

Laboration 4

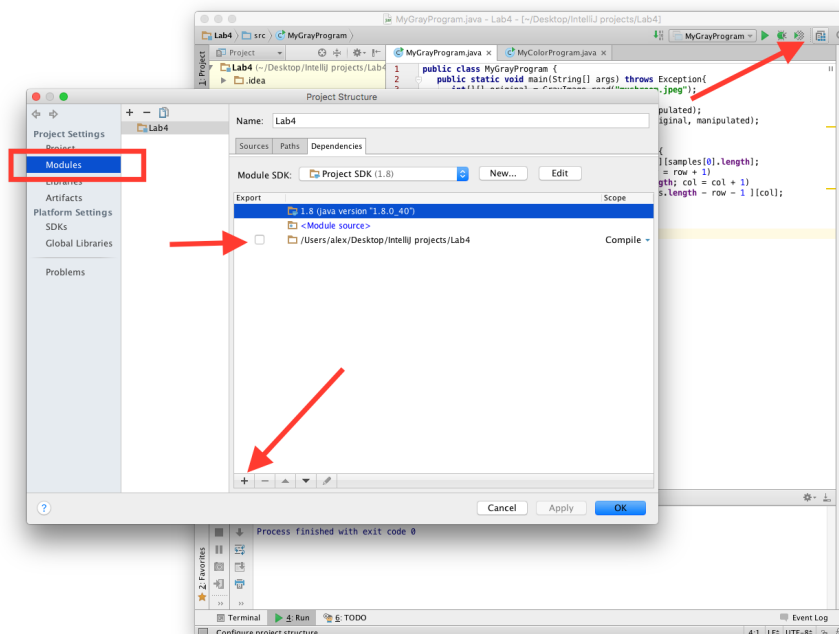
Objekt-orienterad Programmering

Syfte

I denna laboration skall vi återigen behandla transformering av data, denna gång avseende digitala bilder. Syftet med laborationen är att få förståelse för och erfarenhet av att använda flerdimensionella fält.

Uppstart

Börja med att skapa en nytt projekt Lab4. Ladda ner [zip-filen](#) från kursens hemsida och kopiera källkodsfiler (med .java extension) till src mappen i projektet. Sedan skapa en mapp med namnet lib och kopiera klassfilerna (med .class extension) till den här lib mappen. Sista steget är att lägga till lib mappen av projektet som en *dependency*, så att vi kan faktiskt använda class filerna. Man kan lägga till en dependency via 'Project structure' knappen, sedan navigerar man till 'Modules' i 'Project Settings' meny och fortsätter med att klicka på 'Dependencies' tabben. Sedan klickar man på plus och väljer 'JARs or directories...' för och lägger till lib mappen. Det ser ungefär ut som nedanstående bild:



INSTITUTIONEN FÖR DATAVETENSKAP
Chalmers Tekniska Högskola

Besök: Rännvägen 6
Telefon: +46 (0)31-772 6154
E-post: alexg@chalmers.se
Web: <http://www.cse.chalmers.se/edu/course/TDA540/>



Redovisning

Lämna in samtliga källkodsfiler som en komprimerad zip-fil i [Fire](#).

Snabbkurs i digital bildrepresentation

En digital bild kan representeras som ett tvådimensionellt fält av bildpunkter eller pixlar (eng. pixels, för picture elements). I en digital gråskalebild är en bildpunkt ett heltalsvärde i intervallet 0-255, där värdet 0 betecknar svart och värdet 255 betecknar vitt. En bild har en viss storlek, dvs ett visst antal bildpunkter i höjddled och ett visst antal bildpunkter i sidled. En gråskalebild kan således representeras med ett tvådimensionellt fält av typen `int[] []`, där den första dimensionen definierar bildens höjd och den andra dimensionen definierar bildens bredd. En variabel som representera en gråskalebild med höjden `HEIGHT` pixlar och bredden `WIDTH` pixlar kan skapas med satsen:



```
int[] [] grayImage = new int[HEIGHT][WIDTH];
```

Elementet `grayImage[20][40]` anger alltså gråtonen i den pixel som återfinns 20 pixlar nedåt från bildens övre vänstra hörn och 40 pixlar åt höger. Observera att `grayImage[0][0]` är pixeln i övre vänstra hörnet i bilden och att `grayImage[HEIGHT-1][WIDTH-1]` är pixeln i nedre högra hörnet.

En digital färgbild lagras på RGB-format, där varje bildpunkt utgörs av tre heltalsvärden i intervallet 0-255. De enskilda värdena representerar intensiteten av färgerna rött, grönt och blått. En färgbild kan således avbildas med ett tredimensionellt fält av typen `int[] [] []`, är den första dimensionen definierar bildens höjd, den andra dimensionen definierar bildens bredd och den tredje dimensionen representerar färgerna rött, grönt respektive blått. En variabel som representera en färgbild med höjden `HEIGHT` pixlar och bredden `WIDTH` pixlar kan skapas med satsen:



```
int[] [] [] colorImage = new int[HEIGHT][WIDTH][3];
```

Elementer `colorImage[18][56][1]` anger alltså intensiteten av grönt i den pixel som återfinns 18 pixlar nedåt från bildens övre vänstra hörn och 56 pixlar åt höger.

Javas API innehåller ett antal klasser för att hantera både gråskalebilder och färgbilder som lagras på olika bildformat, såsom jpeg, gif och png. För att kunna genomföra laborationen behöver ni inte ha någon kunskap om dessa klasser, utan de används i de färdig skrivna klasserna som bifogas till laborationen.

Uppgift 1: Gråskalebilder

I denna uppgift skall ni skriva ett antal metoder för behandling av gråskalebilder. Till er hjälp har ni klasserna `GrayImage`, `GrayImagePanel`, `GrayImageWindow` och `MyGrayProgram`. Klassen `GrayImage` tillhandahåller de publika klassmetoderna:

`static int[][] read(String fileName)` som läser in en bild, lagrad på gif-, jpg-, jpeg- eller png-format, från filen `fileName` och returnerar den som ett tvådimensionellt heltalsfält; om ursprungsbilden i filen är en färgbild översätts denna till en gråskalebild

`static void write(String fileName, int[][] picture)` som skriver ut fältet `picture` som en gråskalebild på formatet gif, jpg, jpeg eller png till en fil med namnet `fileName`

Klassen `GrayImageWindow` tillhandahåller en konstruktor:

`GrayImageWindow(int[][] picture1, int[][] picture2)` som skapar ett fönster i vilket fälten `picture1` och `picture2` ritas ut som gråskalebilder (enligt nedan)



Klassen `MyGrayProgram` innehåller nedanstående kod och producerar fönstret som visas ovan.

```
public class MyGrayProgram {
    public static void main(String[] args) throws Exception{
        int[][] original = GrayImage.read("mushroom.jpeg");
        int[][] manipulated = upDown(original);
        GrayImage.write("upDownMushroom.jpeg", manipulated);
        GrayImageWindow iw = new GrayImageWindow(original, manipulated);
    }

    public static int[][] upDown(int[][] samples) {
        int[][] newSamples = new int[samples.length][samples[0].length];
        for (int row = 0; row < samples.length; row = row + 1)
            for (int col = 0; col < samples[row].length; col = col + 1)
                newSamples[row][col] = samples[samples.length - row - 1][col];
        return newSamples;
    }
}
```

CHALMERS

I klassen finns metoden `public static int[] [] upDown(int[] [] samples)` som tar ett tvådimensionellt fält `samples` (som representerar en gråskalebild) och returnerar ett nytt tvådimensionellt fält (som också representerar en gråskalebild). Det nya fältet som metoden returnerar är en spegelvänd kopia av fältet `samples` roterad runt den horisontella axeln, dvs första raden i det nya fältet är sista raden i `samples`, andra raden i det nya fältet är näst sista raden i `samples`, osv. Metoden användas alltså för att vända en bild upp och ner.

Metoden `main` läser in en fil (`mushroom.jpeg`) som innehåller en bild och lagrar denna som en gråskalebild i det tvådimensionella fältet `original`, skapar ett nytt fält `manipulated` genom att anropa metoden `upDown` med fältet `original`, skriver ut fältet `manipulated` som en bild till en fil (med namnet `upDownMushroom.jpeg`) och skapar slutligen ett fönster där `original` och `manipulated` ritas ut som bilder.

Din uppgift är att utöka klassen `MyGrayProgram` med följande metoder:

`public static int[] [] leftRight(int[] [] samples)` som returnerar en spegelvänd kopia av `samples` roterad runt den vertikala axeln

`public static int[] [] invert(int[] [] samples)` som returnerar en kopia av `samples` där värdet $s_{i,j}$ för varje element i `samples` har ersätts med värdet $255 - s_{i,j}$

`public static int[] [] toBlackWhite(int[] [] samples)` som returnerar en kopia av `samples` där värde $s_{i,j}$ för varje element i `samples` har ersätts med värdet 0 om $s_{i,j}$ är mindre än 128 och med värdet 255 om $s_{i,j}$ är större eller lika med 128

`public static int[] [] contour(int[] [] samples)` som returnerar konturen av `samples`

En kontur i en svartvit bild definieras enligt nedan:

I konturen av bilden `samples`, skall en pixel vara svart om och endast om `samples[i][j]` är svart och antingen (i, j) ligger på randen i `samples` eller (i, j) har minst en granne som är vit i `samples`. Randen i bilden utgörs av de yttre raderna och kolumnerna. Grannar till (i, j) är alla (i_k, j_k) sådan att $i_k \in i - 1 \dots i + 1$ och $j_k \in j - 1 \dots j + 1$.

Bildpunkter som ligger på randen kan inte direkt beräknas med detta uttryck eftersom dessa bildpunkter saknar en eller flera närliggande punkter. Ni kan bortse från bildpunkterna på randen och låta dessa ha samma värden som i den svartvita varianten av originalbilden. För att rita ut t.ex. den



Original

leftRight

invert

toBlackWhite

contour

spegelvända kopian ändrar ni satsen

CHALMERS

```
int[] [] manipulated = upDown(original);
```

i main-metoden i klassen MyGrayProgram till

```
int[] [] manipulated = leftRight(original);
```

Vill ni lagra den nya bilden som en jpeg-bild ändrar ni också satsen

```
GrayImage.write("upDownMushroom.jpeg", manipulated);
```

exempelvis till

```
GrayImage.write("leftRightMushroom.jpeg", manipulated);
```

Om ni vill använda någon annan bild än den som finns på filen `mushroom.jpeg` kan ni naturligtvis göra detta genom att lagra en annan bildfil i mappen där ni har er programkod och ändra i main-metoden så att denna bildfil läses in i stället för filen `mushroom.jpeg`.

Uppgift 2: Färgbilder

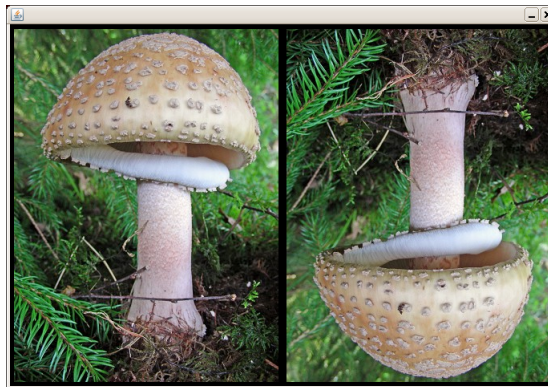
För färgbilder finns klasserna `ColorImage`, `ColorImagePanel`, `ColorImageWindow` och `MyColorProgram`. Klassen `ColorImage` tillhandahåller klassmetoderna:

`static int[][][] read(String fileName)` som läser in en bild på gif-, jpg-, jpeg- eller png-format från filen `fileName` och returnerar den som ett tredimensionellt heltalsfält

`static void write(String fileName, int[][][] picture)` som skriver ut fältet `picture` som en bild på formatet gif, jpg, jpeg eller png till en fil med namnet `fileName`

Klassen `ColorImageWindow` tillhandahåller en konstruktor:

`ColorImageWindow(int[][][] picture1, int[][][] picture2)` som skapar ett fönster i vilket fälten `picture1` och `picture2` ritas ut som färgbilder (enligt nedan)



Klassen `MyColorProgram` har utseendet:

```
public class MyColorProgram {
    public static void main(String[] args) throws Exception {
        int[][][] original = ColorImage.read("mushroom.jpeg");
        int[][][] manipulated = upDown(original);
        ColorImage.write("upDownMushroom.jpeg", manipulated);
        ColorImageWindow iw = new ColorImageWindow(original, manipulated);
    }

    public static int[][][] upDown(int[][][] samples) {
        int[][][] newSamples = new int[samples.length][samples[0].length][3];
        for (int row = 0; row < samples.length; row = row + 1)
            for (int col = 0; col < samples[row].length; col = col + 1)
                for (int c = 0; c < samples[row][col].length; c = c + 1)
                    newSamples[row][col][c] = samples[samples.length - row - 1][col][c];
        return newSamples;
    }
}
```

CHALMERS

1. Din uppgift är att utöka klassen `MyColorProgram` med följande metoder:

- `public static int[][] leftRight(int[][] samples)`
- `public static int[][] invert(int[][] samples)`
- `public static int[][] toGray(int[][] samples)`
- `public static int[][] toBlackWhite(int[][] samples)`



leftRight



invert



toGray



toBlackWhite

Inversen till en färgbild erhålls genom att för varje pixel i originalbilden ersätta intensiteten $s_{i,j}$ för var och en av färgerna röd, grön och blå med värdet $255 - s_{i,j}$.

För att få bästa kvalitén på den gråskalebild (och därmed den svartvita bild) som erhålls från en färgbild skall ni beräkna bildpunkternas luminans. Luminansen L är den för ögat upplevda ljusheten hos en yta och definieras som:

$$L = 0.299r + 0.587g + 0.114b$$

där r är intensiteten av rött, g är intensiteten av grönt och b är intensiteten av blått. I den gråskalebild som beräknas från en färgbild, sätts intensiteten av färgerna rött, grön och blått i varje pixel till värdet L i motsvarande pixel i färgbilden. I den svartvita bild som erhålls från en färgbild, sätt intensiteten av färgerna rött, grön och blått i varje pixel till värdet 0 om L i motsvarande pixel i färgbilden är mindre än 128 och till 255 om L är större eller lika med 128.

2. I bildanalys används ofta så kallade *faltningsfilter* för att lyfta fram sådant som är intressant i bilden. Det kan t.ex. röra sig om att ta bort brus, eller att reducera eller framhäva konturer i bilden. Vid användning av faltningsfilter beräknas ett nytt värde på en bildpunkt genom att, förutom att beakta bildpunkten själv, även beakta närliggande bildpunkter. Ett faltningsfilter kan således anses som en matris. Faltningsfiltret

$$\begin{matrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{matrix}$$

är ett filter som *förändrar skärpan* i bilden. Filtret anger att det nya värdet i en bildpunkt beräknas genom att multiplicera bildpunktens gamla värde med 9 och subtrahera med varje närliggande bildpunkt. Detta kan också uttryckas på följande sätt:

Alla bildpunkter $new_{i,j}$ i den bild som erhålls med faltningsfiltret och som inte ligger på randen i bilden beräknas av uttrycket

$$\begin{aligned} new_{i,j} = & -1 * old_{i-1,j-1} - 1 * old_{i-1,j} - 1 * old_{i-1,j+1} \\ & - 1 * old_{i,j-1} + 9 * old_{i,j} - 1 * old_{i,j+1} \\ & - 1 * old_{i+1,j-1} - 1 * old_{i+1,j} - 1 * old_{i+1,j+1} \end{aligned}$$

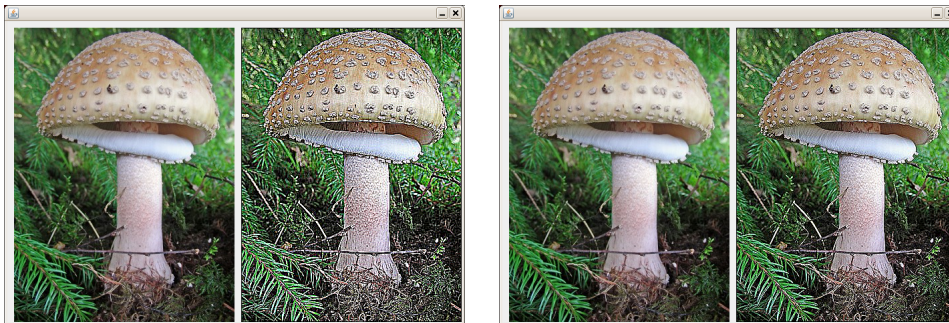
CHALMERS

där $old_{i,j}$ är värdet för bildpunkten (i, j) i bilden som filtreras. I en färgbild appliceras faltningfiltret på varje enskild färgkomponent. Värdet av den nya bildpunkten $new_{i,j}$ kan bli negativt eller större än 255. Detta måste kontrolleras. Negativa värden sätts till 0 och värden större än 255 sätts till 255.

Ett annat faltningsfilter som förändrar skärpan är

$$\begin{matrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{matrix}$$

För att beräkna det nya värdet av en bildpunkt används i detta filter endast 4 av de 8 närbelägna bildpunkterna. Resultatet av de båda filtren ser ni nedan. Utöka klassen `MyColorProgram` med



sharpenOne

sharpenTwo

metoderna:

- `public static int[][][] sharpenOne(int[][][] samples)`
- `public static int[][][] sharpenTwo(int[][][] samples)`

som implementerar de båda ovan beskrivna faltningsfiltren.

3. För att detektera kanter i bilder appliceras två filter på original bilden (ett filter för att få fram kanterna i x -led och ett filter för att få fram kanterna i y -led). Ett välkänt kantdetekteringsfilter är Sobel-filtret, vars faltningsmatriser har följande utseende:

$$\begin{matrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{matrix}$$

och

$$\begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{matrix}$$

Om vi för varje enskild färgkomponent betecknar resultatet från dessa båda filter för bildpunkten (i, j) som $d_x(i, j)$ respektive $d_y(i, j)$, beräknas¹ slutresultatet från Sobel-filteringen som:

$$s(i, j) = \sqrt{d_x(i, j)^2 + d_y(i, j)^2}$$

Utöka klassen `MyColorProgram` med metoden

¹Observera att d_x och d_y behöver inte ligga i $[0 - 255]$ intervallet.

CHALMERS

```
public static int[][][] sobel(int[][][] samples)
```

som implementerar Sobel-filtret enligt beskrivningen ovan.



Sobel-filtrerats



Sobel-filtrerats och sedan inverterats

Slutkommentar

Antalet bearbetningar man kan göra på bilder är näst intill obegränsat. Ni kan säkert komma på egna förslag på metoder att implementera. Den intresserade kan också hitta många ytterligare exempel på bildbehandling och filter genom sökning på nätet. Några lämpliga adresser att börja med är:

- http://en.wikipedia.org/wiki/Image_editing
- http://en.wikipedia.org/wiki/Sobel_operator