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!

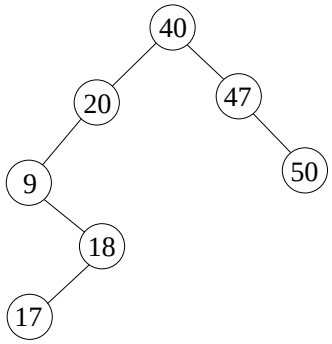
Schrijf leesbaar. Oplossingen die niet leesbaar zijn, kunnen ook niet beoordeeld worden.

Als technieken toegepast moeten worden, toon altijd voldoende tussenstappen om te kunnen zien wat er gebeurt en dat de technieken goed verstaan zijn.

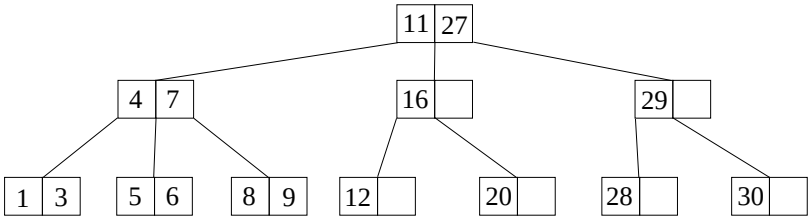
Stellingen uit de les mogen natuurlijk altijd gebruikt worden zonder dat het bewijs opnieuw gegeven moet worden (behalve in gevallen waar het expliciet anders staat)!

1. **Zoekbomen 4 pt**

   - Pas de volgende bewerkingen in de gegeven volgorde toe op de semi-splay boom in de afbeelding: 10 toevoegen, 17 verwijderen, 53 opzoeken.



- Voeg sleutel 2 toe aan de volgende 2-3-boom:



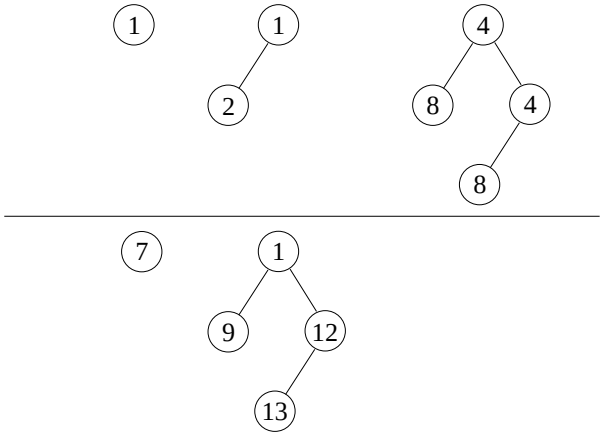
- Stel dat het opzoeken van een sleutel in een zoekboom gebeurt door langs een pad van de wortel naar de top met deze sleutel te lopen. Toon aan dat er voor elk gegeven algoritme (met of zonder herbalanceren) om toppen aan een binaire zoekboom toe te voegen een positieve constante c en een reeks van toevoegbewerkingen bestaat zodat je als je met een lege boom begint en deze n toevoegbewerkingen doet, ten minste $c * n * \log n$ toppen bezoekt.

- Bewijs de volgende uitspraak of geef een tegenvoorbeeld: In een rood-zwart boom met n toppen – z zwarte toppen en r rode toppen – geldt altijd $r \leq \frac{2}{3}n$.

2. **Wachtlijnen 3 pt**

    - Voeg de sleutels 11, 19, 15, 30, 17, 5 en 7 in deze volgorde toe aan een initieel lege skew heap.

- Merge de volgende twee binomiale prioriteitswachtlijnen. Verwijder achteraf het kleinste element. Als er in twee binomiale bomen hetzelfde kleinste element in de wortel zit, verwijder dan het element uit de grotere boom (in de praktijk zou dat misschien niet de beste oplossing zijn).



- Een winkelketen met 8 winkels in België heeft het volgende probleem: elke winkel houdt een prioriteitswachtlijn bij waarin zodra opgemerkt wordt dat een product besteld moet worden deze bestelling geplaatst wordt. Aan elke bestelling wordt een bepaald gewicht toegekend dat de *dringendheid* van de bestelling beschrijft. Dit gewicht bepaalt de positie in de wachtlijn. Op zekere momenten (bijvoorbeeld 's avonds om 8 uur) worden de wachtlijnen dan naar de centrale gestuurd, gemerged en de bestellingen worden vanuit de centrale in volgorde van dringendheid opgestuurd. Omdat alle computers die hiermee bezig zijn ook nog andere dingen moeten doen, is het de bedoeling dat dit alles zo efficiënt mogelijk gebeurt (in de winkels en in de centrale).

  Welke van de wachtlijnen die je hebt gezien zou je voor dit probleem voorstellen? Verantwoord ook waarom deze wachtlijnen het meest geschikt zijn voor het voorbeeldprobleem.

3. **Dynamisch programmeren 3 pt**

Je staat in een stad aan het station en moet naar het gemeentehuis gaan. Je wil altijd alleen straten nemen die je dichter bij jouw doel leiden. Langs de straten zijn verschillende bezienswaardigheden waaraan je een zekere score toekent. De straten hebben dus een zekere waarde. Je wilt nu een weg vinden die je van het station naar het gemeentehuis brengt en waar je de meeste bezienswaardigheden kan zien – of anders: waar de som van de scores van de straten het hoogst is.

Dit probleem kan je modelleren als een gerichte graaf $D = (V,E)$ met een startpunt s, een eindpunt e en zonder gerichte cykels. De top s is de enige top waar alleen maar bogen vertrekken en e is de enige top waar alleen maar bogen naartoe gaan. De bogen hebben gewichten en je zoekt een gericht pad van s naar e zodat de som van de gewichten maximaal is. Gezocht is dus het pad – niet alleen het maximale gewicht.

Deel 1 (preprocessing, 1 pt)

Geef een algoritme dat aan elke top in V een waarde $l(v)$ toekent die de lengte (de gewone lengte – zonder gewichten) van het langste gerichte pad naar v bevat. Dit algoritme hoeft niet met dynamisch programmeren te zijn, maar moet wel polynomiaal zijn. Om volle punten te krijgen moet het algoritme lineair in $|V| + |E|$ te zijn (en dat moet ook bewezen worden).

Deel 2 (het pad berekenen, 2 pt)

Geef een algoritme met dynamisch programmeren dat het pad met de grootste som van gewichten berekent. Daarbij mag $l(v)$ gebruikt worden – om het even of het eerste gedeelte opgelost is of niet – maar het hoeft niet gebruikt te worden als je een andere oplossing beter vindt.

4. **Verzamelingen 1 pt**

   Ons universum is de verzameling $\{0,\ldots,63\}$ en wij werken met een computer waar de lengte van een woord 64 bit is. Je mag (zoals in C) veronderstellen dat alle 64 bit voor de voorstelling van de verzamelingen bruikbaar zijn. M en M' zijn twee verzamelingen die als bitvectoren voorgesteld zijn (het zijn dus bv *unsigned long ints*). Beschrijf (zoals in de les) zo efficiënt mogelijke uitdrukkingen die het volgende betekenen:

   - $i \in M$ en $i \in M'$
   - i is een element van precies één van de verzamelingen M, M'
   - er bestaat een element dat in beide verzamelingen M, M' zit
   - niet elk element $0 \leq j < 64$ komt in M of M' voor.

5. **Geamortiseerde complexiteitsanalyse 3 pt**

   - In de les hebben wij gezien dat een reeks van n bewerkingen op een initieel lege semi-splay boom een kost van $O(n \log n)$ heeft (dus geamortiseerd $O(\log n)$ per bewerking). Bepaal nu de kost van een reeks van n bewerkingen op een semi-splay boom als de boom in het begin niet leeg is, maar al n bewerkingen gebeurd zijn. Voor deze n bewerkingen was de boom wel leeg.

- Wij werken hier met een verzameling van arrays. Voor bewerkingen die een constante kost hebben, wordt gewoon 1 als kost gerekend. De bewerkingen die je kan doen, zijn:

  Toevoegen: Een element wordt aan de datastructuur toegevoegd. Als er nog ruimte is in één van de arrays wordt het daar toegevoegd – anders wordt een lege array van lengte 2 aangemaakt en het aan deze nieuwe array toegevoegd. De kost is in elk geval constant, dus 1.

  Verwijderen: Eén element wordt uit een arbitraire array verwijderd. De kost van deze bewerking is constant, dus 1.

  Opkuisen: Als er m volle arrays met $l_1, \ldots, l_m$ elementen zijn, worden ze samengevoegd tot één nieuwe array met $l_1 + \cdots + l_m$ elementen en lengte $2(l_1 + \cdots + l_m)$. De kost van deze bewerking is $l_1 + \cdots + l_m$.

  Bepaal de maximale en de geamortiseerde kost van een bewerking in een reeks van n willekeurige bewerkingen op een initieel lege verzameling. De $O()$-notatie is voldoende. Gebruik de methode die je het meest geschikt lijkt.

6. **Een algoritme kiezen 1 pt**

Om een examenrooster zo vast te leggen dat er geen botsingen zijn, wordt een computer gebruikt. Een botsing is als twee examens waaraan een student moet deelnemen op dezelfde dag plaatsvinden. De computer gebruikt een algoritme dat voor een vaste volgorde van examens $e_1, \ldots, e_n$ alle permutaties van mogelijke termijnen $t_1, \ldots, t_m$ waar leszalen beschikbaar zijn opsomt. Daarbij kan een t_i meerdere keren opduiken als op die termijn meerdere leszalen beschikbaar zijn. Wij veronderstellen dat $m \geq n$ – anders kunnen de examens natuurlijk niet allemaal plaatsvinden. Voor elke permutatie test het algoritme of er een botsing is als (voor $1 \leq i \leq n$) examen e_i op het moment t_i plaatsvindt. Het algoritme stopt zodra er een oplossing zonder botsingen is gevonden. Maar hoewel aangetoond kan worden dat als je naar alle combinaties kijkt één op 50 combinaties een oplossing zonder botsingen is, duurt het soms heel lang tot er één gevonden wordt. Blijkbaar is de verdeling van geldige oplossingen heel slecht....

Geef de programmaontwikkelaars advies: geef een alternatief en beter algoritme, analyseer dat en geef overtuigende argumenten waarom het de betere keuze is.

7. **Gerandomiseerde algoritmen 1 pt**

   In de les hebben wij gezien dat het Miller-Rabin algoritme alle *speciale getuigen* herkent. Dit wordt gebruikt om de prestatie van het algoritme in te schatten. Maar het algoritme kan soms ook zeggen dat het geteste getal niet priem is zonder dat de beslissing op de definitie van de speciale getuigen gebaseerd is. Beschrijf precies welke gevallen dat zijn.

NOG NIET OMDRAAIEN !