Posting File Compression

- Survey compression method for inverted index file

Contents

Why do we use compression method?

- Access Speed
  - CPU > CACHE > RAM > DISK

- Documents quickly increased with time (about 19billion documents, 2007)
  → Disk I/O (blocking I/O) ↑
  → Bottleneck
  - If disk I/O reduce, we'll get efficiency for improvement in speed.

- So, we use compression method

- also, we get some advantage with compression method
  → improved cache-hit-ratio
  → use efficient memory

Gamma Code Compression

- 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
  - the unary encoding of x is a sequence of x 1's followed by a 0. (5 -> unary : 11111)
- offset = $G - 2^{\lfloor \log_2 G \rfloor}$ in binary encoded in $\lfloor \log_2 G \rfloor$ bits.
- expression : (length, offset)
- Strength
  - this is good theoretically
- Weakness
  - OS approach on 8, 16, 32 bit unit
  - Approach of bit unit happens to overload → when query processing, it happens to down performance.

- ex) compress "14"
  - unary value is numeral array of binary that not over itself.
  - log X = 3 < 14 ➔ ("3" → unary : 111)
  - 111 + 0 + (14 – 2^3) → 111 0 110

d-gap	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

160bit

d-gap	Uncompressed Bit String
1	0
2	10 0
4	110 00
63	111110 11111
180	11111110 0110100

35bit

Variable Byte(VB) Code Compression

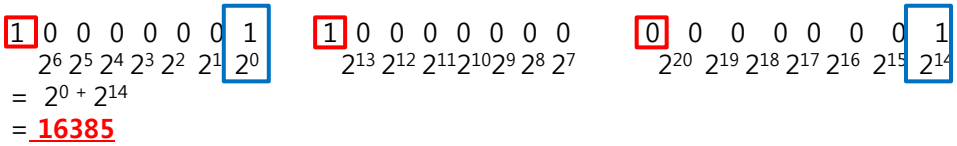
- expression : (Continuation Bit, value)
- Continuation Bit : High 1 bit
  - The last Byte of X value is CB = 0
  - The middle Byte for X value encoding is CB = 1
- ex)

$0 < X < 2^7$	use1 byte	0bbbbbbb
$2^7 < X < 2^{14}$	use 2 byte	1bbbbbbb 0bbbbbbb
$2^{14} < X < 2^{21}$	use 3 byte, and so on	1bbbbbbb 1bbbbbbb 0bbbbbbb

- Strength
  - Usually, H/W approach on byte unit → general
  - Can treat dynamic data (easy insert / delete)
  - simply coding
  - High speed for encoding/decoding
- Weakness
  - In the case of d-gap is very small, it happens to down performance
  - compression rate is low because of byte operation
- Compression Rate
  - best : 25%
  - Worst : 125%

Example

Value	First Byte	Second Byte	Third Byte
0	00000000		
1	00000001		
2	00000010		
4	00000100		
63	00111111		
127	01111111		
128	10000000	00000001	
129	10000001	00000001	
130	10000010	00000001	
180	10110100	00000001	
16,383	11111111	01111111	
16,384	10000000	10000000	00000001
16,385	10000001	10000000	00000001



Pseudo Code

```
VBENCODENUMBER(n)
1  bytes ← ⟨⟩
2  while true
3  do PREPEND(bytes, n mod 128)
4     if n < 128
5        then BREAK
6     n ← n div 128
7  bytes[LENGTH(bytes)] += 128
8  return bytes

VBENCODE(numbers)
1  bytestream ← ⟨⟩
2  for each n ∈ numbers
3  do bytes ← VBENCODENUMBER(n)
4     bytestream ← EXTEND(bytestream, bytes)
5  return bytestream

VBDECODE(bytestream)
1  numbers ← ⟨⟩
2  n ← 0
3  for i ← 1 to LENGTH(bytestream)
4  do if bytestream[i] < 128
5        then n ← 128 × n + bytestream[i]
6        else n ← 128 × n + (bytestream[i] − 128)
7             APPEND(numbers, n)
8             n ← 0
9  return numbers
```

► **Figure 5.4** Variable byte encoding and decoding. The functions div and mod compute integer division and remainder after integer division, respectively. Prepend adds an element to the beginning of a list, e.g., PREPEND($\langle 1,2\rangle, 3$) = $\langle 3,1,2\rangle$. Extend extends a list, e.g., EXTEND($\langle 1,2\rangle, \langle 3,4\rangle$) = $\langle 1,2,3,4\rangle$.

```
...
def writeVInt(i):
    while ( i & ~0x7F) ! = 0:
        writeByte((i & 0x7F) | 0x80)
        i = i >> 7
    writeByte(i)
...
```

Word-Aligned(WA) Compression

- The integer value in a word(32-bits) make up same bits (= has same bit number)

- At the partitioning plan, biggest d-gap of posting list is base.

- Method
  - **Simple-9** : 4 selector bits + 28 data bits
  - **Relatiev-10** : 2 selector bits + 30 data bits
  - **Carryover-12**

- Strength
  - compression rate is high.
  - fast speed

- Weakness
  - it's difficult to treat dynamic data

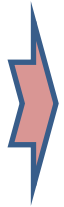
Simple-9

- expression : (4 selector bits, 28 data bits)
- The selection table has 9 rows. As there are 9 different ways to split 28 bits equally.

9 partitioning

selector	Codes (encoding할 수 있는 데이터 수)	Length(bits) (각 코드의 길이)	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

extend



selector	chunk당 d-gap의 개수	d-gap의 크기(bit 수)	미사용 영역의 크기
0	28	1	0
1	1	28	0
2	2	14	0
3	3	9	1
4	4	7	0
5	5	5	3
6	6	4	4
7	7	4	0
8	8	3	4
9	9	3	1
10	10	2	8
11	11	2	6
12	12	2	4
13	13	2	2
14	14	2	0
15	15	1	13

Simple-9 : adapt

doc id list	1	3	9	11	12	14	36	57	102	111	150	154	178	188	10000	10012	11000	11356	12654	13001	13060	13101	13122	13125	13200
d-gap	1	2	6	2	1	2	22	21	45	9	39	4	24	10	9812	12	988	356	1298	347	59	41	21	3	75
Max d-gap	1	2	6	6	6	6	22																		
# in chunk	0	1	2	3	4	5	6																		
bit power	3E+08	16384	512	128	32	16	8																		

	36	57	102	111	150	154
d-gap	22	21	45	9	39	4
Max	22	22	45	45	45	45
#	0	1	2	3	4	5
bit	bit po	3E+08	16384	512	128	32

	154	178	188	10000
d-gap	4	24	10	9812
Max	4	24	24	9812
#	0	1	2	3
bit	3E+08	16384	512	128

2^28	2^14	2^9	2^7	2^5	2^4	2^3	2^2	2^1
3E+08	16384	512	128	32	16	8	4	2

	10000	10012	11000
d-gap	9812	12	988
Max	9812	9812	9812
#	0	1	2
bit	3E+08	16384	512

	11000	11356	12654
d-gap	988	356	1298
max	988	988	1298
#	0	1	2
bit	3E+08	16384	512

selector	28 bits data area						
6	1	2	6	2	1	2	0
5	22	21	45	9	39		1
3	4	24	10				2
2	9812	12					3
2	988	356					4
2	1298	347					5
4	59	41	21	3			6
1	75						7

	12654	13001	13060
d-gap	1298	347	59
max	1298	1298	1298
#	0	1	2
bit	3E+08	16384	512

	13060	13101	13122	13125	13200
d-gap	59	41	21	3	75
max	59	59	59	59	75
#	0	1	2	3	4
bit	3E+08	16384	512	128	32

Relative-10

- expression : (2 selector bits, 30 data bits)
- Usually, this method is good than simple-9 algorithm

10 partitioning

Selector	Codes	Length(bits)	Number 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
-Possible next selector values

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

Relative-10 : adapt

doc id list	1	3	9	11	12	14	36	57	102	111	150	154	178	188	10000	10012	11000	11356	12654	13001	13060	13101	13122	13125	13200
d-gap	1	2	6	2	1	2	22	21	45	9	39	4	24	10	9812	12	988	356	1298	347	59	41	21	3	75
Max d-gap	1	2	6	6	6	6	22																		
# in chunk	0	1	2	3	4	5	6																		
bit power	1E+09	32768	1024	128	64	32	16																		

	36	57	102	111	150	154
d-gap	22	21	45	9	39	4
Max	22	22	45	45	45	45
#	0	1	2	3	4	5
bit	1E+09	32768	1024	128	64	32

	154	178	188	10000
d-gap	4	24	10	9812
max	4	24	24	9812
#	0	1	2	3
bit	1E+09	32768	1024	128

분기 -->

154
4
0
XX

+

178	188	10000
24	10	9812
24	24	9812
0	1	2
XX	XX	XX

2^30	2^15	2^10	2^7	2^6	2^5	2^4	2^3	2^2	2^1
1E+09	32768	1024	128	64	32	16	8	4	2

	10000	10012	11000
d-gap	9812	12	988
max	9812	9812	9812
#	0	1	2
bit	1E+09	32768	1024

	11000	11356	12654
d-gap	988	356	1298
max	988	988	1298
#	0	1	2
bit	1E+09	32768	1024

	12654	13001	13060
d-gap	1298	347	59
max	1298	1298	1298
#	0	1	2
bit	1E+09	32768	1024

	13060	13101	13122	13125	13200
d-gap	59	41	21	3	75
max	59	59	59	59	75
#	0	1	2	3	4
bit	1E+09	32768	1024	128	64

first selector

e	selector	30 bits data area							e	
	01	1	2	6	2	1	2		e	0
	10	22	21	45	9	39			f	1
	11	4							j	2
	10	24	10						i	3
	10	9812	12						i	4
	10	988	356						i	5
	10	1298	347						i	6
	00	59	41	21	3				g	7
	11	75							j	8

Carryover-12

- this method solved a problem of Relative-10
  - Case of 7bit and 4bit in previous table(10 partitioning)

- this method is good than Simple-9, Relative-10, but waste decoding time because of high complexity

	Previously selector available			No previous selector		
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

Slide

- If the codeword of 12 bits use, we waste 6 bits per a word
  - Case of 10 partitioning (Relative-10)

- The compression rate of this method is excellent than carryover-12,
- But this complexity is high than carryover-12
- Mechanism
  - Key difference
    - Selector bit of next word is added to the end of previous word.
- expression : (3 selector bits, 29 data bits)