

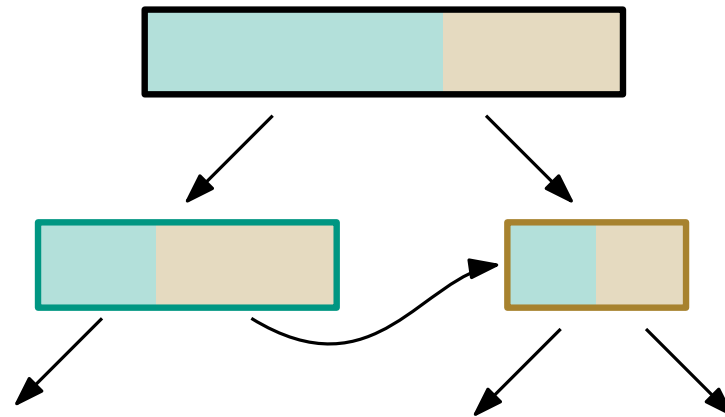
Algorithmen 1

Übung 7 Dynamische Programmierung und Amortisierte Analyse



Dynamische Programmierung

- Methode zum Lösen von Optimierungsproblemen
- optimale Lösung \leftarrow optimale Teillösung + optimale Teillösung
- bestimme Rekurrenz
- verwende optimale Teillösungen wieder



Paletten-Hopping

Problemstellung

- Biber hüpfen gerne auf Palettentürmen
- Palettentürme haben eine unterschiedliche Höhe
- Hopping: nach \rightarrow zu benachbarter Palette
- Biber-Hopping:
Höhendifferenz zwischen Hops ≤ 1
- Maximales Biber-Hopping?

Formal (2D):

- n Zeilen, n Spalten
- Höhe $h(i, j) \in \mathbb{N}$
- Hops $(i, j) \rightarrow (i + 1, j)$ und $(i, j) \rightarrow (i, j + 1)$
- maximiere Länge Biber-Hopping $(b_1, \dots, b_\ell): |h(b_k) - h(b_{k+1})| \leq 1$

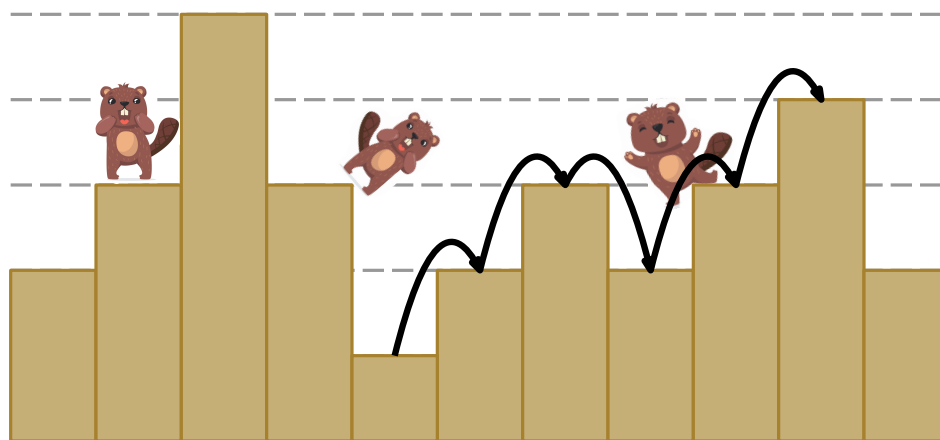
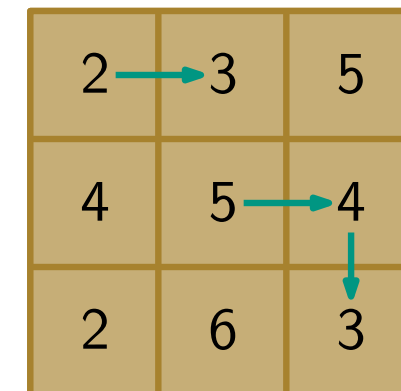


Bild: Funny Beaver Set von pch.vector, Freepik

2D \rightarrow



Paletten-Hopping – Teilprobleme

2	3	5	4	2
4	5	4	5	1
2	6	3	2	3
4	3	2	1	6
3	4	3	2	5

Teilprobleme

- $B(i, j) := \text{max. Länge Biber-Hopping mit Ende } (i, j)$

Welche Infos sind nötig, um $B(i, j)$ zu bestimmen?

Formal (2D):

- n Zeilen, n Spalten
- Höhe $h(i, j) \in \mathbb{N}$
- Hops $(i, j) \rightarrow (i + 1, j)$ und $(i, j) \rightarrow (i, j + 1)$
- maximiere Länge Biber-Hopping
 $(b_1, \dots, b_\ell): |h(b_k) - h(b_{k+1})| \leq 1$

Paletten-Hopping – Teilprobleme

2	3	5	4	2
4	5	4	5	1
2	6	3	2	3
4	3	2	1	6
3	4	3	2	5

Formal (2D):

- n Zeilen, n Spalten
- Höhe $h(i, j) \in \mathbb{N}$
- Hops $(i, j) \rightarrow (i + 1, j)$ und $(i, j) \rightarrow (i, j + 1)$
- maximiere Länge Biber-Hopping
 $(b_1, \dots, b_\ell): |h(b_k) - h(b_{k+1})| \leq 1$

Teilprobleme

- $B(i, j) := \max.$ Länge Biber-Hopping mit Ende (i, j)
- drei Möglichkeiten:
 - von oben erweiterbar
 - von links erweiterbar
 - nicht erweiterbar

$$B(i, j) = B(i - 1, j) + 1$$

$$B(i, j) = B(i, j - 1) + 1$$

$$B(i, j) = 1$$

Rekurrenz

$$B(i, 0) := 0 \text{ und } B(0, j) := 0$$

$$B(i, j) = \max \begin{cases} B(i - 1, j) + 1, \text{ falls } |h(i - 1, j) - h(i, j)| \leq 1 \\ B(i, j - 1) + 1, \text{ falls } |h(i, j) - h(i, j - 1)| \leq 1 \\ 1 \end{cases}$$

Laufzeit von $B(i, j)$ bei rekursiver Implementierung?

■ $O(n)$

■ $O(n^2)$

■ $O(2^n)$

■ $O(n!)$

Paletten-Hopping – Teilprobleme

2	3	5	4	2
4	5	4	5	1
2	6	3	2	3
4	3	2	1	6
3	4	3	2	5

Formal (2D):

- n Zeilen, n Spalten
- Höhe $h(i, j) \in \mathbb{N}$
- Hops $(i, j) \rightarrow (i + 1, j)$ und $(i, j) \rightarrow (i, j + 1)$
- maximiere Länge Biber-Hopping
 $(b_1, \dots, b_\ell): |h(b_k) - h(b_{k+1})| \leq 1$

Teilprobleme

- $B(i, j) := \max.$ Länge Biber-Hopping mit Ende (i, j)
- drei Möglichkeiten:
 - von oben erweiterbar
 - von links erweiterbar
 - nicht erweiterbar

$$B(i, j) = B(i - 1, j) + 1$$

$$B(i, j) = B(i, j - 1) + 1$$

$$B(i, j) = 1$$

Rekurrenz

$$B(i, 0) := 0 \text{ und } B(0, j) := 0$$

$$B(i, j) = \max \begin{cases} B(i - 1, j) + 1, \text{ falls } |h(i - 1, j) - h(i, j)| \leq 1 \\ B(i, j - 1) + 1, \text{ falls } |h(i, j) - h(i, j - 1)| \leq 1 \\ 1 \end{cases}$$

- Werte zwischenspeichern

Iterative Implementierung

2	3	5	4	2
4	5	4	5	1
2	6	3	2	3
4	3	2	1	6
3	4	3	2	5

Rekurrenz

$B(i, 0) := 0$ und $B(0, j) := 0$

$$B(i, j) = \max \begin{cases} B(i-1, j) + 1, & \text{falls } |h(i-1, j) - h(i, j)| \leq 1 \\ B(i, j-1) + 1, & \text{falls } |h(i, j) - h(i, j-1)| \leq 1 \\ 1 \end{cases}$$

Iterative Implementierung

2	3	5	4	2
4	5	4	5	1
2	6	3	2	3
4	3	2	1	6
3	4	3	2	5

~~Rekurrenz~~ Iterativ

$B[i][0] := 0$ und $B[0][j] := 0$

$$B[i][j] = \max \begin{cases} B[i-1][j] + 1, & \text{falls } |h(i-1, j) - h(i, j)| \leq 1 \\ B[i][j-1] + 1, & \text{falls } |h(i, j) - h(i, j-1)| \leq 1 \\ 1 \end{cases}$$

Iterative Implementierung

2	3	5	4	2
4	5	4	5	1
2	6	3	2	3
4	3	2	1	6
3	4	3	2	5

0	0	0	0	0	0
0					
0					
0					
0					
0					

~~Rekurrenz~~ Iterativ

$B[i][0] := 0$ und $B[0][j] := 0$

$$B[i][j] = \max \begin{cases} B[i-1][j] + 1, & \text{falls } |h(i-1, j) - h(i, j)| \leq 1 \\ B[i][j-1] + 1, & \text{falls } |h(i, j) - h(i, j-1)| \leq 1 \\ 1 \end{cases}$$

Iterative Implementierung

2	3	5	4	2
4	5	4	5	1
2	6	3	2	3
4	3	2	1	6
3	4	3	2	5

0	0	0	0	0	0
0	1	2			
0					
0					
0					
0					

~~Rekurrenz~~ Iterativ

$B[i][0] := 0$ und $B[0][j] := 0$

$$B[i][j] = \max \begin{cases} B[i-1][j] + 1, & \text{falls } |h(i-1, j) - h(i, j)| \leq 1 \\ B[i][j-1] + 1, & \text{falls } |h(i, j) - h(i, j-1)| \leq 1 \\ 1 \end{cases}$$

Iterative Implementierung

2	3	5	4	2
4	5	4	5	1
2	6	3	2	3
4	3	2	1	6
3	4	3	2	5

0	0	0	0	0	0
0	1	2	1	2	1
0	1	2	3		
0					
0					
0					

~~Rekurrenz~~ Iterativ

$B[i][0] := 0$ und $B[0][j] := 0$

$$B[i][j] = \max \begin{cases} B[i-1][j] + 1, & \text{falls } |h(i-1, j) - h(i, j)| \leq 1 \\ B[i][j-1] + 1, & \text{falls } |h(i, j) - h(i, j-1)| \leq 1 \\ 1 \end{cases}$$

Iterative Implementierung

2	3	5	4	2
4	5	4	5	1
2	6	3	2	3
4	3	2	1	6
3	4	3	2	5

0	0	0	0	0	0
0	1	2	1	2	1
0	1	2	3	4	2
0	1	3	4	5	6
0	1	2	5	6	1
0	2	3	6	7	2

~~Rekurrenz~~ Iterativ

$B[i][0] := 0$ und $B[0][j] := 0$

$$B[i][j] = \max \begin{cases} B[i-1][j] + 1, \text{ falls } |h(i-1, j) - h(i, j)| \leq 1 \\ B[i][j-1] + 1, \text{ falls } |h(i, j) - h(i, j-1)| \leq 1 \\ 1 \end{cases}$$

Iterative Implementierung

2	3	5	4	2
4	5	4	5	1
2	6	3	2	3
4	3	2	1	6
3	4	3	2	5

0	0	0	0	0	0
0	1	2	1	2	1
0	1	2	3	4	2
0	1	3	4	5	6
0	1	2	5	6	1
0	2	3	6	7	2

Laufzeit von $B(i, j)$ bei iterativer Implementierung?

■ $O(n)$

■ $O(n^2)$

■ $O(2^n)$

■ $O(n!)$

Iterative Implementierung

2	3	5	4	2
4	5	4	5	1
2	6	3	2	3
4	3	2	1	6
3	4	3	2	5

0	0	0	0	0	0
0	1	2	1	2	1
0	1	2	3	4	2
0	1	3	4	5	6
0	1	2	5	6	1
0	2	3	6	7	2

Was ist die maximale Länge eines Biber-Hoppings?

Wie rekonstruiert man eine Lösung?

Rekonstruktion der Lösung

2	3	5	4	2
4	5	4	5	1
2	6	3	2	3
4	3	2	1	6
3	4	3	2	5

A red arrow points up from the cell containing '3' in the first row, second column. A green arrow points left from the same cell to the cell containing '2' in the first row, first column.

0	0	0	0	0	0
0	1	2			
0					
0					
0					
0					

A red arrow points up from the cell containing '2' in the second row, third column. A green arrow points left from the same cell to the cell containing '1' in the second row, second column.

- speichere den passenden Vorgänger

Rekonstruktion der Lösung

2	3	5	4	2
4	5	4	5	1
2	6	3	2	3
4	3	2	1	6
3	4	3	2	5

0	0	0	0	0	0
0	1 ← 2	1 ← 2			1
0	1 ← 2 ← 3				
0					
0					
0					

- speichere den passenden Vorgänger

Rekonstruktion der Lösung

2	3	5	4	2
4	5	4	5	1
2	6	3	2	3
4	3	2	1	6
3	4	3	2	5

0	0	0	0	0	0
0	1 ← 2		1 ← 2		1 ↑
0	1 ← 2 ← 3 ← 4				2 ↑
0	1	3 ↑	4 ↑	5 ← 6	
0	1 ← 2		5 ↑	6 ↑	1 ↑
0	2 ↑	3 ↑	6 ↑	7 ↑	2 ↑

- speichere den passenden Vorgänger

Rekonstruktion der Lösung

2	3	5	4	2
4	→ 5	→ 4	5	1
2	6	↓ 3	→ 2	3
4	3	2	↓ 1	6
3	4	3	↓ 2	5

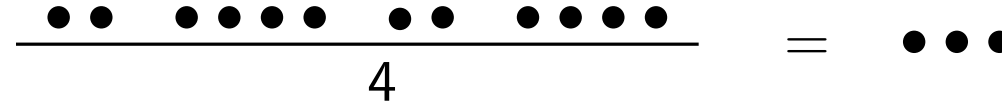
0	0	0	0	0	0
0	1 ← 2	1 ← 2	1	2	1
0	1 ← 2 ← 3 ← 4	2	3	4	2
0	1	3	4 ← 5 ← 6	5	6
0	1 ← 2	5	6	1	1
0	2	3	6	7	2

- speichere den passenden Vorgänger
- verfolge Zeiger zurück

Amortisierte Analyse

Aggregatmethode

- berechne Gesamtkosten
- teile durch Anzahl von Operationen



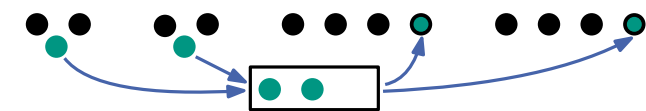
Charging

- verteile Kosten-Tokens von teuren zu günstigen Ops



Kontomethode

- günstige Op zahlt mehr ins Konto ein, teure Op hebt ab
- Achtung: Kontostand ≥ 0



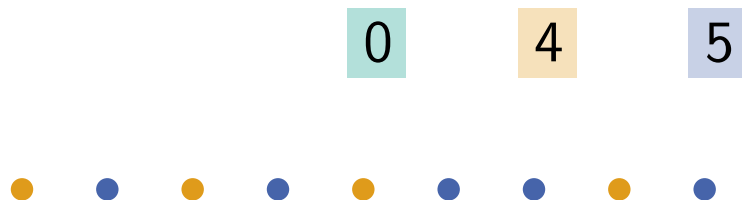
Potentialmethode

- Potential $\Phi(A) \geq 0$ abhängig vom Zustand von A
- amortisierte Kosten = tatsächliche Kosten + $\Phi(A_{\text{nach}}) - \Phi(A_{\text{vor}})$

Amortisierte Analyse

Aufgabe

- Kugeln (drei Farben) in einer Reihe
- Operationen
 - `insert(pos, color)` in $\Theta(1)$
 - `removeMost()`: Lösche eine Farbe mit den meisten Kugeln in $\Theta(n)$
- Einsicht: immer $\geq \frac{n}{3}$ gelöscht



Aggregatmethode

- Gesamtkosten / Anzahl Ops

Charging

- Kostentoken: teure Op \rightarrow günstige Op

Kontomethode

- günstige Op \rightarrow Konto, Konto \rightarrow teure Op

Potentialmethode

- Potential $\Phi(A) \geq 0$ abhängig vom Zustand von A
- amort. Kosten = tats. Kosten + $\Phi(A_{\text{nach}}) - \Phi(A_{\text{vor}})$

Bonusblatt

- besteht nur aus Bonuspunkten
- ab 14 Uhr auf der Webseite
- Aufgaben zu vielen verschiedenen Themen
- Abgabe normal im ILIAS
- nur garantierte Korrektur zwischen 110 und 130 Punkten
- trotzdem gute Übung für die Klausur!