ALGORITMIEK: antwoorden werkcollege 5

opgave 1.
a. Brute force algoritme, direct afgeleid uit de observatie: loop v.l.n.r. door de tekst; als je een A tegenkomt op plek i ($0 \leq i < n-1$), loop dan van daaruit naar rechts en tel het aantal B's; deze aantallen voor alle A's sommeren levert het gevraagde aantal substrings. In C++:

```
int teller = 0;
for ( int i=0; i<n-1; i++ ) {
    if ( text[i] == 'A' ) {
        for ( int j=i+1; j<n; j++ ) {
            if ( text[j] == 'B' ) {
                teller++;
            } // if B
        } // for j
    } // if A
} // for i
// teller geeft nu het gevraagde aantal substrings
```

In de worst case (dat komt voor als `text` uit louter A's bestaat), is het aantal karaktervergelijkingen gelijk aan:

$$n + n - 1 + n - 2 + \cdots 3 + 2 = \frac{1}{2} n * (n+1) - 1,$$

dus kwadratisch.
b. We kunnen een slimmer algoritme schrijven, dat elk karakter uit de tekst maar één keer leest. We lopen nog staads van links naar rechts door de tekst. Merk op dat op het moment dat we een B tegenkomen, het aantal sustrings beginnend met een A en deze B als eindkarakter, precies het aantal A's is dat links van B staat, en die je dus al gelezen hebt. We houden derhalve een tellertje bij dat het aantal gelezen A's bijhoudt, en zodra we een B tegenkomen updaten we de totaalteller met het aantal tot dusver gelezen A's. In C++:

```
int totaalteller = 0;
int Ateller = 0;
for ( int i=0; i<n; i++ ) {
    if ( text[i] == 'A' )
        Ateller++;
    if ( text[i] == 'B' )
        totaalteller+=Ateller;
} // for i
// totaalteller geeft nu het gevraagde aantal substrings
```

Dit algoritme doet altijd $2n$ vergelijkingen, en is dus lineair.

opgave 2.
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.

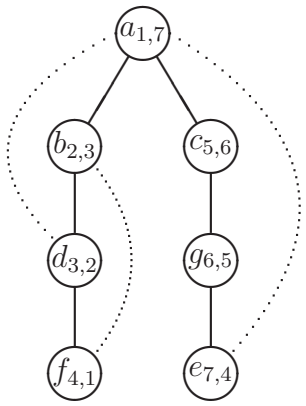
opgave 3.
a. Laat s de som van de getallen in een rij zijn. In het magisch vierkant is de som der elementen van elke rij dan gelijk aan s (evenals de som der elementen van elke kolom, en van de twee hoofddiagonalen). Als we nu alle getallen van alle rijen (dus alle getallen uit het vierkant) optellen krijgen we:

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

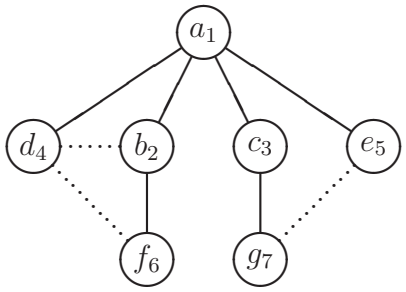
Hieruit volgt dat $s = \frac{1}{2}n(n^2 + 1)$.
b. Nummer de n^2 posities in een n bij n array van 1 t/m n^2. Genereer een permutatie van de getallen 1 t/m n^2 en zet ze in die volgorde in het array (op de plekken 1 t/m n^2). Controleer dan of aan de eisen voor een magisch vierkant is voldaan door rijsommen, 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 4.

a.



b.



Bij **a.** Het eerste subscript geeft de volgorde aan waarin de knopen voor de eerste keer worden bereikt, het tweede subscript de volgorde waarin de knopen helemaal zijn afgehandeld.
Bij **b.** Het subscript geeft de volgorde aan waarin de knopen worden bezocht.

opgave 5.
a.

```
bool ALGORITME DFSAcyclisch (G)
// Implementeert DFS wandeling door gegeven graaf,
// en controleert of deze acyclisch is
// Invoer: Ongerichte graaf G = (V,E)
// Uitvoer:
// * Graaf G met zijn knopen genummerd in de volgorde
//   waarin ze bij DFS wandeling voor het eerst worden ontdekt
```

```
// * true: als graaf acyclisch is
//   false: als graaf kringen bevat; in dat geval worden de knopen
//     in de gevonden kringen afgedrukt

{ for  elke knoop v in V  do
    mark[v] = 0;   // nog niet bezocht
  od
  teller = 0;
  acyclisch = true;
  for  elke knoop v in V  do
    if  mark[v] == 0  then
      dfs (v, null);  // v is wortel in DFS boom, dus heeft geen ouder
    fi
  od
  return acyclisch;
}

dfs (v, u)
// Bezoekt recursief alle nog onbezochte knopen die via een pad
// met v zijn verbonden, en nummert deze in de volgorde waarin
// ze worden ontdekt, met globale variabele 'teller'
// Wanneer een kring ontdekt wordt, drukt het de knopen in de kring af.
{ teller ++;
  mark[v] = teller;
  ouder[v] = u;  // ouder in DFS boom
  for  elke buurknoop w van v  do
    if  mark[w] == 0  then
      dfs (w, v);
    else  // back edge, dus een kring gevonden; druk knopen af
      acyclisch = false;
      printpad (w, v);
      print w;
    fi
  od
}

printpad (w, v)
// print recursief het pad van w naar v,
// zoals dat is opgeslagen in array ouder
{ if (w != v)  then
    printpad (w, ouder[v]);
  fi
  print (v);
}
```

b. Het is voldoende om
• in ALGORITME BFSAcyclisch de globale variabele 'acyclisch' te initialiseren en te retourneren, net als in DFSAcyclisch,
• en om in functie 'bfs' de globale variabele 'acyclisch' false te maken als (mark[w] != 0), bij de else van de if in de functie dus.

c. Nee, het is niet zo dat een van de twee algoritmes altijd minstens zo snel een kring ontdekt als de ander.
Bij de ene graaf wordt een kring eerder met DFS ontdekt, bijvoorbeeld als de kring diep in de eerste subboom van de wortel in de DFS boom zit.
Bij een andere graaf wordt een kring eerder met BFS ontdekt, bijvoorbeeld als de kring vlak bij de wortel in de laatste subboom van de wortel in de BFS boom zit.

opgave 6.

```
bool ALGORITME DFSSamenhangend (G)
// Implementeert DFS wandeling door gegeven graaf,
// en controleert of deze samenhangend is
// Invoer: Ongerichte graaf G = (V,E)
// Uitvoer:
// * Graaf G met zijn knopen genummerd in de volgorde
//   waarin ze bij DFS wandeling voor het eerst worden ontdekt
// * true: als graaf samenhangend is
//   false: als graaf niet samenhangend is

{ for  elke knoop v in V  do
    mark[v] = 0;   // nog niet bezocht
  od
  teller = 0;
  component = 0;
  for  elke knoop v in V  do
    if  mark[v] == 0  then
      component ++;
      dfs (v);
    fi
  od
  if  component == 1  then
    return true;
  else
    return false;
  fi
}

dfs (v)
// Bezoekt recursief alle nog onbezochte knopen die via een pad
// met v zijn verbonden, en nummert deze in de volgorde waarin
// ze worden ontdekt, met globale variabele 'teller'
{ teller ++;
  mark[v] = teller;
  compon[v] = component;
  for  elke buurknoop w van v  do
    if  mark[w] == 0  then
      dfs (w);
    fi
  od
}
```

opgave 7.

a. Wanneer je achtereenvolgens het vakje met S, de buren van het vakje met S, de buren van de buren van het vakje met S (voor zover nog niet bereikt), enz. in de queue stopt, en daarbij steeds de afstand 1 ophoogt, krijg je achtereenvolgens de volgende afstandtabellen:

Afstand

-1	-1	-1	-1	-1	-1
0	-1	-1	-1	-1	-1
⋮	⋮	⋮	-1	-1	-1
-1	-1	⋮	⋮	-1	-1
-1	-1	-1	⋮	-1	-1
-1	-1	-1	-1	-1	-1

Afstand

1	1	-1	-1	-1	-1
0	1	-1	-1	-1	-1
⋮	⋮	⋮	-1	-1	-1
-1	-1	⋮	⋮	-1	-1
-1	-1	-1	⋮	-1	-1
-1	-1	-1	-1	-1	-1

Afstand

1	1	2	-1	-1	-1
0	1	2	-1	-1	-1
⋮	⋮	⋮	-1	-1	-1
-1	-1	⋮	⋮	-1	-1
-1	-1	-1	⋮	-1	-1
-1	-1	-1	-1	-1	-1

Afstand

1	1	2	3	-1	-1
0	1	2	3	-1	-1
⋮	⋮	⋮	3	-1	-1
-1	-1	⋮	⋮	-1	-1
-1	-1	-1	⋮	-1	-1
-1	-1	-1	-1	-1	-1

Afstand

1	1	2	3	4	-1
0	1	2	3	4	-1
⋮	⋮	⋮	3	4	-1
-1	-1	⋮	⋮	4	-1
-1	-1	-1	⋮	-1	-1
-1	-1	-1	-1	-1	-1

Afstand

1	1	2	3	4	5
0	1	2	3	4	5
⋮	⋮	⋮	3	4	5
-1	-1	⋮	⋮	4	5
-1	-1	-1	⋮	5	5
-1	-1	-1	-1	-1	-1

Afstand

1	1	2	3	4	5
0	1	2	3	4	5
⋮	⋮	⋮	3	4	5
-1	-1	⋮	⋮	4	5
-1	-1	-1	⋮	5	5
-1	-1	-1	6	6	6

Afstand

1	1	2	3	4	5
0	1	2	3	4	5
⋮	⋮	⋮	3	4	5
-1	-1	⋮	⋮	4	5
-1	-1	7	⋮	5	5
-1	-1	7	6	6	6

Afstand

1	1	2	3	4	5
0	1	2	3	4	5
⋮	⋮	⋮	3	4	5
-1	8	⋮	⋮	4	5
-1	8	7	⋮	5	5
-1	8	7	6	6	6

Afstand

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

b. Je vindt de kortste paden van S naar D achterstevoren, door in de tabel met afstanden terug te lopen van D naar S, waarbij je iedere stap naar een aangrenzend vakje gaat met een lagere afstand vanaf S. Dus van D (met afstand 9) naar een aangrenzend vakje met afstand 8, naar een aangrenzend vakje met afstand 7, enzovoort. Er zijn tien verschillende kortste paden van S naar D.