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 Opt(i,j) der Wert einer optimalen Lösung für das Rucksackproblem mit Objekten aus {1,…,i} und Rucksackgröße j

(a) Opt(1,j)= w[1] für j≥g[1]

(b) Opt(1,j)= 0 für j<g[1]

(c) Opt(i,j) = max{Opt(i-1,j), w[i] + Opt(i-1,j-g[i])}, falls i>1 und g[i]≤j, und

(d) Opt(i,j) = Opt(i-1,j), falls i>1 und g[i]>j.

Dynamische Programmierung

Rucksack(n,g,w,G)

```
Opt = new array [1,..,n][0,..,G]
for j = 0 to G do                              \* Rekursionsabbruch
    if j<g[1] then Opt[1,j]  = 0
    else Opt[1,j] = w[1]
for i = 2 to n do
    for j = 0 to G do
        if g[i] ≤j then Opt[i,j] = max{Opt[i-1,j], w[i] + Opt[i-1,j-g[i]]}
    else Opt[i,j] = Opt[i-1,j]
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** ∅
2. **else if** g[i]>j **then return** RucksackLösung(Opt,g,w,i-1,j)
3. **else if** Opt[i,j]=w[i] + Opt[i-1,j-g[i]] **then**
   return {i} ∪ RucksackLösung(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 (Score(A,n-1), Score(A,n-2))

Dynamische Programmierung

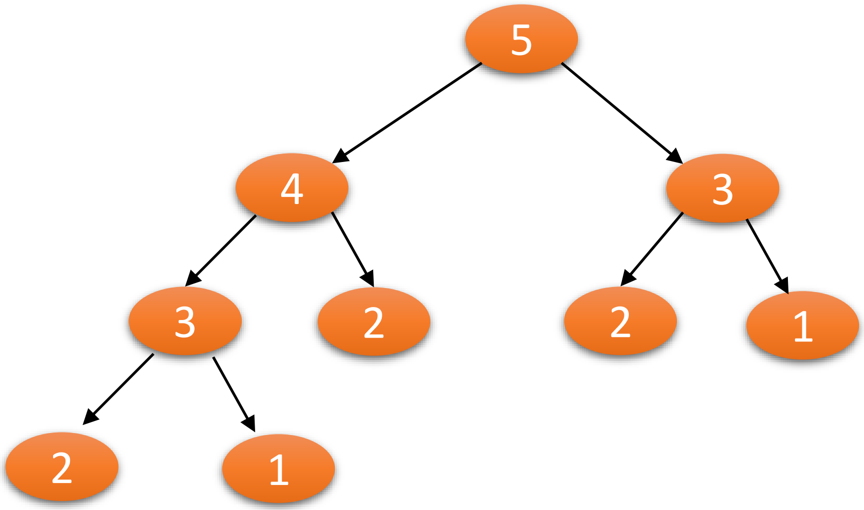
Aufgabe 1

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

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 (Score(A,n-1), Score(A,n-2))



Dynamische Programmierung

Aufgabe 2

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

Dynamische Programmierung

ScoreDP(A,n)

1. Score = new array [1..n]
2. Score[1] = A[1]
3. Score[2] = A[1] + A[2]
4. **for** i=3 **to** n **do**
5.   Score[i] = A[i] + max(Score[i-1],Score[i-2])
6. **return** Score[n]

Dynamische Programmierung

ScoreDP(A,n)

1. Score = new array [1..n]
2. Score[1] = A[1]
3. Score[2] = A[1] + A[2]
4. **for** i=3 **to** n **do**
5.   Score[i] = A[i] + max(Score[i-1],Score[i-2])
6. **return** Score[n]

Laufzeit

- Die Laufzeit des Algorithmus ist O(n)

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

ScoreDP(A,n)

1. Score = new array [1..n]
2. Score[1] = A[1]
3. Score[2] = A[1] + A[2]
4. **for** i=3 **to** n **do**
5. Score[i] = A[i] + max(Score[i-1],Score[i-2])
6. **return** Score[n]

Dynamische Programmierung

ScoreDP(A,n)

1. Score = new array [1..n]
2. Score[1] = A[1]
3. Score[2] = A[1] + A[2]
4. **for** i=3 **to** n **do**
5. Score[i] = A[i] + max(Score[i-1],Score[i-2])
6. **return** Score[n]

A: 10, -5, -15, -50, 20, 17, 3, -7, 10

Score: 10, 5, -5, -45, 15, 32, 35, 28, 45

Dynamische Programmierung

ScoreLösung(A,Score,n)

1. **if** n=1 **then return** {1}
2. **if** n=2 **then return** {1,2}
3. **if** Score[n] = A[n] + Score[n-1] **then return** {n} $\cup$ ScoreLösung(A,Score,n-1)
4. **else return** {n} $\cup$ ScoreLösung(A,Score, n-2)

Dynamische Programmierung

ScoreLösung(A,Score,n)

1. **if** n=1 **then return** {1}
2. **if** n=2 **then return** {1,2}
3. **if** Score[n] = A[n] + Score[n-1] **then return** {n} $\cup$ ScoreLösung(A,Score,n-1)
4. **else return** {n} $\cup$ ScoreLösung(A,Score, n-2)

A: 10, -5, -15, -50, 20, 17, 3, -7, 10

Score: 10, 5, -5, -45, 15, 32, 35, 28, 45

Ausgabe: 9, 7, 6, 5, 3, 1