Advanced Data Structures
Simon Gog – gog@kit.edu
Definition

A succinct data structure uses space „close” to the information-theoretical lower bound, but still supports operations time-efficiently.

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
Example: Succinct indexable dictionary

Example 1
Represent a subset $S \subset [n]$ and support operations:
- $\text{member}(i)$ returns if $i \in S$
- $\text{rank}(i) = |\{s \mid s \in S \land s < i\}|$
- $\text{select}(j) = \min\{k \mid \text{rank}(k + 1) = j \land k \in [n]\}$
- $\text{predecessor}$ and $\text{successor}$ can be answered by using $\text{rank}$ and $\text{select}$

There are $2^n$ different subsets, i.e. we need $\log 2^n = n$ bits of space to distinguish between dictionaries.
Example: Succinct indexable dictionary

Solution
Use a bitvector \( b \) of length \( n \) with

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

plus \( o(n) \)-space indexes to answer

- \( rank(i, 1, b) = \sum_{j=0}^{i-1} b[i] \)
- \( select(j, 1, b) = \min\{ i \mid rank(i + 1, 1, b) = j \} \)
Example: Succinct indexable dictionary

$o(n)$ rank index (Jacobson, FOCS 1989)

- Partition $b$ into super blocks of size $\alpha = \lceil \log^2 n \rceil$
- Store absolute count $A[i] = \sum_{k=0}^{\alpha - 1} b[k]$ for each block $i$ in $\log n$ bits
- Partition each super block into blocks of size $bs = \beta = \lceil \frac{1}{2} \log n \rceil$
- Store relative count $R[i, j] = (\sum_{k=0}^{\beta j - 1} b[k + \alpha i]) - A[i]$ for each block $j$ in super block $i$.
- Use „four Russian trick” for blocks of size $\beta$. Pre-compute lookup table LTB of size $2^{\beta} \beta \log \beta = o(n)$
- Total space: $\frac{n \log n}{\alpha} + \frac{n \log \log n}{\beta} + O(1) = o(n)$ bits
Example: Succinct indexable dictionary

A simple solution to select: binary search over the $o(n)$-space rank structure.

$O(n)$ select structure

- See blackboard.
First consider binary trees

- Number of $n$-node binary trees: $C_n = \frac{1}{n+1} \binom{2n}{n}$
- We need $\log C_n = 2n + o(n)$ bits (using Sterling’s Approximation)
- Operations: $\text{parent}(v)$, $\text{leftchild}(v)$, $\text{rightchild}(v)$
First consider binary trees

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

We need $\log C_n = 2n + o(n)$ bits (using Sterling’s Approximation)

Operations: $\text{parent}(v), \text{leftchild}(v), \text{rightchild}(v)$
Succinct representation of trees

- First consider binary trees
- Number of $n$-node binary trees: $C_n = \frac{1}{n+1} \binom{2n}{n}$
- We need $\log C_n = 2n + o(n)$ bits (using Sterling’s Approximation)
- Operations: $\text{parent}(v)$, $\text{leftchild}(v)$, $\text{rightchild}(v)$
First consider binary trees

Number of \( n \)-node binary trees: \( C_n = \frac{1}{n+1} \binom{2n}{n} \)

We need \( \log C_n = 2n + o(n) \) bits (using Sterling’s Approximation)

Operations: \( \text{parent}(v), \text{leftchild}(v), \text{rightchild}(v) \)
Succinct representation of trees

- First consider binary trees

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

- We need $\log C_n = 2n + o(n)$ bits (using Sterling’s Approximation)

- Operations: $\text{parent}(v), \text{leftchild}(v), \text{rightchild}(v)$
Succinct representation of trees

- First consider binary trees
- Number of \( n \)-node binary trees: \( C_n = \frac{1}{n+1} \binom{2n}{n} \)
- We need \( \log C_n = 2n + o(n) \) bits (using Sterling’s Approximation)
- Operations: \( \text{parent}(v) \), \( \text{leftchild}(v) \), \( \text{rightchild}(v) \)
Succinct representation of trees

- In a very balanced binary tree (like in a heap) operations are easy
- Let 0 be the root identifier
  - \( \text{parent}(v) = \left\lfloor \frac{v-1}{2} \right\rfloor \) for \( v > 0 \)
  - \( \text{leftchild}(v) = 2v + 1 \)
  - \( \text{rightchild}(v) = 2v + 2 \)
- For unbalanced trees the space would be \( 2^d \) bits, where \( d \) is the maximum depth of a node.

Proposal of Jacobson (FOCS 1989):
1. Mark all the nodes of the tree with a 1.
2. Add external nodes to the tree, and mark them all with 0-bits.
3. Read off the bits marking the nodes of the tree in (left-to-right) level-order.
Succinct representation of trees

$b = \begin{array}{cccccccccccccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18
\end{array}$

$\text{rank}(0, 1, b) = 0$

$\text{rank}(1, 1, b) = 1$

$\text{rank}(2, 1, b) = 2$
Succinct representation of trees

- $b$ contains $n$ set bits and is of length $2n + 1$.
- A node is represented by the position of its corresponding 1-bit in $b$.
  - $\text{leftchild}(v) = 2 \cdot \text{rank}(v) + 1$
  - $\text{rightchild}(v) = 2 \cdot \text{rank}(v) + 2$
  - $\text{parent}(v) = \text{select}(\lfloor \frac{v-1}{2} \rfloor + 1, 1, b)$ for $v > 0$
- Total space (including rank and select): $2n + o(n)$ bits

Jaboson also considered rooted, ordered tree with degree higher than 2.
LOUDS – level order unary degree sequence
LOUDS – level order unary degree sequence

pseudo root
LOUDS – level order unary degree sequence

unary decoding of out degree $\rightarrow$ 01 $\leftarrow$ pseudo root
LOUDS – level order unary degree sequence

LOUDS sequence = 0100010011010101111
Each node (except the pseudo root) is represented twice
- Once as „0“ in the child list of its parent
- Once as the terminal („1“) in its child list

Represent node $v$ by the index of its corresponding „0“

I.e. $root$ corresponds to „0“
LOUDS – level order unary degree sequence

00 is_leaf(v)
01 id ← rank(v, 0, LOUDS)
02 p ← select(id + 1, 1, LOUDS)
03 if p + 1 = LOUDS.size() or LOUDS[p + 1] = 1 then
04 return true
05 return false

00 out_degree(v)
01 if is_leaf(v) then
02 return 0
03 id ← rank(v, 0, LOUDS)
04 return select(id + 2, 1, LOUDS) − select(id + 1, 1, LOUDS) − 1
LOUDS – level order unary degree sequence

Get $i$-th child of $v$ and parent:

00  \textbf{child}(v, i)
01    \textbf{if} $i > \text{out\_degree}(v)$ \textbf{then}
02    \textbf{return} ⊥
03    \textbf{id} ← \text{rank}(v, 0, \text{LOUDS})
04    \textbf{return} \text{select}(\text{id} + 1, 1, \text{LOUDS}) + i

00  \textbf{parent}(v)
01    \textbf{if} \text{is\_root}(v) \textbf{then}
02    \textbf{return} ⊥
01    \textbf{pid} ← \text{rank}(v, 1, \text{LOUDS})
04    \textbf{return} \text{select}(\text{pid}, 0, \text{LOUDS})
Conclusion

- Total space for LOUDS representation is $2n + 1 + o(n)$ bits
- All operations take constant time