Dynamisches Programmieren

Franz Scheffler, Pascal Otto

15.01.2015

Inhaltsverzeichnis

Allgemeines

Was versteht man unter dynamischen Programmieren?

Allgemeines

Was versteht man unter dynamischen Programmieren?

- Teile und Herrsche mit Optimalitätsbedingung

Allgemeines

Was versteht man unter dynamischen Programmieren?

- Teile und Herrsche mit Optimalitätsbedingung
- Optimale Gesamtlösung setzt sich aus optimalen Lösungen der Teilprobleme zusammen

Allgemeines

Umformungsprobleme

Allgemeines

Umformungsprobleme

1. Unabhängige überschneidungsfreie Teilprobleme sind selten erreichbar

Allgemeines

Umformungsprobleme

1. Unabhängige überschneidungsfreie Teilprobleme sind selten erreichbar
2. Ungleichmäßig gewichtete Teilprobleme

Allgemeines

Umformungsprobleme

1. Unabhängige überschneidungsfreie Teilprobleme sind selten erreichbar
2. Ungleichmäßig gewichtete Teilprobleme
3. Zahl der zu lösenden Teilprobleme kann unvertretbar hoch sein

Allgemeines

Umformungsprobleme

1. Unabhängige überschneidungsfreie Teilprobleme sind selten erreichbar
2. Ungleichmäßig gewichtete Teilprobleme
3. Zahl der zu lösenden Teilprobleme kann unvertretbar hoch sein
4. Optimalitätsbedingung wird nicht erfüllt

Allgemeines

Wie findet man die optimale Lösung für ein Teilproblem?

Allgemeines

Wie findet man die optimale Lösung für ein Teilproblem?

- Man kennt alle Lösungen für ein Problem und sucht dann die Beste heraus

Allgemeines

Wie findet man die optimale Lösung für ein Teilproblem?

- Man kennt alle Lösungen für ein Problem und sucht dann die Beste heraus
- Man erhält Lösungen indem man sie berechnet

Allgemeines

Wie findet man die optimale Lösung für ein Teilproblem?

- Man kennt alle Lösungen für ein Problem und sucht dann die Beste heraus
- Man erhält Lösungen indem man sie berechnet
- Berechnung erfolgt ursprünglich iterativ

Allgemeines

Wie findet man die optimale Lösung für ein Teilproblem?

- Man kennt alle Lösungen für ein Problem und sucht dann die Beste heraus
- Man erhält Lösungen indem man sie berechnet
- Berechnung erfolgt ursprünglich iterativ
- Berechnete Lösungen trägt man in eine Tabelle ein

Allgemeines

M	1	2	3	4	5	6
1	5	3	4	6	8	9
2	1	3	3	2	4	5
3	0	9	$\longrightarrow$			

Allgemeines

M	1	2	3	4	5	6
1	5	3	4	6	8	9
2	1	3	3	2	4	5
3	0	9	7	2	1	7

Allgemeines

Wie findet man die optimale Lösung für ein Teilproblem?

- iterative Berechnungen liegen meist als Top-Down-Ansatz vor

Allgemeines

Wie findet man die optimale Lösung für ein Teilproblem?

- iterative Berechnungen liegen meist als Top-Down-Ansatz vor
- Diese müssen in Bottom-Up-Verfahren umgewandelt werden

Allgemeines

Wie findet man die optimale Lösung für ein Teilproblem?

- iterative Berechnungen liegen meist als Top-Down-Ansatz vor
- Diese müssen in Bottom-Up-Verfahren umgewandelt werden
- geschieht meist über in einander verschachtelte Schleifen, die die Tabelle repräsentieren

Allgemeines

Wie findet man die optimale Lösung für ein Teilproblem?

- iterative Berechnungen liegen meist als Top-Down-Ansatz vor
- Diese müssen in Bottom-Up-Verfahren umgewandelt werden
- geschieht meist über in einander verschachtelte Schleifen, die die Tabelle repräsentieren
- erfordert mentalen Aufwand

Allgemeines

Wie findet man die optimale Lösung für ein Teilproblem?

- iterative Berechnungen liegen meist als Top-Down-Ansatz vor
- Diese müssen in Bottom-Up-Verfahren umgewandelt werden
- geschieht meist über in einander verschachtelte Schleifen, die die Tabelle repräsentieren
- erfordert mentalen Aufwand
- fehleranfällig

Allgemeines

- rekursive Teilprobleme

Allgemeines

- rekursive Teilprobleme
- Mehrfachberechnungen beim Erstellen der Tabelle

Allgemeines

- rekursive Teilprobleme
- Mehrfachberechnungen beim Erstellen der Tabelle

$\Rightarrow$ Optimierbar mit Memoizing

0/1 - Rucksackproblem

Überlegung

- Gegenstand i wird in optimalen Rucksack gelegt

0/1 - Rucksackproblem

Überlegung

- Gegenstand i wird in optimalen Rucksack gelegt
- Garantiert optimalen Rucksack mit Gegenstand i

0/1 - Rucksackproblem

Überlegung

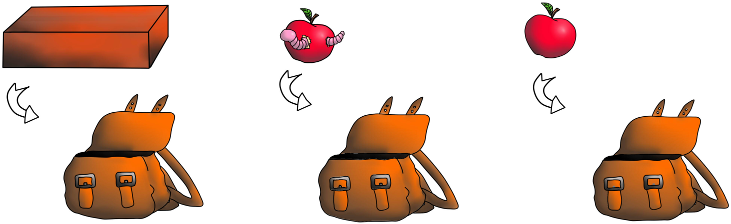
- Gegenstand i wird in optimalen Rucksack gelegt
- Garantiert optimalen Rucksack mit Gegenstand i
- Zur Auswahl stehende Gegenstände werden in bis dahin optimal gefüllten Rucksack gelegt

$\surd$ Optimalitätsbedingung erfüllt

0/1 - Rucksackproblem

Für den Gesamtwert $wert(n, K)$ des Rucksacks gilt:

$$wert(i, g) = \begin{cases} 0, wenn \ \ i = 0 \\ wert(i-1, g), wenn \ \ i > 0 \ \ und \ \ g - g_i < 0 \\ max(wert(i-1, g), wert(i-1, g-g_i) + w_i) \end{cases}$$



Traditionelle iterative Berechnung

- rekursive wert-Funktion kann nicht übernommen werden

Traditionelle iterative Berechnung

- rekursive wert-Funktion kann nicht übernommen werden
- aber die Überlegung die dazu geführt hat

Traditionelle iterative Berechnung

i	1	2	3	4
g_i	2	6	4	5
w_i	6	5	6	4

n = 4
k = 8

M	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0								
2	0								
3	0								
4	0								

- Matrix(Tabelle) vom Typ (z, s)
- z $\rightarrow$ n + 1 Zeilen für jeden Gegenstand
- s $\rightarrow$ K Spalten für die einzelnen Gewichtssummen

Traditionelle iterative Berechnung

i	1	2	3	4
g_i	2	6	4	5
w_i	6	5	6	4

n = 4
k = 8

M	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0							
2	0								
3	0								
4	0								

- wenn $s < g_i$, dann ZeileAlt[s]

Traditionelle iterative Berechnung

i	1	2	3	4
g_i	2	6	4	5
w_i	6	5	6	4

n = 4
k = 8

M	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	6						
2	0								
3	0								
4	0								

- sonst max(ZeileAlt[s], ZeileAlt[$s - g_i$] + w_i)

Traditionelle iterative Berechnung

i	1	2	3	4
g_i	2	6	4	5
w_i	6	5	6	4

n = 4
k = 8

M	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	6	6	6	6	6	6	6
2	0								
3	0								
4	0								

- wenn $s < g_i$, dann ZeileAlt[s]
- sonst max(ZeileAlt[s], ZeileAlt[$s - g_i$] + w_i)

Traditionelle iterative Berechnung

i	1	2	3	4
g_i	2	6	4	5
w_i	6	5	6	4

n = 4
k = 8

M	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	6	6	6	6	6	6	6
2	0	0	6	6	6	6	6	6	11
3	0	0	6	6	6	6	12	12	12
4	0	0	6	6	6	6	12	12	$\boxed{12}$

Aufwandsbetrachtung

- Durch iterative Abarbeitung Aufwand von $\mathcal{O}(n \cdot K)$

Aufwandsbetrachtung

- Durch iterative Abarbeitung Aufwand von $\mathcal{O}(n \cdot K)$
- Sieht man K als Konstante ergibt sich ein scheinbar linearer Aufwand von $\mathcal{O}(n)$

Aufwandsbetrachtung

- Durch iterative Abarbeitung Aufwand von $\mathcal{O}(n \cdot K)$
- Sieht man K als Konstante ergibt sich ein scheinbar linearer Aufwand von $\mathcal{O}(n)$
- Ist nicht der Fall da n und K zusammenhängen

Aufwandsbetrachtung

- Durch iterative Abarbeitung Aufwand von $\mathcal{O}(n \cdot K)$
- Sieht man K als Konstante ergibt sich ein scheinbar linearer Aufwand von $\mathcal{O}(n)$
- Ist nicht der Fall da n und K zusammenhängen
- Begründet dadurch, dass der Algorithmus pseudopolynomial ist

Aufwandsbetrachtung

- Durch iterative Abarbeitung Aufwand von $\mathcal{O}(n \cdot K)$
- Sieht man K als Konstante ergibt sich ein scheinbar linearer Aufwand von $\mathcal{O}(n)$
- Ist nicht der Fall da n und K zusammenhängen
- Begründet dadurch, dass der Algorithmus pseudopolynomial ist
  - Aufwandsbetrachtung nur über Bit-Komplexität, welche die Eingabelänge des Wortes berücksichtigt

Aufwandsbetrachtung

Festellen der Eingabelänge

- Besteht aus $2n + 1$ natürlichen Zahlen

Aufwandsbetrachtung

Festellen der Eingabelänge

- Besteht aus $2n + 1$ natürlichen Zahlen
  - Gewicht-Wert-Paare, Kapazität in Binärdarstellung

Aufwandsbetrachtung

Festellen der Eingabelänge

- Besteht aus $2n + 1$ natürlichen Zahlen
  - Gewicht-Wert-Paare, Kapazität in Binärdarstellung
- Definition von $\widehat{w} = max\{w_1, w_2, ..., w_n\}$ und beschränken der Eingabelänge durch $\mathcal{O}(n \cdot (\log_2 K + \log_2 \widehat{w}))$

Aufwandsbetrachtung

Festellen der Eingabelänge

- Besteht aus $2n + 1$ natürlichen Zahlen
  - Gewicht-Wert-Paare, Kapazität in Binärdarstellung
- Definition von $\widehat{w} = max\{w_1, w_2, ..., w_n\}$ und beschränken der Eingabelänge durch $\mathcal{O}(n \cdot (\log_2 K + \log_2 \widehat{w}))$
- Lässt sich K nun in der Form $K = 2^n$ darstellen und $\widehat{w} \leq 2^n$ so erhält man eine Eingabelänge von $\mathcal{O}(n^2)$

Aufwandsbetrachtung

Festellen der Eingabelänge

- Besteht aus $2n + 1$ natürlichen Zahlen
  - Gewicht-Wert-Paare, Kapazität in Binärdarstellung
- Definition von $\widehat{w} = max\{w_1, w_2, ..., w_n\}$ und beschränken der Eingabelänge durch $\mathcal{O}(n \cdot (\log_2 K + \log_2 \widehat{w}))$
- Lässt sich K nun in der Form $K = 2^n$ darstellen und $\widehat{w} \leq 2^n$ so erhält man eine Eingabelänge von $\mathcal{O}(n^2)$
- Es ergibt sich ein exponentieller Rechenaufwand von $\mathcal{O}(n \cdot 2^n)$

Codebeispiel

Aufwandsbetrachtung

Schlussfolgerung

- Rekursiver Implementation mit Memoizing ist der Vorzug zu geben

Aufwandsbetrachtung

Schlussfolgerung

- Rekursiver Implementation mit Memoizing ist der Vorzug zu geben
- Mindestens so schnell wie iterative Implementation

Aufwandsbetrachtung

Schlussfolgerung

- Rekursiver Implementation mit Memoizing ist der Vorzug zu geben
- Mindestens so schnell wie iterative Implementation
- kann aus rekursiver Vorschrift direkt abgelesen werden

Aufwandsbetrachtung

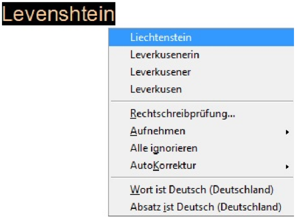
Schlussfolgerung

- Rekursiver Implementation mit Memoizing ist der Vorzug zu geben
- Mindestens so schnell wie iterative Implementation
- kann aus rekursiver Vorschrift direkt abgelesen werden
- keine fehleranfälligen Transformationen notwendig

Levenshtein Distanz / Editierdistanz

Anlass der Betrachtung

- Jede Rechtschreibprüfung arbeitet mit dem Prinzip



- Warum bekommen wir genau diese Vorschläge?

Editierdistanz

Definition

Die Editierdistanz bezeichnet ein Maß für den Unterschied zwischen zwei Zeichenketten. Gemessen wird die Anzahl minimaler Einfüge-, Lösch- und Ersetzungsoperationen von Buchstaben, um zwei Zeichenketten ineinander zu überführen.

- Beispiel

  bello $\Longrightarrow$ wello $\Longrightarrow$ welto $\Longrightarrow$ welt

Buchstabenoperation

Eine Editieroperation hat ihren Preis

- Einfügen - 1
- Löschen - 1
- Ersetzen - 1
- Übereinstimmen - 0

- Suche nach Kombination mit minimalen Kosten

Buchstabenoperation

Obere und unterer Schranke der Distanz

- Mindestens die Längendifferenz

  Bsp. abc $\rightarrow$ abcdef
- Höchsten die Länge der längeren Zeichenkette

  Bsp. xyz $\rightarrow$ abcdef
- nur dann 0 wenn identisch

Ansätze für die Rekursion

- Der letzte Buchstabe wird eingefügt, gelöscht, ersetzt oder übernommen

Ansätze für die Rekursion

- Der letzte Buchstabe wird eingefügt, gelöscht, ersetzt oder übernommen
- Betrachtet man die Zeichenkette ohne den letzten Buchstabe erhält man kleineres Teilproblem

Ansätze für die Rekursion

- Der letzte Buchstabe wird eingefügt, gelöscht, ersetzt oder übernommen
- Betrachtet man die Zeichenkette ohne den letzten Buchstabe erhält man kleineres Teilproblem
- Für dieses wieder optimale Lösung vorhanden

$\surd$ Optimalitätsbedingung erfüllt

Rekursionsgleichung

$$D_{i,j} = min \begin{cases} D_{i-1,j-1} & +0(übereinstimmen) \\ D_{i-1,j-1} & +1(ersetzen) \\ D_{i,j-1} & +1(löschen) \\ D_{i-1,j} & +1(einfügen) \end{cases}$$

$$D_{0,0} = 0$$

$$D_{0,j} = j$$

$$D_{i,0} = i$$

Lösung

- Tabelle mit Memoization
- Größe: $n + 1 \cdot m + 1$

		h	a	l	l	o	w	e	l	t
	0	1	2	3	4	5	6	7	8	9
a	1	1	1	2	3	4	5	6	7	8
l	2	2	2	1	2	3	4	5	6	7
o	3	3	3	2	2	2	3	4	5	6
w	4	4	4	3	3	3	2	3	4	5
z	5	5	5	4	4	4	3	3	4	5
e	6	6	6	5	5	5	4	3	4	5
l	7	7	7	6	5	6	5	4	3	4
t	8	8	8	7	6	6	6	5	4	3

Lösung

		h	a	l	l	o	w	e	l	t
	0	1	2	3	4	5	6	7	8	9
a	1	1	1	2	3	4	5	6	7	8
l	2	2	2	1	2	3	4	5	6	7
o	3	3	3	2	2	2	3	4	5	6
w	4	4	4	3	3	3	2	3	4	5
z	5	5	5	4	4	4	3	3	4	5
e	6	6	6	5	5	5	4	3	4	5
l	7	7	7	6	5	6	5	4	3	4
t	8	8	8	7	6	6	6	5	4	3

$\longrightarrow$ Einfügen $\downarrow$ Löschen $\searrow$ Ersetzen/Beibehalten

Fragen?

Vielen Dank für Ihre Aufmerksamkeit!

Nun ist Zeit für Ihre Fragen.