Dynamisk Programmering

Tre exempel:

- Maximal delsekvens
- Längsta ökande sekvens
- Växling av pengar

Exempel 1: Maximal delsekvens

Problem: Givet en sekvens $A[1 \ldots n]$, finn den största summan i en sammanhängande delsekvens av A.

31	-41	59	26	-53	58	97	-93	-23

M a o, finn j, k så $\sum_{i=j}^{k} A[i]$ maximeras.

Experimentera med problemet!

- "Hm. . . om alla talen är positiva är svaret hela A"

31	41	59	26	53	58	97	93	23

- "Hm. . . om alla talen är negativa är svaret en tom sekvens"

-31	-41	-59	-26	-53	-58	-97	-93	-23

- "Hm. . . en maximal delsekvens börjar alltid med ett positivt tal"

31	-41	59	26	-53	58	97	-93	-23

- "Hm. . . en maximal delsekvens slutar alltid med ett positivt tal"

En naiv lösning

Konstruera en tabell $T[i, j]$ så att $T[j, k] = \sum_{i=j}^{k} A[i]$.

31	-41	59	26	-53	58	97	-93	-23

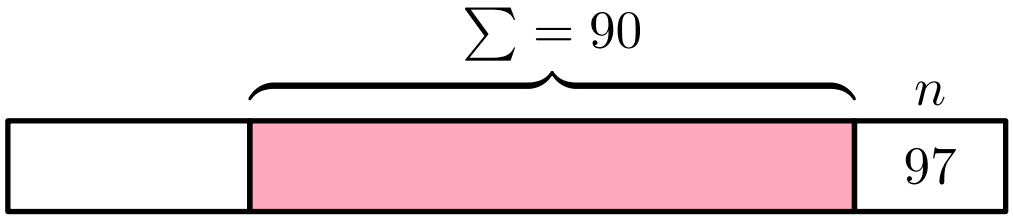
$j \backslash k$	2	3	4	5	6	7	8	9	10
1	-10	49	75	22	80	177	84	61	145
2		18	44	-9	49	146	53	30	114
3			85	32	90	187	94	71	155
4				-27	31	128	35	12	96
5					5	102	9	-14	70
6						155	62	39	123
7							4	-19	65
8								-116	-32
9									61

Kostnad för en naiv lösning

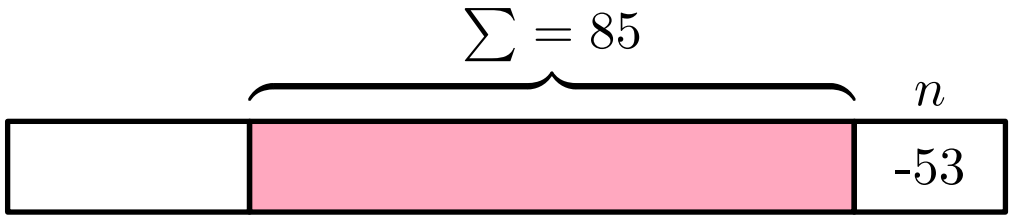
- Att skapa $T[i, j]$ tar $\Theta(n^2)$ utrymme.
- Att hitta maximala delsekvensen tar $\Theta(n^2)$ tid.
- Kan vi göra det på ett billigare sätt?
- Ja, om vi använder dynamisk programmering.

Börja med att hitta en rekursiv formulering

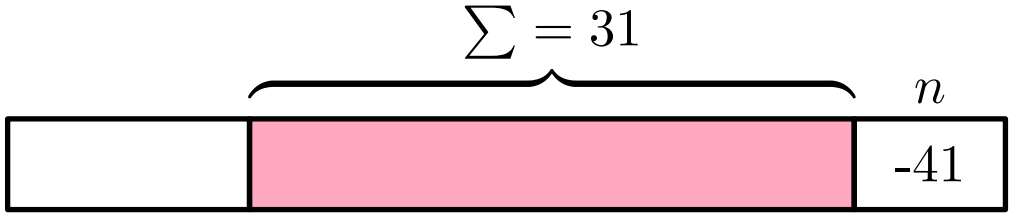
Fråga: Vilken information om de första $n-1$ elementen, *tillsammans* med $A[n]$ ger lösningen för hela sekvensen?



Ska 97 tas med?



Ska -53 tas med?



Ska -41 tas med?

Rekursiv formulering

- Maximala summan i $A[1 \ldots n-1]$ verkar användbar, kalla den $S[n-1]$.
- Värdet på $A[n]$ påverkar också resultatet:
  - Om $A[n] > 0$ ska den definitivt tas med:

$$S[n] = S[n-1] + A[n]$$

  - Om $A[n] \leq 0$ kan den tas med om den inte gör summan negativ:

$$S[n] = \max(S[n-1] + A[n], 0)$$

Rekursiv formulering

Definera den maximala summan i $A[1 \ldots i]$ som

$$S[i] = \begin{cases} S[i-1] + A[i] & \text{om } A[i] > 0 \\ \max(S[i-1] + A[i], 0) & \text{om } A[i] \leq 0 \end{cases}$$

Låt $S[0] = 0$ vara ett basfall.

Vi provar. . .

A:

31	-41	59	26	-53	58	97	-93	-23

S:

31	0	59	85	32	90	187	94	71

- Lösningen är $\max_i S[i]$.
- Start- och slutindex kan rekonstrueras från S. Hur?
- Kostnad: $\Theta(n)$ tid och utrymme.

Exempel 2: Längsta ökande sekvens

Problem: Givet $A[1 \ldots n]$, finns den längsta ökande sekvensen.

9	5	2	8	7	3	1	6	4

Dvs, de utvalda elementen måste vara sorterade från höger till vänster.

I detta fall finns två lösningar: $\langle 2, 3, 4 \rangle$ och $\langle 2, 3, 6 \rangle$.

Börja med att hitta en rekursiv formulering

Fråga: Vilken information om de första $n-1$ elementen, *tillsammans* med $A[n]$ ger lösningen för hela sekvensen?

- Längden av den längsta ökande sekvensen i $A[1 \ldots n-1]$ verkar användbar.
- Detta *är* den längsta sekvensen, om inte $A[n]$ kan utöka den.

- Tyvärr räcker det inte att veta hur lång sekvensen i $A[1 \ldots n-1]$ är.

  Om jag berättar att den längsta ökande sekvensen i $A[1 \ldots n-1]$ har längden 5 och $A[n] = 9$, vad ska du göra då?

- Vi behöver veta den längsta ökande sekvensen som $A[n]$ kan utöka!

Rekursiv formulering

Låt $L[i]$ vara längden av den längsta ökande sekvensen som slutar med $A[i]$.

$$L[i] = 1 + \max_{0<j<i} L[j] \qquad \text{där } A[j] < A[i]$$

$$L[0] = 0$$

Längden av den längst ökande sekvensen av hela A:

$$\max_{1 \leq i \leq n} L[i]$$

Vi provar . . .

$$L[i] = 1 + \max_{0<j<i} L[j] \quad \text{där } A[j] < A[i]$$

$$L[0] = 0$$

A:

9	5	2	8	7	3	1	6	4

L:

1	1	1	2	2	2	1	3	3

Kostnadsanalys

- Varje element $A[i]$ jämförs med $A[1 \ldots i-1]$.
- Detta ger $\Theta(n^2)$ tid.
- Med lite smartare datastrukturer: $\Theta(n \lg n)$.
- Hur kan vi rekonstruera lösningen?

  Varje element $A[i]$ kan peka ut det element som föregår sig själv i den längsta ökande sekvensen.

Exempel 3: Växling av pengar

Problem: För en given summa N och ett obegränsat antal mynt med värden $d_1 \ldots d_n$, beräkna det minsta antalet mynt som behövs för N.

Exempel: För $N = 86$ öre och $d_1 = 1$, $d_2 = 2$, $d_3 = 5$, $d_4 = 10$, $d_5 = 25$, $d_6 = 50$, $d_7 = 100$ är en 50-öring, en 25-öring, en 10-öring och en 1-öring optimalt.

En girig algoritm

GREEDY-MAKE-CHANGE(N)

1. $D = \langle 100, 50, 25, 10, 5, 2, 1 \rangle$
2. $S \leftarrow \emptyset$
3. $s \leftarrow 0$
4. **while** $s \neq N$
5. **do** $d \leftarrow$ largest coin $\in D$ such that $s + d \leq N$
6. **if** no such d
7. **then return** "No solution"
8. $S \leftarrow S \cup \{d\}$
9. $s \leftarrow s + d$
10. **return** S

En girig algoritm är inte alltid optimal

- Antag att vi bara har $D = \langle 100, 25, 10, 1 \rangle$.
- Hitta ett N så att GREEDY-MAKE-CHANGE inte ger en optimal lösning.
- Exempel: $N = 30$.
  - Girig algoritm $\Rightarrow$ 25+1+1+1+1+1
  - Optimalt val $\Rightarrow$ 10+10+10
- Hur hitta ett optimal val? Dynamisk programmering!

En lösning med dynamisk programmering

- Antag en mängd mynt $d_1, \ldots, d_n$ och en summa N.
- Använd en tabell $C[1 \ldots n, 0 \ldots N]$.

  $C[i, j]$ är det minsta antalet mynt som krävs för att betala j öre, med endast mynten $d_1, \ldots d_i$.
- Lösningen till hela problemet är då $C[n, N]$.
- Vi rekonstruerar lösningen senare.

Hitta en rekursiv formulering

$C[i, j]$ är det minsta antalet mynt som krävs för att betala j öre, med endast mynten $d_1, \ldots d_i$.

- Att betala j öre med mynten $d_1, \ldots, d_i$ ger två val:

1. Använd inte mynt d_i (även om det går!):

$$C[i, j] = C[i - 1, j]$$

2. Använd ett mynt d_i och reducera summan:

$$C[i, j] = 1 + C[i, j - d_i]$$

En rekursiv formulering

- Vi väljer givetvis den lösning med minst antal mynt:

$$C[i,j] = \min(C[i-1,j], 1 + C[i, j-d_i])$$

- Om $i < 1$ eller $j < d_i$ kommer vi att referera utanför tabellen.
- Låt $C[i,j] = +\infty$ i sådana situationer.

Vi provar. . .

- Låt $d_1 = 1$, $d_2 = 4$, $d_3 = 6$ och $N = 8$.

Summa:	0	1	2	3	4	5	6	7	8
$d_1 = 1$	0	1	2	3	4	5	6	7	8
$d_2 = 4$	0	1	2	3	1	2	3	4	2
$d_3 = 6$	0	1	2	3	1	2	1	2	2

- $C[3, 8] = \min(C[2, 8], 1 + C[3, 8 - d_3])$

- Lösningen använde alltså *inte* d_3.

MAKE-CHANGE(N)

1. $D = \langle 1, 4, 6 \rangle$ ▷ must be sorted
2. **for** $i \leftarrow 0$ **to** n
3. **do** $C[i, 0] \leftarrow 0$
4. **for** $i \leftarrow 1$ **to** n
5. **do for** $j \leftarrow 1$ **to** N
6. **do if** $i = 1$ **and** $j < d_i$ **then** $C[i, j] \leftarrow +\infty$
7. **elsif** $i = 1$ **then** $C[i, j] \leftarrow 1 + C[i, j - d_1]$
8. **elsif** $j < d_i$ **then** $C[i, j] \leftarrow C[i - 1, j]$
9. **else** $C[i, j] = \min(C[i - 1, j], 1 + C[i, j - d_i])$
10. **return** $C[n, N]$

Analys

- Om en 1-öring finns med kan vi alltid hitta en lösning.
- Annars returneras $+\infty$.
- Tidsanalysen är enkel: $\Theta(nN)$.
- Lösningen kan rekonstrueras genom att starta i $C[n, N]$ och studera alternativen bakåt.

Slutsater om Dynamisk Programmering

- Alla lösningar har egenskapen ”optimal substructure”:

  En optimal lösning innehåller optimala lösningar till delproblemen.

- Motexempel: Hitta längsta vägen från A till C.

- Inte säkert att $LV(A, C) = LV(A, B) + LV(B, C)$ om B ligger på den längsta vägen från A till C.