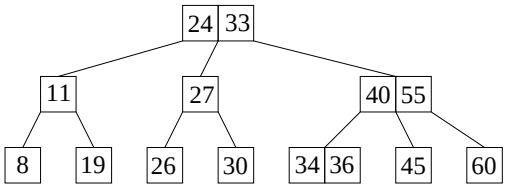
Examen Datastructuren en Algoritmen II

Naam : .....

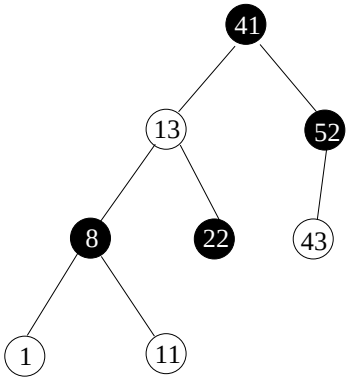
- Lees de hele oefening zorgvuldig voordat je begint ze op te lossen!
  Als je niet goed verstaat wat de vraag of taak is, vraag het aan de lesgever! Voor oefeningen die fout verstaan zijn, kunnen geen punten gegeven worden!
- Schrijf leesbaar. Oplossingen die niet leesbaar zijn, kunnen ook niet beoordeeld 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 37 toe aan de volgende 2-3 boom.



- Voeg sleutel 9 toe aan de volgende rood-zwart boom. Voor het herbalanceren gebruik de manier die het toelaat, soms vroegtijdig met het herbalanceren te stoppen omdat de wortel van de vervangende boom zwart gekleurd kan worden.



- Eén van de motivaties voor de splaybewerking was dat op die manier vaak opgezochte sleutels dichter bij de wortel terechtkomen. Als wij een sleutel verwijderen, is dat natuurlijk geen reden…

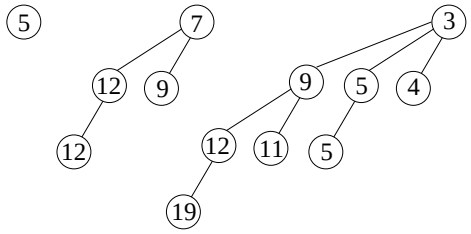
  Stel nu – en alleen voor dit deel – dat wij tijdens het toevoegen en opzoeken splaybewerkingen toepassen, maar tijdens het verwijderen niet.

  Geef een reeks van n bewerkingen die aantoont dat op deze manier de geamortiseerde kost van een reeks van n bewerkingen op een initieel lege boom $\Theta(n)$ per bewerking kan zijn. Geef uitleg.

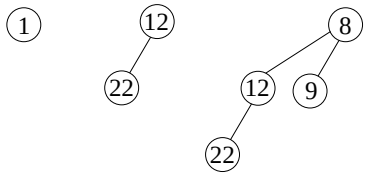
2. **Heaps 2.75 pt**

- Merge de volgende twee binomiale heaps:

Heap 1:



Heap 2:



- Geef de pseudocode voor een recursieve manier om leftist heaps te mergen die – net zoals bij de recursieve manier om skew heaps te mergen – zodra één heap volledig in het rechterpad verwerkt is de rest van de heap waar het rechterpad nog niet volledig overlopen is niet meer overloopt.

  Tip: misschien is het het gemakkelijkste als je van de recursieve skew merge bewerking vertrekt en alleen de nodige wijzigingen doet.

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

3. **Geamortiseerde complexiteit 2 pt**

   In de les hebben wij gezien dat het verwijderen en herbalanceren van rood-zwart bomen misschien niet echt moeilijk is, maar dat er toch veel gevallen zijn waarmee je rekening moet houden. Wij werken nu op een andere manier: als er een sleutel verwijderd wordt, plaats je een *grafsteen* – dat betekent dat je deze sleutel als *niet meer actief* kenmerkt. Als je dan sleutels opzoekt, werk je zoals gewoon, alleen dat als een sleutel met grafsteen wordt opgezocht je *niet aanwezig in de boom* teruggeeft. Als een sleutel wordt toegevoegd die er al met grafsteen inzit, wordt de grafsteen gewoon verwijderd.

   Als je veel verwijderbewerkingen hebt, kan het natuurlijk gebeuren dat je zeer veel meer sleutels met grafstenen hebt dan zonder grafstenen – waardoor de boom inefficiënt wordt. Om dat te voorkomen, wordt altijd als de boom meer dan 50% sleutels met grafstenen bevat de hele boom overlopen en een optimaal gebalanceerde rood-zwart boom met alleen maar de actieve sleutels opgebouwd. In een oefening hebben wij gezien dat dat in tijd $O(s)$ kan met s het aantal sleutels in de oorspronkelijke boom.

   De bewerkingen op deze boom zijn dus de volgende:

   - opzoeken
   - toevoegen
   - verwijderen

   De kost van de opzoekbewerking en de toevoegbewerking is het aantal bezochte toppen. De kost van de verwijderbewerking is het aantal bezochte toppen (op zoek naar de top waar een grafsteen moet worden geplaatst) plus – in het geval dat er meer dan 50% sleutels met grafstenen zijn – het aantal sleutels in de boom (voor het opkuisen).

   - Hoe duur is de duurste bewerking in een reeks van n bewerkingen op een initieel lege rood-zwart boom? De $O()$-notatie is voldoende. Wat is de duurste bewerking?

- Wat is de geamortiseerde kost van een bewerking in een reeks van n bewerkingen op een initieel lege rood-zwart boom? De $O()$-notatie is voldoende. Kies zelf welke methode je gebruikt om te bewijzen dat jouw antwoord juist is, maar misschien is de potentiaalmethode waarbij de potentiaal rekening houdt met het aantal grafstenen in de boom geen slechte keuze.

4. **Online algoritmen 2.75 pt**

- Toon aan dat er voor elke *n* een voorbeeldreeks van gewichten bestaat zodat een optimale oplossing ten minste *n* vrachtwagens gebruikt en waarvoor first fit beter presteert dan first fit dalend.

- Bewijs:

  Voor elk online inpakalgoritme A en elke $k \in \mathbb{N}$ bestaat er een reeks van gewichten waar een optimale oplossing voor het inpakprobleem ten minste k vrachtwagens gebruikt en die door first fit dalend ten minste even goed geplaatst wordt als door A.

- Beschrijf expliciet een algoritme dat best fit dalend implementeert en n gewichten in tijd $O(n \log n)$ kan plaatsen. Leg uit waarom jouw algoritme aan de vereisten voldoet.

  Tip: een zoekboom kan in dit algoritme nuttig blijken…

5. **Gretige heuristieken en branch and bound 2 pt**

In een computernetwerk heb je computers $C_1, \ldots, C_n$ en verschillende verbindingen $e_1, \ldots, e_k$ tussen de computers. Het doel is nu een zo klein mogelijke verzameling $T = \{C_{i_1}, \ldots, C_{i_m}\}$ van computers te vinden die de verbindingen regelmatig kunnen toetsen. Daarvoor is het nodig dat elke verbinding ten minste één eindpunt in T heeft.

- De eerste aanzet is een recursief gretig algoritme. Wij noemen een verbinding gesatureerd als die een eindpunt in T heeft.
  - Begin met $T = \emptyset$.
  - Herhaal totdat elke verbinding gesatureerd is:
    - Voeg een top met de meeste ongesatureerde verbindingen toe aan T.

Bewijs het volgende voor $f = 1.5$:

Voor elk getal g bestaat er een samenhangend netwerk zodat een optimale oplossing $o \geq g$ computers bevat en waar het gretige algoritme een verzameling met ten minste $f \cdot o$ computers construeert.

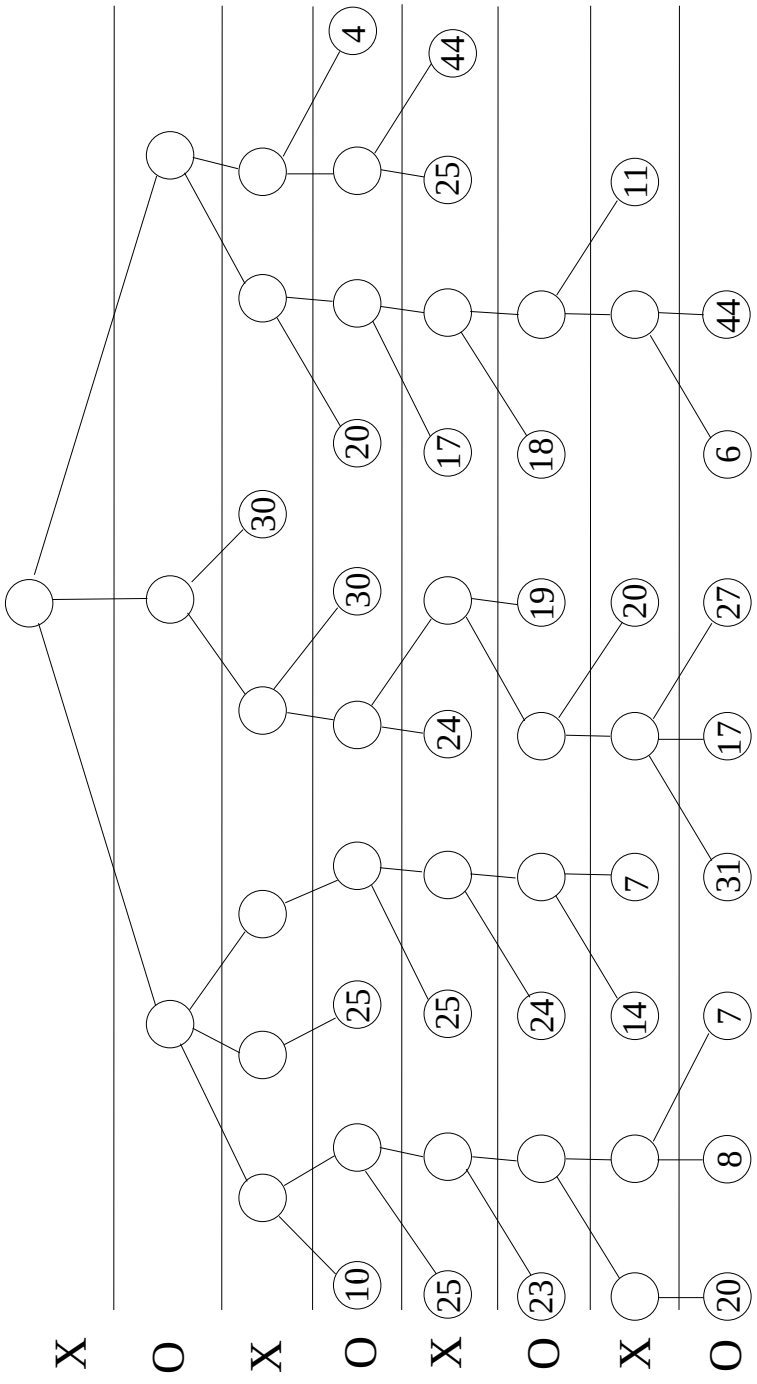
Tip: misschien helpt het aan het museumvoorbeeld in het begin van de lesnota's te denken.

- Geef de pseudocode van een branch and bound algoritme voor dit probleem. Gebruik ten minste één niet triviaal bounding criterium. Als je bv. de verzameling T recursief opbouwt zou het een triviaal criterium zijn als je alleen kijkt of de tot nu toe opgebouwde verzameling al groter is dan de beste die je al hebt.

6. **Spelbomen 1.5 pt**

- Welke eigenschappen heeft een spel nodig zodat het door een spelboom beschreven kan worden die dan door α–β–snoeien zoals gezien geëvalueerd kan worden?

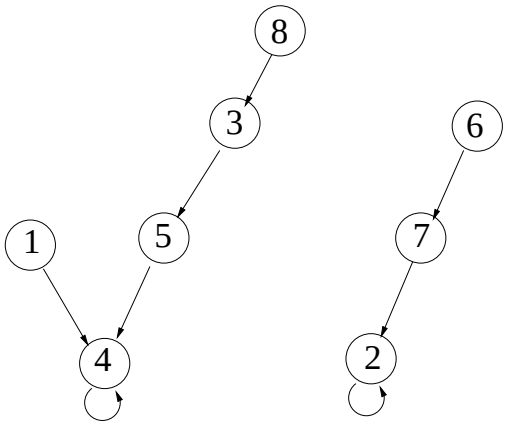
- Pas α–β–snoeien op de volgende spelboom toe om de waarde van het spel te berekenen. Evalueer altijd eerst de linkertakken. Schrijf gebruikte grenzen **aan** de toppen en de teruggegeven waarden **in** de toppen. Plaats streepjes door de bogen die naar deelgrafen leiden die je niet evalueert omdat je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



7. **Verzamelingen 1.5 pt**

- Je werkt op een 64-bit computer met het universum $\{0,\ldots,60\}$. Als je verzamelingen als bitvectoren voorstelt, is dus 1 computerwoord voldoende. Veronderstel dat M, M' verzamelingen zijn en $0 \leq i \leq 60$ Beschrijf voor elk van de volgende operaties op het einde de inhoud van de verzameling Y. Voorbeeld: `Y= M & ~M'`: Y is de verzameling van alle elementen die in M zitten en die bovendien niet bevat zijn in M'.
  - `Y= M & ~(1<<i)`:
  - `Y= M | ((1<<i)-1)`:
  - `Y= (M & ~M') | (M' & ~M)`
  - `Y= M & (M'-1)` waarbij `M'` niet leeg is:

- Pas de relatie $7 \equiv 3$ toe op de volgende union-find datastructuur. Pas union-by-size en path compression toe.



8. **Het algoritme van Miller-Rabin 1 pt**

   Pas de methode van Miller-Rabin toe om te testen of 25 een priemgetal is. Neem als *toevallig* gekozen basis het getal 7.

NOG NIET OMDRAAIEN !