Problem 173: Morse Code

Coach's Resource Guide

Summary of Problem

Given a customized version of Morse Code, students must encode a plaintext message and decode an encoded one.

Topics Covered

- Substitution ciphers
- Huffman coding

Suggested Approach

To begin, students should read in the implementation of Morse Code provided for the test case. Since each test case will provide a different version of the encoding, this must be processed each time. Programming languages that support maps or dictionaries can store the entire code in one of those structures. For example, in Python:

```
englishToMorse = {}
morseToEnglish = {}
for idx in range(26):
  mapping = sys.stdin.readline().split(“ “)
  englishToMorse[mapping[0]] = mapping[1]
  morseToEnglish[mapping[1]] = mapping[0]
```

As each line is read in, it is split at the space separating the English letter from the Morse encoding. These are then placed within the dictionaries; for ease with both encoding and decoding, here we create two dictionaries, one for each direction.

Since the code is guaranteed to be presented in alphabetical order, however, an advanced data structure like this is not necessary. By declaring an array of strings capable of holding 26 items, the encodings can be stored in alphabetical order. For example, in C++:

```
std::string encodings[26];
for (int i = 0; i < 26; i++){
  std::string line;
  std::string encoding;
  getline(cin, line);
  line.copy(encoding, line.length() - 2, 2); // remove the letter and space
  encodings[i] = encoding;
}
```

Once populated, the array will contain all of the encodings in order from A through Z.

To perform the encoding, the characters in the plaintext string should be processed one at a time. If the character is not the first character being read in the line, three spaces should be printed first; this creates the required spacing between letters. Once that is done, if the character is a space, four more spaces should be printed to the output, thereby creating the larger gap of seven spaces between words. Otherwise, the character must be a letter that requires encoding. If a dictionary or map was used to hold the encodings, then we simply need to look up that letter within the data structure and print the associated value.

If an array was used, we have to determine which index to retrieve and print. Since individual characters are stored as numbers, we can perform numeric operations on them. In this case, we know that the array contains the encodings for the letters A through Z, such that the index for A's encoding is 0, B's encoding is 1, and so on. As a result, we can simply calculate the index by subtracting 'A' from the character we're encoding:

- 'A' - 'A' = 0
- 'B' - 'A' = 1

  ...
- 'Z' - 'A' = 25

With this index, we can quickly retrieve the appropriate encoding from the array and print it to the output.

The decoding can be conducted in much the same way. Split the encoded string at each substring of three spaces to separate individual letters. When this is done, individual decoded spaces will be represented as a string containing four spaces; letters will be any string which starts with a period or dash. If using dictionaries, the corresponding letter can be easily retrieved. If using arrays, we will have to search through the array until a matching string is found. We can then add the index number of that string to 'A' to calculate the corresponding letter; the inverse of the calculation performed above. Each letter (or space) should be printed to the output as it is identified.

Additional Background

As mentioned in the problem description, Morse Code was initially developed for use with the telegraph. Unlike many codes, it was never intended to obscure or hide information; it was developed simply to enable communication by using a simple electrical current. Early telegraphs worked by sending pulses of electric current along a wire. A receiver on the opposite end contained an electromagnet, which would generate a magnetic field in response to a current pulse. This, in turn, would cause a visual or audio effect that would be noticed by the receiving person. Since this system was binary - the only signals it could send were "on" and "off" - both sender and receiver had to agree on a system to convert patterns of "on" and "off" to meaningful text.

Artist Samuel Morse created the first standardized system for telegraphs, hence the name "Morse Code." However, there have been several different versions of Morse Code throughout history. Morse's

own original design provided only numerals; telegraphers would have to consult with a code book to determine which word was represented by the number that had been transmitted. Obviously this system was very limiting, and so engineer Alfred Vail (who had originally worked alongside Morse and physicist Joseph Henry to develop the electrical telegraph system) expanded the code. He assigned each letter - and a series of special characters - its own series of dots and dashes. This made the code much more flexible and useful.

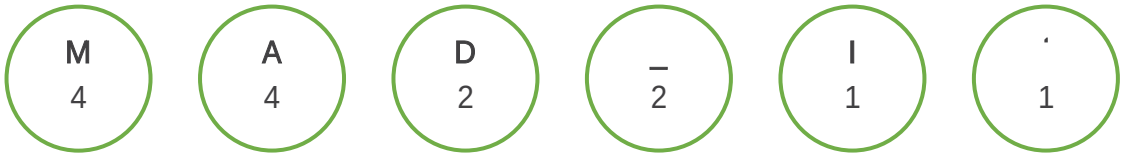
Vail's assignments we're simply random, however; he chose them quite deliberately. One of the keys to ensuring a new system gets widely adopted is convenience. Transmitting messages over a telegraph took time; there had to be clear separation between each dot and dash, to avoid confusing a sequence of dots with a single long dash. As a result, letters with lots of dots and especially lots of dashes took longer to send, and thus were less convenient.

To address this, Vail visited a local newspaper office and made a count of the number of dies for each letter in the office's movable type printing press. He assumed that the more frequently a letter was used in the English language, the more the newspaper would need to print that letter, and the more dies it would need in order to do so. He used this frequency information to inform his assignments of Morse encodings to letters; more common letters received shorter assignments; E, the most common letter, was represented with a single dot, and A, I, and O each had two signal encodings. Uncommon letters received longer encodings; J required a total of two dashes and two dots, requiring over ten times longer to transmit than a single E.

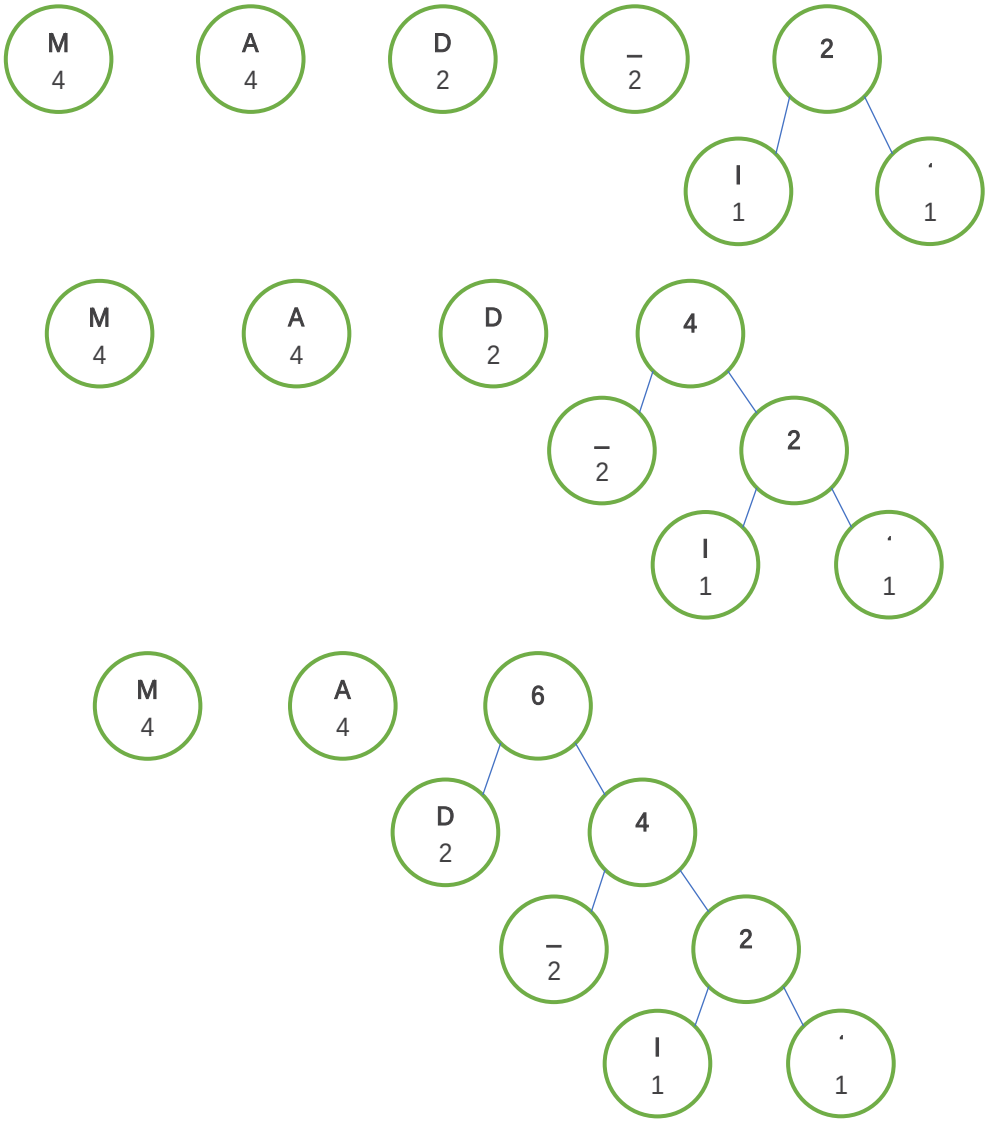
What Vail had essentially done was invented a way of compressing information. Consider a scenario in which each letter required the same amount of time to transmit; let's say an equivalent of 10 dots. Sending a simple greeting such as 'hello' would then take a total of 50 dots to transmit (not counting the 3-dot pauses between letters). However, 'hello' is comprised of very common English letters. With Vail's encodings, 'h' could be sent in the time of seven dots (four dots, plus a dot-long pause between each). As noted above, 'e' is a single dot, and 'o' was a dot-dash. 'l' required a single dash to transmit. With these encodings, 'hello' could be sent within 18 dots (again not counting pauses between letters) - a significant improvement.

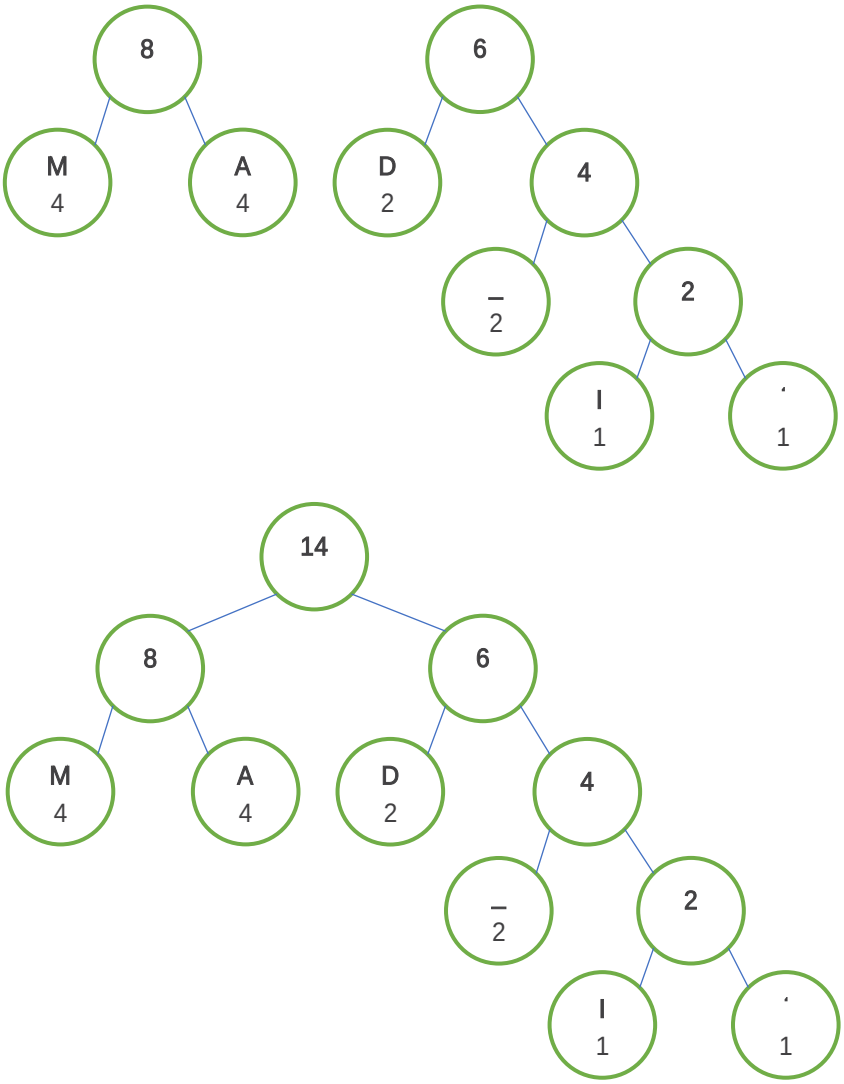
This manner of compressing information would go largely ignored until the advent of the computer; however, in 1952, David Huffman published a paper that presented an algorithm for lossless data compression. Huffman's algorithm essentially presented a way of automatically doing what Vail had done by hand over a century earlier. By constructing a binary tree based upon the frequency of individual symbols in a message, each of those symbols can be assigned a binary code that identifies that symbol. More frequently used symbols will appear higher in the tree and thus will have shorter binary codes.

Consider the phrase "MADAM_I'M_ADAM" - the phrase contains 14 characters. Using the binary ASCII values provided in the Code Quest reference materials, this could be represented with 98 bits of data. Let's see if we can improve upon that. Huffman's algorithm associates each symbol with its frequency (or probability of frequency). For our sample phrase, the frequency of each symbol is as shown:



Huffman's algorithm runs recursively until all symbols have been added to a tree. In each round, the nodes with the smallest frequencies are added as children of a new parent node. The parent node's frequency is the sum of those of its children. Any ties that occur when identifying the smallest frequency can be broken in any consistent manner. So, our tree builds itself from the nodes above like so:





Now that our tree is complete, we can use it to determine the encodings for each character in our original phrase. A left branch in the tree represents a 0 bit, and the right branch a 1 bit; the code used for each character is created by following the path from the root node to that character and noting the bits as you go. As a result, 'M' is encoded as 00, 'A' as 01, 'I' as 1110, and so on. This allows us to encode the message in just 34 bits; just over a third of what was required using normal ASCII encodings. As long as we ensure that our recipient knows the same encodings, they can use that information to restore the original message.

Further Discussion Topics

- Investigate other compression algorithms and compare them to Huffman's. What are the advantages and disadvantages of each?
- Use Huffman's algorithm to create an encoding table for the English alphabet, using the relative frequencies of each letter below. Use this information to create your own optimized version of Morse Code, exchanging 0's and 1's for dots and dashes.

Letter	Frequency	Letter	Frequency	Letter	Frequency
A	8.2%	J	0.15%	S	6.3%
B	1.5%	K	0.77%	T	9.1%
C	2.8%	L	4.0%	U	2.8%
D	4.3%	M	2.4%	V	0.98%
E	13%	N	6.7%	W	2.4%
F	2.2%	O	7.5%	X	0.15%
G	2.0%	P	1.9%	Y	2.0%
H	6.1%	Q	0.095%	Z	0.074%
I	7.0%	R	6.0%		