Tentamen Complexiteit 5 juni 2023
Uitwerking

Opgave 1. (24 punten)

a. (4 punten) Een pad van de wortel naar een blad stelt de serie achtereenvolgende arrayvergelijkingen voor die het algoritme doet op zekere invoer. De lengte van zo'n pad correspondeert precies met het aantal arrayvergelijkingen dat op die invoer wordt gedaan. In het bijbehorende blad is het antwoord van het algoritme voor die invoer bekend: het algoritme stopt. Bladeren kunnen derhalve geassocieerd worden met de eindantwoorden/uitkomsten die het algoritme vindt. De hoogte van de beslissingsboom (lengte van het langste pad) stelt dan ook het aantal arrayvergelijkingen in de worst case voor.

b. (4 punten) Vanwege de speciale vorm van de invoerrijtjes, moet de grootste waarde altijd in de linkerhelft van het array staan. Laat $A[j]$ het grootste element zijn. Dan weten we dus dat $1 \leq j \leq \frac{n}{2}$. Alle elementen in de rechterhelft hebben immers een groter element op de corresponderende plek in de linkerhelft.

Het op-één-na-grootste element kan dan een van de resterende $\frac{n}{2} - 1$ elementen in de linkerhelft zijn, of het element in de rechterhelft op positie $j + \frac{n}{2}$. Van alle andere elementen in de rechterhelft weten we immers dat ze kleiner zijn dan (minstens) twéé andere elementen: het bijbehorende element in de linkerhelft, en (transitief) dat op positie j.

c. (6 punten) Eerst twee belangrijke opmerkingen/stellingen over beslissingsbomen:

- Elk mogelijk antwoord moet als blad kunnen voorkomen omdat het algoritme voor elke mogelijke invoer van het probleem moet werken. Hieruit volgt onmiddellijk dat elke beslissingsboom (corresponderend met een algoritme voor het betreffende probleem) *minstens* zoveel bladeren heeft als dat er antwoorden mogelijk zijn, dus $b = \#\text{bladeren} \geq$ aantal mogelijke antwoorden.
- Verder hebben we een stelling voor binaire bomen die het verband aangeeft tussen de hoogte h en het aantal bladeren b, namelijk $h \geq \lceil \lg b \rceil$.

We moeten dus voor ons probleem het aantal mogelijke antwoorden weten, in dit geval de indices van de grootste en de op-één-na-grootste waarden uit het array, voor arrays van de beschreven vorm. In **b** zagen we dat er $\frac{n}{2}$ mogelijkheden zijn voor de index van het grootste element, en vervolgens nog eens $\frac{n}{2} - 1 + 1 = \frac{n}{2}$ mogelijkheden voor de index van het bijbehorende op-één-na-grootste element. Dit betekent dat $b \geq \frac{n^2}{4}$. Een en ander achter elkaar geschakeld: aantal arrayvergelijkingen in de worst case voor elk algoritme voor het onderhavige probleem $= h \geq \lceil \lg b \rceil \geq \lceil \lg \frac{n^2}{4} \rceil = \lceil \lg n^2 - \lg 4 \rceil = \lceil 2 \lg n - 2 \rceil = \lceil 2 \lg n \rceil - 2 = 2 \lg n - 2$; de laatste gelijkheid gaat op omdat we weten dat $n = 2^k$ voor $k \geq 1$.

d. (4 punten) Door de toernooimethode toe te passen op de linkerhelft van A, kunnen we daarvan het grootste en het op-één-na-grootste element vinden in $\frac{n}{2} + \lg \frac{n}{2} - 2 = \frac{n}{2} + \lg n - \lg 2 - 2 = \frac{n}{2} + \lg n - 3$ vergelijkingen. Het gevonden grootste element, zeg $A[j]$, uit de linkerhelft is ook meteen het grootste element uit het hele array. Het op-één-na-grootste element uit het hele array is dan ofwel het op-één-na-grootste uit de linkerhelft, of $A[j + \frac{n}{2}]$. Hiervoor is nog één vergelijking nodig, wat het geheel brengt op $\frac{n}{2} + \lg n - 2$.

e. (3 punten) Er is een verschil tussen het aantal vergelijkingen dat het algoritme uit **d** doet en de aangetoonde ondergrens uit **c**. Hieruit kunnen we echter nog niets concluderen: het zou kunnen zijn dat het algoritme niet optimaal is, of dat de ondergrens niet scherp is, of zelfs allebei.

f. (3 punten) De toernooimethode is optimaal bewezen voor willekeurige rijtjes. Het algoritme uit **d** doet weliswaar minder vergelijkingen maar maakt gebruik van voorkennis over de invoerrijtjes. Het werkt niet voor alle gevallen en is dus niet in tegenspraak met de optimaliteit van de toernooimethode in het algemene geval.

Opgave 2. (15 punten)

a. (5 punten) In het basisgeval $n = 2$ hebben we precies één witte en één zwarte knoop. Zonder vergelijkingen kunnen we concluderen dat er dan géén takken lopen tussen paren witte of tussen paren zwarte knopen; deze paren zijn er immers niet.

In het recursieve geval voeren we twee keer hetzelfde algoritme uit op een graaf met dezelfde eigenschap maar de helft van het aantal knopen, wat de recurrente term $2T(\frac{n}{2})$ geeft. Vervolgens moeten we nog takken tellen tussen de twee helften. Kandidaten hiervoor zijn paren knopen (u, v) waarbij u en v uit verschillende helften komen en dezelfde kleur hebben. Voor elk van de $\frac{n}{2}$ knopen in de linkerhelft zijn er $\frac{n}{4}$ knopen in de rechterhelft met dezelfde kleur. Dit geeft de resterende term $\frac{n}{2} \times \frac{n}{4} = \frac{n^2}{8}$.

b. (10 punten) We vullen eerst herhaald de recurrente betrekking in in zichzelf:

$$\begin{aligned}
T(n) &= 2T(\frac{n}{2}) + \frac{n^2}{8} && \text{(definitie)}\\
&= 2(2T(\frac{n}{4}) + \frac{n^2}{32}) + \frac{n^2}{8} && \text{(definitie)}\\
&= 4T(\frac{n}{4}) + \frac{n^2}{16} + \frac{n^2}{8} && \text{(herschrijven)}\\
&= 4(2T(\frac{n}{8}) + \frac{n^2}{128}) + \frac{n^2}{16} + \frac{n^2}{8} && \text{(definitie)}\\
&= 8T(\frac{n}{8}) + \frac{n^2}{32} + \frac{n^2}{16} + \frac{n^2}{8} && \text{(herschrijven)}\\
&= 8T(\frac{n}{8}) + \frac{n^2}{8} \times (1 + \frac{1}{2} + \frac{1}{4}) && \text{(herschrijven)}\\
&= 2^3T(\frac{n}{2^3}) + \frac{n^2}{8} \times (\frac{1}{2^0} + \frac{1}{2^1} + \frac{1}{2^2}) && \text{(herschrijven)}
\end{aligned}$$

Vermoedelijke algemene vorm (klopt met de eerste drie gevallen) is dan:

$$T(n) \stackrel{?}{=} 2^\ell T(\frac{n}{2^\ell}) + \frac{n^2}{8} \times \sum_{i=0}^{\ell-1} (\frac{1}{2})^i$$

Om van de recurrente term af te komen redeneren we terug naar het basisgeval: $\frac{n}{2^\ell} = 2 \Leftrightarrow 2^\ell = \frac{n}{2} \Leftrightarrow \ell = \lg \frac{n}{2} = \lg n - 1 = k - 1$. Dit invullen geeft:

$$
\begin{aligned}
T(n) &\stackrel{?}{=} \frac{n}{2} \times T(2) + \frac{n^2}{8} \times \sum_{i=0}^{\lg n-2} (\frac{1}{2})^i \\
&= \frac{n}{2} \times 0 + \frac{n^2}{8} \times (2 - \frac{1}{2^{\lg n-2}}) && (n \geq 4 \text{ dus } \lg n - 2 \geq 1) \\
&= \frac{n^2}{8} \times (2 - \frac{1}{\frac{n}{4}}) = \frac{n^2}{8} \times (2 - \frac{4}{n}) \\
&= \frac{n^2}{4} - \frac{n}{2}
\end{aligned}
$$

Rest om te bewijzen met volledige inductie dat $T(n) = \frac{n^2}{4} - \frac{n}{2}$.

Voor het basisgeval $n = 2$ geldt $T(2) = 0$ en $\frac{2^2}{4} - \frac{2}{2} = 1 - 1 = 0$. Dit gaat dus op.

Voor het inductieve geval nemen we als inductieaanname: stel dat $T(n) = \frac{n^2}{4} - \frac{n}{2}$ voor alle tweemachten $n < N$ en met N ook een tweemacht. Dan moeten we laten zien dat $T(N) = \frac{N^2}{4} - \frac{N}{2}$:

$$
\begin{aligned}
T(N) &= 2T(\frac{N}{2}) + \frac{N^2}{8} && \text{(definitie)} \\
&= 2(\frac{N^2}{16} - \frac{N}{4}) + \frac{N^2}{8} && \text{(inductieaanname op } T(\tfrac{N}{2})) \\
&= \frac{N^2}{8} - \frac{N}{2} + \frac{N^2}{8} \\
&= \frac{N^2}{4} - \frac{N}{2}
\end{aligned}
$$

Hiermee is bewezen dat $T(n) = \frac{n^2}{4} - \frac{n}{2}$ voor alle $n = 2^k, n \geq 2$.

Opgave 3. (29 punten)

a. (5 punten) Merk ten eerste op dat de buitenste loop in feite een for-loop is voor $i = 1, \ldots, n-1$. Voor elke iteratie van de buitenste loop geldt dat $i \leq n - 1$ en dus dat $j \leq n$ (na regel 3). De tweede test op regel 4 gebeurt dus minstens één keer per iteratie van de buitenste loop, m.a.w., minstens $n - 1$ keer.

Daarmee kunnen we meteen de meeste regels al in verstandig verband brengen:

- regels 1 en 14 gebeuren precies $1 \in O(n-1)$ keer (d.w.z., $1 \in O(n-1)$ voor $n \geq 2$).
- de test op regel 2 gebeurt precies $n \in O(n-1)$ keer (d.w.z., $n \in O(n-1)$ voor $n \geq 2$).
- regels 3, 8 en 12 gebeuren precies $n - 1 \in O(n-1)$ keer.
- regels 5 en 6 gebeuren hooguit even vaak als de tweede test op regel 4.
- regels 9 en 10 gebeuren hooguit even vaak als regel 8, dus hooguit $n - 1 \in O(n-1)$ keer.

Rest nog de eerste test op regel 4. Deze gebeurt per iteratie van de buitenste loop hooguit één keer vaker dan de tweede test, dus hooguit $n - 1$ keer vaker. De tweede test gebeurt al minstens $n - 1$ keer, dus dit scheelt hooguit een constante factor 2, wat niet uitmaakt voor de orde van grootte. In orde van grootte gebeurt de eerste test op regel 4 dus hooguit even vaak als de tweede test op dezelfde regel.

b. (7 punten) Zoals beargumenteerd bij **a**, doet dit algoritme voor alle invoeren minstens $n-1$ arrayvergelijkingen: de tweede test op regel 4 gebeurt minstens één keer per iteratie van de buitenste loop. Het beste geval zou dus zijn (als dit mogelijk is) wanneer dit *precies* één keer is per iteratie van de buitenste loop.

Dit is inderdaad mogelijk. Hiervoor moet de tweede test op regel 4 voor geen enkele iteratie van de buitenste loop een tweede keer worden uitgevoerd. Met andere woorden: ofwel de tweede test moet bij de eerste iteratie van de binnenste loop al onwaar zijn (dus $A[j] > A[i]$), of de eerste test moet bij de tweede iteratie van de binnenste loop onwaar zijn (dus $j > n$). Dit laatste kan alleen waar zijn voor de laatste iteratie van de buitenste loop, $i = n-1$, want dan geldt in de tweede iteratie van de binnenste loop $j = i+2 = n+1$. Voor $i = 1, \ldots, n-2$ moet dus gelden dat $A[i+1] > A[i]$.

Met andere woorden: in de best case doet dit algoritme $n-1$ vergelijkingen. Dit komt voor bij precies alle rijtjes $A[1], \ldots, A[n]$ waarvan het deel $A[1], \ldots, A[n-1]$ oplopend is gesorteerd.

c. (9 punten) Voor elke $i = 1, \ldots, n-1$ gebeurt de tweede test op regel 4 *hooguit* voor de waarden $j = i+1, \ldots, n$ (voor $j > n$ faalt de eerste test immers al). Met andere woorden: hooguit $n-i$ keer. Om aan die $n-i$ keer te komen, moeten beide tests op regel 4 slagen voor alle $j < n$, anders termineert de loop eerder. De eerste test slaagt hiervoor triviaal, dus dit is een beperking op de tweede test. Voor $j = n$ slaagt de eerste test en maakt de uitkomst van de tweede test niet uit, want de eerste test faalt sowieso voor $j = n+1$.

Met andere woorden: om zo lang mogelijk bezig te zijn, moet gelden dat $A[j] < A[i]$ voor alle $1 \leq i < j < n$. Dit wil zeggen dat het deelrijtje $A[1], \ldots, A[n-1]$ aflopend is gesorteerd. De waarde van het laatste element maakt niet uit. In dat geval doet het algoritme $\sum_{i=1}^{n-1}(n-i) = \sum_{i=1}^{n-1} n - \sum_{i=1}^{n-1} i = n(n-1) - \frac{n}{2}(n-1) = \frac{n}{2}(n-1)$ vergelijkingen.

d. (8 punten) Er geldt nog steeds dat de tweede test op regel 4 minstens één keer wordt uitgevoerd per iteratie van de buitenste loop. Het verschil is nu dat er mogelijk minder van die iteraties zijn: als ons zwaarste element gewicht k heeft, slaan we de laatste k iteraties over. Het bepalen van het gewicht van dit element kost echter ook minstens k vergelijkingen, dus netto win je er geen vergelijkingen mee. Met andere woorden: het aantal vergelijkingen in de best case blijft hetzelfde, namelijk $n-1$. Wel verandert het aantal bijbehorende rijtjes: wanneer bijvoorbeeld het eerste element gewicht $n-1$ of $n-2$ heeft (d.w.z. groter is dan alle anderen, behalve misschien het laatste) kan dit in $n-1$ vergelijkingen worden geconstateerd, waarna het algoritme alle resterende iteraties overslaat. Dit rijtje levert dus ook $n-1$ vergelijkingen op.

Het aantal vergelijkingen in de worst case verandert wél. Je doet per iteratie van de buitenste loop namelijk alleen meer dan één vergelijking als het huidige element $A[i]$ een gewicht groter dan 0 heeft, maar een gewicht groter dan 0 zorgt er ook voor dat je latere iteraties overslaat. Het is dus niet meer mogelijk om $\frac{n}{2}(n-1)$ vergelijkingen te doen. De invoerrijtjes zoals beschreven bij **c** ($A[1], \ldots, A[n-1]$ aflopend gesorteerd) zorgen er zelfs voor dat $A[1]$ gewicht $n-1$ of $n-2$ heeft, waardoor het algoritme daarop stopt na $n-1$ vergelijkingen. Dit zijn dus best-case-rijtjes geworden. Merk op dat een rijtje waarvan de eerste helft aflopend is gesorteerd nog steeds een kwadratisch aantal vergelijkingen oplevert, dus er zijn nu andere worst-case-rijtjes.

Opgave 4. (32 punten)

a. (10 punten) We geven een *niet-deterministisch polynomiaal* algoritme A voor VC. A heeft als invoer een ongerichte graaf $\mathcal{G} = (V, E)$ en een geheel getal $k > 0$. We mogen aannemen (zie de opgave) dat het aantal knopen van $\mathcal{G}$ bekend is en dat $V = \{1, 2, \ldots, n\}$.

A doet bijvoorbeeld het volgende:

1. Fase 1 (gokfase)

   Er wordt een string s gegenereerd, hierna te interpreteren als een rij gehele getallen $s_1, \ldots, s_m$. Deze zou een vertex cover moeten voorstellen van grootte k; dit wordt gecontroleerd in fase 2. Het genereren van s kan in $O(|s|)$.

2. Fase 2 (verificatiefase)

   Er wordt gecontroleerd of s inderdaad een vertex cover van grootte k voorstelt. De string s moet in dat geval dus een rijtje van k verschillende knopen uit V voorstellen, zodat alle takken in E een uiteinde hebben in de knopen van s. Concreet voeren we de volgende controles uit:

   (a) we controleren dat alle s_i getallen tussen 1 en n zijn (en dus knopen van $\mathcal{G}$ voorstellen): s aflopen en controleren of voor elke s_i geldt dat $1 \leq s_i \leq n$. Dit kan in $O(|s|)$ stappen.

   (b) we controleren dat alle s_i verschillend zijn: s aflopen en voor elke s_i naar rechts lopen en kijken of die waarde nogmaals voorkomt; zo ja, dan stoppen we, zo nee, dan bekijken we de volgende s_i. Dit kan in $O(|s|^2)$ stappen.

   (c) we controleren dat s precies k knopen bevat: s aflopen en tellen. Dit kan in $O(|s|)$ stappen.

   Als aan deze voorwaarden is voldaan stelt s een deelverzameling van de knopen van $\mathcal{G}$ voor ter grootte k. Dit is dus een soort syntactische controle. Dan resteert de belangrijkste controle, namelijk dat s ook een vertex cover voorstelt:

   (d) we controleren dat van alle takken $(u, v) \in E$ ofwel u of v voorkomt in de deelverzameling die wordt voorgesteld door s: E aflopen en voor elke tak in E de knopen in s aflopen totdat één van de uiteinden van de tak is gevonden (of de knopen op zijn). Dit kan in $O(|E| \times |s|)$ stappen.

   Als de vier controles positief zijn, stelt s een vertex cover in $\mathcal{G}$ ter grootte k en retourneert het verificatiealgoritme (fase 2) True. Zodra een van de controles niet klopt, wordt False geretourneerd of beginnen we een oneindige loop.

3. Fase 3 (uitvoerfase)

   Als fase 2 True oplevert, wordt "ja" uitgevoerd. Anders is er geen uitvoer.

Nu geldt: fase 2 geeft True $\Leftrightarrow$ s is een vertex cover in $\mathcal{G}$ ter grootte k. En dus: het antwoord van A op invoer $\langle \mathcal{G}, k \rangle$ is "ja" (per definitie) $\Leftrightarrow$ er is een executie van A die "ja" oplevert $\Leftrightarrow$ er is een string s waarop A in fase 2 True geeft $\Leftrightarrow$ (ons conrete algoritme) er is een string s die een vertex cover in $\mathcal{G}$ ter grootte k voorstelt $\Leftrightarrow$ $\mathcal{G}$ heeft een vertex cover ter grootte k $\Leftrightarrow$ $\langle \mathcal{G}, k \rangle$ is een ja-instantie van VC.

Kortom, A is inderdaad een (niet-deterministisch) algoritme voor VC. Bovendien is het polynomiaal. Immers, voor ja-executies stelt de gegokte s een deelverzameling van de knopen voor. In dat geval is dus $|s| \in O(|V|) \subseteq O(|\mathcal{G}|) \subseteq O(|x|)$. De (ja-)executie met die s kan derhalve zeker wel in: $O(|s|)$ (fase 1) + $O(|s|) + O(|s|^2) + O(|s|) + O(|E| \times |s|)$ (fase 2) + $O(1)$ (fase 3) $\subseteq O(|x|) + O(|x| + O(|x|^2) + O(|x| + O(|x|^2) + O(1) \subseteq O(|x|^2)$ stappen. Er is dus voor ja-instanties een ja-executie die $O(|x|^2)$ stappen doet, en dat is polynomiaal in $|x|$, de grootte van de invoer. Derhalve is A polynomiaal.

Conclusie: A is een niet-deterministisch polynomiaal ($O(|x|^2)$) algoritme voor VC.

b. (5 punten) Als ϕ een ja-instantie is van 3SAT, bestaat er een waardering z van de in ϕ voorkomende logische variabelen die ϕ waarmaakt, d.w.z., die voor alle clausules in ϕ minstens één literal waarmaakt. Hieruit construeren we een vertex cover bestaande uit $n + 2m$ knopen in $\mathcal{G}_\phi$.

Merk allereerst op dat elke vertex cover van $\mathcal{G}_\phi$, per constructie, minstens één knoop moet bevatten van elk paar (v_i, v_i'), om de tak tussen die twee knopen af te dekken. Dit zijn er in totaal minstens n. Verder moet elke vertex cover van $\mathcal{G}_\phi$, wederom per constructie, minstens twee knopen bevatten van elk drietal (w_1^r, w_2^r, w_3^r) om de drie takken tussen die drie knopen af te dekken. Dit zijn er in totaal minstens $2m$. *Elke* vertex cover van $\mathcal{G}_\phi$ bestaat dus uit minstens $n + 2m$ knopen. We zijn op zoek naar een vertex cover met $n + 2m$ knopen, dus deze moet *precies* één knoop bevatten van elk paar (v_i, v_i') en *precies* twee van elk drietal (w_1^r, w_2^r, w_3^r).

Van de paren (v_i, v_i') nemen we telkens v_i als de waardering z de variabele x_i waarmaakt, en v_i' als z de variabele x_i onwaar maakt. Een waardering z geeft elke variabele precies één waarde, dus dit geeft ons precies n knopen. Van de drietallen (w_1^r, w_2^r, w_3^r) nemen we telkens degenen waarvan de bijbehorende literal onder z niet wordt waargemaakt, en zo nodig nog één of twee willekeurige om te komen tot twee. Omdat z waarmakend is, zijn van elke clausule hooguit twee literals onwaar. Dit geeft ons precies $2m$ knopen.

Het is duidelijk dat deze knopen samen alle takken afdekken tussen knopen v_i, v_i' (eerste punt van de constructie van $\mathcal{G}_\phi$) en tussen knopen w_j^r, w_k^r (tweede punt van de constructie). Verder heeft elke knoop w_j^r een tak met de knoop v_i of v_i' die dezelfde literal voorstelt (derde punt van de constructie). Voor elk drietal geldt dat twee van de knopen w_j^r al in onze verzameling zitten en dat de bijbehorende takken daardoor zijn afgedekt. Van de overgebleven knoop w_j^r weten we dat, vanwege onze keuze van knopen, de bijbehorende literal waar is onder z, waardoor (wederom per knoopkeuze) de bijbehorende knoop v_i of v_i' ook in onze verzameling zit en de tak dus wordt afgedekt door v_i of v_i'. Met andere woorden: we hebben precies $n + 2m$ knopen gekozen die samen alle takken afdekken, dus we hebben aangetoond dat $\mathcal{G}_\phi$ een vertex cover heeft ter grootte $n + 2m$ en $T(\phi) = \langle \mathcal{G}_\phi, n + 2m \rangle$ is een ja-instantie van VC.

c. (5 punten) Als $T(\phi)$ een ja-instantie is voor VC, bestaat er een deelverzameling V' van de knopen in $\mathcal{G}_\phi$ bestaande uit $n + 2m$ knopen zodanig dat elke tak in $\mathcal{G}_\phi$ een uiteinde heeft in V'. Zoals beargumenteerd bij **b**, moeten dit n knopen v_i of v_i' zijn (één per paar) en $2m$ knopen w_j^r (twee per drietal).

Kies als waardering z telkens voor variabele x_i de waarde True als $v_i \in V'$ en False als $v_i' \in V'$. Omdat V' van elk paar (v_i, v_i') precies één knoop moet bevatten, geeft dit een goedgedefinieerde waardering z. Verder weten we dat van elk drietal knopen w_j^r, die overeenkomen met de clausules, twee voorkomen in V'. Elk van deze knopen heeft een tak lopen naar een knoop v_i of v_i'. Twee hiervan worden afgedekt door de gekozen knopen in V', maar de derde moet dan worden afgedekt door de bijbehorende knoop v_i of v_i', die dezelfde literal voorstelt als de knoop w_j^r waarmee deze is verbonden. Met andere woorden: door z op deze manier te kiezen, garanderen we dat elke clausule minstens één literal bevat die wordt waargemaakt door z. Daarmee is ϕ een ja-instantie van 3SAT.

d. (2 punten) P is NP-hard als $Q \leq_\mathsf{P} P$ voor alle $Q \in \mathcal{NP}$. P is NP-volledig als (i) P NP-hard is en (ii) $P \in \mathcal{NP}$.

e. (10 punten)

(i) Ja, dat kan. We weten dat (1) 3SAT $\in \mathcal{NPC}$, dus (zie **d**) 3SAT is NP-hard en (zie **d**) alle problemen in $\mathcal{NP}$ reduceren naar 3SAT. We weten ook dat (2) VC $\in \mathcal{NP}$, dus VC $\leq_\mathsf{P}$ 3SAT.

(ii) Ja, dat kan. We weten dat (1) 3SAT $\in \mathcal{NPC}$, dus (zie **d**) 3SAT is NP-hard en (zie **d**) alle problemen in $\mathcal{NP}$ reduceren naar 3SAT. We weten ook dat (3) 3SAT $\leq_\mathsf{P}$ VC, en daarmee

(vanwege de transitiviteit van $\leq_{\mathsf{P}}$) dat alle problemen in $\mathcal{NP}$ reduceren naar VC. Met andere woorden: VC is NP-hard. Verder weten we dat (2) VC $\in \mathcal{NP}$, dus (zie **d**) VC $\in \mathcal{NPC}$.

(iii) Ja, dat is bijzonder. Aangezien $Q \in \mathcal{P}$, zou dit betekenen dat VC ook in polynomiale tijd kan worden opgelost en dus in $\mathcal{P}$ zit. Omdat VC (zie (ii)) NP-volledig is en dus in het bijzonder NP-hard, zouden we daarmee (vanwege de transitiviteit van $\leq_{\mathsf{P}}$) van *elk* probleem in $\mathcal{NP}$ kunnen aantonen dat het in $\mathcal{P}$ zit. Met andere woorden: dit impliceert dat $\mathcal{NP} \subseteq \mathcal{P}$, en, omdat $\mathcal{P} \subseteq \mathcal{NP}$ (bekend van college), dat $\mathcal{P} = \mathcal{NP}$.

(iv) Nee, dit is niet bijzonder. Aangezien $Q \in \mathcal{P}$ en $\mathcal{P} \subseteq \mathcal{NP}$ (bekend van college), geldt ook dat $Q \in \mathcal{NP}$. Omdat VC (zie (ii)) NP-volledig is en dus in het bijzonder NP-hard, weten we daarmee al dat elk probleem in $\mathcal{NP}$ polynomiaal is te reduceren naar VC, inclusief Q.