Tentamen Algoritmiek
Woensdag 7 augustus 2013, 14.00 – 17.00 uur

Geef een **duidelijke toelichting** bij al je antwoorden.

Puntenverdeling: 1: 23; 2: 20; 3: 24; 4: 23; 5: 10; **Veel succes !**

Opgave 1. We bekijken in deze opgave het volgende eenpersoonsspel. Gegeven een (aaneengesloten) rij met n stenen die aan de ene kant wit zijn en aan de andere kant zwart. Het aantal zwarte stenen (stenen met de zwarte kant boven) kan $0, 1, \ldots, n$ zijn, waarmee het aantal witte stenen dus n minus het aantal zwarte stenen is. De bedoeling is om de hele rij stenen weg te spelen via de volgende twee regels:

1. Wijs een zwarte steen aan en haal die weg.
2. Draai vervolgens de twee *directe* buren om, waardoor ze van kleur veranderen. Door het weghalen van stenen kunnen er gaten in de rij ontstaan. Als een zwarte steen maar één directe buur heeft, draai je die om; als er geen buren zijn hoeft er niets te worden omgedraaid.

Voorbeeld: Stel dat je begint met de rij `Z Z Z`. Een mogelijke serie zetten die de hele rij wegspeelt is de volgende: `Z Z Z` $\longrightarrow$ `- W Z` $\longrightarrow$ `- Z -` $\longrightarrow$ `- - -`. Merk op dat het niet mogelijk is om bijvoorbeeld `W Z Z` weg te spelen. Immers weghalen van de middelste steen levert `Z - W` en weghalen van de rechter steen geeft `W W -`. In beide gevallen zijn de witte stenen nooit meer weg te krijgen.

a. Wat zijn voor dit spel toestanden en acties (voor algemene n)? Hoe zien eindtoestanden eruit?
b. Teken de toestand-actie-ruimte voor het geval `W Z Z W Z` ($n = 5$). Geef duidelijk aan of de rij weggespeeld kan worden, en zo ja, hoe dit dan (bijvoorbeeld) zou kunnen.
c. Het blijkt (voor algemene n) dat de rij altijd weg te spelen is als je begint met een *oneven* aantal zwarte stenen (en de rest wit). Een werkende strategie is in dat geval de volgende: pak telkens van linksaf gezien de eerste de beste zwarte steen en haal die weg volgens de regels van het spel. Laat zien dat deze strategie altijd werkt als bij aanvang het aantal zwarte stenen *oneven* is (en de andere stenen dus wit zijn). *Hint:* Wat gebeurt er met het eventueel overblijvende linker gedeelte van de rij? En wat weet je van het rechter gedeelte?
d. Laat via een tegenvoorbeeld met $n \geq 6$, een even aantal (≥ 2) zwarte stenen en twee of meer witte stenen, zien dat de strategie uit **c.** niet (altijd) werkt als het aantal zwarte stenen bij aanvang even is.

Opgave 2. Gegeven een array A ($A[0], A[1], \ldots, A[n-1]$, met $n \geq 2$), dat n *verschillende* gehele getallen bevat. Gevraagd worden de index van de grootste waarde en de index van de kleinste waarde uit A. Neem aan dat n een 2-macht is.

a. Schrijf een eenvoudig brute force algoritme in C++ dat deze twee indices bepaalt. Er mag maar één for-loop gebruikt worden.
Hoeveel vergelijkingen doet je algoritme in het beste/slechtste geval? Licht je antwoord toe.

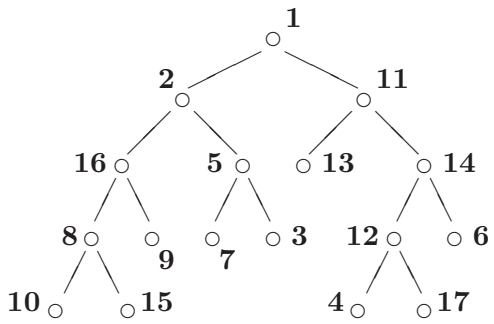
We gaan in **b.** en **c.** verdeel en heers gebruiken om het probleem op te lossen.

b. Geef een decrease by two algoritme in C++ voor het probleem. Er moet hiertoe een *recursieve* functie `void selectie2(int A[ ], int i, int & gr, int & kl)` worden geschreven die de index `gr` van de grootste en de index `kl` van de kleinste waarde oplevert uit het deelarray $A[0], \ldots, A[$`i-1`$]$, met `i` even.
c. Geef een divide-and-conquer algoritme in C++ voor het probleem. Het array dient hiervoor in twee gelijke delen te worden verdeeld. Schrijf hiertoe een *recursieve* C++-functie `void selectie3(int A[ ], int links, int rechts, int & gr, int & kl)` die het probleem oplost voor het deelarray $A[$`links`$], \ldots, $`A`$[$`rechts`$]$ ter lengte een 2-macht.

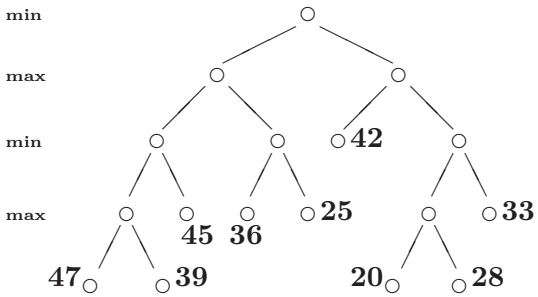
Opgave 3. Gegeven is een binaire boom met ingang (ofwel: wortel) `wortel`, die genummerde knopen bevat (`nummer`) en in elke knoop verder nog een veld `waarde`. Hierin is `wortel` dus een pointer naar een `knoop`, waarbij:

```
struct knoop {
   knoop* links;
   knoop* rechts;
   int nummer;
   int waarde;
} // knoop
```

Rechts een voorbeeldboom, waarbij de getallen bij de knopen de knoopnummers voorstellen.



a. We noemen een binaire boom *vol* als alle *interne* knopen twee kinderen hebben. De voorbeeldboom is een volle binaire boom.
Schrijf een recursieve C++-functie `bool vol(knoop* wortel)`, die bepaalt of de boom met ingang `wortel` vol is.
b. Voer een postorde-wandeling (LRW) uit voor de voorbeeldboom, waarbij je per bezochte knoop niet alleen het knoopnummer geeft, maar ook een bit (0 of 1 dus) dat aangeeft of de betreffende knoop een interne knoop (1) of een blad is (0). Geef de uitvoer als volgt: knoopnummer (bit), knoopnummer (bit), ...
c. Als we de knopen van een volle binaire boom in postorde-volgorde kennen, met per knoop een bit dat aangeeft of een knoop een interne knoop of een blad is, kunnen we eenduidig daaruit de oorspronkelijke boom reconstrueren. Geef nu de volle binaire boom die correspondeert met: $11(0), 2(0), 3(0), 10(0), 12(1), 8(1), 4(1), 5(0), 6(0), 1(0), 13(1), 7(1), 9(1)$
d. We gaan nu kijken naar de `waarde`-velden uit de boom. Veronderstel dat het `waarde`-veld in de bladeren reeds is ingevuld. We berekenen dan de waarde in de overige knopen door per niveau afwisselend het minimum danwel het maximum van de waardes van de kinderen te nemen. Op niveau 0 (de wortel) nemen we het minimum, op niveau 1 het maximum, etcetera.



Voor de voorbeeldboom hiernaast, met de bijbehorende `waarde`-velden voor de bladeren, levert dat voor de wortel de waarde 42 op.

Schrijf een recursieve C++-functie `void minmax(knoop* wortel, int niveau)` die voor een volle binaire boom met ingang `wortel` de `waarde`-velden op de aangegeven manier vult.

Opgave 4. Het Knapzakprobleem luidt als volgt. Gegeven een knapzak met capaciteit W en n objecten, genummerd 1 t/m n, met gewichten $w_1, w_2, \ldots, w_n$ en waardes $v_1, v_2, \ldots, v_n$. Gevraagd: de meest waardevolle deelverzameling der objecten die in de knapzak past (dus met totaalgewicht $\leq W$).

Voorbeeld:
$n = 4$ en het maximale gewicht (capaciteit) $W = 16$. De objecten zijn gesorteerd op v/w.

object	w	v	v/w
1	4	40	10
2	9	54	6
3	7	35	5
4	5	20	4

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, enzovoort.

Voor het knapzakprobleem genereren we, beginnend met de lege verzameling, deeloplossingen (hier deelverzamelingen) door in elke stap wel/niet het volgende object erbij te kiezen.

b. Uitgaande van een deeloplossing is een bovengrens voor de waarde van elke (complete) uitbreiding daarvan als volgt te berekenen:

1. Bij aanvang: $W * (v_1/w_1)$
2. Na de i-de stap: $v + (W - w) * (v_{i+1}/w_{i+1})$, met v de totaalwaarde van de reeds gekozen objecten en w het totaalgewicht daarvan

Leg uit waarom dit voor deeloplossingen inderdaad een bovengrens is voor de waarde die een uitbreiding van die deeloplossing kan hebben.
c. Pas het best-fit-first branch and bound algoritme met de bovengrens uit **b.** toe op het voorbeeld en teken de bijbehorende state space tree. Geef daarin ook aan in welke volgorde de knopen bekeken worden en welke deeloplossingen gesnoeid worden en waarom.

Opgave 5. Het algoritme van Dijkstra bepaalt voor gewogen grafen de (lengtes van) kortste paden vanuit een gegeven knoop naar alle andere knopen.
Pas het algoritme van Dijkstra toe op onderstaande graaf, beginnend in knoop 1. Geef voor elke stap van het algoritme duidelijk aan welke knoop erbij wordt gekozen in U (= verzameling knopen waarvan de kortste afstand vanaf 1 bekend is) en welke labels door die keuze veranderen en hoe.
Geef ook de uiteindelijke boom van kortste paden met daarin de bijbehorende lengtes van de kortste paden vanaf 1 !