technische universität dortmund

SoSe 2018

M. Buchin
A. Krivošija, D. Mezlaf
L. Pfahler, A. Rey

DAP2 – Heimübung 8

Ausgabedatum: 22.05.2018 — Abgabedatum: Mo. 04.06.2018 bis 12 Uhr

Abgabe:

Schreiben Sie unbedingt immer Ihren vollständigen Namen, Ihre Matrikelnummer und Ihre Gruppennummer auf Ihre Abgaben! Beweise sind nur dort notwendig, wo explizit danach gefragt wird. Eine Begründung der Antwort wird allerdings *immer* verlangt.

Aufgabe 8.1 (5 Punkte): (Dynamische Programmierung: Geschickter Wahlkampf)

In k Tagen findet eine Wahl statt. Bob kann an Tag i mit Wahlwerbung a_i neue Wähler gewinnen. Allerdings ist das Werben von Wählern so anstrengend, dass er spätestens nach je zwei Tagen einen Tag Pause braucht. Sein Ziel ist es, so viele neue Wähler wie möglich zu gewinnen.

Wir betrachten ein Array A der Länge k, in dem die Werte $A[i] = a_i > 0$, $1 \leq i \leq k$, gespeichert sind.

Beispielsweise sei $A = [2, 3, 4, 5, 6]$, dann kann Bob am 1., 2., 4. und 5. Tag die maximale Anzahl, nämlich 16 neue Wähler, erreichen.

a) Es bezeichne $D(i, j)$, $1 \leq i \leq k$, $j \in \{0, 1\}$, die maximale Anzahl an neuen Wählern, die Bob an den ersten i Tagen erreichen kann, wenn er entweder Tag i hinzunimmt ($j = 1$) oder an Tag i eine Pause macht ($j = 0$). Geben Sie eine rekursive Form für $D(i, j)$ an.

b) Geben Sie einen Algorithmus in Pseudocode an, der auf dem Prinzip der dynamischen Programmierung beruht und die maximale Anzahl an neuen Wählern zurückgibt, die Bob gewinnen kann.

c) Analysieren Sie die Laufzeit Ihres Algorithmus.

d) Beweisen Sie die Korrektheit Ihrer rekursiven Form aus Teilaufgabe a) mittels Induktion.

Lösung:

a) Sei $D(i, j)$ die maximale Anzahl an neuen Wählern, die Bob an den ersten i Tagen erreichen kann, wenn er entweder Tag i hinzunimmt ($j = 1$) oder an Tag i eine Pause macht ($j = 0$). Wir beobachten folgende Grundsätze:

- Rekursionsabbruch (1): Für $i = 0$ hat Bob keine Wahlkampftage zur Verfügung, kann also keine Wähler gewinnen, also setzen wir $D(0, 1) = D(0, 0) = 0$.

- Rekursionsabbruch (2): Für $i = 1$ ergibt sich die maximale Anzahl an Wählern am ersten möglichen Tag aus der Entscheidung, ob Bob eine Pause macht ($D(1,0) = 0$) oder nicht ($D(1,1) = A[1]$).
- Rekursion (1): Von drei aufeinander folgenden Tagen muss *mindestens* ein Tag Pause eingelegt werden. Falls Bob am Tag i Werbung macht, muss also entweder am Tag $i-1$ oder am Tag $i-2$ eine Pause gemacht werden. Wenn am Tag $(i-2)$ eine Pause eingelegt wurde, können an den Tagen $i-1$ und i Wähler geworben werden. Bei einer Pause am Tag $i-1$ bleiben zumindest die Wähler des aktuellen Tages.
- Rekursion (2): Falls Bob am Tag i keine Werbung macht, legt er hier eine Pause ein und übernimmt die bestmögliche Anzahl der bis hierhin gesammelten Wähler. Dies kann fÃijr ihn Wahlwerbung am Tag $i-2$ oder am Tag $i-1$ bedeuten, da drei aufeinanderfolgende Pausen immer durch Wahlwerbung am mittleren dieser drei Tage verbessert werden können.

Damit haben wir folgende rekursive Formulierung:

$$D(i,j) = \begin{cases} 0 & \text{falls } i = 0 \\ 0 & \text{falls } i = 1 \text{ und } j = 0 \\ A[1] & \text{falls } i = 1 \text{ und } j = 1 \\ A[i] + \max\{D(i-2,0) + A[i-1], D(i-1,0)\} & \text{falls } i \geq 2 \text{ und } j = 1 \\ \max\{D(i-2,1), D(i-1,1)\} & \text{falls } i \geq 2 \text{ und } j = 0 \end{cases}$$

b) Auf der Basis der rekursiven Definition ergibt sich das folgende dynamische Programm zur Berechnung der maximalen Anzahl an Wählern nach i Tagen, wenn die Werte der a_i im Array $A[1..k]$ übergeben werden.

```
Wahlwerbung(Array A):
1  k ← length(A)
2  D ← new Array D[0..k, 0..1]
3  D[0,0] ← 0
4  D[0,1] ← 0
5  D[1,0] ← 0
6  D[1,1] ← A[1]
7  for i ← 2 to k do
8      D[i,0] ← max{D[i-2,1], D[i-1,1]}
9      D[i,1] ← A[i] + max{D[i-2,0] + A[i-1], D[i-1,0]}
10 return max{D[k,0], D[k,1]}
```

c) Bis auf die Allokation des Arrays und die einfache For-Schleife haben alle vorkommenden Zeilen eine konstante ($\mathcal{O}(1)$) Laufzeit. Da die zweite Dimension des Arrays D eine Länge von 2 hat, wird hier ein Array mit $2 \cdot (k+1) \in \mathcal{O}(k)$ allokiert, die Laufzeit der Zeile 2 ist also mit $\mathcal{O}(k)$ abzuschätzen. Auch die For-Schleife trägt eine Laufzeit von $\mathcal{O}(k)$ bei, da sie in jeder ihrer $k-1$ Iterationen zwei Anweisungen konstanter Laufzeit ausführt. Er ergibt sich also für das gesamte Programm eine Laufzeit von $\mathcal{O}(k)$.

d) Wir zeigen die Korrektheit der rekursiven Form mittels vollständiger Induktion.

Behauptung: $D(i,j)$ gibt die maximale Anzahl an Wählern an, die Bob an den ersten i Tagen erreichen kann, wenn er entweder Wahlkampf an Tag i betreibt ($j = 1$) oder an Tag i eine Pause macht ($j = 0$).

I.A. Sei $i = 0$. Dann gibt es keine Tage, an denen Bob Wahlwerbung betreiben kann. Somit kann Bob auch keine Wähler gewinnen. Es ist also $D(0,0) = D(0,1) = 0$. Sei $i = 1$. Dann hat Bob nur genau den ersten Tag zur Verfügung, um mögliche Wähler zu gewinnen. Für $j = 1$ gilt $D(1,1) = a_1$, beträgt also die Anzahl der Stimmen, die Bob für sich gewinnen kann. Für $j = 0$ muss nach Definition an diesem Tag eine Pause eingelegt werden. Es gilt $D(1,0) = 0$ und bei einer Pause gewinnt Bob keine Wählerstimmen.

I.V. Seien $1 < i \leq k$. Dann ist $D(i', j)$ für jedes $j \in \{0,1\}$ und jedes $0 \leq i' < i$ die maximale Anzahl an Wählern an, die Bob an den ersten i' Tagen erreichen kann.

I.S. Sei $1 < i \leq k$. Nehmen wir an, $D(i,j)$ sei nicht optimal im Sinne der Aufgabenstellung. Die optimale Strategie erzielt also eine größere Anzahl an möglichen Wählern v mit $v > D(i,j)$. Wir unterscheiden zwei Fälle mit je zwei Unterfällen:

1. Fall Die bessere Strategie mit Wert v **betreibt Wahlkampf** am Tag i und erzielt mehr als $D(i,1)$ Wählerstimmen:
Betreibt die bessere Strategie auf am Tag $i-1$ Wahlkampf, so muss sie am Tag $i-2$ nach Aufgabenstellung Pause machen. Es gilt $v > D(i,1) = A[i] + \max\{D(i-2,0) + A[i-1], D(i-1,0)\} \geq A[i] + A[i-1] + D(i-2,0)$. Streicht man die Tage i und $i-1$ der besseren Strategie, ergibt dies also eine Strategie, die am Tag $i-2$ keinen Wahlkampf betreibt und $v - A[i] - A[i-1] > D(i-2,0)$ Stimmen gewinnt, ein Widerspruch zur I.V. mit $D(i-2,0)$ als Optimum unter diesen Strategien. Macht die bessere Strategie am Tag $i-1$ Pause, gilt $v > D(i,1) = A[i] + \max\{D(i-2,0) + A[i-1], D(i-1,0)\} \geq A[i] + D(i-1,0)$. Streicht man den letzten Tag der besseren Strategie, erhält man eine Strategie, die am Tag $i-1$ Pause macht und $v - A[i] > D(i-1,0)$ Stimmen erzielt, ein Widerspruch zur I.V. mit $D(i-1,0)$ als Optimum unter diesen Strategien.

2. Fall Die bessere Strategie mit Wert v **macht Pause** am Tag i und erzielt mehr als $D(i,0)$ Wählerstimmen:
Betreibt die bessere Strategie auf am Tag $i-1$ Wahlkampf, gilt $v > D(i,0) = \max\{D(i-2,1), D(i-1,1)\} \geq D(i-2,1)$. Streicht man Tag i der besseren Strategie, ergibt dies eine Strategie, die am Tag $i-1$ keinen Wahlkampf betreibt und $v > D(i-1,0)$ Stimmen gewinnt, ein Widerspruch zur I.V. mit $D(i-1,0)$ als Optimum unter diesen Strategien. Macht die bessere Strategie am Tag $i-1$ Pause, so betreibt sie am Tag $i-2$ Wahlkampf (sonst könnte diese Strategie durch Wahlkampf am Tag $i-1$ verbessert werden, wäre also nicht optimal). Es gilt $v > D(i,0) = \max\{D(i-2,1), D(i-1,1)\} \geq D(i-2,0)$. Streicht man hier die Tage i und $i-1$, ergibt es eine Strategie, die am Tag $i-2$ Wahlkampf macht und $v > D(i-2,1)$ Stimmen erzielt, ein Widerspruch zur I.V. mit $D(i-2,1)$ als Optimum unter diesen Strategien.

Zur vorangegangen Fallunterscheidung möchten wir anmerken, dass im ersten Fall, in dem die als optimal angenommene Lösung v am Tag i Wahlkampf betreibt, nur mit $D(i,1)$ verglichen werden muss und nicht zusätzlich mit $D(i,0)$. Dies kommt daher, dass $D(i,1)$ den optimalen Lösungswert für Strategien mit der Einschränkung angibt, dass eben am i-ten Tag Wahlkampf betrieben wird.

Die Behauptung gilt nun nach dem Prinzip der Induktion für Eingaben beliebiger Länge k, womit die berechnete Anzahl der Wähler $D(k,0)$ bzw. $D(k,1)$ korrekt ist.

Aufgabe 8.2 (5 Punkte): (Dynamische Programmierung: Sparen mit Bonuspunkten)

Alice möchte gern das neue Pfannenset kaufen, das es im Supermarkt für 50 Bonuspunkte günstig zu erwerben gibt. Leider hat sie aktuell keine Bonuspunkte. Im Supermarkt gibt es n Produkte zu kaufen. Produkt i bringt b_i Bonuspunkte und kostet p_i Euro. Zusätzlich gibt es eine Beschränkung, dass alle gekauften Produkte unterschiedlich sein müssen. Alice möchte jetzt herausfinden, wie teuer es für sie mindestens wird, 50 Bonuspunkte zu sammeln.

Entwerfen Sie einen Algorithmus, der die Kosten einer günstigsten Auswahl von Produkten ermittelt, sodass Alice mindestens 50 Bonuspunkte erhält.

a) Wir bezeichnen mit $A(i, j)$ die minimalen Kosten, die Alice bezahlen muss, um mit einer Auswahl aus den ersten i Produkten mindestens j Bonuspunkte zu erhalten. Geben Sie ein rekursive Form für $A(i, j)$ und alle Werte $i = 0, \ldots, n$ und $j = 0, \ldots, 50$ an. Denken Sie insbesondere an die Basisfälle.

b) Geben Sie einen Algorithmus in Pseudocode an, der das Problem basierend auf der rekursiven Form mit dynamischer Programmierung löst.

c) Analysieren Sie die Laufzeit Ihres Algorithmus.

d) Beweisen Sie die Korrektheit Ihrer rekursiven Form aus Teilaufgabe a), zeigen Sie also, dass $A(i, j)$ nach Ihrer Formel wirklich den optimalen Wert für das dort beschriebene Teilproblem enthält.

Lösung:

a) Möchte Alice 0 Punkte sammeln, beträgt der Preis dazu für jede Anzahl von Produkten 0. Ohne jegliche Produkte ist es nicht möglich, Punkte zu sammeln. Diesen Preis bezeichnen wir mit ∞. Da Alice auch mehr als j Punkte sammeln darf, setzen wir $A(i, j) = 0$ für $j < 0$. Sonst betrachten wir zwei Fälle: entweder wählt Alice das Produkt j oder sie wählt es nicht. Es entsteht folgende rekursive Form:

$$A(i,j) = \begin{cases} 0 & \text{falls } i = 0, j = 0 \\ \infty & \text{falls } i = 0, j > 0 \\ 0 & \text{falls } i > 0, j \leq 0 \\ \min\{A(i-1,j), A(i-1,j-b_i) + p_i\} & \text{falls } i \geq 1, j > b_i \\ \min\{A(i-1,j), p_i\} & \text{falls } i \geq 1, j \leq b_i \end{cases}$$

Der letzte Fall ist bereits durch den vorletzten Fall abgedeckt, da $j \leq b_i$ äquivalent zu $j - b_i \leq 0$ ist und damit durch den dritten Fall $A(i-1, j-b_i) = 0$ gilt. Er wird lediglich der Einfachkeit halber separat aufgeführt. Die minimalen Kosten für 50 Punkte werden durch $A(n, 50)$ gegeben.

b) Auf der Basis der rekursiven Definition ergibt sich das folgende dynamische Programm zur Berechnung der minimalen Kosten:

```
Punktesammlung(Array B, Array P):
1  n ← length(B)
2  A ← new Array [0..n, 0..50]
3  for i ← 0 to n do
4      A[i, 0] ← 0
5  for j ← 1 to 50 do
6      A[0, j] ← ∞
7  for i ← 1 to n do
8      for j ← 1 to 50 do
9          if j ≥ B[i] then
10             A[i, j] ← min{A[i − 1, j], A[i − 1, j − B[i]] + P[i]}
11         else
12             A[i, j] ← min{A[i − 1, j], P[i]}
13 return A[n, 50]
```

c) Die Laufzeit wird dominiert durch die Allokation des Arrays A (Zeile 2) und die for-Schleifen der Zeilen 3 und 7, welche jeweils eine Laufzeit von $\mathcal{O}(n)$ fordern. Hier ist zu beachten, dass die 50 als Mindestzahl benötigter Punkte eine Konstante ist. Eine präzise Laufzeitanalyse ergibt für die Zeile 2 eine Laufzeit von $51 \cdot (n+1)$ und für die verschachtelte for-Schleife in Zeile 7 eine Laufzeit von $n+1+n \cdot (51 + 50 \cdot (2 \cdot 50)) \in \mathcal{O}(n)$. Alle weiteren Zeilen, auch die for-Schleife in Zeile 5, tragen eine konstante Laufzeit ($\mathcal{O}(1)$) bei, die Laufzeit des Algorithmus ist also schlussendlich mit $\mathcal{O}(n)$ abzuschätzen.

d) Zu zeigen ist: $A[i, j]$ enthält die minimalen Kosten, um mindestens j Punkte zu sammeln, wenn nur die Produkte $1..i$ zur Auswahl stehen, für alle $0 \leq i \leq n$ und alle $j \in \mathbb{Z}$. Wir führen ein Induktionsbeweis nach i.

I.A. Wenn $i = 0$ ist, haben wir keine Produkte zur Auswahl. Sind $j \leq 0$ Punkte zu sammeln, müssen keine Produkte gekauft werden, der minimale Preis beträgt also 0. Wenn Alice $j > 0$ Punkte braucht, gibt es keine Lösung. Die minimalen Kosten sind dann unendlich groß, d.h. $A(i, j) = \infty$. Die Behauptung gilt somit für $i = 0$.

I.V. Sei $0 < i \leq n$. Dann ist für alle $j \in \mathbb{Z}$ und alle $0 \leq i' \leq i$ der minimale Preis, um j Punkte mit einer Auswahl aus den ersten i' Produkten zu sammeln, durch $A(i', j)$ gegeben.

I.S. Sei $0 < i \leq n$. Für $j \leq 0$ gilt wieder $A(i, j) = 0$, was korrekt ist, wenn keine Punkte zu sammeln sind. Sei jetzt $j > 0$. Wir nehmen an, dass $A(i, j)$ nicht die minimalen Kosten sind, um j Punkte zu sammeln, wenn die Produkte $1..i$ zur Verfügung stehen. Dann gibt es einen kleineren Wert $v < A(i, j)$, sodass man j Punkte mit Kosten v erzielen kann, wenn die Produkte $1..i$ zur Auswahl stehen. Wir machen eine Fallunterscheidung danach, ob diese bessere Lösung das Produkt i kauft oder nicht.

Wenn diese bessere Lösung das Produkt i nicht auswählt, erzielt sie mit einer Auswahl aus den ersten $i-1$ Produkten j Punkte mit Kosten v. Es gilt $v < A(i, j) = \min\{A(i-1, j), A(i-1, j-b_i)+p_i\} \leq A(i-1, j)$, ein Widerspruch zu unserer I.V., die

$A(i-1,j)$ als minimale Kosten angibt, um j Punkte mit den ersten $i-1$ Produkten zu erlangen.

Wählt diese besser Lösung das Produkt i, erziehlt sie ohne dieses Produkt, also mit einer Auswahl aus den ersten $i-1$ Produkten, immernoch $j-b_i$ Punkte mit Kosten $v-p_i$. Es gilt $v < A(i,j) = \min\{A(i-1,j), A(i-1,j-b_i)+p_i\} \leq A(i-1,j-b_i)+p_i$, eine Umformung ergibt $v - p_i < A(i-1,j-b_i)$, ein Widerspruch zur I.V. mit $A(i-1,j-b_i)$ als optimale Kosten für den Erwerb von $j-b_i$ Punkten mit den ersten $i-1$ Produkten.

Die Behauptung ist nun gezeigt für alle $1 \leq i \leq n$ und für alle $j \in \mathbb{Z}$. □