CS112: Modeling Uncertainty in Information Systems
Homework 3

Due Friday, May 18 (see below for times)

Part 1: Written Component

Due Friday, May 18, at the beginning of discussion section. You will submit this part on paper in person.

Please refer to the course academic integrity policy for collaboration rules. In particular, be sure to include a list of anyone with whom you have discussed the assignment. Remember that you will be graded on both the correctness and the clarity of your solutions. You will not receive credit for a solution if the grader can't read your writing or understand your argument. Only a subset of the problems will be graded for credit. Show all of your work. Start early!

IMPORTANT: For each problem, be sure to clearly define any events or random variables that you use in your solution!!

1. Give an example of a code for the alphabet of characters A, B, C, and D that is **not** a prefix code but is uniquely decodable. Describe an algorithm for decoding.

2. A computer system consists of n subsystems, each of which has a lifetime which follows an exponential distribution with parameter λ_i, for $i = 1, 2, \ldots, n$. Each subsystem is independent but the whole computer system fails if any of the subsystems fails. Let X be a random variable denoting the lifetime of the computer system, i.e., the time until the system fails.

   (a) Derive the CDF F_X of X.

   (b) Use your answer from part a to argue that X is an exponential random variable. What is the parameter λ of X?

3. In the morning, you arrive at the train station between 8:10 and 8:30, with all of the times in this range equally likely. If you arrive between 8:10 and 8:15, you can catch a train at 8:15. Otherwise, you can catch a train at 8:30. What is the expected time that you spend waiting for the train to arrive?

4. A program submitted for an assignment has a probability of 0.2 of going into an infinite loop. If it does not go into an infinite loop then its execution time is an exponential random variable X with mean $1/\lambda$. Suppose that the program has been executing for τ seconds. What is the probability that the program is in an infinite loop? (Find an expression in terms of λ and τ.)

5. This problem is about the performance of different scheduling schemes for read/write operations of a disk. Assume there are n independent requests for disk read/write which are uniformly distributed among all disk cylinders. For simplicity, we assume the requests follow a continuous uniform distribution over $(0, C)$, for some $C > 0$. Also we assume that initially, the read/write

head of the disk is at position 0, and after serving these n read/writes the read/write head will move back to position 0.

(a) The first scheme is to collect all the n requests, sort them according to location (cylinder), then serve the requests with one scan. With this scheme, the distance that the read/write head moves is 2 times the maximal distance from 0 among all n requests. Compute the expectation of this distance.

(b) The second scheme is *first come first serve*. With this scheme, the distance that the read/write head moves consists of 3 parts:

i. From location 0 to the location of the first job.
ii. From the location of the 1^{st} job to the 2^{nd} job, from the 2^{nd} to the 3^{rd}, ..., from the $n-1^{st}$ to the n^{th}.
iii. From the location of the n^{th} job back to location 0.

Compute the expectation of this distance (the sum of all 3 parts). Think about which scheme you might prefer when n is large. (You do not have to write this down.)

Part 2: Programming Component

Due Friday, May 18, 2pm. Please submit the code online on courseweb (details below).

For the second half of this assignment, you will write a program to construct a Huffman coding scheme. Recall that Huffman code is a prefix code that uses variable-length bit strings to encode a set of disjoint events (in this homework, events are of the form $A =$ "the outcome is character A") based on a probability distribution over those events. The goal of Huffman coding is to assign a unique bit string to every possible event, such that more probable (i.e., more frequent) events have shorter code words (bit strings) while less probable events have longer code words. This is a desirable property because the average number of bits used to represent each event (and hence any message consisting of some sequence of these events) is reduced. Huffman codes are always *prefix codes*, which means that no code word is a prefix of any other code word. Thus, when reading a bit string encoding of some message (i.e., sequence of events), it is always unambiguously obvious where one code word ends and the next begins.

Building a Huffman Tree

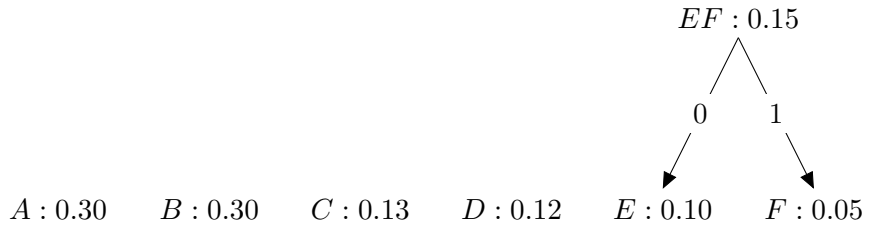
A Huffman code for a specific distribution of events can be obtained by building a binary tree (often called a *Huffman tree*). Here's an example of how the Huffman coding algorithm works.

Suppose you need to want to encode a text which only contains the characters A, B, C, D, E, F. The probability distribution of the corresponding events A, B, C, D, E, F is as follows:

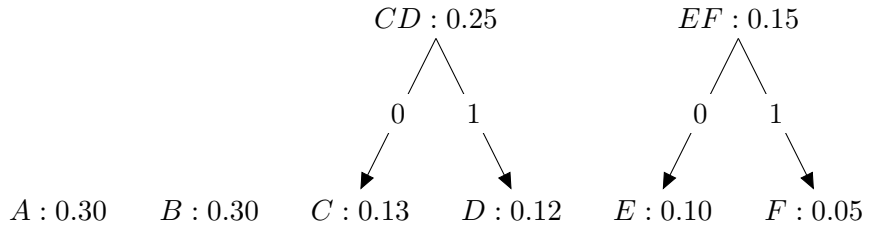
$$A : 0.30 \qquad B : 0.30 \qquad C : 0.13 \qquad D : 0.12 \qquad E : 0.10 \qquad F : 0.05$$

The first step is to find the two events with the smallest probabilities. These are E and F (with probabilities 0.10 and 0.05, respectively). We group events E and F as the children of a new node

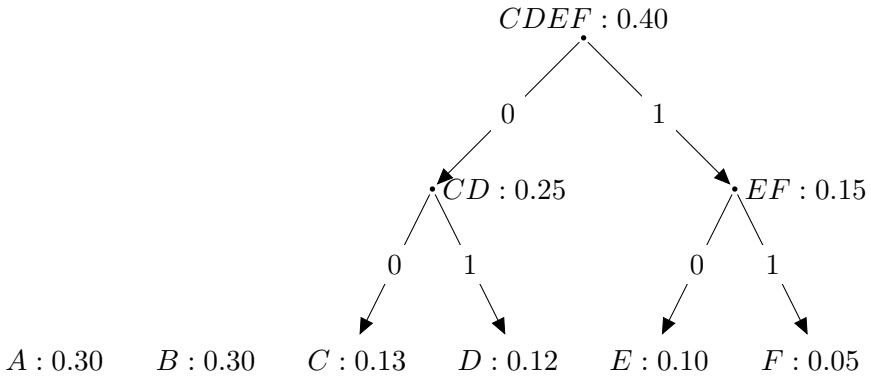
corresponding to event $E \cup F$, whose probability is equal to $P(E \cup F) = P(E) + P(F)$ since E and F are disjoint events. We also place a zero on the left branch and one on the right branch:



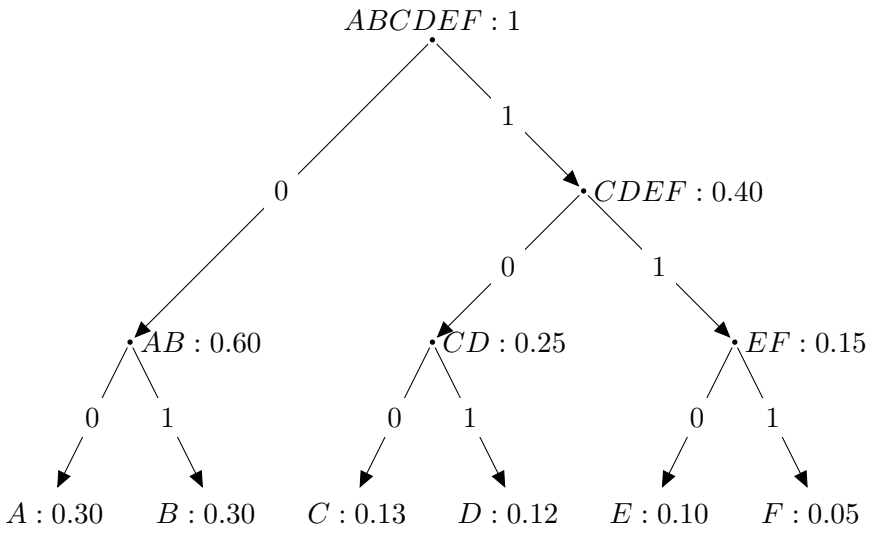
We then find the next two lowest probability events out of our remaining original events (A, B, C, D) and our new event $E \cup F$. Now the two lowest probability events are C and D, so we group them:



Next, $C \cup D$ and $E \cup F$ are combined:



The last two steps combine A and B into $A \cup B$, followed by $A \cup B$ and $C \cup D \cup E \cup F$. The final step results in a single binary tree with our original events A, B, C, D, E, F as leaves:



To determine the code word for any of A, B, C, D, E, and F, we follow the path from the root of the tree to the leaf corresponding to that event, reading off the sequence of ones and zeros on the branches:

Symbol	Probability	Code	Length
A	0.30	00	2
B	0.30	01	2
C	0.13	100	3
D	0.12	101	3
E	0.10	110	3
F	0.05	111	3

Note that a set of events and corresponding probability distribution does not necessarily have a unique Huffman code: For example, in the first step above, when we combined events E and F, we could have associated F with the right branch and E with the left. The resultant Huffman code would have been different, but the lengths of the individual bit strings would have been the same. **Please read the section "Some Important Details" below to see how you should handle this in your program.**

Writing the Program

For this assignment you will write a program that does the following:

1. Scan a text file and calculate the distribution of characters in the file.
2. Build a Huffman tree using the distribution you got from step 1.
3. Use the Huffman tree from step 2 to compress a text file and write it to the file system.
4. Decompress the file you generated in step 3 and write the decompressed file to the file system.

You are provided with skeleton code which you will need to complete. The skeleton code is available on Piazza as an attachment to the note titled "Code for Homework 3". You are required to implement the function "huffmancoding" defined in the file `huffmancoding.cpp`.

To compile the code you can invoke `g++` from the command line as follows:

```
g++ io.cpp huffmancoding.cpp main.cpp
```

Some important details

1. The skeleton code contains four functions to read and write characters and bits to and from the file system. These functions are declared in `io.h`. You are required to use these functions to do all the I/O in your program. In particular, the functions that handle I/O for bits use a particular file format. If your code produces output that does not conform to this format, the grading scripts will mark your output as incorrect.

2. You may want to create additional files to complete the program. You are allowed to do this. However, if you do so, you must submit an additional file called `linking_order.txt` that contains, in a single line, with each file name separated by spaces, the new `.cpp` file names that you have introduced, so that the following command will successfully compile your program:

   ```
   g++ io.cpp <new_files> huffmancoding.cpp main.cpp
   ```

   Where `new_files` is the text of `linking_order.txt`. If you do not do this, the grading scripts will mark your program as failing compilation, and you will not receive credit.

3. When the grading scripts compile your code, the `io.cpp`, `io.h` and `main.cpp` files that you submit will be deleted and replaced with the versions provided to you in the skeleton, so **any changes you make to those files will be ignored**.

4. There are typically many Huffman codes that correspond to a single probability distribution of characters. In particular, the left and right branches in the Huffman tree can be interpreted as representing either a 0 or a 1. Both interpretations lead to valid Huffman codes. For this assignment you are required to interpret the branch that leads to the node with lower frequency as representing a 0 and the one that leads to the node with higher frequency as representing a 1. (If both nodes have the same frequency, you can choose arbitrarily which branch is 0 and which branch is 1).

5. We will be grading your code using scripts on the SEASnet Linux machines. You may develop your code on any platform you want, but it must compile on one of the SEASnet Linux machines. You can log onto `ugrad.seas.ucla.edu` via SSH to access the SEASnet Linux machines. You may also work in the Linux lab (4405 BH) if you choose.

6. All the text files that you will be tested on will be ASCII text files. Please be aware that some .txt files might be in other formats, such as UTF-8, and these might cause you problems. Please make sure you are using ASCII files for testing.

7. Some ASCII characters might not appear in the text that you scan for the distribution of characters. You must still generate codes for them (they will have frequency 0). Even though the characters do not occur in the text you used for finding frequencies, they might occur in the text you have to compress.

8. The I/O functions defined in `io.h` represent bits using the data type `vector<bool>`. `vector<bool>` has some deficiencies and is probably going to be removed from the C++ standard (you can read more about it at http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2007/n2160.html). No good replacement is however part of the standard yet, so we will use it for this project. You will probably not run into any issues with this, but it will be a good thing to keep in mind.

Testing your code

Along with the skeleton code, in the `samples` folder, you are provided with examples of successful runs of the program against which you can test your implementation. The files `alice.hfz` and `runaways.hfz` are compressed versions of `alice.txt` and `runaways.txt` respectively, using the character distribution in the file `warandpeace.txt`. Note the differences between the file sizes of the `.hfz` files and the `.txt` files.

For some smaller examples, you can look at the file `test1.hfz`, which is the compressed version of `test1.txt` using the character distribution in the file `freq1.txt`. Similarly, file `test2.hfz` is the compressed version of `test2.txt` using the character distribution in the file `freq2.txt`.

Submitting your code

All your code for part 2 must be submitted via courseweb by the deadline. **No late assignments will be accepted**. Please submit it as a single zip file containing your code (and `linking_order.txt` if you have added additional `.cpp` files) and nothing else. All the files should be at the parent directory. You do not have to submit the `samples` directory.

Academic Integrity

The usual academic integrity policy applies for this assignment. You may discuss the assignment with anyone you like, but your code and write-up must be your own work.