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.


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
- ...

Example: Text Indexing

A text string $T$ of length $n$ over an alphabet $\Sigma$ can be represented using $n \log |\Sigma| + o(n \log |\Sigma|)$ bits, (or the even the $k$-th order entropy of $T$) to support the following pattern matching queries (given a pattern $P$ of length $m$):
- count the # occurrences of $P$ in $T$,
- report all the occurrences of $P$ in $T$,
- output a substring of $T$ of given length in almost optimal time.

Example: Compressed Suffix Trees

- Given a text string $T$ of length $n$ over an alphabet $\Sigma$, one store it using $O(n \log |\Sigma|)$ bits, to support all the operations supported by a standard suffix tree such as pattern matching queries, suffix links, string depths, lowest common ancestors etc. with slight slowdown.

- Note that standard suffix trees use $O(n \log n)$ bits.

Example: Permutations

A permutation $\pi$ of $1,...,n$

<table>
<thead>
<tr>
<th>$i$</th>
<th>$\pi(i)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
</tr>
</tbody>
</table>

A simple representation: $\pi: \{1,2,3,4,5,6,7,8\}$

- $\pi(i)$ in $O(1)$ time
- $\pi^{-1}(i)$ in $O(n)$ time

Succinct representation: $\pi(1)=6$ $\pi^{-1}(1)=5$

- $(1+\varepsilon) \cdot n \lg n$ bits
  - $\pi(i)$ in $O(1)$ time
  - $\pi^{-1}(i)$ in $O(1/\varepsilon)$ time (‘optimal’ trade-off)
  - $\pi^k(i)$ in $O(1/\varepsilon)$ time (for any positive or negative integer $k$)
- $\lg (n^k) + o(n) (< n \lg n)$ bits (optimal space)
- $\pi^k(i)$ in $O(\log n / \log \log n)$ time

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.

(n is the "problem size")

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
- $2n$ bits are enough to distinguish between any two different binary trees.
- Can we represent an $n$ node binary tree using $2n$ bits?

Standard representation

Binary tree: each node has two pointers to its left and right children

An $n$-node tree takes $2n$ pointers or $2n \lg n$ bits (can be easily reduced to $n \lg n + O(n)$ bits).

Supports finding left child or right child of a node (in constant time).

For each extra operation (eg. parent, subtree size) we have to pay, roughly, an additional $n \lg n$ 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,

$$C(n) = C(n-1)C(0) + C(n-2)C(1) + \cdots + C(0)C(n-1)$$

which is

$$C_1 = 1 \quad \text{ and } \quad C_n = \sum_{i=0}^{n-1} C_i C_{n-i-1} \quad \text{for } n \geq 1$$

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?

**Example 2 (JV)**

Node =

BitVector =

Bvrank =

Rchild(x) = Rank $\lfloor 2x + 1 \rfloor$

**Example 2 (JV)**

Node =

BitVector =

Bvrank =

Parent(5) = 8 => 4th node

4th node is at index 7
Rank/Select on a bit vector

Given a bit vector $B$

$\text{rank}_1(i) = \# 1's$ up to position $i$ in $B$

$\text{select}_1(i) = \text{position of the } i\text{-th } 1 \text{ in } B$

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

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
B: 0 1 1 0 1 0 0 0 1 1 0 1 1 1 1

$\text{rank}_1(5) = 3$

$\text{select}_1(4) = 9$

$\text{rank}_0(5) = 2$

$\text{select}_0(4) = 7$

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

Given a bit vector $B$

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

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

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

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
B: 0 1 1 0 1 0 0 0 1 1 0 1 1 1 1

$\text{rank}_b(5) = 3$

$\text{select}_b(4) = 9$

$\text{rank}_s(5) = 2$

$\text{select}_s(4) = 7$

Binary tree representation

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

  in constant time.

[Jacobson '89]

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]
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.

  $B(m,n) = \left\lfloor \log_{\frac{m}{n}} (m) \right\rfloor$

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)

(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 0 0 1 0 0 1 1 0 0 0 0 0
```

Takes $2n-1$ bits

A tree is uniquely determined by its degree sequence
Supporting operations

Add a dummy root so that each node has a corresponding 1

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

node i corresponds to the i-th 1 in the bit sequence

support: parent, i-th child, degree
(using rank and select)

Level-order unary degree sequence

- Space: $2n+o(n)$ bits
- Supports
  - parent
  - i-th 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 1 0 0 1 1 1 1 1 1 1 0 0 0
```

Takes $2n-1$ bits.

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

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

```
( ( ( ( ) ) ) ) ( ( ( ) ) ( ( ) ) )
```
Operations

parent – enclosing parenthesis
first child – next parenthesis (if ‘open’)
next sibling – open parenthesis following the matching closing parenthesis (if exists)
subtree size – half the number of parentheses between the pair

with o(n) extra bits, all these can be supported in constant time.

Parenthesis representation

- Space: 2n+o(n) bits
- Supports:
  - parent
  - first child
  - next sibling
  - subtree size
  - degree
  - depth
  - height
  - 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 \( (n/k) \lg k \) 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
  - i-th child

in constant time.

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

Ordered tree representations

<table>
<thead>
<tr>
<th></th>
<th>LOUDS</th>
<th>DFUDS</th>
<th>PAREN</th>
<th>PARTITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>parent</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>first</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>sibling</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>depth</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>height</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>degree</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</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)

I/O Model [AV88]
- Parameters
  - N: Elements in structure
  - B: Elements per block
  - M: Elements in main memory

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