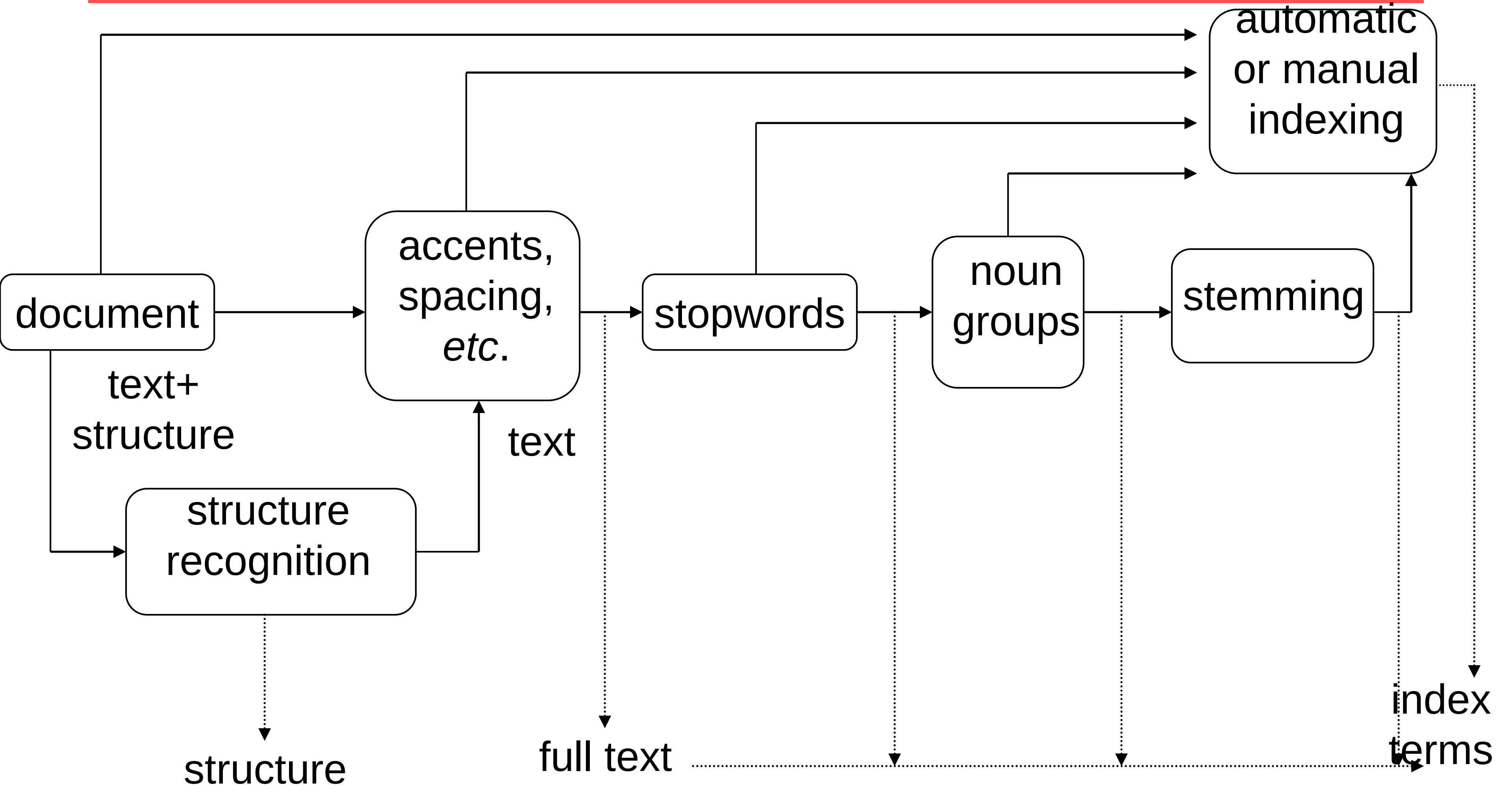
Chapter 6
Text Operations

Logical View of a Document



Text Operations

- Lexical analysis of the text
- Elimination of stopwords
- Stemming
- Selection of index terms
- Construction of term categorization structures

Lexical Analysis of the Text

- Word separators
  - space
  - digits
  - hyphens
  - punctuation marks
  - the case of the letters

Lexical Analysis for Automatic Indexing

- **Lexical Analysis**
  Convert an input stream of characters into stream ***words*** *or* ***token****.*
- **What is a word or a token?**
  Tokens consist of letters.
  - digits: Most numbers are not good index terms.
    counterexamples: case numbers in a legal database, “B6” and “B12” in vitamin database.

Lexical Analysis for Automatic Indexing
(Continued)

- hyphens
  - break hyphenated words: state-of-the-art, state of the art
  - keep hyphenated words as a token: “Jean-Claude”, “F-16”
- punctuation marks: often used as parts of terms, e.g., OS/2, 510B.C.
- case: usually not significant in index terms

Elimination of Stopwords

- A list of stopwords
  - words that are too frequent among the documents
  - articles, prepositions, conjunctions, etc.
- Can reduce the size of the indexing structure considerably
- Problem
  - Search for "**to be** or **not to be**"?

Stopword

- Avoid retrieving almost very item in a database regardless of its relevance.
- Examples
  - conservative approach (ORBIT Search Service): *and, an, by, from, of, the, with*
  - (derived from Brown corpus): 425 words *a, about, above, across, after, again, against, all, almost, alone, along, already, also, although, always, among, an, and, another, any, anybody, anyone, anything, anywhere, are, area, areas, around, as, ask, asked, asking, asks, at, away, b, back, backed, backing, backs, be, because, became, ...*
- Articles, prepositions, conjunctions, …

Stemming

- Example
  - *connect, connected, connecting, connection, connections*
  - effectiveness --> effective --> effect
  - picnicking --> picnic
- Removing strategies
  - affix removal: intuitive, simple
  - table lookup
  - successor variety
  - n-gram

Stemmers

- programs that relate morphologically similar indexing and search terms
- stem at indexing time
  - advantage: efficiency and index file compression
  - disadvantage: information about the full terms is lost
- example (CATALOG system), *stem at search time*

  Look for: system users

  Search Term: users

Term	Occurrences
1. user	15
2. users	1
3. used	3
4. using	2

The user selects the terms he wants by numbers

Conflation Methods

- manual
- automatic (stemmers)
  - affix removal
    longest match vs. simple removal
  - successor variety
  - table lookup
  - n-gram
- evaluation
  - correctness
  - retrieval effectiveness
  - compression performance

Term	Stem
engineering	engineer
engineered	engineer
engineer	engineer

Successor Variety

- **Definition**

  the number of different characters that follow it in words in some body of text

- Example

  a body of text: *able, axle, accident, ape, about*

  successor variety of *apple*

  1st: 4 (*b*, *x*, *c*, *p*)

  2nd: 1 (*e*)

Successor Variety *(Continued)*

- **Idea**
  The successor variety of substrings of a term will decrease as more characters are added until a segment boundary is reached, i.e., the successor variety will ***sharply increase***.
- **Example**
  Test word: READABLE
  Corpus: ABLE, BEATABLE, FIXABLE, READS, READABLE, READING, RED, ROPE, RIPE

Prefix	Successor Variety	Letters
R	3	E, O, I
RE	2	A, D
REA	1	D
READ	**3**	**A, I, S**
READA	1	B
READAB	1	L
READABL	1	E
READABLE	1	blank

n-gram stemmers

- **diagram**
  a pair of consecutive letters
- **shared diagram method**
  association measures are calculated between pairs of terms

$$S = \frac{2C}{A + B}$$

where A: the number of unique diagrams in the first word,
B: the number of unique diagrams in the second,
C: the number of unique diagrams shared by A and B.

n-gram stemmers *(Continued)*

- Example

  statistics => st ta at ti is st ti ic cs

  unique diagrams => at cs ic is st ta ti

  statistical => st ta at ti is st ti ic ca al

  unique diagrams => al at ca ic is st ta ti

$$S = \frac{2C}{A+B} = \frac{2*6}{7+8} = 0.80$$

n-gram stemmers *(Continued)*

- **similarity matrix**
  determine the semantic measures for all pairs of terms in the database

	$word_1$	$word_2$	$word_3$	...	$word_{n-1}$
$word_1$					
$wrod_2$	S_{21}				
$word_3$	S_{31}	S_{32}			
.					
.					
$word_n$	S_{n1}	S_{n2}	S_{n3}	…	$S_{n(n-1)}$

- terms are clustered using a single link clustering method
- more a term clustering procedure than a stemming one

Affix Removal Stemmers

- **procedure**
  Remove suffixes and/or prefixes from terms leaving a stem, and transform the resultant stem.
- **example**: plural forms
  If a word ends in “ies” but not “eies” or “aies”
  then “ies” --> “y”
  If a word ends in “es” but not “aes”, “ees”, or “oes”
  then “es” --> “e”
  If a word ends in “s”, but not “us” or “ss”
  then “s” --> NULL

Index Terms Selection

- Motivation
  - A sentence is usually composed of nouns, pronouns, articles, verbs, adjectives, adverbs, and connectives.
  - Most of the semantics is carried by the **noun** words.
- Identification of noun groups
  - A noun group is a set of nouns whose syntactic distance in the text does not exceed a predefined threshold

Index Terms Selection

- Indexing by single words
  - single words are often ambiguous and not specific enough for accurate discrimination of documents
  - ***bank terminology* vs. *terminology bank***
- Indexing by phrases
  - Syntactic phrases are almost always more specific than single words
- Indexing by single words and phrases

Thesauri

- Peter Roget, 1988
- Example

  cowardly *adj.*

  Ignobly lacking in courage: cowardly turncoats

  Syns: chicken (slang), chicken-hearted, craven, dastardly, faint-hearted, gutless, lily-livered, pusillanimous, unmanly, yellow (slang), yellow-bellied (slang).
- A controlled vocabulary for the indexing and searching

The Purpose of a Thesaurus

- To provide a standard vocabulary for indexing and searching
- To assist users with locating terms for proper query formulation
- To provide classified hierarchies that allow the broadening and narrowing of the current query request

Functions of thesauri

- Provide a standard vocabulary for indexing and searching
- Assist users with locating terms for proper query formulation
- Provide classified hierarchies that allow the broadening and narrowing of the current query request

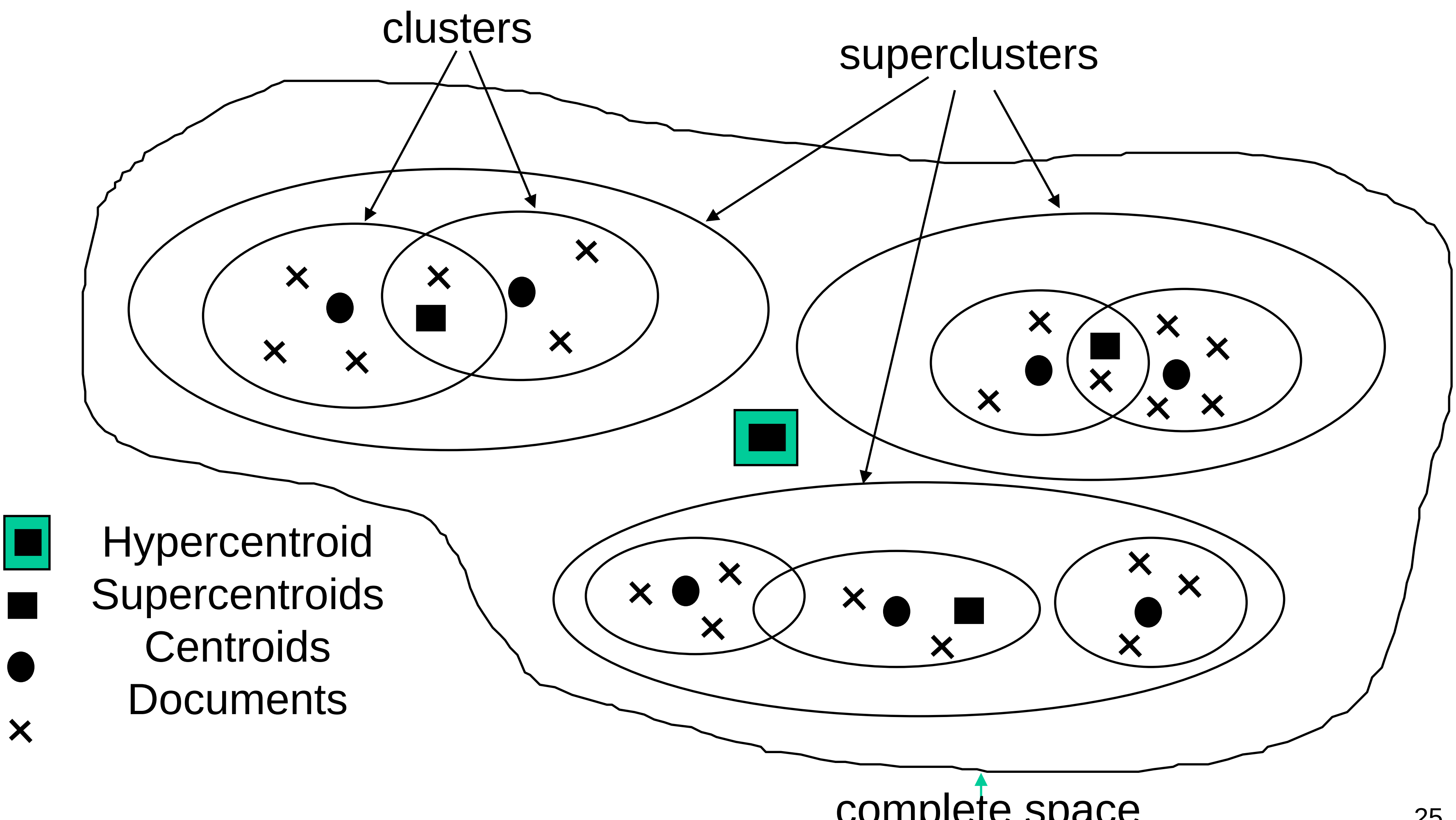
Usage

- **Indexing**
  Select the most appropriate thesaurus entries for representing the document.
- **Searching**
  Design the most appropriate search strategy.
  - If the search does **not** retrieve **enough** documents, the thesaurus can be used to **expand the query**.
  - If the search retrieves **too many** items, the thesaurus can suggest **more specific search vocabulary**.

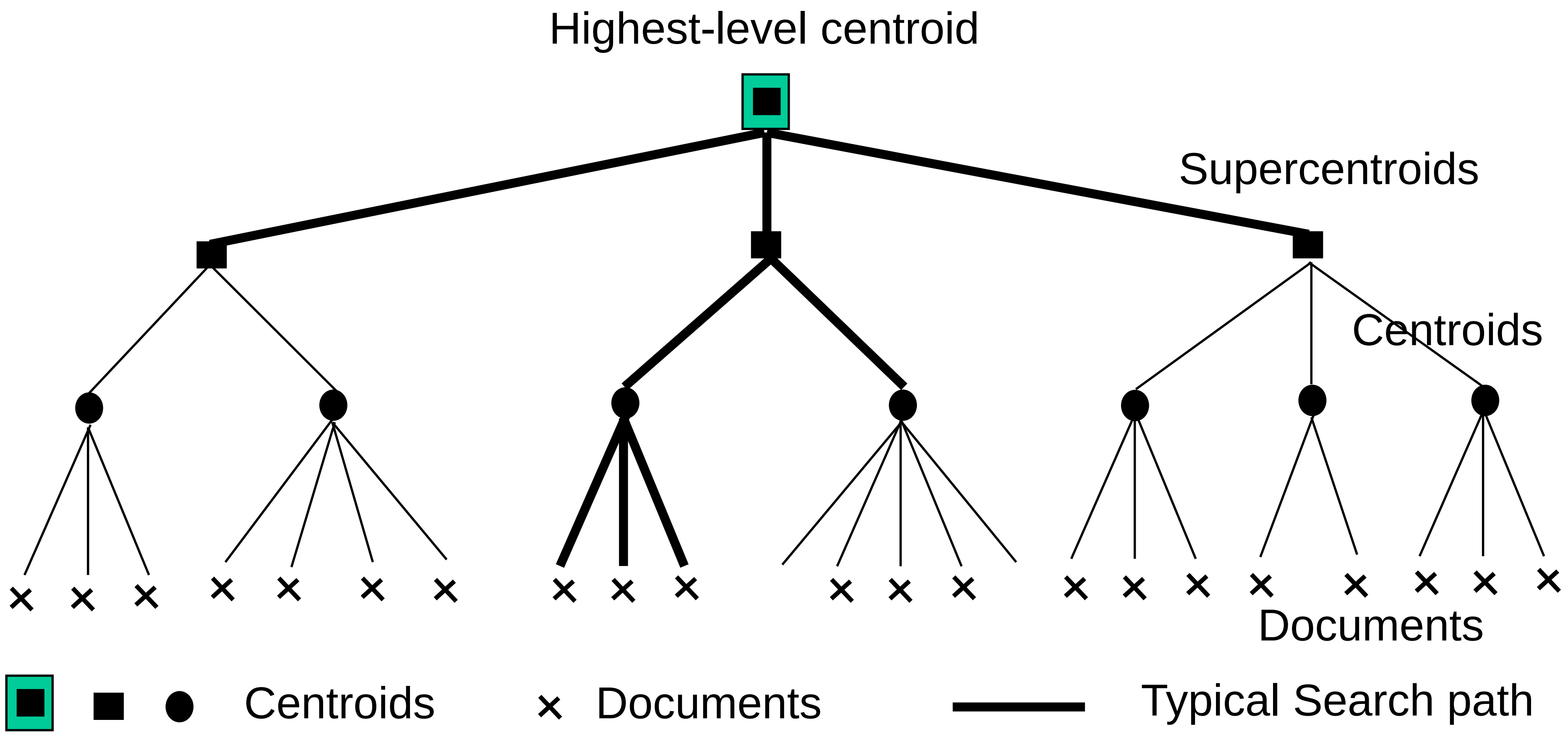
Document Clustering

- Global clustering
  - The grouping of documents accordingly to their occurrence in the whole collection
- Local clustering:
  - The grouping of the local set of retrieved documents by a query

Typical Clustered File Organization



Search Strategy for Clustered Documents



Text Compression

- Finding ways to represent the text in fewer bits or bytes
- Encode/Decode
- Low data communication

Text Compression

- Lossless Compression
  - Text
- Lossy Compression
  - Multimedia

All case **Speed** is important !!!

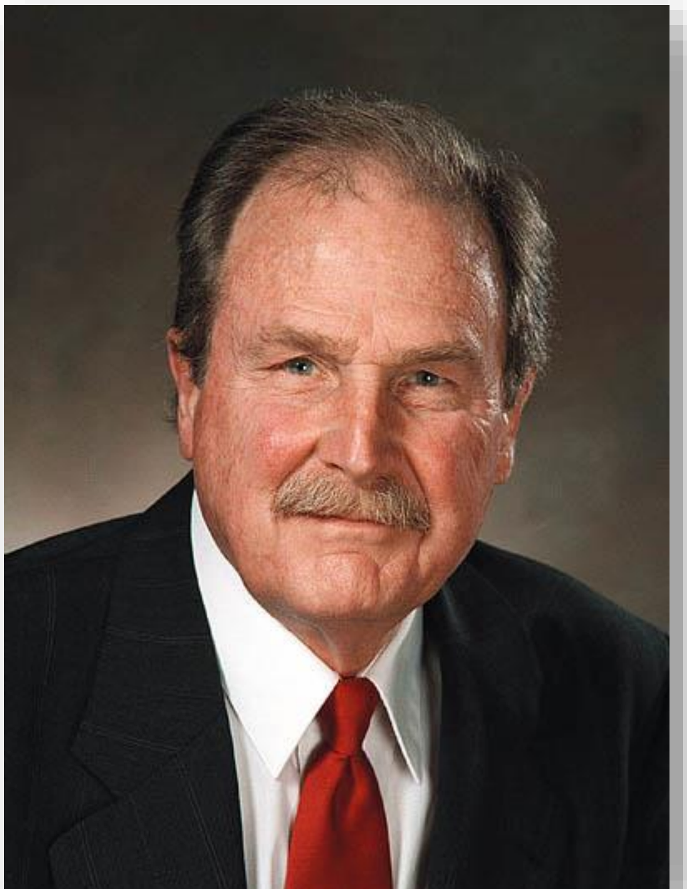
<u>Speed</u>

- Frequency Compute
- Tree Creation
- Dictionary Creation/Using

Text Compression

- Statistical Methods
  - Huffman coding
- Dictionary-based
  - Ziv-Lempel

Huffman Coding



Huffman Coding

จัดทำโดย David A. Huffman

เป็นวิธีการเข้ารหัส Entropy ชนิดหนึ่ง

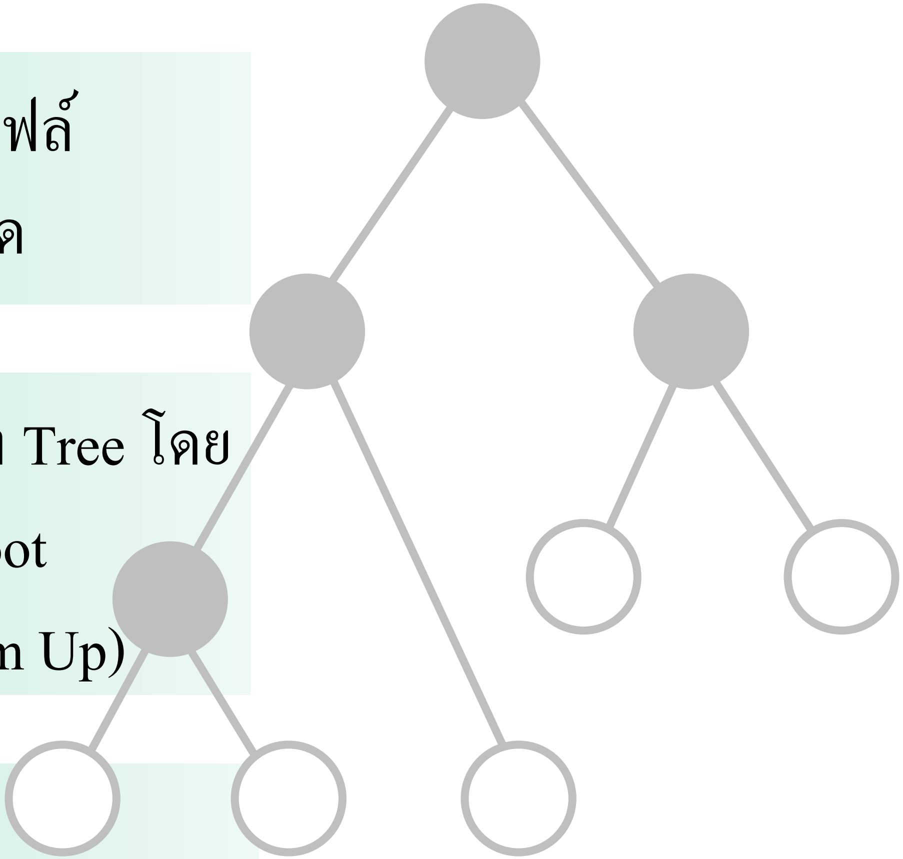
ที่ใช้ในการบีบอัดข้อมูลแบบ

lossless data compression

Huffman Coding

Huffman Coding เป็นวิธีการบีบอัดไฟล์ ที่ให้ผลลัพธ์มีขนาดเล็กลงมากที่สุด

วิธีการของ Huffman นั้น ทำการสร้าง Tree โดยเริ่มจาก Leaf Node ไปหา Root หรือสร้างจากล่างขึ้นบน (Bottom Up)



รหัสที่สร้างโดยวิธีของ Huffman จะเป็นรหัสที่ดีที่สุดเสมอ

การสร้าง Huffman Coding

Huffman Tree หรือ Binary Tree ของรหัส

สร้างโดยใช้**ความถี่**ในการเกิด
ของแต่ละตัวอักษรในเอกสาร

ตัวอักษรที่มีความถี่มาก
จะใช้จำนวน Bit น้อยที่สุด

การสร้าง Huffman Coding

หาความถี่ของตัวอักษร (Character) แต่ละตัว
ที่ปรากฏในเอกสารแล้วเก็บไว้ในคิว (Queue)

นำสองตัวที่มีค่าน้อยที่สุดออกจากคิว
ตัวที่มีค่าน้อยสุดอยู่กิ่งซ้าย ส่วนอีกตัวอยู่กิ่งขวา

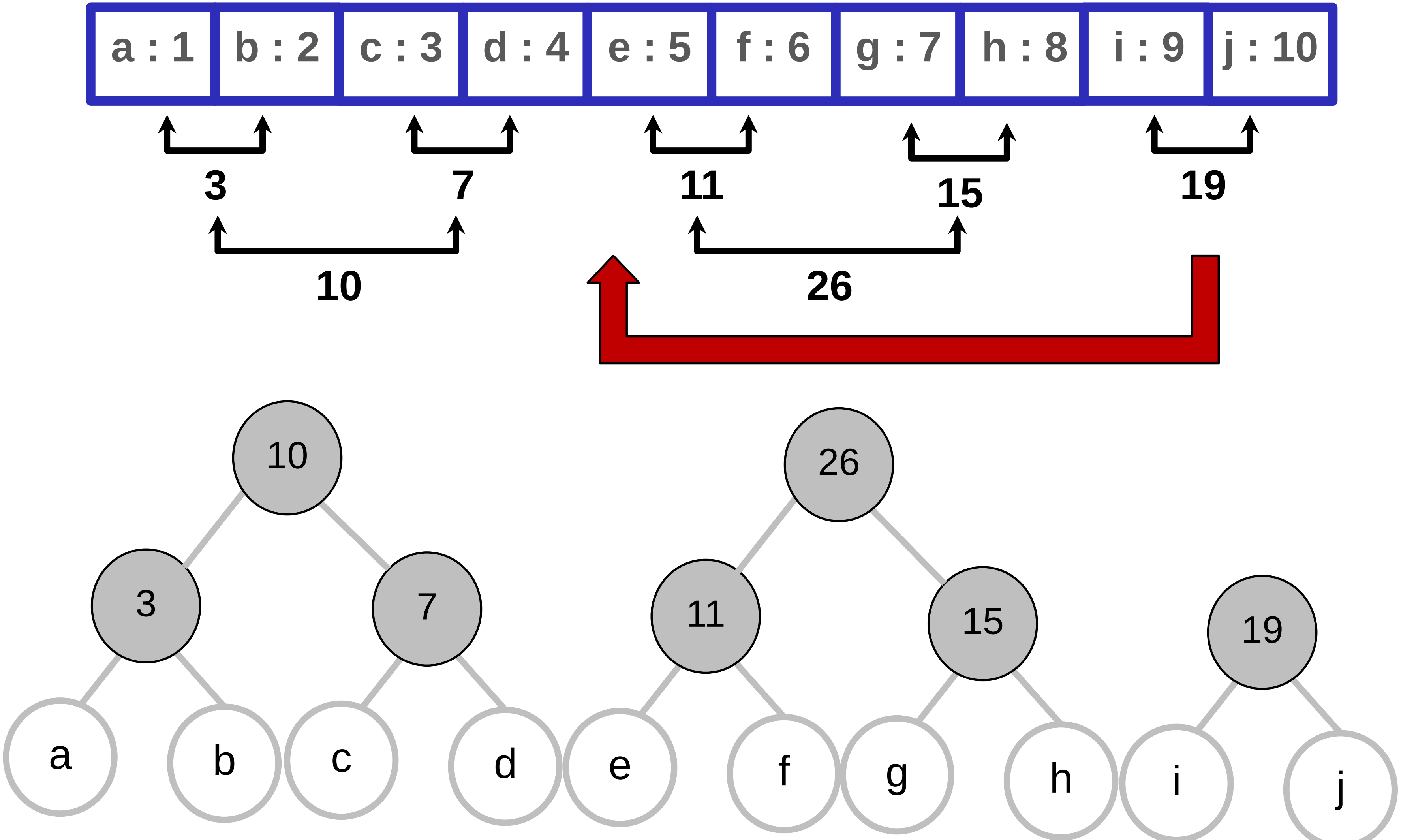
นำผลบวกของทั้งสองตัวมาเป็น Root แล้วไปเก็บในคิว

ทำซ้ำข้อ 2 และ 3 จนทั้งหมดรวมเป็น Tree เดียวกัน

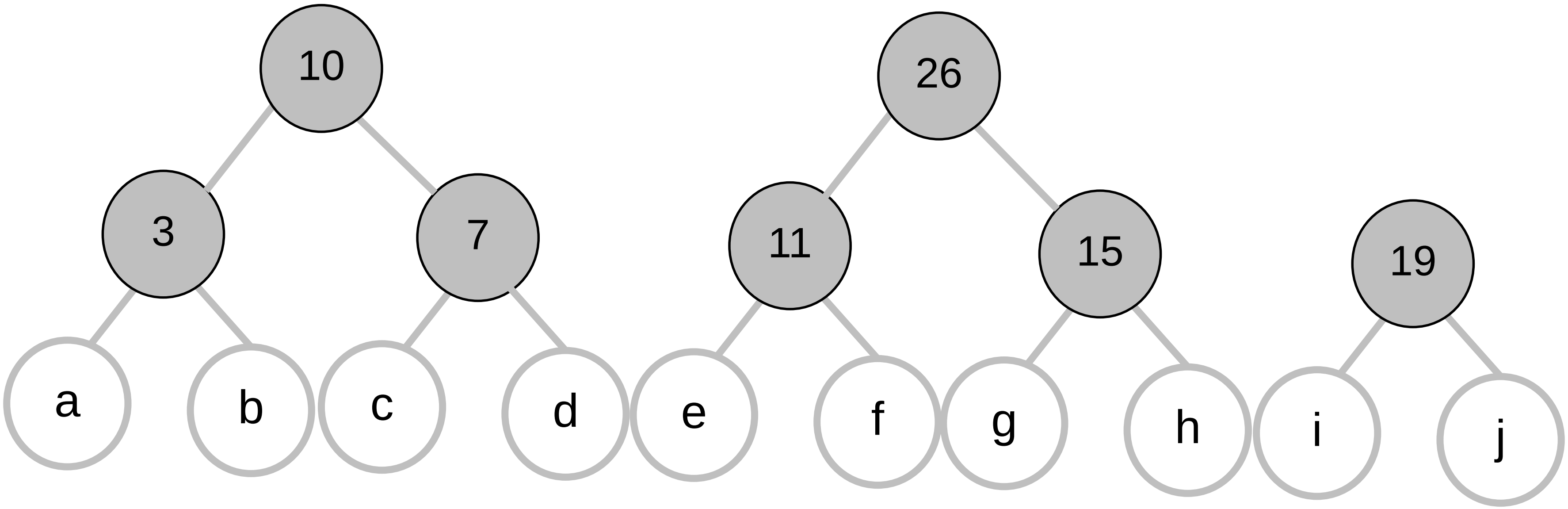
Assign ค่าให้กับแต่ละ Character

Huffman Coding Example

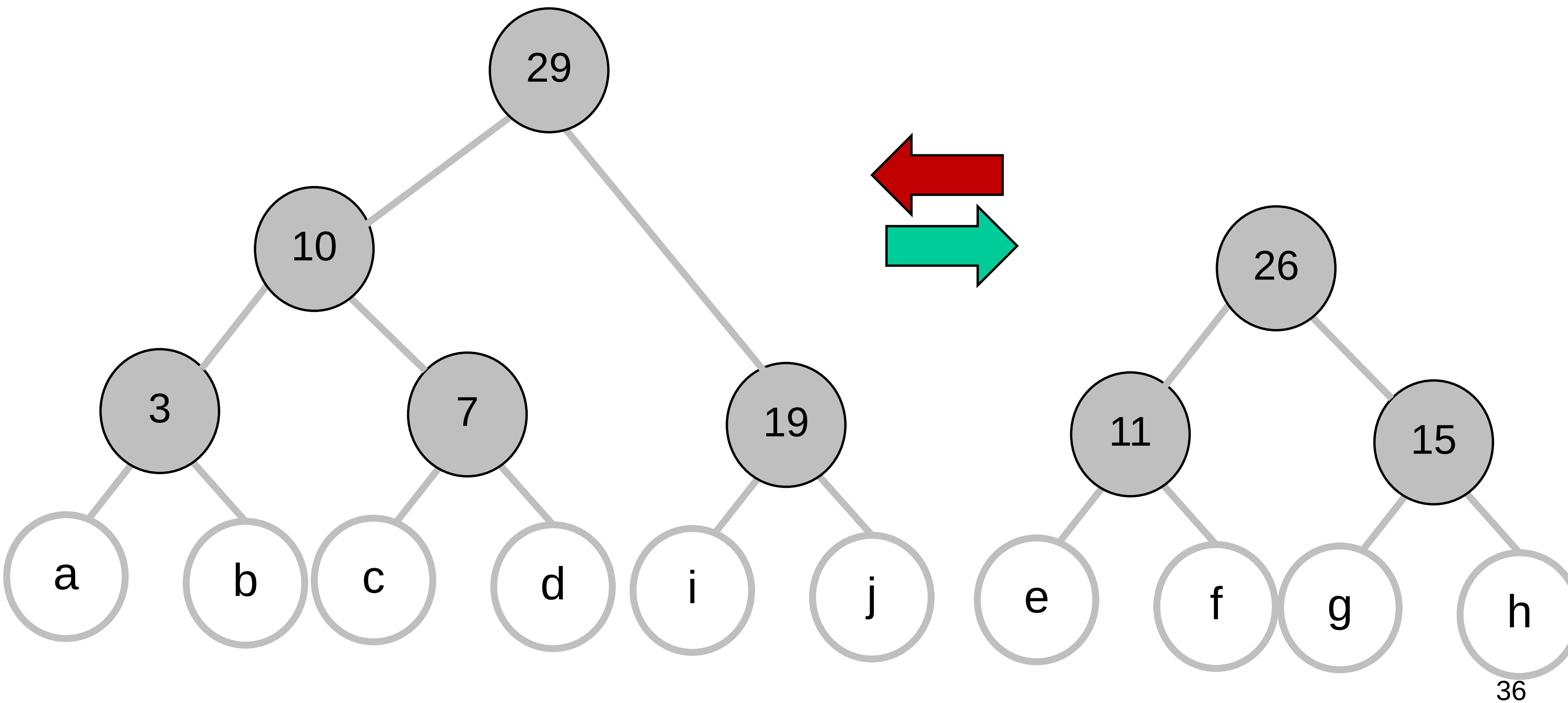
Case 1



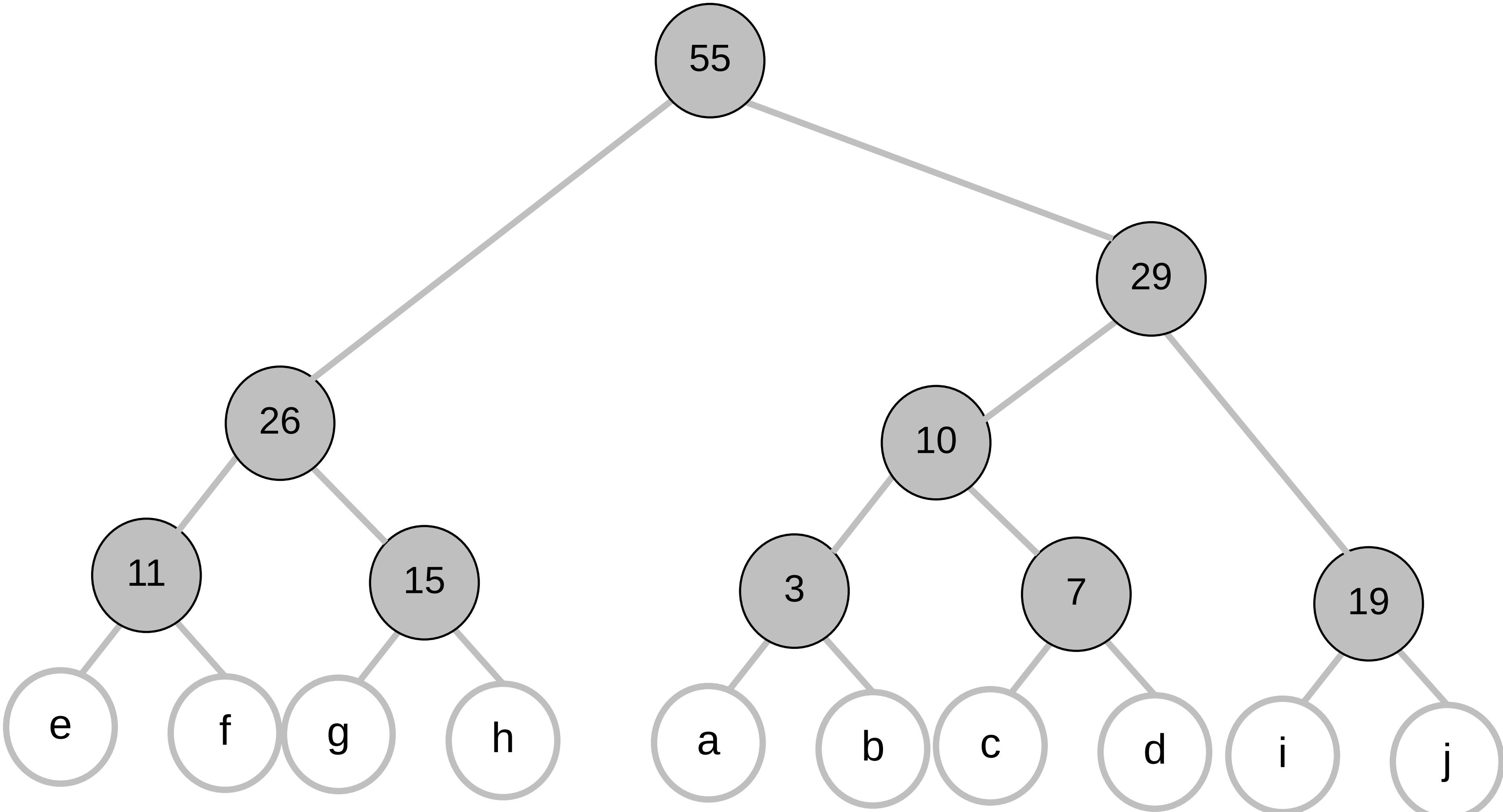
Huffman Coding Example



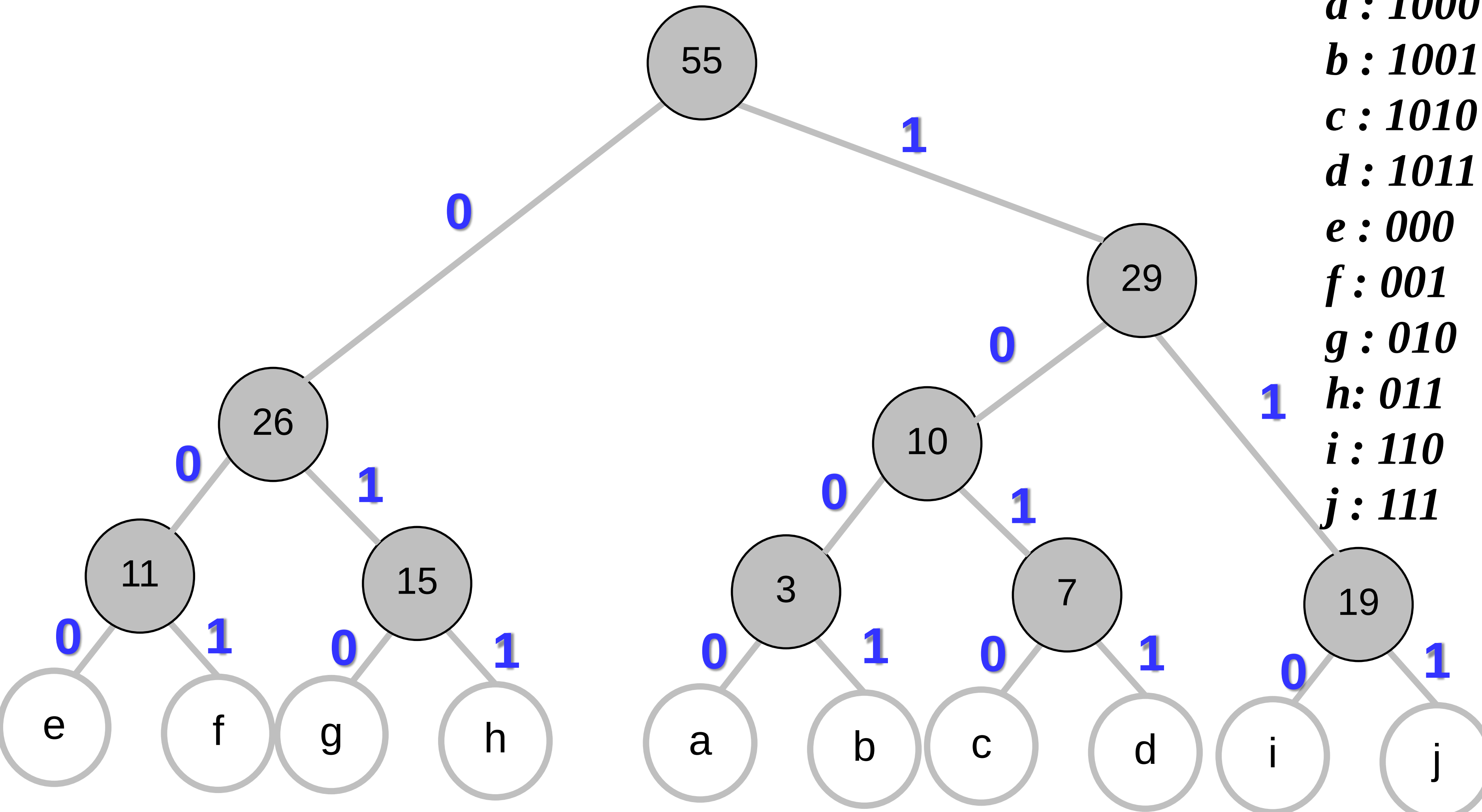
Huffman Coding Example



Huffman Coding Example



Huffman Coding Example



a : 1000
b : 1001
c : 1010
d : 1011
e : 000
f : 001
g : 010
h: 011
i : 110
j : 111

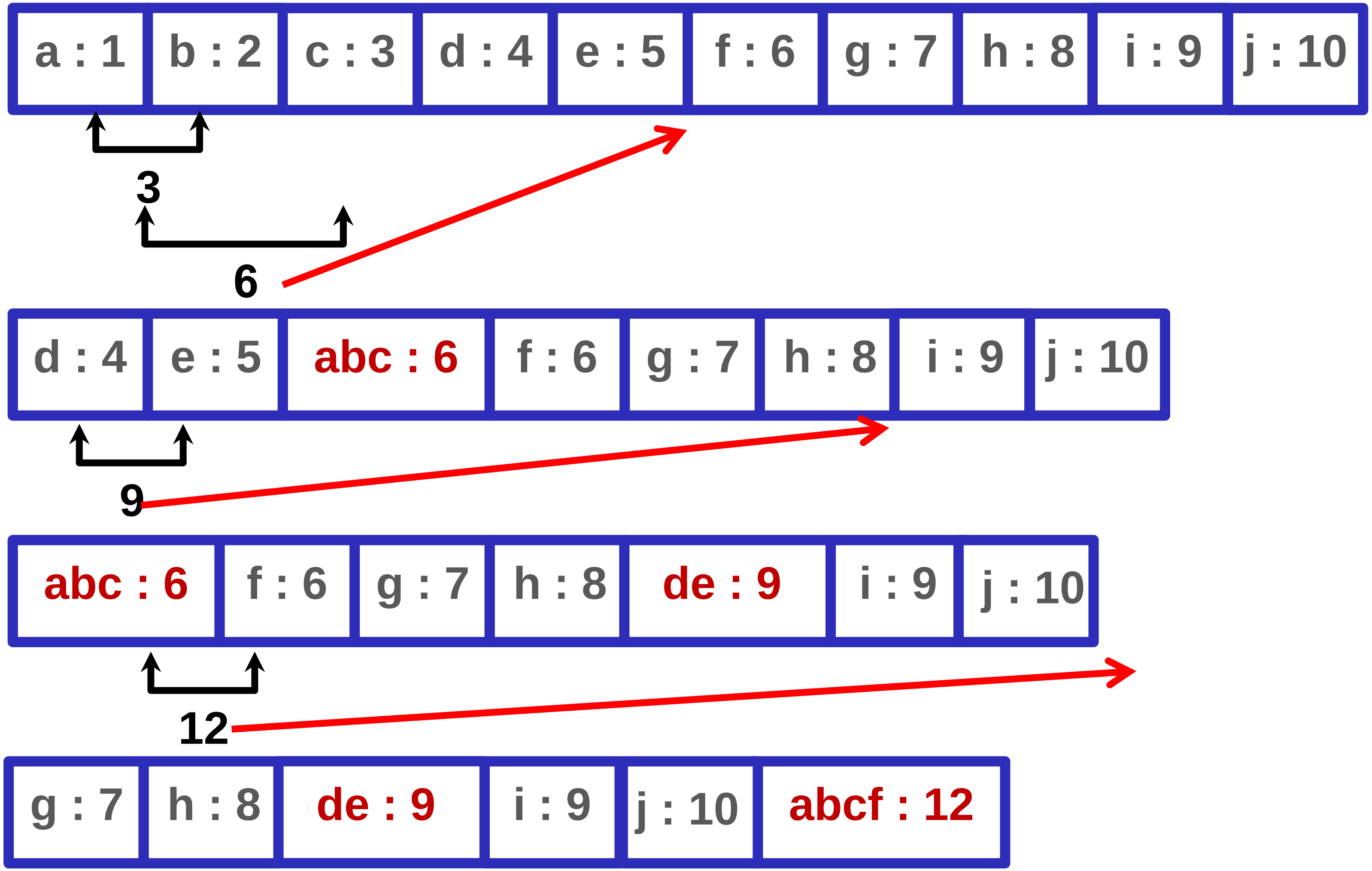
Huffman Coding Example

		Normal
a : 1000	4x1	4x1
b : 1001	4x2	4x2
c : 1010	4x3	4x3
d : 1011	4x4	4x4
e : 000	3x5	4x5
f : 001	3x6	4x6
g : 010	3x7	4x7
h : 011	3x8	4x8
i : 110	3x9	4x9
j : 111	3x10	4x10
$\frac{175}{55} = 3.182$	175	220

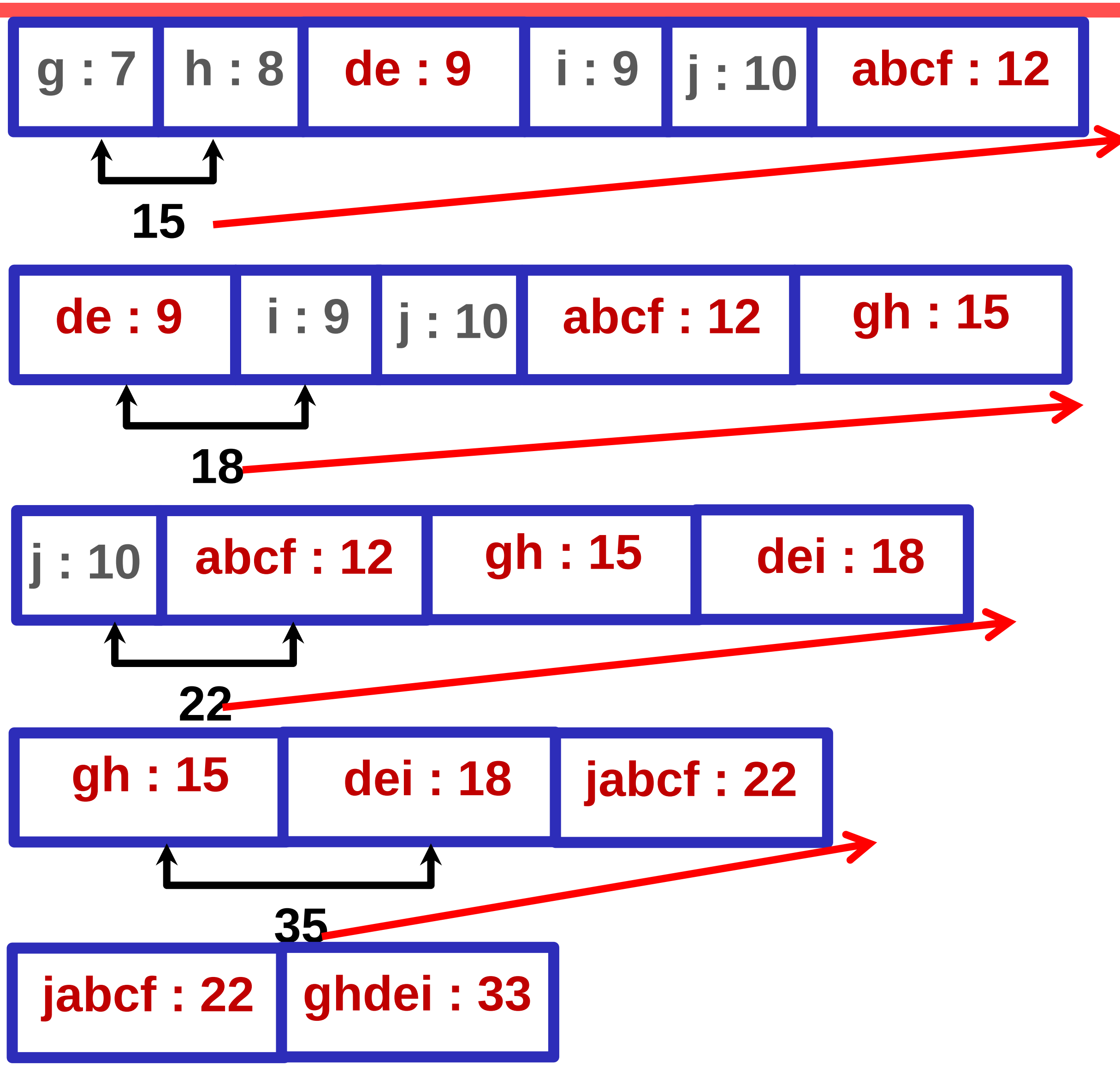
$$\text{Save} = \frac{(220-175)*100}{220} = 20.45\ \%$$

Huffman Coding Example

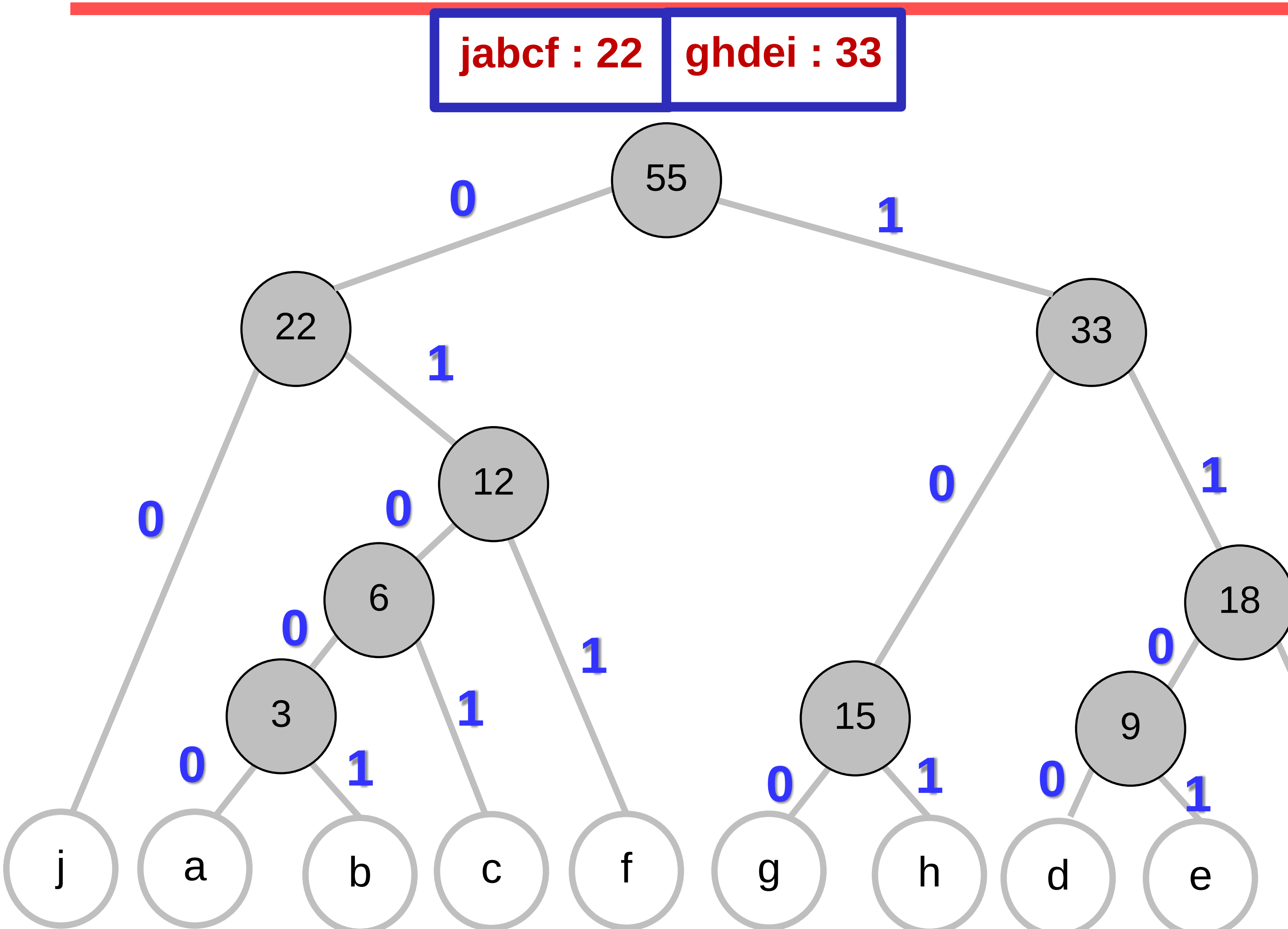
Case 2



Huffman Coding Example



Huffman Coding Example



a : 01000
b : 01001
c : 0101
d : 1100
e : 1101
f : 011
g : 100
h: 101
i : 111
j : 00

Huffman Coding Example

		Normal
a : 01000	5x1	4x1
b : 01001	5x2	4x2
c : 0101	4x3	4x3
d : 1100	4x4	4x4
e : 1101	4x5	4x5
f : 011	3x6	4x6
g : 100	3x7	4x7
h : 101	3x8	4x8
i : 111	3x9	4x9
j : 00	2x10	4x10
$\frac{173}{55} = 3.145$	173	220

$$\text{Save} = \frac{(220-173)*100}{220} = 21.36\ \%$$

Huffman Coding Example

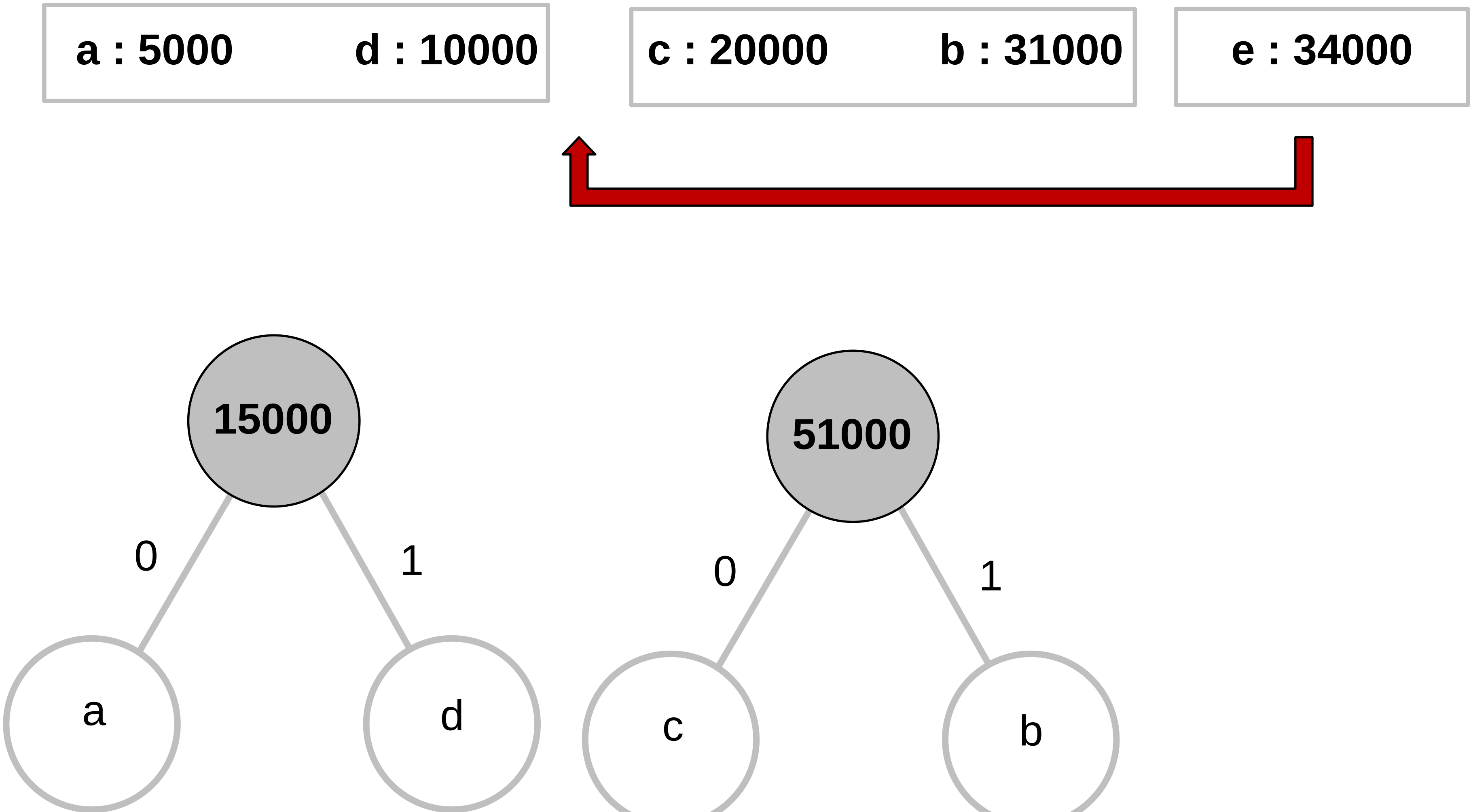
หาความถี่ของตัวอักษรแต่ละตัว

CHARACTER	ความถี่ที่ปรากฏ
a	5000
d	10000
c	20000
b	31000
e	34000

จากนั้นนำมาใส่ในคิว Queue

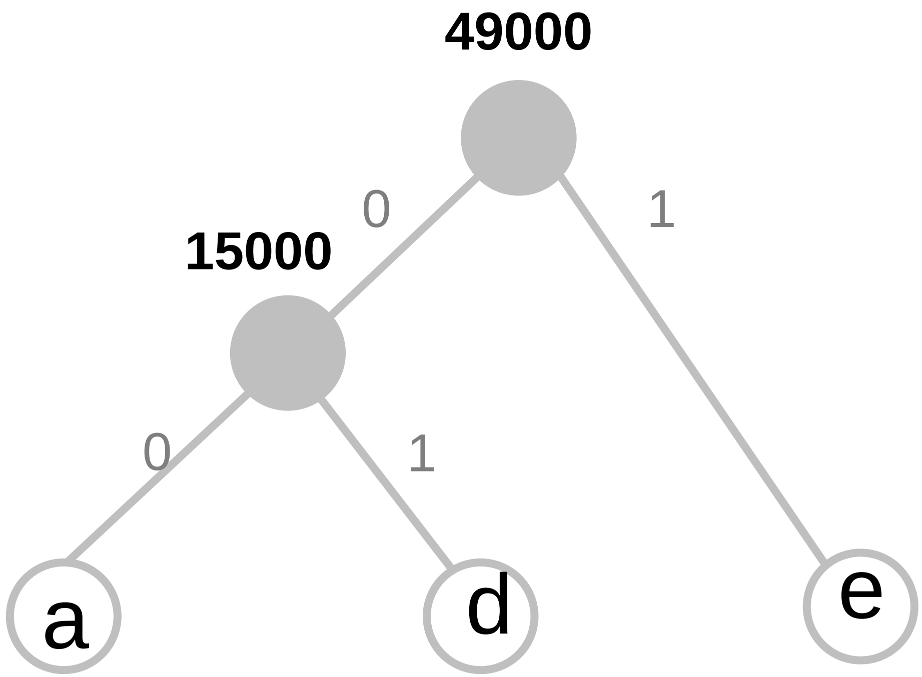
a : 5000	d : 10000	c : 20000	b : 31000	e : 34000

Huffman Coding Example



Huffman Coding Example

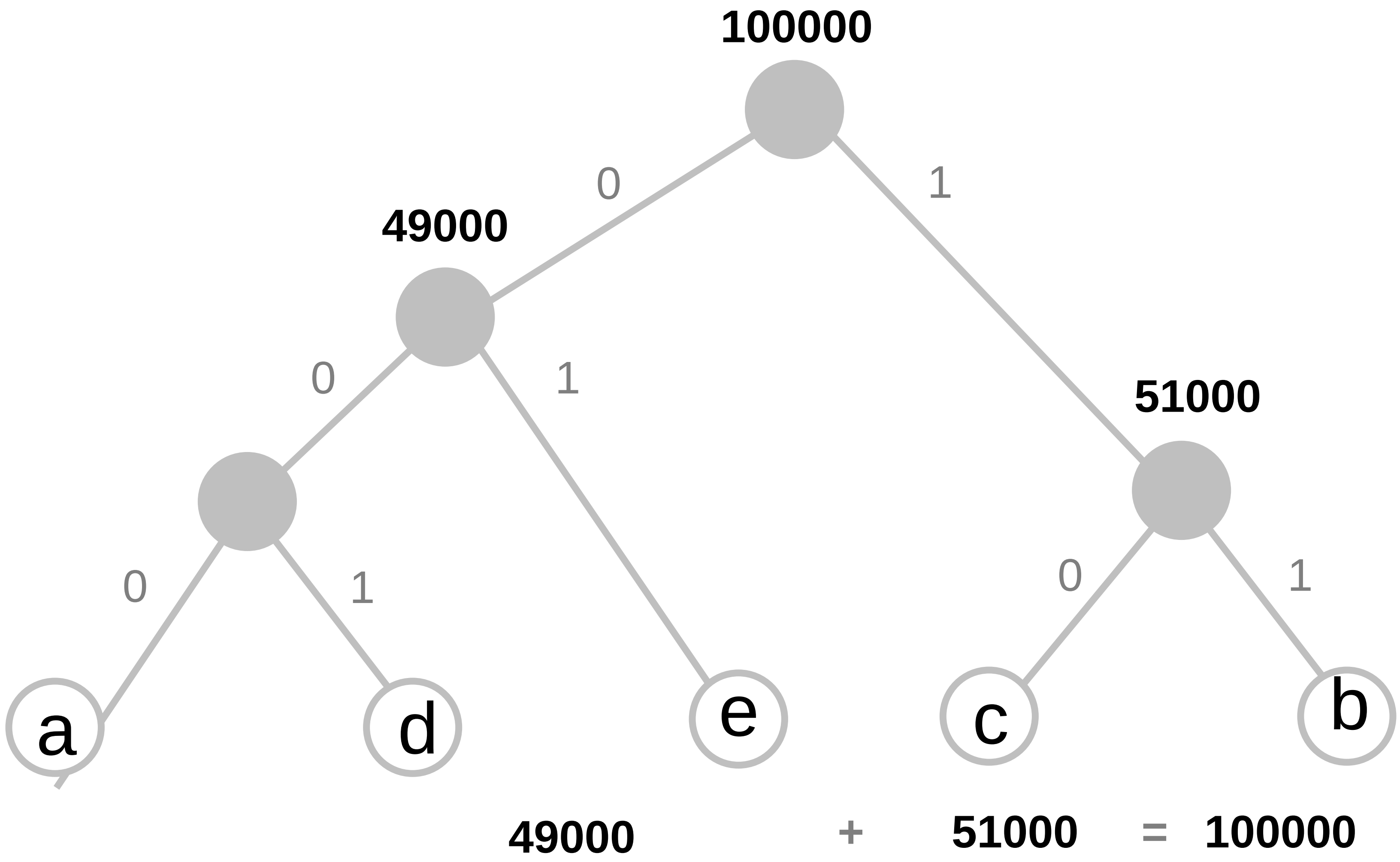
ad : 15000	e : 34000	cb : 51000



15000 + **34000** = **49000**

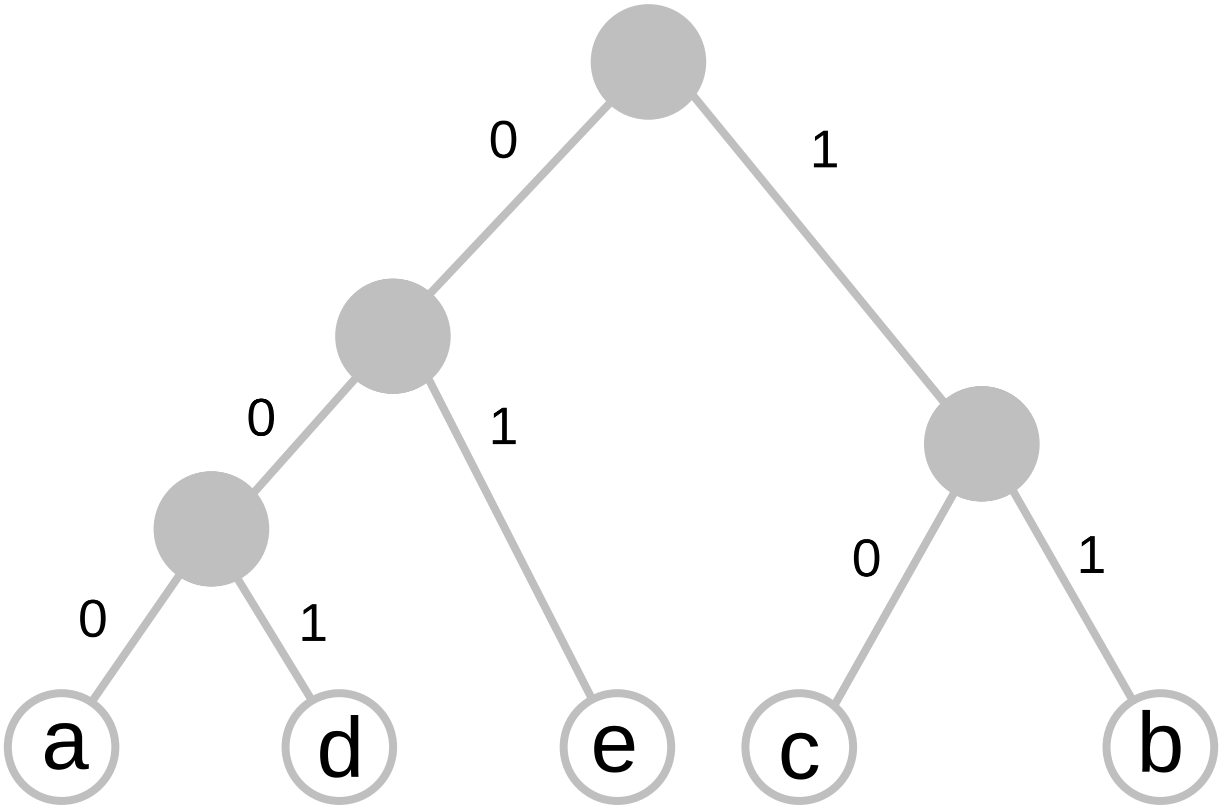
Huffman Coding Example

adecb : 100000



Huffman Coding Example

กำหนดค่าให้กับแต่ละตัวอักษรได้เป็น Code ของแต่ละตัว



a	:	**000**
b	:	**11**
c	:	**10**
d	:	**001**
e	:	**01**

Encoder

ENCODER

a	:	000
b	:	11
c	:	10
d	:	001
e	:	01

e a b a d c e d

SEND

01 000 11 000 001 10 01 001

ติดกัน

Huffman Code

Decoder

RECIEVE

Huffman Code

010001100000110000 1

DECODER

000	:	**a**
11	:	**b**
10	:	**c**
001	:	**d**
01	:	**e**

e a b a d c e d

การเข้ารหัสโดยไม่ใช้ Huffman

จากตัวอย่างข้างต้นมีตัวอักษรที่ปรากฏทั้งหมด 5 ตัว ซึ่งถ้าหากทำการเข้ารหัสโดยไม่ได้ใช้ Huffman Code จะต้องแทนค่าของแต่ละตัวอักษรเป็น 3 บิต ดังนี้

a	:	**000**
b	:	**001**
c	:	**010**
d	:	**011**
e	:	**100**

e a b a d c e d

100 000 001 000 011 010 100 011

การเข้ารหัสโดยไม่ใช้ Huffman

Character	Frequency	Probability	Binary Code	Code Length
a	5000	0.05	000	3
b	31000	0.31	001	3
c	20000	0.20	010	3
d	10000	0.10	011	3
e	34000	0.34	100	3

จำนวนบิตที่ใช้ในการแทนตัวอักษร $\frac{1}{f(T)} \times \sum_{i=0}^{i=n} d(i) * f(i)$

$$= \frac{(5000 \times 3) + (31000 \times 3) + (20000 \times 3) + (10000 \times 3) + (34000 \times 3)}{100000} = 3$$

หรือ =(0.05)*3 + (0.31)*3 + (0.20)*3 + (0.10)*3 + (0.34)*3 = 3 Bit

การเข้ารหัสโดยใช้ Huffman

a	:	000
b	:	11
c	:	10
d	:	001
e	:	01

e a b a d c e d

01000110000011001001

การเข้ารหัสโดยใช้ Huffman

Character	Frequency	Probability	Binary Code	Code Length
a	5000	0.05	000	3
b	31000	0.31	11	2
c	20000	0.20	10	2
d	10000	0.10	001	3
e	34000	0.34	01	2

จำนวนบิตที่ใช้ในการแทนตัวอักษร $\frac{1}{f(T)} \times \sum_{i=0}^{i=n} d(i) * f(i)$

$$= \frac{(5000 \times 3) + (31000 \times 2) + (20000 \times 2) + (10000 \times 3) + (34000 \times 2)}{100000} = 2.15$$

หรือ =(0.05)*3 + (0.31)*2 + (0.2)*2 + (0.1)*3 + (0.34)*2 = **2.15 Bit**

สรุป

e a b a d c e d	∑ Probability * CodeLength
การเข้ารหัสโดยไม่ใช้ Huffman	3 Bit
การเข้ารหัสโดยใช้ Huffman	2.15 Bit

Huffman Coding Example

		Normal
a : 000	3x5000	3x5000
b : 11	2x31000	3x31000
c : 10	2x20000	3x20000
d : 001	3x10000	3x10000
e : 01	2x34000	3x34000
	215000	300000

$$\text{Save} = \frac{(300000-215000)*100}{300000} = 28.33\ \%$$

$$\text{Save} = \frac{(3-2.15)*100}{3} = 28.33\ \%$$

Text Compression

- Dictionary-based
  - Ziv-Lempel

Ziv Lempel Welch

- **Ziv-Lempel Coding หรือ Lempel-Ziv-Welch หรือ LZW**
- **คิดค้นโดย Abraham Lempel และ Jacob Ziv ในปี 1978**
- **ได้รับการพัฒนาโดย Terry Welch ในปี 1984**
- **เป็นอัลกอริธึมสำหรับการบีบอัดข้อมูล**
- **เป็นการบีบอีดที่ให้ผล throughput ที่ดี**
- **นิยมใช้กันอย่างแพร่หลายใน Unix และ ถูกใช้ในการบีบอัด รูปภาพ (GIF)**

Ziv Lempel Welch (example)

Dictionary

1	A
2	B
3	W

Encode Text : WABBAWABBA

P :

C : **W**

P+C : **W**

P

Ziv Lempel Welch (example)

Dictionary

1	A
2	B
3	W
4	WA

Encode Text : WABBAWABBA

P : **W** **P**

C : **W,A**

P+C : **W,WA**

Ziv Lempel Welch (example)

Dictionary

1	**A**
2	**B**
3	**W**
4	**WA**
5	**AB**

Encode Text : WABBAWABBA

P : **W, A** **P**

C : **W,A, B**

P+C : **W,WA,AB**

Ziv Lempel Welch (example)

Dictionary

1	A
2	B
3	W
4	WA
5	AB
6	BB

Encode Text : WABBAWABBA

P : W, A, B → P

C : W, A, B, B

P+C : W, WA, AB, BB

Ziv Lempel Welch (example)

Dictionary

1	A
2	B
3	W
4	WA
5	AB
6	BB
7	BA

Encode Text : WABBAWABBA

P : W, A, B, B → P

C : W, A, B, B, A

P+C : W, WA, AB, BB, BA

Ziv Lempel Welch (example)

Dictionary

1	A
2	B
3	W
4	WA
5	AB
6	BB
7	BA
8	AW

Encode Text : WABBAWABBA

P : W, A, B, B, A P

C : W, A, B, B, A, W

P+C : W, WA, AB, BB, BA, AW

Ziv Lempel Welch (example)

Dictionary

1	A
2	B
3	W
4	WA
5	AB
6	BB
7	BA
8	AW

Encode Text : WABBAWABBA

P : W, A, B, B, A, W → **P**

C : W, A, B, B, A, W, A

P+C : W, WA, AB, BB, BA, AW, WA

Ziv Lempel Welch (example)

Dictionary

1	**A**
2	**B**
3	**W**
4	**WA**
5	**AB**
6	**BB**
7	**BA**
8	**AW**
9	**WAB**

Encode Text : WABBAWABBA

P : **W, A, B, B, A, W, WA** P

C : **W, A, B, B, A, W, A, B**

P+C : **W, WA, AB, BB, BA, AW, WA, WAB**

Ziv Lempel Welch (example)

Dictionary

1	A
2	B
3	W
4	WA
5	AB
6	BB
7	BA
8	AW
9	WAB

Encode Text : WABBAWABBA

P : W, A, B, B, A, W, WA, B

C : W, A, B, B, A, W, A, B, B

P+C : W, WA, AB, BB, BA, AW, WA, WAB, BB,

P

Ziv Lempel Welch (example)

Dictionary

1	A
2	B
3	W
4	WA
5	AB
6	BB
7	BA
8	AW
9	WAB
10	BBA

Encode Text : WABBAWABBA

P : W, A, B, B, A, W, WA, B, BB P

C : W, A, B, B, A, W, A, B, B, A

P+C : W, WA, AB, BB, BA, AW, WA, WAB, BB, BBA

Ziv Lempel Welch (example)

Dictionary

1	A
2	B
3	W
4	WA
5	AB
6	BB
7	BA
8	AW
9	WAB
10	BBA

Encode Text : WABBAWABBA

P : W, A, B, B, A, W, WA, B, BB, A

C : W, A, B, B, A, W, A, B, B, A

P+C : W, WA, AB, BB, BA, AW, WA, WAB, BB, BBA

Ziv Lempel Welch (example)

Dictionary

1	A
2	B
3	W
4	WA
5	AB
6	BB
7	BA
8	AW
9	WAB
10	BBA

Encode Text : WABBAWABBA

P : W, A, B, B, A, W, WA, B, BB, A

C : W, A, B, B, A, W, A, B, B, A

P+C : W, WA, AB, BB, BA, AW, WA, WAB, BB, BBA

Encode(output) : **3 1 2 2 1 4 6 1**

Ziv Lempel Welch (example)

Dictionary

1	A
2	B
3	W

Decode Text : **3 1 2 2 1 4 6 1**

Pw :

Cw : 3

StrP :

StrC : W

P :

C :

P+C:

Output : W

Ziv Lempel Welch (example)

Dictionary

1	A
2	B
3	W
4	WA

Decode Text : **3 1 2 2 1 4 6 1**

Pw : 3

Cw : 1

StrP : W

StrC : A

P : W

C : A

P+C: WA

Output : W A

Ziv Lempel Welch (example)

Dictionary

1	A
2	B
3	W
4	WA
5	AB

Decode Text : 3 1 2 2 1 4 6 1

Pw : 1

Cw : 2

StrP : A

StrC : B

P : A

C : B

P+C: AB

Output : WAB

Ziv Lempel Welch (example)

Dictionary

1	A
2	B
3	W
4	WA
5	AB
6	BB

Decode Text : **3 1 2 2 1 4 6 1**

Pw : 2

Cw : 2

StrP : B

StrC : B

P : B

C : B

P+C: BB

Output : WABB

Ziv Lempel Welch (example)

Dictionary

1	A
2	B
3	W
4	WA
5	AB
6	BB
7	BA

Decode Text : **3 1 2 2 1 4 6 1**

Pw : 2

Cw : 1

StrP : B

StrC : A

P : B

C : A

P+C: BA

Output : WABBA

Ziv Lempel Welch (example)

Dictionary

1	A
2	B
3	W
4	WA
5	AB
6	BB
7	BA
8	AW

Decode Text : **3 1 2 2 1 4 6 1**

Pw : 1

Cw : 4

StrP : A

StrC : WA

P : A

C : W

P+C: AW

Output : WABBAWA

Ziv Lempel Welch (example)

Dictionary

1	A
2	B
3	W
4	WA
5	AB
6	BB
7	BA
8	AW
9	WAB

Decode Text : **3 1 2 2 1 4 6 1**

Pw : 4

Cw : 6

StrP : WA

StrC : BB

P : WA

C : B

P+C: WAB

Output : WABBAWABB

Ziv Lempel Welch (example)

Dictionary

1	A
2	B
3	W
4	WA
5	AB
6	BB
7	BA
8	AW
9	WAB
10	BBA

Decode Text : **3 1 2 2 1 4 6 1**

Pw : 6

Cw : 1

StrP : BB

StrC : A

P : BB

C : A

P+C: BBA

Output : WABBAWABBA