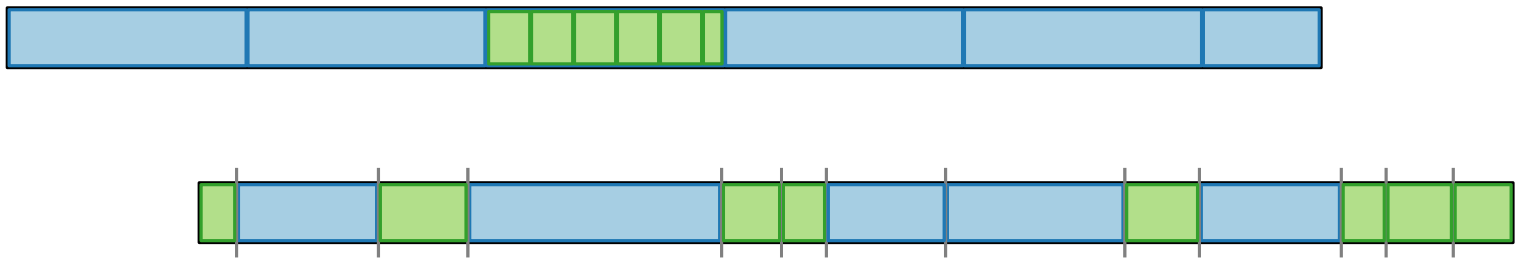
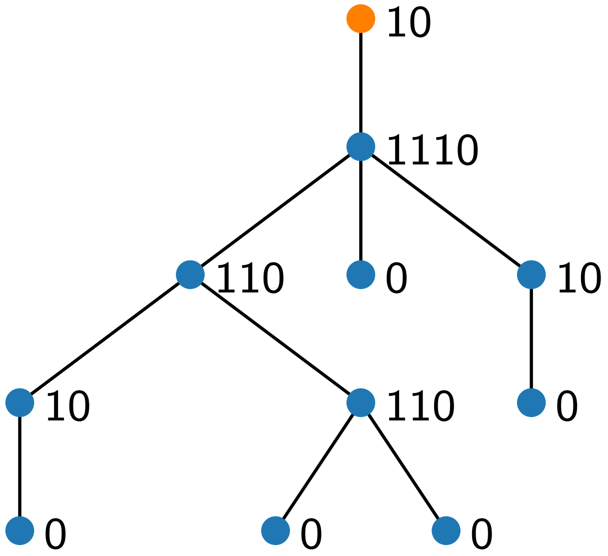
Advanced Algorithms

Succinct Data Structures

Indexable Dictionaries and Trees

Jonathan Klawitter · WS20





Data structures

A **data structure** is a concept to

- **store**,
- **organize**, and
- **manage** data.

As such, it is a collection of

- **data values**,
- their **relations**, and
- the **operations** that can applied to the data.

⇒

- What do we represent?
- How much space is required?
- Dynamic or static?
- Which operations are defined?
- How fast are they?

Remarks.

- We look at data structures as a designer/implementer (and not necessarily as a user).
- To define a data structure and to implement it are two different tasks.

Succinct data structures

Goal.

- Use space "close" to information-theoretical minimum,
- but still support time-efficient operations.

Let L be the information-theoretical lower bound
to represent a class of objects.
Then a data structure, which still supports
time-efficient operations, is called

- **implicit**, if it takes $L + O(1)$ bits of space;
- **succinct**, if it takes $L + o(L)$ bits of space;
- **compact**, if it takes $O(L)$ bits of space.

Succinct data structures

Goal.

- Use space "close" to information-theoretical minimum,
- but still support time-efficient operations.

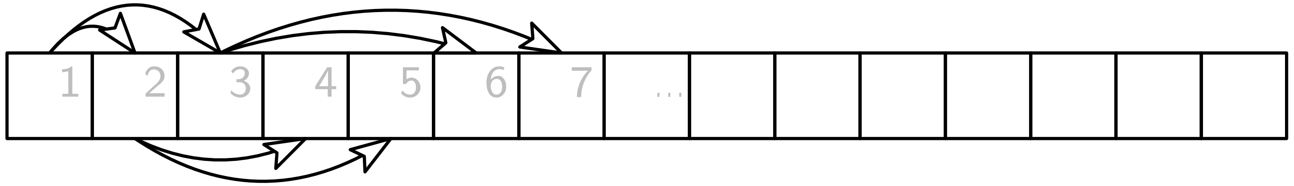
Let L be the information-theoretical lower bound
to represent a class of objects.
Then a data structure, which still supports
time-efficient operations, is called

- **implicit**, if it takes $L + O(1)$ bits of space;
- **succinct**, if it takes $L + o(L)$ bits of space;
- **compact**, if it takes $O(L)$ bits of space.

Examples?

Examples for implicit data structures

- **arrays** to represent lists
  - but why not linked lists?
- **1-dim arrays** to represent multi-dimensional arrays
- **sorted arrays** to represent sorted lists
  - but why not binary search trees?
- **arrays** to represent complete binary trees and heaps



$\texttt{leftChild}(i) = 2i$

$\texttt{rightChild}(i) = 2i + 1$

$\texttt{parent}(i) = \lfloor \frac{i}{2} \rfloor$

And unbalanced trees?

Succinct indexable dictionary

Represent a subset $S \subset [n]$ and support $O(1)$-time operations:

- $\texttt{member}(i)$ returns if $i \in S$
- $\texttt{rank}(i) = \#$ 1's at or before position i
- $\texttt{select}(j) =$ position of jth 1 bit
- `predecessor` and `successor` can be answered using `rank` and `select`

How many different subsets of $[n]$ are there? 2^n

How many bits of space do we need to distinguish them?

$$\log 2^n = n \textbf{ bits}$$

Succinct indexable dictionary

Represent S with a bit vector b of length n where

$$b[i] = \begin{cases} 1 & \text{if } i \in S \\ 0 & \text{otherwise} \end{cases}$$

plus $o(n)$-space structures to answer in $O(1)$ time

- $\texttt{rank}(i) = \#$ 1's at or before position i
- $\texttt{select}(j) =$ position of jth 1 bit

$S = \{3, 4, 6, 8, 9, 14\}$ where $n = 15$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
b	0	0	1	1	0	1	0	1	1	0	0	0	0	1	0

$\texttt{select}(5) = 9$

$\texttt{rank}(9) = 5$

Succinct indexable dictionary

Represent S with a bit vector b of length n where

$$b[i] = \begin{cases} 1 & \text{if } i \in S \\ 0 & \text{otherwise} \end{cases}$$

plus $o(n)$-space structures to answer in $O(1)$ time

- $\texttt{rank}(i) = \#$ 1's at or before position i
- $\texttt{select}(j) =$ position of jth 1 bit

$\Rightarrow$ Exercise: Use them to answer `predecessor` and `successor`.

$S = \{3, 4, 6, 8, 9, 14\}$ where $n = 15$

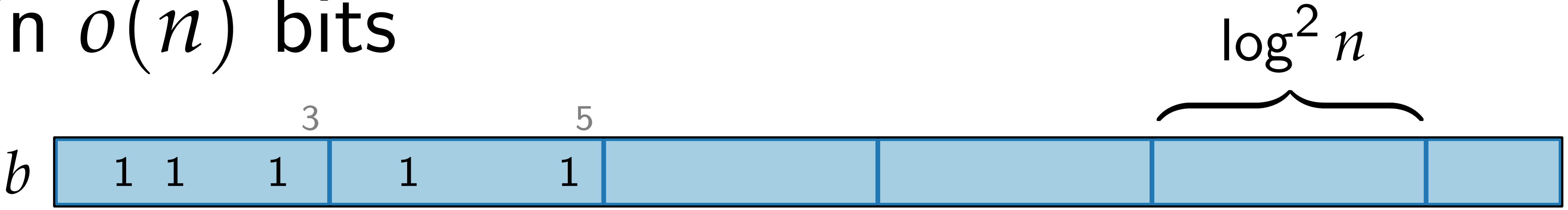
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
b	0	0	1	1	0	1	0	1	1	0	0	0	0	1	0

$\texttt{select}(5) = 9$

$\texttt{rank}(9) = 5 = \texttt{rank}(12)$

$\texttt{rank}(15) = 6$

Rank in $o(n)$ bits

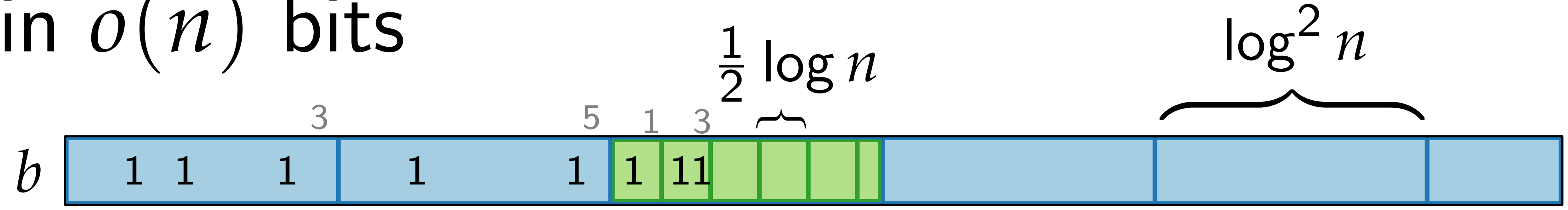


1. Split into $(\log^2 n)$-bit **chunks**

   and store cumulative rank: each $\log n$ bits

$$\Rightarrow O(\underbrace{\frac{n}{\log^2 n}}_{\#\text{ chunks}} \underbrace{\log n}_{\text{rank}}) = O(\frac{n}{\log n}) \subseteq o(n) \text{ bits}$$

Rank in $o(n)$ bits



1. Split into $(\log^2 n)$-bit **chunks**

   and store cumulative rank: each $\log n$ bits

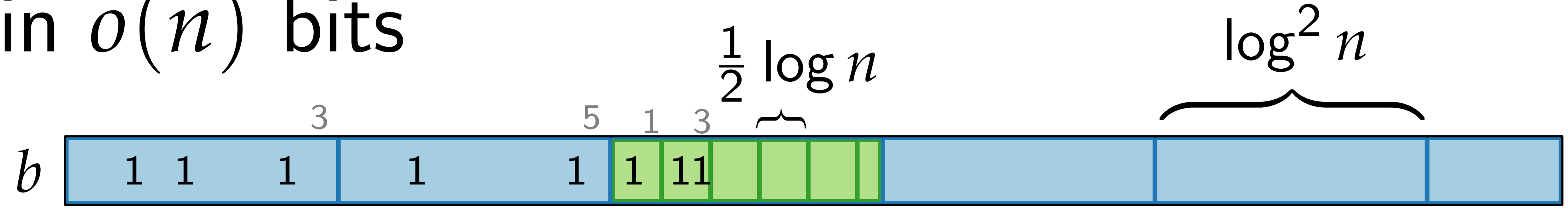
   $$\Rightarrow O\left(\frac{n}{\log^2 n}\log n\right) = O\left(\frac{n}{\log n}\right) \subseteq o(n) \text{ bits}$$

2. Split **chunks** into $(\frac{1}{2}\log n)$-bit **subchunks**

   and store cumulative rank within **chunk**: $2\log\log n$ bits

   $$\Rightarrow O\left(\underbrace{\frac{n}{\log n}}_{\#\text{ subch.}}\underbrace{\log\log n}_{\text{rel. rank}}\right) \subseteq o(n) \text{ bits}$$

Rank in $o(n)$ bits



1. Split into $(\log^2 n)$-bit **chunks**

   and store cumulative rank: each $\log n$ bits

   $$\Rightarrow O\left(\frac{n}{\log^2 n}\log n\right) = O\left(\frac{n}{\log n}\right) \subseteq o(n) \text{ bits}$$

2. Split **chunks** into $(\frac{1}{2}\log n)$-bit **subchunks**

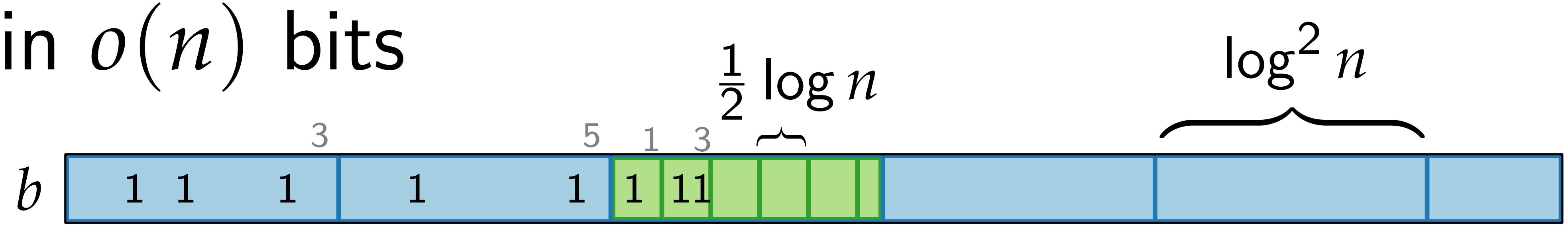
   and store cumulative rank within **chunk**: $2\log\log n$ bits

   $$\Rightarrow O\left(\frac{n}{\log n}\log\log n\right) \subseteq o(n) \text{ bits}$$

3. Use **lookup table** for bitstrings of length $(\frac{1}{2}\log n)$

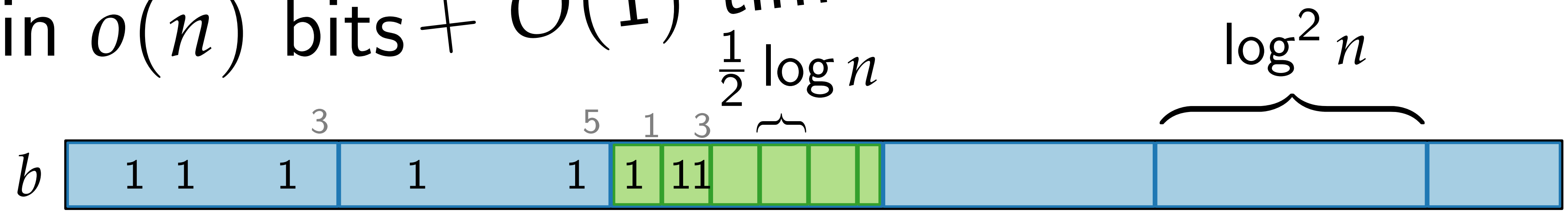
   $$\Rightarrow O(\underbrace{\sqrt{n}}_{\text{bitstring}}\underbrace{\log n}_{\text{query } i}\underbrace{\log\log n}_{\text{answer}}) \subseteq o(n) \text{ bits}$$

Rank in $o(n)$ bits



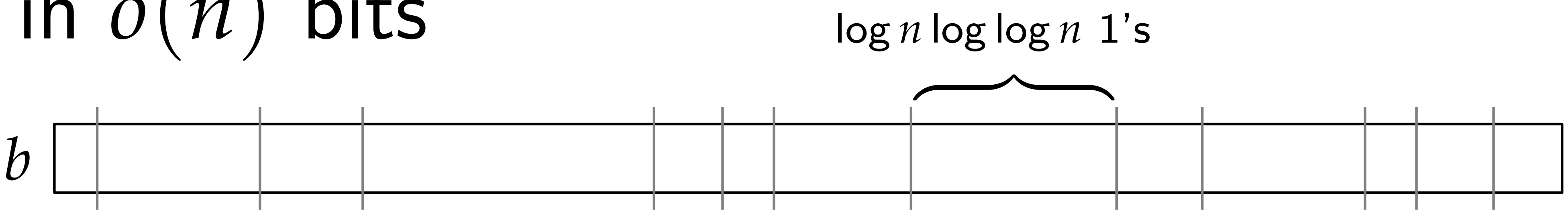
1. Split into $(\log^2 n)$-bit **chunks**
   and store cumulative rank: each $\log n$ bits
   $$\Rightarrow O(\frac{n}{\log^2 n}\log n) = O(\frac{n}{\log n}) \subseteq o(n) \text{ bits}$$
2. Split **chunks** into $(\frac{1}{2}\log n)$-bit **subchunks**
   and store cumulative rank within **chunk**: $2\log\log n$ bits
   $$\Rightarrow O(\frac{n}{\log n}\log\log n) \subseteq o(n) \text{ bits}$$
3. Use **lookup table** for bitstrings of length $(\frac{1}{2}\log n)$
   $$\Rightarrow O(\sqrt{n}\log n\log\log n) \subseteq o(n) \text{ bits}$$
4. $\texttt{rank} = \texttt{rank}$ of **chunk**
   $+$ relative $\texttt{rank}$ of **subchunk** within **chunk**
   $+$ relative $\texttt{rank}$ of element within **subchunk**

Rank in $o(n)$ bits + $O(1)$ time



1. Split into $(\log^2 n)$-bit **chunks**
   and store cumulative rank: each $\log n$ bits
   $$\Rightarrow O(\frac{n}{\log^2 n}\log n) = O(\frac{n}{\log n}) \subseteq o(n) \text{ bits}$$
2. Split **chunks** into $(\frac{1}{2}\log n)$-bit **subchunks**
   and store cumulative rank within **chunk**: $2\log\log n$ bits
   $$\Rightarrow O(\frac{n}{\log n}\log\log n) \subseteq o(n) \text{ bits}$$
3. Use **lookup table** for bitstrings of length $(\frac{1}{2}\log n)$
   $$\Rightarrow O(\sqrt{n}\log n\log\log n) \subseteq o(n) \text{ bits}$$
4. $\texttt{rank} = \texttt{rank}$ of **chunk**
   $+$ relative $\texttt{rank}$ of **subchunk** within **chunk**
   $+$ relative $\texttt{rank}$ of element within **subchunk**
   $$\Rightarrow O(1) \text{ time}$$

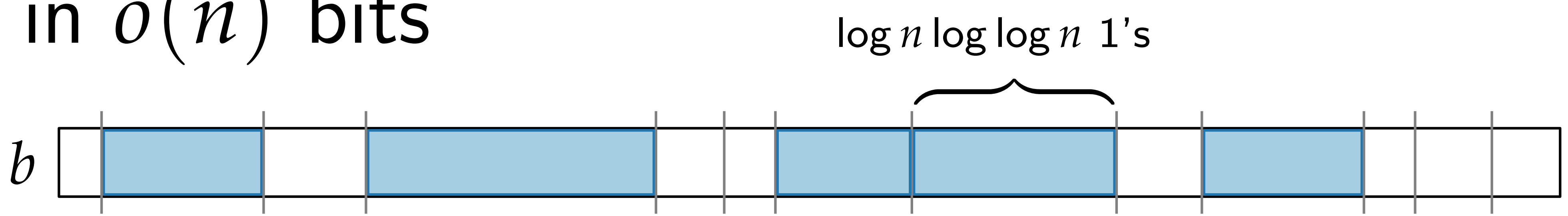
Select in $o(n)$ bits



1. Store indices of every $(\log n \log\log n)$th 1 bit in array

$$\Rightarrow O\Big(\underbrace{\frac{n}{\log n \log\log n}}_{\#\text{ groups}} \underbrace{\log n}_{\text{index}}\Big) = O\Big(\frac{n}{\log\log n}\Big) \subseteq o(n) \text{ bits}$$

Select in $o(n)$ bits



1. Store indices of every $(\log n \log\log n)$th 1 bit in array

$$\Rightarrow O(\frac{n}{\log n \log\log n} \log n) = O(\frac{n}{\log\log n}) \subseteq o(n) \text{ bits}$$

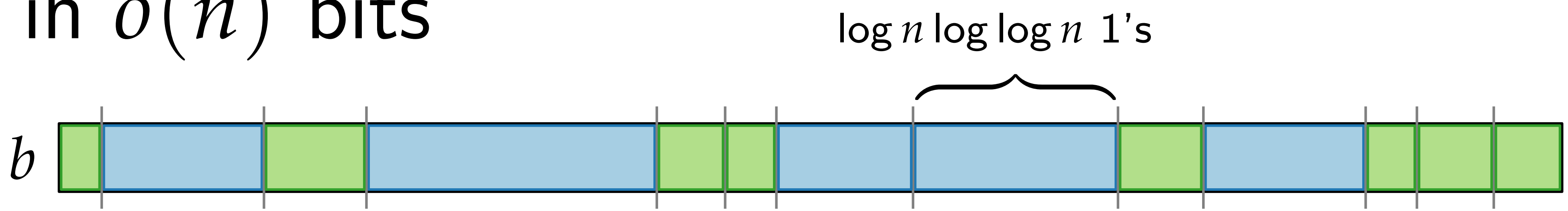
2. Within group of $(\log n \log\log n)$ 1 bits of length r bits:

   if $r \geq (\log n \log\log n)^2$

   then store indices of 1 bits in group in array

$$\Rightarrow O(\underbrace{\frac{n}{(\log n \log\log n)^2}}_{\#\text{ groups}} \underbrace{(\log n \log\log n)}_{\#\text{ 1 bits}} \underbrace{\log n}_{\text{index}}) \subseteq O(\frac{n}{\log\log n})$$

Select in $o(n)$ bits



1. Store indices of every $(\log n \log\log n)$th 1 bit in array

$$\Rightarrow O\left(\frac{n}{\log n \log\log n} \log n\right) = O\left(\frac{n}{\log\log n}\right) \subseteq o(n) \text{ bits}$$

2. Within group of $(\log n \log\log n)$ 1 bits of length r bits:

   if $r \geq (\log n \log\log n)^2$

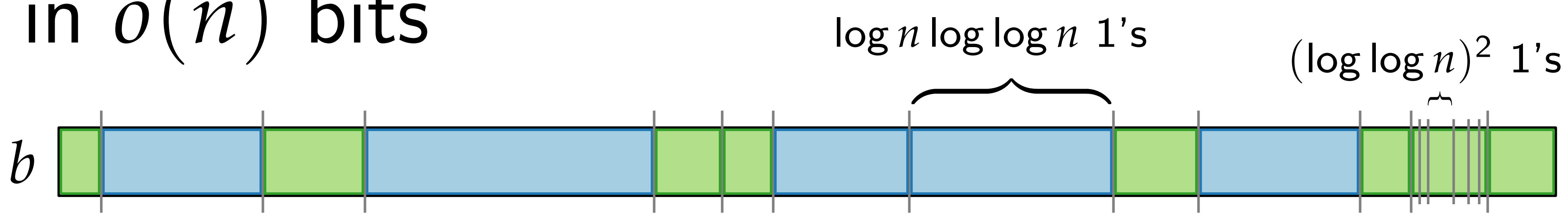
   then store indices of 1 bits in group in array

$$\Rightarrow O\left(\frac{n}{(\log n \log\log n)^2}(\log n \log\log n) \log n\right) \subseteq O\left(\frac{n}{\log\log n}\right)$$

   else problem is reduced to bitstrings of length $r < (\log n \log\log n)^2$

3. Repeat 1. and 2. on reduced bitstrings

Select in $o(n)$ bits



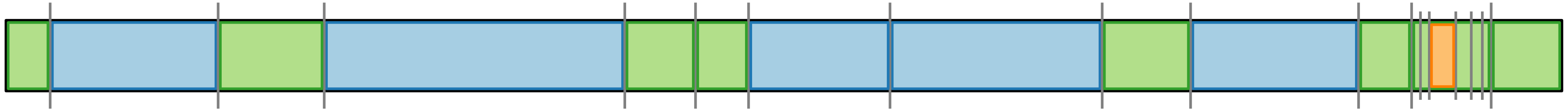
3. Repeat 1. and 2. on reduced bitstrings ($r < (\log n \log\log n)^2$):

   1' Store relative indices of every $(\log\log n)^2$th 1 bit in array

$$\Rightarrow O\left(\frac{n}{(\log\log n)^2} \log\log n\right) = O\left(\frac{n}{\log\log n}\right) \text{ bits}$$

\# subgroups   rel. index

Select in $o(n)$ bits



3. Repeat 1. and 2. on reduced bitstrings ($r < (\log n \log\log n)^2$):

1' Store relative indices of every $(\log\log n)^2$th 1 bit in array

$$\Rightarrow O\left(\frac{n}{(\log\log n)^2}\log\log n\right) = O\left(\frac{n}{\log\log n}\right) \text{ bits}$$

2' Within group of $(\log\log n)^2$th 1 bits of length r' bits:

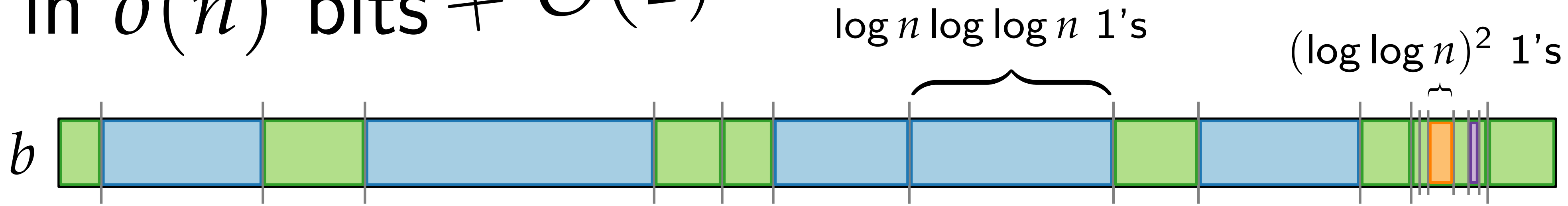
if $r' \geq (\log\log n)^4$

then store relative indices of 1 bits in subgroup in array

$$\Rightarrow O\left(\frac{n}{(\log\log n)^4}(\log\log n)^2\log\log n\right) = O\left(\frac{n}{\log\log n}\right) \text{ bits}$$

\# subgroups   \# 1 bits   rel. index

Select in $o(n)$ bits $+ \ O(1)$ time



3. Repeat 1. and 2. on reduced bitstrings ($r < (\log n \log\log n)^2$):

   1' Store relative indices of every $(\log\log n)^2$th 1 bit in array

   $$\Rightarrow O\left(\frac{n}{(\log\log n)^2} \log\log n\right) = O\left(\frac{n}{\log\log n}\right) \text{ bits}$$

   2' Within group of $(\log\log n)^2$th 1 bits of length r' bits:

   if $r' \geq (\log\log n)^4$

   then store relative indices of 1 bits in subgroup in array

   $$\Rightarrow O\left(\frac{n}{(\log\log n)^4} (\log\log n)^2 \log\log n\right) = O\left(\frac{n}{\log\log n}\right) \text{ bits}$$

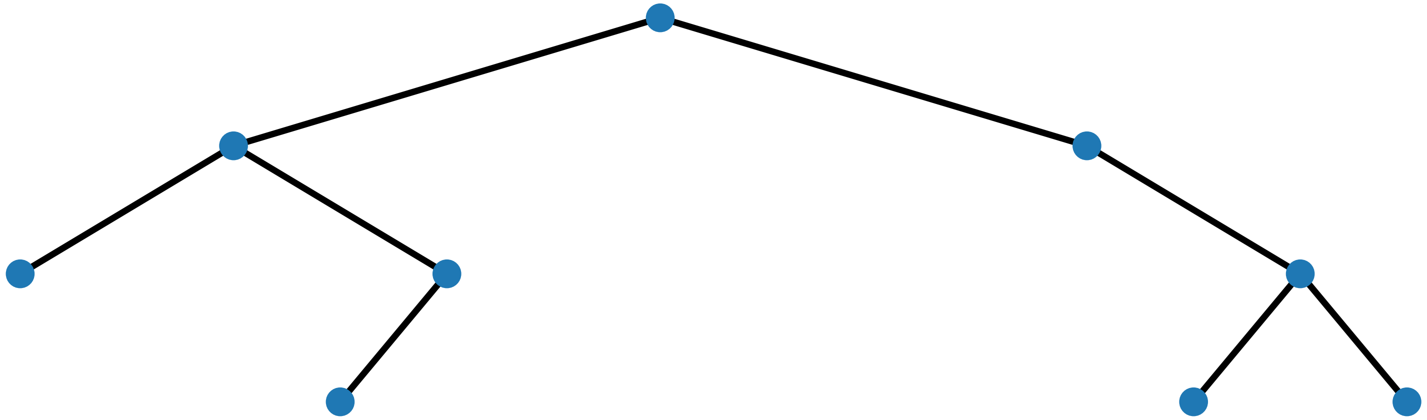
   else problem is reduced to bitstrings of length $r' < (\log\log n)^4$

4. Use lookup table for bitstrings of length $r' \leq (\log\log n)^4 \leq \frac{1}{2}\log n$

   $$\Rightarrow O(\sqrt{n} \log n \log\log n) = o(n) \text{ bits}$$

bitstring query j answer

Succinct representation of binary trees



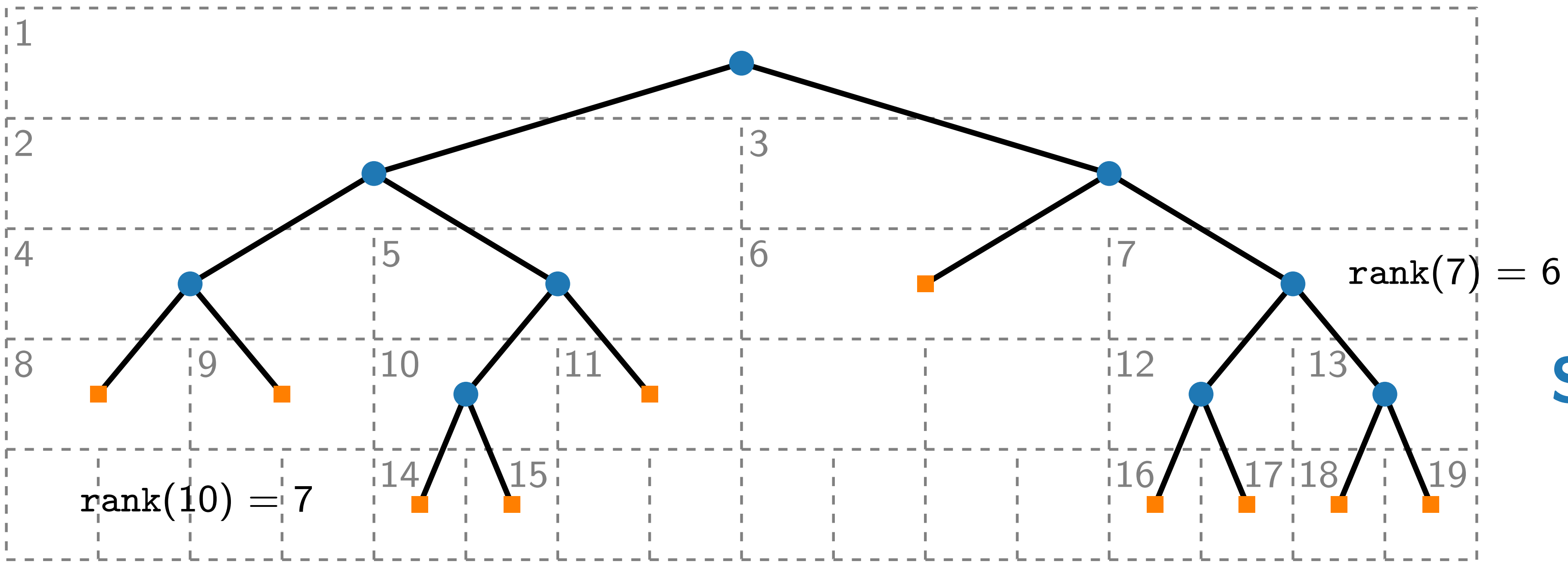
Number of binary trees on n vertices: $C_n = \frac{1}{n+1}\binom{2n}{n}$

$$\log C_n = 2n + o(n) \text{ (by Stirling's approximation)}$$

$\Rightarrow$ We can use $2n + o(n)$ bits to represent binary trees.

Difficulty is when binary tree is not full.

Succinct representation of binary trees



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
b	1	1	1	1	1	0	1	0	0	1	0	1	1	0	0	0	0	0	0

Size.
- 2n +1 bits for b
- $o(n)$ for `rank` and `select`

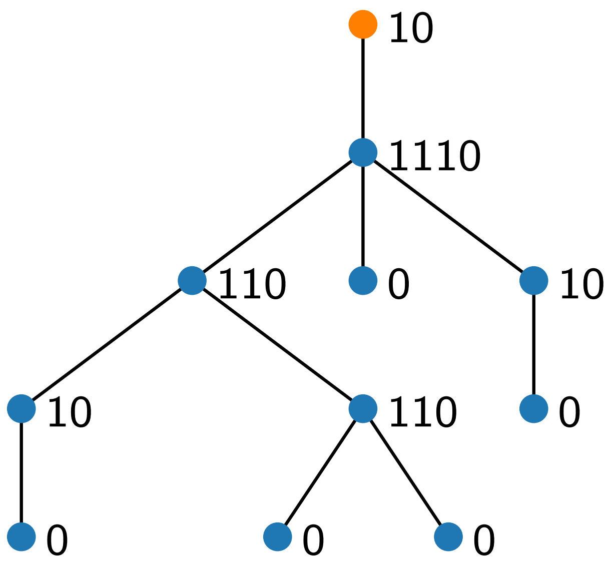
Idea.
- Add external nodes
- Read internal nodes as 1
- Read external nodes as 0
- Use `rank` and `select`

Operations.
- $\texttt{parent}(i) = \texttt{select}(\lfloor \frac{i}{2} \rfloor)$
- $\texttt{leftChild}(i) = 2\,\texttt{rank}(i)$
- $\texttt{rightChild}(i) = 2\,\texttt{rank}(i) + 1$
- $\texttt{rank}(i)$ is index for array storing actual values

Proof is exercise.

Succinct representation of trees - LOUDS

LOUDS = Level Order Unary Degree Sequence



- unary decoding of outdegree
- gives LOUDS sequence

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	1	1	1	0	1	1	0	0	1	0	1	0	1	1	0	0	0	0	0

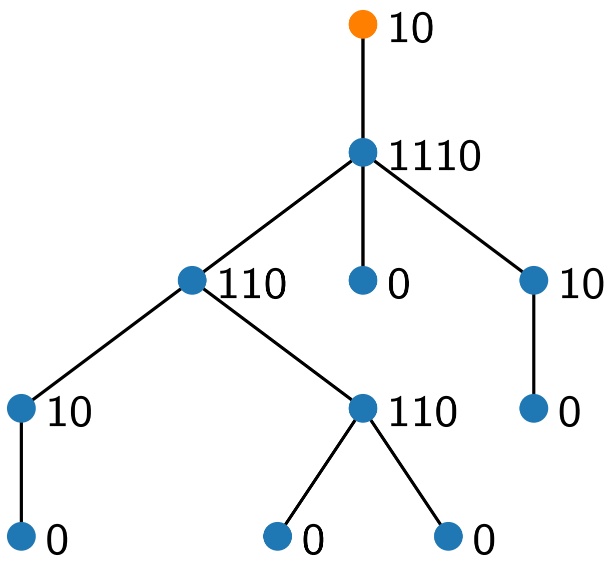
Size.

- each vertex (except root) is represented twice, namely with a 1 and with a 0
- $o(n)$ bits for `rank` and `select`

$\Rightarrow 2n + o(n)$ bits

Succinct representation of trees - LOUDS

LOUDS = Level Order Unary Degree Sequence



- unary decoding of outdegree
- gives LOUDS sequence

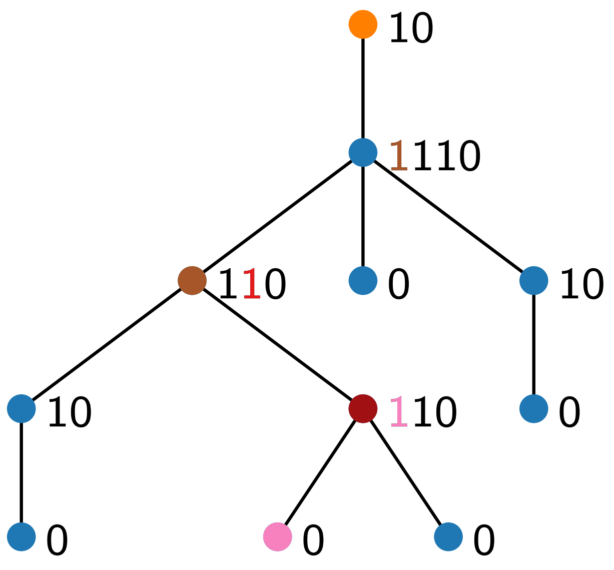
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	1	1	1	0	1	1	0	0	1	0	1	0	1	1	0	0	0	0	0

Operations.

- Let i be index of 1 in louds sequence.
- $\texttt{rank}(i)$ is index for array storing vertex objects/values.

Succinct representation of trees - LOUDS

LOUDS = Level Order Unary Degree Sequence



- unary decoding of outdegree
- gives LOUDS sequence

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	0	1	1	1	0	1	1	0	0	1	0	1	0	1	1	0	0	0	0	0

1	0	1	1	1	0	1	1	0	0	1	0	1	0	1	1	0	0	0	0	0

- $\texttt{firstChild}(i) = \texttt{select}_0(\texttt{rank}_1(i)) + 1$

  $\texttt{firstChild}(8) = \texttt{select}_0(\texttt{rank}_1(8)) + 1$
  $= \texttt{select}_0(6) + 1 = 14 + 1 = 15$

- $\texttt{nextSibling}(i) = i + 1$

- $\texttt{parent}(i) = \texttt{select}_1(\texttt{rank}_0(i))$

  $\texttt{parent}(8) = \texttt{select}_1(\texttt{rank}_0(8))$
  $= \texttt{select}_1(2) = 3$

Exercise: $\texttt{child}(i, j)$ with validity check

Discussion

- Succinct data structures are
  - space efficient
  - support fast operations

  but
  - are mostly static (dynamic at extra cost),
  - number of operations are limited,
  - complex $\rightarrow$ harder to implement
- Rank and select form basis for many succinct representations

Literature

Main reference:

- Lecture 17 of Advanced Data Structures (MIT, Fall'17) by Erik Demaine
- [Jac '89] "Space efficient Static Trees and Graphs"

Recommendations:

- Lecture 18 of Demaine's course on compact & succinct arrays & trees