**PROGRAMMING ASSIGNMENT**

**Evaluation of JPEG Codecs – Lossless Technique**

***Multimedia System 6010 – Fall 2015***

***Khoa Hoang***

**Requirement for the project**:

* Python 2.7.9 (or PyCharm 4.5 or other IDE for python)
* Install few modules such Numpy, Math, Matplotlib

**Report project**

* Copy the value of the pixel of the given image to the text file (name: ***TextImage1.txt***). The program will read the value from this file and load into the program for the calculation of compression and decompression.
* A function - ***Predictor(Dx,ax,bx,cx,ay,by,cy)*** -is created to carry out all the tasks of the assignment. In the argument of ***Predictor:***
  + Dx is the variable of the name of each cases
  + ax, bx, cx are variables for compressing calculation
  + a y, by, and cy are variables for decompressing calculation
* When the program is run, it will call the function ***Predictor*** and will:
  + - Perform the tasks of 7 cases (displaying the value of images).
    - Return the values of the compressed image compression ratio (***Cr***), the compressed Bits/pixel ***(Bpix),*** and the RMS Error ***(RMSE)***.
    - Display a comparing table of those value above
    - Draw two graph to visualize the result of 7 cases of lossless compression (see below)

***Object Cr Bpix RMSE***

X^ = A 2.64 3.03 0.0

X^ = B 2.59 3.09 0.0

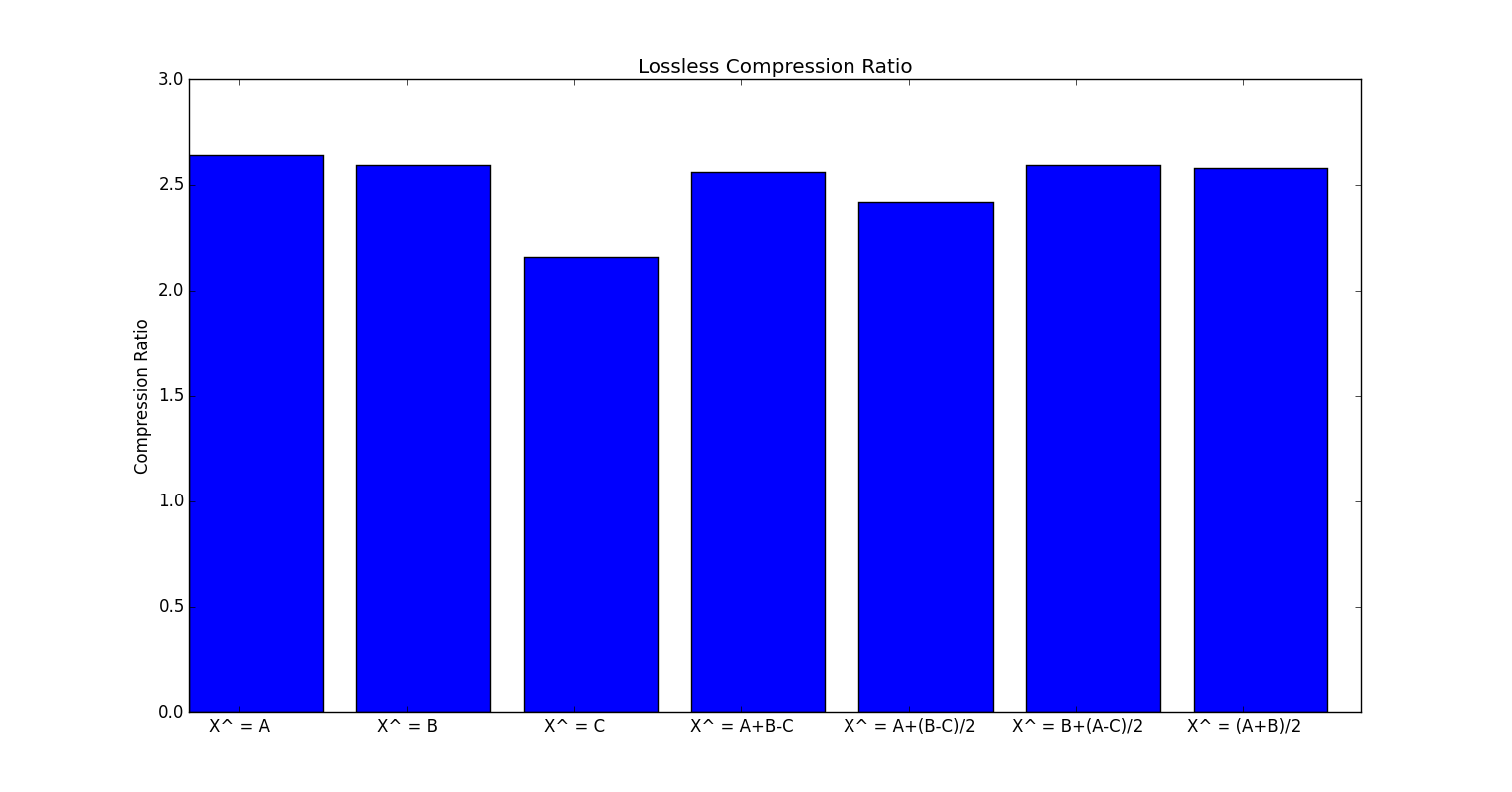
X^ = C 2.16 3.7 0.0

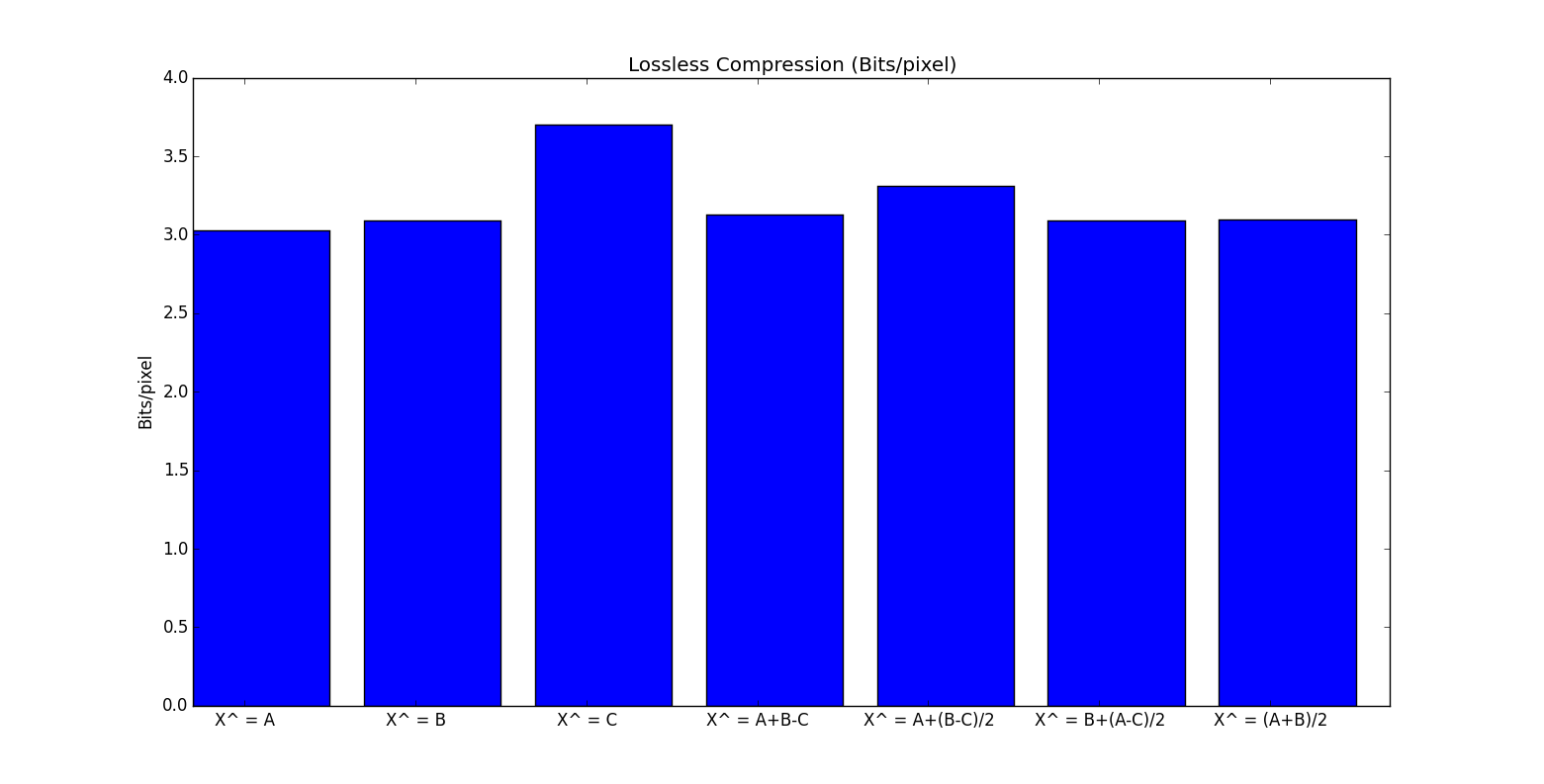
X^ = A+B-C 2.56 3.13 0.0

X^ = A+(B-C)/2 2.42 3.31 4.14

X^ = B+(A-C)/2 2.59 3.09 4.79

X^ = (A+B)/2 2.58 3.1 3.99

Photos:



As the values in the table and the charts showing:

* Generally the compression ratio of lossless is **not high**: the highest is X^=A=, and lowest is X^=C
* The value of Bits/pixel for the compressed image is **very high** (>3). This mean all 7 cases, the compressed images are almost like the original ones. The case X^=C is like the original image the most.
* Theoretically, the RMSE of lossless compression should be zero. However, it is not in the cases of X^=A+(B-C)/2; X^=B+(A-C)/2; and X^=(A+B)/2. The reason is that the Huffman table only works with the integer number. Three cases above might not return integer number. So I have to round the result to convert them from float to integer. This conversion creates a tiny different value of pixel. Thus, it makes the decompressed image different from the original one.
* For the other four cases, after running the program, it returns the sequence of numbers of the decompressed image the same with the sequence of numbers of the original image (please run the program in python to see the result.)