Tentamen Algoritmiek
Dinsdag 9 juni 2015, 14:00 – 17:00 uur

Geef een **duidelijke toelichting** bij al je antwoorden.
Puntenverdeling: 1: 14 ; 2: 23; 3: 24; 4: 25; 5: 14 **Veel succes !**

Opgave 1. We bekijken een tweepersoonsspel genaamd *Clobber*, dat gespeeld wordt op een rechthoekig m bij n bord ($m, n \geq 1$). In de praktijk wordt het spel overigens meestal gespeeld op een 5 bij 6 bord. Bij aanvang van het spel is het bord geheel gevuld met witte en zwarte stenen, waarbij de witte en zwarte stenen om en om zijn neergelegd. Zie onderstaand voorbeeld met $m = n = 3$, waarbij we de witte stenen met een `O` en de zwarte stenen met een `X` aangeven. We hebben twee spelers, in deze opgave Mark en Angela, die om de beurt een zet doen. Een zet is hier het pakken van een eigen steen (een `O` voor Mark, een `X` voor Angela) en het daarmee slaan van een steen van de tegenstander op een aangrenzend (horizontaal of verticaal) vakje. Een steen slaan betekent dat de geslagen steen wordt weggehaald en de eigen steen (degene waarmee je slaat) daarvoor in de plaats wordt gelegd. Het spel is afgelopen zodra een der spelers niet meer kan zetten. In dat geval heeft de tegenstander gewonnen; die heeft de laatste zet gedaan. We spreken af dat Angela (`X` dus) begint.

Voorbeeld van de beginsituatie en een mogelijk spelverloop voor $m = n = 3$. Lege vakjes geven we aan met `-`.

```
O X O     A      X - O     M      X - -     A      X - -     M      X - -
X O X   --->     X O X   --->     X O O   --->     - O O   --->     - - O
O X O            O X O            O X O            X X O            X O O
                                   (*)                               (&)
```

In toestand `(&)` is Angela (`X` dus) aan de beurt. De twee laatste zetten liggen nu vast; Mark doet de laatste zet en wint.

a. (2 punten)
Wat zijn voor dit spel toestanden en acties (voor algemene m en n)?
b. (12 punten)
Teken de toestand-actie-ruimte voor het geval $m = n = 3$, uitgaande van toestand `(*)`, waar Angela aan de beurt is. Geef bij *elke* toestand in je toestand-actie-ruimte aan of deze winnend is voor A of voor M, te beginnen bij de eindstanden. Bepaal zo of `(*)` winnend is voor A of M 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.

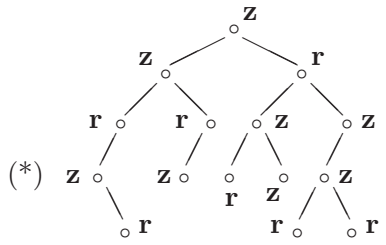
-vervolg op pagina 2

Opgave 2. Gegeven een binaire boom met ingang (ofwel: wortel) `wortel`. Hierin is `wortel` een pointer naar een `knoop`, die er als volgt uitziet:

```
class knoop {
  public:
    knoop* links;
    knoop* rechts;
    char kleur;
    int zwart;
}; // knoop
```

Voorbeeld:

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



Bij aanvang van deze opgave hebben de `kleur`-velden de waarde r (rood) of z (zwart). De `zwart`-velden hebben nog geen waarde.

Een *red-black tree* is een binaire boom waarbij alle knopen rood of zwart gekleurd zijn. Zo'n boom moet aan een aantal speciale eigenschappen voldoen, waaronder deze: (1) de kinderen van een rode knoop zijn altijd zwart en (2) elk pad van de wortel tot een knoop met hooguit één kind heeft hetzelfde aantal zwarte knopen[1]. Om eigenschap (2) te kunnen controleren moeten de `zwart`-velden goed gevuld zijn. Hierover gaan de onderdelen **b** en **c**.
De voorbeeldboom voldoet overigens aan eigenschap (1), maar niet aan (2). Immers, het pad waarbij je vanuit de wortel eerst naar rechts, dan naar links en dan weer naar links loopt bevat twee zwarte knopen, terwijl het pad van de wortel naar (*) drie zwarte knopen heeft.

a. (7 punten)
Schrijf een *recursieve* C++-functie `bool roodzwart(knoop* wortel)` die `true` oplevert dan en slechts dan als voor elke knoop in de boom met ingang `wortel` eigenschap (1) geldt, dus: als een knoop rood is moeten zijn kinderen zwart zijn.
b. (9 punten)
We bekijken nu de `zwart`-velden. In elke knoop moet dit veld aangeven hoeveel zwarte knopen er liggen op het pad van de wortel tot die knoop, waarbij de knoop zelf wordt meegeteld.
Schrijf een *recursieve* C++-functie `void black(knoop* wortel, int noir)`, die de `zwart`-velden van alle knopen uit de boom met ingang `wortel` met de juiste waarde vult. De parameter `noir` geeft aan hoeveel zwarte knopen je al op je pad bent tegengekomen voordat je knoop `wortel` tegenkomt. Bij eerste aanroep is deze waarde dus nul.
In de voorbeeldboom moet het linkerkind van de wortel een `zwart`-waarde 2 krijgen, het rechterkind 1 en van knoop (*) moet het `zwart`-veld 3 worden.
c. (7 punten)
Laten de `zwart`-velden van de boom met ingang `wortel` goed gevuld zijn. Iemand verandert echter stiekem het `zwart`-veld van de onderste knoop van het zigzagpad links-rechts-links-rechts-links-... We moeten de `zwart`-waarde van die knoop dus weer goed zetten.
Schrijf daartoe een *niet-recursieve* C++-functie `void herstel(knoop* wortel)` die het zigzagpad links-rechts-links-rechts-links-... afloopt en in de laatste knoop op dat pad de `zwart`-waarde weer herstelt.

-vervolg op pagina 3

[1] anders geformuleerd: elk pad tot een null-pointer bevat hetzelfde aantal zwarte knopen

Opgave 3. Gegeven een array $A = A[0], A[1], \ldots, A[n-1]$ dat n (≥ 1) verschillende gehele getallen bevat. Neem aan dat n een 2-macht is. We zoeken een aaneengesloten stijgende deelrij van A ter lengte k, dat wil zeggen een rijtje $A[i], A[i+1], \ldots, A[i+k-1]$ met $A[i] < A[i+1] < \ldots < A[i+k-1]$. Hierin is k een geheel getal met $2 \leq k \leq n$.
We gaan dit probleem op drie manieren oplossen: met brute force, met decrease-and-conquer en met divide-and-conquer.
Er moeten bij **a**, **b** en **c** functies geschreven worden die bepalen of A een aaneengesloten stijgend deelrijtje ter lengte k bevat. Zo ja, dan moet de functie een index opleveren waar zo'n rijtje begint. Als zo'n rijtje niet bestaat moet -1 geretourneerd worden.

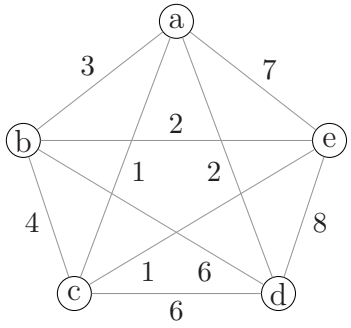
Voorbeeld: Als $A = 12, 16, 13, 15, 18, 20, 17, 11$, dan bevat A een aaneengesloten deelrij ter lengte 3, bijvoorbeeld $A[2], A[3], A[4]$. Echter, A bevat geen aaneengesloten stijgende deelrij ter lengte 5. Voor deze A en $k = 3$ moeten de te schrijven functies dus 2 (of 3) opleveren; als $k = 5$ moet -1 geretourneerd worden.

a. (6 punten)
Geef een voor de hand liggend (brute force) iteratief algoritme dat de gevraagde index (of -1) oplevert. Schrijf hiervoor een C++-functie `int stijgend(int A[ ], int n, int k)`.
b. (8 punten)
Geef een decrease-by-one algoritme voor bovenstaand probleem. Schrijf daartoe een *recursieve* C++-functie `int oplopend(int A[ ], int k, int i)`, die het probleem oplost voor het (deel)array $A[0], A[1], \ldots, A[i]$ ($i \geq 0$). De aanroep `return oplopend(`$A, k, n-1$`);` geeft dan uiteraard het antwoord voor het hele array.
c. (10 punten)
Geef nu een divide-and-conquer algoritme dat het probleem oplost. Het array dient hiervoor in twee gelijke delen te worden verdeeld.
Schrijf een *recursieve* C++-functie `int deelrij(int A[ ], int k, int links, int rechts)` die het probleem oplost voor het deelarray A[`links`], ..., A[`rechts`] ter lengte een 2-macht.
Opmerking: je hoeft hier niet persé een efficiënt algoritme te bedenken; het gaat om het verdeel-en-heers mechanisme.

Bonus (4 punten)
Bepaal hoeveel vergelijkingen tussen array-elementen je algoritme uit **a** doet in de best case en in de worst case en voor wat voor rijtjes dat voorkomt. Leg je antwoord uit.

-vervolg op pagina 4

Opgave 4. We gaan met de trein een rondreis maken langs n grote steden in Nederland. We willen uiteraard wel alle n steden aandoen, maar daarbuiten willen we zo weinig mogelijk tussenstations passeren. Het spoorwegnet tussen deze steden onderling kan beschreven worden met behulp van een complete, ongerichte graaf met n knopen (= de grote steden) en met gewichten (= het aantal tussenstations) op de takken (= spoorwegen). Er wordt dus gevraagd naar een Hamiltonkring met minimaal totaalgewicht. Hieronder staat een voorbeeld van zo'n spoorwegnet met $n = 5$.



Gewichten in tabelvorm:

	a	b	c	d	e
a	-	3	1	2	7
b	3	-	4	6	2
c	1	4	-	6	1
d	2	6	6	-	8
e	7	2	1	8	-

De kring adebca is een Hamiltonkring met totaalgewicht $2 + 8 + 2 + 4 + 1 = 17$. Deze is echter *niet* optimaal.

a. (10 punten)
Leg uit hoe best-fit-first branch and bound werkt voor minimalisatieproblemen 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.
b. (5 punten)
Beschrijf een best-fit-first *branch and bound* algoritme dat ons handelsreizigersprobleem oplost. Leg uit hoe je in dit geval je deeloplossingen genereert en wat voor afschatting je gebruikt voor deeloplossingen en waarom die geldig is.
c. (10 punten)
Pas je methode toe op het voorbeeld en teken de bijbehorende state space tree. Geef daarin aan in welke volgorde de knopen bekeken worden en welke deeloplossingen gesnoeid worden en waarom.

Opgave 5.
a. (2 punten)
Wanneer is een binaire boom een heap (=hoopstructuur)?
b. (2 punten)
We hebben een heap die allemaal verschillende gehele getallen bevat. Waar bevindt zich de grootste waarde? Waar zit de kleinste waarde (meerdere mogelijkheden)?
b. (10 punten)
Teken de met de rij 49 38 57 45 75 66 42 80 69 22 53 corresponderende complete binaire boom en breng deze met behulp van *heapify* in hoopstructuur. Licht duidelijk toe wat er gebeurt (wat doe je en in welke volgorde) en laat tussenstappen zien. Geef het eindresultaat ten slotte ook weer als een array.