ALGORITMIEK: proeftentamen, mei 2007

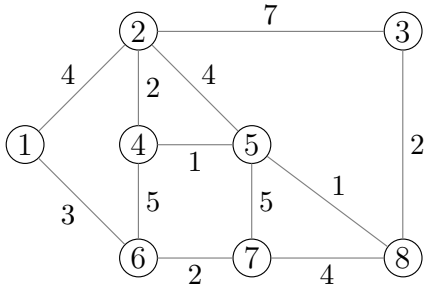
Opgave 1. We bekijken het volgende tweepersoonsspel, dat wordt gespeeld door Hans en Grietje. Voor aanvang van het spel worden n ($n \geq 3$, geheel) pionnen op een rij gezet, waarna de tweede van links wordt verwijderd. Voor $n = 8$ is de beginsituatie dus p - p p p p p p . Hierin geeft p een pion aan.
De spelers doen om de beurt een zet en Hans begint. Een zet is in dit spel het wegnemen van één pion, of het wegnemen van twee direct naast elkaar gelegen pionnen uit de rij. *Echter* in de eerste zet van het spel mag slechts één pion worden weggenomen. Ter illustratie: de stand - - - p p p p - is *niet* in één zet uit p - p p p p p - te verkrijgen, maar p - - - p p p - *wel*.
Degene die de laatste pion weghaalt is de winnaar.

a. Wat zijn voor dit spel toestanden en acties (voor algemene n)?
b. Bepaal het totaal aantal mogelijke toestanden voor algemene n en leg je antwoord uit.
c. Teken de toestand-actie-ruimte voor $n = 5$. Toestanden die in één zet gewonnen kunnen worden (zoals - - - p p) hoef je niet verder uit te werken.
d. Geef bij elke toestand aan of deze winnend is voor de speler die aan de beurt is. (Dat wil zeggen: kan de speler die aan de beurt is altijd winnen, ongeacht wat de tegenstander doet?) Is het spel voor $n = 5$ winnend voor de beginnende speler (hier dus Hans)?

Opgave 2. Gegeven een array met n elementen. We willen verdeel en heers gebruiken om de index van het grootste array-element te vinden.
a. Geef een divide-and-conquer algoritme (in pseudocode of C++) voor het vinden van deze index. Hierbij moet het array in twee ongeveer gelijke delen worden verdeeld.
b. Geef voor $n = 2^k$ de recurrente betrekking waaraan $W(n)$ (= het aantal vergelijkingen dat het algoritme uit **a.** doet) voldoet en laat zien dat $W(n) = n - 1$ door invullen in de recurrente betrekking.
c. Geef ook een decrease-by-two algoritme (in pseudocode of C++) voor het bepalen van deze index.

Opgave 3. Het algoritme van Dijkstra berekent de (lengtes van) kortste paden vanuit een gegeven knoop naar alle andere knopen.

a. Illustreer de werking van het algoritme van Dijkstra (zie volgende pagina) door het algoritme toe te passen op onderstaande graaf, beginnend in knoop 1. Geef voor elke stap van het algoritme duidelijk de labels (`pad`$[v]$) van de knopen aan, en welke knoop v^* erbij wordt gekozen in U.



// invoer: samenhangende gewogen graaf $G = (V, E)$ en startknoop s
// uitvoer: array `pad` dat de lengtes van de kortste paden vanuit s bevat;
// na afloop is `pad`$[v]$ = de lengte van een kortste pad van s naar v
// bij aanvang is `pad`[s] $= 0$; `pad`$[v] = \infty$ voor de andere knopen
$U := \emptyset$; // U bevat de knopen waarvoor de kortste afstand vanuit s bekend is
while ($U \neq V$) **do**
    vind knoop $v^* \in V \setminus U$ met pad$[v^*]$ minimaal;
    $U := U \cup \{v^*\}$;
    for alle knopen v aangrenzend aan v^* **do**
        if pad$[v^*]$+ gewicht(v^*, v) < pad$[v]$ **then**
            pad$[v]$:= pad$[v^*]$+ gewicht(v^*, v);
        fi
    od
od

b. Om te onderzoeken of het algoritme van Dijkstra ook werkt voor grafen met negatieve gewichten veranderen we het gewicht van de tak tussen 3 en 8 in -5. Vindt het algoritme de kortste afstand tussen 1 en 8? En tussen 1 en 3? Motiveer je antwoord.

Opgave 4. Bekijk het partitie-probleem: gegeven n verschillende gehele getallen > 0, verdeel ze —indien mogelijk— in twee disjuncte deelverzamelingen waarvan de som der elementen even groot is. Voorbeeld: de verzameling $\{6, 8, 4, 5, 1\}$ is op te splitsen in $\{6, 5, 1\}$ en $\{8, 4\}$, beide met som 12.
a. Geef in woorden een exhaustive search algoritme voor dit probleem.
b. Geef in woorden of in pseudocode een backtracking algoritme voor dit probleem. Hoe worden (deel)oplossingen stap voor stap gegenereerd? Wat is de restrictie waaraan deeloplossingen moeten voldoen?
c. Illustreer de werking van je backtracking algoritme door het toe te passen op het bovengegeven voorbeeld, en teken de bijbehorende state space tree.

Opgave 5. **a.** Wanneer is een binaire boom een heap (=hoopstructuur)?

We willen een recursieve C++-functie `bool heap(knoop* wortel)` schrijven, die van een gegeven binaire boom met ingang wortel bepaalt of deze aan de heapeigenschap voldoet.

b. Geef een recursieve formulering die aangeeft of een binaire boom aan de heapeigenschap voldoet (dus uitdrukken in (onder andere) de subbomen van de wortel).
c. Schrijf nu de bovengenoemde C++-functie `bool heap(knoop* wortel)`. Hierin is `wortel` een pointer naar een `knoop`, waarbij

```
struct knoop {
   knoop* links;
   knoop* rechts;
   int info;
} // knoop
```

d. Teken de met de rij 44 77 33 50 24 16 61 68 85 40 corresponderende complete binaire boom en breng deze m.b.v. *heapify* in hoopstructuur. Laat tussenstappen zien.
e. Leg uit hoe het sorteeralgoritme heapsort werkt. Voer ter illustratie de eerste drie stappen van heapsort uit op de rij uit **d.**