

Las Vegas Algorithmus für kürzeste Wege

Patrick Kaell
Matr. 202660
Geb. 04.10.1975

Sommersemester 1997

1 Einleitung

Der Las Vegas Algorithmus für kürzeste Wege versucht in einem beliebigen, ungerichteten, zusammenhängenden Graphen den kürzesten Weg zwischen alle Knotenpaare zu finden. Diese Problemstellung wird APSP¹ genannt und ist mit klassischen Algorithmen von Dijkstra, Floyd-Warshall und Johnson in Zeit $O(n^3)$ lösbar². Ziel des Las Vegas Algorithmus ist es solche Problemstellungen mit Hilfe von *Randomization* schneller zu lösen als mit klassischen Algorithmen und zwar in Zeit $O(MM(n) \log^2 n)$. Dabei liefert er im Gegensatz zum Monte Carlo Algorithmus immer die richtige Lösung.

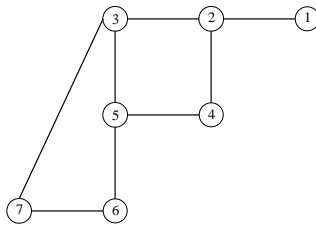
Das APD³ Problem ist eine eingeschränkte Version des APSP Problems und versucht lediglich eine Tabelle mit den minimalen Entfernungen zwischen allen Knotenpaaren des Graphen aufzustellen.

2 Lösungsvorschläge

Definition Sei $G(V, E)$ ein ungerichteter, zusammenhängender Graph mit $V = \{1, \dots, n\}$ und $|E| = m$. Die Adjazenzmatrix A ist eine $n \times n$ 0-1 Matrix mit $A_{ij} = A_{ji} = 1$ genau dann wenn die Kante (i, j) in E liegt.

Sei A gegeben, wir definieren die Entfernungsmatrix⁴ D als eine $n \times n$ Matrix mit ganzen, nicht-negativen Zahlen wobei D_{ij} die Länge des kürzesten Weges zwischen den Knoten i und j ist.

Beispiel Die Adjazenzmatrix A und die Entfernungsmatrix D des Beispielgraphen $G(V, E)$ (siehe Abbildung):



$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}, D = \begin{pmatrix} 0 & 1 & 2 & 2 & 3 & 4 & 3 \\ 1 & 0 & 1 & 1 & 2 & 3 & 2 \\ 2 & 1 & 0 & 2 & 1 & 2 & 1 \\ 2 & 1 & 2 & 0 & 1 & 2 & 3 \\ 3 & 2 & 1 & 1 & 0 & 1 & 2 \\ 4 & 3 & 2 & 2 & 1 & 0 & 1 \\ 3 & 2 & 1 & 3 & 2 & 1 & 0 \end{pmatrix}$$

¹all-pairs shortest paths

²wobei die Algorithmen von Dijkstra und Johnson sogar in Zeit $O(nm + n^2 \log n)$ gelöst werden können

³all-pairs distances

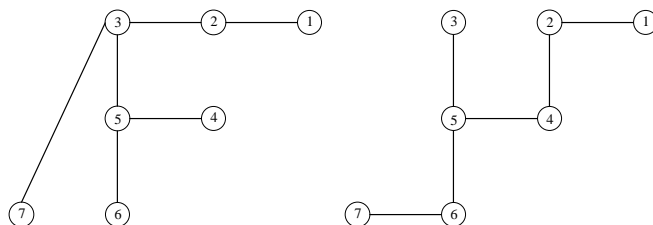
⁴auf Englisch: distance matrix

Bemerkung Die diagonalen Einträge sind in beiden Matrizen A und D immer Null. Da G zusammenhängend ist sind die Einträge immer endlich. Nicht zusammenhängende Graphen muß man in zusammenhängende Graphen zerlegen⁵.

2.1 Klassische Lösung (Breitensuchbaum)

Das APSP Problem kann folgendermaßen in Zeit $O(nm)$ gelöst werden: man errechnet für jeden Knoten $i \in V$ einen Breitensuchbaum⁶ T_i , dessen Wurzel der Knoten i ist. Ein solcher Baum kann in Zeit $O(m)$ errechnet werden. Der (einzige) Weg zwischen dem Knoten i und irgendeinem anderen Knoten j ist der kürzester Weg zwischen den beiden Knoten. Wenn alle Breitensuchbäume gegeben sind, kann die Entfernungsmatrix in Zeit $O(n^2)$ errechnet werden.

Beispiel Mögliche Breitensuchbäume T_5 für den Beispielgraphen G :



Bemerkung Wie in der Einleitung erwähnt, lösen die klassischen Algorithmen von Dijkstra, Floyd-Warshall und Johnson das APSP Problem in Zeit $O(n^3)$. Obwohl es klar ist, daß das APSP und APD Problem zu Lösung im *worst case* $\Omega(n^2)$ Zeit braucht, gibt es keinen Grund zur Annahme daß die $O(nm)$ ⁷ Zeitschranke auch nur nahe an der bestmöglichen ist.

2.2 Matrixmultiplikation

Definition Seien zwei boolesche $n \times n$ Matrizen A und B . Das Produkt C dieser beiden Matrizen

$$C_{ij} = \sum_{k=1}^n A_{ik} B_{kj}$$

errechnet sich im Prinzip genauso wie das Produkt zweier \mathbb{Z} -Matrizen⁸, wobei anstelle einer Multiplikation eine AND-Operation und anstelle einer Addition eine OR-Operation ausgeführt wird.

Anwendung Sei $A = B$ die Adjazenzmatrix des Graphen G . Sei das Produkt $C = A^2$. Der Eintrag (i, j) der Matrix C ist 1, genau dann wenn es einen Weg der Länge 2 zwischen den Knoten i und j gibt. Die Matrix A^l entspricht den Wegen der Länge l .

⁵was man in linearer Zeit machen kann

⁶auf Englisch: breadth-first search tree

⁷was so groß wie $\Theta(n^3)$ werden kann

⁸ \mathbb{Z} ist der Ring der ganzen Zahlen

Beispiel

$$A^2 = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}$$

Bemerkung Ein anderes Konzept ist die unendliche Summe

$$A^* = \sum_{l=0}^{\infty} A^l$$

wobei der Eintrag (i, j) der Matrix A^* genau dann 1 ist, wenn es irgendeinen Weg zwischen den Knoten i und j gibt. Die Matrix A^0 ist die Einheitsmatrix.

Indem wir alle Potenzen der Matrix A von 1 bis n ausrechnen, können wir das APD Problem lösen. Allerdings braucht dieses Verfahren $O(n^4)$ Zeit, wenn wir das klassische Matrixmultiplikationsverfahren benutzen⁹.

Definition Sei $MM(n)$ die Zeit, die man braucht um zwei $n \times n$ \mathbb{Z} -Matrizen miteinander zu multiplizieren.

Man kann die Boolesche Matrixmultiplikation in die \mathbb{Z} -Matrixmultiplikation einbetten, indem man die booleschen Einträge als Ganzzahlenwerte 0 und 1 betrachtet. Dies entspricht der Einbettung des geschlossenen Halbrings der booleschen Algebra in den Ring \mathbb{Z} . Alle bekannten \mathbb{Z} -Matrixmultiplikationsalgorithmen kann man auf jeden beliebigen Ring ausdehnen.

Lemma Die Boolesche Matrixmultiplikation zweier $n \times n$ Matrizen kann mit Hilfe der \mathbb{Z} -Matrixmultiplikation in Zeit $O(MM(n))$ ausgeführt werden.

Beweis Folgende Wahrheitstabellen zeigen, daß man eine AND-Operation durch eine Multiplikation und eine OR-Operation durch eine Addition ersetzen kann, wenn man den booleschen Wahrheitswert *false* durch die Zahl 0 und den booleschen Wahrheitswert *true* durch irgendeine andere Zahl größer als 0 ersetzt.

x	y	$x \text{ and } y$	x	y	$x \cdot y$
false	false	false	0	0	0
false	true	false	0	> 0	0
true	false	false	> 0	0	0
true	true	true	> 0	> 0	> 0

x	y	$x \text{ or } y$	x	y	$x + y$
false	false	false	0	0	0
false	true	true	0	> 0	> 0
true	false	true	> 0	0	> 0
true	true	true	> 0	> 0	> 0

⁹die klassische Matrixmultiplikation braucht $O(n^3)$ Zeit

Also kann man die Multiplikation zwischen zwei booleschen Matrizen mit Hilfe von Algorithmen ausrechnen, die für \mathbb{Z} -Matrizen gedacht sind (und zwar in Zeit $O(MM(n))$).

Bemerkung Die größte Zahl die während der Berechnung auftritt ist n . Grund: Die simulierte AND-Operation spuckt keinen Wert größer als 1 aus und die simulierte OR-Operation (was eigentlich eine Addition ist) wird n Mal hintereinander ausgeführt und bekommt als Parameter das Resultat der simulierten AND-Operation.

Momentan läuft der beste \mathbb{Z} -Matrixmultiplikationsalgorithmus in Zeit $O(n^{2.376})$ (was man für boolesche Matrizen verallgemeinern¹⁰ kann). Leider ergibt das eine überkubische Laufzeit für das APD Problem.

2.3 Lösen des APD Problems in Zeit $O(MM(n))$

Die Idee besteht darin daß man das Lösen des APD Problems auf das Lösen einer einfachen Matrixmultiplikation reduziert. Die Matrixmultiplikation wird auf einem abgeschlossenen, mit ∞ angereicherten \mathbb{R} -Ring¹¹ ausgeführt. Dabei wird die Addition durch die MIN-Operation und die Multiplikation durch die Addition ersetzt.

Definition Sei A eine Matrix. Der Eintrag (i, j) der Matrix A entspricht dem Gewicht¹² der Kante (i, j) , wenn die Kante existiert. Anderenfalls ist der Eintrag ∞ . Das Halbprodukt der beiden Matrizen A und B wird mit

$$C_{ij} = \min_{1 \leq k \leq n} (A_{ik} + B_{kj})$$

errechnet. Das unendliche Produkt A^* ist die Lösung des APD Problems.

Beweis

Induktionsannahme Jeder Eintrag (i, j) der Matrix A^l gibt die Länge des kürzesten Weges zwischen den Knotenpaaren i und j an, wenn der Weg kleiner oder gleich l ist. Anderenfalls ist der Eintrag ∞ .

Induktionsanfang Für $l = 1$ stimmt die Induktionsannahme: Jeder Eintrag (i, j) der Matrix A^1 gibt die Länge des kürzesten Weges zwischen den Knotenpaaren i und j an, wenn die Länge des Weges kleiner oder gleich 1 ist, da jeder Eintrag (i, j) gemäß Definition nur dann eins¹³ ist, wenn die Kante (i, j) existiert. Anderenfalls ist der Eintrag ∞ .

Induktionsschluß Wir haben jetzt gezeigt daß die Induktionsannahme für $l = 1$ stimmt. Wir müssen jetzt nur noch zeigen daß wenn die Induktionsannahme für l stimmt dann auch automatisch für $l + 1$.

¹⁰ siehe Lemma

¹¹ \mathbb{R} ist der Ring der reellen Zahlen

¹² bei ungewichteten Graphen ist das Gewicht einer Kante immer 1

¹³ für ungerichtete Graphen

Nehmen wir also an, daß die Induktionsannahme für l stimmt. Wenn wir die Matrix A^l mit der Matrix A multiplizieren versuchen wir zwischen jedem Knotenpaar (i, j) einen Knoten k^{14} dazwischenzufügen. Laut Formel addieren wir für jeden Knoten k den Eintrag (i, k) der Matrix A^l mit dem Eintrag (k, j) der Matrix A . Laut Formel nehmen wir das kleinste Ergebnis und haben so die Länge des kürzesten Weges, wenn der Weg kleiner oder gleich $l + 1$ ist.

Bemerkung Der Algorithmus hat folgende Nachteile:

- Man kann den Algorithmus nicht vom APD zum APSP Problem verallgemeinern.
- Die Reduktion zur \mathbb{Z} -Matrixmultiplikation erzeugt Zahleneinträge die superlinear mit n wachsen. Da man in der Praxis mit realen¹⁵ Rechnern arbeitet bedeutet das, daß die Laufzeit der arithmetischen Operationen superlinear mit n steigt. Man kann nicht mehr von einer festen Laufzeit für eine bestimmte arithmetischen Operation ausgehen.

2.4 Schlußfolgerung

Im folgenden Kapitel werde ich zeigen wie man das APD Problem auf eine \mathbb{Z} -Matrixmultiplikation reduzieren kann, wobei die Zahleneinträge der Matrizen nur *logarithmisch* mit der Matrizengröße wachsen. Zuletzt werde ich zeigen wie man das Verfahren mit Hilfe von *Randomization* auf das APSP verallgemeinern kann (wobei ein black-box Matrixmultiplikationsalgorithmus benutzt wird).

3 Berechnung der Entfernung zwischen allen Knotenpaaren (APD)

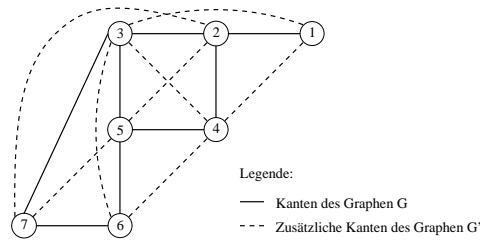
Definition Sei $G'(V, E')$ die Ableitung des Graphen $G(V, E)$. Die Ableitung G' bekommt man indem man eine Kante zwischen jedes Knotenpaar i und j mit $i \neq j \in V$ einfügt, dessen Knoten in G entweder 1 oder 2 voneinander entfernt sind. G' kann auch als G *quadrat* angesehen werden. Sei A' die Adjazenzmatrix und D' die Entfernungsmatrix des Graphen G' .

Bemerkung G ist ein Untergraph ($G \subseteq G'$) von G' .

Beispiel Die Adjazenzmatrix A' und die Entfernungsmatrix D' des Beispielgraphen $G'(V', E)$ (siehe Abbildung):

¹⁴ k ist ein Nachbarknoten von j und kann auch i selbst sein

¹⁵mit realen Rechner meine ich Rechner die aus Silizium bestehen (oder irgendeine andere Implementierung), jedenfalls darf es kein mathematisches Modell sein das von einer festen Laufzeit einer arithmetischen Operation ausgeht



$$A' = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 \end{pmatrix}, D' = \begin{pmatrix} 0 & 1 & 1 & 1 & 2 & 2 & 2 \\ 1 & 0 & 1 & 1 & 1 & 2 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 2 \\ 2 & 1 & 1 & 1 & 0 & 1 & 1 \\ 2 & 2 & 1 & 1 & 1 & 0 & 1 \\ 2 & 1 & 1 & 2 & 1 & 1 & 0 \end{pmatrix}$$

Lemma 3.1 Sei $Z = A^2$, wobei A die Adjazenzmatrix des Graphen G ist. Dann gibt es einen Weg der Länge 2 zwischen den Knoten i und j in G genau dann, wenn $Z_{ij} > 0$. Der Eintrag Z_{ij} gibt sogar die Anzahl der verschiedenen Wege der Länge 2 zwischen den Knoten i und j an.

Beweis

$$Z = A^2 = A \cdot A$$

$$Z_{ij} = \sum_{k=1}^n A_{ik} A_{kj} > 0$$

$$\Leftrightarrow \sum_{k=1}^n A_{ik} A_{kj} \neq 0$$

$$\Leftrightarrow \exists k / A_{ik} A_{kj} \neq 0$$

$$\Leftrightarrow \exists k / A_{ik} \neq 0 \text{ und } A_{kj} \neq 0$$

$$\Leftrightarrow \exists k / A_{ik} = 1 \text{ und } A_{kj} = 1$$

$$\Leftrightarrow \exists k / (i, k) \in G \text{ und } (k, j) \in G$$

$$\Leftrightarrow \text{Es gibt einen Weg der Länge 2 zwischen den Knoten } i \text{ und } j$$

Wir arbeiten nur mit positiven Zahlen ($A_{xy} \in \mathbb{N}$)

Die Matrixeinträge sind entweder 0 oder 1

Ein Weg der Länge 2 zwischen den Knoten i und j geht über einen einzigen anderen Knoten $k \in G$. Es gibt höchstens n verschiedene Wege, da es höchstens n verschiedene Knoten k in G gibt. Wenn es einen Weg der Länge 2 zwischen den Knoten i und j gibt der über den Knoten k geht, dann gibt die Formel $A_{ik} A_{kj}$ den Wert 1 anderenfalls den Wert 0 zurück. Damit ist es klar, daß die Formel $\sum_{k=1}^n A_{ik} A_{kj}$ die Anzahl der verschiedenen Wege der Länge 2 zwischen den Knoten i und j angibt.

Die Matrix $Z = A^2$ kann in Zeit $O(MM(n))$ errechnet werden. Wenn die Matrizen A und Z gegeben sind, ist es einfach die Matrix A' in Zeit $O(n^2)$ zu errechnen. $A_{ij} = 1$ genau dann, wenn $i \neq j$ und wenigstens A_{ij} oder Z_{ij}

verschieden von Null ist. Man muß nur aufpassen daß die Diagonaleinträge von A' auf Null gesetzt sind. Anders ausgedrückt:

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

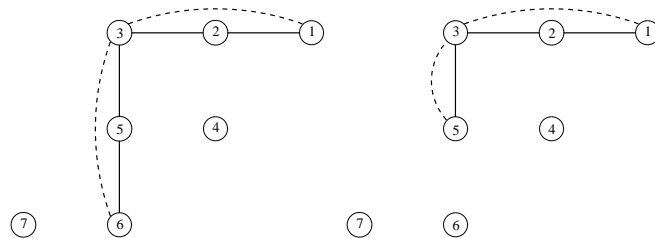
Bemerkung G' ist genau dann komplett wenn G einen Durchmesser¹⁶ von mindestens 2 hat. Falls der Durchmesser von G genau 2 ist¹⁷, kann die APD-Matrix einfach in Zeit $O(n^2)$ errechnet werden:

$$D = 2A' - A$$

Lemma 3.2 Sei irgendein Knotenpaar $i, j \in V$:

$$D_{ij} = \begin{cases} 2D'_{ij} & \text{falls } D_{ij} \text{ gerade ist} \\ 2D'_{ij} - 1 & \text{falls } D_{ij} \text{ ungerade ist} \end{cases}$$

Beispiel Abbildung des kürzesten Weges im Graphen G (durchgezogene Linien) auf den Graphen G' (gestrichelte Linien), wobei D_{ij} im ersten Fall gerade ist und im zweiten Fall ungerade.



Beweis

- Wenn D_{ij} gerade ist, kann man den kürzesten Weg vom Knoten i zum Knoten j in Abschnitte von jeweils zwei Kanten einteilen. Dieser Weg kann man auf den Graphen G' abbilden, wobei die zwei Kanten eines Abschnittes auf eine einzige Kante $\in E' \setminus E$ abgebildet werden. Dieser Weg ist der kürzeste Weg zwischen den beiden Knoten im Graphen G' . Um dies zu zeigen führen wir einen Widerspruchsbeweis: Wir nehmen an, daß es im Graphen G' einen noch kürzeren Weg gibt. Besteht dieser Weg aus auch nur einer einzigen Kante aus E kann er nicht mehr der kürzeste sein. Besteht dieser Weg nur aus Kanten aus $E' \setminus E$ könnte man den Weg wieder auf den Graphen G zurückführen und hätte so für den Graphen G einen noch kürzeren Weg, was gegen die Voraussetzung spricht.

Die Entfernung zwischen zwei Knoten i und j im Graphen G' beträgt also D'_{ij} . Da eine Kante aus G' (in diesem Fall!!!) jeweils zwei Kanten aus G entspricht, haben wir: $D_{ij} = 2D'_{ij}$.

¹⁶Der Durchmesser einer Graphen ist der maximale kürzester Weg zwischen allen Knotenpaaren

¹⁷Normalerweise kann der Graph G natürlich irgendeinen Durchmesser haben

Lemma 3.4 Sei irgendein Knotenpaar $i, j \in V$:

- $\forall k \in G, k$ Nachbar von i/D_{ij} ist gerade $\Rightarrow D'_{kj} \geq D'_{ij}$
- $\forall k \in G, k$ Nachbar von i/D_{ij} ist ungerade $\Rightarrow D'_{kj} \leq D'_{ij}$
- $\exists k \in G, k$ Nachbar von i/D_{ij} ist ungerade $\Rightarrow D'_{kj} < D'_{ij}$

Beweis

- Wir betrachten zunächst den Fall wenn $D_{ij} = 2l$ gerade ist:

$D_{ij} = 2l$ ist gerade	Voraussetzung
$\forall k, k$ Nachbar von $i/D_{kj} \geq 2l - 1$	Nach Lemma 3.3
$D'_{ij} = l$	Nach Lemma 3.2
$D'_{kj} \geq \frac{D_{kj}}{2} \geq l - \frac{1}{2}$	Nach Lemma 3.2
$D'_{kj} \geq l = D'_{ij}$	Da die Entfernungen ganze Zahlen sind ($\in \mathbb{N}$)

- Wir betrachten jetzt den Fall wenn $D_{ij} = 2l - 1$ ungerade ist:

$D_{ij} = 2l - 1$ ist ungerade	Voraussetzung
$\forall k, k$ Nachbar von $i/D_{kj} \leq 2l$	Nach Lemma 3.3
$D'_{ij} = l$	Nach Lemma 3.2
$D'_{kj} \leq \frac{D_{kj}+1}{2} \leq l + \frac{1}{2}$	Nach Lemma 3.2
$D'_{kj} \leq l = D'_{ij}$	Da die Entfernungen ganze Zahlen sind ($\in \mathbb{N}$)

Weiter:

$\exists k, k$ Nachbar von $i/D_{kj} = D_{ij} - 1 = 2l - 2$	Nach Lemma 3.3
$D'_{kj} = l - 1 < l = D'_{ij}$	Nach Lemma 3.2

Definition Sei $\Gamma(i)$ die Menge der Nachbarn den Knotens i in G . Sei $d(i)$ der Grad²⁰ des Knotens i .

Bemerkung $\forall i \in G/Z_{ii} = d(i)$.

Lemma 3.5

- D_{ij} ist gerade $\Leftrightarrow \sum_{k \in \Gamma(i)} D'_{kj} \geq D'_{ij} d(i)$
- D_{ij} ist ungerade $\Leftrightarrow \sum_{k \in \Gamma(i)} D'_{kj} < D'_{ij} d(i)$

²⁰Der Grad eines Knotens ist die Anzahl der Nachbarn eines Knotens

Beweis Um zum Ergebnis im Lemma 3.5 zu kommen, braucht man eigentlich nur die Ungleichungen aus dem Lemma 3.4 über alle Nachbarknoten des Knotens i zusammenzuaddieren.

Bemerkung Mit Hilfe des Lemmas 3.5 haben wir ein effizientes Verfahren die Parität der Längen der kürzesten Wege zu errechnen.

APD Algorithmus

Input: Graph $G(V, E)$ in form of an adjacency matrix A

Output: The APD matrix D for G

1. $Z \leftarrow A^2$
2. **compute** matrix A' such that $A'_{ij} = 1 \Leftrightarrow i \neq j$ and $(A_{ij} = 1 \text{ or } Z_{ij} > 0)$
3. **if** $\forall i \neq j / A'_{ij} = 1$ **then return** $D = 2A' - A$
4. Recursively compute the APD matrix D' for the graph G' with adjacency matrix A' .
5. $S \leftarrow AD'$
6. **return** matrix D with $D_{ij} = \begin{cases} 2D'_{ij} & \text{if } S_{ij} \geq D'_{ij}Z_{ii} \\ 2D'_{ij} - 1 & \text{if } S_{ij} < D'_{ij}Z_{ii} \end{cases}$

Bemerkung Im Schritt 5 benutzen wir die Matrixmultiplikation um

$$\sum_{k \in \Gamma(i)} D'_{kj} = \sum_{k=1}^n A_{ik} D'_{kj} = S_{ij}$$

auszurechnen. Die Länge der Ganzzahlen der Matrizeneinträge überschreitet in diesem Algorithmus $O(\log n)$ nicht.

Satz 3.6 Der APD Algorithmus rechnet die Entfernungsmatrix für einen n -kantigen Graphen G in Zeit $O(\text{MM}(n) \log n)$ aus, indem er die \mathbb{Z} -Matrixmultiplikation benutzt, dessen Einträge von $O(n^2)$ begrenzt sind.

Beweis Nehmen wir an, daß der Graph G den Durchmesser δ hat. Dann hat der Graph G' den Durchmesser $\lceil \frac{\delta}{2} \rceil$. Sei $T(n, \delta)$ die Laufzeit des APD Algorithmus für Eingabegraphen mit n Knoten und einem Durchmesser von δ .

- Wenn $\delta = 1$, dann ist G ein kompletter Graph.
- Wenn $\delta = 2$, dann haben wir: $T(n, \delta) = \text{MM}(n) + O(n^2)$.
- Wenn $\delta > 2$, dann haben wir: $T(n, \delta) = 2\text{MM}(n) + T(n, \lceil \frac{\delta}{2} \rceil) + O(n^2)$.

Nachprüfung Sei $\delta = 2$. Dann hat Schritt 1 des APD Algorithmus eine Laufzeit von $MM(n)$, da es sich um eine Matrixmultiplikation handelt. Schritt 2 und Schritt 3 haben eine Laufzeit von $O(n^2)$.

Sei $\delta > 2$. Dann hat Schritt 1 des APD Algorithmus eine Laufzeit von $MM(n)$, da es sich um eine Matrixmultiplikation handelt. Schritt 2 hat eine Laufzeit von $O(n^2)$. Schritt 3 wird nicht ausgeführt. Schritt 4 rechnet die APD Matrix des Graphen G' (mit n Knoten und einem Durchmesser von $\lceil \frac{\delta}{2} \rceil$) aus. Laufzeit: $T(n, \lceil \frac{\delta}{2} \rceil)$. Schritt 5 hat eine Laufzeit von $MM(n)$, da es sich um eine Matrixmultiplikation handelt. Schritt 6 hat eine Laufzeit von $O(n^2)$.

Aus den Voraussetzungen, daß $\delta < n$ und $MM(n) = \Omega(n^2)$ und die Rekursionstiefe $O(\log n)$ ist, folgt die Laufzeit des APD Algorithmus: $O(MM(n) \log n)$. Da die Einträge in der Entfernungsmatrix von n begrenzt sind, folgt daß die Einträge der S Matrix von n^2 begrenzt sind.

4 Die Boolesche Witness-Matrixmultiplikation

Wir versuchen jetzt den APD Algorithmus auf das APSP Problem zu erweitern, indem wir einen Algorithmus entwerfen der *Zeugnisse*²¹ einer booleschen Matrixmultiplikation sucht.

Definition Seien zwei boolesche $n \times n$ Matrizen A und B . Sei $P = AB$ das Produkt unter der Booleschen Matrixmultiplikation. Ein Witness (Zeugnis) für den Eintrag P_{ij} ist ein Index $k \in \{1, \dots, n\}$, so daß $A_{ik} = B_{kj} = 1$. P_{ij} ist 1 genau dann für P_{ij} ein Witness (Zeugnis) k existiert. Die BPWM²² Matrix des Produktes P ist eine \mathbb{Z} -Matrix W so daß jeder Eintrag W_{ij} ein Witness k für P_{ij} enthält. Wenn kein Witness für P_{ij} existiert haben wir: $W_{ij} = 0$.

Seien die Matrizen A und B gegeben²³. Das BPWM Problem besteht jetzt darin, die Witness Matrix W zu errechnen. Mit Hilfe von *Randomization* kann man das Verfahren erheblich beschleunigen.

Bemerkung Es können bis zu n Witnesses für jeden Eintrag P_{ij} geben. Die Matrixmultiplikation der beiden Matrizen A und B gibt die Matrix C als Ergebnis zurück, dessen Eintrag C_{ij} genau die Anzahl der Witnesses für den Eintrag P_{ij} angibt.

Sei $A = B$ die Adjazenzmatrix des Graphen G ist, dann ist $P_{ij} = 1$ genau dann, wenn es einen Weg der Länge 2 zwischen den Knoten i und j gibt. C_{ij} gibt die Anzahl solcher Wege an. Ein Witness k für P_{ij} ist der Zwischenknoten auf dem Weg der Länge 2 zwischen den Knoten i und j .

Problem Das Lösen des BPWM Problems braucht $O(n^3)$ Zeit, wenn man die *brute-force* Methode benutzt: Das Ausprobieren aller $k \in \{1, \dots, n\}$ als potentieller Witness für P_{ij} braucht $\Omega(n)$ Zeit.

Lösung Es gibt eine einfache Lösung des BPWM Problems für den speziellen Fall wenn es für jeden Eintrag P_{ij} genau ein einziger Witness gibt. Diese Lösung wird jetzt im Lemma 4.1 vorgestellt.

²¹auf Englisch: witness

²²boolean produkt witness matrix

²³wenn nötig auch die Matrix P

Lemma 4.1 Sei \hat{A} die Matrix dessen Einträge man folgendermaßen berechnet: $\hat{A}_{ik} = kA_{ik}$. Die \mathbb{Z} -Matrixmultiplikation der beiden Matrizen \hat{A} und B gibt als Ergebnis eine Matrix zurück, dessen Einträge die Witness für alle Einträge der Matrix P angeben, für die es genau ein einziger Witness gibt. Wenn es für jeden Eintrag P_{ij} genau ein einziger Witness gibt, können wir mit diesem Verfahren die BPWM Matrix für P errechen.

Beweis

$$\begin{aligned}
C &= \hat{A}B \\
C_{ij} &= \sum_{m=1}^n \hat{A}_{im}B_{mj} \\
&= \sum_{m=1}^n mA_{im}B_{mj} \\
\exists! k \in \{1, \dots, n\} / A_{ik}A_{kj} \neq 0 &\Rightarrow C_{ij} = \underbrace{\sum_{m=1}^{k-1} mA_{im}B_{mj}}_0 + \underbrace{kA_{ik}B_{kj}}_k + \underbrace{\sum_{m=k+1}^n mA_{im}B_{mj}}_0 \\
&= k
\end{aligned}$$

Man kann natürlich nicht *a priori* wissen, ob es für irgendeinen Eintrag in P genau ein einziger Witness gibt. Wir können aber mit Hilfe von *Randomization* eine Menge von Indexe herauswählen, wobei die Wahrscheinlichkeit groß ist, daß diese Menge von Indexe genau ein einziger Witness für einen Eintrag von P enthält.

Betrachten wir zunächst mal nur einen Eintrag P_{ij} . Sei w die Anzahl der Witnesses für diesen Eintrag. Wir könnten w bestimmen indem wir die \mathbb{Z} -Matrixmultiplikation benutzen: $C_{ij} = \sum_{k=1}^n A_{ik}B_{kj}$. Wir nehmen an, daß $w \geq 2$, da es sonst einfach wäre die Witness zu finden.

Sei $r \in \mathbb{Z}$, so daß $\frac{n}{2} \leq wr \leq n$. Behauptung: Eine per Zufall ausgewählte Menge von Indexe $R \subseteq \{1, \dots, n\}$, mit $|R| = r$ enthält mit sehr hoher Wahrscheinlichkeit genau ein einziger Witness für P_{ij} .

Um diese Behauptung nachzuprüfen, müssen wir uns eine Urne mit n Kugeln vorstellen, jede Kugel entspricht einem der n Indexe. Die weißen Kugeln entsprechen den Witnesses, die anderen sind schwarz.

Das folgende Lemma zeigt, daß die Wahrscheinlichkeit, daß R genau ein einziger Witness enthält ziemlich groß ist.

Lemma 4.2 Sei eine Urne die mit n Kugeln gefüllt ist. Davon sind w Kugeln weiß, und $n-w$ Kugeln schwarz. Jetzt werden r Kugeln per Zufall herausgewählt, wobei wir $\frac{n}{2} \leq wr \leq n$ haben.

Wahrscheinlichkeit, daß genau eine weiße Kugel herausgewählt wird $\geq \frac{1}{2e}$

Beweis

$$\begin{aligned}
\frac{\binom{w}{1} \binom{n-w}{r-1}}{\binom{n}{r}} &= w \frac{r!}{(r-1)!} \frac{(n-w)!}{n!} \frac{(n-r)!}{(n-w-r+1)!} \\
&= wr \left(\prod_{i=0}^{w-1} \frac{1}{n-1} \right) \left(\prod_{j=0}^{w-2} (n-r-j) \right) \\
&= \frac{wr}{n} \left(\prod_{j=0}^{w-2} \frac{n-r-j}{n-1-j} \right) \\
&\geq \frac{wr}{n} \left(\prod_{j=0}^{w-2} \frac{n-r-j-(w-j-1)}{n-1-j-(w-j-1)} \right) \\
&= \frac{wr}{n} \left(\prod_{j=0}^{w-2} \frac{n-w-(r-1)}{n-w} \right) \\
&= \frac{wr}{n} \left(1 - \frac{r-1}{n-w} \right)^{w-1} \\
&\geq \frac{1}{2} \left(1 - \frac{1}{w} \right)^{w-1}
\end{aligned}$$

Bemerkung Die letzte Ungleichung folgt aus den Ungleichungen $\frac{wr}{n} \geq \frac{1}{2}$ und $\frac{r-1}{n-w} \leq \frac{1}{w}$, die aus der Voraussetzung $\frac{n}{2} \leq wr \leq n$ folgen.

Anwendung Wir nehmen jetzt an, daß die Menge R genau ein einziger Witness für P_{ij} enthält. Wir verändern jetzt das in Lemma 4.1 vorgestellte Verfahren: Die Menge R wird als Indexvektor (R_1, R_2, \dots, R_n) dargestellt, wobei $R_k = 1 \Leftrightarrow k \in R$ und $R_k = 0 \Leftrightarrow k \notin R$. Seien A^R und B^R die Matrizen, die man folgendermaßen berechnet: $A_{ik}^R = kR_k A_{ik}$ und $B_{kj}^R = R_k B_{kj}$.

Lemma 4.3 Nehmen wir an, daß die Menge R genau ein einziger Witness für den Eintrag P_{ij} enthält. Dann gibt die \mathbb{Z} -Matrixmultiplikation der beiden Matrizen A^R und B^R eine Matrix C^R zurück, dessen Eintrag C_{ij}^R den Index des einzigen Witnesses für den Eintrag P_{ij} enthält.

Beweis

$$\begin{aligned}
C^R &= A^R B^R \\
C_{ij}^R &= \sum_{m=1}^n A_{im}^R B_{mj}^R \\
&= \sum_{m=1}^n m R_m A_{im} R_m B_{mj} \\
&= \sum_{m=1}^n m R_m A_{im} B_{mj} \\
\exists! k \in \{1, \dots, n\} / A_{ik} B_{kj} \neq 0 \text{ und } R_k \neq 0 &\Rightarrow C_{ij} = \underbrace{\sum_{m=1}^{k-1} m R_m A_{im} B_{mj}}_0 + \underbrace{k R_k A_{ik} B_{kj}}_k \\
&\quad + \underbrace{\sum_{m=k+1}^n m R_m A_{im} B_{mj}}_0 \\
&= k
\end{aligned}$$

Anwendung Das Produkt der beiden Matrizen A^R und B^R liefert Witnesses für alle Einträge aus P , die genau einen einzigen Witness in R haben. Nach Lemma 4.2 gibt es eine konstante Wahrscheinlichkeit, daß eine per Zufall ausgewählte Menge R der Größe r genau eine einzige Witness für einen Eintrag²⁴ aus P enthält, wobei $\frac{n}{2r} \leq w \leq \frac{n}{r}$. Es ist sehr unwahrscheinlich, daß man nicht für alle Einträge P_{ij} , die Witnesses besitzen einen Witness findet, wenn man das Verfahren für $O(\log n)$ unabhängig ausgewählte Mengen R wiederholt. Selbst dann, wenn nicht für alle in Frage kommenden Einträge aus P , einen Witness gefunden wurde, kann man die restlichen Witnesses mit Hilfe der *brute-force* Methode finden.

Um den verschiedenen möglichen Werten von w Rechnung zu tragen²⁵, müssen wir das Verfahren mit mehreren Werten von r anwenden. Es reicht aber, nur Zweierpotenzen zwischen 1 und n für r auszuprobieren.

BPWM Algorithmus

Input: Two $n \times n$ 0-1 matrices A and B

Output: Witness matrix W for the Boolean matrix $P = AB$

1. $W \leftarrow -AB$
2. **for** $t = 0, \dots, \lceil \log n \rceil$ **do**
 - (a) $r \leftarrow 2^t$
 - (b) **repeat** $\lceil 3.77 \log n \rceil$ **times**
 - i. **choose** random $R \subseteq \{1, \dots, n\}$ with $|R| = r$

²⁴für den es w Witnesses gibt

²⁵da die Gleichung $\frac{n}{2r} \leq w \leq \frac{n}{r}$ erfüllt werden muß

- ii. **compute** A^R and B^R
 - iii. $Z \leftarrow A^R B^R$
 - iv. **for all** (i, j) **do**
 - A. **if** $W_{ij} < 0$ and Z_{ij} is witness **then** $W_{ij} \leftarrow Z_{ij}$
3. **for all** (i, j) **do**
- (a) **if** $W_{ij} < 0$ **then** find witness W_{ij} by brute force

Bemerkung Die Matrix W wird mit der Negation der Anzahl der Witnesses initialisiert²⁶. Der Algorithmus sucht nur Witnesses für negative Einträge²⁷, da die Witnesses für die nicht negativen Einträge²⁸ schon gefunden wurden. Die negativen Einträge in W markieren also die Einträge in P , für die noch nach einem Witness gesucht werden muß. Der letzte Schritt im BPWM Algorithmus benutzt die *brute force* Methode, um die noch nicht gefundenen Witnesses zu suchen, und stellt sicher daß der BPWM Algorithmus Las Vegas ist.

Satz 4.4 Der BPWM Algorithmus ist ein Las Vegas Algorithmus für das BPWM Problem mit einer Laufzeit von $O(\text{MM}(n) \log^2 n)$.

Beweis Schritt 1 hat eine Laufzeit von $\text{MM}(n)$, da es sich um eine Matrixmultiplikation handelt.

Schritt 2 besteht hauptsächlich aus einer $n \times n$ \mathbb{Z} -Matrixmultiplikation, die (in zwei Schleifen verschachtelt) $O(\log^2 n)$ mal ausgeführt wird. Dies ergibt eine Laufzeit von $O(\text{MM}(n) \log^2 n)$.

Sei ein bestimmter $P_{ij} \neq 0$, mit w Witnesses. Die äußere Schleife des Schrittes 2 wird mindestens einmal mit einem Wert r (so daß $\frac{n}{2} \leq wr \leq n$) durchlaufen. Dabei ist die Wahrscheinlichkeit²⁹ daß eine per Zufall ausgewählte Menge R nicht genau ein einziger Witness für P_{ij} enthält höchstens $1 - \frac{1}{2e}$. Da die innere Schleife $3.77 \log n$ mal ausgeführt wird, ist die Wahrscheinlichkeit daß keine Witness für P_{ij} gefunden wird höchstens $(1 - \frac{1}{2e})^{3.77 \log n} \leq \frac{1}{n}$.

Daraus folgt, daß im Schritt 3 wahrscheinlich noch bis zu n Witnesses gefunden werden müssen. Mit Hilfe der *brute force* Methode kann man ein Witness in Zeit $O(n)$ finden. Daraus ergibt sich eine Laufzeit von $O(n^2)$ für den Schritt 3.

Laufzeit des BPWM Algorithmus: $\text{MM}(n) + O(\text{MM}(n) \log^2 n) + O(n^2) = O(\text{MM}(n) \log^2 n)$

5 Berechnung der kürzesten Wege zwischen allen Knotenpaaren (APSP)

Definition Sei G ein Graph mit n Knoten. Die Nachfolgermatrix³⁰ S des Graphen G ist eine $n \times n$ Matrix, wobei der Eintrag S_{ij} den Index des Nach-

²⁶ $W \leftarrow -P = -AB$

²⁷ wenn der Eintrag Null ist, gibt es keinen Witness für diesen Eintrag

²⁸ Wenn ein Witness gefunden wird, wird der negative Eintrag durch den (positiven) Witness-Index ersetzt

²⁹ nach Lemma 4.2

³⁰ auf Englisch: successor matrix

barknoten k des Knotens i enthält, der auf dem kürzesten Weg zwischen den Knoten i und j liegt.

Anwendung Sei die Nachfolgermatrix S gegeben. Mit Hilfe des folgenden Algorithmus kann man den kürzesten Weg zwischen den Knoten i und j ermitteln, wobei die Laufzeit proportional zur Länge des kürzesten Weges ist.

1. $k_1 = i, n = 1$
2. **while** $k_n \neq j$
 - (a) $k_{n+1} \leftarrow S_{k_n, j}$
 - (b) $n \leftarrow n + 1$

Der kürzester Weg ist nach Ablauf des Algorithmus mit der Knotenfolge (k_1, k_2, \dots, k_n) gegeben.

Bemerkung Seien die Adjazenzmatrix A und die Entfernungsmatrix D für den Graphen G gegeben. Sei (i, j) ein Knotenpaar wobei die Knoten i und j mit einer Entfernung von d zueinander stehen. Dann haben wir:

$$S_{ij} = k \Leftrightarrow D_{kj} = d - 1 \text{ und } D_{ik} = 1$$

Definition Sei B^d eine $n \times n$ 0-1 Matrix, so daß wir

$$B_{kj}^d = 1 \Leftrightarrow D_{kj} = d - 1$$

haben. Wenn D gegeben ist, kann B^d in Zeit $O(n^2)$ errechnet werden.

Anwendung Man kann alle Einträge der Nachfolgermatrix S bestimmen, dessen Knotenpaare mit einer Entfernung von d zueinander stehen, indem man die Witnessmatrix des Produktes der beiden Matrizen A und B^d errechnet³¹.

Beweis

$\begin{aligned} \text{BPWM}(A, B^d) &= k \\ \Rightarrow A_{ik} B_{kj}^d &= 1 \\ \Rightarrow D_{ik} = 1 \text{ und } D_{kj} &= d - 1 \\ \Rightarrow S_{ij} &= k \end{aligned}$	<p>Siehe Definition Siehe Bemerkung</p>
--	---

Problem Um alle Einträge der Nachfolgermatrix S zu erhalten muß man das Verfahren für alle n verschiedenen Werte von d wiederholen. Leider ergibt dies eine überkubische Laufzeit für das APSP Problem.

Lösung Ich werde jetzt zeigen, daß man die Nachfolgermatrix S schneller ausrechnen kann, indem man den BPWM Algorithmus statt n mal nur 3 mal ausführt.

Für jedes Knotenpaar (i, j) und Nachbar k von i gilt nach Lemma 3.3 die Ungleichung: $D_{ij} - 1 \leq D_{kj} \leq D_{ij} + 1$. Jeder Nachbar k der die Voraussetzung $D_{kj} = D_{ij} - 1$ erfüllt, könnte in S_{ij} eingetragen werden. Daraus folgt, daß jeder Knoten k der die Voraussetzungen $A_{ik} = 1$ und $D_{kj} \equiv D_{ij} - 1 \pmod{3}$ erfüllt in S_{ij} eingetragen werden könnte.

³¹Laufzeit: $O(MM(n) \log^2 n)$

Definition Sei $s \in \{0, 1, 2\}$ und $D^{(s)}$ eine $n \times n$ 0-1 Matrix so daß

$$D_{kj}^{(s)} = 1 \Leftrightarrow D_{kj} + 1 \equiv s \pmod{3}$$

Man kann alle Einträge der Nachfolgermatrix S bestimmen, indem man die Witnessmatrizen der Produkte zwischen der Matrix A und jedem der Matrizen $D^{(0)}, D^{(1)}, D^{(2)}$ errechnet.

APSP Algorithmus

Input: An $n \times n$ adjacency matrix A for a graph G

Output: The successor matrix S for G

1. **compute** the distance matrix $D = \text{APD}(A)$
2. **for** $s = \{0, 1, 2\}$ **do**
 - (a) **compute** 0-1 matrix $D^{(s)} = 1$ such that $D_{kj}^{(s)} = 1$ if and only if $D_{kj} + 1 \equiv s \pmod{3}$
 - (b) **compute** the witness matrix $W^{(s)} = \text{BPWM}(A, D^{(s)})$
3. **compute** successor matrix S such that $S_{ij} = W_{ij}^{(D_{ij} \pmod{3})}$

Satz 5.1 Der APSP Algorithmus rechnet für einen n -knotigen Graphen G die Nachfolgermatrix S in Zeit $O(\text{MM}(n) \log^2 n)$ aus.

Beweis Schritt 1 hat eine Laufzeit von $O(\text{MM}(n) \log n)$, da es sich um den APD Algorithmus handelt³².

Schritt 2 besteht hauptsächlich aus dem BPWM Algorithmus³³, der in einer Schleife 3 mal ausgeführt wird. Dies ergibt eine Laufzeit von $O(\text{MM}(n) \log^2 n)$ für den Schritt 2.

Schritt 3 hat eine Laufzeit von $O(n^2)$.

Laufzeit des BPWM Algorithmus: $O(\text{MM}(n) \log n) + O(\text{MM}(n) \log^2 n) + O(n^2) = O(\text{MM}(n) \log^2 n)$

³²Der APD Algorithmus hat eine Laufzeit von $O(\text{MM}(n) \log n)$ (siehe Satz 3.6)

³³Der BPWM Algorithmus hat eine Laufzeit von $O(\text{MM}(n) \log^2 n)$ (siehe Satz 4.4)