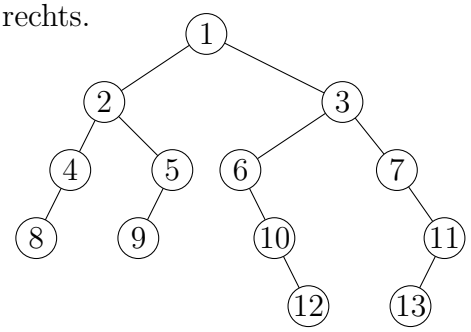
Het tentamen bevat zes opgaven. Gevraagde functies en programma's mogen in pseudo-code gegeven worden. Geef steeds voldoende uitleg. Succes.

1. **a)** Beschrijf de abstracte datastructuur *Priority Queue*: wat wordt opgeslagen, wat zijn de standaard operaties (en wat doen ze)? Abstract!

b) Beschrijf de implementatie *Leftist tree* voor Priority Queues. Wat is de structuur van de bomen? Beschrijf de basisoperatie (ritsen). Hoe worden verwijderen en toevoegen met deze basisoperatie uitgevoerd?

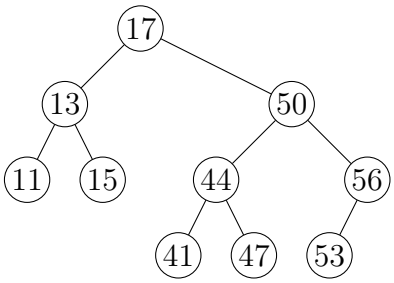
2. **a)** (i) Welk voordeel heeft een symmetrisch bedrade boom boven een gewone binaire boom? Welk nadeel staat hier tegenover?

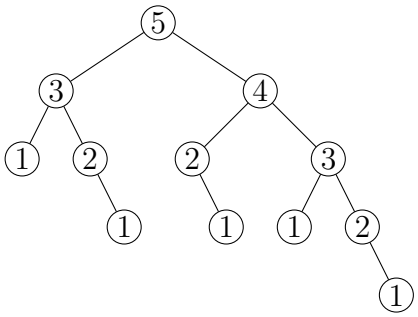
(ii) Breng de symmetrische bedrading aan in onderstaande boom; alleen draden naar rechts.



b) Stel er loopt een rechter draad van knoop x naar knoop y. Laat zien hoe we beginnend in y weer in x terecht kunnen komen. Waarom kan dit op die manier niet gebeuren als de link van x naar y een rechter tak is?

3. *Over AVL-bomen.* Deze twee bomen worden bij de volgende onderdelen gebruikt.





a) Bekijk de boom links. (i) Welke AVL-boom ontstaat als we knoop 17 verwijderen?

Bekijk de oorspronkelijke boom. (ii) Welke AVL-boom ontstaat als we zowel knoop 11 als knoop 15 verwijderen?

Geef uitleg en tussenstappen: welke rotaties vinden waar plaats?

b) Hierboven (rechts) een minimale AVL-boom van hoogte 5, dwz. met zo weinig mogelijk knopen. De hoogte wordt gemeten in het aantal knopen.

(i) Laat zien dat een AVL-boom van hoogte h tenminste $\lceil \frac{h}{2} \rceil$ geheel gevulde nivo's heeft.

(ii) Nu we dit weten, hoeveel knopen heeft een AVL-boom van hoogte h dus ten minste?

4. Hieronder het algoritme van *Floyd-Warshall*, voor kortste paden in een gerichte graaf.

Floyd-Warshall

```
// initially dist equals the adjacency matrix
for k from 1 to n
do   for i from 1 to n
     do   for j from 1 to n
          do   if dist[i,k] + dist[k,j] < dist[i,j]
               then dist[i,j] = dist[i,k] + dist[k,j]
               fi
          od
     od
od
```

a) Oorspronkelijk bedacht Warshall een variant van dit algoritme, namelijk voor transitieve afsluiting van een graaf, de Boolese matrix `pad[i,j]` waarvan de waarde aangeeft of er een pad van i naar j bestaat.

Wat moet er veranderen aan de geschetste code om de transitieve afsluiting te berekenen? (Werk direct met Booleans en logische operaties, en kijk dus niet achteraf of de afstand bestaat.)

b) Terug naar het kortste paden-algoritme. Wat moet er aan de code worden toegevoegd om niet alleen de afstand maar ook het kortste pad zelf te kunnen terugvinden?

Hieronder de waardes (voor oplopende k) die berekend worden voor een graaf die gecodeerd staat in de 0-de matrix. Hoe wordt het kortste pad tussen 3 en 1 gevonden?

A^0	1	2	3	4
1	0	4	1	∞
2	4	0	2	5
3	∞	2	0	∞
4	-2	5	∞	0

A^1	1	2	3	4
1	0	4	1	∞
2	4	0	2	5
3	∞	2	0	∞
4	-2	**2**	**-1**	0

A^2	1	2	3	4
1	0	4	1	**9**
2	4	0	2	5
3	**6**	2	0	**7**
4	-2	2	-1	0

A^3	1	2	3	4
1	0	**3**	1	**8**
2	4	0	2	5
3	6	2	0	7
4	-2	**1**	-1	0

A^4	1	2	3	4
1	0	3	1	8
2	**3**	0	2	5
3	**5**	2	0	7
4	-2	1	-1	0

5. **a)** Zowel bij *binaire zoekbomen* als bij *hashen* kan het zoekgedrag ontaarden in lineair zoeken. (i) Leg voor elk van beide datastructuren uit wanneer dit kan gebeuren. (ii) Als we dit beslist willen vermijden, welk type datastructuur biedt dan uitkomst?

We kijken nu niet naar dit extreme gedrag, maar naar de gemiddelde prestaties. (iii) Waarom is, bij een groot aantal sleutels, een hashmethode te verkiezen boven opslag in een binaire zoekboom?

b) Behalve toevoegen en zoeken, willen we ook regelmatig sleutels verwijderen. Bespreek het verwijderen van sleutels bij hashen.

6. **a)** Pas het algoritme van Huffman toe op symbolen $a, b, \ldots, f$ met respectievelijke frequenties $1, 11, 1, 3, 6, 8$

b) Bespreek waar het algoritme een keuze had (anders dan de keuze tussen links en rechts) en teken de alternatieve boom.

Geef ook een boom die even goed is als de gevonden bomen, maar *niet* door het algoritme gevonden had kunnen worden. Leg uit.