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!

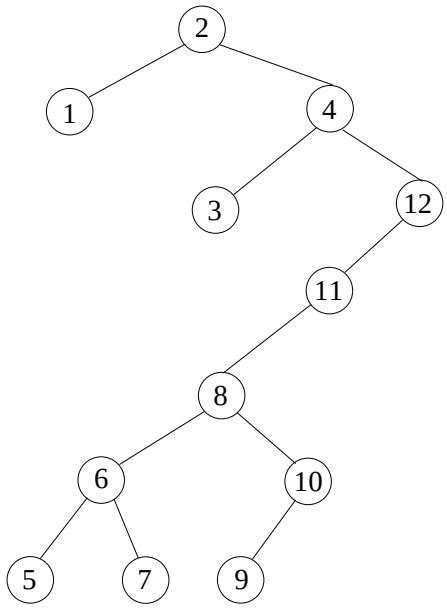
Schrijf leesbaar. Oplossingen die niet leesbaar zijn, kunnen ook niet beoordeeld worden.

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)!

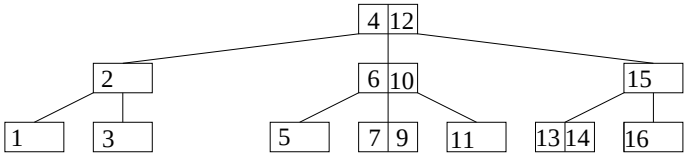
1. **Zoekbomen 3.5 pt**

   - Verwijder sleutel 8 uit de volgende semi-splay boom.



- In de les hebben we bewezen dat een rood-zwart boom een diepte van ten hoogste $2\log(n+1)$ heeft. Bewijs dat dat op een additieve constante na een goede bovengrens is. Tip: construeer rood-zwart bomen met oneven diepte k en $2^{(k+3)/2}-2$ toppen. Geef voor deze dan de diepte als functie van het aantal toppen. Het is zeker duidelijk hoe een rood-zwart boom met diepte 1 en zo weinig mogelijk toppen er moet uitzien. Begin daarmee en bouw een boom met diepte 3. Dan is het principe zeker al duidelijk.

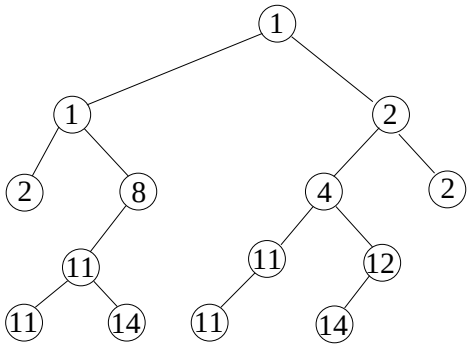
Voeg sleutel 8 toe aan de volgende 2-3 boom:



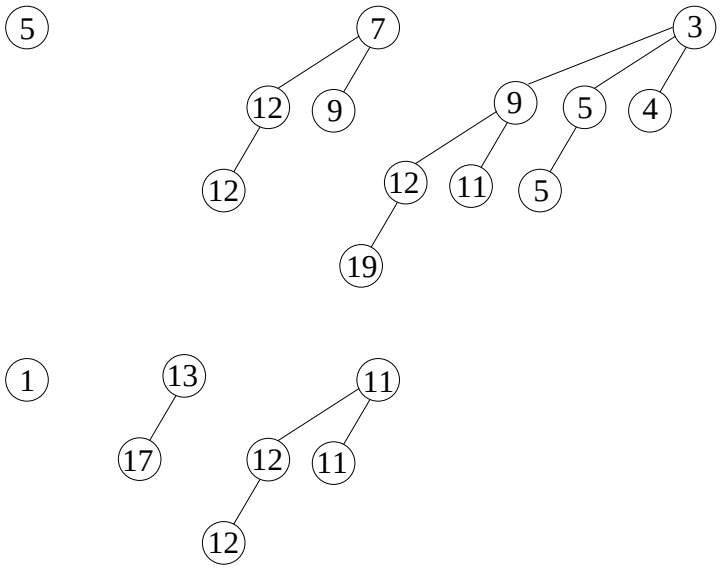
- Gegeven een zoekboom T en een deelboom T' van T. De deelboom T' bevat n sleutels. Geef het bewijs dat T' precies $n+1$ buitenbomen heeft.

2. **Heaps 3.5 pt**

- Verwijder de kleinste sleutel uit de volgende leftist heap:



- Merge de volgende binomiale wachtlijnen.



- Analoog met leftist heaps zou je *leftweight heaps* als volgt kunnen definieren:
  Een leftweight boom is een binaire boom waar elke top ten minste even veel toppen in zijn linkerdeelboom heeft als in zijn rechterdeelbom.
  Een leftweight heap is een leftweight boom waar voor elke top de sleutels in de kinderen ten minste even groot zijn als die in de top.

  Bewijs: In een leftweight heap met n toppen is de rechterpadlengte ten hoogste $\lfloor \log n \rfloor$.

- Bewijs de volgende uitspraak:
  Voor elke leftist heap L bestaat een reeks van toevoeg- en mergebewerkingen op initieel lege leftist heaps zodat L het resultaat van deze bewerkingen is.

3. **Dynamisch programmeren 2 pt**

Een bedrijf moet goederen $g_1, \ldots, g_k$ op één of meerdere vrachtwagens naar hetzelfde adres vervoeren. De goederen worden in volgorde $g_1, \ldots, g_k$ gemaakt en het bedrijf heeft 1 dag nodig voor elk van de goederen en ze worden zonder onderbreking gemaakt – dus g_i op dag i. Als g_i al op dag i wordt vervoerd, is er geen straf. Als g_i pas j dagen later wordt vervoerd, moet een straf $s(g_i, j)$ betaald worden. Dus is $s(g_i, 0) = 0$, maar voor de andere waarden van $s(g_i, j)$ mag je alleen maar veronderstellen dat ze groter dan of gelijk aan 0 zijn. Het vervoeren van een goed heeft een kost C, dus is het mogelijk alle goederen met kost $k * C$ te vervoeren als je elke dag een vrachtwagen stuurt. Alle goederen passen op één vrachtwagen, dus zou het ook mogelijk zijn, maar één vrachtwagen te sturen en alle goederen samen te vervoeren, wat een totale kost (met straffen) van $C + \sum_{i=1}^{k} s(g_i, k-i)$ zou opleveren. Maar misschien is een compromis tussen deze oplossingen optimaal...

Schrijf een polynomiaal algoritme met dynamisch programmeren dat de minimale som van de vervoerkosten en straffen bepaalt. Je moet niet noodzakelijk bijhouden wat de echte strategie is. Geef het algoritme in pseudocode met voldoende uitleg.

4. **Geamortiseerde complexiteitsanalyse 2 pt**

   Wij werken met twee binaire tellers B_1, B_2 die beide oneindig veel bits kunnen gebruiken. Wij beginnen met twee initieel lege tellers. De mogelijke operaties zijn

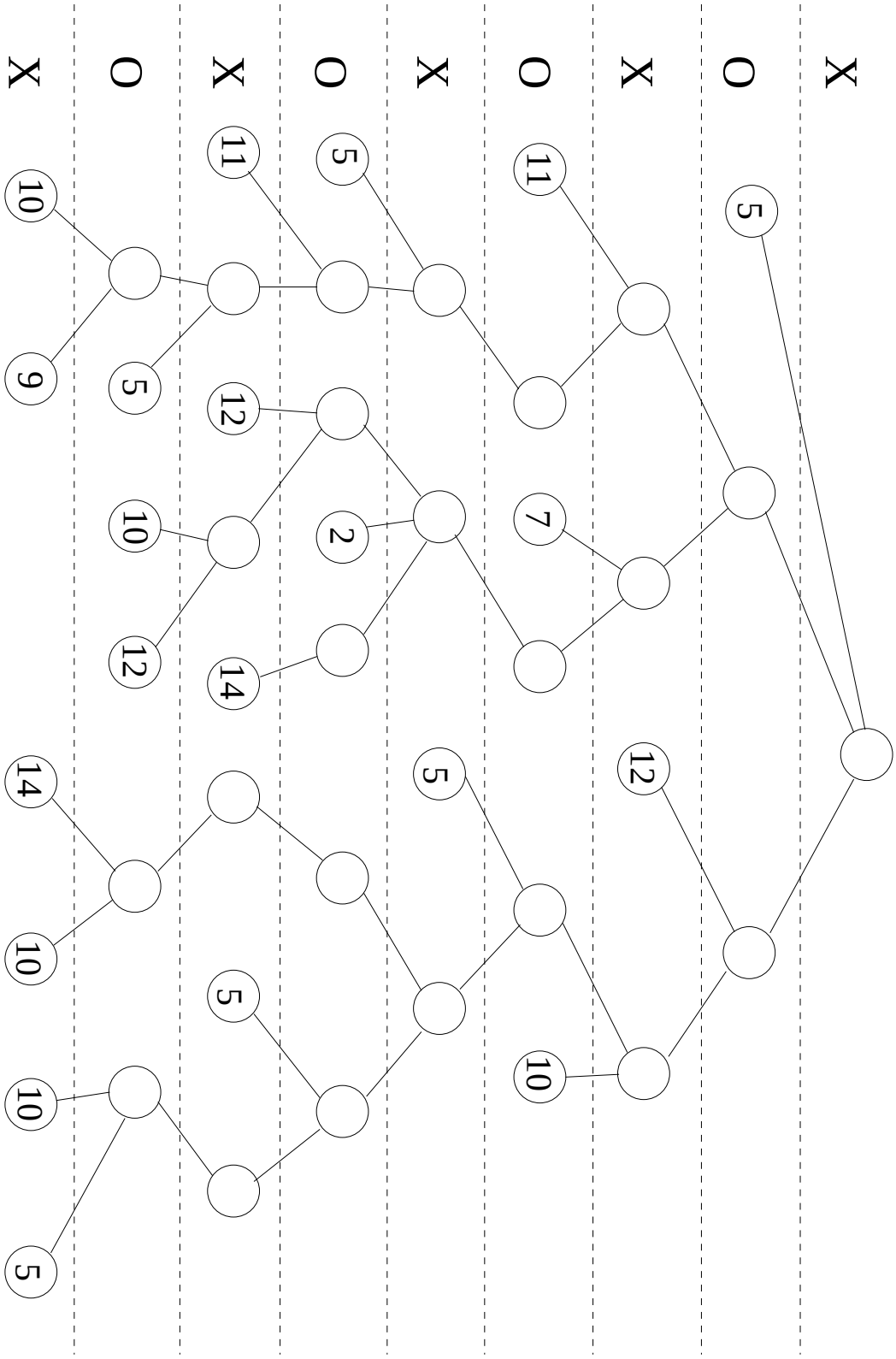
   - 1 bij B_1 optellen
   - 1 bij B_2 optellen
   - B_1 en B_2 optellen, het resultaat in B_2 opslaan en B_1 op 0 zetten.

   De kosten zijn in elk geval het aantal bekeken bits waarbij je ervanuit mag gaan dat je voor B_1 weet wat de bit met de hoogste index is die 1 is en dus bij het optellen al op voorhand weet hoeveel bits van B_1 je moet bekijken. Voor de eerste twee operaties komt dat neer op het aantal gewijzigde bits, maar voor de derde bewerking zou bv. als $B_1 = b_7 b_6 \ldots b_0 = 00001000$ en $B_2 = 10011000$ het resultaat zijn dat $B_1 = 00000000$ en $B_2 = 10100000$. Daarbij moest je 4 bits van B_1 en 6 bits van B_2 bekijken – de kost is dus 10. Twee bits van B_2 moest je niet bekijken.

   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 maar misschien is het niet slecht de potentiaalmethode te gebruiken en voor de potentiaal niet enkel het aantal bits met waarde 1 maar ook de hoogste index van een bit in B_1 die 1 is te gebruiken.

5. α–β–**snoeien 2 pt**

- Pas α–β–snoeien op de volgende spelboom toe om de waarde van het spel te berekenen. Evalueer altijd eerst de linkertakken. Schrijf gebruikte grenzen **aan** de toppen en de teruggegeven waarden **in** de toppen. Plaats streepjes door de bogen die naar deelgrafen leiden die je niet evalueert omdat je snoeit.



- – Geef de definitie van de waarde van een spel dat door spelbomen met gelabelde bladeren – zoals in de les gezien – voorgesteld kan worden.

- – Bewijs de volgende uitspraak of geef een tegenvoorbeeld: Als de waarde van een top w is en de top is een max-top (dus op even afstand van de wortel) dan geeft α–β–snoeien zoals in de les gezien voor deze top altijd een waarde x met $x \leq w$ terug.

6. **Gerandomiseerde algoritmen 2 pt**

- In de les hebben wij een algoritme gezien dat met een zekere kans een bovengrens voor de door een random number generator gegenereerde getallen bepaalt. Hier willen wij meer dan alleen een grens – wij willen precies het grootst mogelijke getal vinden dat gegenereerd kan worden. Maar hier weten we wel iets meer over dit grootst mogelijke getal – namelijk dat het van de vorm $2^n - 1$ is voor een zekere n. Of precies:

  Wij hebben een generator voor toevalsgetallen die toevalsgetallen in een bereik $0, \ldots, 2^n - 1$ voor een zekere n genereert. Beschrijf een algoritme dat het grootste van dit algoritme genereerbare getal bepaalt en met kans ten hoogste $1/(2^{50})$ een fout antwoord geeft. Toon aan dat jouw algoritme deze eigenschap heeft.

- Gebruik de Miller-Rabin priemgetallentest om te testen of 28 een priemgetal is. Als *toevallig* gekozen getal a met $0 < a < 28$ gebruik het getal 9.

7. **Het inpakprobleem 1 pt**

Toon aan: Er bestaat een algoritme dat polynomiaal tijdsbegrensd in n is en voor een gegeven reeks $g_1, \ldots, g_n$ van gewichten en $k \in \mathbb{N}$ kan beslissen of de gewichten op ten hoogste k vrachtwagens passen als en slechts als er een polynomiaal in n tijdsbegrensd algoritme bestaat dat voor een gegeven reeks $g_1, \ldots, g_n$ van gewichten een optimale plaatsing van de gewichten op vrachtwagens kan vinden.

NOG NIET OMDRAAIEN !