(Beknopte) antwoorden opgaven werkcollege 4

Opgave 18

a.

- array-elementen vergelijken is een fundamentele operatie voor dit algoritme; de werking ervan hangt af van de uitslag van die vergelijking
- vanwege $n \geq 3$ en de initialisatie van $found$ en $gehad$ wordt de hele body van de buitenste while voor $i = 1$ altijd uitgevoerd. Immers in de eerste ronde ($i = 1$) geldt dat $i \leq \lceil \frac{n}{2} \rceil$ en $found$ = False en $gehad[i]$ = False. Dat betekent (want j loopt dan van 2 tot en met n) dat de test op regel (12) in elk geval $n - 1$ keer wordt gedaan. Dus $\#(12) \geq n - 1$ (&).
- regels (1), (2), (3), (5) en (6) gebeuren elk respectievelijk 1, $n + 1$, n, 1 en 1 keer; allemaal in $O(n)$ en daarmee in $O(\#(12))$. (Preciezer: $n + 1 = n - 1 + 2 \leq 2(n - 1) \leq 2 \cdot \#(12)$ en de rest analoog.)
- $\#(18) \leq \#(17) = \#(9) = \#(10) \leq \#(8) = \#(21) \leq \#(7) \leq \lceil \frac{n}{2} \rceil + 1$. In orde van grootte kleiner of gelijk aan $\#(12)$, immers $\lceil \frac{n}{2} \rceil + 1 \leq n \in O(\#(12))$ als hierboven.
- regel (13) gebeurt hooguit even vaak als regel (12); regel (15) even vaak als regel (12).
- de test in regel (11) gebeurt per doorgang door de buitenste while hooguit 1 keer vaker dan regel (12), dus in totaal hebben we: $\#(11) \leq \#(12) + \lceil \frac{n}{2} \rceil \leq 2 \cdot \#(12)$ vanwege (&).

Conclusie: in orde van grootte gebeurt elke andere operatie hooguit even vaak als de test in regel (12). Met andere woorden: de test in regel (12) is maatgevend voor de complexiteit.

b.

- $i = 1$: $gehad[1]$, $gehad[3]$ en $gehad[7]$ worden *True*; niet gevonden ($3 < 5\frac{1}{2}$), door; $n - 1 = 10$ vergelijkingen
- $i = 2$: $gehad[2]$ en $gehad[4]$ worden *True*; niet gevonden ($2 < 5\frac{1}{2}$), door; $n - 2 = 9$ vergelijkingen
- $i = 3$: $gehad[3]$ = *True*, sla over; 0 vergelijkingen
- $i = 4$: $gehad[4]$ = *True*, sla over; 0 vergelijkingen
- $i = 5$: $gehad$ wordt *True* op posities $5, 6, 8, 9, 10, 11$; gevonden ($6 > 5\frac{1}{2}$), stop; $n - 5 = 6$ vergelijkingen

Totaal $10 + 9 + 6 = 25$ vergelijkingen.

c. Het slechtste geval komt voor wanneer de body van de buitenste while zo vaak mogelijk gebeurt, dus voor $i = 1, \ldots, \lceil \frac{n}{2} \rceil$, en als daarbij $gehad[i]$ steeds False is. Dat betekent dat $found$ voor $i = 1, \ldots, \lceil \frac{n}{2} \rceil - 1$ False blijft en hooguit True wordt voor $i = \lceil \frac{n}{2} \rceil$ en dat $gehad[i]$ in regel (8) voor elke i op dat moment False is. Dat is dan en slechts dan het geval als: (i) $A[i]$ voor $i = 1, \ldots, \lceil \frac{n}{2} \rceil - 1$

geen major is en (ii) $A[1]$ t/m $A[\lceil \frac{n}{2} \rceil]$ allemaal verschillend zijn. Dus: de eerste $\lceil \frac{n}{2} \rceil$ elementen moeten verschillend zijn (anders sla je daar wat over vanwege *gehad*) en de eerste $\lceil \frac{n}{2} \rceil - 1$ moeten geen major zijn (anders stop je, maar in de laatste iteratie maakt dat niet uit).

In het slechtste geval worden dan $(n-1) + (n-2) + \cdots n - \lceil \frac{n}{2} \rceil = \sum_{i=1}^{\lceil \frac{n}{2} \rceil} (n-i)$ arrayvergelijkingen gedaan en dit is gelijk aan $\frac{1}{2}n(n-1) - \frac{1}{2}\lfloor \frac{n}{2} \rfloor (\lfloor \frac{n}{2} \rfloor - 1) \in \Theta(n^2)$

d. We zagen al dat altijd minsten $n-1$ vergelijkingen gedaan worden; dit aantal kan ook bereikt worden. De best case komt voor dan en slechts dan als ofwel (i) *found* meteen in de eerste ronde ($i = 1$) True wordt, ofwel (ii) als in de eerste ronde *gehad*$[i]$ voor elke $i = 1, \ldots, \lceil \frac{n}{2} \rceil$ op True wordt gezet., dus als de eerste $\lceil \frac{n}{2} \rceil$ elementen gelijk zijn. In het beste dus geval $n-1$ vergelijkingen, als ofwel $A[1]$ major is (dan wordt *found True*), ofwel de eerste $\lceil \frac{n}{2} \rceil$ lementen zijn (wat bij even n geen major hoeft te betekenen) zodat je elke keer *gehad*$[i]$ op *True* hebt.

e. In het slechtste geval doe je nog steeds $\sum_{i=1}^{\lceil \frac{n}{2} \rceil} i$ vergelijkingen, maar nu moeten de eerste $\lceil \frac{n}{2} \rceil - 1$ elementen uniek zijn in het array (en dus ook niet voorkomen in de tweede helft), anders sla je daar dingen over; het laatste bekeken element ($A[\lceil \frac{n}{2} \rceil]$) mag wel voorkomen in de tweede helft, daarna stopt het algoritme toch. Er zijn dus sowieso minder worst case gevallen dan eerst.

Met de aanpassing is de test uit regel (12) niet meer maatgevend. Er is een klasse van invoerrijtjes te construeren waarvoor regel (11) en de nieuwe regel in orde van grootte strikt vaker worden uitgevoerd dan regel (12). Bijvoorbeeld: laat de eerste $\sqrt{n}$ elementen verschillend zijn, en laat alle andere elementen gelijk zijn aan $A[1]$ of $A[2]$, op een manier dat A geen major bevat. Dan wordt regel (12) $O(n) + O(\sqrt{n}^2) = O(n)$ keer uitgevoerd en regels (11) en de nieuwe $O(n\sqrt{n})$.

Opgave 29

We bekijken een implementatie van de toernooimethode met een heapachtige structuur.

a. Gevraagd is aan te tonen dat het algoritme uit de opgave het maximum van A in $H[1]$ zet.

Dit kun je het beste zien door het array af te beelden op een complete binaire boom met $2n-1$ knopen. In de n bladeren (corresponderend met de n arrayelementen $H[n]$ t/m $H[2n-1]$) bevinden zich de waarden $A[1]$ t/m $A[n]$, zie regel 1 van het algoritme. En verder geldt dan: de twee kinderen van $H[i]$ bevinden zich in $H[2i]$ en $H[2i+1]$. Het algoritme berekent telkens het maximum van twee kinderen en slaat deze op in de ouder, totdat we bij de wortel ($H[1]$) zijn aangekomen. De winnaar gaat dus als het ware telkens door naar de volgende ronde. Op deze manier bevat de knoop met $H[i]$ telkens het maximum van de bladeren van de subboom waarvan $H[i]$ de wortel is. Dit herhalende bevat de wortel van de boom uiteindelijk het maximum van de waarden uit alle bladeren, dus het maximum van A.

Teken zelf de bijbehorende boom voor $A = 3; 2; 4; 5; 1$, dus $n = 5$. De toernooiboom wordt van onder naar boven ingevuld, beginnend in de eerste ouderknoop van onder af (dus $H[n-1]$).

b. De elementen die (direct) van de winnaar hebben verloren kunnen als volgt bepaald worden. Begin bovenaan in de boom, dus bij $H[1]$. Dat is de uiteindelijke winnaar. Als er kinderen zijn, bevat het kind waarvan de waarde niet gelijk is aan $H[1]$ (uit de huidige knoop dus) een waarde die (direct) heeft verloren van de winnaar. Verplaats vervolgens de index naar het kind waarvan de waarde wel gelijk is aan die van huidige knoop ($H[1]$ dus). Herhaal dit proces zolang het kan. Dit levert je alle directe verliezers van het maximum op.

c. Merk op dat het op-een-na-grootste element direct moet hebben verloren van de winnaar. Is dat immers niet zo, dan heeft dit element verloren van een ander element, die (al dan niet indirect) verloren heeft van de winnaar. In dat geval kan het element nooit het op-een-na-grootste zijn. Of anders geredeneerd: de op-een-na-grootste heeft alleen van de grootste verloren; van alle andere waarden wint hij.

Opgave 31

Zie uitwerking website

Opgave 32

Voor maximum en minimum: zie hoofdstuk 6.5 van het dictaat. Aangepast aan alleen maximum: alleen interesse in N, W, V. Tabel blijft ongewijzigd (behalve dat alles met WV wegvalt). Beginsituatie is n van type N, en 0 van W en V. Eindsituatie is 0 van N, 1 van W en $n-1$ van V. Per vergelijking (ongeacht type) wordt V hooguit één opgehoogd, dus je hebt minstens $n-1$ vergelijkingen nodig om te komen op $n-1$ van V.

Opgave 33

Gevraagd wordt een adversary-argument voor het controleren van de samenhangendheid van een ongerichte graaf. Het algoritme stelt steeds vragen van het type: zit er een tak tussen knoop v en knoop w? Een adversary-strategy kan als volgt zijn. Deel eerst de n knopen op in twee disjuncte deelverzamelingen V_1 en V_2 ter grootte $\lfloor n/2 \rfloor$ respectievelijk $\lceil n/2 \rceil$. Vervolgens antwoordt de adversary altijd ja als wordt gevraagd of twee knopen verbonden zijn die zij in dezelfde verzameling heeft geplaatst en nee als wordt gevraagd of twee knopen in verschillende verzame- lingen verbonden zijn. Op die manier moet het algoritme minstens $\lfloor n/2 \rfloor \times \lceil n/2 \rceil$ vragen stellen om erachter te komen dat de graaf niet samenhangend is. Het algoritme moet alle mogelijke verbindingen tussen V_1 en V_2 in elk geval gevraagd hebben. De laatst gevraagde mogelijk verbinding tussen V_1 en V_2 geeft in dit geval pas uitsluitsel. (Merk op dat de adversary bij het laatst gevraagde paar knopen (v, w) met $\in V_1$ en $w \in V_2$ ook ja kan antwoorden; het antwoord maakt daar niet uit. In beide gevallen moeten alle paren knopen met de ene knoop in V_1 en de andere in V_2 bevraagd worden.)