Griffin Lynch- gal69
Arbazkhan Pathan - ap1415

Systems Programming - Spring 2019
Asst2 - File Compression

Outline: This projects goal was to allow us to practice the use of the file system API, and use a multitude of data structures to help us implement our code. Our program is based on command line switches that allow the user to go through a directory structure and compute the Huffman codes of all of the files in it. Or the user may just look at the Huffman codebook which will be created after they properly run our program. We used many important function in our code that allow the program to be ran smoothly. Overall we felt that this project was quite difficult, however we got it to work successfully. We are very proud of our work, it may not be the prettiest or fastest, however it is our work and we worked very hard.

- **The Min Heap**
    - The first thing we did in our program was construct a min heap that stored all of the words and frequencies at which they occured from the file given by the user.
    - This min heap is very crucial in storing the data properly as well as it allows us to make the tree for the Huffman codes in the future.
    - The next very important implementation that we conducted on our min heap was that we created a function called “tokenize” which made every word that appeared in the file we are given into a token. The token is necessary to aid in compressing the file. Tokenize also counted the number of occurrences which will allow us to place the word in the proper spot in our Huffman tree.
    - Another very important part of our code was that we created a “heapatize” which takes all of the files words and frequencies and stores them in into the heap of min heap. This allows up to have access to all of the words and frequencies through the head.
    - The next important function in our code was “token”. In token the function took the file given to use by the user and got the tokens in order. It is very important

because when you compress a file, it must be in order. Otherwise the compression will either be incorrect or just extremely inefficient.

- **The Huffman Codes**
    - The first thing we had to do when compressing the given file using Huffman code is create a Huffman Codebook. The Huffman Codebook gets created by using the Huffman tree that we created. We inserted the tokens into the tree, and the highest frequency word was the highest up on the branches. This way the words that show up the most have the shortest frequency which is the main point of the compression algorithm.
- **Command Line Flags**
    - It was required that our program would be able to take in different flags on the command line.
    - The first line that our program is allowed to take in is the ‘b’ which allowed the user to build the codebook. The codebook will be called in the same directory that it was invoked in.
    - The next line that our program takes in is the ‘c’. The ‘c’ allows the user to compress the file indicated using the codebook created. It will go to the <filename>.hcz.
    - We also allow the user to put the ‘d’ flag which decompresses the file (.hcz one) using the codebook created.
    - The final flag that we were required to take was the ‘R’ flag which you can run alongside any of the other flags and they will run in “recursive mode”.
    - To implement this we first check the number of arguments entered. If there are less than three arguments then we throw the error “too few arguments”, if there are more than five arguments then we say “too many arguments”.
    - Next to check which flag was called we decided to use the switch statement which allowed us to to very quickly go through which case is which and what our program needs to do.
- **Time Complexity**

- There are a few main components of our code that we feel it is crucial to deduce the time complexity.
- **Min-Heap** - The time complexity of our min heap is **O(nlogn)** respectfully. This is because inserting and deleting is in O(logn) time. Luckily printing the heap is simply O(1) time.
    - After we run tokenize and heapitize on the heap however is slows down to just O(n).
- **Huffman Coding**
    - The time complexity of a huffman tree is the same of a min heap which is **O(nlogn).** Using a heap to store the weight of each tree is simply O(logn) to figure out which is a smaller frequency. We have to remember that there are O(n) iterations which result in the O(nlogn).
        - Ours slows down slightly to O(n) if the file contains an extremely large amount of data, such as a book or a very long article.

- **Space Complexity**
    - Space Complexity of an algorithm is the total space taken by the algorithm with respect to the input size. Space complexity includes both Auxiliary space and space used by input.
    - The space complexity of both the min-heap and the huffman tree is both **O(n)**. This is because both of those data structures are directly correlated to the input file and how many words are put into the heap or tree.
    - I believe that everyone's space complexity (assuming they did it right) would be O(n).