Tentamen Algoritmiek
Dinsdag 2 juni 2009, 10.00 – 13.00 uur

Geef een duidelijke toelichting bij al je antwoorden.

Opgave 1. We bekijken de volgende variant op het Nim-spel. Als het spel begint liggen er n lucifers midden op tafel. Hierbij is n geheel, > 0 en *oneven*. Er zijn twee spelers, Romeo en Julia, die om de beurt een zet doen. Een zet is hier het wegnemen van één, twee of drie lucifers van de stapel in het midden van de tafel. De weggenomen lucifers legt men voor zich neer.
Het spel is afgelopen als er geen lucifers meer op de stapel midden op tafel liggen. De speler die op dat moment een oneven aantal lucifers in zijn/haar bezit heeft, heeft gewonnen. We nemen aan dat Romeo begint.

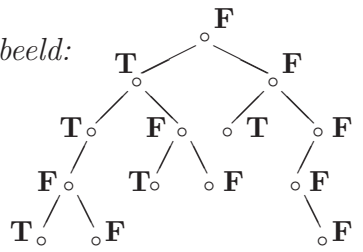
Voorbeeld met $n = 9$: stel dat er op een bepaald moment nog 4 lucifers op tafel liggen, Romeo heeft er 2, Julia 3, en Romeo is aan de beurt. Als hij er nu 3 pakt, dan zal Julia in de volgende beurt de laatste pakken, en heeft Romeo gewonnen (want hij heeft er dan 5, en Julia 4).

a. Wat zijn voor dit spel toestanden en acties (voor algemene n)?
b. Teken de toestand-actie-ruimte voor het geval $n = 5$, uitgaande van de beginsituatie.
c. Geef bij *elke* toestand aan of deze winnend is voor Romeo of voor Julia (bij perfect spel van beide spelers: d.w.z. beide doen steeds de/een zet die naar het beste resultaat leidt).
Is het spel met $n = 5$ winnend of verliezend voor Romeo ? Hoe moet Romeo dan wel Julia spelen om te winnen ? Licht je antwoord toe.
d. Beredeneer, zonder de volledige toestand-actie-ruimte te tekenen, of het spel voor $n = 7$ winnend is voor Romeo als hij begint. Geef duidelijk aan hoe je tot je antwoord komt.

Opgave 2. Gegeven een binaire boom met ingang `wortel`. Een knoop van de boom ziet er uit als hieronder links is aangegeven.

```
struct knoop {
   knoop* links;
   knoop* rechts;
   bool rechterbroer;
   knoop* extra;
}; // knoop
```

Voorbeeld:



a. Schrijf een *recursieve* C++-functie `void initialiseer(knoop* wortel)`, die alle `rechterbroer`-velden op `False` zet, en alle `extra`-velden op NULL.
b. Neem aan dat de `rechterbroer`-velden en de `extra`-velden gevuld zijn zoals in **a** bedoeld. Schrijf nu een *recursieve* C++-functie `void broeders(knoop* wortel)`, die in elke knoop het `rechterbroer`-veld en het `extra`-veld als volgt aanpast. *Als* de betreffende knoop een rechterbroer heeft wordt diens `rechterbroer`-veld `True` en moet de `extra`-pointer naar die rechterbroer wijzen. *Zo niet*, dan blijft het `rechterbroer`-veld `False` en moet de `extra`-pointer naar zijn vader wijzen (NULL indien die niet bestaat).

In de voorbeeldboom staat naast elke knoop de waarde van het `rechterbroer`-veld aangegeven.
c. Gegeven is een pointer `wijzer` die naar een knoop in de boom wijst. De wortel van de boom is nu niet bekend. Schrijf een niet-recursieve C++-functie `int nivo(knoop* wijzer)` die het nivo van de knoop in de boom oplevert. Ter herinnering: de wortel bevindt zich op nivo 0, diens kinderen op nivo 1, etcetera.

Opgave 3. Gegeven een array A ($A[0], \ldots, A[n-1]$, met $n \geq 2$), dat n positieve gehele getallen bevat, voorstellend de prijs van het aandeel LIACS op n opeenvolgende dagen (in het verleden). Men wil nu het volgende weten: wanneer had men het aandeel moeten kopen en wanneer verkopen, om zoveel mogelijk winst te maken? Met andere woorden: (P) voor welke i en j met $0 \leq i \leq j \leq n-1$ is $A[j] - A[i]$ maximaal?

De te schrijven functies hieronder moeten in C++ geschreven worden. Verder geldt: indien er meerdere paren (i, j) (met $j \geq i$) zijn waarvoor $A[j] - A[i]$ maximaal is, dan mag je zelf kiezen welk paar je als antwoord geeft.

a. Geef een brute force algoritme dat zo'n i en j oplevert.
b. We bekijken nu het probleem onder de restrictie dat het aandeel gekocht moet worden in de eerste helft van de periode (dus $0 \leq i \leq \frac{n}{2}-1$) en verkocht in de tweede helft (dus $\frac{n}{2} \leq j \leq n-1$). Neem aan dat n even is.
Geef een eenvoudig algoritme dat dagen i en j oplevert waarvoor de winst $A[j] - A[i]$ maximaal is. Het algoritme moet lineair zijn.
c. Geef een divide-and-conquer algoritme voor het oorspronkelijke probleem (P). Verdeel hierbij het array in twee gelijke delen. Neem aan dat n een 2-macht is. Hier moet dus een recursieve functie `maxwinst(l,r,i,j)` worden geschreven die het probleem oplost voor het deelarray $A[\mathtt{l}], \ldots, A[\mathtt{r}]$ ter lengte een 2-macht.
Gebruik o.a. de methode uit **b** (maar dan aangepast voor deelarrays; je hoeft de functie uit **b** niet helemaal te herschrijven, maar geef wel aan hoe deze –inclusief heading– verandert).

Opgave 4. Door het bedrijf Szybkość moet een groot project worden uitgevoerd. Dit project bestaat uit n verschillende taken die elk door precies één persoon moeten worden gedaan. Gelukkig heeft het bedrijf ook precies n medewerkers in dienst die hiervoor zullen worden ingezet. Iedere persoon zal precies één taak toebedeeld krijgen. Afhankelijk van de competentie doet een medewerker een bepaald aantal uren over een taak. Dit is opgeslagen in een tweedimensionaal array `uren`, waarin `uren`$[i][j]$ het aantal uren is dat medewerker i doet over taak j. Het doel is om het totaal aantal uren dat aan het project besteed wordt te minimaliseren.

Voorbeeld met $n = 4$

	taak 1	taak 2	taak 3	taak 4
medewerker W	90	65	50	1
medewerker X	60	75	8	80
medewerker Y	70	80	85	10
medewerker Z	11	22	80	75

De totale tijd besteed aan het project van de toewijzing W3, X1, Y4, Z2 is 142 uren. Dit is niet minimaal.

a. Geef een *gretig* algoritme voor dit probleem en pas het toe op het voorbeeld. Levert je algoritme een toewijzing met minimale totale tijd op?

b. Beschrijf een *backtracking* algoritme voor het probleem met algemene n en licht dit toe aan de hand van het voorbeeld. Teken ook een gedeelte (ten minste een kwart) van de bijbehorende state-space-tree, en geef daarin de volgorde aan waarin de knopen bekeken worden.
c. Zowel bij backtracking als bij branch and bound worden oplossingen stap voor stap opgebouwd. Leg uit hoe best-fit-first *branch and bound* werkt voor minimalisatieproblemen in het algemeen. Geef daarbij duidelijk aan wat de verschillen zijn met backtracking en waarom branch and bound (meestal) sneller een minimale oplossing zal vinden. Geef ook een mogelijk nadeel van branch and bound.
d. Beschrijf een best-fit-first branch and bound algoritme dat bovenstaand probleem oplost. Doe dit aan hand van het voorbeeld: pas het algoritme daarop toe en teken de bijbehorende state-space-tree. Geef daarin ook aan in welke volgorde de knopen bekeken worden. Wat voor afschatting van het te verwachten totale aantal uren gebruik je voor deeloplossingen?

Veel succes !

Puntenverdeling: 1: 20; 2: 25; 3: 25; 4: 30