Optimal 2-Way Merge Pattern

강지훈
jhkang@cnu.ac.kr
충남대학교 컴퓨터공학과

Optimal 2-Way Merge Pattern

Merging 2 sorted list

- Input: 2 sorted list
  - list1: size N1
  - list2: size N2

- Total number of key comparisons
  - O(N1+N2)

2-Way Merging of 3 Sorted Lists

Input Example

- list1: 20, list2: 30, list3: 50

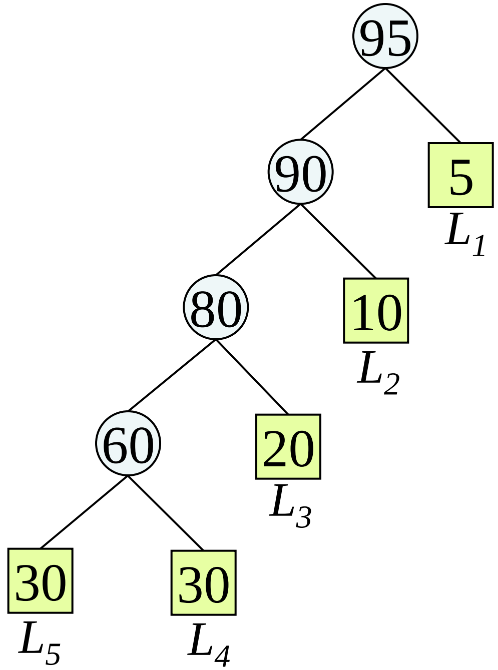
Minimum Key Comparisons?

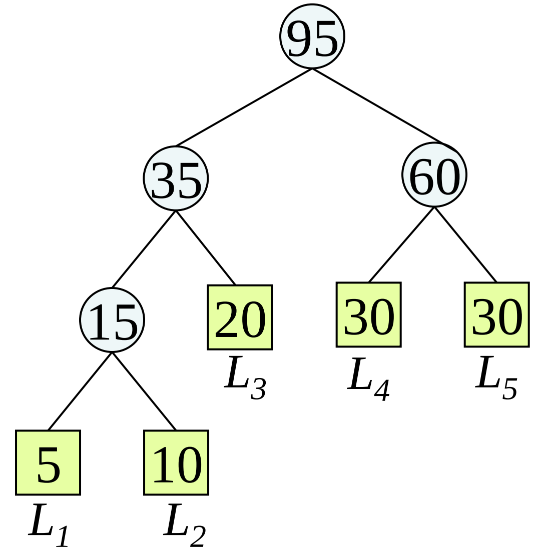
- Case 1:
  - Merge (list1+list2) into list4 => 50 comparisons
  - Merge (list4+list3) into list5 => 100 comparisons
  - Total (50+100)=150 comparisons
- Case 2:
  - Merge (list1+list3) into list4 => 70 comparisons
  - Merge (list4+list2) into list5 => 100 comparisons
  - Total (70+100)=170 comparisons
- Case 3:
  - Merge (list2+list3) into list4 => 80 comparisons
  - Merge (list4+list1) into list5 => 100 comparisons
  - Total (80+100)=180 comparisons

Optimal 2-way merge pattern

Example

- There are 5 sorted lists whose lengths (# of records) are 5, 10, 20, 30, 30 respectively.
- Determine a 2-way merge pattern such that the fewest comparisons are required.





(30+30) + (60+20)
+(80+10) + (90+5) = 325

(5+10) + (15+20) + (30+30)+ (35+60) = 205

(5+10) + (15+20) + (30+30) + (35+60)
= (5+10) + ((5+10) + 20) + (30+30)+
((5+10+20) + (30+30))

= $5 \cdot 3 + 10 \cdot 3 + 20 \cdot 2 + 30 \cdot 2 + 30 \cdot 2$

Weighted External Path Length.

Optimal Merge of Runs

- Weighted External Path Length
  - Let $(n+1)$ nodes with positive weights q_i $(1 \leq i \leq n+1)$ be given.
  - We can construct a binary tree using these nodes as external nodes (i.e. leaves).
  - The Weighted External Path Length is: $\sum_{i=1}^{n+1} q_i \cdot k_i$ where
    - k_i is the path length from the root to the external node with weight q_i .
- We want to determine a binary tree with minimal weighted external path length.

Huffman Code
&
Data Compression

Huffman code

- Input:
  - Consider a message consisting of a sequence of characters.
  - We can know the frequencies for each character in the sequence.
- Goal:
  - We wish to encode each character into a sequence of 0's and 1's such that the total number of bits for the message is minimum.
  - Of course, each character must be distinguishable from others: Prefix Property.

Fixed-Length Encoding Scheme

Encoding Symbols Using Bits:

- Each symbol is represented by a sequence of bits.

Fixed-Length Encoding Schemes

- ASCII: 7 bit encoding scheme
  - But, one byte (8 bits) is used with msb 0 for each symbol (character) in most computers
- EBCDIC: 8 bit encoding scheme
  - (only used in IBM main frames)
- $\lceil \log_2 n \rceil$ bits are enough to represent n symbols
  - 5 bits are enough to represent 32 symbols

Variable-Length Encoding Scheme

- Morse code:
  - Translating each letter into a sequence of dots (short pulses) and dashes (long pulses)
  - Morse code can be considered as a sequence of bits
    - 0 for dot / 1 for dash
- Morse's Approach:
  - One could communicate more efficiently by encoding frequent letters with short strings
  - More frequent letters to shorter stings.
    - 'e':0 , 't':1, 'a':01, ……
  - "pause" is necessary for dealing with ambiguity.
    - 0101: "etet", or "eta", or "aet", or "aa" ?
    - 01 pause 01 for "aa"
  - Morse code using <dot / dash/ pause> actually uses 3 letters.
- We really want to encode everything using only the bits 0 and 1.

Prefix codes

- The ambiguity in Morse code:
  - The bit string that encodes one letter is a prefix of the bit string that encodes another letter.
- A prefix code for a set of letters is a function γ that maps each letter x in S to some sequence of zeros and ones, in such a way that for distinct x, y in S, the sequence of $\gamma(x)$ is not a prefix of the sequence $\gamma(y)$.

Prefix codes: Encoding & Decoding

- How to encode using a prefix code γ:
  - Input: a text x1 x2 x3 …… xn
  - Output: γ(x1) γ(x2) γ(x3)……γ(xn)

- How to decode a message encoded by the prefix code γ :
  - Scan the bit sequence of the input message from left to right.
  - As soon as you've seen enough bits to match the encoding, say γ(x), of some letter x, output this letter x as the first letter of the original text.
  - Now, continue scanning the bit sequence and iterate the above procedure until no more bits

Example of Prefix code

- Prefix code γ for Alphabet S:
  - S = {a, b, c, d, e}
  - $\gamma(a)=11$, $\gamma(b)=01$, $\gamma(c)=001$, $\gamma(d)=10$, $\gamma(e)=000$.
  - Then, γ is a prefix code since any $\gamma(x)$ is not a prefix of any other.
- How to encode: the message “cecab”:
  - $\gamma(c)\gamma(e)\gamma(c)\gamma(a)\gamma(b)$ =001000001101
- How to decode: the encoded message “001000001101”
  - Scan the bits from left to right.
    - The first bit is 0, so the candidate letter is one of {b,c,e}.
    - The next bit is 0, so the candidate letter is one of {c,e}
    - The next bit is 0, so the letter is ‘e’
  - Output ‘e’, which is the first letter of the original message.
  - The remaining message is “0000011101”.
  - And, Iterate.

Example

Char	Frequency	Code 1	Code 2
a	5	000	11
b	4	001	10
c	3	010	00
d	2	011	011
e	1	100	010

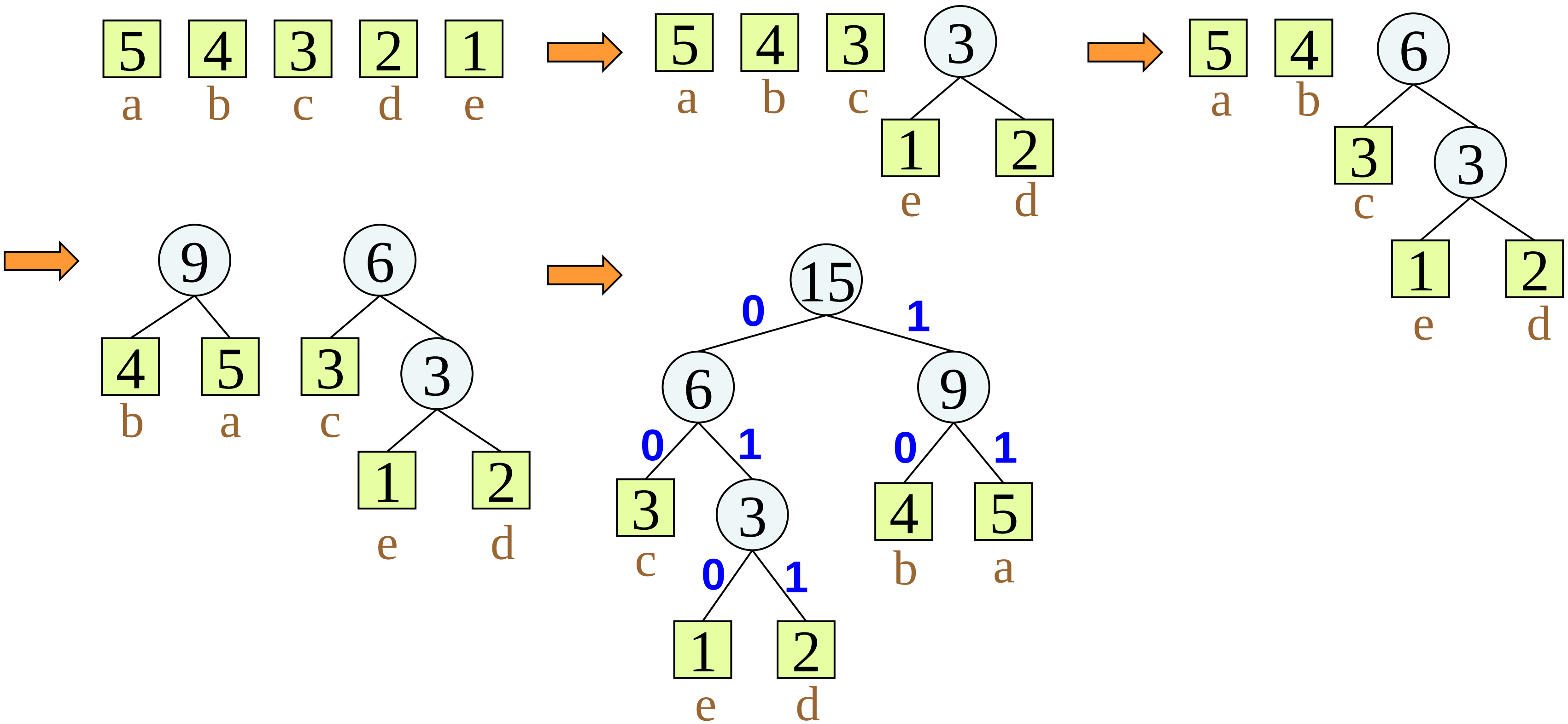
- Message:
  a b a a b c d e a d a b c c b

- Encoding by Code 1
000	001	000	000	001
a	b	a	a	b

010	011	100	000	011	000	001	010	010	001
c	d	e	a	d	a	b	c	c	b
  - Total Length = 3·15 = 45(bits)

- Encoding by Code 2
11	10	11	11	10	00	011	010	11	011	11	10	00	00	10
a	b	a	a	b	c	d	e	a	d	a	b	c	c	b
  - Total Length = 2·5 + 2·4 + 3·3 + 3·2 + 3·1 = 33(bits)

Constructing Optimal 2-Way Merge Tree



- Time Complexity: O(n log n) for n external nodes
  - Use Min Heap for selecting two minimums and inserting the sum: O(log n)

Greedy !

- Greedy Template:
  - Minimum Weight First

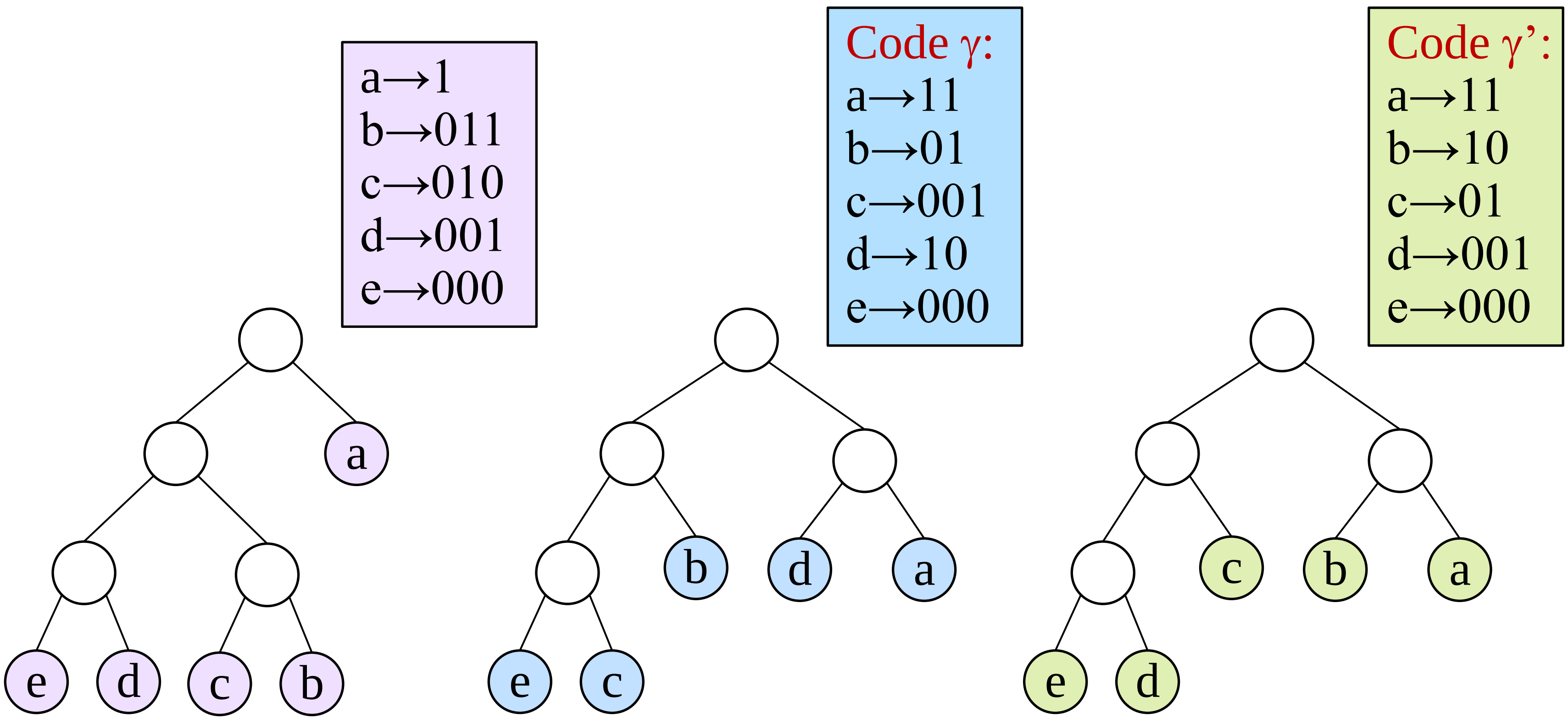
- How to prove Optimality
  - Exchange Argument

Prefix codes Using Binary Trees

- A rooted binary tree T:
  - The number of leaves is equal to the size of the alphabet S.
  - Each leaf is labeled with a distinct letter of S.
  - Left path is labeled with 0, and right path is labeled with 1.

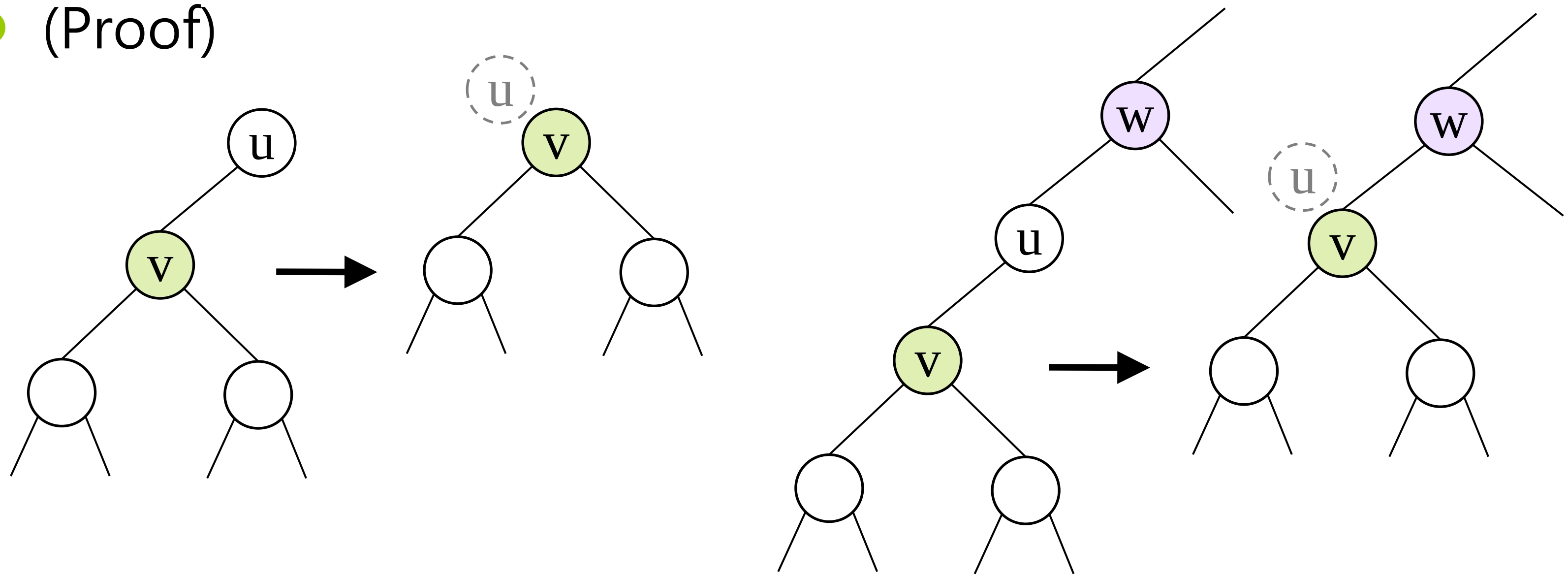
Prefix codes Using Binary Trees

- Theorem: The encoding of S constructed from T is a prefix code.



Prefix codes Using Binary Trees

- A binary tree is full if each node that is not a leaf has 2 children.
- Theorem: The binary tree corresponding to the optimal prefix code is full.
  - (Proof)



The change decreases the number of bits to encode any leaf on the subtree rooted at u.

It Contradicts the optimality of the given tree. ▪

What If We Knew The Tree Structure of the Optimal Prefix Code?

- Theorem: Suppose that u and v are leaves of $T*$, such that depth(u)< depth(v). Suppose that leaf u is labeled with y and leaf v is labeled with z.
  Then $f_y \geq f_z$.
  - (Proof) A quick proof using an Exchange Argument.
    If $f_y < f_z$, exchange the labels at the nodes u and v.
    The overall sum: $ABL(T*) = \sum_{x \in S} f_x * depth(x)$.
    For label y, f_y*depth(u) is changed to f_y*depth(v),
    so the change in ABL is (depth(v) − depth(u))*f_y.
    For label z, f_z*depth(v) is changed to f_z*depth(u),
    so the change in ABL is (depth(u) − depth(v))*f_z.
    Therefore, the change to the overall sum is
    (depth(v) − depth(u))(f_y − f_z).
    If $f_y < f_z$, this change is negative, contradiction. ▪

End
of
Optimal 2-Way Merge Pattern