TU Wien
Institut für Logic and Computation
Forschungsbereich Algorithms and Complexity

186.866 Algorithmen und Datenstrukturen VU

Programmieraufgabe P6

PDF erstellt am: 15. Mai 2025

1 Vorbereitung

Um diese Programmieraufgabe erfolgreich durchführen zu können, müssen folgende Schritte umgesetzt werden:

1. Laden Sie das zugehörige Java-Projekt `P6.zip` aus TUWEL herunter.
2. Entpacken Sie `P6.zip` und öffnen Sie das entstehende Projektverzeichnis in Ihrer Entwicklungsumgebung.
3. Öffnen Sie die nachfolgend angeführte Datei in Ihrem Editor. In dieser Datei sind sämtliche Programmiertätigkeiten durchzuführen. Ändern Sie keine anderen Dateien im Projekt und fügen Sie auch keine neuen hinzu.
   `task/src/main/java/`
   `StudentSolutionForExerciseImplementation.java`
4. Füllen Sie Vorname, Nachname und Matrikelnummer in der Methode `StudentInformation provideStudentInformation()` aus.
5. Gehen Sie sicher, dass Java-Version 21 auf Ihrem System verfügbar ist.

   Alternativ zu einer manuellen Installation, können Sie in der Datei `settings.gradle` das auskommentierte Code-Snippet für eine automatische Installation nutzen.
6. Details zur Ausführung des Codes finden Sie in Abschnitt 6.

2 Hinweise

Einige Hinweise, die Sie während der Umsetzung dieser Aufgabe beachten müssen:

- Lösen Sie die Aufgaben selbst. Verwenden Sie auch keine (KI)-Tools z.B. zur Codegenerierung.
- Sie dürfen beliebig viele Hilfsmethoden schreiben und benutzen. Beachten Sie aber, dass Sie nur die oben geöffnete Datei abgeben und diese Datei mit dem zur Verfügung gestellten Framework lauffähig sein muss.
- Es ist grundsätzlich erlaubt Bibliotheken wie z.B. das Java Collections Framework und die Java Math Klasse zu verwenden. Die Voraussetzung dabei ist, dass die Verwendung die eigentliche Programmieraufgabe nicht abnimmt. Beachten Sie, dass nur die oben genannte Java-Version auf unserem Testsystem zur Verfügung steht.

3 Übersicht

In dieser Programmieraufgabe geht es um eine Variante des Longest Common Subsequence Problems, bei dem für mehrere Eingabesequenzen in Form von Strings eine möglichst lange gemeinsame Teilsequenz gefunden werden soll. Wir führen das Problem in Abschnitt 4 ein. Dieses Optimierungsproblem wollen wir mithilfe von dynamischer Programmierung lösen.

4 Theorie

Die notwendige Theorie zu dynamischer Programmierung kann in den Vorlesungsfolien „Optimierung – Dynamische Programmierung“ gefunden werden. Im folgenden führen wir die notwendigen Begriffe und Definitionen für das Longest Common Subsequence Problem ein.

Eine **Sequenz** $s = a_1 a_2 \dots a_n$ ist eine Zeichenkette bestehend aus $n \in \mathbb{N}$ Zeichen. Jedes Zeichen a_i mit $1 \leq i \leq n$ entstammt dabei aus einem gegebenen **Alphabet** Σ, der Menge der möglichen Zeichen. Eine **Teilsequenz** einer Sequenz s ist eine Sequenz, die durch Löschung von

Zeichen aus s hervorgeht. Wir sagen auch, dass eine Teilsequenz von s in s enthalten ist.

Beispiel: Gegeben ist eine Sequenz s mit $n = 6$ Zeichen:

$s =$ A T C A C A

- A C und A A A sind zwei mögliche Teilsequenzen von s:
  A ~~T~~ C ~~A~~ ~~C~~ ~~A~~
  A ~~T~~ ~~C~~ A ~~C~~ A
- A A C T ist keine Teilsequenz von s, da es nicht möglich ist, diese Sequenz durch Löschung von Zeichen aus s zu erhalten.

Eine **gemeinsame Teilsequenz** zweier Sequenzen $s_1 = a_1 a_2 \dots a_{n_1}$ und $s_2 = b_1 b_2 \dots b_{n_2}$ ist eine Sequenz, die sowohl in s_1 als auch in s_2 enthalten ist. Eine **längste gemeinsame Teilsequenz** von s_1 und s_2 ist eine gemeinsame Teilsequenz mit maximaler Länge. Diese muss nicht eindeutig sein.

Beispiel: Gegeben sind die beiden Sequenzen s_1 und s_2 mit $n_1 = n_2 = 6$:

$s_1 =$ A T C A C A
$s_2 =$ A C T C A A

- A C C A ist eine gemeinsame Teilsequenz von s_1 und s_2:
  A ~~T~~ C ~~A~~ C A
  A C ~~T~~ C A ~~A~~
- A T C A A ist die (eindeutige) längste gemeinsame Teilsequenz von s_1 und s_2:
  A T C A ~~C~~ A
  A ~~C~~ T C A A

Jetzt können wir die beiden betrachteten Optimierungsprobleme definieren:

- **Longest Common Subsequence Problem (LCS)**: Gegeben zwei Sequenzen s_1 und s_2. Finde eine längste gemeinsame Teilsequenz von s_1 und s_2.
- **Constrained Longest Common Subsequence Problem (CLCS)**: Gegeben die Sequenzen s_1, s_2 und s_p. Finde eine möglichst lange gemeinsame Teilsequenz von s_1 und s_2, sodass s_p (s_p wird Pattern genannt) in dieser gemeinsamen Teilsequenz enthalten ist.

Beispiel: Wir erweitern das vorherige Beispiel um die Sequenz s_p mit $n_p = 2$ Zeichen:

$s_1 = \mathrm{A\ T\ C\ A\ C\ A}$
$s_2 = \mathrm{A\ C\ T\ C\ A\ A}$
$s_p = \mathrm{C\ C}$

- A T C A A ist die optimale Lösung für **LCS**, aber keine zulässige Lösung für **CLCS**, da C C nicht enthalten ist.
- A C C A eine zulässige Lösung für **LCS** und die optimale Lösung für **CLCS**, da C C enthalten ist und es keine längere gemeinsame Teilsequenz von s_1 und s_2 gibt, die C C enthält.

Wir lösen **CLCS** mithilfe von dynamischer Programmierung. Sei $s = a_1 a_2 \ldots a_n$ eine Sequenz, dann bezeichnet $s^i = a_1 \ldots a_i$ die Teilsequenz bestehend aus den ersten i Zeichen von s für $i = 0, 1, \ldots, n$, wobei $s^0 = \epsilon$ für die **leere Sequenz** bestehend aus 0 Zeichen steht. Die leere Sequenz ist in jeder anderen Sequenz enthalten.

Bezeichne t_{kij} die Länge der längsten Teilsequenz von s_1^i und s_2^j, die s_p^k enthält für $k = 0, 1, \ldots, n_p$, $i = 0, 1, \ldots, n_1$ und $j = 0, 1, \ldots, n_2$.

Beispiel:

$s_1 = \mathrm{A\ T\ C\ A\ C\ A}$
$s_2 = \mathrm{A\ C\ T\ C\ A\ A}$
$s_p = \mathrm{C\ C}$

- $t_{024} = 2$, da A T die längste Teilsequenz von $s_1^2 = \mathrm{A\ T}$ und $s_2^4 = \mathrm{A\ C\ T\ C}$ ist, die $s_p^0 = \epsilon$ enthält.
- $t_{143} = 2$, da A C die längste Teilsequenz von $s_1^4 = \mathrm{A\ T\ C\ A}$ und $s_1^3 = \mathrm{A\ C\ T}$ ist, die $s_p^1 = \mathrm{C}$ enthält.
- $t_{003} = 0$, da s_1^0 die leere Sequenz ist und es keine Sequenz mit mindestens einem Zeichen geben kann, die in $s_1 = \epsilon$ enthalten ist.

Unser Ziel ist es, die Tabelle t_{kij} rekursiv aufzubauen und aus der Tabelle die Optimallösung mittels Backtracking abzuleiten.

5 Implementierung

5.1 Überprüfung der Zulässigkeit

Implementieren Sie zunächst die Methode `boolean isFeasible(CLCSInstance instance, char[] solution)`. Die Methode soll `true` zurückgeben, falls `solution` eine zulässige Lösung für das **CLCS**-Problem für die Probleminstanz `instance` ist und `false` andernfalls. Sequenzen werden stets als `char`-Arrays verwaltet.

Die Klasse `CLCSInstance` verwaltet eine CLCS-Instanz und stellt folgende Methoden bereit, die Sie verwenden dürfen:

- `boolean isFeasibleLCS(char[] solution)`: Gibt `true` zurück, falls `solution` eine zulässige Lösung für die **LCS**-Variante der Instanz ist.
- `static int getNextOccurence(char[] sequence, char c, int currentPosition)`: Findet das nächste Vorkommen des Zeichens `c` in der Sequenz `sequence` beginnend ab der Stelle `currentPosition` und gibt den entsprechenden Index zurück. Kommt das gesuchte Zeichen `c` ab der Stelle `currentPosition` nicht vor, so wird die Länge von `sequence` zurückgegeben. Die Methode ist `static`, das heißt, dass die Methode nicht an ein Objekt der Klasse `CLCSInstance` gebunden ist. Der Zugriff auf die Methode erfolgt durch Voranstellen des Klassennamens.

  Die nachstehenden Beispiele beziehen sich auf die folgende Sequenz:
  `char[] sequence = new char[]{'A', 'B', 'C', 'B'};`

  - `CLCSInstance.getNextOccurence(sequence, 'C', 0)` gibt `2` zurück. Es ist die Stelle des nächsten `'C'` ab Position `0` (dem Sequenzbeginn).
  - `CLCSInstance.getNextOccurence(sequence, 'B', 1)` gibt `1` zurück.
  - `CLCSInstance.getNextOccurence(sequence, 'A', 1)` gibt `4` (= Länge von `sequence`) zurück, da `'A'` ab dem Index `1` nicht in `sequence` vorkommt.

- `public char[] getS1()`, `public char[] getS2()`, `public char[] getSp()`: Gibt die erste Sequenz s_1, die zweite Sequenz s_2 bzw. die Sequenz s_p zurück.

- `int getN1()`, `int getN2()`, `int getNp()`: Gibt die Anzahl der Zeichen der Sequenzen s_1, s_2 bzw. s_p zurück.
- `public String toString()`, `public void print()`: Gibt eine Stringrepräsentation der Instanz zurück bzw. gibt die Instanz aus.

5.2 Dynamische Programmierung: Tabelle erstellen

Entwickeln Sie eine rekursive Formel für die Bestimmung der Tabelle t_{kij} aus Abschnitt 4 und implementieren Sie anschließend die Methode `void computeDynamicProgrammingTable(CLCSInstance instance, DynamicProgrammingTable table)`. Die Laufzeit soll in $O(n_1 n_2 n_p)$ liegen.

Die Klasse `DynamicProgrammingTable` verwaltet die $(n_p + 1) \times (n_1 + 1) \times (n_2 + 1)$ große Tabelle. Folgende Methoden stellt sie für Sie bereit:

- `int get(int k, int i1, int i2)`: Gibt den (momentanen) Wert von $t_{ki_1i_2}$ zurück.
- `void set(int k, int i1, int i2, int value)`: Setzt den Wert von $t_{ki_1i_2}$ auf den Wert `value`.
- `public String toString()`, `public void print()`: Gibt eine Stringrepräsentation der Tabelle zurück bzw. druckt die Tabelle aus.

Der zweite Parameter `table` ist bereits mit den richtigen Größen initialisiert, Sie brauchen nur noch die Tabelleneinträge korrekt berechnen und hinterlegen. Achten Sie bei der Implementierung auf die korrekte Indizierung.

Hinweise: Überlegen Sie sich zunächst, wie Sie die Tabelle für das **LCS**-Problem aufbauen können (also für den Fall $k = 0$) und erweitern Sie dann Ihre Idee für $k > 0$. Stellen Sie sich dabei insbesondere die Frage, wie Sie $t_{ki_1i_2}$ für $i_1 = 0$ oder/und $i_2 = 0$ geeignet initialisieren können, damit Ihre rekursive Formel funktioniert.

5.3 Dynamische Programmierung: Backtracking CLCS

Implementieren Sie die Methode `char[] backtrackingCLCS(CLCSInstance instance,` `DynamicProgrammingTable table)`. Ziel dieser Methode ist es, aus der in Abschnitt 5.2 bestimmten Tabelle eine optimale Lösung für das **CLCS**-Problem für die Instanz `instance` mittels Backtracking abzuleiten und als `char`-Array zurückzugeben.

5.4 Dynamische Programmierung: Backtracking LCS

Implementieren Sie die Methode `char[] backtrackingLCS(CLCSInstance instance, DynamicProgrammingTable table)`. Ziel dieser Methode ist es, aus der in Abschnitt 5.2 bestimmten Tabelle eine optimale Lösung für das **LCS**-Problem (also ohne Berücksichtigung von s_p) für die Instanz `instance` mittels Backtracking abzuleiten und als `char`-Array zurückzugeben.

6 Testen

Führen Sie im Projektverzeichnis den Befehl `./gradlew run` aus (`.\gradlew.bat run` unter Windows).

Sollten Sie IntelliJ nutzen, können Sie das Projekt auch direkt über die grafische Oberfläche starten. Klicken Sie dazu in der rechten Leiste auf das Gradle-Logo (Elefant) und navigieren Sie zum „run“-Task (siehe Abbildung 1). Mittels Rechtsklick bzw. Sekundärklick können Sie das Projekt ausführen („Run“) oder auch den visuellen Debugger nutzen („Debug“). Die Option „Modify Run Configuration. . . “ erlaubt es Ihnen, die Ausführung permanent als „Run/Debug Configuration“ zu hinterlegen.

Anschließend wird Ihnen in der Konsole eine Auswahl an Testinstanzen angeboten, darunter befindet sich zumindest `abgabe.csv`:

```
Select an instance set or exit:
[1] abgabe.csv
[0] Exit
```

Durch die Eingabe der entsprechenden Ziffer kann entweder eine Testinstanz ausgewählt werden oder das Programm (mittels der Eingabe von `0`) verlassen werden. Wird eine Testinstanz gewählt, dann wird der von Ihnen implementierte Programmcode ausgeführt. Kommt es dabei zu einem Fehler, wird ein Hinweis in der Konsole ausgegeben.

Um nur eine bestimmte Subinstanz auszuführen, hängen Sie `:[Subinstanznummer]` an Ihre Auswahl an (z.B. `1:123`).

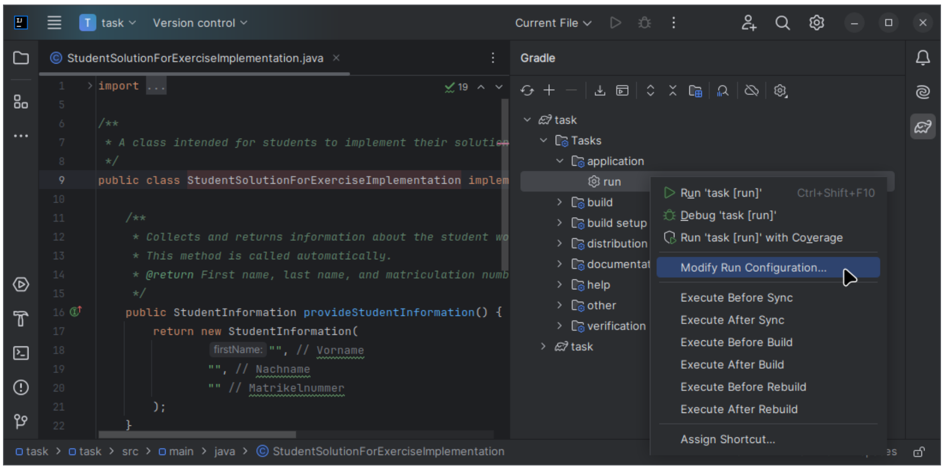

Abbildung 1: Ausführen von Gradle-Tasks in IntelliJ

Relevant für die Abgabe ist das Ausführen der Testinstanz `abgabe.csv`.

Die weiteren Testinstanzen `feasible.csv`, `clcs.csv` und `lcs.csv` sind nur zum Testen der jeweiligen einzelnen Unteraufgaben gedacht. Beachten Sie, dass letztere beiden Testinstanzen nur dann lauffähig sind, wenn Sie die Methode `void computeDynamicProgrammingTable(CLCSInstance instance, DynamicProgrammingTable table)` aus Abschnitt 5.2 implementiert haben.

Hinweis: Für jede der genannten csv-Dateien gibt es eine kleine Version, die nur Instanzen mit $n \leq 50$ enthält, wobei $n = \max\{n_1, n_2\}$ ist. Diese kleinen Instanzen eignen sich für das schnelle Testen der Funktionalität.

7 Evaluierung

Wenn der von Ihnen implementierte Programmcode mit der Testinstanz `abgabe.csv` ohne Fehler ausgeführt werden kann, dann wird nach dem Beenden des Programms im Ordner `results` eine Ergebnisdatei mit dem Namen `solution-abgabe.csv` erzeugt. Beachten Sie, dass Sie die gesamte Testinstanz ausführen und nicht nur eine Subinstanz!

Die Datei `solution-abgabe.csv` beinhaltet Zeitmessungen der Ausführung der Testinstanz `abgabe.csv`, welche in einem Webbrowser visualisiert werden können. (Auch Ergebnisdateien anderer Testinstanzen können zu Testzwecken visualisiert werden.) Öffnen Sie dazu die Datei `visualization.html` in Ihrem Webbrowser und klicken Sie rechts oben auf den Dateiauswahlknopf, um `solution-abgabe.csv` auszuwählen.

Beantworten Sie basierend auf der Visualisierung die Fragestellungen aus dem folgenden Abschnitt.

8 Fragestellungen

Öffnen Sie `solution-abgabe.csv` und bearbeiten Sie folgende Aufgaben- und Fragestellungen:

1. Durch Klicken auf Gruppennamen in der Legende neben der Plots, lassen sich einzelne Gruppen aus- bzw. einblenden. Studieren Sie die Laufzeiten für Ihre implementierte Methode `isFeasible()` aus Abschnitt 5.1. Blenden Sie alles bis auf *Feasible positiv* (die Positivfälle mit Ergebnis `true`) und *Feasible negativ* (die Negativfälle mit Ergebnis `false`) aus. Auf der x-Achse ist $n = \max\{n_1, n_2\}$ und auf der y-Achse ist die Laufzeit abgebildet. Was beobachten Sie? Wie groß ist Ihrer Meinung nach der Einfluss von n_p sowie der Länge der zu überprüfenden Lösung auf das Laufzeitverhalten? Werden die Positivfälle oder die Negativfälle schneller ausgewertet? Drücken Sie im Anschluss in der Menüleiste rechts über dem Plot auf den Fotoapparat, um den Plot als Bild zu speichern.

2. Blenden Sie alles bis auf *CLCS and LCS* aus. Es werden die Laufzeiten für die Berechnung der Tabelle aus Abschnitt 5.2 sowie der Backtracking-Methoden aus den Abschnitten 5.3 und 5.4 addiert und dargestellt. Was beobachten Sie in Hinblick auf das Laufzeitverhalten? Drücken Sie im Anschluss in der Menüleiste rechts über dem Plot auf den Fotoapparat, um den Plot als Bild zu speichern.

3. Unterhalb der Laufzeitgrafiken ist ein Streudiagramm abgebildet. Auf der x-Achse ist $|s_p|/n$ (mit $n = \max\{n_1, n_2\}$), also der Quotient aus der Länge der Sequenz s_p und der Länge der längeren Sequenz, und auf der y-Achse $|\mathrm{sol}_{\mathrm{CLCS}}|/|\mathrm{sol}_{\mathrm{LCS}}|$, also der Quotient aus der Lösungslänge der **CLCS**-Variante und jener der **LCS**-Variante abgebildet. Bei der Erzeugung der Instanzen wurden zwei Alphabete verwendet, eines mit vier und eines mit zwanzig Zeichen. Die beiden Alphabete werden farblich unterschieden. Was beobachten Sie? Gibt es einen Zusammenhang und wenn ja, wie sieht dieser aus? War dieser

Zusammenhang so zu erwarten? Erstellen Sie ebenfalls ein Bild des Plots.

4. Unterhalb des Streudiagramms können Sie die von Ihnen erstellten Tabellen der dynamischen Programmierung für eine Instanz sowie die berechneten Lösungen für die **LCS**- und **CLCS**-Variante der Instanz abbilden. Werte, die kleiner als -99 sind, werden durch -Inf dargestellt. Wählen Sie nun die Instanz *Nummer 327 (n = 6, Alphabetgröße = 4)* aus (es handelt sich dabei um die Beispielinstanz aus Abschnitt 4). Erklären Sie anhand der Tabelle, wie die Tabelleneinträge t_{kij} berechnet werden. Erklären Sie außerdem, wie Ihre implementierten Backtracking-Methoden eine Optimallösung für das **CLCS**-Problem bestimmen. Woran würden Sie allgemein erkennen, dass es **keine** zulässige Lösung für das **CLCS**-Problem gibt?

5. Woran erkennen Sie, dass es keine eindeutige Lösung für das **LCS**-Problem gibt? Suchen Sie sich eine Instanz heraus, bei der die Lösung für die **LCS**-Variante nicht eindeutig ist und bestimmen Sie (per Hand) zumindest zwei optimale Lösungen. Dokumentieren Sie, wie Sie die jeweilige Lösung erhalten.

Falls sich im Zuge der Evaluierung die Darstellung der Plots auf ungewünschte Weise verändert (z.B. durch die Auswahl eines zu kleinen Ausschnitts), können Sie mittels Doppelklick auf den Plot oder Klick auf das Haus in der Menüleiste die Darstellung zurücksetzen.

Fügen Sie Ihre Antworten in einem Bericht gemeinsam mit allen erstellten Bildern der Visualisierungen der Testinstanz `abgabe.csv` zusammen.

9 Abgabe

Laden Sie die Datei `task/src/main/java/StudentSolutionForExerciseImplementation.java` in der TUWEL-Aktivität *Hochladen Source-Code P6* hoch. Fassen Sie diesen Bericht mit den anderen für das zugehörige Abgabegespräch relevanten Berichten in einem PDF zusammen und geben Sie dieses in der TUWEL-Aktivität *Hochladen Bericht Abgabegespräch 2* ab.

10 Nachwort

Dynamische Programmierung ist eine sehr vielseitig einsetzbare Methode im Algorithmenentwurf. Sie kombiniert das Teile-und-Herrsche Prinzip und die Rekursion, wobei das Zwischenspeichern von optimalen Lösungen unabhängiger Teilprobleme immer eine ganz zentrale Rolle spielt. Nur so lassen sich kostspielige Mehrfachberechnungen in den rekursiven Aufrufen vermeiden. Auch wenn wir die dynamische Programmierung in der Vorlesung im breiteren Kontext von NP-schweren Optimierungsproblemen behandelt haben, führt sie üblicherweise zu Polynomialzeitalgorithmen, deren Laufzeit einerseits von der Größe der Lösungstabelle abhängt und andererseits vom Zeitbedarf für die Berechnung eines einzelnen Tabelleneintrags.

In der Vorlesung haben wir bereits mehrere grundlegende Beispiele gesehen, wo sich dynamische Programmierung einsetzen lässt. In der Praxis gibt es aber viele weitere Beispiele für den Einsatz solcher Algorithmen, z.B. bei der Analyse von Genom- und Aminosäuresequenzen in der Bioinformatik (ähnlich dieser Programmieraufgabe), generell bei der Mustersuche in Texten, im Compilerbau bei der Erkennung kontextfreier Sprachen, bei der Optimierung des Seitenlayouts von Texten (z.B. in LaTeX) oder bei der Betrachtung von möglichen Spielzügen z.B. im Schach und ähnlichen Spielen. Weiters spielt die dynamische Programmierung eine wichtige Rolle bei der Berechnung von NP-schweren Graphenproblemen, falls die betrachteten Graphen die Eigenschaft der sogenannten beschränkten Baumweite (treewidth) haben. Das bedeutet, dass sie zwar keine Bäume mehr sein müssen, aber doch noch eine gröbere baumartige Gesamtstruktur haben. In diesem Fall lassen sich viele allgemeine NP-schwere Probleme durch bottom-up Verfahren auf einem zugehörigen Lösungsbaum in (parametrisierter) Polynomialzeit berechnen. Diese Konzepte lernen Sie im Masterstudium kennen, z.B. in den Lehrveranstaltungen 186.814 VU Algorithmics, , und 186.856 Structural Decompositions and Algorithms.

In der Forschung verwenden wir dynamische Programmierung darüber hinaus auch oft bei geometrischen Optimierungsproblemen und in der Graphenvisualisierung, in denen sich Probleminstanzen durch optimal zu wählende geometrische Trennkurven in kleinere, unabhängige Teile zerlegen lassen aus denen sich dann die Gesamtlösung zusammensetzen lässt. Dies taucht z.B. bei der Berechnung von optimalen Beschriftungen von Landkarten oder technischen Zeichnungen mit durch Linien verbundenen Zusatzinformationen am Rand auf. Daher kann es gut sein, dass Sie später in Projekten, Seminaren, oder Abschlussarbeiten auch wieder mit dynamischer Programmierung zu tun haben werden. In jedem Fall gehört die dynamische Programmierung für Sie als künftige Informatiker_innen als eine wichtige Entwurfstechnik in Ihren algorithmischen Werkzeugkasten.