ECE 3790: Engineering Algorithms **Winter 2021**

Lab 4: Huffman Coding and LU Decomposition

March 22-23, 2021

The purpose of this lab is to give you practical coding experience. It will also reinforce concepts we've covered in class.

Important: A portion of your marks for this lab, and every other lab in this course, will be based on evaluation during your specified lab period. **Please be sure to attend your lab section.**

Labs can be performed in groups of two and feel free to interact with your classmates. **Code should be well documented**.

Each group is responsible for turning in a lab report that includes the answers requested at the end of each problem and a title page. The due date is 11:59pm two weeks after the lab.

Important: Please include the following signed statement with your submission.

I (We), [insert name(s)] attest that the work I am (we are) submitting is my (our) own work and that it has not been copied/plagiarized from online or other sources. Any sourced material used for completing this work has been properly cited. [Signature(s)]

Also Important: Labs done in pairs must be submitted with a short paragraph outlining each partner's contribution.

Introduction

This lab covers two class topics and allows you to reinforce concepts by implementing solutions. The lab also gives you practical experience in two important areas: 1) working from someone else's code and 2) Matlab.

Problem 1 – Huffman Coding

In class we went through the process of generating Huffman Codes for a file. The program `Lab_4_Problem_1_Huffman.py` is an incomplete implementation of a Huffman function library (encode/decode). The program uses two python modules: heapq (for the priority queue) and bitstring (for manipulating bitstrings). Your job is to complete the code.

Functions

A) Complete the function `getfilecharactercounts`. It should return a list of unique characters that occur in the file (excluding those that don't occur in the file).

B) Complete the function `createhuffmantree` by processing the min priority queue (core part of the Huffman Algorithm; see Lecture 18 for pseudocode).

C) Complete the function `huffmanencodefile`. This function already produces the bitstring for you but has not written it to file. You will need to think carefully about what else should be added to the file so that it can be decoded, and how to handle the case when the bitstring is not an even number of bytes.

D) Write the function `huffmandecodefile`. This function should read a huffman encoded file.

Important: Think carefully about what you will put in your encoded file in order to make decoding easier. We will have some friendly competition for bonus marks here (more below).

Main Program

Complete the main program. It should read the provided file, encode the file, decode the file, and verify that the file was properly decoded.

Analysis

Working on other people's code can be uncomfortable if you aren't used to it, but it is great practice and happens a lot in industry. We won't do any standard data analysis in this part of the lab, but you are to include a short writeup that:

- Provide a high-level overview of how the entire program works (key steps, associated functions).
- Explains the functionality of the `HuffmanNode` class (include a description of members and member functions).
- Explain what the `createhuffmantree`, `codehuffmantree`, and `huffmanencodefile` functions do and how they do it.

Evaluation and Discussion

Answer the following questions:

1. Explain the process you used to convert the encoded bitstring back into the original data. What information did you include in your file to enable decoding? Why did you choose the information you did?
2. What was the size of your compressed file, and what compression ratio (uncompressed:compressed size) did you achieve. Are you happy with this? Why or why not?
3. What file content would lead to the highest possible compression ratio for this coding scheme? What is the compression ratio limit of this scheme? Justify your answers.
4. Is Huffman coding used in practice? If so for what purposes. If not, why not? Support you answers with references.

Bonus

There is a single bonus available to the group/individual that submits the solution that provides the smallest compressed file. Of course the file must be 1) compressed using Huffman Coding, 2) decoded using your huffmandecodefile function, 3) not resort to semantic arguments (e.g., "you said we had to use our function to decode the file but you didn't say that function couldn't just read the file again to get the decoded file"). In essence these limitations imply that the bonus is available for the group that keep additional information in the file as lean as possible. The bonus will be worth 2% of your total *lab mark* (over all

5 labs), not to exceed 100% of the total course lab mark. We'll also crown you champions of the 2nd annual ECE 3790 Huffman Coding Challenge in class (assuming you want me to, no cash value, and no actual crown will be awarded).

Hand in:

All codes, your analysis writeup, answers to the discussion questions, and your compressed file.

Problem 2 – LU Decomposition

In this problem we will implement LU decomposition factorization and solve routines in Matlab. We are going to force you to use Matlab (or Octave) as it is good practice. We are also going to force you to use Matlab vector operations and subarray indexing (slicing) as it is also good practice.

Functions

A) Implement a function called `LUDecomposition` that, given a square matrix A, returns the LU decomposition of A. Your function should only be two nested for loops. This implies that the third for loop (over entries in a given row) should be implemented using Matlab vector operations. Recall that the ":" operator can be used to refer to or extract portions of a matrix/vector.

B) Implement a function called `LUSolve` that, given L and U and right-hand-side vector $\underline{b}$, solves the system of equations and returns the solution $\underline{x}$. Use vector operations wherever possible.

Main Program

Write a main program `Lab_4_Problem_2` that:

- Randomly generates an $n \times n$ matrix and 5 right-hand-side vectors.
- Produces the LU decomposition (time this) and solves the 5 right-hand-side vectors (time this separately). Remember the Matlab operations `tic` and `toc`.

- Validates the solution by using the built-in Matlab routine `lu` and appropriate solves (you will want to use the Matlab slash operator here - we leave it to you to figure it out). Time the built-in routines separately (like you timed your own routines).

Validation and Data Collection

Once you have completed writing your functions and main program you need to:

- Show that your functions work.
- Use your functions to collect timing data for various matrix sizes and plot the timing data compared to the theoretical complexity of the elimination and substitution steps.

Evaluation and Discussion

Answer the following questions:

1. Describe your system speed and RAM. Given your code, what is the largest system you could get into RAM and solve? Justify your answer. Assuming RAM wasn't an issue, what problem size could you solve in 1 hour? (Hint: don't actually do this.) How much RAM would that problem take?
2. Based on the timing results you obtained (for factoring the matrix and solving 5 right-hand-sides for some specific matrix size n) what timing results would you have expected to see if you applied Gaussian Elimination instead of LU decomposition? Quantify and justify your answer.

Hand in:

Hand in all code, plots, and your answers to the discussion questions.