Huffman Tree For “MERCER SOCCER”:



Zip Output For “MERCER SOCCER”:

8

10 0011

32 010

67 11

69 011

77 0000

79 0010

82 10

83 0001

0000011101101110010000100101111011100011

Explanation:

The zip output above uses the Huffman encoding algorithm. Each character is represented in its ASCII form next to its encoded representation. The number at the top is the number of unique characters in the string. Below the first line, each character is in ascending order in the form of its ASCII number as you go down each line. The encoded representations follow a sequential pattern where a left child on the Huffman tree is a 0 and a right child is a 1. Each time you go down the tree to access a node, you add a 0 or 1 to the code sequence, depending on which way you go. For example, M’s encoded representation is 0000. This is because we must go four times left to access it. Finally, the last line displays the string’s overall encoded representation. Since M is 0000, we start with that. Then we append the encoded algorithm of the next character, E. E is 011, so we append the 011 to M’s code to make it 0000011. We continue this process for each character throughout the string, which is how we eventually end up with a long line of 0’s and 1’s at the bottom.

Unzip Algorithm:

The algorithm will use maps to uncompress files. The algorithm creates maps and takes in an input that then goes into a for-loop and prints out the contents of the map with an iterator j until the inputted string has been fully traversed.

Huffman Class:

**Private variables:**  
vector<HNode> nodes;

int numChars;

bool built;

**Public Methods:**

HuffmanTree();

void insert(char ch, int weight);

bool inTree(char ch);

int GetFrequency(char ch);

void build();

string GetCode(char ch);

void PrintTable();

int numNodes();