



Grundzüge der Informatik 1

Vorlesung 11 - flipped classroom

Dynamische Programmierung

Dynamische Programmierung für Optimierungsprobleme

1. Bestimme rekursive Struktur einer optimalen Lösung durch Zurückführen auf optimale Teillösungen
2. Entwerfe rekursive Methode zur Bestimmung des *Wertes* einer optimalen Lösung.
3. Transformiere rekursive Methode in eine iterative (bottom-up) Methode zur Bestimmung des Wertes einer optimalen Lösung.
4. Bestimmen aus dem Wert einer optimalen Lösung und in 3. ebenfalls berechneten Zusatzinformationen eine optimale Lösung.

Dynamische Programmierung

Das Rucksackproblem

- Rucksack mit begrenzter Kapazität
- Objekte mit unterschiedlichem Wert und unterschiedlicher Größe
- Wir wollen Objekte von möglichst großem Gesamtwert mitnehmen

Beispiel

- Rucksackgröße 6

Größe	5	2	1	3	7	4
Wert	11	5	2	8	14	9

Dynamische Programmierung

Das Rucksackproblem

- Rucksack mit begrenzter Kapazität
- Objekte mit unterschiedlichem Wert und unterschiedlicher Größe
- Wir wollen Objekte von möglichst großem Gesamtwert mitnehmen

Beispiel

- Rucksackgröße 6

Größe	5	2	1	3	7	4
Wert	11	5	2	8	14	9

- Objekt 1 und 3 passen in den Rucksack und haben einen Gesamtwert von 13

Dynamische Programmierung

Das Rucksackproblem

- Rucksack mit begrenzter Kapazität
- Objekte mit unterschiedlichem Wert und unterschiedlicher Größe
- Wir wollen Objekte von möglichst großem Gesamtwert mitnehmen

Beispiel

- Rucksackgröße 6

Größe	5	2	1	3	7	4
Wert	11	5	2	8	14	9

- Objekt 1 und 3 passen in den Rucksack und haben einen Gesamtwert von 13
- Objekte 2,3 und 4 passen und haben Gesamtwert von 15

Dynamische Programmierung

Lösungsansatz

- Bestimme zunächst den Wert einer optimalen Lösung
- Verfolge dazu Ansatz wie bei Maximumssuche und SubsetSum
- Verwende Eingabeordnung der Objekte
- Rekursion: Führe Kosten der Lösung mit n Objekten auf Lösungen mit $n-1$ Objekten zurück
- Leite dann die Lösung selbst aus der Tabelle des dynamischen Programms her

Dynamische Programmierung

Rekursion Rucksackproblem

- Sei $g[i]$ das Gewicht von Objekt i und $w[i]$ sein Wert
 - Sei $\text{Opt}(i,j)$ der Wert einer optimalen Lösung für das Rucksackproblem mit Objekten aus $\{1, \dots, i\}$ und Rucksackgröße j
- (a) $\text{Opt}(1,j) = w[1]$ für $j \geq g[1]$
- (b) $\text{Opt}(1,j) = 0$ für $j < g[1]$
- (c) $\text{Opt}(i,j) = \max\{\text{Opt}(i-1,j), w[i] + \text{Opt}(i-1,j-g[i])\}$, falls $i > 1$ und $g[i] \leq j$, und
- (d) $\text{Opt}(i,j) = \text{Opt}(i-1,j)$, falls $i > 1$ und $g[i] > j$.

Dynamische Programmierung

Rucksack(n,g,w,G)

1. **Opt** = **new array** [1,...,n][0,...,G]
2. **for** j = 0 **to** G **do** * Rekursionsabbruch
3. **if** j < g[1] **then** Opt[1,j] = 0
4. **else** Opt[1,j] = w[1]
5. **for** i = 2 **to** n **do**
6. **for** j = 0 **to** G **do**
7. **if** g[i] ≤ j **then** Opt[i,j] = max{Opt[i-1,j], w[i] + Opt[i-1,j-g[i]]}
8. **else** Opt[i,j] = Opt[i-1,j]
9. **return** Opt[n,G]

Dynamische Programmierung

Wie kann man eine optimale Lösung berechnen?

- Idee: Verwende Tabelle der dynamischen Programmierung
- Fallunterscheidung + Rekursion:
 - Falls das i -te Objekt in einer optimalen Lösung für Objekte 1 bis i und Rucksackgröße j ist, so gib es aus und fahre rekursiv mit Objekt $i-1$ und Rucksackgröße $j-g[i]$ fort
 - Ansonsten fahre mit Objekt $i-1$ und Rucksackgröße j fort

Dynamische Programmierung

RucksackLösung(Opt,g,w,i,j)

1. **if** $i=0$ **return** \emptyset
2. **else if** $g[i]>j$ **then return** RucksackLösung(Opt,g,w,i-1,j)
3. **else if** $\text{Opt}[i,j]=w[i] + \text{Opt}[i-1,j-g[i]]$ **then**
 return $\{i\} \cup \text{RucksackLösung}(\text{Opt},g,w,i-1,j-g[i])$
4. **else return** RucksackLösung(Opt,g,w,i-1,j)

Aufruf

- Nach der Berechnung der Tabelle Opt von Rucksack wird RucksackLösung mit Opt, g,w, $i=n$ und $j=G$ aufgerufen.

Dynamische Programmierung

Wiederholung

- Beim Spiel „Jump“ gibt es ein Feld $A[1..n]$, das jeweils einen Punktwert enthält (der auch negativ sein kann)
- Die Spielfigur startet auf Position 1 und kann sich entweder einen oder 2 Schritte nach rechts bewegen
- Für jedes Feld, auf dem die Figur steht, erhält sie seine Punkte
- Am Ende muss die Figur auf Position n stehen
- Entwickeln Sie eine Rekursion für die optimale Anzahl an erreichbaren Punkten

Dynamische Programmierung

Score(A,n)

1. **if** $n=1$ **then return** $A[1]$
2. **if** $n=2$ **then return** $A[1] + A[2]$
3. **return** $A[n] + \max(\text{Score}(A, n-1), \text{Score}(A, n-2))$

Dynamische Programmierung

Aufgabe 1

- Skizzieren Sie die Anzahl rekursiver Aufrufe für den Fall $n=5$

Dynamische Programmierung

Aufgabe 2

- Lösen Sie das Problem mit Hilfe von dynamischer Programmierung
- Was ist die Laufzeit Ihres Algorithmus?

Dynamische Programmierung

Aufgabe 3

- Beim Spiel „Jump“ gibt es ein Feld $A[1..n]$, das jeweils einen Punktwert enthält (der auch negativ sein kann)
- Die Spielfigur startet auf Position 1 und kann sich entweder einen oder 2 Schritte nach rechts bewegen
- Für jedes Feld, auf dem die Figur steht, erhält sie seine Punkte
- Am Ende muss die Figur auf Position n stehen
- Entwickeln Sie einen Algorithmus, der eine optimale Lösung ausgibt
- Vollziehen Sie Ihre Algorithmen an folgendem Beispiel nach
- 10, -5, -15, -50, 20, 17, 3, -7, 10

Dynamische Programmierung

Aufgabe 4

- Betrachten Sie die Modifikation von Jump, bei der auch Sprünge über 3 Felder erlaubt sind
- Stellen Sie zunächst eine Rekursionsgleichung auf

Dynamische Programmierung

Aufgabe 5

- Betrachten Sie die Modifikation von Jump, bei der auch Sprünge über 3 Felder erlaubt sind
- Entwickeln Sie mit Hilfe von dynamischer Programmierung einen Algorithmus der für diese Variante den Wert einer optimalen Lösung berechnet

Dynamische Programmierung

Aufgabe 6

- Betrachten Sie die Modifikation von Jump, bei der auch Sprünge über 3 Felder erlaubt sind
- Entwickeln Sie dann einen Algorithmus, der eine optimale Lösung berechnet