

# Graphalgorithmen

## Vertretung durch Rainer Feldmann

Vorlesung vom 22. Oktober 2008

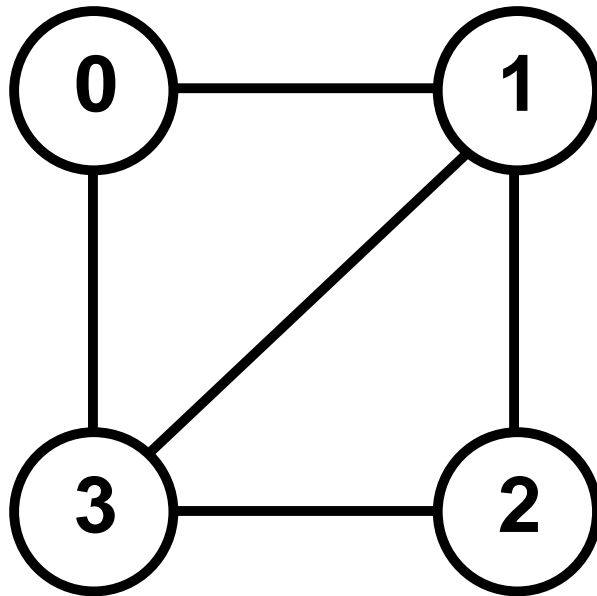
### Programm des Tages:

- Berechnung kürzester Wege



# 1. Voraussetzungen

- Wir betrachten ungerichtete Graphen  $G(V, E)$  mit Knotenmenge  $V = \{1, \dots, n\}$ .
- Ein Graph  $G(V, E)$  ist gegeben durch seine Adjazenzmatrix  $A$ .



$$A = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

## Kürzeste Wege und Matrixmultiplikation

**Definition 1:** Es sei  $G(V, E)$  ein Graph,  $W = (w_{ij})$  eine Gewichtsmatrix, d.h.  $w_{ij}$  ist das Gewicht der Kante von  $i$  nach  $j$ . Definiere  $\delta_{ij}^{(k)}$  als Gewicht des bzgl. des Gewichtes kürzesten Weges von  $i$  nach  $j$  mit höchstens  $k$  Kanten.

**Bemerkung 1:** Es sei  $G(V, E)$  ein Graph und  $W$  eine Gewichtsmatrix für  $G$ . Dann gilt

$$\delta_{ij}^{(0)} = \begin{cases} 0 & i = j \\ \infty & i \neq j \end{cases} \quad \text{und} \quad \delta_{ij}^{(m)} = \min_{1 \leq k \leq n} \{ \delta_{ik}^{(m-1)} + w_{kj} \}, \quad m \geq 1.$$

Ausserdem ist  $\delta(i, j) = \delta_{ij}^{(n-1)}$ .



**Matrixmultiplikation:** Es seien  $D, W \in \mathbb{R}^{n \times n}$ .

$$(D \cdot W)_{ij} = \sum_{k=1}^n D_{ik} \cdot W_{kj}$$

Ersetze  $\cdot$  durch  $+$  und  $+$  durch  $\min$ . Dann gilt für  $D, W \in (\mathbb{R} \cup \{\infty\})^{n \times n}$ :

$$(D * W)_{ij} = \min_{1 \leq k \leq n} \{D_{ik} + W_{kj}\}$$

**Idee:** Berechne  $D^{(n-1)} = (\delta_{ij}^{(n-1)})$  durch fortgesetzte Matrixmultiplikation:

$$D^{(1)} = W, \quad D^{(2)} = W^2 = W * W, \quad D^{(4)} = W^4 = W^2 * W^2, \quad \dots$$

$$D^{(2^{\lceil \log(n-1) \rceil})} = W^{2^{\lceil \log(n-1) \rceil}} = W^{2^{\lceil \log(n-1) \rceil - 1}} * W^{2^{\lceil \log(n-1) \rceil - 1}}$$



## Algorithm:

```
MATRIXMULTIPLY(weight matrix A,B) {  
  for (i = 1; i ≤ n; i++)  
    for (j = 1; j ≤ n; j++) {  
      Cij = ∞;  
      for (k = 1; k ≤ n; k++)  
        Cij = min{Cij, Aik + Bkj};  
    } /* for all i, j */  
  return C;  
} /* MATRIXMULTIPLY */
```

```
MMAPD(weight matrix W) {  
  D(1) = W;  
  m = 1;  
  while (m < n - 1) {  
    D(2m) = matrixMultiply(D(m), D(m));  
    m = 2m;  
  } /* while ... */  
  return D(m);  
} /* MMAPD */
```

- Die Laufzeit des Algorithmus MMAPD ist  $O(\text{MM}(n) \cdot \log n)$ , wobei  $\text{MM}(n)$  die Zeit ist, um  $2n \times n$  Matrizen miteinander zu multiplizieren.  
(Beste bekannte Algorithmen zur Zeit:  $\text{MM}(n) \approx n^{2.376}$ ).
- **Nachteil:** Keine implizite Darstellung der kürzesten Wege, denn dazu wären alle Matrizen  $D^1, \dots, D^m$  notwendig und die Laufzeit würde auf  $O(\text{MM}(n) \cdot n)$  ansteigen.



## 2. Distanzberechnungen

### Problem APD:

**geg:** Ungerichteter, ungewichteter Graph  $G(V, E)$ .

**ges:** Matrix  $D = (\delta_{ij})$ , wobei  $\delta_{ij}$  die Distanz von  $i$  nach  $j$  in  $G$  ist.

### Problem APSP:

**geg:** Ungerichteter, ungewichteter Graph  $G(V, E)$ .

**ges:** Matrix  $D = (\delta_{ij})$  wie bei APD und eine implizite Darstellung der kürzesten Wege.

Im folgenden:

- Lösung von APD in Zeit  $O(\text{MM}(n) \cdot \log n)$  für ungewichtete, ungerichtete Graphen
- Lösung von APSP in Zeit  $O(\text{MM}(n) \cdot \log^2 n)$  für ungewichtete, ungerichtete Graphen durch einen randomisierten Algorithmus.



**Bemerkung 2:** Es sei  $G(V, E)$  ungerichteter, ungewichteter Graph mit Adjazenzmatrix  $A$ . Ist  $Z = A^2$ , so ist  $Z_{ij} > 0$  genau dann wenn es einen Weg der Länge 2 von  $i$  nach  $j$  gibt.  $Z_{ij}$  ist die Anzahl der verschiedenen Wege der Länge 2 von  $i$  nach  $j$ .

**Damit:** Wir können durch eine einfache Matrixmultiplikation eine Matrix  $Z$  berechnen, die es erlaubt, schnell einen Graphen  $G'$  zu berechnen, in dem zwei Knoten  $i$  und  $j$  miteinander verbunden sind genau dann, wenn  $\delta(i, j) \leq 2$  in  $G$  ist:

$$Z = A^2;$$

$$A'_{ij} = \begin{cases} 1, & \text{falls } i \neq j \text{ und } (A_{ij} = 1 \text{ oder } Z_{ij} > 0) \\ 0, & \text{sonst} \end{cases}$$

$G' =$  der durch  $A'$  induzierte Graph



**Lemma 1:** Es sei  $G(V, E)$  ungerichteter, ungewichteter Graph mit Adjazenzmatrix  $A$ . Ist  $A'$  so dass  $A'_{ij} = 1$  gdw.  $i \neq j$  und  $(A_{ij} = 1$  oder  $Z_{ij} > 0)$  und ist  $D'$  die Distanzmatrix des durch  $A'$  gegebenen Graphen  $G'$ , so ist

- a)  $D_{ij} = 2D'_{ij}$ , wenn  $D_{ij}$  gerade ist und
- b)  $D_{ij} = 2D'_{ij} - 1$ , wenn  $D_{ij}$  ungerade.

**Idee** für einen Algorithmus:

1. Berechne die Distanzen  $D'$  für  $G'$  (rekursiv).
2. Berechne die Parität der Distanzen in  $G$ .
3. Dann liefert Lemma 1 eine Berechnungsvorschrift für die Distanzen  $D$  in  $G$ .



### Lemma 2:

- a) Für alle Nachbarn  $k$  von  $i$  gilt  $D_{ij} - 1 \leq D_{kj} \leq D_{ij} + 1$ .
- b) Es existiert eine Nachbar  $k$  von  $i$  so dass  $D_{kj} = D_{ij} - 1$ .

### Lemma 3:

- a) Ist  $D_{ij}$  gerade, so ist  $D'_{kj} \geq D'_{ij}$  für alle Nachbarn  $k \in \Gamma(i)$ .
- b) Ist  $D_{ij}$  ungerade, so ist  $D'_{kj} \leq D'_{ij}$  für alle Nachbarn  $k \in \Gamma(i)$  und es existiert ein  $k \in \Gamma(i)$ , so daß  $D'_{kj} < D'_{ij}$

### Lemma 4:

- a)  $D_{ij}$  ist gerade, genau dann wenn  $\sum_{k \in \Gamma(i)} D'_{kj} \geq D'_{ij} \cdot \text{deg}(i)$ .
- b)  $D_{ij}$  ist ungerade, genau dann wenn  $\sum_{k \in \Gamma(i)} D'_{kj} < D'_{ij} \cdot \text{deg}(i)$ .



```

APD(Adjazenzmatrix  $A$ ) {
  /* berechnet für alle Knotenpaare  $(i, j)$  die Distanzen  $D_{ij}$  */
  /* im ungerichteten Graphen  $G$  gegeben durch  $A$  */
   $Z = A^2$ ;
  berechne  $A'$  mit /*  $A'_{ij} = 1 \Leftrightarrow$  es ex. Weg der Länge 1 oder 2 von  $i$  nach  $j$  */
     $A'_{ij} = 1$  gdw.  $i \neq j$  und  $(A_{ij} = 1$  oder  $Z_{ij} > 0)$ ;
  if  $A'_{ij} = 1 \forall i \neq j$  then return  $D = 2A' - A$ ; /* Rekursionsabbruch */
   $D' = \text{APD}(A')$ ; /* Rekursion */
   $S = A \cdot D'$ ; /*  $S_{ij} = \sum_{k \in \Gamma(i)} D'_{kj}$  */

  return  $D$  mit  $D_{ij} = \begin{cases} 2D'_{ij} & \text{falls } S_{ij} \geq D'_{ij}Z_{ii} \\ 2D'_{ij} - 1 & \text{falls } S_{ij} < D'_{ij}Z_{ii} \end{cases}$ ;
} /* APD */

```

**Satz 1:** Algorithmus APD berechnet die Distanzmatrix  $D$  eines  $n$ -Knoten Graphen  $G$  in Zeit  $O(\text{MM}(n) \log n)$  mit Hilfe der Multiplikation ganzzahliger Matrizen mit Zahlen die  $O(n^2)$  größenbeschränkt sind.



### 3. Das Zeugenproblem für Boolesche Matrizen

**Nun:** Erweiterung des Algorithmus APD so, dass nicht nur die Distanzen  $D_{ij}$  sondern auch für jedes Knotenpaar  $(i, j)$  ein kürzester Weg berechnet wird. Dabei soll die Laufzeit des neuen Algorithmus subkubisch sein.

**Problem:** Es gibt Graphen  $G(V, E)$ , so dass  $\Omega(n^2)$  Knotenpaare Distanz  $\Omega(n)$  haben:

**Beispiel 1:**  $G(V, E)$  ist eine „Linie“:  $V = \{1, \dots, n\}$ ,  $E = \{(i, i + 1) \mid 1 \leq i < n\}$ .  
Setze  $\delta = \frac{n}{2} = \Omega(n)$ . Dann haben

$$\sum_{k=1}^{n-\delta} k = \frac{(n - \delta + 1)(n - \delta)}{2} > \frac{n^2}{8} = \Omega(n^2)$$

Knotenpaare Abstand mindestens  $\delta$  zueinander.

Explizites Speichern der kürzesten Wege würde daher Platzbedarf  $\Omega(n^3)$ , also auch Zeitaufwand  $\Omega(n^3)$  bedeuten.



**Daher:** kürzeste Wege werden nur implizit gespeichert, indem zu jedem Knotenpaar  $(i, j)$  der Nachfolger  $S_{ij}$  von  $i$  auf einem kürzesten Weg nach  $j$  gespeichert wird.

$S = (S_{ij})$  hat  $O(n^2)$  Einträge und ein kürzester Weg von  $i$  nach  $j$  kann in Zeit proportional zu seiner Länge bestimmt werden:

```
 $x_0 = i; k = 0;$   
repeat  
     $x_{k+1} = S_{x_k, j}; k ++;$   
until  $(x_k = j);$ 
```

**Notation 1:** Wir bezeichnen im folgenden

- die boolesche Matrixmultiplikation mit  $\wedge$  als Multiplikation und  $\vee$  als Addition durch „\*“ und
- die ganzzahlige Matrixmultiplikation mit „·“.

**Definition 2:** Es seien  $A, B \in \{0, 1\}^{n \times n}$  boolesche Matrizen,  $P = A * B \in \{0, 1\}^{n \times n}$  das boolesche Produkt von  $A$  und  $B$ . Ein **Zeuge für  $P_{ij}$**  ist ein Index  $k \in \{1, \dots, n\}$ , so dass  $A_{ik} = 1 = B_{kj}$ .

**Bemerkung 3:**

- a)  $P_{ij} = 1 \Leftrightarrow$  es gibt einen Zeugen  $k$  für  $P_{ij}$ .
- b) Ist  $C = A \cdot B$ , so ist  $C_{ij}$  die Anzahl der Zeugen für  $P_{ij}$ .
- c) Ist  $A$  Adjazenzmatrix eines Graphen  $G(V, E)$ ,  $P = A * A$ ,  $C = AA$ , so ist  $P_{ij} = 1$  gdw. es ex. ein Weg der Länge 2 zwischen  $i$  und  $j$ .  $C_{ij}$  ist die Anzahl dieser Wege (vgl. Lemma 2).
- d) Ein Zeuge  $k$  für  $P_{ij} = 1$  ist Zwischenknoten auf einem Weg der Länge 2 von  $i$  nach  $j$ .
- e) Für jedes  $P_{ij}$  kann es bis zu  $n$  Zeugen geben.

Zeugen sind interessant,  
wenn man den APD-Algorithmus so erweitern will, dass er kürzeste Wege berechnet.



**Definition 3:** Es seien  $A, B \in \{0, 1\}^{n \times n}$  boolesche Matrizen,  $P = A * B \in \{0, 1\}^{n \times n}$ . Eine **Zeugenmatrix (BPWM) für  $P$**  ist eine Matrix  $W \in \{0, \dots, n\}^{n \times n}$  mit

$$W_{ij} = \begin{cases} 0, & \text{wenn } P_{ij} = 0 \\ k, & k \text{ ist Zeuge für } P_{ij} = 1 \end{cases}$$

BPWM steht für „Boolean Product Witness Matrix“.

**Problem BPWM:**

**geg:** boolesche Matrizen  $A$  und  $B$ .

**ges:** eine Zeugenmatrix  $W$  für  $P = A * B$ .

Ein trivialer Algorithmus für BPWM (für alle  $(i, j) \in \{1, \dots, n\}^2$  teste alle  $k \in \{1, \dots, n\}$ ) hat Zeitaufwand  $O(n^3)$ .

Wir vereinfachen daher zunächst das Problem BPWM:



## Problem UNIQUEBPWM:

**geg:** boolesche Matrizen  $A$  und  $B$ , so dass jeder Eintrag in  $P = A * B$  einen eindeutigen Zeugen hat.

**ges:** die Zeugenmatrix  $W$  für  $P = A * B$ .

**Lemma 5:** Es seien  $A, B \in \{0, 1\}^{n \times n}$  boolesche Matrizen,  $\hat{A}$  mit  $\hat{A}_{ik} = kA_{ik}$  und  $W = \hat{A}B$ .

a)  $W_{ij}$  ist Zeuge für  $P_{ij}$ , wenn  $P_{ij}$  eindeutigen Zeugen hat.

b) Hat jeder Eintrag von  $P_{ij}$  einen eindeutigen Zeugen, so ist  $W$  Lösung von UNIQUEBPWM.

**Beispiel 2:**  $A = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$ ,  $B = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$ ,  $P = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$ .

$$\hat{A} = \begin{pmatrix} 1 & 2 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \text{ und } W = \begin{pmatrix} 3 & 2 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Die **grünen** und **blauen** Einträge von  $W$  sind schon korrekte Einträge für eine Zeugenmatrix, der **rote** jedoch nicht.



**Folgerung 1:** Problem UNIQUEBPWM kann durch eine ganzzahlige Matrixmultiplikation, also in Zeit  $O(\text{MM}(n))$  gelöst werden.

**Leider:** Wir wollen eine Zeugenmatrix  $W$  für  $P = A * A$  mit einer Adjazenzmatrix  $A$  berechnen. Dabei ist natürlich nicht garantiert, dass die Zeugen immer eindeutig sind.

**Idee:**

- Randomisierung liefert gleichen Effekt wie eindeutige Zeugen für genügend viele Einträge  $P_{ij}$ .
- Die wenigen verbleibenden Einträge werden dann mit dem trivialen Algorithmus berechnet.



## Nicht eindeutige Zeugen

Betrachte boolesche Matrizen  $A, B \in \{0, 1\}^{n \times n}$ ,  $P = A * B$ ,  $C = AB$  und einen festen Eintrag  $P_{ij}$  von  $P$ .

Sei  $w = C_{ij}$  die Anzahl der Zeugen für  $P_{ij}$  (OE  $w \geq 2$ , denn nach den Vorbetrachtungen ist es einfach, einen eindeutigen Zeugen zu finden).

Sei  $r \in \mathbb{N}$  mit  $\frac{n}{2} \leq wr \leq n$ .

**Ziel:** Finde eine Menge  $R \subset \{1, \dots, n\}$  mit  $|R| = r$ , die eindeutigen Zeugen  $k$  für  $P_{ij}$  enthält.



Ziel: Finde eine Menge  $R \subset \{1, \dots, n\}$  mit  $|R| = r$ , die eindeutigen Zeugen  $k$  für  $P_{ij}$  enthält.

**Lemma 6:** Es sei  $n \in \mathbb{N}$ ,  $w \in \{1, \dots, n\}$ ,  $r \in \mathbb{N}$  mit  $\frac{n}{2} \leq wr \leq n$ . Eine Urne enthalte  $n$  Bälle, von denen  $w$  weiss und  $n - w$  schwarz sind. Zieht man zufällig  $r$  Bälle aus der Urne ohne diese zwischendurch zurückzulegen, so ist

$$\Pr[\text{genau ein weisser Ball wurde gezogen}] \geq \frac{1}{2e}$$

**Beispiel 3:**  $n = 8$ ,  $w = 3$ ,  $n - w = 5$ . Einzig mögliche Wahl für  $r$  mit  $\frac{n}{2} \leq wr \leq n$  ist  $r = 2$ :

- 3 Möglichkeiten, genau 2 weisse Bälle zu ziehen
- 10 Möglichkeiten, genau 2 schwarze Bälle zu ziehen
- 15 Möglichkeiten, genau 1 weissen und 1 schwarzen Ball zu ziehen
- 28 Möglichkeiten insgesamt

Damit ist  $\Pr[\text{genau ein weisser Ball wurde gezogen}] = \frac{28-13}{28} = \frac{15}{28} > \frac{1}{2} > \frac{1}{2e} \approx 0.184$ .



Sei jetzt  $R \subset \{1, \dots, n\}$  so, dass  $R$  einen eindeutigen Zeugen für  $P_{ij}$  enthält.

$R$  sei gegeben durch einen Vektor  $R \in \{0, 1\}^n$  mit  $R_k = 1$  gdw.  $k \in R$ . Definiere

- eine Matrix  $A^R \in \mathbb{N}_0^{n \times n}$  durch  $A_{ik}^R = k R_k A_{ik}$  und
- eine Matrix  $B^R \in \mathbb{N}_0^{n \times n}$  durch  $B_{kj}^R = R_k B_{kj}$

Im Vergleich zu Lemma 5:  $\hat{A}$  mit  $\hat{A}_{ik} = k A_{ik}$  und Matrix  $B$ ,  $W = \hat{A}B$ .

$A^R$  ist die Matrix, die aus  $\hat{A}$  entsteht, indem jede Spalte von  $\hat{A}$ , die zu einem Element  $k \notin R$  gehört, durch eine Nullspalte ersetzt wird.

$B^R$  ist die Matrix, die aus  $B$  entsteht, indem jede Zeile von  $B$ , die zu einem Element  $k \notin R$  gehört, durch eine Nullzeile ersetzt wird.

Es gilt

**Lemma 7:** Es seien  $A, B \in \{0, 1\}^{n \times n}$  boolsche Matrizen,  $A^R$  und  $B^R$  wie oben,  $W = A^R B^R$ . Wenn  $P_{ij}$  eindeutigen Zeugen in  $R$  hat, so ist  $W_{ij}$  Zeuge für  $P_{ij}$ . (Beweis wie Lemma 5).



## Idee:

- $W = A^R B^R$  liefert Zeugen für alle Einträge in  $P = A * B$ , die einen eindeutigen Zeugen in  $R$  haben.
- Nach Lemma 6 ist die Wahrscheinlichkeit dafür, dass eine zufällig gewählte Menge  $R$  der Größe  $|R| = r$  einen eindeutigen Zeugen für einen Eintrag von  $P$  mit  $w$  Zeugen enthält größer oder gleich der Konstanten  $\frac{1}{2e}$ , wenn  $\frac{n}{2} \leq wr \leq n$  ist.
- Wir wählen  $O(\log n)$  Mengen  $R$  zufällig aus. Damit wird es sehr unwahrscheinlich, dass für einen Eintrag von  $P$  kein Zeuge identifiziert wird.
- Zeugen für die Einträge, für die kein Zeuge identifiziert wird, werden mit dem trivialen Algorithmus berechnet.
- Da nicht alle Einträge von  $P$  die gleiche Anzahl Zeugen haben, müssen Mengen  $R$  mit verschiedenen Kardinalitäten  $r$  benutzt werden. Es reicht jedoch, für  $r$  die Zweierpotenzen zwischen 1 und  $n$  zu benutzen, da nur die Bedingung  $\frac{n}{2} \leq wr \leq n$  erfüllt sein muss.



```

BPWM(Boolsche Matrizen  $A, B \in \{0, 1\}^{n \times n}$ ) {      /* berechnet eine Zeugenmatrix  $W$  für  $P = A * B$  */
   $W = -AB$ ;                                          /*  $W_{ij} < 0 \Leftrightarrow$  Zeuge für  $P_{ij}$  muss noch gefunden werden */
  for  $t = 0, \dots, \lfloor \log n \rfloor$  do           /* teste alle Zweierpotenzen  $r = 2^t$  zwischen 1 und  $n$  */
     $r = 2^t$ ;
    repeat  $\lceil 3.77 \log n \rceil$  times              /* für  $O(\log n)$  Runden */
      wähle  $R \subseteq \{1, \dots, n\}$  mit  $|R| = r$ ;
      berechne  $A^R$  und  $B^R$ ;                          /* vgl. Lemma 7 */
       $Z = A^R B^R$ ;
      for all  $(i, j)$  do                               /* teste ob neuer Zeuge für  $P_{ij}$  gefunden wurde */
        if  $W_{ij} < 0$  und  $Z_{ij}$  ist Zeuge für  $P_{ij}$  then  $W_{ij} = Z_{ij}$ ;
      for all  $(i, j)$  do
        if  $W_{ij} < 0$  then                               /* kein Zeuge für  $P_{ij}$  gefunden */
          berechne  $W_{ij}$  mit dem trivialen Algorithmus;
    } /* BPWM */

```

**Satz 2:** Algorithmus BPWM ist ein Las Vegas Algorithmus für Problem BPWM mit erwarteter Laufzeit  $O(\text{MM}(n) \log^2 n)$ .



## 4. Randomisierte Berechnung kürzester Wege

**Definition 4:** Es sei  $G(V, E)$  ein Graph mit  $n$  Knoten. Eine **Nachfolgermatrix**  $S$  für  $G$  ist eine  $n \times n$ -Matrix, so dass für alle  $i \neq j$   $S_{ij}$  ein Nachfolger von  $i$  ist, der auf einem kürzesten Weg von  $i$  nach  $j$  liegt. Weiter ist  $S_{ii} = 0$ .

Sei nun  $A$  die Adjazenzmatrix eines Graphen  $G(V, E)$ ,  $D$  die Distanzmatrix von  $G$ . Seien  $i, j \in \{1, \dots, n\}$ ,  $i \neq j$  beliebig mit  $D_{ij} = d$ .

$S_{ij} = k$  ist möglich gdw.  $D_{ik} = 1$  und  $D_{kj} = d - 1$ .

Sei  $B^d \in \{0, 1\}^{n \times n}$  mit  $B_{kj}^d = 1$  gdw.  $D_{kj} = d - 1$  ( $B^d$  kann aus  $D$  in Zeit  $O(n^2)$  berechnet werden).

**Lemma 8:** Es sei  $A$  Adjazenzliste eines Graphen  $G(V, E)$  und  $B^d \in \{0, 1\}^{n \times n}$  wie oben definiert. Der Aufruf  $\text{BPWM}(A, B^d)$  liefert für alle  $i, j$  mit  $D_{ij} = d$  Einträge  $S_{ij}$  einer Nachfolgermatrix  $S$  für  $G$ .



Der Aufruf  $\text{BPWM}(A, B^d)$  liefert für alle  $i, j$  mit  $D_{ij} = d$  Einträge  $S_{ij}$ .

**Problem:** Es gibt  $n$  mögliche Werte für  $d$ .

Damit wäre die Laufzeit eines Algorithmus mindestens  $\Omega(n^{\text{MM}}(n))$  und somit superkubisch.

**Aber:** 3 Berechnungen von Zeugenmatrizen reichen aus, denn:

- Für alle  $i, j$  und alle Nachfolger  $k$  von  $i$  gilt  $D_{ij} - 1 \leq D_{kj} \leq D_{ij} + 1$ .
- Jeder Nachfolger  $k$  von  $i$  mit  $D_{kj} = D_{ij} - 1$  ist ein möglicher Eintrag für  $S_{ij}$ .
- Also: Jedes  $k$  mit  $A_{ik} = 1$  und  $D_{kj} \equiv D_{ij} - 1 \pmod{3}$  ist ein möglicher Eintrag für  $S_{ij}$ .
- Für  $s = 0, 1, 2$  definiere  $D^{(s)}$  durch  $D_{kj}^{(s)} = 1$  gdw.  $D_{kj} + 1 \equiv s \pmod{3}$ .
- Eine Nachfolgermatrix  $S$  von  $A$  kann dann aus den Zeugenmatrizen für  $A * D^{(s)}$ ,  $s = 0, 1, 2$  berechnet werden.



```

APSP(Adjazenzmatrix  $A$  für  $G(V, E)$ ) {
  /* berechnet eine Nachfolgermatrix  $S$  für  $G$  */
  berechne  $D = \text{APD}(A)$ ;                                     /*  $D$  ist Distanzmatrix */
  for  $s = 0, 1, 2$  do
    berechne  $D^{(s)} \in \{0, 1\}^{n \times n}$  so dass  $D_{kj}^{(s)} = 1$  gdw  $D_{kj} + 1 \equiv s \pmod{3}$ ;
    berechne eine Zeugenmatrix  $W^{(s)} = \text{BPWM}(A, D^{(s)})$ ;
    berechne Nachfolgematrix  $S$  mit  $S_{ij} = W_{ij}^{(D_{ij} \bmod 3)}$ ;
} /* APSP */

```

**Satz 3:** Der Algorithmus APSP berechnet eine Nachfolgermatrix eines  $n$ -Knoten Graphen  $G$  in erwarteter Zeit  $O(\text{MM}(n) \log^2 n)$ .

