PP Miniproject 2

Anders Langballe Jakobsen, alja15@student.aau.dk (#20154059)

November 2018

1 Status of the program

All the requested parts of the assignment have been completed. An edge case I found when testing is that if there is only a single unique character, there will be no bitstring and only a tree with its root being a leaf node. In my decoder, I handle it by pattern matching the tree as a leaf and the bitstring as an empty list, and then using the `replicate` function to replicate the leaf's symbol `freq` times.

In addition to the requested tasks, I have implemented file I/O. A corner I have cut here is to not save the Huffman tree in a file. As such, a file can only be decoded as long as it was encoded prior in the same session. Also, some improved error handling would be in order for this part of the miniproject.

2 Beyond string encodings

In my initial implementation my encoder and decoder functions only supported text. There is no inherent issue with this, as literature on the matter of Huffman encodings generally speak of encoding text. However, I saw no reason why this could not be extended to support lists of any type that is an instance of the `Eq` class in order to make the solution more generic. I have not put this constraint on the binary tree data type itself, but only on those functions where it is an actual requirement. Essentially, the type parameter for the binary tree can be any type, but there are cases where the type parameter is required to be equatable, For example, merging two binary trees does constraint the type parameter of the binary tree data type to be equatable, as all it does it to create a branch whose frequency is the sum of its children. On the other hand, building the encoding map requires that the key in the key-value pair is equatable. Obviously, looking up values and avoding duplicate entries would not be possible if they keys are not equatable.