Caijun Qin

PA2 Reflection

a) What is the computational complexity of huffman_encoder (the constructor), get_character_code, encode, and decode?

The time complexity of the Huffman_encoder is $O(n \log n)$, as building the heap from an unsorted vector constituted the longest step and performs in $O(n \log n)$ time. Here, n is the number of characters in the Huffman tree. As a brief breakdown, starting at the furthest non-leaf node N and moving towards the root, heapify is called to preserve the minimum heap property of sub-heap rooted at N. Since heapify takes $O(\log n)$ and the vast majority of nodes in the whole vector are non-leaf, $O(\log n)$ is experienced roughly n times to ensure the whole vector follows the minimum heap property. Space complexity is $O(n)$. The Huffman tree would have a maximal number of nodes in total if each character node still present in the vector merges with the same combined node until the whole tree is formed. Note that combined node refers a node without a character and is formed by merging two (2) pre-existing nodes. In the worst case, 1st leaf merges with 2nd leaf to form 1st combined node, 3rd leaf merges with 1st combined node to form 2nd combined node, 4th leaf merges with 2nd combined node to form 3rd combined node, and so on. This renders $n - 1$ combined nodes plus the original n character leaf nodes. $O(2n)$ then simplifies to $O(n)$ for space complexity. Although many new internal nodes are created, the actual node-based tree is only built as an intermediate and then destroyed after the constructor ends. Just before destruction, a traversal of the tree is completed to catalog the Huffman code for each character. Since C++ does not natively have a bidirectional map in standard library, two (2) libraries with mirrored key-value relations are used. The huffman_encoder class can find the code given the character or vice versa. Therefore, the get_character_code, encode, and decode functions each performs $O(1)$ per character in the given text, as both maps are unordered (uses hashing for lookup). Let m be the length of the given text. The functions encode and decode would each perform in $O(m)$ time.

b) What was the hardest part of the assignment?

As always, debugging was the most painstaking and/or time consuming portion. Several auxiliary functions were coded that were not part of the huffman_encoder class. Ensuring that they work seamlessly within the class's functions was another challenge on its own. Finally, applying unique_ptr for the Huffman tree was the first time I have extensively used smart pointers. This definitely took some research.

c) What did you learn from this assignment?

Figuring out the best practices and correct usages of unique_ptr was something new I learned. In a more general perspective, I learned to adapt to needs not met by the C++ standard library, such as applying a custom bidirectional map. As always, I continued to improve on debugging habits that would more effectively cut down time.