

Datenstrukturen und Algorithmen: Blatt 14

Bernhard Dietrich (6256800)
Lars Fernhomberg (6256030)
Sebastian Kniesburges (6257120)
Marcus Köthenbürger (6258550)

Übungsgruppe 11
Mittwoch, 11:00-13:00 Uhr in D1.303
Matthias Ernst

Aufgabe	1	2	Σ	Korrektor
Punkte				
von	6	8	14	

Aufgabe 1

Wir schreiben $C(n, k)$ für den Binomialkoeffizienten $\binom{n}{k}$. Die Binomialkoeffizienten sind

folgendermaßen rekursiv definiert:

$$C(n, k) = 1 \text{ für } k = 0 \text{ und } k = n$$

$$C(n, k) = C(n-1, k) + C(n-1, k-1) \text{ für } 0 < k < n.$$

Benutzen Sie diese rekursive Definition und die Technik der dynamischen Programmierung, um einen Algorithmus zu entwerfen, der bei Eingabe n, k den Binomialkoeffizienten $C(n, k)$ berechnet. Analysieren Sie die Laufzeit Ihres Algorithmus.

Beispiel für eine entsprechende Tabelle:

n \ k	0	1	2	3	4
1	1	1			
2	1	2	1		
3	1	3	3	1	
4	1	4	6	4	1
5	1	5	10	10	5

Binom(n, k)

```
1. for i ← 0 to n
2.   do C[i,0] ← 1
3. for i ← 1 to k
4.   do for j ← 1 to i
5.     do if j=i
6.       then C[j,i] ← 1
7.       else C[j,i] ← C[j-1,i]+ C[j-1,i-1]
8. return C[n,k]
```

In der ersten for-Schleife (Zeilen 1-2) wird zunächst die Bedingung $C(n, k) = 1$ für $k = 0$ realisiert. Anschließend (Zeilen 3-7) werden die einzelnen Zeilen des zweidimensionalen Arrays durchgegangen um die korrekten Belegungen für $C(n, k)$ zu erhalten. Dabei wird, sofern i und j gleich sind gemäß der Definition eine 1 eingetragen, andernfalls wird die Rekursionsgleichung $C(n, k) = C(n-1, k) + C(n-1, k-1)$ verwendet. Die äußere Schleife wird dabei von 1 bis k durchlaufen, während die innere Schleife von 1 bis zu dem gerade aktuellen Wert der äußeren Schleife läuft. Dieses ist notwendig, da, wenn die innere Schleife

bis zu einem größeren Wert zählen würde, eine Zahl $\binom{n}{k}$ berechnet würde, bei der n größer als k wäre, was aber gemäß Definition nicht möglich ist.

Die erste for-Schleife (Zeilen 1-2) benötigt Laufzeit $O(n)$. Die verschachtelte Vorschleife (Zeilen 3-7) benötigt $O(n \cdot k)$. Die Zeilen 2, 6-8 benötigen jeweils konstante Laufzeit $O(1)$. Da $O(1) \leq \max\{O(n), O(k)\} \leq O(n \cdot k)$ ergibt sich somit für den Algorithmus eine Laufzeit von $O(n \cdot k)$. Eine amortisierte Analyse könnte hier eine etwas realistischere obere Schranke liefern, da bei jedem Durchlauf der inneren Schleife weniger Elemente abgearbeitet werden müssen, da der Anfangspunkt steigt.

Aufgabe 2

Bei einem gerichteten, gewichteten Graph (G, w) mit $G = (V, E)$, ist w eine Funktion, die jeder Kante $(u, v) \in E$ einen positiven reellen Wert zuordnet. Wir bezeichnen den Wert $w(u, v)$ auch mit w_{uv} . Wir betrachten nun das Problem *All-Pairs-Shortest-Path (APSP)*. Hierbei ist ein gerichteter, gewichteter Graph (G, w) mit $G = (V, E)$ gegeben. Gesucht ist für alle Paare $(u, v) \in V^2$ die Länge eines kürzesten Pfades in G , der von u nach v führt. Die Länge eines Pfades ist dabei die Summe der Gewichte der Kanten auf dem Pfad. Existiert kein Pfad von u nach v , so ist die Länge des Pfades von u nach v definiert als ∞ . Benutzen Sie dynamische Programmierung, um einen Algorithmus für ASPS zu entwerfen.

Auf dem kürzesten Pfad zwischen zwei Knoten, müssen alle Teilpfade kürzeste Wege sein (da der Pfad ansonsten nicht der kürzeste wäre). Sei k nun die Anzahl der Knoten, die in dem kürzesten Pfad vorkommen, und d_{uv}^k die Länge des kürzesten Pfades, so gilt für $k = 0$:

$$d_{uv}^{(0)} = \begin{cases} 0, & \text{wenn } u = v \\ \infty, & \text{wenn } u \neq v \end{cases}$$

Für $k \geq 1$ ist d_{uv}^k das minimale Gewicht von $d_{uv}^{(k-1)}$ und dem minimalen Gewicht eines Pfades von u nach v , welcher aus mindestens k Knoten besteht. Es folgt demnach, für $n = |V|$:

$$\begin{aligned} d_{uv}^{(k)} &= \min \left(d_{uv}^{(k-1)}, \min_{1 \leq i \leq n} \left\{ d_{ui}^{(k-1)} + w_{iv} \right\} \right) \\ &= \min_{1 \leq i \leq n} \left\{ d_{ui}^{(k-1)} + w_{iv} \right\} \end{aligned}$$

Um nun den kürzesten Pfad zu ermitteln muss, ausgehend von einer Eingangsmatrix $W = (w_{uv})$, eine Reihe von Matrizen $D^{(1)}, D^{(2)}, D^{(3)}, \dots, D^{(n-1)}$ berechnet werden, wobei $D^{(k)} = (d_{uv}^{(k)})$. Die Endmatrix $D^{(n-1)}$ enthält anschließend die kürzesten Pfade.

Das Problem lässt sich somit auf Matrizenmultiplikation zurückführen:

EXTENDED-SHORTEST-PATHS (D, W)

1. $n \leftarrow \text{rows}[D]$
2. sei $D' = (d'_{ij})$ eine $n \times n$ Matrix
3. for $i \leftarrow 1$ to n
4. do for $j \leftarrow 1$ to n
5. do $d'_{ij} \leftarrow \infty$
6. for $k \leftarrow 1$ to n

7. do $d'_{ij} \leftarrow \min(d'_{ij}, d_{ik} + w_{kj})$
8. return D'

ALL-PAIRS-SHORTEST-PATHS (W)

1. $n \leftarrow \text{rows}[W]$
2. $D^{(1)} \leftarrow W$
3. for $k \leftarrow 2$ to $n-1$
4. do $D^{(k)} \leftarrow \text{EXTENDED-SHORTEST-PATHS}(D^{(k-1)}, W)$
5. return $L^{(n-1)}$