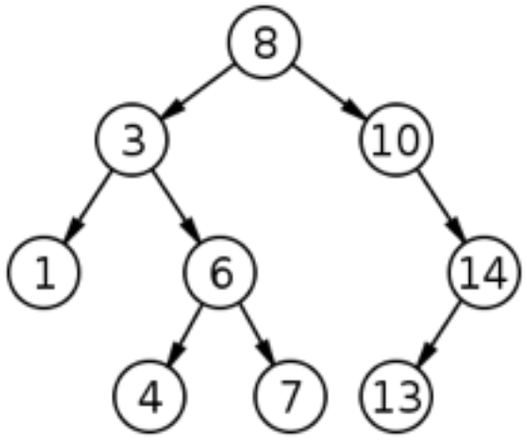
Succinct Data Structures in Information Retrieval

Rossano Venturini

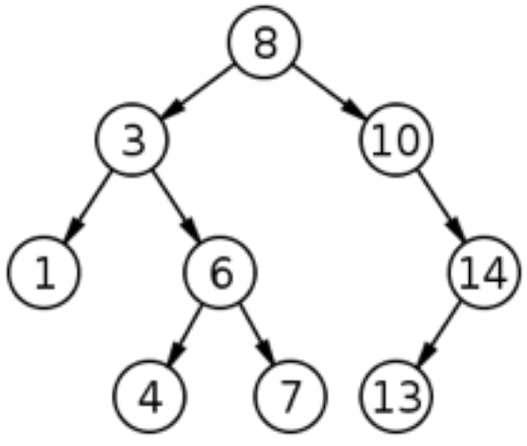
University of Pisa
ISTI-CNR, Pisa

Data Structures



+ Time efficiency
- Add extra data

Data Structures

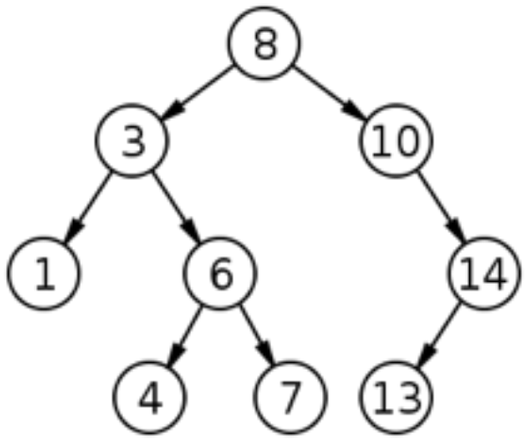


+ Time efficiency
- Add extra data

Data Compression

+ Space efficiency
- Slow access to data

Data Structures



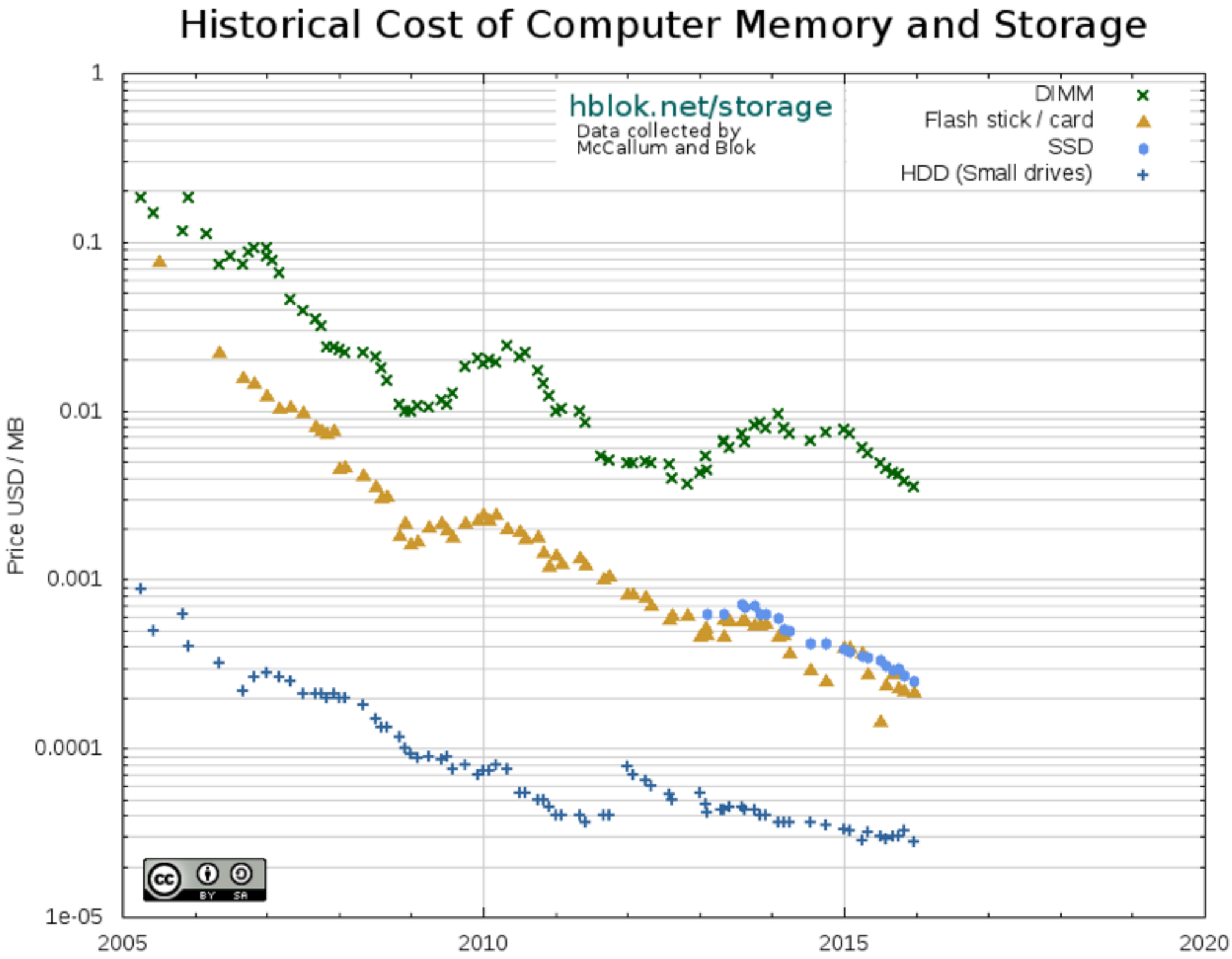
+ Time efficiency
- Add extra data

Data Compression

+ Space efficiency
- Slow access to data

Why?

Why?


Historical Cost of Computer Memory and Storage
hblok.net/storage
Data collected by
McCallum and Blok
DIMM
Flash stick / card
SSD
HDD (Small drives)
Price USD / MB
1
0.1
0.01
0.001
0.0001
1e-05
2005
2010
2015
2020
CC BY SA

Why?

Less space implies faster and cheaper algorithms

Why?

Less space implies faster and cheaper algorithms

VS

Why?

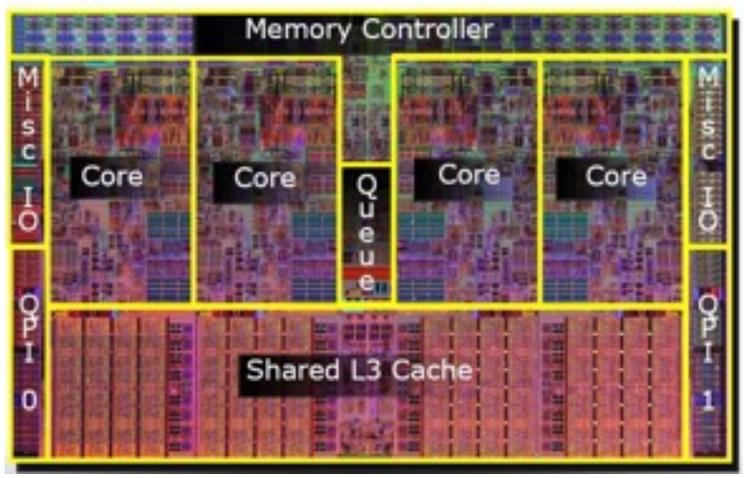
Less space implies faster and cheaper algorithms

VS

VS

Why?

Less space implies faster and cheaper algorithms



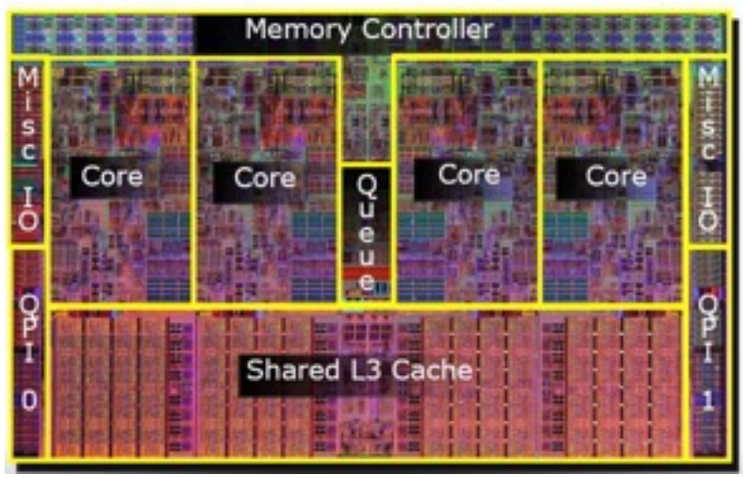
VS

VS

VS

Why?

Less space implies faster and cheaper algorithms
for any dataset size



VS

VS

VS

Active field of research

Integers

FOCS, 1989
FOCS, 1997
SODA, 2002
FOCS, 2008
SODA, 2010

Active field of research

Integers

FOCS, 1989
FOCS, 1997
SODA, 2002
FOCS, 2008
SODA, 2010

Text

FOCS, 2000
SODA, 2000
SODA, 2001
SODA, 2007
PODS, 2011

Active field of research

Integers

FOCS, 1989
FOCS, 1997
SODA, 2002
FOCS, 2008
SODA, 2010

Text

FOCS, 2000
SODA, 2000
SODA, 2001
SODA, 2007
PODS, 2011

Trees

FOCS, 1989
FOCS, 1997
SODA, 2002
SODA, 2007
SODA, 2010

Active field of research

Integers	Text	Trees	Graphs
FOCS, 1989	FOCS, 2000	FOCS, 1989	FOCS, 1997
FOCS, 1997	SODA, 2000	FOCS, 1997	DCC, 2001
SODA, 2002	SODA, 2001	SODA, 2002	WWW, 2004
FOCS, 2008	SODA, 2007	SODA, 2007	ESA, 2008
SODA, 2010	PODS, 2011	SODA, 2010	FOCS, 2009

Active field of research

Integers

FOCS, 1989
FOCS, 1997
SODA, 2002
FOCS, 2008
SODA, 2010

Text

FOCS, 2000
SODA, 2000
SODA, 2001
SODA, 2007
PODS, 2011

Trees

FOCS, 1989
FOCS, 1997
SODA, 2002
SODA, 2007
SODA, 2010

Graphs

FOCS, 1997
DCC, 2001
WWW, 2004
ESA, 2008
FOCS, 2009

Labeled Trees e.g. XML

FOCS, 2005
WWW, 2006

Functions

ICALP, 2003
ICALP, 2004
SODA, 2004
ICALP, 2008
ESA, 2009

Point Sets

SODA, 2003
TALG, 2007
WADS, 2009
SODA, 2011
SOCG, 2011

Images

SPIRE, 2008

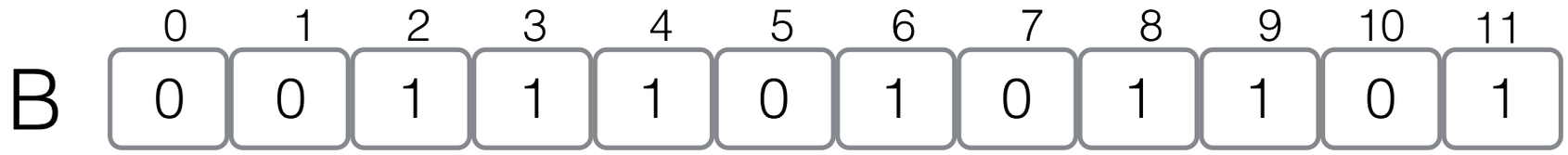
Active field of research

Integers	Text	Trees	Graphs
FOCS, 1989	FOCS, 2000	FOCS, 1989	FOCS, 1997
FOCS, 1997	SODA, 2000	FOCS, 1997	DCC, 2001
SODA, 2002	SODA, 2001	SODA, 2002	WWW, 2004
FOCS, 2008	SODA, 2007	SODA, 2007	ESA, 2008
SODA, 2010	PODS, 2011	SODA, 2010	FOCS, 2009

These results are obtained via a (often non trivial) combination of basic succinct data structures

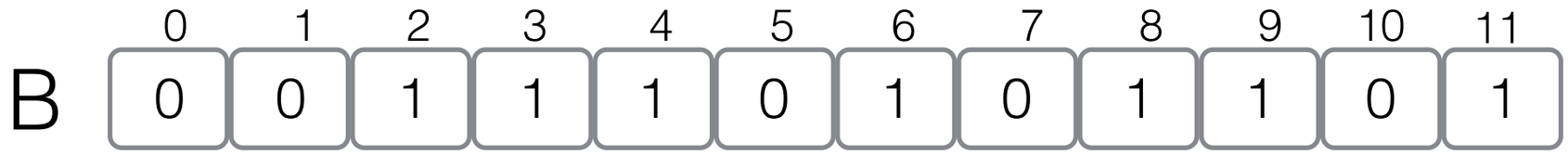
Labeled Trees e.g. XML	Functions	Point Sets	Images
FOCS, 2005	ICALP, 2003	SODA, 2003	SPIRE, 2008
WWW, 2006	ICALP, 2004	TALG, 2007	
	SODA, 2004	WADS, 2009	
	ICALP, 2008	SODA, 2011	
	ESA, 2009	SOCG, 2011	

Rank/Select queries



Rank/Select queries

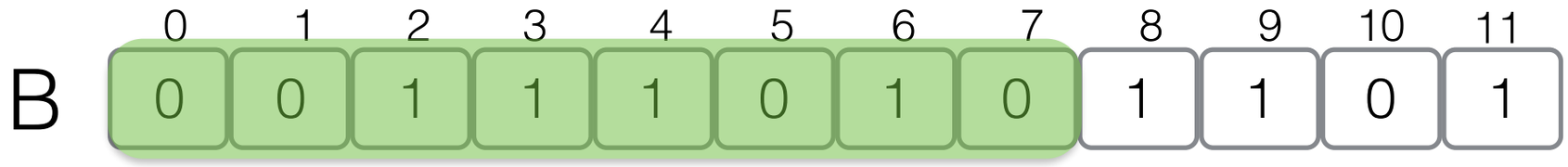
$Rank_0(j)$ = # of 0 in B[0,j]



Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_0(7) = 4$

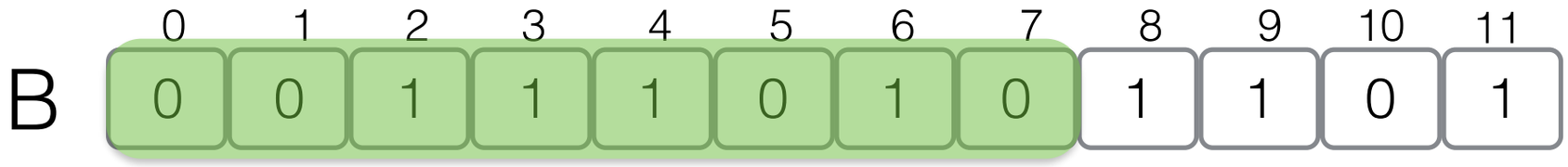


Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

$Rank_0(7) = 4$



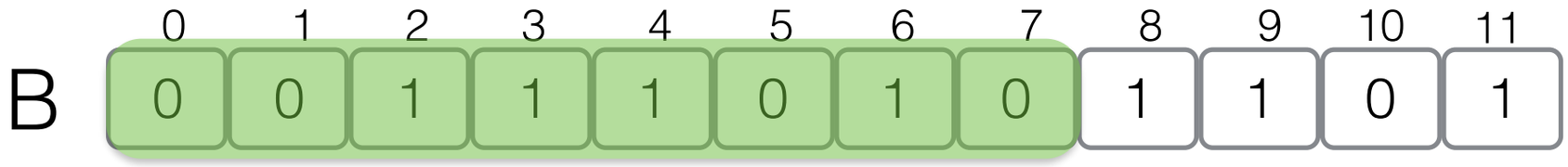
Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

$Rank_0(7) = 4$

$Rank_1(7) = 8 - Rank_0(7) = 4$



Rank/Select queries

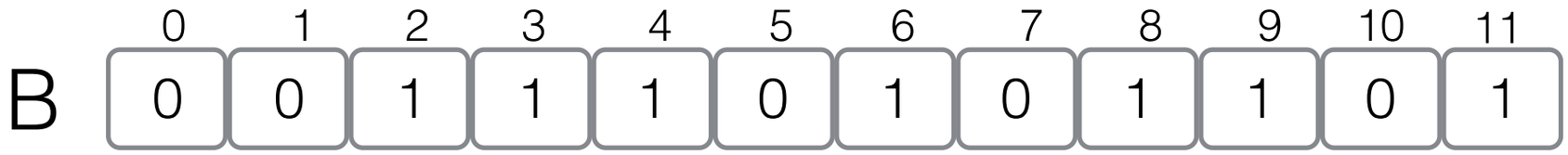
$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

$Rank_0(7) = 4$

$Rank_1(7) = 8 - Rank_0(7) = 4$

$Select_0(j)$ = position of the j-th 0 in B



Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

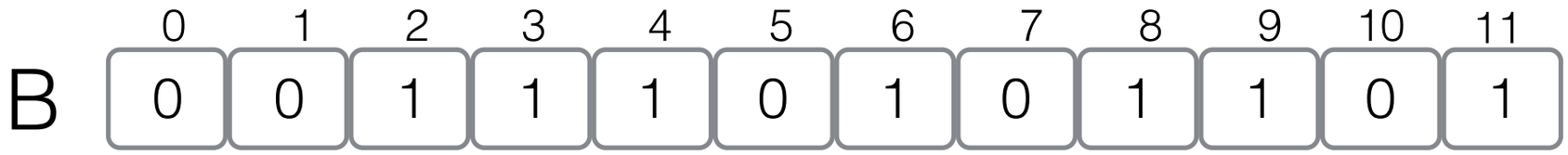
$Rank_1(j)$ = # of 1 in B[0,j]

$Rank_0(7) = 4$

$Rank_1(7) = 8 - Rank_0(7) = 4$

$Select_0(j)$ = position of the j-th 0 in B

$Select_1(j)$ = position of the j-th 1 in B



Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

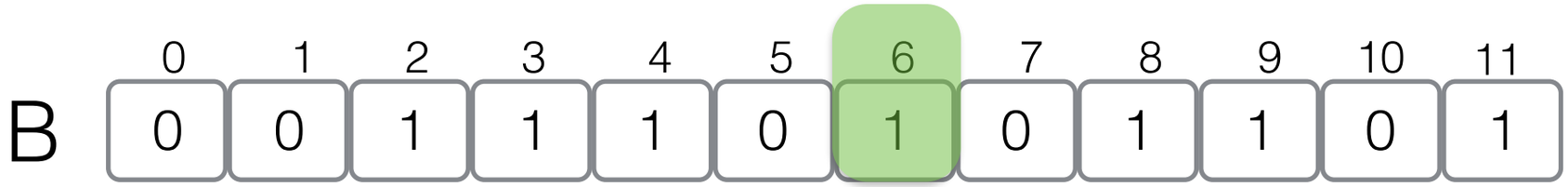
$Rank_0(7) = 4$

$Rank_1(7) = 8 - Rank_0(7) = 4$

$Select_0(j)$ = position of the j-th 0 in B

$Select_1(j)$ = position of the j-th 1 in B

$Select_1(4) = 6$



Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

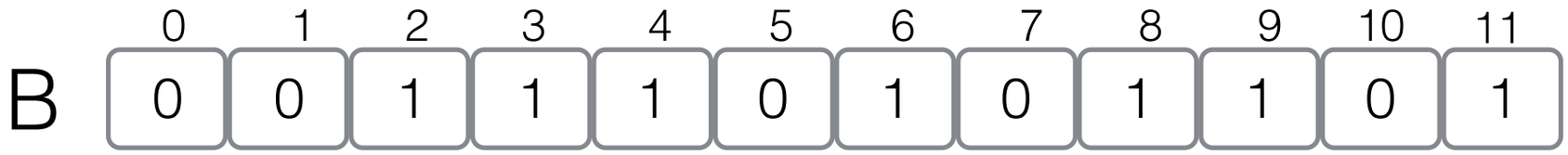
$Rank_1(j)$ = # of 1 in B[0,j]

$Select_0(j)$ = position of the j-th 0 in B

$Select_1(j)$ = position of the j-th 1 in B

Space: n + o(n) bits
Query time: O(1)

$_1(4) = 6$



Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

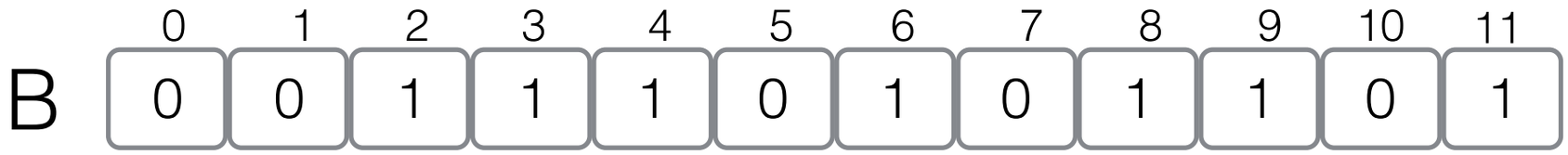
$Rank_1(j)$ = # of 1 in B[0,j]

$Select_0(j)$ = position of the j-th 0 in B

$Select_1(j)$ = position of the j-th 1 in B

Space: n + o(n) bits
Query time: O(1)

$_1(4) = 6$



Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

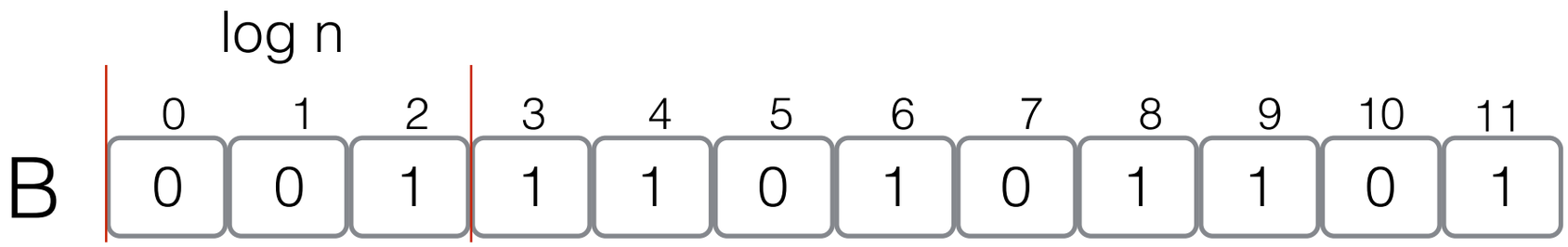
$Rank_1(j)$ = # of 1 in B[0,j]

$Select_0(j)$ = position of the j-th 0 in B

$Select_1(j)$ = position of the j-th 1 in B

Space: n + o(n) bits
Query time: O(1)

$_1(4) = 6$



Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

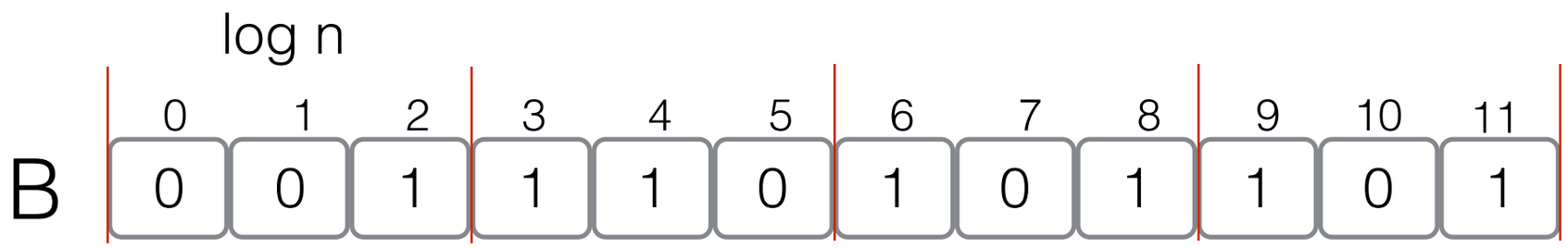
$Rank_1(j)$ = # of 1 in B[0,j]

$Select_0(j)$ = position of the j-th 0 in B

$Select_1(j)$ = position of the j-th 1 in B

Space: n + o(n) bits
Query time: O(1)

$_1(4) = 6$



Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

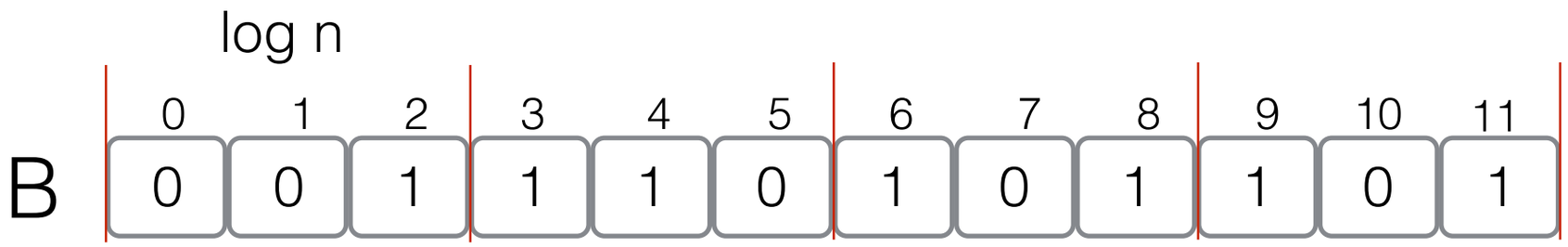
$Select_0(j)$ = position of the j-th 0 in B

$Select_1(j)$ = position of the j-th 1 in B

Space: n + o(n) bits
Query time: O(1)

$_1(4) = 6$

B’



Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

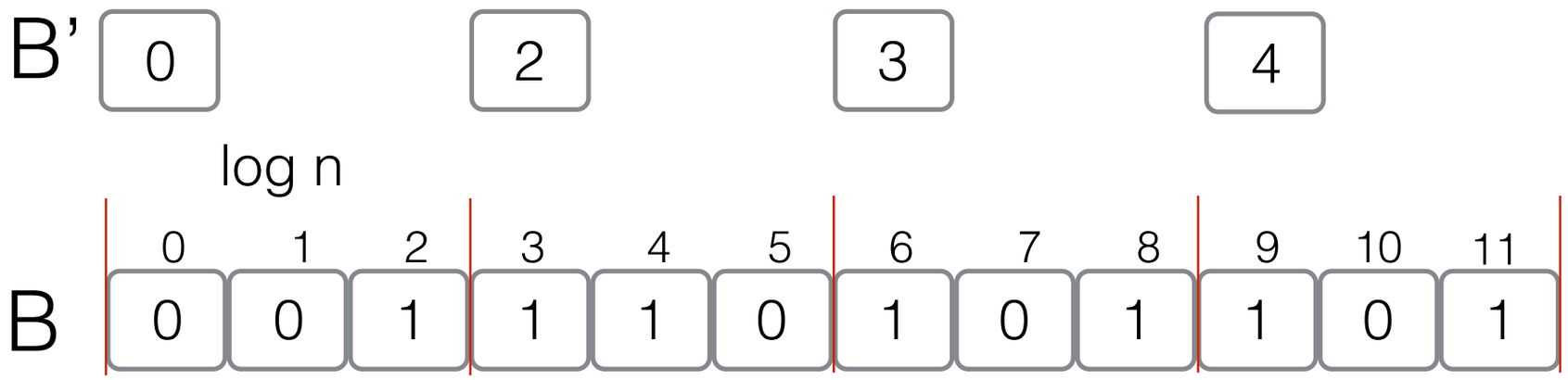
$Rank_1(j)$ = # of 1 in B[0,j]

$Select_0(j)$ = position of the j-th 0 in B

$Select_1(j)$ = position of the j-th 1 in B

Space: n + o(n) bits
Query time: O(1)

$_1(4) = 6$



Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

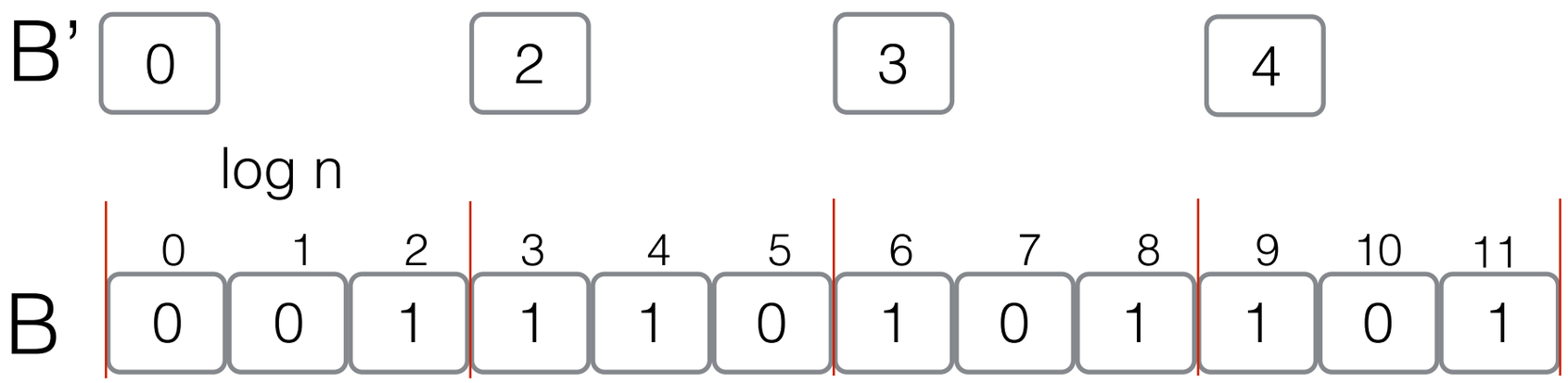
$Select_0(j)$ = position of the j-th 0 in B

$Select_1(j)$ = position of the j-th 1 in B

Space: n + o(n) bits
Query time: O(1)

$_1(4) = 6$

$Rank_0(7) =$



Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

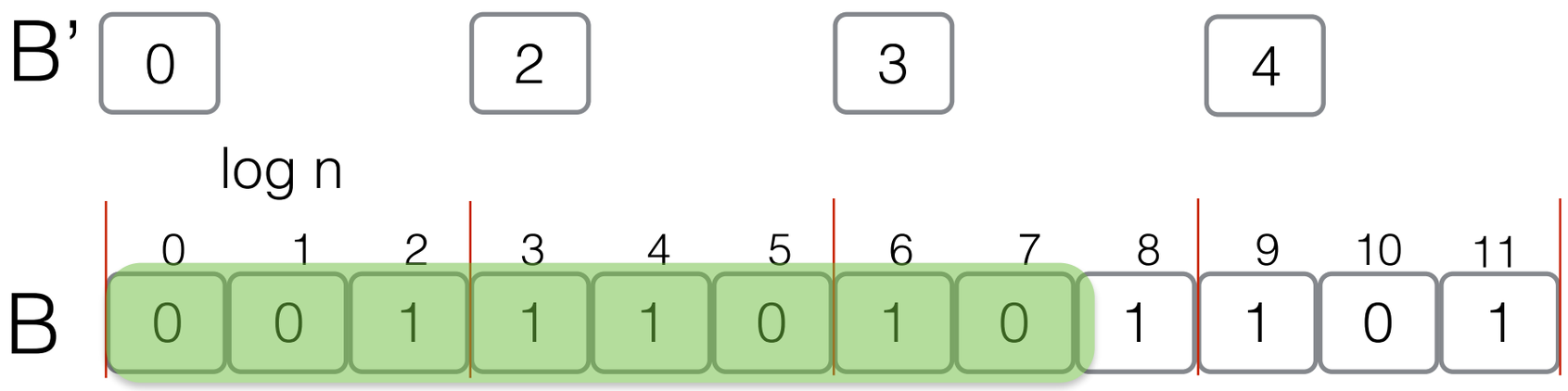
$Select_0(j)$ = position of the j-th 0 in B

$Select_1(j)$ = position of the j-th 1 in B

Space: n + o(n) bits
Query time: O(1)

$_1(4) = 6$

$Rank_0(7) =$



Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

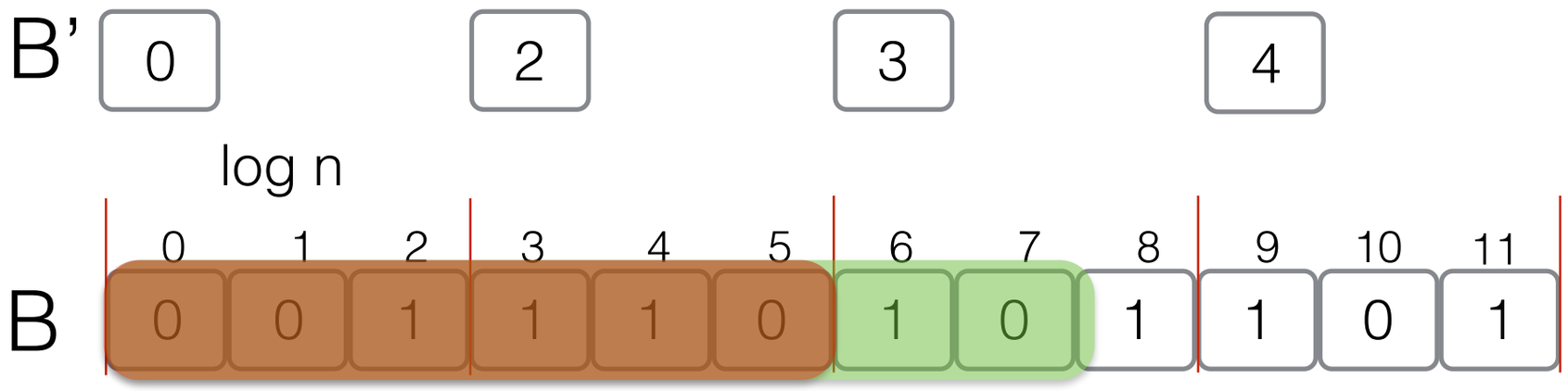
$Select_0(j)$ = position of the j-th 0 in B

$Select_1(j)$ = position of the j-th 1 in B

Space: n + o(n) bits
Query time: O(1)

$_1(4) = 6$

$Rank_0(7) =$



Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

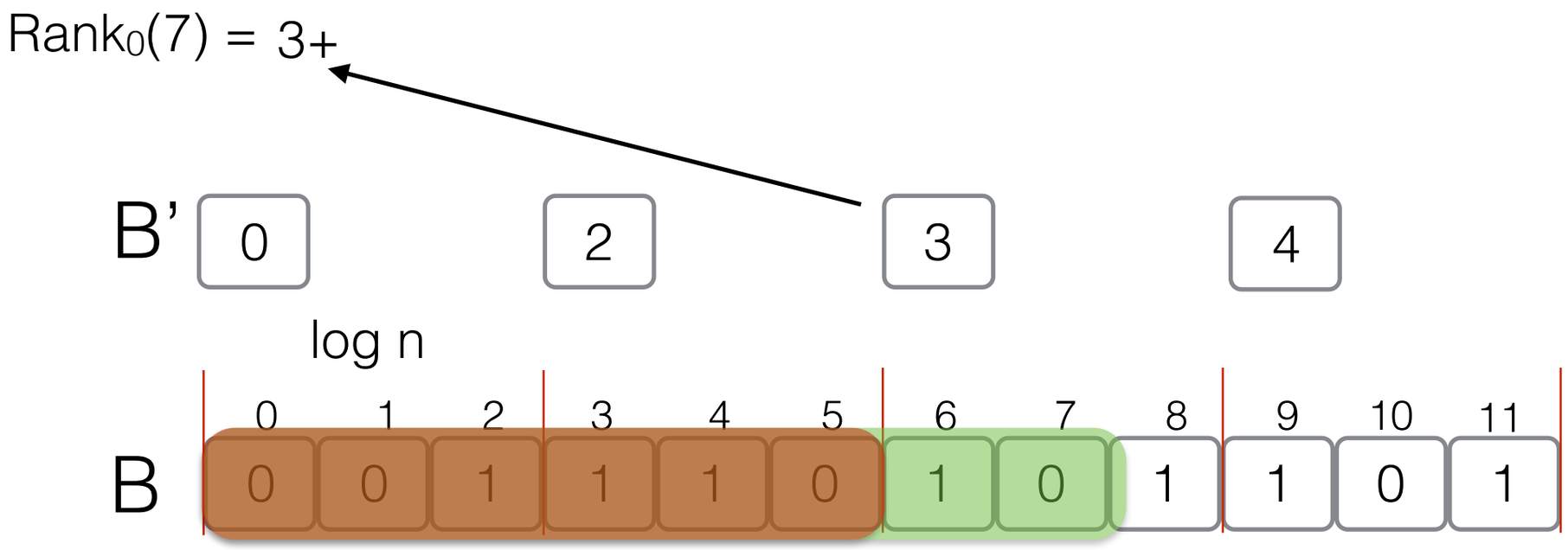
$Rank_1(j)$ = # of 1 in B[0,j]

$Select_0(j)$ = position of the j-th 0 in B

$Select_1(j)$ = position of the j-th 1 in B

Space: n + o(n) bits
Query time: O(1)

$_1(4) = 6$



Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

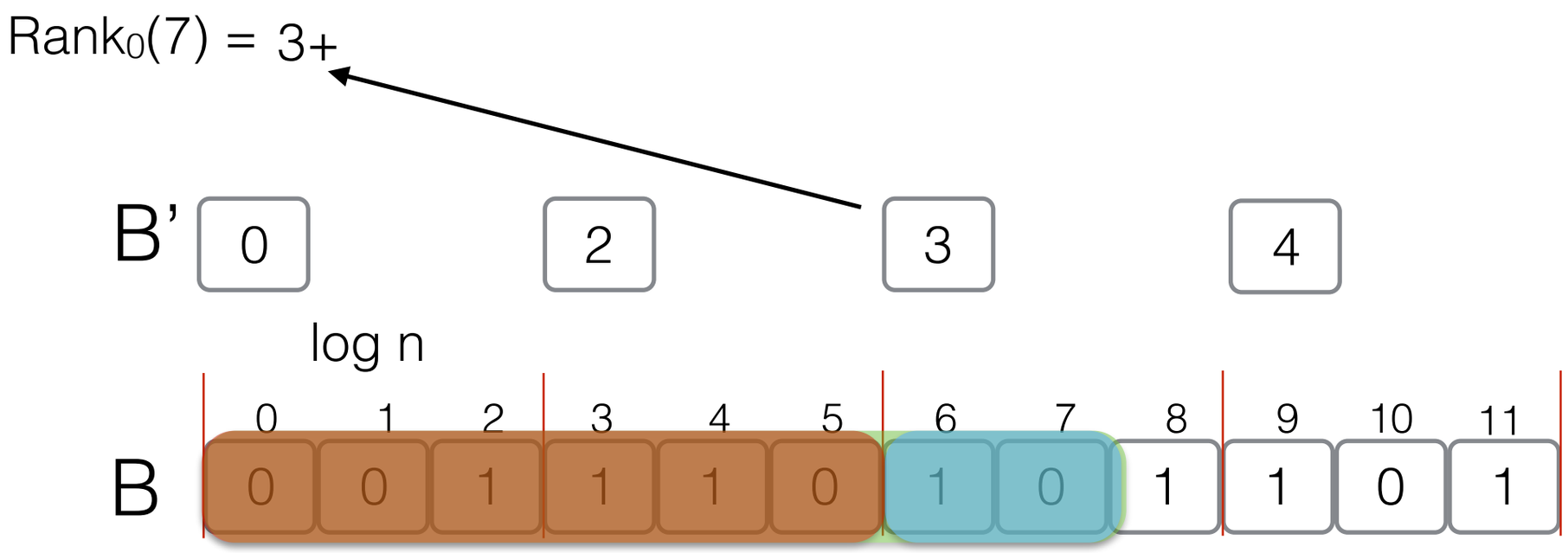
$Rank_1(j)$ = # of 1 in B[0,j]

$Select_0(j)$ = position of the j-th 0 in B

$Select_1(j)$ = position of the j-th 1 in B

Space: n + o(n) bits
Query time: O(1)

$_1(4) = 6$



Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

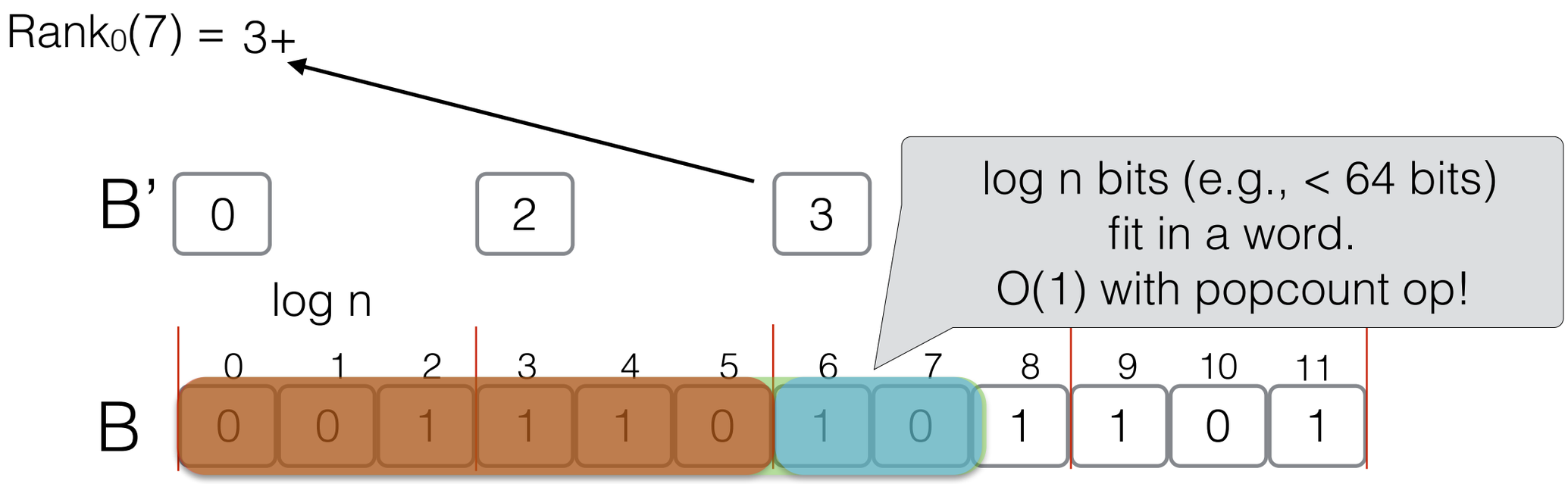
$Rank_1(j)$ = # of 1 in B[0,j]

$Select_0(j)$ = position of the j-th 0 in B

$Select_1(j)$ = position of the j-th 1 in B

Space: n + o(n) bits
Query time: O(1)

$_1(4) = 6$



Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

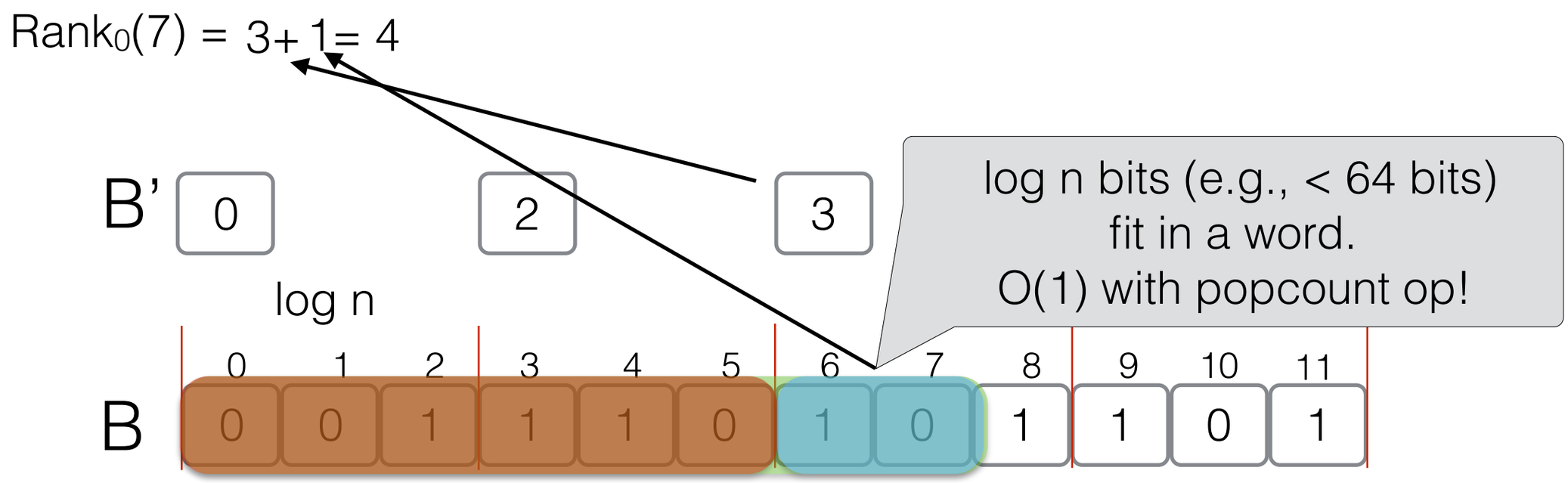
$Rank_1(j)$ = # of 1 in B[0,j]

$Select_0(j)$ = position of the j-th 0 in B

$Select_1(j)$ = position of the j-th 1 in B

Space: n + o(n) bits
Query time: O(1)

$_1(4) = 6$



Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

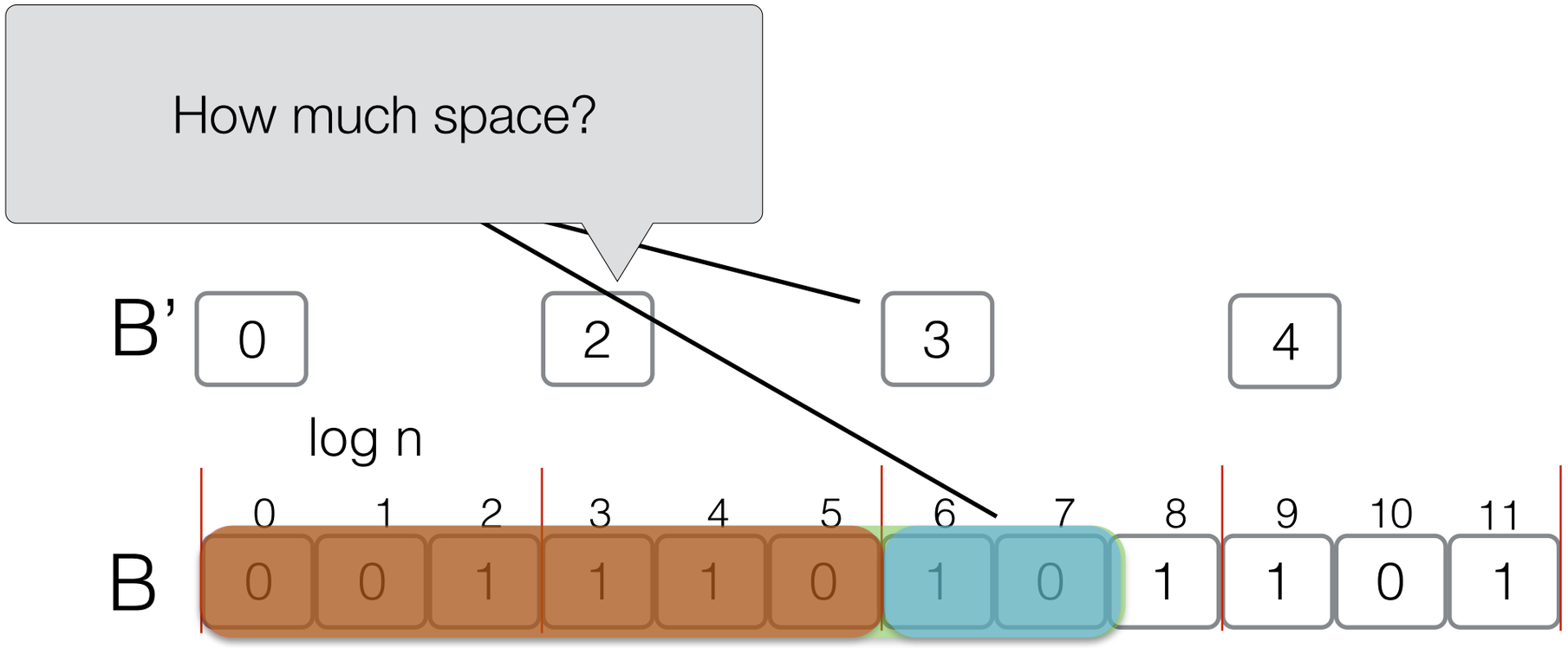
$Rank_1(j)$ = # of 1 in B[0,j]

$Select_0(j)$ = position of the j-th 0 in B

$Select_1(j)$ = position of the j-th 1 in B

Space: n + o(n) bits
Query time: O(1)

$_1(4) = 6$



Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

$Select_0(j)$ = position of the j-th 0 in B

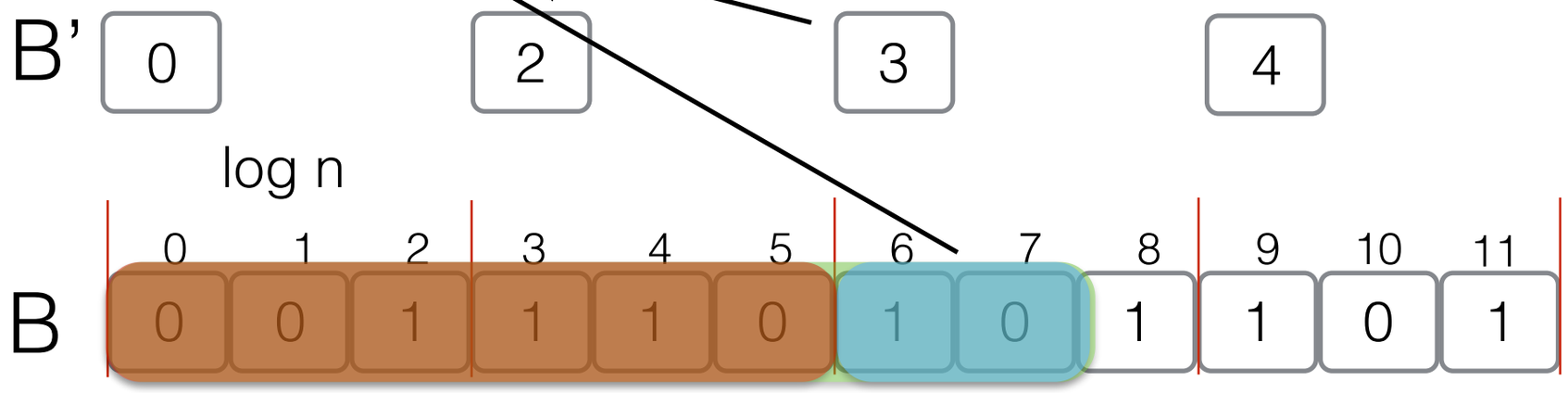
$Select_1(j)$ = position of the j-th 1 in B

Space: n + o(n) bits
Query time: O(1)

$_1(4) = 6$

How much space?

O(n/log n) entries,
each uses O(log n) bits
⇒ O(n) bits :-(



Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

$Rank_1(j)$ = # of 1 in B[0,j]

$Select_0(j)$ = position of the j-th 0 in B

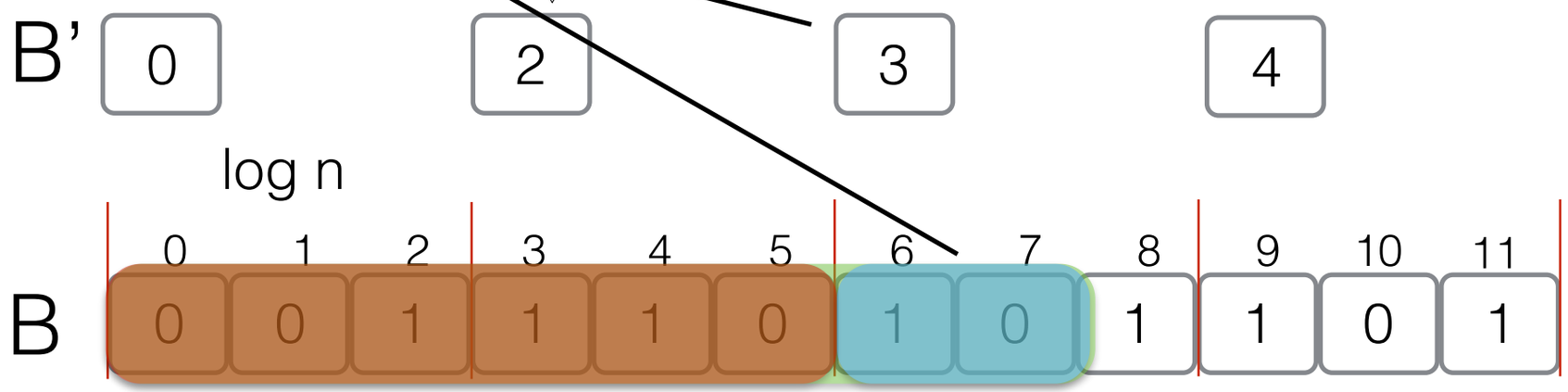
$Select_1(j)$ = position of the j-th 1 in B

Space: n + o(n) bits
Query time: O(1)

Space: O(n) + o(n) bits
Query time: O(1)

How much space?

O(n/log n) entries,
each uses O(log n) bits
⇒ O(n) bits :-(



Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

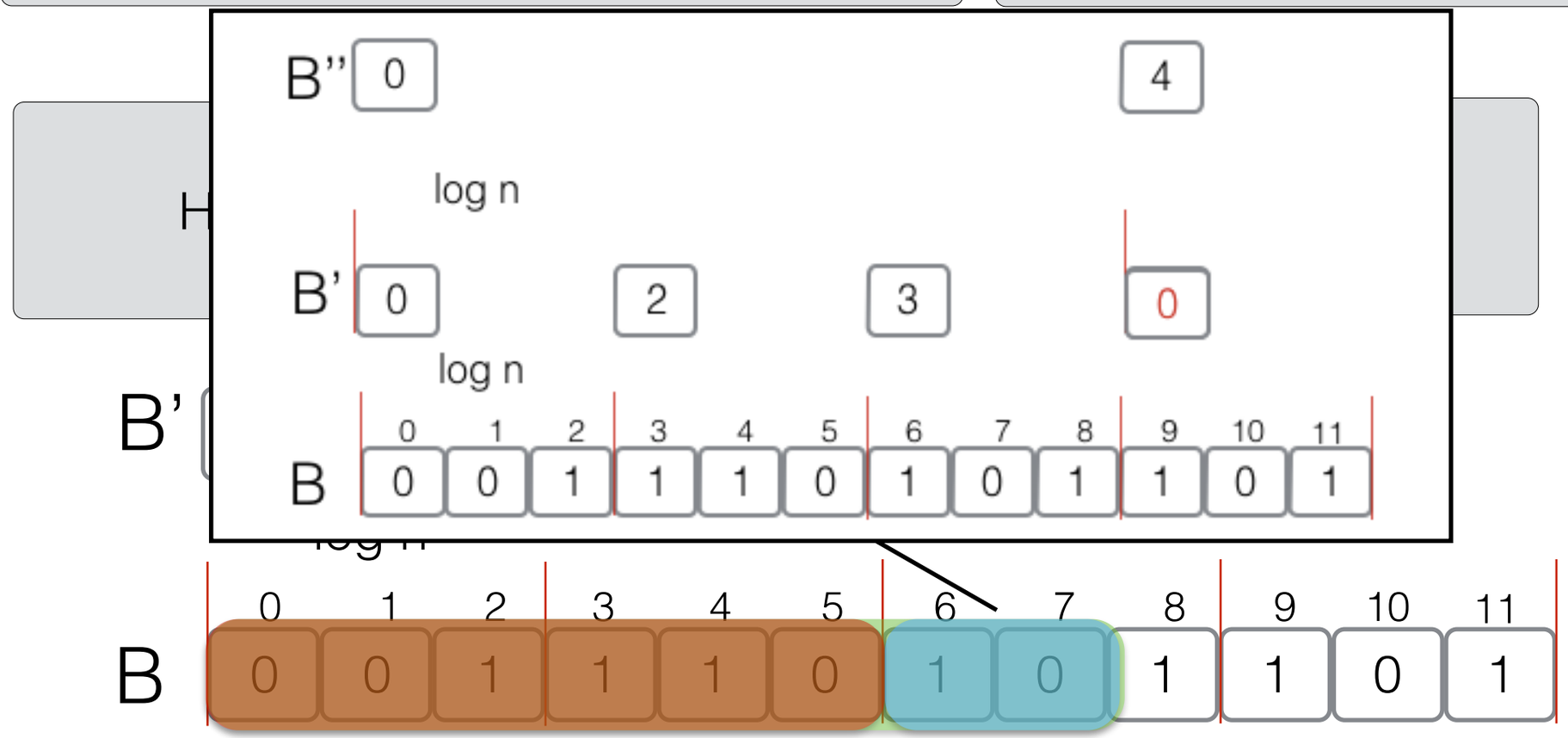
$Rank_1(j)$ = # of 1 in B[0,j]

$Select_0(j)$ = position of the j-th 0 in B

$Select_1(j)$ = position of the j-th 1 in B

Space: n + o(n) bits
Query time: O(1)

Space: O(n) + o(n) bits
Query time: O(1)



Rank/Select queries

$Rank_0(j)$ = # of 0 in B[0,j]

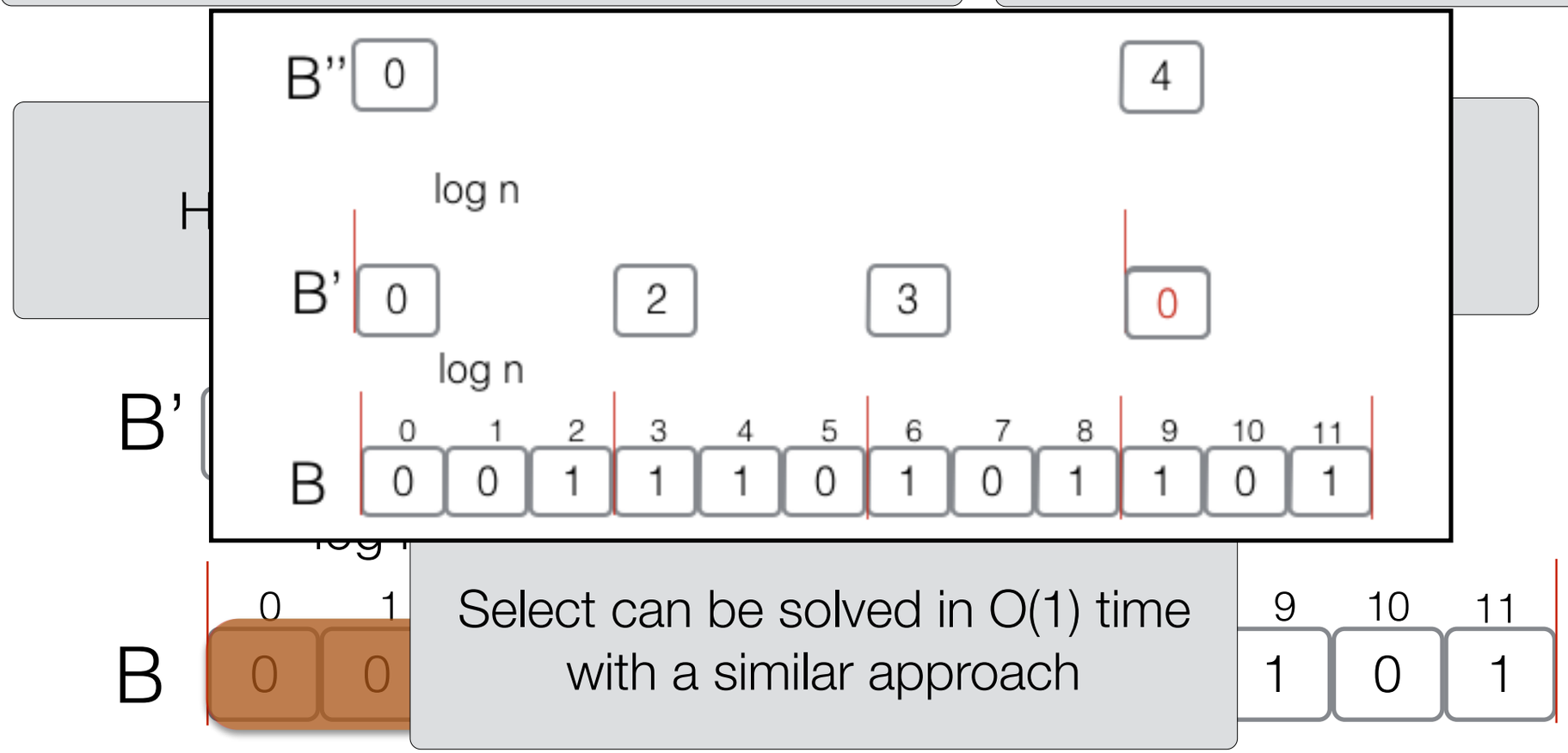
$Rank_1(j)$ = # of 1 in B[0,j]

$Select_0(j)$ = position of the j-th 0 in B

$Select_1(j)$ = position of the j-th 1 in B

Space: n + o(n) bits
Query time: O(1)

Space: O(n) + o(n) bits
Query time: O(1)



Select can be solved in O(1) time with a similar approach

Elias-Fano representation

Given a sequence S of n positive and increasing integers up to m

Space: n log (m/n) + 2n bits
Access(i) in O(1) and NextGEQ(x) in O(log (m/n))

Elias-Fano representation

[Elias-Fano 1975]

Given a sequence S of n positive and increasing integers up to m

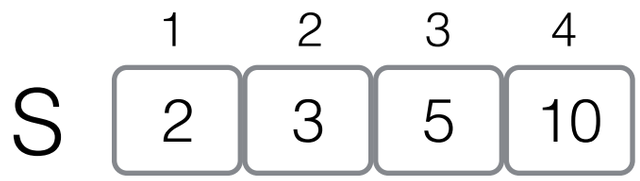
Space: n log (m/n) + 2n bits
Access(i) in O(1) and NextGEQ(x) in O(log (m/n))

Elias-Fano representation

[Elias-Fano 1975]

Given a sequence S of n positive and increasing integers up to m

Space: n log (m/n) + 2n bits
Access(i) in O(1) and NextGEQ(x) in O(log (m/n))



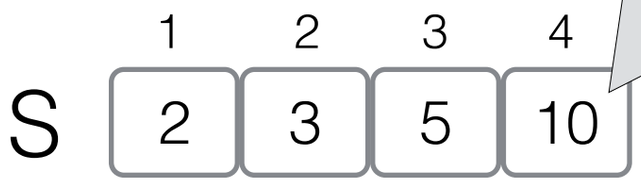
Elias-Fano representation

[Elias-Fano 1975]

Given a sequence S of n positive and increasing integers up to m

Space: n log (m/n) + 2n bits
Access(i) in O(1) and NextGEQ(x) in O(log (m/n))

Binary search trees require
O(n log m) bits :-(
Can we do better?



Elias-Fano representation

[Elias-Fano 1975]

Given a sequence S of n positive and increasing integers up to m

Space: n log (m/n) + 2n bits
Access(i) in O(1) and NextGEQ(x) in O(log (m/n))

Binary search trees require
O(n log m) bits :-(
Can we do better?

	1	2	3	4
S	2	3	5	10

Represent S as a binary vector

B

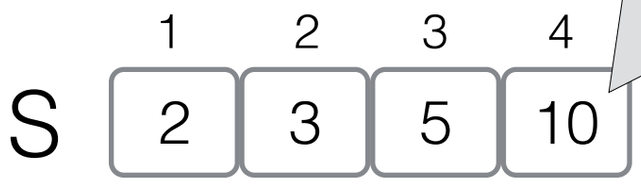
Elias-Fano representation

[Elias-Fano 1975]

Given a sequence S of n positive and increasing integers up to m

Space: n log (m/n) + 2n bits
Access(i) in O(1) and NextGEQ(x) in O(log (m/n))

Binary search trees require
O(n log m) bits :-(
Can we do better?



Represent S as a binary vector

The i-th value of S is $Select_1(i)$

B

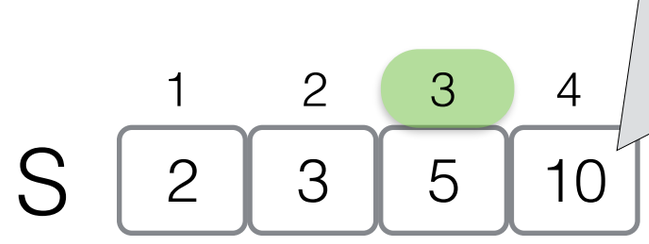
Elias-Fano representation

[Elias-Fano 1975]

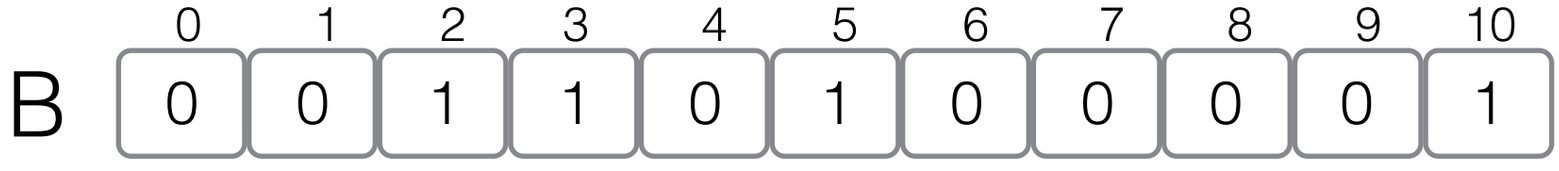
Given a sequence S of n positive and increasing integers up to m

Space: n log (m/n) + 2n bits
Access(i) in O(1) and NextGEQ(x) in O(log (m/n))

Binary search trees require
O(n log m) bits :-(
Can we do better?



Represent S as a binary vector The i-th value of S is $Select_1(i)$



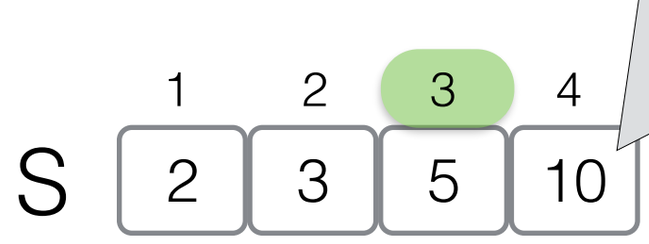
Elias-Fano representation

[Elias-Fano 1975]

Given a sequence S of n positive and increasing integers up to m

Space: n log (m/n) + 2n bits
Access(i) in O(1) and NextGEQ(x) in O(log (m/n))

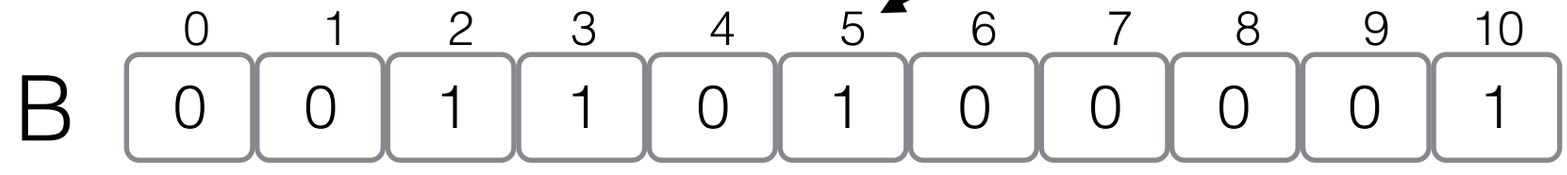
Binary search trees require O(n log m) bits :-(
Can we do better?



Represent S as a binary vector

The i-th value of S is $Select_1(i)$

$Select_1(3)=5$



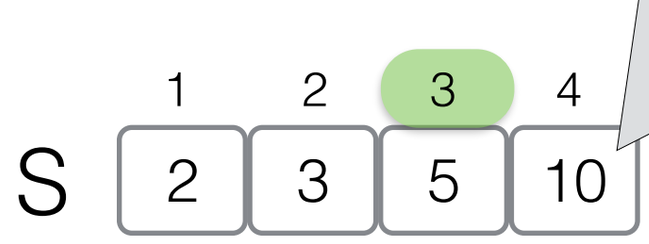
Elias-Fano representation

[Elias-Fano 1975]

Given a sequence S of n positive and increasing integers up to m

Space: n log (m/n) + 2n bits
Access(i) in O(1) and NextGEQ(x) in O(log (m/n))

Binary search trees require
O(n log m) bits :-(
Can we do better?

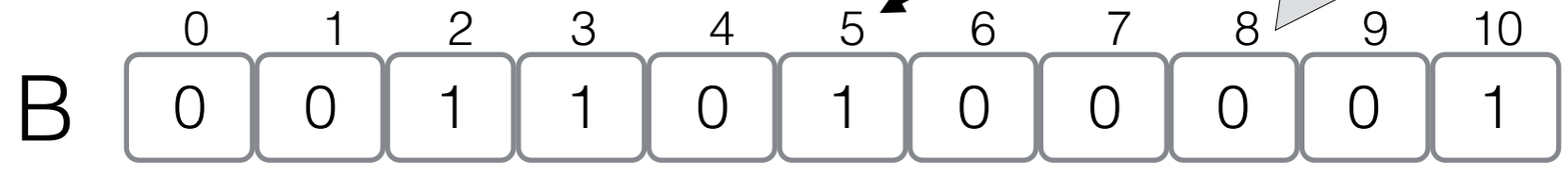


Represent S as a binary vector

The i-th value of

$Select_1(3)=5$

Binary vector requires
m + o(m) bits
Can we do better?



Elias-Fano representation

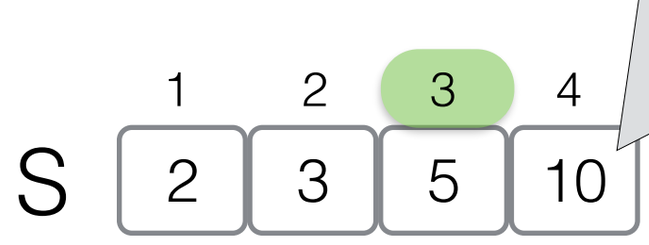
[Elias-Fano 1975]

Given a sequence S of n integers up to m

Average distance between consecutive integers in S

Space: n log (m/n) + 2n bits
Access(i) in O(1) and NextGEQ(x) in O(log (m/n))

Binary search trees require O(n log m) bits :-(
Can we do better?

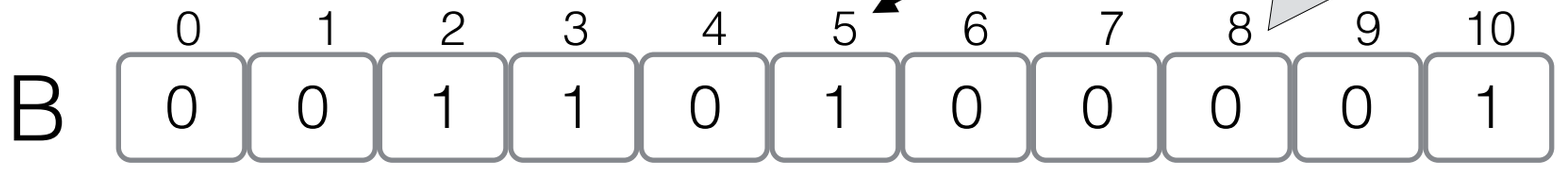


Represent S as a binary vector

The i-th value of

$Select_1(3)=5$

Binary vector requires m + o(m) bits
Can we do better?



Elias-Fano representation

[Elias-Fano 1975]

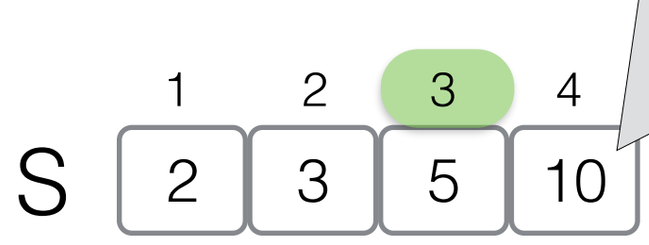
Given a sequence S of n integers up to m

Average distance between consecutive integers in S

Space: n log (m/n) + 2n bits
Access(i) in O(1) and NextGEQ(x) in O(log (m/n))

Largest integer in S

Binary search trees require O(n log m) bits :-(
Can we do better?

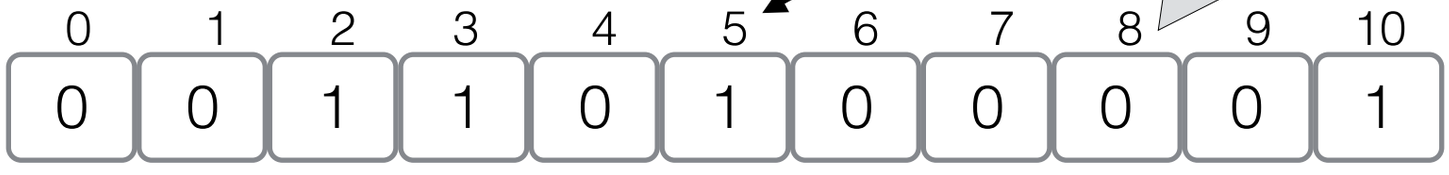


Represent S as a binary vector

The i-th value of

$Select_1(3)=5$

Binary vector requires m + o(m) bits
Can we do better?



Elias-Fano representation

[Elias-Fano 1975]

Given a sequence S
integers up to m

Average distance between
consecutive integers in S

Space: n log (m/n) + 2n bits
Access(i) in O(1) and NextGEQ(x) in O(log (m/n))

Largest integer in S

Binary search
O(n log m) bits :-(
Can we do better?

	1	2	3	4
S	2	3	5	10

A bit for any
possible integer

Binary vector requires
m + o(m) bits
Can we do better?

Represent S as a binary vector

The i-th value of

$Select_1(3)=5$

	0	1	2	3	4	5	6	7	8	9	10
B	0	0	1	1	0	1	0	0	0	0	1

Elias-Fano representation

[Elias-Fano 1975]

Given a sequence S of n positive and increasing integers up to m

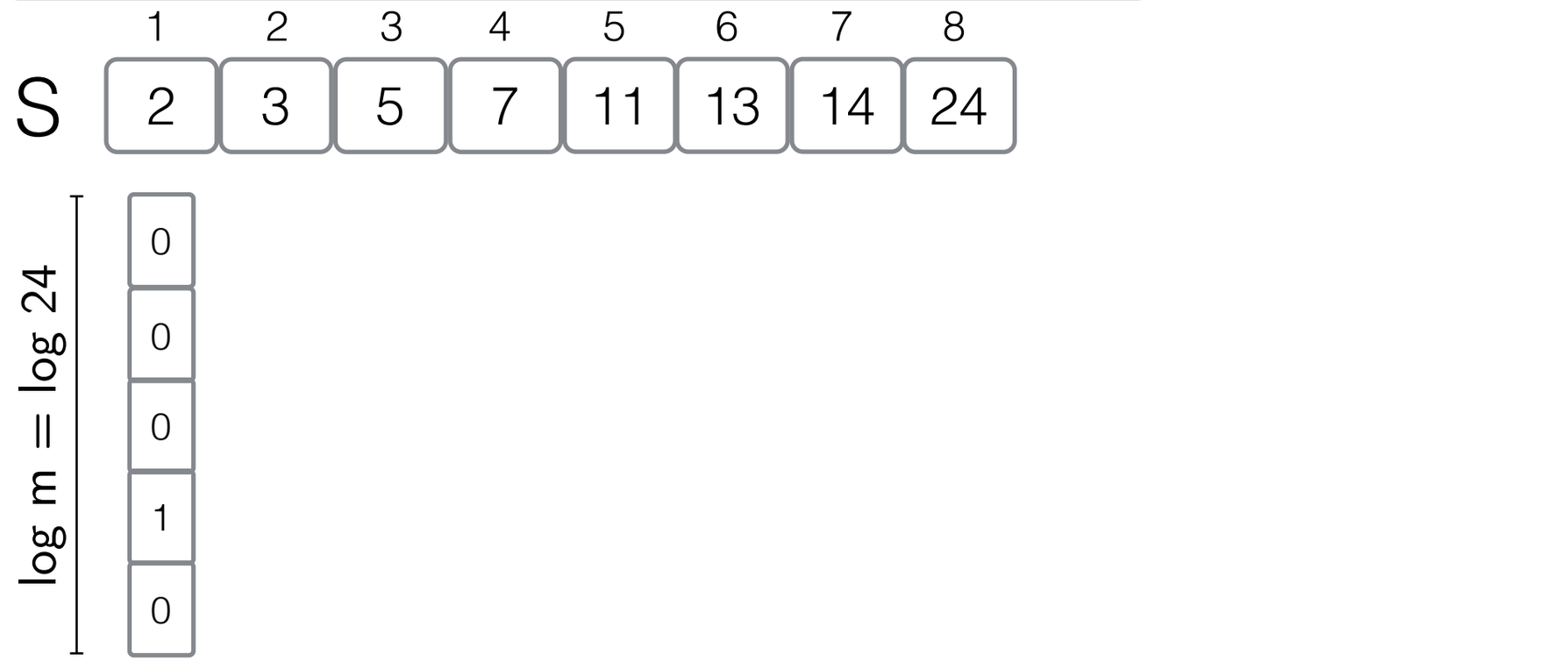
Space: n log (m/n) + 2n bits
Access(i) in O(1) and NextGEQ(x) in O(log (m/n))

Elias-Fano representation

[Elias-Fano 1975]

Given a sequence S of n positive and increasing integers up to m

Space: n log (m/n) + 2n bits
Access(i) in O(1) and NextGEQ(x) in O(log (m/n))

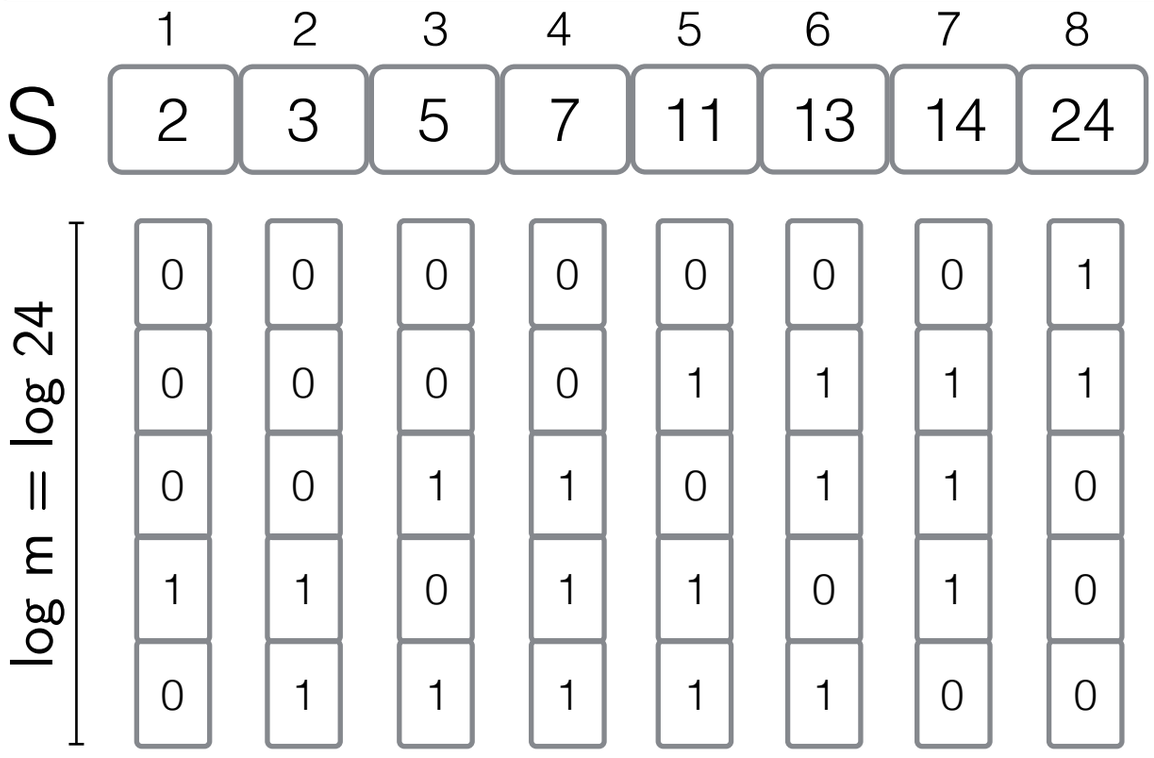


Elias-Fano representation

[Elias-Fano 1975]

Given a sequence S of n positive and increasing integers up to m

Space: n log (m/n) + 2n bits
Access(i) in O(1) and NextGEQ(x) in O(log (m/n))

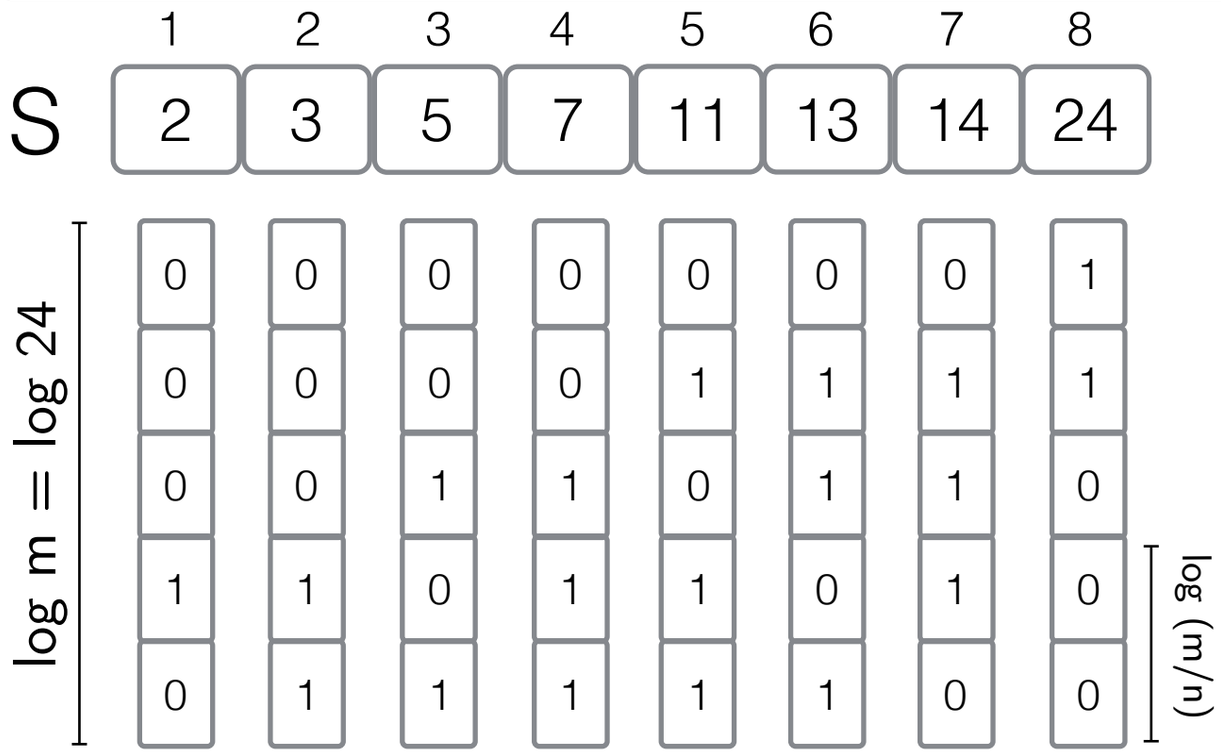


Elias-Fano representation

[Elias-Fano 1975]

Given a sequence S of n positive and increasing integers up to m

Space: n log (m/n) + 2n bits
Access(i) in O(1) and NextGEQ(x) in O(log (m/n))

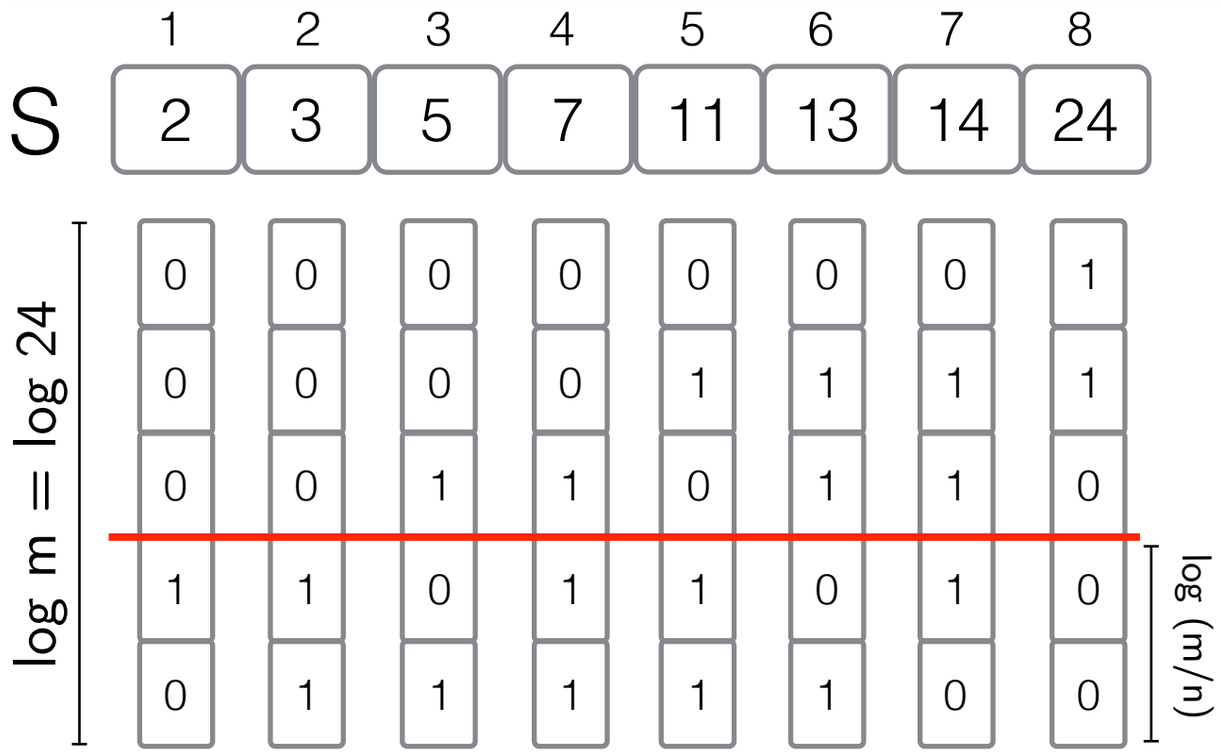


Elias-Fano representation

[Elias-Fano 1975]

Given a sequence S of n positive and increasing integers up to m

Space: n log (m/n) + 2n bits
Access(i) in O(1) and NextGEQ(x) in O(log (m/n))

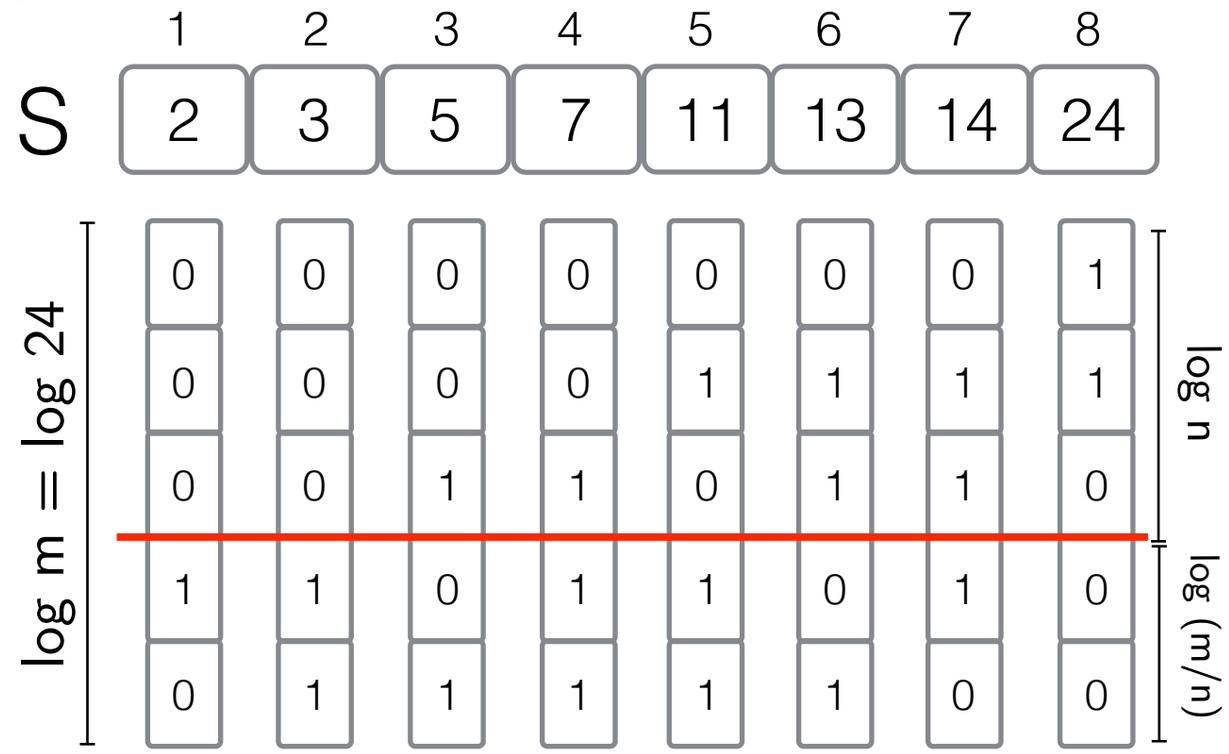


Elias-Fano representation

[Elias-Fano 1975]

Given a sequence S of n positive and increasing integers up to m

Space: n log (m/n) + 2n bits
Access(i) in O(1) and NextGEQ(x) in O(log (m/n))

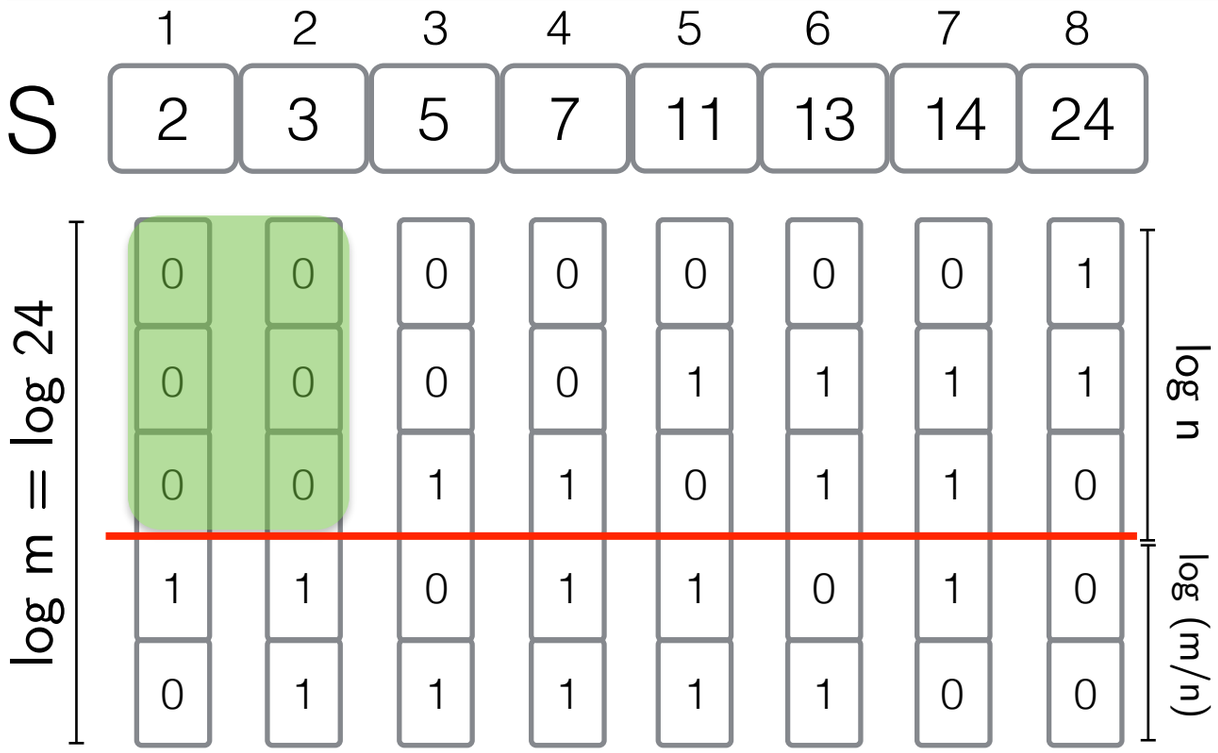


Elias-Fano representation

[Elias-Fano 1975]

Given a sequence S of n positive and increasing integers up to m

Space: n log (m/n) + 2n bits
Access(i) in O(1) and NextGEQ(x) in O(log (m/n))

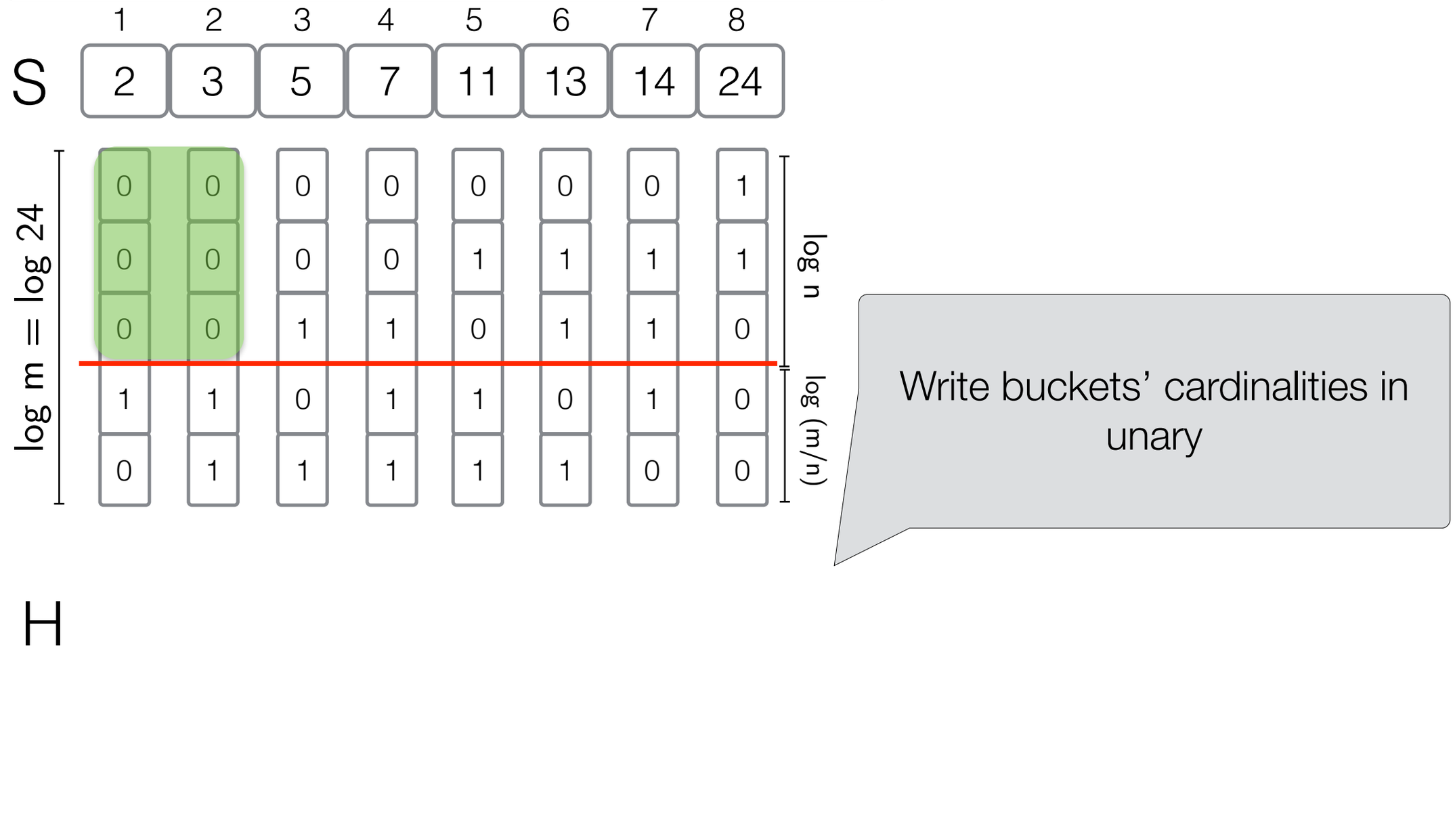


Elias-Fano representation

[Elias-Fano 1975]

Given a sequence S of n positive and increasing integers up to m

Space: n log (m/n) + 2n bits
Access(i) in O(1) and NextGEQ(x) in O(log (m/n))

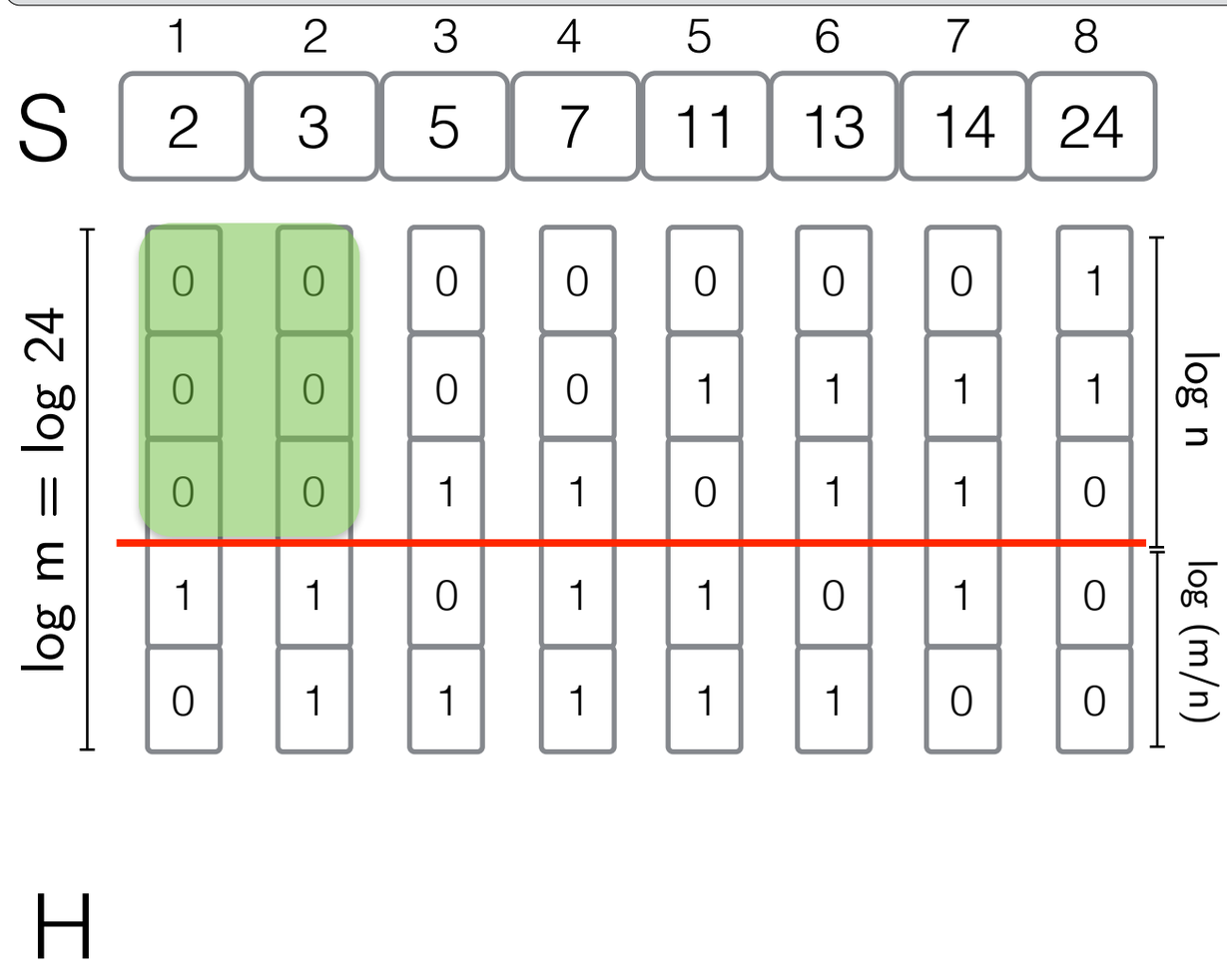


Elias-Fano representation

[Elias-Fano 1975]

Given a sequence S of n positive and increasing integers up to m

Space: n log (m/n) + 2n bits
Access(i) in O(1) and NextGEQ(x) in O(log (m/n))



Max number of buckets
$2^{\log n} = n$

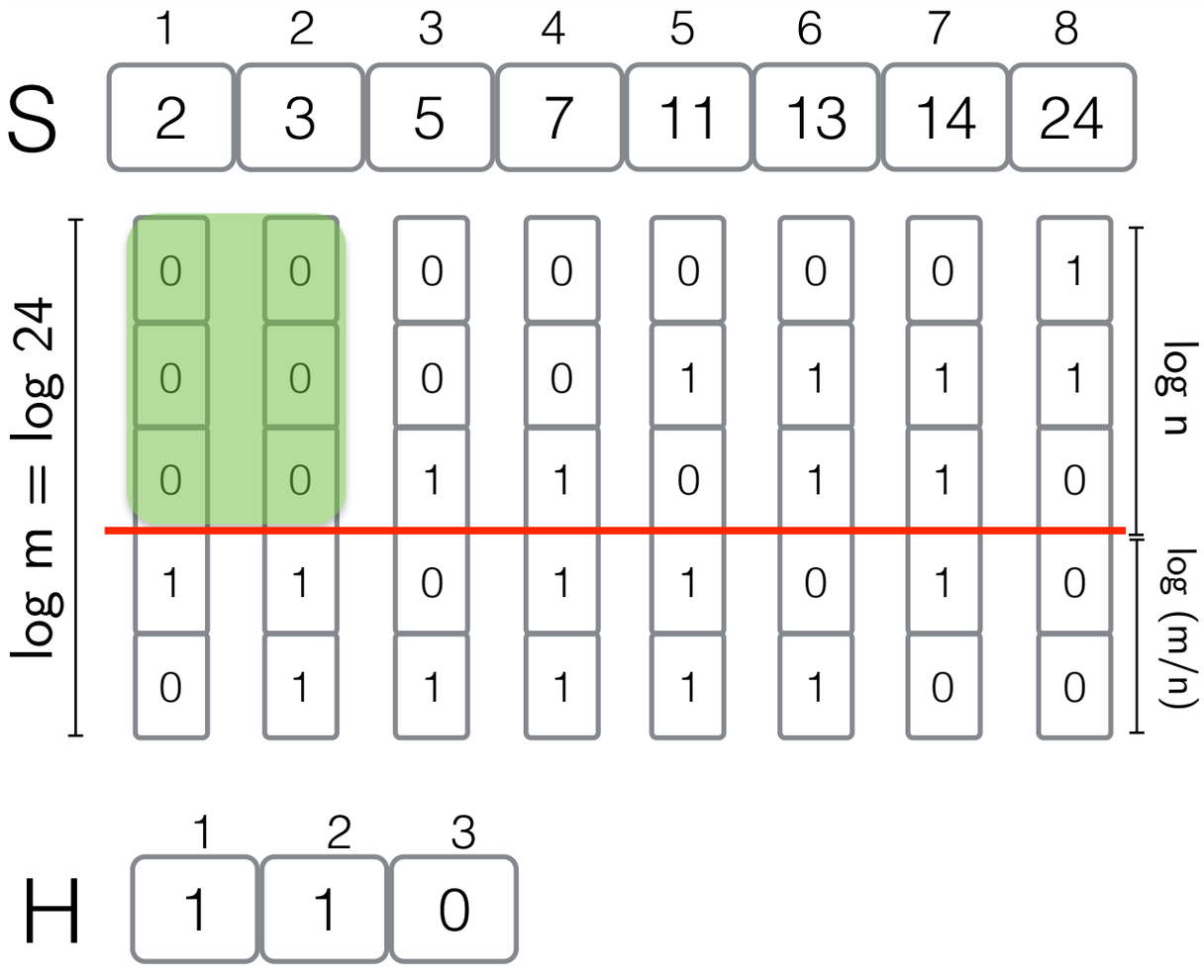
Write buckets' cardinalities in unary

Elias-Fano representation

[Elias-Fano 1975]

Given a sequence S of n positive and increasing integers up to m

Space: n log (m/n) + 2n bits
Access(i) in O(1) and NextGEQ(x) in O(log (m/n))



Max number of buckets
$2^{\log n} = n$

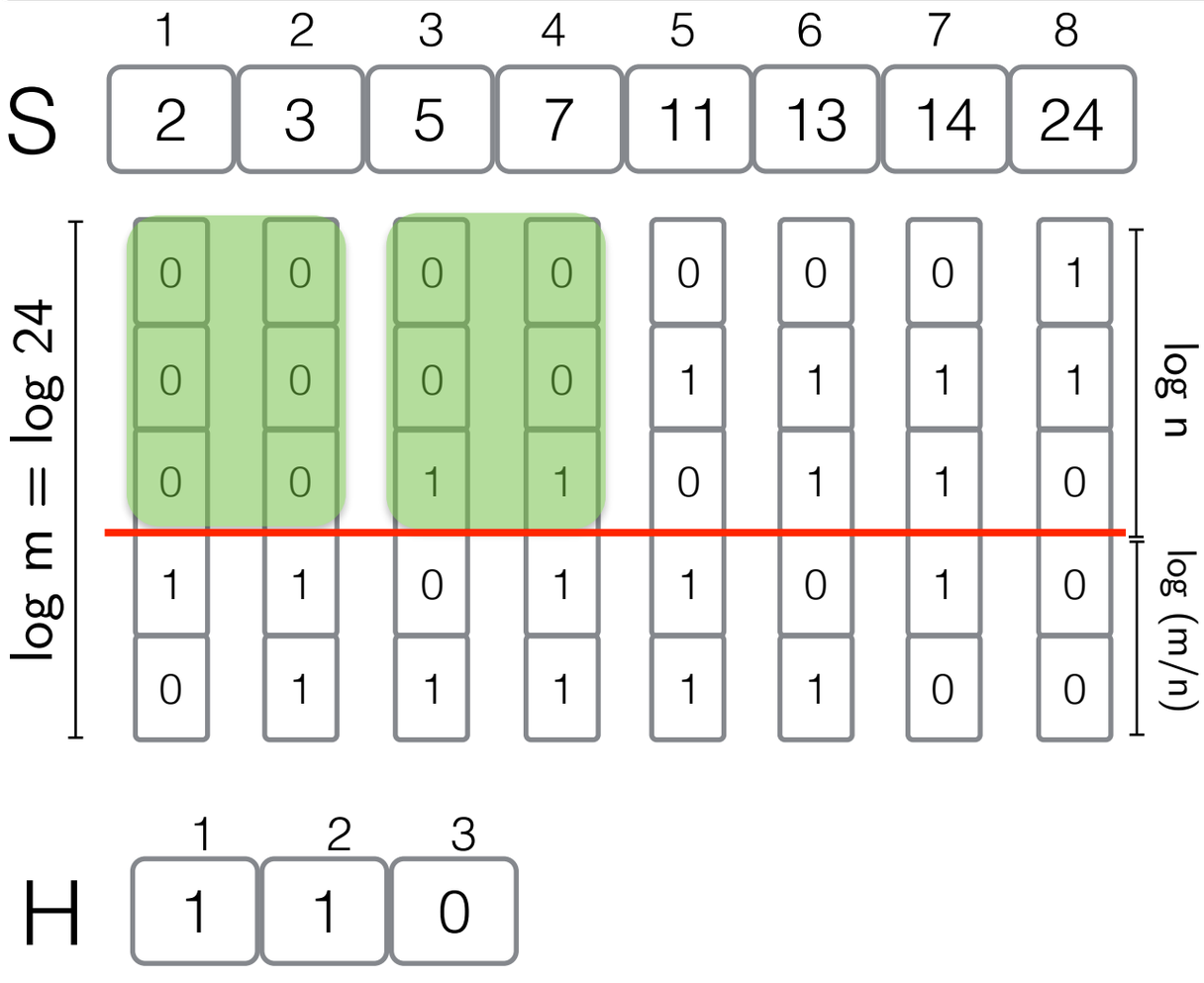
Write buckets' cardinalities in unary

Elias-Fano representation

[Elias-Fano 1975]

Given a sequence S of n positive and increasing integers up to m

Space: n log (m/n) + 2n bits
Access(i) in O(1) and NextGEQ(x) in O(log (m/n))



Max number of buckets
$2^{\log n} = n$

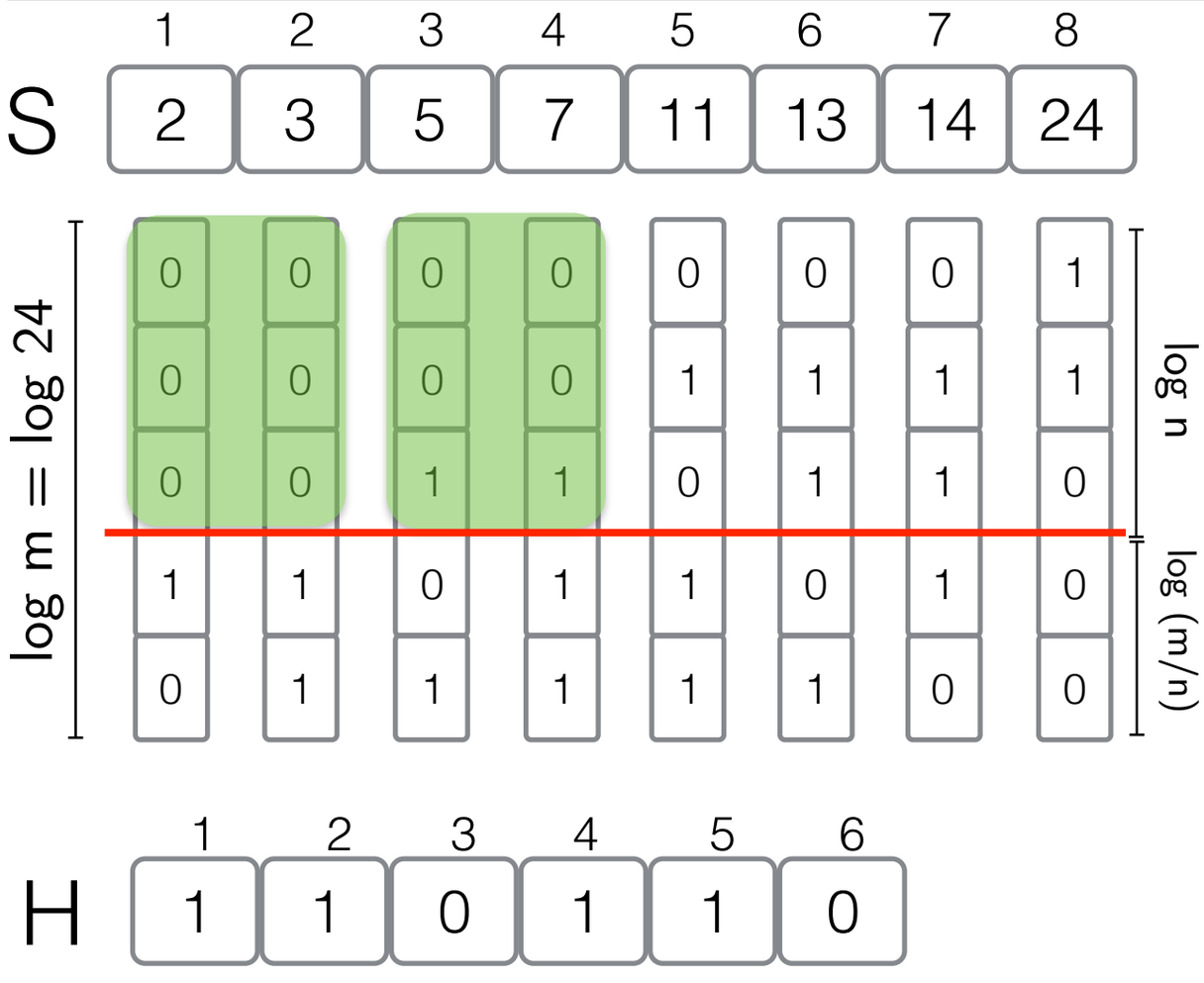
Write buckets' cardinalities in unary

Elias-Fano representation

[Elias-Fano 1975]

Given a sequence S of n positive and increasing integers up to m

Space: n log (m/n) + 2n bits
Access(i) in O(1) and NextGEQ(x) in O(log (m/n))



Max number of buckets
$2^{\log n} = n$

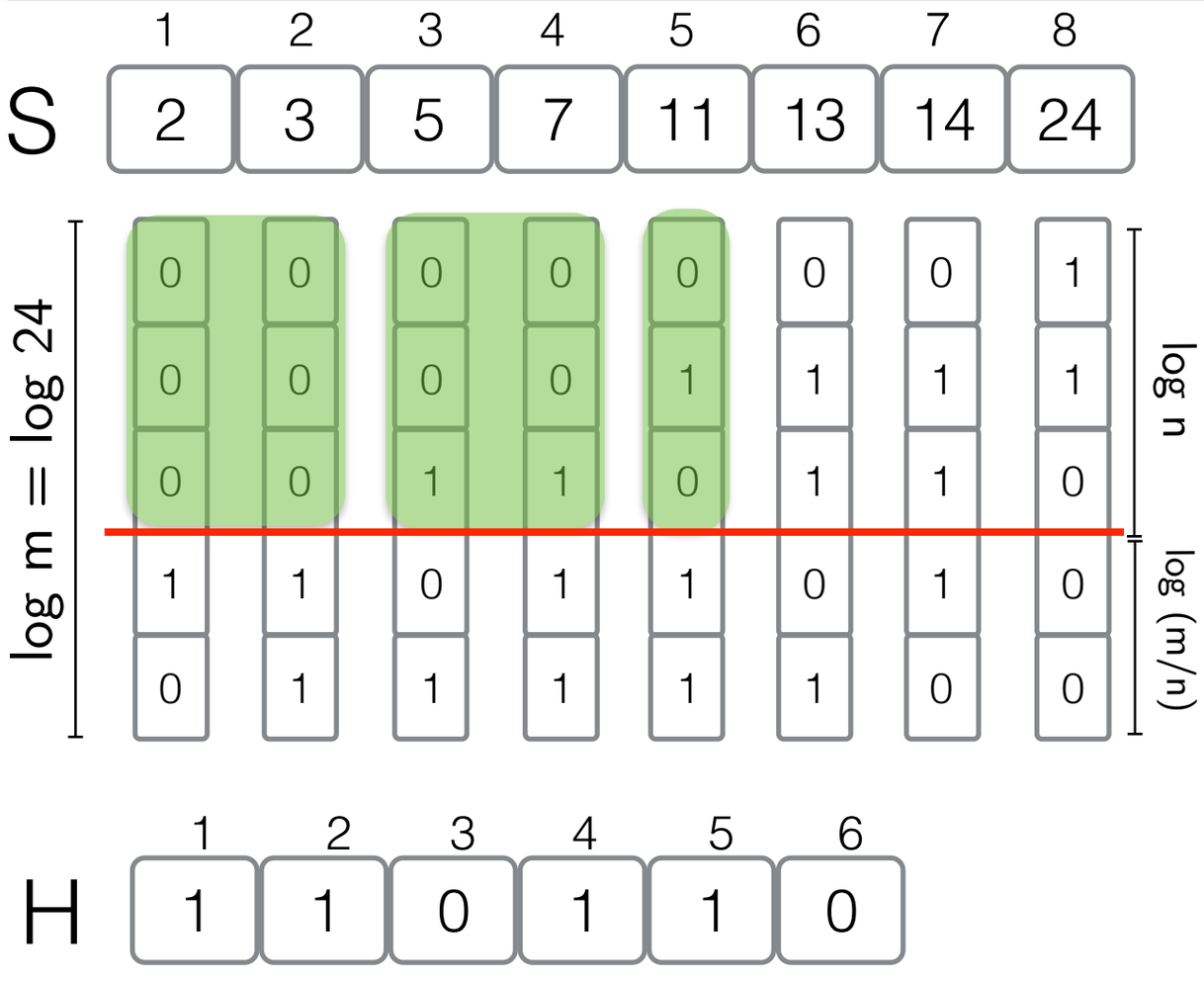
Write buckets' cardinalities in unary

Elias-Fano representation

[Elias-Fano 1975]

Given a sequence S of n positive and increasing integers up to m

Space: n log (m/n) + 2n bits
Access(i) in O(1) and NextGEQ(x) in O(log (m/n))



Max number of buckets
$2^{\log n} = n$

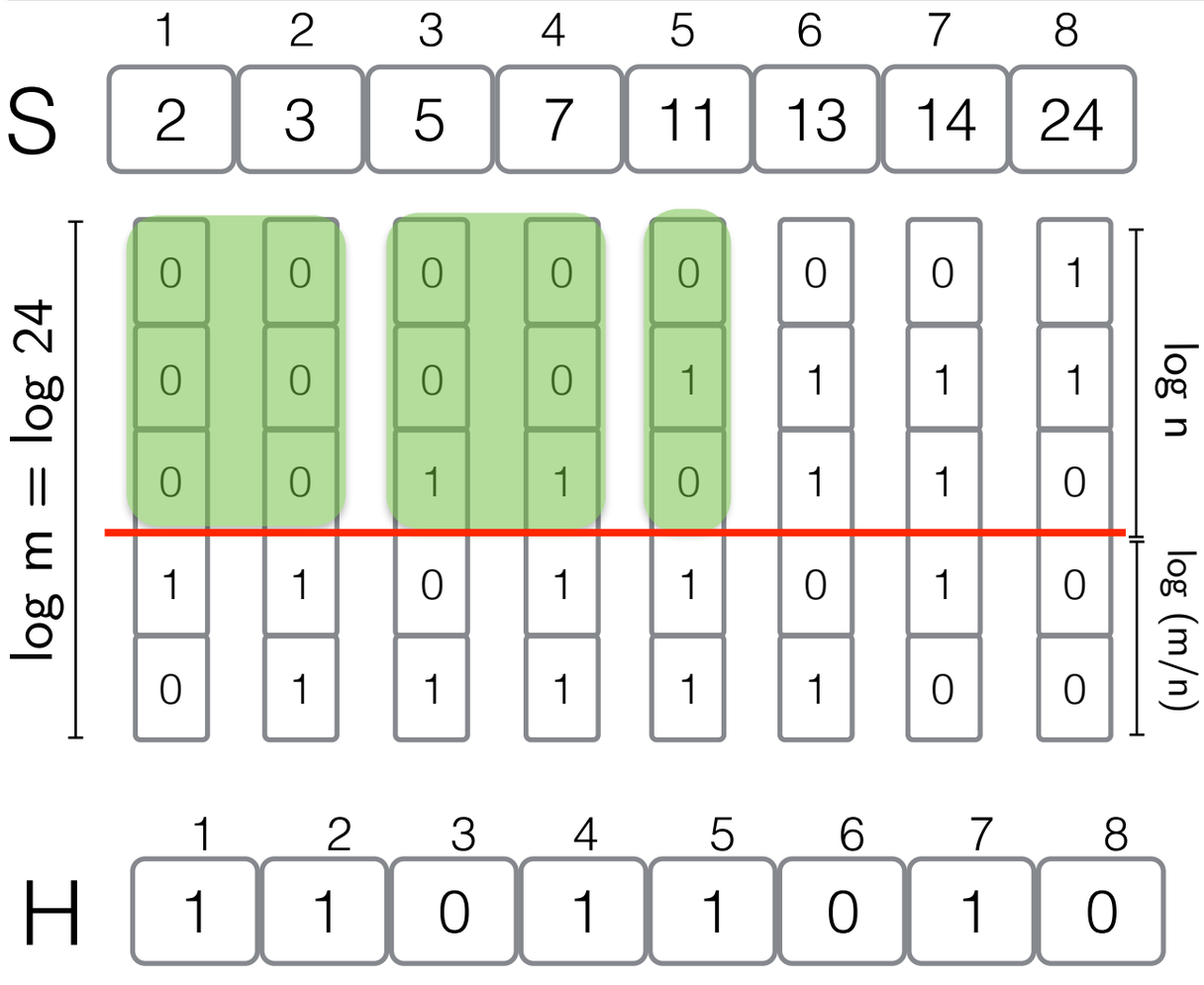
Write buckets' cardinalities in unary

Elias-Fano representation

[Elias-Fano 1975]

Given a sequence S of n positive and increasing integers up to m

Space: n log (m/n) + 2n bits
Access(i) in O(1) and NextGEQ(x) in O(log (m/n))



Max number of buckets
$2^{\log n} = n$

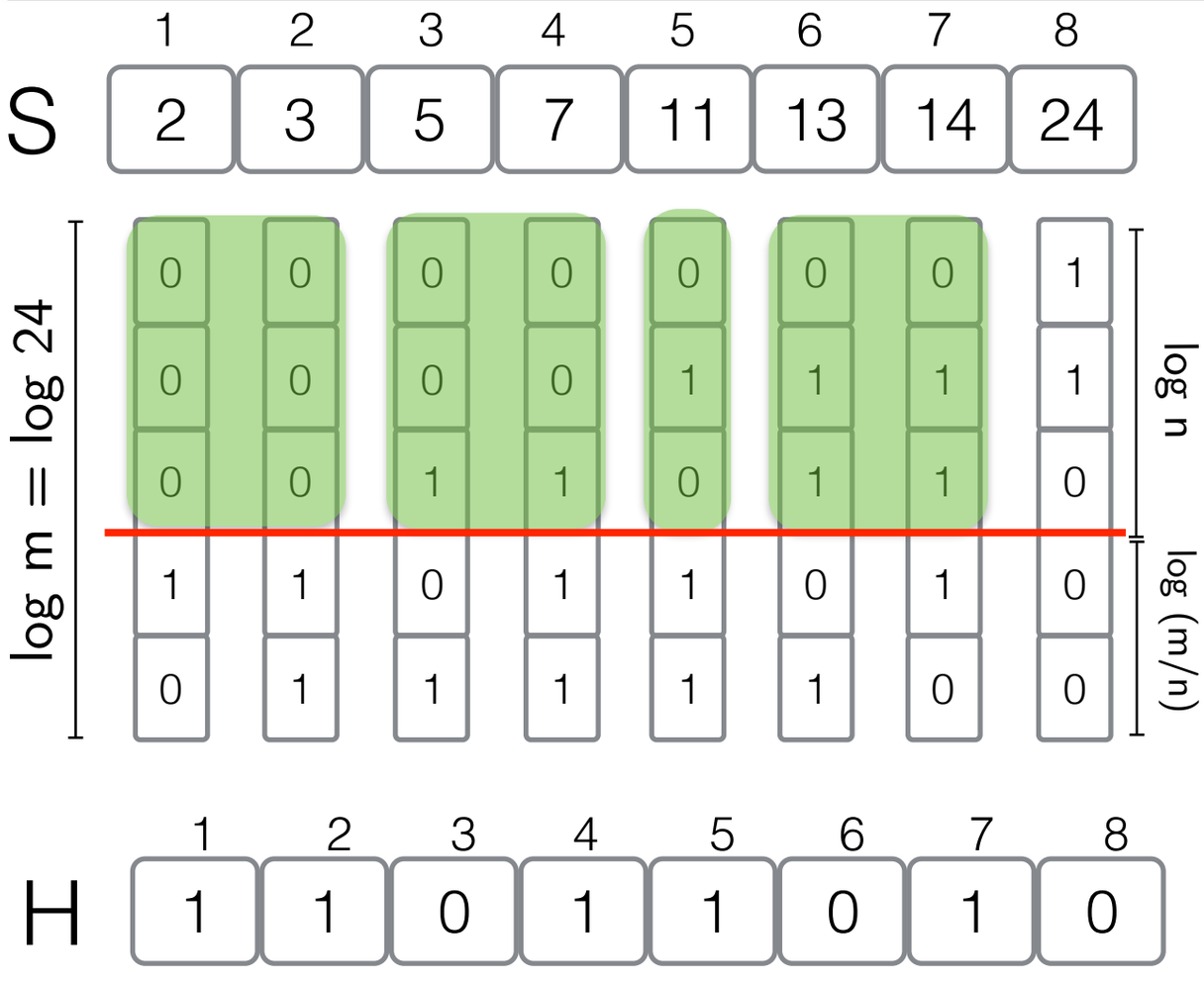
Write buckets' cardinalities in unary

Elias-Fano representation

[Elias-Fano 1975]

Given a sequence S of n positive and increasing integers up to m

Space: n log (m/n) + 2n bits
Access(i) in O(1) and NextGEQ(x) in O(log (m/n))



Max number of buckets
$2^{\log n} = n$

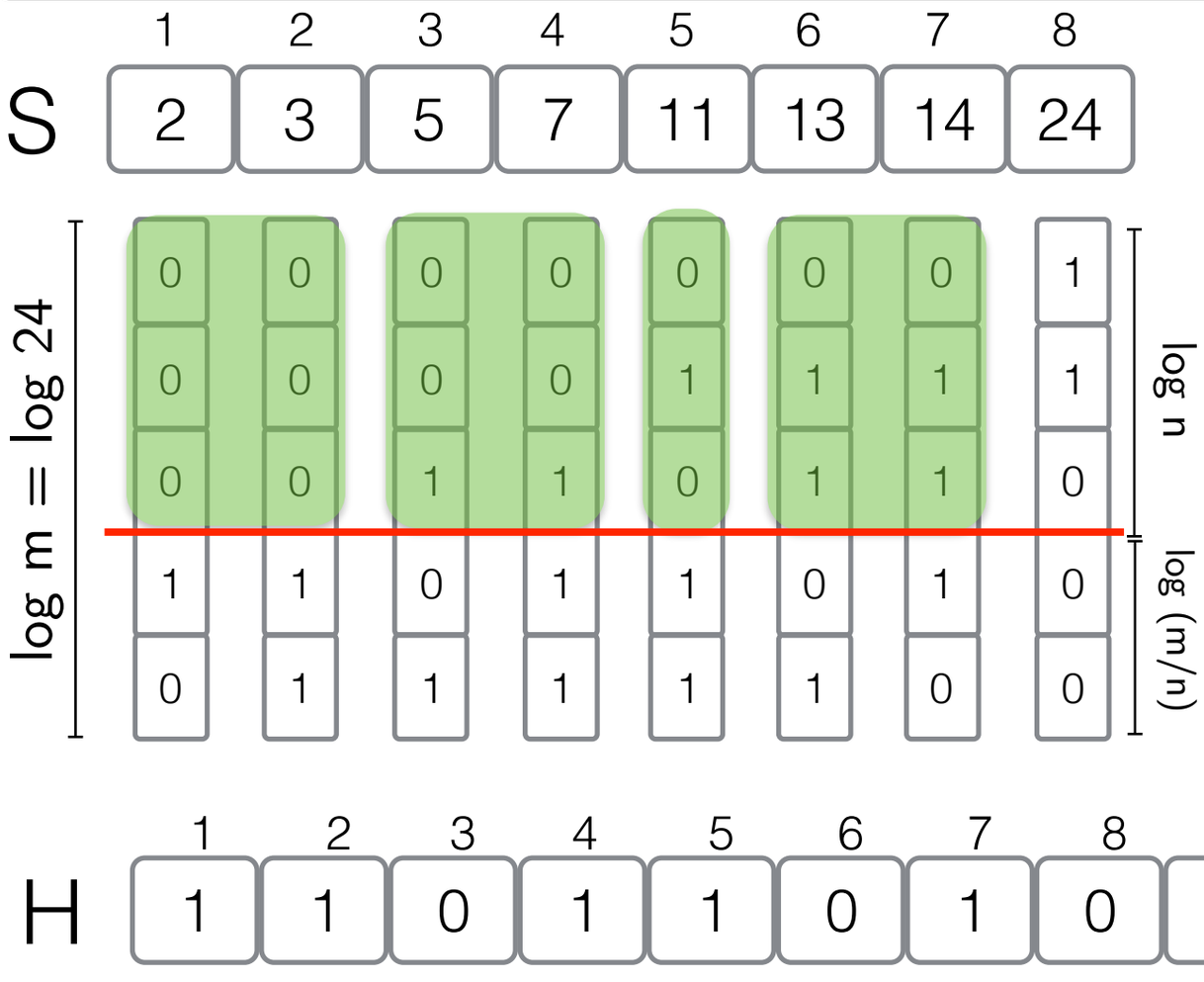
Write buckets' cardinalities in unary

Elias-Fano representation

[Elias-Fano 1975]

Given a sequence S of n positive and increasing integers up to m

Space: n log (m/n) + 2n bits
Access(i) in O(1) and NextGEQ(x) in O(log (m/n))



Max number of buckets
$2^{\log n} = n$

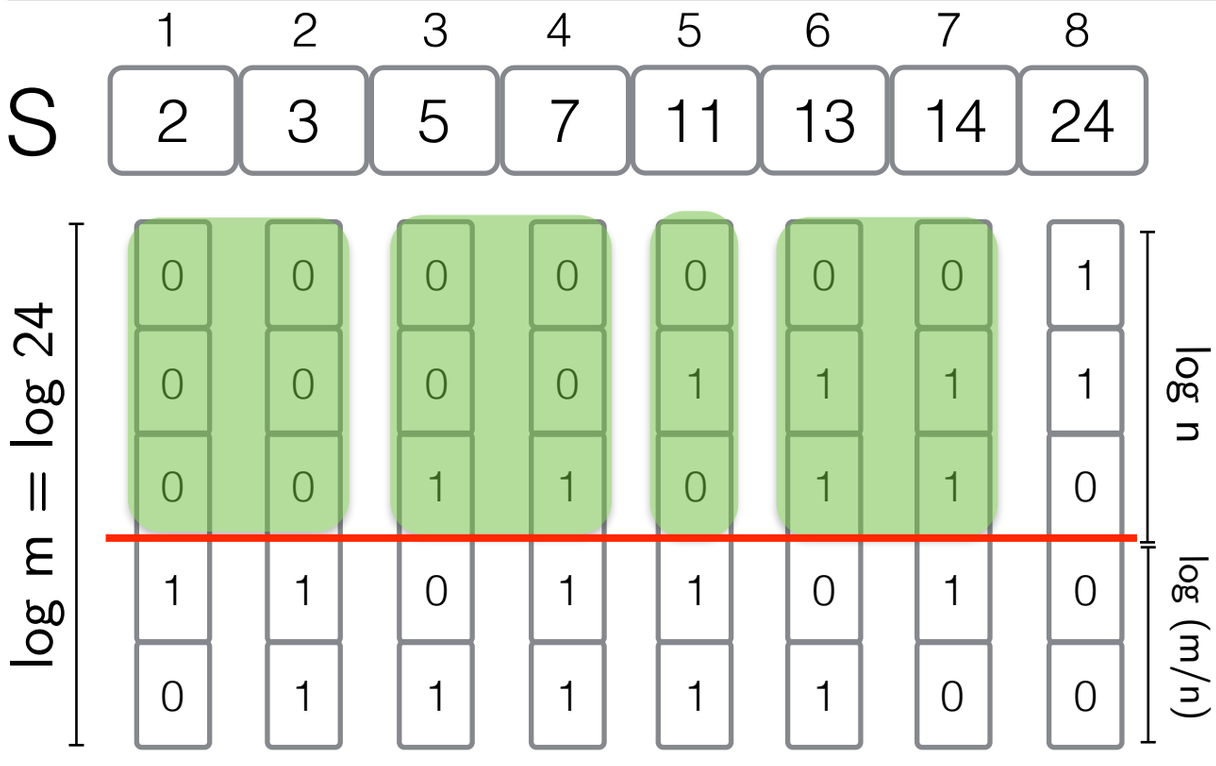
Write buckets' cardinalities in unary

Elias-Fano representation

[Elias-Fano 1975]

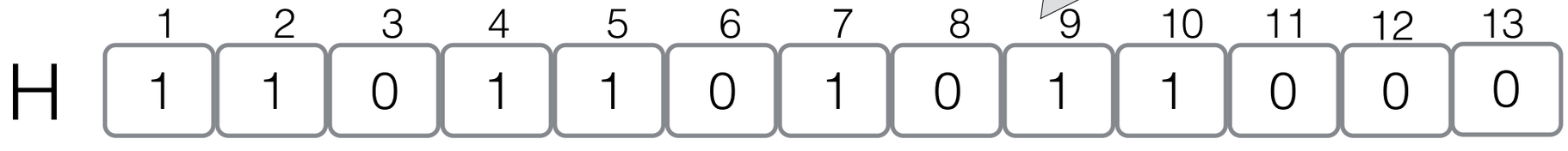
Given a sequence S of n positive and increasing integers up to m

Space: n log (m/n) + 2n bits
Access(i) in O(1) and NextGEQ(x) in O(log (m/n))



Max number of buckets
$2^{\log n} = n$

Write buckets' cardinalities in unary

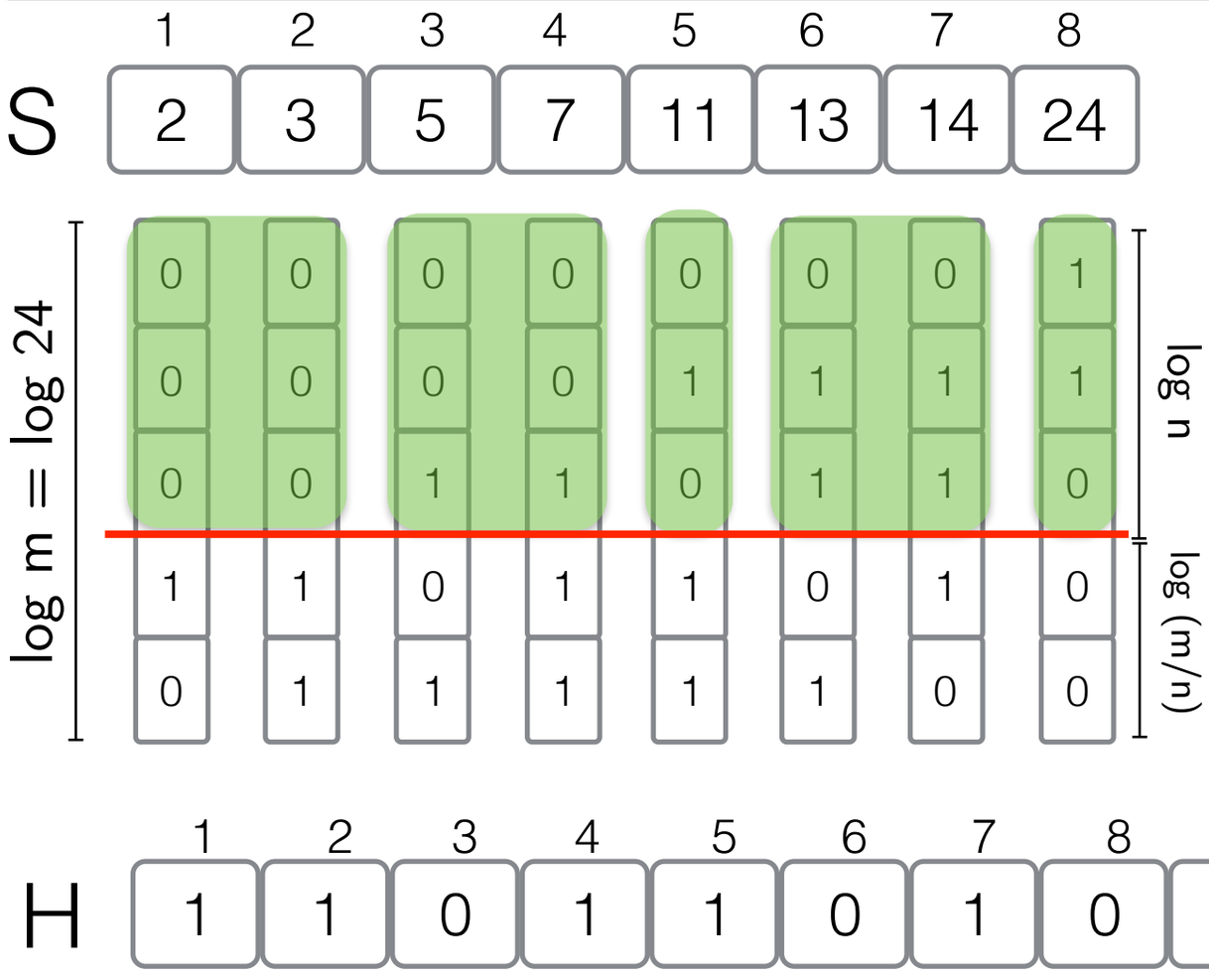


Elias-Fano representation

[Elias-Fano 1975]

Given a sequence S of n positive and increasing integers up to m

Space: n log (m/n) + 2n bits
Access(i) in O(1) and NextGEQ(x) in O(log (m/n))



Max number of buckets
$2^{\log n} = n$

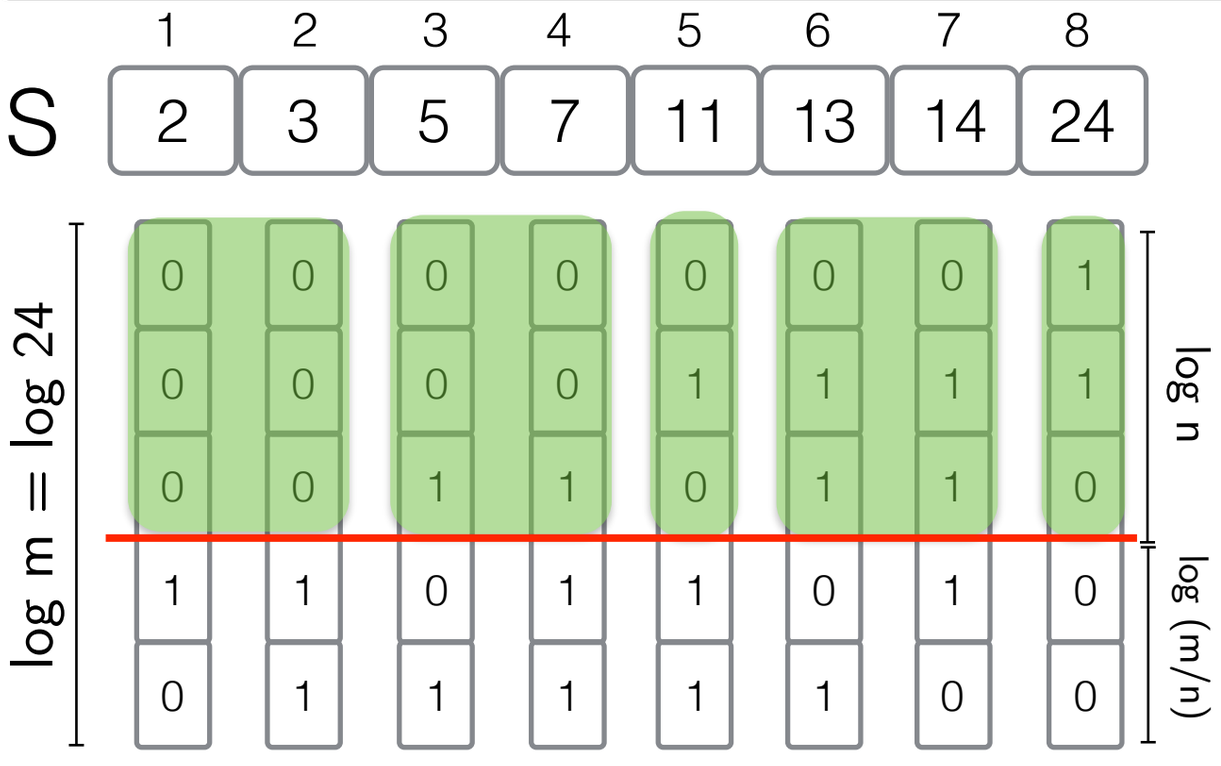
Write buckets' cardinalities in unary

Elias-Fano representation

[Elias-Fano 1975]

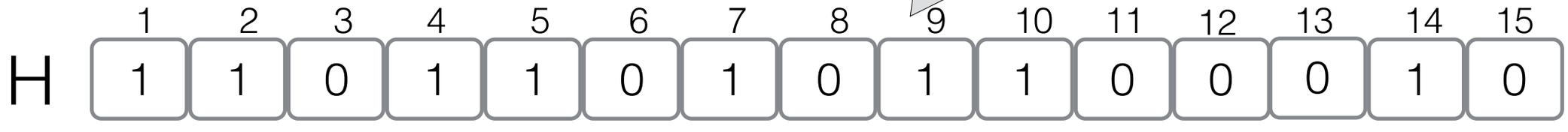
Given a sequence S of n positive and increasing integers up to m

Space: n log (m/n) + 2n bits
Access(i) in O(1) and NextGEQ(x) in O(log (m/n))



Max number of buckets
$2^{\log n} = n$

Write buckets' cardinalities in unary

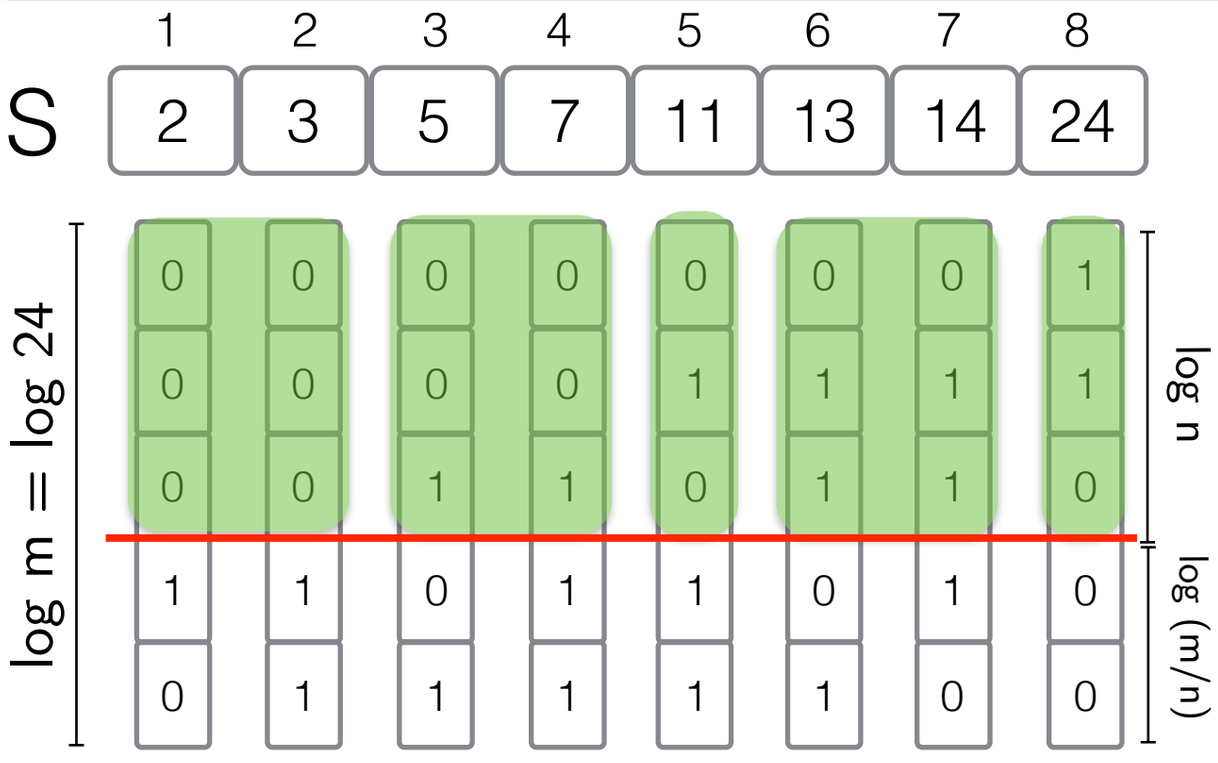


Elias-Fano representation

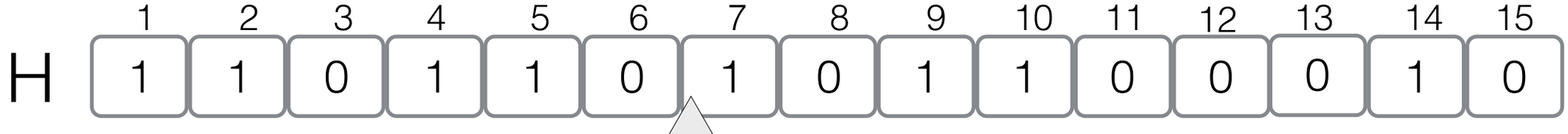
[Elias-Fano 1975]

Given a sequence S of n positive and increasing integers up to m

Space: n log (m/n) + 2n bits
Access(i) in O(1) and NextGEQ(x) in O(log (m/n))



Max number of buckets
$2^{\log n} = n$



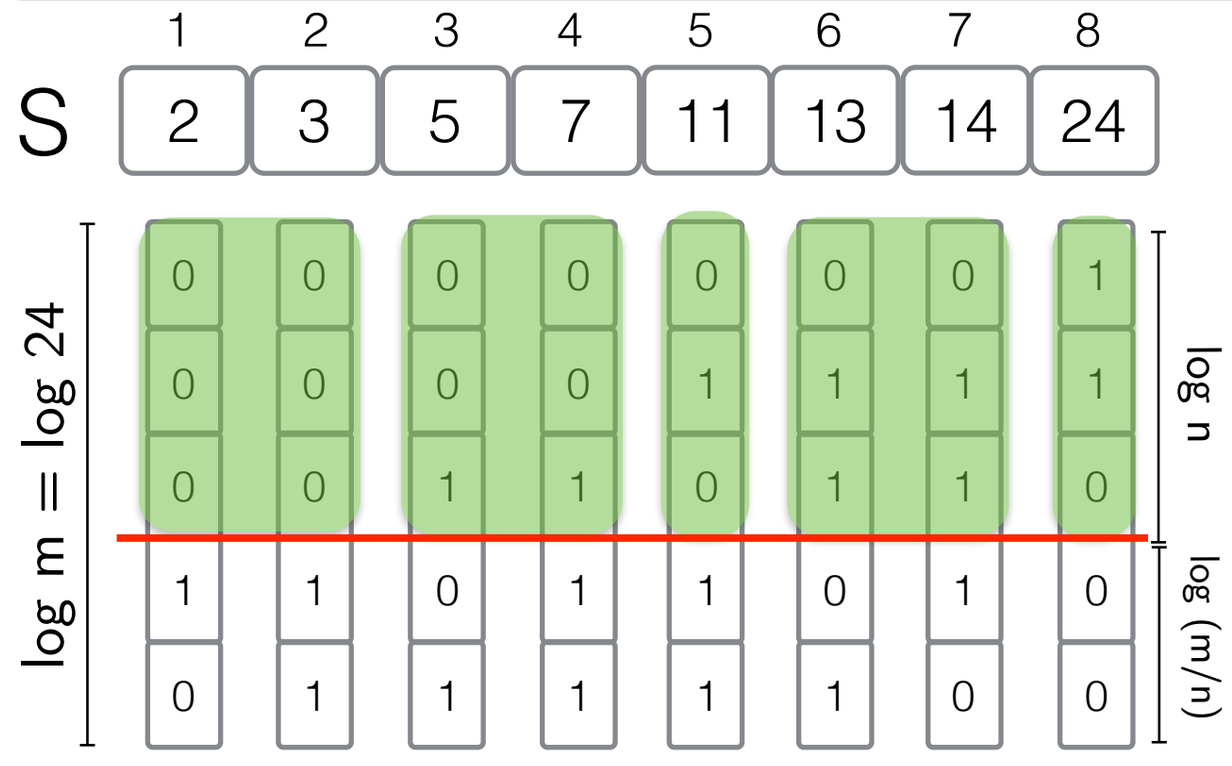
A 1 for each value, at most a 0 for each bucket. <= 2n bits!

Elias-Fano representation

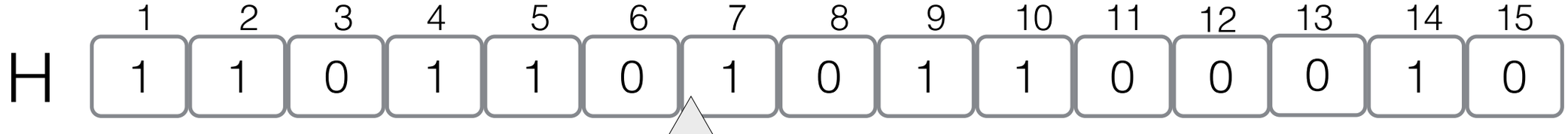
[Elias-Fano 1975]

Given a sequence S of n positive and increasing integers up to m

Space: n log (m/n) + 2n bits
Access(i) in O(1) and NextGEQ(x) in O(log (m/n))



Max number of buckets
$2^{\log n} = n$



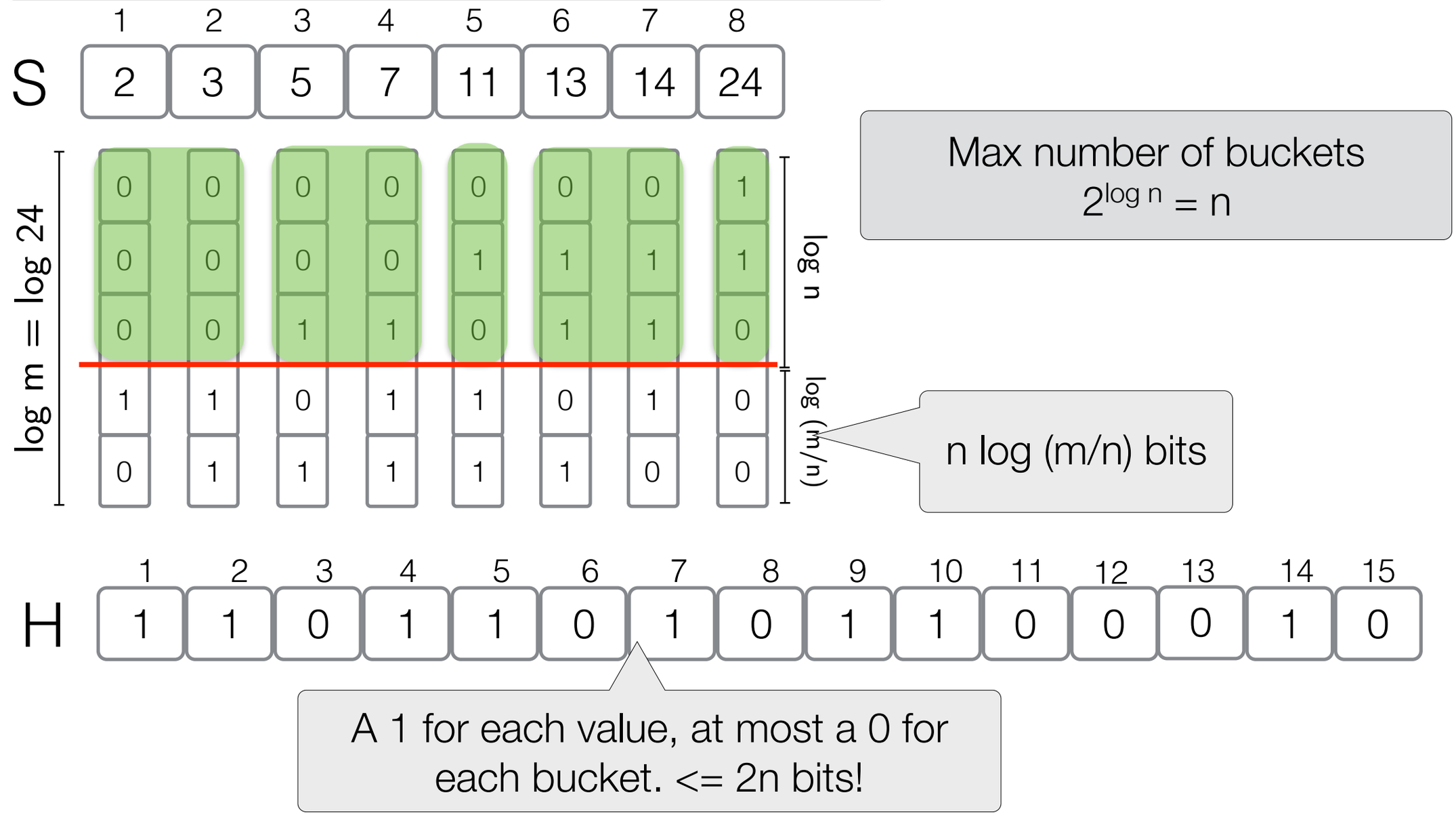
A 1 for each value, at most a 0 for each bucket. <= 2n bits!

Elias-Fano representation

[Elias-Fano 1975]

Given a sequence S of n positive and increasing integers up to m

Space: n log (m/n) + 2n bits
Access(i) in O(1) and NextGEQ(x) in O(log (m/n))



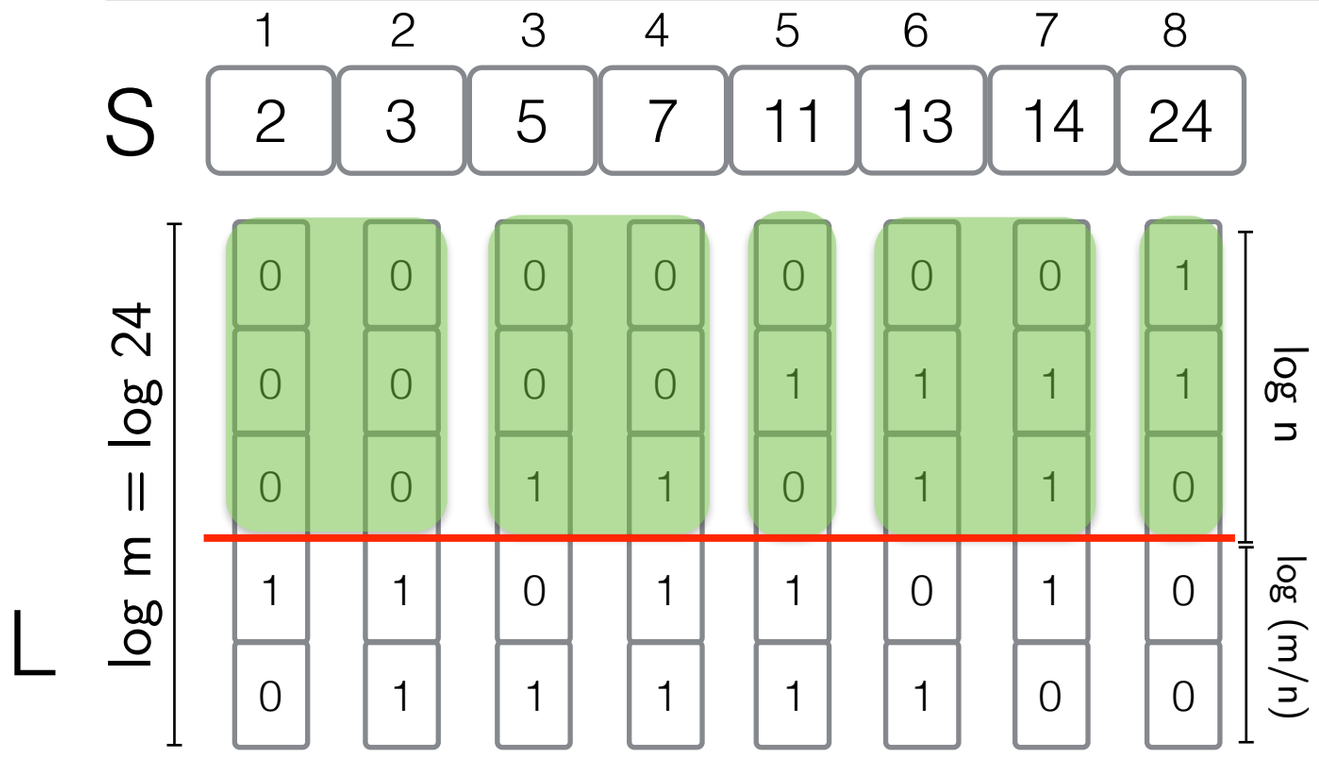
Max number of buckets
$2^{\log n} = n$

Elias-Fano representation

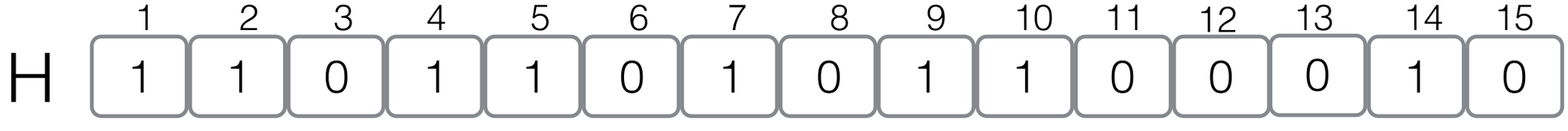
[Elias-Fano 1975]

Given a sequence S of n positive and increasing integers up to m

Space: n log (m/n) + 2n bits
Access(i) in O(1) and NextGEQ(x) in O(log (m/n))



Max number of buckets
$2^{\log n} = n$



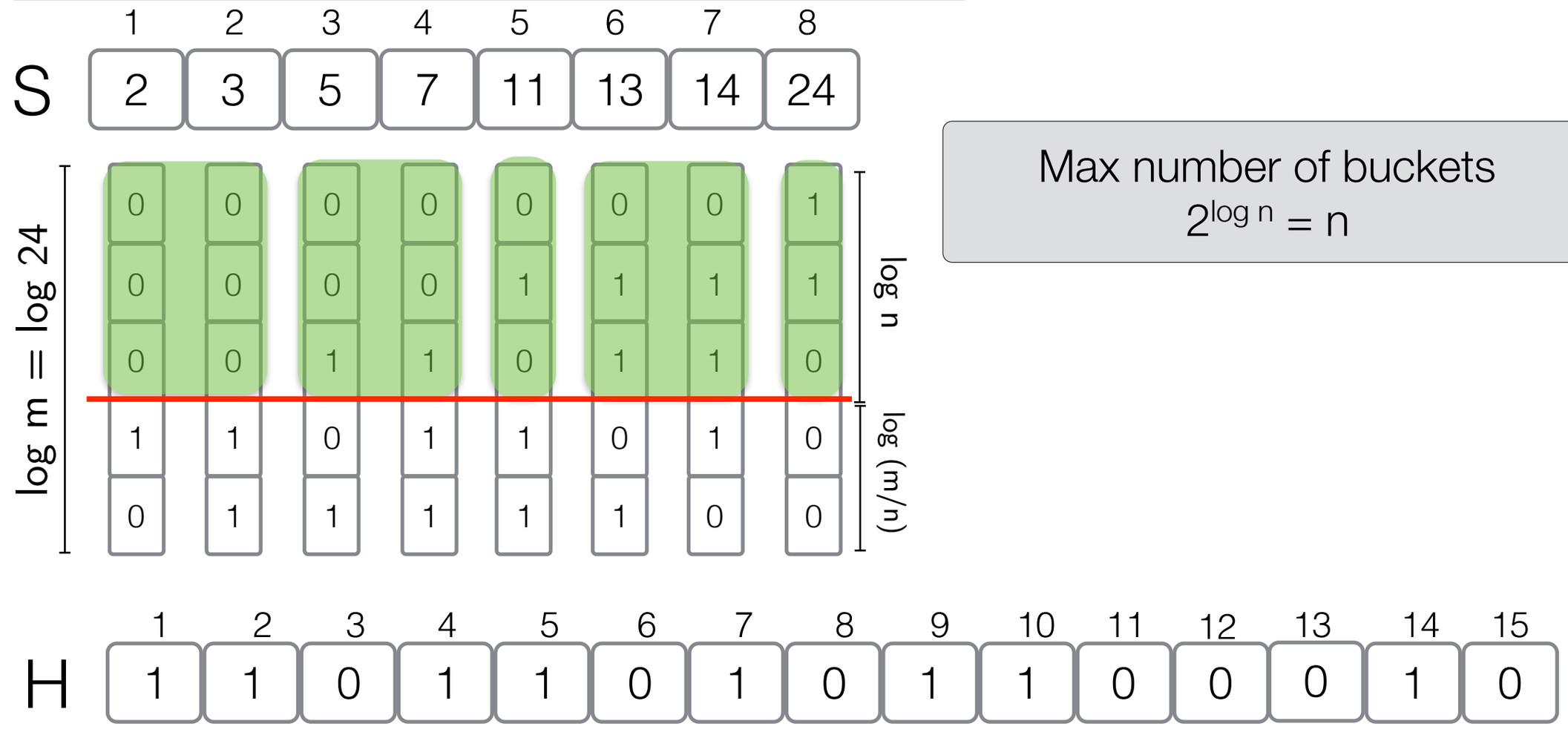
Access(6)

Elias-Fano representation

[Elias-Fano 1975]

Given a sequence S of n positive and increasing integers up to m

Space: n log (m/n) + 2n bits
Access(i) in O(1) and NextGEQ(x) in O(log (m/n))



Max number of buckets
$2^{\log n} = n$

Access(6)

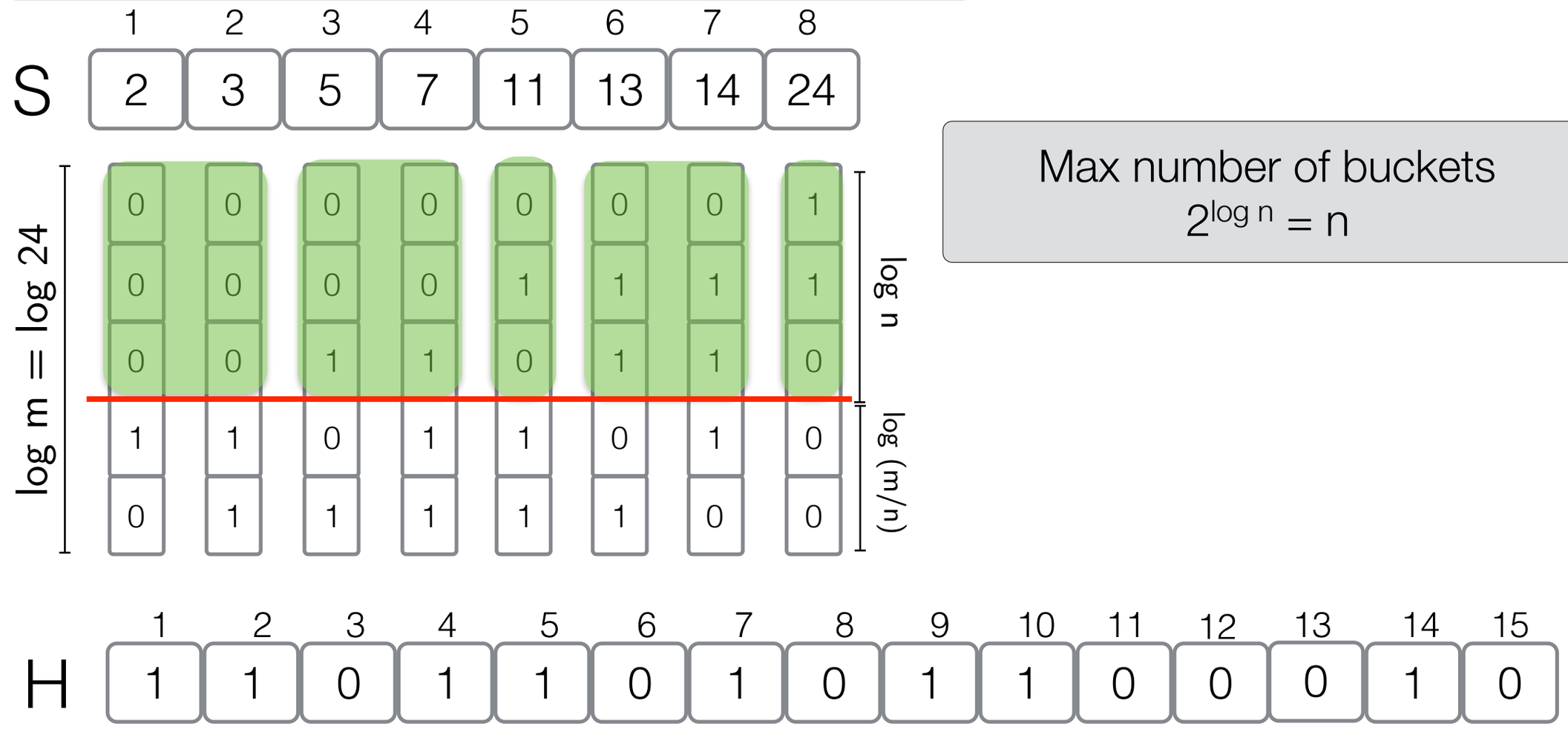
1. Access L in position 6

Elias-Fano representation

[Elias-Fano 1975]

Given a sequence S of n positive and increasing integers up to m

Space: n log (m/n) + 2n bits
Access(i) in O(1) and NextGEQ(x) in O(log (m/n))



Max number of buckets
$2^{\log n} = n$

Access(6)

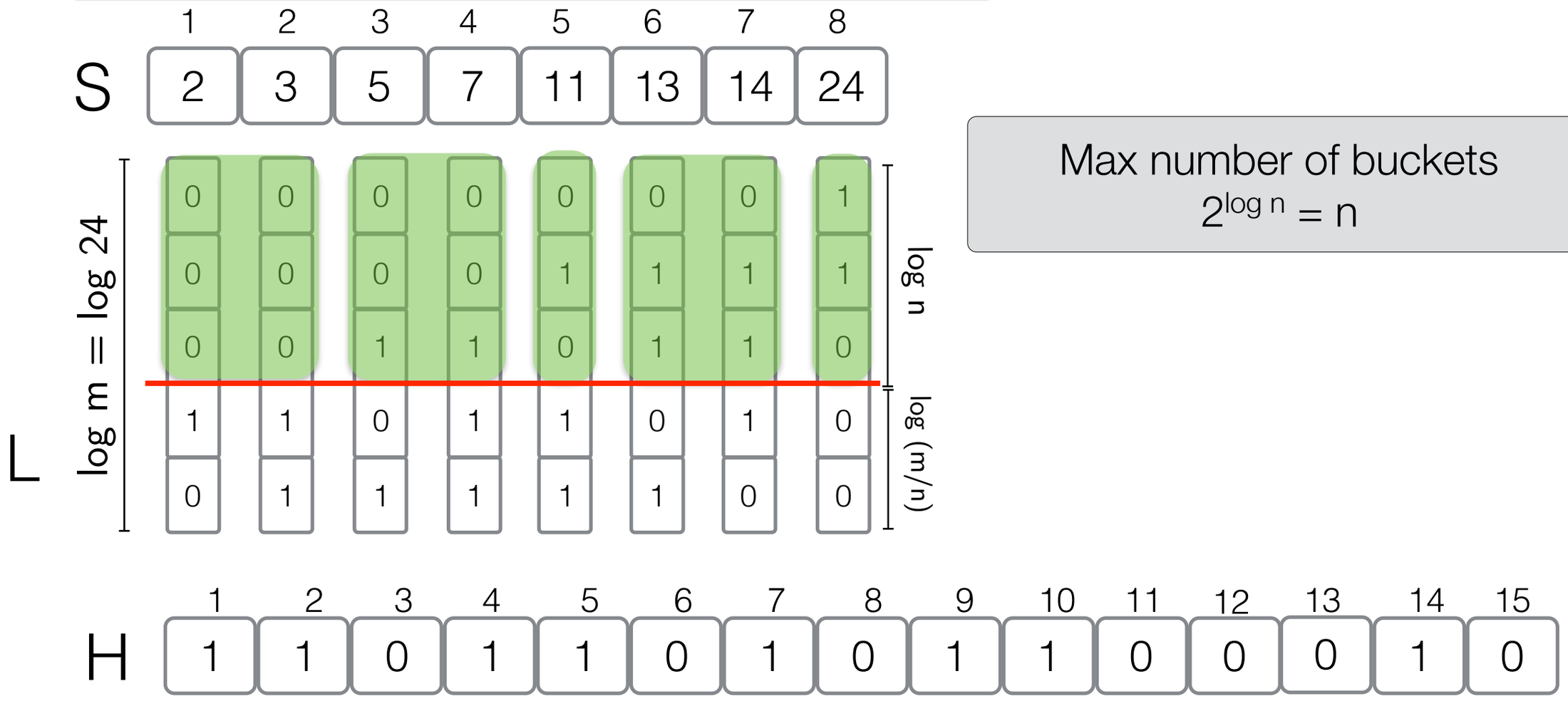
1. Access L in position 6 01

Elias-Fano representation

[Elias-Fano 1975]

Given a sequence S of n positive and increasing integers up to m

Space: n log (m/n) + 2n bits
Access(i) in O(1) and NextGEQ(x) in O(log (m/n))



Max number of buckets
$2^{\log n} = n$

2. $Select_1(6)-6 = 3$

Access(6)

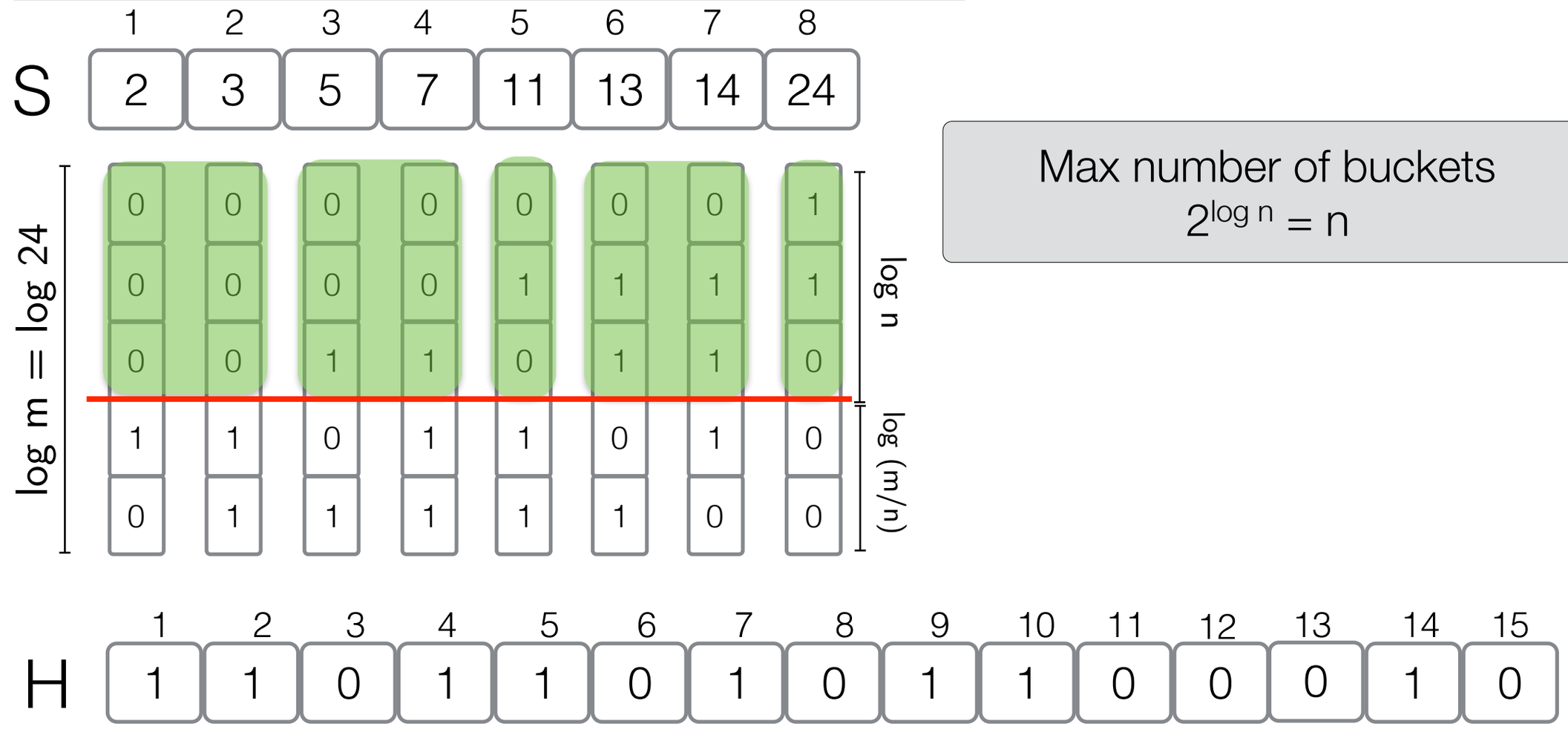
1. Access L in position 6 01

Elias-Fano representation

[Elias-Fano 1975]

Given a sequence S of n positive and increasing integers up to m

Space: n log (m/n) + 2n bits
Access(i) in O(1) and NextGEQ(x) in O(log (m/n))



Max number of buckets
$2^{\log n} = n$

2. $Select_1(6)-6 = 3$ 011 in binary!

Access(6)

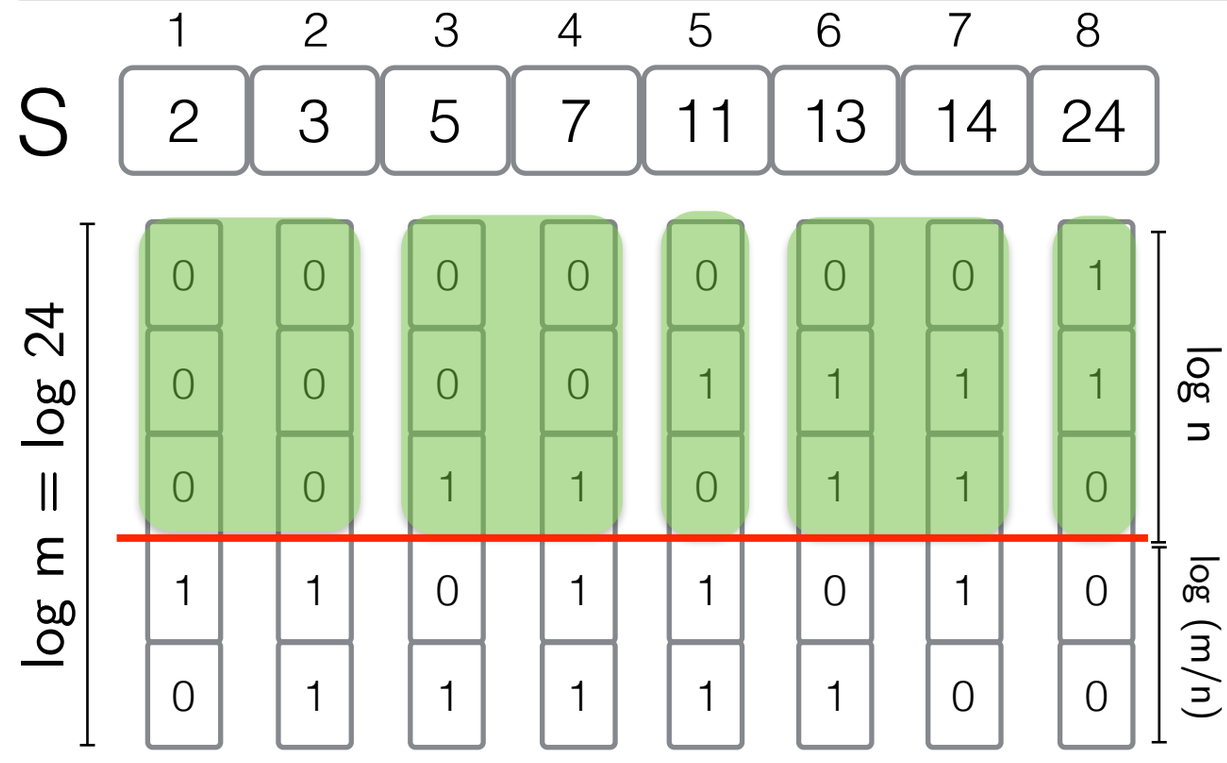
1. Access L in position 6 01

Elias-Fano representation

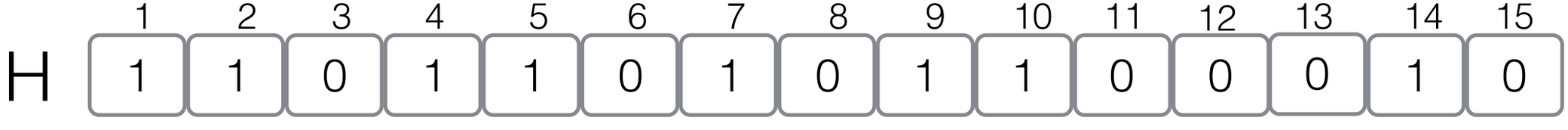
[Elias-Fano 1975]

Given a sequence S of n positive and increasing integers up to m

Space: n log (m/n) + 2n bits
Access(i) in O(1) and NextGEQ(x) in O(log (m/n))



Max number of buckets
$2^{\log n} = n$



Access(6)

1. Access L in position 6 01

2. $Select_1(6)-6 = 3$ 011 in binary!

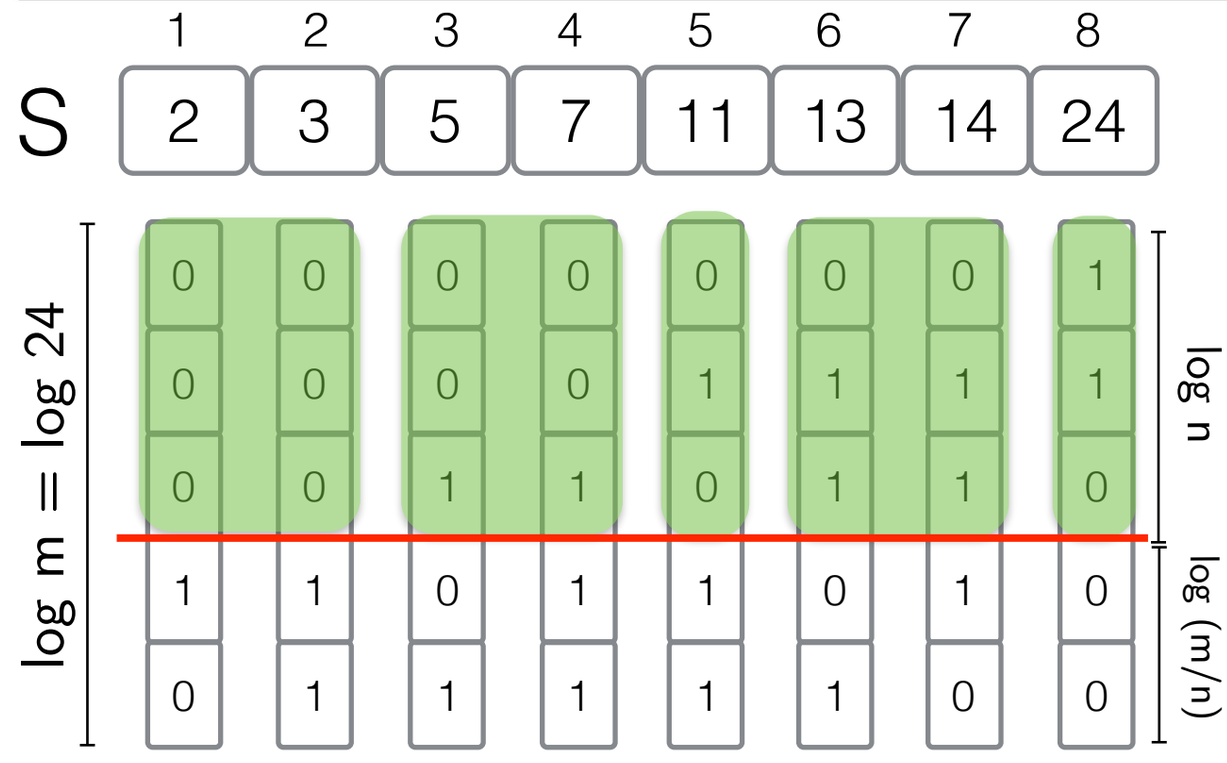
011 01 = 13

Elias-Fano representation

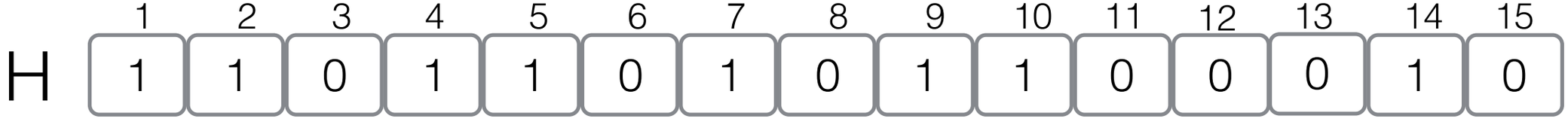
[Elias-Fano 1975]

Given a sequence S of n positive and increasing integers up to m

Space: n log (m/n) + 2n bits
Access(i) in O(1) and NextGEQ(x) in O(log (m/n))



Max number of buckets
$2^{\log n} = n$



Access(6)

NextGEQ queries are similar and need the decompression of only few values

1. Access L in position 6 01

011 01 = 13

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 information retrieval, but it includes features to promote results with good social proximity. It also supports queries that require multiple round-trips to leaves in order to retrieve objects that are more than one edge away from source nodes. Unicorn is designed to answer billions of queries per day at latencies in the hundreds of milliseconds, and it serves as an infrastructural building block for Facebook's Graph Search

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 in terms of both data volume and query volume. The system serves tens of billions of nodes and trillions of edges at scale while accounting for per-edge privacy, and it must also support realtime updates for all edges and nodes while serving billions of daily queries at low latencies.

This paper includes three main contributions:

- We describe how we applied common information re-

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 information retrieval, but it includes features to promote results with good social proximity. It also supports queries that require multiple round-trips to leaves in order to retrieve objects that are more than one edge away from source nodes. Unicorn is designed to answer billions of queries per day at latencies in the hundreds of milliseconds, and it serves as an infrastructural building block for Facebook's Graph Search

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 in terms of both data volume and query volume. The system serves tens of billions of nodes and trillions of edges at scale while accounting for per-edge privacy, and it must also support realtime updates for all edges and nodes while serving billions of daily queries at low latencies.

This paper includes three main contributions:

- We describe how we applied common information re-

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 information retrieval, but it includes features to promote results with good social proximity. It also supports queries that require multiple round-trips to leaves in order to retrieve objects that are more than one edge away from source nodes. Unicorn is designed to answer billions of queries per day at latencies in the hundreds of milliseconds, and it serves as an infrastructural building block for Facebook's Graph Search

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 in terms of both data volume and query volume. The system serves tens of billions of nodes and trillions of edges at scale while accounting for per-edge privacy, and it must also support realtime updates for all edges and nodes while serving billions of daily queries at low latencies.

This paper includes three main contributions:

- We describe how we applied common information re-

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 information retrieval, but it includes features to promote results with good social proximity. It also supports queries that require multiple round-trips to leaves in order to retrieve objects that are more than one edge away from source nodes. Unicorn is designed to answer billions of queries per day at latencies in the hundreds of milliseconds, and it serves as an infrastructural building block for Facebook's Graph Search

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.

14. REFERENCES

[1] Apache Hadoop. http://hadoop.apache.org/.

[2] Apache Thrift. http://thrift.apache.org/.

[3] Description of HHVM (PHP Virtual machine). https://www.facebook.com/note.php?note_id=10150415177928920.

[4] Facebook Graph Search. https://www.facebook.com/about/graphsearch.

[5] Folly GitHub Repository. http://github.com/facebook/folly.

Unicorn: A System for Searching the Social Graph

facebook / folly

Watch 545 | Star 5,406 | Fork 1,214

Branch: master ▾ folly / folly / experimental / EliasFanoCoding.h

ot 22 days ago Make EliasFanoReader and BitVectorReader API more consistent

6 contributors

689 lines (573 sloc) | 20.3 KB | Raw | Blame | History

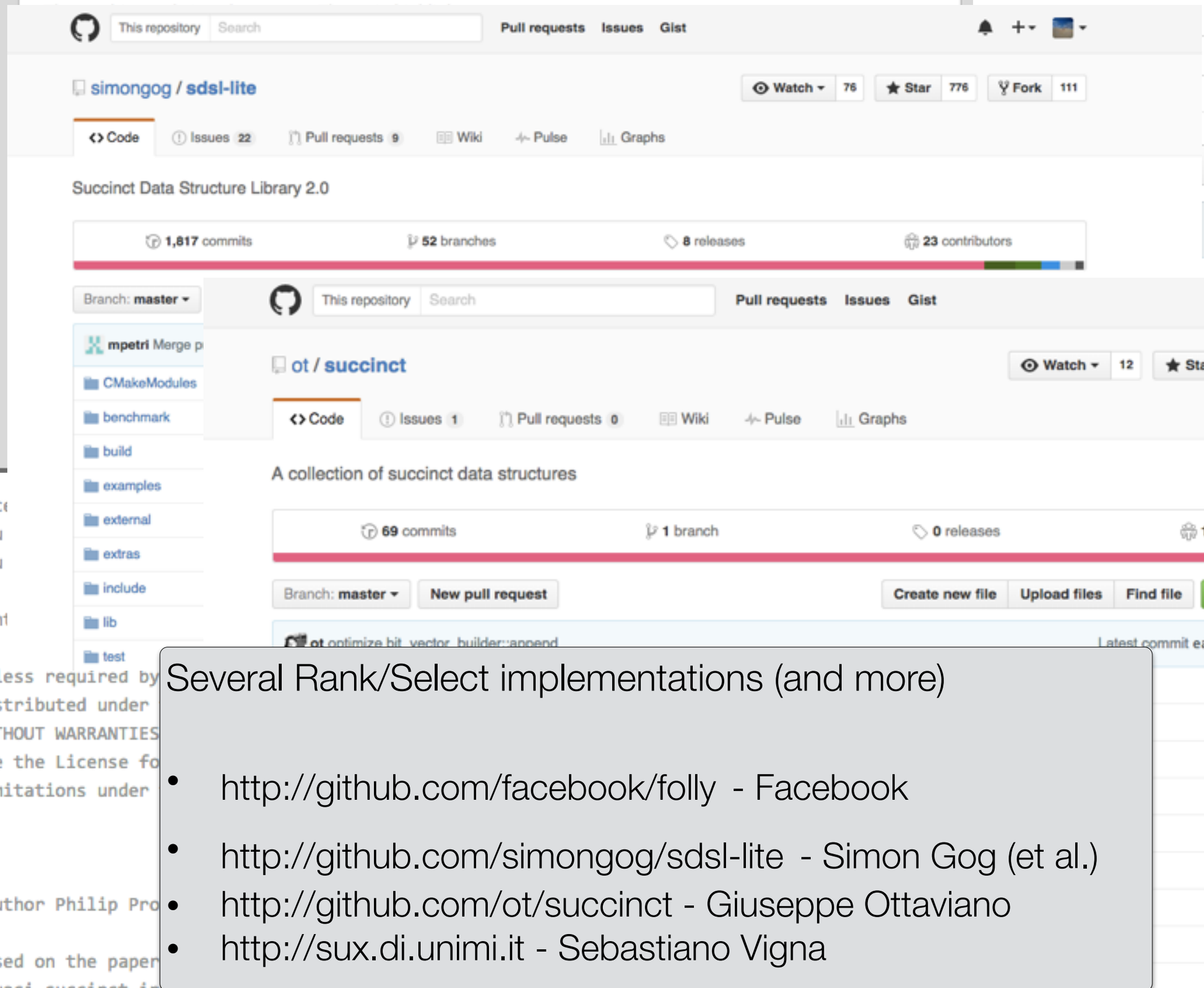
```
1  /*
2   * Copyright 2015 Facebook, Inc.
3   *
4   * Licensed under the Apache License, Version 2.0 (the "License");
5   * you may not use this file except in compliance with the License.
6   * You may obtain a copy of the License at
7   *
8   *   http://www.apache.org/licenses/LICENSE-2.0
9   *
10  * Unless required by applicable law or agreed to in writing, software
11  * distributed under the License is distributed on an "AS IS" BASIS,
12  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
13  * See the License for the specific language governing permissions and
14  * limitations under the License.
15  */
16
17 /**
18  * @author Philip Pronin (philipp@fb.com)
19  *
20  * Based on the paper by Sebastiano Vigna,
21  * "Quasi-succinct indices" (arxiv:1206.4300).
```

:::Sux: Implementing Succinct Data Structures

Introduction

Sux is an umbrella nickname for the results of my fiddling with the implementation of basic succinct data strucures.

al Graph



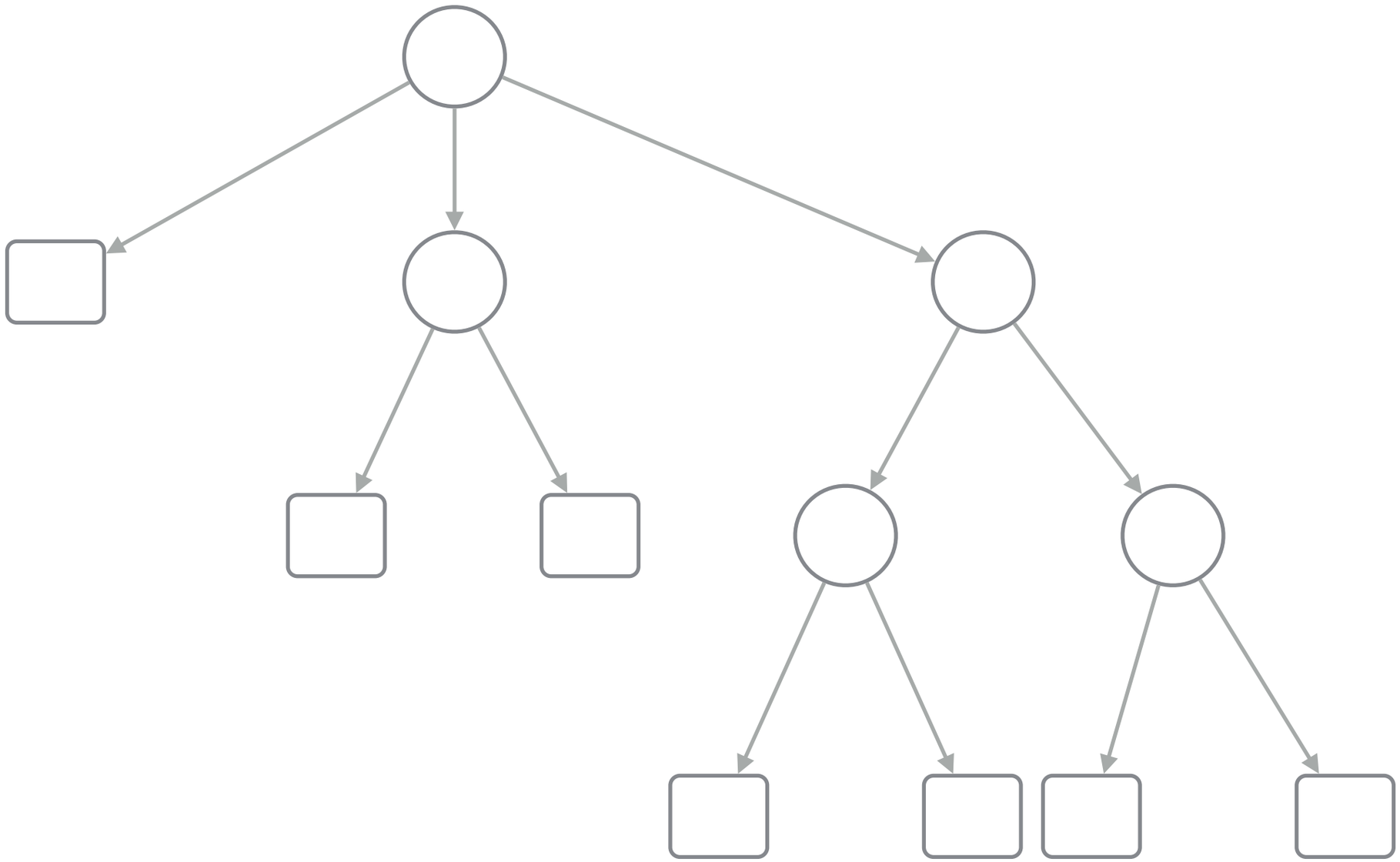
Several Rank/Select implementations (and more)

- http://github.com/facebook/folly - Facebook
- http://github.com/simongog/sdsl-lite - Simon Gog (et al.)
- http://github.com/ot/succinct - Giuseppe Ottaviano
- http://sux.di.unimi.it - Sebastiano Vigna

Applications

Succinct representation of trees

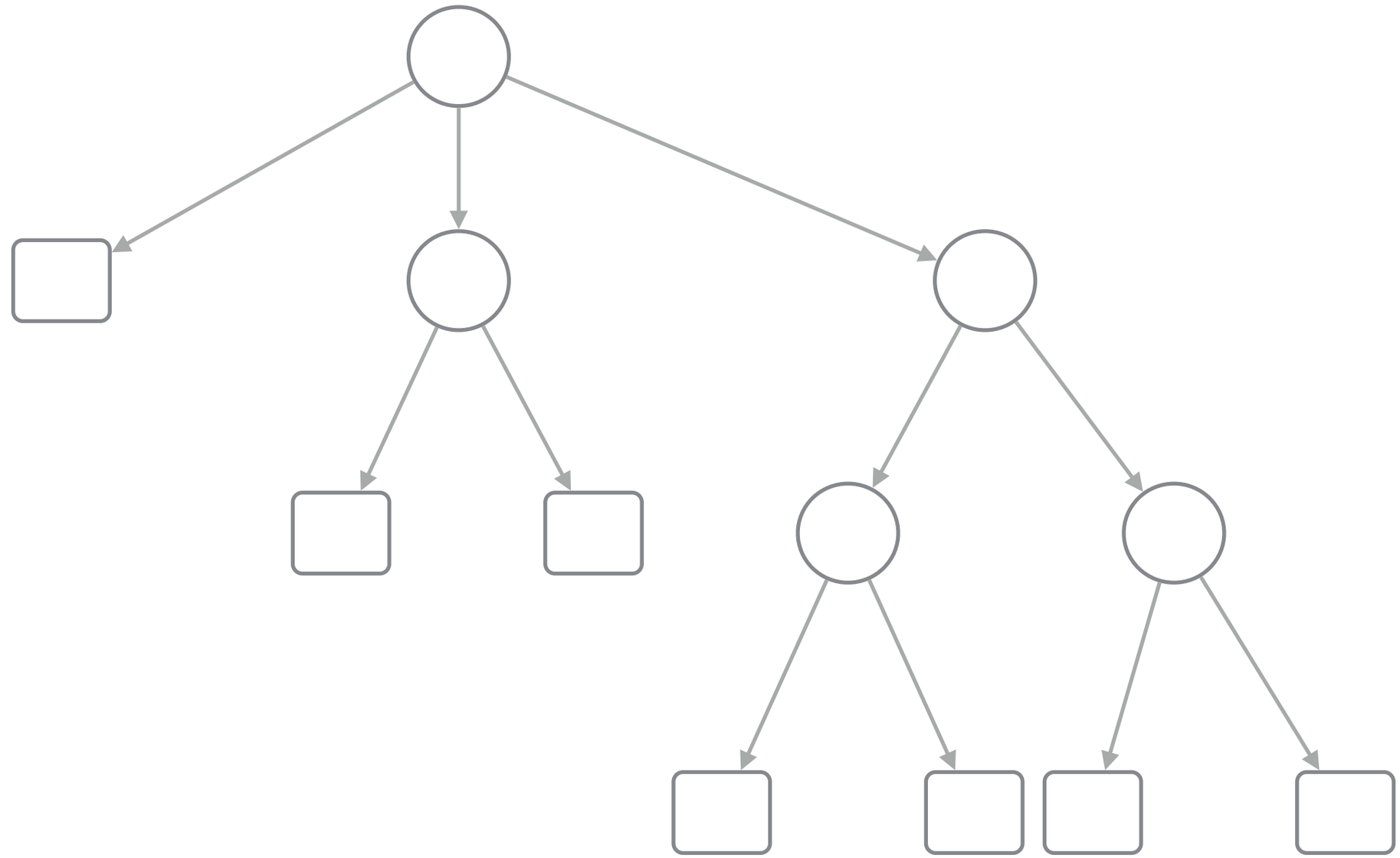
[LOUDS - Level-order unary degree sequence]



Succinct representation of trees

[LOUDS - Level-order unary degree sequence]

Trivial: O(n log n) bits

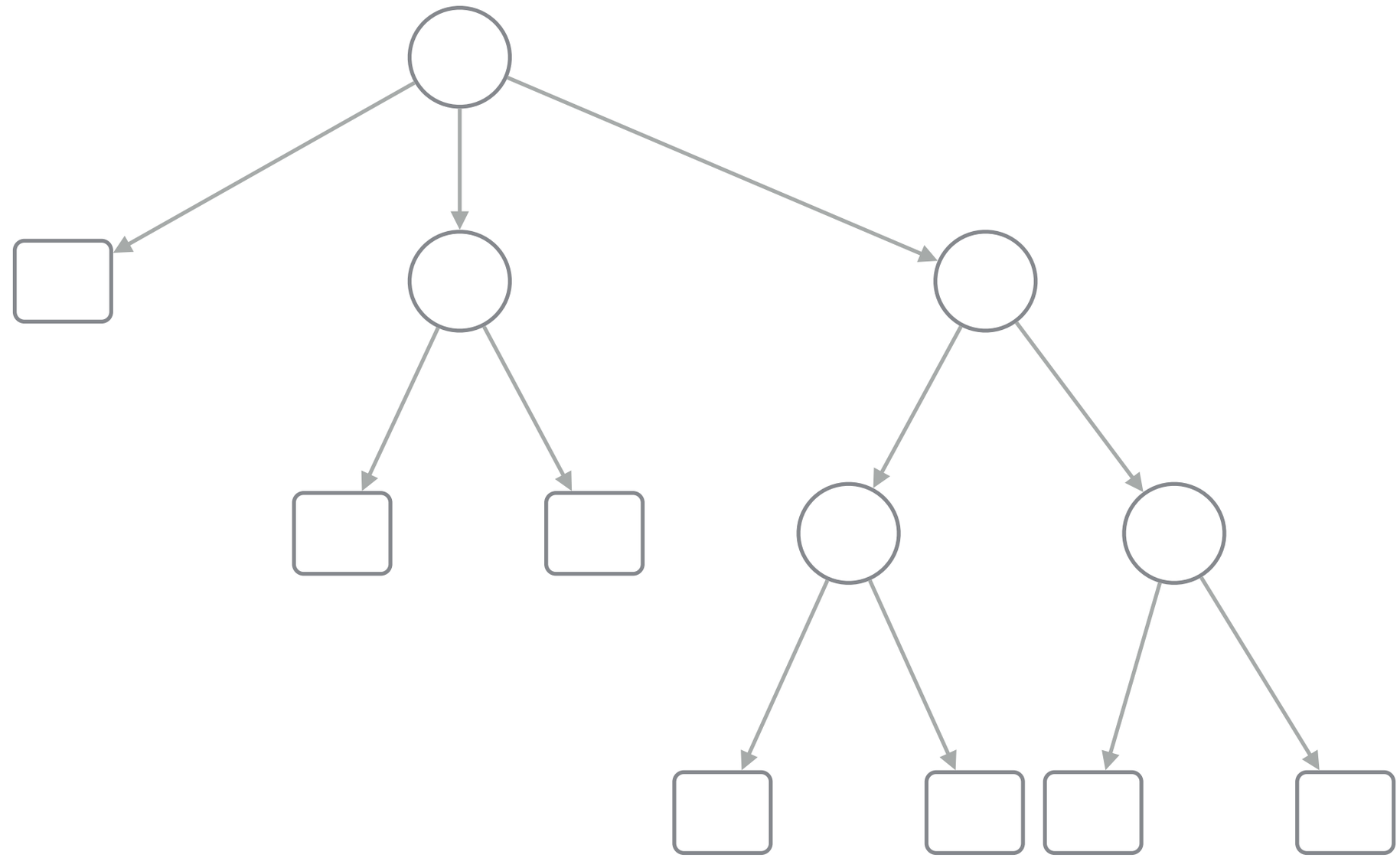


Succinct representation of trees

[LOUDS - Level-order unary degree sequence]

Trivial: O(n log n) bits

Best: 2n bits

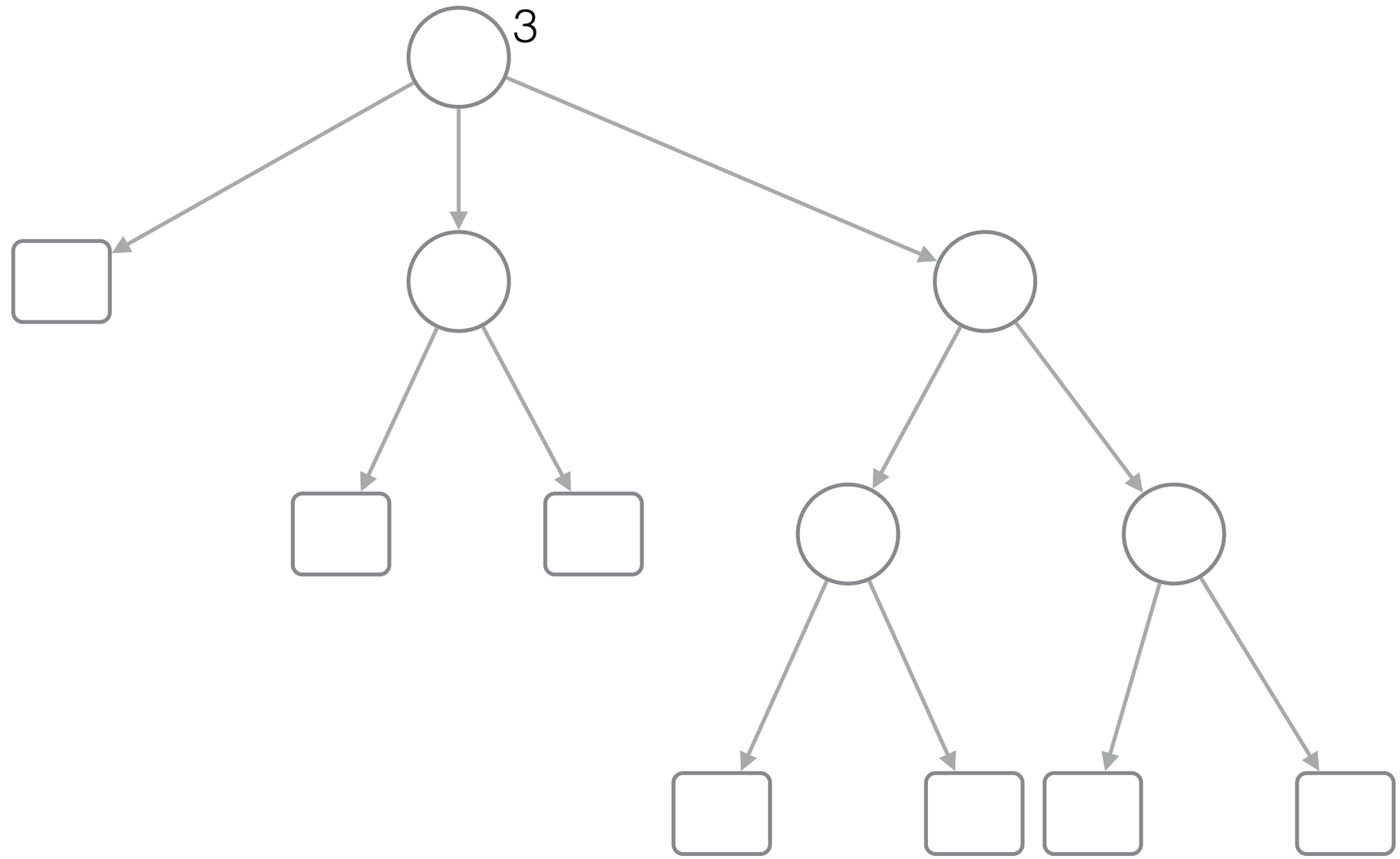


Succinct representation of trees

[LOUDS - Level-order unary degree sequence]

Trivial: O(n log n) bits

Best: 2n bits

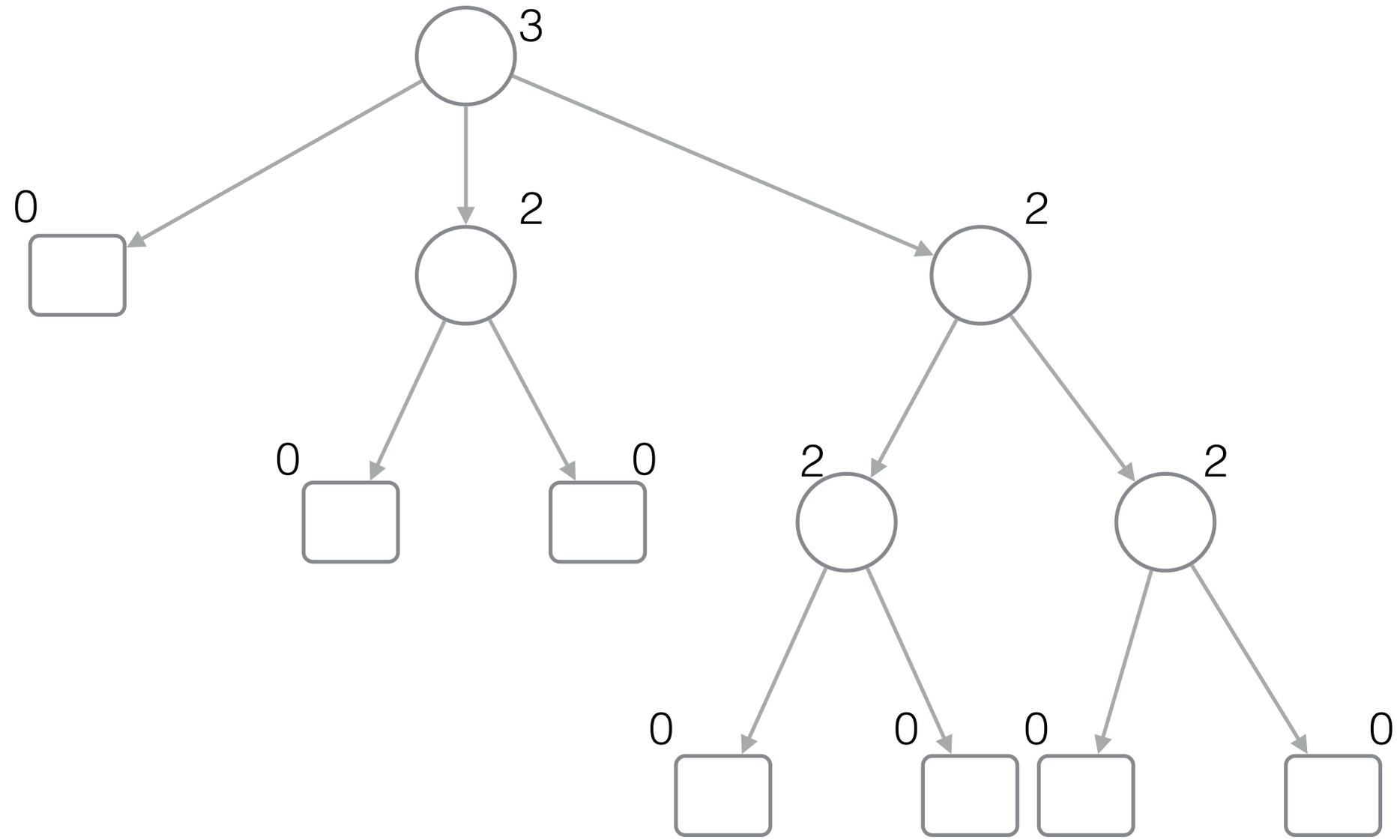


Succinct representation of trees

[LOUDS - Level-order unary degree sequence]

Trivial: O(n log n) bits

Best: 2n bits

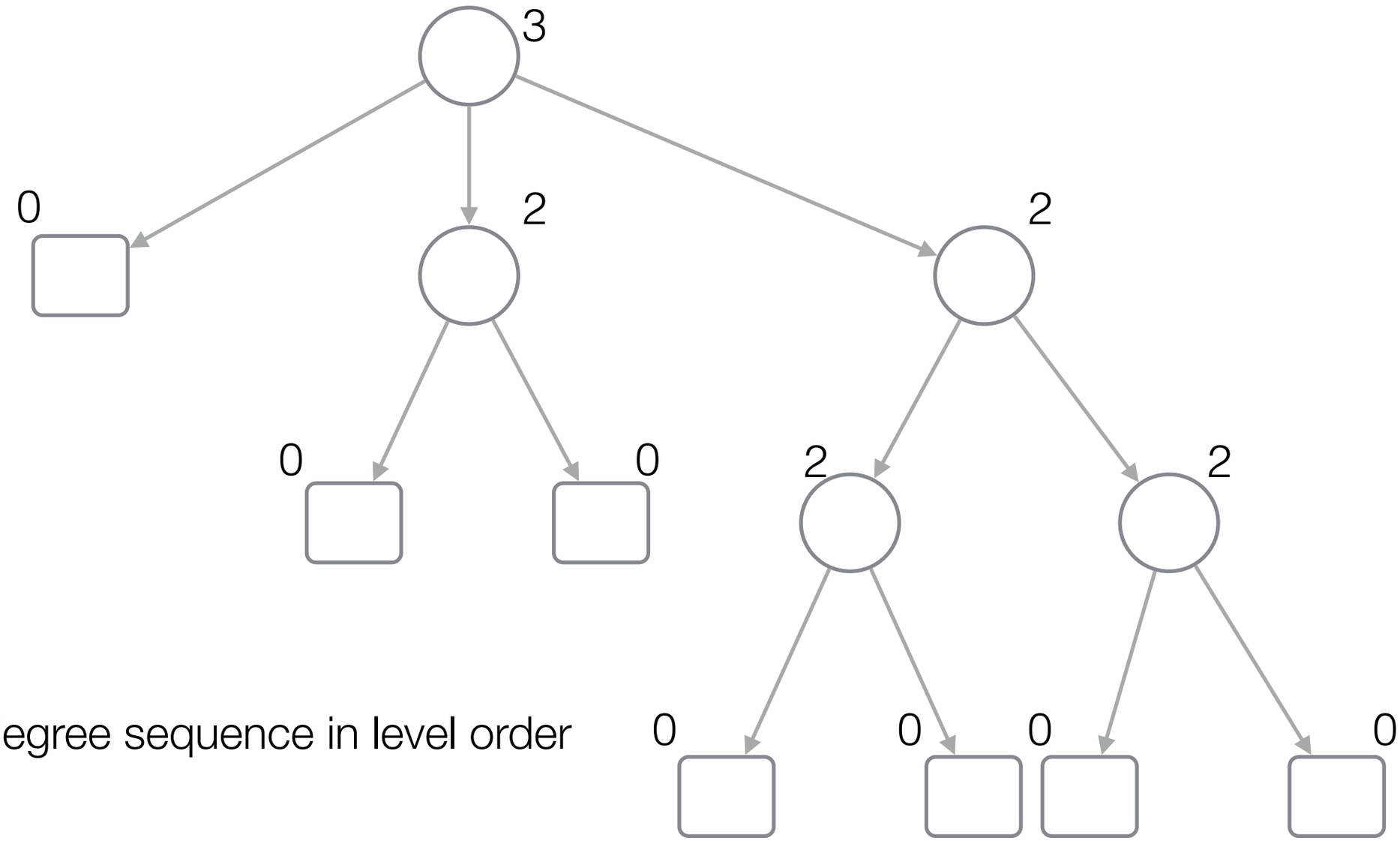


Succinct representation of trees

[LOUDS - Level-order unary degree sequence]

Trivial: O(n log n) bits

Best: 2n bits



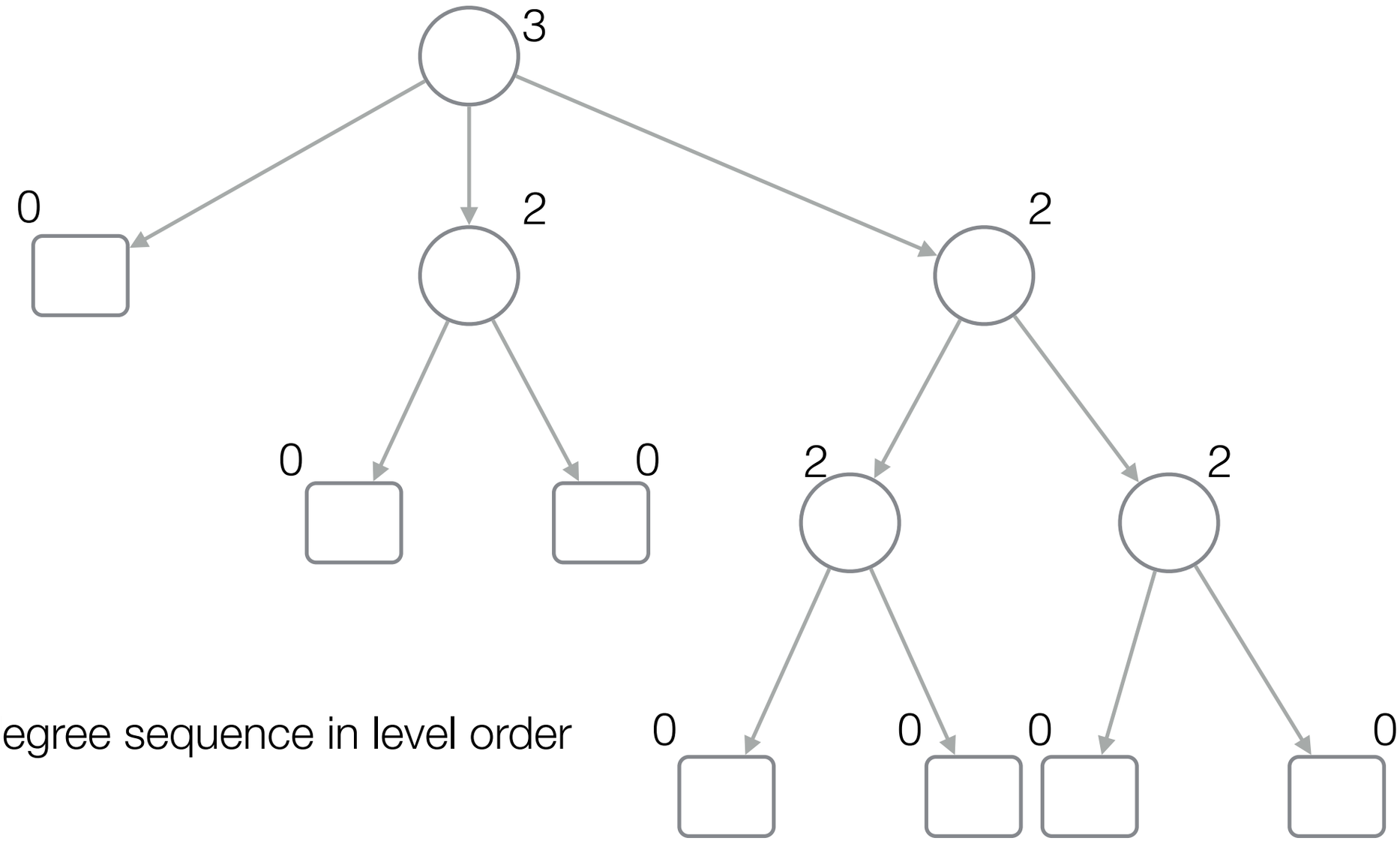
Write the degree sequence in level order

Succinct representation of trees

[LOUDS - Level-order unary degree sequence]

Trivial: O(n log n) bits

Best: 2n bits



Write the degree sequence in level order

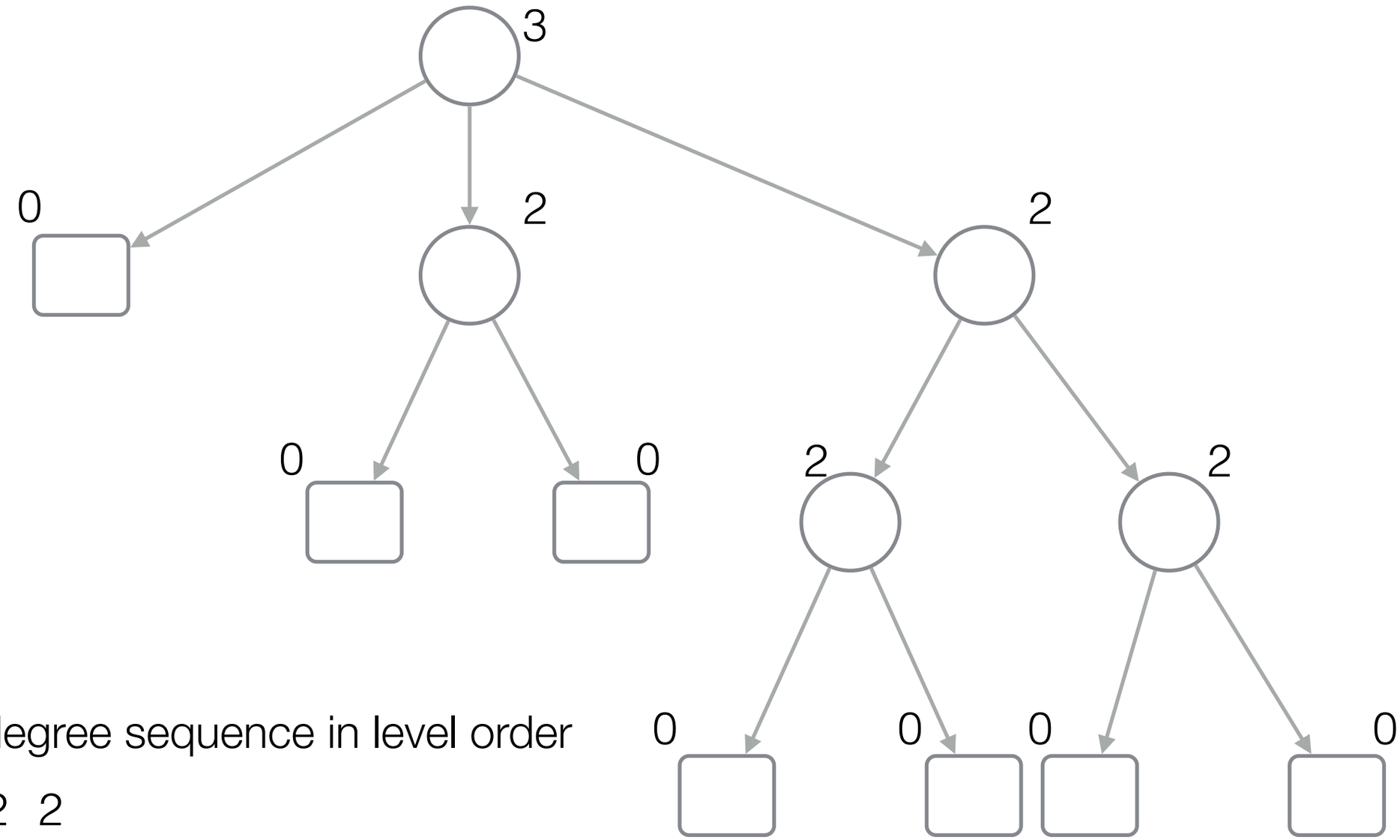
D 3

Succinct representation of trees

[LOUDS - Level-order unary degree sequence]

Trivial: O(n log n) bits

Best: 2n bits



Write the degree sequence in level order

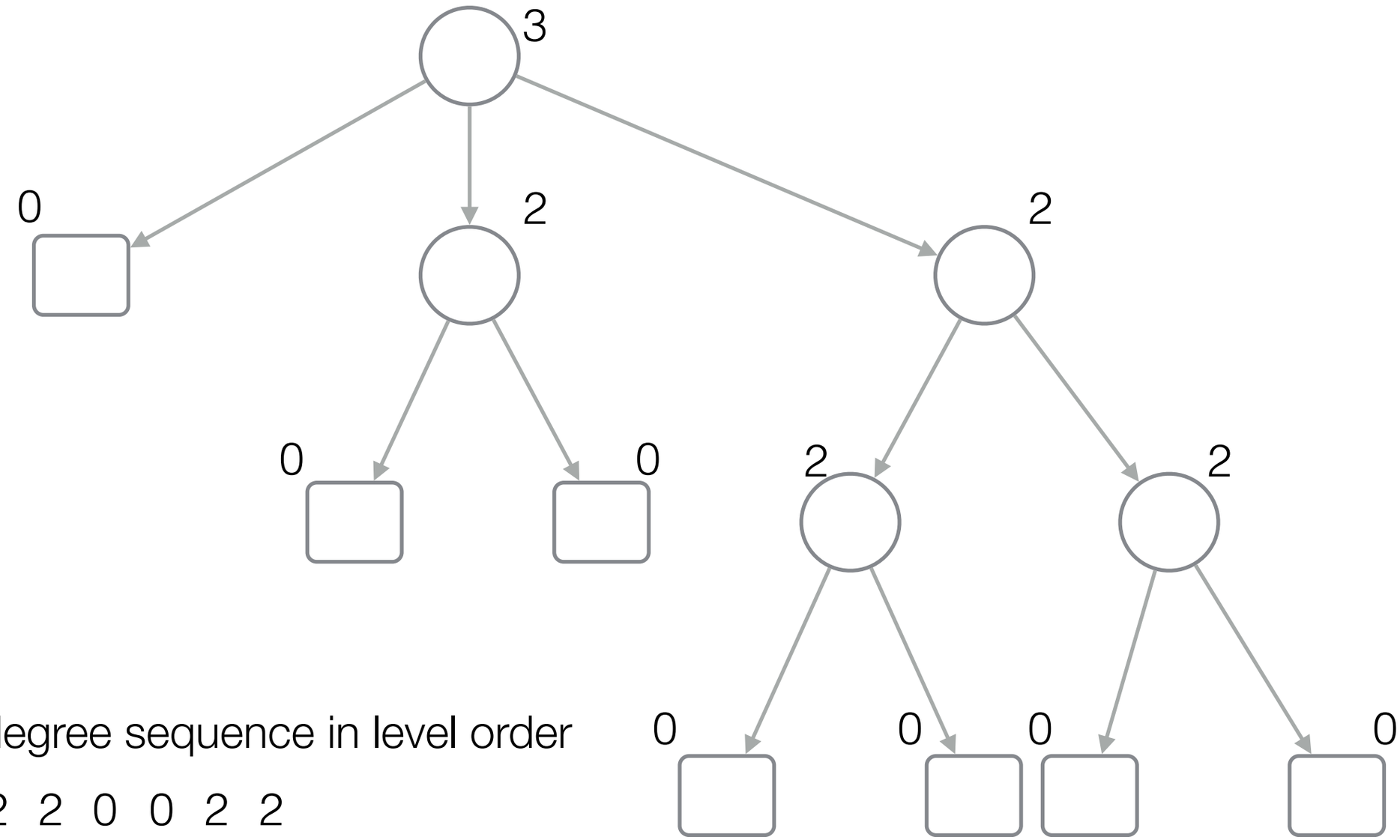
D 3 0 2 2

Succinct representation of trees

[LOUDS - Level-order unary degree sequence]

Trivial: O(n log n) bits

Best: 2n bits



Write the degree sequence in level order

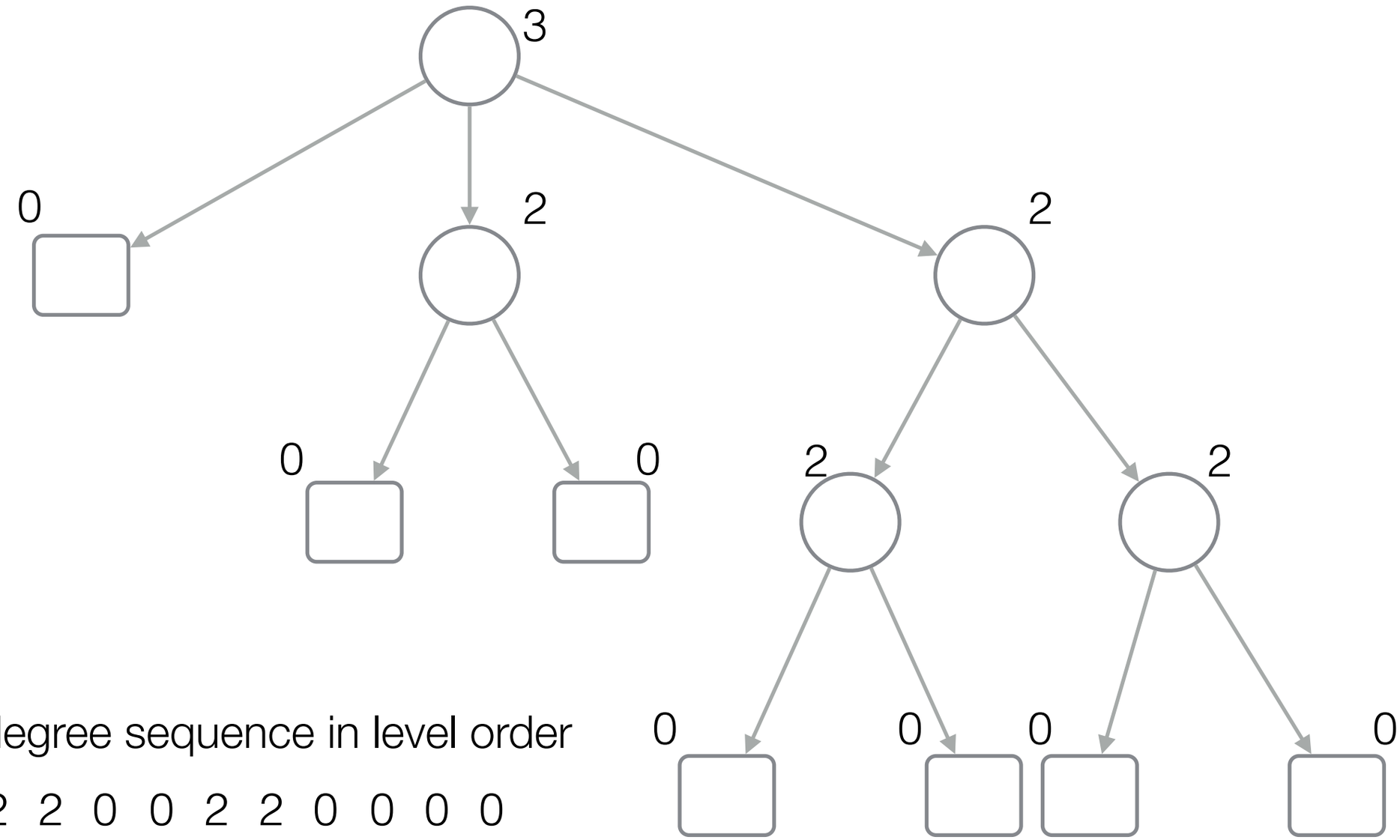
D 3 0 2 2 0 0 2 2

Succinct representation of trees

[LOUDS - Level-order unary degree sequence]

Trivial: O(n log n) bits

Best: 2n bits



Write the degree sequence in level order

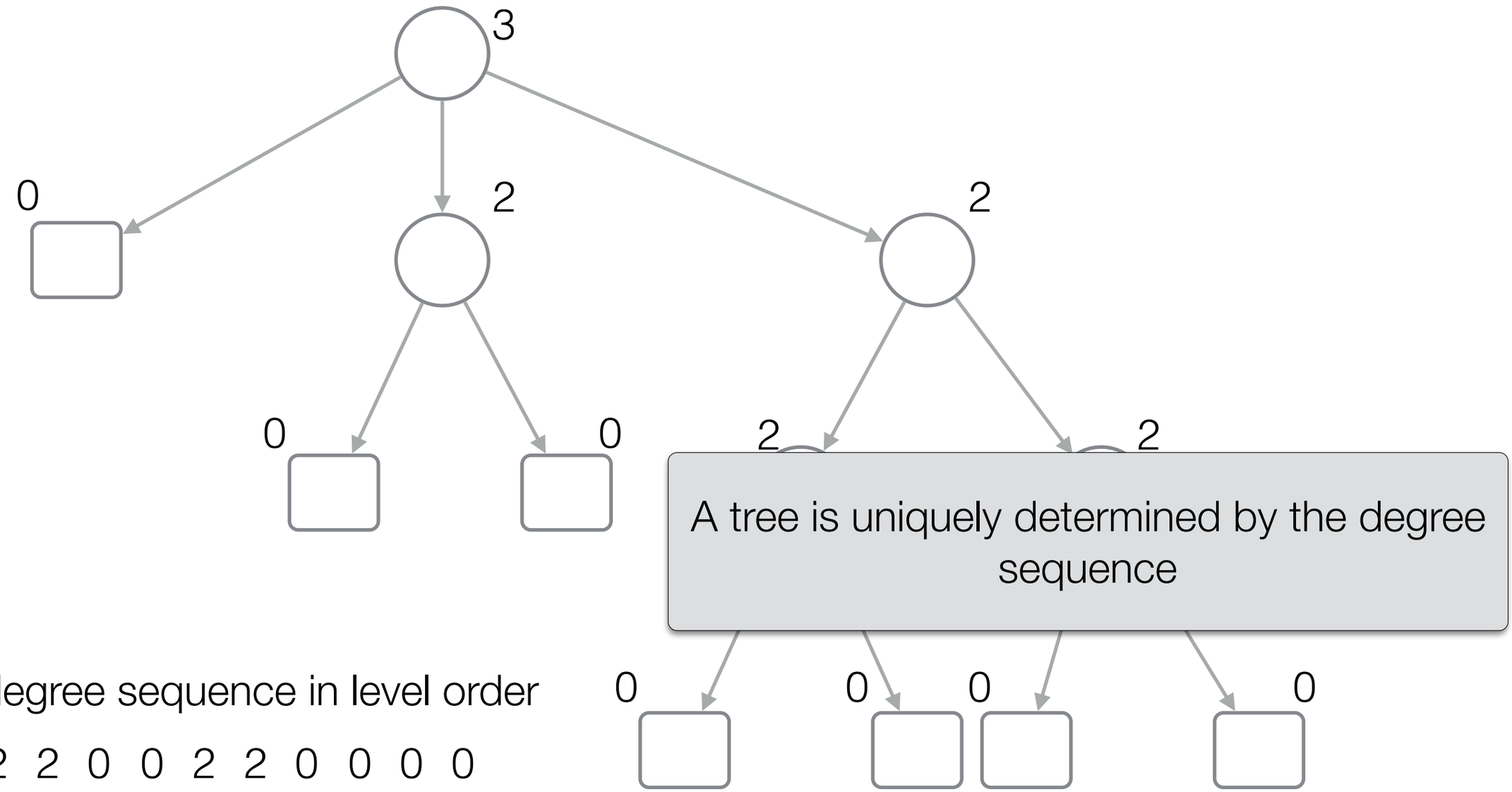
D 3 0 2 2 0 0 2 2 0 0 0 0

Succinct representation of trees

[LOUDS - Level-order unary degree sequence]

Trivial: O(n log n) bits

Best: 2n bits



A tree is uniquely determined by the degree sequence

Write the degree sequence in level order

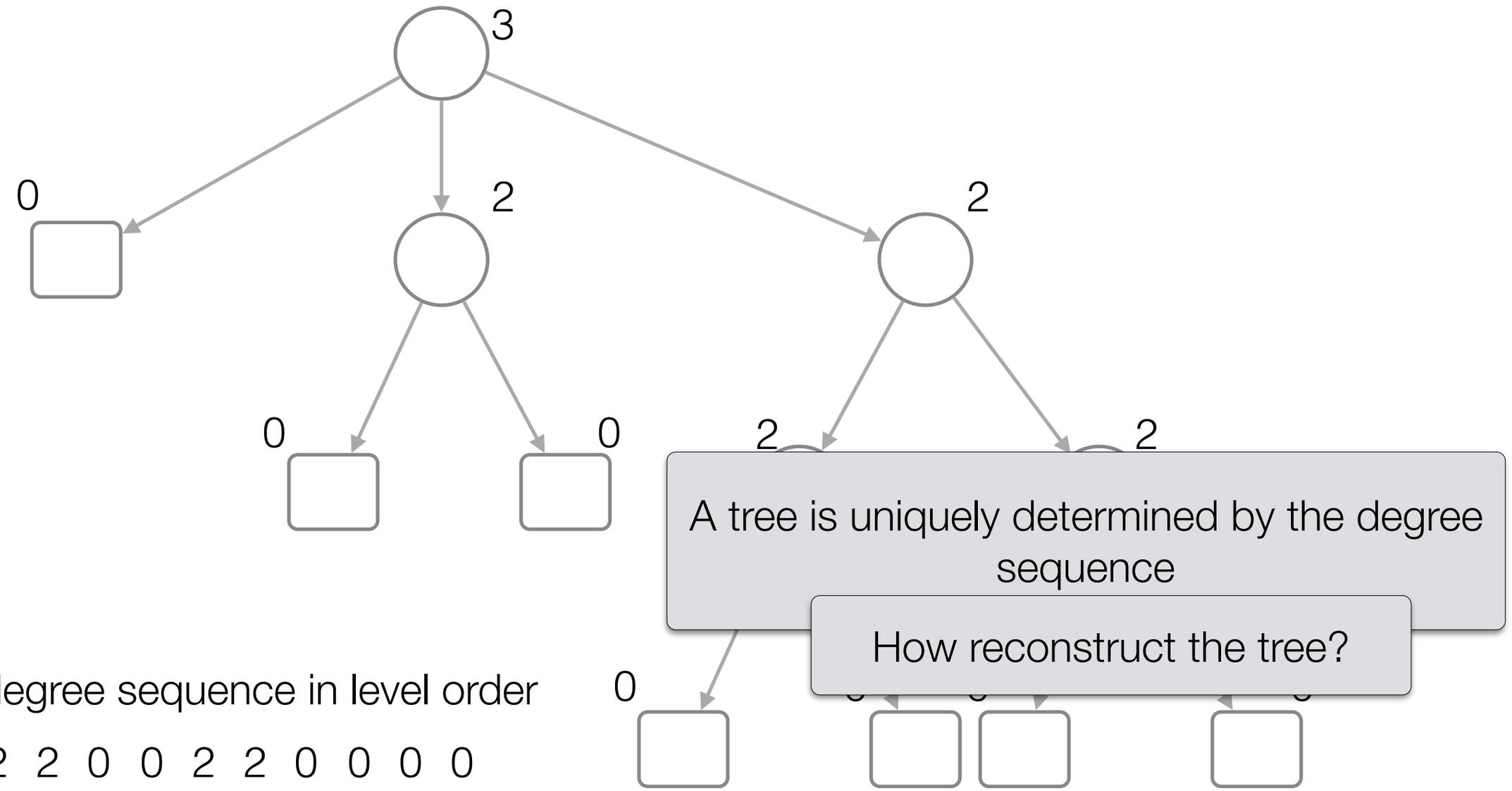
D 3 0 2 2 0 0 2 2 0 0 0 0

Succinct representation of trees

[LOUDS - Level-order unary degree sequence]

Trivial: O(n log n) bits

Best: 2n bits



A tree is uniquely determined by the degree sequence

How reconstruct the tree?

Write the degree sequence in level order

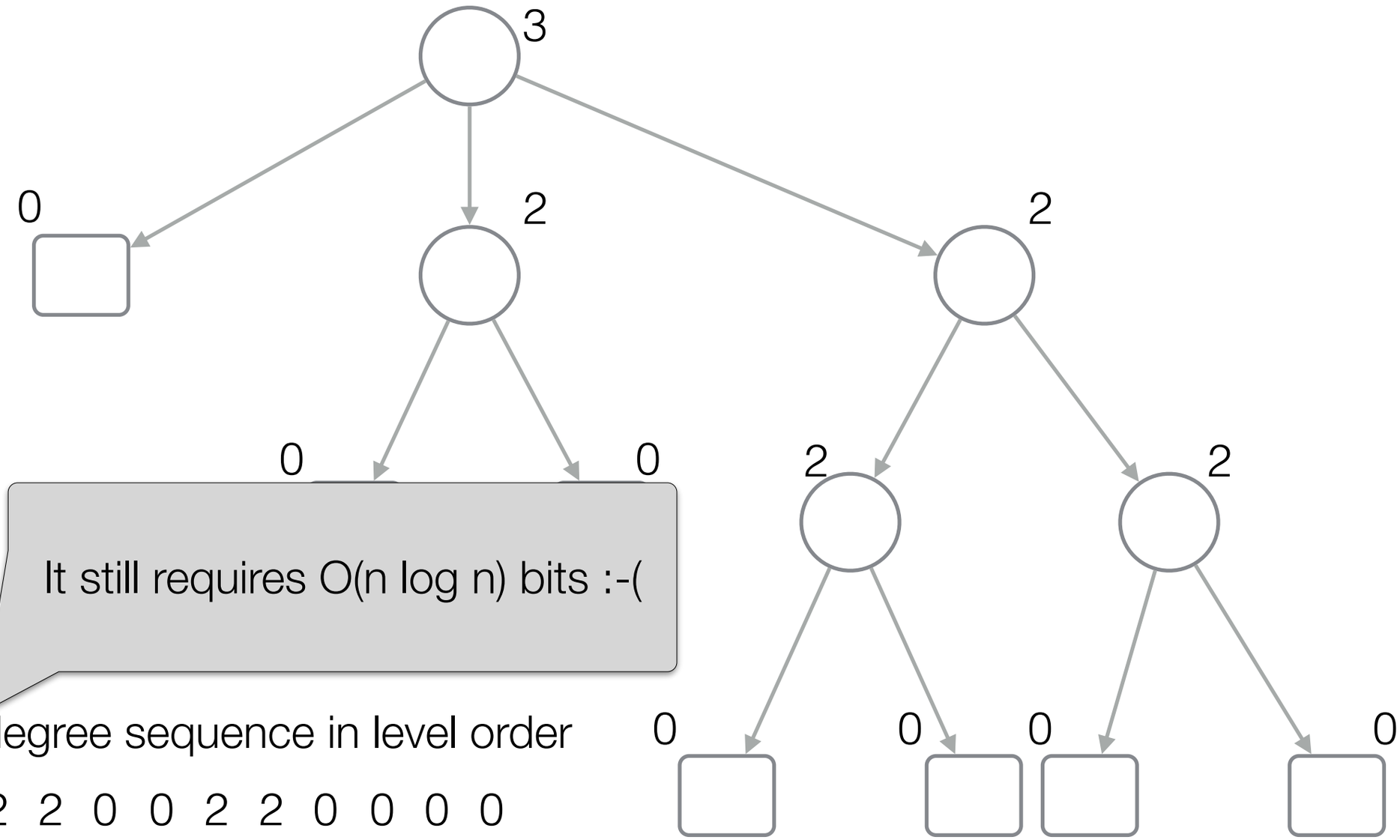
D 3 0 2 2 0 0 2 2 0 0 0 0

Succinct representation of trees

[LOUDS - Level-order unary degree sequence]

Trivial: O(n log n) bits

Best: 2n bits



It still requires O(n log n) bits :-(

Write the degree sequence in level order

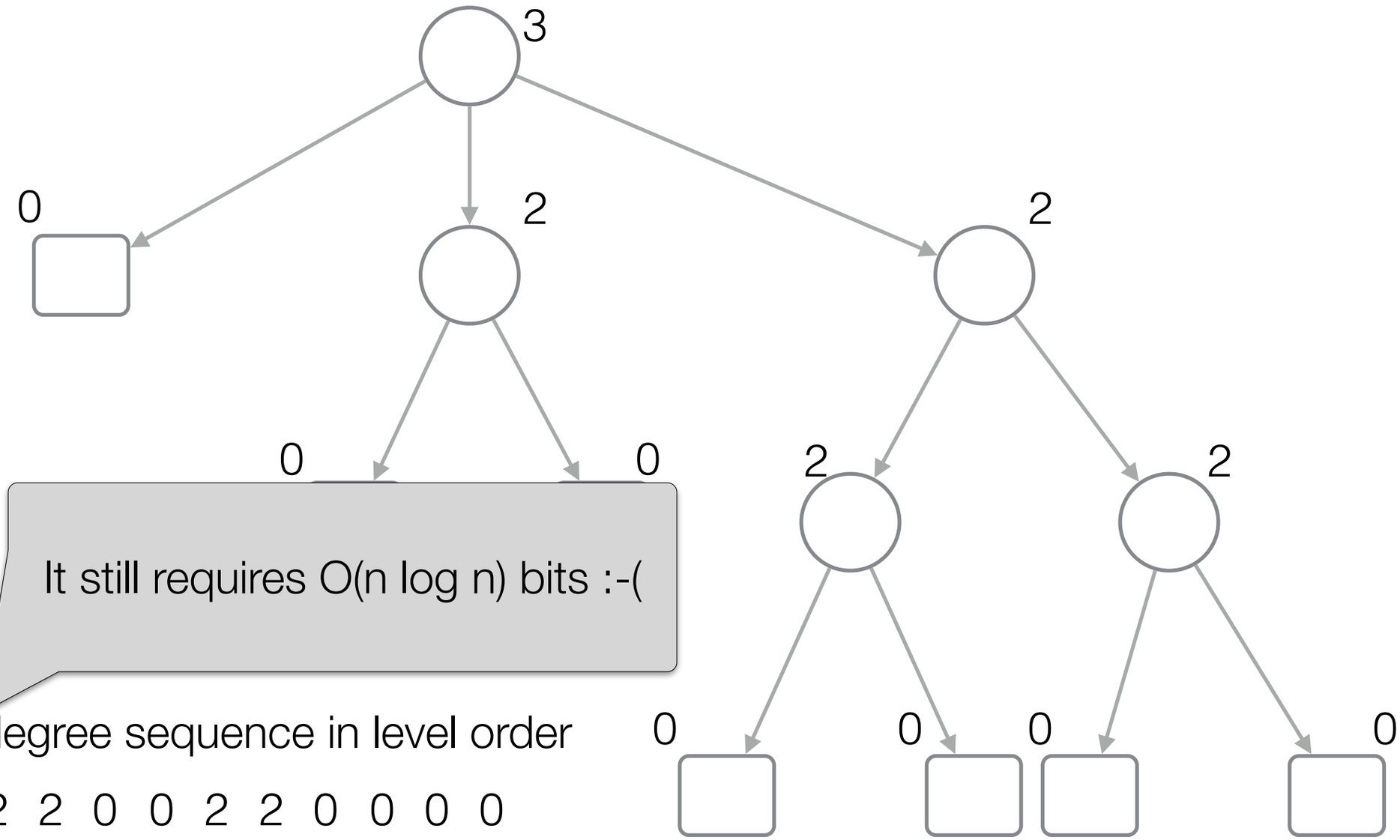
D 3 0 2 2 0 0 2 2 0 0 0 0

Succinct representation of trees

[LOUDS - Level-order unary degree sequence]

Trivial: O(n log n) bits

Best: 2n bits



It still requires O(n log n) bits :-(

Write the degree sequence in level order

D 3 0 2 2 0 0 2 2 0 0 0 0

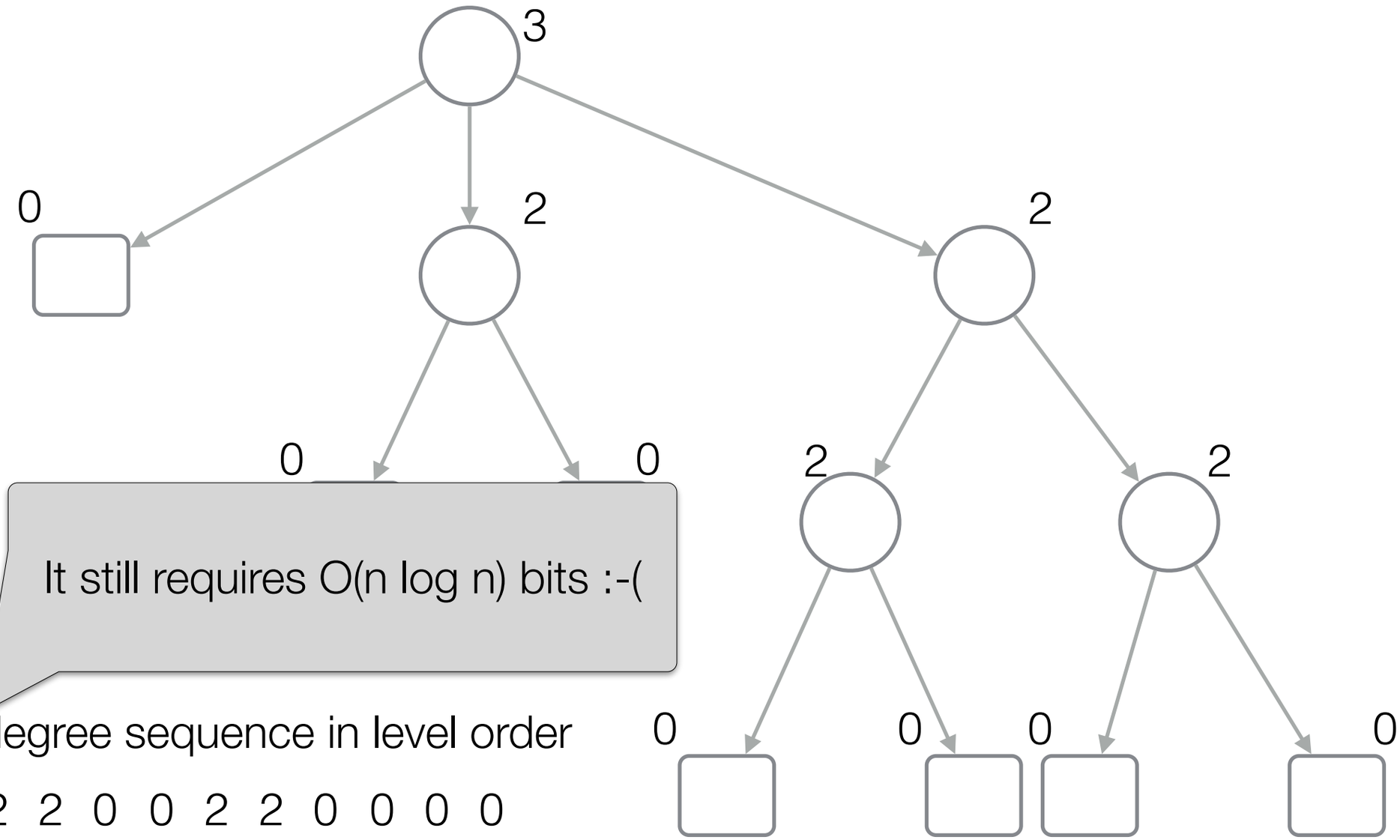
Solution: write them in unary

Succinct representation of trees

[LOUDS - Level-order unary degree sequence]

Trivial: O(n log n) bits

Best: 2n bits



It still requires O(n log n) bits :-(

Write the degree sequence in level order

D 3 0 2 2 0 0 2 2 0 0 0 0

Solution: write them in unary

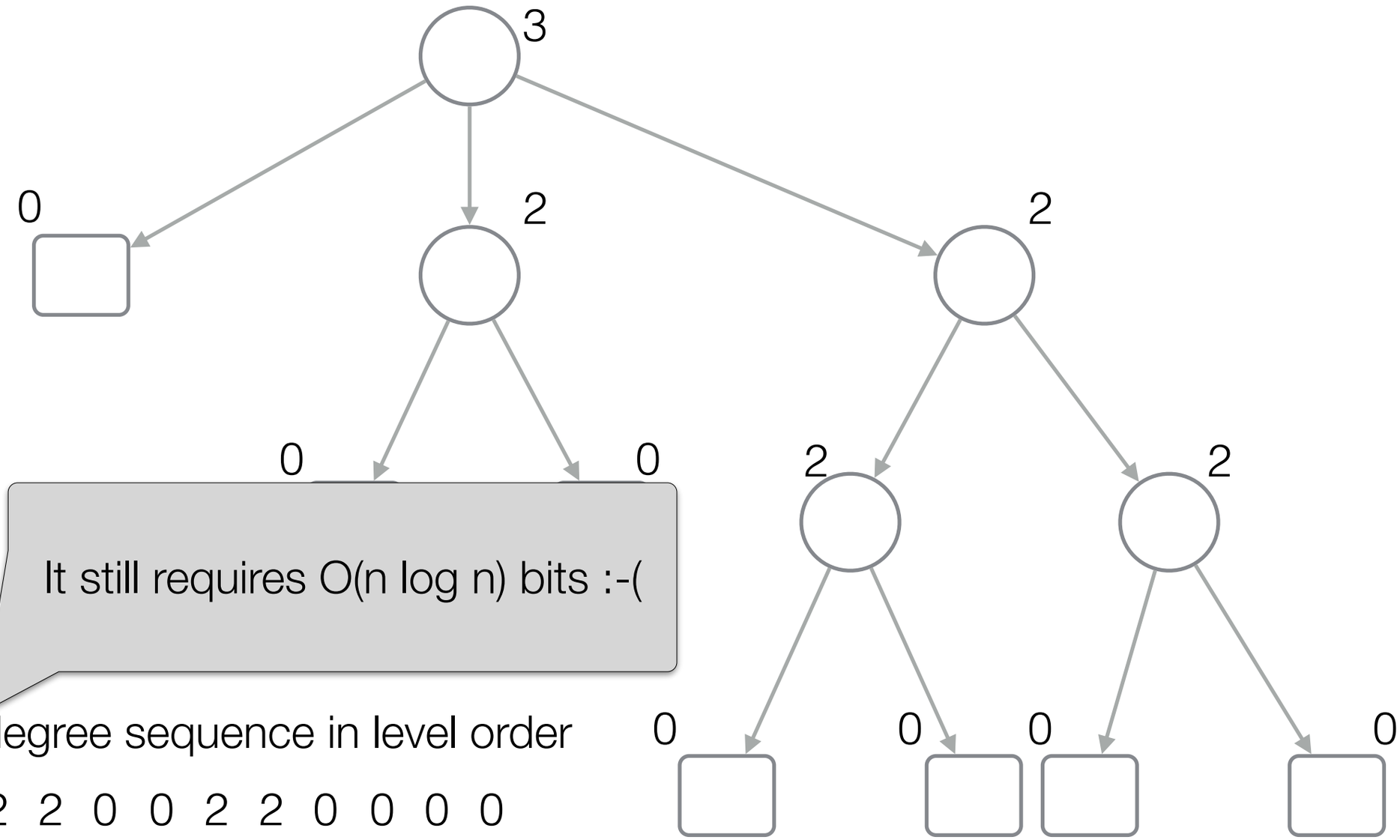
B

Succinct representation of trees

[LOUDS - Level-order unary degree sequence]

Trivial: O(n log n) bits

Best: 2n bits



It still requires O(n log n) bits :-(

Write the degree sequence in level order

D 3 0 2 2 0 0 2 2 0 0 0 0

Solution: write them in unary

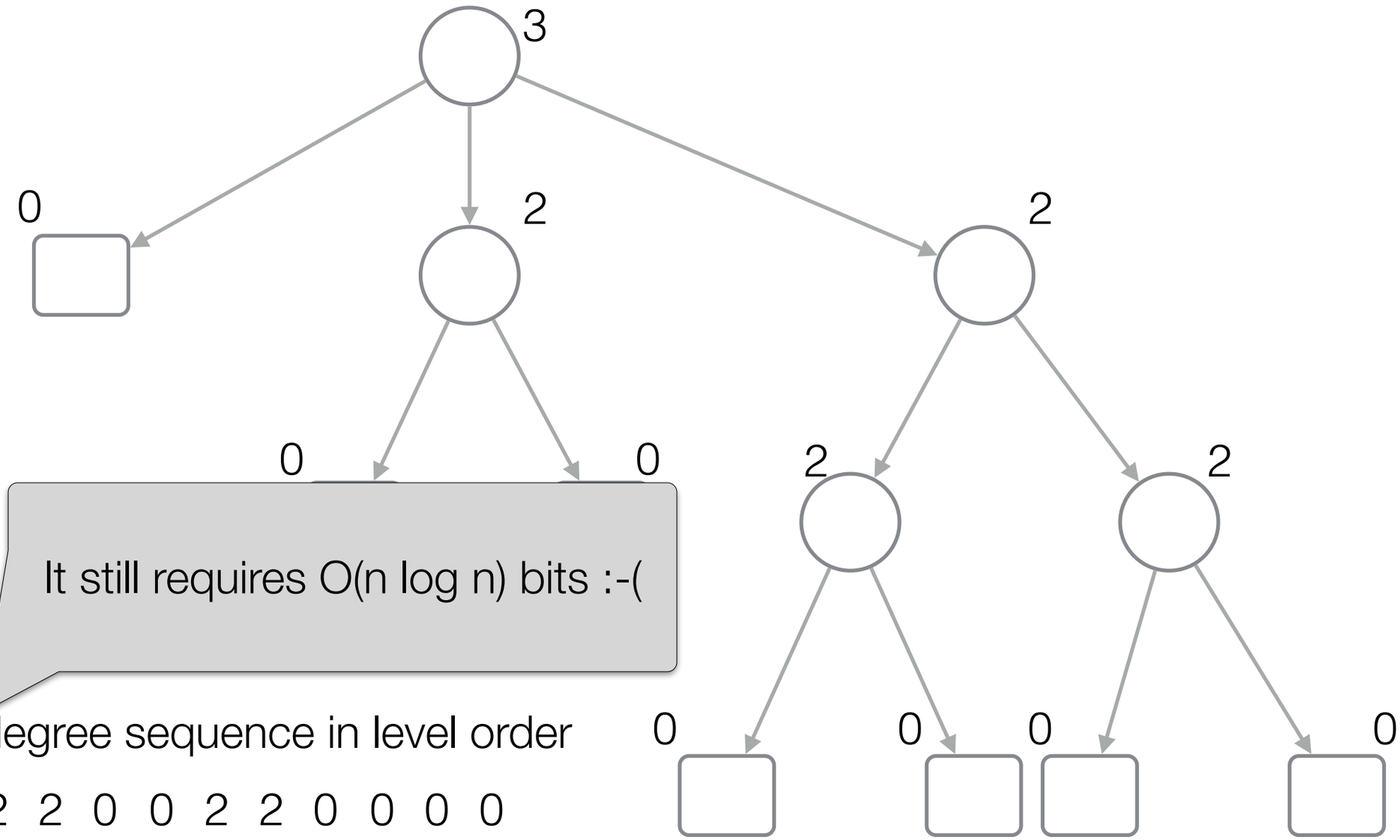
B 1110

Succinct representation of trees

[LOUDS - Level-order unary degree sequence]

Trivial: O(n log n) bits

Best: 2n bits



It still requires O(n log n) bits :-(

Write the degree sequence in level order

D 3 0 2 2 0 0 2 2 0 0 0 0

Solution: write them in unary

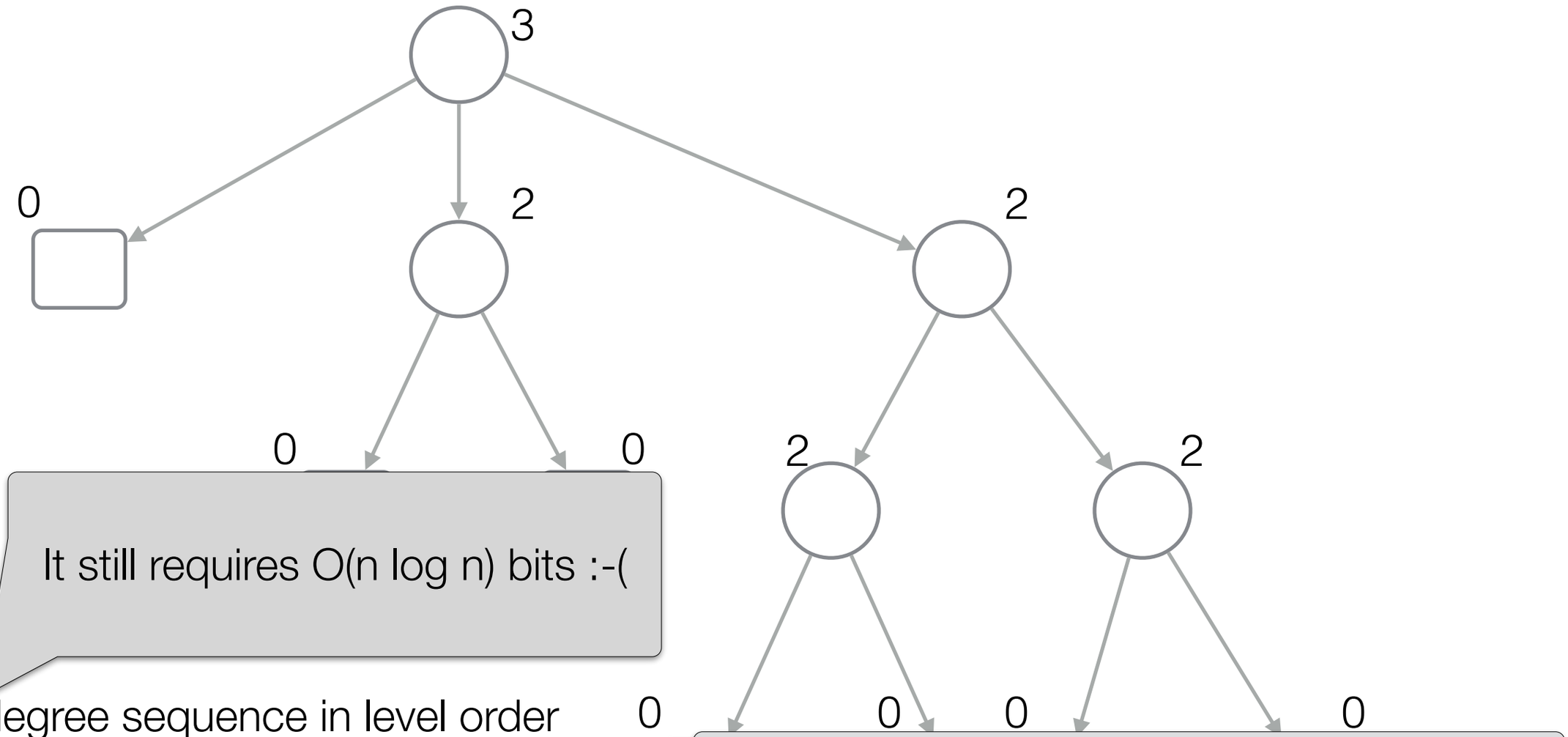
B 1110 0 110 110 0 0 110 110 0 0 0 0

Succinct representation of trees

[LOUDS - Level-order unary degree sequence]

Trivial: O(n log n) bits

Best: 2n bits



It still requires O(n log n) bits :-(

Write the degree sequence in level order

D 3 0 2 2 0 0 2 2 0 0 0 0

Solution: write them in unary

B 1110 0 110 110 0 0 110 110 0 0 0 0

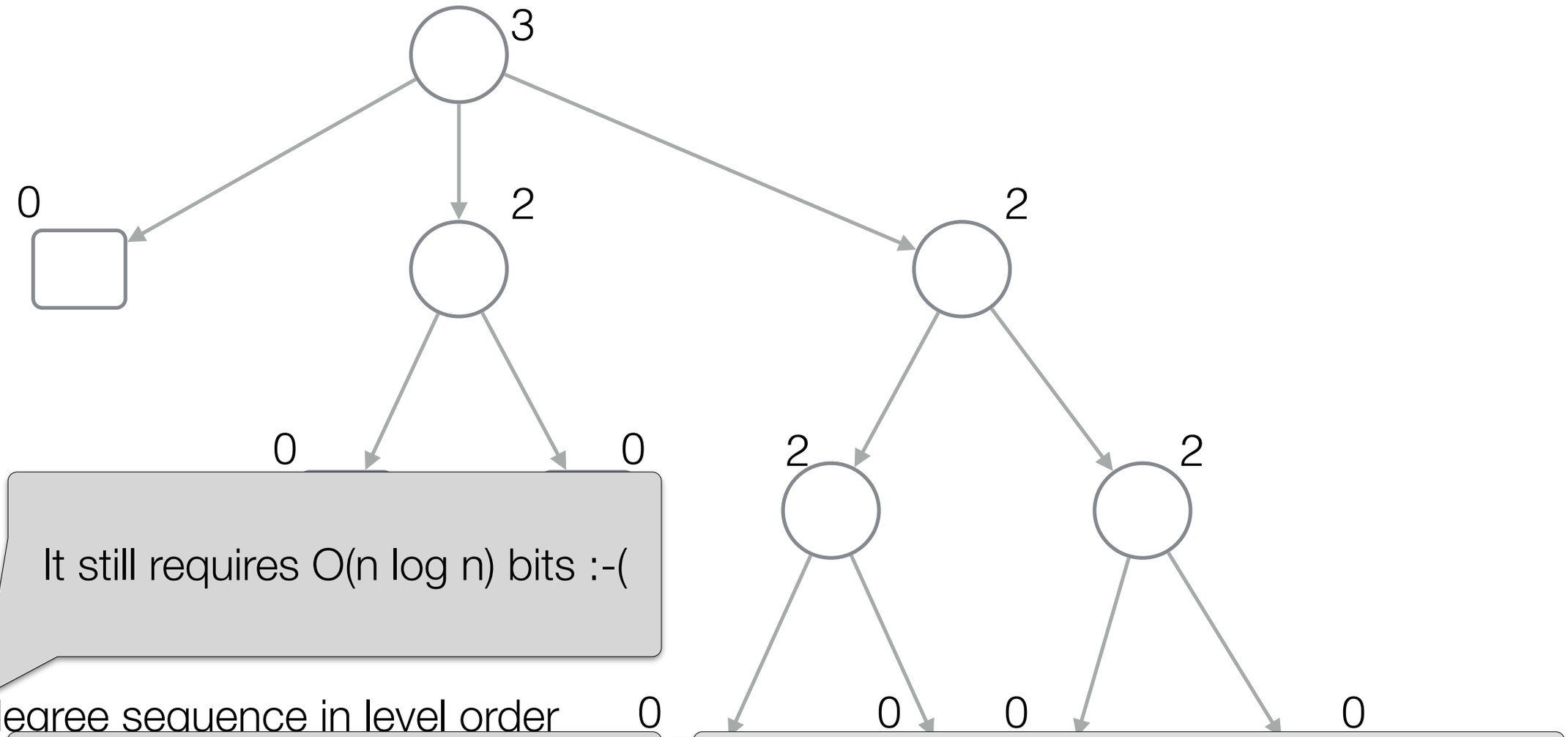
B takes 2n - 1 bits!
For each node we have a 0 and a 1
(but the root)

Succinct representation of trees

[LOUDS - Level-order unary degree sequence]

Trivial: O(n log n) bits

Best: 2n bits



It still requires O(n log n) bits :-(

Write the degree sequence in level order

D 3 0 2 0 2 2 0 0 0 0

Solution: write a 1 for each node and a 0 for each edge

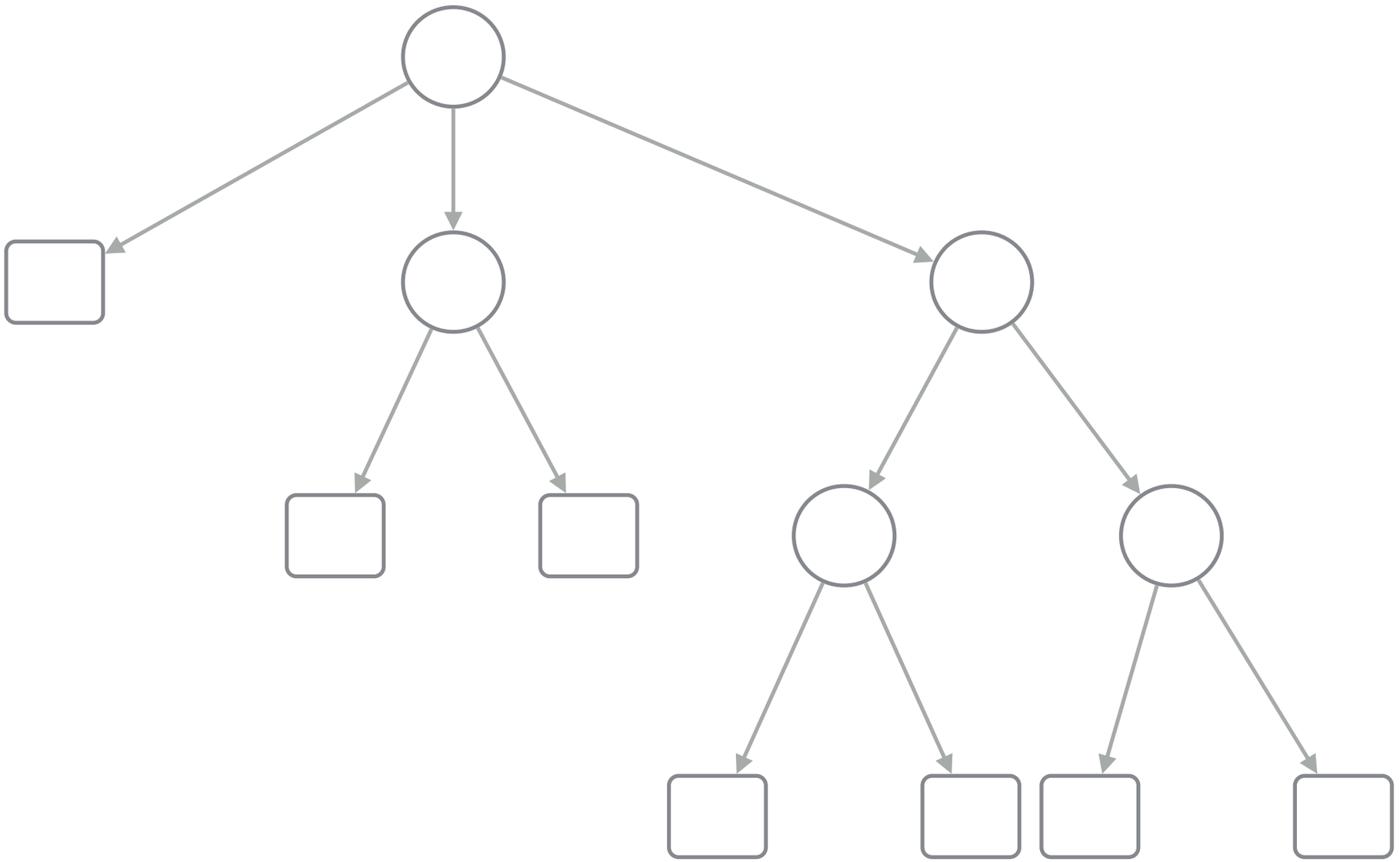
B 1110 0 110 110 0 0 110 110 0 0 0 0

Can we navigate the tree?

B takes 2n - 1 bits!
For each node we have a 0 and a 1
(but the root)

Succinct representation of trees

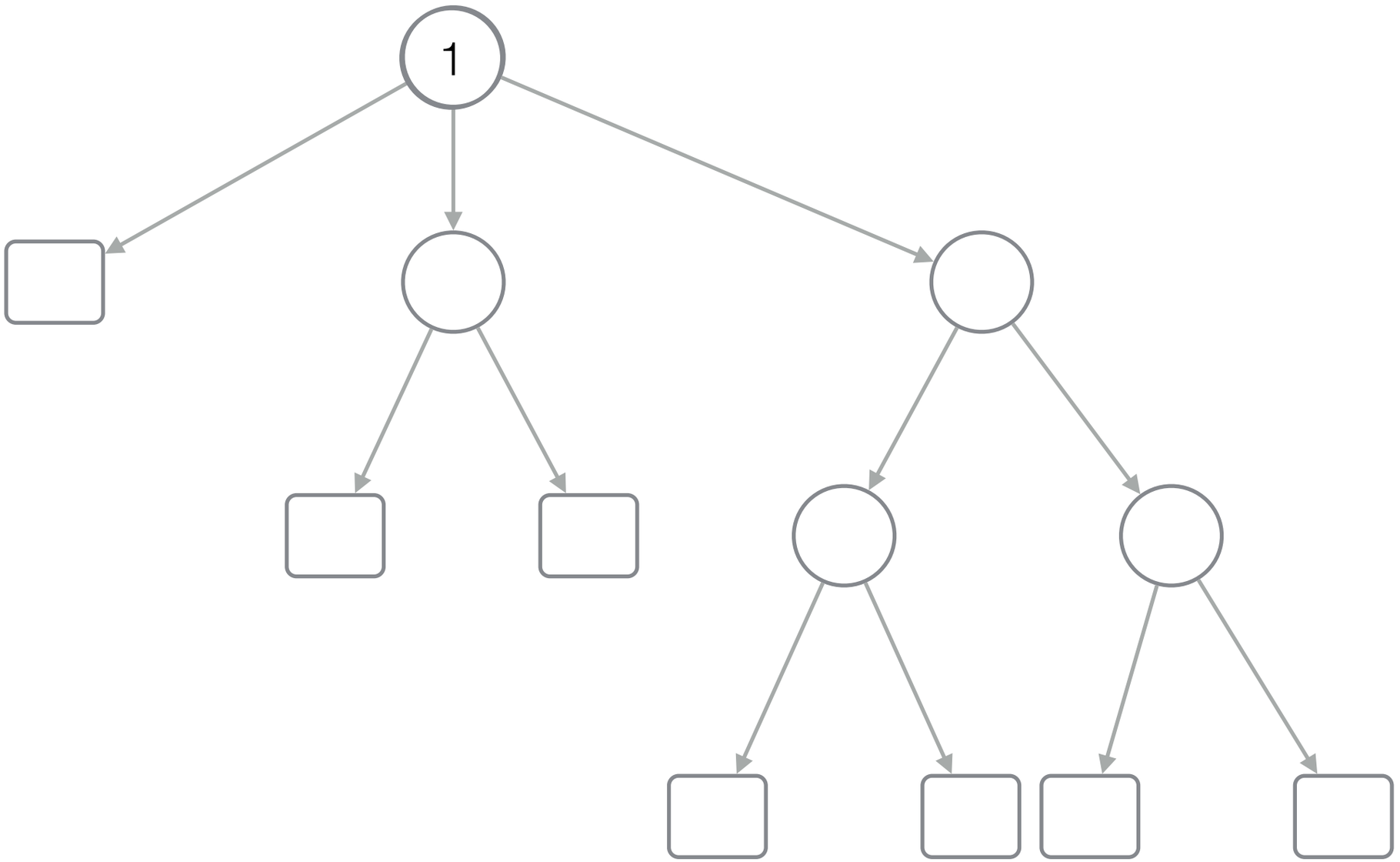
[LOUDS - Level-order unary degree sequence]



B 1 1 1 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0

Succinct representation of trees

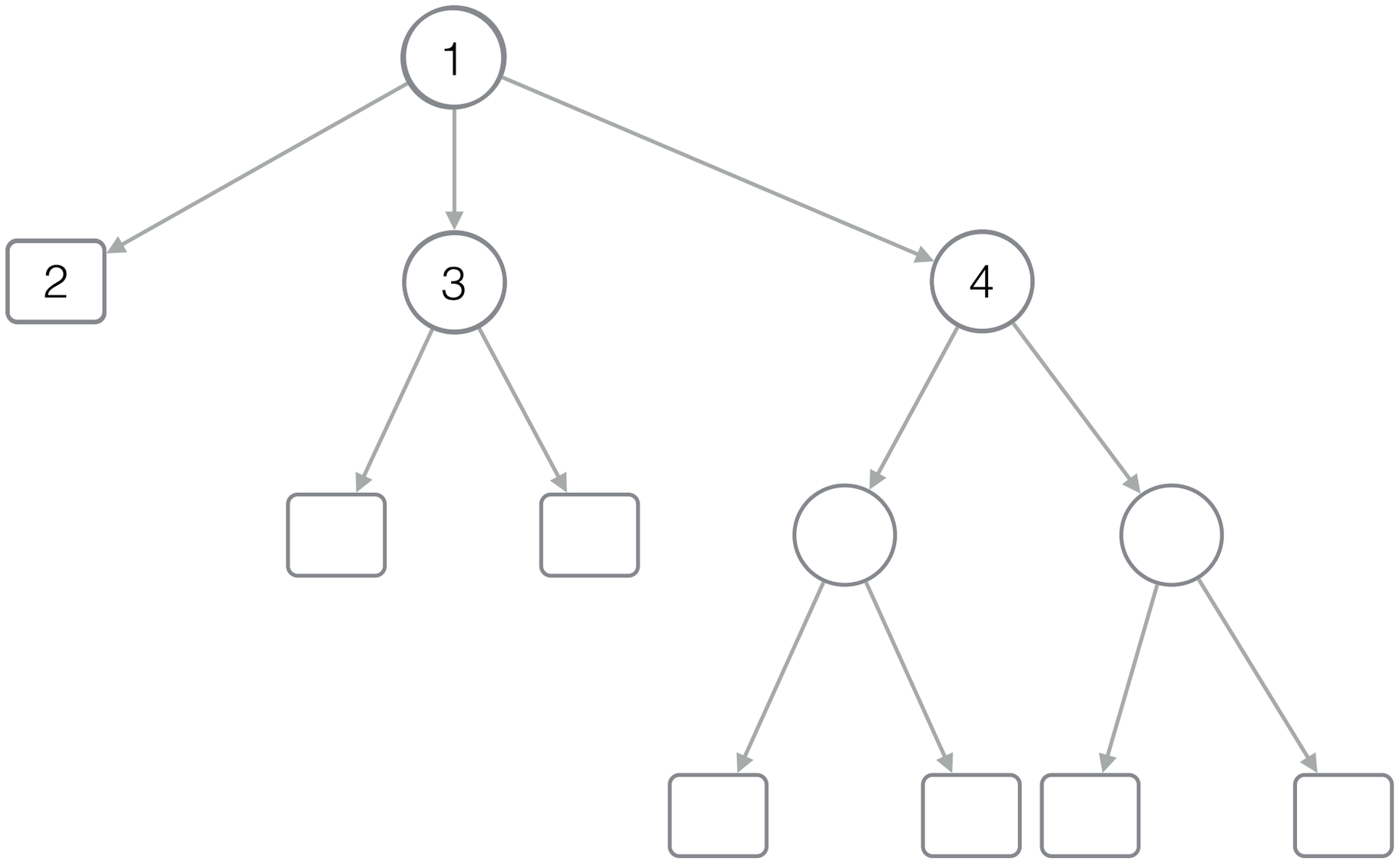
[LOUDS - Level-order unary degree sequence]



B 1 1 1 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0

Succinct representation of trees

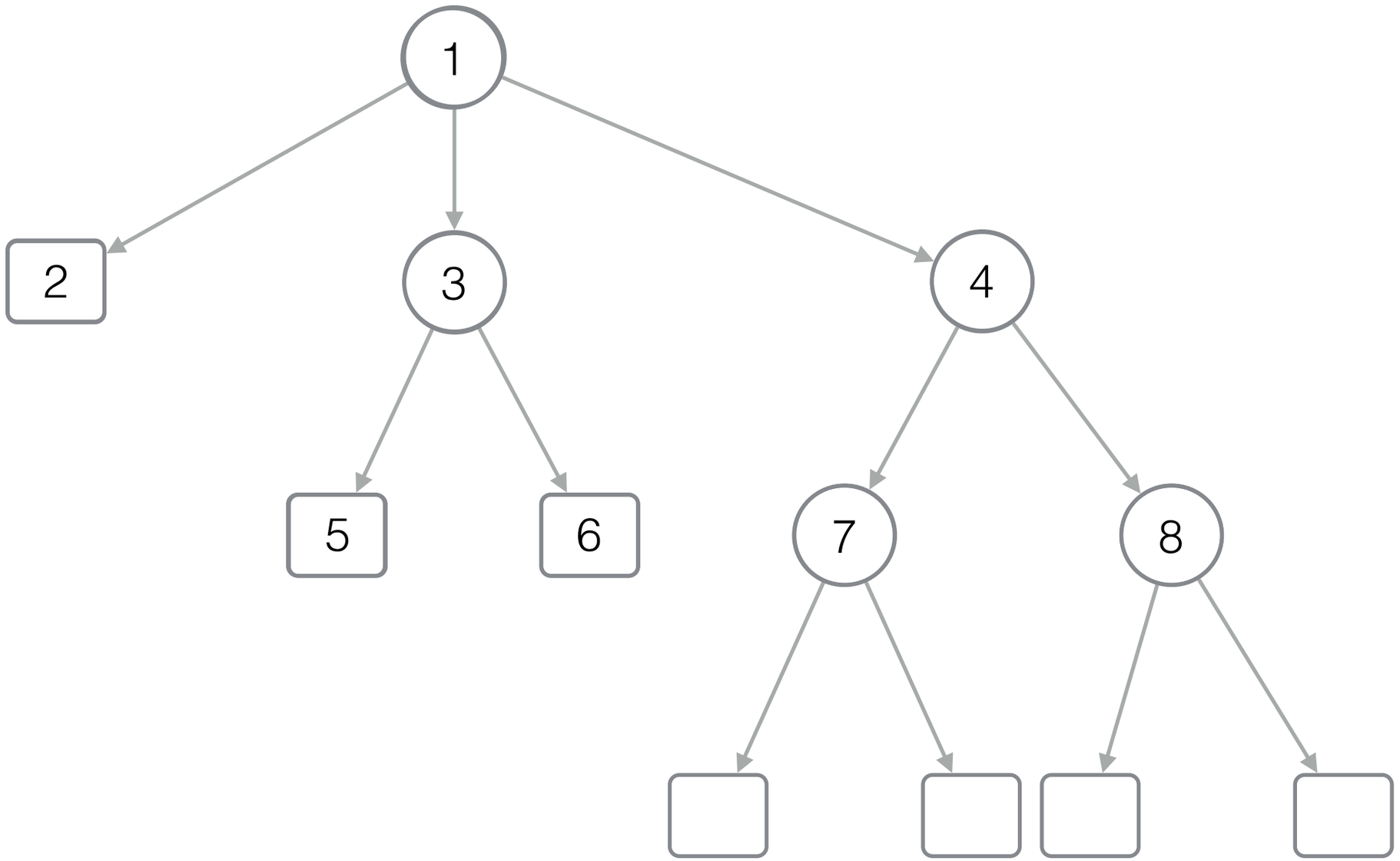
[LOUDS - Level-order unary degree sequence]



B 1 1 1 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0

Succinct representation of trees

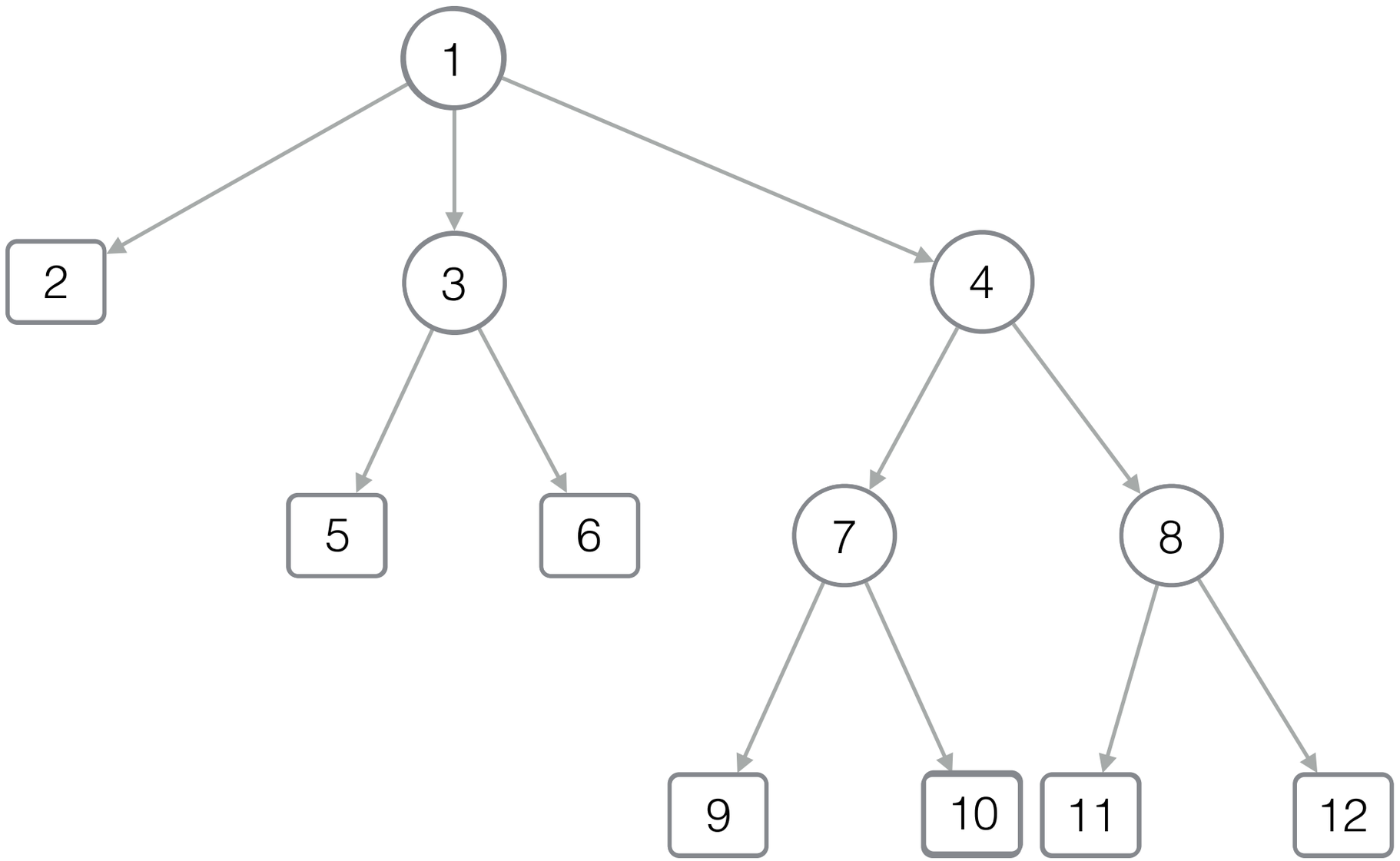
[LOUDS - Level-order unary degree sequence]



B 1 1 1 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0

Succinct representation of trees

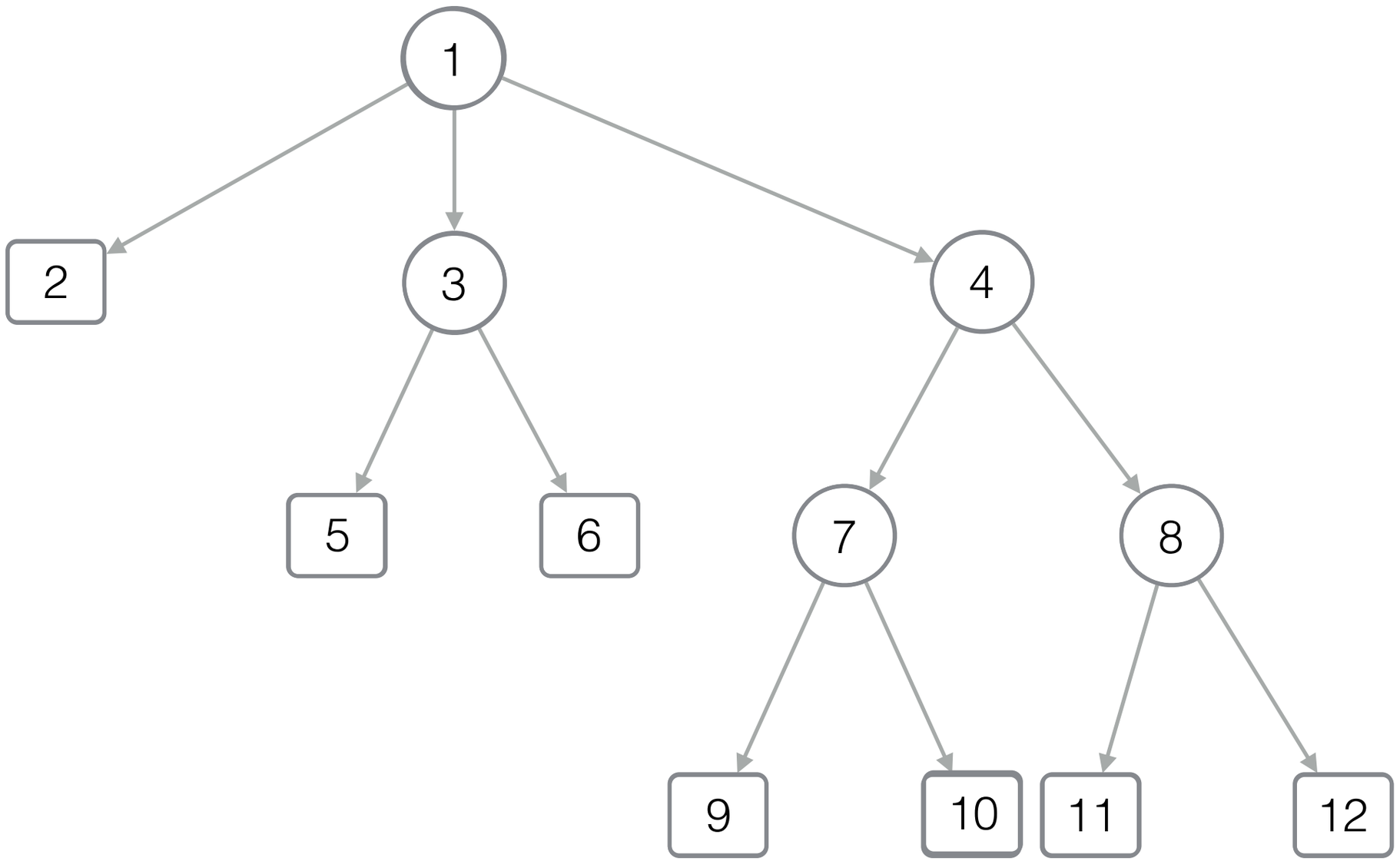
[LOUDS - Level-order unary degree sequence]



B 1 1 1 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0

Succinct representation of trees

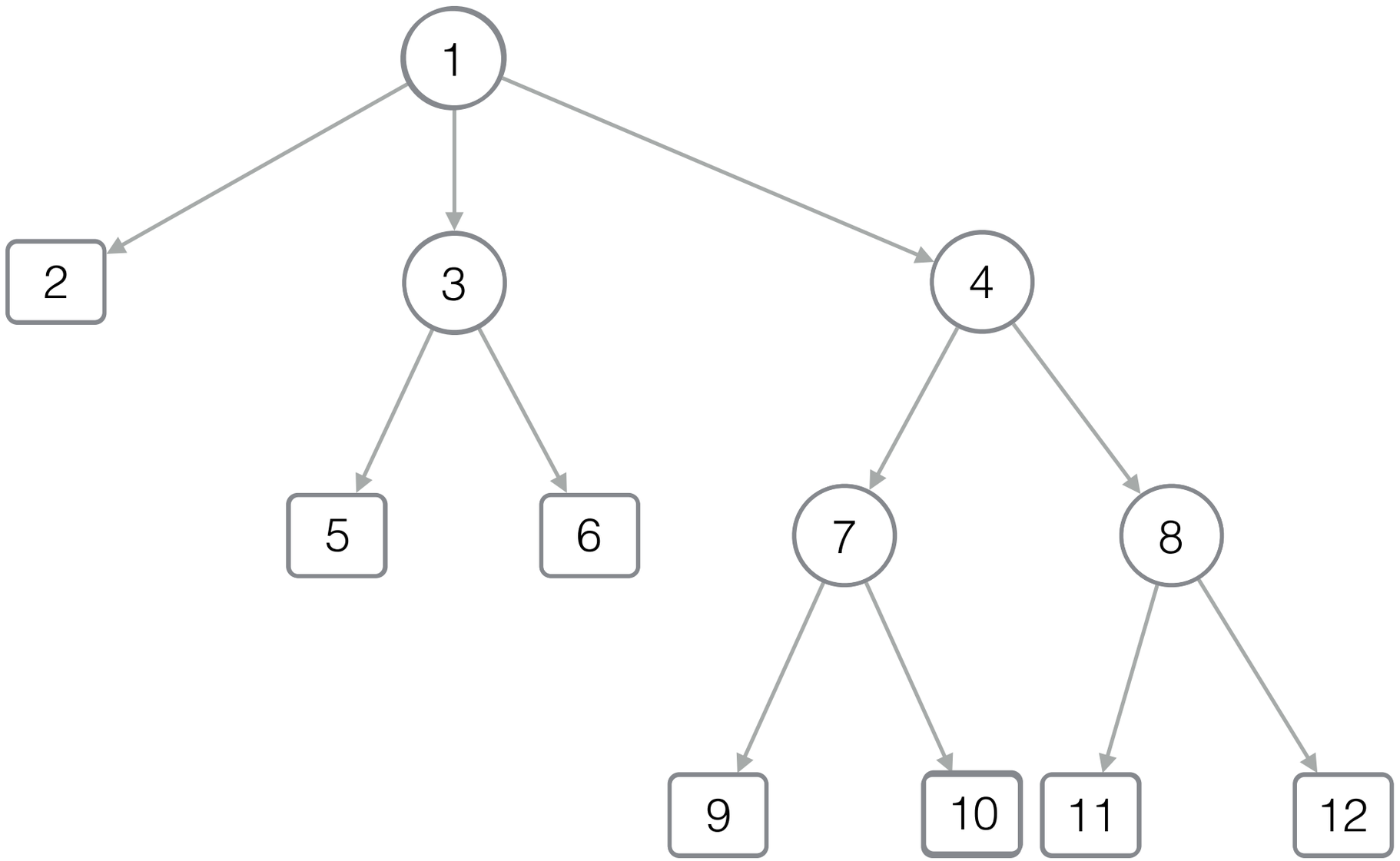
[LOUDS - Level-order unary degree sequence]



B 1 0 1 1 1 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0

Succinct representation of trees

[LOUDS - Level-order unary degree sequence]

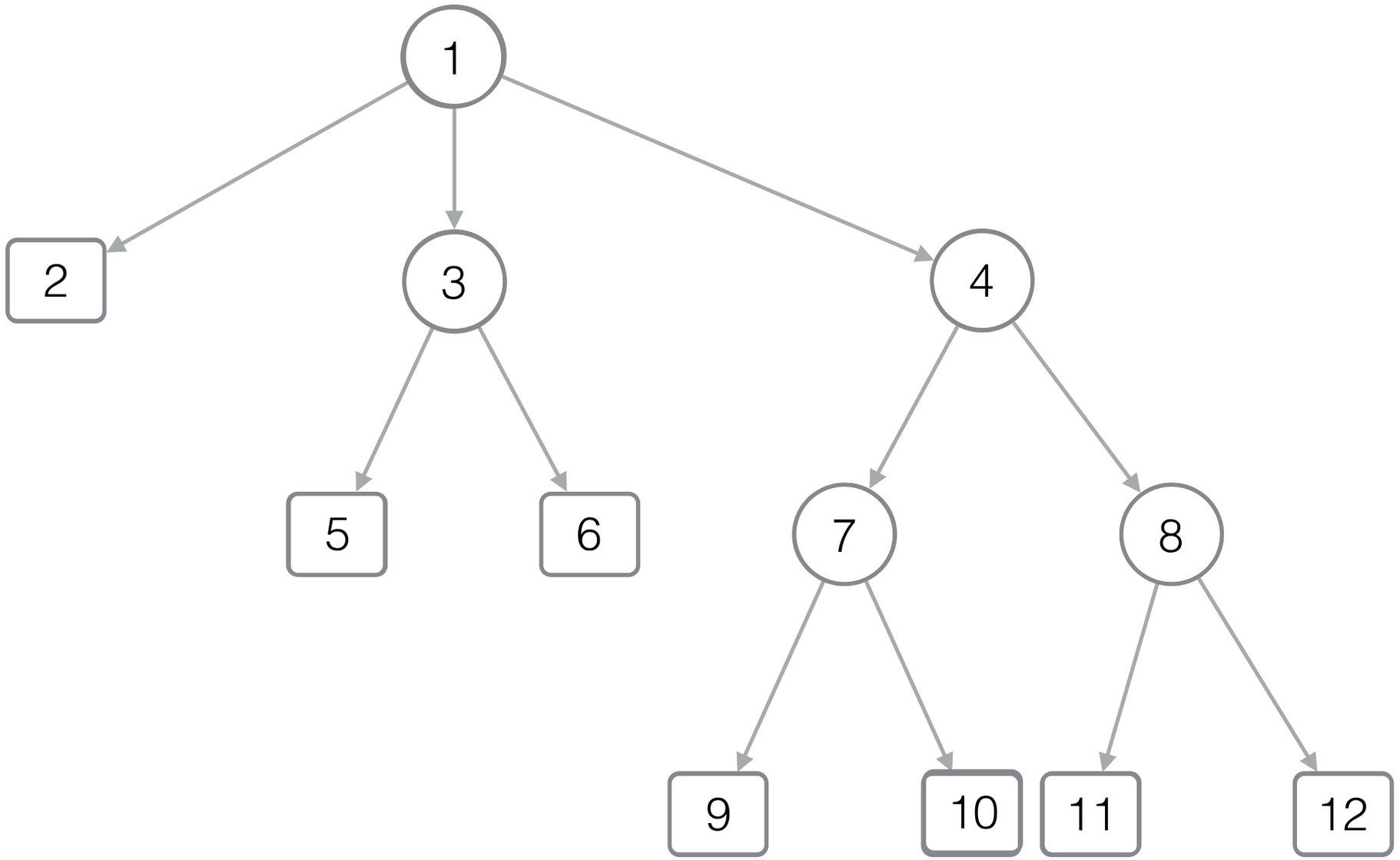


B 1 0 1 1 1 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0

1

Succinct representation of trees

[LOUDS - Level-order unary degree sequence]



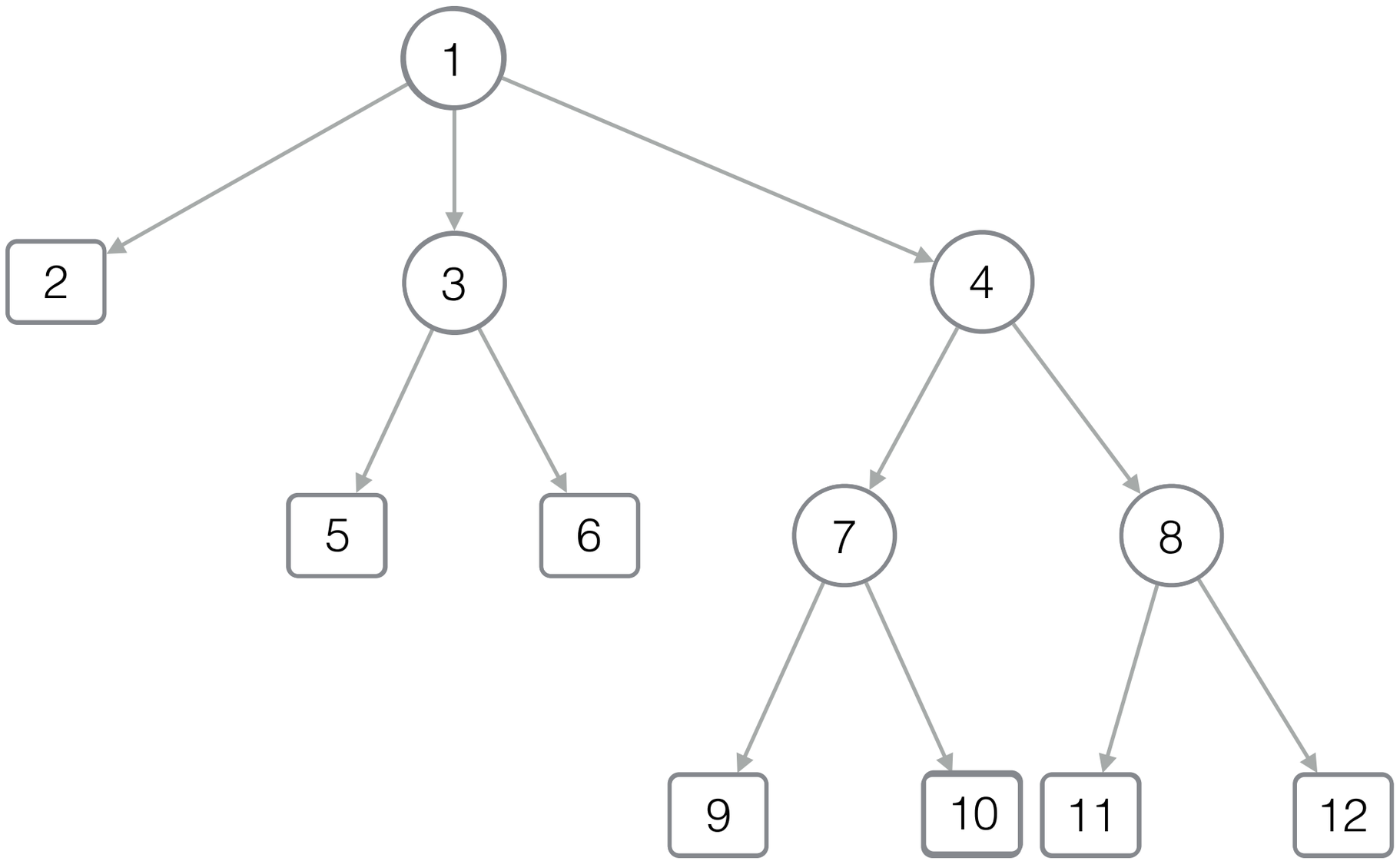
B 1 0 1 1 1 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0

1 2 3 4 5 6 7 8 9 10 11 12

Succinct representation of trees

[LOUDS - Level-order unary degree sequence]

$pos(x) = Select_1(x)$



pos(5)

B 1 0 1 1 1 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0

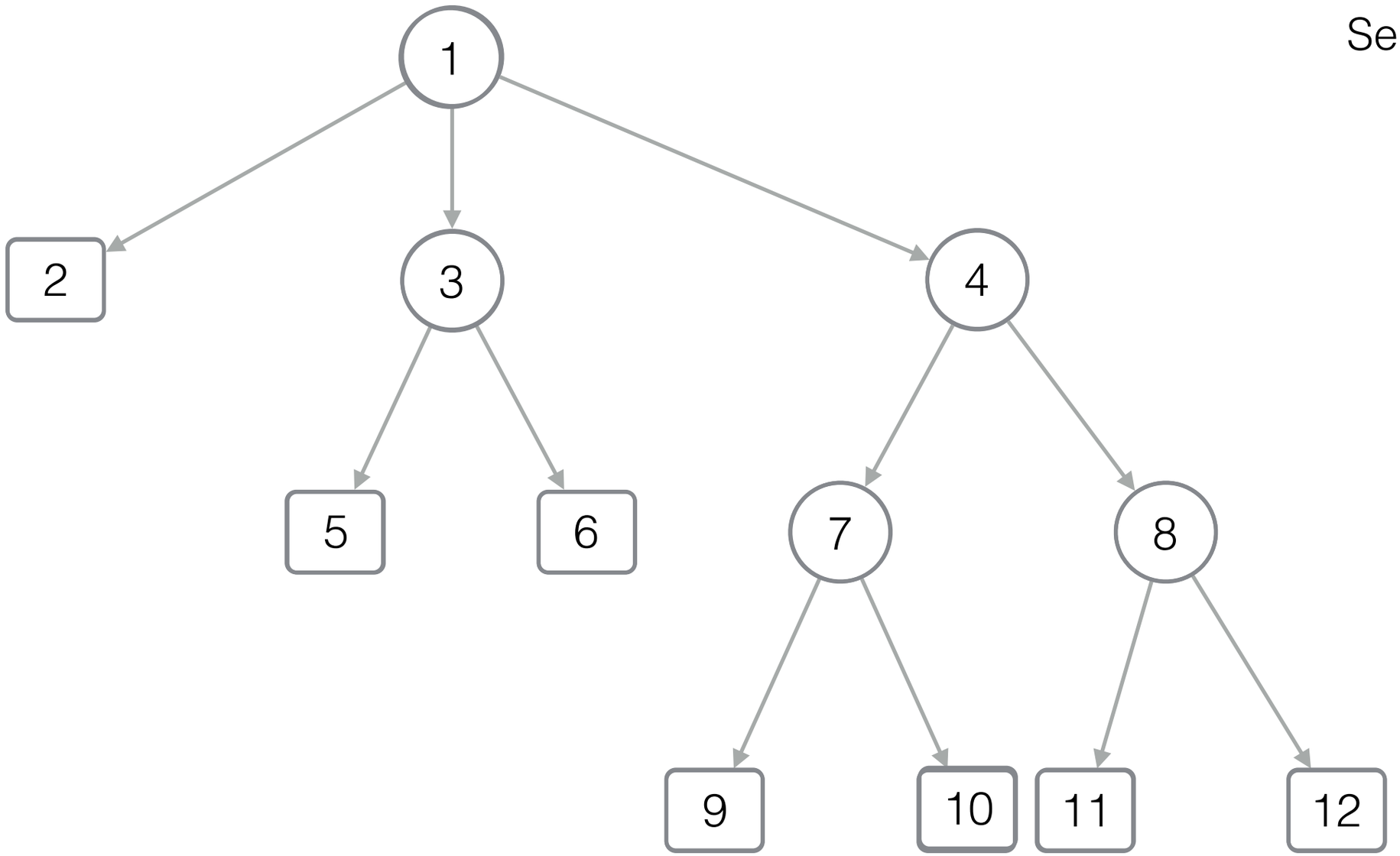
1 2 3 4 5 6 7 8 9 10 11 12

Succinct representation of trees

[LOUDS - Level-order unary degree sequence]

$pos(x) = Select_1(x)$

$degree(x) = Select_0(x+1) - (Select_0(x) + 1)$



B 1 0 1 1 1 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0

1 2 3 4 5 6 7 8 9 10 11 12

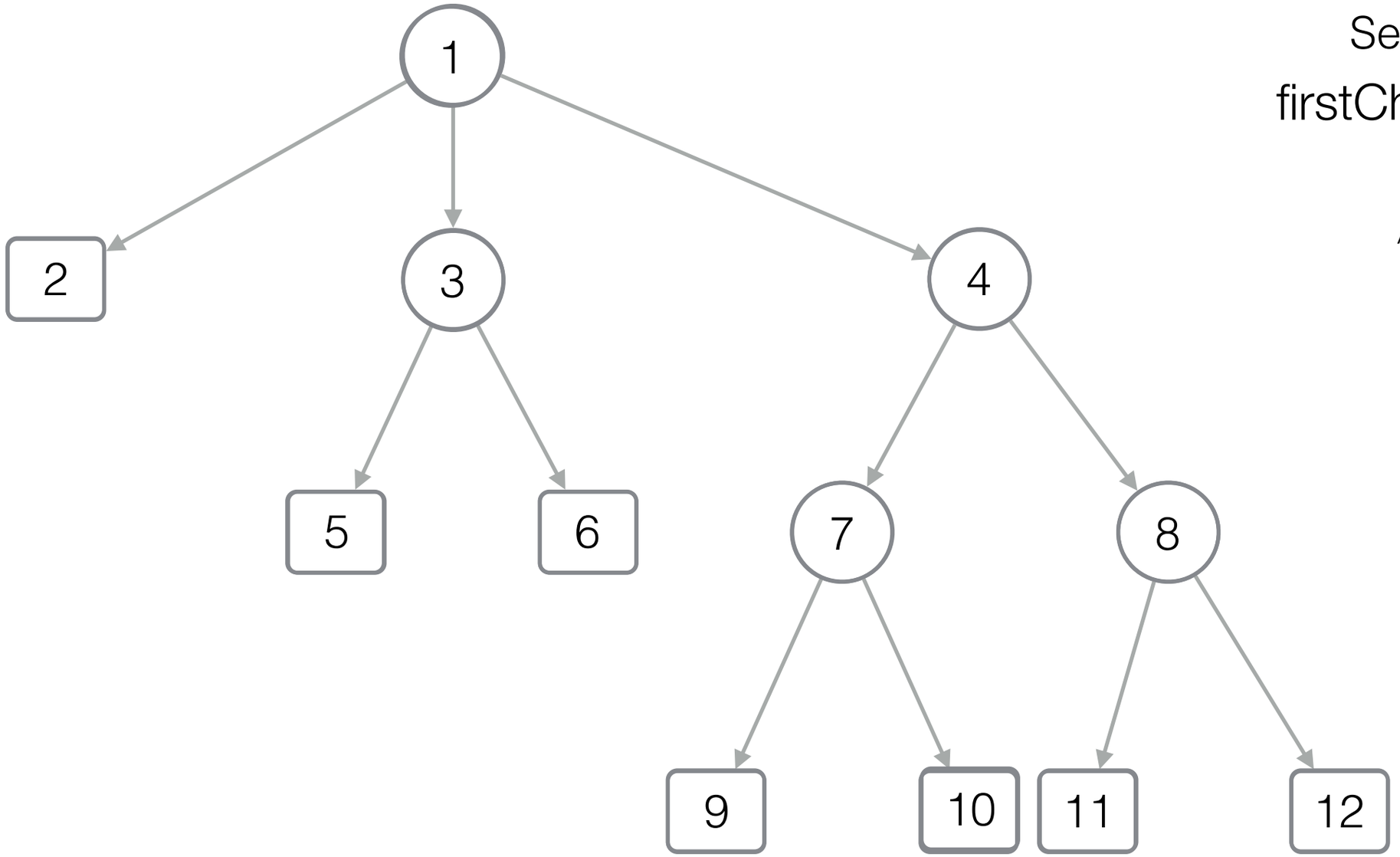
Succinct representation of trees

[LOUDS - Level-order unary degree sequence]

$pos(x) = Select_1(x)$

$degree(x) = Select_0(x+1) - (Select_0(x) + 1)$

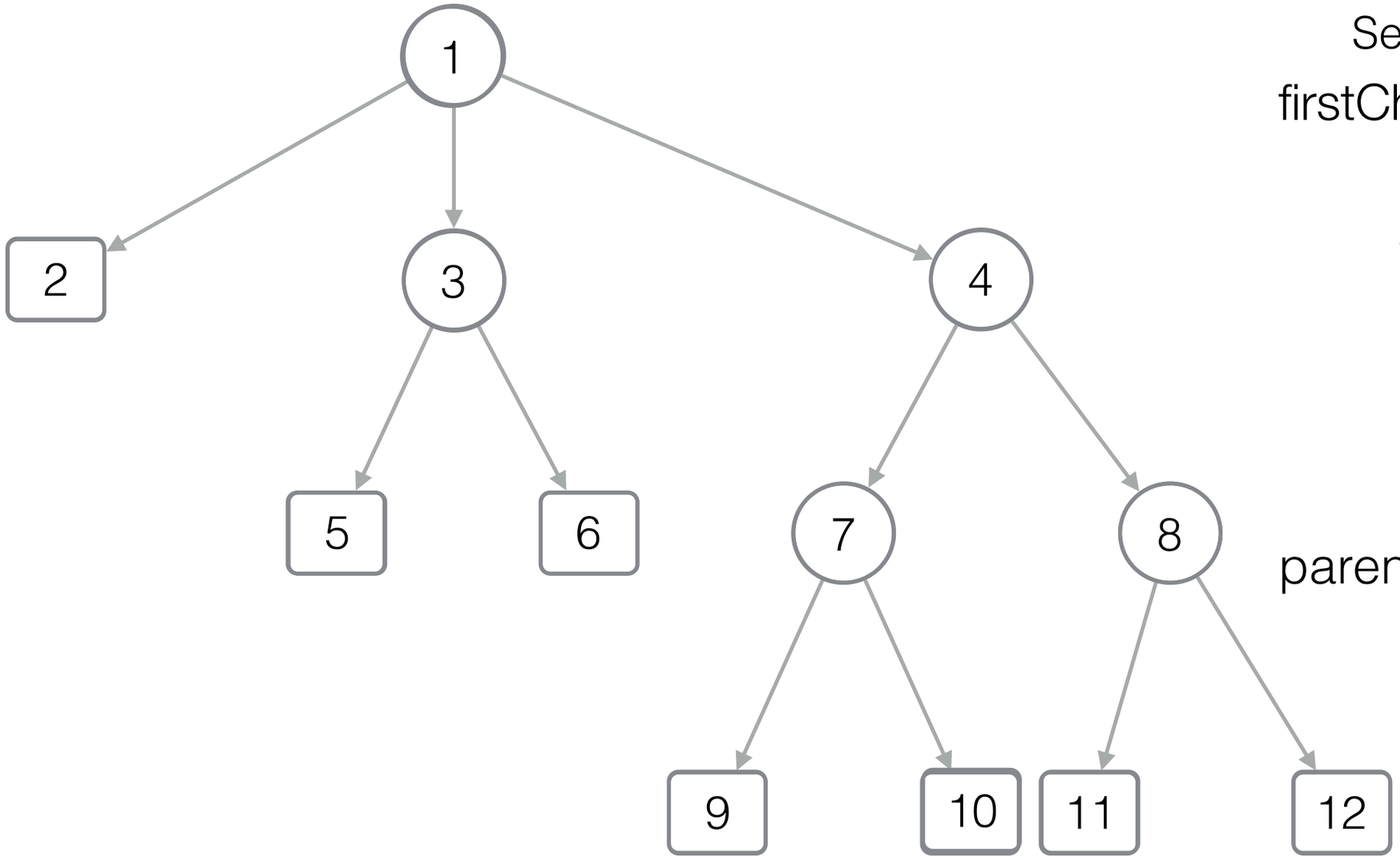
```
firstChild(x) =
    y = Select0(x)+1
    // start of x's children in B
    if B[y] == 0
        return -1    // is a leaf
    else
        return y-x // Rank1(y)
```



B 1 0 1 1 1 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0

1 2 3 4 5 6 7 8 9 10 11 12

Succinct representation of trees



[LOUDS - Level-order unary degree sequence]

pos(x) = $Select_1(x)$

degree(x) =

$Select_0(x+1) - (Select_0(x) + 1)$

firstChild(x) =

y = $Select_0(x)+1$

// start of x's children in B

if B[y] == 0

return -1 // is a leaf

else

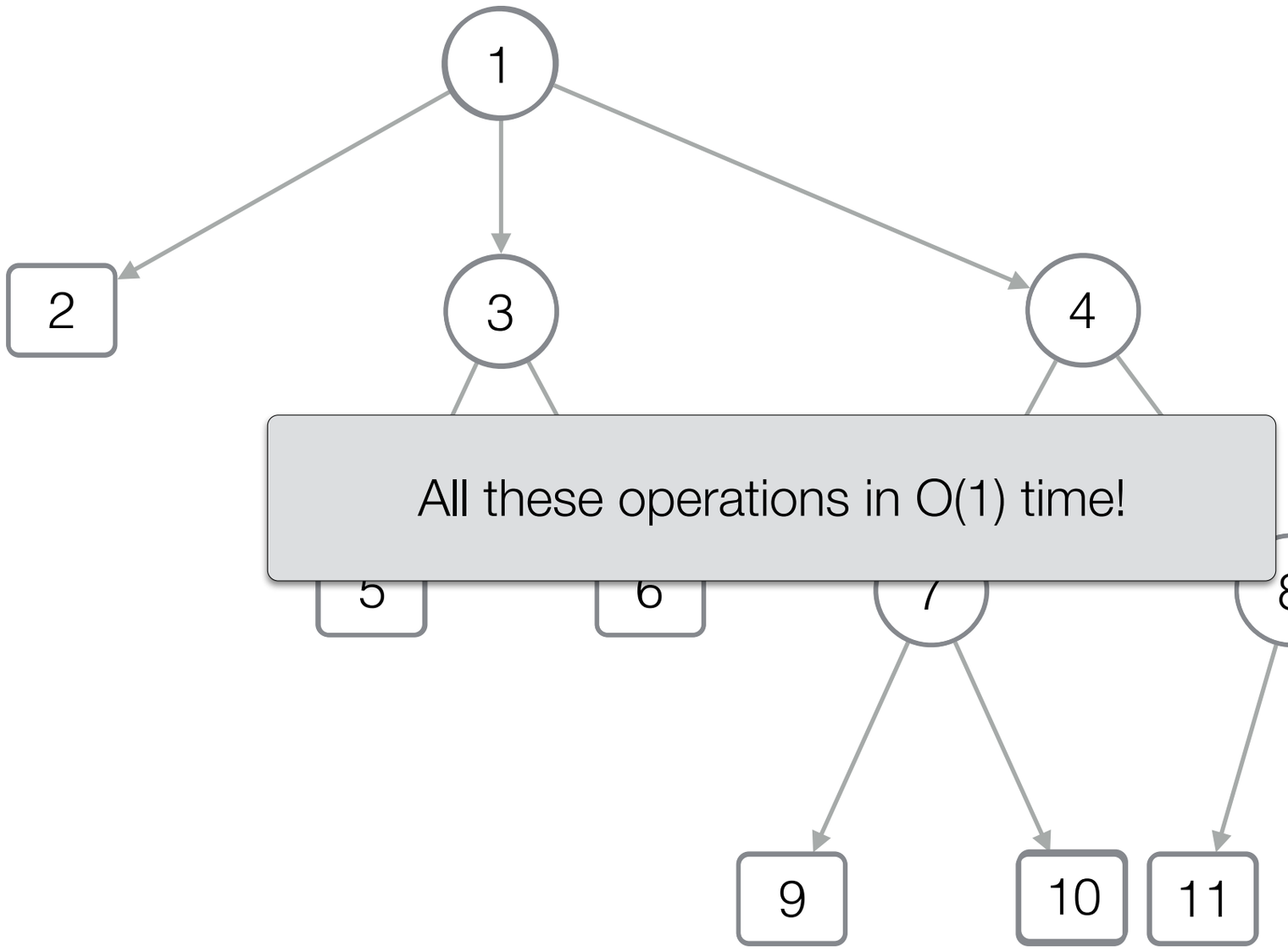
return y-x // $Rank_1(y)$

parent(x) = $Rank_0(pos(x))$

B 1 0 1 1 1 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 0

1 2 3 4 5 6 7 8 9 10 11 12

Succinct representation of trees



[LOUDS - Level-order unary degree sequence]

pos(x) = $Select_1(x)$

degree(x) =

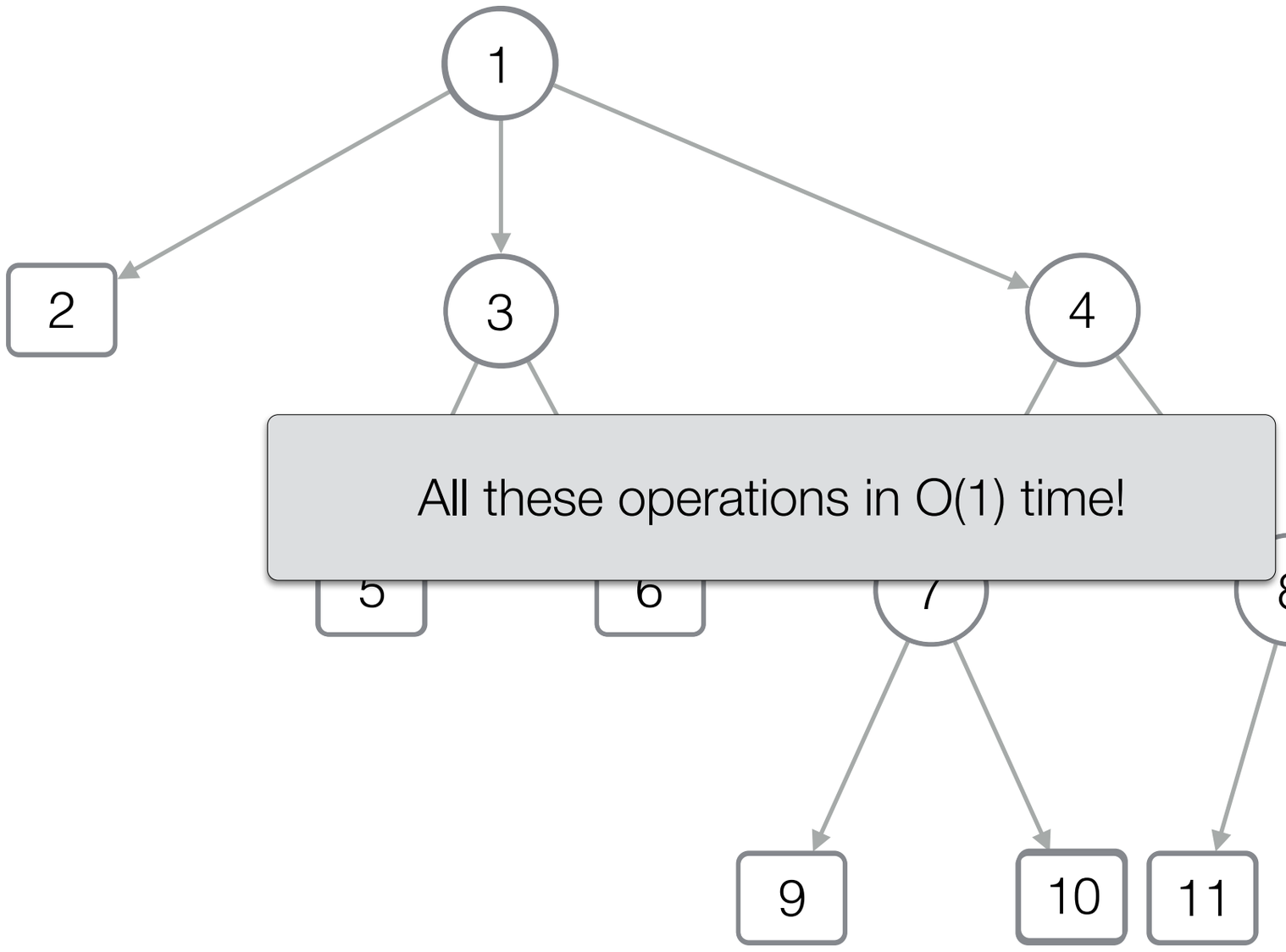
$Select_0(x+1) - (Select_0(x) + 1)$

firstChild(x) =

```
y = Select0(x)+1
// start of x's children in B
if B[y] == 0
    return -1    // is a leaf
else
    return y-x // Rank1(y)
```

parent(x) = $Rank_0(pos(x))$

Succinct representation of trees



[LOUDS - Level-order unary degree sequence]

$pos(x) = Select_1(x)$

$degree(x) = Select_0(x+1) - (Select_0(x) + 1)$

firstChild(x) =

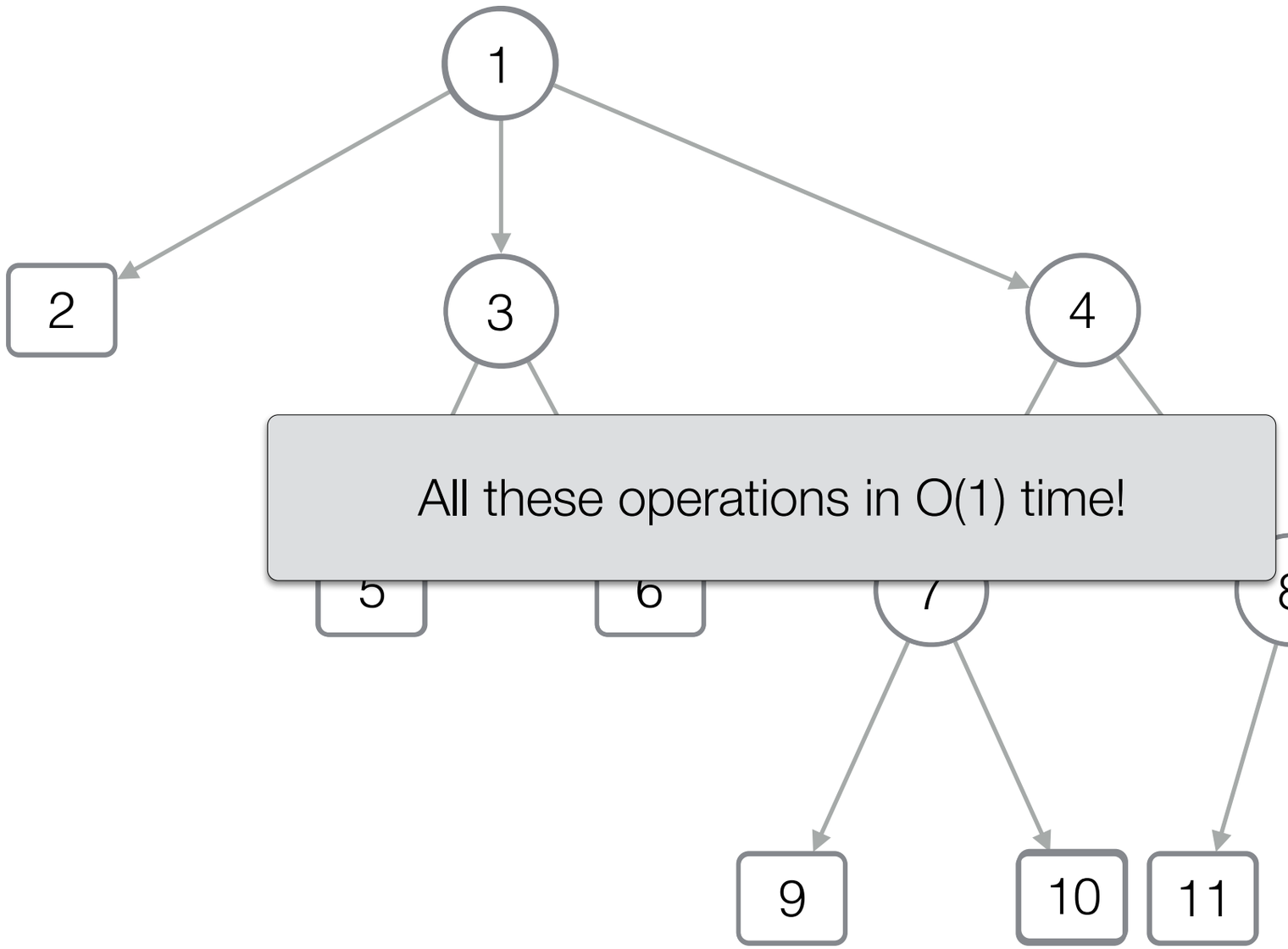
```
y = Select0(x)+1
// start of x's children in B
if B[y] == 0
    return -1    // is a leaf
else
    return y-x // Rank1(y)
```

$parent(x) = Rank_0(pos(x))$

subtreeSize(x) = ?

Not efficient!
Nodes of the subtree are spread in B

Succinct representation of trees



[LOUDS - Level-order unary degree sequence]

$pos(x) = Select_1(x)$

$degree(x) = Select_0(x+1) - (Select_0(x) + 1)$

firstChild(x) =

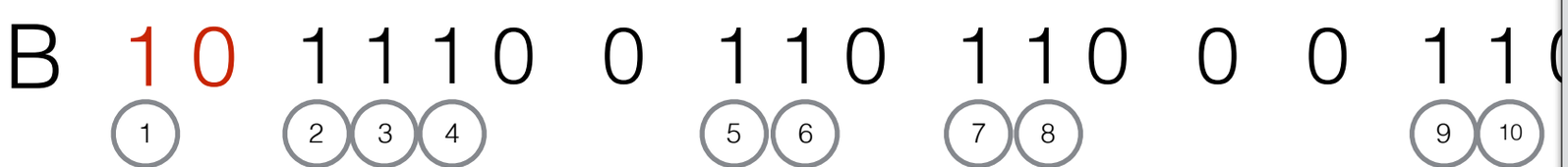
```
y = Select0(x)+1
// start of x's children in B
if B[y] == 0
    return -1    // is a leaf
else
    return y-x // Rank1(y)
```

$parent(x) = Rank_0(pos(x))$

subtreeSize(x) = ?

Not efficient!
Nodes of the subtree are spread in B

There exist more powerful representation



Semi-Indexing Semi-Structured Data in Tiny Space

Giuseppe Ottaviano
Dipartimento di Informatica
Università di Pisa
ottavian@di.unipi.it

Roberto Grossi
Dipartimento di Informatica
Università di Pisa
grossi@di.unipi.it

CIKM'11

ABSTRACT

Semi-structured textual formats are gaining increasing popularity for the storage of document collections and rich logs. Their flexibility comes at the cost of having to load and parse a document entirely even if just a small part of it needs to be accessed. For instance, in data analytics massive collections are usually scanned sequentially, selecting a small number of attributes from each document.

We propose a technique to attach to a raw, unparsed document (even in compressed form) a "semi-index": a succinct data structure that supports operations on the document

The field of applications of semi-structured data is rapidly increasing. These formats are making their way into the realm of storage of *massive* datasets. Their characteristics of being schema-free makes them a perfect fit for the mantra *"Log first, ask questions later"*, as the document schema is often evolving. Natural applications are crawler logs, query logs, user activity in social networks, to name a few.

In this domain JSON (*JavaScript Object Notation*, see [21]) in particular has been gaining momentum: the format is so simple and self-evident that its formal specification fits in a single page, and it is much less verbose than

Semi-Indexing Semi-Structured Data in Tiny Space

Giuseppe Ottaviano
Dipartimento di Informatica
Università di Pisa
ottavian@di.unipi.it

Roberto Grossi
Dipartimento di Informatica
Università di Pisa
grossi@di.unipi.it

Space-Efficient Data Structures for Top-k Completion

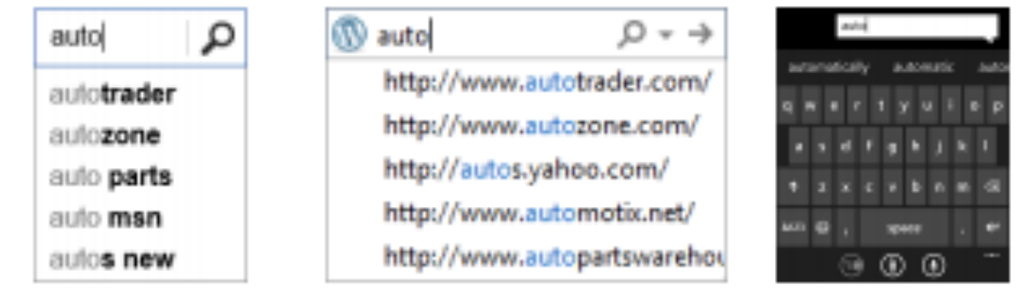
Bo-June (Paul) Hsu
Microsoft Research
One Microsoft Way, Redmond, WA, 98052 USA
paulhsu@microsoft.com

Giuseppe Ottaviano
Dipartimento di Informatica
Università di Pisa
ottavian@di.unipi.it

WWW'13

ABSTRACT

Virtually every modern search application, either desktop, web, or mobile, features some kind of query auto-completion. In its basic form, the problem consists in retrieving from a string set a small number of completions, i.e. strings beginning with a given prefix, that have the highest scores according to some static ranking. In this paper, we focus on the case where the string set is so large that compression is needed to fit the data structure in memory. This is a



(a) Search engine (b) Browser (c) Soft keyboard

Figure 1: Usage scenarios of top-k completion.

Semi-Indexing Semi-Structured Data in Tiny Space

Giuseppe Ottaviano
Dipartimento di Informatica
Università di Pisa
ottavian@di.unipi.it

Roberto Grossi
Dipartimento di Informatica
Università di Pisa
grossi@di.unipi.it

Space-Efficient Data Structures for Top-k Completion

Bo-June (Paul) Hsu
Microsoft Research
One Microsoft Way, Redmond, WA, 98052 USA
paulhsu@microsoft.com

Giuseppe Ottaviano
Dipartimento di Informatica
Università di Pisa
ottavian@di.unipi.it

ABS

Semi-s
ularity
Their
a docu
access
are us
of attr
We pr
ment (
data s

ABSTRA

Virtually
web, or mo
In its basi
a string se
ginning wi
according
on the cas
sion is nee

Faster and Smaller Inverted Indices with Treaps *

Roberto Konow
Dept. of Computer Science
Univ. of Chile, Chile
EIT, Univ. Diego Portales

Gonzalo Navarro
Dept. of Computer Science
Univ. of Chile, Chile

Charles L. A. Clarke
Alejandro López-Ortíz
School of Computer Science
Univ. of Waterloo, Canada

SIGIR'13

ABSTRACT

We introduce a new representation of the inverted index that performs faster ranked unions and intersections while using less space. Our index is based on the treap data structure, which allows us to intersect/merge the document identifiers while simultaneously thresholding by frequency, instead of the costlier two-step classical processing methods. To achieve compression we represent the treap topology using compact data structures. Further, the treap invariants

16]. In the first stage, a fast and simple filtration procedure extracts a subset of a few hundreds or thousands of candidates from the possibly billions of documents forming the collection. In the second stage, more complex learned ranking algorithms are applied to the reduced candidate set in order to obtain a handful of high-quality results. In this paper, we focus on improving the efficiency of the first stage, freeing more resources for the second stage and increasing the overall performance. In contexts where traditional rank-

Inverted indexes representation with Elias-Fano (aka Quasi-succinct indexes)

[Vigna WSDM ’13]

Inverted indexes representation with Elias-Fano (aka Quasi-succinct indexes)

[Vigna WSDM '13]

Venezia 1 3 7 8 9 10 13 16 17

IIR 1 7 8 9

....

Inverted indexes representation with Elias-Fano (aka Quasi-succinct indexes)

[Vigna WSDM '13]

Several (classic) encoding algorithms

Venezia	1	3	7	8	9	10	13	16	17
IIR	1	7	8	9					

....

Inverted indexes representation with Elias-Fano (aka Quasi-succinct indexes)

[Vigna WSDM '13]

Several (classic) encoding algorithms

Fast sequential access

Venezia	1	3	7	8	9	10	13	16	17
IIR	1	7	8	9					

....

Inverted indexes representation with Elias-Fano (aka Quasi-succinct indexes)

[Vigna WSDM '13]

Several (classic) encoding algorithms

Fast sequential access

Very slow random access and NextGEQ

Venezia	1	3	7	8	9	10	13	16	17
IIR	1	7	8	9					

....

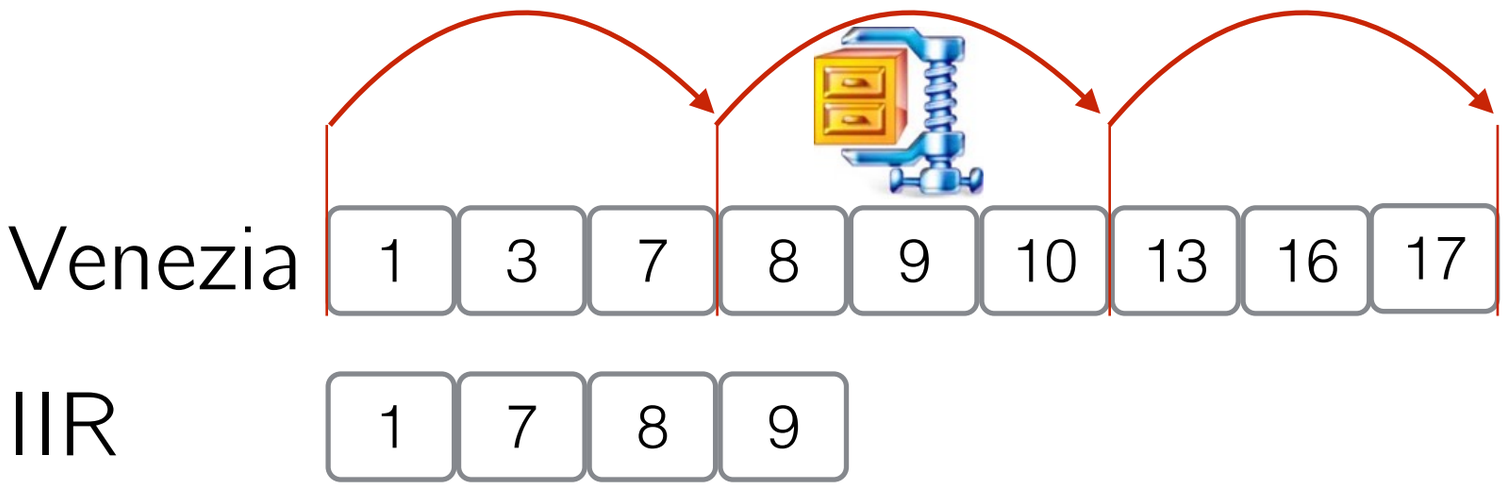
Inverted indexes representation with Elias-Fano (aka Quasi-succinct indexes)

[Vigna WSDM '13]

Several (classic) encoding algorithms

Fast sequential access

Very slow random access and NextGEQ



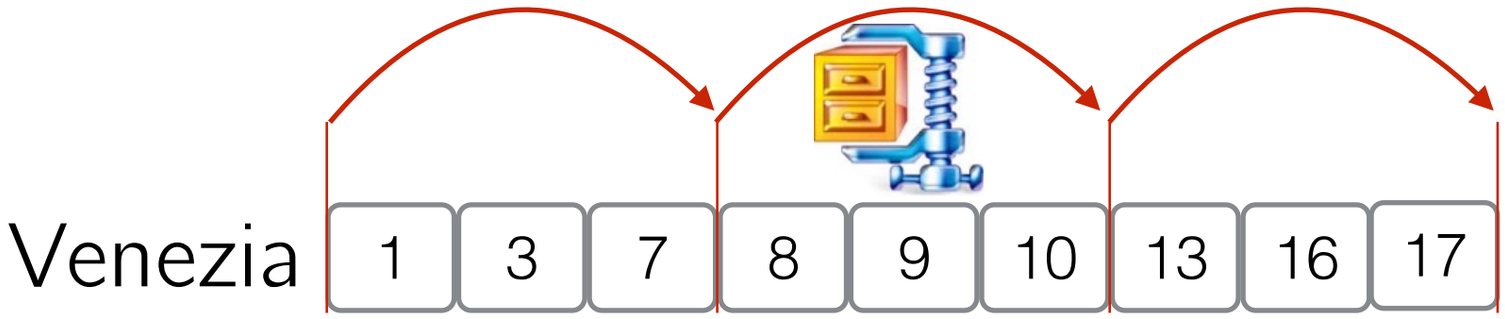
Inverted indexes representation with Elias-Fano (aka Quasi-succinct indexes)

[Vigna WSDM '13]

Several (classic) encoding algorithms

Fast sequential access

~~Very~~ slow random access and NextGEQ



Inverted indexes representation with Elias-Fano (aka Quasi-succinct indexes)

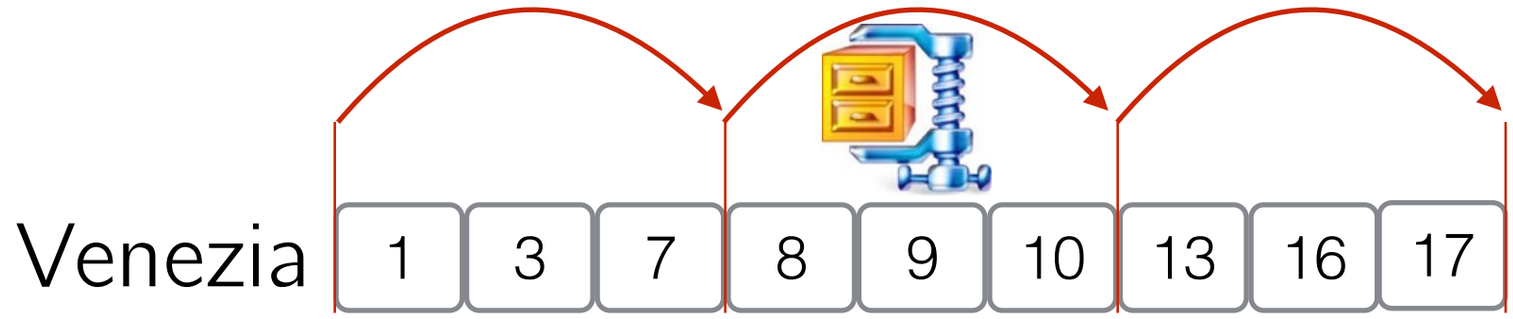
[Vigna WSDM '13]

Several (classic) encoding algorithms

Use Elias-Fano representation

Fast sequential access

~~Very~~ slow random access and NextGEQ



Inverted indexes representation with Elias-Fano (aka Quasi-succinct indexes)

[Vigna WSDM '13]

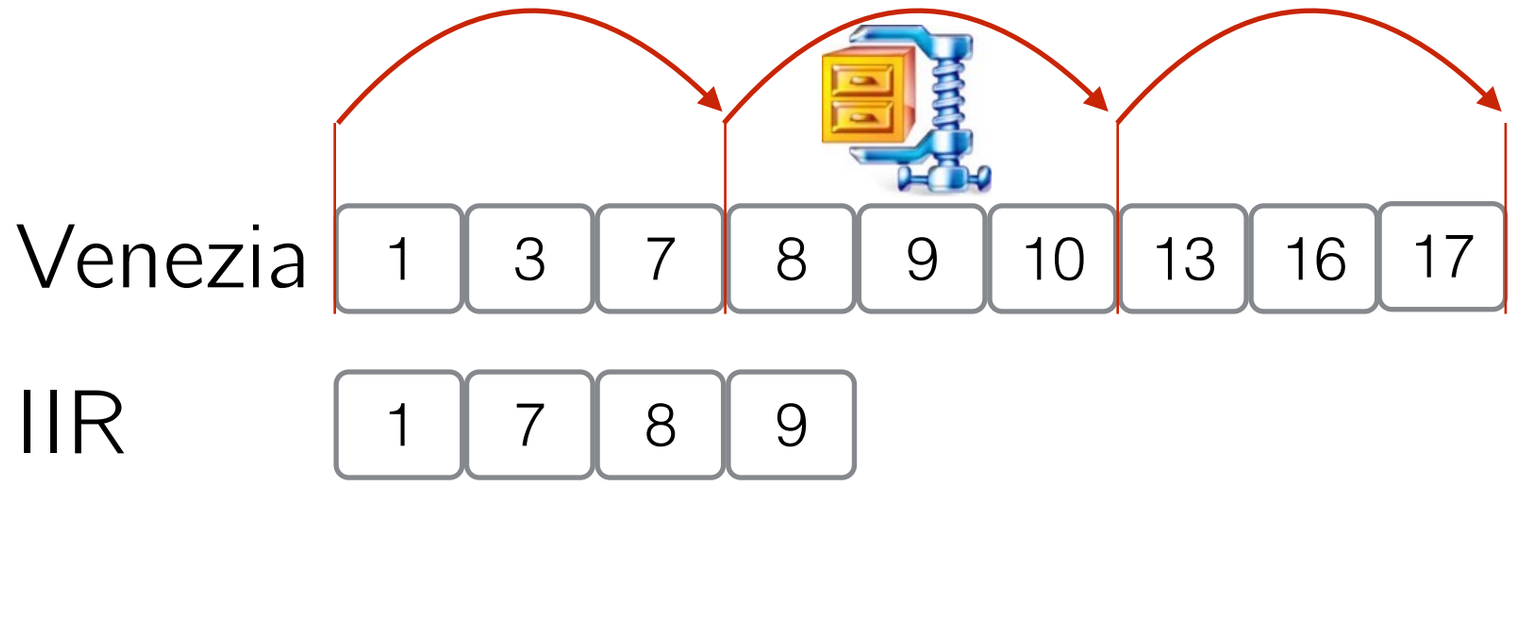
Several (classic) encoding algorithms

Use Elias-Fano representation

Fast sequential access

Slightly slower sequential access

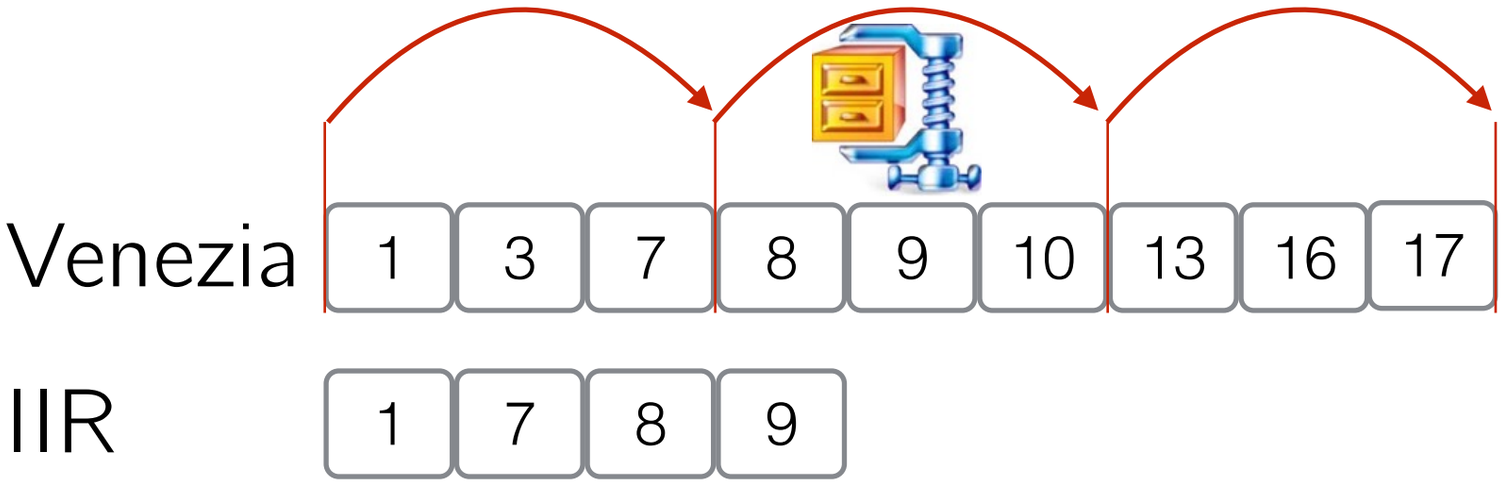
~~Very~~ slow random access and NextGEQ



Inverted indexes representation with Elias-Fano (aka Quasi-succinct indexes)

[Vigna WSDM '13]

Several (classic) encoding algorithms	Use Elias-Fano representation
Fast sequential access	Slightly slower sequential access
~~Very~~ slow random access and NextGEQ	Fast random access and NextGEQ



Results on Gov2 and ClueWeb09

[Experiments from Ottaviano-V. SIGIR '14]

Results on Gov2 and ClueWeb09

[Experiments from Ottaviano-V. SIGIR '14]

Space usage

	Gov2			ClueWeb09		
	space GB	doc bpi	freq bpi	space GB	doc bpi	freq bpi
EF single	7.66	7.53	3.14	19.63	7.46	2.44
Interpolative	4.57	4.03	2.33	14.62	5.33	2.04
OptPFD	5.22	4.72	2.55	17.80	6.42	2.56
Varint-G8IU	14.06	10.60	8.98	39.59	10.99	8.98

Results on Gov2 and ClueWeb09

[Experiments from Ottaviano-V. SIGIR '14]

Space usage

	Gov2			ClueWeb09		
	space GB	doc bpi	freq bpi	space GB	doc bpi	freq bpi
EF single	7.66	7.53	3.14	19.63	7.46	2.44
Interpolative	4.57	4.03	2.33	14.62	5.33	2.04
OptPFD	5.22	4.72	2.55	17.80	6.42	2.56
Varint-G8IU	14.06	10.60	8.98	39.59	10.99	8.98

AND queries in ms

	Gov2		ClueWeb09	
	TREC 05	TREC 06	TREC 05	TREC 06
EF single	2.1	4.7	13.6	15.8
Interpolative	7.5	20.4	55.7	76.5
OptPFD	2.2	5.7	16.6	21.9
Varint-G8IU	1.5	4.0	11.1	14.8

Results on Gov2 and ClueWeb09

[Experiments from Ottaviano-V. SIGIR '14]

Space usage

	Gov2			ClueWeb09		
	space GB	doc bpi	freq bpi	space GB	doc bpi	freq bpi
EF single	7.66	7.53	3.14	19.63	7.46	2.44
Interpolative	4.57	4.03	2.33	14.62	5.33	2.04
OptPFD	5.22	4.72	2.55	17.80	6.42	2.56
Varint-G8IU	14.06	10.60	8.98	39.59	10.99	8.98

AND queries in ms

	Gov2		ClueWeb09	
	TREC 05	TREC 06	TREC 05	TREC 06
EF single	2.1	4.7	13.6	15.8
Interpolative	7.5	20.4	55.7	76.5
OptPFD	2.2	5.7	16.6	21.9
Varint-G8IU	1.5	4.0	11.1	14.8

Results on Gov2 and ClueWeb09

[Experiments from Ottaviano-V. SIGIR '14]

Space usage

	Gov2			ClueWeb09		
	space GB	doc bpi	freq bpi	space GB	doc bpi	freq bpi
EF single	7.66	7.53	3.14	19.63	7.46	2.44
Interpolative	4.57	4.03	2.33	14.62	5.33	2.04
OptPFD	5.22	4.72	2.55	17.80	6.42	2.56
Varint-G8IU	14.06	10.60	8.98	39.59	10.99	8.98

AND queries in ms

	Gov2		ClueWeb09	
	TREC 05	TREC 06	TREC 05	TREC 06
EF single	2.1	4.7	13.6	15.8
Interpolative	7.5	20.4	55.7	76.5
OptPFD	2.2	5.7	16.6	21.9
Varint-G8IU	1.5	4.0	11.1	14.8

Results on Gov2 and ClueWeb09

[Experiments from Ottaviano-V. SIGIR '14]

Space usage

	Gov2			ClueWeb09		
	space GB	doc bpi	freq bpi	space GB	doc bpi	freq bpi
EF single	7.66	7.53	3.14	19.63	7.46	2.44
Interpolative	4.57	4.03	2.33	14.62	5.33	2.04
OptPFD	5.22	4.72	2.55	17.80	6.42	2.56
Varint-G8IU	14.06	10.60	8.98	39.59	10.99	8.98

AND queries in ms

	Gov2		ClueWeb09	
	TREC 05	TREC 06	TREC 05	TREC 06
EF single	2.1	4.7	13.6	15.8
Interpolative	7.5	20.4	55.7	76.5
OptPFD	2.2	5.7	16.6	21.9
Varint-G8IU	1.5	4.0	11.1	14.8

Results on Gov2 and ClueWeb09

[Experiments from Ottaviano-V. SIGIR '14]

Space usage

	Gov2			ClueWeb09		
	space GB	doc bpi	freq bpi	space GB	doc bpi	freq bpi
EF single	7.66	7.53	3.14	19.63	7.46	2.44
Interpolative	4.57	4.03	2.33	14.62	5.33	2.04
OptPFD	5.22	4.72	2.55	17.80	6.42	2.56
Varint-G8IU	14.06	10.60	8.98	39.59	10.99	8.98

AND queries in ms

	Gov2		ClueWeb09	
	TREC 05	TREC 06	TREC 05	TREC 06
EF single	2.1	4.7	13.6	15.8
Interpolative	7.5	20.4	55.7	76.5
OptPFD	2.2	5.7	16.6	21.9
Varint-G8IU	1.5	4.0	11.1	14.8

similar results for other query types

Partitioned Elias-Fano (PEF)

[Ottaviano-V. SIGIR '14]

Partitioned Elias-Fano (PEF)

[Ottaviano-V. SIGIR '14]

Property

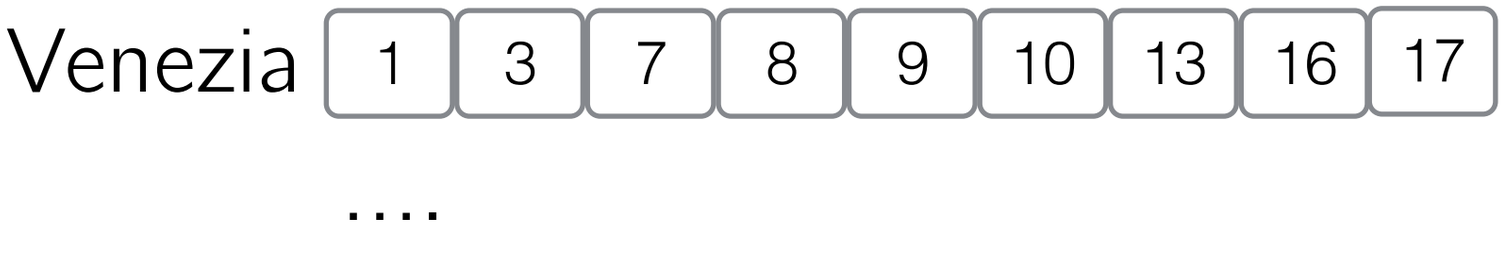
Posting lists have several clusters of almost consecutive values

Partitioned Elias-Fano (PEF)

[Ottaviano-V. SIGIR '14]

Property

Posting lists have several clusters of almost consecutive values

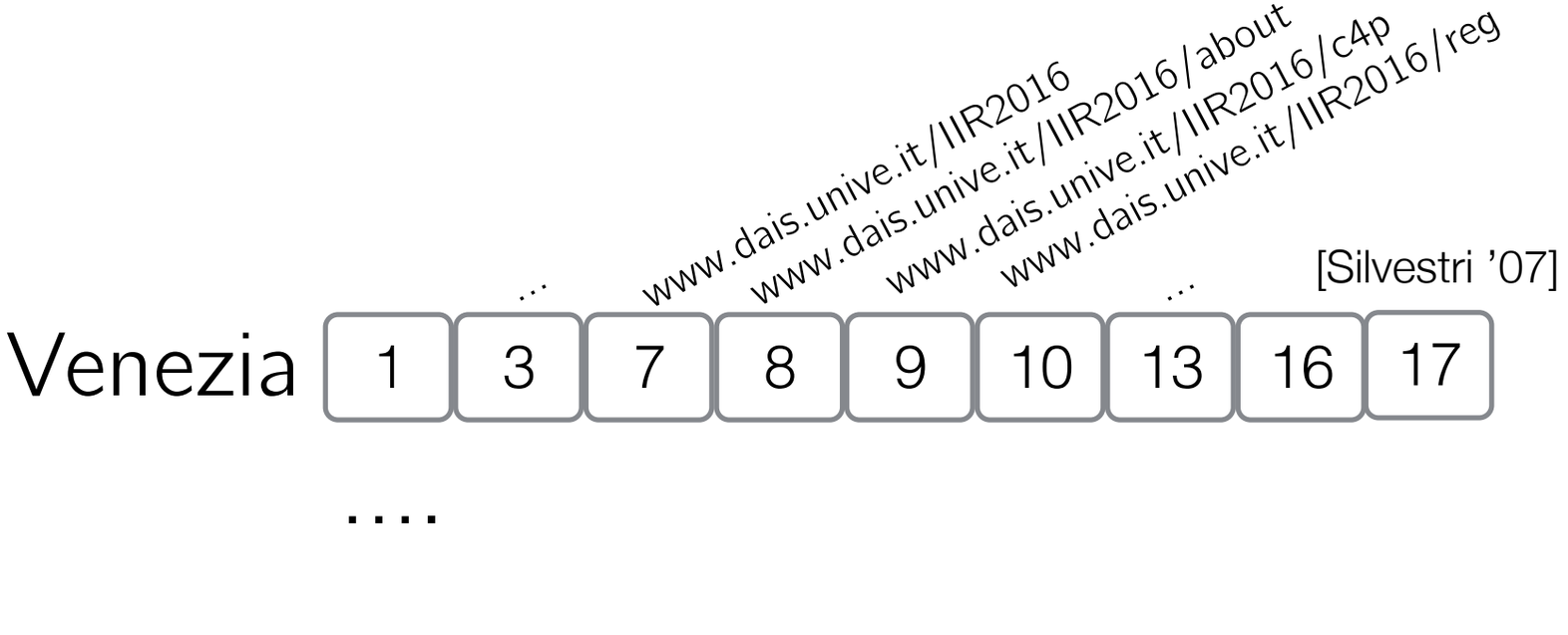


Partitioned Elias-Fano (PEF)

[Ottaviano-V. SIGIR '14]

Property

Posting lists have several clusters of almost consecutive values

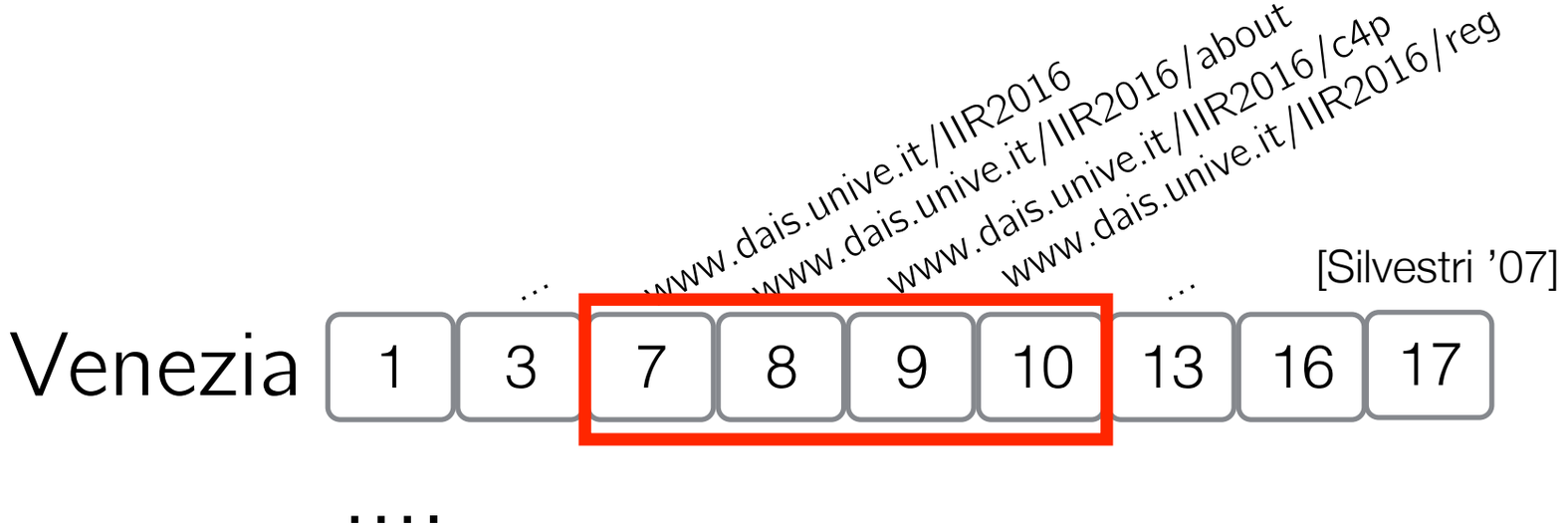


Partitioned Elias-Fano (PEF)

[Ottaviano-V. SIGIR '14]

Property

Posting lists have several clusters of almost consecutive values

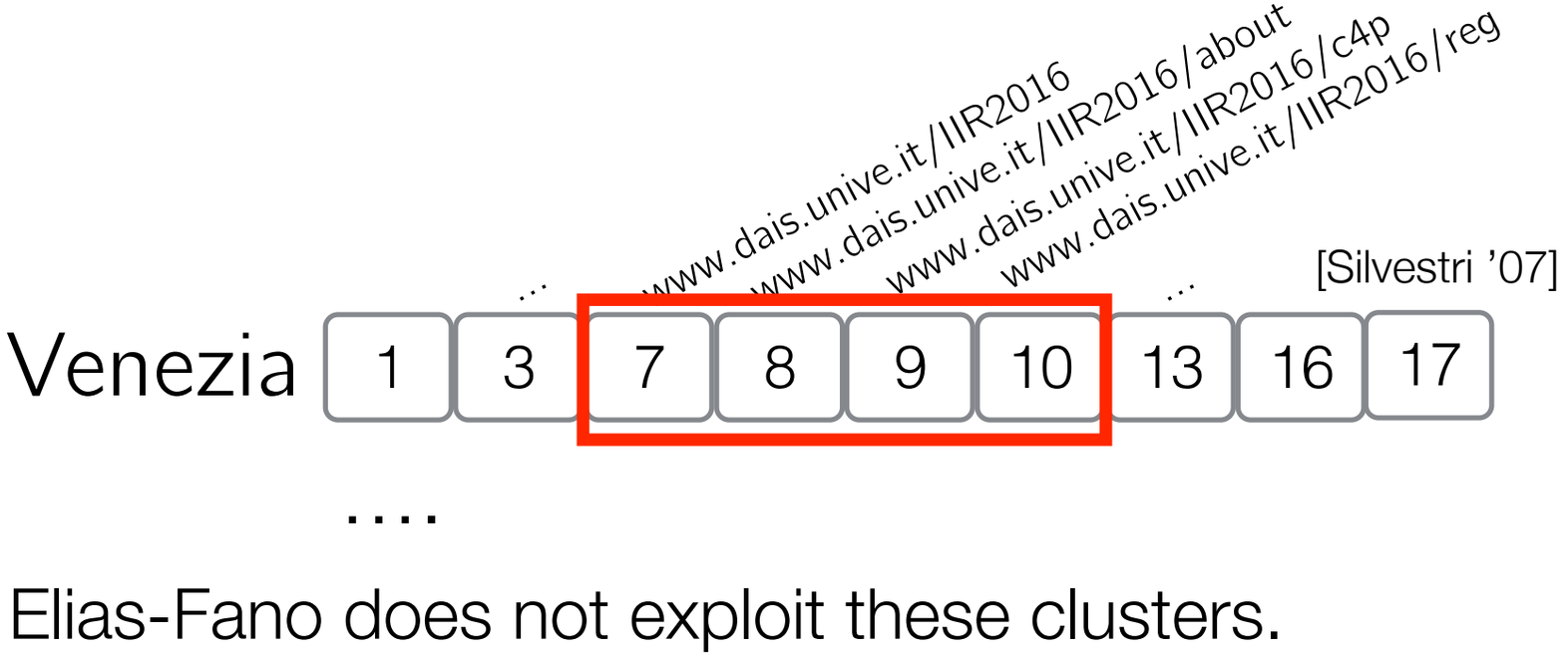


Partitioned Elias-Fano (PEF)

[Ottaviano-V. SIGIR '14]

Property

Posting lists have several clusters of almost consecutive values



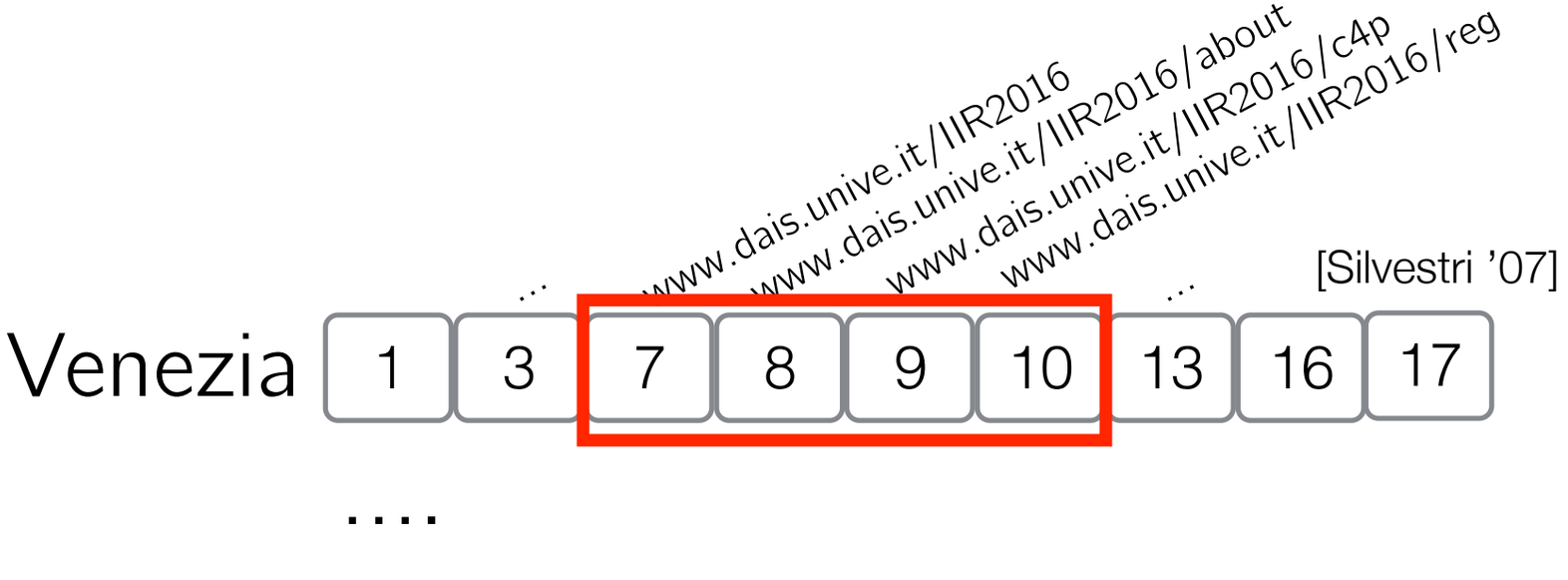
but Elias-Fano does not exploit these clusters.

Partitioned Elias-Fano (PEF)

[Ottaviano-V. SIGIR '14]

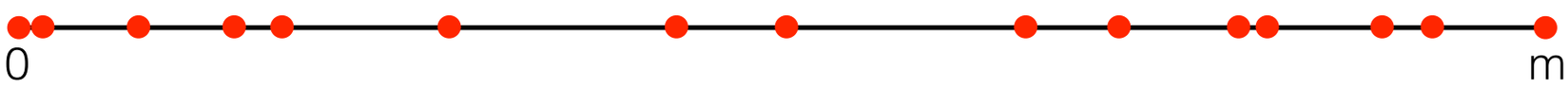
Property

Posting lists have several clusters of almost consecutive values

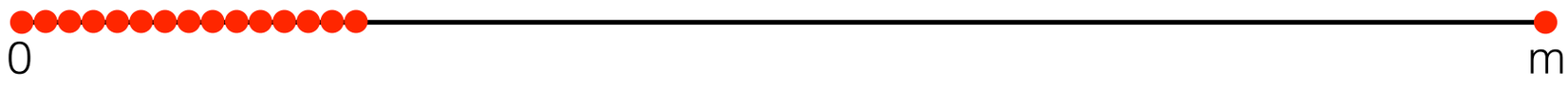


but Elias-Fano does not exploit these clusters.

1) n random numbers up to m



2) highly compressible sequence

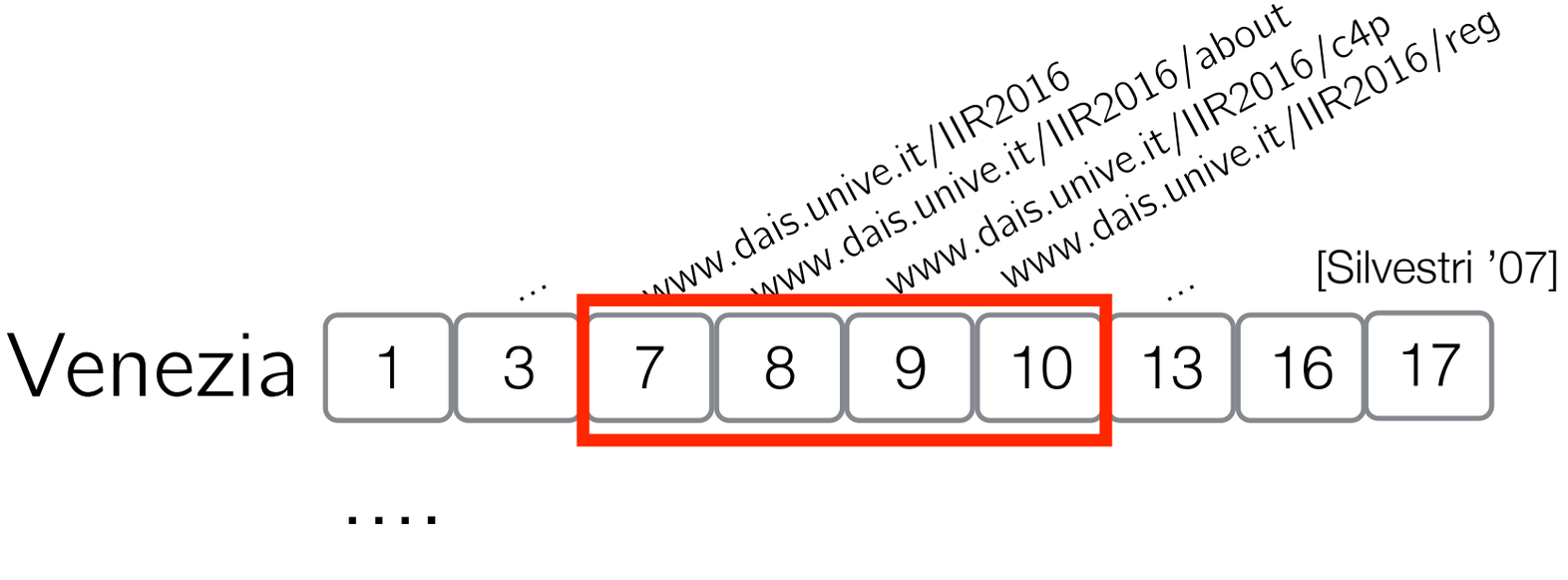


Partitioned Elias-Fano (PEF)

[Ottaviano-V. SIGIR '14]

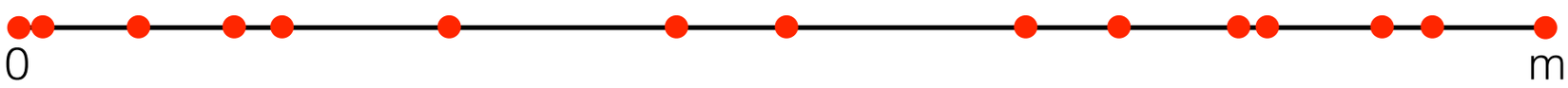
Property

Posting lists have several clusters of almost consecutive values

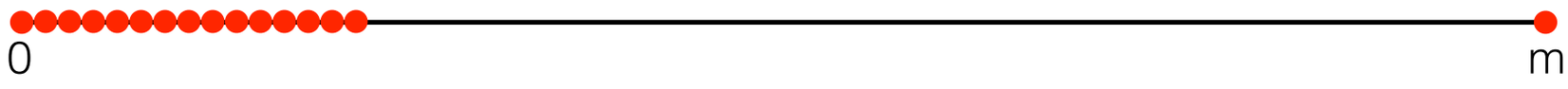


but Elias-Fano does not exploit these clusters.

1) n random numbers up to m



2) highly compressible sequence



Elias-Fano pays n log(m/n) + 2n bits to encode each.

Partitioned Elias-Fano (PEF)

[Ottaviano-V. SIGIR '14]

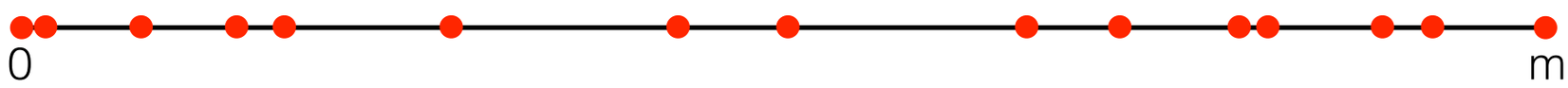
Property

Posting lists have several clusters of almost consecutive values

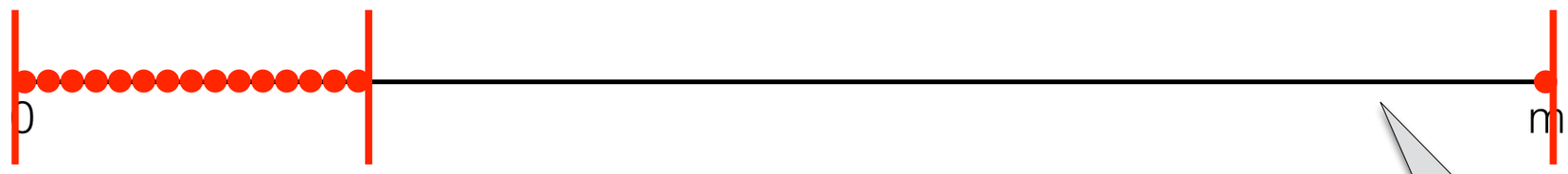
Partition each posting list into chunks and encode each chunk separately with Elias-Fano

but Elias-Fano does not exploit these clusters.

1) n random numbers up to m



2) highly compressible sequence



Elias-Fano pays n log(m/n) + 2n bits to encode each.

2log m bits

Partitioned Elias-Fano (PEF)

[Ottaviano-V. SIGIR '14]

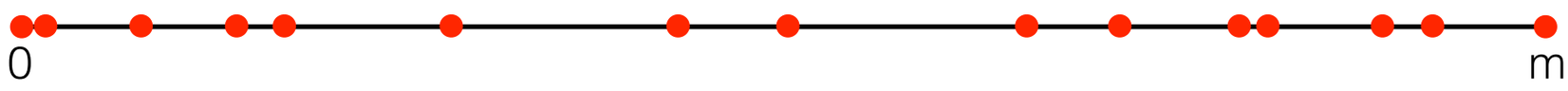
Property

Posting lists have several clusters of almost consecutive values

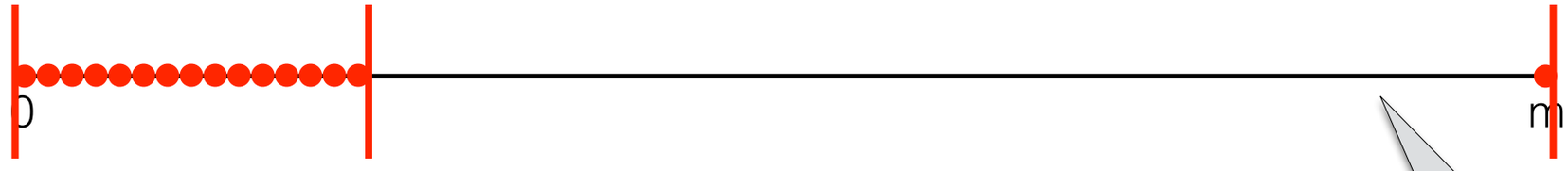
Partition each posting list into chunks and encode each chunk separately with Elias-Fano

Find an optimal partition (i.e., minimize space) in **linear time** with Dynamic Programming.

1) n random numbers up to m



2) highly compressible sequence



Elias-Fano pays n log(m/n) + 2n bits to encode each.

2log m bits

Results on Gov2 and ClueWeb09

Results on Gov2 and ClueWeb09

Space usage

	Gov2						ClueWeb09					
	space GB		doc bpi		freq bpi		space GB		doc bpi		freq bpi	
EF single	7.66	(+64.7%)	7.53	(+83.4%)	3.14	(+32.4%)	19.63	(+23.1%)	7.46	(+27.7%)	2.44	(+11.0%)
PEF	4.65		4.10		2.38		15.94		5.85		2.20	
Interpolative	4.57	(−1.8%)	4.03	(−1.8%)	2.33	(−1.8%)	14.62	(−8.3%)	5.33	(−8.8%)	2.04	(−7.1%)
OptPFD	5.22	(+12.3%)	4.72	(+15.1%)	2.55	(+7.4%)	17.80	(+11.6%)	6.42	(+9.8%)	2.56	(+16.4%)
Varint-G8IU	14.06	(+202.2%)	10.60	(+158.2%)	8.98	(+278.3%)	39.59	(+148.3%)	10.99	(+88.1%)	8.98	(+308.8%)

Results on Gov2 and ClueWeb09

Space usage

	Gov2						ClueWeb09					
	space GB		doc bpi		freq bpi		space GB		doc bpi		freq bpi	
EF single	7.66	(+64.7%)	7.53	(+83.4%)	3.14	(+32.4%)	19.63	(+23.1%)	7.46	(+27.7%)	2.44	(+11.0%)
PEF	4.65		4.10		2.38		15.94		5.85		2.20	
Interpolative	4.57	(−1.8%)	4.03	(−1.8%)	2.33	(−1.8%)	14.62	(−8.3%)	5.33	(−8.8%)	2.04	(−7.1%)
OptPFD	5.22	(+12.3%)	4.72	(+15.1%)	2.55	(+7.4%)	17.80	(+11.6%)	6.42	(+9.8%)	2.56	(+16.4%)
Varint-G8IU	14.06	(+202.2%)	10.60	(+158.2%)	8.98	(+278.3%)	39.59	(+148.3%)	10.99	(+88.1%)	8.98	(+308.8%)

Results on Gov2 and ClueWeb09

Space usage

	Gov2						ClueWeb09					
	space GB		doc bpi		freq bpi		space GB		doc bpi		freq bpi	
EF single	7.66	(+64.7%)	7.53	(+83.4%)	3.14	(+32.4%)	19.63	(+23.1%)	7.46	(+27.7%)	2.44	(+11.0%)
PEF	4.65		4.10		2.38		15.94		5.85		2.20	
Interpolative	4.57	(−1.8%)	4.03	(−1.8%)	2.33	(−1.8%)	14.62	(−8.3%)	5.33	(−8.8%)	2.04	(−7.1%)
OptPFD	5.22	(+12.3%)	4.72	(+15.1%)	2.55	(+7.4%)	17.80	(+11.6%)	6.42	(+9.8%)	2.56	(+16.4%)
Varint-G8IU	14.06	(+202.2%)	10.60	(+158.2%)	8.98	(+278.3%)	39.59	(+148.3%)	10.99	(+88.1%)	8.98	(+308.8%)

Results on Gov2 and ClueWeb09

Space usage

	Gov2						ClueWeb09					
	space GB		doc bpi		freq bpi		space GB		doc bpi		freq bpi	
EF single	7.66	(+64.7%)	7.53	(+83.4%)	3.14	(+32.4%)	19.63	(+23.1%)	7.46	(+27.7%)	2.44	(+11.0%)
PEF	4.65		4.10		2.38		15.94		5.85		2.20	
Interpolative	4.57	(−1.8%)	4.03	(−1.8%)	2.33	(−1.8%)	14.62	(−8.3%)	5.33	(−8.8%)	2.04	(−7.1%)
OptPFD	5.22	(+12.3%)	4.72	(+15.1%)	2.55	(+7.4%)	17.80	(+11.6%)	6.42	(+9.8%)	2.56	(+16.4%)
Varint-G8IU	14.06	(+202.2%)	10.60	(+158.2%)	8.98	(+278.3%)	39.59	(+148.3%)	10.99	(+88.1%)	8.98	(+308.8%)

AND queries in ms

	Gov2		ClueWeb09	
	TREC 05	TREC 06	TREC 05	TREC 06
EF single	2.1 (+10%)	4.7 (+1%)	13.6 (−5%)	15.8 (−9%)
PEF	1.9	4.6	14.3	17.4
Interpolative	7.5 (+291%)	20.4 (+343%)	55.7 (+289%)	76.5 (+341%)
OptPFD	2.2 (+14%)	5.7 (+24%)	16.6 (+16%)	21.9 (+26%)
Varint-G8IU	1.5 (−20%)	4.0 (−13%)	11.1 (−23%)	14.8 (−15%)

Results on Gov2 and ClueWeb09

Space usage

	Gov2						ClueWeb09					
	space GB		doc bpi		freq bpi		space GB		doc bpi		freq bpi	
EF single	7.66	(+64.7%)	7.53	(+83.4%)	3.14	(+32.4%)	19.63	(+23.1%)	7.46	(+27.7%)	2.44	(+11.0%)
PEF	4.65		4.10		2.38		15.94		5.85		2.20	
Interpolative	4.57	(−1.8%)	4.03	(−1.8%)	2.33	(−1.8%)	14.62	(−8.3%)	5.33	(−8.8%)	2.04	(−7.1%)
OptPFD	5.22	(+12.3%)	4.72	(+15.1%)	2.55	(+7.4%)	17.80	(+11.6%)	6.42	(+9.8%)	2.56	(+16.4%)
Varint-G8IU	14.06	(+202.2%)	10.60	(+158.2%)	8.98	(+278.3%)	39.59	(+148.3%)	10.99	(+88.1%)	8.98	(+308.8%)

AND queries in ms

	Gov2		ClueWeb09	
	TREC 05	TREC 06	TREC 05	TREC 06
EF single	2.1 (+10%)	4.7 (+1%)	13.6 (−5%)	15.8 (−9%
PEF	1.9	4.6	14.3	17.4
Interpolative	7.5 (+291%)	20.4 (+343%)	55.7 (+289%)	76.5 (+341%
OptPFD	2.2 (+14%)	5.7 (+24%)	16.6 (+16%)	21.9 (+26%
Varint-G8IU	1.5 (−20%)	4.0 (−13%)	11.1 (−23%)	14.8 (−15%

Results on Gov2 and ClueWeb09

Space usage

	Gov2						ClueWeb09					
	space GB		doc bpi		freq bpi		space GB		doc bpi		freq bpi	
EF single	7.66	(+64.7%)	7.53	(+83.4%)	3.14	(+32.4%)	19.63	(+23.1%)	7.46	(+27.7%)	2.44	(+11.0%)
PEF	4.65		4.10		2.38		15.94		5.85		2.20	
Interpolative	4.57	(−1.8%)	4.03	(−1.8%)	2.33	(−1.8%)	14.62	(−8.3%)	5.33	(−8.8%)	2.04	
OptPFD	5.22	(+12.3%)	4.72	(+15.1%)	2.55	(+7.4%)	17.80	(+11.6%)	6.42	(+9.8%)	2.56	
Varint-G8IU	14.06	(+202.2%)	10.60	(+158.2%)	8.98	(+278.3%)	39.59	(+148.3%)	10.99	(+88.1%)	8.98	(+308.8%)

AND queries in ms

	Gov2		ClueWeb09	
	TREC 05	TREC 06	TREC 05	TREC 06
EF single	2.1 (+10%)	4.7 (+1%)	13.6 (−5%)	15.8 (−9%
PEF	1.9	4.6	14.3	17.4
Interpolative	7.5 (+291%)	20.4 (+343%)	55.7 (+289%)	76.5 (+341%
OptPFD	2.2 (+14%)	5.7 (+24%)	16.6 (+16%)	21.9 (+26%
Varint-G8IU	1.5 (−20%)	4.0 (−13%)	11.1 (−23%)	14.8 (−15%

Inverted Indexes for Phrases and Strings·

Manish Patil
Department of CS
Louisiana State University
USA
mpatil@csc.lsu.edu

Sharma V. Thankachan
Department of CS
Louisiana State University
USA
thanks@csc.lsu.edu

Rahul Shah
Department of CS
Louisiana State University
USA
rahul@csc.lsu.edu

Wing-Kai Hon
Department of CS
National Tsing Hua University
Taiwan
wkhon@cs.nthu.edu.tw

Jeffrey Scott Vitter
Department of EECS
The University of Kansas
USA
jsv@ku.edu

Sabrina Chandrasekaran
Department of CS
Louisiana State University
USA
schand7@lsu.edu

SIGIR'11

ABSTRACT

Inverted indexes are the most fundamental and widely used data structures in information retrieval. For each unique word occurring in a document collection, the inverted index stores a list of the documents in which this word occurs. Compression techniques are often applied to further reduce the space requirement of these lists. However, the index has a shortcoming, in that only predefined pattern queries can be supported efficiently. In terms of string documents where word boundaries are undefined, if we have to index all

Categories and Subject Descriptors

H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing—*Indexing methods*; I.7.3 [**Document and Text Processing**]: Index Generation

General Terms

Algorithms, Experimentation

1 INTRODUCTION

Inverted Indexes for Phrases and Strings

Manish Patil
Department of CS
Louisiana State University
USA
mpatil@csc.lsu.edu

Sharma V. Thankachan
Department of CS
Louisiana State University
USA
thanks@csc.lsu.edu

Rahul Shah
Department of CS
Louisiana State University
USA
rahul@csc.lsu.edu

Wing-Kai Hon
Dep
National

Jeffrey Scott Vitter
Sabrina Chandrasekaran

wkhon(

ABSTRACT

Inverted indexes are
data structures in i
word occurring in a
stores a list of the
Compression techni
the space requirem
has a shortcoming,
can be supported ef
where word boundar

Compact, Efficient and Unlimited Capacity: Language Modeling with Compressed Suffix Trees

Ehsan Shareghi,[♭] Matthias Petri,[♮] Gholamreza Haffari[♭] and Trevor Cohn[♮]
[♭] Faculty of Information Technology, Monash University
[♮] Computing and Information Systems, The University of Melbourne
first.last@{monash.edu,unimelb.edu.au}

EMNLP'15

Abstract

Efficient methods for storing and querying language models are critical for scaling to large corpora and high Markov orders. In this paper we propose methods for modeling extremely large corpora without imposing a Markov condition. At its core, our approach uses a succinct index – a *compressed suffix tree* – which provides near optimal compression while support-

Heafield, 2011; Pauls and Klein, 2011; Sorensen and Allauzen, 2011; Watanabe et al., 2009). However, none of these approaches scale well to very high-order m or very large corpora, due to their high memory and time requirements. An important exception is Kennington et al. (2012), who also propose a language model based on a suffix tree which scales well with m but poorly with the corpus size (requiring memory of about 20× the training corpus).

Inverted Indexes for Phrases and Strings

Manish Patil
Department of CS
Louisiana State University
USA
mpatil@csc.lsu.edu

Sharma V. Thankachan
Department of CS
Louisiana State University
USA
thanks@csc.lsu.edu

Rahul Shah
Department of CS
Louisiana State University
USA
rahul@csc.lsu.edu

Wing-Kai Hon
Dep
National

Jeffrey Scott Vitter
Sabrina Chandrasekaran

wkhon(

ABSTRACT

Inverted indexes are
data structures in i
word occurring in a
stores a list of the
Compression techni
the space requirem
has a shortcoming,
can be supported ef
where word boundar

Efficient methoc
language model
large corpora an
this paper we p
eling extremely
posing a Marko
our approach u
compressed suff
near optimal co

Compact, Efficient and Unlimited Capacity: Language Modeling with Compressed Suffix Trees

Ehsan Shareghi,[♭] Matthias Petri,[♮] Gholamreza Haffari[♭] and Trevor Cohn[♮]
[♭] Faculty of Information Technology, Monash University
[♮] Computing and Information Systems, The University of Melbourne
first.last@{monash.edu,unimelb.edu.au}

Compressing Graphs and Indexes with Recursive Graph Bisection

Laxman Dhulipala[1], Igor Kabiljo[2], Brian Karrer[2], Giuseppe Ottaviano[2], Sergey Pupyrev[2], and Alon Shalita[2]

[1]Carnegie Mellon University
[2]Facebook

KDD'16

Abstract

Graph reordering is a powerful technique to increase the locality of the representations of graphs, which can be helpful in several applications. We study how the technique can be used to improve compression of graphs and inverted indexes.

We extend the recent theoretical model of Chierichetti et al. (KDD 2009) for graph compression, and show how it can be employed for compression-friendly reordering of social networks and web graphs and for assigning document identifiers in inverted indexes. We design and implement a novel theoretically sound reordering algorithm that is based on recursive graph bisection.

Take-home messages

Take-home messages

- Basic succinct data structures are quite easy to implement and very fast in practice

  - Advanced succinct data structures are obtained via a (often non trivial) combination of basic data structures (e.g., trees, graphs, text indexing, …)

Take-home messages

- Basic succinct data structures are quite easy to implement and very fast in practice
    - Advanced succinct data structures are obtained via a (often non trivial) combination of basic data structures (e.g., trees, graphs, text indexing, …)
- Mature field with solutions ready for technology transfer
    - Several succinct data structures implementations are ready-to-use in your projects
    - Give them a chance, if efficiency is your main concern

Take-home messages

- Basic succinct data structures are quite easy to implement and very fast in practice
  - Advanced succinct data structures are obtained via a (often non trivial) combination of basic data structures (e.g., trees, graphs, text indexing, …)
- Mature field with solutions ready for technology transfer
  - Several succinct data structures implementations are ready-to-use in your projects
    - Give them a chance, if efficiency is your main concern
- Several IR, data mining and bioinformatics papers are based on or inspired by succinct data structures
  - Yours may be the next!

Take-home messages

- Basic succinct data structures are quite easy to implement and very fast in practice
    - Advanced succinct data structures are obtained via a (often non trivial) combination of basic data structures (e.g., trees, graphs, text indexing, …)
- Mature field with solutions ready for technology transfer
    - Several succinct data structures implementations are ready-to-use in your projects
        - Give them a chance, if efficiency is your main concern
- Several IR, data mining and bioinformatics papers are based on or inspired by succinct data structures
    - Yours may be the next!
- Collaborations are very welcome!

Take-home messages

- Basic succinct data structures are quite easy to implement and very fast in practice
  - Advanced succinct data structures are obtained via a (often non trivial) combination of basic data structures (e.g., trees, graphs, text indexing, …)
- Mature field with solutions ready for technology transfer
  - Several succinct data structures implementations are ready-to-use in your projects
    - Give them a chance, if efficiency is your main concern
- Several IR, data mining and bioinformatics papers are based on or inspired by succinct data structures
  - Yours may be the next!
- Collaborations are very welcome!
- See you in Pisa!

Tutorial with Simon Gog
July 17, 2016

Thank you!