Gegevensstructuren en Algoritmen
Practicum 2

March 20, 2013

Opgave: 8-Puzzel

Schrijf een programma dat het 8-puzzel probleem (en zijn natuurlijke veralgemeningen) oplost gebruikmakend van het A* zoekalgoritme[1].

Het probleem

Het 8-puzzel probleem is een populaire puzzel uitgevonden door Noyes Palmer Chapman in de jaren '70 van de 19e eeuw. Het wordt gespeeld op een 3×3 rooster met 8 vierkante tegels, genummerd van 1 tot 8 en een lege ruimte. Het doel is de tegels te herordenen zodat ze op volgorde staan. Je mag tegels horizontaal en verticaal verschuiven naar de lege ruimte. Hieronder tonen we een sequentie van geldige verplaatsingen van een initiëel bord configuratie (links) tot het doel (rechts).

```
    1  3        1     3        1  2  3        1  2  3        1  2  3
 4  2  5   =>   4  2  5   =>   4     5   =>   4  5     =>   4  5  6
 7  8  6        7  8  6        7  8  6        7  8  6        7  8

initieel                                                    doel
```

Beste-eerst oplossing

We beschrijven hier een oplossing die een algemene methodologie in artificiële intelligentie illustreert, het A* zoekalgoritme. We definiëren een toestand in het spel als de configuratie van het bord, het aantal verplaatsingen om die configuratie te bekomen, en de vorige toestand. Plaats eerst de initiële toestand (de initiële configuratie, 0 verplaatsingen en null als vorige toestand) in een prioriteitsrij (zie Heapsort, Sectie 2.4). Verwijder dan de toestand met de minimum prioriteit uit de prioriteitsrij en voeg alle naburige toestanden (diegene die met 1 verplaatsing bereikt kunnen worden) toe. Herhaal deze procedure tot de minimum toestand de doeltoestand is.

[1] Dit practicum is gebaseerd op een practicum van Princeton University.

Het succes van deze aanpak hangt af van de keuze van de prioriteitsfunctie voor een toestand. We beschouwen twee prioriteitsfuncties:

1. *Hamming prioriteitsfunctie.* Het aantal tegels in de verkeerde positie, plus het aantal verplaatsingen om de toestand te bereiken vanuit de initiële toestand. Intuïtief gezien zal een toestand met een klein aantal verkeerde tegels dicht bij de doeltoestand liggen, en we verkiezen een toestand die met zo weinig mogelijk verplaatsingen bereikt kan worden.

2. *Manhattan prioriteitsfunctie.* De som van de afstanden (som van de verticale en horizontale afstand) van de tegels naar hun doelpositie, plus het aantal verplaatsingen om de toestand te bereiken vanuit de initiële toestand.

Bijvoorbeeld, de Hamming en Manhattan prioriteiten van de initiële toestand hieronder zijn respectievelijk 5 en 10.

```
8  1  3     1  2  3     1  2  3  4  5  6  7  8    1  2  3  4  5  6  7  8
4     2     4  5  6     ----------------------    ----------------------
7  6  5     7  8        1  1  0  0  1  1  0  1    1  2  0  0  2  2  0  3

initieel    doel        Hamming = 5 + 0          Manhattan = 10 + 0
```

We maken hierbij een belangrijke observatie: willen we een puzzel van een bepaalde toestand in de prioriteitsrij oplossen, dan is het totaal aantal verplaatsingen (inclusief diegene die reeds gedaan zijn) minstens gelijk aan de prioriteit, zowel voor de Hamming als de Manhattan prioriteitsfunctie. (Voor de Hamming prioriteit klopt dit omdat elke verkeerde tegel minstens 1 verplaatsing moet doen om zijn doelpositie te bereiken. Voor de Manhattan prioriteit klopt dit omdat elke verkeerde tegel zijn Manhattan afstand tot de doelpositie moet afleggen. Merk op dat we de lege ruimte (lege tegel) niet meetellen in de berekening van de Hamming of Manhattan prioriteiten.)

Bijgevolg, van zodra we een toestand uit de prioriteitsrij halen, hebben we niet alleen een sequentie van verplaatsingen van de initiële bordconfiguratie tot het bord horend bij de toestand, maar is die sequentie ook de kortste. (Uitdaging voor de wiskundigen onder jullie: bewijs dit.)

Een kritische optimalisatie

Eens je dit algoritme geïmplementeerd hebt, zal je merken dat toestanden overeenkomend met dezelfde bordconfiguratie meerdere keren in de prioriteitsrij voorkomen. Om dit te vermijden, kan je naburige bordconfiguraties weigeren als ze dezelfde zijn als de vorige toestand.

```
8  1  3      8  1  3     8  1  3
4     2      4  2        4     2
7  6  5      7  6  5     7  6  5

vorig        huidig      weiger
```

Jouw taak

Schrijf een programma `Solver.java` dat een initiëel bord configuratie inleest en de minimum sequentie van bord configuraties die de puzzel oplossen uitschrijft. Schrijf ook het totaal aantal verplaatsingen uit.

De invoer bestaat uit de dimensie van het bord N gevolgd door de $N \times N$ initiële configuratie. De lege ruimte wordt aangeduid door 0. Hier is een voorbeeld van een mogelijke invoer:

```
% less puzzle04.txt
3
0  1  3
4  2  5
7  8  6
```

Net als bij de vorige assignment, bieden we ook nu weer een framework aan op basis van `ant`. Concreet zul je de files `Solver.java` en `Board.java` moeten implementeren. Aangezien we onze eigen testen runnen op jou code mag de interface van deze klassen **niet** worden gewijzigd! Compileren van de code kan ook nu weer via[2]:

```
ant -Dboard=Boards/puzzle04.txt
```

waarbij `puzzle04.txt` het bord is.

De uitvoer van het programma ziet er dan als volgt uit:

```
   1  3
4  2  5
7  8  6

1     3
4  2  5
7  8  6

1  2  3
4     5
7  8  6

1  2  3
4  5
7  8  6

1  2  3
4  5  6
```

[2]Deze commandline-opdracht werkt uiteraard ook onder Windows. Windows-gebruikers hoeven dus geen Unix commandline omgeving te installeren en zeker ook geen heel Unix systeem.

```
7  8

Minimum aantal verplaatsingen = 4
```

Merk op dat je programma moet werken voor willekeurige $N \times N$ borden (voor elke N groter dan 1), zelfs als de uitvoeringstijd daarmee heel groot wordt.

In bijlage (`opgave.zip`) hebben we enkele puzzels voorzien. Om je programma te helpen testen, geven we je het minimum aantal verplaatsingen van enkele puzzels: `puzzle22.txt` = 12, `puzzle24.txt` = 3, `puzzle26.txt` = 2.

Onoplosbare puzzels

Niet alle initiële bordconfiguraties kunnen tot de doeltoestand leiden.

```
% less puzzle-impossible3x3.txt
3
 1  2  3
 4  5  6
 8  7  0

% ant -Dboard=Boards/puzzle3x3-impossible.txt
Geen mogelijke oplossing
```

Test daarom eerst de invoer. Deze kan op de volgende manier getest worden. Verplaats eerst de lege plaats naar zijn doelpositie. Plaats nu alle rijen van de puzzel achter elkaar en stel de permutatie op. Bijvoorbeeld voor een permutatie van 1 .. N, e.g. 1 3 2, is p(1) = 1, p(2) = 3, p(3) = 2. De puzzel is enkel oplosbaar indien het teken van de permutatie positief is. Het teken $sgn(p)$ van de permutatie kan berekend worden via:

$$sgn(p) = \frac{\prod_{i<j} p(j) - p(i)}{\prod_{i<j} j - i}. \tag{1}$$

Vragen

Naast het implementeren van de solver, vragen we om onderstaande vragen te beantwoorden. Plaats ook nu weer je verslag in de vorm van een PDF-bestand in de map "report".

1. Leg kort uit hoe je Board.java geïmplementeerd hebt. Indien je extra methoden aan de Board API hebt toe toegevoegd, welke zijn dit en waarom?

2. Leg kort uit hoe je de toestanden in het spel hebt voorgesteld (bord, aantal verplaatsingen en vorige toestand)

3. Geef voor elk van de volgende puzzels het minimum aantal verplaatsingen om de doeltoestand te bereiken. Geef ook de uitvoeringstijd van het A* algoritme voor

de Hamming en de Manhattan prioriteitsfuncties. Duid aan wanneer de oplossing niet binnen een redelijke tijd (< 5 minuten) gevonden kan worden.

Puzzel	Aantal verplaatsingen	Hamming (s)	Manhattan (s)
puzzle28.txt	.	.	.
puzzle30.txt	.	.	.
puzzle32.txt	.	.	.
puzzle34.txt	.	.	.
puzzle36.txt	.	.	.
puzzle38.txt	.	.	.
puzzle40.txt	.	.	.
puzzle42.txt	.	.	.

4. Indien je willekeurige 4×4 of 5×5 puzzels zou willen oplossen, wat zou je verkiezen: meer tijd (bijvoorbeeld $10\times$ zo lang), meer geheugen (bijvoorbeeld $10\times$ zo veel), of een betere prioriteitsfunctie? Waarom?

5. Andere opmerkingen. Je mag ook feedback geven over dit practicum. Hoeveel en wat heb je bijgeleerd, vond je het interessant?

Indienen

De oplossing moet ten laatste vrijdag 29 maart om 14u00 uur ingediend worden via Toledo. Gebruik het commando:

```
ant release
```

om een zip bestand van je code en verslag te maken. Vergeet de naam van het resulterende bestand `build/firstname_lastname.zip` niet te wijzigen. Let op, de zip-file *moet* via dit commando gecreëerd worden!

Veel succes!