

1 Bomen: Quadtrees

Een *Quadtree* is een *recursieve datastructuur* waarmee speciale boom-objecten kunnen worden gerepresenteerd. Een boom is, net als een list, opgebouwd uit knopen, die *zelfverwijzend* (*self-referential*) kunnen zijn. Dit houdt in dat de klasse waarmee zo'n knoop gerepresenteerd wordt één of meerdere instantievariabelen bevat die verwijzen naar objecten uit dezelfde klasse. Voor Quadtrees geldt dat iedere interne knoop precies 4 zelfverwijzingen heeft, waarmee de opvolgers van zo'n knoop bereikt kunnen worden.

2 Leerdoelen

In deze opgave vragen we je om in Java een representatie voor Quadtrees te ontwerpen en te implementeren. Na afloop van deze opdracht ben je in staat om:

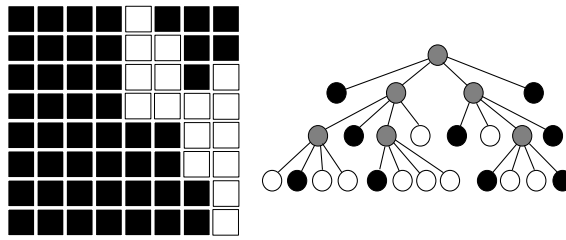
- Recursieve datastructuren te realiseren gebaseerd op zelfverwijzende klassen;
- conversies van en naar zo'n recursief datatype te implementeren gebruikmakende van recursie;
- een geschikte abstracte klasse te introduceren als basis voor de representatie van de verschillende soorten knopen.
- elke concrete knoop te definiëren als uitbreiding van deze basisklasse.

3 Instructie

Bestudeer allereerst de onderdelen uit hoofdstuk 21 van het boek die tijdens het college zijn besproken en de dia's over dit onderwerp. Maak eerst op papier een ontwerp op papier waarin van iedere klasse is aangegeven welke methodes deze bevat.

4 Probleemschets

Quadtrees kunnen gebruikt worden om tweedimensionale plaatjes compact op te slaan. Hierbij wordt uitgegaan van een vierkant plaatje van $N \times N$ pixels. Gemakshalve nemen we aan dat N een tweemacht is. Bovendien bekijken we alleen zwart-wit plaatjes, waarbij dus iedere pixel zwart of wit is. Het idee van het compact opslaan berust op de volgende recursieve procedure: Het $N \times N$ plaatje wordt allereerst opgedeeld in 4 kwadranten ieder met grootte $\frac{N}{2} \times \frac{N}{2}$. Deze worden volgens recursief gecompriëerd tot 4 Quadtrees. Deze 4 onderbomen vormen de 4 kinderen van de (interne) knoop, waarmee het gehele plaatje wordt gerepresenteerd. Het recursieve proces stopt natuurlijk zodra je bij een vierkant bent aanbeland van grootte 1. Voor de representatie van zo'n pixel gebruiken we twee soorten bladeren: één voor de witte en één voor de zwarte pixels. De werkelijke compressie gebeurt op het moment dat je de 4 onderbomen combineert tot een nieuwe boom. Als namelijk alle vier de onderbomen bladeren van dezelfde kleur zijn, dan wordt de gehele boom ook als een blad van die kleur gerepresenteerd. Op deze manier kun je complete (vierkante) vlakken (aangrenzende pixels van dezelfde kleur) door één enkele knoop representeren. We bekijken deze procedure aan de van een voorbeeld:



Links zien we een zwart-wit plaatje, waarbij $N = 8$ en rechts de representatie gebaseerd op Quad-trees. Zoals hierboven is aangegeven, wordt bij de conversie van het plaatje naar een Quadtree dit plaatje allereerst opgesplitst in 4 deelplaatjes van grootte 4. We nummeren deze met de klok mee waarbij we steeds linksboven beginnen. Elk van die 4 plaatjes wordt weer opgedeeld net zo lang tot we bij de individuele pixels zijn aanbeld. Na het converteren van onderbomen wordt steeds gecontroleerd of in plaats van een interne knoop een blad gebruikt kan worden. In dit voorbeeld, blijkt dit op verschillende plaatsen mogelijk te zijn, bijvoorbeeld voor de 4 bij 4 vlakken linksboven en linksonder, die geheel zwart zijn. Ook in de rechterhelft van het plaatje komen we voorbeelden hiervan tegen: rechtsboven bevindt zich een wit en een zwart 2 bij 2 vlak; rechtsonder een wit en twee zwarte 2 bij 2 vlakken. In de boom zie je de 4 bij 4 vlakken terug als de twee zwarte bladeren van de wortel. Ga zelf na hoe de rest van de boom is ontstaan.

5 Implementatie

Voor de representatie van plaatjes maken we gebruik van de klasse `Bitmap`:

```
public class Bitmap {
    // each bit is stored in a two dimensional array named raster
    private final boolean[ ][ ] raster;
    private final int bmWidth, bmHeight;

    /**
     * Creates an empty bitmap of size width * height
     * @param width
     * @param height
     */
    public Bitmap( int width, int height ) {
        raster = new boolean[width][height];
        bmWidth = width;
        bmHeight = height;
    }
}
```

Deze klasse bevat ook methodes om een bit op te vragen of te veranderen alsook een methode om een `Bitmap` om te zetten in een `String`, maar worden om ruimte te sparen hier niet getoond. Neem aan dat we voor wit de waarde **true** en voor zwart de waarde **false** gebruiken.

Voor het representeren van knopen gebruiken we de volgende abstracte basisklasse.

```
public abstract class QTreeNode {
    public abstract void fillBitmap( int x, int y, int width, Bitmap bitmap );
    public abstract void writeNode( Writer out );
    public abstract boolean sameLeaf( QTreeNode other_node );

    public static void fillArea( int x, int y, int width, Bitmap bitmap, boolean val ){
        for (int i = 0; i < width; i++) {
            for (int j = 0; j < width; j++) {
                bitmap.setBit(x+i, y+j, val);
            }
        }
    }
}
```

```

    }
}
}

```

Voor de daadwerkelijke knopen maken we drie uitbreidingen van deze klasse: `GreyNode` voor de interne knopen, `BlackLeaf` voor de zwarte en `WhiteLeaf` voor de witte bladeren. De abstracte methode `fillBitmap` vult de meegegeven bitmap conform de structuur van de boom. Hierbij kan gebruik gemaakt van de statische methode `fillArea` die een compleet vlak met opgegeven waarde vult. Naast deze drie uitbreidingen dien je ook nog een *wrapper* klasse `QTree` te introduceren waarmee Quadtrees van ‘buitenaf’ benaderd kunnen worden. Deze verbergt de interne representatie (d.m.v. gelinkte knopen) van zo’n boom.

```

public class QTree {
    QTreeNode root;

    public QTree( Reader input ) {
        root = readQTree( input );
    }

    public QTree( Bitmap bitmap ) {
        root = bitmap2QTree( 0, 0, bitmap.getWidth(), bitmap );
    }

    public void writeQTree( Writer out ) {
        root.writeNode( out );
    }

    public void fillBitmap ( Bitmap bitmap ) {
        root.fillBitmap(0, 0, bitmap.getWidth(), bitmap);
    }
}

```

Zoals je ziet bevat deze klasse twee constructoren: Met eerste wordt een boom gebouwd die in de vorm van een `Reader` wordt aangeleverd. Bij de tweede constructor wordt de boom op de eerder aangegeven wijze uit een bitmap geconstrueerd. Met de twee andere methodes gebeurt het tegenovergestelde: `writeQTree` schrijft de boom weg naar een `Writer` en `fillBitmap` vult de meegegeven `Bitmap` overeenkomstig de structuur van de boom. `Reader` en `Writer` zijn in Java voorgedefinieerde klassen voor het lezen uit en schrijven naar character streams; zie ook de standaard Java API.

Wegschrijven en inlezen

Met de methode `writeNode` wordt de knoopstructuur als bitreeks weggeschreven. Dit wegschrijven gebeurt door middel van een *pre-order traversal*. Iedere interne knoop wordt door een ‘1’ weergegeven en een blad door een ‘0’. Daarna volgt in geval van een interne knoop de representatie van de 4 onderbomen. Hebben we te maken met een blad dan schrijven we een ‘0’ voor een zwarte en een ‘1’ voor een witte knoop. Ga na dat dit voor de gegeven boom de reeks

“10011010001010010001010101100011000101000000”

oplevert. Het inlezen gebeurt op identieke wijze: komen we een ‘1’ tegen dan volgt daarna meteen de inhoud van een interne knoop. Komen we een ‘0’ tegen dan volgt een blad waarbij het eerstvolgende bit de kleur van dit blad bepaalt.

Conversies

In totaal hebben we 3 verschillende mogelijkheden om een plaatje te representeren: (1) als bitmap, (2) als Quadtree en (3) als een reeks van enen en nullen. In totaal dien je 4 directe conversies tussen deze representatie te implementeren, namelijk van (1) naar (2) en terug en van (2) naar (3) en terug.

6 Hulpbestanden

Alle codefragmenten die in deze opdracht getoond worden, zijn ook terug te vinden op Blackboard in de vorm van een aantal klassedefinities. Je mag deze voor je eigen uitwerking gebruiken mits je ze daar waar nodig op de juiste wijze van commentaar voorziet.

7 Producten

Als producten moeten alle `.java` bestanden ingeleverd worden. Zorg dat wat je inlevert tenminste compileerbaar is, anders wordt het sowieso niet als serieuze inleverpoging beschouwd.

8 Inleveren van je producten

Vóór zondag 29 maart, 11:00 uur, via Blackboard. Lever alle `.java` files uit je project in (dus ook de bestanden die zijn voorgegeven).