

Tiende college algoritmiek

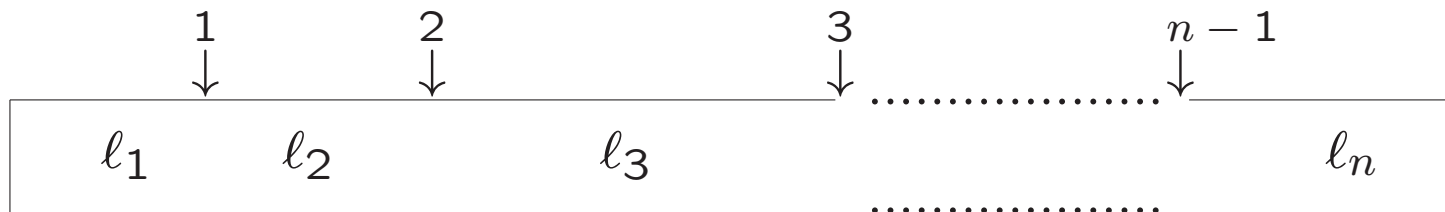
14 april 2021

Dynamisch programmeren

Gretige Algoritmen

Algoritme van Dijkstra

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 $\ell_1, \ell_2, \dots, \ell_n$ 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 $l_1 = 6, l_2 = 8, l_3 = 7, l_4 = 2$. 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 $L(i, j) = l_i + l_{i+1} + \dots + l_j$ te verzagen tot achtereenvolgens stukken van lengte l_i, l_{i+1}, \dots, l_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

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

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 $L(i, j) = l_i + l_{i+1} + \dots + l_j$ te verzagen tot achtereenvolgens stukken van lengte l_i, l_{i+1}, \dots, l_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	→												
i	0
↓	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	→												
i	0
↓		0
			*	*	*	*	*	*	*	*	#	.	.	
				0	*	.	.	
					0	*	.	.	
						0	*	.	.	
							0	.	.	.	*	.	.	

Invulvolgorde

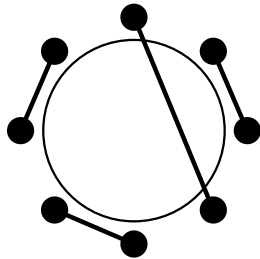
De tabel kan bottom up gevuld worden door alle diagonalen $j = i + d$ af te lopen en per diagonaal bijvoorbeeld van linksboven tot rechtsonder te gaan. Een andere mogelijkheid is de tabel rij voor rij te vullen (van onder naar boven) en per rij (verplicht) van links naar rechts.

```
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
```

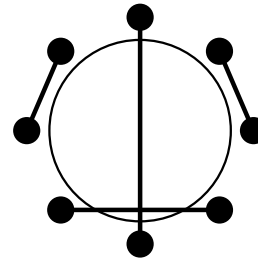

The Bavarian Beer Party (BAPC 2006)

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, \dots , 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 $\frac{n}{2}$ sluihaakjes.

Op positie 1 staat een openingshaakje; het corresponderende sluihaakje staat op positie 2, positie 4, positie 6, \dots , 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, \dots, j op een toegestane manier aan elkaar koppelen. We zoeken dus $M[1][n]$.

Dan geldt:

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

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, \dots, j op een toegestane manier aan elkaar koppelen. We zoeken dus $M[1][n]$.

Dan geldt:

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

Vraag: waarom proberen we in de bovenste regel alleen

$k = i + 1, i + 3, i + 5, \dots, j$?

Iets andere formulering recurrente betrekking zagen:

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

Handen schudden:

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

Gegeven onbeperkt veel munten van d_1, d_2, \dots, 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, \dots, 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, \dots, 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 lokaal 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 **heuristisch**, 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
 - ...

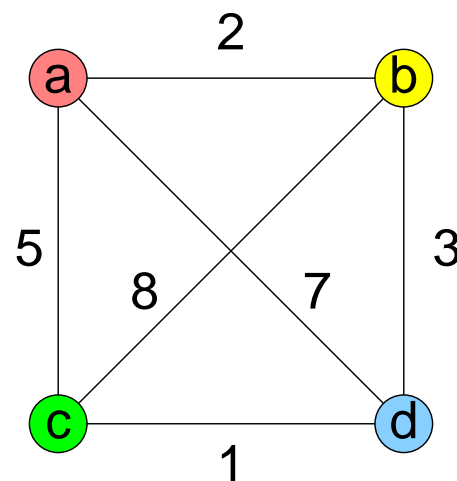
Traveling Salesman Problem (handelsreizigersprobleem)

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, \dots, w_n en waarde v_1, \dots, 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, \dots, w_n en waarde v_1, \dots, 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

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

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, \dots, i_n worden uitgevoerd. Dan is

$$\begin{aligned} T &= t_{i_1} + (t_{i_1} + t_{i_2}) + (t_{i_1} + t_{i_2} + t_{i_3}) + \dots + (t_{i_1} + t_{i_2} + \dots + t_{i_n}) \\ &= n * t_{i_1} + (n - 1) * t_{i_2} + \dots + 2 * t_{i_{n-1}} + t_{i_n} \end{aligned}$$

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 $A = \{1, 2, \dots, n\}$ 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 $e_j \leq b_i$.

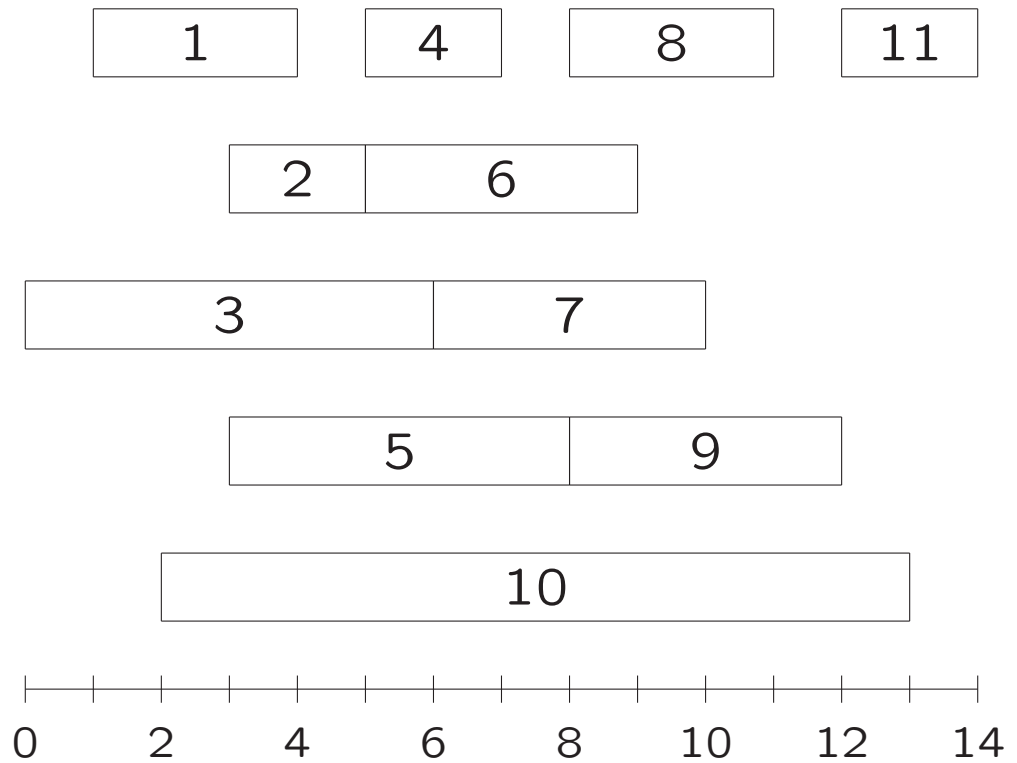
Opdracht: vind een zo groot mogelijke deelverzameling van paarsgewijs compatibele activiteiten uit A .

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 :

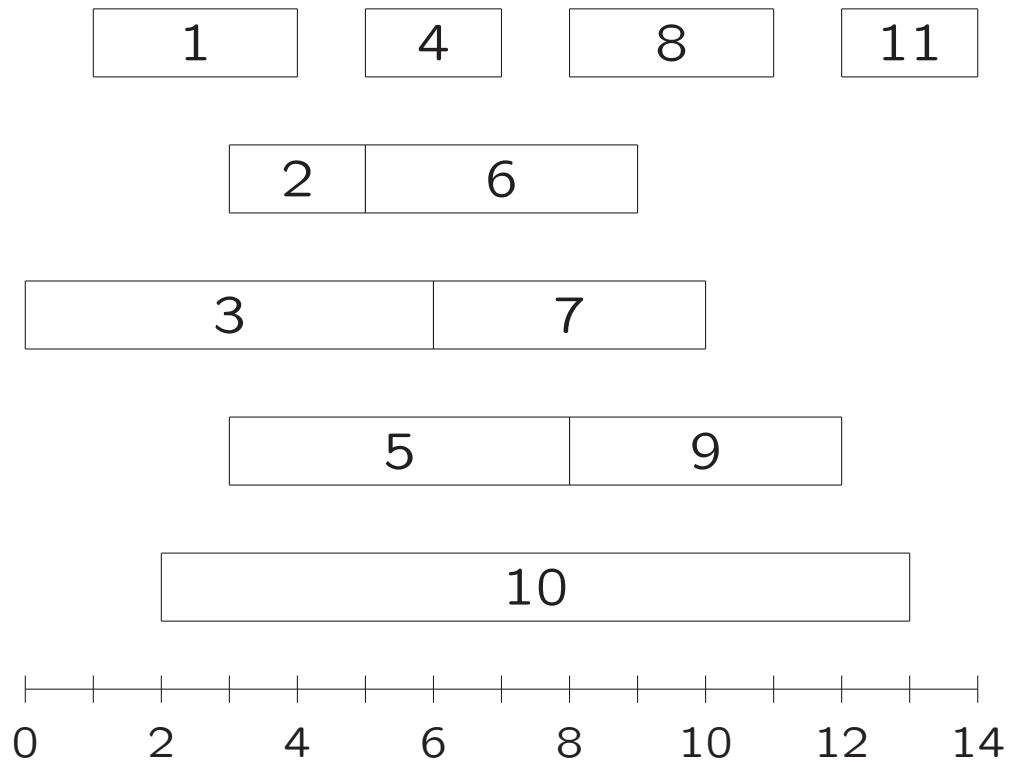
1. selecteer steeds de activiteit die het eerst begint
2. selecteer steeds de activiteit die het kortste duurt
3. selecteer steeds de activiteit die overlapt met zo min mogelijk andere activiteiten
4. selecteer steeds de activiteit die het eerst eindigt

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 (*)
         $A' := A' \cup \{i\}$ ;  $j := i$ ;
    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 (*)  
     $A' := A' \cup \{i\}; j := i;$   
  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 (*)  
     $A' := A' \cup \{i\}; j := i;$   
  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

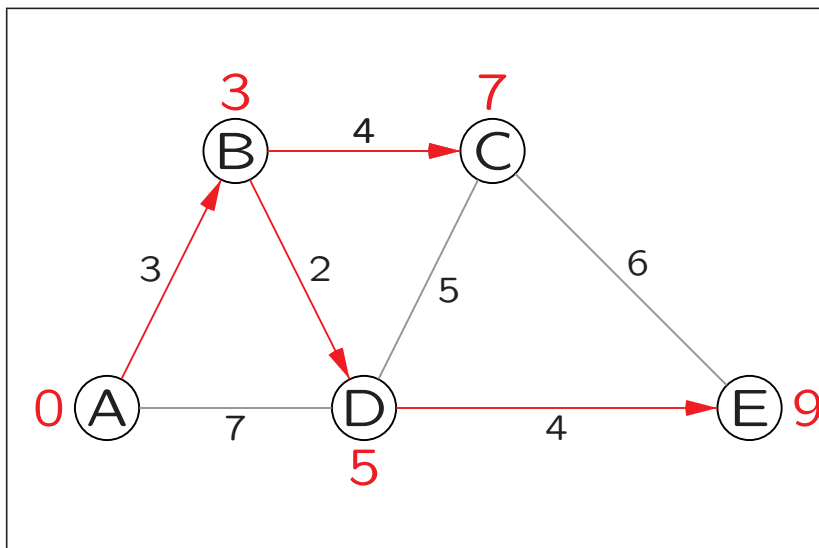
Correctheid: OK (gezien)

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

Gegeven een graaf G met gewichten op de takken, en een beginknoop s . We nemen aan dat alle gewichten ≥ 0 zijn.

Gevraagd: voor elke willekeurige knoop v in de graaf (de lengte van) het/een kortste pad van s naar v .

Merk op dat al deze kortste paden vanuit s samen een boomstructuur vormen.



De kortste paden vanuit A zijn:

A \rightarrow B: lengte 3

A \rightarrow B \rightarrow D: lengte 5

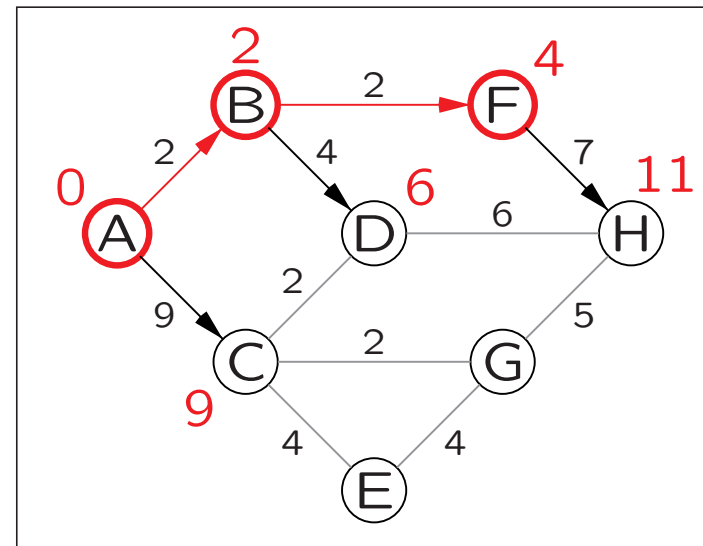
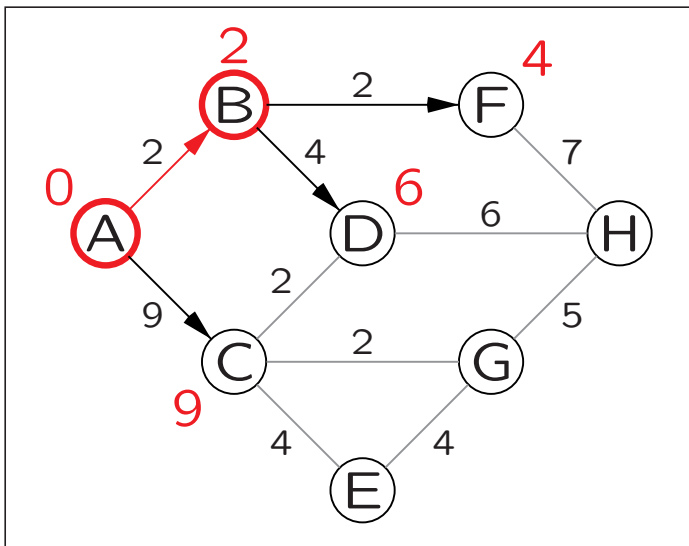
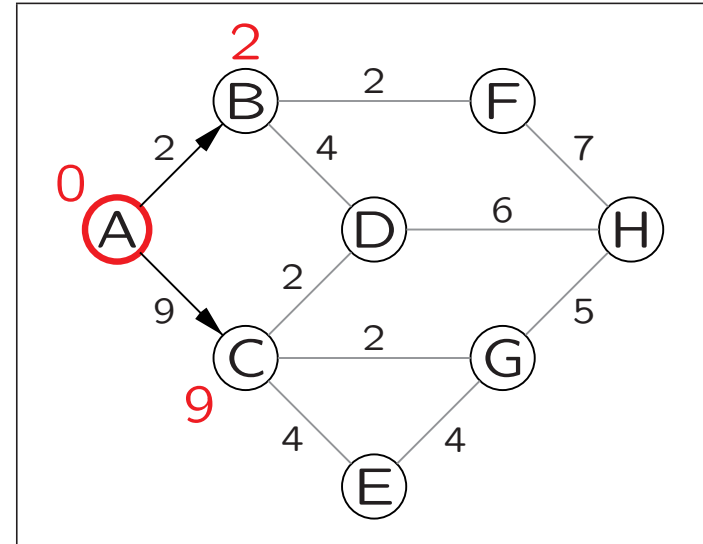
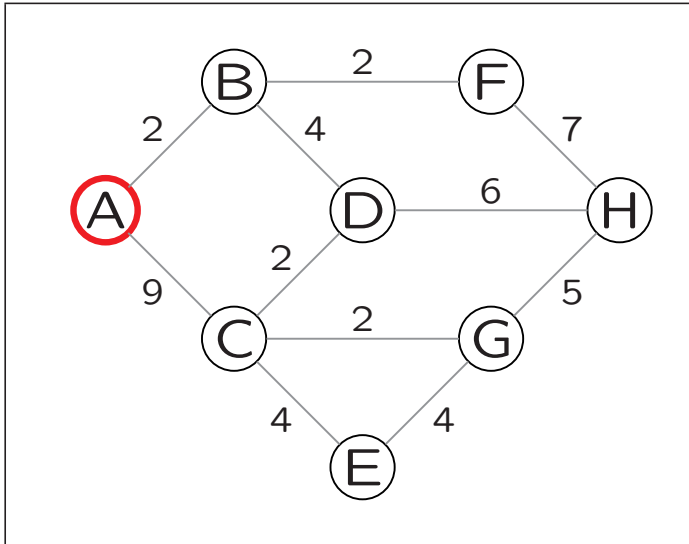
A \rightarrow B \rightarrow C: lengte 7

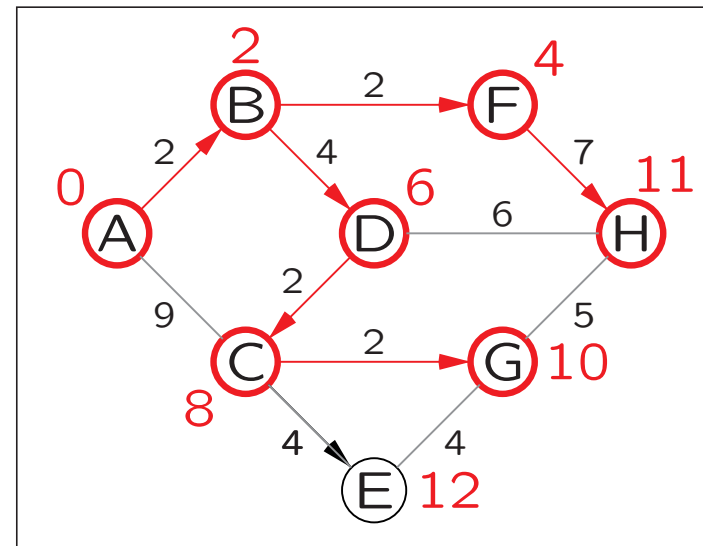
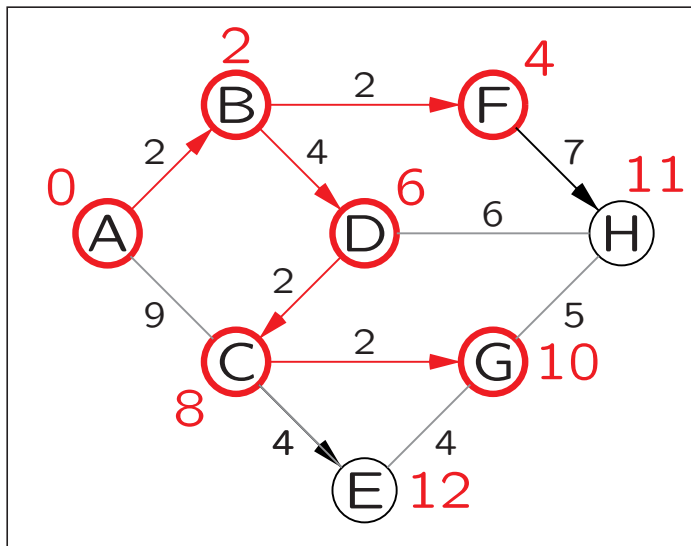
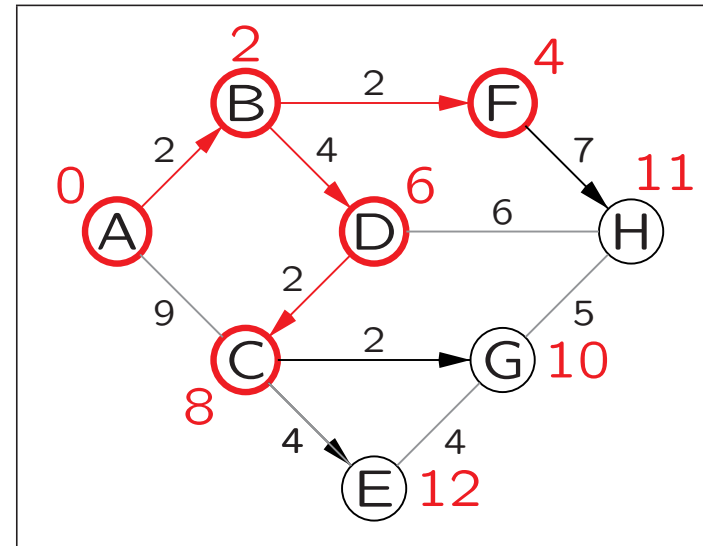
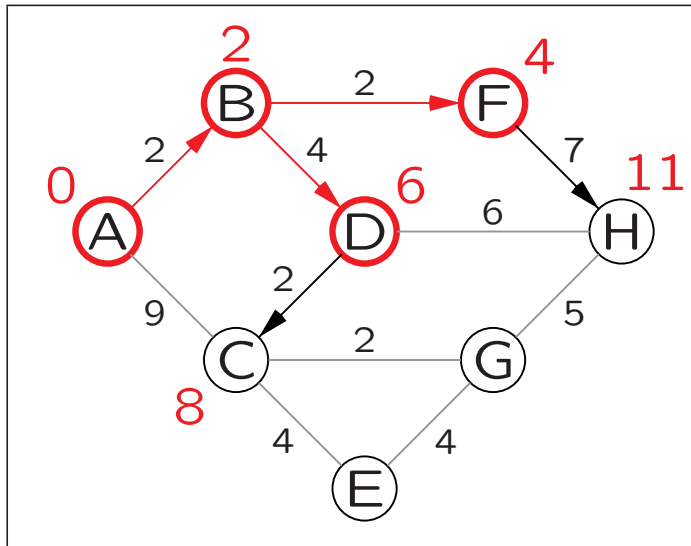
A \rightarrow B \rightarrow D \rightarrow E: lengte 9

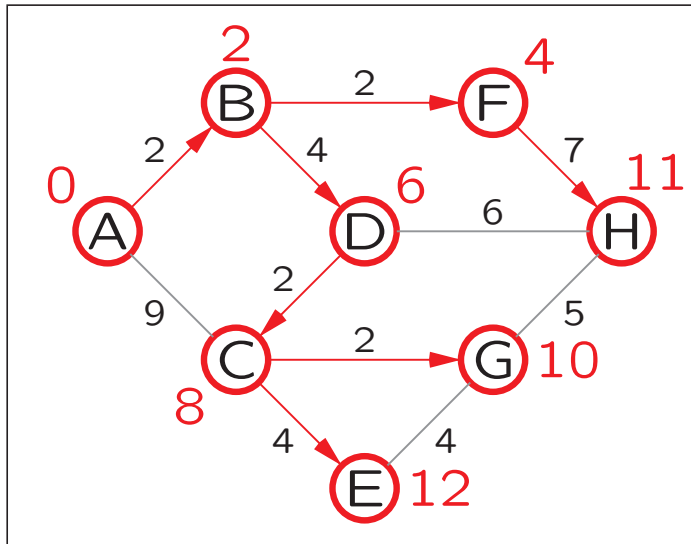
Oplossing: het **algoritme van Dijkstra** is een gretig algoritme, dat de kortste paden van s naar elk van de andere knopen vindt in volgorde van hun lengte. In elke stap wordt een knoop (en daarmee het pad van s naar die knoop) gekozen waarvoor **het tot nu toe bekende pad vanaf s zo kort mogelijk is.**



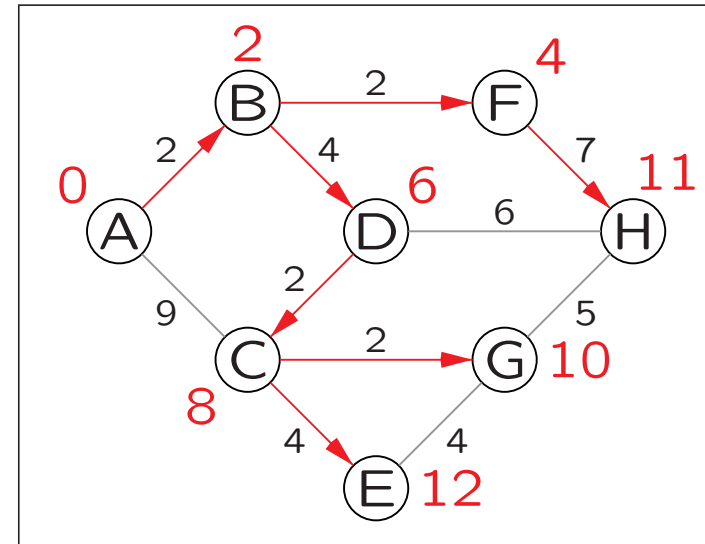
Edsger Wybe Dijkstra (Rotterdam, 11 mei 1930 — Nuenen, 6 augustus 2002) was een Nederlandse wiskundige en informaticus die veel voor de informatica heeft gedaan, met name op het gebied van gestructureerd programmeren. In 1972 werd hij onderscheiden met de Turing Award.







Het algoritme is klaar:
alle knopen gehad



Alle kortste paden vanuit
A met hun lengtes

Als je gewoon wilt doortekenen in één plaatje...

Begin met A, afstand 0.

$A \rightarrow B: 0 + 2 = 2$. OK.

$A \rightarrow C: 0 + 9 = 9$. OK.

Kies B, afstand 2, vanaf A.

$B \rightarrow D: 2 + 4 = 6$. OK.

$B \rightarrow F: 2 + 2 = 4$. OK.

Kies F, afstand 4, vanaf B.

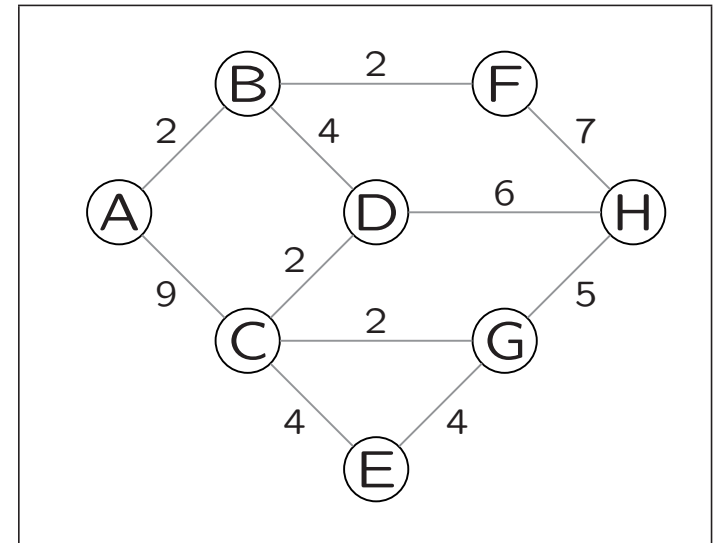
$F \rightarrow H: 4 + 7 = 11$. OK.

Kies D, afstand 6, vanaf B.

$D \rightarrow C: 6 + 2 = 8 < 9$. OK.

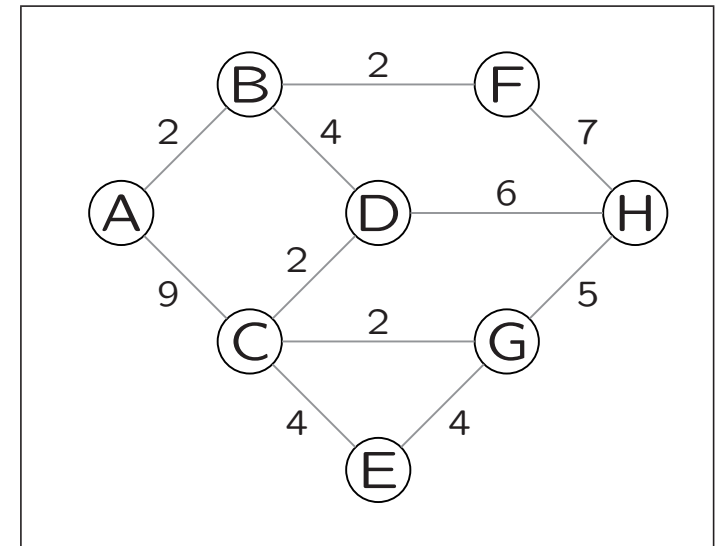
$D \rightarrow H: 6 + 6 = 12 > 11$. X.

Enzovoort.



Als je gewoon wilt doortekenen in één plaatje... (alternatief)

A	B	C	D	E	F	G	H	Actie
0	∞	∞	∞	∞	∞	∞	∞	Begin met A
-	2	9	∞	∞	∞	∞	∞	Kies B, vanaf A
-	-	9	6	∞	4	∞	∞	Kies F, vanaf B
-	-	9	6	∞	-	∞	11	Kies D, vanaf B
-	-	8	-	∞	-	∞	11	Kies C, vanaf D
-	-	-	-	12	-	10	11	Kies G, vanaf C
-	-	-	-	12	-	-	11	Kies H, vanaf F
-	-	-	-	12	-	-	-	Kies E, vanaf C



Een rij in de tabel komt overeen met het array pad in pseudo-code op volgende slide.


```
// invoer: samenhangende gewogen graaf  $G = (V, E)$  en startknoop  $s$   
// uitvoer: array dat de lengtes van de kortste paden vanuit  $s$  bevat;  
// na afloop is  $\text{pad}[v] =$  de lengte van een kortste pad van  $s$  naar  $v$ 
```

```
for  $v \in V$  do  
     $\text{pad}[v] := \infty$ ;  
od  
 $\text{pad}[s] := 0$ ;  
 $U := \emptyset$ ;  
  
while (  $U \neq V$  ) do  
    vind knoop  $v^* \in V \setminus U$  met  $\text{pad}[v^*]$  minimaal;  
     $U := U \cup \{v^*\}$ ;  
    for alle knopen  $v$  aangrenzend aan  $v^*$  do  
        if  $\text{pad}[v^*] + \text{gewicht}(v^*, v) < \text{pad}[v]$  then  
             $\text{pad}[v] := \text{pad}[v^*] + \text{gewicht}(v^*, v)$ ;  
        fi  
    od  
od
```

- **Lezen/leren bij dit college:**
slides, inleiding hoofdstuk 9, 9.2 t/m blz. 353, 9.3
- **Werkcollege** gretige algoritmen / Dijkstra:
donderdag 15 april 2021, 14.15–15.00 (weblecture)
- **Opgaven:**
zie <https://liacs.leidenuniv.nl/~vlietrvan1/algoritmiek/>
- **Interactief vragenuur:**
donderdag 15 april 2021, 15.15–16.00, Kaltura liveroom
- **Practicumbijeenkomst** programmeeropdracht 2:
dinsdag 20 april 2021, 10.30-12.15, Kaltura liveroom
- **Volgend hoorcollege:**
woensdag 21 april 2021, 14.15–16.00 (weblecture)