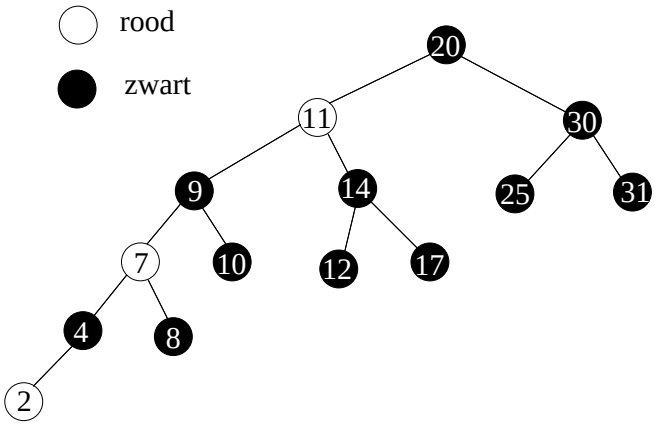
Examen Datastructuren en Algoritmen II

Naam : .....

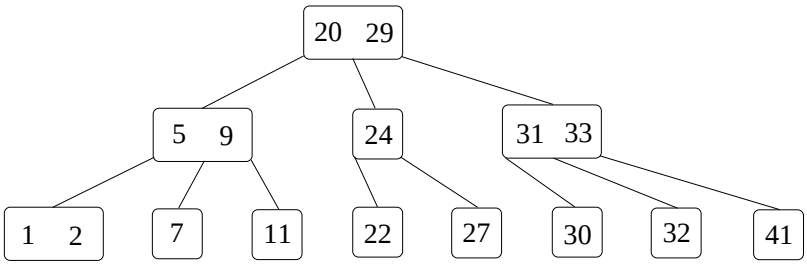
- Lees de hele oefening zorgvuldig voordat je begint ze op te lossen!
  Als je niet goed verstaat wat de vraag of taak is, vraag het aan de lesgever! Voor oefeningen die fout verstaan zijn, kunnen geen punten gegeven worden!
- Schrijf leesbaar. Oplossingen die niet leesbaar zijn, kunnen ook niet beoordeeld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

   - Voeg sleutel 3 toe aan de volgende rood-zwart boom. Gebruik de operaties die geen rekening houden met de kleur van toppen die niet op het pad van de wortel naar de toegevoegde sleutel zitten. Toon alle tussenstappen, maar je hoeft niet voor elke tussenstap de hele boom te tekenen.



- Voeg sleutel 3 toe aan de volgende 2-3 boom. Toon alle tussenstappen, maar je hoeft niet voor elke tussenstap de hele boom te tekenen.



- Een zwakke rood-zwart boom definiëren wij hier als volgt:

  Een zwakke rood-zwart boom is een binaire zoekboom T waarvan de toppen ofwel rood, ofwel zwart gekleurd zijn. Bovendien voldoet T aan de volgende eigenschappen:

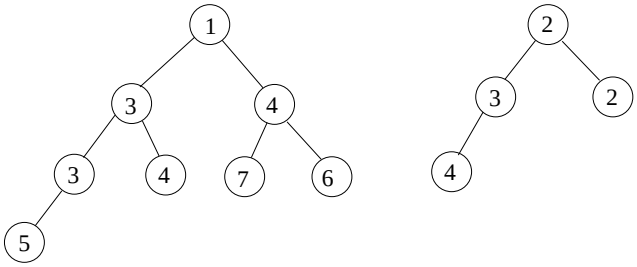
  – De kinderen van een rode top zijn zwart.
  – Elk pad van de wortel naar een NULL-pointer bevat hetzelfde aantal zwarte toppen. Wij schrijven voor dit aantal $z(T)$.

  Wij eisen hier dus bijna hetzelfde als voor een gewone (niet-zwakke) rood-zwart boom, behalve dat de wortel van de boom niet zwart hoeft te zijn.

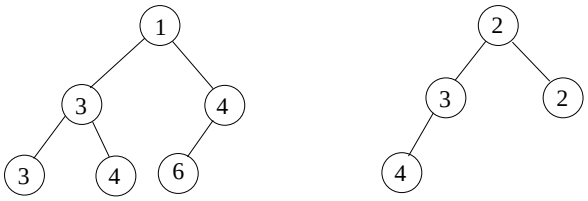
  Bepaal een goede bovengrens voor de diepte van een zwakke rood-zwart boom met n sleutels. Je mag vrij kiezen op welke manier je dat doet – of je het bewijs voor rood-zwart bomen nabootst (met lichte wijzigingen) of resultaten uit de les gebruikt of ...

2. **Heaps 2.75 pt**

- Vul de nulpadlengten in voor de toppen van de volgende twee leftist heaps en merge de heaps dan.



- Merge de volgende twee skew heaps op de niet recursieve manier.



- Voor heaps laten wij altijd toe dat dezelfde sleutel meerdere keren in de heap voorkomt. Leg uit wat het probleem voor de bewerkingen *"een element toevoegen"* en *"verwijder het kleinste element"* voor binomiale wachtlijnen zou zijn, als dubbele sleutels niet toegelaten zijn.

- Geef een gewijzigde vorm van de niet-recursieve skew merge bewerking (die er dus mee begint de rechterpaden te mergen), die als resultaat dezelfde heap oplevert als de recursieve skew merge bewerking uit de les. Omdat voor identieke sleutels de volgorde in de gemergde rechterpaden niet uniek is, mag je veronderstellen dat toppen met dezelfde waarde in dezelfde volgorde in de gemergde rechterpaden opduiken.

3. **De waarde van een spel 0.75 pt**

   Je begint *drie op een rij* met de volgende beginpositie. X kiest eerst. De getallen en lettertekens naast het spelbord staan er alleen zodat jullie in de boom kunnen aangeven welk vakje (bv (a,2)) ingevuld word.

3		O	O
2		X	
1	X	X	O
	a	b	c

   Het spel eindigt zodra voor de eerste keer 3 identieke symbolen op een rij staan (horizontaal, verticaal of diagonaal) of als alle vakjes ingevuld zijn. Als op het moment dat een speler drie op een rij heeft a vakjes ingevuld zijn, wint X a cent als hij het laatste vakje heeft ingevuld – anders verliest hij a cent – het resultaat uit zijn zicht is dus $-a$. Als bij het einde van het spel er geen 3 identieke symbolen op een rij staan, is de winst voor beide spelers 0.

   Teken de spelboom voor dit spel en bereken voor elke top van de boom de waarde van de top.

4. **Geamortiseerde complexiteit 1.5 pt**

   Wij werken met een array die in het begin leeg is en een vaste grootte g heeft. Wij hebben 3 soorten bewerkingen:

   - een element op de eerste vrije plaats toevoegen
   - het laatste element verwijderen
   - een vol array groter maken – om precies te zijn: een array van grootte k waarin k elementen zitten, wordt een array van grootte $2k$

   Door slim gebruik te maken van paging en virtueel geheugen is het mogelijk een array uit te breiden en daarbij ten hoogste een constant aantal C van elementen te moeten kopiëren (waarvoor wij kost C rekenen). Wij willen wel dat de nieuwe elementen met 0 geïnitialiseerd worden, waarvoor wij kost 1 per element rekenen. De kost van een uitbreidbewerking is dus $C+k$ als er k elementen bijkomen. De kost van een toevoegbewerking (zonder uitbreiden) en de kost van een verwijderbewerking zijn 1. Als een array volzit, kan een toevoegbewerking alleen gebeuren, als er eerst een uitbreidbewerking gebeurt.

   - Wat is de duurste bewerking ($\Theta()$-notatie) in een reeks van n toevoeg-, verwijder- en uitbreid-bewerkingen? Geef uitleg.

- Wat is de geamortiseerde kost ($O()$-notatie) van een bewerking in een reeks van n toevoeg-, verwijder- en uitbreidbewerkingen? Geef een bestmogelijke grens. Je kan zelf kiezen op welke manier je de juistheid van de grens bewijst.

5. **Dynamisch programmeren 2 pt**

Een situatie in een restaurant: De vrienden A en B gaan eten. Persoon A moet r_A cent betalen en persoon B moet r_B cent betalen (de prijs is in cent gegeven om gehele getallen te hebben). De gewoonte in het restaurant is dat er maar één rekening per tafel is, dus beslissen de vrienden dat A eerst zijn deel aan B betaalt en B achteraf de hele rekening. De vrienden hebben 5 soorten munten of biljetten ter waarde van w_1 cent, w_2 cent,...,w_5 cent. Je mag veronderstellen dat beide personen genoeg geld hebben – dus: Persoon A heeft totaal $t_A \geq r_A$ cent en persoon B heeft totaal $t_B \geq r_B$ cent. Om precies te zijn heeft persoon A voor $1 \leq i \leq 5$ precies $m_{A,i}$ munten of biljetten ter waarde van w_i en persoon B $m_{B,i}$ munten of biljetten ter waarde van w_i.

De vraag is dus of er een bedrag $x \geq r_A$ is zodat persoon A aan persoon B het bedrag van x kan betalen zodat persoon B met zijn munten $x - r_A$ kan terugbetalen.

Voorbeeld: $w_1 = 20$, $w_2 = 200$, $w_3 = 500$, $w_4 = 2000$, $w_5 = 5000$, A heeft $m_{A,3} = 2$ keer $w_3 = 500$ Cent en $m_{A,4} = 3$ keer $w_4 = 2000$ Cent en moet 3700 Cent betalen. B heeft $m_{B,2} = 10$ keer $w_2 = 200$ Cent en $m_{B,5} = 1$ keer $w_5 = 5000$ Cent. $m_{A,1} = m_{A,2} = m_{A,5} = m_{B,1} = m_{B,3} = m_{AB,4} = 0$. Dan betaalt A $2 * 2000 + 500 = 4500$ Cent aan B en die geeft $4 * 200 = 800$ Cent terug.

Los één van de twee volgende vragen op:

- Geef de pseudocode van een dynamisch programmeren algoritme dat beslist of A een bedrag x aan B kan betalen zodat B het verschil $x - r_A$ terug kan betalen. **1.5 pt**
- Geef de pseudocode van een dynamisch programmeren algoritme dat beslist op welke manier A een bedrag x aan B kan betalen zodat B het verschil $x - r_A$ terug kan betalen en het totale aantal munten dat van eigenaar wisselt minimaal is. Als er zo'n manier bestaat moet het algoritme het minimale aantal munten teruggeven – anders *error*. Het is dus niet noodzakelijk dat het algoritme precies zegt om welke munten het gaat. **2 pt**

6. **Online algoritmen 2.5 pt**

- Gegeven een reeks van n gewichten $1 \geq g_1 \geq g_2 \geq \cdots \geq g_n > \frac{1}{3}$. Bewijs dat First-Fit-dalend deze reeks optimaal plaatst.

- Geef een reeks van gewichten waar First-Fit dalend beter presteert dan Best-Fit dalend. Pas beide algoritmen toe om aan te tonen dat jouw reeks aan deze eigenschap voldoet.

- Wij willen goederen in een opslagplaats plaatsen. Om het eenvoudig te houden, heeft de opslagplaats voor elke $i \in \mathbb{N}, i > 0$ een positie p_i om één pakket op te slaan. De kost om een goed met gewicht g_j op plaats p_i te plaatsen is gelijk aan $g_j * i$. Als ik dus bv. gewichten $3, 4, 2$ heb en ik plaats 3 op positie p_1, 4 op positie p_3 en 2 op positie p_5 is de kost $3*1+4*3+2*5 = 25$. Een optimale plaatsing (met minimale kost) zou kost $4*1+3*2+2*3 = 16$ hebben. Als een gewicht geplaatst is, wordt het niet meer verplaatst. Een online algoritme toegepast op een reeks van gewichten $g_1, \ldots, g_n$ plaatst eerst gewicht g_1 zonder de andere al te kennen, dan g_2, etc.

  Geef een constante $c > 1$ zodat er voor alle m en elk online algoritme A een reeks van ten minste m gewichten bestaat waar het online algoritme een oplossing geeft die ten minste c keer slechter is dan de plaatsing met minimale kost voor deze reeks. Bewijs dat jouw constante deze eigenschap heeft.

  Tips: Probeer niet de optimale constante te vinden – voor dit examen is het voldoende als je het voor om het even welke constante $c > 1$ bewijst. Het is misschien nuttig naar reeksen te kijken met heel veel kleine gewichten en maar twee grote gewichten...

7. **Gretige algoritmen 1 pt**

Je moet k bestanden $f_1, \ldots, f_k$ met gesorteerde sleutels tot 1 groot bestand mergen, waarbij voor $1 \leq i \leq k$ bestand f_i precies b_i sleutels bevat. Je merget elke keer 2 bestanden. De kost om een bestand met x bytes te mergen met een bestand met y bytes is $x+y$. Het doel is om zo lang bestanden te mergen totdat er maar 1 bestand over is.

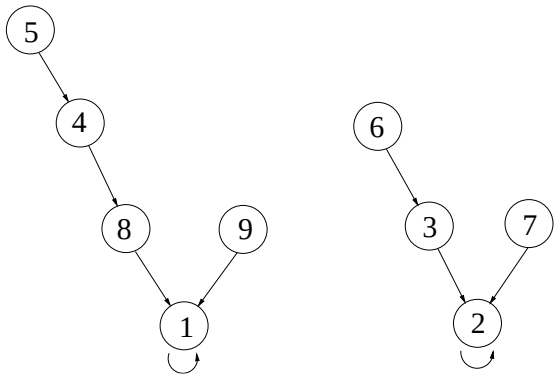
Voorbeeld: je hebt 3 bestanden f_1, f_2, f_3 met $b_1 = 300, b_2 = 7000$ en $b_3 = 20000$ bytes. Als je eerst f_1 en f_2 merget tot een bestand met $b_1 + b_2$ bytes heeft dat kost 7300. Als je die dan merget met bestand f_3 heeft dat kost $(b_1 + b_2) + b_3 = 27300$. Samen is dat 34600. Als je eerst f_1 en f_3 merget (kost 20300) en het resultaat met f_2 (kost 27300) is de totale kost 47600 – dus veel groter.

Geef een efficiënt gretig algoritme dat een volgorde bepaalt om de bestanden te mergen. Geef uitleg waarom je denkt dat jouw algoritme goed presteert en wat het een *gretig* algoritme maakt.

8. **Verzamelingen 1.5 pt**

- Gegeven een universum met elementen $\{1,\ldots,n\}$ en een union-find datastructuur waarin al deze elementen verzamelingen met één element vormen. Geef en bewijs een scherpe bovengrens – geen $O()$-notatie – voor de kost van een duurste union-bewerking in een reeks van $m < n$ union-bewerkingen die union by size en path-compression gebruikt. Als kost rekenen wij het aantal elementen (sleutels) dat tijdens de bewerking bezocht werd.

- Pas de bewerkingen `union(6,9)` en `union(5,7)` in deze volgorde toe op de volgende union-find datstructuur. Gebruik union by size en path compression.



9. **Gerandomiseerde algoritmen 1.5 pt**

- Let op – dit probleem lijkt op het eerste gezicht op het probleem uit de les, maar het is niet helemaal identiek!

  Gegeven een verzameling van computers $V = \{C_1, \ldots, C_n\}$ en k groepen van gebruikers, waarbij sommige gebruikers in meerdere groepen kunnen zitten. Groep i kan computers in de deelverzameling $V_i \subseteq V$ gebruiken. Er is bovendien een grens $r \geq 2$ zodat elke verzameling V_i een grootte van ten minste r heeft.

  Ons doel is op elke computer één van 2 databanken te installeren zodat in elke groep van gebruikers elke databank gebruikt kan worden – of met andere woorden: zodat in elke V_i ten minste één kopie van elke databank is.

  Zou je om dit probleem op te lossen een Las Vegas algoritme of een branch and bound algoritme toepassen? Geef een goede redenering waarom jouw keuze goed is.

- Pas de Miller-Rabin priemgetallentest toe om te testen of 28 een priemgetal is (ook al is 28 even). Neem als toevallig gekozen getal tussen 0 en 28 het getal 9.

NOG NIET OMDRAAIEN !