Algoritmen en Datastructuren
Taak 2: L-vormige betegeling

Tom Naessens
Academiejaar 2010 – 2011

31 maart 2011

Algoritme

Algemeen idee

Het algemeen idee achter mijn algoritme is het volgende: elke keer hij de recursieve methode aanroept, plaatst hij in het midden van het rooster een L-tegel. De opening van de eerste L-tegel is gericht naar de plaats waar het zwart vlakje zich in het rooster bevindt. Als deze L-tegel geplaatst is splitst hij het volledige rooster op in vier delen. Nu plaatst hij in het midden van elk van de vier delen een tegel. Als het zwart vakje zich in het kwadrant bevindt waar een tegel geplaatst wordt, richt deze L-tegel zijn opening naar het zwarte vakje, anders richt hij zijn opening in de richting van het vakje van de L-tegel die door de vorige methode in dat kwadrant geplaatst is. Het volledige rooster wordt zo keer op keer opgedeeld tot de lengte van de zijde twee vakjes bedraagt. Op dit moment plaatst hij nog één L-tegeltje en doet hij verder niets meer.

Specifieke uitwerking

In deze paragraaf zal ik in gaan op hoe ik dit heb geïmplementeerd in Java. Als de methode `tile()` opgeroepen wordt, initialiseert hij een variabele `teller` op 0. Op deze manier wordt `teller` iedere keer terug op 0 gezet voor het geval dat we deze methode meerdere keren oproepen. (Als de teller niet op 0 zou gezet worden en het programma zou meerdere keren opgeroepen worden zou dit geen probleem zijn, maar voor de 'netheid' van het rooster doe ik dit toch.) Hierna wordt de dimensie van het rooster berekend aan de hand van de opgegeven variabele `k`. Aan de hand van deze dimensie wordt een twee-dimensionale tabel opgesteld waar het hokje dat overeen komt met de opgegeven rij en kolom op -1 gezet wordt. In de volgende stap wordt de recursieve methode, `verdeelGrid` opgeroepen met 5 parameters. De eerste parameter is de dimensie van rooster waar in gewerkt wordt. De tweede en derde parameter duiden de coördinaat van het zwarte vakje aan en de twee laatste parameters duiden de linkerbovenhoek van het kader aan waarin gewerkt wordt. In het eerste geval is dit dus 0,0. De variabelen die coördinaten aangeven zijn altijd absoluut tegenover het volledige rooster. De reden waarom ik hier twee 'extra' variabelen in voer zal verder nog duidelijk worden.

Het bepalen van het middelpunt is het eerste wat gebeurt in de methode `verdeelGrid`. Dit wordt opgeslagen in de variabele `mp` (middelpunt). Eigenlijk is het niet letterlijk het middelpunt dat hier wordt aangegeven maar de rechteronderhoek van het linkerboven kwadrant. Dit komt omdat we met

'hokjes' werken, en niet met punten in het vlak. We moeten ook opletten met deze coördinaten aangezien deze wel relatief zijn tegenover het deel van het rooster waar we in werken. We zullen dus altijd bij dit middelpunt een x- en een y-coördinaat moeten optellen, afhankelijk van welke hoek we willen aangeven. Deze 2 coördinaten zijn de variabelen `grow` (grid-row) en `gcol` (grid-col).
Vervolgens wordt er gecontroleerd als de dimensie van het kader groter is dan 1. Met andere woorden, als `k` > 0. Als dit het geval is weten we dat de dimensie minstens 2 op 2 is en dat we al zeker een tegel moeten tekenen. In dit `if`-statement bevinden zich nog een aantal `if`-statements die controleren in welk kwadrant het vakje dat zonet geplaatst is zich bevindt. We moeten hier wel opletten bij het vergelijken aangezien de coördinaten van de variabele `row` en `col` tegenover het volledige grid staan en de coördinaten van het middelpunt de coördinaten aanduiden van het middelpunt tegenover het kleine roostertje waarin we werken. Daarom tellen we telkens bij het middelpunt de variabele `grow` of `gcol` op om zo ook de coördinaten tegenover het grote kader te bekomen. Dit kwadrant wordt uiteindelijk opgeslaan in een variabele. Als dit voorgaande bepaald is roept hij de methode `drawTile` aan die een L-tegel tekent op de gewenste plaats met de gewenste oriëntatie.
Het is echter niet genoeg om alleen maar één tegeltje te tekenen. Het rooster moet ook nog eens opgedeeld worden. Dit is pas nodig als de dimensie groter is dan 2 op 2, of met andere woorden als `k` groter is dan 1.
Zoals in het eerste deel van deze methode wordt eerst gecontroleerd waar het zonet geplaatste tegeltje zich bevindt. We kunnen de variabele `kwadrant` hier hergebruiken om zo de juiste parameters aan de recursieve oproep mee te geven. Deze recursieve oproep doen we eigenlijk 4 keer, namelijk 1 keer voor elk kwadrant. De variabelen verschillen wel per kwadrant. Normaal worden altijd deze parameters meegegeven: $k-1$ (zodat de dimensie van het deelrooster aangepast wordt), de rij en kolom van de 4 punten rond de variabele `mp` en de linkerbovenhoek van het kwadrant waarin zal gewerkt worden na de oproep. Als het zwarte vakje zich in het kwadrant bevindt, wordt in plaats van de rij en kolom van het punt rond de variabele `mp` de rij en kolom van het zwarte vakje meegegeven.
Deze recursieve wordt herhaald zolang hij zichzelf oproept. Namelijk tot dat de variabele `k` in alle kwadranten gelijk is aan 0. Op dit moment gaat hij verder met de methode `tile` en geeft hij zijn uiteindelijke grid terug.

Optimalisaties

Nadat mijn algoritme werkte in alle gevallen heb ik enkele optimalisaties uitgevoerd:

Eerstvooral heb ik de de opsplitsing van de kwadranten opgedeeld. Ik had namelijk eerst 4 `if`-statements waar telkens werd gecontroleerd op twee voorwaarden (het rijnummer van het zwart of net-geplaatste vakje en het kolomnummer van het zwart of net-geplaatste vakje) om mijn rooster op te delen in vier kwadranten. Nu heb ik gebruik gemaakt van in totaal 6 `if`-statements zodat ik enkele lijntjes code kan hergebruiken. Dit is echter wel een minimale aanpassing: veel tijd zal hier zeker niet mee gespaard worden. Ook bij het plaatsen van een L-tegel heb ik dit doorgevoerd voor de leesbaarheid van de code.
Wat ik achteraf ook heb gedaan is mijn recursieve methode in 2 delen gesplitst. Eerst had ik een `if`-statement dat controleerde als `k` groter was dan 0. Als dit het geval was zette ik een tegeltje in de betreffende richting en deelde mijn rooster op in vier delen. Dit opdelen was echter niet nodig als `k` gelijk was aan 1 aangezien er in zo'n kader van twee op twee waar al één hokje gevuld staat maar plaats is voor één L-tegel. Daarom heb ik nu mijn methode in 2 opgesplitst. Als `k` groter is dan 0 zet hij een L-tegel maar deelt hij nog niet meteen op. Pas als `k` groter is dan 1 deelt hij het rooster ook op in vier delen. Dit laatste zal wel een voelbare impact hebben op de uitvoeringstijd. Voordat ik deze optimalisatie doorvoerde deelde de methode nog talloze keren, afhankelijk van de grootte van het originele rooster, het rooster op zonder dat dit nodig was. Nu doet hij hij dat niet meer.
De laatste optimalisatie zal ook niet echt een grote rol spelen in de uitvoeringstijd. Eerst had ik geen gebruik gemaakt van de variabele `kwadrant`. Ik voerde de `if`-constructie, die momenteel nog terug te vinden is in de methode `verdeelGrid` waar hij het kwadrant bepaalt, een tweede keer uit om het grid te verdelen. Na invoering van de variabele `kwadrant` moet hij het middelpunt niet meer met de rij en kolom vergelijken, maar komt hij door de waarde in `kwadrant` te weten welke argumenten hij moet meegeven aan de recursieve oproep.

Tijdscomplexiteit

Aangezien we kunnen berekenen hoeveel keer onze methode aangeroepen wordt kunnen we een rij opstellen. Voor `k` beginnend van 1 heb ik de volgende rij opgesteld:

$$1, 5, 21, 85, 341, 1365, 5461, 21845, 87381, 349525, 1398101$$

We herkennen hier in de volgende recurrente betrekking:

$$T(1) = 1, \qquad T(n) = 4T(n-1) + 1 \quad \text{voor} \quad n > 1.$$

Aangezien een schatting van de oplossing niet meteen voor handen lag heb ik de rij ingevoerd in *WolframAlpha*. Deze gaf mij als resultaat voor de oplossing:

$$T(n) = \frac{1}{3}\left(4^n - 1\right).$$

Vervolgens controleren we als deze schatting voor het speciale geval geldt:

$$T(1) = 1 = \frac{1}{3}\left(4^1 - 1\right) = \frac{1}{3}\left(4 - 1\right) = \frac{1}{3}\left(3\right) = \frac{3}{3} = 1.$$

We bewijzen nu volgens inductie dat

$$T(n+1) = 4T(n) + 1 = \frac{4}{3}(4^n - 1) + 1 = \frac{4^{n+1}}{3} - \frac{4}{3} + \frac{3}{3} = \frac{1}{3}(4^{n+1} - 1)$$

dus de schatting is aangetoond.

Analoog met het algemeen geval dat we terug vinden op p. 166 in het boek bij de substitutiemethode:
In het algemene geval

$$T(1) = a, \qquad T(n) = 4T(n-1) + b \quad \text{voor} \quad n > 1.$$

kan worden bewezen dat

$$T(n) = (a+b)2^{n-1} - b.$$

De oplossing van

$$T(n) = 4T(n-1) + \Theta(1)$$

wordt dus gegeven door

$$T(n) = \Theta(4^n).$$

Met andere woorden kunnen we stellen dat voor een variabele `k` de methode een uitvoeringstijd van $\Theta(4^k)$ zal hebben.