

Tiende college algoritmiek

24 april 2015

Gretige algoritmen

Dijkstra

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
- 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 achter-eenvolgende **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 **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
 -

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	$3 + (3 + 10) + (3 + 10 + 5) = 34$

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 (in elk geval 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 (in elk geval tot de volgende job aan de beurt is) zo klein mogelijk.

Observatie.

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

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

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 .

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 is  $j$   
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: moet bewezen worden

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

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

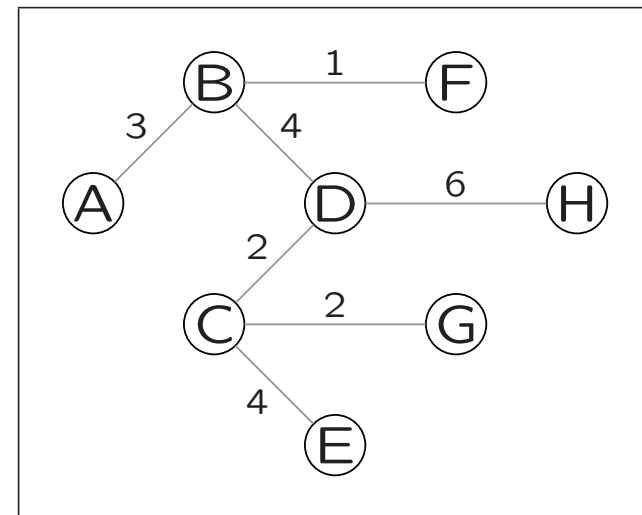
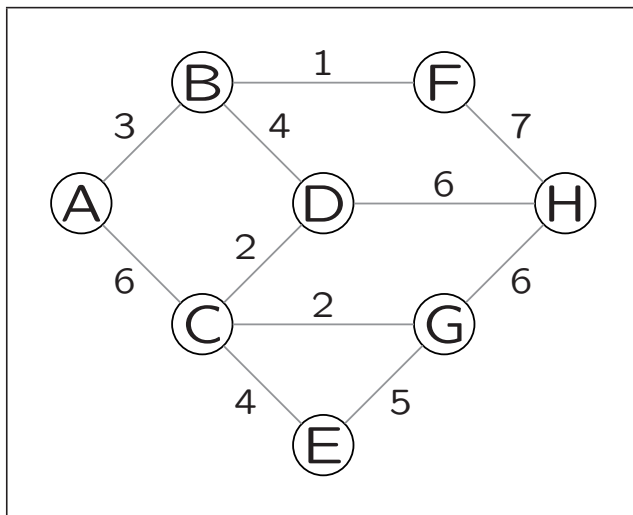
De **correctheid** van het gretige algoritme volgt uit de volgende twee observaties:

1. Er bestaat een maximale paarsgewijs compatibele deelverzameling van A die met activiteit 1 begint (die met de kleinste eindtijd dus).
2. Stel A' is een maximale paarsgewijze compatibele deelverzameling van activiteitenverzameling A , die 1 bevat. Dan is $B' = A' \setminus \{1\}$ een maximale paarsgewijs compatibele deelverzameling van activiteitenverzameling $B = \{i \in A : b_i \geq e_1\}$.

Gegeven een **samenhangende, ongerichte** graaf G met gewichten op de takken.

Definitie: een opspannende boom voor G is een boom die alle knopen van G bevat.

Gevraagd: een **opspannende boom** van G met minimaal totaal gewicht.



```
// invoer: samenhangende, ongerichte gewogen graaf  $G = (V, E)$   
// uitvoer:  $E_T$ , verzameling takken van minimale opspannende boom  
//  $T$  van  $G$ 
```

```
sorteer  $E$  op gewicht (oplopend):  $e_{i_1}, e_{i_2}, \dots, e_{i_m}$ ;
```

```
 $E_T := \emptyset$ ;
```

```
takteller := 0;  $k := 0$ ;
```

```
while takteller <  $|V| - 1$  do
```

```
     $k := k + 1$ ;
```

```
    if  $E_T \cup \{e_{i_k}\}$  is acyclisch then
```

```
         $E_T := E_T \cup \{e_{i_k}\}$ ;
```

```
        takteller := takteller+1;
```

```
    fi
```

```
do
```

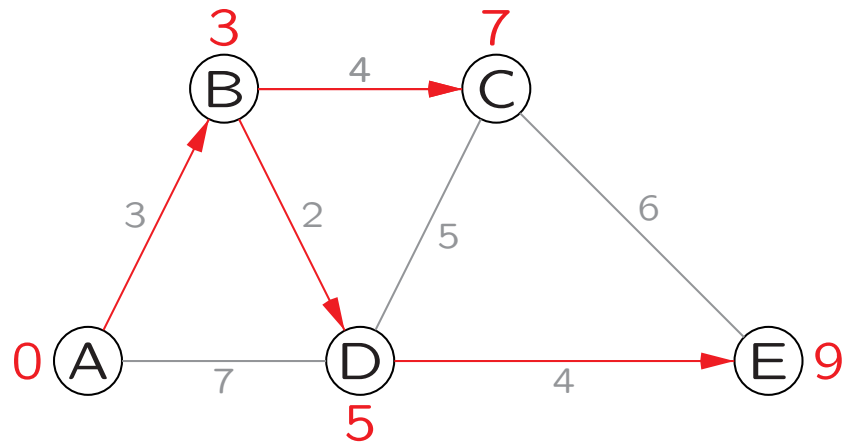
Complexiteit...

Correctheid...

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

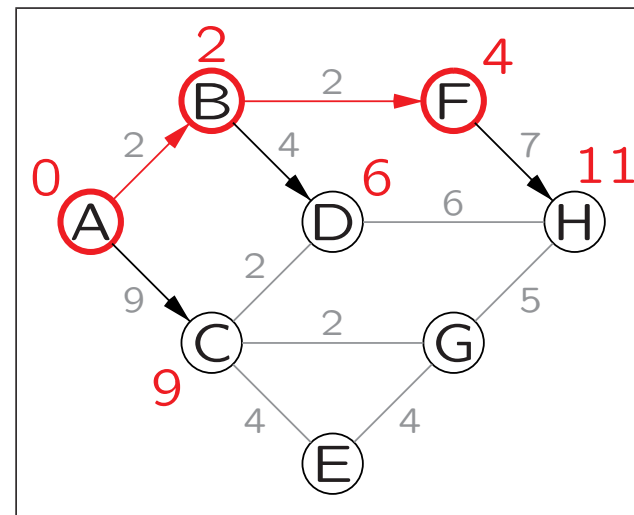
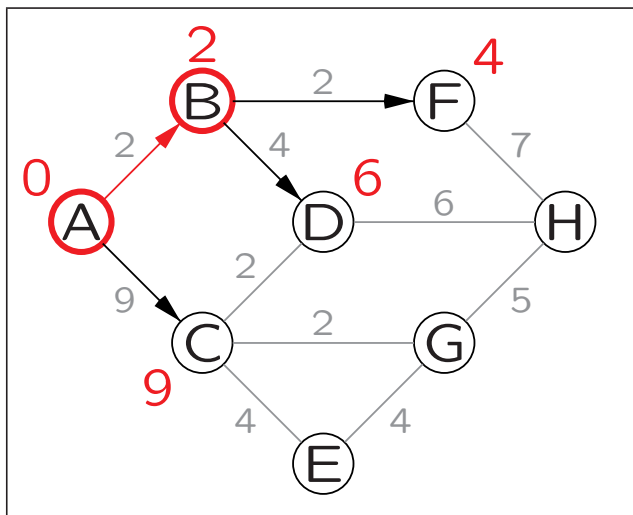
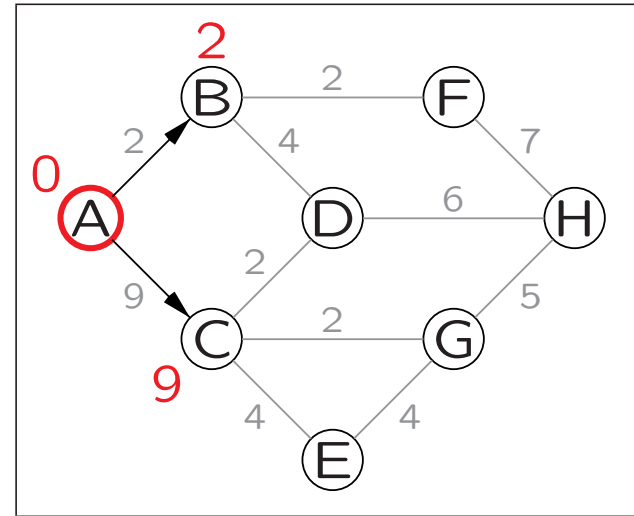
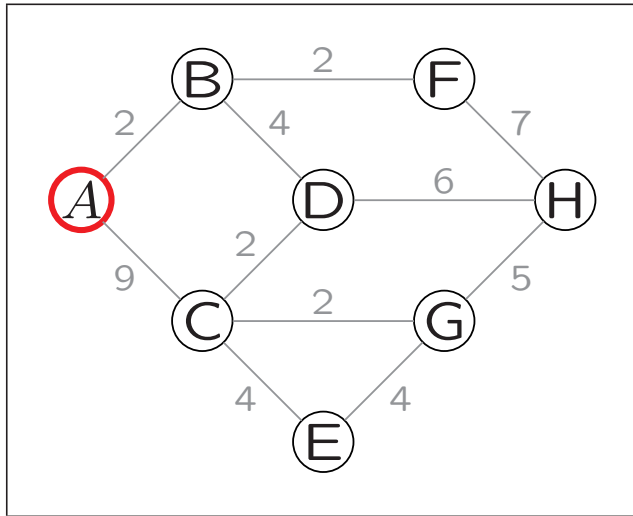
Oplossing: het **algoritme van Dijkstra** is een *gretig* algoritme, dat de kortste paden van s naar elk van de andere knopen vindt in volgorde van hun lengte. In elke stap wordt een knoop (en daarmee het pad van s naar die knoop) gekozen waarvoor **het tot nu toe bekende pad vanaf s zo kort mogelijk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

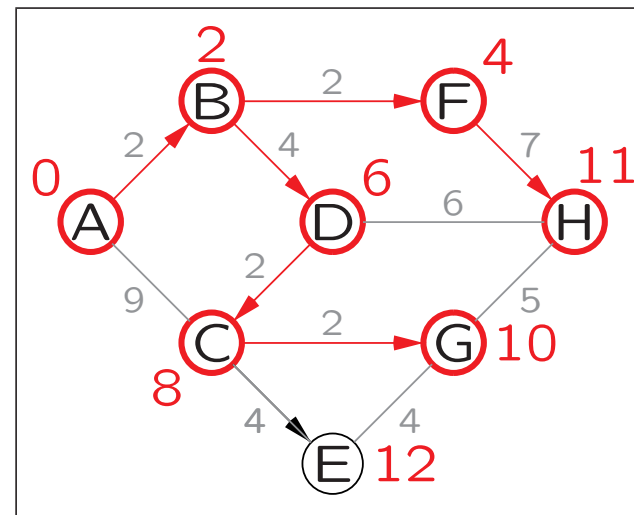
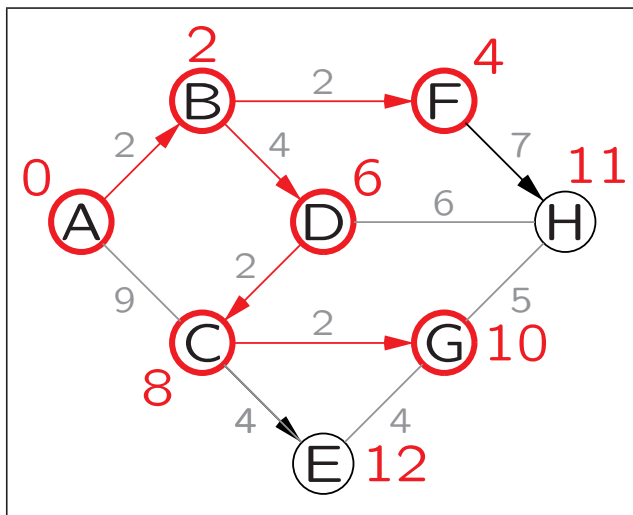
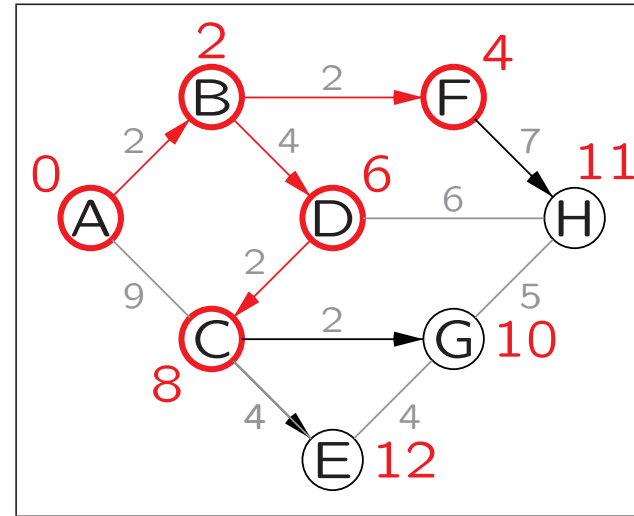
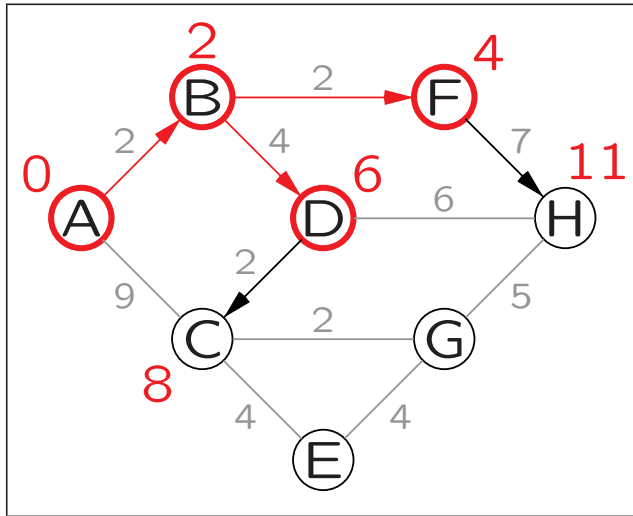
Dijkstra: Edsger W. Dijkstra (1930-2002)

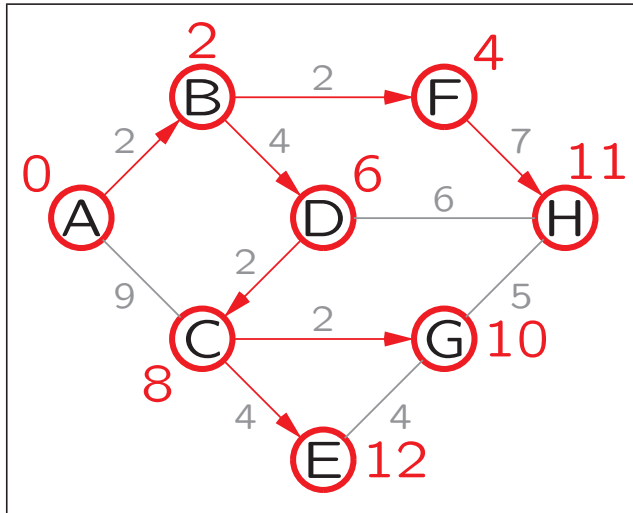


Edsger Wybe Dijkstra (Rotterdam, 11 mei 1930 — Nuenen, 6 augustus 2002) was een Nederlandse wiskundige en informaticus die veel voor de informatica heeft gedaan, met name op het gebied van gestructureerd programmeren. In 1972 werd hij onderscheiden met de Turing Award.

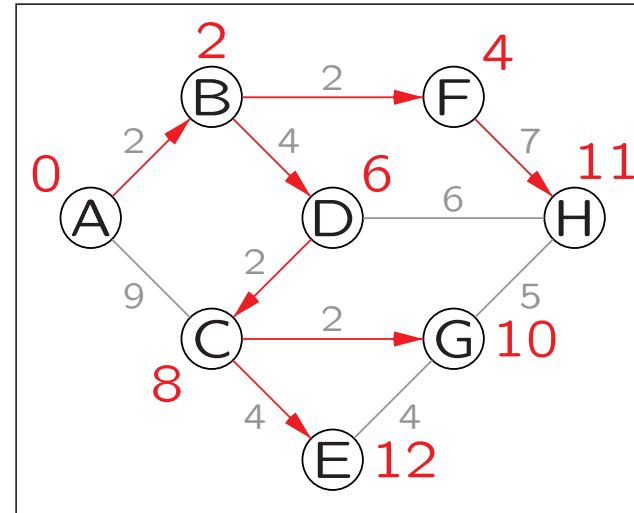
```
// invoer: samenhangende gewogen graaf  $G = (V, E)$  en startknoop  $s$ 
// uitvoer: array dat de lengtes van de kortste paden vanuit  $s$  bevat;
// na afloop is  $\text{pad}[v] =$  de lengte van een kortste pad van  $s$  naar  $v$ 
for  $v \in V$  do
     $\text{pad}[v] := \infty$ ; od
 $\text{pad}[s] := 0$ ;
 $U := \emptyset$ ;
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
```







Het algoritme is klaar:
alle knopen gehad



Alle kortste paden vanuit
A met hun lengtes

Als je gewoon wilt doortekenen in één plaatje, beschrijf daarbij dan wat er gebeurt in elke stap.

Begin met A, afstand 0.

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

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

Kies B, afstand 2, vanaf A.

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

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

Kies F, afstand 4, via B.

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

Kies D, afstand 6, via B.

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

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

Enzovoort.

Als je gewoon wilt doortekenen in één plaatje, kun je ook in een tabel aangeven wat er gebeurt. De achtereenvolgende stappen van het algoritme van Dijkstra voor de voorbeeldgraaf zijn dan:

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

Een rij in de tabel komt overeen met het array `pad` in de pseudo-code op de volgende slide; reeds gekozen knopen aangegeven met –.

```
// 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
```

- In het algoritme bevat U steeds alle knopen waarvan de definitieve kortste afstand vanaf s reeds bepaald is. Voor deze knopen geeft het label $\text{pad}[v]$ al de definitieve kortste afstand aan. **Moet bewezen worden.**
- Voor de andere knopen w geldt na elke ronde (= doorgang door de while):

$$(\#) \text{pad}[w] = \min_{u \in U} \{ \text{pad}[u] + \text{gewicht}(u, w) \}^*$$

Dit volgt direct uit het algoritme.

*Dit betekent dat $\text{pad}[w]$ voor deze knopen $w \notin U$ de lengte van een kortste pad van s naar w aangeeft via uitsluitend knopen van U .

- De volgende *dichtstbijzijnde* knoop v^* wordt gekozen uit de knopen uit $V \setminus U$ die direct grenzen aan U . Nadat deze gekozen is worden de labels aangepast, zodat (#) ook geldt voor de nieuwe U .
- Het is niet zo moeilijk dit algoritme aan te passen zodat ook de kortste paden zelf worden berekend. Sla direct na het aanpassen van het label van knoop v de nieuwe kandidaattak (v^*, v) op:

```
if  $\text{pad}[v^*] + \text{gewicht}(v^*, v) < \text{pad}[v]$  then  
     $\text{pad}[v] := \text{pad}[v^*] + \text{gewicht}(v^*, v);$   
    nieuwe kandidaattak voor  $v$ :  $(v^*, v)$ ;  
fi
```

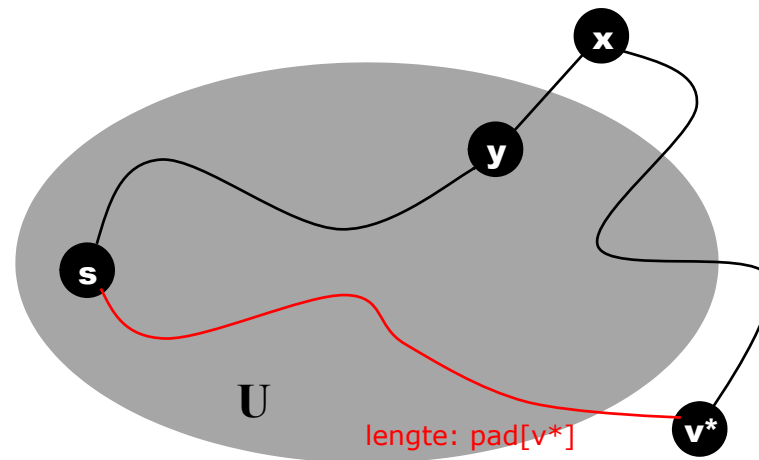
Na elke ronde (dus ook na de laatste, wanneer $U = V$) van het algoritme van Dijkstra geldt:

- U bevat alle knopen waarvan de definitieve kortste afstand vanaf s reeds bepaald is. Voor elke $v \in U$ geeft het label $\text{pad}[v]$ die kortste afstand aan.

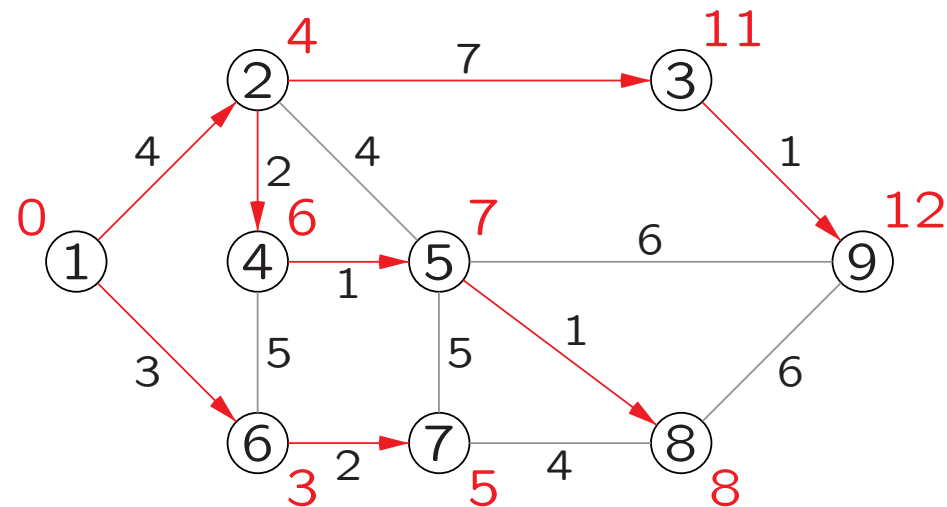
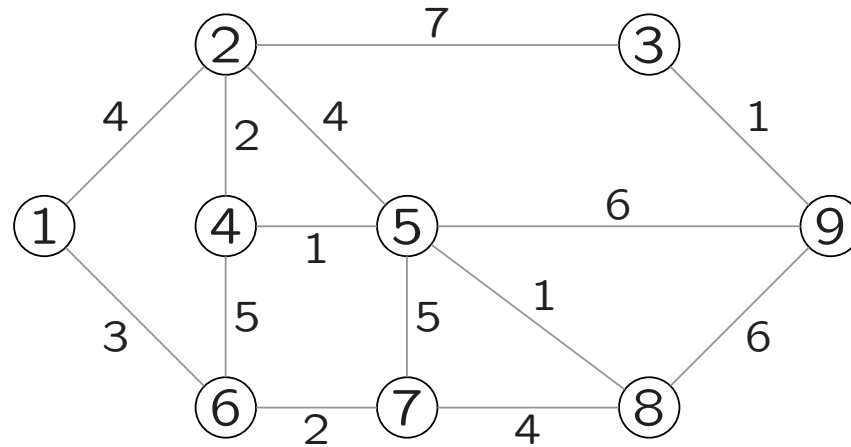
Om dit te bewijzen moet je laten zien dat:

- wanneer v^* wordt toegevoegd aan U , $\text{pad}[v^*]$ inderdaad de lengte van het kortste pad van s naar v^* is

Bekijk, behalve het **pad ter lengte $\text{pad}[v^*]$** van s naar v^* via U , een willekeurig ander pad van s naar v^* . Stel dat x de eerste knoop op dat pad is buiten U , en y de laatste knoop daarvóór. (Deze x kán gelijk zijn aan v^* .) Zij $d(s, y, x, v^*)$ de lengte van dat pad.



Dan geldt: $d(s, y, x, v^*) \geq d(s, y, x) \geq \text{pad}[x] \geq \text{pad}[v^*]$ omdat respectievelijk alle gewichten van de takken ≥ 0 zijn, (\neq) geldt voor x , en v^* gekozen was in deze ronde als “minimale” knoop. QED



De achtereenvolgende stappen van het algoritme van Dijkstra voor de voorbeeldgraaf:

1	2	3	4	5	6	7	8	9	Actie
0	∞	∞	∞	∞	∞	∞	∞	∞	Begin met 1
–	4	∞	∞	∞	3	∞	∞	∞	Kies 6, vanaf 1
–	4	∞	8	∞	–	5	∞	∞	Kies 2, vanaf 1
–	–	11	6	8	–	5	∞	∞	Kies 7, via 6
–	–	11	6	8	–	–	9	∞	Kies 4, via 2
–	–	11	–	7	–	–	9	∞	Kies 5, via 4
–	–	11	–	–	–	–	8	13	Kies 8, via 5
–	–	11	–	–	–	–	–	13	Kies 3, via 2
–	–	–	–	–	–	–	–	12	Kies 9, via 3

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

- **Lezen/leren bij dit college:**
hoofdstuk 9 inl, 9.2 t/m blz. 353, 9.3, sheets
- **Deadline** programmeeropdracht 2:
dinsdag 28 april 2015, 10:00 uur
- **Extra vragenuur (zaal 302/304):**
vrijdag 24 april, 15:30 – 17:00 uur
- **Opgaven en programmeeropdrachten:**
zie <http://liacs.leidenuniv.nl/~graafjmde/ALGO/>
- **Volgend werkcollege:**
donderdag 30 april 2015 in zaal B2
- **Volgend college:**
vrijdag 1 mei 2015, 11:15–13:00, zaal B2.