Examen Datastructuren en Algoritmen II

Naam : .....

Lees de hele oefening zorgvuldig voordat je begint ze op te lossen!
Als je niet goed verstaat wat de vraag of taak is, vraag het aan de lesgever!
Voor oefeningen die fout verstaan zijn, kunnen geen punten gegeven worden!

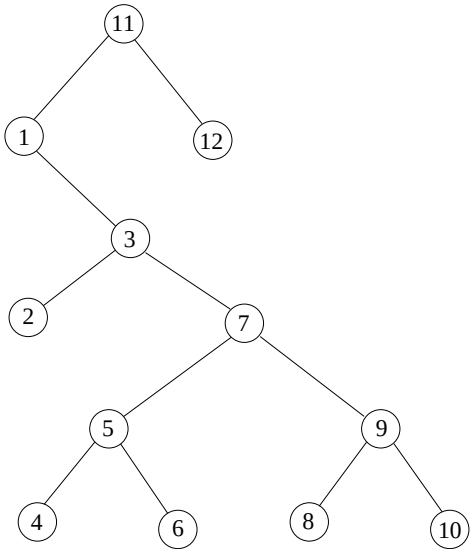
Schrijf leesbaar. Oplossingen die niet leesbaar zijn, kunnen ook niet beoordeeld worden.

Als technieken toegepast moeten worden, toon altijd voldoende tussenstappen om te kunnen zien wat er gebeurt en dat de technieken goed verstaan zijn.

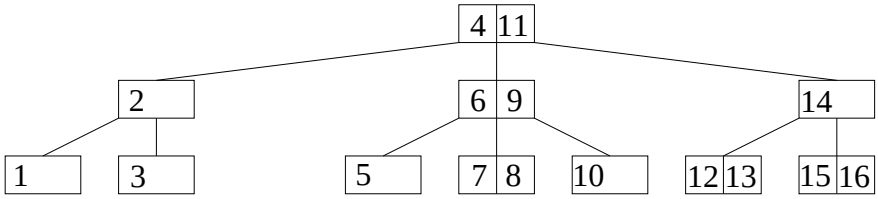
Stellingen uit de les mogen natuurlijk altijd gebruikt worden zonder dat het bewijs opnieuw gegeven moet worden (behalve in gevallen waar het expliciet anders staat)!

1. **Zoekbomen 3 pt**

   - Verwijder sleutel 7 uit de volgende semi-splay boom.



Voeg de sleutels 17, 18, 19 en 20 toe aan de volgende 2-3 boom:



- Wat is de maximale en wat is de minimale diepte van een 2-3 boom? Gevraagd is de exacte formule en niet de $O()$-notatie. Geef uitleg hoe de bomen eruitzien als deze maximale resp. minimale diepte verwezenlijkt wordt.

- In de les werden zoekbomen zo gedefinieerd dat er nooit twee keer dezelfde sleutel bevat kan zijn. Stel nu dat wij semisplay op bomen willen toepassen waarin dezelfde sleutel meerdere keren kan voorkomen (dus in verschillende toppen) maar waar (op één wijziging na) nog altijd dezelfde operaties toegepast kunnen worden. De wijziging is dat als je een sleutel *s* die je wilt toevoegen tijdens het toevoegen in een top *t* al tegenkomt, je geen fout meldt, maar de sleutel in de rechterboom van *t* toevoegt.

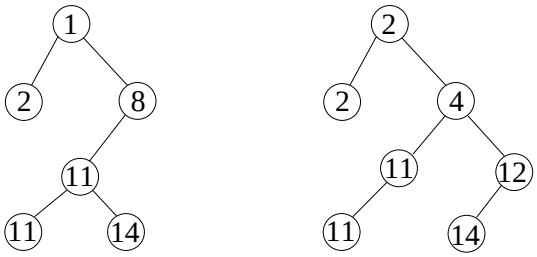
  Maar kan je op deze manier met een semisplay boom werken? Kies één van de volgende alternatieven:

  Ja, dat kan. Geef dan de nieuwe definitie van het woord zoekboom en leg uit waarom de herbalanceerbewerkingen lukken.

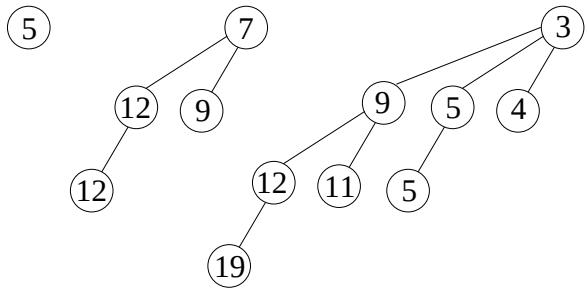
  Nee, dat kan niet. Geef dan een overtuigende redenering waarom dat niet mogelijk is.

2. **Heaps 3.5 pt**

- Merge de volgende twee skew heaps:



- Verwijder de kleinste sleutel uit de volgende binomiale wachtlijn.



- Een keten van bookmakers houdt de op een gebeurtenis (bv een paardenkoers) gemaakte weddenschappen bij in een heap. De weddenschappen zijn opgeslaan volgens de hoogte van de inzet en het is belangrijk altijd heel snel te weten wat de hoogste inzet is (omdat dat vaak opgevraagd wordt). De heaps worden ook doorgestuurd naar een centrale die dezelfde datastructuur gebruikt en waar de heaps tot één grote heap gemerged worden. Natuurlijk moet ook dat efficiënt gebeuren.

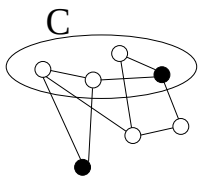
  Welke implementatie van de datastructuur heap zou jij voor deze toepassing voorstellen? Geef ook uitleg wat het voordeel is van jouw datastructuur in vergelijking met andere soorten heaps.

- Bewijs de volgende uitspraak:
  Als een top in een skew heap een rechterkind heeft dan heeft de top ook een linkerkind.

  Het is voldoende het voor heaps te bewijzen die opgebouwd worden zonder dat ooit een sleutel verwijderd werd.

3. **Online algoritmen 1.5 pt**

In een computernetwerk is het de bedoeling dat op sommige computers speciale software geïnstalleerd wordt die de juiste werking van de computer zelf en alle computers die direct met deze verbonden zijn, kan toetsen. De bedoeling is dus voor een verzameling C van computers die regelmatig getest moeten worden een verzameling T van computers te vinden, die zo klein mogelijk is, maar waar alle computers uit C een buur in T hebben. T mag natuurlijk ook toppen bevatten die niet in C zitten. In het volgende voorbeeld vormen de zwarte toppen bv. een minimale verzameling T die de toppen uit C kan toetsen.



Wij willen een algoritme hebben dat een verzameling C van computers die getest kunnen worden kan uitbreiden. Of precies: Stel dat een netwerk gegeven is en een reeks $c_0, \ldots, c_n$ van toppen (computers) in het netwerk. Dan berekent het algoritme eerst een verzameling T_0 van computers die c_0 kan testen, dan een verzameling T_1, die $\{c_0, c_1\}$ kan testen, etc. Daarbij mag de software wel op nieuwe computers geïnstalleerd worden, maar niet van computer verplaatst – of met andere woorden: $T_{i-1} \subseteq T_i$ voor alle $1 \leq i \leq n$. Dat noemen wij een online testset algoritme.

Bewijs de volgende uitspraak:

Voor elk online testset algoritme en elke n bestaat een samenhangend netwerk en een invoerreeks zodat het optimale aantal computers n is en het algoritme ten minste twee keer het optimale aantal computers gebruikt.

4. **Geamortiseerde complexiteit 2 pt**

   Wij werken met een datastructuur D die arrays $A_0, A_1, A_2, \ldots$ bevat, waarbij A_i leeg is of een array met 2^i sleutels waarin alle sleutels gesorteerd zijn. De bewerkingen zijn: teruggeven van de grootste sleutel in D (zonder die te verwijderen), teruggeven van de kleinste sleutel in D (zonder die te verwijderen) en het toevoegen van een nieuwe sleutel s aan een al bestaande datastructuur. De kost van de eerste twee bewerkingen is het aantal arrays waarnaar gekeken moet worden.

   Het toevoegen gebeurt als volgt: Wij plaatsen s in een array van lengte 1. Als A_0 leeg is, nemen wij deze array als A_0, anders mergen wij die met A_0 en krijgen een array van lengte $2 = 2^1$. Elke keer als wij twee arrays mergen tot een array A met lengte 2^i nemen wij in het geval dat A_i leeg is A als A_i en anders mergen wij A met A_i tot een array met grootte 2^{i+1} en gaan recursief door. De kost van het plaatsen in een array rekenen wij als 1 en de kost van het mergen rekenen wij als m waarbij m het aantal betrokken sleutels is.

   Bepaal de maximale **en** de geamortiseerde kost van een bewerking in een reeks van n bewerkingen op een initieel lege datastructuur. Je mag vrij kiezen op welke manier je de kosten bepaalt.

5. **Dynamisch programmeren 2 pt**

- Geef de pseudocode voor een dynamisch programmeren algoritme dat de volgende functie $f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ berekent:

$$f(n,m) = \begin{cases} 2 & \text{als } n = m = 0 \\ f(n-1,m) + f(n,m-1) & \text{als } n = m \text{ en } n,m > 0 \\ f(m,m) * f(n-1,m) & \text{als } n > m \\ f(n,n) * f(n,m-1) & \text{als } n < m \end{cases}$$

- Wat is de complexiteit van jouw algoritme? Is de complexiteit van het algoritme dat de recursieve definite direct implementeert polynomiaal begrensd? Toon aan dat jouw antwoorden juist zijn.

6. **Offline inpakalgoritmen 1 pt**

In de les hebben wij gezien:

Als het optimale aantal vrachtwagens voor een gegeven reeks van gewichten m is, dan gebruikt first fit dalend ten hoogste $\lceil \frac{4}{3}m \rceil$ vrachtwagens.

Wij hebben ook gezegd dat de heuristieken in de praktijk meestal veel beter presteren omdat de pakketten vaak veel kleiner zijn dan in de voorbeelden waarvoor de algoritmen slecht presteren.

Stel nu dat m het optimale aantal vrachtwagens voor een reeks $g_1, \ldots, g_n$ is en dat voor een zekere $x > 3$ het aantal pakketten in deze reeks met een gewicht groter dan $1/x$ ten hoogste m is. Bewijs voor dit geval een betere bovengrens dan $\lceil \frac{4}{3}m \rceil$ voor de prestatie van first fit dalend. Bewijs de beste bovengrens die je kan bewijzen. Resultaten uit de les mogen natuurlijk gebruikt worden!

7. **Een algoritme ontwerpen 1 pt**

   In de les werd altijd gezegd dat het belangrijk is de ideeën en technieken te leren om ze ook in andere samenhangen te kunnen toepassen. Voor het volgende probleem kan je inderdaad een idee toepassen die jullie in de les bij het uitbreiden van arrays hebben gezien:

   Een robot botst tegen een heel lange muur en moet de deur vinden. Maar de robot weet niet of de deur rechts of links van hem zit. De robot kan wel meten hoe ver hij rijdt en hij heeft ook een sensor die kan vaststellen als hij voor een deur staat en waar de muur is. Geef – gebruik makend van de ideeen die je voor het uitbreiden van arrays gezien hebt – een algoritme om de deur te zoeken zodat de robot maar $O(m)$ meters moet lopen totdat hij de deur vindt als de afstand van de deur m meter is. Wat is *het principe* dat je gebruikt?

8. **Verzamelingen 2 pt**

- Het universum is de verzameling $\{1, \ldots, 7\}$. Gebruik een union-find datastructuur met union by size en path compression. In gevallen waar niet uniek bepaald is wat de nieuwe wortel is, neem het element met de kleinere index.

  Pas de bewerkingen `union(6,7)`, `union(5,4)`, `union(1,2)`, `union(4,3)`, `union(5,6)`, `union(2,7)` in deze volgorde toe.

- – Toon aan dat een boom met n toppen in een union-find datastructuur die met union by size en path compression opgebouwd werd ten hoogste diepte $\lfloor \log n \rfloor$ heeft.
  – Is dat een goede bovengrens of is er ook een bovengrens $f(n)$ waarbij $f(n) = o(\log n)$?

NOG NIET OMDRAAIEN !