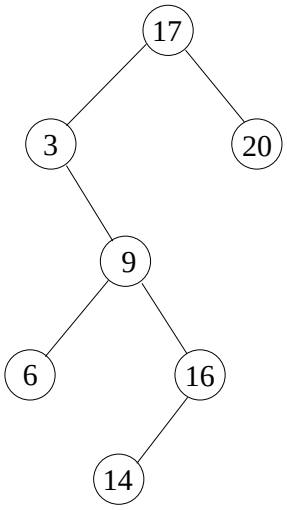
Examen Datastructuren en Algoritmen II

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 12 toe aan de volgende semisplay boom.



- Welke fractie van de toppen in een rood-zwart-boom is ten minste zwart? Geef een bewijs en toon ook aan dat er voor elke n bomen met $m \geq n$ toppen en dit aandeel zwarte toppen bestaan.

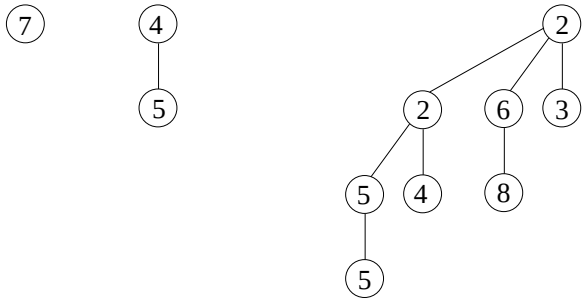
- In een binaire zoekboom T is een deelboom T' gegeven. Voor een top v in T geeft de functie `in_deelboom(v)` de waarde `true` terug als $v \in T'$ en anders `false`. Elke top heeft een pointer naar zijn kleinerdeelboom en naar zijn groterdeelboom. Het doel is nu een array `buitenbomen[]` in te vullen, waar alle pointers naar toppen die wortels van buitenbomen zijn (of NULL, als de buitenboom leeg is) in volgorde opgeslagen zijn.

  Bewijs expliciet aan de hand van de definitie van de volgorde van buitenbomen dat de volgende pseudocode die opgestart wordt met de wortel van T en de waarde van `teller` gelijk aan 0 juist werkt – of geef een voorbeeld dat aantoont dat de pseudocode niet het juiste resultaat oplevert.

```
evalueer_top(t)
{
if (in_deelboom(t))
  { if (t.kleiner=NULL) { buitenbomen[teller]=NULL; teller++; }
    else
     { if (in_deelboom(t.kleiner)) evalueer_top(t.kleiner);
          else { buitenbomen[teller]=t.kleiner; teller++; }
      }
    if (t.groter=NULL) { buitenbomen[teller]=NULL; teller++; }
    else
     { if (in_deelboom(t.groter)) evalueer_top(t.groter)
          else { buitenbomen[teller]=t.groter; teller++; }
      }
   }
else
  { if (t.kleiner!=NULL) evalueer_top(t.kleiner);
    if (t.groter!=NULL) evalueer_top(t.groter);
  }
}
```

2. **Heaps 2 pt**

- Verwijder de kleinste sleutel uit de volgende binomiale prioriteitswachtlijn:



- Je werkt met een skew heap en gebruikt de niet-recursieve manier voor het mergen van heaps. Je hebt een dalende reeks van sleutels $d_1 > d_2 > \cdots > d_n$ en voegt die in deze volgorde toe aan een initieel lege heap. Wat is de kost van de hele reeks van toevoegbewerkingen? Bewijs dat jouw antwoord juist is – en dat betekent dat je niet alleen maar een voorbeeld geeft.

- Je hebt twee reeksen van sleutels $R_1 = a_1, a_2, \ldots, a_n$ en $R_2 = b_1, b_2, \ldots, b_n$. Toon aan dat de tijden voor het toevoegen van de hele reeks R_1 aan een initieel lege binomiale prioriteitswachtlijn en voor het toevoegen van de hele reeks R_2 aan een initieel lege binomiale prioriteitswachtlijn (misschien op een additieve of multiplikatieve constante na) dezelfden zijn.

3. **Geamortiseerde complexiteit 2.75 pt**

Je werkt met een array `list[]` van gelinkte lijsten. Wij noemen deze datastructuur hier lili-array. De gelinkte lijsten bevatten allemaal gehele getallen als sleutels en de volgorde is stijgend, dus bv. $2 \to 7 \to 7 \to 12$.

Als je een nieuwe sleutel wilt toevoegen, maak je een gelinkte lijst met deze sleutel – dus $2^0 = 1$ sleutels – aan en probeert die te plaatsen in de lili-array.

Plaatsen van een gesorteerde gelinkte lijst met 2^i sleutels in een lili-array werkt als volgt:

Als `list[i]` leeg is, wordt de lijst in `list[i]` geplaatst. Anders merge je deze lijst en die in `list[i]` op de gewone manier tot een gesorteerde gelinkte lijst met 2^{i+1} elementen en probeer je die te plaatsen in de lili-array.

Als je met een lege lili-array begint en er staat een lijst in `list[i]`, dan heeft die dus lengte 2^i.

De bewerkingen op deze datastructuur zijn:

(i) toevoegen van een sleutel aan de datastructuur

(ii) uitvoeren van alle sleutels in stijgende volgorde – de lijst is achteraf leeg

---

- Geef de pseudocode van een algoritme dat alle n sleutels van een lili-array in tijd $o(n \log n)$ (klein o) in stijgende volgorde kan uitvoeren. Als je in jouw algoritme gekende algoritmen of datastructuren gebruikt, hoeft de pseudocode voor de werking binnen deze datastructuren niet gegeven te worden.

- Wat is de maximale kost ($O()$-notatie) van een bewerking in een rij van n toevoegbewerkingen op een initieel lege lili-array? Geef uitleg.

- Wat is de geamortiseerde kost ($O()$-notatie) van een bewerking in een rij van n toevoegbewerkingen op een initieel lege lili-array? Je mag zelf kiezen op welke manier je de kost bepaalt – maar de aggregaatmethode is zeker geen slechte keuze als je nog weet hoe die voor de binaire teller werd toegepast. Tip: als je dit gebruikt om te sorteren is de complexiteit beter dan die van bubblesoort.

4. **Dynamisch programmeren 2 pt**

Gegeven een graaf $G = (V,E)$ waar V uit twee disjuncte verzamelingen $U = \{u_1, \ldots, u_n\}$ en $W = \{w_1, \ldots, w_n\}$ bestaat en de bogen allemaal een eindpunt in U en een eindpunt in W hebben. Wij zoeken nu een zo groot mogelijke verzameling M (de letter M staat voor *matching*) van bogen waarvan er geen twee een gemeenschappelijke top hebben, zodat voor twee bogen (u_i, w_j) en (u_k, w_l) met $k > i$ ook geldt dat $l > j$. Wij noemen dat hier een monotone matching. In de volgende afbeelding is de eerste matching (de dikke lijnen) monotoon en de tweede niet.

Geef een dynamisch programmeren algoritme dat in tijd $O(n^2)$ de grootte van een grootste monotone matching in een dergelijke graaf bepaalt. Leg expliciet uit waarom het algoritme tijd $O(n^2)$ vraagt. Veronderstel je iets over de manier waarop de inputgraaf gecodeerd is (als bogenmatrix, als lijst van lijsten, als…)?

5. **Spelbomen 1.25 pt**

   - Twee spelers X en O schrijven afwisselend ofwel 1 ofwel 2 op hun eigen blad papier. Nadat iedereen 2 getallen heeft opgeschreven wordt de som s berekend. Als de som even is wint X s Euro en anders wint O s Euro. Teken de spelboom voor dit spel en bereken de waarde van het spel.

- Ook al kan je de spelboom tekenen en de waarde berekenen, zegt de waarde in het geval dat de spelers pas op het einde kunnen zien wat de andere speler heeft opgeschreven niets over goede en slechte strategieën. Wat is het probleem? Wat hebben wij altijd impliciet verondersteld als wij in de les zeiden dat een speler *een tak in de spelboom kiest*?

6. **Online algoritmen 1.5 pt**

- Wij willen een online algoritme voor het inpakprobleem ontwikkelen dat iets zwakkere eisen heeft dan de online algoritmen uit de les: je hoeft niet elk pakket onmiddelijk te plaatsen, maar je wacht altijd totdat je drie pakketten hebt en (behalve als er op het einde geen drie pakketten meer in de rij zijn) pas dan beslis je hoe deze pakketten het best geplaatst worden. Doordat je meer informatie hebt op het moment dat je de pakketten moet plaatsen, kan je de pakketten dan soms beter plaatsen. Wij noemen een dergelijk algoritme hier een *zwak online inpakalgoritme*.

  Bewijs: Voor elk zwak online inpakalgoritme A bestaat er een rij van gewichten die minimaal m vrachtwagens vraagt en waarvoor A ten minste $\frac{3}{2}m$ vrachtwagens vraagt.

- Geef een voorbeeld waar de best fit online heuristiek beter presteert dan de first fit dalend offline heuristiek en een voorbeeld waar het omgekeerd is.

7. **Gretige algoritmen 1.5 pt**

   Gegeven is een Nederlandse tekst, die wel blanks bevat, maar geen linefeeds. Die moet nu geformateerd worden, zodat elke lijn ten hoogste m tekens bevat, waarbij het alleen toegelaten is blanks door linefeeds te vervangen – scheidingstekens mogen dus niet gebruikt worden. Het linefeed-teken wordt bij de m tekens meegerekend. Het mag verondersteld worden dat er geen deelstrings zonder blanks zijn die langer dan $m - 1$ zijn. Het doel is op het einde zo weinig mogelijk lijnen te hebben.

   - Geef een gretig algoritme voor dit probleem dat een optimale oplossing vindt. Leg uit waarom het een gretig algoritme is.

- Bewijs dat jouw algoritme een optimale oplossing vindt.

8. **Verzamelingen 1.5 pt**

- Gebruik voor het volgende union-find, waarbij je union by size and path compression toepast. In gevallen waar door union by size niet vastgelegd is welke top naar welke moet wijzen, laat altijd de grotere naar de kleinere wijzen.

  Het universum is $\{0, 1, \ldots, 8\}$. Bereken de equivalentieklassen van de door $0 \equiv 1$, $2 \equiv 4$, $8 \equiv 5$, $8 \equiv 4$, $7 \equiv 5$, $0 \equiv 5$ en $8 \equiv 2$ gegenereerde equivalentierelatie. Pas de equivalenties in de gegeven volgorde toe.

- Gegeven een graaf $G = (V, E)$. Als je elke boog $\{v, w\}$ als relatie interpreteert die zegt $v \equiv w$, dan wordt door E een equivalentierelatie op V gedefinieerd.
  - Wat zijn de equivalentieklassen?

  - Welke methode gebruik je om de equivalentieklassen te berekenen – motiveer jouw keuze.

9. **Gerandomiseerde algoritmen 1 pt**

   Pas de Miller-Rabin priemgetallentest toe om te testen of 49 een priemgetal is. Neem als toevallig gekozen getal tussen 0 en 49 het getal 31.

NOG NIET OMDRAAIEN !