Informatik I Übung, Woche 47: Nachtrag zur dynamischen Programmierung

Giuseppe Accaputo

20. November, 2015

Rucksackproblem: Definition

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

Rucksackproblem: rekursiver Algorithmus

Sei n die Anzahl verfügbare Objekte.

1. $n = 0$: der maximale Wert ist 0
2. $n > 0$: Betrachte n-tes Objekt O_n:

   2.1 Lass O_n liegen:

   ```
   Wert1 := MaxWert(n-1, MaxGew)
   ```

   2.2 Pack O_n ein (falls `Gew(O_n)< MaxGew`):

   ```
   Wert2 := Wert(O_n) + MaxWert(n-1, MaxGew
       ↪ - Gew(O_n))
   ```

   - `MaxGew - Gew(O_n)` weil wir O_n einpacken

   2.3 `MaxWert` ist der grössere der beiden oben berechneten Werte:

   ```
   MaxWert := max(Wert1, Wert2)
   ```

Dynamischer Rucksack: Setup

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

- **Zeile**: Objekte, welche in Betracht gezogen werden
- **Spalte**: max. Gewicht des Rucksacks
- **Eintrag** `MaxGewDP(i,j)` enthält den maximalen Wert, der entsteht wenn wir alle Objekte $O_1, O_2, \dots, O_i$ (Zeile `i`) in Betracht ziehen und das max. Gewicht W_j (Spalte `j`) verwenden

Dynamischer Rucksack: Tabelle `MaxGewDP`

- Anzahl Objekte: 9, d.h. $O_1, O_2, \ldots, O_9$
- Max. Gewicht: 20

`MaxGewDP :=`

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

- Wichtig: Dimension der Tabelle ist $(9+1) \times (20+1)$ wegen der Abbruchbedingung

Dynamischer Rucksack: Objekte

- Die Objekte (siehe z.B. Vorlesungsslide 8-2) sind in einem Array `objects` gespeichert, das bspw. wie folgt definiert sein kann:

```
objects : ARRAY[0..8] OF TObject;
```

- `objects` hat die Länge n (Im Falle von Slide 8-2 wäre $n = 9$)
- Beachte: O_i befindet sich an der Position $(i-1)$ im Array (mit Indizes $0, \ldots, 8$), d.h. `o[i-1] :=` O_i
  - `objects[0] :=` O_1
  - `objects[1] :=` O_2
  - ...
  - `objects[8] :=` O_9

Dynamischer Rucksack: Objekte

- `objects[i-1] :=` O_i wird verwendet, weil die Zeilen (Objekte, welche in Betracht gezogen werden) in der Tabelle von $i = 0, 1 \ldots, 9$ gehen, wobei $i = 0$ der Abbruchbedingung entspricht, also der Fall in welchem 0 Objekte verwendet werden
- Ab $i = 1$ wird ein Objekt verwendet, d.h. wir müssen auf's Array zugreifen. Da die Indizes im Array von 0 bis 8 gehen, müssen wir auf das erste Objekt O_1 mittels `objects[1-1] = objects[0]` zugreifen.

Dynamischer Rucksack: Algorithmus

- Für alle max. Gewichte $W_0, W_1 \ldots, W_j, \ldots, W_{20}$ (Spalten) und für alle Objekte $O_1, \ldots, O_i, \ldots, O_9$ (Zeilen) tu folgendes:
  1. Falls $W_j = 0$:

     ```
     MaxGewDP[i,j] := 0;
     ```

  2. Sonst falls $\text{Gewicht}(O_i) > W_j$ (O_i ist zu schwer) dann:

     ```
     MaxGewDP[i,j] := MaxGewDP[i-1,j];
     ```

  3. Ansonsten (O_i hat Platz im Rucksack):
     3.1 Packe O_i nicht ein:

     ```
     W1 := MaxGewDP[i-1, j];
     ```

     3.2 Packe O_i ein:

     ```
     W2 := Wert(objects[i-1]) + MaxGewDP[i-1, j
         ↪  - Gewicht(objects[i-1])]
     ```

     - Verwende den grösseren Wert (Wertmaximierung):

     ```
     MaxGewDP[i,j] := max(W1, W2);
     ```

Dynamischer Rucksack: Step by Step

- **Recall**: Eintrag `MaxGewDP(i,j)` enthält den maximalen Wert, der entsteht wenn wir alle Objekte $O_1, O_2, \ldots, O_i$ (Zeile `i`) in Betracht ziehen und das max. Gewicht W_j (Spalte `j`) verwenden
- Es werden nicht unbedingt alle Objekte $O_1, O_2, \ldots, O_i$ eingepackt, da es darunter Objekte geben kann, die schwerer sind als das max. Gewicht W_j erlaubt

Dynamischer Rucksack: Step by Step, Fall 1

1. Falls $W_j = 0$:

```
MaxGewDP[i,j] := 0;
```

Erklärung: Das aktuelle max. Gewicht ist $W_j = 0$, d.h. es können keine Objekte eingepackt werden, also ist der max. Wert der Ware im Rucksack gleich 0.

Dynamischer Rucksack: Step by Step, Fall 2

2. Sonst falls $\mathrm{Gewicht}(O_i) > W_j$ (O_i ist zu schwer) dann:

```
MaxGewDP[i,j] := MaxGewDP[i-1,j];
```

Erklärung: In diesem Fall ist das Objekt O_i zu schwer und kann daher nicht eingepackt werden; der max. Wert im Rucksack bleibt genau gleich wie im Fall, in welchem O_i erst gar nicht in Betracht gezogen wurde. Dazu speichern wir den max. Wert aus Zelle `MaxGewDP[i-1,j]` ab; `MaxGewDP[i-1,j]` enthält nämlich den max. Wert der entsteht, wenn wir alle Objekte $O_1, O_2, \ldots, O_{i-1}$ (ohne O_i) in Betracht ziehen und diese in den Rucksack mit max. Gewicht W_j versuchen einzupacken.

Dynamischer Rucksack: Step by Step, Fall 3.1

3. Ansonsten (O_i hat Platz im Rucksack):
   3.1 Packe O_i nicht ein:

```
W1 := MaxGewDP[i-1, j];
```

Erklärung: Wir packen O_i nicht ein, d.h. wir verwenden wie vorhin den max. Wert aus Zelle `MaxGewDP[i-1,j]` ab; `MaxGewDP[i-1,j]` enthält nämlich den max. Wert der entsteht, wenn wir alle Objekte $O_1, O_2, \ldots, O_{i-1}$ (ohne O_i) in Betracht ziehen und diese in den Rucksack mit max. Gewicht W_j versuchen einzupacken.

Dynamischer Rucksack: Step by Step, Fall 3.2

3. Ansonsten (O_i hat Platz im Rucksack):
   3.1 ...
   3.2 Packe O_i ein:

```
W2 := Wert(objects[i-1]) + MaxGewDP[i-1,
    ↪ j - Gewicht(objects[i-1])]
```

Erklärung: O_i wird eingepackt, d.h. der max. Wert berechnet sich nun aus der Summe bestehend aus dem Wert von O_i und dem max. Wert der entsteht, wenn wir alle Objekte $O_1, O_2, \ldots, O_{i-1}$ (ohne O_i) in Betracht ziehen und diese in den Rucksack mit max. Gewicht $W_j - \mathrm{Gewicht(O_i)}$ versuchen einzupacken. Hierbei verwenden wir $W_j - \mathrm{Gewicht(O_i)}$ weil wir O_i einpacken und deshalb dessen Gewicht in Betracht ziehen müssen.

Dynamischer Rucksack: Step by Step, Ergebnis aus Fall 3.1 und 3.2

3. Ansonsten (O_i hat Platz im Rucksack):
   3.1 ...
   3.2 ...

   - Verwende den grösseren Wert (Wertmaximierung):

```
MaxGewDP[i,j] := max(W1, W2);
```

Erklärung: Nachdem wir die Fälle 3.1 und 3.2 in Betracht gezogen haben, speichern wir den grösseren Wert zwischen `W1` und `W2` in der Zelle `MaxGewDP[i,j]` ab, da es unser Ziel ist, einen maximalen Wert im Rucksack mit max. Gewicht W_j und den Objekten $O_1, O_2, \ldots, O_i$ zu erreichen.

Dynamischer Rucksack: Beispielberechnung

- Nehmen wir an, die Tabelle `MaxGewDP` sieht aktuell wie folgt aus und wir möchten nun `MaxGewDP[2,2]` (rot) berechnen:

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

- `MaxGewDP[2,2]` entspricht dem max. Wert den wir erhalten, wenn wir die ersten beiden Objekte O_1, O_2 in Betracht ziehen, und sie versuchen in den Rucksack mit max. Gewicht $W_2 = 2$ zu packen

Dynamischer Rucksack: Beispielberechnung

- Das Objekt O_2 hat folgende Eigenschaften:
  1. $\text{Gewicht}(O_2) = 1$
  2. $\text{Wert}(O_2) = 1$
- Da wir den Eintrag `MaxGewDP[2,2]` berechnen möchten, bedeutet dies, dass wir uns in der Iteration `i=2,j=2` befinden, also ist das aktuelle max. Gewicht $W_2 = 2$ und das Objekt, dass gerade betrachtet wird ist O_2

Dynamischer Rucksack: Beispielberechnung

- Betrachten wir den Algorithmus aus Slide 8 sehen wir, dass wir in den 3. Fall gelangen, da $\mathrm{Gewicht}(O_2) \leq W_2$ ist
- Fall 3.1: Wir packen O_2 nicht ein, also erhalten wir

```
W1 := MaxGewDp[i-1,j] := MaxGewDp[1,2] :=
    ↪ 269
```

- Fall 3.2: Wir packen O_2 ein, also erhalten wir:

```
W2 := Wert(objects[1]) + MaxGewDP[1, 1]
    ↪ := 1 + 269 := 270
```

- Berechnung des max. Wert, welches in Zelle `MaxGewDP[2,2]` gespeichert wird:

```
MaxGewDP[2,2] := max(W1, W2) := max(269,
    ↪ 270) := 270
```

Dynamischer Rucksack: Beispielberechnung

- Fall 3.1: `W1 :=` grüne Zelle
- Fall 3.2: `W2 := 1 +` gelbe Zelle
- Ergebnis: `max(W1, W2)` := rote Zelle

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

Übung 10: Dynamisches Wechselgeld, Setup

Es wird eine Tabelle `anz` verwendet, um sich die Anzahl Möglichkeiten wie man einen bestimmten Geldbetrag bezahlen kann zwischenzuspeichern:

- **Zeile**: Münzen, welche in Betracht gezogen werden
- **Spalte**: Betrag
- **Eintrag** `anz(i,j)`: Anzahl Möglichkeiten um den Betrag B_j (Spalte `j`) mit den gegebenen Münzen $M_1, M_2, \ldots, M_i$ (Zeile `i`) darzustellen

Dynamisches Wechselgeld: Münzen

- Die Münzen (siehe `wechselgeld.pas` auf der Infk1 Homepage) sind in einem Array `muenzen` gespeichert

```
muenzen: ARRAY[0..6] OF INTEGER;
```

- Beachte: M_i befindet sich an der Position $(i-1)$ im Array (mit Indizes $0, \ldots, 6$), d.h. `o[i-1] :=` M_i
  - `muenzen[0] :=` M_1
  - `muenzen[1] :=` M_2
  - ...
  - `muenzen[6] :=` M_7
- Siehe Slide 7 für weitere Informationen bezüglich den Indizes

Dynamisches Wechselgeld: Algorithmus

- Auf dem Übungsblatt sind die zu unterscheidenden Fälle erwähnt; verwendet Slide 8 um euch eine Idee vom Algorithmus für's Wechselgeld-Programm zu machen
- Aufgabe 10.1.b: Die erste Zeile entspricht der Abbruchbedingung, diese kann direkt so implementiert werden