Tentamen Algoritmiek
Dinsdag 6 juni 2017, 14.00 – 17.00 uur

Als er om uitleg, toelichting of motivatie gevraagd wordt bij een opgave, is het belangrijk om die ook te geven.
Als je het antwoord op een onderdeel niet weet, en je hebt dat antwoord nodig bij een later onderdeel, dan kun je het antwoord 'kopen' bij de docent.
Globale puntenverdeling: 1: 24; 2: 32; 3: 31; 4: 13. **Veel succes!**

---

1. Brad en Karim hebben een lange vakantie voordat ze weer op mogen draven. Ze doen ondertussen een spelletje om de tijd te doden. Het is een variant van Nim. Ze beginnen met één stapel van $N \geq 2$ lucifers, maar in de loop van het spel kunnen er meerdere stapels onstaan. Om de beurt doen de twee spelers een zet, waarbij Brad begint. Een zet bestaat eruit dat er

   - óf van één (zelf te kiezen) stapel drie lucifers worden weggehaald. Deze drie lucifers verdwijnen uit het spel. Uiteraard moet de gekozen stapel wel minstens drie lucifers bevatten.
   - óf dat één (zelf te kiezen) stapel in twee (ongeveer) even grote stapels wordt verdeeld. Dat wil zeggen: als de stapel $2m$ lucifers bevatte (even dus), moeten de twee nieuwe stapels elk precies m lucifers bevatten. Als de stapel $2m+1$ lucifers bevatte (oneven dus), moet de ene nieuwe stapel $m+1$ lucifers bevatten en de andere m lucifers. De gekozen stapel moet in dit geval minstens twee lucifers bevatten.

   Het spel eindigt als de speler die aan de beurt is, geen zet meer kan doen. De andere speler heeft dan gewonnen.

   (a) Wat zijn voor dit spel toestanden en acties (voor algemene $N \geq 2$)?
   Wanneer is een toestand een eindtoestand? Motiveer dit laatste antwoord.

   (b) Teken de toestand-actie-ruimte voor het geval $N = 6$. Geef bij *elke* toestand in je toestand-actie-ruimte aan of deze winnend is voor B (Brad) of voor K (Karim), te beginnen bij de eindtoestanden. Bepaal zo of het spel met $N = 6$ winnend is voor B of voor K. Toestanden die je al hebt uitgewerkt (of die feitelijk hetzelfde zijn als een toestand die je al hebt uitgewerkt) hoef je niet nogmaals uit te werken. Zet daar wel bij wie er wint, en verwijs naar de reeds uitgewerkte toestand.

   (c) Brad wil graag winnen, maar heeft geen zin om voor elke waarde van N eerst een complete toestands-actie-ruimte te tekenen. Hij zou al een stuk geholpen zijn, als hij wist of een gegeven toestand winnend is voor de speler die aan de beurt is. Kun jij hem helpen?

   Schrijf een *recursieve* C++-functie `bool winnend (...)`, die bepaalt of een gegeven toestand winnend is voor de speler die aan de beurt is. Bedenk hierbij eerst hoe je een toestand representeert, en licht de gekozen representatie toe. Geef (onderdelen van) de toestand vervolgens mee als parameter(s) aan de functie `winnend`.

   Bedenk dat een toestand winnend is voor de speler die aan de beurt is, als er minstens één vervolgstand (een toestand na een mogelijke zet) is die niet winnend is voor de speler die in die vervolgstand aan de beurt is. Je moet bij het aflopen van mogelijke zetten dan ook stoppen zodra je een dergelijke vervolgstand hebt gevonden.

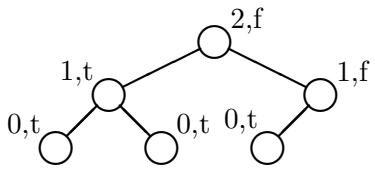
   Als het je niet lukt om voor het concrete spel van Brad en Karim een functie `winnend` te schrijven, dan kun je een deel van de punten verdienen door de algemene opzet van zo'n functie voor een tweepersoonsspel te geven.

---

2. Deze opgave gaat over binaire bomen, bestaande uit knopen uit de klasse `knoop` die er als volgt uitziet:

```
class knoop
{ public:
    knoop* links;
    knoop* rechts;
    int hoogte;
    bool gevuld;
}; // knoop
```

Voorbeeld:
Bij de knopen staat de waarde van de velden `hoogte` en `gevuld` (waarbij t=true, f=false) vermeld ná het aanroepen van de functie uit onderdeel (b).



In deze opgave noemen we een binaire boom *gevuld* als hij zoveel mogelijk knopen bevat, gegeven zijn hoogte.

(a) Teken een binaire boom van hoogte 3 met zoveel mogelijk knopen, ofwel een gevulde binaire boom van hoogte 3.

(b) Ga ervanuit dat we een binaire boom hebben, waarin de velden `hoogte` en `gevuld` nog een random waarde hebben. De boom kan leeg zijn.

Schrijf een *recursieve* C++-functie `void vulvelden (knoop *root)` die in elke knoop `x` in de boom met wortel `root` het veld `hoogte` vult met de hoogte van de subboom met wortel `x`, en het veld `gevuld` true/false maakt als de subboom met wortel `x` wel/niet gevuld is.

(c) Wanneer noemen we een binaire boom *compleet*?

(d) Beredeneer dat als een boom compleet is, dat dan voor ieder interne knoop (ieder niet-blad dus) alle drie de volgende eigenschappen gelden:

- de hoogte van zijn linker subboom is gelijk aan, of 1 hoger dan de hoogte van zijn rechter subboom;
- als de hoogtes van zijn subbomen gelijk zijn, dan is de linker subboom gevuld;
- als de hoogtes van zijn subbomen niet gelijk zijn, dan is de rechter subboom gevuld.

Hint: redeneer vanuit het ongerijmde: "als de $1^e/2^e/3^e$ eigenschap niet geldt, dan. . . ."

(e) Het is niet zo dat elke binaire boom met de drie eigenschappen uit het vorige onderdeel ook compleet is. Als we de eigenschappen iets aanpassen gaat het wel goed. Als voor elke interne knoop in een binaire boom alle drie de volgende eigenschappen gelden, dan is de boom compleet:

- de hoogte van zijn linker subboom is gelijk aan, of 1 hoger dan de hoogte van zijn rechter subboom;
- als de hoogtes van zijn subbomen gelijk zijn, dan is de linker subboom gevuld en is de rechter subboom compleet;
- als de hoogtes van zijn subbomen niet gelijk zijn, dan is de rechter subboom gevuld en is de linker subboom compleet.

Ga er nu vanuit dat de binaire boom niet leeg is, en dat de velden `hoogte` en `gevuld` van de knopen in de boom gevuld zijn met de juiste waardes, zoals beschreven bij onderdeel (b).

Schrijf een *niet-recursieve* C++-functie `bool iscompleet (knoop *root)` die de drie aangepaste eigenschappen gebruikt om te controleren of de boom met wortel `root` compleet is of niet. Als de boom compleet is, dan retourneert de functie true, en anders false.

3. Bij het afscheid van zijn vrienden heeft Eljero een puzzelboekje gekregen waar hij helemaal gek van wordt. Het lijkt zo simpel: elke puzzel bestaat uit twee woorden, die elk bestaan uit karakters 0 en/of 1. Het eerste woord P (van patroon) is M karakters lang. Het tweede woord T (van tekst) is N karakters lang. Er geldt dat $N \geq M \geq 1$ en de vraag is hoe vaak het woord P voorkomt als 'gecombineerd deelwoord' van T. Dat wil zeggen: op hoeveel manieren kun je precies het woord P krijgen door M letters te kiezen uit T en die in de oorspronkelijke volgorde achter elkaar te zetten. Bijvoorbeeld: het woord $P = 01$ komt vier keer voor in $T = 01101$:

$\underline{0}\ \underline{1}\ 1\ 0\ 1$
$\underline{0}\ 1\ \underline{1}\ 0\ 1$
$\underline{0}\ 1\ 1\ 0\ \underline{1}$
$0\ 1\ 1\ \underline{0}\ \underline{1}$

(a) Geef alle voorkomens van $P = 01101$ in $T = 01010101$. Gebruik daarbij een notatie als in het voorbeeld hierboven, of iets vergelijkbaars.

Voor de puzzels in zijn puzzelboekje komt Eljero zelf geen enkele keer uit op het antwoord dat op de laatste pagina van het boekje staat. Hij twijfelt derhalve of de antwoorden wel kloppen. Kun jij hem helpen?

We gaan Eljero helpen met behulp van bottom-up dynamisch programmeren (DP). We nemen daarbij aan dat het patroon op posities 0 t/m $M-1$ van P staat, en dat de tekst op posities 0 t/m $N-1$ van T staat. Vervolgens kijken we naar het deelprobleem dat een (voorste) gedeelte van P voorkomt in een (voorste) gedeelte van T. Concreet definiëren we $\texttt{aantal}[i][j]$ als het aantal voorkomens van de eerste i letters van P als 'gecombineerd deelwoord' van de eerste j letters van T. Uiteindelijk willen we dan $\texttt{aantal}[M][N]$ weten.

(b) Leg uit waarom voor $\texttt{aantal}[i][j]$ geldt:

$$\texttt{aantal}[i][j] = \begin{cases} 0 & \text{als } i > j \\ 1 & \text{als } i = 0 \text{ en } 0 \leq j \leq N \\ \texttt{aantal}[i][j-1] & \text{als } 1 \leq i \leq j \text{ en } P[i-1] \neq T[j-1] \\ \texttt{aantal}[i-1][j-1] + \texttt{aantal}[i][j-1] & \text{als } 1 \leq i \leq j \text{ en } P[i-1] = T[j-1] \end{cases}$$

(c) Laat nu $P = 01101$ en $T = 01010101$. Het array $\texttt{aantal}$ ziet er dan als volgt uit:

		j $\rightarrow$								
		0	1	2	3	4	5	6	7	8
i	0	1	1	1	1	1	1	1	1	1
$\downarrow$	1	0	1	1	2	2	3	3	4	4
	2	0	0	1	1	3	3	6	6	10
	3	0	0	0	0	1	1	4	4	10
	4					...				
	5					...				

Vul de laatste twee rijen van het array in.

(d) Wanneer we op bovenstaande manier voor algemene M en algemene N de puzzel met DP oplossen, wat is dan de tijdcomplexiteit van ons algoritme? Motiveer je antwoord.

Z.O.Z.

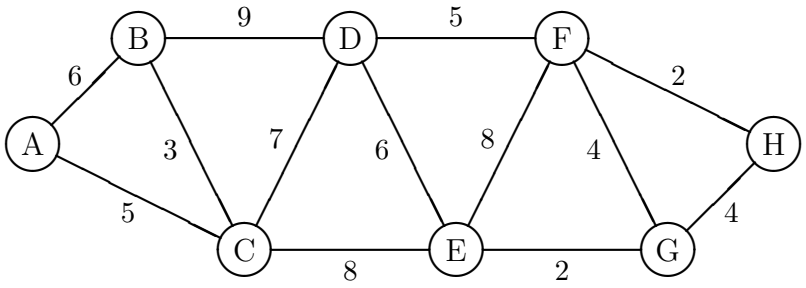
(e) In plaats van met het complete, hierboven beschreven tweedimensionale array `aantal` kunnen we het DP algoritme ook uitvoeren met een ééndimensionaal array (overeenkomend met één rij of één kolom van het tweedimensionale array). Leg uit hoe we dit kunnen doen, en geef een *niet-recursieve* C++-functie `int zoekpatroon (char P[], int M, char T[], int N)`, die een ééndimensionaal array `aantal` als lokale variabele heeft, en met het DP algoritme de puzzel van Eljero oplost.

Als je niet weet hoe je dit met een ééndimensionaal array moet doen, dan kun je een deel van de punten verdienen met een functie die gebruikmaakt van het oorspronkelijke, tweedimensionale array `aantal`.

---

4. (a) Laat G een samenhangende, ongerichte graaf zijn met gewichten op de takken. Wat verstaan we onder een *minimale opspannende boom* van G? Leg hierbij ook uit wat een opspannende boom is.

(b) Laat G de volgende graaf zijn:



Pas het algoritme van Kruskal toe om een minimale opspannende boom van G te bepalen. Leg uit hoe je tewerk gaat. Geef ook duidelijk aan, welke takken je in welke volgorde bekijkt en wel of niet aan je oplossing toevoegt. Teken ook de resulterende boom.
