technische universität
dortmund

LS 2 /
Informatik

Datenstrukturen, Algorithmen und Programmierung 2 (DAP2)

Dynamische Programmierung

Rekursiver Ansatz

- Lösen eines Problems durch Lösen mehrerer kleinerer Teilprobleme, aus denen sich die Lösung für das Ausgangsproblem zusammensetzt
- Unterschied zu Teile&Herrsche: Die Problemgröße schrumpft normalerweise nur um eine Konstante

Phänomen

- Mehrfachberechnungen von Lösungen

Methode

- Lösungen zu Teilproblemen werden iterativ beginnend mit den Lösungen der kleinsten Teilprobleme berechnet (*bottom-up*).
- Speichern einmal berechneter Lösungen in einer Tabelle

Dynamische Programmierung

Problem: Partition

- Eingabe: Menge M mit n natürlichen Zahlen
- Ausgabe: Ja, gdw. man M in zwei Mengen L und R partitionieren kann, so dass $\sum_{x \in L} x = \sum_{x \in R} x$ gilt; nein sonst

Beispiel

- 4, 7, 9, 10, 13, 23

Dynamische Programmierung

Problem: Partition

- Eingabe: Menge M mit n natürlichen Zahlen
- Ausgabe: Ja, gdw. man M in zwei Mengen L und R partitionieren kann, so dass $\sum_{x \in L} x = \sum_{x \in R} x$ gilt; nein sonst

Beispiel

- 4, 7, 9, 10, 13, 23

Dynamische Programmierung

Problem: Partition

- Eingabe: Menge M mit n natürlichen Zahlen
- Ausgabe: Ja, gdw. man M in zwei Mengen L und R partitionieren kann, so dass $\sum_{x \in L} x = \sum_{x \in R} x$ gilt; nein sonst

Beispiel

- 4, 7, 9, 10, 13, 23
- 4 + 7 + 9 + 13 = 33

Dynamische Programmierung

Problem: Partition

- Eingabe: Menge M mit n natürlichen Zahlen
- Ausgabe: Ja, gdw. man M in zwei Mengen L und R partitionieren kann, so dass $\sum_{x \in L} x = \sum_{x \in R} x$ gilt; nein sonst

Beispiel

- 4, 7, 9, 10, 13, 23
- 4 + 7 + 9 + 13 = 33
- 10 +23 = 33
- Ausgabe: Ja

Dynamische Programmierung

Beobachtung

- Sei M eine Menge mit n natürlichen Zahlen.

  M kann genau dann in zwei Mengen L,R mit $\sum_{x \in L} x = \sum_{x \in R} x$ partitioniert werden, wenn es eine Teilmenge L von M gibt mit $\sum_{x \in L} x = W/2$, wobei $W = \sum_{x \in M} x$ die Summe aller Zahlen aus M ist.

Dynamische Programmierung

Beobachtung

- Sei M eine Menge mit n natürlichen Zahlen.

  M kann genau dann in zwei Mengen L,R mit $\sum_{x \in L} x = \sum_{x \in R} x$ partitioniert werden, wenn es eine Teilmenge L von M gibt mit $\sum_{x \in L} x = W/2$, wobei $W = \sum_{x \in M} x$ die Summe aller Zahlen aus M ist.

Neue Frage

- Gibt es $L \subseteq M$ mit $\sum_{x \in L} x = W/2$?

Dynamische Programmierung

Allgemeinere Frage

- Welche Zahlen lassen sich als Summe einer Teilmenge von M darstellen?

Dynamische Programmierung

Formulierung als Funktion

- Sei G[i,j] = 1 , wenn man die Zahl j als Summe einer Teilmenge der ersten i Zahlen aus M darstellen kann
- Sei G[i,j] = 0 , sonst
- Sei G[0,0] = 1
  (Man kann die Null als Summe über die leere Menge darstellen)
- Sei G[0,j] = 0 für j≠0
  (Man kann keine Zahl ungleich 0 als Summe über die leere Menge darstellen)

Rekursion

- G[i,j] = 1, wenn G[i-1,j] =1 oder (j≥M[i] und G[i-1,j-M[i]] = 1)
- G[i,j] = 0, sonst

Dynamische Programmierung

PartitionDynamicProg(M)

```
1.  W←0
2.  for i ← 1 to length[M] do
3.      W ← W+ M[i]
4.  if W ist ungerade then return 0
5.  Initialisiere Feld G[0..length[M]][0..W]
6.  for i ← 0 to length[M] do
7.      for j ← 0 to W/2 do
8.          if j=0 then G[i,j] ← 1
9.          else if i=0 then G[i,j] ← 0
10.                 else if G[i-1,j]=1 or (M[i]≤j und G[i-1,j-M[i]]=1) then G[i,j] ← 1
11.                         else G[i,j] ← 0
12. return G[length[M],W/2]
```

Dynamische Programmierung

Lemma 19

- Algorithmus PartitionDynamicProg ist korrekt.

Beweis

- PartitionDynamicProg berechnet zunächst die Summe W der Elemente aus M. Ist diese ungerade, so kann es keine Partition in zwei gleich große Teilmengen geben.

Dynamische Programmierung

Lemma 19

- Algorithmus PartitionDynamicProg ist korrekt.

Beweis

- PartitionDynamicProg berechnet zunächst die Summe W der Elemente aus M. Ist diese ungerade, so kann es keine Partition in zwei gleich große Teilmengen geben.
- Ansonsten berechnet der Algorithmus die Funktion G, mit
- G[i,0]= 1 für alle i
- G[0,j]= 0 für alle j>0
- G[i,j] = 1, gdw. G[i-1,j]=1 oder (j≥M[i] und G[i-1,j-M[i]]=1)

Dynamische Programmierung

Lemma 19

- Algorithmus PartitionDynamicProg ist korrekt.

Beweis

- PartitionDynamicProg berechnet zunächst die Summe W der Elemente aus M. Ist diese ungerade, so kann es keine Partition in zwei gleich große Teilmengen geben.
- Ansonsten berechnet der Algorithmus die Funktion G, mit
- G[i,0]= 1 für alle i
- G[0,j]= 0 für alle j>0
- G[i,j] = 1, gdw. G[i-1,j]=1 oder (j≥M[i] und G[i-1,j-M[i]]=1)
- Der Algorithmus gibt 1 zurück, gdw. G[length[M],W/2] = 1.

Dynamische Programmierung

Lemma 19

- Algorithmus PartitionDynamicProg ist korrekt.

Beweis

- PartitionDynamicProg berechnet zunächst die Summe W der Elemente aus M. Ist diese ungerade, so kann es keine Partition in zwei gleich große Teilmengen geben.
- Ansonsten berechnet der Algorithmus die Funktion G, mit
- G[i,0]= 1 für alle i
- G[0,j]= 0 für alle j>0
- G[i,j] = 1, gdw. G[i-1,j]=1 oder (j≥M[i] und G[i-1,j-M[i]]=1)
- Der Algorithmus gibt 1 zurück, gdw. G[length[M],W/2] = 1.

Dynamische Programmierung

Lemma 19

- Algorithmus PartitionDynamicProg ist korrekt.

Beweis

- Wir zeigen per Induktion über i:
- G[i,j] = 1, gdw. man j als Summe einer Teilmenge von M[1..i] darstellen kann

Dynamische Programmierung

Lemma 19

- Algorithmus PartitionDynamicProg ist korrekt.

Beweis

- Wir zeigen per Induktion über i:
- G[i,j] = 1, gdw. man j als Summe einer Teilmenge von M[1..i] darstellen kann
- (I.A.) Für i=0, j=0 kann man j als Summe der leeren Teilmenge darstellen
  Für i=0, j>0, kann man j nicht als Summe einer Teilmenge der leeren Menge darstellen

Dynamische Programmierung

Lemma 19

- Algorithmus PartitionDynamicProg ist korrekt.

Beweis

- Wir zeigen per Induktion über i:
- G[i,j] = 1, gdw. man j als Summe einer Teilmenge von M[1..i] darstellen kann
- (I.A.) Für i=0, j=0 kann man j als Summe der leeren Teilmenge darstellen
  Für i=0, j>0, kann man j nicht als Summe einer Teilmenge der leeren Menge darstellen
- (I.V.) Für alle k<i und alle j wird G[k,j] korrekt berechnet

Dynamische Programmierung

Lemma 19

- Algorithmus PartitionDynamicProg ist korrekt.

Beweis

- Wir zeigen per Induktion über i:
- G[i,j] = 1, gdw. man j als Summe einer Teilmenge von M[1..i] darstellen kann
- (I.A.) Für i=0, j=0 kann man j als Summe der leeren Teilmenge darstellen
  Für i=0, j>0, kann man j nicht als Summe einer Teilmenge der leeren Menge darstellen
- (I.V.) Für alle k<i und alle j wird G[k,j] korrekt berechnet
- (I.S.) Z.z. G[i,j] = 1, gdw. man j als Summe einer Teilmenge von M[1..i] darstellen kann.

Dynamische Programmierung

Lemma 19

- Algorithmus PartitionDynamicProg ist korrekt.

Beweis

- (I.S.) Z.z. G[i,j] = 1, gdw. man j als Summe einer Teilmenge von M[1..i] darstellen kann.

Dynamische Programmierung

Lemma 19

- Algorithmus PartitionDynamicProg ist korrekt.

Beweis

- (I.S.) Z.z. G[i,j] = 1, gdw. man j als Summe einer Teilmenge von M[1..i] darstellen kann.
- „<=“ Kann man j als Summe einer Teilmenge von M[1..i], so kann man j entweder als Teilmenge von M[1..i-1] darstellen oder als M[i] vereinigt mit einer Teilmenge von M[1..i-1].

Dynamische Programmierung

Lemma 19

- Algorithmus PartitionDynamicProg ist korrekt.

Beweis

- (I.S.) Z.z. G[i,j] = 1, gdw. man j als Summe einer Teilmenge von M[1..i] darstellen kann.
- „<=“ Kann man j als Summe einer Teilmenge von M[1..i], so kann man j entweder als Teilmenge von M[1..i-1] darstellen oder als M[i] vereinigt mit einer Teilmenge von M[1..i-1]. Im ersten Fall folgt aus (I.V.), dass G[i-1,j]=1 ist und somit auch G[i,j]=1.

Dynamische Programmierung

Lemma 19

- Algorithmus PartitionDynamicProg ist korrekt.

Beweis

- (I.S.) Z.z. G[i,j] = 1, gdw. man j als Summe einer Teilmenge von M[1..i] darstellen kann.
- „<=“ Kann man j als Summe einer Teilmenge von M[1..i], so kann man j entweder als Teilmenge von M[1..i-1] darstellen oder als M[i] vereinigt mit einer Teilmenge von M[1..i-1]. Im ersten Fall folgt aus (I.V.), dass G[i-1,j]=1 ist und somit auch G[i,j]=1. Im zweiten Fall muss die Teilmenge von M[1..i-1] Summe j-M[i] haben.

Dynamische Programmierung

Lemma 19

- Algorithmus PartitionDynamicProg ist korrekt.

Beweis

- (I.S.) Z.z. G[i,j] = 1, gdw. man j als Summe einer Teilmenge von M[1..i] darstellen kann.
- „<=“ Kann man j als Summe einer Teilmenge von M[1..i], so kann man j entweder als Teilmenge von M[1..i-1] darstellen oder als M[i] vereinigt mit einer Teilmenge von M[1..i-1]. Im ersten Fall folgt aus (I.V.), dass G[i-1,j]=1 ist und somit auch G[i,j]=1. Im zweiten Fall muss die Teilmenge von M[1..i-1] Summe j-M[i] haben. Nach (I.V.) ist dann aber G[i-1,j-M[i]]=1 und somit G[i,j] = 1.

Dynamische Programmierung

Lemma 19

- Algorithmus PartitionDynamicProg ist korrekt.

Beweis

- (I.S.) Z.z. G[i,j] = 1, gdw. man j als Summe einer Teilmenge von M[1..i] darstellen kann.
- „<=“ Kann man j als Summe einer Teilmenge von M[1..i], so kann man j entweder als Teilmenge von M[1..i-1] darstellen oder als M[i] vereinigt mit einer Teilmenge von M[1..i-1]. Im ersten Fall folgt aus (I.V.), dass G[i-1,j]=1 ist und somit auch G[i,j]=1. Im zweiten Fall muss die Teilmenge von M[1..i-1] Summe j-M[i] haben. Nach (I.V.) ist dann aber G[i-1,j-M[i]]=1 und somit G[i,j] = 1.
- „=>“ Ist G[i,j]=1, so war entweder G[i-1,j]=1 oder G[i-1,j-M[i]]=1. Nach (I.V.) kann man entweder j oder j-M[i] als Teilmenge von M[1..i-1] darstellen. Somit kann man j als Teilmenge von M[1..i] darstellen.

Dynamische Programmierung

Lemma 19

- Algorithmus PartitionDynamicProg ist korrekt.

Beweis

- (I.S.) Z.z. G[i,j] = 1, gdw. man j als Summe einer Teilmenge von M[1..i] darstellen kann.
- „<=“ Kann man j als Summe einer Teilmenge von M[1..i], so kann man j entweder als Teilmenge von M[1..i-1] darstellen oder als M[i] vereinigt mit einer Teilmenge von M[1..i-1]. Im ersten Fall folgt aus (I.V.), dass G[i-1,j]=1 ist und somit auch G[i,j]=1. Im zweiten Fall muss die Teilmenge von M[1..i-1] Summe j-M[i] haben. Nach (I.V.) ist dann aber G[i-1,j-M[i]]=1 und somit G[i,j] = 1.
- „=>“ Ist G[i,j]=1, so war entweder G[i-1,j]=1 oder G[i-1,j-M[i]]=1. Nach (I.V.) kann man entweder j oder j-M[i] als Teilmenge von M[1..i-1] darstellen. Somit kann man j als Teilmenge von M[1..i] darstellen.

Dynamische Programmierung

Lemma 19

- Algorithmus PartitionDynamicProg ist korrekt.

Beweis

- Somit gilt G[length[M],W/2] = 1 gdw. man W/2 als Summe einer Teilmenge von M[1..length[M]] darstellen kann.

Dynamische Programmierung

Satz 20

- Sei M eine Menge von n natürlichen Zahlen und W die Summe der Zahlen aus M. Algorithmus PartitionDynamicProg löst Partition in Zeit O(nW).

Beweis

- Die Korrektheit des Algorithmus folgt aus Lemma 19.
- Die Laufzeit ist offensichtlich O(nW).

Dynamische Programmierung

Bemerkung

- Partition ist ein NP-vollständiges Problem
- Damit gibt es wahrscheinlich keinen polynomieller Algorithmus für Partition

Warum ist unser Algorithmus nicht polynomiell?

- Die Laufzeit hängt von W ab
- Sind die Zahlen aus M exponentiell groß, so ist die Laufzeit ebenfalls exponentiell

Dynamische Programmierung

Optimale Unterstrukturen

- Ein Problem hat optimale Unterstrukturen, wenn man eine optimale Lösung optimale Lösungen für Unterprobleme enthält
- Dies ist oft ein Indikator, dass dynamische Programmierung eingesetzt werden kann

Längste gemeinsame Teilfolge

Definition:

- Seien $X=(x_1,...,x_m)$ und $Y=(y_1,...,y_n)$ zwei Teilfolgen, wobei $x_i, y_j \in A$ für ein endliches Alphabet A.
- Dann heißt Y Teilfolge von X, wenn es aufsteigend sortierte Indizes $i_1,...,i_n$ gibt mit $x_{ij} = y_j$ für $j = 1,...,n$.

Längste gemeinsame Teilfolge

Definition:

- Seien $X=(x_1,...,x_m)$ und $Y=(y_1,...,y_n)$ zwei Teilfolgen, wobei $x_i, y_j \in A$ für ein endliches Alphabet A.
- Dann heißt Y Teilfolge von X, wenn es aufsteigend sortierte Indizes $i_1,...,i_n$ gibt mit $x_{ij} = y_j$ für j = 1,...,n.

Beispiel:

Folge Y

B	C	A	C

Folge X

A	B	A	C	A	B	C

Längste gemeinsame Teilfolge

Definition:

- Seien $X=(x_1,...,x_m)$ und $Y=(y_1,...,y_n)$ zwei Teilfolgen, wobei $x_i, y_j \in A$ für ein endliches Alphabet A.
- Dann heißt Y Teilfolge von X, wenn es aufsteigend sortierte Indizes $i_1,...,i_n$ gibt mit $x_{ij} = y_j$ für j = 1,...,n.

Beispiel:

Folge Y

B	C	A	C

Folge X

A	B	A	C	A	B	C

Längste gemeinsame Teilfolge

Definition:

- Seien $X=(x_1,...,x_m)$ und $Y=(y_1,...,y_n)$ zwei Teilfolgen, wobei $x_i, y_j \in A$ für ein endliches Alphabet A.
- Dann heißt Y Teilfolge von X, wenn es aufsteigend sortierte Indizes $i_1,...,i_n$ gibt mit $x_{ij} = y_j$ für j = 1,...,n.

Beispiel:

Folge Y

B	C	A	C

Folge X

A	B	A	C	A	B	C

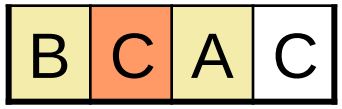
Längste gemeinsame Teilfolge

Definition:

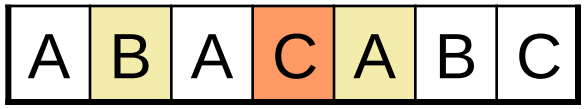
- Seien $X=(x_1,...,x_m)$ und $Y=(y_1,...,y_n)$ zwei Teilfolgen, wobei $x_i, y_j \in A$ für ein endliches Alphabet A.
- Dann heißt Y Teilfolge von X, wenn es aufsteigend sortierte Indizes $i_1,...,i_n$ gibt mit $x_{ij} = y_j$ für j = 1,...,n.

Beispiel:

Folge Y



Folge X



Längste gemeinsame Teilfolge

Definition:

- Seien $X=(x_1,...,x_m)$ und $Y=(y_1,...,y_n)$ zwei Teilfolgen, wobei $x_i, y_j \in A$ für ein endliches Alphabet A.
- Dann heißt Y Teilfolge von X, wenn es aufsteigend sortierte Indizes $i_1,...,i_n$ gibt mit $x_{ij} = y_j$ für j = 1,...,n.

Beispiel:

Folge Y

B	C	A	C

Folge X

A	B	A	C	A	B	C

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

Längste gemeinsame Teilfolge

Definition:

- 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.

Längste gemeinsame Teilfolge

Definition:

- 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:

Folge Z

B	C	A	C

Folge X

A	B	A	C	A	B	C

Folge Y

B	A	C	C	A	B	B	C

- Z ist gemeinsame Teilfolge von X und Y

Längste gemeinsame Teilfolge

Definition:

- 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:

Folge Z

B	C	A	C

Folge X

A	B	A	C	A	B	C

Folge Y

B	A	C	C	A	B	B	C

- Z ist gemeinsame Teilfolge von X und Y

Längste gemeinsame Teilfolge

Definition:

- 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:

Folge Z

B	C	A	C

Folge X

A	B	A	C	A	B	C

Folge Y

B	A	C	C	A	B	B	C

- Z ist gemeinsame Teilfolge von X und Y

Längste gemeinsame Teilfolge

Definition:

- Seien X, Y, Z Folgen über 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.

Längste gemeinsame Teilfolge

Definition:

- Seien X, Y, Z Folgen über 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.

Beispiel:

Folge X

A	B	A	C	A	B	C

Folge Y

B	A	C	C	A	B	B	C

Längste gemeinsame Teilfolge

Definition:

- Seien X, Y, Z Folgen über 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.

Beispiel:

Folge X

A	B	A	C	A	B	C

Folge Y

B	A	C	C	A	B	B	C

Folge Z_1

B	C	A	C

Längste gemeinsame Teilfolge

Definition:

- Seien X, Y, Z Folgen über 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.

Beispiel:

Folge X

A	B	A	C	A	B	C

Folge Y

B	A	C	C	A	B	B	C

Folge Z_1

B	C	A	C

Folge Z_2

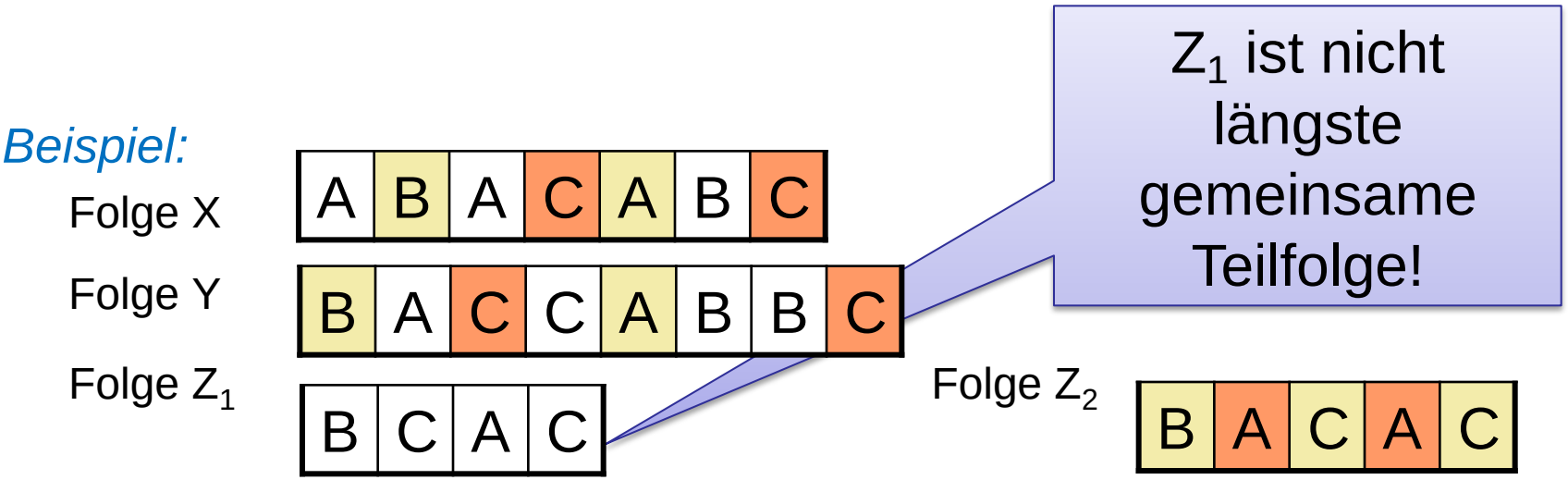
B	A	C	A	C

Längste gemeinsame Teilfolge

Definition:

- Seien X, Y, Z Folgen über 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.

Beispiel:



Problem LCS

Eingabe:

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

Ausgabe:

- Längste gemeinsame Teilfolge Z (**L**ongest **C**ommon **S**ubsequenz)

Problem LCS

Eingabe:

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

Ausgabe:

- Längste gemeinsame Teilfolge Z (**L**ongest **C**ommon **S**ubsequence)

Beispiel:

Folge X

A	B	C	B	D	A	B

Folge Y

B	D	C	A	B	A

Einfacher Ansatz

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!

Struktur von LCS

Satz 21

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 $(y_1,...,y_{n-1})$.

Struktur von LCS

Satz 21

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.

Struktur von LCS

Satz 21

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 $(y_1,...,y_{n-1})$.

Struktur von LCS

Satz 21

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

Optimale Unterstrukturen

Struktur von LCS

Beweis

(1) **Annahme:** $Z=(z_1,...,z_k)$ ist längste gemeinsame Teilfolge, $x_m = y_n$ und $z_k \neq x_m$

Dann können wir $z_{k+1} = x_m$ setzen, um eine gemeinsame Teilfolge von X und Y der Länge k+1 zu erhalten. Widerspruch: Z ist eine *längste* gemeinsame Teilfolge von X und Y.

$\Rightarrow$ $z_k = x_m = y_n$

$\Rightarrow$ $(z_1, z_2, \ldots z_{k-1})$ ist eine gemeinsame Teilfolge der Länge k-1 von

$(x_1, x_2, \ldots x_{m-1})$ und $(y_1, y_2, \ldots y_{n-1})$.

Noch z.z.: $(z_1, z_2, \ldots z_{k-1})$ ist längste gemeinsame Teilfolge von $(x_1, x_2, \ldots x_{m-1})$ und $(y_1, y_2, \ldots y_{n-1})$

Annahme: Es gibt eine gemeinsame Teilfolge W von $(x_1, x_2, \ldots x_{m-1})$ und $(y_1, y_2, \ldots y_{n-1})$, die mindestens Länge k hat. Dann erzeugt das Anhängen von $z_k = x_m$ an W eine gemeinsame Teilfolge von X und Y, deren Länge mindestens k+1 ist. Widerspruch zur Optimalität von Z.

Struktur von LCS

Beweis

(2) Falls $z_k \neq x_m$ dann ist Z eine gemeinsame Teilfolge von $(x_1, x_2, \ldots x_{m-1})$ und Y.

Annahme: Es gibt eine gemeinsame Teilfolge W von $(x_1, x_2, \ldots x_{m-1})$ und Y mit einer Länge größer k.

Dann ist W auch eine gemeinsame Teilfolge von X und Y. Widerspruch: Z ist längste gemeinsame Teilfolge von X und Y.

(3) Der Beweis ist analog zu (2)

Rekursion für Länge von LCS

Korollar 22

Sei C[i][j] die Länge einer längsten gemeinsamen Teilfolge von $(x_1,...,x_i)$ und $(y_1,...,y_j)$. Dann gilt:

$$C[i][j] = \begin{cases} 0 & \text{falls } i = 0 \text{ oder } j = 0 \end{cases}$$

Rekursion für Länge von LCS

Korollar 22

Sei C[i][j] die Länge einer längsten gemeinsamen Teilfolge von $(x_1,...,x_i)$ und $(y_1,...,y_j)$. Dann gilt:

$$C[i][j] = \begin{cases} 0 & \text{falls } i = 0 \text{ oder } j = 0 \\ C[i-1][j-1]+1 & \text{falls } i, j > 0 \text{ und } x_i = y_j \end{cases}$$

Rekursion für Länge von LCS

Korollar 22

Sei C[i][j] die Länge einer längsten gemeinsamen Teilfolge von (x_1,...,x_i) und (y_1,...,y_j). Dann gilt:

$$C[i][j] = \begin{cases} 0 & \text{falls } i = 0 \text{ oder } j = 0 \\ C[i-1][j-1]+1 & \text{falls } i, j > 0 \text{ und } x_i = y_j \\ \max\{C[i-1][j], C[i][j-1]\} & \text{falls } i, j > 0 \text{ und } x_i \neq y_j \end{cases}$$

Rekursion für Länge von LCS

Korollar 22

Sei C[i][j] die Länge einer längsten gemeinsamen Teilfolge von $(x_1,...,x_i)$ und $(y_1,...,y_j)$. Dann gilt:

$$C[i][j] = \begin{cases} 0 & \text{falls } i = 0 \text{ oder } j = 0 \\ C[i-1][j-1]+1 & \text{falls } i, j > 0 \text{ und } x_i = y_j \\ \max\{C[i-1][j], C[i][j-1]\} & \text{falls } i, j > 0 \text{ und } x_i \neq y_j \end{cases}$$

Rekursion für Länge von LCS

Korollar 22

Sei C[i][j] die Länge einer längsten gemeinsamen Teilfolge von $(x_1,...,x_i)$ und $(y_1,...,y_j)$. Dann gilt:

$$C[i][j] = \begin{cases} 0 & \text{falls } i = 0 \text{ oder } j = 0 \\ C[i-1][j-1]+1 & \text{falls } i, j > 0 \text{ und } x_i = y_j \\ \max\{C[i-1][j], C[i][j-1]\} & \text{falls } i, j > 0 \text{ und } x_i \neq y_j \end{cases}$$

Beobachtung:

Rekursive Berechnung der C[i][j] würde zu Berechnung immer wieder derselben Werte führen. Dieses ist ineffizient. Berechnen daher die Werte C[i][j] iterativ, nämlich zeilenweise.

Berechnung der C[i][j] Werte

LCS-Länge(X, Y)

1. m ← length[X]
2. n ← length[Y]
3. **new** array C[0..m][0..n]
4. **for** i ← 0 **to** m **do** C[i][0] ← 0
5. **for** j ← 0 **to** n **do** C[0][j] ← 0
6. **for** i ← 1 **to** m **do**
7. **for** j ← 1 **to** n **do**
8. ➢ Längenberechnung(X, Y, C, i, j)
9. **return** C

Berechnung der C[i][j] Werte

LCS-Länge(X, Y)

1. m ← length[X]
2. n ← length[Y]
3. **new** array C[0..m][0..n]
4. **for** i ← 0 **to** m **do** C[i][0] ← 0
5. **for** j ← 0 **to** n **do** C[0][j] ← 0
6. **for** i ← 1 **to** m **do**
7. **for** j ← 1 **to** n **do**
8. ➢ Längenberechnung(X, Y, C, i, j)
9. **return** C

Berechnung der C[i][j] Werte

LCS-Länge(X, Y)

1. m ← length[X]
2. n ← length[Y]
3. **new** array C[0..m][0..n]
4. **for** i ← 0 **to** m **do** C[i][0] ← 0
5. **for** j ← 0 **to** n **do** C[0][j] ← 0
6. **for** i ← 1 **to** m **do**
7. **for** j ← 1 **to** n **do**
8. ➢ Längenberechnung(X, Y, C, i, j)
9. **return** C

Tabelle für die C[i][j] Werte anlegen.

Berechnung der C[i][j] Werte

LCS-Länge(X, Y)

1. m ← length[X]
2. n ← length[Y]
3. **new** array C[0..m][0..n]
4. **for** i ← 0 **to** m **do** C[i][0] ← 0
5. **for** j ← 0 **to** n **do** C[0][j] ← 0
6. **for** i ← 1 **to** m **do**
7. **for** j ← 1 **to** n **do**
8. ➢ Längenberechnung(X, Y, C, i, j)
9. **return** C

Erste Spalte der Tabelle auf 0 setzen.

Berechnung der C[i][j] Werte

LCS-Länge(X, Y)

1. m ← length[X]
2. n ← length[Y]
3. **new** array C[0..m][0..n]
4. **for** i ← 0 **to** m **do** C[i][0] ← 0
5. **for** j ← 0 **to** n **do** C[0][j] ← 0
6. **for** i ← 1 **to** m **do**
7. **for** j ← 1 **to** n **do**
8. ➢ Längenberechnung(X, Y, C, i, j)
9. **return** C

Erste Reihe der Tabelle auf 0 setzen.

Berechnung der C[i][j] Werte

LCS-Länge(X, Y)

1. m ← length[X]
2. n ← length[Y]
3. **new** array C[0..m][0..n]
4. **for** i ← 0 **to** m **do** C[i][0] ← 0
5. **for** j ← 0 **to** n **do** C[0][j] ← 0
6. **for** i ← 1 **to** m **do**
7. **for** j ← 1 **to** n **do**
8. ➢ Längenberechnung(X, Y, C, i, j)
9. **return** C

Berechnung der C[i][j] Werte

Längenberechnung(X, Y, C, i, j)

1. **if** $x_i = y_j$ **then** C[i][j] ← C[i-1][j-1] +1
2. **else**
3. **if** C[i-1][j] ≥ C[i][j-1] **then** C[i][j] ← C[i-1][j]
4. **else** C[i][j] ← C[i][j-1]

$$C[i][j] = \begin{cases} 0 & \text{falls } i = 0 \text{ oder } j = 0 \\ C[i-1][j-1]+1 & \text{falls } i, j > 0 \text{ und } x_i = y_j \\ \max\{C[i-1][j], C[i][j-1]\} & \text{falls } i, j > 0 \text{ und } x_i \neq y_j \end{cases}$$

Berechnung der C[i][j] Werte

Längenberechnung(X, Y, C, i, j)

1. **if** $x_i = y_j$ **then** C[i][j] ← C[i-1][j-1] +1
2. **else**
3. **if** C[i-1][j] ≥ C[i][j-1] **then** C[i][j] ← C[i-1][j]
4. **else** C[i][j] ← C[i][j-1]

$$C[i][j] = \begin{cases} 0 & \text{falls } i = 0 \text{ oder } j = 0 \\ C[i-1][j-1]+1 & \text{falls } i, j > 0 \text{ und } x_i = y_j \\ \max\{C[i-1][j], C[i][j-1]\} & \text{falls } i, j > 0 \text{ und } x_i \neq y_j \end{cases}$$

Berechnung der C[i][j] Werte

LCS-Länge(X, Y)

1. m ← length[X]
2. n ← length[Y]
3. **new** array C[0..m][0..n]
4. **for** i ← 0 **to** m **do** C[i][0] ← 0
5. **for** j ← 0 **to** n **do** C[0][j] ← 0
6. **for** i ← 1 **to** m **do**
7. **for** j ← 1 **to** n **do**
8. ➢ Längenberechnung(X, Y, C, i, j)
9. **return** C

i	j	0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i							
1	A							
2	B							
3	C							
4	B							
5	D							
6	A							
7	B							

i	j	0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0						
1	A	0						
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

i		j	0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A	
0	x_i		0	0	0	0	0	0	0
1	A		0						
2	B		0						
3	C		0						
4	B		0						
5	D		0						
6	A		0						
7	B		0						

i	j	0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0						
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

i	j	0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0						
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0					
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

i		j	0	1	2	3	4	5	6
		y_j		B	D	C	A	B	A
0	x_i		0	0	0	0	0	0	0
1	A		0	↑ 0					
2	B		0						
3	C		0						
4	B		0						
5	D		0						
6	A		0						
7	B		0						

i	j	0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0					
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

i		j	0	1	2	3	4	5	6
		y_j		B	D	C	A	B	A
0	x_i		0	0	0	0	0	0	0
1	A		0	↑ 0	↑ 0				
2	B		0						
3	C		0						
4	B		0						
5	D		0						
6	A		0						
7	B		0						

i	j	0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0			
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0			
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

i		j	0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A	
0	x_i		0	0	0	0	0	0	0
1	A		0	↑ 0	↑ 0	↑ 0			
2	B		0						
3	C		0						
4	B		0						
5	D		0						
6	A		0						
7	B		0						

i		j	0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A	
0	x_i		0	0	0	0	0	0	0
1	A		0	↑ 0	↑ 0	↑ 0	↖ 1		
2	B		0						
3	C		0						
4	B		0						
5	D		0						
6	A		0						
7	B		0						

i		j	0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A	
0	x_i		0	0	0	0	0	0	0
1	A		0	↑ 0	↑ 0	↑ 0	↖ 1		
2	B		0						
3	C		0						
4	B		0						
5	D		0						
6	A		0						
7	B		0						

i		j	0	1	2	3	4	5	6
		y_j		B	D	C	A	B	A
0	x_i		0	0	0	0	0	0	0
1	A		0	↑ 0	↑ 0	↑ 0	↖ 1		
2	B		0						
3	C		0						
4	B		0						
5	D		0						
6	A		0						
7	B		0						

i	j	0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

i		j	0	1	2	3	4	5	6
		y_j		B	D	C	A	B	A
0	x_i		0	0	0	0	0	0	0
1	A		0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	
2	B		0						
3	C		0						
4	B		0						
5	D		0						
6	A		0						
7	B		0						

i		j	0	1	2	3	4	5	6
		y_j		B	D	C	A	B	A
0	x_i		0	0	0	0	0	0	0
1	A		0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	
2	B		0						
3	C		0						
4	B		0						
5	D		0						
6	A		0						
7	B		0						

i		j	0	1	2	3	4	5	6
		y_j		B	D	C	A	B	A
0	x_i		0	0	0	0	0	0	0
1	A		0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B		0						
3	C		0						
4	B		0						
5	D		0						
6	A		0						
7	B		0						

i	j	0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1					
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

i	j	0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1				
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

i	j	0	1	2	3	4	5	6
	y_j		B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1			
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

i		j	0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A	
0	x_i		0	0	0	0	0	0	0
1	A		0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B		0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C		0						
4	B		0						
5	D		0						
6	A		0						
7	B		0						

i		j	0	1	2	3	4	5	6
		y_j		B	D	C	A	B	A
0	x_i		0	0	0	0	0	0	0
1	A		0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B		0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C		0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B		0						
5	D		0						
6	A		0						
7	B		0						

i	j	0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0						
6	A	0						
7	B	0						

i	j	0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

i	j	0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

i	j	0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

i	j	0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

i		j	0	1	2	3	4	5	6
		y_j		B	D	C	A	B	A
0	x_i		0	0	0	0	0	0	0
1	A		0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B		0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C		0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B		0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D		0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A		0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B		0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

i	j	0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

i		j	0	1	2	3	4	5	6
		y_j		B	D	C	A	B	A
0	x_i		0	0	0	0	0	0	0
1	A		0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B		0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C		0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B		0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D		0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A		0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B		0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

i	j	0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

i	j	0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

i	j	0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

i		j	0	1	2	3	4	5	6
		y_j		B	D	C	A	B	A
0	x_i		0	0	0	0	0	0	0
1	A		0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B		0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C		0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B		0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D		0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A		0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B		0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

i	j	0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

i	j	0	1	2	3	4	5	6
	y_j		B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

i	j	0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Laufzeitanalyse

Lemma 22

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

Beweis

- Die Laufzeit wird durch die Initialisierung des Feldes in Zeile 3 sowie die geschachtelten for-Schleifen (Zeile 6 bis 8) dominiert. Daraus ergibt sich sofort eine von O(nm).

Laufzeitanalyse

Lemma 23

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

Beweisskizze

- Die Korrektheit folgt per Induktion über die Rekursion aus Korollar 22.

Laufzeitanalyse

Lemma 24

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

Beweis

- 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).

Vorgehensweise bei dynamischer Programmierung

1. Bestimme rekursive Struktur einer optimalen Lösung.

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

3. Transformiere rekursiv 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.