Tentamen Complexiteit
Donderdag 8 juni 2017, 10.00 – 13.00 uur

Geef een *duidelijke* toelichting bij al je antwoorden! Veel succes!

Opgave 1. (23 punten)
Gegeven een array $A = A[1], A[2], \ldots, A[n]$ dat $n \geq 2$ verschillende getallen bevat (n is even). Het array heeft de volgende eigenschap: voor alle oneven posities i geldt dat $A[i] < A[i+1]$. Voorbeeld met $n = 10$: $14, 17, 8, 12, 6, 11, 4, 9, 10, 15$. Gevraagd worden de grootste en de op-een-na-grootste waarde uit A. In het voorbeeld zijn dit $A[2] = 17$, respectievelijk $A[10] = 15$.

a. (4 punten)
Gegeven een beslissingsboom corresponderend met een algoritme dat gebaseerd is op arrayvergelijkingen. Wat stelt –in termen van het algoritme– een pad van de wortel naar een blad voor, wat stelt de hoogte van zo'n boom voor en wat kun je zeggen over de bladeren?
b. (8 punten)
Bewijs met behulp van een *beslissingsboomargument* dat elk algoritme voor bovenstaand probleem dat is gebaseerd op arrayvergelijkingen, voor de bekeken soort invoerrijtjes ten minste $\lceil 2 \lg n \rceil - 2$ vergelijkingen moet doen in de worst case. Formuleer je bewijs zorgvuldig en geef aan welke stellingen/eigenschappen je gebruikt.
c. (5 punten)
Geef in woorden of in C++ een algoritme dat het probleem oplost, dus dat de index van de grootste waarde en de index van de op-een-na-grootste waarde uit A oplevert. Je algoritme moet gebaseerd zijn op arrayvergelijkingen en $n - 2$ van zulke vergelijkingen doen. Geef ook duidelijk aan waarom je algoritme $n - 2$ vergelijkingen doet.
d. (6 punten)
Laat $\mathcal{B}$ een ander algoritme zijn (eveneens gebaseerd op het doen van arrayvergelijkingen) dat het probleem oplost. Welke van onderstaande beweringen over $\mathcal{B}$ en het algoritme uit **c** is/zijn in tegenspraak met het bewezene in **b** en welke kan/kunnen wel waar zijn? Licht je antwoorden toe!

(i) Het aantal arrayvergelijkingen dat $\mathcal{B}$ doet in de worst case is $\Theta(\sqrt{n})$.
(ii) Het aantal arrayvergelijkingen dat $\mathcal{B}$ doet in de worst case is $O(\sqrt{\lg n})$.
(iii) Het algoritme uit **c** is optimaal.

Opgave 2. (5+10=15 punten)
We bekijken *Shellsort* met sprongafstanden (= stapgroottes) $\frac{n}{2}, \frac{n}{2^2}, \frac{n}{2^3}, \ldots, 2, 1$.
(i) Leg in woorden uit wat Shellsort doet en waarom het correct sorteert.
(ii) Bewijs dat het aantal arrayvergelijkingen dat Shellsort met bovengenoemde sprongafstanden doet in de worst case $O(n^2)$ is. Neem hierbij aan dat $n = 2^k$ een 2-macht is.
Hint bij het bewijs: laat zien dat het $\frac{n}{2^i}$-sorteren $O(n \cdot 2^i)$ arrayvergelijkingen kost in de worst case ($i = 1, 2, \ldots, k$). Je mag verder gebruiken dat $\sum_{i=0}^{k} 2^i = 2^{k+1} - 1$.

Opgave 3. (14 punten)
Gegeven is een array $A = A[1], A[2], \ldots, A[n]$, dat verschillende getallen bevat en waarbij $n \geq 0$ even is, dus $n = 2k$ voor zekere gehele $k \geq 0$. We gaan het array *recursief* sorteren als volgt. Voor $n \geq 2$ brengen we het array eerst in de vorm zoals in **opgave 1** vereist was, gebruiken vervolgens **1c** om de grootste en de een-na-grootste waarde te bepalen en zetten die (door te verwisselen) op hun definitieve plaats. Het basisgeval is $n = 0$. Het aantal arrayvergelijkingen dat dit algoritme doet noemen we $Z(n)$.

a. (3 punten)
Leg uit waarom $Z(n)$ voldoet aan de volgende recurrente betrekking.

$$Z(n) = \begin{cases} 0 & n = 0 \\ Z(n-2) + \frac{3}{2}n - 2 & n \geq 2, n = 2k \end{cases}$$

b. (11 punten)
Los de recurrente betrekking exact op door deze herhaald in zichzelf in te vullen (substitutiemethode). Schrijf $Z(n)$ als functie van n. Bewijs vervolgens met behulp van volledige inductie dat de aldus gevonden oplossing inderdaad voldoet.
Je mag gebruiken: $\sum_{i=1}^{m} i = \frac{1}{2}m(m+1)$

Opgave 4. (20 punten)
Gegeven is een array $A = A[1], A[2], \ldots, A[n]$ dat $n > 3$ gehele getallen bevat, waarbij n *even* is. We bekijken een algoritme dat de index van de *major* oplevert indien die bestaat, en anders -1. Een waarde uit het array A heet een *major* als deze ten minste $n/2 + 1$ keer voorkomt. Het algoritme is als volgt.

```
(1)   i := 1; major := −1;
(2)   found := False;
(3)   while not found and i < n do
(4)       if A[i] = A[i + 1] then
(5)           j := 1; tel := 0;
(6)           while j ≤ n do
(7)               if A[j] = A[i] then
(8)                   tel := tel + 1;
                  fi
(9)               j := j + 1;
              od
(10)          if tel > n/2 then
(11)              major := i; found := True;
              fi
          fi
(12)      i := i + 2;
      od
```

Toelichting: eenvoudig is in te zien dat als de major M bestaat, er een paar $(A[i], A[i+1])$ moet zijn met i oneven, waarvoor geldt dat $A[i] = A[i+1] = M$. Het algoritme maakt hier-

van gebruik door alle tweetallen $A[i], A[i+1]$ met i oneven af te lopen. Indien $A[i] = A[i+1]$ wordt geteld of de betreffende waarde meer dan $n/2$ keer voorkomt in A en dus een major is (of niet).

a. (4 punten)
Leg uit waarom het vergelijken van array-elementen in regel (7) alleen niet maatgevend is voor de complexiteit van het algoritme.

Het *vergelijken* van array-elementen in regel (4) en (7) *samen* is wel maatgevend voor de complexiteit. In **b** en **c** worden dan ook telkens deze vergelijkingen samen geteld.

b. (7 punten)
Hoeveel vergelijkingen van array-elementen (regel (4) + regel (7)) worden er gedaan in het *slechtste* geval, voor algemene n? Voor wat voor invoerrijtjes komt dat voor? Geef *alle* gevallen en leg op basis van het algoritme uit hoe je aan je antwoord komt. Geef ook een concreet, illustratief voorbeeld van een worst case invoerrijtje met $n = 10$.
c. (9 punten)
Hoeveel vergelijkingen van array-elementen (regel (4) + regel (7)) worden er gedaan in het *beste* geval, voor algemene n? Voor wat voor invoerrijtjes komt dat voor? Geef *alle* gevallen en leg op basis van het algoritme uit hoe je aan je antwoord komt.
Bekijk eerst het best case apart voor (i) het geval dat de major niet bestaat, en (ii) het geval dat de major wel bestaat.

Opgave 5. (28 punten)
We bekijken de volgende twee beslissingsproblemen:

3Kleur: Gegeven een ongerichte graaf $\mathcal{G} = (V, E)$. Kunnen de knopen van $\mathcal{G}$ zo gekleurd worden met hooguit drie kleuren dat geen tweetal aangrenzende knopen dezelfde kleur heeft? Zo'n kleuring zullen we een 3kleuring noemen.

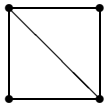
Iso-3Kleur: Gegeven een ongerichte graaf $\mathcal{G} = (V, E)$, waarbij het aantal knopen een drievoud is. Kunnen de knopen van $\mathcal{G}$ zo gekleurd worden met hooguit drie kleuren dat geen tweetal aangrenzende knopen dezelfde kleur heeft en dat elke kleur precies even vaak voorkomt? Zo'n kleuring zullen we een iso-3kleuring noemen.

a. (10 punten)
Toon aan dat Iso-3Kleur $\in \mathcal{N}P$ door een *niet-deterministisch polynomiaal* algoritme voor Iso-3Kleur te geven. Het algoritme heeft dus als invoer een ongerichte graaf $\mathcal{G} = (V, E)$, waarbij het aantal knopen een drievoud is. Het algoritme moet "ja" opleveren dan en slechts dan als $\mathcal{G}$ een ja-instantie is voor Iso-3Kleur (&). Je mag er van uitgaan dat het aantal knopen van $\mathcal{G}$ bekend is en dat $V = \{1, 2, \ldots, m\}$, met m een drievoud.
Leg onder andere uit wat in elke fase van het algoritme gebeurt, en in het bijzonder wat in fase 2 wordt gecontroleerd en hoe, en hoeveel stappen dat kost. Leg ook uit waarom jouw niet-deterministische algoritme aan (&) voldoet en waarom het polynomiaal is in $|\mathcal{G}|$.

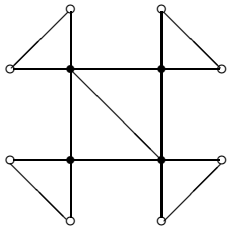
We bekijken een transformatie T die instanties van 3Kleur transformeert naar instanties van Iso-3Kleur. Gegeven een ongerichte graaf $\mathcal{G} = (V, E)$, met $V = \{1, 2, \ldots, n\}$. We

construeren uit $\mathcal{G}$ een ongerichte graaf $\mathcal{G}' = (V', E') = T(\mathcal{G})$ met een drievoud aan knopen. *Elke* knoop $v \in V$ beelden we af op een drietal knopen $\{v, v_1, v_2\}$. We verbinden v_1 en v_2 met elkaar en met v en brengen een tak aan tussen knopen u en v dan en slechts dan als er in $\mathcal{G}$ een tak tussen u en v zit. Samengevat: $V' = \{v, v_1, v_2 \mid v \in V\}$ en $E' = \{(u, v) \mid (u, v) \in E\} \cup \{(v, v_1), (v, v_2), (v_1, v_2) \mid v \in V\}$.
Het is duidelijk dat de constructie van $T(\mathcal{G})$ uit $\mathcal{G}$ polynomiaal is. Samen met **b** volgt dan dat 3Kleur $\leq_{\mathrm{P}}$ Iso-3Kleur.

Voorbeeld:



wordt afgebeeld op



b. (6 punten)
Toon aan dat geldt: $\mathcal{G}$ is een ja-instantie van 3Kleur $\iff$ $T(\mathcal{G})$ is een ja-instantie van Iso-3Kleur. (Ofwel: $\mathcal{G}$ heeft een 3kleuring $\iff$ $T(\mathcal{G})$ heeft een iso-3kleuring).
c. (2 punten)
Wanneer is een beslissingsprobleem P NP-hard? En wanneer NP-volledig? (De definitie van $\mathcal{NP}$ hoeft niet te worden gegeven.)

Van college is bekend dat 3Kleur $\in \mathcal{NPC}$ en dat 2Kleur $\in \mathcal{P}$. Verder mag je in **d** gebruiken dat Iso-3Kleur $\in \mathcal{N}P$ en dat 3Kleur $\leq_{\mathrm{P}}$ Iso-3Kleur (#).

d. (10 punten)
Beantwoord de volgende vragen en geef bij elk antwoord een duidelijke uitleg.
(i) Volgt uit de gegevens hierboven dat 2Kleur $\leq_{\mathrm{P}}$ Iso-3Kleur?
(ii) Waarom kunnen we uit het feit dat 3Kleur $\in \mathcal{NPC}$ concluderen dat ook Iso-3Kleur $\leq_{\mathrm{P}}$ 3Kleur (##)?
(iii) Er geldt dat Iso-3Kleur $\in \mathcal{NPC}$. Volgt dit uit **a** en (#) of uit **a** en (##)?
(iv) Je neef heeft een polynomiale reductie bedacht van Iso-3Kleur naar 2Kleur. Denk je dat zijn bewijs klopt?

EINDE