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

1. Drie losstaande vragen over binaire zoekbomen.

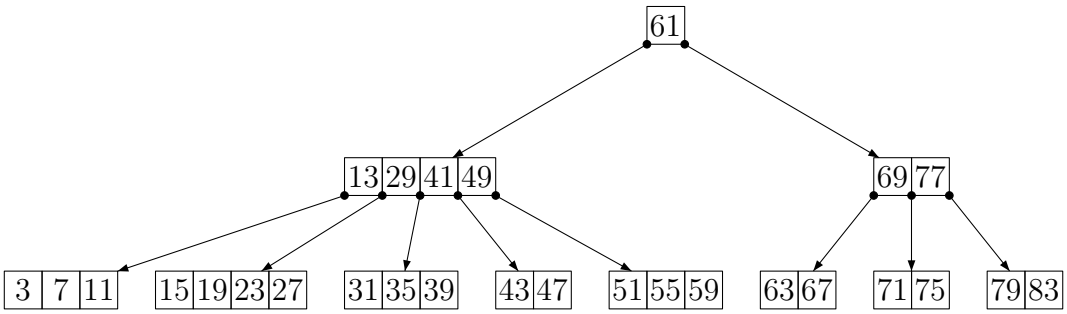
a) Om welke reden kunnen draden worden aangebracht in een binaire zoekboom? Welk nadeel hebben draden?

b) Als we vaak de k-de sleutel (in grootte, dwz. in inorde) in een binaire zoekboom willen vinden (voor een variabele k) hoe kan de boom dan worden aangepast? Wat verandert er aan de (aangepaste boom) wanneer een sleutel wordt toegevoegd?

c) Beschrijf een dubbele rotatie in een binaire zoekboom. Wanneer wordt deze toegepast in AVL-bomen?

2. **a)** Wat is het maximale en minimale aantal sleutels dat geplaatst kan worden in een B-boom van orde m en drie nivo's diep?

Beschouw de volgende B-boom B van orde 5:



Zorg ervoor dat we bij het toevoegen en verwijderen steeds een B-boom van orde 5 behouden. Geef een korte toelichting en (zo nodig) tussenresultaten. Ongewijzigde delen van de boom kunnen schematisch worden aangegeven.

b) i. Voeg aan B de sleutel 24 toe.

ii. Voeg aan B (de originele boom dus) de sleutel 50 toe.

c) Verwijder de sleutel 77 uit B (alweer de originele boom).

3. **a)** Wat is een topologische ordening op de knopen in een gerichte graaf $G = (V, E)$? Wees precies.

b) Hieronder staat een schematische recursieve functie voor *depth-first search*.

```
DFS (KnoopType x)
{   // aanname: knoop x nog niet bezocht
    bezoek x
    markeer x als bezocht
    for (elke knoop w bereikbaar vanuit x)
    {    if (w niet bezocht)
         {   DFS (w)
         }
    }
    // knoop x volledig afgehandeld:
    push x op stapel S
}
```

Beredeneer dat de knopen op de stapel `S` een topologische ordening van de acyclische graaf vormen nadat `DFS` op de achtereenvolgende knopen zonder inkomende takken is aangeroepen.

c) Gegeven is een adjacency-list representatie van een acyclische graaf G en een lijst met een topologische ordening voor G.

Beschrijf hoe het langste pad in G efficient kan worden bepaald. Pseudo-code is welkom.

4. **a)** Bij het algoritme van Knuth-Morris-Pratt berekenen we *failure links* voor ieder van de posities in het patroon dat we zoeken. Wat weten we van het patroon als gegeven is dat de failure link van positie k gelijk is aan r?

b) Bereken op efficiënte wijze de failure links voor het patroon P = `abacaaba`.

c) We zoeken naar P in de tekst T = `abab caab aaab acab acaa baba` (hier staan de spaties louter voor de leesbaarheid). Geef nauwkeurig aan hoe het zoeken volgens het KMP-algoritme gebeurt. Welke letters worden telkens met elkaar vergeleken?

d) Geef een voorbeeld van een type patroon waar KMP significant sneller werkt dan het naïeve algoritme.

(Het naïeve algoritme om patronen in een tekst te zoeken werkt als volgt: we leggen het patroon aan het begin langs de tekst en we vergelijken patroon en tekst letter voor letter. Wanneer er verschil optreedt, schuiven we het patroon één letter verder en we beginnen opnieuw.)