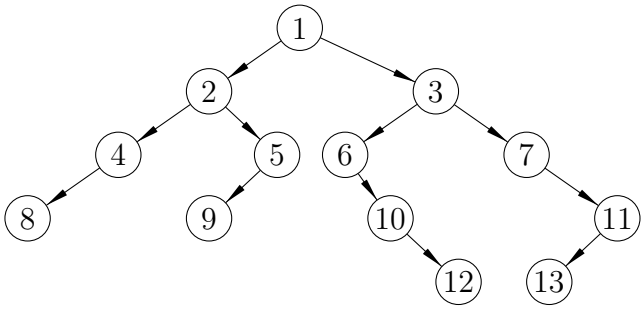
Gevraagde functies en programma's mogen in pseudo-code gegeven worden.
Geef steeds voldoende uitleg.

1) **a.** Welk voordeel heeft een symmetrisch bedrade boom boven een gewone binaire boom? Welk nadeel staat hiertegenover?

b. Breng de symmetrische bedrading aan in onderstaande boom.



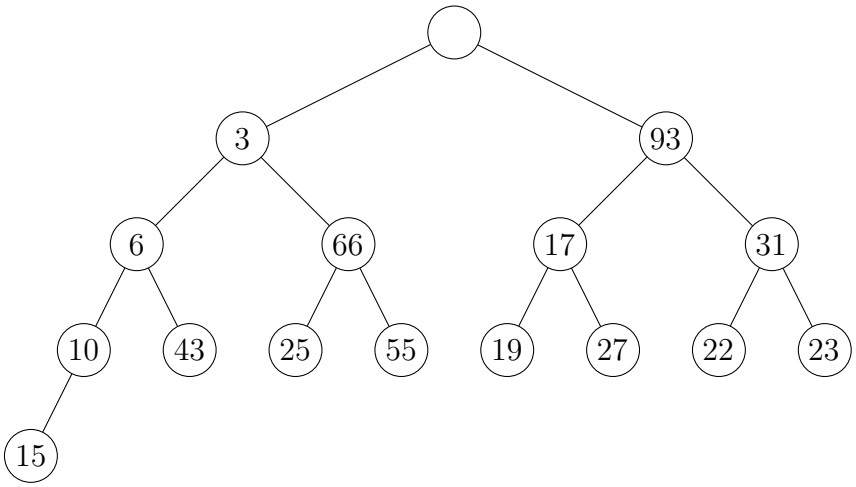
c. Beschrijf hoe we met behulp van takken en draden de vader van een gegeven knoop kunnen vinden als deze knoop het linker kind van zijn vader is. Geef een verklarend plaatje.

d. Schrijf een functie die een gegeven blad uit een bedrade boom verwijdert. Neem hierbij aan dat er een functie `VindVader(deze, li)` bestaat die de vader van de gegeven knoop `deze` oplevert, en 0 (`null`) als `deze` de wortel van de boom is. Verder is `li` een variabele die de waarde 1 (`true`) krijgt als `deze` een linker kind is, en 0 (`false`) anders.

hint: Kunnen er draden *naar* een blad wijzen ?

2) **a.** Geef de definitie van priority queue. Vermeld de operaties inclusief parameters, pre- en postcondities en een (informele) beschrijving.

Gegeven is de volgende SMM heap H:



b. Voeg achtereenvolgens de getallen 11, 54 en 1 toe aan H. Uiteraard dient de boom na iedere toevoeging een SMM heap te zijn.

Geef met tussenresultaten en een korte toelichting duidelijk aan hoe je aan je antwoorden komt.

c. Voer achtereenvolgens de operaties DeleteMax() en DeleteMin() uit op H (de originele SMM heap dus).

Geef ook nu duidelijk aan hoe je aan je antwoord komt.

d. Geef aan op welke plekken in een (willekeurige) SMM heap de één na grootste waarde zou kunnen staan. Geef voor ieder van deze plekken een voorbeeld SMM heap waarbij de één na grootste waarde inderdaad op die plek staat. Neem aan dat de getallen in de heap allemaal verschillend zijn.

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

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

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

Beschouw hashen met twee hash–functies h en p. Het $(i+1)$-ste bezochte adres $h(K,i)$ is zoals gewoonlijk $h(K) - i \cdot p(K)$ (modulo de tabelgrootte M).

d. Welke twee punten zijn belangrijk bij de keuze van de adres-functie h (óók bij lineair hashen) ? Waaraan moet de stap-functie p altijd voldoen ?

Wanneer spreken we van *lineair* hashen, *dubbel* hashen, en van hashen met *secundaire clustering* ?

e. In een tabel $T[0..7]$ met acht elementen zijn de sleutels 60, 56, 98, 73, 63, 50 en 96 terecht gekomen op de plekken 0,2,3,4,5,6 en 7 respectievelijk (plek 1 is vrijgebleven). Besloten wordt om de tabel te vergroten tot 13 elementen $T[0..12]$, en de sleutels *ter plekke* opnieuw te hashen. Hierbij wordt gebruik gemaakt van $h(K) = K \bmod 13$ en $p(K) = (K \bmod 10) + 1$. Voer dit uit.

Geef in het antwoord duidelijk aan in welke volgorde en op welke plaatsen de sleutels geprobeerd (en geplaatst) worden.

nb. om wat rekenwerk te besparen:
modulo 13 geldt dat $60 \equiv 8$, $56 \equiv 4$, $98 \equiv 7$, $73 \equiv 8$, $63 \equiv 11$, $50 \equiv 11$ en $96 \equiv 5$.

4) 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. Geef een efficient algoritme dat de *failure-links* voor een patroon bepaalt.

b. Bepaal de *failure-links* voor het patroon P = `ABACBABCCA`.

c. We zoeken naar P in de tekst T = `ABAB BCAB ACBA BACA BABC` (hier staan de spaties voor de leesbaarheid). Geef nauwkeurig aan hoe het zoeken volgens de KMP-methode gebeurt. Welke letters worden telkens met elkaar vergeleken?

d. Soms constateren we bij het zoeken dat P[i] niet overeenkomt met de letter uit de tekst en dat de *failure-link* wijst naar de letter P[k] die gelijk is aan P[i]. Dit leidt tot een overbodige vergelijking. Geef een eenvoudig algoritme om de *failure-links* aan te passen zodat deze situatie niet optreedt.

e. Pas de *failure-links* uit onderdeel **b.** aan zoals hierboven beschreven.