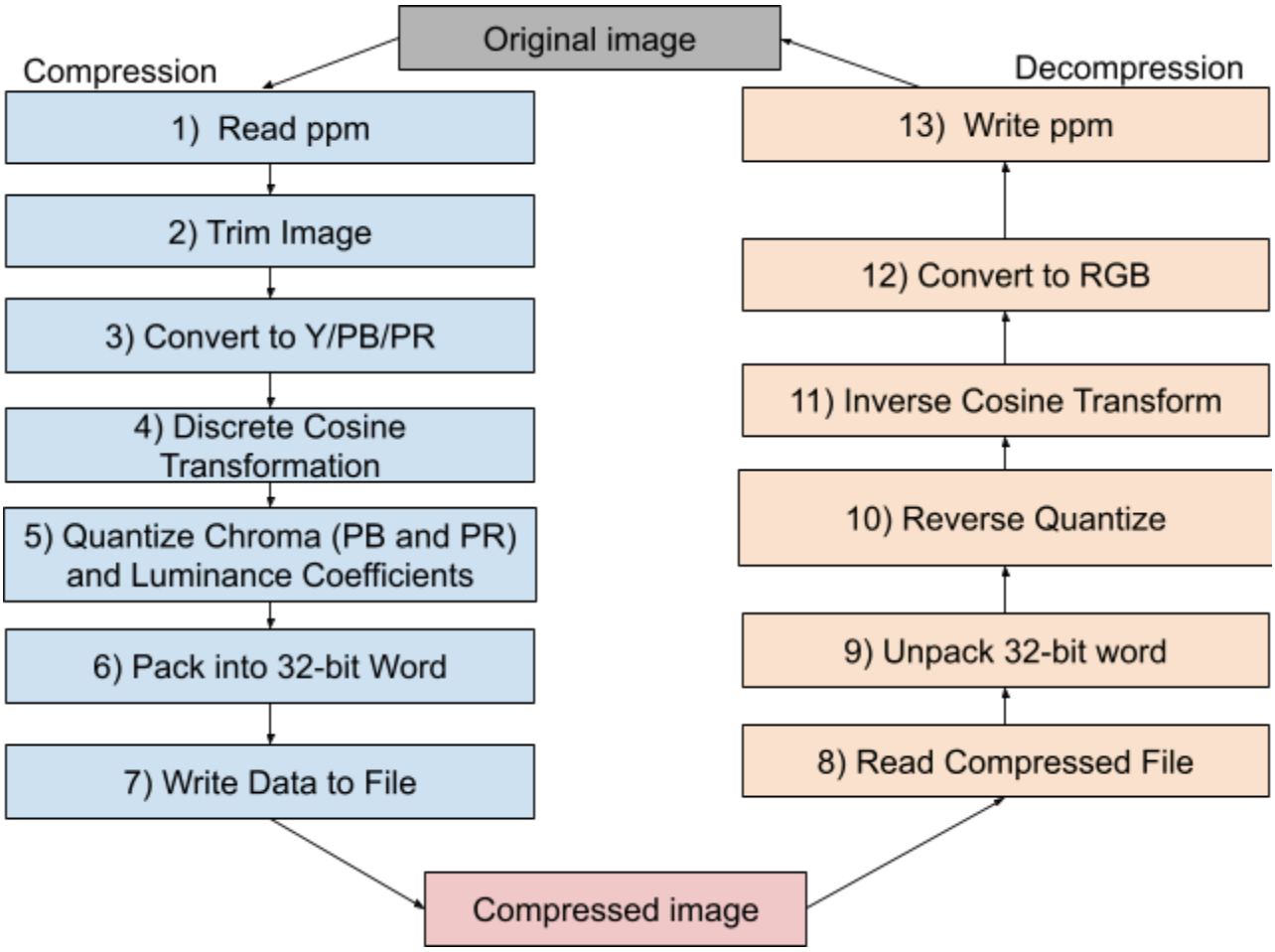
**CS40**
**Homework: Arith**
**Design Overview**

Overall Structure




Steps

Compress:

1) Read ppm image
    a) Description: read a ppm image from stdin or a file and store it into a pmn_ppm structure containing pixel data in rbg
    b) Input: a ppm file (stdin or file)
    c) Output: pnm_ppm structure with the image's width, height, and pixel data in rbg
    d) No info lost
    e) Testing:
        i) Using A2 methods map the ppm into a file. Run this through ppmdiff and make sure that they're the same
        ii) Check that this works both for stdin and through a file
        iii) file with corrupted headers or non-PPM format
        iv) small images like 1x1 and 2x2

v) ppms where the number of columns or rows are bigger the typical buffer size for large file handling

2) Trim image to even dimensions
    a) Description: check if dimensions are odd and remove last row/col to get even dimensions (important so we can get 2x2 pixel blocks)
    b) Input: a pnm_ppm w/ pixel data
    c) Output: a pnm_ppm that may have been trimmed if needed
    d) If the image is odd a row/col will be removed and will cause minor loss of image data
    e) Testing:
        i) Send a normal case through this and then compare using ppmdiff
        ii) Case with a single column
        iii) Case with a single row
        iv) Case of a single block
3) RGB to Y/PB/PR conversion
    a) Description: convert RGB values to the Y (luminance), PB, and PR (chrominance) component video format for easier compression
    b) Input: Pnm_ppm with RGB pixel data
    c) Output: 2x2 block of Y/PB/PR values for each 2x2 block of pixels in the image
    d) No info lost
    e) Testing:
        i) Try a ppm of a given size that has either some or all zero values. Is Pb and Pr still between -0.5 and 0.5?
        ii) Try r, g, and b all of negative values. Is Y still between 0 and 1?
        iii) Try values all of 255, is that still in range?
4) Cosine transformation
    a) Description: apply Discrete Cosine Transformation the Y (luminance) values of the 2x2 pixel block; will have four coefficients: a, b, c, and d, to represent different frequencies of brightness change
    b) Input: 2x2 block of Y values
    c) Output: DCT coefficients a, b, c, d
    d) Floating-point precision may cause some data loss but not major
    e) Testing:
        i) Stdout the quantized representation of Arith40_index_of_chroma(float x) (step 4b) and ensure that all output is between -0.5 and 0.5
        ii) After running through conversion, stdout the a, b, c, and d and ensure the values are between -0.3 and 0.3 with edge values being +-0.53
5) Quantize chroma (PB and PR) and luminance
    a) Description: quantize the PB, PR, and DCT coefficients for more efficient storage; PB and PR chroma values are quantized to 4-bit values and DCT coefficients b, c, and d are quantized to 5-bit signed values
    b) Input: Raw PB and PR values, and DCT coefficients
    c) Output: Quantized PB, PR, and DCT coefficients

    d) lossy compression → quantization process reduces the precision of the PB, PR, and DCT coefficients which discards image data that isn't very perceptible to us
    e) Testing: check if quantized values are in the correct ranges (PB/PR -0.5 to 0.5, DCT coefficients -0.3 to 0.3)
    f) Quantize values near the edges of the ranges and make sure that rounding errors are not significant
6) Pack into 32 bit word
    a) Description: pack quantized PB, PR, and DCT coefficients into a 32-bit word to represent one 2x2 block of the image in the compressed format
    b) Input: Quantized PB, PR, and DCT coefficients
    c) Output: A single 32-bit word for each 2x2 block
    d) No info lost
    e) Testing:
        i) Check that this can be unpacked correctly
        ii) Test with different quantized values like the minimum, maximum, and average
7) Write compressed data to file
    a) Description: write header and sequence of 32-bit words to the output file (original dimensions of image will be in header)
    b) Input: header info and 32-bit words representing image blocks
    c) Output: compressed binary file
    d) No info lost
    e) Testing:
        i) Decompress and ppmdiff and compare with original image
        ii) Check for invalid file paths and permissions

Decompress:

8) read compressed file
    a) Description: read compressed file to get dimensions from the header and get the the sequence of 32-bit words that represent  2x2 block of pixels
    b) Input: compressed binary file
    c) Output: sequence of 32-bit words representing the image blocks
    d) No info lost
    e) Testing:
        i) Write and compress, then decompress and check dimensions
        ii) Test a ppm with an odd width
        iii) Test a ppm with an odd height
        iv) Test with both odd heights and width
        v) Test with even dimensions
9) Unpack 32bit word
    a) Description: Unpack each 32-bit word back into quantized PB, PR, and DCT coefficients
    b) Input: 32 bit word
    c) Output: PB, PR, and DCT coefficients
    d) No info lost

e) Testing:
    i) The unpacked values should match the original quantized values before packing
    ii) Handle cases with corrupted 32 bit words like ones that are missing bits

10) Reverse quantize
    a) Description: convert the quantized PB, PR, and DCT coefficients back to their original floating-point values
    b) Input: PB, PR, and DCT coefficients
    c) Output: approximate PB, PR, and DCT coefficients
    d) No info lost but its an approximation bc of the info loss in the original quantization
    e) Testing:
        i) Check that the values are still in a reasonable range close to the original
        ii) Try extreme values and make sure reverse quantization doesn't produce large errors

11) inverse cosine transform
    a) Description: apply inverse of DCT to get Y (luminance) values from DCT coefficients
    b) Input: DCT coefficients
    c) Output: 2x2 block of Y values
    d) No info lost
    e) Testing:
        i) Compare the inverse DCT output with the original Y values
        ii) Input values from step 4
            (1) Stdout y values and ensure they are between 0 and 1
        iii) Test extreme Y values

12) Convert Y/PB/PR to RGB
    a) Description: convert the Y/PB/PR values back into RGB values for each pixel
    b) Input: 2x2 block of Y/PB/PR values
    c) Output: 2x2 block of RGB values
    d) No info lost but rounding errors may happen
    e) Testing:
        i) Take the numbers that step 2 outputs and run them through this function; make sure that they're the same as the originals. Those tests include:
            (1) All 0 values for the ppm
            (2) R, g, and b all as negatives
            (3) R, g, and b al, as 255

13) Write ppm image
    a) Description: write the decompressed image  in RGB format to standard output
    b) Input: pnm_ppm structure with decompressed RGB pixel data
    c) Output: PPM file representing the decompressed image
    d) No info lost
    e) Testing:
        i) Take output pnm_ppm from step 1 and print it out. Diff this with the original