Tentamen Algoritmiek
Dinsdag 10 juni 2014, 10.00 – 13.00 uur

Geef een **duidelijke toelichting** bij al je antwoorden.
Puntenverdeling: 1: 20; 2: 27; 3: 28; 4: 25; **Veel succes !**

Opgave 1. We bekijken het tweepersoonsspel *Dominono*, dat enigzins lijkt op het bekende boter-kaas-en-eieren (tic-tac-toe). Het wordt gespeeld op een vierkant n bij n bord. Twee spelers, in deze opgave Adam en Eva, zetten om de beurt hun teken (een `X` voor Adam, een `O` voor Eva) in een van de lege vakjes. Het is verboden een 'domino' (twee buurvakjes die hetzelfde teken bevatten) te maken, dat wil zeggen dat nooit een `X` direct naast/onder/boven een reeds aanwezige `X` mag worden geplaatst, en hetzelfde voor een `O`. Bij aanvang is het bord leeg; Adam (kruisje) begint. Het spel is afgelopen zodra een der spelers niet meer kan zetten (dat kan ook gebeuren omdat het bord vol is). In dat geval heeft de tegenstander dus gewonnen.
Om verwarring te verkomen: het doel van dit spel is dus niet om drie op een rij/kolom/diagonaal te krijgen, maar om te zorgen dat de tegenstander niet meer kan zetten.
Hieronder een mogelijk spelverloop, met $n = 3$ en lege vakjes aangeduid met `-`.

```
- - -    A      X - -    E     X - -    A     X - X
- - -   --->    - - -   --->   - - -   --->   - - -   (**)
- - -           - - -          - - O          - - O
```

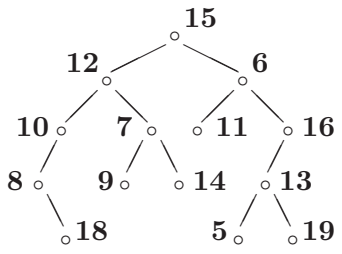
a. Wat zijn voor dit spel toestanden en acties (voor algemene n)?
b. Hoeveel eindstanden zijn er waarin het bord vol is? Motiveer je antwoord.
c. Teken de toestand-actie-ruimte voor het geval $n = 3$, uitgaande van toestand `(**)`. Let op: draai je papier een kwart slag (landscape) om genoeg ruimte te hebben. Geef bij elke eindtoestand aan of deze winnend is voor A of voor E. Bepaal hieruit voor *elke* toestand in je toestand-actie-ruimte voor wie deze winnend is, en ten slotte of `(**)` winnend is voor A of E en hoe gespeeld moet worden om te winnen.
Toestanden waarvan de laatste zetten vastliggen hoef je niet verder uit te werken. Je kunt daar meteen bijzetten wie er wint.

Opgave 2. Gegeven een binaire boom met ingang (ofwel: wortel) `wortel`. Hierin is `wortel` een pointer naar een `knoop`, waarbij:

```
struct knoop {
   knoop* links;
   knoop* rechts;
   int info;
   knoop* vader;
}; // knoop
```

Voorbeeld:

Bij de knopen staat de waarde van het info-veld vermeld.



Bij aanvang van deze opgave hebben de `vader`-velden de waarde `NULL`.

a. We willen voor elke knoop de `vader`-pointer laten wijzen naar zijn ouder in de boom. In de wortel moet de `vader`-pointer NULL blijven. Schrijf hiertoe een *recursieve* C++-functie `void ouders(knoop* wortel)`.

b. Neem aan dat alle `vader`-velden gevuld zijn zoals in **a.** bedoeld en dat de boom twee of meer knopen bevat (dus de wortel is geen blad).
Schrijf een *recursieve* C++-functie `void herfst(knoop* wortel)` die alle bladeren verwijdert. Let op: alleen de bladeren (in het voorbeeld 18, 9, 14, 11, 5, 19), niet de hele boom weggooien.
c. Neem aan dat alle `vader`-velden gevuld zijn zoals in **a.** bedoeld.
Schrijf een *niet-recursieve* C++-functie `int neven(knoop* p)` die het aantal volle neven van de knoop waar p naar wijst retourneert. Hierbij is de wortel van de boom onbekend. Een volle neef is een kind van je oom. In de voorbeeldboom: knoop 14 heeft 1 neef, namelijk 8; knoop 10 heeft er 2, namelijk 11 en 16.

Opgave 3. Jan heeft recentelijk een vakantiehuisje gekocht in een bergachtig gebied. Hij houdt van wandelen en nodigt regelmatig mensen uit voor een wandeling door de streek. Hij heeft een globale kaart van het gebied met de gemiddelde hoogtes per hectare ($100 * 100\ m^2$).
Omdat Jan het gebied nog niet erg goed kent en niet wil verdwalen, besluit hij alleen maar recht naar het zuiden en recht naar het oosten te lopen van hectare naar hectare op de kaart. Hij wandelt zo van de linkerbovenhoek van de kaart, waar zijn huis staat, naar de rechteronderhoek, waar hij zijn gasten op een heerlijke maaltijd trakteert. De linkerbovenhoek bevindt zich op positie (rij, kolom) = $(0, 0)$, de rechteronderhoek op $(n-1, n-1)$.

Voorbeeld: een hoogtekaart $H[i][j]$ voor $n = 6$:

0	3	5	6	5	4
1	2	2	2	7	3
1	8	7	6	6	2
3	9	7	4	5	3
7	8	7	5	6	2
4	5	8	4	5	0

Het hoogteverschil tussen twee aangrenzende hectaren is gedefinieerd als het absolute verschil tussen de twee hoogtes van deze hectaren en is dus altijd ≥ 0.
Het hoogteverschil van een wandelroute is gelijk aan de som van de hoogteverschillen op deze route. Het hoogteverschil van de wandelroute $(3,0)$, $(3,1)$, $(3,2)$, ..., $(3,5)$ is $6+2+3+1+2 = 14$.

a. Voor zijn oma zoekt Jan naar de route met zo min mogelijk hoogteverschil. Beschrijf een gretig algoritme voor het vinden van zo'n route.
Welke route vindt hij wanneer hij dit algoritme toepast op bovenstaand voorbeeld? Wat is daarvan het totale hoogteverschil?

Jan gaat het probleem oplossen met bottom-up dynamisch programmeren. Hierbij gebruikt hij een tweedimensionaal array M, waarbij $M[i][j]$ het minimale hoogteverschil is voor een wandelroute van (i, j) naar $(n-1, n-1)$.

b. Leg uit waarom de volgende recurrente betrekking geldt voor M, met $0 \leq i < n$ en $0 \leq j < n$:

$$M[i][j] = \begin{cases} 0 & \text{als } i = n-1 \text{ en } j = n-1 \\ M[n-1][j+1] + \mathbf{abs}(H[n-1][j+1] - H[n-1][j]) & \text{als } i = n-1 \text{ en } 0 \leq j < n-1 \\ M[i+1][n-1] + \mathbf{abs}(H[i+1][n-1] - H[i][n-1]) & \text{als } j = n-1 \text{ en } 0 \leq i < n-1 \\ \mathbf{min}(M[i+1][j] + \mathbf{abs}(H[i+1][j] - H[i][j]), & \\ \quad M[i][j+1] + \mathbf{abs}(H[i][j+1] - H[i][j])) & \text{als } 0 \leq i < n-1 \text{ en } 0 \leq j < n-1 \end{cases}$$

Hierin is **abs**(a) is de absolute waarde van a. Dus **abs**$(-3) = 3$ en **abs**$(3) = 3$.
Verder geeft **min**(a,b) het minimum van a en b.

c. Schrijf een algoritme in pseudocode of C++ dat, gebruikmakend van het array M en de recurrente betrekking uit vraag **b.**, het gevraagde minimale hoogteverschil berekent. Je mag ervan uitgaan dat functies **min** en **abs** al bestaan.
Geef duidelijk aan in welke volgorde je het array M vult, en waarom juist in die volgorde.

Wanneer we het algoritme uit vraag **c.** toepassen op het voorbeeld, krijgen we het volgende array M:

12	11	9	8	7	6
11	10	10	10	7	5
15	8	7	6	6	4
15	11	9	6	5	3
11	10	9	7	6	2
14	13	10	6	5	0

d. Bepaal uit deze M het minimale hoogteverschil en een bijbehorende wandelroute van $(0, 0)$ naar $(n-1, n-1)$. Leg daarbij in woorden uit hoe je deze wandelroute uit de arrays M en H bepaalt.

Opgave 4. Wie te laat komt bij het college Algoritmiek, moet de volgende keer op taart trakteren. Peters timing is niet zo goed en hij is al een aantal keer te laat geweest. Hij moet dus weer trakteren. Hij komt nog net voor sluitingstijd bij de plaatselijke supermarkt. Daar zijn gelukkig nog enkele taarten over. De taarten zijn voorgesneden en als geheel verpakt. Het aantal porties per verpakking ligt dus vast.
Peter weet niet precies hoeveel studenten er naar het college zullen komen en besluit om het aantal porties te maximaliseren. Peter heeft echter maar € 13,- bij zich. (N.B. dit is een variatie op het knapzak probleem)

a. Leg uit hoe best-fit-first branch and bound werkt voor maximalisatieproblemen in het algemeen. Geef daarbij o.a. aan hoe (deel)oplossingen gegenereerd worden, wat met branch bedoeld wordt en wat met bound, wat best-fit-first betekent, wanneer gesnoeid wordt, enz.

Peter heeft een budget $B =$ € 13,- en er zijn nog precies $n = 4$ voorgesneden taarten beschikbaar. In onderstaande tabel is per taart i het aantal porties p_i en de prijs e_i gegeven. De taarten staan aflopend gesorteerd op het aantal porties per euro.

taart	naam	#porties (p)	euros (e)	#porties/euro (p/e)
1	chocoladecake	16	€ 2,-	8
2	boterkoek	36	€ 12,-	3
3	vruchtenvlaai	14	€ 7,-	2
4	appeltaart	8	€ 8,-	1

Voor deze versie van het knapzakprobleem genereren we, beginnend met de lege verzameling, deeloplossingen (= (deel)verzamelingen) door in elke stap wel/niet de volgende taart erbij te kiezen. We houden bij hoeveel elke verzameling taarten kost en hoeveel porties deze verzameling in totaal bevat.

b. Uitgaande van een deeloplossing is een bovengrens voor het aantal porties in elke (complete) uitbreiding daarvan als volgt te berekenen:

1. Bij aanvang: $B * (p_1/e_1)$
2. Na de i-de stap: $p + (B - b) * (p_{i+1}/e_{i+1})$, met p het totale aantal porties van de reeds gekozen taarten en b de totale prijs daarvan.

Leg uit waarom dit voor een deeloplossing inderdaad een bovengrens is voor het aantal porties dat je kan kopen als je deze verder uitbreidt.
c. Bepaal m.b.v. het best-fit-first branch and bound algoritme hoeveel porties taart Peter maximaal kan kopen. Gebruik hierbij de bovengrens uit vraag **b.** Teken de bijbehorende state space tree. Geef daarin aan in welke volgorde de knopen bekeken worden en welke deeloplossingen gesnoeid worden en waarom.