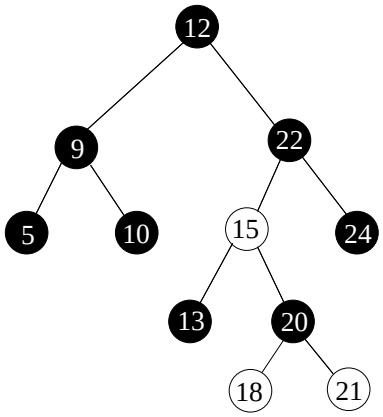
Examen Datastructuren en Algoritmen II

Naam : .....

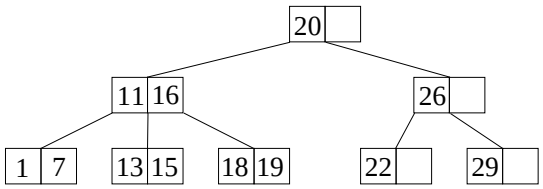
- Lees de hele oefening zorgvuldig voordat je ze begint op te lossen!
  Als je niet goed verstaat wat de vraag of taak is, vraag het aan de lesgever! Voor oefeningen die fout verstaan zijn, kunnen geen punten gegeven worden!
- Schrijf leesbaar. Oplossingen die niet leesbaar zijn, kunnen ook niet beoordeeld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

   - Voeg sleutel 19 toe aan de volgende rood-zwart boom. Gebruik voor het herbalanceren de manier die nooit tot conflicten met de kinderen van de vervangende boom leidt.



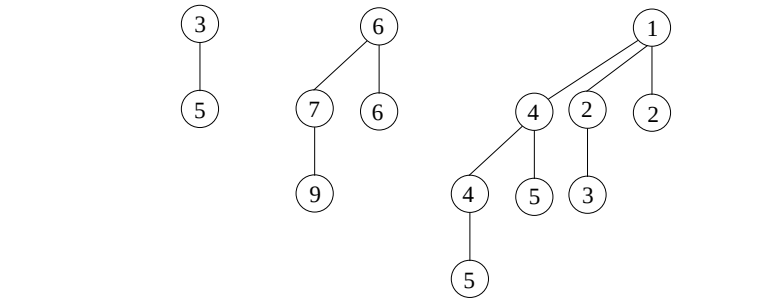
- Verwijder sleutel 22 uit de volgende 2-3 boom

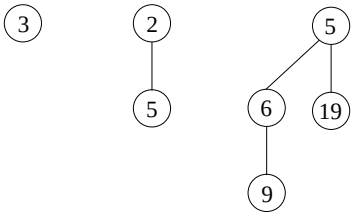


- Stel dat B een semi-splay boom is, die tot en met diepte d compleet is. Voor welke d bestaat er gegarandeerd een reeks van semi-splay toevoeg- en opzoek-bewerkingen toegepast op B zodat de boom die het resultaat van deze bewerkingen is zelfs tot en met diepte 2 niet meer compleet is? Bewijs dat jouw antwoord juist is.

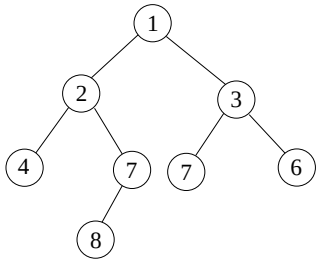
2. **Heaps 3.25 pt**

- Merge de volgende twee binomiale prioriteitswachtlijnen





- Verwijder de kleinste sleutel uit de volgende skew heap. Gebruik de niet recursieve skew merge bewerking.



- Als wij online werken, kan het gebeuren dat voor een binaire hoop of een leftist heap $\Theta(n * \log n)$ stappen nodig zijn om n sleutels aan een initieel lege heap toe te voegen. Een binomiale wachtlijn heeft ook online maar $\Theta(n)$ stappen nodig. Als wij offline werken – dus alle n sleutels onmiddelijk hebben – kan je ook een binaire hoop in tijd $\Theta(n)$ opbouwen (dat weten jullie uit Algoritmen en Datastructuren 1).

  Toon aan: Als je offline werkt – dus alle n sleutels onmiddelijk hebt – kan je ook een leftist heap in tijd $\Theta(n)$ opbouwen.

  Tip: Al genoemde resultaten zouden nuttig kunnen zijn...

- Een binaire boom met sleutels in de toppen heet een *leftweight* heap, als de sleutel in een top nooit groter is dan een sleutel in zijn kinderen en voor elke top ten minste even veel toppen in zijn linkerkinderboom zitten als in zijn rechterkinderboom.

  Geef en bewijs een scherpe bovengrens voor de lengte van het rechterpad van een leftweight heap met n toppen. Geef ook een voorbeeld dat toont dat de grens scherp is.

3. **Geamortiseerde complexiteit 2.5 pt**

- Voor een skew heap noemen wij een top *slecht* als er meer toppen in zijn rechterdeelboom zitten dan in zijn linkerdeelboom en anders *goed*. De potentiaal van een skew heap D is dan

  $\Phi(D) = |\{v \in D | v \text{ is slecht } \}|$

  Kijk nu naar de bewerking die van alle toppen op het rechterpad (op de laatste na) de kinderen wisselt. Stel dat je deze bewerking op een skew heap D toepast en het resultaat is D'. Bereken een bovengrens voor $\Phi(D') - \Phi(D)$ als functie van het aantal goede en slechte toppen op het rechterpad van D.

- Je wilt een zo goed mogelijk gebalanceerde zoekboom opbouwen en daarvoor werk je met een *bufferboom B* van vaste maximale grootte g en een boom T van variabele grootte. Als een element toegevoegd moet worden, dan wordt het altijd eerst aan de bufferboom toegevoegd. Dat heeft constante kost C_1, waarbij C_1 inderdaad $O(\log g)$ is als je de bufferboom bv. als een rood-zwart boom implementeert. Als de bufferboom volzit wordt die met T gemerged tot een zo goed mogelijk gebalanceerde binaire zoekboom (de diepte van de bladeren verschilt dus ten hoogste met 1). Dit mergen heeft kost $C_2 * m$ waarbij m het aantal sleutels in B en T samen is. Als je een sleutel zoekt, moet je die dus zowel in T als in B zoeken. Samen heeft dat kost $C_3 * \log m + C_4$ waarbij m het aantal sleutels in T is. Hierbij zijn C_1, C_2, C_3 en C_4 constanten.

  Wat is de geamortiseerde kost ($O()$-notatie) van een reeks van n toevoeg- of opzoekoperaties op initieel lege bomen B en T? Kies de methode die hiervoor het meest geschikt is en toon aan dat jouw grens een goede grens is.

4. **De waarde van een spel 1.5 pt**

   Dit is een spel waar iedereen wint:

   Gegeven een rij $m_1, \ldots, m_n$ van n munten, waarbij m_i de waarde van de munt op positie i is. Elke speler neemt afwisselend ofwel de eerste ofwel de laatste munt totdat er geen munten meer zijn. Speler X begint en dan komt speler O. Op het einde mag elke speler zijn munten houden.

   Dit is dus een spel dat niet aan de vereisten voldoet die wij voor spelen hadden die je met spelbomen kon modelleren.

   - Modelleer dit spel zo dat het aan de vereisten voldoet. Dat betekent: geef een spel dat wel aan de vereisten voldoet zodat je uit een optimale strategie voor dit gewijzigde spel een optimale strategie voor X in het originele spel kan afleiden.

- – **Als je het eerste deel hebt opgelost:** Teken de spelboom en bereken de waarde voor jouw gewijzigd spel voor de reeks $m_1, \ldots, m_4 = 1, 1, 5, 2$
  – **Als je het eerste deel niet hebt opgelost:** Teken de boom die alle mogelijke volgordes van zetten toont voor de reeks $m_1, \ldots, m_4 = 1, 1, 5, 2$. Als je X was: op welke winst zou je ten minste rekenen en waarom?

5. **Dynamisch programmeren 2 pt**

Je hebt een rij van n steden $s_1, \ldots, s_n$ en twee personen die deze steden moeten bezoeken. Als een persoon een stad s_i bezoekt, mag die achteraf wel geen stad s_j met $j \leq i$ meer bezoeken. Voor $n = 6$ zou het dus in orde zijn als persoon A zijn deel van de steden in volgorde s_1, s_3, s_5, s_6 bezoekt en persoon B de andere steden in de volgorde s_2, s_4, maar het zou niet in orde zijn als A de steden in volgorde s_3, s_1, s_5, s_6 zou bezoeken. Voor elk paar (s_i, s_j) van steden met $i < j$ heb je de afstand $d(s_i, s_j) \geq 1$.

De taak is nu als volgt: persoon A vertrekt in s_1 en persoon B vertrekt in s_2. Gezocht is een partitie van de steden zodat als A en B hun steden in volgorde bezoeken de som van de afstanden minimaal is.

Geef de pseudocode van een algoritme dat de minimale som van afstanden berekent. Geef voldoende uitleg om de pseudocode te verstaan en om aan te tonen dat het algoritme juist is.

Tip: denk er eerst over na welke informatie over deeloplossingen voor de steden $s_1, \ldots, s_{n-1}$ je nodig hebt om te kunnen beslissen wie het best s_n bezoekt. Misschien voldoet een voor de hand liggende 1-dimensionale tabel niet, maar is een 2-dimensionale tabel wel nuttig...

6. **Online algoritmen 1 pt**

Je hebt een online versie van hetzelfde probleem als in de vorige oefening – het wordt hier gewoon voor de duidelijkheid herhaald en om het op dezelfde pagina te hebben:

Je hebt een rij van n steden $s_1, \ldots, s_n$ met gegeven afstanden $d(s_i, s_j) \geq 1$ voor $1 \leq i < j \leq n$ en twee personen die deze steden moeten bezoeken. Als een persoon een stad s_i bezoekt, mag die achteraf wel geen stad s_j met $j \leq i$ meer bezoeken. Voor $n = 6$ zou het dus in orde zijn als persoon A zijn deel van de steden in volgorde s_1, s_3, s_5, s_6 bezoekt en persoon B de andere steden in de volgorde s_2, s_4, maar het zou niet in orde zijn als A de steden in volgorde s_3, s_1, s_5, s_6 zou bezoeken.

De taak is nu als volgt: persoon A vertrekt in s_1 en persoon B vertrekt in s_2. Gezocht is een partitie van de steden zodat als A en B hun steden in volgorde bezoeken de som van de afstanden zo kort mogelijk is – maar je moet het probleem online oplossen: de personen beginnen al met hun tocht voordat alle steden die ze moeten bezoeken binnen zijn. Elke keer als een nieuwe stad gekend is waar ze naartoe moeten, wordt één persoon onmiddelijk verwittigd welke stad hij na de steden die hij al kent nog moet bezoeken.

- Toon aan: voor elk online algoritme, elke $m \in \mathbb{N}$ en elke constante c bestaat er een reeks van $n \geq m$ steden en afstanden $d(s_i, s_j) \geq 1$ voor $1 \leq i < j \leq n$, zodat de door het online algoritme gegeven oplossing ten minste c keer slechter is dan de beste oplossing als alle steden al op voorhand gekend zijn.

7. **Gretige algoritmen 1.25 pt**

Een probleem dat op het inpakprobleem lijkt: Je hebt k vrachtwagens met capaciteit 1 en een rij pakketten met gewichten $g_1, \ldots, g_n$ die allemaal ten hoogste 1 zijn. Er zijn veel meer pakketten dan je op de vrachtwagens kan plaatsen en je wordt betaald per pakket – het doet er dus niet toe of je een licht pakket vervoert of een zwaar pakket – je krijgt hetzelfde betaald. Een gretig algoritme zou dus bv. als volgt kunnen werken:

Sorteer de pakketten in stijgende volgorde van de gewichten $g_1, \ldots, g_n$. Plaats dan zoveel mogelijk van de gewichten door ze in deze volgorde goed op de vrachtwagens te verdelen: plaats een gewicht altijd zo op een beschikbare vrachtwagen, dat de vrije ruimte op deze vrachtwagen zo groot mogelijk is (zo wordt de kans groter dat de grotere gewichten die nog komen nog geplaatst kunnen worden).

- Bewijs: als het mogelijk is om m gewichten te vervoeren, dan is er een oplossing (ook al wordt die misschien niet door dit algoritme gevonden) die de eerste m pakketten van de gesorteerde rij gebruikt.

- Bewijs de volgende stelling met de grootste constante $C > 1$ waarvoor jij de uitspraak kan bewijzen.

  Stelling: Voor elke $k > 1$ bestaan er een getal m en pakketten met gewichten $g_1, \ldots, g_n$, zodat m pakketten maximaal vervoerd kunnen worden, maar de oplossing van het gretige algoritme maar $\frac{m}{C}$ gewichten geeft.

8. **Verzamelingen 1 pt**

Een uittreksel uit de man-pages van `ffs()`:

```
DESCRIPTION
       The  ffs()  function  returns  the position of the first
       (least significant) bit set in the word i.  The least
       significant bit is position 1 and the most significant
       position is, for example,  32  or  64.

RETURN VALUE
       This function return the position of the first bit set,
       or 0 if no bits are set in i.
```

Een graaf met toppenverzameling $\{0,\ldots,n\}$ met $n < 64$ is gegeven door de integer array `g[]`, waarbij voor $0 \leq i \leq n$ de integer $g[i]$ de verzameling van buren van top i als bitvector bevat. Geef de pseudocode van een programma dat alle driehoeken i, j, k met $i < j < k$ van de graaf uitschrijft en waarvan jij denkt dat het goed presteert. Gebruik de bewerkingen voor bitvectoren uit de les.

9. **Gerandomiseerde algoritmen 1 pt**

10. Gegeven een rij van $2 * n$ verschillende getallen $x_1, \ldots, x_{2*n}$. Deze rij is niet gesorteerd, maar je zoekt twee getallen $x_<$ en $x_>$ uit de rij zodat er ten minste n kleinere getallen dan $x_>$ en ten minste n grotere getallen dan $x_<$ zijn.

    Geef een Monte Carlo algoritme dat in tijd $O(1)$ draait en een resultaat geeft dat ten hoogste met kans $c < 2^{-20}$ fout is. Bewijs dat jouw algoritme deze eigenschap heeft.

NOG NIET OMDRAAIEN !