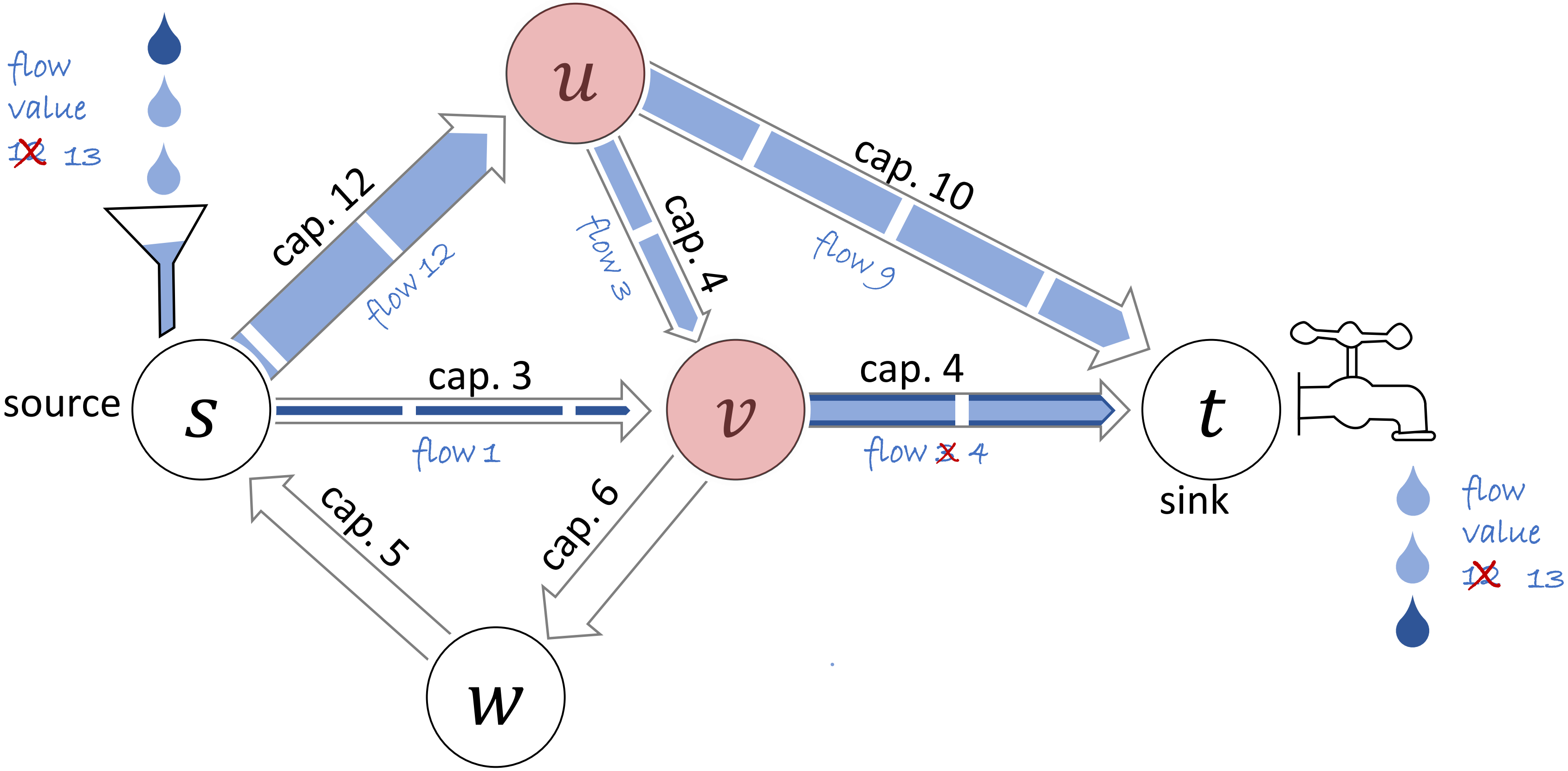
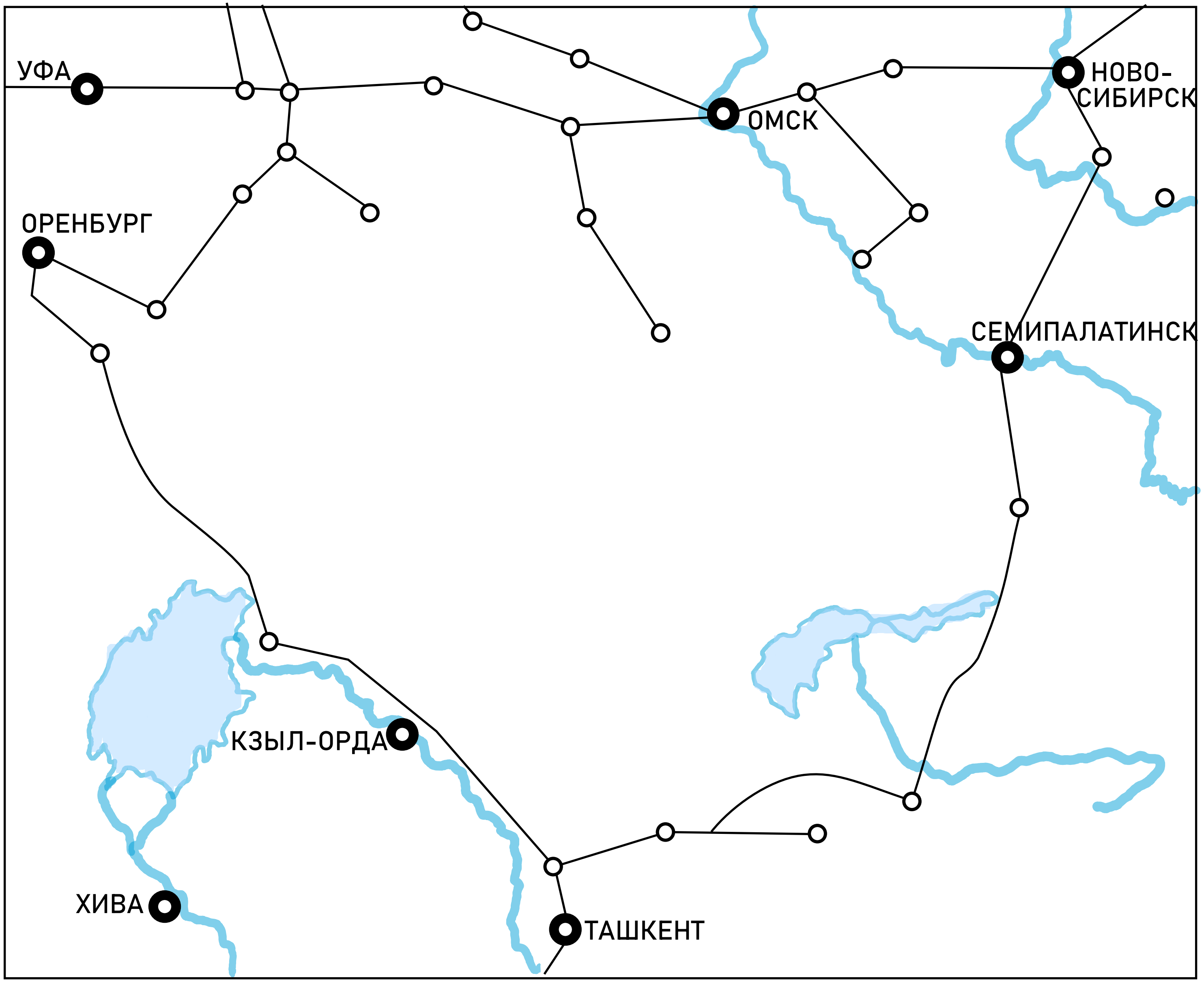
SECTION 6.1

Flow networks



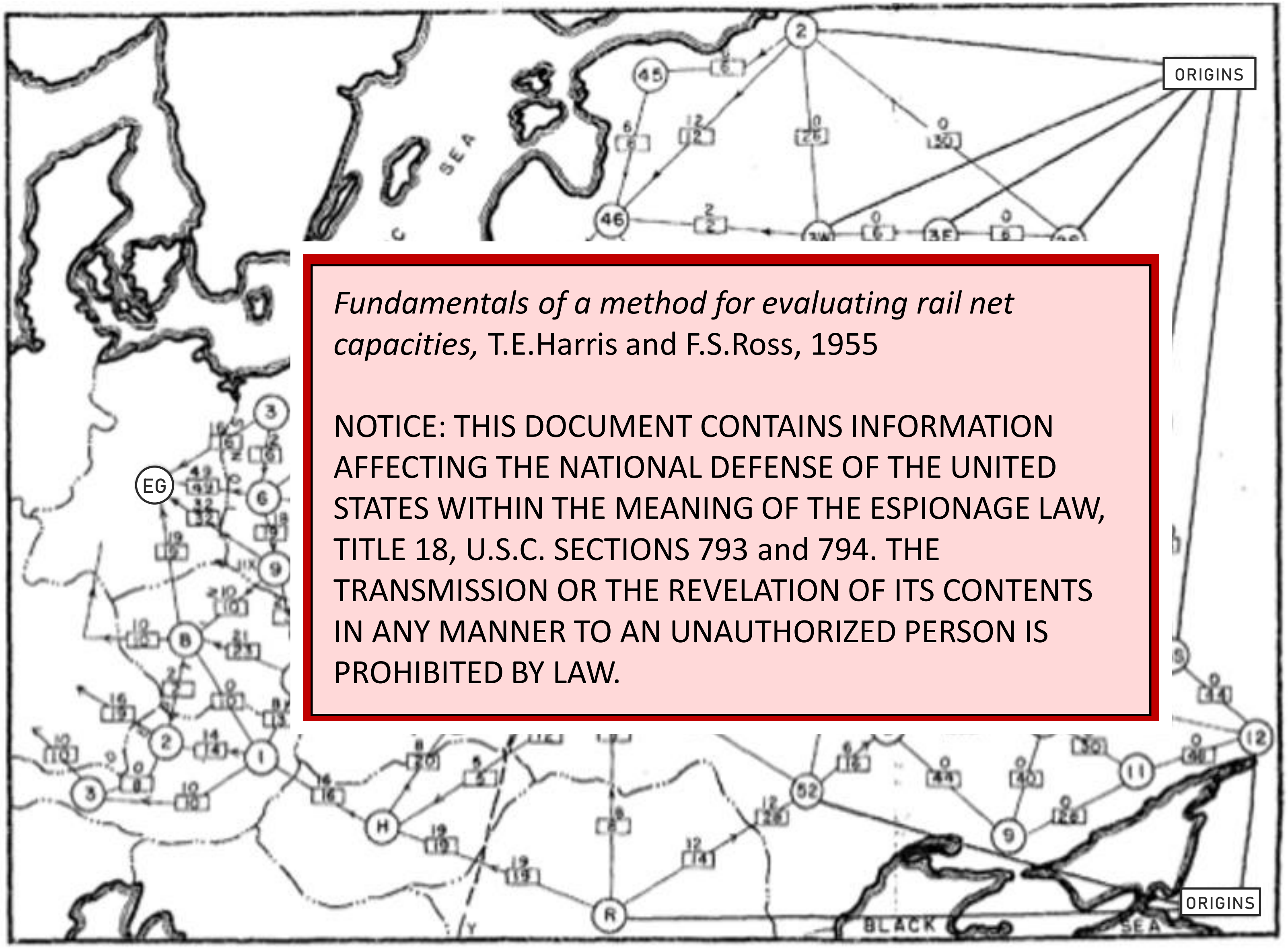
THE FLOW PROBLEM

Consider a directed graph in which each edge has a capacity.
How should we assign a flow to each edge,
so as to maximize the flow value?



Methods of finding the minimum total kilometrage in cargo-transportation planning in space, A.N.Tolstoy, 1930

SECRET RM-1573 10-24-55 -33-



Fundamentals of a method for evaluating rail net capacities, T.E.Harris and F.S.Ross, 1955

NOTICE: THIS DOCUMENT CONTAINS INFORMATION AFFECTING THE NATIONAL DEFENSE OF THE UNITED STATES WITHIN THE MEANING OF THE ESPIONAGE LAW, TITLE 18, U.S.C. SECTIONS 793 and 794. THE TRANSMISSION OR THE REVELATION OF ITS CONTENTS IN ANY MANNER TO AN UNAUTHORIZED PERSON IS PROHIBITED BY LAW.

Fig. 7 — Traffic pattern: entire network available

Legend:

—··— International boundary

(8) Railway operating division

9 / 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

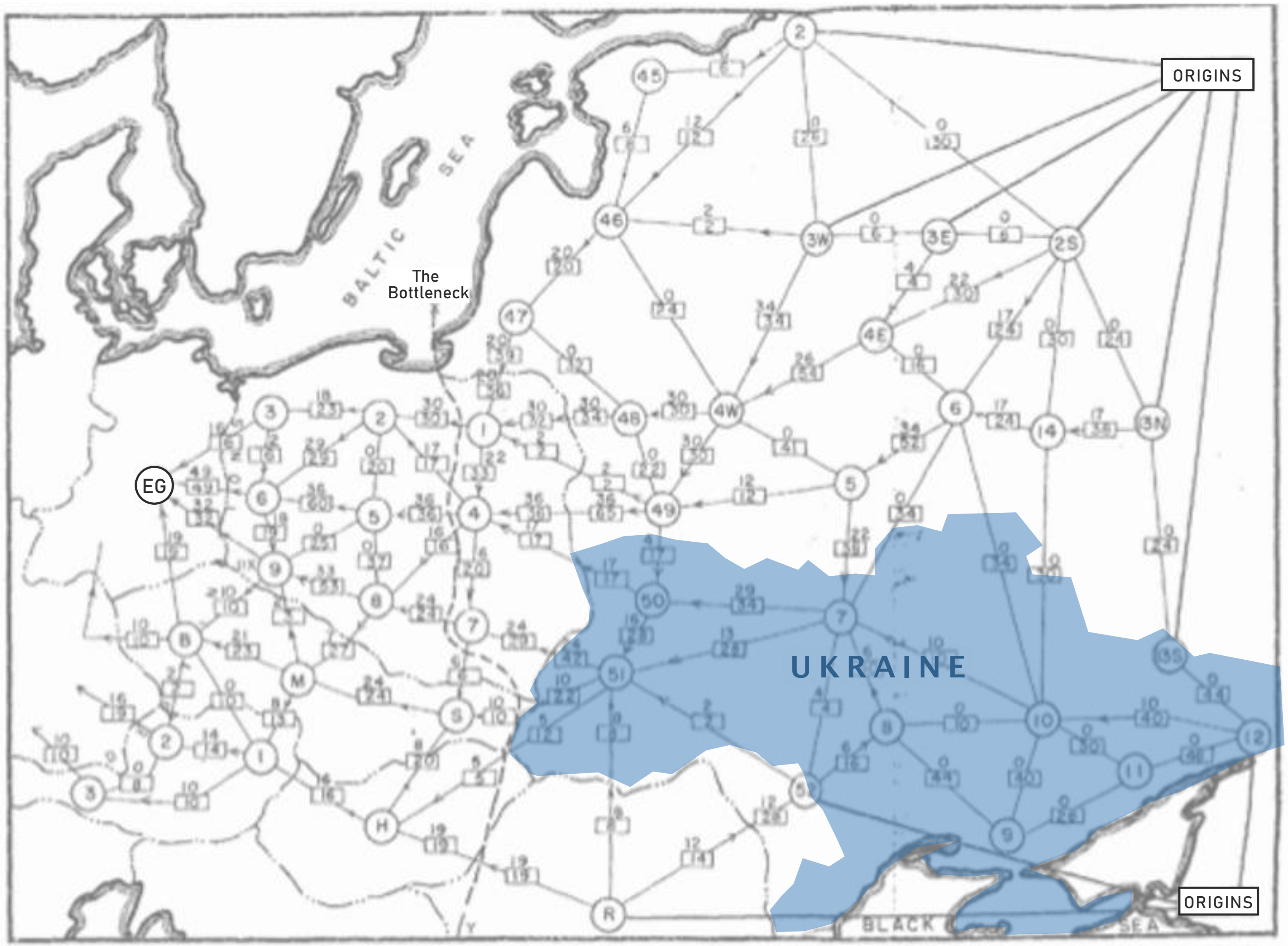
All capacities in trains / √1000's of tons each way per day

Origins: Divisions 2, 3W, 3E, 25, 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



SECRET RM-1573
10-24-55
-33-

Fig. 7 — Traffic pattern: entire network available

Legend:

International boundary

Railway operating division

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 trains / 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

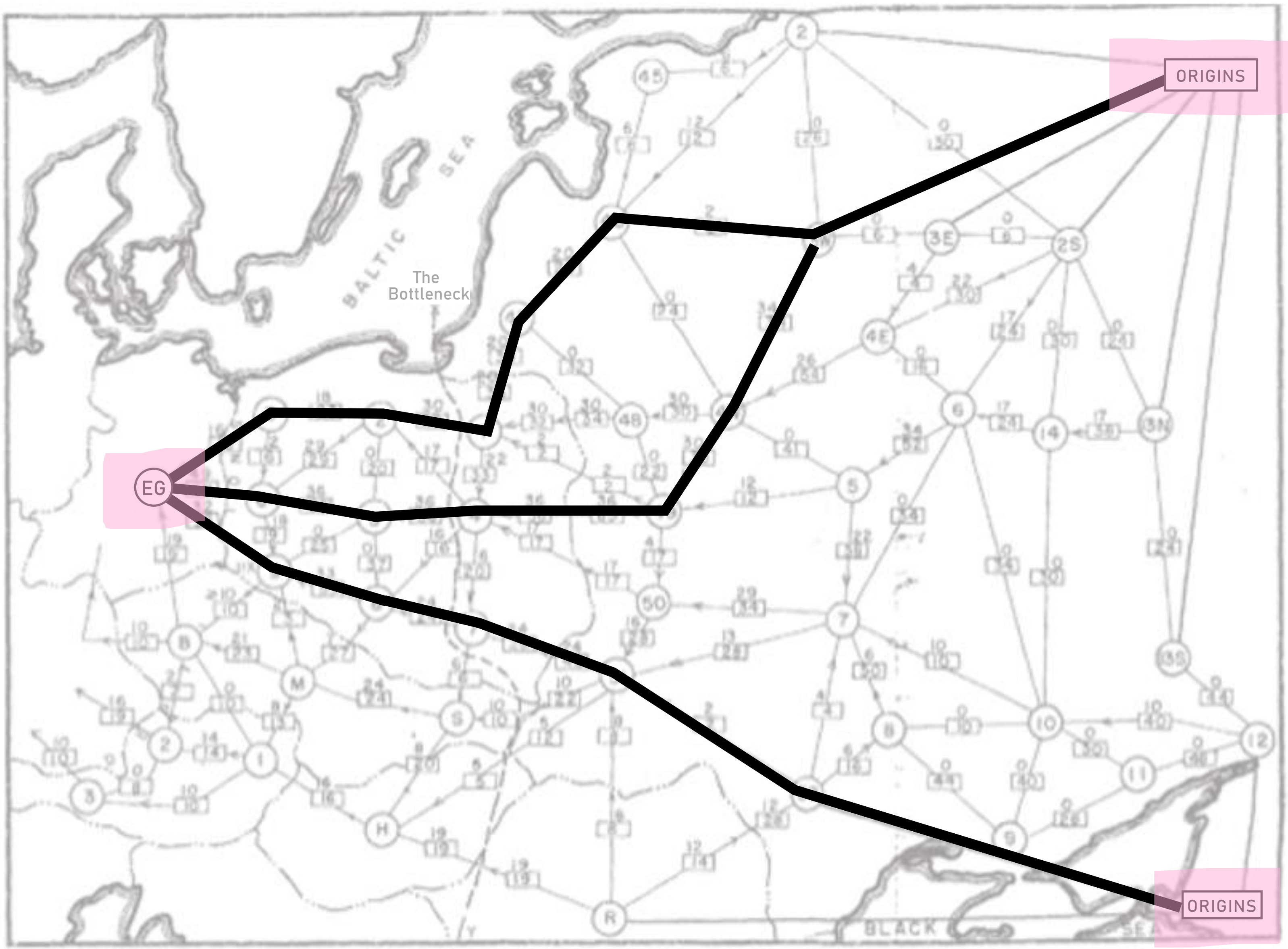


Fig. 7 — Traffic pattern: entire network available

Legend:

International boundary

Railway operating division

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

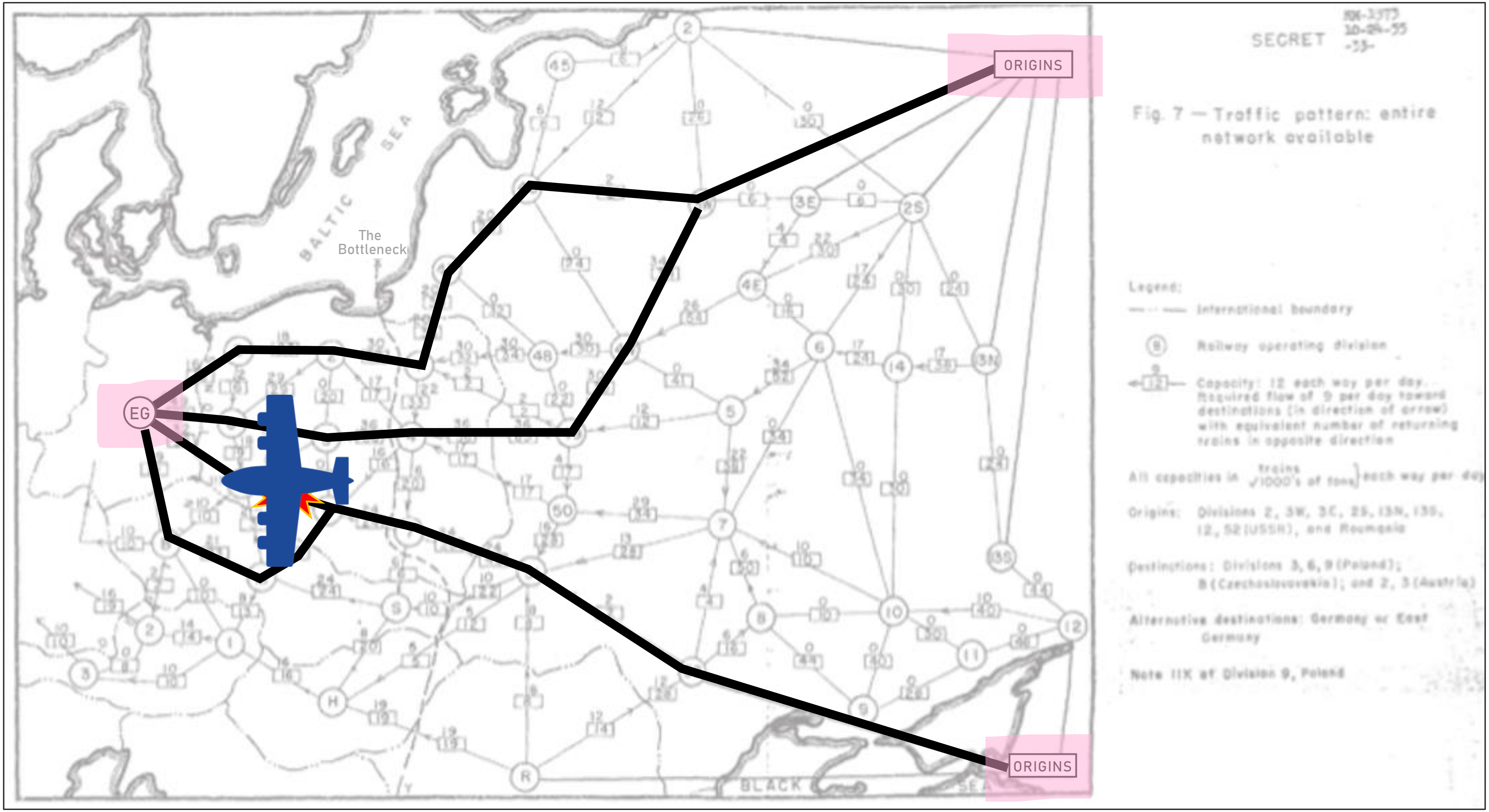
All capacities in trains / 1000's of tons each way per day

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

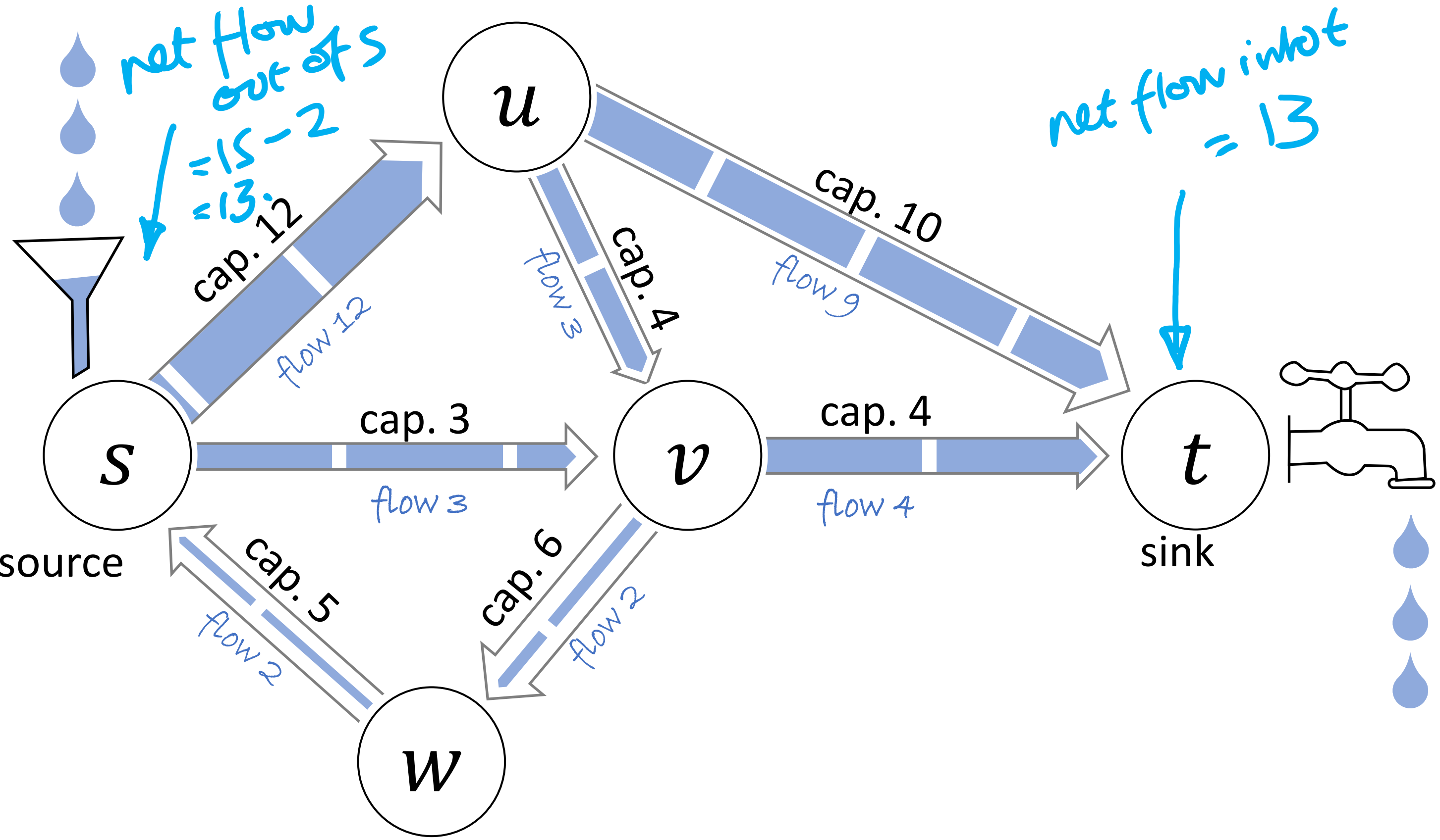
Destinations: Divisions 3, 6, 9 (Poland); B (Czechoslovakia); and 2, 3 (Austria)

Alternative destinations: Germany or East Germany

Note IIK of Division 9, Poland



ORIGINS
The Bottleneck
BALTIC SEA
EG
ORIGINS
BLACK SEA
SECRET
RM-1573
10-24-55
-53-
Fig. 7 — Traffic pattern: entire network available
Legend:
International boundary
Railway operating division
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 trains / 1000's of tons each way per day
Origins: Divisions 2, 3W, 3C, 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 11K of Division 9, Poland



Given a directed graph with a **source vertex** s and a **sink vertex** t, where each edge $u \to v$ has a **capacity** $c(u \to v) > 0$,

a **flow** f is a set of edge labels $f(u \to v)$ such that

- $0 \le f(u \to v) \le c(u \to v)$ on every edge
- total flow in = total flow out, at all vertices other than s and t

FLOW CONSERVATION

e.g. at vertex v,
flow in = 3+3
flow out = 4+2

and the **value of the flow** is

- value(f) = net flow out of s = net flow into t

PROBLEM STATEMENT

Find a flow with maximum possible value (called a *maximum flow*).

In symbolic notation,

FLOW CONSERVATION says that at all vertices other than s and t, tot flow in = tot flow out:

$$\forall v \in V \setminus \{s,t\} : \quad \sum_{w: v \to w} f(v \to w) = \sum_{w: w \to v} f(w \to v)$$

Equivalently,

$$\forall v \in V \setminus \{s,t\} : \quad \sum_{w: v \to w} f(v \to w) - \sum_{w: w \to v} f(w \to v) = 0$$

ie net flow in is zero.

FLOW VALUE

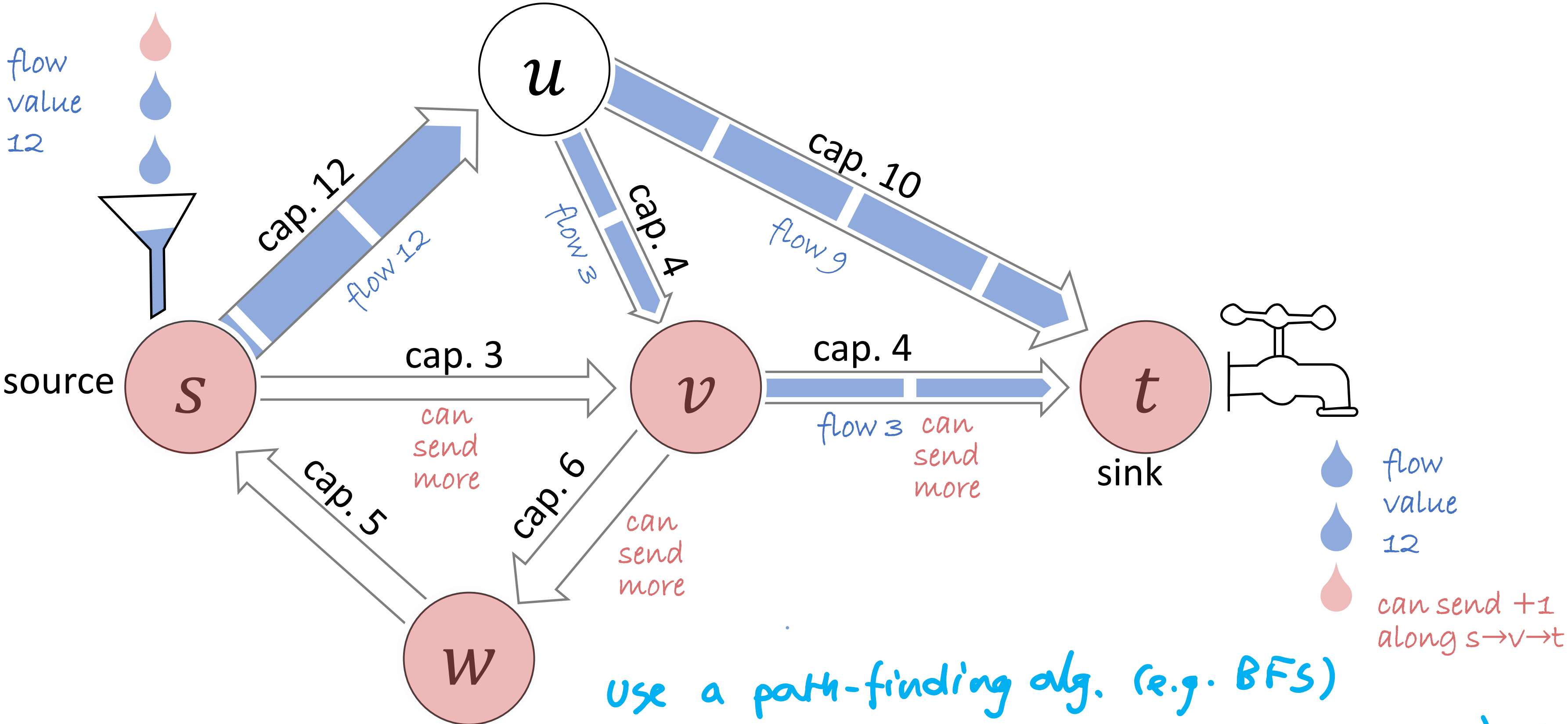
$$= \text{net flow out of } s = \sum_{v: s \to v} f(s \to v) - \sum_{v: v \to s} f(v \to s)$$

$$= \text{net flow into } t = \sum_{v: v \to t} f(v \to t) - \sum_{v: t \to v} f(t \to v)$$

(The example sheet asks you to prove that these two are always equal.
It uses a proof technique from next lecture.)

SECTION 6.2

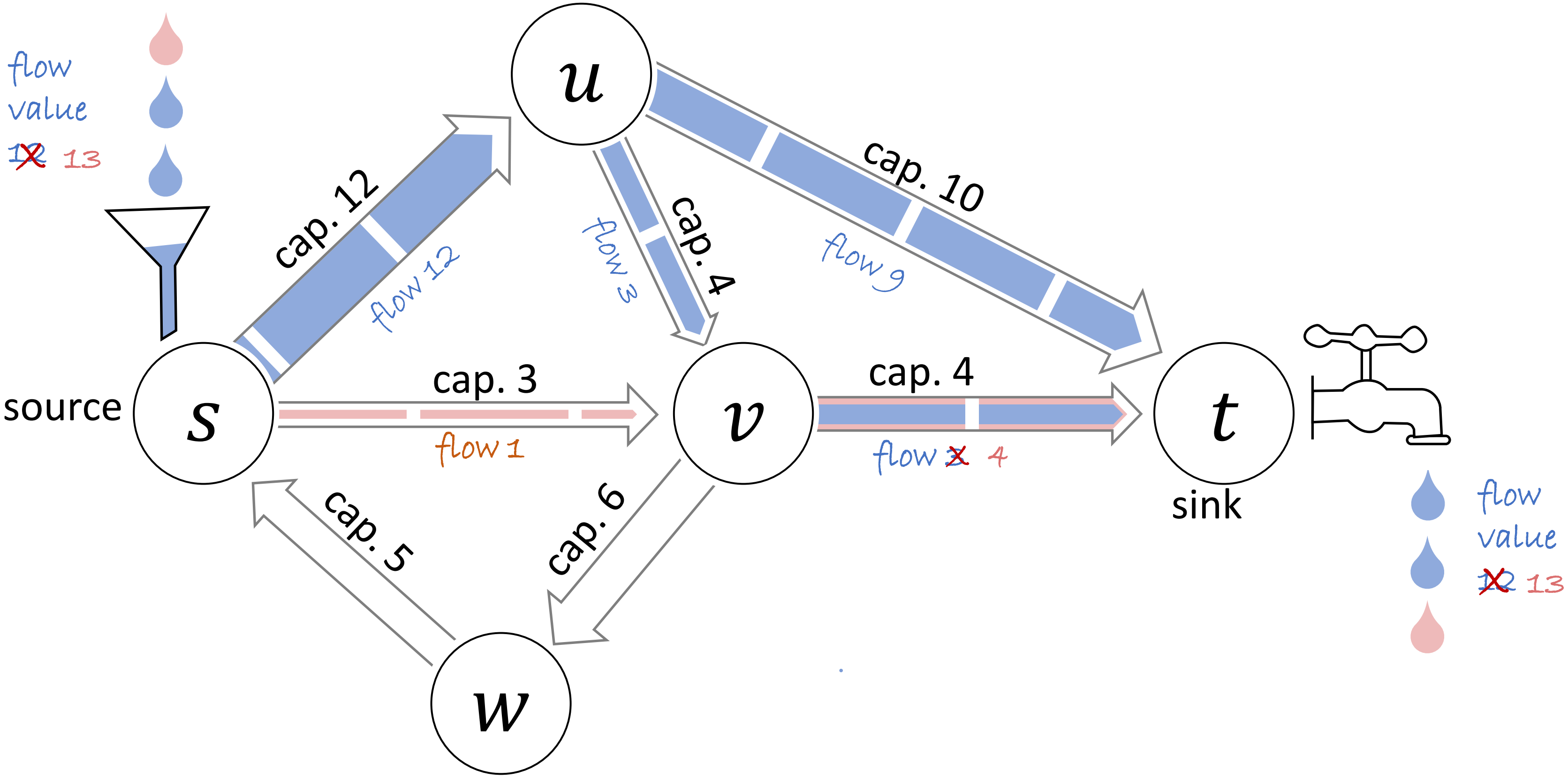
Ford-Fulkerson algorithm



Use a path-finding alg. (e.g. BFS) on the graph of "edges where we can increase capacity": Look for a path from s to t.

SIMPLE GREEDY STRATEGY

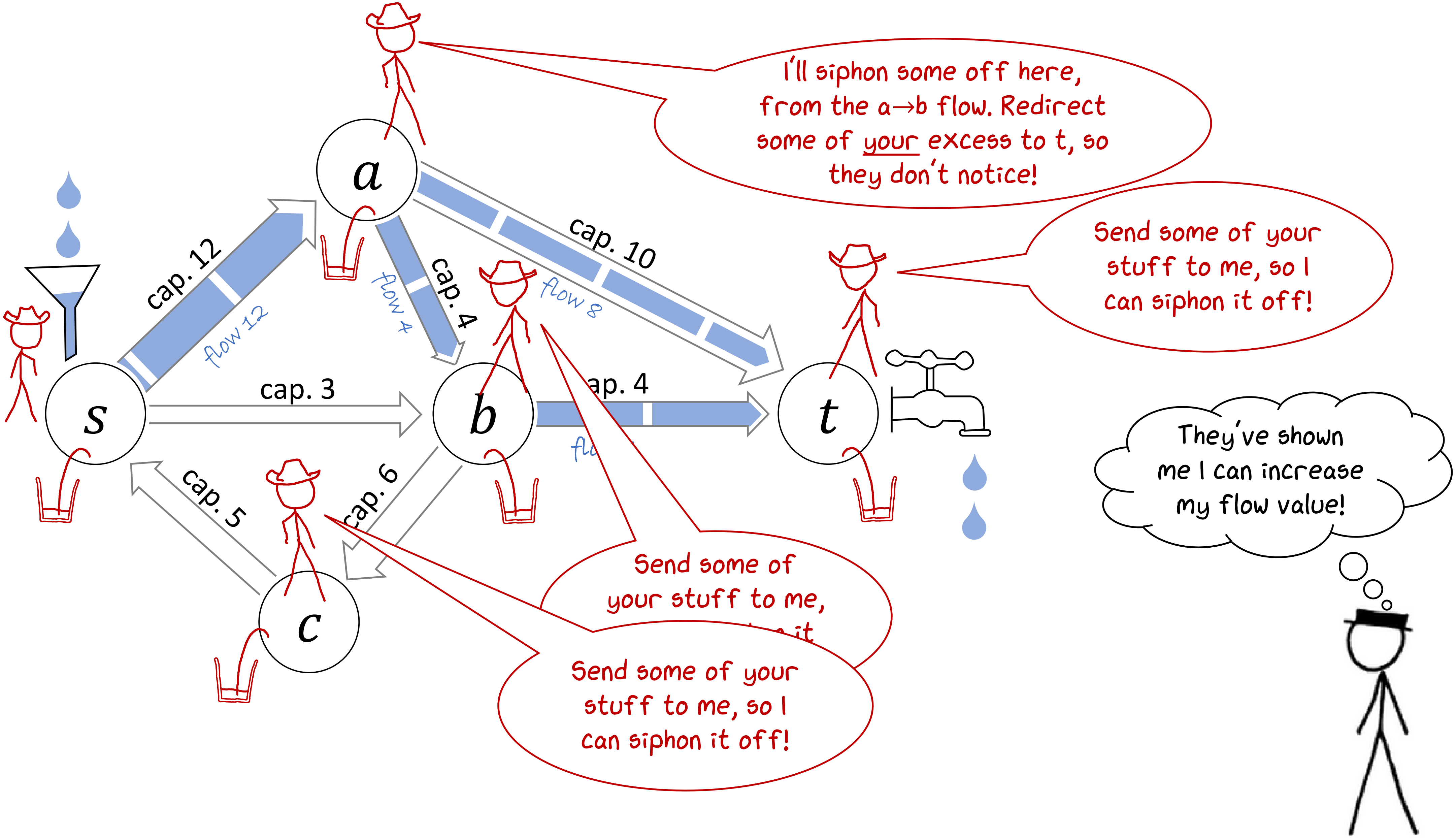
Look for a path to the sink along which we can increase flow, then increase it as much as we can. Repeat this, until we can't reach the sink.



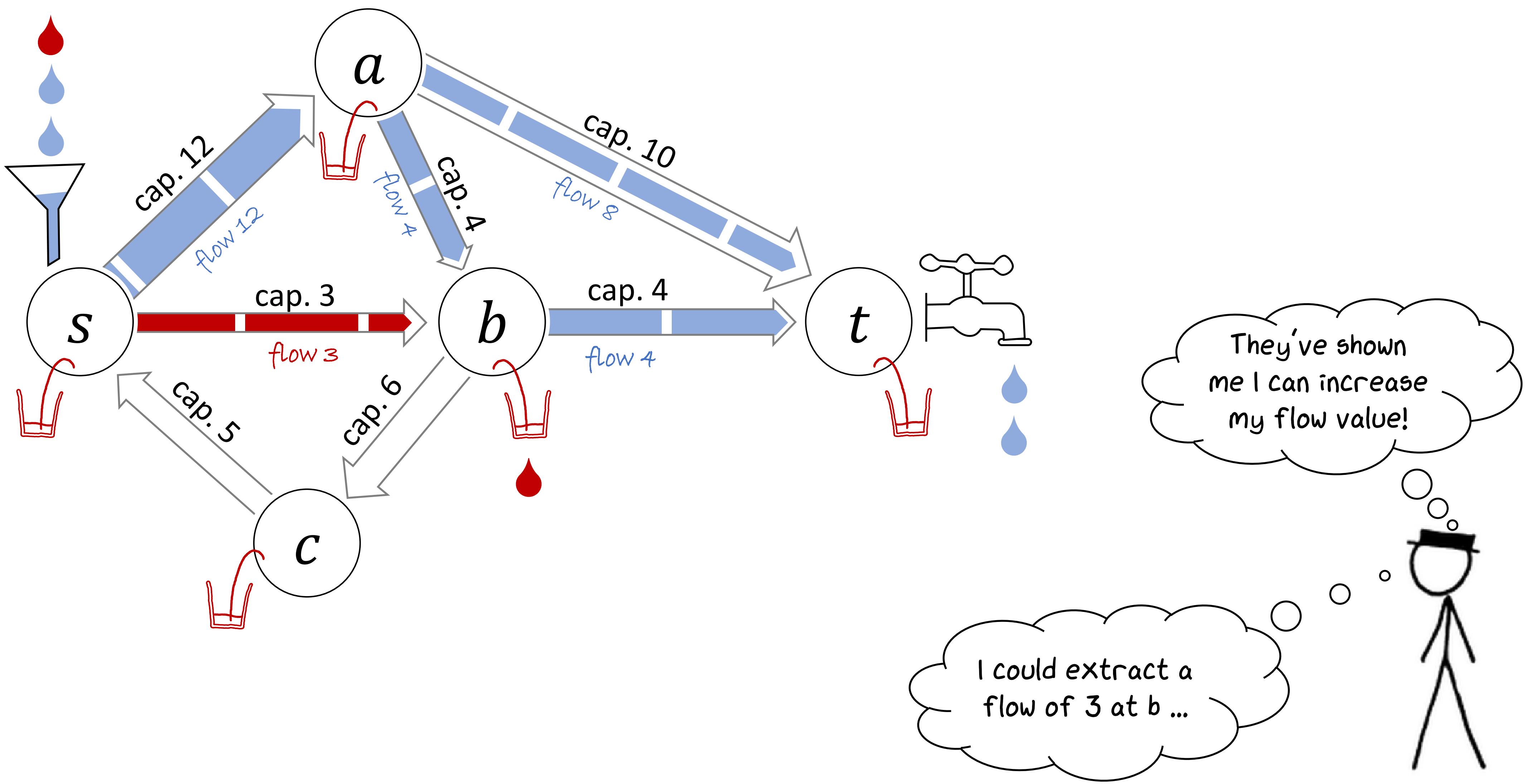
SIMPLE GREEDY STRATEGY

Look for a path to the sink along which we can increase flow, then increase it as much as we can. Repeat this, until we can't reach the sink.

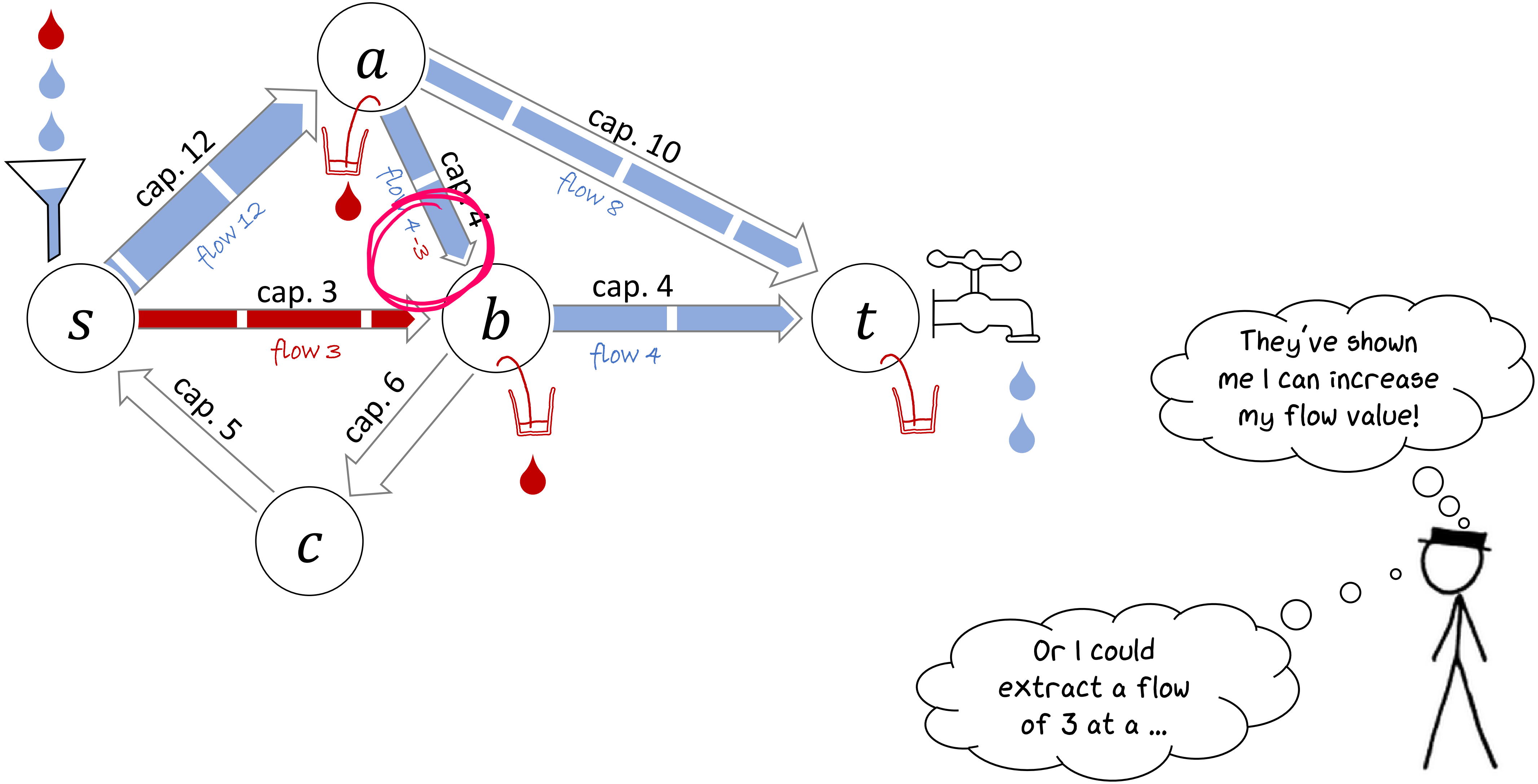
QUESTION. Can you find a larger-value flow than this?



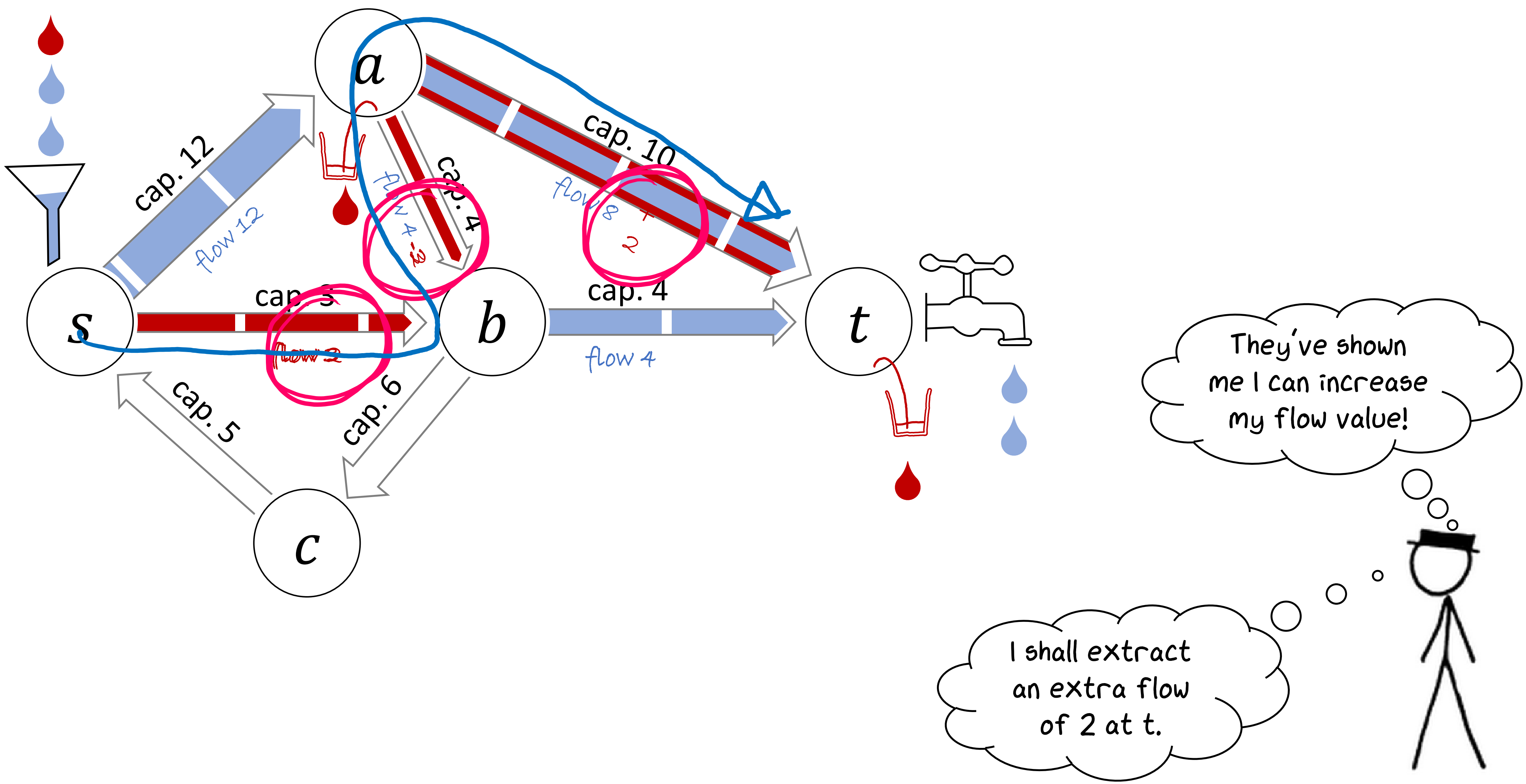
I'll siphon some off here, from the a→b flow. Redirect some of your excess to t, so they don't notice!
Send some of your stuff to me, so I can siphon it off!
Send some of your stuff to me,
Send some of your stuff to me, so I can siphon it off!
They've shown me I can increase my flow value!
a
b
c
s
t
cap. 12
flow 12
cap. 4
flow 4
cap. 10
flow 8
cap. 3
cap. 5
cap. 6



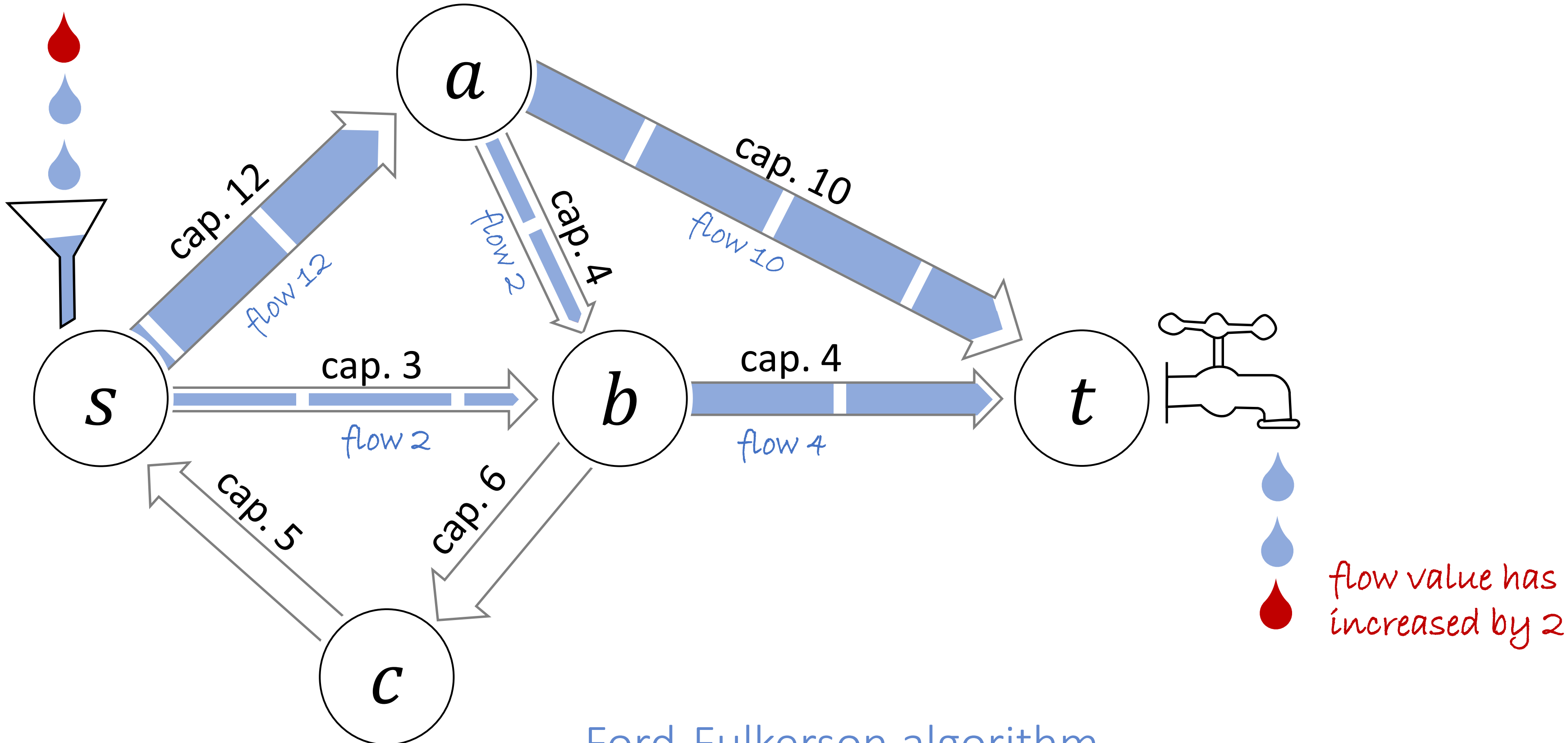
a
s
b
t
c
cap. 12
flow 12
cap. 4
flow 4
cap. 10
flow 8
cap. 3
flow 3
cap. 4
flow 4
cap. 6
cap. 5
They've shown me I can increase my flow value!
I could extract a flow of 3 at b ...



a
s
b
t
c
cap. 12
flow 12
cap. 4
flow 4-3
cap. 10
flow 8
cap. 3
flow 3
cap. 4
flow 4
cap. 5
cap. 6
They've shown me I can increase my flow value!
Or I could extract a flow of 3 at a ...



a
b
c
s
t
cap. 12
flow 12
cap. 10
flow 8 + 2
cap. 4
flow 4 -2
cap. 5
flow 2
cap. 4
flow 4
cap. 5
cap. 6
They've shown me I can increase my flow value!
I shall extract an extra flow of 2 at t.



Ford-Fulkerson algorithm

1. Start with zero flow

while True:

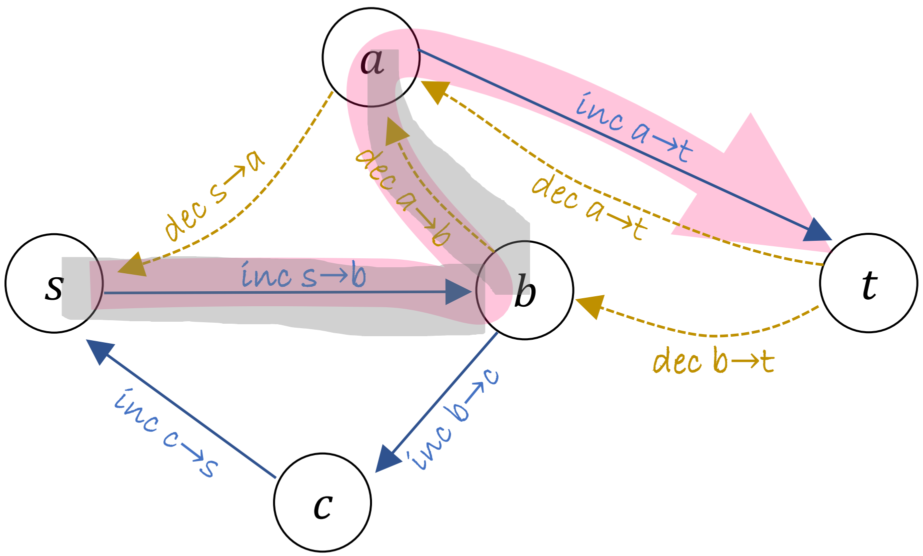
2. Run bandit search to discover if the flow to t can be increased, and, if so, find an appropriate sequence of edges

if t can be reached:

3. update the flow along those edges

if t can't be reached:

break



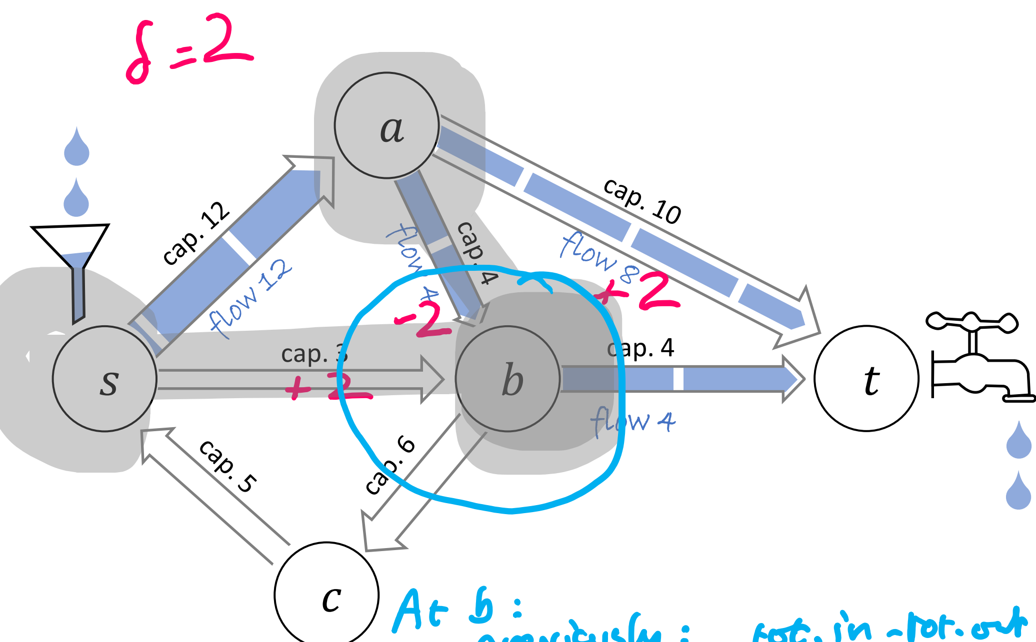
STEP 2A. Build the **residual graph,** which has the same vertices as the flow network, and

- if $f(u \to v) < c(u \to v)$:
  give the residual graph an edge $u \to v$
  with the label "increase flow $u \to v$"
- if $f(u \to v) > 0$:
  give the residual graph an edge $v \to u$
  with the label "decrease flow $u \to v$"

Not a typo!

dec. u→V
V
u

The resid. edge is in the opposite direction to the edge in the flow network.



STEP 2B. Look for a path from s to t in the residual graph. This is called an **augmenting path.**

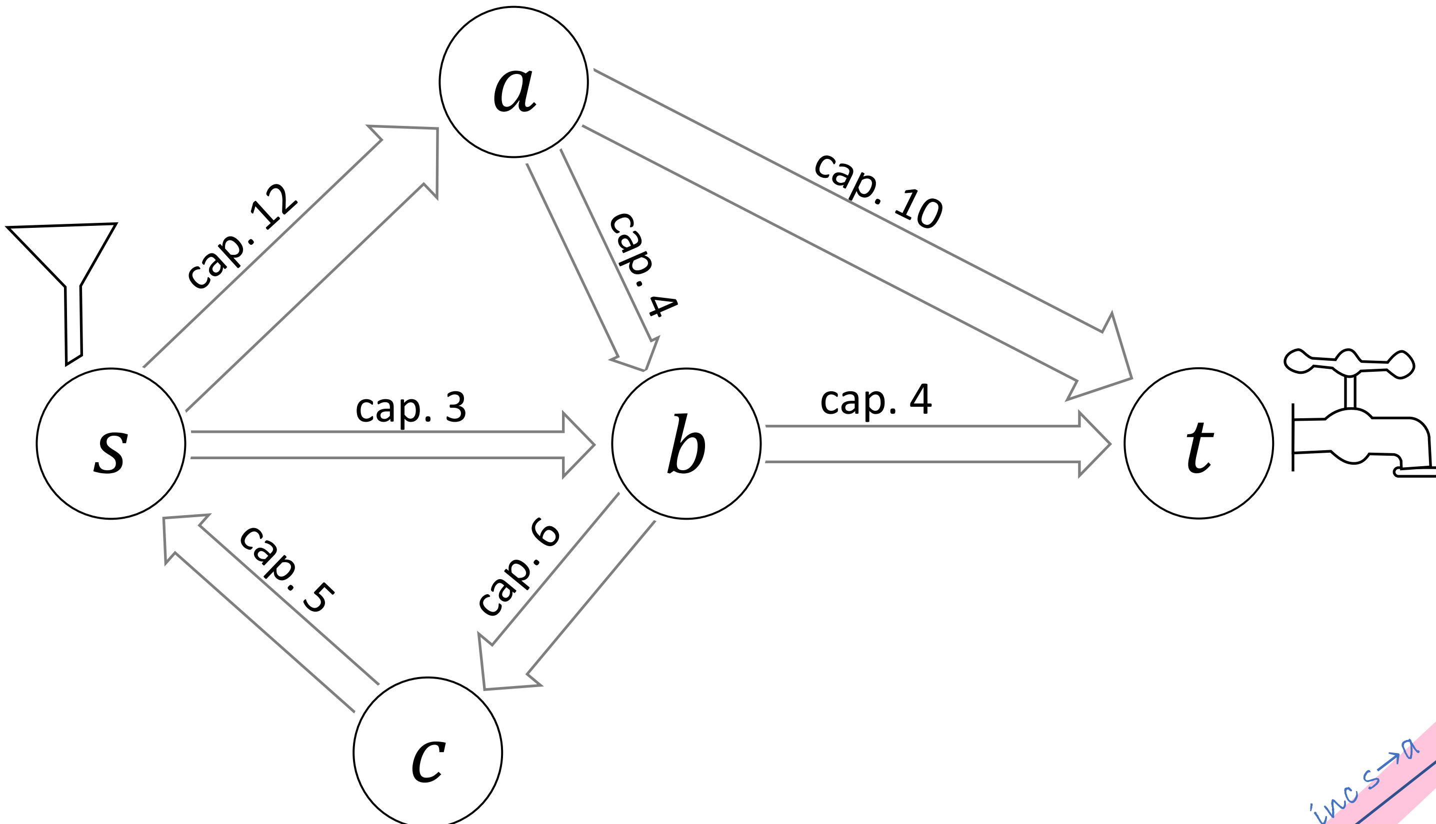
STEP 3. Find an update amount $\delta > 0$ that can be applied to all the edges along the augmenting path. Apply it.

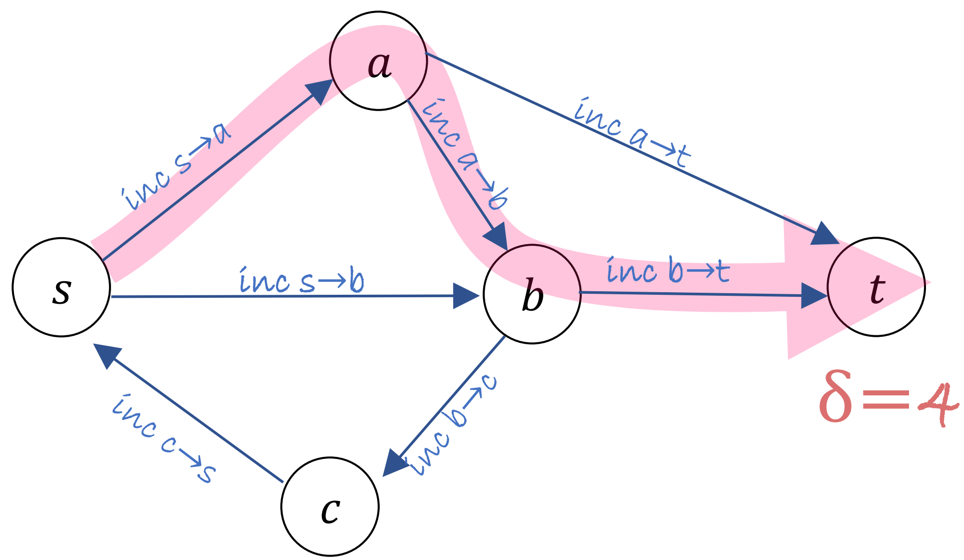
Lemma This yields a valid flow.

Proof
- We chose δ to ensure $0 \le f \le c$.
- flow conservation is still satisfied.

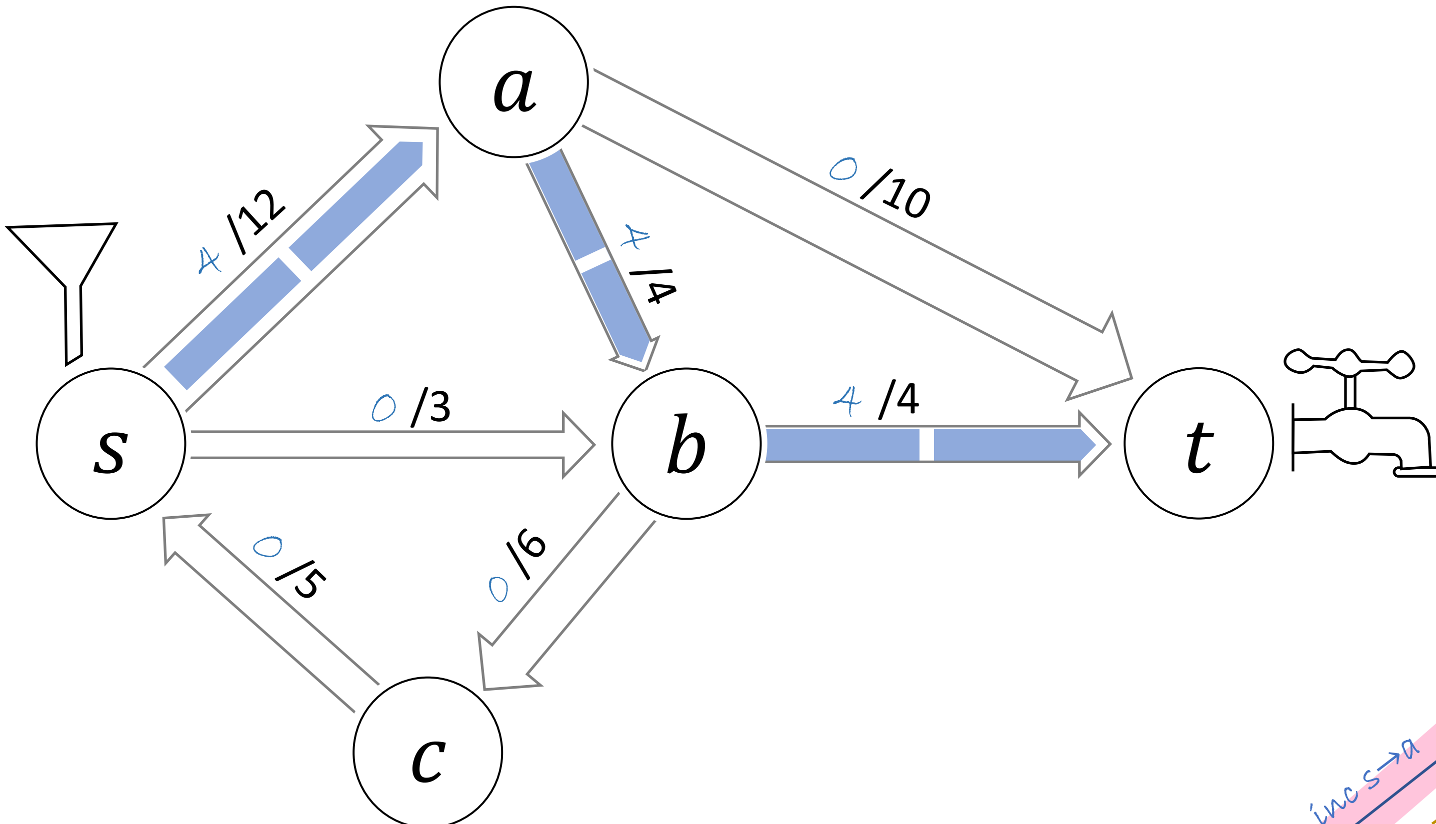
At b:
previously: tot. in - tot. out = 4 - 4 = 0
after: tot. in - tot. out = 4 - 4 = 0.

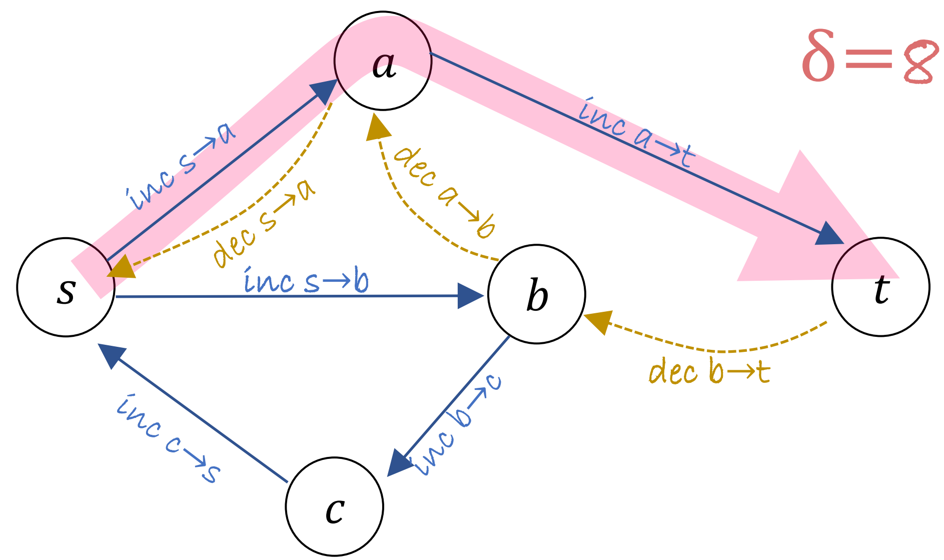
WALKTHROUGH OF FORD-FULKERSON





WALKTHROUGH OF FORD-FULKERSON

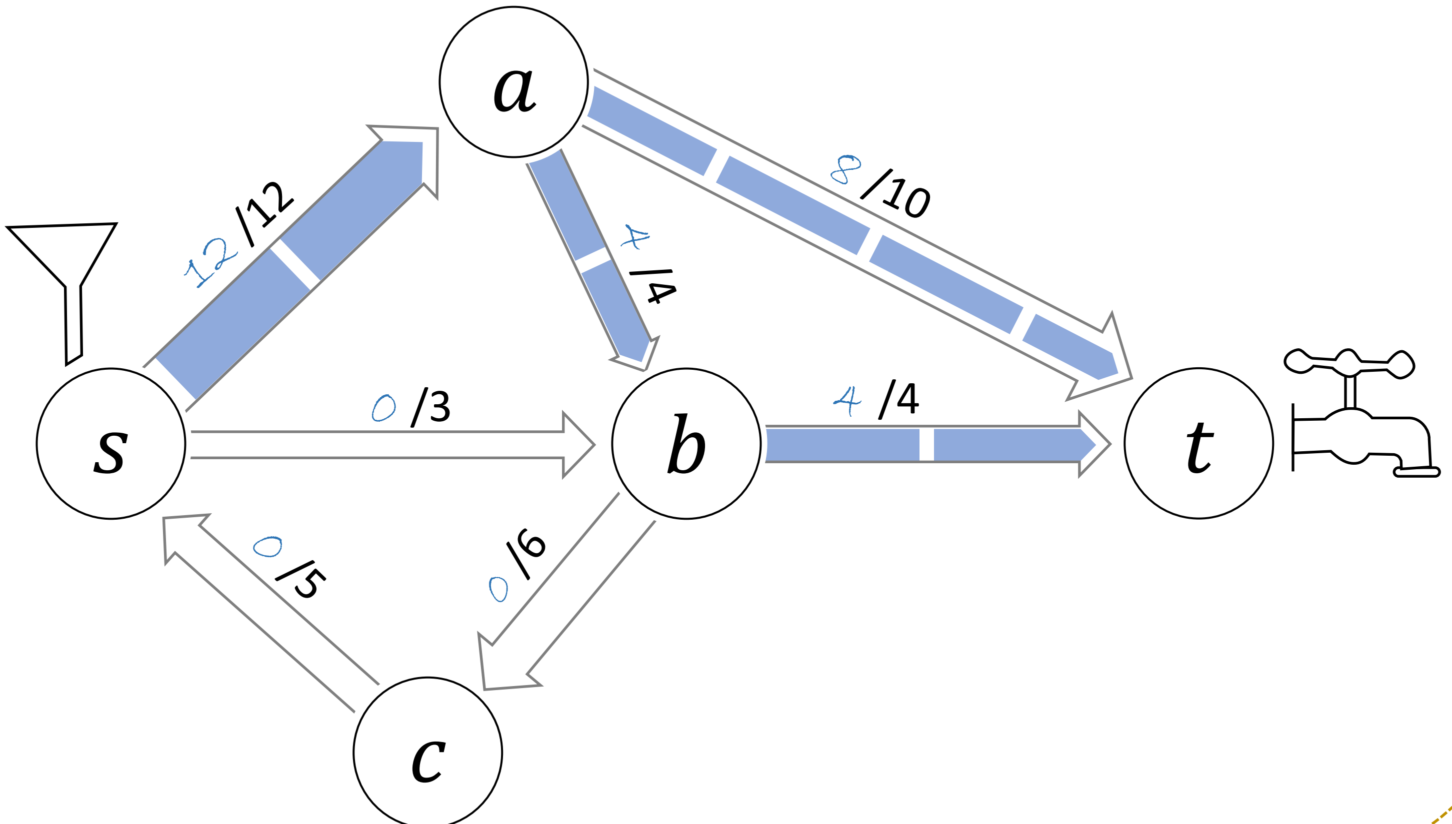


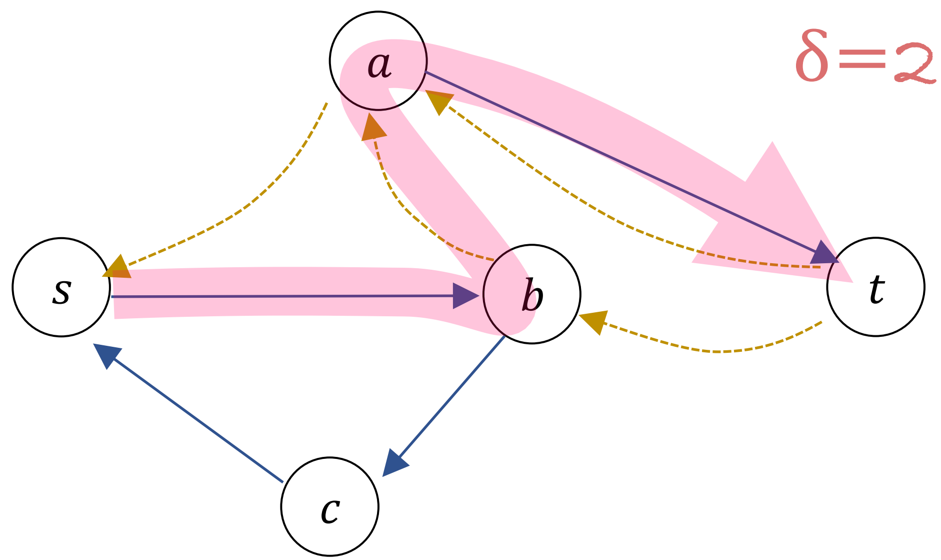


CRUCIAL TRICK

The residual graph doesn't have capacities, it just has edges, so we can use e.g. breadth-first search to find a path. We've reduced flow-finding to path-finding.

WALKTHROUGH OF FORD-FULKERSON



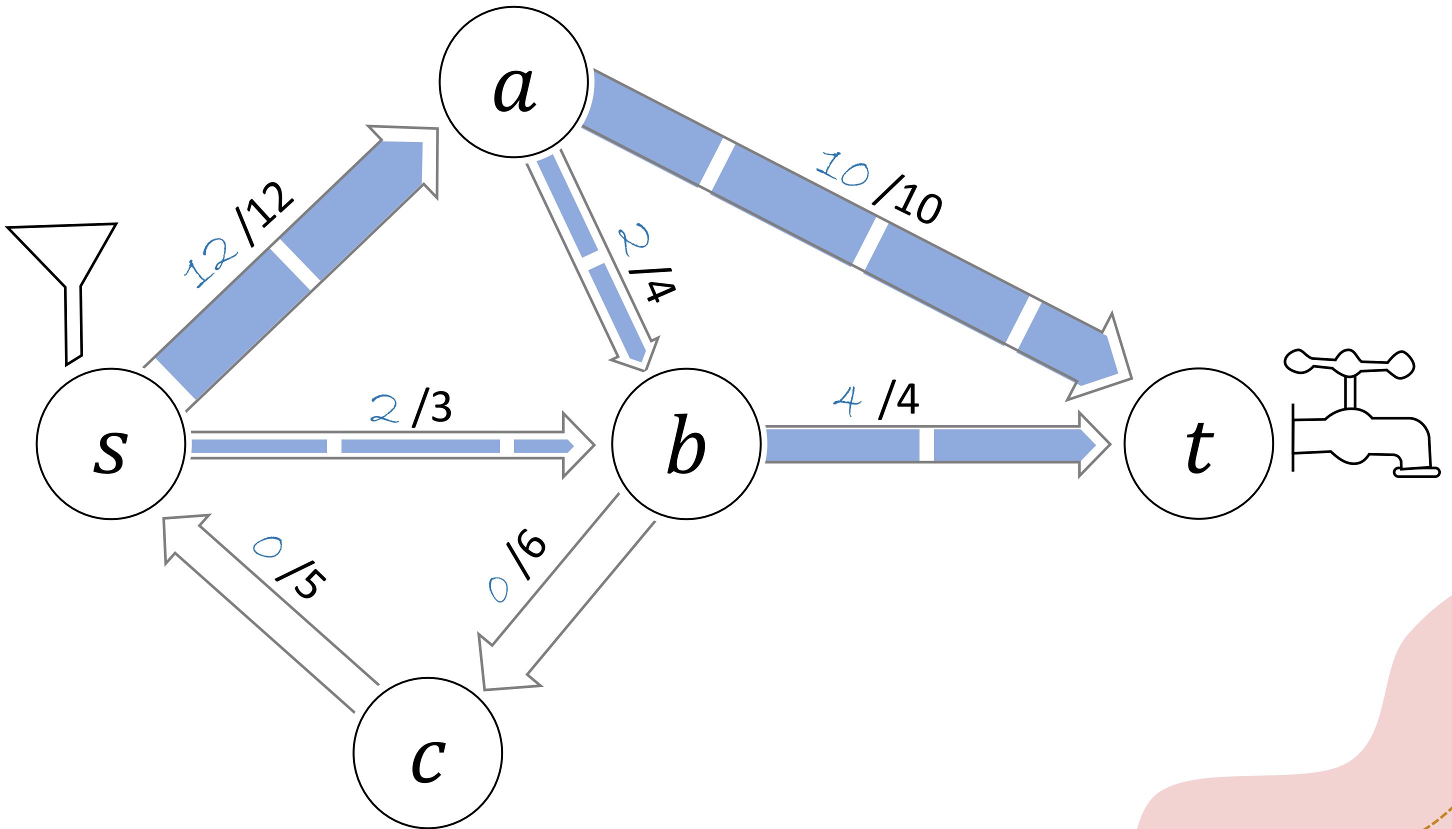


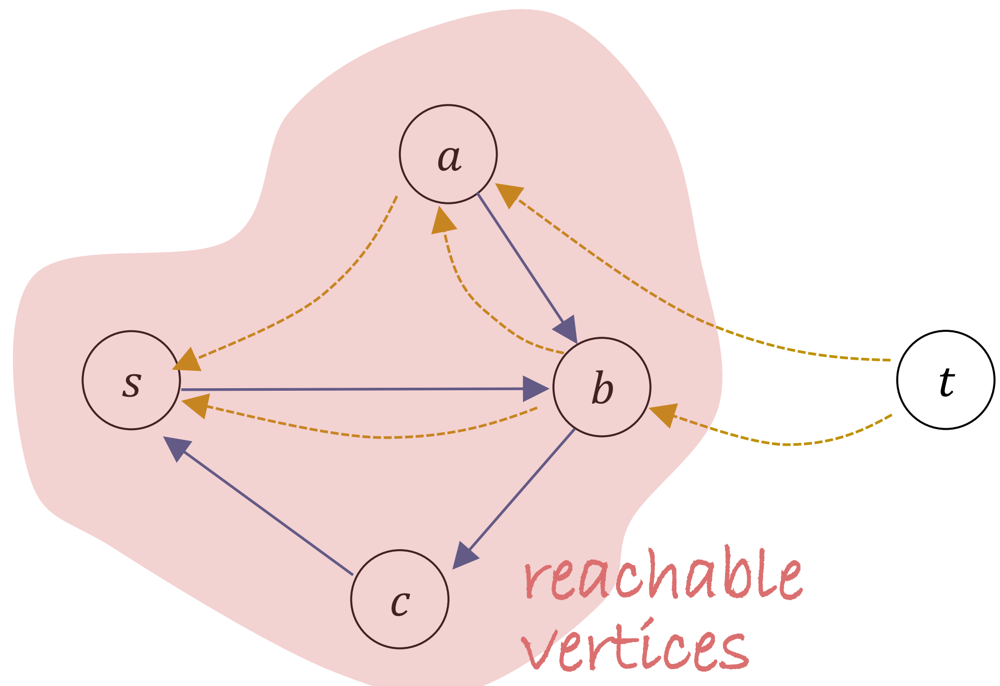
CRUCIAL TRICK

The residual graph doesn't have capacities, it just has edges, so we can use e.g. breadth-first search to find a path. We've reduced flow-finding to path-finding.

WALKTHROUGH OF FORD-FULKERSON

We cannot find an augmenting path in the residual graph. So, terminate.





CRUCIAL TRICK

The residual graph doesn't have capacities, it just has edges, so we can use e.g. breadth-first search to find a path. We've reduced flow-finding to path-finding.

Assume capacities are all integer.

```
def ford_fulkerson(g, s, t):
    # Let f be a flow, initially empty
    for u → v in g.edges:
        f(u → v) = 0

    # Define a helper function for finding an augmenting path
    def find_augmenting_path():
        # Define the residual graph h on the same vertices as g
        for u → v in g.edges:
            if f(u → v) < c(u → v): give h an edge u → v labelled "inc u → v"
            if f(u → v) > 0: give h an edge v → u labelled "dec u → v"
        if h has a path from s to t:
            return some such path, together with the labels of its edges
        else:
            # Let S be the set of vertices the bandits can reach (used in the proof)
            return None

    # Repeatedly find an augmenting path and add flow to it
    while True:
        p = find_augmenting_path()
        if p is None:
            break
        else:
            compute δ, the amount of flow to apply along p, and apply it
            # Assert: δ > 0
            # Assert: f is still a valid flow
```

f is integer

e.g. using BFS $O(V+E)$

δ will be integer.

by construction of residual graph.

$O(V)$

total cost = # iterations × $O(V+E)$.

δ is integer. $\delta > 0$ ∴ $\delta \geq 1$. So flow value changes by at least 1, and remains integer.

⇒ # iterations ≤ max flow value < ∞

The Integrality Lemma. If the capacities are all integers, then the algorithm terminates, and the resulting flow on each edge is an integer.

Algorithms tick max-flow

cl.cam.ac.uk/teaching/2324/Alg...

Deadline 11 March.

Algorithms tick: max-flow

Maximum flow with Ford-Fulkerson / Edmonds-Karp

In this tick you will build a Ford–Fulkerson implementation from scratch. In fact you will implement the Edmonds–Karp variant of Ford–Fulkerson, which uses breadth first search (BFS) to find augmenting paths, and which has $O(VE^2)$ running time.

Computer Science > Data Structures and Algorithms

[Submitted on 1 Mar 2022 (v1), last revised 22 Apr 2022 (this version, v2)]

Maximum Flow and Minimum-Cost Flow in Almost-Linear Time

Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, Sushant Sachdeva

We give an algorithm that computes exact maximum flows and minimum-cost flows on directed graphs with m edges and polynomially bounded integral demands, costs, and capacities in $m^{1+o(1)}$ time. Our algorithm builds the flow through a sequence of $m^{1+o(1)}$ approximate undirected minimum-ratio cycles, each of which is computed and processed in amortized $m^{o(1)}$ time using a new dynamic graph data structure.

Our framework extends to algorithms running in $m^{1+o(1)}$ time for computing flows that minimize general edge-separable convex functions to high accuracy. This gives almost-linear time algorithms for several problems including entropy-regularized optimal transport, matrix scaling, p-norm flows, and p-norm isotonic regression on arbitrary directed acyclic graphs.

m edges

$\forall \varepsilon > 0\ \exists\ \kappa, m_0.$

$\forall m \geq m_0,$

runtime $\leq \kappa m^{1+\varepsilon}$

Subjects: **Data Structures and Algorithms (cs.DS)**

Cite as: arXiv:2203.00671 **[cs.DS]**

(or arXiv:2203.00671v2 **[cs.DS]** for this version)

https://doi.org/10.48550/arXiv.2203.00671

Submission history

From: Li Chen [view email]

[v1] Tue, 1 Mar 2022 18:45:57 UTC (133 KB)

Download:

- PDF
- Other formats

(license)

Current browse context:
cs.DS

< prev | next >

new | recent | 2203

Change to browse by:
cs

References & Citations

- NASA ADS
- Google Scholar
- Semantic Scholar

5 blog links (what is this?)

Export Bibtex Citation

Bookmark