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

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

De Fibonacci queue en binomial queue zijn voorbeelden van priority queues.

b. i. Construeer een *binomial queue* door, uitgaande van een lege queue, achtereenvolgens de volgende getallen toe te voegen: 33, 2, 27, 39, 44, 9, 56 en 1. Uiteraard dient ook het tussenresultaat na iedere toevoeging een binomial queue te zijn. Geef via tussenresultaten en een korte toelichting duidelijk aan hoe je aan je antwoord komt.

ii. Voer op het resultaat van het vorige onderdeel twéé maal de operatie DeleteMin (VerwijderMin) uit.
Geef ook hier via tussenresultaten en een korte toelichting duidelijk aan hoe je aan je antwoord komt.

c. Gegeven is onderstaande *Fibonacci queue* Q. Geef een serie operaties, inclusief eventuele argumenten, die uitgaande van een lege queue Q oplevert.

nb. In bovenstaande tekening van Q zijn knopen die al een subboom hebben verloren niet gemarkeerd. Afhankelijk van de serie operaties kunnen er echter wel knopen gemarkeerd zijn. Geef in je eigen tekeningen wél steeds de markeringen aan.

d. **bonus!** Geef een boom met de heap eigenschap en een minimum aantal knopen zó dat de boom geen Fibonacci queue kan zijn. Laat duidelijk zien waarom deze boom geen Fibonacci queue kan zijn en waarom een boom met minder aantal knopen altijd een Fibonacci queue moet zijn. Een formeel bewijs is niet nodig.

De volgende opgave gaat over de B-bomen van het college, waar sleutels zowel in de bladeren als ook in de interne knopen opgeslagen worden.

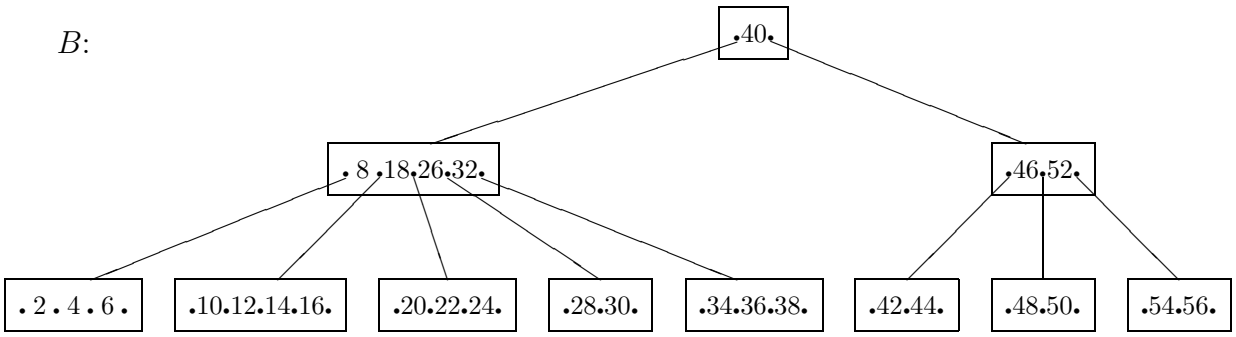
2) **a.** Hoeveel sleutels bevat een knoop in een B-boom van orde m minimaal? En hoeveel maximaal?

b. We beperken ons nu tot B-bomen van orde $m = 5$. Hoeveel sleutels bevat het h-de niveau van een B-boom van orde 5 minimaal? En hoeveel maximaal? Licht je antwoord toe.

nb. het eerste niveau is het niveau waarop de wortel van de B-boom zich bevindt.

Beschouw de volgende B-boom B van orde 5:

B:



In de onderdelen hieronder is het niet nodig om steeds de hele boom over te nemen in je uitwerkingen. Geef het relevante deel van de boom.

c. i. Voeg aan B de sleutel 13 toe.

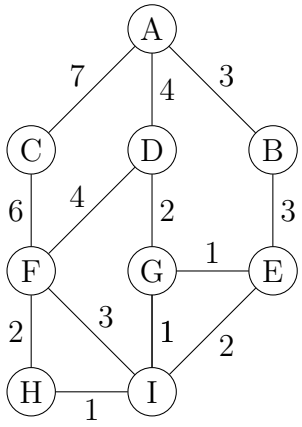
ii. Voeg aan B (de originele boom dus) de sleutel 33 toe.

Zorg ervoor dat we bij het toevoegen een B-boom van orde 5 behouden. Geef een korte toelichting en (zo nodig) tussenresultaten.

d. Verwijder de sleutel 46 uit B (de originele boom dus).

Zorg ervoor dat we bij het verwijderen een B-boom van orde 5 behouden. Geef een korte toelichting en (zo nodig) tussenresultaten.

3) **a.** Gegeven is de volgende ongerichte samenhangende graaf met gewichten op de takken.



Pas het algoritme van Dijkstra toe op de gegeven graaf. Gebruik knoop A als beginknoop van het algoritme.

b. Zowel de algoritmes van Dijkstra en Prim leveren een opspannende boom op.

Wat is het minimum aantal gewichten dat dient te worden gewijzigd in de bovenstaande graaf zodat de algoritmes van Prim en Dijkstra dezelfde opspannende boom opleveren? Geef ook aan welke gewichten je zou kunnen wijzigen om dit voor elkaar te krijgen.

c. Het algoritme van Kruskal is een algoritme voor het bepalen van een minimale opspannende boom in een ongerichte samenhangende graaf. Het algoritme is hieronder gegeven in pseudo code.

Algoritme *Kruskal*
  gegeven ongerichte samenhangende graaf $G = (V, E)$
  begin
    begin met een lege boom T
    while T verbindt niet alle knopen **do**
      kies tak (u, v) uit E met minimaal gewicht
      verwijder (u, v) uit E
      if (u, v) geen kring vormt met takken uit T **then**
        voeg (u, v) aan T toe
      fi
    od
  end

Geef twee datastructuren waar het algoritme van Kruskal efficiënt mee is te implementeren. Wat is de complexiteit van het algoritme bij gebruikmaking van deze datastructuren?

4) **a.** Geef voor beide onderstaande reguliere expressies de bijbehorende boomstructuur. Geef ook de corresponderende postfix notaties.

i. `A*(AA+B*)*` **ii.** `(BB+A*B*A)*`

b. Construeer voor de expressie `A*(AA+B*)*` de bijbehorende eindige automaat. Nummer de knopen in deze automaat in de volgorde waarin ze tijdens de evaluatie van de boom aan de automaat worden toegevoegd.
Teken nogmaals de boom die bij de expressie hoort, en geef bij elke knoop van de boom de nummers van de corresponderende begin- en eindknoop van de eindige automaat.

c. Verifieer dat de string '`ABA`' *niet* en '`ABAA`' *wél* aan de gegeven expressie voldoet, door het uitvoeren van een complete wandeling door de automaat. Geef het zoekproces schematisch in een vertakkende 'boom'structuur weer. Hierbij dienen ook 'doodlopende' paden opgenomen te zijn.

d. De gebruikte methode gaat na of een gegeven string S aan een reguliere expressie X voldoet. Hoe moeten we de methode aanpassen om erachter te komen of er een *substring* van S is die aan X voldoet? (Zonder de zoekactie voor elke substring apart uit te hoeven voeren.)
Bijvoorbeeld: '`ABAABAB`' voldoet niet aan `B(AA)*B`, maar heeft wel een substring die voldoet (nl. '`BAAB`'). Geen van de substrings van '`ABAAABAB`' voldoen aan de expressie `B(AA)*B`.