University of Manitoba

ECE 3790: Engineering Algorithms	Winter 2020

Lab 3: Huffman Coding and LU Decomposition

March 2, 2020

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 one week after the lab.

Introduction

This lab covers two topics. While I would have preferred to only cover one topic, the conceptual reinforcement that comes with implementing algorithms is worth the pain of having two distinct topics in this lab. The lab also gives you practical experience in two important areas: 1) working from someone else's code and 2) Matlab.

Important: There is a chance this lab will also be more work than anticipated. Also, there is a midterm next week. If you find that you need an extension for this lab then do not hesitate to ask. I will *not* give extensions the day a lab is due, so plan accordingly. If you do receive an extension please keep in mind that it may mean you don't get your lab back before the next lab.

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 `codehuffmantree` and `listhuffmancodes` 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.

2. How did to make sure that your encoding/decoding worked?

3. 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?

4. What information did you include in your file to enable decoding? Why did you choose the information you did?

5. What kind of file (data/size) would lead to the highest possible compression ratio for this coding scheme? What is the compression ratio limit of this scheme? Justify your answers.

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. Oh, and I'll crown you champions of the inaugural 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 in-place 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 an in-place LU decomposition of A 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 10 right-hand-side vectors.

- Produces the LU decomposition (time this) and solves the 10 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 array 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. Having written a code to solve systems of equations, what do you now consider to be a "large" system of equations and why?

2. Was it necessary to validate your solution using Matlab? How else could you have validated your solution without using another solver?

3. Based on the timing results you obtained (for factoring the matrix and solving ten 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.

4. *In this lab you generated random matrices. Argue why it is acceptable to generate random matrices and assume that they won't be singular (i.e., that their inverse exists). This question is not worth marks but helps me understand how well you understand linear algebra concepts. Make your most educated guess if you aren't sure.

Hand in:

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

University
of Manitoba