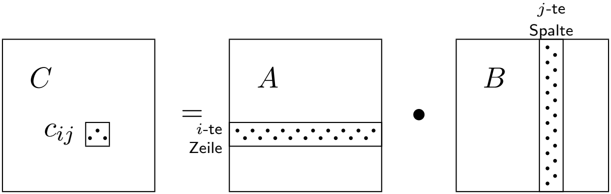
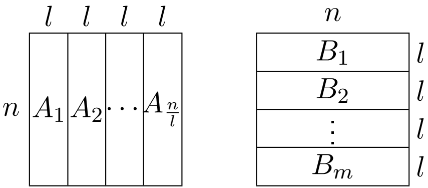
6.3 Der 4-Russen-Algorithmus für boolesche Matrixmultiplikation

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 teilt n).

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



Sei $A = A'_1 \vee A'_2 \vee \ldots \vee A'_m$, $B = B'_1 \vee B'_2 \vee \ldots \vee B'_m$, $C_i := A'_i \cdot B'_i$ für $i = 1, \ldots, m$. Dann gilt

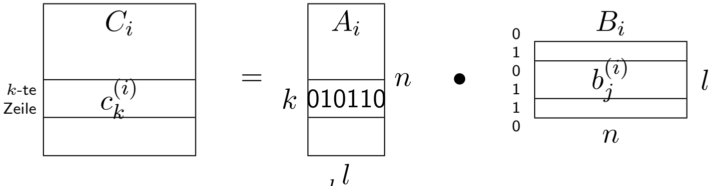
$$C = \bigvee_{i=1}^{m} C_i, \text{ da}$$

$$C = AB = \left(\bigvee_{i=1}^{m} A'_i\right)\left(\bigvee_{i=1}^{m} B'_i\right) = \bigvee_{\substack{1 \le i \le m \\ 1 \le j \le m}} A'_i B'_j = \bigvee_{i=1}^{m} A_i B_i,$$

da $A'_i B'_j = 0$ für $i \neq j$ (A'_i und B'_j sind ja $n \times n$ Matrizen 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 alle überhaupt möglichen $a_k^{(i)}$.

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$[0] := [0, \ldots, 0]$
  for $j := 1$ **to** $2^{\lfloor \log n \rfloor} - 1$ **do**
    $p := \lfloor \log j \rfloor$ **co** p Index der vordersten 1 von j **oc**
    comb$[j]$:=comb$[j - 2^p] \vee b_{(i-1)\lfloor \log n \rfloor + 1 + p}$
  od

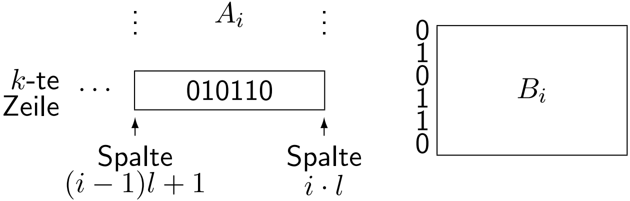
Zeitbedarf:

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

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

```
algorithm four-russians(array a, b, c) =
   co a, b, c als Vektoren von n Zeilenvektoren organisiert oc
   const l = ⌊log n⌋ co wir nehmen an l|n oc
   array comb[0..2^(l−1)] of boolean-vector; int nc
   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 Binärzahl wandeln oc
             nc := 0
             for k := i · l downto (i − 1) · l + 1 do
                  nc := nc + nc+ if a[j, k] then 1 else 0 fi
             od
             c[j] := c[j]∨comb[nc]
        od
   od
```

Beispiel 116



Zeitbedarf:

(a) sequentiell:

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

(b) Vektorrechner der Breite n (Interpretation eines Bitintervalls als Zahl in $\mathcal{O}(1)$ Zeit):

$$\mathcal{O}\left(\frac{n}{l}\cdot(n+n(1+1))\right)=\boxed{\mathcal{O}\left(\frac{n^2}{\log n}\right)\text{ (Vektoroperationen)}}$$

Satz 117

Der 4-Russen-Algorithmus berechnet das Produkt zweier boolescher Matrizen sequentiell in Zeit $\mathcal{O}\left(\frac{n^3}{\log n}\right)$ *bzw. mit* $\mathcal{O}\left(\frac{n^2}{\log n}\right)$ *Bitvektoroperationen der Breite* n.

Beweis:

s.o. $\square$

V.L. Arlazarov, E.A. Dinic, M.A. Kronrod, I.A. Faradzev:
On economical construction of the transitive closure of an oriented graph
Soviet Math. Dokl. **11**, pp. 1209–1210 (1970)

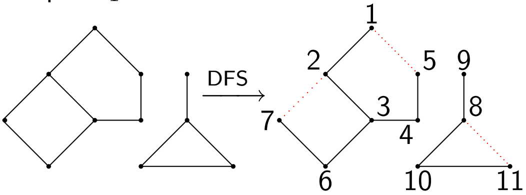
6.4 Transitive Hülle und DFS

Wir erinnern uns an den DFS-Algorithmus:

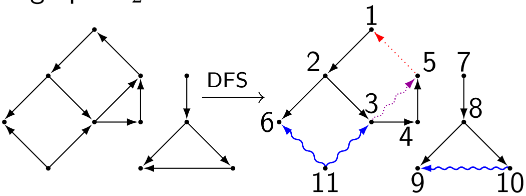
for all nodes v **do** unmark v **od**
$count := 0$
while $\exists$ unvisited v **do**
    $r :=$ pick (random) unvisited node
    push r onto stack
    while stack $\neq \emptyset$ **do**
        $v :=$ pop top element
        if v unvisited **then**
            mark v visited
            push all neighbours of v onto stack
            num[v]$:= ++count$
        fi
    od
od

Beispiel 118

Graph G_1:



Digraph G_2:

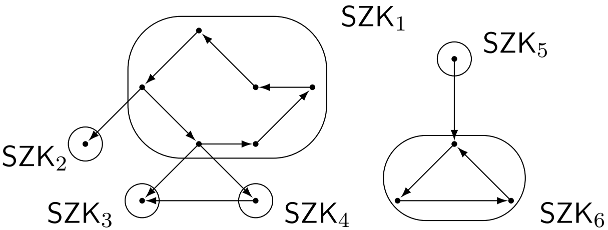


Bezeichnungen:

- Baumkante
- Rückwärtskante
- Querkante
- Vorwärtskante

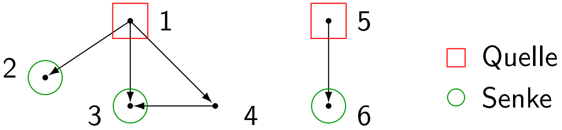
Bem.: Alle Baumkanten zusammen: DFS-Wald (Spannwald).

Beispiel 119 (Starke Zusammenhangskomponenten (in Digraphen))



Schrumpfen $\Downarrow$ der SZK's

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

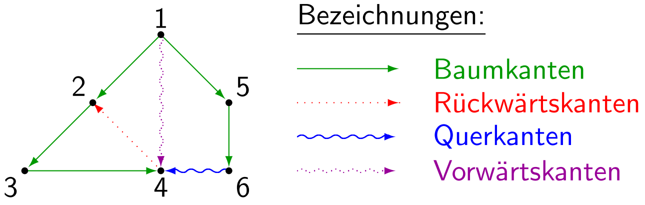


DFS in ungerichteten Graphen mit c Zusammenhangskomponenten, n Knoten, m Kanten: $n-c$ Baumkanten, $m-n+c$ Rückwärtskanten, Zeitaufwand $\mathcal{O}(n+m)$

$$
\begin{array}{cc}
\mathbf{A}^* & \begin{array}{cc} \underline{\quad \mathrm{ZK}_1 \quad} & \underline{\mathrm{ZK}_2} \end{array} \\
\begin{array}{c} \mathrm{ZK}_1 \\ \\ \mathrm{ZK}_2 \end{array} &
\begin{pmatrix}
\boxed{\begin{array}{cccc} 1 & 1 & 1 & 1 \\ 1 & & & 1 \\ & \multicolumn{2}{c}{1} & \\ 1 & & & 1 \\ 1 & 1 & 1 & 1 \end{array}} & 0 \\
0 & \begin{array}{cc} \boxed{} & \\ & \boxed{} \end{array}
\end{pmatrix}
\end{array}
$$

A^* aus DFS mit Aufwand $\Theta(n^2)$

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



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

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 $\mathcal{O}(n \cdot (m+n))$. Sehr gut für dünne Graphen.

7. Ein besserer Algorithmus für das apsd-Problem in ungewichteten Digraphen

Sei $G = (V, E)$ ein Digraph mit der Knotenmenge $\{0, \ldots, n-1\}$, dessen Kanten alle die Länge 1 haben.

Sei $D = (d_{ij})_{0 \leq i,j < n}$ die zugehörige Matrix, mit Einträgen $d_{ij} \in \{0, 1, \infty\}$.

Entsprechend sei D^* die Distanzmatrix, so dass

$$d^*_{ij} = \text{Länge eines kürzesten Wegs zwischen } i \text{ und } j$$

Setze weiterhin

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

Algorithmus a1

co Sei $A = (a_{ij})_{0 \le i,j < n}$ mit $a_{ij} = \begin{cases} 1 \text{ falls } d_{ij} \le 1 \\ 0 \text{ sonst} \end{cases}$ **oc**

$B := I$ **co** boolesche Einheitsmatrix **oc**
for $l := 2$ **to** $r + 1$ **do**
    $B := A \cdot B$
    for $0 \le i, j < n$ **do**
        if $b_{ij} = 1$ **then** $d_{ij}^{(l)} := l$ **else** $d_{ij}^{(l)} := \infty$ **fi**
        if $d_{ij}^{(l-1)} \le l$ **then** $d_{ij}^{(l)} := d_{ij}^{(l-1)}$ **fi**
    od
od

Dieser Algorithmus berechnet $D^{(1)} = D$, $D^{(2)}$, $D^{(3)}, \ldots, D^{(r)}$.

Sei ω eine Zahl ≥ 2, so dass die Matrixmultiplikation in Zeit $\mathcal{O}(n^\omega)$ durchgeführt werden kann (Winograd/Coppersmith: $\omega \leq 2,376$).

Zeitaufwand für Algorithmus a1: $\boxed{\mathcal{O}(rn^\omega)}$

Algorithmus apsd =

Berechne, mit Algorithmus a1, $D^{(l)}$ 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 Zeile i von $D^{(l)}$ das d, $\lceil \frac{l}{2} \rceil \leq d \leq l$, das in dieser Zeile am wenigsten oft vorkommt

    $S_i :=$ Menge der zugehörigen Spaltenindizes

  od

  $l_1 := \lceil \frac{3}{2} l \rceil$

  for $0 \leq i, j < n$ **do**

    $m_{ij} :=$ **if** $S_i \neq \emptyset$ **then** $\min\limits_{k \in S_i} \{d_{ik}^{(l)} + d_{kj}^{(l)}\}$ **else** ∞ **fi**

    if $d_{ij}^{(l)} \leq l$ **then** $d_{ij}^{(l_1)} := d_{ij}^{(l)}$

    elif $m_{ij} \leq l_1$ **then** $d_{ij}^{(l_1)} = m_{ij}$ **else** $d_{ij}^{(l_1)} := \infty$ **fi**

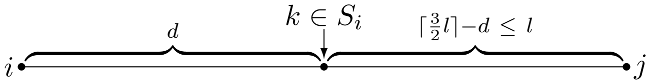
  od

  $l := l_1$

od

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

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



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 $r = n^{\frac{3-\omega}{2}}$. Damit ergibt sich für apsd die Laufzeit $\mathcal{O}\left(n^{\frac{3+\omega}{2}}\right)$.

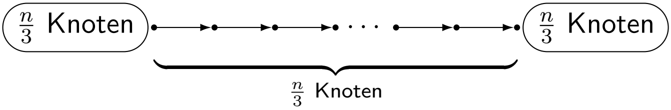
Satz 120

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:

$\surd$ $\square$

Bemerkung: Beim apsp kann die Größe der Ausgabe $\Omega(n^3)$ sein:



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

Satz 121

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.*

Beweis:

Idee: Ersetze $u \xrightarrow{M' \leq M} v$ durch:

$$u = u_0 \longrightarrow u_1 \longrightarrow u_2 \cdots \longrightarrow u_{M'} = v$$

□