Institut für Informatik
Lehrstuhl für Bioinformatik

Ludwig
Maximilians
Universität
München
LMU

# Skriptum
## zur Vorlesung
# Algorithmische Bioinformatik III

*gehalten im Sommersemester 2003*

*Volker Heun*

*Erstellt unter Mithilfe von:*

*Sabine Spreer*

**18. September 2003**
*Version 0.18*

# Vorwort

Dieses Skript entstand parallel zu der Vorlesung *Algorithmische Bioinformatik III* des Sommersemester 2003, die als Ergänzung zu den Vorlesungen *Algorithmische Bioinformatik I* und *Algorithmische Bioinformatik II* dient. Diese Vorlesung wurde speziell für Studenten der Bioinformatik an der Ludwig-Maximilians-Universität München im Rahmen des von der Ludwig-Maximilians-Universität München und der Technischen Universität München gemeinsam veranstalteten Studiengangs Bioinformatik gehalten.

Durch den Neuaufbau des Studienganges bedingt, unterscheiden sich die bisherigen Inhalte der bereits angebotenen Vorlesungen Algorithmische Bioinformatik I und II leicht, so dass dieser dritte Teil ebenfalls Überschneidungen mit den bisher gehaltenen ersten und zweiten Teilen besitzt.

Diese Fassung ist zwar korrigiert, aber noch nicht prinzipiell überarbeitet worden, so dass das Skript an einigen Stellen etwas kurz und unpräzise ist und sicherlich auch noch eine Reihe von (Tipp)Fehlern enthält. Daher bin ich für jeden Hinweis darauf (an heun@bio.informatik.uni-muenchen.de) dankbar.

An dieser Stelle möchte ich insbesondere Sabine Spreer danken, die an der Erstellung dieses Skriptes in LaTeX2e maßgeblich beteiligt war.

München, im September 2003 Volker Heun

# Inhaltsverzeichnis

**1 Optimal Scoring Subsequences** **1**

1.1 Maximal Scoring Subsequence . . . . . . . . . . . . . . . . . . . . . . . . 1

1.1.1 Problemstellung . . . . . . . . . . . . . . . . . . . . . . . . . 1

1.1.2 Biologische Anwendungen . . . . . . . . . . . . . . . . . . . . 2

1.1.3 Naive Lösung . . . . . . . . . . . . . . . . . . . . . . . . . . 3

1.1.4 Lösen durch dynamische Programmierung . . . . . . . . . . . 4

1.1.5 Divide-and-Conquer-Ansatz . . . . . . . . . . . . . . . . . . . 4

1.1.6 Clevere Lösung . . . . . . . . . . . . . . . . . . . . . . . . . 5

1.1.7 Zusammenfassung . . . . . . . . . . . . . . . . . . . . . . . . 6

1.2 All Maximal Scoring Subsequences . . . . . . . . . . . . . . . . . . . 7

1.2.1 Problemstellung . . . . . . . . . . . . . . . . . . . . . . . . . 7

1.2.2 Elementare Eigenschaften der strukturellen Definition . . . . . 9

1.2.3 Ein Algorithmus zur Lösung . . . . . . . . . . . . . . . . . . . 14

1.2.4 Zeitkomplexität . . . . . . . . . . . . . . . . . . . . . . . . . 17

1.3 Maximal Scoring Subsequences mit Längenbeschränkung . . . . . . . 18

1.3.1 Problemstellung . . . . . . . . . . . . . . . . . . . . . . . . . 18

1.3.2 Lösung mittels Dynamischer Programmierung . . . . . . . . . 19

1.4 Maximal Scoring Subsequence mit Schranke . . . . . . . . . . . . . . 20

1.4.1 Problemstellung . . . . . . . . . . . . . . . . . . . . . . . . . 20

1.4.2 Links-Negativität . . . . . . . . . . . . . . . . . . . . . . . . 21

1.4.3 Algorithmus zur Lösung des MSS-UB-Problems . . . . . . . . 22

1.5 Maximum Average Scoring Subsequence . . . . . . . . . . . . . . . . 24

1.5.1 Problemstellung . . . . . . . . . . . . . . . . . . . . . . . . . 24

1.5.2 Rechtsschiefe Folgen und fallend rechtsschiefe Partitionen . . . 24
1.5.3 Algorithmus zur Konstruktion rechtsschiefer Partitionen . . . 27
1.5.4 Ein Algorithmus für MASS . . . . . . . . . . . . . . . . . . . 30

**2 Phylogenetische Bäume 35**

2.1 Einleitung . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
2.1.1 Distanzbasierte Verfahren . . . . . . . . . . . . . . . . . . . 36
2.1.2 Charakterbasierte Methoden . . . . . . . . . . . . . . . . . . 37
2.2 Ultrametriken und ultrametrische Bäume . . . . . . . . . . . . . . 38
2.2.1 Metriken und Ultrametriken . . . . . . . . . . . . . . . . . . 39
2.2.2 Ultrametrische Bäume . . . . . . . . . . . . . . . . . . . . . 41
2.2.3 Charakterisierung ultrametrischer Bäume . . . . . . . . . . . . 44
2.2.4 Konstruktion ultrametrischer Bäume . . . . . . . . . . . . . . 48
2.3 Additive Distanzen und Bäume . . . . . . . . . . . . . . . . . . . 51
2.3.1 Additive Bäume . . . . . . . . . . . . . . . . . . . . . . . . . 51
2.3.2 Charakterisierung additiver Bäume . . . . . . . . . . . . . . . 53
2.3.3 Algorithmus zur Erkennung additiver Matrizen . . . . . . . . 60
2.3.4 4-Punkte-Bedingung . . . . . . . . . . . . . . . . . . . . . . . 61
2.3.5 Charakterisierung kompakter additiver Bäume . . . . . . . . . 64
2.3.6 Konstruktion kompakter additiver Bäume . . . . . . . . . . . 66
2.4 Exkurs: Priority Queues & Fibonacci-Heaps . . . . . . . . . . . . 68
2.4.1 Priority Queues . . . . . . . . . . . . . . . . . . . . . . . . . 68
2.4.2 Realisierung mit Fibonacci-Heaps . . . . . . . . . . . . . . . . 69
2.4.3 Implementierung . . . . . . . . . . . . . . . . . . . . . . . . . 69
2.4.4 Worst-Case Analyse . . . . . . . . . . . . . . . . . . . . . . . 72
2.4.5 Amortisierte Kosten bei Fibonacci-Heaps . . . . . . . . . . . . 74
2.5 Sandwich Probleme . . . . . . . . . . . . . . . . . . . . . . . . . 78

2.5.1 Fehlertolerante Modellierungen . . . . . . . . . . . . . . . . . . 78
2.5.2 Eine einfache Lösung . . . . . . . . . . . . . . . . . . . . . . . 79
2.5.3 Charakterisierung einer effizienteren Lösung . . . . . . . . . . 86
2.5.4 Algorithmus für das ultrametrische Sandwich-Problem . . . . 94
2.5.5 Approximationsprobleme . . . . . . . . . . . . . . . . . . . . . 108

**3 Kombinatorische Proteinfaltung 109**

3.1 Inverse Proteinfaltung . . . . . . . . . . . . . . . . . . . . . . . . . 109
3.1.1 Grand Canonical Model . . . . . . . . . . . . . . . . . . . . . 109
3.1.2 Schnitte in Netzwerken . . . . . . . . . . . . . . . . . . . . . . 111
3.1.3 Abgeschlossene Mengen und minimale Schnitte . . . . . . . . 112
3.2 Maximale Flüsse und minimale Schnitte . . . . . . . . . . . . . . . 116
3.2.1 Flüsse in Netzwerken . . . . . . . . . . . . . . . . . . . . . . . 116
3.2.2 Residuen-Netzwerke und augmentierende Pfade . . . . . . . . 117
3.2.3 Max-Flow-Min-Cut-Theorem . . . . . . . . . . . . . . . . . . 118
3.2.4 Algorithmus von Ford und Fulkerson . . . . . . . . . . . . . . 120
3.2.5 Algorithmus von Edmonds und Karp . . . . . . . . . . . . . . 122
3.2.6 Der Algorithmus von Dinic . . . . . . . . . . . . . . . . . . . . 124
3.3 Erweiterte Modelle der IPF . . . . . . . . . . . . . . . . . . . . . . 124
3.3.1 Erweiterung auf allgemeine Hydrophobizitäten . . . . . . . . . 124
3.3.2 Energie-Landschaften im Grand Canonical Modell . . . . . . . 125

**A Literaturhinweise 131**

A.1 Lehrbücher zur Vorlesung . . . . . . . . . . . . . . . . . . . . . . . 131
A.2 Skripten anderer Universitäten . . . . . . . . . . . . . . . . . . . . 131
A.3 Originalarbeiten . . . . . . . . . . . . . . . . . . . . . . . . . . . . 132

**B Index 135**

# Optimal Scoring Subsequences 1

## 1.1 Maximal Scoring Subsequence

Ziel dieses Abschnittes ist es, (möglichst effiziente) Algorithmen für das Maximal Scoring Subsequence Problem vorzustellen. Dabei werden wir noch einmal kurz die wichtigsten Paradigmen zum Entwurf von Algorithmen vorstellen.

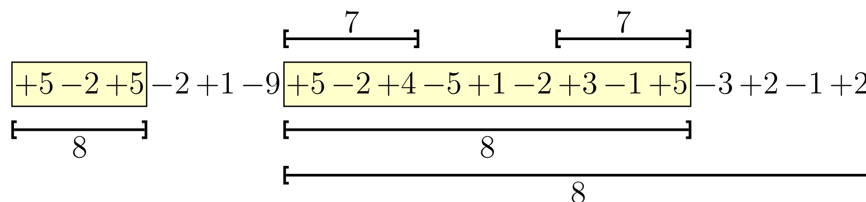
### 1.1.1 Problemstellung

Maximal Scoring Subsequence (MSS)

**Eingabe:** Eine Folge $(a_1, \ldots, a_n) \in \mathbb{R}^n$.
**Gesucht:** Eine (zusammenhängende) Teilfolge $(a_i, \ldots, a_j)$, die $\sigma(i, j)$ maximiert, wobei $\sigma(i, j) = \sum_{\ell=i}^{j} a_\ell$.

**Bemerkung:** Mit Teilfolgen sind in diesem Kapitel immer (sofern nicht anders erwähnt) zusammenhängende (d.h. konsekutive) Teilfolgen einer Folge gemeint (also anders als beispielsweise in der Analysis).

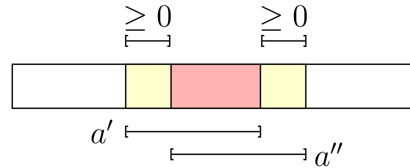
**Beispiel:**




**Bemerkungen:**

- Es sind mehrere Lösungen möglich.
- Ist eine Lösung in der anderen enthalten, so kann man ohne Beschränkung der Allgemeinheit die kürzere wählen.

- Es gibt keine echt überlappenden Lösungen.

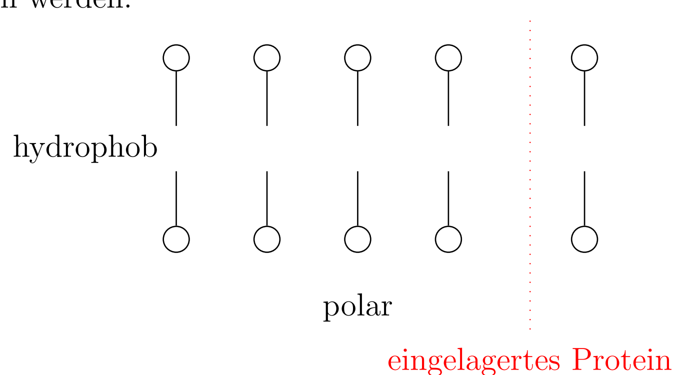



**Annahme:** Es gäbe echt überlappende Lösungen. Die Sequenz gebildet aus der Vereinigung beider Sequenzen (respektive ihrer Indices) müsste dann einen höheren Score haben, da der Score der Endstücke ist $\geq 0$ ist (sonst würde er in den betrachteten Teilfolgen nicht berücksichtigt werden).

### 1.1.2 Biologische Anwendungen

**Transmembranproteine:** Bestimmung der transmembranen Regionen. Eingelagerte Proteine in der Membran sollten einen ähnlichen Aufbau wie die Membran selbst haben, da die Gesamtstruktur stabiler ist. Somit sollten transmembrane Regionen hydrophob sein.

Mit einer geeigneten Gewichtung der Aminosäuren, können solche hydrophoben Regionen mit Hilfe der Lösung eines maximal scoring subsequence Problems gefunden werden.




Für die einzelnen Aminosäuren werden die folgende Werte gewählt:

- hydrophobe Aminosäuren: Score $\in [0 : 3]$;
- hydrophile Aminosäuren: Score $\in [-5 : 0]$.

**Lokalisierung GC-reicher DNS-Abschnitte:** In GC-reichen Regionen der DNS finden sich häufig Gene. Das Auffinden solcher GC-reicher Regionen lässt sich als maximal scoring subsequence Problem beschreiben:

- $C, G \to 1 - p$ für ein $p \in [0 : 1]$;
- $A, T \to -p$.

Zusätzlich können Längenbeschränkungen sinnvoll sein; obere, untere Schranke der Länge für z.B. bekannte Proteine, die man sucht.

**Vergleichende Analyse von Genomen (Mouse/Human):**

- Sequenz-Ähnlichkeiten liegen für Exons bei 85% und für Introns bei 35%. Mit Hilfe eines geeigneten maximal scoring subsequence Problems können somit bei vergleichender Analyse von Genomen Exons und Introns identifiziert werden.
- Variante: Gewichtung des Scores über die Länge zur Vermeidung von "poor Regions" (stark positive Teile, die aber zu kurz sind, werden somit ausgeschlossen).

**Konservierte Regionen:** Bestimmung gut konservierter Regionen in einem mehrfachen Sequenzenalignment durch Gewichtung der Spalten gemäß ihrer Ähnlichkeiten (beispielsweise SP-Maß einer Spalte).

**'Ungapped' local alignment:** Lokales Alignment ohne Lücken (gaps) mit Längenrestriktionen. Man wendet das maximal scoring subsequence Problem auf die Diagonalen der zugehörigen Dot-Matrix an.

### 1.1.3 Naive Lösung

**Idee:** Bestimme $\sigma(i,j) = \sum_{\ell=i}^{j} a_\ell$ für alle $i \leq j \in [1 : n]$.

**Rechenzeit:** Kosten pro Tabelleneintrag $\Theta(j - i + 1)$:

$$\sum_{i=1}^{n} \sum_{j=i}^{n} \Theta(j-i+1) = \Theta\left(\sum_{i=1}^{n} (n-i+1)^2\right) = \Theta\left(\sum_{i=1}^{n} i^2\right) = \Theta(n^3).$$

**Alternative Begründung:** In der Tabelle mit $n^2$ Einträgen benötigt jeder Eintrag maximal $n$ Operationen. Hierbei erhalten wir jedoch nur eine obere Schranke und nicht die korrespondierende untere Schranke für die Laufzeit.

### 1.1.4 Lösen durch dynamische Programmierung

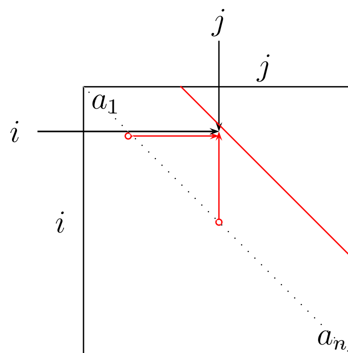
**Rekursionsgleichung:** Es gilt folgende Rekursionsgleichung:

$$\sigma(i,j) = \begin{cases} a_i & \text{für } i = j \\ \sigma(i,k) + \sigma(k+1,j) & \text{für ein } k \in [i:j-1] \end{cases}$$

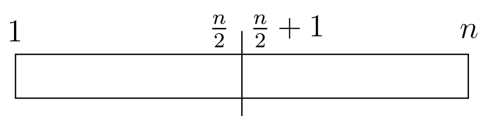
Direktes Lösen dieser Rekursion ist zu aufwendig, da exponentiell viele Aufrufe erfolgen!

In solchen Fällen wird die *Dynamische Programmierung* angewendet.

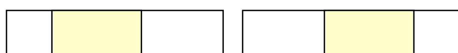
Die Tabellengröße ist $O(n^2)$. Jeder Eintrag kann mit der Rekursionsgleichung in Zeit $O(1)$ berechnet werden. Dabei wird die Tabelle diagonal von der Mitte nach rechts oben aufgefüllt.



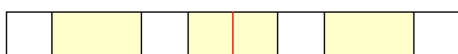

### 1.1.5 Divide-and-Conquer-Ansatz




Divide-Schritt: in der Mitte aufteilen

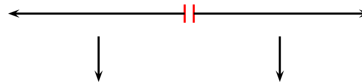


Rekursion auf den beiden Hälften: man erhält jeweils eine optimale Teilfolge



Conquer-Schritt: entscheide, welche Teilfolge den höheren Score besitzt

Man kann dabei aber auch die optimale Teilfolge in der Mitte zerschneiden. Daher muss man zusätzlich von der Mitte aus testen, wie von dort nach rechts bzw. links eine optimale Teilfolge aussieht.



$\max\{\sigma(i, \frac{n}{2}) \mid i \in [1 : \frac{n}{2} + 1]\}$ $\max\{\sigma(\frac{n}{2} + 1, j) \mid j \in [\frac{n}{2} : n]\}$

Dazu bestimmen wir jeweils das Optimum der Hälften, d.h

$$\max\left\{\sigma\left(i, \frac{n}{2}\right) \mid i \in \left[1 : \frac{n}{2} + 1\right]\right\} \qquad \text{und} \qquad \max\left\{\sigma\left(\frac{n}{2} + 1, j\right) \mid j \in \left[\frac{n}{2} : n\right]\right\}.$$

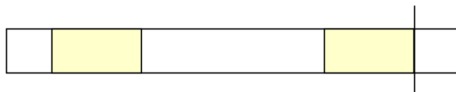
Man überlegt sich leicht, dass die optimale Teilfolge, die die Positionen $n/2$ und $n/2 + 1$ überdeckt, aus der Konkatenation der beiden berechneten optimalen Teilfolgen in den jeweiligen Hälften bestehen muss.

**Laufzeit:** Für die Laufzeit erhalten wir sofort die folgende Rekursionsgleichung, die identisch zur Laufzeitanalyse von Mergesort ist, da das Bestimmen einer optimalen Teilfolge über die Mitte hinweg in Zeit $O(n)$ (wie oben gesehen) geschehen kann:

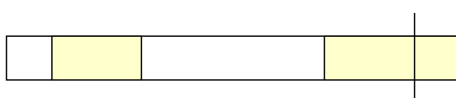
$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + O(n) = O(n \log(n)).$$

### 1.1.6 Clevere Lösung

Wenn wir wie beim Divide-and-Conquer-Ansatz das Problem nicht in der Mitte aufteilen, sondern am rechten Rand, so können wir (ganz im Gegensatz zum Problem des Sortierens) eine effizientere Lösung finden.



Nur das letzte Feld absplitten.



Am Rand gleich die optimale Teilfolge mitbestimmen, z.B. durch den Versuch die aktuelle Randfolge zu verlängern.

Das einziges Problem besteht darin, dass die Folge am rechten Ende einen negativen Score erhält. Dann wähle die leere Folge mit dem Score 0.

CLEVERE LÖSUNG

```
{
    max = 0;   ℓ = 1;   r = 0;
    rmax = 0;   rstart = 1;
    for (i = 1, i ≤ n, i++)
    {
        if (rmax + a_i > 0)
            rmax := rmax + a_i;
        else
        {
            rmax = 0;
            rstart = i + 1;
        }
        if (rmax > max)
        {
            max = rmax;
            ℓ = rstart;
            r = i
        }
    }
}
```

Abbildung 1.1: Algorithmus: Die clevere Lösung

Laufzeit: Offensichtlich erhalten wir eine Laufzeit von $O(n)$.

**Theorem 1.1** *Eine maximal scoring subsequence einer gegebenen reellen Folge lässt sich in Linearzeit mit konstantem zusätzlichen Platzbedarf bestimmen.*

### 1.1.7 Zusammenfassung

In der folgenden Tabelle sind alle Resultate der vorgestellten Algorithmen noch einmal zusammengefasst.

| Algorithmus | Zeit | Platz | Bemerkung |
|---|---|---|---|
| naive Programmierung | $O(n^3)$ | $O(n^2)$ | Tabelle füllen |
| DP | $O(n^2)$ | $O(n^2)$ | geschickter füllen |
| Divide and Conquer | $O(n \log(n))$ | $O(n)$ | die Eingabelänge ist $n$ |
| Clevere Lösung | $O(n)$ | $n + O(1)$ | die Eingabelänge ist $n$ |

In der folgenden Tabelle sind noch die Längen von Folgen angegeben, die in einer Sekunde bzw. einer Minute auf einem gewöhnlichen Rechner verarbeitet werden können.

| Seq-len in | 1sec. | | 1min. | Space |
|---|---|---|---|---|
| Naive | 800 | $\xrightarrow{\times 4}$ | 3.200 | $O(n^2)$ |
| Dyn.Prog. | 4.200 | $\xrightarrow{\times 8}$ | 32.000 | $O(n^2)$ |
| D&C | 1.500.000 | $\xrightarrow{\times 50}$ | 75.000.000 | $O(n)$ |
| Clever | 20.000.000 | $\xrightarrow{\times 60}$ | 1.200.000.000 | $n + O(1)$ |

**8. April**

## 1.2 All Maximal Scoring Subsequences

Nun wollen wir uns mit der Frage beschäftigen, wenn wir nicht nur eine beste, sondern alle besten bzw. alle möglichen Teilfolgen mit positive Score und zwar nach absteigenden Score erhalten wollen. Zuerst einmal müssen wir uns über die Fragestellung klar werden, d.h. was ist überhaupt die gesuchte Lösung.

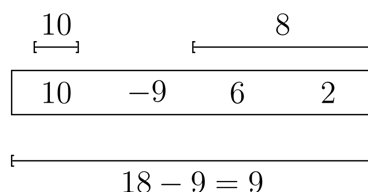
### 1.2.1 Problemstellung

All Maximal Scoring Subsequences (AMSS)

**Eingabe:** Eine Folge $(a_1, \ldots, a_n) \in \mathbb{R}^n$.
**Gesucht:** Alle disjunkten Teilfolgen, die ihren Score maximieren.

Zunächst einmal muss man sich überlegen, was es heißen soll, dass *alle disjunkten Teilfolgen ihren Score maximieren.*

**Beispiel:**



Die lange Folge ist nicht die Lösung, da wir keine überlappenden Folgen haben wollen.

Wir geben zwei mögliche Definition an, wie man alle maximalen Teilfolgen einer Folge definieren kann. Im Folgenden werden wir zeigen, dass die beiden Definitionen äquivalent sind. A priori ist dies überhaupt nicht klar.

**Definition 1.2 (Strukturelle Definition)** *Eine Teilfolge $a' = (a_i, \ldots, a_j)$ einer Folge $a \in \mathbb{R}^n$ heißt* maximal scoring*, wenn die beiden folgenden Eigenschaften erfüllt sind:*

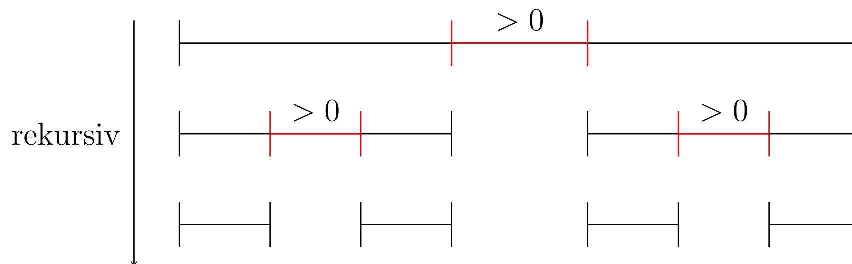
*E1 alle echten Teilfolgen von $a'$ haben einen kleineren Score;*

*E2 keine Oberfolge von $a'$ in $a$ erfüllt E1.*

Neben dieser strukturellen Definition kann man auch noch eine eher algorithmisch angelehnte Definition angeben.

**Definition 1.3 (Prozedurale Definition)** *Sei $a = (a_1, \ldots, a_n) \in \mathbb{R}^n$ eine reelle Folge. Eine kürzeste Teilfolge $(a_i, \ldots, a_j)$ der Folge $a$ mit maximalem Score heißt* maximal scoring*. Teilfolgen aus $(a_1, \ldots, a_{i-1})$ bzw. $(a_{j+1}, \ldots, a_n)$, die für diese maximal scoring sind, sind auch für $a$* maximal scoring.

Aus der prozeduralen Definition kann sofort ein Algorithmus zur Bestimmung aller MSS abgeleitet werden, der in der folgenden Skizze veranschaulicht ist:




Die Laufzeit dieses Algorithmus erfüllt folgende Rekursionsgleichung, da das Auffinden einer maximal scoring subsequence, wie wir im letzten Abschnitt gesehen haben, in Zeit $O(n)$ durchführbar ist:

$$T(n) = O(n) + T(n_1) + T(n_2) \quad \text{mit} \quad n_1 + n_2 < n.$$

Ähnlich zu Quicksort ergibt sich folgende Analyse:

**worst case:** $O(n^2)$

**average case:** $O(n \log(n))$ (mit einer geeigneten Wahrscheinlichkeitsverteilung, die nicht unbedingt realistisch sein muss!)

### 1.2.2 Elementare Eigenschaften der strukturellen Definition

**Lemma 1.4** *Sei $a = (a_1, \ldots, a_n) \in \mathbb{R}^n$. Für jede Teilfolge $a' = (a_i, \ldots, a_j)$ von $a$ sind äquivalent:*

*1) $a'$ erfüllt E1.*

*2) Es gilt, dass das Minimum aller Präfixe von $a$ in $a'$*

$$\sigma(1, i-1) = \min\{\sigma(1,k) \mid k \in [i-1, j]\}$$

*und das Maximum aller Präfixe von $a$ in $a'$*

$$\sigma(1, j) = \max\{\sigma(1,k) \mid k \in [i-1, j]\}$$

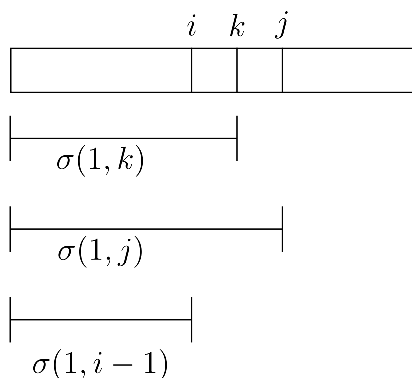
*eindeutig sind!*

*3) $\forall k \in [i : j] : \sigma(i,k) > 0 \wedge \sigma(k,j) > 0$.*

**Beweis:** (1. $\Rightarrow$ 2. $\Rightarrow$ 3. $\Rightarrow$ 1.)

**1. $\Rightarrow$ 2. (durch Widerspruch)**

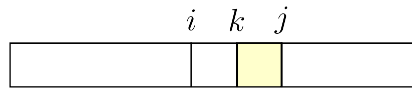
1.Fall: Sei $k \in [i-1 : j]$ mit $\sigma(1,k) \geq \sigma(1,j)$

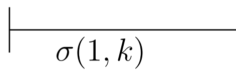


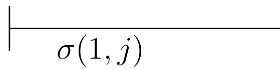

$$\begin{aligned} \sigma(i,k) &= \sigma(1,k) - \sigma(1,i-1) \\ &\geq \sigma(1,j) - \sigma(1,i-1) \\ &= \sigma(i,j) \end{aligned}$$

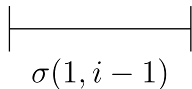
Daraus ergibt sich ein Widerspruch zur Annahme der Eigenschaft E1), da die echte Teilfolge $(a_i, \ldots, a_k)$ von $(a_i, \ldots, a_j)$ einen größeren Score besitzt.

2.Fall: Sei jetzt $k \in [i-1:j]$ mit $\sigma(1,k) \leq \sigma(1,i-1)$.




$$\begin{aligned}\sigma(k+1,j) &= \sigma(1,j)-\sigma(1,k)\\ &\geq \sigma(1,j)-\sigma(1,i-1)\\ &= \sigma(i,j)\end{aligned}$$

Daraus ergibt sich ein Widerspruch zur Annahme der Eigenschaft E1), da die echte Teilfolge $(a_{k+1},\ldots,a_j)$ von $(a_i,\ldots,a_j)$ einen größeren Score besitzt.

**2. ⇒ 3.**

Da das Minimum eindeutig ist, folgt für alle $k \in [i:j]$:

$$\sigma(i,k) = \sigma(1,k) - \underbrace{\sigma(1,i-1)}_{<\sigma(1,k)} > 0.$$

Da das Maximum eindeutig ist, folgt für alle $k \in [i:j]$:

$$\sigma(k,j) = \underbrace{\sigma(1,j)}_{>\sigma(1,k-1)} - \sigma(1,k-1) > 0.$$

**3. ⇒ 1.**

Sei $a' = (a_k,\ldots,a_\ell)$ mit $i \leq k \leq \ell \leq j$ sowie $i \neq k$ oder $\ell \neq j$. Dann gilt:

$$\sigma(k,\ell) = \sigma(i,j) - \underbrace{\sigma(i,k-1)}_{\geq 0} - \underbrace{\sigma(\ell+1,j)}_{\geq 0}$$

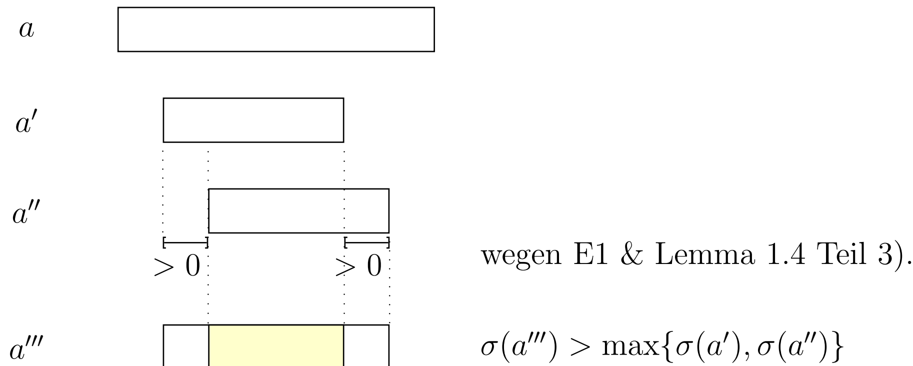
Hinweis: $\sigma(i,k-1)$ und $\sigma(\ell+1,j)$ sind jeweils $\geq 0$, eines davon muss $> 0$ sein, da sonst $a' = a''$ gelten würde.

Damit ist gezeigt, dass E1) gilt. ■

**Lemma 1.5** *Sei $a = (a_1,\ldots,a_n) \in \mathbb{R}^n$. Die maximal scoring subsequences von $a$ sind paarweise disjunkt.*

**Beweis: (durch Widerspruch)**

Seien $a'$ und $a''$ zwei maximal scoring subsequences, die sich überlappen.

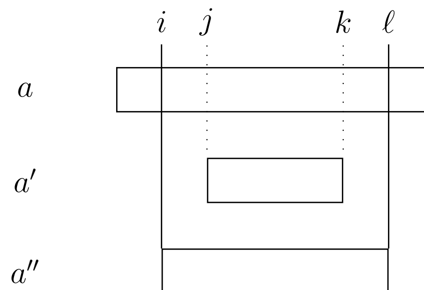



Dies ergibt den gewünschten Widerspruch zur Eigenschaft E2 von $a'$ und $a''$, da $a'''$ E1 erfüllt. ■

**Lemma 1.6** *Sei $a = (a_1, \ldots, a_n) \in \mathbb{R}^n$. Jede Teilfolge $a' = (a_k, \ldots, a_\ell)$, die E1 erfüllt, ist Teilfolge einer maximal scoring subsequence.*

**Beweis: (durch Widerspruch)**

Sei $a' = (a_k, \ldots, a_\ell)$ ein Gegenbeispiel (erfüllt also auch E1) mit maximaler Länge, d.h. $\ell - k$ ist unter allen Gegenbeispielen maximal.




$a'$ erfüllt E1, aber nicht E2 (sonst wäre $a'$ kein Gegenbeispiel). Somit existiert eine echte Oberfolge $a''$ von $a'$, die E1 erfüllt. Würde die Folge $a''$ auch E2 erfüllen, dann wäre $a''$ eine maximal scoring subsequence, die auch eine Oberfolge von $a'$ ist, d.h. $a'$ wäre kein Gegenbeispiel.

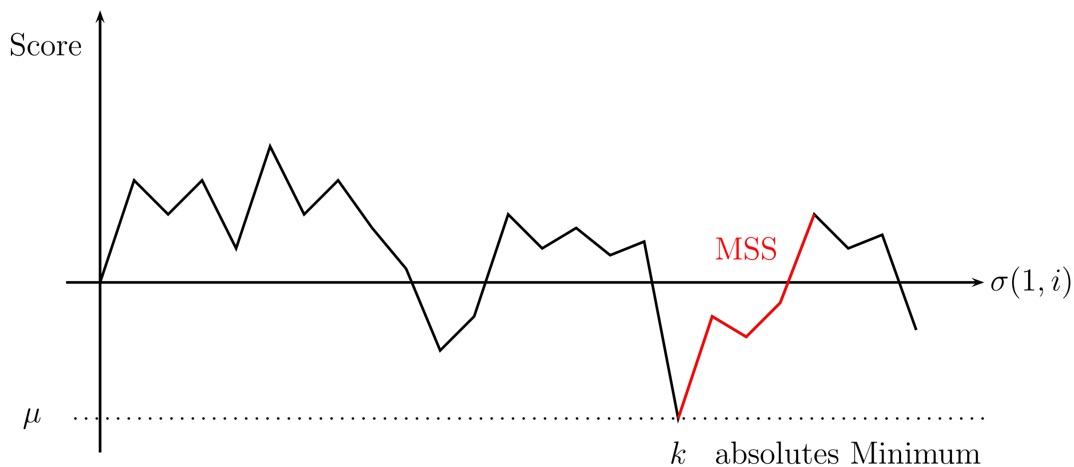
$a''$ ist somit ein längeres Gegenbeispiel als $a'$, da $\ell - i > k - j$. Dies führt zu einem Widerspruch zur Annahme. ■

**Korollar 1.7** *Jedes positive Element ist in einer maximal scoring subsequence enthalten.*

**Korollar 1.8** *Innerhalb jeder Teilfolge, die mit keiner maximal scoring subsequence überlappt, sind die aufaddierten Scores monoton fallend.*

**Lemma 1.9** *Sei $a = (a_1, \ldots, a_n) \in \mathbb{R}^n$. Betrachte $\mu = \min\{\sigma(1,k) \mid k \in [1:n]\}$ (bzw. $\mu = \max\{\sigma(1,k) \mid k \in [1:n]\}$). Sei $k$ der größte (bzw. kleinste) Wert mit $\sigma(1,k) = \mu$. Dann ist $k$ am linken (bzw. rechten) Ende einer maximal scoring subsequence oder am Ende von $a$.*

**Beweis:** Betrachte $\sigma(1,i)$ als Funktion von $i \in [0:n]$:




Allgemein gilt $\sigma(i,j) = \sigma(1,j) - \sigma(1,i-1)$.

Fall 1: $k$ ist in keiner maximal scoring subsequence. Nach Korollar 1.8 ist $k$ am Ende einer nicht-maximal scoring subsequences, also am Ende von $a$ oder Anfang einer maximal scoring subsequence.

Fall 2: $k$ ist in einer maximal scoring subsequence. Nach Lemma 1.4 Teil 3) ist $k$ am linken Ende einer maximal scoring subsequence (da Präfixe echt positiven Score haben müssen), also am Anfang einer maximal scoring subsequence. ■

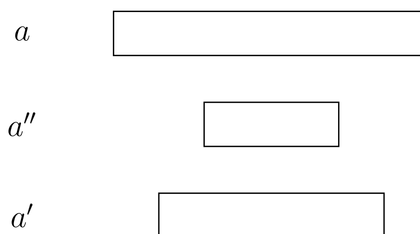
**10.April**

Im Folgenden werden wir die folgende algorithmische Idee verfolgen. Sei $a'$ eine Teilfolge von $a$. Wir versuchen $a'$ auf eine maximal scoring subsequence zu verlängern.

**Notation:** Eine Teilfolge $a'$ von $a$ heißt $a$-MSS, wenn sie eine maximal scoring subsequence von $a$ ist.

**Lemma 1.10** *Sei $a = (a_1, \ldots, a_n) \in \mathbb{R}^n$. Sei $a''$ eine Teilfolge von $a'$ und $a'$ eine Teilfolge von $a$. Ist $a''$ eine $a$-MSS, dann ist $a''$ auch $a'$-MSS.*

**Beweis:** Sei $a''$ eine $a$-MSS und sei sowohl $a'$ eine Teilfolge von $a$ als auch eine Oberfolge von $a''$.



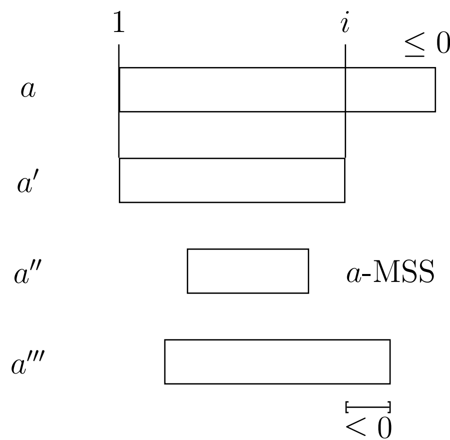

Da $a''$ eine $a$-MSS ist, erfüllt $a''$ die Eigenschaften E1 und E2 bezüglich $a$. Somit erfüllt die Folge $a''$ diese Eigenschaften auch bezüglich der Oberfolge $a'$. ■

**Lemma 1.11** *Sei $a = (a_1, \ldots, a_n) \in \mathbb{R}^n$. Sei $a' = (a_1, \ldots, a_i)$ und $a_k \leq 0$ für alle $k \in [i+1 : n]$. Alle maximal scoring subsequences von $a'$ sind auch maximal scoring subsequences von $a$ und umgekehrt.*

**Beweis:**

'$\Longleftarrow$' Dies ist die Aussage von Lemma 1.10.

'$\Longrightarrow$' Sei $a''$ eine $a'$-MSS.



Damit $a''$ keine $a$-MSS sein kann, muss es eine Oberfolge von $a'$ geben, die E1 erfüllt. Da $a''$ eine $a'$-MSS ist, muss diese Oberfolge $a'''$ in den hinteren Teil (ab Position $i+1$) hineinragen. Eine solche Folge $a'''$ kann nach Lemma 1.4 nicht E1 erfüllen. Dies führt zu dem gewünschten Widerspruch. ■

**Lemma 1.12** *Sei $a = (a_1, \ldots, a_n) \in \mathbb{R}^n$. Sei $a' = (a_i, \ldots, a_j)$ eine a-MSS und sei $a^L = (a_1, \ldots, a_{i-1})$ und $a^R = (a_{j+1}, \ldots, a_n)$. Dann ist eine Teilfolge $a''$ von $a^L$ (bzw. $a^R$) genau dann eine $a^L$-MSS (bzw. $a^R$-MSS), wenn $a''$ eine a-MSS ist.*

**Beweis:**

'$\Longleftarrow$' Dies folgt aus Lemma 1.10. Die maximal scoring subsequences sind disjunkt.

'$\Longrightarrow$' Sei $a''$ eine $a^L$-MSS, aber nicht eine $a$-MSS. Somit erfüllt $a''$ die Eigenschaft E1. Mit Lemma 1.6 folgt, dass $a''$ in $\overline{a}$ enthalten ist, die eine $a$-MSS ist. Weiter gilt: $\overline{a} \not\subseteq a^L$, da keine Teilfolge von $a^L$ sein kann, sonst wäre $a''$ keine $a^L$-MSS. Somit müssen sich $\overline{a}$ und $a'$ überlappen. Dies ist ein Widerspruch zu Lemma 1.5. ■

**Theorem 1.13** *Die strukturelle und prozedurale Definition von maximal scoring subsequences stimmen überein.*

**Beweisidee:** Sei $MSS(a)$ die Menge aller Teilfolgen von $a$, die maximal scoring subsequences sind (gemäß der strukturellen Definition, also die die Eigenschaften E1 und E2 erfüllen).

Sei $a' = (a_i, \ldots, a_j) \in MSS(a)$ mit maximalem Score. Nach Lemma 1.12 gilt:

$$MSS(a) = \{a'\} \cup MSS(a_1, \ldots, a_{i-1}) \cup MSS(a_{j+1}, \ldots, a_n).$$

Den vollständige Beweis lässt sich jetzt formal mittels Induktion führen. ■

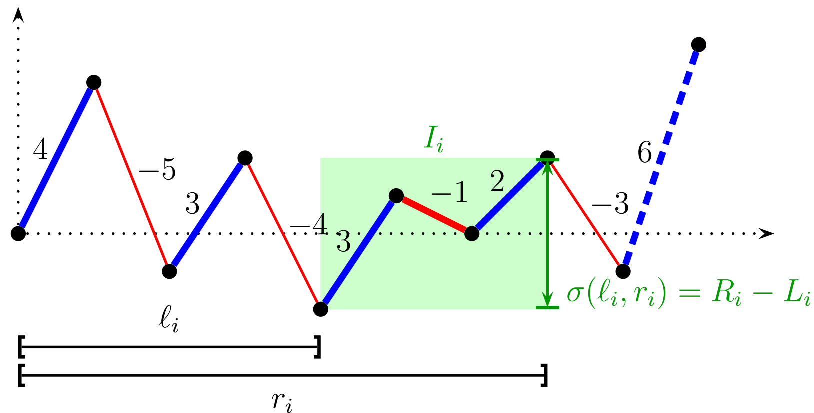
### 1.2.3 Ein Algorithmus zur Lösung

Wir beschreiben jetzt einen Algorithmus zur Ermittlung aller maximal scoring subsequences.

- Die Eingabe ist die Folge $a = (a_1, \ldots, a_n) \in \mathbb{R}^n$.
- Die Elemente werden von links nach rechts verarbeitet. Dabei merken wir uns disjunkte Teilfolgen $I_1, \ldots, I_k$ eines Präfixes von $a$ ($I_1, \ldots, I_k$ werden dabei solche Teilfolgen nach Möglichkeit maximaler Länge des bereits abgearbeiteten Präfixes von $a$ sein, die die Eigenschaft E1 erfüllen).

- $I_i = (a_{\ell_i}, \ldots, a_{r_i})$, d.h. $I_i \hat{=} (\ell_i, r_i)$.
- Setze $L_i = \sigma(1, \ell_i - 1)$ und $R_i = \sigma(1, r_i)$.
- Damit gilt $\sigma(I_i) := \sigma(\ell_i, r_i) = R_i - L_i$.

Das können wir uns wie folgt veranschaulichen:

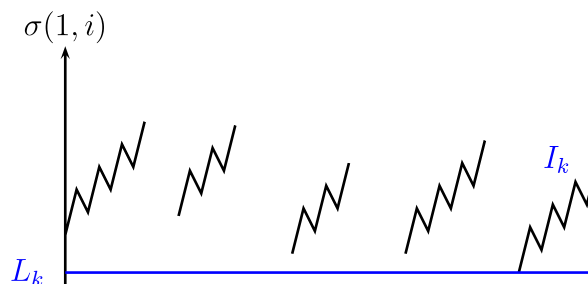



Wir gehen wie folgt vor: Bearbeite die Folge von links nach rechts; Sammle eine Liste von Teilfolgen, die E1 bereits erfüllen.

Sei $a_m$ das aktuell betrachtete Element der Folge $a$:

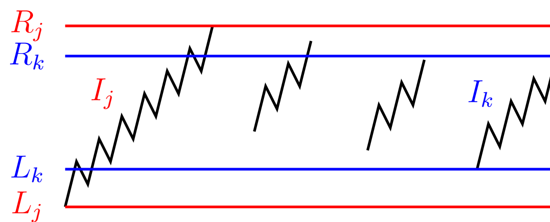
- Ist $a_m \leq 0$, betrachte die nächste Position $m + 1$, da keine maximal scoring subsequence an Position $m$ beginnen oder enden kann.
- Ist $a_m > 0$, erzeuge neue Liste $I_k = (m, m)$ (einelementige Liste), bestimme $L_k$ und $R_k$. (Die Folge $(a_m)$ erfüllt E1, hat aber nicht notwendigerweise maximale Länge.)
- Für ein neues $I_k$ wiederhole das Folgende: Durchsuche $I_{k-1}, \ldots, I_1$ (von rechts nach links!) bis ein maximales $j$ gefunden wird, so dass $L_j < L_k$.

Fall 1: Es existiert gar kein solches $j$:



Nach Lemma 1.9 ist $I_k$ der Anfang einer maximal scoring subsequence in der Teilfolge $a' = (a_1, \ldots, a_m)$. Nach Lemma 1.12 sind $I_1, \ldots, I_{k-1}$ dann auch maximal scoring subsequences in $a'$. Nach Lemma 1.6 ist $I_k$ auch Teilfolge einer $a$-MSS. Die Teilfolge $I_k$ ist sogar Anfang einer $a$-MSS, denn sonst hätten einige Präfixe von einen nichtpositiven Score, was ein Widerspruch zu Lemma 1.4 ist. Somit sind $I_1, \ldots, I_{k-1}$ nach Lemma 1.12 auch jeweils eine $a$-MSS. Daher geben wir jetzt die Teilfolgen $I_1, \ldots, I_{k-1}$ als maximal scoring subsequences aus und setzen $I_1 := I_k$ sowie $k = 1$.

Fall 2: Sei $j$ maximal mit $L_j < L_k$ und $R_j \geq R_k$:



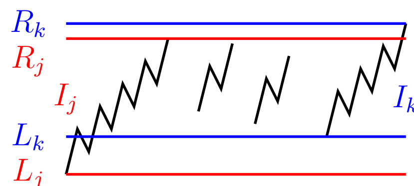

Behauptung.: $I_j$ ist eine $a'$-MSS mit $a' = (a_1, \ldots, a_m)$

E1) Dies gilt offensichtlich nach Konstruktion.

E2) Angenommen, es gäbe eine Oberfolge $a''$ von $I_j$, die E1 erfüllt. Endet $a''$ hinter $I_j$, dann hätte die Teilfolge von $I_j$ von $a''$ einen größeren Score, was ein Widerspruch zur Eigenschaft E1 von $a''$ ist. Endet $a''$ an derselben Position, wie $I_j$, dann hätten wir die Folge $I_j$ schon zur Folge $a''$ verschmolzen, wie dies im folgenden, dritten Fall erläutert wird.

In diesem Fall ist algorithmisch also nichts zu tun.

Fall 3: Sei $j$ maximal mit $L_j < L_k$ und $R_j < R_k$:




Offensichtlich erfüllt die Teilfolge $(a_{\ell_j}, \ldots, a_{r_k})$ die Eigenschaft E1, aber nicht notwendigerweise E2. Somit muss ein neues Intervall generiert und angefügt werden. Alle 3 Fälle sind jetzt wieder neu zu betrachten. Im Allgemeinen bedeutet dies: Bilde ein neues $I_k$ mit $I_j := (\ell_j, r_k)$, setze $k := j$ und wiederhole die Prozedur für $I_k = I_j$.

**15.April**

### 1.2.4 Zeitkomplexität

Wir analysieren jetzt die Zeitkomplexität des soeben vorgestellen Algorithmus:

1.) Durchsuchen von $I_{k-1}, \ldots, I_1$, dies wird später analysiert.

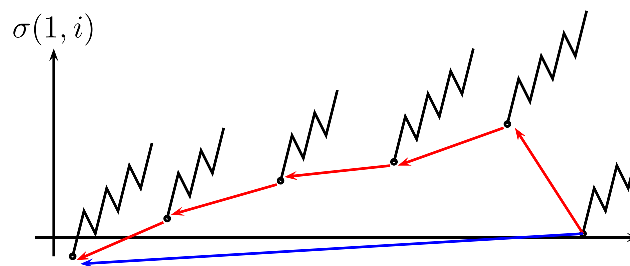
2.) (entspricht dem Gesamtaufwand des ersten Falles) $\nexists j \Rightarrow$ Ausgabe $I_1, \ldots, I_{k-1}$: für jedes $I_\ell$ je $O(1)$, also insgesamt für alle Ausgaben $O(n)$.

3.) (entspricht dem Gesamtaufwand des zweiten Falles) $L_j < L_k \wedge R_j \geq R_k$: Die Ermittlung des zweiten Falles geschieht in konstanter Zeit; also insgesamt für alle Fälle höchstens $O(n)$.

4.) (entspricht dem Gesamtaufwand des dritten Falles) $L_j < L_k \wedge R_j < R_k$: Verschmelzung von $I_j, \ldots, I_k$; dies sind höchstens $n-1$ Verschmelzungen also insgesamt $O(n)$.

    Dabei stellen wir uns eine Verschmelzung von $\ell$ Intervallen als $\ell - 1$ Verschmelzungen von je zwei Intervallen vor. Da es insgesamt maximal $n$ Verschmelzungen disjunkter Intervalle geben kann, folgt obige Behauptung.

**Durchsuchen der Listen:** Beim Durchsuchen der Listen von Intervallen gehen wir etwas geschickter vor. Wir merken uns für jedes Intervall $I_k$ dabei das Intervall $I_j$, für das $L_j < L_k$ und $j$ maximal ist. Beim nächsten Durchlaufen können wir dann einige Intervalle beim Suchen einfach überspringen. Dies ist in der folgenden Abbildung schematisch dargestellt:




Jetzt müssen wir uns nur noch überlegen, welchen Aufwand alle Durchsuche-Operationen insgesamt haben. Dabei werden die Kosten zum einen auf die Aufrufe (der Durchsucheprozedur) und zum anderen auf die ausgeschlossenen Intervalle verteilt.

Bei einem Durchsuchen der Liste werden beispielsweise $\ell$ Intervalle übersprungen. Dann wurden $\ell+2$ Listen inspiziert, was Kosten in Höhe von $O(\ell+1)$ verursacht (man beachte, dass $\ell = 0$ möglich ist). Die Kosten der Intervalle, die dabei ausgeschlossen werden, werden einfach anteilig auf die ausgeschlossenen Intervalle umgelegt. Somit

erhält jedes ausgeschlossene Intervall Kosten von $O(1)$. Die Kosten für den Rest werden auf den Aufruf eines Durchsuch-Durchlaufs umgelegt, die dann ebenfalls konstant sind.

Somit fallen jeweils Kosten $O(1)$ auf ausgeschlossene Intervalle und Aufrufe an.

**Anzahl Aufrufe:** Es kann maximal so viele Aufrufe geben, wie die Schritte 2, 3, und 4 ausgeführt werden, also $O(n)$.

**Anzahl ausgeschlossener Intervalle:** Zum einen können diese nach Schritt 2 ausgegeben werden, von diesen kann es daher ebenfalls maximal $O(n)$ viele geben.

Wenn diese Intervalle nicht selbst ausgegeben werden, müssen diese irgendwann mit anderen Intervallen verschmolzen worden sein (sie können ja nicht einfach verschwinden). Da, wie wir bereits gesehen haben, maximal $O(n)$ Intervalle verschmolzen werden, können auch hierdurch höchstens $O(n)$ Intervalle ausgeschlossen werden.

Mit unserem kleinen Trick kann also auch das Durchsuchen der Intervall-Listen mit einem Aufwand von insgesamt $O(n)$ bewerkstelligt werden.

**Theorem 1.14** *Das All Maximal Scoring Subsequence Problem für eine gegebene reelle Folge kann in Linearzeit gelöst werden.*

## 1.3 Maximal Scoring Subsequences mit Längenbeschränkung

### 1.3.1 Problemstellung

Es wird zusätzlich eine untere $\underline{\lambda}$ und eine obere Schranke $\overline{\lambda}$ vorgegeben, um die Länge der zu betrachtenden Teilfolgen zu beschränken. Es gilt natürlich $\underline{\lambda} \leq \overline{\lambda} \in \mathbb{N}$.

Mit $Seq(n, k)$ bezeichnen wir die Menge aller 0-1-Zeichenreihen mit genau $k$ konsekutiven 1-runs, deren Länge durch $\underline{\lambda}$ nach unten und mit $\overline{\lambda}$ nach oben beschränkt ist.

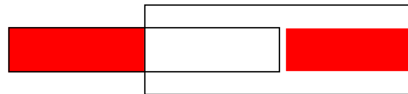
**Notation:**

$$Seq(n, k) \ := \ \{0^* s_1 0^+ s_2 0^+ \cdots 0^+ s_k 0^* \mid s_i \in 1^*, \underline{\lambda} \leq |s_i| \leq \overline{\lambda}\} \cap \{0,1\}^n \ \subseteq \ \{0,1\}^n.$$

All Maximal Scoring Subsequences with Bounds (AMSSB)

**Eingabe:** Eine Folge $(a_1, ..., a_n) \in \mathbb{R}^n$, $\underline{\lambda} \leq \overline{\lambda} \in \mathbb{N}$.
**Gesucht:** Eine Sequenz $s \in \bigcup_{k=0}^{n} Seq(n,k)$, die $\sum_{i=1}^{n} s_i \cdot a_i$ maximiert.

**Bemerkung:** Durch die Längenbeschränkung können nach unserer alten Definition von allen maximal scoring subsequences die einzelnen Teilfolgen der Lösung überlappen. Damit ist ein Greedy-Ansatz nicht mehr effizient möglich. Aus diesem Grund wurde das Problem anders gefasst, nämlich wie oben. Mit Hilfe der Menge $Seq(n,k)$ werden aus der Gesamtfolge Teilstücke ausgewählt (nämlich 1-runs), die wir als Lösungsteilfolgen zulassen wollen.



### 1.3.2 Lösung mittels Dynamischer Programmierung

Wir definieren wieder eine Tabelle $S$ wie folgt:

$$S(n,k) := \max \left\{ \sum_{j}^{i} s_j \cdot a_j \mid s \in Seq(i,k) \right\}$$

Berechne $S(i,k)$ für alle Werte $i \in [0:n]$, $k \in [0:n]$, wobei $\max \emptyset = \max\{\} = -\infty$ (entspricht dem neutralem Element) gilt.

**Rekursionsgleichung:**

$$\begin{aligned} S(i,0) &= 0 \quad \text{für } i \in [0:n] \\ S(i,k) &= -\infty \quad \text{für } i < k \cdot \underline{\lambda} + (k-1) \\ S(i,k) &= \max \left\{ S(i-1,k), S(i-\lambda-1,k-1) + \sum_{j=i-\lambda+1}^{i} a_i \right\} \quad \text{für } \lambda \in [\underline{\lambda}, \min(\overline{\lambda}, i)] \end{aligned}$$

Die Korrektheit der Rekursionsgleichung ergibt sich aus der Tatsache, dass man sich eine optimale Menge von Teilfolgen anschaut und unterscheidet, ob diese mit einer 0 oder einem 1-run endet:

| | | | 0 |
|---|---|---|---|
| | 0 | 1 —— | 1 |

$i - \lambda - 1$ $\quad \underline{\lambda} \leq \lambda \leq \overline{\lambda}$

**Bemerkung:** Es gilt:

$$\sum_{j=i-\lambda+1}^{i} a_j = \sum_{j=1}^{i} a_j - \sum_{j=1}^{i-\lambda} a_j$$

Somit kann man die einzelnen Summen effizient berechnen, wenn für alle $i$ die folgenden Summen kennt:

$$A(i) := \sum_{j=1}^{i} a_j$$

Laufzeit pro Eintrag: $O(\overline{\lambda} - \underline{\lambda}) = O(n)$.

Somit ist die Gesamtlaufzeit mit $O(n^2(\overline{\lambda} - \underline{\lambda})) \leq O(n^3)$ gegeben.

In der Praxis sind die Schranken allerdings so nah beieinander, so dass die Laufzeit bei $O((\overline{\lambda} - \underline{\lambda})n^2) = O(n^2)$ bleibt.

## 1.4 Maximal Scoring Subsequence mit Schranke

Jetzt wollen wir nur **eine** längenbeschränkte maximal scoring subsequence finden. Man beachte, dass diese kein Teil der Lösung aus dem Problem des vorherigen Abschnittes sein muss! Wir werden zunächst nur eine obere Längenbeschränkung für die gesuchte Folge beachten. Eine Hinzunahme einer unteren Längenbeschränkung ist nicht weiter schwierig und wird in den Übungen behandelt.

### 1.4.1 Problemstellung

MAXIMAL SCORING SUBSEQUENCE WITH UPPER BOUND (MSS-UB)

**Eingabe:** Eine Folge $(a_1, \ldots, a_n) \in \mathbb{R}^n$ und $\lambda \in \mathbb{N}$.
**Gesucht:** Eine (zusammenhängende) Teilfolge $(a_i, \ldots, a_j)$ mit $j - i + 1 \leq \lambda$, die $\sigma(i, j)$ maximiert, wobei $\sigma(i, j) = \sum_{\ell=i}^{j} a_\ell$.

### 1.4.2 Links-Negativität

**Definition 1.15** *Eine Folge $(a_1, \ldots a_n) \in \mathbb{R}^n$ heißt* links-negativ, *wenn $\sum_{\ell=1}^{k} a_\ell \leq 0$ für alle $k \in [1 : n-1]$.*
*Eine Partition $a = A_1 \cdots A_k$ heißt* minimal links-negativ, *wenn für alle $i \in [1 : k]$ $A_i$ links-negativ ist und $\sigma(A_i) > 0$ für alle $i \in [1 : k-1]$ ist.*

**Beispiele**

1.) $(-1, 1, -3, 1, 1, 3)$ ist links-negativ,

2.) $(2, -3, -4, 5, 3, -3)$ ist **nicht** links-negativ.
Die Partition $(2)(-3, -4, 5, 3)(-3)$ ist eine minimal links-negative.

**24. April**

**Lemma 1.16** *Jede Folge $a = (a_1, \ldots, a_n) \in \mathbb{R}^n$ lässt sich eindeutig in eine minimal links-negative Partition zerlegen.*

**Beweis: (durch Induktion)**

**Induktionsanfang ($n = 1$):** trivial.

**Induktionsschritt ($n \to n+1$):** Betrachte $a' = (a_0, a_1, \ldots, a_n)$ und sei dann $a = (a_1, \ldots, a_n)$. Nach Induktionsvoraussetzung sei $a = A_1 \cdots A_k$ die(!) minimal links-negative Partition von $a$.

1. Fall ($a_0 > 0$): $a' = (a_0, A_1, \ldots, A_k)$ ($a_0$ vorne anhängen), d.h. $a'$ ist eine eindeutige minimal links-negative Partition.

2 Fall ($a_0 \leq 0$): Wähle ein minimales $i$ mit

$$a_0 + \sum_{j=1}^{i} \sigma(A_j) > 0.$$

Dann ist $(a_0 A_1 \cdots A_i, A_{i+1}, \ldots, A_k)$ eine minimal links-negative Partition.

Der Beweis der Eindeutigkeit sei dem Leser zur Übung überlassen. ■

**Notation:** Sei $a = (a_1, \ldots, a_n) \in \mathbb{R}^n$. Jedes Suffix $a^{(i)} = (a_i, \ldots, a_n)$ von $a$ besitzt dann eine eindeutige minimal links-negative Partition $a^{(i)} = (A_1^{(i)}, \ldots, A_{k_i}^{(i)})$.

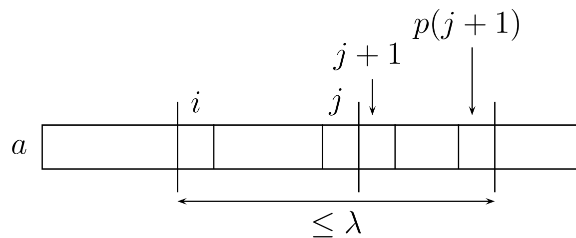
Sei weiter $A_1^{(i)} = (a_i, \ldots, a_{p(i)})$, dann heißt $p(i)$ der *links-negative Zeiger* von $i$.

**Bemerkungen:**

- Ist $a_i > 0$, dann ist $p(i) = i$.
- Ist $a_i \leq 0$, dann ist $p(i) > i$ für $i < n$ und $p(n) = n$.

### 1.4.3 Algorithmus zur Lösung des MSS-UB-Problems

Wenn wir eine betrachtete Teilfolge $(a_i, \ldots, a_j)$ verlängern wollen, müssen wir vom Suffix $(a_{j+1}, \ldots, a_n)$ also mindestens den Teil $(a_{j+1}, \ldots, a_{p(j+1)})$ hinzunehmen. Nach Definition hätte jedes kürzere Stück einen nichtpositiven Score und würde uns nicht helfen. Dies ist in der folgenden Abbildung veranschaulicht.



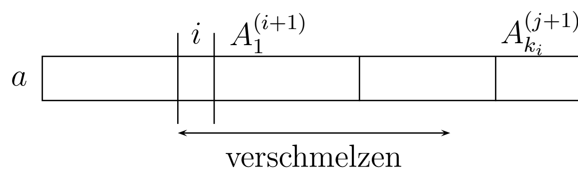

Mit Hilfe der links-negativen Zeiger können wir das Problem mit dem folgenden Algorithmus leicht lösen.

```
LINKS-NEGATIVE PARTITION
{
    j = 1;   ms = 0;   mi = 1;   mj = 0;
    for (i = 1; i ≤ n; i++)
    {
        while ((a ≤ 0) && (i ≤ n))
            i++;
        j := max(i, j)
        while ((j < n) && (p[j + 1] < i + U) && (σ(i, p(i)) > 0))
            j = p(j + 1)
        if (σ(i, j) > ms)
        {
            mi = i;   mj = j;   ms = σ(i, j);
        }
    }
}
```

Abbildung 1.2: Algorithmus: links-negative Partition

**Laufzeit:** Nach jeder Operation wird entweder $i$ oder $j$ erhöht, somit ist die Laufzeit $O(n)$.

Wir haben jedoch die links-negativen Zeiger noch nicht berechnet Wir durchlaufen dazu die Folge vom Ende zum Anfang hin. Treffen wir auf ein Element $a_i > 0$, dann sind wir fertig. Andernfalls verschmelzen wir das Element mit den folgenden Segmenten, bis das Segment einen positiven Score erhält. Dies ist in der folgenden Abbildung illustriert.




Wir können diese Idee im folgenden Algorithmus umsetzen, wobei $s(i) := \sigma(i, p(i))$ bezeichnet.

<u>BERECHNUNG LINKS-NEGATIVER ZEIGER</u>

```
{
    for (i = n; i > 0; i--)
    {
        p(i) = i;
        s(i) = a_i;
        while ((p(i) < n) && (s(i) ≤ 0))
        {
            s(i) = s(i) + s(p(i) + 1);
            p(i) = p(p(i) + 1);
        }
    }
}
```

Abbildung 1.3: Algorithmus: Berechnung linksnegativer Zeiger durch Verschmelzen von Segmenten

**Laufzeit:** Eine simple Abschätzung ergibt, das die Laufzeit im schlimmsten Fall $O(n^2)$ beträgt. Mit einer geschickteren Analyse können wir jedoch wiederum eine lineare Laufzeit zeigen. Es wird pro Iteration maximal ein neues Intervall generiert. In jeder inneren 'while-Schleife' wird ein Intervall eliminiert. Es müssen mindestens so viele Intervalle generiert wie gelöscht (= verschmolzen) werden. Da nur $n$ Intervalle generiert werden, können auch nur $n$ Intervalle gelöscht werden und somit ist die Laufzeit $O(n)$.

## 1.5 Maximum Average Scoring Subsequence

Jetzt wollen wir eine Teilfolge finden, deren Mittelwert (gemittelt über die Länge) maximal ist.

### 1.5.1 Problemstellung

**Notation:** Sei $(a_1, ..., a_n) \in \mathbb{R}^n$, dann ist

$$\begin{aligned} \sigma(i,j) &= \sum_{\ell=i}^{j} a_\ell, \\ \ell(i,j) &= j - i + 1, \\ \mu(i,j) &= \frac{\sigma(i,j)}{\ell(i,j)}. \end{aligned}$$

MAXIMUM AVERAGE SCORING SUBSEQUENCE (MASS)

**Eingabe:** Eine Folge $(a_1, ..., a_n) \in \mathbb{R}^n$, $\lambda \in \mathbb{N}$.
**Gesucht:** Eine (zusammenhängende) Teilfolge $(a_i, ..., a_j)$ mit $\ell(i,j) \geq \lambda$, die $\mu(i,j)$ maximiert, d.h.

$$\mu(i,j) = \max\{\mu(i',j') \mid j' \geq i' + \lambda - 1\}.$$

Wozu haben wir hier die untere Schranke $\lambda$ noch eingeführt? Ansonsten wird das Problem trivial, denn dann ist das maximale Element, interpretiert als einen einelementige Folge, die gesuchte Lösung!

### 1.5.2 Rechtsschiefe Folgen und fallend rechtsschiefe Partitionen

**Definition 1.17** *Eine Folge $A = (a_1, \ldots, a_n)$ heißt* rechtsschief*, wenn der Durchschnitt jedes echten Präfix $(a_1, \ldots, a_i)$ kleiner gleich dem Durchschnitt des korrespondierenden Suffixes $(a_{i+1}, \ldots, a_n)$ ist; d.h. $\mu(1,i) \leq \mu(i+1,n)$ für alle $i \in [1:n-1]$. Eine Partition $A = (A_1, \ldots, A_k)$ von $a$ heißt* fallend rechtsschief*, wenn jedes Segment $A_i$ rechtsschief ist und $\mu(A_i) > \mu(A_j)$ für $i < j$ gilt.*

**Lemma 1.18** *Seien $a$ und $b$ zwei reelle Folgen mit $\mu(a) < \mu(b)$. Dann gilt $\mu(a) < \mu(ab) < \mu(b)$ ($ab$ sei die konkatenierte Folge).*

**Beweis:** Es gilt:

$$\begin{aligned}
\mu(ab) &= \frac{\sigma(ab)}{\ell(ab)} \\
&= \frac{\sigma(a)+\sigma(b)}{\ell(ab)} \\
&= \frac{\mu(a)\cdot\ell(a)+\mu(b)\cdot\ell(b)}{\ell(ab)} \\
&= \frac{\ell(a)}{\ell(ab)}\cdot\mu(a)+\frac{\ell(b)}{\ell(ab)}\cdot\mu(b) \\
&= \frac{\ell(a)}{\ell(ab)}\cdot\mu(a)+\left(1-\frac{\ell(a)}{\ell(ab)}\right)\cdot\mu(b)
\end{aligned}$$

Somit ist zum einen

$$\begin{aligned}
\mu(ab) &= \frac{\ell(a)}{\ell(ab)}\cdot\mu(a)+\left(1-\frac{\ell(a)}{\ell(ab)}\right)\cdot\mu(b) \\
&> \frac{\ell(a)}{\ell(ab)}\cdot\mu(a)+\left(1-\frac{\ell(a)}{\ell(ab)}\right)\cdot\mu(a) \\
&= \mu(a)
\end{aligned}$$

und zum anderen

$$\begin{aligned}
\mu(ab) &= \frac{\ell(a)}{\ell(ab)}\cdot\mu(a)+\left(1-\frac{\ell(a)}{\ell(ab)}\right)\cdot\mu(b) \\
&< \frac{\ell(a)}{\ell(ab)}\cdot\mu(b)+\left(1-\frac{\ell(a)}{\ell(ab)}\right)\cdot\mu(b) \\
&= \mu(b)
\end{aligned}$$

Damit ist die Behauptung gezeigt. ■

**Korollar 1.19** *Seien $a$ und $b$ zwei reelle Folgen mit $\mu(a) \leq \mu(b)$. Dann gilt $\mu(a) \leq \mu(ab) \leq \mu(b)$.*

**Lemma 1.20** *Seien $a$ und $b$ zwei rechtsschiefe Folgen mit $\mu(a) < \mu(b)$. Dann ist auch die Folge $ab$ rechtsschief.*

**Beweis:** Sei $p$ ein beliebiges Präfix von $ab$. Es ist zu zeigen, dass $\mu(p) \leq \mu(q)$ ist, wobei $q$ das zu $p$ korrespondierende Suffix in $ab$ ist.

**1.Fall:** $p = a \Rightarrow \mu(a) < \mu(b)$ nach Voraussetzung.

**2.Fall:** $p$ ist echtes Präfix von $a$, d.h. $a = pa'$. Mit dem vorherigen Korollar gilt (da $\mu(p) \leq \mu(a')$ aufgrund der rechtsschiefen Folge $a$):

$$\mu(p) \leq \mu(pa') \leq \mu(a').$$

Somit ist $\mu(p) \leq \mu(a) \leq \mu(b)$.

Da $\mu(p) \leq \mu(a')$, folgt

$$\begin{aligned} \mu(p) &= \frac{\ell(a')\mu(p) + \ell(b)\mu(p)}{\ell(a'b)} \\ &\leq \frac{\ell(a')\mu(a') + \ell(b)\mu(b)}{\ell(a'b)} \\ &= \frac{\sigma(a') + \sigma(b)}{\ell(a'b)} \\ &= \mu(a'b). \end{aligned}$$

**3.Fall:** Sei $p = ab'$ mit $b = b'b''$. Mit dem vorherigen Korollar folgt:

$$\mu(b') \leq \mu(\underbrace{b'b''}_{=b}) \leq \mu(b'').$$

Somit gilt $\mu(a) \leq \mu(b) \leq \mu(b'')$. Also gilt (da $\mu(a) \leq \mu(b'')$ und $\mu(b') \leq \mu(b'')$):

$$\begin{aligned} \mu(p) &= \mu(ab') \\ &= \frac{\sigma(a) + \sigma(b')}{\ell(ab')} \\ &= \frac{\ell(a)\mu(a) + \ell(b')\mu(b')}{\ell(ab')} \\ &\leq \frac{\ell(a)\mu(b'') + \ell(b')\mu(b'')}{\ell(ab')} \\ &= \mu(b'') \end{aligned}$$

Damit ist das Lemma bewiesen. ■

**Lemma 1.21** *Jede Folge $a = (a_1, \ldots, a_n)$ besitzt eine eindeutig fallend rechtsschiefe Partition.*

**Beweis: (durch Induktion)**

**Induktionsanfang ($n = 1$):** Klar, nach Definition.

**Induktionsschritt ($n \to n+1$):** Wir betrachten eine Folge $(a_1, \ldots, a_n, a_{n+1})$. Sei weiter $(A_1, \ldots, A_k)$ die fallend rechtsschiefe Partition von $(a_1, \ldots, a_n)$.

Gilt $\mu(a_{n+1}) < \mu(A_k)$, dann ist $(A_1, \ldots, A_k, a_{n+1})$ eine neue rechtschiefe Partition.

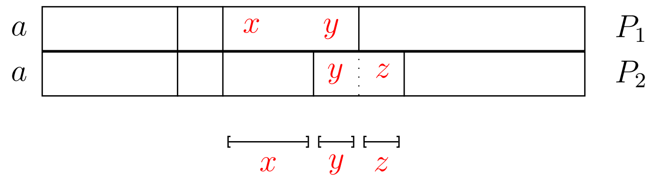
Andernfalls bestimmen wir ein maximales $i$ mit $\mu(A_i \cdots A_k \cdot a_{n+1}) < \mu(A_{i-1})$. Dann behaupten wir, dass die neue Partition $(A_1, \ldots, A_{i-1}, A_i \cdots A_k \cdot a_{n+1})$ rechtsschief ist.

Nach Definition ist $(a_{n+1})$ rechtsschief. Nach dem vorherigen Lemma und der Wahl von $i$ ist dann auch $A_k \cdot a_{n+1}$ rechtsschief. Weiter gilt allgemein für $j \geq i$, dass $A_{j-1} \cdots A_k \cdot a_{n+1}$ rechtsschief ist. Somit ist auch $A_{i-1} \cdots A_k \cdot a_{n+1}$ rechtsschief.

Nach Konstruktion ist die jeweils konstruierte Partition eine fallend rechtsschiefe Partition.

**29. April**

Es bleibt noch die Eindeutigkeit zu zeigen. Nehmen wir an, es gäbe zwei verschiedene solche Partitionen. Betrachten wir, wie in der folgenden Abbildung skizziert, die jeweils linkesten Teilfolgen in ihren Partition, in denen sich die beiden Partitionen unterscheiden.




Da $yz$ nach der Partition $P_2$ rechtsschief ist, gilt $\mu(y) \leq \mu(z)$. Mit Lemma 1.18 folgt, dass $\mu(y) \leq \mu(yz) \leq \mu(z)$.

Nach Wahl der fallend rechtsschiefen Partition $P_2$ gilt $\mu(x) > \mu(yz) \geq \mu(y)$.

Damit ist $\mu(x) > \mu(y)$ und somit ist $xy$ nicht rechtsschief. Dies ist ein Widerspruch zur Annahme, dass $P_1$ eine fallend rechtsschiefe Partition ist. ■

### 1.5.3 Algorithmus zur Konstruktion rechtsschiefer Partitionen

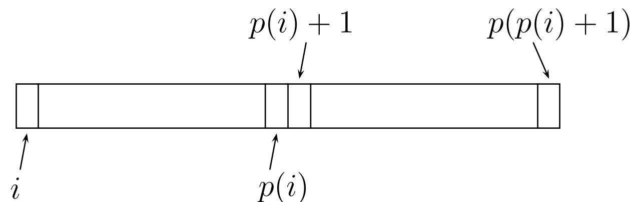
**Notation:** Sei $a = (a_1, \ldots, a_n) \in \mathbb{R}^n$. Jedes Suffix $a^{(i)} = (a_i, \ldots, a_n)$ von $a$ definiert eine eindeutige fallend rechtsschiefe Partition: $a^{(i)} = (A_1^{(i)}, \ldots, A_{k_i}^{(i)})$.

Sei $A_1^{(i)} = (a_i, \ldots, a_{p(i)})$, dann heißt $p(i)$ der *rechtschiefe Zeiger* von $i$.

**Notation:** Im Folgenden verwenden wir der Einfachheit halber die beiden folgenden Abkürzungen:

$$\begin{aligned} s(i) &:= \sigma(i, p(i)), \\ \ell(i) &:= \ell(i, p(i)) = p(i) - i + 1. \end{aligned}$$

Für die Konstruktion der rechtsschiefen Zeiger arbeiten wir uns wieder vom Ende zum Anfang durch die gegebene Folge durch. Für ein neu betrachtetes Element setzen wir zunächst die rechtsschiefe Folge auf diese einelementige Folge. Ist nun der Mittelwert des aktuell betrachten Intervalls kleiner gleich dem Mittelwert des nächsten, so verschmelzen wir diese beiden und machen dieses neue Intervall zum aktuell betrachteten. Andernfalls haben wir die Eigenschaft einer fallend rechtsschiefen Partition sichergestellt. Dies ist in der folgenden Abbildung schematisch dargestellt.




Da diese eindeutig ist, erhalten wir zum Schluss die gewünschte fallend rechtsschiefe Partition samt aller rechtsschiefer Zeiger. Somit können wir den folgenden Algorithmus zur Konstruktion rechtsschiefer Zeiger formalisieren.

<u>Konstruktion rechtsschiefer Zeiger</u>

```
{
    for (i = n; i > 0; i--)
    {
        p(i) = i;
        s(i) = a_i;
        ℓ(i) = 1;
        while ((s(i)/ℓ(i) ≤ s(p(i) + 1)/ℓ(p(i) + 1)) && (p(i) ≤ n))
        {
            s(i) = s(i) + s(p(i) + 1);
            ℓ(i) = ℓ(i) + ℓ(p(i) + 1);
            p(i) = p(p(i) + 1);
        }
    }
}
```

Abbildung 1.4: Algorithmus: Rechtsschiefe Zeiger

**Lemma 1.22** *Die rechtsschiefen Zeiger lassen sich in Zeit $O(n)$ berechnen.*

**Beweis:** Analog zum Beweis der Laufzeit des Algorithmus zur Konstruktion linksnegativer Zeiger. ■

**Lemma 1.23** *Sei $a = (a_1, \ldots, a_n) \in \mathbb{R}^n$ und $(a_i, \ldots, a_j)$ eine kürzeste Teilfolge von $a$ der Länge mindestens $\lambda$, die deren Average Score $\mu(i, j)$ maximiert. Dann gilt $\ell(i, j) = j - i + 1 \leq 2\lambda - 1$.*

**Beweis:** Siehe Übungsblatt 2 Aufgabe 3. ■

**Lemma 1.24** *Seien $a$, $b$ und $c$ drei reelle Folgen mit $\mu(a) < \mu(b) < \mu(c)$. Dann gilt $\mu(ab) < \mu(abc)$.*

Zunächst geben wir ein Gegenbeispiel an, um zu zeigen, dass im vorherigen Lemma die Bedingung $\mu(a) < \mu(b)$ notwendig ist.

**Gegenbeispiel:** $a = 11$
$b = 1$ $\quad \mu(ab) = \frac{12}{2} = 6$
$c = 3$ $\quad \mu(abc) = \frac{15}{3} = 5$

**Beweis:** Nach Lemma 1.18 gilt $\mu(a) < \mu(ab) < \mu(b) < \mu(c)$ Somit gilt $\mu(ab) < \mu(c)$. Nach Lemma 1.18 gilt $\mu(ab) < \mu(abc) < \mu(c)$ und somit ist das Lemma bewiesen. ■

**Lemma 1.25** *Seien $a = (a_1, \ldots, a_n) \in \mathbb{R}^n$ und $p \in \mathbb{R}^n$ und sei $(A_1, \ldots, A_k)$ eine fallend rechtsschiefe Partition von $a$. Sei weiter $m$ maximal gewählt, so dass*

$$\mu(p \cdot A_1 \cdots A_m) = \max\{\mu(p \cdot A_1 \cdots A_i) \mid i \in [0 : k]\}.$$

*Es gilt genau dann $\mu(p \cdot A_1 \cdots A_i) > \mu(A_{i+1})$, wenn $i \geq m$ gilt.*

**Beweis:**

'$\Longrightarrow$:' Sei $i$ so gewählt, dass $\mu(p \cdot A_1 \cdots A_i) > \mu(A_{i+1})$. Da $(A_1, \ldots, A_k)$ eine fallend rechtsschiefe Partition von $a$ ist, gilt $\mu(A_1) > \mu(A_2) > \cdots > \mu(A_k)$. Dann gilt auch $\mu(p \cdot A_1 \cdots A_i) > \mu(A_{i+1}) > \mu(A_{i+2}) > \cdots > \mu(A_k)$. Nach Lemma 1.18 gilt dann $\mu(p \cdot A_1 \cdots A_i) > \mu(p \cdot A_1 \cdots A_{i+1}) > \mu(A_{i+1}) > \cdots$ und somit $i \geq m$.

'$\Longleftarrow$:'

**1. Fall:** $\mu(p \cdot A_1 \cdots A_m) > \mu(A_{m+1}) > \mu(A_{m+2}) > \cdots > \mu(A_k)$.

Nach Lemma 1.18 gilt dann $\mu(p \cdot A_1 \cdots A_{m+1}) > \mu(A_{m+1}) > \mu(A_{m+2})$. Nach Lemma 1.18 gilt $\mu(p \cdot A_1 \cdots A_{m+2}) > \mu(A_{m+2}) > \mu(A_{m+3})$.

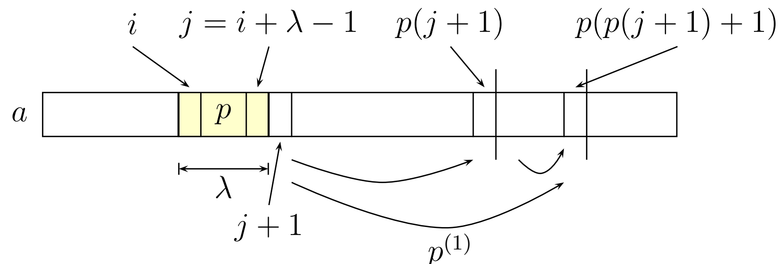
**2. Fall:** $\mu(p \cdot A_1 \cdots A_m) \leq \mu(A_{m+1})$.

Nach Korollar 1.19 gilt $\mu(p \cdot A_1 \cdots A_m) \leq \mu(p \cdot A_1 \cdots A_{m+1})$. Aufgrund der Maximalität von $m$ gilt dann $\mu(p \cdot A_1 \cdots A_m) = \mu(p \cdot A_1 \cdots A_{m+1})$. Daraus ergibt sich ein Widerspruch zur Maximalität von $m$.

Damit ist die Behauptung gezeigt. ■

### 1.5.4 Ein Algorithmus für MASS

Somit können wir einen Algorithmus zur Lösung unseres Problems angeben, der im Wesentlichen auf Lemma 1.25 basiert. Wir laufen von links nach rechts durch die Folge, und versuchen, ab Position $i$ die optimale Sequenz mit maximalem Mittelwert zu finden. Nach Voraussetzung hat diese mindestens die Länge $\lambda$ und nach Lemma 1.23 auch höchstens die Länge $2\lambda - 1$.




Ein einfacher Algorithmus würde jetzt alle möglichen Längen aus dem Intervall $[\lambda : 2\lambda - 1]$ ausprobieren. Der folgende Algorithmus würde dies tun, wenn die Prozedur `locate` geeignet implementiert wäre, wobei `locate` einfach die Länge einer optimalen Teilfolge zurückgibt. Würden wir `locate` mittels einer linearen Suche implementieren, so würden wir einen Laufzeit von $O(n\lambda) = O(n^2)$ erhalten. Abschließend müssen wir nur noch unter allen Paaren $(i, g(i))$ dasjenige bestimmen, bei dem $\mu(i, g(i))$ maximal ist.

Mit Hilfe von Lemma 1.25 können wir aber auch geschickter vorgehen. Das Lemma besagt nämlich, dass wenn wir rechts vom optimalen Schnitt sind, der Mittelwert größer als der des folgenden Intervalls ist. Links vom Optimum gilt, dass der Wert

```
MASS
{
    for (i = 1; i ≤ n; i++)
    {
        j = i + λ − 1;
        if (μ(i, j) < μ(j + 1, p(j + 1)))
            j = locate(i, j);
        g(i) = j;
    }
    return (i, g(i)) s.t. μ(i, g(i)) is maximal;
}
```

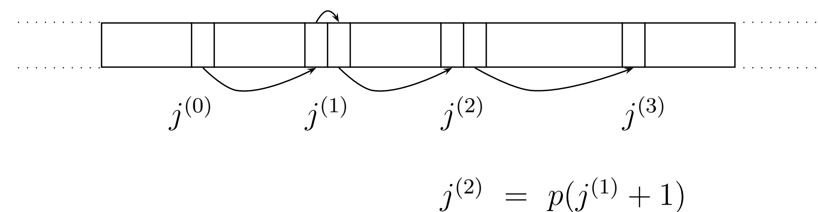
Abbildung 1.5: Algorithmus: MASS

kleiner als der Mittelwert vom folgenden Intervall ist. Somit könnten wir hier eine binäre Suche ausnutzen.

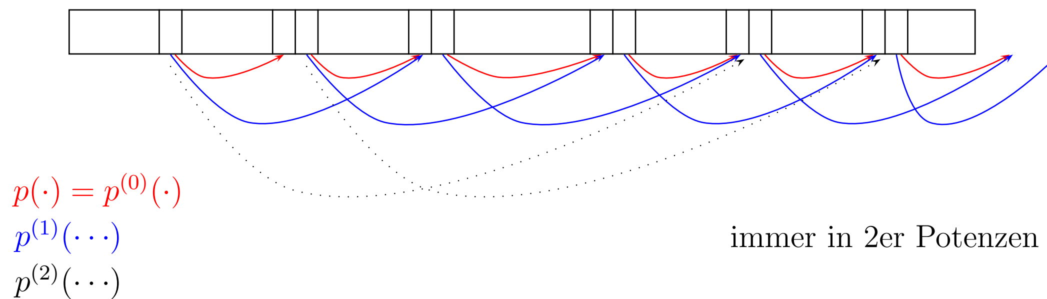
Allerdings kennen wir die Grenzen der Intervalle nicht explizit, sondern nur als lineare Liste über die rechtsschiefen Zeiger. Daher konstruieren wir uns zu den rechtsschiefen Zeiger noch solche, die nicht nur auf das nächste Intervall, sondern auch auf das $2^k$-te folgende Intervall angibt. Dafür definieren wir erst einmal formal die Anfänge der nächsten $2^k$-ten Intervalle sowie die darauf basierenden iterierten rechtsschiefen Zeiger wie folgt:

$$
\begin{aligned}
j^{(0)} &:= j \\
j^{(k)} &= \min\{p(j^{k-1} + 1), n\} \\
p^{(0)}(i) &= p(i) \\
p^{(k)}(i) &= \min\{p^{(k-1)}(p^{(k-1)}(i) + 1), n\}
\end{aligned}
$$

Hierbei gibt $p^{(k)}(i)$ das Ende nach $2^k$ Intervallen an. Dies ist in folgenden Abbildungen illustriert.




$$j^{(2)} = p(j^{(1)} + 1)$$



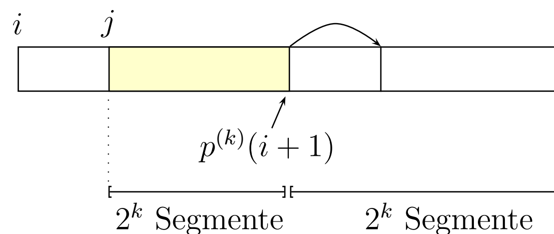

Diese iterierten rechtsschiefen Zeiger lassen sich aus den rechtsschiefen Zeigern in Zeit $O(\log(\lambda))$ berechnen, da wir ja maximal die $2\lambda$-iterierten rechtschiefen Zeiger aufgrund der oberen Längenbeschränkung von $2\lambda - 1$ benötigen.

Mit Hilfe dieser iterierten rechtsschiefen Zeiger können wir jetzt die binäre Suche in der Prozedur `locate` wie im folgenden Algorithmus implementieren.

```
LOCATE (i, j)
{
    for (k = log(λ); k ≥ 0; k++)
    {
        if ((j > n) || (μ(i, j) ≥ μ(j + 1, p(j + 1))))
            return i
        if ((p^(k)(j + 1) < n) &&
          (μ(i, p^(k)(j + 1)) < μ(p^(k)(j + 1) + 1, p(p^(k)(j + 1) + 1))))
            j = p^(k)(j + 1);
    }
    if ((j < n) && (μ(i, j) < μ(j + 1, p(j + 1))))
        j = p(j + 1);
    return j;
}
```

Abbildung 1.6: Algorithmus: locate$(i, j)$

Die Strategie der Suche ist noch einmal in der folgenden Abbildung illustriert:



Zusammenfassend erhalten wir das folgende Theorem.

**Theorem 1.26** *Sei* $(a_1, \ldots, a_n) \in \mathbb{R}^n$*. Eine kürzeste Teilfolge der Länge mindestens* $\lambda$ *und mit maximalen average Score kann in Zeit* $O(n \log(\lambda))$ *gefunden werden.*

Es bleibt eine offenen Frage, ob sich auch die optimale Teilfolge bezüglich des Mittelwerts in linearer Zeit für beliebige untere Schranken $\lambda$ bestimmen lässt.

**6. Mai**

# Phylogenetische Bäume 2

## 2.1 Einleitung

In diesem Kapitel wollen wir uns mit *phylogenetischen Bäumen* bzw. *evolutionären Bäumen* beschäftigen. Wir wollen also die Entwicklungsgeschichte mehrerer verwandter Spezies anschaulich als Baum darstellen bzw. das Auftreten von Unterschieden in den Spezies durch Verzweigungen in einem Baum wiedergeben.

**Definition 2.1** *Ein* phylogenetischer Baum *für eine Menge* $S = \{s_1, \ldots, s_n\}$ *von* $n$ *Spezies ist ein ungeordneter gewurzelter Baum mit* $n$ *Blättern und den folgenden Eigenschaften:*

- *Jeder innere Knoten hat mindestens zwei Kinder;*
- *Jedes Blatt ist mit genau einer Spezies* $s \in S$ *markiert;*
- *Jede Spezies taucht nur einmal als Blattmarkierung auf.*

Ungeordnet bedeutet hier, dass die Reihenfolge der Kinder eines Knotens ohne Belang ist. Die bekannten und noch lebenden (zum Teil auch bereits ausgestorbenen) Spezies werden dabei an den Blättern dargestellt. Jeder (der maximal $n-1$) inneren Knoten entspricht dann einem Ahnen der Spezies, die in seinem Teilbaum die Blätter bilden. In Abbildung 2.1 ist ein Beispiel eines phylogenetischen Baumes angegeben.

Wir wollen uns hier mit der mathematischen und algorithmischen Rekonstruktion von phylogenetischen Bäumen anhand der gegebenen biologischen Daten beschäftigen. Die daraus resultierenden Bäume müssen daher nicht immer mit der biologischen Wirklichkeit übereinstimmen. Die rekonstruierten phylogenetischen Bäume mögen Ahnen vorhersagen, die niemals existiert haben.

Dies liegt zum einen daran, dass die biologischen Daten nicht in der Vollständigkeit und Genauigkeit vorliegen, die für eine mathematische Rekonstruktion nötig sind. Zum anderen liegt dies auch an den vereinfachenden Modellen, da in der Natur nicht nur Unterscheidungen (d.h. Verzweigungen in den Bäumen) vorkommen können, sondern dass auch Vereinigungen bereits getrennter Spezies vorkommen können.

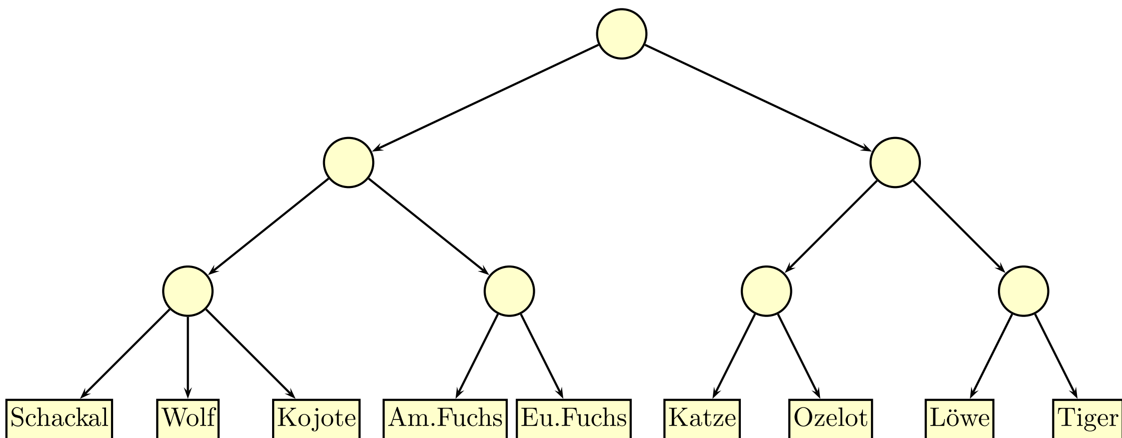



Abbildung 2.1: Beispiel: Ein phylogenetischer Baum

Biologisch würde man daher eher nach einem gerichteten azyklischen Graphen statt eines gewurzelten Baumes suchen. Da diese Verschmelzungen aber eher vereinzelt vorkommen, bilden phylogenetischer Bäume einen ersten Ansatzpunkt, in die dann weiteres biologisches Wissen eingearbeitet werden kann.

In der Rekonstruktion unterscheidet man zwei prinzipiell unterschiedliche Verfahren: distanzbasierte und charakterbasierte Verfahren, die wir in den beiden folgenden Unterabschnitten genauer erörtern werden.

### 2.1.1 Distanzbasierte Verfahren

Bei den so genannten *distanzbasierten Verfahren* wird zwischen den Spezies ein Abstand bestimmt. Man kann diesen einfach als die Zeitspanne in die Vergangenheit interpretieren, vor der sich die beiden Spezies durch Spezifizierung aus einem gemeinsamen Urahn auseinander entwickelt haben.

Für solche Distanzen, also evolutionäre Abstände, können beispielsweise die EDIT-Distanzen von speziellen DNS-Teilsträngen oder Aminosäuresequenzen verwendet werden. Hierbei wird angenommen, dass sich die Sequenzen durch Mutationen auseinander entwickeln und dass die Anzahl der so genannten akzeptierten Mutationen (also derer, die einem Weiterbestehen der Art nicht im Wege standen) zur zeitlichen Dauer korreliert ist. Hierbei muss man vorsichtig sein, da unterschiedliche Bereiche im Genom auch unterschiedliche Mutationsraten besitzen.

Eine andere Möglichkeit aus früheren Tagen sind Hybridisierungsexperimente. Dabei werden durch vorsichtiges Erhitzen die DNS-Doppelstränge zweier Spezies voneinander getrennt. Bei der anschließenden Abkühlung hybridisieren die DNS-Stränge

wieder miteinander. Da jetzt jedoch DNS-Einzelstränge von zwei Spezies vorliegen, können auch zwei Einzelstränge von zwei verschiedenen Spezies miteinander hybridisieren, vorausgesetzt, die Stränge waren nicht zu verschieden.

Beim anschließenden erneuten Erhitzen trennen sich diese gemischten Doppelstränge umso schneller, je verschiedener die DNS-Sequenzen sind, da dann entsprechend weniger Wasserstoffbrücken aufzubrechen sind. Aus den Temperaturen, bei denen sich dann diese gemischten DNS-Doppelstränge wieder trennen, kann man dann ein evolutionäres Abstandsmaß gewinnen.

Ziel der evolutionären Verfahren ist es nun, einen Baum mit Kantengewichten zu konstruieren, so dass das Gewicht der Pfade von den zwei Spezies zu ihrem niedrigsten gemeinsamen Vorfahren dem Abstand entspricht. Ein solcher phylogenetischer Baum, der aufgrund von künstlichen evolutionären Distanzen konstruiert wurde, ist in der Abbildung 2.2 illustriert.

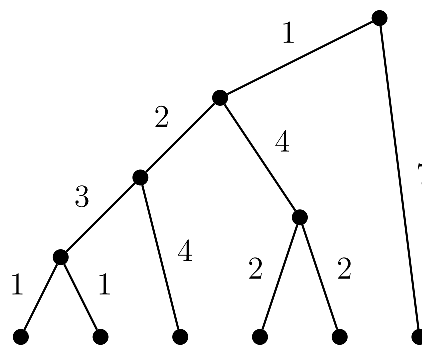



Abbildung 2.2: Beispiel: Ein distanzbasierter phylogenetischer Baum

## 2.1.2 Charakterbasierte Methoden

Bei den so genannten *charakterbasierten Verfahren* verwendet man gewisse Eigenschaften, so genannte *Charaktere*, der Spezies. Hierbei unterscheidet man *binäre Charaktere*, wie beispielsweise „ist ein Säugetier“, „ist ein Wirbeltier“, „ist ein Fisch“, „ist ein Vogel“, „ist ein Lungenatmer“, etc., *numerische Charaktere*, wie beispielsweise Anzahl der Extremitäten, Anzahl der Wirbel, etc, und *zeichenreihige Charaktere*, wie beispielsweise bestimmte Teilsequenzen in der DNS. Bei letzterem betrachtet man oft Teilsequenzen aus nicht-codierenden und nicht-regulatorischen Bereichen der DNS, da diese bei Mutationen in der Regel unverändert weitergegeben werden und nicht durch Veränderung einer lebenswichtigen Funktion sofort aussterben.

Das Ziel ist auch hier wieder die Konstruktion eines phylogenetischen Baumes, wobei die Kanten mit Charakteren und ihren Änderungen markiert werden. Eine Markierung einer Kante mit einem Charakter bedeutet hierbei, dass alle Spezies in dem Teilbaum nun eine Änderung dieses Charakters erfahren. Die genaue Änderung dieses Charakters ist auch an der Kante erfasst.

Bei charakterbasierten Verfahren verfolgt man das Prinzip der minimalen Mutationshäufigkeit bzw. der maximalen Parsimonie (engl. parsimony, Geiz). Das bedeutet, dass man einen Baum sucht, der so wenig Kantenmarkierungen wie möglich besitzt. Man geht hierbei davon aus, dass die Natur keine unnötigen Mutationen verwendet.

In der Abbildung 2.3 ist ein Beispiel für einen solchen charakterbasierten phylogenetischen Baum angegeben, wobei hier nur binäre Charaktere verwendet wurden. Außerdem werden die binären Charaktere hier so verwendet, dass sie nach einer Kantenmarkierung in den Teilbaum eingeführt und nicht gelöscht werden. Bei binären Charakteren kann man dies immer annehmen, da man ansonsten den binären Charakter nur negieren muss.

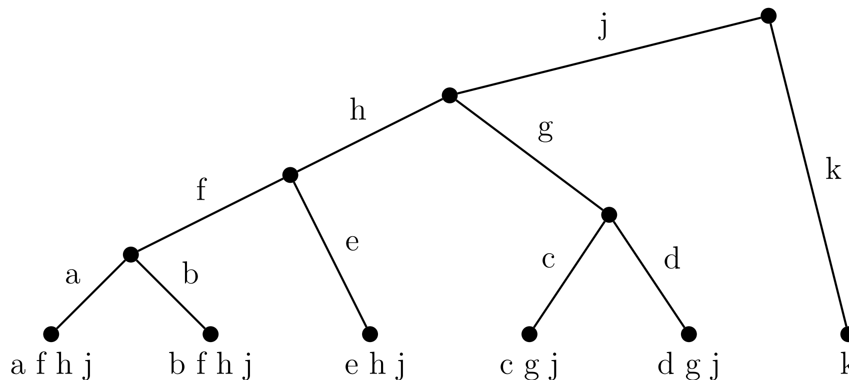



Abbildung 2.3: Beispiel: Ein charakterbasierter phylogenetischer Baum

## 2.2 Ultrametriken und ultrametrische Bäume

Wir wollen uns zuerst mit distanzbasierten Methoden beschäftigen. Dazu stellen wir zuerst einige schöne und einfache Charakterisierungen vor, ob eine gegebene Distanzmatrix einen phylogenetischen Baum besitzt oder nicht.

### 2.2.1 Metriken und Ultrametriken

Zuerst müssen wir noch ein paar Eigenschaften von Distanzen wiederholen und einige hier nützliche zusätzliche Definitionen angeben. Zuerst wiederholen wir die Definition einer Metrik.

**Definition 2.2** *Eine Funktion $d : M^2 \to \mathbb{R}_+$ heißt* Metrik, *wenn*

*(M1)* $\forall x, y \in M : d(x, y) = 0 \Leftrightarrow x = y$ *(Definitheit),*

*(M2)* $\forall x, y \in M : d(x, y) = d(y, x)$ *(Symmetrie),*

*(M3)* $\forall x, y, z \in M : d(x, z) \leq d(x, y) + d(y, z)$ *(Dreiecksungleichung).*

Im Folgenden werden wir auch die folgende verschärfte Variante der Dreiecksungleichung benötigen.

**Definition 2.3** *Eine Metrik heißt* Ultrametrik, *wenn zusätzlich die so genannte* ultrametrische Dreiecksungleichung *gilt:*

$$\forall x, y, z \in M : d(x, z) \leq \max\{d(x, y), d(y, z)\}.$$

Eine andere Charakterisierung der ultrametrischen Ungleichung wird uns im Folgenden aus beweistechnischen Gründen nützlich sein.

**Lemma 2.4** *Sei $d$ eine Ultrametrik auf $M$. Dann sind für alle $x, y, z \in M$ die beiden größten Zahlen aus $d(x, y)$, $d(y, z)$ und $d(x, z)$ gleich.*

**Beweis:** Zuerst gelte die ultrametrische Dreiecksungleichung für alle $x, y, z \in M$:

$$d(x, z) \leq \max\{d(x, y), d(y, z)\}.$$

Ist $d(x, y) = d(y, z)$, dann ist nichts zu zeigen. Sei also ohne Beschränkung der Allgemeinheit $d(x, y) < d(y, z)$. Dann ist auch $d(x, z) \leq d(y, z)$.

Aufgrund der ultrametrischen Ungleichung gilt ebenfalls:

$$d(y, z) \leq \max\{d(y, x), d(x, z)\} = d(x, z).$$

Die letzte Ungleichung folgt aus der obigen Tatsache, dass $d(y, z) > d(y, x)$.

Zusammen gilt also $d(x, z) \leq d(y, z) \leq d(x, z)$. Also gilt $d(x, z) = d(y, z) > d(x, y)$ und das Lemma ist bewiesen. ■

Nun zeigen wir auch noch die umgekehrte Richtung.

**Lemma 2.5** *Sei $d : M^2 \to \mathbb{R}_+$, wobei für alle $x, y \in M$ genau dann $d(x, y) = 0$ gilt, wenn $x = y$. Weiter gelte, dass für alle $x, y, z \in M$ die beiden größten Zahlen aus $d(x, y)$, $d(y, z)$ und $d(x, z)$ gleich sind. Dann ist $d$ eine Ultrametrik.*

**Beweis:** Die Definitheit (M1) gilt nach Voraussetzung.

Für die Symmetrie (M2) betrachten wir beliebige $x, y \in M$. Aus der Voraussetzung folgt mit $z = x$, dass von $d(x, y)$, $d(y, x)$ und $d(x, x)$ die beiden größten Werte gleich sind. Da nach Voraussetzung $d(x, x) = 0$ sowie $d(x, y) \geq 0$ und $d(y, x) \geq 0$ gilt, folgt, dass $d(x, y) = d(y, x)$ die beiden größten Werte sind und somit nach Voraussetzung gleich sein müssen.

Für die ultrametrische Dreiecksungleichung ist Folgendes zu zeigen:

$$\forall x, y, z \in M : d(x, z) \leq \max\{d(x, y), d(y, z)\}.$$

Wir unterscheiden drei Fälle, je nachdem, welche beiden Werte der drei Distanzen die größten sind und somit nach Voraussetzung gleich sind.

**Fall 1 ($d(x, z) \leq d(x, y) = d(y, z)$):** Die Behauptung lässt sich sofort verifizieren.

**Fall 2 ($d(y, z) \leq d(x, y) = d(x, z)$):** Die Behauptung lässt sich sofort verifizieren.

**Fall 3 ($d(x, y) \leq d(x, z) = d(y, z)$):** Die Behauptung lässt sich sofort verifizieren. ∎

Da die beiden Lemmata bewiesen haben, dass von drei Abständen die beiden größten gleich sind, nennt man diese Eigenschaft auch *3-Punkte-Bedingung*.

Zum Schluss dieses Abschnittes definieren wir noch so genannte Distanzmatrizen, aus denen wir im Folgenden die evolutionären Bäumen konstruieren wollen.

**Definition 2.6** *Sei $D = (d_{i,j})$ eine symmetrische $n \times n$-Matrix mit $d_{i,i} = 0$ und $d_{i,j} > 0$ für alle $i, j \in [1 : n]$. Dann heißt $D$ eine* Distanzmatrix.

### 2.2.2 Ultrametrische Bäume

Zunächst einmal definieren wir spezielle evolutionäre Bäume, für die sich, wie wir sehen werden, sehr effizient die gewünschten Bäume konstruieren lassen. Bevor wir diese definieren können, benötigen wir noch den Begriff des niedrigsten gemeinsamen Vorfahren von zwei Knoten in einem Baum.

**Definition 2.7** *Sei* $T = (V, E)$ *ein gewurzelter Baum. Seien* $v, w \in V$ *zwei Knoten von* $T$*. Der* niedrigste gemeinsame Vorfahr *von* $v$ *und* $w$*, bezeichnet als* $\text{lca}(v, w)$ *(engl.* least common ancestor*), ist der Knoten* $u \in V$*, so dass* $u$ *sowohl ein Vorfahr von* $v$ *als auch von* $w$ *ist und es keinen echten Nachfahren von* $u$ *gibt, der ebenfalls ein Vorfahr von* $v$ *und* $w$ *ist.*

Mit Hilfe des Begriffs des niedrigsten gemeinsamen Vorfahren können wir jetzt ultrametrische Bäume definieren.

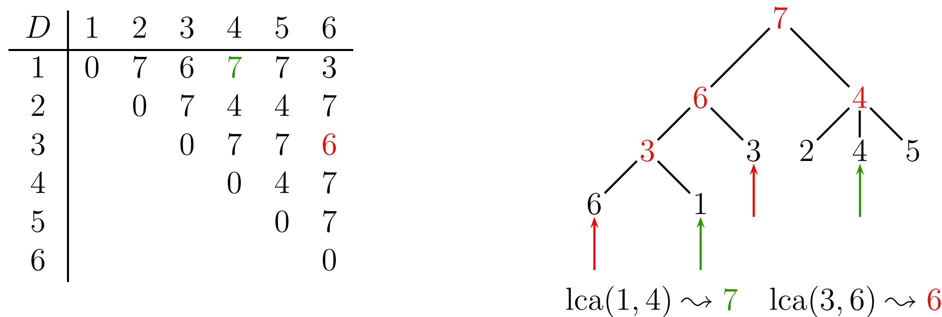
**Definition 2.8** *Sei* $D$ *eine* $n \times n$*-Distanzmatrix. Ein* (strenger) ultrametrischer Baum $T$ *für* $D$ *ist ein Baum* $T = T(D)$ *mit*

1. $T$ *besitzt* $n$ *Blätter, die bijektiv mit* $[1 : n]$ *markiert sind;*
2. *Jeder innere Knoten von* $T$ *besitzt mindestens 2 Kinder, die mit Werten aus* $D$ *markiert sind;*
3. *Entlang eines jeden Pfades von der Wurzel von* $T$ *zu einem Blatt ist die Folge der Markierungen an den inneren Blättern (streng) monoton fallend;*
4. *Für je zwei Blätter* $i$ *und* $j$ *von* $T$ *ist die Markierung des niedrigsten gemeinsamen Vorfahren gleich* $d_{ij}$*.*

In Folgenden werden wir hauptsächlich strenge ultrametrische Bäume betrachten. Wir werden jedoch der Einfachheit wegen immer von ultrametrischen Bäume sprechen. In Abbildung 2.4 ist ein Beispiel für eine $6 \times 6$-Matrix angegeben, die einen ultrametrischen Baum besitzt.

Wir wollen an dieser Stelle noch einige Bemerkungen zu ultrametrischen Bäumen festhalten.

- Nicht jede Matrix $D$ besitzt einen ultrametrischen Baum. Dies folgt aus der Tatsache, dass jeder Baum, in dem jeder innere Knoten mindestens zwei Kinder besitzt, maximal $n - 1$ innere Knoten besitzen kann. Dies gilt daher insbesondere für ultrametrische Bäume Also können in Matrizen, die einen ultrametrischen Baum besitzen, nur $n - 1$ von Null verschiedene Werte auftreten.



| $D$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 0 | 7 | 6 | 7 | 7 | 3 |
| 2 | | 0 | 7 | 4 | 4 | 7 |
| 3 | | | 0 | 7 | 7 | 6 |
| 4 | | | | 0 | 4 | 7 |
| 5 | | | | | 0 | 7 |
| 6 | | | | | | 0 |



Abbildung 2.4: Beispiel: ultrametrischer Baum

- Der Baum $T(D)$ für $D$ heißt auch *kompakte Darstellung* von $D$, da sich eine Matrix mit $n^2$ Einträgen durch einen Baum der Größe $O(n)$ darstellen lässt.
- Die Markierung an den inneren Knoten können als Zeitspanne in die Vergangenheit interpretiert werden. Vor diesem Zeitraum haben sich die Spezies, die in verschiedenen Teilbäumen auftreten, auseinander entwickelt.

Unser Ziel wird es jetzt sein, festzustellen, ob eine gegebene Distanzmatrix einen ultrametrischen Baum besitzt oder nicht. Zuerst einmal überlegen wir uns, dass es es sehr viele gewurzelte Bäume mit $n$ Blättern gibt. Somit scheidet ein einfaches Ausprobieren aller möglichen Bäume aus.

**Lemma 2.9** *Die Anzahl der ungeordneten binären gewurzelten Bäume mit $n$ Blättern beträgt*

$$\prod_{i=2}^{n}(2i-3) = \frac{(2n-3)!}{2^{n-2}\cdot(n-2)!}.$$

Um ein besseres Gefühl für dieses Anzahl zu bekommen, rechnet man leicht nach, dass

$$\prod_{i=2}^{n}(2i-3) \geq (n-1)! \geq 2^{n-2}$$

gilt. Eine bessere Abschätzung lässt sich natürlich mit Hilfe der Stirlingschen Formel bekommen.

**Beweis:** Wir führen den Beweis durch vollständige Induktion über $n$.

**Induktionsanfang ($n = 2$):** Hierfür gilt die Formel offensichtlich, das es genau einem Baum mit zwei markierten Blättern gibt.

**Induktionsanfang ($n-1 \to n$):** Sei $T$ ein ungeordneter binärer gewurzelter Baum mit $n$ Blättern. Der Einfachheit nehmen wir im Folgenden an, dass unser Baum noch eine Superwurzel besitzt, deren einziges Kind die ursprüngliche Wurzel von $T$ ist.

Wir entfernen jetzt das Blatt $v$ mit der Markierung $n$. Der Elter $w$ davon hat jetzt nur noch ein Kind und wir entfernen es ebenfalls. Dazu wird das andere Kind von $w$ jetzt ein Kind des Elters von $w$ (anstatt von $w$). Den so konstruierten Baum nennen wir $T'$. Wir merken noch an, dass genau eine Kante eine „Erinnerung“ an das Entfernen von $v$ und $w$ hat. Falls $w$ die Wurzel war, so bleibt die Superwurzel im Baum und die Kante von der Superwurzel hat sich das Entfernen „gemerkt“.

Wir stellen fest, dass $T'$ ein ungeordneter binärer gewurzelter Baum ist. Davon gibt es nach Induktionsvoraussetzung $\prod_{i=2}^{n-1}(2i-3)$ viele. Darin kann an jeder Kante $v$ mit seinem Elter $w$ entfernt worden sein.

Wie viele Kanten besitzt ein binärer gewurzelter Baum mit $n-1$ Blättern? Ein binärer gewurzelter Baum mit $n-1$ Blättern besitzt genau $n-2$ innere Knoten plus die von uns hinzugedachte Superwurzel. Somit besitzt der Baum

$$(n-1)+(n-2)+1 = 2n-2$$

Knoten. Da in einem Baum die Anzahl der Kanten um eines niedriger ist als die Anzahl der Knoten, besitzt unser Baum $2n-3$ Kanten, die sich an einen Verlust eines Blattes „erinnern“ können. Somit ist die Gesamtanzahl der ungeordneten binären gewurzelten Bäume mit $n$ Blättern genau

$$(2n-3)\cdot\prod_{i=2}^{n-1}(2i-3) = \prod_{i=2}^{n}(2i-3)$$

und der Induktionschluss ist vollzogen. ■

Wir fügen noch eine ähnliche Behauptung für die Anzahl ungewurzelter Bäume an. Der Beweis ist im Wesentlichen ähnlich zu dem vorherigen.

**Lemma 2.10** *Die Anzahl der ungewurzelten (freien) Bäume mit n Blättern, deren innere Knoten jeweils den Grad 3 besitzen, beträgt*

$$\prod_{i=3}^{n}(2i-5) = \frac{(2n-5)!}{2^{n-3}\cdot(n-3)!}.$$

Wir benötigen jetzt noch eine kurze Definition, die Distanzmatrizen und Metriken in Beziehung setzen.

**Definition 2.11** *Eine $n \times n$-Distanzmatrix $M$* induziert *eine Metrik bzw. Ultrametrik auf* $[1 : n]$*, wenn die Funktion $d : [1 : n]^2 \to \mathbb{R}_+$ mit $d(x, y) = M_{x,y}$ eine Metrik bzw. Ultrametrik ist.*

Mit Hilfe der oben erwähnten Charakterisierung einer Ultrametrik, dass von den drei Abständen zwischen drei Punkten, die beiden größten gleich sind, können wir sofort einen einfachen Algorithmus zur Erkennung ultrametrischer Matrizen angeben. Wir müssen dazu nur alle dreielementigen Teilmengen aus $[1 : n]$ untersuchen, ob von den drei verschiedenen Abständen, die beiden größten gleich sind. Falls ja, ist die Matrix ultrametrisch, ansonsten nicht.

Dieser Algorithmus hat jedoch eine Laufzeit $O(n^3)$. Wir wollen im Folgenden einen effizienteren Algorithmus zur Konstruktion ultrametrischer Matrizen angeben, der auch gleichzeitig noch die zugehörigen ultrametrischen Bäume mitberechnet.

### 2.2.3 Charakterisierung ultrametrischer Bäume

Wir geben jetzt eine weitere Charakterisierung ultrametrischer Matrizen an, deren Beweis sogar einen effizienten Algorithmus zur Konstruktion ultrametrischer Bäume erlaubt.

**Theorem 2.12** *Eine symmetrische $n \times n$-Distanzmatrix $D$ besitzt genau dann einen ultrametrischen Baum, wenn $D$ eine Ultrametrik induziert.*

**Beweis:** **$\Rightarrow$:** Die Definitheit und Symmetrie folgt unmittelbar aus der Definition einer Distanzmatrix. Wir müssen also nur noch die ultrametrische Dreiecksungleichung zeigen:

$$\forall i, j, k \in [1 : n] : \ d_{ij} \leq \max\{d_{ik}, d_{jk}\}.$$

Wir betrachten dazu den ultrametrischen Baum $T = T(D)$. Wir unterscheiden dazu drei Fälle in Abhängigkeit, wie sich die niedrigsten gemeinsamen Vorfahren von $i$, $j$ und $k$ zueinander verhalten. Sei dazu $x = \text{lca}(i, j)$, $y = \text{lca}(i, k)$ und $z = \text{lca}(j, k)$. In einem Baum muss dabei gelten, dass mindestens zwei der drei Knoten identisch sind. Die ersten beiden Fälle sind in Abbildung 2.5 dargestellt.

**Fall 1 ($y = z \neq x$):** Damit folgt aus dem rechten Teil der Abbildung 2.5 sofort, dass $d(i, j) \leq d(i, k) = d(j, k)$.

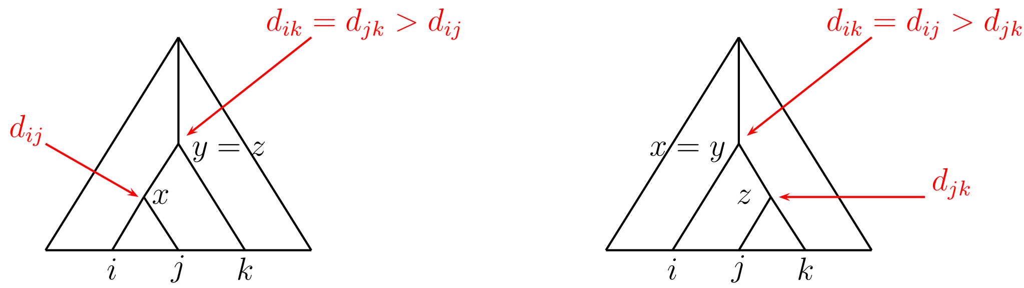



Abbildung 2.5: Skizze: Fall 1 und Fall 2

**Fall 2 ($x = y \neq z$):** Damit folgt aus dem linken Teil der Abbildung 2.5 sofort, dass $d(j,k) \leq d(i,k) = d(i,j)$.

**Fall 3 ($x = z \neq y$):** Dieser Fall ist symmetrisch zu Fall 2 (einfaches Vertauschen von $i$ und $j$).

**Fall 4 ($x = y = z$):** Aus der Abbildung 2.6 folgt auch hier, dass die ultrametrische Dreiecksungleichung gilt, da alle drei Abstände gleich sind. Wenn man statt

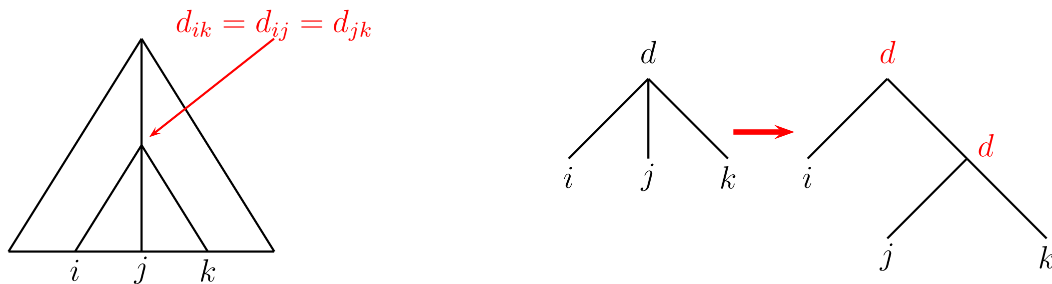



Abbildung 2.6: Skizze: Fall 4 und binäre Neukonstruktion

eines strengen ultrametrischen Baumes lieber einen binären ultrametrischen Baum haben möchte, so kann man ihn beispielsweise wie im rechten Teil der Abbildung 2.6 umbauen.

**8. Mai**

**$\Leftarrow$:** Wir betrachten zuerst die Abstände von Blatt 1 zu allen anderen Blättern. Sei also $\{d_{11}, \ldots, d_{1n}\} = \{\delta_1, \ldots, \delta_k\}$, d.h. $\delta_1, \ldots, \delta_k$ sind die paarweise verschiedenen Abstände, die vom Blatt 1 aus auftreten. Ohne Beschränkung der Allgemeinheit nehmen wir dabei an, dass $\delta_1 < \cdots < \delta_k$. Wir partitionieren dann $[2:n]$ wie folgt:

$$D_i = \{\ell \in [2:n] \ : \ d_{1\ell} = \delta_i\}.$$

Es gilt dann offensichtlich $[2:n] = \biguplus_{i=1}^{k} D_i$. Wir bestimmen jetzt für die Mengen $D_i$ rekursiv die entsprechenden ultrametrischen Bäume. Anschließend konstruieren

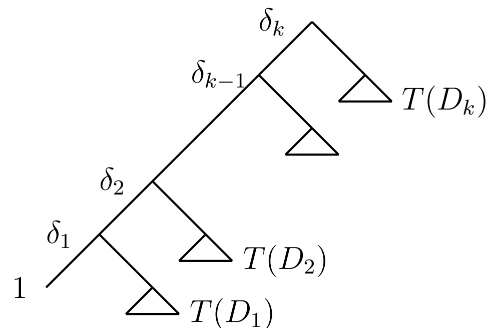



Abbildung 2.7: Skizze: Rekursive Konstruktion ultrametrischer Bäume

wir einen Pfad von der Wurzel zum Blatt 1 mit $k$ inneren Knoten, an die die rekursiv konstruierten Teilbäume $T(D_i)$ angehängt werden. Dies ist in Abbildung 2.7 schematisch dargestellt.

Wir müssen jetzt nachprüfen, ob der konstruierte Baum ultrametrisch ist. Dazu müssen wir zeigen, dass die Knotenmarkierungen auf einem Pfad von der Wurzel zu einem Blatt streng monoton fallend sind und dass der Abstand von den Blättern $i$ und $j$ gerade die Markierung von $\text{lca}(i, j)$ ist.

Für den ersten Teil überlegen wir uns, dass diese sowohl auf dem Pfad von der Wurzel zu Blatt 1 gilt als auch in den Teilbäumen $T(D_i)$. Wir müssen nur noch die Verbindungspunkte überprüfen. Dies ist in Abbildung 2.8 illustriert. Hier sind $x$ und

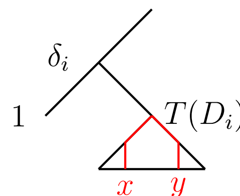



Abbildung 2.8: Skizze: Abstände von $x$ und $y$ und geforderte Monotonie

$y$ zwei Blättern in $T(D_i)$, deren niedrigster gemeinsamer Vorfahre die Wurzel von $T(D_i)$ ist. Wir müssen als zeigen, dass $d_{x,y} < \delta_i$ gilt. Es gilt zunächst aufgrund der ultrametrischen Dreiecksungleichung:

$$d_{xy} \leq \max\{d_{1x}, d_{1y}\} = d_{1x} = d_{1y} = \delta_i.$$

Gilt jetzt $d_{xy} < \delta_i$, dann ist alles gezeigt. Andernfalls gilt $\delta_{xy} = \delta_i$ und wir werden den Baum noch ein wenig umbauen, wie in der folgenden Abbildung 2.9 illustriert. Dabei wird die Wurzel des Teilbaums $T(D_i)$ mit dem korrespondierenden Knoten des Pfades von der Wurzel des Gesamtbaumes zum Blatt 1 miteinander identifiziert und die Kante dazwischen gelöscht. Damit haben wir die strenge Monotonie der Knotenmarkierungen auf den Pfaden von der Wurzel zu den Blättern nachgewiesen.

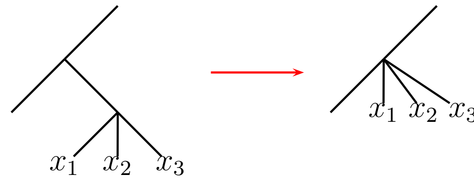



Abbildung 2.9: Skizze: Umbau im Falle nicht-strenger Monotonie

Es ist jetzt noch zu zeigen, dass die Abstände von zwei Blättern $x$ und $y$ den Knotenmarkierungen entsprechen. Innerhalb der Teilbäume $T(D_i)$ gilt dies nach Konstruktion. Ebenfalls gilt dies nach Konstruktion für das Blatt 1 mit allen anderen Blättern.

Wir müssen diese Eigenschaft nur noch nachweisen, wenn sich zwei Blätter in unterschiedlichen Teilbäumen befinden. Sei dazu $x \in V(T(D_i))$ und $y \in V(T(D_j))$, wobei wir ohne Beschränkung der Allgemeinheit annehmen, dass $\delta_i > \delta_j$ gilt. Dies ist in der Abbildung 2.10 illustriert.

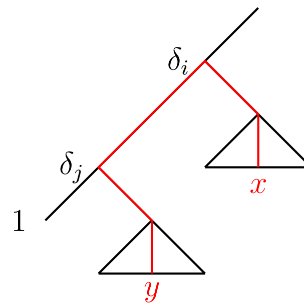



Abbildung 2.10: Skizze: Korrektheit der Abstände zweier Blätter in unterschiedlichen rekursiv konstruierten Teilbäumen

Nach Konstruktion gilt: $\delta_i = d_{1x}$ und $\delta_j = d_{1y}$. Mit Hilfe der ultrametrischen Dreiecksungleichung folgt:

$$\begin{aligned} d_{xy} &\leq \max\{d_{1x}, d_{1y}\} \\ &= \max\{\delta_i, \delta_j\} \\ &= \delta_i \\ &= d_{1x} \\ &\leq \max\{d_{1y}, d_{yx}\} \\ &\quad \text{da } d_{1y} = \delta_j < \delta_i = d_{1x} \\ &= d_{xy} \end{aligned}$$

Daraus folgt also $d_{xy} \leq \delta_i \leq d_{xy}$ und somit $\delta_i = d_{xy}$. Damit ist der Beweis abgeschlossen. ■

Aus dem Beweis folgt unter Annahme der strengen Monotonie (d.h. für strenge ultrametrische Bäume), dass der konstruierte ultrametrische Baum eindeutig ist. Die Konstruktion des ultrametrischen Baumes ist ja bis auf die Umordnung der Kinder eines Knoten eindeutig festgelegt.

**Korollar 2.13** *Sei $D$ eine streng ultrametrische Matrix, dann ist der zugehörige strenge ultrametrische Baum eindeutig.*

Für nicht-strenge ultrametrische Bäume kann man sich überlegen, wie die Knoten mit gleicher Markierung umgeordnet werden können, so dass der Baum ein ultrametrischer bleibt. Bis auf diese kleinen Umordnungen sind auch nicht-streng ultrametrische Bäume im Wesentlichen eindeutig.

### 2.2.4 Konstruktion ultrametrischer Bäume

Wir versuchen jetzt aus dem Beweis der Existenz eines ultrametrischen Baumes einen effizienten Algorithmus zur Konstruktion eines ultrametrischen Baumes zu entwerfen und dessen Laufzeit zu analysieren.

Wir erinnern noch einmal an die Partition von $[2:n]$ durch $D_i = \{\ell : d_{1\ell} = \delta_i\}$ mit $n_i := |D_i|$. Dabei war $\{d_{11}, \ldots, d_{1n}\} = \{\delta_1, \ldots, \delta_k\}$, wobei $\delta_1 < \cdots < \delta_k$. Wir erinnern hier auch noch einmal an Skizze der Konstruktion des ultrametrischen Baumes, wie in Abbildung 2.11.

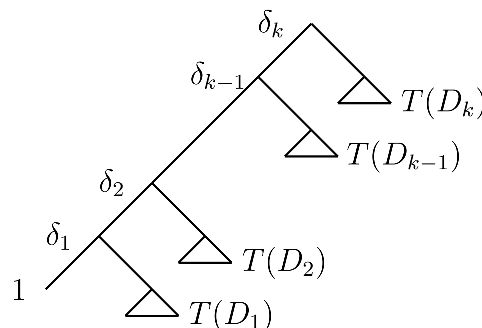



Abbildung 2.11: Skizze: Konstruktion eines ultrametrischen Baumes

Daraus ergibt sich der erste naive Algorithmus, der in Abbildung 2.12 aufgelistet ist. Da jeder Schritt Zeitbedarf $O(n \log(n))$ und es maximal $n$ rekursive Aufrufe geben

1. Sortiere die Menge $\{d_{11}, \ldots, d_{1n}\}$ und bestimme anschließend $\delta_1, \ldots, \delta_k$ mit $\{\delta_1, \ldots, \delta_k\} = \{d_{11}, \ldots, d_{1n}\}$ und partitioniere $[2:n] = \uplus_{i=1}^{k} D_i$. $O(n \log(n))$
2. Bestimme für $D_1, \ldots D_k$ ultrametrische Bäume $T(D_1), \ldots, T(D_k)$ mittels Rekursion. $\sum_{i=1}^{k} T(n_j)$
3. Setze die Teillösungen und den Pfad von der Wurzel zu Blatt 1 zur Gesamtlösung zusammen. $O(k) = O(n)$

Abbildung 2.12: Algorithmus: Naive Konstruktion eines ultrametrischen Baumes

kann (mit jedem rekursiven Aufruf wird ein Knoten des ultrametrischen Baumes explizit konstruiert), ist die Laufzeit insgesamt $O(n^2 \log(n))$.

Wir werden jetzt noch einen leicht modifizierten Algorithmus vorstellen, der eine Laufzeit von nur $O(n^2)$ besitzt. Dies ist optimal, da die Eingabe, die gegebene Distanzmatrix, bereits eine Größe von $\Theta(n^2)$ besitzt.

Dazu beachten wir, dass der aufwendige Teil des Algorithmus das Sortieren der Elemente in $\{d_{11}, \ldots, d_{1n}\}$ ist. Insbesondere ist dies sehr teuer, wenn es nur wenige verschieden Elemente in dieser Menge gibt, da dann auch nur entsprechend wenig rekursive Aufrufe folgen.

Daher werden wir zuerst feststellen, wie viele verschiedene Elemente es in der Menge $\{d_{11}, \ldots, d_{1n}\}$ gibt und bestimmen diese. Die paarweise verschiedenen Elemente dieser Menge können wir in einer linearen Liste (oder auch in einem balancierten Suchbaum) aufsammeln. Dies lässt sich in Zeit $O(k \cdot n)$ (bzw. in Zeit $O(n \log(k))$ bei Verwendung balancierter Bäume) implementieren. Anschließend müssen wir nur noch $k$ Elemente sortieren. Der Algorithmus selbst ist in Abbildung 2.13 aufgelistet.

Damit erhalten wir als Rekursionsgleichung für diesen modifizierten Algorithmus:

$$T(n) = d \cdot k \cdot n + \sum_{i=1}^{k} T(n_i)$$

mit $\sum_{i=1}^{k} n_i = n - 1$, wobei $n_i \geq 1$ für $i \in [1:n]$, und einer geeignet gewählten Konstanten $d$. Hierbei ist zu beachten, dass $O(k \log(k)) = O(k \cdot n)$ ist, da ja $k < n$ ist.

**Lemma 2.14** *Ist $D$ eine ultrametrische $n \times n$-Matrix, dann kann der zugehörige ultrametrische Baum in Zeit $O(n^2)$ konstruiert werden.*

**Beweis:** Es ist nur noch zu zeigen, dass $T(n) \leq c \cdot n^2$ für eine geeignet gewählte Konstante $c$ gilt. Wir wählen $c$ jetzt so, dass zum einen $c \geq 2d$ und zum anderen

1. Bestimme zuerst $k = |\{d_{11}, \ldots, d_{1n}\}|$ und $\{\delta_1, \ldots, \delta_k\} = \{d_{11}, \ldots, d_{1n}\}$.

   Dies kann mit Hilfe linearer Listen in Zeit $O(k \cdot n)$ erledigt werden. Mit Hilfe balancierter Bäume können wir dies sogar in Zeit $O(n \log(k))$ realisieren.

2. Sortiere die $k$ paarweise verschiedenen Werte $\{\delta_1, \ldots, \delta_k\}$.

   Dies kann in Zeit $O(k \log(k)$ erledigt werden.

3. Bestimme die einzelnen Teilbäume $T(D_i)$ rekursiv.

4. Setze die Teillösungen und den Pfad von der Wurzel zu Blatt 1 zur Gesamtlösung zusammen.

   Dies lässt sich wiederum in Zeit $O(k)$ realisieren.

Abbildung 2.13: Algorithmus: Konstruktion eines ultrametrischen Baumes

$T(n) \leq c \cdot n^2$ für alle $n \leq 3$ gilt. Den Beweis selbst führen wir mit vollständiger Induktion über $n$.

**Induktionsanfang ($n \leq 3$):** Nach Wahl von $c$ gilt dies offensichtlich.

**Induktionsschritt ($\rightarrow n$):** Es gilt dann nach Konstruktion des Algorithmus mit $n = \sum_{i=1}^{k} n_i$:

$$
\begin{aligned}
T(n) &\leq d \cdot k \cdot n + \sum_{i=1}^{k} T(n_i) \\
& \quad \text{nach Induktionsvoraussetzung ist } T(n_i) \leq c \cdot n_i^2 \\
&\leq d \cdot k \cdot n + \sum_{i=1}^{k} c \cdot n_i^2 \\
&= d \cdot k \cdot n + c \sum_{i=1}^{k} n_i(n - n + n_i) \\
&= d \cdot k \cdot n + c \sum_{i=1}^{k} n_i \cdot n - c \sum_{i=1}^{k} n_i(n - n_i) \\
& \quad \text{da } x(c-x) > 1(c-1) \text{ für } x \in [1:c-1] \text{, siehe auch Abbildung 2.14} \\
&\leq d \cdot k \cdot n + cn^2 - c \sum_{i=1}^{k} 1(n-1)
\end{aligned}
$$

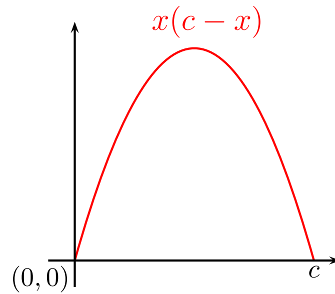



Abbildung 2.14: Skizze: Die Funktion $[0,c] \to \mathbb{R}_+ : x \mapsto x(c-x)$

$$\begin{aligned}
&\leq \quad d \cdot k \cdot n + c \cdot n^2 - c \cdot k(n-1) \\
&\quad\quad \text{da } c \geq 2d \\
&\leq \quad d \cdot k \cdot n + c \cdot n^2 - 2d \cdot k(n-2) \\
&\quad\quad \text{da } 2(n-1) \geq n \text{ für } n \geq 3 \\
&\leq \quad d \cdot k \cdot n + c \cdot n^2 - d \cdot k \cdot n \\
&= \quad c \cdot n^2.
\end{aligned}$$

Damit ist der Induktionsschluss vollzogen und der Beweis beendet. ■

## 2.3 Additive Distanzen und Bäume

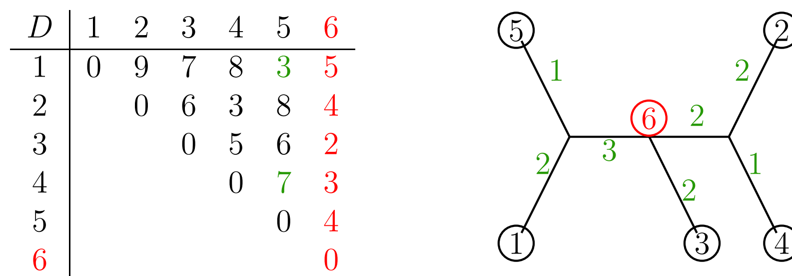
Leider sind nicht alle Distanzmatrizen ultrametrisch. Wir wollen jetzt ein größere Klasse von Matrizen vorstellen, zu denen sich evolutionäre Bäume konstruieren lassen, sofern wir auf eine explizite Wurzel in diesen Bäumen verzichten können.

### 2.3.1 Additive Bäume

Zunächst einmal definieren wir, was wir unter additiven Bäumen, d.h. evolutionären Bäumen ohne Wurzel, verstehen wollen.

**Definition 2.15** *Sei $D$ eine $n \times n$-Distanzmatrix. Sei $T$ ein Baum mit mindestens $n$ Knoten und positiven Kantengewichten, wobei einige Knoten bijektiv mit Werten aus $[1 : n]$ markiert sind. Dann ist $T$ ein* additiver Baum *für $D$, wenn der Pfad vom Knoten mit Markierung $i$ zum Knoten mit Markierung $j$ in $T$ das Gewicht $d_{ij}$ besitzt.*

Wie bei den ultrametrischen Bäumen besitzt auch nicht jede Distanzmatrix einen additiven Baum. Des Weiteren sind nicht markierte Blätter in einem additiven Baum überflüssig, da jeder Pfad innerhalb eines Baum nur dann ein Blatt berührt, wenn dieses ein Endpunkt des Pfades ist. Daher nehmen wir im Folgenden ohne Beschränkung der Allgemeinheit an, dass ein additiver Baum nur markierte Blätter besitzt. In der folgenden Abbildung 2.15 ist noch einmal ein Beispiel einer Matrix samt ihres zugehörigen additiven Baumes angegeben.



| $D$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 0 | 9 | 7 | 8 | 3 | 5 |
| 2 | | 0 | 6 | 3 | 8 | 4 |
| 3 | | | 0 | 5 | 6 | 2 |
| 4 | | | | 0 | 7 | 3 |
| 5 | | | | | 0 | 4 |
| 6 | | | | | | 0 |



Gewicht des Pfades zwischen 5 und 4: $\Rightarrow 1 + 3 + 2 + 1 = 7$
Gewicht des Pfades zwischen 1 und 5: $\Rightarrow 2 + 1 = 3$

Abbildung 2.15: Beispiel: Eine Matrix und der zugehörige additive Baum

**Definition 2.16** *Sei $D$ eine Distanzmatrix. Besitzt $D$ einen additiven Baum, so heißt $D$ eine* additive Matrix. *Ein additiver Baum heißt* kompakt, *wenn alle Knoten markiert sind (insbesondere auch die inneren Knoten). Ein additiver Baum heißt* extern, *wenn nur Blätter markiert sind.*

In der Abbildung 2.15 ist der Baum ohne Knoten 6 (und damit die Matrix ohne Zeile und Spalte 6) ein externer additiver Baum. Durch Hinzufügen von zwei Markierungen (und zwei entsprechenden Spalten und Zeilen in der Matrix) könnte dieser Baum zu einem kompakten additiven Baum gemacht werden.

**Lemma 2.17** *Sei $D$ eine additive Matrix, dann induziert $D$ eine Metrik.*

**Beweis:** Da $D$ eine Distanzmatrix ist, gelten die Definitheit und Symmetrie unmittelbar. Die Dreiecksungleichung sieht man leicht, wenn man sich die entsprechenden Pfade im zu $D$ gehörigen additiven Baum betrachtet. ∎

Die Umkehrung gilt übrigens nicht, wie wir später noch zeigen werden.

**13. Mai**

### 2.3.2 Charakterisierung additiver Bäume

Wir wollen nun eine Charakterisierung additiver Matrizen angeben, mit deren Hilfe sich auch gleich ein zugehöriger additiver Baum konstruieren lässt. Zunächst einmal zeigen wir, dass ultrametrische Bäume spezielle additive Bäume sind.

**Lemma 2.18** *Sei $D$ eine additive Matrix. $D$ ist genau dann ultrametrisch, wenn es einen additiven Baum $T$ für $D$ gibt, so dass es in $T$ einen Knoten $v$ (zentraler Knoten) gibt, der zu allen markierten Knoten denselben Abstand besitzt.*

**Beweis:** **⇒:** Sei $T$ ein ultrametrischer Baum für $D$ mit Knotenmarkierungen $\mu$. Wir erhalten daraus einen additiven Baum $T'$, indem wir als Kantengewicht einer Kante $(v, w)$ folgendes wählen:

$$\gamma(v, w) = \frac{1}{2}(\mu(v) - \mu(w)).$$

Man rechnet jetzt leicht nach, dass für zwei Blätter $i$ und $j$ sowohl das Gewicht des Weges von $i$ nach $\mathrm{lca}(i, j)$ als auch das Gewicht von $j$ nach $\mathrm{lca}(i, j)$ gerade $\frac{1}{2} \cdot d_{ij}$ beträgt. Die Länge des Weges von $i$ nach $j$ beträgt daher im additiven Baum wie gefordert $d_{ij}$.

Falls einen Knoten mit Grad 2 in einem solchen additiven Baum stören (wie sie beispielweise von der Wurzel konstruiert werden), der kann diese, wie in Abbildung 2.16 illustriert, eliminieren. In dieser Abbildung stellt der rote Knoten den zentralen Knoten des additiven Baumes dar.

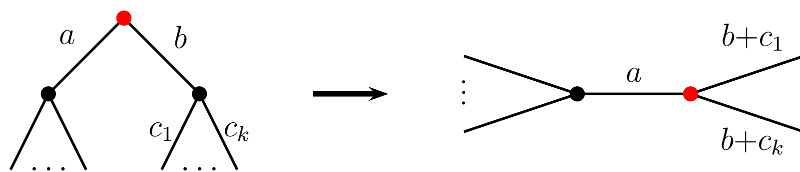



Abbildung 2.16: Skizze: Elimination von Knoten mit Grad 2

**⇐:** Sei $D$ additiv und sei $v$ der Knoten des additiven Baums $T$, der von allen Blättern den gleichen Abstand hat. Man überlegt sich leicht, dass $T$ dann extern additiv sein muss.

Betrachtet man $v$ als Wurzel, so gilt für beliebige Blätter $i$, $j$, dass der Abstand zwischen dem kleinsten gemeinsamen Vorfahr $\ell$ für beide Blätter gleich ist. (Da der Abstand $d_{\ell v}$ fest ist, wäre andernfalls der Abstand der Blätter zu $v$ nicht gleich). Wir

zeigen jetzt, dass für drei beliebige Blätter $i, j, k$ die ultrametrische Dreiecksungleichung $d_{ik} \leq \max\{d_{ij}, d_{jk}\}$ gilt. Sei dazu $\mathrm{lca}(i, j)$ der kleinste gemeinsame Vorfahre von $i, j$.

Falls $\mathrm{lca}(i, j) = \mathrm{lca}(i, k) = \mathrm{lca}(j, k)$ die gleichen Knoten sind, so ist $d_{ik} = d_{ij} = d_{jk}$ und die Dreiecksungleichung gilt mit Gleichheit.

Daher sei jetzt ohne Beschränkung der Allgemeinheit $\mathrm{lca}(i, j) \neq \mathrm{lca}(i, k)$ und dass $\mathrm{lca}(i, k)$ näher an der Wurzel liegt. Da $\mathrm{lca}(i, j)$ und $\mathrm{lca}(i, k)$ auf dem Weg von $i$ zur Wurzel liegen, gilt entweder $\mathrm{lca}(j, k) = \mathrm{lca}(j, i)$ oder $\mathrm{lca}(j, k) = \mathrm{lca}(i, k)$.

Sei ohne Beschränkung der Allgemeinheit $\mathrm{lca}(j, k) = \mathrm{lca}(i, k) = \mathrm{lca}(i, j, k)$. Dann gilt:

$$\begin{aligned}
d_{ij} &= w(i, \mathrm{lca}(i, j)) + w(j, \mathrm{lca}(i, j)) \\
&= 2 \cdot w(j, \mathrm{lca}(i, j)), \\[1ex]
d_{ik} &= w(i, \mathrm{lca}(i, j)) + w(\mathrm{lca}(i, j), \mathrm{lca}(i, j, k)) + w(\mathrm{lca}(i, j, k), k) \\
&= 2 \cdot w(\mathrm{lca}(i, j, k), k), \\[1ex]
d_{jk} &= w(j, \mathrm{lca}(i, j)) + w(\mathrm{lca}(i, j), \mathrm{lca}(i, j, k)) + w(\mathrm{lca}(i, j, k), k) \\
&= 2 \cdot w(\mathrm{lca}(i, j, k), k)
\end{aligned}$$

Daher gilt unter anderem $d_{ij} \leq d_{ik}$, $d_{ik} \leq d_{jk}$ und $d_{jk} \leq d_{ik}$. Somit ist die Dreiecksungleichung immer erfüllt. ■

Wie können wir nun für eine Distanzmatrix entscheiden, ob sie additiv ist, und falls ja, beschreiben, wie der zugehörige additive Baum aussieht? Im vorherigen Lemma haben wir gesehen, wie wir das Problem auf ultrametrische Matrizen zurückführen können.

Wir wollen dies noch einmal mir einer anderen Charakterisierung tun. Wir betrachten zuerst die gegebene Matrix $D$. Diese ist genau dann additiv, wenn sie einen additiven Baum $T_D$ besitzt. Wenn wir diesen Baum wurzeln und die Kanten zu den Blättern so verlängern, dass alle Pfade von der Wurzel zu den Blättern gleiches Gewicht besitzen, dann ist $T'_D$ ultrametrisch. Daraus können wir dann eine Distanzmatrix $D(T'_D)$ ablesen, die ultrametrisch ist, wenn $D$ additiv ist. Dies ist in der folgenden Abbildung 2.17 schematisch dargestellt.

Wir wollen also die gegebene Matrix $D$ so modifizieren, dass daraus eine ultrametrische Matrix wird. Wenn diese Idee funktioniert, können wir eine Matrix auf Additivität hin testen, indem wir die zugehörige, neu konstruierte Matrix auf Ultrametrik hin testen. Für Letzteres haben wir ja bereits einen effizienten Algorithmus kennen gelernt.

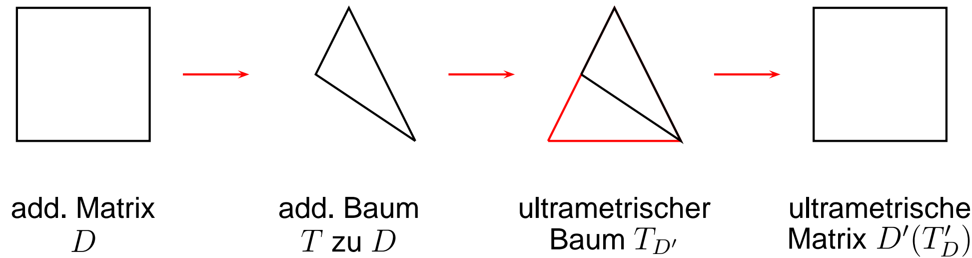
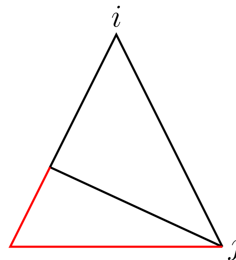



Abbildung 2.17: Skizze:

Sei im Folgenden $D$ eine additive Matrix und $d_{ij}$ ein maximaler Eintrag, d.h.

$$d_{ij} \geq \max \{d_{k\ell} \ : \ k, \ell \in [1 : n]\} .$$

Weiter sei $T_D$ der additive Baum zu $D$. Wir wurzeln jetzt diesen Baum am Knoten $i$. Man beachte, dass $i$ ein Blatt ist. Wir kommen später noch darauf zurück, wie wir dafür sorgen, dass $i$ ein Blatt bleibt. Dies ist in der folgenden Abbildung 2.18 illustriert.




Alle Blätter auf denselben Abstand bringen

Abbildung 2.18: Skizze: Wurzeln von $T_D$ am Blatt $i$

Unser nächster Schritt besteht jetzt darin, alle Blätter (außer $i$) auf denselben Abstand zur neuen Wurzel zu bringen. Wir betrachten dazu jetzt ein Blatt $k$ des neuen gewurzelten Baumes. Wir versuchen die zu $k$ inzidente Kante jetzt so zu verlängern, dass der Abstand von $k$ zur Wurzel $i$ auf $d_{ij}$ anwächst. Dazu setzen wir das Kantengewicht auf $d_{ij}$ und verkürzen es um das Gewicht des restlichen Pfades von $k$ zu $i$, d.h. um $(d_{ik} - d)$, wobei $d$ das Gewicht der zu $k$ inzidenten Kante ist. Dies ist in Abbildung 2.19 noch einmal illustriert. War der Baum vorher additiv, so ist er jetzt ultrametrisch, wenn wir als Knotenmarkierung jetzt das Gewicht des Pfades einen Knotens zu einem seiner Blätter (die alle gleich sein müssen) wählen.

Somit haben wir aus einem additiven einen ultrametrischen Baum gemacht. Jedoch haben wir dazu den additiven Baum benötigt, den wir eigentlich erst konstruieren

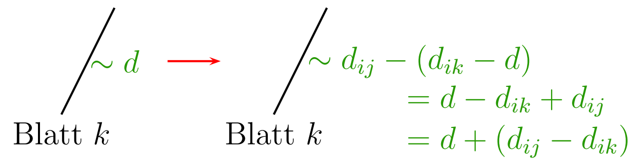



Abbildung 2.19: Skizze: Verlängern der zu Blättern inzidenten Kanten

wollen. Es stellt sich nun die Frage, ob wir die entsprechende ultrametrische Matrix aus der additiven Matrix direkt ohne Kenntnis des zugehörigen additiven Baumes berechnen können. Betrachten wir dazu noch einmal zwei Blätter im additiven Baum und versuchen den entsprechenden Abstand im ultrametrischen Baum zu berechnen. Dazu betrachten wir die Abbildung 2.20.

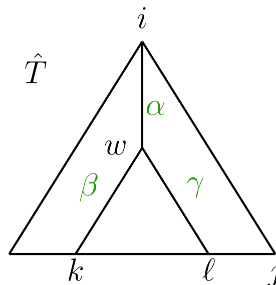



Abbildung 2.20: Skizze: Abstände im gewurzelten additiven Baum $\hat{T}$

Seien $k$ und $\ell$ zwei Blätter, für die wir den Abstand in der entsprechenden ultrametrischen Matrix bestimmen wollen. Sei $w$ der niedrigste gemeinsame Vorfahre von $k$ und $\ell$ im gewurzelten additiven Baum $\hat{T}$ und $\alpha$, $\beta$ bzw. $\gamma$ die Abstände im gewurzelten additiven Baum $\hat{T}$ zwischen der Wurzel und $w$, $w$ und $k$ bzw. $w$ und $\ell$. Es gilt dann

$$\begin{aligned} \beta &= d_{ik} - \alpha \\ \gamma &= d_{il} - \alpha \end{aligned}$$

Im ultrametrischen Baum $T'_D$ gilt dann:

$$\begin{aligned} d_{T'}(w,k) &= d_{T'}(w,\ell) \\ &= \beta + (d_{ij} - d_{ik}) \\ &= \gamma + (d_{ij} - d_{i\ell}). \end{aligned}$$

Damit gilt:

$$\begin{aligned} d_{T'}(w,k) &= \beta + d_{ij} - d_{ik} \\ &= d_{ik} - \alpha + d_{ij} - d_{ik} \\ &= d_{ij} - \alpha \end{aligned}$$

und analog:

$$\begin{aligned} d_{T'}(w,\ell) &= \gamma + d_{ij} - d_{i\ell} \\ &= d_{i\ell} - \alpha + d_{ij} - d_{i\ell} \\ &= d_{ij} - \alpha. \end{aligned}$$

Wir müssen jetzt nur noch $\alpha$ bestimmen. Aus der Skizze in Abbildung 2.20 folgt sofort, wenn wir die Gewichte der Pfade von $i$ nach $k$ sowie $\ell$ addieren und davon das Gewicht des Pfades von $k$ nach $\ell$ subtrahieren:

$$2\alpha = d_{ik} + d_{i\ell} - d_{k\ell}.$$

Daraus ergibt sich:

$$\begin{aligned} d_{T'}(w,k) &= d_{ij} - \frac{1}{2}(d_{ik} + d_{i\ell} - d_{k\ell}), \\ d_{T'}(w,\ell) &= d_{ij} - \frac{1}{2}(d_{ik} + d_{i\ell} - d_{k\ell}). \end{aligned}$$

Somit können wir jetzt die ultrametrische Matrix $D'$ direkt aus der additiven Matrix $D$ berechnen:

$$d'_{k\ell} := d_{T'}(w,k) = d_{T'}(w,\ell) = d_{ij} - \frac{1}{2}(d_{i\ell} + d_{ik} - d_{k\ell}).$$

Wir müssen uns jetzt nur noch überlegen, dass dies wirklich eine ultrametrische Matrix ist, da wir den additiven Baum ja am Blatt $i$ gewurzelt haben. Damit würden wir sowohl ein Blatt verlieren, nämlich $i$, als auch keinen echten ultrametrischen Baum generieren, da dessen Wurzel nur ein Kind anstatt mindestens zweier besitzt. In Wirklichkeit wurzeln wir den additiven Baum am zu $i$ adjazenten Knoten, wie in Abbildung 2.21 illustriert. Damit erhalten wir einen echten ultrametrischen Baum.

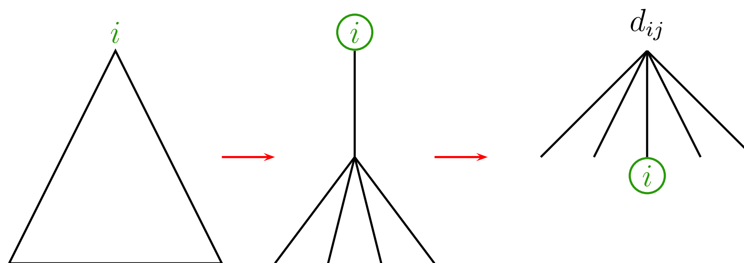



Abbildung 2.21: Skizze: Wirkliches Wurzeln des additiven Baumes

Damit haben wir das folgende Lemma bewiesen.

**Lemma 2.19** *Sei $D$ eine additive Matrix, deren maximaler Eintrag $d_{ij}$ ist, dann ist $D'$ mit*

$$d'_{kl} = d_{ij} - \frac{1}{2}(d_{i\ell} + d_{ik} - d_{kl\ell})$$

*eine ultrametrische Matrix.*

Es wäre schön, wenn auch die umgekehrte Richtung gelten würde, nämlich, dass wenn $D'$ ultrametrisch ist, dass dann bereits $D$ additiv ist. Dies ist leider nicht der Fall, auch wenn dies in vielen Lehrbüchern und Skripten fälschlicherweise behauptet wird. Wir geben hierfür ein Gegenbeispiel in Abbildung 2.22. Man sieht leicht, dass

| $D$ | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 0 | 8 | 4 |
| 2 | | 0 | 2 |
| 3 | | | 0 |

| $D'$ | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 0 | 8 | 8 |
| 2 | | 0 | 3 |
| 3 | | | 0 |

$d'_{12} = 8 - \frac{1}{2}(0 + 8 - 8) = 8$
$d'_{13} = 8 - \frac{1}{2}(0 + 4 - 4) = 8$
$d'_{23} = 8 - \frac{1}{2}(8 + 4 - 2) = 3$

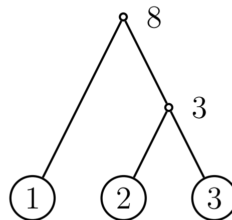



Abbildung 2.22: Gegenbeispiel: $D$ nicht additiv, aber $D'$ ultrametrisch

.

$D'$ ultrametrisch ist. $D$ ist hingegen nicht additiv, weil $d(1,2) = 8$, aber

$$d(1,3) + d(3,2) = 4 + 2 = 6 < 8$$

gilt. Im Baum muss also der Umweg über 3 größer sein als der direkte Weg, was nicht sein kann (siehe auch Abbildung 2.23), denn im additiven Baum gilt die normale Dreiecksungleichung (siehe Lemma 2.17).

Dennoch können wir mit einer weiteren Zusatzbedingung dafür sorgen, dass das Lemma in der von uns gewünschten Weise gerettet werden kann.

**Lemma 2.20** *Sei $D$ eine Distanzmatrix und sei $d_{ij}$ ein maximaler Eintrag von $D$. Weiter sei $D'$ durch $d'_{k\ell} = d_{ij} - \frac{1}{2}(d_{ik} + d_{i\ell} - d_{k\ell})$ definiert. Wenn $D'$ eine ultrametrische Matrix ist und wenn für jedes Blatt $b$ im zugehörigen ultrametrischen Baum $T(D')$ für das Gewicht $\gamma$ der zu $b$ inzidenten Kante gilt: $\gamma \geq (d_{ij} - d_{bi})$, dann ist $D$ additiv.*

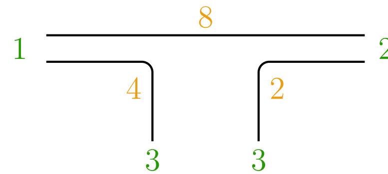



Abbildung 2.23: Gegenbeispiel: „Unmöglicher" additiver Baum

**Beweis:** Sei $T' := T(D')$ ein ultrametrischer Baum für $D'$. Weiter sei $(v, w) \in E(T')$ und es seien $p, q, r, s$ Blätter von $T'$, so dass $\text{lca}(p, q) = v$ und $\text{lca}(r, s) = w$. Dies ist in Abbildung 2.24 illustriert. Hierbei ist $q$ zweimal angegeben, da a priori nicht klar ist, ob $q$ ein Nachfolger von $w$ ist oder nicht. Wir definieren dann das Kantengewicht

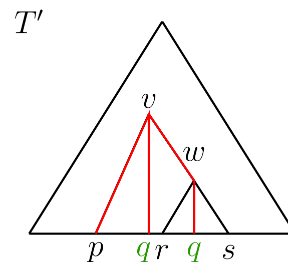



Abbildung 2.24: Skizze: Kante $(v, w)$ in $T'$

von $(v, w)$ durch:

$$\gamma(v, w) = d'_{pq} - d'_{rs}.$$

Damit ergibt sich für

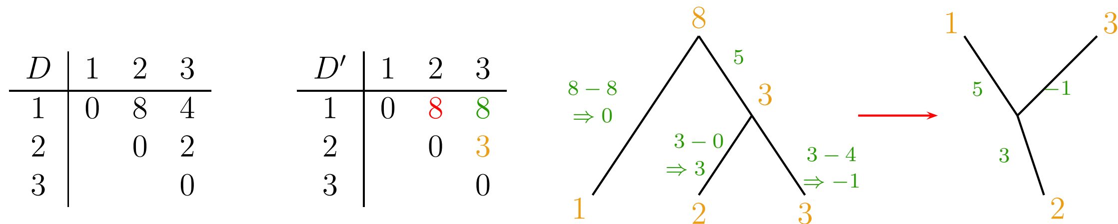
$$\begin{aligned} d_{T'}(k, \ell) &= 2 \cdot d'_{k,\ell} \\ &= 2(d_{ij} - \frac{1}{2}(d_{ik} + d_{i\ell} - d_{k\ell})) \\ &= 2d_{ij} - d_{ik} - d_{i\ell} + d_{k\ell} \end{aligned}$$

Wenn wir jetzt für jedes Blatt $b$ das Gewicht der inzidenten Kante um $d_{ij} - d_{bi}$ erniedrigen, erhalten wir einen neuen additiven Baum $T$. Da nach Voraussetzung, das Gewicht einer solchen zu $b$ inzidenten Kante größer als $d_{ij} - d_{bi}$ ist, bleiben die Kantengewichte von $T$ positiv. Weiterhin gilt in $T$:

$$\begin{aligned} d_T(k, \ell) &= d_{T'}(k, \ell) - (d_{ij} - d_{ik}) - (d_{ij} - d_{i\ell}) \\ &= (2d_{ij} - d_{ik} - d_{i\ell} + d_{k\ell}) - d_{ij} + d_{ik} - d_{ij} + d_{i\ell} \\ &= d_{k\ell}. \end{aligned}$$

Somit ist $T$ ein additiver Baum für $D$ und das Lemma ist bewiesen. ∎

Gehen wir noch einmal zurück zu unserem Gegenbeispiel, das besagte, dass die Ultrametrik von $D'$ nicht ausreicht, um die Additivität von $D$ zu zeigen. Wir schauen einmal was hier beim Kürzen der Gewichte von zu Blättern inzidenten Kanten passieren kann. Dies ist in Abbildung 2.25 illustriert. Wir sehen hier, dass der Baum zwar die gewünschten Abstände besitzt, jedoch negative Kantengewichte erzeugt, die bei additiven Bäumen nicht erlaubt sind.



| $D$ | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 0 | 8 | 4 |
| 2 | | 0 | 2 |
| 3 | | | 0 |

| $D'$ | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 0 | 8 | 8 |
| 2 | | 0 | 3 |
| 3 | | | 0 |



Abbildung 2.25: Gegenbeispiel: $D$ nicht additiv und $D'$ ultrametrisch (Fortsetzung)

### 2.3.3 Algorithmus zur Erkennung additiver Matrizen

Aus der Charakterisierung der additiven Matrizen mit Hilfe ultrametrischer Matrizen lässt sich der folgende Algorithmus zur Erkennung additiver Matrizen und der Konstruktion zugehöriger additiver Bäume herleiten. Dieser ist in Abbildung 2.26 aufgelistet. Es lässt sich leicht nachrechnen, dass dieser Algorithmus eine Laufzeit von $O(n^2)$ besitzt.

1. Konstruiere aus der gegebenen $n \times n$-Matrix $D$ eine neue $n \times n$-Matrix $D'$ mittels $d'_{k\ell} = d_{ij} - \frac{1}{2}(d_{ik} + d_{i\ell} - d_{k\ell})$, wobei $d_{ij}$ ein maximaler Eintrag von $D$ ist.

2. Teste $D'$ auf Ultrametrik. Ist $D'$ nicht ultrametrisch, dann ist $D$ nicht additiv.

3. Konstruiere den zu $D'$ gehörigen ultrametrischen Baum $T'$.

4. Teste, ob sich die Kantengewichte für jedes Blatt $b$ um $d_{ij} - d_{ib}$ erniedrigen lässt. Falls dies nicht geht, ist $D$ nicht additiv, andernfalls erhalten wir einen additiven Baum $T$ für $D$.

Abbildung 2.26: Algorithmus: Erkennung additiver Matrizen

Wir müssen uns nur noch kurz um die Korrektheit kümmern. Nach Lemma 2.19 wissen wir, dass in Schritt 2 die richtige Entscheidung getroffen wird. Nach Lemma 2.20

wird auch im Schritt 4 die richtige Entscheidung getroffen. Also ist der Algorithmus korrekt. Fassen wir das Ergebnis noch zusammen.

**Theorem 2.21** *Es kann in Zeit $O(n^2)$ entschieden werden, ob eine gegebene $n \times n$-Distanzmatrix additiv ist oder nicht. Falls die Matrix additiv ist, kann der zugehörige additive Baum in Zeit $O(n^2)$ konstruiert werden.*

### 2.3.4 4-Punkte-Bedingung

Zum Abschluss wollen wir noch eine andere Charakterisierung von additiven Matrizen angeben, die so genannte *4-Punkte-Bedingung von Buneman*.

**Lemma 2.22 (Bunemans 4-Punkte-Bedingung)** *Eine $n \times n$-Distanzmatrix $D$ ist genau dann additiv, wenn für je vier Punkte $i, j, k, \ell \in [1 : n]$ mit*

$$d_{i\ell} + d_{jk} = \min\{d_{ij} + d_{k\ell}, d_{ik} + d_{j\ell}, d_{i\ell} + d_{jk}\}$$

*gilt, dass $d_{ij} + d_{k\ell} = d_{ik} + d_{j\ell}$.*

Diese 4-Punkte-Bedingung ist in Abbildung 2.27 noch einmal illustriert

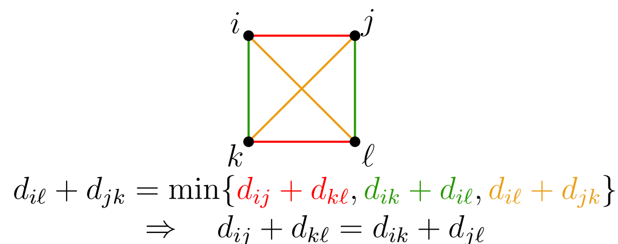



Abbildung 2.27: Skizze: 4-Punkte-Bedingung

**Beweis:** **⇒:** Sei $T$ ein additiver Baum für $D$. Wir unterscheiden jetzt drei Fälle, je nachdem, wie die Pfade in $T$ verlaufen.

**Fall 1:** Im ersten Fall nehmen wir an, die Pfade von $i$ nach $j$ und $k$ nach $\ell$ knotendisjunkt sind. Dies ist in Abbildung 2.28 illustriert. Dann ist aber $d_{i\ell} + d_{jk}$ nicht minimal und somit ist nichts zu zeigen.

**Fall 2:** Im zweiten Fall nehmen wir an, die Pfade von $i$ nach $k$ und $j$ nach $\ell$ knotendisjunkt sind. Dies ist in Abbildung 2.29 illustriert. Dann ist jedoch ebenfalls $d_{i\ell} + d_{jk}$ nicht minimal und somit ist nichts zu zeigen.

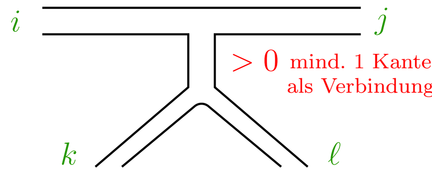



Abbildung 2.28: Skizze: Die Pfade $i \to j$ und $k \to \ell$ sind knotendisjunkt

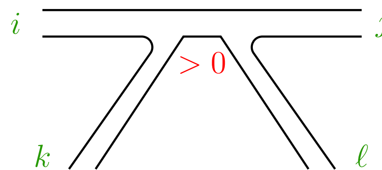



Abbildung 2.29: Skizze: Die Pfade $i \to k$ und $j \to \ell$ sind knotendisjunkt

**Fall 3:** Im dritten und letzten Fall nehmen wir an, die Pfade von $i$ nach $\ell$ und $j$ nach $k$ kantendisjunkt sind und sich daher höchstens in einem Knoten schneiden. Dies ist in Abbildung 2.30 illustriert. Nun gilt jedoch, wie man leicht der Abbildung 2.30

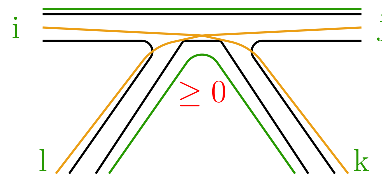



Abbildung 2.30: Skizze: Die Pfade $i \to \ell$ und $j \to k$ sind kantendisjunkt

entnehmen kann:

$$d_{ij} = d_{ik} + d_{j\ell} - d_{k\ell}$$

und somit

$$d_{ij} + d_{k\ell} = d_{ik} + d_{j\ell}.$$

**⇐:** Betrachte $D'$ mit $d'_{k\ell} := d_{ij} - \frac{1}{2}(d_{ik} + d_{i\ell} - d_{k\ell})$, wobei $d_{ij}$ ein maximaler Eintrag von $D$ ist. Es genügt zu zeigen, dass $D'$ ultrametrisch ist.

Betrachte $p, q, r \in [1:n]$ mit $d'_{pq} \leq d'_{pr} \leq d'_{qr}$. Es ist dann zu zeigen: $d'_{pr} = d'_{qr}$. Aus $d'_{pq} \leq d'_{pr} \leq d'_{qr}$ folgt dann:

$$2d_{ij} - d_{ip} - d_{iq} + d_{pq} \;\leq\; 2d_{ij} - d_{ip} - d_{ir} + d_{pr} \;\leq\; 2d_{ij} - d_{iq} - d_{ir} + d_{qr}.$$

Daraus folgt:

$$d_{pq} - d_{ip} - d_{iq} \;\leq\; d_{pr} - d_{ip} - d_{ir} \;\leq\; d_{qr} - d_{iq} - d_{ir}$$

und weiter

$$d_{pq} + d_{ir} \;\leq\; d_{pr} + d_{iq} \;\leq\; d_{qr} + d_{ip}.$$

Aufgrund der 4-Punkt-Bedingung folgt unmittelbar

$$d_{pr} + d_{iq} = d_{qr} + d_{ip}.$$

Nach Addition von $(2d_{ij} - d_{ir})$ folgt:

$$(2d_{ij} - d_{ir}) + d_{pr} + d_{iq} = (2d_{ij} - d_{ir}) + d_{qr} + d_{ip}.$$

Nach kurzer Rechnung folgt:

$$2d_{ij} - d_{ir} - d_{ip} + d_{pr} = 2d_{ij} - d_{ir} - d_{iq} + d_{qr}$$

und somit gilt

$$2d'_{pr} = 2d'_{qr}.$$

Also ist $D'$ nach Lemma 2.4 ultrametrisch und somit ist nach Lemma 2.20 $D$ additiv. ■

Mit Hilfe dieser Charakterisierung werden wir noch zeigen, dass es Matrizen gibt, die eine Metrik induzieren, aber keinen additiven Baum besitzen. Dieses Gegenbeispiel

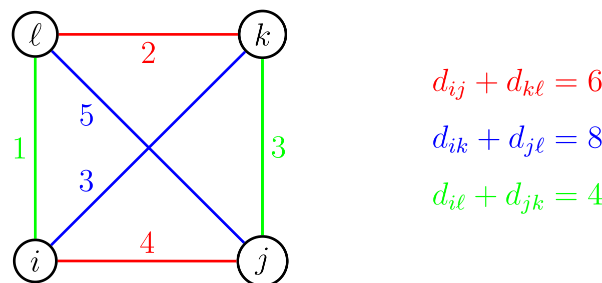



Abbildung 2.31: Gegenbeispiel: Metrische Matrix, die nicht additiv ist

ist in Abbildung 2.31 angegeben. Man sieht leicht, dass die 4-Punkte-Bedingung nicht gilt und somit die Matrix nicht additiv ist. Man überprüft leicht, dass für jedes Dreieck die Dreiecksungleichung gilt, die Abstände also der Definition einer Metrik genügen.

**15. Mai**

### 2.3.5 Charakterisierung kompakter additiver Bäume

In diesem Abschnitt wollen wir eine schöne Charakterisierung kompakter additiver Bäume angeben. Zunächst benötigen wir noch einige grundlegende Definitionen aus der Graphentheorie.

**Definition 2.23** *Sei $G = (V, E)$ ein ungerichteter Graph. Ein Teilgraph $G' \subset G$ heißt* aufspannend, *wenn $V(G') = V(G)$ und $G'$ zusammenhängend ist.*

Der aufspannende Teilgraph enthält also alle Knoten des aufgespannten Graphen. Nun können wir den Begriff eines Spannbaumes definieren.

**Definition 2.24** *Sei $G = (V, E)$ ein ungerichteter Graph. Ein Teilgraph $G' \subset G$ heißt* Spannbaum, *wenn $G'$ ein aufspannender Teilgraph von $G$ ist und $G'$ ein Baum ist.*

Für gewichtete Graphen brauchen wir jetzt noch das Konzept eines minimalen Spannbaumes.

**Definition 2.25** *Sei $G = (V, E, \gamma)$ ein gewichteter ungerichteter Graph und $T$ ein Spannbaum von $G$. Das* Gewicht des Spannbaumes *$T$ von $G$ ist definiert durch*

$$\gamma(T) := \sum_{e \in E(T)} \gamma(e).$$

*Ein Spannbaum $T$ für $G$ heißt* minimal, *wenn er unter allen möglichen Spannbäumen für $G$ minimales Gewicht besitzt, d.h.*

$$\gamma(T) \leq \min \left\{ \gamma(T') \; : \; T' \text{ ist ein Spannbaum von } G \right\}.$$

Im Folgenden werden wir der Kürze wegen einen minimalen Spannbaum oft auch mit MST (engl. *minimum spanning tree*) abkürzen.

**Definition 2.26** *Sei $D$ eine $n \times n$-Distanzmatrix. Dann ist $G(D) = (V, E)$ der zu $D$* gehörige gewichtete Graph, *wobei*

$$\begin{aligned} V &= [1:n], \\ E &= \binom{V}{2} := \left\{ \{v, w\} \; : \; v \neq w \in V \right\}, \\ \gamma(v, w) &= D(v, w). \end{aligned}$$

Nach diesen Definitionen kommen wir zu dem zentralen Lemma dieses Abschnittes, der kompakte additive Bäume charakterisiert.

**Theorem 2.27** *Sei $D$ eine $n \times n$-Distanzmatrix. Besitzt $D$ einen kompakten additiven Baum $T$, dann ist $T$ der minimale Spannbaum von $G(D)$ und ist eindeutig bestimmt.*

**Beweis:** Sei $T$ ein kompakter additiver Baum für $D$ (dieser muss natürlich nicht gewurzelt sein). Wir betrachten ein Knotenpaar $(x, y)$, das durch keine Kante in $T$

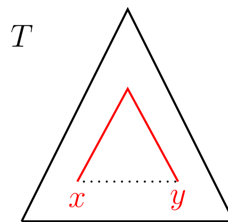



Abbildung 2.32: Skizze: Pfad von $x$ nach $y$ in $T$

verbunden ist. Sei $p = (v_0, v_1, \ldots, v_k)$ mit $v_0 = x$ und $v_k = y$ sowie $k \geq 2$ der Pfad von $x$ nach $y$ in $T$. $\gamma(p)$ entspricht $D(x, y)$, weil $T$ ein additiver Baum für $D$ ist. Da nach Definition einer Distanzmatrix alle Kantengewichte positiv sind und der Pfad aus mindestens zwei Kanten besteht, ist $\gamma(v_{i-1}, v_i) < D(x, y) = \gamma(x, y)$ für alle Kanten $(v_{i-1}, v_i)$ des Pfades $p$.

Sei jetzt $T'$ ein minimaler Spannbaum von $G(D)$. Wir nehmen jetzt an, dass die Kante $(x, y)$ eine Kante des minimalen Spannbaumes ist, d.h. $(x, y) \in E(T')$. Somit verbindet die Kante $(x, y)$ zwei Teilbäume von $T'$. Dies ist in Abbildung 2.32 illustriert.

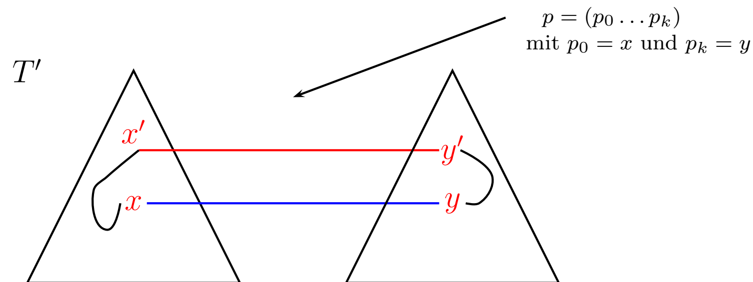



Abbildung 2.33: Skizze: Spannbaum $T'$ von $D(G)$

Da $(x, y)$ keine Kante im Pfad von $x$ nach $y$ im kompakten additiven Baum von $T$ ist, verbindet der Pfad $p$ die beiden durch Entfernen der Kante $(x, y)$ separierten

Teilbäume des minimalen Spannbaumes $T'$. Sei $(x', y')$ die erste Kante auf dem Pfad $p$ von $x$ nach $y$, die nicht innerhalb einer der beiden separierten Spannbäume verläuft. Wie wir oben gesehen haben, gilt $\gamma(x', y') < \gamma(x, y)$.

Wir konstruieren jetzt einen neuen Spannbaum $T''$ für $G(D)$ wie folgt:

$$\begin{aligned} V(T'') &= V(T') = V \\ E(T'') &= (E(T') \setminus \{\{x, y\}\}) \cup \{\{x', y\}\} \end{aligned}$$

Für das Gewicht des Spannbaumes $T''$ gilt dann:

$$\begin{aligned} \gamma(T'') &= \sum_{e \in E(T'')} \gamma(e) \\ &= \sum_{e \in E(T')} \gamma(e) - \gamma(x, y) + \gamma(x', y') \\ &= \sum_{e \in E(T')} \gamma(e) + \underbrace{(\gamma(x', y') - \gamma(x, y))}_{<0} \\ &< \gamma(T'). \end{aligned}$$

Somit ist $\gamma(T'') < \gamma(T)$ und dies liefert den gewünschten Widerspruch zu der Annahme, dass $T'$ ein minimaler Spannbaum von $G(D)$ ist.

Somit gilt für jedes Knotenpaar $(x, y)$ mit $(x, y) \notin E(T)$, dass dann die Kante $(x, y)$ nicht im minimalen Spannbaum von $G(D)$ sein kann, d.h. $(x, y) \notin T'$. Also ist eine Kante, die sich nicht im kompakten additiven Baum befindet, auch keine Kante des Spannbaums.

Dies gilt sogar für zwei verschiedene minimale Spannbäume für $G(D)$. Somit kann es nur einen minimalen Spannbaum geben. ■

### 2.3.6 Konstruktion kompakter additiver Bäume

Die Information aus dem vorherigen Lemma kann man dazu ausnutzen, um einen effizienten Algorithmus zur Erkennung kompakter additiver Matrizen zu entwickeln. Sei $D$ die Matrix, für den der kompakte additive Baum konstruiert werden soll, und somit $G(D) = (V, E, \gamma)$ der gewichtete Graph, für den der minimale Spannbaum berechnet werden soll.

Wir beginnen mit der Knotenmenge $V' = \{v\}$ für eine beliebiges $v \in V$ und der leeren Kantenmenge $E' = \emptyset$. Der Graph $(V', E')$ soll letztendlich der gesuchte minimale Spannbaum werden. Wir verwenden hier Prims Algorithmus zur Konstruktion eines minimalen Spannbaumes, der wieder ein Greedy-Algorithmus sein wird. Wir

versuchen die Menge $V'$ in jedem Schritt um einen Knoten aus $V \setminus V'$ zu erweitern. Von allen solchen Kandidaten wählen wir eine Kante minimalen Gewichtes. Dies ist schematisch in Abbildung 2.34 dargestellt.

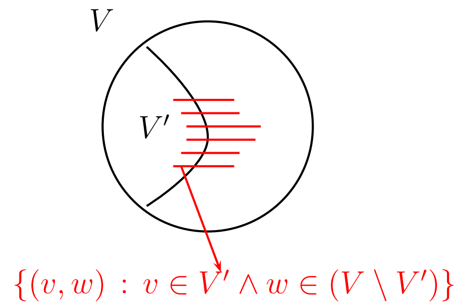



Abbildung 2.34: Skizze: Erweiterung von $V'$

Den Beweis, dass wir hiermit einen minimalen Spannbaum konstruieren, wollen wir an dieser Stelle nicht führen. Wir verweisen hierzu auf die einschlägige Literatur. Den formalen Algorithmus haben wir in Abbildung 2.35 angegeben. Bis auf die blauen Zeilen ist dies genau der Pseudo-Code für Prims Algorithmus zur Konstruktion eines minimalen Spannbaums.

Die blaue for-Schleife ist dazu da, um zu testen, ob der konstruierte minimale Spannbaum wirklich ein kompakter additiver Baum für $D$ ist. Zum einen setzen wir den konstruierten Abstand $d_T$ für den konstruierten minimalen Spannbaum. Der nachfolgende Test überprüft, ob dieser Abstand $d_T$ mit der vorgegebenen Distanzmatrix $\gamma$ übereinstimmt.

Wir müssen uns jetzt nur noch die Laufzeit überlegen. Die while-Schleife wird genau $n = |V|$ Mal durchlaufen. Danach ist $V' = V$. Die Minimumsbestimmung lässt sich sicherlich in Zeit $O(n^2)$ erledigen, da maximal $n^2$ Kanten zu durchsuchen sind. Daraus folgt eine Laufzeit von $O(n^3)$.

Implementiert man Prims-Algorithmus ein wenig geschickter, z.B. mit Hilfe von Priority-Queues, so lässt sich die Laufzeit auf $O(n^2)$ senken. Für die Details verweisen wir auch hier auf die einschlägige Literatur. Auch die blaue Schleife kann insgesamt in Zeit $O(n^2)$ implementiert werden, da jede for-Schleife maximal $n$-mal durchlaufen wird und die for-Schleife selbst maximal $n$-mal ausgeführt wird.

**Theorem 2.28** *Sei $D$ eine $n \times n$-Distanzmatrix. Dann lässt sich in Zeit $O(n^2)$ entscheiden, ob ein kompakter additiver Baum für $D$ existiert. Falls dieser existiert, kann dieser ebenfalls in Zeit $O(n^2)$ konstruiert werden.*

```
MODIFIED_PRIM
{
    E' := ∅;
    V' := {v} für ein beliebiges v ∈ V;
    /* (V', E') wird am Ende der minimale Spannbaum sein */
    while (V' ≠ V)
    {
        Sei e = (x, y), so dass γ(x, y) = min {γ(v, w) : v ∈ V' ∧ w ∈ V \ V'};
        /* oBdA sei y ∈ V' */
        E' := E' ∪ {e};
        V' := V' ∪ {y};
        for all (v ∈ V');
        {
            d_T(v, y) := d_T(v, x) + γ(x, y);
            if (d_T(v, y) ≠ γ(v, y)) reject;
        }
    }
    return (V', E');
}
```

Abbildung 2.35: Algorithmus: Konstruktion eines kompakten additiven Baumes

## 2.4 Exkurs: Priority Queues & Fibonacci-Heaps

In diesem Abschnitt gehen wir in einen kurzen Exkurs auf eine Realisierung von Priority Queues mittels Fibonacci-Heaps ein.

### 2.4.1 Priority Queues

Ein *Priority Queue* ist eine Datenstruktur mit folgenden Operationen:

**empty()** erzeugt eine leere Priority Queue.

**ref insert(elt, key)** fügt ein Element 'elt' in die Priority Queue mit dem Schlüssel 'key' ein, und liefert Referenz 'ref' auf das Element 'elt' zurück.

**int size()** gibt die Anzahl Elemente in der Priority Queue zurück.

**elt delete_min()** liefert ein Element mit dem kleinsten Schlüssel, das in der Priority Queue enthalten ist, und entfernt dieses aus der Priority Queue.

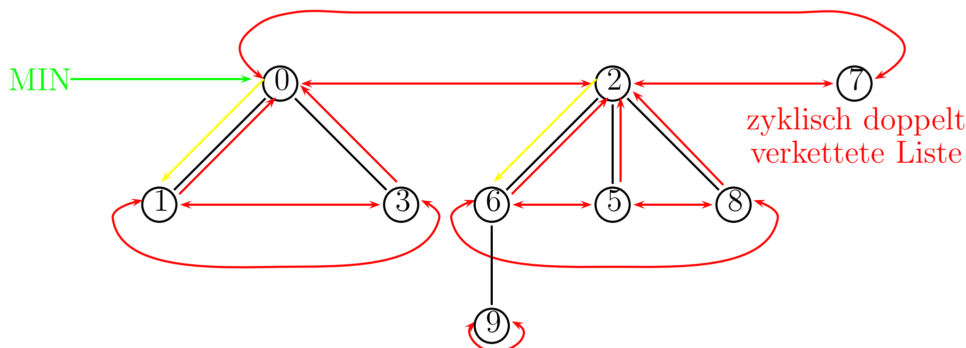
**bool decrease_key(ref)** erniedrigt den Schlüssel des Elements mit Referenz 'ref' in der Priority Queue. Liefert false, falls das Element nicht in der Priority Queue ist und sonst true.

**delete(ref)** löscht das Element mit der Referenz 'ref' aus der Priority Queue.

Meist wird die Operation delete nicht explizit gefordert und auch nicht benötigt. Wir werden sie im Folgenden daher auch außer Acht lasen.

### 2.4.2 Realisierung mit Fibonacci-Heaps

Ein *Heap* ist ein Baum, in dem in den Knoten Elemente mit Schlüssel gespeichert sind, wobei auf den Schlüsseln eine totale Ordnung gegeben ist. Dabei muss ein Heap die *Heap-Bedingung* erfüllen, die besagt, dass für jeden Knoten des Baumes mit Ausnahmen der Wurzel gilt, dass der Schlüssel des Elements im Elter-Knoten kleiner gleich dem Schlüssel des betrachteten Knotens sein muss. Ein *Fibonacci-Heap* ist ein Wald, deren Bäume Heaps sind.




**Definition 2.29** *Der* Rang *eines Knoten in einem Fibonacci-Heap ist die Anzahl seiner Kinder.*

### 2.4.3 Implementierung

Für die Implementierung von Fibonacci-Heaps vereinbaren wir das Folgende:

- Die Wurzeln der Heaps eines Fibonacci-Heaps werden in einer doppelt verketteten zyklischen Liste gehalten. Diese Liste wird im Folgenden auch als *Wurzelliste* bezeichnet.

- Ebenso werden die Kinder eines Knotens in einer doppelt verketten zyklischen Liste gehalten.
- Die Wurzel eines Heaps mit dem Element mit einem kleinsten Schlüssel wird mit dem so genannten Min-Zeiger memoriert.
- Jeder Knoten mit Ausnahme der Wurzeln von Heaps haben einen Verweis auf ihren Elter.
- Jeder Knoten besitzt eine Verweis auf eines seiner Kinder (nicht auf alle, denn die sind ja dann über die doppelt verkettete zyklische Kette der Geschwister-Knoten zugänglich).
- Die Knoten eines Fibonacci-Heaps können markiert sein (die Wurzeln werden dabei niemals markiert sein).

**22. Mai**

Nun geben wir eine Realisierung einer Priority Queue basierend auf einem Fibonacci-Heap an.

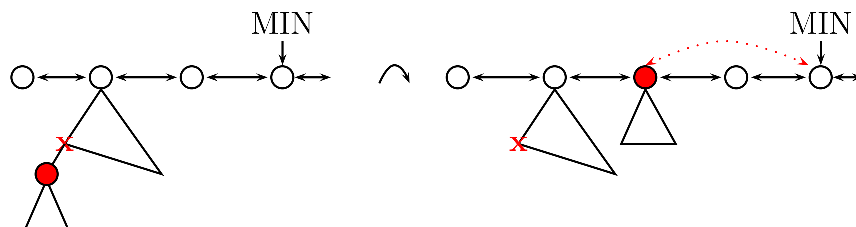
**size** Die Anzahl der Elemente der Priority Queue wird in einer Variablen gespeichert. Diese wird zu Beginn mit 0 initialisiert und wird bei insert's um 1 erhöht und bei delete_min's um 1 erniedrigt.

**insert** Füge ein neues Element als einelementigen Baum in die Wurzelliste ein. Aktualisiere dann den Min-Pointer.

**empty** Gib eine leere Wurzelliste zurück.

**decrease_key** Die Realisierung erfolgt wie folgt:

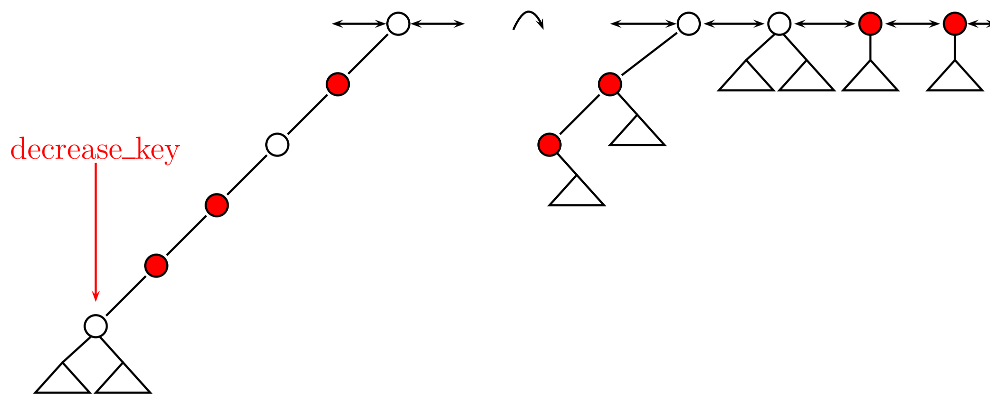
- Hänge den Teilbaum gewurzelt am gegebenen Element ab und füge diesen Teilbaum in die Wurzelliste ein.
- Erniedrige den Wert in der Wurzel dieses Teilbaumes, dabei bleibt die Heap-Bedingung offensichtlich erfüllt.




- Aktualisiere den Min-Pointer.

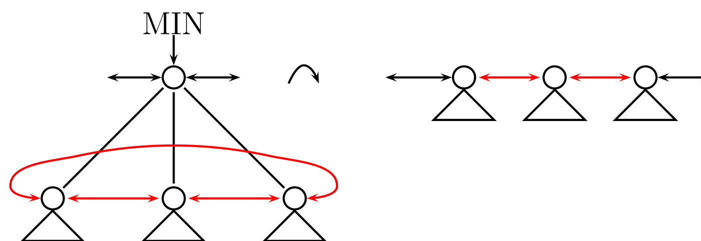
- Markiere den Elter des abgehängten Elements (außer es war eine Wurzel)
- War dieser Knoten bereits markiert, so wiederhole Verfahren des Abhängens mit diesem Knoten.

  Dies kann ein mehrfaches Abhängen von Teilbäumen zur Folge haben und wird als *kaskadenartiger Schnitt* bezeichnet.



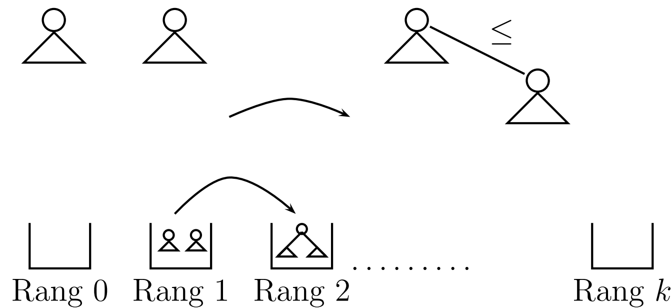

**delete_min** Die Realisierung erfolgt wie folgt:

- Gib das Element, auf das der Min-Pointer zeigt, aus und lösche es aus dem Heap.




- Füge die doppelt verkettet Liste der Kinder des gelöschten Knotens in die Wurzelliste ein.
- Aufräumen der Wurzelliste: Verschmelze Bäume, deren Wurzel gleichen Rang besitzen. Beim Verschmelzen ist dabei zu beachten, dass die Wurzel mit dem kleineren Schlüssel zum Elter der Wurzel des anderen Baumes

gemacht wird. Somit bleibt die Heap-Bedingung für den neu entstandenen Baum erhalten.




Die Aufräumaktion selbst wird mit Hilfe eines Feldes der Größe $O(\log(n))$ geführt, das in jedem Eintrag einen Heap aufnehmen kann. Die Heaps werden der Reihe nach in das Feld eingebracht, wobei ein Heap, deren Wurzel Rang $k$ hat, in den Index $k$ des Feldes geschrieben wird. Ist bereits ein Heap in einem Index, so werden diese beiden Heaps gemäß der Heap-Bedingung verschmolzen. Dann wird wieder versucht den neuen Heap im Feld an der richtigen Indexposition abzuspeichern.

Zum Schluss wird das Feld geleert und gleichzeitig sukzessive eine neue Wurzelliste aufgebaut.

- Bestimme beim sukzessiven Aufbau der neuen Wurzelliste nebenbei den neuen Min-Zeiger.

### 2.4.4 Worst-Case Analyse

Zunächst einmal wollen wir die worst-case Kosten der einzelnen Priority Queue Operationen abschätzen.

**Lemma 2.30** *Sei $v$ ein Knoten des Fibonacci-Heaps und seien $v_1, \ldots, v_k$ seine Kinder in der Reihenfolge, wie sie Kinder von $v$ wurden. Dann ist der Rang von $v_i$ mindestens $i - 2$.*

**Beweis: (durch Induktion)**

**Induktionsanfang ($i \leq 2$):** Es ist nichts zu zeigen

**Induktionsschritt ($\rightarrow i$):** Betrachte den Zeitpunkt als $v_i$ ein Kind von $v$ wurde. Dann hatte $v$ bereits die Kinder $v_1, \ldots, v_{i-1}$ (eventuell auch mehr). Der Rang von $v$ war zu diesem Zeitpunkt daher mindestens $i - 1$. Zu diesem Zeitpunkt war der

Rang von $v_i$ ebenfalls mindestens $i-1$, da beim Verschmelzen die Wurzeln gleiche Ränge gehabt haben müssen.

Wie viele Kinder kann $v_i$ seitdem verloren haben? Maximal eines, da $v_i$ nach einem decrease_key auf einem Kind von $v_i$ markiert wird.

Jedes weitere decrease_key würde $v_i$ vom Elter $v$ trennen.

Somit ist der Rang von $v_i$ mindestens $i-2$. ■

**Definition 2.31** *Die* Fibonacci-Zahlen *sind wie folgt definiert:* $f_0 = 0$*,* $f_1 = 1$ *und* $f_{n+2} = f_{n+1} + f_n$ *für* $n \geq 2$*.*

**Lemma 2.32** *Es gilt für alle* $k \in \mathbb{N}$*:*

$$f_{k+2} = 1 + \sum_{i=1}^{k} f_i.$$

**Beweis:** Übungsaufgabe. ■

**Lemma 2.33** *Sei* $v$ *ein Knoten eines Fibonacci-Heaps mit Rang* $k$*, dann ist die Größe des an* $v$ *gewurzelten Teilbaumes mindestens* $f_{k+2}$*.*

**Beweis: (durch Induktion)**

**Induktionsanfang ($k \in \{0, 1\}$):**

- Ist der Rang $= 0$, dann ist die Größe des zugehörigen Teilbaumes genau $1 = f_2$.
- Ist der Rang $= 1$, dann ist die Größe des zugehörigen Teilbaumes mindestens $2 = f_3$.

**Induktionsschritt ($\rightarrow k$):** Sei $v$ ein Knoten mit Rang $k$. Nach Lemma 2.30 hat das $i$-te Kind Rang mindestens $i-2$. Nach Induktionsvoraussetzung ist die Größe des Teilbaumes von $i$-ten Kind mindestens $f_{(i-2)+2} = f_i$. Somit gilt für die Größe des Teilbaumes von $v$:

$$|T(v)| \geq \underbrace{1}_{\text{Wurzel}} + \underbrace{\sum_{i=2}^{k} f_i}_{2.,\ldots,k.\text{ Kind}} + \underbrace{1}_{1.\text{Kind}} = 1 + \sum_{i=1}^{k} f_i = f_{k+2}$$

Daraus folgt für die Analyse: $f(n) = \Theta(\varphi^n)$ mit $\varphi = \frac{1+\sqrt{5}}{2}$. Damit ist der Rang der Knoten durch $O(\log(n))$ beschränkt.

**Zusammenfassung:**

| Operation | Aufwand |
|---|---|
| insert | $O(n)$ |
| decrease_key | $O(n)$ |
| delete_min | $O(n)$ |

**27. Mai**

### 2.4.5 Amortisierte Kosten bei Fibonacci-Heaps

Die worst-case Kosten sind also sehr teuer. Man überlegt sich jedoch leicht, dass teure Operationen (wie delete_min's mit großen kaskadenartigen Schnitten) nicht zu oft hintereinander vorkommen können. Daher analysieren wir noch die amortisierten Kosten der Priority Queue Operationen. Dies sind die Kosten im Mittel, wenn wir den gesamten Lebenszyklus einer Priority Queue betrachten.

**Ziel:** Wir versuchen mit Hilfe eines Sparkontos die Kosten abzuschätzen. Wir werden bei billigen Operationen schon etwas vorweg sparen, um später teure Operationen vom Sparkonto bezahlen zu können.

Als *amortisierte Kosten* bezeichnen wir dabei die Kosten, die für jeden Schritt bezahlt werden. Das sind zum einen die Kosten, die wir direkt aus der Geldbörse bezahlen, und das, was wir auf das Sparkonto einzahlen. Das Abheben vom Sparkonto betrachten wir nicht, da wir diese Kosten bereits beim Einzahlen berücksichtigt haben.

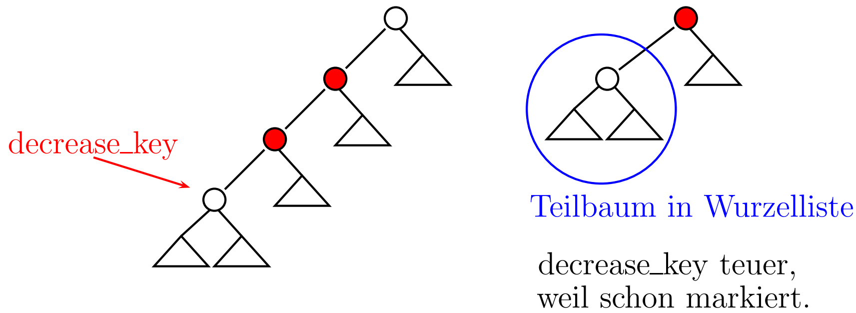
Im Folgenden beschreiben wir, bei welchen Operationen wir wieviel sparen und versuchen eine intuitive Beschreibung zu geben, warum.

**insert:** 1 Kosteneinheit auf Sparkonto.

Wir sparen schon einmal für das Aufräumen der Wurzelliste für den Fall, dass der eingefügte Knoten einmal in der Wurzelliste steht und bei einer Aufräumaktion beteiligt ist.

**decrease_key:** 2 Kosteneinheiten auf Sparkonto.

Wir sparen einmal für einen möglichen Schnitt des Elters innerhalb eines kaskadenartigen Schnittes und einmal für das Aufräumen der Wurzelliste für die Wurzel des Teilbaumes, den wir in die Wurzelliste einfügen.




**Kostenanalyse:** Wir untersuchen jetzt die anfallenden amortisierten Kosten für die einzelnen Operationen (außer für size und empty, da diese trivialerweise konstante Kosten haben).

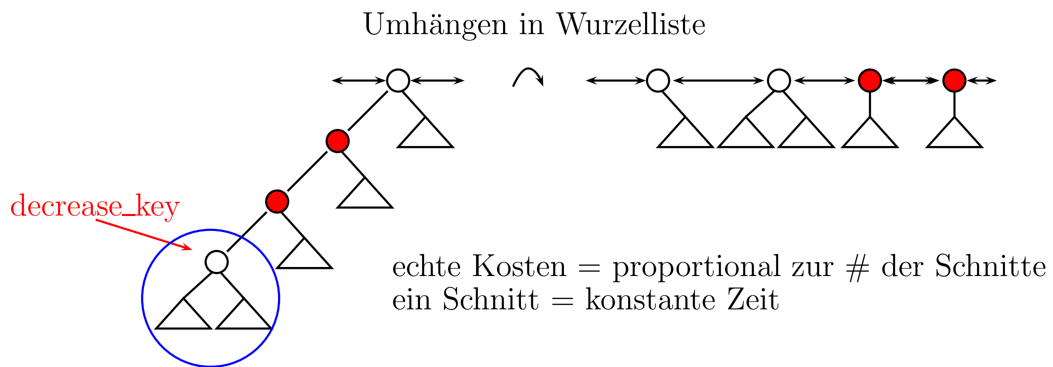
**insert:** Für die insert-Operation gilt:

  konstante Zeit
\+ konstante Zeit (sparen)

---

  konstante Zeit

**decrease_key:** für die decrease_key Operation gilt:

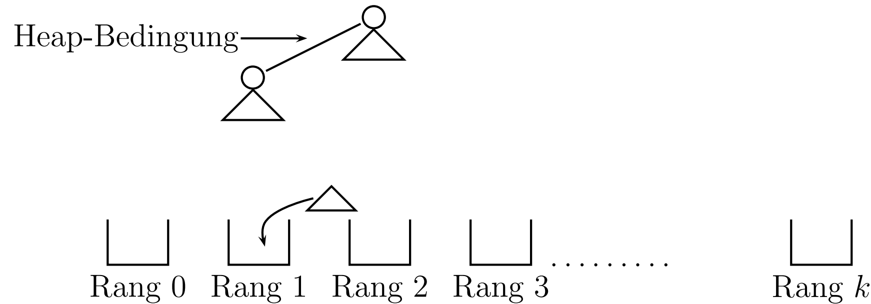



  konstante Zeit (für 1.Schnitt)
  Null (kaskadenartiger Schnitt durch Abheben vom Sparkonto)
\+ 2x konstante Kosten (sparen)

---

  konstante Zeit

**delete_min:** Verschmelzen von Bäumen mit gleichem Rang




Bezahle Verschmelzen vom Sparkonto (vom neuen Kind der Wurzel).

Die Kosten für das Aufräumen müssen direkt aus der Geldbörse bezahlt werden.

$$\begin{array}{rl} & \text{Null (Verschmelzen wird vom Sparkonto bezahlt)} \\ + & O(\log(n)) \text{ (Konstruktion der Wurzelliste)} \\ \hline & O(\log(n)) \end{array}$$

**Theorem 2.34** *Beginnend mit einem leeren Fibonacci-Heap kann eine Folge von $\ell$ Operationen (insert, decrease_key, delete_min, size) in Zeit $O(\ell + k \cdot \log(m))$ ausgeführt werden, wobei $m$ die maximale Anzahl von Elementen im Fibonacci-Heap ist und $k \leq \ell$ die Anzahl der delete_min-Operationen.*

Die erweiterte Version des Prim-Algorithmus basierend auf Priority-Queues ist in Abbildung 2.36 auf Seite 77 angegeben.

**Laufzeit PQ-Prim:** Zur Analyse des Algorithmus von Prim basierend auf Priority Queues stellen wir zunächst für die Anzahl der Aufrufe von Priority Queue Operationen fest:

- Anzahl insert's: $= n - 1 \leq n$
- Anzahl delete_min's: $= n - 1 \leq n$
- Anzahl decrease_key's: $\leq n^2$

```
PQ-Prim
{
    E = ∅;
    V' = {v};        // (für ein v ∈ V)
    PQ q.empty();

    for all (w ∈ V \ {v})
    {
        q.insert(w, γ(v, w));
        pred[w]=v;
    }

    while (V' ≠ V)
    {
        y :=q.delete_min();
        x =pred[y];
        E' = E' ∪ {x, y};
        V' = V' ∪ {y};
        for all (v ∈ V' \ {y})
        {
            d_T(v, y) = d_T(v, x) + γ(x, y);
            if (d_T(v, y) ≠ γ(v, y)) reject;
        }
        for all (w ∈ V \ V')
            if (γ(y, w) <q.key(w))
            {
                q.decrease_key(w, γ(y, w));
                pred[w]=y;}
            }
    }
}
```

Abbildung 2.36: Algorithmus: PQ-Prim

Somit ist die Laufzeit $O(n^2 + 2n + n\log(n)) = O(n^2)$. (Die Eingabegröße ist ja schon $n^2$, somit ist die Laufzeit eigentlich linear). Wir rezitieren hier der Vollständigkeit halber noch einmal das Gesamtergebnis.

**Theorem 2.35** *Sei $D$ eine $n \times n$-Distanzmatrix. Dann lässt sich in Zeit $O(n^2)$ entscheiden, ob ein kompakter additiver Baum für $D$ existiert. Falls dieser existiert, kann dieser ebenfalls in Zeit $O(n^2)$ konstruiert werden.*

## 2.5 Sandwich Probleme

Hauptproblem bei den bisher vorgestellten Verfahren war, dass wir dazu die Distanzen genau wissen mussten, da beispielsweise eine leicht modifizierte ultrametrische Matrix in der Regel nicht mehr ultrametrisch ist. Aus diesem Grund werden wir in diesem Abschnitt einige Problemestellungen vorstellen und lösen, die Fehler in den Eingabedaten modellieren.

### 2.5.1 Fehlertolerante Modellierungen

Bevor wir zur Problemformulierung kommen, benötigen wir noch einige Notationen, wie wir Matrizen zwischen zwei anderen Matrizen einschachteln können.

**Notation 2.36** *Seien $M$ und $M'$ zwei $\times n$-Matrizen, dann gilt $M \leq M'$, wenn $M_{i,j} \leq M'_{i,j}$ für alle $i, j \in [1 : n]$ gilt. Für drei Matrizen $M$, $M'$ und $M''$ gilt $M \in [M', M'']$, wenn $M' \leq M \leq M''$ gilt.*

Nun können wir die zu untersuchenden Sandwich-Probleme angeben.

ADDITIVES SANDWICH PROBLEM

**Eingabe:** Zwei $n \times n$-Distanzmatrizen $D_\ell$ und $D_h$.
**Gesucht:** Eine additive Distanzmatrix $D \in [D_\ell, D_h]$, sofern eine existiert.

ULTRAMETRISCHES SANDWICH PROBLEM

**Eingabe:** Zwei $n \times n$-Distanzmatrizen $D_\ell$ und $D_h$.
**Gesucht:** Eine ultrametrische Distanzmatrix $D \in [D_\ell, D_h]$, sofern eine existiert.

Im Folgenden bezeichnet $\|M\|$ eine Norm einer Matrix. Hierbei wird diese Norm jedoch nicht eine Abbildungsnorm der durch $M$ induzierten Abbildung sein, sondern wir werden Normen verwenden, die eine $n \times n$-Matrix als einen Vektor mit $n^2$ Einträgen interpretieren. Beispielsweise können wir die so genannten $p$-Normen verwenden:

$$\|D\|_p = \left( \sum_{i=1}^{n} \sum_{j=1}^{n} |M_{i,j}|^p \right)^{1/p},$$
$$\|D\|_\infty = \max \{ |M_{i,j}| : i, j \in [1 : n] \}.$$

Additives Approximationsproblem

**Eingabe:** Eine $n \times n$-Distanzmatrix $D$.
**Gesucht:** Eine additive Distanzmatrix $D'$, die $\|D - D'\|$ minimiert.

Ultrametrisches Approximationsproblem

**Eingabe:** Eine $n \times n$-Distanzmatrix $D$.
**Gesucht:** Eine ultrametrische Distanzmatrix $D'$, die $\|D - D'\|$ minimiert.

Für die weiteren Untersuchungen benötigen wir noch einige Notationen.

**Notation 2.37** *Sei $M$ eine $n \times n$-Matrix, dann ist* $\mathrm{MAX}(M)$ *der maximale Eintrag von $M$, d.h.* $\mathrm{MAX}(M) = \max\{M_{i,j} : i,j \in [1:n]\}$.

*Sei $T$ ein additiver Baum mit $n$ markierten Knoten (mit Markierungen aus $[1:n]$) und $d_T(i,j)$, der durch den additiven Baum induzierte Abstand zwischen den Knoten mit Markierung $i$ und $j$, dann ist* $\mathrm{MAX}(T) = \max\{d_T(i,j) : i,j \in [1:n]\}$.

*Sei $M$ eine $n \times n$-Matrix und $T$ ein additiver Baum mit $n$ markierten Knoten, dann schreiben wir $T \leq M$ bzw. $T \geq M$, wenn $d_T(i,j) \leq M_{i,j}$ bzw. $d_T(i,j) \geq M_{i,j}$ für alle $i,j \in [1:n]$ gilt.*

*Für zwei Matrizen $M$ und $M'$ sowie einen additiven Baum $T$ gilt $T \in [M, M']$, wenn $M \leq T \leq M'$ gilt.*

Wir wollen noch einmal kurz daran erinnern, wie man aus einem ultrametrischen Baum $T = (V, E, \mu)$ mit der Knotenmarkierung $\mu : V \to \mathbb{R}_+$ einen additiven Baum $T = (V, E, \gamma)$ mit der Kantengewichtsfunktion $\gamma : E \to \mathbb{R}_+$ erhält, indem man $\gamma$ wie folgt definiert:

$$\forall (v,w) \in E : \gamma(v,w) := \frac{\mu(v) - \mu(w)}{2} > 0.$$

Somit können wir ultrametrische Bäume immer auch als additive Bäume auffassen.

**3. Juni**

### 2.5.2 Eine einfache Lösung

In diesem Abschnitt wollen wir zuerst eine einfache Lösung angeben, um die Ideen hinter der Lösung zu erkennen. Im nächsten Abschnitt werden wir dann eine effizientere Lösung angeben, die von der Idee her im Wesentlichen gleich sein wird, aber dort nicht so leicht bzw. kaum zu erkennen sein wird.

Zuerst zeigen wir, dass wenn es einen ultrametrischen Baum gibt, der der Sandwich-Bedingung genügt, es auch einen ultrametrischen Baum gibt, dessen Wurzelmarkierung gleich dem maximalem Eintrag der Matrix $D_\ell$ ist. Dies liefert einen ersten Ansatzpunkt für einen rekursiven Algorithmus.

**Lemma 2.38** *Seien $D_\ell$ und $D_h$ zwei $n \times n$-Distanzmatrizen. Wenn ein ultrametrischer Baum $T$ mit $T \in [D_\ell, D_h]$ existiert, dann gibt es einen ultrametrischen Baum $T'$ mit $T' \in [D_\ell, D_h]$ und $\mathrm{MAX}(T') = \mathrm{MAX}(D_\ell)$.*

**Beweis:** Wir beweisen die Behauptung durch Widerspruch. Wir nehmen also an, dass es keinen ultrametrischen Baum $T' \in [D_\ell, D_h]$ mit $\mathrm{MAX}(T') = \mathrm{MAX}(D_\ell)$ gibt.

Sei $T' \in [D_\ell, D_h]$ ein ultrametrischer Baum, der $s := \mathrm{MAX}(T') - \mathrm{MAX}(D_\ell)$ minimiert. Nach Voraussetzung gibt es mindestens einen solchen Baum und nach unserer Annahme für den Widerspruchsbeweis ist $s > 0$.

Wir konstruieren jetzt aus $T'$ einen neuen Baum $T''$, indem wir nur die Kantengewichte der Kanten in $T''$ ändern, die zur Wurzel von $T'$ bzw. $T''$ inzident sind. Zuerst definieren wir $\alpha := \frac{1}{2}\min\{\gamma(e), s\} > 0$. Wir bemerken, dass dann $\alpha \leq \frac{\gamma(e)}{2}$ und $\alpha \leq \frac{s}{2}$ gilt.

Sei also $e$ ein Kante, die zur Wurzel von $T''$ inzident ist. Dann setzen wir

$$\gamma''(e) = \gamma'(e) - \alpha.$$

Hierbei bezeichnet $\gamma'$ bzw. $\gamma''$ die Kantengewichtsfunktion von $T'$ bzw. $T''$. Zuerst halten wir fest, dass die Kantengewichte der Kanten, die zur Wurzel inzident sind, weiterhin positiv sind, da

$$\gamma(e) - \alpha \;\geq\; \gamma(e) - \frac{\gamma(e)}{2} \;=\; \frac{\gamma(e)}{2} \;>\; 0.$$

Da wir Kantengewichte nur reduzieren, gilt offensichtlich weiterhin $T'' \leq D_h$. Wir müssen also nur noch zeigen, dass auch $D_\ell \leq T''$ weiterhin gilt. Betrachten wir hierzu zwei Blätter $v$ und $w$ in $T''$ und die Wurzel $r(T'')$ von $T''$. Wir unterscheiden jetzt zwei Fälle, je nachdem, ob der kürzeste Weg von $v$ nach $w$ über die Wurzel führt oder nicht.

**Fall 1 ($\mathrm{lca}(v, w) \neq r(T'')$):** Dann wird der Abstand von $d_{T''}$ gegenüber $d_{T''}$ für diese Blätter nicht verändert und es gilt:

$$d_{T''}(v, w) = d_{T'}(v, w) \geq D_\ell(v, w).$$

**Fall 2 ($\mathbf{lca}(v, w) = r(T'')$):** Dann werden die beiden Kantengewichte der Kanten auf dem Weg von $v$ zu $w$ die inzident zur Wurzel sind um jeweils $\alpha$ erniedrigt und wir erhalten:

$$\begin{aligned} d_{T''}(v,w) &= d_{T'}(v,w) - 2\alpha \\ &\quad \text{da } d_{T'}(v,w) = s + \mathrm{MAX}(D_\ell) \\ &= s + \mathrm{MAX}(D_\ell) - 2\alpha \\ &\quad \text{da } 2\alpha \leq s \\ &\geq s + \mathrm{MAX}(D_\ell) - s \\ &= \mathrm{MAX}(D_\ell) \\ &\geq D_\ell(v,w). \end{aligned}$$

Also ist $D_\ell \leq T''$. Somit haben wir einen ultrametrischen Baum $T'' \in [D_\ell, D_h]$ konstruiert, für den

$$\begin{aligned} \mathrm{MAX}(T'') - \mathrm{MAX}(D_\ell) &\leq \mathrm{MAX}(T') - 2\alpha - \mathrm{MAX}(D_\ell) \\ &\quad \text{da } \alpha > 0 \\ &< \mathrm{MAX}(T') - \mathrm{MAX}(D_\ell) \\ &= s. \end{aligned}$$

gilt. Dies ist offensichtlich ein Widerspruch zur Wahl von $T'$ und das Lemma ist bewiesen. ∎

Das vorherige Lemma legt die Definition niedriger ultrametrische Bäume nahe.

**Definition 2.39** *Ein ultrametrischer Baum $T \in [D_\ell, D_h]$ heißt* niedrig, *wenn $\mathrm{MAX}(T) = \mathrm{MAX}(D_\ell)$.*

Um uns im weiteren etwas leichter zu tun, benötigen wir einige weitere Notationen.

**Notation 2.40** *Sei $T = (V, E)$ ein gewurzelter Baum. Dann bezeichnet $\mathcal{L}(T)$ die Menge der Blätter von $T$. Für $v \in \mathcal{L}(T)$ bezeichnet*

$$\mathcal{L}(T, v) := \{w \in \mathcal{L}(T) \,:\, \mathrm{lca}(v, w) \neq r(T)\}$$

*die Menge aller Blätter die sich im selben Teilbaum der Wurzel von $T$ befinden wie $v$ selbst.*

Aus dem vorherigen Lemma und den Notationen folgt jetzt unmittelbar das folgende Korollar.

**Korollar 2.41** *Für jeden niedrigen ultrametrischen Baum $T \in [D_\ell, D_h]$ gilt:*

$$\forall x, y \in \mathcal{L}(T) : (D_\ell(x,y) = \mathrm{MAX}(D_\ell)) \Rightarrow (d_T(x,y) = \mathrm{MAX}(D_\ell)).$$

Bevor wir zum zentralen Lemma kommen, müssen wir noch eine weitere grundlegende Definition festlegen.

**Definition 2.42** *Seien $D_\ell \leq D_h$ zwei $n \times n$-Matrizen und seien $k, \ell \in [1:n]$, dann ist der Graph $G_{k,\ell} = (V, E_{k,\ell})$ wie folgt definiert:*

$$\begin{aligned} V &:= [1:n], \\ E_{k,l} &:= \{(i,j) \ : \ D_h(i,j) < D_\ell(k,\ell)\}. \end{aligned}$$

*Mit $C(G_{k,\ell}, v)$ bezeichnen wir die Zusammenhangskomponente von $G_{k,\ell}$, die den Knoten $v$ enthält.*

Nun kommen wir zu dem zentralen Lemma für unseren einfachen Algorithmus, das uns beschreibt, wie wir mit Kenntnis des Graphen $G_{k\ell}$ (oder besser dessen Zusammenhangskomponenten) den gewünschten ultrametrischen Baum konstruieren können.

**Lemma 2.43** *Seien $D_\ell \leq D_h$ zwei $n \times n$-Matrizen und seien $k, \ell \in [1:n]$ so gewählt, dass $D_\ell(k,\ell) = \mathrm{MAX}(D_\ell)$. Wenn ein niedriger ultrametrischer Baum $T \in [D_\ell, D_h]$ existiert, dann gibt es auch einen niedrigen ultrametrischen Baum $T' \in [D_\ell, D_h]$ mit*

$$\mathcal{L}(T', v) = V(C(G_{k,\ell}, v))$$

*für alle $v \in \mathcal{L}(T)$.*

**Beweis:** Wir beweisen zuerst die folgende Behauptung:

$$\forall T \in [D_\ell, D_h] : V(C(G_{k,l}, v)) \subseteq \mathcal{L}(T, v).$$

Wir führen diesen Beweis durch Widerspruch. Sei dazu $x \in V(C(G_{k,\ell}, v)) \setminus \mathcal{L}(T, v)$. Da $x \in V(C(G_{k,\ell}, v))$ gibt es einen Pfad $p$ von $v$ nach $x$ in $G_{k\ell}$ (siehe dazu auch Abbildung 2.37). Sei $(y, z) \in p$ die erste Kante des Pfades von $v$ nach $x$ in $G_{k\ell}$, so dass $y \in \mathcal{L}(t, v)$, aber $z \notin \mathcal{L}(T, v)$ gilt. Da $(y, z) \in E(C(G_{k,\ell}, v)) \subseteq E_{k,\ell}$ ist, gilt $D_h(y, z) < D_\ell(k, \ell)$.

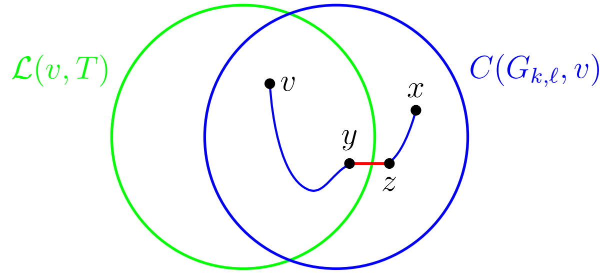



Abbildung 2.37: Skizze: $\mathcal{L}(T,v)$ und $C(G_{k,\ell},v)$

Da $y \in \mathcal{L}(T,v)$, aber $z \notin \mathcal{L}(T,v)$ ist, gilt $\text{lca}(y,z) = r(T)$. Somit ist

$$d_T(y,z) \geq \text{MAX}(D_\ell) = D_\ell(k,\ell).$$

Daraus folgt unmittelbar, dass

$$d_T(y,z) \geq D_\ell(k,\ell) > D_h(y,z).$$

Dies ist aber offensichtlich ein Widerspruch zu $T \in [D_\ell, D_h]$ und somit ist die Behauptung gezeigt.

Wir zeigen jetzt, wie ein Baum $T$ umgebaut werden kann, so dass er die gewünschte Eigenschaft des Lemmas erfüllt. Sei also $T$ ein niedriger ultrametrischer Baum mit $T \in [D_\ell, D_h]$. Sei weiter $S := \mathcal{L}(T,v) \setminus V(C(G_{k,\ell},v))$. Ist $S$ leer, so ist nichts mehr zu zeigen. Sei also $s \in S$ und $x \in V(C(G_{k,\ell},v))$. Nach Wahl von $s$ und $x$ gibt es in $G_{k,\ell}$ keinen Pfad von $s$ nach $x$. Somit gilt

$$D_h(x,s) \geq D_\ell(k,\ell) = \text{MAX}(D_\ell) = \text{MAX}(T) \geq d_T(x,s) \geq D_\ell(x,s).$$

Wir bauen jetzt $T$ zu $T'$ wie folgt um. Betrachte den Teilbaum $T_1$ der Wurzel $r(T)$ von $T$ der sowohl $x$ als auch $s$ enthält. Wir duplizieren $T_1$ zu $T_2$ und hängen an die Wurzel von $T''$ sowohl $T_1$ als auch $T_2$ an. In $T_1$ entfernen wir alle Blätter aus $S$ und in $T_2$ entfernen wir alle Blätter aus $V(C(G_{k,\ell},v))$ (siehe auch Abbildung 2.38). Anschließend räumen wir in den Bäumen noch auf, indem wir Kanten entfernen, die zu keinen echten Blättern mehr führen. Ebenso löschen wir Knoten, die nur noch ein Kind besitzen, indem wir dieses Kind zum Kind des Elters des gelöschten Knoten machen. Letztendlich erhalten wir einen neuen ultrametrischen Baum $T''$.

Wir zeigen jetzt, dass $T'' \in [D_\ell, D_h]$ ist. Da wir die Knotenmarkierung der überlebenden Knoten nicht ändern, bleibt der ultrametrische Baum niedrig. Betrachten wir zwei Blätter $x$ und $y$ , die nicht zu $T_1$ oder $T_2$ gehören. Da der Pfad in $T''$ derselbe wie in $T'$ ist, gilt weiterhin

$$D_\ell(x,y) \leq d_{T''}(x,y) \leq D_h(x,y).$$

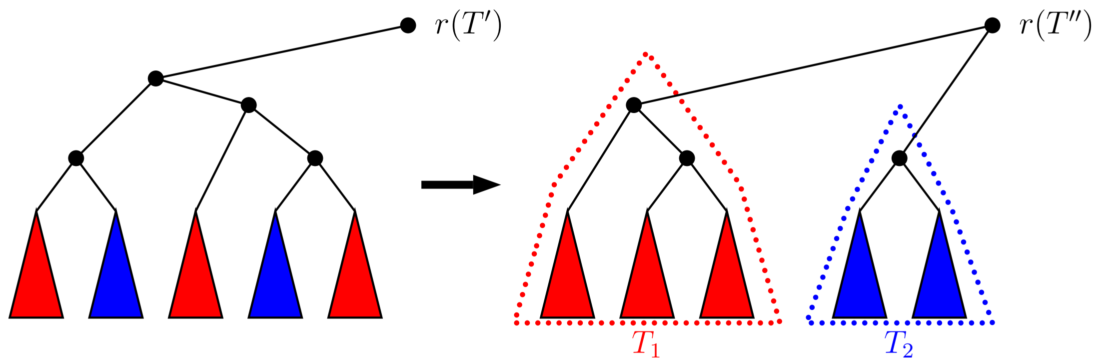



Abbildung 2.38: Skizze: Umbau des niedrigen ultrametrischen Baumes

Gehört ein Knoten $x$ zu $T_1$ oder $T_2$ und der andere Knoten $y$ nicht zu $T_1$ und $T_2$, so ist der Pfad nach die Duplikation bezüglich des Abstands derselbe. Es gilt also wiederum

$$D_\ell(x, y) \leq d_{T''}(x, y) \leq D_h(x, y).$$

Gehören beide Knoten $v$ und $w$ entweder zu $T_1$ oder zu $T_2$, dann hat sich der Pfad nach der Duplikation bzgl. des Abstands auch wieder nicht geändert, also gilt

$$D_\ell(x, y) \leq d_{T''}(x, y) \leq D_h(x, y).$$

Es bleibt der Fall zu betrachten, dass ein Knoten zu $T_1$ und einer zu $T_2$ gehört. Hier hat sich der Pfad definitiv geändert, da er jetzt über die Wurzel von $T''$ führt. Sei $s$ der Knoten in $T_1$ und $x$ der Knoten in $T_2$. Wir haben aber bereits gezeigt, dass für solche Knoten gilt:

$$D_h(x, s) \geq d_{T''}(x, s) \geq D_\ell(x, s).$$

Somit ist der Satz bewiesen. ■

**5. Juni**

Aus diesem Beweis ergibt sich unmittelbar die folgende Idee für unseren Algorithmus. Aus der Kenntnis des Graphen $G_{k\ell}$ für eine größte untere Schranke $D_\ell(k, \ell)$ für zwei Spezies $k$ und $\ell$ beschreiben uns die Zusammenhangskomponenten die Partition der Spezies, wie diese in den verschiedenen Teilbäumen an der Wurzel des ultrametrischen Baumes hängen müssen, sofern es überhaupt einen gibt. Somit können wir die Spezies partitionieren und für jede Menge der Partition einen eigenen ultrametrischen Baum rekursiv konstruieren, deren Wurzeln dann die Kinder der Gesamtwurzel des zu konstruierenden ultrametrischen Teilbaumes werden. Damit ergibt sich der folgende, in Abbildung 2.39 angegebene Algorithmus.

- Bestimme $k, \ell \in [1:n]$, so dass $D_\ell(k,\ell) = \mathrm{MAX}(D_\ell)$. $O(n^2)$
- Konstruiere $G_{k,\ell}$. $O(n^2)$
- Bestimme die Zusammenhangskomponenten $C_1, \ldots, C_m$ von $G_{k,\ell}$. $O(n^2)$
- Konstruiere rekursiv ultrametrische Bäume für die einzelnen Zusammenhangskomponenten. $\sum_{i=1}^{m} T(|C_i|)$
- Baue aus den Teillösungen $T_1, \ldots, T_m$ für $C_1, \ldots, C_m$ einen ultrametrischen Baum, indem man die Wurzeln der $T_1, \ldots, T_m$ als Kinder an eine neue Wurzel hängt, die als Knotenmarkierung $\mathrm{MAX}(D_\ell)$ erhält. $O(n)$

Abbildung 2.39: Algorithmus: Algorithmus für das ultrametrische Sandwich-Problem

Für die Korrektheit müssen wir nur noch zeigen, dass die Partition nicht trivial ist, d.h., dass der Graph $G_{k,\ell}$ nicht zusammenhängend ist.

**Lemma 2.44** *Seien $D_\ell \leq D_h$ zwei $n \times n$-Distanzmatrizen und seien $k, \ell \in [1:n]$ so gewählt, dass $D_\ell(k,\ell) = \mathrm{MAX}(D_\ell)$ gilt. Wenn $G_{k,\ell}$ zusammenhängend ist, dann kann es keine ultrametrische Matrix $U \in [D_\ell, D_h]$ geben.*

**Beweis:** Angenommen, es gäbe eine ultrametrische Matrix $U \in [D_\ell, D_h]$. Da $G_{k,\ell}$ zusammenhängend ist, gibt es einen Pfad $p = (v_1, \ldots, v_m)$ mit $v_1 = k$ und $v_m = \ell$ in $G_{k,\ell}$. Aufgrund der ultrametrischen Matrix $U$ gilt dann (da diese ja die ultrametrische Dreiecksungleichung erfüllt):

$$\begin{aligned}
U(k,\ell) &\leq \max\{U(k,v_2), U(v_2,\ell)\} \\
&\leq \max\{U(k,v_2), \max\{U(v_2,v_3), U(v_3,\ell)\}\} \\
&\leq \max\{U(k,v_2), U(v_2,v_3), U(v_3,\ell)\} \\
&\leq \max\{U(k,v_2), U(v_2,v_3), U(v_3,v_4), U(v_4,\ell)\} \\
&\leq \vdots \\
&\leq \max\{U(k,v_2), U(v_2,v_3), \ldots, U(v_{m-1},\ell)\} \\
&\quad \text{da ja } k = v_1 \text{ und } \ell = v_m \\
&= \max\{U(v_1,v_2), U(v_2,v_3), \ldots, U(v_{m-1},v_m)\} \\
&\quad \text{da } U \in [D_\ell, D_h] \\
&\leq \max\{D_h(v_1,v_2), D_h(v_2,v_3), \ldots, D_h(v_{m-1},v_m)\} \\
&\quad \text{nach Konstruktion von } G_{k,\ell} \\
&< D_\ell(k,\ell)
\end{aligned}$$

Damit erhalten wir also $U(k,\ell) < D_\ell(k,\ell)$. Dies ist aber offensichtlich ein Widerspruch dazu, dass $U \in [D_\ell, D_h]$. ■

Damit ist die Korrektheit bewiesen. In Abbildung 2.40 ist ein Beispiel zur Illustration der Vorgehensweise des einfachen Algorithmus angegeben.

Für die Laufzeit erhalten wir

$$T(n) = O(n^2) + \sum_{i=1}^{m} T(n_i)$$

mit $\sum_{i=1}^{m} n_i = n$. Wir überlegen uns, was bei einem rekursiven Aufruf geschieht. Bei jedem rekursiven Aufruf wird eine Partition der Menge $[1:n]$ verfeinert. Da wir mit der trivialen Partition $\{[1:n]\}$ starten und mit $\{\{1\},\ldots,\{n\}\}$ enden, kann es also maximal $n-1$ rekursive Aufrufe geben. Somit ist die Laufzeit durch $O(n \cdot n^2)$ beschränkt.

**Theorem 2.45** *Seien $D_\ell \leq D_h$ zwei $n \times n$-Distanzmatrizen. Ein ultrametrischer Baum $T \in [D_\ell : D_h]$ kann in Zeit $O(n^3)$ konstruiert werden, sofern überhaupt einer existiert.*

### 2.5.3 Charakterisierung einer effizienteren Lösung

In diesem Abschnitt wollen wir zeigen, dass wir das ultrametrische Sandwich Problem sogar in Zeit $O(n^2)$ lösen können. Dies ist optimal, da ja bereits die Eingabematrizen die Größe $\Theta(n^2)$ besitzen. Dazu benötigen wir erst noch die Definition der Kosten eines Pfades.

**Definition 2.46** *Sei $G = (V, E, \gamma)$ ein gewichteter Graph. Die* Kosten *$c(p)$ eines Pfades $p = (v_0, \ldots, v_n)$ ist definiert durch*

$$c(p) := \max\left\{\gamma(v_{i-1}, v_i) \ : \ i \in [1:n]\right\}.$$

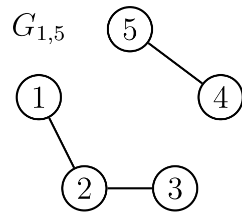
*Mit $D(G, v, w)$ bezeichnen wir die minimalen Kosten eines Pfades von $v$ nach $w$ in $G$.*

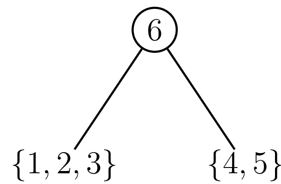
$$D(G, v, w) := \min\left\{c(p) \ : \ p \text{ ist ein Pfad von } v \text{ nach } w\right\}.$$

Warum interessieren wir uns überhaupt für die Kosten eines Pfades? Betrachten wir einen Pfad $p = (v_0, \ldots, v_n)$ in einem gewichteten Graphen $G(D)$, der von einer ultrametrischen Matrix $D$ induziert wird. Seien dabei $\gamma(v, w)$ die Kosten der Kante

| $D_\ell$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 2 | 3 | 6 |
| 2 | | 0 | 4 | 5 | 5 |
| 3 | | | 0 | 4 | 5 |
| 4 | | | | 0 | 1 |
| 5 | | | | | 0 |

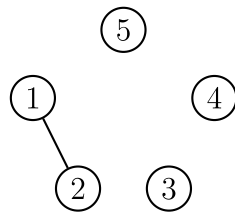
| $D_h$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 3 | 6 | 8 | 8 |
| 2 | | 0 | 5 | 6 | 8 |
| 3 | | | 0 | 6 | 8 |
| 4 | | | | 0 | 3 |
| 5 | | | | | 0 |

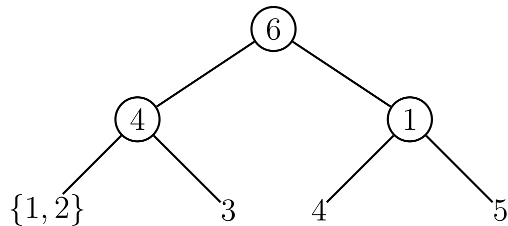



| $D_\ell$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 2 | 3 | 6 |
| 2 | | 0 | 4 | 5 | 5 |
| 3 | | | 0 | 4 | 5 |
| 4 | | | | 0 | 1 |
| 5 | | | | | 0 |

| $D_h$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 3 | 6 | 8 | 8 |
| 2 | | 0 | 5 | 6 | 8 |
| 3 | | | 0 | 6 | 8 |
| 4 | | | | 0 | 3 |
| 5 | | | | | 0 |



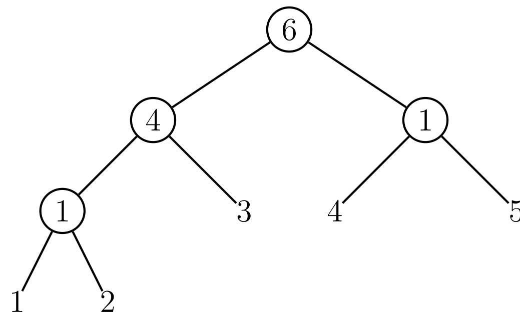




Abbildung 2.40: Beispiel: Einfache Lösung des ultrametrischen Sandwich-Problems

$(v,w)$. Wie groß ist nun der Abstand $d_D(v_0,v_n)$ von $v_0$ zu $v_n$? Unter Berücksichtigung der ultrametrischen Dreiecksungleichung erhalten wir:

$$\begin{aligned}
d_D(v_0,w_n) &\leq \max\{d_D(v_0,v_{n-1}),\gamma(v_{n-1},v_n)\}\\
&\leq \max\{\max\{d_D(v_0,v_{n-2}),\gamma(v_{n-2},v_{n-1})\},\gamma(v_{n-1},v_n)\}\\
&= \max\{d_D(v_0,v_{n-2}),\gamma(v_{n-2},v_{n-1}),\gamma(v_{n-1},v_n)\}\\
&\vdots\\
&\leq \max\{\gamma(v_0,v_1),\ldots,\gamma(v_{n-2},v_{n-1}),\gamma(v_{n-1},v_n)\}\\
&= c(p)
\end{aligned}$$

Die letzte Gleichung folgt nur für den Fall, dass wir einen Pfad mit minimalen Kosten gewählt haben. Somit sind die Kosten eines Pfades in dem zur ultrametrischen Matrix gehörigem gewichteten Graphen eine obere Schranke für den Abstand der Endpunkte dieses Pfades.

Im folgenden Lemma erhalten wir dann eine weitere Charakterisierung, wann zwei Knoten $k$ und $\ell$ im zugehörigen Graphen $G_{k\ell}$ durch einen Pfad verbunden sind. Man beachte, dass wir hier nicht beliebige Knotenpaare betrachten, sondern genau das Knotenpaar, dessen maximaler Abstand gemäß der unteren Schranke den Graphen $G_{k\ell}$ definiert.

**Lemma 2.47** *Seien $D_\ell \leq D_h$ zwei Distanzmatrizen. Zwei Knoten $k$ und $\ell$ befinden sich genau dann in derselben Zusammenhangskomponente von $G_{k,\ell}$, wenn $D_\ell(k,\ell) > D(G(D_h),k,\ell)$.*

**Beweis:** **⇒:** Wenn sich $k$ und $\ell$ in derselben Zusammenhangskomponente von $G_{k,\ell}$ befinden, dann gibt es einen Pfad $p$ von $k$ nach $\ell$. Für alle Kanten $(v,w) \in p$ gilt daher (da sie Kanten in $G_{k,\ell}$ sind): $\gamma(v,w) < D_\ell(k,l)$. Somit ist auch das Maximum der Kantengewichte durch $D_\ell(k,\ell)$ beschränkt und es gilt $D(G(D_h),k,\ell) < D_\ell(k,\ell)$.

**⇐:** Gelte nun $D(G(D_h),k,\ell) < D_\ell(k,\ell)$. Dann existiert nach Definition von $D(\cdot,\cdot)$ und $G_{k,\ell}$ ein Pfad $p$ in $G(D_h)$, so dass das Gewicht jeder Kante durch $D_\ell(k,\ell)$ beschränkt ist. Somit ist $p$ auch ein Pfad in $G_{k,\ell}$ von $v$ nach $w$. Also befinden sich $v$ und $w$ in derselben Zusammenhangskomponente von $G_{k,\ell}$. ■

**Notation 2.48** *Sei $T$ ein Baum. Den eindeutigen einfachen Pfad von $v \in V(T)$ nach $w \in V(T)$ bezeichnen wir mit $p_T(v,w)$.*

Im Folgenden sei $T$ ein minimaler Spannbaum von $G(D_h)$. Mit obiger Notation gilt dann, dass $D(T,v,w) = c(p_T(v,w))$. Wir werden jetzt zeigen, dass wir die oberen

Schranken, die eigentlich durch die Matrix $D_h$ gegeben sind, durch den zugehörigen minimalen Spannbaum von $G(D_h)$ mit viel weniger Speicherplatz darstellen können.

**Lemma 2.49** *Sei $D_h$ eine Distanzmatrix und sei $T$ ein minimaler Spannbaum des Graphen $G(D_h)$. Dann gilt $D(T, v, w) = D(G(D_h), v, w)$ für alle Knoten $v, w \in V(T) = V(G(D_h))$.*

**Beweis:** Zuerst halten wir fest, dass jeder Pfad in $T$ auch ein Pfad in $G(D_h)$ ist. Somit gilt in jedem Falle

$$D(G(D_h), v, w) \leq D(T, v, w).$$

Für einen Widerspruchsbeweis nehmen wir jetzt an, dass es zwei Knoten $v$ und $w$ mit

$$D(G(D_h), v, w) < D(T, v, w)$$

gibt. Dann existiert ein Pfad $p$ in $G(D_h)$ von $v$ nach $w$ mit $c(p) < D(T, v, w)$.

Wir betrachten jetzt den eindeutigen Pfad $p_T(v, w)$ im minimalen Spannbaum $T$. Sei jetzt $(x, y)$ eine Kante in $p_T(v, w)$mit maximalem Gewicht, also $\gamma(x, y) = c(p_T)$. Wir entfernen jetzt diese Kante aus $T$ und erhalten somit zwei Teilbäume $T_1$ und $T_2$, die alle Knoten des Graphen $G(D_h)$ beinhalten. Dies ist in der Abbildung 2.41 illustriert.

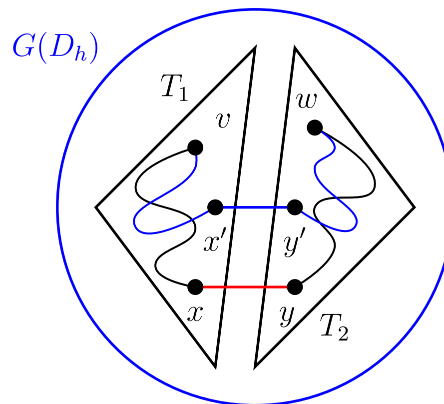



Abbildung 2.41: Skizze: Spannender Wald $\{T_1, T_2\}$ von $G(D_h)$ nach Entfernen von $\{x, y\}$ aus dem minimalen Spannbaum $T$

Sei $(x', y')$ die erste Kante auf dem Pfad $p$ in $G(D_h)$, die die Bäume $T_1$ und $T_2$ verbindet, d.h. $x' \in V(T_1)$ und $y' \in V(T_2)$. Nach Voraussetzung gilt, dass

$$D_h(x', y') < D_h(x, y),$$

da jede Kante des Pfades $p$ nach Widerspruchsannahme leichter sein muss als die schwerste Kante in $p_T$ und da $(x, y)$ ja eine schwerste Kante in $p_T$ war.

Dann könnten wir jedoch einen neuen Spannbaum $T'$ mittels

$$E(T') = (E(T) \setminus \{(x, y)\}) \cup \{(x', y')\}$$

konstruieren. Dieser hat dann Gewicht

$$\gamma(T') \;=\; \gamma(T) + \underbrace{\gamma(x', y' - \gamma(x, y)}_{<0} \;<\; \gamma(T).$$

Somit hätten wir einen Spannbaum mit einem kleineren Gewicht konstruiert als der des minimalen Spannbaumes, was offensichtlich ein Widerspruch ist. ■

**12. Juni**

Damit haben wir gezeigt, dass wir im Folgenden die Informationen der Matrix $D_h$ durch die Informationen im minimalen Spannbaum $T$ von $G(D_h)$ ersetzen können.

**Definition 2.50** *Seien $D_\ell \leq D_h$ zwei $n \times n$-Distanzmatrizen. Zwei Knoten $v, w \in [1 : n]$ heißen* separabel, *wenn $D_\ell(v, w) \leq D(G(D_h), v, w)$ gilt.*

Die Separabilität von zwei Knoten ist eine nahe liegende Eigenschaft, die wir benötigen werden, um zu zeigen, dass es möglich ist einen ultrametrischen Baum zu konstruieren, in dem der verbindende Pfad sowohl die untere als auch die obere Schranke für den Abstand einhält. Wir werden dies gleich formal beweisen, aber wir benötigen dazu erst noch ein paar Definitionen und Lemmata. Zuerst halten wir aber noch das folgende Korollar fest, das aus dem vorherigen Lemma und der Definition unmittelbar folgt.

**Korollar 2.51** *Seien $D_\ell \leq D_h$ zwei $n \times n$-Distanzmatrizen und sei $T$ ein minimaler Spannbaum von $G(D_h)$. Zwei Knoten $v, w \in [1 : n]$ sind genau dann separabel, wenn $D_\ell(v, w) \leq D(T, v, w)$ gilt.*

Bevor wir den zentralen Zusammenhang zwischen paarweiser Separabilität und der Existenz ultrametrischer Sandwich-Bäume zeigen, benötigen wir noch die folgende Definition.

**Definition 2.52** *Seien $D_\ell \leq D_h$ zwei $n \times n$-Distanzmatrizen und sei $T$ ein minimaler Spannbaum von $G(D_h)$. Eine Kante $(x, y)$ des Pfades $p_T(v, w)$ im minimalen Spannbaum, der $v$ und $w$ verbindet, heißt* link-edge, *wenn sie eine Kante maximalen Gewichtes in $p_T(v, w)$ ist, d.h. wenn*

$$D_h(x, y) = c(p_T(v, w)) = D(T, v, w)$$

*gilt. Mit $\mathrm{Link}(v, w)$ bezeichnen wir die Menge der Link-edges für das Knotenpaar $(v, w)$.*

Anschaulich ist die Link-Edge eines Pfades im zur oberen Schrankenmatrix gehörigen Graphen diejenige, die den maximalen Abstand von zwei Knoten bestimmt, die durch diesen Pfad verbunden werden. Aus diesem Grund werden diese eine besondere Rolle spielen.

**Definition 2.53** *Seien $D_\ell \leq D_h$ zwei $n \times n$-Distanzmatrizen und sei $T$ ein minimaler Spannbaum von $G(D_h)$. Für jede Kante $(x, y) \in E(T)$ im minimalen Spannbaum ist die* cut-weight $\mathrm{CW}(x, y)$ *wie folgt definiert:*

$$\mathrm{CW}(x, y) := \max \left\{ D_\ell(v, w) \ : \ (x, y) \in \mathrm{Link}(x, y) \right\}.$$

Die cut-weight einer Kante ist der maximale Mindestabstand zweier Knoten, deren Verbindungspfad diese Kante als link-edge besitzt. Um ein wenig mehr Licht in die Bedeutung der cut-weight zu bringen, betrachten wir jetzt nur die maximale auftretende cut-weight eines minimalen Spannbaumes des zu den Maximalabständen gehörigen Graphen.

Für diese maximale cut-weight $c^*$ gilt dann $c^* = \mathrm{MAX}(D_\ell)$, wie man sich leicht überlegt. Wie im einfachen Algorithmus werden wir jetzt versuchen alle schwereren Kanten in $G_{k\ell}$ (der ja ein Teilgraph von $G(D_h)$ ist) zu entfernen. Statt $G_{k\ell}$ betrachten wir jedoch den minimalen Spannbaum $T$ von $G(D_h)$ (der ja nach Definition auch ein Teilgraph von $G(D_H)$ ist).

In $G_{k\ell}$ werden alle schwereren Kanten entfernt, damit $G_{k\ell}$ in mehrere Zusammenhangskomponenten zerfällt. Zuerst überlegt man sich, dass es auch genügen würde, so viele von den schwersten Kanten zu entfernen, bis zwei Zusammenhangskomponenten entstehen. Dies wäre jedoch algorithmisch sehr aufwendig und würde in der Regel keinen echten Zeitvorteil bedeuten. Im minimalen Spannbaum $T$ lässt sich

dies hingegen sehr leicht durch Entfernen der Kante mit der maximalen cut-weight erzielen. Im Beweis vom übernächsten Lemma werden wir dies noch genauer sehen.

Das folgende Lemma stellt noch einen fundamentalen Zusammenhang zwischen Kantengewichten in Kreisen zu additiven Matrizen gehörigen gewichteten Graphen und der Eigenschaft einer Ultrametrik dar, die wir im Folgenden benötigen, um leicht von einer additiven Matrix nachweisen zu können, dass sie bereits ultrametrisch ist.

**Lemma 2.54** *Eine additive Matrix $M$ ist genau dann ultrametrisch ist, wenn für jede Folge $(i_1, \ldots, i_k) \in \mathbb{N}^k$ paarweise verschiedener Werte mit $k \geq 3$ gilt, dass in der Folge $(M(i_1, i_2), M(i_2, i_3), \ldots, M(i_{k-1}, i_k), M(i_k, i_1))$ das Maximum mehrfach angenommen wird.*

**Beweis:** **$\Leftarrow$:** Sei $M$ eine additive Matrix und es gelte für alle $k \geq 3$ und für alle Folgen $(i_1, \ldots, i_k) \subset \mathbb{N}^k$ mit paarweise verschiedenen Folgengliedern, dass

$$\max\{M(i_1, i_2), M(i_2, i_3), \ldots, M(i_{k-1}, i_k), M(i_k, i_1)\} \tag{2.1}$$

nicht eindeutig ist.

Wenn man in (2.1) $k = 3$ einsetzt, gilt insbesondere für beliebige $a, b, c \in \mathbb{N}$, dass

$$\max\{M(a, b), M(b, c), M(c, a)\}$$

nicht eindeutig ist und damit ist $M$ also ultrametrisch.

**$\Rightarrow$:** Sei $M$ nun ultrametrisch, dann ist $M$ auch additiv. Wir müssen nur noch (2.1) zu zeigen. Sei $S = (i_1, \ldots, i_k) \in \mathbb{N}^k$ gegeben. Wir betrachten zunächst $i_1$, $i_2$ und $i_3$. Aus der fundamentalen Eigenschaft einer Ultrametrik wissen wir, dass das Maximum von $(M(i_1, i_2), M(i_2, i_3), M(i_3, i_1))$ nicht eindeutig ist. Wir beweisen (2.1) per Induktion:

Gilt für $j$, dass das Maximum von $(M(i_1, i_2), M(i_2, i_3), \ldots, M(i_{j-1}, i_j), M(i_j, i_1))$ nicht eindeutig ist, so gilt dies auch für $j + 1$. Dazu betrachten wir $i_1$, $i_j$ und $i_{j+1}$. Weiter wissen wir, dass $\max\{M(i_1, i_j), M(i_j, i_{j+1}), M(i_{j+1}, i_1)\}$ nicht eindeutig ist, d.h. einer der Werte ist kleiner als die anderen beiden. Betrachten wir wie sich das Maximum ändert, wenn wir $M(i_j, i_1)$ herausnehmen und dafür $M(i_j, i_{j+1})$ und $M(i_{j+1}, i_1)$ dazugeben.

**Fall 1:** $M(i_1, i_j)$ ist der kleinste Wert. Durch das Hinzufügen von $M(i_j, i_{j+1})$ und $M(i_{j+1}, i_1)$ kann das Maximum nicht eindeutig werden, denn beide Werte sind gleich. Liegen sie unter dem alten Maximum ändert sich nichts, liegen sie darüber, bilden beide das nicht eindeutige neue Maximum. Das Entfernen von $M(i_1, i_j)$ spielt nur eine Rolle, falls $M(i_1, i_j)$ vorher ein Maximum war. In dem Fall sind aber $M(i_j, i_{j+1})$ und $M(i_{j+1}, i_1)$ das neue, nicht eindeutige Maximum.

**Fall 2:** $M(i_{j+1}, i_1)$ ist der kleinste Wert. Dann ist $M(i_1, i_j) = M(i_j, i_{j+1})$. Das Entfernen von $M(i_1, i_j)$ wird durch das Hinzufügen von $M(i_j, i_{j+1})$ wieder ausgeglichen. $M(i_{j+1}, i_1)$ wird auch hinzugefügt, spielt aber keine Rolle.

**Fall 3:** $M(i_j, i_{j+1})$ ist der kleinste Wert. Dann ist $M(i_1, i_j) = M(i_1, i_{j+1})$. Dieser Fall ist analog zu Fall 2. ■

Nun kommen wir zum Beweis unseres zentralen Lemmas, dass es genau dann einen ultrametrischen Baum für eine gegebene Sandwich-Bedingung gibt, wenn alle Knoten paarweise separabel sind. Der Bewies in die eine Richtung wird konstruktiv sein und wird uns somit einen effizienten Algorithmus an die Hand geben.

**Lemma 2.55** *Seien $D_\ell \leq D_h$ zwei $n \times n$-Distanzmatrizen. Ein ultrametrischer Baum $U \in [D_\ell, D_h]$ existiert genau dann, wenn jedes Paar $v, w \in [1 : n]$ von Knoten separabel ist.*

**Beweis:** **⇒:** Für einen Widerspruchsbeweis nehmen wir an, dass $v$ und $w$ nicht separabel sind. Dann ist $D_\ell(v, w) > D_h(x, y)$ für alle $\{x, y\} \in \text{Link}(v, w)$. Für jede Kante $\{a, b\}$ in $p_T(v, w)$ gilt dann $D_h(a, b) \leq D_h(x, y)$ für alle $\{x, y\} \in \text{Link}(v, w)$. Also ist $\{v, w\} \notin p_T(v, w)$. Damit ist $D_\ell(x, y) > D_h(a, b)$ für alle $\{a, b\} \in \text{Link}(v, w)$. Damit muss die Kante $\{x, y\}$ im Kreis, gebildet aus $p_T(x, y)$ und der Kante $\{x, y\}$, in einer ultrametrischen Matrix das eindeutige Maximum sein. Dies steht jedoch im Widerspruch zu Lemma 2.54 und diese Implikation ist bewiesen.

**⇐:** Sei $T$ ein minimaler Spannbaum von $G(D_h)$ und sei $E(T) = \{e_1, \ldots, e_{n-1}\}$, wobei $\text{CW}(e_1) \geq \cdots \geq \text{CW}(e_{n-1})$. Wir entfernen jetzt sukzessive die Kanten aus $T$ gemäß der cut-weight der Kanten. Dabei wird durch jedes Entfernen ein Baum in zwei neue Bäume zerlegt.

Betrachten wir jetzt nachdem Entfernen der Kanten $e_1, \ldots, e_{i-1}$ den entstandenen Wald und darin den Baum $T'$, der die Kante $e_i = \{v, w\}$ enthält. Das Entfernen der Kante $e_i = \{v, w\}$ zerlegt den Baum $T'$ in zwei Bäume $T_1'$ und $T_2'$. Wir setzten dann $d_U(v, w) := \text{CW}(e_i)$ für alle $v \in V(T_1')$ und $w \in V(T_2')$.

Wir zeigen als erstes, dass $d_U(v, w) \geq D_\ell(v, w)$ für alle $v, w \in [1 : n]$. Wir betrachten die Kante $e$, nach deren Entfernen die Knoten $v$ und $w$ im Rest des minimalen Spannbaums nicht mehr durch einen Pfad verbunden sind. Ist $e$ eine link-edge in $p_T(v, w)$, dann gilt nach Definition der cut-weight: $\text{CW}(e) \geq D_\ell(v, w)$. Ist $e$ hingegen keine link-edge in $p_T(v, w)$, dann gilt $\text{CW}(e) \geq \text{CW}(e')$ für jede link-edge $e' \in \text{Link}(v, w)$, da wir die Kanten gemäß ihrer absteigenden cut-weight aus dem minimalen Spannbaum $T$ entfernen. Somit gilt nach Definition der cut-weight:

$$\text{CW}(e) \geq \text{CW}(e') \geq D_\ell(v, w).$$

Wir zeigen jetzt, dass ebenfalls $d_U(v,w) \leq D_h(v,w)$ für alle $v,w \in [1:n]$ gilt. Nach Definition der cut-weight gilt, dass ein Knotenpaar $\{x,y\}$ existiert, so dass für alle $\{a,b\} \in \text{Link}(x,y)$ gilt: $D_\ell(x,y) = \text{CW}(a,b)$. Da $x$ und $y$ nach Voraussetzung separabel sind, gilt

$$\text{CW}(a,b) = D_\ell(x,y) \leq D(T,x,y) = D_h(a,b).$$

Die letzte Gleichheit folgt aus der Tatsache, dass $\{a,b\} \in \text{Link}(x,y)$. Aufgrund der Konstruktion des minimalen Spannbaumes gilt weiterhin $D_h(a,b) \leq D_h(v,w)$. Somit gilt $d_U(v,w) = \text{CW}(a,b) \leq D_h(v,w)$.

Damit haben wir gezeigt, dass $U \in [D_\ell, D_h]$. Wir müssen zum Schluss nur noch zeigen, dass $U$ ultrametrisch ist. Nach Lemma 2.54 genügt es zu zeigen, dass in jedem Kreis in $G(U)$ das Maximum nicht eindeutig ist. Sei also $C$ ein Kreis in $G(U)$ und $(v,w)$ eine Kante maximalen Gewichtes in $C$. Falls es mehrere davon geben sollte, wählen wir eine solche, deren Endpunkte in unserem Konstruktionsverfahren durch Entfernen von Kanten im minimalen Spannbaum $T$ zuerst separiert wurden.

Wir betrachten jetzt den Zeitpunkt in unserem Konstruktionsverfahren von $d_U$, als $d_U(v,w)$ gesetzt wurde. Zu diesem Zeitpunkt wurde $v$ und $w$ in zwei Bäume $T'$ und $T''$ aufgeteilt. Ferner sind die Knoten dieses Kreises nach Wahl der Kante $\{v,w\}$ alle Knoten des Kreises in diesen beiden Teilbäumen enthalten. Da es in $C$ noch einen anderen Weg von $v$ nach $w$ gibt, muss zu diesem Zeitpunkt auch für eine andere Kante $\{v',w'\}$ der Wert $d_U(v',w')$ ebenfalls festgelegt worden sein. Nach unserer Konstruktion gilt dann natürlich $d_U(v,w) = d_U(v',w')$ und in $C$ ist das Maximum nicht eindeutig. ■

**17. Juni**

### 2.5.4 Algorithmus für das ultrametrische Sandwich-Problem

Der Beweis des vorherigen Lemmas liefert unmittelbar den folgenden Algorithmus, der in Abbildung 2.42 angegeben ist. Die Korrektheit des Algorithmus folgt im Wesentlichen aus dem Beweis des vorherigen Lemmas (wir gehen später noch auf ein paar implementierungstechnische Besonderheiten ein). Wir müssen uns nur noch um die effiziente Implementierung kümmern. Einen Algorithmus zur Bestimmung minimaler Spannbäume haben wir bereits kennen gelernt. Wir werden hier noch eine andere Möglichkeit darstellen, die für unsere Zwecke besser geeignet ist. Ebenso müssen wir uns noch um die effiziente Berechnung der cut-weights kümmern. Außerdem müssen wir noch erklären was kartesische Bäume sind und wie wir sie konstruieren können.

1. Bestimme minimalen Spannbaum $T$ für $G(D_h)$. $O(n^2)$
2. Bestimme dabei den Kartesischen Baum $R$ für den Zusammenbau von $T$ $O(n^2)$
3. Bestimme den cut-weight der einzelnen Kanten aus $T$ mit Hilfe des Kartesischen Baumes. $O(n^2)$
4. Baue den minimalen Spannbaum durch Entfernen der Kanten absteigend nach den cut-weights der Kanten ab und bauen parallel den ultrametrischen Baum U wieder auf. $O(n)$

Abbildung 2.42: Algorithmus: Effiziente Lösung des ultrametrische Sandwich-Problems

#### 2.5.4.1 Kruskals Algorithmus für minimale Spannbäume

Zuerst stellen wir eine alternative Methode zu Prims Algorithmus zur Berechnung minimaler Spannbäume vor. Auch hier werden wir wieder den Greedy-Ansatz verwenden. Wir beginnen jedoch anstatt mit einem Baum aus einem Knoten, den wir sukzessive zu einem Spannbaum erweitern, mit einem Wald von Bäumen, die zu Beginn aus einelementigen Bäumen bestehen. Dabei stellt jeder Knoten genau einen Baum dar. Dann fügen wir sukzessive Kanten in diesem Wald hinzu, um dabei zwei Bäume mittels dieser Kante zu einem neuen größeren Baum zu verschmelzen. Nachdem wir $n-1$ Kanten hinzugefügt haben, haben wir unseren Spannbaum konstruiert.

Da wir gierig vorgehen, d.h. leichte Kanten bevorzugen, werden wir zuerst die Kanten aufsteigend nach Gewicht sortieren und versuchen diese in dieser Reihenfolge zu verwenden. Dabei müssen wir nur beachten, dass wir keine Kreise generieren (da Bäume durch Kreisfreiheit und Zusammenhang charakterisiert sind). Dazu merken wir uns mit einer so genannten *Union-Find-Datenstruktur* wie die Mengen der Knoten auf die verschiedenen Menge im Wald verteilt sind. Wenn wir feststellen, dass eine Kante innerhalb eines Baumes (also innerhalb einer Menge) verlaufen würde, dann verwerfen wir sie, andernfalls wird sie aufgenommen. Dieser Algorithmus ist im Detail in Abbildung 2.43 angegeben.

Halten wir noch das Ergebnis fest, wobei wir im nächsten Teilabschnitt noch zeigen werden, dass $\mathcal{UF}(n) = O(n^2)$ ist.

**Lemma 2.56** *Sei $G = (V, E, \gamma)$ ein gewichteter Graph. Ein minimaler Spannbaum $T$ für $G$ kann mit Hilfe des Kruskal-Algorithmus in Zeit $O(m \log(m) + n^2 + \mathcal{UF}(n))$ berechnet werden, wobei $\mathcal{UF}(n)$ die Zeit ist, die eine Union-Find-Datenstruktur bei $n^2$ Find- und $n$ Union-Operationen benötigt.*

```
KRUSKAL
{
    /* O.B.d.A. gelte γ(e_1) ≤ ··· ≤ γ(e_m) mit E = {e_1, ..., e_m} */
    set E' = ∅;
    /* (V, E') wird der konstruierte minimale Spannbaum sein */
    for (i = 1; i ≤ m; i++)
    {
        Sei e_i = {v, w};
        k = FIND(v);
        ℓ = FIND(w);
        if (k ≠ ℓ)
        {
            E' = E' ∪ {e_i};
            UNION(k, ℓ);
            if (|E'| = |V| − 1)
                return (V, E');
        }
    }
}
```

Abbildung 2.43: Algorithmus: Kruskals Algorithmus für minimale Spannbäume

Leider enthält die Laufzeit den Term $O(m \log(m))$, der für das Sortieren der Kantengewichte benötigt wird. Somit hilft uns Kruskals Algorithmus nur, wenn die Anzahl der Kanten des gegebenen Graphen subquadratisch sind, genauer, wenn $m = O(n^2/log(n))$ gilt.

Für unser Sandwich-Problem ist dies nur hilfreich, wenn die gegebenen Schranken-Distanzmatrizen nicht vollständig sind (in der oberen bzw. unteren Schrankenmatrix werden unbesetzte Einträge als $\infty$ bzw. $-\infty$ interpretiert). Man kann sich überlegen (was wir hier nicht tun wollen, siehe Originalliteratur), dass sich der Algorithmus dann nur auf die definiten Einträge beschränken kannund trotzdem weiterhin korrekt bleibt.

Kruskals Algorithmus funktioniert auch in Zeit $O(n^2)$, wenn die zu sortierenden Kantengewicht aus einem sehr begrenzten Zahlenbereich stammen, so dass wir effizientere Sortierverfahren wie Bucket- oder Radixsort anstelle von vergleichsbasierten Sortierverfahren verwenden können.

Zwar ist das Verfahren von Kruskal im folgenden einfacher für die Konstruktion von Kartesischen Bäumen anwendbar, aber wir können die Kartesischen Bäume für unsere Zwecke auch rekursiv mit Hilfe von Prims Algorithmus in Zeit $O(n^2)$ konstruieren.

#### 2.5.4.2 Union-Find-Datenstruktur

Jetzt müssen wir noch genauer auf die so genannte Union-Find-Datenstruktur eingehen. Eine Union-Find-Datenstruktur für eine Grundmenge $U$ beschreibt eine Partition $\mathcal{P} = \{P_1, \ldots, P_\ell\}$ für $U$ mit $U = \bigcup_{i=1}^{\ell} P_i$ und $P_i \cap P_j = \emptyset$ für alle $i \neq j \in [1 : \ell]$. Dabei ist zu Beginn $\mathcal{P} = \{P_1, \ldots, P_{|U|}\}$ mit $P_u = \{u\}$ für alle $u \in U$. Weiterhin werden die beiden folgenden elementaren Operationen zur Verfügung gestellt:

**Find-Operation:** Gibt für eine Element $u \in U$ den Index der Menge in der Mengenpartition zurück, die $u$ enthält.

**Union-Operation:** Vereinigt die beiden Menge mit Index $i$ und Index $j$ und vergibt für diese vereinigte Menge einen neuen Index.

Die Realisierung erfolgt durch zwei Felder. Dabei nehmen wir der Einfachheit halber an, dass $U = [1 : n]$ ist. Ein Feld von ganzen Zahlen namens Index gibt für jedes Element $u \in U$ an, welchen Index die Menge besitzt, die $u$ enthält. Ein weiteres Feld von Listen namens Liste enthält für jeden Mengenindex eine Liste von Elementen, die in der entsprechenden Menge enthalten sind.

Die Implementierung der Prozedur Find ist nahe liegend. Es wird einfach der Index zurück gegeben, der im Feld Index gespeichert ist. Für die Union-Operation werden wir die Elemente einer Menge in die andere kopieren. Die umkopierte Menge wird dabei gelöscht und der entsprechende Index der wiederverwendeten Menge wird dabei recycelt. Um möglichst effizient zu sein, werden wir die Elemente der kleineren Menge in die größere Menge kopieren. Die detaillierte Implementierung ist in Abbildung 2.44 angegeben.

Wir überlegen uns jetzt noch die Laufzeit dieser Union-Find-Datenstruktur. Hierbei nehmen wir an, dass wir $k$ Find-Operationen ausführen und maximal $n - 1$ Union-Operationen. Mehr Union-Operationen machen keinen Sinn, da sich nach $n - 1$ Union-Operationen alle Elemente in einer Menge befinden.

Offensichtlich kann jede Find-Operation in konstanter Zeit ausgeführt werden. Für die Union-Operation ist der Zeitbedarf proportional zur Anzahl der Elemente in der kleineren Menge, die in der Union-Operation beteiligt ist. Somit ergibt sich für die maximal $n - 1$ möglichen Union-Operationen:

$$\sum_{\sigma=(L,L')} \sum_{\substack{i \in L \\ |L| \leq |L'|}} O(1).$$

Um diese Summe jetzt besser abschätzen zu können vertauschen wir die Summationsreihenfolge. Anstatt die äußere Summe über die Union-Operation zu betrachten,

```
UNION-FIND
function INITIALIZE(int n)
{
    int Index[n];
    for (i = 1; i ≤ n; i++)
        Index[i] = i;
    <int> Liste[n];
    for (i = 1; i ≤ n; i++)
        Liste[i] = <i>;
}

function FIND(int u)
{
    return Index[u];
}

function UNION(int i, j)
{
    if (Liste[i].size() ≤ Liste[j].size())
    {
        for all (u ∈ Liste[i]) do
        {
            Liste[j].add(u);
            Index[u] = j;
            Liste[i].remove(u);
        }
    }
    else
    {
        for all (u ∈ Liste[j]) do
        {
            Liste[i].add(u);
            Index[u] = i;
            Liste[j].remove(u);
        }
    }
}
```

Abbildung 2.44: Algorithmus: Union-Find

summieren wir für jedes Element in der Grundmenge, wie oft es bei einer Union-Operation in der kleineren Menge sein könnte.

$$\sum_{\sigma=(L,L')} \sum_{\substack{i \in L \\ |L| \leq |L'|}} O(1) = \sum_{u \in U} \sum_{\substack{\sigma=(L,L') \\ u \in L \\ |L| \leq |L'|}} O(1).$$

Was passiert mit einem Element, dass sich bei einer Union-Operation in der kleineren Menge befindet? Danach befindet es sich in einer Menge die mindestens doppelt so groß wie vorher ist, da diese mindestens so viele neue Element in die Menge hinzubekommt, wie vorher schon drin waren. Damit kann jedes Element maximal $\log(n)$ Mal in einer kleineren Menge bei einer Union-Operation gewesen sein, da sich dieses Element dann in einer Menge mit mindestens $n$ Elementen befinden muss.

Da die Grundmenge aber nur $n$ Elemente besitzt, kann danach überhaupt keine Union-Operation mehr ausgeführt werden, da sich dann alle Elemente in einer Menge befinden. Somit ist die Laufzeit für ein Element durch $O(\log(n))$ beschränkt. Da es maximal $n$ Elemente gibt, ist die Gesamtlaufzeit aller Union-Operationen durch $O(n \log(n))$ beschränkt.

**Theorem 2.57** *Sei $U$ mit $|U| = n$ die Grundmenge für die vorgestellte Union-Find-Datenstruktur. Die Gesamtlaufzeit von $k$ Find- und maximal $n-1$ Union-Operationen ist durch $O(k + n\log(n))$ beschränkt.*

Es gibt noch effizienter Implementierungen von Union-Find-Operationen, die wir hier aber nicht benötigen und daher auch nicht näher darauf eingehen wollen. Wir verweisen statt dessen auf die einschlägige Literatur.

#### 2.5.4.3 Kartesische Bäume

Kommen wir nun zur Definition eine kartesischen Baumes.

**Definition 2.58** *Sei $M$ eine Menge von Objekten, $E \subset \binom{M}{2}$ eine Menge von Paaren, so dass der Graph $(M, E)$ ein Baum ist und $\gamma : E \to \mathbb{R}$ eine Gewichtsfunktion auf $E$. Ein* kartesischer Baum *für $(M, E)$ ist rekursiv wie folgt definiert.*

- *Ist $|M| = 1$, dann ist der einelementige Baum mit der Wurzelmarkierung $m \in M$ ein kartesischer Baum.*
- *Ist $|M| \geq 2$ und $\{v_1, v_2\} \in E$ mit $\gamma(\{v_1, v_2\}) = \max\{\gamma(e) : e \in E\}$. Sei weiter $T'$ der Wald, der aus $T$ durch Entfernen von $\{v_1, v_2\}$ entsteht, d.h. $V(T') = V(T)$ und $E(T') = E(T) \setminus \{v_1, v_2\}$. Sind $T_1$ bzw. $T_2$ kartesische Bäume für $C(T', v_1)$ bzw. $C(T', v_2)$, dann ist der Baum mit der Wurzel, die mit $\{v_1, v_2\}$ markiert ist und deren Teilbäume der Wurzel gerade $T_1$ und $T_2$ sind, ebenfalls ein kartesischer Baum.*

Wir bemerken hier noch explizit an, dass die Elemente aus $M$ nur als Blattmarkierungen auftauchen und dass alle inneren Knoten mit den Elementen aus $E$ markiert sind.

Betrachtet man auf der Menge $[1 : n]$ als Baum eine lineare Liste mit

$$E = \{\{i, i+1\} : i \in [1 : n-1]\} \quad \text{mit} \quad \gamma(i, i+1) = \max\{F[i], F[i+1]\},$$

wobei $F$ ein Feld von Zahlen ist, so erhält man im Wesentlichen einen Heap, in dem die Werte an den Blättern stehen und die inneren Knoten den maximalen Wert ihres Teilbaumes besitzen, wenn man, wie hier üblich, anstatt der Kante in den inneren Knoten das Gewicht der Kante einträgt. Diese Struktur wird manchmal auch als kartesischer Baum bezeichnet.

Der kartesische Baum für einen minimalen Spannbaum lässt sich jetzt bei der Konstruktion mit Hilfe des Kruskal-Algorithmus sehr leicht mitkonstruieren. Man überlegt sich leicht, dass bei der Union-Operation, die zugehörigen kartesischen Bäume mit einer neuen Wurzel vereinigt werden, wobei die Kante in der Wurzel genau die Kante ist, die bei der Konstruktion des minimalen Spannbaumes hinzugefügt wird.

**Lemma 2.59** *Sei $G = (V, E, \gamma)$ ein gewichteter Graph und $T$ ein minimaler Spannbaum von $G$. Der kartesische Baum für $T$ kann bei der Konstruktion des minimalen Spannbaumes $T$ basierend auf Kruskals Algorithmus parallel mit derselben Zeitkomplexität mitkonstruiert werden.*

#### 2.5.4.4 Berechnung der cut-weights

Wir wollen jetzt zeigen, dass wir die cut-weights einer Kante $e \in E$ im minimalen Spannbaum $T$ mit Hilfe von lca-Anfragen im kartesischen Baum $T$ bestimmen können. Dazu gehen wir durch die Matrix $D_\ell$ und bestimmen für jedes Knotenpaar $(i, j)$ den niedrigsten gemeinsamen Vorfahren $\text{lca}(i, j)$ im kartesischen Baum $R$.

Die dort gespeicherte Kante $e$ ist nach Konstruktion des kartesischen Baumes eine Kante mit größtem Gewicht auf dem Pfad von $i$ nach $j$ im minimalen Spannbaum $T$, d.h. $e \in \text{Link}(i, j)$. Daher werden wir dort die cut-weight $\text{CW}(e)$ dieser Kante mittels $\text{CW}(e) = \max\{\text{CW}(e), D_\ell(i, j)\}$ aktualisieren. Wir bemerken an dieser Stelle, dass es durchaus noch andere link-edges auf dem Pfad von $i$ nach $j$ im minimalen Spannbaum geben kann. Daher wird die cut-weight nicht für jede Kante korrekt berechnet. Wir gehen auf diesen „Fehler“ am Ende diese Abschnittes noch ein.

**Lemma 2.60** *Sei $G = (V, E, \gamma)$ ein gewichteter Graph und $T$ ein minimaler Spannbaum von $G$. Der kartesische Baum $R$ für den minimalen Spannbaum $T$ kann mit derselben Zeit konstruiert werden wie der minimale Spannbaum, sofern hierfür der Algorithmus von Kruskal verwendet wird.*

Nachdem wir jetzt die wesentlichen Schritte unseres effizienten Algorithmus weitestgehend verstanden haben, können wir uns einem Beispiel zuwenden. In der Abbildung 2.45 ist ein Beispiel angegeben, wie unser effizienter Algorithmus vorgeht.

#### 2.5.4.5 Least Common Ancestor Queries

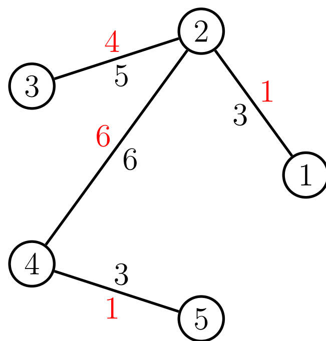
Wir müssen uns nun nur noch überlegen, wie wir $O(n^2)$ lca-Anfragen in Zeit $O(n^2)$ beantworten können. Dazu werden wir das lca-Problem auf das Range Minimum Query Problem reduzieren, das wie folgt definiert ist.

RANGE MINIMUM QUERY

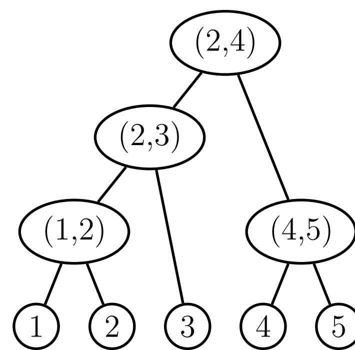
**Eingabe:** Eine Feld $F$ der Länge $n$ von reellen Zahlen und $i \leq j \in [1 : n]$.
**Gesucht:** Ein Index $k$ mit $F[k] = \min\{F[\ell] : \ell \in [i : j]\}$.

| $D_\ell$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 2 | 3 | 6 |
| 2 | | 0 | 4 | 5 | 5 |
| 3 | | | 0 | 4 | 5 |
| 4 | | | | 0 | 1 |
| 5 | | | | | 0 |

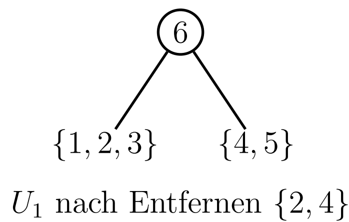
| $D_h$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 3 | 6 | 8 | 8 |
| 2 | | 0 | 5 | 6 | 8 |
| 3 | | | 0 | 6 | 8 |
| 4 | | | | 0 | 3 |
| 5 | | | | | 0 |



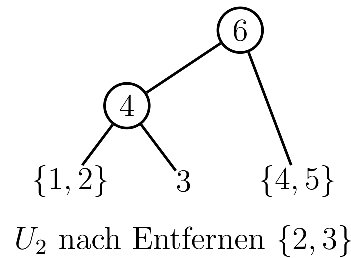

Minimaler Spannbaum $T$



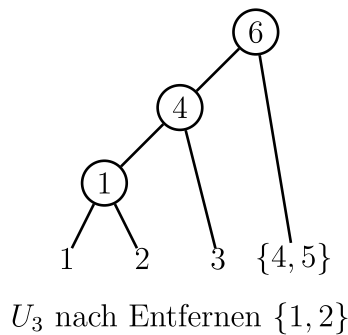

Kartesischer Baum $R$



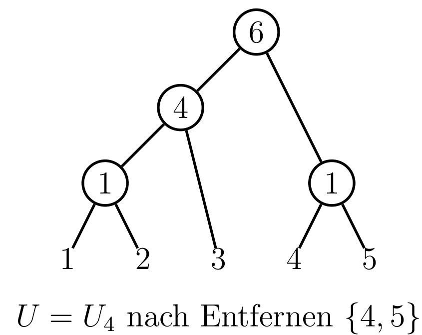

$U_1$ nach Entfernen $\{2, 4\}$




$U_2$ nach Entfernen $\{2, 3\}$




$U_3$ nach Entfernen $\{1, 2\}$




$U = U_4$ nach Entfernen $\{4, 5\}$

Abbildung 2.45: Beispiel: Lösung eines ultrametrischen Sandwich-Problems

Wir werden später zeigen, wie wir mit einem Preprocessing in Zeit $O(n^2)$ jede Anfrage in konstanter Zeit beantworten können. Für die Reduktion betrachten wir die so genannte *Euler-Tour* eines Baumes.

**Definition 2.61** *Sei $T = (V, E)$ ein gewurzelter Baum mit Wurzel $r$ und seien $T_1, \ldots, T_\ell$ die Teilbäume, die an der Wurzel hängen. Die* Euler-Tour *durch $T$ ist eine Liste von $2n - 1$ Knoten, die wie folgt rekursiv definiert ist:*

- *Ist $\ell = 0$, d.h. der Baum besteht nur aus dem Blatt $r$, dann ist diese Liste durch $(r)$ gegeben.*
- *Für $\ell \geq 1$ seien $L_1, \ldots, L_\ell$ mit $L_i = (v_1^{(i)}, \ldots, v_{n_i}^{(i)})$ für $i \in [1 : \ell]$ die Euler-Touren von $T_1, \ldots, T_\ell$. Die Euler-Tour von $T$ ist dann durch*

  $$(r, v_1^{(1)}, \ldots, v_{n_1}^{(1)}, r, v_1^{(2)}, \ldots, v_{n_2}^{(2)}, r, \ldots, r, v_1^{(\ell)}, \ldots, v_{n_\ell}^{(\ell)}, r)$$

  *definiert.*

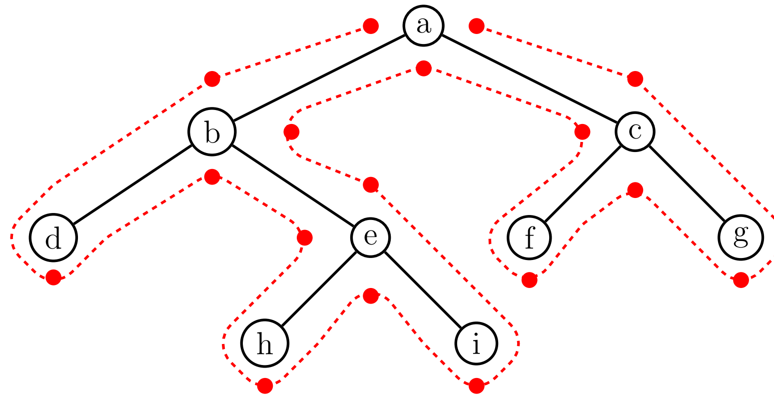
Der Leser sei dazu aufgefordert zu verifizieren, dass die oben definierte Euler-Tour eines Baumes mit $n$ Knoten tatsächlich eine Liste mit $2n - 1$ Elementen ist.

Die Euler-Tour kann sehr leicht mit Hilfe einer Tiefensuche in Zeit $O(n)$ berechnet werden. Der Algorithmus hierfür ist in Abbildung 2.46 angegeben. Man kann sich die

```
Euler-Tour
{
    /* r(T) bezeichne die Wurzel von T */
    output r(T);
    /* N(r(T)) bezeichne die Menge der Kinder der Wurzel von T */
    for all (v ∈ N(r(T)))
    {
        /* T(v) bezeichne den Teilbaum mit Wurzel v */
        Euler-Tour(T(v))
        output r(T);
    }
}
```

Abbildung 2.46: Algorithmus: Konstruktion einer Euler-Tour

Euler-Tour auch bildlich sehr schön als das Abmalen der Bäume anhand ihrer äußeren Kontur vorstellen, wobei bei jedem Antreffen eines Knotens des Baumes dieser in die Liste aufgenommen wird. Die ist in Abbildung 2.47 anhand eines Beispiels illustriert.




| Euler-Tour | a | b | d | b | e | h | e | i | e | b | a | c | f | c | g | c | a |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DFS-Nummer | 1 | 2 | 3 | 2 | 4 | 5 | 4 | 6 | 4 | 2 | 1 | 7 | 8 | 7 | 9 | 7 | 1 |

Abbildung 2.47: Beispiel: Euler-Tour

Zusammen mit der Euler-Tour, der Liste der abgelaufenen Knoten, betrachten wir zusätzlich noch DFS-Nummern des entsprechenden Knotens, die bei der Tiefensuche in der Regel mitberechnet werden (siehe auch das Beispiel in der Abbildung 2.47).

Betrachten wir jetzt die Anfrage an zwei Knoten des Baumes $i$ und $j$. Zuerst bemerken wir, dass diese Knoten, sofern sie keine Blätter sind, mehrfach vorkommen können. Wir wählen jetzt für $i$ und $j$ willkürlich einen der Knoten, der in der Euler-Tour auftritt, als einen Repräsentanten aus. Zuerst stellen wir fest, dass in der Euler-Tour der niedrigste gemeinsame Vorfahren von $i$ und $j$ in der Teilliste, die durch die beiden Repräsentanten definiert ist, vorkommen muss, was man wie folgt sieht.

Wir nehmen an, dass $i$ in der Euler-Tour vor $j$ auftritt. In der Euler-Tour sind alle Knoten $v$, die nach einem Repräsentanten von $i$ auftauchen und eine kleinere DFS-Nummer als $i$ besitzen, ein Vorfahre von $i$. Da die DFS-Nummer von $v$ kleiner als die von $i$ ist und $v$ in der Euler-Tour nach $i$ auftritt, ist die DFS-Prozedur von $v$ noch aktiv, als der Knoten $i$ besucht wird. Das ist genau die Definition dafür, dass $i$ ein Nachfahre von $v$ ist. Analoges gilt für den Knoten $j$.

Wir betrachten jetzt nur die Knoten der Teilliste zwischen den beiden Repräsentanten von $i$ und $j$, deren DFS-Nummer kleiner als die von $i$ ist. Nach dem obigen sind dies Vorfahren von $i$. Nur einer davon, nämlich der mit der kleinsten DFS-Nummer, ist auch ein Vorfahre von $j$ und muss daher der niedrigste gemeinsame Vorfahre von $i$ und $j$ sein. Diesen Knoten haben wir nämlich besucht, bevor wir in den Teilbaum von $j$ eingedrungen sind. Alle Vorfahren davon haben wir mit Betrachten der Teilliste eliminiert, die mit einem Repräsentanten von $j$ endet.

Damit können wir das folgende Zwischenergebnis festhalten.

**Lemma 2.62** *Gibt es eine Lösung für das Range Minimum Query Problem, dass für das Preprocessing Zeit $O(p(n))$ und für eine Anfrage $O(q(n))$ benötigt, so kann das Problem des niedrigsten gemeinsamen Vorfahren mit einen Zeitbedarf für das Preprocessing in Zeit $O(n + p(2n - 1))$ und für eine Anfrage in Zeit $O(q(2n - 1))$ gelöst werden.*

#### 2.5.4.6 Range Minimum Queries

Damit können wir uns jetzt ganz auf das Range Minimum Query Problem konzentrieren. Offensichtlich kann ohne ein Preprocessing eine einzelne Anfrage mit $O(j-i) = O(n)$ Vergleichen beantwortet werden. Das Problem der Range Minimum Queries ist jedoch insbesondere dann interessant, wenn für ein gegebenes Feld eine Vielzahl von Range Minimum Queries durchgeführt werden. In diesem Fall können mit Hilfe einer Vorverarbeitung die Kosten der einzelnen Queries gesenkt werden.

Ein triviale Lösung würde alle möglichen Anfragen vorab berechnen. Dazu könnte eine zweidimensionale Tabelle $Q[i, j]$ angelegt werden. Dazu würde für jedes Paar $(i, j)$ das Minimum der Bereichs $F[i : j]$ mit $j - i$ Vergleichen bestimmt werden. Dies würde zu einer Laufzeit für die Vorverarbeitung von

$$\sum_{i=1}^{n} \sum_{j=i}^{n} (j - i) = \Theta(n^3)$$

führen. In der Abbildung 2.48 ist ein einfacher, auf dynamischer Programmierung basierender Algorithmus angegeben, der diese Tabelle in Zeit $O(n^2)$ berechnen kann. Damit erhalten wir das folgende Resultat für das Range Minimum Query Problem.

**Theorem 2.63** *Für das Range Minimum Query Problem kann mit Hilfe einer Vorverarbeitung, die mit einem Zeitbedarf von $O(n^2)$ ausgeführt werden kann, jede Anfrage in konstanter Zeit beantwortet werden.*

Es gibt bereits wesentlich effizientere Verfahren für das Range Minimum Query Problem. In unserem Zusammenhang ist dieses leicht zu erzielende Ergebnis jedoch bereits völlig ausreichend und wir verweisen für die anderen Verfahren auf die einschlägige Literatur. Wir halten das für uns wichtige Ergebnis noch fest.

**Theorem 2.64** *Sei $T$ ein gewurzelter Baum mit $n$ Knoten. Nach einer Vorverarbeitung, die in Zeit $O(n^2)$ durchgeführt werden kann, kann jede Anfrage nach einem niedrigsten gemeinsamen Vorfahren zweier Knoten aus $T$ in konstanter Zeit beantwortet werden.*

**24. Juni**

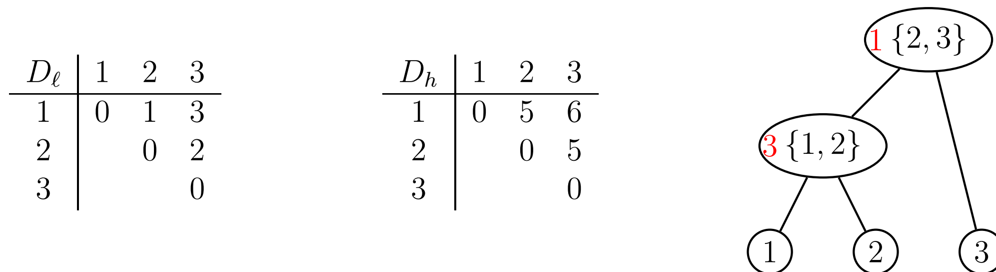
```
RMQ
{
    for (i = 1; i ≤ n; i++)
        T[i, i] = i;

    for (i = 1; i ≤ n; i++)
        for (j = 1; i + j ≤ n; j++)
        {
            if (F[T[i, i + (j − 1)]] ≤ F[i + j])
                T[i, i + j] = T[i, i + j − 1];
            else
                T[i, i + j] = i + j;
        }
}
```

Abbildung 2.48: Algorithmus: Preprocessing für Range Minimum Queries

#### 2.5.4.7 Reale Berechnung der cut-weights

Wie bereits schon angedeutet berechnet unser effizienter Algorithmus nicht wirklich die cut-weights der Kanten des minimalen Spannbaumes. Dies passiert genau dann, wenn es im minimalen Spannbäume mehrere Kanten desselben Gewichtes gibt. Dazu betrachten wir das Beispiel, das in Abbildung 2.49 angegeben ist.



| $D_\ell$ | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 0 | 1 | 3 |
| 2 | | 0 | 2 |
| 3 | | | 0 |

| $D_h$ | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 0 | 5 | 6 |
| 2 | | 0 | 5 |
| 3 | | | 0 |



Abbildung 2.49: Beispiel: Falsche Berechnung der cut-weights

Hier stellen wir fest, das alle Kanten des minimalen Spannbaumes ein Kantengewicht von 5 besitzen. Somit sind alle Kanten des Baumes link-edges. Zuerst stellen wir fest, dass die cut-weight der Kante $\{2, 3\}$, die in der Wurzel des kartesischen Spannbaumes, mit 3 korrekt berechnet wird, da ja auch die Kante $\{1, 3\}$ der niedrigste gemeinsame Vorfahre von 1 und 3 im kartesischen Baum ist.

Im Gegensatz dazu wird die cut-weight der Kante $\{1, 2\}$ des minimalen Spannbaumes falsch berechnet. Diese Kante erhält die cut-weight 1, da die Kante $\{1, 2\}$ der

niedrigste gemeinsame Vorfahre von 1 und 2 ist. Betrachten wir jedoch den Pfad von 1 über 2 nach 3, dann stellen wir fest, dass auch die Kante $\{1, 2\}$ eine link-edge im Pfad von 1 nach 3 im minimalen Spannbaum ist und die cut-weight der Kante $\{1, 2\}$ im minimalen Spannbaum daher ebenfalls 3 sein müsste.

Eine einfache Lösung wäre es die Kantengewicht infinitesimal so zu verändern, dass alle Kantengewichte im minimalen Spannbaum eindeutig wären. Wir können jedoch auch zeigen, dass der Algorithmus weiterhin korrekt ist, obwohl er die cut-weights von manchen Kanten falsch berechnet.

Zuerst überlegen wir uns, welche Kanten eine falsche cut-weight bekommen. Dies kann nur bei solchen Kanten passieren, deren Kantengewicht im minimalen Spannbaum nicht eindeutig ist. Zum anderen müssen diese Kanten bei der Konstruktion des minimalen Spannbaumes vor den anderen Kanten gleichen Gewichtes im minimalen Spannbaum eingefügt worden sein. Dies bedeutet, dass diese Kante im kartesischen Baum ein Nachfahre einer anderen Kante des minimalen Spannbaum gleichen Gewichtes sein muss.

Überlegen wir uns, was im Algorithmus passiert. Wir entfernen ja die Kanten aus dem minimalen Spannbaum nach fallendem cut-weight. Somit wird von zwei Kanten im minimalen Spannbaum, die dasselbe Kantengewicht besitzen und dieselbe cut-weight besitzen sollten, die Kante entfernt, die sich weiter oben im kartesischen Baum befindet. Damit zerfällt der minimale Spannbaum in zwei Teile und die beiden Knoten der untere Schranke, die für die cut-weight der entfernten Kante verantwortlich sind, befinden sich in zwei verschiedenen Zusammenhangskomponenten.

Somit ist die cut-weight der Kante, die ursprünglich falsch berechnet worden, in der neuen Zusammenhangskomponente jetzt nicht mehr ganz so falsch. Entweder ist sie für die konstruierte Zusammenhangskomponente korrekt und wir können mit unserem Algorithmus fortfahrten und er bleibt korrekt. War sie andernfalls falsch, befindet sich in minimalen Spannbaum dieser Zusammenhangskomponente eine weitere Kante mit demselben Gewicht, die im kartesischen Baum ein Vorfahre der betrachteten Kante ist und die ganze Argumentation wiederholt sich.

Also obwohl der Algorithmus nicht die richtigen cut-weights berechnet, ist zumindest immer die cut-weight der Kante mit der schwersten cut-weight korrekt berechnet und dies genügt für die Korrektheit des Algorithmus völlig, wie eine kurze Inspektion des zugehörigen Beweises ergibt.

### 2.5.5 Approximationsprobleme

Wir kommen jetzt noch einmal zu dem Approximationsproblem für Distanzmatrizen zurück.

Ultrametrisches Approximationsproblem

**Eingabe:** Eine $n \times n$-Distanzmatrizen $D$.
**Gesucht:** Eine ultrametrische Distanzmatrix $D'$, die $\|D - D'\|$ minimiert.

Das entsprechende additive Approximationsproblem ist leider $\mathcal{NP}$-hart. Das ultrametrische Approximationsproblem kann für die Maximumsnorm $\|\cdot\|_\infty$ in Zeit $O(n^2)$ gelöst werden. Für die anderen $p$-Normen ist das Problem ebenfalls wieder $\mathcal{NP}$-hart.

**Theorem 2.65** *Das ultrametrische Approximationsproblem für die Maximumsnorm kann in linearer Zeit (in der Größe der Eingabe) gelöst werden.*

**Beweis:** Sei $D$ die gegebene Distanzmatrix. Eigentlich müssen wir nur ein minimales $\varepsilon > 0$ bestimmen, so dass es eine ultrametrische Matrix $U$ mit $U \in [D - \varepsilon, D + \varepsilon]$ gibt. Hierbei ist $D + x$ definiert durch $D = (d_{i,j} + x)_{i,j}$.

Wir berechnen also zuerst wieder den minimalen Spannbaum $T$ für $G(D)$. Dann berechnen wir die cut-weights für die Kanten des minimalen Spannbaumes $T$. Dabei wählen wir für jede Kante $e$ ein minimales $\varepsilon_e$, so dass die Knotenpaare separabel sind, d.h. $\mathrm{CW}(e) - \varepsilon_e \leq D(e) + \varepsilon_e$ für alle Kanten $e \in E(T)$ .

Damit dies für alle Kanten des Spannbaumes gilt, wählen $\varepsilon$ als das Maximum dieser, d.h.

$$\varepsilon := \max\left\{\varepsilon_e \,:\, e \in E(T)\right\} = \frac{1}{2}\max\left\{\mathrm{CW}(e) - D(e) \,:\, e \in E(T)\right\}.$$

Dann führen wir denselben Algorithmus wie für das ultrametrische Sandwich Problem mit $D_\ell := D - \varepsilon$ und $D_h = D + \varepsilon$ durch. ■

# Kombinatorische Proteinfaltung 3

## 3.1 Inverse Proteinfaltung

In diesem Abschnitt werden wir uns mit der Modellierung der inversen Proteinfaltung beschäftigen. Bei der inversen Proteinfaltung ist die räumliche Struktur des Proteins (besser des Backbones) bekannt, es wird nur die Zuordnung der Aminosäuren auf die einzelnen Positionen gesucht.

### 3.1.1 Grand Canonical Model

Ein erstes, bekanntes Modell für die inverse Proteinfaltung wurde von Sun, Brem, Chan und Dill beschrieben. Hierbei ist die Konformation des betrachteten Proteins vollständig bekannt, d.h. die Positionen der einzelnen Atome (oder zumindest der $C_\alpha$-Atome) sowie die Oberflächen der einzelnen Aminosäurereste zur Lösung sind bekannt. Die letzten Werte können auch durch Computerberechnungen und einige Vereinfachungen näherungsweise berechnet werden. Ziel ist es, eine Zuordnung von Aminosäuren auf die einzelnen Positionen zu bestimmen, so dass eine gegebene (im Folgenden genauer beschriebene) Energiefunktion minimiert wird.

**Gegeben:** Vollständige 3-dimensionale Struktur des Proteins. Durch Angabe des Ortes der $C_\alpha$-Atome und eventuell der Seitenketten (1. Kohlenstoffatom der Seitenkette). Für die Seitenketten ist jeweils die Oberfläche bekannt, die zur Lösung exponiert ist.

Sei $S \in \{H, P\}^n$, wobei $H(S) = \{i \mid S_i = H\}$, dann ist

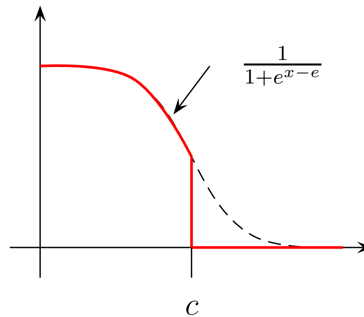
$$\Phi(S) = \alpha \cdot \sum_{\substack{i,j \in H(S) \\ i < j-2}} g(d_{ij}) + \beta \cdot \sum_{i \in H(S)} s_i$$

eine Energiefunktion in Abhängigkeit von $S$, die es zu minimieren gilt.

- $H$ steht für *hydrophob*, $P$ für *polar*.
- $s_i$ entspricht der Oberfläche der $i$-ten Seitenkette.
- $d_{ij}$ entspricht dem Abstand der $i$-ten Aminosäure zur $j$-ten Aminosäure.

- Die Funktion $g$ ist eine Funktion, die kleinere Abstände belohnt, z.B.

$$g(x) = \begin{cases} \frac{1}{1+e^{x-c}} & \text{für } x \leq c, \\ 0 & \text{sonst.} \end{cases}$$




  Ganz einfach kann $g$ auch wie folgt gewählt werden:

$$g(x) = \begin{cases} 1 & \text{für } x \leq c, \\ 0 & \text{sonst.} \end{cases}$$

- $\alpha$ und $\beta$ sind Skalierungsfaktoren mit $\alpha < 0$ und $\beta > 0$, z.B. $\alpha = -2$ und $\beta = 1/3$.

**Gesucht:** $S \in \{H, P\}^n$, die $\Phi(S)$ minimiert.

Hierbei wird durch die Energiefunktion $\Phi$ der Ausbildung hydrophober Kerne Rechnung getragen. Nahe beieinander liegende hydrophobe Aminosäurereste werden belohnt, während hydrophobe Aminosäurereste, die zur Lösungsflüssigkeit exponiert sind, bestraft werden.

Weiterhin wird im Grand Canonical Modell implizit unterstellt, dass eine Folge von Aminosäuren, die die Energiefunktion für die vorgegebene dreidimensionale Struktur minimiert, ebenfalls wieder die vorgegebene dreidimensionale Struktur als native Faltung einnimt. Dies ist natürlich a priori überhaupt nicht klar. Es kann durchaus sein, dass eine solche, die Energiefunktion minimierende Aminosäuresequenz eine ganz andere native Faltung, d.h. dreidimensionale Struktur, einnimmt.

Wir wollen im Folgenden das gegebene Problem mit Hilfe einer Reduktion auf ein Schnitt-Problem in Graphen in polynomieller Zeit sehr effizient lösen.

**26. Juni**

### 3.1.2 Schnitte in Netzwerken

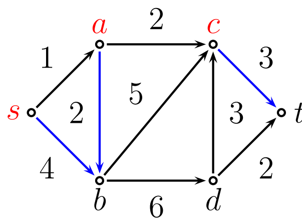
Um die geforderte Reduktion beschreiben zu können, müssen wir erst noch die Begriffe eines Netzwerks und Schnitten darin formalisieren.

**Definition 3.1** *Ein Netzwerk ist ein gerichteter Graph* $G = (V, E, \gamma)$*, wobei* $\gamma : E \to \mathbb{R}_+^*$ *ist.*

Im Allgemeinen betrachtet man in solchen Netzwerken noch zwei ausgezeichnete Knoten, namentlich $s, t \in V$. Um deutlich zu machen, welches die beiden ausgezeichneten Knoten sind, spricht man auch von *s-t-Netzwerken*.

Des Weiteren wird für alle nicht vorhandenen gerichteten Kanten ihr Gewicht auf 0 gesetzt, d.h $\gamma(u, v) = 0$ für $(u, v) \notin E$. Dann ist die Kantengewichtsfunktion natürlich als Funktion $\gamma : V \times V \to \mathbb{R}_+$ zu verstehen.

**Beispiel:**




Schnitt $(V_1, V_2)$ von $G$.

$V_1 = \{s, a, c\}$

$V_2 = \{b, d, t\}$

$c(V_1, V_2) = 3 + 2 + 4 + 9$

**Definition 3.2** *Sei* $G = (V, E, \gamma)$ *ein Netzwerk und seien* $s, t \in V$*. Ein s-t-*Schnitt *ist eine Partition* $(V_1, V_2)$ *von* $V = V_1 \cup V_2$ *mit* $s \in V_1$*,* $t \in V_2$*.*
*Die* Kapazität *eines s-t-Schnittes* $(V_1, V_2)$ *ist gegeben durch:*

$$c(V_1, V_2) = \sum_{\substack{x \in V_1, y \in V_2 \\ (x,y) \in E}} \gamma(x, y).$$

**Achtung:** Man beachte, dass bei der Kapazität eines Schnittes $(X, Y)$ nur Kanten von $X$ nach $Y$, aber nicht von $Y$ nach $X$ berücksichtigt werden. Somit gilt im Allgemeinen:

$$\begin{aligned} c(V_1, V_2) &\neq c(V_2, V_1), \\ c(V_1, V_2) &\neq -c(V_2, V_1). \end{aligned}$$

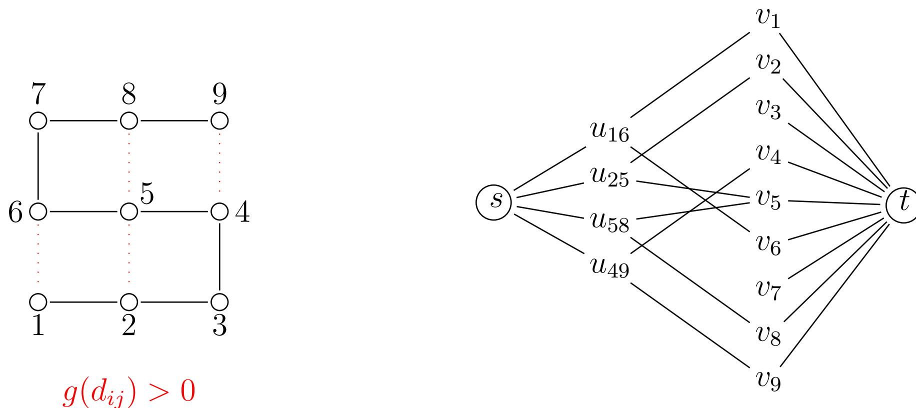
Für die Struktur und ihre zugehörige Energie-Funktion $\Phi$ definieren wir ein $s$-$t$-Netzwerk $G(\Phi) = (V, E, \gamma)$ mittels

$$V = \{s, t\} \cup \{v_i \mid i \in [1:n]\} \cup \underbrace{\{u_{ij} \mid i < j - 2 \wedge g(d_{ij}) > 0\}}_{=:U},$$

$$E = \{(s, u_{ij}) \mid u_{ij} \in U\} \cup \{v_i, t) \mid i \in [1:n]\} \cup \{(u_{ij}, v_i), (u_{ij}, v_j) \mid u_{ij} \in U\}.$$

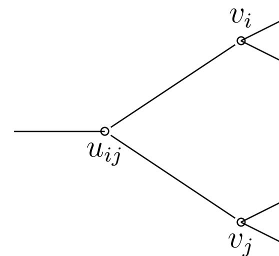
Die Angabe der Kantengewichtsfunktion folgt später.

**Beispiel:** Wir geben der Einfachheit halber nur für eine zweidimensionale Struktur das zugehörige Netzwerk an:




Kommen wir nun zur Definition der Kantengewichtsfunktion des Netzwerks $G(\Phi)$ für die gegebene Struktur und ihrer zugehörigen Energiefunktion $\Phi$:

$$\begin{aligned}
\gamma(s, u_{ij}) &:= |\alpha| \cdot g(d_{ij}) > 0, \\
\gamma(v_i, t) &:= \beta \cdot s_i > 0, \\
\gamma(u_{ij}, v_i) = \gamma(u_{ij}, v_j) &:= B + 1, \\
B &:= |\alpha| \sum_{i<j-2} g(d_{ij}) \;\geq\; 0.
\end{aligned}$$




### 3.1.3 Abgeschlossene Mengen und minimale Schnitte

Für die weiteren Untersuchungen werden abgeschlossene Mengen und minimale Schnitte in dem zur Energiefunktion zugehörigen Netzwerk eine wichtige Rolle spielen.

**Definition 3.3** *Sei $\Phi$ eine Energiefunktion und $G(\Phi)$ das zugehörige s-t-Netzwerk. Eine Menge $X \subseteq V(G(\Phi))$ heißt abgeschlossen, wenn*

*i)* $s \in X$ *und* $t \notin X$,

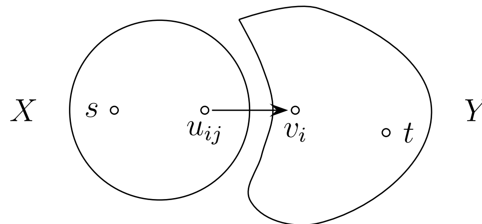
*ii)* $\forall u_{ij} \in V: \; u_{ij} \in X \Leftrightarrow v_i \in X_1 \wedge v_j \in X$.

**Lemma 3.4** *Wenn $(X, Y)$ ein minimaler (bzgl. der Kapazität) s-t-Schnitt in $G$ ist (mit $s \in X$), dann ist $X$ abgeschlossen.*

**Beweis:** $G$ besitzt offensichtlich einen Schnitt mit Kapazität $B$, nämlich den Schnitt $(\{s\}, V \setminus \{s\})$. Sei $(X, Y)$ ein minimaler $s$-$t$-Schnitt. Wir müssen zeigen, dass $X$ abgeschlossen ist.

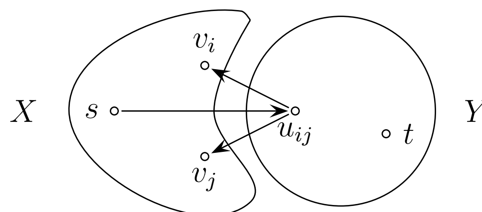
**Annahme:** $X$ ist nicht abgeschlossen.

<u>1. Fall:</u> Sei $u_{ij} \in X \wedge v_i \notin X$:




Somit gilt $c\ (X, Y) \geq \gamma(u_{ij}, v_i) = B + 1$. Dies ist jedoch ein Widerspruch zur Minimalität von $c(X, Y)$.

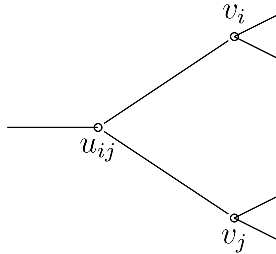
<u>2. Fall:</u> $v_i, v_j \in X \wedge u_{ij} \notin X$:




Betrachte $(X', Y')$ mit $X' = X \cup \{u_{ij}\}$ und $Y' = Y \setminus \{u_{ij}\}$.

Somit gilt $c(X', Y') < c(X, Y)$. Das ergibt einen Widerspruch zur Minimalität von $c(X, Y)$. ■

**Idee:**




Ist $u_{ij} \in X$, dann werden die Positionen $i$ und $j$ als hydrophob markiert.

**Notation:** Sei $S \in \{H, P\}^n$, dann bezeichne:

- $H(S) := \{i \mid s_i = H\} \;\hat{=}\; \{v_i \mid s_i = H\}$
- $X(S) := \{s, v_i, v_j, u_{ij} \mid v_i \in H(S) \wedge v_j \in H(S) \wedge g(d_{ij}) > 0\} \subseteq V(G(\Phi))$

Sei $X$ eine abgeschlossene Menge in $G(\Phi)$, dann bezeichne:

- $S(X) \in \{H, P\}^n$, wobei genau dann$(S(X))_i = H$, wenn $i \in X$.

**Lemma 3.5** *Sei $X$ eine abgeschlossene Menge in $G(\Phi)$, dann ist die Kapazität des s-t-Schnittes $(X, V \setminus X)$ mit $s \in X$ gleich $B + \Phi(S(X))$.*

**Beweis:** Da $X$ eine abgeschlossene Menge ist, gilt für alle Kanten $(x, y) \in X \times Y$ (wobei $Y := V \setminus X$), dass sie von folgender Form sind:

- Entweder $(x, y) = (v_i, t)$, wenn $v_i \in X$,
- oder $(x, y) = (s, v_j)$, wenn $v_i \notin X \vee v_j \notin X$.

Es gibt natürlich noch weitere Kanten zwischen $X$ und $Y$, nämlich $(u_{ij}, v_i) \in Y \times X$. Da diese aber von $Y$ nach $X$ verlaufen, sind diese für die Kapazität des Schnittes nicht von Interesse.

Es gilt also:

$$
\begin{aligned}
c(X, Y) &= \sum_{\substack{u_{ij} \in V \\ \{v_i, v_j\} \not\subseteq X}} \gamma(s, u_{ij}) + \sum_{v_i \in X} \gamma(v_i, t) \\
&= \sum_{\substack{u_{ij} \in V \\ \{v_i, v_j\} \not\subseteq X}} |\alpha| \cdot \gamma(d_{ij}) + \sum_{v_i \in X} \beta \cdot s_i
\end{aligned}
$$

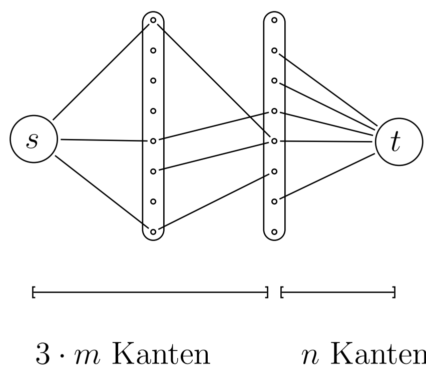
$$
\begin{aligned}
&= \sum_{u_{ij} \in V} |\alpha| \cdot g(d_{ij}) - \sum_{\substack{u_{ij} \in V \\ \{v_i, v_j\} \subseteq X}} |\alpha| \cdot g(d_{ij}) + \sum_{v_i \in X} \beta \cdot s_i \\
&= B - \sum_{\substack{u_{ij} \in V \\ \{v_i, v_j\} \subseteq X}} |\alpha| \cdot g(d_{ij}) + \sum_{v_i \in X} \beta \cdot s_i \\
&= B + \sum_{\substack{u_{ij} \in V \\ \{v_i, v_j\} \subseteq X}} \alpha \cdot g(d_{ij}) + \sum_{v_i \in X} \beta \cdot s_i \\
&= B + \Phi(S(X)).
\end{aligned}
$$

■

Somit können wir statt der Minimierung der Energiefunktion $\Phi$ auch die Minimierung eines Schnittes im zugehörigen Netzwerk betrachten. Sei dazu im Folgenden $m := \#\{\{i,j\} \mid g(d_{ij}) > 0\}$ die Anzahl der Paare von Aminosäuren der gegebenen Proteinstruktur, deren Abstand unter die vorgegebene Grenze fällt. Halten wir das Ergebnis dieses Abschnittes im folgenden Lemma fest.

**Lemma 3.6** *Das inverse Proteinfaltungsproblem im Grand Canonical Modell kann in Zeit $O(n^2)$ auf ein 'Min-Cut-Problem' in einem Netzwerk mit $O(n+m)$ Knoten und Kanten reduziert werden.*

**Beweis:** Wir müssen nur noch die Behauptung über die Anzahl der Kanten im konstruierten Netzwerk beweisen. Siehe dazu die folgende Skizze des Netzwerks:




Offensichtlich hat jeder Knoten aus $U$ einen Grad von 3, und $t$ ist zu genau $n$ Kanten inzident. Damit sind alle Kanten abgedeckt. Da nach Voraussetzung $|U| = m$ ist, haben wir also genau $3m + n$ Kanten. ■

In der „Praxis“ gilt: $m = O(n)$

*Bemerkung:* $s$ bedeutet in der Regel die *Quelle* des Flusses (*engl. source*) und $t$ die *Senke* des Flusses (*engl. target oder sink*).

**1. Juli**

## 3.2 Maximale Flüsse und minimale Schnitte

In diesem Abschnitt wollen wir uns mit Algorithmen zur Bestimmung eines maximalen Flusses in einem gegebenen Netzwerk beschäftigen.

### 3.2.1 Flüsse in Netzwerken

Formalisieren wir zunächst, was wir unter einem Fluss in einem Netzwerk verstehen wollen.

**Definition 3.7 (Flüsse in Netzwerken)** *Sei $G = (V, E, \gamma)$ ein $s$-$t$-Netzwerk. Ein* Fluss *in einem $s$-$t$-Netzwerk $G$ ist eine Funktion $\varphi : V \times V \to \mathbb{R}$, wobei gilt:*

*i)* $\forall u, v \in V : \ \varphi(u, v) = -\varphi(v, u)$ *(*Schiefsymmetrie*);*

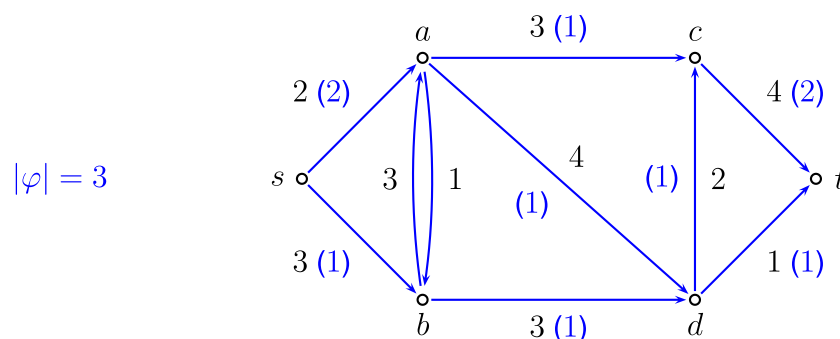
*ii)* $\forall u, v \in V : -\gamma(v, u) \leq \varphi(u, v) \leq \gamma(u, v)$ *(*Kapazitätsbedingung*),*

*Erinnerung:* $\gamma(u, v) = 0 \Leftrightarrow (u, v) \notin E$*;*

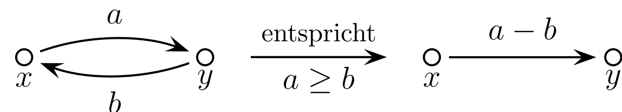
*iii)* $\forall u \in V \setminus \{s, t\} : \sum_{v \in V} \varphi(u, v) = 0$ *(*Kirchhoffsche Regel*).*

*Mit* $|\varphi| = \sum_{v \in V} \varphi(s, v)$ *bezeichnen wir den Betrag des Flusses* $\varphi$*.*

Beispiel:



Die *anti-parallelen* Flüsse (z.B. '$a \to b$' und '$b \to a$') werden in $\varphi$ nicht betrachtet, da wir solche anti-parallelen Flüsse immer wie folgt vermeiden können:




**Maximaler Fluss**

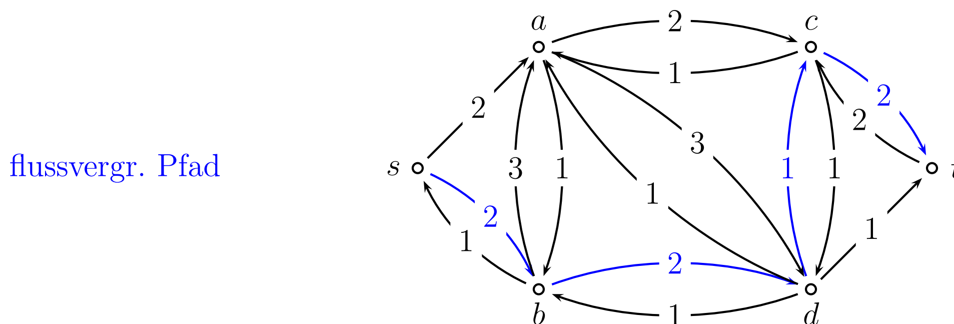
**Eingabe:** Ein $s$-$t$ Netzwerk $G = (V, E, \gamma)$.
**Gesucht:** Ein maximaler Fluss $\varphi$ von $s$ nach $t$ in $G$.

### 3.2.2 Residuen-Netzwerke und augmentierende Pfade

Zur Verbesserung (d.h. Vergrößerung) von Flüssen in Netzwerken benötigen wir die Begriffe von Residuen-Netzwerken und augmentierenden Pfaden.

**Definition 3.8** *Sei $G = (V, E, \gamma)$ ein $s$-$t$-Netzwerk und sei $\varphi$ ein Fluss von $s$ nach $t$ in $G$. Das zugehörige* Residuen-Netzwerk *ist ein $s$-$t$-Netzwerk $G_\varphi(V, E', \rho)$ mit $\rho(u, v) := \gamma(u, v) - \varphi(u, v)$ und $E' = \{(u, v) \mid \rho(u, v) > 0\}$.*

Beispiel: Das folgende Residuennetzwerk resultiert aus dem Beispiel für den Fluss im $s$-$t$-Netzwerk auf der vorherigen Seite.




**Definition 3.9** *Sei $G$ ein $s$-$t$-Netzwerk und $\varphi$ ein Fluss von $s$ nach $t$ in $G$. Ein Pfad von $s$ nach $t$ in $G_\varphi$ heißt* augmentierender Pfad *oder* flussvergrößernder Pfad.

### 3.2.3 Max-Flow-Min-Cut-Theorem

In diesem Abschnitt wollen wir den zentralen Zusammenhang zwischen minimalen Schnitten und maximalen Flüssen in Netzwerken herstellen.

**Theorem 3.10** *Sei $G = (V, E, \gamma)$ ein Netzwerk mit Quelle $s$ und Senke $t$. Dann sind für einen Fluss $\varphi$ von $s$ nach $t$ in $G$ die folgenden Aussagen äquivalent:*

*i) $\varphi$ ist ein maximaler Fluss;*

*ii) das Residuen-Netzwerk $G_\varphi$ enthält keinen augmentierenden Pfad;*

*iii) es existiert ein $s$-$t$-Schnitt $(S, T)$ von $G$ mit $|\varphi| = c(S, T)$.*

**Beweis: „i) ⇒ ii)“** durch Beweis der Kontraposition:

Nach Voraussetzung existiert ein augmentierender Pfad $p$ in $G_\varphi$ von $s$ nach $t$. Sei also $\mu := min\{\rho(u, v) \mid (u, v) \in p\} > 0$. Definiere

$$\varphi'(u,v) = \begin{cases} \varphi(u,v) + \mu & \text{für } (u,v) \in p \\ \varphi(u,v) - \mu & \text{für } (v,u) \in p \\ \varphi(u,v) & \text{sonst} \end{cases}$$

Es bleibt zu zeigen, dass $\varphi'$ ein Fluss in $G$ ist.

- Die Schiefsymmetrie folgt aus der Konstruktion.
- Die Kirchhoffsche Regel $\sum_v \varphi(u, v) = 0$ für $v \notin \{s, t\}$ folgt ebenfalls nach Konstruktion.
- Es bleibt zu zeigen, dass die Kapazitäten der Kanten des Netzwerkes eingehalten werden:

Wir werden dafür drei Fälle unterscheiden:

**1. Fall:** $(u, v) \notin p \wedge (v, u) \notin p$: Hier ist nichts zu zeigen, da $\varphi'(u, v) = \varphi(u, v)$.

**2. Fall:** $(u, v) \in p$:

$$\begin{aligned} \varphi'(u,v) &= \varphi(u,v) + \mu \\ &\leq \varphi(u,v) + \rho(u,v) \\ &= \varphi(u,v) + (\gamma(u,v) - \varphi(u,v)) \\ &= \gamma(u,v). \end{aligned}$$

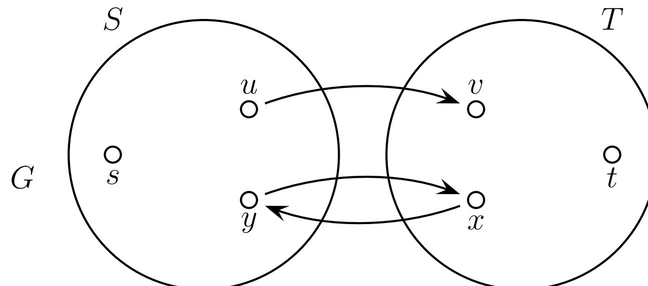
**3. Fall:** $(v, u) \in p$:

$$\begin{aligned}
\varphi'(u,v) &= \varphi(u,v) - \mu \\
&\geq \varphi(u,v) - \rho(v,u) \\
&= \varphi(u,v) - (\gamma(v,u) - \varphi(v,u)) \\
&= -\gamma(v,u) + \underbrace{(\varphi(u,v) + \varphi(v,u))}_{=0} \\
&= -\gamma(u,v).
\end{aligned}$$

Somit gilt $\varphi'(u,v) \in [-\gamma(v,u), \gamma(u,v)]$, also ist $\varphi'$ ein Fluss in $G$.

Weiter gilt nach Konstruktion $|\varphi'| = |\varphi| + \mu > |\varphi|$. Somit ist $\varphi$ kein maximaler Fluss in $G$. □

**„ii) $\Rightarrow$ iii)“** Sei $S = \{v \in V \mid \exists s \xrightarrow{*} v \text{ in } G_\varphi\}$, sei $T := V \setminus S$ und sei $s \in S$, $t \notin S$, $t \in T$. Dann ist $(S, T)$ ein $s$-$t$-Schnitt.




Ist $(u, v) \in E$ und $(u, v) \notin E'$, dann gilt $\varphi(u, v) = \gamma(u, v)$.

Ist $(x, y) \in E$ und $(y, x) \notin E'$, dann gilt $\varphi(x, y) = 0$.

Ist $(x, y) \in E$ und $(y, x) \in E$ und $(y, x) \notin E'$, dann gilt $\varphi(y, x) = \gamma(y, x)$ und somit $\varphi(x, y) = -\gamma(y, x) < 0$.

Mit Hilfe einer Übungsaufgabe von Blatt 9 folgt:

$$\begin{aligned}
|\varphi| &= \sum_{(s,y)\in E} \varphi(s,y) - \sum_{(y,s)\in E, (s,y)\notin E} \varphi(y,s) \\
&= \sum_{\substack{(x,y)\in E \\ x\in S, y\in T}} \varphi(x,y) - \sum_{\substack{(y,x)\in E \\ x\in S, y\in T, (x,y)\notin E}} \varphi(y,x)
\end{aligned}$$

$$= \sum_{\substack{(x,y)\in E\\ x\in S, y\in T}} \underbrace{\varphi(x,y)}_{=\gamma(x,y)} - \sum_{\substack{(y,x)\in E\\ x\in S, y\in T, (x,y)\notin E}} \underbrace{\varphi(y,x)}_{=0}$$
$$= c(S,T).$$

□

**„iii) ⇒ i)"** Es gilt: $|\varphi| \leq c(S,T)$ für alle $s$-$t$-Schnitte $(S,T)$ von $G$. Nach Voraussetzung existiert ein $s$-$t$-Schnitt $(S,T)$ mit $|\varphi| = c(S,T)$. Somit ist $\varphi$ ein maximaler Fluss. ■

**Korollar 3.11 (Max-Flow-Min-Cut-Theorem)** *Sei $G$ ein $s$-$t$-Netzwerk, dann ist die Kapazität eines minimalen $s$-$t$-Schnittes in $G$ gleich dem Betrag eines maximalen Flusses von $s$ nach $t$ in $G$.*

**3. Juli**

### 3.2.4 Algorithmus von Ford und Fulkerson

Aus der gewonnenen Erkenntnis lässt sich der folgende Algorithmus von Ford und Fulkerson zur Konstruktion eines maximalen Flusses in einem Netzwerk angeben.

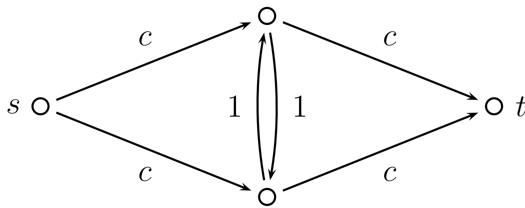
```
MaxFlow (G = (V, E, γ))
{
    Starte mit φ ≡ 0
    loop
    {
        Konstruiere Residuennetzwerkes G_φ
        Suche augmentierenden Pfad in G_φ (z.B. mit Tiefensuche)
        if (kein augmentierenden Pfad) break
        Sonst vergrößere Fluss mit Hilfe des augmentierenden Pfades
    }
    Nun ist φ der maximale Fluss in G
}
```

Abbildung 3.1: Algorithmus von Ford-Fulkerson

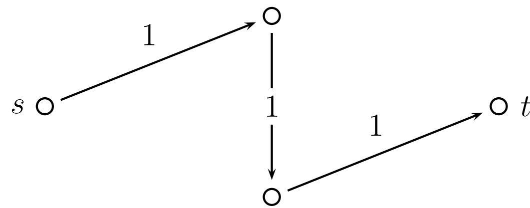
Laufzeit: Für die Laufzeitanalyse nehmen wir an, dass $\gamma : E \to \mathbb{N}$. Sei $c$ die größte Kapazität, d.h. $c = \max\{\gamma(e) \mid e \in E\}$. Dann gilt $|\varphi| \leq |E| \cdot c$. Da wir eine ganzzahlige Kantengewichtsfunktion angenommen haben, wird in jedem Schleifendurchlauf der Fluss um mindestens 1 erhöht. Somit kann es maximal $O(c \cdot |E|)$ Schleifendurchläufe geben. Jeder Schleifendurchlauf kann mit einer Tiefensuche in Zeit $O(n+m)$ bewerkstelligt werden. Somit ist die Laufzeit $O(|E|^2 \cdot c)$.

Wir wollen noch anmerken, dass bei Verwendung von irrationalen Kantengewichten der Algorithmus von Ford und Fulkerson nicht notwendigerweise terminieren muss. In der Praxis sind irrationale Kantengewichte sowieso kaum von Bedeutung, da deren Darstellung in einem Computer schon einen gehörigen Aufwand erfordert.

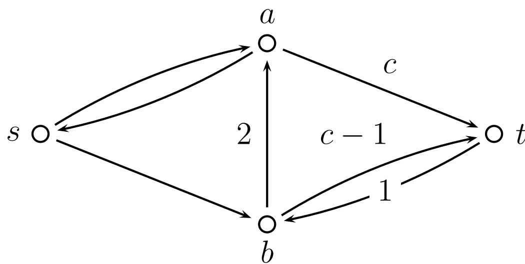
Beispiel:
$G_\varphi$:



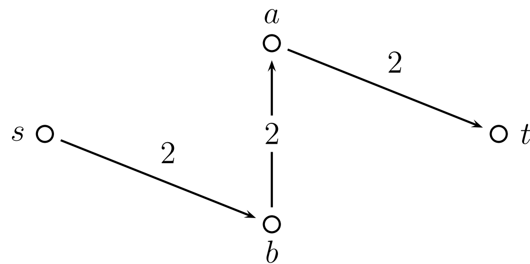

Augmentierender Pfad:



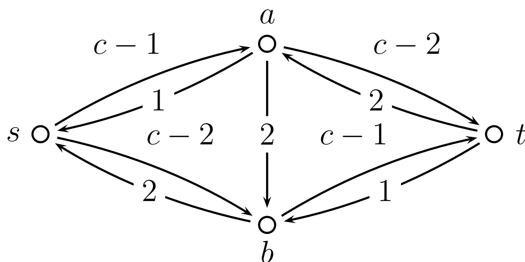

ergibt $G_\varphi$:




nächster augmentierender Pfad:




ergibt $G_\varphi$:




springt immer zwischen den 2 Möglichkeiten hin und her $\Rightarrow$ in diesem Fall können $\Theta(c)$ Iterationen möglich sein.

Somit hängt die Laufzeit von der größten Kapazität einer Kante ab und kann somit exponentiell in der Eingabegröße sein. Solche Algorithmen nennt man auch *pseudopolynomiell*, da sie bei kleinen Kapazitäten polynomielle Laufzeiten besitzen, jedoch bei großen Kapazitäten (im Verhältnis zur Eingabegröße) exponentielle Laufzeiten bekommt.

**Lemma 3.12** *Mit Hilfe des Algorithmus von Ford-Fulkerson kann der maximale Fluss in einen Netzwerk* $G = (V, E, \gamma)$ *mit ganzzahliger Kantengewichtsfunktion und mit* $\gamma(e) \leq C$ *für alle* $e \in E$ *in Zeit* $O(C \cdot |E|^2)$ *berechnet werden.*

### 3.2.5 Algorithmus von Edmonds und Karp

Einw Verbesserung des Algorithmus von Ford-Fulkerson lässt sich durch Verwendung kürzester augmentierender Pfade erzielen. Dies lässt sich mit einer Breitensuche statt einer Tiefensuche leicht implementieren. Die zuerst gefundenen augmentierenden Pfade sind dann die kürzesten. Dies führt zu einer Verbesserung der Laufzeit des Algorithmus, wie wir gleich sehen werden.

**Notation:** $\ell_\varphi(v)$ bezeichne die Länge eines kürzesten Pfades von $s$ nach $v$ in $G_\varphi$.

**Lemma 3.13** *Werden nur kürzeste augmentierende Pfade gewählt, so sind die Längen der kürzesten augmentierenden Pfade monoton steigend.*

**Beweis:** (durch Widerspruch)

Sei $v \in V \setminus \{s\}$ ein Knoten, dessen Abstand von $s$ nach einer Flussvergrößerung kürzer geworden ist. Somit gilt $\ell_{\varphi'}(v) < \ell_\varphi(v)$, wobei $\varphi'$ aus $\varphi$ durch Flussvergrößerung entsteht. Unter allen solchen Knoten, wählen wir einen Knoten $v$ mit minimalen $\ell_{\varphi'}(v)$.

Sei $p$ ein kürzester Pfad in $G_{\varphi'}$ von $s$ nach $v$ und sei $u$ der direkte Vorgänger von $v$ auf diesem Pfad.

Es gilt $\ell_{\varphi'}(u) = \ell_{\varphi'}(v) - 1$ und $(u, v) \in E(G_{\varphi'})$. Nach Wahl von $v$ ist $\ell_{\varphi'}(u) \geq \ell_\varphi(u)$.

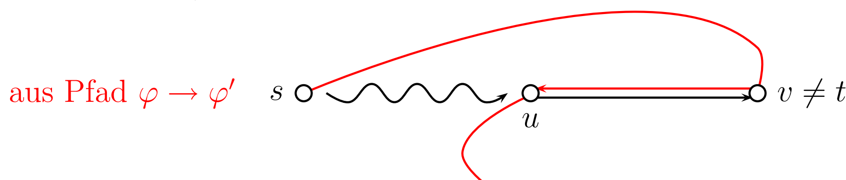
**Behauptung:** $(u, v) \notin E(G_\varphi)$

<u>Annahme:</u> $(u, v) \in E(G_\varphi)$. Dann gilt:

$$\begin{aligned}
\ell_\varphi(v) &\leq \ell_\varphi(u) + 1 \\
&\leq \ell_{\varphi'}(u) + 1 \\
&= \ell_{\varphi'}(v) - 1 + 1 \\
&= \ell_{\varphi'}(v).
\end{aligned}$$

Dies führt zu einem Widerspruch zur Wahl von $v$. □

Wie kann $(u, v)$ in $G_{\varphi'}$ hinzu gekommen sein?




Ein kürzester Pfad von $s \to u$ geht über $v$ und wurde im augmentierenden Pfad gewählt.

$$\begin{aligned}\ell_\varphi(v) &= \ell_\varphi(u) - 1 \\ &\leq \ell_{\varphi'}(u) - 1 \\ &= \ell_{\varphi'}(v) - 1 - 1 \\ &= \ell_{\varphi'}(v) - 2.\end{aligned}$$

Dies führt zu einem Widerspruch zu $\ell_\varphi(v) \geq \ell_{\varphi'}(v)$. ■

**Notation:** Eine Kante $(u, v)$ eines augmentierenden Pfades $p$ in $G_\varphi$ heißt *kritisch*, wenn $\rho(u, v) = \min\{\rho(e) \mid e \in p\}$. Eine kritische Kante wird oft auch als *Flaschenhals* bezeichnet.

**Lemma 3.14** *Jede Kante in $G$ kann maximal $\frac{|V|}{2}$ Mal kritisch werden.*

**Beweis:** Sei $(u, v) \in E(G)$. Wenn $(u, v)$ das erste Mal kritisch wird, gilt:

$$\ell_\varphi(v) = \ell_\varphi(u) + 1.$$

Somit verschwindet die Kante $(u, v)$ aus dem Residuen-Netzwerk $G_{\varphi'}$. Die Kante $(u, v)$ kann erst wieder auftauchen, wenn Fluss von $v$ nach $u$ verschickt wird (Kante an Kapazitätsgrenze).

Sei $G_{\varphi''}$ das Residuen-Netzwerk, in dem $(v, u)$ in einem kürzesten augmentierenden Pfad auftritt.

Es gilt dann: $\ell_{\varphi''}(u) = \ell_{\varphi''}(v) + 1$.

$$\begin{aligned}\ell_{\varphi''}(u) &= \ell_{\varphi''}(v) + 1 \\ &\geq \ell_\varphi(v) + 1 \\ &= \ell_\varphi(u) + 2.\end{aligned}$$

Somit wächst nach jedem kritischen Zustand von $(u, v)$ der Abstand für $u$ von $s$ um 2. Dies kann aber maximal $\frac{|V|}{2}$ Mal passieren, da jeder einfache Pfad nur die maximale Länge $|V|$ haben kann. ■

**Theorem 3.15** *Der Algorithmus von Edmonds-Karp, der nur kürzeste augmentierende Pfade verwendet, benötigt zur Bestimmung eines maximalen Flusses in einem Netzwerk $G = (V, E, \gamma)$ maximal $O(|V| \cdot |E|^2)$ Zeit.*

**Beweis:** Jede Kante wird maximal $O(|V|)$ kritisch. Es gibt maximal $O(|E|)$ Kanten, also kann es maximal $O(|V| \cdot |E|)$ viele Flussvergrößerungen geben. Jede Flussvergrößerung benötigt mit Hilfe einer Breitensuche Zeit $O(|V| + |E|)$. ■

### 3.2.6 Der Algorithmus von Dinic

Die Breitensuche findet nicht nur einen kürzesten, sondern alle kürzesten augmentierenden Pfade ohne wesentlichen zusätzlichen Mehraufwand mehr oder weniger gleichzeitig. Man könnte also auch alle kürzesten augmentierende Pfade gleichzeitig für eine Flussvergrößerung verwenden. Der daraus resultierende Algorithmus (Algorithmus von Dinic) hat eine Laufzeit von $O(|V|^2 \cdot |E|)$.

**Theorem 3.16** *Der Algorithmus von Dinic, der alle kürzesten augmentierenden Pfade gleichzeitig verwendet, benötigt zur Bestimmung eines maximalen Flusses in einem Netzwerk $G = (V, E, \gamma)$ maximal $O(|V|^2 \cdot |E|)$ Zeit.*

Auf den Beweis dieses Satzes wollen wir an dieser Stelle nicht weiter eingehen und verweisen auf die entsprechenden Lehrbücher.

**8. Juli**

## 3.3 Erweiterte Modelle der IPF

Zum Schluss wollen wir uns mit ein paar Erweiterungen und den damit zusammenhängenden Fragestellungen zum Grand Canonical Model der inversen Proteinfaltung widmen.

### 3.3.1 Erweiterung auf allgemeine Hydrophobizitäten

In diesem Abschnitt wollen wir uns die Frage stellen, ob wir auch für kontinuierliche Hydrophobizitäten eine optimale Zuordnung im Grand Canonical Modell in polynomieller Zeit berechnen können.

Bislang haben wir nur zwischen hydrophoben und polaren Aminosäuren unterschieden, die Einteilung war also diskret. Nun wollen wir beliebige reelle Werte zwischen 0 und 1 als Hydrophobizität zuslassen. 1 steht dabei für hydrophobe und 0 für polare Aminosäuren. Werte dazwischen können eine genauere Abstufung zwischen mehr oder weniger hydrophoben Aminosäuren darstellen, da auch die in der Natur vorkommenden Aminosäuren eine unterschiedliche Hydrophobizität besitzen.

Wir lassen jetzt für jede Aminosäureposition des Proteins einen Wert $z_i \in [0, 1]$ zu: Für die Energiefunktion erhalten wir dann:

$$\Phi(S) = \alpha \sum_{i<j-2} z_i \cdot z_j \cdot g(d_{ij}) + \beta \sum_{i=1}^{n} s_i \cdot z_i.$$

Wir suchen also jetzt eine Folge $(z_1, \ldots, z_n) \in [0,1]^n$, so dass $\Phi(S)$ minimal wird.

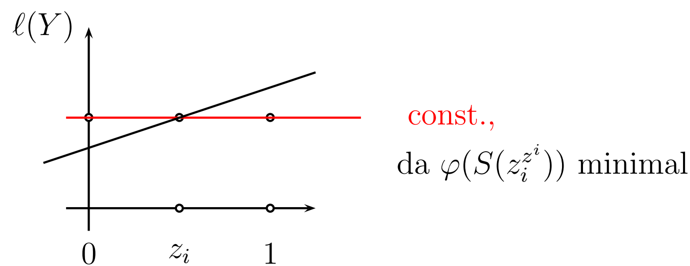
**Lemma 3.17** *Für jede Struktur $S$ und ihre zugehörige Energiefunktion $\Phi$ gibt es eine optimale Folge $z = (z_1, \ldots, z_n)$ mit $z_i \in \{0,1\}$.*

**Beweis:** Sei $z$ eine optimale Folge, d.h. $\Phi(S)$ ist minimal. Sei $z$ eine solche optimale Folge, die $\#\{i \mid z_i \in (0,1)\}$ minimiert.

**Behauptung:** $\forall i \in [1:n]: \ z_i \in \{0,1\}$.

Sei $z_i \in (0,1)$. Definiere $z_i^y := (z_1, \ldots, z_{i-1}, y, z_{i+1}, \ldots, z_n)$. Wir betrachten jetzt die Funktion $\ell : [0,1] \to \mathbb{R}$ vermöge $y \mapsto \varphi(S(z_i^y))$.

**Fakt:** $\ell$ ist eine affine Funktion:




Da $\ell$ eine affine Funktion ist, muss $\ell$ sogar konstant sein, da sonst das eindeutige Minimum am Rand des Intervalls $[0,1]$ angenommen wird.

Dann gilt mit $z' = (z_1, \ldots, z_{i-1}, 0, z_{i+1}, \ldots, z_n)$, dass $\Phi(S(z')) = \Phi(S(z_i^{z_i}) = \Phi(z))$. Dies ergibt einen Widerspruch zur Wahl von $z$. ■

Somit machen also im Grand Canonical Model solche Erweiterungen von Hydrophobizitäten keinen Sinn. Wir erhalten dieselben Lösungen, wie im diskreten Modell.

### 3.3.2 Energie-Landschaften im Grand Canonical Modell

Wie sieht eine Energieminimums-Landschaft im Grand Canonical Model aus? Sei $\Omega = \{S \in \{H,P\}^n \mid \Phi(S) = \min \Phi\}$, d.h. $\Omega$ ist die Menge der optimalen Lösungen von $\Phi$. Wir wollen jetzt untersuchen, ob diese Minima beispielsweise durch Punktmutationen ineinander überführbar sind. Dies kann Hinweise auf die Stabilität solcher Proteine gegenüber Punktmutationen implizieren.

Die folgende Darstellung lässt sich auch auf allgemeinere Mutationen anstelle von Punktmutationen verallgemeinern. Da die Methoden jedoch im Wesentlichen identisch sind, verweisen wir auf die Originalliteratur.

**Frage:** Ist $\Omega$ zusammenhängend (bzgl. (Punkt-)Mutation)?

Wir betrachten zur Beantwortung dieser Frage die folgende Funktion $f$:

$$f : 2^{[1:n]} \to \mathbb{R} : f(x) = \varphi(S(X)),$$

wobei $S(X)$ wiederum durch die Beziehung $(S(X))_i = H \Leftrightarrow i \in X$ definiert ist.

Erinnerung: $2^{[1:n]} = \{X \subseteq [1:n]\}$, also die Potenzmenge von $[1:n]$.

**Definition 3.18** *Sei $M$ eine Menge. Eine Funktion $\varphi : M \to \mathbb{R}$ heißt submodular, wenn gilt:*

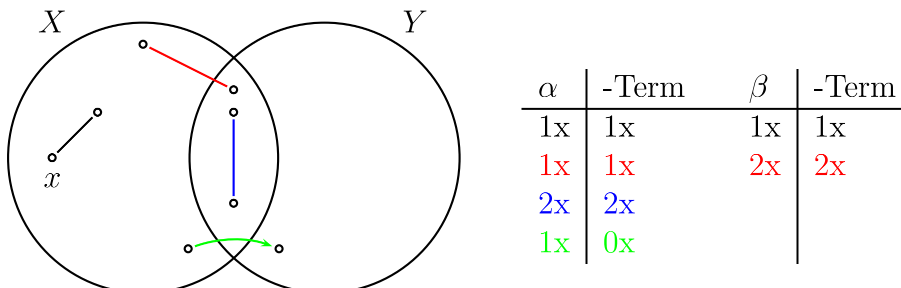
$$\forall X, Y \in M : \ \varphi(X \cap Y) + \varphi(X \cup Y) \leq \varphi(X) + \varphi(Y).$$

**Lemma 3.19** *Die betrachtete Energiefunktion $f$ ist submodular.*

**Beweis:** Die Energiefunktion sieht wie folgt aus:

$$f(X) = \varphi(S(X)) = \alpha \cdot \sum_{\substack{i<j-2 \\ i,j \in X}} g(d_{ij}) + \beta \sum_{i \in X} s_i$$

Um die Submodularität zu untersuchen, betrachten wir zunächst alle auftretenden Kanten, die im ersten Term aufaddiert werden. In der folgende Abbildung 3.2 sind die verschiedenen Fälle illustriert, die wir im Folgenden unterscheiden werden:




| $\alpha$ | -Term | $\beta$ | -Term |
|---|---|---|---|
| 1x | 1x | 1x | 1x |
| 1x | 1x | 2x | 2x |
| 2x | 2x | | |
| 1x | 0x | | |

Abbildung 3.2: Skizze zum Beweis der Submodularität

Sei $e$ die untersuchte Kante, die auch in der Energiefunktion berücksichtigt wird.

**$e \in (X \setminus Y) \times (X \setminus Y)$:** Somit wird $e$ in $f(X \cup Y)$ auf der linken und in $f(X)$ auf der rechten Seite aufaddiert.

**$e \in (X \setminus Y) \times (X \cap Y)$:** Somit wird $e$ in $f(X \cup Y)$ auf der linken und in $f(X)$ auf der rechten Seite aufaddiert.

**$e \in (X \cap Y) \times (X \cap Y)$:** Somit wird $e$ sowohl in $f(X \cup Y)$ als auch in $f(X \cap Y)$ auf der linken Seite aufaddiert. Auf der rechten Seite wird die Kante ebenfalls in $f(X)$ und $f(Y)$ berücksichtigt.

**$e \in (X \setminus Y) \times (Y \setminus X)$:** Somit wird $e$ in der linken Seite in $f(X \cup Y)$ berücksichtigt. Auf der rechten Seite wird diese Kante, aber nirgendwo aufsummiert. Da jedoch alle Summanden im ersten Teil der Gewichtsfunktion negativ sind, ist somit die linke Seite kleiner gleich der rechten Seite.

Die anderen nicht explizit aufgeführten Fälle verhalten sich analog zu einem der obigen Fälle.

Es bleibt noch die Summanden in der zweiten Summe zu berücksichtigen. Hier gehen nur einzelnen Aminosäuren ein. Sei also $v$ der untersuchte Knoten:

**$v \in X \setminus Y$:** Dieser Wert wird sowohl in $f(X \cup Y)$ auf der linken als auch in $f(X)$ auf der rechten Seite aufaddiert.

**$v \in X \cap Y$:** Dieser Wert wird sowohl in $f(X \cup Y)$ und $f(X \cap Y)$ auf der linken als auch in $f(X)$ und $f(Y)$ auf der rechten Seite je zweimal aufaddiert.

In jedem Fall ist die Summe gleich und der Beweis der Submodularität abgeschlossen. ■

**Notation:** $X, X' \subseteq [1 : n]$ heißen *adjazent*, wenn $|X \triangle X'| = 1$. $(X_1, \ldots, X_t)$ heißt eine *Kette*, wenn $X_i$ und $X_{i+1}$ adjazent sind für alle $i \in [1 : t-1]$.

Wir können nun die zu Beginn gestellte Frage wie folgt formulieren bzw. formalisieren:

**Frage:** Gilt für $X, Y \in \Omega$, dass eine $X$-$Y$-Kette existiert.

**Lemma 3.20** *Wenn $X, Y \in \Omega$, dann gilt $X \cup Y \in \Omega$ und $X \cap Y \in \Omega$.*

**Beweis:** Wir nehmen zunächst an, dass $f(X \cap Y) \leq f(X \cup Y)$ gilt.

Nach der submodularen Gleichung gilt

$$f(X \cap Y) + f(X \cup Y) \leq 2\mu,$$

da $\mu = f(X) = f(Y)$ der minimale Wert ist.

Mit $f(X \cap Y) \leq f(X \cup Y)$ folgt, dass $2 \cdot f(X \cap Y) \leq 2\mu$ und somit $f(X \cap Y) = \mu$ und daher muss nach Definition $X \cap Y \in \Omega$ sein.

Damit gilt aber auch

$$f(X \cup Y) \leq 2\mu - f(X \cap Y) = 2\mu - \mu = \mu.$$

Also ist $f(X \cap Y) = \mu$ und somit $X \cap Y \in \Omega$.

Der Fall, dass $f(X \cap Y) \geq f(X \cup Y)$ gilt, lässt sich analog beweisen. ∎

**Lemma 3.21** *Es existieren eindeutige Mengen $\underline{X}, \overline{X} \in \Omega$, so dass $\underline{X} \subseteq Y \subseteq \overline{X}$ für alle $Y \in \Omega$.*

**Beweis:** Wir definieren: $\underline{X} := \bigcap_{Y \in \Omega} Y$ und $\overline{X} := \bigcup_{Y \in \Omega} Y$. Nach dem vorherigen Lemma gilt, dass $\underline{X} \in \Omega$ und $\overline{X} \in \Omega$, da $\Omega$ endlich ist. Offensichtlich gilt $\underline{X} \subseteq Y \subseteq \overline{X}$ für $Y \in \Omega$. Wären diese nicht eindeutig und es gäbe auch $\underline{X'}$ bzw. $\overline{X'}$ mit denselben Eigenschaften, so wären $\underline{X} \cap \underline{X'}$ bzw. $\overline{X} \cup \overline{X'}$ andere Kandidaten und wir erhalten einen Widerspruch. ∎

**Notation:** Eine *Kette* $(X_1, \ldots, X_t)$ heißt *monoton*, wenn $X_1 \subseteq \cdots \subseteq X_t$ gilt.

**Lemma 3.22** *Sei $X, Y, Z \in \Omega$ mit $X \subseteq Y \subseteq Z$. Wenn es eine monotone $X$-$Z$-Kette in $\Omega$ gibt, dann gibt es auch eine monotone $X$-$Y$-Kette in $\Omega$.*

**Beweis:** Sei

$$X = X_1 \subseteq \ldots \subseteq X_t = Z$$

eine monotone $X$-$Z$-Kette. Betrachte die Kette

$$X_1 \cap Y \subseteq X_2 \cap Y \subseteq \cdots \subseteq X_t \cap Y.$$

Da $X_1 \cap Y = X$ und $X_t \cap Y = Y$, ist dies eine monotone $X$-$Y$-Kette. ∎

**Lemma 3.23** *Sei $Y \in \Omega$. Wenn es eine $\underline{X}$-$Y$-Kette in $\Omega$ gibt, dann gibt es auch eine monotone $\underline{X}$-$Y$-Kette.*

**Beweis:** Sei $C$ eine $\underline{X}$-$Y$-Kette in $\Omega$ kürzester Länge. Angenommen, diese sei nicht monoton und $(X1, \ldots, X_i)$ sei das maximale monotone Präfix davon:

$$C \; : \; \underline{X} = X_1 \subset \cdots \subset X_i \supset X_{i+1} \cdots X_t = Y.$$

Somit gibt es eine monotone $\underline{X}$-$X_i$-Kette in $\Omega$. Da $X_{i+1} \subset X_i$ ist, gibt es nach Lemma 3.22 eine monotone $\underline{X}$-$X_{i+1}$-Kette $C' = (X'_1, \ldots, X'_{t'})$ in $\Omega$. Aufgrund der Monotonie muss $C'$ kürzer als $(X_1, \ldots, X_{i+1})$ in $C$ sein. Dann ist aber

$$(X'_1, \ldots, X'_{t'}, X_{i+2}, \ldots, X_t)$$

eine kürzere $\underline{X}$-$Y$ Kette als $C$ und wir erhalten den gewünschten Widerspruch. ■

Aus den beiden letzten Lemmata erhalten wir sofort das folgende Korollar.

**Korollar 3.24** *$\Omega$ ist genau dann zusammenhängend, wenn es eine monotone $\underline{X}$-$\overline{X}$-Kette gibt.*

**Beweis:** **⇒:** Wenn $\Omega$ zusammenhängend ist, dann gibt es eine $\underline{X}$-$\overline{X}$-Kette in $\Omega$. Nach Lemma 3.23 gibt es dann auch eine monotone $\underline{X}$-$\overline{X}$-Kette in $\Omega$.

**⇐:** Seien $X, Y \in \Omega$. Da nach Lemma 3.21 $\underline{X} \subseteq X \subseteq \overline{X}$ und $\underline{X} \subseteq Y \subseteq \overline{X}$ gilt, und es eine monotone $\underline{X}$-$\overline{X}$-Kette in $\Omega$ gibt, folgt mit Lemma 3.22 auch eine monotone $\underline{X}$-$X$-Kette und eine monotone $\underline{X}$-$Y$-Kette. Diese beiden lassen sich leicht zu einer $X$-$Y$-Kette zusammensetzen. ■

Dieses Korollar gibt einen kurzen „Beweis“ für den Zusammenhang von $\Omega$ an.

**Notation:** $X \in \Omega$ heißt *Sackgasse*, wenn $X \neq \underline{X}$ für alle $X' \subseteq X$ mit $|X' \triangle X| = 1$ gilt, dass $X' \notin \Omega$.

**Lemma 3.25** *$\Omega$ ist genau dann zusammenhängend, wenn es keine Sackgassen in $\Omega$ gibt.*

**Beweis:** **⇒:** Wenn $\Omega$ zusammenhängend ist, dann gibt es für alle $X \in \Omega$ eine monotone $\underline{X}$-$X$-Kette. Somit kann es keine Sackgassen geben.

**⇐:** Sei $\Omega$ nicht zusammenhängend und sei $X \in \Omega$ so gewählt, dass es keine $\underline{X}$-$X$-Kette in $\Omega$ gibt. Unter allen möglichen solcher $X$, wählen wir eines mit kleinster Kardinalität. Dann ist aber $X$ eine Sackgasse. Angenommen es gäbe ein $X' \in \Omega$, das durch Entfernen eines Elements aus $X$ entsteht. Dann würde es aufgrund der

Minimalität von $X$ eine $\underline{X}$-$X'$-Kette in $\Omega$ und somit auch eine $\underline{X}$-$X$-Kette in $\Omega$ geben, was den gewünschten Widerspruch liefert. ■

Somit haben wir jetzt auch einen kurzen Beweis dafür, dass $\Omega$ nicht zusammenhängend ist.

Damit können wir jetzt den folgenden Algorithmus zur Bestimmung des Zusammenhangs von $\Omega$ angeben.

```
Zusammenhang in Ω
{
    Berechne X̲ = Min(f) und X̄ = Max(f)
    W := X̄
    while (W ≠ X̲)
    {
        Bestimme: i ∈ W : f(W \ {i}) = f(i)
        Gibt es kein solches i → return reject
        W := W \ {i}
    }
    return accept
}
```

Abbildung 3.3: Algorithmus: Zusammenhang in $\Omega$ bestimmen

Es ist bekannt, dass man für submodulare Funktionen deren zugehörige Mengen $\underline{X}$ und $\overline{X}$ in polynomieller Zeit bestimmen kann. Wir gehen hier nicht auf die Details ein, sondern verweisen auf Lehrbücher im Bereich der kombinatorischen Optimierung.

**Theorem 3.26** *Es kann in polynomieller Zeit entschieden werden, ob die Minimums-Landschaft einer gegebenen Energiefunktion $\Phi$ im Grand Canonical Modell zusammenhängend ist oder nicht.*

**10. Juli**

# Literaturhinweise A

## A.1 Lehrbücher zur Vorlesung


Peter Clote, Rolf Backofen: *Introduction to Computational Biology*; John Wiley and Sons, 2000.

Richard Durbin, Sean Eddy, Anders Krogh, Graeme Mitchison: *Biological Sequence Analysis*; Cambridge University Press, 1998.

Dan Gusfield: *Algorithms on Strings, Trees, and Sequences — Computer Science and Computational Biology*; Cambridge University Press, 1997.

David W. Mount: *Bioinformatics — Sequence and Genome Analysis*, Cold Spring Harbor Laboratory Press, 2001.

Pavel A. Pevzner: *Computational Molecular Biology - An Algorithmic Approach*; MIT Press, 2000.

João Carlos Setubal, João Meidanis: *Introduction to Computational Molecular Biology*; PWS Publishing Company, 1997.

Michael S. Waterman: *Introduction to Computational Biology: Maps, Sequences, and Genomes*; Chapman and Hall, 1995.


## A.2 Skripten anderer Universitäten


Bonnie Berger: *Introduction to Computational Molecular Biology*, Massachusetts Institute of Technology, http://theory.lcs.mit.edu/˜bab/class/01-18.417-home.html;

Bonnie Berger, Mona Sing *Introduction to Computational Molecular Biology*, Massachusetts Institute of Technology, http://theory.lcs.mit.edu/˜mona/18.417-home.html;

Paul Fischer: *Einführung in die Bioinformatik*, Universität Dortmund, Lehrstuhl II, WS2001/2002, http://ls2-www.cs.uni-dortmund.de/lehre/winter200102/bioinf/

Volker Heun: *Algorithmische Bioinformatik I/II*, Technischen Universität München und Ludwig-Maximilians-Universität München, WS 2001 & SS2002; http://www.bio.informatik.uni-muenchen.de/personen/heun/lecturenotes/

Richard Karp, Larry Ruzzo: *Algorithms in Molecular Biology*; CSE 590BI, University of Washington, Winter 1998. http://www.cs.washington.edu/education/courses/590bi/98wi/

Larry Ruzzo: *Computational Biology*, CSE 527, University of Washington, Fall 2001; http://www.cs.washington.edu/education/courses/527/01au/

Georg Schnittger: *Algorithmen der Bioinformatik*, Johann Wolfgang Goethe-Universität Frankfurt am Main, Theoretische Informatik, WS 2000/2001, http://www.thi.informatik.uni-frankfurt.de/BIO/skript2.ps.

Ron Shamir: *Algorithms in Molecular Biology* Tel Aviv University, http://www.math.tau.ac.il/~rshamir/algmb.html; http://www.math.tau.ac.il/~rshamir/algmb/01/algmb01.html.

Ron Shamir: *Analysis of Gene Expression Data, DNA Chips and Gene Networks*, Tel Aviv University, 2002; http://www.math.tau.ac.il/~rshamir/ge/02/ge02.html;

Martin Tompa: *Computational Biology*, CSE 527, University of Washington, Winter 2000. http://www.cs.washington.edu/education/courses/527/00wi/

## A.3 Originalarbeiten

Ting Chen, Ming-Yang Kao: On the Informational Asymmetry Between Upper and Lower Bounds for Ultrametric Evolutionary Trees, *Proceedings of the 7th Annual European Symposium on Algorithms, ESA'99*, Lecture Notes in Computer Science 1643, 248–256, Springer-Verlag, 1999.

Martin Farach, Sampath Kannan, Tandy Warnow: A Robust Model for Finding Optimal Evolutionary Trees, *Algorithmica*, Vol. 13, 155–179, 1995.

P. Fariselli, M. Finelli, D. Marchignoli, P.L. Martelli, I. Rossi, and R. Casadio: MaxSubSeq: An Algorithm for Segment-Length Optimization. T he Case Study of the Transmembrane Spanning Segments, Bioinformatics, Vol. 19, 500-505, 2003.

J.M. Kleinberg: Efficient Algorithms for Protein Sequence Design and the Analysis of Certain Evolutionary Fitness Landscapes, Proceedings of the 3rd ACM International Conference on Computational Molecular Biology, RECOMB'99, 1999.

Y.-L. Lin, T. Jiang, K.-M. Chao: Efficient Algorithms for Locating the Length-Constrained Heaviest Segments, with Applications to Biomolecular Sequence Analysis, Proceedings of the 27th International Symposium on Mathematical Foundations of Computer Science, MFCS'02, Lecture Notes in Computer Science, Vol. 2420, 459-470, 2002.

W.L. Ruzzo, M. Tompa: A Linear Time Algorithm for Finding All Maximal Scoring Subsequences, Proceedings of the 7th International Conference on Intelligent Systems for Molecular Biology, ISMB'99, 234-241, 1999.

# Index B


## Symbole

$p$-Norm, 78
$s$-$t$-Netzwerk, 111
$s$-$t$-Schnitt, 111
3-Punkte-Bedingung, 40
4-Punkte-Bedingung, 61

## A

additive Matrix, 52
additiver Baum, 51
  externer, 52
  kompakter, 52
Additives Approximationsproblem, 79
Additives Sandwich Problem, 78
adjazent, 127
amortisierte Kosten, 74
Approximationsproblem
  additives, 79
  ultrametrisches, 79, 108
aufspannend, 64
aufspannender Graph, 64
augmentierender Pfad, 117

## B

Baum
  additiver, 51
  additiver kompakter, 52
  evolutionärer, 35
  externer additiver, 52
  kartesischer, 100
  niedriger ultrametrischer, 81
  phylogenetischer, 35
  strenger ultrametrischer, 41
  ultrametrischer, 41
binärer Charakter, 37
Bunemans 4-Punkte-Bedingung, 61

## C

Charakter, 37
  binärer, 37
  numerischer, 37
  zeichenreihiges, 37
charakterbasiertes Verfahren, 37
cut-weight, 91

## D

distanzbasiertes Verfahren, 36
Distanzmatrix, 40
dynamische Programmierung, 105
Dynamische Programmierung, 4

## E

Euler-Tour, 103
evolutionärer Baum, 35
externer additiver Baum, 52

## F

fallend rechtsschief, 24
Fibonacci-Heap, 69
Fibonacci-Zahlen, 73
Flaschenhals, 123
Fluss, 116
flussvergrößernder Pfad, 117

## G

Gewicht eines Spannbaumes, 64
Graph
  aufspanneder, 64

## H

Heap, 69
Heap-Bedingung, 69

## I

induzierte Metrik, 44
induzierte Ultrametrik, 44

## K

Kapazität, 111
Kapazitätsbedingung, 116
kartesischer Baum, 100
kaskadenartiger Schnitt, 71
Kette, 127, 128
Kirchhoffsche Regel, 116
kompakte Darstellung, 42
kompakter additiver Baum, 52
Kosten, 86
kritisch, 123

## L

least common ancestor, 41
link-edge, 91
links-negativ, 21
links-negative Zeiger, 21

## M

Matrix
    additive, 52
maximal scoring, 8
Metrik, 39
    induzierte, 44
minimal links-negativ, 21
minimaler Spannbaum, 64
minimum spanning tree, 64
monoton, 128

## N

niedriger ultrametrischer Baum, 81
niedrigste gemeinsame Vorfahr, 41
Norm, 78
numerischer Charakter, 37

## P

phylogenetischer Baum, 35
Priority Queue, 68
pseudo-polynomiell, 121

## Q

Quelle, 116

## R

Rang, 69
Range Minimum Query, 101
rechtschiefe Zeiger, 27
rechtsschief, 24
Residuen-Netzwerk, 117

## S

Sackgasse, 129
Sandwich Problem
    additives, 78
    ultrametrisches, 78
Schiefsymmetrie, 116
Senke, 116
separabel, 90
Spannbaum, 64
    Gewicht, 64
    minimaler, 64
strenger ultrametrischer Baum, 41

## U

Ultrametrik, 39
    induzierte, 44
ultrametrische Dreiecksungleichung, 39
ultrametrischer Baum, 41
    niedriger, 81
Ultrametrisches Approximationsproblem, 79, 108
Ultrametrisches Sandwich Problem, 78
Union-Find-Datenstruktur, 95

## V

Verfahren
    charakterbasiertes, 37
    distanzbasiertes, 36

## W

Wurzelliste, 69

## Z

zeichenreihige Charakter, 37
zugehöriger gewichteter Graph, 64