CLRS3 lemma 24.15 (used in Bellman-Ford). Consider a weighted directed graph. Consider any shortest path from s to t,

$$s = v_0 \to v_1 \to \cdots \to v_k = t.$$

Suppose we initialize the data structure by

$v.\mathtt{dist} = \infty$ for all vertices other than s

$s.\mathtt{dist} = 0$

and then we perform a sequence of relaxation steps that includes, in order, relaxing $v_0 \to v_1$, then $v_1 \to v_2$, then … then $v_{k-1} \to v_k$. After these relaxations, and at all times thereafter, $v_k.\mathtt{dist} = \mathrm{distance}(s \text{ to } v_k)$.

We'll prove by induction that, after the ith edge has been relaxed,

$$v_i.\mathtt{dist} = \mathrm{distance}(s \text{ to } v_i)$$

BASE CASE $i = 0$: Note that $s = v_0$. We initialized $s.\mathtt{dist} = 0$, and $\mathrm{distance}(s \text{ to } s) = 0$, so the induction hypothesis is true.

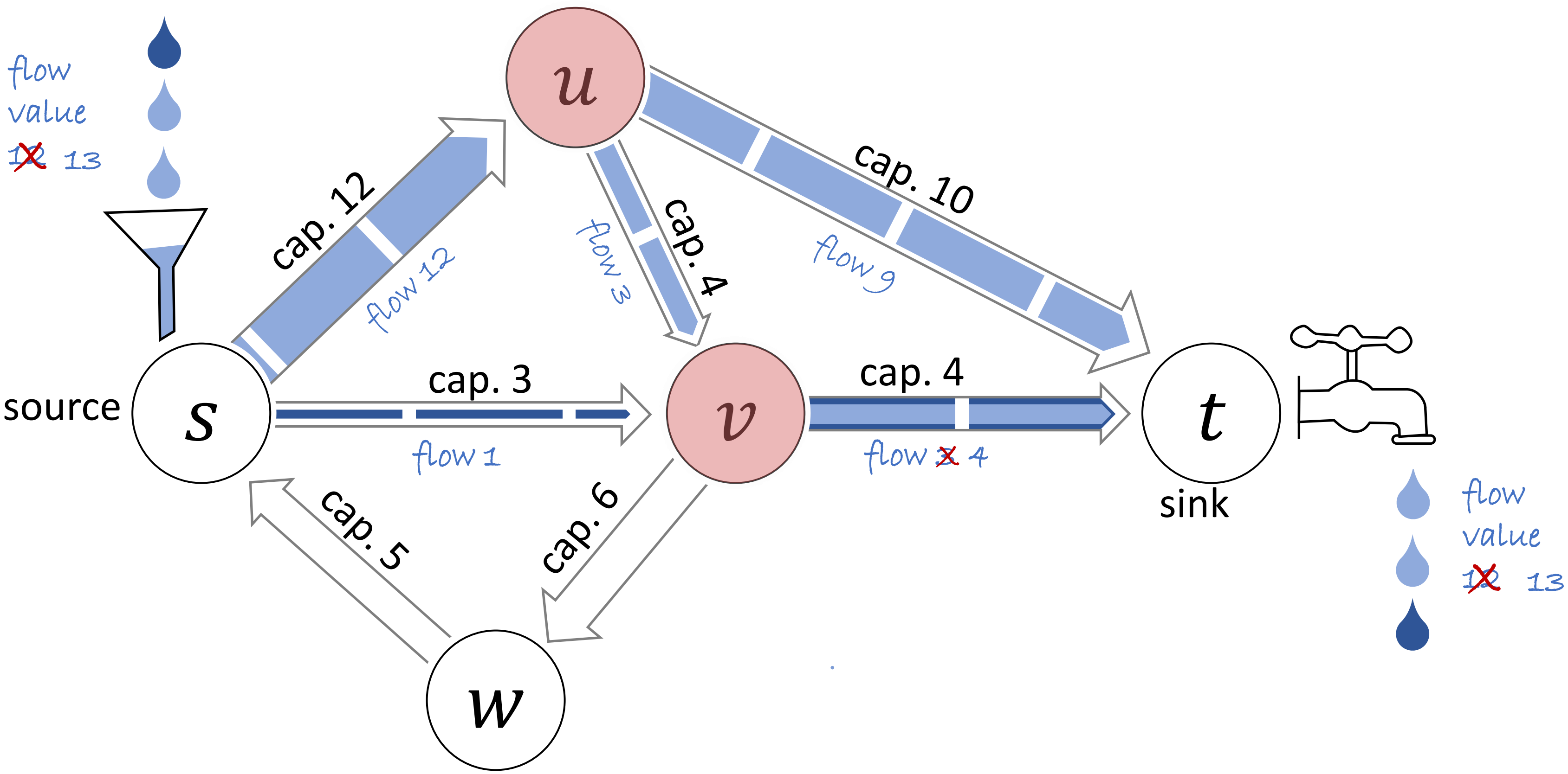
If there's a graph with -ve weight cycle, it's possible that distance (s to s) = -∞!

INDUCTION STEP: …

So, is this proof right, wrong, or not even wrong?

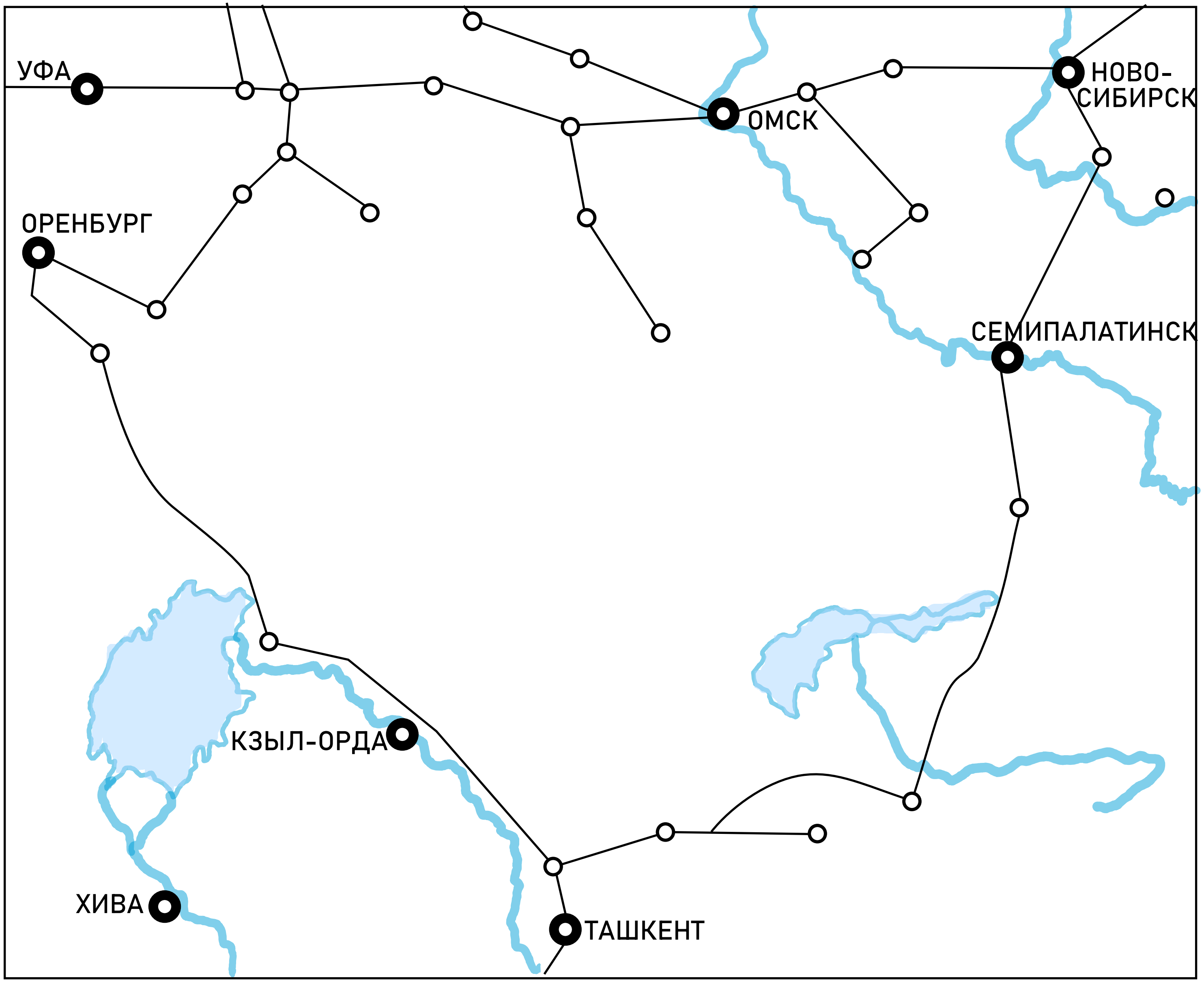
SECTION 6.1

Flow networks

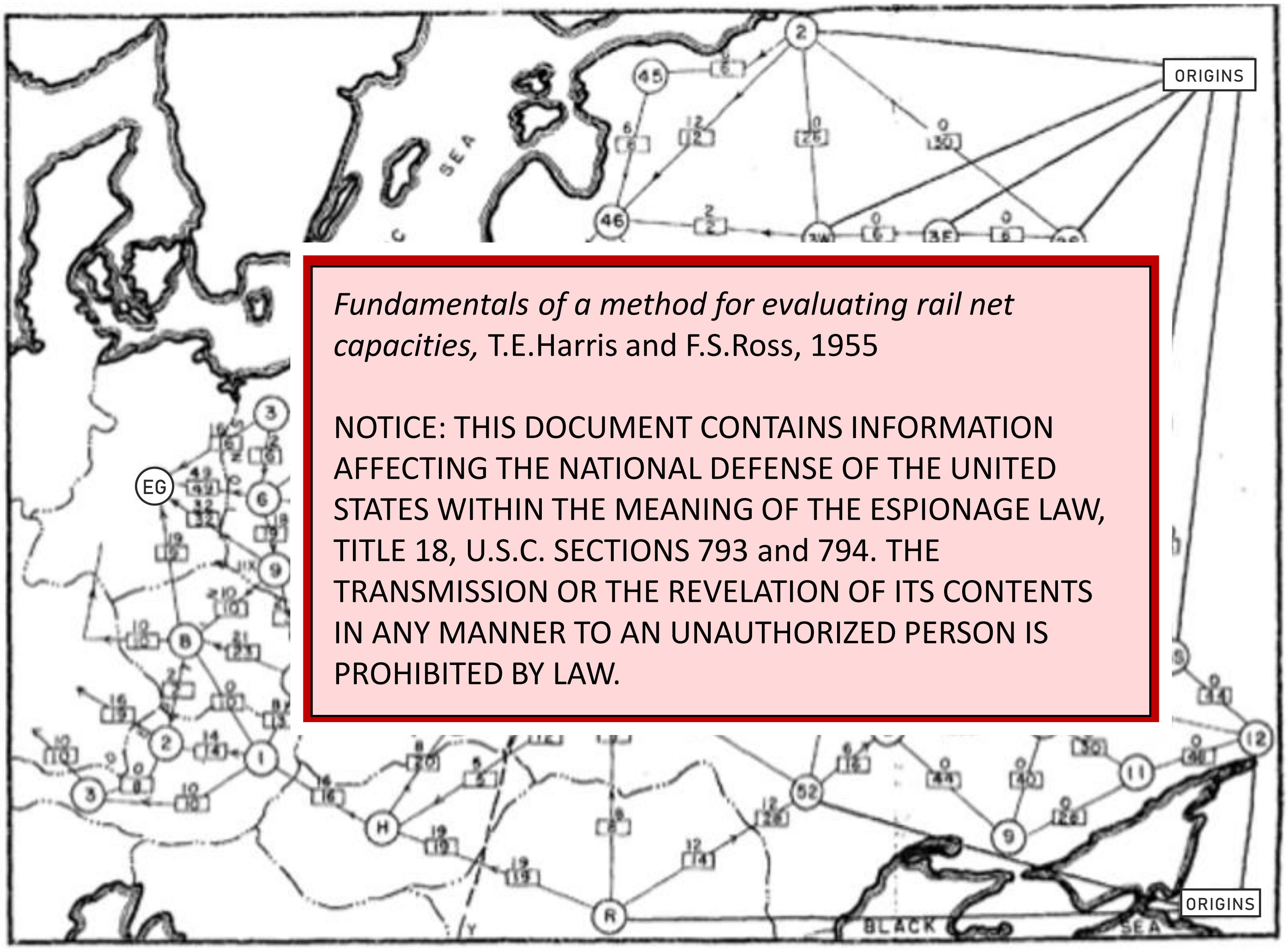


THE FLOW PROBLEM

Consider a 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



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.

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

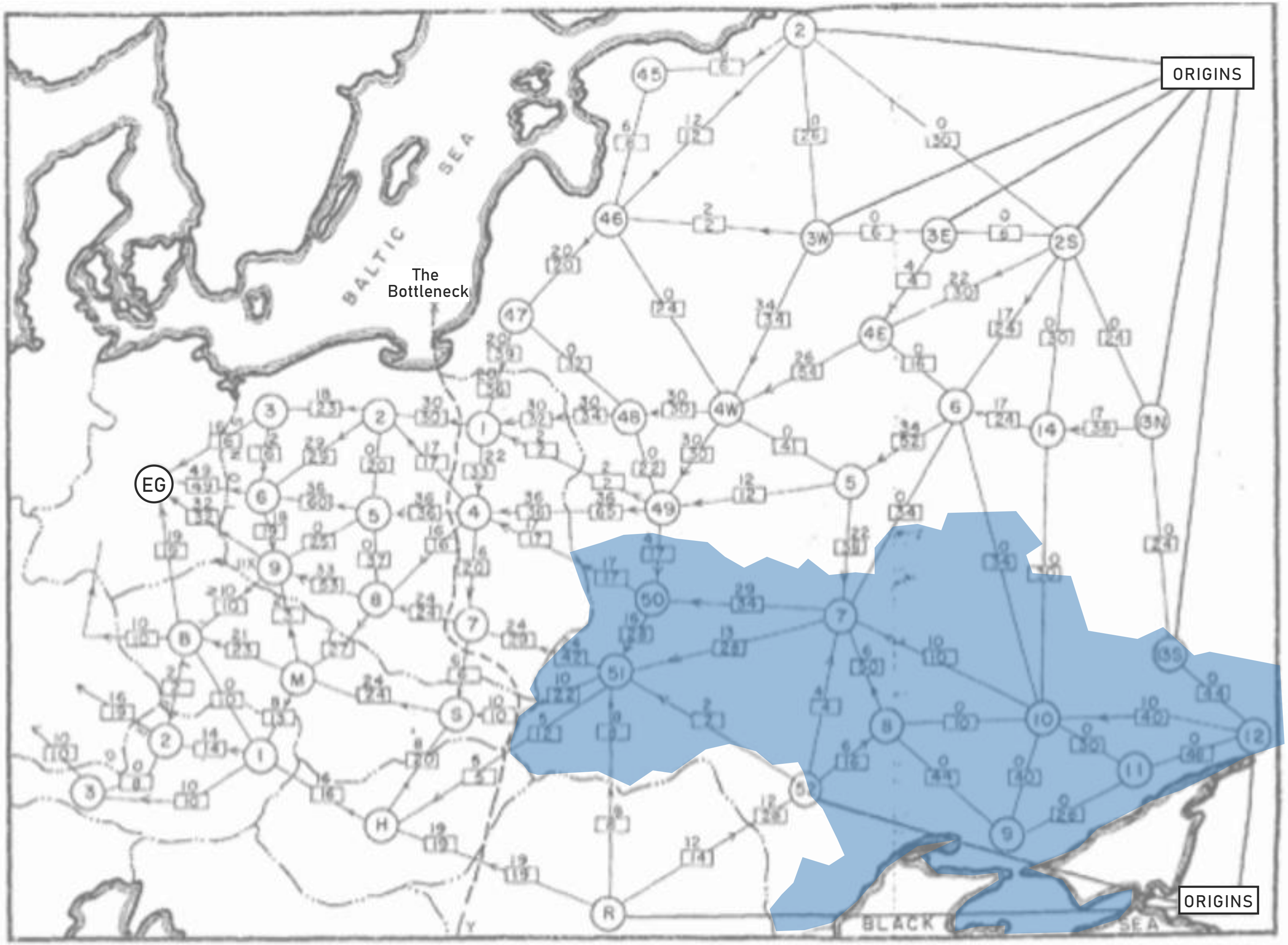
All capacities in trains / $\sqrt{}$1000's of tons each way per day

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

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

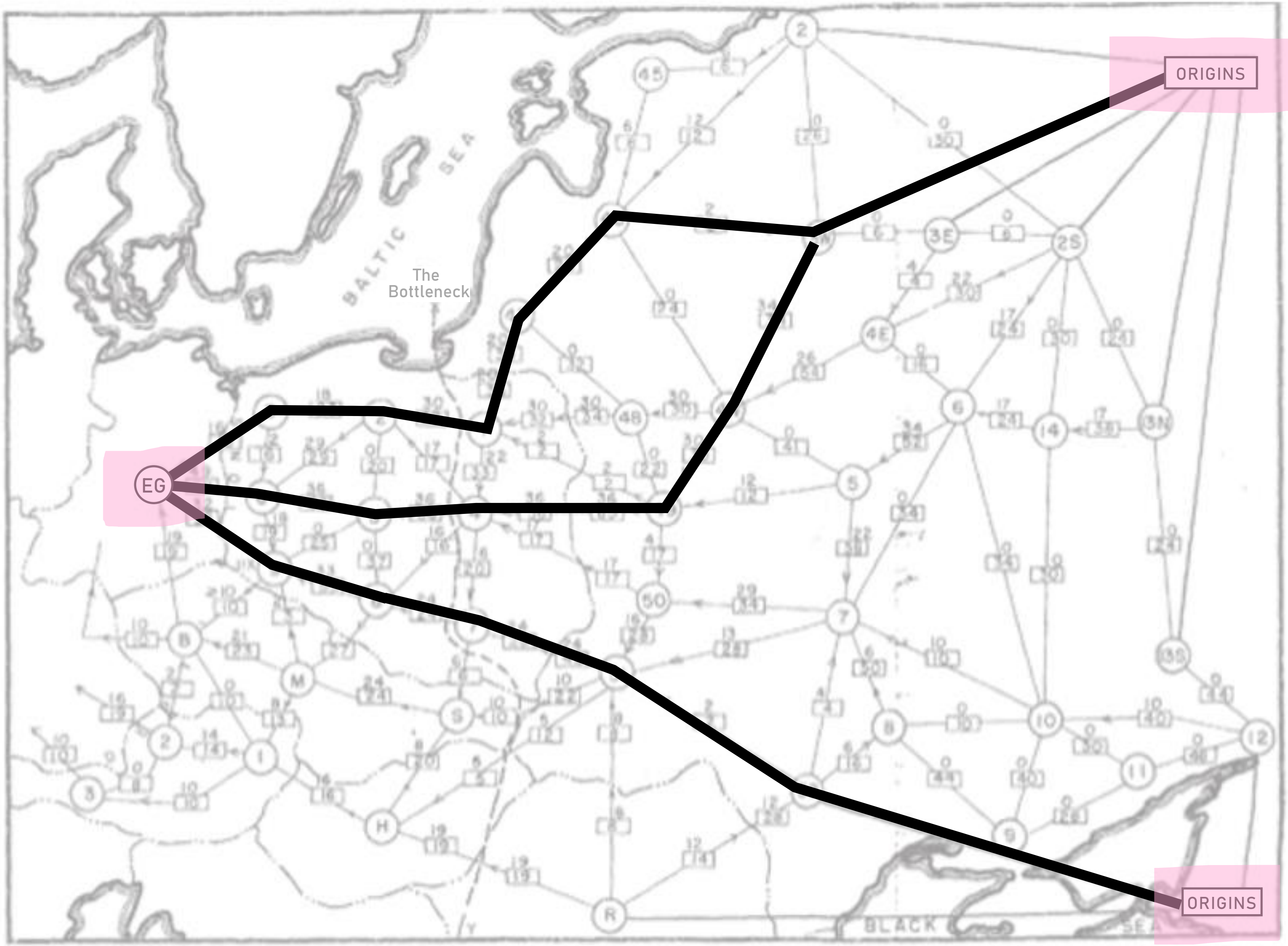


Fig. 7 — Traffic pattern: entire network available

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, 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

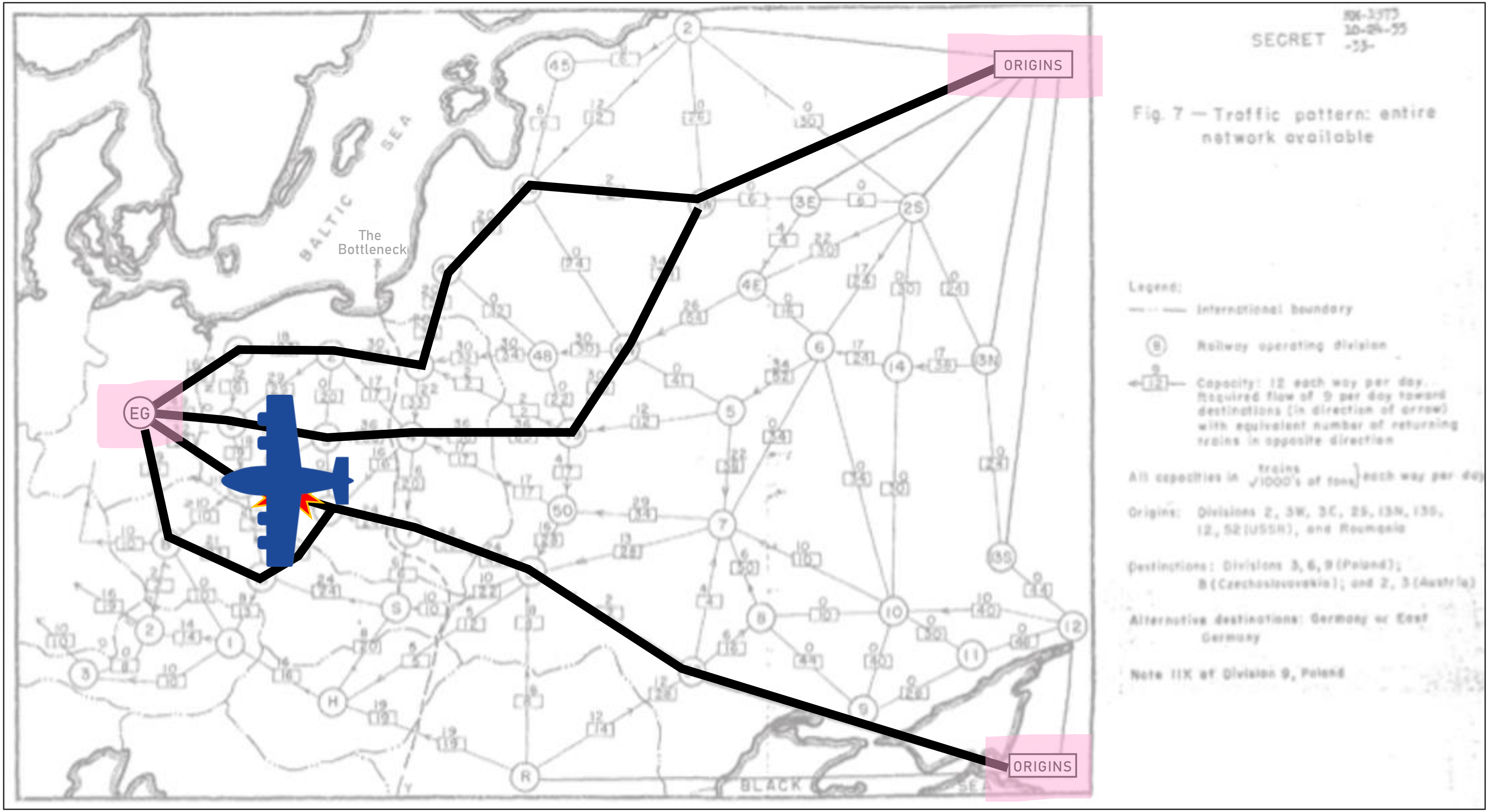
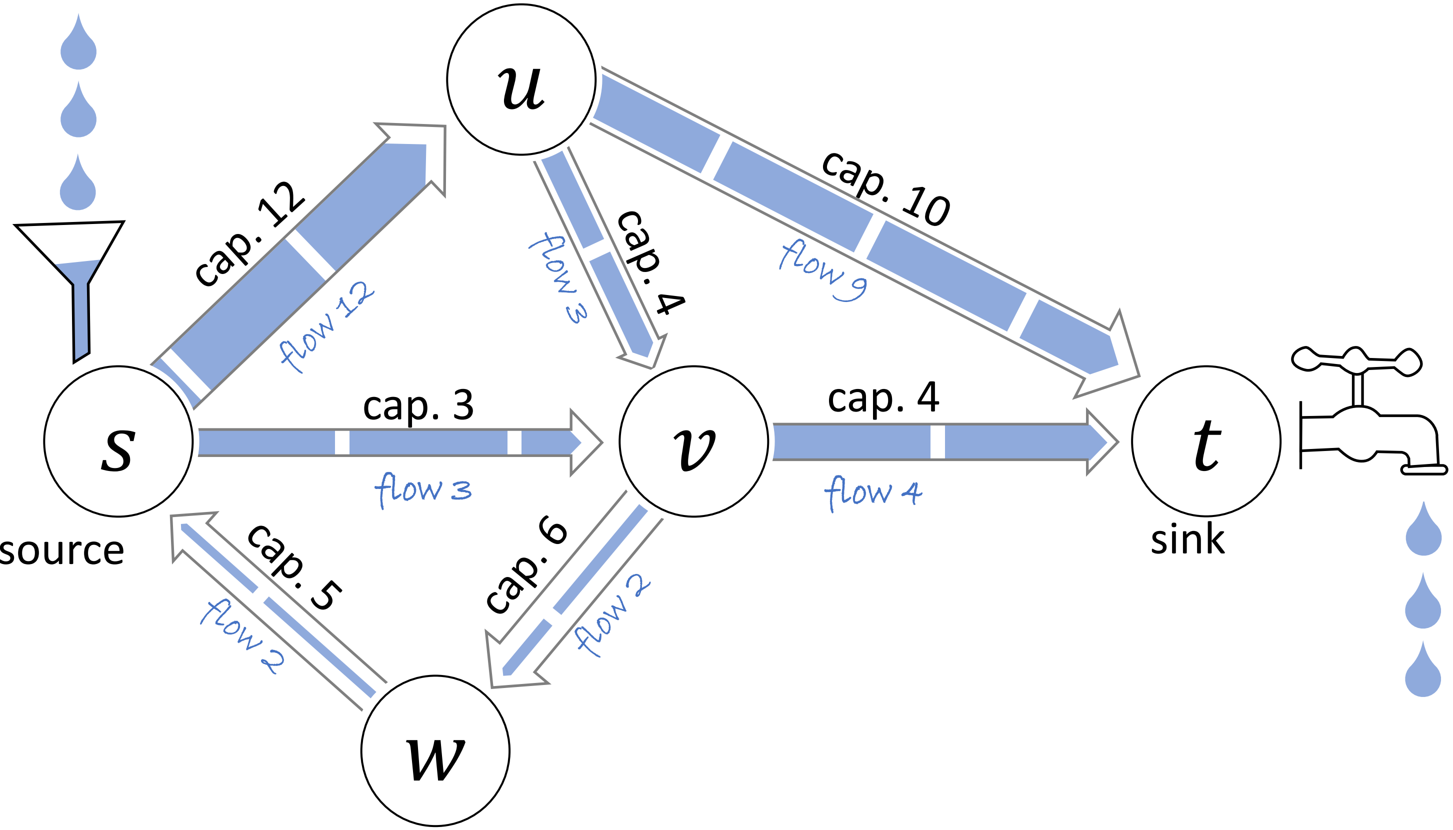


Fig. 7 — Traffic pattern: entire network available



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

and the **value of the flow** is

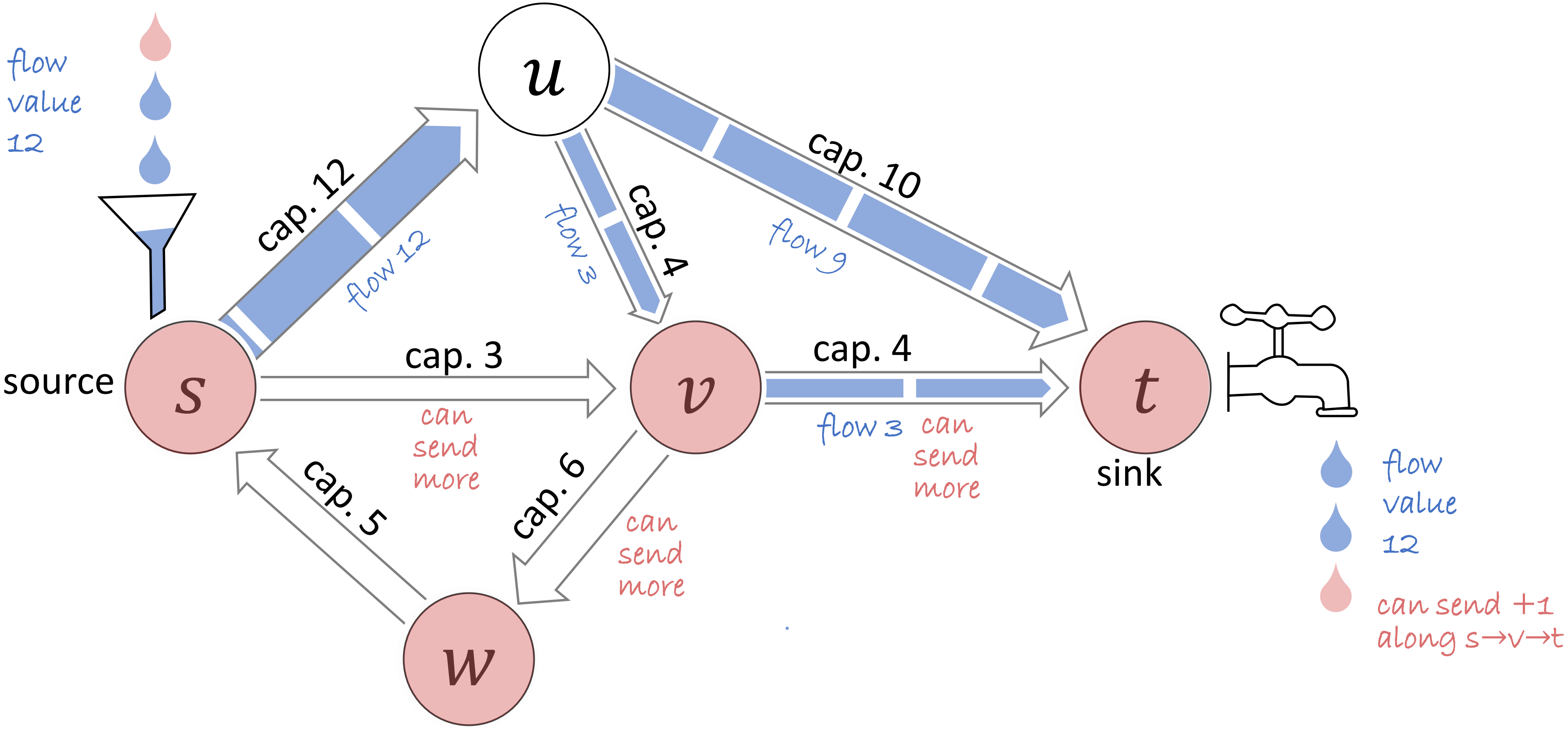
- value(f) = net flow out of s = net flow into t

PROBLEM STATEMENT

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

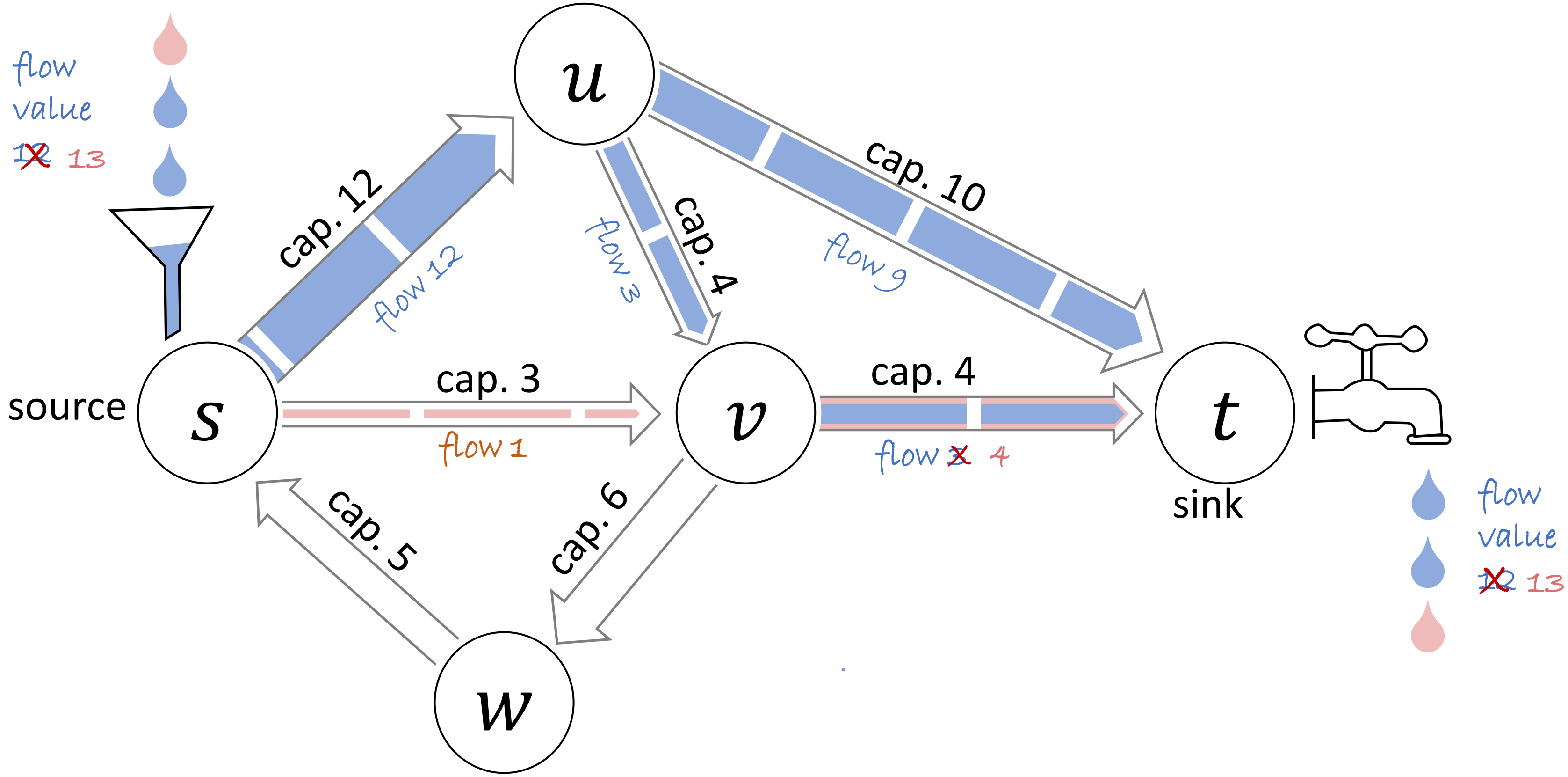
SECTION 6.2

Ford-Fulkerson algorithm



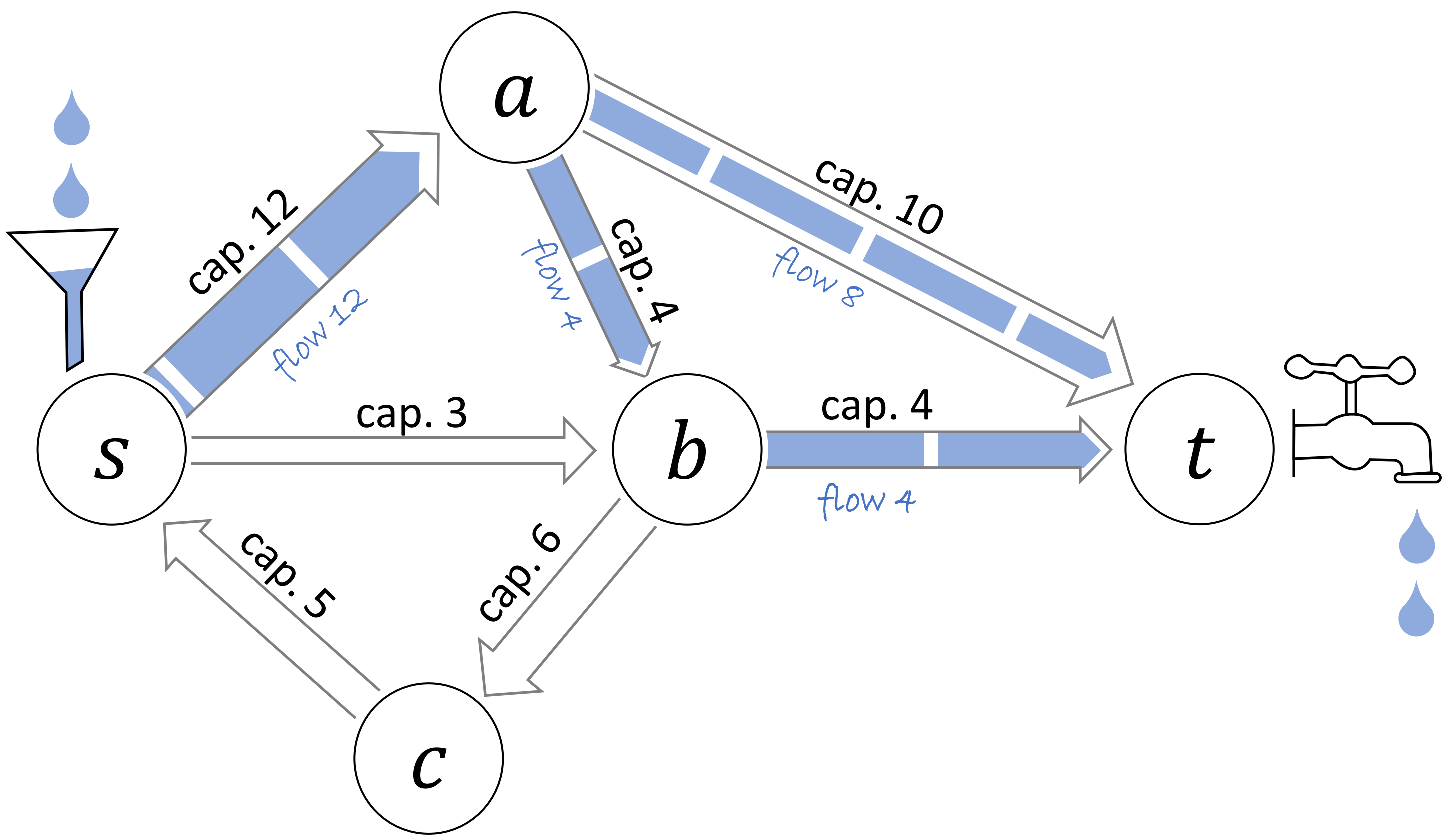
SIMPLE GREEDY STRATEGY

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.

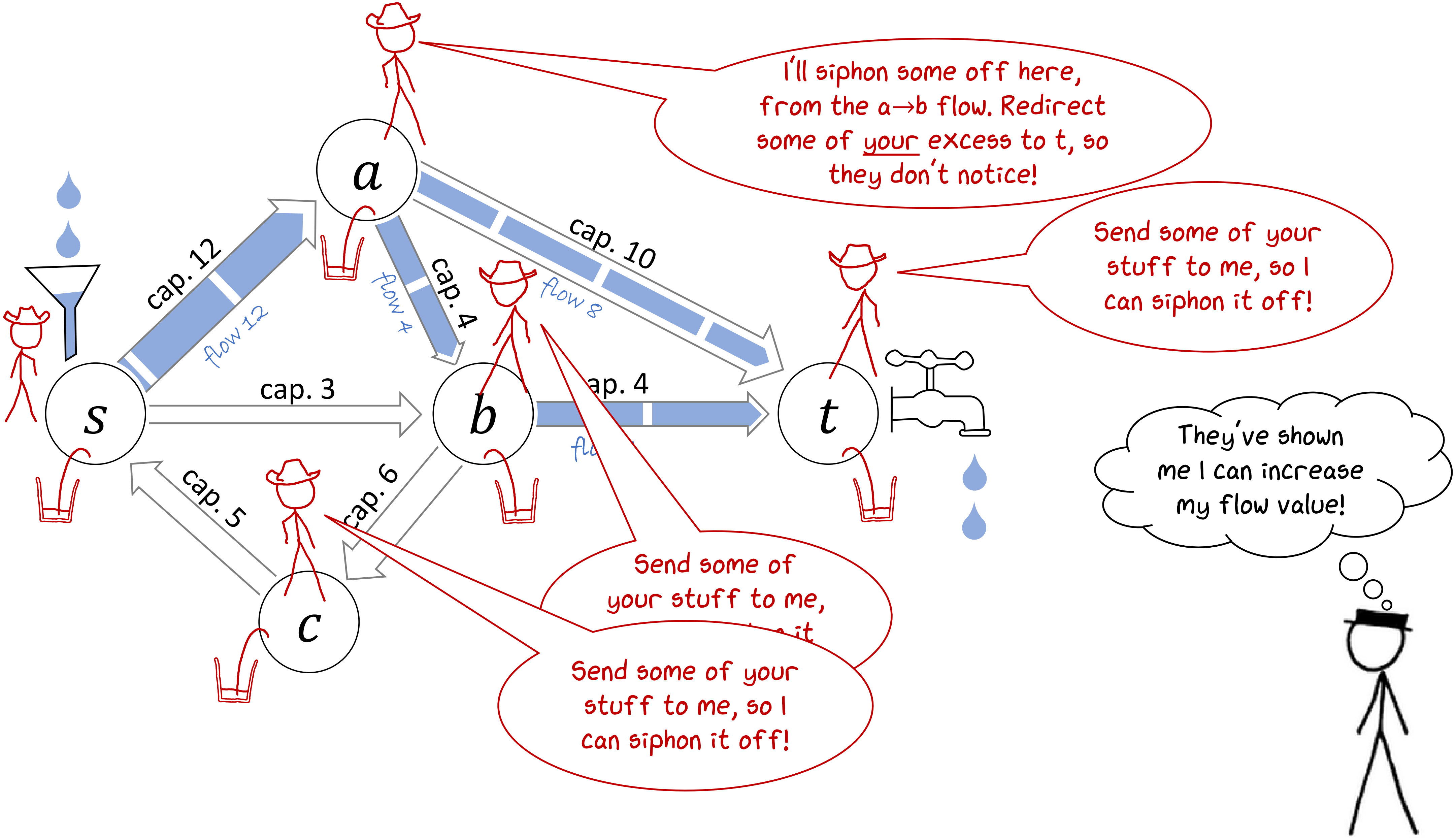


SIMPLE GREEDY STRATEGY

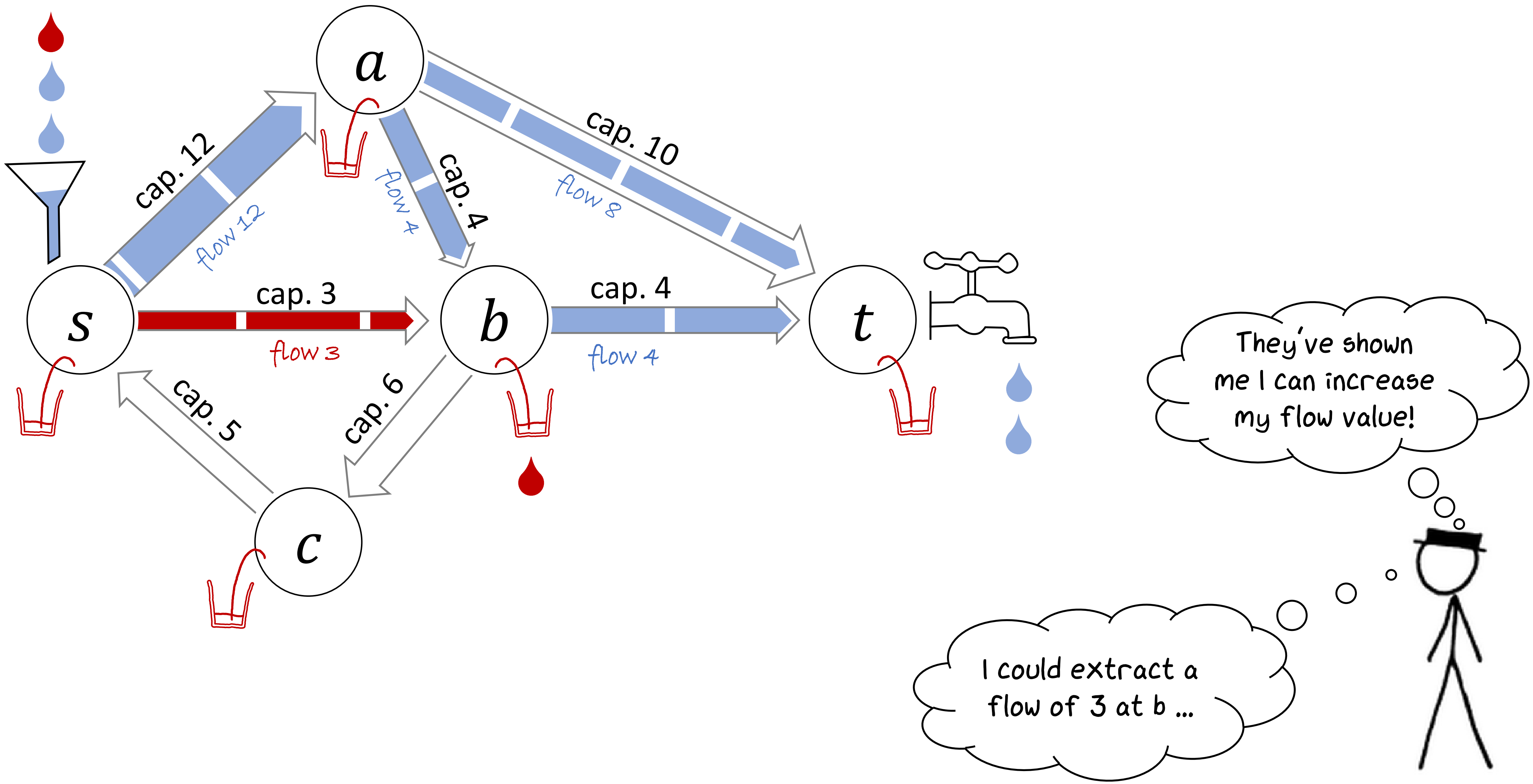
Look for a path to the sink along which we can increase flow, then increase it as much as we can.
Repeat this, until we can't reach the sink.



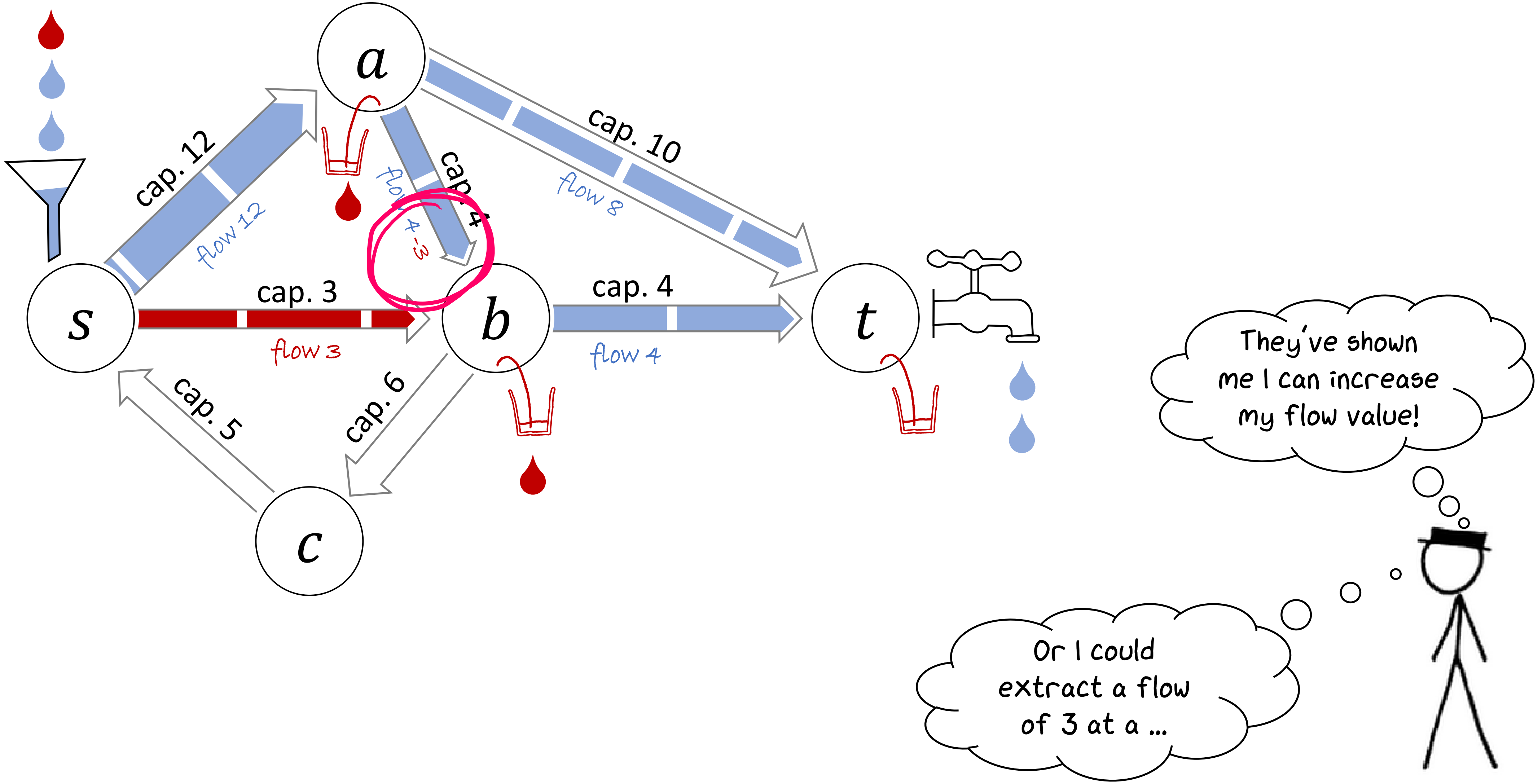
QUESTION. Can you find a larger-value flow than this?



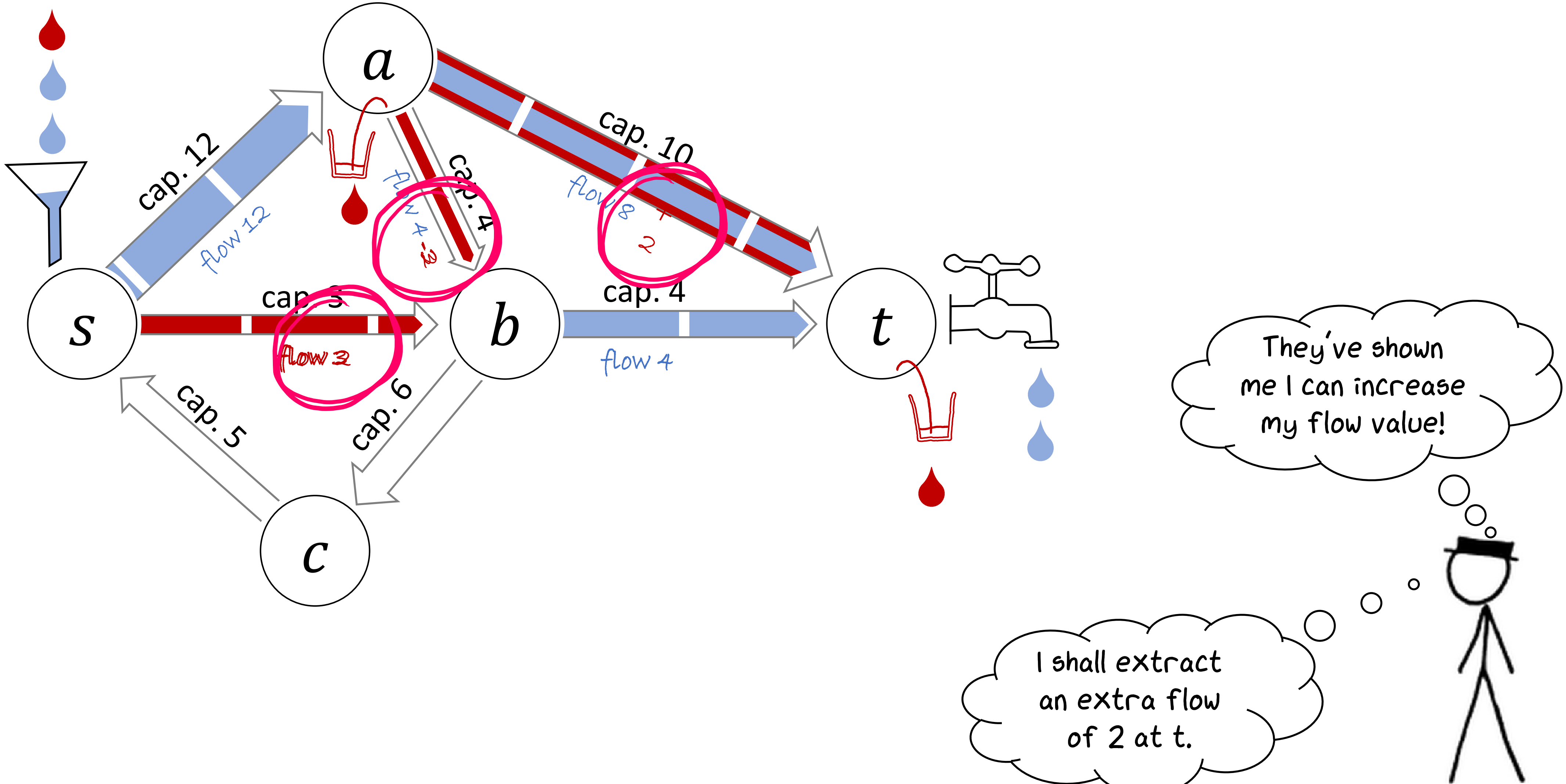
I'll siphon some off here, from the a→b flow. Redirect some of your excess to t, so they don't notice!
Send some of your stuff to me, so I can siphon it off!
Send some of your stuff to me,
Send some of your stuff to me, so I can siphon it off!
They've shown me I can increase my flow value!
a
s
b
c
t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

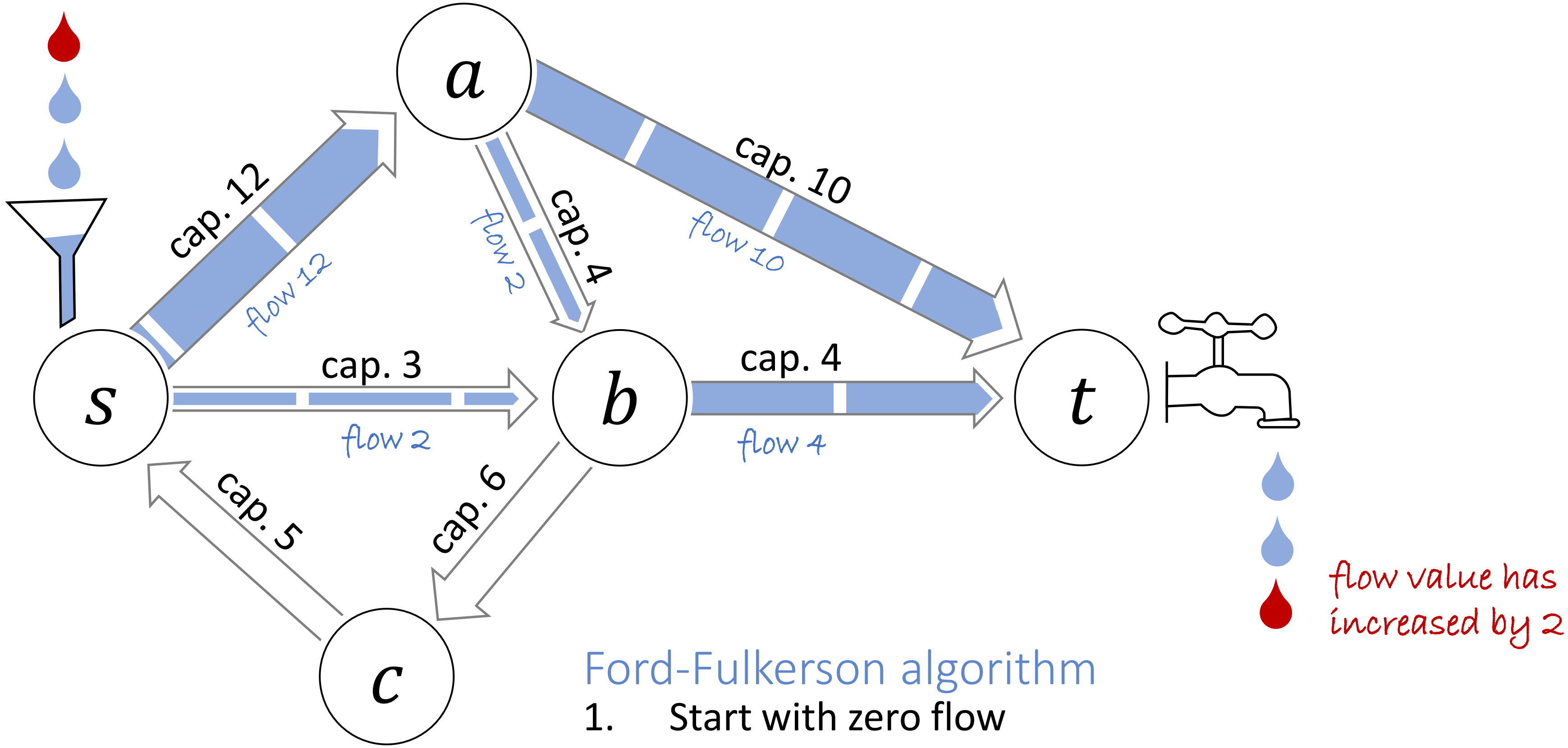


a
s
b
t
c
cap. 12
flow 12
cap. 4
flow 4
cap. 10
flow 8
cap. 3
flow 3
cap. 4
flow 4
cap. 5
cap. 6
They've shown me I can increase my flow value!
I could extract a flow of 3 at b ...



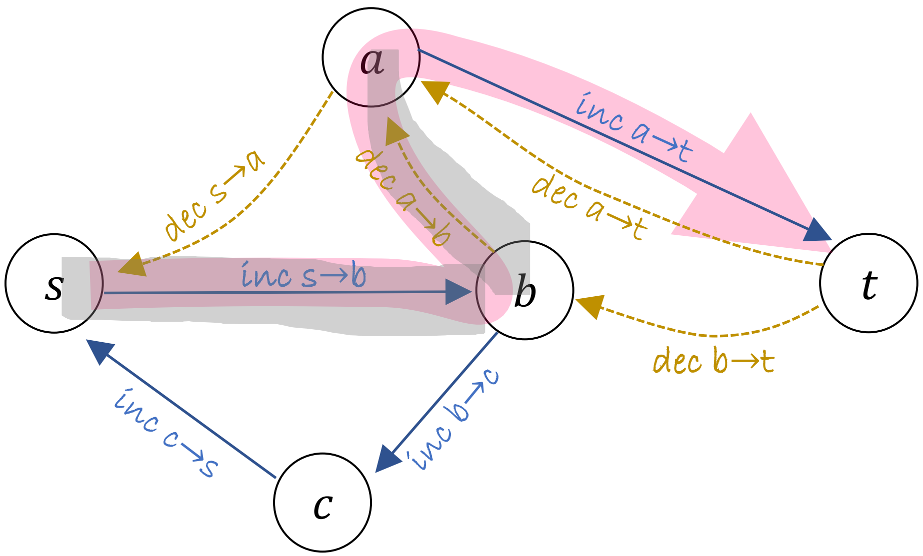
a
s
b
t
c
cap. 12
flow 12
cap. 3
flow 3
cap. 4
flow 4-3
cap. 10
flow 8
cap. 4
flow 4
cap. 5
cap. 6
They've shown me I can increase my flow value!
Or I could extract a flow of 3 at a ...

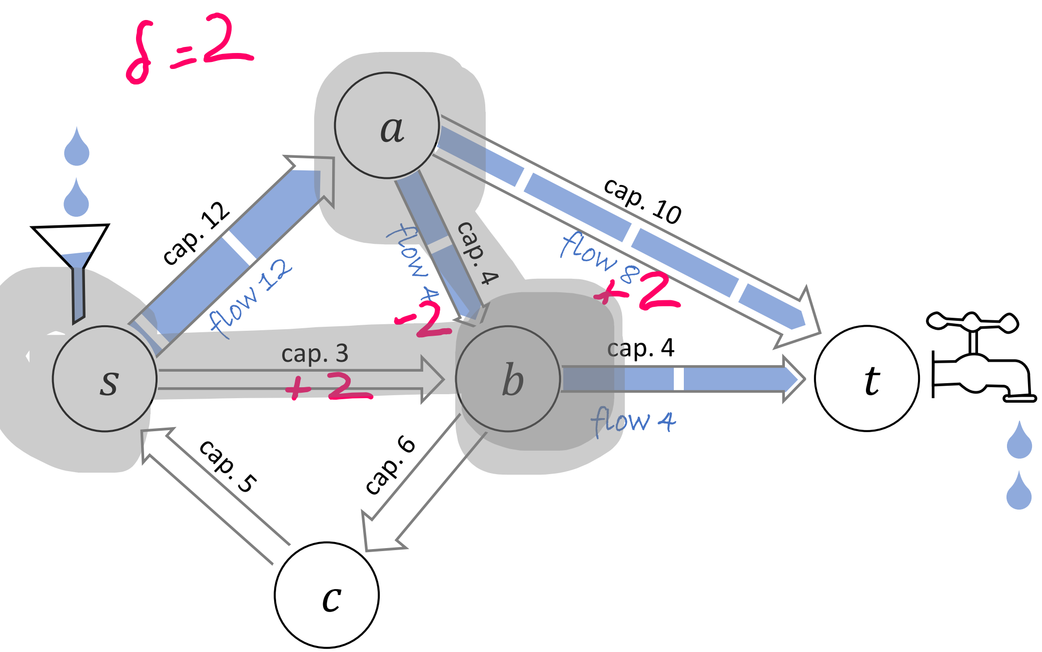

a
b
c
s
t
cap. 12
flow 12
cap. 10
flow 8 + 2
cap. 4
flow 4 -2
cap. 5
flow 3
cap. 4
flow 4
cap. 5
cap. 6
They've shown me I can increase my flow value!
I shall extract an extra flow of 2 at t.



Ford-Fulkerson algorithm

1. Start with zero flow
2. Run bandit search to discover if the flow to t can be increased, and if so find an appropriate sequence of edges
3. If t can be reached:
   update the flow along those edges, then go back to step 2
4. If t can't be reached: terminate.



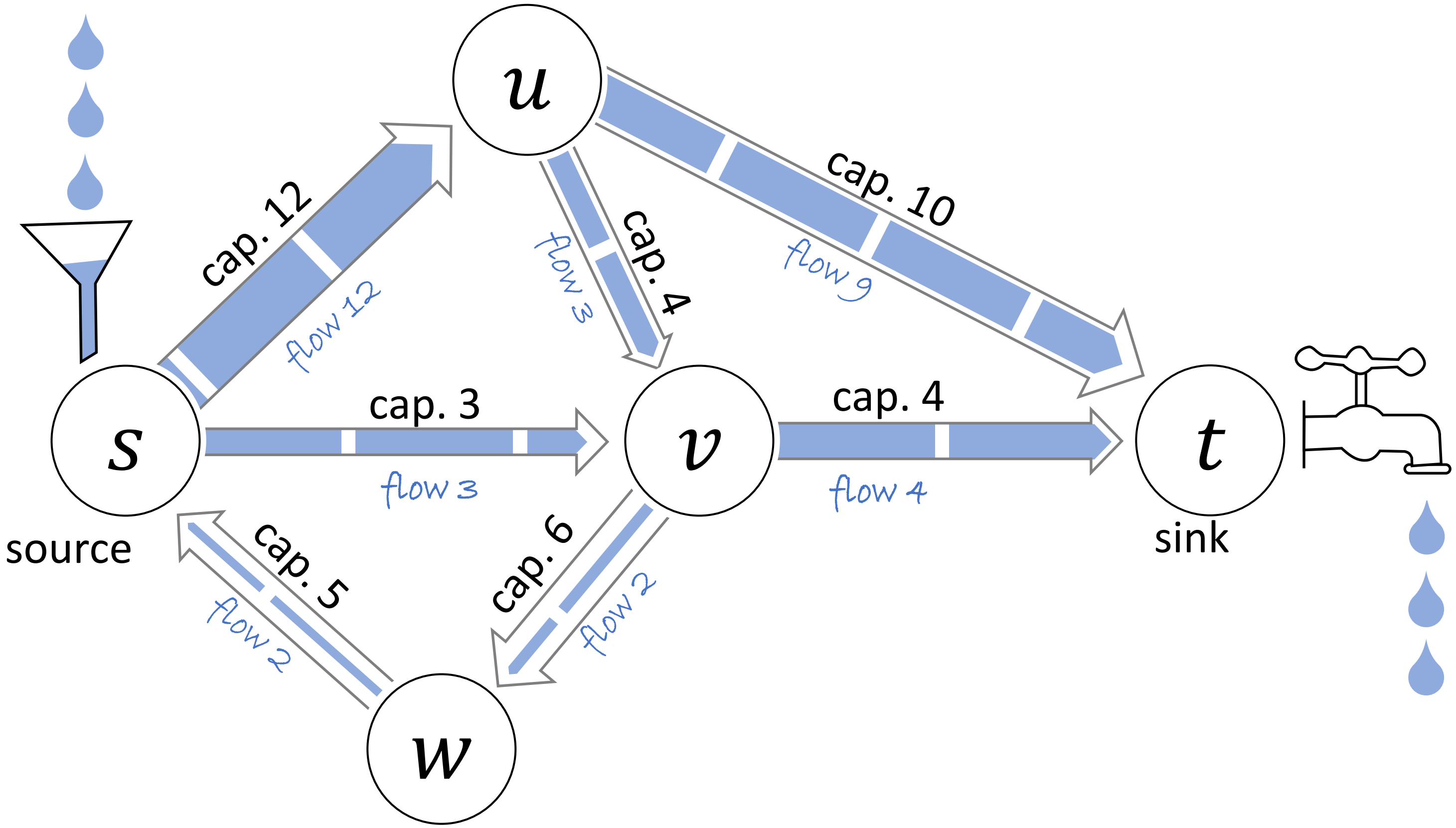


STEP 2A. Build the **residual graph,** which has the same vertices as the flow network, and

- 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 $u \to v$"

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.



EXERCISE. Find a way to increase the flow value.

6. Graphs and subgraphs

Lecture 17 6.1 Flow networks (9:31) code — subgraphs

6.2 Ford-Fulkerson algorithm (21:55)

Lecture 18

Lecture 19

Algorithms tick max-flow

cl.cam.ac.uk/teaching/2223/Algorithm2/ticks/max-flow.html

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.

```
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
```

The Integrality Lemma. If the capacities are all integers, then the algorithm terminates, and the resulting flow on each edge is an integer. The running time is $O(\mathrm{val}(f^*) \times E)$ where f^* is a max flow.

6. Graphs and subgraphs

Lecture 17 6.1 Flow networks (9:31) code — subgraphs

6.2 Ford-Fulkerson algorithm (21:55)

Lecture 18

Lecture 19

Algorithms tick max-flow

cl.cam.ac.uk/teaching/2223/Algorithm2/ticks/max-flow.html

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.

[2203.00671] Maximum Flow and

https://arxiv.org/abs/2203.00671

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.

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)