CS 202 Assignment 1 (2021 Term 2)

Submission information: You will submit two files, one zip file for Q1 and a pdf for Q2,3,4. The Q1 zip must be named q1.zip and should contain you contain your code files and no subfolders (meaning all files directly in zip, no folder containing code inside zip). Follow the code file naming convention specified in q1.

1. (6 marks) In this assignment you are going to implement a zipping functionality. In particular, you will perform Huffman encoding and decoding for different given input files. We have given you three test input files (test_1.txt, …, test_3.txt) and we have many more (> 50) test input files at our end which will be used to test your code. Your score depends on correctness of the answers from the code (runtime is not an issue here, unless you code takes more than 1 min for any one file input).
    a. (3 marks) **Encoder**: Given the sequence of characters that you read from the input file filename.txt, implement a Huffman encoding of the characters. Save your encoding of the file in a file named filename.payload – we will test the size of this file against an expected output file. Your score depends on how small is the file size you produce (also, file size cannot be smaller than the optimal coding). Along with the above output, you must also save your encoding tree and number of total characters in a file named filename.metadata - this file will not checked but you need it for the next item below.
    b. (3 marks) **Decoder**: Taking only filename as input produce the file filename_unzipped.txt. The contents of filename_unzipped.txt should exactly match filename.txt – you must read ONLY filename.payload and filename.metadata for this decoding task. [Do not just copy content from original file into this unzipped output – we have checks for this]

   Important: You should try to follow the convention stated next or else your filename.payload will not match the least size possible of expected payload. A Huffman code produces bits which you must store in a binary form in the payload file. Here is a hint of how to do that (link). However, any file is written in chunks of bytes (8 bits) always. So, if the number of bits in your Huffman code for the input is not a multiple of 8, you need to pad with 0 bits so as to make the total number of bits a multiple of 8 and then write to file.

   This assignment will be auto-graded, that is, we will run your code using a script. Hence, it is very important to follow the naming convention we list below:

   1) You are given a file q1.py. This file has an encoder and decoder function that takes in filename (without file extension) as input; you must implement these functions. **Do not change the name of the file q1.py or the functions inside it**. You can assume that filename.txt is in the same folder.
   2) Produce the output filename.payload and filename.metadata from the code you write in encoder function and store it in the same folder as the encoder code.

3) The decoder also takes filename as input (again without any extension, you should add required extension in code). You can assume that the filename.payload and filename.metadata are in the same folder as decoder.py.
4) Produce the output filename_unzipped.txt in the same folder as the decoder code.

Summary of File Naming conventions
(for each test case, you are given test_X.txt)

5) For example, given files : test_1.txt; You call encoder(test_1) to output test_1.metadata, test_1.payload

6) Then call decoder to output test_1_unzipped.txt

Hints:

- You will need to read and write files in binary mode, for example, **open(filename+".payload", 'rb') when reading**
- Useful tool: https://hexed.it/
- You are free to use Huffman code implementation from internet (please cite reference in this case)
- If on your laptop you fail to read our provided text file with some encoding issue, try open(filename, encoding="utf-8") for reading the file. It is also very important that you use same encoding when writing the final filename_unzipped.txt file (no need of encoding for payload which is a binary file). If still does not work, contact us.

2. (1 mark) In this problem we will see how to multiply faster. Normally, we treat multiplication as one operation (meaning taking O(1) time). This is ok because the number of digits of numbers you multiply has a fixed upper bound (normally). However, for high precision computing you may be asked to multiply numbers with a large number of digits. Now, we consider the number of digits as the input size n (I am going to use base 2 numbers here). Let us see you multiply efficiently: split your n digit input numbers x, y into two halves as shown below

$$x = \boxed{\quad x_L \quad}\boxed{\quad x_R \quad} = 2^{n/2}x_L + x_R$$

$$y = \boxed{\quad y_L \quad}\boxed{\quad y_R \quad} = 2^{n/2}y_L + y_R.$$

For example, a number 10101110 is same as $2^4 * 1010 + 1110$. With this context, we can write multiplication as below

$$xy = (2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R) = 2^n\, x_L y_L + 2^{n/2}\,(x_L y_R + x_R y_L) + x_R y_R.$$

Note that the above is screaming **divide and conquer**. You have turned your multiplication of two n digit numbers into 4 multiplications of two n/2 digits numbers and additions (additions are linear time) and multiplication with $2^{n/2}$, which is very easy (just like multiplication with powers of 10 in decimal puts 0s at end, multiplication by powers of 2 is just putting 0s at the end for binary numbers). Write the recurrence relation with the information given above and then use master's method to find the complexity of the divide and conquer approach.

3. (2 marks) Continuing from the previous problem, note the following trick for the term in the bracket $x_L y_R + x_R y_L$

   $$x_L y_R + x_R y_L = (x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R.$$

   What is happening above is that on the right side we perform three multiplication, but we already compute $x_L y_L$ and $x_R y_R$ (in the recursive formula for xy in the previous problem). So, this trick has allows us to use three multiplication instead of the four for the recursive multiplication of xy from the previous problem. Use this information to write a new recurrence and use master's method to get better complexity for multiplication.

4. (1 mark) You know that Dijkstra's algorithm does not work with negative weights edges and somehow your software company knows only Dijkstra's algorithm and you have to go through 6 months of approval to be allowed to use any other single source shortest path algorithm (try to stay away from such companies!). So, you have a library from where you can call Dijkstra's algorithm dij(s, G) on a source node s and graph G. Now, you received this graph below as input from a client, unfortunately, with a negative edge and your source s is in graph G1. Describe how you can still solve this problem using just dij(s, G) (you cannot modify the function dij).