Ariella Mann and Austin Kee
CS40
Arith design doc
2/25/24

C Step #	Compression	Decompression	DC Step #
1	<u>Read in from file on command line</u> (or standard input if the file is not given) - Pnm_rdr with throw any exceptions if the file is not properly formatted - The information in the file is represented as scaled integers/RGB values and will be stored in a pnm_ppm - **Input**: Command line arguments - **Output**: Pnm_ppm with information from the file stored inside - **Testing**: print out the information in the pnm_ppm to make sure it was properly stored	<u>Print out decompressed image</u> - Use pnm_ppmwrite to print out the 2D array to standard input to show the decompressed image - **Input**: 2D array - **Output**: decompressed image - **Testing**: use ppmdiff to diff the decompressed image with the original	13
2	<u>Trim image</u> - If the dimensions are not even integers, then reset the height/width to the integer minus 1 to remove the last column or row - **Input**: the pnm_ppm that the information from the file was read into - **Output**: newly trimmed image of all the same information but a potentially new width/height - **Info lost**: if the width or height is not an even integer, the last row or col will be lost. - **Testing**: print out the width and height before trimming and after to ensure it worked properly		

3	<u>Convert to component video</u> - Divide each value by the maxVal as read in from the file - Cast every value as a float - Go through each pixel in image and convert the RGB float to component video using given operations for Y, Pb, and Pr. - **Input**: RGB value - **Output**: Component video - **Info lost**: We lose the information about each RGB value from the image in this step - **Testing**: print out component video information and ensure all operations were properly performed	<u>Convert from component video to floats</u> - Perform reverse equations to change component video back to RGB values - The integers will range from 0 to the denominator value that was chosen - Set each pixel to the RGB values by setting in pixmap→pixels - **Input**: Component video - **Output**: floats - **Testing**: compare to floats uses in compression steps	12
4	<u>Pack into 32 bit code</u> - Take average of four pixels in 2x2 block which is Pb and Pr - Convert to four bit value using given function → this function takes a chroma between -0.5 and +0.5 - Use Y value which was calculated before, and transform into cosine coefficients (a, b, c, d) using given operations - Convert b, c, d to 5-bit value (cast as signed int from unsigned int) - Bitpack.c: Store a, b, c, d, index(Pb), index(Pr) into a 32 bit code word - **Input**: 2x2 block - **Output**: 32 bit code - **Info lost**: Quantization for chroma and luma numbers results in losing information because one value represents a range of values in order to store the “rounded” value more compactly.	<u>Read in 32 bit words</u> - Flip each word from big endian order to little endian order - Unpack the code to get the a,b,c,d pb and pr variables and store into local variables from code word - Convert four bit chroma values using the equation provided to return floats - Inverse the cosine transformation to get Y1, Y2, Y3, Y4 - **Input**: 32 bit code - **Output**: information for each 2x2 block - **Testing**: Comparison to information from compression steps	8, 9, 10, 11
5	<u>Write compressed binary image to standard output</u> - Print out header - Flip code word to be in big endian - Print out each code word using row major	<u>Read in the header of compressed file</u> - Use fscanf with the same string used in the header from Compression step - Allocate space for a 2D array of pixels of the given width and height to store all of the final RGB values inue for	6, 7

	- **Input**: 32 bit code - **Output**: stdout statements	denominator - **Input**: file of compressed image