Dynamisches Programmieren

1957 R. Bellmann: Dynamic Programming für math. Optimierungsprobleme

Methode für Probleme,

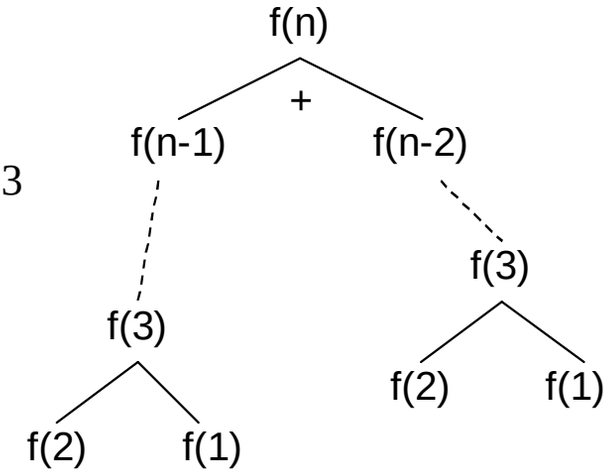
1. die rekursiv beschreibbar sind,
2. die dem Optimalitätsprinzip genügen,

   d. h. eine optimale Lösung für das Ausgangsproblem setzt sich aus optimalen Lösungen für die kleineren Probleme, auf die rekurriert wird, zusammen, und
3. deren Berechnung mehrfach gleiche Teillösungen benötigt

Fibonacci-Zahlen

1. Rekursive Beschreibung des Problems:

$$f(1) = 1$$
$$f(2) = 1$$
$$f(n) = f(n\text{-}1) + f(n\text{-}2), \text{ falls } n >= 3$$



Komplexität: $O(2^n)$

Idee!
Spare möglichst viele Mehrfach-Berechnungen

2. Bestimme die Menge R der kleineren Probleme, auf die bei Lösung eines Problems P (direkt oder indirekt) rekuriert wird:

   hier: für das Problem n ergibt sich R = {1, ..., n-1}

3. Bestimme eine mit der Aufrufreihenfolge kompatible Reihenfolge P_1, ... , P_r der Probleme in R, so dass bei der Lösung von Problemen P_x, $1 \leq x \leq r$, nur auf Probleme P_y mit Index y kleiner als x rekurriert wird.

   hier: Reihenfolge 1, 2, ..., n-1

4. Sukzessives Berechnen und Speichern von Lösungen für P_1, ... , P_r in dieser Reihenfolge (wobei berechnete Lösungen solange gespeichert bleiben, wie sie noch benötigt werden).

i	f(i)
1	1
2	1
3	f(2)+f(1)=2
4	f(3)+f(2)=3

=> O(n)

Anm.: Aus der Tabelle ist ersichtlich, dass nur zwei Hilfsvariablen benötigt werden.

Rucksackproblem (Optimierungsproblem)

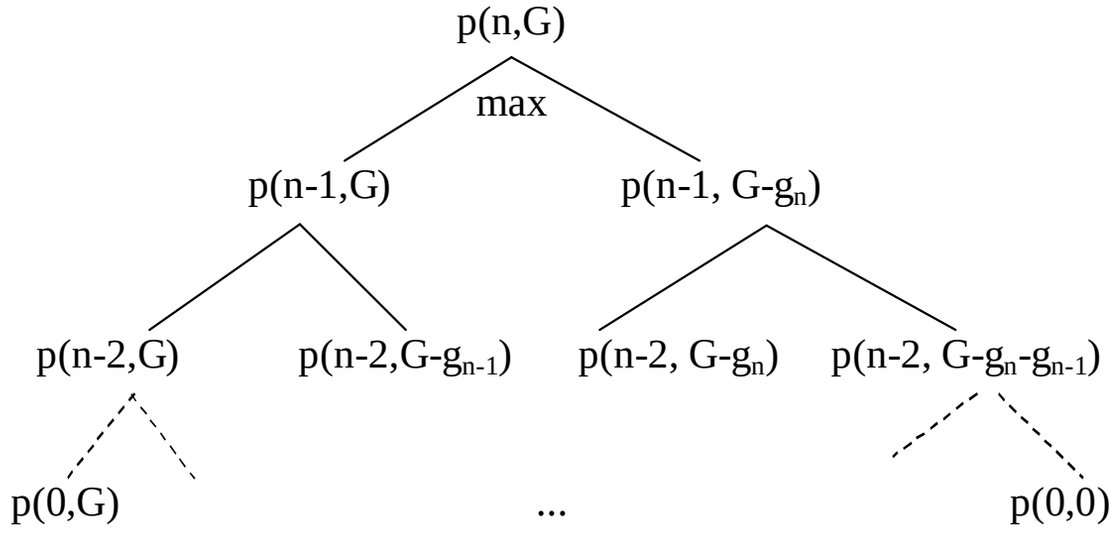
Gegeben:

- n Gegenstände mit einem Wert w_i und einem Gewicht g_i
- Rucksack, der maximal das Gewicht G fassen kann

(n, w_i, g_i, G aus $\mathbb{N}$)

Gesucht: Füllung des Rucksacks mit maximalem Wert

1. Rekursive Beschreibung des Problems:



$$p(0, g) = 0$$

$$p(i, g) = \begin{cases} \max\left(p(i-1, g), p(i-1, g-g_i) + w_i\right) & \text{wenn } g \geq g_i \\ p(i-1, g) & \text{sonst} \end{cases} \qquad i \neq 0$$

Mit p(i,g) := max. Wert für die Fassungskapazität g unter Berücksichtigung der ersten i Gegenstände

=> $O(2^n)$, aber: es gibt nur n*G verschiedene Teilprobleme!

2. Menge R der kleineren Probleme, auf die rekuriert wird:

hier: R = {(0,0), ..., (n-1,0), (1,0), ... , ..., (n-1, G), (n, 0), ... , (n, G-1)}

3. Aufrufreihenfolge der Probleme in R

hier: für i= 0,1, ..., n: für g=1, ..., G !! (n,G) wird zuletzt berechnet

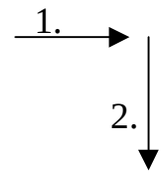
4a. Sukzessives Berechnen und Speichern von Lösungen in dieser Reihenfolge

Bsp. p(4,7) mit

i	**1**	**2**	**3**	**4**
$\mathbf{g_i}$	1	3	4	6
$\mathbf{w_i}$	1	3	2	2

p(i,g) =

i \ **g**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	*0*	0	0	0	0	0	0	0
1	*0*	1	1	1	1	1	1	1
2	0	1	1	*3*	4	4	4	4
3	0	1	1	3	4	4	4	*5*
4	0	1	1	3	4	4	4	*5*



=> O(n*G)

4b. Konstruktion des "Weges" von p(0,0) nach p(n,G)

- falls p(i-1,g) = p(i,g)
  Gegenstand i ist nicht in der Lösung, bei p(i-1,g) fortfahren
- sonst
  Gegenstand i ist in der Lösung, bei p(i-1,g-g_i) fortfahren

Anm.

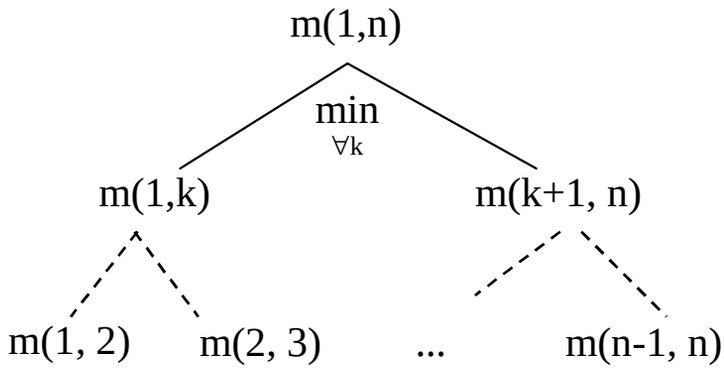
Der Index i wird so gewählt, dass die g_i aufsteigend sortiert sind.

Matrixmultiplikation

Gegeben: n Matrizen $M_1, \ldots, M_n$
mit den Dimensionen $p_0 \times p_1, \quad p_1 \times p_2, \quad \ldots, \quad p_{n-1} \times p_n$

Gesucht: Wie muss $M_1 \times \ldots \times M_n$ geklammert werden, so dass die Anzahl der erforderlichen Multiplikationen minimal ist (Beschränkung auf die Ausgabe der Anzahl der erforderlichen Multiplikationen)?

1. Rekursive Beschreibung des Problems:
Matrix i hat die Dimension $p_{i-1} \times p_i$
m(i,j) := minimale Anzahl an Multiplikationen für die Matrizen i bis j (i≠j)



$$m(i,j) = \begin{cases} \min(m(i,k) + m(k+1,j) + p_{i-1}p_k p_i \mid i < k < j-2) & \text{falls } j > i+1 \\ p_{i-1}p_i p_{i+1} & \text{sonst} \end{cases}$$

2. Menge R der kleineren Probleme

R = {(i,j) | 1≤ j-i < n-1 }

3. Aufrufreihenfolge

for abstand := 1 to n-1: for i:= 1 to n-abstand *j:=i+abstand*

4. Sukzessive Berechnung und Speicherung der Lösungen
m(i,j)=

i \ j	1	2	3	4
1	-			
2	-	-		
3	-	-	-	
4	-	-	-	-

Anm.
das jeweilige k für m(i,j) wird durch „Probieren“ ermittelt
i < k < j

=> $O(n^2)$

Weitere Beispiele

Longest Common Subsequence Problem

Gegeben: Zwei Folgen X und Y

Gesucht: Die längste Teilfolge, die – nicht notwendigerweise zusammenhängend – in beiden Folgen enthalten ist.

Editierdistanz

Gegeben: zwei Zeichenketten u, v mit den Operationen
Löschen eines Symbols aus einer Zeichenkette
Einfügen eines Symbols in eine Zeichenkette
Ersetzen eines Symbols in einer Zeichenkette

Gesucht: Editiersequenz s mit minimaler Editierdistanz D(u,v) = minimale Anzahl der obigen Operationen um die obige Zeichenkette u in die Zeichenkette v zu überführen

Wettbewerbsaufgaben

IOI95: Shop-Aufgabe
BWInf 13.1.3: Rätselecke
BWInf 5.2.3: Ähnlichkeit von Zeichenketten

Dynamische Programmierung im Vergleich zu Teile-und-Herrsche

- Gemeinsamkeit:

  Lösen eines Problems durch separates Lösen kleinerer Teilprobleme

- Unterschied:

  bei Teile-und-Herrsche sind die Teilproblem typischerweise nur halb so groß wie das Ausgangsproblem

  => typische Laufzeit O(n*log(n))

  - dies gilt bei Dynamischer Programmierung nicht, daher müssen Mehrfachberechnungen vermieden werden.