데이터압축 3장 과제물

Huffman Coding

2019112110 이정민

-Language: python -

먼저, Huffman code는 주어진 문자열 또는 데이터의 각 심볼에 대해 빈도수 또는 확률에 기반하여 가변 길이의 코드를 할당하는 방식으로, 더 자주 등장하는 심볼에는 짧은 코드를, 덜 자주 등장하는 심볼에는 긴 코드를 할당하여 전체적인 데이터의 표현 길이를 최소화한다.

기본 Huffman 코딩은 전체 코드 길이의 최소화에 중점을 두지만, 때로는 생성된 코드들의 길이 분산도 중요한 고려 대상이 될 수 있다. 분산이 큰 경우, 데이터 처리 시 성능 저하가 발생할 수 있으며, 특히 실시간 시스템이나 네트워크 통신 등 일정한 처리 속도가 요구되는 환경에서 문제가 될 수 있다.

Code

huffman code 2019112110 이정민.py > ...

```python
#동국대학교 정보통신공학과 2019112110 이정민
import heapq
from collections import Counter, defaultdict

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

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

```

```python
def build_huffman_tree(text):
    if len(text) == 0:
        return None

    frequency = Counter(text)
    priority_queue = [Node(char, freq) for char, freq in frequency.items()]
    heapq.heapify(priority_queue)

    while len(priority_queue) > 1:
        left = heapq.heappop(priority_queue)
        right = heapq.heappop(priority_queue)

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

        heapq.heappush(priority_queue, merged)

    return priority_queue[0]
```

```python
def encode_huffman_tree(root, current_code, codes):
    if root is None:
        return

    if root.char is not None:
        codes[root.char] = current_code
        return

    encode_huffman_tree(root.left, current_code + "0", codes)
    encode_huffman_tree(root.right, current_code + "1", codes)

def huffman_encoding(text):
    root = build_huffman_tree(text)
    codes = {}
    encode_huffman_tree(root, "", codes)
    encoded_text = ''.join(codes[char] for char in text)
    return encoded_text, codes
```

```python
def huffman_decoding(encoded_text, codes):
    reverse_codes = {value: key for key, value in codes.items()}
    current_code = ""
    decoded_text = ""

    for bit in encoded_text:
        current_code += bit
        if current_code in reverse_codes:
            decoded_text += reverse_codes[current_code]
            current_code = ""

    return decoded_text
```

```python
def calculate_variance(codes):
    code_lengths = [len(code) for code in codes.values()]
    avg_length = sum(code_lengths) / len(code_lengths)
    variance = sum((length - avg_length) ** 2 for length in code_lengths) / len(code_lengths)
    return variance
```

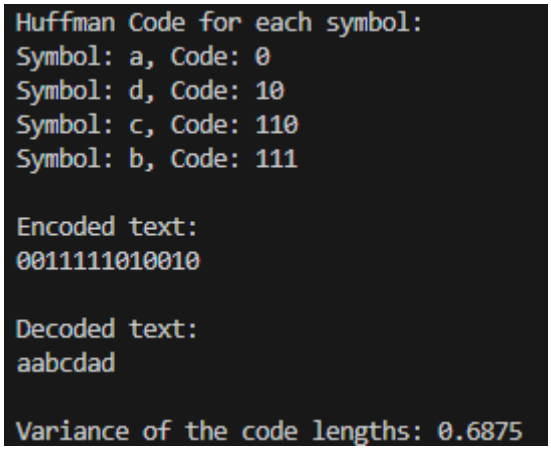
```python
if __name__ == "__main__":
    input_string = "abcdea"
    encoded_text, codes = huffman_encoding(input_string)
    print("Huffman Code for each symbol:")
    for symbol, code in codes.items():
        print(f"Symbol: {symbol}, Code: {code}")

    print("\nEncoded text:")
    print(encoded_text)

    decoded_text = huffman_decoding(encoded_text, codes)
    print("\nDecoded text:")
    print(decoded_text)

    variance = calculate_variance(codes)
    print(f"\nVariance of the code lengths: {variance}")
```

문자열을 입력 받고 문자열에서의 각 문자의 빈번도를 기반으로 확률을 측정한 뒤 Huffman 코딩을 하게끔 코드를 작성했다.

EX: encode $\{a, a, b, c, d, a, d\}$

이 예제의 문자열을 넣고 시뮬레이션 해봤다.

-출력화면-



빈도가 가장 높은 a가 가장 짧은 codeword를 갖고

빈도가 가장 낮은 b, c가 가장 긴 codeword를 갖는다

Encoded text, Decoded text, variance 또한 확인할 수 있다