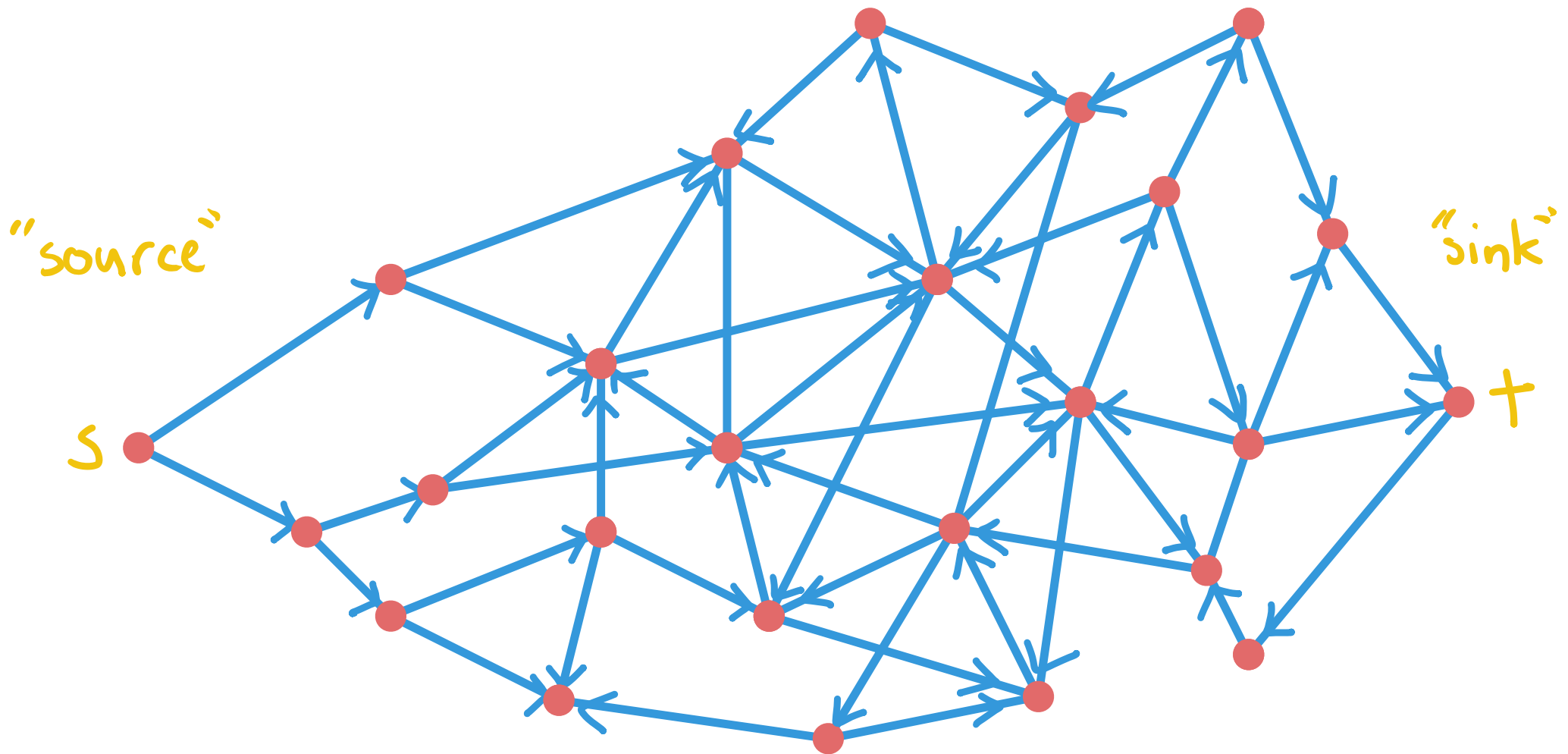
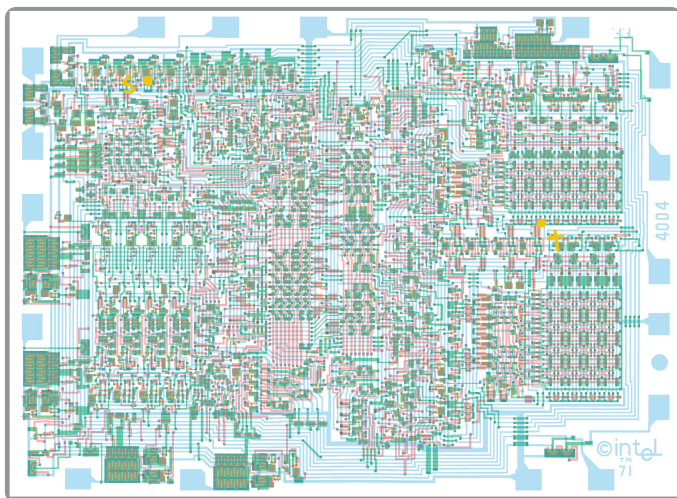
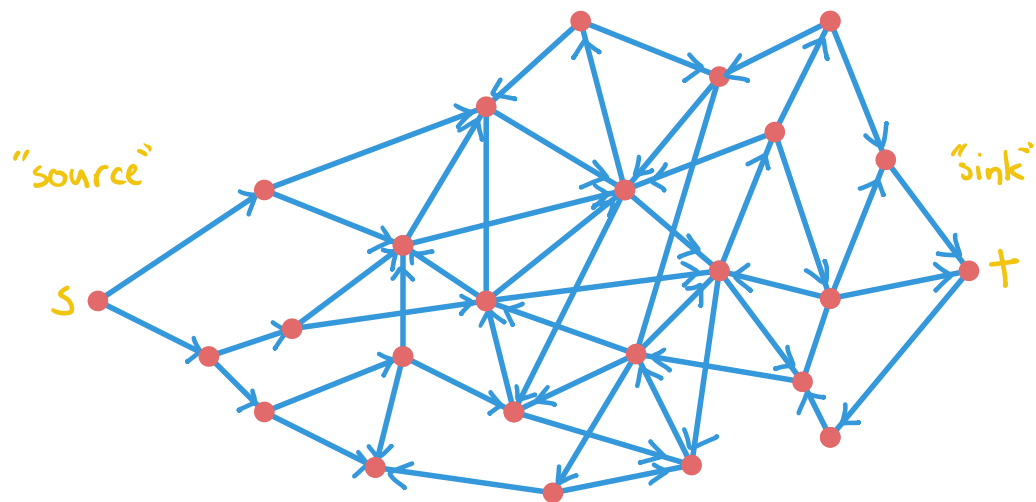


Network Flow

Measuring connectivity

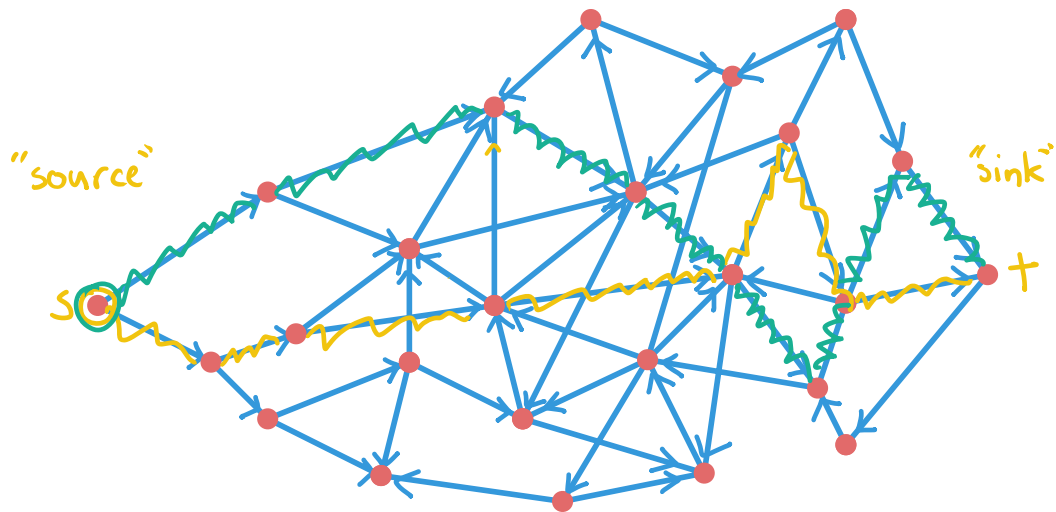


how well connected are s and t?

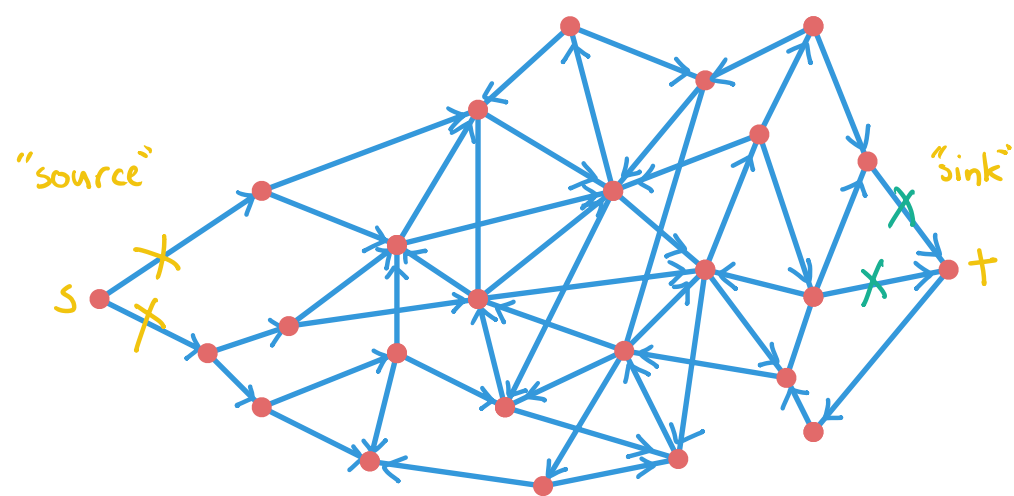


- ① Is there an $s \rightsquigarrow t$ path?
- ② How many distinct $s \rightsquigarrow t$ paths?
- ③ how many k -edge $s \rightsquigarrow t$ walks?
(k fixed)
- ④ length of shortest $s \rightsquigarrow t$ path?
- ⑤ how many shortest $s \rightsquigarrow t$ paths?
- ⑥ In a random walk starting from s , what is the expected # of steps until we reach t ?
- ⑦ max # of edge-disjoint $s \rightsquigarrow t$ paths?
- ⑧ min # of edge deletions to disconnect $s \rightsquigarrow t$?

Measuring connectivity

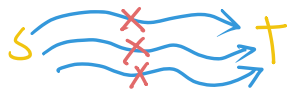


⑦ max # of edge-disjoint $s \rightarrow t$ paths?



⑧ min # of edge deletions to disconnect $s \rightarrow t$?

path $p \xrightarrow{\text{packs into}} \text{edge } e \in P$
 $\xleftarrow{\text{Covers}}$



path $p \xrightarrow{\text{packs into}} \text{edge } e \in P$
 $\xleftarrow{\text{Covers}}$

Lemma

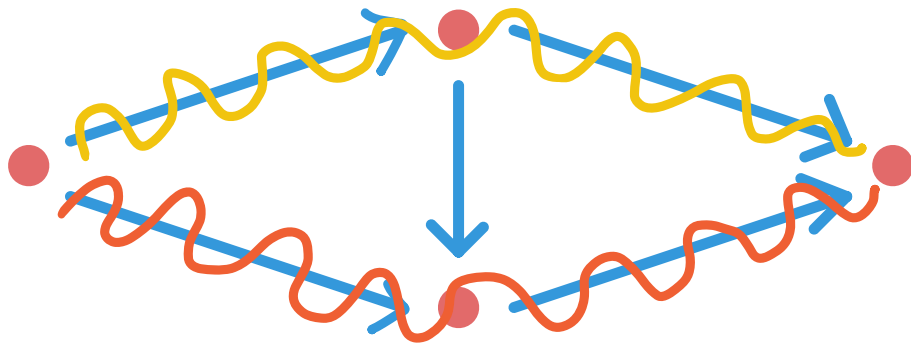
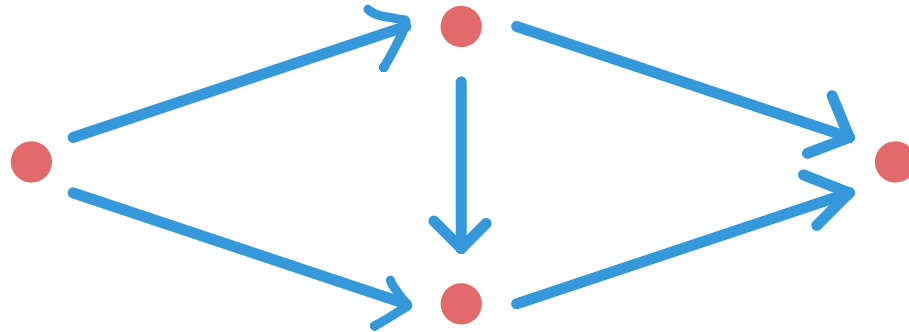
$$\left(\begin{array}{c} \text{max \#} \\ \text{disjoint} \\ s \rightsquigarrow t \text{ paths} \end{array} \right) \leq \left(\begin{array}{c} \text{min \# edges} \\ \text{in any } s \rightsquigarrow t \text{ cut} \end{array} \right)$$

* 2nd example of
 packing $\leq$ covering

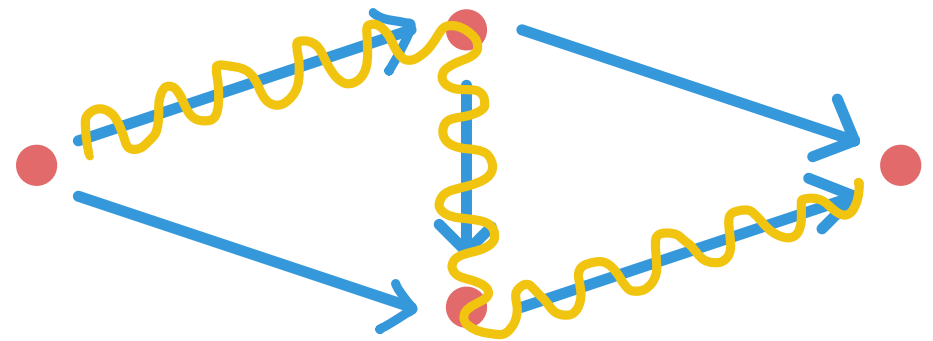
Iterative/greedy

① while there is a path $p: s \rightsquigarrow t$

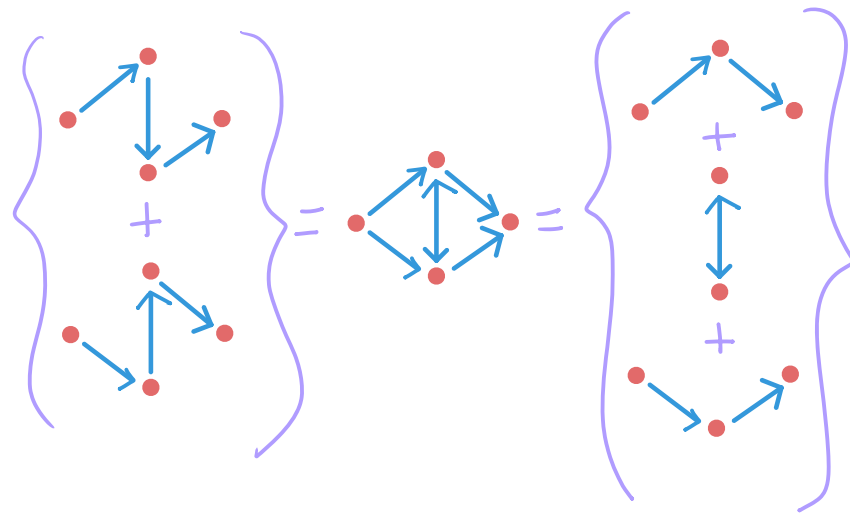
@ take p and remove all its edges from the graph



(good)



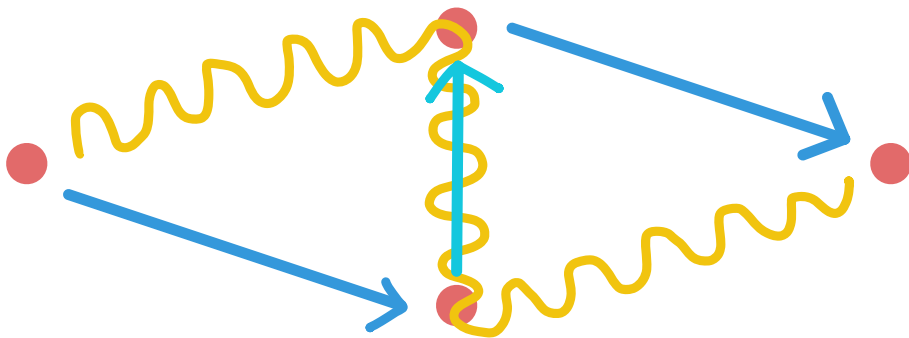
(bad)



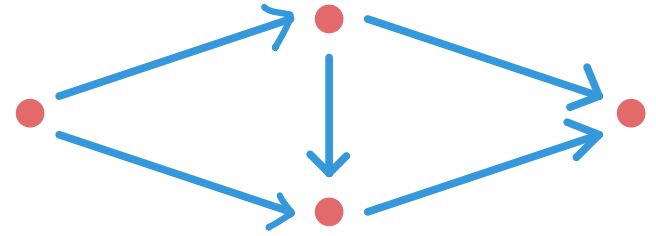
"opposite edges"



"cancel out"



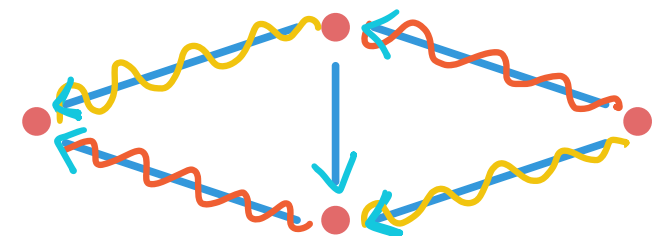
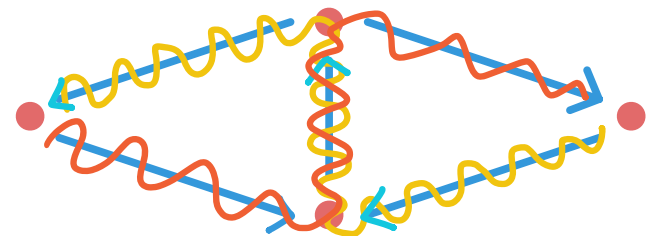
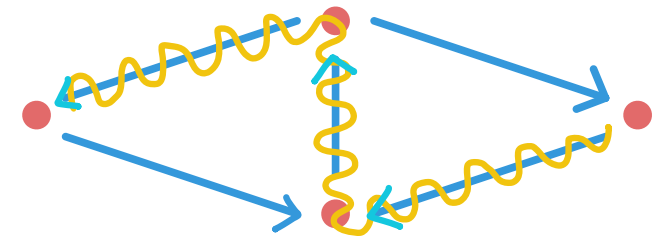
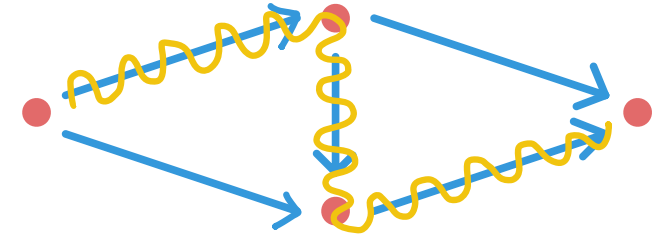
we should be allowed
to take reverse edges
in subsequent paths



augmenting-paths($G = (V, E), s, t$)

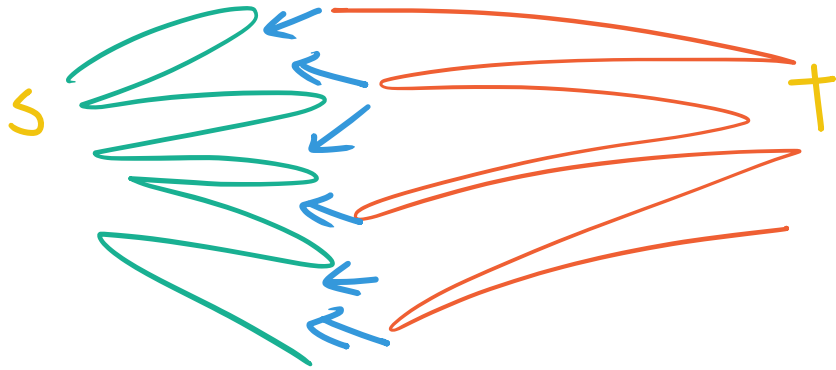
/ Ford and Fulkerson [FF56]*

1. $P \leftarrow \emptyset$
2. While there is an (s, t) -path $p : s \rightsquigarrow t$:
 - A. Remove all of p 's edges from the graph.
 - B. Add the reverse of all of p 's edges to the graph.
 - C. Uncross p with previously selected paths, and add p to P .



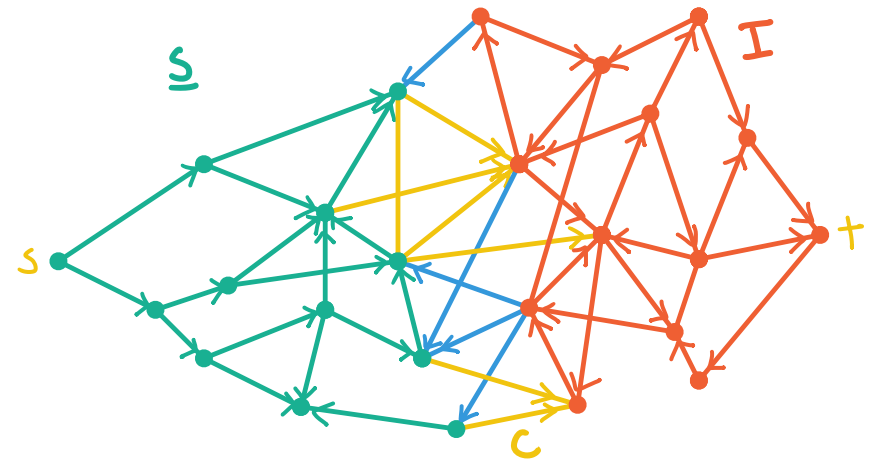
Proving optimality

algorithm terminates w/ $s \rightarrow t$ disconnected in residual graph



let $\underline{S}$ = vertices reachable from s in residual graph
 $\underline{I} = V \setminus \underline{S}$

$C = \delta^+(\underline{S})$ = edges from $\underline{S}$ to $\underline{I}$



over the course of the algorithm:

- ① each $s \rightarrow t$ path has (exactly) 1 more edge from $\underline{S}$ to $\underline{I}$ than from $\underline{I}$ to $\underline{S}$
- ② each $s \rightarrow t$ path decreases $|C|$ by 1
- ③ At the end, $|C| = 0$.

$$\Rightarrow (\text{\# paths packed}) = (\text{\# edges in } C)$$

$$(\text{\# paths packed}) = (\text{max \# disjoint } s \rightarrow t \text{ paths}) = (\text{min \# edges in any } s \rightarrow t \text{ cut}) = (\text{\# edges in } C)$$

equal

$$(\# \text{ paths packed}) \stackrel{=}{=} \left(\begin{array}{c} \text{max \#} \\ \text{disjoint} \\ s \rightarrow t \text{ paths} \end{array} \right) \stackrel{=}{=} \left(\begin{array}{c} \text{min \# edges} \\ \text{in any } s \rightarrow t \text{ cut} \end{array} \right) \stackrel{=}{=} \left(\begin{array}{c} \text{\# edges} \\ \text{in } C \end{array} \right)$$

Theorems

$$\textcircled{1} \left(\begin{array}{c} \text{max \#} \\ \text{disjoint} \\ s \rightarrow t \text{ paths} \end{array} \right) = \left(\begin{array}{c} \text{min \# edges} \\ \text{in any } s \rightarrow t \text{ cut} \end{array} \right)$$

$$\textcircled{2} (\# \text{ paths packed}) = \left(\begin{array}{c} \text{max \#} \\ \text{disjoint} \\ s \rightarrow t \text{ paths} \end{array} \right)$$

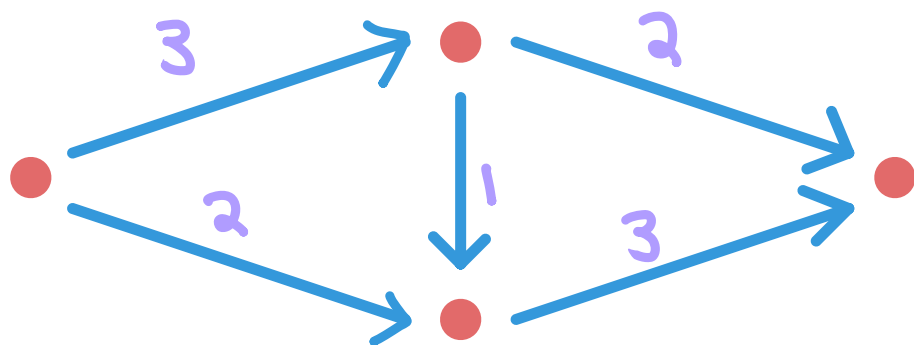
$$\textcircled{3} \left(\begin{array}{c} \text{min \# edges} \\ \text{in any } s \rightarrow t \text{ cut} \end{array} \right) = \left(\begin{array}{c} \text{\# edges} \\ \text{in } C \end{array} \right)$$

1. $P \leftarrow \emptyset$
2. While there is an (s, t) -path $p: s \rightsquigarrow t$:
 - A. Remove all of p 's edges from the graph.
 - B. Add the reverse of all of p 's edges to the graph.
 - C. Uncross p with previously selected paths, and add p to P .³
3. $S \leftarrow \{\text{vertices still reachable from } s\}$.
4. Return the (s, t) -path packing P and the (s, t) -cut $\partial^+(S)$.

*/

Theorem 14.5 ([FF56]). Let $G = (V, E)$ be a directed graph with m edges, and let $s, t \in V$ be distinct vertices. Let λ be the size of the max-flow/min-cut from s to t . Then $\text{augmenting-paths}(G, s, t)$ produces both a maximum packing and a minimum cut in $O(m\lambda)$ time.

Capacities



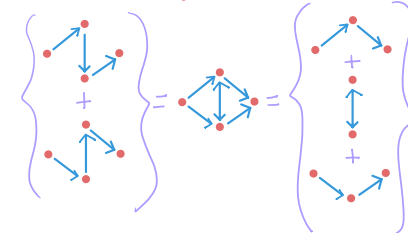
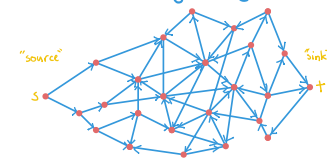
capacity $c(e) > 0$: reuse edge e up to $c(e)$ times

- if $c(e)$ large, Augmenting-Paths not polynomial time

- if $c(e)$ not integral (and just real-valued)
what about max fractional path packing

How to compute

⑦ max # of edge-disjoint $s \rightsquigarrow t$ paths?



Lemma let p, q be edge-disjoint $s \rightsquigarrow t$ paths



then there is p', q' : $s \rightsquigarrow t$ disjoint without opposite pairs of edges



$$\textcircled{1} \left(\begin{array}{l} \text{max \#} \\ \text{disjoint} \\ s \rightsquigarrow t \text{ paths} \end{array} \right) = \left(\begin{array}{l} \text{min \#} \\ \text{edges in any} \\ s \rightsquigarrow t \text{ cut} \end{array} \right)$$

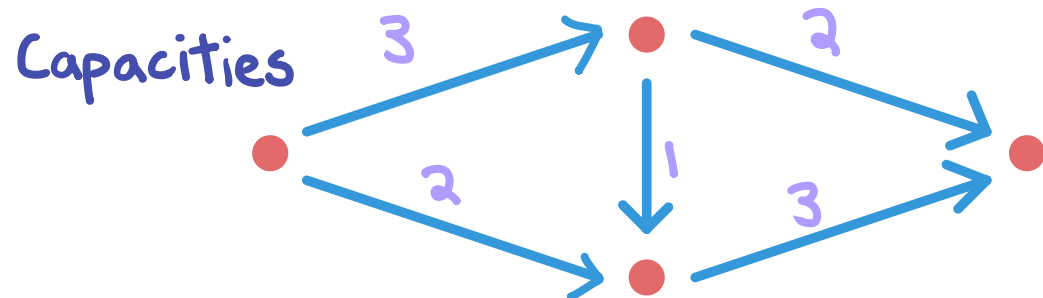
$$\textcircled{2} \left(\begin{array}{l} \text{\# paths} \\ \text{packed} \end{array} \right) = \left(\begin{array}{l} \text{max \#} \\ \text{disjoint} \\ s \rightsquigarrow t \text{ paths} \end{array} \right)$$

$$\textcircled{3} \left(\begin{array}{l} \text{min \#} \\ \text{edges in any} \\ s \rightsquigarrow t \text{ cut} \end{array} \right) = \left(\begin{array}{l} \text{\# edges} \\ \text{initially in } C \end{array} \right)$$

1. $P \leftarrow \emptyset$
2. While there is an (s, t) -path $p: s \rightsquigarrow t$:
 - A. Remove all of p 's edges from the graph.
 - B. Add the reverse of all of p 's edges to the graph.
 - C. Uncross p with previously selected paths, and add p to P .³
3. $S \leftarrow \{\text{vertices still reachable from } s\}$.
4. Return the (s, t) -path packing P and the (s, t) -cut $\partial^+(S)$.

*/

Theorem 14.5 ([FF56]). Let $G = (V, E)$ be a directed graph with m edges, and let $s, t \in V$ be distinct vertices. Let λ be the size of the max-flow/min-cut from s to t . Then $\text{augmenting-paths}(G, s, t)$ produces both a maximum packing and a minimum cut in $O(m\lambda)$ time.



capacity $c(e) > 0$: reuse edge e up to $c(e)$ times

• if $c(e)$ large, Augmenting-Paths not polynomial time

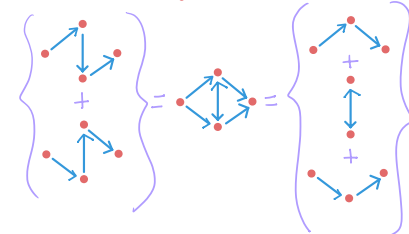
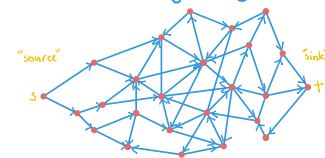
• if $c(e)$ not integral (and just real-valued) what about max fractional path packing

Good news:

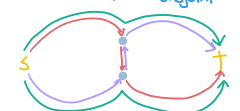
Our duality theorems still hold

How to compute

⑦ max # of edge-disjoint $s \rightsquigarrow t$ paths?



Lemma let p, q be edge-disjoint $s \rightsquigarrow t$ paths



then there is $p', q': s \rightsquigarrow t$ disjoint without opposite pairs of edges



$$\textcircled{1} \left(\begin{array}{c} \text{max \#} \\ \text{disjoint} \\ s \rightsquigarrow t \text{ paths} \end{array} \right) = \left(\begin{array}{c} \text{min \#} \\ \text{edges in any} \\ s \rightsquigarrow t \text{ cut} \end{array} \right)$$

$$\textcircled{2} \left(\begin{array}{c} \# \text{ paths} \\ \text{packed} \end{array} \right) = \left(\begin{array}{c} \text{max \#} \\ \text{disjoint} \\ s \rightsquigarrow t \text{ paths} \end{array} \right)$$

$$\textcircled{3} \left(\begin{array}{c} \text{min \#} \\ \text{edges in any} \\ s \rightsquigarrow t \text{ cut} \end{array} \right) = \left(\begin{array}{c} \# \text{ edges} \\ \text{initially in } C \end{array} \right)$$

remaining issues algorithmic ~ next time ~

Max Flow "liquid" version of path packing

Input $G=(V,E)$ directed capacities $c:E \rightarrow \mathbb{R}_{>0}$

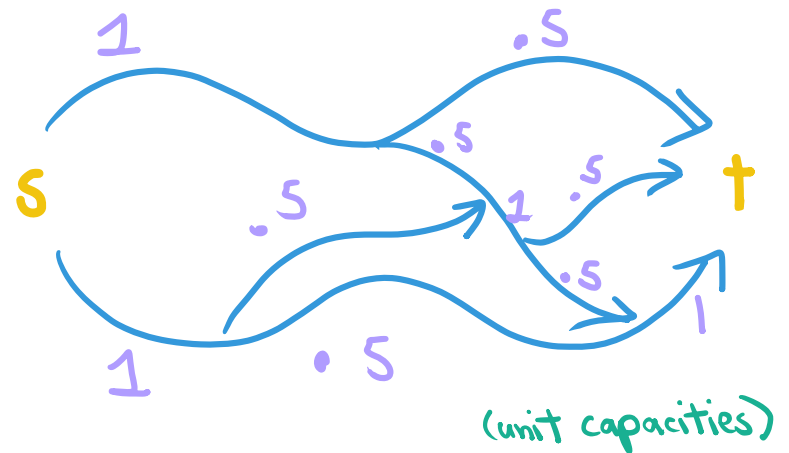
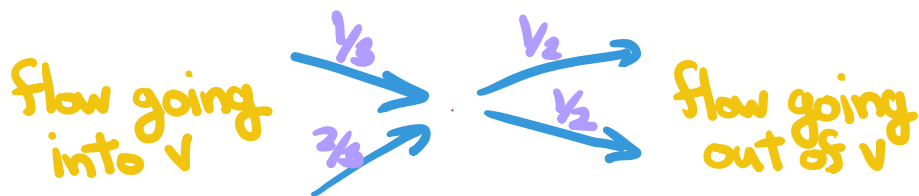
source $s \in V$, sink $t \in V$

"s-t flow" $f:E \rightarrow \mathbb{R}_{\geq 0}$ s.t.

(a) "capacity constraints"
 $0 \leq f(e) \leq c(e) \quad \forall e \in E$

(b) "conservation of flow"

$$\sum_{(u,v) \in E} f(u,v) = \sum_{(v,w) \in E} f(v,w) \quad \forall v \notin \{s,t\}$$



(b) "conservation of flow"

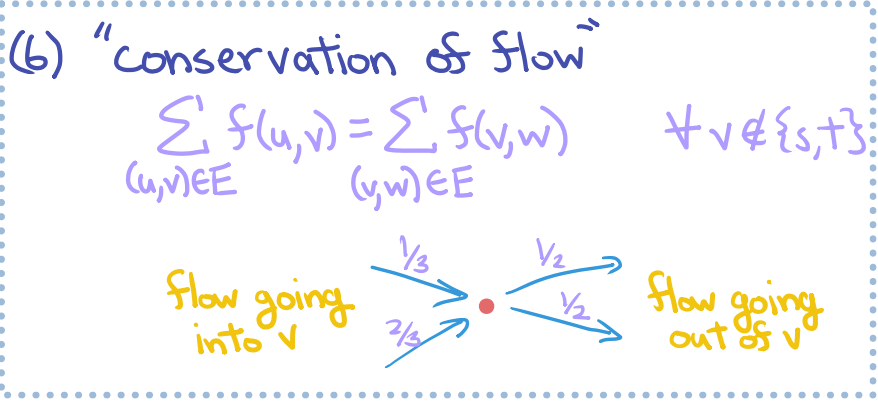
$$\sum_{(u,v) \in E} f(u,v) = \sum_{(v,w) \in E} f(v,w) \quad \forall v \notin \{s,t\}$$

flow going into v flow going out of v

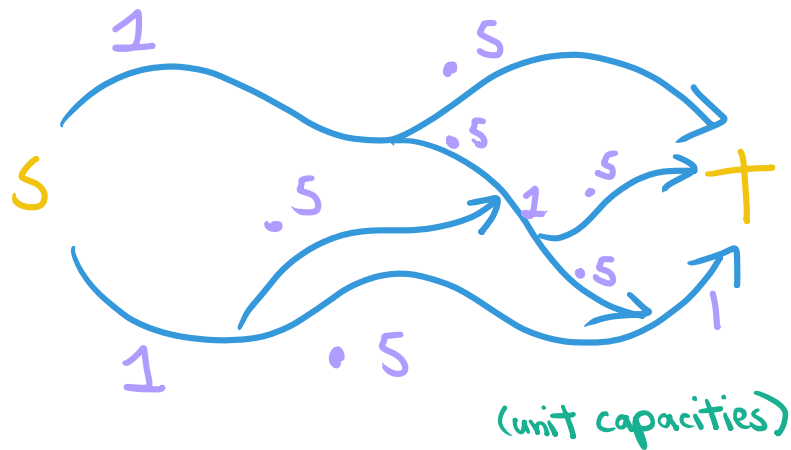
(b) "conservation of flow"

$$\sum_{(u,v) \in E} f(u,v) = \sum_{(v,w) \in E} f(v,w) \quad \forall v \notin \{s,t\}$$

flow going into v flow going out of v



net flow out of s = net flow into t



"net flow from s to t",
"size of flow"

Max flow problem

find $s \rightarrow t$ flow of maximum size

"linear program"

max



over $f \in \mathbb{R}^E$

"capacity constraints" s.t. $0 \leq f(e) \leq c(e)$

$\forall e \in E$

"conservation of flow"

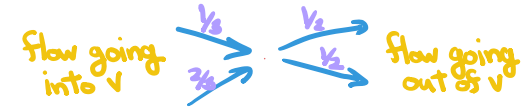
$$\sum_{(u,v) \in E} f(u,v) = \sum_{(v,w) \in E} f(v,w)$$

$\forall v \notin \{s, t\}$

"s-t flow" $f: E \rightarrow \mathbb{R}_{\geq 0}$ s.t.

(a) "capacity constraints"
 $0 \leq f(e) \leq c(e) \quad \forall e \in E$

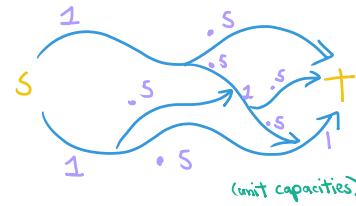
(b) "conservation of flow"
 $\sum_{(u,v) \in E} f(u,v) = \sum_{(v,w) \in E} f(v,w) \quad \forall v \notin \{s, t\}$



"linear program" because objective & constraints
all linear functions of f

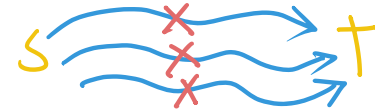
How does

Max Flow

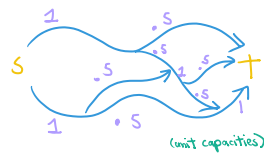


relate to

max path packings, min cut?

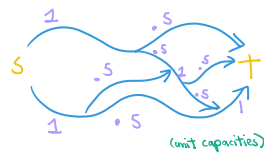



max path packings vs


Max Flow

s

t


 max path packings vs Max Flow
 

let $x: P_{s,t} \rightarrow \mathbb{R}_{\geq 0}$ encode a path packing

define $f: E \rightarrow \mathbb{R}_{\geq 0}$ as:

$$f(e) = \sum_{p: e \in p} x_p$$

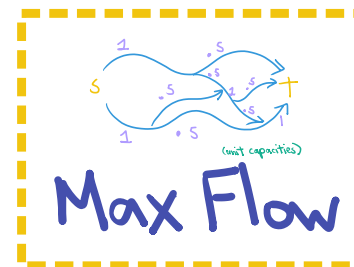
(flow = usage)

then f is feasible flow of same size

$\Rightarrow$



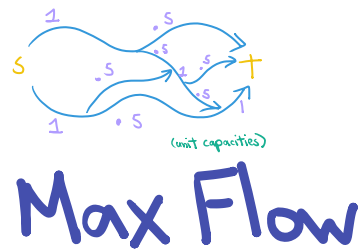
$\leq$



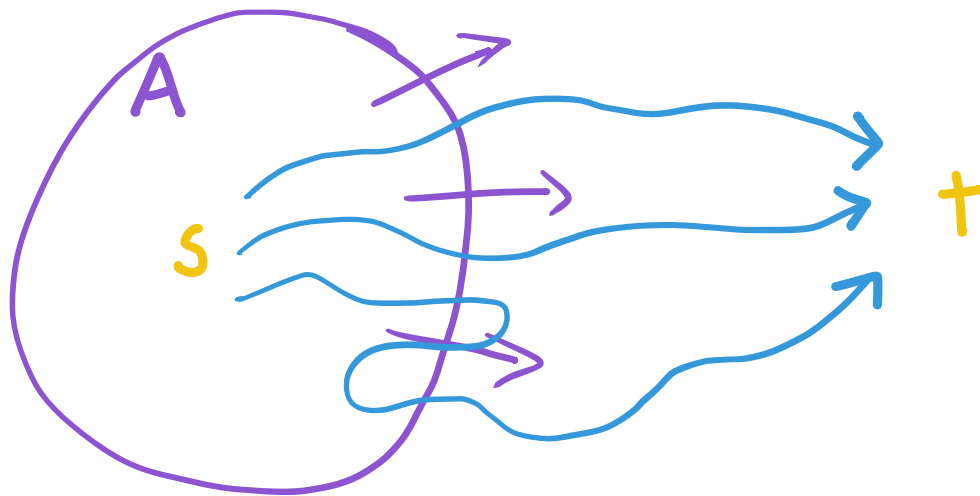
(Exercise about converse)



vs



let $C = \mathcal{Q}^+(A)$ min-cut, $f: E \rightarrow \mathbb{R}_{\geq 0}$ max flow
 ($A \subset V$ w/ $s \in A, t \notin A$)



$$(\text{size of } f) = (\text{net flow out of } A) \leq (\text{capacity of } \mathcal{Q}^+(A))$$



max path packings vs Max Flow

let $x: P_{s,t} \rightarrow \mathbb{R}_{\geq 0}$ encode a path packing

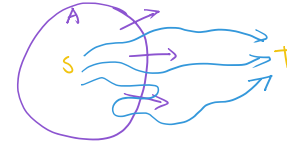
define $f: E \rightarrow \mathbb{R}_{\geq 0}$ as:

$$f(e) = \sum_{p: e \in p} x_p$$

then f is feasible flow of same size

min cut vs Max Flow

let $C = \mathcal{J}^+(A)$ min-cut, $f: E \rightarrow \mathbb{R}_{\geq 0}$ max flow
($A \subset V$ w/ $s \in A, t \notin A$)

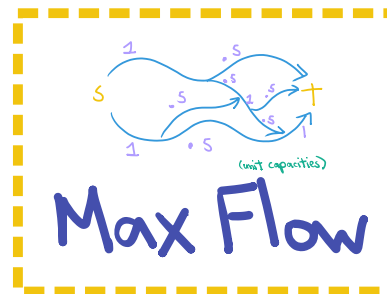


$$(\text{size of } f) = (\text{net flow out of } A) \leq (\text{capacity of } \mathcal{J}^+(A))$$

we showed



$\leq$



$\leq$



equal

max path packings vs Max Flow

let $x: P_{st} \rightarrow \mathbb{R}_{\geq 0}$ encode a path packing

define $f: E \rightarrow \mathbb{R}_{\geq 0}$ as:

$$f(e) = \sum_{p \ni e} x_p$$

then f is feasible flow of same size

min cut vs Max Flow

let $C = \mathcal{J}^c(A)$ min-cut, $f: E \rightarrow \mathbb{R}_{\geq 0}$ max flow

(MFC of unit flow)

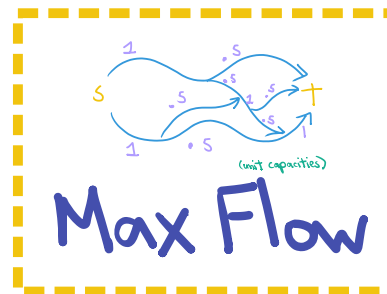


(size of f) = (net flow out of A) $\leq$ (capacity of $\mathcal{J}^c(A)$)

Theorem



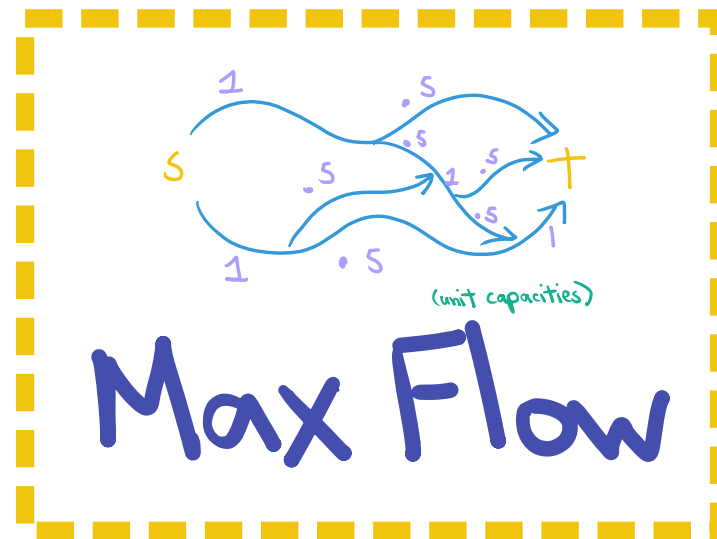
=



=



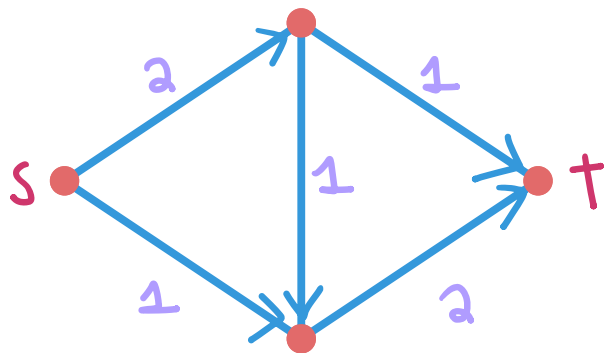
Next: Strongly Polynomial Times



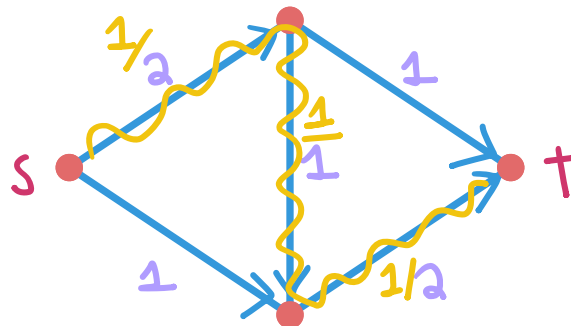
Residual graphs w/r/t flows

(exactly what you would expect)

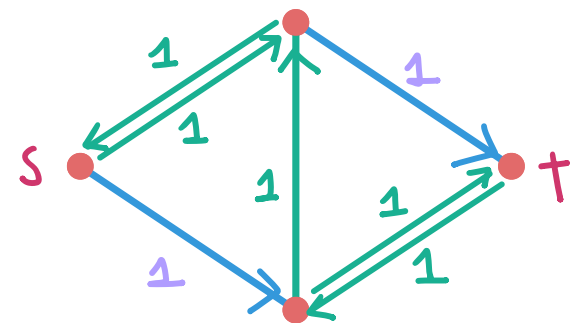
(capacities)



(1 unit flow)

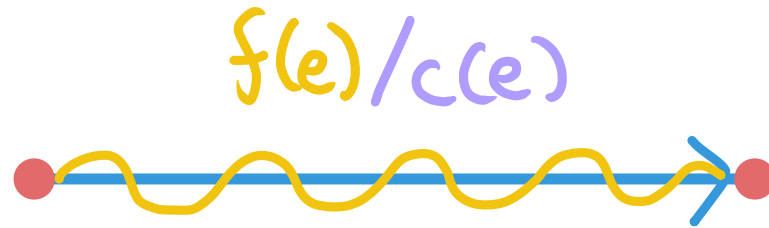


(residual edges)



let $f: E \rightarrow \mathbb{R}_{\geq 0}$ be (s,t) -flow

$e \in E$ w/ capacity $c(e)$

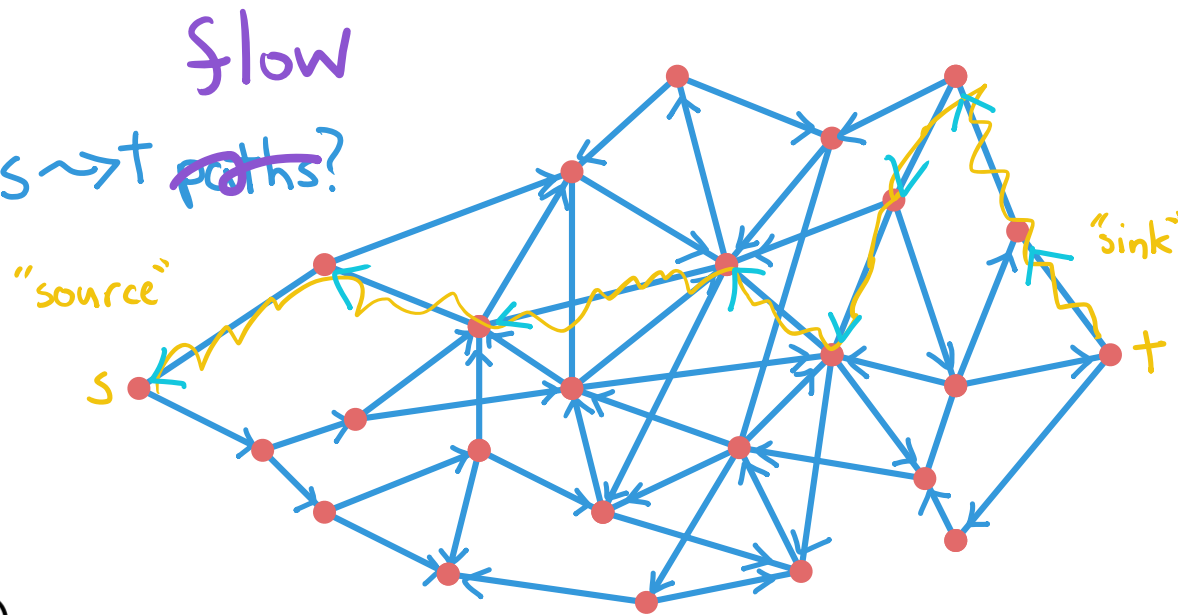


this leaves $c(e) - f(e)$ capacity in forwards direction
and $f(e)$ capacity backwards



How to compute

⑦ max # of ~~edge-disjoint~~ $s \rightsquigarrow t$ ~~paths?~~



augmenting-paths($G = (V, E), s, t$)

/ Ford and Fulkerson [FF56]*

1. $P \leftarrow \emptyset$

2. While there is an (s, t) -path $p : s \rightsquigarrow t$:

A. ~~Remove all of p 's edges from the graph.~~

B. ~~Add the reverse of all of p 's edges to the graph.~~

C. ~~Uncross p with previously selected paths, and add p to P .~~

update residual graph

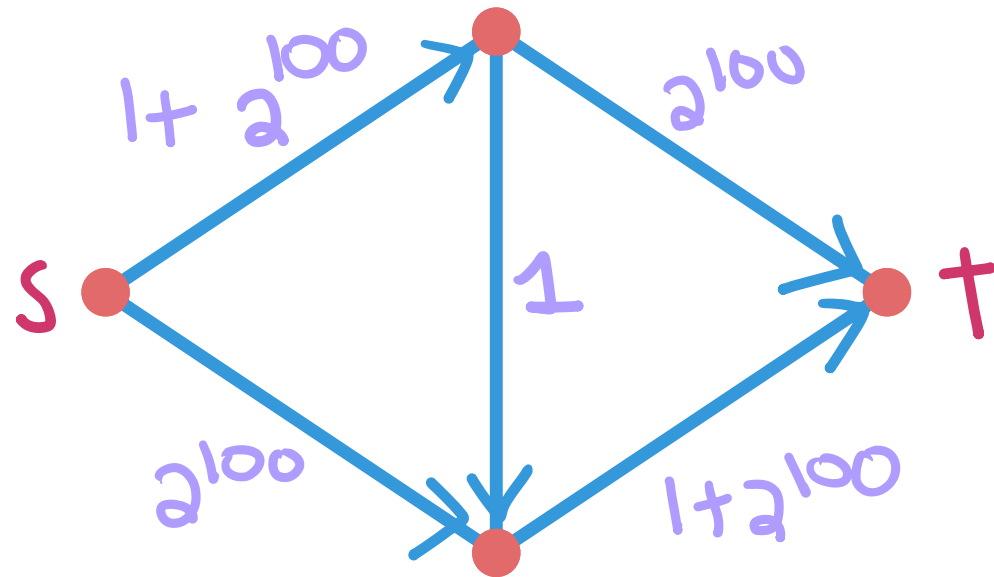
route as much flow as possible

reasonable, but not quite polynomial time

① while there is a path $p: s \rightsquigarrow t$:

② take p and route as much flow as possible

③ update residual graph



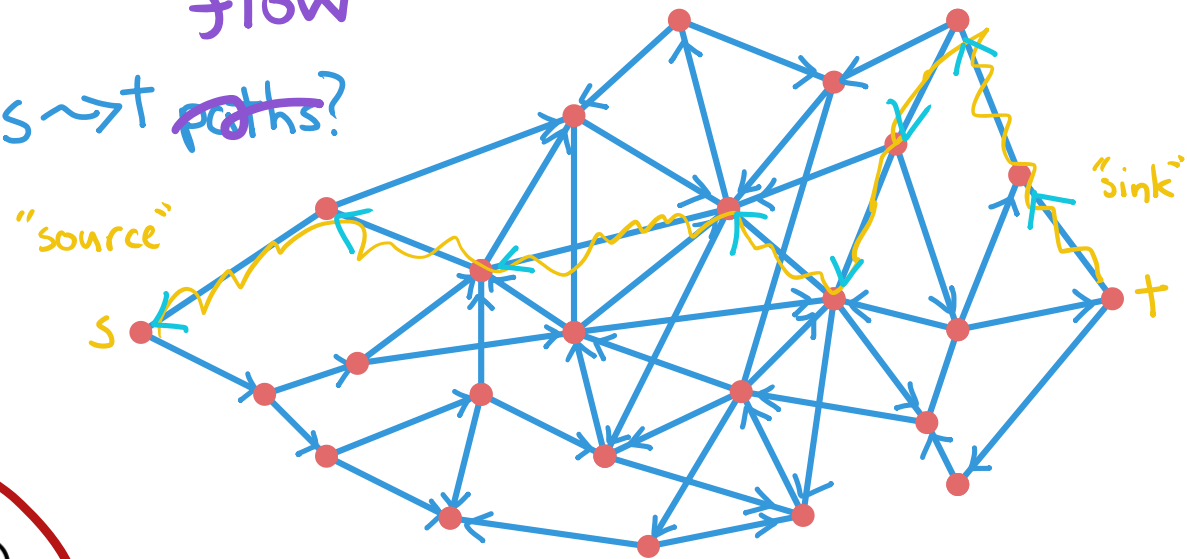
How to compute

⑦ max # of ~~edge-disjoint~~ $s \rightsquigarrow t$ ~~paths?~~

shortest path
(Edmonds, Karp)

"source"

s



augmenting-paths($G = (V, E), s, t$)

/* Ford and Fulkerson [FF56]

1. $P \leftarrow \emptyset$

2. While there is an (s, t) -path $p : s \rightsquigarrow t$:

A. ~~Remove all of p 's edges from the graph.~~

B. ~~Add the reverse of all of p 's edges to the graph.~~

C. ~~Uncross p with previously selected paths, and add p to P .~~

update residual graph

route as much flow as possible

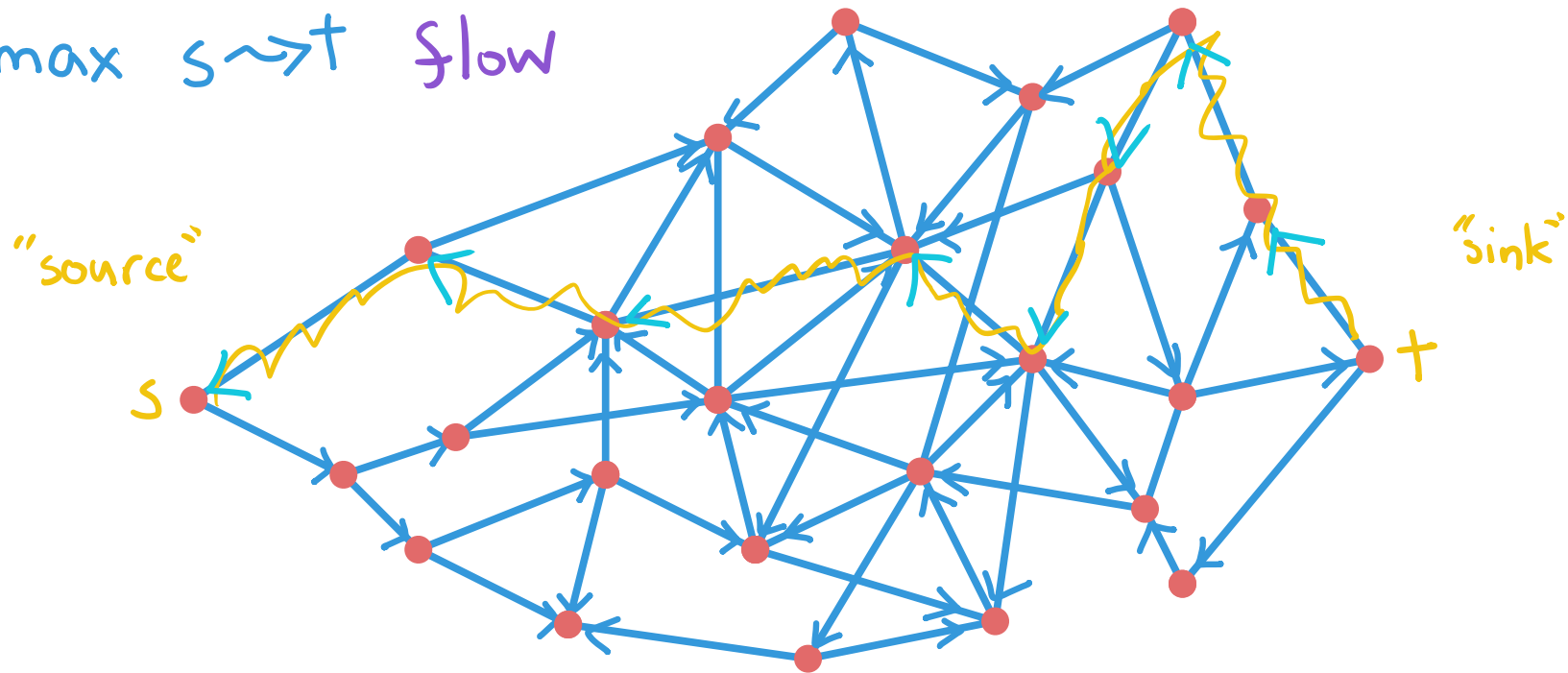
reasonable, but not quite polynomial time

(cleaned up)

(Edmonds, Karp)

How to compute

⑦ max $s \rightsquigarrow t$ flow



① while there is a path $p: s \rightsquigarrow t$:

Ⓐ take shortest path p and route as much flow as possible

Ⓑ update residual graph

Analysis

- ① while there is a path $p: s \rightsquigarrow t$:
 - Ⓐ take shortest path p and route as much flow as possible
 - Ⓑ update residual graph

Key
Lemma

$\forall v \in V$, distance from s to v in residual graph never decreases

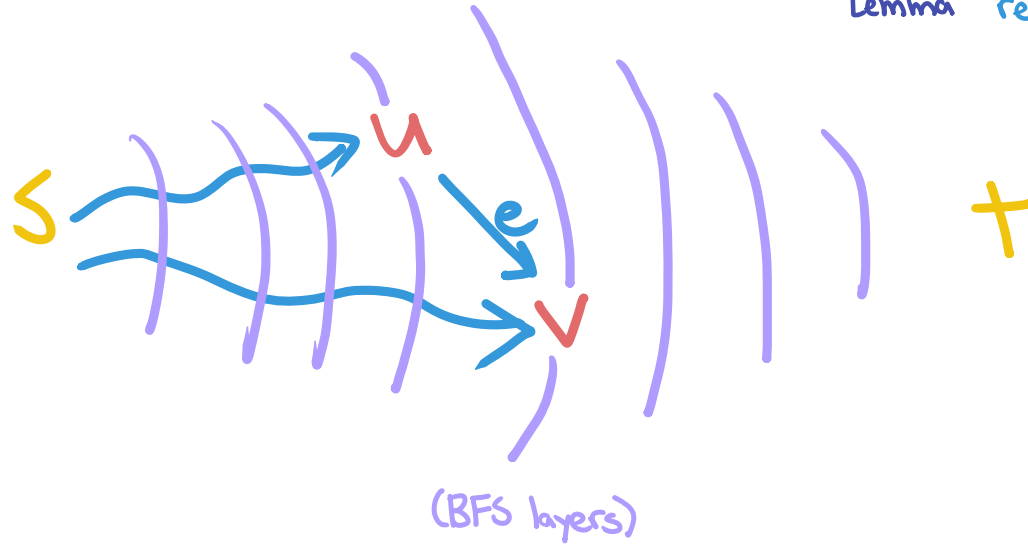
let us first assume lemma is true

fix $e=(u \rightarrow v) \in E$.

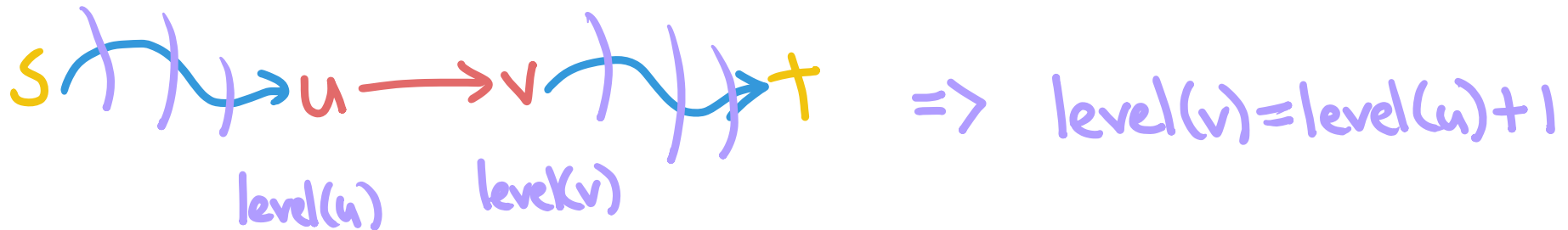
consider distances from s to u & v

- ① while there is a path $p: s \rightsquigarrow t$:
 - @ take shortest path p and route as much flow as possible
 - ⑥ update residual graph

Key $\forall v \in V$, distance from s to v in
Lemma residual graph never decreases



① e removed from ^(residual graph) RG , $\Rightarrow e$ in shortest $s \rightsquigarrow t$ path



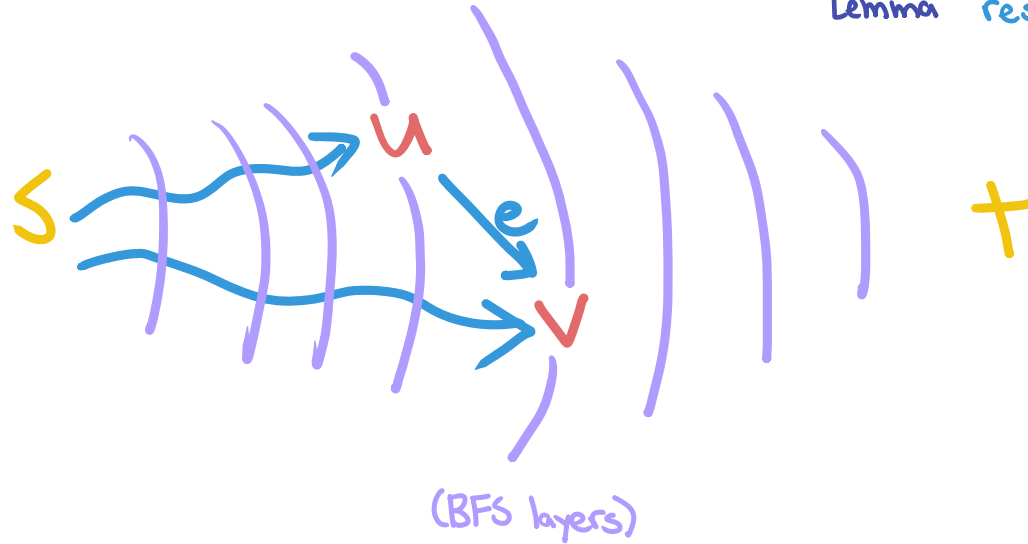
let us first assume lemma is true

fix $e = (u \rightarrow v) \in E$.

consider distances from s to u & v

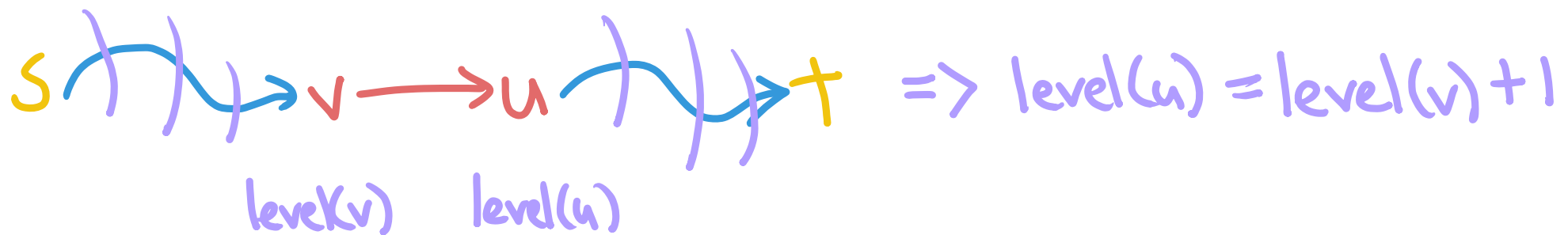
- ① while there is a path $p: s \rightsquigarrow t$:
- ② take shortest path p and route as much flow as possible
- ③ update residual graph

Key Lemma $\forall v \in V$, distance from s to v in residual graph never decreases



② $e = (u \rightarrow v)$ added back in to RG

$\Rightarrow (v \rightarrow u)$ was in shortest $s \rightsquigarrow t$ path



let us first assume lemma is true

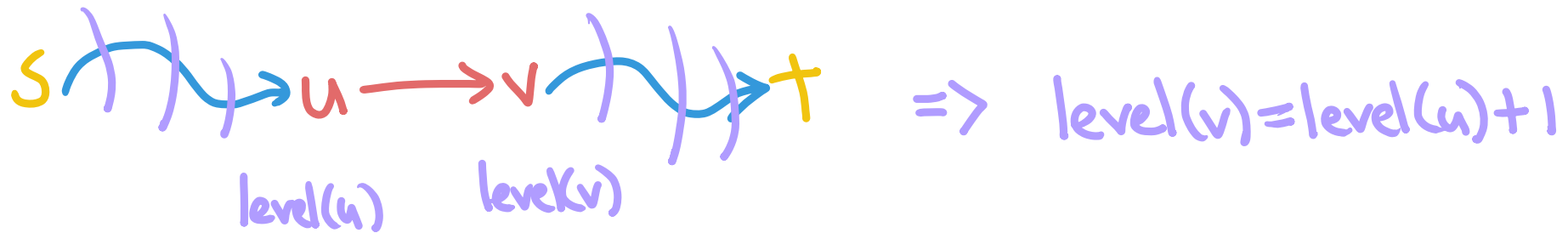
fix $e=(u \rightarrow v) \in E$.

consider distances from s to u & v

- ① while there is a path $p: s \rightsquigarrow t$:
 - @ take shortest path p and route as much flow as possible
 - ⑥ update residual graph

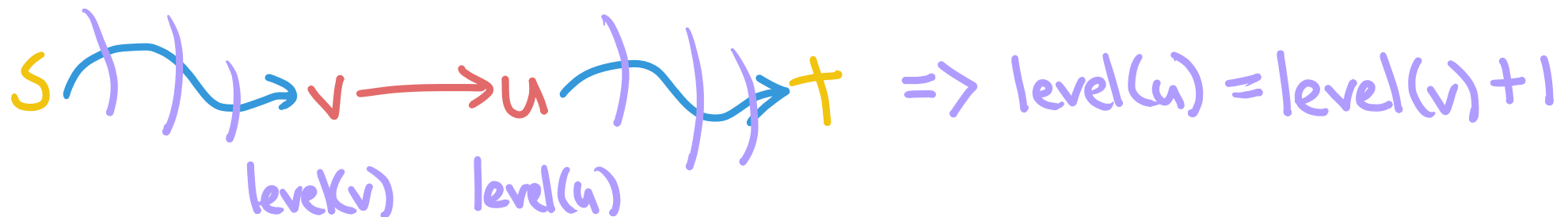
Key $\forall v \in V$, distance from s to v in
Lemma residual graph never decreases

① e removed from ^(residual graph) RG , $\Rightarrow e$ in shortest $s \rightsquigarrow t$ path



② $e=(u \rightarrow v)$ added back in to RG

$\Rightarrow (v \rightarrow u)$ was in shortest $s \rightsquigarrow t$ path



let us first assume lemma is true

fix $e = (u \rightarrow v) \in E$.

consider distances from s to u & v

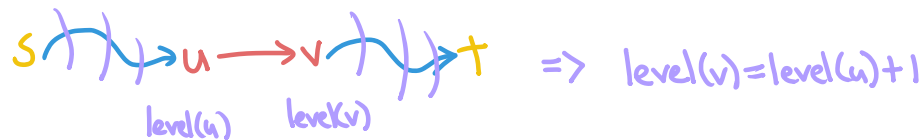
① while there is a path $p: s \rightsquigarrow t$:

Ⓐ take shortest path p and
route as much flow as possible

Ⓑ update residual graph

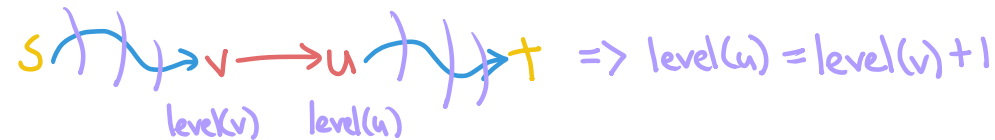
Key $\forall v \in V$, distance from s to v in
lemma residual graph never decreases

① e removed from ^(residual graph) RG , $\Rightarrow e$ in shortest $s \rightsquigarrow t$ path



② $e = (u \rightarrow v)$ added back in to RG

$\Rightarrow (v \rightarrow u)$ was in shortest $s \rightsquigarrow t$ path



each event ①/② $\Rightarrow \text{level}(u), \text{level}(v)$ have switched order.

levels nondecreasing by key lemma

max finite level is $n-1$

$\Rightarrow e$ gets deleted/added at most n times

let us first assume lemma is true

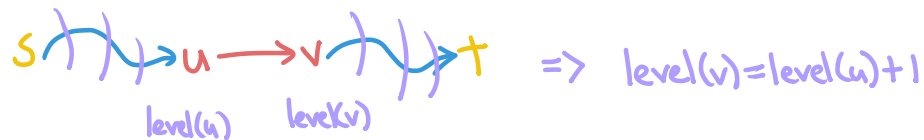
fix $e = (u \rightarrow v) \in E$.

consider distances from s to u & v

- ① while there is a path $p: s \rightsquigarrow t$:
 - Ⓐ take shortest path p and route as much flow as possible
 - Ⓑ update residual graph

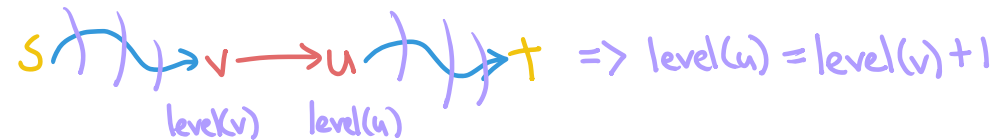
Key $\forall v \in V$, distance from s to v in Lemma residual graph never decreases

① e removed from RG , $\Rightarrow e$ in shortest $s \rightsquigarrow t$ path



② $e = (u \rightarrow v)$ added back in to RG

$\Rightarrow (v \rightarrow u)$ was in shortest $s \rightsquigarrow t$ path



✓ e gets deleted/added at most n times

Last observation each iteration removes (at least) one edge from residual graph. why?

b/c we route as much as possible

= $O(mn)$ iterations

- ① while there is a path $p: s \rightsquigarrow t$:
 - Ⓐ take shortest path p and route as much flow as possible
 - Ⓑ update residual graph

Running time

$O(m)$ per iteration

$O(mn)$ iterations

$O(m^2n)$ time

Key $\forall v \in V$, distance from s to v (" $l(v)$ ")
Lemma in residual graph never decreases

- ① while there is a path $p: s \rightsquigarrow t$:
- ② take shortest path p and route as much flow as possible
- ③ update residual graph

proof

suppose we augment along $p: s \rightsquigarrow t$
need to show no levels decrease

proof by induction on level after updating residual graph.

Base case: level 0

$s \rangle \rangle \rangle$

s is the only vertex at level 0, which never changes

General case: level $\beta > 0$

let $l(v) = d$ before updating
 $l(v) = \beta > 0$ after updating

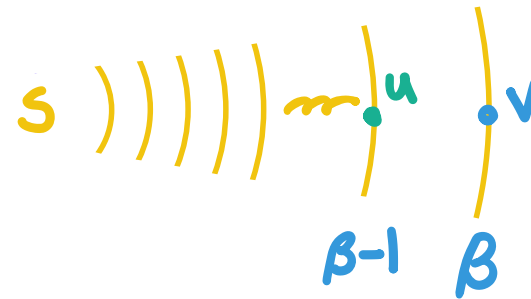
- ① while there is a path $p: s \rightarrow t$:
- ② take shortest path p and route as much flow as possible
- ③ update residual graph

Key $\forall v \in V$, distance from s to v in
lemma residual graph never decreases

Before



After



$u =$ 2nd to last vertex on shortest (s, v) path after

level(u) = $\beta - 1$ after augmenting

$\Rightarrow$ level(u) $\leq \beta - 2$ before augmenting

$\Rightarrow (v \rightarrow u)$ not on augmenting path

$\Rightarrow (u \rightarrow v)$ in residual graph
before augmenting

$\Rightarrow \text{level}(v) \leq \text{level}(u) + 1 \leq \beta - 1$
before augmenting

① while there is a path $p: s \rightsquigarrow t$:

② take shortest path p and
route as much flow as possible

③ update residual graph

Key $\forall v \in V$, distance from s to v in
lemma residual graph never decreases

Edmonds, Karp

- ① while there is a path $p: s \rightsquigarrow t$:
 - Ⓐ take shortest path p and route as much flow as possible
 - Ⓑ update residual graph

computes max (s,t) -flow, min (s,t) -cut in $O(m^2n)$ time

Key Lemma $\forall v \in V$, distance from s to v in residual graph never decreases

① while there is a path $p: s \rightarrow t$:

② take shortest path p and
route as much flow as possible

③ update residual graph

Key $\forall v \in V$, distance from s to v in
lemma residual graph never decreases

① while there is a path $p: s \rightarrow t$:

② take shortest path p and
route as much flow as possible

③ update residual graph

Key $\forall v \in V$, distance from s to v in
lemma residual graph never decreases

① while there is a path $p: s \rightarrow t$:

② take shortest path p and
route as much flow as possible

③ update residual graph

Key $\forall v \in V$, distance from s to v in
lemma residual graph never decreases

① while there is a path $p: s \rightarrow t$:

② take shortest path p and
route as much flow as possible

③ update residual graph

Key $\forall v \in V$, distance from s to v in
lemma residual graph never decreases

Exercise 18.5. Let $G = (V, E)$ be a directed graph with integer edge capacities $c : E \rightarrow \mathbb{N}$. Let each edge be assigned a color red, white, or blue. Recall that an *American walk* is a walk where the edges alternate red, white, blue, red, white, blue, ... Here the first color may be any color; all that matters is that we keep cycling through the colors patriotically. Given $s, t \in V$, consider the problem of computing the size of a maximum packing of American (s, t) -walks.

(Here the definitions are just like for (s, t) -paths. A “packing of walks” is a collection of walks that, for each edge e , uses the edge e at most as many times as its capacity. The “size of a packing” refers to the number of walks.)

Exercise 19.8. Recall that in chess, a *rook* () is a piece that can move horizontally or vertically as far as it wants. (It cannot move diagonally.) We say that a rook on a particular square is *attacking* another chess piece at another square if the rook can be moved horizontally or vertically from its square to the square of that chess piece.

Consider the following problem which we call the “ n -rooks problem”. At a high level, there is an $n \times n$ board where some of the squares have been marked off as “unavailable”. The remaining squares are “available”. The goal is to place n rooks in the available squares so that none of the rooks are attacking each other. We call this a “non-attacking placement” of the n rooks. (While rooks cannot be placed on unavailable squares, they can still move over unavailable squares to attack other pieces on the other side of unavailable squares.)

Below we draw a non-attacking placement of 4 rooks on a 4×4 chess board. Unavailable squares are marked by X. Rooks are marked by .

X			X
	X	X	X
	X		
X		X	

Formally, the input consists of an $n \times n$ bit-array $A[1..n, 1..n] \in \{\mathbf{true}, \mathbf{false}\}^{n \times n}$, where $A[i, j] = \mathbf{true}$ indicates that square (i, j) is available, and $A[i, j] = \mathbf{false}$ indicates that square (i, j) is unavailable. The goal is to decide if there is a non-attacking placement of n rooks in the available squares. For this problem, either (a) design and analyze a polynomial time algorithm (the faster the better), or (b) prove that the problem is NP-Hard.

Exercise 19.3. Suppose we are given two-dimensional array $A[1..m][1..n]$ of non-negative real values such that each row and column sum is an integer. We want to round A to an integer matrix, replacing each entry $x \in A$ with either $\lceil x \rceil$ or $\lfloor x \rfloor$, while maintaining the sum in any row or column of A . For example,

$$\begin{pmatrix} 1.2 & 3.4 & 2.4 \\ 3.9 & 4.0 & 2.1 \\ 7.9 & 1.6 & 0.5 \end{pmatrix} \text{ rounds to } \begin{pmatrix} 1 & 4 & 2 \\ 4 & 4 & 2 \\ 8 & 1 & 1 \end{pmatrix}.$$

Consider the problem of computing such a rounding, or declaring that none exists. For this problem, either (a) design and analyze a polynomial time algorithm (the faster the better), or (b) prove that the problem is NP-Hard.