**Dynamische Programmierung** 7. November 2016

# Handout 7: DP Aufgabensammlung

Sebastian Millius, Sandro Feuz, Daniel Graf

**Thema:** Dynamic Programming, Aufgaben

Im folgenden sind ein paar Beispielaufgaben zur dynamischen Programmierung zusammengetragen. Jede ist mit einem (subjektiven) Schwierigkeitsgrad von 1 (leicht) bis 5 (schwer) versehen. Ich werde keine Lösungen zu den Aufgaben erstellen, wenn jedoch jemand längere Zeit an der Aufgabe verbracht hat und dann gerne die Aufgabe vorgelöst bekommen möchte, so werde ich dies in der jeweils nächsten Übungsstunde machen. Es gilt jedoch: selbst lösen bringt viel mehr als nachvollziehen!

**Beispiel: The Integer Knapsack Problem (Doppelte Items erlaubt)**
Gegeben sind $n$ Arten von Gegenständen, wobei die $i$te Art ein Ganzzahlgewicht von $w_i$ und einen Wert von $v_i$ hat. Versuche einen Rucksack mit Tragekapazität $C$ mit einer optimalen Auswahl der Gegenstände zu füllen. Man kann eine unbeschränkte Anzahl Gegenstände der selben Art mitnehmen.

*Lösung*: Es sei $M(j)$ der Maximale Wert für einen Rucksack der Kapazität $j$. Wir können $M(j)$ rekursiv mit Hilfe der Lösung für Teilprobleme auf die folgende Art ausdrücken

$$M(j) = \begin{cases} \max\left\{ M(j-1), \max\limits_{i=1\ldots n, w_i \leqslant j} M(j-w_i) + v_i \right\} & \text{falls } j \geqslant 1 \\ 0 & \text{falls } j \leqslant 0 \end{cases}$$

In einer bottom-up Art kann jedes $M(j)$ in $\mathcal{O}(n)$ sequentiell berechnet werden. Die totale Laufzeit ist deshalb $\mathcal{O}(nC)$. Der Speicherverbrauch ist $\mathcal{O}(C)$. Die Wert der optimalen Auswahl ist durch $M(C)$ gegeben. Die Auswahl selbst kann durch Rückverfolgung oder durch Mitführen von "Backpointern" erhalten werden.

## Wechselgeld (1)

Gegeben sind eine Menge von $n$ von verschiedene Münzenarten mit Wert $c[1], \ldots, c[M]$. Von jeder Art sind beliebig viele Münzen verwendbar. Finde zu einem gegebenen Geldwert $W$, ob es möglich ist, diesen mit den gegebenen Münzen zu bezahlen. Falls es möglich ist, finde eine kleinste Kombination von Münzen, die den gewünschten Wert ergeben.

## Wechselgeld 2 (1)

Gegeben sind eine Menge von $n$ von verschiedene Münzenarten mit Wert $c[1], \ldots, c[M]$. Von jeder Art sind beliebig viele Münzen verwendbar. Finde zu einem gegebenen Geldwert $W$, die Anzahl Möglichkeiten, mit den Münzen den Geldwert zu wechseln.

## Handys (1)

Gegeben sind die $n$ Handys deiner Freunde die jeweils eine Länge $L[i]$, $i = 1, .., n$ haben. Gesucht ist eine Teilmenge dieser Handys, so dass die Gesamtlänge genau einen Meter misst.

## Waage (1)

Bei den alten Balance-Waagen legt man das zu wägende Gut auf die eine Waagschale und belegt dann die Waagschalen so mit geeichten Gewichtssteinen, dass die Waage im Gleichgewicht ist. Wir verfügen über $n$ Gewichtssteine mit Gewichten $w[i]$, $i = 1, ..., n$.

Gegeben ist ein Gewicht $W$, bestimme, ob das geforderte Gewicht mit den zur Verfügung stehenden Gewichtssteinen abgewogen werden kann.

## Bücher (2)

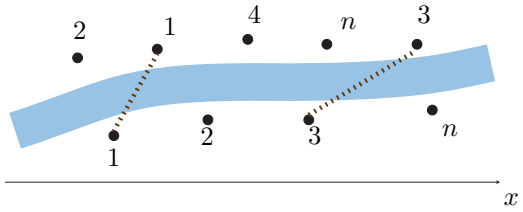
Gegeben sind eine Menge $n$ von Büchern und die Breite $B$ des Bücherregals. Jedes der $n$ Bücher hat eine Breite $b[i]$ und einen Wert $v[i]$ (der ursprüngliche Kaufpreis). Gesucht ist eine Teilmenge der Bücher, die alle auf das Regal passen und deren gesamter Wert zusammen maximal ist.

## Türme (2)

Gegeben sind $n$ Holzklötze, die jeweils eine Höhe $h(i)$, Breite $b(i)$ und Tiefe $t(i)$ haben. Gesucht ist der grösst möglichste Turm der mit diesen Bauklötzen gebaut werden kann, wobei ein Holzklotz nur auf einen anderen gestellt werden kann, wenn die Auflagefläche des unteren strikt grösser ist als die des Klotzes der daraufgestellt werden kann (d.h. sowohl Breite und Länge der Auflagefläche des oberen Klotzes müssen strikt kleiner sein als die des unteren Klotzes). Die Klötze dürfen um die Z-Achse um ein Vielfaches von 90 Grad gedreht werden (Höhe bleibt also, der Klotz darf nicht gekippt werden).

## Brücken (2)

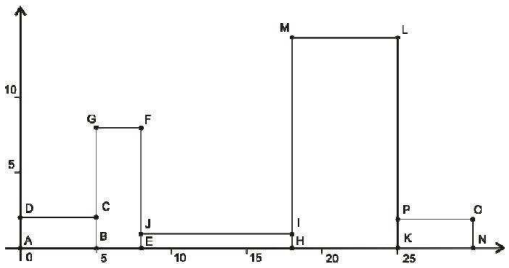
Gegeben ist eine 2D-Karte mit einem einem Fluss. An der unteren und oberen Seite des Flusses sind jeweils $n$ Städte plaziert. Die $x$-Koordinanten der unteren Städte ist gegeben durch $a[1], \ldots, a[n]$. Jede Stadt hat eine Partner-Stadt auf der anderen Seite mit der sie Handel betreibt. Die $x$-Koordinate der Partner-Stadt $i$ ist gegeben durch $b[1], \ldots, b[n]$.




Man möchte nun die Partner-Städte mit Brücken verbinden. Allerdings dürfen sich keine zwei Brücken kreuzen. Wieviele Brücken können maximal gebaut werden?

## Rechtecke (2)

Gegeben sind $N$ Rechtecke $R_1, \ldots, R_N$, jedes mit einer festen Höhe und Breite. Die Rechtecke sollen in der gegebenen Reihenfolge aneinandergereiht werden, wobei jeweils eine Rechtecksseite auf der X-Achse liegt. Du kannst für jedes Rechteck bestimmen, welche Orientierung (ob "stehend" oder "liegend") es haben soll und sollst dadurch die obere Seite des entstehenden Polygons maximieren. Die obere Linie ist einfach der Umfang des ganzen Polygons ohne die linke, rechte und untere Seite. In folgendem Bild wäre die obere Linie $\overline{DC} + \overline{CG} + \overline{GF} + \overline{FJ} + \overline{JI} + \ldots + \overline{PO}$.




## Integer Knapsack Problem (2)

Das gleiche Problem wie das Beispiel oben, nur dass es nur ein Gegenstand jeder Art gibt.

Löse das gleiche Problem, wenn es von der $i$ten Art $a[i] \geqslant 1$ Gegenstände gibt.

## Balanced Partition (3)

Gegeben sind eine Menge von $n$ Ganzzahlen, jede im Bereich $0 \ldots K$. Teile diese Zahlen in zwei Mengen so dass $|S_1 - S_2|$ minimal ist, wobei $S_1$ und $S_2$ die Summe der Elemente in den zwei Teilmengen bezeichnet.

## Längste gemeinsame Teilfolge (3)

Gegeben sind zwei Folgen $X[1..n], Y[1..m]$. Eine Folge $Z[1..k]$ heisst *Teilfolge* einer Folge $X[1..n]$, falls es aufsteigende Indizes $i_1 < i_2 < ... < i_k$ gibt, so dass $\forall j = 1, ..., k : Z[j] = X[i_j]$ ist.

Z.B.: $X = \{B, C, A, D\}$ und $Z = \{B, A\}$.

Eine Folge $Z$ heisst *gemeinsame Teilfolge* von $X$ und $Y$, falls $Z$ sowohl eine Teilfolge von $X$, als auch von $Y$ ist.

Gesucht ist für zwei gegebene Folgen $X[1..n], Y[1..m]$ die Länge der längsten gemeinsamen Teilfolge.

## Levenshtein Distanz (3)

Die Levenshtein Distanz zwischen zwei Wörtern ist die minimale Anzahl von *Einfügen*, *Löschen*, und *Ersetzen* Buchstaben Operationen um ein Wort in ein anderes zu ändern. Z.B. ist die Levenshtein Distanz zwischen $FOOD$ und $MONEY$ höchstens 4, da: FOOD $\rightarrow$ MOOD $\rightarrow$ MOND $\rightarrow$ MONED $\rightarrow$ MONEY

Für zwei gegebene Wörter, berechne ihre Levenshtein Distanz.

## Palindrom (3)

Gegeben ein String von Character: $c_1, \ldots, c_n$. Eine Teilsequenz ist ein *Palindrom* falls sie vorwärts wie rückwärts gleich geschrieben ist. Zum Beispiel ist "a,b,a,c,a,b,a" ein Palindrom. Gebe einen $\mathcal{O}(n^2)$ Algorithmus an, der das längste Palindrom in einem String findet.

## Baummatching (3)

Gegeben ist ein gewichteter Baum, das heisst jede Kante des Baums hat ein bestimmtes Gewicht. Ein Matching auf einem Baum ist eine Teilmenge der Kanten, so dass keine zwei Kanten einen selben Endknoten haben (keine zwei Kanten berühren sich). Gesucht ist ein Matching, bei dem die Summe der Gewichte der darin enthaltenen Kanten, maximal ist. (ehemalige Prüfungsaufgabe)

## *Hübsche* Folgen (3)

Eine Folge $S$ (der Länge $n$) heisst *hübsch*, falls sie folgende Eigenschaften erfüllt:

- $\sum_{i=1}^{n} S_i = 0$
- Keine startende Teilfolge $S_1, S_2, ...S_i$ hat eine negative Summe.
- Keine startende Teilfolge $S_1, S_2, ...S_i$ hat eine Summe grösser $C$ (const.).

Eine Folge, die nicht *hübsch* ist, heisst *hässlich*. Sei $C = 10$, dann ist $1, -1, 7, 3, -6, 1, -5$ eine *hübsche* Folge wogegen die Folgen $-1, 1, 7, 3, -6, 1, -5, 1, -1, 8, 3, -7, 1, -5, 1, -1, 7, 3, -6, 2, -5$ *hässlich* sind.

Gegeben ist eine *hässliche* Folge und ein festes $C < 1000$. Die Folge soll durch Löschen von möglichst wenig Elementen in eine *hübsche* Folge verwandelt werden. Wieviele Elemente musst du mindestens löschen?

## Cutting Sticks (4)

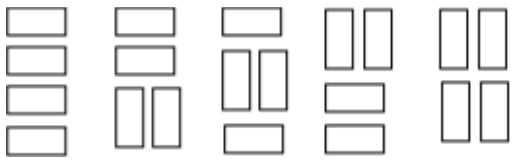
Gegeben ist ein Holzbrett der Länge $l$, das an $n$ verschiedenen Stellen durchgesägt werden soll. Das Durchsägen eines Holzstückes kostet soviel wie die Länge des Holzes. Es ist einfach einzusehen, dass die Reihenfolge der Schnitte wichtig für den Gesamtpreis ist.

Betrachte beispielsweise ein Brett der Länge 10m das an den Stellen 2,4 and 7 Meter von einem Ende durchgesägt werden soll. Es gibt verschiedene Möglichkeiten. Man kann zuerst an der Stelle 2 dann 4 dann 7 durchsägen. Was zu einem Preis von $10+8+6 = 24$ führt, da das erste Stück 10 Meter, das resultierende 8 und das letzte 6 Meter lang war. Eine andere Reihenfolge wäre zuerst bei 4 dann bei 2 und zuletzt bei 7 zu schneiden. Der Preis wäre dann $10 + 4 + 6 = 20$ was besser ist.

Für ein gegebenes Holzbrett und Schnittpunkte, finde eine optimale Reihenfolge für das Sägen.

## Domino (4)

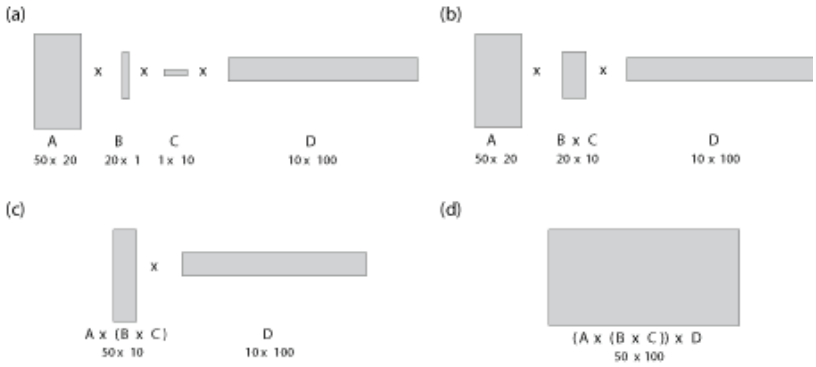
Gegeben ist eine spezielles Schachbrett. Es ist 4 Felder hoch und $N$ Felder breit und soll volständig mit Dominosteinen belegt werden. Jeder Domino besetzt zwei (nebeneinanderliegende) Felder. Deine Aufgabe ist es herauszufinden, auf wieviele Arten das Schachbrett für ein bestimmtes $N$ vollständig mit Steinen belegt werden kann. Für $N = 2$ ist die Antwort 5, du siehst alle 5 Möglichkeiten in folgendem Bild:



## Matrix Kettenprodukt (4)

Angenommen wir wollen die 4 Matrizen, $A$, $B$ ,$C$ ,$D$, mit den Dimensionen $50 \times 20$, $20 \times 1$, $1 \times 10$, und $10 \times 100$ multiplizieren.

Matrix Multiplikation ist nicht kommutativ (im Allgemeinen, $A \times B \neq B \times A$), aber assoziativ, d.h. z.B. $A \times (B \times C) = (A \times B) \times C$. Das heisst wir können das Produkt auf viele verschiedene Arten ausrechnen, abhängig davon, wie wir die Klammern setzen. Sind manche besser als andere?




Um eine $m \times n$ Matrix mit einer $n \times p$ Matrix zu multiplzieren sind $m \cdot n \cdot p$ Multiplikationen notwendig. Mit dieser Formel kommt man für das Beispiel auf:

$A \times B \times C \times D$:

| Klammerung | Kosten | Total |
|---|---|---|
| $A \times ((B \times C) \times D)$ | $20 \cdot 1 \cdot 10 + 20 \cdot 10 \cdot 100 + 50 \cdot 20 \cdot 100$ | 120'200 |
| $(A \times (B \times C)) \times D$ | $20 \cdot 1 \cdot 10 + 50 \cdot 20 \cdot 10 + 50 \cdot 10 \cdot 100$ | 60'200 |
| $(A \times B) \times (C \times D)$ | $50 \cdot 20 \cdot 1 + 1 \cdot 10 \cdot 100 + 50 \cdot 1 \cdot 100$ | 7'000 |

Die Klammerung macht also einen erheblichen Unterschied.

Allgemein ist das Problem: Bestimme die Kosten einer Optimal Klammerung von $A_1 \times A_2 \times ... \times A_n$, wobei die Matrizen $A_i$ die Dimensionen $m_0 \times m_1, m_1 \times m_2, ...m_{n-1} \times m_n$ haben.

## (Bonus) Schneiderei (5)

Gegeben ist ein rechteckiges Stück wertvollen Stoffes mit (ganzzahliger) Länge $l$ und Breite $b$. Wir haben eine Liste von $n$ Produkten die aus diesem Stoff geschneidert werden können. Jedes Produkt $i$ braucht ein Stoffstück der Länge $a[i]$ und Breite $b[i]$. Der Verkaufspreis des $i$ten Produktes ist $c[i]$. Nehme an dass alle $a[i], b[i], c[i]$ positive natürliche Zahlen sind.

Wir haben eine Schneidemaschine die jedes rechteckige Stoffteil durchschneiden kann (entweder horizontal oder vertikal) (mögliche Schneidelinien sind also immer parallel zu der Länge oder Breite und teilen vollständig das Stoffstück).

Gebe einen Algorithmus an, der den maximalen Ertrag angibt, der aus diesem Stoff erhalten werden kann. Von jedem Produkt können so viele wie möglich produziert werden (wenn gewünscht).