EECS 325 – Fall 2023
Analysis of Algorithms

Homework 3

Instructions:

- This homework is due by 11:59PM Pacific Time on **Friday, October 27th**. You will receive one grace period allowing you to submit a homework assignment up to two days late for the quarter. Subsequent assignments submitted up to two days late will incur a 20% penalty. Assignments submitted more than two days late will not be graded and will receive a score of 0. Your lowest homework score of the quarter (after applying late penalties) will be dropped.
- You must submit your written solutions as a single .pdf file on Canvas with your name at the top. Typesetting your written solutions using LaTeX is strongly encouraged. You must write your programming solutions as a single .py file. We have provided a skeleton file.
- You are welcome and encouraged to discuss the problems in groups of up to three people, but **you must write up and submit your own solutions and code**. You must also write the names of everyone in your group on the top of your submission. Students in the honors and non-honors section may collaborate.
- The primary resources for this class are the lectures, lecture slides, the CLRS and Erickson algorithms textbooks, the teaching staff, your (up to two) collaborators, and the class Piazza forum. We strongly encourage you only to use these resources. If you do use another resource, make sure to cite it and explain why you needed it.
- There are 2 questions. The questions are worth 40 points in total.
- You must justify all of your answers unless specifically stated otherwise.

Questions

Question 1. (Solving recurrences, 15 points.) The following three recurrences are master recurrences of the form $T(n) = aT(n/b) + n^c$ for integers $a \geq 1, b \geq 2$ and a number $c \geq 0$. In each case, you may assume that n is a power of b (i.e., $n = b^m$ for some integer $m \geq 0$), and that in the base case $T(1) = 1$.

Solve the following three recurrences using each of the recursion tree method and the Master Theorem. When using the recursion tree method, you must show your work. When using the Master Theorem, say which of the three cases you're using and why it applies.

a. $T(n) = T(n/3) + 10n$.

b. $T(n) = 3T(n/3) + n$.

c. $T(n) = 7T(n/2) + n^2$.

Question 2. (Implementing Huffman codes, 25 points.) Implement Huffman coding using a helper function and a `HuffmanCode` class. (You must use the same interface that we have provided.) The helper function and class constructor and methods are as follows (see also the provided skeleton source code):

- The helper function `get_frequencies(s)` takes a string `s` as input and outputs a dictionary (map) `F` whose keys are all symbols occurring in the string and whose values are the frequencies of the symbols (number of times the given symbol appears in the string). For example, if `'e'` occurs 1025 times in `s` and `'z'` occurs 7 times, `F['e'] = 1025` and `F['z'] = 7`.

  (**Hint**: This function is provided for you in the Python skeleton code.)

- The `HuffmanCode` constructor takes as input a dictionary (map) `F` of the form produced by `get_frequencies`. It produces a pair consisting of a the root node `T` of the Huffman code tree (used by `decode`) and a dictionary (map) `C` (used by `encode`). The keys of `C` are all symbols occurring in the input string `s`.

  (**Hint**: Construct `T` using the algorithm from class/CLRS, and then use BFS or DFS to build `C` from `T`.)

- The `HuffmanCode` method `encode(m)` takes as input a message `m` and uses the code map `C` to encode the message into a compressed binary message.

- The `HuffmanCode` method `decode(c)` takes as input a compressed binary message `c` and uses the tree `T` to decode it into a string.

- The `HuffmanCode` method `get_cost()` returns the cost of `T`, as defined in CLRS Equation 16.4.

  (**Hint**: You compute the cost just using `F` and `C`. You can compute the code's cost as part of the constructor.)

- The `HuffmanCode` method `get_average_cost()` returns the cost of `T` divided by the sum of all frequency values in `F`.

Note that for correctness we must have `m` = `decode(encode(m))` with respect to a fixed Huffman code object (which in turn has a fixed code map `C` and tree `T`).

Examples:
Frequency map `F` = `get_frequencies(s)`, where `s` is the text of the Gettysburg Address:

```
{'F': 1, 'o': 92, 'u': 21, 'r': 79, ' ': 274, 's': 43, 'c': 31, 'e': 165, 'a': 102,
'n': 76, 'd': 58, 'v': 24, 'y': 10, 'g': 27, 'f': 26, 't': 124, 'h': 80, 'b': 13,
'i': 65, ',': 22, 'w': 26, 'L': 1, 'p': 15, 'l': 41, 'm': 13, 'q': 1, '.': 10,
'\n': 4, 'N': 1, 'W': 2, '-': 15, 'I': 3, 'B': 1, 'T': 2, 'k': 3, 'G': 1}
```

Code map `C` in HuffmanCode object built from `F` (tree `T` not shown):

```
[space] 00          r 0100              h 0101              e 011
l 10000             s 10001             o 1001              y 1010000
G 1010001000        B 1010001001        k 101000101         I 101000110
\n 101000111        , 101001            v 101010            m 1010110
b 1010111           a 1011              w 110000            f 110001
g 110010            - 1100110           p 1100111           d 11010
i 11011             t 1110              c 111100            N 11110100000
q 11110100001       F 11110100010       L 11110100011       W 1111010010
T 1111010011        . 11110101          u 1111011           n 11111
```

Encoding example using the code map `C` (part of the HuffmanCode object built from `F`):

```
encode("oregon state rules") =
"1001010001111001010011111100100011110101111100110001001111011100000111000 1" .
```

Decoding example using the code tree `T` (part of the HuffmanCode object built from `F`):

```
decode("1100101001001010111011101110101001101001000 1") = "go beavers" .
```

Calls to `get_cost()` and `get_average_cost()` for the HuffmanCode object built from `F` should give:

$$\texttt{get\_cost()} = 6185\ ,$$
$$\texttt{get\_average\_cost()} = 4.201766\ldots$$

Note: It is possible to correctly implement the `HuffmanCode` class, but get a different code map `C` on input `F` than in the example. However, the cost and average cost of your tree should match those given above.