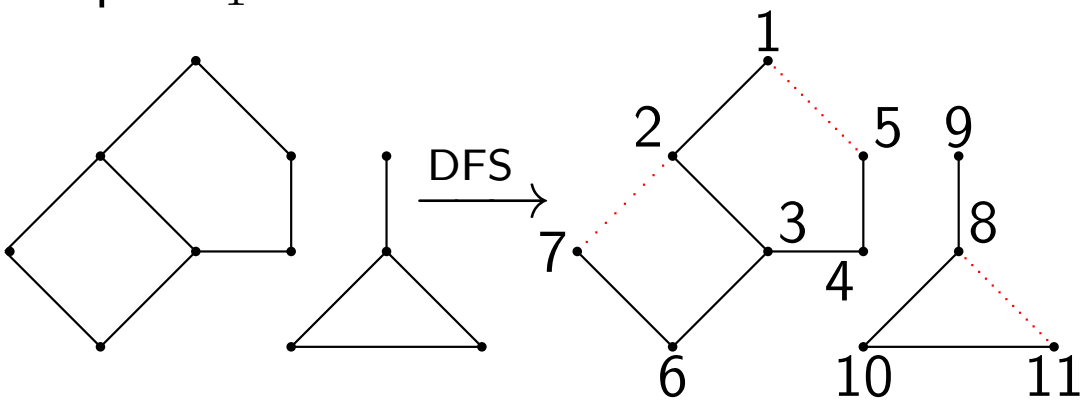
6.4 Transitive Hülle und DFS

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

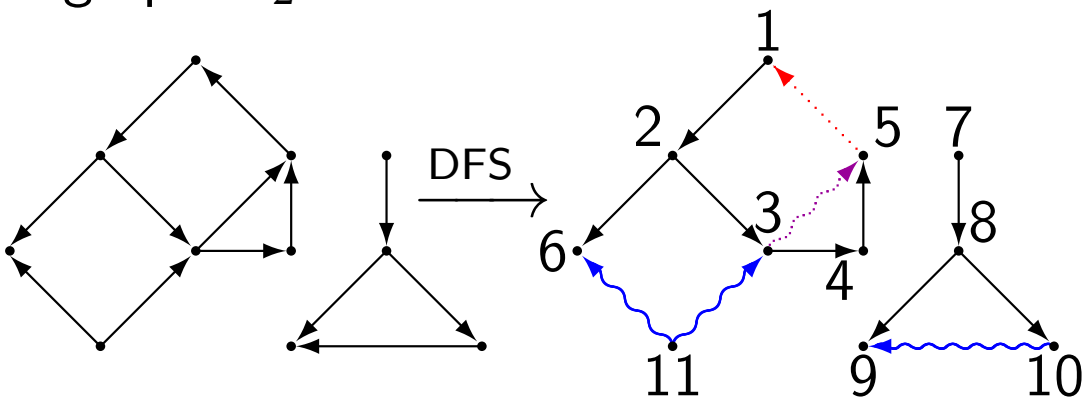
Graph G_1:



Bezeichnungen:

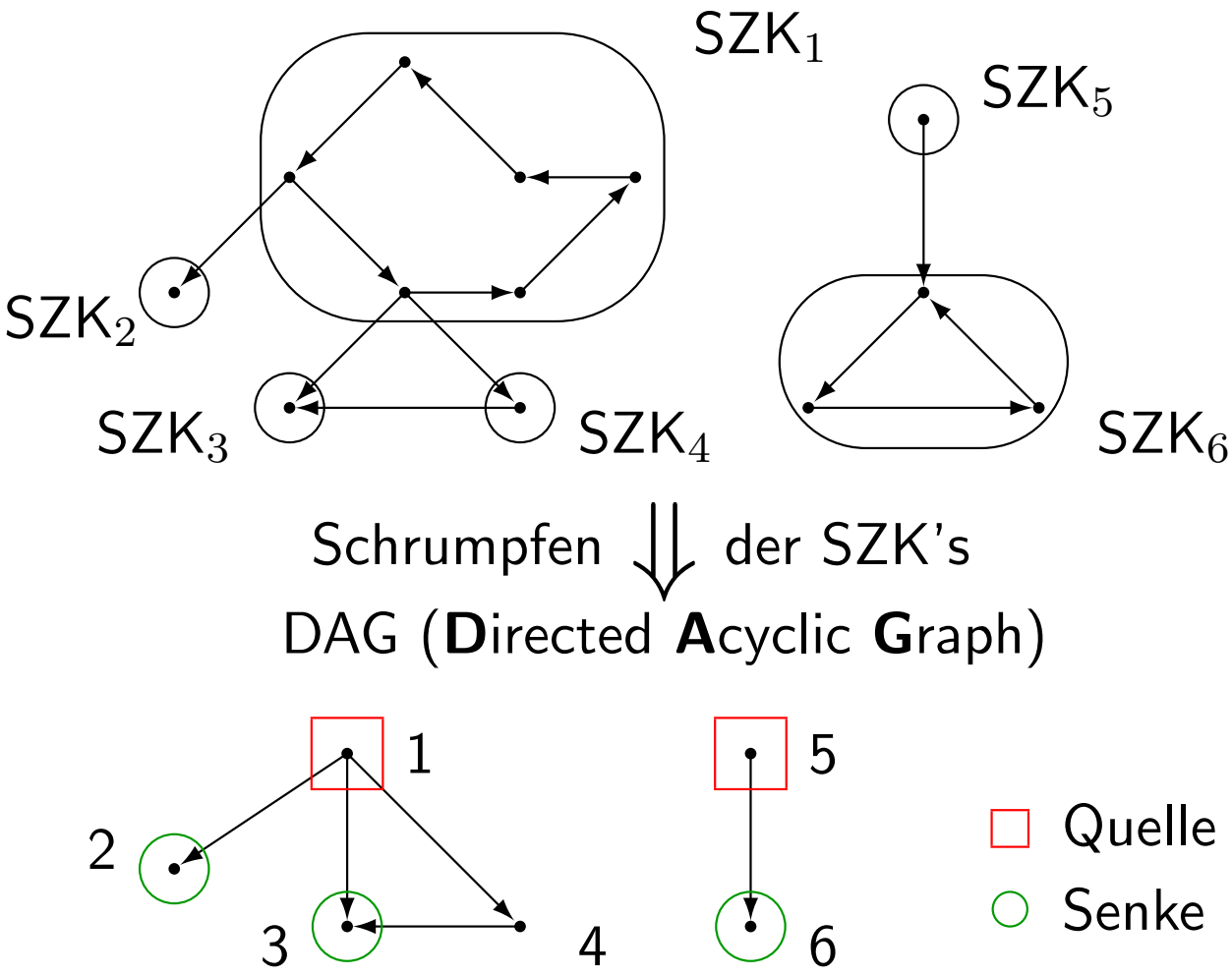
- Baumkante
- Rückwärtskante
- Querkante
- Vorwärtskante

Digraph G_2:

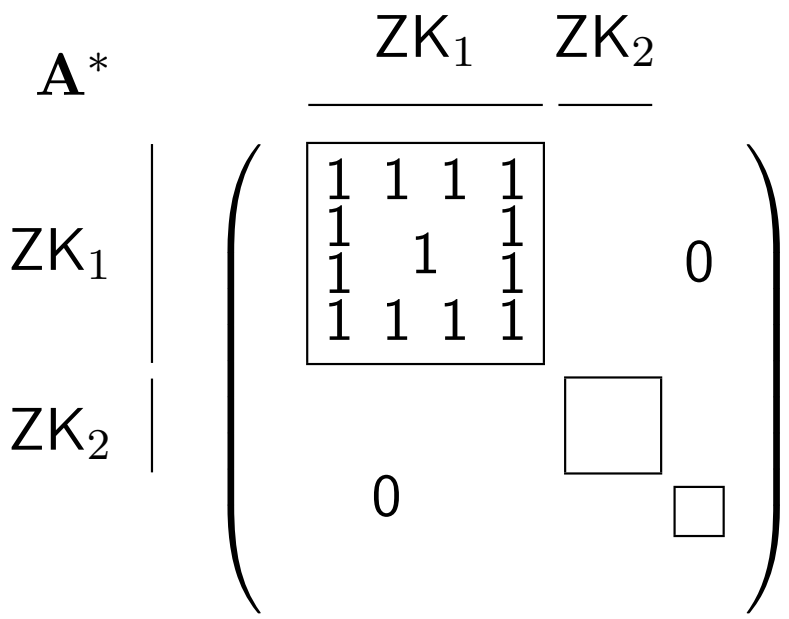


Bem.: Alle Baumkanten zusammen: DFS-Wald (Spannwald).

Beispiel 119 (Starke Zusammenhangskomponenten (in Digraphen))

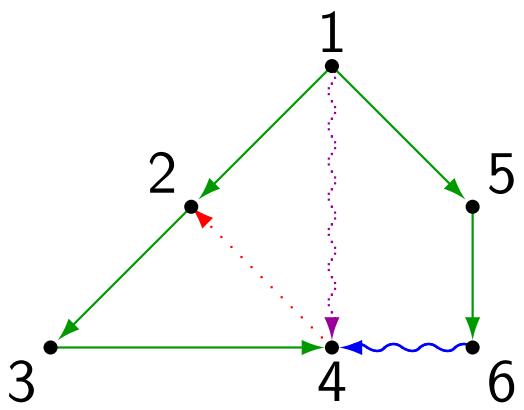


DFS in ungerichteten Graphen mit c Zusammenhangskomponenten, n Knoten, m Kanten: $n - c$ Baumkanten, $m - n + c$ Rückwärtskanten, Zeitaufwand $\mathcal{O}(n + m)$



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

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



Bezeichnungen:

Baumkanten
Rückwärtskanten
Querkanten
Vorwärtskanten

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

Entsprechend sei D^* die Distanzmatrix, so dass

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

Setze weiterhin

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

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**

  $B := A \cdot B$ **co** boolesches Matrixprodukt **oc**

  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, $\left\lceil \frac{l}{2} \right\rceil \leq d \leq l$, das in dieser Zeile am wenigsten oft vorkommt

    $S_i :=$ Menge der zugehörigen Spaltenindizes

  od

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

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

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

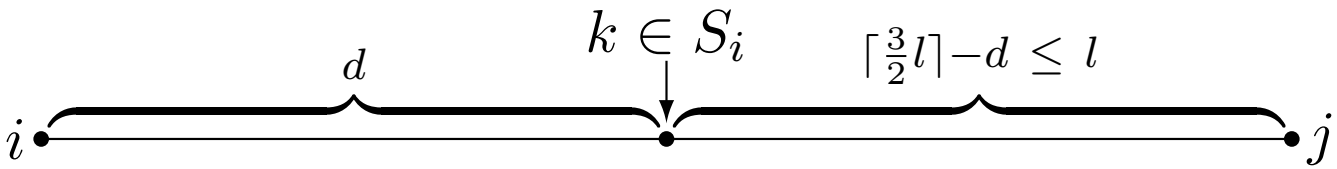
  od

  $l := l_1$

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 $\mathcal{O}\left(n^{\frac{3+\omega}{2}}\right)$.

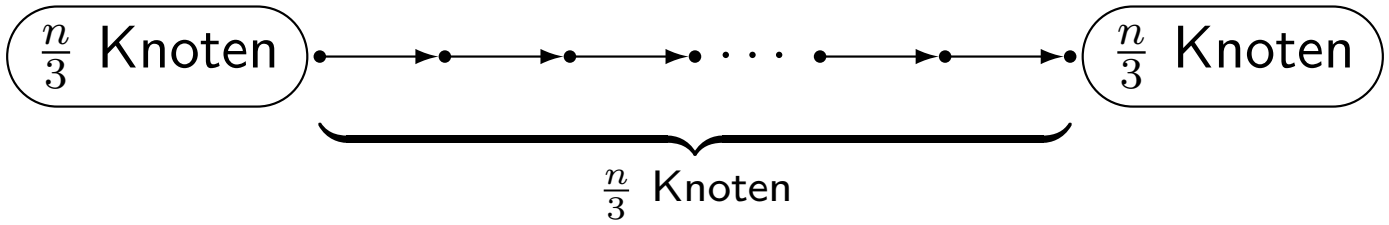
Satz 120

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:

$\surd$ □

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

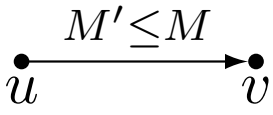


Verwende stattdessen die kürzeste-Pfade-Nachfolger-Matrix $N \in [0, \ldots, n-1]^{n \times n}$, wo n_{ij} die Nummer des zweiten Knoten auf „dem“ kürzesten Pfad von Knoten i zu Knoten j ist.
Eine solche Matrixdarstellung für die kürzesten Pfade zwischen allen Knotenpaaren kann in Zeit $\mathcal{O}(n^{\frac{3+\omega}{2}}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

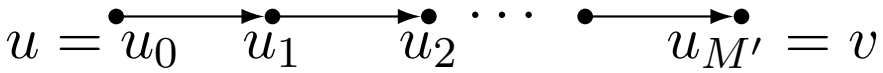
Satz 121

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 

durch:



□

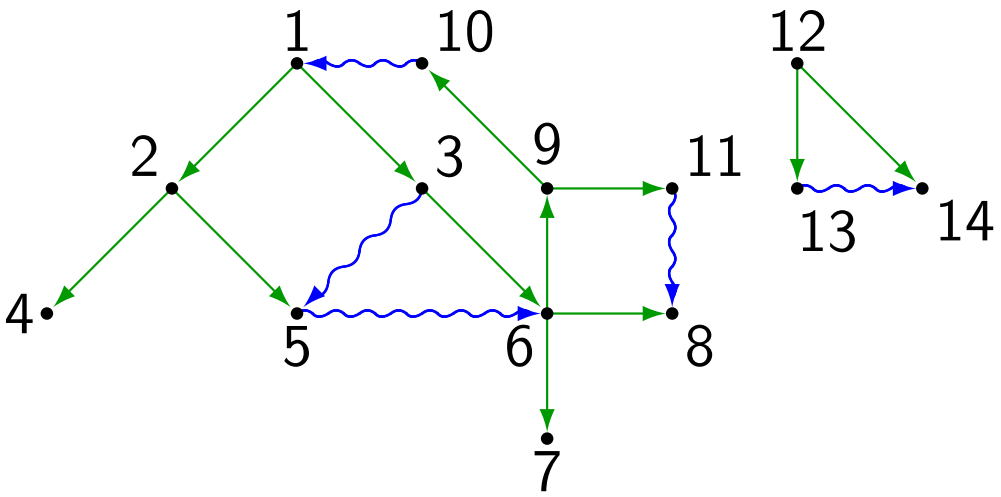
8. Ein Algorithmus für die transitive Hülle in Digraphen mit linearer erwarteter Laufzeit

Wir nehmen an, dass die Wahrscheinlichkeit eines Digraphen mit n Knoten und m Kanten eine Funktion (nur) von n und m ist.
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 Digraphen mit n Knoten und m Kanten betrachten.

Erinnerung: 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 122



Bezeichnungen:

Baumkanten

Querkanten

algorithm exp-lin-transitive-closure

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

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

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