Tiende college algoritmiek

18 april 2023

Dynamisch programmeren

Gretige Algoritmen

Algoritme van Dijkstra

Laat $F[i][j]$ de waarde zijn van de meest waardevolle deelverzameling van de eerste i ($0 \leq i \leq n$) objecten, die in een knapzak met capaciteit j ($0 \leq j \leq W$) past. We zoeken dus $F[n][W]$. We nemen hier impliciet aan dat W een positief geheel getal is.

Dan geldt (want object i zit er wel of niet in):

$$F[i][j] = \begin{cases} \max\{F[i-1][j], v_i + F[i-1][j-w_i]\} & \text{als } j \geq w_i \\ F[i-1][j] & \text{als } j < w_i \end{cases}$$

En we definiëren:

$$F[0][j] = 0 \text{ voor } j \geq 0 \text{ en } F[i][0] = 0 \text{ voor } i \geq 0$$

Gegeven onbeperkt veel munten van $d_1, d_2, \ldots d_m$ eurocent, en een te betalen bedrag van n ($n \geq 0$) eurocent. Alle d_i zijn > 0 en verschillend. **Gevraagd**: het minimale aantal munten dat nodig is om het bedrag van n eurocent te betalen.

Voorbeeld:

type munt	waarde
1	1
2	4
3	6

te betalen bedrag: 8

Vier manieren om te betalen: 6 + 1 + 1; 4 + 4; 4 + 1 + 1 + 1 + 1; 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1. Dus het gevraagde minimale aantal is: 2 (twee munten van 4 cent).

KZP	MP
n objecten	m munten
gewicht w_i	waarde d_i
totaal gewicht $\leq$ capaciteit W	totale waarde $=$ bedrag n
waarde v_i	'kosten' 1
max. totale waarde	min. totale 'kosten'
elk object $\leq$ 1 keer	munt mag meer keer

Laat $\texttt{munt}[i][j]$ het minimale aantal munten zijn dat nodig is om een bedrag van j eurocent te betalen, wanneer alleen munten van $d_1, d_2, \ldots, d_i$ $(i \geq 1)$ worden gebruikt. We zoeken dus $\texttt{munt}[m][n]$.

Dan geldt (want d_i wordt wel of niet gebruikt):

$$\texttt{munt}[i][j] = \begin{cases} \ldots & \text{als } i > 1, j \geq d_i \\ \ldots & \text{als } i > 1, 0 < j < d_i \\ \ldots & \text{als } i = 1, 0 < j < d_1 \\ \ldots & \text{als } i = 1, j \geq d_1 \\ \ldots & \text{als } i \geq 1, j = 0 \end{cases}$$

Het kan eenvoudiger, want

KZP	MP
n objecten	m munten
gewicht w_i	waarde d_i
totaal gewicht $\leq$ capaciteit W	totale waarde $=$ bedrag n
waarde v_i	'kosten' 1
max. totale waarde	min. totale 'kosten'
elk object $\leq$ 1 keer	munt mag meer keer

Bij muntenprobleem dus geen noodzaak om bij te houden welke munten we al bekeken hebben.

Laat $\texttt{munt}[j]$ het minimale aantal munten zijn dat nodig is om een bedrag van j eurocent te betalen. We zoeken dus $\texttt{munt}[n]$.
Neem voor het gemak even aan dat de muntsoorten oplopend zijn gesorteerd ($d_1 < d_2 < \ldots < d_m$).

Dan geldt:

$$\texttt{munt}[j] = \begin{cases} \ldots & \text{als } j \geq d_1 \\ \ldots & \text{als } 0 < j < d_1 \\ \ldots & \text{als } j = 0 \end{cases}$$

Voorbeeld:

type munt	waarde
1	4
2	6

te betalen bedrag: 8

Laat $\texttt{munt}[j]$ het minimale aantal munten zijn dat nodig is om een bedrag van j eurocent te betalen. We zoeken dus $\texttt{munt}[n]$.
Neem voor het gemak even aan dat de muntsoorten oplopend zijn gesorteerd ($d_1 < d_2 < \ldots < d_m$).

Dan geldt:

$$\texttt{munt}[j] = \begin{cases} \min_{d_i \leq j} \{1 + \texttt{munt}[j - d_i]\} & \text{als } j \geq d_1 \\ \infty & \text{als } 0 < j < d_1 \\ 0 & \text{als } j = 0 \end{cases}$$

Vul array `munt` van links naar rechts.

```
munt[0] := 0;
for j := 1 to n do
    tmp := ∞;
    i := 1;
    while i ≤ m and d_i ≤ j do
        if 1 + munt[j − d_i] < tmp then
            tmp := 1 + munt[j − d_i];
        fi
        i++;
    od
    munt[j] := tmp;
od
```

Complexiteit MP met 1-d DP:
tijd $\Theta(m * n)$; extra geheugen: $\Theta(n)$
(Net als MP met 2-d DP (met eendimensionaal array))

Voorbeeld:

type munt	waarde
1	1
2	4
3	6

te betalen bedrag: 8

j	0	1	2	3	4	5	6	7	8
munt$[j]$	0	1	2	3	1	2	1	2	?

Voorbeeld:

type munt	waarde
1	1
2	4
3	6

te betalen bedrag: 8

j	0	1	2	3	4	5	6	7	8
munt[j]	0	1	2	3	1	2	1	2	?

j	0	1	2	3	4	5	6	7	8
munt[j]	0	1	2	3	1	2	1	2	2

Vind benodigde munten terug in tabel:

j	0	1	2	3	4	5	6	7	8
munt[j]	0	1	2	3	1	2	1	2	2

1. Een (eenvoudige) variatie is: gegeven een bedrag van n euro, is dat te betalen met muntsoorten $d_1, \ldots d_m$? Dit kan geheel analoog aan het optimalisatieprobleem worden opgelost met DP. Gebruik een array `munt`, waarbij `munt`$[j]$ = True als het bedrag j gemaakt kan worden, en anders False.

Vul array `munt` van links naar rechts.

```
munt[0] := 0;
for j := 1 to n do
    tmp := ∞;
    i := 1;
    while i ≤ m and d_i ≤ j do
        if 1 + munt[j − d_i] < tmp then
            tmp := 1 + munt[j − d_i];
        fi
        i++;
    od
    munt[j] := tmp;
od
```

Complexiteit MP met 1-d DP:
tijd $\Theta(m * n)$; extra geheugen: $\Theta(n)$
Net als MP met 2-d DP (met eendimensionaal array)

Vul array `munt` van links naar rechts.

```
munt[0] := true;
for j := 1 to n do
    tmp := false;
    i := 1;
    while i ≤ m and d_i ≤ j and not tmp do
        if munt[j − d_i] then
            tmp := true;
        fi
        i++;
    od
    munt[j] := tmp;
od
```

Complexiteit MP met 1-d DP:
tijd $\Theta(m * n)$; extra geheugen: $\Theta(n)$
Net als MP met 2-d DP (met eendimensionaal array)

1. Een (eenvoudige) variatie is: gegeven een bedrag van n euro, is dat te betalen met muntsoorten $d_1, \ldots d_m$? Dit kan geheel analoog aan het optimalisatieprobleem worden opgelost met DP. Gebruik een array `munt`, waarbij `munt`$[j]$ = True als het bedrag j gemaakt kan worden, en anders False.

2. Een ander algoritme voor het muntenprobleem:

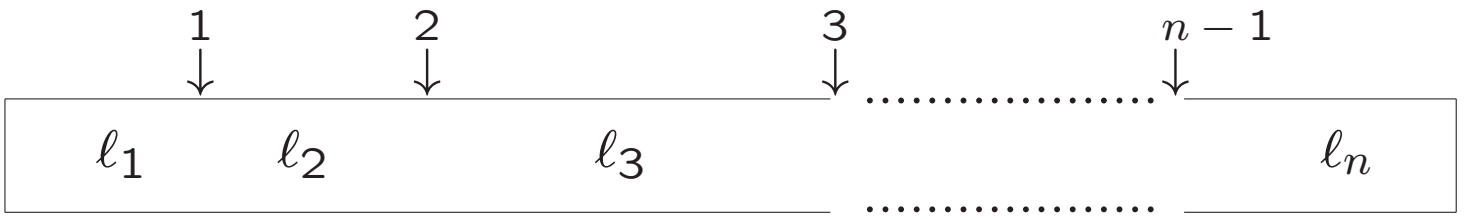
   betaal n met $d_1, \ldots d_m$::

   geef de grootste munt $d_i \leq n$;

   betaal $n - d_i$ met $d_1, \ldots d_i$

   Dit is een zogenaamd gretig algoritme. Bovenstaand algoritme is erg snel, maar het levert niet altijd een optimale oplossing (soms ook geen oplossing, terwijl er wel een oplossing is).

Een houtzaagmolen rekent voor het in twee stukken zagen van een stam van lengte ℓ precies ℓ euro, ongeacht de plek waar dit moet gebeuren. Na bestudering van de knoesten op een boomstam van lengte ℓ wordt besloten dat deze in achtereenvolgens (v.l.n.r. gezien) stukken van lengtes $\ell_1, \ell_2, \ldots, \ell_n$ gezaagd moet worden. (De hele boomstam heeft dus lengte $\sum_{i=1}^{n} \ell_i$.) De plekken waar gezaagd gaat worden zijn dus van tevoren bekend. Er zijn hier $n-1$ zaagplekken.



Merk op dat de volgorde van zagen van invloed is op de prijs.

Voorbeeld

Laat $n = 4$ en $\ell_1 = 6, \ell_2 = 8, \ell_3 = 7, \ell_4 = 2$. De boomstam heeft dus lengte 23.

- Stel we zagen achtereenvolgens op plek 1, dan plek 2 en dan plek 3. De kosten zijn dan $23 + 17 + 9 = 49$ euro.
- Stel we zagen achtereenvolgens op plek 3, dan plek 2 en dan plek 1. De kosten zijn dan $23 + 21 + 14 = 58$ euro.

Probleem

Bepaal de minimale kosten om de gegeven boomstam in stukken met de opgegeven lengtes ℓ_i te zagen (zaagplekken dus bekend).

Deelproblemen

Het probleem brengen we terug tot het bepalen van de minimale kosten $Z[i][j]$ die moeten worden gemaakt om de (deel)stam van stukken i t/m j, ter lengte $L(i,j) = \ell_i + \ell_{i+1} + \cdots + \ell_j$ te verzagen tot achtereenvolgens stukken van lengte $\ell_i, \ell_{i+1}, \ldots, \ell_j$. Alle ℓ_i, en dus ook alle $L(i,j)$ en alle zaagplekken, zijn gegeven. Het oorspronkelijke probleem is dan het bepalen van $Z[1][n]$. Merk op dat altijd $1 \leq i \leq j \leq n$.

Recurrente betrekking

$$Z[i][j] = \ldots$$

Deelproblemen

Het probleem brengen we terug tot het bepalen van de minimale kosten $Z[i][j]$ die moeten worden gemaakt om de (deel)stam van stukken i t/m j, ter lengte $L(i,j) = \ell_i + \ell_{i+1} + \cdots + \ell_j$ te verzagen tot achtereenvolgens stukken van lengte $\ell_i, \ell_{i+1}, \ldots, \ell_j$. Alle ℓ_i, en dus ook alle $L(i,j)$ en alle zaagplekken, zijn gegeven. Het oorspronkelijke probleem is dan het bepalen van $Z[1][n]$. Merk op dat altijd $1 \leq i \leq j \leq n$.

Recurrente betrekking

$$Z[i][j] = \begin{cases} L(i,j) + \min_{i \leq k \leq j-1}\{Z[i][k] + Z[k+1][j]\} & \text{als } i < j \\ 0 & \text{als } i = j \end{cases}$$

De $Z[i][j]$ op plek # wordt berekend uit Z-waarden op de plekken met een *; dus uit dezelfde rij en dezelfde kolom.

	j	$\rightarrow$										
i	0	.	.	.	.	.	.	.	.	.	.	.
$\downarrow$		0	.	.	.	.	.	.	.	.	.	.
			*	*	*	*	*	*	*	#	.	.
				0	.	.	.	.	.	*	.	.
					0	.	.	.	.	*	.	.
						0	.	.	.	*	.	.
							0	.	.	*	.	.

Invulvolgorde. . .

De $Z[i][j]$ op plek # wordt berekend uit Z-waarden op de plekken met een *; dus uit dezelfde rij en dezelfde kolom.

	j	$\rightarrow$										
i	0	.	.	.	.	.	.	.	.	.	.	.
$\downarrow$		0	.	.	.	.	.	.	.	.	.	.
			*	*	*	*	*	*	*	#	.	.
				0	.	.	.	.	.	*	.	.
					0	.	.	.	.	*	.	.
						0	.	.	.	*	.	.
							0	.	.	*	.	.

Invulvolgorde

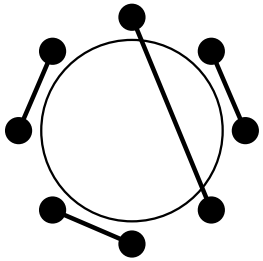
De tabel kan bottom up gevuld worden door alle diagonalen $j = i + d$ af te lopen en per diagonaal bijvoorbeeld van linksboven tot rechtsonder te gaan. Een andere mogelijkheid is de tabel rij voor rij te vullen (van onder naar boven) en per rij (verplicht) van links naar rechts.

```
void vulkosten ( int n ) {  // L en Z globaal
   int i, j;
   for (i = 1; i <= n; i++)
      Z[i][i] = 0;
   for (i = n-1; i > 0; i--) {
      for (j = i+1; j <= n; j++ ) {
         min = Z[i][i] + Z[i+1][j];
         for (k=i+1; k<j; k++) {
            if ( Z[i][k] + Z[k+1][j] < min )
                min = Z[i][k] + Z[k+1][j];
         }  // min bevat nu het minimum
         Z[i][j] = L[i][j] + min;
      }  // for j
   }  for i
}  // vulkosten
```

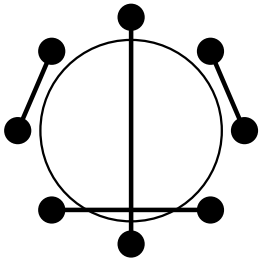
The Bavarian Beer Party (BAPC 2006)

We hebben een kring van n studenten (n even), waarvan de studierichting bekend is. Iedereen moet precies één andere student de hand schudden. Dit handen schudden moet zodanig gebeuren dat geen enkel tweetal armen elkaar kruist.

Voorbeeld met acht personen:



Geen kruisende armen:
toegestaan



Kruisende armen:
niet toegestaan

De bedoeling is om het aantal paren studenten met dezelfde studierichting dat elkaar een hand geeft te maximaliseren.

Oplossing:

Nummer de studenten met de klok mee van 1 t/m n.

Student 1 moet de hand schudden met student. . .

Oplossing:

Nummer de studenten met de klok mee van 1 t/m n.

Student 1 moet de hand schudden met student 2, student 4, student 6, ..., of student n. Zeg student k.

Vervolgens moeten studenten 2 t/m $k-1$ met elkaar handen schudden, en moeten studenten $k+1$ t/m n met elkaar handen schudden.

Oplossing:

Nummer de studenten met de klok mee van 1 t/m n.

Student 1 moet de hand schudden met student 2, student 4, student 6, ..., of student n. Zeg student k.

Vervolgens moeten studenten 2 t/m $k-1$ met elkaar handen schudden, en moeten studenten $k+1$ t/m n met elkaar handen schudden.

Alsof je de tafel 'openvouwt' tussen student n en 1. Een toegestane koppeling van studenten die elkaar de hand schudden komt nu overeen met een rijtje van n corresponderende haakjes: $\frac{n}{2}$ openingshaakjes en $\frac{n}{2}$ sluithaakjes.

Op positie 1 staat een openingshaakje; het corresponderende sluithaakje staat op positie 2, positie 4, positie 6, ..., of positie n. Zeg positie k.

Vervolgens moet je corresponderende haakjes plaatsen op posities 2 t/m $k-1$, en corresponderende haakjes plaatsen op posities $k+1$ t/m n.

Laat $Z[i][j] = 1$ als studenten i en j de **Z**elfde studierichting volgen, en $Z[i][j] = 0$ anders. Laat $M[i][j]$ het **M**aximale aantal koppels studenten zijn dat elkaar de hand schudt, én dezelfde studierichting volgt, als we studenten $i, \ldots, j$ op een toegestane manier aan elkaar koppelen. We zoeken dus $M[1][n]$.

Dan geldt:

$$M[i][j] = \ldots$$

Laat $Z[i][j] = 1$ als studenten i en j de **Z**elfde studierichting volgen, en $Z[i][j] = 0$ anders. Laat $M[i][j]$ het **M**aximale aantal koppels studenten zijn dat elkaar de hand schudt, én dezelfde studierichting volgt, als we studenten $i, \ldots, j$ op een toegestane manier aan elkaar koppelen. We zoeken dus $M[1][n]$.

Dan geldt:

$$M[i][j] = \begin{cases} \max_{k=i+1,i+3,\ldots,j} \{Z[i][k] + M[i+1][k-1] + M[k+1][j]\} & \text{als } i \leq j \text{ en } j-i \text{ is oneven } (i \text{ schudt hand met } k) \\ 0 & \text{als } i \leq j \text{ en } j-i \text{ is even} \\ 0 & \text{als } i > j \end{cases}$$

Vraag: waarom proberen we in de bovenste regel alleen $k = i+1, i+3, i+5, \ldots, j$?

Iets andere formulering recurrente betrekking zagen:

$$Z[i][j] = \begin{cases} \min_{i \le k \le j-1}\{L(i,j) + Z[i][k] + Z[k+1][j]\} & \text{als } i < j \\ 0 & \text{als } i = j \end{cases}$$

Handen schudden:

$$M[i][j] = \begin{cases} \max_{k=i+1,i+3,\ldots,j} \{Z[i][k] + M[i+1][k-1] + M[k+1][j]\} \\ \quad \text{als } i \le j \text{ en } j-i \text{ is oneven } (i \text{ schudt hand met } k) \\ 0 \quad \text{als } i \le j \text{ en } j-i \text{ is even} \\ 0 \quad \text{als } i > j \end{cases}$$

Gegeven onbeperkt veel munten van $d_1, d_2, \ldots d_m$ eurocent, en een te betalen bedrag van n ($n \geq 0$) eurocent. Alle d_i zijn > 0 en verschillend.

Gevraagd: het minimale aantal munten dat nodig is om het bedrag van n eurocent te betalen.

Voorbeeld:

type munt	waarde
1	1
2	4
3	6

te betalen bedrag: 8

Vier manieren om te betalen: 6 + 1 + 1; 4 + 4; 4 + 1 + 1 + 1 + 1; 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1. Dus het gevraagde minimale aantal is: 2 (twee munten van 4 cent).

Een gretige strategie (recursief geformuleerd):

betaal n met $d_1, \ldots d_m$ (voor het gemak oplopend gesorteerd)::

if $(n = 0)$ klaar;

else geef de grootste munt $d_i \leq n$ (restrictie);

// dan is het nog te betalen bedrag zo klein mogelijk

// en dus heb je zo weinig mogelijk munten

// nodig (hoop je)

betaal $n - d_i$ met $d_1, \ldots d_i$.

Bovenstaand algoritme is erg eenvoudig en snel, en levert voor het geval van de gebruikelijke euro-munten (munten van 1, 2, 5, 10, 20, 50, 100, 200 eurocent) het optimale antwoord. Dit is echter niet het geval voor de muntwaarden uit het voorbeeld. (Bovendien gaat dit algoritme ervan uit dat het bedrag te betalen is. Anders is nog een kleine aanpassing nodig.)

- Greed = hebzucht
- Voor oplossen van optimalisatieproblemen
  (of loop risico dat niet aan alle eisen voldaan is)
- Oplossing wordt stap voor stap opgebouwd
- In elke stap wordt een gretige keuze gemaakt waarmee de huidige deeloplossing wordt uitgebreid
- Dat wil zeggen: een (locale) keuze die op dat moment de beste lijkt (de grootste directe winst oplevert)
- De vraag is of dat leidt tot een globaal optimale oplossing

De oplossing wordt dus opgebouwd via een serie achtereenvolgende gretige keuzes. Deze keuzes

- zijn consistent met de restricties van het probleem
- zijn locaal optimaal, d.w.z. de best uitziende keuze in die stap
- zijn onherroepelijk: keuzes kunnen niet meer worden teruggedraaid

Een gretig algoritme ziet er dus ruwweg zo uit:

while nog niet alle stappen zijn gedaan **do**
    doe een keuze die in eerste instantie de grootste winst lijkt op te leveren
od

Uitbreiden van deeloplossingen moet uiteraard wel steeds in overeenstemming met de geldende restricties.

Soms leveren gretige algoritme een optimale oplossing, en soms/vaak niet. In dat geval is de gretige strategie een heuristiek, die bijvoorbeeld leidt tot een goede, maar meestal niet optimale oplossing. Of: de gretige strategie leidt vaak, maar niet altijd, tot een optimale oplossing.

- Optimale oplossing
  - Muntenprobleem voor de gebruikelijke euromunten
  - Sommige planningsproblemen
  - Kortste paden in een graaf (Dijkstra)
  - Minimale opspannende boom (Prim, Kruskal)
  - ...
- Benadering
  - Handelsreizigersprobleem
  - Knapzakprobleem
  - ...

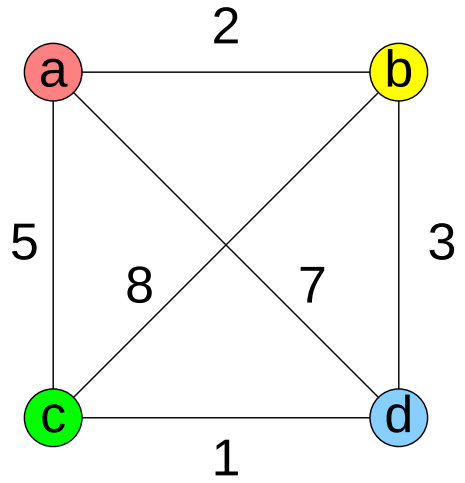
Traveling Salesman Problem (handelsreizigersprobleem)

Gegeven n steden waarvan alle onderlinge afstanden bekend zijn.

Gevraagd: de/een kortste route die elke stad precies één keer aandoet, en weer terugkeert in het vertrekpunt.

Ofwel: vind de/een kortste Hamiltonkring in een samenhangende gewogen (volledige) graaf.

Voorbeeld:



Knapzakprobleem

Gegeven n objecten, met gewicht $w_1, \ldots, w_n$ en waarde $v_1, \ldots, v_n$, en een knapzak met capaciteit W.

Gevraagd: de meest waardevolle deelverzameling der objecten die in de knapzak past (dus met totaalgewicht $\leq W$).

Voorbeeld:

object	gewicht	waarde
1	8	42
2	3	14
3	4	40
4	5	27

knapzakcapaciteit 12

Knapzakprobleem

Gegeven n objecten, met gewicht $w_1, \ldots, w_n$ en waarde $v_1, \ldots, v_n$, en een knapzak met capaciteit W.

Gevraagd: de meest waardevolle deelverzameling der objecten die in de knapzak past (dus met totaalgewicht $\leq W$).

Voorbeeld:

object	gewicht	waarde	w/g
1	8	42	5.25
2	3	14	4.67
3	4	40	10
4	5	27	5.4

knapzakcapaciteit 12

Zie ook Levitin, Exercise 9.1.3.

Gegeven n jobs, genummerd 1 t/m n, die allemaal na elkaar door één processor moeten worden uitgevoerd. Van elke job i is de executietijd t_i gegeven. De jobs moeten zo na elkaar worden gepland dat de totale hoeveelheid tijd in het systeem van alle jobs samen wordt geminimaliseerd. De tijd die job i in het systeem doorbrengt is de wachttijd + de executietijd t_i.

We willen dus

$$T = \sum_{i=1}^{n} \texttt{(tijd die job i in het systeem doorbrengt)}$$

minimaliseren.

De waarde van T hangt af van de volgorde waarin de jobs door de processor worden uitgevoerd.

Voorbeeld: $n = 3$; Jobs: $1, 2, 3$ met $t_1 = 5, t_2 = 10, t_3 = 3$.

volgorde	T
1 2 3	$5 + (5 + 10) + (5 + 10 + 3) = 38$
1 3 2	$5 + (5 + 3) + (5 + 3 + 10) = 31$
2 1 3	$10 + (10 + 5) + (10 + 5 + 3) = 43$
2 3 1	$10 + (10 + 3) + (10 + 3 + 5) = 41$
3 1 2	$3 + (3 + 5) + (3 + 5 + 10) = 29$
3 2 1	$3 + (3 + 10) + (3 + 10 + 5) = 34$

Alle volgordes aflopen. . .

Idee voor een gretige oplossing: kies in elke stap de job met de kleinste executietijd van de nog resterende jobs. Door die keuze houd je de wachttijd voor de overige jobs op dat moment (tot de volgende job aan de beurt is) zo klein mogelijk.

Voor het voorbeeld levert deze gretige strategie de optimale oplossing.

Idee voor een gretige oplossing: kies in elke stap de job met de kleinste executietijd van de nog resterende jobs. Door die keuze houd je de wachttijd voor de overige jobs op dat moment (tot de volgende job aan de beurt is) zo klein mogelijk.

Voor het voorbeeld levert deze gretige strategie de optimale oplossing.

Observatie.
Stel dat de jobs in de volgorde $i_1, i_2, \ldots, i_n$ worden uitgevoerd. Dan is

$$T = t_{i_1} + (t_{i_1} + t_{i_2}) + (t_{i_1} + t_{i_2} + t_{i_3}) + \ldots + (t_{i_1} + t_{i_2} + \ldots + t_{i_n})$$

$$= n * t_{i_1} + (n-1) * t_{i_2} + \ldots + 2 * t_{i_{n-1}} + t_{i_n}$$

Algoritme

Sorteer de jobs in oplopende volgorde van hun executietijd;
// Dit is de optimale volgorde om de jobs uit te voeren

Complexiteit

Sorteren kan in $O(n \lg n)$ stappen, dus dit algoritme ook.

Correctheid

Dit algoritme levert altijd een optimale oplossing. Dit moet wel expliciet bewezen worden.
Hiertoe laten we het volgende zien. Stel dat de volgorde van uitvoering zo is dat er twee jobs $a := i_k$ en $b := i_{k+1}$ zijn met $t_a > t_b$ (dus b wordt direct na a uitgevoerd maar heeft kortere executietijd). Dan krijg je een betere oplossing door de volgorde van deze twee jobs om te keren.

Gegeven een verzameling $A = \{1, 2, \ldots, n\}$ met activiteiten die allemaal gebruik willen maken van een of andere "resource" (bijvoorbeeld een collegezaal). Deze resource kan maar door één activiteit tegelijk gebruikt worden. Activiteit i vindt plaats gedurende het tijdsinterval $[b_i, e_i)$. De begin- en eindtijden b_i en e_i zijn voor alle activiteiten bekend.

Definitie: activiteiten i en j heten compatibel als $[b_i, e_i)$ en $[b_j, e_j)$ niet overlappen, dus als $e_i \leq b_j$ of $e_j \leq b_i$.

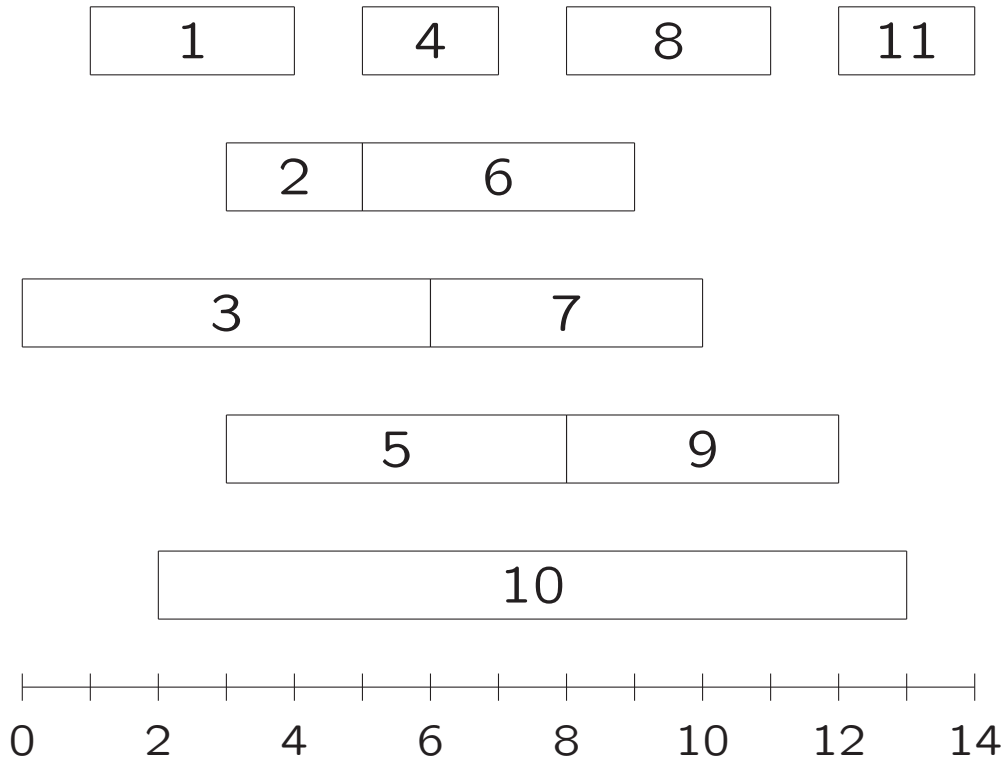
Opdracht: vind een zo groot mogelijke deelverzameling van paarsgewijs compatibele activiteiten uit A.

Alle deelverzamelingen aflopen...

i	b_i	e_i
1	1	4
2	3	5
3	0	6
4	5	7
5	3	8
6	5	9
7	6	10
8	8	11
9	8	12
10	2	13
11	12	14

Optimale oplossing: . . .

i	b_i	e_i
1	1	4
2	3	5
3	0	6
4	5	7
5	3	8
6	5	9
7	6	10
8	8	11
9	8	12
10	2	13
11	12	14



Optimale oplossing: . . .

Gretig algoritme: in elke stap

- wordt een activiteit i gekozen die compatibel is met de reeds gekozen activiteiten en die op dat moment het beste lijkt (gretige keuze)

Mogelijke gretige keuzes voor het kiezen van activiteit i: . . .

Gretig algoritme: in elke stap

- wordt een activiteit i gekozen die compatibel is met de reeds gekozen activiteiten en die op dat moment het beste lijkt (gretige keuze)

Mogelijke gretige keuzes voor het kiezen van activiteit i:

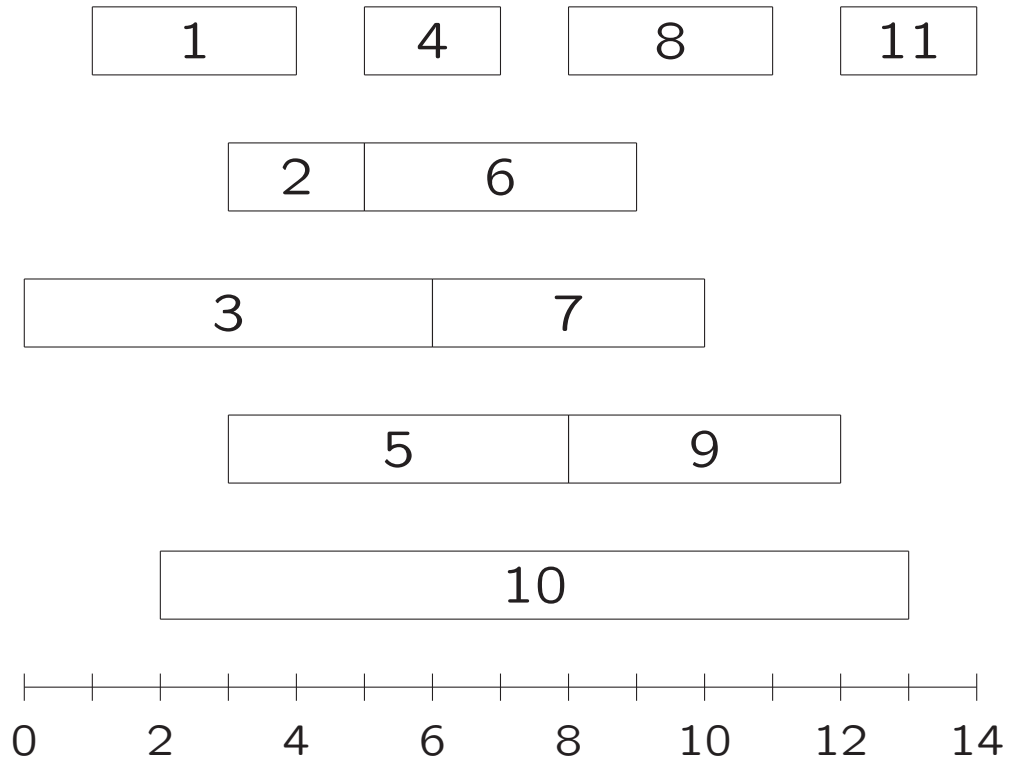
1. selecteer steeds de activiteit die het eerst begint
2. selecteer steeds de activiteit die het kortste duurt
3. selecteer steeds de activiteit die overlapt met zo min mogelijk andere activiteiten
4. selecteer steeds de activiteit die het eerst eindigt

Welke gretige keuzes voor het kiezen van activiteit i leiden altijd tot een optimale oplossing?

1. selecteer de activiteit die het eerst begint: werkt niet
2. selecteer de activiteit die het kortste duurt: werkt niet
3. selecteer de activiteit die overlapt met zo min mogelijk andere activiteiten: werkt niet
4. selecteer de activiteit die het eerst eindigt: werkt wel

```
// neem aan dat A oplopend gesorteerd is op eindtijd e_i,
// anders eerst even sorteren: O(n lg n)
A' := {1};
j := 1;
// de laatst aan A' toegevoegde activiteit
for i := 2 to n do
// loop activiteiten af in volgorde van eindtijd
    if i is compatibel met A' then (*)
        A' := A' ⋃ {i}; j := i;
    fi
od
// A' bevat nu een optimale paarsgewijs
// compatibele deelverzameling van A
```

i	b_i	e_i
1	1	4
2	3	5
3	0	6
4	5	7
5	3	8
6	5	9
7	6	10
8	8	11
9	8	12
10	2	13
11	12	14



Optimale oplossing: $\{1, 4, 8, 11\}$

// neem aan dat A oplopend gesorteerd is op eindtijd e_i,
// anders eerst even sorteren: $O(n \lg n)$
$A' := \{1\}$;
$j := 1$;
// de laatst aan A' toegevoegde activiteit
for $i := 2$ to n do
// loop activiteiten af in volgorde van eindtijd
    if i is compatibel met A' then (*)
        $A' := A' \bigcup \{i\}$; $j := i$;
    fi
od
// A' bevat nu een optimale paarsgewijs
// compatibele deelverzameling van A

Correctheid: . . .

De **correctheid** van het gretige algoritme volgt uit de volgende twee observaties:

1. Er bestaat een optimale oplossing die met activiteit 1 begint (die met de kleinste eindtijd dus).

2. . . .

De **correctheid** van het gretige algoritme volgt uit de volgende twee observaties:

1. Er bestaat een optimale oplossing die met activiteit 1 begint (die met de kleinste eindtijd dus).

2. Stel A' is een optimale oplossing van het oorspronkelijke probleem, dus met activiteitenverzameling A, die 1 bevat. Dan is $B' = A' \setminus \{1\}$ een optimale oplossing van het probleem met activiteitenverzameling $B = \{i \in A : b_i \geq e_1\}$.

```
// neem aan dat A oplopend gesorteerd is op eindtijd e_i,
// anders eerst even sorteren: O(n lg n)
A' := {1};
j := 1;
// de laatst aan A' toegevoegde activiteit
for i := 2 to n do
// loop activiteiten af in volgorde van eindtijd
    if i is compatibel met A' then (*)
        A' := A' ⋃ {i}; j := i;
    fi
od
// A' bevat nu een optimale paarsgewijs
// compatibele deelverzameling van A
```

(*) Merk op: i is compatibel met A' als hij compatibel is met de laatst toegevoegde activiteit.

Dus (*) wordt: `if` $b_i \geq e_j$ `then`

Correctheid: OK (gezien)

Complexiteit: $O(n)$ als A reeds gesorteerd is

- **Lezen/leren bij dit college:**
  slides, inleiding hoofdstuk 9, 9.2 t/m blz. 353, 9.3

- **Werkcollege** dynamisch programmeren
  vrijdag 21 april 2023, 13.15–15.00, 313

- **Opgaven:**
  zie `https://liacs.leidenuniv.nl/~vlietrvan1/algoritmiek/`

- **Practicumbijeenkomst** programmeeropdracht 1/2:
  Woensdag 19 april 2023, 13.15–15.00, computerzalen Snellius

- **Volgend hoorcollege:**
  dinsdag 25 april 2023, 13.15–15.00

- Het bewijs van de correctheid van het vierde gretige algoritme voor het activiteitenselectieprobleem (slides 52-54) hebben we dit jaar overgeslagen. Het is dan ook geen tentamenstof.