Algorithmen und
Berechnungskomplexität I
Wintersemester 2022/23

Abgabe: 09.11.2022 bis 10:00 Uhr

UNIVERSITÄT BONN

Prof. Dr. Anne Driemel
Frederik Brüning, Jan Eube
Institut für Informatik

Übungsblatt 4

Aufgabe 4.1: Rucksackproblem (5 Punkte)

Wir betrachten das Rucksackproblem. Gegeben sind 5 Gegenstände der Form $a_i = (w_i, p_i)$ durch

$$\{a_1 = (1,3), a_2 = (2,4), a_3 = (4,7), a_4 = (3,5), a_5 = (7,11)\}.$$

Der erste Wert bezeichnet das Gewicht w_i, der zweite den Nutzen der Gegenstände p_i; das Gewicht von a_1 ist beispielsweise $w_1 = 1$ und dessen Nutzen ist $p_1 = 3$.

Bestimmen Sie für einen Rucksack mit Kapazität 10 eine optimale Lösung für das Rucksackproblem. Geben Sie an, welche Gegenstände den Rucksack optimal füllen. Stellen Sie hierzu, wie in der Vorlesung besprochen, eine Tabelle von Teillösungen auf um dieses Rucksackproblem mithilfe dynamischer Programmierung zu lösen. Die Tabelle soll hierbei für jedes Teilproblem neben den erzielten maximalen Nutzen auch eine binäre Variable enthalten, welche angibt, ob der in der entsprechenden Zeile neu berücksichtige Gegenstand Teil der optimalen Lösung ist (entspricht dem Wert 1) oder nicht (entspricht dem Wert 0). Geben Sie an, wie die optimale Lösung aus der in der Tabelle gespeicherten Information konstruiert werden kann.

Aufgabe 4.2: Wechselgeldproblem (5 Punkte)

Sei ein Betrag Z sowie eine Liste an Münzwerten $c_1, ..., c_n$ (wobei n beliebig groß sein kann) gegeben. Ziel ist es nun eine minimale Anzahl an Münzen zu finden, sodass die Summe der Münzwerte Z entspricht. Beachten Sie hierbei, dass Münzen mehrfach verwendet werden können und, dass es potentiell unmöglich ist Z mit den gegebenen Münzen zu bilden (z.B. Wenn es nur die Münzwerte $c_1 = 2$ und $c_2 = 5$ gibt, kann damit nicht $Z = 3$ gebildet werden).

(a) Geben Sie eine Rekursion R an, welche für ein gegebenes Münzsystem $c_1, ..., c_n$ für ein beliebiges i berechnet, wie viele Münzen benötigt werden um i darzustellen (falls i nicht gebildet werden kann soll gelten, dass $R(i) = \perp$). Sie können hierbei annehmen, dass die Münzwerte $c_1, .., c_n$ aufsteigend sortiert sind. Beweisen Sie die Korrektheit Ihrer Rekursion mittels Induktion.

(b) Nutzen Sie Ihre Rekursion aus Aufgabenteil (a) um mithilfe Dynamischer Programmierung einen möglichst effizienten Algorithmus zu entwerfen, welcher entscheidet wie viele Münzen minimal benötigt werden um Z zu bilden. Analysieren Sie die Laufzeit Ihres Algorithmus (sie sollte in $O(nZ)$ liegen).

Aufgabe 4.3: Dynamische Programmierung (5 + 5 = 10 Punkte)

Gegeben sei ein String (Zeichenkette) als Array $S[1, \ldots, n]$. Eine *Palindrom-Zerlegung* von S der Größe k ist eine Zerlegung von $S[1, \ldots, n]$ in $A_1 = S[1, \ldots, i_1]$, $A_2 = S[i_1 + 1, \ldots, i_2]$, $\ldots$, $A_k = S[i_{k-1} + 1, \ldots, i_k]$, wobei $i_1 < i_2 < \cdots < i_k$ und $i_k = n$ gilt, sodass jedes A_i ein *Palindrom* ist. Eine Zeichenkette wird dabei als Palindrom bezeichnet, wenn sie vor- und rückwärts gelesen denselben Text ergibt. Im Folgenden wollen wir einen Algorithmus mit Laufzeit $O(n^2)$ entwerfen, der für einen gegebenen String S das kleinste $k \in \mathbb{N}$ berechnet, sodass S eine Palindrom-Zerlegung $A_1, \ldots, A_k$ besitzt.

Beispiel: Der String `ANNAMAGOTTO` besitzt zwei kleinste Palindrom-Zerlegungen der Größe 5:

- {`ANNA`, `M`, `A`, `G`, `OTTO`} und
- {`A`, `NN`, `AMA`, `G`, `OTTO`}.

Bearbeiten Sie zu diesem Zweck die folgenden Teilaufgaben:

(a) Konstruieren Sie zunächst einen Algorithmus, der ein Array $L[i, j]$ mit $i, j \in \{1, \ldots, n\}$ und $i \leq j$ berechnet, das angibt, ob das Teilwort $S[i, \ldots, j]$ ein Palindrom ist oder nicht.

Hinweis: Falls Sie Teilaufgabe (a) nicht lösen können, nehmen Sie für die weiteren Teilaufgaben das Array $L[i, j]$ als gegeben an.

(b) Nutzen Sie das Array aus Teilaufgabe (a) um einen Algorithmus zu entwerfen, welcher das kleinste k bestimmt, sodass S in k Palindrome zerlegt werden kann.

Gehen Sie bei beiden Teilaufgaben in zwei Schritten vor: Geben Sie zunächst eine geeignete Rekursion an, die das Problem löst. Nutzen Sie daraufhin dynamische Programmierung um einen Algorithmus mit Laufzeit $O(n^2)$ zu entwerfen. Begründen Sie die Korrektheit ihrer Lösung in jedem Schritt. Zeigen Sie dass ihr Algorithmus die Laufzeit $O(n^2)$ hat.