Het tentamen bevat vijf opgaven. Graag elke opgave op een nieuw blad beginnen.
Geef steeds voldoende uitleg. Succes.

1. **a)** Wat is de tijdscomplexiteit (O) van de volgende operaties?

i. Een element toevoegen in een ongesorteerd gelinkte lijst
ii. Toevoegen in een AVL gebalanceerde binaire zoekboom
iii. Toevoegen in een hashtabel (average case)
iv. Toevoegen in een hashtabel (worst case)

Drie vragen over het zoeken op waarde in de respectieve structuren.

b) Vergeleken met gelinkte lijsten zijn datastructuren geïmplementeerd met gebalanceerde binaire zoekbomen (zoals `set` en `map` in C++) en datastructuren geïmplementeerd met hashtabellen (zoals `unordered_set` en `unordered_map`) beter in zoektijd.

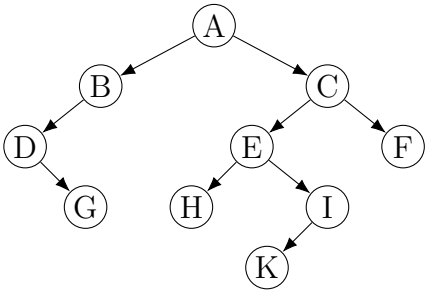
Wanneer zou u toch een gelinkte lijst gebruiken in plaats van een gebalanceerde binaire zoekboom of een hashtabel? Wat soort functionaliteit bieden gelinkte lijsten dat niet mogelijk is in de andere twee datastructuren?

c) Verklaar de verschillen in de werkkarakteristieken van datastructuren die met gebalanceerde binaire zoekbomen zijn geïmplementeerd (zoals `set` in C++) versus datastructuren die met hashtabellen zijn geïmplementeerd (zoals `unordered_set`).

Wanneer kiest u het ene boven het andere?

d) Collision in een hashtabel vindt plaats als sleutel k moet worden toegevoegd maar $h(k)$ al bezet is. Beschrijf twee manieren om een collision op te lossen. In andere woorden, als $h(k)$ al bezet is, waar / hoe wordt k toegevoegd?

2. **a)** Neem onderstaande boom over, en breng symmetrische bedrading aan (alleen rechter draden).



We bekijken de volgende abstracte manier om een inorde (symmetrische) wandeling door een boom uit te voeren.

inorde
```
move to first node
repeat
   visit node
   move to successor node
until done
```

b) Leg uit hoe de opdrachten *move to first*, *move to successor* en *done* worden geïmplementeerd in een bedrade binaire boom. Gebruik pseudo code.

c) Idem, nu in een onbedrade boom, maar gebruik makend van een stapel.

3. We gaan in deze opgave uit van max-heaps.

a) Geef de definitie van een *binary heap*, en beschrijf de twee basis-operaties *bubble up* en *trickle down*.

b) Illustreer hoe de priority queue operaties *DeleteMax* en *Insert*(80) worden uitgevoerd, aan de hand van de volgende heap: $[98, 37, 70, 30, 29, 58, 23, 15, 7]$.

c) Stel dat twee priority queues (elk gerepresenteerd als binairy heap) moeten worden samengevoegd tot één priority queue. Leg uit hoe we dat in lineaire tijd kunnen doen.

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. Die bepaalt de Boolese matrix `conn[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?

Hoe vinden we met die informatie vervolgens het kortste pad?

5. **a)** Wat is de tijdscomplexiteit (O) van de volgende operaties?

i. Naïef zoeken naar een patroon in een tekst

ii. Knuth-Moris-Pratt (KMP) patroon zoeken

b) Bereken de foutkoppelingen (*failure links*) $FLink[k]$ die door het KMP-algoritme worden gebruikt voor elke positie k in het zoekpatroon

$$\mathrm{P} = 12301234$$

c) We zoeken naar P in de tekst

$$\mathrm{T} = 0012\,3001\,2301\,234$$

Geef nauwkeurig aan hoe het zoeken volgens het KMP-algoritme gebeurt. Welke letters worden telkens met elkaar vergeleken?