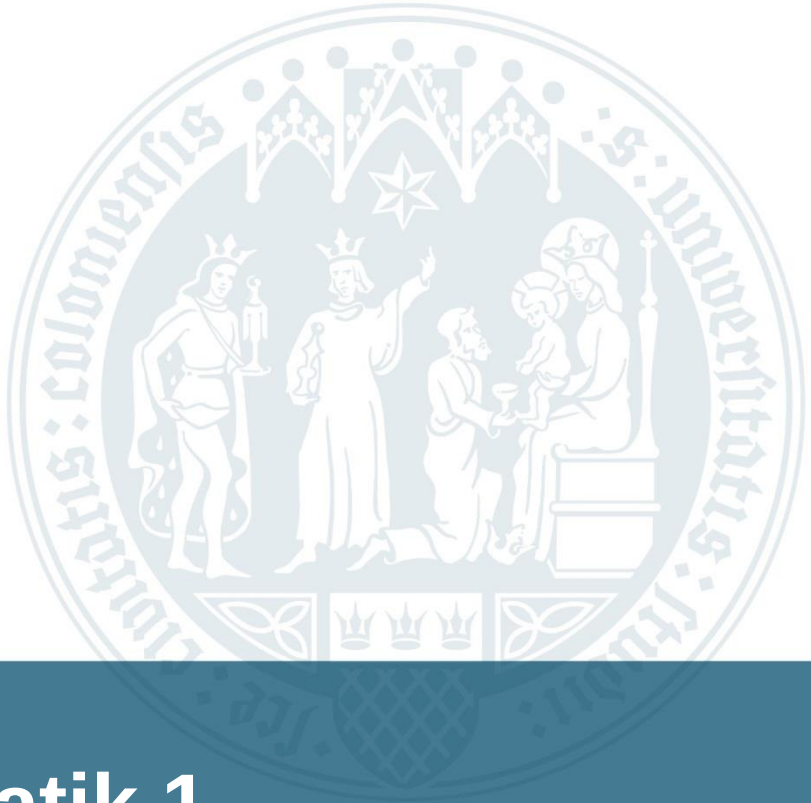
Universität zu Köln



Grundzüge der Informatik 1

Vorlesung 10

Überblick

Überblick

- Wiederholung
  - Laufzeit rekursive Berechnung der Fibonacci-Zahlen
  - Verbesserung durch Speichern der Lösungen
  - Iterative Lösung
  - Prinzip der dynamischen Programmierung
  - Ein erstes einfaches Beispiel
- Partition und SubsetSum
  - Problemdefinition
  - Erstellen der Rekursionsgleichung
  - Entwicklung des Algorithmus

Dynamische Programmierung

```
Fib2(n)
1.  if n=1 then return 1
2.  if n=2 then return 1
3.  return Fib2(n-1) + Fib2(n-2)
```

Fibonacci Zahlen

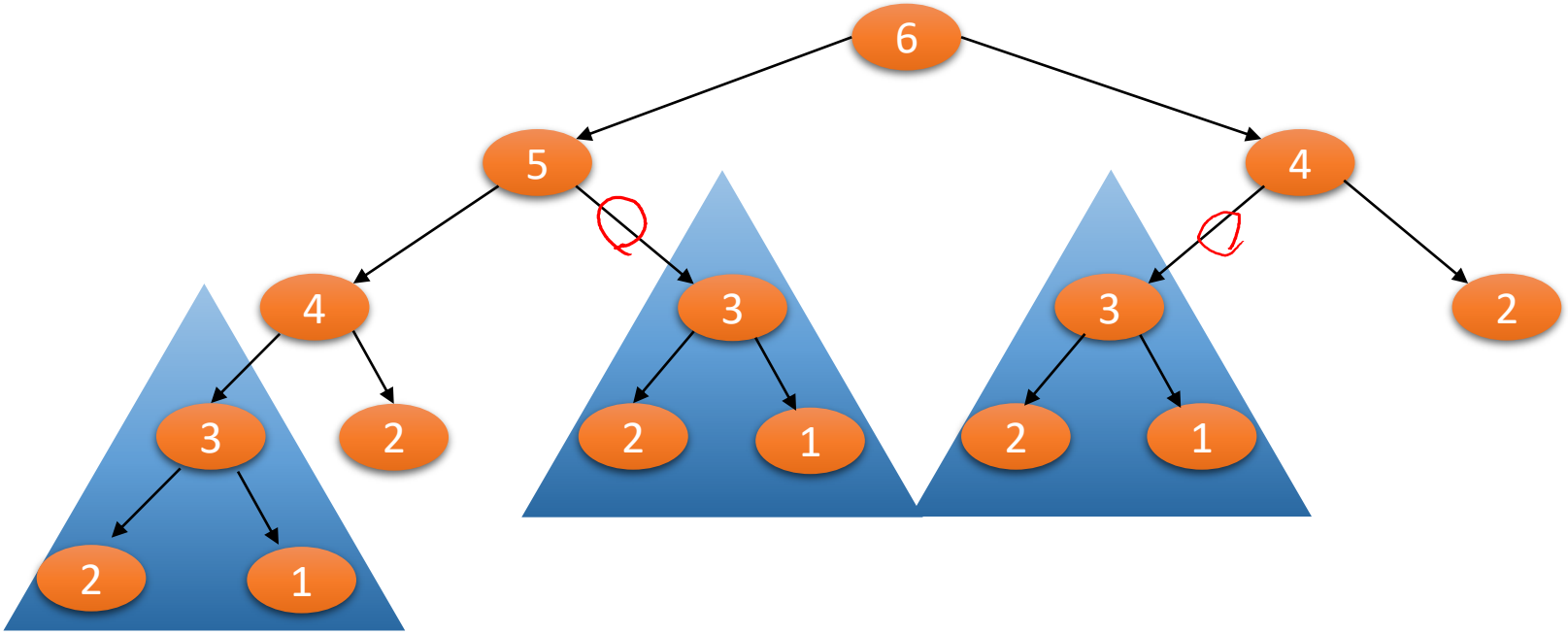
- Fib(1)=1
- Fib(2)=1
- Fib(n) = Fib(n-1) + Fib(n-2)

Dynamische Programmierung

Warum ist die Laufzeit so schlecht?

- Betrachte Rekursionsbaum von Fib2(6)

Bei der Berechnung von Fib2(6) wird Fib2(3) dreimal aufgerufen!



Dynamische Programmierung

Zwischenspeichern von Rechenergebnisse

- Idee: Wir speichern die Ergebnisse, die wir bereits kennen

Dynamische Programmierung

Beobachtung

- Die Tabelle wird bottom-up ausgefüllt

Vereinfachter Code

```
Fib1(n)
1.  F = new array[1…n]
2.  F[1] = 1
3.  F[2] = 1
4.  for i=3 to n do
5.     F[i] = F[i-1] + F[i-2]
6.  return F[n]
```

Dynamische Programmierung

Dynamische Programmierung

- Beschreibe optimale Lösung einer gegebenen Instanz durch optimale Lösungen „kleinerer“ Instanzen (hier kleinere Fibonacci-Zahlen)
- Beschreibe Rekursionsabbruch
- Löse die Rekursion „bottom-up“ durch schrittweises Ausfüllen einer Tabelle der benötigten Teillösungen

Wann verbessert der Ansatz die Laufzeit?

- Die Anzahl unterschiedlicher Funktionsaufrufe (Größe der Tabelle) ist klein
- Bei einer „normalen Ausführung“ des rekursiven Algorithmus ist mit vielen Mehrfachausführungen zu rechnen

Dynamische Programmierung

Entwicklung der Rekursionsgleichung

- Eingabe besteht aus n Elementen
- Idee: Ordne die Elemente von 1 bis n (das Eingabefeld A gibt z.B. eine solche Ordnung)
- Drücke optimale Lösung für die ersten i Elemente als Funktion der optimalen Lösung ersten i-1 Elemente aus

Beispiel

- Sei Max(i) = $\max_{1 \leq j \leq i}\{A[j]\}$
- Dann gilt Max(1) = A[1] (Rekursionsabbruch)
- Max(i) = max {Max(i-1), A[i]}

Dynamische Programmierung

MaxSucheDP(A,n)

1. Max = **new array** [1...n]
2. Max[1] = A[1]
3. **for** i=2 **to** n **do**
4. Max[i] = max{Max[i-1], A[i]}
5. **return** Max[n]

Dynamische Programmierung - Partition

Partition

- Gegeben: Menge M von natürlichen Zahlen
- Aufgabe: Entscheide, ob M in zwei Menge L und R aufgeteilt werden kann, so dass $\sum_{x \in L} x = \sum_{y \in R} y$ (Antwort ist *true* oder *false*)

Beispiel

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

Dynamische Programmierung - Partition

Partition

- Gegeben: Menge M von natürlichen Zahlen
- Aufgabe: Entscheide, ob M in zwei Menge L und R aufgeteilt werden kann, so dass $\sum_{x \in L} x = \sum_{y \in R} y$ (Antwort ist *true* oder *false*)

Beispiel

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

Dynamische Programmierung - Partition

Hintergrund

- Partition ist NP-vollständig
- Die Frage, ob man Partition in *polynomieller* Laufzeit lösen kann, ist äquivalent zur Frage ob P gleich NP ist
- Dies ist eines der wichtigsten offenen Probleme der Theoretischen Informatik
- Das Problem gehört zu den 7 Millennium Problems des Clay Mathematics Institute auf deren Lösung ein Preisgeld von je einer Million Dollar ausgesetzt ist

Dynamische Programmierung - Partition

Beobachtung

- Sei M eine Menge von natürlichen Zahlen
- Sei W die Summe der Zahlen aus M, d.h. W= $\sum_{x \in M} x$
- M kann genau dann in zwei Mengen L und R aufgeteilt werden kann, so dass $\sum_{x \in L} x = \sum_{y \in R} y$ gilt, wenn es eine Teilmenge L von M gibt mit $\sum_{x \in L} x = W/2$

Neue Frage

- Gibt es L$\subseteq$M mit $\sum_{x \in L} x = W/2$?
- Allgemeiner: Für gegebenes U, gibt es L$\subseteq$M mit $\sum_{x \in L} x = U$?

Dynamische Programmierung - SubsetSum

Allgemeinere Fragestellung (SubsetSum)

- Gibt es L⊆M mit $\sum_{x \in L} x = U$?

Herangehensweise

- Sei M ={ $x_1,\ldots,x_n$ } (wir definieren eine Reihenfolge der Elemente, z.B. Reihenfolge im Eingabefeld)
- Definiere Indikatorfunktion Ind(U,m) mit
- $$Ind(U,m) = \begin{cases} true\,, & \text{wenn } L \subseteq \{x_1,..,x_m\} \text{ gibt mit } \sum_{y \in L} y = U \\ false, & \text{sonst} \end{cases}$$
- Gesucht ist Rekursion für Ind(U,m)

Dynamische Programmierung - SubsetSum

Schritt 1 – Die Rekursion finden

- Wir starten mit unserem Beispiel 4, 7, 9, 10, 23, 13
- Sei $x_1 = 4$, $x_2 = 7$, $x_3 = 9$, $x_4 = 10$, $x_5 = 23$, $x_6 = 13$
- Wir wollen Partition lösen, d.h. wir wollen wissen, ob eine Teilmenge mit Summe 33 existiert
- also sei U = 33
- In unserem Fall gibt es eine Lösung, z.B. $L = \{x_1 , x_2 , x_3 , x_6\}$
- Da wir 6 Zahlen in M haben, setzen wir n=6
- Wir wollen also Ind(U,n) = Ind(33,6) bestimmen

Dynamische Programmierung - SubsetSum

Herangehensweise

- Wie können wir die Lösung rekursiv formulieren, indem wir auf Teillösungen zurückgreifen?
- Wir betrachten das letzte Element der Eingabe nach unserer Ordnung (also x_6)
- Wir wissen nicht, ob x_6 zu einer Lösung gehört
- Es gibt zwei Fälle:
  - x_6 gehört zu einer Lösung
  - x_6 gehört nicht zu einer Lösung
- Wir machen eine Fallunterscheidung

Dynamische Programmierung - SubsetSum

Fall 1:

- x_6 gehört zu einer Lösung L
- Dann können wir L schreiben als $L = \{x_6\} \cup (L \setminus \{x_6\}) = \{x_6\} \cup \{x_1, x_2, x_3\}$
- Da L eine Lösung ist, muss die Summe der Zahlen aus L gleich U = 33 sein
- Damit ist die Summe der Zahlen aus $L \setminus \{x_6\}$ gerade $U - x_6 = 33-13 = 20$ sein

Fazit

- Wir suchen also eine neue Teilmenge L‘ aus $\{x_1,..., x_5\}$ mit Summe U‘=20
- Wir müssen also nur noch die ersten 5 Elemente betrachten und auf diesen ein Teilproblem lösen

Dynamische Programmierung - SubsetSum

Fall 2:

- x_6 gehört nicht zu einer Lösung L
- Dann müssen wir x_6 nicht weiter betrachten
- Wir suchen also eine Teilmenge L‘ aus $\{x_1,.., x_5\}$ mit Summe U
- Wir müssen also nur noch die ersten 5 Elemente betrachten

Dynamische Programmierung - SubsetSum

Allgemeine Formulierung

- Sei M={ $x_1,\ldots,x_n$ } eine Menge mit n natürlichen Zahlen
- Sei L(U,n) eine Teilmenge von { $x_1,\ldots,x_n$ } mit Summe U
- Ist $x_n \in$ L(U,n), dann gilt L(U,n) = L(U-x_n, n-1) $\cup$ {x_n}
- Ist $x_n \notin$ L(U,n), dann gilt L(U,n) = L(U, n-1)

Formulierung der Rekursion

- Wir wissen also: Ist Ind(U,n) = *true*, dann ist entweder Ind(U,n-1) = *true* oder Ind(U-x_n, n-1) = *true*
- Umgekehrt gilt auch: Ist Ind(U,n-1) = *true* oder Ind(U-x_n, n-1) = *true* *dann ist* Ind(U,n) = *true*

Dynamische Programmierung - SubsetSum

Die Rekursion

- Wenn n>1, dann gilt:
- $$Ind(U,n) = \begin{cases} true, & \text{wenn } U \geq x_n \text{ und } Ind(U - x_n, n-1) = true \\ & \text{oder } Ind(U, n-1) = true \\ false, & \text{sonst} \end{cases}$$

Rekursionsabbruch:

- Wenn U>0 und n=1, dann gilt:
- $$Ind(U,1) = \begin{cases} true, & \text{wenn } x_1 = U \\ false, & \text{sonst} \end{cases}$$
- Wenn U=0 und n=1 dann gilt:
- $Ind(0,1) = true$

Dynamische Programmierung - SubsetSum

```
SubsetSum(A, U, n)
1.  Ind = new array [0..U] [1..n]
2.  for j=1 to n do
3.      Ind[j,1] = false
4.  Ind[0,1] = true                    \* leere Menge
5.  Ind[A[1],1] = true                 \* Menge {A[1]}
6.  for i=2 to n do
7.      for u=0 to U do
8.          Ind[u,i] = false
9.          if Ind[u,i-1] = true then Ind[u,i] = true
10.         if u≥A[i] und Ind[u-A[i], i-1] = true then Ind[u,i] = true
11. return Ind[U,n]
```

Dynamische Programmierung - SubsetSum

	U=0	1	2	3	4	5	6	7	8
i=1	✓	✓	–	–	–	–	–	–	–
2	✓	✓	✓	✓	–	–	–	–	–
3	✓	✓	✓	✓	✓	✓	✓	–	–
4	✓	✓	✓	✓	✓	✓	✓	✓	✓

- M= {1,2,3,5}

Dynamische Programmierung - SubsetSum

Laufzeitanalyse

- Die Laufzeit wird durch die zwei geschachtelten for-Schleifen dominiert und ist höchstens $O(1)\cdot n\cdot U$.

Erweiterung der O-Notation auf zwei Variablen

- $O(f(n,m)) = \{ g(n,m) \mid \exists\ n_0, m_0, c>0$ so dass für alle $n \geq n_0$, $m \geq m_0$ gilt, dass $g(n,m) \leq c \cdot f(n,m)\}$

Bemerkung

- Diese Definition kann in konstruierten Fällen zu nicht gewünschten Aussagen führen! (z.B. wenn $g(1,m) = m^2$ ist und $g(n,m)=m$ für $n>1$)

Dynamische Programmierung - SubsetSum

Satz 10.1

- Die Laufzeit von Algorithmus SubsetSum(A,U,n) ist O(nU).

Beweis

- Die Laufzeit ist offensichtlich O(nU).

Dynamische Programmierung - SubsetSum

Satz 10.2

- Algorithmus SubsetSum(A,n,U) löst das SubsetSum Problem.

Beweis

- Wir zeigen per Induktion über i die folgende Schleifeninvariante:
- Ind[u,i]= true, gdw. es eine Teilmenge der ersten i Zahlen aus A gibt, die sich zu u aufsummieren
- Induktionsanfang:
- Ind[0,1] wird in Zeile 4 auf true gesetzt.
- Ind[A[1],1] wird in Zeile 5 auf *true* gesetzt.
- Damit gilt Ind[u,1] = true, gdw. es eine Teilmenge der ersten Zahl gibt, die sich zu u aufsummiert (die leere Menge oder die Menge {A[1]})

Dynamische Programmierung - SubsetSum

Satz 10.2

- Algorithmus SubsetSum(A,n,U) löst das SubsetSum Problem.

Beweis

- Induktionsannahme:
- Ind[u,i-1]= true, gdw. es eine Teilmenge der ersten i-1 Zahlen aus A gibt, die sich zu u aufsummieren

Dynamische Programmierung - SubsetSum

Satz 10.2

- Algorithmus SubsetSum(A,n,U) löst das SubsetSum Problem.

Beweis

- Induktionsschluss:
- „<=“
- Gibt es eine Teilmenge von A[1..i], die sich zu u aufsummiert, so kann man u entweder als Summe einer Teilmenge von A[1..i-1] darstellen oder als Summe von A[i] vereinigt mit einer Teilmenge von A[1..i-1].
- Im ersten Fall folgt aus der Induktionsannahme dass Ind[u,i-1]=*true* ist und somit auch Ind[u,i]=*true*.
- Im zweiten Fall muss die Teilmenge von A[1..i-1] Summe u-A[i] haben. Nach Induktionsannahme ist dann aber Ind[u-A[i],i]=*true* und somit Ind[u,i] = *true*.

Dynamische Programmierung - SubsetSum

Satz 10.2

- Algorithmus SubsetSum(A,n,U) löst das SubsetSum Problem.

Beweis

- „=>“
- Ist Ind[u,i]=true, so war entweder Ind[u,i-1]=true oder Ind[u-A[i],i-1]=true.
- Nach Induktionsannahme kann man entweder u oder u-A[i] als Summe einer Teilmenge von A[1..i-1] darstellen.
- Somit kann man u als Summe einer Teilmenge von A[1..i] darstellen.

Dynamische Programmierung

Aufgabe

- Wir haben bisher nur das Entscheidungsproblem gelöst
- Wie kann man mit Hilfe der erstellten Tabelle die Teilmenge finden, die sich zu U aufsummiert?

Dynamische Programmierung - SubsetSum

Die Rekursion

- Wenn n>1, dann gilt:
- $$Ind(U,n) = \begin{cases} true, & \text{wenn } U \geq x_n \text{ und } Ind(U - x_n, n-1) = true \\ & \text{oder } Ind(U, n-1) = true \\ false, & \text{sonst} \end{cases}$$

Rekursionsabbruch:

- Wenn U>0 und n=1, dann gilt:
- $$Ind(U,1) = \begin{cases} true, & \text{wenn } x_1 = U \\ false, & \text{sonst} \end{cases}$$
- Wenn U=0 und n=1 dann gilt:
- $Ind(0,1) = true$

Dynamische Programmierung

Zusammenfassung

- Partition und SubsetSum
  - Problemdefinition
  - Erstellen der Rekursionsgleichung
  - Entwicklung des Algorithmus

Referenzen

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