Gegevensstructuren en Algoritmen: Practicum 2

Mathias Van Herreweghe - r0456156

4 april 2014

1 Introductie

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 x 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.

In de code die bij dit verslag is toegevoegd wordt gebruik gemaakt van het A* algoritme met hamming en manhattan als prioriteitsfuncties.

2 Minimum aantal verplaatsingen en uitvoertijd

Puzzel	Aantal verplaatsingen	hamming (s)	manhattan (s)
puzzle28.txt	28	5s	2s
puzzle30.txt	30	8s	2s
puzzle32.txt	32	$>$ 5m	5s
puzzle34.txt	34	$>$ 5m	2s
puzzle36.txt	36	$>$ 5m	13s
puzzle38.txt	38	$>$ 5m	12s
puzzle40.txt	40	$>$ 5m	3s
puzzle42.txt	42	$>$ 5m	9s

3 Prioriteitsfuncties

We nemen grootte N als het aantal tegels op een één zijde van een bord.

1	2	3
4	5	6
7	8	0

Puzzel met N = 3.

3.1 Hamming - complexiteit (array accesses)

Hamming heeft een complexiteit van $N^2 + 1 =\sim\ N^2$ in het best-case geval, dit is wanneer alle tegels op een juiste plaats staan. Worst-case is de complexiteit $\sim\ 2 * N^2$, dit is wanneer alle tegels op een foute plaats staan. Dit hangt namelijk af van de if-conditie in de binnenste lus, omdat ik gebruik maak van een && operator om twee booleans te controleren, zal de tweede niet gecontroleerd worden als de eerste onwaar is. Als de eerste waar is wordt de tweede ook gecontroleerd en gebeuren er dus dubbel zoveel array accesses. Gemiddeld kan men veronderstellen dat de helft van de tegels niet op hun correcte plaats staan, dit leidt tot een average-case van $\sim\ 1,5 * N^2$.

3.2 Manhattan - complexiteit (array accesses)

Manhattan zal een complexiteit hebben van $3 * N^2 - 2 =\sim\ 3 * N^2$. Als de conditie waar is, zal in totaal 3 keer een array oproep gedaan worden in de binnenste lus. Omdat er maar voor 1 tegel (namelijk 0) maar 1 array access geldt in plaats van 3, worden er in de praktijk 2 array accesses minder gedaan maar dit wordt verwaarloosd in de tilde-notatie .

4 IsSolvable

Ik heb gebruikt gemaakt van een andere methode dan die van in de opgave. Mijn implementatie is gebaseerd op een document van Universidad Simón Bolívar.[1] Deze wijze is (in mijn ogen) overzichtelijker en beter begrijpbaar dan die van de opgave.

4.1 Werking en complexiteit

Ten eerste pas ik rij-linearisatie toe op de 2-dimensionale rij om een 1-dimensionale rij te bekomen, dit maakt de volgende stap gemakkelijker en efficiënter. Het lege vakje (waarde 0) verwaarlozen we en laten we uit de nieuw bekomen rij, dit omdat we deze niet willen meerekenen in de volgende stap.

Ten tweede worden het aantal inversies geteld. Een inversie is als tegel met een hoger cijfer voor een tegel met een lager cijfer in de rij voorkomt. Als dan de som van de inversies een even getal is, is de puzzel oplosbaar. Is de som oneven, dan is de puzzel onoplosbaar.

Het eerste deel, de rij-linearisatie, heeft een complexiteit van $N^2 - 1 =\sim N^2$. Het tweede deel heeft een complexiteit van $\sim N^2/2$. De deling door 2 valt af te leiden uit het feit dat de binnenste lus een steeds kleiner bereik moet afleggen, gaande van N tot 0 (dit is op het moment dat $i = j = N$), het gemiddelde hiervan leidt tot de helft.

De totale complexiteit van isSolvable komt dus neer op $N^2 + N^2/2 =\sim N^2$.

5 Aantal bord posities worst-case in geheugen, in functie van de bord grootte n

We nemen grootte N als het totaal aantal tegels op een bord.

1	2	3
4	5	6
7	8	0

Puzzel met N = 9.

Er kunnen maximum $N!/2$ verschillende borden in het geheugen zitten, de helft van mogelijke bordcombinaties (namelijk de onoplosbare) kunnen niet bereikt worden vanaf een oplosbare puzzel. Daarom wordt $N!$ door 2 gedeeld. De faculteit komt voort uit het feit dat bij elke gevonden puzzel er 1 nieuwe uniek puzzel minder gevonden kan worden;
$N * N - 1 * ... * 2 * 1 = N!$.

6 Betere prioriteitsfunctie(s)

De meest gebruikte prioriteitsfunctie voor het A* algoritme is manhattan volgens de resultaten van mijn zoekopdrachten, waaronder deze.[2]

Er is echter een betere prioriteitsfunctie, namelijk additive pattern database heuristic (met statische 6-6-3 partitionering). Manhattan (en hamming) scoren minder goed dan dit algoritme omdat ze geen rekening houden met het moeten verschuiven van de andere tegels, alvorens de huidige tegel op zijn bestemming kan raken. Hierdoor ontstaat een vrij grote onderschatting. Het additive pattern database algoritme houdt hier wel rekening mee. Hierdoor zal dit algoritme meestal een hogere waarde teruggeven dan manhattan en dus ook dichter bij het werkelijk aantal benodigde verschuivingen.

De verwerkingstijd van dit efficiënter algoritme is veel minder lang dan manhattan, hamming en dergelijke. Dit komt voornamelijk omdat er veel minder borden hoeven opgeslagen te worden.

Het geheugengebruik is ook minder aangezien enkel de triples van posities waarbij de som groter is dan manhattan, opgeslagen worden.[3]

Voor een complete en diepgaande tekst over dit algoritme verwijs ik je door naar dit artikel.[3]

7 Meer tijd, meer geheugen of betere prioriteitsfunctie

Het beste is een betere prioriteitsfunctie te gebruiken die de noodzaak voor meer tijd of geheugen wegneemt. Dat wil zeggen dat het minder tijd en geheugen gebruikt. Zo zal men moeilijkere en grotere puzzels kunnen oplossen dan dat men zou kunnen met enkel meer tijd of meer geheugen.

8 Algoritme voor grotere puzzels binnen praktische tijd

Het IDA* (Iterative-Deepening-A*) algoritme kan gemakkelijk binnen praktische tijd grotere puzzels vinden. Het is een combinatie van het A* algoritme en het DFS (Dept-First Search) algoritme.

Nadelig is dat IDA* een beetje trager is dan A* omdat het meerdere borden meerdere malen zal overlopen. IDA* met additive pattern database heuristic (met statische 6-6-3 partitionering)(kijk hiervoor naar sectie 6) zorgt er dan weer voor dat het algoritme veel sneller is dan A* met manhattan en dergelijke, snel genoeg voor grotere en moeilijkere puzzels dus. Het voordeel is dan weer wel dat er minder geheugen gebruikt wordt waardoor er minder kans is dat het bruikbaar geheugen vol raakt en het zoeken naar de oplossing dus gestopt zou moeten worden. Dit is omdat A* veel ongebruikte borden in de priority queue bijhoudt, IDA* doet dit niet aangezien het enkel de borden op zijn gebruikte pad bijhoudt.

Referenties

[1] `http://ldc.usb.ve/~gpalma/ci2693sd08/puzzleFactible.txt` (2008).

[2] `http://heuristicswiki.wikispaces.com/N+-+Puzzle`.

[3] en Richard E. Korf enSarit Hanan, A. F. Additive pattern database heuristics `http://arxiv.org/pdf/1107.0050` (2004).