Huffman Coding

The first task was to process the initial input, the string; so I used a dictionary to store character-frequency pairs. Dictionaries have constant time insertion and access which is why I chose it.

Building the tree starts with organizing the values in nodes ordered by lowest frequency in the priority list. I used this data structure because it inserts with priority, the lowest frequency in linear speed. From here it builds the tree by removing the first two elements and adding a new tree node until the tree is the only thing left. Removal from the queue takes constant time while insertion in the tree takes O(log n). By using the huffman tree, the most common characters are nearest while the least common are further away; this data structure allows for the code assignment for the character in a way that is reversible; this traversal takes O(log n) in time complexity. It then stores the codes in a dictionary.

Encoding it would loop through the input and match the letters with the dictionary keys. To decode it would follow the path of the tree to reach the leaf node. Encoding the input takes constant time while decoding it takes O(log n). As for space, it uses a priority queue, an internal tree, and a dictionary so its complexity is linear.

Sources

Bhadaniya, S. (2021, July 2). *Heap in python: Min heap and max heap implementation*. FavTutor. From https://favtutor.com/blogs/heap-in-python

CS3 Data Structures & Algorithms. 7.18. Huffman Coding Trees - CS3 Data Structures & Algorithms. (n.d.). From https://opendsa-server.cs.vt.edu/ODSA/Books/CS3/html/Huffman.html

Kanojiya, H. (2022, October 3). *Priority queue using linked list*. GeeksforGeeks. From https://www.geeksforgeeks.org/priority-queue-using-linked-list/

Kealy, T. (2013, November 29). *Mapping a list to a Huffman tree whilst preserving relative order*. Stack Overflow. From https://stackoverflow.com/questions/20223488/mapping-a-list-to-a-huffman-tree-whilst-preserving-relative-order