ALGORITMIEK: antwoorden werkcollege 6

opgave 1.
Bereken eerst S, de som van de n getallen. Als S oneven is, dan kun je stoppen, want dan heeft het probleem zeker geen oplossing. Als S even is, genereer dan alle deelverzamelingen (dat zijn er maximaal 2^n) totdat een deelverzameling met som der elementen $\frac{S}{2}$ wordt gevonden, of totdat alle deelverzamelingen bekeken zijn. In het eerste geval zijn die deelverzameling en de verzameling van de resterende getallen de gevraagde opdeling. In het andere geval is er geen oplossing. Merk op dat je kunt volstaan met het genereren van alleen de deelverzamelingen met niet meer dan $\frac{n}{2}$ elementen.

opgave 2.
a. Laat s de som van de getallen in een rij zijn. In het magisch vierkant is de som der elementen van elke rij dan gelijk aan s (evenals de som der elementen van elke kolom, en van de twee hoofddiagonalen). Als we nu alle getallen van alle rijen (dus alle getallen uit het vierkant) optellen krijgen we:

$$s \times n = 1 + 2 + \cdots n^2 - 1 + n^2 = \frac{1}{2}n^2(n^2 + 1)$$

Hieruit volgt dat $s = \frac{1}{2}n(n^2 + 1)$.
b. Nummer de n^2 posities in een n bij n array van 1 t/m n^2. Genereer een permutatie van de getallen 1 t/m n^2 en zet ze in die volgorde in het array (op de plekken 1 t/m n^2). Controleer dan of aan de eisen voor een magisch vierkant is voldaan door rijsommen, kolomsommen en hoofddiagonaalsommen te bepalen en te kijken of die alle gelijk aan s (zie **a.**) zijn. Zo ja, dan is een magisch vierkant gevonden; zo nee, genereer de volgende permutatie.

opgave 3.
a. Er zijn n knopen; elke knoop kan met m mogelijke kleuren gekeurd worden; dus er zijn in principe maximaal m^n mogelijke kleuringen. Daarvan zullen er in het algemeen een heleboel niet aan de restrictie (buren hebben verschillende kleuren) voldoen. Echter voor de graaf die n knopen heeft en geen takken zijn al deze kleuringen goed. Overigens is het minimale aantal kleuren nodig om zo'n graaf te kleuren gelijk aan 1.
b. De complete graaf met n knopen bevat per definitie tussen elk tweetal knopen een tak: elke knoop is dus verbonden met alle $n - 1$ andere. Ergo: deze graaf heeft ten minste n kleuren nodig, en het kán ook met n: elke knoop een andere kleur. Merk op dat *elke* graaf met n knopen met n kleuren gekleurd kan worden.
c. Genereer alle m^n verschillende kleuringen van de graaf en controleer van elk daarvan of deze voldoet aan de restrictie: loop dus alle buurparen af en controleer of buren een andere kleur hebben. Het genereren van de kleuringen kan bijvoorbeeld stap voor stap gebeuren, zoals bij opgave 5. wordt voorgesteld.

opgave 4.
Genereer de magische vierkanten stap voor stap: vul de vakjes (i, j) van het vierkant (bijvoorbeeld rij voor rij en per rij van links naar rechts) met de getallen 1 t/m n^2 (een voor een). Controleer of de betreffende waarde niet al eerder (in een eerdere rij/kolom) voorkwam (vergelijk permutaties genereren van college). Zo ja, dan volgende getal proberen, zo nee dan controleren of de tot dusver verkregen rijsom (rij i)/kolomsom (kolom j) niet al te groot is (*). Zo ja, dan heeft het geen zin op verder te gaan, dus probeer op plek

(i, j) de volgende waarde. Zo nee, breid de deeloplossing dan op dezelfde manier verder uit. Als alle waarden op een plek geprobeerd zijn moet de waarde van de vorige positie worden herzien.
(*) Te groot betekent bijv. dat de huidige som al groter dan $\frac{n(n^2+1)}{2}$ is, maar kan ook wat subtieler. Je kunt, gegeven het aantal vakjes dat in de betreffende rij/kolom nog gevuld moet worden, een afschatting maken van hetgeen ten minste de rijsom/kolomsom zal worden. Overigens kun je ook op soortgelijke manier een afschatting geven van de maximaal te bereiken rijsom/kolomsom, en daaruit kun je mogelijk concluderen dat je de waarde $\frac{n(n^2+1)}{2}$ niet meer kan bereiken. Dat is ook een reden om niet verder te gaan met uitbreiden.

opgave 5.
a. Merk op: als $m \geq n$ is een kleuring met hooguit m kleuren altijd mogelijk (het kan nl. met n): geef elke knoop een andere kleur. Als $m < n$ moet je echt wat doen.
Kleur de knopen een voor een, *bijvoorbeeld* in de volgorde $1, 2, \ldots, n$. Probeer de aan de beurt zijnde knoop te kleuren met achtereenvolgens de kleuren 1 t/m m (*bijvoorbeeld*). Controleer of de buren van deze knoop allemaal een andere kleur hebben dan de kleur die je zojuist probeert. Zo ja, dan kan de knoop deze kleur krijgen en ga je verder op dezelfde manier met de volgende knoop. Zo nee, probeer dan de volgende knoop. Als je alle kleuren voor een bepaalde knoop geprobeerd hebt moet de kleur van de vorige knoop herzien worden.
b. Duidelijk is dat beide grafen met 3 kleuren gekleurd kunnen worden. Op de ene manier vindt je direct een goede kleuring, op de andere manier moet je eerdere keuzes herzien om tot een oplossing te komen.

opgave 6.
a. Het Latijns vierkant van orde 3 is:

```
1  2  3
2  3  1
3  1  2
```

Doe de rest zelf.
b. Recursief backtracking algoritme:

```
void latijnsvierkant (int n, int A[n][n], int i, int j) {
   int getal, k;
   bool okee;

   if ( i == n )
       drukaf (n, A);
       // i.pv. afdrukken zou je hier ook het aantal
       // Latijnse vierkanten kunnen tellen
   else {
       for ( getal = 1; getal <= n; getal++ ) {
       // controleer of getal al in rij i of kolom j staat
           okee = true;
           for ( k = 0; k < j; k++ ) {
               if ( A[i][k] == getal )
                  okee = false;
           } // for
```

```
            if ( okee )
                for ( k = 0; k < i; k++ ) {
                    if ( A[k][j] == getal )
                        okee = false;
                } // for
            if ( okee ) {
                A[i][j] = getal;
                if ( j < n-1 )
                    latijnsvierkant (n, A, i, j+1);
                else
                    latijnsvierkant (n, A, i+1, 1);
            } // if
        } // for
    } // else
} // einde
```

opgave 7.

```
void langste(int A[ ], int n, int vanaf, int deelrij[ ],
             int lengte, int & max){
   int i;
   for ( i=vanaf; i<n; i++ ){
       if ( deelrij[lengte] < A[i] ) { // consistent: nog steeds stijgend
           deelrij[lengte+1] = A[i];   // uitbreiden dus
           if ( lengte+1 > max )       // langste deelrij tot nu toe?
               max = lengte+1;
           langste(A, n, i+1, deelrij, lengte+1, max);
       }
   }
}
```

Aanroep: langste(A, n, 0, deelrij, 0);
De eerste aanroep gaat goed omdat `deelrij[0]=0`, de $A[i]$ groter dan 0 zijn en de echte deelrij pas op positie 1 begint.

opgave 8.
We gebruiken in main `int part[MAX]` voor het opslaan van de partities, met aantaldelen (het aantal delen waaruit de partitie bestaat) $<$ `MAX` We laten `part`[0] ongebruikt: de partitie begint in `part`[1].

```
   void partitie (int part[ ], int gedaan, int aantaldelen, int getal,
                  int vanaf) {
   // Maakt alle partities van getal in aantaldelen, elk deel >= vanaf.
   // Al gedaan: gedaan delen. Resultaat komt steeds in part.

     int j;
     if ( ( getal == 0 ) && ( aantaldelen == 0 ) )
        drukaf (part,gedaan); // nieuwe partitie gevonden
     else
        if ( ( getal != 0 ) && ( aantaldelen != 0 ) )
          for ( j = vanaf; j <= getal; j++ ) {
            // of j <= getal - (aantaldelen-1) * vanaf
             part[gedaan+1] = j;    // volgend deel wordt j
             partitie(part, gedaan+1, aantaldelen-1, getal-j, j)
          }  // for
   }  // partitie
```

Aanroep: `partitie (part,0,k,n,1);`
Voorbeeld: `partitie (part,0,3,17,5)` maakt alle partities van 17 in 3 delen waarbij elk deel ≥ 5 is, te weten $17 = 5 + 5 + 7 = 5 + 6 + 6$.