Huffman Coding

- Compression technique that considers the data to be a sequence of symbols
- The algorithm uses a table of the frequencies of occurrence of the symbols to build up an optimal way of representing each symbol as a binary string.
- Each symbol representation is obtained by constructing a binary tree: Huffman Tree.
- Using Huffman Coding, high occurrence symbols have a shorter code, while lower occurrence symbols have a longer code.

Building the Huffman Tree

1. Start two empty queues *S* and *T.*
2. For each symbol create a BT (binary tree) with a single node. Each node contains the symbol's frequency as **weight**.
3. Place each BT in queue *S* in increasing order of frequency.
4. Repeat until *S* is empty and T has more than one BT
   a. Dequeue the two smallest weight BTs from *S* and *T.*
   b. Create a new BT whose root weight is the sum of the two dequeued BT's weights. Add the two dequeued BTs as children.
   c. Enqueue the new BT to *T.*

Building the Huffman Tree

1. Start two empty queues *S* and *T.* O(1)
2. For each symbol create a BT (binary tree) with a single node. Each node contains the symbol's frequency as **weight**. O(n)
3. Place each BT in queue *S* in increasing order of frequency. O(n)
4. Repeat until *S* is empty and T has more than one BT
   a. Dequeue the two smallest weight BTs from *S* and *T.* O(1)
   b. Create a new BT whose root weight is the sum of the two dequeued BT's weights. Add the two dequeued BTs as children. O(1)
   c. Enqueue the new BT to *T.* O(1)

loop is executed n-1 times: O(n)

Average Code Length and Prefix Property

Average code length

$$\sum_{s \in Symbols} (s_{probabilityOfOccurrence} * s_{codeLegth})$$

No code is a prefix of another, because all symbols are the leaves of the Huffman tree