Technische Universität München
Institut für Informatik

Effiziente Algorithmen und Datenstrukturen
WS 1997/98 - Kapitel 4

Stand: 10. März 1998

Dozent:

Prof. Dr. Ernst W. Mayr

Mitschrift:

Martin Stumpf
Thomas Schöpf
Martin Hans
Guido Wimmel

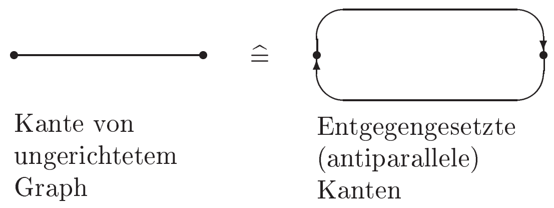
Inhaltsverzeichnis

Kapitel 4

Kürzeste Pfade

4.1 Grundlegende Begriffe

Betrachte Digraph $G = (V, A)$ oder Graph $G = (V, E)$.



Längenfunktion:

$$\mathrm{d} : A \longrightarrow \mathbb{R}^+ \text{ (bzw. } \longrightarrow \mathbb{R})$$

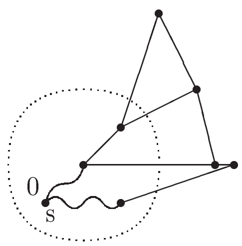
OBdA: $A = V^2$, $\mathrm{d}(x, y) = +\infty$ für Kanten, die eigentlich nicht vorhanden sind.
$\mathrm{dis}(v, w) :=$ Länge eines kürzesten Pfades von v nach $w \in \mathbb{R}^+ \cup \{+\infty\}$

Arten von kürzeste-Pfade-Problemen:

1. single-pair-shortest-path
2. single-source-shortest-path: gegeben G, d und $s \in V$, bestimme für alle $v \in V$ die Länge eines kürzesten Pfades von s nach v (bzw. einen kürzesten Pfad von s nach v)
3. all-pairs-shortest-path

4.2 Dijkstra's Algorithmus

Gegeben: $G = (V, A), (A = V^2)$, Distanzfunktion $d : A \to \mathbb{R}^+ \cup \{+\infty\}$. Startknoten s. G durch Adjazenzlisten dargestellt.



```
algorithm sssp:=
⌜
S := {s}; dis[s] := 0; initialisiere eine Priorityqueue, die alle Knoten v ∈ V \ {s}
enthält mit Schlüssel dis[v] := d(s, v);
for alle v ∈ V − {s} do
      from[v] := s
od
while S ≠ V do
      v := Find_Min(Q);
      Delete_Min(Q);
      S := S ∪ {v};
      for alle w ∈ V − S, d(v, w) < ∞ do
            if dis[v] + d(v, w) < dis[w] then
                  Decrease_Key(Q, w, dis[v] + d(v, w));
                        co Decrease_Key verändert dis[w] oc
                  from[w] := v;
            fi
      od
od
⌞
```

Laufzeit (mit Fibonacci-Heaps):
Initialisierung: $\mathcal{O}(n)$
Sonstiger Aufwand: $\mathcal{O}(m)$ (z.B. Decrease_Key)
Zeitbedarf also $\mathcal{O}(m + n \log n)$ mit $n = |V|$ und $m :=$ Anzahl wirklicher Kanten

Korrektheit: Wir behaupten, daß in dem Moment, in dem ein $v \in V - \{s\}$ Ergebnis der Find_Min Operation ist, der Wert dis$[v]$ des Schlüssels von v gleich der Länge eines kürzesten Pfades von s nach v ist.

Beweis: durch Widerspruch: Sei $v \in V - \{s\}$ der erste Knoten, für den diese Behauptung nicht stimmt und sei

$$s \longrightarrow s_1 \longrightarrow s_2 \cdots \longrightarrow s_r \longrightarrow v_1 \longrightarrow v_2 \cdots \longrightarrow v_q \longrightarrow v$$

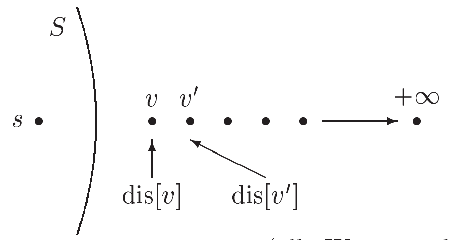
ein kürzester Pfad von s nach v, mit einer Länge $<$ dis$[v]$. Dabei sind $s_1, \ldots, s_r \in S$, $v \notin S$ [$r = 0$ und/oder $q = 0$ ist möglich]. Betrachte Pfad $s \longrightarrow s_1 \cdots \longrightarrow s_r \longrightarrow v_1$ seine Länge ist $<$ dis$[v]$, also dis$[v_1] <$ dis$[v]$. Widerspruch zur Wahl von v. □

Satz: Dijkstra's Algorithmus (mit Fibonacci-Heaps) löst das Single-Source-Shortest-Path-Problem in Zeit $\mathcal{O}(m + n \log n)$.

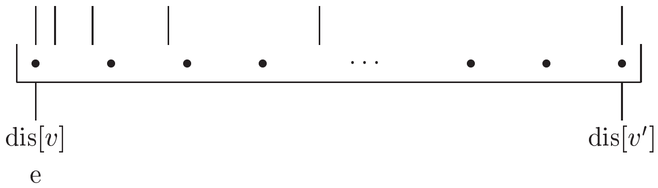
Bemerkung:

1. Verwendet man Dijkstra's Algorithmus mit d-Heaps, so erhält man Laufzeit $\mathcal{O}(m \log_{2+\frac{m}{n}} n)$
2. Mikkel Thorup (STOC '98): $\mathcal{O}(m)$
3. Dijkstra's Algorithmus mit Radix-Heaps (Akuja, Mehlhorn, Orlin, Tarjan; Faster Algorithms for the Shortest Path Problem. J ACM 37(1990), S. 213-223)

4.3 Dikstra's Algorithmus mit Radix-Heaps



d(v) $\in [0, ..., C]$ (alle Werte sind zwischen dis[v] und nC)



Einstufige Radix-Heaps:

Menge von $B = \lceil \log_2(C+1) \rceil + 2$ Buckets (jeder Bucket als doppelt verkettete Liste implementiert). Jeder Bucket hat Größe size, mit

$$\text{size}(i) = \begin{cases} 1 & \text{für } i = 1 \\ 2^{i-2} & \text{für } i = 2, \ldots, B-1 \\ nC+1 & \text{für } i = B \end{cases}$$

Beobachtung:

$$\sum_{i=1}^{j-1} \text{size}(i) \geq \min\{\text{size}(j), C+1\} \text{ für } j = 2, \ldots, B$$

Wir ordnen den Buckets Intervalle zu, indem wir für jeden Bucket i eine obere Intervallgrenze u(i) angeben. Das Intervall für Bucket i ist dann [u($i-1$)+1,u(i)]. Setze zu Beginn u(0) := -1. Die u(i) ändern sich im Algorithmus, die size(i) bleiben konstant. Zu Beginn:

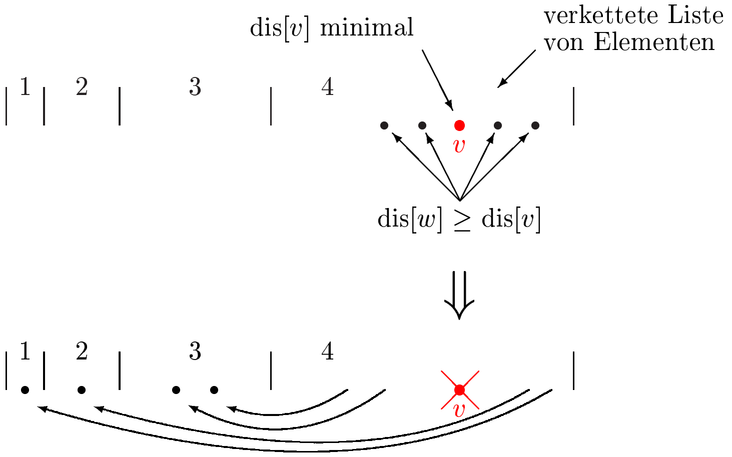
$$\left.\begin{array}{ll} \text{u}(i) & := 2^{i-1} - 1 \text{ für } i = 1, \ldots, B-1 \\ \text{u}(B) & := nC + 1 \end{array}\right\rangle \begin{array}{l} \text{Größe des dem Bucket i} \\ \text{zugeordneten Intervalls:} \\ u(i) - u(i-1) \leq \text{size}(i) \end{array}$$

- Initialisierung: Einfügen aller Schlüssel in die Buckets: durchlaufe, für Schlüssel k, die Buckets B, $B-1$, ... bis wir Bucket j finden mit u(j) $< k$. Füge k in Bucket $j+1$ ein. Kosten: $n \cdot \log C$.

- Decrease_Key: wie bei Insert, beginne Suche mit momentanem Bucket des Schlüssels. Kosten für alle Decrease_Key Operationen: $\mathcal{O}(1)$ pro Decrease_Key $+\mathcal{O}(n \log C)$.

- Delete_Min: Suche von links den ersten nicht leeren Bucket j. Falls $j = 1$ ist alles ok (wegen size(1)=1). Falls $j \geq 2$: suche minimales Element v im Bucket j (v wird als Find_Min zurückgegeben und aus der Priority Queue gelöscht), verteile alle anderen Elemente im Bucket j auf die Buckets $1, \ldots, j-$

1, nachdem die u(i)'s wie folgt aktualisiert wurden:
u(0) := dis[v] − 1
u(1) := dis[v]
u(i) := min{u(i − 1) + size(i),u(j)} für $i = 2, \ldots, j-1$



Kosten für alle Delete_Mins: $\mathcal{O}(\log C)$ pro Delete_Min + $\mathcal{O}(n \log C)$.

Damit Gesamtkosten für Dijkstra's Algorithmus mit einstufigen Radix-Heaps:

$$\mathcal{O}(m + m + n\log C + n\log C + n\log C) = \mathcal{O}(m + n\log C)$$

Satz: Dijkstra's Algorithmus mit einfstufigen Radix-Heaps hat Zeitkomplexität $\mathcal{O}(m + n\log C)$.

Verbesserungen:

1. zweistufige Heaps: $\mathcal{O}(m + n \cdot \frac{\log C}{\log\log C})$.
2. mehrstufige Heaps + Fibonacci-Heaps: $\mathcal{O}(m + n\sqrt{\log C})$.

4.4 Floyd's Algorithmus für das All-pairs-shortest-path-Problem

(auch genannt „Kleene's Algorithmus“)

Sei $G = (V, E)$ mit Distanzfunktion d ≥ 0 gegeben. Sei o.B.d.A. $V = \{v_1, \ldots, v_n\}$. Sei c_{ij}^k := Länge eines kürzesten Pfades $v_i = v^{(0)} - v^{(1)} - v^{(2)} - \ldots - v^{(r-1)} - v^{(r)} = v_j$ mit $v^{(l)} \in \{v_1, \ldots, v_k\}$ für $l = 1, \ldots, r-1$.

<u>algorithm</u> floyd:=
⌜
<u>for</u> alle (i,j) <u>do</u> $c_{ij}^{(0)}$:= d_{ij} <u>od</u>
<u>for</u> $k := 1$ <u>to</u> n <u>do</u>
    <u>for</u> alle (i,j), $1 \leq i,j \leq n$ <u>do</u>
        $c_{ij}^{(k)} := \min\left\{c_{ij}^{(k-1)},\ c_{ik}^{(k-1)} + c_{kj}^{(k-1)}\right\}$
    <u>od</u>
<u>od</u>
⌞

Laufzeit: $\Theta(n^3)$.

Korrektheit:
Zu zeigen: $c_{ij}^{(k)}$ des Algorithmus $= c_{ij}^k$ (damit sind für $k = n$ die Kosten der kürzesten Pfade durch $c_{ij}^{(k)}$ gegeben).

Beweis: Richtig für $k = 0$. Induktionsschluß: Ein kürzester Pfad von v_i nach v_j mit inneren Knoten $\in \{v_1, \ldots, v_{k+1}\}$ enthält entweder v_{k+1} gar nicht als inneren Knoten, oder er enthält v_{k+1} genau einmal als inneren Knoten. Im ersten Fall wurde dieser Pfad also bereits für $c_{ij}^{(k)}$ betrachtet, hat also Länge $\geq c_{ij}^{(k)}$. Im zweiten Fall setzt er sich aus einem kürzesten Pfad P_1 von v_i nach v_{k+1} und einem kürzesten Pfad von P_2 von v_{k+1} nach v_j zusammen, wobei alle inneren Knoten von P_1 und $P_2 \in \{v_1, \ldots, v_k\}$ sind. Also ist die Länge des Pfades $= c_{i,k+1}^{(k)} + c_{k+1,j}^{(k)}$. □

Satz: Floyd's Algorithmus für das All-pair-shortest-path-Problem hat Zeitkomplexität $\mathcal{O}(n^3)$.

4.5 Min-Plus-Matrix-Produkt und Min-Plus-Transitive Hülle

Ring $\mathbb{R}(+, \bullet)$ — $\nearrow$ Gruppe, $\nwarrow$ Halbgruppe

Semiring $\mathbb{N}(+, \bullet)$ — $\nearrow$ Halbgruppe, $\nwarrow$ Halbgruppe

Semiring über $\mathbb{R} \cup \{\infty\}$ mit Operationen min und $+$, jeweils Halbgruppe.
Distributivgesetz $a + \min\{b, c\} = \min\{a + b, a + c\}$

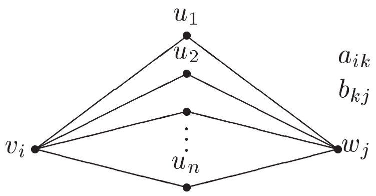
Normale Matrixmultiplikation:

$$A = (a_{ij})_{1 \leq i,j \leq n}, \quad B = (b_{ij})_{1 \leq i,j \leq n}, \quad I = (\delta_{ij})_{1 \leq i,j \leq n}$$

$$C = A \cdot B = (c_{ij})_{1 \leq i,j \leq n}, \quad c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

Entsprechend für Min-Plus:

$$c_{ij} = \min\{a_{ik} + b_{kj}; 1 \leq k \leq n\}$$



$a_{ik} = \mathrm{d}(v_i, u_k)$
$b_{kj} = \mathrm{d}(u_k, w_j)$

Anwendung: $A = B$
kürzeste Wege von v_i nach w_j in einem Graph $(A = B)$

$$I_{\min,+} = \begin{pmatrix} 0 & & \infty \\ & \ddots & \\ \infty & & 0 \end{pmatrix}$$

Sei A Entfernungsmatrix, $A = (a_{ij})_{1 \leq i,j \leq n} = (\mathrm{d}(v_i, v_j))_{1 \leq i,j \leq n}$. Setze $a_{ii} = 0$ für $i = 1, \ldots, n$.

Betrachte A^2 (min-plus-Produkt):
$A^2 = (a_{ij}^{(2)})_{1\le i,j\le n}$.
Dann ist $a_{ij}^{(2)}$ die Länge eines kürzesten Pfades von v_i nach v_j, der höchstens zwei Kanten enthält. Induktion: $a_{ij}^{(k)}$ ist die Länge des kürzesten Pfades von v_i nach v_j mit höchstens $\underline{k}$ Kanten.
Falls die $\mathrm{d}(v_i, v_j)$ alle ≥ 0, gibt es immer kürzeste Pfade, die $\le n-1$ Kanten enthalten.

Alternative Lösung des All-Pairs-Shortest-Path-Problems: berechne A^{n-1} (min-plus). Es genügt auch $A^{2^{\lceil \log(n-1)\rceil}}$ (nicht $A^2, A^3, A^4, \ldots$ berechnen, sondern wiederholt quadrieren).

$A^* := \min\limits_{i\ge 0}\{A^i\}$ heißt Min-Plus-Transitive Hülle.
Bemerkung: min wird Komponentenweise gebildet. Wenn $d \ge 0$, dann $A^* = A^{n-1}$.

4.6 Boolesche Matrixmuliplikation und Transitive Hülle

Ersetze in 4.5. Distanzmatrix durch (boolesche) Adjazenzmatrix und ersetze min-plus durch $\vee, \wedge$, d.h.:

$$C = A \cdot B; \quad c_{ij} = \bigvee_{k=1}^{n} a_{ik} \wedge b_{kj}$$

Setze $a_{ii} = 1, 1 \le i \le n$
Dann gilt für A^k (boolesches Produkt, $A^0 = I$)

$$a_{ij} = \begin{cases} 1 & \text{falls es im Graphen einen Pfad von } v_i \text{ nach } v_j \\ & \text{bestehend aus } \le k \text{ Kanten gibt} \\ 0 & \text{falls es im Graphen keinen Pfad von } v_i \text{ nach } v_j \\ & \text{bestehend aus } \le k \text{ Kanten gibt} \end{cases}$$

Transitive Hülle

$$A^* := \bigvee_{i\ge 0} A^i$$

ist damit die Adjazenzmatrix der transitiven Hülle des zugrundeliegenden Graphen $(= A^{n-1})$.
Sei $\mathrm{M}(n)$ die Zeitkomplexität für das boolesche Produkt zweier $n \times n$ - Matrizen, $\mathrm{T}(n)$ die Zeitkomplexität für die transitive Hülle einer $n \times n$ booleschen Matrix.

Satz: Falls $\underbrace{\mathrm{T}(3n) \le c\mathrm{T}(n)}_{\substack{\text{sicher erfüllt, falls} \\ \text{T polynomiell}}}$ und $\underbrace{\mathrm{M}(2n) \ge 4\mathrm{M}(n)}_{\substack{\text{sicher erfüllt, falls} \\ \mathrm{M}(n) \ge n^2}}$, dann gilt $\mathrm{T}(n) = \Theta(\mathrm{M}(n))$.

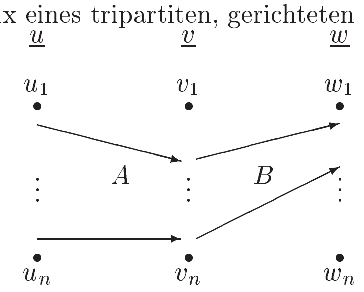
Beweis:

- Matrixmultiplikation $\prec$[1] transitive Hülle:
  Gegeben: A, B; Gesucht: $C = A \cdot B$ (boolesches Produkt); Setze:

$$L = \underbrace{\begin{pmatrix} 0 & A & 0 \\ 0 & 0 & B \\ 0 & 0 & 0 \end{pmatrix}}_{3n} \left.\vphantom{\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}}\right\} 3n$$

  L ist Adjazenzmatrix eines tripartiten, gerichteten Graphen, denn:

$$\begin{array}{c|ccc} & \underline{u} & \underline{v} & \underline{w} \\ \hline \underline{u} & 0 & A & 0 \\ \underline{v} & 0 & 0 & B \\ \underline{w} & 0 & 0 & 0 \end{array}$$



  Daher kann L^* leicht bestimmt werden:

$$L^* = \begin{pmatrix} I & A & AB \\ 0 & I & B \\ 0 & 0 & I \end{pmatrix}$$

  Also gilt: $\mathrm{M}(n) \leq \mathrm{T}(3n) = \mathcal{O}(\mathrm{T}(n))$.

- Transitive Hülle $\prec$ Matrixmultiplikation
  Gegeben: L $n \times n$ Boolesche Matrix; Gesucht: L^*; Annahme: n ist Zweierpotenz. Teile auf:

$$L = \begin{pmatrix} \underbrace{A}_{\frac{n}{2}} & \underbrace{B}_{\frac{n}{2}} \\ C & D \end{pmatrix} \begin{array}{l} \}\,\frac{n}{2} \\ \}\,\frac{n}{2} \end{array} ; \quad L^* = \begin{pmatrix} E & F \\ G & H \end{pmatrix}$$

  Dann gilt:

$E =$	$(A \vee BD^*C)^*$	betrachte alle Pfade von der ersten Hälfte der Knoten zur ersten Hälfte der Knoten
$F =$	EBD^*	analog
$G =$	D^*CE	analog
$H =$	$D^* \vee GF$	analog

  Um L^* zu berechnen, benötigen wir zwei Transitive-Hülle-Berechnungen und sechs Matrixprodukte für Matrizen der Dimension $\frac{n}{2} \times \frac{n}{2}$, plus Aufwand für $\vee$, der $\leq c'n^2$ ist. Also:

$$\begin{array}{lll|ll} \mathrm{T}(n) & \leq 2\mathrm{T}(\frac{n}{2}) + 6\mathrm{M}(\frac{n}{2}) + c'n^2 & & \text{Behauptung:} & \mathrm{T}(n) \leq c\mathrm{M}(n) \\ & & & \text{Beweis:} & \text{durch Induktion:} \\ & & & & n = 1\surd \\ & \leq 2c\mathrm{M}(\frac{n}{2}) + 6\mathrm{M}(\frac{n}{2}) + c'n^2 & & \text{Vorraus.:} & \mathrm{M}(2n) \geq 4\mathrm{M}(n) \\ & & & & \mathrm{M}(n) \geq n^2 \\ & \leq \frac{1}{4}(2c + 6 + 4c')\mathrm{M}(n) & & & \\ & \leq c\mathrm{M}(n) & & & \end{array}$$

[1] $x \prec y$ bedeutet „reduziere x auf y“

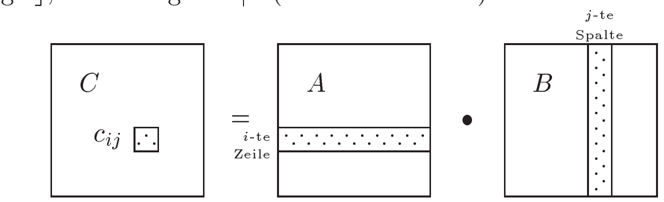
falls $c \geq \frac{1}{4}(2c + 6 + 4c')$, also falls $c \geq 3 + 2c'$.
Also $\mathrm{T}(n) = \mathcal{O}(\mathrm{M}(n))$.

□

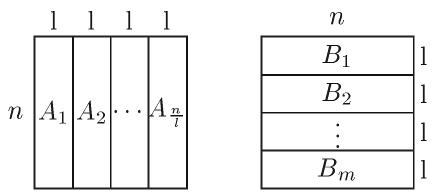
4.7 Der 4-Russen-Algorithmus für Boolesche Matrixmultiplikation

Autoren:
V.L. Avlazarov, E.A. Dinic, M.A. Kronod, I.A. Faradzev : "On economical construction of the transitive closure of a directed graph", Soviet Math. Doklady 11 (1970), p. 1209–1210.

Gegeben zwei Boolesche $n \times n$ Matrizen A, B; gesucht: $C = A \cdot B$.
Sei $l := \lfloor \log n \rfloor$, o.B.d.A. gelte $l | n$ (l ist Teiler von n).



Teile A auf (setze $m := \frac{n}{l}$):



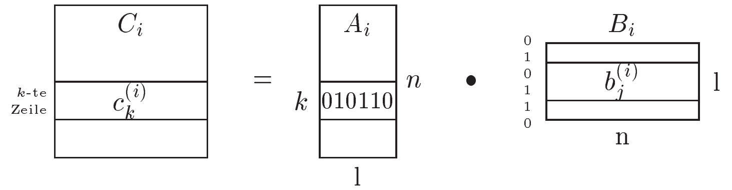
Sei $C_i := A_i \cdot B_i$ für $i = 1, \ldots, m$. Dann gilt

$$C = \bigvee_{i=1}^{m} C_i, \text{ da}$$

$$C = AB = \left(\bigvee_{i=1}^{m} A_i \right) \left(\bigvee_{i=1}^{m} B_i \right) = \bigvee_{\substack{1 \leq i \leq m \\ 1 \leq j \leq m}} A_i B_j = \bigvee_{i=1}^{m} A_i B_i,$$

da $A_i B_j = 0$ für $i \neq j$ (fasse A_i und B_j als $n \times n$ Matrizen auf mit 0 außerhalb des jeweiligen Streifens). Gegeben die C_i's, benötigen wir Zeit $\mathcal{O}(mn^2)$.

Betrachte eine Zeile von C_i:



$$c_k^{(i)} = \bigvee_{j=1}^{l} a_k^{(i)} \cdot b_j^{(i)}$$

Der Algorithmus berechnet einfach zunächst alle booleschen Linearkombinationen der Zeilen von B_i (Prozedur bcomb), und damit $c_k^{(i)}$ für jedes mögliche $a_k^{(i)}$.
Betrachte A, B und C als Matrizen von Zeilenvektoren:

$$A = \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix}, \quad B = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}, \quad C = \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix}$$

```
proc bcomb(int i) =
⌜
comb[0] := [0, ... ,0];
for j := 1 to 2^⌊log n⌋ − 1 do
      p := ⌊log j⌋;    co p Index der vordersten 1 von j oc
      comb[j] :=comb[j − 2^p] ∨ b_((i−1)⌊log n⌋+p+1)
od
⌞
```

Zeitbedarf:

(a) sequentiell: $\mathcal{O}(n^2)$

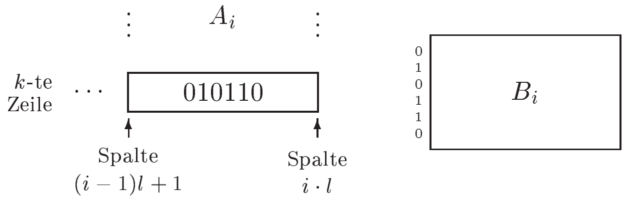
(b) Vektoroperationen der Breite n: $\mathcal{O}(n)$

```
algorithm four-russians(array a, b, c) =
⌜
co die Matrizen a, b, c sind als Vektor von n Zeilenvektoren organisiert oc
const l = ⌊log n⌋ co wir nehmen l|n an oc;
array comb[0..2^(l−1)] of boolean-vector;

for i := 1 to n do c[i] := [0, ... ,0] od

for i := 1 to n/l do     co Berechne die C_i's oc
      bcomb(i);
      for j := 1 to n do
            co Bitmuster in der j-ten Zeile von A_i in Binärzahl wandeln oc
            nc := 0;
            for k := i · l downto (i − 1) · l + 1 do
                  if a[j, k] then nc := nc + nc + 1 else nc := nc + nc fi
            od
            c[j] := c[j]∨comb[nc];
      od
od
⌞
```

Beispiel:



Zeitbedarf:

(a) sequentiell:
$\mathcal{O}\left(\frac{n}{l} \cdot \left(n^2 + n(l+n)\right)\right) = \mathcal{O}\left(\frac{n^3}{l}\right) = \mathcal{O}\left(\frac{n^3}{\log n}\right)$

(b) Vektorrechner der Breite n (Interpretation eines Bitintervalls als Zahl in $\mathcal{O}(1)$ Zeit):
$\mathcal{O}\left(\frac{n}{l} \cdot (n + n(1+1))\right) = \mathcal{O}\left(\frac{n^2}{\log n}\right)$ (Vektoroperationen)

Satz: Der 4-Russen Algorithmus berechnet das Produkt zweier Boolescher Matrizen sequentiell in Zeit $\mathcal{O}\left(\frac{n^3}{\log n}\right)$ bzw. mit $\mathcal{O}\left(\frac{n^2}{\log n}\right)$ Vektoroperationen.

Beweis: $\checkmark$ (s.o.) □

4.8 Transitive Hülle und DFS[2]

Tiefensuche(DFS):

```
proc DFS-visit(node v) =
⌜
markiere v;
num[v] := ++count;
for alle Nachbarn u von v do
    if u unmarkiert then DFS-visit(u) fi
od
⌞
```

```
algorithm DFS =
⌜
for alle Knoten v do unmarkiere(v); num(v) := 0 od;
count := 0;
for alle Knoten v do
    if v unmarkiert then DFS-visit(v) fi
od
⌞
```

Bemerkung: Funktioniert für Graphen und Digraphen.

Nachbarn in Graphen und Digraphen:

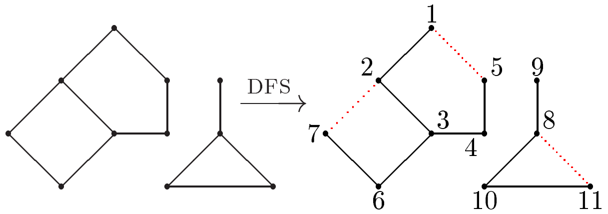
u Nachbar von v
v Nachbar von u

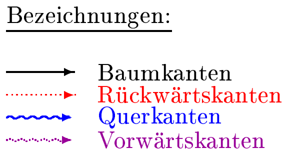
u Nachbar von v

[2]DFS: Depth First Search

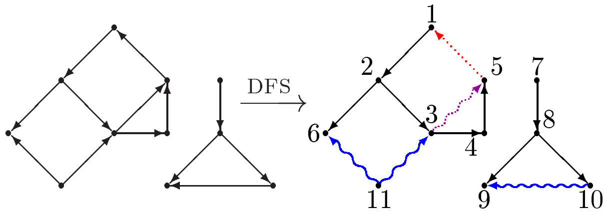
Beispiele für DFS:

Graph G_1:

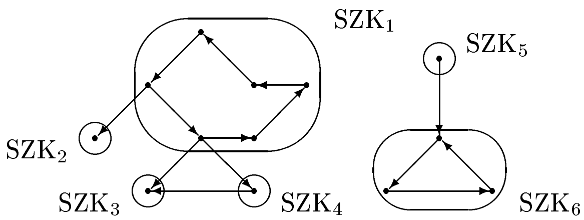




Digraph G_2:

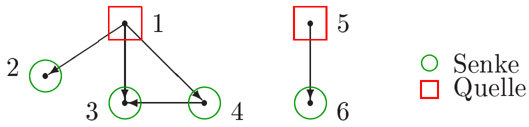


Bemerkung: Alle Baumkanten zusammen ergeben den DFS-Wald (Spannwald).



Schrumpfen $\Downarrow$ der SZK's

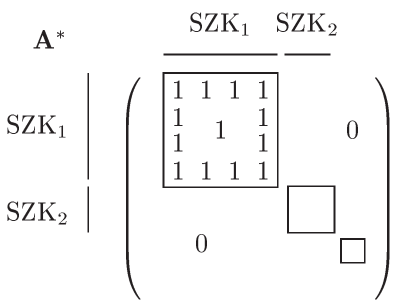
DAG (**D**irected **A**cyclic **G**raph)



Bemerkung: SZK: starke Zusammenhangskomponente (in Digraphen)

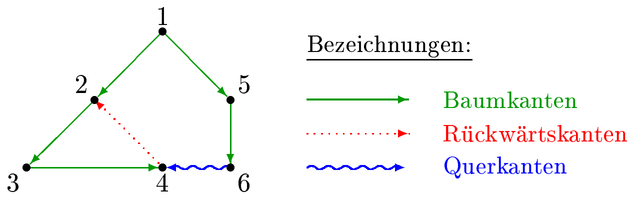
DFS in ungerichteten zyklischen Graphen, n Knoten, m Kanten, $n-1$ Baumkanten, $m-n+1$ Rückwärtskanten.

Zeitaufwand: $\mathcal{O}(n+m)$



A^* aus DFS mit Aufwand $\Theta(n^2)$ in ungerichteten Graphen (Anm.: SZK = ZK).

DFS in gerichteten Graphen (Digraphen), n Knoten, m Kanten, Baum-, Vorwärts-, Rückwärts- und Querkanten:



Zeitaufwand: $\mathcal{O}(n+m)$
Es gilt: u ist von v aus auf einem gerichteten Pfad erreichbar gdw. u Knoten in dem DFS-Baum (DFS-visit(v)) mit Wurzel v ist.
Also: Transitive Hülle in Zeit $\mathcal{O}(n \cdot (m+n))$ bzw. $\mathcal{O}(n \cdot m)$. Sehr gut für dünne Graphen (also solche mit relativ wenigen Kanten).

4.9 Ein besserer Algorithmus für das All-pairs-shortest-distance (bzw. paths)-Problem in ungewichteten Digraphen

$$\left.\begin{array}{l}(d_{ij})_{0\le i,j<n} \text{ Distanzmatrix}\\ D^* \text{ Distanzmatrix für Transitive Hülle}\end{array}\right\} \text{alle Kantenlängen} = 1$$

$$D^{(l)} = (d_{ij}^{(l)})_{0\le i,j<n} \text{ mit } d_{ij}^{(l)} = \begin{cases} d_{ij}^* \text{ falls } d_{ij}^* \le l \\ \infty \text{ sonst}\end{cases}$$

<u>algorithm</u> 1 =
⌜

<u>co</u> Sei $A = (a_{ij})_{0\le i,j<n}$ mit $a_{ij} = \begin{cases} 1 \text{ falls } d_{ij} \le 1 \\ 0 \text{ sonst}\end{cases}$ <u>oc</u>
<u>co</u> Sei B die Boolsche Einheitsmatrix <u>oc</u>
<u>for</u> $l := 2$ <u>to</u> $r+1$ <u>do</u>
    $B := A \cdot B$;
    <u>for</u> $0 \le i,j < n$ <u>do</u>
        <u>if</u> $b_{ij} = 1$ <u>then</u> $d_{ij}^{(l)} := l$ <u>else</u> $d_{ij}^{(l)} := \infty$ <u>fi</u>;
        <u>if</u> $d_{ij}^{(l-1)} \le l$ <u>then</u> $d_{ij}^{(l)} := d_{ij}^{(l-1)}$ <u>fi</u>
    <u>od</u>
<u>od</u>
⌞

Dieser Algorithmus berechnet $D^{(1)} = D, D^{(2)}, D^{(3)}, \ldots, D^{(r)}$.
Sei ω eine Zahl ≥ 2, so daß die Matrixmultiplikation in der Zeit $\mathcal{O}(n^\omega)$ durchgeführt werden kann (Winograd/Coppersmith: $\omega \le 2{,}376$).

Zeitaufwand für Algorithmus 1: $\mathcal{O}(rn^\omega)$

algorithm apsd =

⌜

Berechne, mit Algorithmus 1, $D^{(l)}$ für $l = 1, \ldots, r$;
$l := r$;
for $s := 1$ to $\left\lceil \log_{\frac{3}{2}} \frac{n}{r} \right\rceil$ do
    for $i := 0$ to $n-1$ do
        finde in der i-ten Zeile von $D^{(l)}$ den Wert d, $\lceil \frac{l}{2} \rceil \leq d \leq l$, so daß d in der i-ten Zeile am wenigsten oft vorkommt;
        $S_i :=$ Menge der Spaltenindizes, in denen dieses d in der i-ten Zeile von $D^{(l)}$ vorkommt;
    od
    $l_1 := \lceil \frac{3}{2} l \rceil$;
    for $0 \leq i, j < n$ do
        if $S_i \neq \emptyset$ then $m_{ij} := \min\limits_{k \in S_i} \{d_{ik}^{(l)} + d_{kj}^{(l)}\}$ else $m_{ij} := \infty$ fi;
        if $d_{ij}^{(l)} \leq l$ then $d_{ij}^{(l_1)} := d_{ij}^{(l)}$
        elseif $m_{ij} \leq l_1$ then $d_{ij}^{(l_1)} = m_{ij}$
        else $d_{ij}^{(l_1)} := \infty$
        fi
    od
    $l := l_1$;
od

⌞

apsd berechnet:
Zunächst $D^{(1)}, \ldots, D^{(r)}$, dann $D^{(l)}$ für

$$l = r, \left\lceil \frac{3}{2} r \right\rceil, \left\lceil \frac{3}{2} \left\lceil \frac{3}{2} r \right\rceil \right\rceil, \ldots, \underbrace{n'}_{\geq n}$$

$$i \bullet\!\!-\!\!\overbrace{\qquad\qquad\qquad}^{d}\!\!-\!\!\underset{k \in S_i}{\bullet}\!\!-\!\!\overbrace{\qquad\qquad\qquad}^{\lceil \frac{3}{2} l \rceil - d \,\leq\, l}\!\!-\!\!\bullet j$$

Da für d $\frac{l}{2}$ Werte zur Auswahl stehen, gilt:

$$|S_i| \leq \frac{2n}{l}$$

Damit ist die Laufzeit:

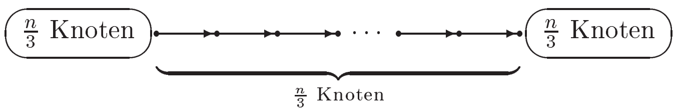
$$\mathcal{O}\left(rn^{\omega} + \sum_{s=1}^{\left\lceil \log_{\frac{3}{2}} \frac{n}{r} \right\rceil} \frac{n^3}{\left(\frac{3}{2}\right)^s \cdot r} \right) = \mathcal{O}\left(rn^{\omega} + \frac{n^3}{r} \right)$$

Setze r so, daß die beiden Summanden gleich sind, also $r = n^{\frac{3-\omega}{2}}$, dann ergibt sich damit für apsd die Laufzeit $\mathcal{O}\left(n^{\frac{3+\omega}{2}}\right)$.

Satz: Das All-Pairs-Shortest-Distance-Problem für Digraphen mit Kantenlänge = 1 läßt sich in Zeit $\mathcal{O}(n^{\frac{3+\omega}{2}})$ lösen (ω Exponent für Matrixmultiplikation).

Beweis: $\surd$ □

Bemerkung: APSP: Problem: Ausgabegröße kann $\Omega(n^3)$ sein:



Kürzeste Pfade-Nachfolger-Matrix $N \in [0, \ldots, n-1]^{n\times n}$
n_{ij} = Nummer des zweiten Knoten auf „dem“ kürzesten Pfad von Knoten i zu Knoten j. So eine Matrixdarstellung für die kürzesten Pfade zwischen allen Knotenpaaren kann in Zeit $\mathcal{O}(n^{\frac{3+\omega}{2}} \cdot \log^c n)$ für ein $c > 0$ bestimmt werden.

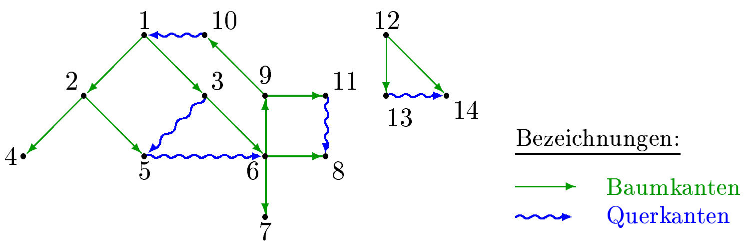
Korollar: Für Digraphen mit Kantenlängen $\in \{1, 2, \ldots, M\}, M \in \mathbb{N}$, kann APSD in Zeit $\mathcal{O}((Mn)^{\frac{3+\omega}{2}})$ gelöst werden.

Idee: Ersetze $u \xrightarrow{M' \leq M} v$ durch $u{=}u_0 \longrightarrow u_1 \longrightarrow u_2 \cdots \longrightarrow u_{M'}{=}v$

4.10 Ein Algorithmus für die transitive Hülle in Digraphen mit linearer erwarteter Laufzeit

Die Wahrscheinlichkeit eines Digraphen mit n Knoten und m Kanten ist (nur) eine Funktion von n und m. Daraus folgt (wir lassen der Einfachheit halber Schleifen (=Kanten $v \to v$) zu), daß jede Kante (u, v) mit Wahrscheinlichkeit $\frac{m}{n^2}$ vorhanden ist, falls wir Graphen mit n Knoten und m Kanten betrachten.
Einschub: Breitensuche (BFS): Schlange, queue, FIFO:

Durchlaufe Graphen, indem wir, solange möglich den vordersten Knoten v aus der queue nehmen, ihn behandeln und die Liste seiner noch nicht behandelten Nachbarn in die queue hinten einfügen.
Beispiel:



<u>algorithm</u> exp-lin-transitive-closure

(C.P. Schnorr, An Algorithm for Transitive Closure with Linear Expected Time, SIAM Journal on Computing <u>7</u> (1978), pages 127-133)
Phasen:

0. Konstruiere die linear geordneten Adjazenzlisten $L_i^r, i = 1, \ldots, n$, des Graphen G^r (entsteht aus G durch Umkehrung aller Knoten).
   Beispiel:

$$\begin{array}{lcccc|lcccc} L_1 & = & 4 & 1 & 2 & L_1^r & = & 1 & 2 & 4 \\ L_2 & = & 1 & 4 & 3 & L_2^r & = & 1 & 3 & 4 \\ L_3 & = & 3 & 2 & & L_3^r & = & 2 & 3 & \\ L_4 & = & 1 & 4 & 2 & L_4^r & = & 1 & 2 & 4 \end{array}$$

Ersetze ebenfalls alle L_i durch L_i^{rr} ($\rightarrow$ sortierte Adjazenzlisten)

1. Berechne für jeden Knoten i in BFS-Art eine Liste S_i von von i aus erreichbaren Knoten, so daß (i) oder (ii) gilt:

   (i) $|S_i| < \lfloor \frac{n}{2} \rfloor + 1$ und S_i enthält alle von i aus erreichbaren Knoten

   (ii) $|S_i| = \lfloor \frac{n}{2} \rfloor + 1$

2. Entsprechende Listen P_i von Vorgängern von i

3. for $i := 1$ to n do
   $L_i^* := S_i \cup \{j; i \in P_j\} \cup \{j; |S_i| = |P_j| = \lfloor \frac{n}{2} \rfloor + 1\}$;
   od

   Bilde $(L_i^*)^{rr}$ für $i = 1, \dots, n$

Korrektheit: $j \in L_i^*$ gdw. (i, j) in transitiver Hülle.

Satz: Sei die Wahrscheinlichkeit eines Graphen (lediglich) eine Funktion der Anzahl n seiner Knoten und der der Anzahl m seiner Kanten. Dann ist der Erwartungswert für die Laufzeit des Schnorr'schen Algorithmus $\mathcal{O}(n + m^*)$, wobei m^* der Erwartungswert für die Anzahl der Kanten in der transitiven Hülle ist.

Beweis: Das Durchlaufen des Graphen (von V_1 aus für die Konstruktion von S_1) in BFS-Manier liefert eine Folge $(a_t)_{t \geq 1}$ von Knoten. Sei $L_{\sigma(\nu)}$ die ν-te Adjazenzliste, die in der BFS erkundet wird,

$$L_{\sigma(\nu)} = (a_t; \mathrm{h}(\nu) < t \leq \mathrm{h}(\nu + 1))$$

Sei $\Delta L_{\sigma(\nu)}$ die Menge der a_t in $L_{\sigma(\nu)}$, und sei

$$S_i(t) := \{a_1, a_2, \dots, a_t\}$$

Wie bereits gezeigt, ist

$$\mathrm{WS}(j \in L_i) = \frac{m}{n^2}, \quad \forall i, j$$

Alle n-Tupel von geordneten Listen $L_1, L_2, ..., L_n$ mit

$$m = \sum_{i=1}^{n} |L_i|$$

sind aufgrund der Voraussetzungen über die Wahrscheinlichkeitsverteilung gleich wahrscheinlich.

Also:
Beobachtung 1: Die Mengen $\Delta L_{\sigma(\nu)}, \nu = 1, 2, \dots$ sind (paarweise) unabhängig.
Beobachtung 2: Die $\Delta L_{\sigma(\nu)}$ sind gleichverteilt, d.h. für alle $A \subseteq \{1, \dots, n\}$ gilt:

$$\mathrm{WS}(\Delta L_{\sigma(\nu)} = A) = (\frac{m}{n^2})^{|A|}(1 - \frac{m}{n^2})^{n-|A|}$$

Lemma: Sei $A \subseteq \{1, \dots, n\}$ mit $|A| = k$. Sei $(\bar{a}_t)_{t \geq 1}$ eine Folge von unabhängigen gleichverteilten Zufallsvariablen mit Wertemenge $\{1, \dots, n\}$. Dann gilt:

$$\min\{t; \bar{a}_t \notin A\} \quad \text{hat Erwartungswert} \quad 1 + \frac{k}{n-k}.$$

Beweis: $\mathrm{WS}(\bar{a}_1, \dots, \bar{a}_r \in A$ $\underline{\text{und}}$ $\bar{a}_{r+1} \notin A) = \left(\frac{k}{n}\right)^r \cdot (1 - \frac{k}{n})$.
Also ist der Erwartungswert für $\min\{t; \bar{a}_t \notin A\}$:

$$\begin{aligned}
&1 + \sum_{r=0}^{\infty} r \left(\frac{k}{n}\right)^r \left(1 - \frac{k}{n}\right) \\
&= 1 + \frac{k}{n}\left(1 - \frac{k}{n}\right) \sum_{r \geq 1} r \left(\frac{k}{n}\right)^{r-1} \\
&= 1 + \frac{k}{n}\left(1 - \frac{k}{n}\right) \frac{1}{\left(1 - \frac{k}{n}\right)^2} \\
&= 1 + \frac{\frac{k}{n}}{1 - \frac{k}{n}} \\
&= 1 + \frac{k}{n-k}
\end{aligned}$$

□

Anmerkung:

$$\sum_{r=0}^{\infty} r z^{r-1} = \frac{d}{dz}\frac{1}{1-z} = \frac{1}{(1-z)^2}$$

Lemma: Sei $(\bar{a}_t)_{t \geq 1}$ wie oben, $k \leq \frac{n}{2}$. Dann hat $\min\{t; |\bar{a}_1, \bar{a}_2, \dots, \bar{a}_t| > k\}$ Erwartungswert $\leq \frac{3}{2}(k+1)$.

Beweis: Wegen der Additivität des Erwartungswertes gilt: Der gesuchte Erwartungswert ist

$$\begin{aligned}
&\leq \sum_{\nu=0}^{k} \left(1 + \frac{\nu}{n - \nu}\right) \\
&\leq k + 1 + \sum_{\nu=1}^{k} \frac{\nu}{\frac{n}{2}} \\
&\leq k + 1 + \frac{k(k+1)}{n} \\
&\leq \frac{3}{2}(k+1)
\end{aligned}$$

□

Wenn wir jedes $L_{\sigma(\nu)}$ in sich zufällig permutieren, erhalten wir eine Folge $(\bar{a}_t)'_{t \geq 1}$, von Zufallsvariablen, so daß

$$|\{a_1, \dots, a_t\}| = |\{a'_1, \dots, a'_t\}| \text{ für } t = \mathrm{h}(\nu), \forall \nu$$

Da die a'_t innerhalb eines jeden Intervalls $\mathrm{h}(\nu) < t \leq \mathrm{h}(\nu + 1)$ paarweise verschieden sind, gilt für sie die vorhergehende Abschätzung erst recht. Daher sind $(|S_1(t)|)_{t \geq 1}$ und $(|S_1(t)'|)_{t \geq 1}$ gleichverteilte Folgen von Zufallsvariablen, denn dies gilt zunächst für alle Zeitpunkte der Form $t = \mathrm{h}(y)$; da aber für diese Zeitpunkte $S_1(t) = S'_1(t)$

ist und h() zufällig verteilt ist, ist auch die Verteilung der beiden Folgen zwischen diesen Intervallgrenzen identisch.

Sei s der Erwartungswert und sei $|S_i|$ die tatsächliche Größe von S_i nach der Phase 1. Dann

$$s = \sum_{k=1}^{\lfloor \frac{n}{2} \rfloor + 1} k \cdot \mathrm{WS}(|S_i| = k)$$

Die erwartete Anzahl der Schritte (und damit die Kosten) in der BFS sei q. Dann gilt:

$$q \leq \sum_{k=1}^{\lfloor \frac{n}{2} \rfloor + 1} \mathrm{WS}(|S_i| = k) \frac{3}{2} k = \frac{3}{2} s.$$

Also ist der Aufwand des Algorithmus für die Phasen 1 und 2 im Erwartungswert $\leq 3sn$. Da $ns = m^*$, und da die Kosten der anderen Phasen offensichtlich durch $\mathcal{O}(n + m + m^*)$ beschränkt sind, folgt die Behauptung. □

4.11 Matrizenmultiplikation à la Strassen

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}; \quad c_{ij} = a_{i1} b_{1j} + a_{i2} b_{2j}$$

Bilde:

$$\begin{aligned}
m_1 &:= (a_{12} - a_{22})(b_{21} + b_{22}) \\
m_2 &:= (a_{11} + a_{22})(b_{11} + b_{22}) \\
m_3 &:= (a_{11} - a_{21})(b_{11} + b_{12}) \\
m_4 &:= (a_{11} + a_{12}) b_{22} \\
m_5 &:= a_{11}(b_{12} - b_{22}) \\
m_6 &:= a_{22}(b_{21} - b_{11}) \\
m_7 &:= (a_{21} + a_{22}) b_{11}
\end{aligned}$$

Dann gilt:

$$\begin{aligned}
c_{11} &:= m_1 + m_2 - m_4 - m_6 \\
c_{12} &:= m_4 + m_5 \\
c_{21} &:= m_6 + m_7 \\
c_{22} &:= m_2 - m_3 + m_5 - m_7
\end{aligned}$$

Sei $n = 2^k$ und $\mathrm{M}(n)$ die Anzahl arithmetischer Operationen (Mult, Add, Sub), die Strassen's Algorithmus bei $n \times n$ Matrizen benötigt:

$$\begin{aligned}
\mathrm{M}(1) &= 1 \\
\mathrm{M}(n) &= 7\mathrm{M}\left(\tfrac{n}{2}\right) + 18 \left(\tfrac{n}{2}\right)^2
\end{aligned}$$

Expansion dieser Rekursionsgleichung →

$$\mathrm{M}(n) = 7^{k+1} - 6n^2 = 7 \cdot 2^{k \log_2 7} - 6n^2 = 7n^{\log_2 7} - 6n^2 < 7n^{2{,}81} - 6n^2$$

<u>procedure</u> Matmult(A, B, n) =

⌜

<u>co</u> A und B sind $n \times n$ Matrizen <u>oc</u>
<u>if</u> $n < 16$ <u>then</u> berechne $A \cdot B$ mit klassischer Methode
<u>else</u> <u>if</u> n gerade <u>then</u>
spalte A und B in je 4 $\frac{n}{2} \times \frac{n}{2}$ Matrizen auf und wende Strassen's Formeln an. Führe die Multiplikationen rekursiv mit Matmult aus.
<u>else</u>
spalte A und B in je eine $(n-1) \times (n-1)$ Matrix (A_{11}, B_{11}) und drei verbleibende Blöcke auf. Wende Matmult rekursiv auf A_{11}, B_{11} an und berechne die anderen Produkte mit klassischer Methode.
<u>fi</u>
<u>fi</u>

⌞

Satz: Der Matmult-Algorithmus hat folgende Eigenschaften:

1. Für $n < 16$ ist die Anzahl der arithmetischen Operationen genauso hoch wie bei der klassischen Methode.
2. Für $n \geq 16$ ist die Anzahl der arithmetischen Operationen echt kleiner als bei der klassischen Methode.
3. Matmult braucht nie mehr als $4.8 n^{\log_2 7}$ arithmetische Operationen.

Beweis:

1. $\surd$

2. Sei n gerade und seien A, B zwei $n \times n$ Matrizen. Wir wenden Strassen's Formeln an, führen aber die 7 Multiplikationen der $\frac{n}{2} \times \frac{n}{2}$ Matrizen mit klassischer Methode aus.
Gesamtzahl arithmetischer Operationen:
$$7\left(2\left(\frac{n}{2}\right)^3 - \left(\frac{n}{2}\right)^2\right) + 18\left(\frac{n}{2}\right)^2 = \frac{7}{4}n^3 + \frac{11}{4}n^2$$
Dies ist besser als die klassische Methode, falls:
$$\frac{7}{4}n^3 + \frac{11}{4}n^2 < 2n^3 - n^2 \Leftrightarrow \frac{15}{4}n^2 < \frac{1}{4}n^3 \Leftrightarrow 15 < n \Leftrightarrow n \geq 16$$
Sei n ungerade. Multiplizieren wir $A \cdot B$ mit klassischer Methode, so auch $A_{11} \cdot B_{11}$. Also brauchen wir durch Anwendung von Strassen's Formeln auf die $(n-1) \times (n-1)$ Matrizen ($n-1$ gerade) weniger Operationen, wenn $n - 1 \geq 16$ ist.
Durch Induktion über die Anzahl der rekursiven Anwendungen von Strassen's Formeln folgt die Behauptung.

3. Sei $\mathrm{M}'(n)$ die Anzahl arithmetischer Operationen. Dann ist:
$$\mathrm{M}'(n) = \begin{cases} 2n^3 - n^2 & \text{falls } n < 16 \\ 7\mathrm{M}'\left(\frac{n}{2}\right) + \frac{18}{4}n^2 & \text{falls } n \geq 16 \text{ gerade} \\ 7\mathrm{M}'\left(\frac{n-1}{2}\right) + \frac{42}{4}n^2 - 17n + \frac{15}{2} & \text{falls } n \geq 16 \text{ ungerade} \end{cases}$$
Wir definieren für $x \in \mathbb{R}^+$:
$$\overline{\mathrm{M}}(x) = \begin{cases} 2x^3 - x^2 & \text{falls } x < 32 \\ 7\overline{\mathrm{M}}\left(\frac{x}{2}\right) + \frac{42}{4}x^2 & \text{falls } x \geq 32 \end{cases}$$

Dann gilt:

$$\overline{\mathrm{M}}(n) \geq \mathrm{M}'(n) \text{ für alle } n \in \mathbb{N}$$

Weiterhin ist

$$\overline{\mathrm{M}}'(x) = \sum_{i=0}^{k-1} \left[7^i \cdot \tfrac{42}{4} \left(\tfrac{x}{2^i}\right)^2\right] + 7^k \left(2 \cdot \left(\tfrac{x}{2^k}\right)^3 - \left(\tfrac{x}{2^k}\right)^2\right) \text{ für } x \geq 32, \text{ wobei}$$
$$k := \min\left\{l \,\middle|\, \tfrac{x}{2^l} < 32\right\}$$

Mit $k = \lfloor \log_2 x \rfloor - 4 = \log_2 x - t$ für $t \in [4, 5]$ erhalten wir:

$$\overline{\mathrm{M}}(x) \leq 7^{\log_2 x} \left(13 \left(\frac{4}{7}\right)^t + 2 \left(\frac{8}{7}\right)^t\right) \leq 4,8 \cdot 7^{\log_2 x} \text{ für } x \geq 32.$$

Für $x < 32$ direkt nachrechnen.

$\square$