[CSE3081(2반)] 알고리즘 설계와 분석

2020학년도 2학기

강의자료

(2020.11.10 화요일)

서강대학교 공과대학 컴퓨터공학과
임 인 성 교수

본 강의에서 제작하여 제공하는 PDF 파일, 동영상, 그리고 예제 코드 등의 강의 자료의 저작권은 특별히 명기되어 있지 않은 한 서강대학교에 있습니다.

본인의 학습 목적 외에 공개된 장소에 올리거나 타인에게 배포하는 등의 행위를 금합니다. 협조 부탁합니다.

[주제 5]
Greedy Methods

Algorithm Design Techniques

- Divide-and-Conquer Method
- Dynamic Programming Method
- Greedy Method
- Backtracking Method
- Local Search Method
- Branch-and-Bound Method
- Etc.

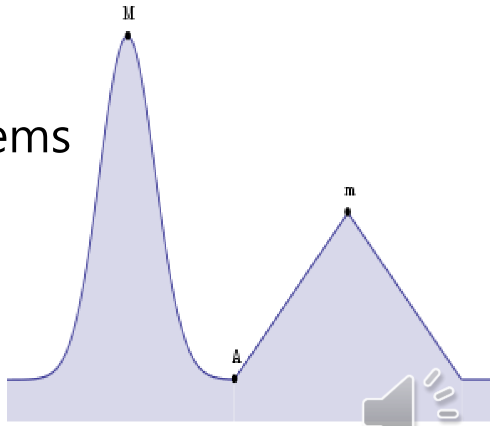
The Greedy Method

- A technique to follow the problem-solving heuristic of making the locally optimal choice at each stage.

- **Strategy**
  - Make the choice that appears best at each moment!
  - ✓ It is hoped to arrive at a globally optimal solution by making a locally optimal choice.

- **Pros and cons**
  - Simple and straightforward to design an algorithm.
  - Does not guarantee the optimal solution to all problems
    - **Local maximum versus global maximum**

https://en.wikipedia.org/wiki/Greedy_algorithm



Huffman Coding

- **Data compression**
  - Data compression can save storage space for files.
  - Huffman coding is just one of many data compression techniques.

- **Problem**
  - Given a file, find a binary character code for the characters in the file, which represents the file in the **least** number of bits.

- **Example**
  - Original text file: `ababcbbbc`
  - Huffman codes: `a = 10, b = 0, c = 11`

  → Compressed file: `1001001100011`

b b
0 0 0 1 1 1

Is it possible to have a code set where `a = 01`, `b = 0`, and `c = 11`?

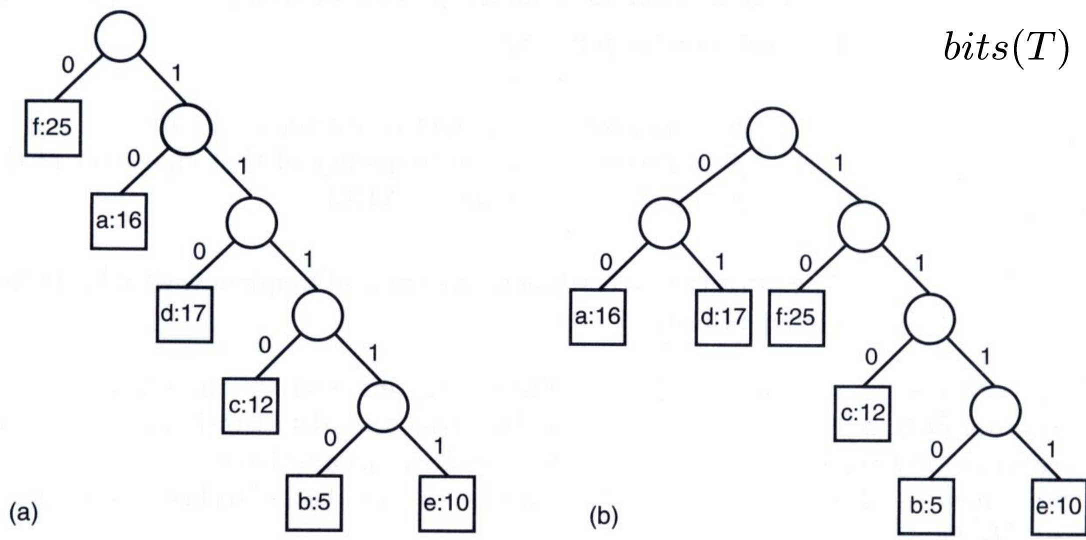
Prefix Codes

- **No codeword can be a prefix of any other code**.
  - Otherwise, decoding is impossible!

Uniquely decodable!

  - ☺ Example 1
    - `a = 00, b = 1110, c = 110, d = 01, e = 1111, f = 10`
  - ☹ Example 2
    - `a = 00, b = 1100, c = 110, d = 01, e = 1111, f = 10`
- **Binary trees corresponding to prefix codes**
  - The code of a character *c* is the label of the path from the root to *c*.
  - Decoding of an encoded file is trivial.

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

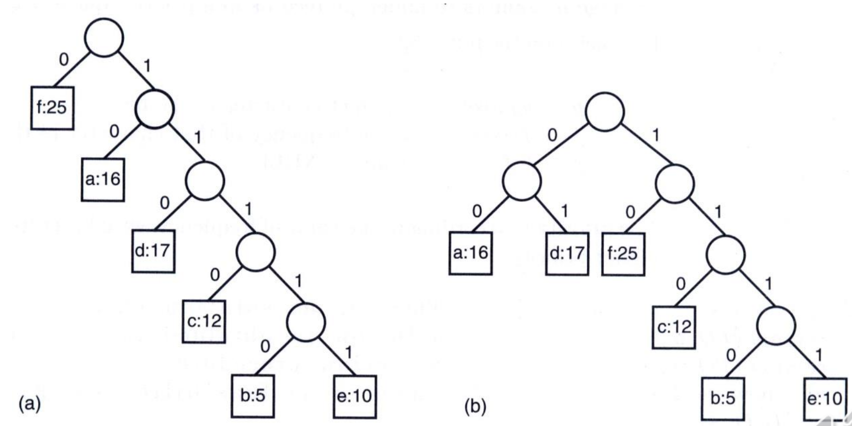


- **Problem**
  - Given a file F to be encoded with a character set $V = \{v_1, v_2, \ldots, v_n\}$, find an optimal prefix binary code with a corresponding binary tree T that **minimizes** the cost function

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

where $freq(v_i)$ is the number of times v_i occurs in F, and $depth(v_i)$ is the depth v_i of in T.

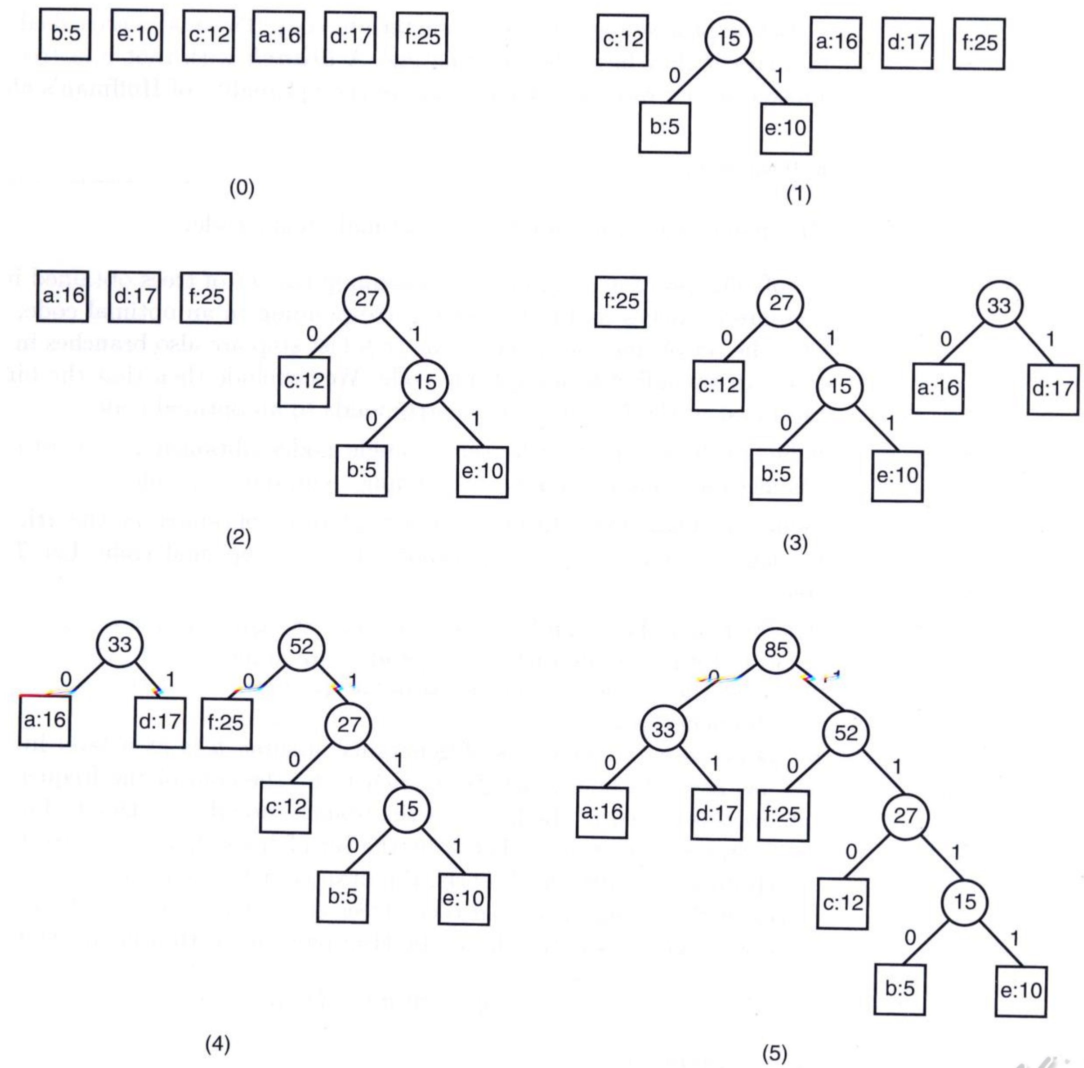
✓ A Greedy approach successfully finds an optimal code.



Huffman's Algorithm

- **Idea**
  - Put the rarest characters at the bottom of the tree.

- A **greedy** approach
  - Repeat the following until only one tree is left:
    ① Start from a set of single node trees.
    ② Pick up two trees u and v with **the lowest frequencies**.
    ③ Merge them by adding a root node w where the frequency of the new node is the sum of those of u and v.
    ④ Replace u and v by w.



Implementation and Time Complexity

- **Implementation issues**
  - How can you manage a dynamic set to which the following operations occur frequently:
    - Delete the elements with the highest priority from the list.
    - Insert an element with some priority into the list.
    - ✓ The answer is to use Priority Queue.

  - The priority queue can be implemented in many ways. Which one would you use?

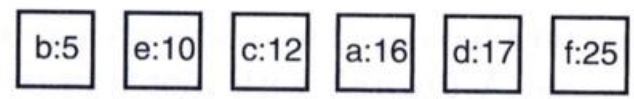
Representation	Insertion	Deletion
Unordered array	$O(1)$	$O(n)$
Unordered linked list	$O(1)$	$O(n)$
Sorted array	$O(n)$	$O(1)$
Sorted linked list	$O(n)$	$O(1)$
Heap	$O(\log n)$	$O(\log n)$

✓ The answer is to use the priority queue based on (min) heap.

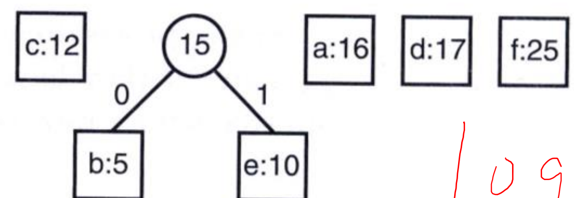
$\rightarrow$ $O(n \log n)$ time

```
typedef struct _node {
  char symbol;
  int freq;
  struct _node *left;
  struct _node *right;
} NODE;
NODE *u, *v, *w;
…
for (i = 1; i <= n; i++) {
  /* insert the n single-node trees */
}
for (i = 1; i <= n-1; i++) {
  u = PQ_delete();
  v = PQ_delete();
  w = make_a_new_node();
  w->left = u;
  w->right = v;
  w->freq = u->freq + v->freq;
  PQ_insert(w);
}
w = PQ_delete();
/* w points to the optimal tree. */
```

$O(n)$



$\log n + \log n-1 + \log n-2 + \cdots + \log 2$

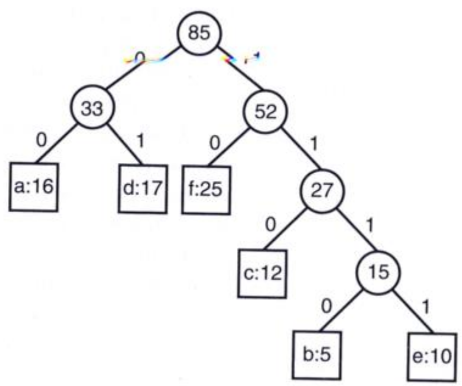


$= \log n! = O(n \log n)$

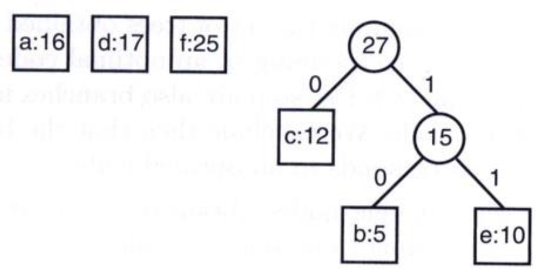
Correctness of the Huffman's Algorithm

siblings, branch

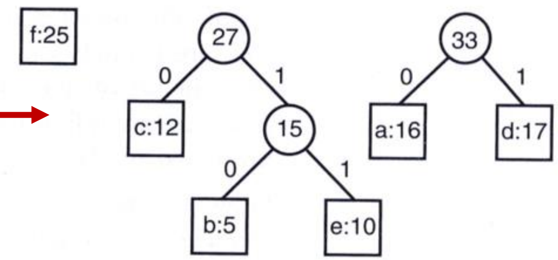
- **(Proof by mathematical induction)** If the set of trees obtained in the *i*-th step are branches in a binary tree corresponding to an optimal code, then the set of trees obtained in the (*i*+1)st step are also branches in a binary tree corresponding to an optimal code.



Optimal binary tree

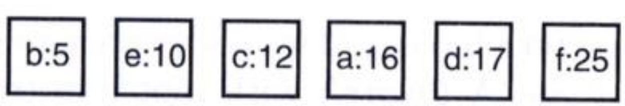


i-th step

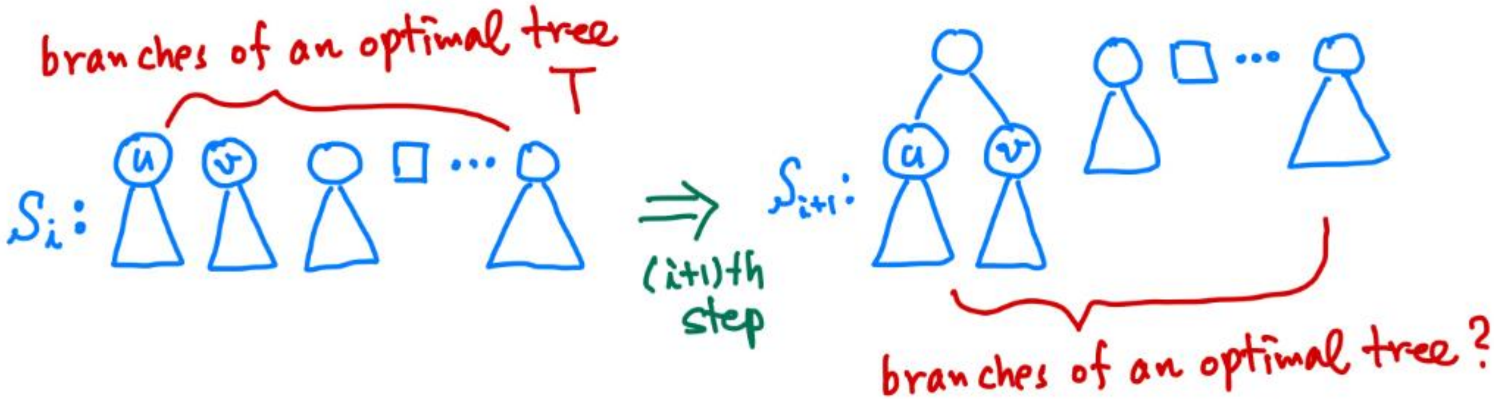


(i+1)-th step

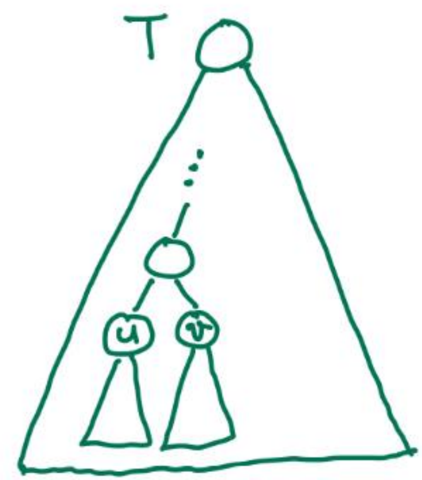
  – **(Base step)** When $k = 0$, each tree is trivially a branch of an optimal tree.



– **(Induction step)** Suppose that the proposition is true when $k = i$, that S is the set of trees that exist after the ith step, and that T is the corresponding optimal tree. Let u and v be the root of the trees combined in the $(i+1)$st step.



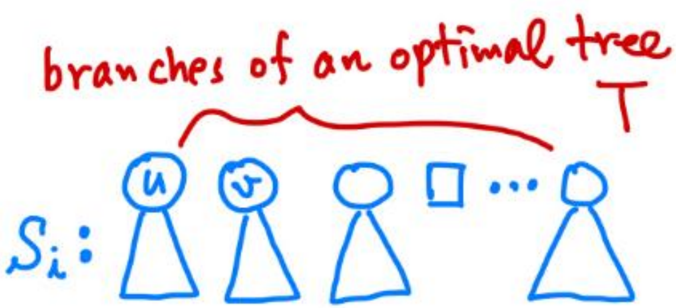
• **Case 1:** If u and v are siblings in T, we are done.

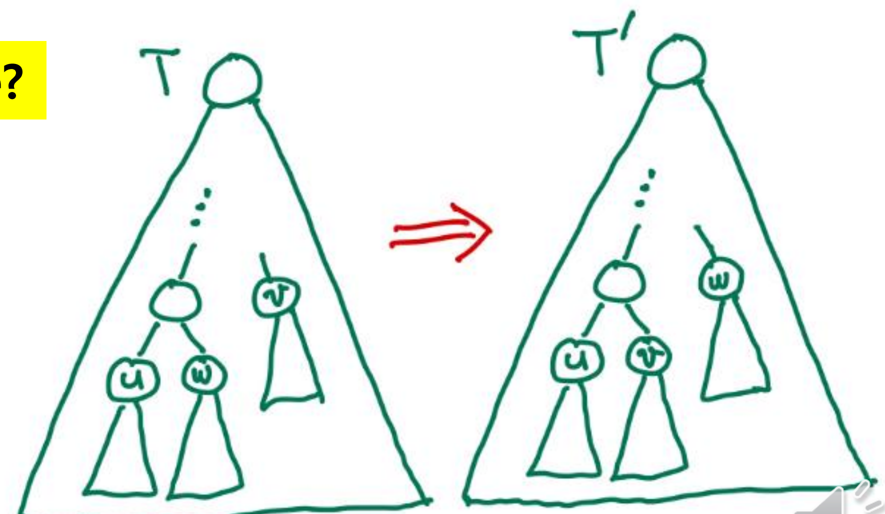


- **Case 2:** Otherwise, assume that u is at a level in T at least as low as v, and that w is the u's sibling in T.
  - The branch in T with root w is one of the trees in S or contains one of those trees as a subtree. ← why?
  - Therefore, $freq(w) \geq freq(v)$ and $depth(w) \geq depth(v)$ in T
  - If we create a new tree T' by swapping the two branches with root v and w, then

    $bits(T') = bits(T) + (depth(w) - depth(v))*(freq(v) - freq(w)) \leq bits(T)$.
  - Since $bits(T) \leq bits(T')$, T' is also optimal. Hence, the proposition also holds when $k = i+1$. □

What happens if all the steps are done?





Maximum Non-overlapping Intervals

- **Problem:** Let $A = \{a_1, a_2, \cdots, a_n\}$ be a set of n activities, where a_i has start times s_i and finish times f_i ($0 \le s_i < f_i < \infty$). If selected, activity a_i takes place druing the time interval $[s_i, f_i)$. Two activities a_i and a_j are called *compatible* if the time intervals $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap. Now, select a largest set S of mutually compatible activities. (We assume that the activities are given in such a way that $f_1 \le f_2 \le f_3 \le \cdots \le f_{n-1} \le f_n$.)

- **Example**