Linear Algebra, F2020	Posted: October 3, 2020
Project 1	
Lecturer: Hyang-Won Lee	**Due: October 15, 2020**

Important Notes

- This is a programming project for demonstrating the idea of image compression. You will need to write a computer program (in any language you want). You are allowed to any libraries or APIs available in the language you will be using.
- Before you work on this project, you should watch the video as instructed on the course website.
- If you do collaborate with other fellow students, please print the names of your collaborators
- Upload your report and code at ecampus

1. (**50 points**) Write a computer program that performs the following:
   (a) Take a grayscale image and form a matrix A. Each entry in A is a pixel in the image, and the size of matrix A is equal to the resolution of the image. If needed, you might want to chop off the image and make A an $n \times n$ matrix.
   (b) For given $n = 2^t, t = 0, 1, ...$, construct an n-point Haar matrix H in the lecture. Every column in H_n should be normalized, so that $H^{-1} = H^T$.
   (c) Perform 2-D Discrete Haar Wavelet Transform (DHWT) $B = H^T A H$.
   (d) For given $k = 2^s, s = 0, 1, ...$, construct an $n \times n$ matrix $\hat{B}$ such that
   - its $k \times k$ upper left corner is equal to the $k \times k$ upper left corner of B
   - all the other entries are zero

   So if $k = n$, $\hat{B} = B$.
   (e) Perform the inverse 2-D Discrete Haar Wavelet Transform (IDHWT) $\hat{A} = H \hat{B} H^T$.
   (f) Show the reconstructed image $\hat{A}$ on the screen.

2. (**50 points**) Get any 2 grayscale images of any format from anywhere (Internet, your personal photos, etc.). It is recommended that you get one image with low frequency components, and one image filled with high frequency components. Use your computer program to do the following:
   (a) As k increases, observe the quality of reconstructed image.
   (b) Describe any difference between low-freq. image and high-freq. image.

(c) Discuss any findings or thoughts.

3. (**50 points**) For n-point Haar matrix H, define H_l and H_h as follows:

$$H^T = \begin{bmatrix} H_l \\ H_h \end{bmatrix},$$

where H_l and H_h have the same number of rows. That is, H_l and H_h are the top half and bottom half rows of H^T. Note that H_l (or H_h) is a part of basis capturing relatively low (or high) frequency components.

(a) Show that the DHWT $B = H^T AH$ is given as

$$H^T AH = \begin{bmatrix} H_l A H_l^T & H_l A H_h^T \\ H_h A H_l^T & H_h A H_h^T \end{bmatrix}$$

(b) Show that the IDHWT $A = HBH^T$ is given as

$$HBH^T = H_l^T H_l A H_l^T H_l + H_l^T H_l A H_h^T H_h + H_h^T H_h A H_l^T H_l + H_h^T H_h A H_h^T H_h$$

(c) Each term in (b) is an $n \times n$ matrix. Show each term in (b) as an image, and discuss what each of matrices contains.

(d) Repeat (a) and (b) for the first term in (b). That is, let $H_l \begin{bmatrix} H_{ll} \\ H_{lh} \end{bmatrix}$, and substitute this into the first term in (b) and show that

$$H_l^T H_l A H_l^T H_l = H_{ll}^T H_{ll} A H_{ll}^T H_{ll} + H_{ll}^T H_{ll} A H_{lh}^T H_{lh} + H_{lh}^T H_{lh} A H_{ll}^T H_{ll} + H_{lh}^T H_{lh} A H_{lh}^T H_{lh}.$$

Show each term as an image on the screen, and interpret them.

4. (**This problem requires the knowledge studied in data structures and algorithms, and will not be graded. Do as much as you can, if you enjoy challenges**) This example illustrates that careful understanding of mathematical properties can sometimes be used to (significantly) reduce the computational complexity of an algorithm. Consider the matrix vector multiplication $\mathbf{y} = H_n\mathbf{x}$ where $\mathbf{x}$ is an n-dimensional column vector and H_n is the n-point Haar matrix. For simplicity, assume that H_n is the original Haar matrix before normalization. Naively, this transformation requires n^2 multiplications and n^2 summations, and so, $2n^2$ arithmetic operations in total. We will construct a routine that does the same transformation with only $O(n)$ computations. (이 문제에서는 수학적 성질을 주의깊게 관찰해서 알고리듬의 복잡도를 줄이는 예를 다루고자 한다. 이를 위해, H_n이 n-point Haar matrix, $\mathbf{x}$는 n 차원 열벡터라고 할 때 $y = H_n\mathbf{x}$를 계산하는 것을 생각하자. 논의를 간단하게 하기 위해 H_n은 normalize를 하기 전의 Haar matrix라고 가정하자. 단순한 행렬과 벡터의 곱을 생각하면 이 곱셈 $H_n\mathbf{x}$는 n^2번의 곱셈과 n^2번의 덧셈을 필요로 한다. 이 문제에서는 동일한 계산을 수행하지만 $O(n)$의 복잡도를 갖는 알고리듬을 만들고자 한다)

(a) The *mod − p perfect shuffle permutation matrix* $P_{p,r}$ treats the components of the input column vector $\mathbf{x} \in \mathbb{R}^n$, $n = pr$, as cards in a deck. The deck is cut into p equal "piles" and reassembled by taking one card from each pile in turn. Thus, if $p = 3$ and $r = 4$, then the piles are $\mathbf{x}(1:4) = [x_1; x_2; x_3; x_4]$, $\mathbf{x}(5:8) = [x_5; x_6; x_7; x_8]$, and $\mathbf{x}(9:12) = [x_9; x_{10}; x_{11}; x_{12}]$ (here, we are using MATLAB syntax for reading entries of a vector), and ($mod - p$ perfect shuffle permutation matrix $P_{p,r}$는 곱해지는 열벡터 $\mathbf{x} \in \mathbb{R}^n (n = pr)$의 원소를 마치 쌓여있는 카드를 다루듯이 섞는다. 먼저 카드의 deck은 p개의 동일한 더미로 나누고, 각 더미로부터 카드를 하나씩 뽑아서 카드들을 재조합한다. 예를 들어, $p = 3, r = 4$라고 하면, 더미들은 $\mathbf{x}(1:4) = [x_1; x_2; x_3; x_4]$, $\mathbf{x}(5:8) = [x_5; x_6; x_7; x_8]$, and $\mathbf{x}(9:12) = [x_9; x_{10}; x_{11}; x_{12}]$가 되고, $P_{3,4}$를 $\mathbf{x}$에 곱하면 다음과 같다)

$$\mathbf{y} = P_{3,4}\mathbf{x} = \begin{bmatrix} \mathbf{x}(1:4:12) \\ \mathbf{x}(2:4:12) \\ \mathbf{x}(3:4:12) \\ \mathbf{x}(4:4:12) \end{bmatrix} = \begin{bmatrix} x_1 \\ x_5 \\ x_9 \\ x_2 \\ x_6 \\ x_{10} \\ x_3 \\ x_7 \\ x_{11} \\ x_4 \\ x_8 \\ x_{12} \end{bmatrix}.$$

Prove that $P_{p,r}^T = P_{r,p}$.

(b) For $n = 2^t, t = 1, 2, ...$ and $n = 2m$, prove that

$$P_{2,m}^T H_n = (H_2 \otimes I_m) \begin{bmatrix} H_m & 0 \\ 0 & I_m \end{bmatrix}.$$

(c) For $\mathbf{x} \in \mathbb{R}^n$ and $n = 2m$, define $\mathbf{x}_T = \mathbf{x}(1:m)$ and $\mathbf{x}_B = (m+1:n)$. Prove that

$$\mathbf{y} = H_n\mathbf{x} = P_{2,m} \begin{bmatrix} H_m\mathbf{x}_T + \mathbf{x}_B \\ H_m\mathbf{x}_T - \mathbf{x}_B \end{bmatrix},$$

that is,

$$\mathbf{y}(1:2:n) = H_m\mathbf{x}_T + \mathbf{x}_B \qquad \mathbf{y}(1:2:n) = H_m\mathbf{x}_T - \mathbf{x}_B.$$

(d) By applying the above idea, write a recursive routine for the computation of $\mathbf{y} = H_n\mathbf{x}$ that requires only $O(n)$ computations. You must prove that the computation complexity of your routine is $O(n)$.

(e) Compare the runtime of the naive computation of $\mathbf{y} = H_n\mathbf{x}$, and the runtime of the recursive computation.