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 be applied to the data.
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 be applied to the data.

**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.
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 be applied to the data.

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.

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

Goal.
■ Use space “close” to information-theoretical minimum,
■ but still support time-efficient operations.
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 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;
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
Examples for *implicit* data structures

- **arrays** to represent lists
  - but why not linked lists?
Examples for implicit data structures

- arrays to represent lists
  - but why not linked lists?

- 1-dim arrays to represent multi-dimensional arrays
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?
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

```
leftChild(i) = 2i
rightChild(i) = 2i + 1
parent(i) = ⌊i / 2⌋
```

1-dim arrays to represent multi-dimensional arrays
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

  \[
  \text{leftChild}(i) = 2i \\
  \text{rightChild}(i) = 2i + 1 \\
  \text{parent}(i) = \left\lfloor \frac{i}{2} \right\rfloor
  \]
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

leftChild\( (i) = 2i \)

rightChild\( (i) = 2i + 1 \)

parent\( (i) = \left\lfloor \frac{i}{2} \right\rfloor \)

And unbalanced trees?
Succinct indexable dictionary

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

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

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

- **member ($i$)** returns if $i \in S$
- **rank ($i$)** = number of 1's at or before position $i$
- **select ($j$)** = position of $j$th 1 bit
- predecessor and successor can be answered using rank and select

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

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

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

- **member($i$)** returns if $i \in S$
- **rank($i$)** = \# 1's at or before position $i$
- **select($j$)** = position of $j$th 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?
**Succinct indexable dictionary**

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

- $\text{member}(i)$ returns if $i \in S$
- $\text{rank}(i) = \# \text{1's at or before position } i$
- $\text{select}(j) = \text{position of } j\text{th 1 bit}$
- 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?

$$\log 2^n = n \text{ bits}$$
Succinct indexable dictionary

Represent $S$ with a bit vector $b$ of length $n$ where

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

Represent $S$ with a bit vector $b$ of length $n$ where

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

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

$$b = \begin{array}{ccccccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0
\end{array}$$
**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

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

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

$$b = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$
**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

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

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

$\begin{array}{cccccccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
b & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0
\end{array}$

$\text{select}(5) =$
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

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

$$S = \{3, 4, 6, 8, 9, 14\} \text{ where } n = 15 \quad \text{select}(5) = 9$$

$$b = \begin{array}{cccccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
\end{array}$$
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

- rank$(i)$ = number of 1's at or before position $i$
- select$(j)$ = position of $j$th 1 bit

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

select$(5) = 9$

rank$(9) =$
**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

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

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

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
</table>
$b$ | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0   | 0   | 0   | 0   | 1   | 0   |

$\text{select}(5) = 9$
$\text{rank}(9) = 5$
**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

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

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

\[
\begin{array}{cccccccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
b & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0
\end{array}
\]

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

$\text{rank}(9) = 5 = \text{rank}(12)$
**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

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

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

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
</table>
$b$ | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |

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

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

$\text{rank}(15) =$
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

- $\text{rank}(i) =$ # 1’s at or before position $i$
- $\text{select}(j) =$ position of $j$th 1 bit

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

\[
\begin{array}{cccccccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
\hline
b & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
\end{array}
\]

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

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

$\text{rank}(15) = 6$
**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

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

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

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

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

<p>| | | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{select}(5)$</td>
<td>$9$</td>
<td>$\text{rank}(9)$</td>
<td>$5 = \text{rank}(12)$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\text{rank}(15)$</td>
<td>$6$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Rank in $o(n)$ bits

$b$
Rank in $o(n)$ bits

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

and store cumulative rank: each $\log n$ bits
Rank in $o(n)$ bits

1. Split into $(\log^2 n)$-bit chunks and store cumulative rank: each $\log n$ bits
Rank in $o(n)$ bits

1. Split into $(\log^2 n)$-bit chunks
   and store cumulative rank: each log $n$ bits

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

```
   b   1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
   3 5 1 3
   \frac{1}{2} \log n
   log^2 n
```
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
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

   $$\Rightarrow O\left(\frac{n}{\log n \log \log n}\right) \subseteq o(n) \text{ bits}$$
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 \((\frac{1}{2} \log n)\)
   \[ \Rightarrow O \left( \sqrt{n \log n \log \log n} \right) \subseteq o(n) \text{ bits} \]
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 $(\frac{1}{2} \log n)$
   \[\Rightarrow O\left(\sqrt{n} \log n \log \log n\right) \subseteq o(n) \text{ bits}\]

4. rank = rank of chunk + relative rank of subchunk within chunk + relative 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
   \[ \Rightarrow O\left( \frac{n}{\log^2 n} \log n \right) = O\left( \frac{n}{\log n} \right) \subseteq o(n) \text{ bits} \]

2. Split **chunks** into $(1/2 \log n)$-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 $(1/2 \log n)$
   \[ \Rightarrow O\left( \sqrt{n} \log n \log \log n \right) \subseteq o(n) \text{ bits} \]

4. **rank** = rank of **chunk**
   + relative rank of **subchunk** within **chunk**
   \[ \Rightarrow O(1) \text{ time} \]
   + relative rank of element within **subchunk**
Select in $o(n)$ bits

$b$
Select in $o(n)$ bits

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

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

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

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

\}  log n log log n 1’s

- # groups
- # 1 bits
- index
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 $r < (\log n \log \log n)^2$
Select in $o(n)$ bits

1. Store indices of every $(\log n \log \log n)$th 1 bit in array
   
   $\Rightarrow O\left(\frac{n}{\log n \log \log n} \log n\right) = O\left(\frac{n}{\log \log n}\right) \subseteq o(n)$ 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 \((r < (\log n \log \log n)^2)\):
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
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

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

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

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

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

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

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

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

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

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

\[ \Rightarrow O\left(\sqrt{n \log n \log \log n}\right) = o(n) \text{ bits} \]
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 $r' \leq (\log \log n)^4 \leq \frac{1}{2} \log n$

\[ \Rightarrow O\left(\sqrt{n \log n \log \log n}\right) = o(n) \text{ bits} \]
Succinct representation of binary trees

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

$log C_n = 2n + o(n)$ (by Stirling’s approximation)
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.
Succinct representation of binary trees

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

$log C_n = 2n + o(n)$ (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

Idea.

- Add external nodes
- Read internal nodes as 1
- Read external nodes as 0
- Use rank and select
Succinct representation of binary trees

Idea.
- Add external nodes
- Read internal nodes as 1
- Read external nodes as 0
- Use rank and select
Succinct representation of binary trees

Idea.
- Add external nodes
- Read internal nodes as 1
- Read external nodes as 0
- Use rank and select
Succinct representation of binary trees

Idea.
- Add external nodes
- Read internal nodes as 1
- Read external nodes as 0
**Succinct representation of binary trees**

**Idea.**
- Add external nodes
- Read internal nodes as 1
- Read external nodes as 0

**Operations.**
- \( \text{parent}(i) = ? \)
- \( \text{leftChild}(i) = ? \)
- \( \text{rightChild}(i) = ? \)
Succinct representation of binary trees

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

Operations.
- $\text{parent}(i) = \ ?$
- $\text{leftChild}(i) = \ ?$
- $\text{rightChild}(i) = \ ?$
Succinct representation of binary trees

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

Operations.
- parent(i) = ?
- leftChild(i) = ?
- rightChild(i) = ?
Succinct representation of binary trees

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

Operations.
- parent(i) = ?
- leftChild(i) = ?
- rightChild(i) = ?
Succinct representation of binary trees

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

Operations.
- parent(i) = ?
- leftChild(i) = 2 rank(i)
- rightChild(i) = 2 rank(i) + 1

\[
\begin{array}{cccccccccccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]
Succinct representation of binary trees

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

Operations.
■ \( \text{parent}(i) = \text{select}(\lfloor \frac{i}{2} \rfloor) \)
■ \( \text{leftChild}(i) = 2 \text{rank}(i) \)
■ \( \text{rightChild}(i) = 2 \text{rank}(i) + 1 \)
Succinct representation of binary trees

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

Operations.
- parent(i) = select(⌊i/2⌋)
- leftChild(i) = 2 rank(i)
- rightChild(i) = 2 rank(i) + 1

Proof is exercise.
Succinct representation of binary trees

Operations.

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

Idea.

\- Add external nodes
\- Read internal nodes as 1
\- Read external nodes as 0
\- Use rank and select
**Succinct representation of binary trees**

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

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

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

**Proof is exercise.**
Succinct representation of trees - **LOUDS**

**LOUDS** = Level Order Unary Degree Sequence
Succinct representation of trees - **LOUDS**

**LOUDS** = Level Order Unary Degree Sequence
Succinct representation of trees - **LOUDS**

**LOUDS** = **Level** **Order** **Unary** **Degree** **Sequence**

- unary decoding of outdegree
Succinct representation of trees - **LOUDS**

**LOUDS** = Level Order Unary Degree Sequence

![Diagram of LOUDS representation](image)
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
0  0  0  1  1  1  0  0  1  0  0  1  0  0  1  0  1  1  0  0  0  0
```
Succinct representation of trees - **LOUDS**

**LOUDS = Level Order Unary Degree Sequence**

- **unary decoding of outdegree**
- **gives LOUDS sequence**

Size.
- each vertex (except root) is represented twice, namely with a 1 and with a 0
- \( o(n) \) bits for rank and select
Succinct representation of trees - LOUDS

LOUDS = Level Order Unary Degree Sequence

- unary decoding of outdegree
- gives LOUDS sequence

1 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

\[ \Rightarrow 2n + o(n) \text{ bits} \]
Succinct representation of trees - **LOUDS**

**LOUDS** = Level Order Unary Degree Sequence

- unary decoding of outdegree
- gives LOUDS sequence

Operations.

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

```
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
```
Succinct representation of trees - LOUDS

LOUDS = Level Order Unary Degree Sequence

- unary decoding of outdegree
- gives LOUDS sequence

\[
\text{firstChild}(i) = \text{select}_0(\text{rank}_1(i)) + 1
\]
Succinct representation of trees - LOUDS

LOUDS = Level Order Unary Degree Sequence

- unary decoding of outdegree
- gives LOUDS sequence

\[ \text{firstChild}(i) = \text{select}_0(\text{rank}_1(i)) + 1 \]

\[ \text{firstChild}(8) = \text{select}_0(\text{rank}_1(8)) + 1 \]
Succinct representation of trees - LOUDS

LOUDS = Level Order Unary Degree Sequence

- unary decoding of outdegree
- gives LOUDS sequence

firstChild(i) = select₀(rank₁(i)) + 1

firstChild(8) = select₀(rank₁(8)) + 1 = select₀(6) + 1
Succinct representation of trees - LOUDS

LOUDS = Level Order Unary Degree Sequence

- unary decoding of outdegree
- gives LOUDS sequence

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

firstChild(8) = select\(_0\)(rank\(_1\)(8)) + 1
= select\(_0\)(6) + 1 = 14 + 1 = 15
Succinct representation of trees - LOUDS

LOUDS = Level Order Unary Degree Sequence

- unary decoding of outdegree
- gives LOUDS sequence

firstChild(i) = select_0(rank_1(i)) + 1

firstChild(8) = select_0(rank_1(8)) + 1
= select_0(6) + 1 = 14 + 1 = 15

nextSibling(i) = i + 1
Succinct representation of trees - **LOUDS**

**LOUDS** = **Level Order Unary Degree Sequence**

- **unary decoding of outdegree**
- **gives LOUDS sequence**

- `firstChild(i) = select_0(rank_1(i)) + 1`
- `firstChild(8) = select_0(rank_1(8)) + 1`
  = `select_0(6) + 1 = 14 + 1 = 15`

- `nextSibling(i) = i + 1`

**Exercise:** `child(i, j)` with validity check
Succinct representation of trees - **LOUDS**

**LOUDS** = **Level** Order **Unary** Degree **Sequence**

- unary decoding of outdegree
- gives LOUDS sequence

**firstChild**($i$) = $\text{select}_0(\text{rank}_1(i)) + 1$

firstChild(8) = $\text{select}_0(\text{rank}_1(8)) + 1$

= $\text{select}_0(6) + 1 = 14 + 1 = 15$

**parent**($i$) = $\text{select}_1(\text{rank}_0(i))$

**nextSibling**($i$) = $i + 1$

**Exercise:** $\text{child}(i, j)$ with validity check
Succinct representation of trees - LOUDS

LOUDS = Level Order Unary Degree Sequence

- unary decoding of outdegree
- gives LOUDS sequence

- firstChild(i) = select_0(rank_1(i)) + 1
- parent(i) = select_1(rank_0(i))
- nextSibling(i) = i + 1

Exercise: child(i, j) with validity check
Succinct representation of trees - LOUDS

LOUDS = Level Order Unary Degree Sequence

- unary decoding of outdegree
- gives LOUDS sequence

\[ \text{firstChild}(i) = \text{select}_0(\text{rank}_1(i)) + 1 \]

\[ \text{firstChild}(8) = \text{select}_0(\text{rank}_1(8)) + 1 = \text{select}_0(6) + 1 = 14 + 1 = 15 \]

\[ \text{nextSibling}(i) = i + 1 \]

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

\[ \text{parent}(i) = \text{select}_1(\text{rank}_0(i)) \]

\[ \text{parent}(8) = \text{select}_1(\text{rank}_0(8)) = \text{select}_1(2) \]
Succinct representation of trees - LOUDS

LOUDS = Level Order Unary Degree Sequence

- **unary decoding of outdegree**
- **gives LOUDS sequence**

```
1 0 1 1 1 0 1 1 0 0 1 0 1 0 1 1 0 0 0 0
1 0 1 1 1 0 1 1 0 0 1 0 1 0 1 1 0 0 0 0
```

- `firstChild(i) = select_0(rank_1(i)) + 1`
- `firstChild(8) = select_0(rank_1(8)) + 1`
  = `select_0(6) + 1 = 14 + 1 = 15`

- `nextSibling(i) = i + 1`

**Exercise:** `child(i, j)` with validity check

```
10 110 0 10
10 110 0 0
```

- `parent(i) = select_1(rank_0(i))`
- `parent(8) = select_1(rank_0(8))`
  = `select_1(2) = 3`
Discussion

- Succinct data structures are
  - space efficient
  - support fast operations
  
  but
  - are mostly static (dynamic at extra cost),
  - number of operations are limited,
  - complex → harder to implement
Discussion

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