



## Uitwerkingen

1. We vermenigvuldigen een rijtje matrices met dimensies 5, 10, 3, 12, 5. Er zijn totaal 405 vermenigvuldigingen nodig, en de optimale haakjes zijn  $(A \times B) \times (C \times D)$ , met matrices plus dimensies  $A : 5 \times 10$ ,  $B : 10 \times 3$ ,  $C : 3 \times 12$ ,  $D : 12 \times 5$ .

We vinden bijvoorbeeld:

$$N_{01} = N_{00} + N_{11} + d_0 d_1 d_2 = 0 + 0 + 150 = 150$$

$$N_{12} = N_{11} + N_{22} + d_1 d_2 d_3 = 0 + 0 + 360 = 360$$

$$N_{23} = N_{22} + N_{33} + d_2 d_3 d_4 = 0 + 0 + 180 = 180$$

En vervolgens (we geven hier direct de beste mogelijkheid):

$$N_{02} = N_{01} + N_{22} + d_0 d_2 d_3 = 150 + 0 + 180 = 330$$

$$N_{13} = N_{11} + N_{23} + d_1 d_2 d_4 = 0 + 180 + 150 = 330$$

En tenslotte:

$$N_{03} = N_{01} + N_{23} + d_0 d_2 d_4 = 150 + 180 + 75 = 405$$

2. Even alleen het idee: voeg aan het algoritme MatrixChain een extra parameter toe die bij een update van  $N_{ij}$  ook de haakjes-posities aangeeft. (Kan nog verder uitgewerkt worden.)
3. Een alternatief algoritme voor houthakken dat top-down werkt maar sub-resultaten hergebruikt. Input: een array  $p[1 \dots n]$  van prijzen, en een lengte  $n$ .

**Algorithm** houthakkenMemo( $p, n$ )

```
new array  $r[0 \dots n]$ 
for  $i := 1$  to  $n$  do
   $r[i] := -\infty$ 
return houthakkenAux( $p, n, r$ )
```

Hierbij wordt de volgende hulp-procedure gebruikt:

**Algorithm** houthakkenAux:( $p, n, r$ )

```

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

```

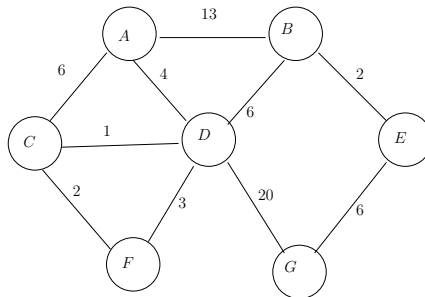
4. Een Longest Common Subsequence (LCS) van de rijtjes  $[1, 0, 0, 1, 0, 1, 0, 1]$  en  $[0, 1, 0, 1, 1, 0, 1, 1, 0]$  heeft lengte 6. Dit wordt gevonden door toepassen van het algoritme:

```

[0,0,0,0,0,0,0,0,0,0]
[0,0,1,1,1,1,1,1,1,1]
[0,1,1,2,2,2,2,2,2,2]
[0,1,1,2,2,2,3,3,3,3]
[0,1,2,2,3,3,3,4,4,4]
[0,1,2,3,3,3,4,4,4,5]
[0,1,2,3,4,4,4,5,5,5]
[0,1,2,3,4,4,5,5,5,6]
[0,1,2,3,4,5,5,6,6,6]

```

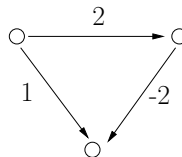
5. Pas Dijkstra's kortste pad algoritme toe op de volgende graaf, met  $A$  als startknoop. Geef stap voor stap aan wat er gebeurt, door in elke stap de graaf met de relevante data te tekenen.



De vraag is om de relevante data in de graaf aan te geven. We doen het hier eerst even in een tabel:

	A	B	C	D	E	F	G
$\emptyset$	0	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
{A}	0	13	6	4	$\infty$	$\infty$	$\infty$
{A, D}	0	10	5	4	$\infty$	7	24
{A, C, D}	0	10	5	4	$\infty$	7	24
{A, C, D, F}	0	10	5	4	$\infty$	7	24
{A, B, C, D, F}	0	10	5	4	12	7	24
{A, B, C, D, E, F}	0	10	5	4	12	7	18
{A, B, C, D, E, F, G}	0	10	5	4	12	7	18

6. Een klein voorbeeld waaruit blijkt dat Dijkstra's kortste pad algoritme niet (correct) werkt voor grafen met negatieve gewichten (zie ook werkcollege 11):



7. Als je in het voorbeeld van de vorige opgave de gewichten allemaal met 2 (of met 3 ophooft, krijg je een gerichte graaf met positieve gewichten. Daarop het algoritme van Dijkstra toepassen levert nog steeds niet het juiste kortste pad van linksboven naar middenonder.
8. Ja het algoritme is dan nog steeds correct. Alle burens van de knoop die als laatste uit  $Q$  wordt gehaald hebben op dat moment al de juiste waarde (dus het gewicht van de kortste pad vanaf de startknoop), dus het maakt niet uit als je de update niet uitvoert.

### Nog meer opgaven

- (Dit is ongeveer opgave R-5.2.)  
We representeren  $T$  door een priority queue, geïmplementeerd met een heap.
- Toon aan dat een expressie van  $n$  elementen met een binaire operator precies  $n - 1$  paren haakjes heeft.  
We kunnen de expressie zien als een binaire boom met de knopen als binaire operator en de bladeren als de elementen. Bij elke binaire operator (oftewel knoop) hoort precies één haakjespaar.  
Basisgeval:  $n = 2$ , dus twee elementen, oftewel twee bladeren. Dan is er één knoop, oftewel één binaire operator, dus ook één haakjespaar. Inductiestap: Stel een expressie met  $m$  elementen heeft  $m - 1$  haakjesparen.

We bekijken een expressie met  $m + 1$  elementen. In de boomrepresentatie voegen we een blad toe, en om dat aan te hechten dus ook ergens een knoop. Het aantal knopen was  $m - 1$ , en wordt dus  $m$ . Dus een binaire expressie met  $m + 1$  elementen heeft  $m$  haakjesparen.

3. Memoization toevoegen heeft geen zin in het geval van merge-sort omdat de subproblemen zoals die ook zichtbaar zijn in de recursie-boom niet overlappen. Er worden dus geen subresultaten meer dan eens berekend, en we kunnen dus geen tijdswinst boeken door het hergebruiken van subresultaten.
4. Het (dynamic programming) algoritme voor Longest Common Subsequence is in  $\mathcal{O}(mn)$ . De eerste for-loop kost  $m$  stappen, de tweede  $n$ , en de derde  $mn$ .
5. Hoe kun je het algoritme voor LCS aanpassen zó dat het de lengte van het langste monotoon (strict) stijgende gemeenschappelijke deelrijtje geeft?  
Idee:
6. Het idee van een algoritme voor deze aanpassing: voeg het item met het kleinste gewicht toe, net zolang tot dat niet meer kan omdat het maximum gewicht dan overschreden zou worden.
7. De edge-list implementatie: We hebben een lijst van knopen:

$[A, B, C, D, E, F, G]$

Verder is er een lijst met kanten. In de lijst met kanten geven we per kant expliciet aan welke knopen aan de twee uiteinden (in ons geval is er geen richting) zitten. We schrijven de lijst als tabel, het linker deel is de identifier van de kant, het rechterdeel zijn twee referenties naar knopen die zijn eindpunt zijn.

$AB,$	$[A, B]$
$AC,$	$[A, C]$
$AD,$	$[A, D]$
$BD,$	$[B, D]$
$BE,$	$[B, E]$
$CD,$	$[C, D]$
$CF,$	$[C, F]$
$DF,$	$[D, F]$
$DG,$	$[D, G]$
$EG$	$[E, G]$

Als we de lijst van knopen, en de lijst van kanten met referentie naar incidente knopen alletwee als gelinkte lijst implementeren dan is het opvragen

van alle buren van een knoop in  $\mathcal{O}(m)$  omdat we de hele lijst van kanten moeten doorlopen. (Hier gaan we uit van een graaf met  $n$  knopen en  $m$  kanten.)

De adjacency-list implementatie: De lijst van knopen bevat per knoop ook welke kanten er incident mee zijn.

$A,$   $[AB, AC, AD]$   
 $B,$   $[AB, BD, BE]$   
 $C,$   $[AC, CD, CF]$   
 $D,$   $[AD, BD, CD, DF, DG]$   
 $E,$   $[BE, EG]$   
 $F,$   $[CF, DF]$   
 $G,$   $[DG, EG]$

De lijst van kanten, met per kant een verwijzing naar de bijbehorende knopen (zoals in Figure 6.4 van het boek) is dan net als hierboven.