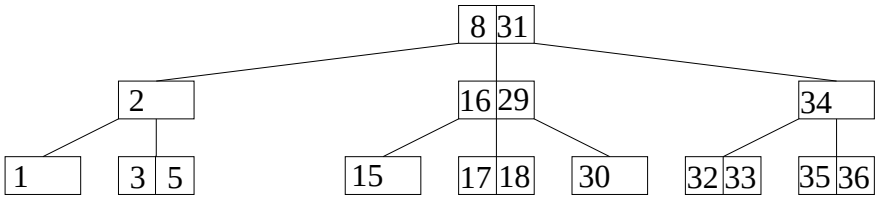
Examen Datastructuren en Algoritmen II

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!

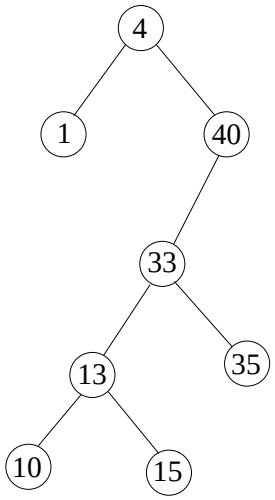
1. **Zoekbomen 2.25 pt**

- Voeg eerst de sleutel 6 en dan de sleutel 19 toe aan de volgende 2-3 boom.



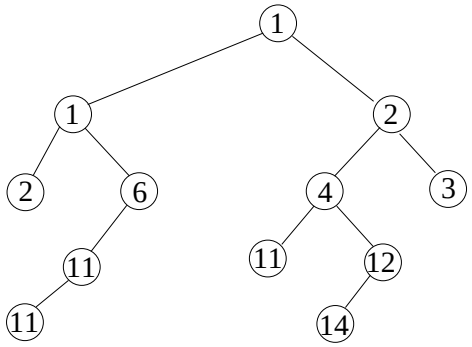
- Bewijs één van de volgende twee uitspraken. Met *linkerdeelboom* en *rechterdeelboom* is daarbij de deelboom direct links of direct rechts van de sleutel bedoeld en niet bv. de deelboom helemaal links als het om de tweede sleutel in de top gaat.
  - Er bestaat een constante c zodat voor alle 2-3 bomen T en alle sleutels $x \in T$ met l toppen in de linkerdeelboom en r toppen in de rechterdeelboom van x geldt $l \leq c * r$.
  - Er bestaat geen constante c zodat voor alle 2-3 bomen T en alle sleutels $x \in T$ met l toppen in de linkerdeelboom en r toppen in de rechterdeelboom van x geldt $l \leq c * r$.

- Voeg eerst 12 toe aan de volgende semi-splay boom en verwijder dan 33.



2. **Heaps 2 pt**

- Verwijder het kleinste element uit de volgende leftist heap.



- Geef een redenering die aantoont dat het volgende geldt:

  Stel dat A, B twee skew heaps zijn waarin geen 2 sleutels gelijk zijn. Als C_r het resultaat van een recursieve skew merge bewerking is en C_n van een niet recursieve skew merge bewerking van deze twee heaps, dan zijn de rechterpaden van C_r en C_n even lang.

- Geef een reeks van verschillende sleutels zodat als je de sleutels in deze volgorde toevoegt aan een initieel lege skew heap het resultaat van de recursieve skew merge bewerkingen verschilt van dat van de niet recursieve skew merge bewerkingen. Toon expliciet de bewerkingen.

3. **Geamortiseerde complexiteit 1.5 pt**

Het volgende bewijs is **fout**. Beschrijf en verbeter de fout. Gebruik de gegeven potentiaal maar maak een juiste analyse.

De datastructuur waarmee wij werken is een gelinkte lijst waarin de getallen gesorteerd opgeslaan zijn. Het grootste element is het eerste element in de lijst. Als een nieuw element x toegevoegd moet worden, dan wordt `lijst = plaats(lijst,x)` toegepast. De pseudocode voor `plaats` is:

```
plaats(element,x)
{
if (element is leeg)
  { maak een nieuw element n_e aan met sleutel(n_e)=x en
                                 volgend_element(n_e)=leeg
    return n_e }
else
  {
    if (sleutel(element)>x)
     { volgend_element(element)=plaats(volgend_element,x)
       return element
     }
    else
     { maak een nieuw element n_e aan met sleutel(n_e)=x en
                                  volgend_element(n_e)=element
       return n_e
     }
  }
}
```

De kost van een toevoegbewerking is (op een constante na) het aantal vergelijkingen. In een lijst met n elementen kan dat dus in het slechtste geval n zijn. Wij zullen nu tonen dat de geamortiseerde kost van een reeks van n toevoegbewerkingen op een initieel lege lijst $O(n)$ is, dus constant per bewerking.

Wij gebruiken de potentiaalmethode en definieren $\Phi(D) = -(m^2/2) + (3/2)m$ waarbij m het aantal elementen in de lijst is.

Als wij nu een element tot de datastructuur D met m elementen toevoegen en het resultaat is D' dan geldt

$\Phi(D') - \Phi(D) = -(m+1)^2/2 + \frac{3}{2}(m+1) + (m^2)/2 - \frac{3}{2}m =$
$-(m^2+2m+1)/2 + \frac{3}{2}m + \frac{3}{2} + m^2/2 - \frac{3}{2}m = -m+1$

Als wij nu de tabel opstellen, krijgen wij

	echte kost	gewijzigde kost
1 element toevoegen	aantal k vergelijkingen	$k+\Phi(D')-\Phi(D)=$ $k-m+1\leq 1$

De laatste ongelijkheid volgt daarbij omdat wij nooit meer dan m vergelijkingen kunnen hebben voor een lijst met m elementen, dus $k \leq m$. De geamortiseerde kost van een reeks van n toevoegbewerkingen op een initieel lege dergelijke lijst is dus $O(n)$ – of $O(1)$ per bewerking.

4. **Dynamisch programmeren 2.5 pt**

   Een palindroom (of *keerwoord*) is een string $s[0], \ldots, s[n]$ zodat $s[i] = s[n-i]$ voor alle $0 \leq i \leq n$, dus bv. *daad, droomoord, etc.*.

   De taak is nu in een gegeven string $t[0], \ldots, t[n]$ een deelstring van maximale lengte te vinden die een palindroom is. Elke string bevat natuurlijk een palindroom, omdat strings met lengte 1 triviale palindromen zijn.

   Geef een algoritme dat gebruik maakt van dynamisch programmeren en in tijd $O(n^2)$ draait. Geef de pseudocode en voldoende uitleg.

5. **Offline inpakalgoritmen 1 pt**

   In de les hebben we bewezen, dat *first fit dalend* en *best fit dalend* ten hoogste $\lceil 4/3m \rceil$ vrachtwagens gebruiken als m het minimaal mogelijke aantal is.

   Toon aan dat er een constante $c > 1$ bestaat zodat er voor elke n_0 een $n \geq n_0$ en een reeks $g_1, \ldots, g_n$ van gewichten bestaan zodat first fit dalend en best fit dalend ten minste $c * m$ vrachtwagens gebruiken.

   Verspil geen tijd om te proberen een reeks te vinden die met een constante dicht bij $4/3$ werkt – de bewezen grens was namelijk niet optimaal en zo'n reeks bestaat dus niet. Als de constante iets groter dan 1 is, is dat voldoende voor deze oefening!

6. **Gretige heuristieken 1.5 pt**

Een *matching* of *koppeling* in een graaf $G = (V,E)$ is een deelverzameling M van de boogverzameling E, zodat geen twee bogen in M een gemeenschappelijke top hebben.

- Geef een gretig algoritme dat probeert een maximale matching te construeren en gegarandeerd een matching vindt met grootte $c * m(G)$ waarbij $c > 0$ een constante is en $m(G)$ de maximale grootte van een matching in G. Geef expliciet de eigenschap die het volgens jou tot een gretig algoritme maakt.

- Bewijs dat het algoritme altijd een matching van grootte $c * m(G)$ vindt (voor een constante $c > 0$) en geef de waarde van c voor jouw algoritme.

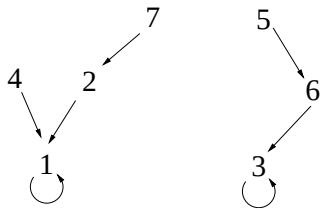
7. **α–β–snoeien 1.5 pt**

- Bewijs de volgende uitspraak of geef een tegenvoorbeeld:
  Als *t* een top in een spelboom is waar het maximum van de waarden van de kinderen wordt gezocht en die op een constant pad ligt en de functie `get_waarde(t)` uit de les die α–β–snoeien implementeert, wordt voor deze top opgeroepen, dan geeft de functie voor deze top altijd de exacte waarde van de top terug.

8. **Verzamelingen 2 pt**

- Bewijs dat een boom met diepte k in een union-find datastructuur die met union by size werd opgebouwd ten minste 2^k elementen bevat.

- Pas de relatie $7 \equiv 5$ toe op de volgende union-find datastructuur. Gebruik union by size met path compression.



- Ons universum is de verzameling $\{0,\ldots,n\}$ met $n < 60$ en wij werken met een computer waar de lengte van een woord 64 bit is. M en M' zijn twee verzamelingen die als bitvectoren voorgesteld zijn en i is een element van $\{0,\ldots,n\}$. Beschrijf (zoals in de les) zo efficiënt mogelijke uitdrukkingen die het volgende betekenen:
  - Er is geen element $x < 30$ dat in M en M' zit.
  - Het element i is een element van beide verzamelingen M, M'.
  - Elk element $0 \leq j < 60$ komt in precies één van de verzamelingen M, M' voor.

9. **Gerandomiseerde algoritmen 1.75 pt**

- In een gegeven complete binaire zoekboom van (grote) diepte *d* zijn records in de bladeren volgens een zekere orderelatie opgeslaan – maar die ken je niet. Je zoekt een record met een zekere eigenschap (bv: record behoort tot een huisdier dat tussen 2 en 4 jaren oud is). Je weet wel, dat 10% van de records deze eigenschap hebben, maar niet waar die zitten.
  - Geef een Las-vegas algoritme voor dit probleem.

Voor de volgende 3 vragen moet er geen redenering gegeven worden **waarom** jouw antwoord juist is:

* Hoe duur is één keer uitvoeren van jouw algoritme als er *n* records in de boom zitten?

* Hoe groot is de kans het algoritme meer dan 10 keer te moeten uitvoren?

    - Hoe groot is de kans dat het algoritme oneindig blijft draaien?

– Waarom is hier een Las-vegas algoritme een betere keuze dan gewoon recursief de bladeren te doorzoeken?

- In de les hadden wij de volgende definitie:

  Gegeven een oneven getal x. Dan kunnen wij $x-1$ schrijven als $x-1=2^t u$ met een oneven getal u en $t \geq 1$.
  Als nu $b \in \{1,\ldots,x-1\}$ en
  $b^u (\mod x) \notin \{1,-1\}$ **en** $\forall 0 < s < t : b^{2^s u} \not\equiv -1 (\mod x)$
  dan noemen wij b een *speciale getuige* voor x.

  Bovendien hebben wij gezegd dat voor een oneven getal $x > 3$ dat geen priemgetal is ten minste $\frac{3}{4}$ van de getallen $1,\ldots,x-1$ getuigen zijn. Stel nu dat x wel een priemgetal is. Welke fractie van de getallen $1,\ldots,x-1$ zijn dan getuigen? Bewijs dat jouw antwoord juist is.
  Tip: met de resultaten uit het hoofdstuk over de Miller Rabin priemgetallentest – die natuurlijk gebruikt mogen worden – is dat geen moeilijke oefening...

NOG NIET OMDRAAIEN !