Lab 8: Implementing Huffman decoding

For this lab you will implement the decoding function for a Huffman-encoded string. Decoding relies on the fact that the code words in the encoding set follows the prefix code condition:

Any code word in the set cannot be a prefix for another code word.

A prefix is the leftmost number of characters (digits) in a code word. For example, the encoding set: {10, 110, 0, 111} is a valid encoding set. If we try to add 11 to the set, we would violate the prefix condition, since 11 is a prefix for both 110 and 111.

Write a function (‘Huffman_decode(encoding, message)), which takes two parameters:

1- ‘encoding’: this is a dictionary where the key is the code word (a string of 0’s and 1’s) and the value is the character that is represented by this code word.
2- ‘message’: the encoded word, a string of 0’s and 1’s

Your function does not need to perform validation on the encoding set. You can assume that the encoding was produced by a correct implementation of the Huffman coding algorithm. You should however check the given encoded message to detect errors in the encoding.

Hint: use the maximum length of a key to decide that there is an error. If a portion of the encoded message is longer than max length of key, and this portion does not exist in the encoding set, then this is an error condition.

This function should be tested with an encoding such as the following:

encoding = {'11': 'A', '100': 'B', '0': 'C', '101': 'D'}

and creating a variety of strings made up of the letters A-D. Encode the strings by hand. For example: ‘ABCD’ encodes to ‘111000101’

The code words in the encoding set should be of type string.

Keep in mind that this function will be used for testing your project 4 for correct encoding.