Lab 2: Lossless Coding

Introduction

In this lab, explore the power of entropy-based, variable length coding (VLC) techniques. Specifically, we will focus on how to implement a Huffman encoder. You may choose to create an implementation in either MATLAB or OpenCV, to compress grayscale and colour components of an image.

The first part of this lab is for you to understand how to generate a histogram from a set of symbols present in a source (such as an audio sequence or image). You will then use the histogram to calculate the Entropy associated with a given source, and use this to assess the potential for entropy-based compression on different sources.

In the second part of this lab, you will design and construct the core elements of the Huffman coding process:

1. Using histogram results to build a binary tree for encoding the source.
2. Extracting prefix codes for each symbol from the binary tree.
3. Using the codes to encode the source into a bit stream.
4. Using the binary tree to decode the bit stream.

Ultimately, the goal is to apply this Huffman coder to an actual source (e.g. image file). However, you should pick a small test sequence to simplify the problem and build your implementation around that first. For instance, if you have trouble building the tree via a program, you may construct the tree manually, and use that tree with your function to encode/decode for parts 3 & 4.

Part I: Calculating the Entropy of a Source

1. Create a function `myEntropy(X)` that calculates relative occurrence of different symbols within given input sequence (X). The function should return a vector of probabilities corresponding to the elements in X.

2. Use your program to calculate the entropy for the following test sequence ('space' should be treated as a symbol):

'HUFFMAN IS THE BEST COMPRESSION ALGORITHM'

Don't use the built-in MATLAB function entropy().

2. Measure the entropy in at least two different image files: one grayscale, and one colour image - for the colour image, find the entropy in the chroma channels (for this you should do a colour space conversion from RGB —> YCbCr first, then apply your program to each chroma channel separately (the symbols are the intensity values). Comment on the results (what do the results signify?)

4.Calculate the compression ratio that would result for each sequence assuming a code could be found that gave this ideal entropy (compare against the original bit depth used for the sample: e.g. 8 bit grayscale image (uses 8 bpp).

Part II: Huffman Encoder

As stated in the introduction, our primary goal for this lab is to construct the various components of the Huffman Encoder, namely:

1. Use histogram to build a binary tree for encoding the source.
2. Extract prefix codes for each symbol from the binary tree.
3. Use the codes to encode the source into a bit stream.
4. Use the binary tree to decode the bit stream.

Do not use built-in MATLAB functions such as huffmandict(), huffmanenco().

Step 1:

We start by building a function that constructs the actual binary tree. The simplest way to implement a Huffman tree is through a 2D array. However, you are free to use any alternative method (e.g. pointers in C). Assuming that there are initially n symbols to encode, the size of the array should be: HuffTree[2n-1][4] (i.e. (2n-1) x 4)

Each entry (row) in the array represents one node in the tree. The first n nodes are leaf nodes (the original symbols), the remaining nodes are merged nodes (non-leaf nodes generated when we merge two existing nodes).

The 1st column indicates the weight (freq./prob.) of a given symbol (or merged node)
The 2nd column indicates the index of the nodes parent (in the tree).

The 3rd & 4th columns indicate the index of the left and right children of the current node (in the tree). A node can have a “-1” for parent/child if there is no parent/child (e.g. leaf nodes have left and right child = -1 (see figure in the next page)

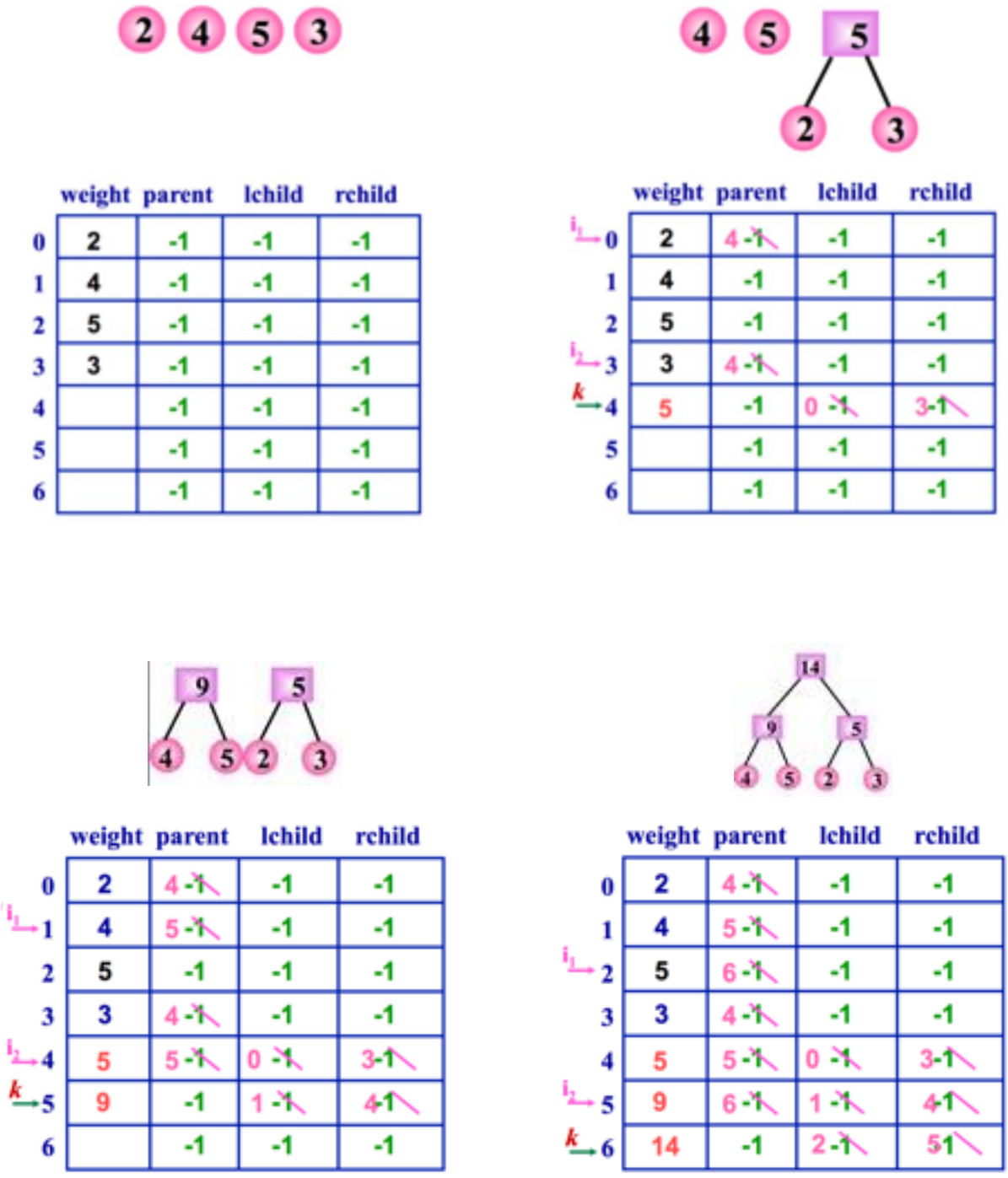

Figure 1: Huffman encoding process

Step 2:

Write a function that converts the Huffman tree to a list of code words.

Step 3:
Use the code words to encode an input sequence (e.g. use sequence from Part I initially, then apply to an image file).

Step 4:
Use the Huffman tree to decode the encoded sequence (from step 3) - note that the last node in the 2D array is always the “root” node, so you can trace the tree from here. Hint: recursion is useful here

---

In your report, you should include sample inputs and resultant outputs (e.g. if you worked on an image, show the image as a figure, and the resulting histogram/probability of intensity pixels, entropy, codebook, compression ratio achieved, etc.

Submission:

You must demo your lab to your TA during the lab session. All group members MUST be present. Submit report (PDF in IEEE format as per D2L), and source files (MATLAB *.m or OpenCV *.cpp) on your D2L submission folder. Only one member per group needs to submit. **Submit all the files, including the image files you have used.**

DEMO DUE: **Week of March 4**
REPORT DUE: **March 8 - 11.59 PM**

RUBRIC	5 (pts)	5 (pts)
	Code and Demo	Report