Algorithmus 6 Matrixmultiplikation (rekursiv)

Vorbedingung: A, B sind $n \times n$ Matrizen, n ist Zweierpotenz

Prozedur MATMULT(A, B)
  Sei C eine neue $n \times n$-Matrix
  Falls $n = 1$ **dann**
    $C[1,1] := A[1,1] \cdot B[1,1]$
  sonst
    $C_{1,1} :=$ MATMULT$(A_{1,1}, B_{1,1}) +$ MATMULT$(A_{1,2}, B_{2,1})$
    $C_{1,2} :=$ MATMULT$(A_{1,1}, B_{1,2}) +$ MATMULT$(A_{1,2}, B_{2,2})$
    $C_{2,1} :=$ MATMULT$(A_{2,1}, B_{1,1}) +$ MATMULT$(A_{2,2}, B_{2,1})$
    $C_{2,2} :=$ MATMULT$(A_{2,1}, B_{1,2}) +$ MATMULT$(A_{2,2}, B_{2,2})$
  Ende Falls
  Antworte C
Ende Prozedur

Der Algorithmus von Strassen folgt ebenso wie Algorithmus 6 der teile-und-herrsche-Methode. Er unterscheidet sich von ihm dadurch, dass er eine geschicktere Darstellung der $C_{i,j}$ findet, die mit sieben Multiplikationen (von $n' \times n'$-Matrizen) auskommt. Seien nämlich

$$\begin{aligned}
P_1 &= A_{1,1} \cdot (B_{1,2} - B_{2,2}),\\
P_2 &= (A_{1,1} + A_{1,2}) \cdot B_{2,2},\\
P_3 &= (A_{2,1} + A_{2,2}) \cdot B_{1,1},\\
P_4 &= A_{2,2} \cdot (B_{2,1} - B_{1,1}),\\
P_5 &= (A_{1,1} + A_{2,2}) \cdot (B_{1,1} + B_{2,2}),\\
P_6 &= (A_{1,2} - A_{2,2}) \cdot (B_{2,1} + B_{2,2}), \text{ und}\\
P_7 &= (A_{1,1} - A_{2,1}) \cdot (B_{1,1} + B_{1,2}).
\end{aligned}$$

Dann gilt

$$\begin{aligned}
C_{1,1} &= P_5 + P_4 - P_2 + P_6,\\
C_{1,2} &= P_1 + P_2,\\
C_{2,1} &= P_3 + P_4, \text{ und}\\
C_{2,2} &= P_5 + P_1 - P_3 - P_7,
\end{aligned}$$

wovon man sich durch eine kurze Rechnung überzeugen kann. Der Strassen-Algorithmus ist dann wie folgt wobei wir (wie oben) die Abkürzungen $A_{i,j}$, $B_{i,j}$ und $C_{i,j}$ auch im Pseudocode verwenden.

Algorithmus 7 Strassen-Algorithmus

Vorbedingung: A, B sind $n \times n$ Matrizen, n ist Zweierpotenz

Prozedur STRASSEN(A, B)
  Sei C eine neue $n \times n$-Matrix
  Falls $n = 1$ **dann**
    $C[1,1] := A[1,1] \cdot B[1,1]$
  sonst
    $P_1 := \text{STRASSEN}(A_{1,1}, B_{1,2} - B_{2,2})$
    $P_2 := \text{STRASSEN}(A_{1,1} + A_{1,2}, B_{2,2})$
    $P_3 := \text{STRASSEN}(A_{2,1} + A_{2,2}, B_{1,1})$
    $P_4 := \text{STRASSEN}(A_{2,2}, B_{2,1} - B_{1,1})$
    $P_5 := \text{STRASSEN}(A_{1,1} + A_{2,2}, B_{1,1} + B_{2,2})$
    $P_6 := \text{STRASSEN}(A_{1,2} - A_{2,2}, B_{2,1} + B_{2,2})$
    $P_7 := \text{STRASSEN}(A_{1,1} - A_{2,1}, B_{1,1} + B_{1,2})$
    $C_{1,1} := P_5 + P_4 - P_2 + P_6$
    $C_{1,2} := P_1 + P_2$
    $C_{2,1} := P_3 + P_4$
    $C_{2,2} := P_5 + P_1 - P_3 - P_7$
  Ende Falls
  Antworte C
Ende Prozedur

Die Laufzeit vom Strassen-Algorithmus erfüllt also:

$$T(n) = \begin{cases} \Theta(1) & \text{falls } n = 1 \\ 7T(\frac{n}{2}) + \Theta(n^2) & \text{falls } n > 1 \end{cases}$$

Im nächsten Kapitel werden wir zeigen dass die asymptotische Lösung dieser Rekursionsgleichung $T(n) = \Theta(n^{\log 7})$ ist. Da $\log 7 \approx 2{,}807$ wird dadurch eine Verbesserung des direkten sowie des rekursiven Verfahrens erreicht. Da die Konstanten des Strassen-Algorithmus aber größer sind als bei den beiden anderen Verfahren, wird in der Praxis meist eine Kombination verwendet: für hinreichend große Matrizen wird der Strassen-Algorithmus eingesetzt, für kleinere ein direkteres Verfahren.

Der Algorithmus von Strassen ist ein gutes Beispiel für einen teile-und-herrsche Algorithmus mit einer trickreichen Aufteilungs- und damit auch Kombinationsphase.

Bemerkung 3.1. Die Einschränkung auf quadratische Matrizen und n einer Zweierpotenz ist weder für den einfachen teile-und-herrsche Algorithmus, noch für den Strassen-Algorithmus notwendig. Diese Einschränkung dient lediglich der einfacheren Darstellung da durch sie einige Sonderfälle nicht behandelt werden müssen. In der Tat kann für beliebige n, m, l die Multiplikation einer $n \times m$- mit einer $m \times l$-Matrix durch Multiplikationen von Matrizen kleinerer Größe dargestellt werden, durch eine Zerlegung der Form $n = n_1 + n_2$, $m = m_1 + m_2$, $l = l_1 + l_2$. Setzt man zum Beispiel für $x \in \{n, m, l\}$ jeweils $x_1 = \lceil \frac{x}{2} \rceil$ und $x_2 = \lfloor \frac{x}{2} \rfloor$ erhält man einen allgemeineren teile-und-herrsche Algorithmus. Ähnliches gilt für den Strassen-Algorithmus. Für diesen ist allerdings notwendig dass alle $A_{i,j}$ die selbe Größe haben und dass alle $B_{i,j}$ die selbe Größe haben, d.h. also dass n, m und l gerade sind. Das kann erreicht werden durch Anfügen einer Nullzeile oder Nullspalte im Bedarfsfall.

Bemerkung 3.2. Der Algorithmus von Strassen wurde im Jahr 1969 publiziert und hat, wie beschrieben, eine Laufzeitkomplexität von $O(n^{2{,}807\cdots})$. Seitdem konnten weitere Verbesserungen der asymptotischen Laufzeit der Matrixmultiplikation erreicht werden. Ein signifikanter

Schritt vorwärts war der Algorithmus von Coppersmith-Winograd aus dem Jahr 1990 mit einer Laufzeitkomplexität von $O(n^{2,376})$. Über diesen hinaus konnten bis heute nur geringe Verbesserungen erreicht werden, so z.B. $O(n^{2,374})$ von Stothers 2010 und $O(n^{2,373})$ von Williams 2011. Diese weiteren Algorithmen haben allerdings so große Konstanten, dass sie in der Praxis keine Bedeutung haben.

Untere Schranken zur Matrixmultiplikation sind kaum bekannt. Klar ist, dass $\Omega(n^2)$ eine triviale untere Schranke ist, da die Eingabe der Größe $2n^2$ ja gelesen und die Ausgabe der Größe n^2 geschrieben werden muss. Eine untere Schranke von $\Omega(n^2 \log n)$ auf der Größe einer gewissen, eingeschränkten, Klasse von Schaltkreisen wurde von Raz 2003 bewiesen.

3.3 Dichtestes Punktepaar

Wir betrachten jetzt ein erstes fundamentales geometrisches Berechnungsproblem. Für Punkte $p_1, p_2 \in \mathbb{R}^2$ sei $\mathrm{d}(p_1, p_2)$ die übliche euklidische Distanz, d.h.

$$\mathrm{d}(\begin{pmatrix} x_1 \\ y_1 \end{pmatrix}, \begin{pmatrix} x_2 \\ y_2 \end{pmatrix}) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}.$$

Wir wollen das folgende Problem lösen:

Dichtestes Punktepaar
Eingabe: Eine endliche Menge $P \subseteq \mathbb{R}^2$
Ausgabe: $p, q \in P$ so dass $p \neq q$ und $\mathrm{d}(p, q)$ minimal

Klar ist, dass ein auf erschöpfender Suche basierender Algorithmus existiert, der einfach alle Paare von Punkten ausprobiert, siehe Algorithmus 8. Es gibt $\binom{n}{2}$ Paare die alle in jedem Fall

Algorithmus 8 Erschöpfende Suche nach dichtestem Punktepaar

```
Prozedur DPP-Suche(P)
    d_min := ∞
    Für i := 1, ..., P.Länge
        Für j := i + 1, ..., P.Länge
            Falls d(P[i], P[j]) < d_min dann
                d_min := d(P[i], P[j])
                a := (P[i], P[j])
            Ende Falls
        Ende Für
    Ende Für
    Antworte a
Ende Prozedur
```

durchlaufen werden, damit hat dieser Algorithmus Laufzeit $\Theta(n^2)$.

Wir werden sehen, dass es möglich ist mit einem Algorithmus der dem teile-und-herrsche Prinzip folgt (selbst im schlechtesten Fall) eine Laufzeit von $O(n \log n)$ zu erreichen.