Huffman Coding

Dongjing Wang

December 23, 2022

1 Functions

- `create_frequency_table`
  - **Implementation:** Python already provides dictionaries as data structures. So we only need to traverse the string and change the key and value in the dictionary according to the frequency of occurrence.
  - **Function:** This function generates a frequency table from the input string based on a dictionary.
- `build_huffman_tree`
  - **Implementation:** First, change the characters and frequencies in the frequency table into nodes and join the priority queue. Then, the two least frequent nodes are dequeued, and they are merged into a larger one and pushed into the queue. Repeat these steps when there are more than one node in the priority queue. The last remaining node is the root of the Huffman tree.
  - **Function:** The function of `build_huffman_tree` is to change the frequency table into a Huffman tree. This will provide a lot of convenience for obtaining characters and corresponding codes.
- `get_codes`
  - **Implementation:** `get_codes` uses recursion. Since the child on the left in the Huffman tree is represented as 0 and the child on the right is 1, `get_codes` uses recursion to traverse the tree and put each character and the corresponding code into the dictionary.
  - **Function:** The function of `get_codes` is to sort out the characters and corresponding frequencies in the Huffman tree and put them into the dictionary and return.
- `encode`
  - **Implementation:** Traverse the string and replace it with the corresponding code according to the dictionary.

  - **Function:** This is the last step of encoding. It is responsible for converting the string into a string of binary numbers and returning it.

- `decode_build_huffman_tree`
  - **Implementation:** First create a list of tuples, put all the nodes in it and sort them according to the binary code. The next step is to create a Huffman tree based on the code. 0 means left, 1 means right. The next step is to perform the above steps for all nodes in the list.
  - **Function:** Because the decoding process requires not only binary numbers (or strings), but also the corresponding Huffman tree. Allowing users to manually input the root of the Huffman tree may cause many problems (for example, each person's definition of the node of the Huffman tree will be different), so it is a good way to let the user input the corresponding code table that can build the Huffman tree. And this function is to convert the user's code table into a Huffman tree. It may be a little different from the previous function of building a Huffman tree with a frequency table.

- `decode`
  - **Implementation:** Traverse the entire binary number (or string), and put characters instead of code into the string according to the corresponding code (that is, the "trajectory" in the Huffman tree) (but it is also possible to create a new variable to load the string like me).
  - **Function:** This is the last step in decoding. This function is mainly to find the character according to the code in the Huffman tree and the binary number (or string) and put it into the replacement code.

- `print_tree`
  - **Implementation:** The implementation of this function is very similar to the previous implementation of `get_codes`. This function will remember the path and the nodes reached and print it directly (not store it in a dictionary like `get_codes`).
  - **Function:** This function is what I used to debug several times to build Huffman tree.

2 Design

2.1 Encoding

During the encoding process, the user can provide the text to be compressed through standard input or through a file. The encoding process is the same. First, we first build a frequency table based on the text provided by the user.

Then construct the Huffman tree according to the frequency table. After that, we get the characters and their corresponding codes from the Huffman tree. Finally, we use the corresponding code to replace the characters in the text. And after that ask the user if they want to output and save to a file.

2.2 Decoding

The input of decoding and encoding is actually similar. Both text and standard input can be accepted. After obtaining the user's input, the first step is to rebuild the corresponding Huffman tree according to the code table. After that, binary numbers (or strings) can be replaced according to characters and corresponding codes. Since the final generated text is already readable (if the original text is not garbled when compressed), it is printed directly on the standard output.