

datastructuren en algoritmen
2011 10 03
college 9

- **datastructuren: lineair en hierarchisch**
stacks, queues, vectoren, lijsten, rijtjes, priority queues
bomen, binaire bomen, binaire zoekbomen, AVL-bomen, heaps
- **sorteer-algoritmen**
heap-sort, selection sort, insertion sort, merge sort, quick sort,
quick select
- **hashing**

voorbeeld greedy algoritme: geld wisselen

- **probleem:**
maak bedrag N met zo min mogelijk muntjes/briefjes
- **voorbeeld:**
maak bedrag van 4,35 euro
- **greedy algoritme voor euro-muntenstelsel:**
vul steeds aan met zo groot mogelijke eenheid

voorbeeld dynamic programming: geld wisselen

- **probleem:**
maak bedrag N met zo min mogelijk muntjes/briefjes
- **voorbeeld:**
maak 6 met munten uit $\{1, 3, 4\}$
- **meer algemeen:**
geef optimale oplossing voor munten van 1,
dan voor munten van 1, 3,
dan voor munten van 1, 3, 4
gebruik deeloplossingen

greedy algoritmen en dynamic programming

- **greedy algoritme:**
kies steeds lokaal optimale oplossing
om tot globaal optimale oplossing te komen
alleen als probleem **greedy choice property** heeft
- **dynamic programming:**
vaak bottom-up aanpak
waarbij we overlappende deeloplossingen gebruiken
lijkt wel een beetje op verdeel en heers maar niet perse met recursie

schema

- recap en intro
- hout hakken
- maximaal subarray
- matrix vermenigvuldiging
- fractional knapsack
- knapsack 0/1
- task scheduling
- materiaal

schema

- recap en intro
- hout hakken
- maximaal subarray
- matrix vermenigvuldiging
- fractional knapsack
- knapsack 0/1
- task scheduling
- materiaal

hout hakken

- **probleem:**
gegeven een stuk hout van n decimeter
gegeven prijzen p_i met $i = 1, \dots, n$ voor stuk van i decimeter
bepaal (met gratis hakken) hoe je moet hakken voor de grootste winst
- **hoeveel mogelijkheden zijn er?**
 2^{n-1}

hout hakken: voorbeeld

- stuk hout van lengte 4
- prijzen:

lengte i	1	2	3	4
prijs p_i	1	5	8	9

houthakken: tijdscomplexiteit

- bekijk voorbeeld voor $n = 4$
- recurrente betrekking:

$$T(n) = 1 + \sum_{j=0}^{n-1} T(j)$$

- intuïtief: voor elke hakplek bekijken we mogelijkheid wel of niet

hout hakken: top-down algoritme

input: array $p[1 \dots n]$ met prijzen en int n

output: maximum winst voor hout ter lengte n

Algorithm houthak(p, n):

```
if  $n = 0$  then
  return 0
 $q := -\infty$ 
for  $i := 1$  to  $n$  do
   $q := \max(q, p[i] + \text{houthak}(p, n - i))$ 
return  $q$ 
```

houthakken: dynamic programming algoritme

Algorithm houthakken(p, n):

```
new array  $b[0 \dots n]$ 
 $b[0] := 0$ 
for  $j := 1$  to  $n$  do
   $q := -\infty$ 
  for  $i := 1$  to  $j$  do
     $q := \max(q, p[i] + b[j - i])$ 
   $b[j] := q$ 
return  $b[n]$ 
```

houthakken: tijdscomplexiteit

- bekijk voorbeeld met $n = 4$
- bekijk subprobleem graaf
- $T(n) = n^2$

maximaal subarray: voorbeeld

vind de maximale som van elementen van een subarray

$[-10, 10, 5, -3, 2, 1]$

levert hier 15

schema

- recap en intro
- hout hakken
- maximaal subarray
- matrix vermenigvuldiging
- fractional knapsack
- knapsack 0/1
- task scheduling
- materiaal

maximaal subarray: naief algoritme in $\mathcal{O}(n^2)$

```
Algorithm maxSubArray( $A, n$ ):  
   $max := 0$   
  for  $left := 0$  to  $n - 1$  do  
     $sum := 0$   
    for  $right := left$  to  $n - 1$  do  
       $sum := sum + A[right]$   
      if  $sum > max$  then  
         $max := sum$   
  return  $max$ 
```

maximaal subarray: dynamic programming

- $B[r]$:
maximale som van subarray dat eindigt op rank r
- $B[0] = \max\{A[0], 0\}$
- $B[r] = \max\{0, B[r-1] + A[r]\}$
of lege subarray dat eindigt op r
of maximaal subarray dat eindigt op $r-1$ met r erbij
- maximaal subarray van geheel:
maximum van de $B[i]$'s

schema

- recap en intro
- hout hakken
- maximaal subarray
- matrix vermenigvuldiging
- fractional knapsack
- knapsack 0/1
- task scheduling
- materiaal

maximaal subarray: algoritme

door Kadane, in $\mathcal{O}(n)$

Algorithm $\text{maxSubarray}(A, n)$:

Input: array A containing n integers

Output: maximum subarray sum

$B =$ new array of length n

$B[0] = \max(A[0], 0)$

$max = B[0]$

for $r = 1$ **to** $n - 1$ **do**

$B[r] = \max(0, B[r-1] + A[r])$

$max = \max(max, B[r])$

done

return max

matrix vermenigvuldiging

- matrix vermenigvuldiging is associatief dwz $(AB)C = A(BC)$
je mag de haakjes zetten zoals je wilt
- het aantal elementaire stappen hangt af van hoe de haakjes staan

$$A = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad B = (1 \ 1 \ 1) \quad C = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

- $(AB)C$ levert $6 + 6$ vermenigvuldigingsstappen,
 $A(BC)$ levert $2 + 3$ vermenigvuldigingsstappen
- hoe vinden we een beste haakjes-configuratie?

matrix vermenigvuldiging: opsommen

- bekijk alle haakjes-mogelijkheden
- bereken per mogelijkheid het aantal operaties
- kies dan een beste haakjes-configuratie

MAAR

- **duur:**
aantal mogelijkheden is Catalan nummer van aantal matrices
exponentieel, bijna 4^n

matrix vermenigvuldigen: dynamic programming

- **probleem:**
geef beste haakjes voor $A_0 \cdot \dots \cdot A_{n-1}$
dimensie van A_i is $d_i \times d_{i+1}$
- $N_{i,j}$:
beste haakjes voor $A_i \cdot \dots \cdot A_j$
- **voorbeeld:**
stel de beste haakjes
 $(A_0 \cdot \dots \cdot A_i) \cdot (A_{i+1} \cdot \dots \cdot A_{n-1})$
dan is het aantal operates $N_{0,i} + N_{i+1,n-1} + 1$
- **definitie $N_{i,j}$:**

$$N_{i,j} = \min_{i \leq k < j} \{N_{i,k} + N_{k+1,j} + d_i \cdot d_{k+1} \cdot d_{j+1}\}$$

- **uitleg dimensies:**
 $A_i \cdot \dots \cdot A_k : d_i \times d_{k+1}$
 $A_{k+1} \cdot \dots \cdot A_j : d_{k+1} \times d_{j+1}$

matrix vermenigvuldigen: greedy

- kies steeds product met minimaal aantal operaties
- levert niet perse optimale oplossing

$$A = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 1 \end{pmatrix} \quad C = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

- dan AB kost 4 en BC kost 6
- toch is $A(BC)$ met totaal 12 beter dan $(AB)C$ met totaal 16

matrix vermenigvuldigen: algoritme

eerst $N_{i,i} = 0$, dan $N_{i,i+s-1}$

matrixChain(d_0, d_1, \dots, d_{n-1}):

for $i = 0$ to $n - 1$ do $N[i, i] = 0$ done

for $s = 2$ to n do

for $i = 0$ to $n - s$ do

$j = i + s - 1$

$N[i, j] = +\infty$

for $k = i$ to $j - 1$ do

$ops = N[i, k] + N[k + 1, j] + d_i \cdot d_{k+1} \cdot d_{j+1}$

$N[i, j] = \min(N[i, j], ops)$

done

done

done

toepassing

- $d_0 = 2, d_1 = 1, d_2 = 2, d_3 = 4, d_4 = 3.$
- A_0 is $2 \times 1, A_1$ is $1 \times 2, A_2$ is $2 \times 4, A_3$ is $4 \times 3.$
- $N[0, 0] = 0, N[1, 1] = 0, N[2, 2] = 0, N[3, 3] = 0$
- $N[0, 1] = 4, N[1, 2] = 8, N[2, 3] = 24$
- $N[0, 2] = \min(0 + 8 + 8, 4 + 0 + 16) = 16,$
 $N[1, 3] = \min(0 + 24 + 6, 8 + 0 + 12) = 20$
- $N[0, 4] = \min(0 + 20 + 6, 4 + 24 + 12, 16 + 0 + 24) = 26$
- optimaal: $A_0 \cdot ((A_1 \cdot A_2) \cdot A_3).$

fractional knapsack: voorbeeld

- 50 kilo juwelen, waarde 1 miljoen euro
- 1 kilo kauwgum, waarde 20 euro
- 5 kilo diamanten, waarde 5 miljoen euro
- 10 kilo goud, waarde 500000 euro
- rugzak voor 20 kilo! wat nu?

we nemen natuurlijk zoveel mogelijk waarde per gewicht

- diamanten: 1 miljoen euro/kilo
- goud: 0.05 miljoen euro/kilo
- juwelen: 0.02 miljoen euro/kilo
- kauwgum: veel minder

schema

- recap en intro
- hout hakken
- maximaal subarray
- matrix vermenigvuldiging
- fractional knapsack
- knapsack 0/1
- task scheduling
- materiaal

fractional knapsack: voorbeeld

- 5 kilo diamanten, waarde 1 miljoen euro/kilo
neem 5 kilo
- 10 kilo goud, waarde 0.05 miljoen euro/kilo
neem 10 kilo, dan totaal 15 kilo
- 50 kilo juwelen, waarde 0.02 miljoen euro/kilo
neem 5 kilo, dan totaal 20 kilo
- totale waarde: 5.6 miljoen euro

fractional knapsack: probleem

- **gegeven:**
verzameling S met n items,
elk item i heeft gewicht w_i en benefit b_i
maximaal totaalgewicht W
- **doel:**
geef $0 \leq x_i \leq w_i$ voor elke $i \in S$ zodat
 $\sum_{i \in S} b_i \cdot \frac{x_i}{w_i}$ maximaal
maar onder constraint $\sum_{i \in S} x_i \leq W$

fractional knapsack: correctheid algoritme

bekijk een niet-greedy algoritme A ; dan zijn er i en j met

- $v_j < v_i$
de relatieve waarde van i is groter dan die van j
- $x_i < w_i$
toch kiest A niet alles van i
- $x_j > 0$
terwijl A wel wat van het (minder waardevolle) j kiest

maar dan levert A geen optimale oplossing, want:

- $a = \min(x_j, w_i - x_i)$
vervang deel a van j door hetzelfde deel van het betere i !
(of alles x_j , of zoveel als nog over is van i , dus $w_i - x_i$)

fractional knapsack: algoritme

- **greedy choice:**
we kiezen steeds item met maximale $\frac{b_i}{w_i}$
- **algoritme in $\mathcal{O}(n \log n)$:**
fractionalKnapsack(S, \vec{b}, \vec{w}, W):
 for each $i \in S$ **do**
 $x_i = 0$
 $v_i = b_i/w_i$
 sort S such that elements are descending w.r.t. v_i
 while $W > 0$ **do**
 $i = S.\text{remove}(S.\text{first}())$
 $x_i = \min(w_i, W)$
 $W = W - x_i$
 done

schema

- recap en intro
- hout hakken
- maximaal subarray
- matrix vermenigvuldiging
- fractional knapsack
- knapsack 0/1
- task scheduling
- materiaal

knapsack 0/1: voorbeeld

- 7 kilo goudstuk
- 5 kilo goudstuk
- 4 kilo goudstuk
- rugzak voor 10 kilo! wat nu?
- de greedy aanpak levert 7 dus niet optimaal

knapsack 0/1: idee algoritme

- S_k bevat elementen $1, \dots, k$
- $B[k, w]$ is beste selectie uit S_k met totaalgewicht w
- hoe vind je $B[k, w]$?

als $w_k > w$: item k kan er niet bij
dus $B[k, w] = B[k - 1, w]$

als $w_k \leq w$: item k kan (maar hoeft niet) erbij
dan $B[k, w] = \max\{B[k - 1, w], B[k - 1, w - w_k] + b_k\}$

knapsack 0/1: probleem

- **gegeven:**
verzameling S met n items
elk item i heeft gewicht w_i en benefit b_i
maximaal totaalgewicht W
- **doel:**
kies items $T \subseteq S$ zodat
 $\sum_{i \in T} b_i$ maximaal
onder constraint $\sum_{i \in T} w_i \leq W$
- **naieve aanpak:**
bekijk alle 2^n deelverzamelingen van S (hmm)
- greedy werkt niet (ook niet met gewogen waarden)

knapsack 0/1: idee algoritme

- voor elke $k = 0, 1, \dots, n$ bekijken we S_k
- voor elke S_k bekijken we $w = 0, 1, \dots, W$
- in $\mathcal{O}(nW)$
stel $W = 2^n$ dan niet zo gunstig
- een pseudo-polynomiaal algoritme-idee
- wordt onwaarschijnlijk geacht dat iemand een polynomiaal algoritme vindt

knapsack 0/1: voorbeeld toepassing

- item 1 met $w_1 = 3$ en $b_1 = 9$
- item 2 met $w_2 = 2$ en $b_2 = 5$
- item 3 met $w_3 = 2$ en $b_3 = 5$
- maximaal totaalgewicht $W = 4$

we runnen het algoritme:

- $B[0,0] = 0, B[0,1] = 0, B[0,2] = 0, B[0,3] = 0, B[0,4] = 0$
- $B[1,0] = 0, B[1,1] = 0, B[1,2] = 0, B[1,3] = 9, B[1,4] = 9$
- $B[2,0] = 0, B[2,1] = 0, B[2,2] = 5, B[2,3] = 9, B[2,4] = 9$
- $B[3,0] = 0, B[3,1] = 0, B[3,2] = 5, B[3,3] = 9, B[3,4] = 10$

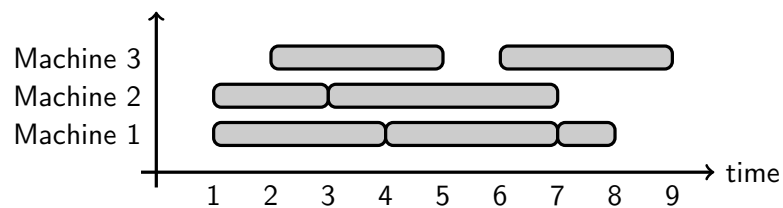
task scheduling: voorbeeld

doe alle taken op zo min mogelijk machines

taken:

$[1,4], [1,3], [2,5], [3,7], [4,7], [6,9], [7,8]$

indeling:

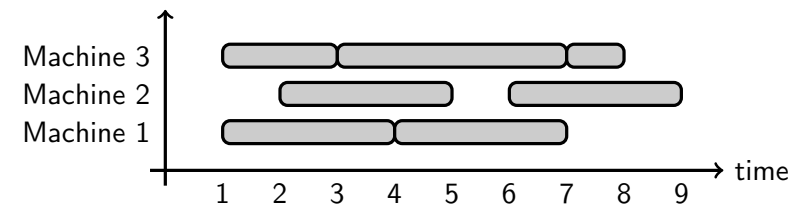


schema

- recap en intro
- hout hakken
- maximaal subarray
- matrix vermenigvuldiging
- fractional knapsack
- knapsack 0/1
- task scheduling
- materiaal

task scheduling

- gegeven: verzameling T van taken met begin- en eind-tijd
- schedule alle taken op zo min mogelijk machines



task scheduling: algoritme

- greedy choice:
kies steeds taak met kleinste begin-tijd
- neem alleen nieuwe machine erbij als echt nodig

taskSchedule(T, \vec{s}, \vec{f}):

$m = 0$ number of machines

sort T such that elements are ascending w.r.t. s_i

while $\neg T.isEmpty()$ **do**

$i = T.remove(T.first())$

if a machine $j \leq m$ has time for task i **then**

 schedule i on machine j

else

$m = m + 1$

 schedule i on machine m

done

scheduling tijd

- n klanten bij postkantoor, elk met geholpen-tijd t_i
- doel: minimaliseer aanwezig-tijd van alle klanten samen
- hoeveel verschillende volgordes zijn er?
- geef greedy algoritme dat een beste volgorde geeft

task scheduling: correctheid algoritme

- stel algoritme gebruikt m machines,
- er is een schedule met minder dan m machines
- bekijk toekenning van machine m door algoritme
- laat geen ruimte voor alternatieven

scheduling met deadlines

elke taak i heeft:

- een deadline d_i
- een benefit b_i
- duur 1

vraag: geef een uitvoerbare verzameling taken met maximale totale benefit

schema

- recap en intro
- hout hakken
- maximaal subarray
- matrix vermenigvuldiging
- fractional knapsack
- knapsack 0/1
- task scheduling
- **materiaal**

materiaal

- boek 5.3, 5.1