A&W 2026

G-13 Timo Stucki, LFW E 13

Week 11

Minitest 6

Dynamic Programming

Applications of Networks and Flows with Exercise

Min-Cut

Smallest Enclosing Circle

Infos

ETH zürich

Feel free to contact me:

- tistucki@student.ethz.ch
- Discord: timostucki

Material:

- timostucki.com

Minitest 6

Dynamic Programming Exercises Recap

Neighbourhood Burglary, Airport Security

Networks and Flows

Recap

Networks and Flows

Definition 3.4. Ein *Netzwerk* ist ein Tupel $N = (V, A, c, s, t)$, wobei gilt:

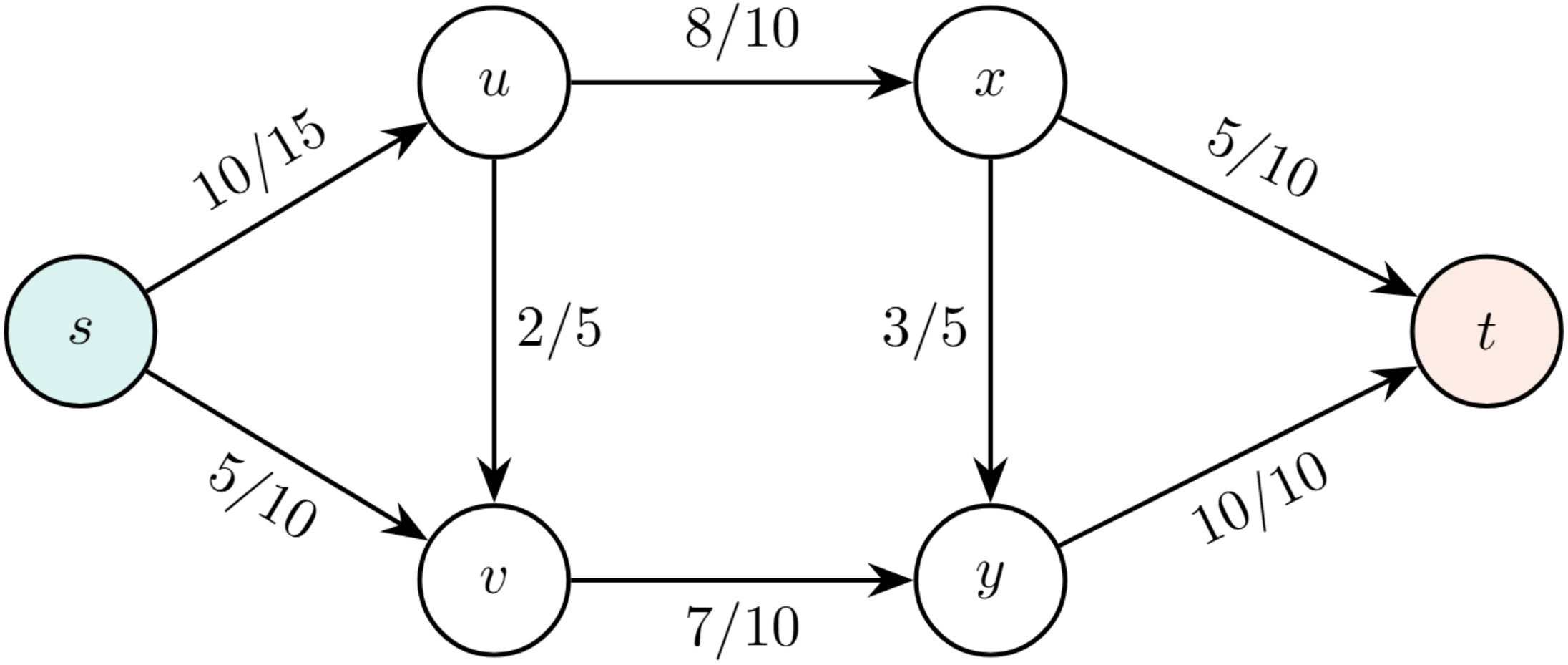
(V, A) ist ein gerichteter Graph,

$s \in V$, die *Quelle* (engl.: source),

$t \in V \setminus \{s\}$, die *Senke* (engl.: target), und

$c : A \to \mathbb{R}_0^+$, die *Kapazitätsfunktion* (engl.: capacity function).

Networks and Flows



Networks and Flows

Definition 3.5. Gegeben sei ein Netzwerk $N = (V, A, c, s, t)$. Ein Fluss in N ist eine Funktion $f : A \to \mathbb{R}$ mit den Bedingungen

$0 \leq f(e) \leq c(e)$ für alle $e \in A$, die *Zulässigkeit*, und

für alle $v \in V \setminus \{s, t\}$ gilt

$$\sum_{u \in V:\,(u,v) \in A} f(u, v) = \sum_{u \in V:\,(v,u) \in A} f(v, u)$$

die *Flusserhaltung*.

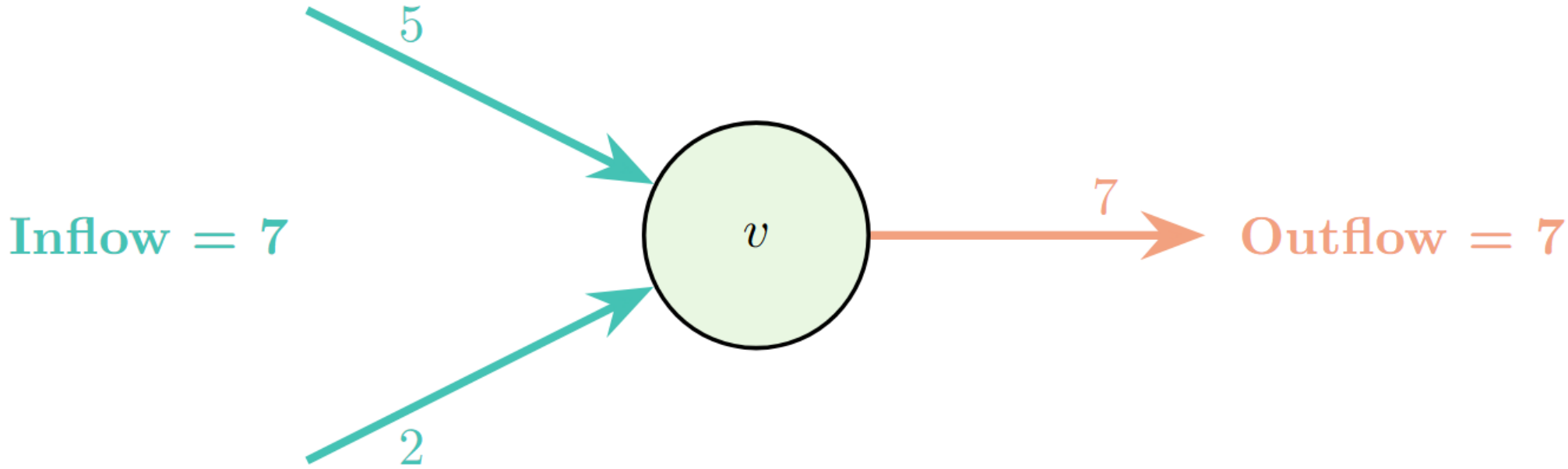
Der *Wert* (engl.: value) eines Flusses f ist durch

$$\mathsf{val}(f) := \mathsf{netoutflow}(s) := \sum_{u \in V:\,(s,u) \in A} f(s, u) - \sum_{u \in V:\,(u,s) \in A} f(u, s)$$

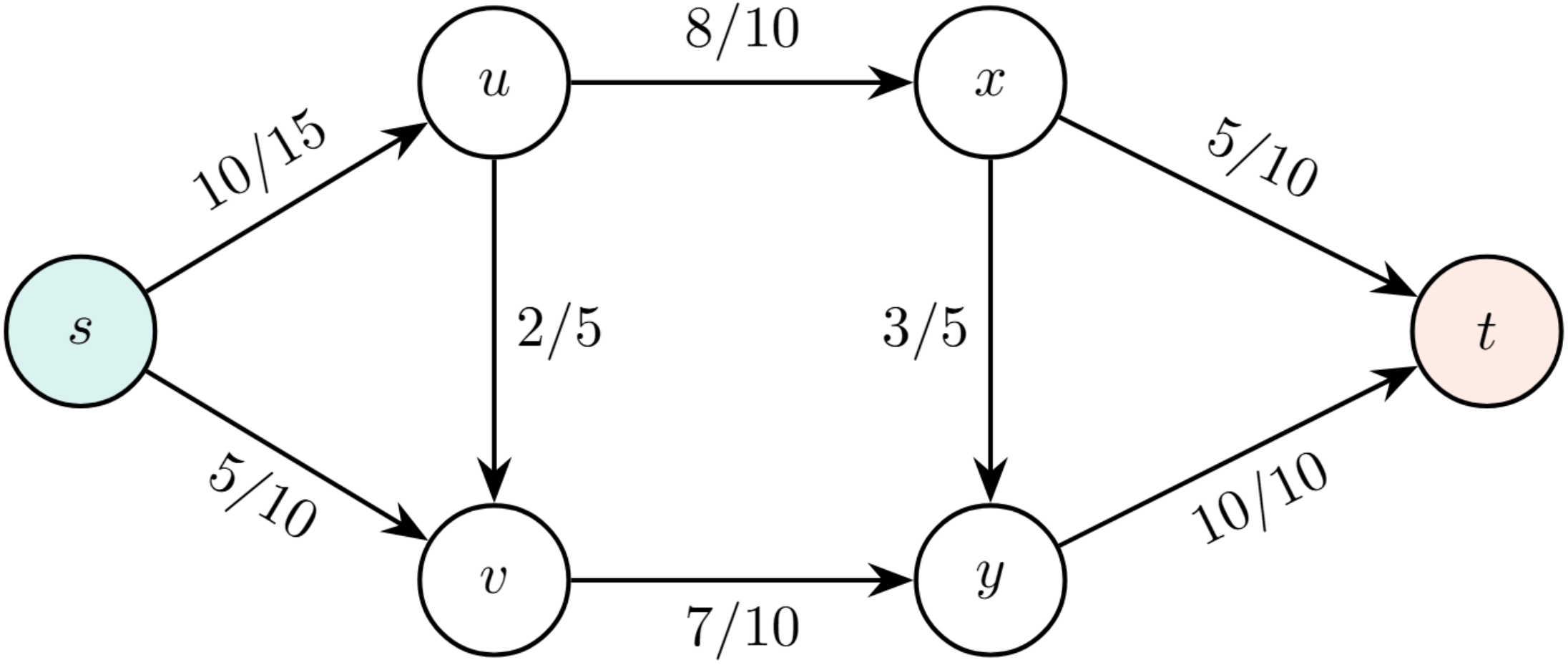
definiert. Wir nennen f *ganzzahlig*, wenn $f(e) \in \mathbb{Z}\ \forall e \in A$.

Networks and Flows

For an intermediate node $v \in V \setminus \{s, t\}$, the inflow must match the outflow.



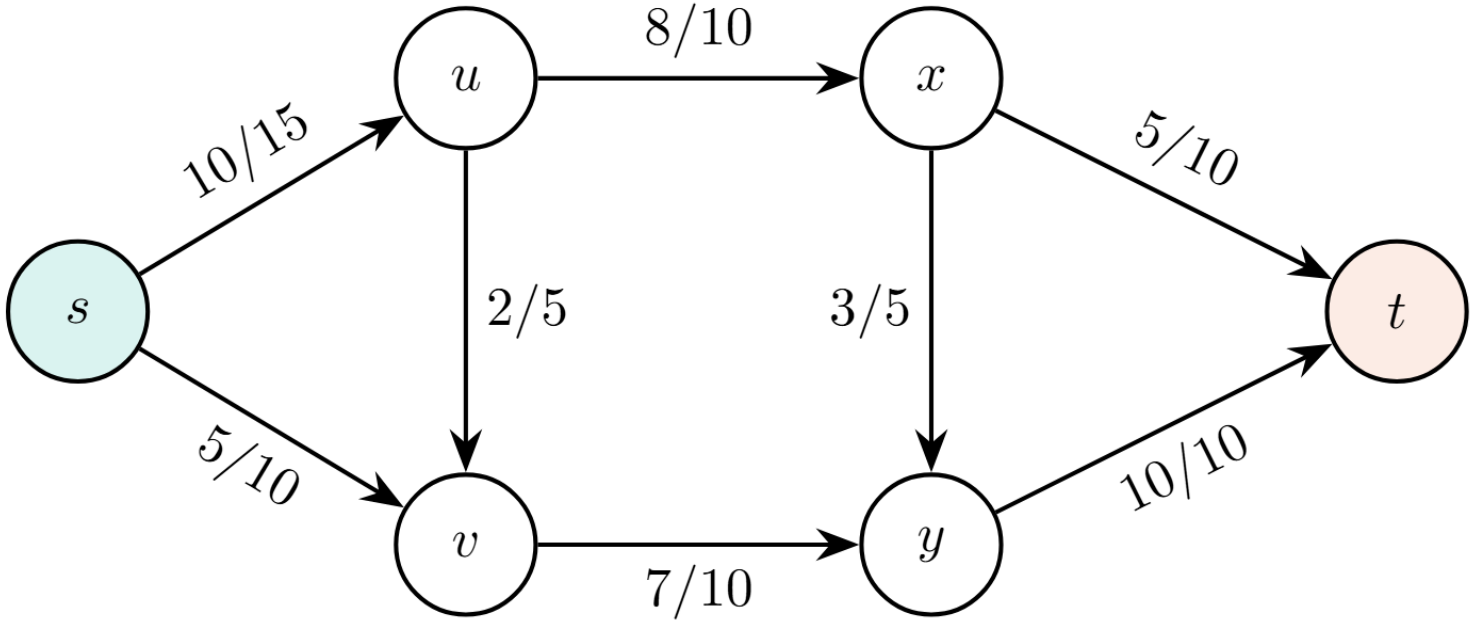
Networks and Flows



Networks and Flows

Lemma 3.6. Der Nettozufluss der Senke gleicht dem Wert des Flusses, d.h.

$$\text{netinflow}(t) := \sum_{u \in V:\,(u,t) \in A} f(u,t) - \sum_{u \in V:\,(t,u) \in A} f(t,u) \quad = \quad \text{val}(f)\ .$$



Networks and Flows

Lemma 3.8. Ist f ein Fluss und (S, T) ein s-t-Schnitt in einem Netzwerk (V, A, c, s, t), so gilt

$$\mathrm{val}(f) \leq \mathrm{cap}(S, T) \ .$$

Networks and Flows

Lemma 3.8. Ist f ein Fluss und (S, T) ein s-t-Schnitt in einem Netzwerk (V, A, c, s, t), so gilt

$$\mathrm{val}(f) \leq \mathrm{cap}(S, T) \ .$$

Beweis. Für eine Partition (U, W) von V setzen wir

$$f(U, W) := \sum_{(u,w) \in (U \times W) \cap A} f(u, w) \ .$$

Nun behaupten wir

$$\mathrm{val}(f) \overset{(i)}{=} f(S, T) - f(T, S) \overset{(ii)}{\leq} f(S, T) \overset{(iii)}{\leq} \mathrm{cap}(S, T), \quad \ldots$$

Networks and Flows

Lemma 3.8. Ist f ein Fluss und (S, T) ein s-t-Schnitt in einem Netzwerk (V, A, c, s, t), so gilt

$$\mathrm{val}(f) \leq \mathrm{cap}(S, T) \ .$$

Satz 3.9 („Maxflow-Mincut Theorem"). Jedes Netzwerk $N = (V, A, c, s, t)$ erfüllt

$$\max_{f \text{ Fluss in } N} \mathrm{val}(f) = \min_{(S,T) \ s\text{-}t\text{-Schnitt in } N} \mathrm{cap}(S, T)$$

Networks and Flows

Definition 3.10. Sei $N = (V, A, c, s, t)$ ein Netzwerk ohne entgegen gerichtete Kanten und sei f ein Fluss in N. Das Restnetzwerk $N_f := (V, A_f, r_f, s, t)$ ist wie folgt definiert:

(1) Ist $e \in A$ mit $f(e) < c(e)$, dann ist e auch eine Kante in A_f, mit $r_f(e) := c(e) - f(e)$.

(2) Ist $e \in A$ mit $f(e) > 0$, dann ist e^{opp} in A_f, mit $r_f(e^{opp}) = f(e)$.

(3) Nur Kanten wie in (1) und (2) beschrieben finden sich in A_f.

$r_f(e)$, $e \in A_f$, nennen wir die *Restkapazität* der Kante e.

Networks and Flows

Satz 3.11. Ein Fluss f in einem Netzwerk N ist ein maximaler Fluss gdw. es im Restnetzwerk N_f keinen gerichteten Pfad von der Quelle s zur Senke t gibt. Für jeden solchen maximalen Fluss gibt es einen s-t-Schnitt (S, T) mit $\mathrm{val}(f) = \mathrm{cap}(S, T)$.

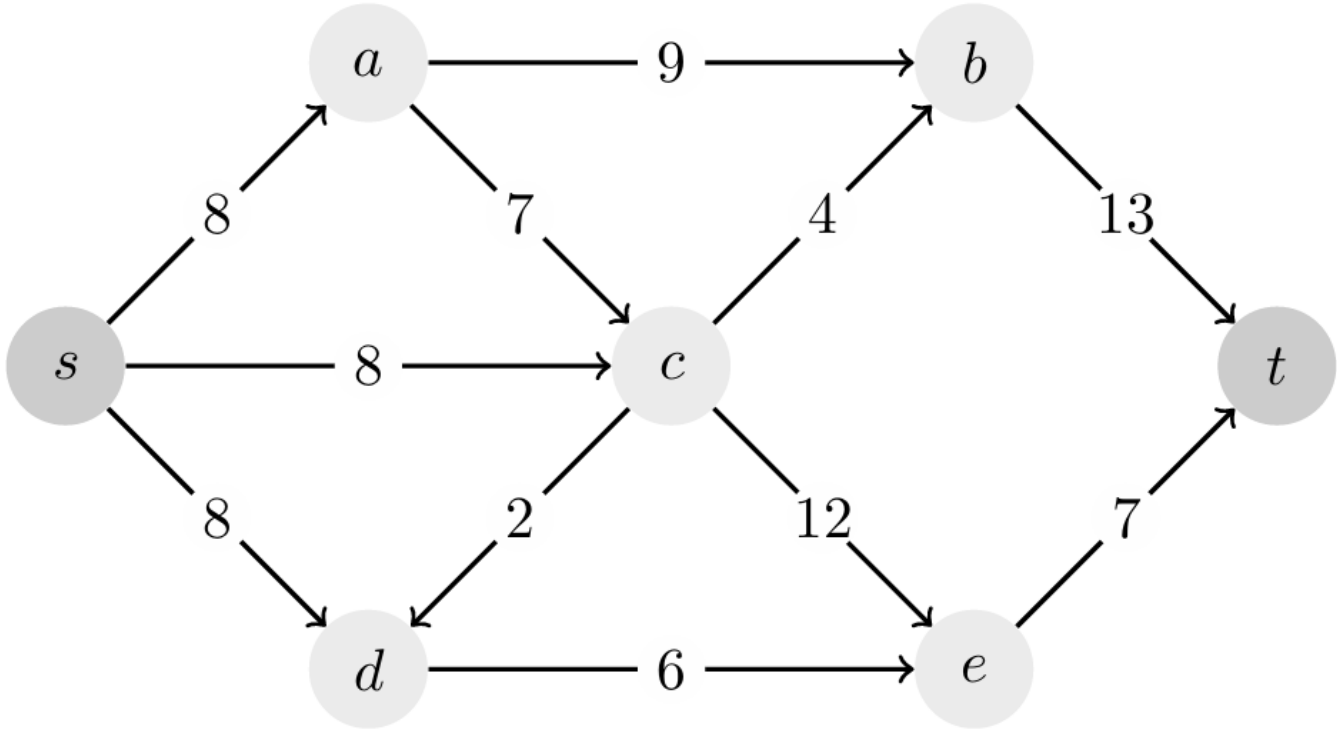
→ **Let's see this in an exercise**

Exercise S11

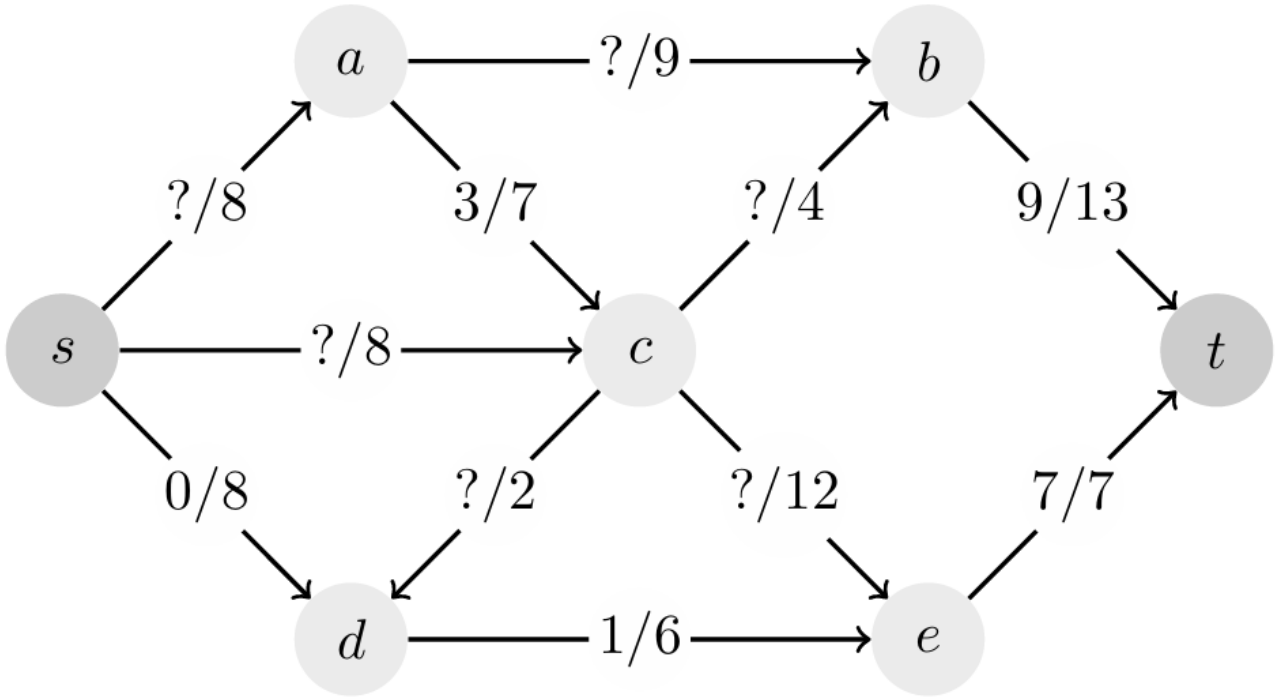
Flows

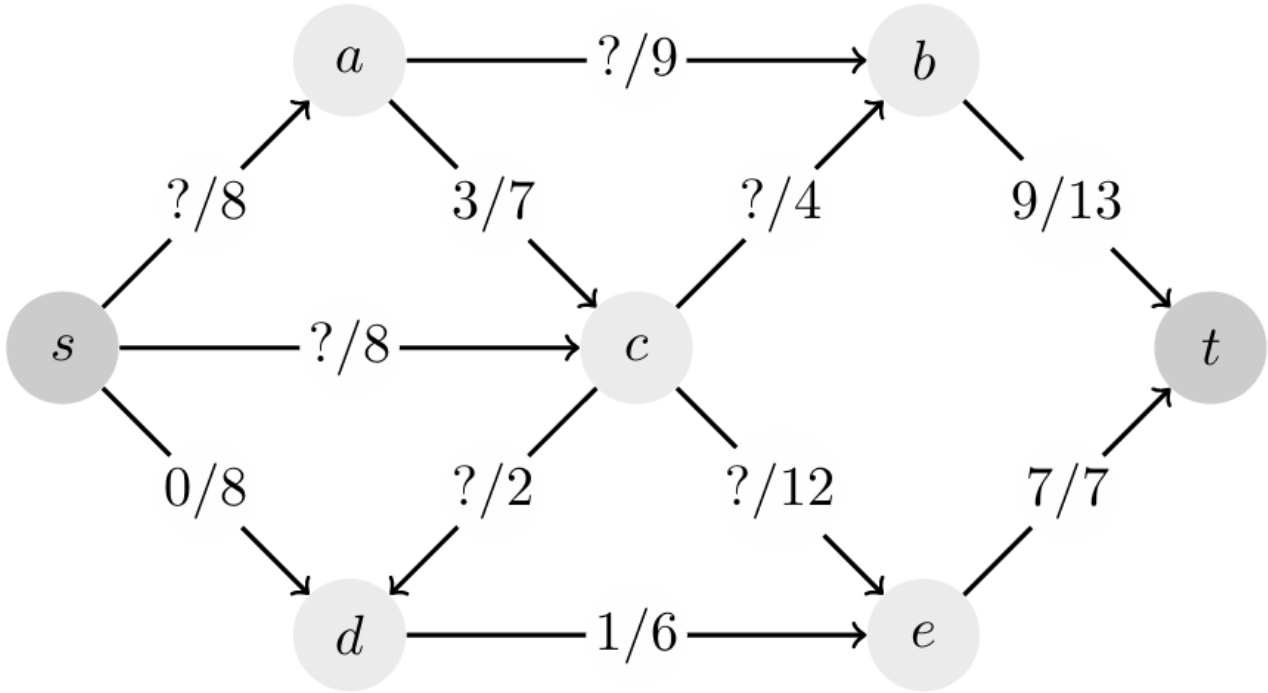
Exercise S11.1 – *Flows*

Consider the following network N. The number at each edge represents its capacity.



(a) Fill in the missing values so that they form a feasible (not necessarily maximum) flow:





(b) Construct the residual network ("Restnetzwerk").

(c) Find an augmenting s-t-path and augment the flow along this path. If necessary, repeat this step until you have found a maximum flow.

(d) Prove that your flow is a maximum flow by finding a minimum cut in N.

Networks and Flows

Recap

Networks and Flows

FORD-FULKERSON(V, A, c, s, t)

1: $f \leftarrow \mathbf{0}$ ▷ Fluss konstant 0
2: **while** $\exists$ s-t-Pfad P in (V, A_f) **do** ▷ augmentierender Pfad
3: Erhöhe den Fluss entlang P ▷ wie in Beweis zu Satz 3.11
4: **return** f ▷ maximaler Fluss

Networks and Flows

FORD-FULKERSON(V, A, c, s, t)

1: $f \leftarrow \mathbf{0}$ ▷ Fluss konstant 0
2: **while** $\exists$ s-t-Pfad P in (V, A_f) **do** ▷ augmentierender Pfad
3: Erhöhe den Fluss entlang P ▷ wie in Beweis zu Satz 3.11
4: **return** f ▷ maximaler Fluss

Satz 3.12. Sind in einem Netzwerk ohne entgegen gerichtete Kanten alle Kapazitäten ganzzahlig und höchstens U, so gibt es einen ganzzahligen maximalen Fluss, der in Zeit $O(mnU)$ berechnet werden kann (m ist die Anzahl Kanten, n die Anzahl Knoten im Netzwerk).

Networks and Flows

Number of Iterations

The value of any flow f is bounded by the capacity of the cut $(\{s\}, V \setminus \{s\})$. Since there are at most $n-1$ edges leaving the source s, each with capacity $\leq U$, the maximum flow value is bounded by:

$$val(f) \leq cap(\{s\}, V \setminus \{s\}) \leq (n-1)U < nU$$

Because the total flow value cannot exceed nU, the algorithm terminates after at most nU **iterations**.

Complexity per Iteration

In each iteration, the algorithm performs the following steps:

- Construction of the **residual network** N_f.
- Search for an **augmenting path** from source s to sink t (e.g., using BFS or DFS).
- These operations can be completed in $O(m)$ **time**.

$$\text{Total Runtime} = O(nU) \cdot O(m) = \mathbf{O(mnU)}$$

Applications of Networks and Flows

Applications of Networks and Flows

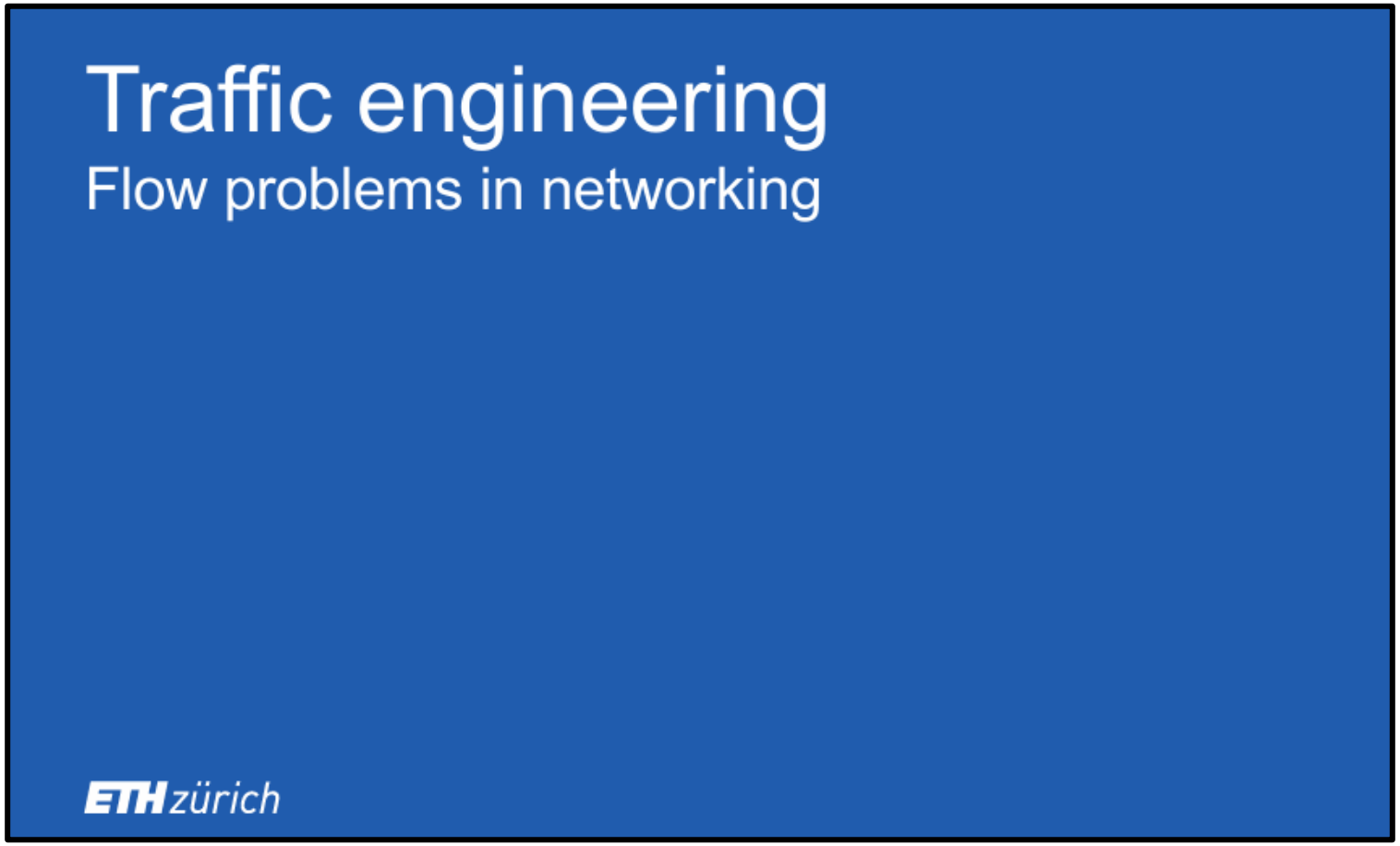
You have seen in lecture:

- Matchings, hier: bipartites maximum Matching in $O(mn)$ (geht besser in $O((m+n)\sqrt{n})$ [Hopcroft&Karp'73]).
- Schnitten zwischen Knoten u und v (Knoten-/Kantenzusammenhang).
- Kantendisjunkten Pfaden (auch knotendisjunkten Pfaden).
- Beweis Satz von Menger (aus Maxflow-Mincut).

Bildsegmentierung. Gegeben ein Bild (aus Pixeln mit Farbwerten), trenne Vordergrund von Hintergrund.

Applications of Networks and Flows

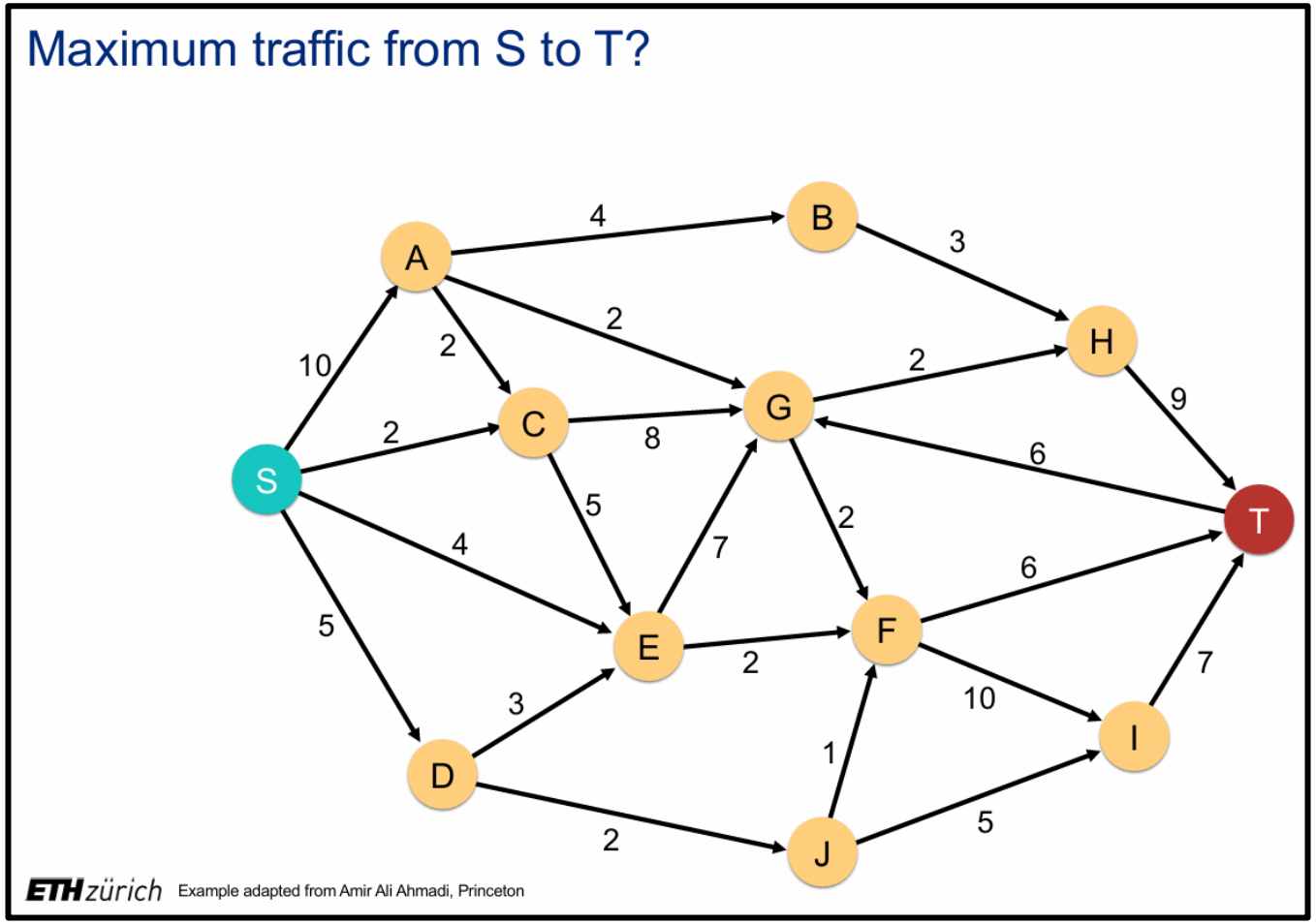
What **you** will 100% use it for:



“Computer Networks” lecture FS26

Applications of Networks and Flows

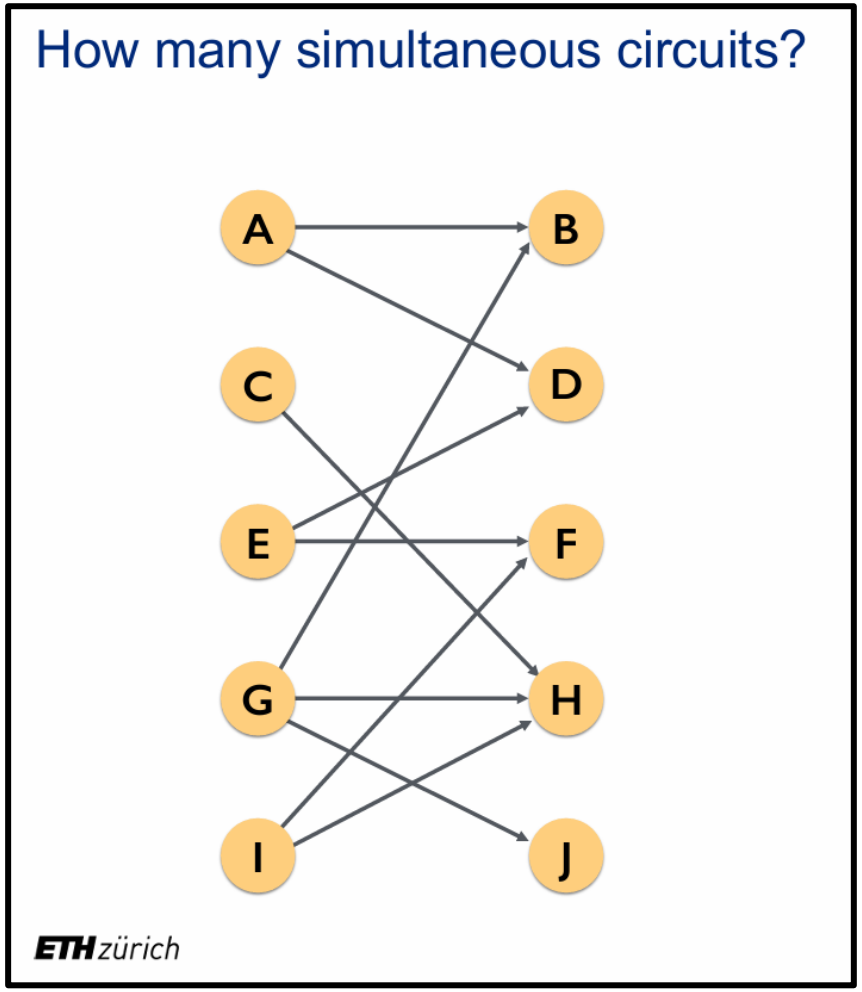
What **you** will 100% use it for:



"Computer Networks" lecture FS26

Applications of Networks and Flows

What **you** will 100% use it for:



"Computer Networks" lecture FS26

Exercise S12

“Integrality and Matching”

Lemma 3.15. Die maximale Grösse eines Matchings im bipartiten Graph G ist gleich dem Wert eines maximalen Flusses im Netzwerk N.

Exercise S12.1 – *Integrality and Matching*

We have already seen Frobenius' Theorem: *Let $k \geq 1$. Every k-regular bipartite graph contains a perfect matching.* The proof we have seen uses Hall's theorem. In this exercise, we want to find another proof using our knowledge about flows.

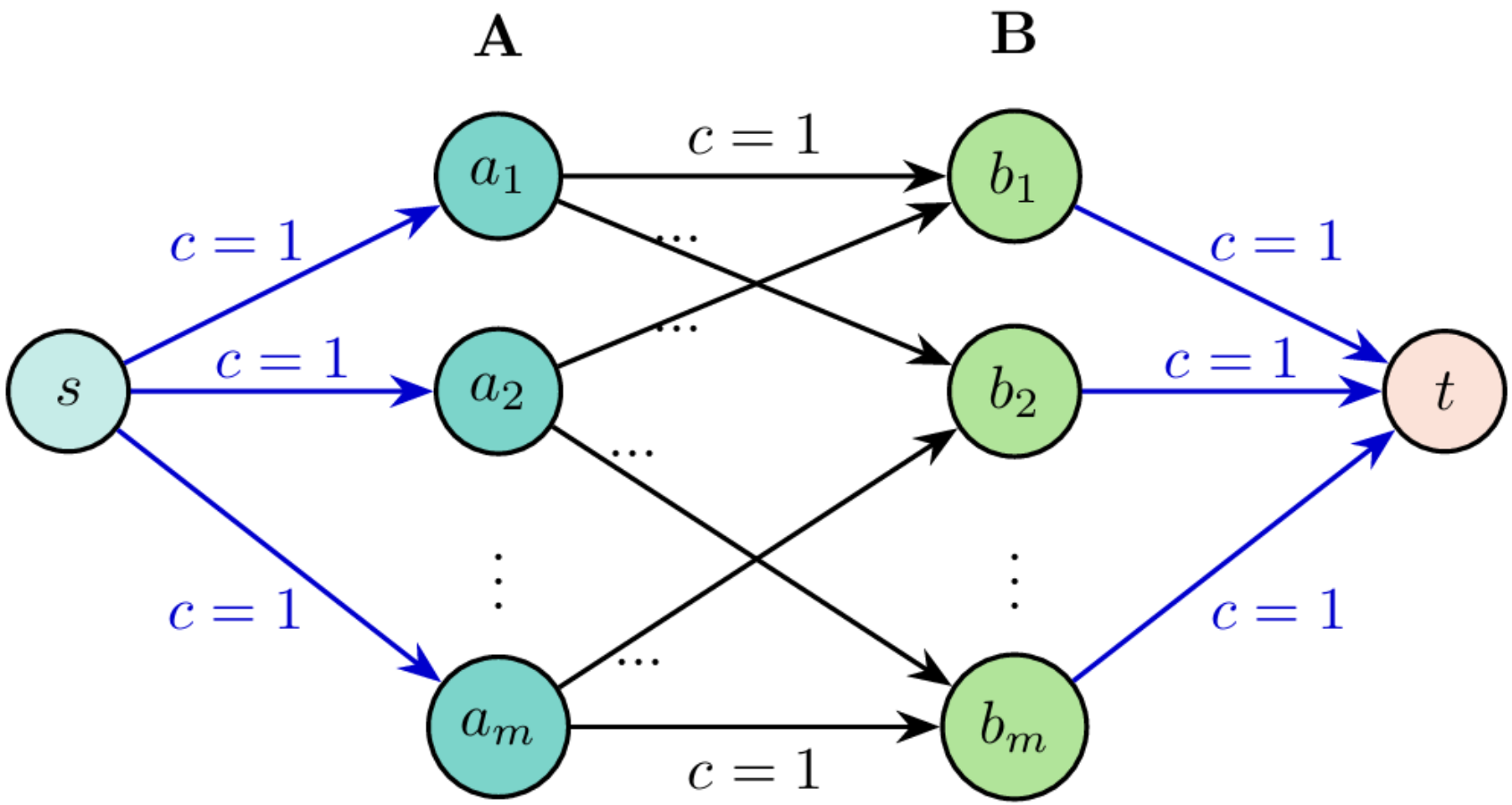
Let $G = (A \dot{\cup} B, E)$ be a k-regular graph for $k \geq 1$.

(a) Describe how to model bipartite matching on G as a flow problem in a network $N = (V, A, c, s, t)$.

(b) Construct explicitly a (not necessarily integer!) flow f on N with $\text{val}(f) = n$. Conclude that $\text{maxflow}(N) = n$.
Hint: you don't have to construct an integral flow. Make sure to define the flow value on all edges of your network N!

(c) Using the result from (b), show that there exists an integer valued flow f' with $\text{val}(f') = n$. Using f', prove that G contains a perfect matching M.

(a) We do the same construction as always: we direct all edges from A to B. Furthermore, we add a source s, and all edges (s, a) for $a \in A$, and we add a sink t and all edges (b, t) for $b \in B$. Each edge gets capacity 1. There is a one to one correspondence between matchings in G and *integral* flows of value $|A| = |B|$ in N. (See Lemma 3.15.)



Exercise S12.1 – *Integrality and Matching*

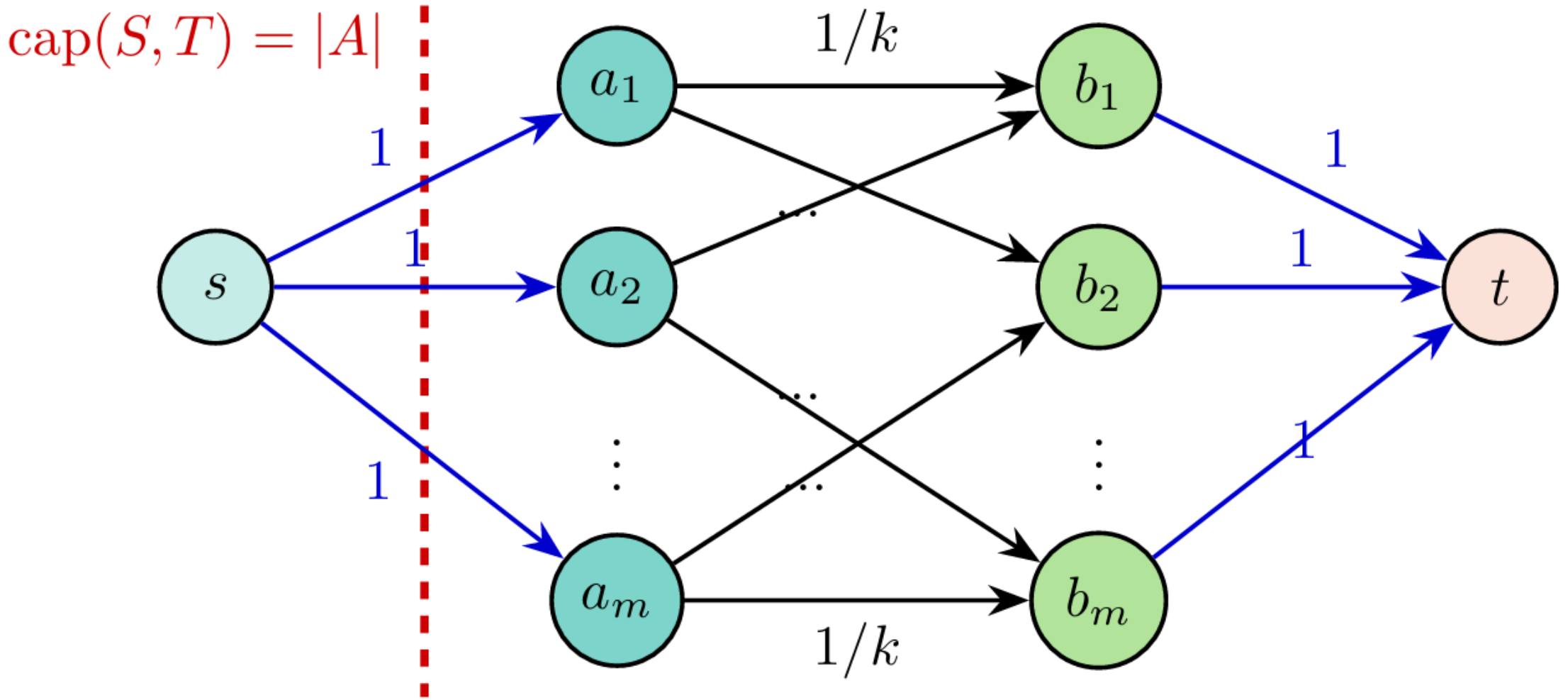
We have already seen Frobenius' Theorem: *Let $k \geq 1$. Every k-regular bipartite graph contains a perfect matching.* The proof we have seen uses Hall's theorem. In this exercise, we want to find another proof using our knowledge about flows.

Let $G = (A \dot{\cup} B, E)$ be a k-regular graph for $k \geq 1$.

(a) Describe how to model bipartite matching on G as a flow problem in a network $N = (V, A, c, s, t)$.

(b) Construct explicitly a (not necessarily integer!) flow f on N with $\text{val}(f) = n$. Conclude that $\text{maxflow}(N) = n$.
Hint: you don't have to construct an integral flow. Make sure to define the flow value on all edges of your network N!

(c) Using the result from (b), show that there exists an integer valued flow f' with $\text{val}(f') = n$. Using f', prove that G contains a perfect matching M.



Each vertex in A has k outgoing edges: $\sum f_{\text{out}} = k \cdot (1/k) = 1.$
Each vertex in B has k incoming edges: $\sum f_{\text{in}} = k \cdot (1/k) = 1.$

(b) For our flow f, we set $f(e) = 1/k$ for all edges between A and B. For the remaining edges, we set the flow to 1. Because all capacities are 1, they are satisfied. Flow conservation is also satisfied: each vertex in A has one incoming edge with flow value 1 and k outgoing edges with flow value $1/k$ each. For vertices in B it is exactly the other way around. Here we used that G is k-regular. The value of this flow equals $|A| = |B|$. The cut $cap(\{s\}, U \cup W \cup \{t\})$ has capacity $|A|$. Thus, f is a maximal flow.

Exercise S12.1 – *Integrality and Matching*

We have already seen Frobenius' Theorem: *Let $k \geq 1$. Every k-regular bipartite graph contains a perfect matching.* The proof we have seen uses Hall's theorem. In this exercise, we want to find another proof using our knowledge about flows.

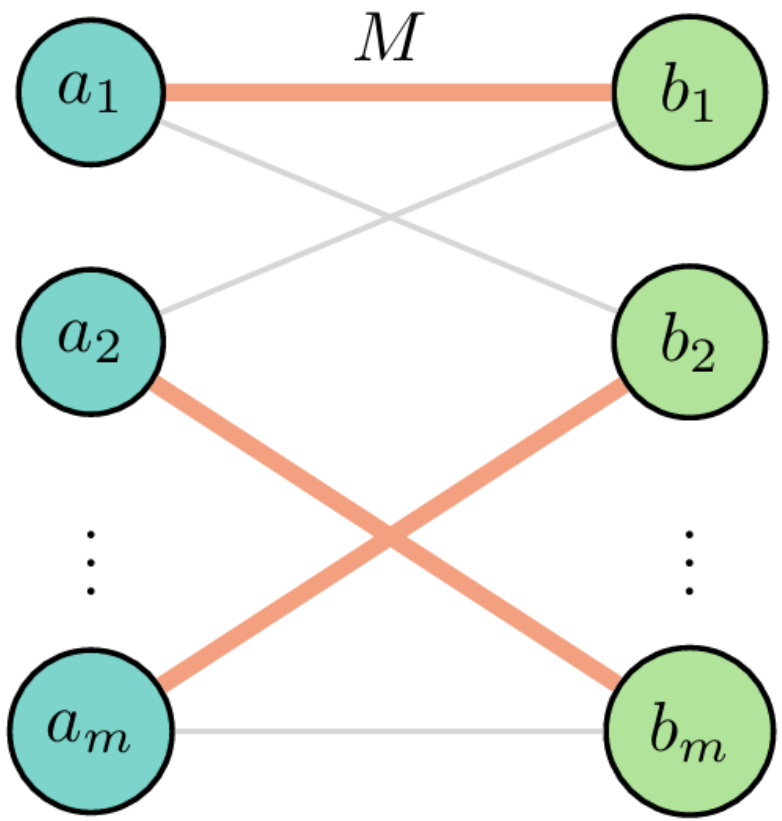
Let $G = (A \dot{\cup} B, E)$ be a k-regular graph for $k \geq 1$.

(a) Describe how to model bipartite matching on G as a flow problem in a network $N = (V, A, c, s, t)$.

(b) Construct explicitly a (not necessarily integer!) flow f on N with $\text{val}(f) = n$. Conclude that $\text{maxflow}(N) = n$.
Hint: you don't have to construct an integral flow. Make sure to define the flow value on all edges of your network N!

(c) Using the result from (b), show that there exists an integer valued flow f' with $\text{val}(f') = n$. Using f', prove that G contains a perfect matching M.

(c) By Theorem 3.12, the maximum flow in N and the maximum integral flow in N have the same value. Thus, by (b), there is an integral flow of value $|A|$. By Lemma 3.15, G has a matching of size $|A| = |B|$. This is already a perfect matching.



Algorithms – Highlights

Algorithms – Highlights

Previously:

Algorithms – Highlights

Previously:

Probability Theory

Algorithms – Highlights

Previously:

Probability Theory

Randomised (non-graph) Algorithms

Algorithms – Highlights

Algorithms – Highlights

Randomised

Deterministic

Algorithms – Highlights

Randomised

Deterministic

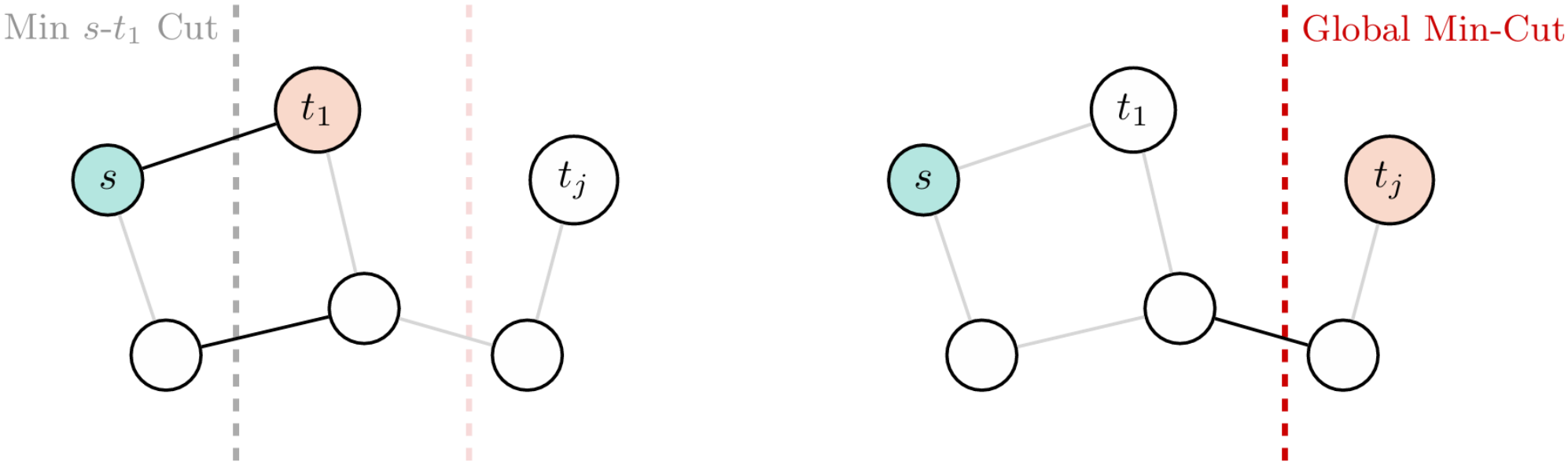
Min-Cut

Min-Cut

8. The MIN-CUT problem can be solved by fixing one vertex s and computing minimum s-t cuts for all $t \in V \setminus \{s\}$.

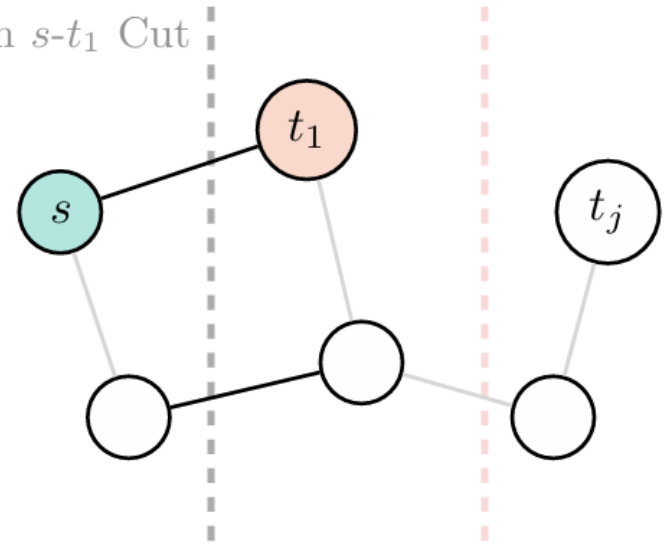
Answer: TRUE

Explanation. [Script page 192] Inelegant illustration:

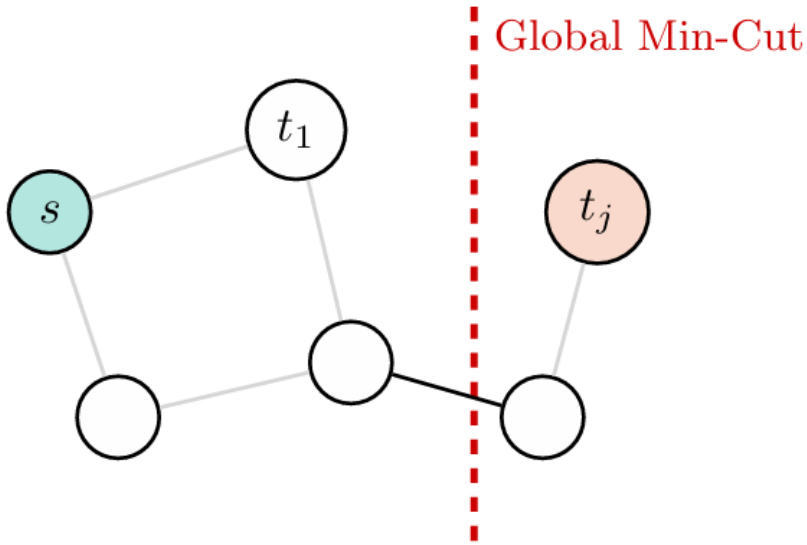


Min-Cut

Explanation. [Script page 192] Inelegant illustration:



Iteration 1: Trying with t_1
s, t_1 are on **different** sides of cut.
Cut size $= 2$
(we first need to try with all
other t's to see if this is $\mu(\mathbf{G})$)



Iteration j: Trying with t_j
s, t_1 are on **different** sides of cut.
Cut size $= 1 = \mu(\mathbf{G})$

Min-Cut

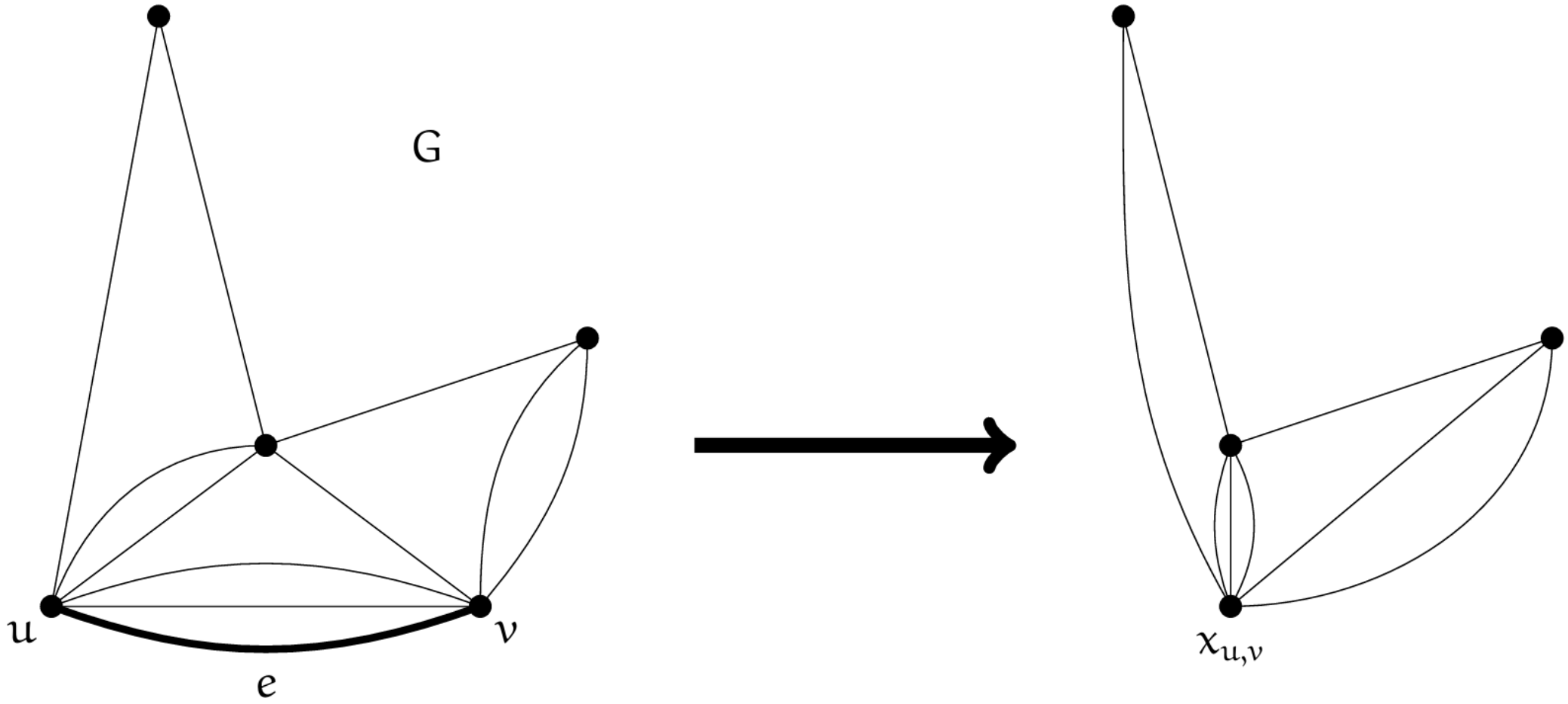
Runtime of finding maximum flow in $O(nm \log n) = O(n^3 \log n)$ using Dynamic Trees n-1 times:

$$O(n^4 \log n)$$

→ **Use randomised algorithm to be faster than this** 🎉 🎉 🎉

Min-Cut

Edge contraction:

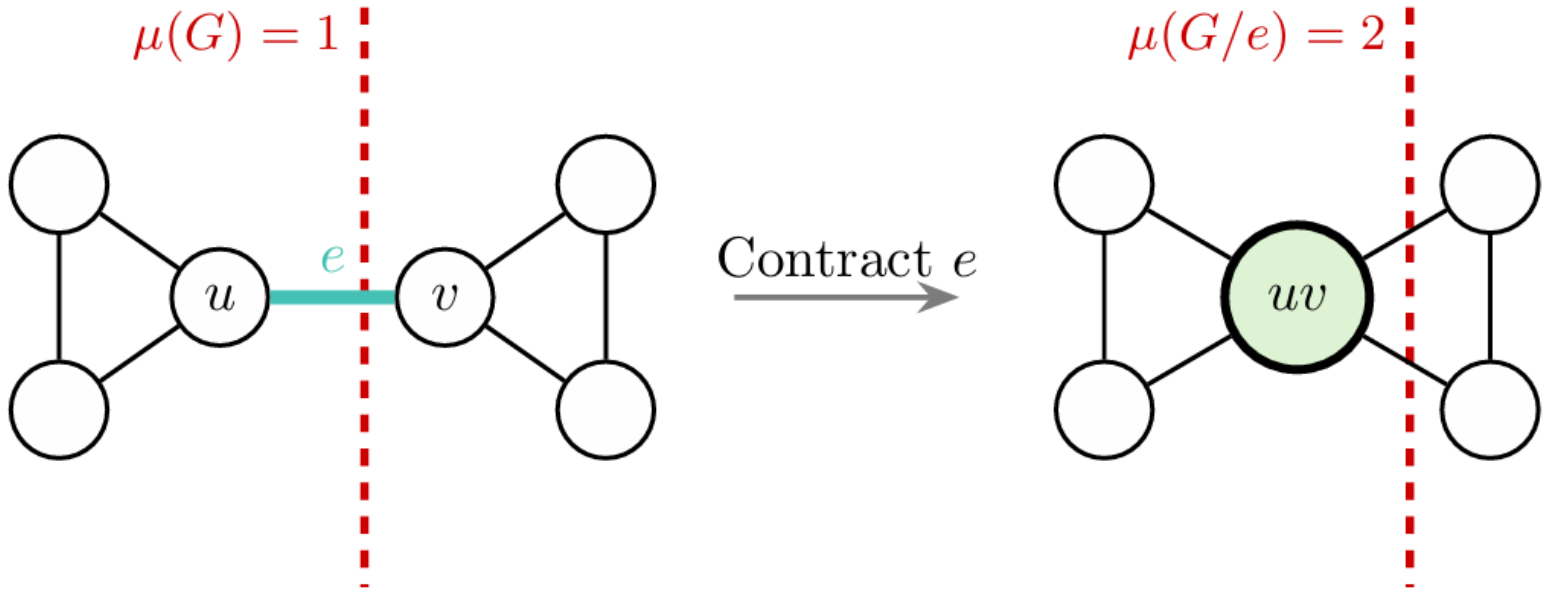


Min-Cut

9. If e is an edge of G, then always $\mu(G/e) \leq \mu(G)$.

Answer: FALSE

Explanation. Here $\mu(G)$ denotes the size of the minimum cut (edge connectivity) of the graph. **Counterexample:**



Original Graph G **Contracted Graph G/e**

Min-Cut

Lemma 3.20. Sei G ein Graph und e eine Kante in G. Dann gilt $\mu(G/e) \geq \mu(G)$ und falls es in G einen minimalen Schnitt C mit $e \notin C$ gibt, dann gilt $\mu(G/e) = \mu(G)$.

Min-Cut

CUT(G) — G zusammenhängender Multigraph

1: **while** $|V(G)| > 2$ **do**
2: $\quad e \leftarrow$ gleichverteilt zufällige Kante in G
3: $\quad G \leftarrow G/e$
4: **return** Grösse des eindeutigen Schnitts in G

Min-Cut

Lemma 3.21. Sei $G = (V, E)$ ein Multigraph mit n Knoten. Falls e gleichverteilt zufällig unter den Kanten in G gewählt wird, dann gilt

$$\Pr[\mu(G) = \mu(G/e)] \geq 1 - \frac{2}{n}.$$

Beweis. Sei C ein minimaler Schnitt in G und sei $k := |C| = \mu(G)$. Sicherlich ist der Grad jedes Knotens in G mindestens k, da die zu einem Knoten inzidenten Kanten immer einen Schnitt bilden. Es gilt daher $|E| = \frac{1}{2}\sum_{v \in V} \deg(v) \geq \frac{kn}{2}$. Wir erinnern uns an $e \notin C \Rightarrow \mu(G/e) = \mu(G)$ und somit

$$\Pr[\mu(G) = \mu(G/e)] \geq \Pr[e \notin C] = 1 - \frac{|C|}{|E|} \geq 1 - \frac{k}{kn/2} = 1 - \frac{2}{n},$$

was zu zeigen war. □

Min-Cut

$\hat{p}(G) :=$ Wahrscheinlichkeit, dass $\textsc{Cut}(G)$ den Wert $\mu(G)$ ausgibt und

$$\hat{p}(n) := \inf_{G=(V,E),\,|V|=n} \hat{p}(G) \ .$$

Man beachte, dass $\hat{p}(2) = 1$.

Lemma 3.22. Es gilt für alle $n \geq 3$

$$\hat{p}(n) \geq (1 - 2/n) \cdot \hat{p}(n-1).$$

Min-Cut

$$\hat{p}(n) \geq \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \cdots \frac{3}{5} \cdot \frac{2}{4} \cdot \frac{1}{3} \cdot \underbrace{\hat{p}(2)}_{=1} = \frac{2}{n(n-1)}$$

Lemma 3.23. Es gilt $\hat{p}(n) \geq \frac{2}{n(n-1)} = 1/\binom{n}{2}$ für alle $n \geq 2$.

Min-Cut

CUT(G) — G zusammenhängender Multigraph

1: **while** $|V(G)| > 2$ **do**
2: $\quad e \leftarrow$ gleichverteilt zufällige Kante in G
3: $\quad G \leftarrow G/e$
4: **return** Grösse des eindeutigen Schnitts in G

Min-Cut

Satz 3.24. Für den Algorithmus der $\lambda\binom{n}{2}$-maligen Wiederholung von $\textsc{Cut}(G)$ gilt:

(1) Der Algorithmus hat eine Laufzeit von $O(\lambda n^4)$.

(2) Der kleinste angetroffene Wert ist mit einer Wahrscheinlichkeit von mindestens $1 - e^{-\lambda}$ gleich $\mu(G)$.

Min-Cut

Runtime of finding maximum flow in $O(nm \log n) = O(n^3 \log n)$ using Dynamic Trees n-1 times:

$$O(n^4 \log n)$$

→ **Use randomised algorithm to be faster than this** 🎉 🎉 🎉

Bootstrapping

$z(t) = O(t^4 \log t)$ Zeit mit $p^*(t) = 1$, oder nach Satz 3.24 (mit $\lambda = 1$) in $z(t) = O(t^4)$ Zeit mit $p^*(t) = 1 - e^{-1}$).

CUT1(G) — G zusammenhängender Multigraph

1: **while** $|V(G)| > t$ **do**
2: $e \leftarrow$ gleichverteilt zufällige Kante in G
3: $G \leftarrow G/e$
4: **return** Grösse des eindeutigen Schnitts in G ▷ in Zeit $O(z(t))$

Min-Cut

Bootstrapping

$z(t) = O(t^4 \log t)$ Zeit mit $p^*(t) = 1$, oder nach Satz 3.24 (mit $\lambda = 1$) in $z(t) = O(t^4)$ Zeit mit $p^*(t) = 1 - e^{-1}$).

CUT1(G) — G zusammenhängender Multigraph

1: **while** $|V(G)| > t$ **do**
2: $e \leftarrow$ gleichverteilt zufällige Kante in G
3: $G \leftarrow G/e$
4: **return** Grösse des eindeutigen Schnitts in G ▷ in Zeit $O(z(t))$

Min-Cut

Now we repeat this process, and eventually:

> Wo führt das hin? Es konvergiert zu einem Verfahren mit einer Laufzeit von $O(n^2 \text{poly}(\log n))$, man kann sich das wie einen „Grenzwertalgorithmus“ vorstellen.

(Details in lecture/script on page 197)

We achieve **$O(n^2 \text{poly}(\log n))$** runtime 🎉 🎉 🎉

Key: Cut(G) always looses the most success probability in the last few steps and by iterating over these separately (bootstrapping), we don’t have to redo the whole process at every failure in the last steps.

Smallest Enclosing Circle