**Lemma 2.3.** *Sei $G = (V, E)$ ein endlicher, nicht-leerer, zusammenhängender Graph. Dann ist $|E| \geq |V| - 1$.*

*Beweis.* Wir gehen mit Induktion nach $|V|$ vor. Falls $|V| = 1$ ist das Resultat trivial. Sei nun $|V| \geq 2$. Da $G$ zusammenhängend ist existiert ein Knoten $v$ mit $\mathrm{d}(v) \geq 1$. Seien $\{v, w_1\}, \ldots, \{v, w_k\}$ alle zu $v$ inzidenten Kanten. Sei $G' = (V', E')$ der Graph der aus $G$ entsteht wenn wir $v$ und sowie $\{v, w_1\}, \ldots, \{v, w_k\}$ löschen. Dann zerfällt $G'$ in $l \leq k$ Zusammenhangskomponenten $(V_1, E_1), \ldots, (V_l, E_l)$ und mit der Induktionshypothese gilt

$$|E| = k + \sum_{i=1}^{l} |E_i| \geq k + \sum_{i=1}^{l} (|V_i| - 1) \geq \sum_{i=1}^{l} |V_i| = |V| - 1.$$

□

**Definition 2.7.** Ein Graph $G = (V, E)$ heißt *minimal zusammenhängend* falls er zusammenhängend ist und für alle $E' \subset E$ gilt: $(V, E')$ ist nicht zusammenhängend.

**Definition 2.8.** Ein Graph $G = (V, E)$ heißt *maximal zyklenfrei* falls er zyklenfrei ist und für alle $E' \supset E$ gilt: $(V, E')$ enthält einen Zyklus.

**Satz 2.3.** *Sei $G = (V, E)$ ein endlicher, nicht-leerer Graph. Dann sind äquivalent:*

1. *$G$ ist ein Baum.*
2. *Je zwei Knoten von $G$ sind durch genau einen Pfad verbunden.*
3. *$G$ ist minimal zusammenhängend.*
4. *$G$ ist zusammenhängend und $|E| = |V| - 1$.*
5. *$G$ ist zyklenfrei und $|E| = |V| - 1$.*
6. *$G$ ist maximal zyklenfrei.*

*Beweis.* 1. ⇒ 2: Da $G$ zusammenhängend ist, gibt es von $u$ nach $v$ einen Pfad. Angenommen es gibt zwei unterschiedliche Pfade von $u$ nach $v$. Dann gibt es einen ersten Knoten $w_1$ an dem sie sich unterscheiden und einen letzten Knoten $w_2$ an dem sie sich unterschieden. Die beiden Pfade von $w_1$ nach $w_2$ bilden dann einen Zyklus.

2. ⇒ 3.: Da je zwei Knoten durch einen Pfad verbunden sind ist $G$ zusammenhängend. Sei nun $(u, v) \in E$, dann ist $u, v$ der einzige Pfad von $u$ nach $v$ und damit ist $(V, E \setminus \{\{u, v\}\})$ nicht zusammenhängend.

3. ⇒ 4.: $G$ ist zusammenhängend und damit ist $|E| \geq |V| - 1$ wegen Lemma 2.3. Es bleibt zu zeigen dass $|E| \leq |V| - 1$ ist. Angenommen $|E| > |V| - 1$ dann würde $G$ wegen Lemma 2.2 einen Zyklus enthalten. Aus diesem könnte eine beliebige Kante gelöscht werden ohne den Zusammenhang zu zerstören, $G$ wäre also nicht minimal zusammenhängend.

4. ⇒ 5.: Angenommen $G$ enthält einen Zyklus $v_1, \ldots, v_k$. Für $l \in \{k, \ldots, |V|\}$ definieren wir einen Teilgraphen $G_l = (V_l, E_l)$ von $G$ mit $|V_l| = |E_l| = l$ wie folgt: Falls $l = k$, dann ist besteht $G_k$ aus dem Zyklus $v_1, \ldots, v_k$. Sei $l > k$. Da $G$ zusammenhängend ist, existiert ein $\{v, w\} \in E$ so dass $v \in V_{l-1}$ und $w \notin V_{l-1}$. Wir definieren $V_l = V_{l-1} \cup \{w\}$ und $E_l = E_{l-1} \cup \{\{v, w\}\}$. Dann gilt $|V_l| = |E_l| = l$. Für $l = |V|$ erhalten wir also einen Teilgraphen $(V, E_{|V|})$ von $(V, E)$. Damit ist $|E_V| = |V| \leq |E|$ was aber $|E| = |V| - 1$ widerspricht.

5.⇒ 6.: Angenommen es gäbe $E' \supset E$ so dass $(V, E')$ zyklenfrei wäre, dann wäre wegen Lemma 2.2 ja $|E'| \leq |V| - 1$ was $|E'| > |E| = |V| - 1$ widerspricht.

6.⇒ 1.: Es reicht zu zeigen dass $G$ zusammenhängend ist. Angenommen $G$ wäre nicht zusammenhängend. Seien dann $v$ und $w$ Knoten aus verschiedenen Zusammenhangskomponenten und $G' = (V, E \cup \{\{v, w\}\})$. Dann ist $G'$ immer noch zyklenfrei, denn jeder Zyklus in $G'$ wäre entweder Zyklus in $G$ (Widerspruch) oder er würde die Kante $\{v, w\}$ mindestens zwei Mal enthalten woraus sich ein Zyklus in jeder der beiden Zusammenhangskomponenten bilden ließe (Widerspruch). $G$ ist also nicht maximal zyklenfrei, Widerspruch. □

**Korollar 2.1.** *Ein endlicher Wald $(V, E)$ besteht aus $|V| - |E|$ Bäumen.*

*Beweis.* Angenommen $(V, E)$ besteht aus den $m$ Bäumen $(V_1, E_1), \ldots, (V_m, E_m)$. Dann ist

$$|E| = \sum_{i=1}^{m} |E_i| \overset{\text{Satz 2.3}}{=} \sum_{i=1}^{m} (|V_i| - 1) = \sum_{i=1}^{m} |V_i| - m = |V| - m$$

und damit $m = |V| - |E|$. □

Wir kommen jetzt zu einer Anwendung von Bäumen. Angenommen wir wollen eine Menge $V$ von Orten (z.B. Computern in einem Gebäude oder Pins auf einer Leiterplatte) verbinden. Die Knoten $V$ bilden gemeinsam mit möglichen Verbindungen $E \subseteq V \times V$ einen ungerichteten Graphen. Zusätzlich sind die Kosten des Legens einer Verbindung bekannt, d.h. eine Kostenfunktion $c : E \to \mathbb{R}_{\geq 0}$ ist gegeben. Der Graph der Verbindungen soll also $G' = (V, E')$ sein wobei $E' \subseteq E$ ist, $G'$ zusammenhängend ist und $\sum_{e \in E} c(e)$ minimal sein soll.

**Korollar 2.2.** *Sei $G = (V, E)$ ein zusammenhängender Graph und $E' \subseteq E$ so dass $G' = (V, E')$ minimal zusammenhängend ist. Dann ist $G'$ ein Baum.*

*Beweis.* Folgt unmittelbar aus Satz 2.3. □

Der gesuchte Verbindungsgraph ist also ein Baum. Das motiviert die folgenden Definitionen und das folgende Berechnungsproblem.

**Definition 2.9.** Sei $G = (V, E)$ ein zusammenhängender Graph. Ein *Spannbaum* von $G$ ist ein Baum $B = (V, E')$ mit $E' \subseteq E$.

Der Baum $B$ spannt also $G$ auf.

**Definition 2.10.** Sei $G = (V, E)$ ein zusammenhängender Graph und sei $c : E \to \mathbb{R}_{\geq 0}$. Ein *minimaler Spannbaum* von $G$ bezüglich $c$ ist ein Spannbaum $B = (V, E')$ von $G$ so dass $\sum_{e \in E'} c(e)$ minimal ist unter aller Spannbäumen von $G$.

| **Minimaler Spannbaum** |
| --- |
| Eingabe: Ein zusammenhängender Graph $G$ und eine Kostenfunktion $c : E \to \mathbb{R}_{\geq 0}$ |
| Ausgabe: Ein minimaler Spannbaum von $G$ bezüglich $c$ |

Wir werden später effiziente Algorithmen zur Bestimmung eines minimalen Spannbaums kennen lernen.

# Kapitel 3

# Teile und Herrsche

In der Praxis auftretende Berechnungsprobleme haben oft die Eigenschaft dass ihre Instanzen zerlegt werden können und Lösungen der kleineren Instanzen zur Lösung der ursprünglichen Instanz hilfreich sind. Diese Vorgehensweise zur Lösung eines Berechnungsproblems ist so weit verbreitet, dass sich dafür ein eigener Begriff eingebürgert hat: die *teile-und-herrsche Strategie*. Algorithmen, die der teile-und-herrsche Strategie folgen bestehen üblicherweise aus drei Phasen:

1. *Aufteilung* der Eingabeinstanz in mehrere Instanzen kleinerer Größe
2. *Lösung* der kleineren Instanzen durch rekursiven Aufruf des Algorithmus
3. *Kombination* der Lösungen der kleineren Instanzen zu einer Lösung der ursprünglichen Instanz

Die Basis dieser Rekursion wird normalerweise dadurch gebildet, dass Instanzen deren Größe eine Konstante ist trivial gelöst werden können.

## 3.1 Sortieren durch Verschmelzen

Das Sortierproblem kann durch einen teile-und-herrsche Algorithmus gelöst werden. Die Grundidee dazu ist 1. das Eingabedatenfeld in zwei (zirka) gleich große Teile zu teilen, 2. jeden dieser beiden Teile unabhängig vom anderen durch einen rekursiven Aufruf zu sortieren und 3. die beiden sortierten Datenfelder zu einem einzigen sortierten Datenfeld zu verschmelzen. Deshalb wird dieser Algorithmus auch als "Sortieren durch Verschmelzen" (engl. *merge sort*) bezeichnet. Von den beiden Schritten 1 und 2 sollte klar sein wie sie umgesetzt werden können. Der Ansatz zur Realisierung des 3. Schritts besteht darin zwei Indizes zu verwenden, jeweils einen für jedes der Eingabedatenfelder, und sie so durch die Eingabedatenfelder laufen zu lassen, dass das aktuelle Minimum immer unter einem der beiden steht. Dieses aktuelle Minimum wird in das Ausgabedatenfeld übertragen. In Algorithmus 3 ist diese Idee als Pseudocode ausformuliert.

Basierend auf der Prozedur Verschmelzen kann nun Sortieren durch Verschmelzen wie in Algorithmus 4 als Pseudocode formuliert werden. Für $i \leq j$ ist die Notation $A := B[i, \ldots, j]$ eine Abkürzung für die Herstellung eines Datenfeldes $A$ dessen Inhalt genau den Einträgen $i$ bis $j$ des Datenfeldes $B$ entspricht. Eine solche Kopie kann durch eine einfache Schleife in Laufzeit $\Theta(j - i)$ hergestellt werden.

Wir wollen nun die Laufzeit von Sortieren durch Verschmelzen analysieren. Die Prozedur "Verschmelzen" benötigt Laufzeit $\Theta(A.Länge + B.Länge)$. Sei $T(n)$ die Laufzeit von Sortieren durch

**Algorithmus 3** Verschmelzen

**Prozedur** VERSCHMELZEN($A, B$)
  Sei $C$ ein neues Datenfeld der Länge $A.Länge + B.Länge$
  $i := 1$
  $j := 1$
  **Für** $k := 1, \ldots, A.Länge + B.Länge$
    **Falls** $j > B.Länge$ **oder** ( $i \leq A.Länge$ **und** $A[i] \leq B[j]$ ) **dann**
      $C[k] := A[i]$
      $i := i + 1$
    **sonst**
      $C[k] := B[j]$
      $j := j + 1$
    **Ende Falls**
  **Ende Für**
  **Antworte** $C$
**Ende Prozedur**

**Algorithmus 4** Sortieren durch Verschmelzen (engl. *merge sort*)

**Prozedur** VSORTIEREN($A$)
  **Falls** $A.Länge = 1$ **dann**
    **Antworte** $A$
  **sonst**
    $m := \left\lceil \frac{A.Länge}{2} \right\rceil$
    $L := A[1, \ldots, m]$
    $R := A[m + 1, \ldots, A.Länge]$
    $L' :=$ VSORTIEREN($L$)
    $R' :=$ VSORTIEREN($R$)
    **Antworte** VERSCHMELZEN($L', R'$)
  **Ende Falls**
**Ende Prozedur**

Verschmelzen wenn das Eingabedatenfeld die Länge $n$ hat. Klar ist bereits dass $T(1) = \Theta(1)$. Für $n \geq 2$ beobachten wir: Die Aufteilung in Teilprobleme benötigt Laufzeit $\Theta(n)$, das Lösen der Teilprobleme benötigt $T(\lceil \frac{n}{2} \rceil) + T(\lfloor \frac{n}{2} \rfloor)$, das Verschmelzen der beiden Lösungen, die ja Größe $m$ und $n-m$ haben, benötigt $\Theta(n)$. Insbesondere können wir hier beobachten, dass die Laufzeit von Sortieren durch Verschmelzen, anders als die von Einfügesortieren, nur von $n = A.\mathit{Länge}$ abhängt, nicht aber vom Inhalt von $A$. Somit ist die Laufzeit im besten Fall gleich der Laufzeit im schlechtesten Fall und gleich der Laufzeit im Durchschnittsfall.

Insgesamt erhalten wir für die Laufzeit

$$T(n) = \begin{cases} \Theta(1) & \text{für } n = 1 \\ T(\lceil \frac{n}{2} \rceil) + T(\lfloor \frac{n}{2} \rfloor) + \Theta(n) & \text{für } n \geq 2 \end{cases}.$$

Bei dieser Darstellung der Laufzeit handelt es sich um eine *Rekursionsgleichung*. Die Analyse eines teile-und-herrsche Algorithmus führt typischerweise auf eine Rekursionsgleichung einer solchen Form. In Kapitel 4 werden wir zeigen, dass für die obige Rekursionsgleichung $T(n) = \Theta(n \log n)$ gilt[1]. Für den Augenblick wollen wir uns damit begnügen die folgende einfache Beobachtung zu machen, die den Kern der obigen Rekursionsgleichung enthält.

**Satz 3.1.** *Sei* $S : \{2^k \mid k \geq 1\} \to \mathbb{N}$ *definiert durch* $S(n) = \begin{cases} 2 & \textit{falls } n = 2 \\ 2S(\frac{n}{2}) + n & \textit{falls } n > 2 \end{cases}$, *dann ist* $S(n) = n \log n$.

*Beweis.* Zu zeigen ist also, für alle $k \geq 1$, dass $S(2^k) = k \cdot 2^k$. Für $k = 1$ ist das per definitionem erfüllt. Für $k \geq 2$ haben wir $S(2^k) = 2 \cdot S(2^{k-1}) + 2^k =^{\text{IH}} 2 \cdot (k-1) \cdot 2^{k-1} + 2^k = k \cdot 2^k$. □

## 3.2 Matrixmultiplikation

Wir werden annehmen, dass eine Matrix im Speicher abgelegt wird als ein Datenfeld, dessen Elemente Datenfelder einer festen Länge sind. Falls also $A$ eine Matrix ist, so ist $A[i]$ die $i$-te Zeile von $A$ und damit $A[i][j]$ das $j$-te Element der $i$-ten Zeile. Um die Notation etwas abzukürzen, schreiben wir stattdessen auch $A[i, j]$. Wir betrachten das folgende Problem:

| **Matrixmultiplikation** |
|---|
| Eingabe: eine $n \times m$-Matrix $A$ und eine $m \times l$-Matrix $B$ |
| Ausgabe: $A \cdot B$ |

Seien $n, m, l \geq 1$ und $A = (a_{i,j})_{\substack{1 \leq i \leq n \\ 1 \leq j \leq m}}$ und $B = (b_{i,j})_{\substack{1 \leq i \leq m \\ 1 \leq j \leq l}}$ Matrizen. Dann ist deren Produkt $A \cdot B = C = (c_{i,j})_{\substack{1 \leq i \leq n \\ 1 \leq j \leq l}}$ bekannterweise durch

$$c_{i,j} = \sum_{k=1}^{m} a_{i,k} \cdot b_{k,j}$$

definiert. Diese Formel induziert sofort Algorithmus 5. Dessen Laufzeit kann aufgrund der drei verschachtelten Schleifen sofort als $\Theta(n \cdot l \cdot m)$ erkannt werden.

[1] In dieser Vorlesung werden wir mit log immer den Logarithmus zur Basis 2 notieren

**Algorithmus 5** Matrixmultiplikation (direkt)

**Vorbedingung:** $n, m, l \geq 1$, $A$ ist $n \times m$-Matrix, $B$ ist $m \times l$-Matrix

```
Prozedur MATMULT(A, B)
    Sei C eine neue n × l-Matrix
    Für i := 1, ..., n
        Für j := 1, ..., l
            C[i, j] := 0
            Für k := 1, ..., m
                C[i, j] := C[i, j] + A[i, k] · B[k, j]
            Ende Für
        Ende Für
    Ende Für
    Antworte C
Ende Prozedur
```

Um die Analyse zu vereinfachen, werden wir von nun an annehmen, dass $n = m = l$ und dass $n$ eine Zweierpotenz ist. Für $n = 2n'$ kann eine $n \times n$-Matrix in vier $n' \times n'$-Matrizen geteilt werden.

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \quad B = \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} \quad C = \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}$$

Damit kann eine $n \times n$-Matrix mit Elementen aus einer Menge $R$ identifiziert werden mit einer $2 \times 2$-Matrix deren Elemente $n' \times n'$-Matrizen mit Elementen aus $R$ sind (Diese Beobachtung kann präzisiert werden, z.B. für einen Ring $R$, indem man zeigt, dass die Matrizenringe $R^{n \times n}$ und $(R^{n' \times n'})^{2 \times 2}$ isomorph sind). Für $C = A \cdot B$ erhalten wir damit die Darstellung

$$\begin{aligned} C_{1,1} &= A_{1,1}B_{1,1} + A_{1,2}B_{2,1} \\ C_{1,2} &= A_{1,1}B_{1,2} + A_{1,2}B_{2,2} \\ C_{2,1} &= A_{2,1}B_{1,1} + A_{2,2}B_{2,1} \\ C_{2,2} &= A_{2,1}B_{1,2} + A_{2,2}B_{2,2} \end{aligned}$$

der $n \times n$-Multiplikation durch $n' \times n'$-Multiplikation. Diese Beobachtung induziert das in Algorithmus 6 angegebene einfache teile-und-herrsche Verfahren. Hier schreiben wir im Sinne der soeben diskutierten Teilung $A_{1,1}$ für $A[1, \ldots, n'; 1, \ldots, n']$, $A_{1,2}$ für $A[1, \ldots, n'; n' + 1, \ldots, n]$, usw. auch für $B$ und $C$. Die Laufzeit dieses Algorithmus erfüllt also:

$$T(n) = \begin{cases} \Theta(1) & \text{falls } n = 1 \\ 8T(\frac{n}{2}) + \Theta(n^2) & \text{falls } n \geq 2 \end{cases}$$

Diese Rekursionsgleichung hat die asymptotische Lösung $T(n) = \Theta(n^3)$ wie wir im Kapitel 4 sehen werden. Dieser teile-und-herrsche Algorithmus ist also asymptotisch nicht effizienter als das direkte Verfahren. Man könnte nun glauben, dass der teile-und-herrsche-Ansatz für die Matrixmultiplikation keine Verbesserung des direkten Verfahren erlaubt, oder sogar dass ein Algorithmus mit einer Laufzeit von weniger als $\Theta(n^3)$ gar nicht möglich ist, da dies ja der natürlichen Definition des Produkts entspricht. Beides wäre allerdings ein Irrtum wie der Algorithmus von Strassen zeigt.