

# Gegevensstructuren en Algoritmen: Practicum 3

Academiejaar 2013-2014

## Inhoudsopgave

<b>1 Gedragscode</b>	<b>1</b>
<b>2 Wat is de bedoeling van dit practicum</b>	<b>1</b>
<b>3 Technische uitwerking</b>	<b>5</b>
<b>4 Deadline</b>	<b>6</b>
<b>5 Communicatie</b>	<b>6</b>
<b>A Hoe Ant installeren</b>	<b>6</b>
<b>B Hoe Ant gebruiken</b>	<b>7</b>

## 1 Gedragscode

(Laatste update gedragscode: 3 maart 2014)

De practica worden gequoteerd, en het examenreglement is dan ook van toepassing. Soms is er echter wat onduidelijkheid over wat toegestaan is en niet inzake samenwerking bij opdrachten zoals deze.


De oplossing en/of verslag en/of programmacode die ingediend wordt moet volledig het resultaat zijn van werk dat je zelf gepresteerd hebt. Je mag je werk uiteraard bespreken met andere studenten, in de zin dat je praat over algemene oplossingsmethoden of algoritmen, maar de bespreking mag niet gaan over specifieke code of verslagtekst die je aan het schrijven bent, noch over specifieke resultaten die je wenst in te dienen. Als je het met anderen over je practicum hebt, mag dit er dus NOOIT toe leiden, dat je op om het even welk moment in het bezit bent van een geheel of gedeeltelijke kopie van het opgeloste practicum of verslag van anderen, onafhankelijk van of die code of verslag nu op papier staat of in elektronische vorm beschikbaar is, en onafhankelijk van wie de code of het verslag geschreven heeft (mede-studenten, eventueel uit andere studiejaar, volledige buitenstaanders, internet-bronnen, e.d.). Dit houdt tevens ook in dat er geen enkele geldige reden is om je code of verslag door te geven aan mede-studenten.

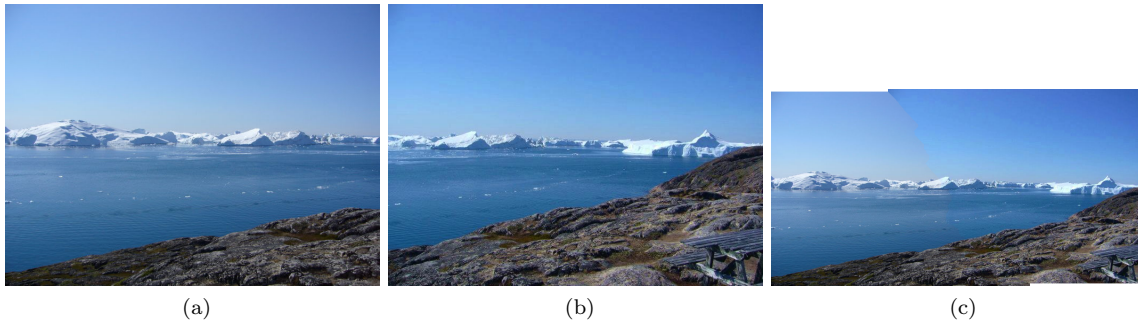
Elke student is verantwoordelijk voor de code en het werk dat hij of zij indient. Als tijdens de beoordeling van het practicum er twijfels zijn over het feit of het practicum zelf gemaakt is (bvb. gelijkaardige code, grafieken, of oplossingen met andere practica), zal de student gevraagd worden hiervoor een verklaring te geven. Indien dit de twijfels niet wegwerkt, zal er worden overgegaan worden tot het melden van een onregelmatigheid, zoals voorzien in het examenreglement.

## 2 Wat is de bedoeling van dit practicum

In dit practicum implementeer je een Image Compositing algoritme wat twee afbeeldingen zal samenvoegen. Varianten van dit algoritme worden in de praktijk gebruikt om meerdere foto's samen te voegen tot een

groot panorama. Voor onze implementatie is het de bedoeling is dat de grens tussen de twee afbeeldingen zo weinig mogelijk opvalt. Een mogelijke manier om dit te bereiken is deze grens zo te kiezen zodat het verschil tussen de twee afbeeldingen op deze grens geminimaliseerd wordt. Een voorbeeld hiervan is in Figuur 1 weergegeven. Verder moet je ook een verslag schrijven waarin je enkele vragen beantwoord (zie sectie 2.4).

 De focus van dit practicum ligt zowel op de code als het verslag.



Figuur 1: Afbeelding `zee1.png` (a) en `zee2.png` (b) worden samengevoegd tot (c).

## 2.1 Het programma

Ons programma zal twee afbeeldingen samenvoegen door deze op elkaar te plakken, en dan een grens te zoeken die zo weinig mogelijk opvalt. Concreet gebeurt dit via het commando:

```
ant run -Dimg1=./images/zee1.png -Dimg2=./images/zee2.png
```

Dit voegt de afbeeldingen `zee1.png` en `zee2.png` aan elkaar. Optioneel kan je ook de argumenten `offsetx` en `offsety` meegeven. Hierdoor zal `img2` met `offsetx` pixels naar rechts geschoven en met `offsety` pixels naar beneden. Als `offsety` een negatieve waarde heeft wordt `img2` naar boven geschoven ten opzichte van `img1`. Op deze manier is het mogelijk om (manueel) een betere stitching te zoeken. Bijvoorbeeld, de stitching in Figuur 1 is gemaakt met het commando:

```
ant run -Dimg1=./images/zee1.png -Dimg2=./images/zee2.png -Doffsetx=318 -Doffsety=-7
```

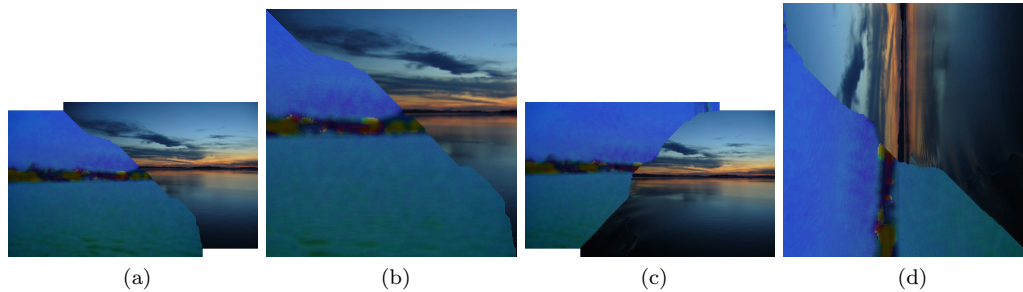
Om je programma te testen zijn er zijn 3 voorbeelden van in- en uitvoer gegeven:

img1	img2	offset	output
zee1.png	zee2.png	(318, -7)	zee-out.png
color1.png	color2.png	(0, 0)	color-out.png
spiraal1.png	spiraal2.png	(0, 0)	spiraal-out.png

## 2.2 Jouw taak

Jouw taak is het schrijven van een klasse `gna.Stitcher` met daarin de methodes:

- **seam**: geeft de gevonden grens (seam) terug als een reeks `Positions`. Deze grens is een lijn doorheen de overlappende delen van de afbeelding, zodat op de lijn de afstand tussen de twee afbeeldingen geminimaliseerd wordt.
- **floodfill**: op basis van de gevonden grens markeert dit algoritme, voor elke pixel, welke afbeelding gebruikt moet worden in de samengevoegde afbeelding.
- **stitch**: deze functie combineert `seam` en `floodfill`.



Figuur 2: Afhankelijk van de offsets kan de grens vertrekken in de linker- of rechterbovenhoek, maar de methode `seam` krijgt alleen overlappende delen, gedraaid zodat de grens linksboven vertrekt.

Let op, je mag de package structuur niet wijzigen! Nieuwe klassen toevoegen mag uiteraard wel (zet alle code in de directory `gna`). De drie bovenstaande methoden moeten de specificaties volgen die we in commentaar bij die methodes geschreven hebben in `Stitcher.java`, plus de beschrijving die we in dit verslag geven. Nieuwe methodes toevoegen mag. We raden aan om je implementatie van de drie methoden te testen door JUnit-tests te schrijven. Probeer een zo efficiënt mogelijke implementatie te maken!

### 2.2.1 Stitch

De methode `stitch` krijgt alleen de overlappende delen van de ingevoerde afbeeldingen. Deze worden op voorhand getransformeerd zodat de grens altijd van de linkerbovenhoek naar de rechterbenedenhoek loopt. Het teruggegeven beeld is niet de samengevoegde afbeelding, maar geeft aan op welke plaatsen welke afbeelding gebruikt wordt. Markeer de pixels op de grenslijn met de speciale grensconstante (`Stitch.SEAM`). Pixels uit de eerste afbeelding worden gemarkeerd met de integer `Stitch.IMAGE1`, pixels uit de tweede afbeelding met `Stitch.IMAGE2`. Na het bepalen van de grenslijn, zal je een **floodfill** algoritme moeten gebruiken om de overige pixels in te vullen met `Stitch.IMAGE1` of `Stitch.IMAGE2`.

### 2.2.2 Floodfill

Floodfill is een standaard recursief algoritme. Als je de afbeelding voorstelt als een graaf waarbij elke pixel verbonden is met zijn linker-, rechter-, onder- en bovenbuur, dan komt floodfill overeen met diepte-eerst zoeken. In pseudo-code ziet het algoritme er als volgt uit:

```
flood(pixel)
    Is pixel already colored or on the seam?
        Yes: stop
        No: fill pixel
            flood(above pixel)
            flood(below pixel)
            flood(right pixel)
            flood(left pixel)
```

Als je floodfill recursief implementeert, eindig je waarschijnlijk met een `StackOverflowError`. In plaats daarvan zal je een iteratieve versie moet ontwikkelen die niet steunt op recursieve oproepen.

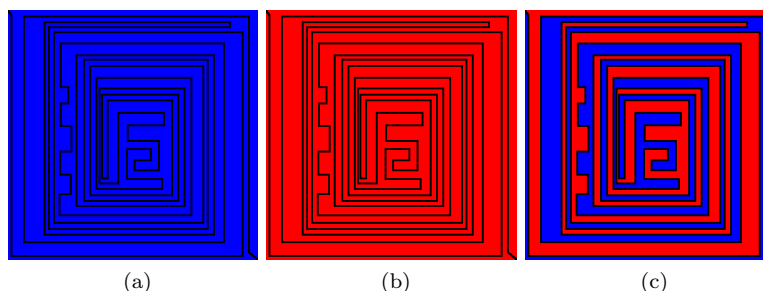
Bij de implementatie mag je veronderstellen dat de afbeelding slechts in 2 regio's is opgedeeld.

### 2.2.3 Seam

Afhankelijk van de offsets kan de grens vertrekken in de linker- of rechterbovenhoek, maar de methode `seam` krijgt alleen overlappende delen, gedraaid zodat de grens linksboven vertrekt (zie figuur 2). Dus de grens

tussen de twee afbeeldingen loopt altijd van linksboven naar rechtsonder, en moet zo worden gekozen zodat de kost van deze grens zo klein mogelijk is. De kost is per pixel de afstand tussen de kleurwaarden van de twee afbeeldingen in deze pixel. Deze wordt berekend met `ImageCompositor.pixelSqDistance(int x, int y)`. De totale kost is de som van de kosten van alle pixels op de grens. Bij het zoeken van een grens moet je diagonaal aangrenzende pixels als aangrenzend beschouwen. In het pad kan je dus bijvoorbeeld rechtstreeks van positie (7, 7) naar (8, 8) gaan. Gebruik Dijkstra's kortste-pad algoritme.

De grens tussen twee afbeeldingen kan eender welk pad zijn. Een voorbeeld van een niet-triviale grens is gegeven in Figuur 3. Dit resultaat krijg je door de afbeeldingen `spiraal1.png` en `spiraal2.png` te stitchen zonder `offset`'s.



Figuur 3: Afbeelding `spiraal1.png` (a) en `spiraal2.png` (b) worden samengevoegd tot (c).

## 2.3 Links

Voor geïnteresseerde studenten: je kan online gemakkelijk toepassingen vinden van gelijkaardige algoritmes:

- Mosaics of Scenes with Moving Objects: <http://users.soe.ucsc.edu/~davis/panorama/>
- Interactive Digital Photomontage: <http://grail.cs.washington.edu/projects/photomontage/>

## 2.4 Verslag

Naast de implementatie 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. Een populaire manier om kleuren voor te stellen, die ook in dit practicum gebruikt wordt, is een kleur voor te stellen door drie getallen: het eerste getal stelt de hoeveelheid rood voor, het tweede de hoeveelheid groen en het derde de hoeveelheid blauw. De getallen kunnen variëren van 0 tot 255. Bijvoorbeeld: (255,0,0) is dus rood, (123,0,0) is donkerrood, (0,0,0) is zwart, (255,255,255) is wit, (255,0,255) is paars, (100,100,100) is grijs, enz. Voor twee kleuren,  $(r_x, g_x, b_x)$  en  $(r_y, g_y, b_y)$  drukt  $\sqrt{(r_x - r_y)^2 + (g_x - g_y)^2 + (b_x - b_y)^2}$  uit hoe fel twee kleuren verschillen, m.a.w. wat de “afstand” tussen de twee kleuren is. De functie `ImageCompositor.pixelSqDistance(int x, int y)` die je in dit practicum gebruikt, geeft het kwadraat van de afstand tussen twee kleuren terug.

Gegeven onderstaande afbeeldingen:

(7,0,0)	(0,1,0)	(0,0,1)
(2,0,0)	(0,8,0)	(0,0,5)
(1,0,0)	(0,1,0)	(0,0,8)
(0,0,0)	(0,0,0)	(0,0,0)
(0,0,0)	(0,0,0)	(0,0,0)
(0,0,0)	(0,0,0)	(0,0,0)

Stel je programma wordt uitgevoerd op die twee afbeeldingen (met geen offset). Geef de grafe waarop het korste pad algoritme wordt uitgevoerd<sup>1</sup>. Wat is het resulterende kortste pad?

2. Stel dat we een andere afstandsfunctie zouden gebruiken om de afstand tussen twee kleuren te berekenen, bijvoorbeeld  $\sqrt{(r_x - r_y)^2 + (g_x - g_y)^2}$ . Zouden er afbeeldingen bestaan waarop je programma sneller of trager loopt met de nieuwe kleurafstandsfunctie? Of blijven afbeeldingen waarop het programma traag was even traag? Verklaar je antwoord.
3. Stel dat je afbeelding niet in het geheugen past, maar wel op harde schijf. Je software kan een pixel ophalen van de harde schijf. Om de uitvoerafbeelding weg te schrijven kan je ook pixels schrijven naar de harde schijf. Beschrijf hoe je je programma zou kunnen aanpassen om zulke grote afbeeldingen te kunnen stitchen.
4. Wat is de best-case tijdscomplexiteit van je programma?
5. Stel dat je programma niet het korste pad zou zoeken, maar het langste niet-cyclische pad. Hoe zou de afbeelding die je programma als uitvoer geeft er uit zien?
6. Optioneel: Als je wilt kan je leuke afbeeldingsparen en het resultaat van samenvoeging in de map `myexamples` plaatsen. Ant zal ze dan opnemen in de zip file. Een aantal creatieve beelden zullen we op Toledo plaatsen. Zorg wel dat je zip file niet te groot wordt.

### 3 Technische uitwerking

1. Voorbereiding:
  - (a) Installeer het programma Ant én voer het eens uit. Doe dit in het begin, zo kom je vlak voor de deadline niet voor verrassingen te staan. Dit document bevat een sectie Hoe Ant installeren.
  - (b) Schrijf JUnit tests die automatisch de correctheid van je (hulp)functies testen.
2. Het echte werk
  - (a) Implementeer in `gna.Stitcher` de methodes `seam`, `floodfill`, en `stitch`. Je mag de package structuur en signature van deze methodes niet wijzigen! De main methode (in `Main.java`) wordt uitgevoerd via het commando:

```
ant run -Dimg1=./images/zee1.png -Dimg2=./images/zee2.png
```

Dit commando kan je aanpassen om andere afbeeldingen samen te voegen.
  - (b) Voer tijdens je implementatie regelmatig `ant test` uit. Dit checkt je oplossing op veelvoorkomende fouten. Ook je eigen JUnit tests worden dan uitgevoerd (a.o. de tests in bestanden waarvan de bestandsnaam begint met “Test” worden uitgevoerd). Als je je tests of de main methode via Eclipse wilt uitvoeren, zal je in Eclipse `lib/libpract.jar` moeten toevoegen aan je build path.
  - (c) Voer de tests uit en schrijf het verslag. Zie sectie Wat is de bedoeling van dit practicum. Denk eraan om tijdens het implementeren je code met ant te testen!
3. Releasen
  - (a) Voer `ant test` uit om te controleren op veelvoorkomende fouten.
  - (b) Controleer of op je verslag je naam en studentnummer staat zodat we bij het printen weten welk verslag van wie is.

---

<sup>1</sup>Dit mag maar hoeft geen computertekening te zijn; je mag bijvoorbeeld ook een scan gebruiken of ascii-art

- (c) Maak een zip file met Ant. Deze opgave bevat een sectie Hoe Ant gebruiken. Ant is verplicht. Zo heeft iedere zip-file dezelfde directory structuur; ervaring leert dat dit niet lukt als studenten de ZIP file met de hand maken. Hierdoor kunnen we sneller verbeteren en dus sneller feedback geven. Als je .zip file duidelijk niet met Ant is gemaakt, zullen we hem niet verbeteren.
- (d) Ant maakt een zip file `build/firstname_lastname_studentnumber.zip`. Vul hier je voornaam, achternaam en studentnummer (inclusief letters ‘r’, ‘s’ of ‘m’) in in de bestandsnaam. Verwissel niet je voornaam en je achternaam.
  - i. Fout: `Janssens-Jan-r0123456.zip`
  - ii. Fout: `Jan-Janssens-0123456.zip`
- (e) Open je ZIP file en controleer of alles er in zit.
- (f) Upload je ZIP file op Toledo. Je hoeft geen papieren verslag in te dienen.  
Indien Toledo down is, mail dan een screenshot hiervan en je zip file naar de verantwoordelijke van dit practicum (zie sectie Communicatie).

## 4 Deadline

De deadline is vrijdag 30 mei 2014 14:00. Wie te laat indient loopt het risico een 0 te behalen op het practicum.

## 5 Communicatie

Stel je vragen via het Toledo forum.

De verantwoordelijke van dit practicum is **mathy** punt **vanhoef** apenstaartje **cs** punt **kuleuven** punt **be**. Hier kan je ook naar mailen voor zaken die je niet publiekelijk kan communiceren. Andere practica kunnen andere verantwoordelijken hebben.

## A Hoe Ant installeren

Ant is niet standaard bijgeleverd bij Java en ook niet bij Windows. Je moet Ant dus eerst installeren.

**Windows thuis:** Het tweede google-resultaat over hoe je Ant installeert onder Windows levert <http://code.google.com/p/winant/> op. Dit is een heel eenvoudige installer. Deze installer vraagt wat de directory is waar JDK is geïnstalleerd; dit is typisch zoets als `C:\Program Files\Java\jdk1.7.0_17` (afhankelijk van welke JDK je precies hebt).

**Linux thuis:** Voor Ubuntu en Debian: de installatie is eenvoudigweg “`sudo apt-get install ant`” in-typen in een terminalvenster. Voor andere distributies: gebruik je package manager.

**PC-labo computerwetenschappen (gebouw 200A):** Ant is reeds geïnstalleerd.

**LUDIT pc-labo:** Ongekend; het is waarschijnlijk veel makkelijker ant op een eigen machine te installeren.

**Mac OS X:** <sup>2</sup>

1. Open een terminal
2. Type volgende commando's:

---

<sup>2</sup>Bron: <http://gauravstomar.blogspot.be/2011/09/installing-or-upgrading-ant-in-mac-osx.html>

```
curl -O http://apache.petsads.us//ant/binaries/apache-ant-1.9.3-bin.zip
unzip apache-ant-1.9.3-bin.zip
sudo mkdir -p /usr/local/
sudo cp -rf apache-ant-1.9.3 /usr/local/apache-ant
export PATH=/usr/local/apache-ant/bin:$PATH
echo 'export PATH=/usr/local/apache-ant/bin:$PATH' >> ~/.profile
```

Ant is nu geïnstalleerd. Als je bij het uitvoeren van “ant” de error krijgt dat je JDK moet installeren, dan moet je dat doen. Je hebt JDK (Java Development Kit) nodig om Java programma’s te compileren in het algemeen, dus ook als je Java programma’s compileert via Ant.

## B Hoe Ant gebruiken

1. Start een terminalvenster (dit werkt ook onder Windows: menu start, dan execute, dan “cmd” intypen; zie anders <http://www.google.com/search?q=how+to+open+windows+command>)
2. Navigeer naar de directory waar je bestanden voor dit practicum staan, meer bepaald de directory waar zich `build.xml` in bevindt. Met “cd” verander je van directory en met “ls” (unix) of “dir” (Windows) toon je de bestanden en directories in de huidige directory.
3. Type “**ant release**”. Ant doet een aantal checks om je tegen een aantal fouten te beschermen. Check dus of Ant geen error gaf.