ALGORITMIEK: opgaven werkcollege 5

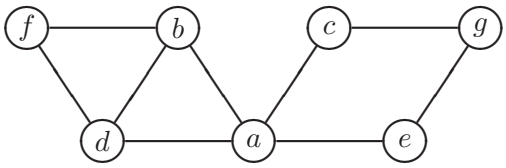
Brute force en exhaustive search

Opgave 1. (Levitin: opgave 3.2.8)
Formuleer bij **a.** en **b.** je algoritme in woorden of in C++.
Beschouw het probleem om in een gegeven tekst het aantal substrings te tellen dat begint met een `A` en eindigt met een `B`. (Bijvoorbeeld: er zijn vier van zulke substrings in `CABAAXBYA`.)
a. Ontwerp een brute-force algoritme voor dit probleem en bepaal zijn complexiteit.
Merk op dat het aantal gevraagde substrings beginnend met een `A` die op plek i ($0 \leq i < n-1$) in de tekst staat, precies gelijk is aan het aantal `B`'s in de tekst rechts van positie i. Baseer je brute-force algoritme op deze observatie.
b. Ontwerp een efficiënter algoritme voor dit probleem.

Opgave 2. (Levitin: opgave 3.4.6) Beschouw het **partition problem** (opsplitsingsprobleem): gegeven n positieve integers, splits ze op in twee disjuncte deelverzamelingen met elk dezelfde som van hun elementen. (Natuurlijk kent dit probleem niet altijd een oplossing.) Ontwerp een exhaustive-search algoritme voor dit probleem. Probeer het aantal deelverzamelingen dat het algoritme moet genereren te minimaliseren.
Er hoeft geen C++-code geschreven te worden. Formuleer je algoritme gewoon in woorden.

Opgave 3. (Levitin: opgave 3.4.10, a, b)
Magische vierkanten. Een magisch vierkant van orde n is een plaatsing van de getallen 1 tot en met n^2 in een $n \times n$ matrix, zodat elk getal precies één keer voorkomt, en elke rij, elke kolom en elke hoofddiagonaal (linksboven-rechtsonder en rechtsboven-linksonder) dezelfde som heeft.
a. Toon aan dat als er een magisch vierkant van orde n bestaat, de genoemde som gelijk moet zijn aan $n(n^2+1)/2$.
b. Ontwerp een exhaustive-search algoritme om alle magische vierkanten van orde n te genereren.
Er hoeft geen C++-code geschreven te worden: formuleer je algoritme in woorden.

Opgave 4. Beschouw de volgende ongerichte graaf G:



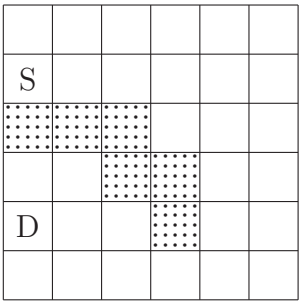
a. (Levitin, opgave 3.5.1.b)
Maak een DFS wandeling door G, startend in knoop a. Wanneer een knoop meerdere buren heeft, handel die dan in alfabetische volgorde af.
Construeer ook de bijbehorende DFS boom. Geef daarin aan in welke volgorde de knopen voor de eerste keer worden bereikt (en op de stapel gezet), en in welke volgorde ze helemaal zijn afgehandeld (van de stapel worden gehaald).

b. (Levitin, opgave 3.5.4)
Maak een BFS wandeling door G, startend in knoop a. Wanneer een knoop meerdere buren heeft, handel die dan in alfabetische volgorde af.
Construeer ook de bijbehorende BFS boom. Geef daarin aan in welke volgorde de knopen worden bezocht.

Opgave 5.
a. Pas de pseudo-code voor een DFS wandeling door een ongerichte graaf (zie slides van het hoorcollege, of Levitin paragraaf 3.5) zo aan, dat het algoritme controleert of de graaf acyclisch is. Als dat het geval is, moet het algoritme **true** teruggeven. Als dat niet het geval is (de graaf bevat dus minstens één kring), moet het algoritme **false** teruggeven en tevens de knopen van de gevonden kring afdrukken, in hun volgorde in de kring.
b. (naar Levitin, opgave 3.5.6.a)
Pas de pseudo-code voor een BFS wandeling door een ongerichte graaf (zie slides van het hoorcollege, of Levitin paragraaf 3.5) zo aan, dat het algoritme controleert of de graaf acyclisch is. Als dat het geval is, moet het algoritme **true** teruggeven. Als dat niet het geval is (de graaf bevat dus minstens één kring), moet het algoritme **false** teruggeven.
c. (Levitin, opgave 3.5.6.b)
We kunnen dus zowel DFS als BFS gebruiken om te ontdekken dat een ongerichte graaf kringen bevat. Is het zo dat een van de twee algoritmes hiermee altijd minstens zo snel klaar is als de ander?
Zo ja, welk van de twee is minstens zo snel als de andere, en waarom is dat zo? Zo nee, geef een voorbeeld van een graaf waarbij DFS sneller ontdekt dat er een kring is, en een voorbeeld van een graaf waarbij BFS sneller ontdekt dat er een kring is.

Opgave 6. Pas de pseudo-code voor een DFS wandeling door een ongerichte graaf (zie slides van het hoorcollege, of Levitin paragraaf 3.5) zo aan, dat het algoritme controleert of de graaf samenhangend is. Als dat zo is, moet het algoritme **true** teruggeven, en anders **false**. Tevens moet(en) de samenhangende component(en) van de graaf genummerd worden, te beginnen bij nummer 1, en elke knoop moet het nummer van zijn component krijgen.

Opgave 7. Beschouw het volgende 6x6 bord:



Op dit bord kun je in één stap van een vakje naar een aangrenzend vakje (horizontaal, verticaal, diagonaal) lopen. De gearceerde vakjes zijn verboden terrein.
a. Voer een Breadth First Search uit op dit bord, uitgaande van vakje S. Schrijf in elk (niet-gearceerd) vakje de resulterende afstand vanaf S.
b. Bepaal, uitgaande van je antwoord bij **a.**, alle kortste paden van S naar D. Hoeveel verschillende kortste paden van S naar D zijn er?