Dynamisk programmering

Kombinatoriske optimeringsproblemer

Kombinatorisk struktur: En struktur opbygget af et endeligt antal enkeltdele. Eksempler:

- Rute fra A til B.
- Pakning af lastbil eller containerskib.
- Undervisningsskema.
- Produktionsplan for y ordrer og x produktionsmaskiner.

Kombinatoriske optimeringsproblemer

Kombinatorisk struktur: En struktur opbygget af et endeligt antal enkeltdele. Eksempler:

- Rute fra A til B.
- Pakning af lastbil eller containerskib.
- Undervisningsskema.
- Produktionsplan for y ordrer og x produktionsmaskiner.

Kombinatorisk optimeringsproblem: man ønsker at finde den bedste kombinatoriske struktur blandt mange mulige. Eksempler:

- Hurtigste rute fra A til B.
- Mest profitable pakning af lastbil eller containerskib.
- Undervisningsskema med mindst mulig undervisning efter kl. 16.
- Produktionsplan for y ordrer og x produktionsmaskiner med færrest overskridelser af leveringsfrister.

Dynamisk programmering

Dynamisk programmering [Bellman, 1950-57]: en metode til at udvikle algoritmer til kombinatoriske optimeringsproblemer.

Dynamisk programmering

Dynamisk programmering [Bellman, 1950-57]: en metode til at udvikle algoritmer til kombinatoriske optimeringsproblemer.

Dynamisk programmering er et specialtilfælde af Divide-and-Conquer metoden—dvs. er en rekursiv metode, som opbygger løsninger til større problemer ud fra løsninger til mindre problemer.

Observation:

- Normalt i rekursive metoder: delproblemer typisk halvt så store og der er ingen gentagelser af delproblemer forskellige steder i rekursionstræet.
- Nogle rekursive metoder: Kald af delproblemer hvis størrelse kun er reduceret med én. Der vil så ofte opstå gentagelser af delproblemer forskellige steder i rekursionstræet, hvilket ofte gør køretiden eksponentiel.

Dynamisk programmering

Kernen i dynamisk programmering er følgende idé:

- Lav en tabel over løsninger på delproblemer, så disse kun skal løses én gang hver. Dette ændrer normalt køretiden fra eksponentiel til polynomiel.

Dynamisk programmering

Kernen i dynamisk programmering er følgende idé:

- Lav en tabel over løsninger på delproblemer, så disse kun skal løses én gang hver. Dette ændrer normalt køretiden fra eksponentiel til polynomiel.

Mere generelt bruges begrebet dynamisk programmering om

- Udvikling af rekursive løsninger for optimeringsproblemer, hvor nogle delproblemer i rekursionen kun er reduceret $O(1)$ i størrelse. Man bruger så idéen ovenfor til at implementere den rekursive løsning effektivt.

Den kreative del er at finde den rekursive beskrivelse af løsningen. At derefter bruge idéen ovenfor er ret ens fra problem til problem.

Dynamisk programmering

Den kreative del er at finde den rekursive beskrivelse af løsningen.

Dynamisk programmering

Den kreative del er at finde den rekursive beskrivelse af løsningen.

Følgende er ofte en god angrebsvinkel:

1. Hvad kunne være en god beskrivelse af størrelsen af et problem, udtrykt ved ét, to eller evt. flere heltalsindekser? Det giver så en tabel med én, to eller flere dimensioner.
2. Analysér hvordan en optimal løsning for en given problemstørrelse må bestå af en "sidste del" og "resten", hvor man om "resten" kan argumentere, at denne må være en optimal løsning for et mindre problem af samme type. Derved kan fås en rekursiv beskrivelse af løsninger.

Den sidste egenskab kaldes, at der er "optimale delproblemer".

Princippet forstås bedst gennem eksempler.

Eksempel: Maltes problem

Eksempel: Maltes problem

En masse guldkæder i overskud:

Foto: © Kaspar Wenstrup

Eksempel: Maltes problem

Du har en guldkæde med n led. Den kan deles i mindre længder (med n led tilsammen, dvs. ingen led går tabt). Guldsmeden køber guldkæder af forskellige længder til forskellige priser:

længde i (antal led):	1	2	3	4	5	6	7	8	9	$\cdots$
pris p_i (1.000 kr.):	1	5	8	9	10	17	17	20	24	$\cdots$

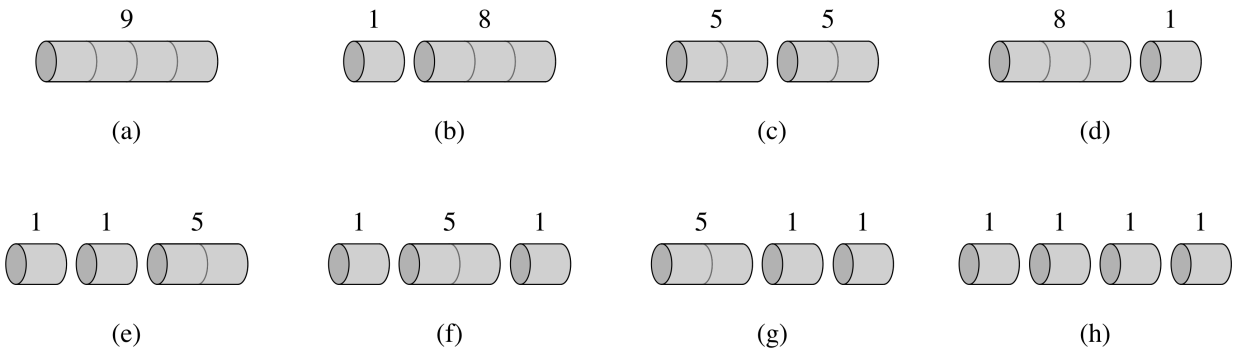
Hvordan skal du opdele din lange guldkæde for at optimere din salgspris?

Eksempel: Maltes problem

Du har en guldkæde med n led. Den kan deles i mindre længder (med n led tilsammen, dvs. ingen led går tabt). Guldsmeden køber guldkæder af forskellige længder til forskellige priser:

længde i (antal led):	1	2	3	4	5	6	7	8	9	$\cdots$
pris p_i (1.000 kr.):	1	5	8	9	10	17	17	20	24	$\cdots$

Hvordan skal du opdele din lange guldkæde for at optimere din salgspris?

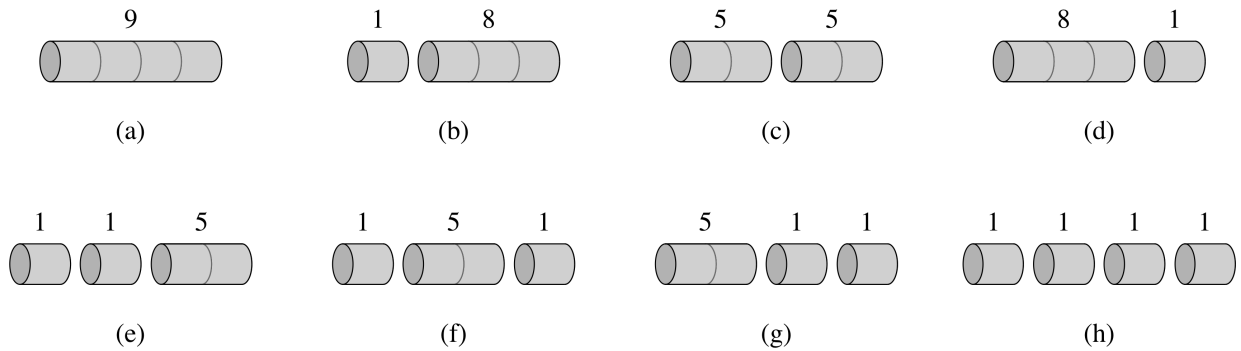


Eksempel: Maltes problem

Du har en guldkæde med n led. Den kan deles i mindre længder (med n led tilsammen, dvs. ingen led går tabt). Guldsmeden køber guldkæder af forskellige længder til forskellige priser:

længde i (antal led):	1	2	3	4	5	6	7	8	9	$\cdots$
pris p_i (1.000 kr.):	1	5	8	9	10	17	17	20	24	$\cdots$

Hvordan skal du opdele din lange guldkæde for at optimere din salgspris?



Der er 2^{n-1} forskellige opdelinger, så det er ikke en effektiv algoritme blot at prøve dem alle.

Optimale delproblemer

Enhver opdeling af en kæde af længde n må bestå af:

- Et sidste stykke af længde $k \leq n$.
- En opdeling af resten, dvs. en opdeling af en kæde af længde $n - k$.

Optimale delproblemer

Enhver opdeling af en kæde af længde n må bestå af:

- Et sidste stykke af længde $k \leq n$.
- En opdeling af resten, dvs. en opdeling af en kæde af længde $n - k$.

Observation (optimale delproblemer):

> For en *optimal* opdelingen af kæden af længde n, må opdelingen af resten selv være optimal for en kæde af længde $n - k$. For hvis der fandtes en ægte bedre opdeling af resten, kunne man bruge den i stedet og derved forbedre den optimale opdeling af kæden af længde n.

Kald *værdien* af en optimal opdeling af en kæde af længde n for $r(n)$.

Det er klart at $r(0) = 0$. Vi vil gerne finde $r(n)$ for $n > 0$.

Rekursiv formel for $r(n)$

Opsummering: en optimal opdeling T for længde n består af:

- Et sidste stykke med længde $k \leq n$.
- En optimal opdeling af resten, dvs. en optimal opdeling af en kæde af længde $n - k$.

Værdien $r(n)$ af T er derfor lig $p_k + r(n - k)$, så det lugter af rekursion.

Rekursiv formel for $r(n)$

Opsummering: en optimal opdeling T for længde n består af:

- Et sidste stykke med længde $k \leq n$.
- En optimal opdeling af resten, dvs. en optimal opdeling af en kæde af længde $n - k$.

Værdien $r(n)$ af T er derfor lig $p_k + r(n - k)$, så det lugter af rekursion.

Men: vi kender desværre ikke k!

Rekursiv formel for $r(n)$

Værdien $r(n)$ af T er derfor lig $p_k + r(n-k)$, så det lugter af rekursion, men vi kender desværre ikke k.

Rekursiv formel for $r(n)$

Værdien $r(n)$ af T er derfor lig $p_k + r(n-k)$, så det lugter af rekursion, men vi kender desværre ikke k.

Derfor gør vi sådan:

Lad T_i (for $i = 1 \ldots n$) være en opdelingen bestående af et sidste stykke af længde i, samt en optimal opdeling af resten.

Værdien af T_i er $p_i + r(n-i)$.

T_k har værdi $p_k + r(n-k)$ lige som T og er derfor optimal for længde n.

- Så (mindst) én af T_1, T_2, T_3,..., T_n er optimal for længde n.
- Naturligvis kan ingen T_i have en værdi bedre end optimal.

Heraf:

$$r(n) = \max_{1 \le i \le n} (p_i + r(n-i)), \qquad r(0) = 0$$

Beregne de optimale værdier

$$r(n) = \max_{1 \leq i \leq n} (p_i + r(n-i)), \qquad r(0) = 0$$

Dvs. $r(n)$ er (matematisk set) rekursivt defineret ud fra mindre instanser.

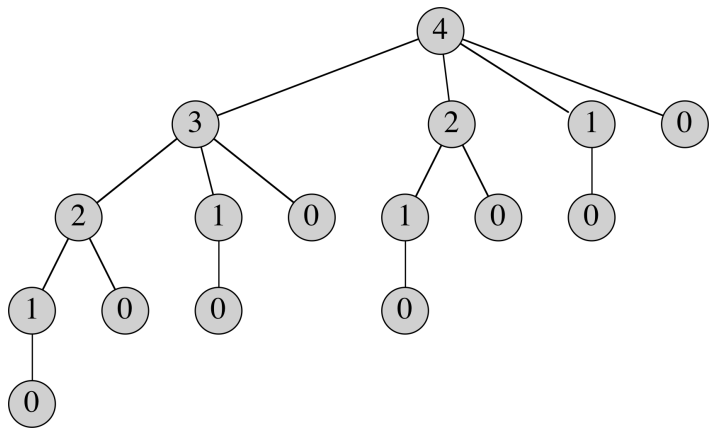
Er rekursion også en god løsning, algoritmisk set?

Beregne de optimale værdier

$$r(n) = \max_{1 \le i \le n} (p_i + r(n-i)), \qquad r(0) = 0$$

Dvs. $r(n)$ er (matematisk set) rekursivt defineret ud fra mindre instanser.

Er rekursion også en god løsning, algoritmisk set?



Man kan vise via induktion at der er $1 + (1 + 2 + 4 + \cdots + 2^{n-1}) = 2^n$ knuder i rekursionstræet. Så køretiden vil blive $\Theta(2^n)$

Problemet er gentagelser blandt delproblemers delproblemer.

Brug en tabel

Fokusér i stedet på en tabel over værdien af de optimale løsninger.

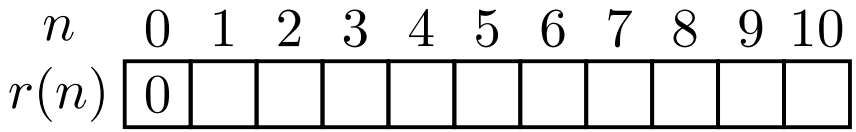
Start: $r(0) = 0$

n	0	1	2	3	4	5	6	7	8	9	10
$r(n)$	0										

Brug en tabel

Fokusér i stedet på en tabel over værdien af de optimale løsninger.

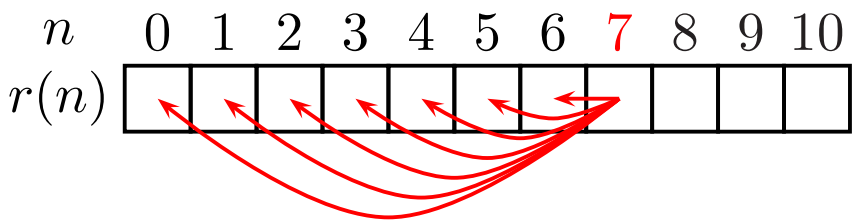
Start: $r(0) = 0$



Et felt kan fyldes ud via

$$r(n) = \max_{1 \le i \le n} (p_i + r(n-i))$$

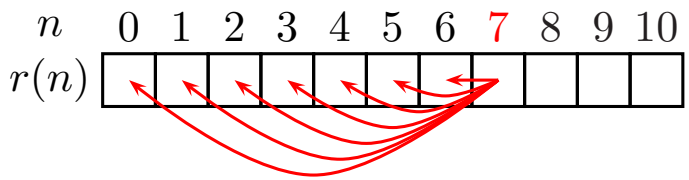
hvis de foregående felter er fyldt ud. Denne afhængighed kan vi illustrere:



Deraf følger, at vi kan beregne $r(n)$ bottom-up, dvs. for stigende n.

Køretid

$$r(n) = \max_{1 \le i \le n} (p_i + r(n-i)), \qquad r(0) = 0$$

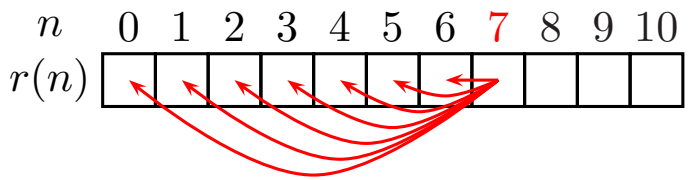


Denne beregning kan laves med to simple for-loops (det ydre går gennem tabellen felt for felt, det inderste finder max for hvert felt):

```
r[0] = 0
for k = 1 to n:
    max = −∞
    for i = 1 to k:
        x = p[i] + r[k − i]:
        if x > max:
            max = x
    r[k] = max
```

Køretid

$$r(n) = \max_{1 \le i \le n} (p_i + r(n-i)), \qquad r(0) = 0$$



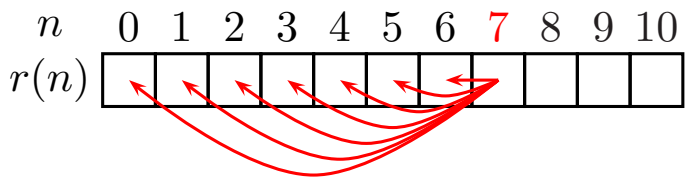
Denne beregning kan laves med to simple for-loops (det ydre går gennem tabellen felt for felt, det inderste finder max for hvert felt):

```
r[0] = 0
for k = 1 to n:
    max = −∞
    for i = 1 to k:
        x = p[i] + r[k − i]:
        if x > max:
            max = x
    r[k] = max
```

Tid:

Køretid

$$r(n) = \max_{1 \le i \le n} (p_i + r(n-i)), \qquad r(0) = 0$$



Denne beregning kan laves med to simple for-loops (det ydre går gennem tabellen felt for felt, det inderste finder max for hvert felt):

```
r[0] = 0
for k = 1 to n:
    max = −∞
    for i = 1 to k:
        x = p[i] + r[k − i]:
        if x > max:
            max = x
    r[k] = max
```

Tid: $O(1 + 2 + 3 + 4 + \cdots + n) = \Theta(n^2)$

Eksempel

Brug

$$r(n) = \max_{1 \le i \le n} (p_i + r(n-i)), \qquad r(0) = 0$$

og priserne p_i:

længde i	1	2	3	4	5	6	7	8	9	10
pris p_i	1	5	8	9	10	17	17	20	24	26

til at udfylde tabellen over $r(n)$ fra højre mod venstre:

n	0	1	2	3	4	5	6	7	8	9	10
$r(n)$	0	1	5	8	10	13	17	18	22	...	

Eksempel

Brug

$$r(n) = \max_{1 \le i \le n} (p_i + r(n-i)), \qquad r(0) = 0$$

og priserne p_i:

længde i	1	2	3	4	5	6	7	8	9	10
pris p_i	1	5	8	9	10	17	17	20	24	26

til at udfylde tabellen over $r(n)$ fra højre mod venstre:

n	0	1	2	3	4	5	6	7	8	9	10
$r(n)$	0	1	5	8	10	13	17	18	22	25	...

Eksempel

Brug

$$r(n) = \max_{1 \le i \le n} (p_i + r(n-i)), \qquad r(0) = 0$$

og priserne p_i:

længde i	1	2	3	4	5	6	7	8	9	10
pris p_i	1	5	8	9	10	17	17	20	24	26

til at udfylde tabellen over $r(n)$ fra højre mod venstre:

n	0	1	2	3	4	5	6	7	8	9	10
$r(n)$	0	1	5	8	10	13	17	18	22	25	27

Find selve løsningen

Tallet $r(n)$ er kun *værdien* af den optimale løsning. Hvad hvis vi gerne vil have *selve løsningen* (de enkelte længder, guldkæden skal brydes op i)?

Gem længden $s(n)$ af det *sidste stykke* for en optimal løsning for længde n. Dvs. gem det i som giver max i den rekursive ligning.

$$r(n) = \max_{1 \le i \le n} (p_i + r(n-i)), \qquad r(0) = 0$$

længde i	1	2	3	4	5	6	7	8	9	10
pris p_i	1	5	8	9	10	17	17	20	24	26

længde n	0	1	2	3	4	5	6	7	8	9	10
optimal værdi $r(n)$	0	1	5	8	10	13	17	18	22	25	27
sidste længde $s(n)$	0	1	2	3	2	2	6	1	2	3	2

while $n > 0$
  print $s[n]$
  $n = n - s[n]$

Memoization

Rekursion: $\Theta(2^n)$. Struktureret tabeludfyldning: $\Theta(n^2)$

Kan de to kombineres?

Memoization

Rekursion: $\Theta(2^n)$. Struktureret tabeludfyldning: $\Theta(n^2)$

Kan de to kombineres? Ja.

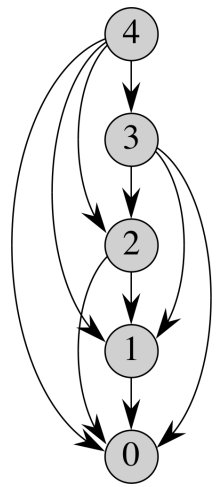
$\textsc{GuldKæde}(n)$
  if $n = 0$
    return 0
  else if $r[n]$ allerede udfyldt i tabel
    return $r[n]$
  else
    $x = \max_{1 \leq i \leq n}(p_i + \textsc{GuldKæde}(n - i))$
    $r[n] = x$
    return x

Memoization

Rekursion: $\Theta(2^n)$. Struktureret tabeludfyldning: $\Theta(n^2)$

Kan de to kombineres? Ja.

```
GuldKæde(n)
    if n = 0
        return 0
    else if r[n] allerede udfyldt i tabel
        return r[n]
    else
        x = max_{1≤i≤n}(p_i + GuldKæde(n − i))
        r[n] = x
        return x
```



En pil i figuren, der viser et delproblems afhængighed af andre, vil blive en kant i rekursionstræet præcis én gang (første gang delproblemet nås).

Så samme køretid $\Theta(n^2)$ og pladsforbrug $\Theta(n)$ som for bottom-up udfyldning af tabellen. Men nok en lidt dårligere konstant i praksis.