Satz

Sei G ein Flussnetzwerk, $w(e) \in \mathbb{N}$ oder $w(e) \in \mathbb{Q}$. Dann existiert ein maximaler Fluss.

Beweis: Sei $\phi_0 \equiv 0$ der Nullfluss.
Wir vergrößern einen vorhandenen Fluss mittels augmentierender Pfade, also um $\delta \in \mathbb{N}$ bzw. $\delta \in \frac{1}{k}\mathbb{N}$.
Nach endlich vielen Schritten erhalten wir $\phi_{\max}$. □

Folgerung

Sei G ein Flussnetzwerk. Dann existiert ein maximaler Fluss.

Beweis: Folgt aus Stetigkeitsgründen aus dem vorigen Satz, denn die Menge aller Flüsse, aufgefasst als Teilmenge von $\mathbb{R}^{|E|}$, ist kompakt. □

Flüsse

Wir definieren das Restnetzwerk $G_\phi = (V_\phi, E_\phi, c_\phi)$:

$$
\begin{aligned}
V(G_\phi) &= V \\
E(G_\phi) &= \{(u,v) \mid (u,v) \in E \text{ mit } \phi(u,v) < w(u,v) \\
&\qquad \text{oder } (v,u) \in E \text{ mit } \phi(v,u) > 0\}
\end{aligned}
$$

$$
c_\phi(u,v) = \begin{cases} w(u,v) - \phi(u,v), & \text{falls } (u,v) \in E(G_\phi) \cap E; \\ \phi(u,v), & \text{falls } (u,v) \in E(G_\phi) \text{ und } (v,u) \in E. \end{cases}
$$

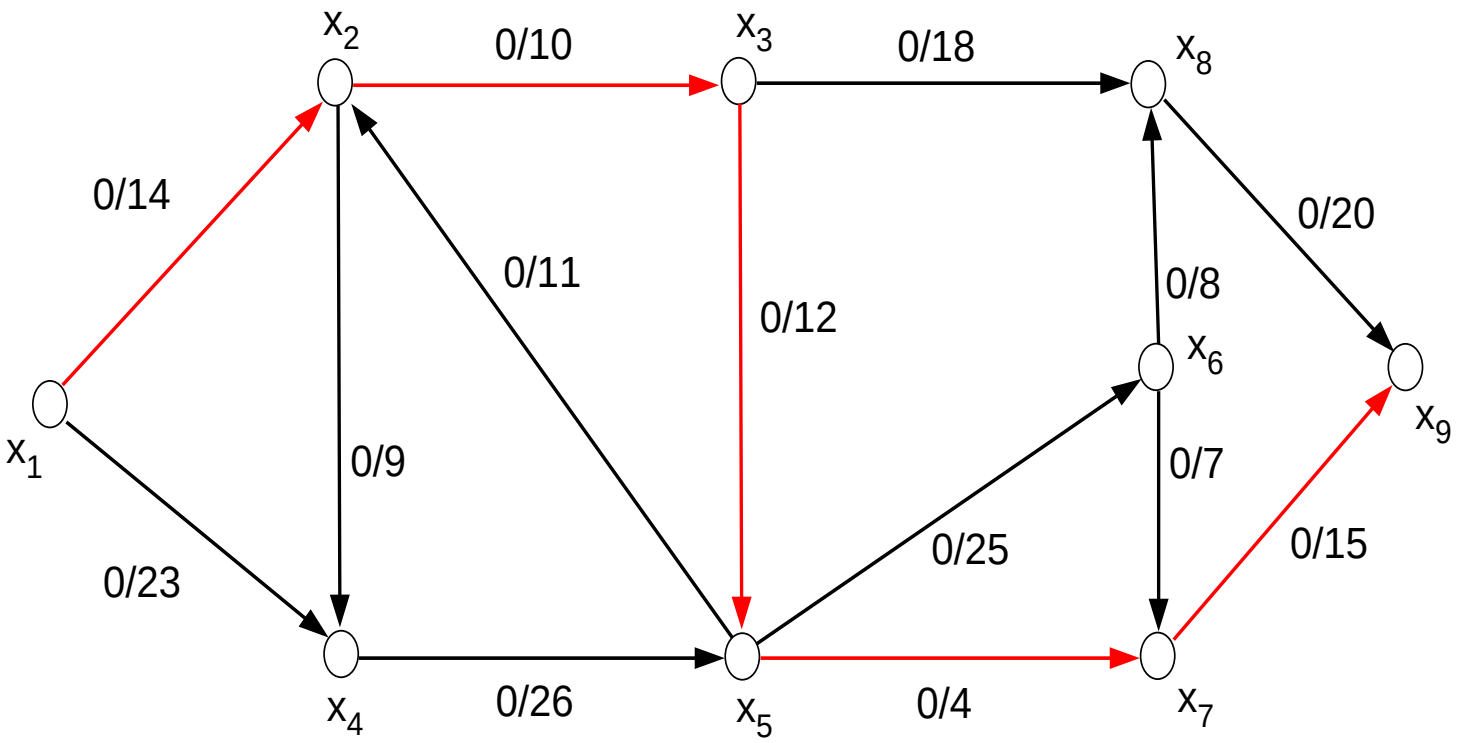
Algorithm FIND-PATH(V, E, w, s, t, ϕ)

```
 1: δ(s) := ∞; W := {s}; S = ∅;
 2: Sei P eine leere Liste
 3: while |W| > |S| do
 4:     S := W
 5:     for x ∈ W do
 6:         for y ∈ Γ⁺(x) ∩ (V \ W) do
 7:             if φ(x,y) < w(x,y) then
 8:                 π(y) := x
 9:                 δ(y) := min(δ(x), w(x,y) − φ(x,y))
10:                 W := W ∪ {y}
11:             end if
12:         end for
13:         for y ∈ Γ⁻(x) ∩ (V \ W) do
14:             if φ(y,x) > 0 then
15:                 π(y) := x;
16:                 δ(y) := min(δ(x), φ(y,x))
17:                 W := W ∪ {y}
18:             end if
19:         end for
20:     end for
21: end while
21: if t ∉ W then
22:     return [FALSE,P,0]
23: else
24:     x := t
25:     while x ≠ s do
26:         PUSH((π(x), x), P)
27:         x := π(x)
28:     end while
29:     return [TRUE,P,δ(t)]
30: end if
```

Algorithm FORD-FULKERSON(V, E, w, s, t)

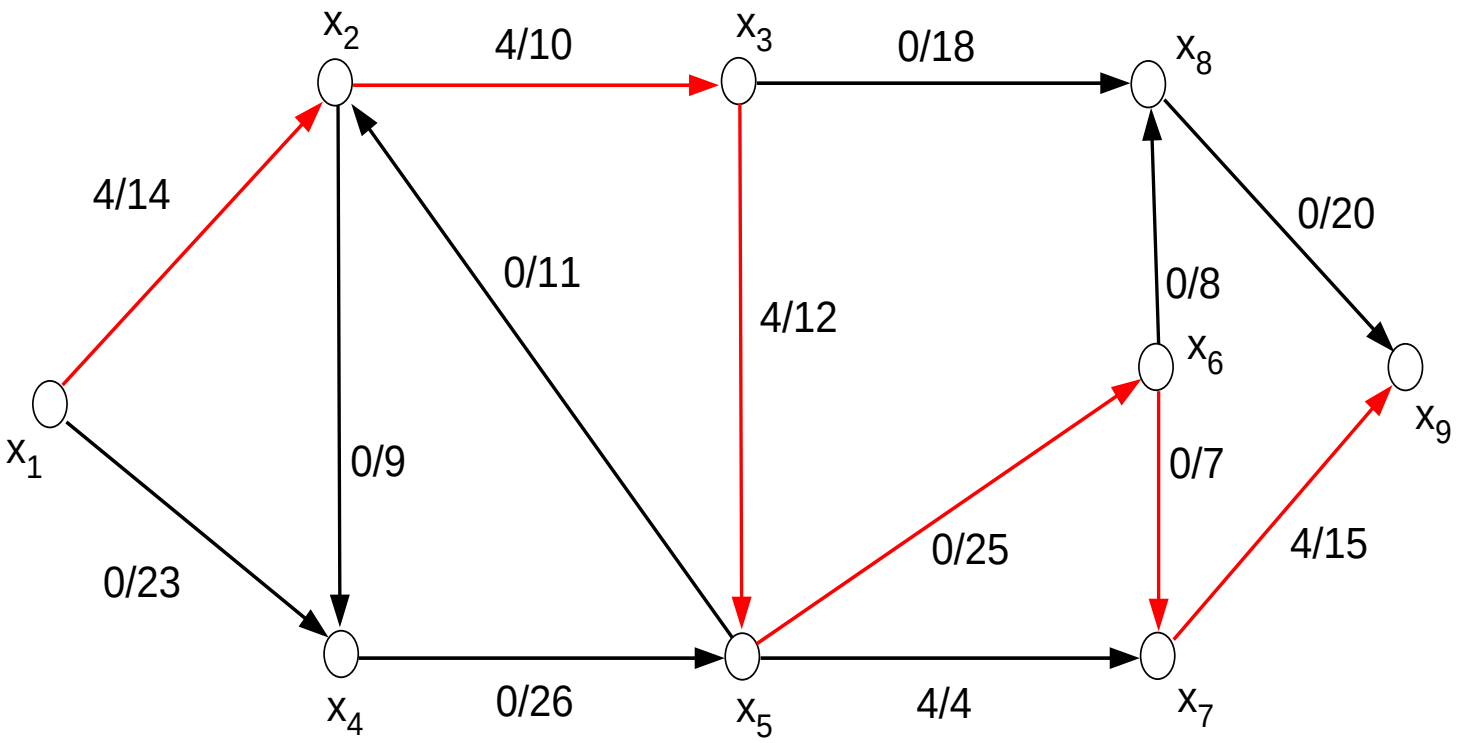
```
 1: for (u, v) ∈ E do
 2:     φ(u, v) := 0
 3: end for
 4: while PATH(G_φ, s, t)=TRUE do
 5:     p =FINDPATH(G_φ, s, t)
 6:     δ := min{c_φ(u, v) | (u, v) ∈ E(p)}
 7:     for (u, v) ∈ E(p) do    % ∃ augmentierender Pfad
 8:         if (u, v) ∈ E then
 9:             φ(u, v) := φ(u, v) + δ
10:         else
11:             φ(u, v) := φ(u, v) − δ
12:         end if
13:     end for
14: end while
```

Flüsse



$\pi(x_2) := x_1, \delta(x_2) := 14, \quad \pi(x_4) := x_1, \delta(x_4) := 23, \quad \pi(x_3) := x_2, \delta(x_3) := 10,$
$\pi(x_5) := x_3, \delta(x_5) := 10, \quad \pi(x_8) := x_3, \delta(x_8) := 10, \quad \pi(x_6) := x_5, \delta(x_6) := 10,$
$\pi(x_7) := x_5, \delta(x_7) := 4, \quad \pi(x_9) := x_7, \delta(x_9) := 4$

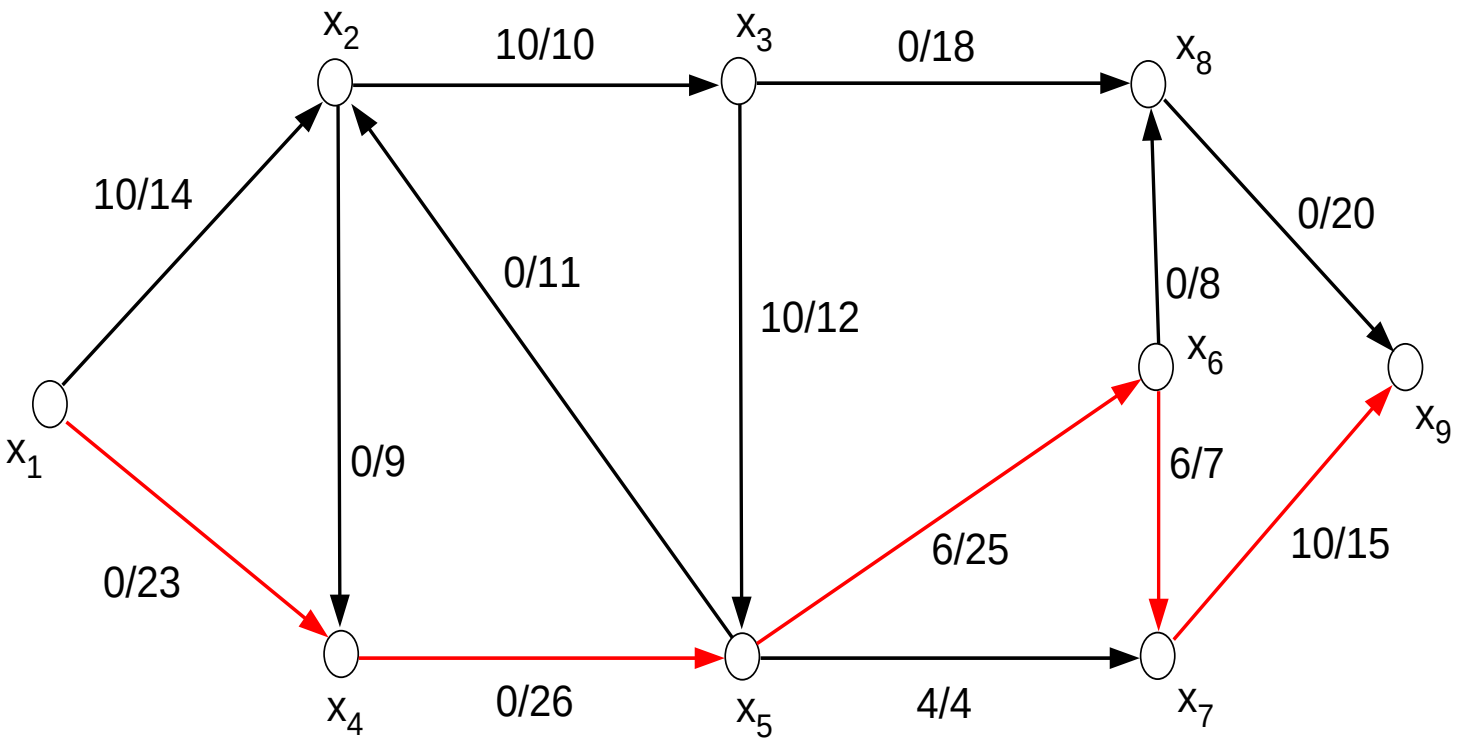
Flüsse



$x_1 \longrightarrow x_2 \longrightarrow x_3 \longrightarrow x_5 \longrightarrow x_6 \longrightarrow x_7 \longrightarrow x_9$

$\delta(x_9) = 6$

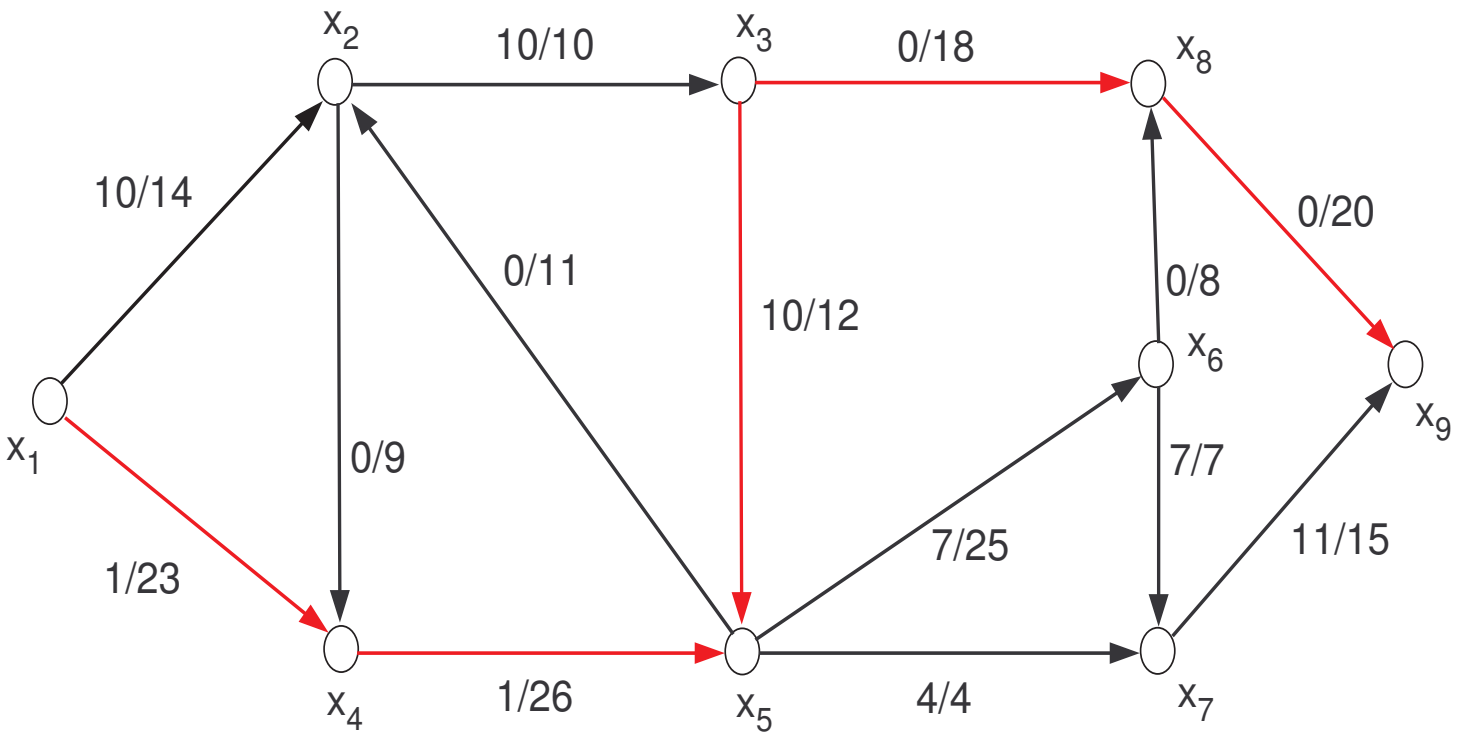
Flüsse



$x_1 \longrightarrow x_4 \longrightarrow x_5 \longrightarrow x_6 \longrightarrow x_7 \longrightarrow x_9$

$\delta(x_9) = 1$

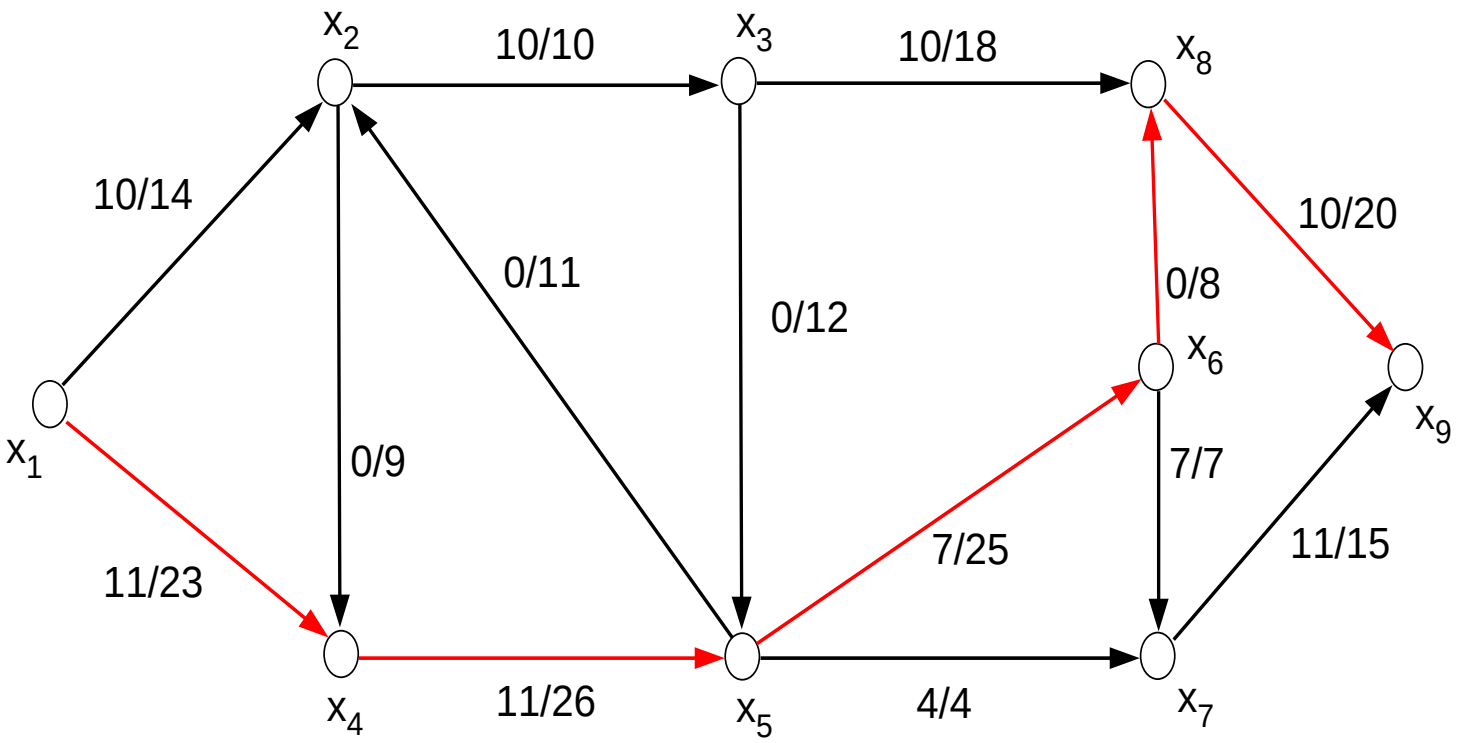
Flüsse



$\pi(x_2) := x_1, \delta(x_2) := 4, \quad \pi(x_4) := x_1, \delta(x_4) := 22, \quad \pi(x_5) := x_4, \delta(x_5) := 22,$
$\pi(x_6) := x_5, \delta(x_6) := 18, \quad \pi(x_5) := -x_3, \delta(x_3) := 10, \quad \pi(x_8) := x_3, \delta(x_8) := 10$
$\pi(x_9) := x_8, \delta(x_9) := 10,$

$x_1 \xrightarrow{+10} x_4 \xrightarrow{+10} x_5 \xleftarrow{-10} x_3 \xrightarrow{+10} x_8 \xrightarrow{+10} x_9$

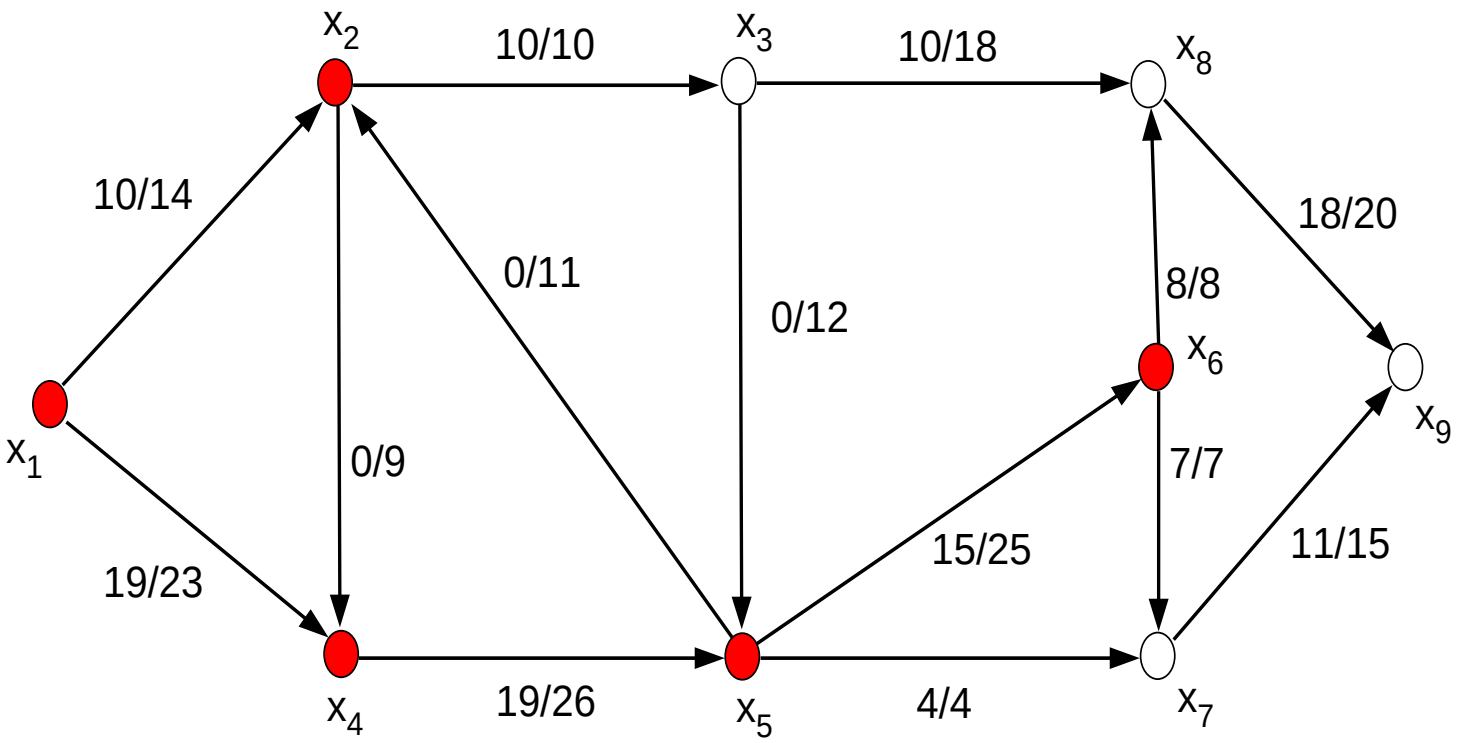
Flüsse



$x_1 \longrightarrow x_4 \longrightarrow x_5 \longrightarrow x_6 \longrightarrow x_8 \longrightarrow x_9$

$\delta(x_9) = 8$

Flüsse


x2
10/10
x3
10/18
x8
10/14
18/20
0/11
0/12
8/8
x6
x1
0/9
7/7
x9
15/25
11/15
19/23
x4
19/26
x5
4/4
x7

Flüsse

Laufzeit:

- $\mathcal{O}\left(|\phi_{\max}|\left(|V|+|E|\right)\right) = \mathcal{O}\left(\max_{e\in E} w(e)\left(|V|+|E|\right)\right)$; nicht polynomiell
- Finden der augmentierenden Pfade über BFS: Algorithmus von Edmonds-Karp; $\mathcal{O}\left(\left(|V||E|^2\right)\right)$
- Pfade mit großem δ suchen: Algorithmus von Dinic; $\mathcal{O}\left(\left(|V|^2|E|\right)\right)$
- Algorithmus von Goldberg-Tarjan: $\mathcal{O}\left(\left(|V|^2\sqrt{|E|}\right)\right)$. Arbeitet nicht mit augmentierenden Pfaden, sondern abgewandelte Flüsse und Push-Relabel-Techniken)

16) Die schnelle Fouriertransformation

Die schnelle Fouriertransformation

Wir betrachten Polynome $A(x), B(x) \in \mathbb{C}[x]$ mit $\deg A, \deg B < n$ (Gradschranke n), also

$$A(x) = \sum_{i=0}^{n-1} a_i x^i,\ B(x) = \sum_{i=0}^{n-1} b_i x^i.$$

Für

$$C(x) = A(x) + B(x),$$

gilt $\deg C < n$. Berechnungsaufwand: $\Theta(n)$.

$$D(x) = A(x)B(x) = \sum_{i=0}^{2n-2} d_i x^i,$$

$$d_i = \sum_{j=0}^{i} a_j b_{i-j}, \qquad \deg D < 2n - 1.$$

Darstellung von Polynomen

Koeffizientendarstellung (KD):

$$A(x) = \sum_{i=0}^{n-1} a_i x^i \hat{=} (a_0, a_1, \ldots, a_{n-1}) = \mathbf{a}$$

Auswerten: mittels *Horner-Schema*:

$$A(x_0) = a_0 + x_0\,(a_1 + x_0(a_2 \ldots x_0(a_{n-2} + x_0 a_{n-1})))$$

Aufwand: $\Theta(n)$
Ebenso für die Addition:

$$A(x)\hat{=}\mathbf{a},\ B(x)\hat{=}\mathbf{b} \implies A(x) + B(x)\hat{=}\mathbf{a} + \mathbf{b}.$$

Multiplikation $D(x) = A(x)B(x) : \mathbf{d} = \mathbf{a} * \mathbf{b}$ (Faltung).
Aufwand: $\Theta(n^2)$

Die schnelle Fouriertransformation

Stützstellendarstellung (SSD):

$$\{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}, \text{ wobei } A(x_i) = y_i.$$

Berechnen einer SSD aus einer KD: n-mal auswerten; $\Theta(n^2)$.

Umgekehrt: Interpolation, denn man sucht Lösung von

$$V(x_0, x_1, \ldots, x_{n-1}) \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

mit

$$V(x_0, x_1, \ldots, x_{n-1}) := \begin{pmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^{n-1} \\ \vdots & \vdots & \vdots & & \vdots \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \ldots & x_{n-1}^{n-1} \end{pmatrix} \quad \text{(Vandermonde-Matrix).}$$

Bem.: $\det V \neq 0 \iff x_0, \ldots, x_n$ paarweise verschieden.

Die schnelle Fouriertransformation

Umsetzung mittels

- Lösen eines linearen Gleichungssystems: $\Theta\left(n^3\right)$;
- schneller mit Lagrange-Interpolation: $\Theta\left(n^2\right)$.

$$A(x) = \sum_{k=0}^{n-1} y_k \frac{\prod_{j \neq k}(x - x_j)}{\prod_{j \neq k}(x_k - x_j)}.$$

Operationen für SSD:

- Addition: $C(x_k) = A(x_k) + B(x_k)$, sofern Stützstellen für SSD von A und B gleich. Aufwand $\Theta(n)$.
- Multiplikation: $D(x_k) = A(x_k)B(x_k)$ ($\deg D = \deg A + \deg B$!) Erweiterte SSD von $A(x)$ und $B(x)$ nötig:

$$\begin{aligned} A(x) &\leftrightarrow \{(x_0, y_0), \ldots, (x_{2n-1}, y_{2n-1})\}, \\ B(x) &\leftrightarrow \{(x_0, y_0'), \ldots, (x_{2n-1}, y_{2n-1}')\}, \\ D(x) &\longleftrightarrow \{(x_0, y_0 y_0'), \ldots, (x_{2n-1}, y_{2n-1} y_{2n-1}')\}. \end{aligned}$$

  Aufwand $\Theta(n)$ für erweiterte SSD von $A(x)$ und $B(x)$.

Die schnelle Fouriertransformation

- Auswerten: Konvertieren in KD, dann Auswerten.

Schnelle Multiplikation von Polynomen in KD:

- schnelle Konversion KD $\leftrightarrow$ SSD nötig;
- geschickte Wahl der Stützstellen!

Idee:

- Auswerten mit diskreter Fouriertransformation (DFT), liefert SSD;
- Interpolation mit inverser DFT (IDFT);
- DFT und IDFT als Divide & Conquer:
  Fast Fourier Transform (FFT)

Die schnelle Fouriertransformation

Vorgangsweise bei der Multiplikation:
Gegeben:

$$A(x) = \sum_{i=0}^{n-1} a_i x^i, \qquad B(x) = \sum_{i=0}^{n-1} b_i x^i,$$

Gradschranke n, o.B.d.A.: $n = 2^k$ (mit Nullen auffüllen)

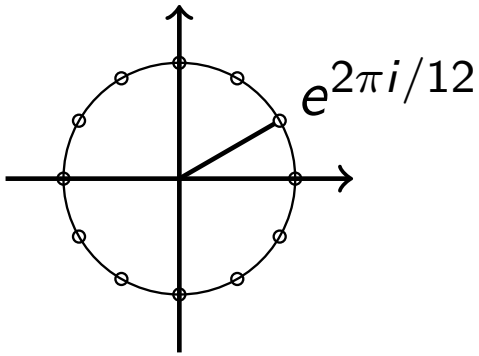
(i) Verdoppeln der Gradschranke: $A(x)$ und $B(x)$ um n Nullkoeffizienten erweitern;

(ii) Auswerten: SSD der Länge $2n$ von $A(x)$, $B(x)$ (FFT);

(iii) Punktweise Multiplikation: $C(x_k) = A(x_k)B(x_k)$;

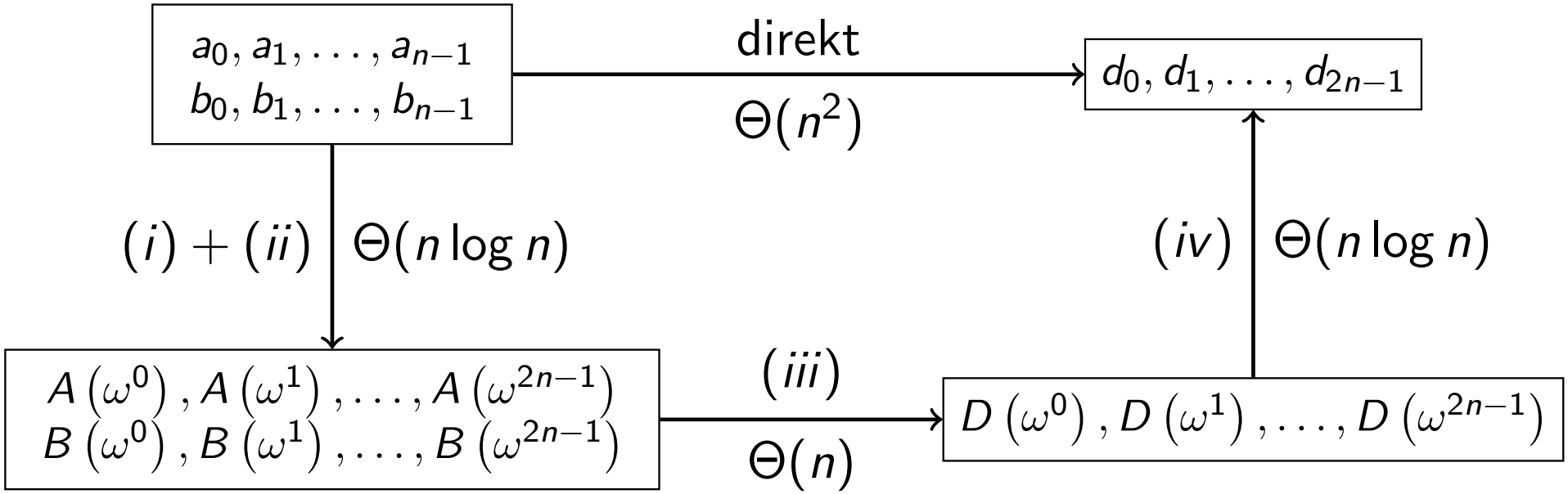
(iv) Interpolation: KD von $C(x)$ (IDFT, wieder mit FFT)

Wahl der Stützstellen: Potenzen von $\omega = \sqrt[2n]{1} = e^{\pi i/n}$



Die schnelle Fouriertransformation

Schema:



DFT und FFT

Für $\zeta_n = e^{2\pi i/n}$ ist $(\{\zeta_n^0, \zeta_n^1, \ldots, \zeta_n^{n-1}\}, \cdot) \cong (\mathbb{Z}_n, +)$.

Lemma

$$\zeta_{dn}^{dk} = \zeta_n^k, \qquad \zeta_{2n}^2 = \zeta_n, \qquad \sum_{j=0}^{n-1} (\zeta_n^k)^j = 0 \text{ für } n \nmid k$$

DFT: Sei $A(x) = \sum_{j=0}^{n-1} a_j x^j$ und $\omega = \zeta_n$. Dann gilt

$$y_k := A(\omega^k) = \sum_{j=0}^{n-1} a_j \omega^{kj}.$$

Für $\mathbf{y} = (y_0, \ldots, y_{n-1})$ und $\mathbf{a} = (a_0, \ldots, a_{n-1})$ haben wir $\mathbf{y} = \mathrm{DFT}_n(\mathbf{a})$, also

$$\mathbf{y} = V(1, \omega, \ldots, \omega^{n-1})\mathbf{a}.$$

Die schnelle Fouriertransformation

Sei $n = 2^k$ und

$$A_0(x) = a_0 + a_2 x + a_4 x^2 + \cdots + a_{n-2} x^{\frac{n}{2}-1},$$
$$A_1(x) = a_1 + a_3 x + a_5 x^2 + \cdots + a_{n-1} x^{\frac{n}{2}-1}.$$

Dann gilt $A(x) = A_0(x^2) + xA_1(x^2)$, d.h. man muss $A_i(\omega^{2j})$ bestimmen.
Die ω^{2j} sind die $\frac{n}{2}$ Wurzeln $\sqrt[n/2]{1}$.
Jede tritt genau zweimal auf. Dadurch wird die Berechnung von DFT_n in 2 $\mathrm{DFT}_{n/2}$-Berechnungen aufgeteilt.

Die schnelle Fouriertransformation

Algorithm REKURSIVE-FFT(**a**)

```
 1: n := |a|        % Bem.: n = 2^k
 2: if n = 1 then
 3:     return a
 4: end if
 5: ω := e^{2πi/n}    % Aufwand: Θ(n)
 6: ω' := 1
 7: a_0 := (a_0, a_2 ..., a_{n-2})
 8: a_1 := (a_1, a_3 ..., a_{n-1})
 9: y_0 :=RECURSIVE-FFT(a_0)
10: y_1 :=RECURSIVE-FFT(a_1)
11: for k = 0 to n/2 - 1 do
12:     y_k := y_{0,k} + ω' y_{1,k}
13:     y_{k+(n/2)} := y_{0,k} - ω' y_{1,k}
14:     ω' := ω' ω
15: end for
16: return y
```

Laufzeit: $T(n) = 2T(\frac{n}{2}) + \Theta(n)$ und daher $T(n) = \Theta(n \log n)$.

Die schnelle Fouriertransformation

Korrektheit:
Induktion: $n = 1 : y_0 = a_0\omega^0 = a_0$ ✓

Sei ω' das aktuelle ω^k: In der **for**-Schleife gilt $\omega' = \omega^k$.
Ziel: $y_k = A(\omega^k) = A_0(\omega^{2k}) + \omega^k A_1(\omega^{2k})$.

$$y_{0,k} = A_0(\omega^{2k}) = A_0(\zeta_{\frac{n}{2}}^k), \text{ für } 0 \le k \le \frac{n}{2} - 1,$$

$$y_{1,k} = A_1(\zeta_{\frac{n}{2}}^k), \text{ für } 0 \le k \le \frac{n}{2} - 1,$$

denn im rekursiven Aufruf wird $\omega = \zeta_{n/2}$ gesetzt. Sei $0 \le k \le (n/2) - 1$.

$$\begin{aligned} y_k &= y_{0,k} + \omega' y_{1,k} = A_0(\omega^{2k}) + \omega^k A_1(\omega^{2k}) = A(\omega^k), \\ y_{k+\frac{n}{2}} &= y_{0,k} - \omega' y_{1,k} \\ &= A_0(\omega^{2k}) - \omega^k A_1(\omega^{2k}) \\ &= A_0(\omega^{2k+n}) + \omega^{k+\frac{n}{2}} A_1(\omega^{2k+n}) = A(\omega^{k+\frac{n}{2}}). \end{aligned}$$

Die schnelle Fouriertransformation

IDFT: Sei $\mathbf{y} = \mathrm{DFT}_n(\mathbf{a})$ und $V_n = V(1, \omega, \omega^2, \ldots, \omega^{n-1})$, also $\mathbf{y} = V_n\mathbf{a}$ und $\mathbf{a} = V_n^{-1}\mathbf{y}$

Lemma

Die Inverse von V_n ist gegeben durch

$$V_n^{-1} = \frac{1}{n} V(1, \omega^{-1}, \omega^{-2}, \ldots, \omega^{-n+1}).$$

Beweis: Sei V_n' die oben genannte Matrix. Dann gilt

$$(V_n' \cdot V_n)_{j,k} = \frac{1}{n} \sum_{\ell=0}^{n-1} \omega^{-\ell j} \omega^{\ell k} = \frac{1}{n} \sum_{\ell=0}^{n-1} \left(\omega^{k-j}\right)^{\ell}.$$

Wir haben $-(n+1) \leq k - j \leq n - 1$.

- Wenn $k = j$, dann $\omega^{k-j} = 1$ und folglich $(V_n' \cdot V_n)_{j,j} = 1$.
- Wenn $k \neq j$, dann $n \not| k - j$, daher $\omega^{k-j} \neq 1$ und folglich $(V_n' \cdot V_n)_{j,k} = 0$.

□

Die schnelle Fouriertransformation

Algorithmus für IDFT:

- In FFT **a** und **y** vertauschen;
- ω durch $\omega^{-1} = \overline{\omega}$ ersetzen;
- am Ende Ergebnis durch n dividieren.

Daher ist der Aufwand zur Berechnung von DFT_n^{-1} ebenfalls $\Theta(n \log n)$.

Berechnung der Faltung:

Satz

Seien $\mathbf{a}, \mathbf{b} \in \mathbb{C}^n$, $n = 2^k$. *Wir setzen* $\mathbf{a}' = (\mathbf{a}, \mathbf{0})$ *und* $\mathbf{b}' = (\mathbf{b}, \mathbf{0})$. *Dann gilt*

$$\mathbf{a} * \mathbf{b} = \mathrm{DFT}_{2n}^{-1}(\mathrm{DFT}_{2n}(\mathbf{a}') \odot \mathrm{DFT}_{2n}(\mathbf{b}')).$$

17) Der Euklidische Algorithmus

Der Euklidische Algorithmus

Satz

Für $a \in \mathbb{Z}$ und $m \in \mathbb{N}^+$ existiert genau ein Paar $(q, r) \in \mathbb{Z}^2$ mit $0 \leq r < m$ und $a = qm + r$.

Beweis: Mit $q = \left\lfloor \frac{a}{m} \right\rfloor$ haben wir $\frac{a}{m} - 1 < q \leq \frac{a}{m}$ und daher

$$\underbrace{a - \frac{a}{m}m}_{0} \leq \underbrace{a - qm}_{r} < \underbrace{a - \left(\frac{a}{m} - 1\right)m}_{m}$$

Eindeutigkeit: Aus $a = q_1 m + r_1 = q_2 m + r_2$ folgt

$$0 = (q_1 - q_2)m + r_1 - r_2$$

Das impliziert

$$-m + 1 \leq r_2 - r_1 \leq m - 1 \text{ und somit } q_1 = q_2, \quad r_1 = r_2. \qquad \square$$

Der Euklidische Algorithmus

Definition

Wir sagen b teilt a, $b|a$, wenn es ein $k \in \mathbb{Z}$ gibt mit $a = kb$. Eine ganze Zahl d heißt größter gemeinsamer Teiler von a und b, $d = \mathrm{ggT}(a, b)$, wenn $d|a$ und $d|b$ und weiters jede Zahl t mit $t|a$ und $t|b$ auch d teilt.

Satz

Seien a, b nicht beide gleich 0. Wir setzen

$$M := \{ax + by \mid x, y \in \mathbb{Z} \text{ und } ax + by > 0\}.$$

Dann ist $s := \min M$ ein größter gemeinsamer Teiler von a und b.

Beweis: Gelte zunächst $a = 0$ oder $b = 0$, o.B.d.A. $b = 0$ (und daher $a \neq 0$).
Dann gilt $s = |a|$, also $s = \mathrm{ggT}(a, 0)$.

Der Euklidische Algorithmus

Gelte nun $a \neq 0$ und $b \neq 0$.

Nach Konstruktion gibt es x und y mit $s = ax + by$. Sei $q = \left\lfloor \frac{a}{s} \right\rfloor$.
Dann gilt

$$a \mod s = a - qs = a - q(ax + by) = (1 - qx)a - qyb \in M \cup \{0\}.$$

Wegen $0 \leq a \mod s < s$ muss dann $a \mod s = 0$ gelten.
Analog: $b \mod s = 0$, also $s|a$ und $s|b$.

Sei nun t ein gemeinsamer Teiler von a und b.
Dann gilt $t|ax + by$, also $t|s$ und somit $s = \mathrm{ggT}(a, b)$. □

Bemerkung: Wenn d_1 und d_2 ggT von a und b sind, dann gilt
$d_1|d_2$ und $d_2|d_1$, also gibt es $x, y \in \mathbb{Z}$ mit $d_2 = xd_1$ und $d_1 = yd_2$.
Daraus folgt $d_2 = xyd_2$, gleichbedeutend mit $1 = xy$,
also $x = y = 1$ oder $x = y = -1$ und somit $d_2 = \pm d_1$.

Der Euklidische Algorithmus

Definition

$$\mathrm{ggT}(a,b) := \begin{cases} s = \min M, & \textit{falls } (a,b) \neq (0,0); \\ 0, & \textit{falls } (a,b) = (0,0). \end{cases}$$

Satz

Sei $a, b \in \mathbb{Z}$, $a \geq 0$, $b > 0$. *Dann gilt*

$$\mathrm{ggT}(a,b) = \mathrm{ggT}(b, a \mod b).$$

Beweis: Sei $d = \mathrm{ggT}(a,b)$ und $t = \mathrm{ggT}(b, a \mod b)$.
Dann ist d ein Teiler von $a - \left\lfloor \frac{a}{b} \right\rfloor b$ und b daher gilt $d|t$.
t ist Teiler von b und von $a \mod b = a - \left\lfloor \frac{a}{b} \right\rfloor b$.
Daher teilt t sowohl a als auch b. Es folgt $t|d$. □

Der Euklidische Algorithmus

Algorithm EUKLID(a, b)

```
1: if b = 0 then
2:     return a
3: else
4:     return EUKLID(b, a mod b)
5: end if
```

Beispiel:

EUKLID(90,12) =EUKLID(12,6) =EUKLID(6,0)=6.

2 rekursive Aufrufe.

Der Euklidische Algorithmus

Lemma

Sei $a > b \geq 1$. Weiters führe $\mathrm{EUKLID}(a, b)$ k rekursive Aufrufe aus. Dann gilt $a \geq F_{k+2}$ und $b \geq F_{k+1}$.

Beweis: Induktion nach k:

$k = 1$: $b \geq 1 = F_2$ und $a > b$, also $a \geq 2 = F_3$. ✓

$k \to k + 1$: Wegen $b > a \mod b$ ist $x > y$ in jedem Aufruf der Form EUKLID(x, y).
EUKLID(a, b) =EUKLID$(b, a \mod b)$ und dies führt k rekursive Aufrufe aus.
Daraus folgt $b \geq F_{k+2}$ und $(a \mod b) \geq F_{k+1}$ und daher weiters

$$\begin{aligned} b + (a \mod b) &= b + \left(a - \left\lfloor \frac{a}{b} \right\rfloor b\right) \\ &= a - b\left(\left\lfloor \frac{a}{b} \right\rfloor - 1\right) \leq a. \end{aligned}$$

Also insgesamt $a \geq b + (a \mod b) \geq F_{k+2} + F_{k+1} = F_{k+3}$. □

Der Euklidische Algorithmus

Folgerung (Satz von Lamé)

Die Anzahl der rekursive Aufrufe von $\mathrm{EUKLID}(a, b)$ *ist kleiner als* k*, falls* $b < F_{k+1}$ *und* $a > b \geq 1$*.*

Bemerkungen:

- Schranke bestmöglich: $\mathrm{EUKLID}(F_{k+1}, F_k)$ führt $k-1$ rekursive Aufrufe aus.
  - Beweis: Induktion nach k
  - $k = 2$: EUKLID(2,1)=EUKLID(1,0)=1.
  - $\mathrm{EUKLID}(F_{k+1}, F_k) = \underbrace{\mathrm{EUKLID}(F_k, \underbrace{F_{k+1} \bmod F_k}_{F_{k-1}})}_{k-2 \text{ Aufrufe}}$
- $F_k \sim \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^k$ und folglich gilt $k = \Theta(\log b)$.
- EUKLID ist linear in der Bitlänge der kleineren Zahl.

Der Euklidische Algorithmus

Der erweiterter Euklidische Algorithmus

$\exists x, y \in \mathbb{Z} : d = \mathrm{ggT}(a, b) = ax + by$

gegeben: a, b; gesucht: d, x, y.

Algorithm EUKLID+(a, b)

```
1: if b = 0 then
2:     return (a, 1, 0)
3: else
4:     (d', x', y') :=EUKLID+(b, a mod b)
5:     (d, x, y) := (d', y', x' − ⌊a/b⌋ y')
6:     return (d, x, y)
7: end if
```

Der Euklidische Algorithmus

Beispiel:

a	b	$\lfloor \frac{a}{b} \rfloor$	d	x	y
356	132	2			
132	92	1			
92	40	2			
40	12	3			
12	4	3			
4	0	–	4	1	0

$$(d, x, y) := (d', y', x' - \lfloor \tfrac{a}{b} \rfloor \, y')$$

Der Euklidische Algorithmus

Beispiel:

a	b	$\lfloor \frac{a}{b} \rfloor$	d	x	y
356	132	2	4		
132	92	1	4		
92	40	2	4		
40	12	3	4		
12	4	3	4		
4	0	–	4	1	0

$$(d, x, y) := (d', y', x' - \lfloor \tfrac{a}{b} \rfloor \, y')$$

Der Euklidische Algorithmus

Beispiel:

a	b	$\lfloor \frac{a}{b} \rfloor$	d	x	y
356	132	2	4		
132	92	1	4		
92	40	2	4		
40	12	3	4		
12	4	3	4	0	
4	0	–	4	1	0•

$$(d, x, y) := (d', y', x' - \lfloor \tfrac{a}{b} \rfloor \, y')$$

Der Euklidische Algorithmus

Beispiel:

a	b	$\lfloor \frac{a}{b} \rfloor$	d	x	y
356	132	2	4		
132	92	1	4		
92	40	2	4		
40	12	3	4		
12	4	3•	4	0	$1\ (= 1 - 3 \cdot 0)$
4	0	–	4	1•	0•

$(d, x, y) := (d', y', x' - \lfloor \frac{a}{b} \rfloor y')$

Der Euklidische Algorithmus

Beispiel:

a	b	$\lfloor \frac{a}{b} \rfloor$	d	x	y
356	132	2	4		
132	92	1	4		
92	40	2	4		
40	12	3	4	1	
12	4	3	4	0	$1\bullet$
4	0	–	4	1	0

$$(d, x, y) := (d', y', x' - \lfloor \tfrac{a}{b} \rfloor \, y')$$

Der Euklidische Algorithmus

Beispiel:

a	b	$\lfloor \frac{a}{b} \rfloor$	d	x	y
356	132	2	4		
132	92	1	4		
92	40	2	4		
40	12	3•	4	1	-3 $(= 0 - 3 \cdot 1)$
12	4	3	4	0•	1•
4	0	–	4	1	0

$(d, x, y) := (d', y', x' - \lfloor \frac{a}{b} \rfloor y')$

Der Euklidische Algorithmus

Beispiel:

a	b	$\lfloor \frac{a}{b} \rfloor$	d	x	y
356	132	2	4		
132	92	1	4		
92	40	2	4	-3	
40	12	3	4	1	-3
12	4	3	4	0	1
4	0	–	4	1	0

$$(d, x, y) := (d', y', x' - \lfloor \tfrac{a}{b} \rfloor \, y')$$

Der Euklidische Algorithmus

Beispiel:

a	b	$\lfloor \frac{a}{b} \rfloor$	d	x	y
356	132	2	4		
132	92	1	4		
92	40	2	4	-3	$7\ (= 1 - 2 \cdot (-3))$
40	12	3	4	1	-3
12	4	3	4	0	1
4	0	–	4	1	0

$(d, x, y) := (d', y', x' - \lfloor \frac{a}{b} \rfloor y')$

Der Euklidische Algorithmus

Beispiel:

a	b	$\lfloor \frac{a}{b} \rfloor$	d	x	y
356	132	2	4		
132	92	1	4	7	
92	40	2	4	-3	7
40	12	3	4	1	-3
12	4	3	4	0	1
4	0	–	4	1	0

$$(d, x, y) := (d', y', x' - \lfloor \tfrac{a}{b} \rfloor \, y')$$

Der Euklidische Algorithmus

Beispiel:

a	b	$\lfloor \frac{a}{b} \rfloor$	d	x	y
356	132	2	4		
132	92	1	4	7	-10 $(= -3 - 1 \cdot 7)$
92	40	2	4	-3	7
40	12	3	4	1	-3
12	4	3	4	0	1
4	0	–	4	1	0

$$(d, x, y) := (d', y', x' - \lfloor \tfrac{a}{b} \rfloor \, y')$$

Der Euklidische Algorithmus

Beispiel:

a	b	$\lfloor \frac{a}{b} \rfloor$	d	x	y
356	132	2	4	-10	27 $(= 7 - 2 \cdot (-10))$
132	92	1	4	7	-10
92	40	2	4	-3	7
40	12	3	4	1	-3
12	4	3	4	0	1
4	0	–	4	1	0

$$(d, x, y) := (d', y', x' - \lfloor \tfrac{a}{b} \rfloor \, y')$$

Der Euklidische Algorithmus

Beispiel:

a	b	$\lfloor \frac{a}{b} \rfloor$	d	x	y
356	132	2	4	-10	27
132	92	1	4	7	-10
92	40	2	4	-3	7
40	12	3	4	1	-3
12	4	3	4	0	1
4	0	–	4	1	0

Probe: $-10 \cdot 356 + 27 \cdot 132 = 4$

$(d, x, y) := (d', y', x' - \lfloor \frac{a}{b} \rfloor y')$

Der Euklidische Algorithmus

Beispiel:

a	b	$\lfloor \frac{a}{b} \rfloor$	d	x	y
356	132	2	4	-10	27
132	92	1	4	7	-10
92	40	2	4	-3	7
40	12	3	4	1	-3
12	4	3	4	0	1
4	0	–	4	1	0

Probe: $-10 \cdot 356 + 27 \cdot 132 = 4$ ✓

$(d, x, y) := (d', y', x' - \lfloor \frac{a}{b} \rfloor y')$

Der Euklidische Algorithmus

Korrektheit

Wenn $d = \mathrm{ggT}(a, b)$, $b = 0$, dann

$\mathrm{ggT}(a, b) = \mathrm{ggT}(a, 0) = 1 \cdot a + 0$ ✓

Sei $b \neq 0$:

$$\begin{aligned} d = \mathrm{ggT}(a, b) &= ax + by = \mathrm{ggT}(b, a \mod b) \\ &= bx' + (a \mod b)y' \\ &= bx' + \left(a - \left\lfloor \frac{a}{b} \right\rfloor b\right) y' \\ &= ay' + \left(x' - \left\lfloor \frac{a}{b} \right\rfloor y'\right) b \end{aligned}$$

Daher gilt $x = y'$, $y = \left(x' - \left\lfloor \frac{a}{b} \right\rfloor\right) y'$.

Laufzeit: EUKLID+ und EUKLID gleich aufwändig; $\mathcal{O}(\log b)$.

Anwendung: Modulare lineare Gleichungen

$$ax \equiv b\ (n) \text{ bzw. } \bar{a}\bar{x} = \bar{b} \text{ in } \mathbb{Z}_n.$$

Definition

$$\langle a \rangle := \langle \bar{a} \rangle := \{\bar{a} \cdot m \mid\ m > 0\} \subseteq \mathbb{Z}_n$$

Bemerkungen:

- $\langle a \rangle$ kann mit $\{a \cdot m \mod n \mid\ m > 0\} \subseteq \mathbb{Z}$ identifiziert werden.
- $(\langle a \rangle, +)$ ist Untergruppe von $(\mathbb{Z}_n, +)$.
  Daher ist $|\langle a \rangle|$ ein Teiler von n.

Der Euklidische Algorithmus

Satz

Seien $a, n \in \mathbb{N}^+$ *und* $d = \mathrm{ggT}(a, n)$. *Dann gilt*

$$\langle a \rangle = \langle d \rangle = \left\{ \bar{0}, \bar{d}, \overline{2d}, \ldots, \overline{\left(\frac{n}{d} - 1\right) d} \right\}$$

Beweis: Es gibt $x, y \in \mathbb{Z}$ mit $ax + ny = d$. Daraus folgt $ax \equiv d\ (n)$ und somit $\bar{d} \in \langle a \rangle$.

Wegen $\bar{d} \in \langle a \rangle$ haben wir $\langle d \rangle \subseteq \langle a \rangle$.

Noch z.z.: $\langle a \rangle \subseteq \langle d \rangle$:
Da d ein Teiler von a ist, existiert $k \in \mathbb{Z}$ mit $dk = a$. Somit gilt $\bar{a} \in \langle d \rangle$ und daher $\langle a \rangle \subseteq \langle d \rangle$. □

Folgerung

$ax \equiv b\ (n)$ ist genau dann lösbar, wenn $d = \mathrm{ggT}(a, n)$ ein Teiler von b ist.

Der Euklidische Algorithmus

Folgerung

Wenn $ax \equiv b\ (n)$ lösbar ist, dann gibt es d verschiedene Lösungen modulo n, wobei $d = \mathrm{ggT}(a, n)$.

Beweis: Wenn $ax \equiv b\ (n)$, dann ist $\bar{b} \in \langle a \rangle$ und $|\langle a \rangle| = \frac{n}{d}$.
Daher ist $(am \mod n)_{m=0,1,\ldots,n-1}$ periodisch.
Wegen $\mathrm{ord}_{\langle a \rangle}(a) = \frac{n}{d}$ ist $\frac{n}{d}$ die Periodenlänge.
Daher wird die Periode d mal wiederholt und $\bar{b}$ tritt d-mal auf. □

Satz

Sei $d = \mathrm{ggT}(a, n) = ax' + ny'$ (mit $x', y' \in \mathbb{Z}$) ein Teiler von b.
Dann ist $x_0 = x' \frac{b}{d} \mod n$ Lösung von $ax \equiv b\ (n)$.

Beweis: Folgt direkt aus $ax' \equiv d\ (n)$. □

Folgerung

Die Kongruenz $ax \equiv b\ (n)$ hat die Lösungen $x_i = x_0 + i\frac{n}{d} \mod n$, $i = 0, 1, \ldots, d-1$.

Der Euklidische Algorithmus

Folgerung

Sei $n > 1$, $\mathrm{ggT}(a, n) = 1$, *dann ist* $ax \equiv b\ (n)$ *eindeutig lösbar. Insbesondere ist dann* $ax \equiv 1\ (n)$ *eindeutig lösbar.* ($\bar{x} = \bar{a}^{-1}$ *in* $\mathbb{Z}_n$)

Algorithm SOLVE(a, b, n)

```
1: (d, x', y') :=EUKLID+(a, n)
2: if d|b then
3:     x_0 := (x'b)/d  mod n
4:     for i = 0 to d − 1 do
5:         print x_0 + (in)/d  mod n
6:     end for
7: else
8:     print „unlösbar!“
9: end if
```

18) Der chinesische Restsatz

Der chinesische Restsatz dient

- zum Lösen modularer linearer Gleichungssysteme,
- zur Beschreibung struktureller Ähnlichkeiten zwischen dem Ring $\mathbb{Z}_n$ und dem direkten Produkt von $\mathbb{Z}_{n_1}, \ldots, \mathbb{Z}_{n_r}$,
- dem Design effizienter Algorithmen, wenn das Arbeiten in $\mathbb{Z}_{n_1}, \ldots, \mathbb{Z}_{n_r}$ einfacher ist als in $\mathbb{Z}_{n_1} \times \cdots \times \mathbb{Z}_{n_r}$.

Lemma

Sei $m = pq$, $\mathrm{ggT}(p, q) = 1$. Dann ist $x \equiv y\ (m)$ äquivalent zu $x \equiv y\ (p) \wedge x \equiv y\ (q)$.

Beweis: „$\Longrightarrow$“: $x \equiv y\ (m) \Leftrightarrow \exists k \in \mathbb{Z} :\ x = y + km$
Daraus folgt $x = y + kpq$ und somit $x \equiv y\ (p) \wedge x \equiv y\ (q)$.

„$\Longleftarrow$“: Wegen $x \equiv y\ (p) \wedge x \equiv y\ (q)$ gibt es ein ℓ mit
$x - y = \ell p \equiv 0\ (q)$
Es gilt $\ell \equiv 0\ (q)$, also gibt es ℓ' mit $x - y = \ell' pq$.
Somit gilt $x \equiv y\ (m)$. □

Folgerung

Sei $m = m_1 \cdots m_r$ und für alle $i \neq j$ gelte $\mathrm{ggT}(m_i, m_j) = 1$, dann ist $x \equiv y\ (m)$ ist äquivalent zu $\forall 1 \leq i \leq r\ :\ x \equiv y\ (m_i)$.

Satz (Chinesischer Restsatz)

Sei $m = m_1 \cdots m_r$ und für alle $i \neq j$ gelte $\mathrm{ggT}(m_i, m_j) = 1$. Dann besitzt das System von Kongruenzen $x \equiv a_i\ (m_i)$, $1 \leq i \leq r$ eine eindeutige Lösung modulo m, welche lautet:

$$x \equiv \sum_{j=1}^{r} \frac{m}{m_j} a_j b_j\ (m), \quad \textit{wobei} \quad \frac{m}{m_j} b_j \equiv 1\ (m_j).$$

Beispiel: Gegeben sei

$$\left.\begin{array}{l} 3x \equiv 2\ (5) \\ 2x \equiv 7\ (11) \end{array}\right\} \Leftrightarrow \begin{array}{l} x \equiv 4\ (5) \\ x \equiv 9\ (11) \end{array} \qquad \begin{array}{l} m_1 = 5, \quad m_2 = 11, \quad m = 55, \\ a_1 = 4, \quad a_2 = 9 \end{array}$$

$$\left.\begin{array}{rcl} 11b_1 \equiv 1\ (5) & \Longrightarrow & b_1 = 1 \\ 5b_2 \equiv 1\ (11) & \Longrightarrow & b_2 = 9 \end{array}\right\} \Longrightarrow \underbrace{x \equiv 11 \cdot 4 \cdot 1 + 5 \cdot 9 \cdot 9}_{449} \equiv 9\ (55)$$

Beweis: 1. Teil: x ist Lösung

Die m_i sind paarweise relativ prim, daher gilt $\mathrm{ggT}\left(\frac{m}{m_j}, m_j\right) = 1$.
Folglich existieren die b_i, $i = 1, \ldots, r$.

Für $i \neq j$ gilt $\frac{m}{m_j} \equiv 0 \ (m_i)$. Daraus folgt (modulo m_i)

$$x \equiv \sum_{j=1}^{r} \frac{m}{m_j} a_j b_j \equiv \frac{m}{m_i} b_i a_i \equiv a_i \ (m_i)$$

2. Teil: x ist eindeutige Lösung

Seien x und y Lösungen modulo m, also

$$\forall i \ : \ x \equiv a_i \ (m_i) \quad \text{und} \quad y \equiv a_i \ (m_i).$$

$\Longrightarrow \ \forall i \ : \ x \equiv y \ (m_i)$ und aus dem Lemma folgt $x \equiv y \ (m)$ $\square$