

## Logisch Programmeren en Zoektechnieken

Tweede deelttoets. 24 december 2010. 9:00-12:00u

**Opg. 1** Twee belangrijke sorteer-algoritmen voor lijsten zijn 'bubblesort' en 'quicksort'.

(a) Geef voor beide algoritmen een zo nauwkeurig mogelijke beschrijving. (een implementatie hoeft niet, mag wel).

(b) Zoals de namen al suggereren verdient 'quicksort' een zekere voorkeur. I.h.a. is deze namelijk efficiënter dan 'bubblesort'. Leg uit waarom.

**Opg. 2** Hieronder staat een spel-situatie afgebeeld van het razend populaire 'boter-kaas-en-eieren' ('tic-tac-toe').

		x
	o	

Laten we dit soort situaties een eenvoudige Prolog lijst-representatie geven:

```
[leeg, leeg, x, leeg, leeg, leeg, leeg, o, leeg]
```

Stel dat we voor dit soort representaties van toestanden ook een correct `goal/2` predicaat geïmplementeerd hebben dat voor elk van de twee spelers bepaalt of een gegeven toestand een overwinning vertegenwoordigt.

(a) Schrijf een `move/3` predicaat in de vorm

```
move(+Speler, +Toestand, -VolgendeToestand)
```

zodanig dat `+Speler` (x of o) de speler is die aan de beurt is, `+Toestand` de huidige spel-toestand is en `-VolgendeToestand` een toestand is die door een legale zet van de beurt-speler ontstaat.

(b) Vul je programma verder aan met een predicaat `solve/4` in de vorm

```
solve(+Speler, +BeurtSpeler, +Toestand, -Strategie).
```

Dit predicaat bepaalt, indien mogelijk, een winnende strategie (`-Strategie`) voor een gegeven speler (`+Speler`). Hierbij is `+BeurtSpeler` de speler die aan de beurt is, `+Toestand` de huidige toestand en `-Strategie` de 'oplossing-boom' ('solution tree').<sup>1</sup>

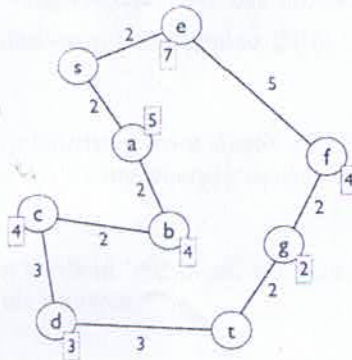
**Opg. 3** De belangrijkste twee zoek-technieken waarbij geen gebruik gemaakt wordt van kosten-functies zijn 'depth-first search' en 'breadth-first search'.

(a) De eerstgenoemde wordt vaker gebruikt dan de laatstgenoemde op grond van overwegingen van efficiëntie. Wat is in dit opzicht het voordeel van 'depth-first search' t.o.v. 'breadth-first search'?

(b) Wat zijn de voordelen van 'breadth-first search' t.o.v. 'depth-first search'?

**Opg. 4** Aan de ommezijde is een plaatje gegeven van mogelijke routes van een begintoestand `s` naar een eindtoestand `t`. De verbindingen geven de mogelijke toestands-overgangen aan, en de cijfers bij die verbindingen de kosten van de desbetreffende overgang. De omkaderde cijfers bij de verschillende tussentoestanden zijn schattingen van de nog te maken kosten vanuit die toestand tot de eindtoestand `t`. Dit is de zogenaamde *heuristische kosten-functie*.

<sup>1</sup>Hint: `solve/4` werkt hetzelfde als `solve/3` voor and/or-search als `+Speler = +BeurtSpeler` en als `solve_all/3` voor and/or-search indien dit laatste niet het geval is.



(a) Leg aan de hand van dit voorbeeld uit hoe het A\*-algoritme stap voor stap het kortste pad van s naar t bepaalt.

(b) In het hierboven gegeven voorbeeld is de heuristische functie steeds optimistisch over de nog te maken kosten om tot de gewenste eindtoestand te komen. D.w.z. dat de heuristische functie steeds lager is dan de werkelijke kosten tot de eindtoestand. Voor het succesvol toepassen van het A\*-algoritme is dat i.h.a. van belang. In welk opzicht? Leg uit waarom.

Opg. 5 Het logische inferentie-systeem dat Prolog aandrijft bestaat uit resolutie en unificatie.

(a) Hoe luidt de propositie-logische versie van de resolutie-regel?

(b) Laat op grond van resolutie zien dat  $\neg p \wedge r$  een geldige conclusie is op grond van de aannames  $p \vee q \vee r$ ,  $\neg p \vee \neg q \vee \neg r$ ,  $\neg p \vee q$  en  $\neg q \vee r$ .

Opg. 6 Veronderstel dat we een volledige afstandentabel voor een aantal steden in Nederland in een Prolog-database hebben opgeslagen met behulp van een predicaat afstand/3:

```
stad(amsterdam).
stad(denhaag).
...
afstand(amsterdam, utrecht, 40).
afstand(denhaag, rotterdam, 30).
afstand(utrecht, maastricht, 170).
...
```

Voor elk tweetal steden uit het bestand is op deze wijze hun onderlinge afstand gerepresenteerd.

(a) Schrijf een korte Prolog-implementatie van een predicaat route/3 wat een zo kort mogelijke route langs alle voorkomende steden bepaalt zodanig dat in alle steden precies één keer gestopt wordt. We spreken af dat we het predicaat implementeren volgens het volgende patroon:

```
route(+Start, -Lijst, -Afstand)
```

waarbij +Start de plaats van vertrek aanduidt, -Lijst de lijst van steden in de volgorde van bezoek en -Afstand de totale afstand die afgelegd is.

(b) Geef een zo nauwkeurig mogelijke schatting van de tijds-complexiteit van je implementatie.