CS4551 Multimedia Software Systems (Spring 2018)

DCT-Based Image Compression

- No due date. No submission is required.
- **Do not use any Java built-in image class methods, library, or tools to complete this task.**

What your program should do

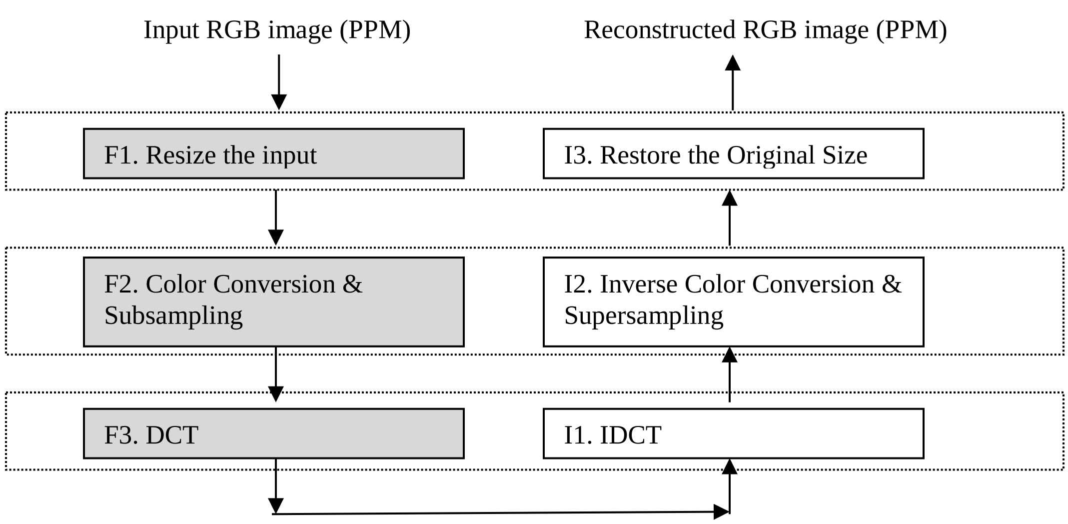
Name your main application as *CS4551_[Your_Last_Name].java*. Your program should accept one command line argument indicating for the input PPM file name.

<eg> On Command Prompt

```
java CS4551_Doe Ducky.ppm
```

DCT-based Image Compression

Implement a DCT transform algorithm.



Encoding Steps	*Decoding Steps*
F1. Read and resize the input image Read the input ppm file containing RGB pixel values for encoding. First, if the image size is not a multiple of 8 in each dimension, make (increase) it become a multiple of 8 and pad with zeros. For example, if your input image size is 21x14, make it become 24x16 and fill the extra pixels with zeros (black pixels).	**I3. Remove Padding and Display the image** Display the reconstructed image. Remember that you padded with zeros if the input image size is not multiple of 8 in both dimensions (width and height). Restore the original input image size by removing extra padded rows and columns.
F2. Color space transformation and Subsampling	**I2. Inverse Color space transformation and Supersampling**

Transform each pixel from RGB to YCbCr using the equation below: $$\begin{pmatrix} Y \\ Cb \\ Cr \end{pmatrix} = \begin{bmatrix} 0.2990 & 0.5870 & 0.1140 \\ -0.1687 & -0.3313 & 0.5000 \\ 0.5000 & -0.4187 & -0.0813 \end{bmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$ Initially, RGB value ranges from 0 and 255. After color transformation, Y should range from 0 to 255, while Cb and Cr should range from -127.5 to 127.5. (*Truncate if necessary.*) Subtract 128 from Y and 0.5 from Cb and Cr so that they span the same range of values [-128,127] Subsample Cb and Cr using 4:2:0 (MPEG1) chrominance subsampling scheme. ***If Cb(Cr) is not divisible by 8, pad with zeros.***	Supersample Cb and Cr so that each pixel has Cb and Cr. This means that one Cb will be used for 4 Y values. Add 128 to the values of the Y component and 0.5 to the values of the Cb and Cr components. If using a color image, transform from the YCbCr space to the RGB space according to the following equation: $$\begin{pmatrix} \mathrm{R} \\ \mathrm{G} \\ \mathrm{B} \end{pmatrix} = \begin{bmatrix} 1.0000 & 0 & 1.4020 \\ 1.0000 & -0.3441 & -0.7141 \\ 1.0000 & 1.7720 & 0 \end{bmatrix} \begin{pmatrix} \mathrm{Y} \\ \mathrm{Cb} \\ \mathrm{Cr} \end{pmatrix}$$ ***Common mistake:** After this step, you have to make sure that the resulting RGB values are in the range between 0 and 255. Truncate if necessary.*
F3. Discrete Cosine Transform Perform the DCT for Y image using the following steps: • Divide the image into 8x8 blocks. Scan each block in the image in raster order (left to right, top to bottom) • For each 8x8 block, perform the DCT transform to get the values F_{uv} from the values f_{xy}. The elements F_{uv} range from -2^{10} to 2^{10}. Check max and min and assign -2^{10} or 2^{10} for the values outside of the range so that the values range from -2^{10} to 2^{10}. Perform the DCT for Cb image and Cr image, too.	**I1. Inverse DCT** Perform the inverse DCT to recover the values f_{xy} from the values F_{uv} and recover Y, Cb, Cr images**.**

DCT Formula

$$F_{uv} = \frac{1}{4} C_u C_v \sum_{x=0}^{7} \sum_{y=0}^{7} f_{xy} \cos\frac{(2x+1)u\pi}{16} \cos\frac{(2y+1)v\pi}{16}$$

$C_u = 1/\sqrt{2}$ for $u=0$, $C_u = 1$ otherwise. $C_v = 1/\sqrt{2}$ for $v=0$, $C_v = 1$ otherwise. f_{xy} is the x-th row and y-th column pixel of the 8x8 image block (x and y range from 0 to 7). The element F_{uv} is DCT coefficient value in the u-th row and v-th column after DCT transformation. (u and v range from 0 to 7).

The inverse DCT Formula

$$f'_{xy} = \frac{1}{4} \sum_{u=0}^{7} \sum_{v=0}^{7} C_u C_v F'_{uv} \cos\frac{(2x+1)u\pi}{16} \cos\frac{(2y+1)v\pi}{16}$$