Grundzüge der Informatik 1

Vorlesung 12

Dynamische Programmierung

Überblick

- Wiederholung
  - Rucksackproblem
  - Entwicklung der Rekursionsgleichung
  - Entwicklung des Algorithmus
  - Finden einer optimalen Lösung
- Dynamische Programmierung auf Zeichenketten
  - Längste gemeinsame Teilfolge
  - Finden einer Rekursionsgleichung
  - Entwicklung des Algorithmus
  - Finden einer optimalen Lösung

Dynamische Programmierung

Dynamische Programmierung für Optimierungsprobleme

1. Bestimme rekursive Struktur einer optimalen Lösung durch Zurückführen auf optimale Teillösungen

2. Entwerfe rekursive Methode zur Bestimmung des *Wertes* einer optimalen Lösung.

3. Transformiere rekursive Methode in eine iterative (bottom-up) Methode zur Bestimmung des Wertes einer optimalen Lösung.

4. Bestimmen aus dem Wert einer optimalen Lösung und in 3. ebenfalls berechneten Zusatzinformationen eine optimale Lösung.

Dynamische Programmierung

Das Rucksackproblem

- Rucksack mit begrenzter Kapazität
- Objekte mit unterschiedlichem Wert und unterschiedlicher Größe
- Wir wollen Objekte von möglichst großem Gesamtwert mitnehmen

Beispiel

- Rucksackgröße 6

Größe	5	2	1	3	7	4
Wert	11	5	2	8	14	9

- Objekt 1 und 3 passen in den Rucksack und haben einen Gesamtwert von 13
- Objekte 2,3 und 4 passen und haben Gesamtwert von 15

Dynamische Programmierung - Rucksackproblem

Lemma 11.3 (Rekursive Struktur einer optimalen Lösung)

- Sei $O \subseteq \{1,..,i\}$ eine optimale Lösung für das Rucksackproblem mit Objekten aus $\{1,…,i\}$ und Rucksackgröße j. Es bezeichne $Opt(i,j)$ den Wert dieser optimalen Lösung. Dann gilt:

(a) Ist Objekt i in O enthalten, so ist $O \setminus \{i\}$ eine optimale Lösung für das Rucksackproblem mit Objekten aus $\{1,…,i-1\}$ und Rucksackgröße $j-g[i]$. Insbesondere gilt $Opt(i,j)=w[i] + Opt(i-1, j-g[i])$.

(b) Ist Objekt i nicht in O enthalten, so ist O eine optimale Lösung für das Rucksackproblem mit Objekten aus $\{1,…,i-1\}$ und Rucksackgröße j. Insbesondere gilt $Opt(i,j) = Opt(i-1, j)$.

Dynamische Programmierung - Rucksackproblem

Korollar 11.4 (Rekursion zu den Kosten einer opt. Lösung)

- Es gilt

(a) $Opt(1,j) = w[1]$ für $j \geq g[1]$

(b) $Opt(1,j) = 0$ für $j < g[1]$

(c) $Opt(i,j) = \max\{Opt(i-1,j), w[i] + Opt(i-1,j-g[i])\}$, falls $i>1$ und $g[i] \leq j$, und

(d) $Opt(i,j) = Opt(i-1,j)$, falls $i>1$ und $g[i]>j$.

Universität zu Köln

Dynamische Programmierung - Rucksackproblem

Rucksack(n,g,w,G)

```
Opt = new array [1,..,n][0,..,G]
for j = 0 to G do                          \* Rekursionsabbruch
    if j<g[1] then Opt[1,j]  = 0
    else Opt[1,j] = w[1]
for i = 2 to n do
    for j = 0 to G do
        if g[i] ≤j then Opt[i,j] = max{Opt[i-1,j], w[i] + Opt[i-1,j-g[i]]}
    else Opt[i,j] = Opt[i-1,j]
return Opt[n,G]
```

Laufzeit

- $O(nG)$

Dynamische Programmierung - Rucksackproblem

Beobachtung

- Sei R der Wert einer optimalen Lösung für Objekte aus {1,..,i}
- Falls g[i]≤j und Opt(i-1,j-g[i]) +w[i]= R, so ist Objekt i in mindestens einer optimalen Lösung enthalten

Dynamische Programmierung
- Rucksackproblem

RucksackLösung(Opt,g,w,i,j)

1. **if** i=0 **return** ∅
2. **else if** g[i]>j **then return** RucksackLösung(Opt,g,w,i-1,j)
3. **else if** Opt[i,j]=w[i] + Opt[i-1,j-g[i]] **then**
   return {i} ∪ RucksackLösung(Opt,g,w,i-1,j-g[i])
4. **else return** RucksackLösung(Opt,g,w,i-1,j)

Aufruf

- Nach der Berechnung der Tabelle Opt von Rucksack wird RucksackLösung mit Opt, g,w, i=n und j=G aufgerufen.

Dynamische Programmierung
- Rucksackproblem

Satz 11.6

Mit Hilfe der Algorithmen Rucksack und RucksackLösung kann man in $O(nG)$ Zeit eine optimale Lösung für das Rucksackproblem berechnen, wobei n die Anzahl der Objekte ist und G die Größe des Rucksacks.

.

Dynamische Programmierung
- Längste gemeinsame Teilfolge

Ziel

- Nutzen von dynamischer Programmierung für Probleme auf Strings

Ansatz

- Rekursive Beschreibung einer optimalen Lösung
- Rekursion für den Wert einer optimalen Lösung (hier: Länge eines Strings)
- Entwicklung des Algorithmus
- Herleiten des Lösungsstrings aus der Tabelle des dynamischen Programms

Dynamische Programmierung
- Längste gemeinsame Teilfolge

Definition 12.1 (Teilfolge)

- Ein *Alphabet* bezeichnet eine endliche Menge von Zeichen
- Seien $X=(x_1,...,x_m)$ und $Y=(y_1,...,y_n)$ zwei Folgen, wobei $x_i, y_j \in A$ für ein endliches Alphabet A.
- Y heißt *Teilfolge* von X, wenn es aufsteigend sortierte Indizes $i_1,...,i_n$ gibt mit $x_{i_j} = y_j$ für $j = 1,...,n$.

Dynamische Programmierung
- Längste gemeinsame Teilfolge

Definition 12.1 (Teilfolge)

- Ein *Alphabet* bezeichnet eine endliche Menge von Zeichen
- Seien $X=(x_1,...,x_m)$ und $Y=(y_1,...,y_n)$ zwei Folgen, wobei x_i, $y_j \in A$ für ein endliches Alphabet A.
- Y heißt *Teilfolge* von X, wenn es aufsteigend sortierte Indizes $i_1,...,i_n$ gibt mit $x_{i_j} = y_j$ für $j = 1,...,n$.

Beispiel

- Folge Y: BCAC
- Folge X: ABACABC
- Y ist Teilfolge von X (Wähle $i_1=2$, $i_2=4$, $i_3=5$ und $i_4=7$)

Dynamische Programmierung - Längste gemeinsame Teilfolge

Definition 12.2 (Gemeinsame Teilfolge)

- Seien X, Y, Z Folgen über A.
- Dann heißt Z *gemeinsame Teilfolge* von X und Y, wenn Z Teilfolge sowohl von X als auch von Y ist.

Beispiel

- Z = BCAC
- X = ABACABC
- Y = BACCABBC

Dynamische Programmierung
- Längste gemeinsame Teilfolge

Definition 12.2 (Gemeinsame Teilfolge)

- Seien X, Y, Z Folgen über A.
- Dann heißt Z *gemeinsame Teilfolge* von X und Y, wenn Z Teilfolge sowohl von X als auch von Y ist.

Beispiel

- Z = BCAC
- X = ABACABC
- Y = BACCABBC
- Z ist gemeinsame Teilfolge von X und Y

Dynamische Programmierung - Längste gemeinsame Teilfolge

Definition 12.3 (Längste gemeinsame Teilfolge)

- Seien X, Y, Z Folgen über Alphabet A.
- Dann heißt Z längste gemeinsame Teilfolge von X und Y, wenn Z gemeinsame Teilfolge von X und Y ist und es keine andere gemeinsame Teilfolge von X und Y gibt, die größere Länge als Z besitzt.

X = ABABCCBAB

Y = CBBABC

Z = BBAB

Z' = CBAB

Dynamische Programmierung
- Längste gemeinsame Teilfolge

Eingabe

- Folge $X=(x_1,...,x_m)$
- Folge $Y=(y_1,...,y_n)$

Ausgabe

- Längste gemeinsame Teilfolge Z von X und Y

Motivation

- Vergleich von Texten (z.B. Unix Befehl „diff“)
- Vergleich von Genomsequenzen

Dynamische Programmierung
- Längste gemeinsame Teilfolge

Einfacher Algorithmus

- Erzeuge alle möglichen Teilfolgen von X
- Teste für jede Teilfolge von X, ob auch Teilfolge von Y
- Merke zu jedem Zeitpunkt bisher längste gemeinsame Teilfolge

Laufzeit

- 2^m mögliche Teilfolgen
- Exponentielle Laufzeit!

ABAABCD

AAC

Dynamische Programmierung
- Längste gemeinsame Teilfolge

Aufgabe

- Versuchen Sie, eine Rekursion für die Länge der längsten gemeinsamen Teilfolge herzuleiten

Dynamische Programmierung - Längste gemeinsame Teilfolge

Lemma 12.4 (Rekursive Struktur der optimalen Lösung)

Seien $X=(x_1,...,x_m)$ und $Y=(y_1,...,y_n)$ beliebige Folgen und sei $Z=(z_1,...,z_k)$ eine längste gemeinsame Teilfolge von X und Y. Dann gilt

1. Ist $x_m = y_n$, dann ist $z_k = x_m = y_n$ und $(z_1,...,z_{k-1})$ ist eine längste gemeinsame Teilfolge von $(x_1,...,x_{m-1})$ und $(y_1,...,y_{n-1})$.
2. Ist $x_m \neq y_n$ und $z_k \neq x_m$, dann ist Z eine längste gemeinsame Teilfolge von $(x_1,...,x_{m-1})$ und Y.
3. Ist $x_m \neq y_n$ und $z_k \neq y_n$, dann ist Z eine längste gemeinsame Teilfolge von X und $(y_1,...,y_{n-1})$.

Dynamische Programmierung
- Längste gemeinsame Teilfolge

Beweis von (1)

- Seien X,Y,Z wie im Lemma und sei $x_m = y_n$.
- Da Z eine längste gemeinsame Teilfolge von X und Y ist, muss $z_k = x_m = y_n$ gelten (ansonsten würden wir durch Anhängen von x_m an Z eine längere Teilfolge erhalten)
- Damit ist $(z_1, z_2, \dots z_{k-1})$ eine gemeinsame Teilfolge der Länge k-1 von $(x_1, x_2, \dots x_{m-1})$ und $(y_1, y_2, \dots y_{n-1})$.
- Gäbe es nun eine Teilfolge Z* von $(x_1, x_2, \dots x_{m-1})$ und $(y_1, y_2, \dots y_{n-1})$ mit Länge mindestens k, dann würden wir durch Anhängen von x_m an Z* eine längere Teilfolge von X und Y erhalten als Z
- Dies ist ein Widerspruch zur Wahl von Z! Damit ist $(z_1, z_2, \dots z_{k-1})$ eine längste gemeinsame Teilfolge von $(x_1, x_2, \dots x_{m-1})$ und $(y_1, y_2, \dots y_{n-1})$.
- Damit ist (1) bewiesen

Dynamische Programmierung
- Längste gemeinsame Teilfolge

Beweis von (2) und (3)

- Falls $z_k \neq x_m$ dann ist Z eine längste gemeinsame Teilfolge von $(x_1, x_2, \dots x_{m-1})$ und Y.
- Gäbe es eine gemeinsame Teilfolge Z* von $(x_1, x_2, \dots x_{m-1})$ und Y mit einer Länge größer k, dann wäre diese auch eine gemeinsame Teilfolge von X und Y, die länger als Z ist.
- Widerspruch zur Wahl von Z!
- Damit ist Z auch längste gemeinsame Teilfolge von $(x_1, x_2, \dots x_{m-1})$ und Y, was (2) zeigt
- Der Beweis von (3) ist analog!

Universität zu Köln

Dynamische Programmierung
- Längste gemeinsame Teilfolge

Korollar 12.5 (Finden der Rekursion)

- Sei L(i,j) die Länge einer längsten gemeinsamen Teilfolge von von $(x_1,\dots,x_i)$ und $(y_1,\dots,y_j)$.
- Dann gilt
- $$L(i,j) = \begin{cases} 0 & \textit{, falls } i=0 \textit{ oder } j=0 \\ L(i-1,j-1)+1 & \text{, falls } i,j>0 \text{ und } x_i = y_j \\ \max\{L(i-1,j), L(i,j-1)\} & \text{, falls } i,j>0 \text{ und } x_i \neq y_j \end{cases}$$

Beweis (Teil 1)

- Ist i=0 oder j=0, so ist eine der beiden Folgen leer und somit ist die längste gemeinsame Teilfolge ebenfalls leer und hat Länge 0
- Ist $x_i = y_j$, so folgt die Aussage aus Lemma 12.4, Teil (1)

Dynamische Programmierung
- Längste gemeinsame Teilfolge

Korollar 12.5 (Finden der Rekursion)

- Sei L(i,j) die Länge einer längsten gemeinsamen Teilfolge von von $(x_1,\dots,x_i)$ und $(y_1,\dots,y_j)$.
- Dann gilt
- $$L(i,j) = \begin{cases} 0 & \text{, falls } i=0 \text{ oder } j=0 \\ L(i-1,j-1)+1 & \text{, falls } i,j>0 \text{ und } x_m = y_n \\ \underline{\max\{L(i-1,j), L(i,j-1)\}} & \text{, falls } i,j>0 \text{ und } x_m \neq y_n \end{cases}$$

Beweis (Teil 2)

- Ansonsten folgt die Aussage aus Lemma 12.4 (2) und (3) mit der Tatsache, dass eine gemeinsame Teilfolge von $(x_1,\dots,x_{i-1})$ und $(y_1,\dots,y_j)$ oder $(x_1,\dots,x_i)$ und $(y_1,\dots,y_{j-1})$ auch eine gemeinsame Teilfolge von $(x_1,\dots,x_i)$ und $(y_1,\dots,y_j)$ ist

Universität zu Köln

Dynamische Programmierung
- Längste gemeinsame Teilfolge

LCS-Länge(X, Y, m, n)

```
L = new array [0..m][0..n]
for i = 0 to m do L[i][0] = 0
for j = 0 to n do L[0][ j] = 0
for i = 1 to m do
    for j = 1 to n do
        if X[i]= Y[j] then L[i][ j] = L[i-1][ j-1] +1
        else
            if L[i-1][ j] ≥ L[i][ j-1] then L[i][ j] = L[i-1][ j]
            else L[i][ j] = L[i][ j-1]
return L[m][n]
```

Dynamische Programmierung
- Längste gemeinsame Teilfolge

	Y	A	B	C	B	D	A	B
X	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

Dynamische Programmierung
- Längste gemeinsame Teilfolge

Lemma 12.6

Der Algorithmus LCS-Länge hat Laufzeit O(nm), wenn die Folgen X,Y Längen n bzw. m haben.

Beweis

- Die Laufzeit wird durch die Initialisierung des Feldes in Zeile 1 sowie die geschachtelten for-Schleifen (Zeilen 4 bis 9) dominiert. Daraus ergibt sich sofort eine Laufzeit von O(nm).

Dynamische Programmierung - Längste gemeinsame Teilfolge

Lemma 12.7

Algorithmus LCS-Länge berechnet die Länge einer längsten gemeinsamen Teilfolge.

Beweisskizze

- Die Korrektheit folgt per Induktion über die Rekursion aus Korollar 12.5.

Dynamische Programmierung
- Längste gemeinsame Teilfolge

Lemma 12.8

Die Ausgabe der längsten gemeinsamen Teilfolge anhand der Tabelle hat Laufzeit O(n+m), wenn die Folgen X,Y Längen n bzw. m haben.

Beweisskizze

- In jedem Schritt bewegen wir uns entweder eine Zeile nach oben oder eine Spalte nach links. Daher ist die Laufzeit durch die Anzahl Zeilen plus die Anzahl Spalten begrenzt. Dies ist O(n+m).

Dynamische Programmierung

Zusammenfassung

- Dynamische Programmierung auf Zeichenketten
  - Längste gemeinsame Teilfolge
  - Finden einer Rekursionsgleichung
  - Entwicklung des Algorithmus
  - Finden einer optimalen Lösung

Referenzen

- T. Cormen, C. Leisserson, R. Rivest, C. Stein. Introduction to Algorithms. The MIT press. Second edition, 2001.