DAP2 UB8
Anja Rey, Gr.23 , Briefkasten 22

Max Springenberg, 177792

June 12, 2017

8.1 Dynamische Programmierung

gegeben:
n Uebungsblaetter
$1 \leq i \leq n$
Liste mit Bepunktungen $A \in \mathbb{N}^n_{\geq 0}$

8.1.1

B mit $[b_0 ... b_n]$
Existieren keine Uebungsblaetter, so koennen auch keine Punkte vergeben werden, daher $B[0] = 0$
Existiert nur ein Uebungsblatt, so ist dessen Punktzahl auch die maximale Punktzahl, daher $B[1] = A[1]$
Ansonsten muss zwischen der maximalen Punktzahl bis zum vorherigen Uebungsblattes und der maximalen Punktzahl bis zum Uebungsblatt davor plus der Punktzahl fuer das aktuelle Uebungsblatt selbst abgewogen werden.

$$B[i] = \begin{cases} 0 & ,(i = 0) \\ A[i] & ,(i = 1) \\ max\{B[i-1], A[i] + B[i-2]\} & ,sonst \end{cases}$$

8.1.2

```
Uebungsblaetter(A):
1    n ←length(A)
2    B ←new Array[0..n]
3    B[0] ←0
4    B[1] ←A[1]
5    for i ←2 to n do
6        B[i] ←maxB[i-1], A[i] + B[i-2]
7    return B[n]
```

8.1.3

1-4: Konstante Laufzeit $\Theta(4)$
5-6: $\Theta(1 + n * 1)$
7: Konstante Laufzeit $\Theta(1)$
Insgesamt:
$\Theta(4 + 1 + n * 1 + 1) = \Theta(n + 5)$ damit in $O(n)$

8.1.4

Aussage:
B[i] sei die maximale Punktzahl fuer Alle Blaetter bis A[i].

Induktion ueber i.

I.A.
i = 1:
Da es keine weiteren Blaetter gibt ist B[i] = A[i], damit korrekt.

I.V.
Die Aussage gelte fuer $i' \in \mathbb{N}$ mit $0 < i' < i \leq n$ beliebig, aber fest.

I.S.
fuer i > i': Annahme: m > B[i] sei die maximale Punktzahl fuer alle Blaetter bis A[i]
Das Maximum ist entweder:
(i) die maximale Punktzahl bis zum vorherigen Blatt A[i-1], oder
(ii) die Punktzahl vom aktuellem Blatt A[i] plus der maximalen Punktzahl bis zwei Blaetter zuvor A[i-2]

(i) wird mit i ¿ i-1 nach der I.V. durch B[i-1] ermittelt
(ii) das maximum bis A[i-2] wird nach der I.V. ebenfalls durch B[i-2] ermittelt.
demnach gilt:
max{ maximale Punktzahl(A[i-1]), A[i] + maximale Punktzahl(A[i-2]) } $= max\{B[i-1], A[i] + B[i-2]\} = B[i]$

Demnach kann m nicht groesser B[i] sein und die Aussage ist bestaetigt.

8.2 Dynamische Programmierung

gegeben:
n Bruecken
$i, n \in \mathbb{N}, 1 \leq i \leq n$
$A, B \in \mathbb{N}_{\geq 0}^{n-1}, C \in \mathbb{N}_{\geq 0}^{n}$
$C[1] = C[n] = 0$

8.2.1

$\forall i \leq 0$ gilt: es wurden noch keine Brezeln gesammelt oder abgegeben.
$\forall i > 1$ gilt: es koennen bereits Brezeln gesammelt und wieder abgegben wurden sein. Damit muss der erwerbt von $A[i], B[i]$ mit dem jeweiligen Fall einer Abgabe in $C[i+1]$ abgewogen werden und der maximale Wert genommen werden.

Fuer p(i), dass die maximale Brezeln-Anzahl in Buda angibt und q(i), dass analog fuer Pest funktioniert ergeben sich somit die Rekursionsgleichungen:

$$p(i) = \begin{cases} 0 & , (i \leq 1) \\ max\{A[i] + p(i-1), B[i] + q(i-1) - C[i+1]\} & , sonst \end{cases}$$

$$q(i) = \begin{cases} 0 & , (i \leq 1) \\ max\{A[i] + p(i-1) - C[i+1], B[i] + q(i-1)\} & , sonst \end{cases}$$

8.2.2

```
Bruecken(A, B, C):
    n ←minlength(a), length(B)
    Buda ←new Array[1..n]
    Pest ←new Array[1..n]
    Pest[0] ←0
    Buda[0] ←0
    for i ←2 to n do
        Buda[i] ←max{
            A[i] + Buda[i − 1],
            B[i] + Pest[i − 1] − C[i + 1]
        }
        Pest[i] ←max{
            A[i] + Buda[i − 1] − C[i + 1],
            B[i] + Pest[i − 1]
        }
    return Buda[n]
```

8.2.3

1-5: Konstante Laufzeit $\Theta(5)$
6-8: $\Theta(1 + n * 2)$
9: Konstante Laufzeit $\Theta(1)$

Insgesamt damit:
$\Theta(5 + 1 + n * 2 + 1) = \Theta(2 * n + 7)$ damit in $O(n)$

8.2.4

Aussage:
p(i) berechnet die maximale Anzahl an Brezeln in Buda und q(i) berechnet die maximale Anzahl an Brezeln in Pest.
Induktion ueber i.

I.A.
i = 1:
Zu Beginn gibt es noch keine Brezeln und es wurden auch noch keine abgenommen, eine Brueckenueberfuehrung ist kostenlos, daher ist das Ergebnis fuer alle Faelle 0 = q(1) = p(1).

I.V.
Die Aussage gelte fuer $i' \in \mathbb{N}$ mit $0 < i' < i \leq n$ beliebig, aber fest.

I.S.
fuer i = i' + 1:
1.
p(i) muss fuer i > 1 ein Maximum ermitteln:
es kann entweder der Weg vom letzten Maximum p(i-1) ueber A[i] mit:
(p(i-1) + A[i])

oder der Weg vom letzen Maximum des anderen Ufers q(i-1) ueber B[i] unter Subtraktion des Zolles der Bruecke C[i+1] mit:
(q(i-1) + B[i] - C[i+1])

zum Maximum fhren.
Dabei sind q(i-1) und p(i-1) nach I.V. korrekt.
demnach liefert:
max{(p(i-1) + A[i]), (q(i-1) + B[i] - C[i+1]}
offensichtlich das Maximum fuer p(i)

Analog fuer q(i):
q(i) muss fuer i > 1 ein Maximum ermitteln:
es kann entweder der Weg vom letzten Maximum q(i-1) ueber B[i] mit:

(q(i-1) + B[i])

oder der Weg vom letzen Maximum des anderen Ufers p(i-1) ueber A[i] unter Subtraktion des Zolles der Bruecke C[i+1] mit:
(p(i-1) + A[i] - C[i+1])

zum Maximum fhren.
Dabei sind q(i-1) und p(i-1) nach I.V. korrekt.
demnach liefert:
max{(q(i-1) + B[i]), (p(i-1) + A[i] - C[i+1]}
offensichtlich das Maximum fuer p(i)

Damit ist die Aussage bestaetigt.