In [7]:

```python
import heapq

class Node:
    def __init__(self, char, probability):
        self.char = char
        self.probability = probability
        self.left = None
        self.right = None

    def __lt__(self, other):
        return self.probability < other.probability

def build_huffman_tree(char_probabilities):
    nodes = [Node(char, prob) for char, prob in char_probabilities]
    heapq.heapify(nodes)

    while len(nodes) > 1:
        left = heapq.heappop(nodes)
        right = heapq.heappop(nodes)
        parent = Node(None, left.probability + right.probability)
        parent.left = left
        parent.right = right
        heapq.heappush(nodes, parent)

    return nodes[0]

def generate_huffman_codes(root, code='', mapping={}):
    if root:
        if root.char is not None:
            mapping[root.char] = code
        generate_huffman_codes(root.left, code + '0', mapping)
        generate_huffman_codes(root.right, code + '1', mapping)

def compress_text(text, huffman_mapping):
    compressed_text = ''.join(huffman_mapping[char] for char in text)
    return compressed_text

def calculate_compression_percentage(original_bits, compressed_bits):
    return (1 - (compressed_bits / original_bits)) * 100

# Define characters and their probabilities
char_probabilities = [('A', 0.45), ('B', 0.13), ('C', 0.12), ('D', 0.16), ('E', 0.

# Build Huffman tree
huffman_tree = build_huffman_tree(char_probabilities)

# Generate Huffman codes
huffman_mapping = {}
generate_huffman_codes(huffman_tree, mapping=huffman_mapping)

# Read input from file
with open('input.txt', 'r') as file:
    original_text = file.read()

# Calculate original bits and compressed bits
original_bits = len(original_text) * 3  # Each character is encoded in 3 bits
compressed_text = compress_text(original_text, huffman_mapping)
compressed_bits = len(compressed_text)

# Write compressed text to file
with open('compressed.txt', 'w') as file:
    file.write(compressed_text)
```

```python
# Calculate compression percentage
compression_percentage = calculate_compression_percentage(original_bits, compresse

# Output results
print(f"Huffman Codes:")
for char, code in huffman_mapping.items():
    print(f"{char}: {code}")

print(f"\nOriginal Text: {original_text}")
print(f"Compressed Text: {compressed_text}")
print(f"Original Bits: {original_bits} bits")
print(f"Compressed Bits: {compressed_bits} bits")
print(f"Compression Percentage: {compression_percentage:.2f}%")
```

```
Huffman Codes:
A: 1
C: 011
B: 010
F: 0011
E: 0010
D: 000

Original Text: ABCDABCABDAEDCBFEDCFEDBAFECD
Compressed Text: 10100110001010011101000010010000011010001100100000110011001000001
0100110010011000
Original Bits: 84 bits
Compressed Bits: 81 bits
Compression Percentage: 3.57%
```