CMPS 2200 Assignment 4

In this assignment we'll look at randomness in computation, and huffman coding.

To make grading easier, please place all written solutions directly in `answers.md`, rather than scanning in handwritten work or editing this file.

All coding portions should go in `main.py` as usual.

Part 1: From "Maybe" to "Definitely"

At your new job designing algorithms for really hard problems, you're put to work solving problem X. Your predecessor has left you with an algorithm $\mathcal{A}$ for problem X that has a deterministic worst-case work, but only produces the correct output with a certain probability of success. Moreover, we can also check whether the correct result was produced with $O(W(n))$ work in the worst case.

Let $\mathcal{A}(\mathcal{I})$ denote the output of an algorithm $\mathcal{A}$ on input $\mathcal{I}$. So $\mathcal{A}(\mathcal{I})$ has a probability of ϵ of being correct and a failure probability of $1-\epsilon$. Furthermore let $\mathcal{C}(\mathcal{A}(\mathcal{I}))$ denote the output of (deterministically) checking $\mathcal{A}$'s solution.

1a) You find that ϵ is too small to be reliable. You want to be able to have *any* guaranteed success probability δ, for $\epsilon < \delta < 1$. Use $\mathcal{A}$ to construct an algorithm $\mathcal{A}'$, where $\mathcal{A}'(\mathcal{I}, \delta)$ is the correct output with probability δ. It is sufficient to give a high level description of $\mathcal{A}'$. What is the work of $\mathcal{A}'$ in terms of n, δ, and ϵ? (**Hint**: Each run of $\mathcal{A}$ is independent and does not depend on previous runs.)

.

.

enter answers in `answers.md` .

.

1b) Your boss and co-workers are impressed, but you want to do even better. Show how to convert $\mathcal{A}$ into an algorithm that always produces the correct result, but has an expected runtime that depends on $W(n)$ and the success probability ϵ. .

.

enter answers in `answers.md` .

.

Part 2: Fixed-Length vs. Variable-Length Codes

In class we looked at the Huffman coding algorithm for data compresssion. Let's implement the algorithm and look at its empirical performance on a dataset of 5 text files.

2a) We have implemented a means to compute character frequencies in a text file with the function `get_frequencies` in `main.py`. Compute cost for a fixed length encoding for each text file.

2b) Complete the implementation of Huffman coding in `make_huffman_tree`. Note that we manipulate binary trees in the priority queue using the object `TreeNode`. Moreover, once the tree is constructed, we must compute the actual encodings by traversing the Huffman tree that has been constructed. To do this, complete the implementation of `get_code`, which is a typical recursive binary tree traversal. That is, given a tree node, we recursively visit the left and right subtrees, appending a `0` or `1` to the encoding in each direction as appropriate. If we visit a leaf of the tree (which represents a character in the alphabet) we store the collected encoding for that character in `code`.

2c) Now implement `huffman_cost` to compute the cost of a Huffman encoding for a character set with given frequencies.

2d) Test your implementation of Huffman coding on the 5 given text files, and fill out a table of the encoding cost of each file for fixed-length and Huffman. Fill out a final column which gives the ratio of Huffman coding cost to fixed-length coding cost. Do you see a consistent trend? If so, what is it?

enter answer in `answers.md`

2e) Suppose that we used Huffman coding on a document with alphabet Σ in which every character had the same frequency. What is the expected cost of a Huffman encoding for the document? Is it consistent across documents?

enter answer in `answers.md`