Uitwerkingen bij voorbeeldtentamen 1 **horend bij editie 6 van het tekstboek**

OPGAVE 1

Bekijk onderstaand algoritme recAlg.

5 punten

a Bepaal recAlg(5) en laat zien hoe u het antwoord hebt verkregen.

5 punten

b Wat berekent recAlg in het algemeen? Verklaar uw antwoord.

5 punten

c Bepaal de complexiteit van recAlg. Licht uw antwoord toe.

5 punten

d Breng een kleine verandering aan in recAlg zodanig dat deze efficiënter wordt. Bepaal de complexiteit van de veranderde recAlg. Licht uw antwoord toe.

```
Algorithm recAlg(n):
Input: Integer n
Output: Integer

result ← 1
  if n > 1 then
    result ← 1 + recAlg(n - 1) + recAlg(n - 1)
  { end if }
return result
```

UITWERKING OPGAVE 1

a Het eenvoudigste is om achteraan te beginnen.
recAlg(1) = 1
recAlg(2) = 1 + 2*1 = 3
recAlg(3) = 1 + 2*3 = 7
recAlg(4) = 1 + 2*7 = 15
recAlg(5) = 1 + 2*15 = 31

b recAlg(n) = $2^n - 1$
Dit is in te zien door de uitwerking van a te herschrijven:
recAlg(1) = 1 = $2^0$
recAlg(2) = $2^0 + 2*2^0 = 2^0 + 2^1$
recAlg(3) = $2^0 + 2*(2^0 + 2^1) = 2^0 + 2^1 + 2^2$
recAlg(4) = $2^0 + 2*(2^0 + 2^1 + 2^2) = 2^0 + 2^1 + 2^2 + 2^3$
… (eigenlijk met volledige inductie te bewijzen)
recAlg(n) = $2^0 + \ldots + 2^{n-1} = 2^n - 1$

c Het aantal recursieve aanroepen verdubbelt steeds en is bij recAlg(n) gelijk aan $2^{n-1}$.
Per aanroep hebben we 1 toekenning of 1 toekenning en 2 optellingen, dus altijd O(1), en dus is het totaal aantal primitieve bewerkingen evenredig met $2^{n-1}$.
De complexiteit is dus $O(2^n)$ en zelfs $\Theta(2^n)$ omdat er geen sprake is van een verschil tussen een slechtste en beste geval.

d Vervang result ← 1 + recAlg(n - 1) + recAlg(n - 1)
door result ← 1 + 2 * recAlg(n - 1).
Nu is de complexiteit $\Theta(n)$ want het aantal recursieve aanroepen bij recAlg(n) is gelijk aan n - 1.

OPGAVE 2

Deze opgave bestaat uit 2 onafhankelijke deelopgaven

8 punten

a Maak opgave C-7.30 uit het tekstboek

7 punten

b Maak opgave C-7.33 uit het tekstboek

UITWERKING OPGAVE 2

a

```
public Position<E> positionAtIndex(index i) {
  if (i < 0 | | i >= size( )) throw new
            IndexOutOfBoundsExcpetion("Invalid index");
  Position<E> curr = first( );
  for (int k = 0; k  <  i; k++)
      curr = after(curr);
  return curr;
}
```

b

```
public void moveToFront(p){
  E element = p.getElement();
  addFirst(element);
  remove(p);
}
```

OPGAVE 3

10 punten

Waar kan in een heap het element met de grootste sleutel staan? Licht uw antwoord toe.

UITWERKING OPGAVE 3

In een heap geldt de eis dat elk knooppunt m.u.v. de wortel een grotere of dezelfde sleutel bevat als zijn ouder.
Dus bij allemaal verschillende sleutels zal de grootste alleen in een blad kunnen staan.
Als de grootste sleutel echter meermaals voorkomt, kan deze ook in een intern knooppunt staan, mits er onder alleen maar duplicaten voorkomen.
Alleen als alle sleutels gelijk zijn, kan de 'grootste' in de wortel staan.

OPGAVE 4

Gegeven is de rij *S* 5 16 22 45 2 10 18 30 50 12 1

5 punten

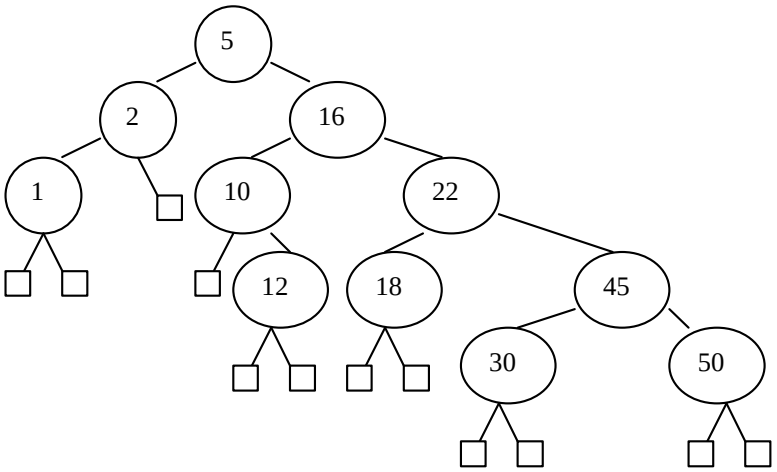
a Voeg de getallen van *S* in de gegeven volgorde (eerst 5, dan 16, …) toe aan een lege binaire zoekboom en teken het eindresultaat.

10 punten

b Voeg de getallen van *S* in de gegeven volgorde toe aan een lege AVL-boom en teken het tussenresultaat direct voor en na elke rotatie en teken het eindresultaat.

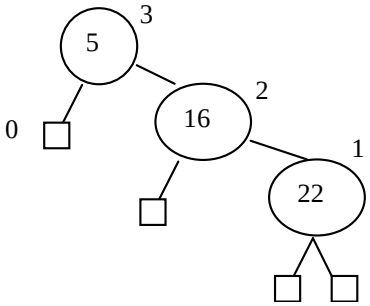
UITWERKING OPGAVE 4

a

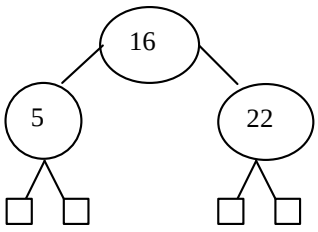



b

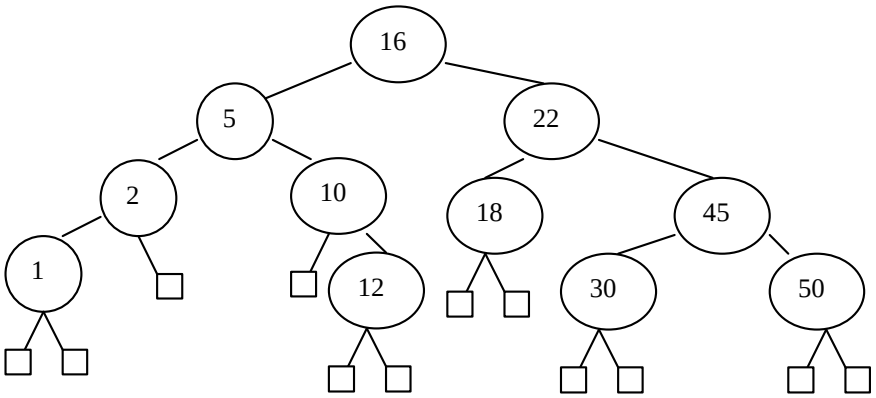
Na toevoeging van de eerste drie is het hoogteverschil 2:




Enkele rotatie:




Verder geen rotaties meer nodig, dus meteen het eindresultaat:




OPGAVE 5

10 punten

Een hash-tabel is niet geschikt is om een geordende map te realiseren. Licht dit toe.

UITWERKING OPGAVE 5

Een hash-tabel is geschikt voor de realisatie van een ongeordende map (zie blz 378 ev).
We hoeven dus alleen te laten zien dat een hash-tabel niet geschikt is voor realisatie van de methoden die specifiek zijn voor geordende maps zoals genoemd op blz 396. Uiteraard gaan we er van uit dat de sleutels geordend kunnen worden, anders gezegd dat ze een lineair geordend verzameling vormen.
In een hash-tabel wordt een item opgeslagen door op de sleutel de hash-functie toe te passen die de hash-code als functiewaarde geeft. Ook al zijn de sleutels geordend, hun hash-codes zijn dat in het algemeen niet, dus een hash-tabel is een ongeordende tabel.
Als nu een methode firstEntry of lastEntry gerealiseerd moet worden, betekent dat dat de gehele hash-tabel doorzocht moet worden en dat is zeer inefficiënt. Hetzelfde geldt voor de andere methoden.

OPGAVE 6

8 punten

a We passen BFS toe op de gerichte graaf van figuur 14.11 (a) op blz. 607 van het tekstboek.
Kies een knooppunt van waaruit de gehele graaf wordt doorlopen en geef een volgorde van de knooppunten waarin deze bezocht kunnen worden door BFS.

7 punten

b Geef de verschillende soorten kanten (zowel tree edges als nontree edges) die door BFS worden benoemd.

UITWERKING OPGAVE 6

a De knooppunten worden in geval van BFS niveau voor niveau nagelopen.
Als we beginnen met knooppunt BOS, dan is dat het $0^e$ niveau.
In het $1^e$ niveau vinden we dan JFK en MIA
Het $2^e$ niveau levert SFO , DFW en LAX op
In het $3^e$ niveau is er nog ORD .

Een mogelijke volgorde is BOS, JFK, MIA, SFO, DFW, LAX, ORD.

Opmerking
Als er verschillende kanten van het ene niveau naar het volgende niveau lopen dan worden ze in willekeurige volgorde bezocht en dus zijn er ook andere mogelijkheden voor deze volgorde. JFK en MIA zouden bijvoorbeeld omgekeerd kunnen staan, ook de drie knooppunten in niveau twee kunnen onderling in een andere volgorde staan.

b BFS verdeelt de edges in tree edges (discovery edges) en nontree edges. De tree edges zijn kanten die leiden naar een nog niet bezocht knooppunt, en die samen met de knooppunten een boom vormen met als wortel het beginknooppunt. In de voorbeeldgraaf zijn dat voor de gekozen volgorde: BOS → JFK , BOS → MIA , JFK → SFO , JFK → DFW , MIA → LAX en DFW → ORD.

Er zijn twee soorten nontree edges. Cross edges, kanten die een knooppunt verbinden met een knooppunt dat noch zijn voorouder noch zijn nakomeling is in de BFS boom: JFK → MIA, MIA → DFW, DFW → LAX, DFW → SFO en LAX → ORD, en back edges, die een knooppunt met een voorouder in de BFS boom verbinden: JFK → BOS en ORD → DFW.

OPGAVE 7

We bekijken de string X = 'dogs do not spot hot pots or cats'.

8 punten

a Geef een frequentietabel en een Huffman-boom voor (de karakters van) X en geef de bijbehorende code van de karakters 'd', 'o', 'g' en ' '.

7 punten

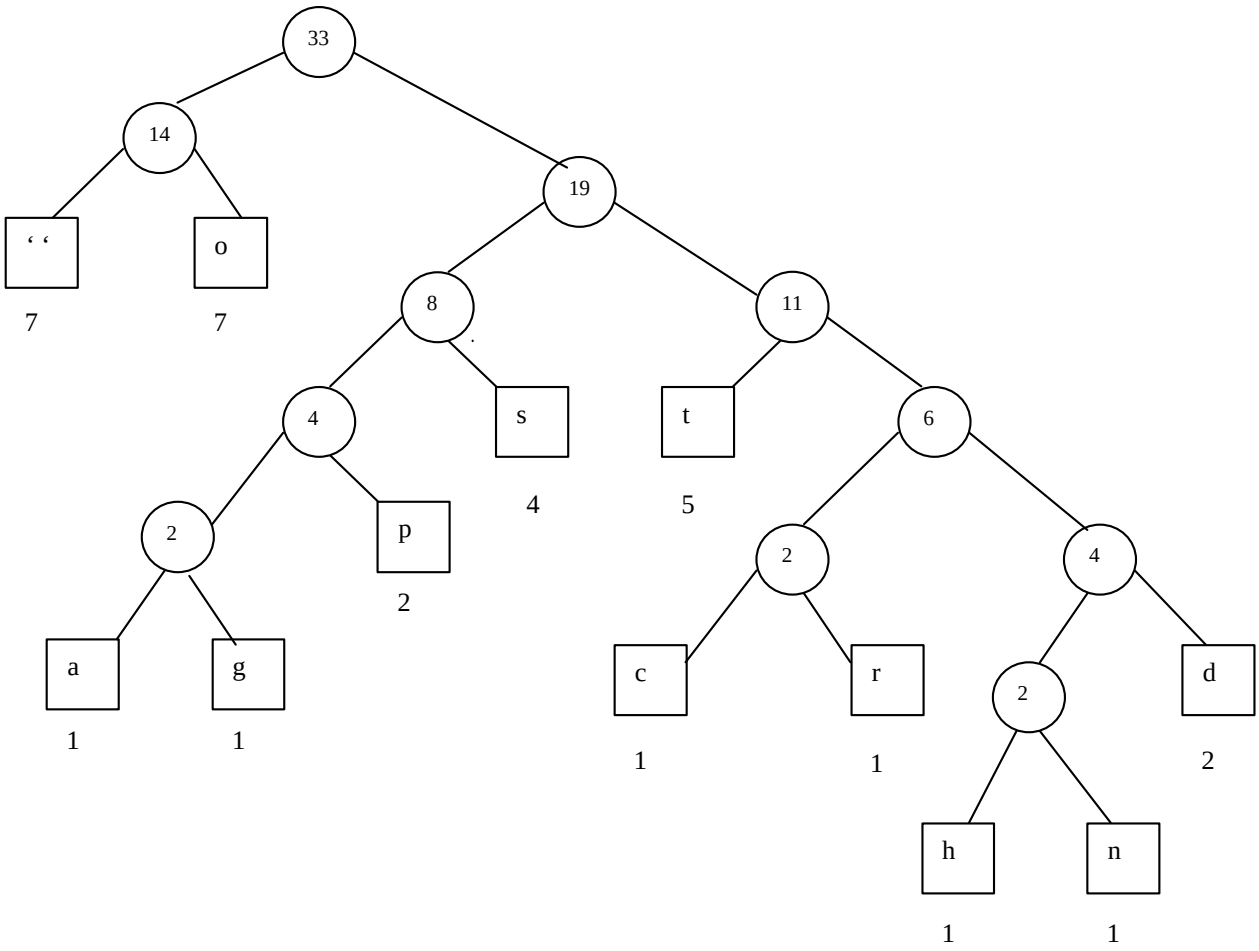
b Zijn er andere Huffman-bomen voor X mogelijk? Zo ja, geef er één en geef de bijbehorende code van de karakters 'd', 'o', 'g' en ' '.; zo nee, waarom niet?

UITWERKING OPGAVE 7

a De frequentietabel van de string X = 'dogs do not spot hot pots or cats' is:

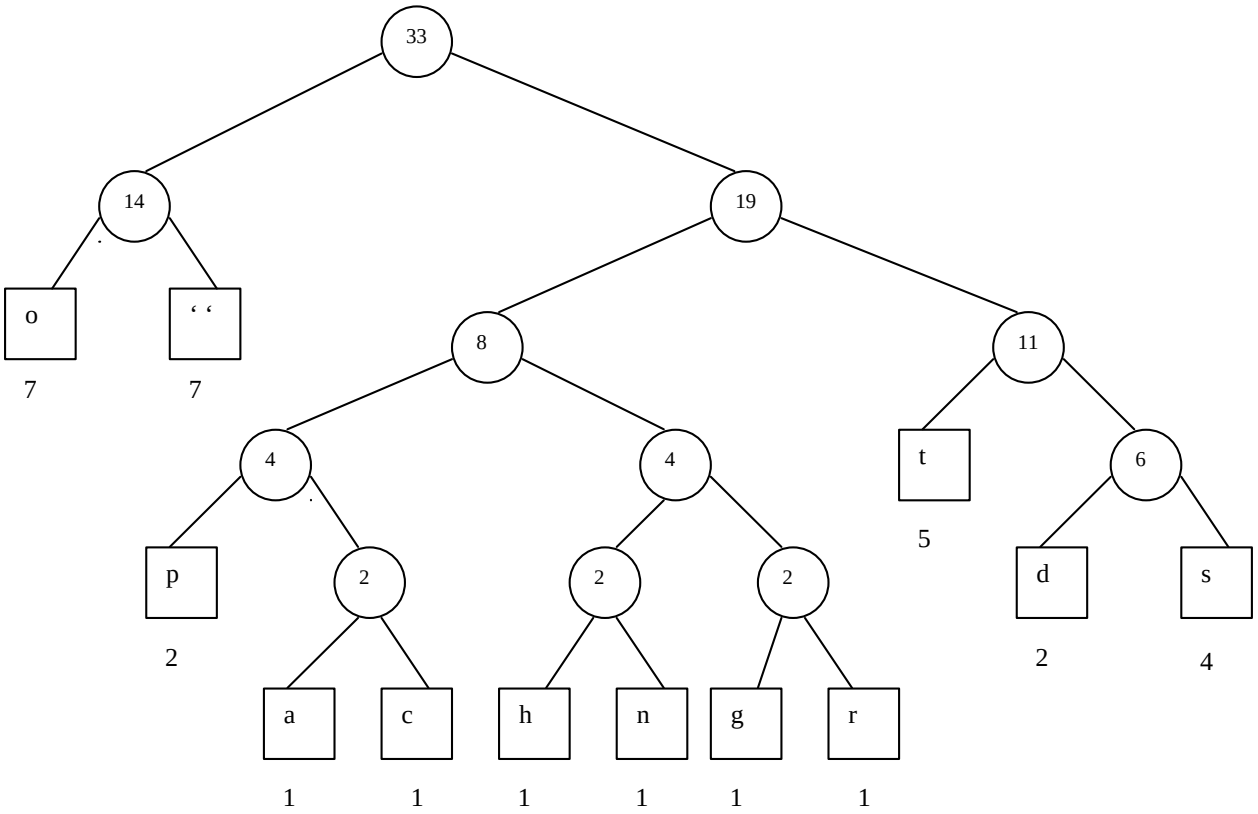
| ' ' | A | c | d | g | h | n | o | p | r | s | t |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 1 | 1 | 2 | 1 | 1 | 1 | 7 | 2 | 1 | 4 | 5 |

Na verwerking met het Huffman-coderingsalgoritme ontstaat de volgende boom:




De codes worden dan:
d = 11111
o = 01
g = 10001
‘ ‘ = 00

b Er zijn andere Huffman-bomen mogelijk. Het hangt af van de volgorde in de frequentietabel (of liever gezegd de volgorde waarin de elementen in eerste instantie in Q worden gezet) en ook van de implementatie van de priorityqueue-methoden: nl. om de heaporde te herstellen bij insertie en removal wordt een ouder die groter is dan zijn kind(eren) met het kleinste kind omgewisseld. Echter als beide kinderen even groot zijn geeft het boek geen duidelijke richtlijnen en kun je blijkbaar kiezen tussen het linker en rechterkind waarmee de ouder verwisseld wordt. Dus zowel verschillen in de volgorde van de karakters als verschillen in de manier van ‘heapen’ kunnen tot een andere boom leiden. Onderstaande boom ontstaat door de spatie als laatste karakter toe te voegen aan Q in plaats van als eerste karakter.




De codes worden dan:
d = 1110
o = 00
g = 10110
‘ ‘ = 01