# Tweede deeltoets Algoritmiek (INFOAL)

## 11-11-2010

Uitwerkingen

(De opmaak is over de jaren verloren gegaan maar de antwoorden zijn nog correct en gecontroleerd door de docent)

**Opgave 1, Paden in een grichte graaf.**
Noem het aantal knopen n, en aantal kanten m.
**(a)** Complexiteit BFS: n+m
DFS: n+m
Dijk: (n+m).lgn (Implementatie met heap)

**(b)** Bij een MST algoritme zijn kanten altijd ongericht, en er wordt van samenhang van de graaf uitgegaan. Je kunt dus een pad vinden, waar je tegen sommige edges in moet lopen en dat is niet toegestaan. Ook is niet duidelijk, hoe een MST algoritme "signaleert" dat er geen pad is.

Fout antwoord: MST geeft niet altijd kortste paden.
Er wordt namelijk helemaal niet naar een kortst pad gevraagd.

**(c)** Als elk pad opgeleverd kan worden, kan het aantal ronden van FF erg hoog zijn (afhankelijk van getallen in de invoer, niet begrensd in n en m). Edmonds en karp hebben aangetoond dat als je BFS gebruikt, er hoogstens n.m ronden nodig zijn.

Fout antwoord: BFS is veel sneller dan DFS; zie namelijk a.

**(d)** Je kunt elk van de drie zoekalgoritmen gebruiken, maar een kleine aanpassing maken die de rode kanten negeert.

**(e)** Doe een BFS (of DFS) vanuit s, en markeer alle knopen die je bezoekt (marker A). Doe ook een BFS vanuit t in de omgekeerde graaf (markeer met B). Check of er een edge is met beginpunt A en eindpunt B.

Slecht antwoord 1: verwijder de rode kanten, dan voor elke rode kant {voeg 'm toe en doe BFS}. Dit kost je namelijk voor elke rode een BFS!
Levert slechts 3 (van de 4) punten.

Slecht antwoord 2: Doe BFS/DFS, en houd bij of je al een rode bent overgestoken. Als dat zo is, zoek je alleen over witte verder.
Het kan gebeuren dat je een knoop eerst via een rode kant vindt, en later over een pad zonder rode. Als je daar niets over zegt, hoogstens 2pt.

**(f)** De twee BFS-en kosten n+m, het omkeren van de graaf ook, en het bekijken van de edges kan ook in die tijd.
Totaal O(n+m).

Slecht antwoord: Ja, zie d, voor mensen die bij d slecht antwoord 2 gaven. Dan slechts 1 vd 2 punten.

**(g)** Maak van de graaf een gewogen graaf waarin de rode kanten een erg hoog gewicht hebben. Het moet zo hoog zijn, dat de kosten van 1 rode edge hoger zijn dan elk simpele pad van niet-rode edges, dus neem gewicht n (en 1 voor de andere kanten). Nu kun je zoeken met Dijkstra.

Het probleem is ook in O(n+m) tijd op te lossen met "multi-source BFS". Begin ronde 0 met s in een queue te smijten.
Ronde i begint met de queue gevuld met knopen die via i rode kanten bereikbaar zijn, waarbij de rode kant de laatste in het pad is.
In ronde i doe je BFS, maar je negeert rode kanten.
Als de queue leeg is, zijn ALLE knopen gemarkeerd die te bereiken zijn via i rode kanten.
De ronde eindigt door nogmaals alle knopen langs te gaan die in deze ronde zijn bezocht, maar nu kijk je naar hun rode kanten en zet de eindpunten in de queue, waarna je klaar bent om ronde i+1 te doen.

**Opgave 2, randomiserende algoritmen.**

**(a)** Een Monte Carlo algoritme termineert in elke executie, bij elke invoer is er een kans p dat het antwoord correct is.

**(b)** Een deterministisch algoritme bekijkt minstens de helft plus 1 van de getallen. Het kan namelijk pech hebben en allemaal even getallen zien, en pas als je er helft plus 1 hebt gezien weet je zeker dat het type A is.

En jongens en meisjes, denk even na!!! Is het genoeg om ceil(n/2) elementen te bekijken? (HINT: Nee.)

**(c)** Bij een Las Vegas mag wel de terminatie van random trekkingen afhangen, maar de correctheid van het antwoord niet. Dat betekent dat antwoord A pas gegeven kan worden als de helft plus 1 is bekeken.
Het beste algoritme doet (helft plus 1) random trekkingen zonder teruglegging. Oneven: output B. Even: doorgaan.
Als alle trekkingen even waren: output A.

Het antwoord is altijd correct.
De tijd is n/2+1 voor een type-A rij en verwacht O(1) voor een type-B rij.
In feite is terminatie van dit algoritme ook deterministisch zodat het algoritme ook aan de definitie van Sherwood voldoet.

Fout antwoord: trek door tot je een oneven ziet.
Dit algoritme termineert NOOIT bij een type-A. En let op, de terminatiekans wordt altijd WORST CASE over de invoeren genomen. Deze oplossing heeft dus terminatiekans 0. Geen punten!

**(d)** Trek 7 getallen. Allemaal even: output A.
Er is een oneven: output B.

Het is hier natuurlijk dodelijk om 0,99*(n/2) getallen te gaan bekijken. Even rekenen met kansformules uit Wiskundige Technieken!

**(e)** Rekentijd: 7 (constant).
Als de rij van type A is, worden er alleen even getallen getrokken, het resultaat is dan altijd A, dus correct.
Als de rij van type B is, heeft elke trekking een kans van <= 1/2 om even te zijn. De kans dat ALLE trekkingen even zijn is dan (1/2)^7, dus 1/128 wat kleiner is dan 1%

**(f)** Het is een B-test. Immers, de uitkomst "B" is altijd correct, terwijl de uitkomst "A" kan komen als er wel naar een oneven getal is "gezocht", maar het algoritme dit niet heeft gezien.

**(g)** Nee, sortering maakt in dit probleem geen bal uit, omdat ELK patroon van even/oneven getallen ook kan voorkomen in een gesorteerde rij.

Matig antwoord: Nee je hebt er niks aan want het algoritme kijkt op random plekken.
De vraag is namelijk NIET:
- of je gegeven algoritme profiteert van sortering
maar WEL
- of er een BETER ALGORITME mogelijk is als er van sortering gebruik mag worden gemaakt.

Briljante gedachtengang van Koen: dit kan wel als d=S[n]-S[1] klein is. De even/oneven getallen komen dan, door sortering, voor in "runs" waarbij minstens 1 oneven run lengte n/d moet hebben om aan de helft te komen. Je kunt dus alleen op posities veelvoud van n/d kijken.

**Opgave 3, Complexiteit.**

**(a)** Vaststellen dat er geen snel algoritme bestaat is niet zo gemakkelijk. Zo'n optimaliseringsvraagstuk is meestal in NP: dat wil zeggen, dat je van een gegeven oplossing kunt verifieren dat hij een bepaalde kwaliteit haalt. Je zult je dan als doel stellen: bewijzen dat het NP-compleet is. Om ZEKER te weten dat er geen algoritme bestaat, moet je dan ook nog "even" bewijzen dat NP /= P, maar dat is waarschijnlijk buiten je bereik.

Om te bewijzen dat het NPC is moet je een BEKEND NPC probleem nemen (bv TSP, SubsetSum oid) en dat reduceren tot jullie probleem. Dus: geef een algoritme dat een (bv.) TSP-instantie afbeeldt op een instantie van jullie probleem (met behoud van

oplossing).

**(b)** Dat het probleem NPC is, sluit niet uit dat er bruikbare algoritmen bestaan, alleen dat zo'n algoritme snel kan werken bij grote invoeren.

1. Ga na dat er in jullie project alleen dermate kleine instanties voorkomen, dat ze voldoende snel kunnen.
2. Zoek naar een exponentieel algoritme, maar met een zo klein mogelijk grondtal: een 1,02^n algoritme is beter dan 3^n.
3. Gebruik een approximatie (of heuristiek).