Elias-Fano Encoding

A powerful tool for data structure design

Giulio Ermanno Pibiri
giulio.pibiri@di.unipi.it
University of Pisa, and ISTI-CNR

Tokyo, 10/04/2018

Problem

Consider a sequence S[0,n) of n *positive* and *monotonically increasing integers*, i.e., S[i-1] ≤ S[i] for 1 ≤ i ≤ n-1, possibly repeated.

How to represent it as a *bit vector* in which each original integer is *self-delimited*, using as few as possible bits?

Problem

Consider a sequence S[0,n) of n *positive* and *monotonically increasing integers*, i.e., $S[i-1] \leq S[i]$ for $1 \leq i \leq n-1$, possibly repeated.

How to represent it as a *bit vector* in which each original integer is *self-delimited*, using as few as possible bits?

Huge research corpora describing different space/time trade-offs.

- Elias gamma/delta [Elias-1974]
- Variable Byte [Salomon-2007]
- Varint-G8IU [Stepanov et al.-2011]
- Simple-9/16 [Anh and Moffat 2005-2010]
- PForDelta (PFD) [Zukowski et al.-2006]
- OptPFD [Yan et al.-2009]
- Binary Interpolative Coding [Moffat and Stuiver-2000]

Inverted Indexes

Given a *textual collection* D, each document can be seen as a (multi-)set of terms. The set of terms occurring in D is the *lexicon* T.

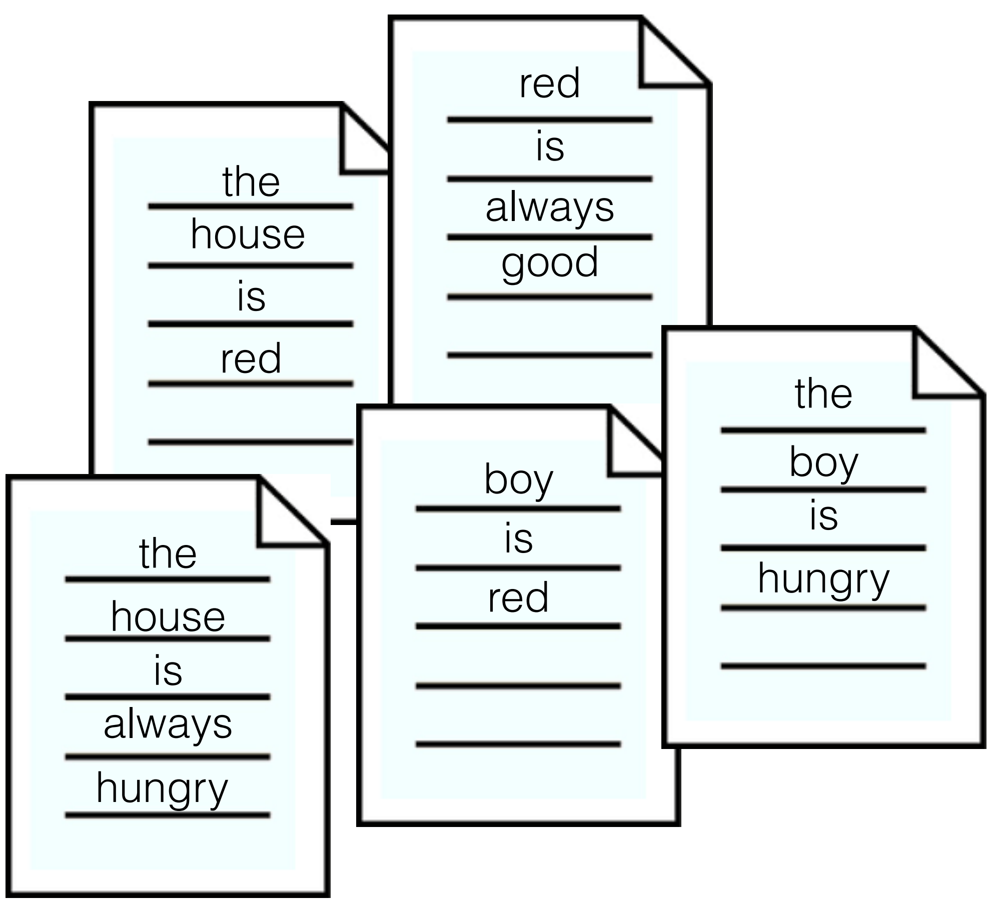
For each term t in T we store in a list L_t the identifiers of the documents in which t appears.

The collection of all inverted lists $\{L_{t_1},\ldots,L_{t_T}\}$ is the inverted index.

Given a *textual collection* D, each document can be seen as a (multi-)set of terms. The set of terms occurring in D is the *lexicon* T.

For each term t in T we store in a list L_t the identifiers of the documents in which t appears.

The collection of all inverted lists $\{L_{t_1},\ldots,L_{t_T}\}$ is the inverted index.

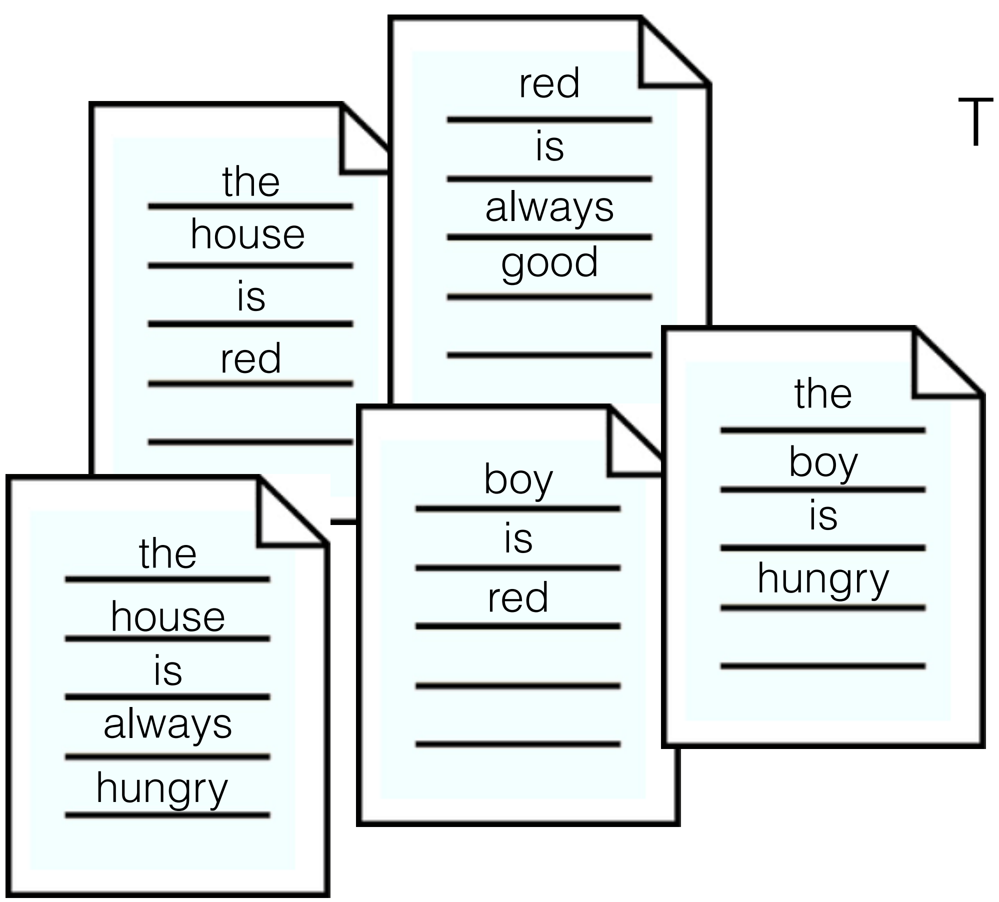


Inverted Indexes

Given a *textual collection* D, each document can be seen as a (multi-)set of terms. The set of terms occurring in D is the *lexicon* T.

For each term t in T we store in a list L_t the identifiers of the documents in which t appears.

The collection of all inverted lists $\{L_{t_1},\dots,L_{t_T}\}$ is the inverted index.



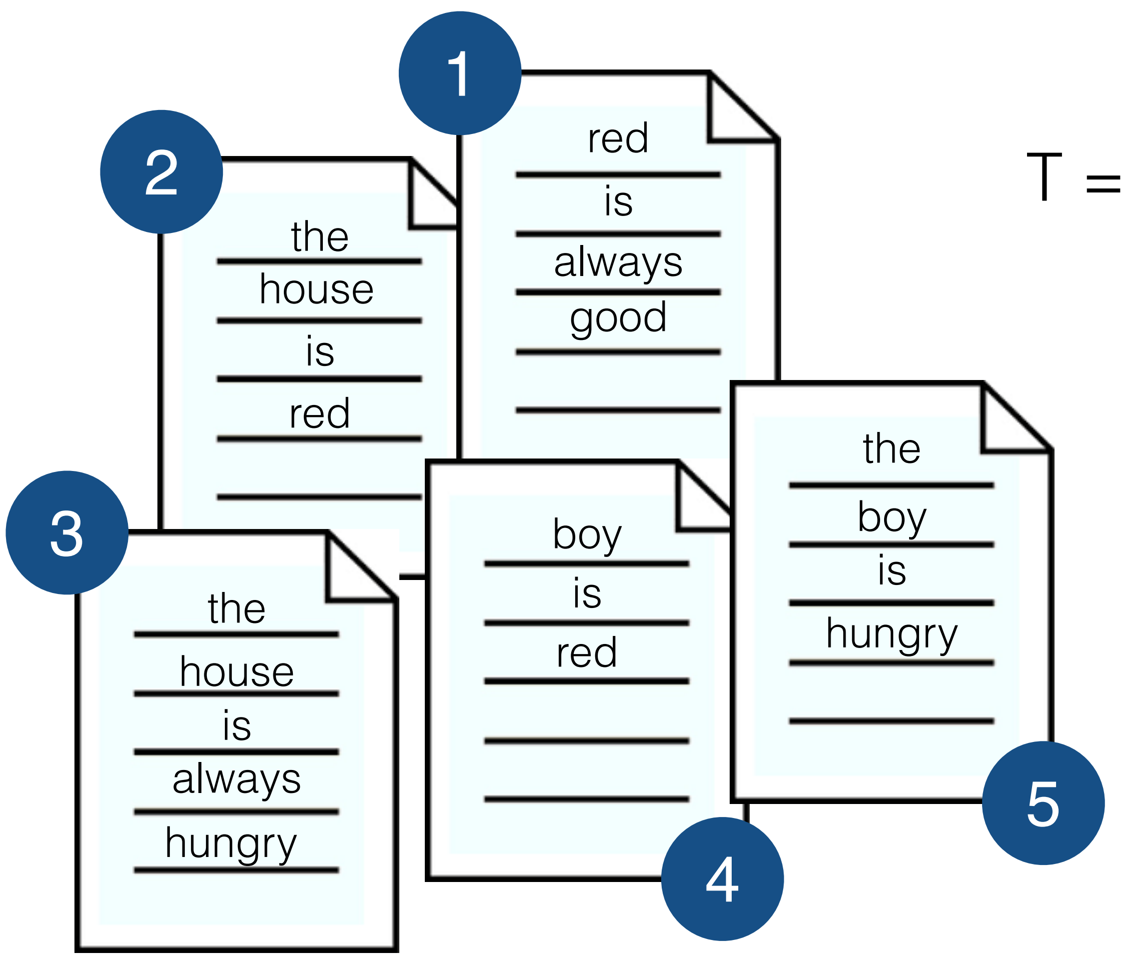
$$T = \{\overset{t_1}{\text{always}}, \overset{t_2}{\text{boy}}, \overset{t_3}{\text{good}}, \overset{t_4}{\text{house}}, \overset{t_5}{\text{hungry}}, \overset{t_6}{\text{is}}, \overset{t_7}{\text{red}}, \overset{t_8}{\text{the}}\}$$

Inverted Indexes

Given a *textual collection* D, each document can be seen as a (multi-)set of terms. The set of terms occurring in D is the *lexicon* T.

For each term t in T we store in a list L_t the identifiers of the documents in which t appears.

The collection of all inverted lists $\{L_{t_1},\ldots,L_{t_T}\}$ is the inverted index.

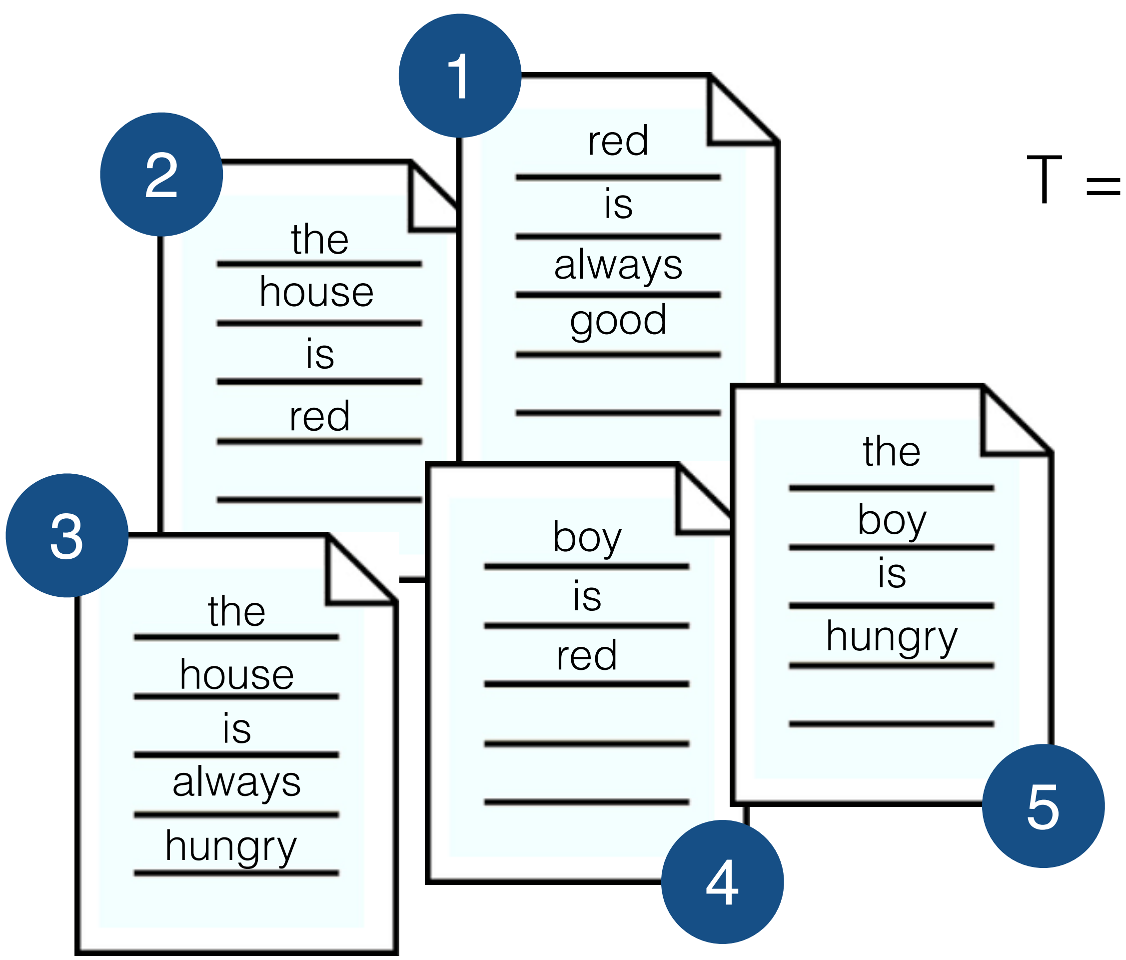


$$T = \{\overset{t_1}{\text{always}}, \overset{t_2}{\text{boy}}, \overset{t_3}{\text{good}}, \overset{t_4}{\text{house}}, \overset{t_5}{\text{hungry}}, \overset{t_6}{\text{is}}, \overset{t_7}{\text{red}}, \overset{t_8}{\text{the}}\}$$

Given a *textual collection* D, each document can be seen as a (multi-)set of terms. The set of terms occurring in D is the *lexicon* T.

For each term t in T we store in a list L_t the identifiers of the documents in which t appears.

The collection of all inverted lists $\{L_{t_1},\ldots,L_{t_T}\}$ is the inverted index.



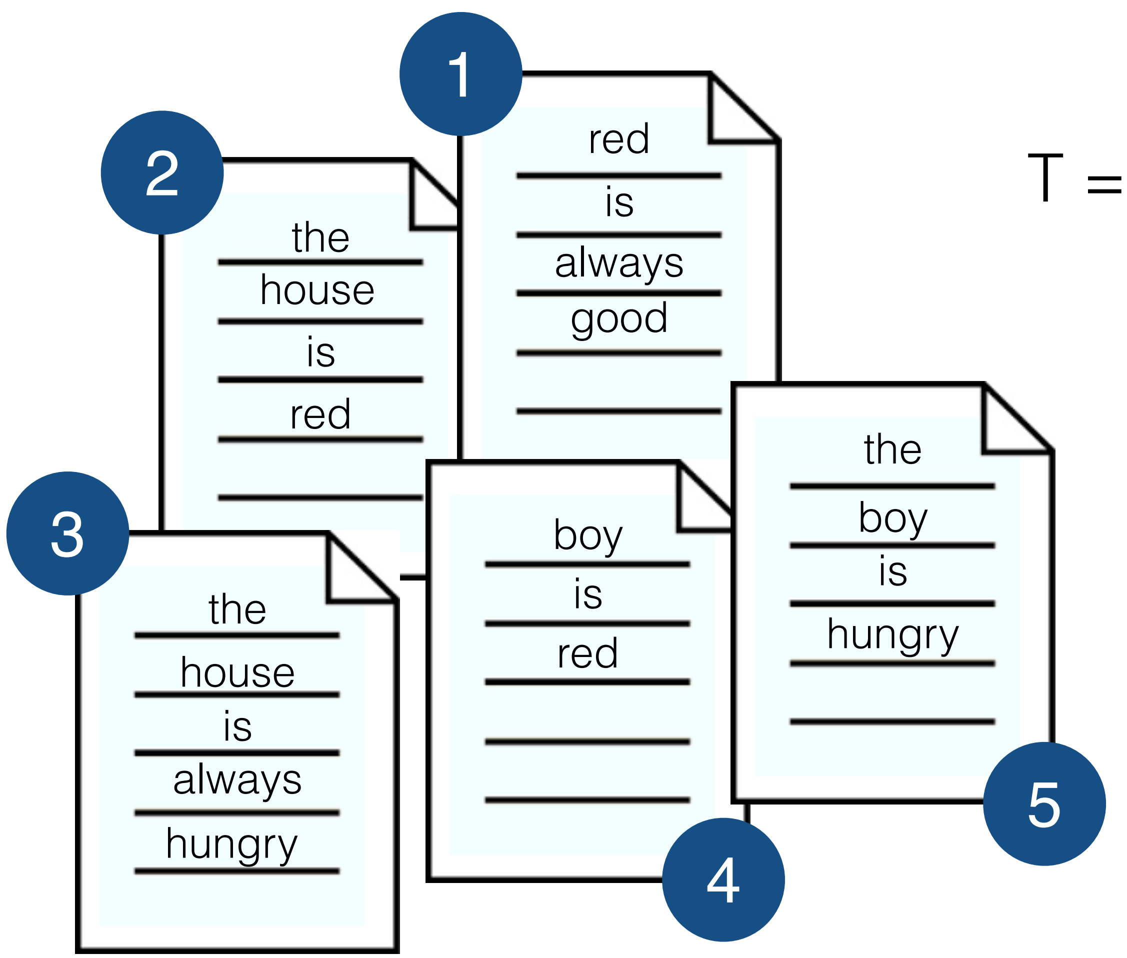
$$T = \{\overset{t_1}{\text{always}}, \overset{t_2}{\text{boy}}, \overset{t_3}{\text{good}}, \overset{t_4}{\text{house}}, \overset{t_5}{\text{hungry}}, \overset{t_6}{\text{is}}, \overset{t_7}{\text{red}}, \overset{t_8}{\text{the}}\}$$

$L_{t_1}=[1, 3]$
$L_{t_2}=[4, 5]$
$L_{t_3}=[1]$
$L_{t_4}=[2, 3]$
$L_{t_5}=[3, 5]$
$L_{t_6}=[1, 2, 3, 4, 5]$
$L_{t_7}=[1, 2, 4]$
$L_{t_8}=[2, 3, 5]$

Inverted Indexes

Inverted Indexes owe their popularity to the *efficient resolution of queries*, such as: “return me all documents in which terms $\{t_1,\ldots,t_k\}$ occur”.

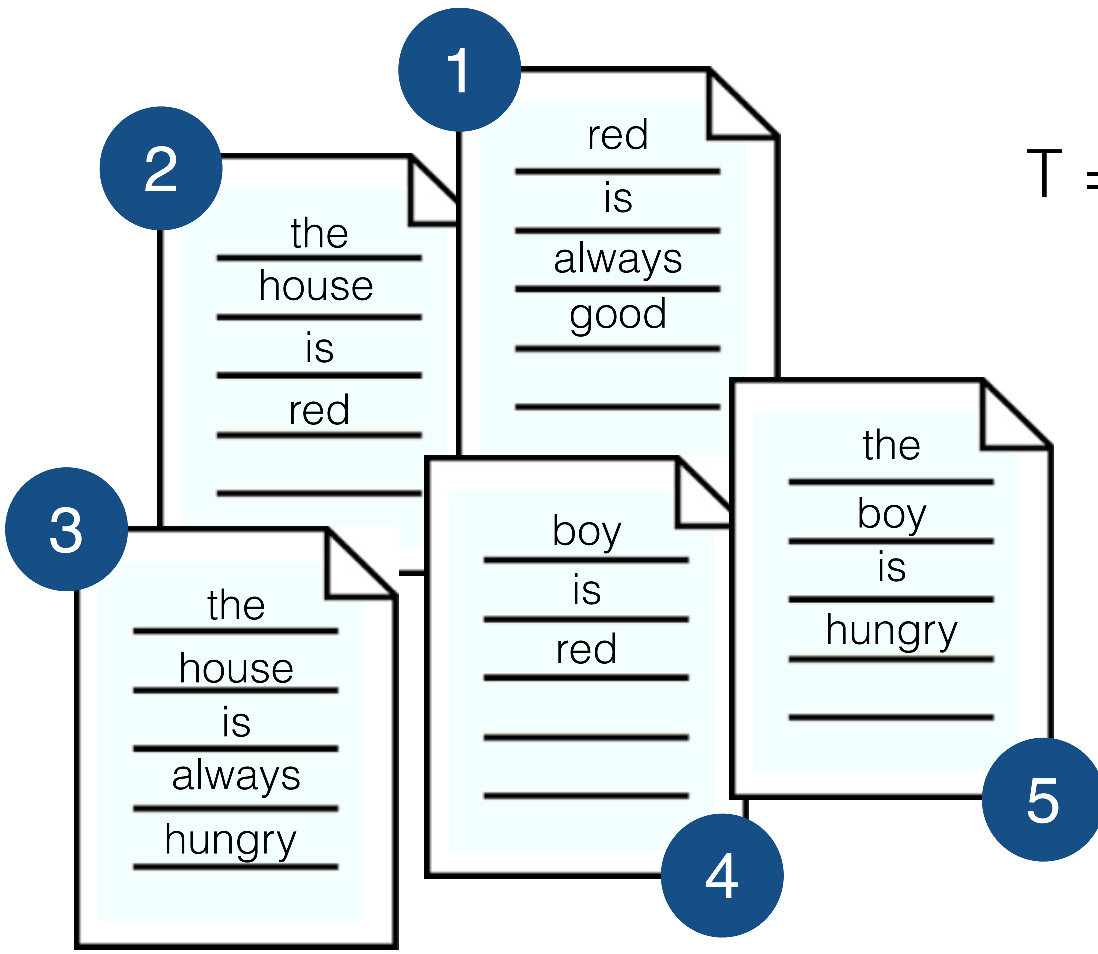
Inverted Indexes owe their popularity to the *efficient resolution of queries*, such as: “return me all documents in which terms $\{t_1,\ldots,t_k\}$ occur”.



$$T = \{\overset{t_1}{\text{always}}, \overset{t_2}{\text{boy}}, \overset{t_3}{\text{good}}, \overset{t_4}{\text{house}}, \overset{t_5}{\text{hungry}}, \overset{t_6}{\text{is}}, \overset{t_7}{\text{red}}, \overset{t_8}{\text{the}}\}$$

$L_{t_1}=[1, 3]$
$L_{t_2}=[4, 5]$
$L_{t_3}=[1]$
$L_{t_4}=[2, 3]$
$L_{t_5}=[3, 5]$
$L_{t_6}=[1, 2, 3, 4, 5]$
$L_{t_7}=[1, 2, 4]$
$L_{t_8}=[2, 3, 5]$

Inverted Indexes owe their popularity to the *efficient resolution of queries*, such as: “return me all documents in which terms $\{t_1,\ldots,t_k\}$ occur”.

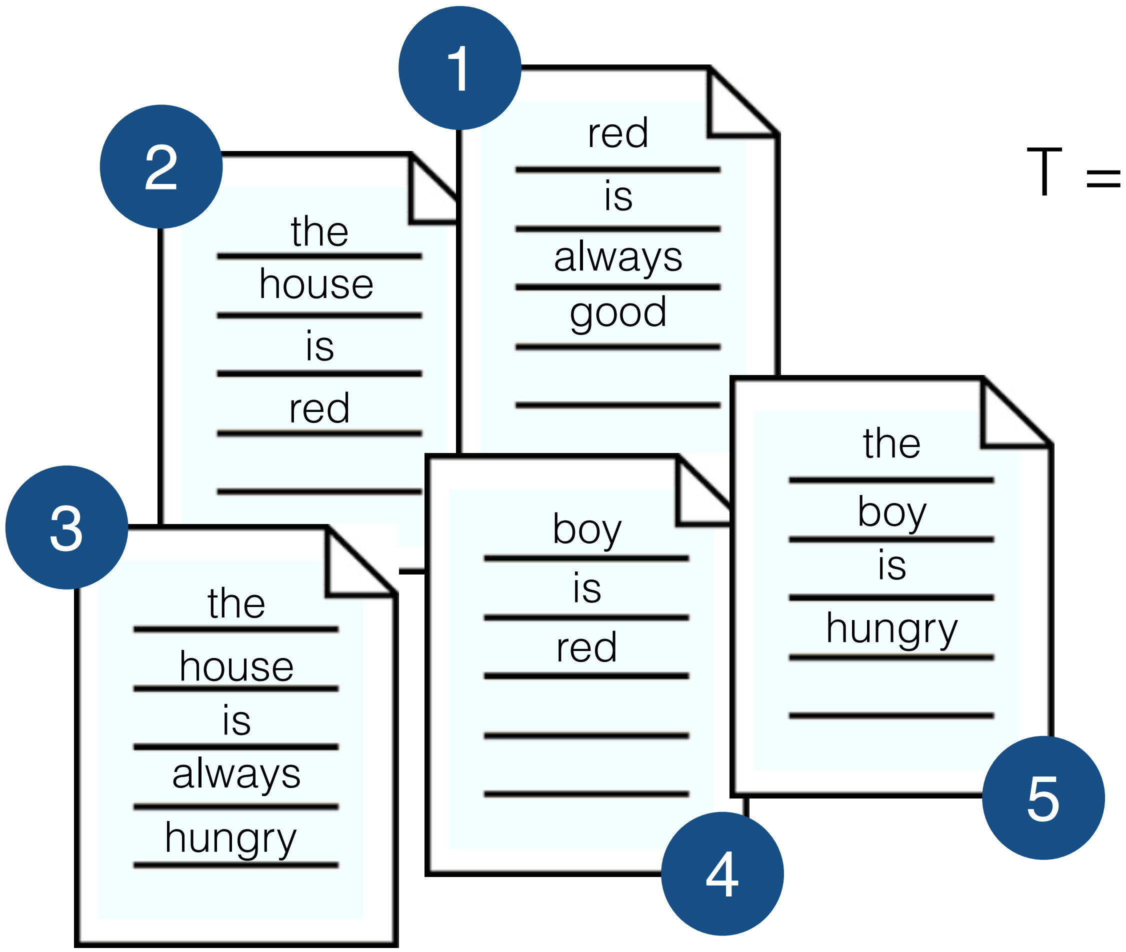


$$T = \{\overset{t_1}{\text{always}}, \overset{t_2}{\text{boy}}, \overset{t_3}{\text{good}}, \overset{t_4}{\text{house}}, \overset{t_5}{\text{hungry}}, \overset{t_6}{\text{is}}, \overset{t_7}{\text{red}}, \overset{t_8}{\text{the}}\}$$

$L_{t_1}=[1, 3]$
$L_{t_2}=[4, 5]$
$L_{t_3}=[1]$
$L_{t_4}=[2, 3]$
$L_{t_5}=[3, 5]$
$L_{t_6}=[1, 2, 3, 4, 5]$
$L_{t_7}=[1, 2, 4]$
$L_{t_8}=[2, 3, 5]$

$q = \{\text{boy, is, the}\}$

Inverted Indexes owe their popularity to the *efficient resolution of queries*, such as: “return me all documents in which terms $\{t_1,\ldots,t_k\}$ occur”.



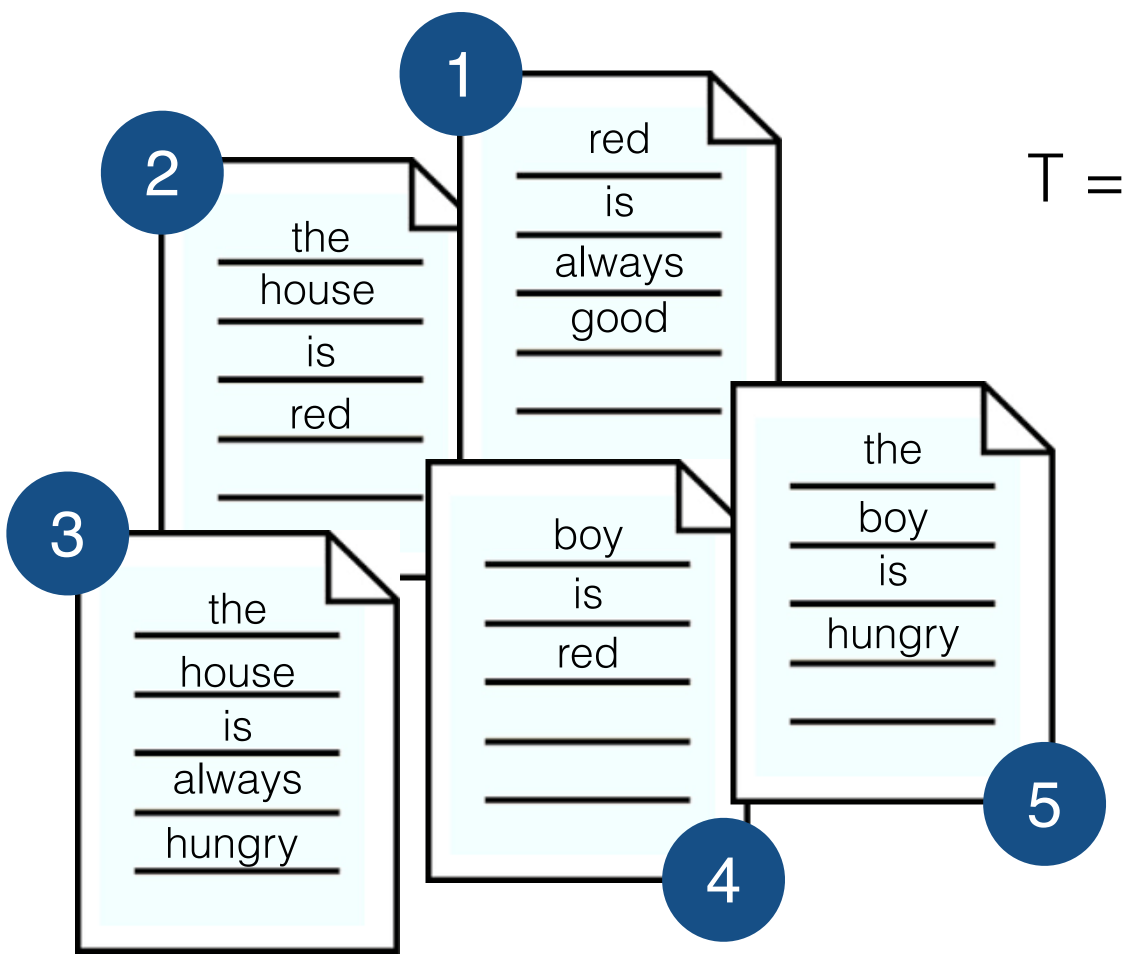
$$T = \{\overset{t_1}{\text{always}}, \overset{t_2}{\text{boy}}, \overset{t_3}{\text{good}}, \overset{t_4}{\text{house}}, \overset{t_5}{\text{hungry}}, \overset{t_6}{\text{is}}, \overset{t_7}{\text{red}}, \overset{t_8}{\text{the}}\}$$

$L_{t_1}=[1, 3]$
$L_{t_2}=[4, 5]$
$L_{t_3}=[1]$
$L_{t_4}=[2, 3]$
$L_{t_5}=[3, 5]$
$L_{t_6}=[1, 2, 3, 4, 5]$
$L_{t_7}=[1, 2, 4]$
$L_{t_8}=[2, 3, 5]$

$q = \{\text{boy, is, the}\}$

Inverted Indexes

Inverted Indexes owe their popularity to the *efficient resolution of queries*, such as: "return me all documents in which terms $\{t_1,\ldots,t_k\}$ occur".



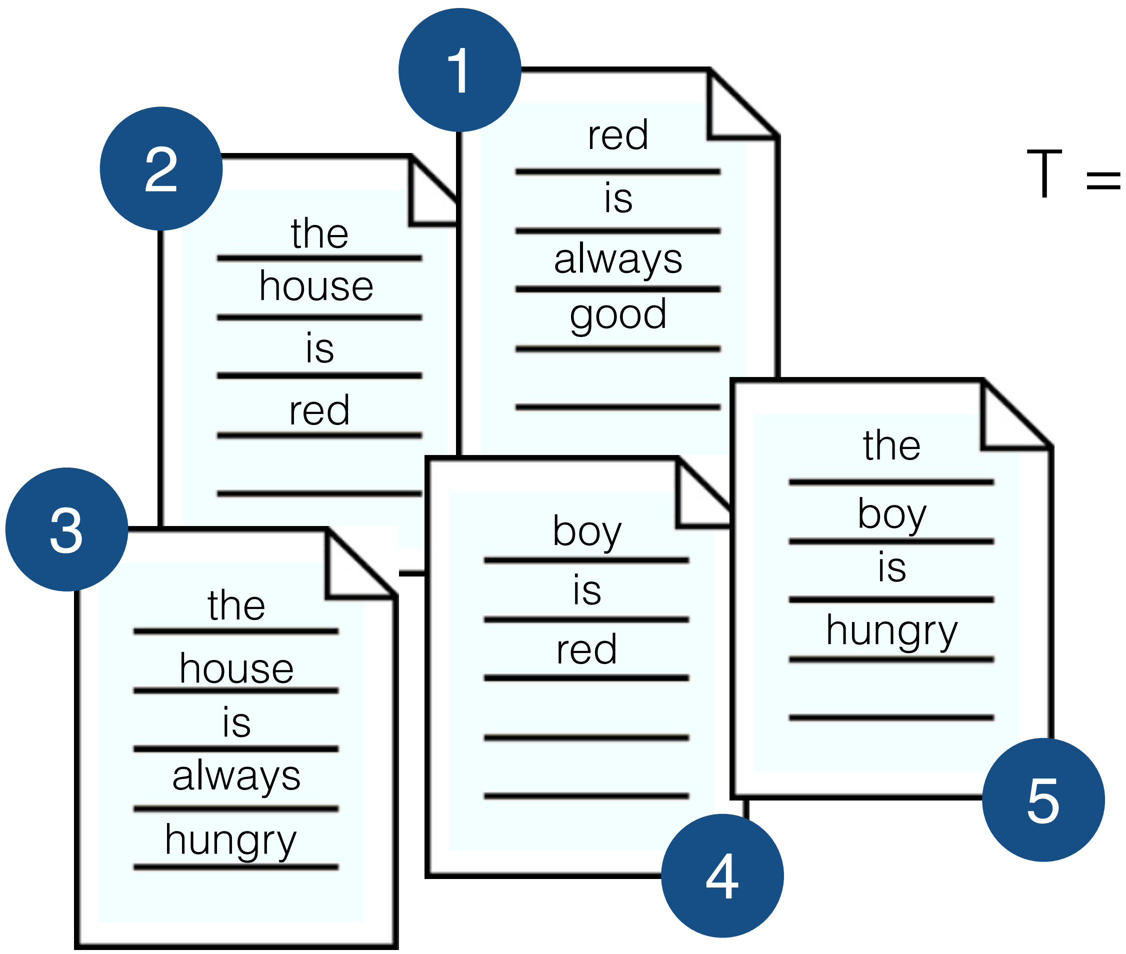
$$T = \{\overset{t_1}{\text{always}}, \overset{t_2}{\text{boy}}, \overset{t_3}{\text{good}}, \overset{t_4}{\text{house}}, \overset{t_5}{\text{hungry}}, \overset{t_6}{\text{is}}, \overset{t_7}{\text{red}}, \overset{t_8}{\text{the}}\}$$

$L_{t_1}=[1, 3]$
$L_{t_2}=[4, 5]$
$L_{t_3}=[1]$
$L_{t_4}=[2, 3]$
$L_{t_5}=[3, 5]$
$L_{t_6}=[1, 2, 3, 4, 5]$
$L_{t_7}=[1, 2, 4]$
$L_{t_8}=[2, 3, 5]$

$q = \{boy, is, the\}$

Inverted Indexes

Inverted Indexes owe their popularity to the *efficient resolution of queries*, such as: “return me all documents in which terms $\{t_1,\ldots,t_k\}$ occur”.



$$T = \{\overset{t_1}{\text{always}}, \overset{t_2}{\text{boy}}, \overset{t_3}{\text{good}}, \overset{t_4}{\text{house}}, \overset{t_5}{\text{hungry}}, \overset{t_6}{\text{is}}, \overset{t_7}{\text{red}}, \overset{t_8}{\text{the}}\}$$

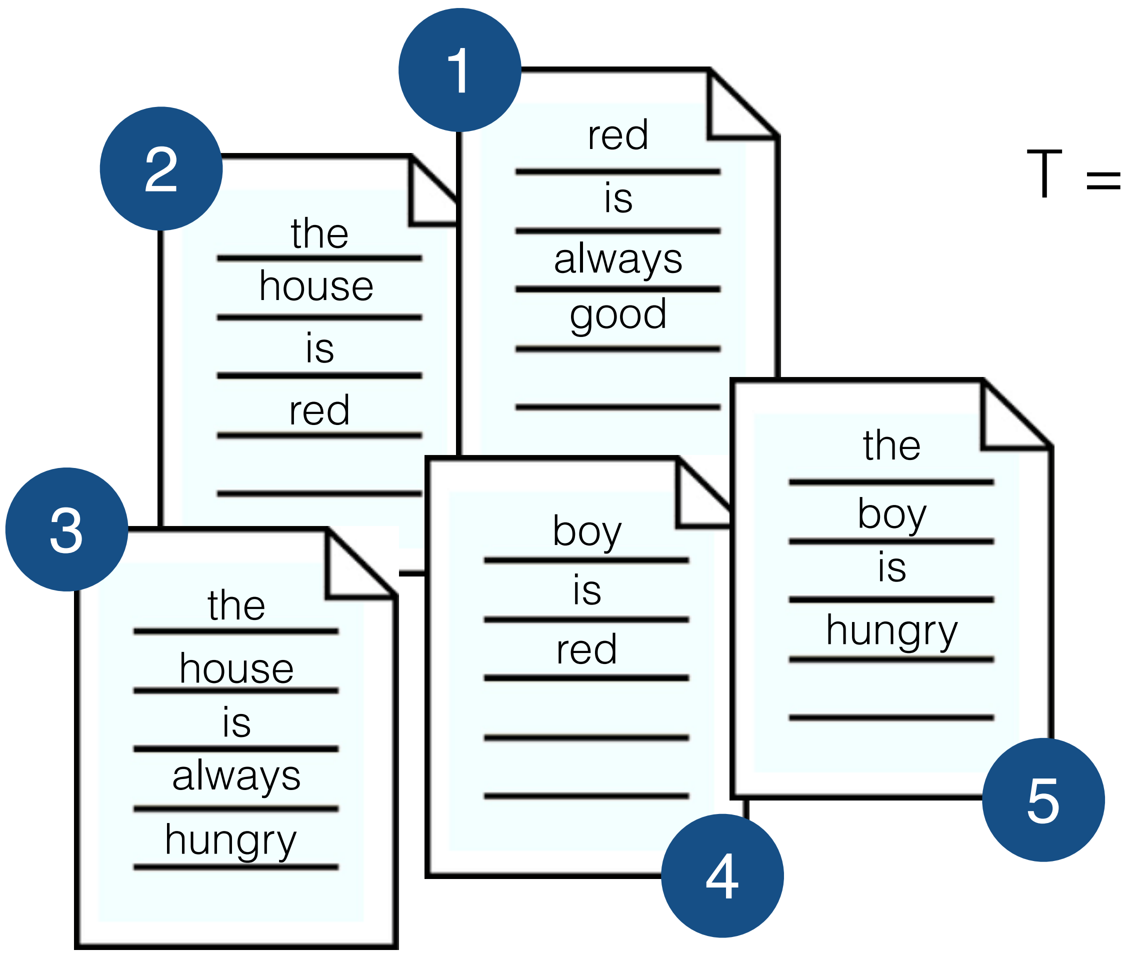
$L_{t_1}=[1, 3]$
$L_{t_2}=[4, 5]$
$L_{t_3}=[1]$
$L_{t_4}=[2, 3]$
$L_{t_5}=[3, 5]$
$L_{t_6}=[1, 2, 3, 4, 5]$
$L_{t_7}=[1, 2, 4]$
$L_{t_8}=[2, 3, 5]$

$q = \{\text{boy, is, the}\}$

$q = \{\text{good, hungry}\}$

Inverted Indexes

Inverted Indexes owe their popularity to the *efficient resolution of queries*, such as: “return me all documents in which terms $\{t_1,\dots,t_k\}$ occur”.



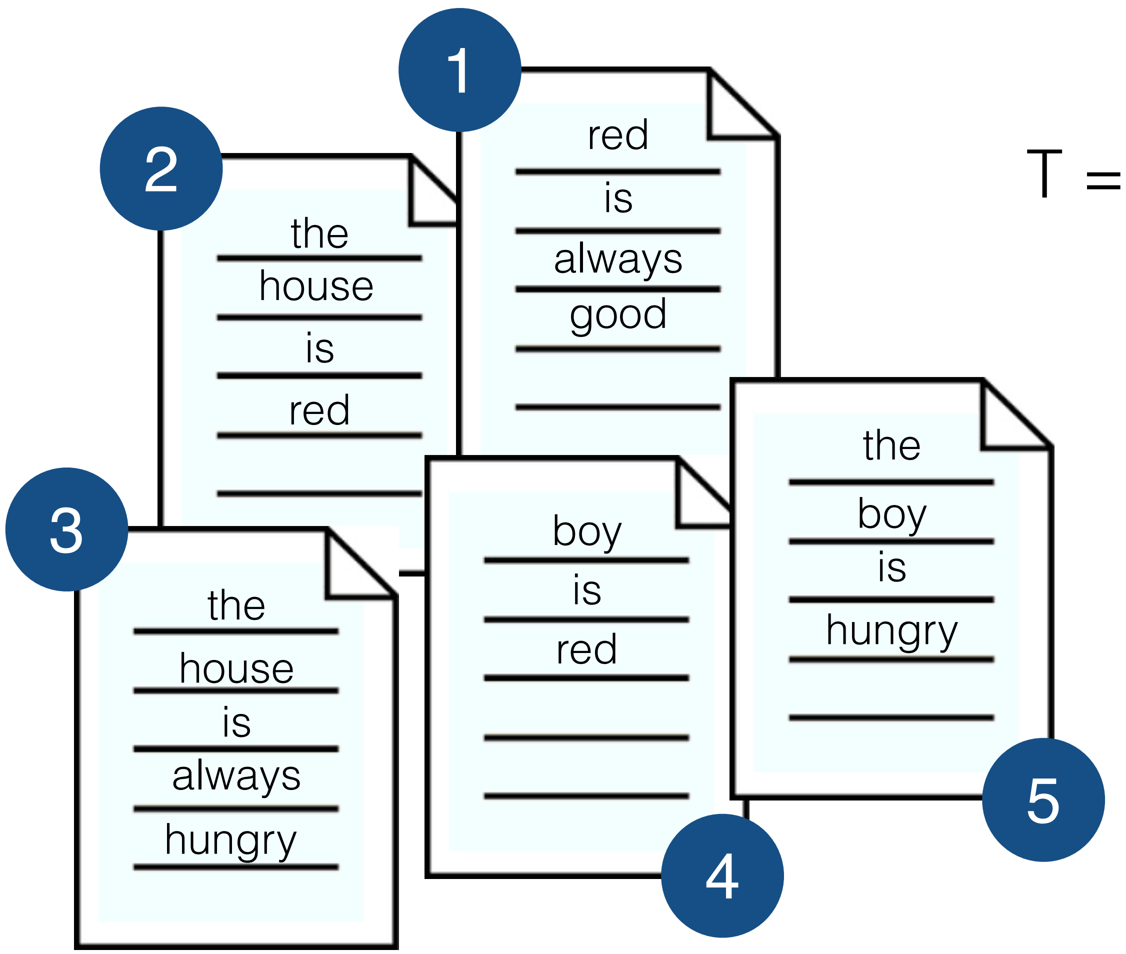
$$T = \{\overset{t_1}{\text{always}}, \overset{t_2}{\text{boy}}, \overset{t_3}{\text{good}}, \overset{t_4}{\text{house}}, \overset{t_5}{\text{hungry}}, \overset{t_6}{\text{is}}, \overset{t_7}{\text{red}}, \overset{t_8}{\text{the}}\}$$

$L_{t_1}=[1, 3]$
$L_{t_2}=[4, 5]$
$L_{t_3}=[1]$
$L_{t_4}=[2, 3]$
$L_{t_5}=[3, 5]$
$L_{t_6}=[1, 2, 3, 4, 5]$
$L_{t_7}=[1, 2, 4]$
$L_{t_8}=[2, 3, 5]$

$q = \{boy, is, the\}$

$q = \{good, hungry\}$

Inverted Indexes owe their popularity to the *efficient resolution of queries*, such as: “return me all documents in which terms $\{t_1,\ldots,t_k\}$ occur”.



$$T = \{\overset{t_1}{\text{always}}, \overset{t_2}{\text{boy}}, \overset{t_3}{\text{good}}, \overset{t_4}{\text{house}}, \overset{t_5}{\text{hungry}}, \overset{t_6}{\text{is}}, \overset{t_7}{\text{red}}, \overset{t_8}{\text{the}}\}$$

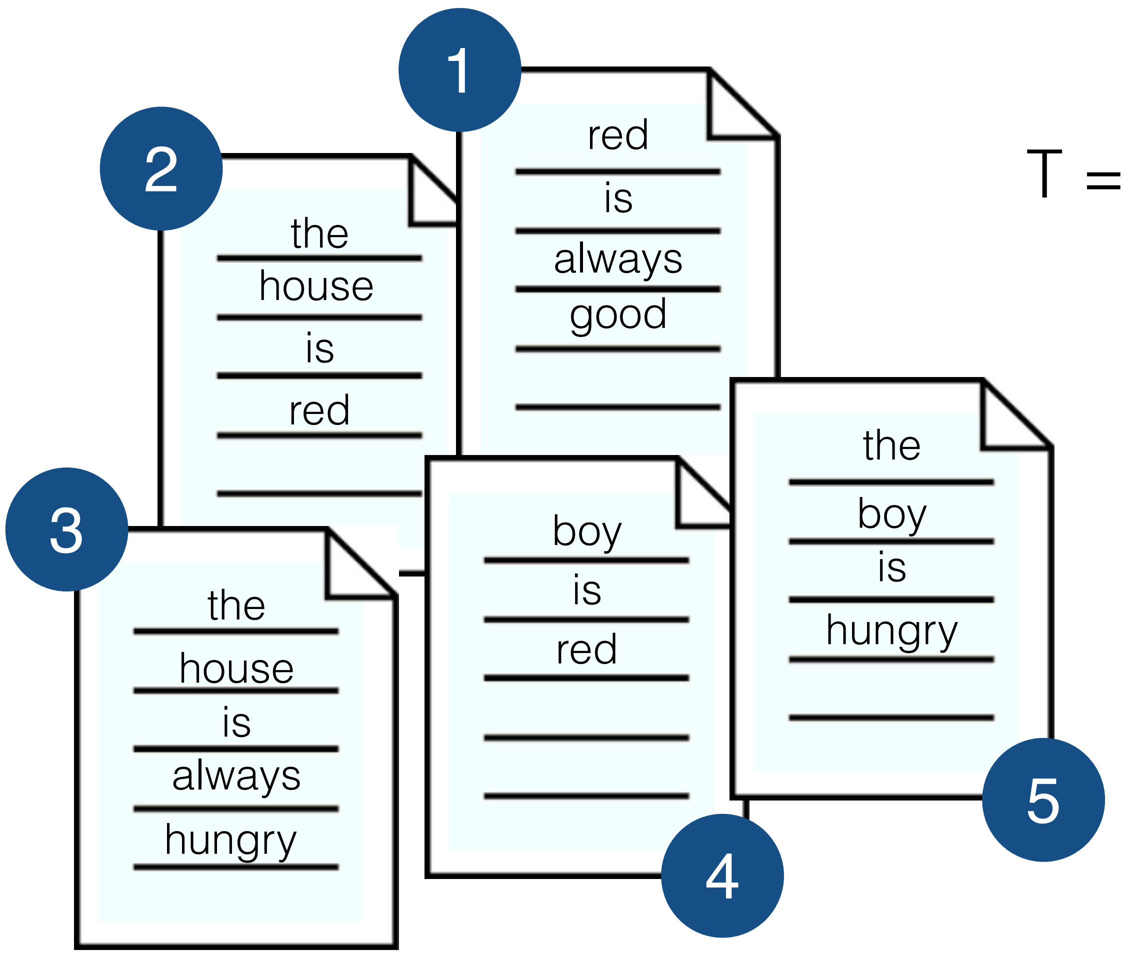
$L_{t_1}=[1, 3]$
$L_{t_2}=[4, 5]$
$L_{t_3}=[1]$
$L_{t_4}=[2, 3]$
$L_{t_5}=[3, 5]$
$L_{t_6}=[1, 2, 3, 4, 5]$
$L_{t_7}=[1, 2, 4]$
$L_{t_8}=[2, 3, 5]$

$q = \{\text{boy, is, the}\}$

$q = \{\text{good, hungry}\}$

Inverted Indexes

Inverted Indexes owe their popularity to the *efficient resolution of queries*, such as: "return me all documents in which terms $\{t_1,\ldots,t_k\}$ occur".



$$T = \{\overset{t_1}{\text{always}}, \overset{t_2}{\text{boy}}, \overset{t_3}{\text{good}}, \overset{t_4}{\text{house}}, \overset{t_5}{\text{hungry}}, \overset{t_6}{\text{is}}, \overset{t_7}{\text{red}}, \overset{t_8}{\text{the}}\}$$

$L_{t_1}=[1, 3]$
$L_{t_2}=[4, 5]$
$L_{t_3}=[1]$
$L_{t_4}=[2, 3]$
$L_{t_5}=[3, 5]$
$L_{t_6}=[1, 2, 3, 4, 5]$
$L_{t_7}=[1, 2, 4]$
$L_{t_8}=[2, 3, 5]$

$q = \{\text{boy, is, the}\}$

$q = \{\text{good, hungry}\}$

intersection

Genesis - 1970s

Peter Elias
[1923 - 2001]

Robert Fano
[1917 - 2016]

Robert Fano. *On the number of bits required to implement an associative memory*. Memorandum 61, Computer Structures Group, MIT (1971).

Peter Elias. *Efficient Storage and Retrieval by Content and Address of Static Files*. Journal of the ACM (JACM) 21, 2, 246–260 (1974).

Genesis - 1970s

Peter Elias
[1923 - 2001]

Robert Fano
[1917 - 2016]

Robert Fano. *On the number of bits required to implement an associative memory*. Memorandum 61, Computer Structures Group, MIT (1971).

Peter Elias. *Efficient Storage and Retrieval by Content and Address of Static Files*. Journal of the ACM (JACM) 21, 2, 246–260 (1974).

40 years later!

Sebastiano Vigna. *Quasi-succinct indices*.

In Proceedings of the 6-th ACM International Conference on Web Search and Data Mining (WSDM), 83-92 (2013).

Elias-Fano Encoding

3	1
4	2
7	3
13	4
14	5
15	6
21	7
43	8

Elias-Fano Encoding

	3	1
	4	2
	7	3
	13	4
	14	5
	15	6
	21	7
u =	43	8

Elias-Fano Encoding

0 0 0 0 1 1	3	1
0 0 0 1 0 0	4	2
0 0 0 1 1 1	7	3
0 0 1 1 0 1	13	4
0 0 1 1 1 0	14	5
0 0 1 1 1 1	15	6
0 1 0 1 0 1	21	7
1 0 1 0 1 1	u = 43	8

Elias-Fano Encoding

low

$r = \lceil \log(u/n) \rceil$

0 0 0 0 1 1	3	1
0 0 0 1 0 0	4	2
0 0 0 1 1 1	7	3
0 0 1 1 0 1	13	4
0 0 1 1 1 0	14	5
0 0 1 1 1 1	15	6
0 1 0 1 0 1	21	7
1 0 1 0 1 1	u = 43	8

Elias-Fano Encoding

low

$r = \lceil \log(u/n) \rceil$

0 0 0	0 1 1	3	1
0 0 0	1 0 0	4	2
0 0 0	1 1 1	7	3
0 0 1	1 0 1	13	4
0 0 1	1 1 0	14	5
0 0 1	1 1 1	15	6
0 1 0	1 0 1	21	7
1 0 1	0 1 1	u = 43	8

L = 011100111101110111101011

Elias-Fano Encoding

high $\lceil \log u \rceil - r$	low $r = \lceil \log(u/n) \rceil$		
0 0 0	0 1 1	3	1
0 0 0	1 0 0	4	2
0 0 0	1 1 1	7	3
0 0 1	1 0 1	13	4
0 0 1	1 1 0	14	5
0 0 1	1 1 1	15	6
0 1 0	1 0 1	21	7
1 0 1	0 1 1	$u =$ 43	8

L = 011100111101110111101011

Elias-Fano Encoding

	high	low		
	$\lceil \log u \rceil - r$	$r = \lceil \log(u/n) \rceil$		
	0 0 0	0 1 1	3	1
3	0 0 0	1 0 0	4	2
	0 0 0	1 1 1	7	3
	0 0 1	1 0 1	13	4
	0 0 1	1 1 0	14	5
	0 0 1	1 1 1	15	6
	0 1 0	1 0 1	21	7
	1 0 1	0 1 1	u = 43	8

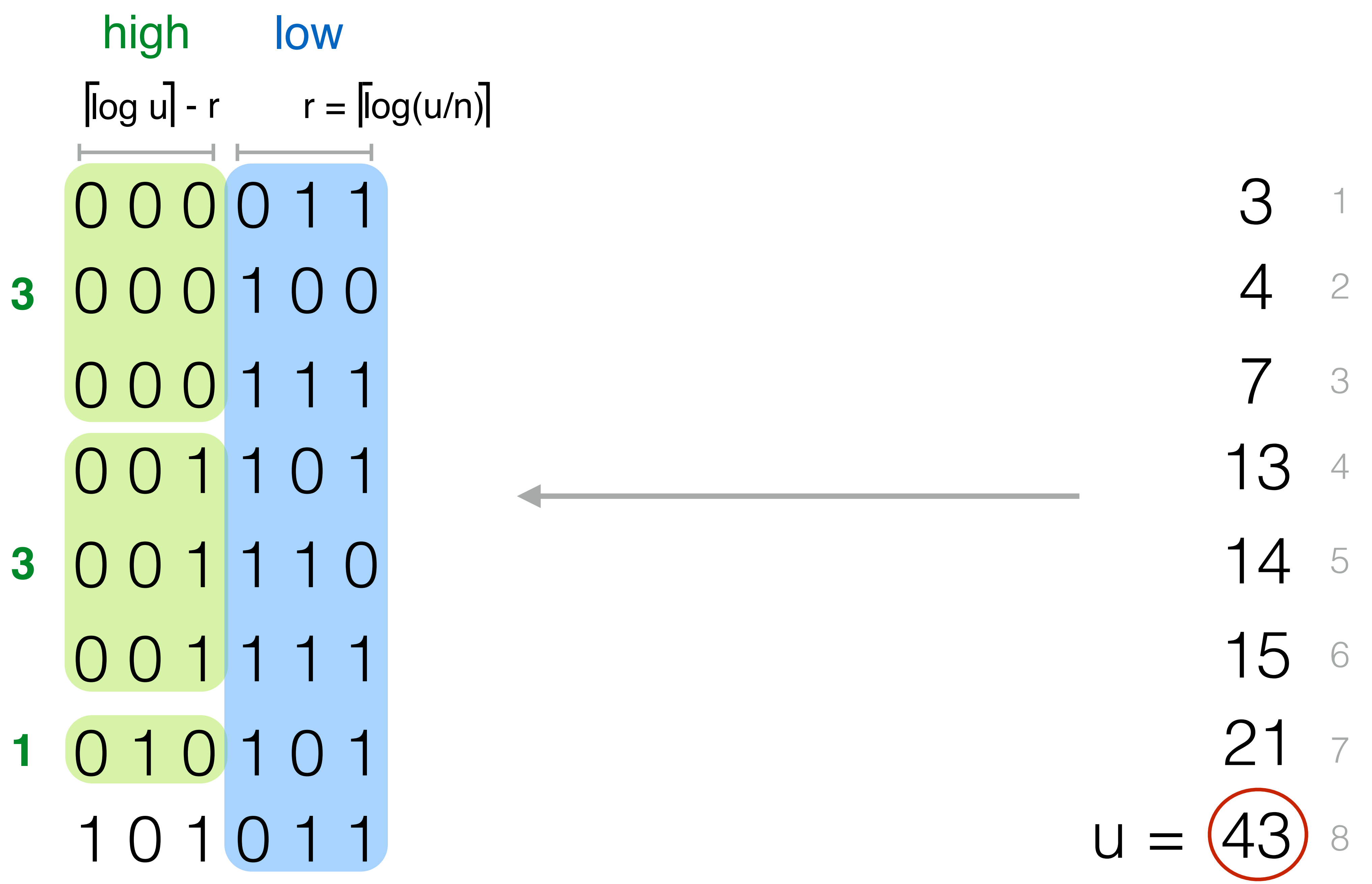
L = 011100111101110111101011

Elias-Fano Encoding

	high	low		
	$\lceil \log u \rceil - r$	$r = \lceil \log(u/n) \rceil$		
	0 0 0	0 1 1	3	1
3	0 0 0	1 0 0	4	2
	0 0 0	1 1 1	7	3
	0 0 1	1 0 1	13	4
3	0 0 1	1 1 0	14	5
	0 0 1	1 1 1	15	6
	0 1 0	1 0 1	21	7
	1 0 1	0 1 1	u = 43	8

L = 011100111101110111101011

Elias-Fano Encoding



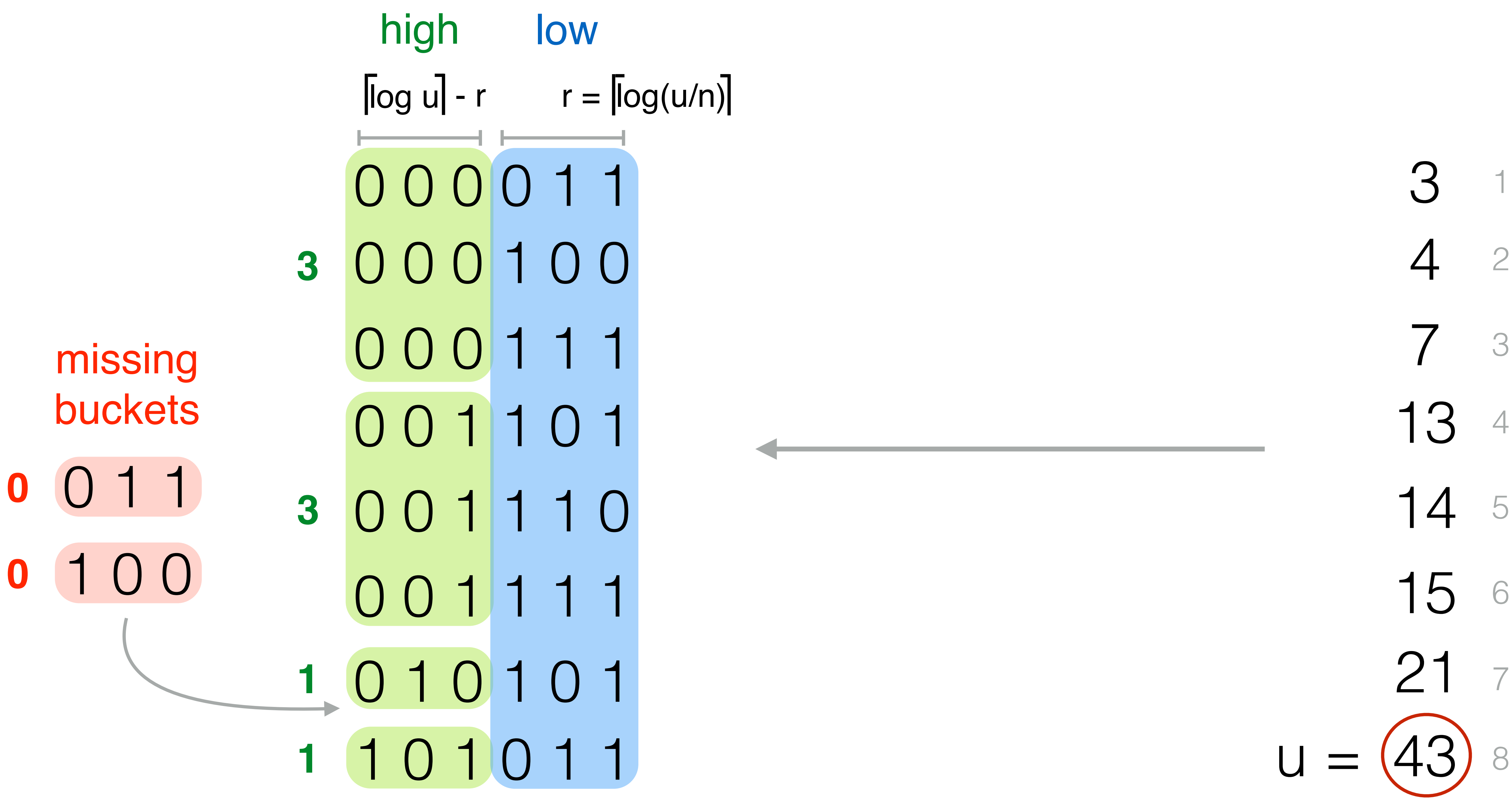
L = 011100111101110111101011

Elias-Fano Encoding

	high	low		
	$\lceil \log u \rceil - r$	$r = \lceil \log(u/n) \rceil$		
	0 0 0	0 1 1	3	1
3	0 0 0	1 0 0	4	2
	0 0 0	1 1 1	7	3
	0 0 1	1 0 1	13	4
3	0 0 1	1 1 0	14	5
	0 0 1	1 1 1	15	6
1	0 1 0	1 0 1	21	7
1	1 0 1	0 1 1	u = 43	8

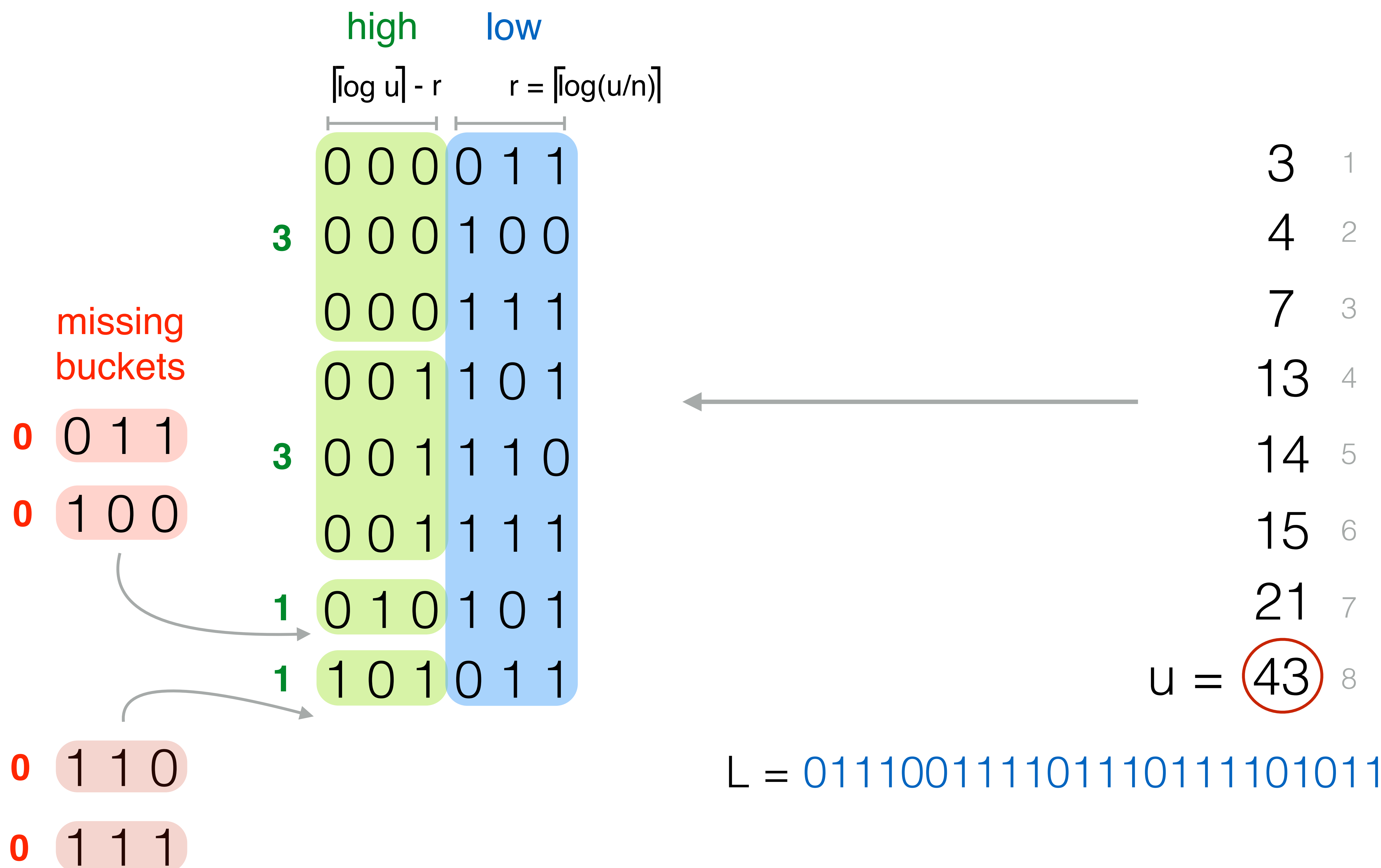
L = 011100111101110111101011

Elias-Fano Encoding

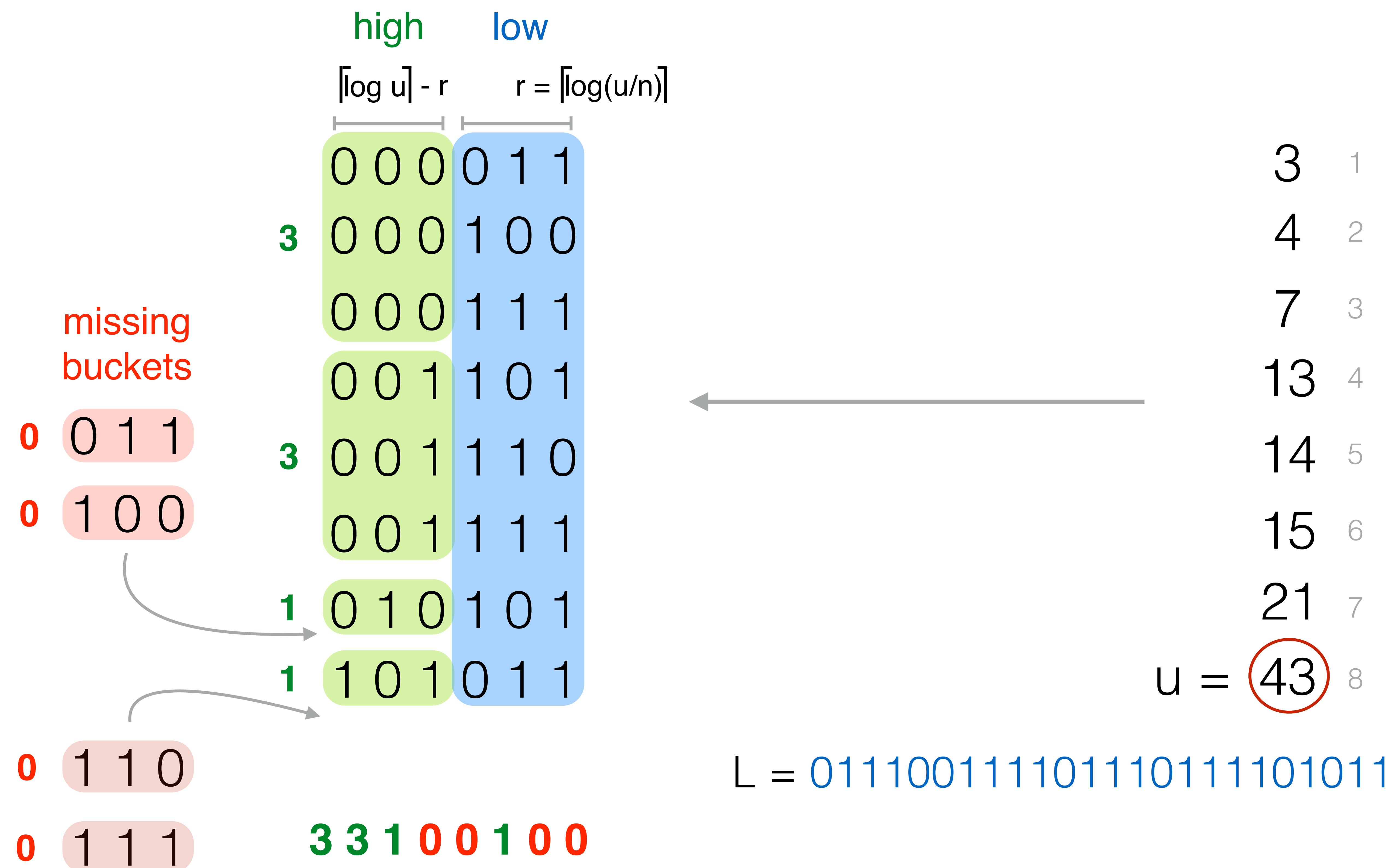


L = 011100111101110111101011

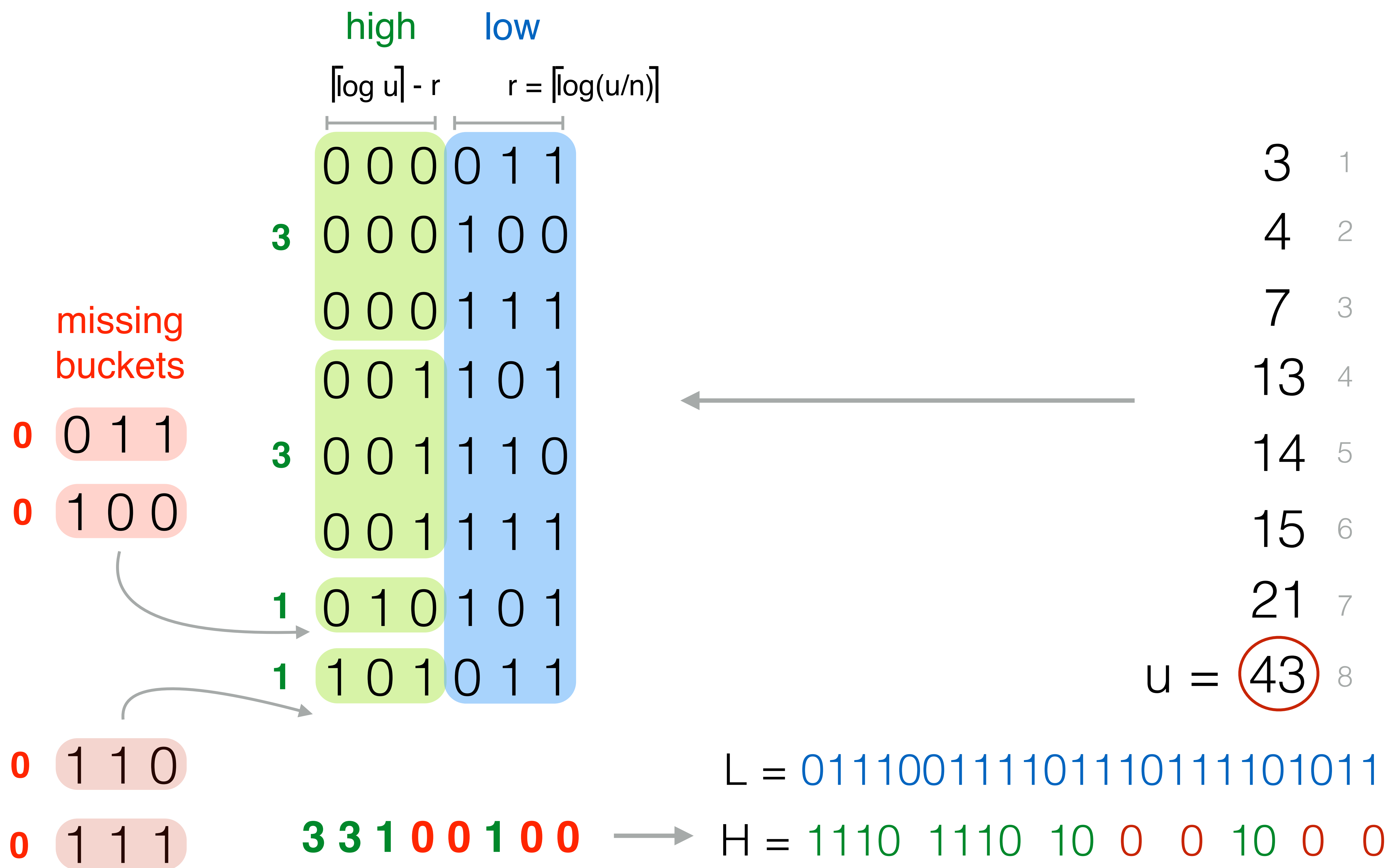
Elias-Fano Encoding



Elias-Fano Encoding



Elias-Fano Encoding



Properties - Space

$EF(S[0,n)) = ?$

Properties - Space

$$EF(S[0,n)) = ?$$

$\lceil \log(u/n) \rceil$

L = 0111001111011101111101011

H = 1110 1110 10 0 0 10 0 0

Properties - Space

$$EF(S[0,n)) = n\left\lceil \log\frac{u}{n} \right\rceil$$

$\lceil \log(u/n) \rceil$

L = 01110011110111011111101011

H = 1110 1110 10 0 0 10 0 0

Properties - Space

$$EF(S[0,n)) = n\left\lceil \log \frac{u}{n} \right\rceil$$

$\lceil \log(u/n) \rceil$

L = 01110011110111011111101011

H = 1110 1110 10 0 0 10 0 0

n ones

$$EF(S[0,n)) = n\left\lceil \log \frac{u}{n} \right\rceil$$

$\lceil \log(u/n) \rceil$

L = 01110011110111011110101 1

H = 1110 1110 10 0 0 10 0 0

n ones

We store a 0 whenever we change bucket.

Properties - Space

$$EF(S[0,n)) = n\left\lceil \log \frac{u}{n} \right\rceil$$

$\lceil \log(u/n) \rceil$

L = 0111001111011101111101011

H = 1110 1110 10 0 0 10 0 0

We store a 0 whenever we change bucket.

n ones

$2^{\lfloor \log n \rfloor}$ zeros

Properties - Space

1

$$EF(S[0,n)) = n\left\lceil \log \frac{u}{n} \right\rceil + 2n \text{ bits}$$

$\lceil \log(u/n) \rceil$

L = 0111001111011101111101011

H = 1110 1110 10 0 0 10 0 0

We store a 0 whenever we change bucket.

n ones

$2^{\lfloor \log n \rfloor}$ zeros

Properties - Space

$$EF(S[0,n)) = n\left\lceil \log\frac{u}{n} \right\rceil + 2n \text{ bits}$$

$\lceil \log(u/n) \rceil$

L = 01110011110111011111101011

H = 1110 1110 10 0 0 10 0 0

n ones

$2^{\lfloor \log n \rfloor}$ zeros

Properties - Space

$$EF(S[0,n)) = n\left\lceil \log \frac{u}{n} \right\rceil + 2n \text{ bits}$$

$\lceil \log(u/n) \rceil$

L = 011100111011101111101011

H = 1110 1110 10 0 0 10 0 0

n ones

Properties - Space

$$EF(S[0,n)) = n\left\lceil \log\frac{u}{n} \right\rceil + 2n \text{ bits}$$

$\lceil \log(u/n) \rceil$

L = 01110011110111011110101 1

H = 1110 1110 10 0 0 10 0 0

Properties - Space

$$EF(S[0,n)) = n\left\lceil \log \frac{u}{n} \right\rceil + 2n \text{ bits}$$

Properties - Space

$$EF(S[0,n)) = n\left\lceil \log \frac{u}{n} \right\rceil + 2n \text{ bits}$$

Is it good or not?

Properties - Space

$$EF(S[0,n)) = n\left\lceil \log\frac{u}{n}\right\rceil + 2n \text{ bits}$$

Is it good or not?

Information Theoretic Lower Bound

The minimum number of bits needed to describe a set $\mathcal{X}$ is $\lceil \log|\mathcal{X}| \rceil$ bits.

Properties - Space

$$EF(S[0,n)) = n\left\lceil \log \frac{u}{n} \right\rceil + 2n \text{ bits}$$

Is it good or not?

Information Theoretic Lower Bound

The minimum number of bits needed to describe a set $\mathcal{X}$ is

$$\left\lceil \log |\mathcal{X}| \right\rceil \text{ bits.}$$

$\mathcal{X}$ is the set of all monotone sequence of length n drawn from a universe u.

$$|\mathcal{X}|\ ?$$

$$EF(S[0,n)) = n\left\lceil \log \frac{u}{n} \right\rceil + 2n \text{ bits}$$

Is it good or not?

Information Theoretic Lower Bound

The minimum number of bits needed to describe a set $\mathcal{X}$ is

$\left\lceil \log |\mathcal{X}| \right\rceil$ bits.

$\mathcal{X}$ is the set of all monotone sequence of length n drawn from a universe u.

$|\mathcal{X}|$?

0000000000000000000

$$EF(S[0,n)) = n\left\lceil \log \frac{u}{n} \right\rceil + 2n \text{ bits}$$

Is it good or not?

Information Theoretic Lower Bound

The minimum number of bits needed to describe a set $\mathcal{X}$ is $\lceil \log |\mathcal{X}| \rceil$ bits.

$\mathcal{X}$ is the set of all monotone sequence of length n drawn from a universe u.

$|\mathcal{X}|$?

000**1**000000000000000

3

$$EF(S[0,n)) = n\left\lceil \log \frac{u}{n} \right\rceil + 2n \text{ bits}$$

Is it good or not?

Information Theoretic Lower Bound

The minimum number of bits needed to describe a set $\mathcal{X}$ is $\left\lceil \log |\mathcal{X}| \right\rceil$ bits.

$\mathcal{X}$ is the set of all monotone sequence of length n drawn from a universe u.

$|\mathcal{X}|$?

000**1**00**1**000000000000

3 6

Properties - Space

$$EF(S[0,n)) = n\left\lceil \log\frac{u}{n} \right\rceil + 2n \text{ bits}$$

Is it good or not?

Information Theoretic Lower Bound

The minimum number of bits needed to describe a set $\mathcal{X}$ is $\lceil \log|\mathcal{X}| \rceil$ bits.

$\mathcal{X}$ is the set of all monotone sequence of length n drawn from a universe u.

$|\mathcal{X}|$?

000**1**00**1**000**1**0000000

3 6 10

Properties - Space

1

$$EF(S[0,n)) = n\left\lceil \log\frac{u}{n}\right\rceil + 2n \text{ bits}$$

Is it good or not?

Information Theoretic Lower Bound

The minimum number of bits needed to describe a set $\mathcal{X}$ is $\left\lceil \log|\mathcal{X}| \right\rceil$ bits.

$\mathcal{X}$ is the set of all monotone sequence of length n drawn from a universe u.

$|\mathcal{X}|$?

000**1**00**1**000**11**000000

3 6 10 11

Properties - Space

$$EF(S[0,n)) = n\left\lceil \log \frac{u}{n} \right\rceil + 2n \ \text{ bits}$$

Is it good or not?

Information Theoretic Lower Bound

The minimum number of bits needed to describe a set $\mathcal{X}$ is $\lceil \log |\mathcal{X}| \rceil$ bits.

$\mathcal{X}$ is the set of all monotone sequence of length n drawn from a universe u.

$|\mathcal{X}|$?

000**1**00**1**000**11**00000**1**

3 6 10 11 17

Properties - Space

1

$$EF(S[0,n)) = n\left\lceil \log \frac{u}{n} \right\rceil + 2n \text{ bits}$$

Is it good or not?

Information Theoretic Lower Bound

The minimum number of bits needed to describe a set $\mathcal{X}$ is $\lceil \log|\mathcal{X}| \rceil$ bits.

$\mathcal{X}$ is the set of all monotone sequence of length n drawn from a universe u.

$|\mathcal{X}|$?

000**1**00**1**000**11**00000**1**

3 6 10 11 17

With possible repetitions!
(*weak* monotonicity)

Properties - Space

$$EF(S[0,n)) = n\left\lceil \log \frac{u}{n} \right\rceil + 2n \text{ bits}$$

Is it good or not?

Information Theoretic Lower Bound

The minimum number of bits needed to describe a set $\mathcal{X}$ is $\lceil \log|\mathcal{X}| \rceil$ bits.

$\mathcal{X}$ is the set of all monotone sequence of length n drawn from a universe u.

$$|\mathcal{X}| = \binom{u+n}{n}$$

0001001000110000001

3 6 10 11 17

With possible repetitions!
(*weak* monotonicity)

Properties - Space

$$EF(S[0,n)) = n\left\lceil \log \frac{u}{n} \right\rceil + 2n \ \text{ bits}$$

Is it good or not?

Information Theoretic Lower Bound

The minimum number of bits needed to describe a set $\mathcal{X}$ is $\lceil \log |\mathcal{X}| \rceil$ bits.

$\mathcal{X}$ is the set of all monotone sequence of length n drawn from a universe u.

$$|\mathcal{X}| = \binom{u+n}{n}$$

$$\left\lceil \log \binom{u+n}{n} \right\rceil \approx n \log \frac{u+n}{n}$$

000**1**00**1**000**11**00000**1**

3 6 10 11 17

With possible repetitions!
(*weak* monotonicity)

Properties - Space

$$EF(S[0,n)) = \boxed{n\left\lceil \log\frac{u}{n}\right\rceil + 2n \text{ bits}}$$

Is it good or not?

(less than half a bit away [Elias-1974])

Information Theoretic Lower Bound

The minimum number of bits needed to describe a set $\mathcal{X}$ is $\lceil \log|\mathcal{X}| \rceil$ bits.

$\mathcal{X}$ is the set of all monotone sequence of length n drawn from a universe u.

$$|\mathcal{X}| = \binom{u+n}{n}$$

$$\boxed{\left\lceil \log\binom{u+n}{n} \right\rceil \approx n \log\frac{u+n}{n}}$$

optimal

000**1**00**1**000**11**00000**1**

3 6 10 11 17

With possible repetitions!
(*weak* monotonicity)

Properties - Operations

Properties - Operations

`Access` to each S[i] in O(1) worst-case

Properties - Operations

`Access` to each S[i] in O(1) worst-case

`Predecessor(x)` = $\max\{S[i] \mid S[i] < x\}$

`Successor(x)` = $\min\{S[i] \mid S[i] \geq x\}$

queries in $O\left(\log \frac{u}{n}\right)$ worst-case

`Access` to each S[i] in O(1) worst-case

$$\texttt{Predecessor(x)} = \max\{S[i] \mid S[i] < \mathtt{x}\}$$

$$\texttt{Successor(x)} = \min\{S[i] \mid S[i] \geq \mathtt{x}\}$$

queries in $O\left(\log \frac{u}{n}\right)$ worst-case

`Rank/Select` on bitmaps

Definition

Given a bitvector B of n bits:

$\mathtt{Rank}_{0/1}(\mathtt{i})$ = # of 0/1 in B[0,$\mathtt{i}$)

$\mathtt{Select}_{0/1}(\mathtt{i})$ = position of $\mathtt{i}$-th 0/1

`Rank/Select` on bitmaps

Definition

Given a bitvector B of n bits:

$\mathtt{Rank}_{0/1}(\mathtt{i})$ = # of 0/1 in B[0,$\mathtt{i}$)

$\mathtt{Select}_{0/1}(\mathtt{i})$ = position of $\mathtt{i}$-th 0/1

Examples

B = 1010110101011110101110101

`Rank/Select` on bitmaps

Definition

Given a bitvector B of n bits:

$\texttt{Rank}_{0/1}(\texttt{i})$ = # of 0/1 in B[0,$\texttt{i}$)

$\texttt{Select}_{0/1}(\texttt{i})$ = position of $\texttt{i}$-th 0/1

Examples

B = 1010110101011110101 10101

$\texttt{Rank}_0(5) = 2$

Rank/Select on bitmaps

Definition

Given a bitvector B of n bits:

$\mathtt{Rank}_{0/1}(\mathtt{i})$ = # of 0/1 in B[0,$\mathtt{i}$)

$\mathtt{Select}_{0/1}(\mathtt{i})$ = position of $\mathtt{i}$-th 0/1

Examples

B = 1010110101011110101110101

$\mathtt{Rank}_{0}(5) = 2$

$\mathtt{Rank}_{1}(7) = 4$

Rank/Select on bitmaps

Definition

Given a bitvector B of n bits:

$\mathtt{Rank}_{0/1}(\mathtt{i})$ = # of 0/1 in B[0,$\mathtt{i}$)

$\mathtt{Select}_{0/1}(\mathtt{i})$ = position of $\mathtt{i}$-th 0/1

Examples

B = 1010110101011110101101 01

$\mathtt{Rank}_0(5) = 2$ $\mathtt{Select}_0(5) = 10$

$\mathtt{Rank}_1(7) = 4$

`Rank/Select` on bitmaps

Definition

Given a bitvector B of n bits:

$\texttt{Rank}_{0/1}(\texttt{i})$ = # of 0/1 in B[0,$\texttt{i}$)

$\texttt{Select}_{0/1}(\texttt{i})$ = position of $\texttt{i}$-th 0/1

Examples

B = 1010110101011110101101

$\texttt{Rank}_0(5) = 2$ $\texttt{Select}_0(5) = 10$

$\texttt{Rank}_1(7) = 4$ $\texttt{Select}_1(7) = 11$

Random Access

$S = [3, 4, 7, 13, 14, 15, 21, 43]$

1 2 3 4 5 6 7 8

Random `Access`

S = [3, 4, 7, 13, 14, 15, 21, 43]

1 2 3 4 5 6 7 8

`Access(4)` = S[4] = ?

Random Access

$S = [3, 4, 7, 13, 14, 15, 21, 43]$

1 2 3 4 5 6 7 8

Access(4) $= S[4] = ?$

$H = 1110111010001000$

$L = \underbrace{011}_{r = \lceil \log(u/n) \rceil}100111011101111101011$

Random Access

$S = [3, 4, 7, 13, 14, 15, 21, 43]$

1 2 3 4 5 6 7 8

Access(4) = S[4] = ?

Recall: we store a 0 whenever we change bucket.

H = 1110111010001000

L = 011100111101110111101011

$r = \lceil \log(u/n) \rceil$

Random Access

S = [3, 4, 7, 13, 14, 15, 21, 43]

1 2 3 4 5 6 7 8

Access(4) = S[4] = ?

Recall: we store a 0 whenever we change bucket.

H = 11101 11010001000

L = 011100111101110111101011

$r = \lceil \log(u/n) \rceil$

Random Access

$S = [3, 4, 7, 13, 14, 15, 21, 43]$

1 2 3 4 5 6 7 8

Access(4) = S[4] = ?

Recall: we store a 0 whenever we change bucket.

H = 1110111010001000

L = 011100111101110111101011

$r = \lceil \log(u/n) \rceil$

Access(i) = Select₁(i)

Random Access

S = [3, 4, 7, 13, 14, 15, 21, 43]

1 2 3 4 5 6 7 8

Access(4) = S[4] = ?

Recall: we store a 0 whenever we change bucket.

H = 1110111010001000

L = 011100111101110111101011

$r = \lceil \log(u/n) \rceil$

$\text{Access}(i) = \text{Rank}_0(\text{Select}_1(i))$

Random Access

S = [3, 4, 7, 13, 14, 15, 21, 43]

1 2 3 4 5 6 7 8

Access(4) = S[4] = 001000

Recall: we store a 0 whenever we change bucket.

H = 11101110100010000

L = 011100111011101111010111

$r = \lceil \log(u/n) \rceil$

$\mathrm{Access}(i) = \mathrm{Rank}_0(\mathrm{Select}_1(i))$

Random Access

S = [3, 4, 7, 13, 14, 15, 21, 43]

1 2 3 4 5 6 7 8

Access(4) = S[4] = 001000

Recall: we store a 0 whenever we change bucket.

H = 11101 11010001000

L = 011100111101110111101011

$r = \lceil \log(u/n) \rceil$

$$\text{Access}(i) = \text{Rank}_0(\text{Select}_1(i)) = \text{Select}_1(i) - i$$

Random Access

$S = [3, 4, 7, 13, 14, 15, 21, 43]$

1 2 3 4 5 6 7 8

$\texttt{Access(4)} = S[4] = 001000$

Recall: we store a 0 whenever we change bucket.

H = 11101 11010001000

L = 011100111101110111101011

$r = \lceil \log(u/n) \rceil$

$\texttt{Access(i)} = \texttt{Select}_1\texttt{(i)} - \texttt{i}$

Random Access

S = [3, 4, 7, 13, 14, 15, 21, 43]

1 2 3 4 5 6 7 8

Access(4) = S[4] = 001101

Recall: we store a 0 whenever we change bucket.

H = 1110111010001000

L = 011100111101110111101011

$r = \lceil \log(u/n) \rceil$

```
Access(i) = Select₁(i) - i << r | L[(i-1)r,ir)
```

Random Access

S = [3, 4, 7, 13, 14, 15, 21, 43]

1 2 3 4 5 6 7 8

Access(4) = S[4] = 001 101

Recall: we store a 0 whenever we change bucket.

H = 1110111010001000

L = 011100111101110111101011

$r = \lceil \log(u/n) \rceil$

$\texttt{Access(i)} = \texttt{Select}_1\texttt{(i) - i << r | L[(i-1)r,ir)}$

Random Access

S = [3, 4, 7, 13, 14, 15, 21, 43]

1 2 3 4 5 6 7 8

Access(4) = S[4] = 001101

Access(7) = S[7] = ?

Recall: we store a 0 whenever we change bucket.

H = 1110111010001000

L = 011100111101110111101011

$r = \lceil \log(u/n) \rceil$

`Access(i) = Select`$_1$`(i) - i << r | L[(i-1)r,ir)`

Random Access

S = [3, 4, 7, 13, 14, 15, 21, 43]

1 2 3 4 5 6 7 8

Access(4) = S[4] = 001 101

Access(7) = S[7] = ?

Recall: we store a 0 whenever we change bucket.

H = 1110111010001000

L = 011100111101110111101011

$r = \lceil \log(u/n) \rceil$

Access(i) = Select$_1$(i) - i << r | L[(i-1)r,ir)

Random Access

$S = [3, 4, 7, 13, 14, 15, 21, 43]$

1 2 3 4 5 6 7 8

Access(4) = S[4] = 001 101

Access(7) = S[7] = 010000

Recall: we store a 0 whenever we change bucket.

H = 1110111010001000

L = 011100111101110111101011

$r = \lceil \log(u/n) \rceil$

```
Access(i) = Select₁(i) - i << r | L[(i-1)r,ir)
```

Random Access

S = [3, 4, 7, 13, 14, 15, 21, 43]

1 2 3 4 5 6 7 8

Access(4) = S[4] = 001 101

Access(7) = S[7] = 010 101

Recall: we store a 0 whenever we change bucket.

H = 1110111010001000

L = 011100111101110111101011

$r = \lceil \log(u/n) \rceil$

```
Access(i) = Select₁(i) - i << r | L[(i-1)r,ir)
```

Random Access

S = [3, 4, 7, 13, 14, 15, 21, 43]

1 2 3 4 5 6 7 8

Access(4) = S[4] = 001 101

Access(7) = S[7] = 010 101

Recall: we store a 0 whenever we change bucket.

H = 1110111010001000

L = 011100111101110111101011

$r = \lceil \log(u/n) \rceil$

Complexity: O(1)

Access(i) = Select$_1$(i) - i << r | L[(i-1)r,ir)

Available Implementations

Library	Author(s)	Link	Language
folly	Facebook, Inc.	https://github.com/facebook/folly	C++
sdsl	Simon Gog	https://github.com/simongog/sdsl-lite	C++
ds2i	Giuseppe Ottaviano Rossano Venturini Nicola Tonellotto	https://github.com/ot/ds2i	C++
Sux	Sebastiano Vigna	http://sux.di.unimi.it	Java/C++

Killer applications

1. Inverted Indexes

Killer applications

1. Inverted Indexes

2. Social Networks

Killer applications

1. Inverted Indexes

2. Social Networks

Unicorn: A System for Searching the Social Graph

Michael Curtiss, Iain Becker, Tudor Bosman, Sergey Doroshenko,
Lucian Grijincu, Tom Jackson, Sandhya Kunnatur, Soren Lassen, Philip Pronin,
Sriram Sankar, Guanghao Shen, Gintaras Woss, Chao Yang, Ning Zhang

Facebook, Inc.

ABSTRACT

Unicorn is an online, in-memory social graph-aware indexing system designed to search trillions of edges between tens of billions of users and entities on thousands of commodity servers. Unicorn is based on standard concepts in informa-

rative of the evolution of Unicorn's architecture, as well as documentation for the major features and components of the system.

To the best of our knowledge, no other online graph retrieval system has ever been built with the scale of Unicorn

Killer applications

1. Inverted Indexes

2. Social Networks

Unicorn: A System for Searching the Social Graph

Michael Curtiss, Iain Becker, Tudor Bosman, Sergey Doroshenko,
Lucian Grijincu, Tom Jackson, Sandhya Kunnatur, Soren Lassen, Philip Pronin,
Sriram Sankar, Guanghao Shen, Gintaras Woss, Chao Yang, Ning Zhang

Facebook, Inc.

ABSTRACT

Unicorn is an online, in-memory social graph-aware indexing system designed to search trillions of edges between tens of billions of users and entities on thousands of commodity servers. Unicorn is based on standard concepts in informa-

rative of the evolution of Unicorn's architecture, as well as documentation for the major features and components of the system.

To the best of our knowledge, no other online graph retrieval system has ever been built with the scale of Unicorn

Killer applications

1. Inverted Indexes

2. Social Networks

Unicorn: A System for Searching the Social Graph

Michael Curtiss, Iain Becker, Tudor Bosman, Sergey Doroshenko,
Lucian Grijincu, Tom Jackson, Sandhya Kunnatur, Soren Lassen, Philip Pronin,
Sriram Sankar, Guanghao Shen, Gintaras Woss, Chao Yang, Ning Zhang
Facebook, Inc.

ABSTRACT

Unicorn is an online, in-memory social graph-aware indexing system designed to search trillions of edges between tens of billions of users and entities on thousands of commodity servers. Unicorn is based on standard concepts in informa-

Open Source

All Unicorn index server and aggregator code is written in C++. Unicorn relies extensively on modules in Facebook's "Folly" Open Source Library [5]. As part of the effort of releasing Graph Search, we have open-sourced a C++ implementation of the Elias-Fano index representation [31] as part of Folly.

Killer applications

1. Inverted Indexes

2. Social Networks

3. Compressed Tries for N-Grams

N-grams - Introduction

Strings of *N* words.

N typically ranges from 1 to 5.

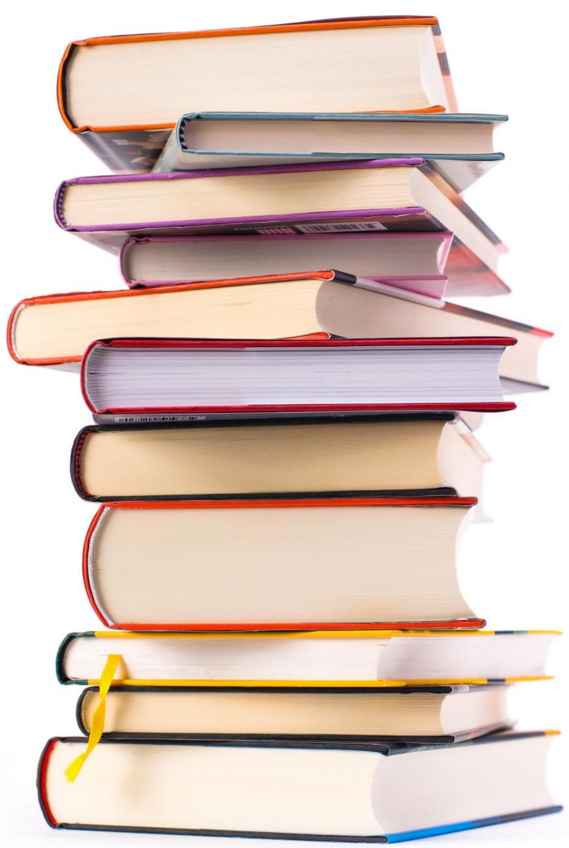
Extracted from text using a *sliding window* approach.

N-grams - Introduction

Strings of *N* words.

N typically ranges from 1 to 5.

Extracted from text using a *sliding window* approach.

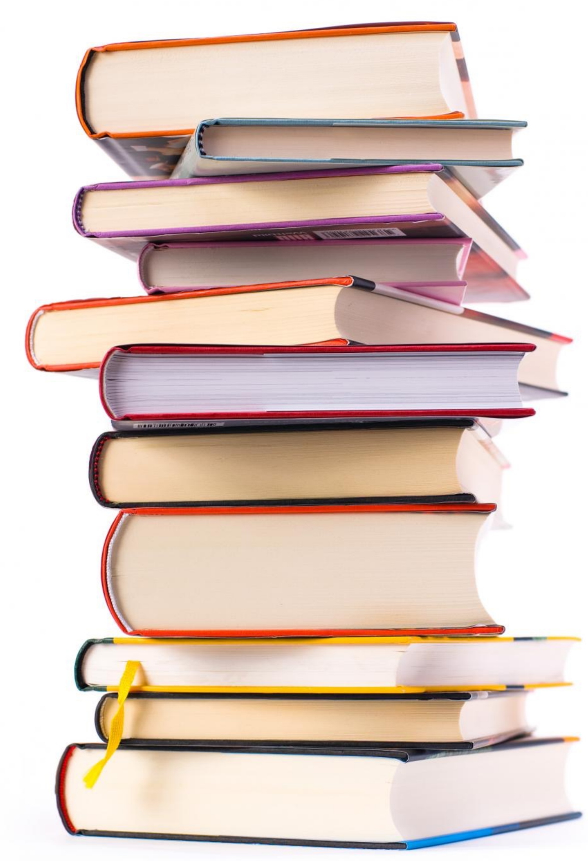


N-grams - Introduction

Strings of *N* words.

N typically ranges from 1 to 5.

Extracted from text using a *sliding window* approach.



Google Books

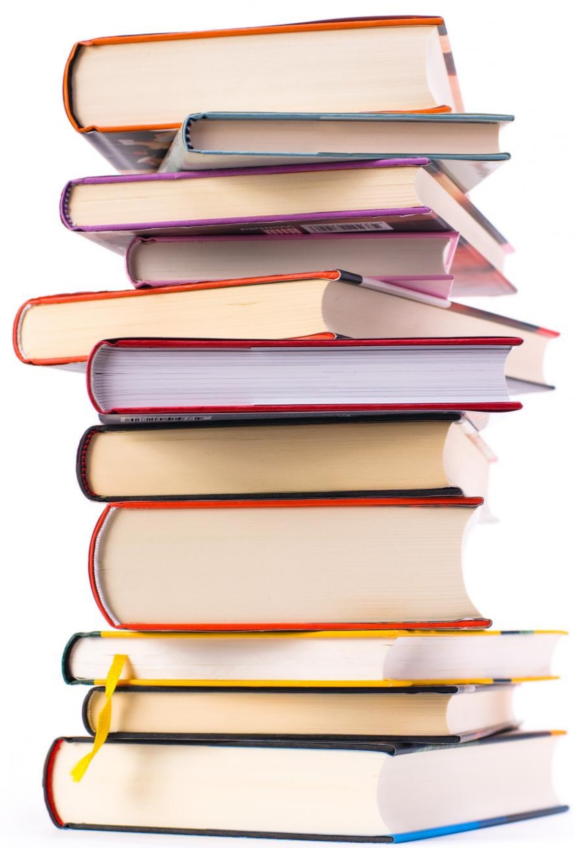
≈ 6% of the books ever published

N-grams - Introduction

Strings of *N* words.

N typically ranges from 1 to 5.

Extracted from text using a *sliding window* approach.



Google Books

≈ 6% of the books ever published

N	number of grams
1	24,359,473
2	667,284,771
3	7,397,041,901
4	1,644,807,896
5	1,415,355,596

More than 11 billion grams.

N-grams - Challenge

Store massive *N*-grams datasets in **compressed space** such that given a pattern, we can **return its value efficiently**.

N-grams - Challenge

Store massive *N*-grams datasets in **compressed space** such that given a pattern, we can **return its value efficiently**.

N-Gram **values**

N-grams - Challenge

Store massive *N*-grams datasets in **compressed space** such that given a pattern, we can **return its value efficiently**.

frequency count
(integer)

N-Gram **values**

N-grams - Challenge

Store massive *N*-grams datasets in **compressed space** such that given a pattern, we can **return its value efficiently**.

N-Gram **values** → frequency count (integer)

N-Gram **values** → probability weight (floating point)

N-grams - Challenge

Store massive *N*-grams datasets in **compressed space** such that given a pattern, we can **return its value efficiently**.

N-Gram **values**

→ frequency count (integer)

→ probability weight (floating point)

Efficient **map**

N-grams - Challenge

Store massive *N*-grams datasets in **compressed space** such that given a pattern, we can **return its value efficiently**.

N-Gram **values** → frequency count (integer)
→ probability weight (floating point)

Efficient **map** → **hash**
+ time
- space

N-grams - Challenge

Store massive *N*-grams datasets in **compressed space** such that given a pattern, we can **return its value efficiently**.

N-Gram **values**

frequency count (integer)

probability weight (floating point)

Efficient **map**

hash
+ time
- space

trie
+ space
- time

N-grams - Challenge

Store massive *N*-grams datasets in **compressed space** such that given a pattern, we can **return its value efficiently**.

N-Gram **values** → frequency count (integer)

N-Gram **values** → probability weight (floating point)

Efficient **map** → **hash**: + time, - space

Efficient **map** → **trie**: + space, - time

Active field of research
Many software libraries

- **KenLM** [Heafield, WMT 2011]
- **BerkeleyLM** [Pauls and Klein, ACL 2011]
- **ExpGram** [Watanabe at el., IJCNLP 2009]
- **IRSTLM** [Federico et al., ACL 2008]
- **RandLM** [Talbot and Osborne, ACL 2007]
- **SRILM** [Stolcke, INTERSPEECH 2002]

Trie Indexing

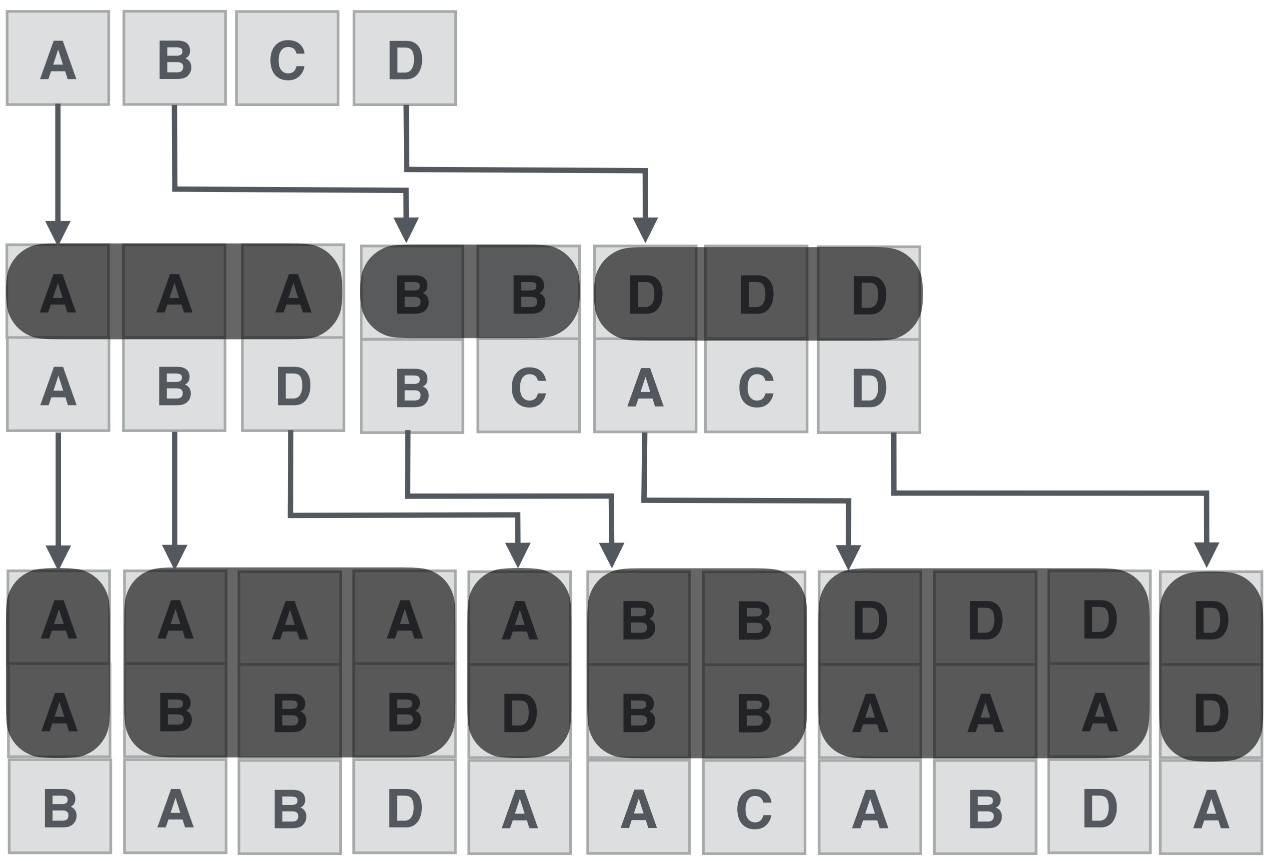
Trie Indexing

A	B	C	D

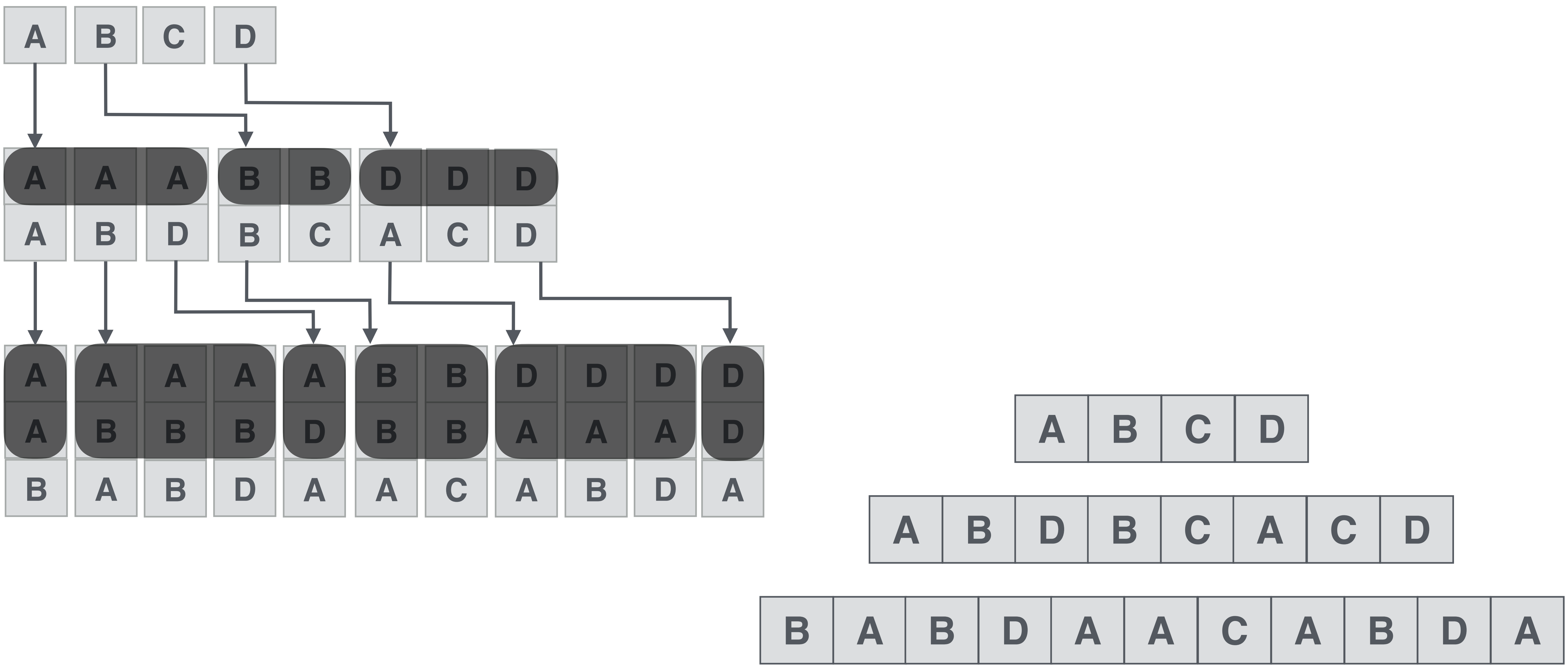
A	A	A	B	B	D	D	D
A	B	D	B	C	A	C	D

A	A	A	A	A	B	B	D	D	D	D
A	B	B	B	D	B	B	A	A	A	D
B	A	B	D	A	A	C	A	B	D	A

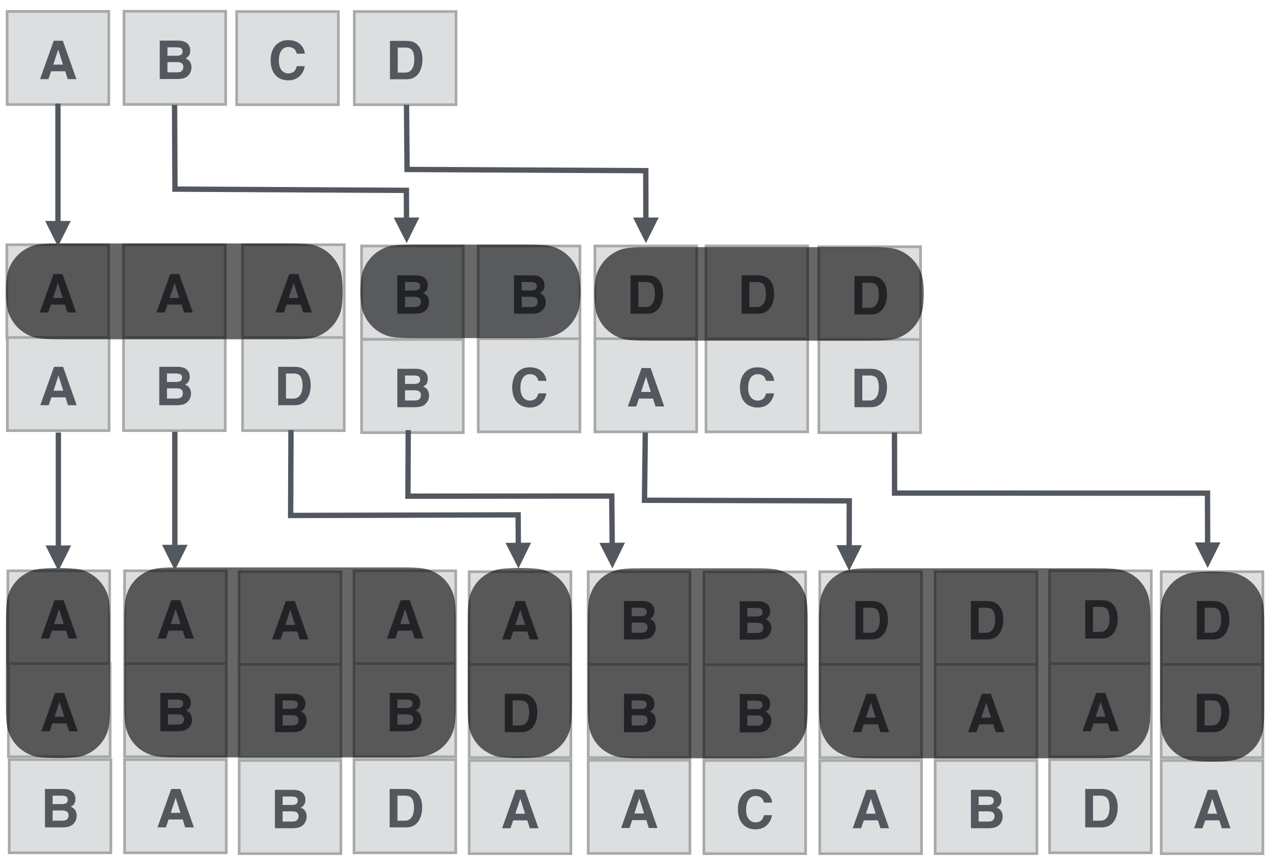
Trie Indexing

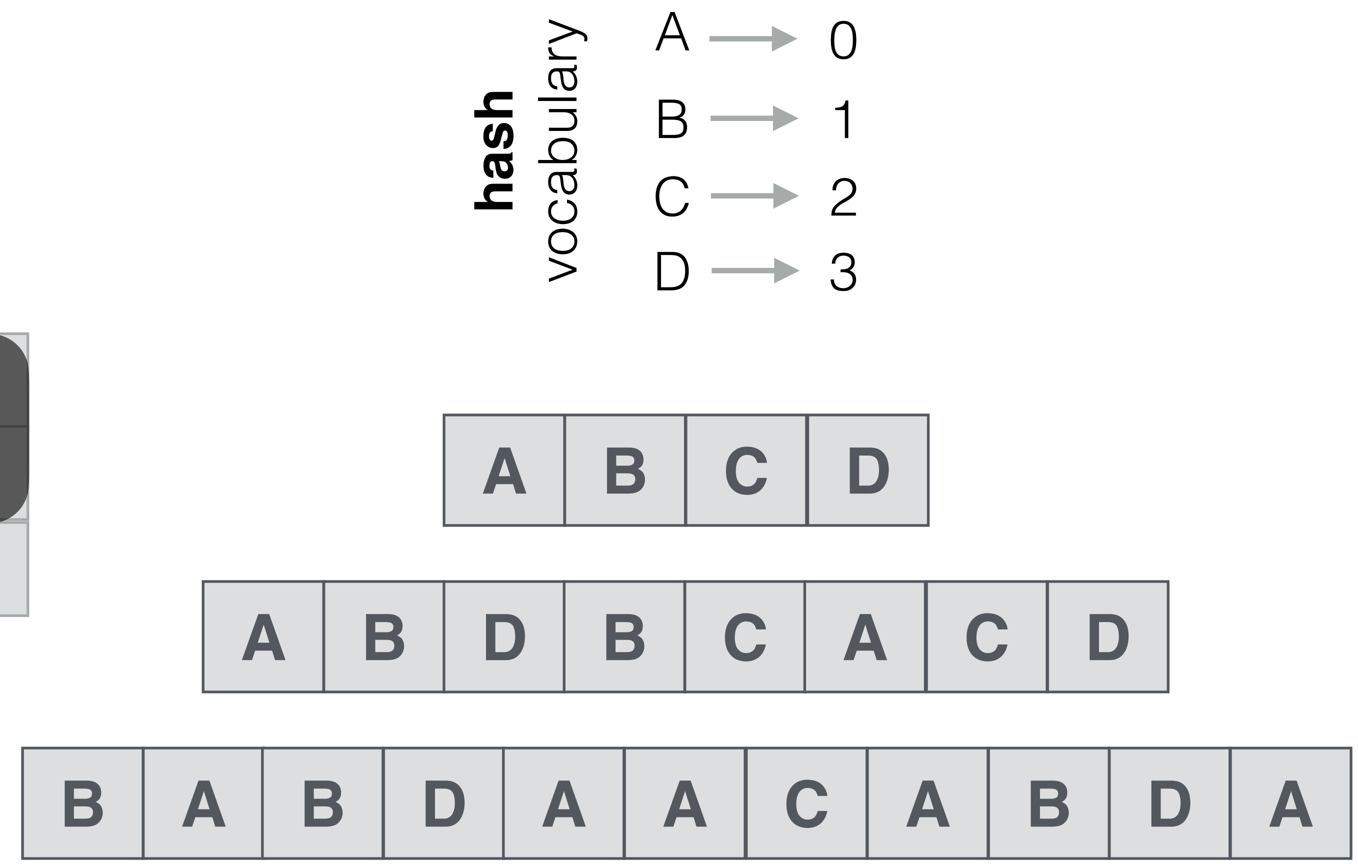


Trie Indexing

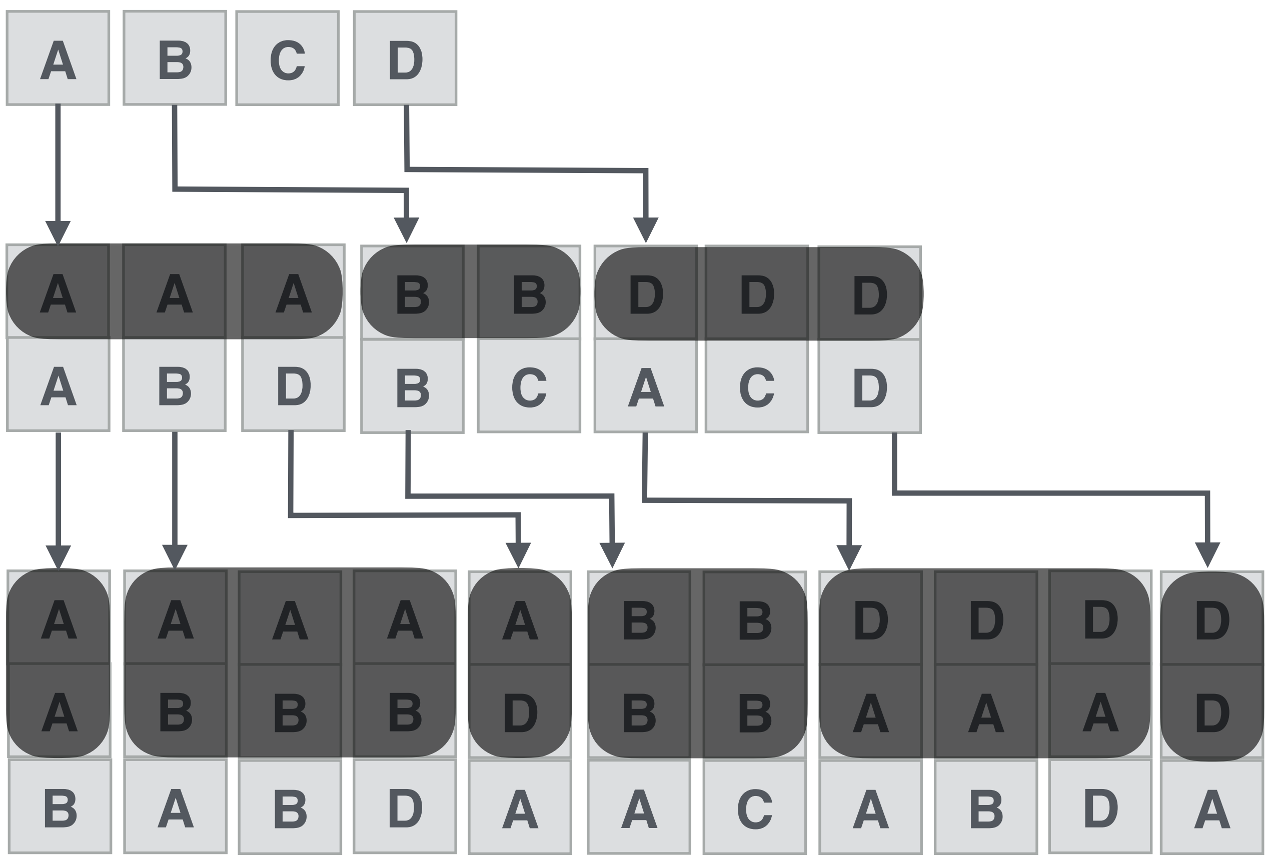


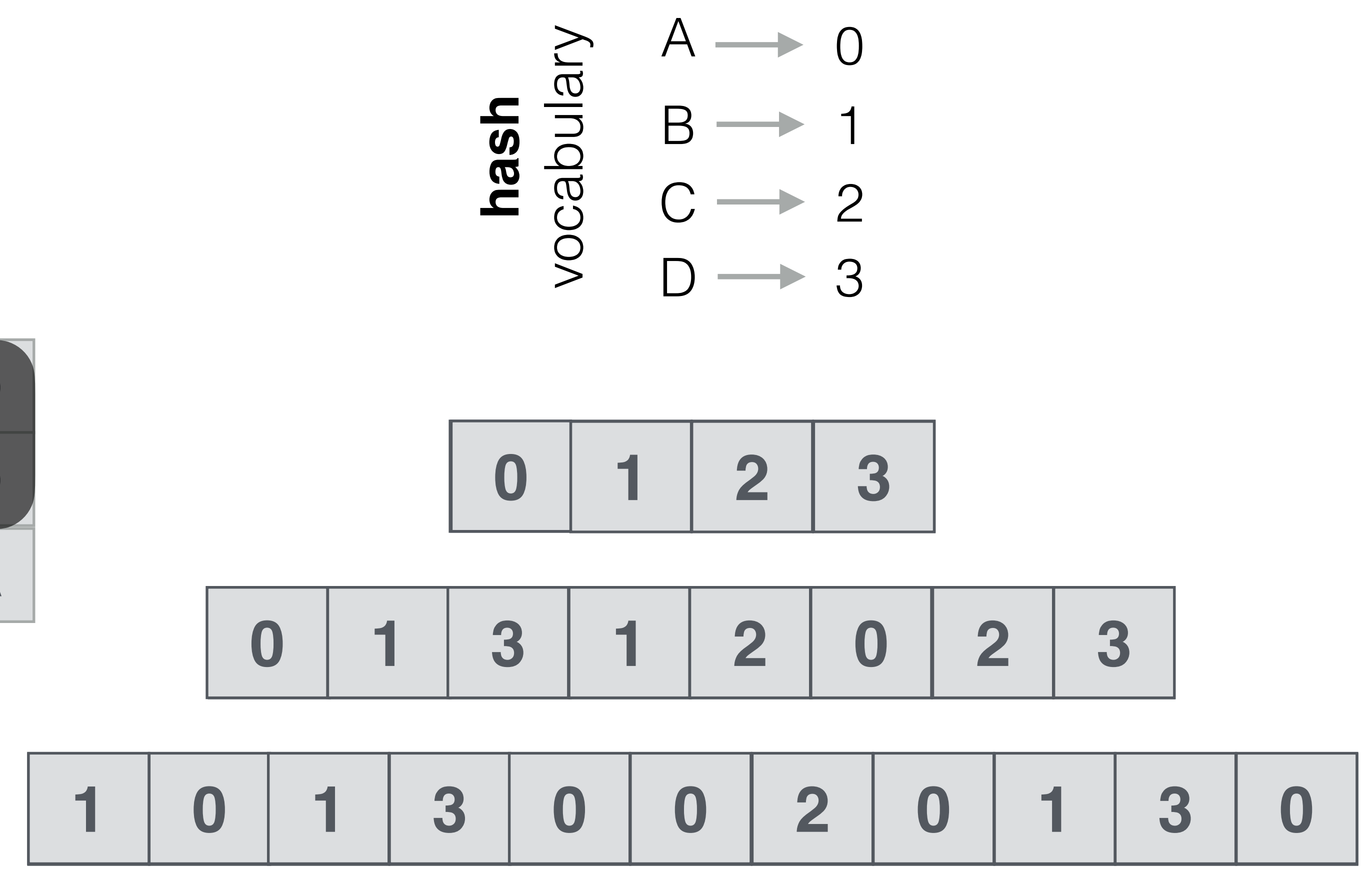
Trie Indexing



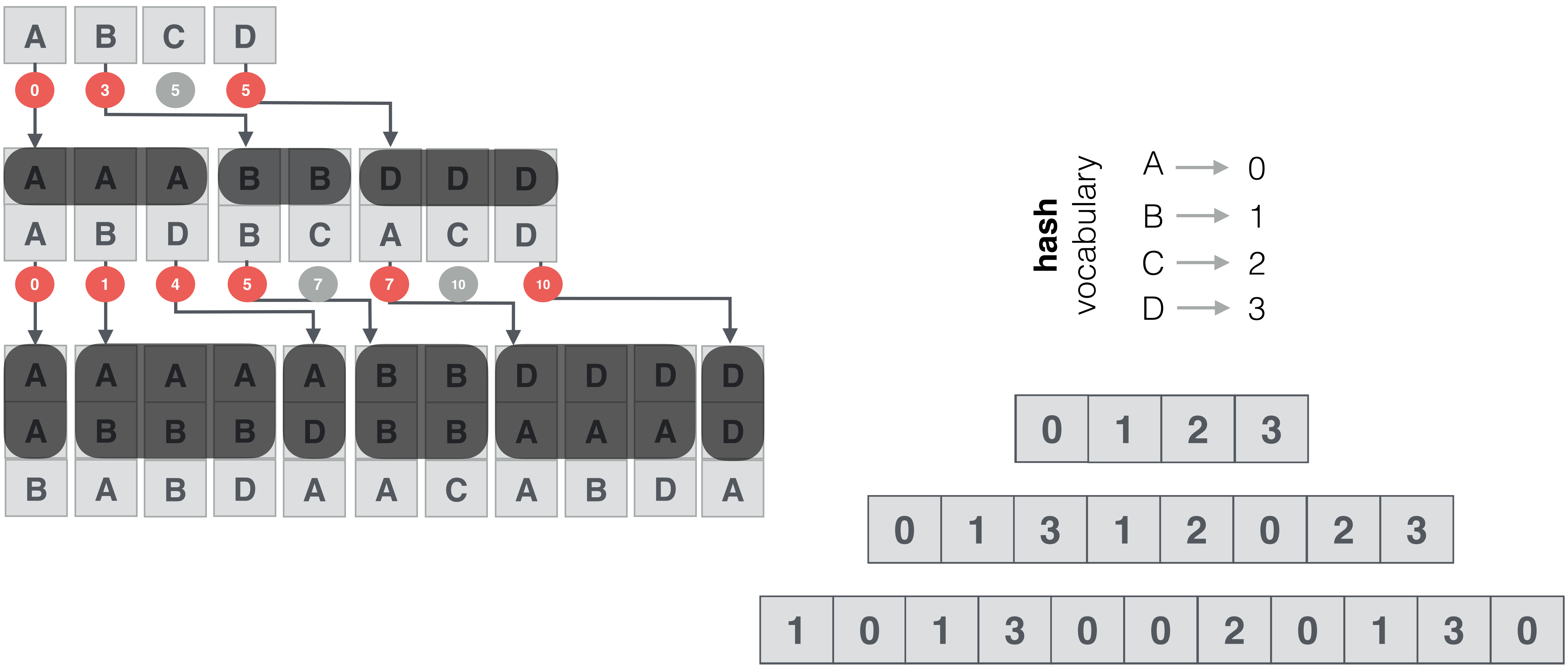


Trie Indexing

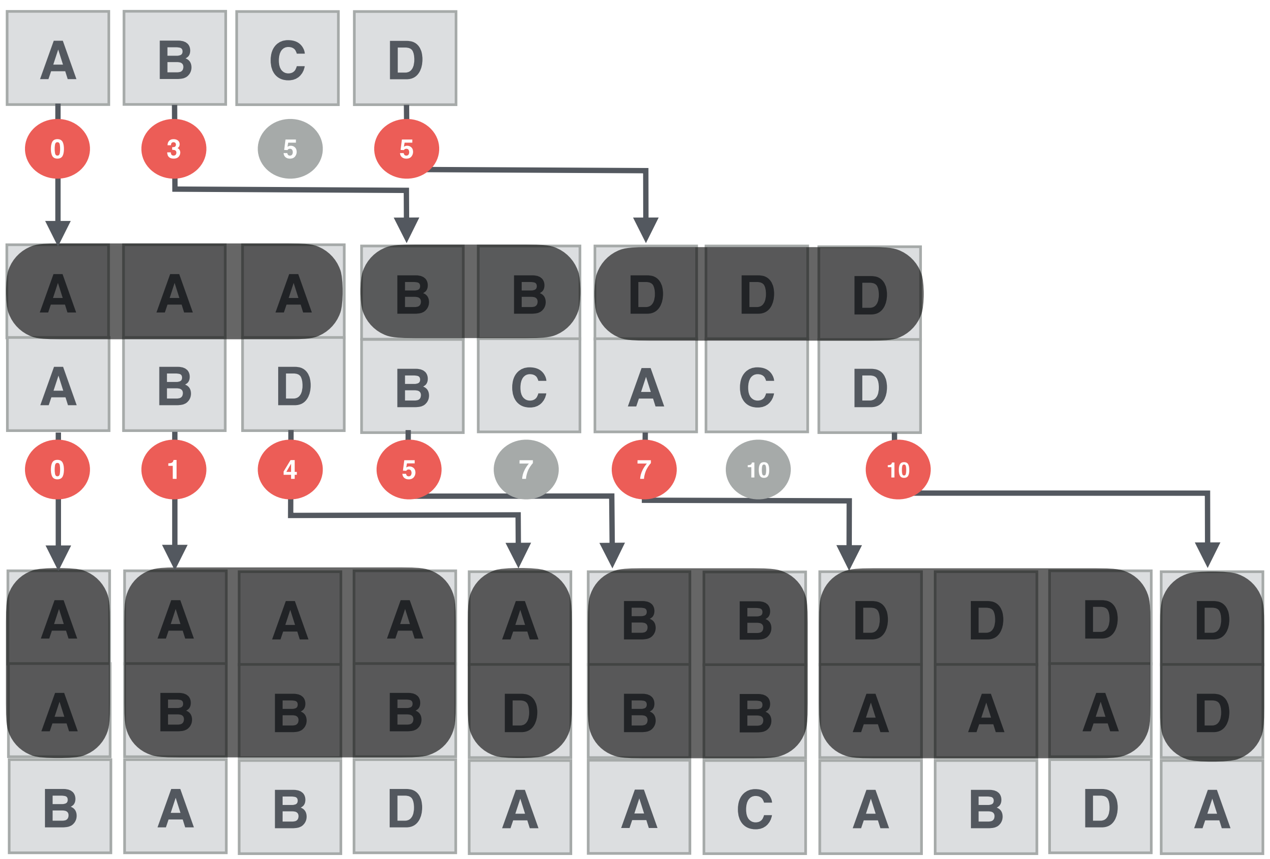


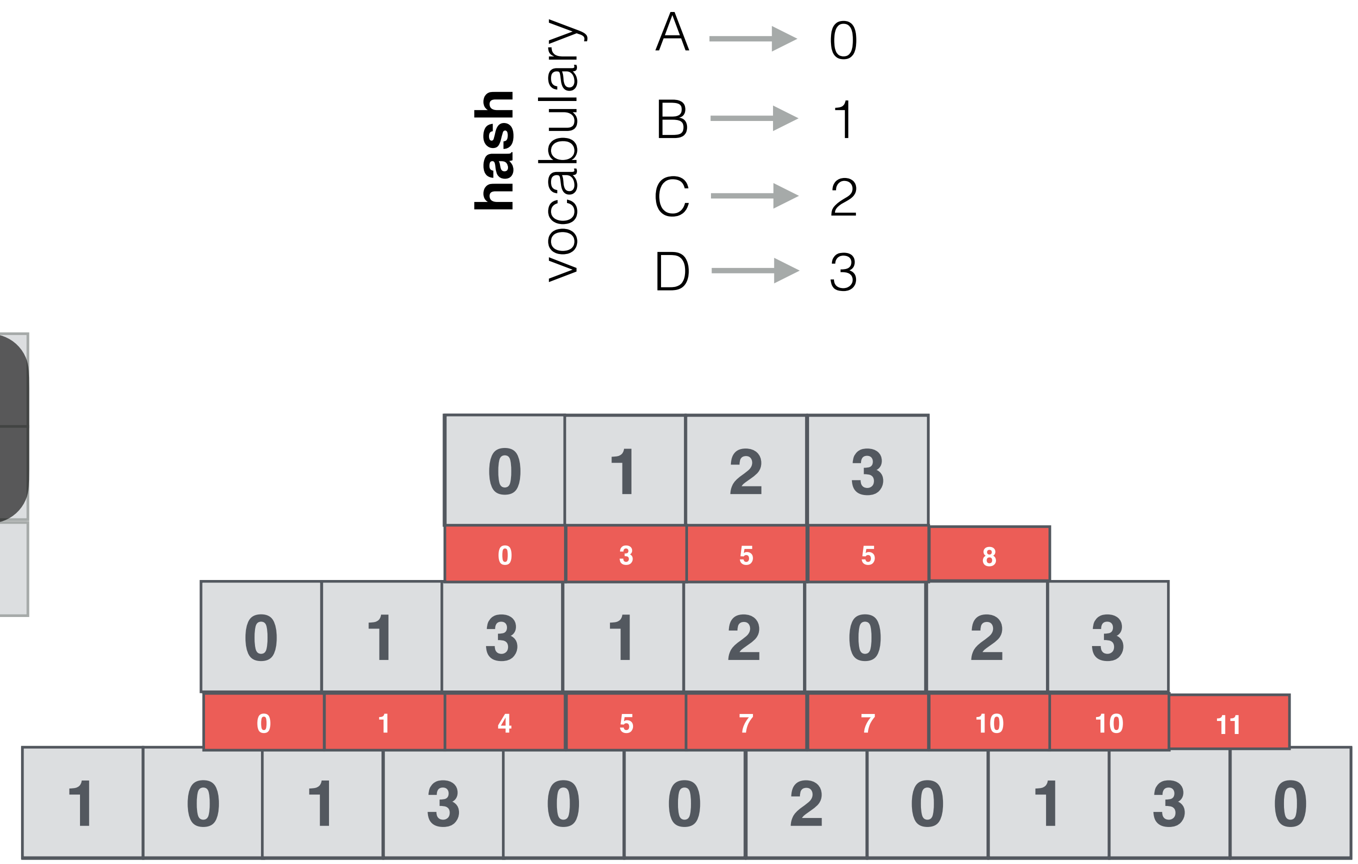


Trie Indexing

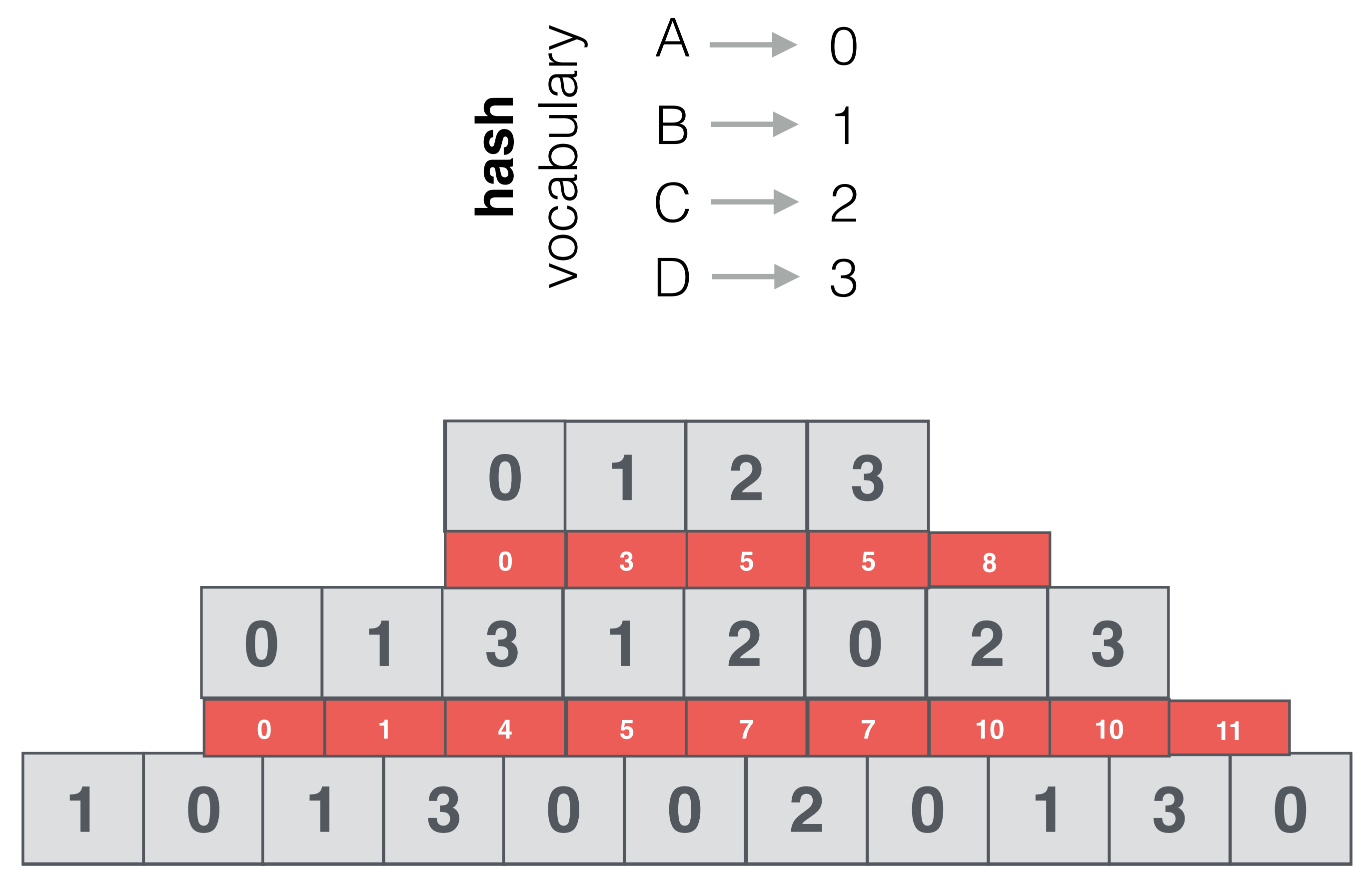


Trie Indexing



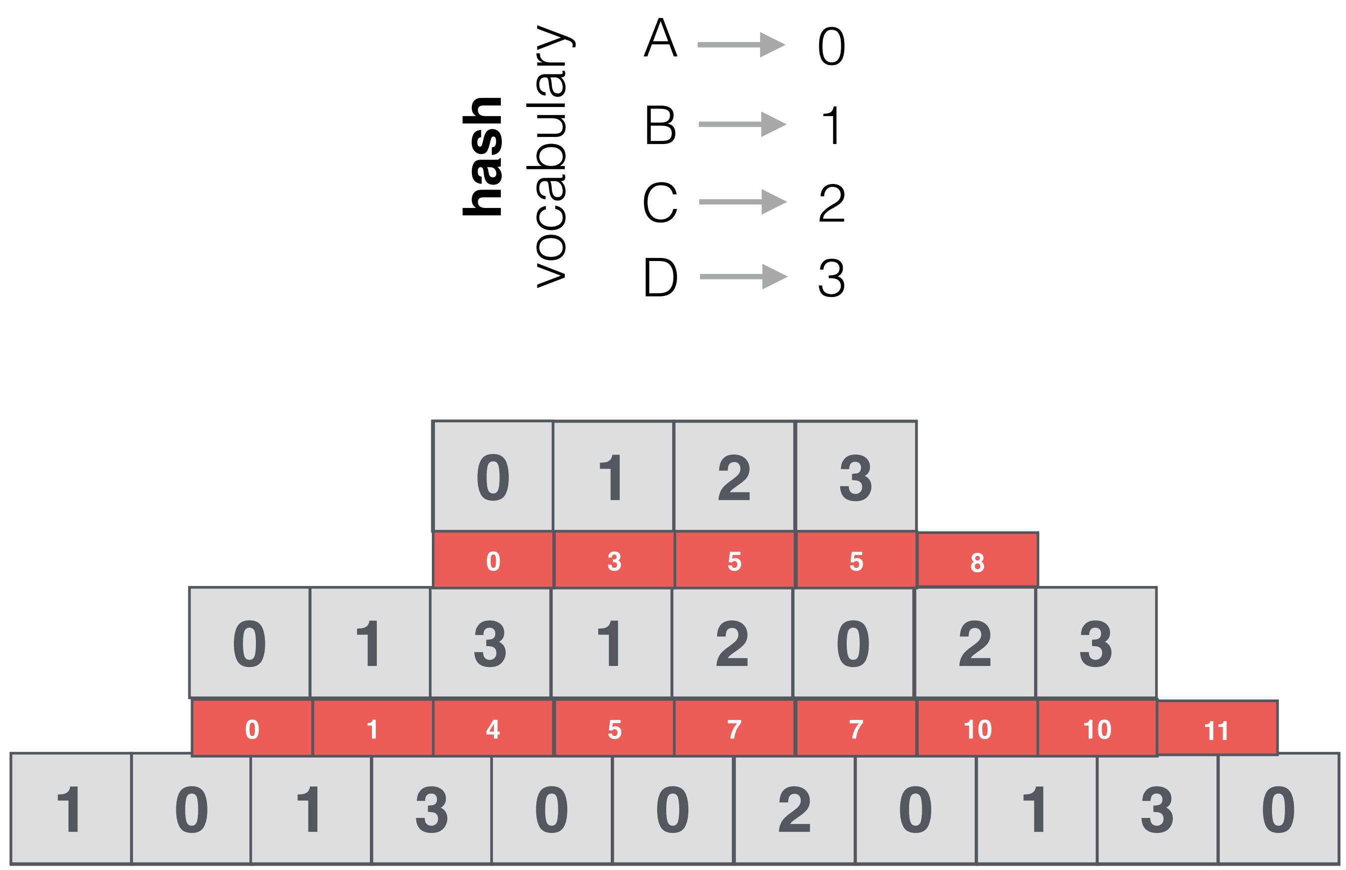


Trie Indexing



Trie Indexing

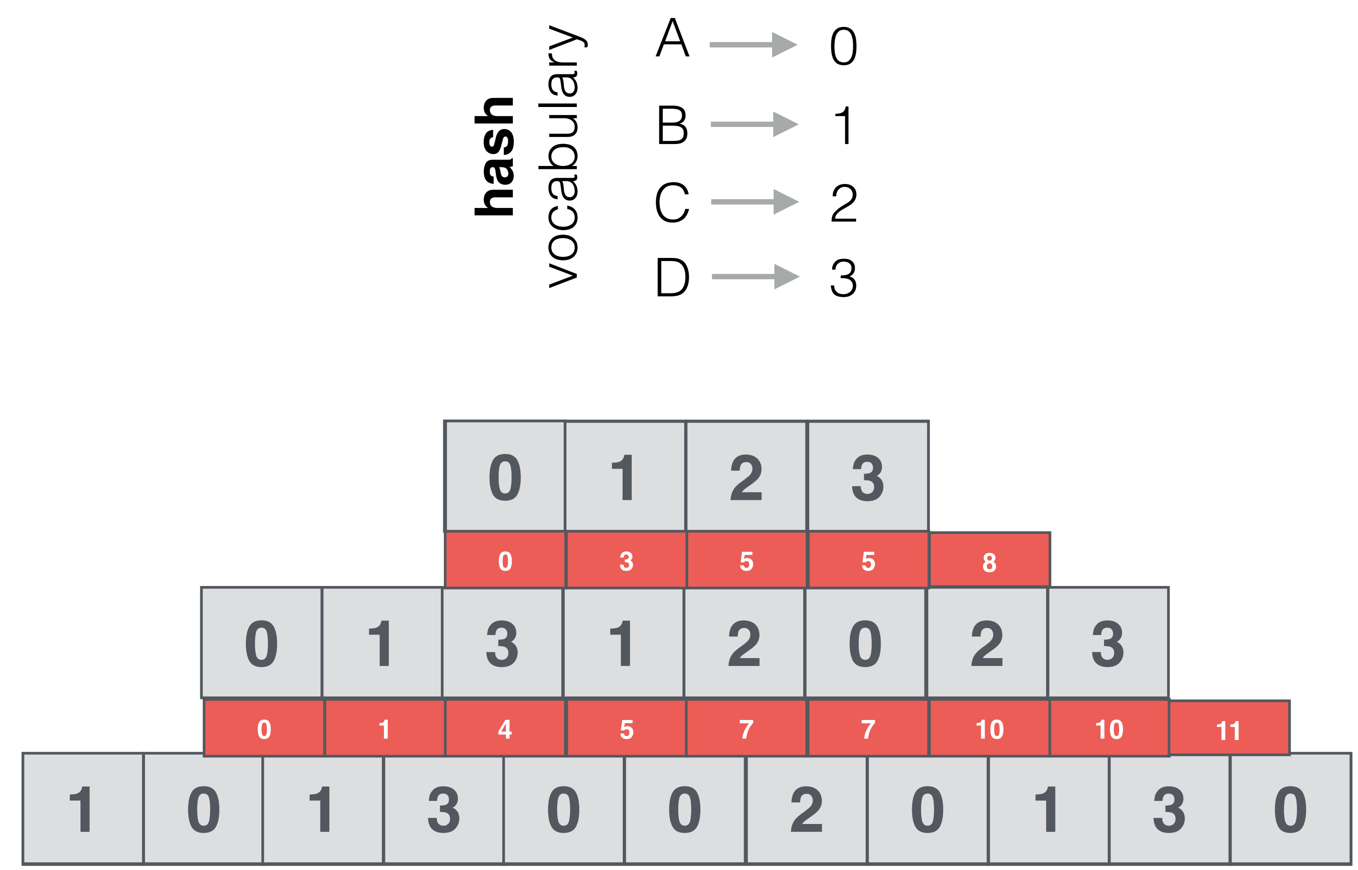
We need an encoder for integer sequences, supporting fast **random** Access.



Trie Indexing

We need an encoder for integer sequences, supporting fast **random** Access.

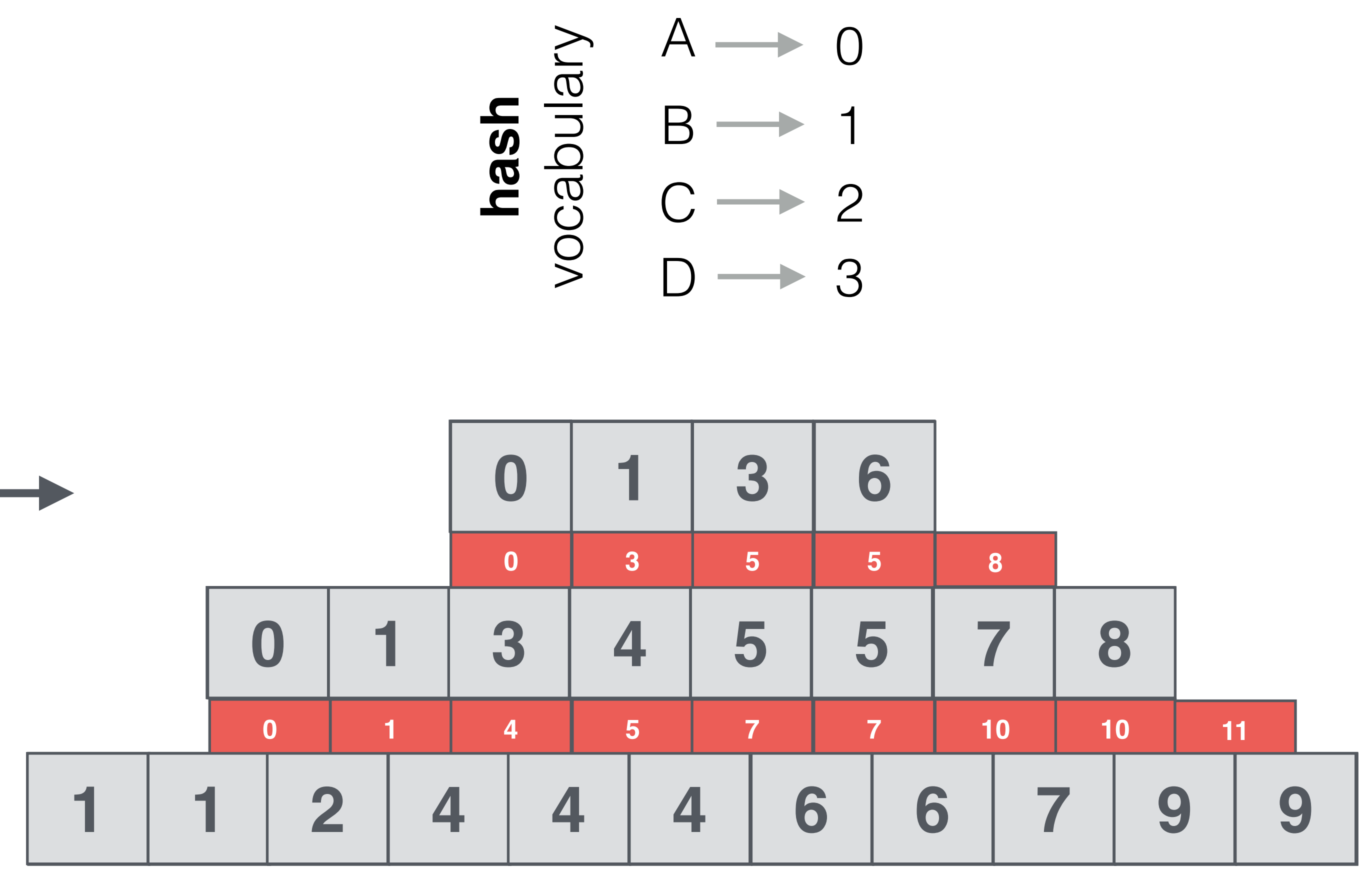
Take *range-wise* prefix sums on gram-ID sequences.



Trie Indexing

We need an encoder for integer sequences, supporting fast **random** Access.

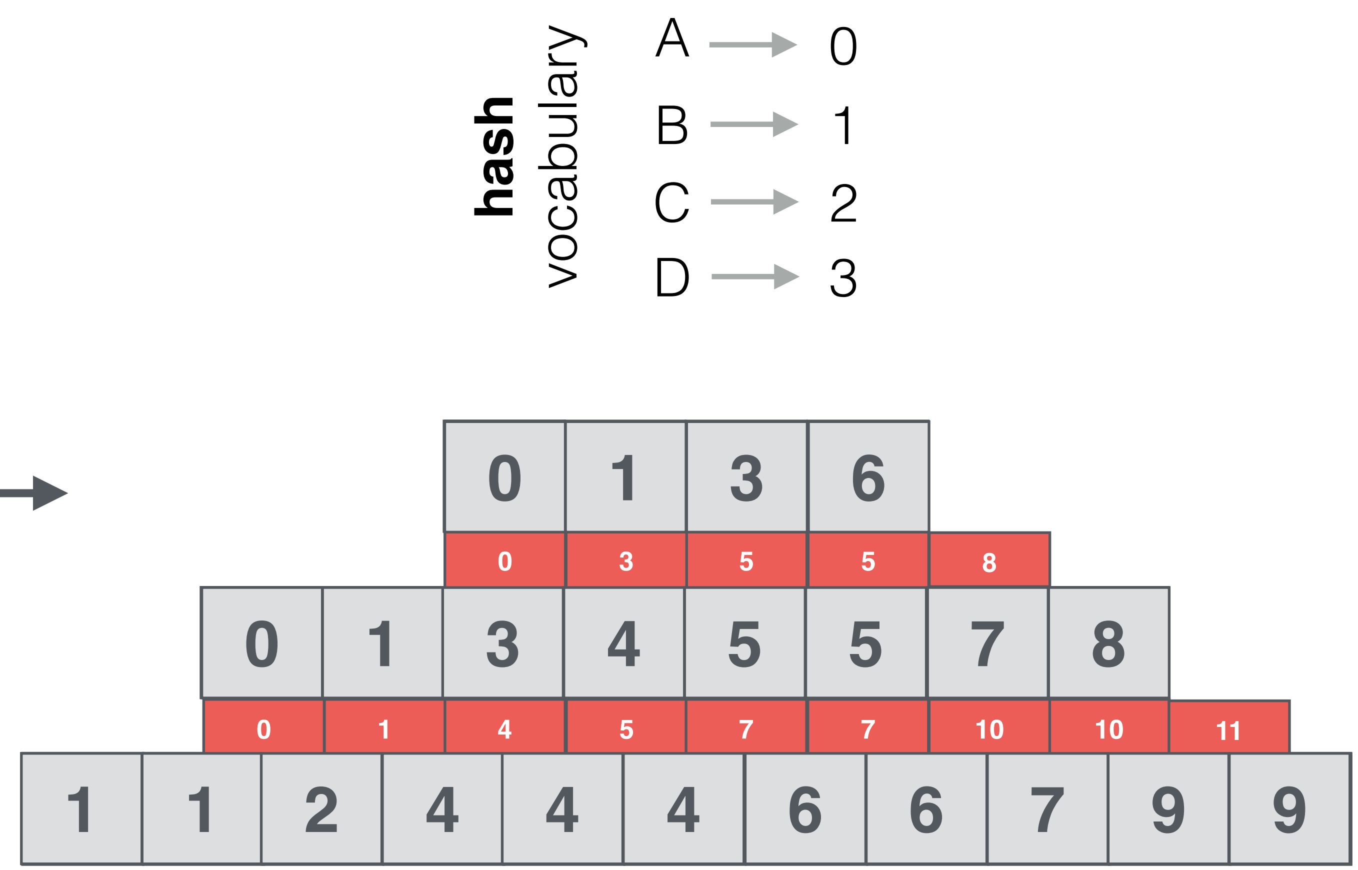
Take *range-wise* prefix sums on gram-ID sequences.



Trie Indexing

We need an encoder for integer sequences, supporting fast **random** Access.

Take *range-wise* prefix sums on gram-ID sequences.



Elias-Fano Tries

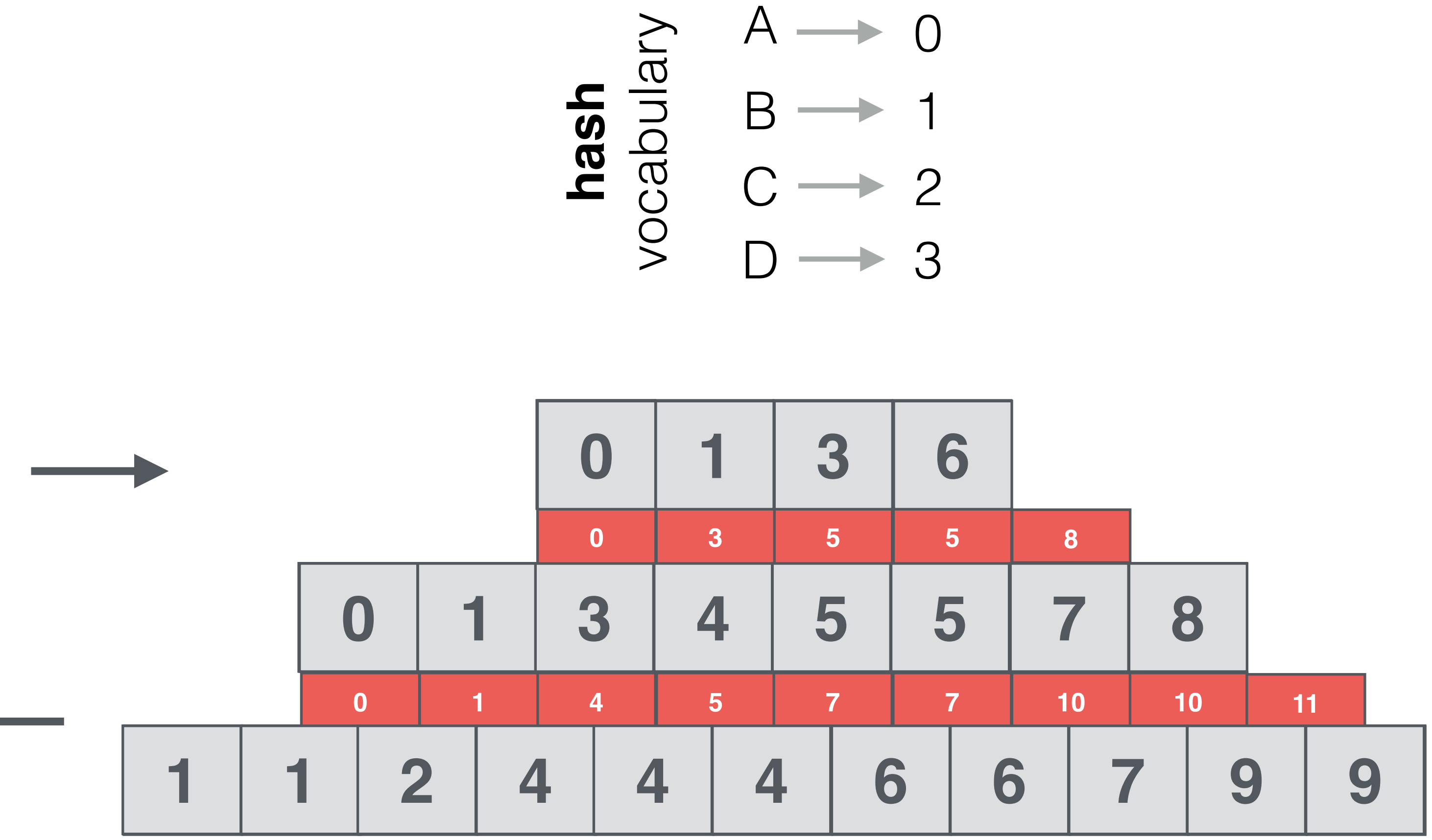
One Successor query per level

Constant-time random Access

Trie Indexing

We need an encoder for integer sequences, supporting fast **random** Access.

Take *range-wise* prefix sums on gram-ID sequences.



Elias-Fano Tries

One Successor query per level

Constant-time random Access

Remember:
Elias-Fano takes
log(u/n) + 2 bits
per integer

Context-based ID Remapping

Observation: the number of words following a given context is **small**.

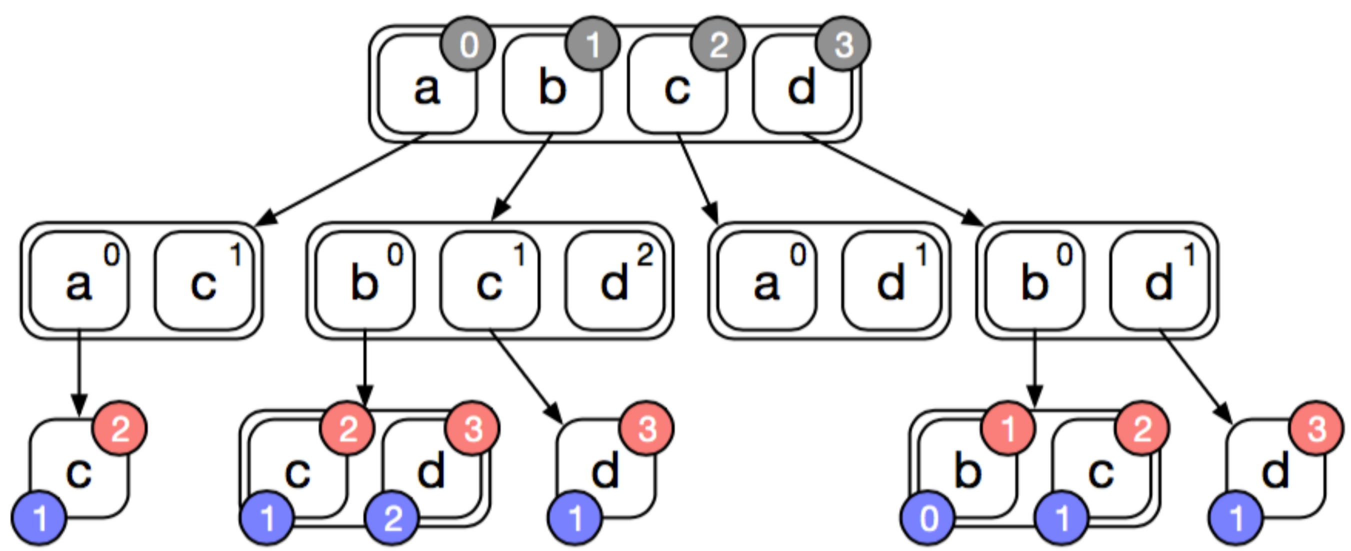
High-level idea: map a word ID to the **position** it takes within its *sibling* IDs (the IDs following a context of fixed length k).

Context-based ID Remapping

Observation: the number of words following a given context is **small**.

High-level idea: map a word ID to the **position** it takes within its *sibling* IDs (the IDs following a context of fixed length *k*).

$k = 1$

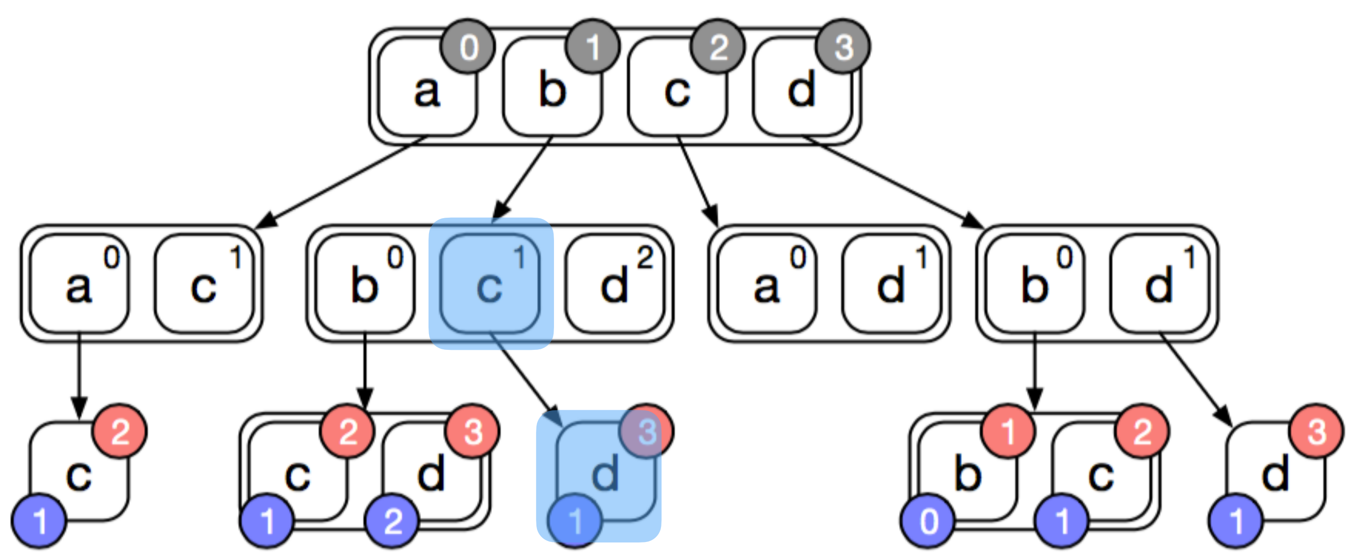


Context-based ID Remapping

Observation: the number of words following a given context is **small**.

High-level idea: map a word ID to the **position** it takes within its *sibling* IDs (the IDs following a context of fixed length *k*).

$k = 1$

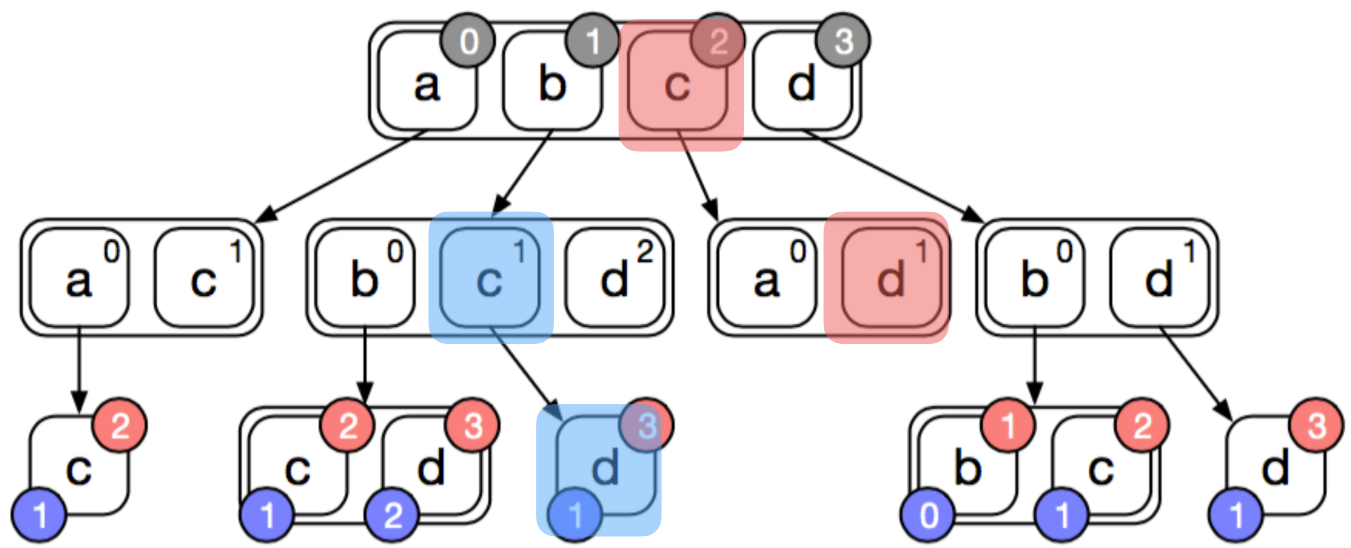


Context-based ID Remapping

Observation: the number of words following a given context is **small**.

High-level idea: map a word ID to the **position** it takes within its *sibling* IDs (the IDs following a context of fixed length *k*).

$k = 1$

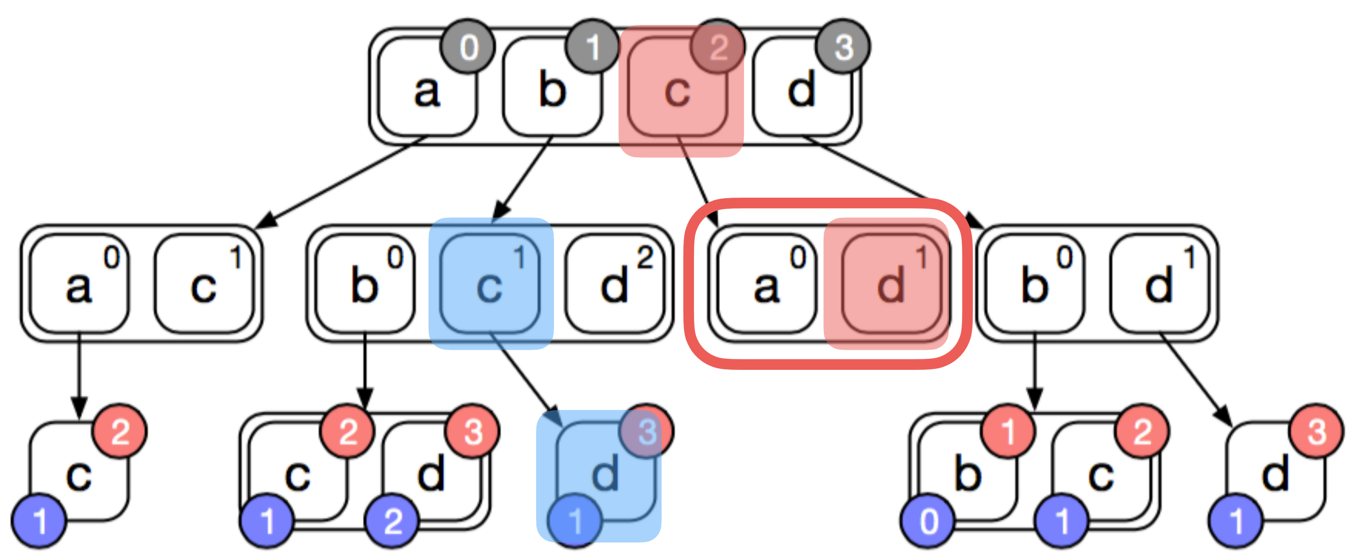


Context-based ID Remapping

Observation: the number of words following a given context is **small**.

High-level idea: map a word ID to the **position** it takes within its *sibling* IDs (the IDs following a context of fixed length *k*).

$k = 1$



Context-based ID Remapping

Observation: the number of words following a given context is **small**.

High-level idea: map a word ID to the **position** it takes within its *sibling* IDs (the IDs following a context of fixed length k).

Context-based ID Remapping

Observation: the number of words following a given context is **small**.

High-level idea: map a word ID to the **position** it takes within its *sibling* IDs (the IDs following a context of fixed length *k*).

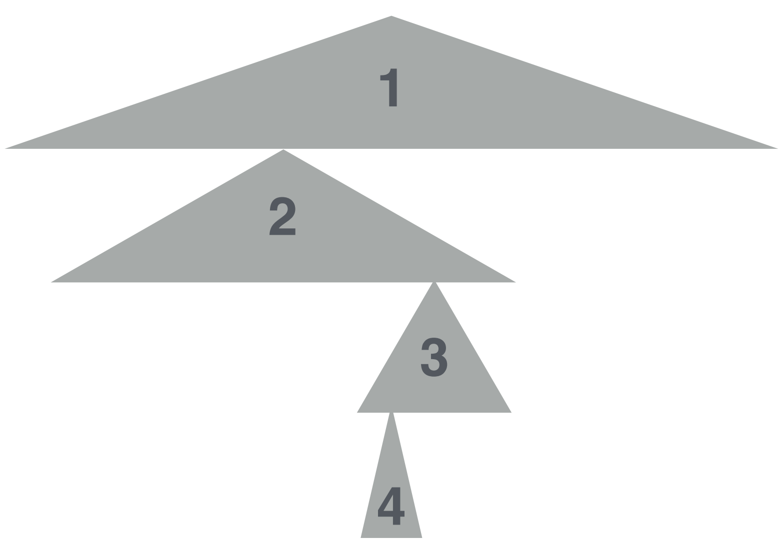
- **Millions** of unigrams.
- Height 5: **longer** contexts.
- The number of siblings has a **funnel-**shaped distribution.

Context-based ID Remapping

Observation: the number of words following a given context is **small**.

High-level idea: map a word ID to the **position** it takes within its *sibling* IDs (the IDs following a context of fixed length *k*).

- **Millions** of unigrams.
- Height 5: **longer** contexts.
- The number of siblings has a **funnel-**shaped distribution.

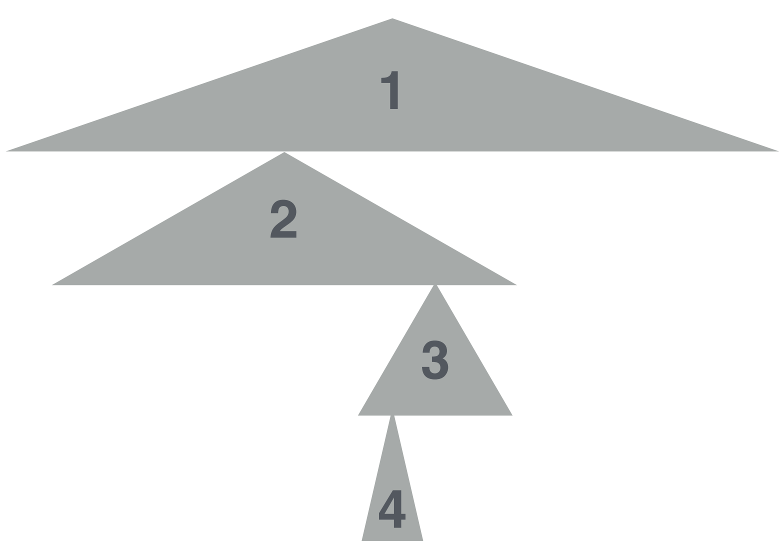


Context-based ID Remapping

Observation: the number of words following a given context is **small**.

High-level idea: map a word ID to the **position** it takes within its *sibling* IDs (the IDs following a context of fixed length *k*).

- **Millions** of unigrams.
- Height 5: **longer** contexts.
- The number of siblings has a **funnel-**shaped distribution.



u/n by varying context-length *k*

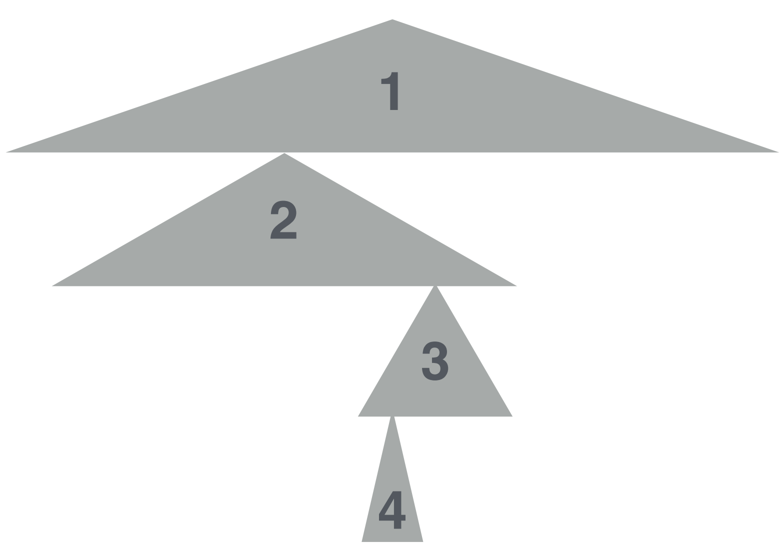
	k	3-grams	4-grams	5-grams
Europarl	0	2404	2782	2920
	1	213 (×11.28)	480 (×5.79)	646 (×4.52)
	2	2404	48 (×57.95)	101 (×28.91)
YahooV2	0	7350	7197	7417
	1	753 (×9.76)	1461 (×4.93)	1963 (×3.78)
	2	7350	104 (×69.20)	249 (×29.79)
GoogleV2	0	4050	6631	6793
	1	1025 (×3.95)	2192 (×3.03)	2772 (×2.45)
	2	4050	221 (×30.00)	503 (×13.50)

Context-based ID Remapping

Observation: the number of words following a given context is **small**.

High-level idea: map a word ID to the **position** it takes within its *sibling* IDs (the IDs following a context of fixed length *k*).

- **Millions** of unigrams.
- Height 5: **longer** contexts.
- The number of siblings has a **funnel-**shaped distribution.



u/n by varying context-length *k*

	k	3-grams	4-grams	5-grams
Europarl	0	2404	2782	2920
	1	213 (×11.28)	480 (×5.79)	646 (×4.52)
	2	2404	48 (×57.95)	101 (×28.91)
YahooV2	0	7350	7197	7417
	1	753 (×9.76)	1461 (×4.93)	1963 (×3.78)
	2	7350	104 (×69.20)	249 (×29.79)
GoogleV2	0	4050	6631	6793
	1	1025 (×3.95)	2192 (×3.03)	2772 (×2.45)
	2	4050	221 (×30.00)	503 (×13.50)

Experimental Analysis - EF/PEF (R)Trie

N	Europarl	YahooV2	GoogleV2
	n	n	n
1	304 579	3 475 482	24 357 349
2	5 192 260	53 844 927	665 752 080
3	18 908 249	187 639 522	7 384 478 110
4	33 862 651	287 562 409	1 642 783 634
5	43 160 518	295 701 337	1 413 870 914
Total	101 428 257	828 223 677	11 131 242 087
gzip bpg	6.98	6.45	6.20

Test machine
Intel Xeon E5-2630 v3, 2.4 GHz
193 GB of RAM, Linux 64 bits

C++ implementation
gcc 5.4.1 with the highest optimization setting

Experimental Analysis - EF/PEF (R)Trie

N	Europarl n	YahooV2 n	GoogleV2 n
1	304 579	3 475 482	24 357 349
2	5 192 260	53 844 927	665 752 080
3	18 908 249	187 639 522	7 384 478 110
4	33 862 651	287 562 409	1 642 783 634
5	43 160 518	295 701 337	1 413 870 914
Total	101 428 257	828 223 677	11 131 242 087
gzip bpg	6.98	6.45	6.20

Test machine
Intel Xeon E5-2630 v3, 2.4 GHz
193 GB of RAM, Linux 64 bits

C++ implementation
gcc 5.4.1 with the highest optimization setting

		Europarl		YahooV2		GoogleV2	
		bpg	$\mu s \times$ query	bpg	$\mu s \times$ query	bpg	$\mu s \times$ query
	EF	1.97	1.28	2.17	1.60	2.13	2.09
	PEF	1.87 (−4.99%)	1.35 (+5.93%)	1.91 (−12.03%)	1.73 (+8.00%)	1.52 (−28.60%)	1.91 (−8.79%)
CONTEXT-BASED ID REMAPPING $k = 1$	EF	1.67 (−15.30%)	1.58 (+23.86%)	1.89 (−12.92%)	2.05 (+28.07%)	1.91 (−10.24%)	3.03 (+44.61%)
	PEF	1.53 (−22.36%)	1.61 (+25.89%)	1.63 (−24.91%)	2.16 (+35.22%)	1.31 (−38.71%)	2.30 (+9.88%)
CONTEXT-BASED ID REMAPPING $k = 2$	EF	1.46 (−25.62%)	1.60 (+25.17%)	1.68 (−22.32%)	2.08 (+30.23%)	—	—
	PEF	1.28 (−34.87%)	1.64 (+28.12%)	1.38 (−36.15%)	2.15 (+34.81%)	—	—

Experimental Analysis - EF/PEF (R)Trie

N	Europarl n	YahooV2 n	GoogleV2 n
1	304 579	3 475 482	24 357 349
2	5 192 260	53 844 927	665 752 080
3	18 908 249	187 639 522	7 384 478 110
4	33 862 651	287 562 409	1 642 783 634
5	43 160 518	295 701 337	1 413 870 914
Total	101 428 257	828 223 677	11 131 242 087
gzip bpg	6.98	6.45	6.20

Test machine
Intel Xeon E5-2630 v3, 2.4 GHz
193 GB of RAM, Linux 64 bits

C++ implementation
`gcc` 5.4.1 with the highest optimization setting

		Europarl		YahooV2		GoogleV2	
		bpg	$\mu s \times$ query	bpg	$\mu s \times$ query	bpg	$\mu s \times$ query
	EF	1.97	1.28	2.17	1.60	2.13	2.09
	PEF	1.87 (−4.99%)	1.35 (+5.93%)	1.91 (−12.03%)	1.73 (+8.00%)	1.52 (−28.60%)	1.91 (−8.79%)
CONTEXT-BASED ID REMAPPING $k=1$	EF	1.67 (−15.30%)	1.58 (+23.86%)	1.89 (−12.92%)	2.05 (+28.07%)	1.91 (−10.24%)	3.03 (+44.61%)
CONTEXT-BASED ID REMAPPING $k=1$	PEF	1.53 (−22.36%)	1.61 (+25.89%)	1.63 (−24.91%)	2.16 (+35.22%)	1.31 (−38.71%)	2.30 (+9.88%)
CONTEXT-BASED ID REMAPPING $k=2$	EF	1.46 (−25.62%)	1.60 (+25.17%)	1.68 (−22.32%)	2.08 (+30.23%)	—	—
CONTEXT-BASED ID REMAPPING $k=2$	PEF	1.28 (−34.87%)	1.64 (+28.12%)	1.38 (−36.15%)	2.15 (+34.81%)	—	—

Experimental Analysis - EF/PEF (R)Trie

N	Europarl n	YahooV2 n	GoogleV2 n
1	304 579	3 475 482	24 357 349
2	5 192 260	53 844 927	665 752 080
3	18 908 249	187 639 522	7 384 478 110
4	33 862 651	287 562 409	1 642 783 634
5	43 160 518	295 701 337	1 413 870 914
Total	101 428 257	828 223 677	11 131 242 087
gzip bpg	6.98	6.45	6.20

Test machine
Intel Xeon E5-2630 v3, 2.4 GHz
193 GB of RAM, Linux 64 bits

C++ implementation
`gcc` 5.4.1 with the highest optimization setting

		Europarl		YahooV2		GoogleV2	
		bpg	$\mu s \times$ query	bpg	$\mu s \times$ query	bpg	$\mu s \times$ query
	EF	1.97	1.28	2.17	1.60	2.13	2.09
	PEF	1.87 (−4.99%)	1.35 (+5.93%)	1.91 (−12.03%)	1.73 (+8.00%)	1.52 (−28.60%)	1.91 (−8.79%)
CONTEXT-BASED ID REMAPPING $k = 1$	EF	1.67 (−15.30%)	1.58 (+23.86%)	1.89 (−12.92%)	2.05 (+28.07%)	1.91 (−10.24%)	3.03 (+44.61%)
	PEF	1.53 (−22.36%)	1.61 (+25.89%)	1.63 (−24.91%)	2.16 (+35.22%)	1.31 (−38.71%)	2.30 (+9.88%)
CONTEXT-BASED ID REMAPPING $k = 2$	EF	1.46 (−25.62%)	1.60 (+25.17%)	1.68 (−22.32%)	2.08 (+30.23%)	—	—
	PEF	1.28 (−34.87%)	1.64 (+28.12%)	1.38 (−36.15%)	2.15 (+34.81%)	—	—

Context-based ID Remapping

- reduces space by more than **36%** on average → **you will** notice this**!**

Experimental Analysis - EF/PEF (R)Trie

N	Europarl n	YahooV2 n	GoogleV2 n
1	304 579	3 475 482	24 357 349
2	5 192 260	53 844 927	665 752 080
3	18 908 249	187 639 522	7 384 478 110
4	33 862 651	287 562 409	1 642 783 634
5	43 160 518	295 701 337	1 413 870 914
Total	101 428 257	828 223 677	11 131 242 087
gzip bpg	6.98	6.45	6.20

Test machine
Intel Xeon E5-2630 v3, 2.4 GHz
193 GB of RAM, Linux 64 bits

C++ implementation
gcc 5.4.1 with the highest optimization setting

		Europarl		YahooV2		GoogleV2	
		bpg	$\mu s \times$ query	bpg	$\mu s \times$ query	bpg	$\mu s \times$ query
	EF	1.97	1.28	2.17	1.60	2.13	2.09
	PEF	1.87 (−4.99%)	1.35 (+5.93%)	1.91 (−12.03%)	1.73 (+8.00%)	1.52 (−28.60%)	1.91 (−8.79%)
CONTEXT-BASED ID REMAPPING $k = 1$	EF	1.67 (−15.30%)	1.58 (+23.86%)	1.89 (−12.92%)	2.05 (+28.07%)	1.91 (−10.24%)	3.03 (+44.61%)
CONTEXT-BASED ID REMAPPING $k = 1$	PEF	1.53 (−22.36%)	1.61 (+25.89%)	1.63 (−24.91%)	2.16 (+35.22%)	1.31 (−38.71%)	2.30 (+9.88%)
CONTEXT-BASED ID REMAPPING $k = 2$	EF	1.46 (−25.62%)	1.60 (+25.17%)	1.68 (−22.32%)	2.08 (+30.23%)	—	—
CONTEXT-BASED ID REMAPPING $k = 2$	PEF	1.28 (−34.87%)	1.64 (+28.12%)	1.38 (−36.15%)	2.15 (+34.81%)	—	—

Context-based ID Remapping

- reduces space by more than **36%** on average ⟶ **you will** notice this**!**

Experimental Analysis - EF/PEF (R)Trie

N	Europarl n	YahooV2 n	GoogleV2 n
1	304 579	3 475 482	24 357 349
2	5 192 260	53 844 927	665 752 080
3	18 908 249	187 639 522	7 384 478 110
4	33 862 651	287 562 409	1 642 783 634
5	43 160 518	295 701 337	1 413 870 914
Total	101 428 257	828 223 677	11 131 242 087
gzip bpg	6.98	6.45	6.20

Test machine
Intel Xeon E5-2630 v3, 2.4 GHz
193 GB of RAM, Linux 64 bits

C++ implementation
`gcc` 5.4.1 with the highest optimization setting

		Europarl		YahooV2		GoogleV2	
		bpg	$\mu s \times$ query	bpg	$\mu s \times$ query	bpg	$\mu s \times$ query
	EF	1.97	1.28	2.17	1.60	2.13	2.09
	PEF	1.87 (−4.99%)	1.35 (+5.93%)	1.91 (−12.03%)	1.73 (+8.00%)	1.52 (−28.60%)	1.91 (−8.79%)
CONTEXT-BASED ID REMAPPING $k = 1$	EF	1.67 (−15.30%)	1.58 (+23.86%)	1.89 (−12.92%)	2.05 (+28.07%)	1.91 (−10.24%)	3.03 (+44.61%)
	PEF	1.53 (−22.36%)	1.61 (+25.89%)	1.63 (−24.91%)	2.16 (+35.22%)	1.31 (−38.71%)	2.30 (+9.88%)
CONTEXT-BASED ID REMAPPING $k = 2$	EF	1.46 (−25.62%)	1.60 (+25.17%)	1.68 (−22.32%)	2.08 (+30.23%)	—	—
	PEF	1.28 (−34.87%)	1.64 (+28.12%)	1.38 (−36.15%)	2.15 (+34.81%)	—	—

Context-based ID Remapping

- reduces space by more than **36%** on average → **you will** notice this**!**
- brings approximately **30%** more time → **will you** notice this**?**

Experimental Analysis - Overall comparison

	Europarl		YahooV2		GoogleV2	
	bpg	μs × query	bpg	μs × query	bpg	μs × query
PEF-Trie	1.87	1.35	1.91	1.73	1.52	1.91
PEF-RTrie	1.28	1.64	1.38	2.15	1.31	2.30
BerkeleyLM C.	1.70 (−8.89%) (+32.90%)	2.83 (+108.88%) (+72.70%)	1.69 (−11.41%) (+22.04%)	3.48 (+101.84%) (+61.70%)	1.45 (−4.87%) (+10.83%)	4.13 (+116.57%) (+79.76%)
BerkeleyLM H.3	6.70 (+258.81%) (+423.40%)	0.97 (−28.46%) (−40.85%)	7.82 (+310.38%) (+465.36%)	1.13 (−34.35%) (−47.41%)	9.24 (+507.79%) (+608.07%)	2.18 (+13.95%) (−5.42%)
BerkeleyLM H.50	7.96 (+326.03%) (+521.45%)	0.97 (−28.49%) (−40.88%)	9.37 (+391.32%) (+576.87%)	0.96 (−44.27%) (−55.35%)	—	—
Expgram	2.06 (+10.18%) (+60.73%)	2.80 (+106.61%) (+70.82%)	2.24 (+17.36%) (+61.68%)	9.23 (+435.33%) (+328.87%)	—	—
KenLM T.	2.99 (+60.11%) (+133.56%)	1.28 (−5.47%) (−21.84%)	3.44 (+80.39%) (+148.52%)	1.94 (+12.32%) (−10.01%)	—	—
Marisa	3.61 (+93.09%) (+181.66%)	2.06 (+52.00%) (+25.67%)	3.81 (+99.60%) (+174.98%)	3.24 (+87.96%) (+50.58%)	—	—
RandLM	1.81 (−3.06%) (+41.41%)	4.39 (+224.20%) (+168.04%)	2.02 (+6.18%) (+46.29%)	5.08 (+194.35%) (+135.82%)	2.60 (+70.73%) (+98.90%)	9.25 (+384.54%) (+302.19%)

Experimental Analysis - Overall comparison

	Europarl		YahooV2		GoogleV2	
	bpg	$\mu s \times$ query	bpg	$\mu s \times$ query	bpg	$\mu s \times$ query
PEF-Trie	1.87	1.35	1.91	1.73	1.52	1.91
PEF-RTrie	1.28	1.64	1.38	2.15	1.31	2.30
BerkeleyLM C.	1.70 (−8.89%) (+32.90%)	2.83 (+108.88%) (+72.70%)	1.69 (−11.41%) (+22.04%)	3.48 (+101.84%) (+61.70%)	1.45 (−4.87%) (+10.83%)	4.13 (+116.57%) (+79.76%)
BerkeleyLM H.3	6.70 (+258.81%) (+423.40%)	0.97 (−28.46%) (−40.85%)	7.82 (+310.38%) (+465.36%)	1.13 (−34.35%) (−47.41%)	9.24 (+507.79%) (+608.07%)	2.18 (+13.95%) (−5.42%)
BerkeleyLM H.50	7.96 (+326.03%) (+521.45%)	0.97 (−28.49%) (−40.88%)	9.37 (+391.32%) (+576.87%)	0.96 (−44.27%) (−55.35%)	—	—
Expgram	2.06 (+10.18%) (+60.73%)	2.80 (+106.61%) (+70.82%)	2.24 (+17.36%) (+61.68%)	9.23 (+435.33%) (+328.87%)	—	—
KenLM T.	2.99 (+60.11%) (+133.56%)	1.28 (−5.47%) (−21.84%)	3.44 (+80.39%) (+148.52%)	1.94 (+12.32%) (−10.01%)	—	—
Marisa	3.61 (+93.09%) (+181.66%)	2.06 (+52.00%) (+25.67%)	3.81 (+99.60%) (+174.98%)	3.24 (+87.96%) (+50.58%)	—	—
RandLM	1.81 (−3.06%) (+41.41%)	4.39 (+224.20%) (+168.04%)	2.02 (+6.18%) (+46.29%)	5.08 (+194.35%) (+135.82%)	2.60 (+70.73%) (+98.90%)	9.25 (+384.54%) (+302.19%)

Experimental Analysis - Overall comparison

	Europarl bpg	Europarl μs × query	YahooV2 bpg	YahooV2 μs × query	GoogleV2 bpg	GoogleV2 μs × query
PEF-Trie	1.87	1.35	1.91	1.73	1.52	1.91
PEF-RTrie	1.28	1.64	1.38	2.15	1.31	2.30
BerkeleyLM C.	1.70 (−8.89%) (+32.90%)	2.83 (+108.88%) (+72.70%)	1.69 (−11.41%) (+22.04%)	3.48 (+101.84%) (+61.70%)	1.45 (−4.87%) (+10.83%)	4.13 (+116.57%) (+79.76%)
BerkeleyLM H.3	6.70 (+258.81%) (+423.40%)	0.97 (−28.46%) (−40.85%)	7.82 (+310.38%) (+465.36%)	1.13 (−34.35%) (−47.41%)	9.24 (+507.79%) (+608.07%)	2.18 (+13.95%) (−5.42%)
BerkeleyLM H.50	7.96 (+326.03%) (+521.45%)	0.97 (−28.49%) (−40.88%)	9.37 (+391.32%) (+576.87%)	0.96 (−44.27%) (−55.35%)	—	—
Expgram	2.06 (+10.18%) (+60.73%)	2.80 (+106.61%) (+70.82%)	2.24 (+17.36%) (+61.68%)	9.23 (+435.33%) (+328.87%)	—	—
KenLM T.	2.99 **2.3X** (+133.56%)	1.28 (−5.47%) (−21.84%)	3.44 **2.5X** (+148.52%)	1.94 (+12.32%) (−10.01%)	—	—
Marisa	3.61 (+93.09%) (+181.66%)	2.06 (+52.00%) (+25.67%)	3.81 (+99.60%) (+174.98%)	3.24 (+87.96%) (+50.58%)	—	—
RandLM	1.81 (−3.06%) (+41.41%)	4.39 (+224.20%) (+168.04%)	2.02 (+6.18%) (+46.29%)	5.08 (+194.35%) (+135.82%)	2.60 (+70.73%) (+98.90%)	9.25 (+384.54%) (+302.19%)

Experimental Analysis - Overall comparison

	Europarl		YahooV2		GoogleV2	
	bpg	μs × query	bpg	μs × query	bpg	μs × query
PEF-Trie	1.87	1.35	1.91	1.73	1.52	1.91
PEF-RTrie	1.28	1.64	1.38	2.15	1.31	2.30
BerkeleyLM C.	1.70 (−8.89%) (+32.90%)	2.83 (+108.88%) (+72.70%)	1.69 (−11.41%) (+22.04%)	3.48 (+101.84%) (+61.70%)	1.45 (−4.87%) (+10.83%)	4.13 (+116.57%) (+79.76%)
BerkeleyLM H.3	6.70 (+258.81%) (+423.40%)	0.97 (−28.46%) (−40.85%)	7.82 (+310.38%) (+465.36%)	1.13 (−34.35%) (−47.41%)	9.24 (+507.79%) (+608.07%)	2.18 (+13.95%) (−5.42%)
BerkeleyLM H.50	7.96 (+326.03%) (+521.45%)	0.97 (−28.49%) (−40.88%)	9.37 (+391.32%) (+576.87%)	0.96 (−44.27%) (−55.35%)	—	—
Expgram	2.06 (+10.18%) (+60.73%)	2.80 (+106.61%) (+70.82%)	2.24 (+17.36%) (+61.68%)	9.23 (+435.33%) (+328.87%)	—	—
KenLM T.	2.99 [illegible] 2.3X (+133.56%)	1.28 (−5.47%) (−21.84%)	3.44 [illegible] 2.5X (+148.52%)	1.94 (+12.32%) (−10.01%)	—	—
Marisa	3.61 (+93.09%) (+181.66%)	2.06 (+52.00%) (+25.67%)	3.81 (+99.60%) (+174.98%)	3.24 (+87.96%) (+50.58%)	—	—
RandLM	1.81 (−3.06%) (+41.41%)	4.39 (+224.20%) (+168.04%)	2.02 (+6.18%) (+46.29%)	5.08 (+194.35%) (+135.82%)	2.60 (+70.73%) (+98.90%)	9.25 (+384.54%) (+302.19%)

Experimental Analysis - Overall comparison

	Europarl		YahooV2		GoogleV2	
	bpg	μs × query	bpg	μs × query	bpg	μs × query
PEF-Trie	1.87	1.35	1.91	1.73	1.52	1.91
PEF-RTrie	1.28	1.64	1.38	2.15	1.31	2.30
BerkeleyLM C.	1.70 (−8.89%) (+32.90%)	2.83 ([illegible]) (+72.70%) 2X	1.69 (−11.41%) (+22.04%)	3.48 ([illegible]) (+61.70%) 2X	1.45 (−4.87%) (+10.83%)	4.13 ([illegible]) (+79.76%) 2X
BerkeleyLM H.3	6.70 (+258.81%) ([illegible]) 2.5÷	0.97 (−28.46%) (−40.85%)	7.82 (+310.38%) ([illegible]) 3.1÷	1.13 (−34.35%) (−47.41%)	9.24 ([illegible]) ([illegible]) 5.5X	2.18 (+13.95%) (−5.42%)
BerkeleyLM H.50	7.96 ([illegible]) (+521.45%) 5.2X	0.97 (−28.49%) (−40.88%)	9.37 ([illegible]) (+576.87%) 5.8X	0.96 (−44.27%) (−55.35%)	—	—
Expgram	2.06 (+10.18%) (+60.73%)	2.80 ([illegible]) (+70.82%) 2X	2.24 (+17.36%) (+61.68%)	9.23 ([illegible]) (+328.87%) 3.5X	—	—
KenLM T.	2.99 ([illegible]) (+133.56%) 2.3X	1.28 (−5.47%) (−21.84%)	3.44 ([illegible]) (+148.52%) 2.5X	1.94 (+12.32%) (−10.01%)	—	—
Marisa	3.61 (+93.09%) ([illegible]) 2.8X	2.06 (+52.00%) (+25.67%)	3.81 (+99.60%) ([illegible]) 2.7X	3.24 (+87.96%) (+50.58%)	—	—
RandLM	1.81 (−3.06%) (+41.41%)	4.39 ([illegible]) (+168.04%) 2.5X	2.02 (+6.18%) (+46.29%)	5.08 ([illegible]) (+135.82%) 2.5X	2.60 (+70.73%) (+98.90%)	9.25 ([illegible]) (+302.19%) 3X

Experimental Analysis - Overall comparison

	Europarl		YahooV2		GoogleV2	
	bpg	μs × query	bpg	μs × query	bpg	μs × query
PEF-Trie	1.87	1.35	1.91	1.73	1.52	1.91
PEF-RTrie	1.28	1.64	1.38	2.15	1.31	2.30
BerkeleyLM C.	1.70 (−8.89%) (+32.90%)	2.83 (+[illegible]) (+72.70%) 2X	1.69 (−11.41%) (+22.04%)	3.48 (+[illegible]) (+61.70%) 2X	1.45 (−4.87%) (+10.83%)	4.13 (+[illegible]) (+79.76%) 2X
BerkeleyLM H.3	6.70 (+258.81%) (+[illegible]) 2.5÷	0.97 (−28.46%) (−40.85%)	7.82 (+310.38%) (+[illegible]) 3.1÷	1.13 (−34.35%) (−47.41%)	9.24 (+[illegible]) (+[illegible]) 5.5X	2.18 (+13.95%) (−5.42%)
BerkeleyLM H.50	7.96 (+[illegible]) (+521.45%) 5.2X	0.97 (−28.49%) (−40.88%)	9.37 (+[illegible]) (+576.87%) 5.8X	0.96 (−44.27%) (−55.35%)	—	—
Expgram	2.06 (+10.18%) (+60.73%)	2.80 (+[illegible]) (+70.82%) 2X	2.24 (+17.36%) (+61.68%)	9.23 (+[illegible]) (+328.87%) 3.5X	—	—
KenLM T.	2.99 (+[illegible]) (+133.56%) 2.3X	1.28 (−5.47%) (−21.84%)	3.44 (+[illegible]) (+148.52%) 2.5X	1.94 (+12.32%) (−10.01%)	—	—
Marisa	3.61 (+[illegible]) (+[illegible]) 2.8X	2.06 (+52.00%) (+25.67%)	3.81 (+[illegible]) (+[illegible]) 2.7X	3.24 (+87.96%) (+50.58%)	—	—
RandLM	1.81 (−3.06%) (+41.41%)	4.39 (+[illegible]) (+168.04%) 2.5X	2.02 (+6.18%) (+46.29%)	5.08 (+[illegible]) (+135.82%) 2.5X	2.60 (+70.73%) (+98.90%)	9.25 (+[illegible]) (+302.19%) 3X

Experimental Analysis - Overall comparison

	Europarl		YahooV2		GoogleV2	
	bpg	μs × query	bpg	μs × query	bpg	μs × query
PEF-Trie	1.87	1.35	1.91	1.73	1.52	1.91
PEF-RTrie	1.28	1.64	1.38	2.15	1.31	2.30
BerkeleyLM C.	1.70 (−8.89%) (+32.90%)	2.83 ([illegible]) (+72.70%) 2X	1.69 (−11.41%) (+22.04%)	3.48 ([illegible]) (+61.70%) 2X	1.45 (−4.87%) (+10.83%)	4.13 ([illegible]) (+79.76%) 2X
BerkeleyLM H.3	6.70 (+258.81%) ([illegible]) 2.5÷	0.97 (−28.46%) (−40.85%)	7.82 (+310.38%) ([illegible]) 3.1÷	1.13 (−34.35%) (−47.41%)	9.24 ([illegible]) ([illegible]) 5.5X	2.18 (+13.95%) (−5.42%)
BerkeleyLM H.50	7.96 ([illegible]) (+521.45%) 5.2X	0.97 (−28.49%) (−40.88%)	9.37 ([illegible]) (+576.87%) 5.8X	0.96 (−44.27%) (−55.35%)	—	—
Expgram	2.06 (+10.18%) (+60.73%)	2.80 ([illegible]) (+70.82%) 2X	2.24 (+17.36%) (+61.68%)	9.23 ([illegible]) (+328.87%) 3.5X	—	—
KenLM T.	2.99 ([illegible]) (+133.56%) 2.3X	1.28 (−5.47%) (−21.84%)	3.44 ([illegible]) (+148.52%) 2.5X	1.94 (+12.32%) (−10.01%)	—	—
Marisa	3.61 ([illegible]) ([illegible]) 2.8X	2.06 (+52.00%) (+25.67%)	3.81 ([illegible]) ([illegible]) 2.7X	3.24 (+87.96%) (+50.58%)	—	—
RandLM	1.81 (−3.06%) (+41.41%)	4.39 ([illegible]) (+168.04%) 2.5X	2.02 (+6.18%) (+46.29%)	5.08 ([illegible]) (+135.82%) 2.5X	2.60 (+70.73%) (+98.90%)	9.25 ([illegible]) (+302.19%) 3X

- Elias-Fano Tries substantially **outperform ALL** previous solutions in **both space and time**.
- As fast as the state-of-the-art (KenLM) but more than twice smaller.

Summary

Elias-Fano encodes *monotone integer sequences* in *space close to the information theoretic minimum*, while allowing *powerful search operations*, namely `Predecessor/Successor` queries and random `Access`.

Successfully applied to crucial problems, such as *inverted indexes, social graphs* and *tries* representation.

Several *optimized* software implementations are available.

References

[Fano-1971] Robert Mario Fano. *On the number of bits required to implement an associative memory*. Memorandum 61, Computer Structures Group, MIT (1971).

[Elias-1974] Peter Elias. *Efficient Storage and Retrieval by Content and Address of Static Files*. Journal of the ACM (JACM) 21, 2, 246–260 (1974).

[Jacobson-1989] Guy Jacobson. *Succinct Static Data Structures*. Ph.D. Thesis, Carnegie Mellon University (1989).

[Clark-1996] David Clark. *Compact Pat Trees*. Ph.D. Thesis, University of Waterloo (1996).

[Moffat and Stuiver-2000] Alistair Moffat and Lang Stuiver. *Binary Interpolative Coding for Effective Index Compression.* Information Retrieval Journal 3, 1, 25–47 (2000).

[Anh and Moffat-2005] Vo Ngoc Anh and Alistair Moffat. *Inverted Index Compression Using Word-Aligned Binary Codes.* Information Retrieval Journal 8, 1, 151–166 (2005).

[Salomon-2007] David Salomon. *Variable-length Codes for Data Compression.* Springer (2007).

[Vigna-2008] Sebastiano Vigna. *Broadword implementation of rank/select queries*. In Workshop in Experimental Algorithms (WEA), 154-168 (2008).

[Yan *et al.*-2009] Hao Yan, Shuai Ding, and Torsten Suel. *Inverted index compression and query processing with optimized document ordering.* In Proceedings of the 18th International Conference on World Wide Web (WWW). 401–410 (2009).

[Anh and Moffat-2010] Vo Ngoc Anh and Alistair Moffat. *Index compression using 64-bit words.* In Software: Practice and Experience 40, 2, 131–147 (2010).

[Zukowski *et al.*-2010] Marcin Zukowski, Sandor Hèman, Niels Nes, and Peter Boncz. *Super-Scalar RAM-CPU Cache Compression.* In Proceedings of the 22nd International Conference on Data Engineering (ICDE). 59–70 (2006).

[Stepanov *et al.*-2011] Alexander Stepanov, Anil Gangolli, Daniel Rose, Ryan Ernst, and Paramjit Oberoi. *SIMD-based decoding of posting lists.* In Proceedings of the 20th ACM International Conference on Information and Knowledge Management (CIKM). 317–326 (2011).

[Zhou *et al.*-2013] Dong Zhou, David Andersen, Michael Kaminsky. *Space-Efficient, High-Performance Rank and Select Structures on Uncompressed Bit Sequences*. In Proceedings of the 12-nd International Symposium on Experimental Algorithms (SEA), 151-163 (2013).

[Vigna-2013] Sebastiano Vigna. *Quasi-succinct indices*. In Proceedings of the 6-th ACM International Conference on Web Search and Data Mining (WSDM), 83-92 (2013).

[Curtiss *et al.*-2013] Michael Curtiss *et al. Unicorn: A System for Searching the Social Graph*. In Proceedings of the Very Large Database Endowment (PVLDB), 1150-1161 (2013).

[Ottaviano and Venturini-2014] Giuseppe Ottaviano, Rossano Venturini. *Partitioned Elias-Fano Indexes*. In Proceedings of the 37-th ACM International Conference on Research and Development in Information Retrieval (SIGIR), 273-282 (2014).

Thanks for your attention, time, patience!

Any questions?

successor **example**

$S = [3, 4, 7, 13, 14, 15, 21, 43]$

1 2 3 4 5 6 7 8

H = 1110111010001000

L = 011100111101110111101011

`successor` example

S = [3, 4, 7, 13, 14, 15, 21, 43]

1 2 3 4 5 6 7 8

`successor(12)` = ?

H = 1110111010001000

L = 011100111101110111101011

`successor` example

S = [3, 4, 7, 13, 14, 15, 21, 43]

1 2 3 4 5 6 7 8

`successor(12)` = ?

001100

H = 1110111010001000

L = 011100111101110111101011

successor **example**

S = [3, 4, 7, 13, 14, 15, 21, 43]

1 2 3 4 5 6 7 8

successor(12) = ?

h_{12} = 001100

H = 1110111010001000

L = 01110011110111011111101011

successor example

$S = [3, 4, 7, 13, 14, 15, 21, 43]$

1 2 3 4 5 6 7 8

successor(12) = ?

$h_{12} = 001100$

$p_1 = select_0(h_x) - h_x$

$p_2 = select_0(h_x + 1) - h_x - 1$

H = 1110111010001000

L = 011100111101110111101011

successor example

$S = [3, 4, 7, 13, 14, 15, 21, 43]$

1 2 3 4 5 6 7 8

$\texttt{successor(12)} = ?$

$h_{12} = 001100$

$p_1 = \texttt{select}_0(h_x) - h_x$

$p_2 = \texttt{select}_0(h_x+1) - h_x - 1$

$H = \cancel{1110}0111010001000$

$L = 011100111011101111101011$

successor example

$S = [3, 4, 7, 13, 14, 15, 21, 43]$

1 2 3 4 5 6 7 8

`successor(12)` = ?

h_{12} = 001100

```
p1 = select0(hx)-hx
p2 = select0(hx+1)-hx-1
```

H = ~~1110~~1110~~10001000~~

L = 011100111101110111101011

successor **example**

$$S = [3, 4, 7, 13, 14, 15, 21, 43]$$

1 2 3 4 5 6 7 8

$\mathtt{successor}(12) = ?$

$h_{12} = 001100$

$p_1 = \mathtt{select}_0(h_x) - h_x$

$p_2 = \mathtt{select}_0(h_x + 1) - h_x - 1$

$H = \cancel{1110}1110\cancel{10001000}$

$L = \cancel{011100111}101110111\cancel{101011}$

p_1 p_2

successor **example**

$S = [3, 4, 7, 13, 14, 15, 21, 43]$

1 2 3 4 5 6 7 8

$\mathtt{successor}(12) = ?$

$h_{12} = 001100$

$p_1 = \mathtt{select}_0(h_x) - h_x$

$p_2 = \mathtt{select}_0(h_x + 1) - h_x - 1$

H = ~~1110~~1110~~10001000~~

L = ~~011100111~~101110111~~101011~~

p_1 p_2

binary search
in $[p_1, p_2)$

successor **example**

$S = [3, 4, 7, 13, 14, 15, 21, 43]$

1 2 3 4 5 6 7 8

$\texttt{successor}(12) = 13$

$h_{12} = 001100$

$p_1 = \texttt{select}_0(h_x) - h_x$
$p_2 = \texttt{select}_0(h_x+1) - h_x - 1$

$H = \cancel{1110}1110\cancel{10001000}$

$L = \cancel{011100111}101110111\cancel{101011}$

p_1 p_2

→ *binary search* in $[p_1,p_2)$

successor **example**

$S = [3, 4, 7, 13, 14, 15, 21, 43]$

1 2 3 4 5 6 7 8

successor(12) = 13

$h_{12} = 001100$

$p_1 = \mathrm{select}_0(h_x) - h_x$

$p_2 = \mathrm{select}_0(h_x+1) - h_x - 1$

H = ~~1110~~1110~~10001000~~

L = ~~011100111~~101110111~~101011~~

p_1 p_2

→ *binary search* in $[p_1, p_2)$

Complexity: $O\left(\log \frac{u}{n}\right)$