Informatik I Übung, Woche 47

Giuseppe Accaputo

19. November, 2015

Plan für heute

1. Lernziele für die heutige Übungsstunde
2. Vorbesprechung Übung 10
3. Nachbesprechung Übung 9

Lernziele

- Dynamische Programmierung

Rucksackproblem

Rucksackproblem: Unser Rucksack kann nur 20 kg tragen, wir wollen jedoch den Wert der Ware, welche wir mitnehmen können maximieren.

Rucksackproblem: Abbruchbedingung

Frage: Wie sieht die Abbruchbedingung aus?

Rucksackproblem: Abbruchbedingung

Frage: Wie sieht die Abbruchbedingung aus?

Antwort: Wenn keine Objekte mehr vorhanden sind zum einpacken (max. Wert ist dann 0)

Rucksackproblem: Welche Fälle müssen wir unterscheiden?

Frage: Welche Möglichkeiten gibt es, um den max. Wert zu berechnen?

Rucksackproblem: Welche Fälle müssen wir unterscheiden?

Antwort: Sei o_t das letzte Objekt in der Liste.

1. Abbruchbedingung: Wenn keine Objekte mehr vorhanden sind zum einpacken, dann ist der max. Wert 0
2. Sonst: Ist das Gewicht von o_t grösser als das max. Gewicht, dann ignorieren wir o_t und suchen mit den restlichen Objekten nach einem maximalen Wert
3. Sonst:
   - Wir packen o_t nicht ein und suchen mit den restlichen Objekten nach einem maximalen Wert
   - Wir packen o_t ein. Der max. Wert berechnet sich nun aus dem Wert von o_t plus dem max. Wert den wir erhalten, wenn wir die restlichen Objekte noch in Betracht ziehen (neues Gewicht)
   - Der finale max. Wert ist der grössere der beiden Werte von oben.

Laufzeit der Implementation des Rucksackproblems

- Laufzeit $= 2^n$

Die Laufzeit kann erheblich verkürzt werden, wenn wir sämtliche vorherige Lösungen kennen $\Longrightarrow$ dynamische Programmierung

Dynamische Programmierung

Strategie: Finde Lösungen für kleinere Probleme und konstruiere daraus sukzessiv Lösungen für immer grössere Probleme, bis das eigentliche Gesamtproblem gelöst ist

Rucksackproblem und dynamische Programmierung

Für die dynamische Programmierung verwenden wir eine Tabelle und setzen diese wie folgt auf:

- **Zeile**: Objekte, welche in Betracht gezogen werden
- **Spalte**: max. Gewicht des Rucksacks
- **Eintrag** `max(i,j)`: Maximaler Wert, der entsteht wenn wir alle Objekte in Zeile `i` in Betracht ziehen und das max. Gewicht in Spalte `j` verwenden

Rucksackproblem und dynamische Programmierung

- Anzahl Objekte: 10
- Max. Gewicht: 20

	0	1	2	...	19	20
0						
1						
2						
...						
9						
10						

- Wichtig: Dimension der Tabelle ist $(10+1) \times (20+1)$, da die Abbruchbedingung miteinbezogen wird in der Tabelle (0 Objekte haben keinen Wert)!

Rucksackproblem und dynamische Programmierung

Frage: Wo in der Tabelle befindet sich der maximale Wert, der entsteht wenn wir alle Objekte in Betracht ziehen und das max. Gewicht verwenden?

Rucksackproblem und dynamische Programmierung

Frage: Wo in der Tabelle befindet sich der maximale Wert, der entsteht wenn wir alle Objekte in Betracht ziehen und das max. Gewicht verwenden?

Antwort: In der Zelle `max(n,m)`, wobei `n` der Anzahl Objekte entspricht welche wir in Betracht ziehen und `m` das max. Gewicht ist.

	0	1	2	...	19	20
0						
1						
2						
...						
10						

Rucksackproblem und dynamische Programmierung

Sei o_t das letzte Objekt in der Liste.

1. Abbruchbedingung: Wenn keine Objekte mehr vorhanden sind zum einpacken, dann ist der max. Wert 0
2. Sonst: Ist das Gewicht von o_t grösser als das max. Gewicht, dann ignorieren wir o_t und suchen mit den restlichen Objekten nach einem maximalen Wert
3. Sonst:
   - Wir packen o_t nicht ein und suchen mit den restlichen Objekten nach einem maximalen Wert
   - Wir packen o_t ein. Der max. Wert berechnet sich nun aus dem Wert von o_t plus dem max. Wert den wir erhalten, wenn wir die restlichen Objekte noch in Betracht ziehen (neues Gewicht)
   - Der finale max. Wert ist der grössere der beiden Werte von oben.

Rucksackproblem und dynamische Programmierung: Beispiel

Beispiel: Wir wollen den Eintrag `max(2,2)` berechnen, d.h. wir haben 2 Objekte (`i = 2`), wobei das letzte Objekt 1 kg schwer ist und 1.- kostet und das max. Gewicht beträgt 2 kg (`j = 2`)

	0	1	2	...	19	20
0	0	0	0	0	0	0
1	0	269	269	269	269	269
2	0	269	?			
...						

Rucksackproblem und dynamische Programmierung: Beispiel

Da das letzte Objekt o_t nur 1 kg wiegt geraten wir in den 3. Fall: Gewicht von o_t ist kleiner oder gleich dem max. Gewicht (siehe Folie 15). Wir benötigen also zwei Werte und speichern in `max(2,2)` den grösseren Wert der beiden:

1. max. Wert der sich berechnet, wenn wir o_t nicht einpacken (gelbe Zelle)
2. max. Wert der sich berechnet, wenn wir o_t einpacken: Wert von o_t plus der Wert, den wir erhalten wenn wir die restlichen Objekte in Betracht ziehen (neues Gewicht; grüne Zelle)

Rucksackproblem und dynamische Programmierung: Beispiel

1. max. Wert der sich berechnet, wenn wir o_t nicht einpacken (gelbe Zelle) $= 269$
2. max. Wert der sich berechnet, wenn wir o_t einpacken: Wert von o_t plus der Wert, den wir erhalten wenn wir die restlichen Objekte in Betracht ziehen (neues Gewicht; grüne Zelle) $= 269 + 1 = 270$

$\Longrightarrow$ `max(2,2):=` $\max\{269, 270\} = 270$

	0	1	2	...	19	20
0	0	0	0	0	0	0
1	0	269	269	269	269	269
2	0	269	270	...		

Wechselgeld

Wechselgeld: Auf wieviele Arten kann man einen bestimmten Geldbetrag bezahlen.

Wechselgeld: Abbruchbedingung

Wenn keine Münzen mehr vorhanden sind, dann kann hat man genau...

1. ...1 Möglichkeit um den Betrag 0 darzustellen
2. ...0 Möglichkeiten um einen Betrag > 0 darzustellen.

Wechselgeld: Welche Fälle müssen wir unterscheiden?

Frage: Welche Möglichkeiten gibt es, um die Anzahl Möglichkeiten ein Betrag mit gegebenen Münzen zu berechnen?

Wechselgeld: Welche Fälle müssen wir unterscheiden?

Antwort: Sei m_t die grösste Münze in der Liste.

1. Abbruchbedingung: Wenn keine Münzen vorhanden sind, dann hat man 0 Möglichkeiten um einen Betrag > 0 darzustellen, jedoch genau 1 Möglichkeit um den Betrag 0 darzustellen
2. Fall 1: Man verwendet m_t nicht, und berechnet, auf wie viele Arten man den Betrag x nur mit den Münzen $(m_1, m_2, \ldots, m_{t-1})$ darstellen kann
3. Fall 2: Man verwendet m_t mindestens einmal, und berechnet auf wie viele Arten man den Betrag $x - m_t$ mit allen Münzen $(m_1, m_2, \ldots, m_t)$ darstellen kann

Wechselgeld und dynamische Programmierung

Für die dynamische Programmierung verwenden wir eine Tabelle und setzen diese wie folgt auf:

- **Zeile**: Münzen, welche in Betracht gezogen werden
- **Spalte**: Betrag
- **Eintrag** `comb(i,j)`: Anzahl Möglichkeiten um den Betrag in Spalte `j` mit den gegebenen Münzen in Zeile `i` darzustellen