HO
GENT

Classic Computer Science Algorithms

Oplossingen Oefeningen

Stijn Lievens

Professionele Bachelor in de Toegepaste Informatica

Inhoudsopgave

HOOFDSTUK 1

Zoeken en Sorteren

1.3 Oefeningen

1. Ga in de array (1 2 3 4 6 7 8 9 10) achtereenvolgens op zoek naar de waarden 3, 5 en 10. Maak hierbij gebruik van het algoritme ZOEK-BINAIR. Houd tijdens je simulatie de waarden bij van de variabelen l, r en m.

2. (**Dodona**) Implementeer de iteratieve methode voor binair zoeken.

3. (**Dodona**) Herwerk en implementeer de oplossingsmethode voor sorteren door selectie zodat niet langer het grootste element naar achteren wordt gebracht maar het kleinste element naar voor. Het reeds gesorteerde deel van de array bevindt zich m.a.w. steeds vooraan.

4. Pas de oplossingsmethode beschreven in het opgegeven algoritme aan zodat de elementen niet langer van klein naar groot worden gesorteerd maar van groot naar klein.

   a) sorteren door tussenvoegen

   b) sorteren door mengen

5. (**Dodona**) Bubble sort is een oplossingsmethode voor het sorteren van een array a van lengte n. Algoritme 1.1 beschrijft deze methode in pseudo-code.

   a) Implementeer deze methode in Python.

Algoritme 1.1 De methode bubbleSort.

Invoer De array a is gevuld met n elementen.
Uitvoer De array a is gesorteerd.

```
1: function BUBBLESORT(a)
2:     for i = 0 ... n − 2 do
3:         for j = n − 1 ... i + 1 by −1 do
4:             if a[j − 1] > a[j] then
5:                 wissel de elementen a[j − 1] en a[j]
6:             end if
7:         end for
8:     end for
9: end function
```

b) Tel het exact aantal keer, in functie van n, dat de controle van regel 4, nl. $a[j-1] > a[j]$ wordt uitgevoerd. Gebruik een gesloten formule om je resultaat weer te geven. Leid hieruit de $\mathcal{O}$-notatie voor de uitvoeringstijd van het algoritme bubbleSort af.

c) Stel $a = (13, 7, 8, 1)$. Volg in een overzichtelijke tabel de uitwerking van BUBBLESORT(a). Noteer in de tabel minstens de verschillende waarden die i, j en a doorlopen.

6. Hieronder volgen een aantal codefragmenten f_i. Bepaal voor elk van deze fragmenten de waarde van x in functie van de invoerwaarde n. Leid hieruit de asymptotische uifvoeringstijd, t.t.z. de $\mathcal{O}$-notatie, af van deze verschillende codefragmenten. Tracht je resultaten experimenteel te verifiëren door een programma te schrijven dat de verhouding berekent van de $f_i(n)$ over je theoretisch voorspelde uitvoeringstijd. Wanneer je de juiste uitvoeringstijd hebt dan zou deze verhouding zo goed als constant moeten zijn (voor grote waarden van n).

a)
```
1: function SOM1(n)
2:     x ← 0
3:     for i = 1 ... n do
4:         x ← x + 1
5:     end for
6:     return (x)
7: end function
```

b)
```
1: function SOM2(n)
2:     x ← 0
3:     for i = 1 ... 2n do
```

```
4:          for j = 1 ... n do
5:              x ← x + 1
6:          end for
7:      end for
8:      return (x)
9:  end function
```

c)
```
1:  function SOM3(n)
2:      x ← 0
3:      for i = 1 ... n do
4:          for j = 1 ... i do
5:              for k = 1 ... j do
6:                  x ← x + 1
7:              end for
8:          end for
9:      end for
10:     return (x)
11: end function
```

d)
```
1:  function SOM4(n)
2:      x ← 0
3:      for i = 1 ... n do
4:          for j = 1 ... i do
5:              for k = 1 ... i do
6:                  x ← x + 1
7:              end for
8:          end for
9:      end for
10:     return (x)
11: end function
```

e)
```
1:  function SOM5(n)
2:      x ← 0
3:      i ← n
4:      while i ≥ 1 do
5:          x ← x + 1
6:          i ← ⌊i/2⌋
7:      end while
8:      return (x)
9:  end function
```

f)
```
1:  function SOM6(n)
```

```
2:      x ← 0
3:      i ← n
4:      while i ≥ 1 do
5:          for j = 1 . . . i do
6:              x ← x + 1
7:          end for
8:          i ← ⌊i/2⌋
9:      end while
10:     return (x)
11: end function
```

g)
```
1: function SOM7(n)
2:      x ← 0
3:      i ← n
4:      while i ≥ 1 do
5:          for j = 1 . . . n do
6:              x ← x + 1
7:          end for
8:          i ← ⌊i/2⌋
9:      end while
10:     return (x)
11: end function
```

1.4 Oplossingen

1. Er wordt gevraagd de methode zoekBinair uit te werken voor verschillende inputwaarden voor *zoekItem*. De array *rij* is steeds de gegeven array.
   De lengte van de gegeven array is 9, dus $n = 9$.

   In een tabel wordt weergegeven welke waarden de variabelen l, r en m achtereenvolgens zullen aannemen wanneer het algoritme wordt uitgevoerd voor de gegeven input. De waarde van $rij[m]$ wordt eveneens bijgehouden in de tabel.

   - $zoekItem = 3$.

l	r	m	$rij[m]$	
0	8			De voorwaarde van lijn 4 is voldaan.
		4	6	De voorwaarde van lijn 6 is niet voldaan.
	4			
				De lus wordt opnieuw uitgevoerd.
		2	3	De voorwaarde van lijn 6 is niet voldaan.
	2			
				De lus wordt opnieuw uitgevoerd.
		1	2	De voorwaarde van lijn 6 is voldaan.
2				De lus stopt.

Op het ogenblik dat de lus stopt is de waarde van $l = r = 2$. Het element $rij[2] = 3$. Dus de selectievoorwaarde van lijn 12 is niet voldaan. Lijn 15 wordt uitgevoerd. De variabele *index* wordt de waarde 2 toegewezen. Dit komt overeen met de index van het gezochte element in de gegeven array.

- $zoekItem = 5$.

l	r	m	$rij[m]$	
0	8			De voorwaarde van lijn 4 is voldaan.
		4	6	De voorwaarde van lijn 6 is niet voldaan.
	4			
				De lus wordt opnieuw uitgevoerd.
		2	3	De voorwaarde van lijn 6 is voldaan.
3				
				De lus wordt opnieuw uitgevoerd.
		3	4	De voorwaarde van lijn 6 is voldaan.
4				De lus stopt.

Op het ogenblik dat de lus stopt is de waarde van $l = r = 4$. Het element $rij[4] = 6$. De selectievoorwaarde van lijn 12 is voldaan. Lijn 13 wordt uitgevoerd. De variabele *index* wordt de waarde -1 toegewezen. Het element 5 komt inderdaad niet voor in de gegeven array.

- $zoekItem = 10$.

l	r	m	$rij[m]$	
0	8			De voorwaarde van lijn 4 is voldaan.
		4	6	De voorwaarde van lijn 6 is voldaan.
5				
				De lus wordt opnieuw uitgevoerd.
		6	8	De voorwaarde van lijn 6 is voldaan.
7				
				De lus wordt opnieuw uitgevoerd.
		7	9	De voorwaarde van lijn 6 is voldaan.
8				De lus stopt.

Op het ogenblik dat de lus stopt is de waarde van $l = r = 8$. Het element $rij[8] = 10$. De selectievoorwaarde van lijn 12 is niet voldaan. Lijn 15 wordt uitgevoerd. De variabele *index* wordt de waarde 8 toegewezen. Er geldt inderdaad: $rij[8] = 10$.

Onafhankelijk van de waarde van het te zoeken element werd de lus steeds drie keer uitgevoerd. De uitvoeringstijd voor de methode zoekBinair is niet afhankelijk van de positie van het te zoeken element. Staat het element helemaal vooraan of helemaal achteraan of komt het niet voor in de array, de lus wordt steeds evenveel keer herhaald.
De uitvoeringstijd is wel afhankelijk van de lengte van de array!

Zoals in de theoie wordt vermeld, is de uitvoeringstijd van het algoritme zoekBinair in alle situaties logaritmisch.

$$T(n) = \mathcal{O}(\lg n)$$

met n de lengte van de gegeven array.

2. Zie Dodona.

3. Zie Dodona.

4. a) De logica van het sorteeralgoritme blijft ongewijzigd, enkel de manier waarop elementen worden vergeleken wijzigt. Op de lijn

$$\textbf{while } j > 0 \textbf{ and } x < a[j-1] \textbf{ do}$$

wordt de correcte plaats bepaald voor het element $a[i]$ door de grotere elementen $a[j-1]$ een positie naar rechts op te schuiven (in de opdrachten van de lus). Het enige wat we moeten doen

om van groot naar klein te sorteren is de vergelijking $x < a[j-1]$ omkeren. Dit wordt m.a.w. de nieuwe lijn in een voor de rest ongewijzigd algoritme:

$$\textbf{while } j > 0 \textbf{ and } x > a[j-1] \textbf{ do}$$

b) Bij het sorteren door mengen worden de elementen effectief vergeleken en verplaatst in de functie MERGE. Op de lijn

$$\textbf{if } a[i] \leq a[j] \textbf{ then}$$

wordt bepaald dat welk element het kleinste is en dat wordt als volgende in de hulp geplaatst. Om van groot naar klein te sorteren moeten we dus enkel deze lijn aanpassen naar:

$$\textbf{if } a[i] \geq a[j] \textbf{ then}$$

De rest van het algoritme blijft ongewijzigd.

Opmerking: omdat de logica voor sorteren onafhankelijk is van het vergelijken van de elementen (i.e. het bepalen welk van twee elementen het kleinste is) wordt in Python de mogelijkheid gegeven om een "key"functie mee te geven met het sorteeralgoritme. Deze "key"-functie bepaalt de volgorde waarin de elementen gesorteerd zullen worden.

5. a) Zie Dodona.

b) Voor inputgrootte n geldt:

	voor i	wordt j	aantal j's
	0	$n-1, n-2, \ldots, 1$	$n-1$
	1	$n-1, n-2, \ldots, 2$	$n-2$
	$\vdots$	$\vdots$	$\vdots$
	$n-2$	$n-1$	1
Totaal aantal waarden	$n-1$		$1+2+\cdots+(n-1)$

Het totaal aantal keer dat de binnenste lus en dus eveneens lijn 4 zal uitgevoerd worden is

$$\sum_{i=1}^{n-1} i = 1+2+\cdots+(n-1) = \frac{(n-1)\cdot n}{2}.$$

Hieruit mogen we besluiten dat de uitvoeringstijd kwadratisch zal toenemen:

$$T(n) = \mathcal{O}(n^2).$$

b) We sorteren de array $a = (13, 7, 8, 1)$ met bubbleSort. De lengte van a is 4, dus $n = 4$.

i	j	a	Opmerking
		$(13,7,8,1)$	De input.
0			
	3		De voorwaarde van lijn 4 is voldaan.
		$(13,7,1,1)$	
		$(13,7,1,8)$	
	2		De voorwaarde van lijn 4 is voldaan.
		$(13,1,1,8)$	
		$(13,1,7,8)$	
	1		De voorwaarde van lijn 4 is voldaan.
		$(1,1,7,8)$	
		$(1,13,7,8)$	
	0		De iteratievoorwaarde van lijn 3 is niet langer voldaan.
1			
	3		De voorwaarde van lijn 4 is niet voldaan.
			Er wordt geen wissel uitgevoerd.
	2		De voorwaarde van lijn 4 is voldaan.
		$(1,7,7,8)$	
		$(1,7,13,8)$	
	1		De iteratievoorwaarde van lijn 3 is niet langer voldaan.
2			
	3		De voorwaarde van lijn 4 is voldaan.
		$(1,7,8,8)$	
		$(1,7,8,13)$	
	2		De iteratievoorwaarde van lijn 3 is niet langer voldaan.
3			De iteratievoorwaarde van de buitenste lus is niet langer voldaan

De gesorteerde array a werd teruggeven.

6. a) Telkens de teller i een nieuwe waarde aanneemt en $i \leq n$ geldig is, wordt de opdracht $x \leftarrow x + 1$ uitgevoerd. De teller i neemt alle waarden van 1 tot en met n aan. Dit betekent dat de binnenste lus n keer wordt uitgevoerd of dus

   - De opdracht $x \leftarrow x + 1$ wordt n keer uitgevoerd.
   - $T(n) = \mathcal{O}(n)$.

   Meer symbolisch:

$$x = \sum_{i=1}^{n} 1 = \underbrace{1 + 1 + \cdots + 1}_{n \text{ keer}} = n.$$

Dit bepaalt onmiddellijk de lineaire tijdscomplexiteit van dit codefragment.

b) Telkens j een nieuwe waarde krijgt en $j \leq n$ geldig is, wordt de opdracht $x \leftarrow x + 1$ uitgevoerd. We maken gebruik van een tabel om het aantal dergelijke j's te tellen.

	voor i	wordt j	aantal j's
	1	$1, 2, \ldots, n$	n
	2	$1, 2, \ldots, n$	n
	$\vdots$	$\vdots$	$\vdots$
	$2n$	$1, 2, \ldots, n$	n
Totaal aantal waarden	$2n$		$2n \cdot n$

Besluit:

- De opdracht $x \leftarrow x + 1$ wordt $2n^2$ keer uitgevoerd.
- $T(n) = \mathcal{O}(n^2)$.

Meer symbolisch:

$$x = \sum_{i=1}^{2n} \sum_{j=1}^{n} 1 = \sum_{i=1}^{2n} n = \underbrace{n + n + \cdots n}_{2n \text{ keer}} = 2n^2.$$

c) Verder bouwend op de vorige twee puntjes weten we dat

$$x = \sum_{i=1}^{n} \sum_{j=1}^{i} \sum_{k=1}^{j} 1.$$

De binnenste som (over k) evalueert naar j m.a.w.

$$x = \sum_{i=1}^{n} \sum_{j=1}^{i} j.$$

De som over j kennen we ook reeds:

$$\sum_{j=1}^{i} j = 1 + 2 + 3 + \cdots + i = \frac{(i+1)i}{2}.$$

De waarde voor x is m.a.w.

$$x = \sum_{i=1}^{n} \frac{(i+1)i}{2} = \frac{1}{2} \sum_{i=1}^{n} i^2 + \frac{1}{2} \sum_{i=1}^{n} i.$$

Men kan aantonen dat[1]:

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6},$$

zodat, na wat rekenwerk volgt dat

$$x = \frac{n(n+1)(n+2)}{6}.$$

Dit betekent dat $T(n) = \mathcal{O}(n^3)$.

Opmerking. Deze som volgt een mooi patroon:

$$\sum_{i=1}^{n} 1 = \frac{n}{1!}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{i} 1 = \frac{n(n+1)}{2!}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{i} \sum_{k=1}^{j} 1 = \frac{n(n+1)(n+2)}{3!}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{i} \sum_{k=1}^{j} \sum_{l=1}^{k} 1 = \frac{n(n+1)(n+2)(n+3)}{4!}$$

d) De waarde van x wordt in dit geval gegeven door:

$$x = \sum_{i=1}^{n} \sum_{j=1}^{i} \sum_{k=1}^{i} 1,$$

zoals voorheen kunnen we de binnenste twee sommaties gemakkelijk bepalen:

$$x = \sum_{i=1}^{n} i^2.$$

Deze waarde van deze som werd reeds (zonder bewijs) gegeven in vorig puntje:

$$x = \frac{n(n+1)(2n+1)}{6}.$$

Opnieuw geldt $T(n) = \mathcal{O}(n^3)$.

[1]Gebruik bv. Wolfram Alpha

e) De methode bevat slechts een enkele lus. We tellen het aantal i's waarvoor de opdracht $x \leftarrow x + 1$ wordt uitgevoerd. Om de analyse te vereenvoudigen veronderstellen we dat n een (natuurlijke) macht van 2 is, of anders gezegd, we kunnen n schrijven als 2^k, voor $k \in \mathbb{N}$. Merk op dat

$$k = \log_2(n) = \lg(n),$$

waarbij lg een verkorte schrijfwijze is voor de logaritmische functie met grondtal 2.

	i	aantal i's
	$n = 2^k$	1
	$\frac{n}{2} = 2^{k-1}$	1
	$\frac{n}{4} = 2^{k-2}$	1
	$\vdots$	$\vdots$
	$\frac{n}{n} = 2^{k-k} = 2^0$	1
Totaal		$k+1$

met

$$k + 1 = \log_2 n + 1 \quad (n = 2^k).$$

Besluit:

- De opdracht $x \leftarrow x + 1$ wordt $1 + \log_2 n$ keer uitgevoerd.
- $T(n) = \mathcal{O}\big(\lg(n)\big)$.

f) We veronderstellen opnieuw dat $n = 2^k$ met $k \in \mathbb{N}$. Dit laat ons toe, het aantal keer dat de instructie $x \leftarrow x + 1$ wordt uitgevoerd, exact te tellen.

	voor i	wordt j	aantal j's
	$n = 2^k$	$1, \ldots, 2^k$	2^k
	$\frac{n}{2} = 2^{k-1}$	$1, \ldots, 2^{k-1}$	2^{k-1}
	$\frac{n}{2^2} = 2^{k-2}$	$1, \ldots, 2^{k-2}$	2^{k-2}
	$\vdots$	$\vdots$	$\vdots$
	$\frac{n}{2^k} = 2^{k-k} = 2^0$	1	1
Totaal			$2^0 + 2^1 \cdots + 2^{k-2} + 2^{k-1} + 2^k$

Nu geldt

$$\begin{aligned} 2^0 + 2^1 + \cdots + 2^{k-2} + 2^{k-1} + 2^k &= \sum_{l=0}^{k} 2^l \\ &= \frac{2^{k+1} - 1}{2 - 1} \\ &= 2n - 1 \end{aligned}$$

Besluit:

- De opdracht $x \leftarrow x + 1$ wordt $2n - 1$ keer uitgevoerd.
- $T(n) = \mathcal{O}(n)$.

We zien hier m.a.w. een geneste lus met een lineaire tijdscomplexiteit.

g) We veronderstellen opnieuw dat $n = 2^k$, met $k \in \mathbb{N}$, en dus $k = \log_2 n$.

	voor i	wordt j	aantal j's
	$n = 2^k$	$1, \ldots, n$	n
	$\frac{n}{2} = 2^{k-1}$	$1, \ldots, n$	n
	$\frac{n}{2^2} = 2^{k-2}$	$1, \ldots, n$	n
	$\vdots$	$\vdots$	$\vdots$
	$\frac{n}{2^k} = 2^{k-k} = 2^0 = 1$	$1, \ldots, n$	n
Totaal			$(k+1)n$

met

$$(k+1)n = (\log_2 n + 1)n = n \log_2 n + n.$$

Besluit:

- De opdracht $x \leftarrow x + 1$ wordt $n \lg(n) + n$ keer uitgevoerd.
- $T(n) = \mathcal{O}\big(n \lg n\big)$. De lineaire term in $n \lg(n) + n$ wordt immers gedomineerd door de term $n \lg(n)$ wanneer n zeer groot wordt. Dit komt omdat $\lg(n)$ ook oneindig groot wordt, maar wel zeer traag!

HOOFDSTUK 2

Gelinkte Lijsten

2.6 Oefeningen

1. Breid de klasse GelinkteLijst uit met een methode *size*. De methode heeft als resultaat het aantal elementen van een gelinkte lijst.

2. a) Pas in de klasse GelinkteLijst de constructor GelinkteLijst aan zodat de methode een lege gelinkte lijst met ankercomponent aanmaakt.

   b) Breid deze klasse uit met een methode invert. Deze methode heeft als resultaat een gelinkte lijst l_2. De gelinkte lijst l_2 bevat dezelfde elementen als een bestaande gelinkte lijst l_1 maar de elementen komen voor in omgekeerde volgorde. De gelinkte lijst l_1 is na afloop ongewijzigd.

3. Pas de klasse Knoop aan zodat met de objecten van deze klasse een dubbelgelinkte lijst kan aangemaakt worden. Noem de nieuwe klasse KnoopDubbel.
   Schrijf vervolgens voor dubbelgelinkte lijsten met twee ankercomponenten de basisfuncties van de klasse DubbelGelinkteLijst in pseudocode uit.

DubbelGelinkteLijst
- *eerste* : KnoopDubbel - *laatste*: KnoopDubbel
+ DubbelGelinkteLijst( ) + verwijder(*ref*: KnoopDubbel) : Element + voegToeVoor(*ref*: KnoopDubbel, *x*: Element) : / + voegToeNa(*ref*: KnoopDubbel, *x*: Element) : / + zoek(*x*: Element) : *ref*: KnoopDubbel

De klasse KnoopDubbel wordt als inwendige klasse (inner class) van de klasse DubbelGelinkteLijst geïmplementeerd.

4. Een wachtrij is een FIFO-datastructuur. Het element dat het eerst op de wachtrij werd geplaatst is ook het eerste dat wordt verwijderd. Een wachtrij heeft m.a.w. twee uiteinden: een *kop* en een *staart*. Elementen worden toegevoegd aan de kant van de staart en verwijderd aan de kant van de kop.

   a) Een wachtrij kan geïmplementeerd worden door twee referenties k en s. De knoop k refereert naar de kop van de wachtrij, de knoop s refereert naar de staart van de wachtrij.
   Herschrijf alle basisbewerkingen voor een wachtrij corresponderend met deze implementatie. Deze zijn
      - *Queue*( ): constructor, maakt een nieuwe wachtrij aan waarna de wachtrij bestaat als lege wachtrij;
      - *empty*( ): controleert of een wachtrij al dan niet leeg is;
      - *enqueue*( ): voegt een gegeven element toe aan de staart van een wachtrij;
      - *dequeue*( ): verwijdert het element aan de kop in een wachtrij en retourneert het verwijderde element;
      - *front*( ): retourneert het voorste element, m.a.w. de kop, van een wachtrij, zonder het te verwijderen.

   b) Breid deze klasse uit met een methode *invert*. Deze methode plaatst de elementen van een bestaande wachtrij in omgekeerde volgorde.
   Je algoritme mag geen nieuwe knopen alloceren. Je code mag tevens het veld 'data' van geen enkele knoop wijzigen.

5. Implementeer de methodes om

- een postfix uitdrukking te evalueren
- een infix uitdrukking om te zetten naar een postfix uitdrukking
- een infix uitdrukking te evalueren. Dit is dan een soort van eenvoudige rekenmachine.

2.7 Oplossingen

1. Het basisidee is om de gelinkte lijst te overlopen van begin tot einde en in elke iteratie een teller te verhogen.

   Invoer de gelinkte lijst is aangemaakt.

   Uitvoer het aantal elementen in de gelinkte lijst werd berekend en geretourneerd.

   ```
   1: function SIZE
   2:     aantal ← 0
   3:     ref ← eerste
   4:     while ref ≠ null do
   5:         aantal ← aantal + 1
   6:         ref ← ref.volgende
   7:     end while
   8:     return aantal
   9: end function
   ```

2. a) De ankercomponent is nu een "dummy" Knoop.

      Invoer /

      Uitvoer er werd een nieuwe (lege) gelinkte lijst aangemaakt

      ```
      1: function GELINKTELIJST
      2:     eerste ← nieuwe Knoop()
      3: end function
      ```

   b) Om de nieuwe gelinkte lijst aan te maken overlopen we de originele gelinkte lijst (van voor naar achter). Voor elke Knoop wordt er een nieuwe Knoop aangemaakt die net na de ankercomponent wordt toegevoegd aan de nieuwe lijst. Op die manier wordt de volgorde omgekeerd.

      Invoer de gelinkte lijst bestaat

      Uitvoer de gelinkte lijst l_2 werd geretourneerd. De gelinkte lijst l_2 bevat dezelfde waarden als de (eerste) gelinkte lijst maar in omgekeerde volgorde.

```
1: function INVERT
2:     l_2 ← nieuwe GelinkteLijst()
3:     ref ← eerste.volgende        ▷ verwijst impliciet naar "this"
4:     while ref ≠ null do
5:         x ← ref.data
6:         l_2.voegToe(l_2.eerste, x)
7:         ref ← ref.volgende
8:     end while
9:     return l_2
10: end function
```

3. Het UML-diagram van de klasse KnoopDubbel is eenvoudig.

KnoopDubbel
- *data* : Element - *vorige* : KnoopDubbel - *volgende* : KnoopDubbel
+ KnoopDubbel( )

Het aanmaken van een nieuwe KnoopDubbel is eveneens voor de hand liggend.

Invoer
Uitvoer een nieuwe KnoopDubbel werd aangemaakt

```
1: function KNOOPDUBBEL
2:     data ← null
3:     vorige ← null
4:     volgende ← null
5: end function
```

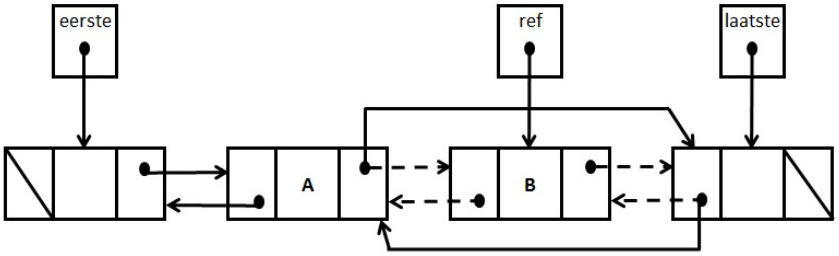
Bij een dubbelgelinkte lijst met ankercomponenten die leeg is wijst *volgende* component van *eerste* naar *laatste*, en de *vorige* component van *laatste* wijst naar *eerste*. Dat is dan ook de manier om een lege dubbelgelinkte lijst aan te maken.

Invoer
Uitvoer een lege dubbelgelinkte lijst werd aangemaakt

```
1: function KNOOPDUBBEL
2:     eerste ← nieuwe KnoopDubbel()
3:     laatste ← nieuwe KnoopDubbel()
4:     eerste.volgende ← laatste
```

5: laatste.vorige ← eerste
6: **end function**

Figuur 2.1 illustreert het verwijderen van een knoop in een dubbelgelinkte lijst. De implementatie hiervan komt voornamelijk neer op het aanpassen van twee referenties, nl. één referentie in de KnoopDubbel voor en na de huidige KnoopDubbel.

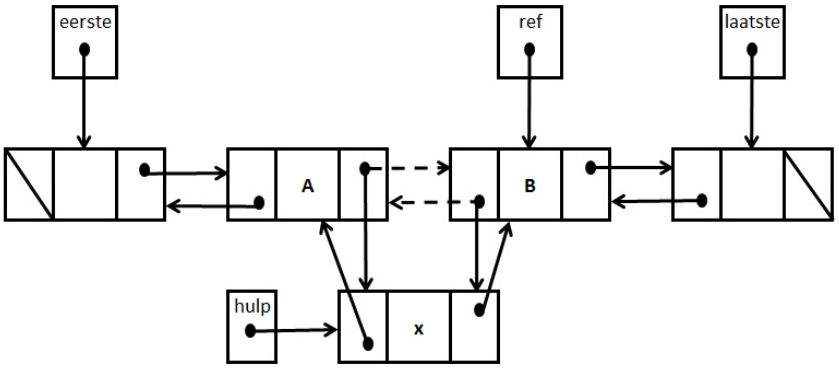


Figuur 2.1: De knoop met referentie *ref* verwijderen uit een dubbelgelinkte lijst.

Invoer *ref* verwijst naar een geldige KnoopDubbel binnen de dubbelgelinkte lijst
Uitvoer De KnoopDubbel *ref* werd verwijderd uit de dubbelgelinkte lijst. De waarde in deze KnoopDubbel werd geretourneerd
1: **function** VERWIJDER(ref)
2: $x \leftarrow$ ref.data
3: ref.volgende.vorige ← ref.vorige
4: ref.vorige.volgende ← ref.volgende
5: **return** x
6: **end function**

Figuur 2.2 illustreert de werkwijze voor het toevoegen van een knoop vóór een gegeven knoop *ref*. Er moet een nieuwe KnoopDubbel aangemaakt worden en 4 referenties moeten een nieuwe waarde krijgen. Het is belangrijk deze referenties in de juiste volgorde aan te passen.

Invoer *ref* wijst naar een KnoopDubbel in de dubbelgelinkte lijst
Uitvoer een nieuwe KnoopDubbel met de waarde x werd ingevoegd vóór de aangewezen KnoopDubbel
1: **function** VOEGTOEVOOR(ref, x)



Figuur 2.2: Een nieuwe knoop toevoegen vóór een gegeven referentie *ref*.

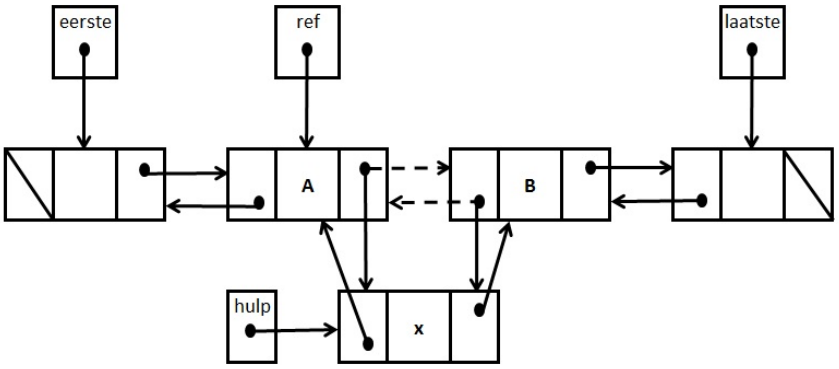
```
2:     hulp ← nieuwe KnoopDubbel()
3:     hulp.data ← x
4:     hulp.volgende ← ref
5:     hulp.vorige ← ref.vorige
6:     ref.vorige.volgende ← hulp
7:     ref.vorige ← hulp
8: end function
```

Figuur 2.3 illustreert de werkwijze voor het toevoegen van een knoop na een gegeven knoop *ref*. Je ziet onmiddellijk dat dit hetzelfde stramien volgt als invoegen vóór een gegeven knoop. De implementatie is dan ook zeer gelijklopend. Je kan echter de implementatie nog korter maken door gebruik te maken van de implementatie van de methode voegToeVoor.

Invoer *ref* wijst naar een KnoopDubbel in de dubbelgelinkte lijst
Uitvoer een nieuwe KnoopDubbel met de waarde x werd ingevoegd na de aangewezen KnoopDubbel

```
1: function VOEGTOENA(ref, x)
2:     voegToeVoor(ref.volgende, x)
3: end function
```

We zoeken vooraan en achteraan m.b.v. twee referenties, ref_1 en ref_2. We stoppen met zoeken wanneer het element x is gevonden of wan-



Figuur 2.3: Een nieuwe knoop toevoegen na een gegeven referentie *ref*. Dit is uiteraard equivalent met het invoegen van de nieuwe knoop voor de knoop met de waarde *B*.

neer de beide referenties het midden bereikt hebben. Bij een oneven aantal knopen zullen de beide referenties in het midden samenvallen. Bij een even aantal knopen is het midden bereikt wanneer de beide referenties naast elkaar staan.

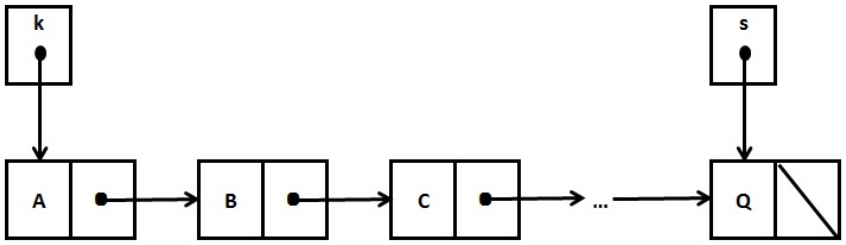
Invoer de dubbelgelinkte lijst bestaat.

Uitvoer de referentie naar een knoop met dataveld x werd geretourneerd, indien x niet voorkomt in de lijst werd de referentie null geretourneerd.

```
1:  function ZOEK(x)
2:      ref ← null
3:      ref_1 ← eerste.volgende
4:      ref_2 ← laatste.vorige
5:      while ref_1 ≠ ref_2 and ref_1.vorige ≠ ref_2 and ref_1.data ≠ x and
    ref_2.data ≠ x do
6:          ref_1 ← ref_1.volgende
7:          ref_2 ← ref_2.vorige
8:      end while
9:      if ref_1 ≠ laatste and ref_1.data = x then
10:         ref ← ref_1
11:     else
```

```
12:         if ref2 ≠ eerste and ref2.data = x then
13:             ref ← ref2
14:         end if
15:     end if
16:     return ref
17: end function
```

4. a) Figuur 2.4 is een illustratie van een wachtrij geïmplementeerd als gelinkte lijst. De wijzers k en s zijn zodanig gekozen dat het gemakkelijk is om het element bij k te verwijderen, en dat het eveneens gemakkelijk is om een element toe te voegen aan de kant van s.



Figuur 2.4: Een wachtrij geïmplementeerd als een gelinkte lijst.

De klasse Queue in UML

Queue
- k : Knoop - s : Knoop
+ Queue( ) + empty( ) : boolean + enqueue(x : Element) : / + dequeue( ) : Element + front( ) : Element

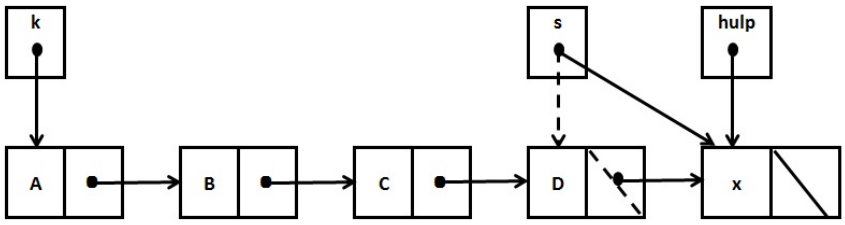
De constructor De constructor Queue maakt een nieuw object van de klasse Queue aan. We gebruiken in dit geval *geen* ankercomponenten.

Invoer
Uitvoer er werd een nieuwe lege wachtrij aangemaakt
1: **function** QUEUE
2: $k \leftarrow$ null
3: $s \leftarrow$ null
4: **end function**

Controleren of een wachtrij leeg is De implementatie zal ervoor zorgen dat de wachtrij leeg is als en slechts als k en s gelijk zijn aan null. Bovendien zullen k en s ofwel steeds allebei gelijk zijn aan null of allebei verschillend zijn van null, zodat het voldoende is om slechts één van de twee wijzers te controleren.

Invoer de wachtrij bestaat
Uitvoer de returnwaarde is **true** als en slechts de wachtrij leeg is, en **false** in het andere geval
1: **function** EMPTY
2: **return** $k =$ null
3: **end function**

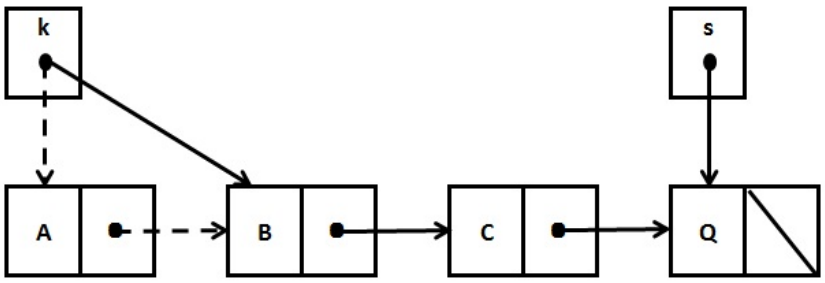
Toevoegen van een element aan een wachtrij Figuur 2.5 illustreert het toevoegen van een knoop aan een wachtrij die geïmplementeerd is als gelinkte lijst. Er moet een nieuwe knoop worden aangemaakt en ingeschakeld in de lijst. Ook de wijzer s moet worden aangepast. Als speciaal geval moet ook rekening worden gehouden met het feit dat de wachtrij (a priori) leeg zou kunnen zijn, in dit geval moet ook de kop k worden aangepast.



Figuur 2.5: Een element toevoegen aan een wachtrij.

```
1: function ENQUEUE(x)
2:     hulp ← nieuwe Knoop( )
3:     hulp.data ← x
4:     hulp.volgende ← null                ▷ Voor de duidelijkheid
5:     if empty() then                     ▷ verwijst naar deze wachtrij
6:         k ← hulp
7:     else
8:         s.volgende ← hulp                                  ▷ s ≠ null
9:     end if
10:    s ← hulp
11: end function
```

Verwijderen van een element uit een wachtrij Figuur 2.6 illustreert het verwijderen van een element aan de kop van een wachtrij die geïmplementeerd is als gelinkte lijst. Die komt hoofdzakelijk neer op het wijzigen van de kop. Indien we echter het laatste element verwijderen, dan zetten we ook de staart s op null.



Figuur 2.6: Een element verwijderen aan de kop van een wachtrij.

Invoer de wachtrij bestaat en is niet leeg
Uitvoer de kop van de wachtrij werd verwijderd en geretourneerd

```
1: function DEQUEUE
2:     x ← k.data
3:     if k = s then      ▷ in dit geval bevat q slechts één element
```

```
4:        k ← null
5:        s ← null
6:    else
7:        k ← k.volgende
8:    end if
9:    return x
10: end function
```

De kop van de wachtrij retourneren Het is zeer eenvoudig om de waarde van de kop terug te geven.

Invoer de wachtrij bestaat en is niet leeg
Uitvoer het data-element vooraan de wachtrij werd geretourneerd

```
1: function FRONT
2:     return k.data
3: end function
```

b) De wachtrij omkeren kan gebeuren zonder nieuwe knopen aan te maken. Er worden enkel wijzers verplaatst. In essentie moet het *volgende*-veld van elke knoop verwijzen naar zijn voorganger in de lijst (i.e. de knoop die ernaar wijst). Echter, alvorens het *volgende*-veld te overschrijven moet ook een referentie naar de opvolger worden bijgehouden omdat we daar anders niet meer kunnen geraken. Dit proces wordt herhaald totdat het einde van de gelinkte lijst wordt bereikt. Op dat moment wordt de rol van k en s omgewisseld.

Invoer de wachtrij bestaat
Uitvoer de wachtrij werd omgekeerd

```
1: function INVERT
2:     vorige ← null
3:     huidige ← k
4:     while huidige ≠ null do
5:         volgende ← huidige.volgende  ▷ referentie bijhouden
6:         huidige.volgende ← vorige    ▷ wijzer omkeren
7:         vorige ← huidige             ▷ doorschuiven
8:         huidige ← volgende
9:     end while
10:    k, s ← s, k                      ▷ Wisselen k en s
11: end function
```

5. Zie Dodona

HOOFDSTUK 3

Hashtabellen

3.4 Oefeningen

1. Voeg alle (verschillende) letters uit het woord 'DEMOCRATISCH' toe aan een hashtabel. De grootte van de hashtabel wordt bepaald door $N = 5$, dus er zijn vijf plaatsen (buckets) in de hashtabel. In de hashtabel wordt er geen waarde opgeslagen corresponderend met de letters. De te gebruiken hashcode is $11 \times k$ met k de positie van de letter in het alfabet.

   Hoe evolueert de hashtabel?

2. Beschouw het volgende probleem: gegeven een array a bestaande uit gehele getallen en een geheel getal t. Retourneer twee verschillende indices i en j zodanig dat $a_i + a_j = t$.

   a) De meest voor de hand liggende oplossing is om alle koppels (i, j) met $i < j$ te overlopen en te verifiëren of $a_i + a_j = t$. Schrijf deze oplossing uit in pseudocode en analyseer de tijdscomplexiteit.

   b) Op welke manier kan je gebruikmaken van een hashtabel om deze tijdscomplexiteit te verbeteren? Je mag er hierbij van uitgaan dat toevoegen aan en opzoeken in de hashtabel kan gebeuren in constante tijd. Vergelijk de hoeveelheid extra geheugen die deze oplossing nodig heeft met de oplossing van vorig puntje.

3.5 Oplossingen

1. De opgegeven letters moeten aan de hashtabel toegevoegd worden.

   - Een letter kan op unieke manier voorgesteld worden door zijn positie in het alfabet. De letter A staat op de eerste positie, de letter B op de tweede positie, enz..
   - De hashcode corresponderend met een letter werd gegeven. De hashcode is $11 \times k$ met k de positie van de letter in het alfabet.

     $$\text{hashcode} : \textit{letter} \mapsto 11 \times k$$

   - Een mogelijke hashfunctie die zal zorgen voor een gelijke verdeling van de overlap is de hashcode (mod N). Aangezien N een priemgetal is, is dit een goede keuze.

     $$h : \textit{letter} \mapsto \textit{letter}.\text{hashcode}(\,) \pmod{N}$$

   We plaatsen de resultaten voor de verschillende letters in een tabel.

Letter	k	Hashcode	Positie in hashtabel (bucket)	
D	4	$11 \times 4 = 44$	$44 \pmod 5 = 4$	
E	5	$11 \times 5 = 55$	$55 \pmod 5 = 0$	
M	13	$11 \times 13 = 143$	$143 \pmod 5 = 3$	
O	15	$11 \times 15 = 165$	$165 \pmod 5 = 0$	$\rightarrow$ overlapping
C	3	$11 \times 3 = 33$	$33 \pmod 5 = 3$	$\rightarrow$ overlapping
R	18	$11 \times 18 = 198$	$198 \pmod 5 = 3$	$\rightarrow$ overlapping
A	1	$11 \times 1 = 11$	$11 \pmod 5 = 1$	
T	20	$11 \times 20 = 220$	$220 \pmod 5 = 0$	$\rightarrow$ overlapping
I	9	$11 \times 9 = 99$	$99 \pmod 5 = 4$	$\rightarrow$ overlapping
S	19	$11 \times 19 = 209$	$209 \pmod 5 = 4$	$\rightarrow$ overlapping
H	8	$11 \times 8 = 88$	$88 \pmod 5 = 3$	$\rightarrow$ overlapping

   Verschillende letters komen op dezelfde positie terecht, m.a.w. er is overlap. Er zijn verschillende manieren om met deze overlap om te gaan. We bekijken twee mogelijke oplossingen.

   Gesloten hashing

   Alle letters komen in de tabel te staan op de corresponderende positie. Wanneer twee letters botsen moet de toe te voegen letter doorschuiven

naar de eerste vrije positie. Voor deze toepassing evolueert de tabel als volgt:

t_0	t_1	t_2	t_3	t_4	opmerkingen:
				D	de letter D moet op de 4-de positie
E				D	
E			M	D	
E	O		M	D	O moet doorschuiven naar de eerste vrije positie
E	O	C	M	D	C moet doorschuiven naar de eerste vrije positie

De tabel is vol. Er is geen plaats meer voor de resterende letters. De tabel moet gekopieerd worden naar een grotere tabel, bijvoorbeeld een tabel van grootte $N = 17$.
De positie van de reeds opgeslagen waarden moet opnieuw berekend worden. De tabel wordt groter dus de hashfunctie moet aangepast worden. Een mogelijke hashfunctie voor de nieuwe tabel is

$$h : letter \mapsto letter.\text{hashcode}(\,) \pmod{17}.$$

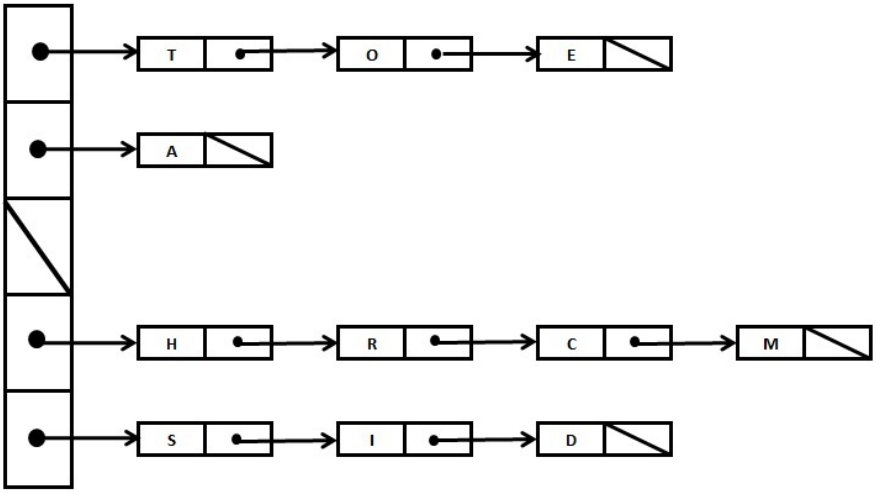
Open hashing

Alle letters, die op eenzelfde positie terecht komen, worden in dezelfde gelinkte lijst opgeslagen. Figuur 3.1 stelt de corresponderende hashtabel voor.

2. a) Deze oplossingsmethode heeft een tijdscomlexitiet van $\mathcal{O}(n^2)$ waarbij n de lengte van de rij a voorstelt. De hoeveelheid (extra) geheugen is constant.

   b) Initialiseer een lege hashtabel h. Overloop de array a van voor naar achter. Controleer op positie i of $t - a_i$ tot de hashtabel behoort. Als dit het geval is retourneer dan het tupel bestaande uit de waarde die geassocieerd is met $t - a_i$ in de hashtabel en i. Als $t - a_i$ niet tot de hashtabel behoort voeg dan de associatie $a_i \to i$ aan de hashtabel.

   Deze oplossingsmethode heeft een lineaire tijdscomplexiteit, maar heeft wel $\mathcal{O}(n)$ extra geheugen nodig (voor de hashtabel).



Figuur 3.1: De opgebouwde hashtabel.

HOOFDSTUK 4

Bomen

4.1.1 Oefeningen

1. Bekijk de boom in Figuur 4.1. Beantwoord de volgende vragen:

   a) Geef de wortel van de boom.

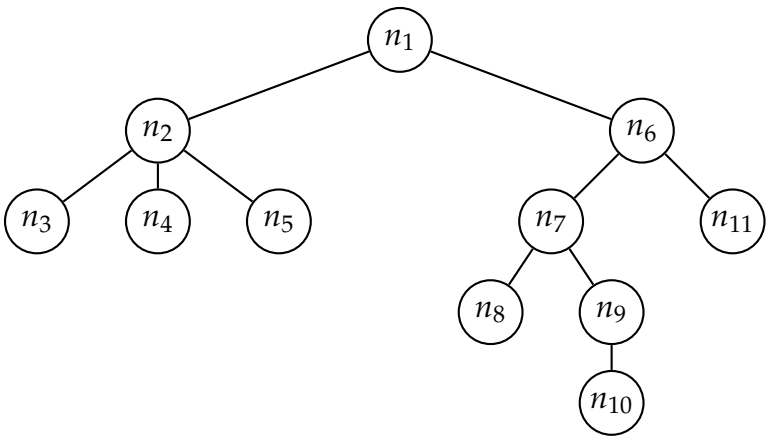
   b) Geef de verzamelingen T_1 t.e.m. T_m volgens Definitie 4.1.

   c) Geef de kinderen van elke top in de boom.

   d) Geef de graad van elke top in de boom.

   e) Geef de graad van de boom T.

   f) Welke toppen zijn broers van elkaar?



Figuur 4.1: Een voorbeeldboom.

g) Geef de bladeren van de boom.

h) Geef de afstammelingen van n_6.

i) Geef de voorouders van n_{10}.

j) Maak een tabel waarin voor elke top zijn hoogte en diepte wordt gegeven.

4.1.2 Oplossingen

1. a) De wortel van de boom is de top n_1.

b) Er zijn twee deelbomen T_1 en T_2 die gegeven worden door

$$T_1 = \{n_2, n_3, n_4, n_5\}$$
$$T_2 = \{n_6, n_7, n_8, n_9, n_{10}, n_{11}\}.$$

c) In tabelvorm vinden we

top	kinderen
n_1	n_2, n_6
n_2	n_3, n_4, n_5
n_6	n_7, n_{11}
n_7	n_8, n_9
n_9	n_{10}

De andere toppen hebben geen kinderen.

d) De graad van een top is het aantal kinderen van die top. We leiden het antwoord dus zeer eenvoudig af uit het antwoord op de vorige vraag. We antwoorden opnieuw in tabelvorm

top	graad	top	graad
n_1	2	n_7	2
n_2	3	n_8	0
n_3	0	n_9	1
n_4	0	n_{10}	0
n_5	0	n_{11}	0
n_6	2		

e) De graad van de *boom* is de *maximale graad van zijn toppen*. In dit geval is de graad van de boom dus 3.

f) Toppen zijn broers van elkaar als ze dezelfde ouder hebben. We vinden nu gemakkelijk dat de volgende verzamelingen toppen bestaan uit broers:

$$\{n_2, n_6\}, \quad \{n_3, n_4, n_5\}, \quad \{n_7, n_{11}\}, \quad \{n_8, n_9\}.$$

g) De bladeren van de boom zijn de toppen zonder kinderen, of anders gezegd de toppen met graad nul. Dit zijn de volgende toppen:

$$\{n_3, n_4, n_5, n_8, n_{10}, n_{11}\}.$$

h) De afstammelingen van de top n_6 zijn de kinderen van n_6 samen met de afstammelingen (recursief) van deze kinderen. We vinden:

$$\{n_7, n_{11}, n_8, n_9, n_{10}\}.$$

i) De voorouders van n_{10} zijn de ouder van n_{10} (i.e. de top n_9) samen met de voorouders (recursief) van deze ouder:

$$\{n_9, n_7, n_6, n_1\}.$$

j) De diepte van een top geeft de "afstand" tot de wortel; de hoogte van een blad is steeds nul en voor een andere top één meer dan de maximum hoogte van zijn kinderen.

top	hoogte	diepte	top	hoogte	diepte
n_1	4	0	n_7	2	2
n_2	1	1	n_8	0	3
n_3	0	2	n_9	1	3
n_4	0	2	n_{10}	0	4
n_5	0	2	n_{11}	0	2
n_6	3	1			

4.2.3 Oefeningen

1. Bereken hoeveel null-referenties er zullen zijn bij de array-van-kinderen voorstelling van de boom in Figuur 4.1. Wat is de verhouding van het aantal effectief gebruikte referenties tot het aantal voorziene referenties?

2. Teken de array-van-kinderen voorstelling van de boom in Figuur 4.1.

3. Bereken hoeveel null-referenties er zullen zijn bij eerste-kind-volgende-broer voorstelling van de boom in Figuur 4.1. Wat is de verhouding van het aantal effectief gebruikte referenties tot het aantal voorziene referenties?

4. Wat is de verhouding van het aantal effectief gebruikte referenties tot het aantal voorziene referenties voor een willekeurige gewortelde boom van graad k met n toppen.

5. Teken de eerste-kind-volgende-broer voorstelling van de boom in Figuur 4.1.

4.2.4 Oplossingen

1. De voorbeeldboom heeft 11 toppen en de graad van de *boom* is 3. Er zijn in totaal dus $11 \times 3 = 33$ referenties wanneer we de array-van-kinderen voorstelling gebruiken.

   Het aantal effectief gebruikte referenties is $11 - 1 = 10$. Wanneer we de pijlen omkeren dan wijst elke top naar zijn ouder, en iedere top behalve de wortel heeft juist één ouder. Er is dus steeds één pijl minder dan er toppen zijn.

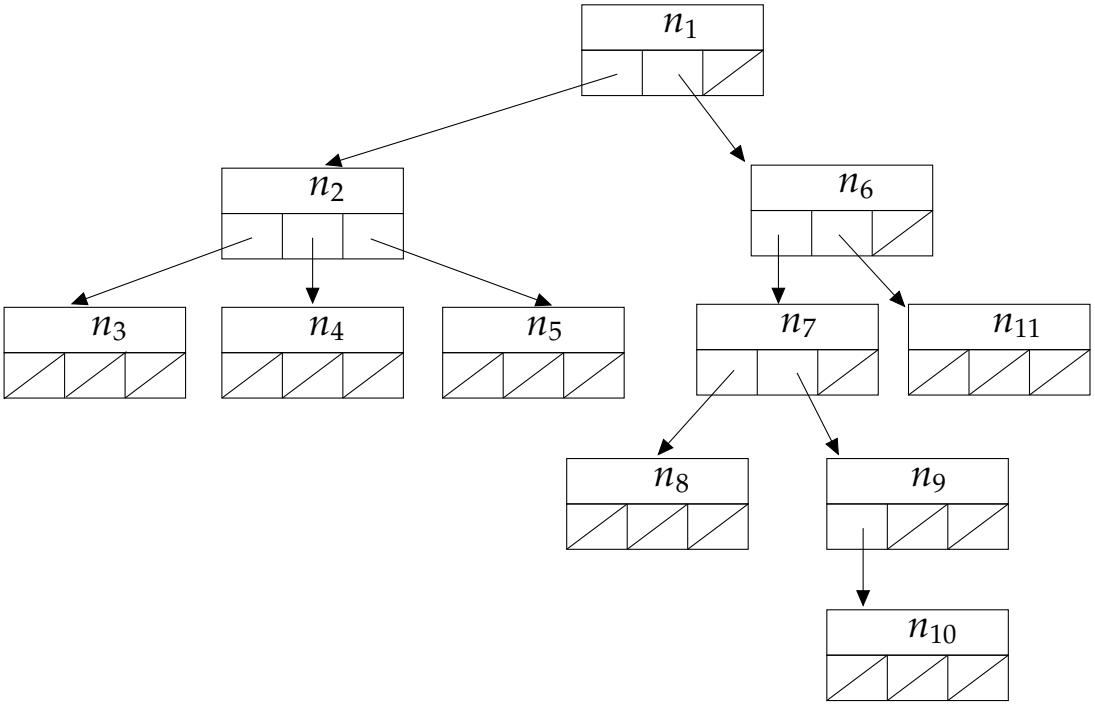
   Er zijn dus slechts 10 referenties met een waarde verschillend van `null`.

   De verhouding van het aantal gebruikte t.o.v. het totaal aantal referenties is dus
   $$\frac{10}{33} = 0.30303 \approx \frac{1}{3}.$$

2. De oplossing van deze opgave zie je in Figuur 4.2

3. Bij de eerste-kind-volgende-broer voorstelling zijn er steeds twee referenties per top. In totaal zijn er nu dus 22 referenties. Het aantal effectief gebruikte referenties is nog steeds 10. Inderdaad, als we de pijlen omkeren dan wijst elke top behalve de wortel ofwel naar zijn ouder ofwel naar zijn vorige broer.

   De verhouding van het aantal gebruikte t.o.v. het aantal voorziene referenties is nu
   $$\frac{10}{22} = 0.45455 \approx \frac{1}{2}.$$



Figuur 4.2: De array-van-kinderen voorstelling van de voorbeeldboom voor de oefeningen.

Dit is dus heel wat beter dan bij de array-van-kinderen voorstelling. Bovendien is deze verhouding *onafhankelijk* van de graad van de boom.

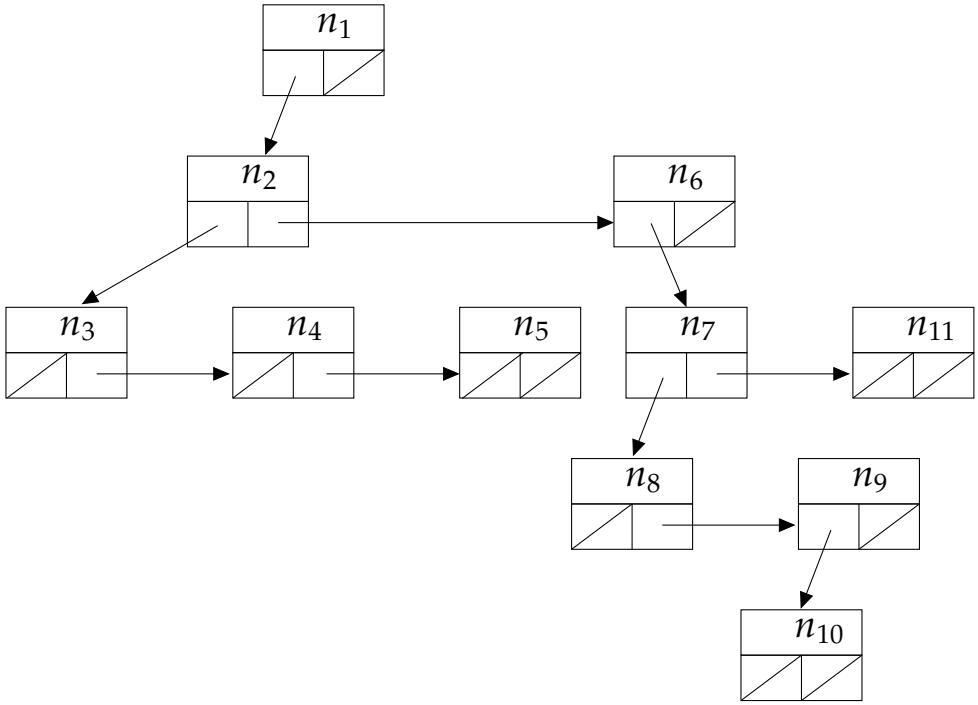
4. We kunnen de redenering uit de vorige oefening veralgemenen. Wanneer de boom n toppen heeft dan zijn er in de eerste-kind-volgende-broer voorstelling $2n$ referenties. Het aantal gebruikte referenties is steeds $n - 1$. Immers, wanneer we de pijlen omkeren dan wijst elke top, behalve de wortel, ofwel naar zijn vorige broer of naar zijn ouder. De gevraagde verhouding is dus

$$\frac{n-1}{2n} \approx \frac{1}{2} \quad \text{als } n \text{ groot is.}$$

5. De oplossing van deze oefening wordt getoond in Figuur 4.3.

4.3.3 Oefeningen

1. Geef de volgorde waarin de toppen worden bezocht wanneer de boom in Figuur 4.1 respectievelijk in preorde en postorde wordt doorlopen.



Figuur 4.3: De eerste-kind-volgende-broer voorstelling van de voorbeeldboom voor de oefeningen.

2. Geef code analoog aan Algoritme 4.1 om een gewortelde boom in postorde te doorlopen.

3. Geef code analoog aan Algoritme 4.2 om de hoogte van een gewortelde boom te berekenen. Baseer je op formule (4.2).

4.3.4 Oplossingen

1. Bij preorde bezoeken we eerst de wortel van de boom en daarna (recursief) in preorde de deelbomen onder de wortel. We vinden voor preorde

$$n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8, n_9, n_{10}, n_{11}.$$

Voor postorde vinden we

$$n_3, n_4, n_5, n_2, n_8, n_{10}, n_9, n_7, n_{11}, n_6, n_1.$$

2. De code voor het in postorde doorlopen van een gewortelde boom is heel gelijkaardig aan die voor het in preorde doorlopen van een gewortelde boom: behalve het wijzigen van de naam van de methodes

moeten we enkel de "visit" functie verplaatsen zodat deze wordt uitgevoerd *na* het uitvoeren van de recursieve oproepen. In concreto krijgen we dan:

Invoer Een gewortelde boom T, en een visit functie.
Uitvoer De visit functie is aangeroepen voor elke top van de boom.

```
1: function PostOrde(T,visit)
2:     PostOrdeRecursief(T.wortel, visit)          ▷ start met de wortel
3: end function
4: function PostOrdeRecursief(v, visit)
5:     for all w ∈ kinderen(v) do      ▷ implementatie-onafhankelijk
6:         PostOrdeRecursief(w, visit)
7:     end for
8:     visit(v)
9: end function
```

3. Het is relatief eenvoudig om de formule (4.2) om te zetten in een recursief algoritme.

Invoer Een gewortelde boom T.
Uitvoer De hoogte van de boom.

```
1: function Hoogte(T)
2:     return HoogteRecursief(T.wortel)          ▷ Start met de wortel
3: end function
4: function HoogteRecursief(v)
5:     h ← −1                          ▷ Dan kunnen we altijd +1 doen
6:     for all w ∈ kinderen(v) do
7:         h ← max(h, HoogteRecursief(w))
8:     end for
9:     return h + 1      ▷ Wanneer m = 0 dan zal h nog steeds −1 zijn
10: end function
```

4.4.4 Oefeningen

1. a) Teken de binaire boom met labels 0 t.e.m. 9 waarvoor de inorde sequentie

$$9, 3, 1, 0, 4, 2, 7, 6, 8, 5$$

is terwijl de postorde sequentie

$$9, 1, 4, 0, 3, 6, 7, 5, 8, 2$$

is.

b) Doe nu hetzelfde voor de volgende sequenties, of leg uit waarom zo'n binaire boom niet bestaat:

$$\text{inorde: } 9, 3, 1, 0, 4, 2, 7, 6, 8, 5$$

en

$$\text{postorde: } 9, 1, 4, 0, 3, 6, 5, 7, 8, 2$$

4.4.5 Oplossingen

1. a) Uit de postorde sequentie weten we onmiddellijk dat 2 de wortel van de boom is want bij de postorde sequentie staat de wortel van de boom steeds op de laatste plaats. Uit de inorde sequentie leiden we dan af dat de toppen in de linkerdeelboom

$$9, 3, 1, 0, 4$$

zijn, terwijl de toppen in de rechterdeelboom

$$7, 6, 8, 5$$

zijn, want bij de inorde sequentie staat de wortel steeds tussen de toppen van de linker- en rechterdeelboom. We zien dat in de postorde sequentie de verzameling toppen

$$\{9, 3, 1, 0, 4\}$$

wel degelijk voorkomt vóór al de toppen van de verzameling

$$\{7, 6, 8, 5\}.$$

Op dit moment zijn de twee sequenties nog steeds consistent met elkaar.

We kunnen dit proces nu verder zetten voor de twee geïdentificeerde deelsequenties, bv.

$$\text{inorde: } 9, 3, 1, 0, 4 \quad \text{postorde: } 9, 1, 4, 0, 3$$

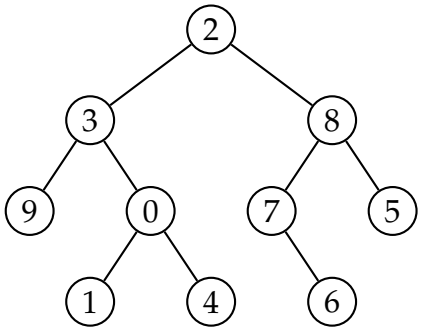
en

$$\text{inorde: } 7, 6, 8, 5 \quad \text{postorde: } 6, 7, 5, 8.$$

Uit de eerste postorde sequentie zien we dat 3 de wortel moet zijn van de linkerdeelboom, en uit de bijhorende inorde sequentie volgt dan dat 9 de enige top is in de linkerdeelboom van de top 3. De toppen in de rechterdeelboom van 3 zijn dus 1, 0 en 4. Aangezien 0 als laatste voorkomt in de postorde sequentie is dit de wortel van deze deelboom. Uit de inorde sequentie zien we dan dat 1 onmiddellijk links moet zitten van 0 en 4 onmiddellijk rechts.

In de rechterdeelboom bestaande uit de toppen $\{7, 6, 8, 5\}$ is 8 de wortel. In de rechterdeelboom van 8 zit enkel de top 5. In de linkerdeelboom de toppen 7 en 6; dit zien we allebei uit de inorde sequentie. We zien dat 7 de wortel moet zijn van de linkerdeelboom van 8. Uit de inorde sequentie volgt dat 6 het rechterkind is van 7.

De volledige binaire boom ziet er als volgt uit:

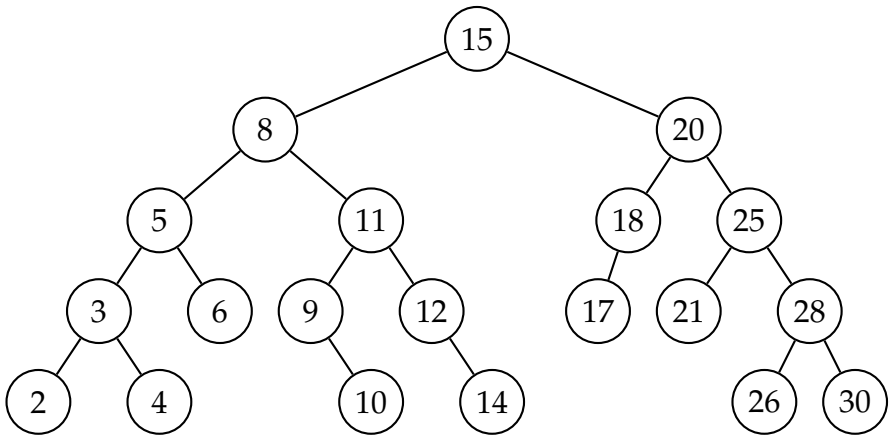


Men gaat gemakkelijk na dat deze boom inderdaad de vooropgestelde inorde en postorde sequentie van toppen heeft. De redenering die we hebben gemaakt toont ook aan dat dit de *enige* binaire boom die consistent is met de gegeven sequenties.

b) Net als in de vorige opgave zien we dat 2 de wortel moet zijn van de binaire boom en dat $9, 3, 1, 0, 4$ resp. $7, 6, 8, 5$ de toppen zijn van de linker- resp. rechterdeelboom.

In de postorde sequentie komen alle toppen van de linkerdeelboom ook effectief vóór de toppen van de rechterdeelboom.

De linkerdeelboom is net dezelfde als in de vorige oefening, dus we concentreren ons nu op de rechterdeelboom met de toppen $7, 6, 8, 5$. Uit de postorde sequentie zien we dat 8 de wortel moet zijn van deze boom en uit de inorde sequentie volgt dat 7 en 6 de linkerdeelboom uitmaken, terwijl de enige top in de rechterdeel-



Figuur 4.4: Een binaire zoekboom.

boom van 8 de top 5 is. Echter, in de postorde sequentie komt 5 (uit de rechterdeelboom) vóór 7 (uit de linkerdeelboom). Dit is natuurlijk onmogelijk.

Er is m.a.w. geen enkele binaire boom die de gegeven inorde en postorde sequentie heeft.

4.5.5 Oefeningen

1. Geef de binaire zoekboom die opgebouwd wordt door de volgende sleutels

$$4, 7, 5, 8, 11, 3, 2, 9, 10, 6$$

één voor één aan de zoekboom toe te voegen in de gegeven volgorde.

2. Veronderstel dat men een binaire zoekboom opbouwt door sleutels één voor één toe te voegen aan een initieel lege boom.

   a) Geef een rij van lengte 7 die een binaire zoekboom van minimale diepte oplevert.

   b) Geef een rij van lengte 15 die een binaire zoekboom van minimale diepte oplevert.

   c) Geef een rij van lengte 5 die een binaire zoekboom van maximale diepte oplevert. Wanneer krijg je een over het algemeen een slecht gebalanceerde binaire zoekboom?

3. Beschouw de binaire zoekboom in Figuur 4.4. Voer de volgende opdrachten één na één uit.

   a) Welke toppen worden er bezocht bij het zoeken naar de top 12?

   b) Welke toppen worden er bezocht bij het zoeken naar de top 27?

   c) Voeg een top met sleutelwaarde 23 toe aan de zoekboom. Teken de resulterende zoekboom.

   d) Voeg vervolgens een top met sleutelwaarde 22 toe aan de zoekboom. Teken de resulterende zoekboom.

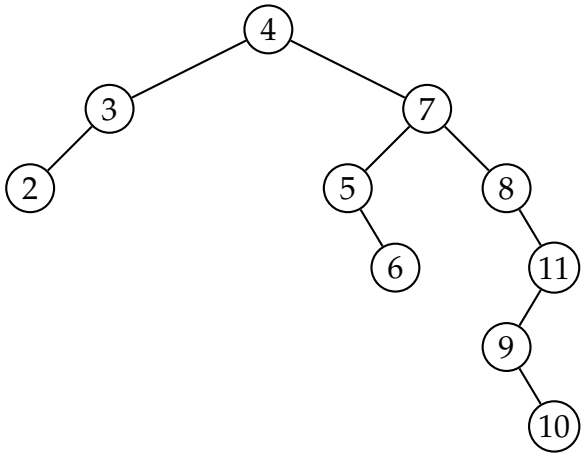
   e) Verwijder de top met waarde 4 uit de zoekboom. Teken de resulterende zoekboom.

   f) Verwijder vervolgens de top met waarde 18 uit de zoekboom. Teken de resulterende zoekboom.

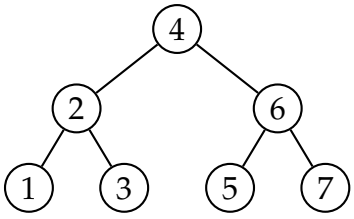
   g) Verwijder vervolgens de top met waarde 20 uit de zoekboom. Teken de resulterende zoekboom.

4.5.6 Oplossingen

1. Hier krijgen we volgende binaire zoekboom.



2. a) De "ideale" zoekboom met 7 elementen ziet er zo uit:



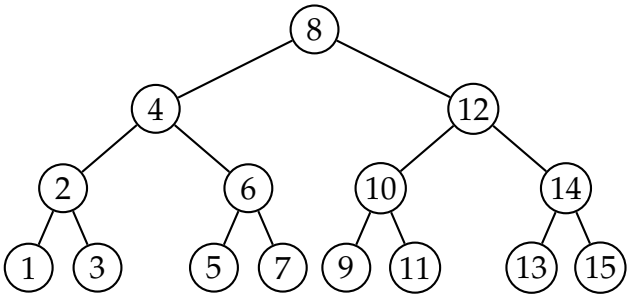
Een *mogelijke* volgorde van toevoegen is de volgende:

$$4, 2, 6, 1, 3, 5, 7.$$

Er zijn echter nog veel andere volgordes mogelijk, bv.

$$4, 2, 1, 3, 6, 5, 7.$$

b) De zoekboom met minimale diepte bestaande uit 15 elementen heeft de volgende vorm:



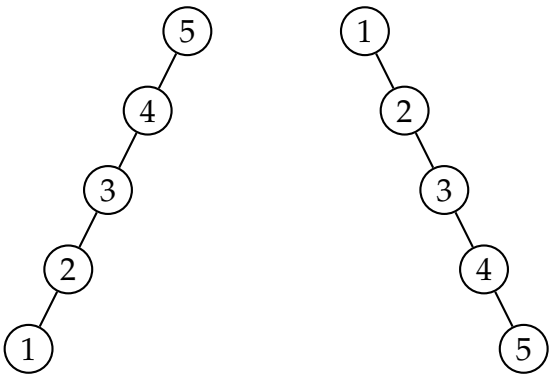
Merk op dat de linkerdeelboom van de wortel dezelfde is als in de vorige oefening. De sleutels van de rechterdeelboom zijn exact 8 groter dan de overeenkomstige labels in de linkerdeelboom. Een *mogelijke* volgorde van invoegen is:

$$8, 4, 12, 2, 6, 10, 14, 1, 3, 5, 7, 9, 11, 13, 15$$

Dit is niveau per niveau. De pre-orde volgorde is ook een volgorde die zal leiden tot dezelfde boom:

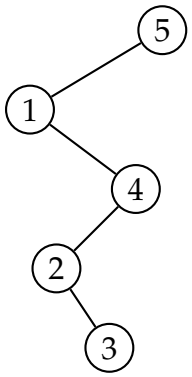
$$8, 4, 2, 1, 3, 6, 5, 7, 12, 10, 9, 11, 14, 13, 15.$$

c) Wanneer je de sleutels toevoegt in dalende of stijgende volgorde dan krijg je de volgende zoekbomen met maximale diepte:



Stijgende en dalende volgorde van labels zijn echter niet de enige manieren om een slecht gebalanceerde boom te vinden. Hieronder nog een ander voorbeeld waarbij de labels werden toegevoegd in de volgorde

$$5, 1, 4, 2, 3$$



3. a) De volgende toppen worden bezocht wanneer we zoeken naar de top met label 12:

   15 (links) 8 (rechts) 11 (rechts) 12 (gevonden)

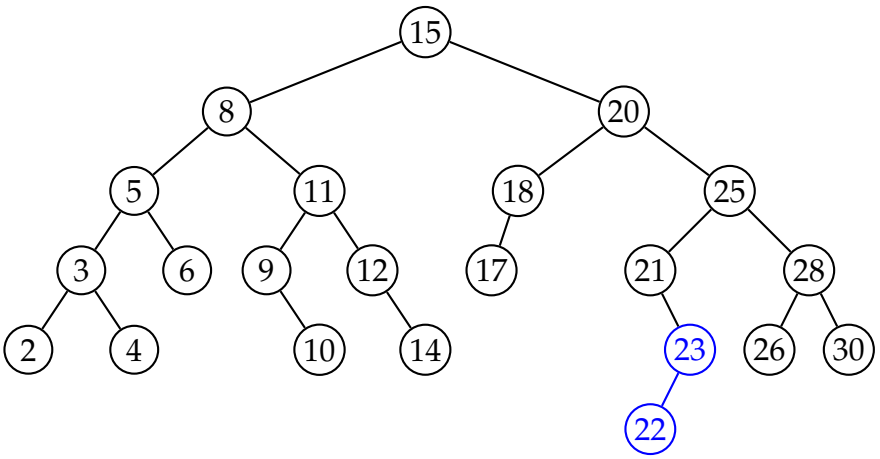
   b) De volgende toppen worden bezocht wanneer we zoeken naar de top met label 27:

   15 (rechts) 20 (rechts) 25 (rechts) 28 (links) 26 (rechts is leeg).

   In de top 26 zouden we m.a.w. naar rechts moeten gaan in onze zoektocht naar de top 27. Deze rechterdeelboom is echter leeg. Dit toont aan dat de top 27 niet tot de boom behoort.

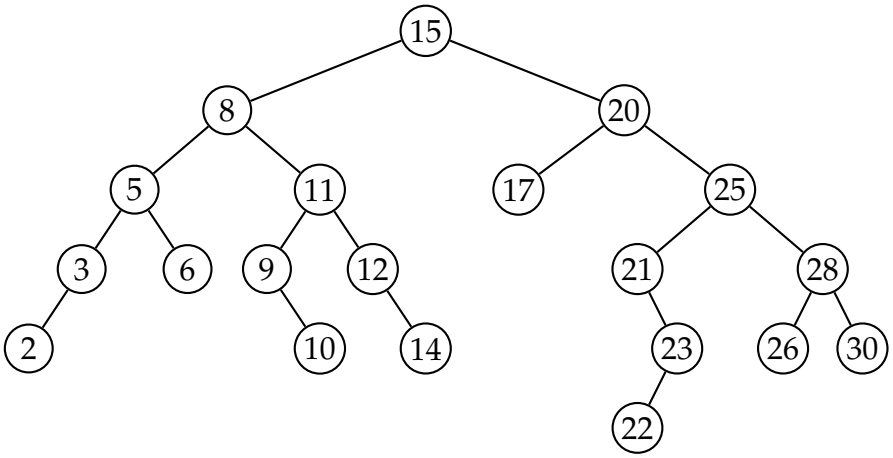
   c) Toevoegen van top 23 resulteert in een nieuw blad rechts van top 21.

   d) Toevoegen van top 22 resulteert in een nieuw blad links van de nieuw toegevoegde top 23. Het resultaat van deze twee toevoegingen wordt hieronder getoond.

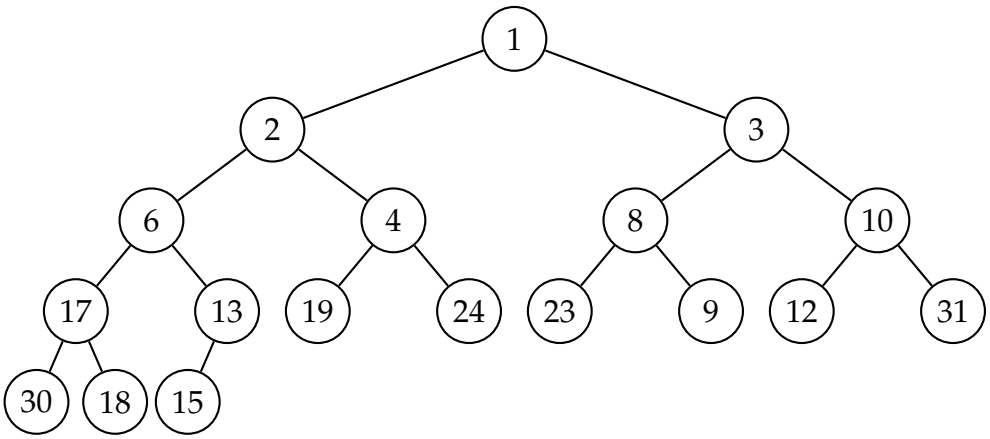


e) Verwijderen van top 4 uit de binaire zoekboom is zeer eenvoudig aangezien 4 een blad is. Dit blad hoeft enkel maar verwijderd te worden (door de rechterreferentie van 3 terug op "null" te zetten).

f) Verwijderen van top 18 is ook relatief eenvoudig aangezien de top 18 slechts één kind heeft; 18 is het linkerkind van zijn ouder (20) en dus wordt het enige kind van 18 (17 in dit geval) het linkerkind van 20. Het resultaat van de laatste twee verwijderbewerkingen wordt hieronder getoond:

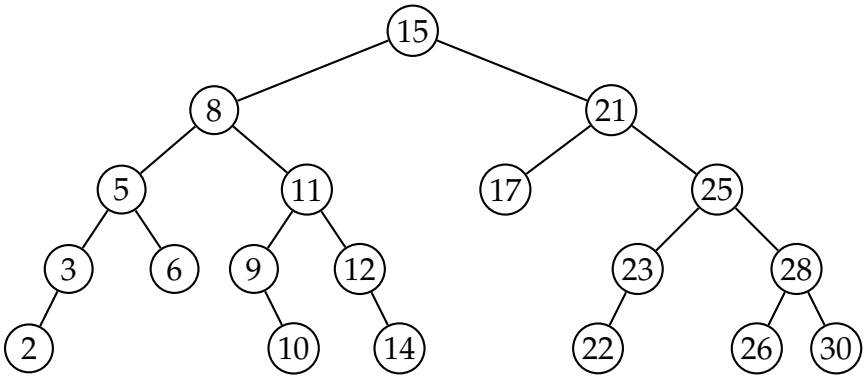


g) De top 20 heeft twee kinderen. We zoeken zijn opvolger, dit is de kleinste top in zijn rechterdeelboom. In dit geval is dit top 21. We verwijderen de top 21 uit de boom. Uiteraard heeft 21 hoogstens één kind en dus is dit gemakkelijk. De top 21 is het linkerkind van zijn ouder 25; de deelboom met wortel 23 wordt de linkerdeelboom van 25. Vervolgens vervangen we (de waarde)



Figuur 4.5: Een binaire hoop.

20 door (de waarde van) zijn opvolger 21. Het resultaat zie je hieronder.



4.6.8 Oefeningen

1. Start met een lege binaire hoop. Voeg achtereenvolgens de volgende elementen toe aan de binaire hoop:

   $$11, 13, 1, 15, 6, 5, 9, 16, 3, 10, 7, 4, 12, 14, 2.$$

   Teken de resulterende hoop na elke toevoeging.

2. Beschouw de binaire hoop in Figuur 4.5. Verwijder de drie kleinste elementen uit deze binaire hoop. Teken de binaire hoop na elke verwijdering.

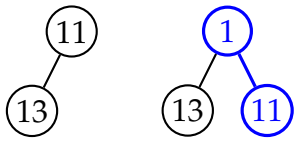
3. Wat worden de relaties in Eigenschap 4.36 wanneer een binaire hoop wordt opgeslaan in een array met als eerste index 0?

4. Waar kan het maximale element zich bevinden in een binaire hoop, aannemende dat de binaire hoop bestaat uit verschillende elementen.

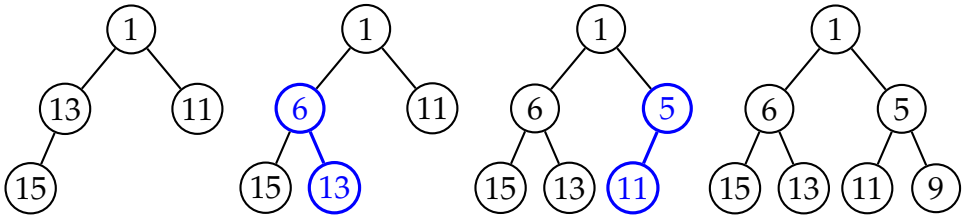
4.6.9 Oplossingen

1. We starten met een binaire hoop bestaande uit slechts één top:

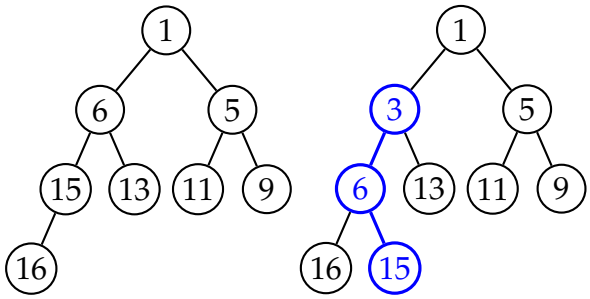
Bij het toevoegen van 13 is er geen enkele verwisseling nodig; bij het toevoegen van 1 is er één verwisseling nodig.



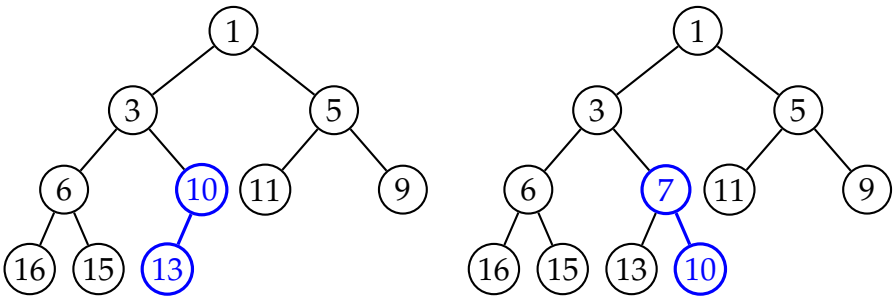
Voor het toevoegen van 15 is er geen enkele verwisseling nodig; voor het toevoegen van 6 en 5 zijn er telkens 1 verwisseling nodig, terwijl voor het toevoegen van 9 er geen enkele verwisseling nodig is.



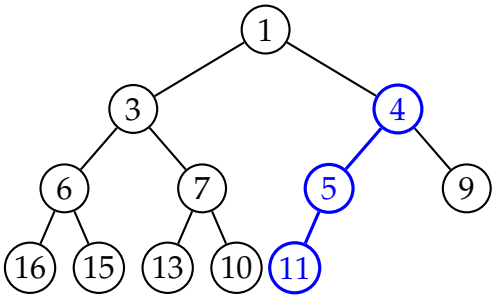
Voor het toevoegen van 16 is er geen enkele verwisseling nodig; voor het toevoegen van 3 zijn er twee verwisselingen nodig.



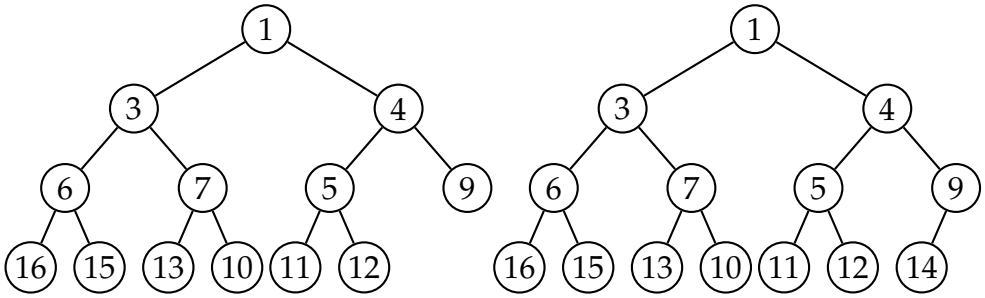
Voor het toevoegen van 10 en 7 is er telkens één verwisseling nodig:



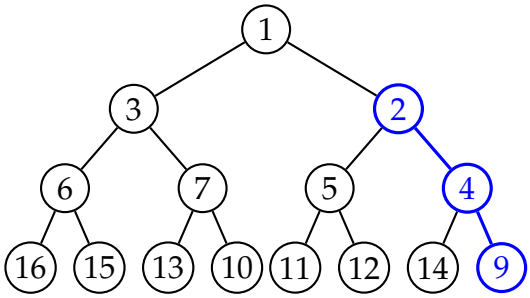
Voor het toevoegen van 4 zijn er twee verwisselingen nodig:



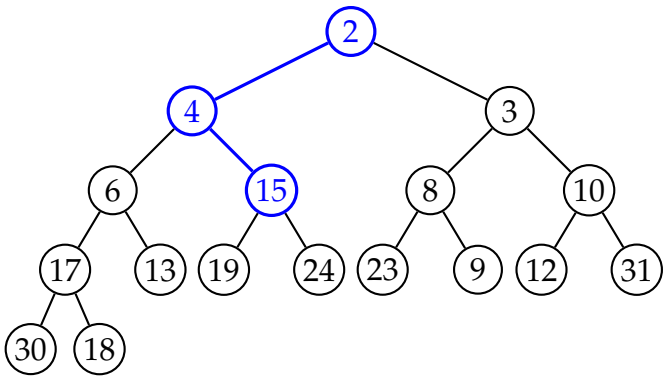
Voor het toevoegen van 12 en 14 zijn er geen verwisselingen nodig:



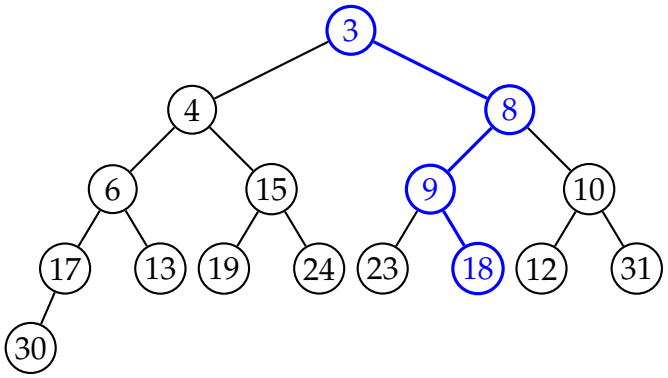
Voor het toevoegen van 2, tenslotte, zijn er twee verwisselingen nodig:



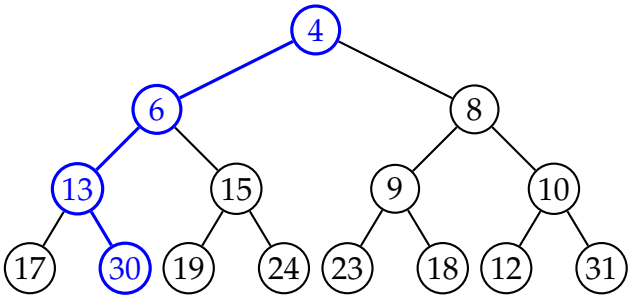
2. Het kleinste element 1 bevindt zich in de wortel van de boom. We verwijderen het laatste blad (met waarde 15) en plaatsen 15 in de wortel van de boom. Nu moet 15 eventueel naar beneden bubbelen teneinde de ordeningseigenschap van binaire hopen te herstellen. We krijgen:



Nu is 2 het kleinste element en wordt dit vervangen door de waarde 18 in het laatste blad. De waarde 18 bubbelt dan omlaag zodat de ordeningseigenschap van binaire hopen weer geldig is:



We herhalen deze procedure nu nogmaals voor het kleinste element 3 en de waarde 30 in het laatste blad.



3. Als de toppen genummerd worden vanaf nul dan zien we voor de eerste toppen dat

top	kinderen
0	1 en 2
1	3 en 4
2	5 en 6
3	7 en 8
⋮	⋮

   Hieruit blijkt duidelijk dat

$$\text{left}(i) = 2i + 1, \quad \text{right}(i) = 2i + 2 \quad \text{en} \quad \text{parent}(i) = \lfloor (i-1)/2 \rfloor.$$

4. In een binaire hoop waar alle elementen verschillend zijn is de waarde van een top steeds *strikt kleiner* dan de waarde van zijn kinderen. Dit betekent dat het maximale element zich steeds in een *blad* moet bevinden.

   Merk op dat dit niet echt veel helpt om snel dit maximum te lokaliseren. Als er n toppen zijn in de binaire hoop dan zijn er $\lceil n/2 \rceil$ blaadjes. Het vinden van het maximum in een binaire hoop is dus een bewerking met een *lineaire* tijdscomplexiteit, terwijl het vinden van het minimum een *constante* tijdscomplexiteit heeft.

HOOFDSTUK 5

Graafalgoritmes

5.1.1 Oefeningen

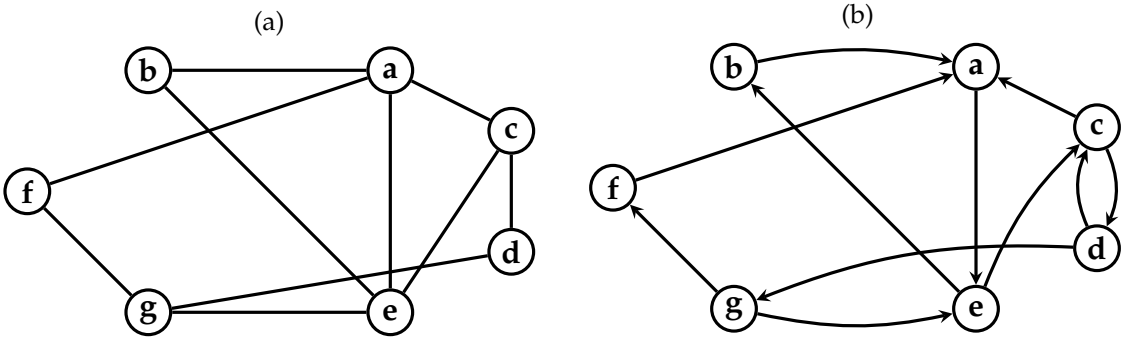
1. Beschouw de gerichte en ongerichte graaf in Figuur 5.1.

   a) Geef de bogenverzameling van deze twee grafen.

   b) Geef voor beide grafen de verzameling buren(e). Wat is de graad van de knoop e in beide gevallen?

   c) Vind het kortste pad (i.e. het pad met de kleinste lengte) van b naar d in beide grafen.

   d) Vind in beide grafen de langste enkelvoudige cykel die d bevat.



Figuur 5.1: Twee voorbeeldgrafen voor de oefeningen.

5.1.2 Oplossingen

1. a) De bogenverzameling E_o van de ongerichte graaf is

$$E_o = \{(a,b),(a,c),(a,e),(a,f),(b,e),(c,d),(c,e),(d,g),(e,g),(f,g)\}.$$

Opmerking: omdat het hier gaat om een ongerichte graaf mag de volgorde van de "koppels" omgekeerd worden.
Voor de gerichte graaf vinden we de bogenverzameling E_g:

$$E_g = \{(a,e),(b,a),(c,a),(c,d),(d,c),(d,g),(e,b),(e,c),(f,a),(g,e),(g,f)\}.$$

Elke boog (v,w) heeft hier een staart v en een kop w; de volgorde van de koppels is in dit geval dus wel van belang.

b) In de ongerichte graaf zijn de buren van de knoop e:

$$\text{buren}(e) = \{a,b,c,g\},$$

en bijgevolg is de graad van de knoop e gelijk aan vier.
In de gerichte graaf vinden we:

$$\text{buren}(e) = \{b,c\}.$$

In dit geval is de graad van e dus slechts twee. Er zijn slechts twee bogen waarvoor de *staart* gelijk is aan e.

c) In de ongerichte graaf zien we (op het zicht) dat er minstens drie stappen (bogen) nodig zijn om vanuit b de knoop d te bereiken. De mogelijke paden zijn:

$$b,a,c,d$$

en

$$b,e,c,d$$

en

$$b,e,g,d.$$

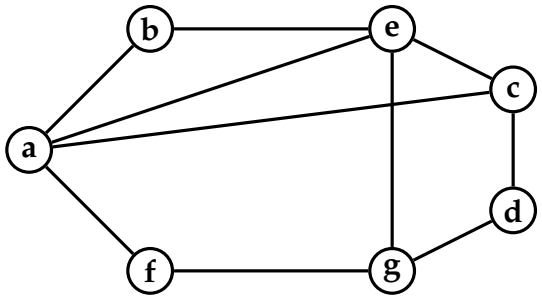
Voor de gerichte graaf zijn er minimaal vier stappen (bogen) nodig. In dit geval is er slechts één kortste pad en dit wordt gegeven door

$$b,a,e,c,d.$$

Later in de cursus komen systematische manieren aan bod om (één) kortste pad te vinden.

d) We moeten een zo lang mogelijk pad vinden waarbij elke knoop (behalve d) hoogstens één keer voorkomt en zodanig dat alle bogen in het pad verschillend zijn.

Hieronder zie je een *andere voorstelling* van *dezelfde* ongerichte graaf:



Het is nu onmiddellijk duidelijk dat de gevraagde cykel gegeven wordt door:

$$d, c, e, b, a, f, g, d$$

In dit geval heeft de graaf dus een zogenaamde *Hamiltoniaanse cykel*, een enkelvoudige cykel die elke knoop juist éénmaal bezoekt.

Voor de gerichte graaf zie je dat een cykel die eindigt in d steeds c als voorlaatste knoop moet hebben, en bijgevolg dus (als we niet onmiddellijk teruggaan naar d) de knoop e als voorganger van c moet hebben. Voor knoop e zijn de mogelijke voorgangers g en a. Als we g kiezen dan zijn we onmiddellijk terug in d (pad d, g, e, c, d); wanneer we echter a kiezen dan kunnen we nog naar f, dan g en tenslotte d (steeds in omgekeerde volgorde). De langste enkelvoudige cykel die d bevat is dus

$$d, g, f, a, e, c, d.$$

5.2.3 Oefeningen

1. Beschouw de gerichte en ongerichte graaf in Figuur 5.1.

   a) Geef voor beide grafen de adjacentiematrix. Je mag veronderstellen dat a rangnummer 1 heeft, b rangnummer 2 enzovoort.

b) Geef voor beide grafen de adjacentielijst-voorstelling. Je mag veronderstellen dat a rangnummer 1 heeft, b rangnummer 2 enzovoort.

2. Hoe berekent men de graad van een top i van een graaf G wanneer de adjacentiematrix A van de graaf gegeven is? Geef een algoritme. Wat is de tijdscomplexiteit van deze methode?

3. Hoe berekent men de graad van een top i van een graaf G wanneer de adjacentielijst-voorstelling van de graaf gegeven is? Geef een algoritme. Wat is de tijdscomplexiteit van deze methode?

5.2.4 Oplossingen

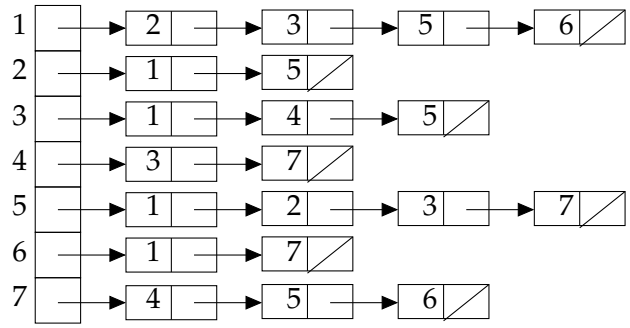
1. a) Voor de ongerichte graaf is de adjacentiematrix

$$A_o = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}.$$

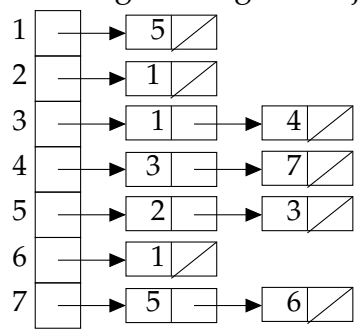
Voor de gerichte graaf vinden we:

$$A_g = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}.$$

b) Voor de ongerichte graaf vinden we de volgende adjacentielijst-voorstelling.



Voor de gerichte graaf krijgen we de volgende adjacentielijst-voorstelling:



2. We berekenen de graad van een knoop i door rij i van de adjacentiematrix te overlopen en te tellen hoeveel maal men een "1" aantreft.

   Het is duidelijk dat deze methode een tijdscomplexiteit heeft die lineair is in het aantal knopen van de graaf, of $\Theta\big(\#(V)\big)$.

   Het algoritme zou er als volgt kunnen uitzien:

   Invoer Een gerichte of ongerichte graaf $G = (V, E)$ met orde $n > 0$. Een knoop i waarvan de graad moet berekend worden. De adjacentiematrix van de graaf wordt gegeven in het veld met de naam "adjacentiematrix".

   Uitvoer De graad van de knoop i werd berekend.

   ```
   1: function GraadAdjacentiematrix(G, i)
   2:     g ← 0                                  ▷ Initialiseer de graad op 0
   3:     A ← G.adjacentiematrix
   4:     for j ← 1 to n do
   5:         g ← g + A[i][j]                    ▷ Overloop rij i
   6:     end for
   7:     return g
   8: end function
   ```

3. Wanneer de adjacentielijstvoorstelling van de graaf gegeven is, dan moeten we de *lineair geschakelde lijst* op positie i gaan overlopen om te

tellen hoeveel elementen die bevat.

Dit vereist evenveel stappen als de graad van de beschouwde knoop en dus is de tijdscomplexiteit $\Theta(\text{graad}(i))$. De uitvoeringstijd hangt dus enkel af van de graad van de knoop en *niet* van het totaal aantal knopen.

In pseudocode zou dit algoritme er als volgt kunnen uitzien:

Invoer Een gerichte of ongerichte graaf $G = (V, E)$ met orde $n > 0$. Een knoop i waarvan de graad moet berekend worden. De adjacentielijst van de graaf wordt gegeven in het veld met de naam "adjacentielijst".

Uitvoer De graad van de knoop i werd berekend.

```
1:  function GRAADADJACENTIELIJST(G, i)
2:      g ← 0                              ▷ Initialiseer de graad op 0
3:      L ← G.adjacentielijst              ▷ L is een array van referenties
4:      p ← L[i]                           ▷ De eerste referentie voor knoop i
5:      while p ≠ null do
6:          g ← g + 1
7:          p ← p.next                     ▷ De volgende referentie opzoeken
8:      end while
9:      return g
10: end function
```

5.3.5 Oefeningen

1. Een ongerichte graaf is GECONNECTEERD als en slechts als er een pad bestaat tussen elke twee knopen v en w.

   a) Ga na dat de bovenstaande definitie equivalent is met zeggen dat er een pad bestaat van een bepaalde knoop s naar alle andere knopen.

   b) Schrijf een methode ISGECONNECTEERD die nagaat of een ongerichte graaf geconnecteerd is (return-waarde **true**) of niet (return-waarde **false**). Doe dit door de methode BREEDTEEERST aan te passen.

2. Vind alle mogelijke topologische sorteringen van de graaf in Figuur 5.10.

3. Vind de compilatievolgorde van de modules in Figuur 5.9 wanneer de labels in dalende volgorde worden doorlopen.

4. Veronderstel nu dat er in de graaf van Figuur 5.9 een extra boog $(8, 6)$ wordt toegevoegd. Pas nu het algoritme voor topologisch sorteren toe.

5.3.6 Oplossingen

1. a) We moeten bewijzen dat de uitspraken

   > er bestaat een pad tussen elke twee knopen van de ongerichte graaf G

   en

   > er bestaat een pad van één knoop s naar alle andere knopen van de ongerichte graaf G

   equivalent zijn.

   Het is onmiddellijk duidelijk dat de eerste uitspraak de tweede impliceert.

   Omgekeerd, veronderstel dat de tweede uitspraak waar is. We moeten nu aantonen dat er een pad bestaat tussen twee willekeurige knopen v en w van de ongerichte graaf G. We weten bij veronderstelling dat er een pad bestaat van s naar v:

   $$s \rightsquigarrow v$$

   Omdat de graaf *ongericht* is kunnen we dit pad ook "omkeren" en bestaat er dus ook een pad van v naar s. Merk op: deze redenering is niet geldig voor een gerichte graaf.

   $$v \rightsquigarrow s. \tag{5.1}$$

   Bovendien bestaat er bij veronderstelling ook een pad van s naar w:

   $$s \rightsquigarrow w. \tag{5.2}$$

   Als we de paden in (5.1) en (5.2) aan elkaar "plakken" dan vinden we een pad van v (via s) naar w:

   $$v \rightsquigarrow s \rightsquigarrow w.$$

   Dit toont aan dat beide uitspraken equivalent zijn.

b) De methode BREEDTEEERST bepaalt welke knopen bereikt kunnen worden vanuit een bepaalde startknoop s. Om de methode ISGECONNECTEERD te schrijven moeten we enkel bijhouden hoeveel knopen kunnen bereikt worden vanuit s en controleren of dit aantal gelijk is aan het totaal aantal knopen n. Indien dit zo is, dan is de ongerichte graaf geconnecteerd, anders niet.

In de implementatie kiezen we als startknoop s de knoop 1.

Invoer Een ongerichte graaf $G = (V, E)$ met orde $n > 0$. De knopen zijn genummerd van 1 tot n, i.e. $V = \{1, 2, \ldots, n\}$.

Uitvoer **true** als de graaf geconnecteerd is, **false** anders.

```
1:  function IsGeconnecteerd(G)
2:      D ← [false, false, ..., false]
3:      D[1] ← true                          ▷ Kies 1 als startknoop
4:      a ← 1                  ▷ a telt het aantal bereikbare knopen
5:      Q.init()
6:      Q.enqueue(1)                ▷ Plaats knoop 1 op de wachtrij
7:      while Q ≠ ∅ do
8:          v ← Q.dequeue()
9:          for all w ∈ buren(v) do
10:             if D[w] = false then
11:                 D[w] ← true
12:                 a ← a + 1 ▷ nieuw ontdekte knoop: a verhogen
13:                 Q.enqueue(w)
14:             end if
15:         end for
16:     end while
17:     return a = n                  ▷ Vergelijking, geen toewijzing
18: end function
```

We introduceerden dus een nieuwe variabele a met de eigenschap dat a steeds telt hoeveel keer er **true** staat in de array D.

2. Het is duidelijk dat d steeds de laatste knoop moet zijn in een topologische sortering aangezien dit de enige knoop is met graad 0. Bovendien moet a steeds de eerste knoop zijn in de topologische sortering want alle andere knopen zijn bereikbaar vanuit a. De knopen b en c zijn niet gerelateerd en kunnen dus in een willekeurige volgorde staan. De twee topologische sorteringen zijn m.a.w.

$$a, b, c, d \quad \text{en} \quad a, c, b, d.$$

3. Voor deze oefening nemen we aan dat op lijn 4 van het algoritme van topologisch sorteren de knopen overlopen worden in de volgorde $10, 9, 8, \ldots, 1$. Bovendien nemen we aan dat op lijn 17 de buren ook steeds in dalende volgorde worden doorlopen zodat bv. voor knoop 7 de buren gegeven worden als 9 en 5 (in die volgorde).

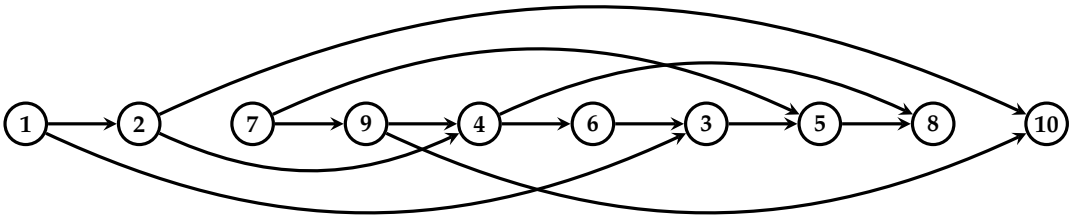
   We houden (de inhoud van) de array D bij, de stapel van de methode-oproepen en de lijst S.

	1	2	3	4	5	6	7	8	9	10	stack	S
(a)	0	0	0	0	0	0	0	0	0	0		$\emptyset$
(b)	0	0	0	0	0	0	0	0	0	1	10	$\emptyset$
(c)	0	0	0	0	0	0	0	0	0	2	10	10
(d)	0	0	0	0	0	0	0	0	1	2	9	10
(e)	0	0	0	1	0	0	0	0	1	2	9, 4	10
(f)	0	0	0	1	0	0	0	1	1	2	9, 4, 8	10
(g)	0	0	0	1	0	0	0	2	1	2	9, 4	8, 10
(h)	0	0	0	1	0	1	0	2	1	2	9, 4, 6	8, 10
(i)	0	0	1	1	0	1	0	2	1	2	9, 4, 6, 3	8, 10
(j)	0	0	1	1	1	1	0	2	1	2	9, 4, 6, 3, 5	8, 10
(k)	0	0	1	1	2	1	0	2	1	2	9, 4, 6, 3	5, 8, 10
(l)	0	0	2	1	2	1	0	2	1	2	9, 4, 6	3, 5, 8, 10
(m)	0	0	2	1	2	2	0	2	1	2	9, 4	6, 3, 5, 8, 10
(n)	0	0	2	2	2	2	0	2	1	2	9	4, 6, 3, 5, 8, 10
(o)	0	0	2	2	2	2	0	2	2	2		9, 4, 6, 3, 5, 8, 10
(p)	0	0	2	2	2	2	1	2	2	2	7	9, 4, 6, 3, 5, 8, 10
(q)	0	0	2	2	2	2	2	2	2	2		7, 9, 4, 6, 3, 5, 8, 10
(r)	0	1	2	2	2	2	2	2	2	2	2	7, 9, 4, 6, 3, 5, 8, 10
(s)	0	2	2	2	2	2	2	2	2	2		2, 7, 9, 4, 6, 3, 5, 8, 10
(t)	1	2	2	2	2	2	2	2	2	2	1	2, 7, 9, 4, 6, 3, 5, 8, 10
(u)	2	2	2	2	2	2	2	2	2	2		1, 2, 7, 9, 4, 6, 3, 5, 8, 10

   In de lijst S lezen we de gevonden topologische sortering af:

$$1, 2, 7, 9, 4, 6, 3, 5, 8, 10.$$

   In Figuur 5.2 zie je nog maar eens een andere voorstelling van de graaf van de softwaremodules. Deze keer volgens de nieuw gevonden topologische sortering. Je ziet opnieuw dat alle bogen vooruit wijzen.



Figuur 5.2: Een tweede topologische sortering van de graaf van de softwaremodules.

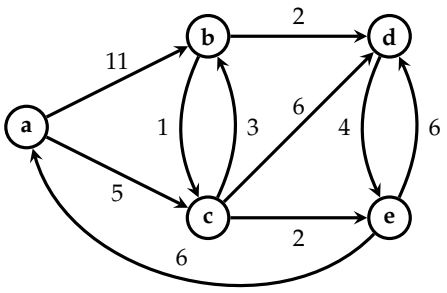
4. Wanneer een boog $(8, 6)$ wordt toegevoegd aan de graaf, dan heeft de graaf een cykel. We kijken of het algoritme de cykel detecteert.

	1	2	3	4	5	6	7	8	9	10	stack	S
(a)	0	0	0	0	0	0	0	0	0	0		$\emptyset$
(b)	1	0	0	0	0	0	0	0	0	0	1	$\emptyset$
(c)	1	1	0	0	0	0	0	0	0	0	1, 2	$\emptyset$
(d)	1	1	0	1	0	0	0	0	0	0	1, 2, 4	$\emptyset$
(e)	1	1	0	1	0	1	0	0	0	0	1, 2, 4, 6	$\emptyset$
(f)	1	1	1	1	0	1	0	0	0	0	1, 2, 4, 6, 3	$\emptyset$
(g)	1	1	1	1	1	1	0	0	0	0	1, 2, 4, 6, 3, 5	$\emptyset$
(h)	1	1	1	1	1	1	0	1	0	0	1, 2, 4, 6, 3, 5, 8	$\emptyset$

Op dit moment zal het algoritme de knoop 6 zal bezoeken als buur van de knoop 8. Echter, op dit moment staat de $D[6]$ reeds op "1" en is dus "bezig", en het algoritme heeft hier de cykel $6, 3, 5, 8, 6$ ontdekt. Op dit moment krijgt de variabele cycleDetected de waarde **true**. Hierdoor zullen geen verdere recursieve oproepen gebeuren want de conditie op lijn 18 evalueert nu steeds naar **false**. In het hoofdalgoritme wordt op lijn 8 waargenomen dat cycleDetected de waarde **true** heeft. Hierdoor eindigt het algoritme SORTEERTOPOLOGISCH met de waarde **false** om aan te geven dat er geen topologische sortering bestaat

5.4.3 Oefeningen

1. In Algoritme 5.5 wordt nu enkel de afstand van elke knoop v tot de startknoop s bijgehouden. In veel toepassingen heeft men echter ook



Figuur 5.3: Een gewogen, gerichte graaf.

een pad nodig dat deze minimale afstand realiseert.

a) Pas de pseudo-code van Algoritme 5.5 aan zodanig dat er een tweede array P wordt teruggeven zodanig dat $P[v]$ de knoop geeft die de voorganger (predecessor) is van v op een kortste pad van s naar v.

b) Pas je aangepaste algoritme toe op de gerichte graaf in Figuur 5.9 startend vanaf knoop 1. Ga ervan uit dat knopen steeds worden bezocht in stijgende volgorde. Hoe zit de array P er na afloop uit?

c) Schrijf een algoritme dat als invoer de array P neemt en een knoop v. Het algoritme geeft een lijst terug die het kortste pad van s naar v bevat (in de juiste volgorde).

2. Beschrijf hoe je volgend probleem kan oplossen als een kortste pad probleem. Gegeven een lijst van Engelstalige 5-letterwoorden. Woorden worden *getransformeerd* door juist één letter van het woord te vervangen door een andere letter. Geef een algoritme dat nagaat of een woord w_1 omgezet kan worden in een woord w_2. Indien dit het geval is dan moet je algoritme ook de tussenliggende woorden tonen voor de kortste sequentie van transformaties die w_1 in w_2 omzet.

3. Vind voor de graaf in Figuur 5.4 de lengte van het kortste pad van Brugge naar alle andere steden. Voer hiertoe het algoritme van Dijkstra uit.

4. Vind voor de graaf in Figuur 5.3 (de lengte van) het kortste pad van de knoop a naar alle andere knopen. Voer hiertoe het algoritme van Dijkstra uit (en houd ook bij wat de kortste paden zijn).

5.4.4 Oplossingen

1. a) Het idee is dat we starten met een array P die op een default-waarde is geïnitialiseerd die aangeeft dat er nog geen voorganger is gevonden. Wij gebruiken als deze default-waarde de waarde 0 omdat wij aannemen dat onze knopen genummerd zijn van 1 t.e.m. n en dus is 0 geen geldig knoopnummer.

   Telkens wanneer we een nieuwe knoop ontdekken vullen we de gepaste entry van de array P in.

   Invoer Een gerichte of ongerichte ongewogen graaf $G = (V, E)$ met orde $n > 0$. Een knoop s waarvan het zoeken vertrekt. De knopen zijn genummerd van 1 tot n, i.e. $V = \{1, 2, \ldots, n\}$.

   Uitvoer De array P met $P[v]$ de voorganger van v op een kortste pad van s naar v; als $P[v] = 0$ dan is er geen pad van s naar v.

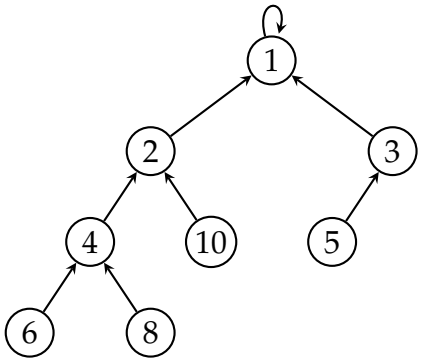
```
1:  function KortstePadOngewogenMetPad(G, s)
2:      P ← [0, 0, ..., 0]                          ▷ n keer 0
3:      P[s] ← s                          ▷ voorganger van s is zichzelf
4:      Q.init()                                    ▷ wachtrij van knopen
5:      Q.enqueue(s)
6:      while Q ≠ ∅ do
7:          v ← Q.dequeue()
8:          for all w ∈ buren(v) do
9:              if P[w] = 0 then                    ▷ w nog niet ontdekt
10:                 P[w] ← v                        ▷ w is gevonden via v
11:                 Q.enqueue(w)
12:             end if
13:         end for
14:     end while
15:     return P
16: end function
```

   b) We houden de array P bij:

	1	2	3	4	5	6	7	8	9	10
(a)	0	0	0	0	0	0	0	0	0	0
(b)	1	0	0	0	0	0	0	0	0	0
(c)	1	1	0	0	0	0	0	0	0	0
(d)	1	1	1	0	0	0	0	0	0	0
(e)	1	1	1	2	0	0	0	0	0	0
(f)	1	1	1	2	0	0	0	0	0	2
(g)	1	1	1	2	3	0	0	0	0	2
(h)	1	1	1	2	3	4	0	0	0	2
(i)	1	1	1	2	3	4	0	4	0	2

We zien dus dat er geen pad is van 1 naar 7 noch naar 9 aangezien zowel $P[7]$ als $P[9]$ nog steeds de waarde nul hebben.

De informatie in de array P kan ook als volgt worden voorgesteld:



Hieruit zie je duidelijk dat de array P in essentie de kortste paden bevat maar in *omgekeerde volgorde* gezien vanuit de startknoop s.

c) Het idee is om te starten bij de knoop v en steeds achteruit te lopen totdat we de knoop s bereiken. De knopen die onderweg worden ontmoet moeten *vooraan* de lijst worden toegevoegd. We controleren echter eerst of er wel een pad is.

Invoer Een array P met voorgangers vanuit een knoop s. Een knoop v waarnaar het pad moet worden gevonden. De knopen zijn genummerd van 1 tot n, i.e. $V = \{1, 2, \ldots, n\}$.

Uitvoer Een lijst van knopen startend bij s en eindigend bij v. Indien er geen pad bestaat van s naar v dan is de lijst leeg.

```
1: function GeefPad(P, s, v)
2:     if P[v] = 0 then                ▷ Controleer of het pad bestaat
3:         return ∅
```

```
4:      end if
5:      L ← [v]                          ▷ lijst met één element
6:      c ← v                            ▷ c is de "huidige"knoop
7:      while c ≠ s do
8:          c ← P[c]
9:          L ← c :: L                   ▷ c vooraan toevoegen aan lijst
10:     end while
11:     return L
12: end function
```

Om onszelf te overtuigen dat het algoritme inderdaad correct werkt doorlopen we het nog eens stap voor stap voor de array

i	1	2	3	4	5	6	7	8	9	10
$P[i]$	1	1	1	2	3	4	0	4	0	2

met $s = 1$ en $v = 8$.

s	v	c	L
1	8		
			[8]
		8	
		4	
			[4, 8]
		2	
			[2, 4, 8]
		1	
			[1, 2, 4, 8]

Het algoritme eindigt met als pad $1 \to 2 \to 4 \to 8$. Dit is inderdaad het correcte antwoord.

2. Dit probleem kan aangepakt worden m.b.v. het aangepaste algoritme KORTSTEPADONGEWOGENMETPAD. Elk woord stelt een knoop van de graaf voor. Twee knopen zijn adjacent als hun woorden slechts op één plaats van elkaar verschillen.

3. We passen het algoritme van Dijkstra toe en we tonen telkens de inhoud van de array D wanneer de lus op regel 5 start[1]. De elementen die tot S behoren (en waarvoor de kortste afstand dus gekend is) worden aangeduid met een sterretje.

[1] Behalve de eerste regel die de beginwaarde van D toont.

	1	2	3	4	5	6	7	8	9	10	11
(a)	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
(b)	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
(c)	0*	57	∞	∞	∞	133	∞	∞	∞	∞	∞
(d)	0*	57*	112	115	143	133	∞	∞	∞	∞	∞
(e)	0*	57*	112*	115	142	133	∞	∞	211	147	195
(f)	0*	57*	112*	115*	142	133	∞	∞	211	147	194
(g)	0*	57*	112*	115*	142	133*	210	∞	211	147	194
(h)	0*	57*	112*	115*	142*	133*	210	∞	211	147	194
(i)	0*	57*	112*	115*	142*	133*	185	∞	211	147*	194
(j)	0*	57*	112*	115*	142*	133*	185*	315	211	147*	194
(k)	0*	57*	112*	115*	142*	133*	185*	315	211	147*	194*
(l)	0*	57*	112*	115*	142*	133*	185*	315	211*	147*	194*
(m)	0*	57*	112*	115*	142*	133*	185*	315*	211*	147*	194*

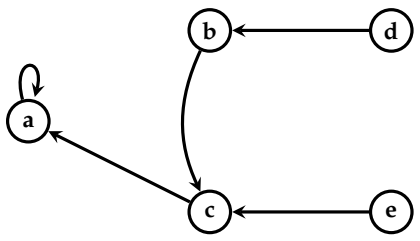
Uit deze tabel kan je waarnemen dat voor steden Leuven (5), Namen (7) en Hasselt (11) het eerste gevonden pad niet het kortste is.

4. We houden opnieuw de array D bij en we houden ook een array P bij (net zoals bij het aangepaste algoritme voor het kortste pad in ongewogen grafen). We vinden:

D	a	b	c	d	e
(a)	0	∞	∞	∞	∞
(b)	0*	11	5	∞	∞
(c)	0*	8	5*	11	7
(d)	0*	8	5*	11	7*
(e)	0*	8*	5*	10	7*
(f)	0*	8*	5*	10*	7*

P	a	b	c	d	e
(a)	a	∅	∅	∅	∅
(b)	a	a	a	∅	∅
(c)	a	c	a	c	c
(d)	a	c	a	c	c
(e)	a	c	a	b	c
(f)	a	c	a	b	c

We duiden de inhoud van de array P aan op onderstaande figuur. Dit geeft meteen ook de mogelijkheid om de (omgekeerde) paden te vinden.



5.5.4 Oefeningen

1. Vind een minimale kost opspannende boom m.b.v. het algoritme van Prim voor de graaf in Figuur 5.4. Neem als startknoop "Brugge".

2. Vind een minimale kost opspannende boom m.b.v. het algoritme van Kruskal voor de graaf in Figuur 5.4.

5.5.5 Oplossingen

1. Het algoritme van Prim is een gulzige versie van het algoritme voor generiek zoeken waarbij telkens de goedkoopste boog wordt gekozen die het ontdekte gebied uitbreidt met een nieuwe knoop.

   In Tabel 5.1 kan je volgen wat er gebeurt. In Figuur 5.4 staan de bogen van de minimale kost opspannende boom in het vetjes getekend.

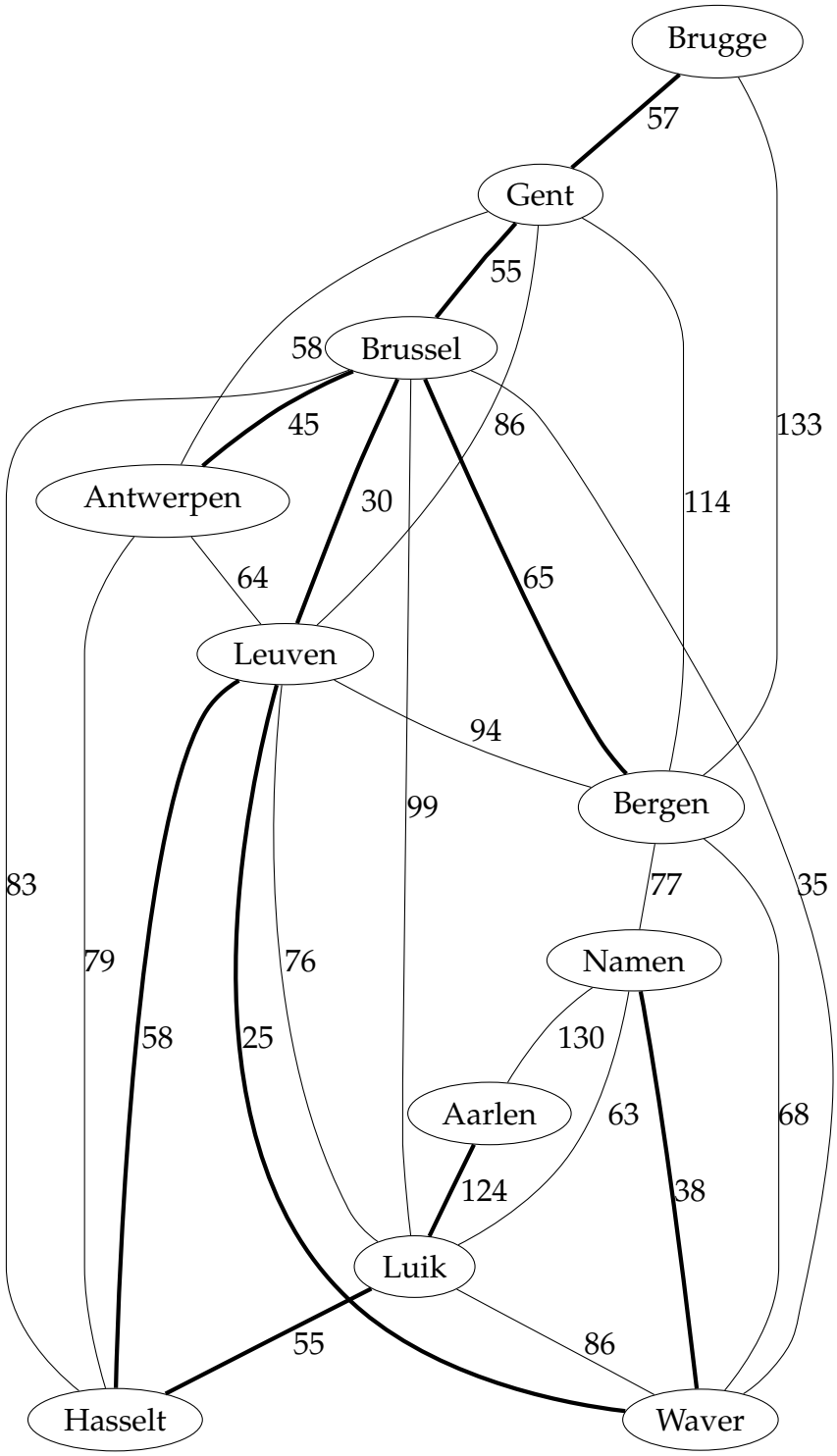
2. Bij het algoritme van Kruskal worden de bogen eerst gesorteerd in stijgende volgorde van gewicht. De bogen worden in deze volgorde overlopen en wanneer het toevoegen van een boog geen cykel veroorzaakt dan wordt die toegevoegd. Dit gaat verder tot alle bogen overlopen zijn (of tot er $n - 1$ bogen zijn toegevoegd).

   Om uit te vissen of het toevoegen van een boog een cykel veroorzaakt gaan we hier als volgt tewerk. Initieel behoort elke knoop tot zijn eigen "component", genummerd van 1 tot n. Een component is een verzameling knopen die geconnecteerd zijn m.b.v. reeds gekozen bogen.

   Wanneer een boog wordt toegevoegd tussen twee componenten i en j dan worden deze samengevoegd tot één component. Als $i < j$ dan veranderen we het componentnummer van de knopen die tot j behoorden eenvoudigweg in i. We mogen nooit een boog toevoegen tussen twee knopen in dezelfde component omdat dit een cykel zou veroorzaken. Inderdaad, het feit dat de twee knopen v en w tot dezelfde component behoren betekent dat er reeds een pad bestaat van v naar w m.b.v. bogen die tot de gekozen bogen behoren. Toevoegen van de boog (w, v) zou van dit pad een cykel maken. Dit is uiteraard niet toegelaten.

gemarkeerde knopen	mogelijke bogen	gekozen boog
$\{1\}$	$1 \xrightarrow{57} 2, 1 \xrightarrow{133} 6$	$1 \xrightarrow{57} 2$
$\{1,2\}$	$1 \xrightarrow{133} 6, 2 \xrightarrow{55} 3, 2 \xrightarrow{58} 4$ $2 \xrightarrow{86} 5, 2 \xrightarrow{114} 6$	$2 \xrightarrow{55} 3$
$\{1,2,3\}$	$1 \xrightarrow{133} 6, 2 \xrightarrow{58} 4, 2 \xrightarrow{86} 5, 2 \xrightarrow{114} 6$ $3 \xrightarrow{45} 4, 3 \xrightarrow{30} 5, 3 \xrightarrow{65} 6, 3 \xrightarrow{99} 9$ $3 \xrightarrow{35} 10, 3 \xrightarrow{83} 11$	$3 \xrightarrow{30} 5$
$\{1,2,3,5\}$	$1 \xrightarrow{133} 6, 2 \xrightarrow{58} 4, 2 \xrightarrow{114} 6$ $3 \xrightarrow{45} 4, 3 \xrightarrow{65} 6 , 3 \xrightarrow{99} 9$ $3 \xrightarrow{35} 10, 3 \xrightarrow{83} 11, 5 \xrightarrow{64} 4$ $5 \xrightarrow{94} 6, 5 \xrightarrow{76} 9, 5 \xrightarrow{25} 10, 5 \xrightarrow{58} 11$	$5 \xrightarrow{25} 10$
$\{1,2,3,5,10\}$	$1 \xrightarrow{133} 6, 2 \xrightarrow{58} 4, 2 \xrightarrow{114} 6$ $3 \xrightarrow{45} 4, 3 \xrightarrow{65} 6 , 3 \xrightarrow{99} 9$ $3 \xrightarrow{83} 11, 5 \xrightarrow{64} 4$ $5 \xrightarrow{94} 6, 5 \xrightarrow{76} 9, 5 \xrightarrow{58} 11$ $10 \xrightarrow{68} 6, 10 \xrightarrow{38} 7, 10 \xrightarrow{86} 9$	$10 \xrightarrow{38} 7$
$\{1,2,3,5,10,7\}$	$1 \xrightarrow{133} 6, 2 \xrightarrow{58} 4, 2 \xrightarrow{114} 6$ $3 \xrightarrow{45} 4, 3 \xrightarrow{65} 6 , 3 \xrightarrow{99} 9$ $3 \xrightarrow{83} 11, 5 \xrightarrow{64} 4$ $5 \xrightarrow{94} 6, 5 \xrightarrow{76} 9, 5 \xrightarrow{58} 11$ $10 \xrightarrow{68} 6, 10 \xrightarrow{86} 9$ $7 \xrightarrow{77} 6, 7 \xrightarrow{130} 8, 7 \xrightarrow{63} 9$	$3 \xrightarrow{45} 4$
$\{1,2,3,5,10,7,4\}$	$1 \xrightarrow{133} 6, 2 \xrightarrow{114} 6, 3 \xrightarrow{65} 6$ $3 \xrightarrow{99} 9, 3 \xrightarrow{83} 11$ $5 \xrightarrow{94} 6, 5 \xrightarrow{76} 9, 5 \xrightarrow{58} 11$ $10 \xrightarrow{68} 6, 10 \xrightarrow{86} 9, 7 \xrightarrow{77} 6$ $7 \xrightarrow{130} 8, 7 \xrightarrow{63} 9, 4 \xrightarrow{79} 11$	$5 \xrightarrow{58} 11$
$\{1,2,3,5,10,7,4,11\}$	$1 \xrightarrow{133} 6, 2 \xrightarrow{114} 6, 3 \xrightarrow{65} 6, 3 \xrightarrow{99} 9$ $5 \xrightarrow{94} 6, 5 \xrightarrow{76} 9$ $10 \xrightarrow{68} 6, 10 \xrightarrow{86} 9, 7 \xrightarrow{77} 6$ $7 \xrightarrow{130} 8, 7 \xrightarrow{63} 9, 11 \xrightarrow{55} 9$	$11 \xrightarrow{55} 9$
$\{1,2,3,5,10,7,4,11,9\}$	$1 \xrightarrow{133} 6, 2 \xrightarrow{114} 6, 3 \xrightarrow{65} 6, 5 \xrightarrow{94} 6$ $10 \xrightarrow{68} 6, 7 \xrightarrow{77} 6 , 7 \xrightarrow{130} 8, 9 \xrightarrow{124} 8$	$3 \xrightarrow{65} 6$
$\{1,2,3,5,10,7,4,11,9,6\}$	$7 \xrightarrow{130} 8, 9 \xrightarrow{124} 8$	$9 \xrightarrow{124} 8$
$\{1,2,3,5,10,7,4,11,9,6,8\}$	$\emptyset$	$\emptyset$

Tabel 5.1: Stap voor stap uitvoering van het algoritme van Prim op de graaf met Belgische provinciehoofdsteden.



Figuur 5.4: Minimale kost opspannende boom gevonden met het algoritme van Prim startend vanuit de knoop "Brugge".

We starten met de 26 bogen in stijgende volgorde van gewicht:

$$5 \xrightarrow{25} 10,3 \xrightarrow{30} 5,3 \xrightarrow{35} 10,7 \xrightarrow{38} 10,3 \xrightarrow{45} 4,2 \xrightarrow{55} 3,$$
$$9 \xrightarrow{55} 11,1 \xrightarrow{57} 2,2 \xrightarrow{58} 4,5 \xrightarrow{58} 11,7 \xrightarrow{63} 9,4 \xrightarrow{64} 5,$$
$$3 \xrightarrow{65} 6,6 \xrightarrow{68} 10,5 \xrightarrow{76} 9,6 \xrightarrow{77} 7,4 \xrightarrow{79} 11,3 \xrightarrow{83} 11,2 \xrightarrow{86} 5,$$
$$9 \xrightarrow{86} 10,5 \xrightarrow{94} 6,3 \xrightarrow{99} 9,2 \xrightarrow{114} 6,8 \xrightarrow{124} 9,7 \xrightarrow{130} 8,1 \xrightarrow{133} 6$$

Initieel behoren alle knopen tot hun eigen component:

knoop	1	2	3	4	5	6	7	8	9	10	11
component	1	2	3	4	5	6	7	8	9	10	11

De eerste boog $5 \xrightarrow{25} 10$ verbindt twee knopen in twee verschillende componenten. Door deze boog smelten deze twee componenten samen tot één component:

knoop	1	2	3	4	5	6	7	8	9	10	11
component	1	2	3	4	5	6	7	8	9	5	11

De volgende boog $3 \xrightarrow{30} 5$ verbindt twee knopen uit componenten 3 en 5. Alle knopen van component 5 worden toegevoegd aan component 3:

knoop	1	2	3	4	5	6	7	8	9	10	11
component	1	2	3	4	3	6	7	8	9	3	11

De boog $3 \xrightarrow{35} 10$ wordt *niet* gekozen omdat die twee knopen uit dezelfde component (3 in dit geval) verbindt. Deze boog zou dus een cykel veroorzaken onder de gekozen bogen.

De boog $7 \xrightarrow{38} 10$ wordt gekozen; de knoop 7 wordt toegevoegd aan de component 3:

knoop	1	2	3	4	5	6	7	8	9	10	11
component	1	2	3	4	3	6	3	8	9	3	11

De boog $3 \xrightarrow{45} 4$ wordt ook gekozen, en de knoop 4 wordt toegevoegd aan de component 3:

knoop	1	2	3	4	5	6	7	8	9	10	11
component	1	2	3	3	3	6	3	8	9	3	11

De boog 2 $\xrightarrow{55}$ 3 wordt ook gekozen. Alle knopen van component 3 worden toegevoegd aan component 2[2].

knoop	1	2	3	4	5	6	7	8	9	10	11
component	1	2	2	2	2	6	2	8	9	2	11

De volgende boog 9 $\xrightarrow{55}$ 11 wordt ook gekozen. De knopen 9 en 11 vormen nu samen één component.

knoop	1	2	3	4	5	6	7	8	9	10	11
component	1	2	2	2	2	6	2	8	9	2	9

De boog 1 $\xrightarrow{57}$ 2 wordt ook gekozen:

knoop	1	2	3	4	5	6	7	8	9	10	11
component	1	1	1	1	1	6	1	8	9	1	9

De boog 2 $\xrightarrow{58}$ 4 wordt *niet* gekozen want deze boog verbindt twee knopen uit dezelfde component. De boog 5 $\xrightarrow{58}$ 11 wordt dan weer wel gekozen. De twee knopen uit de component 9 worden toegevoegd aan component 1:

knoop	1	2	3	4	5	6	7	8	9	10	11
component	1	1	1	1	1	6	1	8	1	1	1

De bogen 7 $\xrightarrow{63}$ 9 en 4 $\xrightarrow{64}$ 5 worden allebei niet gekozen. De boog 3 $\xrightarrow{65}$ 6 wordt wel gekozen en de knoop 6 wordt toegevoegd aan component 1.

knoop	1	2	3	4	5	6	7	8	9	10	11
component	1	1	1	1	1	1	1	8	1	1	1

[2] In de praktijk is het efficiënter om de kleinste component van naam te veranderen.

Alle bogen tot voor $8 \xrightarrow{124} 9$ worden niet gekozen want deze verbinden allemaal twee knopen uit dezelfde component (nl. component 1). De boog $8 \xrightarrow{124} 9$ wordt gekozen en nu behoren alle knopen tot dezelfde component:

knoop	1	2	3	4	5	6	7	8	9	10	11
component	1	1	1	1	1	1	1	1	1	1	1

Dit betekent dat we een minimale kost opspannende boom hebben gevonden. In dit geval bestaat de minimale kost opspannende boom dus uit de bogen:

$$\begin{aligned} T = \{&5 \xrightarrow{25} 10, 3 \xrightarrow{30} 5, 7 \xrightarrow{38} 10, 3 \xrightarrow{45} 4, 2 \xrightarrow{55} 3, \\ &9 \xrightarrow{55} 11, 1 \xrightarrow{57} 2, 5 \xrightarrow{58} 11, 3 \xrightarrow{65} 6, 8 \xrightarrow{124} 9\}. \end{aligned}$$

In dit geval is dit dezelfde minimale kost opspannende boom als gevonden met het algoritme van Prim. Het totale gewicht van deze boom is:

$$\begin{aligned} \text{gewicht}(T) &= 25 + 30 + 38 + 45 + 55 + 55 + 57 + 58 + 65 + 124 \\ &= 552. \end{aligned}$$

5.6.1 Oefeningen

1. Beschouw opnieuw de acht steden in Figuur 5.18, maar veronderstel nu dat het gewicht van een boog gegeven wordt door de zogenaamde Manhattan-distance tussen de twee knopen, dus

$$d\big((x_1, y_1), (x_2, y_2)\big) = |x_2 - x_1| + |y_2 - y_1|$$

   a) Ga na dat de Manhattan-distance aan de driehoeksongelijkheid voldoet. **Hint**: voor de absolute waarde geldt dat

$$|x + y| \leq |x| + |y|.$$

   b) Pas het benaderende algoritme voor het oplossen van het handelsreizigersprobleem toe op dit probleem. Gebruik Kruskals algoritme om de minimale opspannende boom te construeren. Wanneer meerdere bogen kunnen gekozen worden, kies dan steeds de lexicografisch kleinste boog. Neem de knoop a als wortel van de opspannende boom.

5.6.2 Oplossingen

1. a) We moeten aantonen dat voor elk willekeurig drietal punten (x_1, y_1), (x_2, y_2) en (x_3, y_3) geldt dat:

$$d\big((x_1, y_1), (x_3, y_3)\big) \leq d\big((x_1, y_1), (x_2, y_2)\big) + d\big((x_2, y_2), (x_3, y_3)\big).$$

We starten met het linkerlid:

$$\begin{aligned}
&d\big((x_1, y_1), (x_3, y_3)\big) \\
&\quad = |x_3 - x_1| + |y_3 - y_1| \\
&\quad = |x_3 - x_2 + x_2 - x_1| + |y_3 - y_2 + y_2 - y_1| \\
&\quad \leq |x_3 - x_2| + |x_2 - x_1| + |y_3 - y_2| + |y_2 - y_1| \\
&\quad = \big(|x_2 - x_1| + |y_2 - y_1|\big) + \big(|x_3 - x_2| + |y_3 - y_2|\big) \\
&\quad = d\big((x_1, y_1), (x_2, y_2)\big) + d\big((x_2, y_2), (x_3, y_3)\big).
\end{aligned}$$

Dit toont aan dat de Manhattan-distance aan de driehoeksongelijkheid voldoet.

b) We bouwen de adjacentiematrix (met gewichten) op wanneer de afstand tussen de steden gemeten wordt m.b.v. de Manhattan-distance:

$$A = \begin{pmatrix}
0 & 2 & 5 & 2 & 5 & 6 & 7 & 6 \\
2 & 0 & 3 & 4 & 5 & 4 & 5 & 4 \\
5 & 3 & 0 & 7 & 8 & 5 & 8 & 3 \\
2 & 4 & 7 & 0 & 3 & 4 & 5 & 6 \\
5 & 5 & 8 & 3 & 0 & 3 & 2 & 7 \\
6 & 4 & 5 & 4 & 3 & 0 & 3 & 4 \\
7 & 5 & 8 & 5 & 2 & 3 & 0 & 7 \\
6 & 4 & 3 & 6 & 7 & 4 & 7 & 0
\end{pmatrix}$$

Als we nu een minimale kost opspannende boom opbouwen m.b.v. het algoritme van Kruskal dan worden de volgende bogen gekozen:

$$(a, b), (a, d), (e, g), (b, c), (c, h), (d, e), (e, f).$$

Dit is dezelfde minimale kost opspannende boom als reeds gevonden in het voorbeeld in de theorie (zij het met andere gewichten). De bijhorende benaderende oplossing zal dus identiek zijn.

HOOFDSTUK 6

Zoekalgoritmes

6.6 Oefeningen

1. (**?**, Oefening 3.4) Beschouw twee vrienden die in verschillende steden wonen, bv. in Roemenië. Bij elke actie kunnen we elke vriend simultaan naar een naburige stad op de kaart verplaatsen. De hoeveelheid tijd nodig om zich van stad i naar de aanpalende stad j te verplaatsen is gelijk aan de afstand $d(i, j)$. Bij elke actie moet de vriend die eerst aankomt wachten tot de andere ook aankomt. De twee vrienden willen zo vlug als mogelijk samenkomen.

   a) Geef een gedetailleerde beschrijving van het zoekprobleem.

   b) Beschouw $D(i, j)$ als de afstand in vogelvlucht tussen de twee steden i en j. Welke van volgende heuristieken zijn toelaatbaar wanneer de eerste vriend zich in stad i en de tweede zich in stad j bevindt.

      i. $D(i, j)$
      ii. $2 \cdot D(i, j)$
      iii. $D(i, j)/2$

   c) Zijn er toestanden (in de wetenschap dat er een pad is tussen alle steden op de kaart) waarvoor geen oplossing bestaat? Leg uit.

2. Een aantal robots (bv. k) leven in een rooster waarin sommige locaties muren zijn. Twee robots kunnen zich nooit op dezelfde locatie bevinden. Elke robot heeft zijn eigen bestemming. Bij elke tijdseenheid

verplaatsen de robots zich simultaan naar een aanpalend (vrij) vierkant of blijven ze staan. Twee robots die zich naast elkaar bevinden kunnen niet van plaats wisselen in één tijdseenheid. Elke tijdseenheid kost één punt. Beantwoord de volgende vragen:

a) Geef een minimale correcte voorstelling van een toestand in de toestandsruimte.

b) Schat de grootte van de toestandsruimte in voor een rooster met als afmetingen $M \times N$.

c) Welke van volgende heuristieken zijn toelaatbaar? Beargumenteer je antwoord, meer bepaald: geef een situatie waarvan je aangeeft dat de gegeven heuristiek niet toelaatbaar is wanneer je dit beweert.

   i. Som van de Manhattan afstanden voor elke robot tot zijn doellocatie.

   ii. Som van de kosten van de optimale paden indien de robots zich alleen in de omgeving voortbewegen, m.a.w. zonder obstructie door andere robots.

   iii. Maximum van de Manhattan afstanden vanuit elke robotpositie tot zijn doelpositie.

   iv. Maximum van de kosten van de optimale paden indien de robots zich alleen in de omgeving voortbewegen, m.a.w. zonder obstructie door andere robots.

   v. Aantal robots die zich niet op hun doellocatie bevinden.

3. Beschouw de toestandsruimtegraaf in Figuur 6.1. De starttoestand is steeds S en de doeltoestand is G. Voer een aantal zoekalgoritmes uit op deze toestandsruimtegraaf. Indien er ergens "random" een plan uit de open lijst moet gekozen worden, neem je het plan dat eindigt in de lexicografisch kleinste toestand. Hierdoor wordt de oplossing steeds uniek.

   De heuristiek gebruikt door de geïnformeerde zoekmethoden staat gegeven in onderstaande tabel:

toestand	S	A	B	C	D	E	G
h	6	0	6	4	1	10	0

Geef voor onderstaande blinde en geïnformeerde zoekmethodes het pad naar de doeltoestand. Geef ook aan welke toestanden geëxpandeerd werden en dit in de juiste volgorde van hun expansie. De geïnformeerde zoekmethoden gebruiken de heuristiek h.

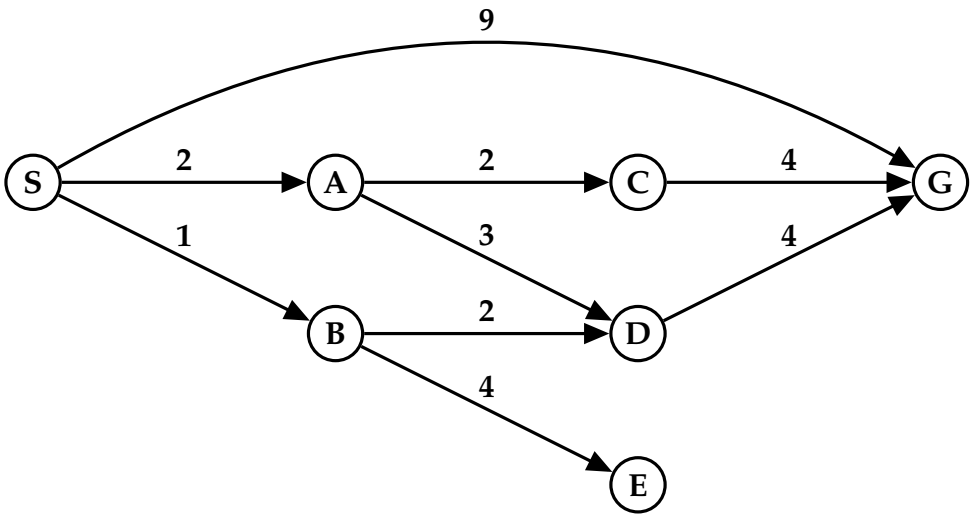
a) Diepte-eerst (boomgebaseerd)

b) Gulzig beste-eerst (boomgebaseerd)

c) Uniforme kost zoeken (boomgebaseerd)

d) A^* (boomgebaseerd)

e) A^* (graafgebaseerd)



Figuur 6.1: Toestandsruimtegraaf voor vraag 3.

4. Pas het gulzig beste eerst algoritme toe om van Arad naar Bucharest te gaan op de kaart van Roemenië in Figuur 2.11. Gebruik de afstand in vogelvlucht uit Tabel 2.2 als heuristiek.

5. Gebruik A* om van Lugoj naar Bucharest te gaan op de kaart van Roemenië in Figuur 2.11 met de afstand in vogelvlucht als heuristiek. Teken de opgebouwde zoekboom en geef aan in welke volgorde de plannen verwijderd worden van de open lijst. Los deze oefening eerst op voor boomgebaseerd zoeken en vervolgens voor graafgebaseerd zoeken.

Figuur 6.2: Illustratie van het spel Rush Hour.

6. Rush Hour wordt gespeeld op een $n \times n$ spelbord. Op het spelbord staan een aantal auto's, die elk twee aaneengrenzende vakjes beslaan, en een aantal vrachtwagens, die elk drie aaneengrenzende vakjes beslaan. Het spelbord bevat aan één van de zijden een uitgang. De spelregels zijn heel eenvoudig: je kan bij elke beurt een willekeurige auto of vrachtwagen een aantal vakjes voorwaarts of achterwaarts verplaatsen in de richting waarin deze geplaatst is op het spelbord. Uiteraard kunnen voertuigen elkaar niet overlappen en kunnen ze niet over elkaar springen. Het probleem is dat een specifieke auto (op Figuur 6.2 gemarkeerd met X) naar de uitgang van het spelbord moet geleid worden. Hierbij moet de totale afgelegde afstand (som van alle afstanden door verplaatsen van auto's en/of vrachtwagens) zo laag mogelijk gehouden worden.

   Geef minstens twee aanvaardbare heuristieken voor dit probleem verschillend van de triviale aanvaardbare heuristiek $h = 0$.

6.7 Oplossingen

1. a) We veronderstellen dat er n steden zijn, genummerd 1 t.e.m. n. Een *toestand* wordt voorgesteld door een koppel (i, j) waarbij i (resp. j) de huidige stad van de eerste (resp. tweede) vriend voorstelt. De *acties* die de vrienden kunnen ondernemen zijn zich simultaan verplaatsen naar een naburige stad:

$$(i, j) \longrightarrow (i', j'),$$

waarbij i en i' naburige steden zijn en j en j' ook. Dit geeft meteen ook het *transitiemodel* aan, want we nemen aan dat elke actie steeds lukt. De kost van de acties is

$$c\big((i, j),\ (i, j) \longrightarrow (i', j'),\ (i', j')\big) = \max\big(d(i, i'), d(j, j')\big),$$

want de vriend die als eerste aankomt moet steeds wachten tot de andere vriend ook aankomt. De totale kost voor één actie wordt dus bepaald door de traagste vriend of de grootste afstand. De initiële toestand kan om het even welke toestand (i, j) zijn. De *doeltest* voor een toestand (i, j) bestaat uit controleren of i en j gelijk zijn aan elkaar.

b) Enkel de derde heuristiek is aanvaardbaar. Beschouw hiertoe een toestandsruimte met drie steden op een rechte lijn waarbij de twee vrienden starten op de twee uiterste steden. De derde stad ligt precies in het midden van de twee steden. In dit geval is de kost van de optimale oplossing precies gelijk aan $D(i, j)/2$. Aangezien de eerste twee heuristieken een grotere waarde aangeven zijn ze niet aanvaardbaar.

c) Veronderstel een toestand (i, j) waarbij i en j zodanig zijn dat alle "paden" tussen de twee steden uit een oneven aantal stappen bestaat. Vanuit deze toestand kunnen de twee vrienden elkaar nooit bereiken. Inderdaad, voor elke opvolger (i', j') geldt dat alle paden tussen i' en j' ook uit een oneven aantal stappen bestaat (want anders was er een pad tussen i en j met een even aantal stappen). Aangezien een doelstand bestaat uit twee steden die *nul* (en dus een even aantal) stappen van elkaar verwijderd zijn kunnen de twee vrienden elkaar nooit bereiken. Zo'n zoekprobleem heeft m.a.w. geen oplossing.

2. a) Aangezien het rooster waarin de robots leven "vast" is voor een specifiek probleem is het nodig en voldoende om *de positie van elke robot* te kennen om te weten wat de toestand is.

   b) In een eerste orde benadering zijn er voor elke robot $M \times N$ posities mogelijk. Het aantal toestanden wordt dus benaderd door

   $$(M \times N)^k.$$

   Een iets nauwkeurigere berekening levert:

   $$(M \times N) \times (M \times N - 1) \times (M \times N - 2) \times \cdots \times (M \times N - k + 1).$$

   c) Voor alle tegenvoorbeelden gebruiken we de volgende toestand waarbij P_i de huidige positie voorstelt van robot i en G_i de bestemming van robot i voorstelt:

G_1	P_1	P_2	G_2

   Voor deze toestand ziet men onmiddellijk dat de optimale oplossing kost 1 heeft: de actie "robot 1 beweegt naar links terwijl robot 2 simultaan naar rechts beweegt" levert in één stap een eindtoestand op.

   i. Dit is geen toelaatbare heuristiek: de waarde van deze heuristiek is 2, terwijl de optimale oplossing kost 1 heeft.

   ii. Dit is geen toelaatbare heuristiek: de waarde van deze heuristiek is 2, terwijl de optimale oplossing kost 1 heeft.

   iii. Dit is een toelaatbare heuristiek. Elke robot moet minstens de Manhattan afstand naar zijn eindlocatie overbruggen en bij elke actie kan de Manhattan afstand naar zijn eindlocatie (voor een specifieke robot) hoogstens met één verminderen. Het *maximum* van de Manhattan afstanden is m.a.w. een toelaatbare heuristiek. Deze heuristiek is de oplossing van het vereenvoudigd probleem waarbij we de muren wegdenken en de robots elkaar niet hinderen.

   iv. Dit is eveneens een toelaatbare heuristiek: het is de oplossing van het vereenvoudigd probleem waarin we veronderstellen dat de robots elkaar niet hinderen. Merk op dat het berekenen van deze heuristiek heel wat tijdsintensiever is dan het berekenen van de voorgaande heuristiek.

v. Dit is geen toelaatbare heuristiek. Voor het voorbeeld is de waarde van deze heuristiek gelijk aan 2 terwijl de optimale oplossing kost 1 heeft.

3. De oplossing voor de verschillende zoekmethoden staan in onderstaande tabel:

zoekmethode	pad naar doel	geëxpandeerde knopen
Diepte-eerst (boomgebaseerd)	$S \to A \to C \to G$	$S, A, C, (G)$
Gulzig beste-eerst (boomgebaseerd)	$S \to G$	$S, A, (G)$
Uniforme kost zoeken (boomgebaseerd)	$S \to B \to D \to G$	$S, B, A, D, C, D, E, (G)$
A^* (boomgebaseerd)	$S \to B \to D \to G$	$S, A, D, B, D, (G)$
A^* (graafgebaseerd)	$S \to A \to C \to G$	$S, A, D, B, C, (G)$

Merk op dat bij boomgebaseerde zoekmethoden dezelfde toestand meerdere malen kan geëxpandeerd worden terwijl dit bij graafgebaseerde methoden niet zo is. Men ziet gemakkelijk dat de heuristiek toelaatbaar is. Daarom vindt A* in de boomgebaseerde vorm een optimale oplossing (dezelfde als uniforme kost zoeken). Maar, de heuristiek is niet consistent: zo geldt bv. voor de toestanden S en A (waarbij A een opvolger is van S) dat

$$h(S) > c(S, a, A) + h(A).$$

Voor een consistente heuristiek moet echter voor elke toestand s en opvolger s' gelden dat:

$$h(s) \leq c(s, a, s') + h(s').$$

Graafgebaseerde A* is dus niet gegarandeerd om een optimale oplossing te vinden. In dit geval wordt er inderdaad een suboptimale oplossing gevonden.

4. Bij gulzig beste eerst is de f-waarde gelijk aan de heuristische waarde h. Initieel bestaat de open lijst enkel uit het plan $(A, 366)$. Dit plan

wordt geëxpandeerd en de plannen $S \to A$, $T \to A$ en $Z \to A$ worden aan de open lijst toegevoegd:

$$(S \to A, 253), (T \to A, 329), (Z \to A, 374).$$

Het plan $S \to A$ heeft de laagste f-waarde en wordt als volgende geëxandeerd. De volgende plannen worden toegevoegd aan de open lijst:

$$(A \to S \to A, 366), (F \to S \to A, 176),$$
$$(O \to S \to A, 380), (RV \to S \to A, 193).$$

Van de zes plannen op de open lijst heeft $F \to S \to A$ de laagste f-waarde. Dit plan wordt geëxpandeerd en de open lijst wordt aangevuld met:

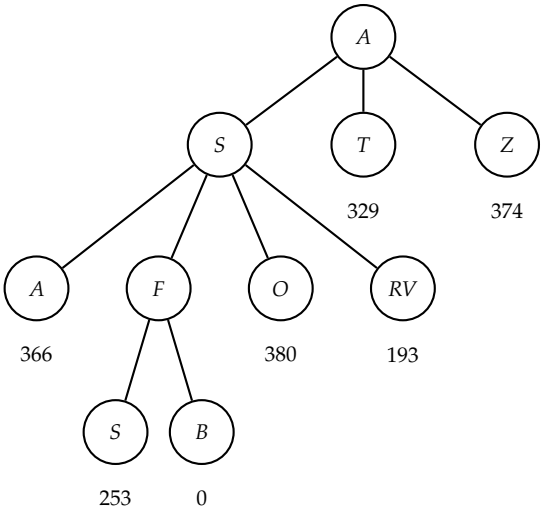
$$(S \to F \to S \to A, 253), (B \to F \to S \to A, 0).$$

Het plan $B \to F \to S \to A$ heeft de laagste f-waarde. Aangezien het doel bereikt werd stop het algoritme met de oplossing:

$$A \to S \to F \to B.$$

Dit is *niet* de optimale oplossing.

De opgebouwde zoekboom ziet er als volgt uit:



5. We ordenen de open lijst op prioriteit. Plannen worden, zoals in de cursus, achterstevoren genoteerd maar (om plaats te besparen) zonder de pijltjes tussen de verschillende toestanden. Bij boomgebaseerd zoeken evolueert de open lijst als volgt.

- $(L, 0 + 244 = 244)$
- $([M, L], 70 + 241 = 311), ([T, L], 111 + 329 = 440)$
- $([L, M, L], 140 + 244 = 384), ([D, M, L], 145 + 242 = 387),$
  $([T, L], 111 + 329 = 440)$
- $([D, M, L], 145 + 242 = 387), ([T, L], 111 + 329 = 440),$
  $([M, L, M, L], 210 + 241 = 451), ([T, L, M, L], 251 + 329 = 580)$
- $([C, D, M, L], 265 + 160 = 425), ([T, L], 111 + 329 = 440),$
  $([M, L, M, L], 210 + 241 = 451), ([M, D, M, L], 220 + 241 = 461),$
  $([T, L, M, L], 251 + 329 = 580)$
- $([T, L], 111 + 329 = 440), ([M, L, M, L], 210 + 241 = 451),$
  $([M, D, M, L], 220 + 241 = 461), ([P, C, D, M, L], 403 + 100 = 503),$
  $([T, L, M, L], 251 + 329 = 580), ([RV, C, D, M, L], 411 + 193 = 604),$
  $([D, C, D, M, L], 385 + 242 = 627)$
- $([M, L, M, L], 210 + 241 = 451), ([M, D, M, L], 220 + 241 = 461),$
  $([L, T, L], 222 + 244 = 466), ([P, C, D, M, L], 403 + 100 = 503),$
  $([T, L, M, L], 251 + 329 = 580), ([A, T, L], 229 + 366 = 595),$
  $([RV, C, D, M, L], 411 + 193 = 604), ([D, C, D, M, L], 385 + 242 = 627)$
- $([M, D, M, L], 220 + 241 = 461), ([L, T, L], 222 + 244 = 466),$
  $([P, C, D, M, L], 403 + 100 = 503), ([L, M, L, M, L], 280 + 244 = 524),$
  $([D, M, L, M, L], 285 + 242 = 527), ([T, L, M, L], 251 + 329 = 580),$
  $([A, T, L], 229 + 366 = 595), ([RV, C, D, M, L], 411 + 193 = 604),$
  $([D, C, D, M, L], 385 + 242 = 627)$
- $([L, T, L], 222 + 244 = 466), ([P, C, D, M, L], 403 + 100 = 503),$
  $([L, M, L, M, L], 280 + 244 = 524), ([D, M, L, M, L], 285 + 242 = 527),$
  $([L, M, D, M, L], 290 + 244 = 534), ([D, M, D, M, L], 295 + 242 = 537),$
  $([T, L, M, L], 251 + 329 = 580), ([A, T, L], 229 + 366 = 595),$
  $([RV, C, D, M, L], 411 + 193 = 604), ([D, C, D, M, L], 385 + 242 = 627)$
- $([P, C, D, M, L], 403 + 100 = 503), ([L, M, L, M, L], 280 + 244 = 524),$
  $([D, M, L, M, L], 285 + 242 = 527), ([M, L, T, L], 292 + 241 = 533),$
  $([L, M, D, M, L], 290 + 244 = 534), ([D, M, D, M, L], 295 + 242 = 537),$
  $([T, L, M, L], 251 + 329 = 580), ([A, T, L], 229 + 366 = 595),$
  $([RV, C, D, M, L], 411 + 193 = 604), ([D, C, D, M, L], 385 + 242 = 627),$
  $([T, L, T, L], 333 + 329 = 662)$
- $([B, P, C, D, M, L], 504 + 0 = 504), ([L, M, L, M, L], 280 + 244 = 524),$
  $([D, M, L, M, L], 285 + 242 = 527), ([M, L, T, L], 292 + 241 = 533),$
  $([L, M, D, M, L], 290 + 244 = 534), ([D, M, D, M, L], 295 + 242 = 537),$
  $([T, L, M, L], 251 + 329 = 580), ([A, T, L], 229 + 366 = 595),$
  $([RV, C, D, M, L], 411 + 193 = 604), ([D, C, D, M, L], 385 + 242 = 627),$
  $([T, L, T, L], 333 + 329 = 662), ([RV, P, C, D, M, L], 500 + 193 = 693),$
  $([C, P, C, D, M, L], 541 + 160 = 701)$

Het algoritme eindigt met de actiesequentie

$$L \to M \to D \to C \to P \to B$$

als oplossing. De kost van deze oplossing is 504.
Bij graafgebaseerd zoeken krijgen we het volgende:

- $(L, 0 + 244 = 244)$
- $([M, L], 70 + 241 = 311), ([T, L], 111 + 329 = 440)$
- $([L, M, L], 140 + 244 = 384), ([D, M, L], 145 + 242 = 387),$
  $([T, L], 111 + 329 = 440)$
  Dit plan wordt niet geëxpandeerd want L is reeds geëxpandeerd.
- $([D, M, L], 145 + 242 = 387), ([T, L], 111 + 329 = 440)$
- $([C, D, M, L], 265 + 160 = 425), ([T, L], 111 + 329 = 440),$
  $([M, D, M, L], 220 + 241 = 461)$
- $([T, L], 111 + 329 = 440), ([M, D, M, L], 220 + 241 = 461),$
  $([P, C, D, M, L], 403 + 100 = 503), ([RV, C, D, M, L], 411 + 193 = 604),$
  $([D, C, D, M, L], 385 + 242 = 627)$
- $([M, D, M, L], 220 + 241 = 461), ([L, T, L], 222 + 244 = 466)$
  $([P, C, D, M, L], 403 + 100 = 503), ([A, T, L], 229 + 366 = 595),$
  $([RV, C, D, M, L], 411 + 193 = 604), ([D, C, D, M, L], 385 + 242 = 627)$
  Dit plan wordt niet geëxpandeerd want M is reeds geëxpandeerd.
- $([L, T, L], 222 + 244 = 466)$
  $([P, C, D, M, L], 403 + 100 = 503), ([A, T, L], 229 + 366 = 595),$
  $([RV, C, D, M, L], 411 + 193 = 604), ([D, C, D, M, L], 385 + 242 = 627)$
  Dit plan wordt niet geëxpandeerd want L is reeds geëxpandeerd.
- $([P, C, D, M, L], 403 + 100 = 503), ([A, T, L], 229 + 366 = 595),$
  $([RV, C, D, M, L], 411 + 193 = 604), ([D, C, D, M, L], 385 + 242 = 627)$
- $([B, P, C, D, M, L], 504 + 0 = 504), ([A, T, L], 229 + 366 = 595),$
  $([RV, C, D, M, L], 411 + 193 = 604), ([D, C, D, M, L], 385 + 242 = 627),$
  $([RV, P, C, D, M, L], 500 + 193 = 693), ([C, P, C, D, M, L], 541 + 160 = 701)$

Het algoritme vindt hier (uiteraard) dezelfde oplossing want de gebruikte heuristiek is consistent en in dit geval is er juist één optimale oplossing.

6. Opdat de weg naar de uitgang vrij zou moeten alle auto's en vrachtwagens die tussen de auto gemarkeerd met X en de uitgang staan minstens over een afstand van 1 verplaatst worden. Dit geeft een eerste toelaatbare heuristiek.

   De auto gemarkeerd met X moet minstens de afstand tot de uitgang overbruggen. De afstand tot de uitgang van de auto gemarkeerd met X geeft een tweede toelaatbare heuristiek.

   Merk op dat in dit geval, omdat de bewegingen van de andere auto's en de auto gemarkeerd met X onafhankelijk zijn de *som* van beide

voorgaande heuristieken ook toelaatbaar is. Opletten: in het algemeen is de som van toelaatbare heuristieken *niet* toelaatbaar.

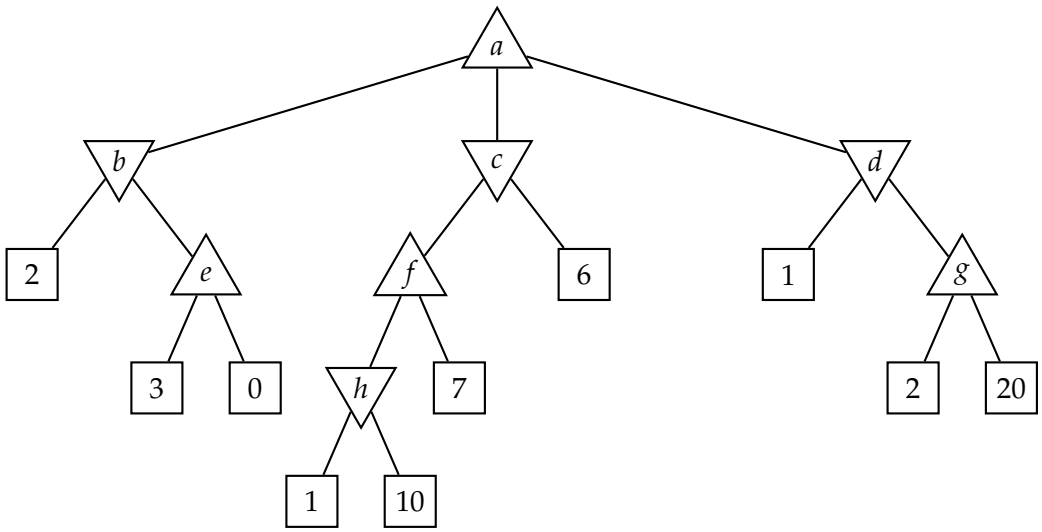
HOOFDSTUK 7

Zoeken met een Tegenstander

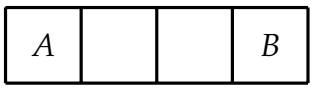
7.5 Oefeningen

1. Gegeven de spelboom in Figuur 7.1 voor een twee-speler nulsomspel waarbij Max als eerste aan de beurt is. Beide spelers zijn rationaal. In elk blad van de boom (i.e. voor elke eindtoestand) staat de waarde van de opbrengstfunctie (vanuit het oogpunt van Max) genoteerd.

   a) Geef (door aanduiding op de figuur) voor elke toestand de minimax-waarde van die toestand.



Figuur 7.1: Spelboom voor de oefeningen.



Figuur 7.2: Begintoestand van het spel beschreven in Vraag 2.

b) Welke actie zal Max ondernemen volgens het minimax algoritme?

c) In welke toestand zal het spel eindigen? Omcirkel deze op de figuur.

d) Wanneer α-β-snoeien wordt toegepast, dan kunnen er eventueel bepaalde delen van de boom gesnoeid worden. Duid deze aan op de figuur door een kruis te plaatsen op de takken die kunnen gesnoeid worden.

2. (**?**, Oef 5.8) Beschouw volgend eenvoudig spel. De begintoestand is gegeven in Figuur 7.2. Speler A, die start in vakje één, speelt de rol van Max, B, die start in vakje vier, van Min. Bij elke zet moet een speler zijn pion verplaatsen naar aan aangrenzend vakje (in beide richtingen). Indien het aangrenzend vakje ingenomen wordt door de pion van de tegenstander dan mag er over deze pion worden gesprongen. De speler die eerst aan de overkant van het bord geraakt wint. Dus als A als eerste vakje vier bereikt dan is de waarde van het spel voor A gelijk aan $+1$. Indien B als eerste vakje één bereikt dan is de waarde van het spel voor A gelijk aan -1.

   a) Teken de volledige spelboom gebruikmakend van de volgende conventies:
      - Schrijf elke toestand als (s_A, s_B) waarbij s_A en s_B de locatie van de pion van respectievelijk A en B aanduiden. De begintoestand wordt bv. genoteerd als $(1, 4)$.
      - Trek rond elke eindtoestand een vierkant en schrijf de waarde (vanuit het standpunt van Max) naast dit vierkant.
      - Trek een dubbel vierkant rond *lusstaten*, i.e. rond staten die reeds voorkomen op het pad naar de wortel van de spelboom. Aangezien hun waarde voorlopig onbepaald is noteer je dit met een "?".

   b) Schrijf bij elke knoop zijn berekende minimax waarde. Leg uit hoe je bent omgegaan met de "?".

c) Leg uit waarom het standaard minimax algoritme zou falen voor deze spelboom. Schets hoe je dit probleem zou aanpakken. Baseer je hiervoor op je antwoord in puntje (b) van deze vraag. Is je oplossing algemeen geldig?

3. Werk de spelboom voor tic tac toe uit tot op diepte twee, maar houd rekening met de symmetrieën van het bord om de spelboom beheersbaar te houden. Voor de eerste zet zijn er dan geen 9 maar slechts drie mogelijke zetten, nl. in een hoek, aan een rand en centraal. Beantwoord vervolgens de volgende vragen.

   a) Geef voor elke toestand op diepte twee de waarde van de evaluatiefunctie EVAL zoals gedefinieerd in (3.1).

   b) Geef op basis van deze evaluatiefunctie de h-minimax waarde en de h-minimax beslissing.

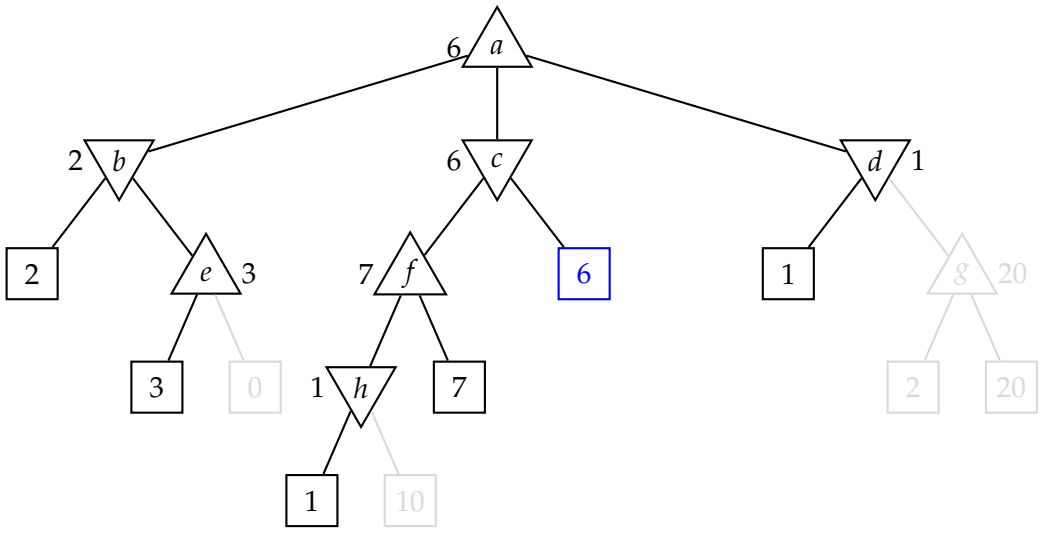
   c) Pas α-β snoeien toe. Welke takken kunnen er gesnoeid worden? Neem voor deze vraag aan dat het evalueren van de toppen in de *optimale* volgorde gebeurt zodanig dat er maximaal kan gesnoeid worden.

4. Veronderstel dat, voor de spelboom in Figuur (3.2) van de cursustekst, de speler Max *weet* dat Min "lui" is en steeds de eerste actie kiest. Wat is dan de beste actie voor Max? Is deze gelijk aan de minimax beslissing?

5. Schrijf, in pseudocode, het hoofdprogramma voor een agent die een bepaald spel gaat spelen tegen een menselijke speler. De computer speelt eerst. Je mag gebruikmaken van de functie MINIMAXDECISION. Je mag verder veronderstellen dat s.SHOWSTATE de toestand van het spel op het scherm toont (zodat de menselijke speler kan zien wat de toestand is). Met GETANDPARSEACTION lees je de actie die de menselijke speler wil doen in van het toetsenbord en zet je deze om naar een "actie" a. Met s.DOACTION(a) voer je actie a uit op de toestand s.

7.6 Oplossingen

1. a) De aanduiding van de minimax waarden vindt men in Figuur 7.3 van deze oefeningenbundel.



Figuur 7.3: Geannoteerde spelboom voor de oefeningen. Delen die gesnoeid worden door $\alpha - \beta$ staan aangeduid in het lichtgrijs.

b) Max onderneemt die actie die hem de minimax waarde zal opleveren. In dit geval is dit dus de tweede actie (naar toestand c).

c) Het spel zal eindigen in de blauwgekleurde eindtoestand (zie Figuur 7.3 van deze oefeningenbundel) met waarde 6.

d) In top e weet Max dat de beste waarde voor Min op het huidig pad (β dus) gelijk is aan 2. Aangezien na het evalueren van het eerste kind van e de waarde v van deze top reeds gelijk is aan 3 (en zeker niet kan kleiner worden) weet Max dat Min nooit ervoor zal kiezen op deze top te bereiken. De volgende kinderen van e kunnen dus worden gesnoeid.

   Op het moment dat top h wordt geëvalueerd weet Min dat Max reeds de waarde 2 kan bereiken op het huidige pad, i.e. $\alpha = 2$. Na het evalueren van het eerste kind van h is de waarde v van deze top gelijk aan 1. Omdat $v \leq \alpha$ kunnen de volgende kinderen gesnoeid worden.

   Op het moment dat de top d wordt geëvalueerd heeft α de waarde 6. Na het evalueren van het eerste kind is de waarde v van deze top gelijk aan 1. Omdat $v \leq \alpha$ kunnen de volgende kinderen van d gesnoeid worden.

2. a) De volledige spelboom staat getekend in Figuur 7.4.

b) Om de minimaxwaarde te kennen van $(3, 2)$ moeten we het minimum berekenen van -1 en een onbekende waarde *die ofwel* -1 *of* 1 *is*. Het minimum is dus steeds -1, *wat ook de andere waarde is*. De minimax waarde van $(3, 2)$ is m.a.w. gelijk aan -1. Voor de toestand $(1, 3)$ is dezelfde redenering geldig.

c) Het standaard minimax algoritme, dat in essentie diepte eerst verloopt, zou vastraken in een oneindige lus. De lusstaten worden immers standaard niet gedetecteerd.

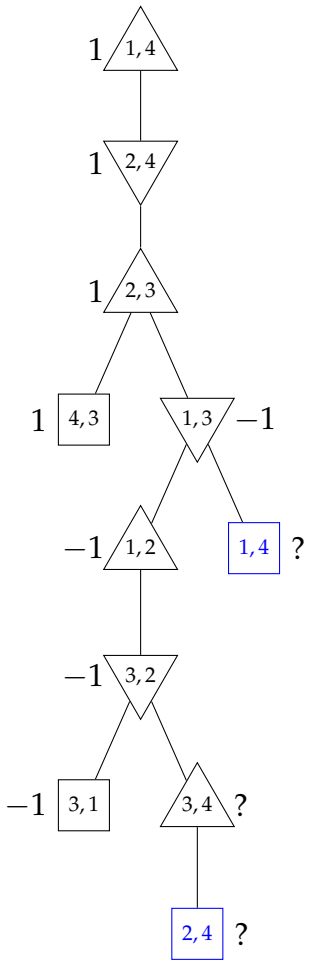
De oplossing bestaat erin om lusstaten te detecteren en ze een ongekende waarde toe te kennen. In dit geval loste dit het probleem op. Maar als men bv. een spel zou hebben waar er ook gelijkspel zou kunnen optreden dan is het niet duidelijk wat de uitkomst is van het vergelijken van een ongekende waarde met een gelijkspel. Andere methodes (buiten de scope van deze cursus) zijn dan nodig om de minimax waarde te bepalen.

3. a) De spelboom wordt getoond in Figuur 7.5. De waarde van de functie EVAL wordt bij elk blad aangeduid. De EVAL functie telt in dit geval het aantal rijen waarop een X staat en trekt daarvan het aantal rijen waarop een O staat af. (In dit geval zijn er geen rijen waarop twee X'en of twee O's voorkomen).

b) Het minimax algoritme werd gebruikt om de h-waarden van de andere toppen te bepalen. Op basis van deze waarden zie je dat de optimale actie erin bestaat om de X centraal te plaatsen.

c) Wanneer de toppen optimaal geordend zouden zijn dan zijn ze op het eerste niveau geordend van grote h-waarde naar kleine h-waarde, i.e. eerst centraal, dan in een hoek en tenslotte op het midden van een rij. Op het volgende niveau zouden de toppen dan geordend moeten zijn van kleine h-waarde naar grote h-waarde (omdat hier Min aan zet is). Wanneer men dit zou doen dan kunnen heel wat toppen gesnoeid worden, nl. al de toppen die in Figuur 7.5 in het grijs zijn aangeduid.

4. Wanneer Max weet dat Min steeds de eerste actie neemt dan kan Max de derde actie ondernemen om zo de maximale waarde 14 te bekomen. Deze waarde en actie zijn niet gelijk aan de minimax waarde en actie. De minimax waarde is in dit geval strikt hoger.

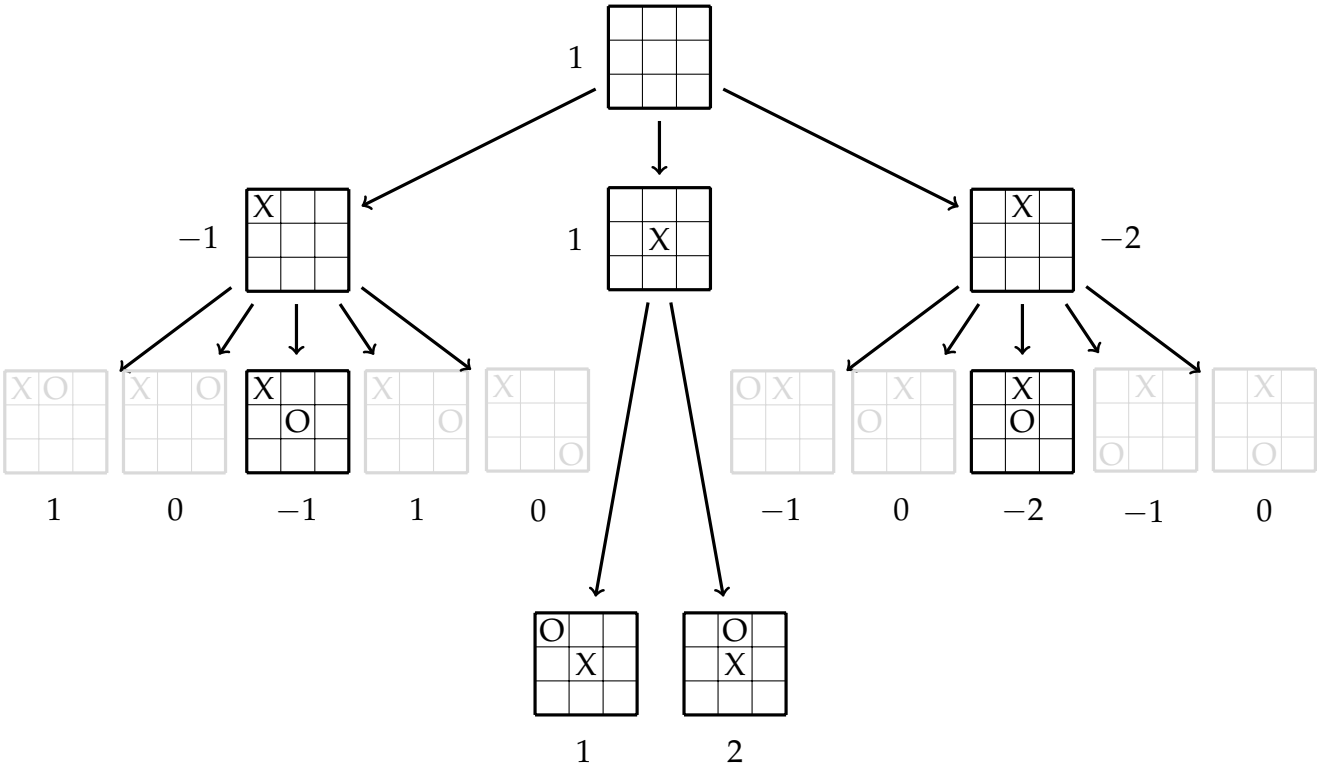


Figuur 7.4: Spelboom voor het spel in Vraag 2. De lusstaten zijn aangeduid in het blauw. We zien dat speler A het spel wint. De spelboom heeft immers een minimax waarde van 1.

Merk op: wanneer Max toch volgens Minimax speelt en Min niet, dan krijgt Max in dit geval een opbrengst van 12, i.p.v. de opbrengst 3 wanneer Min ook volgens het minimax algoritme speelt.

Dit illustreert dat de waarde voor Max niet slechter kan worden als Min niet volgens minimax speelt, maar dat er tegen zo'n speler er eventueel strategieën kunnen zijn die *nog meer* opleveren. Wanneer Min echter deze strategie zou te weten komen dan kan deze daar in principe ook misbruik van maken!

5. Voor deze opgave zijn er natuurlijk een aantal verschillende oplossin-



Figuur 7.5: Spelboom voor tic tac toe. De spelboom is uitgewerkt tot op diepte 2 en er werd rekening gehouden met de symmetrieën van het spel. De grijze toppen kunnen gesnoeid worden m.b.v. α-β snoeien *op voorwaarde dat de toppen optimaal geordend werden*.

gen mogelijk. Alle oplossingen zullen echter gebruik moeten maken van een lus waarin MINIMAXDECISION zal aangeroepen worden om te bepalen wat de computer moet doen. De lus (en het spel) eindigt wanneer *s*.ISTERMINALSTATE de waarde **true** teruggeeft. Een mogelijke oplossing wordt gegeven in Algoritme 5 van deze oplossingenbundel.

```
1: function PLAYGAME
2:     s ← initiële toestand spel
3:     while s.ISTERMINALSTATE ≠ true do
4:         PRINT "De huidige toestand is " s.SHOWSTATE
5:         a ← MINIMAXDECISION(s)
6:         PRINT "De computer doet zet " a
7:         s.DOACTION(a)
8:         if s.ISTERMINALSTATE ≠ true then
9:             PRINT "De huidige toestand is " s.SHOWSTATE
10:            PRINT "Geef je zet."
11:            a ← GETANDPARSEACTION()
12:            s.DOACTION(a)
13:        end if
14:    end while
15:    PRINT "De eindtoestand is " s.SHOWSTATE
16:    PRINT "De computer haalt " s.GETUTILITY(Max) " punten."
17:    PRINT "Jij haalt " s.GETUTILITY(Min) " punten."
18: end function
```

HOOFDSTUK 8

Machinaal Leren

8.4 Oefeningen

1. a) De hoeveelheid regenval wordt gewoonlijk uitgedrukt in een aantal mm per dag. Veronderstel dat je wil voorspellen hoeveel regen er morgen zal vallen op een bepaalde plaats. Je beschikt hiervoor over de historische weerobservaties.

   Wat is er van toepassing?

   - classificatie
   - regressie
   - clustering
   - gesuperviseerd leren
   - ongesuperviseerd leren

   b) Sommige problemen worden best aangepakt met algoritmes voor gesuperviseerd leren en andere met algoritmes voor ongesuperviseerd leren. Welke van onderstaande problemen worden het best aangepakt met *gesuperviseerd leren*?

   - Het onderzoeken van een grote collectie emails waarvan geweten is dat ze spam zijn teneinde uit te vissen of er verschillende categorieën van spam zijn.
   - Gegeven historische data m.b.t. de leeftijd en lengte van kinderen, tracht de lengte van kinderen te voorspellen op basis van hun leeftijd.

- Trachten de artikels van verschillende nieuwsbronnen te groeperen in "gerelateerde" artikels.
- Gegeven 1000 artikels van mannelijke auteurs en 1000 artikels van vrouwelijke auteurs, tracht uit te vissen of een nieuw artikel (waarvan de auteur niet gekend is) geschreven is door een mannelijke of vrouwelijke auteur.

2. Veronderstel dat het K-gemiddelden algoritme uitgevoerd wordt met $K = 3$ en dat de drie zwaartepunten momenteel gegeven worden door:

$$\mu^{(1)} = (1,2), \quad \mu^{(2)} = (-3,0) \quad \text{en} \quad \mu^{(3)} = (4,2).$$

Gegeven het voorbeeld

$$x^{(i)} = (-1,2).$$

Wat wordt de waarde van $c^{(i)}$, of anders gezegd, aan welke cluster wordt dit voorbeeld toegewezen?

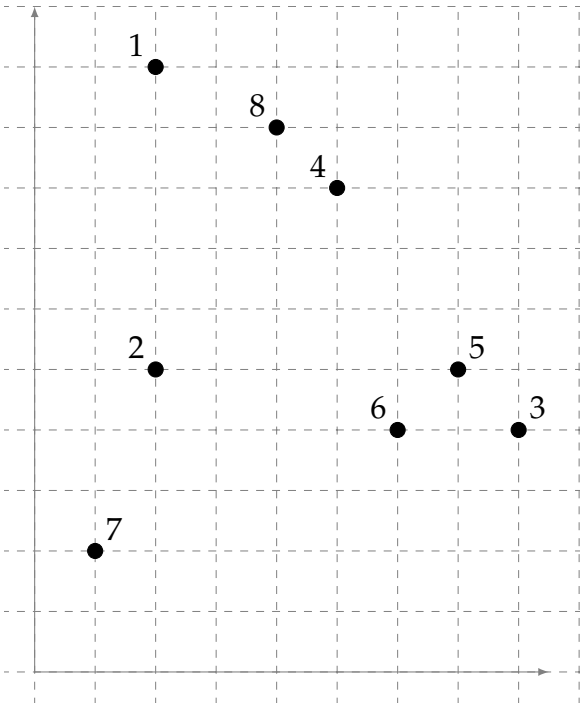
3. Gegeven de volgende ongelabelde dataset bestaande uit 8 voorbeelden:

i	$x_1^{(i)}$	$x_2^{(i)}$	i	$x_1^{(i)}$	$x_2^{(i)}$
1	2	10	5	7	5
2	2	5	6	6	4
3	8	4	7	1	2
4	5	8	8	4	9

Deze dataset wordt ook voorgesteld in Figuur 8.1.

a) Veronderstel dat de drie initiële zwaartepunten gekozen worden bij $\mu^{(1)} = x^{(1)}$, $\mu^{(2)} = x^{(4)}$ en $\mu^{(3)} = x^{(7)}$. Aan welke cluster wordt elk van de datapunten toegewezen. Wat worden de nieuwe zwaartepunten? Duid deze aan op een figuur.

b) Voer nu bijkomende iteraties van het algoritme uit tot het algoritme convergeert. Wat zijn de resulterende clusters? Komen deze overeen met hetgeen je verwacht?



Figuur 8.1: Grafische voorstelling van de dataset in Oefening 3.

8.5 Oplossingen

1. a) De volgende begrippen zijn van toepassingen:
   - regressie: we trachten immers een reëel getal te voorspellen.
   - gesuperviseerd leren: we beschikken over historische weerobservaties (die ook de hoeveelheid regenval aangeven)

   b) De volgende problemen worden best aangepakt met gesuperviseerd leren:
   - Gegeven historische data m.b.t. de leeftijd en lengte van kinderen, tracht de lengte van kinderen te voorspellen op basis van hun leeftijd.
   - Gegeven 1000 artikels van mannelijke auteurs en 1000 artikels van vrouwelijke auteurs, tracht uit te vissen of een nieuw artikel (waarvan de auteur niet gekend is) geschreven is door een mannelijke of vrouwelijke auteur.

2. We berekenen de afstand van het punt $x^{(i)} = (-1, 2)$ tot elk van de

drie zwaartepunten: We vinden

$$\begin{aligned}\|x^{(i)} - \mu^{(1)}\| &= \sqrt{(-1-1)^2 + (2-2)^2} \\ &= \sqrt{(-2)^2 + 0^2} \\ &= \sqrt{4}\end{aligned}$$

en

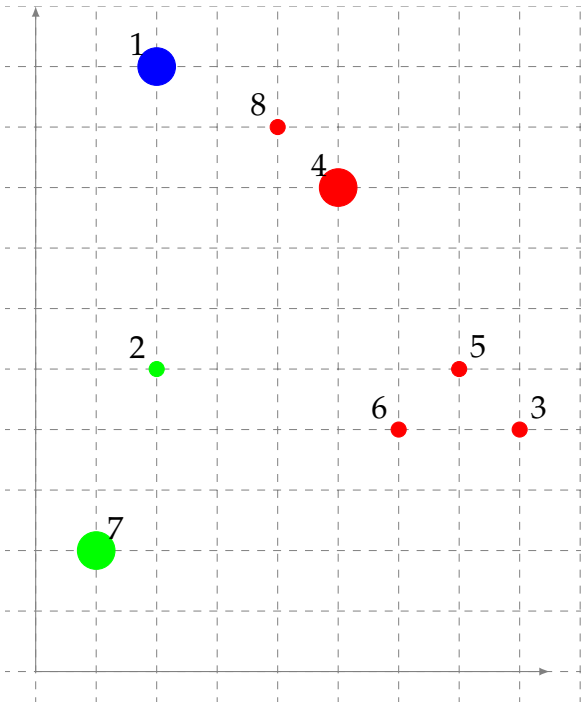
$$\begin{aligned}\|x^{(i)} - \mu^{(2)}\| &= \sqrt{(-1+3)^2 + (2-0)^2} \\ &= \sqrt{2^2 + 2^2} \\ &= \sqrt{8}\end{aligned}$$

en tenslotte

$$\begin{aligned}\|x^{(i)} - \mu^{(3)}\| &= \sqrt{(-1-4)^2 + (2-2)^2} \\ &= \sqrt{(-5)^2 + 0^2} \\ &= \sqrt{25}.\end{aligned}$$

Aangezien $\mu^{(1)}$ duidelijk het dichtst bij $x^{(i)}$ ligt krijgt $c^{(i)}$ de waarde 1.

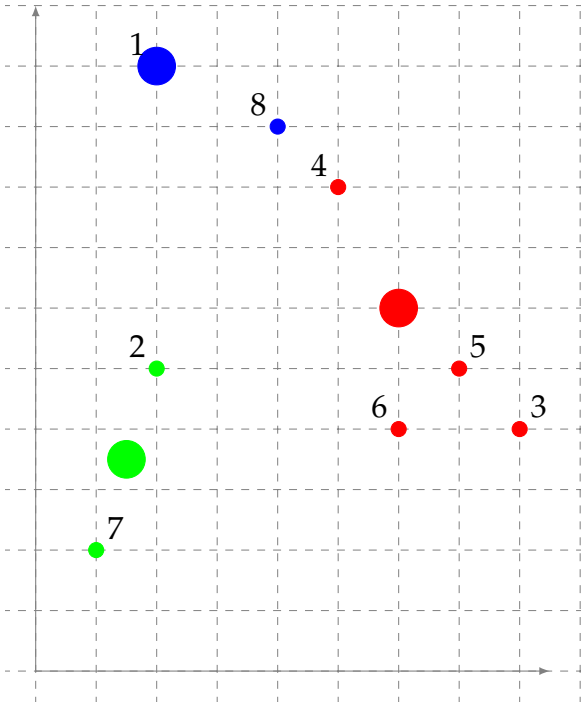
3. We zoeken de initiële assignaties door deze rechtstreeks op de figuur aan te duiden:



De nieuwe cluster zwaartepunten zijn:

$$\mu^{(1)} = x^{(1)}$$
$$\mu^{(2)} = (x^{(3)} + x^{(4)} + x^{(5)} + x^{(6)} + x^{(8)})/5 = (6,6)$$
$$\mu^{(3)} = (x^{(2)} + x^{(7)})/2 = (1.5,3.5)$$

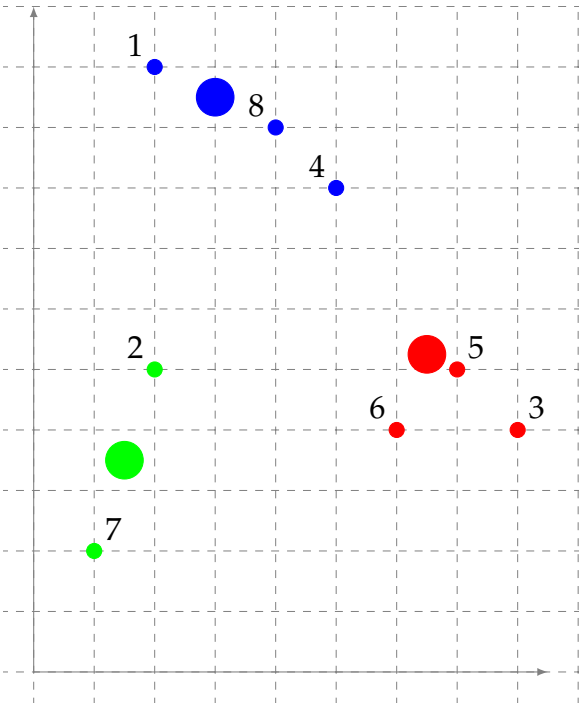
De nieuwe assignaties worden aangeduid op onderstaande figuur:



De nieuwe zwaartepunten worden:

$$\mu^{(1)} = (x^{(1)} + x^{(8)})/2 = (3,9.5)$$
$$\mu^{(2)} = (x^{(3)} + x^{(4)} + x^{(5)} + x^{(6)})/4 = (6.5,5.25)$$
$$\mu^{(3)} = (x^{(2)} + x^{(7)})/2 = (1.5,3.5)$$

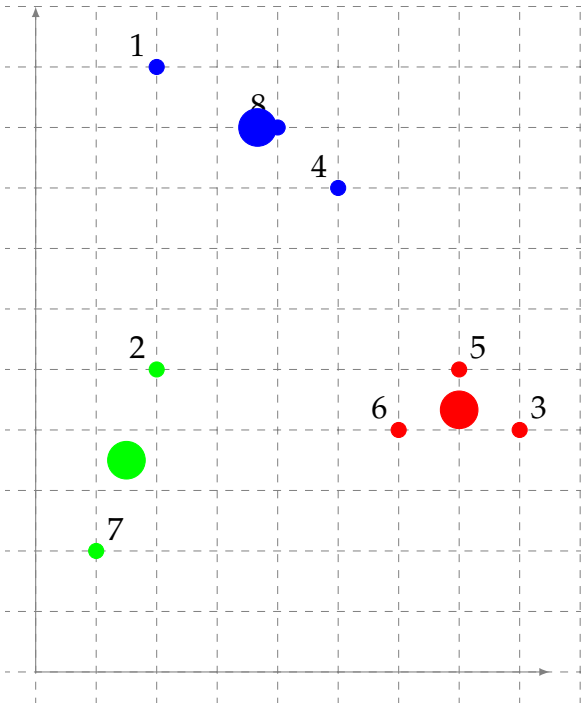
De nieuwe assignaties worden:



De nieuwe zwaartepunten worden:

$$\mu^{(1)} = (x^{(1)} + x^{(4)} + x^{(8)})/3 = (3.667, 9)$$
$$\mu^{(2)} = (x^{(3)} + x^{(5)} + x^{(6)})/3 = (7, 4.33)$$
$$\mu^{(3)} = (x^{(2)} + x^{(7)})/2 = (1.5, 3.5)$$

De assignaties worden nu:



Op dit moment stopt het algoritme. De assignaties wijzigen niet meer.

De drie ontdekte clusters zijn inderdaad de drie clusters die men ook visueel onderscheidt.