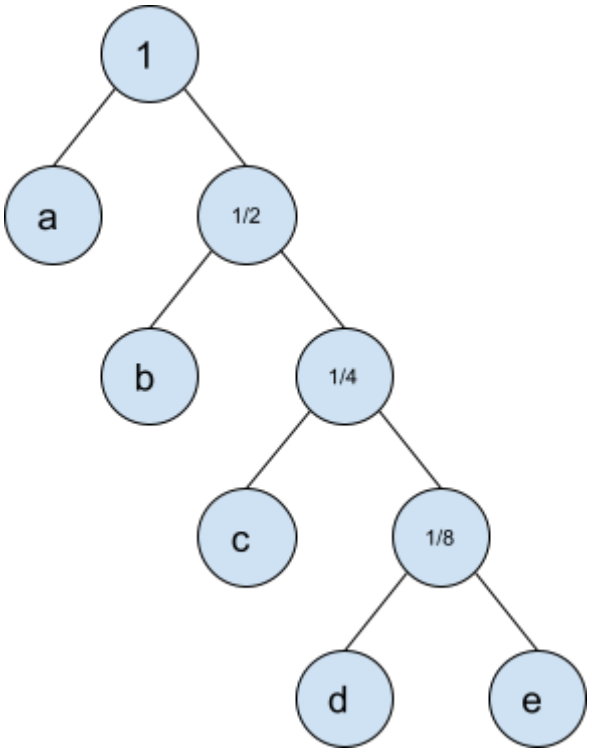
Homework 1 Report

Jeffrey Luo (jl834) and Nathan Kim (njk24)

1.

a. A: 0, b: 10, c: 110, d: 1110, 1111

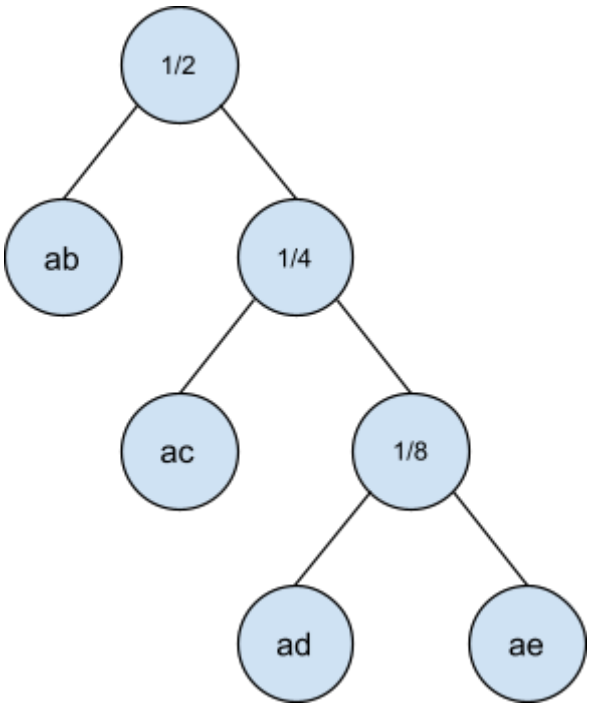


b. Number of times A appears = 1,000,000 / 2 = 500,000
Number of times B appears = 1,000,000 / 4 = 250,000
Number of times C appears = 1,000,000 / 8 = 125,000
Number of times D appears = 1,000,000 / 16 = 62,500
Number of times E appears = 1,000,000 / 16 = 62,500

Number of bits for A = 500,000 * 1 = 500,000
Number of bits for B = 250,000 * 2 = 500,000
Number of bits for C = 125,000 * 3 = 375,000
Number of bits for D = 62,500 * 4 = 250,000
Number of bits for E = 62,500 * 4 = 250,000

500,000 + 500,000 + 375,000 + 250,000 + 250,000 = **1,875,000 bits**

c.



ab: 0, ac: 10, ad: 110, ae: 111
Number of bits for AB = 250,000 * 1 = 250,000
Number of bits for AC = 125,000 * 2 = 250,000
Number of bits for AD = 62,500 * 3 = 187,500
Number of bits for AE = 62,500 * 3 = 187,500

250,000 + 250,000 + 187,500 + 187,500 = **875,000 bits**

Because we know that “A” is going to be included before any other letter that appears, we can just encode “AB” and “AC” instead of “A”, “B”, and “C” separately. This would save space by over half.

2.

a. An unsorted array implementation gives O(n) extraction time, whereas a binary heap implementation gives O(log(n)) extraction time. Insertion time for an unsorted array implementation is O(1), while insertion for a binary heap implementation is O(log(n)). Therefore, the runtime of the algorithm with an unsorted array implementation is O(mn), whereas using a heap is O(mlogn), (There are m iterations to perform on, so we multiply by the extraction time - since extraction time is the most expensive). Since m is small in relation to n, we are essentially comparing n and logn, which will result in a huge difference as n gets larger.
b. In general, a binary tree takes O(logn) lookup time. However, when a tree becomes very unbalanced, it can take O(n) time for a lookup. This can happen if all the characters have the same frequency.
c. Decoding will be faster than encoding. This is because when decoding, there are no extra steps that need to be done. For example, given the code 111, the

algorithm simply goes down the right path 3 times. However, when encoding, the algorithm needs to figure out which path to go down, resulting in extra work to be done.

3.

4.

a. Message1: 2351
   Message2: 24362
b. There are 26 unique characters in Message1, so each character in fixed length encoding would take $\log_2(26) = 5$ bits. There are 500 characters in the Message1, so Message1 will take 2500 bits.
   There are 36 unique characters in Message2, so each character in fixed length encoding would take $\log_2(36) = 6$ bits. There are 5760 characters in Message2, so Message2 will take 34560 bits.
c. Message1 had both a smaller message, and also had less unique characters.

5.

a. Have a single dictionary that will hold the characters as a key, and the number of times the character appears as the value. Loop over the text, modifying the dictionary accordingly. This will result in an O(n) runtime.
b.
   i. There are 804332 total characters in the file
   ii. There are 89 unique characters
   iii. The most common character is ‘ ’, which appears 126988 times in the file
c. Took 7.1 seconds to encode from the tree
d. Took 3.7 seconds to decode from the tree
e. Encoding with a map took 1.4 seconds, which is much faster than what happened in part c. Part c) should have had a runtime of O(nlogn) while part e) should have had a runtime of O(n). This is expected as traversal for every character takes much longer than a look up with a map because it is O(1) lookup.