Suppose that $Z$ is the information-theoretical optimal number of bits needed to store some data. A representation of this data is called

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


In computer science, a succinct data structure is a data structure which uses an amount of space that is "close" to the information-theoretic lower bound, but (unlike other compressed representations) still allows for efficient query operations. The concept was originally introduced by Jacobson to encode bit vectors, unlabeled trees, and planar graphs. Unlike general lossless data compression algorithms, succinct data structures retain the ability to use them in-place, without decompressing them first. A related notion is that of a compressed data structure, in which the size of the data structure depends upon the particular data being represented.

In computer science, a succinct data structure is a data structure which uses an amount of space that is "close" to the information-theoretic lower bound, but (unlike other compressed representations) still allows for efficient query operations. The concept was originally introduced by Jacobson to encode bit vectors, unlabeled trees, and planar graphs. Unlike general lossless data compression algorithms, succinct data structures retain the ability to use them in-place, without decompressing them first. A related notion is that of a compressed data structure, in which the size of the data structure depends upon the particular data being represented.

Succinct Representations of Trees

S. Srinivasa Rao

Seoul National University

Outline

- Succinct data structures
  - Introduction
  - Examples
- Tree representations
  - Motivation
  - Heap-like representation
  - Jacobson’s representation
  - Parenthesis representation
  - Partitioning method
  - Comparison and Applications
- Rank and Select on bit vectors

Succinct data structures

- Goal: represent the data in close to optimal space, while supporting the operations efficiently.
  (optimal -- information-theoretic lower bound)
  Introduced by [Jacobson, FOCS ’89]
- An "extension" of data compression.
  (Data compression:
  - Achieve close to optimal space
  - Queries need not be supported efficiently)
Applications

- Potential applications where
  - memory is limited: small memory devices like PDAs, mobile phones etc.
  - massive amounts of data: DNA sequences, geographical/astronomical data, search engines etc.

Examples

- Trees, Graphs
- Bit vectors, Sets
- Dynamic arrays
- Text indexes
  - suffix trees/suffix arrays etc.
- Permutations, Functions
- XML documents, File systems (labeled, multi-labeled trees)
- DAGs and BDDs
  - ...

Memory model

- Word RAM model with word size $\Theta(\log n)$ supporting
  - read/write
  - addition, subtraction, multiplication, division
  - left/right shifts
  - AND, OR, XOR, NOT
  - operations on words in constant time.

(Succinct Tree Representations

Motivation

Trees are used to represent:

- Directories (Unix, all the rest)
- Search trees (B-trees, binary search trees, digital trees or tries)
- Graph structures (we do a tree based search)
- Search indexes for text (including DNA)
  - Suffix trees
  - XML documents
  - ...

Drawbacks of standard representations

- Standard representations of trees support very few operations. To support other useful queries, they require a large amount of extra space.

- In various applications, one would like to support operations like "subtree size" of a node, "least common ancestor" of two nodes, "height", "depth" of a node, "ancestor" of a node at a given level etc.
Drawbacks of standard representations

- The space used by the tree structure could be the dominating factor in some applications.
  
  - Eg. More than half of the space used by a standard suffix tree representation is used to store the tree structure.

- "A pointer-based implementation of a suffix tree requires more than 20n bytes. A more sophisticated solution uses at least 12n bytes in the worst case, and about 8n bytes in the average. For example, a suffix tree built upon 700MB of DNA sequences may take 40GB of space."
  
  -- Handbook of Computational Molecular Biology, 2006

Can we improve the space bound?

- There are less than $2^n$ distinct binary trees on $n$ nodes.
  
  - "The Art of Computer Programming", Volume 4, Fascicle 4: Generating all trees
    
    - [Link](http://www.cs.fau.de/~uno/fasc4a.ps.gz)

- $2n$ bits are enough to distinguish between any two different binary trees.

- Can we represent an $n$ node binary tree using $2n$ bits?

How Many Binary Trees Are There?

There are five distinct shapes of binary trees with three nodes:

But how many are there for $n$ nodes?

Let $C_n$ be the number of distinct binary trees with $n$ nodes. This is equal to the number of trees that have a root, a left subtree with $i$ nodes, and a right subtree of $(n-1-i)$ nodes, for each $i$. That is,

$$C_n = C_i C_{n-1-i} + C_{i-1} C_{n-i} + \cdots + C_{n-2} C_1 + C_{n-1} C_0$$

which is

$$C_0 = 1 \quad \text{and} \quad C_n = \sum_{i=0}^{n-1} C_i C_{n-1-i} \quad \text{for} \quad n \geq 1.$$

http://cs.lmu.edu/~ray/notes/binarytrees/

Heap-like notation for a binary tree

Add external nodes

Label internal nodes with a 1 and external nodes with a 0.

Write the labels in level order

One can reconstruct the tree from this sequence

An $n$ node binary tree can be represented in $2n+1$ bits.

What about the operations?
Heap-like notation for a binary tree

left child(x) = \[2x\]
right child(x) = \[2x+1\]
parent(x) = \[\lfloor x/2 \rfloor\]

Example 2 (JV)

Node=
BitVector=
Bvrank=
Rchild( ) = Rank \( \lfloor 2^4 + 1 \rfloor = 6 \)

Succinct tree representations

\[1010101110000\]
Rank/Select on a bit vector

Given a bit vector $B$

$$\text{rank}(i) = \# 1's \text{ up to position } i \text{ in } B$$

$$\text{select}(i) = \text{position of the } i\text{-th }1 \text{ in } B$$

(similarly $\text{rank}_0$ and $\text{select}_0$)

---

Given a bit vector of length $n$, by storing an additional $o(n)$-bit structure, we can support all four operations in $O(1)$ time.

An important substructure in most succinct data structures.

Implementations: [Kim et al.], [Gonzalez et al.], ...

---

Binary tree representation

- A binary tree on $n$ nodes can be represented using $2n+o(n)$ bits to support:
  - $\text{parent}$
  - $\text{left child}$
  - $\text{right child}$

  **in constant time.**

  [Jacobson '89]

---

Supporting Rank

- Store the rank up to the beginning of each block: $(m/b) \log m$ bits
- Store the 'rank within the block' up to the beginning of each sub-block: $(m/b)(b/s) \log b$ bits
- Store a pre-computed table to find the rank within each sub-block: $2^s \log s$ bits

---

Rank/Select on bit vector

- Choosing $b = (\log m)^2$, and $s = (1/2)\log n$ makes the overall space to be $O(m \log \log m / \log m)$ ($= o(m)$) bits.
- Supports rank in constant time.
- Select can also be supported in constant time using an auxiliary structure of size $O(m \log \log m / \log m)$ bits.

  [Clark-Munro '96] [Raman et al. '01]

  http://alexbowe.com/rrr/ -> RRR succinct rank/select index

---

Lower bounds for rank and select

- If the bit vector is read-only, any index (auxiliary structure) that supports rank or select in constant time (in fact in $O(\log m)$ bit probes) has size $\Omega(m \log \log m / \log m)$

  [Miltersen '05] [Golynski '06]

---

Space measures

- Bit-vector (BV):
  - space used be $m + o(m)$ bits.

- Bit-vector index:
  - bit-sequence stored in read-only memory
  - index of $o(m)$ bits to assist operations

- Compressed bit-vector: with $n$ 1's
  - space used should be $B(m,n) + o(m)$ bits.

  $$B(m,n) = \left\lfloor \log \left( \begin{array}{c} m \\ n \end{array} \right) \right\rfloor$$
Results on Bitvectors
- Elias (JACM 74)
- Jacobson (FOCS 89)
- Clark+Munro (SODA 96)
- Pagh (SICOMP 01)
- Raman et al (SODA 02)
- Miltersen (SODA 04)
- Golynski (ICALP 06)
- Gupta et al.

Implementations:
- Geary et al. (TCS 06)
- Kim et al. (WEA 05)
- Delpratt et al. (WEA 06, SOFSEM 07)
- Okanohara+Sadakane (ALENEX 07)

(Roman et al (SODA 02))

Ordered trees?
- A rooted ordered tree (on n nodes):

Navigational operations:
- parent(x) = a
- first child(x) = b
- next sibling(x) = c

Other useful operations:
- degree(x) = 2
- subtree size(x) = 4

Ordered trees
- A binary tree representation taking 2n+o(n) bits that supports parent, left child and right child operations in constant time.
- There is a one-to-one correspondence between binary trees (on n nodes) and rooted ordered trees (on n+1 nodes).
- Gives an ordered tree representation taking 2n+o(n) bits that supports first child, next sibling (but not parent) operations in constant time.
- We will now consider ordered tree representations that support more operations.

Level-order degree sequence
Write the degree sequence in level order
3 2 0 3 0 1 0 2 0 0 0 0
But, this still requires n log n bits
Solution: write them in unary
1 1 1 0 1 1 0 1 1 1 0 0 1 0 0 1 0 0 1 1 0 0 0 0
Takes 2n-1 bits

Level-order unary degree sequence
- Space: 2n+o(n) bits
- Supports
  - parent
  - ith child (and hence first child)
  - next sibling
  - degree
  - in constant time.

Does not support subtree size operation.
[Jacobson '89]
[Implementation: Delpratt-Rahman-Raman '06]
Another approach

Write the degree sequence in depth-first order

3 2 0 1 0 0 3 0 2 0 0 0

In unary:
1 1 1 0 1 1 0 0 1 0 0 0 1 1 0 0 1 1 0 0 0 0

Takes 2n-1 bits.

The representation of a subtree is together.

Supports subtree size along with other operations.
(Apart from rank/select, we need some additional operations.) Which?

Depth-first unary degree sequence (DFUDS)

- Space: $2n+o(n)$ bits
- Supports
  - parent
  - i-th child (and hence first child)
  - next sibling
  - degree
  - subtree size
    - in constant time.

[benoit et al. '05] [jansson et al. '07]

Other useful operations

XML based applications:

level ancestor(x, l); returns the ancestor of x at level l

eg. level ancestor(11, 2) = 4

Suffix tree based applications:

LCA(x, y); returns the least common ancestor of x and y

eg. LCA(7, 12) = 4

Parenthesis representation

Associate an open-close parenthesis-pair with each node

Visit the nodes in pre-order, writing the parentheses

length: $2n$
space: $2n$ bits

One can reconstruct the tree from this sequence

Parenthesis representation

- Space: $2n+o(n)$ bits
- Supports:
  - parent
  - first child
  - next sibling
  - subtree size
  - degree
  - depth
  - height
  - level ancestor
  - LCA
  - leftmost/rightmost leaf
  - number of leaves in the subtree
  - next node in the level
  - pre/post order number
  - i-th child

in constant time.

[Munro-Raman '97] [Munro et al. '01] [sadakane '03] [Lu-Yeh '08]
[Implementation: Geary et al., CPM-04]
**A different approach**

- If we group $k$ nodes into a block, then pointers with the block can be stored using only $\lg k$ bits.

- For example, if we can partition the tree into $n/k$ blocks, each of size $k$, then we can store it using $(n/k) \cdot \lg n + (n/k) \lg k + (n/k) \lg n + n \lg k$ bits.

A careful two-level 'tree covering' method achieves a space bound of $2n+o(n)$ bits.

**Tree covering method**

- **Space:** $2n+o(n)$ bits

- **Supports:**
  - parent
  - first child
  - next sibling
  - subtree size
  - degree
  - depth
  - height
  - level ancestor
  - LCA
  - leftmost/rightmost leaf
  - number of leaves in the subtree
  - next node in the level

  in constant time.

[Geary et al. ’04] [He et al. ’07] [Farzan-Munro ’08]

**Ordered tree representations**

<table>
<thead>
<tr>
<th>LOUDS</th>
<th>X</th>
<th>X</th>
<th>X</th>
<th>X</th>
<th>X</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFUDS</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PAREN</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PARTITION</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Unified representation**

- A single representation that can *emulate* all other representations.

- Result: A $2n+o(n)$ bit representation that can *generate* an arbitrary word ($O(\log n)$ bits) of DFUDS, PAREN or PARTITION in constant time

- Supports the union of all the operations supported by each of these three representations.

[Farzan et al. ’09]

**Applications**

- Representing
  - suffix trees
  - XML documents (supporting XPath queries)
  - file systems (searching and Path queries)
  - representing BDDs
  - ...

**Open problems**

- Making the structures dynamic (there are some existing results)

- Labeled trees (two different approaches supporting different sets of operations)

- Other memory models
  - External memory model (a few recent results)
  - Flash memory model
  - (So far mostly RAM model)
References

- Jacobson, FOCS 89
- Munro-Raman-Rao, FSTTCS 98 (JAlg 01)
- Benoit et al., WADS 99 (Algorithmica 05)
- Lu et al., SODA 01
- Sadakane, ISSAC 01
- Geary-Raman-Raman, SODA 04
- Munro-Rao, ICALP 04
- Jansson-Sadakane, SODA 06

Implementation:
- Geary et al., CPM 04
- Kim et al., WEA 05
- Gonzalez et al., WEA 05
- Delpratt-Rahman-Raman, WAE 06

Thank You