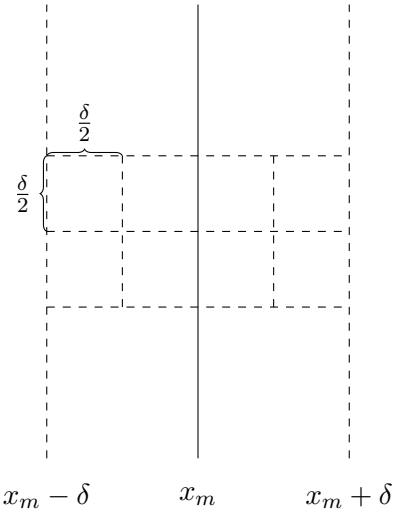

Abbildung 3.1: Kombination der Lösungen der Teilprobleme

Sei $P \subseteq \mathbb{R}^2$ eine endliche Menge von Punkten. Die Grundidee des Verfahrens ist wie folgt:

1. *Aufteilung*: Wir teilen das Problem entlang einer vertikalen Gerade, genauer: Sei $x_m \in \mathbb{R}$ und $P = Q \uplus R$ so dass $|Q| = \left\lceil \frac{|P|}{2} \right\rceil$, $|R| = \left\lfloor \frac{|P|}{2} \right\rfloor$, für alle $\begin{pmatrix} x \\ y \end{pmatrix} \in Q$: $x \leq x_m$ und für alle $\begin{pmatrix} x \\ y \end{pmatrix} \in R$: $x \geq x_m$. Man beachte, dass sowohl P als auch Q Punkte mit x-Koordinate x_m enthalten können.

2. *Lösung*: Mit Hilfe rekursiver Aufrufe bestimmen wir das dichteste Punktepaar $q_1, q_2 \in Q$ sowie das dichteste Punktepaar $r_1, r_2 \in R$. Sei $\delta = \min\{\mathrm{d}(q_1, q_2), \mathrm{d}(r_1, r_2)\}$.

3. *Kombination*: Das dichteste Punktepaar von P ist nun entweder q_1, q_2 oder r_1, r_2 oder ein Paar q, r wobei $q \in Q$ und $r \in R$. Natürlich können wir jetzt nicht alle (quadratisch vielen) Paare in $Q \times R$ durchsuchen wenn wir nur Laufzeit $O(n \log n)$ verwenden wollen. Mit einer kurzen Überlegung kann man aber zeigen, dass es ausreicht linear viele Paare zu überprüfen. Der Schlüssel dazu ist, dass wir δ bereits kennen. Falls nämlich ein Paar $q = \begin{pmatrix} q_x \\ q_y \end{pmatrix} \in Q$, $r = \begin{pmatrix} r_x \\ r_y \end{pmatrix} \in R$ das dichteste Punktepaar von P ist, muss nämlich $q_x, r_x \in [x_m - \delta, x_m + \delta]$ sein, d.h. dass q und r in einem Schlauch der Breite 2δ um x_m liegen. Weiters muss natürlich auch $|q_y - r_y| < \delta$ sein. Also liegen q und r in einem $2\delta \times \delta$ großen Rechteck das um die Gerade $x = x_m$ zentriert ist. Dieses Rechteck stellen wir uns nun als unterteilt in 8 Zellen der Größe $\frac{\delta}{2} \times \frac{\delta}{2}$ vor, siehe Abbildung 3.1. Jeder der Zellen (als Produkt geschlossener Intervalle) auf der linken Seite (von $x = x_m$) enthält höchstens einen Punkt von Q: würde eine Zelle nämlich zwei Punkte $q_1', q_2' \in Q$ enthalten, dann wäre $\mathrm{d}(q_1', q_2') \leq \sqrt{\left(\frac{\delta}{2}\right)^2 + \left(\frac{\delta}{2}\right)^2} = \frac{\delta}{\sqrt{2}} < \delta$ was ein Widerspruch ist. Analog dazu enthält auch jede Zelle auf der rechten Seite höchstens einen Punkt von R. In diesem Rechteck gibt es also höchstens 8 Punkte von P. Um alle für das dichteste in Frage kommenden Punktepaare $q \in Q, r \in R$ zu überprüfen reicht es also, die Punkte im Schlauch rund um die Gerade $x = x_m$ nach y-Koordinate zu sortieren und für jeden Punkt den Abstand zu seinen 7 nächsten in der sortierten Liste zu überprüfen. Falls auf diese Weise ein Punktepaar $q \in Q, r \in R$ mit $\mathrm{d}(q, r) < \delta$ gefunden wird, so ist dieses das dichteste in P, falls nicht, dann ist das dichteste Punktepaar in P je nachdem entweder q_1, q_2 oder r_1, r_2. Damit haben wir uns also geometrisch davon überzeugt, dass diese Idee für einen teile-und-herrsche-Algorithmus sinnvoll ist und wir können uns an die konkrete Realisierung machen.

Ein Aspekt der durch die obige Diskussion noch nicht vollständig festgelegt ist, ist was passieren soll wenn mehrere Punkte in P auf der Gerade $x = x_m$ liegen. In solchen nicht-deterministischen Situationen ist es häufig nützlich, eine einfach zu berechnende Determinisierung festzulegen. Dazu definieren wir:

Definition 3.1. Die *lexikographische Ordnung* $<_{\text{lex}}$ auf $\mathbb{R}^2$ ist festgelegt durch:

$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} <_{\text{lex}} \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} \text{ genau dann wenn } 1.\ x_1 < x_2 \text{ oder}$$
$$2.\ x_1 = x_2 \text{ und } y_1 < y_2.$$

Wir schreiben $\leq_{\text{lex}}$ für die reflexive Hülle von $<_{\text{lex}}$ und beobachten, dass $<_{\text{lex}}$ eine totale Ordnung auf $\mathbb{R}^2$ ist. Wir können die Aufteilung des Problems für die Menge P nun also deterministisch spezifizieren als: Sei $p_m \in P$, $Q = \{p \in P \mid p \leq_{\text{lex}} p_m\}$, $R = \{p \in P \mid p >_{\text{lex}} p_m\}$ so dass $|Q| = \left\lceil \frac{|P|}{2} \right\rceil$ und $R = \left\lfloor \frac{|P|}{2} \right\rfloor$. Dann ist x_m die x-Koordinate von p_m. Falls also mehrere Punkte in P die x-Koordinate x_m haben gibt es auf der Gerade $x = x_m$ einen Punkt so dass alle darunter liegenden Punkte in Q sind und alle darüber liegenden in R.

Wir benötigen also eine Darstellung der Punkte sortiert nach $\leq_{\text{lex}}$ sowie eine in der die selben Punkte nach y-Koordinate sortiert sind. Das können wir erreichen, indem wir zwei Datenfelder verwenden die die selben Punkte enthalten: eines wird nach $\leq_{\text{lex}}$ sortiert und eines nach der y-Koordinate. Ein wichtiger Punkt ist nun dass wir es uns nicht leisten können, nach der Teilung erneut zu sortieren. Das würde in jedem Schritt Zeit $O(n \log n)$ verbrauchen und um eine Gesamtlaufzeit von $O(n \log n)$ zu erreichen müssen wir in jedem Schritt mit Zeit $O(n)$ auskommen. Dieses Hindernis kann überwunden werden, indem wir diese beiden Sortierungen einmal zu Beginn des Algorithmus (in Zeit $O(n \log n)$) berechnen und dann sicherstellen, dass die Sortierung durch den gesamten Algorithmus hindurch beibehalten wird.

Zur Realisierung dieses Ansatzes benötigen wir noch eine weitere Operation. Falls wir aus einer Menge X alle Elemente x auswählen wollen, die eine Eigenschaft $P(x)$ haben, dann schreiben wir diese Menge als $\{x \in X \mid P(x)\}$. Eine analoge Operation auf Datenfeldern kann wie folgt definiert werden: Sei A ein Datenfeld, dann ist $\langle x \in A \mid P(x) \rangle$ ein Datenfeld das alle Elemente von A enthält, die die Eigenschaft P erfüllen und zwar *in der Reihenfolge, in der sie in A vorkommen*. Algorithmus 9 führt diese Auswahl durch. Falls die Überprüfung der Eigenschaft

Algorithmus 9 Auswahl aus einem Datenfeld

```
Prozedur AUSWAHL_P(A)
    Sei B ein neues Datenfeld
    j := 1
    Für i = 1, ..., A.Länge
        Falls P(A[i]) dann
            B[j] := A[i]
            j := j + 1
        Ende Falls
    Ende Für
    Antworte B
Ende Prozedur
```

P konstante Zeit erfordert (was üblicherweise bei uns der Fall sein wird), dann läuft dieser Algorithmus in Zeit $\Theta(n)$ wobei $n = A.\text{Länge}$.

Der gesamte Algorithmus zur Bestimmung des dichtesten Punktepaares kann als Pseudocode wie in Algorithmus 10 geschrieben werden.

Algorithmus 10 Teile-und-herrsche Algorithmus für dichtestes Punktepaar

Prozedur DPP-TH(P)
  $P := P$ aufsteigend nach $\leq_{\text{lex}}$ sortiert
  $P_y := P$ aufsteigend nach y-Koordinate sortiert
  Antworte DPP-TH-Rek(P, P_y)
Ende Prozedur

Prozedur DPP-TH-Rek(P, P_y)
  Falls $P.Länge \leq 3$ **dann**
    Antworte DPP-Suche(P)
  Ende Falls

  $m := \left\lceil \frac{P.Länge}{2} \right\rceil$
  $Q := P[1, \ldots, m]$
  $R := P[m+1, \ldots, P_x.Länge]$
  $Q_y := \langle p \in P_y \mid p \leq_{\text{lex}} P[m] \rangle$
  $R_y := \langle p \in P_y \mid p >_{\text{lex}} P[m] \rangle$

  $(q_1, q_2) :=$ DPP-TH-Rek(Q, Q_y)
  $(r_1, r_2) :=$ DPP-TH-Rek(R, R_y)
  $\delta := \min\{\mathrm{d}(q_1, q_2), \mathrm{d}(r_1, r_2)\}$

  $S_y := \langle p \in P_y \mid P[m].x - \delta \leq p.x \leq P[m].x + \delta \rangle$
  $s_1 := (0, 0)$
  $s_2 := (\infty, \infty)$
  Für $i := 1, \ldots, S_y.Länge - 1$
    Für $j := i+1, \ldots, \min\{i+7, S_y.Länge\}$
      Falls $\mathrm{d}(S_y[i], S_y[j]) < \mathrm{d}(s_1, s_2)$ **dann**
        $s_1 := S_y[i]$
        $s_2 := S_y[j]$
      Ende Falls
    Ende Für
  Ende Für

  Falls $\mathrm{d}(s_1, s_2) < \delta$ **dann**
    Antworte (s_1, s_2)
  sonst falls $\mathrm{d}(r_1, r_2) < \mathrm{d}(q_1, q_2)$ **dann**
    Antworte (r_1, r_2)
  sonst
    Antworte (q_1, q_2)
  Ende Falls
Ende Prozedur

Für die Laufzeitanalyse von DPP-TH sei $n = |P|$. Die Laufzeit von DPP-TH ist $\Theta(n \log n)$ für das Sortieren plus der Laufzeit von DPP-TH-Rek. Für die Laufzeitkomplexität von DPP-TH-Rek erhalten wir

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 3 \\ T(\lceil \frac{n}{2} \rceil) + T(\lfloor \frac{n}{2} \rfloor) + \Theta(n) & \text{if } n > 3 \end{cases}$$

Diese Rekursionsgleichung ist beinahe identisch zu jener die wir aus Sortieren durch Verschmelzen erhalten haben. In der Tat gilt auch hier $T(n) = \Theta(n \log n)$ wie wir in Kapitel 4 zeigen werden. Somit ist auch die Laufzeit des Gesamtalgorithmus DPP-TH $\Theta(n \log n)$.

Wir sehen also, dass die Kombinationsphase in einem teile-und-herrsche-Algorithmus durchaus auch trickreicher sein kann als das beim Sortieren durch Verschmelzen der Fall ist.

Kapitel 4

Rekursionsgleichungen

Eine Rekursionsgleichung definiert eine Folge $(x_n)_{n\geq 0}$ durch eine Gleichung, die x_n basierend auf n und den x_i mit $i < n$ definiert sowie hinreichend vielen Anfangswerten. Typischerweise ist man daran interessiert, einen geschlossenen Ausdruck für x_n zu finden, d.h. einen, in dem keine x_i mehr vorkommen.

Beispiel 4.1. Die bekannte Rekursionsgleichung

$$F_n = F_{n-1} + F_{n-2} \text{ für } n \geq 2 \text{ mit } F_0 = 0 \text{ und } F_1 = 1$$

definiert die Folge der Fibonacci-Zahlen $0, 1, 1, 2, 3, 5, 8, 13, \ldots$.

4.1 Lineare Rekursionsgleichungen erster Ordnung

Definition 4.1. Die *Ordnung* einer Rekursionsgleichung ist das kleinste k so dass die Definition von x_n von $\{x_{n-1}, \ldots, x_{n-k}\}$ abhängt.

So hat zum Beispiel die Fibonacci-Gleichung die Ordnung 2.

Definition 4.2. Eine *lineare Rekursionsgleichung erster Ordnung* ist eine Rekursionsgleichung der Form

$$x_n = a_n x_{n-1} + b_n \text{ für } n \geq 1$$

mit festgelegten Anfangswert x_0.

Beispiel 4.2. Die Rekursionsgleichung

$$x_n = x_{n-1} + b_n \text{ für } n \geq 1 \text{ mit } x_0 = 0$$

hat als Lösung $x_n = \sum_{i=1}^{n} b_i$.

Die Rekursionsgleichung

$$x_n = a_n x_{n-1} \text{ für } n \geq 1 \text{ mit } x_0 = 1$$

hat als Lösung $x_n = \prod_{i=1}^{n} a_i$.

Diese beiden Beispiele können zu dem folgenden Resultat verallgemeinert werden:

Satz 4.1. *Die Rekursionsgleichung $x_n = a_n x_{n-1} + b_n$ für $n \geq 1$ mit festgelegten Anfangswert $x_0 = b_0$ hat die Lösung*

$$x_n = \sum_{i=0}^{n} b_i \prod_{j=i+1}^{n} a_j.$$

Beweis. Mit Induktion: Für $n = 0$ gilt $x_0 = b_0$. Für $n > 0$ haben wir

$$x_n = a_n x_{n-1} + b_n = a_n \left(\sum_{i=0}^{n-1} b_i \prod_{j=i+1}^{n-1} a_j \right) + b_n = \left(\sum_{i=0}^{n-1} b_i \prod_{j=i+1}^{n} a_j \right) + b_n = \sum_{i=0}^{n} b_i \prod_{j=i+1}^{n} a_j.$$

□

4.2 Lineare Rekursionsgleichungen k-ter Ordnung

Wir wollen nun lineare Rekursionsgleichungen beliebiger Ordnung betrachten, schränken uns dabei aber auf den Fall konstanter Koeffizienten ein.

Definition 4.3. Eine *homogene lineare Rekursionsgleichung mit konstanten Koeffizienten k-ter Ordnung* ist von der Form

$$x_n = c_{k-1} x_{n-1} + \cdots + c_0 x_{n-k} \text{ für } n \geq k \tag{4.1}$$

Falls die Konstanten $c_0, \ldots, c_{k-1}$ Elemente eines Körpers K sind, können wir die Rekursionsgleichung (4.1) über diesem Körper K auffassen. Wir sagen dass $(a_n)_{n\geq 0}$ eine Lösung von (4.1) in K ist falls alle $a_n \in K$ sind und $a_n = c_{k-1} a_{n-1} + \cdots + c_0 a_{n-k}$ für alle $n \geq k$. Wir definieren $K^\omega = \{(a_n)_{n\geq 0} \mid a_n \in K \text{ für alle } n \geq 0\}$ und stellen fest, dass K^ω ein Vektorraum unendlicher Dimension über K ist.

Lemma 4.1. *Sei K ein Körper, seien $c_0, \ldots, c_{k-1} \in K$, dann ist die Lösungsmenge von (4.1) in K ein Untervektorraum von K^ω mit Dimension k.*

Beweis. Klar ist, dass die Lösungsmenge eine Teilmenge von K^ω ist. Weiters ist die Lösungsmenge nicht leer, da z.B. $(0)_{n\geq 0}$ eine Lösung von (4.1) ist. Seien nun $(a_n)_{n\geq 0}, (b_n)_{n\geq 0}$ Lösungen von (4.1) und $\lambda, \mu \in K$, dann ist für $n \geq k$

$$\begin{aligned} a_n &= c_{k-1} a_{n-1} + \ldots + c_0 a_{n-k}, \\ b_n &= c_{k-1} b_{n-1} + \ldots + c_0 b_{n-k}, \end{aligned}$$

und damit auch

$$\lambda a_n + \mu b_n = c_{k-1}(\lambda a_{n-1} + \mu b_{n-1}) + \ldots + c_0(\lambda a_{n-1} + \mu b_{n-1}),$$

d.h. also auch $(\lambda a_n + \mu b_n)_{n\geq 0}$ ist eine Lösung von (4.1) und damit ist die Lösungsmenge ein Unterraum von K^ω. Seien $a_0, \ldots, a_{k-1}$ beliebig, dann ist $(a_n)_{n\geq 0}$ eindeutig bestimmt, und kann geschrieben werden als $(a_n)_{n\geq 0} = a_0 e_0 + \cdots + a_{k-1} e_{k-1}$ wobei $e_i \in K^\omega$ jene Lösung von (4.1) ist, die durch $e_{i,j} = \delta_{i,j}$ für $j = 0, \ldots, k-1$ bestimmt wird. Also ist $e_0, \ldots, e_{k-1}$ Basis des Lösungsraums und damit ist seine Dimension k. □