Uppgifter till föreläsning 1 om dynamisk programmering, DD2350 adk20

Talföljder Betrakta följande rekursion för en talföljd.

$$S_n = \begin{cases} 0 & \text{om } n = 0, \\ 1 & \text{om } n = 1, \\ 2 & \text{om } n = 2, \\ S_{n-1} + S_{n-2} + S_{n-3} & \text{om } n > 2 \end{cases}$$

Räkna ut de 7 första talen i följden för hand. Skriv pseudokod för en dynamisk programmeringsalgoritm som beräknar S_n. Hur mycket av historiken behöver man spara under beräkningens gång?

Generaliserade talföljder - tvådimensionell rekursion Givet följande rekursion, konstruera en algoritm som beräknar $M[i, j]$ med dynamisk programmering. Argumentera för att algoritmens beräkningsordning fungerar. Analysera algoritmens tidskomplexitet.

$$M[i, j] = \begin{cases} 0 & \text{om } i = 0 \text{ eller } j = 0, \\ M[i-1, j] + M[i-1, j-1] + 1 & \text{annars.} \end{cases}$$

Matriskedjemultiplikation Matriskedjemultiplikation är problemet att hitta bästa sättet att multiplicera ihop en kedja av matriser av varierande storlek. Genom att multiplicera ihop matriserna med varandra i en listig ordning kan antalet operationer som krävs minimeras. Att multiplicera en $a \times b$-matris och en $b \times c$-matris med vanlig matrismultiplikation tar abc talmultiplikationer.

Anta att man vill beräkna matrisprodukten $A_1 A_2 \cdots A_n$ där matris A_i har dimension $r_{i-1} \times r_i$. Låt $M[i, j]$ stå för det minimala antalet multiplikationer som behövs för att räkna ut matrisprodukten $A_i A_{i+1} \cdots A_j$. Rekursionsekvationen för $M[i, j]$ är:

$$M[i, j] = \begin{cases} 0 & \text{om } i = j \\ \min_{i \le k < j}(M[i, k] + M[k + 1, j] + r_{i-1} r_k r_j) & \text{om } i < j \end{cases}$$

Konstruera en beräkningsordning och en algoritm som beräknar M. Analysera algoritmens tidskomplexitet.

Lösningar

Lösning till Talföljder

0, 1, 2, 3, 6, 11, 20

För att beräkna ett tal i följden behöver man bara känna till dom tre närmast föregående talen. Beräkningsordningen blir stigande index från basfallen och uppåt.

```
Talfoljd(n) =
    s[0] = 0
    s[1] = 1
    s[2] = 2
    for i = 3 to n do
      s[i] = s[i-1]+s[i-2]+s[i-3]
    return s[n]
```

□

Lösning till Generaliserade talföljder
Vi måste börja med ett basfall, men om vi tänker oss en matris M som ska fyllas i med rätt värden, så har hela översta raden (där $i = 0$) och hela vänstra kolumnen (där $j = 0$) värdet 0, som inte beräknas rekursivt. En typisk beräkningsordning för ett program skulle vara en rad i taget uppifrån eller en kolumn i taget från vänster till höger. Vi väljer rad för rad, och sätter värdena på första raden. Därefter loopar vi över alla andra rader och sätter värdena kolumn för kolumn. Beräkningsordningen fungerar, eftersom den information vi behöver ha för att beräkna ett nytt värde utom basfallen bara är vad som stått på tidigare rader, samt indexet för den rad vi befinner oss på. Vi kommer att kunna beräkna hela M. Här är vår algoritm:

for $j \leftarrow 0$ **to** n
    $M[0, j] \leftarrow 0$
for $i \leftarrow 1$ **to** m
    $M[i, 0] \leftarrow 0$
    for $j \leftarrow 1$ **to** n
        $M[i, j] \leftarrow M[i-1, j] + M[i-1, j-1] + 1$

Tiden domineras av den nästlade for-slingan och är alltså $\Theta(nm)$. □

Lösning till Matriskedjemultiplikation
Basfallen är $M[i,i]$, det vill säga diagonalen i (den triangulära) matrisen M. Beräkningsordning: Beräkna elementen efter stigande avstånd från diagonalen, alltså så att indexen i och j skiljer sig mer och mer.

```
OptimalMatmul(M[1..n, 1..n], r) =
    for i ← 1  to n do M[i, i] ← 0
    for len ← 1  to n − 1
        for i ← 1  to n − len
            j ← i + len
            M[i, j] ← ∞
            for k ← i  to j − 1
                t ← M[i, k] + M[k + 1, j] + r_{i−1} · r_k · r_j
                if t < M[i, j] then M[i, j] ← t
```

Det minimala antalet multiplikationer beräknas av `OptimalMatmul(M, r)`. Svaret finns då i $M[1,n]$. Algoritmen har komplexiteten $O(n^3)$. □
