ALGORITMIEK: opgaven werkcollege 5

Exhaustive search en backtracking

Opgave 1. Uit Levitin: opgave 3.4.6. Er hoeft geen C++-code geschreven te worden. Formuleer je algoritme gewoon in woorden.

Opgave 2. Uit Levitin: opgave 3.4.9. a, b. Bij 3.4.9.b. hoeft geen C++-code geschreven te worden: formuleer je algoritme gewoon in woorden.

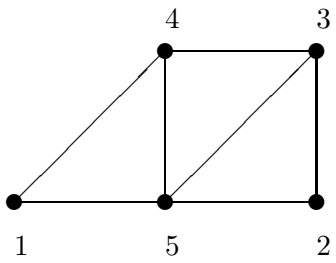
Opgave 3. Graafkleuringsprobleem: gegeven een ongerichte graaf met n knopen en een positief geheel getal m. Kunnen de knopen van de graaf zodanig gekleurd worden met hooguit m kleuren, dat geen tweetal aangrenzende knopen dezelfde kleur heeft? Zo ja, geef zo'n kleuring.
a. Hoeveel verschillende kleuringen zijn er maximaal mogelijk met m kleuren? Zijn er grafen die op zoveel manieren gekleurd kunnen worden?
b. Wat is het minimale aantal kleuren dat nodig is om een complete graaf met n knopen te kleuren?
c. Geef (in woorden) een exhaustive search algoritme dat bepaalt of de knopen van een gegeven graaf met hooguit m kleuren kunnen worden gekleurd, en dat zo ja, ook zo'n kleuring oplevert.

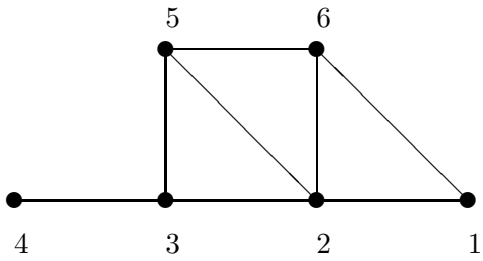
Opgave 4. Geef (in woorden) een backtracking algoritme voor het genereren van magische vierkanten. Geef duidelijk aan hoe je magische vierkanten stap voor stap opbouwt, en welke restrictie je in elke stap controleert.

Opgave 5.
a. Geef (in woorden) een backtracking algoritme voor het graafkleuringsprobleem.
b. Veronderstel, dat we bij het genereren van een graafkleuring elke knoop steeds eerst met kleur 1, dan met kleur 2, etc. proberen te kleuren. Er kan dan nog gekozen worden in welke volgorde de knopen worden doorlopen (en gekleurd). Deze volgorde heeft invloed op het aantal stappen dat het kost om een oplossing te vinden. We bekijken twee gevallen:

1. de knopen worden in de volgorde 1 t/m n doorlopen
2. vanuit de knoop die we nu kleuren kiezen we als volgende knoop de eerste buur (d.w.z. die met het laagste knoopnummer) die we nog niet gehad hebben. Van daaruit op dezelfde manier verder naar de volgende knoop. Begin vanuit knoop 1.

Pas beide methoden toe op onderstaande grafen met $m = 3$ en bekijk het verschil.





Opgave 6. Een *Latijns vierkant* van orde n is een n bij n vierkant (matrix) die in elke rij en elke kolom de getallen 1 t/m n bevat. Er geldt dus dat in elke rij en in elke kolom elk van die getallen slechts één keer voorkomt. De bedoeling is nu om alle Latijnse vierkanten te genereren: zonder beperking der algemeenheid mogen we aannemen dat in de eerste rij en de eerste kolom de getallen 1 t/m n in die volgorde reeds vaststaan.

a. Een voor de hand liggende manier om een Latijns vierkant stap voor stap op te bouwen is om het vierkant rij voor rij, en per rij van links naar rechts te vullen. In elke stap wordt dan gecontroleerd of aan de restricties is voldaan. Gebruik dit backtracking algoritme om (met de hand) alle Latijnse vierkanten van orde 3 (dit is er maar 1) en van orde 4 (dit zijn er 4) te genereren (waarbij de eerste rij en kolom reeds als hierboven vastliggen).
b. Schrijf nu een recursieve C++-functie `void latijnsvierkant(n, A, i, j)` die alle Latijnse vierkanten van orde n m.b.v. backtracking genereert en afdrukt. We nemen weer aan dat de eerste rij en de eerste kolom vastliggen. De i en j geven aan dat je nu het vakje met coördinaten (i, j) probeert te vullen, waarbij alle eerdere vakjes reeds goed gevuld zijn. Neem aan dat een functie `void drukaf(n, A)`, die een n bij n matrix afdrukt, reeds bestaat.

Opgave 7. Gegeven een rij van n positieve (> 0) gehele getallen, opgeslagen in een array `A`. Gevraagd: de lengte van het/een langste stijgende deelrijtje. (Zie ook college.) Een voorbeeld: twee stijgende deelrijen van de rij $R = 7, 4, 3, 2, 7, 5, 3, 6, 4, 5$ zijn bijvoorbeeld $4, 5, 6$ en $2, 3, 4, 5$. Deze laatste is de langst mogelijke stijgende deelrij. De rij $5, 6, 7$ is geen deelrij van R.
Een backtracking algoritme bouwt deelrijtjes bijvoorbeeld stap voor stap op door van links naar rechts telkens een nieuw element van `A` toe te voegen aan het reeds opgebouwde stijgende deelrijtje. De toe te voegen elementen zijn die waarden die in A allemaal rechts van de als laatste toegevoegde waarde staan. Voorbeeld: de deelrij $4, 5, 6$ van R kan worden uitgebreid met 4 resp. 5 tot een nieuwe deelrij van R. In elke stap wordt dan gecontroleerd of de tot dusver geconstrueerde stijgende deelrij met de toe te voegen waarde nog wel stijgend is. Zo ja, dan wordt de waarde toegevoegd en wordt het deelrijtje op dezelfde manier verder uitgebreid. Zo nee, dan wordt de volgende waarde uit `A` geprobeerd. Schrijf een recursieve C++-functie `void langste(A, n, vanaf, deelrij, lengte, max)` die de lengte van zo'n langste stijgende deelrij oplevert. Hierin is de tot dusver opgebouwde stijgende deelrij opgeslagen in `deelrij[1]` t/m `deelrij[lengte]`, en ga je die uitbreiden met de elementen uit `A[vanaf]` t/m `A[n-1]`. Laat `deelrij[0] = 0`. Verder geeft `max` de lengte van de tot dusver gevonden langste deelrij aan.

Opgave 8. Een partitie van het natuurlijke getal n in k delen is een rijtje gehele getallen $0 < n_1 \leq n_2 \leq \cdots \leq n_k$ met $n_1 = n_2 + \cdots + n_k = n$. Schrijf een recursieve C++-functie die met behulp van backtracking alle partities van n in k delen genereert en afdrukt.