Johannes Blömer
u.v.a.m.

Paderborn, den 7. August 2004

Musterlösungen zur Vorlesung

Datenstrukturen und Algorithmen

SS 2004

Blatt 14

AUFGABE 1:
Um mit Hilfe der in der Aufgabe angegebenen Rekursion den Binomialkoeffizienten $C(n,k)$ zu berechnen, benötigen wir alle Werte $C(l,j), 1 \leq l \leq n, 1 \leq j \leq \min\{l-1,k\}$. Diese Werte berechnen wir mit dem folgenden Algorithmus BINOMIAL(n,k). Die Werte $C(n,k)$ werden dabei in einem zweidimensionalem Array $C[l][j]$ gespeichert.

BINOMIAL(n,k)

```
1 for l ← 1 to n
2     do C[l][0] ← 1
3 for l ← 1 to k
4     do C[l][l] ← 1
5 for l ← 1 to n
6     do for j ← 1 to min{l − 1, k}
7            do C[l][j] ← C[l − 1][j] + C[l − 1][j − 1]
8 return C[n][k]
```

Die Korrektheit von BINOMIAL folgt unmittelbar aus der Rekursionsformel. Nun zur Laufzeitanalyse. Zeilen 2, 4, 7 benötigen pro Durchlauf jeweils konstante Zeit. Damit benötigt die FOR-Schleife in Zeile 1 Zeit $\mathcal{O}(n)$ und die FOR-Schleife in Zeile 3 benötigt Zeit $\mathcal{O}(k)$. Schließlich benötigen die geschachtelten FOR-Schleifen in Zeilen 5,6 zusammen Zeit $\mathcal{O}(nk)$. Damit ist die Gesamtlaufzeit von BINOMIAL $\mathcal{O}(nk)$.

AUFGABE 2:
Wir setzen $d_{uv}^{(k)} = \infty$, falls kein Pfad von u nach v existiert auf dem abgesehen von u und v nur Knoten aus $\{1,\ldots,k\}$ liegen. Ausserdem definieren wir $w_{uv} = \infty$, falls in G keine Kante von u nach v existiert. Insgesamt erhalten wir dann folgende Rekursion für die Werte $d_{uv}^{(k)}$:

$$\begin{aligned} d_{uv}^{(0)} &= w_{uv} \\ d_{uv}^{(k)} &= \min\{d_{uv}^{(k-1)}, d_{uk}^{(k-1)} + d_{kv}^{(k-1)}\}, \text{ für } k > 0. \end{aligned}$$

Die erste Zeile ergibt sich dabei aus der Definition von $d_{uv}^{(0)}$ und w_{uv}. Die zweite Zeile sieht man folgendermaßen. Der kürzeste Pfad von u nach v, der ausser u,v nur Knoten aus $\{1,\ldots,k\}$ enthält, führt entweder nicht über den Knoten k oder er führt über den Knoten k. Im ersten Fall ist die Länge des kürzesten Pfades gegeben durch $d_{uv}^{(k-1)}$. Führt der kürzeste Pfad hingegegen über k, ist die Länge des kürzesten Pfades die Summe der Länge des kürzesten Pfades von u nach k und der Länge des kürzesten Pfades von k nach v. Auf beiden Pfaden dürfen aber ausser u,k bzw. k,v nur Knoten aus $\{1,\ldots,k-1\}$ liegen. In diesem Fall ist also die Länge des kürzesten Pfades gegeben durch $d_{uk}^{(k-1)} + d_{kv}^{(k-1)}$. Bilden wir das Minimum aus beiden Möglichkeiten erhalten wir die Formel

$$d_{uv}^{(k)} = \min\{d_{uv}^{(k-1)}, d_{uk}^{(k-1)} + d_{kv}^{(k-1)}\}.$$

Für $k = n$ ist $d_{uv}^{(n)}$ genau die Länge eines kürzesten Pfades von u nach v. In diesem Fall sind ja alle Knoten auf dem Pfad erlaubt. Um nun mit Hilfe der obigen Rekursion das Problem *APSP* zu lösen, nehmen wir an, dass die Gewichte w_{uv} in einem zweidimensionalen Array W gespeichert sind. Die Anzahl der Zeilen in W bezeichnen wir mit $rows[W]$. Sie ist gegeben durch die Anzahl n der Knoten in G. Weiter speichern wir für $k = 0,1,\ldots,n$ die Werte $d_{uv}^{(k)}$ in einem zweidimensionalen Array $D^{(k)}$ ab. Der folgende Algorithmus, der mit Hilfe der dynamischen Programmierung und mit der Rekursion für $d_{uv}^{(k)}$ die Länge aller kürzesten Pfade berechnet, ist als FLOYD-WARSHALL-Algorithmus bekannt. Eingabe für den Algorithmus ist nur das Array W mit den Kantengewichten w_{uv}.

FLOYD-WARSHALL(W)

```
1 n ← rows[W]
2 D^(0) ← W
3 for k ← 1 to n
4     do  for u ← 1 to n
5             do  for v ← 1 to n
6                     do  d_uv^(k) ← min(d_uv^(k-1), d_uk^(k-1) + d_kv^(k-1))
7 return D^(n)
```

Die Korrekteit des Algorithmus FLOYD-WARSHALL ergibt sich unmittelbar aus den Ausführungen von oben. Um die Laufzeit zu analysieren, beobachten wir, dass jeder Durchlauf der Zeile 6 konstante Zeile beansprucht. Wegen der dreifach geschachtelten FOR-Schleifen in Zeilen 3,4,5 wird Zeile 6 genau n^3-mal ausgeführt, wobei $n = |V|$. Damit ist die Laufzeit von FLOYD-WARSHALL $\mathcal{O}(|V|^3)$.