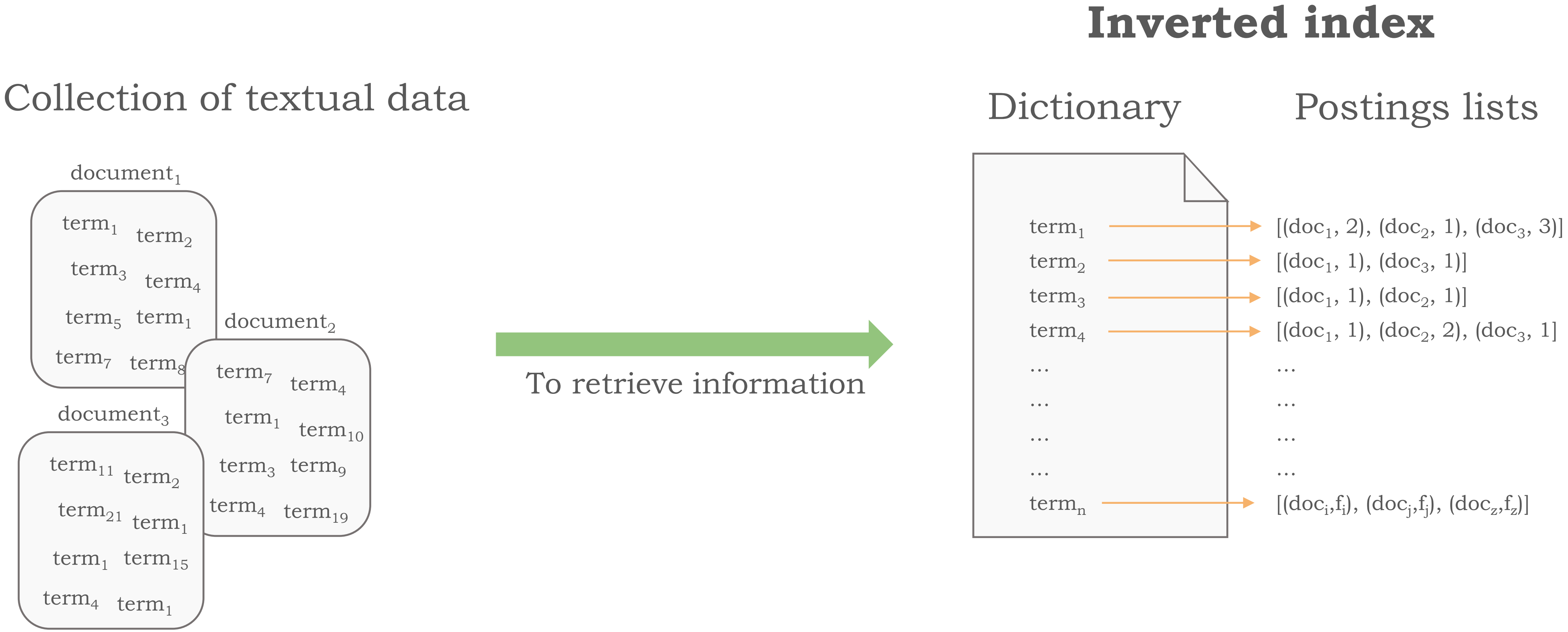
Overview and comparison of three techniques for inverted index compression

Golomb coding, Binary Interpolative Coding, and Simple-9

Silvia Imeneo

Information Retrieval A.Y. 2023-2024
Data Science and Scientific Computing

RECAP ON THE INVERTED INDEX



PROBLEM!

Inverted index

Dictionary — Postings lists

$term_1$ → [(doc_1, 2), (doc_2, 1), …, …, …, …, …, …., …., ($doc_{\mathbf{3,822,901}}$, 20)]

$term_2$ → [(doc_1, 1), …, …, …, …, ($doc_{\mathbf{41,002,859}}$, 11)]

$term_3$ → [(doc_1, 1), …, …, …, …, …, …, …, …, ($doc_{\mathbf{29,835,800}}$, 1)]

$term_4$ → [(doc_1, 5), (doc_2, 3), …, …, …, ($doc_{\mathbf{9,672,865}}$, 5)]

… …

… …

… …

… …

$term_{\mathbf{VERY\_LARGE\_N}}$ → [(doc_i,f_i), (doc_j,f_j), …, …, …, …, …, ($doc_{\mathbf{milion\_something}}$,f)]

In moder large-scale search engines, an inverted index indexes **millions** of documents

billions of integers to store

BENEFITS OF COMPRESSION

- Less disk space needed to store the inverted index
- More info fits into main memory = higher use of caching
- Faster query processing

TYPES OF COMPRESSION

Lossless: all information is preserved

Lossy: some information is discarded

Compression of **dictionary**

Compression of **postings lists**

We'll see 3 **lossless** techniques to compress **postings lists**

GOLOMB CODING

GOLOMB CODING

Encodes a postings list by encoding **each integer individually**

Example:

We have a postings list S = [4, 10, 11, 12, 15, 20, 21, 28, 29]

We compute the *d*-gaps D = [4, 6, 1, 1, 3, 5, 1, 7, 1]

We encode each integer individually

- encode 4
- encode 6
- encode 1
- etc…

GOLOMB CODING

How to

1. Start with the integer to compress x

GOLOMB CODING
How to

1. Start with the integer to compress

x

2. Represent it as two parts

quotient	remainder
$q = \left\lfloor \frac{(x-1)}{k} \right\rfloor$	$r = x - (q * k) - 1$

GOLOMB CODING

How to

1. Start with the integer to compress

x

2. Represent it as two parts

quotient	remainder
$q = \left\lfloor \frac{(x-1)}{k} \right\rfloor$	$r = x - (q * k) - 1$

3. Compute two auxiliary quantities

$b = \lfloor \log_2(k) \rfloor$	$p = 2^{b+1} - k$

GOLOMB CODING

k is the base of the Golomb code

It depends on the distribution of the integers in the postings list → they are assumed to follow a Bernoulli model

An estimated value is $k \approx 0.69 \times$ mean(*array_of_integers*)

x

quotient	remainder
$= \left\lfloor \frac{(x-1)}{k} \right\rfloor$	$r = x - (q * k) - 1$

3. Compute two auxiliary quantities

$b = \lfloor \log_2(k) \rfloor$	$p = 2^{b+1} - k$

GOLOMB CODING

How to (cont.)

quotient

$$q = \left\lfloor \frac{(x-1)}{k} \right\rfloor$$

remainder

$$\boldsymbol{r} = x - (q * k) - 1$$

$$b = \lfloor \log_2(k) \rfloor$$

$$\mathbf{p} = 2^{b+1} - k$$

4. Based on those four quantities:

GOLOMB CODING

How to (cont.)

quotient	remainder		
$q = \left\lfloor \frac{(x-1)}{k} \right\rfloor$	$\boldsymbol{r} = x - (q * k) - 1$	$b = \lfloor \log_2(k) \rfloor$	$\mathbf{p} = 2^{b+1} - k$

4. Based on those four quantities:

4a. If **r** < **p**, the Golomb code is: q in **unary** code concat. r in **binary** code

GOLOMB CODING
How to (cont.)

quotient	remainder		
$q = \left\lfloor \frac{(x-1)}{k} \right\rfloor$	$\boldsymbol{r} = x - (q * k) - 1$	$b = \lfloor \log_2(k) \rfloor$	$\mathbf{p} = 2^{b+1} - k$

4. Based on those four quantities:

4a. If **r** < **p**, the Golomb code is: q in **unary** code concat. r in **binary** code

4b. If **r** ≥ **p**, the Golomb code is: q in **unary** code concat. r + p in **binary** code

GOLOMB CODING
Example

The integer to compress is **x = 9**

We choose as base **k = 3**

GOLOMB CODING

Example

The integer to compress is **x = 9**

We choose as base **k = 3**

1. We compute the quotient $q = \left\lfloor \frac{(9-1)}{3} \right\rfloor = 2$

quotient $q = \left\lfloor \frac{(x-1)}{k} \right\rfloor$	

GOLOMB CODING
Example

The integer to compress is **x = 9**

We choose as base **k = 3**

1. We compute the quotient $q = \left\lfloor \frac{(9-1)}{3} \right\rfloor = 2$

2. We compute the remainder $r = 9 - (2 * 3) - 1 = 2$

quotient $q = 2$	remainder $r = x - (q * k) - 1$

GOLOMB CODING
Example

The integer to compress is **x = 9**

We choose as base **k = 3**

1. We compute the quotient $q = \left\lfloor \frac{(9-1)}{3} \right\rfloor = 2$

2. We compute the remainder $r = 9 - (2 * 3) - 1 = 2$

3. We compute $b = \lfloor \log_2(3) \rfloor = 1$

quotient $q = 2$	remainder $r = 2$
$b = \lfloor \log_2(k) \rfloor$	

GOLOMB CODING
Example

The integer to compress is **x = 9**

We choose as base **k = 3**

1. We compute the quotient $q = \left\lfloor \frac{(9-1)}{3} \right\rfloor = 2$

2. We compute the remainder $r = 9 - (2 * 3) - 1 = 2$

3. We compute $b = \lfloor \log_2(3) \rfloor = 1$

4. We compute $p = 2^{1+1} - 3 = 1$

quotient $q = 2$	remainder $r = 2$
$b = 1$	$p = 2^{b+1} - k$

GOLOMB CODING

Example

Since **r > p**

$q = 2$	$r = 2$
$b = 1$	$p = 1$

GOLOMB CODING

Example

Since **r > p**

the Golomb code is:

q in **unary** code	concat.	r + p in **binary** code

$q = 2$ | $r = 2$

$b = 1$ | $p = 1$

GOLOMB CODING
Example

$q = 2$	$r = 2$
$b = 1$	$p = 1$

Since **r > p**

the Golomb code is:

q in **unary** code — concat. — r + p in **binary** code

2 in unary code — concat. — 2 + 1 = 3 in binary code

GOLOMB CODING

Example

$q = 2$	$r = 2$
$b = 1$	$p = 1$

Since **r > p**

the Golomb code is:

q in **unary** code	concat.	r + p in **binary** code
2 in unary code	concat.	2 + 1 = 3 in binary code
110	concat.	11

GOLOMB CODING
Example

Since **r > p**

the Golomb code is:

q in **unary** code	concat.	r + p in **binary** code
2 in unary code	concat.	2 + 1 = 3 in binary code
110	concat.	11

$q = 2$	$r = 2$
$b = 1$	$p = 1$

$G_3(9) = 110\ 11$

BINARY INTERPOLATIVE CODING

BINARY INTERPOLATIVE CODING

Encodes elements in a postings list **by using the already-encoded ones**

- ! The encoding of an integer *x* is not fixed
- ! The same integer may be encoded differently over different postings lists

Recursive algorithm

BINARY INTERPOLATIVE CODING

We have a sequence $S = [s_1, s_2, s_3, \ldots, s_m, \ldots, s_n]$

For it we always know the parameters:

- *n* **number** of elements in S
- *low* **lower-bound** to the **lowest value** in the sequence
- *hi* **upper-bound** to the **highest value** in the sequence
- *m* the index of the **middle** element of the sequence

BINARY INTERPOLATIVE CODING

$S = [s_1, s_2, \ldots, s_m, \ldots, \ldots, s_n]$

At the **first iteration**:

- $n \quad = |S|$
- $low \quad = 0$
- $hi \quad = s_n$
- $m \quad = \frac{(n+1)}{2}$

BINARY INTERPOLATIVE CODING

$S = [s_1, s_2, \ldots, s_m, \ldots, \ldots, s_n]$

At the first iteration:

- $n \quad = |S|$
- $low \quad = 0$
- $hi \quad = s_n$
- $m \quad = \frac{(n+1)}{2}$

These parameters are recomputed at each iteration

BINARY INTERPOLATIVE CODING
How to

$S = [s_1, s_2, \ldots, \mathbf{s_m}, \ldots, \ldots, s_n]$

1. The first value that we encode is s_m, the one at position m

! 2. Instead of encoding s_m itself we encode $\boldsymbol{s_m - low - m + 1}$

3. We then **split the sequence** at s_m in left and right sub-sequences

4. We **iterate** the algorithm on both subsequences:
   - Recompute the parameters
   - Find the value in the middle
   - Encode is has $s_m - low - m + 1$
   - Split the sequence in two halves

BINARY INTERPOLATIVE CODING

Example

[3, 4, 7, 11, 13, 15, 21, 25, 36, 38, 54]

Our postings list

BINARY INTERPOLATIVE CODING

Example

[3, 4, 7, 11, 13, 15, 21, 25, 36, 38, 54]

(m = 6; n = 11; low = 0; hi = 54)

We compute the initial parameters

- $n = |S| = 11$
- $low = 0$
- $hi = s_n = 54$
- $m = \frac{(n+1)}{2} = 6$

BINARY INTERPOLATIVE CODING

Example

[3, 4, 7, 11, 13, **15**, 21, 25, 36, 38, 54]
(m = 6; n = 11; low = 0; hi = 54)

We retrieve the value in the middle, $\mathbf{s_m}$

BINARY INTERPOLATIVE CODING

Example

[3, 4, 7, 11, 13, **15**, 21, 25, 36, 38, 54]
(m = 6; n = 11; low = 0; hi = 54)

encode 10

We encode $\mathbf{s_m - low - m + 1} =$

$= 15 - 0 - 6 + 1 = 10$

BINARY INTERPOLATIVE CODING

Example

[3, 4, 7, 11, 13, **15**, 21, 25, 36, 38, 54]
(m = 6; n = 11; low = 0; hi = 54)

encode 10

[3, 4, 7, 11, 13]

[21, 25, 36, 38, 54]

We split the initial sequence into a left and a right subsequence

BINARY INTERPOLATIVE CODING

Example

[3, 4, 7, 11, 13, **15**, 21, 25, 36, 38, 54]
(m = 6; n = 11; low = 0; hi = 54)

encode 10

[3, 4, 7, 11, 13]
(m = 3; n = 5; low = 0; hi = 14)

[21, 25, 36, 38, 54]
(m = 3; n = 5; low = 16; hi = 54)

We recompute the parameters

n_{left} low_{left} hi_{left} m_{left}

n_{right} low_{right} hi_{right} m_{right}

BINARY INTERPOLATIVE CODING

Example

[3, 4, 7, 11, 13, **15**, 21, 25, 36, 38, 54]
(m = 6; n = 11; low = 0; hi = 54)

encode 10

[3, 4, **7**, 11, 13]
(m = 3; n = 5; low = 0; hi = 14)

[21, 25, **36**, 38, 54]
(m = 3; n = 5; low = 16; hi = 54)

We retrieve the value in the middle

BINARY INTERPOLATIVE CODING
Example

[3, 4, 7, 11, 13, **15**, 21, 25, 36, 38, 54]
(m = 6; n = 11; low = 0; hi = 54)

encode 10

[3, 4, **7**, 11, 13]
(m = 3; n = 5; low = 0; hi = 14)

encode 5

[21, 25, **36**, 38, 54]
(m = 3; n = 5; low = 16; hi = 54)

encode 18

We encode s_m – low – m +1

BINARY INTERPOLATIVE CODING

Example

[3, 4, 7, 11, 13, **15**, 21, 25, 36, 38, 54]
(m = 6; n = 11; low = 0; hi = 54)

encode 10

[3, 4, **7**, 11, 13]
(m = 3; n = 5; low = 0; hi = 14)

encode 5

[3, 4]

[11, 13]

[21, 25, **36**, 38, 54]
(m = 3; n = 5; low = 16; hi = 54)

encode 18

[21, 25]

[38, 54]

Split the sub-sequences

BINARY INTERPOLATIVE CODING

Example

[3, 4, 7, 11, 13, **15**, 21, 25, 36, 38, 54]
(m = 6; n = 11; low = 0; hi = 54)

encode 10

[3, 4, **7**, 11, 13]
(m = 3; n = 5; low = 0; hi = 14)

encode 5

[21, 25, **36**, 38, 54]
(m = 3; n = 5; low = 16; hi = 54)

encode 18

[3, 4]
(m = 1; n = 2; low = 0; hi = 6)

[11, 13]
(m = 1; n = 2; low = 8; hi = 14)

[21, 25]
(m = 1; n = 2; low = 16; hi = 35)

[38, 54]
(m = 1; n = 2; low = 37; hi = 54)

Recompute the parameters

BINARY INTERPOLATIVE CODING

Example

[3, 4, 7, 11, 13, **15**, 21, 25, 36, 38, 54]
(m = 6; n = 11; low = 0; hi = 54)

encode 10

[3, 4, **7**, 11, 13]
(m = 3; n = 5; low = 0; hi = 14)

encode 5

[21, 25, **36**, 38, 54]
(m = 3; n = 5; low = 16; hi = 54)

encode 18

[**3**, 4]
(m = 1; n = 2; low = 0; hi = 6)

encode 3

[**11**, 13]
(m = 1; n = 2; low = 8; hi = 14)

encode 3

[**21**, 25]
(m = 1; n = 2; low = 16; hi = 35)

encode 5

[**38**, 54]
(m = 1; n = 2; low = 37; hi = 54)

encode 1

find s_m and encode s_m – low – m +1

BINARY INTERPOLATIVE CODING

Example

- [3, 4, 7, 11, 13, **15**, 21, 25, 36, 38, 54]
  (m = 6; n = 11; low = 0; hi = 54)
  encode 10
  - [3, 4, **7**, 11, 13]
    (m = 3; n = 5; low = 0; hi = 14)
    encode 5
    - [**3**, 4]
      (m = 1; n = 2; low = 0; hi = 6)
      encode 3
      - [**4**]
        (m = 1; n = 1; low = 4; hi = 6)
        encode 0
    - [**11**, 13]
      (m = 1; n = 2; low = 8; hi = 14)
      encode 3
      - [**13**]
        (m = 1; n = 1; low = 12; hi = 14)
        encode 1
  - [21, 25, **36**, 38, 54]
    (m = 3; n = 5; low = 16; hi = 54)
    encode 18
    - [**21**, 25]
      (m = 1; n = 2; low = 16; hi = 35)
      encode 5
      - [**25**]
        (m = 1; n = 1; low = 22; hi = 35)
        encode 3
    - [**38**, 54]
      (m = 1; n = 2; low = 37; hi = 54)
      encode 1
      - [**54**]
        (m = 1; n = 1; low = 39; hi = 54)
        encode 15

BINARY INTERPOLATIVE CODING

Space needed for encoding

The encoding happens using $\lceil \log_2(hi - low - n + 1) \rceil$ bits, the logarithm of the interval in which the middle value lies

BINARY INTERPOLATIVE CODING

Example

[3, 4, 7, 11, 13, **15**, 21, 25, 36, 38, 54]
(m = 6; n = 11; low = 0; hi = 54)

encode 10

[3, 4, **7**, 11, 13]
(m = 3

[21, 25, **36**, 38, 54]

This encoding takes $\lceil \log_2(\mathit{hi} - \mathit{low} - n + 1) \rceil =$
$= \lceil \log_2(54 - 0 - 11 + 1) \rceil =$ **6 bits**

[**3**, 4]
(m = 1; n = 2; low = 0; hi

encode 3

encode 3

encode 5

54)

encode 1

[**4**]
(m = 1; n = 1; low = 4; hi = 6)

encode 0

[**13**]
(m = 1; n = 1; low = 12; hi = 13)

encode 1

[**25**]
(m = 1; n = 1; low = 22; hi = 35)

encode 3

[**54**]
(m = 1; n = 1; low = 39; hi = 54)

encode 15

BINARY INTERPOLATIVE CODING

Example

[3, 4, 7, 11, 13, **15**, 21, 25, 36, 38, 54]
(m = 6; n = 11; low = 0; hi = 54)

encode 10 using **6 bits**

[3, 4, **7**, 11, 13]
(m = 3; n = 5; low = 0; hi = 14)

encode 5 using **4 bits**

[21, 25, **36**, 38, 54]
(m = 3; n = 5; low = 16; hi = 54)

encode 18 using **6 bits**

[**3**, 4]
(m = 1; n = 2; low = 0; hi = 6)

encode 3 using **3 bits**

[**11**, 13]
(m = 1; n = 2; low = 8; hi = 14)

encode 3 using **3 bits**

[**21**, 25]
(m = 1; n = 2; low = 16; hi = 35)

encode 5 using **5 bits**

[**38**, 54]
(m = 1; n = 2; low = 37; hi = 54)

encode 1 using **5 bits**

[**4**]
(m = 1; n = 1; low = 4; hi = 6)

encode 0 using **2 bits**

[**13**]
(m = 1; n = 1; low = 12; hi = 14)

encode 1 using **1 bits**

[**25**]
(m = 1; n = 1; low = 22; hi = 35)

encode 3 using **5 bits**

[**54**]
(m = 1; n = 1; low = 39; hi = 54)

encode 15 using **5 bits**

BINARY INTERPOLATIVE CODING

Example

[3, 4, 7, 11, 13, **15**, 21, 25, 36, 38, 54]

encode 10 using **6 bits**

By *pre-order* visiting the tree, we can obtain:

- the sequence of values that we have encoded
- the associated bits needed for the encoding

[**3**,

encode 3 u

4]

ng **5 bits**

[**4**]

encode 0 using **2 bits**

[**13**]

encode 1 using **1 bits**

[**25**]

encode 3 using **5 bits**

[**54**]

encode 15 using **5 bits**

BINARY INTERPOLATIVE CODING

Example

pre-order visiting

START

[3, 4, 7, 11, 13, **15**, 21, 25, 36, 38, 54]

encode 10 using **6 bits**

[3, 4, **7**, 11, 13]

encode 5 using **4 bits**

[21, 25, **36**, 38, 54]

encode 18 using **6 bits**

[**3**, 4]

encode 3 using **3 bits**

[**11**, 13]

encode 3 using **3 bits**

[**21**, 25]

encode 5 using **5 bits**

[**38**, 54]

encode 1 using **5 bits**

[**4**]

encode 0 using **2 bits**

[**13**]

encode 1 using **1 bits**

[**25**]

encode 3 using **5 bits**

[**54**] **END**

encode 15 using **5 bits**

BINARY INTERPOLATIVE CODING

Example

pre-order visiting

START

[3, 4, 7, 11, 13, **15**, 21, 25, 36, 38, 54]

encode 10 using **6 bits**

[3, 4, **7**, 11, 13]

encode 5 using **4 bits**

[21, 25, **36**, 38, 54]

encode 18 using **6 bits**

[**3**, 4]

encode 3 using **3 bits**

[**11**, 13]

encode 3 using **3 bits**

[**21**, 25]

encode 5 using **5 bits**

[**38**, 54]

encode 1 using **5 bits**

[**4**]

encode 0 using **2 bits**

[**13**]

encode 1 using **1 bits**

[**25**]

encode 3 using **5 bits**

[**54**] **END**

encode 15 using **5 bits**

BINARY INTERPOLATIVE CODING

Example

pre-order visiting

START

[3, 4, 7, 11, 13, **15**, 21, 25, 36, 38, 54]

encode 10 using **6 bits**

[3, 4, **7**, 11, 13]

encode 5 using **4 bits**

[**3**, 4]

encode 3 using **3 bits**

[**4**]

encode 0 using **2 bits**

[**11**, 13]

encode 3 using **3 bits**

[**13**]

encode 1 using **1 bits**

[21, 25, **36**, 38, 54]

encode 18 using **6 bits**

[**21**, 25]

encode 5 using **5 bits**

[**25**]

encode 3 using **5 bits**

[**38**, 54]

encode 1 using **5 bits**

[**54**]

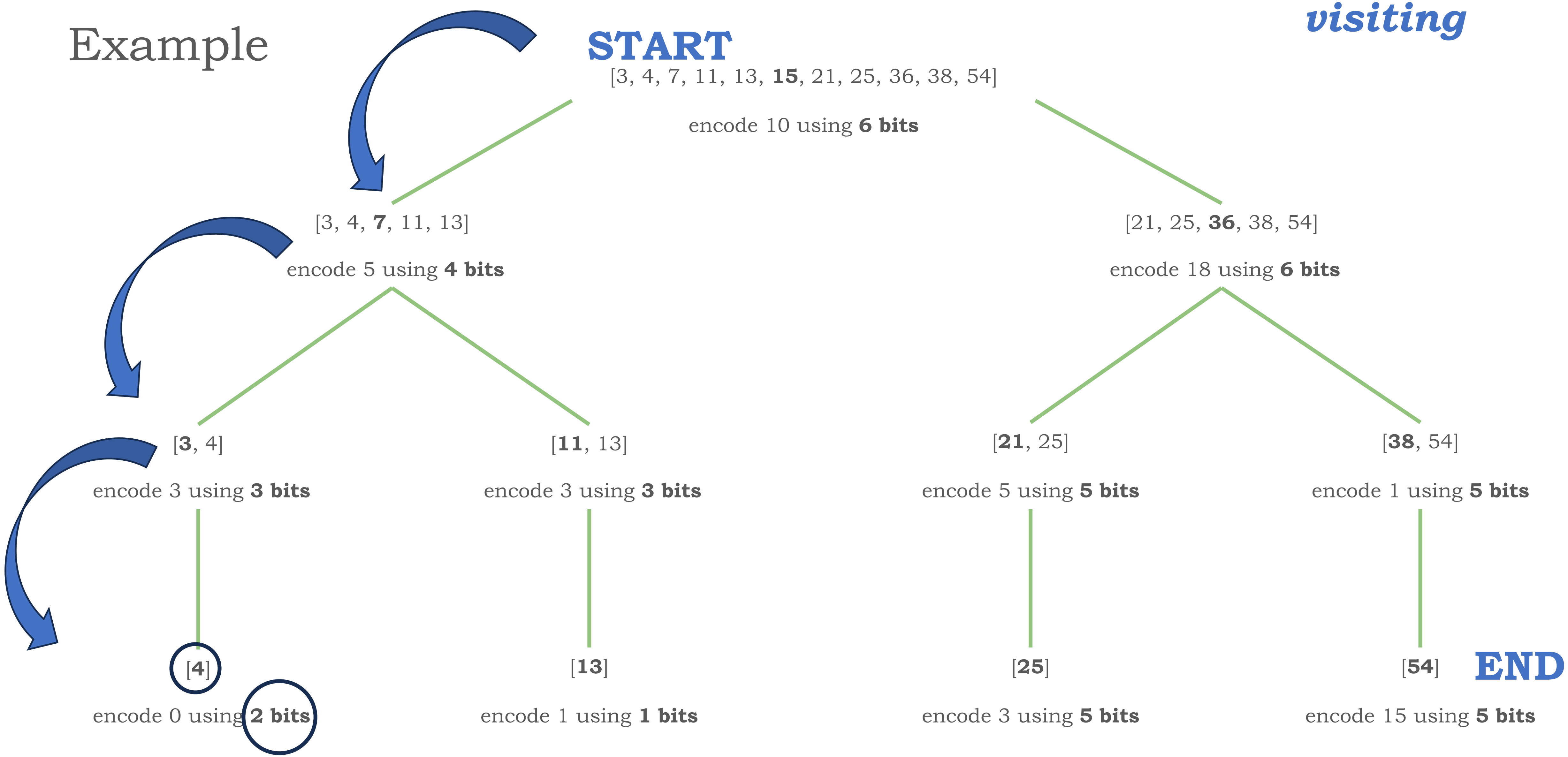
encode 15 using **5 bits**

END

BINARY INTERPOLATIVE CODING

Example

pre-order visiting



BINARY INTERPOLATIVE CODING

Example

pre-order visiting

START

[3, 4, 7, 11, 13, **15**, 21, 25, 36, 38, 54]

encode 10 using **6 bits**

[3, 4, **7**, 11, 13]

encode 5 using **4 bits**

[**3**, 4]

encode 3 using **3 bits**

[**4**]

encode 0 using **2 bits**

[**11**, 13]

encode 3 using **3 bits**

[**13**]

encode 1 using **1 bits**

[21, 25, **36**, 38, 54]

encode 18 using **6 bits**

[**21**, 25]

encode 5 using **5 bits**

[**25**]

encode 3 using **5 bits**

[**38**, 54]

encode 1 using **5 bits**

[**54**]

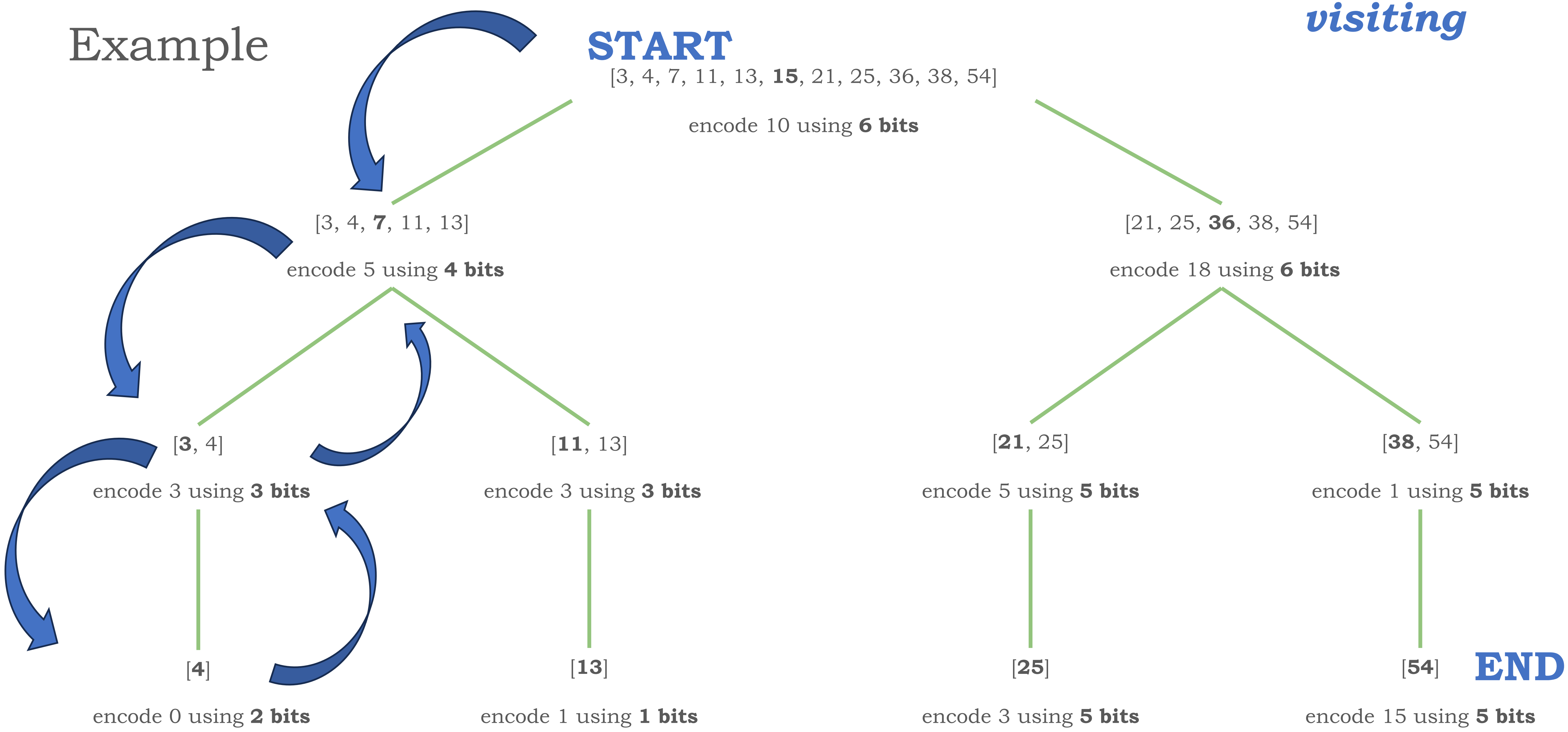
encode 15 using **5 bits**

END

BINARY INTERPOLATIVE CODING

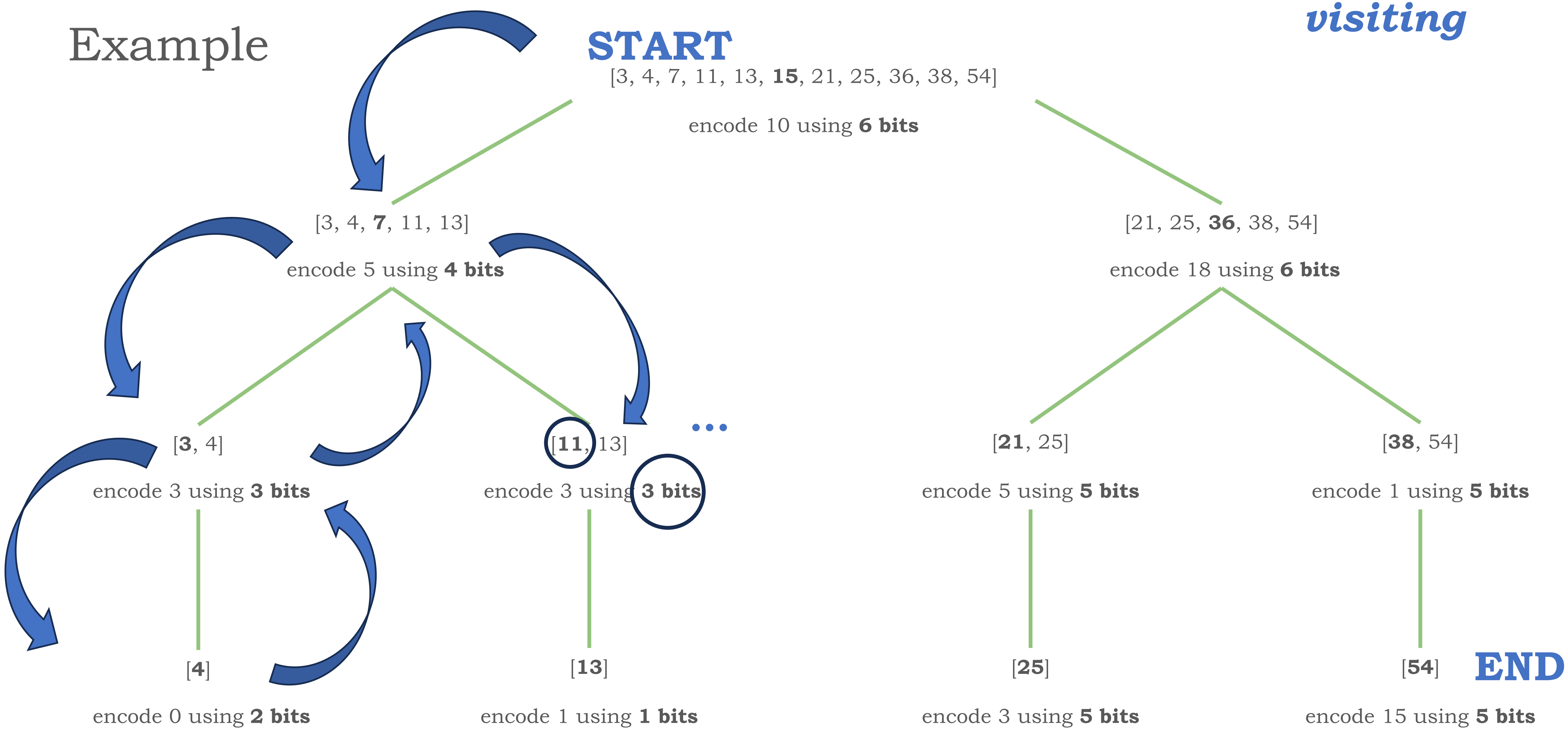
Example

pre-order visiting



BINARY INTERPOLATIVE CODING
Example

pre-order visiting



BINARY INTERPOLATIVE CODING

Example

pre-order visiting

START

[3, 4, 7, 11, 13, **15**, 21, 25, 36, 38, 54]

encode 10 using **6 bits**

[3, 4, **7**, 11, 13]

encode 5 using **4 bits**

[21, 25, **36**, 38, 54]

encode 18 using **6 bits**

[**3**, 4]

encode 3 using **3**

[**38**, 54]

encode 1 using **5 bits**

[**4**]

encode 0 using **2 bits**

[**13**]

encode 1 using **1 bits**

[**25**]

encode 3 using **5 bits**

[**54**] **END**

encode 15 using **5 bits**

The sequence of values that we have encoded:
[10, 5, 3, 0, 3, 1, 18, 5, 3, 1, 15]

The associated bits needed for the encoding:
[6, 4, 3, 2, 3, 1, 6, 5, 5, 5, 5]

SIMPLE - 9

SIMPLE - 9

- Encodes multiple elements **all at the same time**
- Works with *d*-gaps lists
- Uses **fixed memory units** $\longrightarrow$ 32-bit units
- Packs as many integers as possible into each unit

SIMPLE - 9
How to

Out of the 32 bits:

- 4 bits used for the *selector*
- 28 bits used to store data

The *selector* indicates the number of integers stored in the
28 bits assuming that **each integer takes the same number of bits**.

SIMPLE - 9
How to

9 possible ways of packing integers in 28 bits

4-bit selector	Integers	Bits per integer	Wasted bits
0000	28	1	0
0001	14	2	0
0010	9	3	1
0011	7	4	0
0100	5	5	3
0101	4	7	0
0110	3	9	1
0111	2	14	0
1000	1	28	0

SIMPLE - 9
How to

9 possible ways of packing intege

4-bit selector	Integers
0000	28
0001	14
0010	9
0011	7
0100	5
0101	4
0110	3
0111	2
1000	1

- The *d*-gap list can be encoded using multiple 32-bit units
- Different memory units can follow a different *selector*
- In a single unit each integer takes the same amount of bits

SIMPLE - 9
Example

1. Initial postings list:

S=[4, 10, 11, 12, 15, 20, 21, 28, 29, 42, 62, 63, 75, 95]

SIMPLE - 9

Example

1. Initial postings list:

S=[4, 10, 11, 12, 15, 20, 21, 28, 29, 42, 62, 63, 75, 95]

2. Corresponding *d*-gap list:

D = [4, 6, 1, 1, 3, 5, 1, 7, 1, 13, 20, 1, 12, 20]

SIMPLE - 9
Example

1. Initial postings list:

S=[4, 10, 11, 12, 15, 20, 21, 28, 29, 42, 62, 63, 75, 95]

2. Corresponding *d*-gap list:

D = [4, 6, 1, 1, 3, 5, 1, 7, 1, 13, 20, 1, 12, 20]

|D| = 14 integers to encode **using 32-bit memory units**

SIMPLE - 9

Example

D = [4, 6, 1, 1, 3, 5, 1, 7, 1, 13, 20, 1, 12, 20]

4-bit selector	Integers	Bits per integer	Wasted bits
0000	28	1	0
0001	14	2	0
0010	9	3	1
0011	7	4	0
0100	5	5	3
0101	4	7	0
0110	3	9	1
0111	2	14	0
1000	1	28	0

SIMPLE - 9

Example

D = [4, 6, 1, 1, 3, 5, 1, 7, 1, 13, 20, 1, 12, 20]

Option 1: we can pack 14 integers if each takes 2 bits

4-bit selector	Integers	Bits per integer	Wasted bits
0000	28	1	0
0001	14	2	0
0010	9	3	1
0011	7	4	0
0100	5	5	3
0101	4	7	0
0110	3	9	1
0111	2	14	0
1000	1	28	0

SIMPLE - 9
Example

D = [4, 6, 1, 1, 3, 5, 1, 7, 1, 13, 20, 1, 12, 20]

Option 1: we can pack 14 integers if each takes 2 bits

4-bit selector	Integers	Bits per integer	Wasted bits
0000	28	1	0
0001	14	2	0
0010	9	3	1
0011	7	4	0
0100	5	5	3
0101	4	7	0
0110	3	9	1
0111	2	14	0
1000	1	28	0

An integer takes 2 bits if its value is $< 2^2 = 4$

SIMPLE - 9
Example

4-bit selector	Integers	Bits per integer	Wasted bits
0000	28	1	0
0001	14	2	0
0010	9	3	1
0011	7	4	0
0100	5	5	3
0101	4	7	0
0110	3	9	1
0111	2	14	0
1000	1	28	0

D = [4, 6, 1, 1, 3, 5, 1, 7, 1, 13, 20, 1, 12, 20]

Option 1: we can pack 14 integers if each takes 2 bits

An integer takes 2 bits if its value is $< 2^2 = 4$

We have many values that are not < 4 ✗

SIMPLE - 9
Example

4-bit selector	Integers	Bits per integer	Wasted bits
0000	28	1	0
0001	14	2	0
0010	9	3	1
0011	7	4	0
0100	5	5	3
0101	4	7	0
0110	3	9	1
0111	2	14	0
1000	1	28	0

D = [**4, 6, 1, 1, 3, 5, 1, 7, 1**, 13, 20, 1, 12, 20]

~~Option 1: we can pack 14 integers if each takes 2 bits~~

Option 2: we can pack the first 9 integers if each takes 3 bits

SIMPLE - 9
Example

D = [**4, 6, 1, 1, 3, 5, 1, 7, 1**, 13, 20, 1, 12, 20]

~~Option 1: we can pack 14 integers if each takes 2 bits~~

Option 2: we can pack the first 9 integers if each takes 3 bits

4-bit selector	Integers	Bits per integer	Wasted bits
0000	28	1	0
0001	14	2	0
0010	9	3	1
0011	7	4	0
0100	5	5	3
0101	4	7	0
0110	3	9	1
0111	2	14	0
1000	1	28	0

An integer takes 3 bits if its value is $< 2^3 = 8$

SIMPLE - 9

Example

4-bit selector	Integers	Bits per integer	Wasted bits
0000	28	1	0
0001	14	2	0
0010	9	3	1
0011	7	4	0
0100	5	5	3
0101	4	7	0
0110	3	9	1
0111	2	14	0
1000	1	28	0

D = [**4, 6, 1, 1, 3, 5, 1, 7, 1**, 13, 20, 1, 12, 20]

~~Option 1: we can pack 14 integers if each takes 2 bits~~

Option 2: we can pack the first 9 integers if each takes 3 bits

An integer takes 3 bits if its value is $< 2^3 = 8$

All of our first 9 integers are < 8

SIMPLE - 9
Example

D = [**4, 6, 1, 1, 3, 5, 1, 7, 1**, 13, 20, 1, 12, 20]

Stored in **one** 32-bit unit as:

0010 011 101 000 000 010 100 000 110 0006

The 3rd selector

4 bits

The 9 integers

27 bits

1 unused bit

4-bit selector	Integers	Bits per integer	Wasted bits
0000	28	1	0
0001	14	2	0
0010	**9**	**3**	**1**
0011	7	4	0
0100	5	5	3
0101	4	7	0
0110	3	9	1
0111	2	14	0
1000	1	28	0

SIMPLE - 9
Example

D = [4, 6, 1, 1, 3, 5, 1, 7, 1, **13, 20, 1, 12, 20**]

5 integers left

4-bit selector	Integers	Bits per integer	Wasted bits
0000	28	1	0
0001	14	2	0
0010	9	3	1
0011	7	4	0
0100	5	5	3
0101	4	7	0
0110	3	9	1
0111	2	14	0
1000	1	28	0

SIMPLE - 9
Example

D = [4, 6, 1, 1, 3, 5, 1, 7, 1, **13, 20, 1, 12, 20**]

5 integers left

Option 1: we can pack 5 integers if each takes 5 bits

4-bit selector	Integers	Bits per integer	Wasted bits
0000	28	1	0
0001	14	2	0
0010	9	3	1
0011	7	4	0
0100	5	5	3
0101	4	7	0
0110	3	9	1
0111	2	14	0
1000	1	28	0

SIMPLE - 9
Example

4-bit selector	Integers	Bits per integer	Wasted bits
0000	28	1	0
0001	14	2	0
0010	9	3	1
0011	7	4	0
0100	5	5	3
0101	4	7	0
0110	3	9	1
0111	2	14	0
1000	1	28	0

D = [4, 6, 1, 1, 3, 5, 1, 7, 1, **13, 20, 1, 12, 20**]

5 integers left

Option 1: we can pack 5 integers if each takes 5 bits

An integer takes 5 bits if its value is < $2^5 = 32$

SIMPLE - 9
Example

4-bit selector	Integers	Bits per integer	Wasted bits
0000	28	1	0
0001	14	2	0
0010	9	3	1
0011	7	4	0
0100	5	5	3
0101	4	7	0
0110	3	9	1
0111	2	14	0
1000	1	28	0

D = [4, 6, 1, 1, 3, 5, 1, 7, 1, **13, 20, 1, 12, 20**]

5 integers left

Option 1: we can pack 5 integers if each takes 5 bits

An integer takes 5 bits if its value is < $2^5 = 32$

All of our 5 integers are < 32

SIMPLE - 9

Example

4-bit selector	Integers	Bits per integer	Wasted bits
0000	28	1	0
0001	14	2	0
0010	9	3	1
0011	7	4	0
0100	**5**	**5**	**3**
0101	4	7	0
0110	3	9	1
0111	2	14	0
1000	1	28	0

D = [4, 6, 1, 1, 3, 5, 1, 7, 1, **13, 20, 1, 12, 20**]

Stored in another 32-bit unit as:

0100 01100 10011 00000 01011 10011

The 5th selector

4 bits

The 5 integers

25 bits

3 unused bit

SIMPLE - 9
Example

D = [4, 6, 1, 1, 3, 5, 1, 7, 1, **13, 20, 1, 12, 20**]

4-bit selector	Integers	Bits per integer	Wasted bits
0000	28	1	0
0001	14	2	0
0010	9	3	1
0011	7	4	0
0100	**5**	**5**	**3**
0101	4	7	0
		9	1
		14	0
		28	0

Total used space is **two 32-bit memory units:**

- One using the 3rd selector
- One using the 5th selector

COMPARISON

COMPARISON

Study from Moffat and Anh (2005)

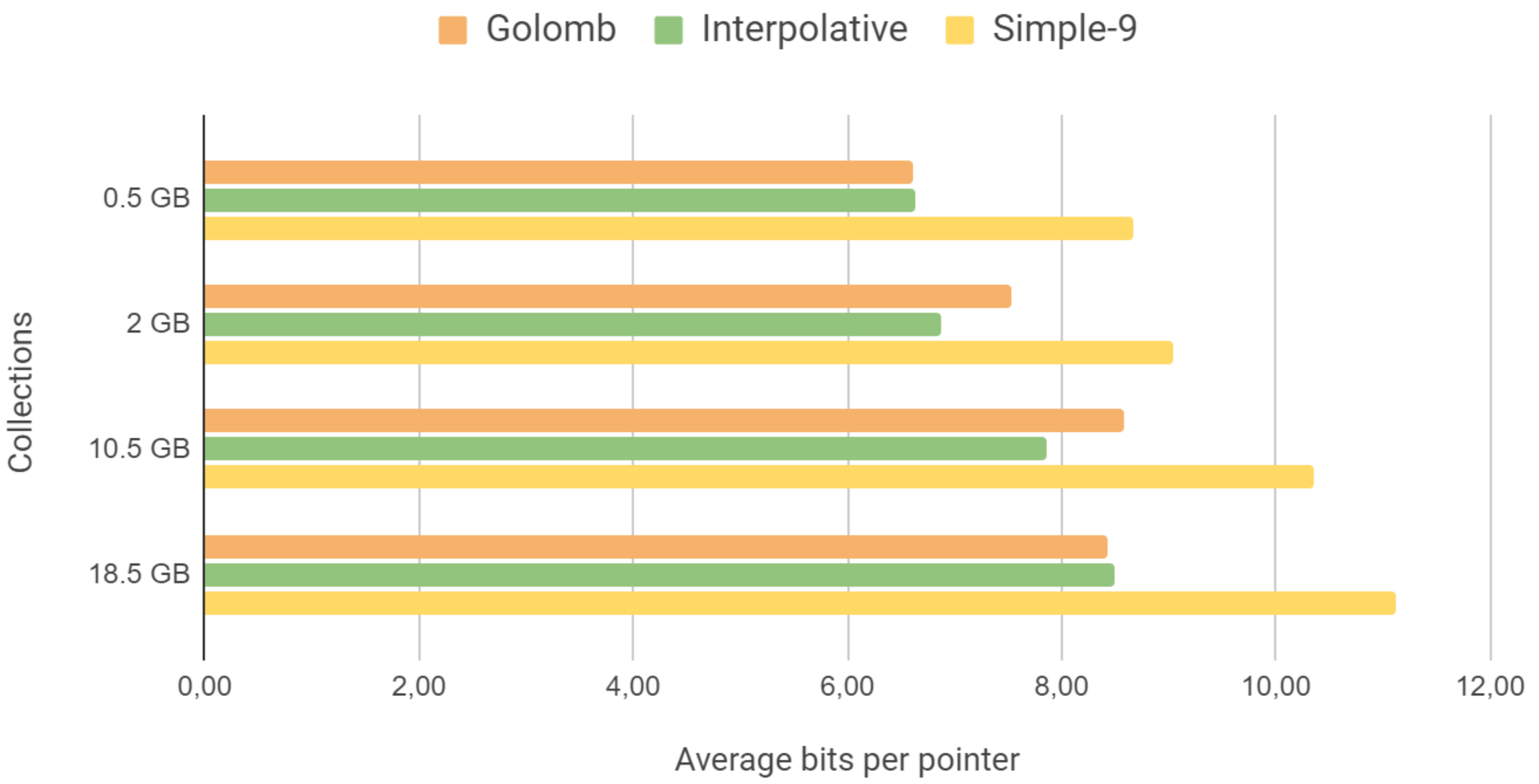
Comparing the 3 methods over different size textual data collections:

0.5GB 2GB 10.5GB 18.5GB

Object of the comparison:

- **Compression effectiveness** → **space**
  Measured as average number of bits per pointer
- **Query processing speed** → **time**
  Measured as average elapsed time between when:
  1) the query is received
  2) the result is output

COMPARISON
Compression effectiveness

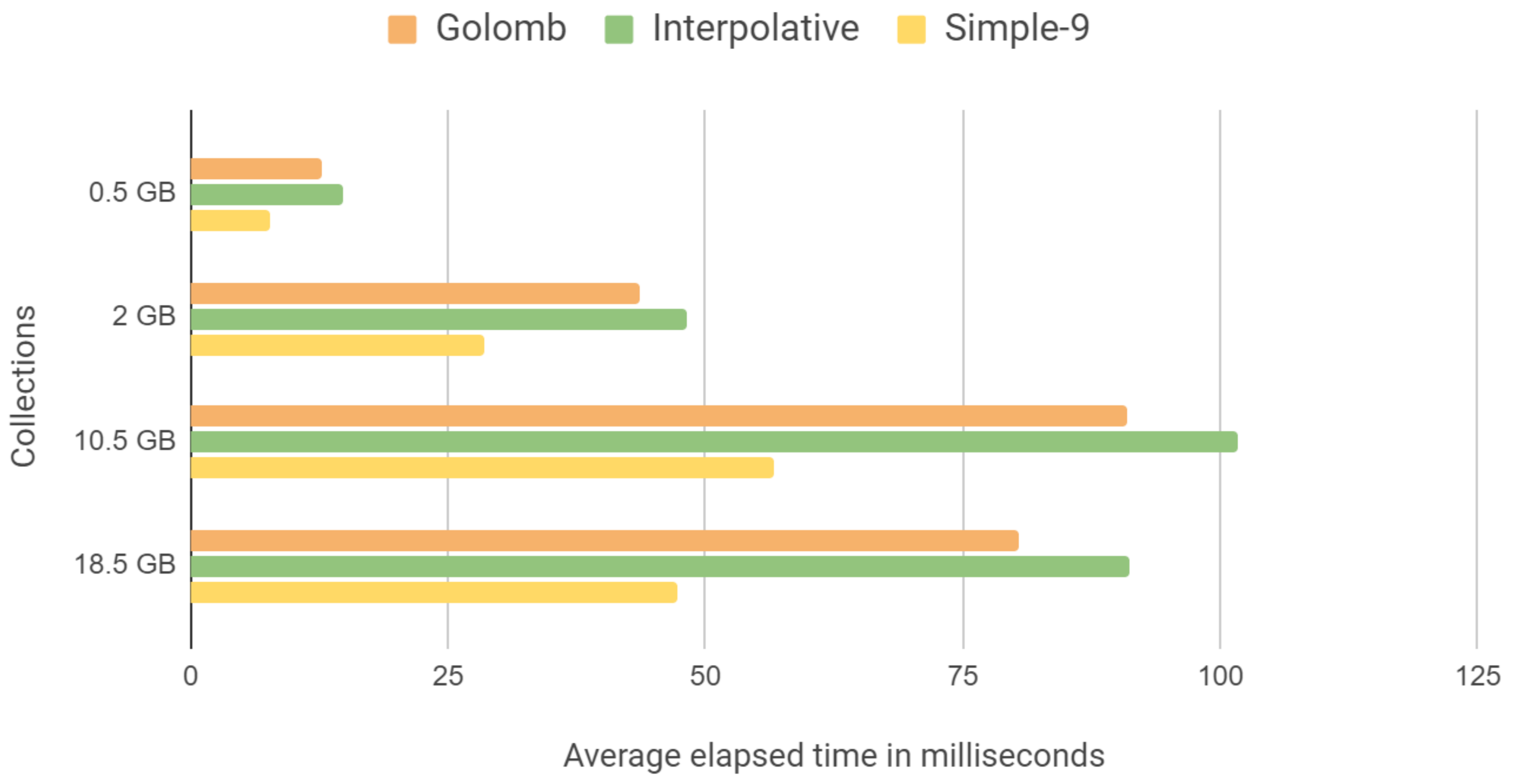


The Binary Interpolative Coding is the best one

least bits needed

The Simple – 9 is the one that requires most space for the encoding

COMPARISON
Query processing speed



The Simple – 9 is by far the best one

least processing time needed

The Binary Interpolative Coding is the one that requires most time for query processing

CONCLUSION

There is **no best** technique

Always consider **trade-off** between:

- saved space
- processing time → affected by decoding speed!

Choose based on data that you have:

- <u>Golomb</u>: optimal if documents are randomly scattered
- <u>Binary Interpolative</u>: optimal if documents are clustered
- <u>Simple-9</u>: optimal for long postings lists with small *d*-gaps

THANK YOU