Advanced Algorithms

Winter term 2019/20

Lecture 9. Succinct data structures
(Based on lectures from Simon Gog and from Erik Demaine)

Jonathan Klawitter
Chair for Computer Science I
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.
Examples for *implicit* data structures
Examples for *implicit* data structures

- **array** to represent list; but why not linked list?
Examples for implicit data structures

- array to represent list; but why not linked list?
- 1-dim array to represent multi-dimensional array
Examples for implicit data structures

- array to represent list; but why not linked list?
- 1-dim array to represent multi-dimensional array
- sorted array to represent sorted list; but why not binary search tree?
Examples for **implicit** data structures

- **array** to represent list; but why not linked list?
- **1-dim array** to represent multi-dimensional array
- **sorted array** to represent sorted list; but why not binary search tree?
- **array** to represent complete binary tree or heap

![Diagram of a binary heap]

leftChild(i) =

rightChild(i) =

parent(i) =
Examples for implicit data structures

- array to represent list; but why not linked list?

- 1-dim array to represent multi-dimensional array

- sorted array to represent sorted list; but why not binary search tree?

- array to represent complete binary tree or heap

1-dim array to represent multi-dimensional array

```
1 2 3 4 5 6 7 ...
```

leftChild(i) = 2i
rightChild(i) = 2i + 1
parent(i) = ⌊i/2⌋
Examples for implicit data structures

- **array** to represent list; but why not linked list?
- **1-dim array** to represent multi-dimensional array
- **sorted array** to represent sorted list; but why not binary search tree?
- **array** to represent complete binary tree or heap

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

And unbalanced trees?
Succinct indexable dictionary

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

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

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

- $\text{member}(i)$ returns if $i \in S$
- $\text{rank}(i) =$ number of 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 $\text{select}$

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 $O(1)$ 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 $O(1)$ operations:

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

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

How many bits of space do we need to distinguish them? $\log 2^n = n$ 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 $n = 15$

$$b = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$
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 $n = 15$

$$b = [0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0]$$
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 $i$
- $\text{select}(j) = \text{position of } j\text{th 1 bit}$

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

\[ b = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \]

$\text{select}(5) =$
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

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

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

```
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 0 1 0
```

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

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

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

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

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

$\text{rank}(9) =$
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

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

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

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

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

$\text{rank}(9) = 5$
Rank in $o(n)$ 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
   $$\Rightarrow \mathcal{O}\left(\frac{n}{\log^2 n} \log n\right) = \mathcal{O}\left(\frac{n}{\log n}\right) \subseteq o(n) \text{ bits}$$

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

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

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

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

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

2. Within group of $(\log n \log \log n)$ 1 bits, say $r$ bits:
Select in $o(n)$ bits

1. Store indices of every $(\log n \log \log n)$th 1 bit in array
   $$\Rightarrow O\left(\frac{n}{\log n \log \log n} \log n\right) = O\left(\frac{n}{\log \log n}\right) = o(n) \text{ bits}$$

2. Within group of $(\log n \log \log n)$ 1 bits, say $r$ bits:
   
   if $r \geq (\log n \log \log n)^2$
   then store indices of 1 bits in group in array
   $$\Rightarrow O\left(\frac{n}{(\log n \log \log n)^2} (\log n \log \log n) \log n\right) = O\left(\frac{n}{\log \log n}\right)$$

# groups  # 1 bits  index
Select in $o(n)$ bits

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

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

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

- if $r \geq (\log n \log \log n)^2$
  then store indices of 1 bits in group in array

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

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

2. Within group of $(\log n \log \log n)$ 1 bits, say $r$ bits:
   if $r \geq (\log n \log \log n)^2$
   then store indices of 1 bits in group in array
   \[ \Rightarrow O\left(\frac{n}{(\log n \log \log n)^2} (\log n \log \log n \log n) \log n\right) = O\left(\frac{n}{\log \log n}\right) \]
   else 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

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

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

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

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

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

then store relative indices of 1 bits in subgroup in array

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

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

2’. Within group of $(\log \log n)^2$th 1 bits, say $r'$ bits:

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

then store relative indices of 1 bits in subgroup in array

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

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

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

if $r' \geq (\log \log n)^4$
then store relative indices of 1 bits in subgroup in array

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

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

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

$$\Rightarrow O\left(\sqrt{n} \log n \log \log n\right) = o(n) \text{ bits}$$
Select in $o(n)$ bits and $O(1)$ time $(\log \log n)^2$ 1’s

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, say $r'$ 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 reduced to bitstrings of length $r' < (\log \log n)^4$

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

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

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

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

Operations we want to support:
parent($v$), leftChild($v$), rightChild($v$)
Succinct representation of binary trees

Number of binary trees on \( n \) vertices: 
\[
C_n = \frac{1}{n+1} \binom{2n}{n}
\]

\[
\log C_n = 2n + o(n) \quad \text{(by Stirling’s approximation)}
\]

Operations we want to support:
\[
\text{parent}(v), \text{leftChild}(v), \text{rightChild}(v)
\]

Idea:
- add \textbf{external} nodes
- read \textbf{internal} nodes as 1
- read \textbf{external} nodes as 0
- use rank and select
Succinct representation of binary trees
Succinct representation of binary trees
Succinct representation of binary trees
Succinct representation of binary trees
Succinct representation of binary trees

leftChild(i) = 2^\text{rank}(i)
rightChild(i) = 2^\text{rank}(i) + 1

\begin{tabular}{cccccccccccccccccc}
  \hline
  \text{b} & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0
  \hline
\end{tabular}
Succinct representation of binary trees

![Binary Tree Diagram]

- leftChild(i) = 2 \cdot \text{rank}(i)
- rightChild(i) = 2 \cdot \text{rank}(i) + 1

\text{rank}(7) = 6
Succinct representation of binary trees

\[ \text{leftChild}(i) = 2^\text{rank}(i) \]

\[ \text{rightChild}(i) = 2^\text{rank}(i) + 1 \]

\[
\begin{array}{c}
\text{rank}(7) = 6 \\
\text{rank}(10) = 7
\end{array}
\]

\[ b = 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \]

\[
\begin{array}{c}
\text{leftChild}(i) = \\
\text{rightChild}(i) =
\end{array}
\]
Succinct representation of binary trees

- leftChild(i) = $2 \times \text{rank}(i)$
- rightChild(i) = $2 \times \text{rank}(i) + 1$

\[
\begin{array}{ccccccccccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 \\
1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0
\end{array}
\]
Succinct representation of binary trees

- leftChild(i) = 2 \text{rank}(i)
- rightChild(i) = 2 \text{rank}(i) + 1
- parent(i) = 

\text{rank}(7) = 6
Succinct representation of binary trees

leftChild(i) = 2 \cdot \text{rank}(i)

rightChild(i) = 2 \cdot \text{rank}(i) + 1

parent(i) = \text{select}(\lfloor \frac{i}{2} \rfloor)
Succinct representation of binary trees

- leftChild\( (i) = 2 \text{rank}(i) \)
- rightChild\( (i) = 2 \text{rank}(i) + 1 \)
- parent\( (i) = \text{select}(\lfloor \frac{i}{2} \rfloor) \)

use \text{rank}(i) for index in array storing actual values
Succinct representation of binary trees

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

- Size: 2n + 1 bits for $b$, plus $o(n)$ for rank and select

- use rank(i) for index in array storing actual values
Succinct representation of trees - LOUDS

Level order unary degree sequence
Succinct representation of trees - LOUDS

Level order unary degree sequence
Succinct representation of trees - LOUDS

Level order unary degree sequence

- unary decoding of outdegree
Succinct representation of trees - LOUDS

Level order unary degree sequence

unary decoding of outdegree
Succinct representation of trees - LOUDS

Level order unary degree sequence
- unary decoding of outdegree
- gives LOUDS sequence

- each node represented twice
- use index of its corresponding 1
Succinct representation of trees - LOUDS

Level order unary degree sequence
- unary decoding of outdegree
- gives LOUDS sequence

- each node represented twice
- use index of its corresponding 1

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

Level order unary degree sequence
- unary decoding of outdegree
- gives LOUDS sequence

- each node represented twice
- use index of its corresponding 1

⇒ $2n + o(n)$ bits

- firstChild($i$) = select$_0$(rank$_1$(i)) + 1
Succinct representation of trees - LOUDS

Level order unary degree sequence
- unary decoding of outdegree
- gives LOUDS sequence
- each node represented twice
- use index of its corresponding 1

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

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

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

Level order unary degree sequence

- unary decoding of outdegree
- gives LOUDS sequence

- each node represented twice
- use index of its corresponding 1

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

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

\[ = \text{select}_0(6) + 1 \]

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

Level order unary degree sequence

- unary decoding of outdegree
- gives LOUDS sequence

- each node represented twice
- use index of its corresponding 1

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

\[ \text{firstChild}(8) = \text{select}_0(\text{rank}_1(8)) + 1 = \text{select}_0(6) + 1 = 10 + 1 = 11 \]

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

Level order unary degree sequence

- unary decoding of outdegree
- gives LOUDS sequence

- each node represented twice
- use index of its corresponding 1

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

\[ \text{firstChild}(8) = \text{select}_0(\text{rank}_1(8)) + 1 = \text{select}_0(6) + 1 = 10 + 1 = 11 \]

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

Level order unary degree sequence
- unary decoding of outdegree
- gives LOUDS sequence
- each node represented twice
- use index of its corresponding 1
  ⇒ 2n + o(n) bits

■ firstChild(i) = select_0(rank_1(i)) + 1
  firstChild(8) = select_0(rank_1(8)) + 1
  = select_0(6) + 1 = 10 + 1 = 11

■ nextSibling(i) = i + 1

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

Level order unary degree sequence
- unary decoding of outdegree
- gives LOUDS sequence
- each node represented twice
- use index of its corresponding 1
  ⇒ 2n + o(n) bits

- firstChild(i) = select₀(rank₁(i)) + 1
  firstChild(8) = select₀(rank₁(8)) + 1
  = select₀(6) + 1 = 10 + 1 = 11

- nextSibling(i) = i + 1

- parent(i) = select₁(rank₀(i))

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

Level order unary degree sequence

- unary decoding of outdegree
- gives LOUDS sequence

- each node represented twice
- use index of its corresponding 1

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

\[ \text{firstChild}(8) = \text{select}_0(\text{rank}_1(8)) + 1 \]
\[ = \text{select}_0(6) + 1 = 10 + 1 = 11 \]

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

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

\[ \text{parent}(8) = \text{select}_1(\text{rank}_0(8)) = \text{select}_1(2) = 3 \]

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

- Rank and select form basis for many succinct representations
References

- Lecture 17 of Advanced Data Structures (MIT, Fall’17) by Erik Demaine
- see also Lecture 18 on compact & succinct suffix arrays & trees
References

- Lecture 17 of Advanced Data Structures (MIT, Fall’17) by Erik Demaine
- see also Lecture 18 on compact & succinct suffix arrays & trees

- Guy Jacobson “Space efficient Static Trees and Graphs”, FOCS’89
- also contains how to store planar graphs in linear space