Input:

```
class Node:
    def create_node(char, freq):
        node = Node()  # Create an instance of Node
        node.char = char
        node.freq = freq
        node.left = None
        node.right = None
        return node

def find_two_smallest(nodes):
    min1 = min2 = float('inf')
    idx1 = idx2 = -1

    for i in range(len(nodes)):
        if nodes[i].freq < min1:
            min2, idx2 = min1, idx1
            min1, idx1 = nodes[i].freq, i
        elif nodes[i].freq < min2:
            min2, idx2 = nodes[i].freq, i

    return idx1, idx2

def build_huffman_tree(frequencies):
    nodes = [Node.create_node(char, freq) for char, freq in frequencies]

    while len(nodes) > 1:
        idx1, idx2 = find_two_smallest(nodes)
        left = nodes.pop(idx1)
        right = nodes.pop(idx2 - 1 if idx2 > idx1 else idx2)

        merged = Node.create_node(None, left.freq + right.freq)
        merged.left = left
        merged.right = right

        nodes.append(merged)

    return nodes[0]

def generate_codes(node, code, codes):
    if node is None:
        return

    if node.char is not None:
        codes[node.char] = code
        return

    generate_codes(node.left, code + "0", codes)
    generate_codes(node.right, code + "1", codes)

def encode(data, codes):
    result = ""
    for char in data:
```

```
        result += codes[char]
    return result

def decode(encoded_data, root):
    result = []
    current = root
    for bit in encoded_data:
        if bit == '0':
            current = current.left
        else:
            current = current.right

        if current.char is not None:
            result.append(current.char)
            current = root

    return ''.join(result)

def huffman_encoding_program():
    data = input("Enter the string to encode: ")

    frequency = {}
    for char in data:
        if char in frequency:
            frequency[char] += 1
        else:
            frequency[char] = 1

    frequencies = list(frequency.items())
    huffman_tree = build_huffman_tree(frequencies)

    codes = {}
    generate_codes(huffman_tree, "", codes)

    print("Huffman Codes:", codes)

    encoded_data = encode(data, codes)
    print("Encoded data:", encoded_data)

    decoded_data = decode(encoded_data, huffman_tree)
    print("Decoded data:", decoded_data)

huffman_encoding_program()
```

Output:

```
Enter the string to encode: Algorithm
Huffman Codes: {'g': '000', 'o': '001', 'r': '010', 'i': '011', 't': '100', 'h':
'101', 'm': '110', 'A': '1110', 'l': '1111'}
Encoded data: 11101111000001010011100101110
Decoded data: Algorithm
```