CMSI 2130 – Classwork 5

Instructions:
This worksheet gives you some important practice with the fundamentals of Huffman Coding, Bloom Filters, and Edit Distance!

- Provide answers to each of the following questions and write your responses in the blanks. If you are expected to show your work in arriving at a particular solution, space will be provided for you.
- Place the names of your group members below:

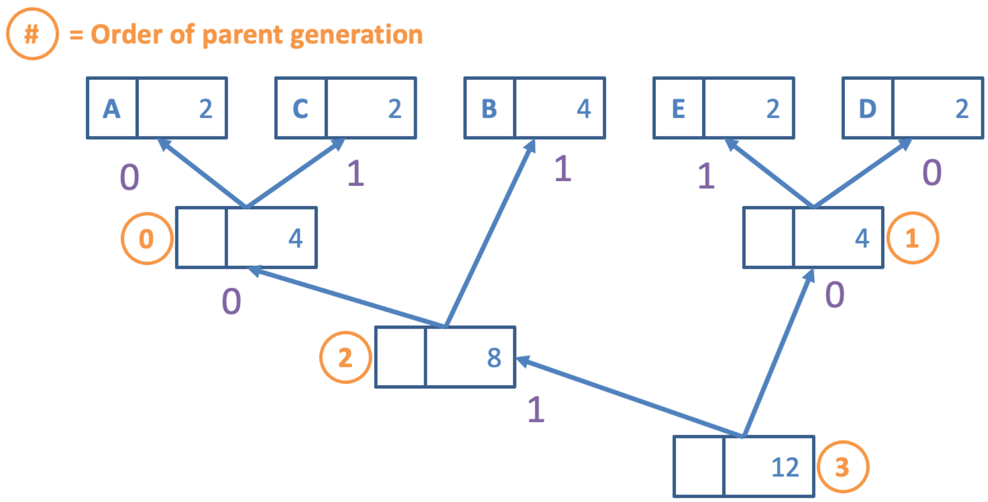
Group Members:

1. Mike Hennessy
2. Cameron Scolari
3. ______________________________

Problem 1 – Huffman Coding

As practice making *unique* Huffman Tries, we'll incorporate a tiebreaking criteria in the case that multiple nodes share the same minimal frequency. In particular: For any ties in priority, use ascending alphabetic order **<u>of earliest letter in a SUBTREE first</u>** (e.g., if a leaf node representing letter C and a parent node with children D and B have the same frequencies, the parent node will be popped first because B is earlier than C alphabetically). **<u>Nodes popped first are placed at the parent's 0-child reference and those popped second at the parent's 1-child reference.</u>**

Here's an example trie that would have been created from the text "AACCBBBBEEDD" in accordance with our tiebreaking rules. <u>Carefully study how this trie was found before going on!</u>



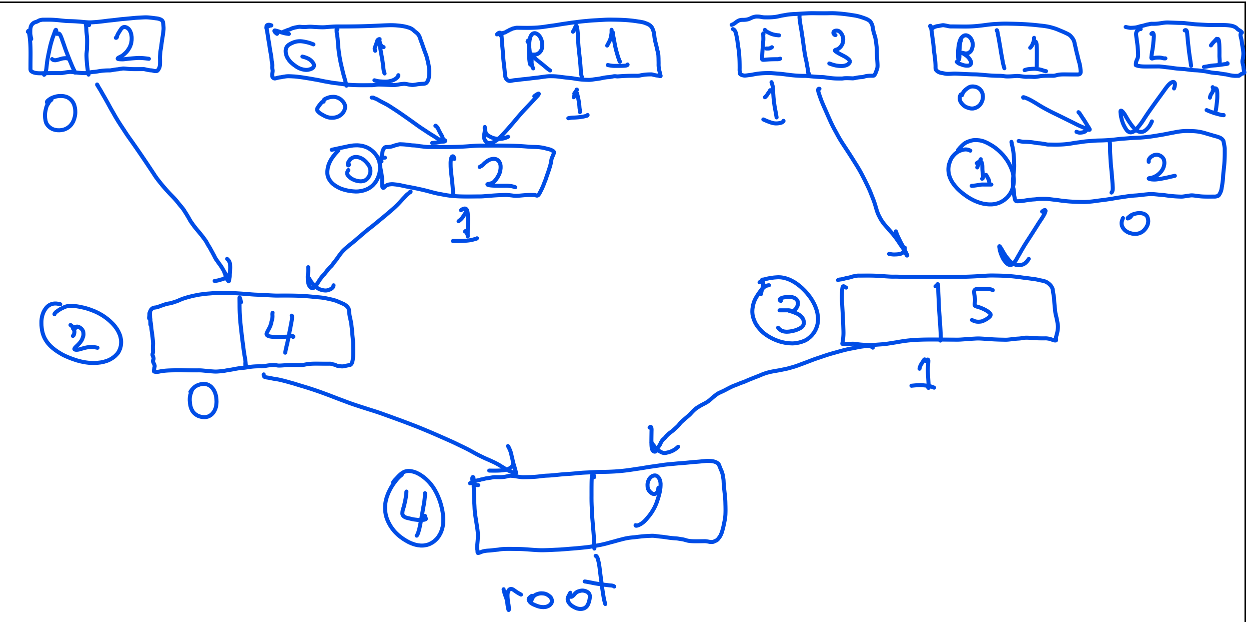
Let's walk through the steps of Huffman Coding together so you can get a solid grasp on the complete process. Your task, compress the corpus consisting of the string:

AGREEABLE

1.1. Find the frequencies of each character to encode in the corpus "AGREEABLE".

Character	Frequency
A	2
G	1
R	1
E	3
B	1
L	1

1.2. Find the Huffman Trie associated with the characters and frequencies from 1.1 above, i.e., that encodes the corpus "AGREEABLE" **<u>using the tiebreaking rules specified on the previous page!</u>**



1.3. Using the Huffman Trie you found in Part 1.2., decode the following anagram (and therefore, equivalently proportioned character distribution) to determine Andrew's favorite animal* (*may not be a real animal). This relies on having done 1.2. correctly, and a hint that they have *not* been done correctly is if each space-separated bitcode below does *not* map to a single character (e.g., 11 should be a letter, then 00, then 011, …).

11	00	011	100	11	010	11	00	101
E	A	R	B	E	G	E	A	L

1.4. Find the encoding map derived from the Huffman Trie in 1.2.

Character	New Bitcode
A	00
G	010
R	011
E	11
B	100
L	101

1.5. Provide the new bitcode associated with the compressed corpus using the encoding map from 1.4. (spacing added below for clarity and convenience).

A	G	R	E	E	A	B	L	E
00	010	011	11	11	00	100	101	11

1.6. Provide the <u>header bitstring</u> that would be used to store the Huffman Trie that you found in 1.2.; you may leave letters as placeholders for their actual ASCII encoding.

001A01G1R001B1L1E

As you know, in digital character encoding schemas, each character is represented by its bitcode, which translates to a corresponding decimal representation quite simply:

$$0100\ 1101 = 2^7 * 0 + 2^6 * 1 + 2^5 * 0 + 2^4 * 0 + 2^3 * 1 + 2^2 * 1 + 2^1 * 0 + 2^0 * 1 = 77$$

While machines need only represent these characters using their bitcodes, it is handy for us humans to be able to check their decimal representation in an encoding map, like ASCII-extended. **<u>You can use an online decimal to binary converter for this if you so choose!</u>**

Let's get some practice with that:

1.7. For each of the following characters, fill in the table that encodes each using its binary and decimal representation.

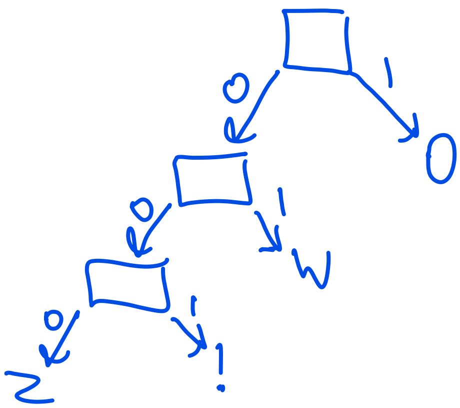
Character	Decimal Representation	Bitcode Representation
!	33	00100001
O	79	0100 1111
W	87	01010111
Z	90	0101 1010

1.8. Using the table you completed in 1.7., decode the following Huffman Trie (as may be transmitted in a header) as expressed in bitstring format. The characters at the leaves will be given in their bitcode representation (spaces added for clarity, you're welcome).

0 0 0 $\underbrace{101011010}_{90}$ $\underbrace{100100001}_{33}$ $\underbrace{101010111}_{87}$ $\underbrace{101001111}_{79}$

Z ! W O

~~0~~ ~~0~~ ~~0~~ 1 Z 1 ! 1 W 1 O



1.9. Using the Huffman Trie produced above, decode the following message (if you did the previous parts correctly, it will spell an exclamation of joy):

0111101000001

01 = W
111 = OOO
01 = W
000 = Z
001 = !

WOOOWZ!

Problem 2 – Bloom Filters

I know this is a little out of order compared to the compression lectures, but meh. Consider we have the following Bloom Filter with 16 buckets, and 3 hash functions, f_0, f_1, f_2 and are inserting the following items A, B, C, D.

Item i	$f_0(i)$	$f_1(i)$	$f_2(i)$
A	1	4	9
B	10	2	4
C	6	11	13
D	2	6	10

2.1. In the filter buckets below, assume all bits start out as 0 and are flipped to 1 when an inserted item is hashed to it. Show the state of the filter below after inserting A, B, C, D.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	0	1	0	1	0	0	1	1	1	0	1	0	0

2.2. We are now querying the Bloom Filter from 2.1. on the following items A, E, F; for each, determine if it is a true/false negative/positive and label it as such in the far right column:

Item i	$f_0(i)$	$f_1(i)$	$f_2(i)$	T/F Neg/Pos?
A	1	4	9	T Pos
E	4	6	14	T Neg
F	2	10	9	F Pos

2.3. Using the values of m, k, n for the filter completed in 2.1, determine the likelihood of a false positive *for an average* Bloom Filter with these values, showing work in the space below:

$$p = \left(1 - \left[1 - \frac{1}{m}\right]^{k \cdot n}\right)^k = \left(1 - \left[1 - \frac{1}{16}\right]^{3 \cdot 4}\right)^3 = .1566 = p$$

$n = 4 \quad m = 16 \quad k = 3$

$$\boxed{P = 15.66\%}$$

Let's label our filter from 2.1 as B_0 and another, equivalently sized, filter below as B_1, which stores items of the same type as B_0 and uses the same 3 hash functions:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	0	1	1	0	1	0	0	1	1	1	0	0	0	1

2.4. Indicate the state of a third Bloom Filter $B_2 = B_0 \cup B_1$ (i.e., the set union of B_0, B_1).

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	1	1	0	1	0	0	1	1	1	0	1	0	1

Problem 3 – Edit Distance

Let's round things off with a thorough example just doing some good, old-fashioned edit-distance using dynamic programming.

3.1. Using bottom-up dynamic-programming, compute the edit distance between the following strings found from a snake learning to type:

$$Dist(\text{"}parisss\text{"}, \text{"}parsimony\text{"})$$

$R \downarrow C \rightarrow$	Ø	P	A	R	I	S	S	S
Ø	0	1	2	3	4	5	6	7
P	1	0	1	2	3	4	6	6
A	2	1	0	1	2	3	4	5
R	3	2	1	+D 0	1	2	3	4
S	4	3	2	1	1	1	2	3
I	5	4	3	2	1	1	2	3
M	6	5	4	3	+D 2	+R 2	2	3
O	7	6	5	4	3	3 +R	3	3
N	8	7	6	5	4	4	4	+R 4
Y	9	8	7	6	5	5	5	5

3.2. Using the table generated in 3.1., determine *a single* list of transformations consisting of "R" = replacement, "T" = transposition, "I" = insertion, and "D" = deletion, with any ties broken in that same priority order that turn the row string into the column string. For example, if a cell could've been decided by both Transposition and Deletion, we'd choose Transposition because it comes earlier in the list of priorities [R, T, I, D]. Provide this in top-down order, i.e., the first transformation listed should be the one nearest the bottom-right cell of the table.

R R R D D