Max Wisniewski , Alexander Steen

Tutor: Lena Schlipf

Aufgabe 1 Varianten der Vorlesungsbeispiele

(a) Betrachten Sie folgende Variation des Einkaufsproblems:
Zu einer Ware i existiert nun abgesehen vom Preis p_i und einem Wert w_i nun auch noch eine Häufigkeit h_i. Weiterhin haben wir ein Budget B und wollen als Ziel nun eine *Multmenge* von Artikeln finden, das bei B die Summe der Werte maximiert.
Zeigen Sie, dass diese Variante auch mit Laufzeit $O(nB)$ läuft.

Lösung:

Der Algorithmus ist in seiner Rekursionsgleichung davon ausgegangen, dass wir nach dem Kauf eines Artikels diesen nicht nocheinmal betrachten wollen und ist Aufgrund dieser Annahmen in jedem Schritt eine Zeile in der Tabelle nach oben gegangen (pro Zeilen hatten wir einen Artikel). Streichen wir das unbedingte nach oben gehen, erhalten wir die folgende Rekursionsvorschrift (Die Bezeichner sind, wie in der Vorlesung gewählt):

$$\begin{array}{lcrcl} \forall b \leq B & : & E[0,b] & = & 0 \\ \forall 0 \leq m \leq n & : & E[m,0] & = & 0 \\ \forall b < p[m] & : & E[m,b] & = & E[m-1,b] \\ \forall b \geq p[m] & : & E[m,b] & = & \max\{E[m-1,b], E[m,b-p[m]] + w[m]\} \end{array}$$

Der Unterschied hier besteht nur in der letzten Zeile. Wenn wir einen Artikel genommen haben, so können wir uns dafür entscheiden ihn erneut zu nehmen.

Was man an dieser Stelle nun gut sehen kann, ist dass wir nur auf ein anderes Feld im Array zugreifen ($E[m, b-p[m]]$ statt $E[m-1, b-p[m]]$). Der Algorithmus wird sich auch nur in diesem Zugriff unterscheiden.

Wir haben also wider nur $\Theta(n \cdot B)$ Zellen (Platzverbrauch) und pro Zelle müssen wir 2 Zahlen vergleichen und einen Wert addieren.
Dies macht eine Laufzeit von $\Theta(n \cdot B)$

(b) In der Vorlesung haben Sie gesehen, wie das Rundreiseproblem mit Hilfe von dynamischen Programieren gelöst werden kann. Arbeiten Sie die Details des Algorithmus aus und geben Sie Pseudecode an, um eine optimale Tour zu berechnen.

Lösung:

Aus der Vorlesung haben wir eine Menge $S \subseteq \{1, .., n-1\}$ Teilmenge von $m \in \{0, ..., n-1\} \setminus S$. $T[S, m]$ sind nun die kosten des optimalsten Weges, der in 0 anfängt, alle Städte aus S besucht und in m endet. Die Rekursion, mit der wir das Ergebnis finden wollen ist:

$$\begin{array}{rcl} \forall m \ : \ T[\emptyset, m] & = & d(0, m) \\ T[S, m] & = & \min\limits_{a \in S} T[S \setminus \{a\}, a] + d(a, m) \end{array}$$

Unsere erste Überlegung um den Algorithmus aufzuschreiben ist, dass wir alle Teilmenge von $\{1, ..., n-1\}$ aufstellen müssen. Bei jeder dieser Teilfolgen interessiert uns nur, ob das Element drin ist oder nicht. Wir verwenden also eine Bitmap, die sich am günstigesten über die Zahl 2^{n-1} darstellen. (Die 0 ist nicht Teil unserer Menge, dies hat zur Folge, dass wir im Algorithmus die Elemente immer um eins versetzt angucken müssen).

```
for m := 0 to n-1 do
  T[0,m] := 0;
for m := 0 to n-1 do
  for s := 1 to 2^{n-1} do
    minValue := ∞;
    for a:= 0 to n-1 do
      if s & 1 << (a-1) then
        minValue := min (minValue, T[s & ~(1 << (a-1)), a] +
            d(a,m));
    T[s, m] = minValue;

return T[2^{n-1}, 0];
```

Platzbedarf: Wir haben ein Feld der Dimension $n \times 2^{n-1}$, dies kostet uns also $\Theta(n \cdot 2^n)$ Platz.
Laufzeit: Pro Feld, müssen wir alle unsere Städte durchgehen und gucken, ob das Bit in unserer Zahl gesetzt ist. Das testen kostet uns konstante Laufzeit, das minimum bestimmen auch. Sollte die Bestimung der Distanz auch konstant sein, so haben wir pro Feld eine Laufzeit von $\Theta(n)$.
Insgesammt haben wir (was man auch an den 3 for Schleifen sehen könnte) eine Laufzeit von $\Theta(n^2 \cdot 2^n)$

Aufgabe 2 Münzwechseln

(a) Entwerfen Sie einen Algorithmus, der berechnet, auf weiviele Arten ein Euro (und allgemeiner n *Cent*) mit beliebig viel Münzen ≤ 1 Euromünze gewechselt werden kann. Die Lösung soll dynamische Programmierung verwenden und für beliebige Wärungen funktionieren.

Lösung:

(a) **Teillösungen finden:** Wie in der Aufgabe gegeben, nehmen als $C[n, i]$ die Anzahl der Möglichkeiten mit den i kleinsten Münzen n Cent darzustellen. Wir benötigen indes nicht, dass die Münzen wirklich sortiert sind.

(b) **Rekursion aufstellen:**

Als Rekursion nehmen wir eine, die der vom Einkaufsproblem aus Aufgabe 1 ähnelt:

$$\begin{array}{lcl} \forall i & : & C[0,i] = 0 \\ \forall n & : & C[n,0] = 0 \\ \forall n < w[i] & : & C[n,i] = C[n,i-1] \\ \forall n \geq w[i] & : & C[n,i] = C[n,i-1] + C[n-w[i],i] \end{array}$$

Erklärung:
Sollten wir kein Geld mehr wechseln müssen oder keine Münzen merh haben, gibt es 0 Möglichkeiten es zu wechseln.
Wenn wir weniger Geld haben, als die Münze wert ist, versuchen wir es mit der nächsten Münze. Da wir unsere Münzen vorher nicht sortieren, kann uns dieser Fall direckt bis zum C[n,0] Fall durchbubbeln.
Wenn dies alles nicht zutrifft. Gibt es auf einem Feld genaus 2 Möglichkeiten. Wir nehmen die Münze und ziehen den Geldbetrag ab oder wir nehmen diese Münzenart nie wieder. Diese beiden Möglichkeiten addieren wir einfach auf und erhalten so das korrekte Ergebnis.

Diese Löung zählt keine Möglichkeit doppelt. Wenn wir die j-te Münze vom Wert j nehmen wollen, gibt es für uns genau eine Möglichkeit dort hinzukommen und zwar, wenn wir vorher schon $j-1$ Münzen vom selben Wert genommen haben. (Der zweite Summand der Gleichung). Wir können zwar öfters auf das Feld kommen, aber nur über einen anderen Weg, d.h. eine andere Münzenkombination.

Geucht ist $C[N,I]$, wobei N das gesammte Geld ist, dass wir wechseln wollen und I die gesammten Münzen characterisiert.

(c) **Implementierung**

Beschreibe hier C wieder das Feld unserer Ergebnisse und w den Wert der Münzen. N der Betrag der gewechselt werden soll und I die Anzahl der Münzen.

```
for n := 0 to N do
  C[n,0] := 0;
for i:= 0 to I do
  C[0,i] := 0;
for n := 1 to N do
  for i := 1 to I do
    C[n,i] = C[n,i-1];
    if n >= w[i] then
      C[n,i] = C[n,i] + C[n-w[i],i];
return C[N,I];
```

(b) Analysieren Sie die Laufzeit Ihres Algorithmus.

Lösung:

Unser Algorithmus benötigt keine Vorverarbeitungszeit. Wir benötigen also keine auf - oder absteigend sortierten Münzewerte.
Platzbedarf: Das Feld hat die Größe $(N+1) \times (I+1) \Rightarrow \Theta(N \cdot I)$
Laufzeit: Pro Feld haben wir eine Addition und einen Vergleich (wahlweise noch 2 Zuweisungen). Dies ist pro Feld eine Laufzeit von $\Theta(1)$. Macht eine Gesammtlaufzeit von $\Theta(N \cdot I)$

(c) Implementieren Sie Ihren Algorithmus in *Java* und wenden sie ihn auf das Problem in *a)* an, sowie 1 $ in 1-, 5-, 10- und 25- Centstücke an.

Lösung:

Nicht bearbeitet.

Aufgabe 3 Versteckte Markov-Modelle

(a) Betrachten Sie das folgende Markov-Modell.

- Zustände: $Q = \{q, r, s\}$
- Alphabet: $\Sigma = \{a, b\}$
- Anfangsverteilung: $\{q : 0.1, r : 0.4, s : 0.5\}$
- Ausgabeverteilung für q $\{a : 0.2, b : 0.8\}$
- Ausgabeverteilung für r $\{a : 0.7, b : 0.3\}$
- Ausgabeverteilung für s $\{a : 0.5, b : 0.5\}$
- Übergangsverteilung für q: $\{q : 0.8, r : 0.1, s : 0.1\}$
- Übergangsverteilung für r: $\{q : 0.3, r : 0.3, s : 0.4\}$
- Übergangsverteilung für s: $\{q : 0.2, r : 0.4, s : 0.4\}$

Benutzen Sie den Viterbi-Algorithmus, um die wahrscheinlichste Erklärung für die Ausgabefolge *abba* zu ermitteln.

Lösung:

Wir stellen ein Tabelle, wie in der Vorlseung auf (wir zeigen hier immer nur auf, von wo der beste Wert kam und wie hoch seine Wahrscheinlichkeit ist):

	q	r	s
a	$0.1 * 0.2 = 0.02$	$0.7 * 0.7 = 0.49$	$0.5 * 0.5 = 0.25$
b	r : $0.49 * 0.5 * 0.8 = 0.1176$	r : $0.49 * 0.3 * 0.3 = 0.00441$	r : $0.49 * 0.4 * 0.5 = 0.098$
b	q : 0.075264	s : 0.01176	s : 0.0196
a	q : 0.01204224	s : 0.005488	q : 0.0037632

In der letzten Zeile nehmen wir wieder das Maximum. Das macht als wahrscheinlichste Folge für die Ausgabe *abba rqqq*.

(b) Bei einer Implementierung des Viterbi-Algorithmus rechnet man oft mit den Werten $\log p_i$ statt den Wahrscheinlichkeiten p_i. Erklären Sie, warum das eine gute Idee ist.

Lösung:
Zunächst sollte man einmal Betrachten, ob das Ergebnis, dass man erhält, immer noch das selbe ist.
Dies ist aber gegeben, da, wie in Mafi gehört, der Logarithmus *monoton steigt*. Dies bedeutet für uns, dass :

$$\forall p_1, p_2 \in \mathbb{R} \ : \ p_1 \leq p_2 \Rightarrow \log p_1 \leq \log p_2.$$

Als nächstes bleibt noch offen, warum man das ganze tun sollte.

Betrachten wir einmal die folgende Logarithmus-Rechenregel:

$$\forall p_0, ..., p_n \in \mathbb{R} \ : \ \log(p_0 \cdot ... \cdot p_n) = \log p_0 + \log p_1 + ... + \log p_n$$

Wir können also, indem wir den Logarithmus unserer Werte betrachten ein neues Teilergebnis bekommen, indem wir die Werte aufaddieren. Da die Addition in den meisten Fällen schneller geht, bietet es sich an.

Wir können also die Logarithmen der Wahrscheinlichkeiten addieren und am Ende wird immer noch der Größte Wert die Wahrscheinlichste Möglichkeit sein.

(c) Eine Anwendung von versteckten Markov-Modellen ist die Fehlerkorrektur. Beschreiben Sie, wie man mit einem *versteckten Markov-Modell* eine Übertragung eines Satzes deutscher Sprache über eine Leitung die zu 10% gestört ist, überträgt, modeliert.

Lösung:

Die versteckten Zustände sind das Alphabet $\Sigma = \{a, ..., z, \ ' \ '\}$. Nun haben wir eine gestörte Ausgabe, aber das was wir bekommen ist immer noch das Alphabet Σ. Die beiden müssen in der Beobachtung nun nicht übereinstimmen, aber es gilt auf jedenfall $Q = \Sigma$.

Nun benötigen wir nur noch Ausgabewahrscheinlichkeit, Start- und Übergangswahrscheinlichkeit.

Die Ausgabewahrscheinlichkeit wurde uns über den Kanal gegeben. Die Abbildung auf das richtige Zeichen ist immer 0.9. Die restlichen Zeichen nehmen wir gleichverteilt an. Sollte man nähere Informationen über den Kanal haben, könnte man hier auch noch weiter differenzieren.

Die Start- und Übergangswahrscheinlichkeit könnten wir exeprimentell bestimmen. Indem wir z.B. 10 Millliarden deutsche Texte nehmen und betrachten, mit welchem Buchstaben Sätze beginnen und wie die Wahrscheinlichkeit von Zeichen zu Zeichen ist.

Dabei könnte man Ergebnisse bekommen, wie:
Die größte Wahrscheinlichkeit für einen Anfang ist 'd', weil viele Sätze mit 'der', 'die, 'das' anfangen.
'e' ist der Häufigste Buchstabe im deutschen Alphabet, daher wird er in vielen Fällen eine leicht erhöhte Wahrscheinlichkeit besitzen.
Doppelte Buchstaben sind in den meisten Fällen nicht sehr wahrscheinlich. Insbesondere Kombinationen wie yy kommt selten vor.