Inverted Index Compression

전 희 원

Preface

- Compression in Information Retrieval
- Dictionary compression
  - Dictionary-as-string
  - Block storage
  - Conclusion
- Posting file compression
  - Fixed Length Compression
  - Variable Length Compression
    - Gamma encoding
    - Variable byte encoding
    - Word-aligned compression
  - Conclusion

Compression in Information Retrieval

- Lossless compression
  - Dictionary compression
  - Posting files compression
- Lossy compression
  - case-folding
  - Stemming
  - Stop word
  - No number
  - Spell correction
  - Etc…

Dictionary compression

Term	Doc #	Freq
ambitious	2	1
be	2	1
brutus	1	1
brutus	2	1
capitol	1	1
caesar	1	1
caesar	2	2
did	1	1
enact	1	1
hath	2	1
I	1	2
i'	1	1
it	2	1
julius	1	1
killed	1	2
let	2	1
me	1	1
noble	2	1
so	2	1
the	1	1
the	2	1
told	2	1
you	2	1
was	1	1
was	2	1
with	2	1

Term	N docs	Tot Freq
ambitious	1	1
be	1	1
brutus	2	2
capitol	1	1
caesar	2	3
did	1	1
enact	1	1
hath	1	1
I	1	2
i'	1	1
it	1	1
julius	1	1
killed	1	2
let	1	1
me	1	1
noble	1	1
so	1	1
the	2	2
told	1	1
you	1	1
was	2	2
with	1	1

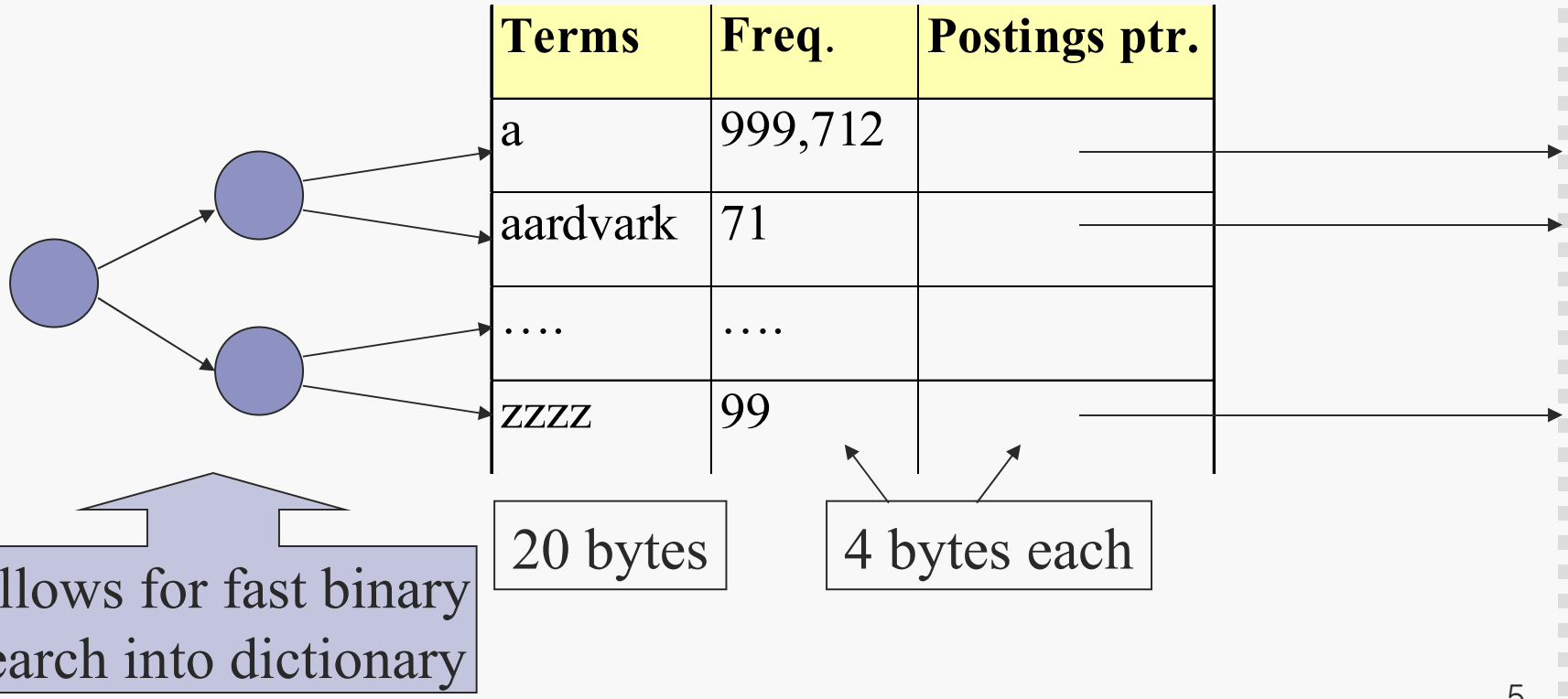
Doc #	Freq
2	1
2	1
1	1
2	1
1	1
1	1
2	2
1	1
1	1
2	1
1	2
1	1
2	1
1	1
1	2
2	1
1	1
2	1
2	1
1	1
2	1
2	1
2	1
1	1
2	1
2	1

Usually in memory

Gap-encoded, on disk

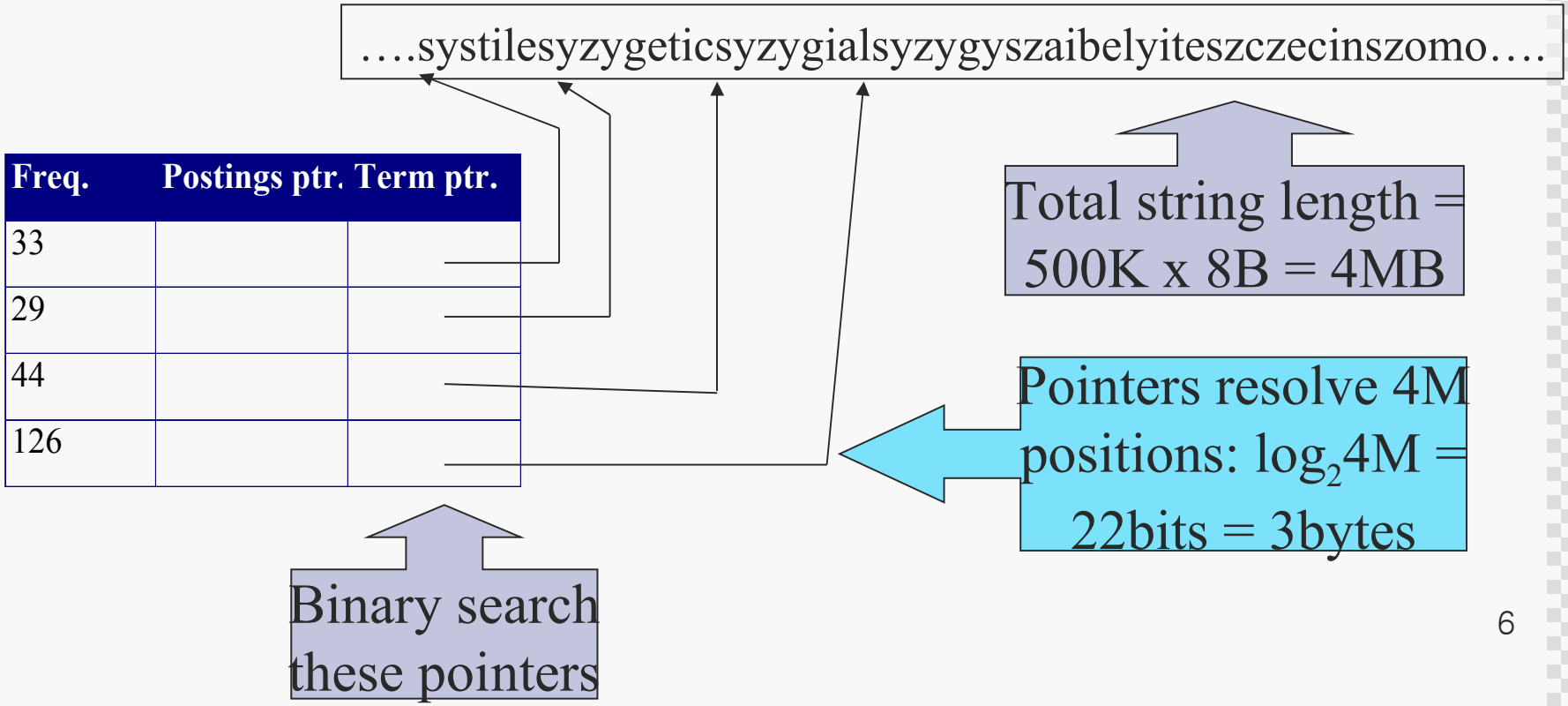
Dictionary compression (con't)

- Array of fixed-width entries
  - 500,000 terms; 28 bytes/term = 14MB



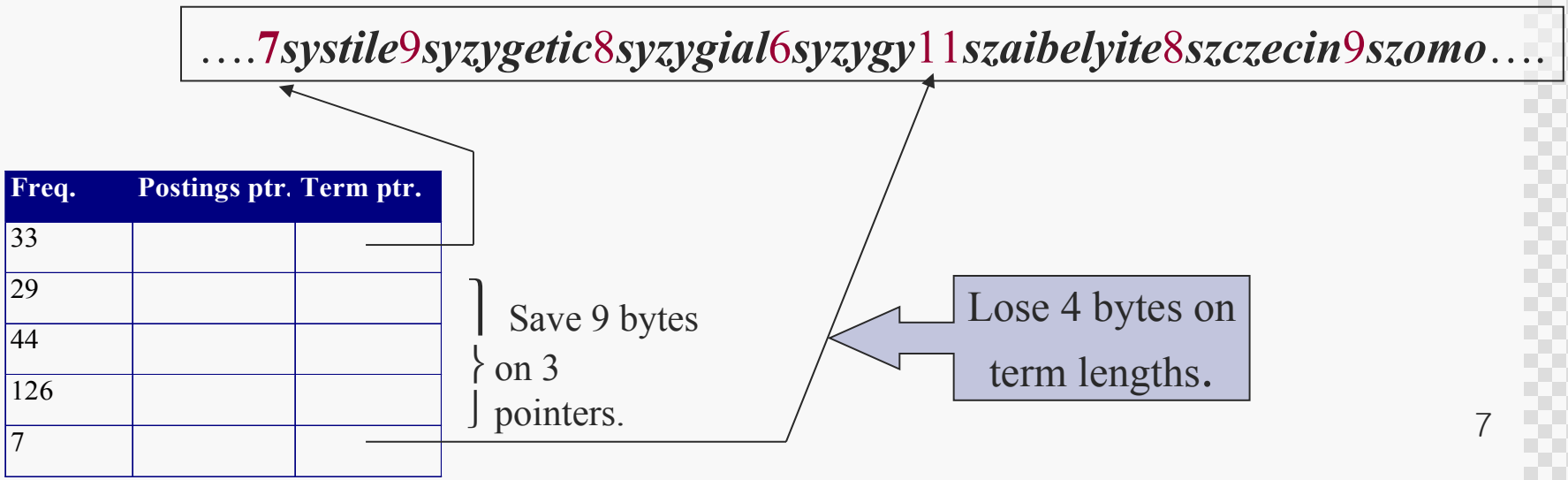
Dictionary compression (con't)

- Store dictionary as a (long) string of characters:
  - 500,000 terms; 19 bytes/term = 9.5MB



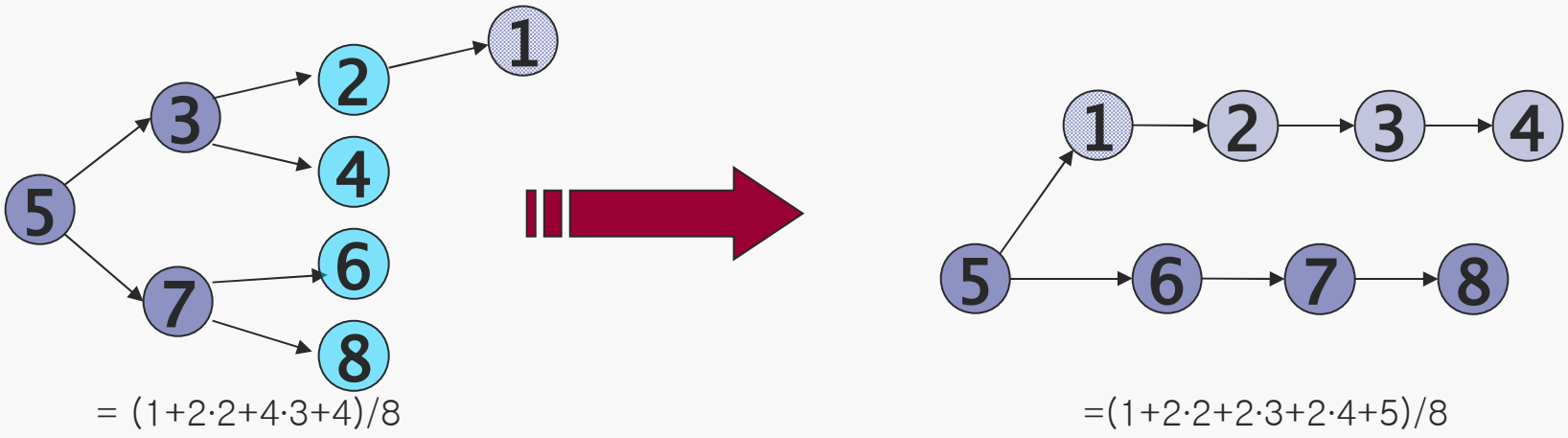
Dictionary compression (con't)

- Blocking
  - *k*th term에 대한 포인터를 저장
    - Example below: *k*=4.
  - Term길이 정보를 저장해야 한다.(1byte)
  - 500,000 terms; 8.8MB



Dictionary compression (con't)

- Impact on search
  - 8 document
    - binary search ave. = 2.6 compares
    - Block of 4 (binary tree), ave. = 3 compares

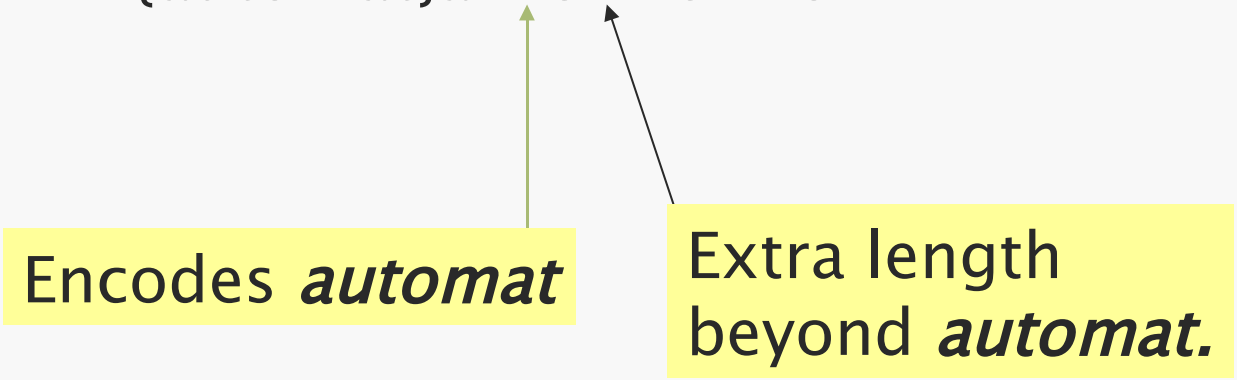


Dictionary compression (con't)

- Front-coding
  - Sorting된 term은 일반적으로 prefix를 공유한다.

8*automata*8*automate*9*automatic*10*automation*

→8{*automat*}*a*1◊*e*2◊*ic*3◊*ion*



Begins to resemble general string compression.

Dictionary compression (con't) Conclusion

- statistics
  - 800,000 doc * 200 token * avg. 6 chars = 960MB

representation	size in MB
dictionary, fixed-width	19.2
dictionary, term pointers into string	10.8
~, with blocking	10.3
~, with blocking & front coding	7.9

Dictionary compression for Reuters-RCV1.

- Blocking + front coding + ??
- Etc
  - K factor == Indexinterval in Lucene.

Posting file compression

- Posting에 있어서 두 가지 문제점
  - Ex. Reuters-RCV1
    - 800,000 doc, 200 token/doc, avg.6byte length/token
    - 매우 흔하지 않은 단어 (ex. calpurnia)
    - 매우 흔한 단어 (ex. the)
  - 따라서 최소한의 bit를 사용해서 저장해야 한다.

- Posting file entry (d-gap)
  - Doc ID는 오름차순으로 저장
    - *Brutus*: 1, 3, 7, 70, 250 …
    - 1, 2, 4, 63, 180 … (d-gap)

Fixed Length Compression

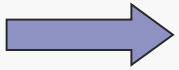
- Byte-aligned(BA) compression

Length	Value

  - Length : represent number of bytes
  - 만일 바이트 길이로 2비트를 쓴다면?

Value	Uncompressed Bit String
1	00000000 00000000 00000000 00000001
2	00000000 00000000 00000000 00000010
4	00000000 00000000 00000000 00000100
63	00000000 00000000 00000000 00111111
180	00000000 00000000 00000000 10110100

Use 160 bit



Value	Uncompressed Bit String
1	00 000001
2	00 000010
4	00 000100
63	00 111111
180	01 000000 10110100

Use 48 bit

Variable Length Compression : Gamma encoding

- Gamma codes for gap encoding

Length	Offset

  - *length* is $\lfloor \log_2 G \rfloor$ in unary and uses $\lfloor \log_2 G \rfloor + 1$ bits to specify the length of the binary encoding of the offset
  - *offset* = $G - 2^{\lfloor \log_2 G \rfloor}$ in binary encoded in $\lfloor \log_2 G \rfloor$ bits.

Value	Compressed Bit String
1	0
2	10 0
4	110 00
63	111110 11111
180	11111110 0110100

Use 35 bit

the unary encoding of *x* is a sequence of *x* 1's followed by a 0.

More Practical

- 이론상 좋다. (ex. gamma code) But…
  - OS는 8, 16, 32 bit 단위로 접근한다.
    - 세부 비트로 접근하는 건 실제로는 과부하다.
    - 쿼리 프로세싱시 퍼포먼스 하강.
- 실전에서는 간단한 byte/word-aligned 방법이 낫다.
  - See Scholer et al., Anh and Moffat references
- 현재 대부분의 하드웨어에서 바이트 단위가 가장 작은 접근 단위이다.
  - Suggest use of variable byte code

Variable byte encoding

- 바이트 단위 연산

Continuation Bit	Value

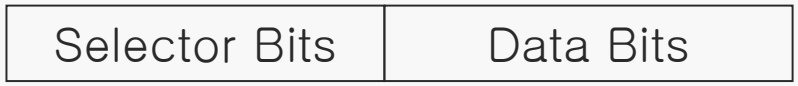
  - Continuation Bit : High 1 bit
    - X Value의 마지막 Byte : CB= 0
    - X Value를 인코딩 하기위해 중간에 존재하는 Byte : CB=1
  - Example
    - $0 < X < 2^7$, use 1 byte
      - 0bbbbbbb
    - $2^7 \leq X < 2^{(7*2)}$, use 2 bytes
      - 1bbbbbbb 0bbbbbbb
    - $2^{(7*2)} \leq X < 2^{(7*3)}$, use 3 bytes, and so on …
      - 1bbbbbbb 1bbbbbbb 0bbbbbbb
  - Used by many commertial/research system
    - Fast Decoding
    - Best 25%, Worst 125% (32-bit Integer)
    - Inefficient for very small gap

Value	Compressed Bit String
1	00000001
2	00000010
4	00000100
63	00111111
180	10110100 00000001

Use 48 bit

Word-aligned compression

- Variable byte encoding은 gap이 상당히 작을 경우 성능이 좋지 않다.
- Word-aligned compression
  - 하나의 워드에 들어있는 Integer값은 같은 Bits 정보로 구성이 된다.



  - 적당한 파티셔닝 계획은 posting list에서 가장 큰 Doc ID Gap을 기준으로 한다.
  - Anh과 Moffat은 파티셔닝 정책에 따른 세가지 Word-aligned compression 방법론을 제시했다.[1]
    - Simple-9
    - Relative-10
    - Carryover-12
      - 개선책 : Slide [2]

Word-aligned compression Simple-9

- Use 28 bits for data and 4 bits for selector field
  - The selection table has nine rows. As there are nine different ways to split 28 bits equally.

4 Selector Bits	28 Data Bits

9 partitioning

selector	Codes	Length (bits)	Number of Unused bits
a	28	1	0
b	14	2	0
c	9	3	1
d	7	4	0
e	5	5	3
f	4	7	0
g	3	9	1
h	2	14	0
i	1	28	0

Word-aligned compression Relative-10

- 전반적으로 Simple-9 알고리즘에 비해 성능이 좋다

2 Selector Bits	30 Data Bits

10 partitionings

Selector	codes	Length of each Code (bits)	# of unused bits
a	30	1	0
b	15	2	0
c	10	3	0
d	7	4	2
e	6	5	0
f	5	6	0
g	4	7	2
h	3	10	0
i	2	15	0
J	1	30	0

Transfer matrix

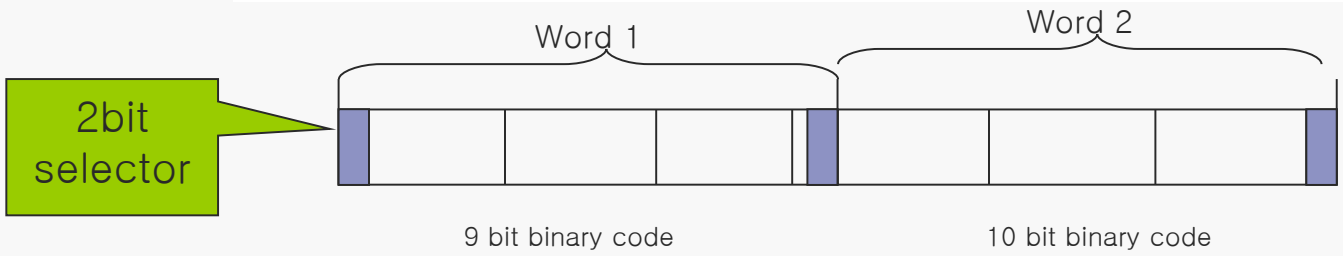
	a	b	c	c	e	f	g	h	i	j
a	0	1	2							3
b	0	1	2							3
c		0	1	2						3
c			0	1	2					3
e				0	1	2				3
f					0	1	2			3
g						0	1	2		3
h							0	1	2	3
i							0	1	2	3
j							0	1	2	3

Word-aligned compression Carryover-12

- Relative-10 알고리즘의 단편화 문제를 어느 정도 해결하는 알고리즘
  - Case of 7bit and 4bit in previous table
- 앞의 세가지 알고리즘에 비해 가장 좋은 압축률을 보여주나, 가장 복잡도가 크고 decoding에 가장 많은 시간이 걸린다.

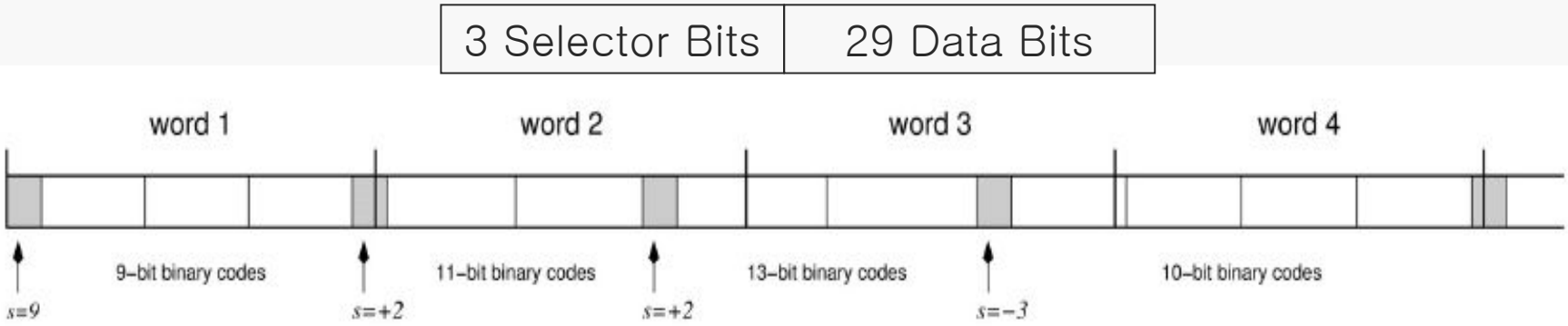
30, 32 data bit 각각 12개의 옵션존재

Selector	Previously selector available			No previous selector		
	Length of each code (bits)	Number of codes	Bits for next selector?	Length of each code (bits)	Number of codes	Bits for next selector?
a	1	32		1	30	
b	2	16		2	15	
c	3	10	Yes	3	10	
d	4	8		4	7	Yes
e	5	6	Yes	5	6	
f	6	5	Yes	6	5	
g	7	4	Yes	7	4	Yes
h	8	4		9	3	Yes
i	10	3	Yes	10	3	
j	15	2	Yes	14	2	Yes
k	16	2		15	2	
l	28	1	Yes	28	1	Yes



Word-aligned compression slide

- 만일 12bit의 codeword를 사용한다면 각 워드당 6비트의 용량 손실이 있을 것이다. (case of Relative-10)



- Slide mechanism
  - Key Difference : 이전 word의 끝부분에 다음 워드의 selector bit를 바로 추가한다.
- 알고리즘이 carry에 비해 복잡하지만 우월한 압축률을 보여주며 decoding 속도가 약간 증가하였다.
  - Relative-10 알고리즘과의 비교는?

Conclusion

- Aspect
  - IO
    - view
      - 저장공간의 최소화
      - Byte나 word단위 데이터 접근 문제
      - 쿼리 타임의 Uncompress 퍼포먼스
    - Con.
      - Word-align, variable byte encoding
  - Doc ID
    - view
      - Contents의 특성에 따른 Doc ID 정렬 가능성
    - Con.
      - Word-align(Relative-10)
- Winner
  - **Word-align compression(Relative-10)**

Q & A

References

- [1] Inverted Index Compression Using Word-Aligned Binary Codes / Anh, Vo Ngoc ; Moffat, Alistair (Information retrieval, v.8 no.1, 2005, pp.151-166)
- [2] Improved word-aligned binary compression for text indexing / Anh, V.N. ; Moffat, A. (IEEE transactions on knowledge and data engineering, v.18 no.6, 2006, pp.857-861)
- [3] Information Retrieval : Algorithm and Heuristics (Springer. 2005)
- [4] Introduction to Information Retrieval, (Cambridge University Press. 2007.)