18. Dynamisches Programmieren: Iterierte Matrizenmultiplikation

18.1. **Definition** (Iterierte Matrixmultiplikation)

Das Problem der iterierten Matrixmultiplikation ist folgendermaßen definiert: Es ist eine Folge von n Matrizen (A_1, A_2, ..., A_n) gegeben. Die Matrix A_i ist eine $p_{i-1}\times p_i$-Matrix, wobei p_{i-1}, $p_i \in \mathbb{N}$ und i = 1, ..., n. Gesucht ist eine Klammerung, die die Anzahl der skalaren Multiplikationen bei der Berechnung von $A_1 \cdot A_2 \cdot ... \cdot A_n$ minimiert.

18.2. **Satz** (Anzahl korrekter Klammerungen)

Sei P(n) die Anzahl der korrekten Klammerungen bei n Elementen. Dann gilt:

(a) $$P(n) = \frac{1}{n}\binom{2(n-1)}{n-1}$$

$C(m) = \frac{1}{m+1}\binom{2m}{m}$ ist die Folge der *Catalanschen Zahlen*. Also ist P(n) die (n–1)-te Catalansche Zahl.

(b) Es ist $C(m) \geq \frac{2m}{m+1}$, also nicht von $O(m^k)$.

18.3. **Algorithmus** (Rekursive Lösung der iterativen Matrixmultiplikation)

Prozedur Rec-Mat-Chain(p, i, j)

```
1. if i = j then return 0
2. for all i, j do   m[i, j] := ∞
3. for k = i to j–1
4.     do   q := Rec-Mat-Chain(p, i, k) + Rec-Mat-Chain(p, k+1, j)
5.          if q < m[i, j] then m[i, j] := q
6. return m[i, j]
```

18.4. **Folgerung** (Laufzeit von Rec-Mat-Chain)

Die für die Zeilen 1. + 2. der Prozedur benötigte Zeit sei $\Omega(1)$, ebenso die für die Zeilen 5. + 6. benötigte Zeit. Sei T(m) = Zeit von Rec-Mat-Chain(p, i, j) mit j – i = m – 1. Dann gilt: $T(n) \geq 2^{n-1}$.

18.5. **Algorithmus** (Iterierte Matrixmultiplikation mit Tabellieren)

Eingabe: $p = (p_0, p_1, ..., p_n)$, $A = (A_1, ..., A_n)$
Ausgabe: Optimale Klammerung für das Matrixprodukt $A_1 \cdot A_2 \cdot ... \cdot A_n$.

Prozedur Mat-chain-order(p)

```
1. for i = 1 to n
2.     do m[i, i] := 0
3. for l = 2 to n
4.     do for i = 1 to n – l + 1
5.          do   j := i + l – 1
```

```
6.              m[i, j] := ∞
7.              for k = i to j – 1
8.                 do  q := m[i, k] + m[k+1, j] + p_{i-1}·p_k·p_j
9.                     if q < m[i, j]
10.                       then m[i, j] := q
11.                            s[i, j] := k      ; die s[i, j] werden für die Unterteilung der A_i benötigt
12. return m und s
```

Prozedur Matrix-Ketten-Mult(A, s, i, j)

```
1.  if j > i
2.     then X := Matrix-Ketten-Mult(A, s, i, s[i, j])
3.          Y := Matrix-Ketten-Mult(A, s, s[i, j]+1, j)
4.          return Matrix-Mult(X, Y)
5.  else return A_i
```