Rapport practicum 2

Gegevensstructuren & Algoritmen

Samuel Debruyn (r0305472)

5-4-2013

Inhoud

1. Implementatie van de klasse Board

De klasse board heb ik uitgebreid om de toestanden in het spel bij te houden. Niet alleen worden de grootte van het bord en 2-dimensionale array met de tegels bijgehouden, maar ook het aantal stappen om tot deze configuratie te komen en previous, het vorige knooppunt in de graaf. Deze laatste twee zijn bij het initiële bord niet gedefinieerd, anders is het huidige bord een resultaat uit neighbors() van het bord dat in previous opgeslagen is. Moves is dan eentje meer dan het aantal moves bij het bord uit previous.

Daarnaast heb ik hashCode() geïmplementeerd om met een HashSet te kunnen werken in de klasse Solver (‘gesloten’ lijst met configuraties die al gecontroleerd zijn). Deze hashCode() werkt met Arrays.deepHashcode() van de 2-dimensionale array die de tegels van de bord voorstelt. Omdat de berekening hiervan heel wat rekenkracht vergt wordt deze na de eerste berekening bijgehouden in het object zelf. Bij de volgende keer dat hashCode() aangeroepen wordt, wordt de hashcode niet opnieuw berekend.

Bij equals() heb ik gebruik gemaakt van de methode deepEquals() uit Arrays voor de 2-dimensionale array die de tegels van het bord bijhoudt.

Ook heb ik enkele hulpmethodes aangemaakt. Zo slaat de constructor de grootte (N) van het bord op als size. De methode getEmpty() geeft de indices van het lege vakje in het huidige bord terug en de methode getDestination() wordt in manhattan() gebruikt om voor een getal de indices van de doelpositie te vinden.

In de methode neighbors() maak ik vier keer gebruik van try & catch. Ik probeer telkens het lege vakje te verplaatsen naar een positie boven, onder, links of rechts van de huidige positie. Indien hierbij een ArrayIndexOutOfBoundsException gegooid wordt dan bevindt het vakje zich op de rand van het bord en gaat de methode verder met de volgende zijde. De borden met verplaatsingen die wel lukken worden in een ArrayList bijgehouden.

Om in neighbors() en isSolvable() gemakkelijk een kopie van het huidige bord te maken heb ik een copy constructor gemaakt. Deze constructor neemt als argument een bord aan en genereert een kopie van dit bord.

2. Toestanden in het spel

Alle gegevens die een toestand in het spel bepalen (aantal verplaatsingen tot huidige configuratie, de vorige configuratie en het huidige bord) worden in de klasse Board bijgehouden zoals uitgelegd in de eerste alinea van vraag 1.

Het vorige bord wordt bijhouden in previous, het aantal verplaatsingen in moves en de eigenschappen van het huidige bord zijn size en tiles[][]. Deze laatste twee variabelen zijn final en worden in de constructor gezet. Voor de andere twee bestaat er een setter. Voor de vier variabelen bestaat er een getter.

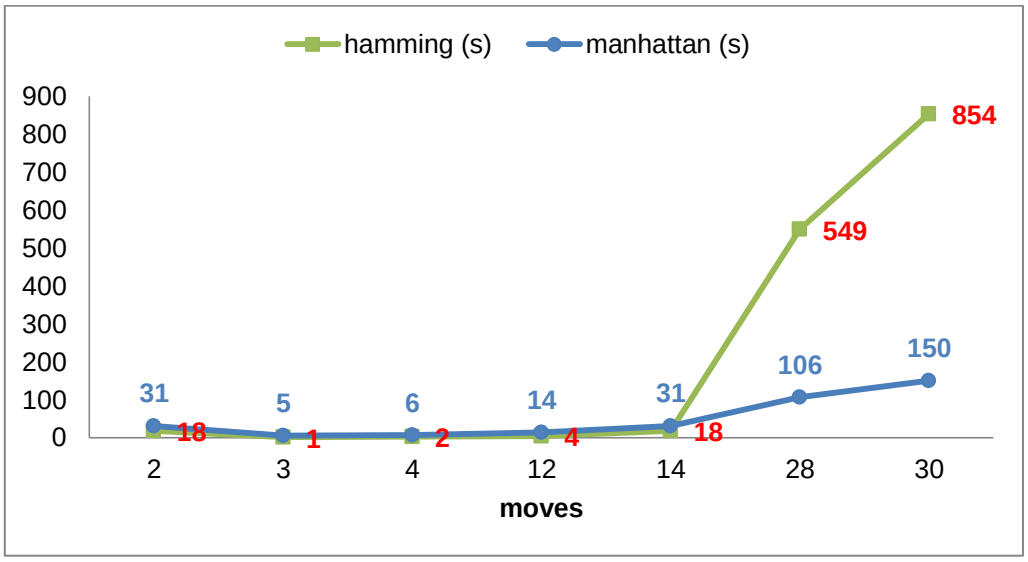
3. Uitvoeringstijden puzzels

Gemeten op een Asus X53Sv met de klasse Measure.

Puzzel	Verplaatsingen	Hamming (s)	Manhattan (s)
puzzle28.txt	28	0	0
puzzle30.txt	30	0	0
puzzle32.txt	32	/	1
puzzle34.txt	34	/	0
puzzle36.txt	36	/	2
puzzle38.txt	38	/	2
puzzle40.txt	40	/	1
puzzle42.txt	42	/	9

De puzzels na puzzle30.txt konden niet gevonden worden met de Hamming prioriteitsfunctie. De uitvoering duurde telkens langer dan vijf minuten en lukte niet door een geheugentekort.

4. Willekeurige 4x4 of 5x5 puzzels



We zien op bovenstaande grafiek dat de hamming prioriteitsfunctie efficiënter is voor puzzels waarbij minder moves nodig zijn om de puzzel op te lossen. Vanaf de puzzel met 28 moves klopt dit echter niet meer en is de manhattan prioriteitsfunctie efficiënter. Met de hamming prioriteitsfunctie lukt het zelfs niet meer om de puzzels op te lossen waarbij meer 32 moves of meer nodig zijn.

Daarnaast merkte ik dat de prioriteitswachtrij telkens veel groter werd bij de hamming prioriteitsfunctie dan bij de manhattan prioriteitsfunctie. Echter vereiste de manhattan prioriteitsfunctie meer rekenkracht dan de hamming prioriteitsfunctie.

Tot slot lijkt het mij dus het beste om voor een betere prioriteitsfunctie te kiezen. Een functie die een mooi evenwicht zoekt tussen geheugenverbruik en nodige rekenkracht zou het programma nog efficiënter kunnen maken, zowel voor puzzels waarbij weinig moves nodig als puzzels waarbij veel moves nodig zijn. Dit geldt natuurlijk enkel indien de grootte steeds ongeveer gelijk is (4x4 of 5x5).

5. Feedback

Dit practicum heeft mij veel meer bijgeleerd dan het vorige practicum. Het was daarnaast ook leuker om te maken. Het vorige practicum was iets te ingewikkeld en de moeilijkheidsgraad lag naar mijn inzien te hoog. Ik ben blij dat de deadline uitgesteld werd zodat we eerst les kregen over de prioriteitswachtrij in Java. Het is altijd motiverender om een practicum te maken dat handelt over zaken die we in de les behandeld hebben.

Toch heb ik weer meer tijd dan vooraf verwacht in dit practicum moeten steken. Ik zou persoonlijk liever hebben dat de practica op meer punten van het totaal zouden meetellen in plaats van de practica kleiner te maken.