Algorithmen und Komplexität

Dynamische Programmierung

Markus Ullrich Norbert Baum

Fachbereich Informatik - IIb07
Hochschule Zittau/Görlitz

28. Mai 2009

Wie sieht es mit langen Ketten aus?

$$\underset{(120x17)}{A_1} \cdot \underset{(17x56)}{A_2} \cdot \underset{(56x37)}{A_3} \cdots \underset{(110x112)}{A_n}$$

$$n>5$$

Wie sieht es mit langen Ketten aus?

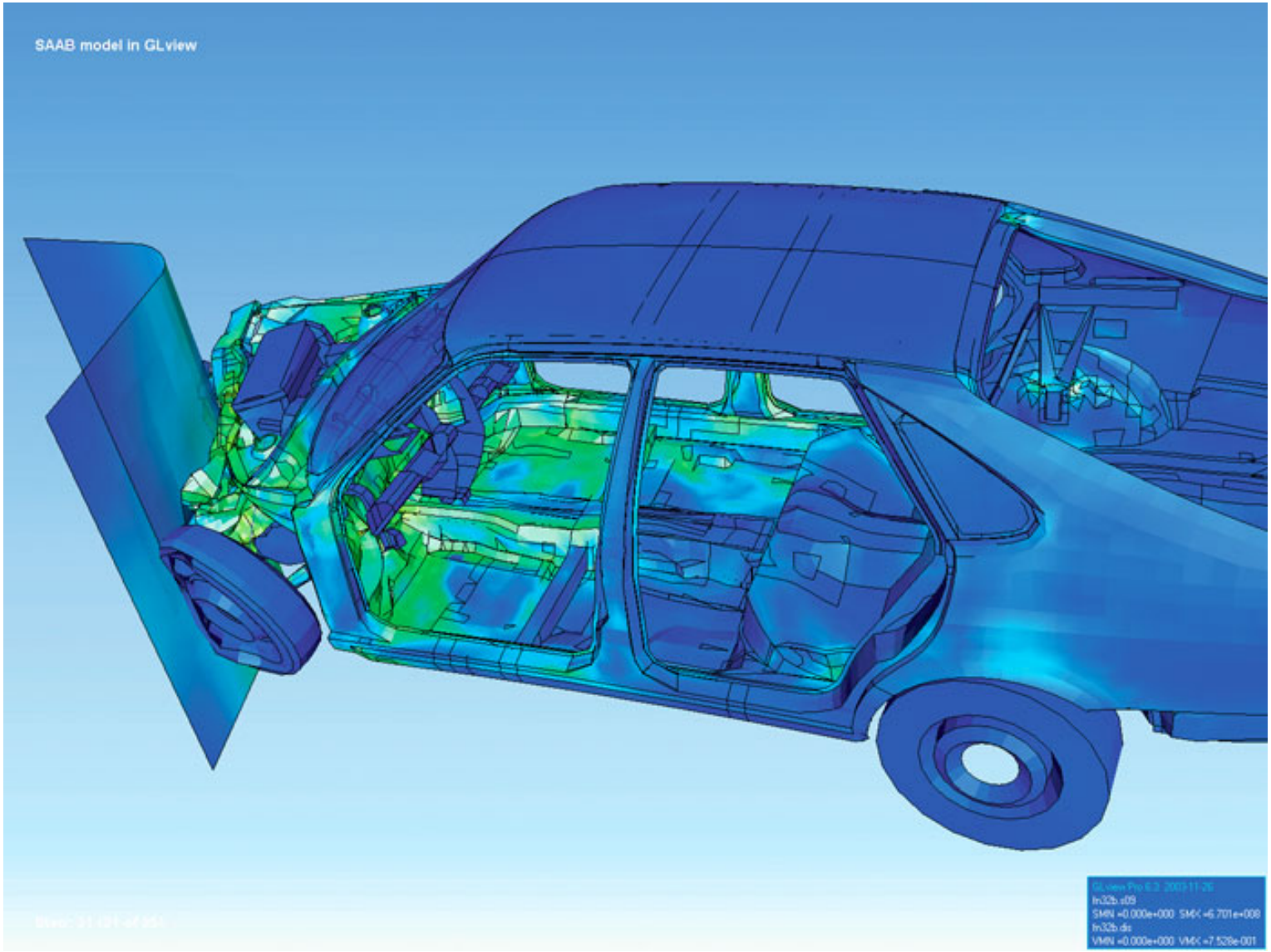
$$\underset{(120x17)}{A_1} \cdot \underset{(17x56)}{A_2} \cdot \underset{(56x37)}{A_3} \cdots \underset{(110x112)}{A_n}$$

$$n>5$$

Auftreten in der Praxis:

- Verflechtungsproblem
- Finite Elemente (z.B. im Maschinenbau)

Anwendung: Matrix-Ketten-Multiplikation



Quelle: http://www.dasautoblog.com/2007/06/interessantes_w.html

Bottom-up-Ansatz

Teile und Herrsche

- Top-down-Ansatz
- nur die nötigsten Teilprobleme werden berechnet
- doppelte Berechnungen möglich

Dynamisches Programmieren

- Bottom-up-Ansatz
- alle Teilprobleme werden berechnet
- eintragen bereits berechneter Werte in Tabelle

$\Rightarrow$ keine erneute Berechnung erforderlich

Schwierigkeiten

- selten unabhängige und überlappungsfreie Teilprobleme
- unausgewogene Teilprobleme
- nicht immer gilt die Optimalitätsbedingung
- Anzahl der zu lösenden Teilprobleme zu groß

Überlappende Teilprobleme

- immer wieder lösen der selben Teilprobleme
  = Überlappende Teilprobleme

$\Rightarrow$ Gesamtzahl der *verschiedenen* Teilprobleme muss möglichst klein sein

- durch speichern in Tabelle kann darin bei Bedarf nachgeschlagen werden
- Aber: Mehrfachberechnungen durch rekursive Lösung kann vorkommen

Iterativ oder rekursiv?

Ist die iterative Berechnung der rekursiven vorzuziehen?

Iterativ oder rekursiv?

Iterativ

Vorteile:

- keine doppelte Berechnung von Werten

Nachteile:

- Berechnung unnötiger Werte möglich
- rekursiven Ansatz transformieren
  - ist schwierig
  - und fehleranfällig

Rekursiv

Vorteile:

- keine unnötigen Werte werden berechnet
- Wiederverwendung des rekursiven Ansatzes

Nachteile:

- doppelte Berechnung von Werten möglich

Iterativ oder rekursiv?

Memoizing

- rekursive Berechnung mit Tabelle für Teillösungen
- Eintrag schon vorhanden? $\Rightarrow$ übernehmen
- sonst berechnen und eintragen

Allgemein gilt:

- lösen aller Teilprobleme erforderlich $\Rightarrow$ iterative Lösung um konstanten Faktor schneller (da konstant: vernachlässigbar)
- sonst: Rekursion mit memoizing

Rückkehr des Rucksackproblems

Rucksackproblem

- Optimierungsproblem der Kombinatorik
- Optimale Auswahl von n-Gegenständen mit dem Ziel des maximalen Werts und nicht Überschreitung des Gewichts
- Beladung von Schiffen, LKWs, Gepäckträgern in den Alpen...

Implementation

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

Implementation

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

Psudocode iterativ

```
FOR i := 1 TO Anzahl der gegenstaende STEP 1 DO
 Kopiere die zeileNeu auf zeileAlt
 gi := das Gewicht vom dem aktuellen Gegenstand(i)
 FOR g := 0 TO kapazitaet STEP 1 Do
  wenn g < gi dann
   zeileNeu[g] := zeileAlt[g]
  sonst
   temp := zeileAlt[g-gi] + Wert von gegenstaend[i]
   zeileNeu[g] := Max(temp, zeileAlt[g])
  wenn_ende
 next
next
```

Dimension der Tabelle & Aufwandsbetrachtung

Dimension der Tabelle

- baut sich aus den Gegenständen und der Kapazität zusammen
- Zeilen stellen Gegenstände dar
- Spalten die einzelnen Kapazitäten
- eine zusätzliche Spalte und Zeile für die Initialisierung

Auwandsbetrachtung

- scheinbar linearer Aufwand, wenn man K als Konstante auffasst
- jedoch hängt K mit n zusammen
- $\Rightarrow$ exponentieller Aufwand

Aufrufbeispiel

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

Kapazität = 5

Nr	Gewicht	Wert
1	4	1
2	1	4
3	3	2
4	2	3

Aufrufbeispiel

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

Kapazität = 5

Nr	Gewicht	Wert
1	4	1
2	1	4
3	3	2
4	2	3

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0					

Aufrufbeispiel

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

Kapazität = 5

Nr	Gewicht	Wert
1	4	1
2	1	4
3	3	2
4	2	3

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0				

Aufrufbeispiel

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

Kapazität = 5

Nr	Gewicht	Wert
1	4	1
2	1	4
3	3	2
4	2	3

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0		

Aufrufbeispiel

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

Kapazität = 5

Nr	Gewicht	Wert
1	4	1
2	1	4
3	3	2
4	2	3

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	1	

Aufrufbeispiel

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

Kapazität = 5

Nr	Gewicht	Wert
1	4	1
2	1	4
3	3	2
4	2	3

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	1	1

Aufrufbeispiel

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

Kapazität = 5

Nr	Gewicht	Wert
1	4	1
2	1	4
3	3	2
4	2	3

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	1	1
2	0					

Aufrufbeispiel

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

Kapazität = 5

Nr	Gewicht	Wert
1	4	1
2	1	4
3	3	2
4	2	3

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	1	1
2	0	4				

Aufrufbeispiel

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

Kapazität = 5

Nr	Gewicht	Wert
1	4	1
2	1	4
3	3	2
4	2	3

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	1	1
2	0	4	4	4		

Aufrufbeispiel

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

Kapazität = 5

Nr	Gewicht	Wert
1	4	1
2	1	4
3	3	2
4	2	3

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	1	1
2	0	4	4	4	4	

Aufrufbeispiel

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

Kapazität = 5

Nr	Gewicht	Wert
1	4	1
2	1	4
3	3	2
4	2	3

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	1	1
2	0	4	4	4	4	5

Aufrufbeispiel

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

Kapazität = 5

Nr	Gewicht	Wert
1	4	1
2	1	4
3	3	2
4	2	3

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	1	1
2	0	4	4	4	4	5
3	0					

Aufrufbeispiel

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

Kapazität = 5

Nr	Gewicht	Wert
1	4	1
2	1	4
3	3	2
4	2	3

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	1	1
2	0	4	4	4	4	5
3	0	4	4			

Aufrufbeispiel

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

Kapazität = 5

Nr	Gewicht	Wert
1	4	1
2	1	4
3	3	2
4	2	3

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	1	1
2	0	4	4	4	4	5
3	0	4	4	4		

Aufrufbeispiel

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

Kapazität = 5

Nr	Gewicht	Wert
1	4	1
2	1	4
3	3	2
4	2	3

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	1	1
2	0	4	4	4	4	5
3	0	4	4	4	6	

Aufrufbeispiel

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

Kapazität = 5

Nr	Gewicht	Wert
1	4	1
2	1	4
3	3	2
4	2	3

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	1	1
2	0	4	4	4	4	5
3	0	4	4	4	6	6

Aufrufbeispiel

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

Kapazität = 5

Nr	Gewicht	Wert
1	4	1
2	1	4
3	3	2
4	2	3

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	1	1
2	0	4	4	4	4	5
3	0	4	4	4	6	6
4	0					

Aufrufbeispiel

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

Kapazität = 5

Nr	Gewicht	Wert
1	4	1
2	1	4
3	3	2
4	2	3

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	1	1
2	0	4	4	4	4	5
3	0	4	4	4	6	6
4	0	4				

Aufrufbeispiel

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

Kapazität = 5

Nr	Gewicht	Wert
1	4	1
2	1	4
3	3	2
4	2	3

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	1	1
2	0	4	4	4	4	5
3	0	4	4	4	6	6
4	0	4	4			

Aufrufbeispiel

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

Kapazität = 5

Nr	Gewicht	Wert
1	4	1
2	1	4
3	3	2
4	2	3

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

Aufrufbeispiel

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

Kapazität = 5

Nr	Gewicht	Wert
1	4	1
2	1	4
3	3	2
4	2	3

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

ohne memoizing

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

ohne memoizing

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

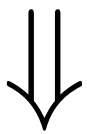
Pseudocode rekursiv

```
Prozedur: gesamtwertRekursiv
Parameter: int i, int kapazitaet
rekursiveAufrufe := rekursiveAufrufe+1
wenn i == 0 dann
  0
sonst
  wenn gi > kapazitaet dann
    gesamtwertRekursiv(i-1,kapazitaet)
  sonst
    max(gesamtwert(i-1,kapazitaet),
        gesamtWert(i-1,kapazitaet-gi)+wi)
  wenn_ende
wenn_ende
```

ohne memoizing

Aufrufbeispiel

```
gesamtwertRekursiv(120);
rekursiveAufrufe;
```



```
Ergebnis rekursiv ohne memoizing: 183
Rekursive Aufrufe: 2026
```

Feststellungen

- selbes Ergebnis wie iterativ
- erheblich mehr rekursive Aufrufe als Teilprobleme

mit memoizing

Pseudocode mit memoizing

```
Prozedur: gesamtWertMemo
Parameter: int i, int kapazitaet
wenn memoliste (i,kapazitaet) enthält dann
  memoliste(i,kapazitaet)
sonst
  memoliste hinzufügen ((i,kapazitaet),
            gesamtWertRekursiv(i,kapazitaet))
wenn_ende
```

mit memoizing

Aufrufbeispiel

```
gesamtwertRekursivMemo(120);
rekursiveAufrufe;
getLengthMemoListe();
```

1.Aufruf

```
Ergebnis rekursiv mit memoizing: 183
Rekursive Aufrufe: 502
Länge der Memoliste: 501
```

mit memoizing

Aufrufbeispiel

```
gesamtwertRekursivMemo(120);
rekursiveAufrufe;
getLengthMemoListe();
```

1.Aufruf

```
Ergebnis rekursiv mit memoizing: 183
Rekursive Aufrufe: 502
Länge der Memoliste: 501
```

2.Aufruf

```
Ergebnis rekursiv mit memoizing: 183
Rekursive Aufrufe: 1
Länge der Memoliste: 501
```

Fazit

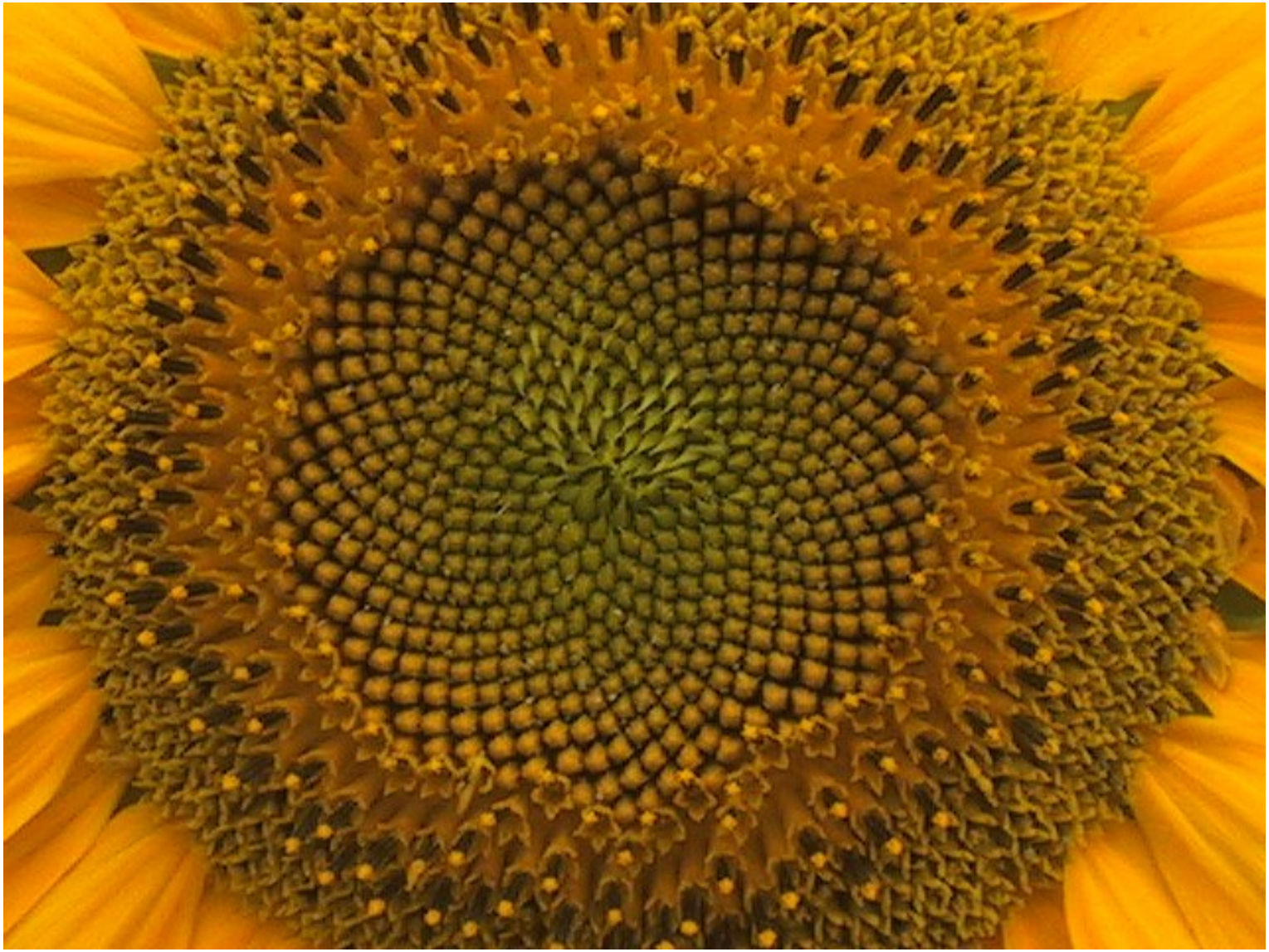
Vorteile Rekursion mit Memoizing

- mindestens so schnell wie iterativ
- keine Berechnung nicht benötigter Teillösungen
- keine fehleranfällige Transformation nötig

$$\Downarrow$$

rekursive Variante mit memoizing sollte bevorzugt werden

Weitere Anwendungen



Quelle: http://www.fotoeinsichten.de/sonnenblume/sonnenblume.jpg

Vorgehensweise beim Aufdecken

Allgemeines Muster

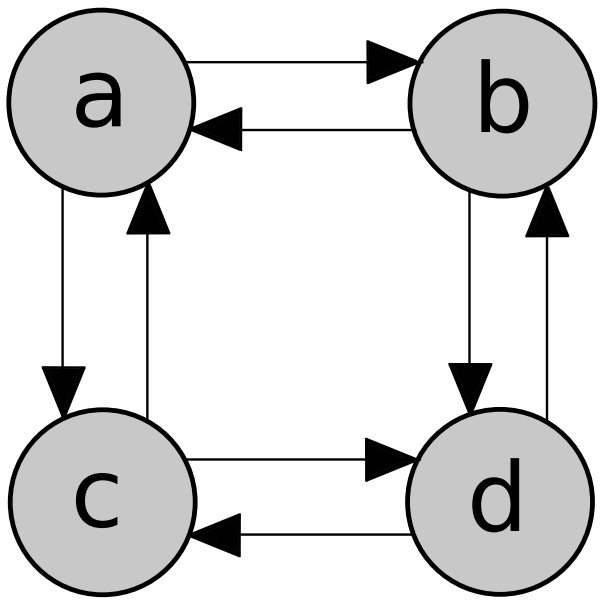
1. Problem auf Entscheidungen zurückführen
2. es wird angenommen, dass die richtigen Entscheidungen getroffen werden
3. darauß die Teilprobleme ableiten
4. Nachweis das Teilprobleme selbst optimal sein müssen

Faustregel - Menge der Teilprobleme möglichst klein halten

Beispiel: Ungewichteter einfacher Pfad

Gegeben:

- ein gerichteter Graph $G = (V, E)$
- die Knoten $u, v \in V$



Beispiel: Ungewichteter einfacher Pfad

Ungewichteter kürzester Pfad

Gesucht: kürzester Pfad $p = u \rightsquigarrow v$ mit $u \neq v$

- Einteilung in Teilpfade: $p_1 = u \rightsquigarrow w$ und $p_2 = w \rightsquigarrow v$
- Summe der Kanten in p_1 und p_2 = Summe der Kanten in p
- Nachweis: p_1 und p_2 müssen optimal sein

Beispiel: Ungewichteter einfacher Pfad

Ungewichteter längster Pfad

Gesucht: längster Pfad $p = u \rightsquigarrow v$ mit $u \neq v$

- p muss ein einfacher Pfad sein
- Vorgehen analog zum kürzesten Pfad
- Einteilung in Teilpfade: $p_1 = u \rightsquigarrow w$ und $p_2 = w \rightsquigarrow v$
- Aber: Teilprobleme sind nicht optimal

$\Rightarrow$ Es existiert keine optimale Teilstruktur für dieses Problem.

Naive Lösung

Naive Lösung

Anzahl der Klammerungen zählen

$$P(n) = \begin{cases} 1, & \text{wenn } n = 1 \\ \sum_{k=1}^{n-1} P(k)P(n-k), & \text{wenn } n \geq 2 \end{cases}$$

$\Rightarrow$ exponentieller Aufwand: 2^n

$\Rightarrow$ schlechte Lösung!

Lösen mit dynamischer Programmierung

Die Struktur einer optimalen Klammerung

- jede Matrix A_i besitzt die Dimension p_{i-1} x p_i
- Berechnung des Produktes $A_i A_{i+1} \cdots A_j = A_{i..j}$ mit $i \leq j$
- Gilt $i < j$, dann existiert ein k mit $i \leq k < j$ welches das Produkt $A_{i..j}$ zwischen A_k und A_{k+1} aufspaltet.

$\Rightarrow$ Kosten von $A_{i..k}$ + Kosten von $A_{k+1..j}$ = Kosten von $A_{i..j}$

$\Rightarrow$ ist $A_{i..j}$ optimal geklammert, so müssen auch $A_{i..k}$ und $A_{k+1..j}$ optimal geklammert sein

Lösen mit dynamischer Programmierung

Eine rekursive Lösung

- $m[i,j]$ = Minimum der skalaren Operationen zu Berechnung von $A_{i..j}$
- gilt $i = j$, sind keine Operationen notwendig $\Rightarrow m[i,i] = 0$
- für $i < j$, teilt optimale Klammerung das Produkt $A_{i..j}$ zwischen A_k und A_{k+1}
- Erinnerung: jede Matrix A_i besitzt die Dimension p_{i-1} x p_i

$\Rightarrow m[i,j] = m[i,k] + m[k+1,j] + p_{i-1} \cdot p_k \cdot p_j$

Lösen mit dynamischer Programmierung

Eine rekursive Lösung

$$m[i,j] = \begin{cases} 0, & \text{falls } i = j \\ \min\limits_{i \le k < j} \{m[i,j] = m[i,k] + m[k+1,j] + p_{i-1} \cdot p_k \cdot p_j\}, & \text{falls } i < j \end{cases}$$

Danke für ihre Aufmerksamkeit