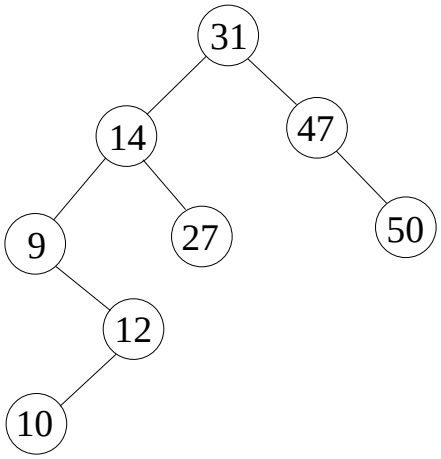
Examen Datastructuren en Algoritmen II

Naam : .....

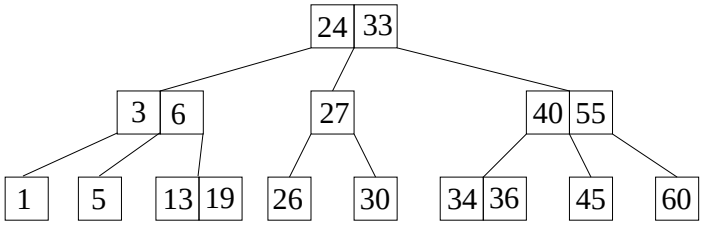
- Lees de hele oefening zorgvuldig voordat je begint ze op te lossen!
  Als je niet goed verstaat wat de vraag of taak is, vraag het aan de lesgever! Voor oefeningen die fout verstaan zijn, kunnen geen punten gegeven worden!
- Schrijf leesbaar. Oplossingen die niet leesbaar zijn, kunnen ook niet beoordeeld 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)!
- Geef alleen dan een antwoord als je denkt dat je de oplossing kent. Verspil geen tijd met de poging gewoon lange teksten te schrijven waarin zekere sleutelwoorden opduiken – zoals dat vaak geprobeerd wordt. Dergelijke oplossingen halen nooit punten en voor bijzonder slechte oplossingen worden punten afgetrokken!

1. **Zoekbomen 2.5 pt**

   - Pas de bewerking *"verwijder sleutel 11"* toe op de volgende semi-splay boom:



- Pas de bewerking *“voeg sleutel 11 toe”* toe op de volgende 2-3 boom:



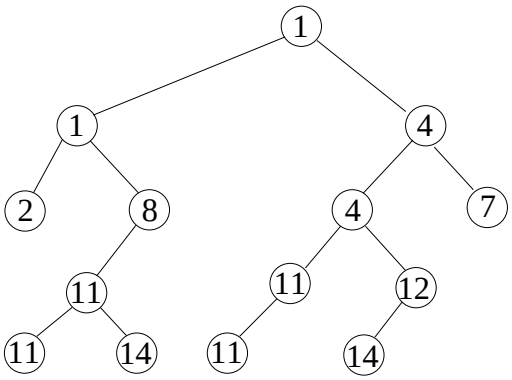
- Je hebt een (niet noodzakelijk binaire) zoekboom met n sleutels, waarvan b in toppen zitten die ten minste één leeg kind hebben.

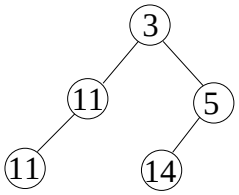
  Toon aan dat $b \leq n \leq 2b - 1$.

  Tip: Stellingen uit de les en de oefeningen kunnen hier nuttig zijn...

2. **Heaps 3.25 pt**

- Merge de volgende twee leftist heaps:





- Wij noemen een binaire boom goed gebalanceerd als het verschil tussen het kortste en het langste pad van de wortel naar een blad klein is en slecht gebalanceerd als dat verschil groot is.

  Welke bomen zijn efficiënter als leftist heaps – goed of slecht gebalanceerde binaire bomen? Geef uitleg.

- Geef drie skew heaps A, B, C met paarsgewijs verschillende sleutels zodat als deze heaps in de volgorde eerst A met B en dan het resultaat met C merget en je doet deze twee bewerkingen één keer beide met de recursieve skew merge bewerking en één keer beide met de gewone skew merge bewerking, het resultaat na de recursieve skew merge bewerking een korter rechterpad heeft dan het resultaat na de gewone skew merge bewerking.

  Pas de bewerkingen ook toe om te tonen dat jouw voorbeeld aan de vereisten voldoet.

- Stel dat je een binomiale heap hebt, die maar 1 binomiale boom bevat en die heeft diepte $k > 0$.

  Bewijs expliciet: Als je het kleinste element uit deze wachtlijn verwijdert, is het resultaat een binomiale heap die voor elke diepte $0, \ldots, k-1$ precies 1 binomiale boom bevat.

  Tip: Als je een *"bewijs"* hebt dat de definitie van een binomiale boom helemaal niet gebruikt, dan is er iets mis...

3. **Inpakalgoritmen 1 pt**

In het volgende gebruiken wij dat je in een zoekboom niet alleen naar een vaste sleutel s kan zoeken en die verwijderen, maar ook naar de grootste sleutel die kleiner dan of gelijk aan s is. Natuurlijk is het mogelijk dat ook hier `niet gevonden` wordt teruggegeven als alle gewichten groter dan s zijn. Als je met een rood-zwart boom werkt, is deze bewerking ook in tijd $O(\log n)$ mogelijk als er n sleutels in de boom zitten.

Gegeven een reeks $g_1, \ldots, g_k$ van gewichten die allemaal ten hoogste 1 zijn. Het volgende algoritme berekent een plaatsing van gewichten op vrachtwagens in tijd $O(k \log k)$. Welke uit de les gekende plaatsing is het? Bewijs dat jouw antwoord juist is – een antwoord zonder bewijs telt niet mee.

```
Maak een rood-zwart boom B aan die alle gewichten bevat.
Maak een lege lijst L aan (voor gevulde vrachtwagens).
Maak een lege vrachtwagen V aan.

Herhaal totdat B leeg is:
  { Stel r gelijk aan de vrije ruimte op V.
    Zoek het grootste gewicht g in B met g<=r.
    Als g gevonden wordt
            { verwijder g uit B en voeg g aan V toe }
    anders
            { Voeg V aan L toe
              en maak een nieuwe lege vrachtwagen V aan.
            }
   }
Als V niet leeg is, voeg die aan L toe.
```

4. **Geamortiseerde complexiteit 2 pt**

   Een toepassing werkt met een leftist heap en er wordt vastgesteld dat de prestatie niet zo goed is als je zou verwachten. Een analyse toont, dat de oorzaak veel cache-misses zijn. Tijdens het toevoegen en verwijderen worden de pointers natuurlijk gewijzigd en blijkbaar gebeurt het, dat de kinderen van toppen vaak ergens anders in het geheugen zitten dan hun ouders.

   Als oplossing werk je met een leftist heap waar de bogen twee kleuren hebben: groen (voor "*vemoedelijk goed*" – het kind zit dicht in het geheugen) en rood (voor "*mogelijk slecht*" – misschien zit het kind ergens anders in het geheugen). Elke keer dat je een nieuwe top t als kind van t' toevoegt of bij het mergen het kind t van een top t' verandert, is de boog van t' naar t achteraf rood. Als 90% van de bogen rood zijn, bouw je een heap met identieke structuur in het geheugen op. Je gebruikt een heuristiek voor het plaatsen van de toppen in het geheugen, die voor weinig cache-misses zorgt als je van een ouder naar zijn kind gaat. Alle bogen in de nieuw in het geheugen geplaatste heap zijn groen. Dit plaatsen vraagt tijd $O(s)$ als s het aantal sleutels in de leftist heap is.

   Je hebt voor deze heap drie bewerkingen:

   - toevoegen van een nieuwe sleutel
   - verwijderen van de kleinste sleutel
   - opzoeken van de kleinste sleutel

   De kost van een bewerking is altijd het aantal bezochte toppen plus – in het geval dat de heap in het geheugen opnieuw geschikt wordt – het aantal sleutels in de heap. In het geval van het opzoeken van de kleinste sleutel worden geen pointers gewijzigd en dus wordt daar de heap ook nooit opnieuw gebouwd.

   Voor de volgende vragen is de $O()$-notatie voldoende.

   - Wat is de duurste mogelijke bewerking in een reeks van n bewerkingen op een initieel lege gekleurde leftist heap? Hoe duur is deze bewerking?

- Wat is de geamortiseerde kost van een bewerking in een reeks van n bewerkingen op een initieel lege gekleurde leftist heap? Misschien is de potentiaalmethode waarbij de potentiaal rekening houdt met het aantal rode bogen een goede keuze, maar je mag zelf kiezen welke methode je gebruikt om te bewijzen dat jouw antwoord juist is.

5. **Dynamisch programmeren 1.75 pt**

   Gegeven twee reeksen $A = a_1, \ldots a_n$ en $B = b_1, \ldots b_m$ van natuurlijke getallen. Wij zeggen dat een reeks $c_1, \ldots, c_k$ een deelreeks van A is, als er indices $i_1 < i_2 < \cdots < i_k$ zijn zodat voor $1 \leq j \leq k$ geldt dat $c_j = a_{i_j}$ – en analoog voor B. Gezocht is nu een reeks $c_1, \ldots, c_k$, die deelreeks van A **en** B is en waarvoor $\sum_{i=1}^{k} c_i$ maximaal is. Je hoeft als resultaat niet de reeks $c_1, \ldots, c_k$ te geven, maar alleen de som $\sum_{i=1}^{k} c_i$.

   Geef de pseudocode voor een algoritme met dynamisch programmeren dat dit probleem in tijd $O(n * m)$ kan oplossen. Geef ook voldoende uitleg.

   Tip: Kijk misschien niet alleen naar de reeksen $A = a_1, \ldots a_n$ en $B = b_1, \ldots b_m$, maar ook naar reeksen die alleen uit de eerste i getallen van A en de eerste j getallen van B bestaan.

6. **Gretige heuristieken 1 pt**

   In een graaf *G* wordt een zo groot mogelijke geïnduceerde deelboom gezocht. Dat is een boom *B*, die een deelgraaf van *G* is, en waarvoor geen bogen tussen toppen van *B* bestaan die niet ook in *B* zijn. Het is een probleem waarvan gekend is dat het heel moeilijk is (NP-compleet) – het lijkt dus zinvol heuristieken te gebruiken.

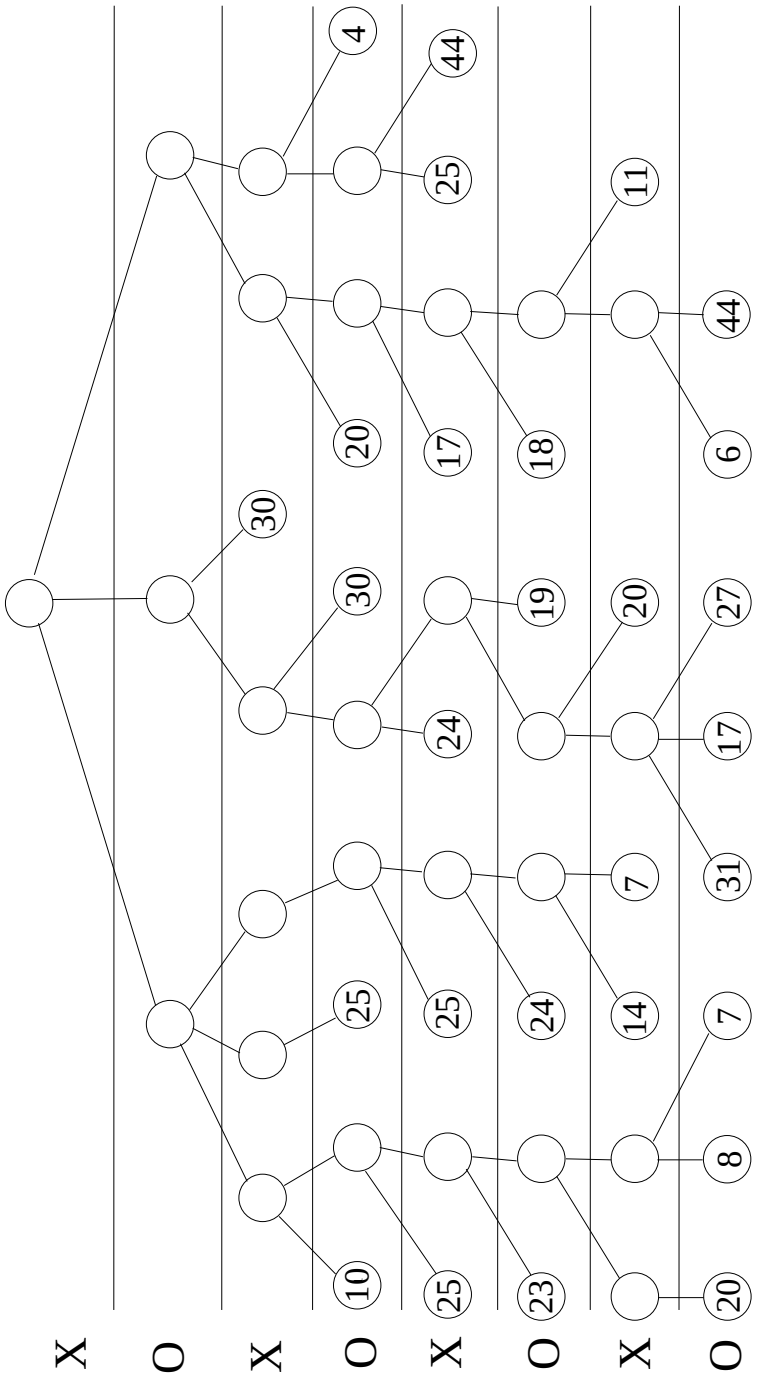
   Geef de pseudocode van een gretig algoritme voor dit probleem waarvan je denkt dat het in veel gevallen goed zal presteren. Leg uit waarom jouw algoritme *gretig* is en waarom je denkt dat het vaak goed zal presteren.

7. **Spelbomen 1.75 pt**

- X en O spelen een spelletje op een zoekboom Z waar de sleutels natuurlijke getallen zijn. Ze bouwen een pad van de wortel naar een blad. X kiest op de even diepten naar welk kind gegaan wordt en O kiest op de oneven diepten. Als een blad bereikt wordt, worden de sleutels op het pad opgeteld. Als de som s even is, betaalt O s cent aan X, anders betaald X s cent aan O.

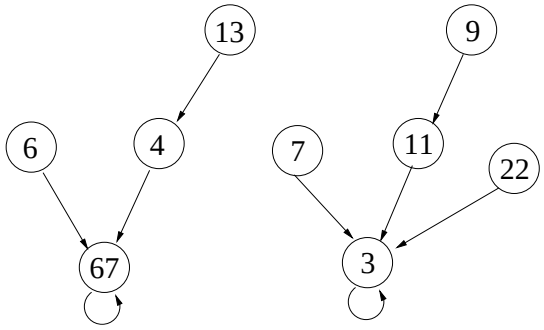
  Teken de spelboom voor het geval dat Z de complete binaire boom met diepte 2 is die de sleutels $1, 3, 4, 6, 7, 8, 10$ bevat. Je hoeft de waarde van het spel niet te berekenen.

- 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.



8. **Verzamelingen 1.25 pt**

- Pas de bewerking *"union(4,9)"* toe op de volgende union-find datastructuur. Gebruik union-by-size en path compression.



- Geef een reeks van n union-find bewerkingen die gebruik maken van union-by-size maar niet van path compression zodat de reeks van bewerkingen als ze toegepast wordt op een universum met in het begin allemaal geïsoleerde elementen een geamortiseerde kost van $\Theta(\log n)$ per bewerking heeft. Toon aan dat jouw reeks deze eigenschap heeft.

9. **Gerandomiseerde algoritmen 1.5 pt**

- In een computernetwerk heb je 200 computers. Op elk van de computers moet één van 3 datasets d_1, d_2, d_3 opgeslagen worden. Het doel is de data zo te verdelen dat elke computer elke dataset van een computer kan halen die op afstand ten hoogste 2 in het netwerk zit.

  Je weet dat elke computer ten minste 15 computers op afstand 1 of 2 heeft.

  Geef de pseudocode van een Las Vegas algoritme om een toekenning van datasets te vinden die aan de vereisten voldoet. Bewijs dat jouw algoritme een toekenning met de gewenste eigenschappen kan vinden.

  Tip: $\frac{2}{3}^{16} < 0.0016$.

- Voor veel van de problemen waar je een Las Vegas algoritme kan toepassen, kan je ook een branch en bound algoritme toepassen (bv ook bij het algoritme in punt 1 van vraag 8). Wanneer zou je de voorkeur geven aan een Las Vegas algoritme en wanneer aan een branch and bound algoritme? Geef redenen voor jouw keuze.

NOG NIET OMDRAAIEN !