

Algorithmen und Wahrscheinlichkeit

Woche 11

Ilya Maier

Theorie Recap

Konvexe Hülle

Gegeben: eine endliche Punktenmenge $P \subseteq \mathbb{R}^d$

Gesucht: die konvexe Hülle von P , $\text{conv}(P)$

Konvexe Hülle: die minimale konvexe Menge, die P beinhaltet

Randkante: geordnetes Paar qr für $q, r \in P$ mit $q \neq r$ s.d. alle Punkte von P links von qr liegen

JarvisWrap(P)

- 1) $h \leftarrow 0$
- 2) $p_{\text{now}} \leftarrow$ Punkt mit kleinster x -Koordinate
- 3) **repeat**
- 4) $q_h \leftarrow p_{\text{now}}$
- 5) $p_{\text{now}} \leftarrow \text{FindNext}(q_h)$
- 6) $h \leftarrow h + 1$
- 7) **until** $p_{\text{now}} = p_0$
- 8) **return** (q_0, \dots, q_{h-1})

Laufzeit: $\mathcal{O}(n \cdot h)$, wobei $h := \#\text{Ecken in } \text{conv}(P)$

→ $\mathcal{O}(n^2)$ worst case

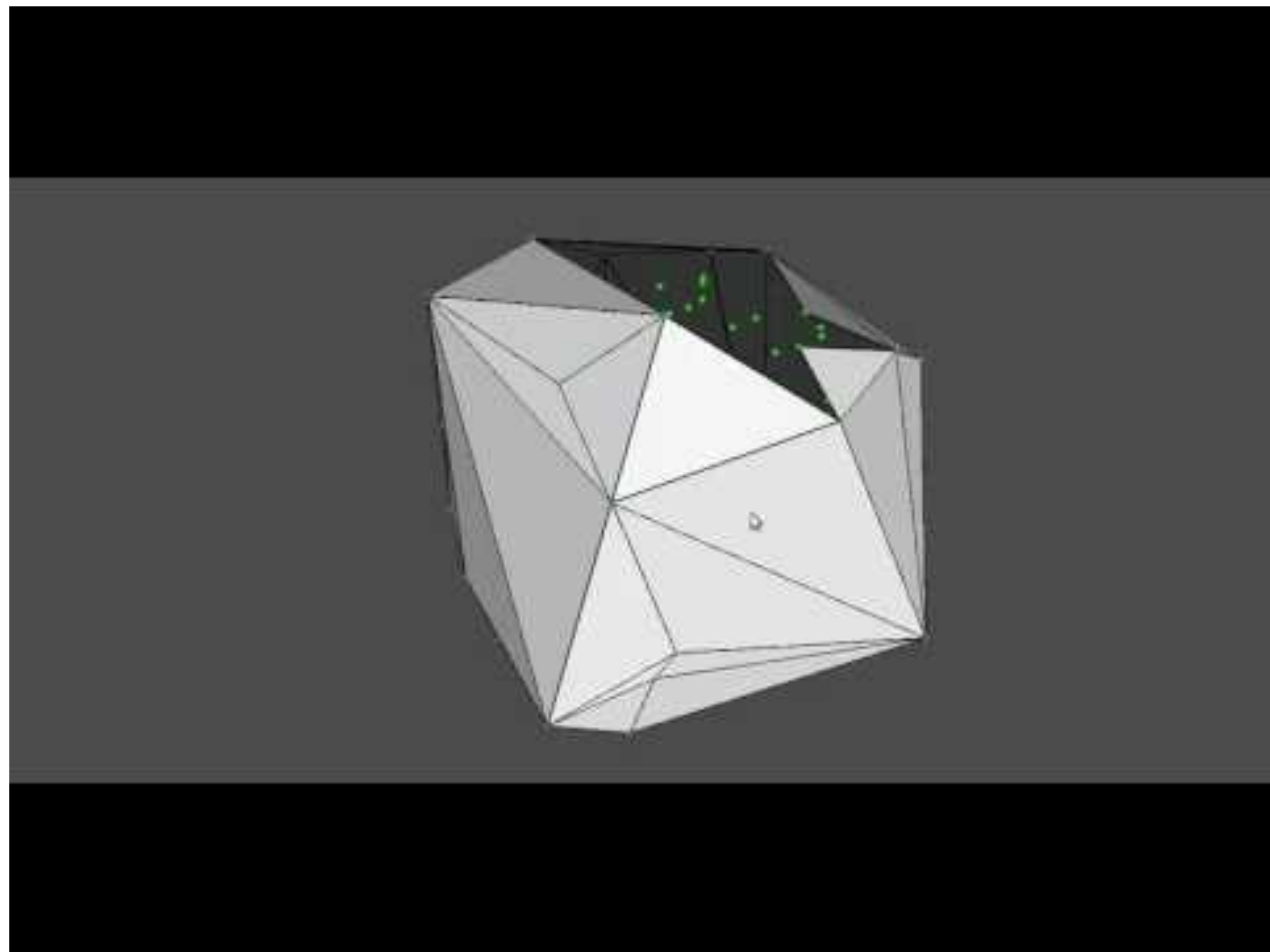
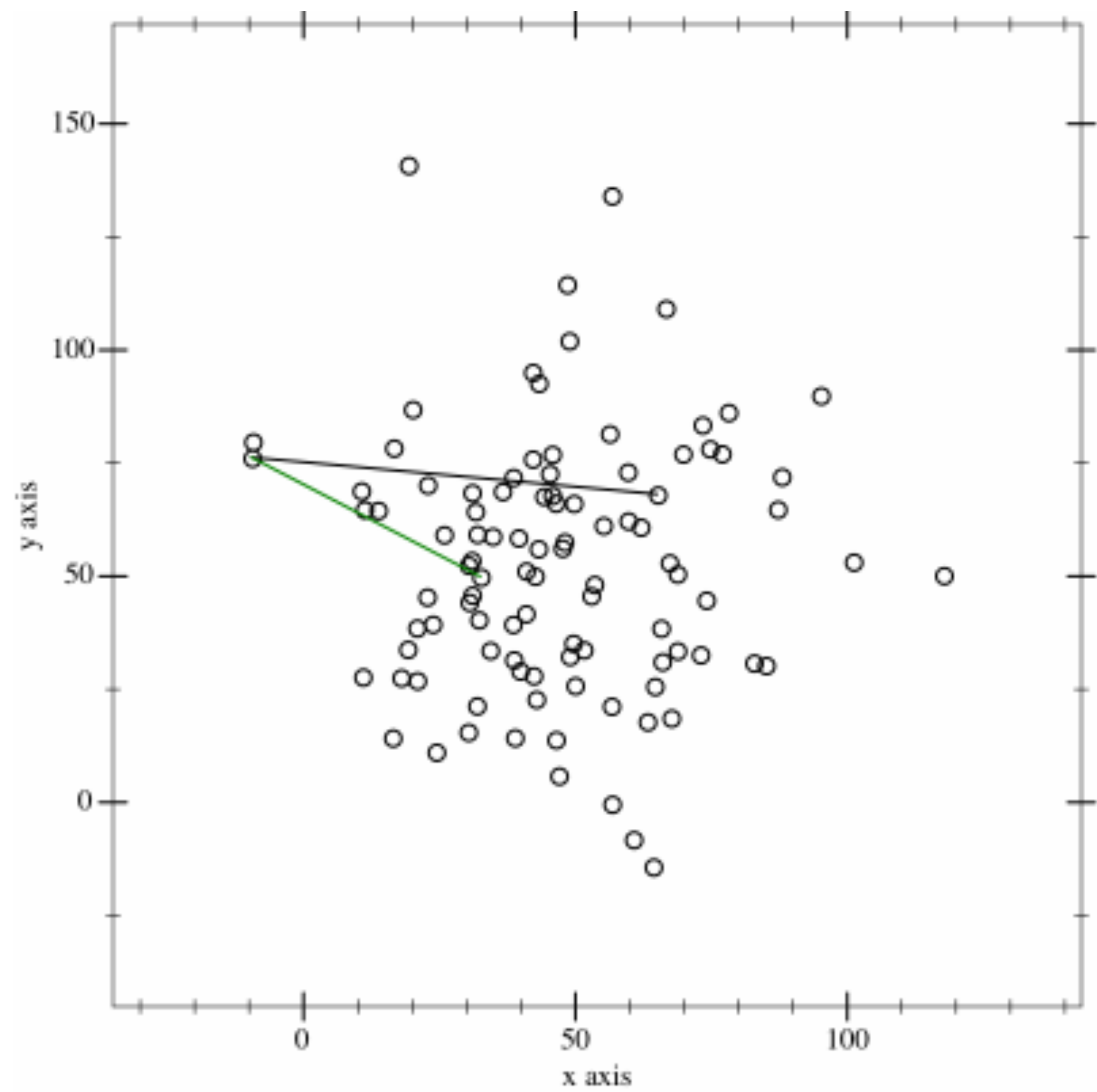
→ $\mathcal{O}(n)$ wenn h konstant

FindNext(q)

- 1) Wähle beliebig $p_0 \in P \setminus \{q\}$
- 2) $q_{\text{next}} \leftarrow p_0$
- 3) **for all** $p \in P \setminus \{q, p_0\}$ **do**
- 4) **if** p **rechts** von qq_{next} **then**
- 5) $q_{\text{next}} \leftarrow p$
- 6) **return** q_{next}

p rechts von qq_{next}

$$\iff (q_x - p_x)(q_{\text{next},y} - p_y) < (q_y - p_y)(q_{\text{next},x} - p_x)$$



Konvexe Hülle - Lower Bound der Laufzeit

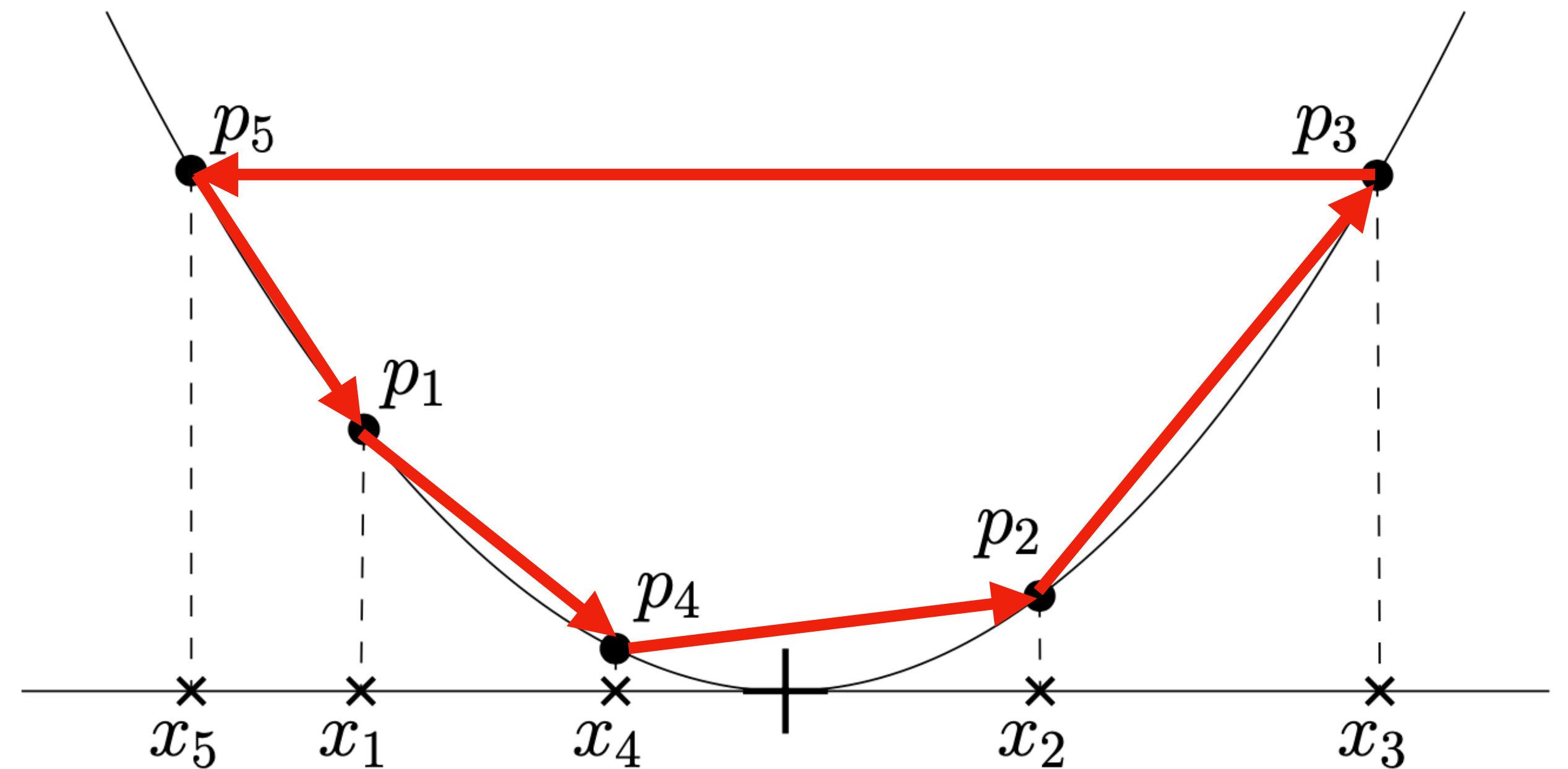
Reduktion vom Sortieren:

$$(x_1, \dots, x_n) \rightarrow ((x_1, x_1^2), \dots, (x_n, x_n^2))$$

können wir Konvexe Hülle in $t(n)$ bestimmen

so können wir in $\mathcal{O}(t(n) + n)$ sortieren

$$\Rightarrow t(n) \in \Omega(n \cdot \log n)$$



Konvexe Hülle - Lokal Verbessern

LocalRepair(p_1, p_2, \dots, p_n) (p_1, p_2, \dots, p_n) sortiert

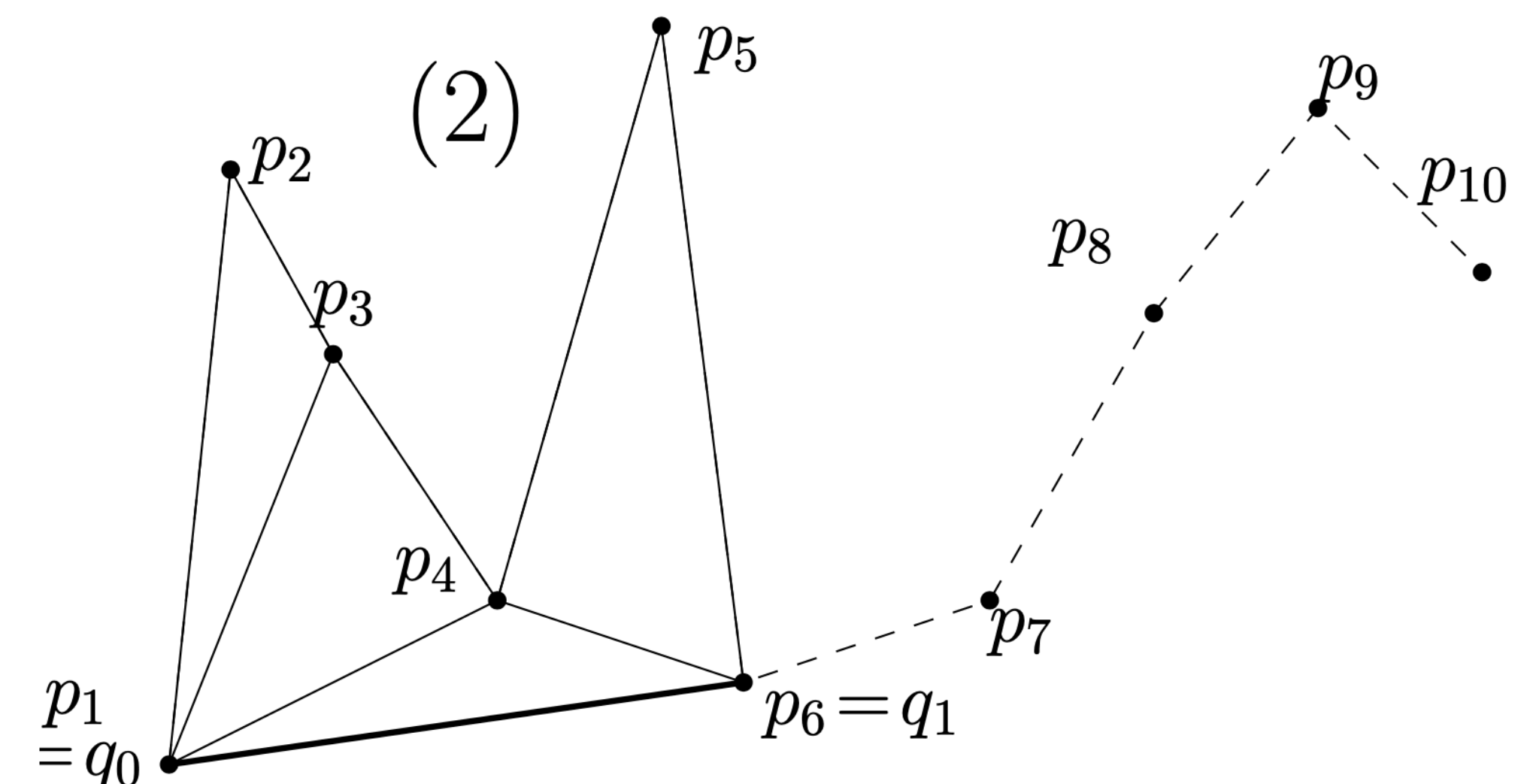
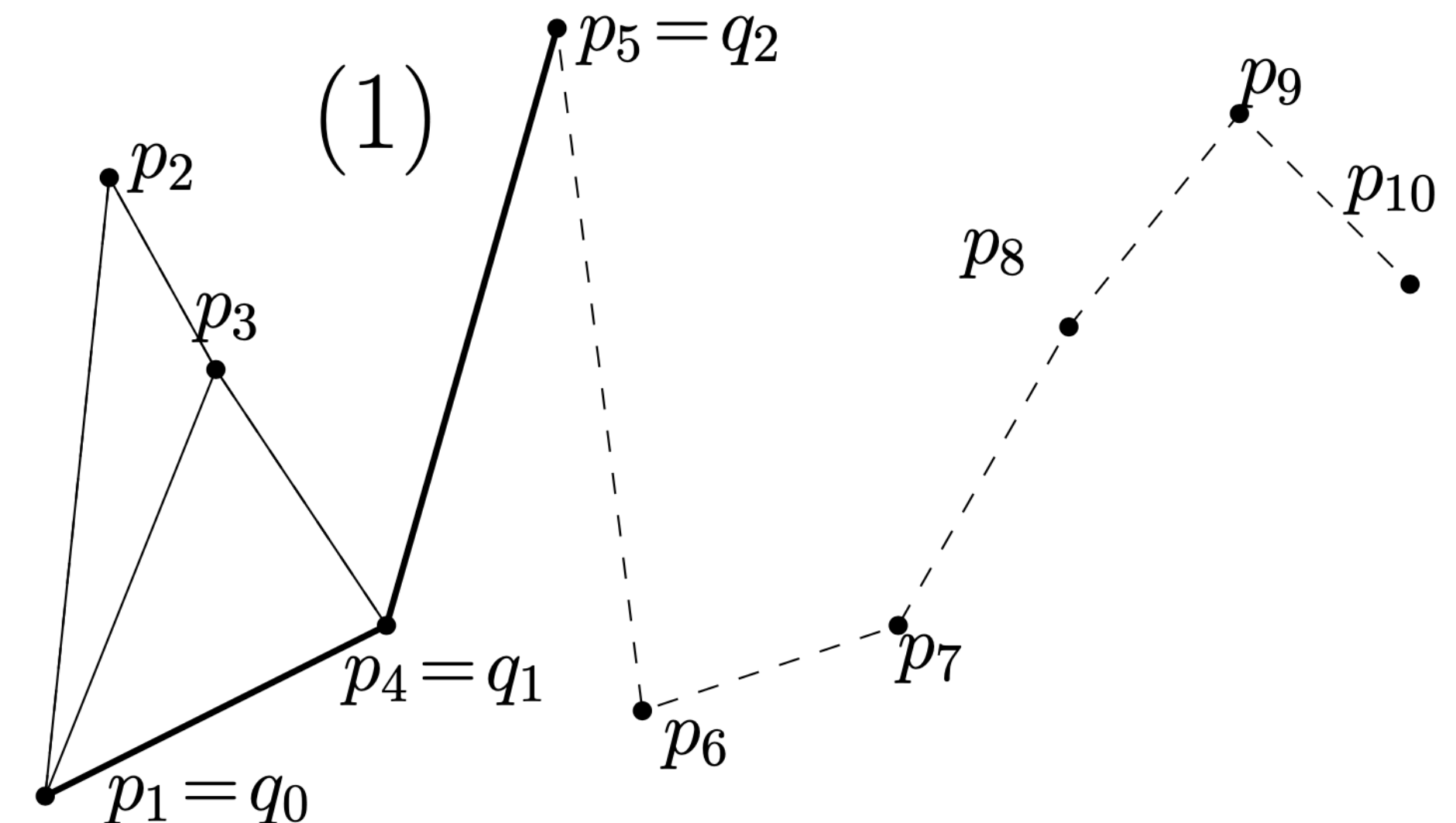
```

1:  $q_0 \leftarrow p_1; h \leftarrow 0$ 
2: for  $i \leftarrow 2$  to  $n$  do                     $\triangleright$  unterer Rand, links nach rechts
3:   while  $h > 0$  und  $q_h$  links von  $q_{h-1}p_i$  do
4:      $h \leftarrow h - 1$ 
5:    $h \leftarrow h + 1; q_h \leftarrow p_i$ 
6:    $\triangleright (q_0, \dots, q_h)$  untere konvexe Hülle von  $\{p_1, \dots, p_i\}$ 
7:  $h' \leftarrow h$ 
8: for  $i \leftarrow n - 1$  downto  $1$  do     $\triangleright$  oberer Rand, rechts nach links
9:   while  $h > h'$  und  $q_h$  links von  $q_{h-1}p_i$  do
10:     $h \leftarrow h - 1$ 
11:    $h \leftarrow h + 1; q_h \leftarrow p_i$ 
12: return  $(q_0, q_1, \dots, q_{h-1})$ 

```

Analyse

- Sortieren $\rightarrow \mathcal{O}(n \log n)$
 - $2n - 2 - h$ Mal erfolgreiche Tests (neues Dreieck) $\rightarrow \mathcal{O}(n)$
 - $2n - 2$ erfolglose Tests (p_i wird zu einer Kante) $\rightarrow \mathcal{O}(n)$
- \rightarrow Wenn die Punkte schon sortiert sind $\rightarrow \mathcal{O}(n)$ statt $\mathcal{O}(n \log n)$



Netzwerke und Flüsse

Netzwerk $N := (V, A, c, s, t)$

- (V, A) ist ein gerichteter Graph
- $s \in V$ ist die Quelle
- $t \in V$ ist die Senke
- $c : A \rightarrow \mathbb{R}_0^+$ die Kapazitätsfunktion

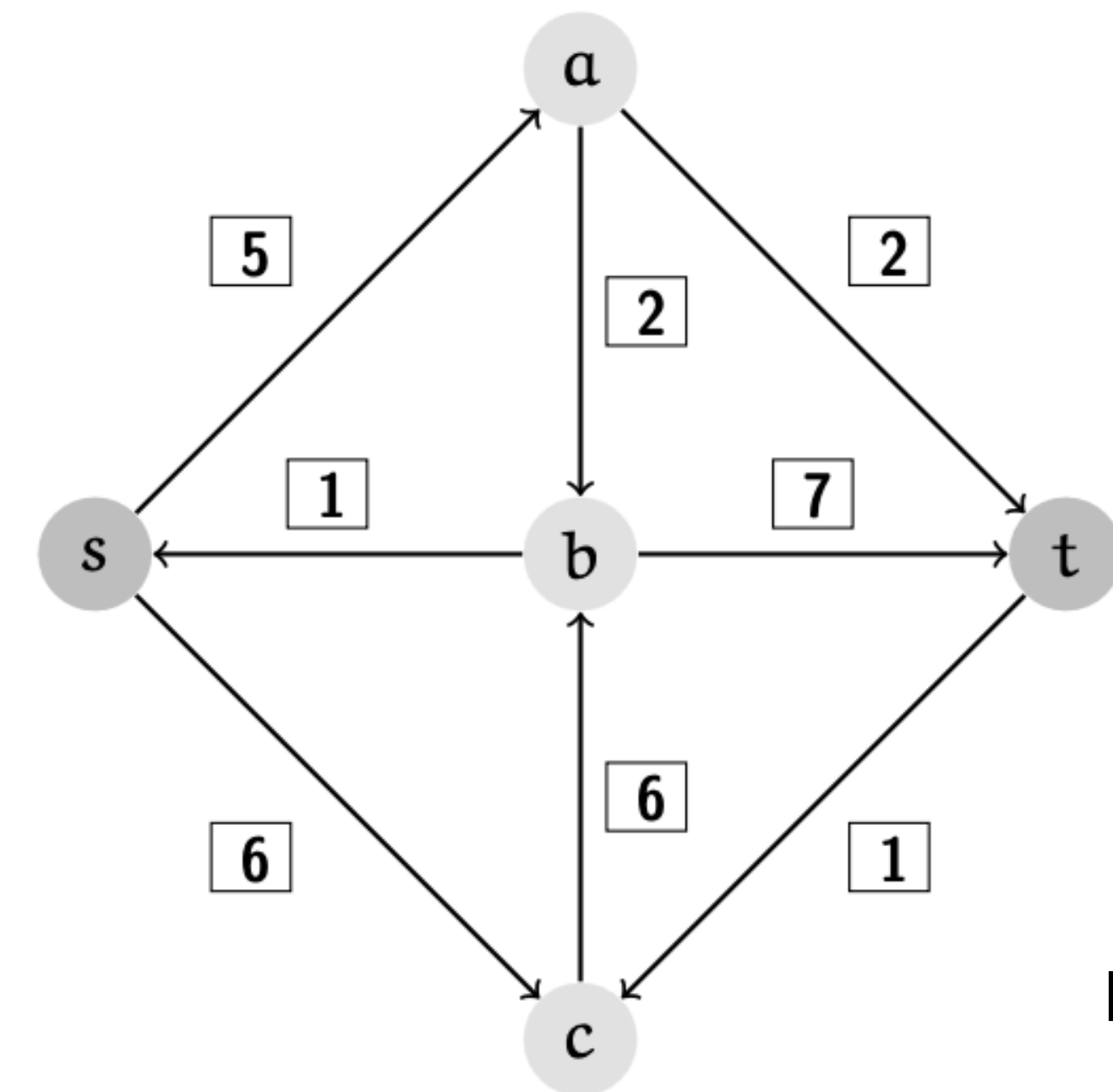
Fluss f in $N : f : A \rightarrow \mathbb{R}_0^+$

- **Zulässigkeit:** $0 \leq f(e) \leq c(e)$ für alle $e \in A$
- **Flusserhaltung:** Für alle $v \in V \setminus \{s, t\}$ gilt

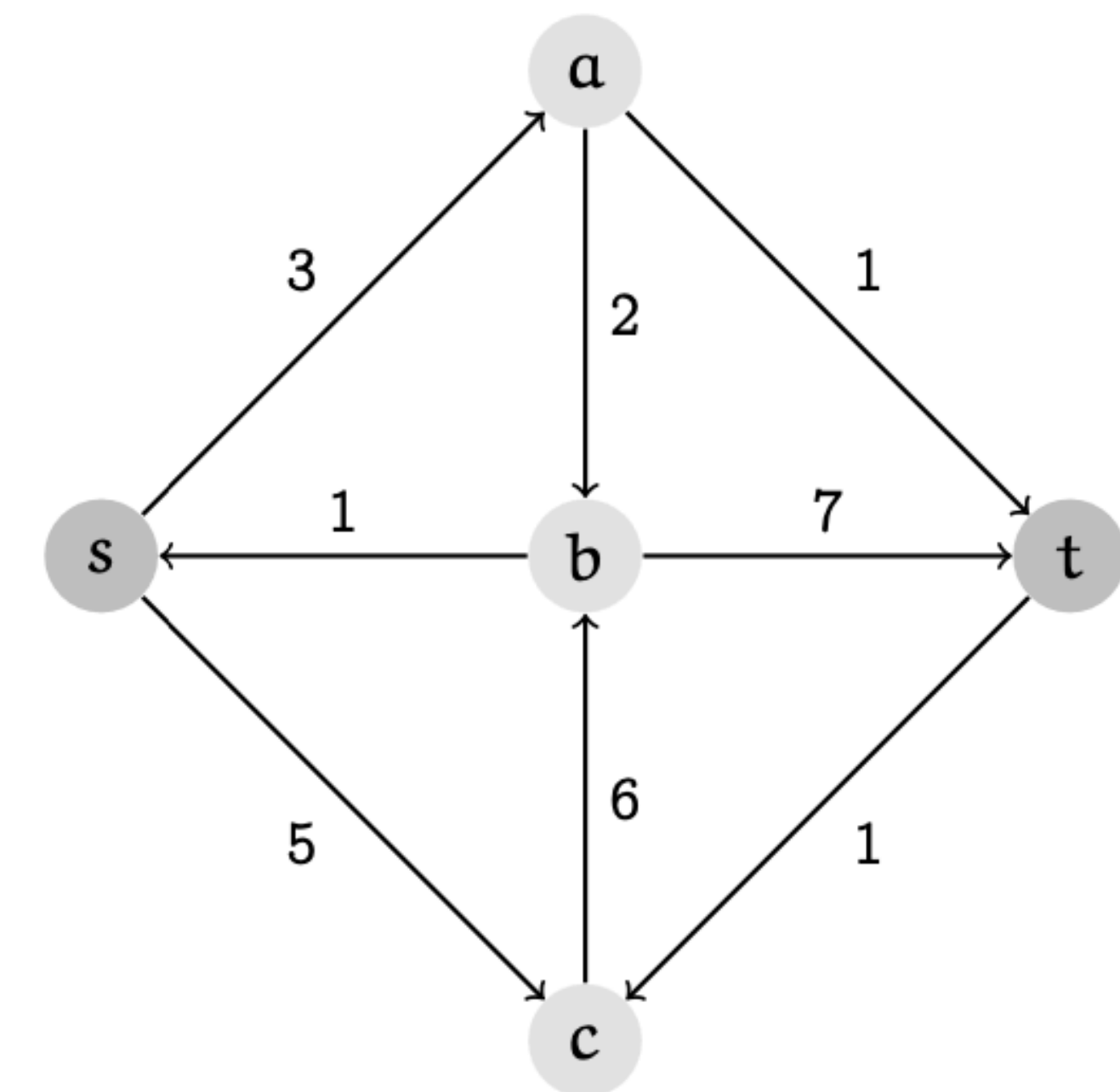
$$\sum_{u \in V: (u,v) \in A} f(u, v) = \sum_{u \in V: (v,u) \in A} f(v, u)$$

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

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



Netzwerk



Fluss

Schnitte

s-t-Schnitt in $N := (V, A, c, s, t)$

- eine Partition von V : (S, T)

- $s \in S, t \in T$

- $\text{cap}(S, T) = \sum_{(u,w) \in (S \times T) \cap A} c(u, w)$

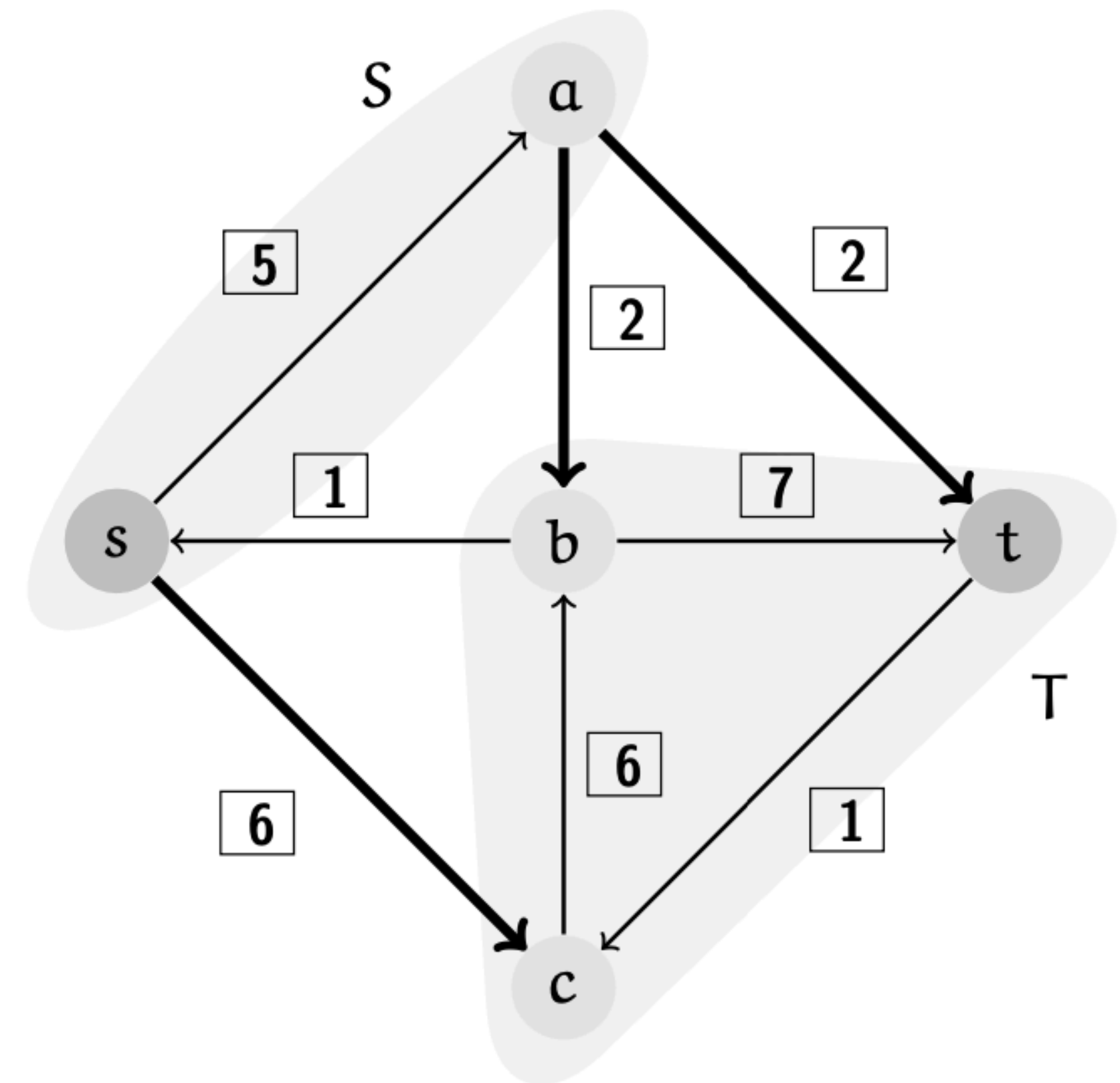
Flüsse und Schnitte

- für einen Fluss f und einen Schnitt (S, T) : $\text{val}(f) \leq \text{cap}(S, T)$

- für einen Fluss f : $\exists (S, T) : \text{val}(f) = \text{cap}(S, T) \implies f$ is maximal

- **MaxFlowMinCut-Theorem:**

$$\max_f \text{val}(f) = \min_{(S,T)} \text{cap}(S, T)$$



$$\text{cap}(S, T) = 2 + 2 + 6 = 10$$

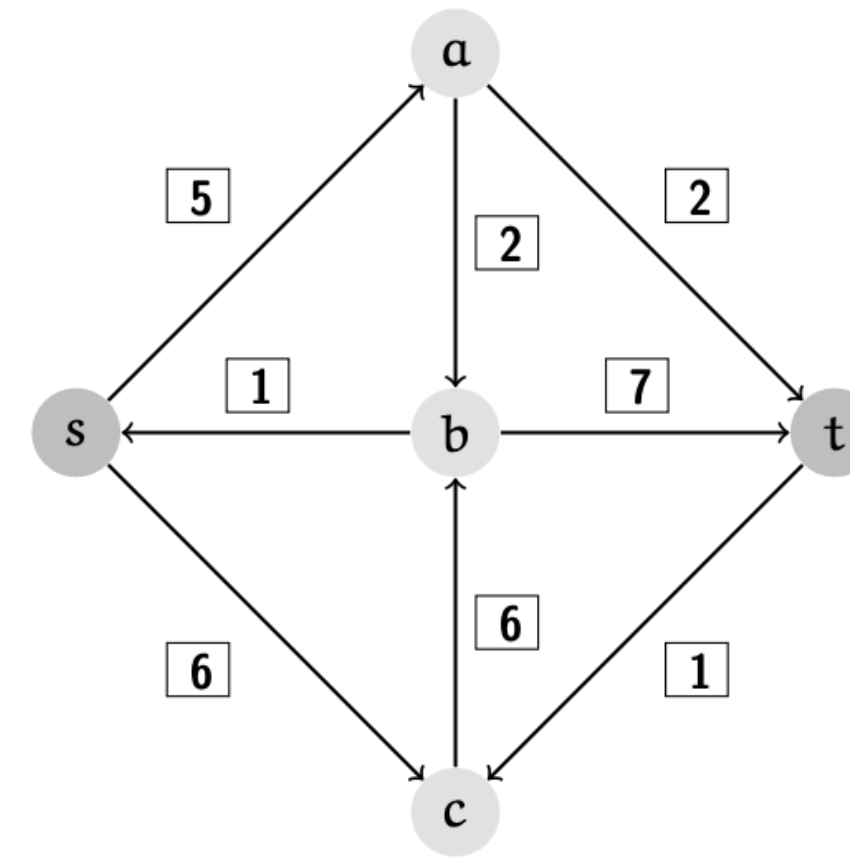
Restnetzwerk

Restnetzwerk $N_f = (V, A_f, r_f, s, t)$ für N, f

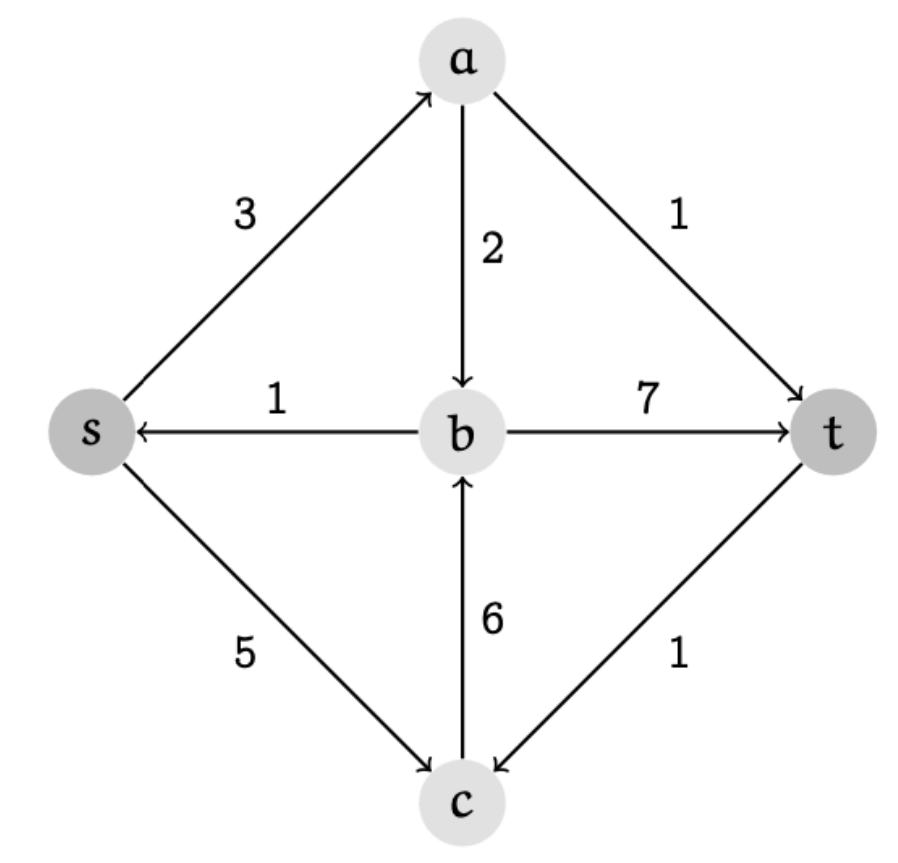
- N hat keine entgegen gerichtete Kanten
- $\forall e \in A : f(e) < c(e) \implies e \in A_f \wedge r_f(e) = c(e) - f(e)$
- $\forall e \in A : f(e) > 0 \implies e^{opp} \in A_f \wedge r_f(e^{opp}) = f(e)$
- A_f hat keine weiteren Kanten

Restnetzwerk und MaxFlow

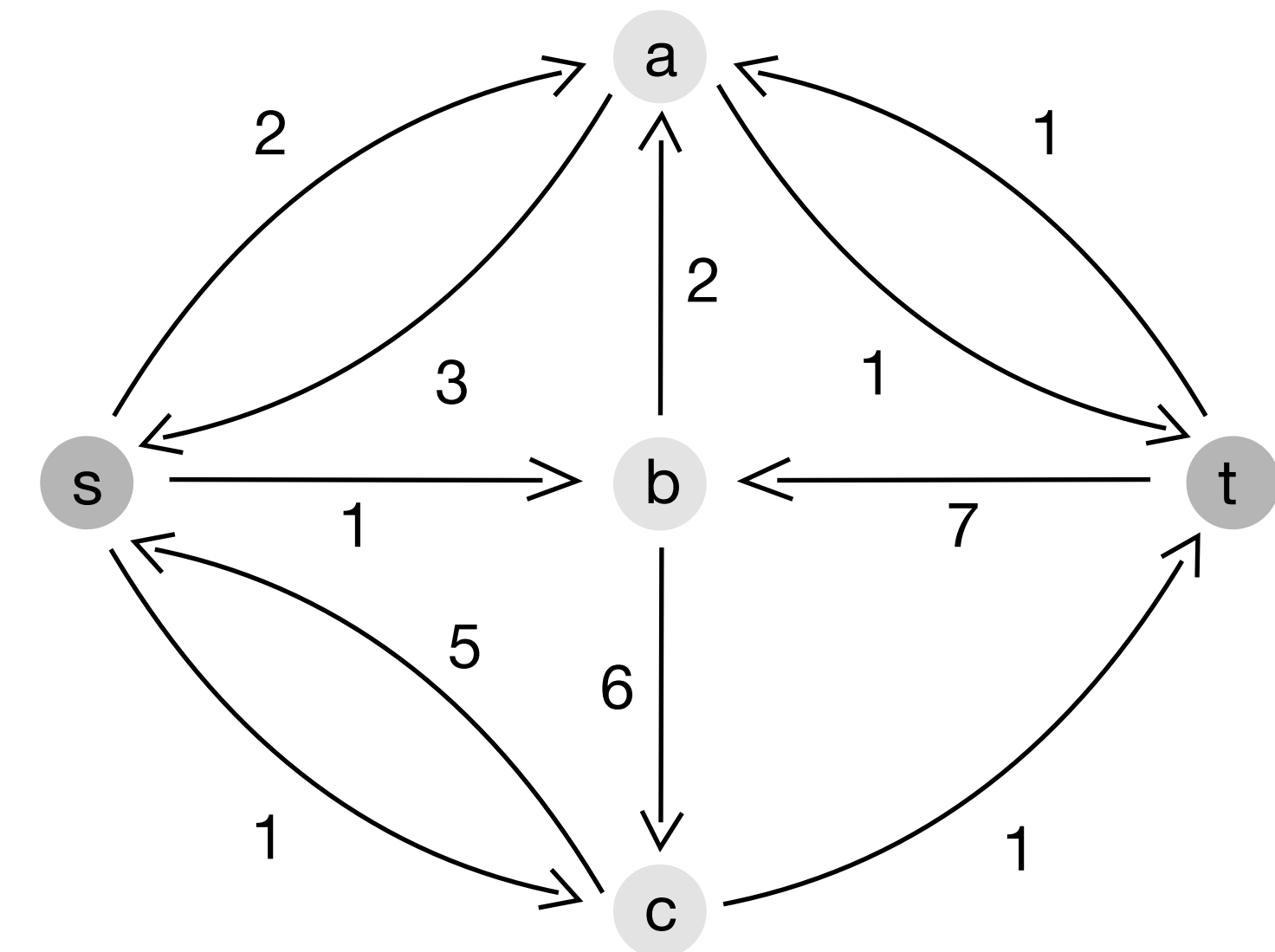
f ist ein maximaler Fluss $\iff \neg \exists$ gerichteter s-t-Pfad in N_f



Netzwerk



Fluss



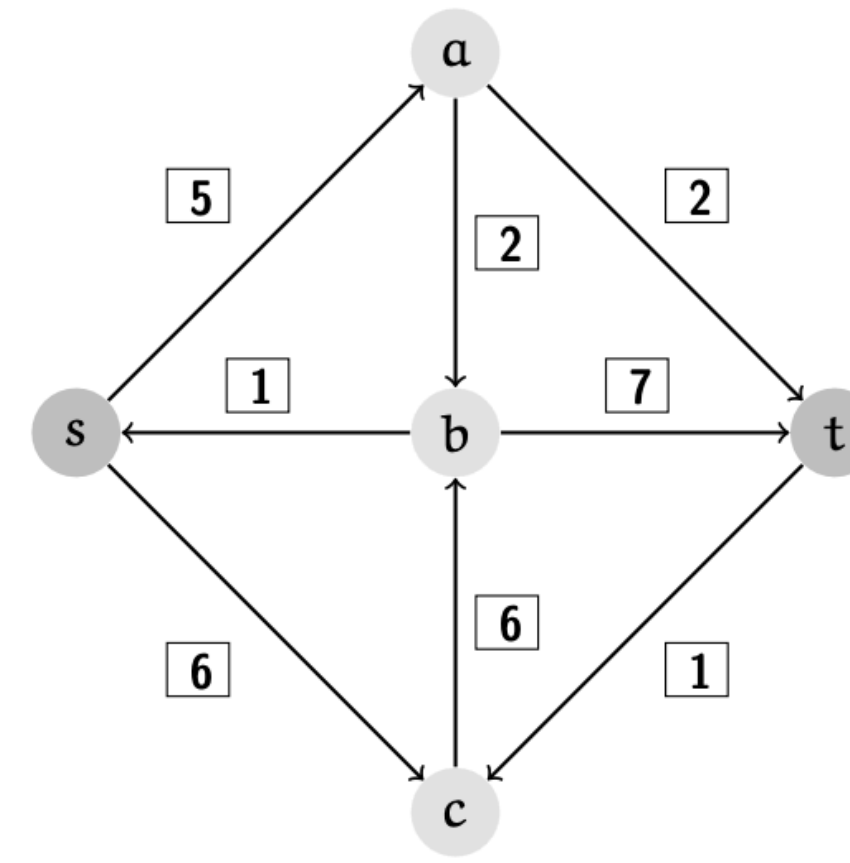
Restnetzwerk

Restnetzwerk $N_f = (V, A_f, r_f, s, t)$ für N, f

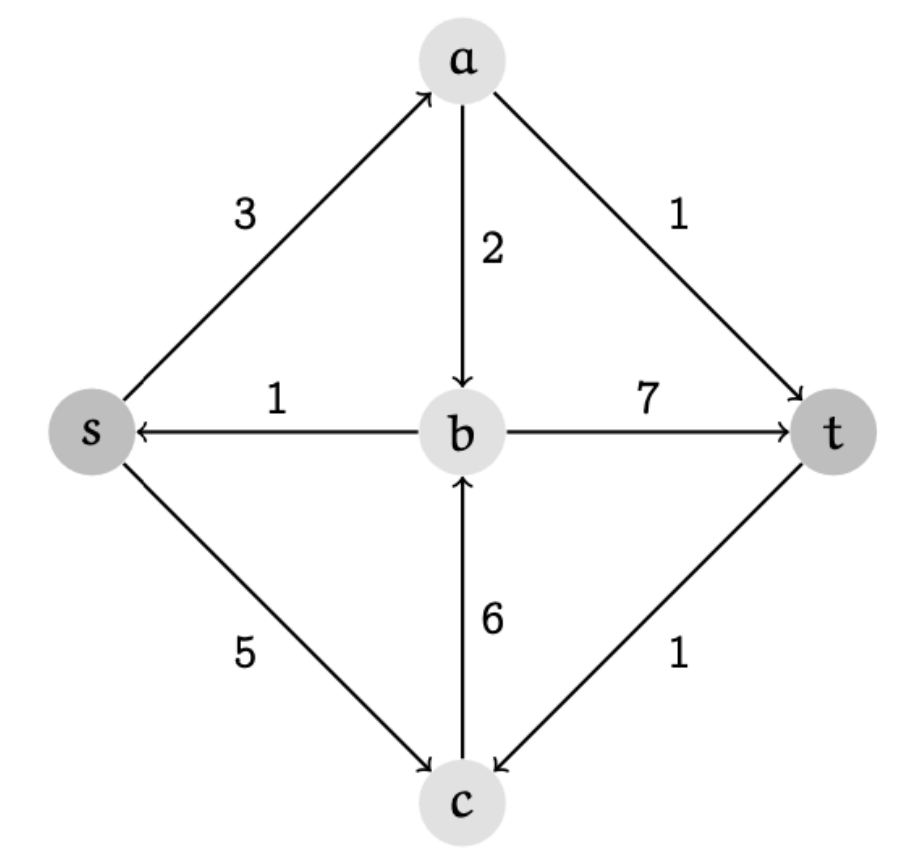
- N hat keine entgegen gerichtete Kanten
- $\forall e \in A : f(e) < c(e) \implies e \in A_f \wedge r_f(e) = c(e) - f(e)$
- $\forall e \in A : f(e) > 0 \implies e^{opp} \in A_f \wedge r_f(e^{opp}) = f(e)$
- A_f hat keine weiteren Kanten

Restnetzwerk und MaxFlow

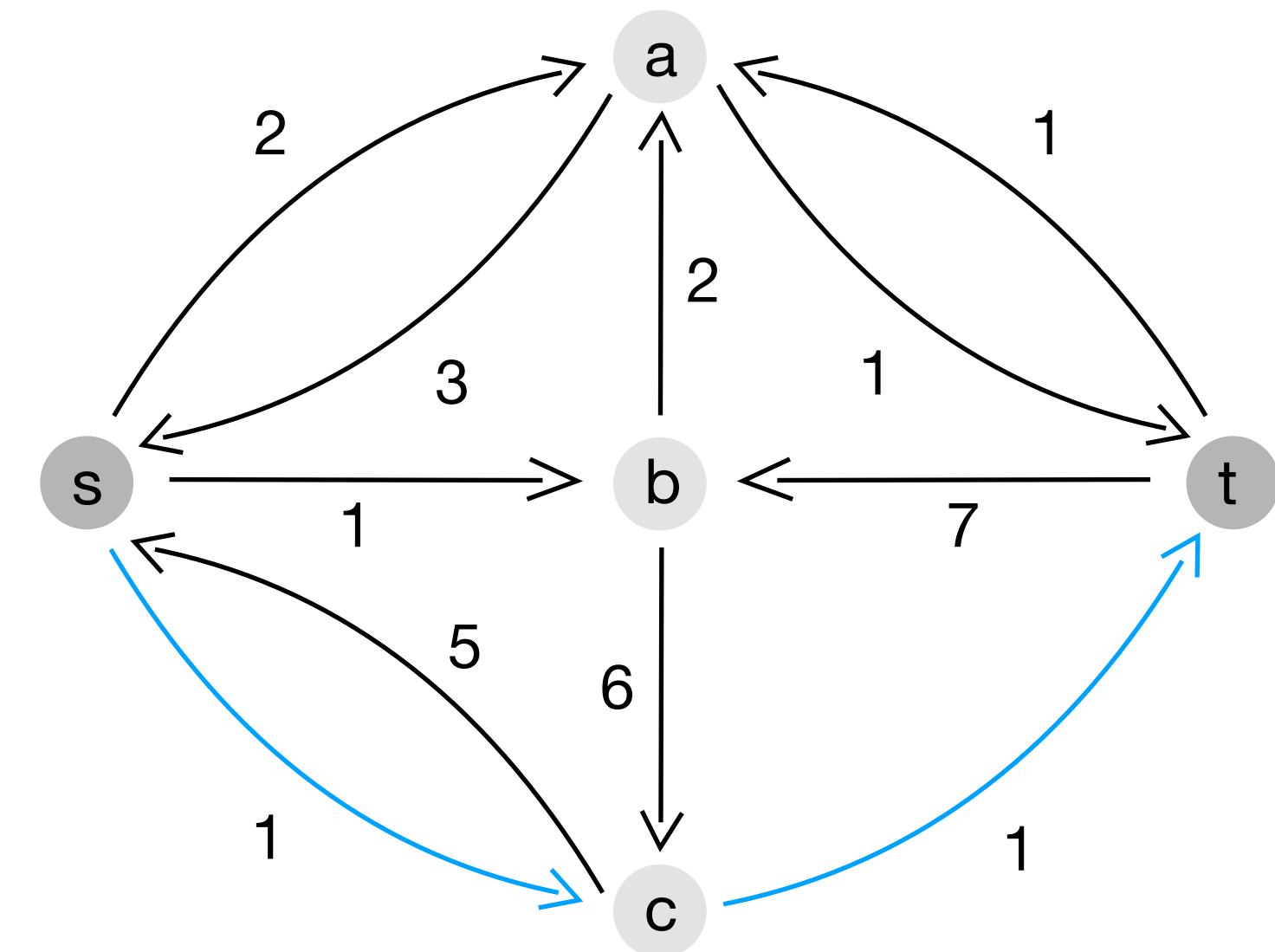
f ist ein maximaler Fluss $\iff \neg \exists$ gerichteter s-t-Pfad in N_f



Netzwerk



Fluss



Jahr	Autor(en)	Name	Laufzeiten
1956	Ford, Fulkerson	Algorithmus von Ford und Fulkerson	$\mathcal{O}(m \cdot n \cdot u_{\max})$, falls alle Kapazitäten ganzzahlig sind
1969	Edmonds, Karp	Algorithmus von Edmonds und Karp	$\mathcal{O}(m^2 \cdot n)$
1970	Dinic	Algorithmus von Dinic	$\mathcal{O}(m \cdot n^2)$
1973	Dinic, Gabow		$\mathcal{O}(m \cdot n \cdot \log(u_{\max}))$
1974	Karzanov		$\mathcal{O}(n^3)$
1977	Cherkassky		$\mathcal{O}(n^2 \cdot \sqrt{m})$
1980	Galil, Naamad		$\mathcal{O}(m \cdot n \cdot \log(n)^2)$
1983	Sleator, Tarjan		$\mathcal{O}(m \cdot n \cdot \log(n))$
1986	Goldberg, Tarjan	Goldberg-Tarjan-Algorithmus	$\mathcal{O}\left(m \cdot n \cdot \log\left(\frac{n^2}{m}\right)\right)$
1987	Ahuja, Orlin		$\mathcal{O}(m \cdot n + n^2 \cdot \log(u_{\max}))$
1987	Ahuja, Orlin, Tarjan		$\mathcal{O}\left(m \cdot n \cdot \log\left(2 + \frac{n \cdot \sqrt{\log(u_{\max})}}{m}\right)\right)$
1990	Cheriyān, Hagerup, Mehlhorn		$\mathcal{O}\left(\frac{n^3}{\log(n)}\right)$
1990	Alon		$\mathcal{O}\left(m \cdot n + \sqrt[3]{n^8} \cdot \log(n)\right)$
1992	King, Rao, Tarjan		$\mathcal{O}(m \cdot n + n^{2+e})$
1993	Philipps, Westbrook ^[2]		$\mathcal{O}\left(m \cdot n \cdot \log_{\frac{m}{n}}(n) + n^2 \cdot \log(n)^{2+e}\right)$
1994	King, Rao, Tarjan ^[3]		$\mathcal{O}\left(m \cdot n \cdot \log_{\frac{m}{n \cdot \log(n)}}(n)\right)$
1997	Goldberg, Rao ^[4]		$\mathcal{O}\left(\min\{\sqrt{m}, \sqrt[3]{n^2}\} \cdot m \cdot \log\left(\frac{n^2}{m}\right) \cdot \log(u_{\max})\right)$
2012	Orlin, King, Rao, Tarjan		$\mathcal{O}(n \cdot m)$
2022	Chen, Kyng, Liu Peng, Gutenberg, Sachdeva ^[5]		$m^{1+o(1)}$ für ganzzahlige Kapazitäten, die polynomiell beschränkt sind

Algorithmen

Ford-Fulkerson Algorithmus

- 1) $f \leftarrow 0$
- 2) **while** \exists s-t-Pfad P in N_f **do**
- 3) **Augmentiere** den Fluss entlang P
- 4) **return** f

- kann unendlich laufen wenn $c : A \rightarrow \mathbb{R}$
- läuft immer endlich wenn $c : A \rightarrow \mathbb{N}_0$
- Laufzeit: $\mathcal{O}(mnU)$ wobei $c : A \rightarrow \mathbb{N}_0$ und $U = \max_{e \in A} c(e)$

Andere Methode/Algorithmen

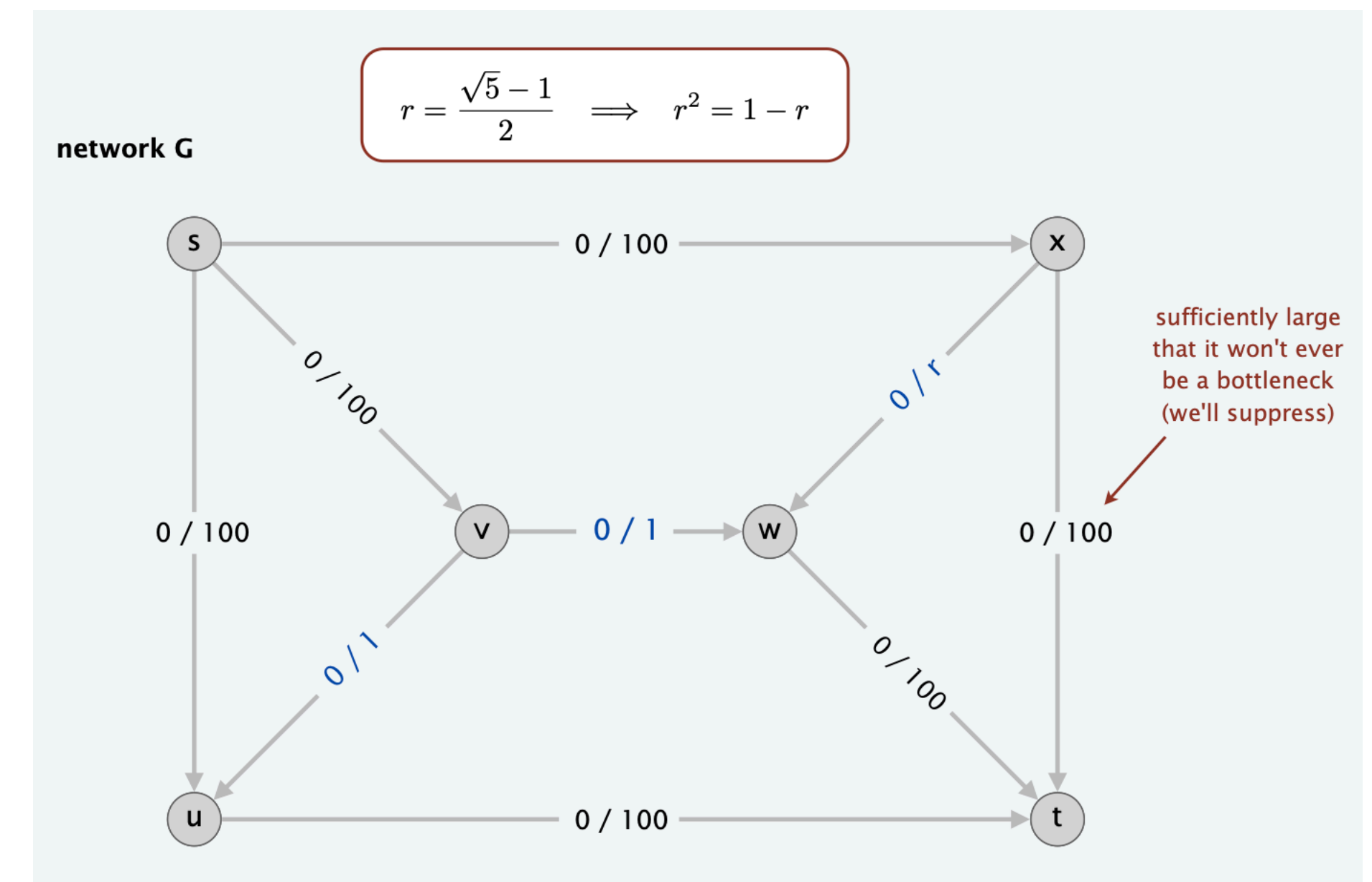
- 1) Capacity Scaling (Kapazitäten ganzzahlig + höchstens U)
 $\mathcal{O}(mn(1 + \log U))$
- 2) Dynamic Trees
 $\mathcal{O}(mn \log n)$

Satz für Ford-Fulkerson Algorithmus

Für N mit $c : A \rightarrow \mathbb{N}_0^{\leq U}$ gilt:

- 1) es gibt einen ganzzahligen maximalen Fluss
- 2) der Algo findet den Fluss in $\mathcal{O}(mnU)$

Pathological Example:



Kahoot

Aufgaben