

18. Divide & Conquer

Generische Optimierungsverfahren:

- Systematische Suche
 - lass nichts aus
- Divide and Conquer
 - löse das Ganze in Teilen
- Dynamische Programmierung
 - mache nie etwas zweimal
- Greedy Verfahren
 - schau niemals zurück
- Lokale Suche
 - denke global, handle lokal

Systematische Suche

Prinzip: durchsuche **gesamten** Lösungsraum

Auch bekannt als „Brute Force“

Vorteil: sehr einfach zu implementieren

Nachteil: sehr zeitaufwendig und sollte daher nur für kleine Instanzen verwendet werden

Systematische Suche

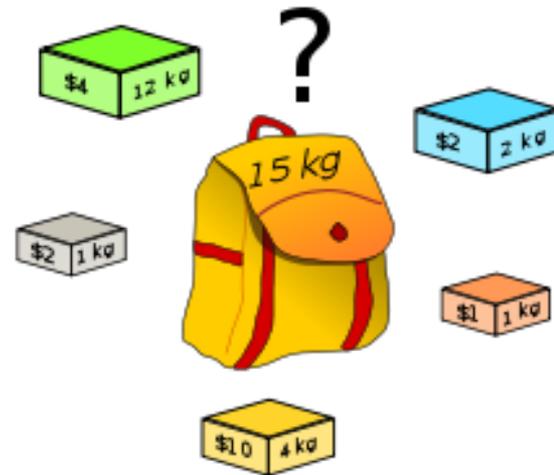
Beispiele:

- Suche in unsortierter Liste
- Suche über Broadcasting in unstrukturierten verteilten Systemen (Peer-to-Peer Systeme)
- Rucksackproblem (siehe nächste Folie)

Systematische Suche

Rucksackproblem:

- **Eingabe:** n Objekte mit Gewichten w_1, \dots, w_n und Werten v_1, \dots, v_n und Rucksack mit Kapazität W
- **Ausgabe:** Objektmenge M maximalen Wertes, die in Rucksack passt



Systematische Suche

Lösung zum Rucksackproblem:

Probiere **alle Teilmengen** von Objekten aus und merke die Menge **M** von Objekten mit $\sum_{i \in M} w_i \leq W$, die bisher den maximalen Wert hatte

Aufwand: $O(2^n)$, da es 2^n Möglichkeiten gibt, Teilmengen aus einer **n**-elementigen Menge zu bilden.

Divide & Conquer

Teile & Herrsche:

- Problem in Teilprobleme aufteilen
- Teilprobleme rekursiv lösen
- Lösung aus Teillösungen zusammensetzen

Probleme:

- Wie setzt man zusammen?
[erfordert algorithmisches Geschick und Übung]
- Laufzeitanalyse (Auflösen der Rekursion)
[ist normalerweise nach Standardschema; erfordert ebenfalls Übung]

Divide & Conquer

Beispiele:

- Mergesort
- Quicksort
- Binary Search
- Arithmische Operationen wie Multiplikation großer Zahlen oder **Matrixmultiplikation**
- **Selektion**
- **Nächstes-Paar-Problem**

Matrix Multiplikation

$$\begin{pmatrix} 3 & 7 & 5 & 4 \\ 0 & 3 & 2 & 4 \\ 10 & 2 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 2 & 1 & 1 & 0 \\ 0 & 0 & 1 & 3 \\ 3 & 1 & 1 & 0 \\ 2 & 3 & 2 & 2 \end{pmatrix} = \begin{pmatrix} & & & \\ & & & \\ & & & \\ & & & \end{pmatrix}$$

Matrix Multiplikation

$$A=(a_{ij})_{1 \leq i,j \leq n}$$

$$B=(b_{ij})_{1 \leq i,j \leq n}$$

$$C=(c_{ij})_{1 \leq i,j \leq n}$$

$$\begin{pmatrix} 3 & 7 & 5 & 4 \\ 0 & 3 & 2 & 4 \\ 10 & 2 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 2 & 1 & 1 & 0 \\ 0 & 0 & 1 & 3 \\ 3 & 1 & 1 & 0 \\ 2 & 3 & 2 & 2 \end{pmatrix} = \begin{pmatrix} 29 & 20 & 23 & 29 \\ 14 & 14 & \dots \\ \dots \\ \dots \end{pmatrix}$$

$$c_{ij} = \sum_{k=1}^n a_{ik} \cdot b_{kj}$$

Matrix Multiplikation

Teile & Herrsche:

- Problem: Berechne das Produkt zweier $n \times n$ Matrizen
- Eingabe: Matrizen X, Y
- Ausgabe: Matrix $Z = X \cdot Y$

$$X = \begin{pmatrix} x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} \\ x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} \\ x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} \\ x_{4,1} & x_{4,2} & x_{4,3} & x_{4,4} \end{pmatrix}, \quad Y = \begin{pmatrix} y_{1,1} & y_{1,2} & y_{1,3} & y_{1,4} \\ y_{2,1} & y_{2,2} & y_{2,3} & y_{2,4} \\ y_{3,1} & y_{3,2} & y_{3,3} & y_{3,4} \\ y_{4,1} & y_{4,2} & y_{4,3} & y_{4,4} \end{pmatrix}$$

Matrix Multiplikation

MatrixMultiplikation(Array X, Y, n)

1. **new** array Z[1,...,n][1,...,n]
2. **for** i ← 1 **to** n **do**
3. **for** j ← 1 **to** n **do**
4. Z[i][j] ← 0
5. **for** k ← 1 **to** n **do**
6. Z[i][j] ← Z[i][j] + X[i][k] · Y[k][j]
7. **return** Z

Matrix Multiplikation

MatrixMultiplikation(Array X, Y, n)

Laufzeit:

1. **new** array Z[1,...,n][1,...,n]
2. **for** i ← 1 **to** n **do**
3. **for** j ← 1 **to** n **do**
4. Z[i][j] ← 0
5. **for** k ← 1 **to** n **do**
6. Z[i][j] ← Z[i][j] + X[i][k] · Y[k][j]
7. **return** Z

$\Theta(n^2)$

Matrix Multiplikation

MatrixMultiplikation(Array X, Y, n)

1. **new** array Z[1,...,n][1,...,n]
2. **for** i ← 1 **to** n **do**
3. **for** j ← 1 **to** n **do**
4. Z[i][j] ← 0
5. **for** k ← 1 **to** n **do**
6. Z[i][j] ← Z[i][j] + X[i][k] · Y[k][j]
7. **return** Z

Laufzeit:

$\Theta(n^2)$

$\Theta(n)$

Matrix Multiplikation

MatrixMultiplikation(Array X, Y, n)

1. **new** array Z[1,...,n][1,...,n]
2. **for** i ← 1 **to** n **do**
3. **for** j ← 1 **to** n **do**
4. Z[i][j] ← 0
5. **for** k ← 1 **to** n **do**
6. Z[i][j] ← Z[i][j] + X[i][k] · Y[k][j]
7. **return** Z

Laufzeit:

$\Theta(n^2)$

$\Theta(n)$

$\Theta(n^2)$

Matrix Multiplikation

MatrixMultiplikation(Array X, Y, n)

1. **new** array Z[1,...,n][1,...,n]
2. **for** i ← 1 **to** n **do**
3. **for** j ← 1 **to** n **do**
4. Z[i][j] ← 0
5. **for** k ← 1 **to** n **do**
6. Z[i][j] ← Z[i][j] + X[i][k] · Y[k][j]
7. **return** Z

Laufzeit:

$\Theta(n^2)$

$\Theta(n)$

$\Theta(n^2)$

$\Theta(n^2)$

Matrix Multiplikation

MatrixMultiplikation(Array X, Y, n)

1. **new** array Z[1,...,n][1,...,n]
2. **for** i ← 1 **to** n **do**
3. **for** j ← 1 **to** n **do**
4. Z[i][j] ← 0
5. **for** k ← 1 **to** n **do**
6. Z[i][j] ← Z[i][j] + X[i][k] · Y[k][j]
7. **return** Z

Laufzeit:

$\Theta(n^2)$

$\Theta(n)$

$\Theta(n^2)$

$\Theta(n^2)$

$\Theta(n^3)$

Matrix Multiplikation

MatrixMultiplikation(Array X, Y, n)

1. **new** array Z[1,...,n][1,...,n]

2. **for** i ← 1 **to** n **do**

3. **for** j ← 1 **to** n **do**

4. Z[i][j] ← 0

5. **for** k ← 1 **to** n **do**

6. Z[i][j] ← Z[i][j] + X[i][k] · Y[k][j]

7. **return** Z

Laufzeit:

$\Theta(n^2)$

$\Theta(n)$

$\Theta(n^2)$

$\Theta(n^2)$

$\Theta(n^3)$

$\Theta(n^3)$

Matrix Multiplikation

MatrixMultiplikation(Array X, Y, n)

1. **new** array Z[1,...,n][1,...,n]
2. **for** i ← 1 **to** n **do**
3. **for** j ← 1 **to** n **do**
4. Z[i][j] ← 0
5. **for** k ← 1 **to** n **do**
6. Z[i][j] ← Z[i][j] + X[i][k] · Y[k][j]
7. **return** Z

Laufzeit:

$\Theta(n^2)$

$\Theta(n)$

$\Theta(n^2)$

$\Theta(n^2)$

$\Theta(n^3)$

$\Theta(n^3)$

$\Theta(1)$

Matrix Multiplikation

MatrixMultiplikation(Array X, Y, n)

1. **new** array Z[1,...,n][1,...,n]
2. **for** i ← 1 **to** n **do**
3. **for** j ← 1 **to** n **do**
4. Z[i][j] ← 0
5. **for** k ← 1 **to** n **do**
6. Z[i][j] ← Z[i][j] + X[i][k] · Y[k][j]
7. **return** Z

Laufzeit:

$\Theta(n^2)$

$\Theta(n)$

$\Theta(n^2)$

$\Theta(n^2)$

$\Theta(n^3)$

$\Theta(n^3)$

$\Theta(1)$

$\Theta(n^3)$

Matrix Multiplikation

Teile und Herrsche:

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \times \begin{pmatrix} E & F \\ G & H \end{pmatrix} = \begin{pmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{pmatrix}$$

Aufwand:

- 8 Multiplikationen von $n/2 \times n/2$ Matrizen
- 4 Additionen von $n/2 \times n/2$ Matrizen

Matrix Multiplikation

Teile und Herrsche:

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \times \begin{pmatrix} E & F \\ G & H \end{pmatrix} = \begin{pmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{pmatrix}$$

Aufwand:

- 8 Multiplikationen von $n/2 \times n/2$ Matrizen
- 4 Additionen von $n/2 \times n/2$ Matrizen

Laufzeit:

- $T(n) = 8 \cdot T(n/2) + \Theta(n^2)$

Matrix Multiplikation

Laufzeit:

- $T(n) = 8 \cdot T(n/2) + k \cdot n^2$
 \uparrow \uparrow \uparrow
 a b f(n)

Master Theorem:

- $f(n) = k \cdot n^2$
- $a=8, b=2$

Matrix Multiplikation

Laufzeit:

- $T(n) = 8 \cdot T(n/2) + k \cdot n^2$
 \uparrow \uparrow \uparrow
 a b f(n)

Master Theorem:

- $f(n) = k \cdot n^2$
- $a=8, b=2$
- **Fall 1:** Laufzeit $\Theta(n^{\log_b a}) = \Theta(n^3)$

Matrix Multiplikation

Laufzeit:

- $T(n) = 8 \cdot T(n/2) + k \cdot n^2$
 \uparrow \uparrow \uparrow
 a b f(n)

Master Theorem:

- $f(n) = k \cdot n^2$
- $a=8, b=2$
- **Fall 1:** Laufzeit $\Theta(n^{\log_b a}) = \Theta(n^3)$
- Formaler Beweis durch Induktion!!

Matrix Multiplikation

Laufzeit:

- $T(n) = 8 \cdot T(n/2) + k \cdot n^2$
 \uparrow \uparrow \uparrow
 a b f(n)

Master Theorem:

- $f(n) = k \cdot n^2$
- $a=8, b=2$
- **Fall 1:** Laufzeit $\Theta(n^{\log_b a}) = \Theta(n^3)$
- Formaler Beweis durch Induktion
- **Nicht besser als einfacher Algorithmus**

Matrix Multiplikation

Teile und Herrsche (Algorithmus von Strassen):

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \times \begin{pmatrix} E & F \\ G & H \end{pmatrix} = \begin{pmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{pmatrix}$$

Trick:

$$P_1 = A \cdot (F - H) \quad P_5 = (A + D) \cdot (E + H)$$

$$P_2 = (A + B) \cdot H \quad P_6 = (B - D) \cdot (G + H)$$

$$P_3 = (C + D) \cdot E \quad P_7 = (A - C) \cdot (E + F)$$

$$P_4 = D \cdot (G - E)$$

Matrix Multiplikation

Teile und Herrsche (Algorithmus von Strassen):

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \times \begin{pmatrix} E & F \\ G & H \end{pmatrix} = \begin{pmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{pmatrix}$$

Trick:

$$P_1 = A \cdot (F - H)$$

$$P_2 = (A + B) \cdot H$$

$$P_3 = (C + D) \cdot E$$

$$P_4 = D \cdot (G - E)$$

$$P_5 = (A + D) \cdot (E + H)$$

$$P_6 = (B - D) \cdot (G + H)$$

$$P_7 = (A - C) \cdot (E + F)$$

$$AE + BG = P_5 + P_4 - P_2 + P_6$$

$$AF + BH = P_1 + P_2$$

$$CE + DG = P_3 + P_4$$

$$CF + DH = P_5 + P_1 - P_3 - P_7$$

Matrix Multiplikation

Teile und Herrsche:

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \times \begin{pmatrix} E & F \\ G & H \end{pmatrix} = \begin{pmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{pmatrix}$$

Trick:

$$P_1 = A \cdot (F - H)$$

$$P_5 = (A + D) \cdot (E + H)$$

$$P_2 = (A + B) \cdot H$$

$$P_6 = (B - D) \cdot (G + H)$$

$$P_3 = (C + D) \cdot E$$

$$P_7 = (A - C) \cdot (E + F)$$

$$P_4 = D \cdot (G - E)$$

$$AE + BG = P_5 + P_4 - P_2 + P_6$$

$$AF + BH = P_1 + P_2$$

$$CE + DG = P_3 + P_4$$

$$CF + DH = P_5 + P_1 - P_3 - P_7$$

7 Multiplikationen!!!

Matrix Multiplikation

Laufzeit:

- $T(n) = 7 \cdot T(n/2) + k \cdot n^2$
 \uparrow \uparrow \uparrow
 a b f(n)

Master Theorem:

- $f(n) = k \cdot n^2$

Matrix Multiplikation

Laufzeit:

- $T(n) = 7 \cdot T(n/2) + k \cdot n^2$
 \uparrow \uparrow \uparrow
 a b f(n)

Master Theorem:

- $f(n) = k \cdot n^2$
- $a=7, b=2$
- **Fall 1:** Laufzeit $\Theta(n^{\log_b a})$

Matrix Multiplikation

Laufzeit:

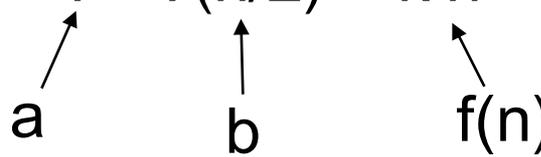
- $T(n) = 7 \cdot T(n/2) + k \cdot n^2$
 \uparrow \uparrow \uparrow
 a b f(n)

Master Theorem:

- $f(n) = k \cdot n^2$
- $a=7, b=2$
- **Fall 1:** Laufzeit $\Theta(n^{\log_b a}) = \Theta(n^{\log_2 7})$

Matrix Multiplikation

Laufzeit:

- $T(n) = 7 \cdot T(n/2) + k \cdot n^2$


The diagram shows the recurrence relation $T(n) = 7 \cdot T(n/2) + k \cdot n^2$. Three arrows point from labels below to parts of the equation: an arrow from 'a' points to the constant 7, an arrow from 'b' points to the term $T(n/2)$, and an arrow from 'f(n)' points to the term $k \cdot n^2$.

Master Theorem:

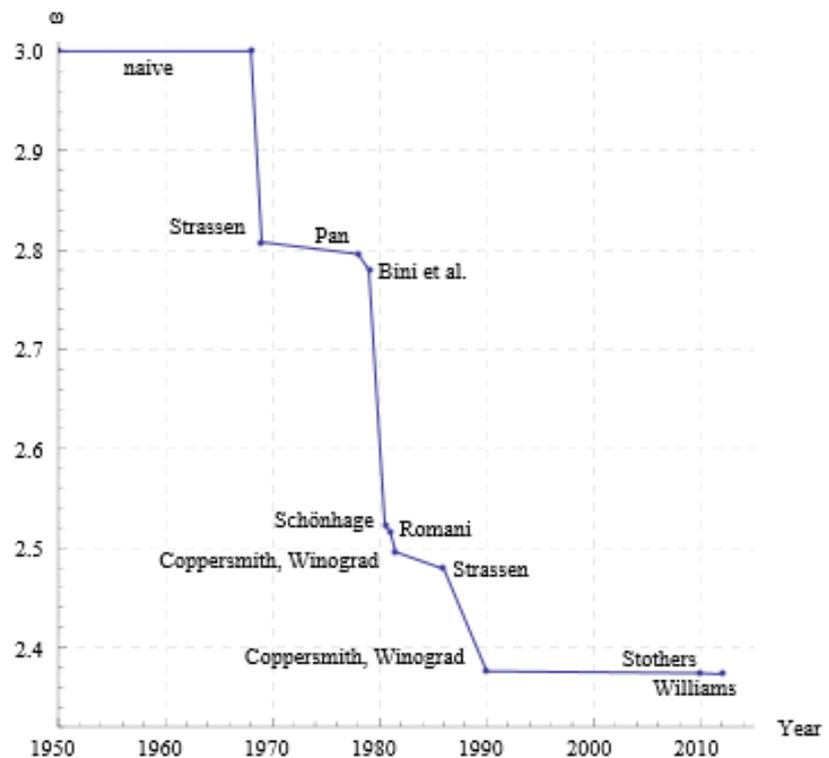
- $f(n) = k \cdot n^2$
- $a=7, b=2$
- **Fall 1:** Laufzeit $\Theta(n^{\log_b a}) = \Theta(n^{\log_2 7}) = \Theta(n^{2,81})$
- Verbesserter Algorithmus!
(Erheblicher Unterschied für große Eingaben)

Matrix Multiplikation

Satz 18.1

Das Produkt zweier $n \times n$ Matrizen kann in $\Theta(n^{2,81})$ Laufzeit berechnet werden.

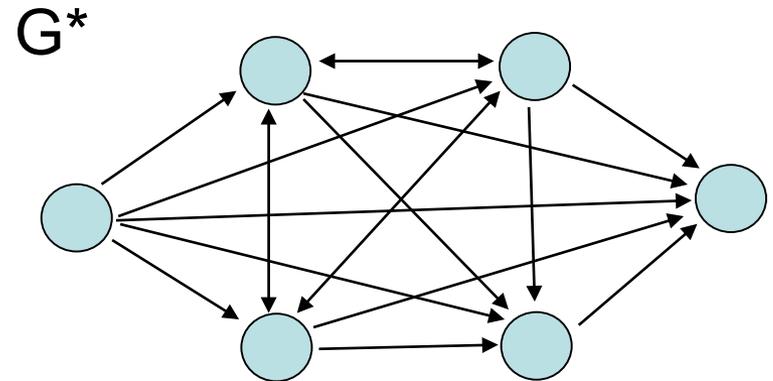
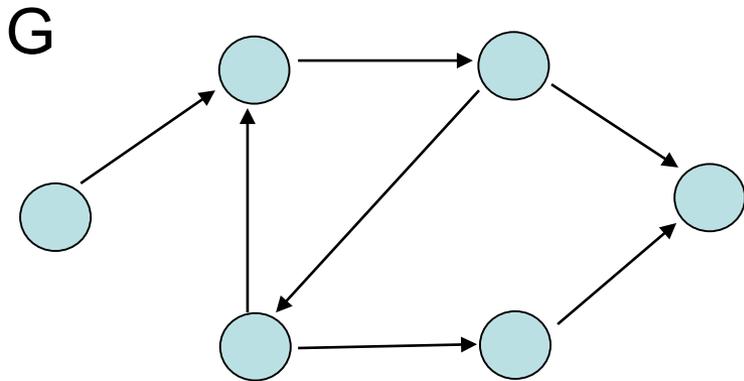
Seitdem verschiedene Verbesserungen:



Matrix Multiplikation

Das transitive Hülle Problem:

- Gegeben sei ein gerichteter, ungewichteter Graph $G=(V,E)$
- Gesucht: Die transitive Hülle $G^*=(V,E^*)$ von G , wobei $E^*=\{(u,v): \text{es gibt Weg von } u \text{ nach } v \text{ in } G\}$



Das transitive Hülle Problem:

- Gegeben sei ein gerichteter, ungewichteter Graph $G=(V,E)$
- Gesucht: Die transitive Hülle $G^*=(V,E^*)$ von G , wobei $E^*=\{(u,v): \text{es gibt Weg von } u \text{ nach } v \text{ in } G\}$

Transitive Hülle:

- In $O(|V|^2+|V| |E|)$ Zeit mit Breiten- oder Tiefensuche von jedem Knoten
- Geht das auch schneller?

Matrix Multiplikation

Graphen und Matrixmultiplikation:

- Sei A die $n \times n$ -Adjazenzmatrix von Graph G mit Knotenmenge $\{1, \dots, n\}$
- Was ist $A \cdot A$?

Behauptung 15.2:

Sei $Z = A \cdot A$. Dann gilt $z_{ij} > 0$, g.d.w. es in G einen Pfad der Länge 2 von Knoten i zu Knoten j gibt.

Matrix Multiplikation

Behauptung 15.2:

Sei $Z' = A \cdot A + A$. Dann gilt, dass $z'_{ij} > 0$, g.d.w. es einen Weg der Länge 1 oder 2 von Knoten i zu Knoten j gibt.

Konstruiere Matrix B mit:

- $b_{ij} = 1 \iff z'_{ij} > 0$

Behauptung 15.3:

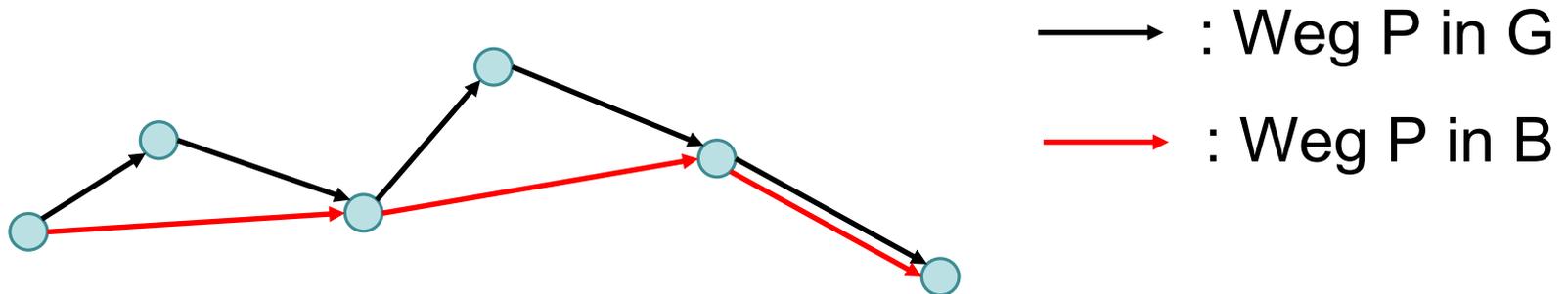
Matrix B hat einen Weg von Knoten i nach j , g.d.w. Matrix A einen solchen Weg hat.

Matrix Multiplikation

Behauptung 15.4:

Sei P ein Weg der Länge $k > 1$ in G . Dann hat P maximal Länge $(2/3)k$ in dem von Matrix B beschriebenen Graph G' .

Beweis:



Matrix Multiplikation

Behauptung 15.4:

Sei P ein Weg der Länge $k > 1$ in G . Dann hat P maximal Länge $(2/3)k$ in dem von Matrix B beschriebenen Graph G' .

Konsequenz aus Beh. 15.3 und 15.4:

- Wenn wir die Berechnung von $B \log_{3/2} n$ mal iterieren, haben wir die transitive Hülle berechnet

Matrix Multiplikation

TransitiveHülle(A)

1. **for** $i \leftarrow 1$ **to** $\log_{3/2} n$ **do**
2. $Z' \leftarrow A A+A$
3. **for** $i \leftarrow 1$ **to** n **do**
4. **for** $j \leftarrow 1$ **to** n **do**
5. **if** $z'_{ij} > 0$ **then** $b_{ij} \leftarrow 1$ **else** $b_{ij} \leftarrow 0$
6. $A \leftarrow B$
7. **return** A

Matrix Multiplikation

Satz 15.5:

Der Algorithmus TransitiveHülle berechnet die transitive Hülle eines Graphen G in $O(M(n) \log n)$ Zeit, wobei $M(n)$ die Laufzeit zur Matrixmultiplikation bezeichnet.

Divide & Conquer

Beispiele:

- Mergesort
- Quicksort
- Binary Search
- Arithmische Operationen wie Multiplikation großer Zahlen oder Matrixmultiplikation
- **Selektion**
- Nächstes-Paar-Problem

Selektion

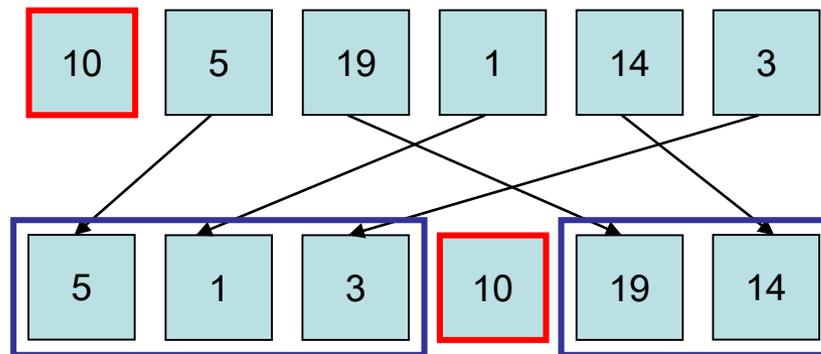
Problem: finde k -kleinstes Element in einer Folge von n Elementen

Lösung: sortiere Elemente (z.B. Mergesort), gib k -tes Element aus \rightarrow Zeit $O(n \log n)$

Geht das auch schneller??

Selektion

Ansatz: verfahren ähnlich zu Quicksort



- j : Position des Pivotelements
- $k < j$: mach mit linker Teilfolge weiter
- $k > j$: mach mit rechter Teilfolge weiter

Selektion

Quickselect(A, l, r, k)

▷ $A[l..r]$: Restfeld, k : k -kleinstes Element, $l \leq k \leq r$

if $r=l$ then return $a[l]$

$i \leftarrow$ Randomized-Partition(A, l, r) ▷ siehe Kapitel 6

if $k < i$ then $x \leftarrow$ Quickselect($A, l, i-1, k$)

if $k > i$ then $x \leftarrow$ Quickselect($A, i+1, r, k$)

if $k = i$ then $x \leftarrow a[k]$

return x

Zum Vergleich Quicksort(A, l, r):

if $l < r$ then

$i \leftarrow$ Partition(A, l, r)

Quicksort($A, l, i-1$)

Quicksort($A, i+1, r$)

Quickselect

- $C(n)$: erwartete Anzahl Vergleiche

Satz 18.2: $C(n)=O(n)$

Beweis:

- Pivot ist **gut**: keine der Teilfolgen länger als $2n/3$
- Sei $p=\Pr[\text{Pivot ist gut}]$



- $p=1/3$

Quickselect

- Pivot **gut**: Restaufwand $\leq C(2n/3)$
- Pivot **schlecht**: Restaufwand $\leq C(n)$

$$C(n) \leq n + p \cdot C(2n/3) + (1-p) \cdot C(n)$$

$$\Rightarrow C(n) \leq n/p + C(2n/3)$$

$$\leq 3n + C(2n/3) \leq 3(n + 2n/3 + 4n/9 + \dots)$$

$$\leq 3n \sum_{i \geq 0} (2/3)^i$$

$$\leq 3n / (1 - 2/3) = 9n$$

BFPRT-Algorithmus

Gibt es auch einen deterministischen
Selektionsalgorithmus mit linearer Laufzeit?

Ja, den **BFPRT-Algorithmus** (benannt nach den Erfindern Blum, Floyd, Pratt, Rivest und Tarjan).

BFPRT-Algorithmus

- Sei m eine ungerade Zahl ($5 \leq m \leq 21$).
- Betrachte die Zahlenmenge $S = \{a_1, \dots, a_n\}$.
- Gesucht: k -kleinste Zahl in S

Algorithmus BFPRT(S, k):

1. if $|S| < m$ then berechne k -kleinstes Element x in S , return x
2. Teile S in $\lceil n/m \rceil$ Blöcke auf, davon $\lfloor n/m \rfloor$ mit m Elementen
3. Sortiere jeden dieser Blöcke (z.B. mit Insertionsort)
4. $S' :=$ Menge der $\lceil n/m \rceil$ Mediane der Blöcke.
5. $m \leftarrow \text{BFPRT}(S', \lceil |S'|/2 \rceil)$ ▷ berechnet Median der Mediane
6. $S_1 \leftarrow \{x \in S \mid x < m\}$; $S_2 \leftarrow \{x \in S \mid x > m\}$ ▷ m : wie Pivot in Quicksort
7. if $k \leq |S_1|$ then return BFPRT(S_1, k)
8. if $k > |S_1| + 1$ then return BFPRT($S_2, k - |S_1| - 1$)
9. return m

BFPRT-Algorithmus

- Sei m eine ungerade Zahl ($5 \leq m \leq 21$).
- Betrachte die Zahlenmenge $S = \{a_1, \dots, a_n\}$.
- Gesucht: k -kleinste Zahl in S

Algorithmus BFPRT(S, k):

1. if $|S| < m$ then berechne k -sortierten Folge
2. Teile S in $\lceil n/m \rceil$ Blöcke auf, $\lceil n/m \rceil$ mit m Elementen
3. Sortiere jeden dieser Blöcke (z.B. mit Insertionsort)
4. $S' :=$ Menge der $\lceil n/m \rceil$ Mediane der Blöcke.
5. $m \leftarrow \text{BFPRT}(S', \lceil |S'|/2 \rceil)$ \triangleright berechnet Median der Mediane
6. $S_1 \leftarrow \{x \in S \mid x < m\}$; $S_2 \leftarrow \{x \in S \mid x > m\}$ \triangleright m : wie Pivot in Quicksort
7. if $k \leq |S_1|$ then return BFPRT(S_1, k)
8. if $k > |S_1| + 1$ then return BFPRT($S_2, k - |S_1| - 1$)
9. return m

Median: Wert in der Mitte einer sortierten Folge

BFPRT-Algorithmus

Laufzeit $T(n)$ des BFPRT-Algorithmus:

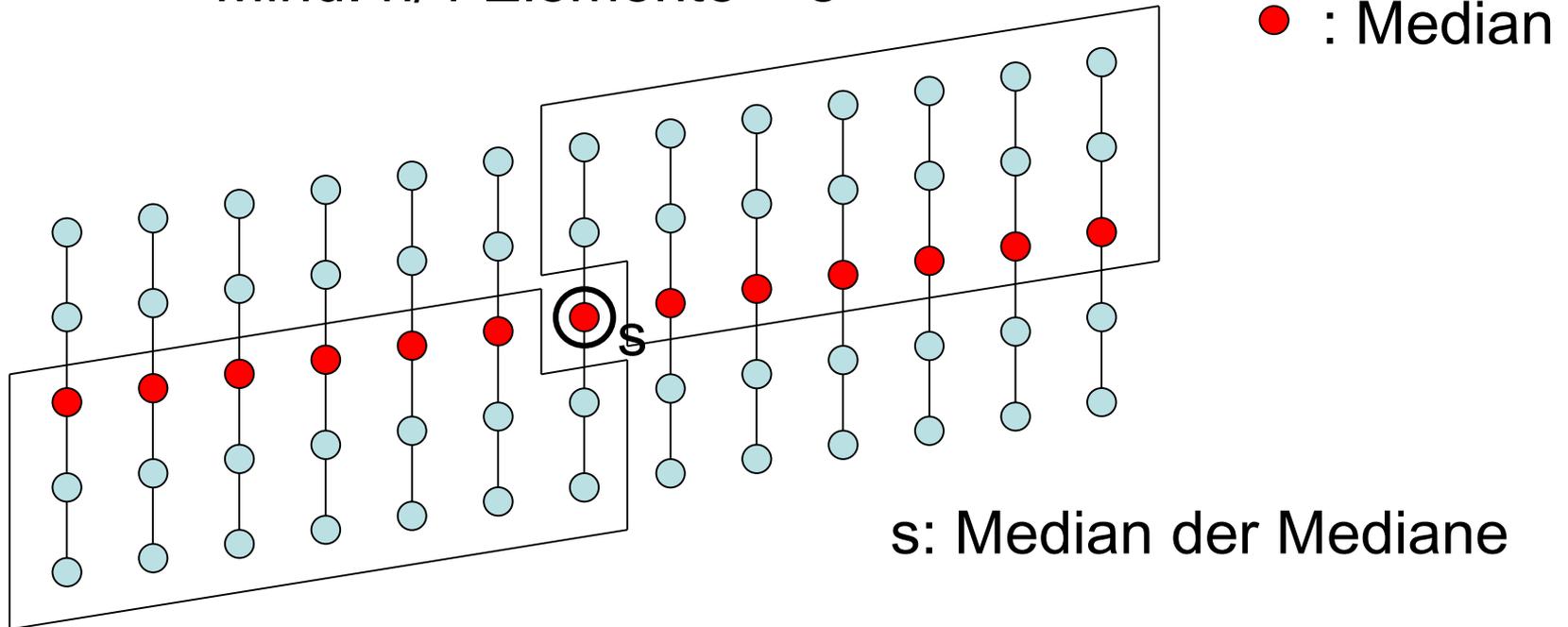
- Schritte 1-3: $O(n)$
- Schritt 4: $T(\lceil n/m \rceil)$
- Schritt 5: $O(n)$
- Schritt 6 bzw. 7: ???

Lemma 18.3: Schritt 6/7 ruft BFPRT mit maximal $\lfloor (3/4)n \rfloor$ Elementen auf

BFPRT-Algorithmus

Beweis: O.B.d.A. sei $m=5$

Mind. $n/4$ Elemente $> s$



s: Median der Mediane

Mind. $n/4$ Elemente $< s$

BFPRT-Algorithmus

Laufzeit für $m=5$:

$$T(n) \leq T(\lfloor (3/4)n \rfloor) + T(\lceil n/5 \rceil) + c \cdot n$$

für eine Konstante c .

Satz 18.4: $T(n) \leq d \cdot n$ für eine Konstante d .

Beweis: Übung.

Divide & Conquer

Beispiele:

- Mergesort
- Quicksort
- Binary Search
- Arithmische Operationen wie Multiplikation großer Zahlen oder Matrixmultiplikation
- Selektion
- **Nächstes-Paar-Problem**

Nächstes-Paar-Problem

Nächstes-Paar-Problem:

- **Eingabe:** Menge S von n Punkten $P_1=(x_1,y_1), \dots, P_n=(x_n,y_n)$ im 2-dimensionalen Euklidischen Raum
- **Ausgabe:** Punktpaar mit kürzester Distanz

Annahme: n ist Zweierpotenz

Nächstes-Paar-Problem

Algo für Nächstes-Paar-Problem:

- Sortiere Punkte gemäß x -Koordinate (z.B. Mergesort, Zeit $O(n \log n)$)
- Löse danach Nächstes-Paar-Problem rekursiv durch Algo **ClosestPair**

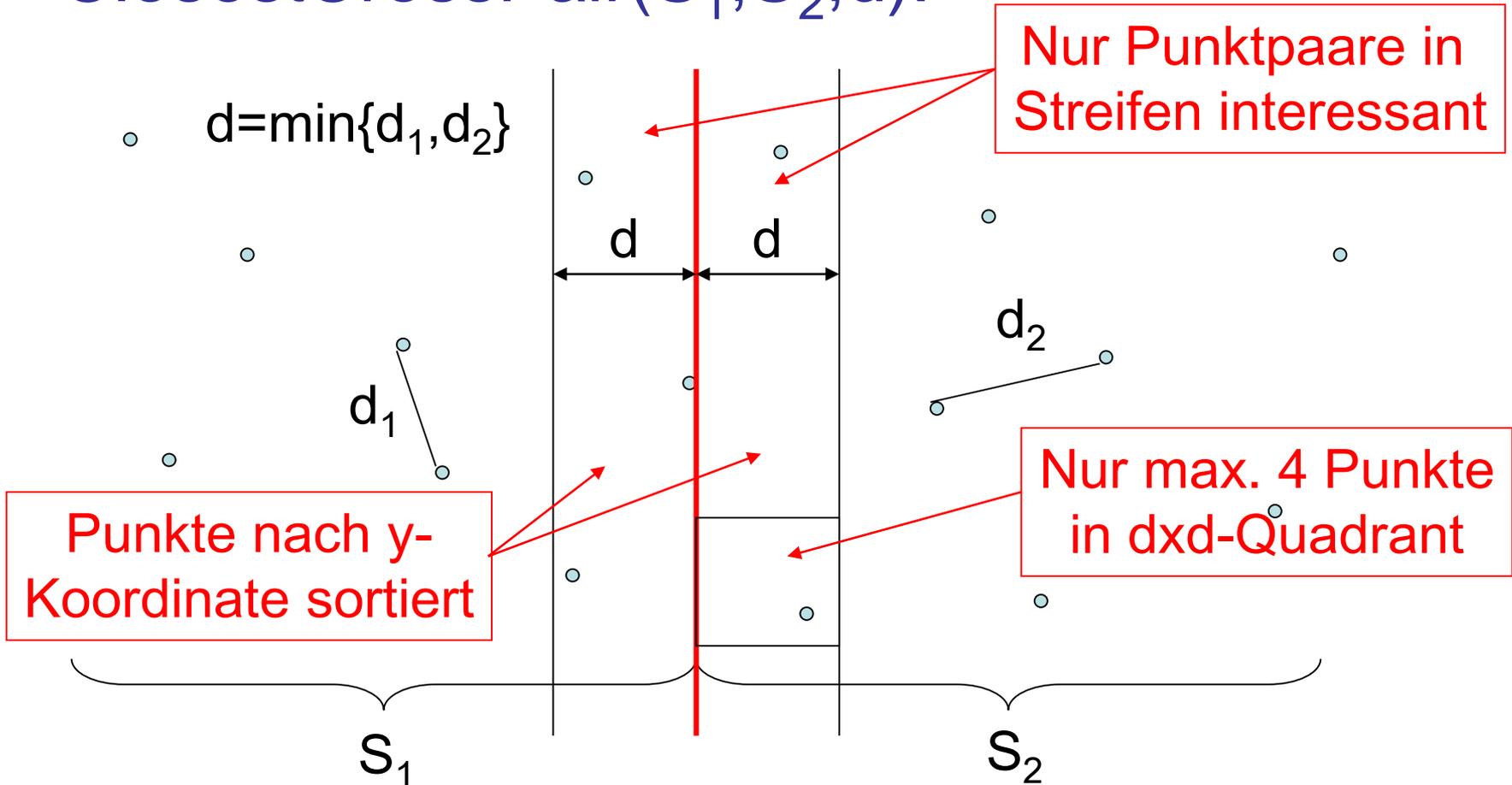
Nächstes-Paar-Problem

Algo ClosestPair(S):

- Eingabe: nach x -Koordinate sortierte Punktmenge S
- $|S|=2$: sortiere S gemäß y -Koordinate und gib Distanz zwischen den Punkten in S zurück
- $|S|>2$:
 - teile S in der Mitte (Position $n/2$) in S_1 und S_2
 - $d_1 := \text{ClosestPair}(S_1)$; $d_2 := \text{ClosestPair}(S_2)$
 - $d := \min\{\text{ClosestCrossPair}(S_1, S_2, \min(d_1, d_2)), d_1, d_2\}$
 - Führe $\text{Merge}(S_1, S_2)$ durch, so dass S am Ende nach y -Koordinate sortiert ist (S_1, S_2 bereits nach y sortiert)
 - gib d zurück

Nächstes-Paar-Problem

ClosestCrossPair(S_1, S_2, d):



Nächstes-Paar-Problem

ClosestCrossPair:

- Durchlaufe die (nach der y -Koordinate sortierten) Punkte in S_1 und S_2 von oben nach unten (wie in $\text{Merge}(S_1, S_2)$) und merke die Mengen M_1 und M_2 der 8 zuletzt gesehenen Punkte in S_1 und S_2 im d -Streifen
- Bei jedem neuen Knoten in M_1 , berechne Distanzen zu Knoten in M_2 , und bei jedem neuen Knoten in M_2 , berechne Distanzen zu Knoten in M_1
- Gib am Ende minimal gefundene Distanz zurück

Nächstes-Paar-Problem

Laufzeit für Nächstes-Paar-Algo:

- Mergesort am Anfang: $O(n \log n)$
- Laufzeit $T(n)$ von ClosestPair Algo:

$$T(1)=O(1), T(n)=2T(n/2)+O(n)$$

Gesamtlaufzeit: $O(n \log n)$

Brute force: $O(n^2)$