Advanced Data Structures

Lecture 02: Succinct Trees

Florian Kurpicz

The slides are licensed under a Creative Commons Attribution-ShareAlike 4.0 International License. 
www.creativecommons.org/licenses/by-sa/4.0 | commit 3c6d2d4 compiled at 2023-04-24-09:00
Recap: Rank Queries on Bit Vectors (1/2)

\[ \text{rank}_\alpha(i) \] \# of \( \alpha \)s before \( i \)

\[ \text{select}_\alpha(j) \] position of \( j \)-th \( \alpha \)

\[ \text{rank}_0(5) \] \# of 0s w.r.t. super-block

\[ \text{select}_1(5) \] # of 0s w.r.t. BV

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
</table>

Block

Super-block
Lemma: Binary Rank- and Select-Queries

Given a bit vector of size $n$, there exist data structures that can be computed in time $O(n)$ of size $o(n)$ bits that can answer rank and select queries on the bit vector in $O(1)$ time.

Word RAM

- unlimited memory
- words of size $w \in \Theta \log n$
- constant time load and store
- constant time bit operations on words
Plan for Today

- represent tree with \( n \) nodes using \( 2n \) bits
- make succinct tree fully-functional using additional \( o(n) \) bits

- trees are important
  - searching for keys
  - maintaining directories
  - representations of parsings
  - ...

- different representations
- supporting different operations
Preliminaries

- a tree is an acyclic connected graph $G = (V, E)$ with a root $r \in V$
- degree $\delta$ is the number of children
- leaves have degree 0
- depth of a node is the length of the path from the root to that node
Level Ordered Unary Degree Sequence (1/2) \[\text{Jac88}\]

- represent tree level-wise
- use \(\leq 2\) bits per node

**Definition: LOUDS**
Starting at the root, all nodes on the same depth
- are visited from left to right and
- for node \(v\), \(\delta(v)\) 1’s followed by a 0 are appended to the bit vector that contains an initial 10

**Lemma: Space Usage of LOUDS**
Representing a tree with \(n\) nodes requires \(2n + 1\) bits using LOUDS

- write down the LOUDS representation of this example tree
Level Ordered Unary Degree Sequence (2/2)

ab ch id ejkfg
1011100110011001100000

- node start at pertinent 0
What is Fully-Functional?

Operations
- degree \(i\) is leaf
- \(i\)-th child
- parent
- subtree size
- depth
- lowest common ancestor
- rank (pre- or post-order)
- ...

[Diagram of a tree with nodes labeled a, b, c, d, e, f, g, h, i, j, k]
Making LOUDS Fully-Functional

- degree of $p$: $\text{degree of } p = p - \text{select}_0(\text{rank}_0(p)) - 1$
- $i$-th child of $p$:
  $\text{select}_0(\text{rank}_1(\text{select}_0(\text{rank}_0(p)))) + i + 1$
- parent of $p$:
  $\text{select}_0(\text{rank}_0(\text{select}_1(\text{rank}_0(p)))) + 1$

- explanation on the board 📚

- subtree size 📊 PINGO
From Bit Vectors to Parentheses

- instead of 0 and 1
- use ( and )

requires the same space
- can add relation between parentheses

Definition: Balanced String of Parentheses

A string of parentheses is balanced, if for each left parenthesis there exist unique right parenthesis to its right.

- findclose(i): find the right parenthesis matching the left parenthesis at position i
- findopen(i): find the left parenthesis matching the right parenthesis at position i
- excess(i): find the difference between the number of left and right parentheses before position i
- enclose(i): given a parentheses pair with the left parenthesis at position i, return the position of the closest left parenthesis belonging to the parentheses pair enclosing it

- how can we answer excess queries?
all parentheses operations can be answered in $O(1)$ time using $o(n)$ bits space

here, a little bit simpler

- $excess(i) = rank_{[\cdot]}(i) - rank_{[\cdot]}(i)$
- $fwd\_search(i, d) = \min\{j > i : excess(j) - excess(i - 1) = d\}$
- $bwd\_search(i, d) = \max\{j < i : excess(i) - excess(j - 1) = d\}$

- $findclose(i) = fwd\_search(i, 0)$
- $findopen(i) = bwd\_search(i, 0)$
- $enclose(i) = bwd\_search(i, 2)$

can be answered with a min-max-tree
Definition: Range Min-Max Tree

Given a bit vector $B$ of length $n$ and a block size $b$, store for each consecutive block (from $s$ to $e$) of $BV$

- total excess in block: $\text{excess}(e) - \text{excess}(s - 1)$
- minimum left-to-right excess in block: $\min\{\text{excess}(p) - \text{excess}(s - 1) : p \in [s, e]\}$

and build a binary tree over these blocks, where each node stores the same total information for blocks in all its leaves.

Lemma: Range Min-Max Tree Space

A range min-max tree with block size $b$ for a bit vector of size $n$ requires $n + O((n/b) \log n)$ bits of space.
### fwdsearch in a Range Min-Max Tree

- scan block
- if not found traverse tree
- identify block in tree
- scan block

- process $c$ bits at a time
- first align with next $c$ bits
- requires $O(c + b/c)$ time

- going up and down tree in $O(\log(n/b))$ time
- scanning last block requires $O(c + b/c)$ time

### Improvements

- two level approach
- build range min-max trees for chunks of size $\Theta(\log^3 n)$
- $O(\log \log n)$ query time inside a chunk
- can result in total query time of $O(\log \log n)$

- by choosing $b = c \log n$ this requires
- $O(\log n)$ time and
- $n + O(n/(c \log n)) = n + o(n)$ bits space
Balanced Parentheses (1/2) [MR01]

- represent tree as depth-first traversal
- using balanced parentheses

Definition: BP

Starting at the root, traverse the tree in depth-first order and append a
- left parenthesis if a node is visited the first time
- right parenthesis if a node is visited the last time
to the bit vector

Lemma: Space Usage of BP

Representing a tree with $n$ nodes requires $2n$ bits using $BP$
Balanced Parentheses (2/2)

- node starts at first parenthesis
- subtree structure is encoded in parentheses

ab cd ef g h ij k
(()()(()()())(()()())(()())())
Making BP Fully-Functional

- subtree size of p: \( \frac{\text{findclose}(p) - p + 1}{2} \)
- parent of p: \( \text{enclose}(p) \)

Complicated Constant Time [NS14]
- degree
- \( i \)-th child

\[
\text{subtree size of } p = \frac{\text{findclose}(p) - p + 1}{2}
\]

\[
\text{parent of } p = \text{enclose}(p)
\]

\[
\text{degree}
\]

\[
\text{\( i \)-th child}
\]
Advantages and Disadvantages of Both Approaches

- LOUDS cannot answer subtree size
- BP cannot easily answer $i$-th child and degree
- All other operations can be done easily
Definition: DFUDS

Starting at the root, traverse tree in depth-first order and append
- for node $v$, $\delta(v)$ left parentheses and
- a right parenthesis if $v$ is visited the first time
to the bit vector that initially contains a left parenthesis to make them balanced.

Lemma: Space Usage of DFUDS

Representing a tree with $n$ nodes requires $2n$ bits using DFUDS.

- write down the DFUDS representation of this example tree.
Depth First Unary Degree Sequence (2/2)

- node starts at first parenthesis
- subtree structure is encoded

```
a  bc  de  fghi  jk  ((((()))))((()))(()))((()))((()))((((()))))
```

Diagram:

```
a
  b
  c  d
  e  f  g
  h  j  k
  i
```

Karlruhe Institute of Technology
Making DFUDS Fully-Functional

- degree of $p$: $\text{select}(\text{rank}(p) + 1) - p$
- $i$-th child of $p$: $\text{findclose}(\text{select}(\text{rank}(p) + 1) - i) + 1$
- parent of $p$: $\text{select}(\text{rank}(\text{findopen}(p - 1))) + 1$
- subtree size of $p$: $(\text{findclose}(\text{enclose}(p)) - p)/2 + 1$

- explanation on the board 📚
Conclusion and Outlook

This Lecture
- three succinct tree representations
- different advantages and disadvantages
- min-max-trees

Next Lecture
- succinct graphs

Advanced Data Structures
- BV
- succinct trees
Bibliography I


