Adaptive Searching in Succinctly Encoded Binary Relations and Tree-Structured Documents

J. Barbay[1] A. Golynski[1] **J. Ian Munro**[1] S. S. Rao[2]

[1]School of Computer Science
University of Waterloo, Canada.

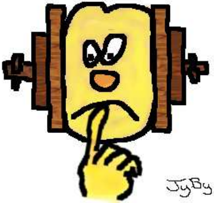
[2]Dept. of Theoretical Computer Science
IT University of Copenhagen, Denmark.

December 2005

Outline

Global Pointers = Evil

- Introduced mainly for trees [Jacobson, 1989].
- Applied to Strings:
  - binary [Clark and Munro, 1996].
  - larger alphabet [Golynski et al., 2006].
- Applied to Trees:
  - cardinal [Benoit et al., 1999].
  - ordinal [Munro and Raman, 2001].
  - partitioned [Geary et al., 2004].
  - labeled [Ferragina et al., 2005].



Strings

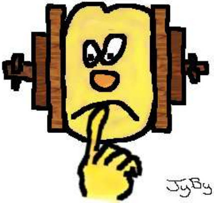
String Succinct Encodings support

- `string_rank`(α, x): nb. of α-occurrences before pos. x;
- `string_select`(α, r): position of r-th α-occurrence.

Example:

0	0	0	1	0	0	0	1	0	0

- `string_rank`$(1, 6) =$
- `string_select`$(1, 2) =$



Strings

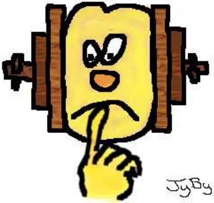
String Succinct Encodings support

- string_rank(α, x): nb. of α-occurrences before pos. x;
- string_select(α, r): position of r-th α-occurrence.

Example:

0	0	0	1	0	0	0	1	0	0

- string_rank$(1, 6) = 1$
- string_select$(1, 2) =$



Strings

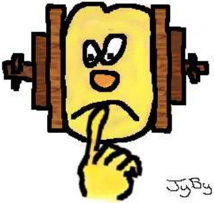
String Succinct Encodings support

- string_rank(α, x): nb. of α-occurrences before pos. x;
- string_select(α, r): position of r-th α-occurrence.

Example:

0	0	0	1	0	0	0	1	0	0

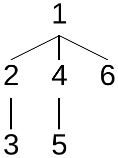
- string_rank$(1, 6) = 1$
- string_select$(1, 2) = 8$

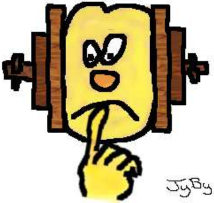


(Unlabeled) Trees

Tree Succinct Encodings support

- navigation operators: `child`(x, r), `depth`(x), `leveled_ancestor`(x, i);
- ranking operators: `tree_rank`(x), `tree_select`(r);
- other useful ones: `isanc`(x, y), `childrank`(x), `degree`(x), `nbdesc`(x).

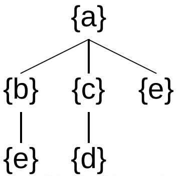


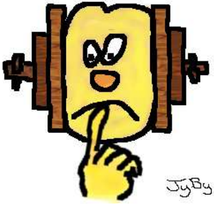


Labeled Trees

Labeled Tree Succinct Encodings support

- `labeltree_anc`(α, x):
  first α-ancestor of x;
- `labeltree_desc`(α, x):
  first α-descendant of x;
- `labeltree_child`(α, x):
  first α-child of x.





Outline

1. Introduction
   - Succinct Data Structures
   - Adaptive Algorithms
2. Our Results
   - Binary Relations
   - Intersection Algorithm
   - (Multi-)Labeled Trees
   - Path Query Algorithm
3. Conclusion

Input Size $\neq$ Diffi culty

- First as *output dependent* analysis for Convex Hull [Kirkpatrick and Seidel, 1986].
- Extensively applied to Sorting [Estivill-Castro and Wood, 1992].
- Applied to Union, Difference, and Intersection [Demaine et al., 2000, Barbay and Kenyon, 2002, Barbay, 2003.]

Algorithm for Intersection

The algorithm from [Barbay and Kenyon, 2002]:

1. **If** $x = \infty$, exit;
2. **If** k labels match, output x, pick next α-object, go to 1;
   Else pick next set α;
3. **If** x matches α, go to 2;
   Else pick next α-object, go to 1.

1	2	3		5	6	7	8	
		3	4					9
1	2	3	4					9

$R = \{$

Intersection of k sets computed in $O(\delta k)$ searches.

Algorithm for Intersection

The algorithm from [Barbay and Kenyon, 2002]:

1. **If** $x = \infty$, exit;
2. **If** k labels match, output x, pick next α-object, go to 1; **Else** pick next set α;
3. **If** x matches α, go to 2; **Else** pick next α-object, go to 1.

1	2	3		5	6	7	8	
		3	4					9
1	2	3	4					9

$R = \{$

Intersection of k sets computed in $O(\delta k)$ searches.

Algorithm for Intersection

The algorithm from [Barbay and Kenyon, 2002]:

1. **If** $x = \infty$, exit;
2. **If** k labels match, output x, pick next α-object, go to 1;
   Else pick next set α;
3. **If** x matches α, go to 2;
   Else pick next α-object, go to 1.

1	2	3		5	6	7	8	
		3	4					9
1	2	3	4					9

$R = \{$

Intersection of k sets computed in $O(\delta k)$ searches.

Algorithm for Intersection

The algorithm from [Barbay and Kenyon, 2002]:

1. **If** $x = \infty$, exit;
2. **If** k labels match, output x, pick next α-object, go to 1; **Else** pick next set α;
3. **If** x matches α, go to 2; **Else** pick next α-object, go to 1.

1	2	3		5	6	7	8	
		3	4					9
1	2	3	4					9

$R = \{$

Intersection of k sets computed in $O(\delta k)$ searches.

Algorithm for Intersection

The algorithm from [Barbay and Kenyon, 2002]:

1. **If** $x = \infty$, exit;
2. **If** k labels match, output x, pick next α-object, go to 1; **Else** pick next set α;
3. **If** x matches α, go to 2; **Else** pick next α-object, go to 1.

1	2	3		5	6	7	8	
		3	4					9
1	2	3	4					9

$R = \{3$

Intersection of k sets computed in $O(\delta k)$ searches.

Algorithm for Intersection

The algorithm from [Barbay and Kenyon, 2002]:

1. **If** $x = \infty$, exit;
2. **If** k labels match, output x,
   pick next α-object, go to 1;
   Else pick next set α;
3. **If** x matches α, go to 2;
   Else pick next α-object, go to 1.

1	2	3		5	6	7	8	
		3	4					9
1	2	3	4					9

$R = \{3$

Intersection of k sets computed in $O(\delta k)$ searches.

Algorithm for Intersection

The algorithm from [Barbay and Kenyon, 2002]:

1. **If** $x = \infty$, exit;
2. **If** k labels match, output x, pick next α-object, go to 1;
   Else pick next set α;
3. **If** x matches α, go to 2;
   Else pick next α-object, go to 1.

1	2	3		5	6	7	8	
		3	4					9
1	2	3	4					9

$R = \{3$

Intersection of k sets computed in $O(\delta k)$ searches.

Algorithm for Intersection

The algorithm from [Barbay and Kenyon, 2002]:

1. **If** $x = \infty$, exit;
2. **If** k labels match, output x,
   pick next α-object, go to 1;
   Else pick next set α;
3. **If** x matches α, go to 2;
   Else pick next α-object, go to 1.

1	2	3		5	6	7	8	
		3	4					9
1	2	3	4					9

$R = \{3$

Intersection of k sets computed in $O(\delta k)$ searches.

Algorithm for Intersection

The algorithm from [Barbay and Kenyon, 2002]:

1. **If** $x = \infty$, exit;
2. **If** k labels match, output x, pick next α-object, go to 1; **Else** pick next set α;
3. **If** x matches α, go to 2; **Else** pick next α-object, go to 1.

1	2	3		5	6	7	8	
		3	4					9
1	2	3	4					9

$R = \{3\}$

Intersection of k sets computed in $O(\delta k)$ searches.

Outline

What is a Binary Relation?

Consider a binary relation defined by:

- n objects (the references to web-pages),
- σ labels (the keywords),
- t pairs from $[n] \times [\sigma]$ (the index).

$$\sigma \left\{ \underbrace{\begin{array}{ccccc} 1 & 0 & \dots & 0 & 1 \\ 1 & & & & 0 \\ \vdots & & (t \text{ ones}) & & \vdots \\ 0 & & & & 1 \\ 0 & 1 & \dots & 1 & 0 \end{array}}_{n} \right.$$

String Representation

We encode it as

- one string *ROWS* on alphabet $[\sigma]$ of length t;
- one binary string *NEWCOLUMN* of length $n + t$.

For instance:

$$\begin{matrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{matrix} \Rightarrow$$

$ROWS = 1, 3, 2, 3, 1, 2, 3, 2$

$NEWCOLUMN = 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1$

This uses $(t \lg \sigma + n + t)$ bits.

String Representation

We encode it as

- one string *ROWS* on alphabet $[\sigma]$ of length t;
- one binary string *NEWCOLUMN* of length $n + t$.

For instance:

$$\begin{matrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{matrix} \Rightarrow$$

$ROWS = 1, 3, 2, 3, 1, 2, 3, 2$
$NEWCOLUMN = 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1$

This uses $(t \lg \sigma + n + t)$ bits.

String Representation

We encode it as

- one string *ROWS* on alphabet $[\sigma]$ of length t;
- one binary string *NEWCOLUMN* of length $n + t$.

For instance:

$$\begin{array}{cccc} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{array} \Rightarrow$$

$ROWS = 1, 3, 2, 3, 1, 2, 3, 2$

$NEWCOLUMN = 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1$

This uses $(t \lg \sigma + n + t)$ bits.

String Representation

We encode it as

- one string *ROWS* on alphabet $[\sigma]$ of length t;
- one binary string *NEWCOLUMN* of length $n+t$.

For instance:

$$\begin{array}{cccc} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{array} \Rightarrow$$

$ROWS = 1,3,2,3,1,2,3,2$

$NEWCOLUMN = 0,0,1,0,0,1,0,0,0,1,0,0,1$

This uses $(t \lg \sigma + n + t)$ bits.

Operators on Binary Relations:

We propose two distinct encodings using $(t \times o(\lg \sigma))$ additional bits, which support

Random Access	$O(\lg\lg\sigma)$	$O(\lg\lg\sigma)$;
Rank on the rows	$O(\lg\lg\sigma)$	$O(\lg\lg\sigma \lg\lg\lg\sigma)$;
Select on the rows	$O(1)$	$O(\lg\lg\sigma)$;
Rank on the columns	$O\left((\lg\lg\sigma)^2\right)$	$O(\lg\lg\sigma)$;
Select on the columns	$O(\lg\lg\sigma)$	$O(1)$.

This is much better than $O(\lg n)$, using posting lists!

Outline

Improved Result on Intersection

When non-deterministic requires δ steps, deterministic requires time

- $O(\delta k \lg(n/\delta k))$ with arrays,
- $O(\delta k \lg \lg \sigma)$ with our encoding.

Improvement factor: $\lg(n/\delta k)/\lg \lg \sigma$.

1	2	3		5	6	7	8
		3	4				
1	2	3	4				

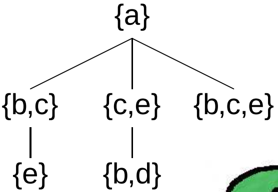
$R = \{3\}$

Outline

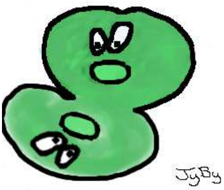
What's a Multi-Labeled Tree?

A *Multi-Labeled Tree* is defined by:

- n nodes,
- σ labels,
- t pairs from $[n] \times [\sigma]$.



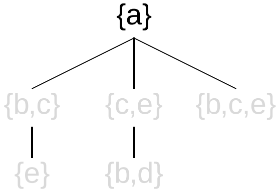
It's like an XML tree, except that several labels can be associated to each node.



Separate Labels and Structure.

We encode it as

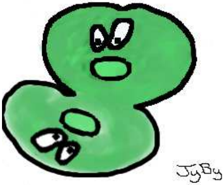
- one string LABELS on alphabet $[\sigma]$;
- one binary string NODES of length t;
- the tree structure in $2n$ bits.



For instance, our tree corresponds to:

$$\begin{aligned} \texttt{LABELS} &= a, b, c, e, c, e, b, d, b, c, e \\ \texttt{NODES} &= 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1 \end{aligned}$$

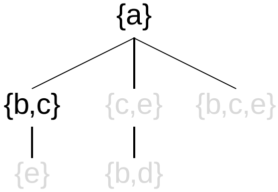
This uses $(t \lg \sigma + t + 2n)$ bits.



Separate Labels and Structure.

We encode it as

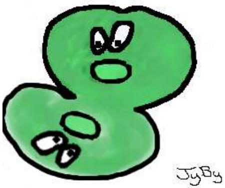
- one string LABELS on alphabet $[\sigma]$;
- one binary string NODES of length t;
- the tree structure in $2n$ bits.



For instance, our tree corresponds to:

$$\begin{aligned} \texttt{LABELS} &= a, b, c, e, c, e, b, d, b, c, e \\ \texttt{NODES} &= 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1 \end{aligned}$$

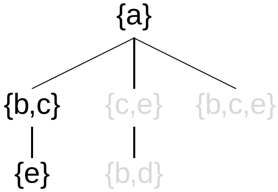
This uses $(t \lg \sigma + t + 2n)$ bits.



Separate Labels and Structure.

We encode it as

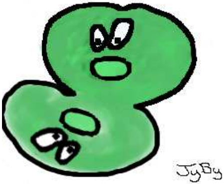
- one string LABELS on alphabet $[\sigma]$;
- one binary string NODES of length t;
- the tree structure in $2n$ bits.



For instance, our tree corresponds to:

$$\begin{aligned} \texttt{LABELS} &= a, b, c, e, c, e, b, d, b, c, e \\ \texttt{NODES} &= 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1 \end{aligned}$$

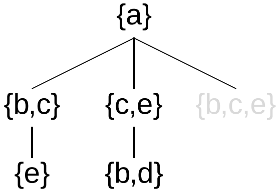
This uses $(t \lg \sigma + t + 2n)$ bits.



Separate Labels and Structure.

We encode it as

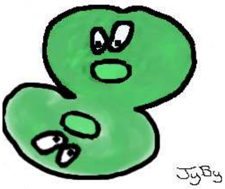
- one string LABELS on alphabet $[\sigma]$;
- one binary string NODES of length t;
- the tree structure in $2n$ bits.



For instance, our tree corresponds to:

$$\begin{aligned} \texttt{LABELS} &= a, b, c, e, c, e, b, d, b, c, e \\ \texttt{NODES} &= 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1 \end{aligned}$$

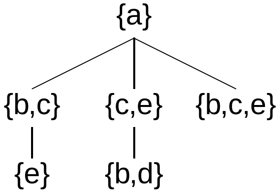
This uses $(t \lg \sigma + t + 2n)$ bits.



Separate Labels and Structure.

We encode it as

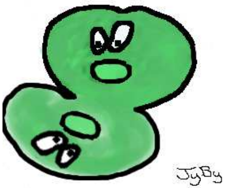
- one string LABELS on alphabet $[\sigma]$;
- one binary string NODES of length t;
- the tree structure in $2n$ bits.



For instance, our tree corresponds to:

$$\begin{aligned} \texttt{LABELS} &= a, b, c, e, c, e, b, d, b, c, e \\ \texttt{NODES} &= 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1 \end{aligned}$$

This uses $(t \lg \sigma + t + 2n)$ bits.

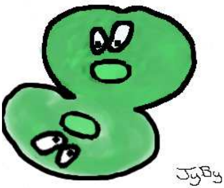


Operators on Multi-Labeled Trees:

We propose two distinct encodings using $(t \times o(\lg \sigma))$ additional bits, which support in time $O(\lg \lg \sigma)$

1. `labeltree_desc` and `labeltree_anc` on non-recursive multi-labeled trees;
2. `labeltree_desc` and `labeltree_child` on any multi-labeled trees.

For simple labeled trees, much better than [Geary], $2n + n(\lg \sigma + O(\sigma \lg \lg \lg n / \lg \lg n))$ ours is $2n + n(\lg \sigma + o(\lg \sigma))$ bits.

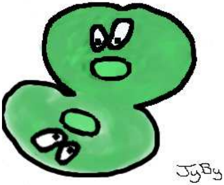


Operators on Multi-Labeled Trees:

We propose two distinct encodings using $(t \times o(\lg \sigma))$ additional bits, which support in time $O(\lg \lg \sigma)$

1. `labeltree_desc` and `labeltree_anc` on non-recursive multi-labeled trees;
2. `labeltree_desc` and `labeltree_child` on any multi-labeled trees.

For simple labeled trees, much better than [Geary],
$2n + n(\lg \sigma + O(\sigma \lg \lg \lg n / \lg \lg n))$
ours is $2n + n(\lg \sigma + o(\lg \sigma))$ bits.

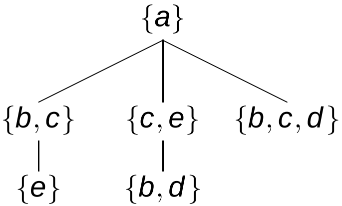


Outline

What is a Path Query?

Given a non-recursive multi-labeled tree and k labels, find nodes x s.t. rooted path matches k labels.

$Q(a, d, e)$

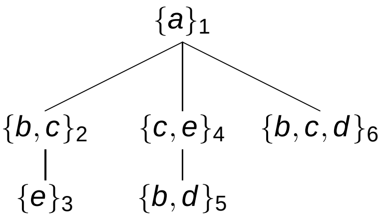


$\Rightarrow$ File System Search.

What is a Path Query?

Given a non-recursive multi-labeled tree and k labels, find nodes x s.t. rooted path matches k labels.

$Q(a, d, e)$

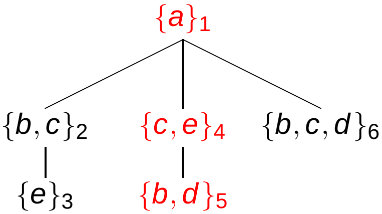


$\Rightarrow$ File System Search.

What is a Path Query?

Given a non-recursive multi-labeled tree and k labels, find nodes x s.t. rooted path matches k labels.

$Q(a, d, e)$

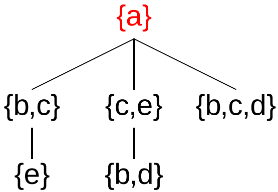


$\Rightarrow$ File System Search.

Our Algorithm:

1. **If** $x = \infty$, exit;
2. **If** all labels match, output x, pick next α-node, go to 1; **Else** pick next label α;
3. **If** x matches α or has a α-ancestor, go to 2;
4. **If** x has a α-descendant, pick the first one, go to 2; **Else** pick next α-node, go to 1.

This algorithm solves Path queries in time $O(\delta k \lg \lg \sigma)$.

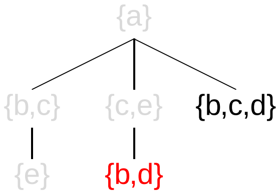


$Q(a, d, e) = \{$

Our Algorithm:

1. **If** $x = \infty$, exit;
2. **If** all labels match, output x,
   pick next α-node, go to 1;
   Else pick next label α;
3. **If** x matches α or has a α-ancestor,
   go to 2;
4. **If** x has a α-descendant,
   pick the first one, go to 2;
   Else pick next α-node, go to 1.

This algorithm solves Path queries in time $O(\delta k \lg\lg \sigma)$.

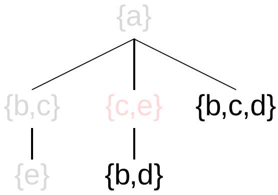


$Q(a, d, e) = \{$

Our Algorithm:

1. **If** $x = \infty$, exit;
2. **If** all labels match, output x,
   pick next α-node, go to 1;
   Else pick next label α;
3. **If** x matches α or has a α-ancestor,
   go to 2;
4. **If** x has a α-descendant,
   pick the first one, go to 2;
   Else pick next α-node, go to 1.

This algorithm solves Path queries in time $O(\delta k \lg\lg \sigma)$.

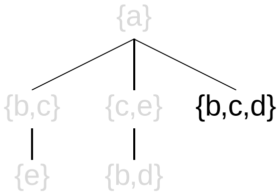


$Q(a, d, e) = \{$

Our Algorithm:

1. **If** $x = \infty$, exit;
2. **If** all labels match, output x,
   pick next α-node, go to 1;
   Else pick next label α;
3. **If** x matches α or has a α-ancestor,
   go to 2;
4. **If** x has a α-descendant,
   pick the first one, go to 2;
   Else pick next α-node, go to 1.

This algorithm solves Path queries
in time $O(\delta k \lg \lg \sigma)$.

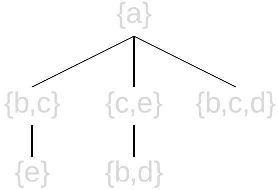


$Q(a, d, e) = \{5$

Our Algorithm:

1. **If** $x = \infty$, exit;
2. **If** all labels match, output x, pick next α-node, go to 1; **Else** pick next label α;
3. **If** x matches α or has a α-ancestor, go to 2;
4. **If** x has a α-descendant, pick the first one, go to 2; **Else** pick next α-node, go to 1.

This algorithm solves Path queries in time $O(\delta k \lg\lg \sigma)$.



$Q(a, d, e) = \{5\}$

Optimality

This is close to optimal:
For any δ, n, k, σ, t, there is a path query such that

1. for any deterministic algorithm there is distribution on non-recursive multi-labeled tree,
2. for any randomized algorithm there is one non-recursive multi-labeled tree,

on which they perform on average $\Omega(\delta k)$ searches.

Outline

1. Introduction
   - Succinct Data Structures
   - Adaptive Algorithms
2. Our Results
   - Binary Relations
   - Intersection Algorithm
   - (Multi-)Labeled Trees
   - Path Query Algorithm
3. Conclusion

Summary

- Succinct encodings improve space and time.
- Labeled Trees use optimal space.
- Adaptive almost as good as Non-Deterministic!

- Future Work
  - Other type of queries on trees.
  - Applications to algorithms on graphs.
  - Support for all labeled-based operators at once on (multi-)labeled trees.

Main References

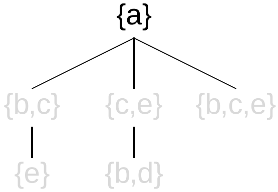
- Barbay, J. and Kenyon, C. (*SODA'02*).
  Adaptive intersection and t-threshold problems.
- Geary, R. F., Raman, R., and Raman, V. (*SODA '04*).
  Succinct ordinal trees with level-ancestor queries.
- Golynski, A., Munro, J. I., and Rao, S. S. (*SODA'06*).
  Rank/select operations on large alphabets:
  a tool for text indexing.

Encoding for Effi cient Child Queries.

We still encode it as

- one string LABELS on alphabet $[\sigma]$;
- one binary string NODES of length t;
- the tree structure in $2n$ bits.

but in a different order.



For instance, the previous tree corresponds to:

$$\texttt{LABELS} = a, b, c, c, e, b, c, e, e, b, d$$
$$\texttt{NODES} = 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1$$

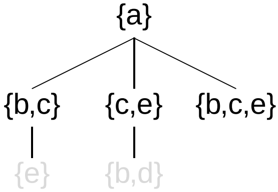
This uses $(t \lg \sigma + t + 2n)$ bits.

Encoding for Efficient Child Queries.

We still encode it as

- one string LABELS on alphabet $[\sigma]$;
- one binary string NODES of length t;
- the tree structure in $2n$ bits.

but in a different order.



For instance, the previous tree corresponds to:

$$\begin{aligned} \texttt{LABELS} &= a, b, c, c, e, b, c, e, e, b, d \\ \texttt{NODES} &= 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1 \end{aligned}$$

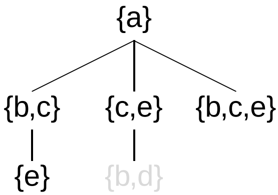
This uses $(t \lg \sigma + t + 2n)$ bits.

Encoding for Effi cient Child Queries.

We still encode it as

- one string LABELS on alphabet $[\sigma]$;
- one binary string NODES of length t;
- the tree structure in $2n$ bits.

but in a different order.



For instance, the previous tree corresponds to:

$$\texttt{LABELS} = a, b, c, c, e, b, c, e, e, b, d$$
$$\texttt{NODES} = 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1$$

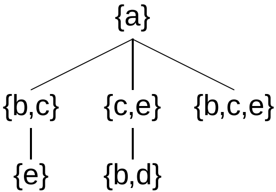
This uses $(t \lg \sigma + t + 2n)$ bits.

Encoding for Effi cient Child Queries.

We still encode it as

- one string LABELS on alphabet $[\sigma]$;
- one binary string NODES of length t;
- the tree structure in $2n$ bits.

but in a different order.



For instance, the previous tree corresponds to:

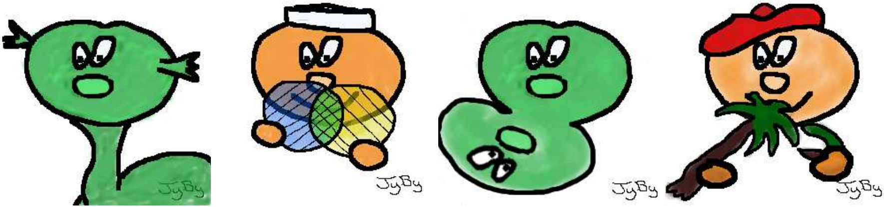
$$\begin{aligned} \text{LABELS} &= a, b, c, c, e, b, c, e, e, b, d \\ \text{NODES} &= 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1 \end{aligned}$$

This uses $(t \lg \sigma + t + 2n)$ bits.

Entropy and Compression

By taking advantage of the frequencies of labels, we can attain the entropy lower bound on a string.

But what is the entropy of an array, of a tree?



XPath Index

Two distinct encodings, supporting

1. labeltree_desc and labeltree_anc
2. labeltree_desc and labeltree_child

Potential extensions are:

- labeltree_anc on recursive multi-labeled trees;
- labeltree_anc and labeltree_child at once (with labeltree_desc).

XPath Index

Two distinct encodings, supporting

1. labeltree_desc and labeltree_anc
2. labeltree_desc and labeltree_child

Potential extensions are:

- labeltree_anc on recursive multi-labeled trees;
- labeltree_anc and labeltree_child at once (with labeltree_desc).

Information Retrieval

There is a (small) catch.

- In real applications, labels are strings.
- For exact match, simply add a (succinct) suffix tree *S*.
- For sub-string match, each query label corresponds to
  - a subtree of the suffi x tree *S*;
  - i.e. an interval in the pre-order traversal of *S*.

Can we extend rank and select to intervals of labels?