Greedy Algorithms

- 각 단계에서 가장 좋을거라 생각되는 선택을 취함
- 반드시 최적의 해를 구한다고 보장할 수는 없다
  - Greedy-choice property를 가지는 경우에만 최적해를 구한다
- ex)
  - activity selection
  - huffman code
  - minimum spanning tree algorithms
  - dijkstra's algorithm for shortest paths from a single source
  - rod-cutting : dynamic으로 풀어야함

An Activity-Selection Problem

- Activity들이 finish time의 **단조 증가 순으로 정렬**되어 있을 때

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

- 활동 시간이 겹치지 않게 compatible activites의 최대 집합을 찾는 문제

$$\{a_3, a_9, a_{11}\}$$
$$\boxed{\begin{matrix}\{a_1, a_4, a_8, a_{11}\} \\ \{a_1, a_4, a_9, a_{11}\}\end{matrix}}$$

- The activity selection problem exhibits optimal substructure

S_{ij}: f_i 이후에 시작하고 s_j 이전에 끝나는 a_i 들의 집합

$c[i,j]$ = size of optimal solution for S_{ij} $\quad c[i,j] = c[i,k] + c[k,j] + 1$

$$c[i,j] = \begin{cases} 0 & \text{if } S_{ij} = \emptyset , \\ \max\limits_{a_k \in S_{ij}} \{c[i,k] + c[k,j] + 1\} & \text{if } S_{ij} \neq \emptyset . \end{cases}$$

  - The greedy choice : a_i with smallest f_i = the first activity is always in the solution

Let $S_k = \{a_i \in S : s_i \geq f_k\}$ $\;a_k$ 가 끝난 이후에 시작하는 activities 의 집합

→ solution = { a_1, solution of S_1 }

Recursive Activity Selector

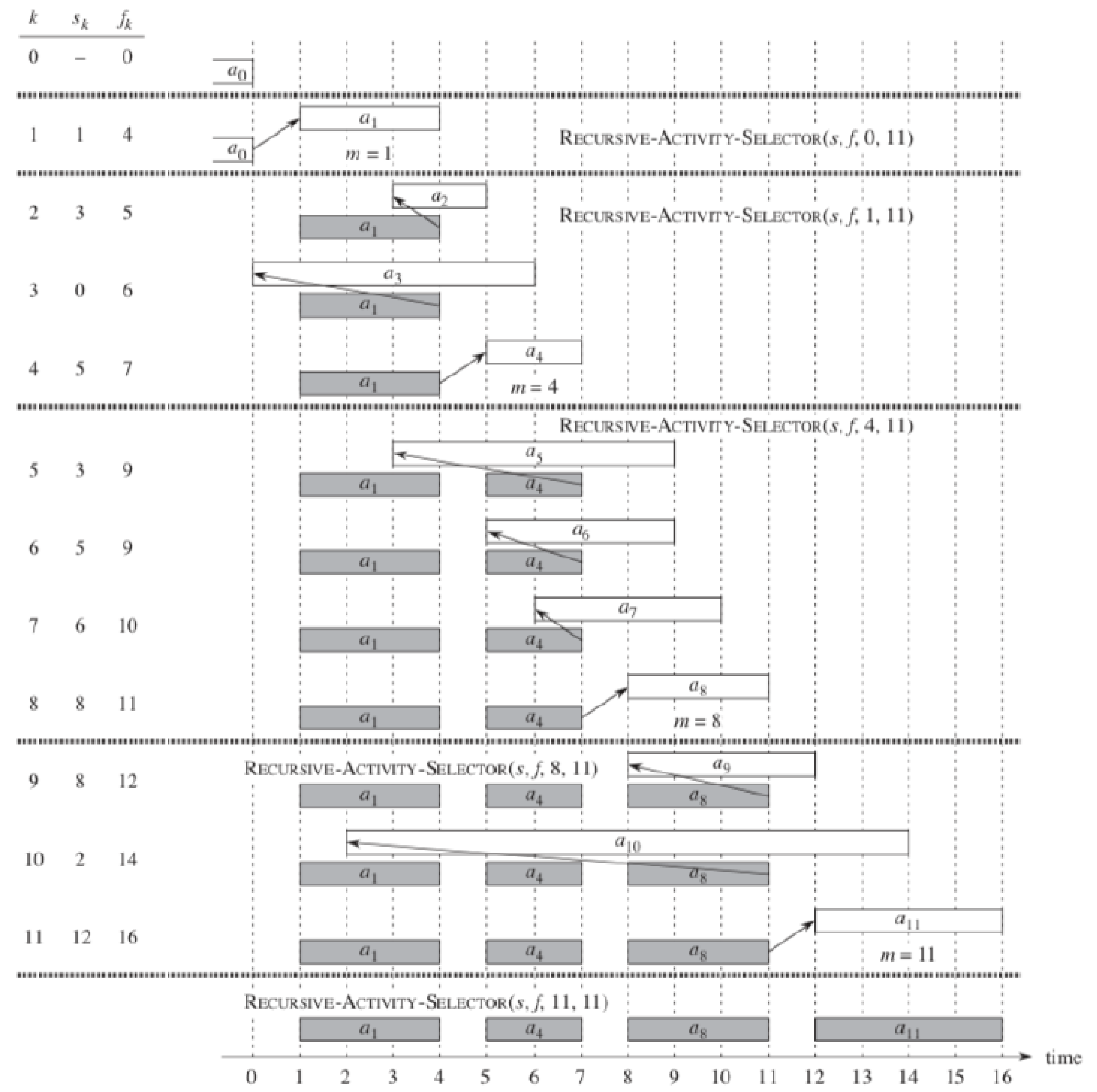
- Returns a maximum-size set of mutually compatible activities in S_k

Let $S_k = \{a_i \in S : s_i \geq f_k\}$

size of the original problem

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
m = k + 1
while m ≤ n and s[m] < f[k]        // find the first activity in S_k to finish
    m = m + 1
if m ≤ n
    return {a_m} ∪ RECURSIVE-ACTIVITY-SELECTOR(s, f, m, n)
else return Ø
```

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16



Iterative Function

```
GREEDY-ACTIVITY-SELECTOR(s, f)
n = s.length
A = {a_1}
k = 1
for m = 2 to n
    if s[m] ≥ f[k]
        A = A ∪ {a_m}
        k = m
return A
```

$m \rightarrow$

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

$$= \Theta(n)$$

Elements of Greedy Algorithms

- Greedy algorithm 만들기
  - 하나의 (grreedy) 선택을 하면 나머지 부분도 하나의 subproblem만 남도록 최적화 문제를 세워라
  - Prove that there is always an optimal solution to the original problem that makes the greedy choice
  - Demonstrate optimal structure(Greed choice와 subproblem의 optimal solution을 결합하면 전체 문제의 optimal solution을 얻는다는 것을 보임)
- When can we use a greedy algorithms?
  - greedy-choice property + optimal substructure

Greedy-Choice Property

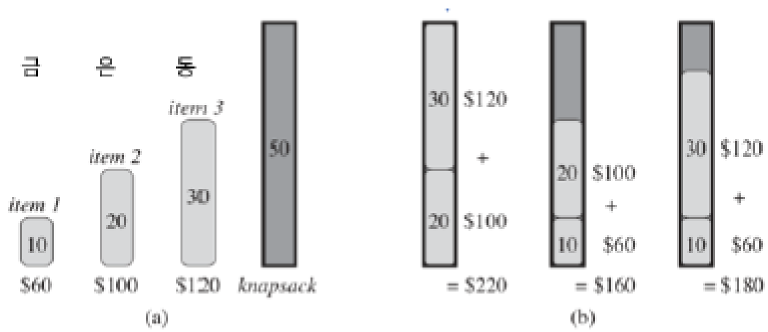
- 어떤 선택을 할지 고려할 때 부분 문제들의 결과를 고려할 필요없이 현재 고려 중인 문제에서 최적인 문제를 선택해도 된다
  - dynamic programming : subproblem들의 해를 먼저 구한다 (bottom-up)
  - greedy algorithm : choice를 먼저 한 다음 나머지 subproblem을 푼다 (top-down)
    - ex) rod-cutting problem은 greedy-choice property가 없다 -> dynamic으로 풀어야 함

Optimal Substructure

- An optimal solution to the problem contains optimal solutions to subproblems

0-1 Knapsack Problem

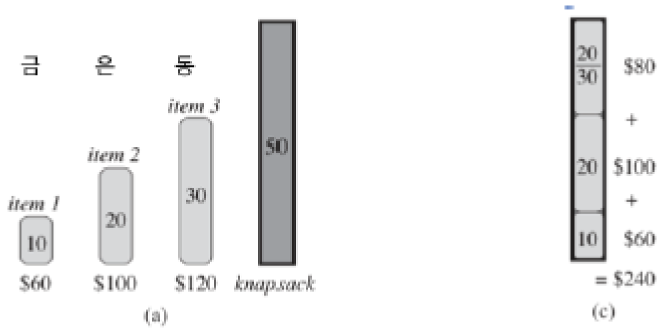
- n items : w_i is a weight of i-th item, v_i is a value of i-th item
- W : knapsack capacity
- problem : choose a set of items maximizing total value, and not exceeding knapsack capacity



- Has optimal substructure but not greedy-choice property -> dynamic programming

Fractional knapsack problem

- n items : w_i is a weight of i-th item, v_i is a value of i-th item
- W : knapsack capacity
- problem : choose a set of fractional items maximizing total value, and not exceeding knapsack capacity



- Has optimal substructure and greedy-choice property -> greedy algorithm $O(n\ lg\ n)$
  - sort items in value / wieght ($= v_i / w_i$)

```
FRACTIONAL-KNAPSACK(v, w, W)
  load = 0
  i = 1
  while load < W and i ≤ n
      if w_i ≤ W − load
          take all of item i
      else take (W − load)/w_i of item i
      add what was taken to load
      i = i + 1
```

Huffman Codes(for data compression)

	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

- a~f 문자를 100000개 포함한 파일의 길이
  - fixed-length (3-bit) code 사용 : 300000 비트 필요
  - variable-length code 사용 : 파일 크기 줄이기 가능(가변적) - 224000 비트 필요
    - codeword의 끝을 어떻게 알 것인가

Prefix codes

	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

- 어느 codeword도 다른 codeword의 prefix가 아니다
  - guarantees unambiguity in decoding

Problem

- 주어진 문자 분포에 대해 B(T)를 최소화하는 최적의 Prefix Code를 만들어라

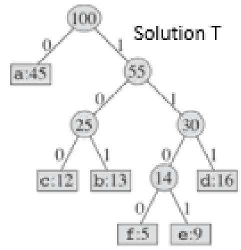
  $B(T) = \sum_{c \in C} c.freq * d_T(c)$

  - C : 파일에 사용된 문자 집합
  - $c.freq$: 문자 c가 파일에서 사용된 빈도 (사용된 횟수 / 전체 문자 갯수)
  - $d_T(c)$: 만들어진 code tree에서 문자 c를 나타내는 노드의 depth(root 에서부터 edge의 갯수)

Problem C

	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5

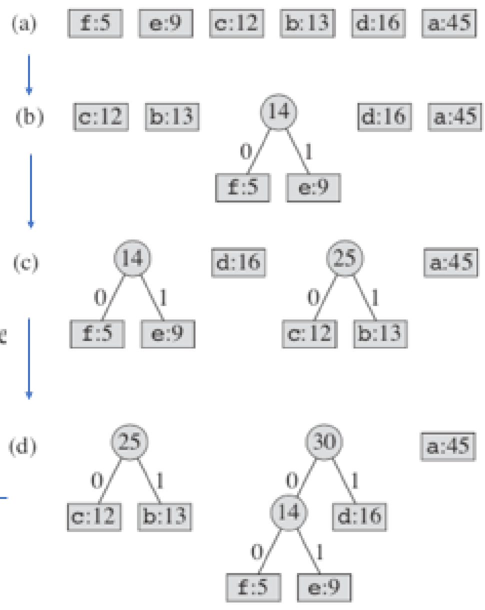
Solution T

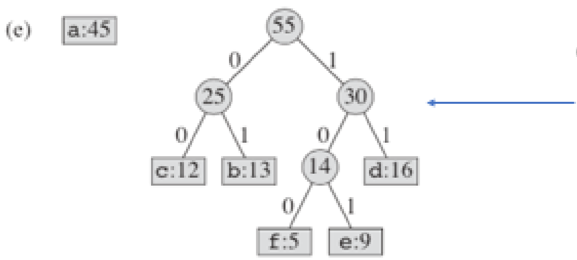


  - 최적의 prefix code는 항상 full binary tree로 표현됨(자식이 0개 또는 2개)

```
HUFFMAN(C)
  n = |C|
  Q = C
  for i = 1 to n − 1
      allocate a new node z
      z.left = x = EXTRACT-MIN(Q)
      z.right = y = EXTRACT-MIN(Q)
      z.freq = x.freq + y.freq
      INSERT(Q, z)
  return EXTRACT-MIN(Q)      // return the root of the
```

greedy choice



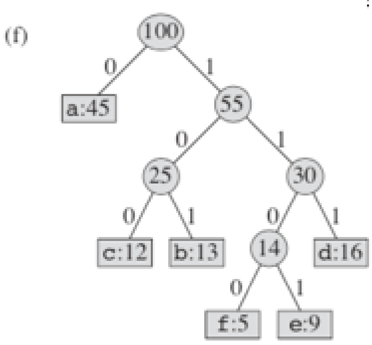



```
HUFFMAN(C)
  n = |C|
  Q = C
  for i = 1 to n − 1
      allocate a new node z
      z.left = x = EXTRACT-MIN(Q)
      z.right = y = EXTRACT-MIN(Q)
      z.freq = x.freq + y.freq
      INSERT(Q, z)
  return EXTRACT-MIN(Q)      // return the root of the tree
```

$O(n)$ BUILD_MIN_HEAP

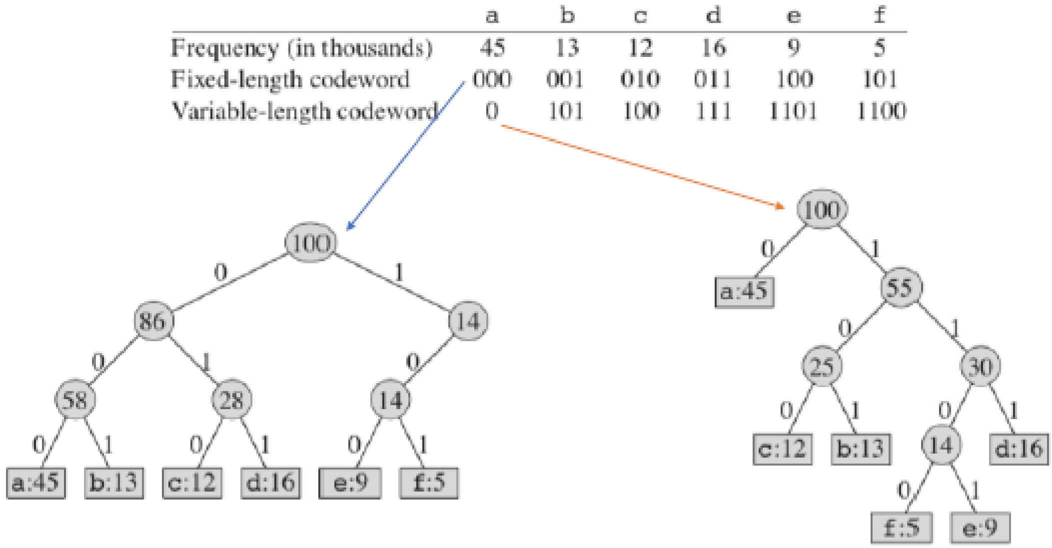
$O(\lg n)$

(f)



→Q is implemented in a binary min-heap

→ $O(n \lg n)$

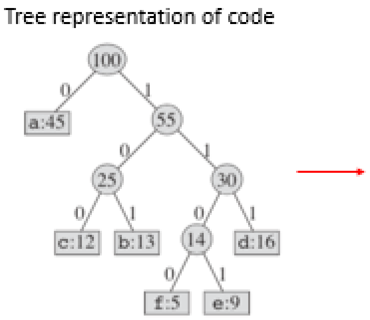


	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

Optimizing prefix code with respect to $B(T)$

$B(T) = \sum_{c \in C} c.freq * d_T(c)$

- C : 파일에 사용된 문자 집합
- $c.freq$: 문자 c가 파일에서 사용된 빈도 (사용된 횟수 / 전체 문자 갯수)
- $d_T(c)$: 만들어진 code tree에서 문자 c를 나타내는 노드의 depth(root 에서부터 edge의 갯수)
  = codeword의 길이



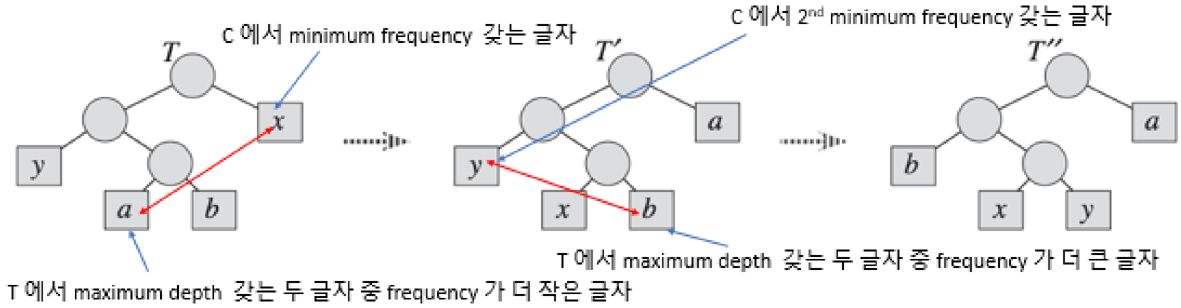
	a	b	c	d	e	f
c.freq	0.45	0.13	0.12	0.16	0.09	0.05
$d_T(c)$	1	3	3	3	4	4

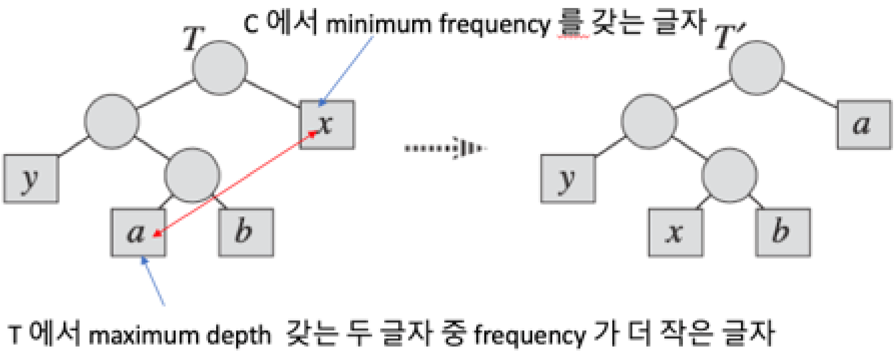
Lemma 16.2

- Let C be an alphabet in which each character $c \in C$ has frequency $c.freq$
- Let x and y be two characters in C having the lowest frequencies
- Then there exists an optimal prefix code for C in which the codewords for x and y have the same length and differ only in the last bit

증명

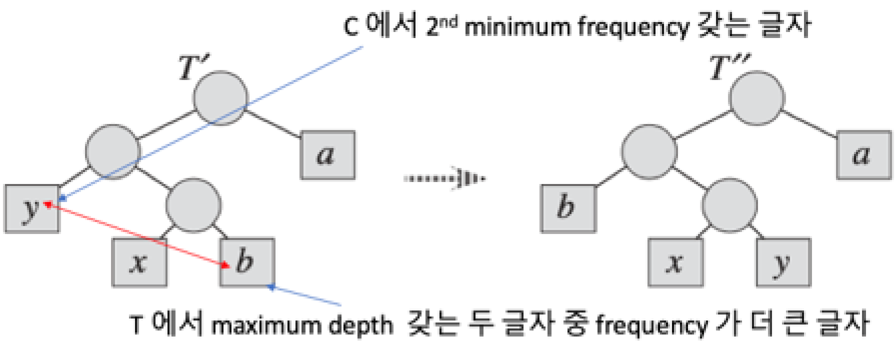
- 주어진 문제에 대한 임의의 optimal prefix code를 나타내는 tree T 를 변형하여 x와 y의 최대 깊이를 갖는 sibling leaf node가 되는 T''를 만들면 T''도 optimal prefix code를 나타냄을 보임
  - $B(T) = B(T'')$임을 보임





$$
\begin{aligned}
&B(T) - B(T') \\
&\quad = \sum_{c \in C} c.freq \cdot d_T(c) - \sum_{c \in C} c.freq \cdot d_{T'}(c) \\
&\quad = x.freq \cdot d_T(x) + a.freq \cdot d_T(a) - x.freq \cdot d_{T'}(x) - a.freq \cdot d_{T'}(a) \\
&\quad = x.freq \cdot d_T(x) + a.freq \cdot d_T(a) - x.freq \cdot d_T(a) - a.freq \cdot d_T(x) \\
&\quad = (a.freq - x.freq)(d_T(a) - d_T(x)) \\
&\quad \geq 0 ,
\end{aligned}
$$

T는 optimal prefix code 를 표현하므로 $B(T) = B(T')$



마찬가지로 $B(T') = B(T'')$

따라서 $B(T) = B(T'')$

Lemma 16.3

- Let C be an alphabet in which each character $c \in C$ has frequency $c.\,freq$
- Let x and y be two characters in C having the lowest frequencies
- Let $C' = C - \{x, y\} \cup \{z\}$. In C'
- $z.\,freq = x.\,freq + y.\,freq$ and $c.\,freq$ are same as in C for all other characters
- Let T' be any tree representing an optimal prefix code for C'
- Then the tree T, obtained from T' by replacing the leaf node for z with an internal node having x and y as children , represents an optimal prefix code for the alphabet C

For each character $c \in C - \{x, y\}$, we have that $d_T(c) = d_{T'}(c) \rightarrow c.freq \cdot d_T(c) = c.freq \cdot d_{T'}(c)$

$$d_T(x) = d_T(y) = d_{T'}(z) + 1 \longrightarrow \begin{aligned} x.freq \cdot d_T(x) + y.freq \cdot d_T(y) &= (x.freq + y.freq)(d_{T'}(z) + 1) \\ &= z.freq \cdot d_{T'}(z) + (x.freq + y.freq) \end{aligned}$$

$$B(T) = B(T') + x.freq + y.freq$$

Proof of Lemma 16.3 by contradiction

- T가 C의 optimal prefix code 를 나타내지 않는다고 가정

T, T'' for C
T', T''' for C'

→ C 의 optimal prefix code 를 나타내는 T'' s.t. $B(T'') < B(T)$ 가 있음
→ T'' 은 x와 y가 sibling leaf node 임 by lemma 16.2
→ T''을 변형하여 x와 y의 공통 부모 노드를 z 로 바꾼 트리 T'''을 만들면 C'의 prefix code
→ $\boxed{B(T''') < B(T')}$ 라서 T'은 C'의 optimal prefix code 가 아님 (모순)

For each character $c \in C - \{x, y\}$, we have that $d_{T''}(c) = d_{T'''}(c) \rightarrow c.freq\ d_{T''(c)} = c.freq\ d_{T'''}(c)$

$$d_{T''(x)} = d_{T''(y)} = d_{T'''}(z) + 1 \longrightarrow \begin{aligned} x.freq\ d_{T''}(x) + y.freq\ d_{T''}(y) &= (x.freq + y.freq)(d_{T'''}(z) + 1) \\ &= z.freq\ d_{T'''}(z) + x.freq + y.freq \end{aligned}$$

$$B(T'') = B(T''') + x.freq + y.freq$$

$$\begin{aligned} B(T''') &= B(T'') - x.freq - y.freq \\ &< B(T) - x.freq - y.freq \\ &= B(T') , \end{aligned}$$

- Lemma 16.2, 16.3에 의해 Huffman code algorithm은 optimal prefix code를 만든다