

1 Data Abstractie

Waar hieronder gesproken wordt van 'specificatie' bedoelen we een nauwkeurige beschrijving in begrijpelijk Nederlands van de werking van de operaties en een preciese opgave van de syntax van de bijbehorende procedures.

- 1.1** In het theoriedictaat wordt aangenomen dat de stapel een abstracte datastructuur is met daarop operaties voor initialisatie, toevoegen, verwijderen, bekijken bovenste element, en testen of de stapel leeg is.

We gebruiken hiervoor een constructor en operaties als Push, Pop, Top, en IsLeeg.

- a** Geef nauwkeurige beschrijvingen van deze operaties. Geef ook eventuele pre- en post-condities.
- b** Het antwoord op het vorige onderdeel van deze opgave geeft een *specificatie* van de stapel. Bespreek nu een mogelijke *representatie* en *implementatie*.
- c** Werk deze implementatie uit tot een C++ programma.
- 1.2** In Hoofdstuk 2 van het theoriedictaat staat onderstaande versie van een algoritme om met behulp van een stapel een wandeling uit te voeren in een binaire boom. (Dit algoritme is, in iets andere bewoordingen, ook terug te vinden in het dictaat Algoritmiek.)

```
Stapel S;                                1
S.Push (Wortel);                          2
while (not S.IsLeeg)                       3
do Deze = S.Pop;                           4
  while (Deze ≠ NULL)                      5
  do Bezoek (Deze); // Bezoek is elders gedeclareerd 6
    S.Push (Deze→Rechts );                7
    Deze = Deze→Links;                    8
  od                                       9
od                                        10
```

- a** In het algoritme komen twee datastructuren voor: de stapel en de binaire boom. Welke aannamen zijn gedaan over de implementatie van deze datastructuren ? (Geef aan op welke plek dit blijkt.)
- b** Doe een suggestie om het algoritme abstracter te maken. Dat wil zeggen zorg dat er minder (lieft géén) eisen impliciet aan de implementatie van de gebruikte datastructuren opgelegd worden.
- c** Bespreek een implementatie van de binaire boom waarbij bovenstaand algoritme gebruikt kan worden om een wandeling te maken. (Maar dan niet de implementatie die oorspronkelijk bedoeld werd.)
- 1.3** Beschouw de graaf uit de discrete wiskunde als datastructuur.
- a.** Doe een voorstel voor de datastructuur Graaf.

b De meest gebruikte representaties voor grafen zijn de *adjacency-list* en de *adjacency-matrix*. Zijn deze ook geschikt voor de implementatie van alle operaties uit het vorige onderdeel?

1.4 Een citaat uit het theorieboek, de paragraaf over AVL-bomen en rotaties: “ Er zijn [bij het opbergen van een verzameling sleutels] doorgaans vier problemen die de aandacht verdienen:

- (A) Vind een item met een zekere sleutel.
- (B) Vind, bij een gegeven k , de k -de sleutel in grootte.
- (C) Voeg een item toe op de juiste – al gevonden – plaats.
- (D) Verwijder een – al gevonden – item. ”

Vervolgens wordt voor drie representaties (geordend array, enkel verbonden lijst en AVL-boom) de complexiteit van deze operaties besproken.

Geef een specificatie van een abstracte datastructuur die aansluit bij bovenstaande beschrijving.

1.5 Bij veel toepassingen heeft men behoefte aan een datastructuur die elementen bevat die bestaan uit twee onderdelen, de sleutel (informatie) en de prioriteit (een getal dat de belangrijkheid aangeeft). We kunnen dan van elementen de sleutel en de prioriteit opvragen, en deze laatste aanpassen. Aan de verzameling elementen kunnen we nieuwe elementen toevoegen, we kunnen het element met de hoogste prioriteit opzoeken en verwijderen, en we kunnen kijken of de verzameling leeg is.

a Doe een voorstel voor een specificatie.

b Idem, voor de representatie. (Spielen mag: Hoofdstuk 3.1)

1.6 Bij moderne computerwerkoppervlakken bestaat het scherm uit zogenaamde *windows*, rechthoeken op het scherm die gedeeltelijk over elkaar heen kunnen liggen. Met behulp van een aanwijsinstrument (de muis) kunnen windows geselecteerd worden.

a Specificeer een abstracte datastructuur die dit geheel modelleert. De elementen van de datastructuur zijn windows, kenmerkende eigenschappen van een window zijn oa. de coördinaten van de hoekpunten. Nodig zijn in ieder geval procedures voor het initialiseren van een window, verwijderen, wijzigen afmetingen, opvragen van het window waar de muis zich in bevindt, en het bovenopleggen van een gegeven window.

b Bespreek een mogelijke implementatie. (Op efficiëntie hoeft ditmaal niet gelet te worden.)

2 (Wandelingen in) Bomen

2.1 Het vijfde getal van Catalan ($b_5 = \frac{1}{5+1} \binom{2 \cdot 5}{5}$) kan op de hieronder geschetste wijze uit b_0, \dots, b_4 gevonden worden. Verklaar dit verband.

$$\frac{1}{14} + \frac{1}{5} + \frac{2}{4} + \frac{5}{5} + \frac{14}{14} = 42$$

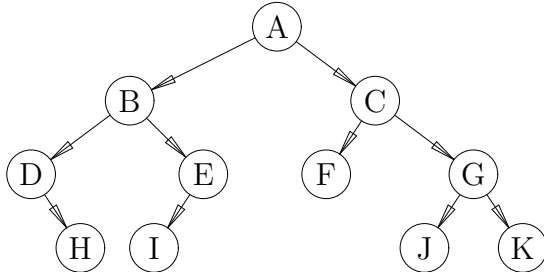
2.2 Een even aantal ($2n$) soldaten van ongelijke lengte stelt zich op in twee even lange rijen. Op hoeveel manieren kunnen zij gaan staan als hun lengtes in beide rijen van links naar rechts aflopend moeten zijn en tevens elke man in de achterste rij langer moet zijn dan de man die voor hem staat (in de eerste rij) ?

2.3 a. Geef algoritmes voor de symmetrische (LWR-) en niveau-orde wandelingen in een binaire boom. Gebruik daarbij stapels en rijen.

b Geef de recursieve versies van de pre-orde, symmetrische en post-orde boomwandel algoritmes.

2.4 We nummeren de knopen van een boom op twee manieren, volgens de pre- en de post-orde. Beschouw twee knopen in de boom. Kunnen we aan de bijbehorende nummerparen zien of de ene knoop een voorouder van de andere is?

2.5 Breng de symmetrische bedrading aan in de volgende boom:



2.6 a. Geef een implementatie in C++ van een symmetrisch bedrade boom.

b Schrijf een algoritme dat de symmetrische draden aanbrengt in een aanvankelijk niet bedrade boom. Probeer zowel een recursieve als een niet recursieve variant te vinden.

2.7 a. Schrijf een programma dat een knoop als linker kind toevoegt aan een gegeven knoop in een symmetrisch bedrade boom.

b Als a, maar nu voor het verwijderen van een linker subboom.

2.8 Neem aan dat we een (symmetrisch) bedrade boom beschouwen die tevens zoekboom is; bij een symmetrische wandeling treffen we de sleutels dus in oplopende volgorde aan.

a Knoop S bevat een wijzer naar knoop T . Kan men aan de sleutelwaarden bij S en T zien of de wijzer een tak dan wel een draad is?

b Geef een algoritme dat een LWR-wandeling uitvoert in een symmetrisch bedrade binaire zoekboom die *niet* is voorzien van tag's die de draden markeren. Neem aan dat alle sleutels verschillend zijn. Aanwijzing: welk pad zouden we volgen naar een symmetrische opvolger van een knoop? Hoe verhouden de waarden van de knopen zich op dat pad?

2.9 a. Nummer de knopen van een (symmetrisch) bedrade boom in pre-orde volgorde. Van knoop i naar knoop j loopt een link. Bepaalt de relatie tussen i en j (dwz. $i < j$ of $i > j$) of de link een tak dan wel een draad is? Bekijk linker- en rechterlinks apart.

b Gegeven is de volgende declaratie in C++.

```
enum TagType {Tak, Draad};
template <class Tp>
class Cel
{
    Tp    info;
    int  links, rechts;
    TagType ltag, rtag;
};
Cel<int> Boom[MaxAantal+1];
```

Kunnen we de knopen van een binaire boom op een slimme manier in het array ordenen zodat de tags overbodig worden? Geef dan een aangepaste versie van de functie LWR_opvolger.

2.10 Is er enige reden om bedrading te beperken tot de symmetrische versie? Onderzoek welke wandelingen mogelijk zijn met bv. pre-orde draden.

2.11 a. We bezoeken de knopen van de boom uit opgave 2.5 met het linkomkerings-algoritme. Geef aan hoe de pointers in de boom staan bij het eerste, tweede en derde bezoek aan E (de situaties waarin Deze naar E wijst). Geef bovendien aan welke waardes de tag-bits hebben.

b Idem voor de drie situaties die ontstaan direct ná deze drie bezoeken.

2.12 a. Neem aan dat de knopen van een binaire boom een geheel getal kunnen bevatten. Ga na of het mogelijk is het boomwandel-algoritme met linkomkering zo aan te passen dat na afloop van de boomwandeling elke knoop zijn gewicht bevat. (Het *gewicht* van knoop A is het totaal aantal knopen in de subboom met A als wortel - de wortel inbegrepen.)

b Idem voor het niveau van een knoop.

2.13 Pas de representatie van binaire zoekbomen zó aan dat ook de opdracht 'geef k -e sleutel' efficiënt uitgevoerd kan worden.

3 Speciale Bomen en Toepassingen

3.1 Deze opgave gaat over de datastructuur *union-find*.

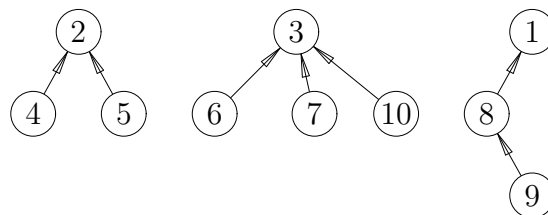
We gaan uit van N objecten (zeg getallen), die onderverdeeld zijn in een aantal (disjuncte) verzamelingen. Elk van deze verzamelingen krijgt een *naam*, voor het gemak de naam van één van de objecten uit de verzameling. Voor een aantal toepassingen zijn we geïnteresseerd in een datastructuur die deze verzamelingen kan bijhouden en bovendien de volgende twee operaties toelaat. *Union*: gegeven de naam van twee verzamelingen, voeg deze bij elkaar; *Find*: gegeven een object, geef de naam van de verzameling waartoe het object behoort. In deze opgave worden als objecten verder de getallen $1, \dots, N$ genomen. Initieel representeert de datastructuur de verzamelingen $\{1\}, \{2\}, \dots, \{N\}$.

Bij de meest voor de hand liggende oplossing krijgt elke verzameling de naam van het kleinste element. Een array $\text{Verz}[1..N]$ geeft van elk getal het kleinste element van de verzameling waartoe het element behoort.

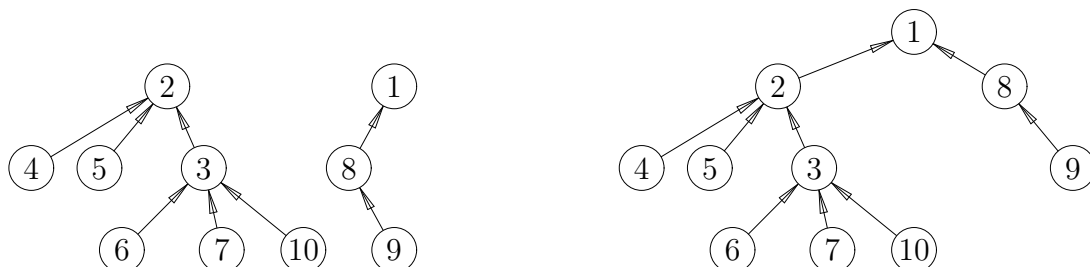
a Implementeer de operaties Union en Find.

Een alternatieve oplossing is om de verzamelingen als boomstructuur in het array op te bergen. Elke boom representeert een verzameling; de naam van die verzameling is dan het getal in de wortel.

vb. We gaan uit van de getallen 1 tm. 10 en de *partitie* $\{2, 4, 5\}, \{3, 6, 7, 10\}, \{1, 8, 9\}$. Een mogelijke representatie is onderstaand bos.



In de wortel staat dan steeds het kleinste element van de verzameling. Voegen we de eerste twee verzamelingen samen dan krijgen we —na het aanleggen van een tak van de kleinste wortel van de twee naar de grootste— de volgende situatie (twee linker bomen). In dit voorbeeld is daarbij de diepte van de eerste boom toegenomen. Vervolgens, na deze twee samengevoegd te hebben (rechts):



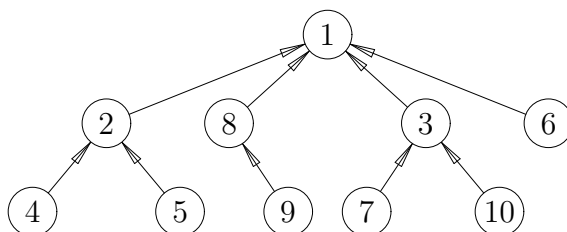
b Implementeer de gesuggereerde representatie met behulp van een array.

c Vergelijk het aantal stappen nodig voor het uitvoeren van Find en Union voor de implementatie uit **b** met die voor de implementatie uit **a**. Laat zien dat er een reeks

van maximaal n operaties Union en n operaties Find is die $\mathcal{O}(n^2)$ stappen nodig heeft, in de implementatie van **b**.

- d** Een mogelijke verbetering van de representatie is om niet steeds het kleinste getal als wortel te kiezen, maar de wortel van de hoogste boom. Implementeer deze verbetering.
- e** Laat zien dat een boom met k knopen, geconstrueerd met Union en Find zoals beschreven in het vorige onderdeel, een hoogte van maximaal $\lfloor \lg k \rfloor + 1$ heeft. (Een boom van één knoop heeft hoogte 1.) Concludeer hieruit dat een reeks van n operaties Union en Find in het slechtste geval $\mathcal{O}(n \cdot \lg n)$ stappen nodig heeft.
- f** Geef een reeks van $k - 1$ operaties Union die een boom van hoogte $\lfloor \lg k \rfloor + 1$ oplevert. Neem voor k een macht van twee.

De diepte van de bomen kan na een aantal Union opdrachten flink toenemen. Dit beïnvloedt de zoektijden voor de procedure Find nadelig. Een mogelijke oplossing is *pad-compressie*. Bij elke Find opdracht doorlopen we het pad naar de wortel twee maal. Eerst bepalen we de wortel, dan maken we van elke knoop op dit pad de ouder gelijk aan de wortel. In ons voorbeeld levert Find(6) de waarde 1 af, en laat de volgende boom achter:



- g** Schrijf deze Find Procedure. (Merk op dat de hoogte van bomen —die we gebruiken voor een efficiënte versie van Union— niet zo eenvoudig meer bepaald kan worden. Dit is geen bezwaar: de waarde die we bijhouden geeft nog steeds een bovengrens aan.)
 - h** Voor de verbetering van Union uit **d**, bestaat een mogelijk alternatief: werk met het gewicht (dwz. het aantal knopen) in plaats van de hoogte. Geldt de conclusie uit **e**, dan weer?
 - i** Knopen waarvoor we een gewicht/hoogte moeten opslaan zijn precies de wortels van de bomen, waarvoor we geen ouder hoeven te onthouden. Laat zien dat we deze gegevens in één array kunnen combineren.
- 3.2** Neem aan dat het array $A[1..N]$ gevuld is met gehele getallen. Er zijn twee voor de hand liggende manieren om dit array tot een heap om te zetten.

Bij de eerste manier werken we top-down: Als $A[1..i-1]$ al een heap-structuur heeft, voegen we $A[i]$ toe en borrelen we dit element naar de juiste plaats. De tweede aanpak is bottom-up. We voegen telkens $A[i]$ samen met de twee heaps gevormd door de kinderen $A[2i]$ en $A[2i+1]$ met al hun nakomelingen. Nu zakt $A[i]$ juist naar beneden.

a Werk deze twee algoritmen verder uit tot pseudo-code of C++ functies.

b Bespreek de efficiëntie van de methoden. Welke manier heeft de voorkeur?

3.3 (‡) Onderzoek hoe twee *heaps* gebruikt kunnen worden om van een verzameling snel zowel het grootste als het kleinste element te kunnen vinden (en verwijderen), terwijl nieuwe elementen in logaritmische tijd toegevoegd kunnen worden. Gebruik één van de *heaps* als *maxheap* (elke knoop heeft een sleutel die niet groter is dan die van zijn ouder) de ander als *minheap* (idem, maar sleutels niet kleiner dan die van ouder). Zorg ervoor dat tijdens het gebruik beide *heaps* ongeveer evenveel elementen blijven bevatten.

3.4 a. Hoe ziet een lege SMM heap er uit?

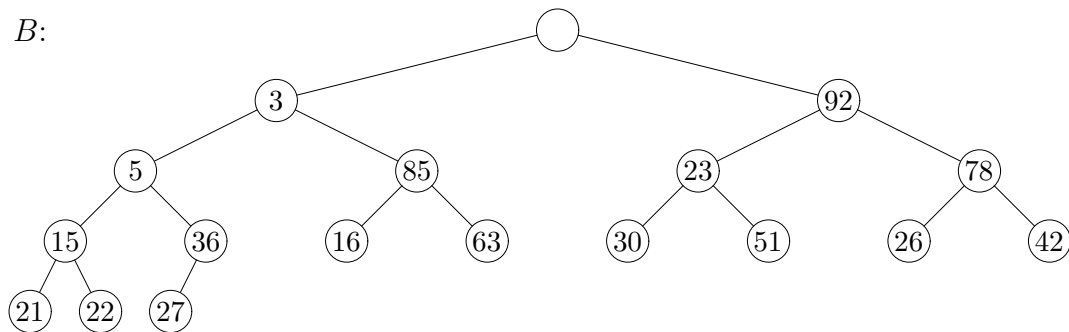
En hoe ziet een SMM heap met 1 sleutelwaarde er uit?

b Construeer een SMM heap met getallen als sleutels door, uitgaande van een lege SMM heap, achtereenvolgens de volgende getallen toe te voegen: 51, 16, 23, 63, 92, 85, 21, 36, 5, 78, 30, 3.

Uiteraard dient de boom na iedere toevoeging een SMM heap te zijn, dus borrel zonodig getallen omhoog of omlaag in de heap.

Geef via tussenresultaten en een korte toelichting duidelijk aan hoe je aan je antwoord komt.

3.5 Beschouw de volgende SMM heap B :



a Voer twee opeenvolgende DeleteMin-operaties uit op B .

Maak met tussenresultaten en een korte toelichting duidelijk wat je doet en waarom.

b Voer twee opeenvolgende DeleteMax-operaties uit op B (we beginnen dus weer met de originele boom).

Maak met tussenresultaten en een korte toelichting duidelijk wat je doet en waarom.

c Schrijf een algoritme voor DeleteMax, dat wil zeggen: een algoritme dat uit een willekeurige, niet-lege SMM heap het maximum verwijdert en de waarde retourneert.

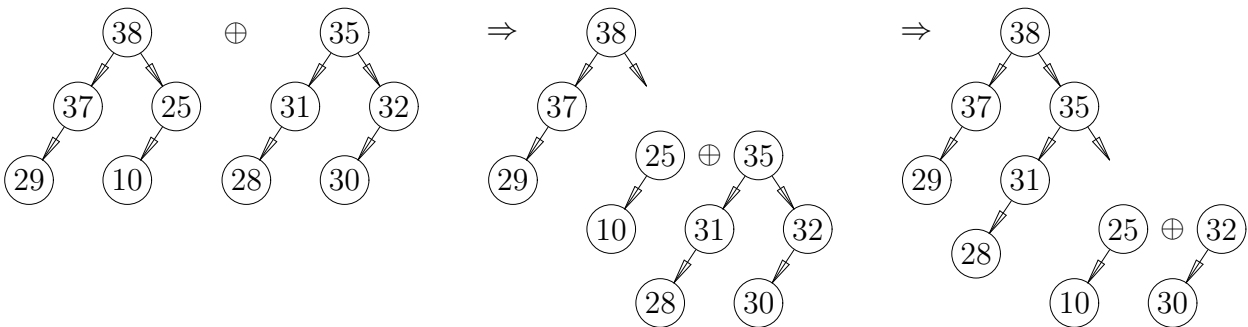
3.6 Deze opgave gaat over een alternatief voor de datastructuur *heap*. We bekijken binaire bomen, in de gewone pointer representatie. We vatten de bomen daarbij op als *uitgebreide* bomen: NULL-pointers corresponderen dan met (toegevoegde) bladeren.

```

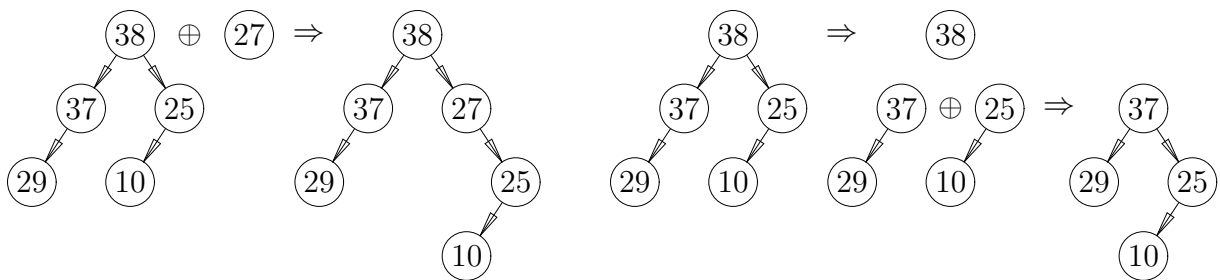
struct Knoop;
typedef Knoop *Wijzer;
struct Knoop
{
    int Waarde;
    Wijzer Links, Rechts;
    int Bladafstand;
};

```

We noemen een binaire boom met de heap-eigenschap (maar niet noodzakelijk compleet) een *max-boom*. Twee max-bomen B_1 en B_2 kunnen we samenvoegen mbv. een recursieve functie die we *rits* zullen noemen. Kijk eerst naar de waarden in de wortels van de twee bomen. Als B_1 de grootste waarde bevat dan gaat dit als volgt (verwissel anders de bomen): de rits van B_1 en B_2 ontstaat door in B_1 de rechter subboom te vervangen door de rits van B_2 en die rechter subboom. De linker subboom blijft ongewijzigd. Een voorbeeld:



- Schrijf een functie `Wijzer Rits (Wijzer B1, Wijzer B2)`; die, uitgaande van twee bomen B_1 en B_2 , de rits van B_1 en B_2 oplevert. De oorspronkelijke bomen hoeven niet bewaard te blijven.
- Zoals uit onderstaande voorbeelden blijkt, kunnen de operaties `Voegtoe` (om aan een max-boom een waarde toe te voegen) en `VerwijderMax` (om de grootste waarde uit een max-boom te verwijderen) eenvoudig mbv. de functie `rits` geïmplementeerd worden. Doe dit.



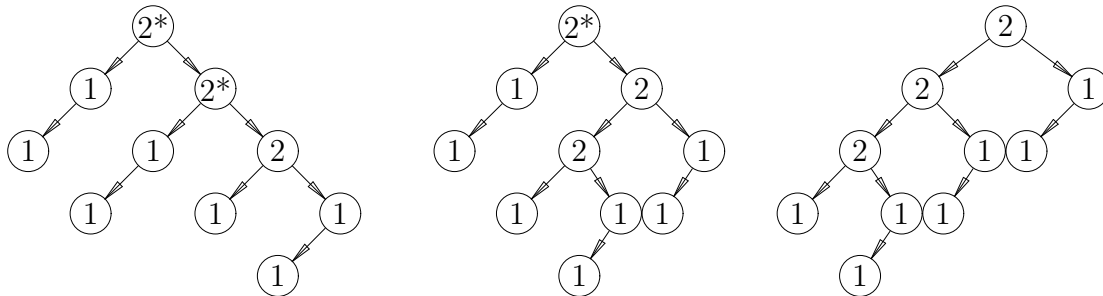
We volgen bij het ritsen steeds de rechtertakken (links blijft intact). De functie `rits` is dus erg efficiënt als in beide de te ritsen bomen de paden van de wortel naar het

meest rechter blad relatief kort zijn. Ken daartoe aan elke knoop in de boom de afstand toe naar het dichtsbijzijnde blad. We noemen deze waarde de *bladafstand* van de knoop. Voor de NULL-bladeren zou die waarde 0 zijn.

- c Schrijf een functie die de waarde BladAfstand voor een knoop uitrekent als de BladAfstand van de kinderen van de knoop reeds bekend is.

Een max-boom heet nu *links-hellend* als voor elke (interne) knoop v geldt dat $v \rightarrow \text{Links} \rightarrow \text{BladAfstand} \geq v \rightarrow \text{Rechts} \rightarrow \text{BladAfstand}$ (waarbij we voor NULL \rightarrow BladAfstand voor het gemak 0 lezen). Wanneer we twee links-hellende max-bomen ritsen ontstaat een max-boom die niet meer links-hellend hoeft te zijn. Op het pad naar rechts vanuit de wortel zijn de knopen immers veranderd. Om weer een links-hellende max-boom te maken, moeten op dit pad soms linker en rechter subbomen omgedraaid worden.

- d Pas de functie Rits zó aan dat een links-hellende max-boom ontstaat, die overal de juiste BladAfstand bevat. Je mag aannemen dat de oorspronkelijke bomen reeds de juiste waarden bevatten, en allebei links-hellend zijn.

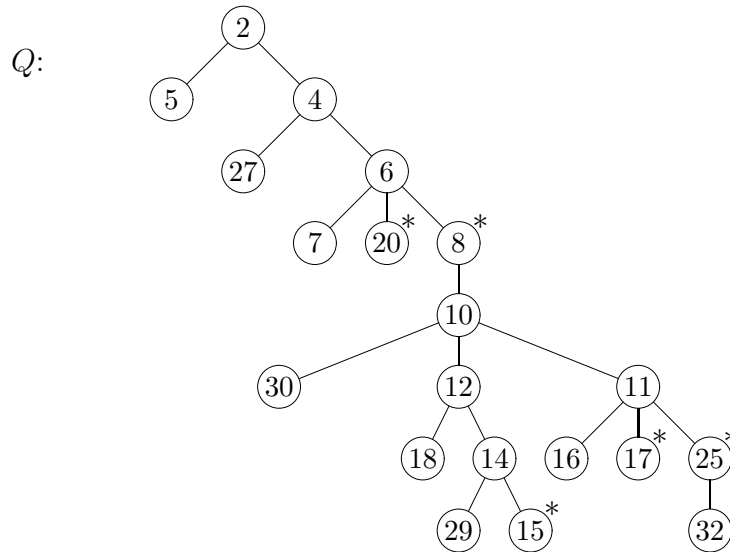


Het eerste plaatje hierboven geeft de uiteindelijke boom die ontstaat bij het eerst gegeven rits voorbeeld, maar nu met BladAfstand van elke knoop gegeven. De twee andere plaatjes laten zien hoe de links-hellende eigenschap hersteld wordt.

(tentamen december 1990)

- e (‡) Toon aan dat de constructie uit het vorige onderdeel inderdaad korte paden oplevert.

3.7 Beschouw de volgende Fibonacci queue Q (bestaande uit precies één boom):



Knopen die een kind verloren hebben nadat ze uit de wortellijst zijn gehaald, zijn gemarkeerd met een $*$.

a Voer op Q achtereenvolgens de volgende operaties uit (steeds voortbouwend op het resultaat van de voorafgaande operatie):

- Insert (19)
- Insert (3)
- Insert (23)
- Decrease (14, 10) (verlaag de waarde 14 tot 10)
- Decrease (11, 9)
- Decrease (18,11)
- DeleteMin

Geef via tussenresultaten en een korte toelichting duidelijk aan hoe je aan je antwoord komt.

b ‡ Geef operaties Insert, DeleteMin, Merge, Decrease (voor zover nodig) waarmee je, onafhankelijk van de preciese implementatie van DeleteMin, de bovenstaande Fibonacci queue Q kunt opbouwen vanuit een lege Fibonacci queue.

3.8 Implementeer Huffmans algoritme. Kies een geschikte manier om de bomen te representeren. Hoe bepalen we steeds de kleinste frequenties ?

3.9 Bepaal Huffman codes voor de volgende symbolen en bijbehorende frequentie verdelingen.

a A 20%, B 20%, C 20%, D 20%, E 20%

b A 5%, B 15%, C 20%, D 25%, E 35%

3.10 Als bovenstaande opgave, nu voor A 40%, B 20%, C 20%, D 10%, E 10% ;

b Geef de belangrijkste verschillen tussen binaire zoekbomen en bomen voor Huffman codes.

3.11 We stellen de volgende codering voor: $a \rightarrow 00$, $b \rightarrow 001$ en $c \rightarrow 101$. Kan dit een (onderdeel van een) code zijn gevonden mbv. Huffmans algoritme? Laat zien dat een verzonden boodschap te ontcijferen is.

Lukt dit ook voor $a \rightarrow 00$, $b \rightarrow 001$ en $c \rightarrow 100$?

3.12 Gegeven zijn 2^n letters met gelijke frequentie. Bewijs dat Huffmans algoritme een complete binaire boom oplevert.

3.13 Een aantal reeds gesorteerde files met gegevens moet samengevoegd worden tot één gesorteerd file. Men kan daarvoor de volgende methode gebruiken, *3-way merge* genoemd. Drie van de files worden geopend waarvan men steeds de eerste elementen blijft bekijken. Het kleinste van de drie eerste elementen wordt weggeschreven naar een nieuw file. Op deze manier reduceert men het aantal files met twee. Herhaald toepassen van de methode leidt uiteindelijk tot één gesorteerd file.

a Laat zien dat het totaal aantal verplaatste file elementen afhangt van de volgorde waarop de files bij elkaar genomen worden.

b Geef een algoritme dat bij gegeven file-lengtes het aantal verplaatsingen minimaliseert. Probeer ook de correctheid te bewijzen.

c Wat levert het algoritme bij file-lengtes 12, 7, 2, 4, 23, 17, 11 en 34?

3.14 In de (wiskundige) coderingstheorie worden coderingen bestudeerd waarbij alle letters afgebeeld worden op strings van dezelfde lengte. Wanneer we de letters uit het alfabet A coderen als een string van n bits levert dit $|A|^n$ strings op; de verzameling van deze strings noemen we C .

De *Hamming-afstand* $d(x, y)$ tussen twee woorden $x = x_1x_2 \dots x_n$ en $y = y_1y_2 \dots y_n$ uit C is het aantal plaatsen waar ze verschillen; dit wordt (voor binaire codes) uitgedrukt door de formule

$$d(x, y) = \sum_{i=1}^n |x_i - y_i|.$$

a Laat $A = \{a, b\}$; $n = 3$. Geef een zo goed mogelijke codering, dwz. kies de codewoorden op maximale afstand. Wat is de afstand tussen de strings die horen bij a en b ? Hoe gevoelig is de codering voor fouten? Hoeveel fouten bij verzenden van een codewoord kunnen nog *verbeterd* worden; hoeveel als fout *herkend*?

b Idem voor $A = \{a, b, c\}$ en $n = 3$.

c Geef een algemene relatie tussen de Hamming-afstand tussen woorden uit C en het aantal fouten dat herkend resp. verbeterd kan worden.

Een code C noemen we een *perfecte e-fouten verbeterende code* als voor elke string met n bits precies één string uit C te vinden is zodat de onderlinge afstand maximaal

e is. (De 'bollen' van straal e rond de codewoorden vullen de ruimte van alle n bit strings zonder overlappen op.)

- d Geef zo'n code voor $A = \{a, b\}$ en $n = 3$. Wat is e ?
- e Bestaat zo'n code voor $A = \{a, b, c\}$ en $n = 3$?
- f Bewijs: voor zo'n code geldt

$$|A| \cdot \sum_{i=0}^e \binom{n}{i} = 2^n$$

- g Bestaat voor $A = \{a, b, c\}$ een n en een e zodat de bijbehorende code perfect e -fout verbeterend wordt?

3.15 Dit was opgave 2 bij het tentamen Datastructuren van 19 december 2001.

Deze opgave gaat over ZLW (de-)codering van teksten over de karakters **a**, **b**, **c** en **d**. De codeerboom (een trie) die bij (de-)codering wordt opgebouwd, is dus 4-air (en niet 256-air zoals in de programmeeropdracht van najaar 2001). Als eerste code (de code voor de 'string' **a**) nemen we de code 0.

- a Codeer de tekst **cdbcacacdaaabc** met het ZLW algoritme. Bouw de codeerboom op en laat duidelijk zien welke substrings van de tekst met welke code worden gecodeerd. Geef toelichting bij de stappen die u achtereenvolgens maakt.

Ga ervanuit dat er voldoende codes beschikbaar zijn, ofwel dat de codeerboom niet 'vol' raakt.

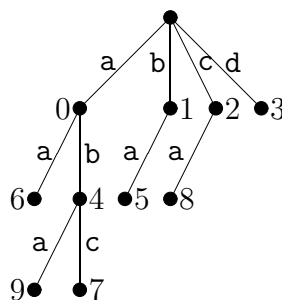
- b Veronderstel dat we bij codering van een tekst met het ZLW algoritme de volgende reeks codes hebben gekregen:

1 3 3 4 0 2 6 10 0 1 13 9.

Reconstrueer de originele tekst. Bouw ook nu (achteraf) de codeerboom op en laat zien welke substrings van de tekst met welke code zijn gecodeerd. Geef toelichting bij de stappen die u achtereenvolgens maakt.

N.B.: dit onderdeel staat (in principe) volkomen los van het vorige onderdeel

- c Veronderstel dat we een bepaalde tekst (in zijn geheel) met het ZLW algoritme gecodeerd hebben en dat we daarbij de volgende codeerboom hebben opgebouwd:



Hoe kan de tekst eruit gezien hebben? Als er meerdere mogelijkheden zijn, geef ze dan allemaal. Verklaar uw antwoord.

Ga ervanuit dat de codeerboom nog niet ‘vol’ is.

- d** Bij codering van een grote tekst kan het voorkomen dat alle beschikbare codes toegekend zijn, terwijl nog niet de gehele tekst gecodeerd is. Geef drie verschillende manieren waarop we het coderen van de tekst kunnen vervolgen (een van deze manieren was uitdrukkelijk voorgeschreven in de programmeeropdracht van najaar 2001).

3.16 We bekijken binaire bomen waarvan alleen de bladeren eigen sleutels en bijbehorende gegevens bevatten; de sleutels in de bladeren worden van links naar rechts steeds groter. De interne knopen hebben steeds twee niet lege subbomen. Ze bevatten de sleutel van het meest rechtse blad van de bijbehorende linker subboom. Deze bomen noemen we hier *zoekbomen*.

- a** Leg uit hoe de inhoud van interne knopen gebruikt kan worden om naar een gegeven sleutel in een zoekboom te zoeken.

Van de sleutels K_1, K_2, \dots, K_n waarvan de gegevens in de bladeren van zo'n (statische) zoekboom opgeslagen dienen te worden is de te verwachten frequentie-verdeling bekend: de kans dat naar K_i gezocht zal worden is α_i . Er wordt niet naar sleutels gezocht die niet in de zoekboom voorkomen.

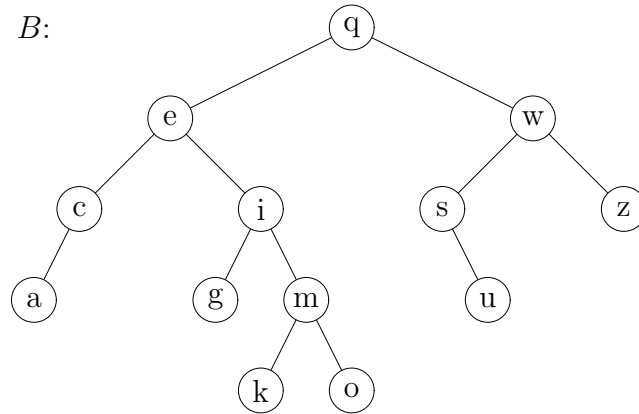
Laat T een binaire boom zijn met gegeven gewichten voor elk van de bladeren. De *gewogen blad-afstand* van T is de som (over de bladeren) van het niveau van elk blad maal zijn gewicht (hierbij nemen we aan dat het niveau van de wortel 1 is).

- b** We kennen aan elk van de bladeren van een zoekboom de frequentie van de opgeslagen sleutel toe als gewicht en rekenen de gewogen blad-afstand uit. Hoe houdt deze grootte verband met de te verwachten zoeklengte naar een sleutel in de boom? Geef een korte uitleg. Bij de zoeklengte tellen we ook het blad waarin de sleutel staat mee.
- c** Druk de gewogen blad-afstand B van een zoekboom T uit in de gewogen blad-afstanden B_1 en B_2 van de twee subbomen van de wortel van T en het totaal gewicht F van de bladeren van T . Verklaar het antwoord.

(tentamen januari 1990)

- 3.17 a.** Wat is het minimale en wat het maximale aantal knopen in een AVL-boom met hoogte $h \geq 0$ ($h = 0$ correspondeert met een lege boom, $h = 1$ met een enkele knoop)?

b Beschouw de onderstaande AVL-boom B :



Laat zien welke AVL-boom uit B ontstaat als:

- i. aan B een knoop p wordt toegevoegd,
- ii. aan B (de originele boom dus) een knoop l wordt toegevoegd,
- iii. uit B (de originele boom dus) de knoop z wordt verwijderd,
- iv. aan B (de originele boom dus) eerst de knopen d en b worden toegevoegd (in die volgorde), waarna knoop z wordt verwijderd.

Het moet steeds een AVL-boom blijven! (gebruik zonodig rotaties)

3.18 a. Welke AVL-boom ontstaat wanneer de volgende getallen in de aangegeven volgorde in de boom worden opgeslagen? Het moet steeds een binaire AVL-boom blijven: 79, 51, 32, 59, 65, 71, 37, 94, 86, 15, 23, 44.

b Bereken de interne en de externe padlengte voor deze boom. Wat is het verwachte aantal sleutelvergelijkingen nodig voor een succesvolle zoekpoging, ervan uitgaande dat elk van de aanwezige getallen een even grote kans heeft om gezocht te worden? Idem, maar nu voor een niet succesvolle zoekpoging, ervan uitgaande dat een niet aanwezig getal met gelijke kans behoort tot een van de 13 intervallen tussen en buiten de in de boom aanwezige getallen.

c Verwijder het getal 65 uit de boom en geef de resulterende AVL-boom.

3.19 Deze opgave is afgeleid van opgave 4 uit het tentamen Algoritmiëk van 3 juni 1992.

a Gegeven een binaire boom ter hoogte 5 (hoogte 1 correspondeert met een enkele knoop).

- i. Hoeveel knopen bevat zo'n boom maximaal? En hoeveel minimaal? Geef van beide gevallen een voorbeeld.
Hoeveel verschillende binaire bomen met hoogte 5 zijn er met dat maximale, respectievelijk minimale aantal knopen?
- ii. Hoeveel knopen kan zo'n boom maximaal, respectievelijk minimaal bevatten als deze tevens AVL-boom is?
Geef van beide gevallen een voorbeeld.

- b** Maak met behulp van plaatjes duidelijk wat een enkele rotatie naar rechts is. Schrijf vervolgens in C++ een stukje programma dat zo'n rotatie uitvoert bij de wortel van een binaire boom. Gebruik de volgende representatie van binaire bomen:

```

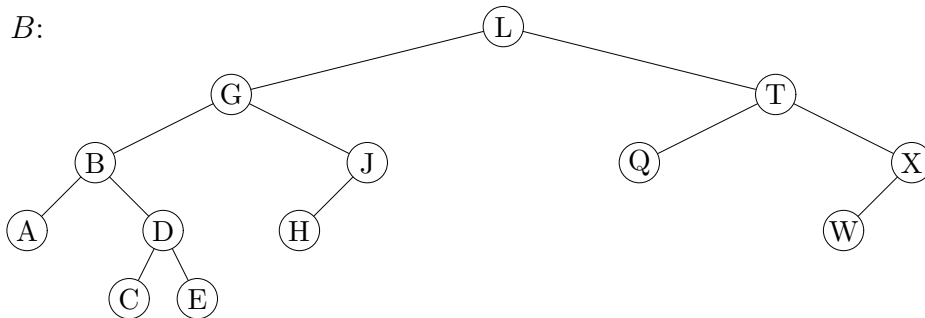
struct Knoop;
typedef Knoop *Wijzer;
struct Knoop
{
    char Info;
    Wijzer Links, Rechts;
};

```

- c** Waar moet geroteerd worden om de AVL-structuur te herstellen als er met het gewone verwijderingsalgoritme voor zoekbomen (d.w.z.: verwijderen van een knoop met twee kinderen gaat via verwisselen met de 'grootste kleinere') een sleutel is verwijderd?

Hoeveel rotaties zijn er dan maximaal nodig?

- d** Gegeven de volgende AVL-boom B :



Verwijder met behulp van het gewone zoekboom-verwijderingsalgoritme uit B de sleutel T . Zorg er vervolgens voor (met behulp van rotaties) dat de boom een AVL-boom blijft. Geef via tussenresultaten en een korte toelichting duidelijk aan hoe u aan uw antwoord komt.

3.20 (‡) Geef bij alle antwoorden voldoende uitleg.

- a** De Fibonacci getallen worden gegeven door $F_0 = 0$, $F_1 = 1$ en $F_i = F_{i-1} + F_{i-2}$ voor $i \geq 2$. Er geldt $F_i \approx c\phi^i$, met $\phi = \frac{1}{2}(1 + \sqrt{5})$ en c een zekere constante waarde.
- Hoeveel knopen bevat een AVL-boom van hoogte $h \geq 1$ maximaal, en hoeveel minimaal (hierbij heeft de boom met één knoop hoogte $h = 1$)?
 - Leid hieruit een onder- en een bovengrens af voor de hoogte h van een AVL-boom met n knopen. Welke algemene conclusie kun je hieruit trekken?
- b**
- Hoeveel verschillende AVL-bomen van hoogte $h \geq 1$ met een maximaal aantal knopen zijn er?
 - Wat is het minimale aantal interne knopen in een AVL-boom van hoogte $h \geq 1$? En wat het minimale aantal bladeren?
 - Hoeveel verschillende AVL-bomen van hoogte $h \geq 1$ met een minimaal aantal knopen zijn er dus?

- c i. Tot en met welk niveau is een AVL-boom van hoogte $h \geq 1$ in ieder geval in zijn geheel gevuld met knopen (de wortel bevindt zich op niveau 1)?
- ii. Leid hieruit een andere (weliswaar ruime) bovengrens af voor de hoogte h van een AVL-boom met n knopen.

3.21 a. Bepaal de B-boom van orde 5 die ontstaat als

k, a, m, p, w, l, q, d, v, s, b, h, x, z, g, f, t, i, j, n, r, o
 uit het gewone alfabet (met de normale ordening) in de aangegeven volgorde aan de aanvankelijk lege boom worden toegevoegd. De B-eigenschap moet steeds behouden blijven.

b Welke boom ontstaat als eerst p en daarna m verwijderd wordt?

c Als (a), maar nu een B-boom van orde 4, en met
 w, p, l, m, v, a, x, d, s, e, r, t.

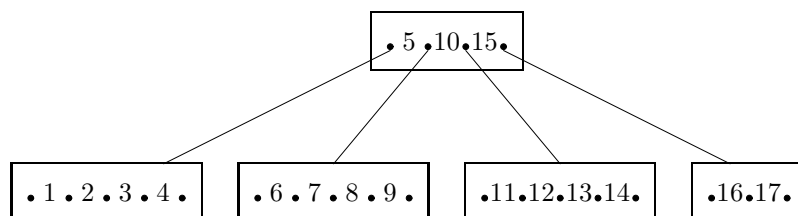
d Welke boom ontstaat als uit deze boom r verwijderd wordt?

3.22 a. Wat is de minimaal mogelijke orde van een B-boom?

b Hoeveel sleutels zijn er minimaal opgeslagen op het h -de niveau ($h \geq 1$) van een B-boom van orde 5 en hoogte $\geq h$? Idem voor maximaal.

c Welke hoogte kan een B-boom van orde 5 hebben als daarin 490 sleutels zijn opbergen?

d Geef een volgorde van de getallen 1, 2, ..., 16, 17 zodat bij het opbergen in een aanvankelijk lege B-boom van orde 5 de onderstaande boom ontstaat.

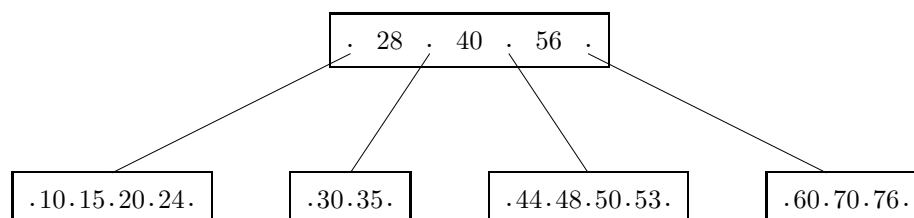


3.23 Dit was opgave 5 bij het tentamen Algoritmie van 17 augustus 1992.

a Hoeveel sleutels kan een knoop in een B-boom van orde 5 maximaal bevatten? En hoeveel minimaal?

b Stel dat we een B-boom van orde 5 hebben met drie knopen. Hoeveel sleutels bevat hij dan minimaal? En hoeveel maximaal?

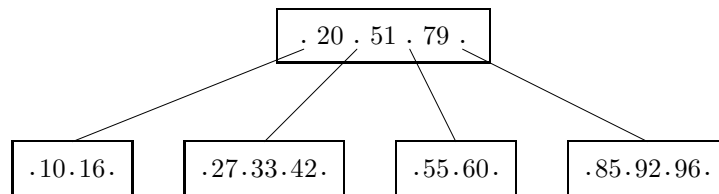
c Gegeven de volgende B-boom van orde 5:



Welke boom ontstaat als we achtereenvolgens de getallen 25 en 45 toevoegen? De boom moet uiteraard steeds een B-boom van orde 5 blijven.
Geef via tussenresultaten en begeleidende tekst duidelijk aan hoe u aan uw antwoord komt.

3.24 Dit was opgave 5 bij het tentamen Algoritmie van 11 juni 1993.

- a**
- i. Hoeveel sleutels kan een knoop in een B-boom van orde 6 maximaal bevatten? En hoeveel minimaal?
 - ii. Hoe hoog kan een B-boom van orde 6 met zes knopen minimaal zijn? En hoe hoog maximaal? (Leg uit!)
 - iii. Hoeveel sleutels kan een B-boom van orde 6 met zes knopen minimaal bevatten? En hoeveel maximaal? (Leg uit!)
- b** Gegeven de volgende B-boom van orde 6:



Welke boom ontstaat als we hieruit achtereenvolgens de getallen 51, 60 en 85 verwijderen? (Verwijderen dient te geschieden op de gebruikelijke wijze, dus via verwisselen met grootste kleinere, etc.!) De boom moet uiteraard steeds een B-boom van orde 6 blijven.
Geef duidelijk - via tussenresultaten en begeleidende tekst - aan hoe je aan je antwoord komt.

- c** In C++ kan een B-boom van orde m (die verschillende gehele getallen bevat) als volgt gerepresenteerd worden:

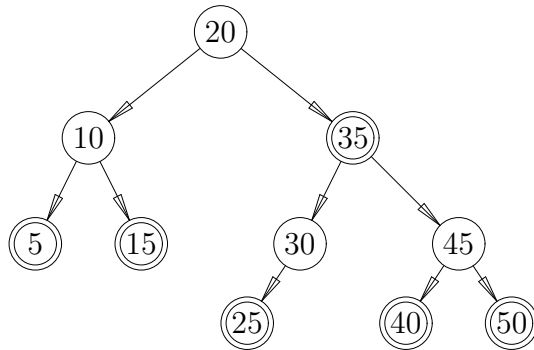
```

struct BKnoop;
typedef BKnoop *BWijzer;
struct Element
{
    int Info;
    BWijzer Wijzer;
};
struct BKnoop
{
    int Aantal; // kan uiteindelijk alleen
               // waarden 1 ... m-1 aannemen
    BWijzer Vader, Links;
    Element Inhoud[m-1];
};
  
```

Zij een pointer Kind van type BWijzer gegeven, die wijst naar een knoop in een B-boom met daarin het minimale aantal sleutels dat een knoop mag bevatten. Zij gegeven dat Kind een rechterbuur heeft die meer dan het minimaal toegestane aantal sleutels bevat. Schrijf nu een stukje C++-tekst dat de meest rechtse sleutel uit de knoop waar Kind naar wijst, verwijdert en daarbij leent bij zijn rechterbuur. De betreffende knoop zit dus op het onderste niveau.

3.25 Dit was opgave 2 bij het tentamen Datastructuren van 3 januari 2002.

- a** Wat zijn de kenmerkende eigenschappen van de rood-zwart boom (*red-black tree*)? Deze boom is op het college gepresenteerd als implementatie van de B-boom van orde $m = 4$.
- b** Gegeven is de volgende rood-zwart boom, waarbij ‘rode’ knopen een extra cirkel hebben gekregen.



Welke boom ontstaat als we hieraan toevoegen

- i. eerst 55 en vervolgens 7.
- ii. eerst 7 en vervolgens 55.

Benoem de achtereenvolgende operaties en geef relevante tussenresultaten.

- c** Hoeveel knopen (rode en zwarte samen) heeft een rood-zwart boom van hoogte h maximaal? En minimaal?

De hoogte van de boom wordt hier gemeten in het aantal knopen op het langste pad van wortel naar blad.

Voor het minimale aantal hoeft u alleen even [óf oneven, naar keuze] hoogtes te bespreken.

4 Tabellen

- 4.1** Neem aan dat we een dubbel verbonden lijst hebben waarin de sleutels geordend staan. Rechtstreeks toegang tot de lijst hebben we slechts via het eerste en laatste element. Bekend is verder het aantal elementen van de lijst (neem voor het gemak aan dat dit aantal van de vorm $2^k - 1$ is). Hoeveel stappen heeft de binaire zoekmethode voor deze datastructuur (in het slechtste geval) nodig?
- 4.2** Ongeveer duizend boektitels dienen opgeborgen te worden in een tabel mbv. een hash-functie.
- We stellen voor om het adres van een titel te bepalen door de ASCII waarden van de eerste vijf letters bij elkaar op te tellen. Is dit een verstandige keus?
 - Doe zelf een voorstel voor een hash-functie. Bespreek de gevaren voor primaire en secundaire clustering.
- 4.3** Uitgaande van een alfabet van 32 letters en een tabelgrootte M kiezen we als hash-adres voor de string $X_n \dots X_1 X_0$ de waarde $X_n \cdot 32^n + \dots + X_1 \cdot 32 + X_0 \bmod M$. We identificeren hierbij letters met de getallen $0, \dots, 31$ zoals in C gebruikelijk.
- Tot welke waarde van n is $X_n \cdot 32^n + \dots + X_1 \cdot 32 + X_0$ nog direct als `int` te berekenen (in uw eigen favoriete compiler)?
 - Hoe kan ook voor lange strings de waarde $X_n \cdot 32^n + \dots + X_1 \cdot 32 + X_0 \bmod M$ nog gemakkelijk bepaald worden?
- 4.4** Om het probleem van botsingen bij hashen aan te pakken stellen we een methode voor die in het midden houdt tussen *chaining* en open adressering. We voorzien elk tabelelement (behalve van een sleutel en bijbehorende informatie) van een index verwijzend naar een ander tabelelement. We gebruiken deze index om –bij botsingen– lijsten in de tabel zelf aan te leggen. Verder houden we een index bij die wijst naar de onderste lege plek in de tabel; hier wordt de eerstvolgende nieuwe sleutel gezet.
- Geef een algoritme dat in de beschreven structuur zoekt en invoegt. Neem als sleutels positieve gehele getallen; gebruik 0 om lege plaatsen aan te geven.
 - Bovenstaande methode heeft als nadeel dat lijsten door elkaar heen kunnen gaan lopen. Dit kan voorkómen worden door bij het starten van een nieuwe lijst op een hash-adres het daar eventueel aanwezige element uit een andere lijst te verhuizen naar een vrije locatie. Daartoe dient ook de index die naar dit element verwijst aangepast te worden.

Het blijkt handig te zijn om cyclische lijsten aan te leggen en het begin van elke lijst te markeren met een daarvoor beschikbare Boolean. Pas het algoritme uit **a** aan.
- 4.5** Een functie $q(K)$ heet een *aanvulling* van de adres-functie $h(K)$ als het mogelijk is K te berekenen uitgaande van de waarden $h(K)$ en $q(K)$.
- Merk op dat de functie $q(K) = K$ een aanvulling voor elke adres-functie vormt. Geef 'redelijke' aanvullingen voor adres-functies verkregen door deling ($h(K) =$

$K \bmod M$) of door bit-extractie ($K = b_1 \dots b_r$, $h(K) = b_{i_1} \dots b_{i_t}$, $t < r$ voor vaste i_1, \dots, i_t).

b Laat zien dat we voor het algoritme uit **b** van bovenstaande opgave kunnen volstaan met het opslaan van een aanvulling $q(K)$ in plaats van de volledige sleutel. Lukt dit ook voor de versie uit **a**. (waar lijsten kunnen gaan overlappen)? Laat zien dat voor adres-functies met een 'redelijke' aanvulling deze beperkte opslag de voor de indexen gebruikte ruimte (vrijwel) compenseert.

4.6 De zoekmethoden *lineair*, *kwadratisch* en *random* hashen kunnen allen beschreven worden door een simpele functie $h(K, i)$ die de plek aangeeft waar de sleutel K in de $(i + 1)$ -ste stap gezocht wordt.

a Geef deze functie voor elk van de drie methoden. Gebruik hierbij een adres-functie $h(K)$.

Om clustering te vermijden willen we graag dat het onwaarschijnlijk is dat twee sleutels die in een zoekstap op hetzelfde adres terecht komen dat ook in de volgende zoekstap doen. We willen dus graag dat $h(K_1, i) = h(K_2, j)$ en $h(K_1, i + 1) = h(K_2, j + 1)$ niet al te vaak tegelijkertijd voorkomen (dwz. niet meer dan verwacht wanneer de adressen geheel willekeurig gekozen zouden worden).

b Onderzoek het tegelijkertijd vóórkomen van beide gelijkheden voor de drie hash-methoden. Wanneer gebeurt dit?

c Probeer een permutatie aan te geven voor een tabel met 10 elementen, zó dat de verschillen $h(K, i + 1) - h(K, i)$ steeds ongelijk zijn (modulo 10) bij ongelijke i .

4.7 Bewijs dat de eis $(p(K), M) = 1$ voor een stapfunctie p bij een tabelgrootte M essentieel is om alle adressen te kunnen bereiken.

4.8 Zij T een tabel, van type `int T[M]`.

In de tabel moet een rij van $N (< M)$ positieve gehele getallen opgeborgen worden.

a Beschrijf de methode die gebruik maakt van een dubbele hash-functie.

b Laat $M = 25$. Doe een voorstel voor de benodigde hash-functie(s).

c Welke tabel ontstaat als de voorgestelde methode wordt toegepast bij de volgende rij getallen :

41, 62, 13, 84, 35, 96, 57, 28, 79, 53, 19, 88, 27, 10, 2, 38, 66, 40, 90, 6.

4.9 In de tabel T van type `int T[17]` wordt een rij van 12 willekeurige positieve getallen (niet groter dan 100) opgeslagen met behulp van een dubbele hash-functie. Voorgesteld worden de functies $h(k) = k \bmod 17$ en $p(k) = (k \bmod 16) + 1$.

a Is er bij deze keuze gevaar voor primaire of secundaire clustering? Geef eventueel verbeteringen.

- b** Welke tabel ontstaat als deze (evt. verbeterde) hash-functies gebruikt worden om achtereenvolgens aan de tabel toe te voegen

41, 84, 57, 53, 27, 38, 90, 24, 50, 62, 13 en 96.

- 4.10** Zij gegeven een tabel T van type `int T[49]`. In deze tabel worden positieve gehele getallen opgeborgen mbv. de hash-functie $h(k) = k \bmod 49$ en één van de volgende stapfuncties

$$p_1(k) = 3,$$

$$p_2(k) = (k \bmod 6) + 1,$$

$$p_3(k) = (k^2 \bmod 7) + 1.$$

- a** Beschouw de rij getallen 50, 33, 36, 1, 99, 197, 54, 5.

Gebruik elk van de gegeven stapfuncties om de getallen in T op te bergen. Welke tabel ontstaat in elk van de drie gevallen?

- b** Bespreek de verschillende stapfuncties.

- Zijn ze toelaatbaar (zijn alle bezochte adressen verschillend)?
- Krijgen we lineair hashen? En dubbel hashen?
- Is er gevaar voor primaire of secundaire clustering?

- 4.11** Een tabel met 7 elementen wordt gevuld met behulp van de hash-functies $h(K) = K \bmod 7$ en $p(K) = (K \bmod 5) + 1$.

- a** Plaats in deze tabel de getallen 23, 46, 30, 20, 39 en 45.

- b** Vergroot de tabel tot 13 elementen. Herplaats de getallen uit de tabel (*ter plekke*) met lineair hashen (stapgrootte 1).

- 4.12** Voor welk type hashen is het algoritme voor ter plekke opnieuw hashen geschikt? Is het algoritme ook geschikt te maken voor andere vormen van hashen?

- 4.13 a.** Aan welke voorwaarde moet een stapfunctie (tweede hashfunctie) beslist voldoen (waarom)? Wanneer spreken we van *secundaire* clustering?

- b** Positieve gehele getallen kleiner dan 100 worden opgeborgen in een tabel `int T[11]` gebruik makend van een adresfunctie $h(K) = K \bmod 10$ (de tabelgrootte is elf!). Voorgesteld worden de stapfuncties

$$p_1 = \begin{cases} K \bmod 5 & \text{als } K \text{ geen 5-voud is,} \\ 7 & \text{anders.} \end{cases}$$

$$p_2(K) = (K \text{ div } 10) + 1.$$

Zijn p_1 en p_2 toelaatbaar? Is er sprake van dubbel hashen (met onafhankelijke hash-functies)? Is er gevaar voor secundaire clustering?

- c** Gekozen wordt voor de stapfunctie p_2 . Vul de tabel met de getallen 48, 36, 80, 40, 11, 64, 22, 18, 93 en 04 (in deze volgorde). Geef hierbij aan op welke plaatsen elk van de getallen geprobeerd wordt.

d Omdat de tabel nu vol is worden de sleutels (*ter plekke*) in een grotere hashtabel geplaatst. De nieuwe tabel bevat 15 plaatsen en wordt gevuld via *lineair* hashen (stapgrootte 1) met als adres de sleutelwaarde modulo 15. Laat duidelijk zien hoe dit gebeurt. Geef weer aan op welke plaatsen elk van de getallen geprobeerd wordt.
(tentamen januari 89)

4.14 a. We beschouwen een hash-tabel die gevuld is door de sleutels van groot naar klein toe te voegen. Laat met een voorbeeld zien dat de sleutels niet langs elk zoekpad van groot naar klein hoeven te liggen.

b Bewijs dat wél geldt dat de sleutels die gevonden worden tijdens het zoeken naar sleutel K (die in de tabel voorkomt) alle groter zijn dan K .

4.15 Bekijk een tabel `int T[5]` en de hash-functies $h(k) = k \bmod 5$ en $p(k) = 1$. We nemen aan dat de aangeboden waarden voor k uniform verdeeld zijn in $[1..1000]$.

a Wat is de kans op botsing ná het opbergen van het eerste getal?

b Wat is de kans dat bij het opbergen van drie getallen in een aanvankelijk lege tabel gebruik gemaakt moet worden van $p(k)$?

4.16 Bepaal, om een beter inzicht te krijgen in de verschillen tussen de diverse hash-methoden voor niet al te volle tabellen, de eerste vier termen van de reeksontwikkeling (rond 0) van de formules voor de verwachte aantallen zoekstappen. Gebruik hierbij dat voor $|\alpha| < 1$,

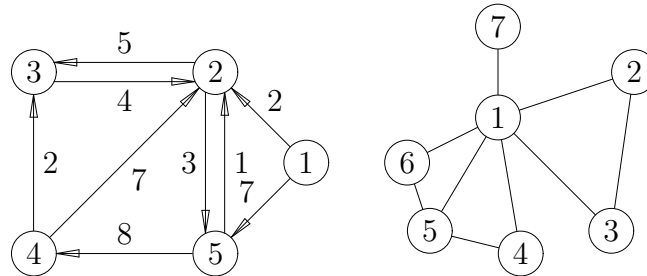
$$\ln(1 - \alpha) = -\alpha - \frac{1}{2}\alpha^2 - \frac{1}{3}\alpha^3 - \frac{1}{4}\alpha^4 - \dots,$$

$$\frac{1}{(1 - \alpha)} = 1 + \alpha + \alpha^2 + \alpha^3 + \alpha^4 + \dots \text{ en}$$

$$\frac{1}{(1 - \alpha)^2} = 1 + 2\alpha + 3\alpha^2 + 4\alpha^3 + 5\alpha^4 + \dots$$

5 Grafen

5.1 Geef van onderstaande grafen een *adjacency matrix* en een *adjacency list* representatie.



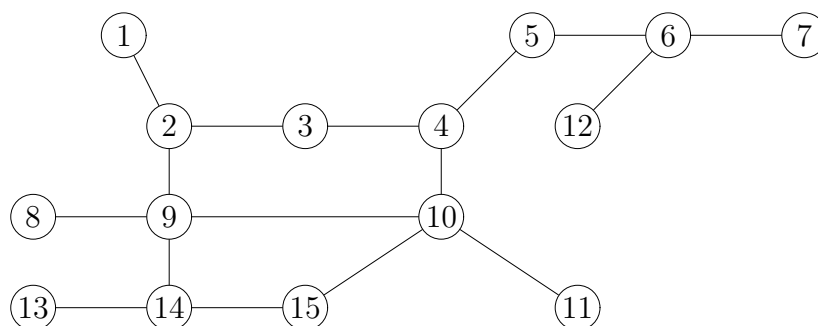
5.2 Bepaal van deze grafen tevens de opspannende bomen voor de *breadth-first* en *depth-first* wandelingen. Geef ook de volgorde aan waarin de knopen bezocht worden.

5.3 Beschouw de volledige graaf op n knopen. Hoe ziet de opspannende boom van deze graaf er uit voor elk van de beide graafwandelingen?

5.4 Implementeer de *depth-first* graafwandeling met behulp van een stapel (dus zonder gebruik te maken van recursie).

5.5 a. Door bij *depth-first search* niet gewoon de nieuwe knopen op de stapel te zetten, maar ze steeds te voorzien van informatie die hun 'ouder' aangeeft (dwz. de knoop die bezocht werd toen de nieuwe knoop op de stapel kwam) kunnen we op eenvoudige wijze uitprinten hoe de *dfs*-opspannende boom van de graaf er uitziet. Leg dit uit.

b. Bekijk onderstaande ongerichte graaf. Geef de volgorde aan waarin de knopen bij een *depth-first* wandeling bezocht worden. Welke knopen staan er op de stapel wanneer we knoop 9 bezoeken? Welke knopen zijn de ouders van de knopen op de stapel?



c. Hoe kan het *dfs*-algoritme aangepast worden om te testen of een ongerichte graaf een kring bevat (dwz. geen boomstructuur heeft)?

Hoe kunnen we zo'n kring ook op eenvoudige wijze afdrukken?

5.6 Schrijf een algoritme dat, voor een gegeven knoop in een graaf, een knoop op maximale afstand in de graaf oplevert. Hierbij meten we de afstand tussen twee knopen als het minimale aantal takken van paden tussen beide knopen.

5.7 Geef een algoritme dat, voor een gegeven gerichte graaf, bepaalt of er een knoop is van waaruit alle andere knopen bereikbaar zijn, en — als dit het geval is — deze knoop uitvoert.

5.8 a. Werkt het algoritme van Prim ook wanneer in de graaf negatieve gewichten voorkomen ?

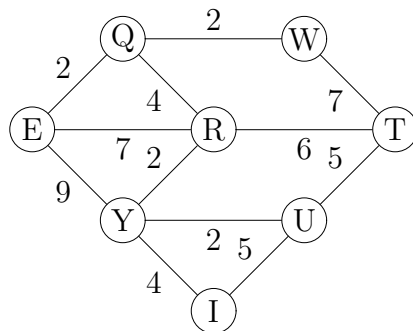
b. Wanneer een graaf een kring met een negatief totaal gewicht bestaat, dan is het kortste pad tussen knopen niet altijd netjes gedefinieerd. Als we wél negatieve gewichten toelaten, maar eisen dat er geen negatieve kringen ontstaan, geeft Dijkstra's kortste paden algoritme dan altijd het juiste resultaat ?

5.9 Wanneer volgens het algoritme van Prim een minimale opspannende boom van een graaf bepaald wordt 'groeit' deze boom vanuit de startknoop. Er bestaat een nog *gretiger* algoritme voor het bepalen van een minimale opspannende boom: Voeg telkens een tak van minimaal gewicht toe die geen kring vormt met de eerder gekozen takken. Dit heet het algoritme van *Kruskal*.

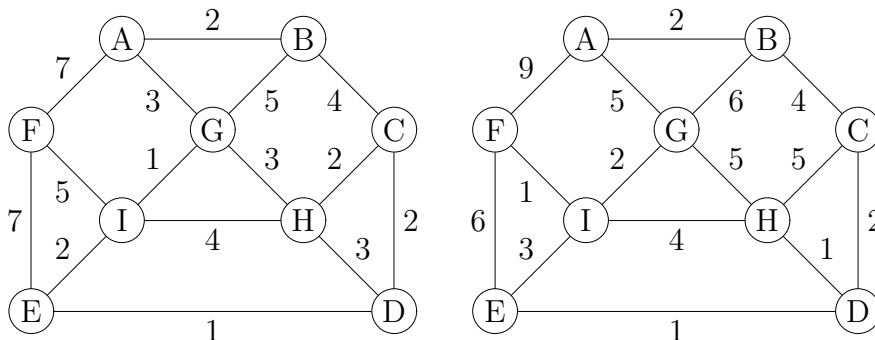
a. Pas dit algoritme toe op onderstaande graaf.

b. Bewijs dat dit algoritme correct is.

c. (‡) Probeer dit algoritme te implementeren. Op welke punten wijkt de implementatie af van die voor Prim's algoritme? In gunstige of ongunstige zin ?



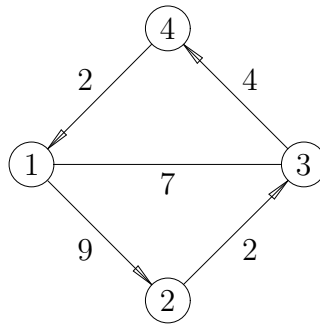
5.10 Bepaal van onderstaande grafen de minimale opspannende boom (algoritme van Prim) respectievelijk de kortste paden (Dijkstra) uitgaande van knoop A.



5.11 (‡) Een *Euler wandeling* is een kring in de graaf die alle takken bevat. Geef een algoritme dat een Euler wandeling bij een gegeven (ongerichte) graaf bepaalt (zo

deze bestaat). Bij Discrete Wiskunde wordt een karakterisering voor grafen met een Euler wandeling bewezen: een ongerichte graaf heeft een Euler wandeling indien elke knoop een even graad heeft (en de graaf verbonden is).

- 5.12 (‡) De *diameter* van een (ongerichte) boom is de maximale afstand tussen twee knopen in de boom. Geef een algoritme dat bij een gegeven ongerichte graaf een opspannende boom van minimale diameter bepaalt. Bewijs dat het gegeven algoritme correct is.
- 5.13 Pas het algoritme van Floyd (alle afstanden in de graaf) zó aan dat ook daadwerkelijk de kortste paden op eenvoudige wijze bepaald kunnen worden. *Aanwijzing.* Gebruik een matrix Pad waarvan element $\text{Pad}[i, j]$ aangeeft of het kortste pad van i naar j via de tak (i, j) loopt, danwel via een andere knoop. Schrijf een (recursieve?) procedure die aan de hand van deze matrix de kortste paden af kan drukken.
- 5.14 Pas het algoritme van Floyd toe op onderstaande graaf. Geef de tussenresultaten.



- 5.15 a. Zij G een gerichte graaf (zonder gewichten). De transitieve afsluiting van G is de graaf met dezelfde knopen als G , maar waarin een tak (i, j) te vinden is precies wanneer er in G een *wandeling* van i naar j is. Geef een algoritme dat de transitieve afsluiting van een gegeven gerichte graaf berekent.
- b. (‡) Idem voor ongerichte grafen. Kan het algoritme hiervoor efficiënter gemaakt worden?
- 5.16 Gebruik het algoritme voor topologisch sorteren om te bepalen of de (gerichte) graaf met onderstaande takken een kring bevat.

$(9, 2), (3, 7), (7, 5), (5, 8), (8, 6), (4, 6), (1, 3), (7, 4), (9, 5), (2, 8)$

- 5.17 Implementeer topologisch sorteren in C++. Gebruik een array om voor elke knoop het aantal inkomende takken aan te geven. Laat zien hoe we *ditzelfde* array kunnen gebruiken om een lijst aan te leggen die alle knopen zonder inkomende takken bevat.
- 5.18 Een *netwerk* is een gerichte graaf met gewichten en een speciale begin- en eindknoop. De takken van zo'n netwerk geven projecten aan en het bijbehorende gewicht de tijdsduur van het project. In zo'n netwerk komen geen kringen voor. De graaf geeft de afhankelijkheden tussen de projecten weer. Een project (i, j) kan pas gestart worden als alle projecten die eindigen in knoop i beëindigd zijn.

a. Hoe bepalen we voor elke knoop i het (vroegste) moment waarop de projecten die uitgaan van knoop i gestart kunnen worden? We nemen hierbij aan dat de projecten die van de beginknoop uitgaan op tijdstip nul starten.

b. In sommige projecten kan vertraging ontstaan zonder dat dit effect heeft op de aanvangstijd van de andere projecten. Hoe kan bij elke knoop het laatste moment bepaald worden waarop alle inkomende projecten moeten eindigen zonder dat dit vertraging oplevert?

c. Geef aan hoe projecten gevonden kunnen worden waar beslist geen vertraging opgelopen mag worden zonder dat dit de totale benodigde tijd beïnvloedt. Deze projecten worden *kritiek* genoemd.

5.19 Voor een aantal projecten geven we *naam*, *tijdsduur* en projecten die *eerder uitgevoerd moeten zijn* vóórdat we het betreffende project mogen beginnen.

Naam:	A	B	C	D	E	F	G	H	I	J	K	L	M	N
Duur:	3	2	3	1	3	3	2	3	5	1	2	2	7	4
Na:	-	A	B	C	B	B	F,H	A	A	H	D,E,G,J	I	I	K,L,M

Bepaal de totale tijdsduur benodigd voor het uitvoeren van de projecten. Welke projecten zijn 'kritiek' ?

6 Patroonherkenning

6.1 Bepaal de *failure-links* volgens de methode van Knuth-Morris-Pratt voor het herkennen van elk van de volgende strings.

a. abaaba

b. aaab

c. aabaacaabaababa

d. aaaabc

6.2 a. Bij het uitvoeren van het algoritme van KMP kan het voorkomen dat de *failure-link* van plaats k wijst naar plaats r , waarbij $P_k = P_r$. Wat is het effect bij het uitvoeren van het algoritme, dwz. bij het zoeken van een patroon in de tekst?

b. Geef een algoritme dat, nádat de *failure-links* zijn opgesteld, de *failure-links* zo aanpast dat bovenstaande situatie niet meer voorkomt.

6.3 (‡) Een gerichte graaf met speciale begin- en eindknoten, en waarvan alle pijlen voorzien zijn van een letter heet wel een *eindige automaat* (colleges IFI en FTA1). Ook bij herkenning van één enkel patroon kan van zo'n graaf handig gebruik gemaakt worden. De knopen representeren weer posities in het patroon; de automaat beschrijft de handelingen van het zoekalgoritme. Het algoritme volgt een wandeling door de graaf, waarbij de letter langs de pijl aangeeft welke pijl gevolgd gaat worden voor de volgende letter uit de invoer tekst.

a. Geef aan hoe zo'n eindige automaat gevonden kan worden uit de *failure-links* van het KMP algoritme.

b. Bepaal eindige automaten voor elk van de patronen uit opgave 1.

Voorbeeld. Voor het patroon ABCABABC ziet de eindige automaat er als volgt uit.

Startknoop = 1, Eindknoop = 9 (klaar, tekst gevonden). Onderstreepte knopen geven 'succesvolle vergelijkingen' aan.

	1	2	3	4	5	6	7	8	van
label A	<u>2</u>	2	2	<u>5</u>	2	<u>7</u>	2	2	naar
B	1	<u>3</u>	1	1	<u>6</u>	1	<u>8</u>	1	
C	1	1	<u>4</u>	1	1	4	1	<u>9</u>	

6.4 Hoeveel lettervergelijkingen worden gedaan bij het bepalen van de *failure-links* voor het patroon $P = a \dots ab?$ ($m - 1$ a's gevolgd door een b).

Idem voor $P = a \dots abc$ ($m - 2$ a's gevolgd door bc).

6.5 a. Breid het algoritme van Knuth-Morris-Pratt uit tot de *wild-card* *, dwz. op de plaats van * mag een willekeurig aantal (eventueel nul) symbolen gelezen worden.

b. (‡) Idem voor de *wild-card* ?, die aangeeft dat één willekeurig symbool in de tekst mag voorkomen.

- c. (‡) Geef de *failure-links* voor de patronen aa^*baab en $aa?baab$. Bepaal ook de bijbehorende eindige automaten.
- 6.6** (‡) Leg uit hoe het leggen van *failure-links* uit te breiden is tot zoekacties in een tekst waarbij een eindig aantal patronen gegeven is.
- 6.7** De methode van Knuth-Morris-Pratt wordt gebruikt om een patroon $P[1..m]$ in een tekst $T[1..Lengte]$ te zoeken. Daartoe worden *failure-links* opgesteld.
- a. Stel de *failure-link* bij positie i in P wijst naar positie k in P : $Flink[i] = k$. Wat geldt dan voor de woorden $P[1] \dots P[i]$ en $P[1] \dots P[k]$? In het bijzonder, wanneer geldt $Flink[i] = 0$?
- b. Bepaal de *failure-links* voor het patroon $ABCABCABBC$.
- c. Het kan voorkomen dat de *failure-link* van positie i wijst naar k , waarbij bovendien de letters $P[i]$ en $P[k]$ gelijk zijn. Welke consequentie heeft dat bij het zoeken? Beschrijf een algoritme dat, nádat de standaard *failure-links* zijn opgesteld, deze aanpast zodat de zojuist beschreven situatie niet voorkomt.
- d. (‡) Bij een *eindige automaat* representatie van het zoek-patroon geven we voor elke letter op elke plaats in het patroon aan waar het zoeken vervolgd moet worden. Daarbij wordt dus niet alleen onderscheid gemaakt tussen *goed* en *fout*. Geef zowel een mogelijk voordeel als een mogelijk nadeel van deze representatie.
(tentamen januari 89)
- 6.8** Geef reguliere expressies voor de woorden (met alleen 0-en en 1-en)
- a. waarin elke 0 door een 1 gevolgd wordt.
- b. waarin een even aantal 1-en voorkomt.
- 6.9** Hoe reageert het ontledingsalgoritme op uitdrukkingen als $(A+B)^*BC+$, $(A+B)^**A$, $(A+(B+C))$, en $A+B+*$?
- 6.10** a. Probeer zoveel mogelijk de recursie te verwijderen uit het algoritme dat expressies in automaten omzet.
- b. (‡) Vergelijk de *run-tijden* van beide algoritmes (probeer bv. eens uitdrukkingen als $A+B+C+ \dots +Z$. Breng verslag uit aan het management.
- 6.11** De voorgestelde notatie mbv. *reguliere expressies* is nogal omslachtig als we daarmee een willekeurige letter willen uitdrukken; probeer maar eens een expressie op te stellen voor alle woorden (met letters A tm. Z) waarvan de derde letter een A is.
- a. Breid de reguliere expressies uit met een notatie voor 'willekeurige letter'. Druk hiermee het hierboven omschreven patroon uit.
- b. Onderzoek welke veranderingen deze uitbreiding noodzakelijk maakt in de BNF-beschrijving van reguliere expressies, de ontleding van expressies, in de representatie van eindige automaten en in het algoritme dat uitzoekt of een gegeven woord door een geconstrueerde automaat gerepresenteerd wordt.

6.12 Na de verbetering voorgesteld in bovenstaande opgave blijven de uitdrukkingen voor 'woorden die beginnen met één der letters A...R' en 'woorden die *niet* eindigen op Z' nog tamelijk omvangrijk.

a. Doe een voorstel voor een uitbreiding van reguliere expressies.

b. Herhaal onderdeel **b.** van de vorige opgave.

6.13 (‡) De operaties Ster, Plus en Conc (zoals in het theoriedictaat geïmplementeerd) voeren overdadig veel takken zonder letters in. Onderzoek of dit wel zo noodzakelijk is. Het zou bv. aantrekkelijk zijn als nutteloze takken uit de representatie van termen als **ABABAB** zouden verdwijnen, zonder dat dit ten koste gaat van de overzichtelijkheid van representatie en algoritmes.

6.14 a. Ontleed de expressie $(AAA + AB^*A)^*$; geef de afleidingsboom.

b. Construeer een automaat die deze expressie representeert; geef de bijbehorende semantische boom. Elk van de knopen in deze boom correspondeert met een gedeelte van de uiteindelijke automaat. Geef steeds de start- en eindknopen aan.

c. Controleer voor elk van de volgende woorden of zij aan de expressie voldoen: AAAAA, ABBAAAA, AABAABA, ABAABA.

6.15 a. Geef van elk van de volgende reguliere expressies de structuur mbv. een boom weer. (De interne knopen van deze boom worden voorzien van één van de symbolen \cdot , $+$ of $*$, de bladeren van een letter.)

i. $(AA^*+B^*)^*$ ii. $(A+B)(AA+B)^*$ iii. ABC^*A+BC

b. Geef, voor elk der bovenstaande expressies, de postfix notatie en een string (van tenminste 5 letters) die aan de expressie voldoet.

c. Construeer voor de expressie $(AA^*+B^*)^*$ de bijbehorende eindige automaat. Nummer de knopen van deze automaat in de volgorde waarin ze tijdens de postorde wandeling door de boom aan de automaat worden toegevoegd. Teken nogmaals de boom die bij de expressie hoort, en geef bij elke knoop van de boom de start- en eindknoop van het corresponderende deel van de eindige automaat.

d. Laat zien dat de string **AAB** aan de expressie $(AA^*+B^*)^*$ voldoet, door het uitvoeren van een complete wandeling door de graaf. (Het geven van één passend pad is als antwoord niet voldoende.) Geef in ieder geval voor elk van de letters van **AAB** de knopen die vóór, tijdens en ná het lezen van die letters tijdens de wandeling bezocht worden duidelijk aan.

(tentamen december 1990)

6.16 a. Geef voor beide onderstaande reguliere expressies de bijbehorende boomstructuur én postfix notatie.

i. $(A^*+AAB)^*B$ ii. $(AB+BA+BB^*B)^*$

b. Uitgaande van de boomstructuur van een expressie kan op systematische wijze een eindige automaat geconstrueerd worden die dezelfde strings representeert. Leg kort uit hoe dit gebeurt.

c. Met behulp van de graafrepresentatie gegeven door de eindige automaat kunnen we verifiëren of een gegeven woord aan de expressie voldoet door op systematische manier door de graaf te wandelen. Geef een korte uitleg.

Voor de eerder gegeven expressie $(A^*+AAB)^*B$ wordt de volgende automaat gevonden: Startknoop 11, Eindknoop 14.

Knoopnr	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Volg1	2	1	11	5	6	7	8	9	11	2	12	13	14	-
Volg2	-	3	-	-	-	-	-	-	-	4	10	-	-	-
Letter	A	-	-	A	-	A	-	B	-	-	-	-	B	-

d. Ga na of de string AAAB aan de gegeven expressie voldoet, door het uitvoeren van een wandeling door de automaat. Geef deze wandeling schematisch in een boomstructuur weer. Hierbij dienen ook 'doodlopende' paden opgenomen te zijn.

e. Idem voor de string AAAA. (Het zoekproces voor de eerste drie letters is gelijk aan dat bij onderdeel d. Er kan dus volstaan worden met het geven van de wandeling vanaf de derde letter.)

(tentamen december 1991)