Arithmetic coding assignment

In this assignment you'll make a simple **Python** program that encodes/decodes some text using the arithmetic coding principle. To avoid numeric precision issues as much as possible and focus on the principle of the algorithm, we'll use the `Decimal` class (in the `decimal` module). The numeric precision (see `decimal.getcontext().prec`) will be set to 200.

The input string to be encoded will be read from a text file, and the only allowed characters are `' abcdefghijklmnopqrstuvwxyz'` (a space and the lowercase letters). A newline may be present in the input file as it is sometimes automatically added by a text file editor, but the newline (`\n` or `\r\n`) is completely ignored and will not be present in the input string that's read from the file. The maximum length of the input string that's obtained in this way, is 100 characters; an error *must* be generated when it is exceeded.

Instead of working with a true bitstream to write the encoded output in, the output will simply be a string containing the '0' and '1' characters. The number of these characters will then correspond to the number of bits used. The encoder will not only produce such a binary string, but will also output the alphabet together with the counts for each character. *The entries must be ordered in the same way as the allowed characters*. For example, if the input string is `an aardvark`, then the alphabet info will be:

```
[ [" ", 1], ["a", 4], ["d", 1], ["k", 1], ["n", 1], ["r", 2], ["v", 1] ]
```

Note that from such information, the total length of the input will also be known. Because of this, *no terminator symbol* will be used to mark the end of the input.

To aid your implementation, create a few helper functions:

- `getAlphabetFractions(alphabetInfo)`
  This function takes alphabet info like the example above as input (i.e. an array of length-two arrays), and calculates the bottom and top of the range associated with each character (make sure to use the `Decimal` class here). It returns a tuple containing a Python dictionary with this range information for each character, as well as the total number of characters.

  For the example above, this dictionary would look like

```
{' ': {'bottom': Decimal('0'),
       'top': Decimal('0.090909...')},
 'a': {'bottom': Decimal('0.090909...'),
       'top': Decimal('0.454545...')},
 'd': {'bottom': Decimal('0.454545...'),
       'top': Decimal('0.545454...')},
 'k': {'bottom': Decimal('0.545454...'),
       'top': Decimal('0.636363...')},
 'n': {'bottom': Decimal('0.636363...'),
       'top': Decimal('0.727272...')},
 'r': {'bottom': Decimal('0.727272...'),
       'top': Decimal('0.909090...')},
 'v': {'bottom': Decimal('0.909090...'),
       'top': Decimal('1')}
}
```

and the total length would be 11.

- `binStringToDecimalFraction(binStr)`
  This function takes a string of '0' and '1' characters as input and returns the `Decimal` object that corresponds to this fraction. For example, if the string `01101` is passed as its argument, this means that the binary number 0.01101 is used, which is

$$0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} + 0 \times 2^{-4} + 1 \times 2^{-5} = 0.40625$$

  The function should therefore return `Decimal('0.40625')` in this case.

The goal is to write a program `codec.py` of which the first argument can be either `test`, `encode` or `decode`, and the other arguments depend on the mode that was selected.

`test` mode

In the `test` mode, the program takes one additional argument: the name of the file containing an input string (as described above). The input read from this file is encoded using a function with signature `encodeToRange(text)`.

Apart from validating the input, this function creates the alphabet info for the string in `text`. It subsequently generates the bottom and top values (again represented as `Decimal` objects) of the interval that encodes this string, according to the arithmetic coding algorithm. The function returns a triplet `(bottom, top, alphabetInfo)`.

After processing the input using `encodeToRange`, the average of the returned interval boundaries is used as a representation of the string. Create a function called `decodeDecimalFraction(value, alphabetInfo)` that takes this value as well as the alphabet info as input, and decodes the value again into a string.

Compare the string that this `decodeDecimalFraction` function returns to the input string, and verify that it's the same.

`encode` mode

The `encode` version takes three additional arguments: the first is the name of the file containing the input string, the second is the ouput filename which will contain the binary string that encodes the message, and the last argument is the filename to which the alphabet info will be written.

You'll first need to process the input string using the `encodeToRange` function that you already wrote. The alphabet info needs to be written to the specified file in JSON format, for which you can use e.g. `json.dumps` from the `json` module. The bottom and top values of the returned interval will be used to create a binary string.

To do this, create a function `generateBinaryStringInRange(bottom, top)` that builds a binary representation (a string containing '0' and '1' characters) of a number that lies within the interval specified by 'bottom' and 'top'. The function then returns this binary string. Note that after you've generated the binary string, you can verify that it indeed lies within the range using the `binStringToDecimalFraction` function.

The binary string that's obtained this way must be written to the specified file.

`decode` mode

In the `decode` mode, there are also three additional argument: the first is the name of the file containing the binary string, the second is the filename in which the alphabet info is stored and the third is the name of the file to which the decoded string should be written.

The alphabet info in the file can be interpreted using e.g. `json.loads`, and because you've already written the `binStringToDecimalFraction` function, you can use its ouput in the `decodeDecimalFraction` function. The only thing left to do, is write the result to the specified file.

Additional remarks

1. Make sure that your implementation follows the specifications in the assignment. Verification will be done using scripts, so if, for example, your program does not use the specified arguments or in a different order, these scripts will not be able to verify that your program works.
2. Don't be fooled by thinking that if your program works on one or two short inputs, that it will work in general. Test your program on many different inputs, with many different lengths. Again, verification will be done using automated scripts, which will use many different lengths, up to the specified maximum.