Huffman Encoding

20.01.29

DeepByun Study

Huffman Encoding

- 무손실 데이터 압축 알고리즘 (Lossless data compression algorithm)
- 입력된 문자열에서 각 문자의 빈도수에 따라 다른 길이의 코드를 부여 (variable-length code)
- 입력 문자에 배정된 코드(variable-length code)는 Prefix Code

  - Prefix code란?

    : 고유하게 디코딩 가능한 코드

    Ex) {0, 11} Prefix code

    {0, 1, 11} Non Prefix code

  - Lossless data compression

    : 압축된 데이터로부터 원본 데이터를 정보의 손실 없이 복구 가능

Huffman Encoding

1. 주어진 텍스트에서 각 문자의 출현 빈도수를 구한다.

2. 가장 작은 빈도수를 가지는 노드 2개를 합쳐 부모 노드를 만든다.

3. 위를 노드가 1개 남을 때까지 반복

4. 완성한 트리의 왼쪽 간선에는 0, 오른쪽 간선에는 1

5. 트리의 각 leaf node가 압축하고자 하는 문자가 되며,

root node에서 leaf node까지의 간선들을 합한 것이 해당 문자의 허프만 코드.

Huffman Encoding

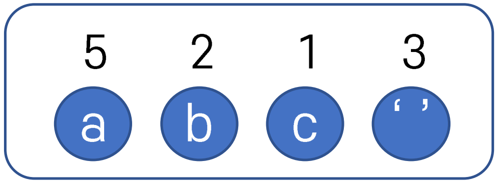
1. 주어진 텍스트에서 각 문자의 출현 빈도수를 구한다.

예) “abc ab a aa”

Huffman Encoding

1. 주어진 텍스트에서 각 문자의 출현 빈도수를 구한다.

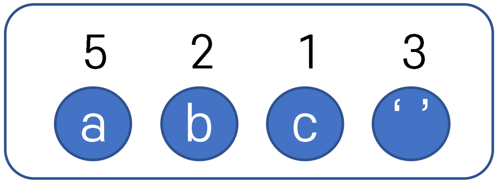
예) “abc ab a aa”



Huffman Encoding

1. 주어진 텍스트에서 각 문자의 출현 빈도수를 구한다.

예) “abc ab a aa”

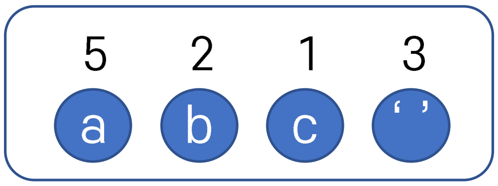


2. 가장 작은 빈도수를 가지는 노드 2개를 합쳐 부모 노드를 만든다.

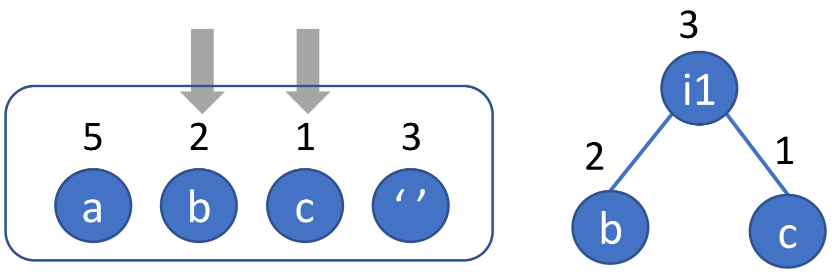
Huffman Encoding

1. 주어진 텍스트에서 각 문자의 출현 빈도수를 구한다.

예) “abc ab a aa”



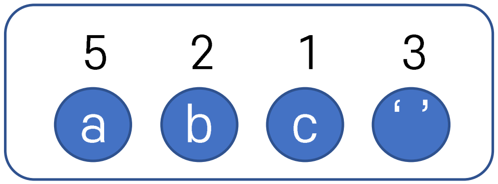
2. 가장 작은 빈도수를 가지는 노드 2개를 합쳐 부모 노드를 만든다.



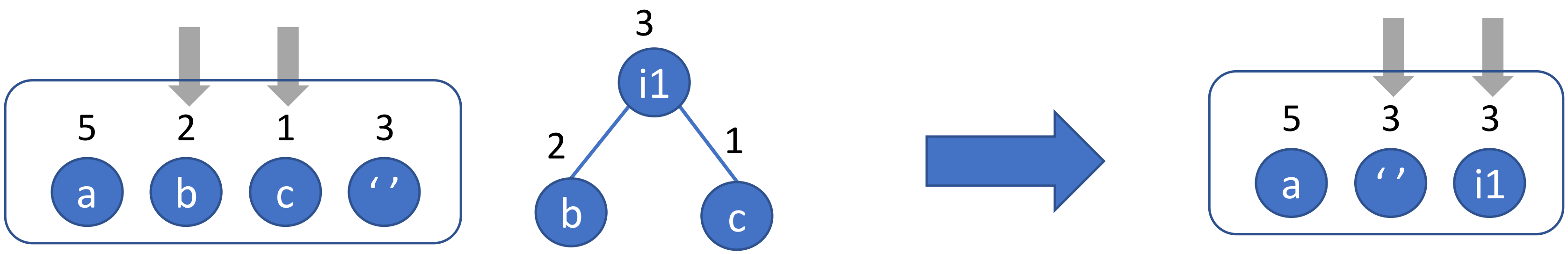
Huffman Encoding

1. 주어진 텍스트에서 각 문자의 출현 빈도수를 구한다.

예) “abc ab a aa”

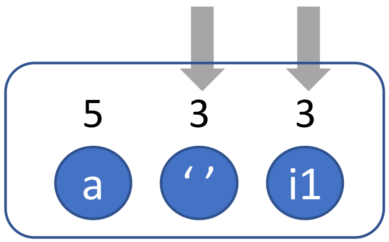


2. 가장 작은 빈도수를 가지는 노드 2개를 합쳐 부모 노드를 만든다.



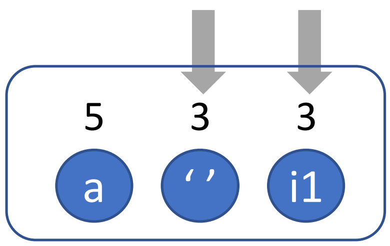
Huffman Encoding

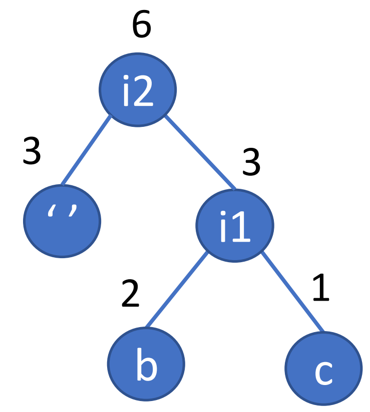
3. 위를 반복해 노드가 1개 남을 때까지 반복



Huffman Encoding

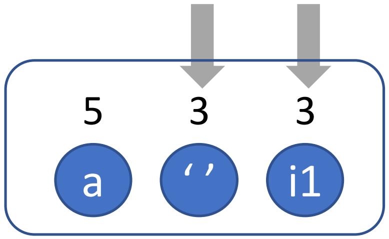
3. 위를 반복해 노드가 1개 남을 때까지 반복

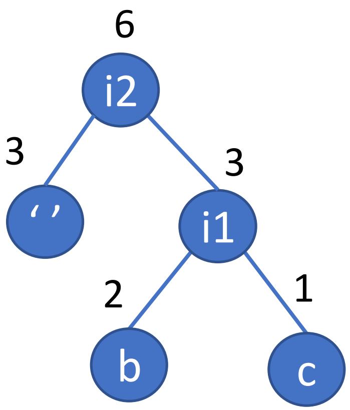


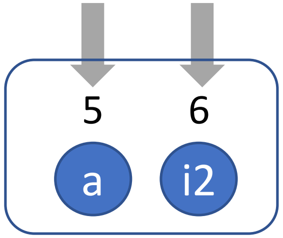


Huffman Encoding

3. 위를 반복해 노드가 1개 남을 때까지 반복

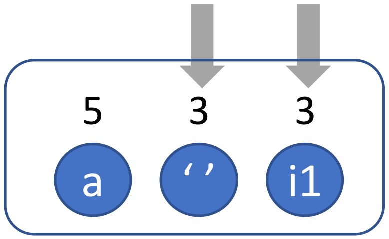


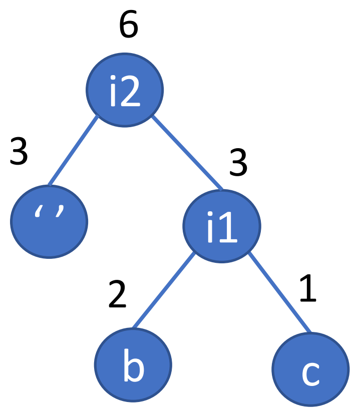


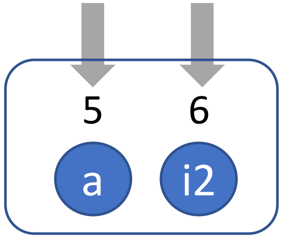


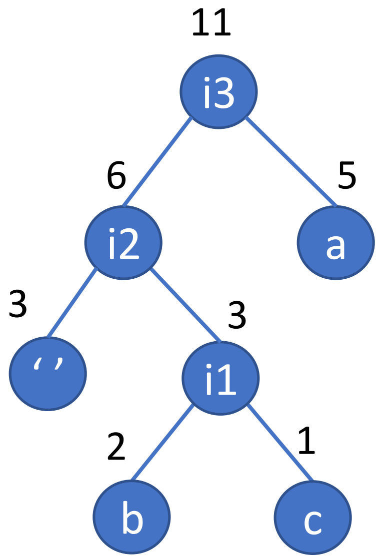
Huffman Encoding

3. 위를 반복해 노드가 1개 남을 때까지 반복







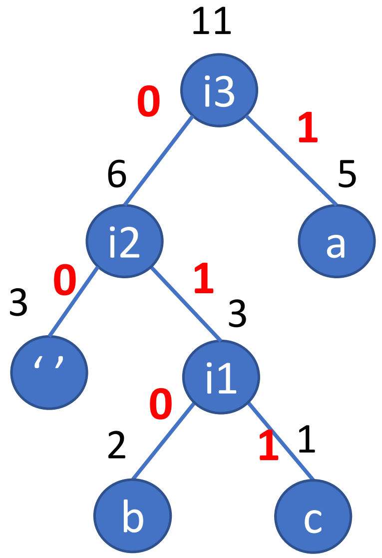


Huffman Encoding

4. 완성한 트리의 왼쪽 간선에는 0, 오른쪽 간선에는 1

5. 트리의 각 leaf node가 압축하고자 하는 문자가 되며,

root node에서 leaf node까지의 간선들을 합한 것이 해당 문자의 허프만 코드.

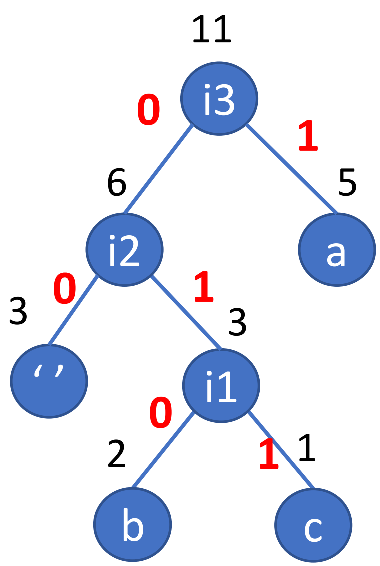


Huffman Encoding

4. 완성한 트리의 왼쪽 간선에는 0, 오른쪽 간선에는 1

5. 트리의 각 leaf node가 압축하고자 하는 문자가 되며,

root node에서 leaf node까지의 간선들을 합한 것이 해당 문자의 허프만 코드.



a: 1
‘ ’: 00
b: 010
c: 011

Huffman Encoding

- 압축 결과

"abc ab a aa"

a:	1
' ':	00
b:	010
c:	011

Huffman Encoding

- 압축 결과

"abc ab a aa"

a: 1
' ': 00
b: 010
c: 011

a	b	c		a	b		a		a	a
1	010	011	00	1	010	00	1	00	1	1

- ASCII Code: 88 (8 bits × 11 symbols) bits are required to encode the text
- Compression Code: Only 20 bits are required

Entropy via Huffman Code

확률 분포의 고유한 **예측 불가능성** 또는 **무작위성**

→ 해당 분포에서 **도출된 데이터를 압축할 수 있는 정도에** 따라 측정 가능

more compressible ≡ less random ≡ more predictable

나올 수 있는 값이 n개라고 하면 각 확률은 p1,p2,...,pn.

분포에서 m개의 값이 추출될 경우, i번째 값은 약 mPi 번 뽑힐 것이다. (m이 충분히 클 경우)

number of bits needed to encode the sequence is:

$$\sum_{i=1}^{n} m p_i \log\left(\frac{1}{p_i}\right) \qquad (1)$$

Entropy via Huffman Code

Thus the average number of bits needed to encode a single draw from the distribution is:

$$\sum_{i=1}^{n} m p_i \log\left(\frac{1}{p_i}\right) \tag{1}$$

$$\sum_{i=1}^{n} p_i \log\left(\frac{1}{p_i}\right) \tag{2}$$

entropy - measure of how much randomness it contains

Entropy via Huffman Code

Entropy(S) is the **expected number of bits** needed to encode classes of randomly drawn member of S can be derived from the Huffman Code algorithm as follows:

$$S \cong \sum n_i \log_2 n(i) \cong \sum n_i \log_2 \frac{n(S)}{n_i}$$

$$E(S) = \sum \frac{n_i}{n(S)} \log_2 \frac{n(S)}{n_i} = \sum p_i \log_2 \frac{1}{p_i} \quad (\because p_i = \frac{n_i}{n(S)})$$

n_i : number of occurrence of a symbol s_i
$n(S)$: number of entire symbols
$n(i)$: number of symbol types