Woche 13 – Übersicht & Tricks

(Disclaimer: Die hier vorzufindenden Notizen haben keinerlei Anspruch auf Korrektheit oder Vollständigkeit und sind nicht Teil des offiziellen Vorlesungsmaterials. Alle Angaben sind ohne Gewähr.)

Anmerkungen zu Serie 11: Allgemeines

-

All-to-All Shortest Path Problem - Einführung:

In einem gewichteten Graphen $G = (V, E)$ möchten wir die Distanz zwischen 2 Knoten für alles Paare (u, v) mit $u, v \in V$ berechnen. Um dieses Problem zu lösen könnte zum Beispiel Dijkstras Algorithmus verwendet werden, der dann von jedem Knoten $s \in V$ aus gestartet wird. Dies benötigt insgesamt Zeit $O\big(|V| \log|V| * (|V| + |E|)\big)$ und funktioniert natürlich nur, wenn alle Kantengewichte nicht-negativ sind. Im Falle allgemeiner Kantengewichtsfunktionen kann der Algorithmus von Bellman-Ford verwendet werden, was zu einer Gesamtlaufzeit von $O(|V|^2|E|)$ führt. Nun schauen wir uns zwei Algorithmen an, die für einen Graphen mit allgemeinen Kantengewichten die kürzesten Pfade zwischen allen Paaren zweier Knoten schneller berechnet.

Floyd-Warshall-Algorithmus:

Der Floyd-Warshall-Algorithmus beruht auf einem uns bereits bekanntem Konzept, nämlich DP! Er funktioniert folgendermassen: Wir Speichern in einem DP-Table ab, was die kürzesten Pfade zwischen allen Paaren zweier Knoten ist, wenn auf dem Pfad zwischen 2 Knoten nur Knoten mit Knotennummern $\leq i$ vorkommen dürfen. Dabei startet i bei 0 und iteriert nach oben bis n, weil dann haben wir die kürzesten Pfade für alle Knotenpaare, da auf den Pfaden dann alle Knoten vorkommen dürfen.

Wir gehen wie folgt vor: Zuerst nummerieren wir alle Knoten von 1 bis n durch. Danach erstellen wir einen $n \times n \times (n+1)$-DP-Table. Wir speichern ab: $DP[u][v][i]$: Kürzeste Pfad von u nach v mit Zwischenknotennummern $\leq i$.

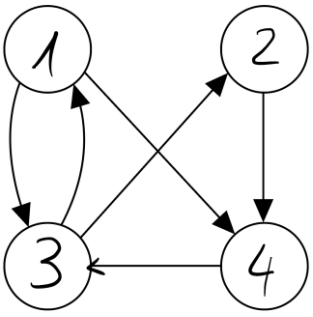
Als Base Cases haben wir $i = 0$, da wenn $i = 0$, dann gibt es nur kürzeste Pfade von einem Knoten zu sich selbst oder zu direkt benachbarten Knoten:

$$DP[u][v][0] = \begin{cases} 0 & falls\ u = v \\ w(u,v) & falls\ (u,v) \in E \\ \infty & sonst. \end{cases}$$

Jetzt zur Berechnung der restlichen Werte. Es gibt immer zwei Möglichkeiten für einen kürzesten Pfad von u nach v mit Zwischenknotennummern $\leq i$. Entweder der i-te Knoten kommt vor oder nicht. Falls nicht, dann ist der Pfad identisch zu dem mit Zwischenknotennummern $\leq i-1$. Falls i aber auf dem kürzesten Pfad vorkommt, dann ist dieser Pfad der kürzeste Pfad von u nach i verknüpft mit dem kürzesten Pfad von i nach v. Dabei dürfen auf beiden dieser Pfade die Zwischenknotennummern höchsten $i-1$ sein. Somit haben wir:

$$DP[u][v][i] = \min\left(DP[u][v][i-1], DP[u][i][i-1] + DP[i][v][i-1]\right)$$

Hier nun das Beispiel aus der Vorlesung:



i=0:

u \ v	1	2	3	4
1	0	∞	1	1
2	∞	0	∞	1
3	1	1	0	∞
4	∞	∞	1	0

i=1:

	1	2	3	4
1	0	∞	1	1
2	∞	0	∞	1
3	1	1	0	2
4	∞	∞	1	0

i=2:

	1	2	3	4
1	0	∞	1	1
2	∞	0	∞	1
3	1	1	0	2
4	∞	∞	1	0

i=3:

	1	2	3	4
1	0	2	1	1
2	∞	0	∞	1
3	1	1	0	2
4	2	2	1	0

i=4:

	1	2	3	4
1	0	2	1	1
2	3	0	2	1
3	1	1	0	2
4	2	2	1	0

Die Laufzeit für diesen Algorithmus ist sehr leicht zu bestimmen, da wir $O(|V|^3)$ Einträge haben und jeden Eintrag in konstanter Zeit berechnen können. Somit haben wir eine Gesamtlaufzeit von $O(|V|^3)$.
Da wir für die Werte des DP-Tables für ein i nur die Werte des Tables bei $i-1$ brauchen, müssen wir gar kein 3D-DP-Table, sondern können uns die Werte für $i-1$ in einem separaten 2D-Array abspeichern. Bei genauem Hinsehen fällt auf, dass man in dem Array, in dem wir die Werte für $i-1$ abgespeichert haben auch einfach die Werte für i berechnen können. Also brauchen wir nur ein $|V| \times |V|$ – Array und haben einen Extraspeicherplatz von $O(|V|^2)$.

Johnson-Algorithmus:

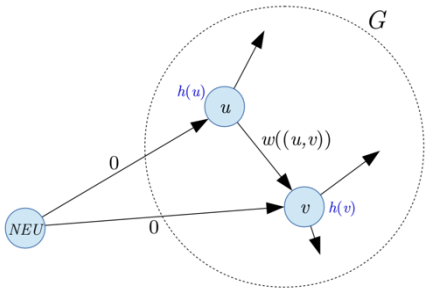
Wir haben gesehen, dass wir das Problem auch lösen können, indem wir $|V|$ mal Dijkstra laufen lassen und haben dann eine $O\big(|V|\log|V| * (|V| + |E|)\big)$ Laufzeit. Das Problem hierbei ist, dass der

Graph keine negativen Kantengewichte haben darf. Dies wollen wir nun beheben, indem wir alle Kanten positiv machen.

Wir gehen folgendermassen für einen Graphen $G = (V, E, w)$ vor: Zunächst machen wir eine Kopie $G' = (V', E', w')$ von G. Wir fügen einen neuen Knoten NEU zu V' hinzu und verbinden NEU zu jedem Knoten in V' mit einer Kante mit Gewicht 0. Dazu führen wir für jeden Knoten $v \in V' \backslash \{NEU\}$ eine Höhe $h(v)$ ein, die wir später genau wählen werden. Dazu definieren wir eine neue Kantengewichtsfunktion $w'\big((u,v)\big) = w\big((u,v)\big) + h(u) - h(v)$. Aufgrund dieser Funktion können wir 2 Eigenschaften feststellen:
1. Ein kürzester st-Pfad $P = \langle s, u_1, \dots, u_k, t\rangle$ in G ist auch ein kürzester uv-Pfad in G', da $w'(P) = w(P) + h(s) - h(t)$, da sich die restlichen Höhen aufheben und $h(s)$ und $h(t)$ sind konstant und unabhängig vom Pfad.
2. Die Kosten eines Kreises $K = \langle u_1, u_2, \dots, u_k, u_1\rangle$ bleibt bezüglich w und w' konstant, also $w(K) = w'(K)$.

Nun müssen wir die Funktion h so definieren, dass die Kantengewichtsfunktion w' stets nicht-negativ ist. Wir wählen: $h(v) =$ Die Länge des kürzesten Pfades vom neu eingefügten Knoten NEU zu v. Die Höhe von allen Knoten können wir mit einem Aufruf von Bellman-Ford berechnen. Falls wir einen negativen Kreis finden, brechen wir ab.

Durch diese Funktion haben wir für benachbarte Knoten u und v: $h(v) \leq h(u) + w\big((u,v)\big)$, da wir wissen, dass ein möglicher Pfad von NEU zu v besteht aus dem Pfad von NEU zu u plus die Kante (u,v). Somit gilt:
$w'\big((u,v)\big) = w\big((u,v)\big) + h(u) - h(v)$
$\geq w\big((u,v)\big) + h(u) - \big(h(u) + w(u,v)\big) = 0$



Somit haben wir gezeigt, dass alle Kantengewichte nicht-negativ sind. Jetzt können wir von jedem Knoten $u \in V \backslash \{NEU\}$ Dijkstra laufen lassen, um die kürzesten Pfade zu allen $v \in V \backslash \{NEU\}$ zu berechnen. Danach müssen nur noch die gefundenen Distanzen um $h(u) - h(v)$ reduziert werden.

Zur Laufzeit: Den neuen Knoten Neu und die Kanten einzufügen dauert $O(|V|)$. Der Bellman Ford Algorithmus kostet Zeit $O(|V||E|)$. Die neuen Kantengewichte $w'((u,v))$ können in $O(|E|)$ berechnet werden. Wir führen Dijkstra $|V|$ mal durch, was $O\big(|V|\log|V| * (|V| + |E|)\big)$ dauert. Somit haben wir eine Gesamtlaufzeit von $O\big(|V|\log|V| * (|V| + |E|)\big)$. Somit ist dieser Algorithmus für einen Graphen mit einer kleinen Kantenmenge schneller als der Floyd-Warschall-Algortihmus.