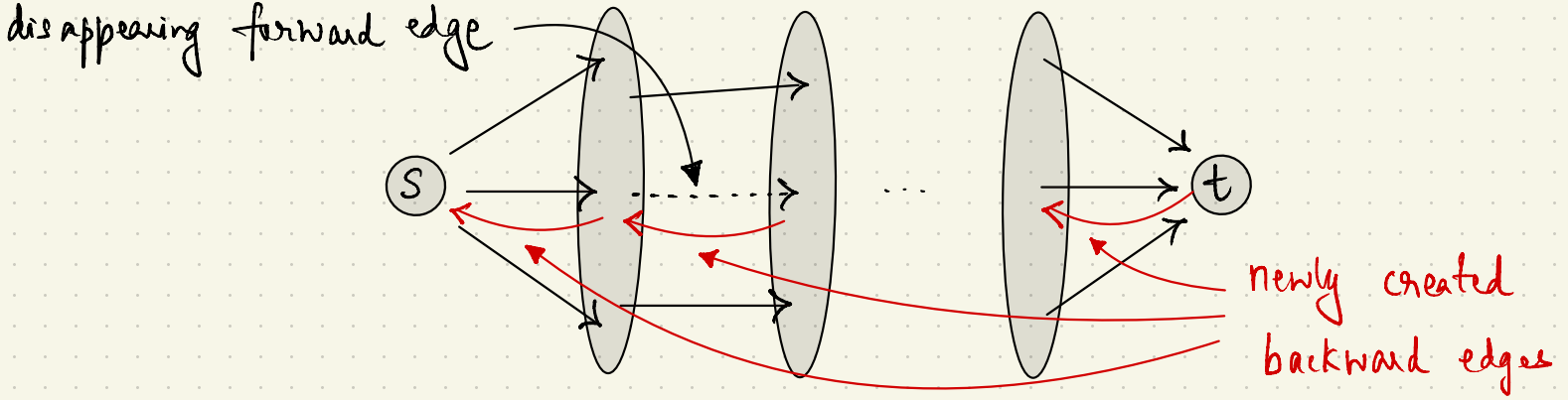
After augmentation along P:

* Some forward edge(s) in G_f might disappear (i.e., not present in $G_{f'}$).

PROOF OF KEY LEMMA

(a) $d(f)$ never decreases during the execution of the algorithm

How does the edge structure change upon augmentation along P?



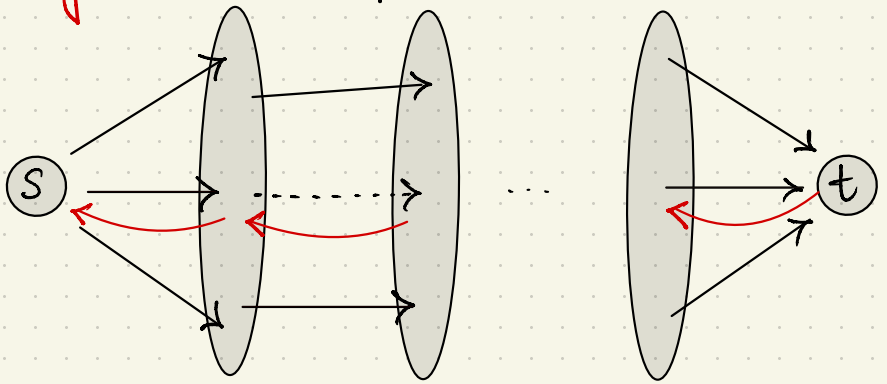
After augmentation along P:

* some forward edge(s) in G_f might disappear (i.e., not present in $G_{f'}$).
* some backward edge(s) in G_f are created (these might show up as forward / backward / sideways edges in $G_{f'}$).

PROOF OF KEY LEMMA

(a) d(f) never decreases during the execution of the algorithm

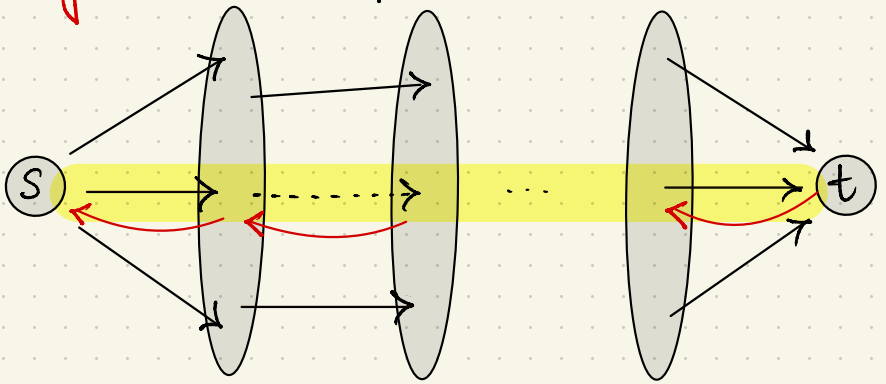
NOTE: Any newly created edge must be a backward edge between consecutive layers with respect to the old residual graph G_f.



PROOF OF KEY LEMMA

(a) $d(f)$ never decreases during the execution of the algorithm

NOTE: Any newly created edge must be a backward edge between consecutive layers with respect to the old residual graph G_f.



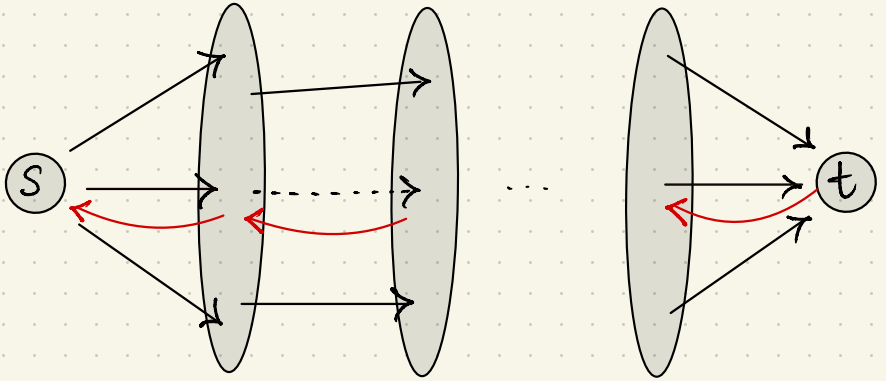
This is because each such edge is created in response to the augmentation along path P in the old residual graph G_f.

Recall that P only uses forward edges between consecutive layers.

PROOF OF KEY LEMMA

(a) $d(f)$ never decreases during the execution of the algorithm

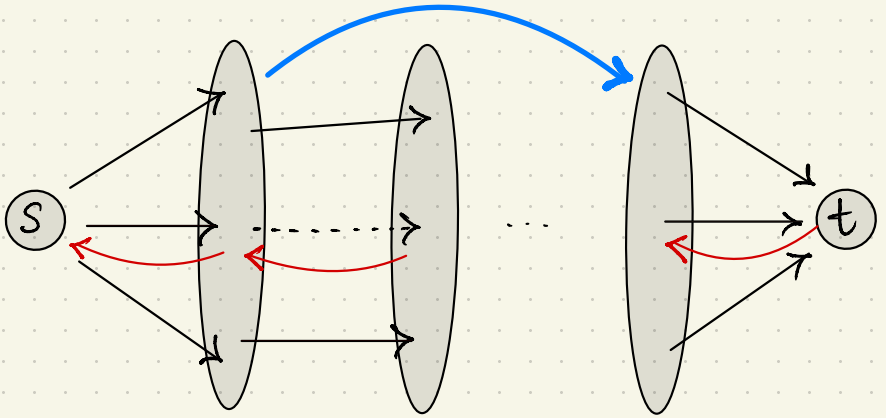
Let P' be any shortest s–t path in the updated residual graph $G_{f'}$.



PROOF OF KEY LEMMA

(a) $d(f)$ never decreases during the execution of the algorithm

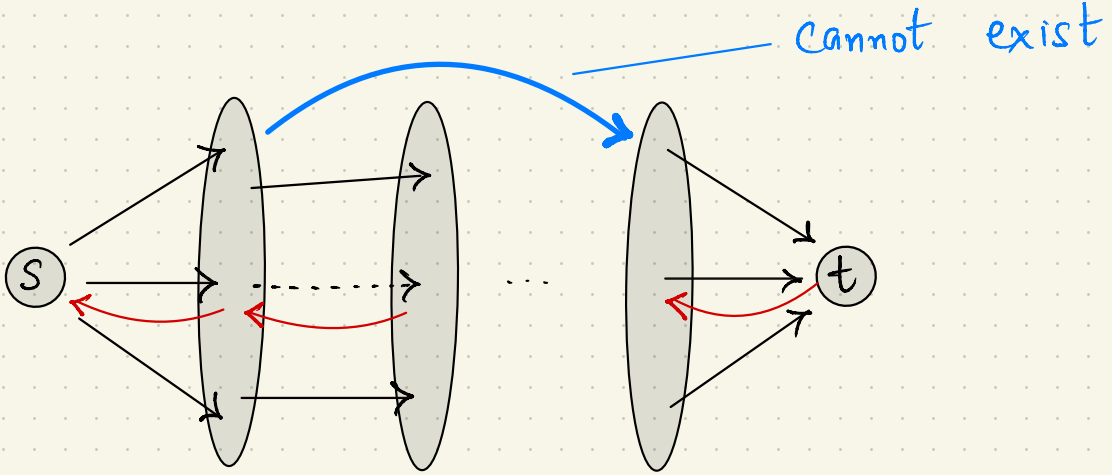
Let P' be any shortest s–t path in the updated residual graph $G_{f'}$.



For P' to be shorter than $d(f)$, it must use at least one shortcut edge that goes from some layer j to layer $(j+2)$ or more, where the layers are as defined in the old layered graph L_f.

PROOF OF KEY LEMMA

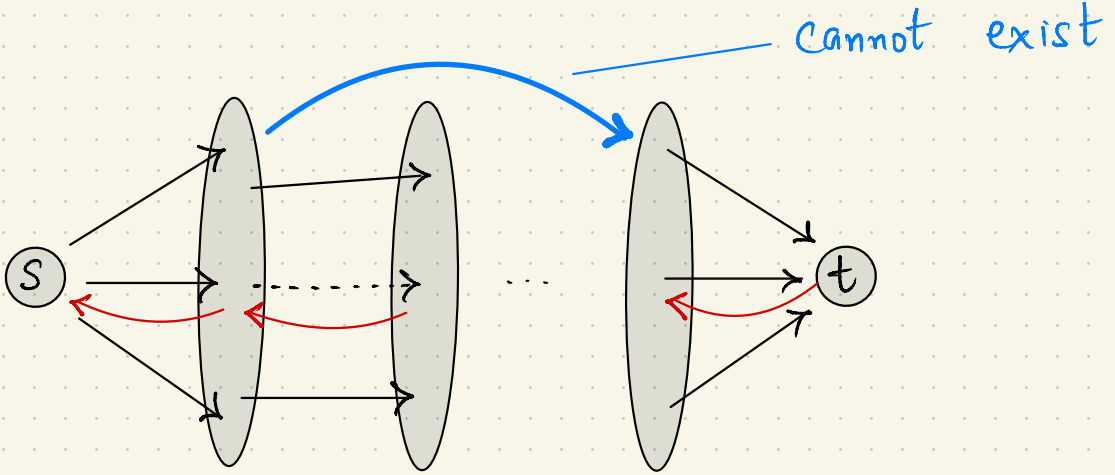
(a) $d(f)$ never decreases during the execution of the algorithm



But such a shortcut edge does not exist ! Why ?

PROOF OF KEY LEMMA

(a) d(f) never decreases during the execution of the algorithm

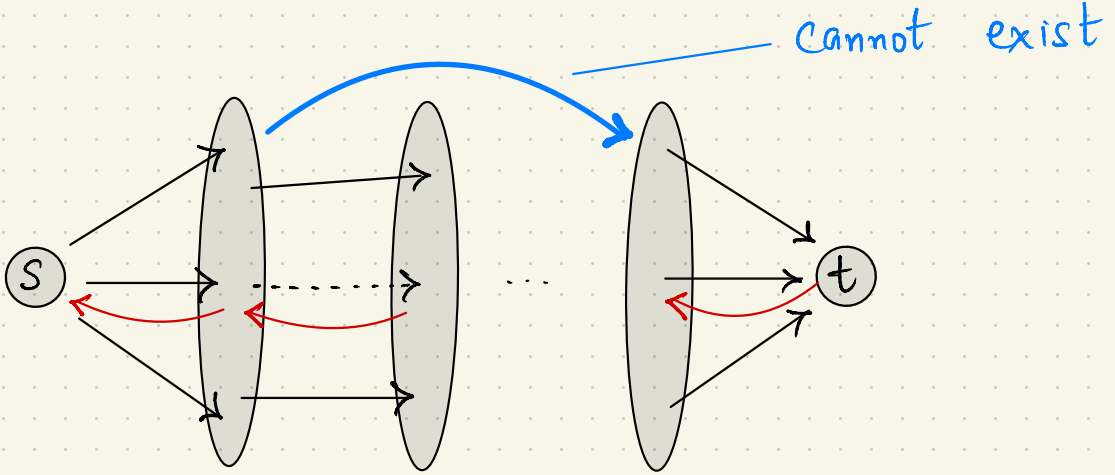


But such a shortcut edge does not exist! Why?

* No such edge in G_f, otherwise BFS would have identified it.

PROOF OF KEY LEMMA

(a) $d(f)$ never decreases during the execution of the algorithm



But such a shortcut edge does not exist! Why?

* No such edge in G_f, otherwise BFS would have identified it.
* " " " $G_{f'}$ because newly created edges only go backwards with respect to layers in old graph G_f.

PROOF OF KEY LEMMA

Key lemma: Fix a network G. For a flow f, define:

$$d(f) = \begin{cases} \text{the number of edges in a shortest } s\text{-}t \text{ path in } G_f \\ \text{with positive residual capacity} \\ +\infty \quad \text{if no such path exists.} \end{cases}$$

Then,

(a) $d(f)$ never decreases during the execution of the algorithm

(b) $d(f)$ increases at least once per m iterations.

PROOF OF KEY LEMMA

Key lemma: Fix a network G. For a flow f, define:

$$d(f) = \begin{cases} \text{the number of edges in a shortest } s\text{-}t \text{ path in } G_f \\ \text{with positive residual capacity} \\ +\infty \quad \text{if no such path exists.} \end{cases}$$

Then,

(a) $d(f)$ never decreases during the execution of the algorithm

(b) $d(f)$ increases at least once per m iterations.

PROOF OF KEY LEMMA

(b) $d(f)$ increases at least once per m iterations.

PROOF OF KEY LEMMA

(b) d(f) increases at least once per m iterations.

Consider a run of k consecutive iterations of Edmonds-Karp where the value of d(f) stays unchanged (say d(f) = c).

PROOF OF KEY LEMMA

(b) $d(f)$ increases at least once per m iterations.

Consider a run of k consecutive iterations of Edmonds-Karp where the value of $d(f)$ stays unchanged (say $d(f) = c$).

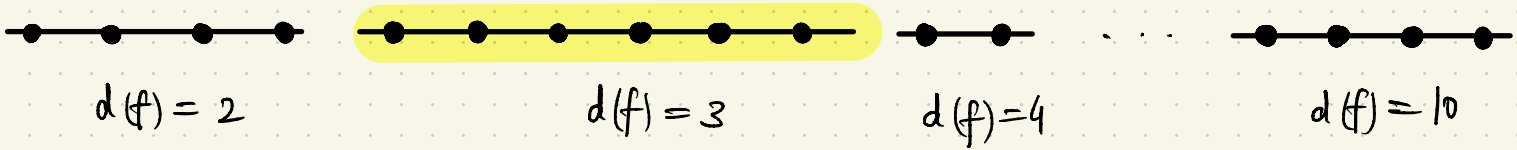
Want to show: $k \leq m$

PROOF OF KEY LEMMA

(b) $d(f)$ increases at least once per m iterations.

Consider a run of k consecutive iterations of Edmonds-Karp where the value of $d(f)$ stays unchanged (say $d(f) = c$).

Want to show: $k \leq m$

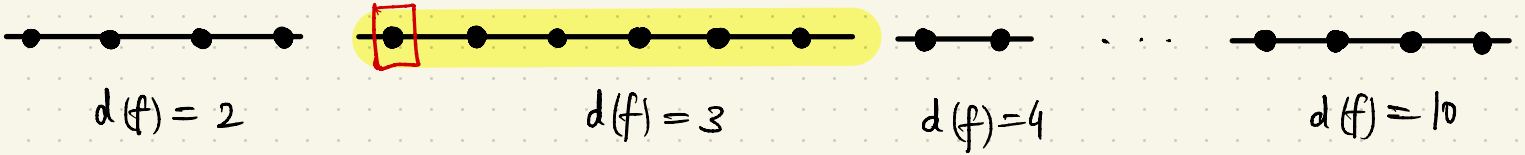


PROOF OF KEY LEMMA

(b) $d(f)$ increases at least once per m iterations.

Consider a run of k consecutive iterations of Edmonds-Karp where the value of $d(f)$ stays unchanged (say $d(f) = c$).

Want to show: $k \leq m$



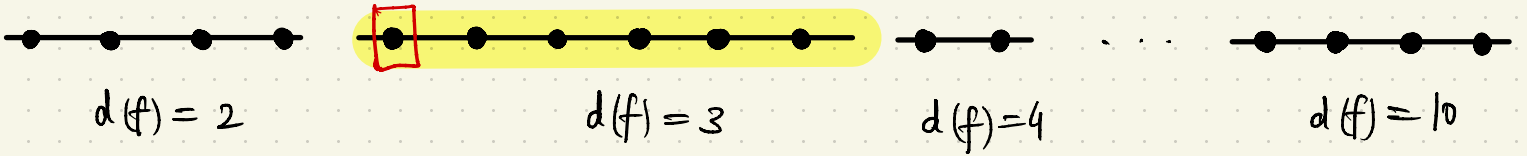
Save a copy of the layered graph L_f before the start of first such iteration

PROOF OF KEY LEMMA

(b) $d(f)$ increases at least once per m iterations.

Consider a run of k consecutive iterations of Edmonds-Karp where the value of $d(f)$ stays unchanged (say $d(f) = c$).

Want to show: $k \leq m$



Save a copy of the layered graph L_f before the start of first such iteration

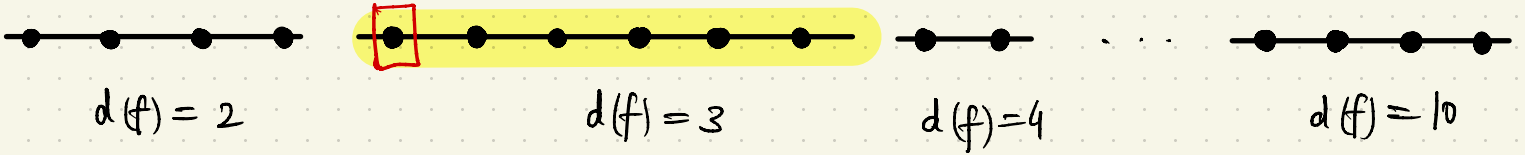
Let F = set of edges in the saved copy

PROOF OF KEY LEMMA

(b) $d(f)$ increases at least once per m iterations.

Consider a run of k consecutive iterations of Edmonds-Karp where the value of $d(f)$ stays unchanged (say $d(f) = c$).

Want to show: $k \leq m$



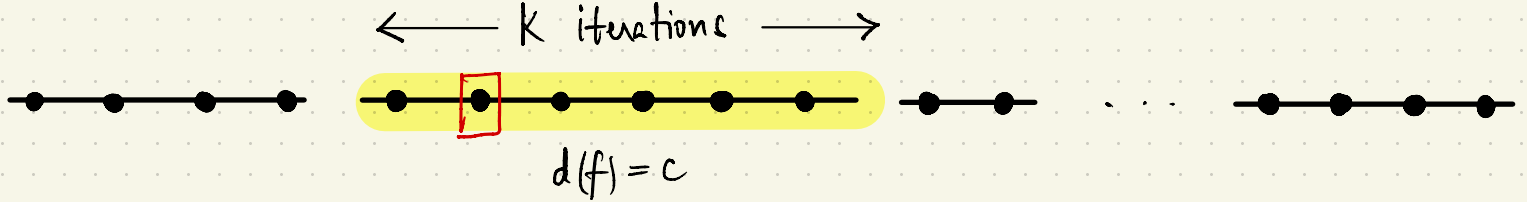
Save a copy of the layered graph L_f before the start of first such iteration

Let F = set of edges in the saved copy

Let $V_0 = \{s\}, V_1, V_2, \ldots, V_c$ be the layers in the saved copy.

PROOF OF KEY LEMMA

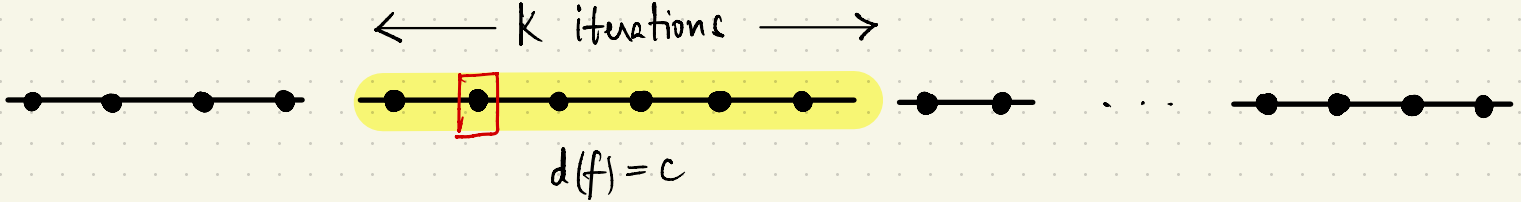
(b) $d(f)$ increases at least once per m iterations.



Consider the Second of these k iterations.

PROOF OF KEY LEMMA

(b) $d(f)$ increases at least once per m iterations.

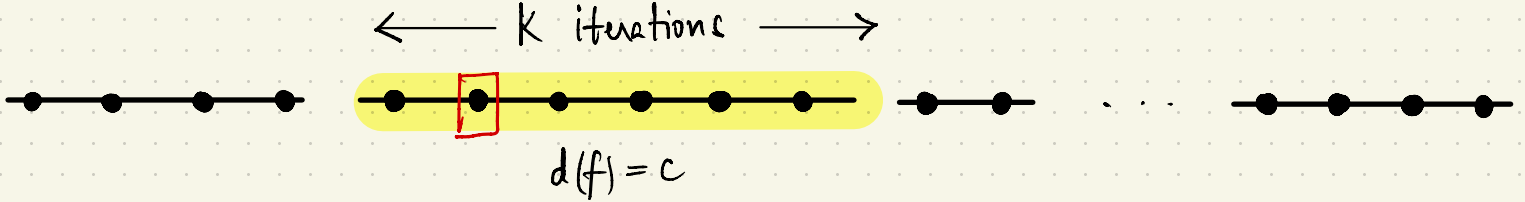


Consider the Second of these k iterations.

As argued in proof of part (a), any newly introduced edge after the first iteration goes from layer V_i to V_{i-1}.

PROOF OF KEY LEMMA

(b) $d(f)$ increases at least once per m iterations.



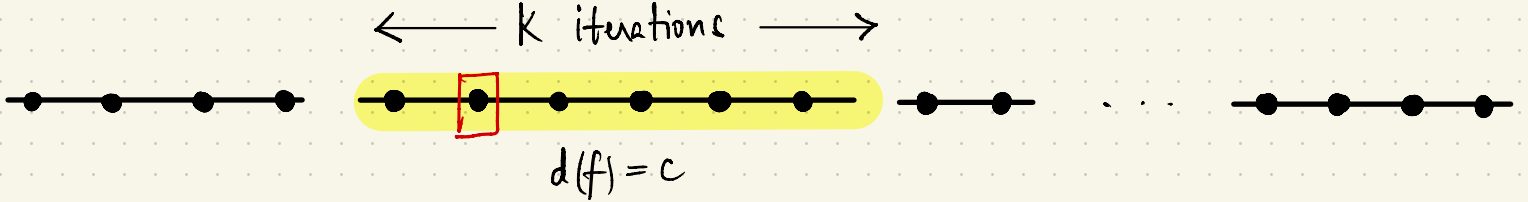
Consider the Second of these k iterations.

As argued in proof of part (a), any newly introduced edge after the first iteration goes from layer V_i to V_{i-1}.

defined wrt saved copy in first iteration

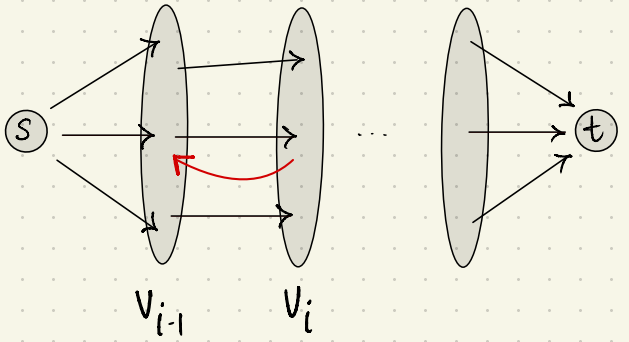
PROOF OF KEY LEMMA

(b) $d(f)$ increases at least once per m iterations.



Consider the Second of these k iterations.

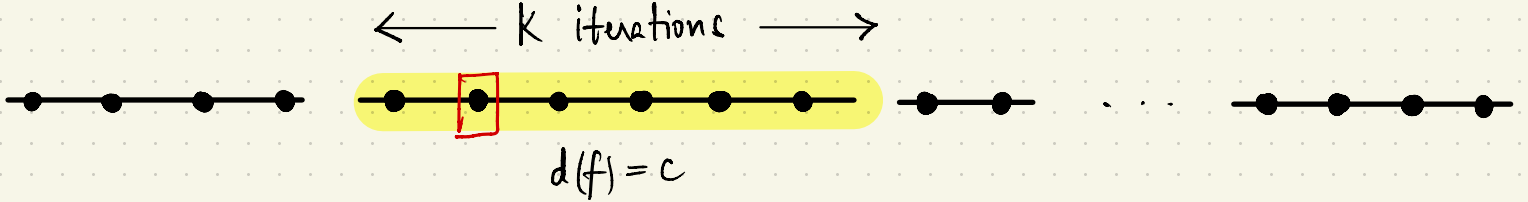
As argued in proof of part (a), any newly introduced edge after the first iteration goes from layer V_i to V_{i-1}.



defined wrt saved copy in first iteration

PROOF OF KEY LEMMA

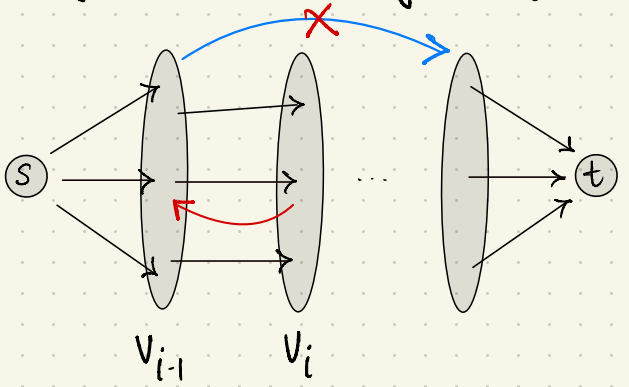
(b) $d(f)$ increases at least once per m iterations.



Consider the Second of these k iterations.

As argued in proof of part (a), any newly introduced edge after the first iteration goes from layer V_i to V_{i-1}.

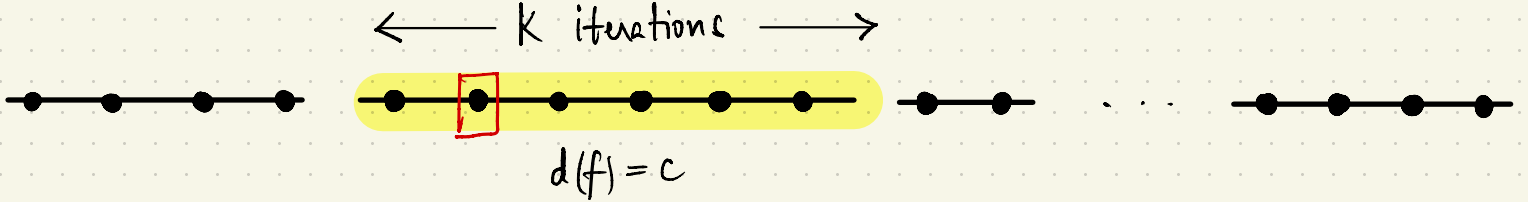
$\Rightarrow$ no shortcut edges



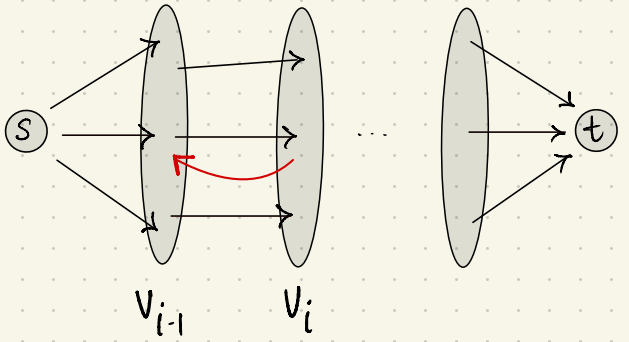
defined wrt saved copy in first iteration

PROOF OF KEY LEMMA

(b) $d(f)$ increases at least once per m iterations.

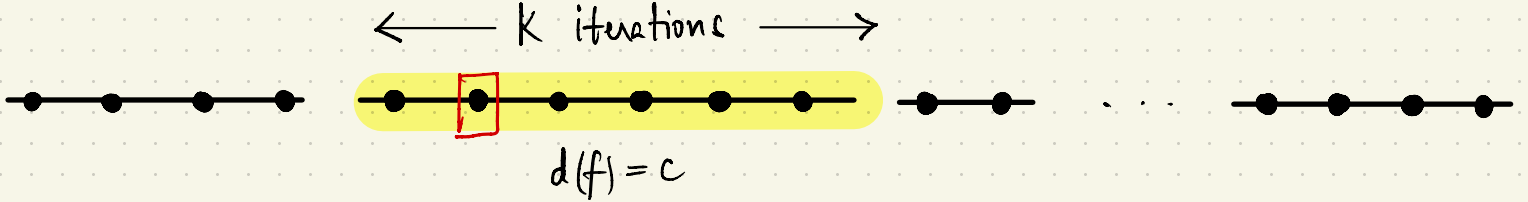


Recall: The shortest (s,t) path in second iteration is of length $d(f) = c$.



PROOF OF KEY LEMMA

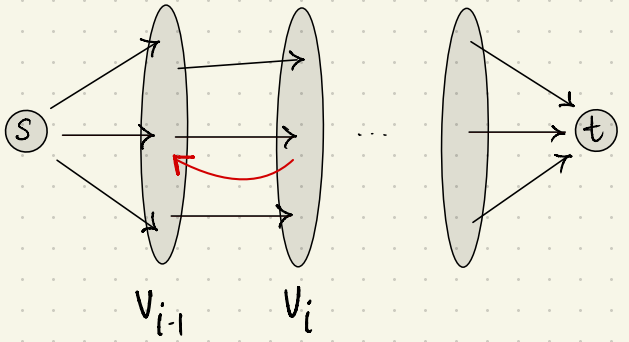
(b) $d(f)$ increases at least once per m iterations.



Recall: The shortest (s,t) path in second iteration is of length $d(f) = c$.

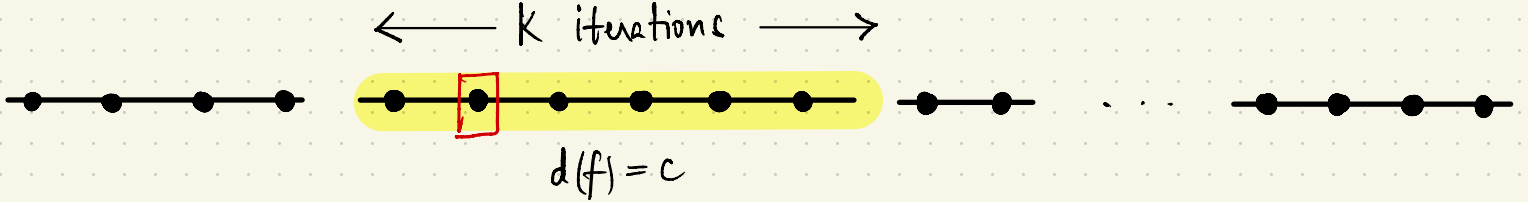
No shortcut edges $\Rightarrow$ any shortest (s,t) path must only use forward edges

with respect to saved copy of L_f



PROOF OF KEY LEMMA

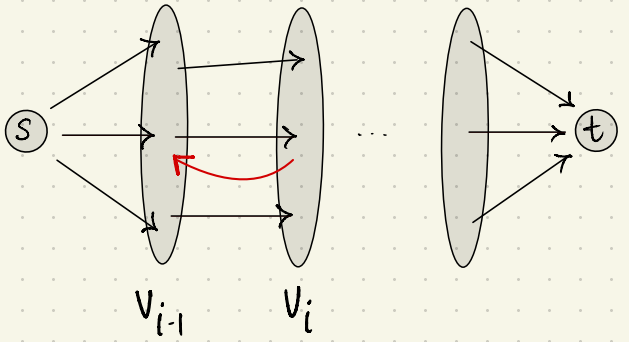
(b) d(f) increases at least once per m iterations.



Recall: The shortest (s,t) path in second iteration is of length d(f) = c.

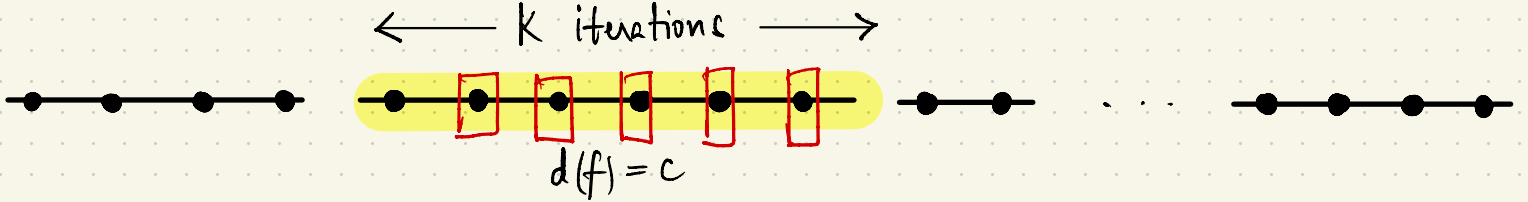
No shortcut edges ⇒ any shortest (s,t) path must only use forward edges with respect to saved copy of L_f

⇒ every shortest (s,t) path in second iteration is contained in the edge set of saved copy of L_f.

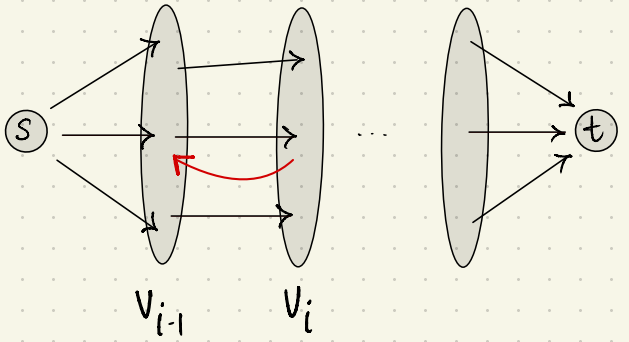


PROOF OF KEY LEMMA

(b) $d(f)$ increases at least once per m iterations.

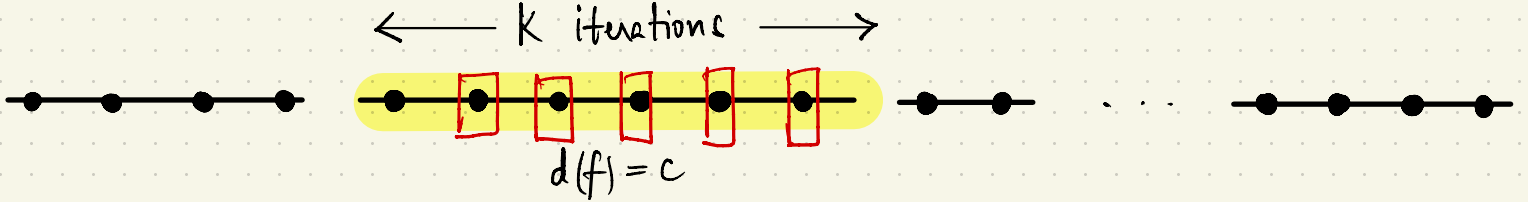


Inductively, for any subsequent iteration where $d(f) = c$, any shortest (s,t) path is contained in the saved copy L_f.



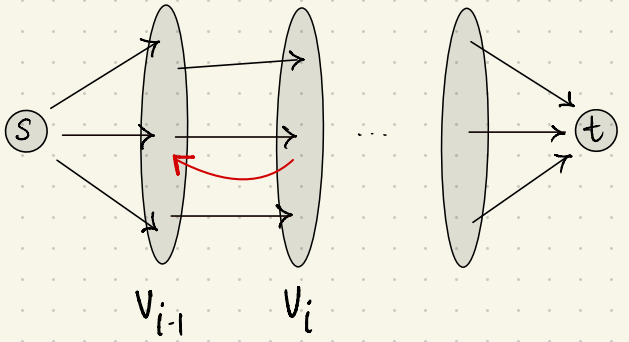
PROOF OF KEY LEMMA

(b) $d(f)$ increases at least once per m iterations.



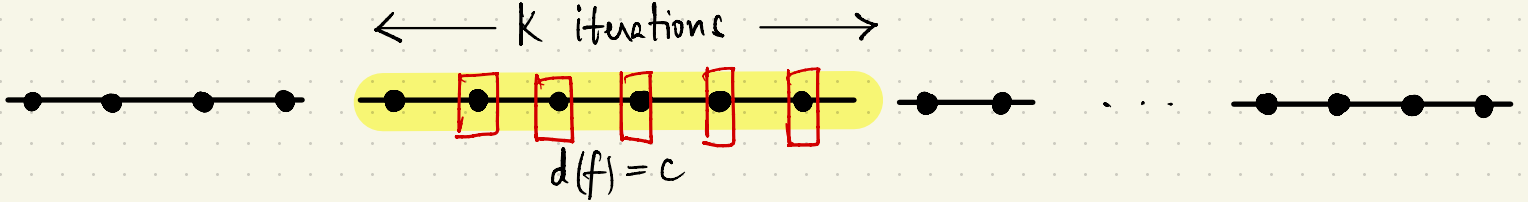
Inductively, for any subsequent iteration where $d(f) = c$, any shortest (s,t) path is contained in the saved copy L_f.

Edmonds-Karp invokes one of these paths in each iteration.

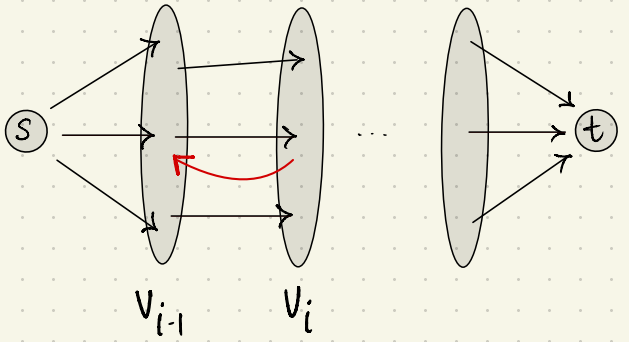


PROOF OF KEY LEMMA

(b) $d(f)$ increases at least once per m iterations.

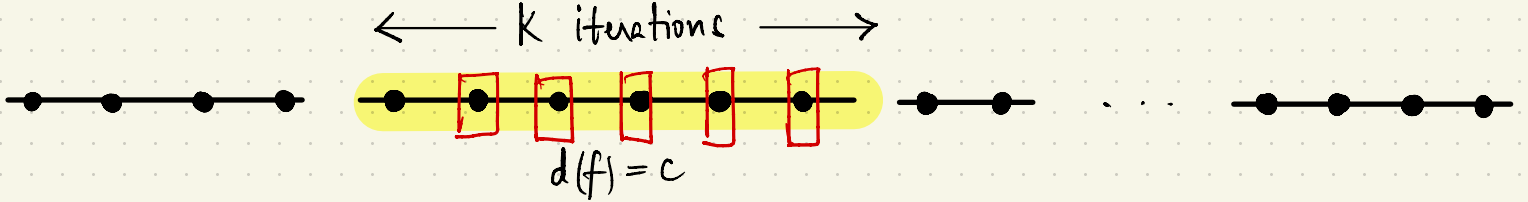


In each iteration, some forward edge in the chosen path is zeroed out.



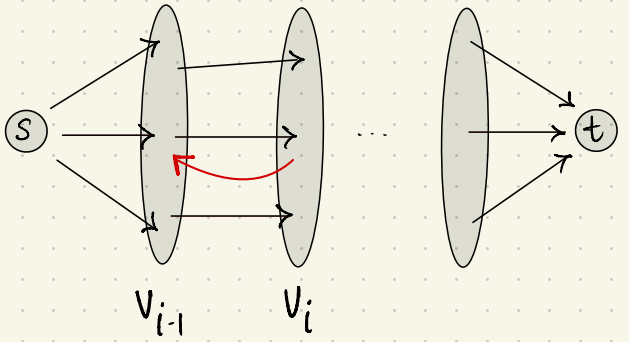
PROOF OF KEY LEMMA

(b) $d(f)$ increases at least once per m iterations.



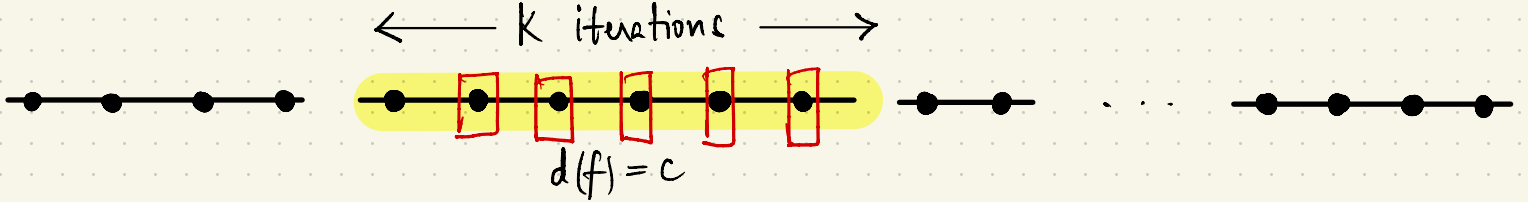
In each iteration, some forward edge in the chosen path is zeroed out.

$\Rightarrow$ the number of "available" edges for Edmonds-Karp decreases by 1 in every iteration.



PROOF OF KEY LEMMA

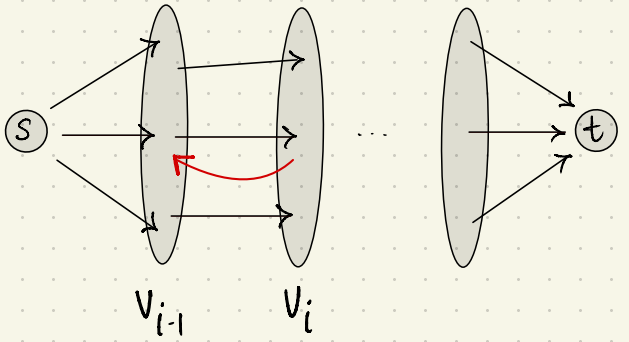
(b) $d(f)$ increases at least once per m iterations.



In each iteration, some forward edge in the chosen path is zeroed out.

$\Rightarrow$ the number of "available" edges for Edmonds-Karp decreases by 1 in every iteration.

$\Rightarrow$ # iterations $\leq m$.



PROOF OF KEY LEMMA

Key lemma: Fix a network G. For a flow f, define:

$$d(f) = \begin{cases} \text{the number of edges in a shortest } s\text{-}t \text{ path in } G_f \\ \text{with positive residual capacity} \\ +\infty \quad \text{if no such path exists.} \end{cases}$$

Then,

(a) $d(f)$ never decreases during the execution of the algorithm

(b) $d(f)$ increases at least once per m iterations.

PROOF OF KEY LEMMA

Key lemma : Fix a network G. For a flow f, define :

$$d(f) = \begin{cases} \text{the number of edges in a shortest } s\text{-}t \text{ path in } G_f \\ \text{with positive residual capacity} \\ \\ +\infty \quad \text{if no such path exists.} \end{cases}$$

Then,

(a) $d(f)$ never decreases during the execution of the algorithm

(b) $d(f)$ increases at least once per m iterations.

Corollary : Edmonds-Karp runs in $O(m^2 n)$ time.

Method	Year	Complexity	Description
Linear programming			Constraints given by the definition of a legal flow. See the linear program here.
Ford–Fulkerson algorithm	1955	$O(EU)$	As long as there is an open path through the residual graph, send the minimum of the residual capacities on that path.
Edmonds–Karp algorithm	1970	$O(VE^2)$	A specialization of Ford–Fulkerson, finding augmenting paths with breadth-first search.

Wikipedia entry on "Maximum Flow Problem"

Method	Year	Complexity	Description
Linear programming			Constraints given by the definition of a legal flow. See the linear program here.
Ford–Fulkerson algorithm	1955	$O(EU)$	As long as there is an open path through the residual graph, send the minimum of the residual capacities on that path.
Edmonds–Karp algorithm	1970	$O(VE^2)$	A specialization of Ford–Fulkerson, finding augmenting paths with breadth-first search.

⋮

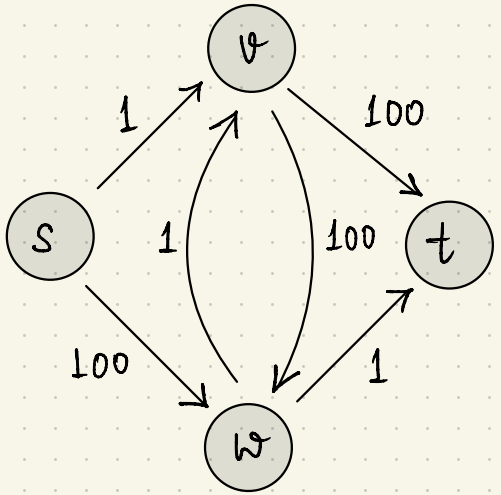
Wikipedia entry on "Maximum Flow Problem"

Chen, Kyng, Liu, Peng, Gutenberg and Sachdeva's algorithm[10]	2022	$O(E^{1+o(1)} \log U)$ The exact complexity is not stated clearly in the paper, but appears to be $E \exp O(\log^{7/8} E \log\log E) \log U$	Chen, Kyng, Liu, Peng, Gutenberg and Sachdeva's algorithm solves maximum flow and minimum-cost flow in almost linear time by building the flow through a sequence of $E^{1+o(1)}$ approximate undirected minimum-ratio cycles, each of which is computed and processed in amortized $E^{o(1)}$ time using a dynamic data structure.
Bernstein, Blikstad, Saranurak, Tu[26]	2024	$O(n^{2+o(1)} \log U)$ randomized algorithm when the edge capacities come from the set $\{1, \ldots, U\}$	The algorithm is a variant of the push-relabel algorithm by introducing the *weighted* variant. The paper establishes a weight function on directed and acyclic graphs (DAG), and attempts to imitate it on general graphs using directed expander hierarchies, which induce a natural vertex ordering that produces the weight function similar to that of the DAG special case. The randomization aspect (and subsequently, the $n^{o(1)}$ factor) comes from the difficulty in applying directed expander hierarchies to the computation of *sparse cuts*, which do not allow for natural dynamic updating.

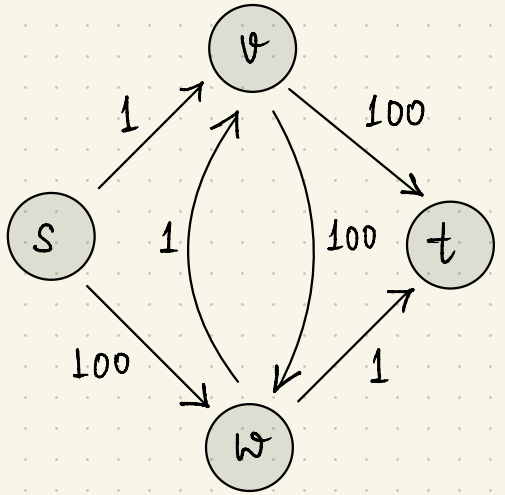
APPLICATIONS

1. MINIMUM CUT PROBLEM

MINIMUM CUT PROBLEM

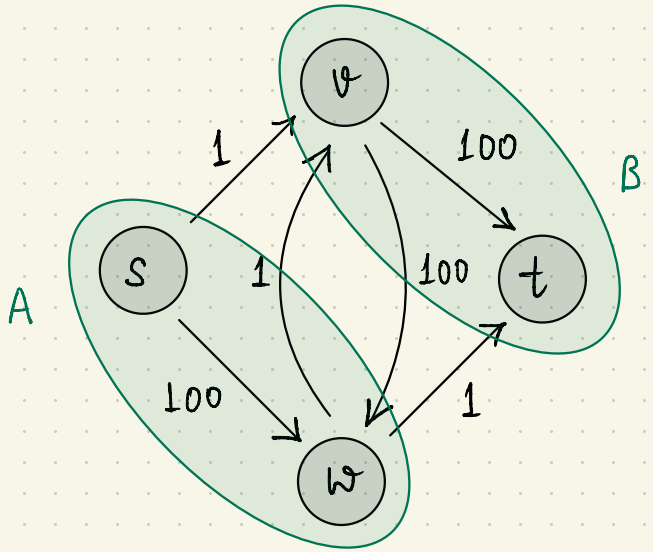


MINIMUM CUT PROBLEM



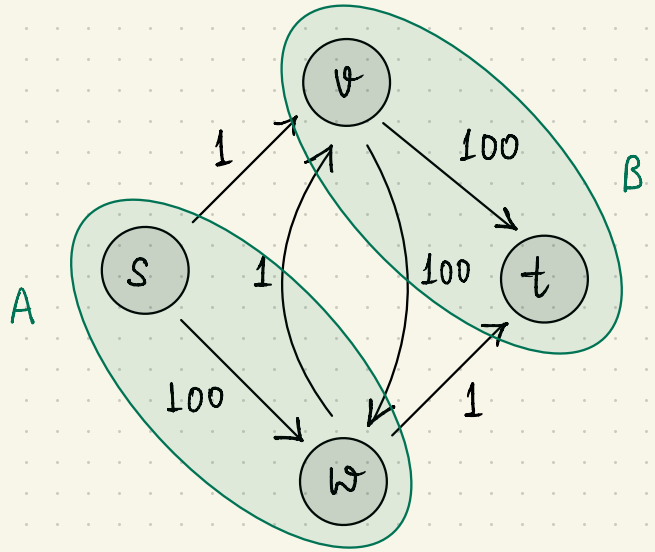
Goal: compute (s,t)-cut (A,B) minimizing $\sum_{e \in \delta^+(A)} u_e$.

MINIMUM CUT PROBLEM



Goal: compute (s,t)-cut (A,B) minimizing $\sum_{e \in \delta^+(A)} u_e$.

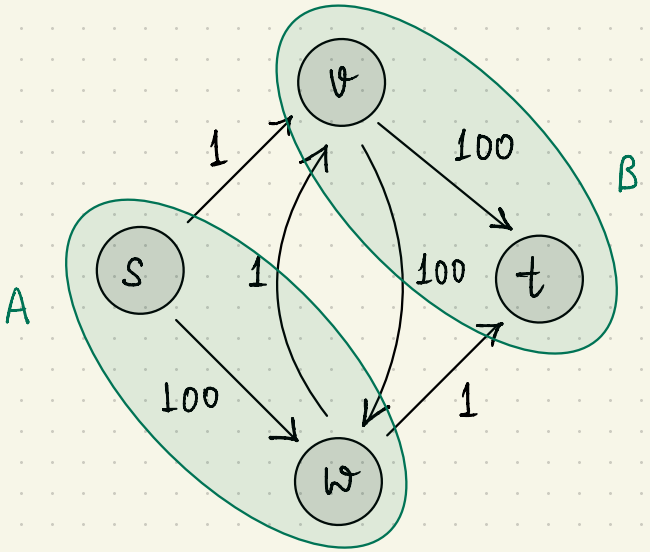
MINIMUM CUT PROBLEM



Goal: compute (s,t)-cut (A,B) minimizing $\sum_{e \in \delta^+(A)} u_e$.

Recall: (s,t)-minimum cut problem reduces to max flow in linear time.

MINIMUM CUT PROBLEM



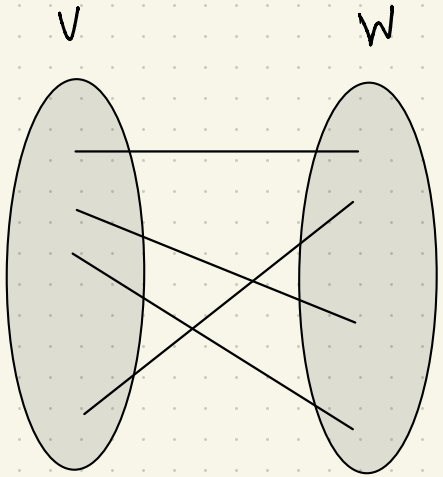
Application in image segmentation (Additional reading)

2. BIPARTITE MATCHING

BIPARTITE MATCHING

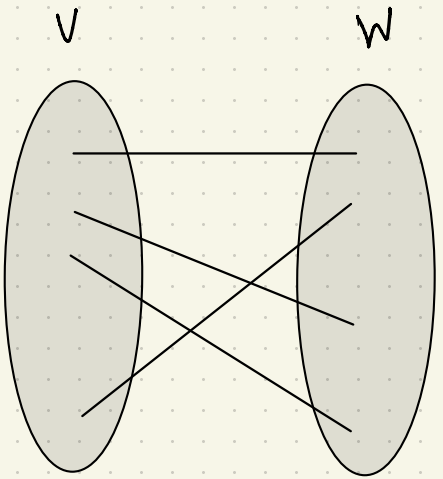
BIPARTITE MATCHING

input : an undirected bipartite graph $G = (V \cup W, E)$.



BIPARTITE MATCHING

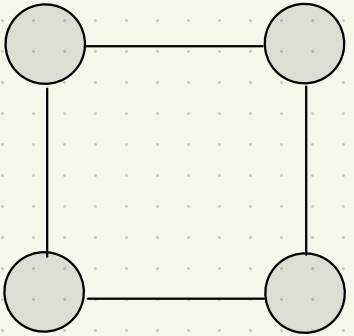
input : an undirected bipartite graph
$G = (V \cup W, E)$.



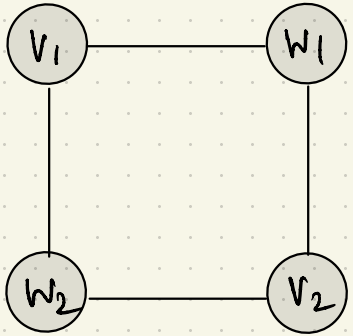
goal : a maximum cardinality matching

subset of edges
with no shared endpoints

BIPARTITE MATCHING

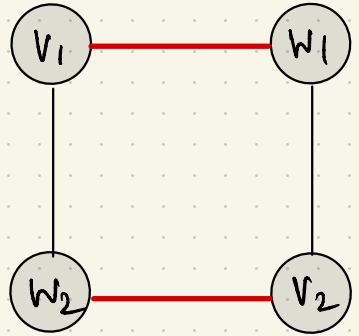


BIPARTITE MATCHING

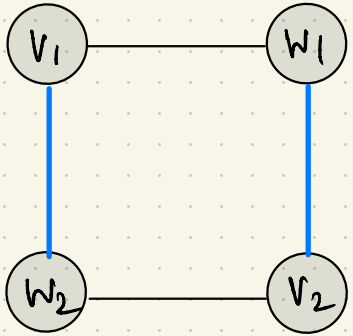


Bipartite

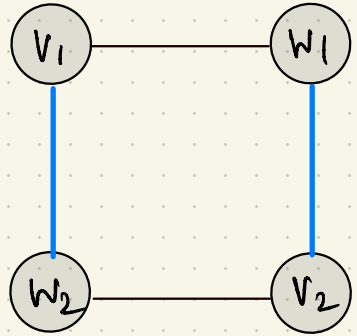
BIPARTITE MATCHING



BIPARTITE MATCHING

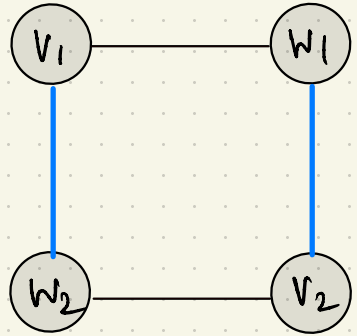


BIPARTITE MATCHING



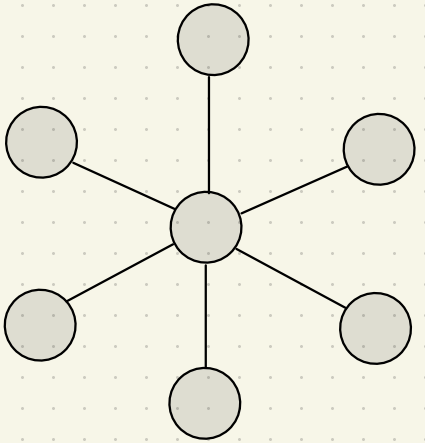
Size of maximum matching = 2

BIPARTITE MATCHING



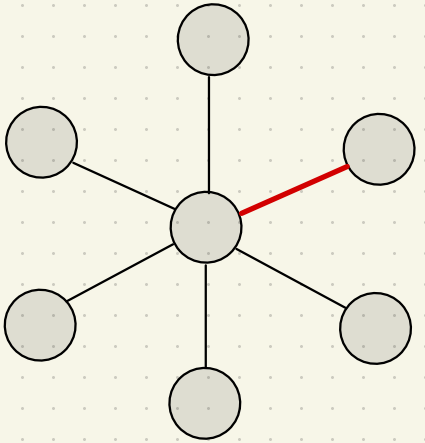
Size of maximum matching $= 2 = \frac{|V|}{2}$

BIPARTITE MATCHING



Size of maximum matching = ?

BIPARTITE MATCHING



Size of maximum matching = $1 << \frac{|V|}{2}$

BIPARTITE MATCHING

Many, many applications:

- job applicants and positions
- courses and slots (room + time)
- organ donors and patients

⋮

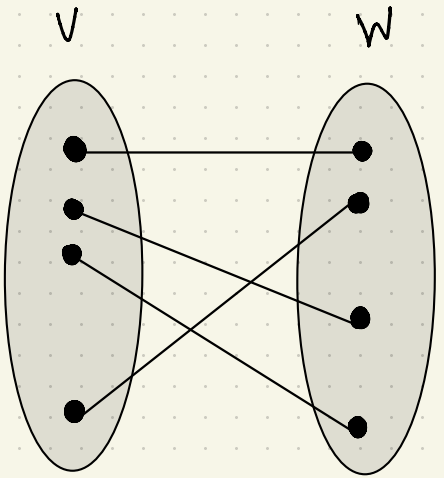
BIPARTITE MATCHING

Claim: Bipartite matching reduces to maximum flow in linear time.

BIPARTITE MATCHING

Claim: Bipartite matching reduces to maximum flow in linear time.

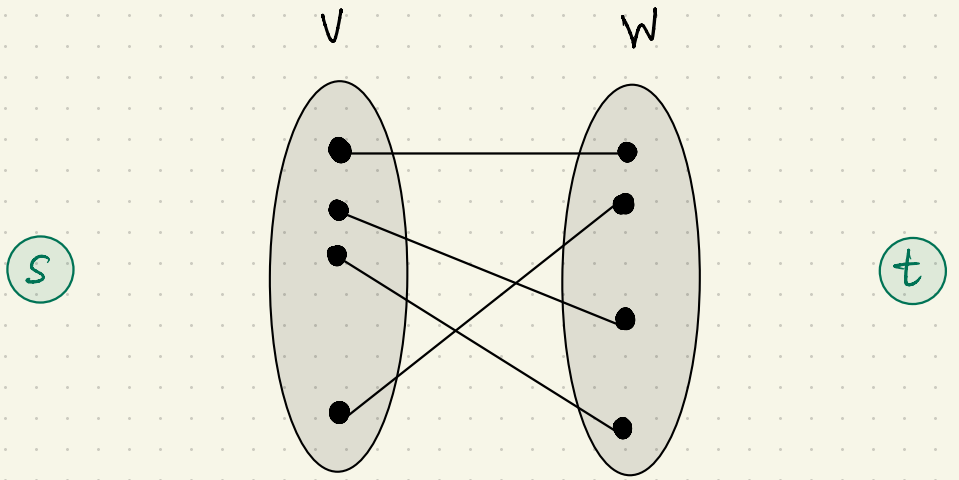
Proof:



BIPARTITE MATCHING

Claim: Bipartite matching reduces to maximum flow in linear time.

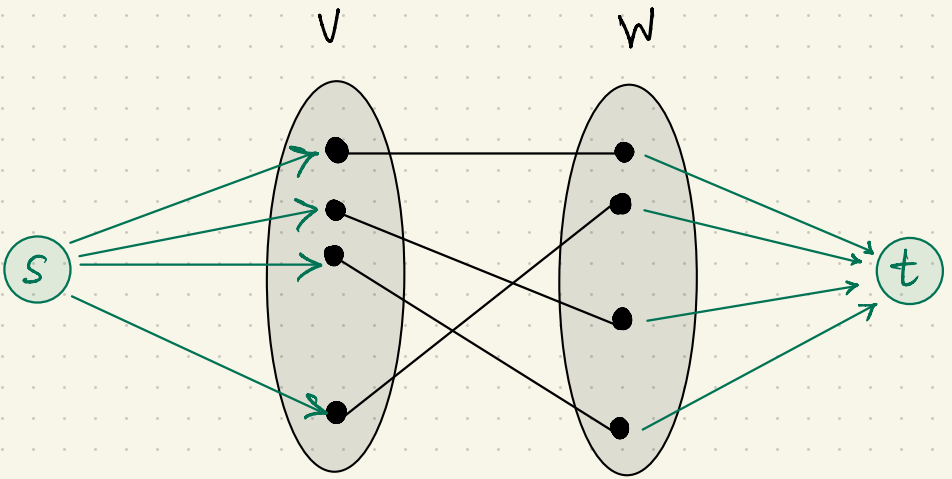
Proof:



BIPARTITE MATCHING

Claim: Bipartite matching reduces to maximum flow in linear time.

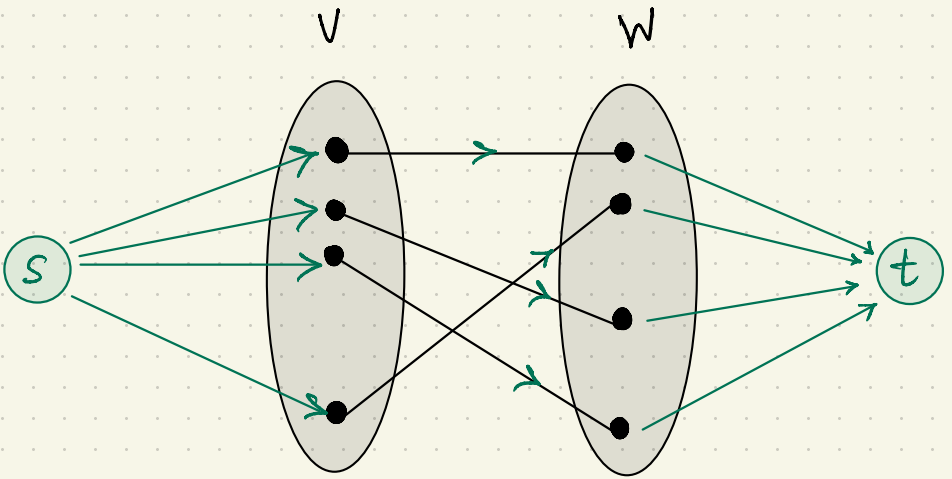
Proof:



BIPARTITE MATCHING

Claim: Bipartite matching reduces to maximum flow in linear time.

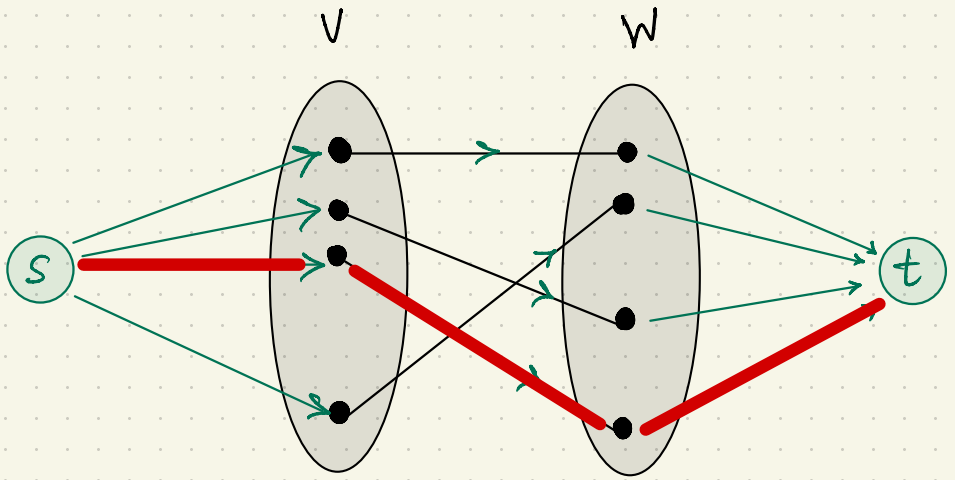
Proof:



BIPARTITE MATCHING

Claim: Bipartite matching reduces to maximum flow in linear time.

Proof:

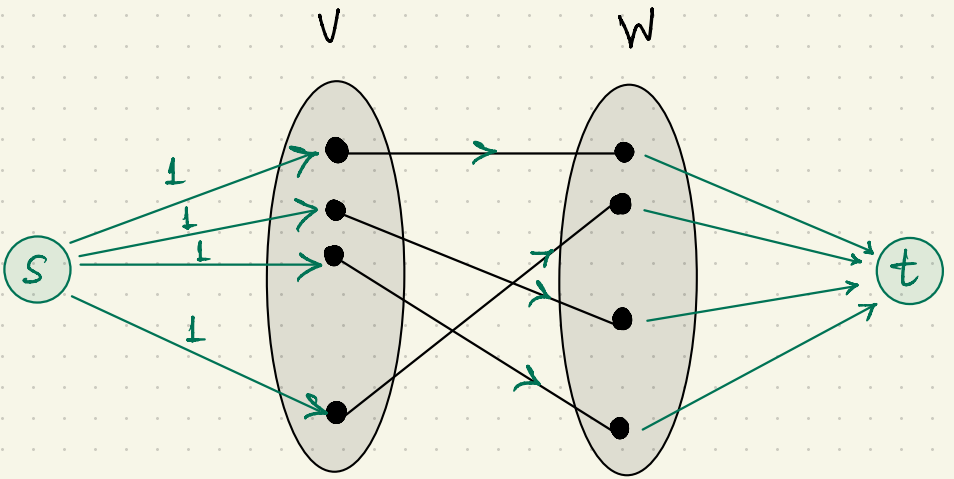


NOTE: Layered graph $\Rightarrow$ every (s,t) flow path includes exactly one edge of given bipartite graph

BIPARTITE MATCHING

Claim: Bipartite matching reduces to maximum flow in linear time.

Proof:

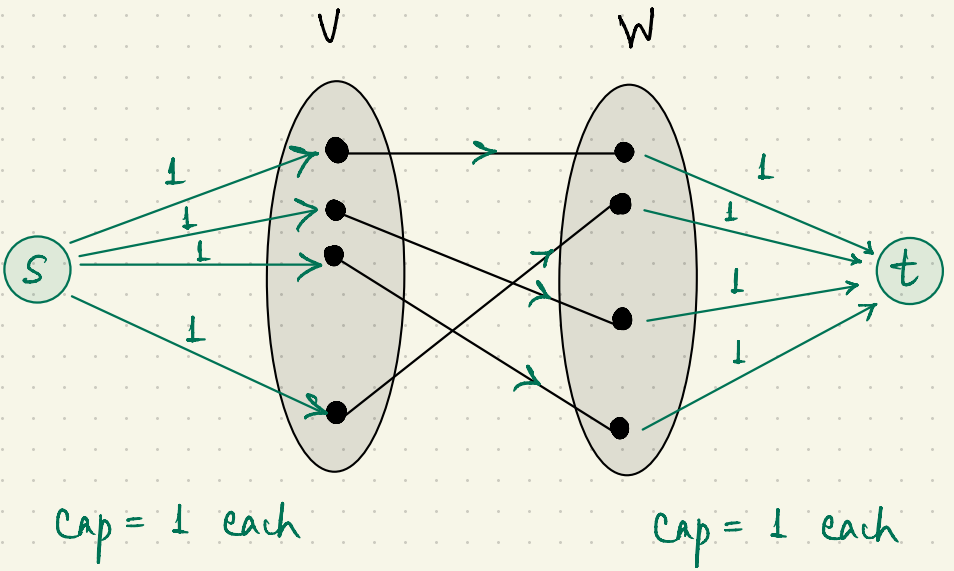


Cap = 1 each

BIPARTITE MATCHING

Claim: Bipartite matching reduces to maximum flow in linear time.

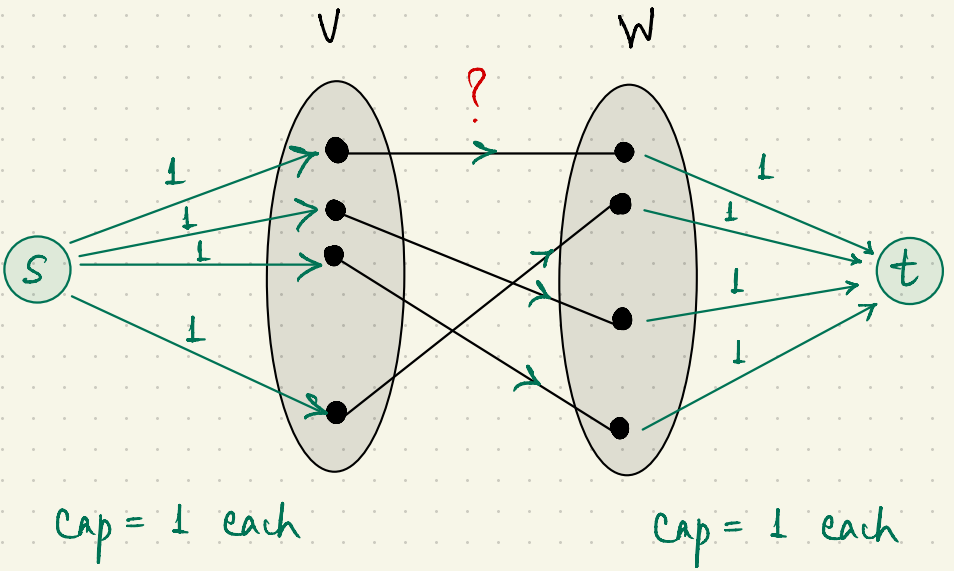
Proof:



BIPARTITE MATCHING

Claim: Bipartite matching reduces to maximum flow in linear time.

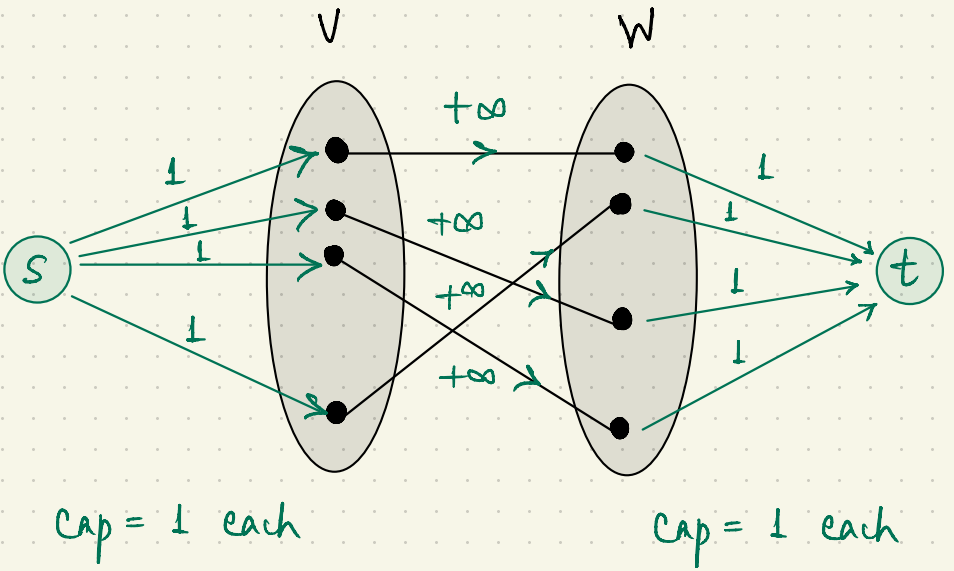
Proof:



BIPARTITE MATCHING

Claim: Bipartite matching reduces to maximum flow in linear time.

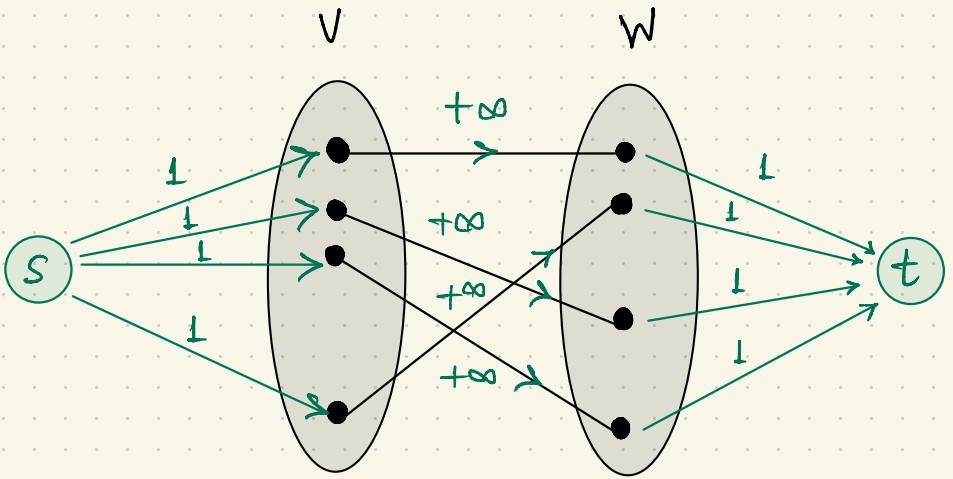
Proof:



BIPARTITE MATCHING

Claim: Bipartite matching reduces to maximum flow in linear time.

Proof:



Cap = 1 each Cap = 1 each

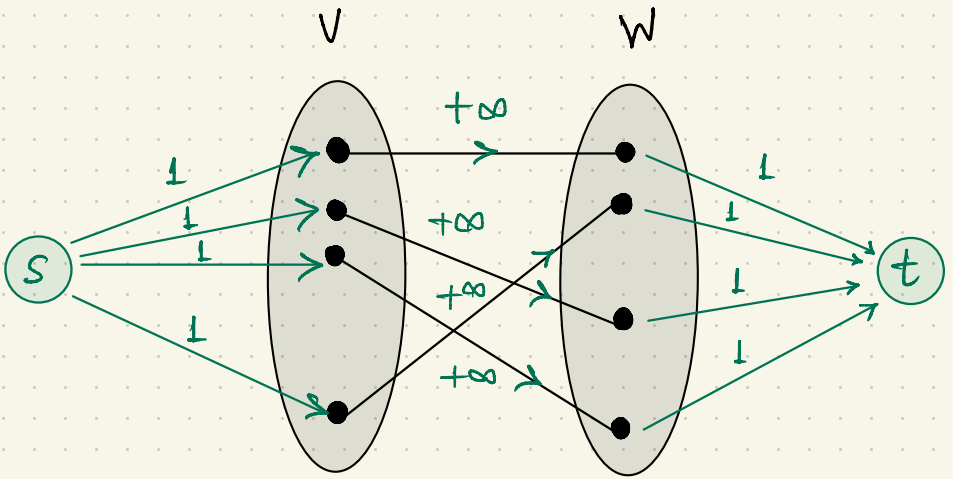
Bipartite matching $\longleftrightarrow$ Integral Flow (Check!)

Size of matching = Value of flow

BIPARTITE MATCHING

Claim: Bipartite matching reduces to maximum flow in linear time.

Proof:



Cap = 1 each Cap = 1 each

Bipartite matching ⟷ Integral Flow (Check!)

Size of matching = Value of flow

achieved by Ford-Fulkerson and Edmonds-Karp