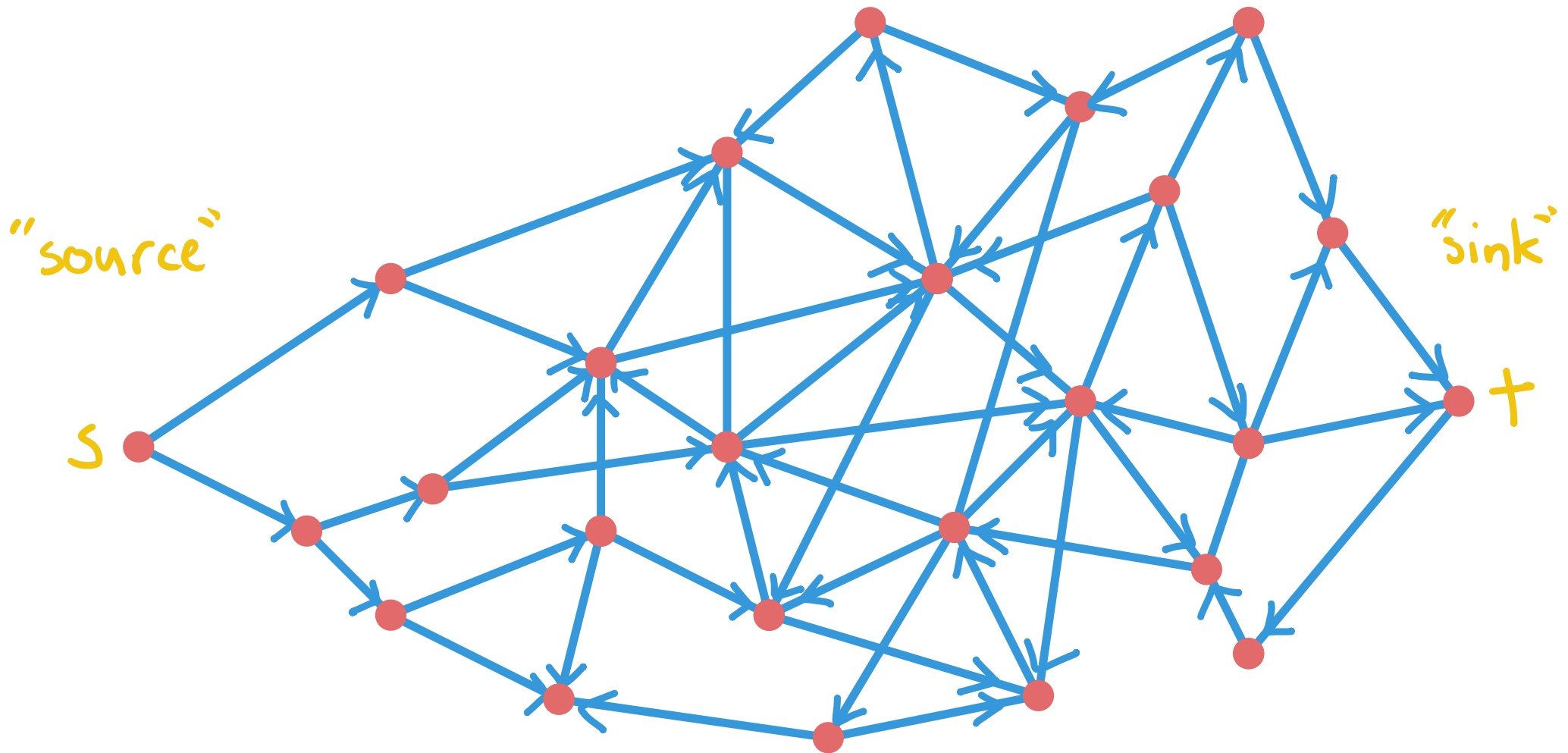


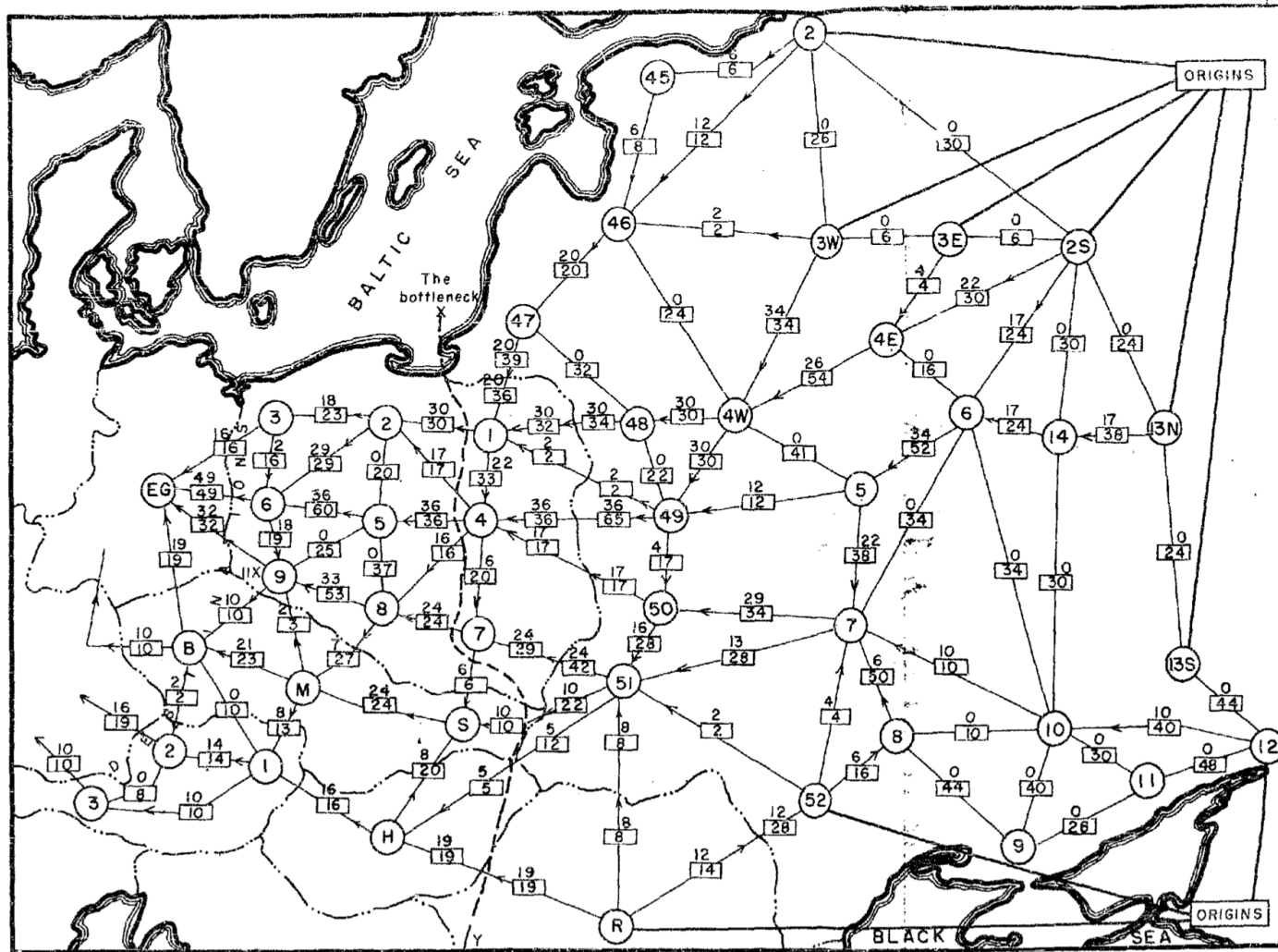
Packing and Covering Paths

Measuring connectivity



how well-connected are s and t?

Physical infrastructure



SECRET
RM-1373
10-24-55
-33-

Fig. 7 — Traffic pattern: entire network available

Legend:

— International boundary

(B) Railway operating division

← 12 → Capacity: 12 each way per day.
Required flow of 9 per day toward destinations (in direction of arrow) with equivalent number of returning trains in opposite direction

All capacities in $\sqrt{1000}$'s of tons } each way per day

Origins: Divisions 2, 3W, 3E, 2S, 13N, 13S, 12, 52 (USSR), and Roumania

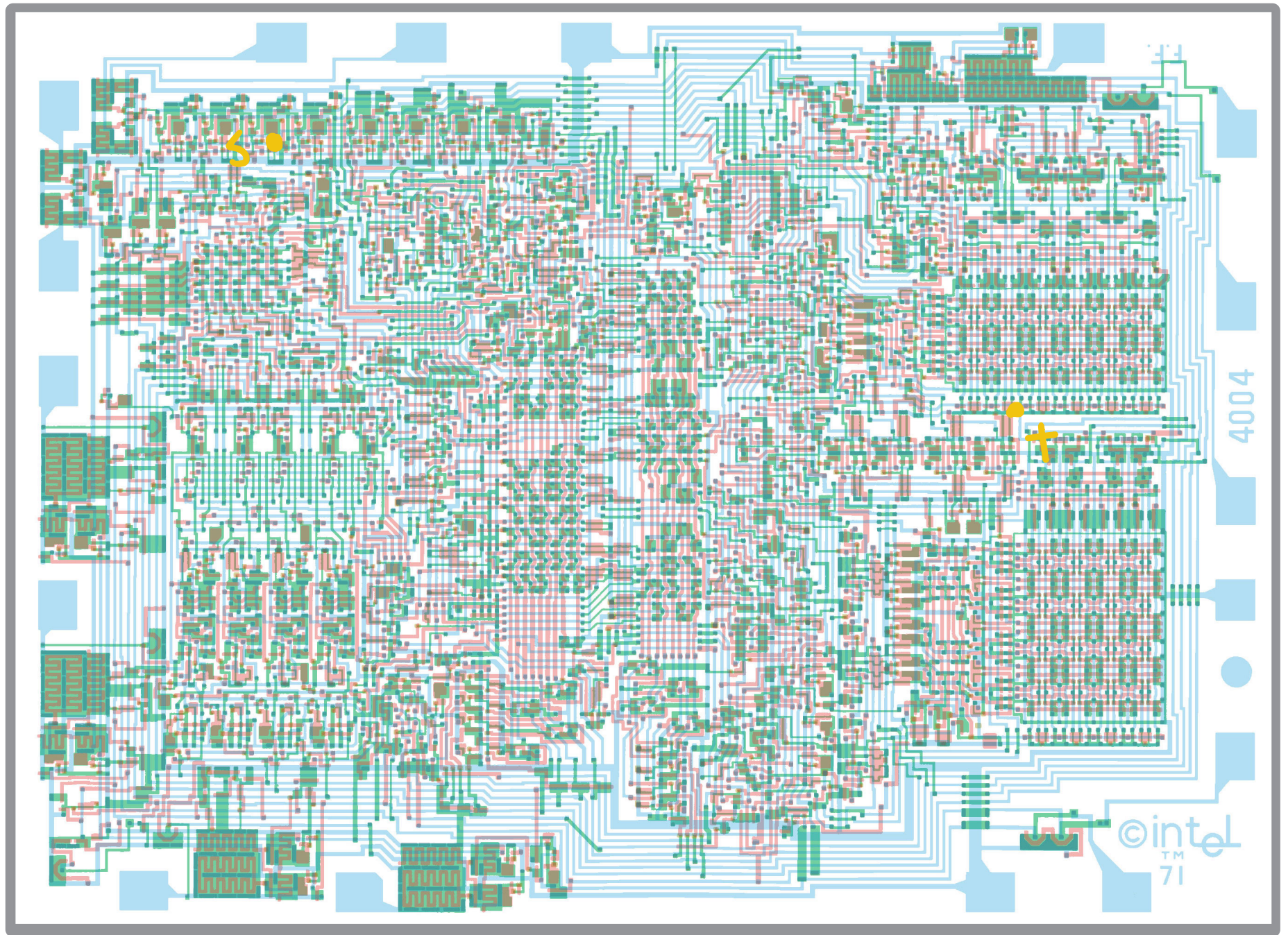
Destinations: Divisions 3, 6, 9 (Poland);
B (Czechoslovakia); and 2, 3 (Austria)

Alternative destinations: Germany or East Germany

Note IIX of Division 9, Poland

(from Jeff's notes)

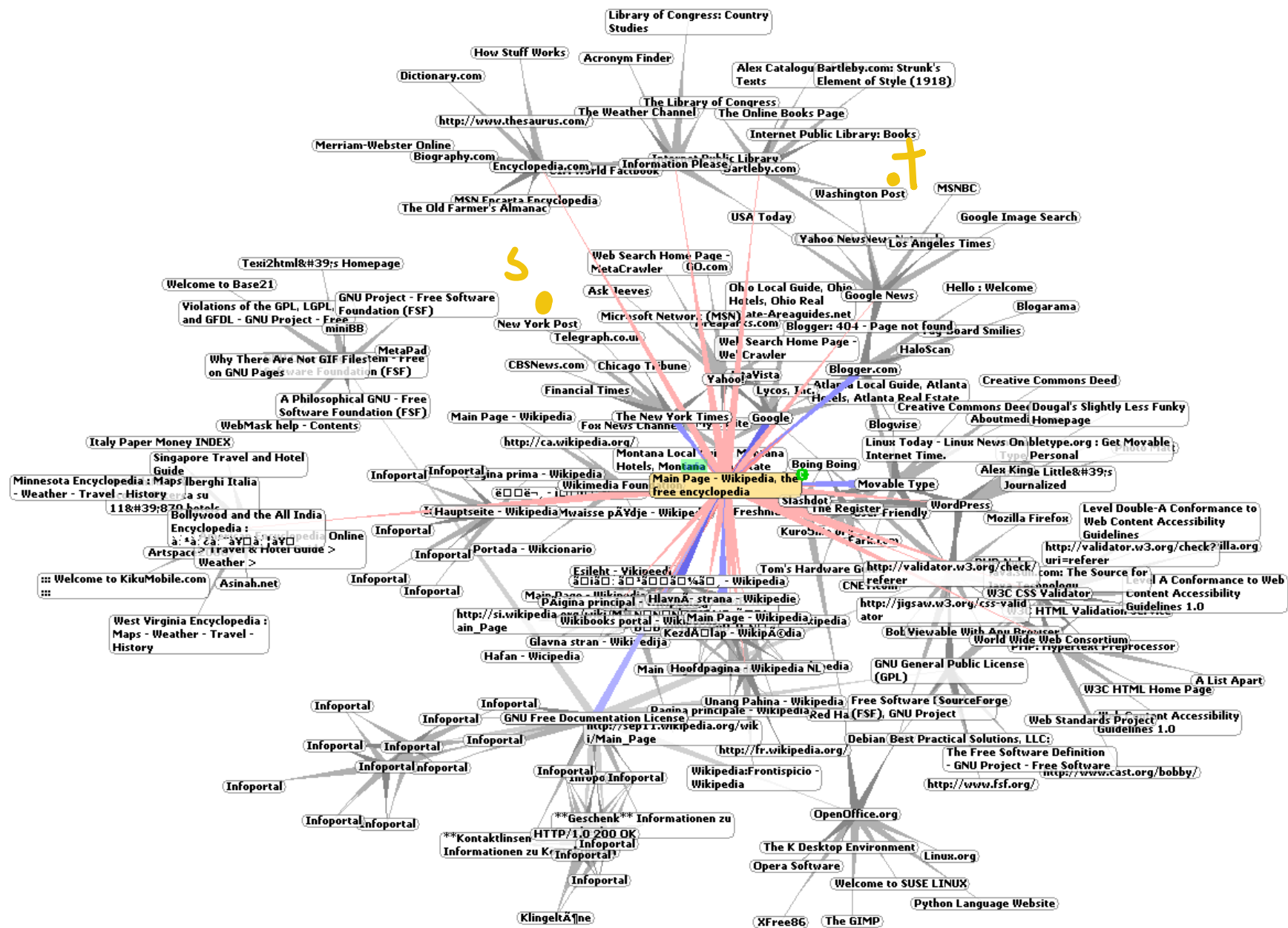
CPU's



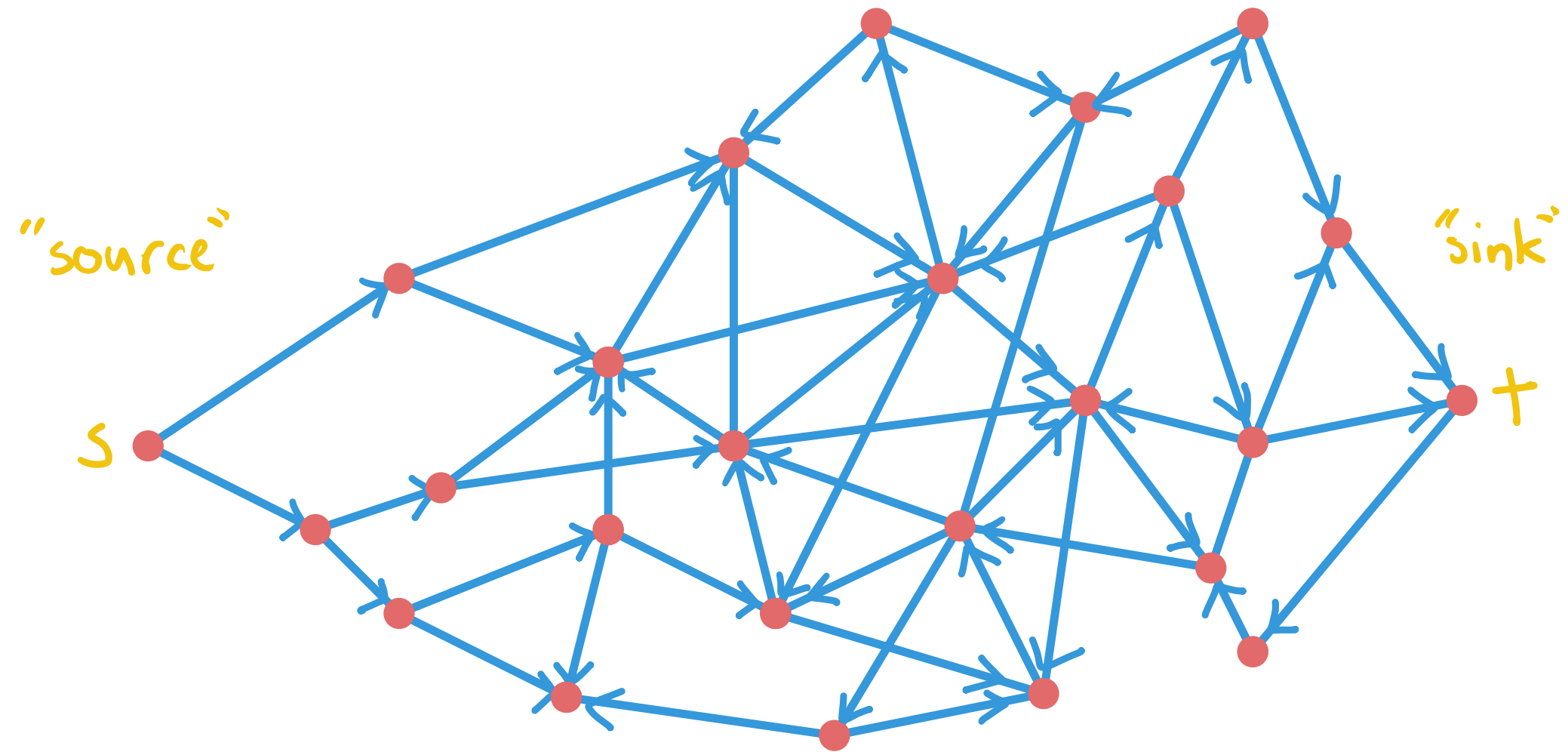
social network



www

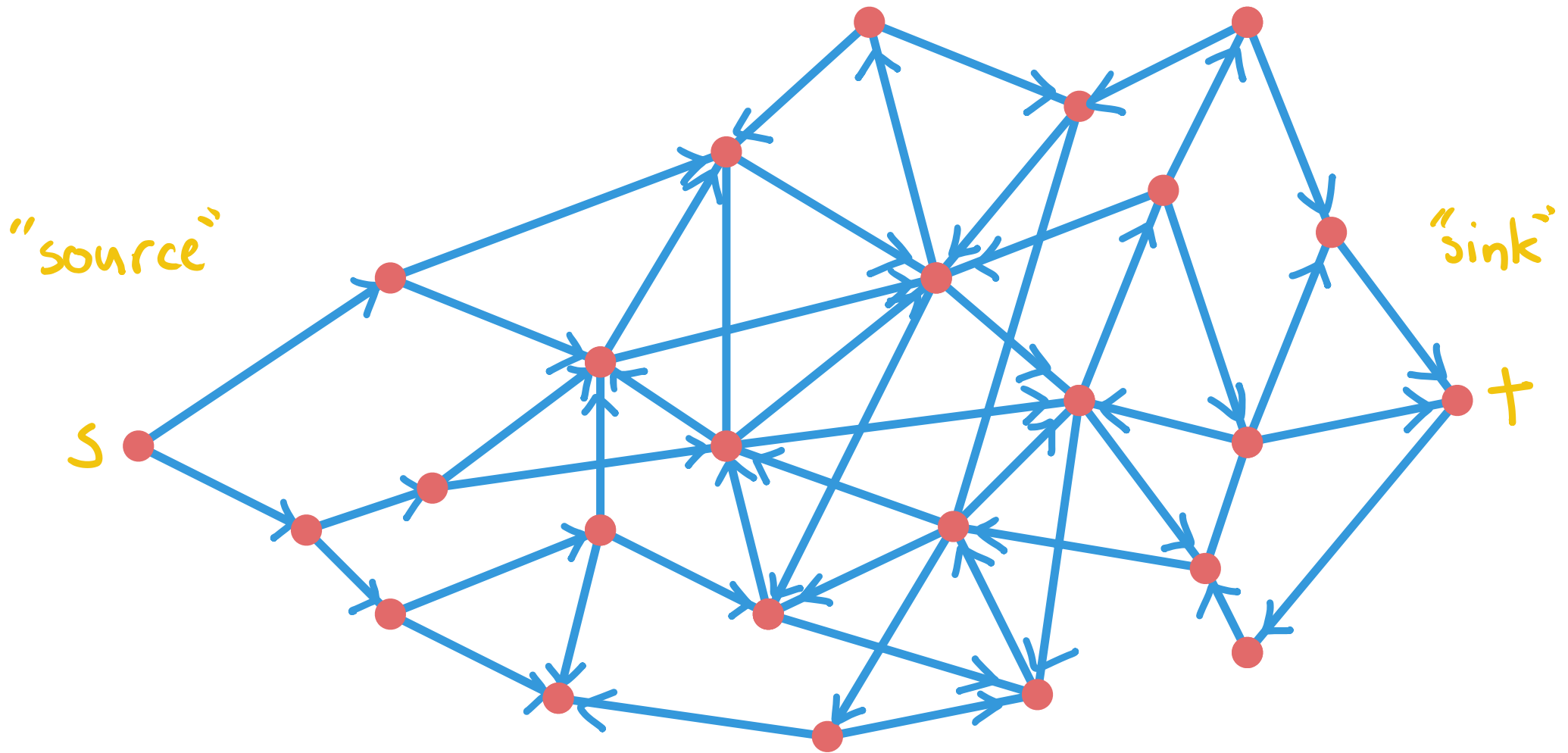


Measuring connectivity



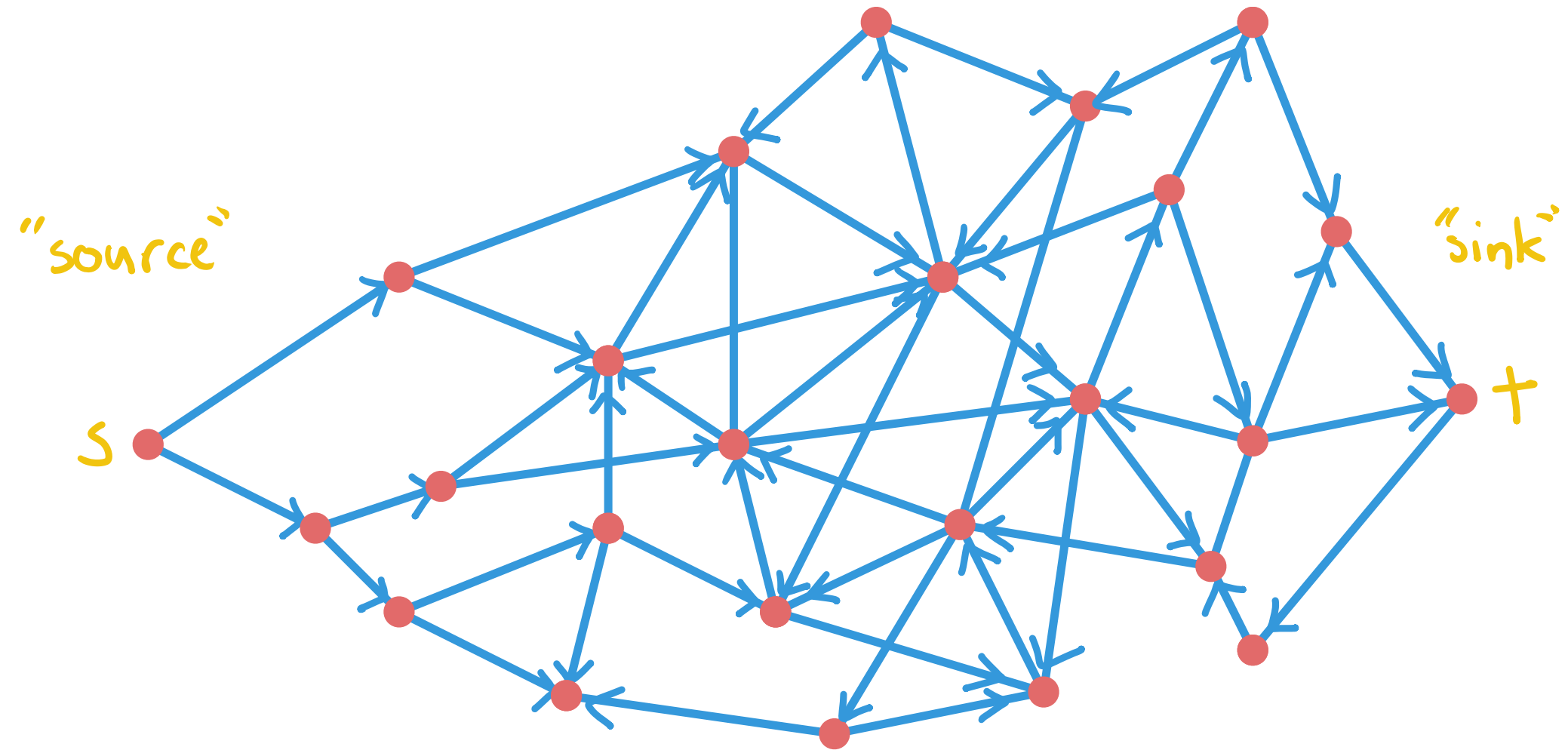
① Is there an $s \rightsquigarrow t$ path?

Measuring connectivity



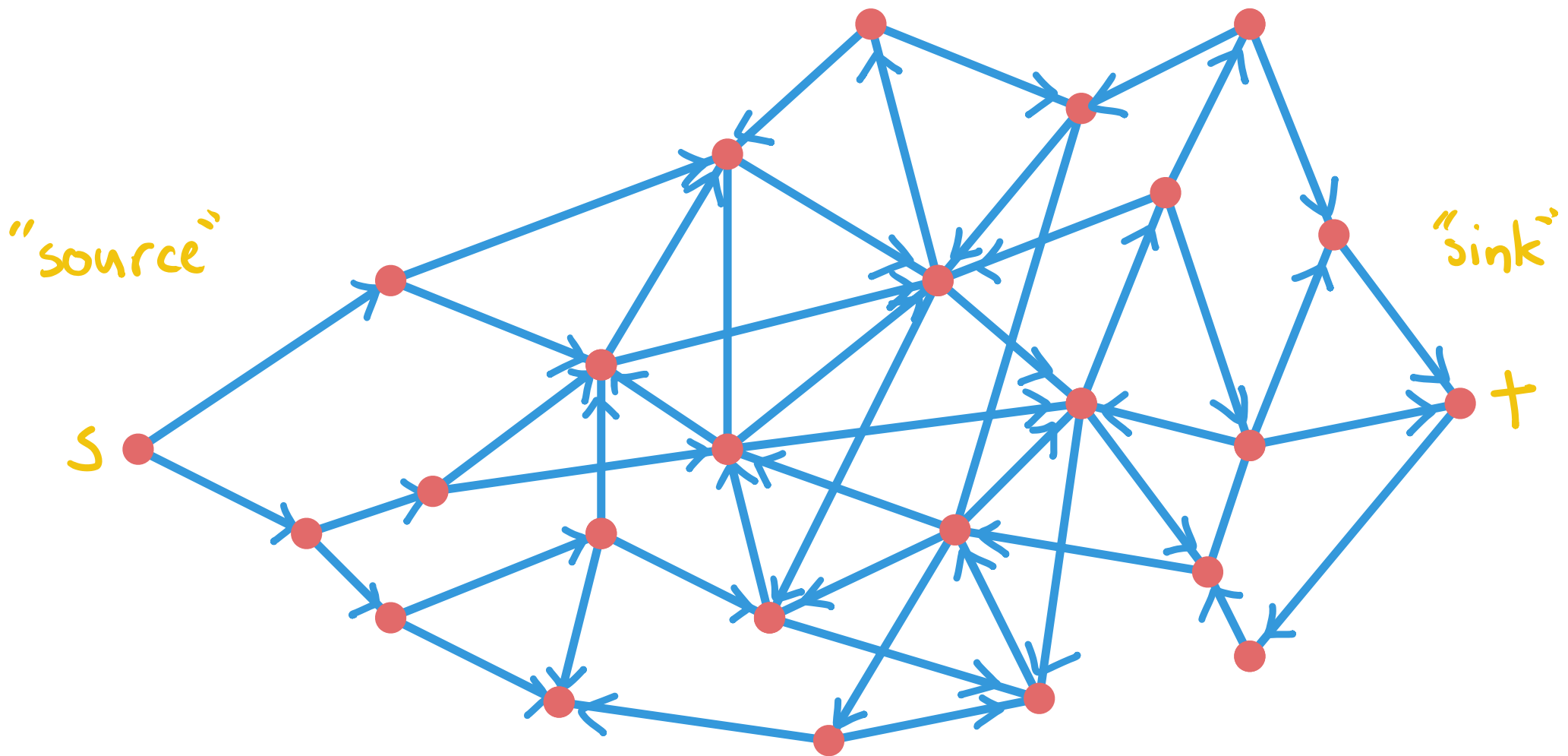
② How many distinct $s \rightsquigarrow t$ paths?

Measuring connectivity



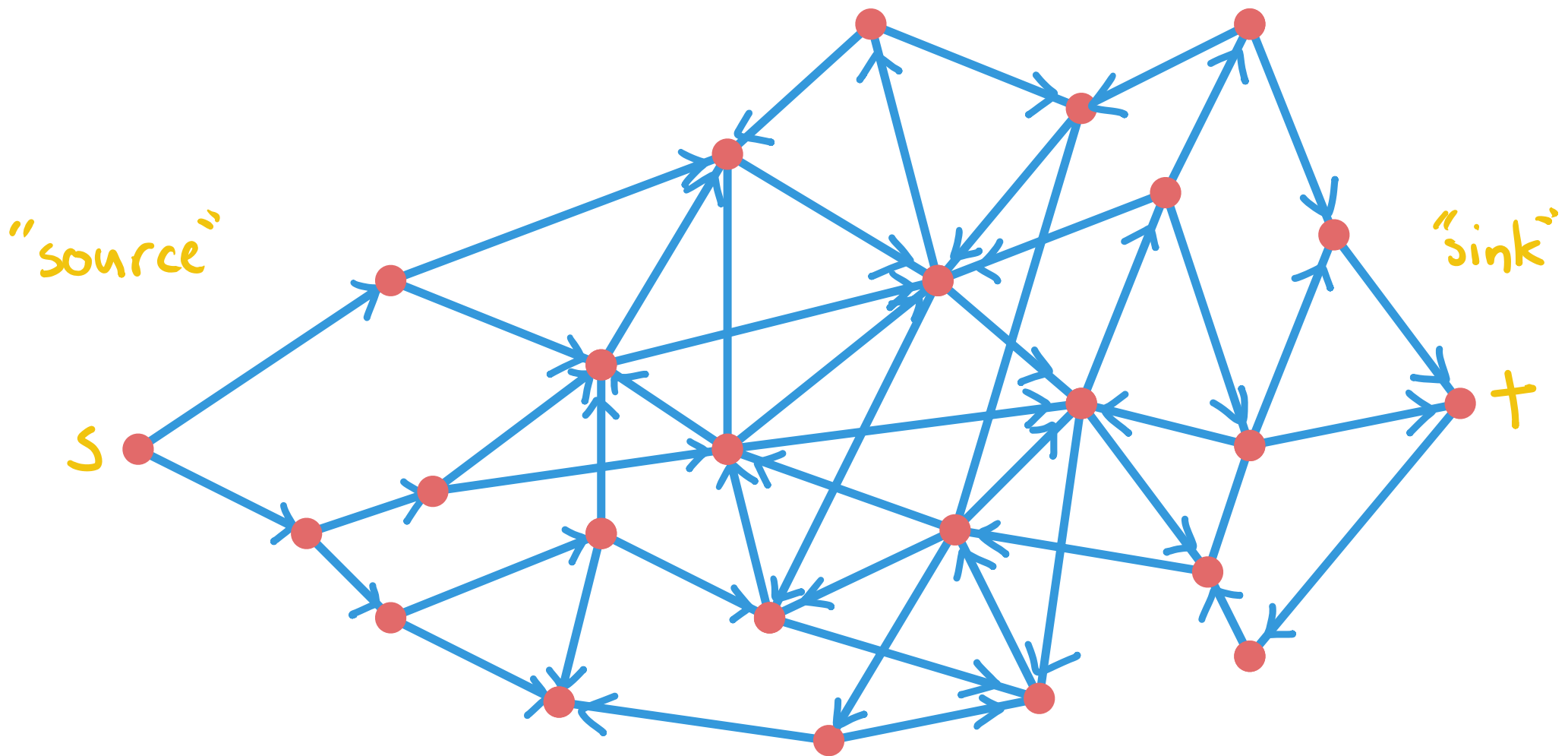
③ how many k -edge $s \rightsquigarrow t$ walks?
(k fixed)

Measuring connectivity



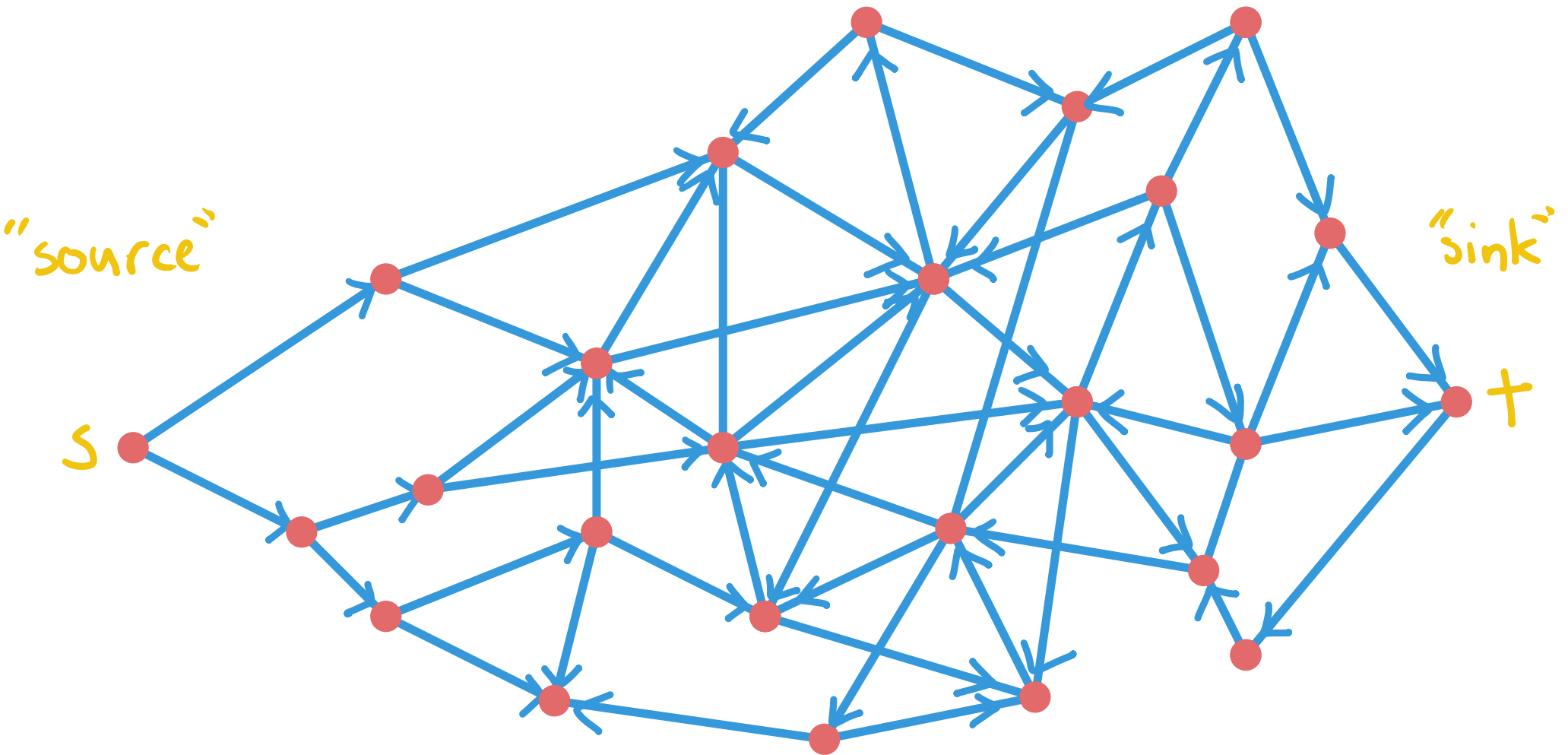
⑤ how many shortest $s \rightsquigarrow t$ paths?

Measuring connectivity



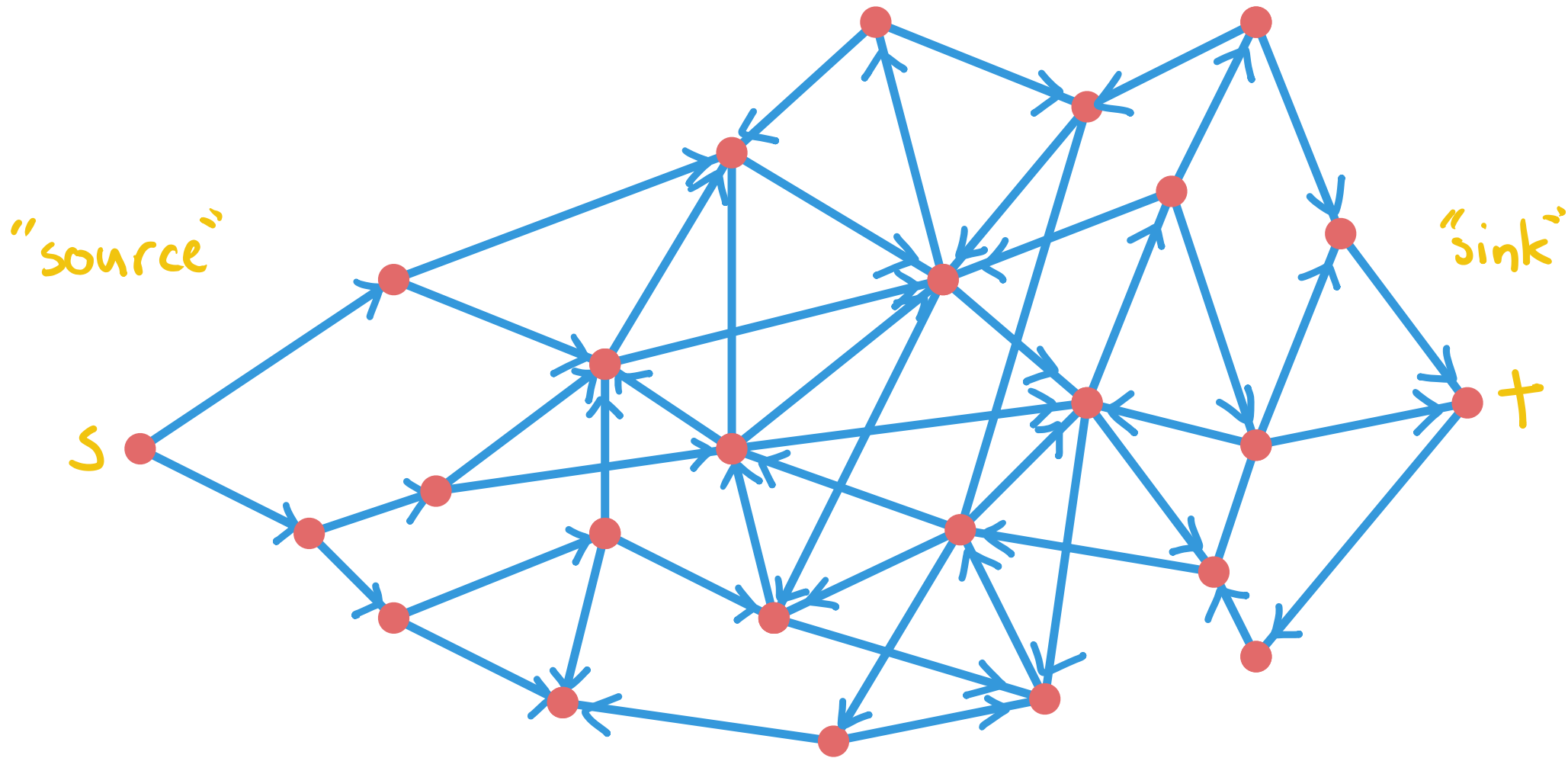
⑥ In a random walk starting from s , what is the expected # of steps until we reach t ?

Measuring connectivity

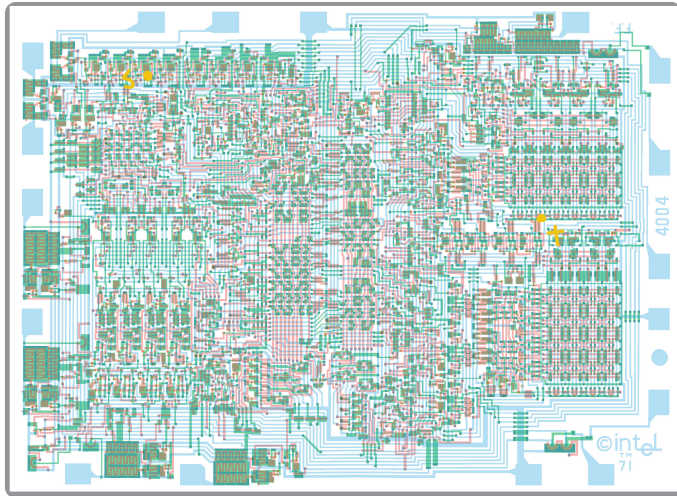
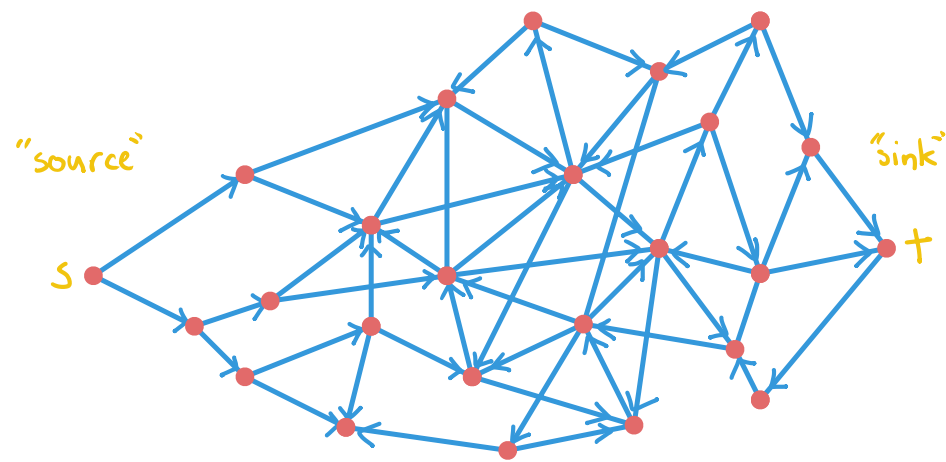


⑦ max # of edge-disjoint $s \rightsquigarrow t$ paths?

Measuring connectivity

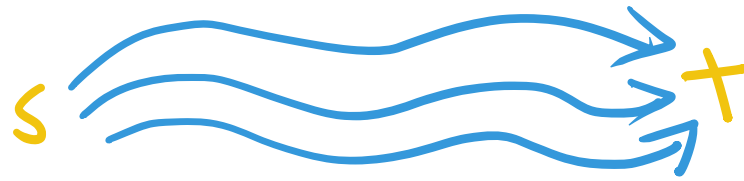


⑧ min # of edge deletions to disconnect $s \rightarrow t$?



- 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 $s \rightsquigarrow t$?

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



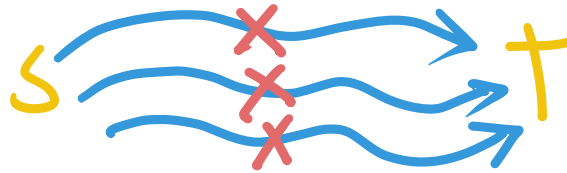
"path packing" collection of edge-disjoint paths
"packing paths into edges"

Question ⑦ = max cardinality path packing

example of "packing problem"

path p "packs into" each edge $e \in p$

⑧ min # edge deletions to disconnect $s \rightsquigarrow t$



" $s \rightarrow t$ cut"

set of edges whose removal disconnects t from s

Question ⑧ = min s - t cut

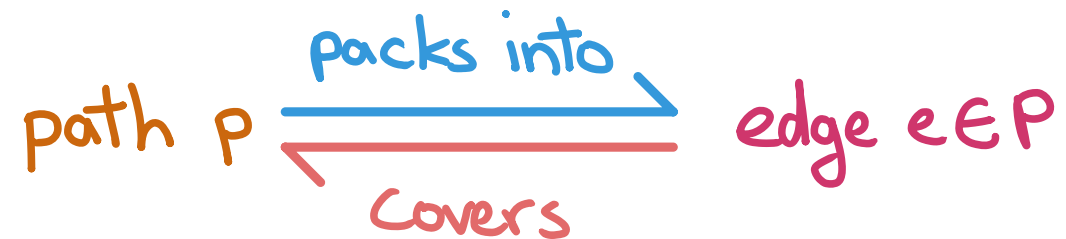
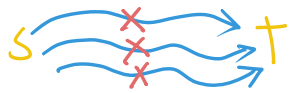
example of "covering problem"

edge e "covers" path p containing e

goal: min edge set covering all $s \rightsquigarrow t$ paths

Duality

path p $\xrightarrow{\text{packs into}}$ 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

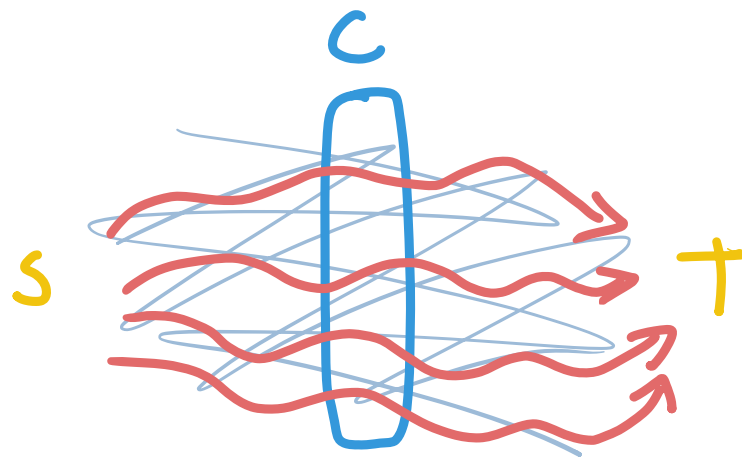
$$\left(\begin{array}{c} \text{max \#} \\ \text{disjoint} \\ s \rightsquigarrow t \text{ paths} \end{array} \right) \leq \left(\begin{array}{c} \text{min \# edges} \\ \text{in any } s \rightsquigarrow t \text{ cut} \end{array} \right)$$

Proof let

P : collection of edge disjoint paths

$C \subseteq E$: s - t cut

each path overlaps C



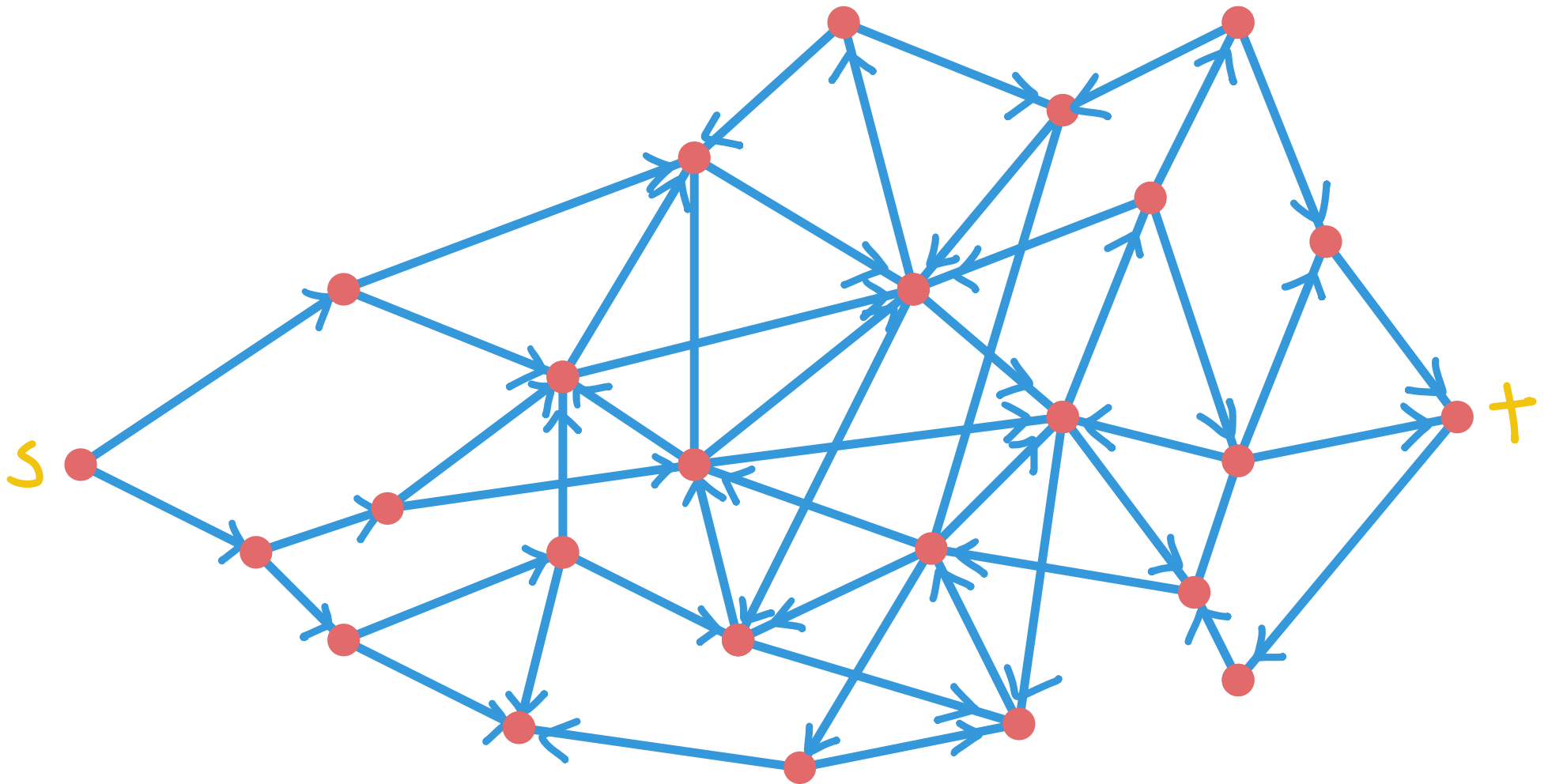
$$|P| \leq \sum_{p \in P} |p \cap C| = \left| \bigcup_{p \in P} (p \cap C) \right| \leq |C|$$

paths are
edge-disjoint

How to compute

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

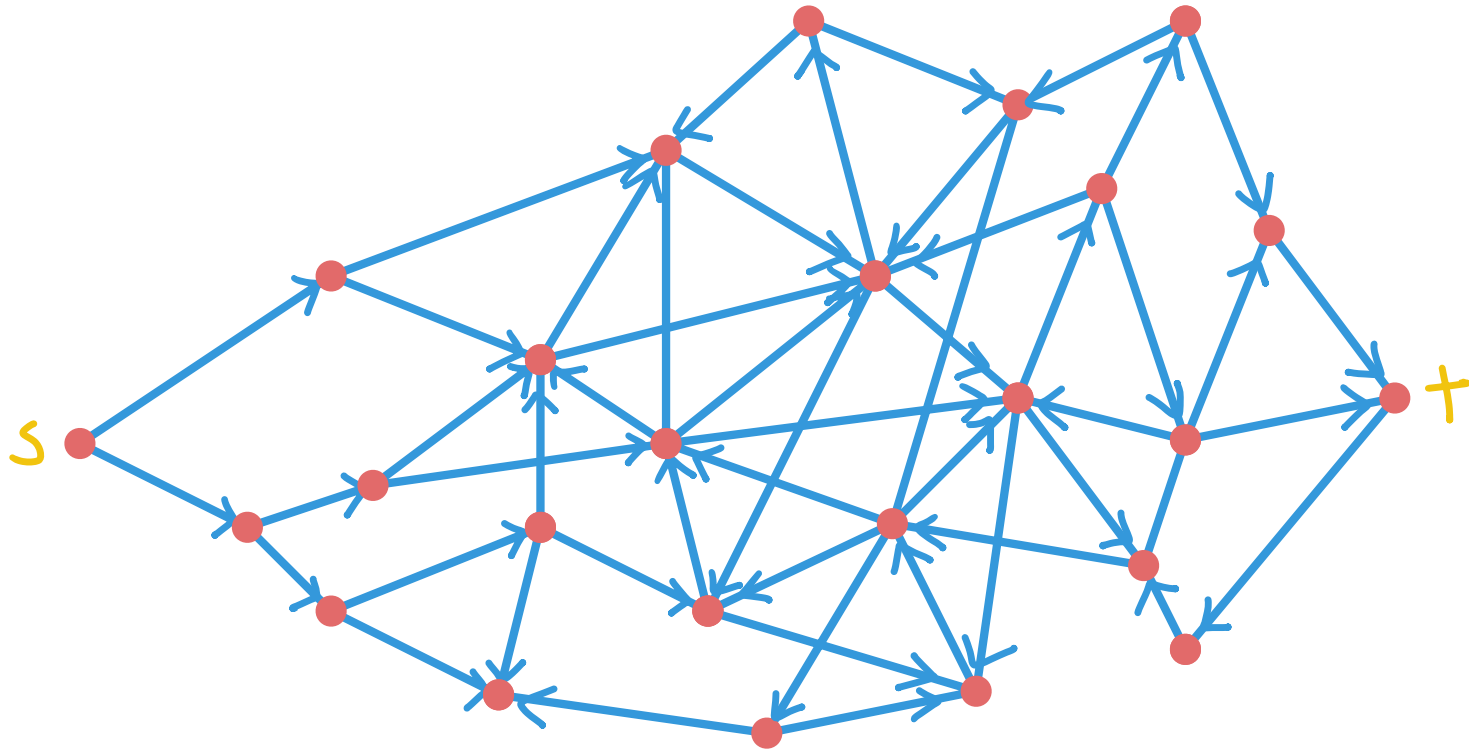
⑦ max # edge-disjoint $s \rightsquigarrow t$ paths?



How to compute

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

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



Iterative/greedy

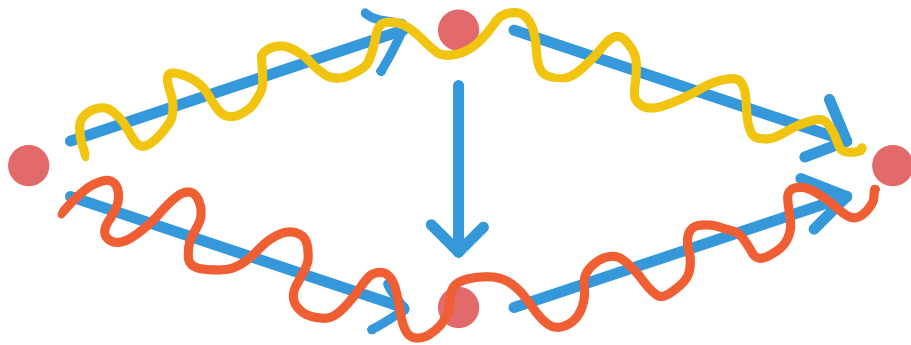
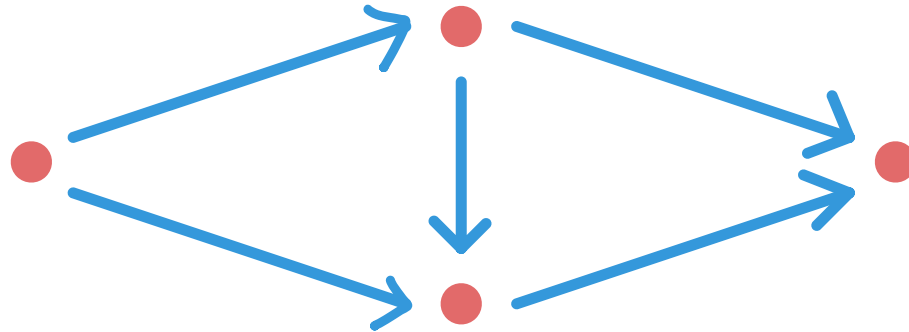
① while there is a path $p: s \rightsquigarrow t$

@ take p and remove all its edges from the graph

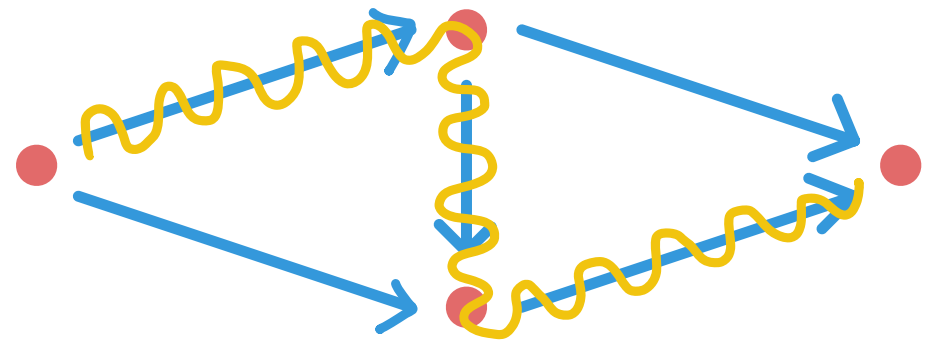
Iterative/greedy

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

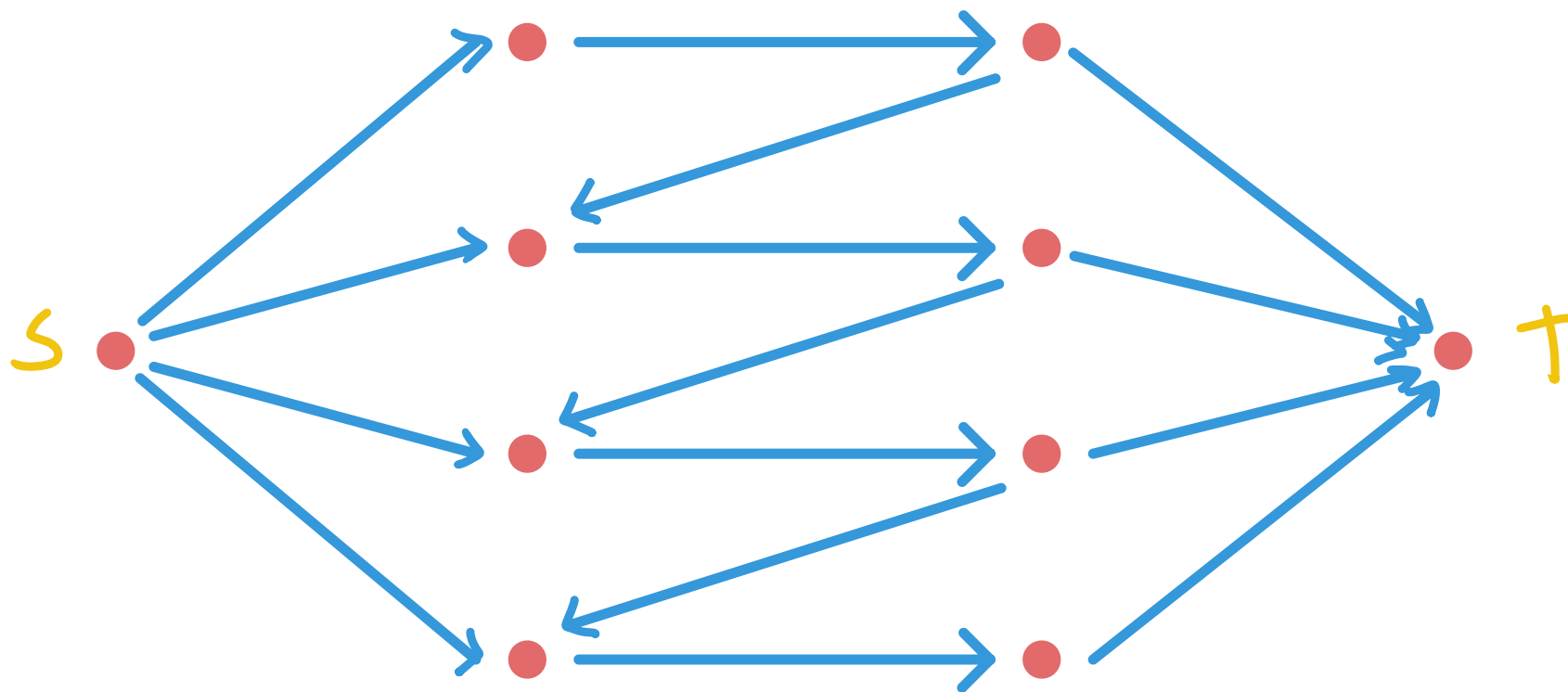
@ take p and remove all its edges from the graph

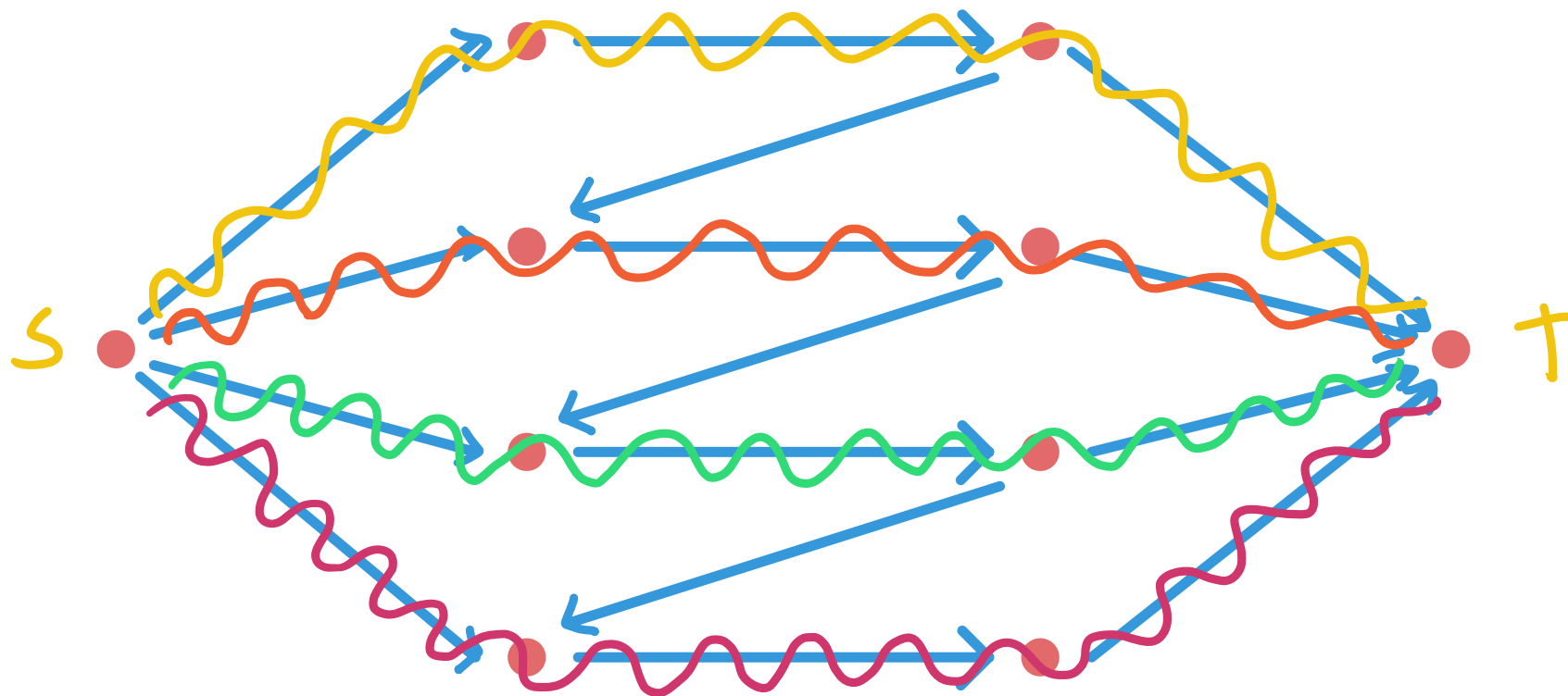


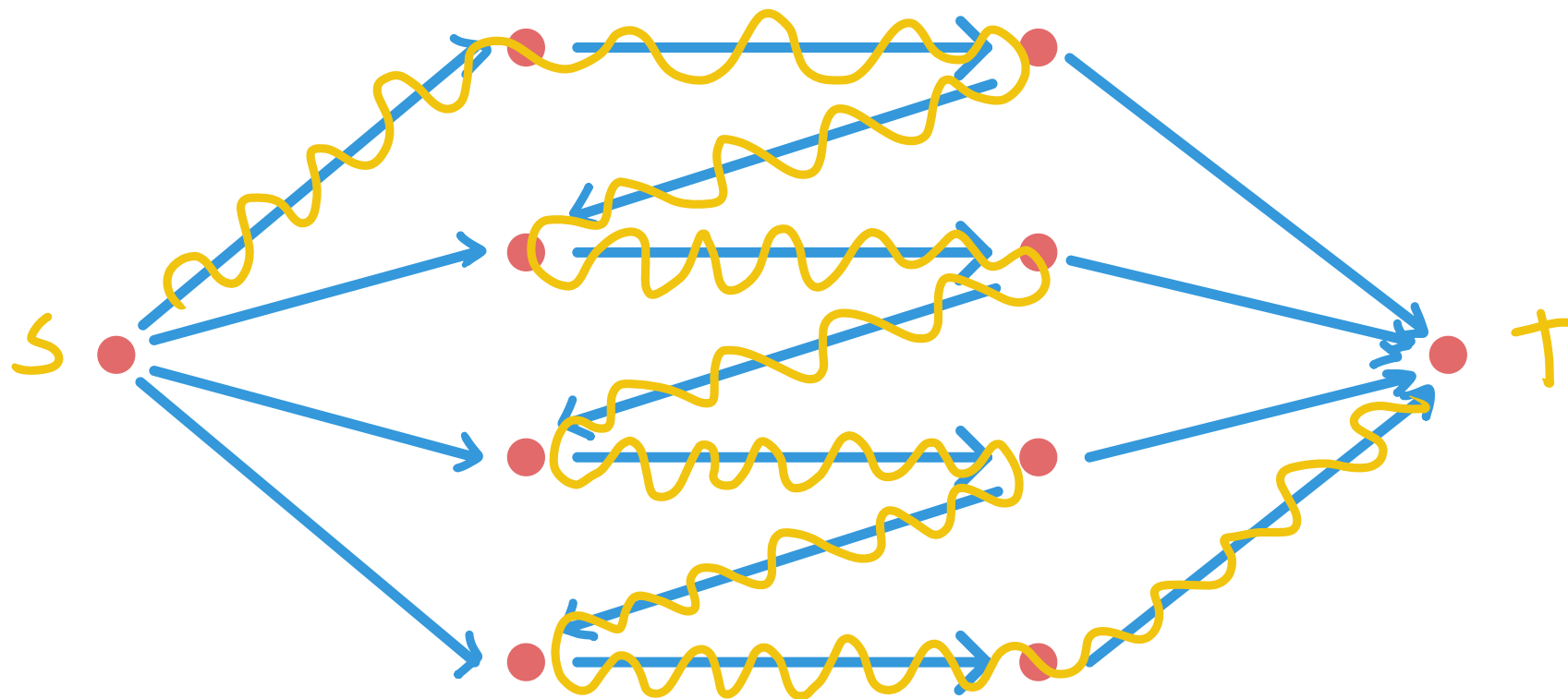
(good)

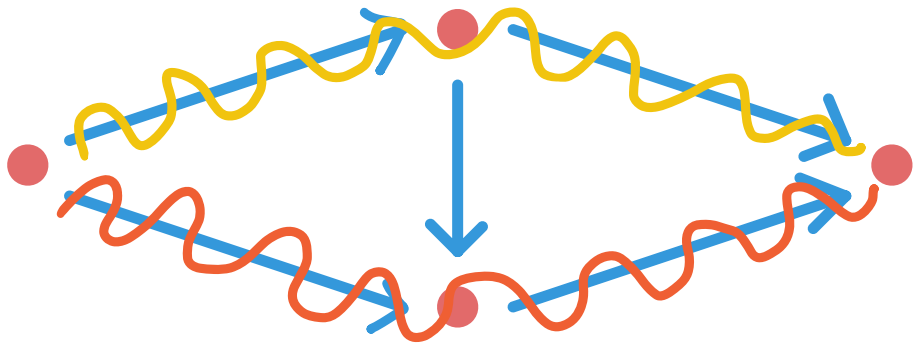
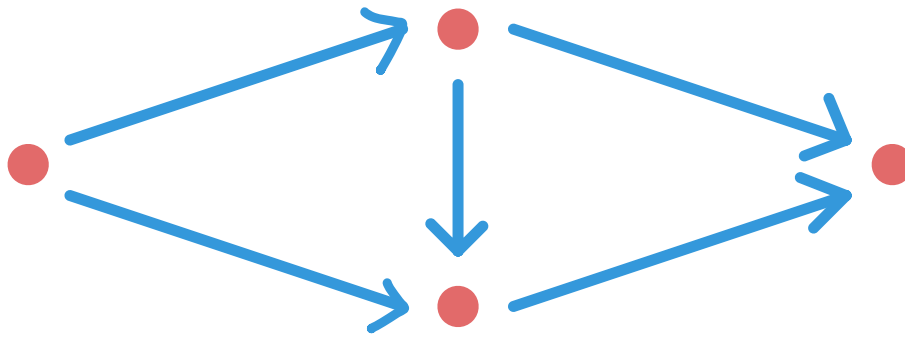


(bad)

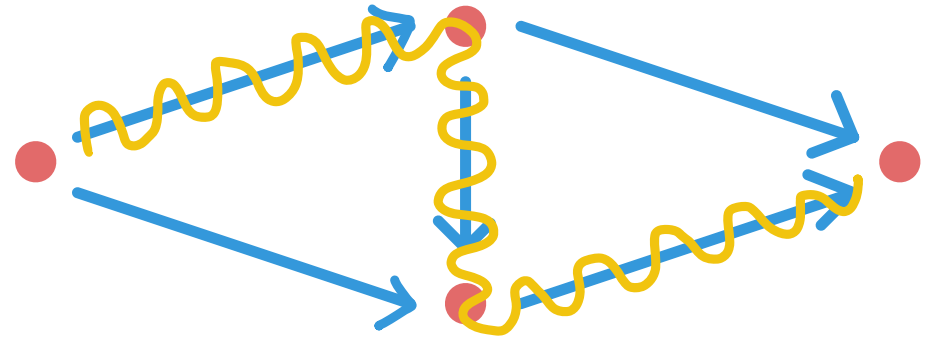




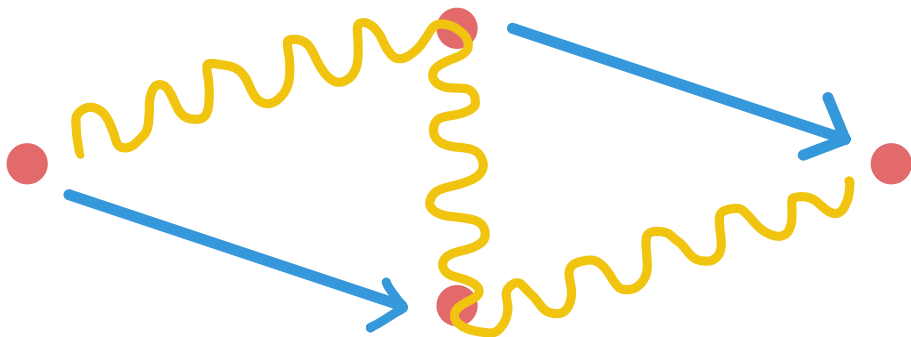




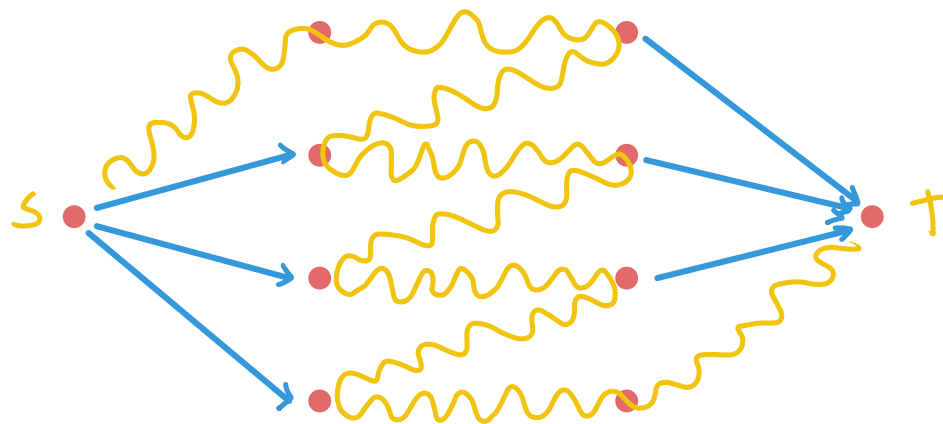
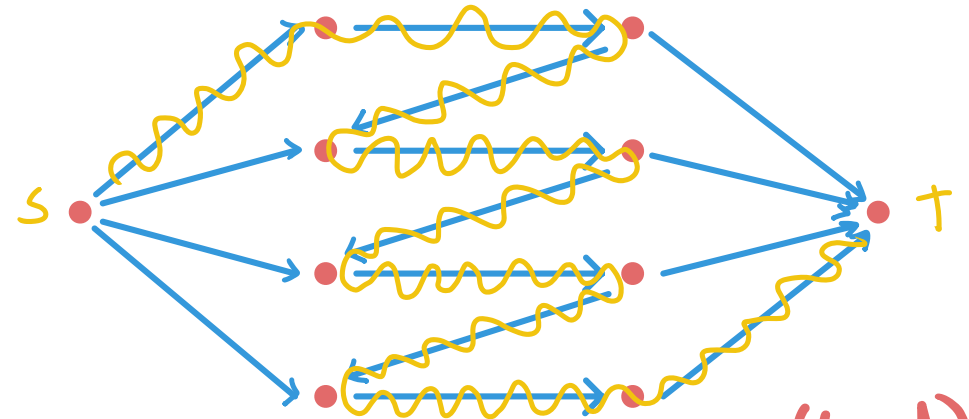
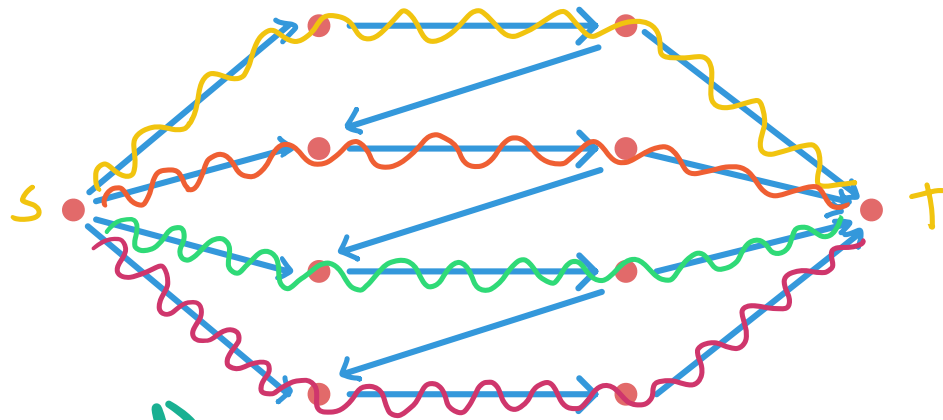
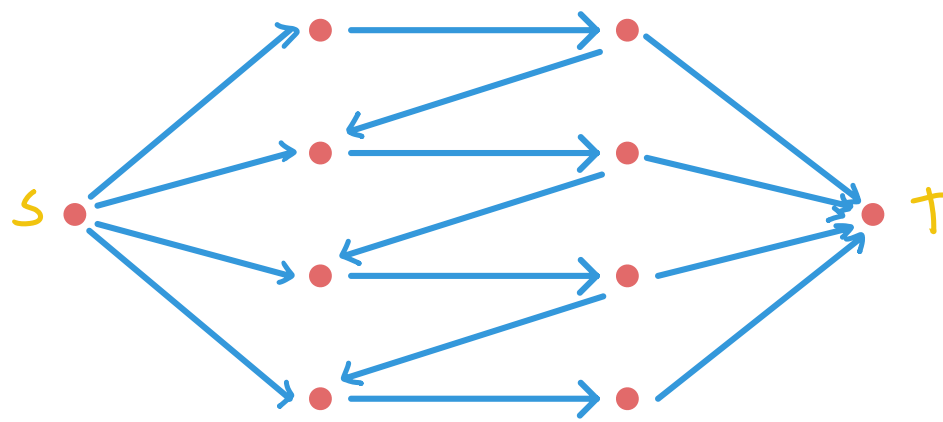
(good)



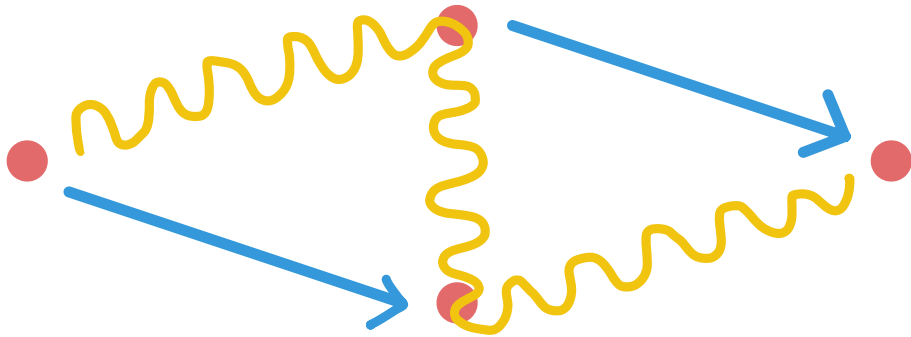
(bad)



① while there is a path $p: s \rightsquigarrow t$
 @ take p and remove all its edges from the graph



① while there is a path $p: s \rightsquigarrow t$
 @ take p and remove all its edges from the graph

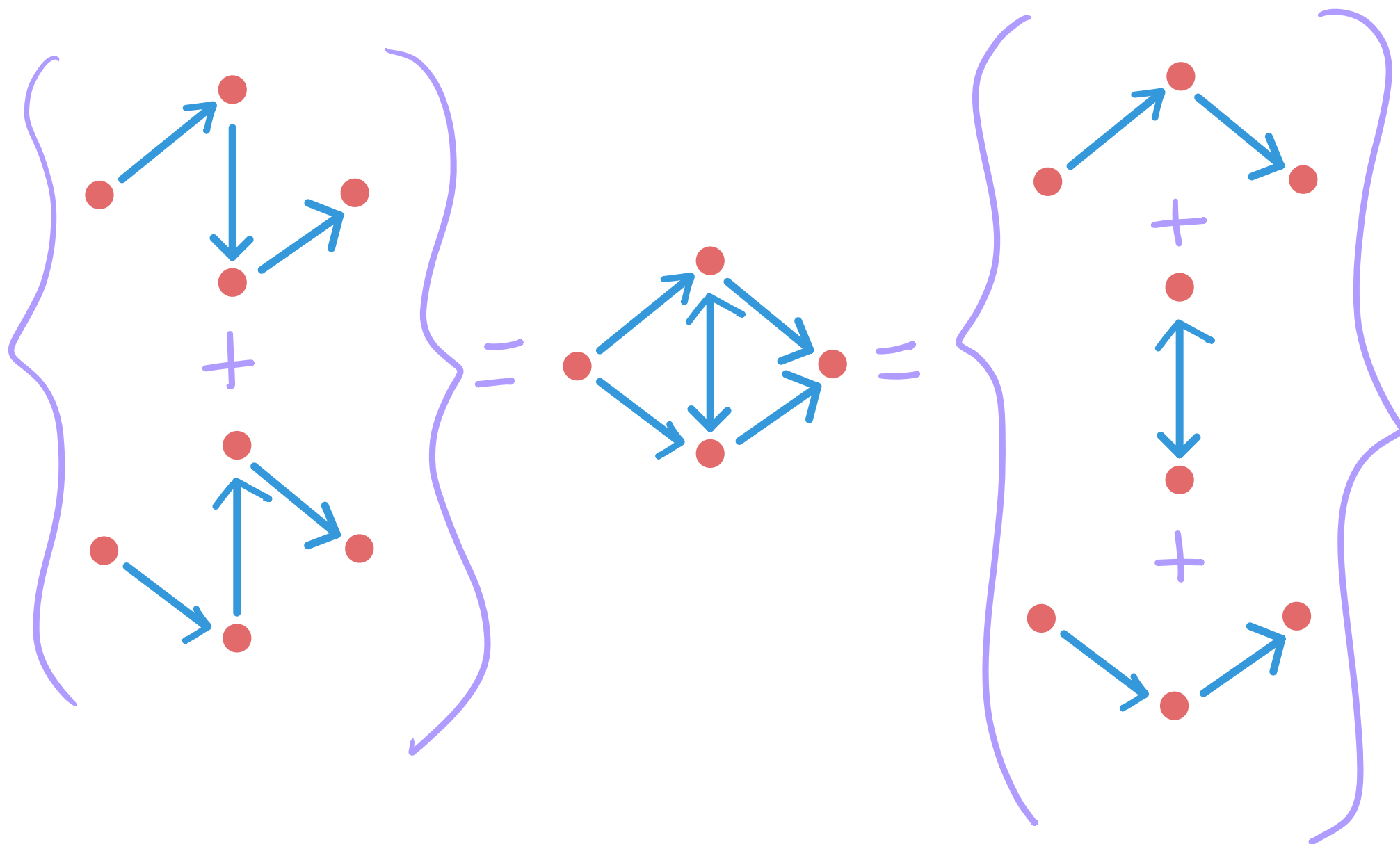


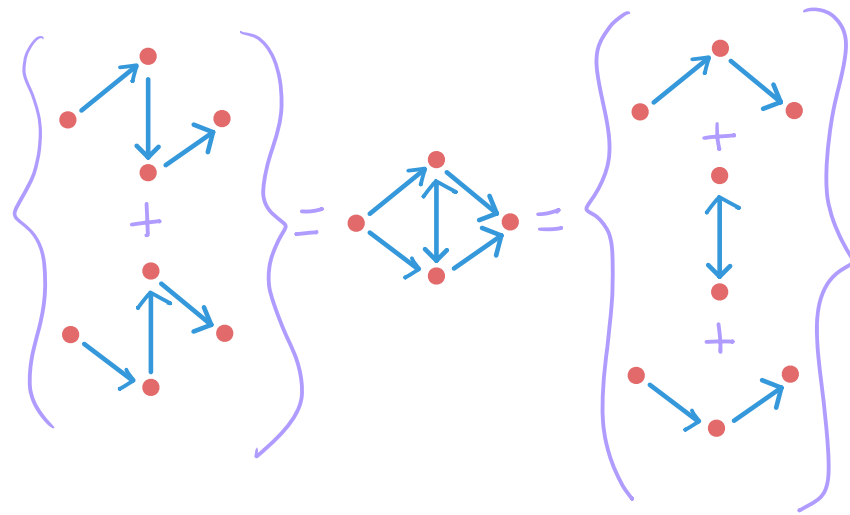
- ① while there is a path $p: s \rightsquigarrow t$
 - @ take p and remove all its edges from the graph

logic: ripping out paths ensures paths are disjoint

is this necessary?

Consider:

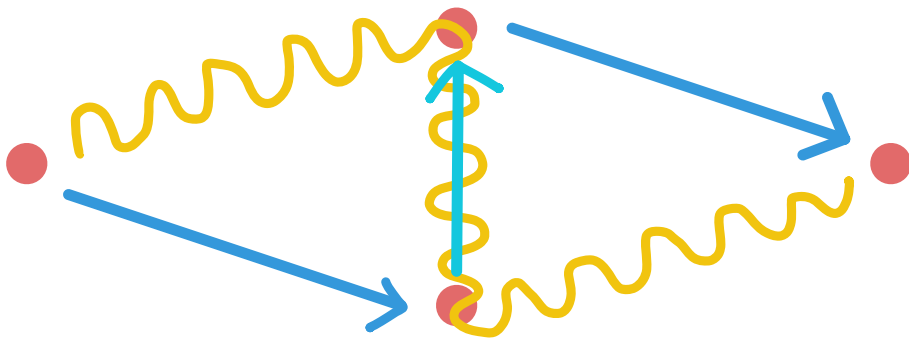




"opposite edges"

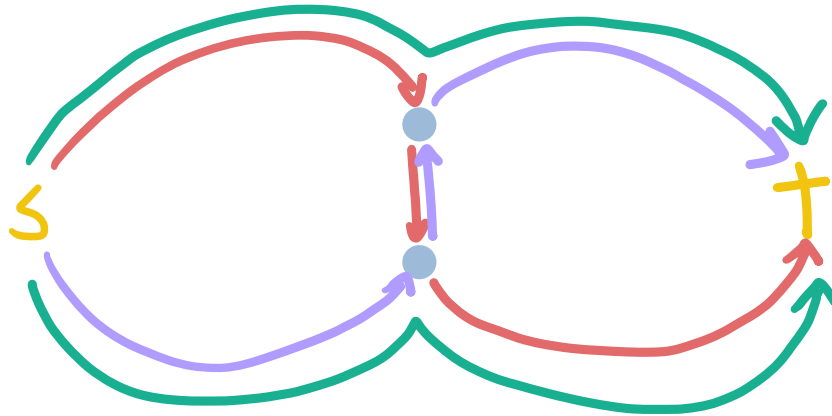


"cancel out"



we should be allowed
to take reverse edges
in subsequent paths

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



then there are edge-disjoint paths $p', q': s \rightsquigarrow t$
without any opposite pairs of edges

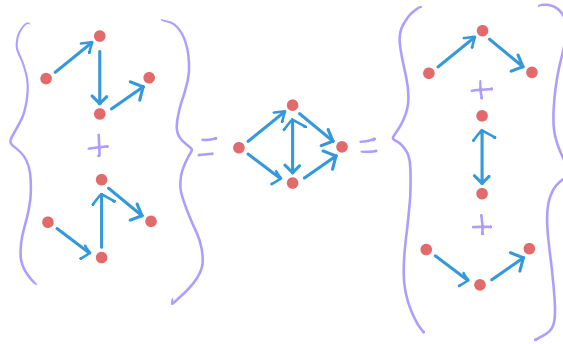
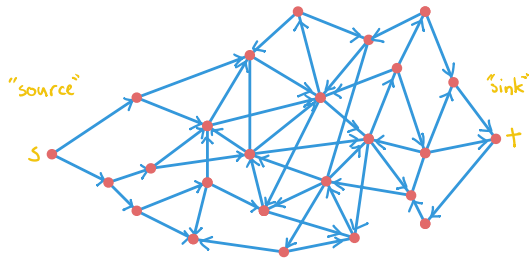
Proof: try it!

hint: by induction on # of opposite pairs

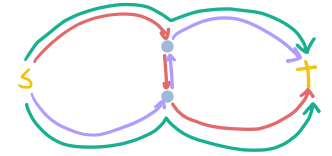
(work space)

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 are edge-disjoint paths $p', q' : s \rightsquigarrow t$ without any opposite pairs of edges

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

/ Ford and Fulkerson [FF56]*

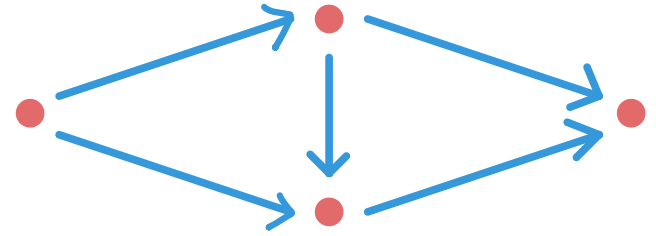
1. $P \leftarrow \emptyset$

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.

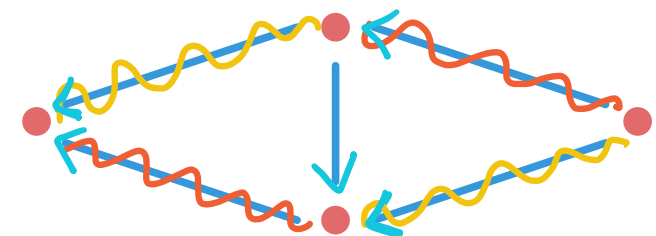
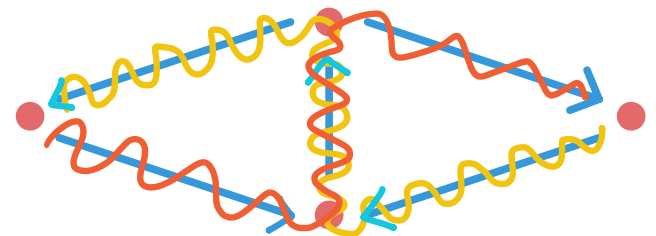
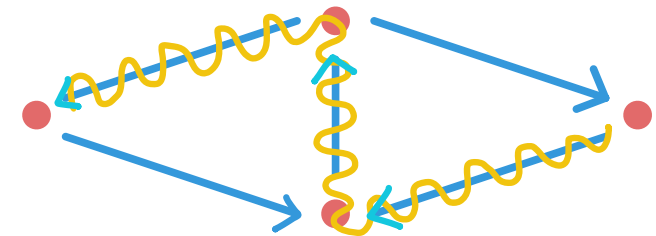
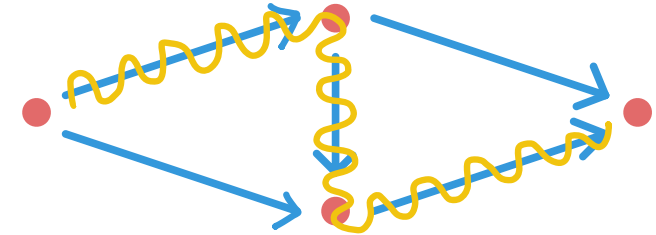
C. Uncross p with previously selected paths, and add p to P .



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

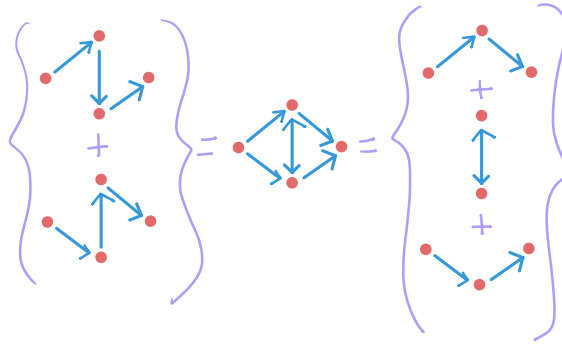
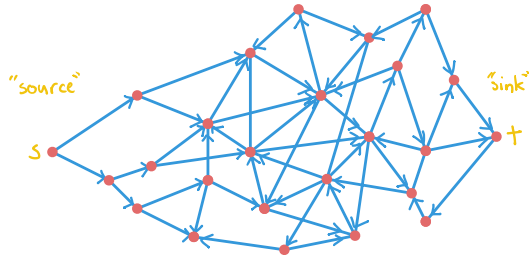
/ Ford and Fulkerson [FF56]*

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 .

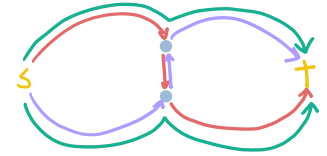


How to compute

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



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



then there are edge-disjoint paths $p', q' : s \rightsquigarrow t$ without any opposite pairs of edges

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.

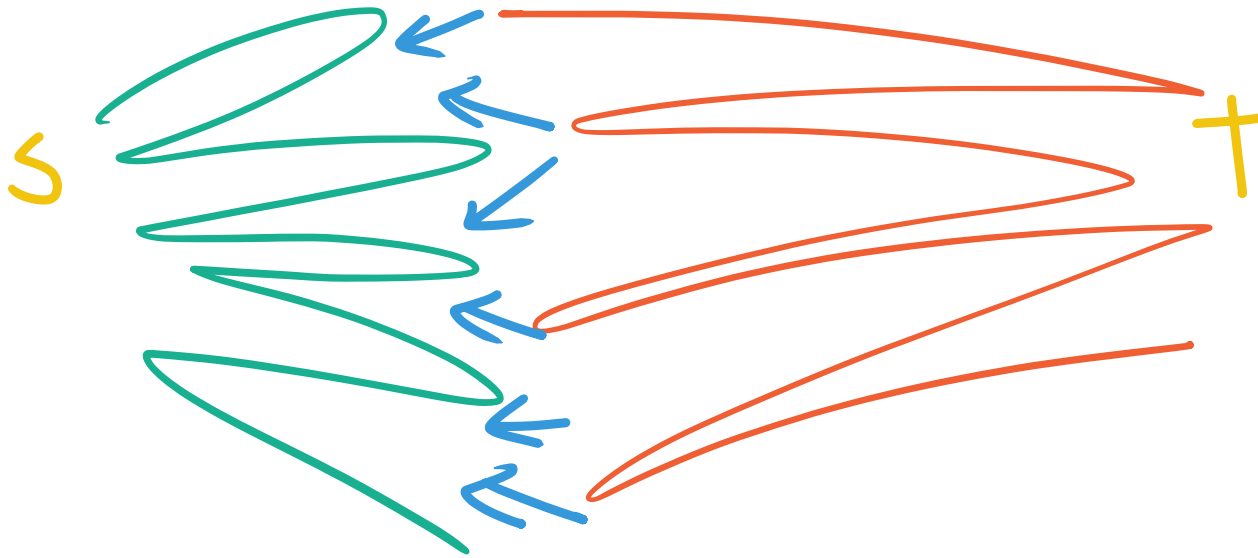
C. Uncross p with previously selected paths, and add p to P .

"residual graph"



Proving optimality

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



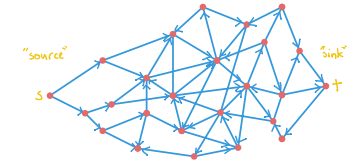
let S = vertices reachable from s in residual graph

$$I = V \setminus S$$

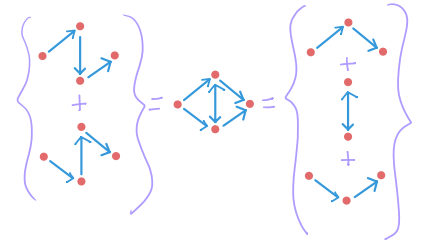
$$C = \delta^+(S) = \text{edges from } S \text{ to } T$$

How to compute

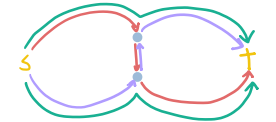
⑦ max # of edge-disjoint $s \rightsquigarrow t$ paths?



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



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

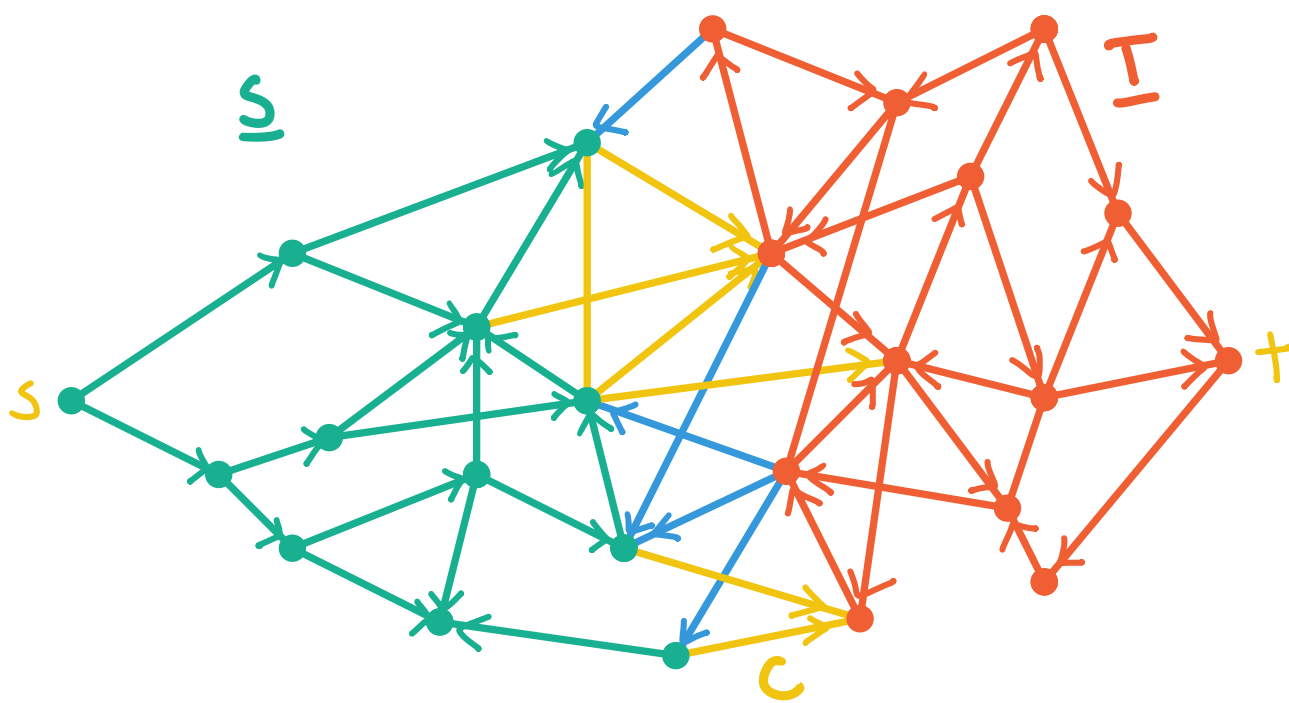


then there are edge-disjoint paths $p', q' : s \rightsquigarrow t$
without any opposite pairs of edges

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 .



over the course of the algorithm:

① each $s \rightsquigarrow t$ path has (exactly) 1 more edge from $\underline{S}$ to $\underline{I}$ than from $\underline{I}$ to $\underline{S}$

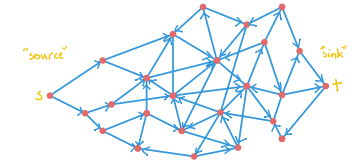
② each $s \rightsquigarrow t$ path decreases $|C|$ by 1

③ At the end, $|C| = 0$.

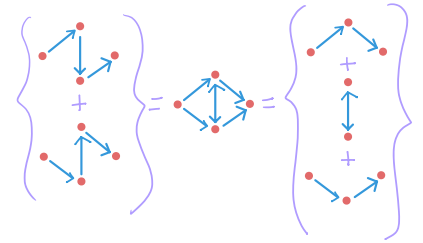
$$\Rightarrow (\# \text{ paths packed}) = (\# \text{ edges in } C)$$

How to compute

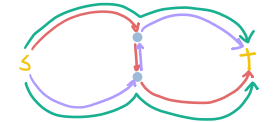
⑦ max # of edge-disjoint $s \rightsquigarrow t$ paths?



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



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



then there are edge-disjoint paths $p', q' : s \rightsquigarrow t$ without any opposite pairs of edges

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 .

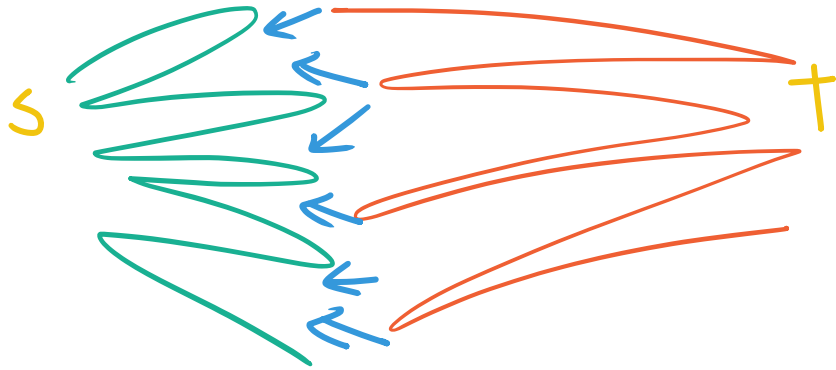
$\underline{S}$ = vertices reachable from s in residual graph

$\underline{I} = V \setminus \underline{S}$

$C = \delta^+(S)$
= edges from S to T

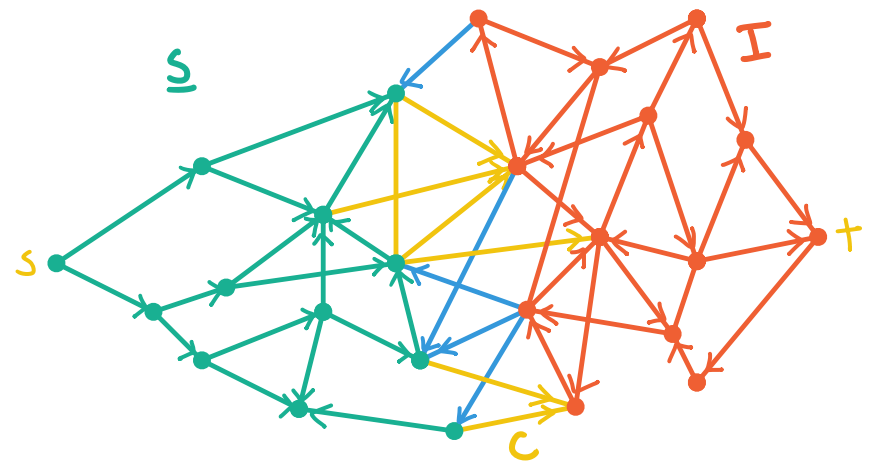
Proving optimality

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, $|C| = 0$.

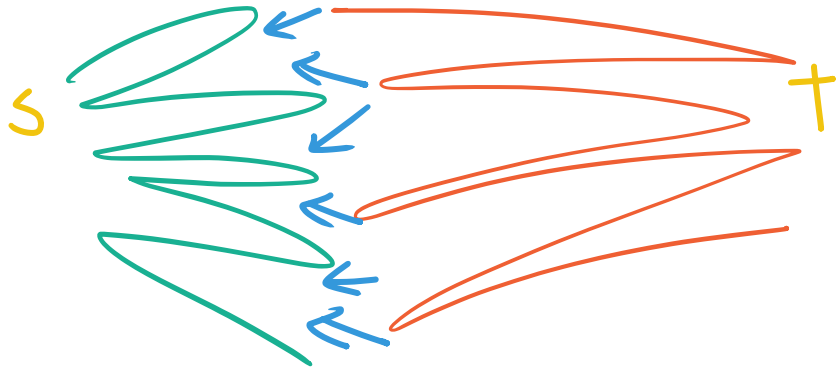
$$\Rightarrow (\text{\# paths packed}) = (\text{\# edges in } C)$$

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

equal

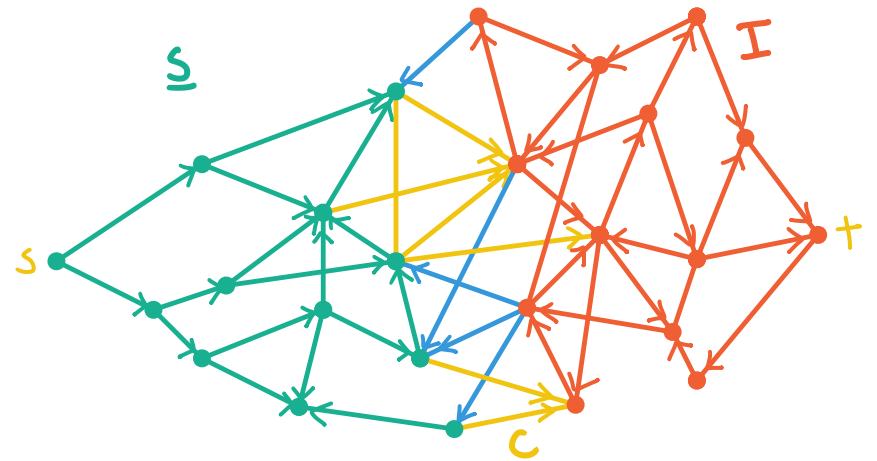
Proving optimality

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



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

$C = \delta^+(S) =$ edges from S to T



over the course of the algorithm:

- ① each $s \rightarrow t$ path has (exactly) 1 more edge from S to I than from I to 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

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

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

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

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

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

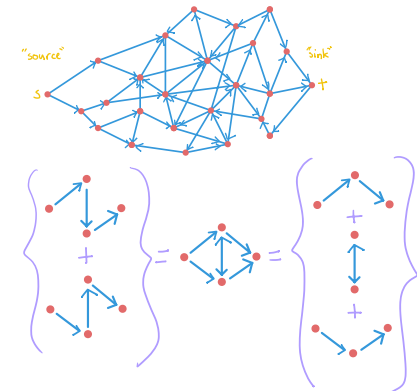
/ Ford and Fulkerson [FF56]*

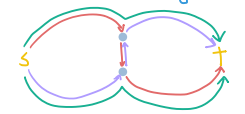
1. $P \leftarrow \emptyset$
2. While there is an (s, t) -path $p : s \rightsquigarrow t$:
 - A. Remove all of p 's edges from the graph.
 - B. Add the reverse of all of p 's edges to the graph.
 - C. Uncross p with previously selected paths, and add p to P .³
3. $S \leftarrow \{\text{vertices still reachable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

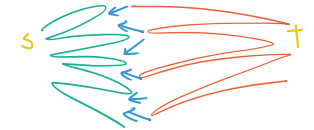
$|\partial^+(S)| = |P|$, so both are optimal

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



- ① $(\max \# \text{ disjoint } s \rightsquigarrow t \text{ paths}) = (\min \# \text{ edges in any } s \rightsquigarrow t \text{ cut})$
- ② $(\# \text{ paths packed}) = (\max \# \text{ disjoint } s \rightsquigarrow t \text{ paths})$
- ③ $(\min \# \text{ edges in any } s \rightsquigarrow t \text{ cut}) = (\# \text{ edges in } C)$

Running time let $\lambda = \text{OPT}$

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

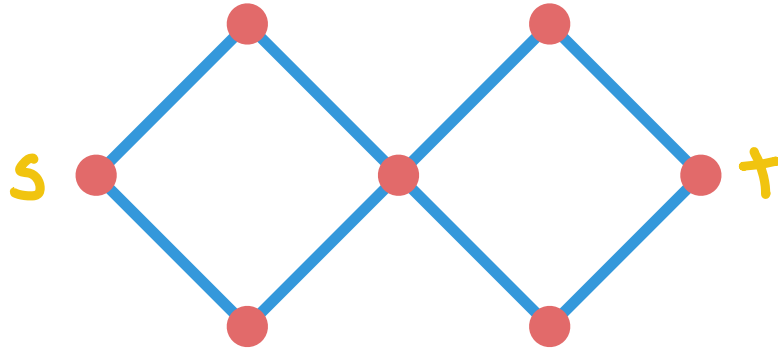
/ Ford and Fulkerson [FF56]*

**/*

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 .³
3. $S \leftarrow \{\text{vertices still reachable from } s\}$.
4. Return the (s, t) -path packing P and the (s, t) -cut $\partial^+(S)$.

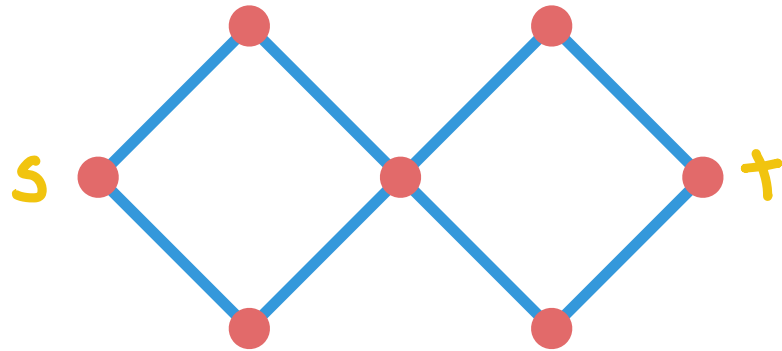
$O(m)$ per iteration $\times O(\lambda)$ iterations = $O(m\lambda)$ time

So we have $O(m\lambda)$ algorithm for max edge-disjoint paths
what about max vertex-disjoint paths?

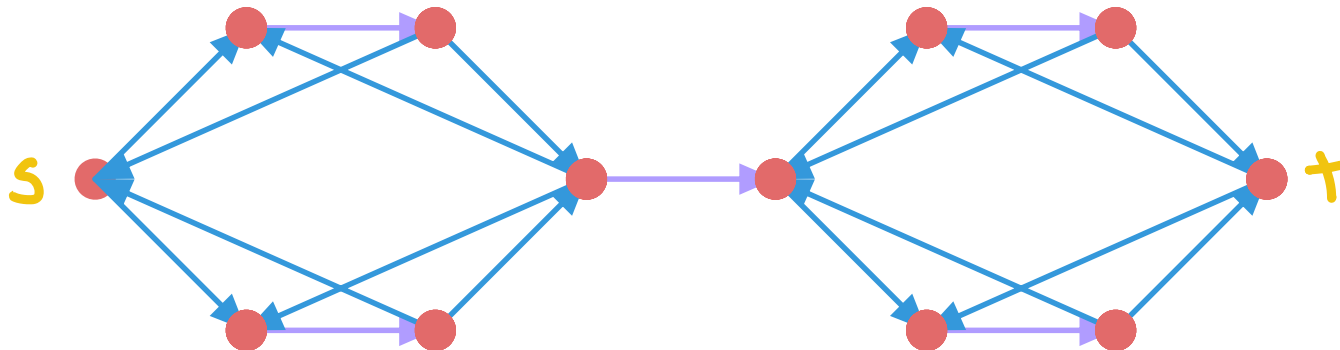


(2 edge-disjoint paths, 1 vertex disjoint path)

So we have $O(m\lambda)$ algorithm for max edge-disjoint paths
what about max vertex-disjoint paths?



(2 edge-disjoint paths, 1 vertex disjoint path)

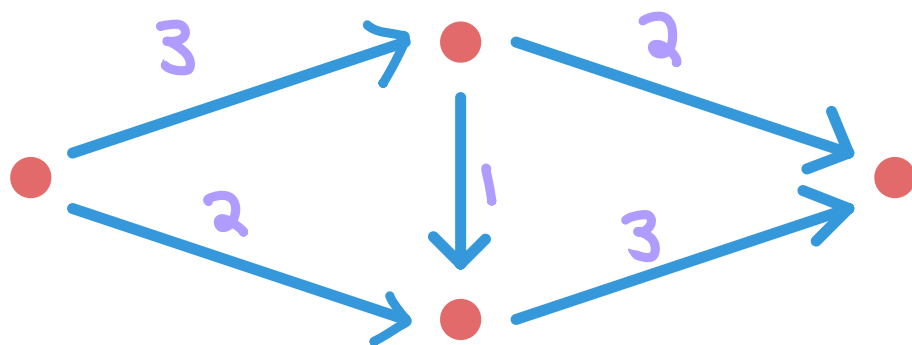


1. $P \leftarrow \emptyset$
2. While there is an (s, t) -path $p: s \rightsquigarrow t$:
 - A. Remove all of p 's edges from the graph.
 - B. Add the reverse of all of p 's edges to the graph.
 - C. Uncross p with previously selected paths, and add p to P .³
3. $S \leftarrow \{\text{vertices still reachable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



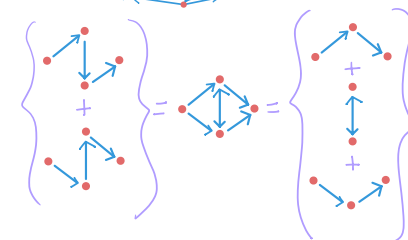
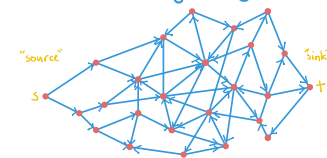
capacity $c(e) > 0$: reuse edge e up to $c(e)$ times

- 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

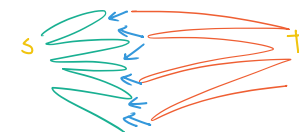
⑦ max # of edge-disjoint $s \rightsquigarrow t$ paths?



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



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

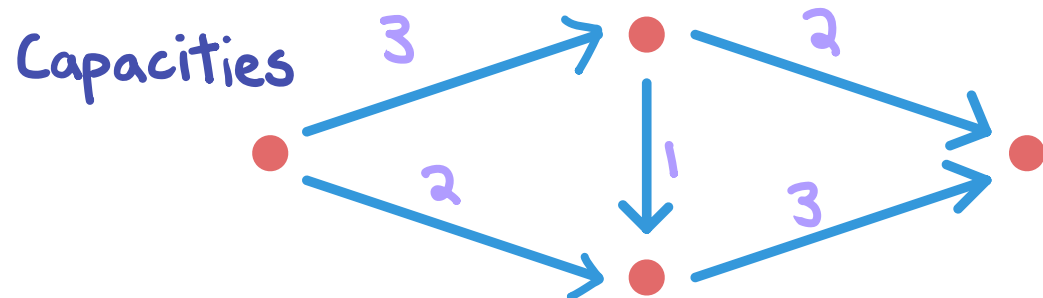
$$\textcircled{2} \left(\begin{array}{l} \text{\# paths} \\ \text{packed} \end{array} \right) = \left(\begin{array}{l} \text{max \#} \\ \text{disjoint} \\ s \rightsquigarrow t \text{ paths} \end{array} \right)$$

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

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

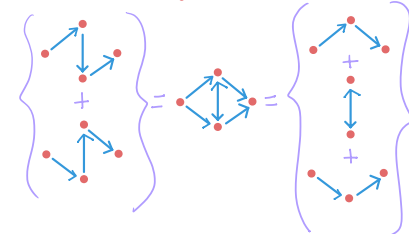
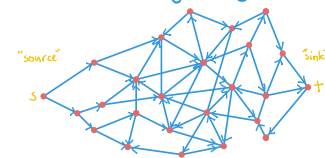
• if $c(e)$ not integral (and just real-valued) what about max fractional path packing

Good news:

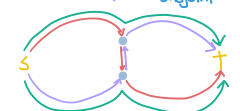
Our duality theorems still hold

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



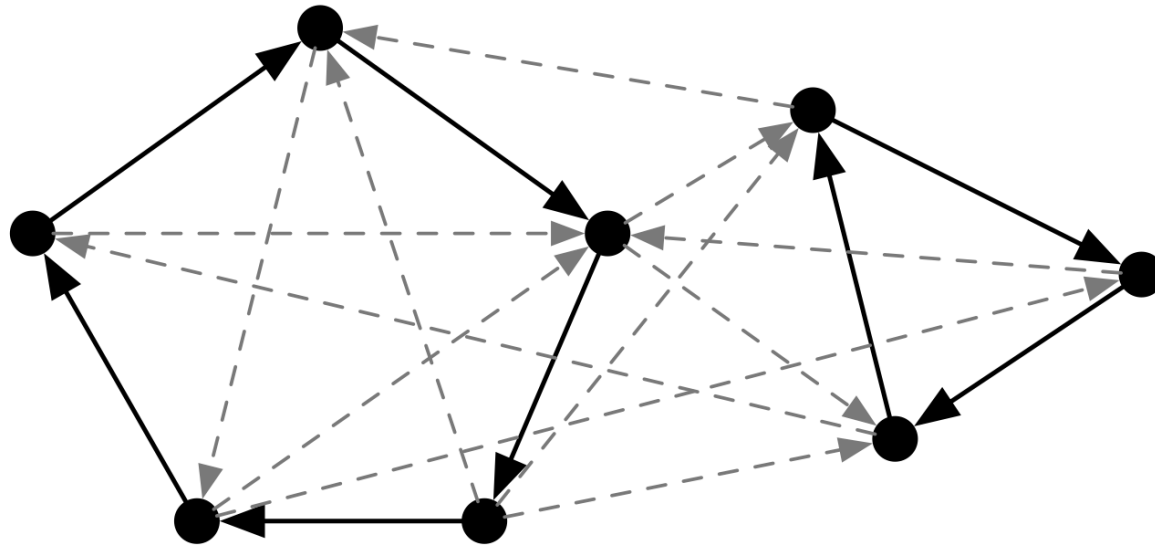
$$\textcircled{1} \left(\begin{array}{c} \text{max \#} \\ \text{disjoint} \\ s \rightsquigarrow t \text{ paths} \end{array} \right) = \left(\begin{array}{c} \text{min \#} \\ \text{edges in any} \\ s \rightsquigarrow t \text{ cut} \end{array} \right)$$

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

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

remaining issues algorithmic ~ next time ~

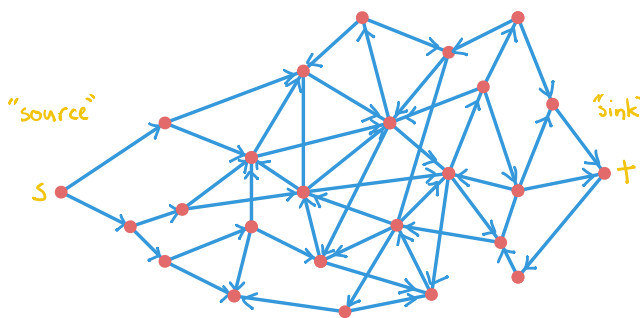
Exercise 19.10. A *cycle cover* of a given directed graph $G = (V, E)$ is a set of vertex-disjoint cycles that cover every vertex in G . You can also think of a cycle cover as a subset of edges $F \subseteq E$ such that each vertex v has exactly one edge in F entering v , and exactly one edge in F leaving v . This implies that F has exactly n edges. Below is a directed graph where the solid arcs form a cycle cover, and the dashed arcs are the remaining edges in G .



Consider the problem of deciding if a directed graph G contains a cycle cover. 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.

Chapter 17

Packing and covering paths



Let $G = (V, E)$ be a directed graph¹, and let $s, t \in V$ be two distinct vertices. We call s the *source* and t the *sink*. A *path packing* is a collection of edge disjoint paths. An (s, t) -path packing is a path packing of (s, t) -paths. The *maximum (s, t) -path packing problem*² is to

find a maximum cardinality packing of (s, t) -paths.

An (s, t) -cut is a set of edges whose removal leaves no paths from s to t . The *minimum (s, t) -cut problem* is to

find the minimum cardinality (s, t) -cut.

Both can be understood as a generalization of reachability, discussed before. Reachability was concerned with whether there is a single connection from s to t . Both the maximum flow and minimum cut problems measure the *strength* of the connection from s to t .

¹ G is allowed to be a *multi-graph*, with multiple copies of the same edge.

²This problem is also called the *uncapacitated maximum (s, t) -flow problem* for reasons we discuss later.

We are interested in these questions both algorithmically and (graph-)structurally. Algorithmically the problems are highly non-trivial, as there are exponentially many possible paths from s to t to take into account. It is not obvious that there is a polynomial time algorithm for either problem.

17.1 Duality

The (s, t) -path packing and (s, t) -cut problem are *dual* packing and covering problems, in the following sense. We are selecting paths that “pack in” to the edges of the graph – each path uses up all of its edges. Conversely, an (s, t) -cut must contain at least one edge from every (s, t) -path. That is, we are trying to “cover” the paths with edges, where we interpret each edge as a set that covers all the (s, t) -paths that contain that edge. In short:

paths pack into edges, and edges cover paths.

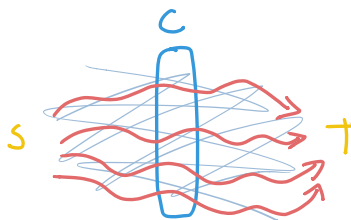
As with any packing and covering problem, we have the following inequality.

$$(\max (s, t)\text{-path packing}) \leq (\min (s, t)\text{-cut}). \quad (17.1)$$

Indeed, let P denote a path packing, and let $C \subseteq E$ be an (s, t) -cut. We have

$$|P| = \sum_{p \in P} 1 \stackrel{(a)}{\leq} \sum_{p \in P} |p \cap C| \stackrel{(b)}{\leq} |C|.$$

Above, we treat each path $p \in P$ as a subset of edges. (a) is because, as an (s, t) -cut, C contains at least one edge from every (s, t) -path. (b) is because the paths $p \in P$ are edge disjoint, so the sets $p \cap C$ over $p \in P$ are also disjoint. The following is a conceptual sketch of our argument.



The inequality (17.1) inspires some basic questions. Is the inequality ever equal? Is the inequality ever strict?

17.1.1 Greedy heuristics

Consider now the computational problem of finding the maximum (s, t) -path packing, or finding the minimum (s, t) -cut. For packing, consider the obvious approach of repeatedly taking (s, t) -paths, and adding it to a growing collection.

greedy-path-packing($G = (V, E), s, t$)

1. $P \leftarrow \emptyset$.
2. While there is a path p in G from s to t : // BFS or DFS
 - A. Add p to P .
 - B. Remove all the edges in p from the graph.
3. Return P .

An equally simple and natural approach to the (s, t) -cut problem is as follows. Start with $C = E$; obviously C is an (s, t) -cut. Then repeatedly try to remove edges from C as long as C remains an (s, t) -cut.

greedy-cut($G = (V, E), s, t$)

1. $C \leftarrow E$.
2. While there is an edge $e \in C$ such that $C - e$ is an (s, t) -cut
 - A. Remove e from C .
3. return C .

Ideally, we would like to show that these algorithms return the optimum solutions of their respective problems. For starters, it is clear from their terminating conditions that they produce *locally optimal* solutions. **greedy-path-packing** terminates when no more paths can be added to the current solution. **greedy-cut** terminates when no more edges can be removed from the current solution. That is:

Lemma 17.1. *greedy-path-packing returns a maximal (s, t) -path packing P , and greedy-cut returns a minimal (s, t) -cut.*

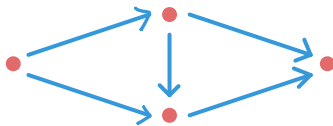
Tautologically, we have

$$(\text{maximal path packing}) \leq (\text{maximum path packing})$$

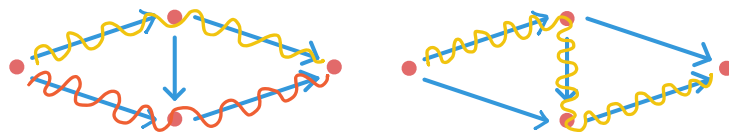
and

$$(\text{minimal } (s, t)\text{-cut}) \geq (\text{minimum } (s, t)\text{-cut}).$$

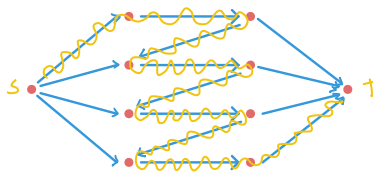
But these relations are only useful if we can strengthen the inequalities to equalities. Unfortunately, simple counter examples show that they are not equal. Consider the following graph.



Let s be the leftmost vertex and t the rightmost vertex. Clearly the maximum packs two paths from left to right, by going along the top and along the bottom. But a single path using the middle edge also gives a maximal path packing.



Thus a maximal path packing is not a maximum path packing. It is not hard to find simple counter examples that exaggerate the difference in size between a maximal and maximum path packing.



It is also easy to come up with counter examples that show that a minimal (s, t) -cut is not a minimum (s, t) -cut.

The point is that

$$(\text{maximal path packing}) \neq (\text{maximum path packing})$$

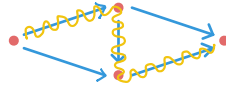
and

$$(\text{minimal } (s, t)\text{-cut}) \neq (\text{minimum } (s, t)\text{-cut}).$$

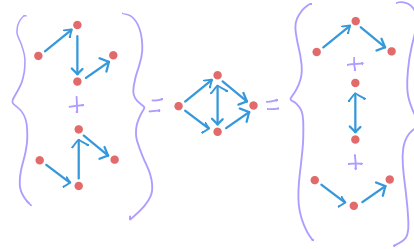
This is different from our previous discussion on spanning trees (Chapter 16). There, one can show that any maximal forest is a maximum forest, and that any minimal spanning set is a minimum spanning set. Moreover, the optimal solution in both problems have the same value, meeting at a spanning forest. This makes the spanning tree problems conducive to greedy algorithms where we simply keep adding edges to a forest, or keep removing edges from a spanning set, until we can no longer do so while maintaining feasibility. But such simple greedy algorithms apparently do not work for (s, t) -flow and (s, t) -cut.

17.2 Augmenting paths

Recall our very simple counter example for the greedy path packing approach.



If we remove the selected path (in yellow), the two unused edges are separated and so there is no path remaining. But consider the following “path arithmetic”:

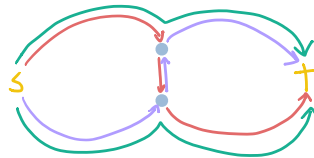


Call two directed edges *opposite edges* if they have are the reverse of one another. The two paths on the LHS have a pair of opposite edges. We can “uncross” the two paths at the opposite edges, combining the initial part of one path with the remaining part of the other. The result is another path packing except now there are no opposite pairs of edges. This intuition leads to the following observation.

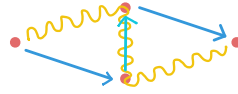
Lemma 17.2. *Let p, q be two edge-disjoint (s, t) -paths. Then there are two edge disjoint (s, t) -paths p', q' with the following properties:*

1. p' and q' use a subset of the edges in $p \cup q$.
2. p' and q' have no opposite pairs of edges.

The proof is by induction on the number of opposite pairs of edges, and left to the reader.



Knowing now that we can uncross opposite pairs of edges and maintain a path packing, consider again our simple counter example. The problem before is that we used the middle edge going down, leaving a disconnect in the middle of the graph. But suppose we inserted a new edge in the opposite direction, going from bottom to top.



The new (s, t) -path in this augmented, *residual graph* can be uncrossed with the previously selected path to obtain two paths in the original graph. This clever idea leads to the following algorithm, due to Ford and Fulkerson [FF56]. On first pass, the reader should ignore (3) and focus on the path packing portion of the algorithm. Below, $\delta^+(S) \stackrel{\text{def}}{=} \{(u, v) \in E : u \in S, v \notin S\}$ denotes the “out-cut” of edges leaving S .

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

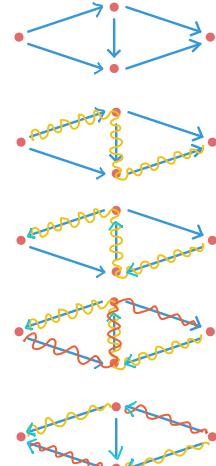
/ Ford and Fulkerson [FF56]*

**/*

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 .³
3. $S \leftarrow \{\text{vertices still reachable from } s\}$.
4. Return the (s, t) -path packing P and the (s, t) -cut $\delta^+(S)$.

³Strictly speaking, after uncrossing, we may end up with a collection of walks rather than paths. But any feasible “walk-packing” can be converted into a path-packing by shortcutting any cycles that may arise.

A simulation of **augmenting-paths** on our simple counterexample is given on the right. Here we start with a simple graph, and in the first iteration take the “bad path” in yellow. We then add the reverse edges, indicated by the new arrows in teal. With these new edges we now have another path, using the middle edge in reverse, from left to right. We update the residual graph as reflected by the updated teal arrows. In the last residual graph, there is no (s, t) -path remaining.

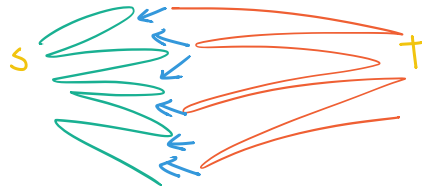


We remark that one can skip steps (1), (2.C), and (3) (and instead count the number of iterations of (2)) if the goal is only to compute the size of the maximum path packing.

Next we will argue that **augmenting-paths** returns an optimum solution. To analyze **augmenting-paths** we use the following definitions. First, we define the *residual graph* of a path packing as the graph obtained by removing the paths from the graph, and adding their reverse. The **augmenting-paths** algorithm maintains a residual graph of the current path-(or walk)-packing at any point in time. The key to the proof is the following observation.

Observation 17.3. *Let $S \subset V$ be any set of vertices with $s \in S$ and $t \notin T$. Let $p : s \rightsquigarrow t$ be an (s, t) -path. Then p has exactly one more edge for $\delta^+(S)$ than from $\delta^-(S)$. In particular, the residual graph w/r/t p decreases $|\delta^+(S)|$ by 1 and increases $|\delta^-(S)|$ by 1.*

The **augmenting-paths** algorithm terminates only when the residual graph, here denoted G' , has no (s, t) -path. Let S denote the set of vertices reachable from S in the final residual graph, and let $T = V \setminus S$ denote the remaining vertices. Note that $s \in S$ and $t \in T$. Below, S is in green and T is in red.



Let $k = \delta^+(S | G)$ number of edges of the out cut of S in the original input graph. Let us observe the changes to $\delta^+(S | G)$ over the course of the **augmenting-paths** algorithm. Each iteration, we increase our path packing by one, and the size of $\delta^+(S)$ decreases by one in the residual graph. Thus the algorithm terminates in exactly k iterations, with $|P| = k$. But we already know, from duality, that any path packing has at most k paths. Thus the path packing P produced by the **augmenting-path** algorithm is certifiably optimal. Moreover, since the size of any (s, t) -path packing is

a lower bound for the size of any (s, t) -cut, we simultaneously obtain the fact that $\delta^+(S)$ is a minimum (s, t) -cut.

We have now accomplished two things. First, we have established a mathematical connection between the maximum number of edge-disjoint paths and the minimum number of edges in a (s, t) -cut. This is called Menger's theorem.

Theorem 17.4 (Menger's theorem). *In any directed graph with integer capacities, and fixed source and sink, we have*

$$(\text{maximum } (s, t)\text{-path packing}) = (\text{minimum } (s, t)\text{-cut}).$$

In particular, there exists an integral maximum flow.

Second, we have obtained an algorithm computing the maximum (s, t) -path packing and minimum (s, t) -cut with the following guarantee.

Theorem 17.5 ([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 `augmenting-paths`(G, s, t) produces both a maximum packing and a minimum cut in $O(m\lambda)$ time.*

Proof. We have already shown the correctness of the algorithm. For the running time, each successive path can be obtained by BFS or DFS. By maintaining the paths as doubly linked lists, it is easy to “uncross” the selected paths in $O(n)$ time per path. $\square$

For uncapacitated graphs, λ is at most m , so `augmenting-paths` is polynomial time. However, the next chapter will consider the generalization where edges have capacities $c : E \rightarrow \mathbb{R}_{>0}$, that allow us to reuse an edge e up to $c(e)$ times. Here λ may be exponentially large, and this running time is no longer satisfactory.

17.3 Multicommodity flow

We now consider a generalization to path packing with multiple source sink pairs. This is sometimes called *integral multicommodity flow*; here we will call it multicommodity path packing to emphasize the combinatorial nature and distinguish from the continuous version of the problem that we have not discussed.

Let $(s_1, t_1), \dots, (s_k, t_k)$ be k source-sink pairs. For each i , let $d_i \in \mathbb{N}$ be a given demand. The goal is to find a path packing that, for each i , contains d_i (s_i, t_i) -paths. The (s, t) -path packing discussed earlier is a special case with $k = 1$.

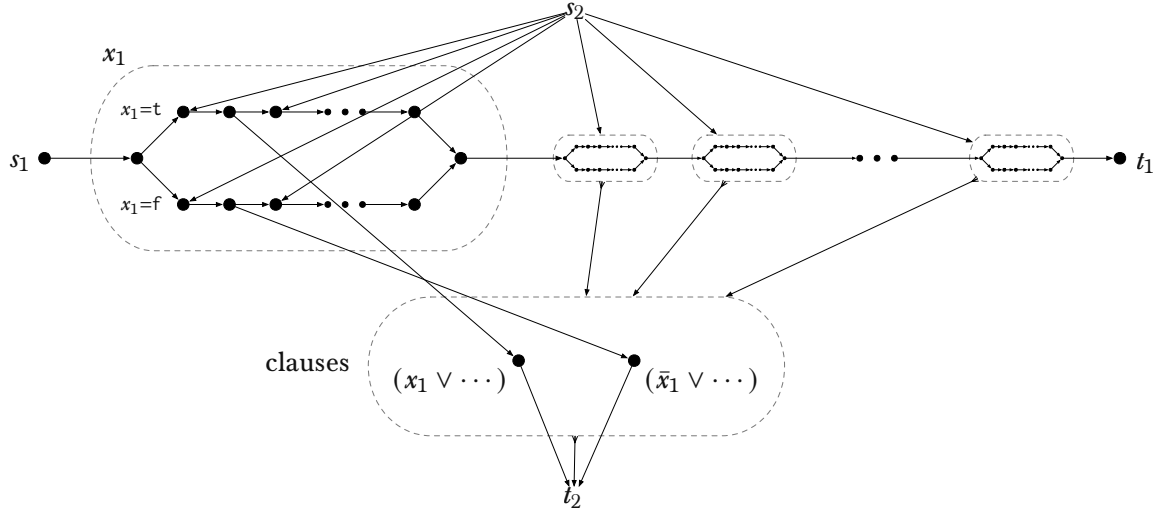


Figure 17.1: Reduction from SAT to multicommodity path packing, with two commodities and a simple directed graph.

Theorem 17.6. *The integral multicommodity flow problem is NP-Hard, even in uncapacitated graphs.*

Proof. We present a reduction from 3-SAT by Even, Itai, and Shamir [EIS76].

1. For every variable x_j , we make a gadget that looks like two parallel paths with the same starting point and two end point. One of these paths will encode the assignment $x_j = \text{true}$ and the other will encode the assignment $x_j = \text{false}$.
 - (a) We introduce two vertices A_j^- and A_j^+ .
 - (b) Let $i_1, \dots, i_k$ be the indices of the clauses C_i satisfied by $x_j = \text{true}$. For each index i_ℓ , we create two vertices $B_{i_\ell, j}^-$ and $B_{i_\ell, j}^+$. We add edges to form a path from A_j^- to A_j^+ through the pairs. More explicitly, we add edges from A_j^- to $B_{i_1, j}^-$, from $B_{i_\ell, j}^-$ to $B_{i_\ell, j}^+$ for each i_ℓ , and from $B_{i_\ell, j}^+$ to $B_{i_{\ell+1}, j}^-$ for each $\ell < k$.
 - (c) We similarly construct an (A_j^-, A_j^+) -path for the clauses satisfied by $x_j = \text{false}$.
2. We connect the variable-gadgets by adding edges from A_j^+ to A_{j+1}^- for $j = 1, \dots, n-1$.

3. We create a new source s_1 and a new sink t_1 . We then add edges from s_1 to A_1^- and from A_n^+ to t_1 . Observe that any (s_1, t_1) path has to choose, for each variable x_j , one of the two paths in the gadget for x_j .
4. For each clause C_i , we make variable that we also call C_i . For every variable x_j appearing in C_i , we add an edge from $B_{i,j}^+$ to (the vertex) C_i .
5. We create a new source s_2 and a new sink t_2 . We add edges from s_2 to $B_{i,j}^-$ for all i, j and from C_i to t_2 for all i .
6. We make (s_1, t_1) one commodity with requirement one, and (s_2, t_2) another commodity with requirement m .

While the construction is lengthy to write out in full, with some help from Fig. 17.1, it is easy to see what is going on. To pack m (s_2, t_2) -paths, the solution has to pack one path per clause. Each of these paths goes through the “middle edge” of an assignment that satisfies it. Meanwhile, any (s_1, t_1) -path has to traverse, for each variable x_j , either the “ $x_j = \text{true}$ ” path or the “ $x_j = \text{false}$ ” path. Any feasible solution, based on which paths the (s_1, t_1) -path takes, gives a feasible solution for the SAT formula. Meanwhile a feasible solution maps to an (s_1, t_1) -path, after which it is either pack m paths (s_2, t_2) through all the vertices. We leave the verification of details to the reader. $\square$

Spring 2025 lecture notes. Click on the links below for the following files:

- [Handwritten notes prepared before the lecture.](#)
- [Handwritten notes annotated during the presentation.](#)
- Unfortunately we lost the recording when the iPad ran out of batteries; please see the 2024 lecture instead.

Spring 2024 lecture notes. Click on the links below for the following files:

- [Handwritten notes prepared before the lecture.](#)
- [Handwritten notes annotated during the presentation.](#)
- [Recorded video lecture.](#)

Spring 2022 lecture materials. Click on the links below for the following files:

- [Handwritten notes prepared before the lecture.](#)
- [Handwritten notes annotated during the presentation.](#)
- [Recorded video lecture.](#)

17.4 Exercises

Exercise 17.1. Let $G = (V, E)$ be a directed graph and $s, t \in V$, where we assume there is no edge from s to t . A collection of (s, t) -paths $p_1, \dots, p_k$ are said to be *vertex-disjoint* if the interior vertices (excluding s and t) of the paths are disjoint. A set of vertices $C \subseteq V \setminus \{s, t\}$ is a *vertex (s, t) -cut* if removing C (and all incident edges) from G disconnects s from t . Consider the problems of (a) computing the maximum cardinality collection of vertex disjoint (s, t) -paths, and (b) the minimum vertex (s, t) -cut. For both of these problems, either (a) design and analyze a polynomial time algorithm (the faster the better), or (b) prove that the problem is NP-Hard.

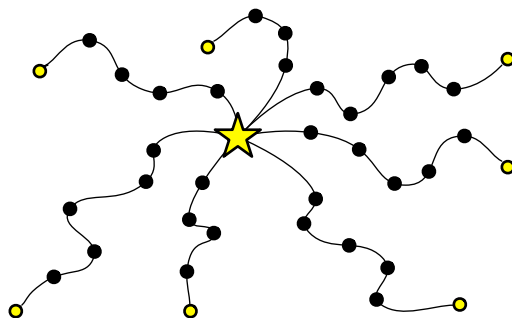
Exercise 17.2. Let $G = (V, E)$ be a directed graph. For $s, t \in V$, let $\lambda(s, t)$ denote the size of the minimum (s, t) -cut. Prove the following “triangle inequality” for cuts:

$$\lambda(a, b) \geq \min\{\lambda(a, c), \lambda(c, b)\}$$

for any three vertices $a, b, c \in V$.

Exercise 17.3. Let $G = (V, E)$ be an undirected graph, and let $a, b, c \in V$ be three distinct vertices. We define an (a, b, c) -*path* as a path from a to c that goes through b . Consider the problem of deciding if there exists an (a, b, c) -path. 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 17.4. A *star* is defined as a tree where at most one vertex has degree > 2 . This special vertex is called the *center* of the star. (A star can have no particular center when it is just a path.)

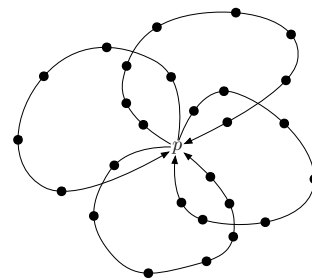


Let $G = (V, E)$ be an undirected graph and $L \subseteq V$ a fixed set of vertices. Here we are interested in stars (as subgraphs of G , and centered anywhere) with as many leaves in L as possible. Consider the problem of computing the maximum number of leaves in S that can be obtained by any star in G . 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 17.5. According to Wikipedia, a pinwheel is a simple child's toy made of a wheel of paper or plastic curls attached at its axle to a stick by a pin. It is designed to spin when blown upon by a person or by the wind.

A similar toy had developed independently in Polynesia (known as pekapeka or pe'ape'a) using either coconut palm leaflets or strips of pandanus leaves;⁴⁵ in colder climates like that of New Zealand (the toy also called pepepe in Māori), phormium leaves are used.⁶

Let $G = (V, E)$ be a directed graph. A *pinwheel* (in G) is defined as a collection of directed cycles all containing a common vertex p and vertex disjoint otherwise. p is called the *pin* of the pinwheel, and each cycle is called a *curl*. Consider the problem of computing the maximum number of curls in any pinwheel. 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.

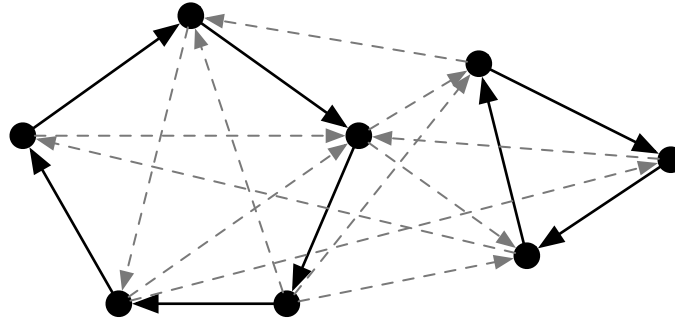


⁴Koch, Gerd (1984). [The Material Culture of Tuvalu](#). Suva: Institute of Pacific Studies, University of the South Pacific. p. 162. ISBN 9820202051.

⁵Te Rangi Hiroa (1930). [Samoan Material Culture](#). Bernice P. Bishop Museum. p. 552.

⁶Beattie, Herries (1994). [Traditional Lifeways of the Southern Maori: The Otago University Museum Ethnological Project, 1920](#). University of Otago Press. p. 68. ISBN 978-0-908569-79-3.

Exercise 19.10. A *cycle cover* of a given directed graph $G = (V, E)$ is a set of vertex-disjoint cycles that cover every vertex in G . You can also think of a cycle cover as a subset of edges $F \subseteq E$ such that each vertex v has exactly one edge in F entering v , and exactly one edge in F leaving v . This implies that F has exactly n edges. Below is a directed graph where the solid arcs form a cycle cover, and the dashed arcs are the remaining edges in G .



Consider the problem of deciding if a directed graph G contains a cycle cover. 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.