Tiende college algoritmiek

19 april 2013

Restant DP

Gretige algoritmen

Werkwijze DP

1. Druk de (waarde van de) oplossing van het probleem uit in (de waarde van) oplossingen van deelproblemen.

2. Stel een recurrente betrekking op (recursieve formulering).

3. Je kunt nu nog kiezen tussen DP met recursie of bottom up DP. We gaan hier bottom up te werk.

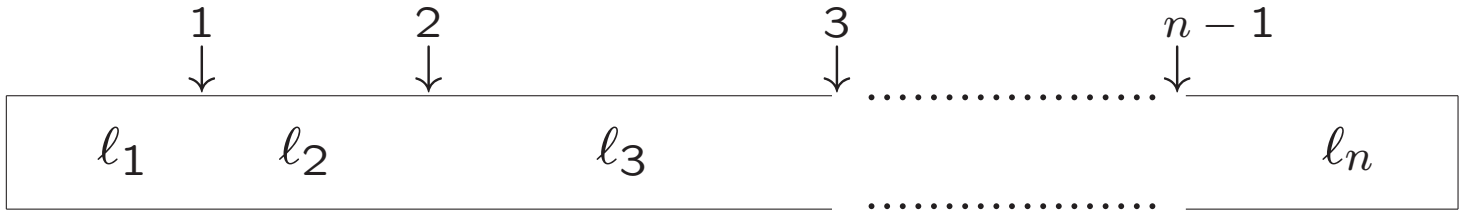
4. Definieer een geschikte tabel en ga na wat de berekeningsvolgorde moet zijn.

5. Vul aldus bottom up de tabel in (algoritme).

6. Let op geheugenbesparing (indien gewenst).

7. Pas je algoritme zo aan dat je uit de tabel niet alleen een waarde maar ook de (optimale) oplossing zelf kunt halen. Dit kan achteraf uit de tabel of via een geschikt hulparray dat tijdens het vullen van de tabel wordt opgebouwd.

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 zaagplek $i-1$ tot zaagplek 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 $1 \leq i \leq j \leq n$. Verder $Z[i][i] = 0$; er hoeft niet gezaagd te worden.

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	.	.	.	*	.	.
							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 in het plaatje) 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
```

- 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 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.

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.)

Toepassingen gretige aanpak

- 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

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

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

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. De volgende job is dan zo snel mogelijk aan de beurt.

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

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

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

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 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.

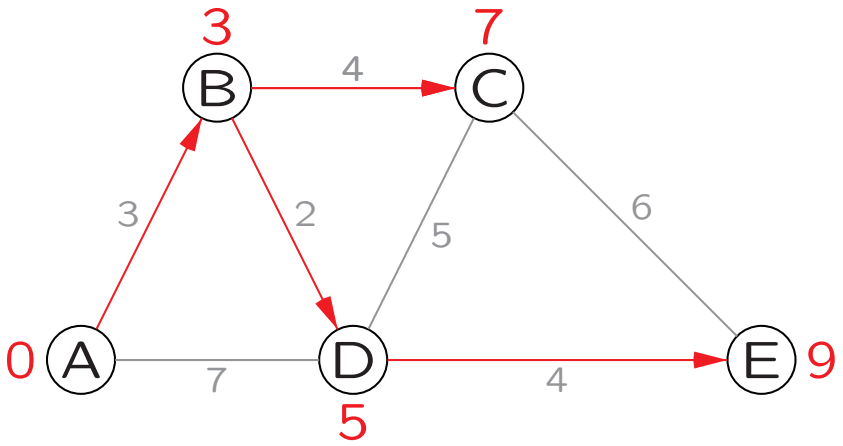
Merk op dat al deze kortste paden vanuit s samen een boomstructuur vormen.

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.

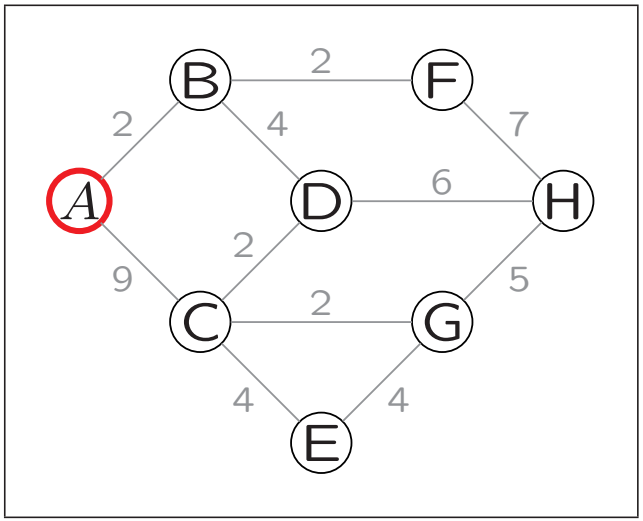
Voorbeeld kortste-paden-boom

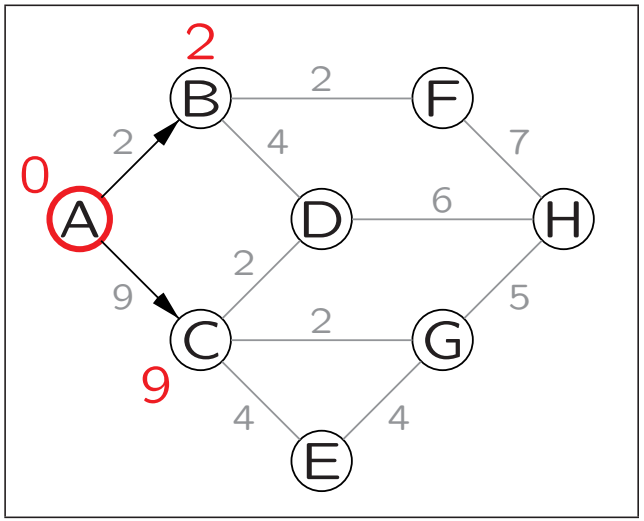


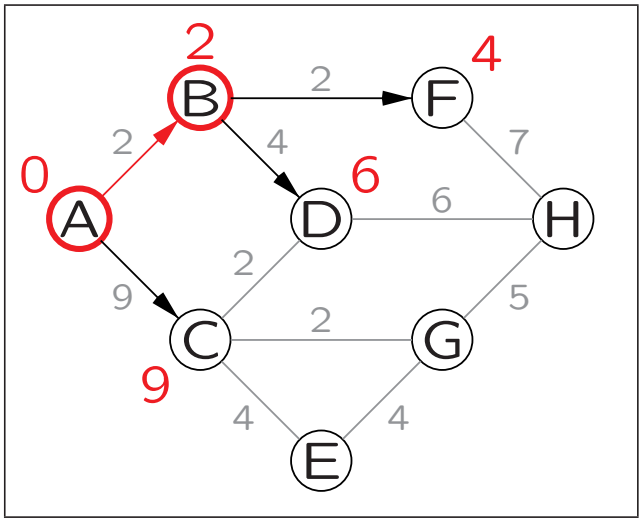
De kortste paden vanuit A zijn:

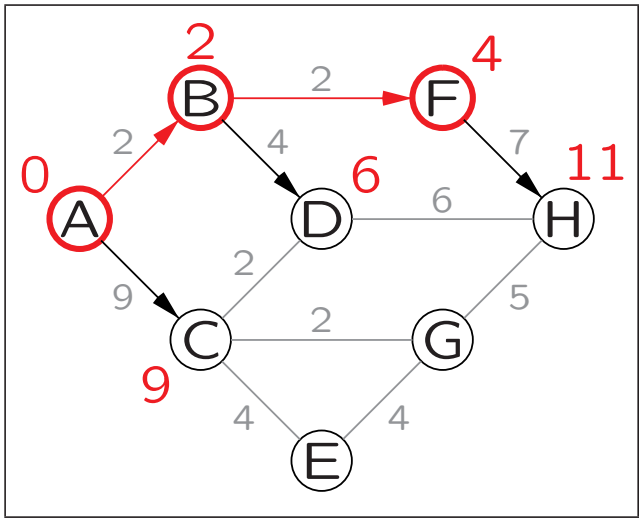
A $\longrightarrow$ B: lengte 3
A $\longrightarrow$ B $\longrightarrow$ D: lengte 5
A $\longrightarrow$ B $\longrightarrow$ C: lengte 7
A $\longrightarrow$ B $\longrightarrow$ D $\longrightarrow$ E: lengte 9

Voorbeeld -1-

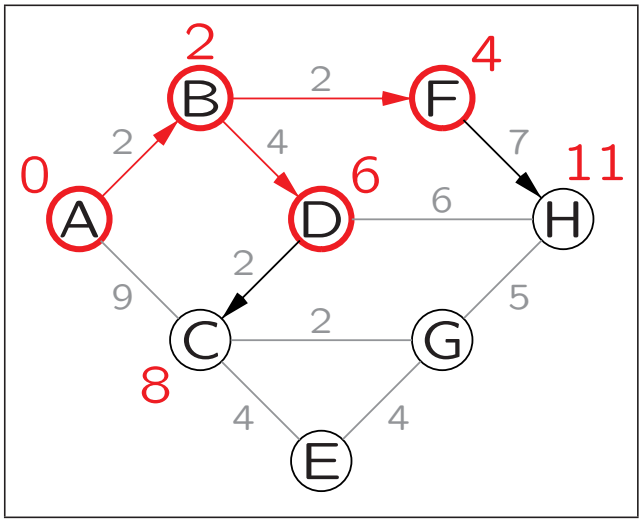

A
B
C
D
E
F
G
H
2
2
4
7
6
2
9
2
5
4
4

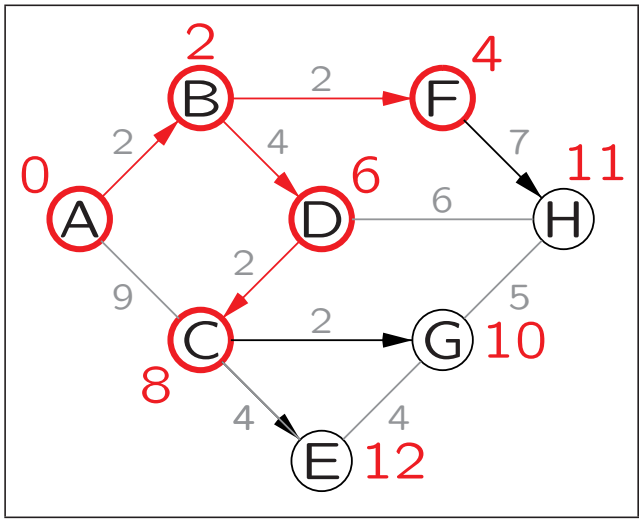

0
2
9
A
B
C
D
E
F
G
H
2
2
4
7
6
2
2
5
4
4

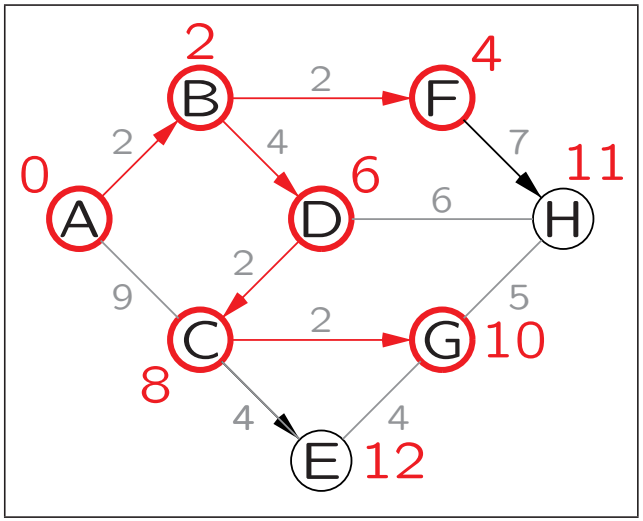

0
2
4
6
9
A
B
C
D
E
F
G
H
2
2
4
7
6
2
2
5
4
4

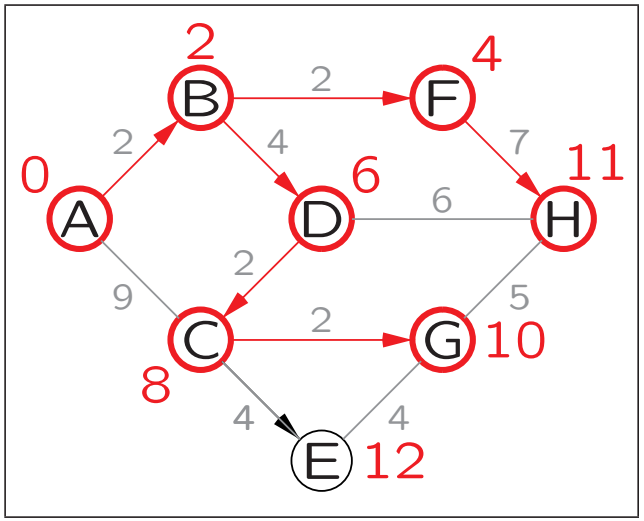

0
2
4
6
11
9
A
B
C
D
E
F
G
H
2
2
4
7
6
2
2
5
4
4

Voorbeeld -2-

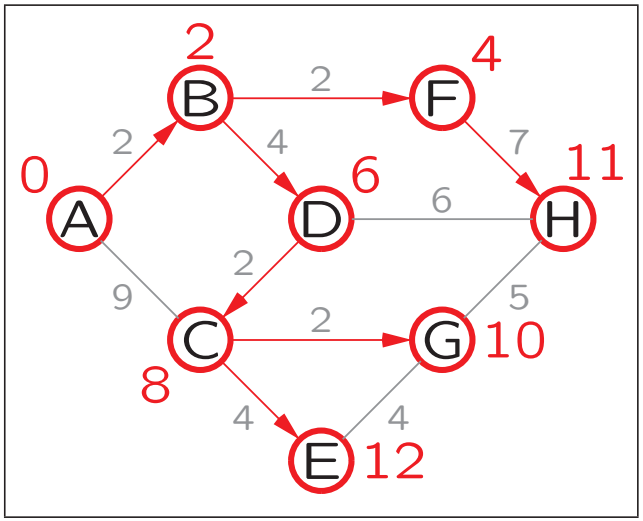

A
B
C
D
E
F
G
H
0
2
8
6
4
11
2
2
9
4
6
2
2
7
6
5
4
4


A
B
C
D
E
F
G
H
0
2
8
6
4
12
11
10
2
2
9
4
6
2
2
7
6
5
4
4

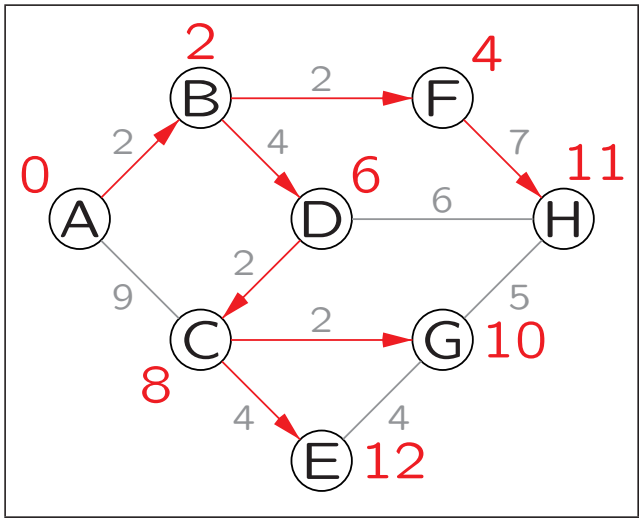

A
B
C
D
E
F
G
H
0
2
8
6
4
12
11
10
2
2
9
4
6
2
2
7
6
5
4
4


A
B
C
D
E
F
G
H
0
2
8
6
4
12
11
10
2
2
9
4
6
2
2
7
6
5
4
4

Voorbeeld -3-



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: $0 + 2 = 2$. 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, via B.

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

Kies D, afstand 6, via 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, kun je ook in een tabel aangeven wat er gebeurt.

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.

// 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 pad[v] = de lengte van een kortste pad van s naar v
for $v \in V$ **do**
    pad[v] := ∞; **od**
pad[s] := 0;
$U := \emptyset$;
while ($U \neq V$) **do**
    vind knoop $v^* \in V \setminus U$ met pad[v^*] minimaal;
    $U := U \cup \{v^*\}$;
    for alle knopen v aangrenzend aan v^* **do**
        if pad[v^*]+ gewicht(v^*, v) $<$ pad[v] **then**
            pad[v] := pad[v^*]+ 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 pad[v] al de definitieve afstand aan. Moet bewezen worden.
- Voor de andere knopen w geeft pad[w] steeds de kortste afstand vanuit s naar w via uitsluitend knopen uit U. Moet bewezen worden.
- De volgende dichtstbijzijnde knoop wordt gekozen uit de knopen uit $V \setminus U$ die direct grenzen aan U. Nadat deze gekozen is worden de labels aangepast.
- In elke stap wordt zo voor elke knoop uit $V \setminus U$ de kortste afstand vanaf s via de reeds afgehandelde knopen uit U (en bijbehorende takken) bepaald.
- Het is niet zo moeilijk dit algoritme aan te passen zodat ook de kortste paden zelf worden berekend.

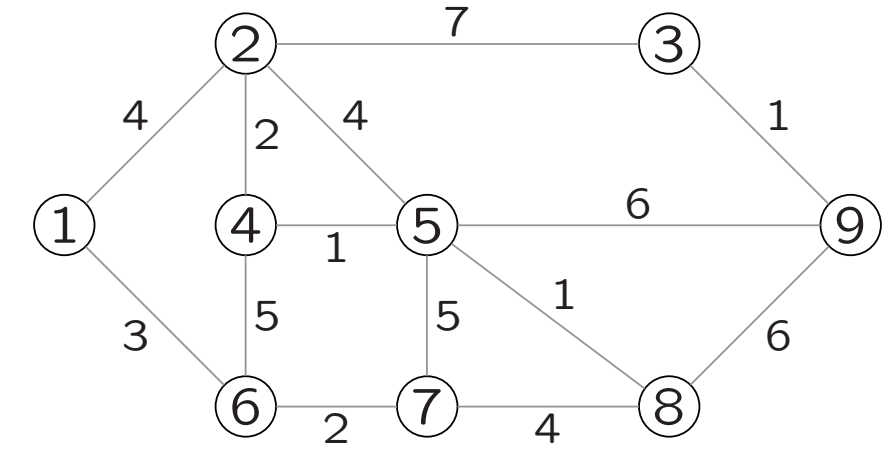
Na elke ronde (dus ook na de laatste, wanneer $U = V$) van het algoritme van Dijkstra geldt:

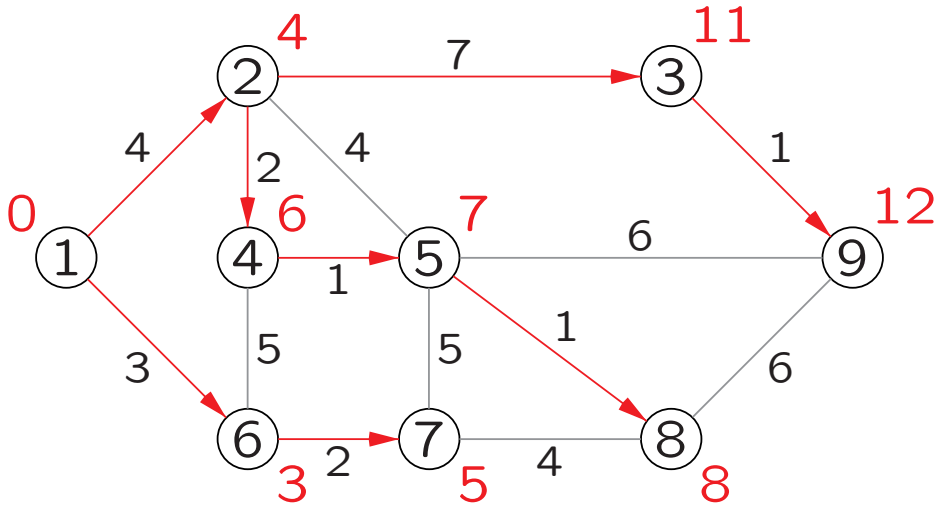
1. U bevat alle knopen waarvan de definitieve kortste afstand vanaf s reeds bepaald is. Voor elke $v \in U$ geeft het label pad[v] die kortste afstand aan.
2. Voor de andere knopen w ($w \notin U$) is pad[w] de kortste afstand vanuit s naar w via uitsluitend knopen uit U.

Om 1. en 2. te bewijzen moet je laten zien dat:

a. wanneer v^* wordt toegevoegd aan U, pad[v^*] inderdaad de lengte van het kortste pad van s naar v^* bevat
b. na elke update van de labels na toevoeging van v^* de pad[w] nog steeds de kortste afstand via (de nieuwe) U aangeven, voor alle $w \notin U$

Nog een voorbeeld





- **Deadline programmeeropdracht 2:** 19 april 2013
- **Lezen/leren bij dit college:**

  Inleiding hoofdstuk 9; paragraaf 9.3; sheets
- **Volgend werkcollege:**

  donderdag 25 april 2013 in zaal 412
- **Opgaven en programmeeropdrachten:**

  zie http://www.liacs.nl/home/graaf/ALGO/
- **Volgend college:** vrijdag 26 april 2013