파이썬으로 배우는 알고리즘 기초

Chap 4. 탐욕 알고리즘

4.4

허프만 코드와 허프만 알고리즘

4.4 허프만 코드와 허프만 알고리즘

- **이진 코드** (binary code)
  - **데이터 파일을 이진 코드로 인코딩하여 저장 (압축 파일 등)**
    - **길이가 고정된** (fixed-length) **이진 코드**
    - **길이가 변하는** (variable-length) **이진 코드**

  - **만약, 사용하는 문자의 집합이** {a, b, c}**라면?**

a: 00	a: 10
b: 01	b: 0
c: 10	c: 11

4.4 허프만 코드와 허프만 알고리즘

File = "ababcbbbc", $S = \{a, b, c\}$

Character	Fixed-lengh Binary Code
a	00
b	01
c	10

Character	Variable-length Binary Code
a	10
b	0
c	11

a	b	a	b	c	b	b	b	c
00	01	00	01	10	01	01	01	10

- It takes 18 bits with this encoding

a	b	a	b	c	b	b	b	c
10	0	10	0	11	0	0	0	11

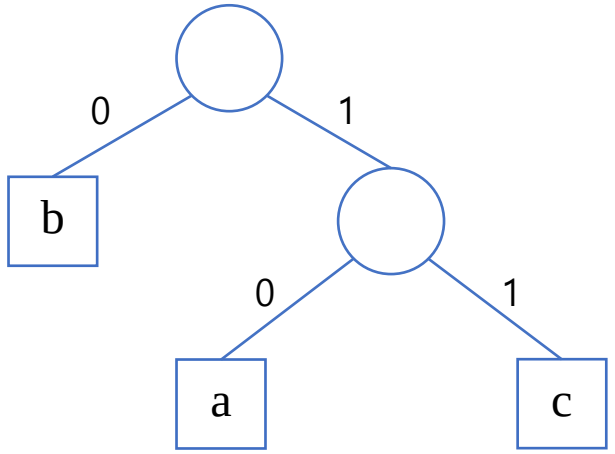
- It takes 13 bits with this encoding
- 'b' occurs most frequently:
  - Encode 'b' with one bit (0)
- Encode 'a' and 'c' starting with 1
  - 'a': 10, 'c': 11

4.4 허프만 코드와 허프만 알고리즘

- **최적 이진코드 문제** (optimal binary code)
  - 주어진 파일에 있는 문자들을 이진코드로 표현할 때
  - 필요한 비트의 개수가 최소가 되는 이진코드를 찾아라.

4.4 허프만 코드와 허프만 알고리즘

- **전치 코드** (prefix code)
  - 길이가 변하는 이진코드의 특수한 형태
  - 한 문자의 코드워드가 다른 문자의 코드워드의 앞부분이 될 수 없다.
    - 예) 01이 'a'의 코드워드라면 011은 'b'의 코드워드가 될 수 없다.
  - 전치코드의 표현: 모든 전치코드는 리프노드가 코드문자인 이진트리로 표현 가능
  - 전치코드의 장점: 파일을 파싱(디코딩)할 때 뒷부분을 미리 볼 필요가 없다.

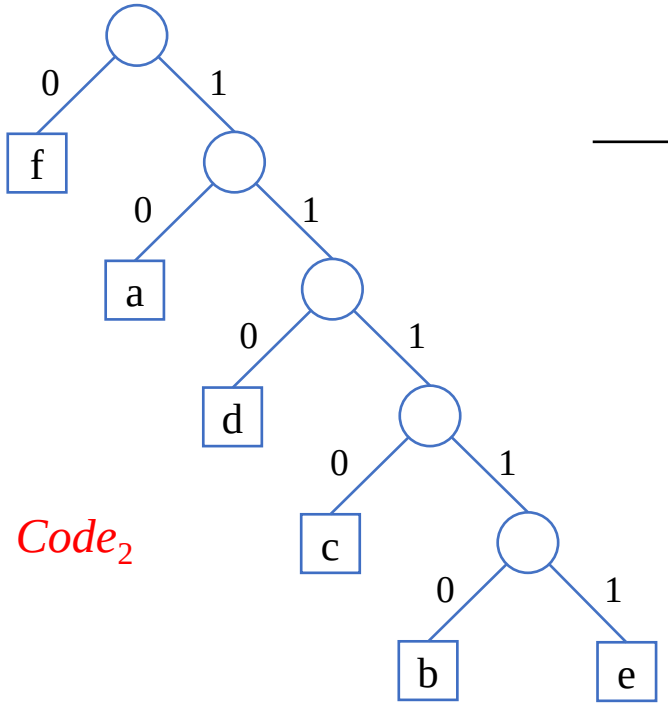


- b: 0
- a: 10
- c: 11

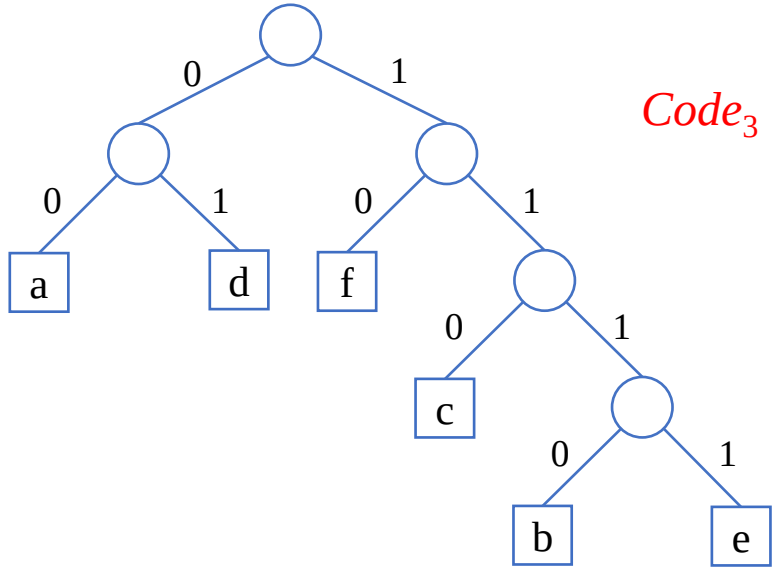
4.4 허프만 코드와 허프만 알고리즘

$$S = \{a, b, c, d, e, f\}$$

Character	Frequency	$Code_1$	$Code_2$	$Code_3$
a	**16**	000	10	00
b	**5**	001	11110	1110
c	**12**	010		
d	**17**	011		
e	**10**	100		
f	**25**	101		



$Code_2$



$Code_3$

4.4 허프만 코드와 허프만 알고리즘

- **허프만 코드** (Huffman's code)
  - **허프만 알고리즘에 의해 생성된 최적 이진코드**

- **허프만 알고리즘** (Huffman's algorithm)
  - **허프만 코드에 해당하는 이진트리를 구축하는 탐욕 알고리즘**

4.4 허프만 코드와 허프만 알고리즘

- 최적 이진코드를 위한 이진트리의 구축
  - 데이터 파일을 인코딩하는 데 필요한 비트의 수 계산
  - 주어진 이진트리를 T라 하자.
  - $\{v_1, v_2, \cdots, v_n\}$: 데이터 파일 내의 문자들의 집합
  - $frequency(v_i)$: 문자 v_i 가 파일 내에서 나타나는 빈도수
  - $depth(v_i)$: 이진 트리 T에서 문자 v_i 노드의 깊이

$$bits(T) = \sum_{i=1}^{n} frequency(v_i)depth(v_i)$$

- $\text{bits}(T_2) = 231$
- $\text{bits}(T_3) = 212$

4.4 허프만 코드와 허프만 알고리즘

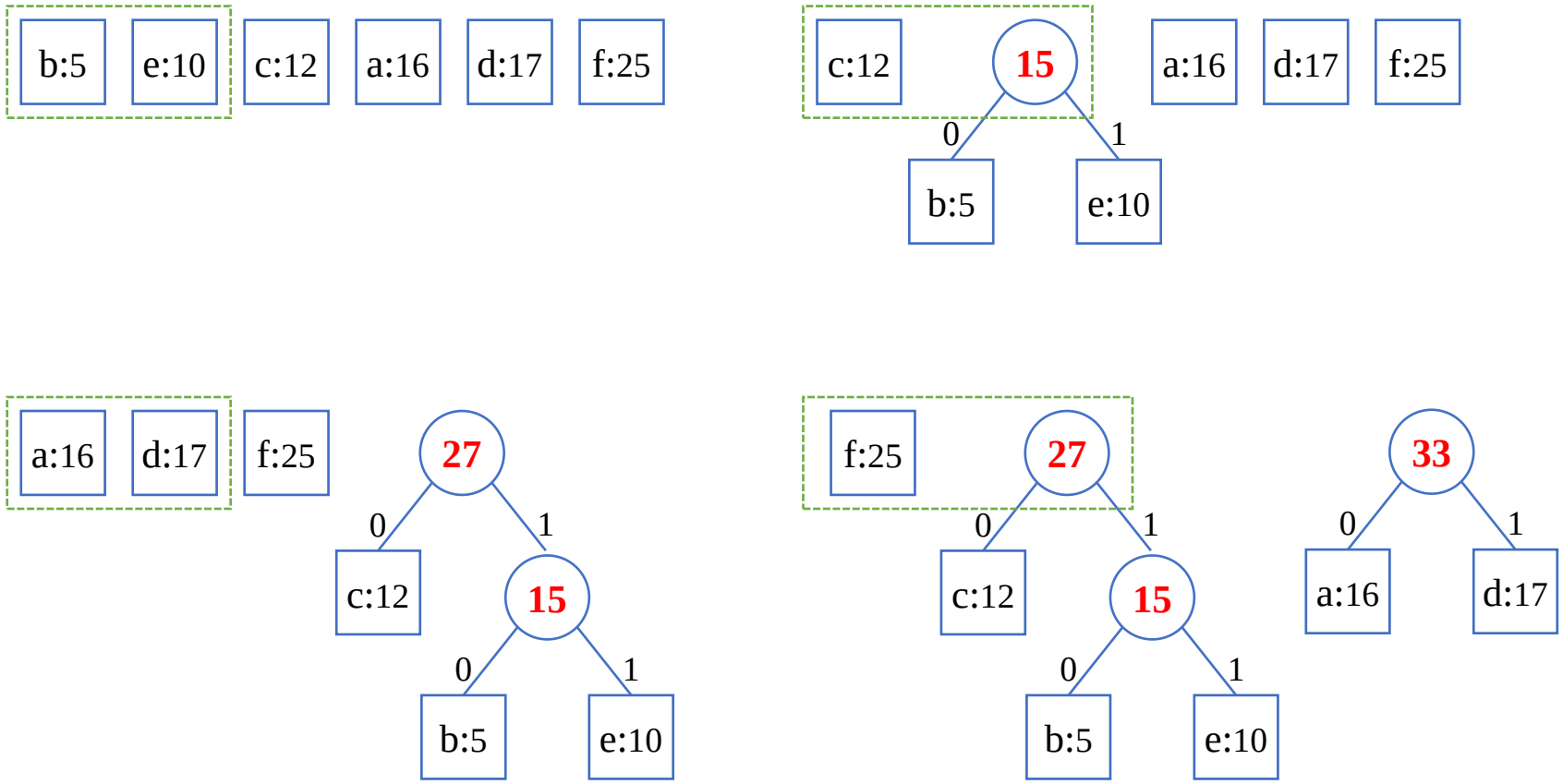
- **허프만 알고리즘: 탐욕 알고리즘(The Greedy Approach)**

  n: 주어진 데이터 파일내 문자의 개수

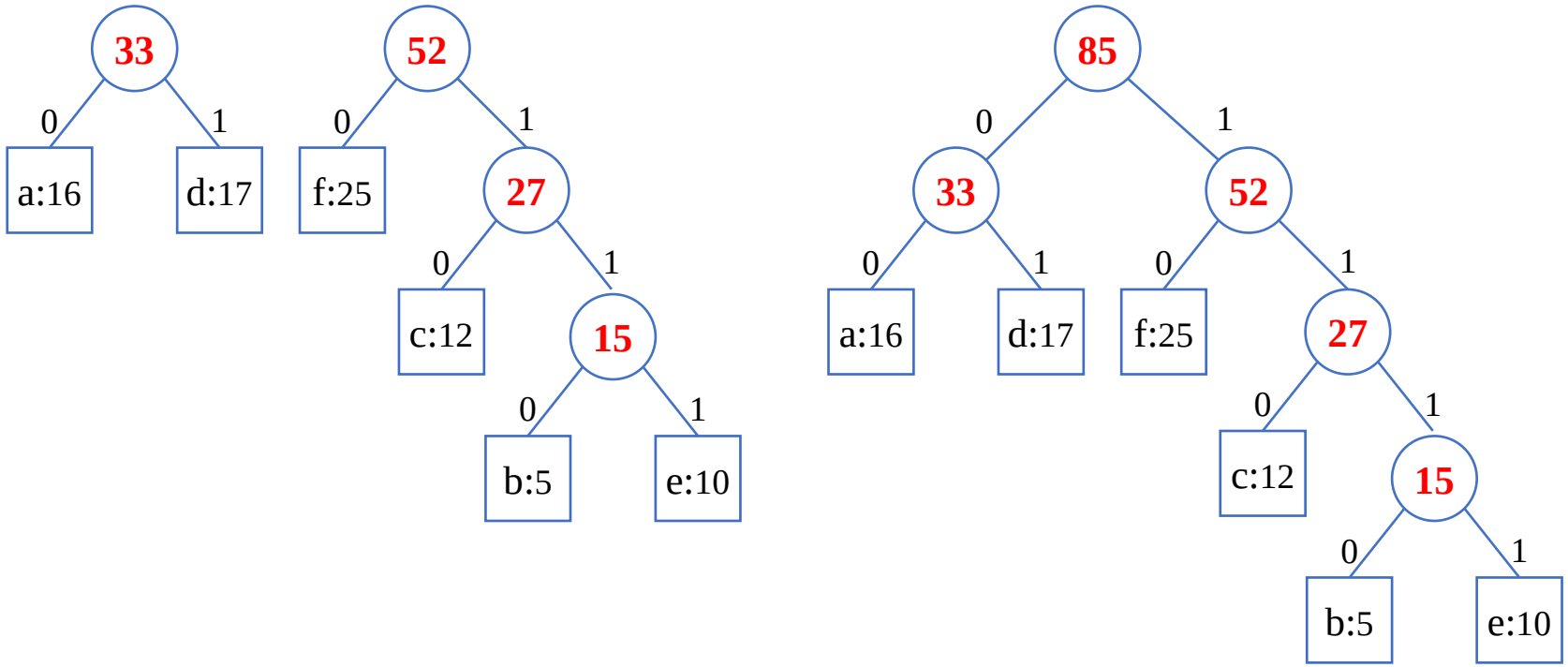
  PQ: 빈도수가 낮은 노드를 먼저 리턴하는 우선순위큐(min-heap)

```
for i in [1..n-1]:
    remove(PQ, p)
    remove(PQ, q)
    r = new node
    r->left = p
    r->right = q
    r->frequency = p->frequency + q->frequency
    insert(PQ, r)
remove(PQ, r)
return r
```

4.4 허프만 코드와 허프만 알고리즘



4.4 허프만 코드와 허프만 알고리즘



Algorithm 4.5: Huffman's Algorithm for the Huffman Code

```
class HuffNode:

    def __init__ (self, symbol, freq):
        self.symbol = symbol
        self.freq = freq
        self.left = None
        self.right = None
```

Algorithm 4.5: Huffman's Algorithm for the Huffman Code

```
    def preorder(self):
        print(self.freq, end=" ")
        if (self.left is not None):
            self.left.preorder()
        if (self.right is not None):
            self.right.preorder()

    def inorder(self):
        if (self.left is not None):
            self.left.inorder()
        print(self.freq, end=" ")
        if (self.right is not None):
            self.right.inorder()
```

Algorithm 4.5: Huffman's Algorithm for the Huffman Code

```
def huffman (n, PQ):
    for _ in range(n - 1):
        p = PQ.get()[1]
        q = PQ.get()[1]
        r = HuffNode(' ', p.freq + q.freq)
        r.left = p
        r.right = q
        PQ.put((r.freq, r))
    return PQ.get()[1]
```

4.4 허프만 코드와 허프만 알고리즘


```
codes = ['b', 'e', 'c', 'a', 'd', 'f']
freqs = [5, 10, 12, 16, 17, 25]

from queue import PriorityQueue

PQ = PriorityQueue()
for i in range(len(codes)):
    node = HuffNode(codes[i], freqs[i])
    PQ.put((node.freq, node))

root = huffman(len(codes), PQ)

print("Preorder:", end=" ")
root.preorder()
print("\nInorder:", end=" ")
root.inorder()
print()
```

주니온TV@Youtube

자세히 보면 유익한 코딩 채널

https://bit.ly/2JXXGqz

- 여러분의 **구독**과 **좋아요**는 강의제작에 큰 힘이 됩니다.
- 강의자료 및 소스코드: **구글 드라이브**에서 다운로드
  (다운로드 주소는 영상 하단 설명란 참고)

  https://bit.ly/3fN0q8t