4장 그리디 알고리즘

choose locally optimal

그 순간마다 최적인 답을 선택하는 것

MST

Minumum Spanning Trees

가중치의 합이 최소가 되는 신장 트리 (트리는 그래프의 특수한 경우)

MST를 구하는 알고리즘

1. 프림 알고리즘
2. 크루스칼 알고리즘

프림 알고리즘

O (E log V) - E는 간선의 개수, V는 정점의 개수

O(n^2)

노드를 추가할 때 마다 연결된 간선을 저장한다.

사이클이 만들어지지 않는 최솟값을 지닌 간선을 택한다.

F′은 MST를 구성하는 최종적인 엣지(간선)들

F는 MST를 구해가는 과정의 엣지들

최종적인 구성인 F′에서 추가하려는 e가 포함되지 않는 경우 → 사이클 발생
따라서 사이클을 해제하면서 mst를 유지하는 e′를 F′에서 제외한다. 이때 e와 e′의 값은 작거나 같아야한다.(같을 수 밖에 없다)

크루스칼 알고리즘

O (E log E) - E는 간선의 개수

최악의 경우 O(n^2 lg n) - 간선의 수가 많을 경우 dense

1. 엣지들의 모음
2. 사이클을 만들지 않는 선에서 엣지들의 최솟값들을 추가
3. 모든 노드들에 도달할 수 있으면 종료

merge 연산

1. 노드 1,2,3,4,5 .... 들이 초기에는 자기 자신을 가르키도록 한다.
2. 노드1와 노드2를 연결한다고 가정할 때, 노드 인덱스 중 작은 값을 가르킨다.
3. merge연산에서 사이클 판단 하는 방법
   노드 1 - 노드 3, 노드1 - 노드2 가 연결되었다고 가정한다.
   노드3이 최종적으로 가리키는 곳과 노드2가 최종적으로 가리키는 곳이 동일하다면 노드2 3을 merge하면 사이클이 발생한다.

Prefix Code

a: 10, b: 0, c: 11 처럼 되어있는 것이 Prefix Code

a(10)은 어느 코드의 시작부분에 포함되지 않는다.

b(11) 또한 어느 코드의 시작부분에 포함되지 않는다. → 0으로 시작하는건 b 뿐

c(11) 또한 어느 코드의 시작 부분에 포함되지 않는다.

허프만 코드 (최적 합병 트리) - n log n

O(n log n)

압축 방법 중 하나

자주 나타나면 비트수를 적게, 덜 나타나면 비트 수를 많게

자주 나타나면 루트에 가깝게 있고 덜 나타나면 멀리 있다.

허프먼 코드 작성 법

1. 데이터(값)의 각각의 빈도수 파악 (빈도수 = 가중치)
2. 가중치가 낮은 순으로 오름차순 정렬 (a:1, b:2 ... f:6)
3. 가중치가 낮은 것 끼리 합산

아래 유튜브 참고하기

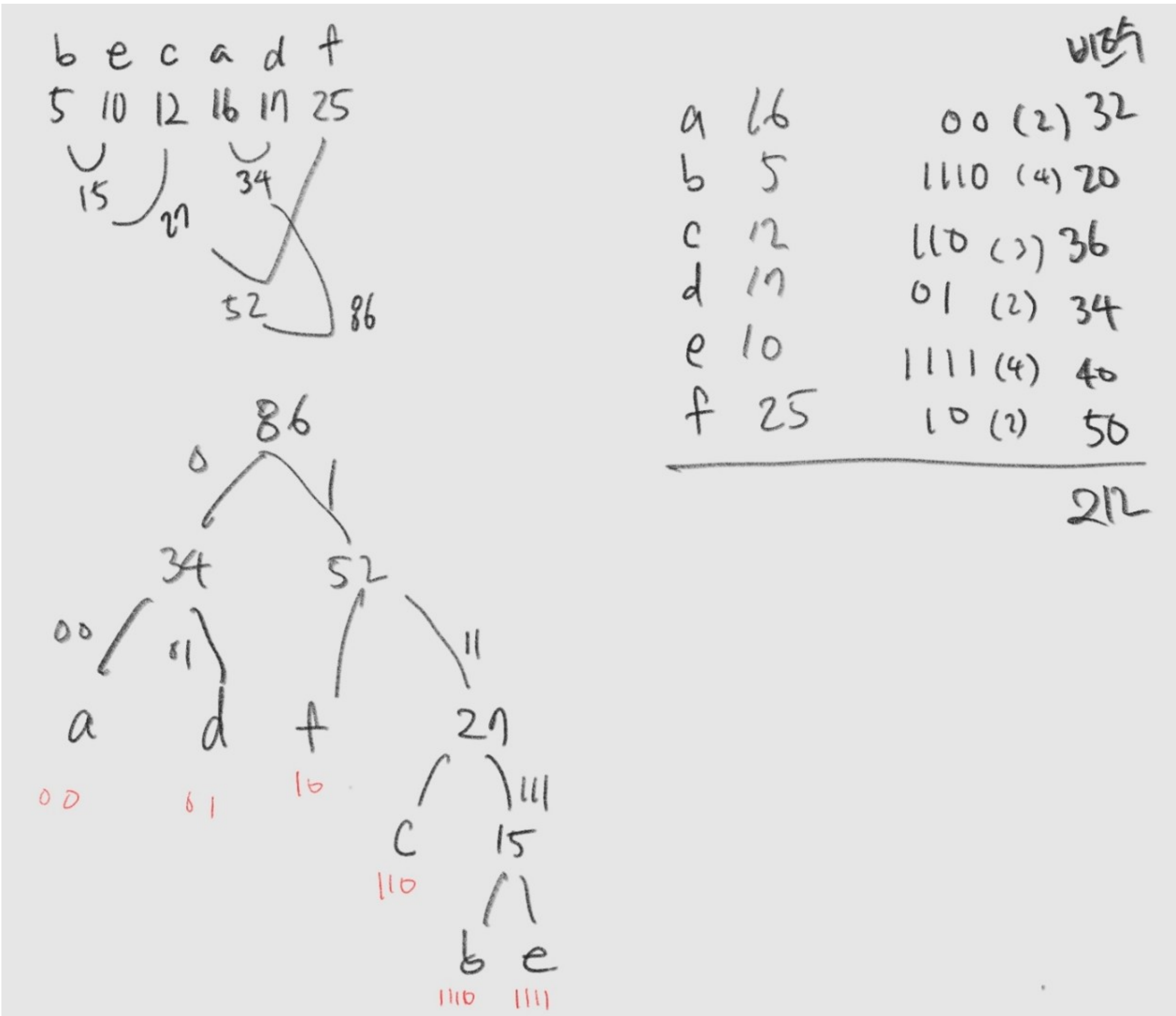
https://www.youtube.com/watch?v=ctmKoVsLJ8M

아래 예시도 풀어보기(각 알파벳의 하프먼 코드, 이진 트리, 비트 수, 총 비트 합 그려보기)

- Example

b:5 e:10 c:12 a:16 d:17 f:25

정답



4.4.2 Huffman's Algorithm

```
for (i = 1; i < n; i++) {
        remove(PQ, p);
        remove(PQ, q);
        r = new nodetype();
        r.left = p;
        r.right = q;
        r.frequency = p.frequency + q.frequency;
        insert(PQ, r);
}
remove(PQ, r);
return r;
```

Min priority queue

```
class nodetype
{
        char symbol;
        int frequency;

        nodetype left;
        nodetype right;
}
```

Time complexity
Θ(n log n)

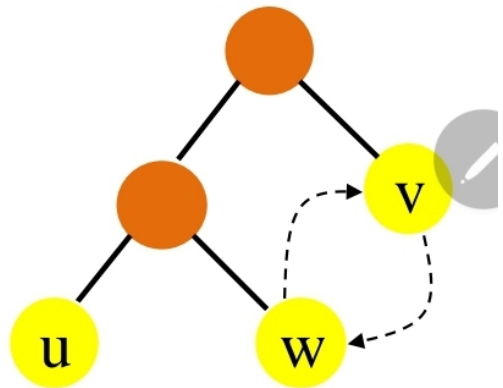
37

– Induction Step – continued

frequency(w) ≥ frequency(v) – *why?*

depth(w) (≥) depth(v) in T

Create a new tree T' by swapping the positions of the branches rooted at v & w.



cost(T')

= cost(T) – depth(w) * frequency(w) - depth(v) *frequency(v) + depth(w) * frequency(v) + depth(v) * frequency(w)

= cost(T) + (depth(w) – depth(v)) * ($f(v) - f(w)$)

↳ ⊕ ↳ ⊖

≤ cost(T)

Hence, T' is optimal.

냅색 알고리즘

분할이 가능할 경우 - 그리디 (무게당 가치로 접근) (value / weight)

분할이 불가능할 경우 - DP

분할이 불가능 할경우 DP를 사용하는 냅색 - O(2^n(

❑ Dynamic Programming Approach to the 0-1 Knapsack Problem

$\Theta(2^n)$

$$P[i][w] = \begin{cases} \max(\ P[i-1][w]\ ,\ p_i + P[i-1][w-w_i]\ ) & \text{if } w_i \le w \\ P[i-1][w] & \text{if } w_i > w \end{cases}$$

➔ The value we are looking for is P[n][W].

➔ We can determine this value using a two dimensional array P[0..n][0..W] where

$$\begin{array}{ll} P[0][w] = 0 & 0 \le w \le W \\ P[i][0] = 0 & 0 \le i \le n \end{array}$$

51