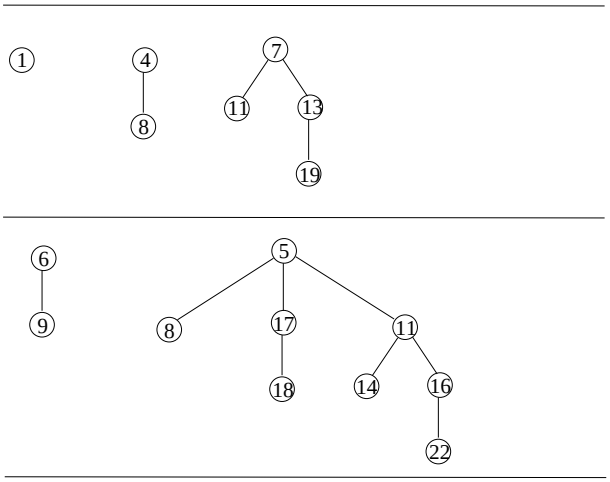
Examen Datastructuren en Algoritmen II

Naam : .....

1. **Binomiale prioriteitswachtlijnen:**

   Stel dat het bekend is hoe twee binomiale prioriteitswachtlijnen moeten gemerged worden. Beschrijf hoe elementen worden toegevoegd en hoe het kleinste element wordt verwijderd.

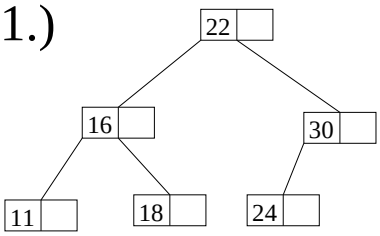
   Merge de volgende twee binomiale prioriteitswachtlijnen:



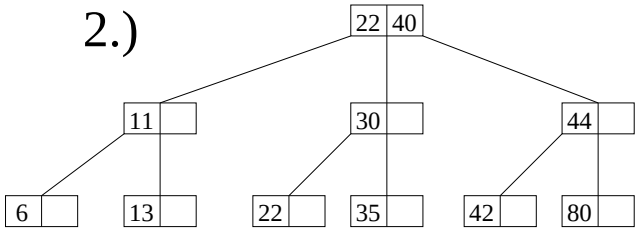
2. **2-3-bomen:**

- Wat is de definitie van een 2-3-boom?
- Welke van de volgende bomen voldoet niet aan deze definitie en waarom niet?

1.)



2.)



- Voeg de volgende reeks van getallen toe aan een initieel lege 2-3-boom: $15, 20, 30, 40, 35, 6, 10$. Verwijder ten slotte de sleutel 40.

3. **Dynamisch programmeren:**

   Beschrijf een algoritme met dynamisch programmeren voor het wisselgeldprobleem en toon aan hoe het werkt voor het voorbeeld 1,73 euro met de normale euromunten (1, 2, 5, 10, 20 en 50 cent en 1 en 2 euro).

4. **Benaderende algoritmen:**

   Deze oefening gaat over off-line algoritmen voor het inpakprobleem:

   - Geef een voorbeeld waar first-fit dalend beter presteert dan best-fit dalend.
   - Geef een voorbeeld waar best-fit dalend beter presteert dan first-fit dalend.
   - Geef een oneindige reeks van voorbeelden (dus een reeks die afhankelijk is van bij-voorbeeld k) waar best-fit dalend en first-fit dalend allebei niet het optimum vinden.

5. **Tonen dat iets arbitrair slecht presteert:**

   Een Greedy-algoritme voor het (gewone) TSP kan bijvoorbeeld zo werken: Begin met de goedkoopste boog en maak het pad altijd langer (zoals in de cursus) maar kies altijd de goedkoopste boog, die het pad verlengt.

   Toon dat dit algoritme arbitrair slechte resultaten kan opleveren. Dus: toon dat er voor elk getal k TSP's zijn, zodat dit algoritme Hamiltoniaanse cykels berekent die tenminste k keer duurder zijn, dan de goedkoopste.

6. **Een probleem vertalen:**

   Een vertegenwoordiger moet n steden bezoeken en zoekt de snelste manier om dat te doen en terug te komen. Voor elk paar a, b van steden is de tijd bekend die nodig is om van a naar b te rijden zonder door één van de andere steden te gaan. Als het belangrijk is dat hij elke stad precies één keer bezoekt (bijvoorbeeld als hij echt slechte produkten verkoopt...) dan is dit probleem equivalent met het handelsreizigersprobleem. Maar normaal is dat voor hem niet belangrijk — hij zoekt gewoon de kortste rondreis waarin elke stad tenminste één keer wordt bezocht.

   Vertaal dit probleem naar het normale TSP. Schets een polynomiaal algoritme dat als resultaat een input voor het normale TSP heeft, zodat als wij de lengte van een kortste Hamiltoniaanse cykel in dit TSP kennen, wij ook de lengte van een kortste rondreis voor de vertegenwoordiger kennen. Toon aan dat jouw algoritme aan de eisen voldoet.

7. **Spelstrategieën:**

   Een wisselgeldspel: Gegeven een bedrag b van eurocent en een getal m dat het maximale aantal zetten bepaalt. De regels zijn als volgt: In het begin ligt 0 cent op tafel. Speler X begint. Als een speler aan de beurt is en er ligt al een bedrag van $t < b$ cent op tafel, moet hij er precies één munt bijleggen zodanig dat nog steeds ten hoogste een bedrag van b op tafel ligt (1 euro telt natuurlijk als 100 cent, etc). Het spel is gedaan als er b cent op tafel ligt. Als er op dit moment ten hoogste m munten liggen, mag speler X het bedrag houden, anders speler Y. De winst is natuurlijk, wat de **andere** speler op tafel legde.

   Stel nu dat alleen maar munten van 1, 2 en 10 cent toegelaten zijn (anders wordt de vertakking gewoon te groot om het op papier te doen). Wat is de waarde van het spel met $b = 15$ en $m = 5$.

   Misschien kan je hier ook branch and bound en dynamisch programmeren toepassen om niet de hele spelboom te moeten uitwerken...

8. **Gerandomiseerde algoritmen:**

   Gegeven is een random-number-generator, die getallen $1, \ldots, k$ allemaal met kans $1/k$ genereert — maar het getal k is niet bekend.

   Beschrijf een algoritme, dat de random-number-generator gebruikt en een getal k' berekent dat met grote kans een goede bovengrens is, of precies: waarvoor met kans 99.9% geldt dat $k <= k' <= (10/9) * k$. Toon aan dat jouw algoritme aan de eisen voldoet.