Algorithmen & Komplexität

Angelika Steger
Institut für Theoretische Informatik

steger@inf.ethz.ch

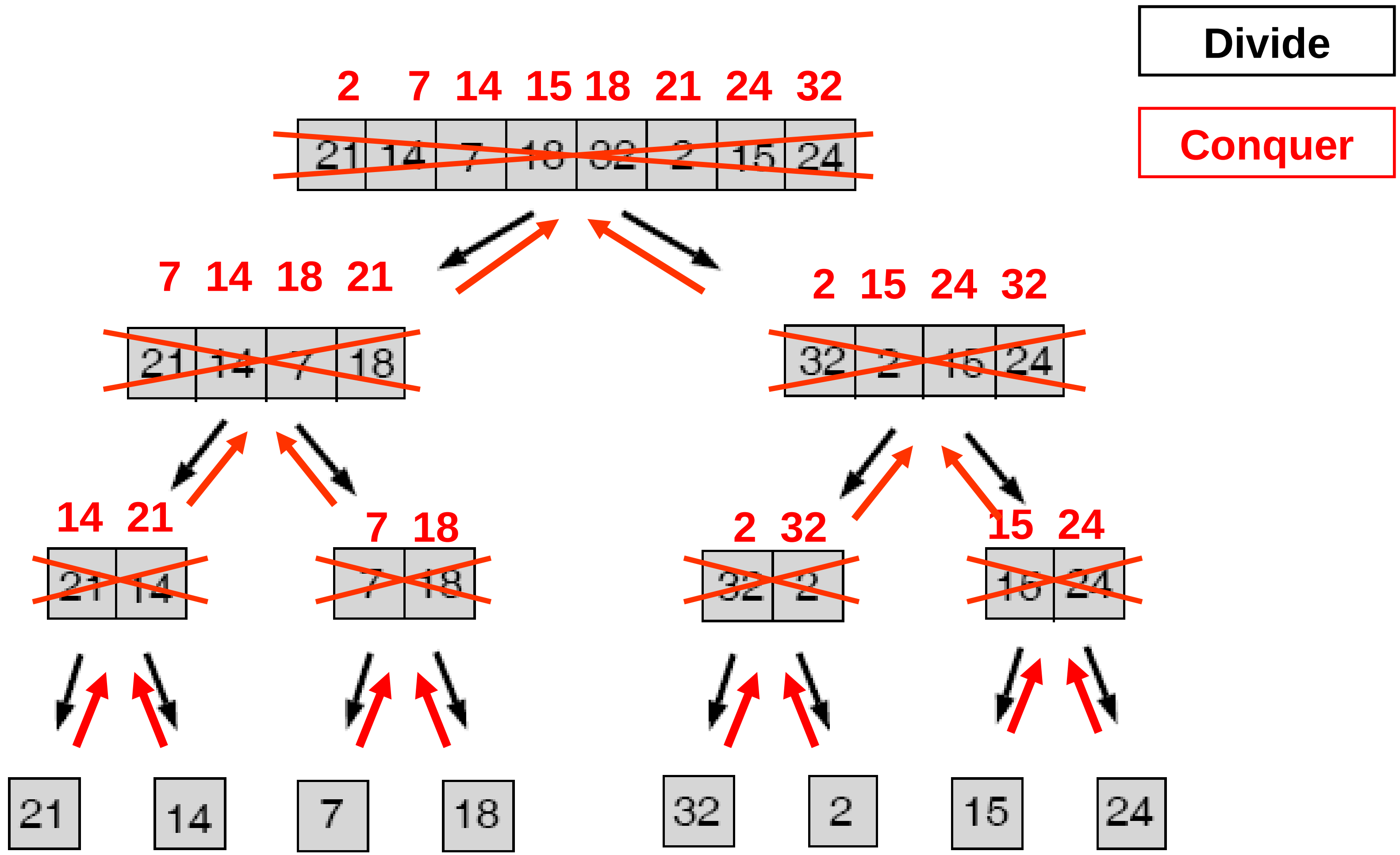
Kapitel 3:

Algorithmische Grundprinzipien

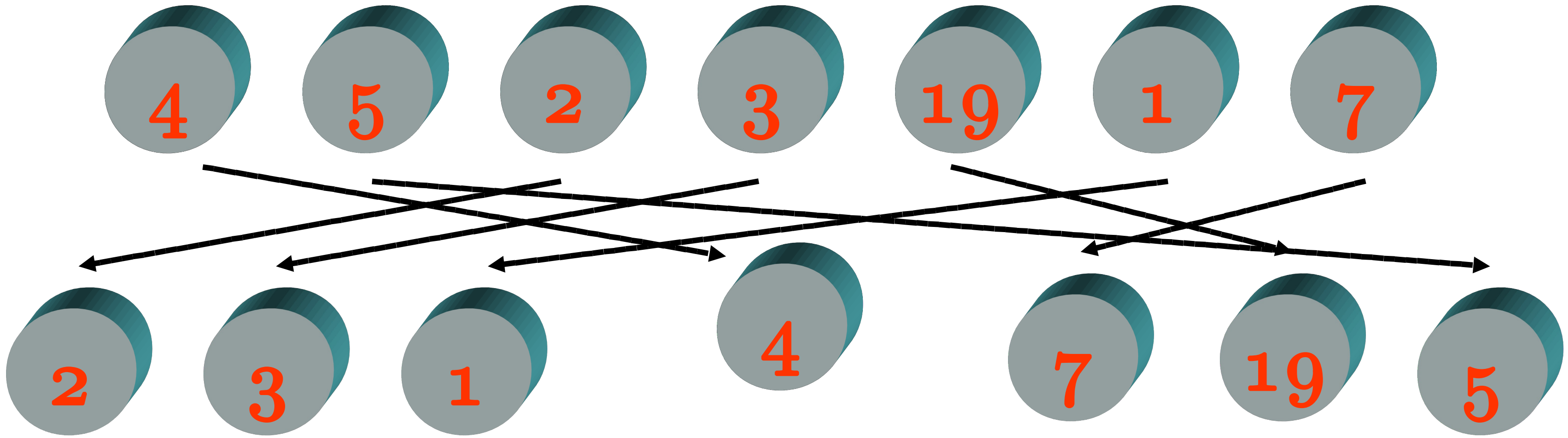
Kapitel 3.1:

Divide & Conquer Algorithmen

MergeSort - Beispiel



QuickSort - Beispiel



Zahlen kleiner als 4

Zahlen grösser als 4

Divide

Wähle erstes Element und teile übrige Elemente in „kleiner“ und „grösser“ auf

Conquer

>>>>>>>

Algorithmus von Strassen

A und B sind $2^k x 2^k$ Matrizen:

$$C = A \cdot B$$

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

Divide

Zurückführung auf 7 Multiplikationen von $2^{k-1} x 2^{k-1}$ Matrizen
Rekursive Anwendung.

Conquer

Berechne C (17 Additionen von $2^{k-1} x 2^{k-1}$ Matrizen)

3.1 Divide & Conquer Verfahren

MergeSort $C_n = C_{\lceil n/2 \rceil} + C_{\lfloor n/2 \rfloor} + n-1$

QuickSort

Binäre Suche $B_n = B_{\lceil n/2 \rceil} + 1$

Alg. von Strassen $T_n = 7\ T_{\lceil n/2 \rceil} + 15\ n^2$

Das Master-Theorem

erlaubt Gaussklammern

Satz 3.1 *(Master-Theorem) Seien $\alpha \geq 1$, $\beta > 1$ und $C \geq 0$ Konstanten und sei $f(n)$ eine positive Funktion. Weiter seien $c_1(n), \ldots, c_\alpha(n)$ Funktionen mit $|c_i(n)| \leq C$ für alle $1 \leq i \leq \alpha$ und $n \in \mathbb{N}$. Ist dann $T(n)$ eine Funktion mit $T(1) = 0$, die für $n \geq 1$ die Rekursionsgleichung*

$$T(n) = T(n/\beta + c_1(n)) + \cdots + T(n/\beta + c_\alpha(n)) + f(n)$$

erfüllt, dann gilt

$$T(n) = \begin{cases} \Theta(n^{\log_\beta \alpha}), & \text{falls } f(n) = O(n^{\log_\beta \alpha - \epsilon}) \text{ für ein } \epsilon > 0, \\ \Theta(f(n)\log n), & \text{falls } f(n) = \Theta(n^{\log_\beta \alpha}(\log n)^\delta) \text{ für ein } \delta \geq 0, \\ \Theta(f(n)), & \text{falls } f(n) = \Omega(n^{\log_\beta \alpha + \epsilon}) \text{ für ein } \epsilon > 0. \end{cases}$$

MergeSort: $C_n = C_{\lceil n/2 \rceil} + C_{\lfloor n/2 \rfloor} + n - 1$

also $\alpha = 2$, $\beta = 2$, $f(n) = n - 1$: 2. Zeile

Binäre Suche: $C_n = C_{\lceil n/2 \rceil} + 1$

also $\alpha = 1$, $\beta = 2$, $f(n) = 1$: 2. Zeile

Strassen: $T_n = 7T_{\lceil n/2 \rceil} + 15n^2$

also $\alpha = 7$, $\beta = 2$, $f(n) = 15n^2$: 1. Zeile

Kapitel 3:

Algorithmische Grundprinzipien

Kapitel 3.2:

Dynamische Programmierung

Kap. 3: Algorithmische Grundprinzipien

3.1 Divide & Conquer Verfahren „top-down“

- MergeSort
- QuickSort
- Binäre Suche
- Alg. von Strassen

3.2 Dynamische Programmierung „bottom-up“

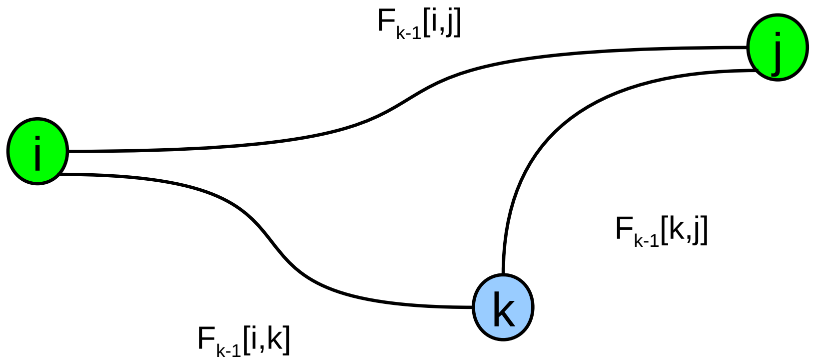
- Alg. von Floyd-Warshall
- Knapsack Problem

Algorithmus von Floyd-Warshall

Idee:

Berechne eine Folge F_k von Matrizen für k = 1, 2, …, n:

$F_k[i, j]$:= Länge eines kürzesten Pfades zwischen i und j, der nur innere Knoten aus der Menge {1, 2, …, k} hat.



k = 0: trivial

k-1 ➔ k:

$$F_k[i,j] = \min \{ F_{k-1}[i, j], F_{k-1}[i, k] + F_{k-1}[k, j] \}$$

Rucksackproblem

Gegeben:

Eine Kapazität $B \in \mathbf{N}$ des Rucksacks und

n Objekte mit Gewichten $w_1, \ldots, w_n \in \mathbf{N}$ und Profiten $p_1, \ldots, p_n \in \mathbf{N}$.

Gesucht:

Eine optimale Packung des Rucksacks, d.h.

eine Teilmenge $I \subseteq [n]$ mit $\Sigma_{i \varepsilon I}\, w_i \leq B$ und

$$\Sigma_{i \varepsilon I}\, p_i = \max \{ \Sigma_{i \varepsilon I'}\, p_i : I' \subseteq [n] \text{ mit } \Sigma_{i \varepsilon I'}\, w_i \leq B \}$$

Rucksackproblem

Idee:

Berechne zunächst Teillösungen, wobei nur die Objekte 1, 2, …, i berücksichtigt werden:

$f[i, t] :=$

minimal mögliches Gewicht des Rucksacks, wenn der Profit mindestens t betragen soll und nur die ersten i Objekte zur Verfügung stehen.

$i = 1$: $f[1, t] = w_1$ für $t \leq p_1$ und $f[1, t] = \infty$ sonst.

$i-1 \Rightarrow i$:

$$f[i, t] = \min \{ f[i-1, t], \; w_i + f[i-1, t-p_i] \}$$

Rucksackproblem

max. möglicher Profit

Algorithmus 3.1 KNAPSACK PACKING: Berechnung des Wertes

Eingabe: $n, w_1, \ldots, w_n, p_1, \ldots, p_n, B$

Ausgabe: $\max\{\sum_{i \in I} p_i \mid I \subseteq [n], \sum_{i \in I} w_i \leq B\}$

$p \leftarrow \sum_{i=1}^{n} p_i$;

for t **from** 1 **to** p_1 **do** $f[1,t] \leftarrow w_1$;

for t **from** $p_1 + 1$ **to** p **do** $f[1,t] \leftarrow \infty$;

for i **from** 2 **to** n **do begin**

  for t **from** 1 **to** p **do begin**

    if $t \leq p_i$ **then**

      $f[i,t] \leftarrow \min\{f[i-1,t], w_i\}$;

    else

      $f[i,t] \leftarrow \min\{f[i-1,t], w_i + f[i-1, t-p_i]\}$;

  end

end

return $\max\{t \mid f[n,t] \leq B\}$;

max. möglicher Profit mit Gesamtgewicht ≤ B

Kapitel 3:

Algorithmische Grundprinzipien

Kapitel 3.3:

Greedy-Algorithmen

Kap. 3: Algorithmische Grundprinzipien

3.1 Divide & Conquer Verfahren „top-down“

- MergeSort
- QuickSort
- Binäre Suche
- Alg. von Strassen

3.2 Dynamische Programmierung „bottom-up“

- Alg. von Floyd-Warshall
- Knapsack Problem

3.3 Greedy-Algorithmen

- Alg. von Kruskal
- (Alg. von Prim, Djikstra)

3.3. Greedy-Algorithmen

Definition: Ein **Matroid** $M=(S,U)$ besteht aus einer endlichen Menge S und einer Familie von unabhängigen Mengen $U \subseteq Power(S)$ von Teilmengen von S, so dass die folgenden drei Bedingungen erfüllt sind:

(M1) $\emptyset \in U$

(M2) $A \in U$ und $B \subseteq A \quad \Rightarrow \quad B \in U$

(M3) $A,B \in U$ und $|B| > |A| \quad \Rightarrow \quad \exists\, b \in B \setminus A$ mit $A \cup \{b\} \in U$

Greedy-Alg. für Matroide

Algorithmus 3.2 Greedy-Algorithmus für Matroide

Eingabe: Matroid $\mathcal{M} = (S, \mathcal{U})$, Gewichtsfunktion $w : S \to \mathbb{R}$.

Ausgabe: Basis A mit minimalem Gewicht: $w(A) = \min\{w(B) \mid B \text{ Basis}\}$.

$A := \emptyset$;

while A ist keine Basis von $\mathcal{M}$ **do begin**

$X := \{x \in S \setminus A \mid A \cup \{x\} \in \mathcal{U}\}$;

wähle $x_0 \in X$, so dass $w(x_0) = \min_{x \in X} w(x)$;

$A := A \cup \{x_0\}$;

end