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

Jérémy Barbay[1] A. Golynski[1] J. I. Munro[1] S. S. Rao[2]

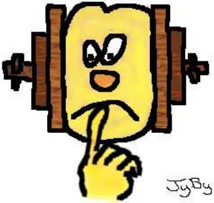
[1]David R. Cheriton
School of Computer Science,
University of Waterloo, Canada.

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

July 2, 2006

Global Pointers = Evil

- Introduced mainly for trees [Jacobson, 1989].
- Applied to Strings:
  - binary [Jacobson, 1989], and [Clark and Munro, 1996].
  - larger alphabet [Grossi et al., 2003; 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].



Example: Strings

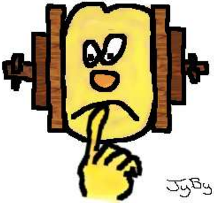
String Succinct Encodings support

- `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`$(1, 2) =$



Example: Strings

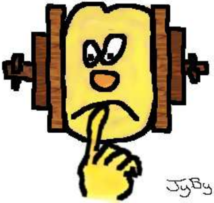
String Succinct Encodings support

- 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) $=$



Example: Strings

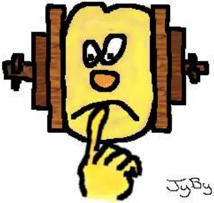
String Succinct Encodings support

- `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

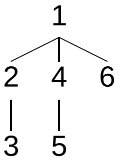
- `string_rank`$(1, 6) = 1$
- `string_select`$(1, 2) = 8$



(Unlabeled) Trees

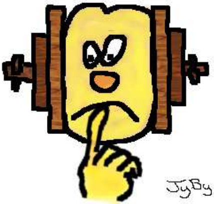
Tree Succinct Encodings support

- 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).



Notation: n nodes.

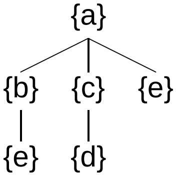
$2n + o(n)$ bits, constant time.



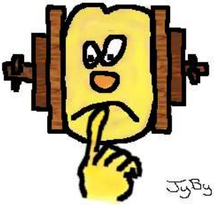
Labeled Trees

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.



Notation: n nodes, σ labels.



Space issues

- Geary *et al.*'s encodings uses

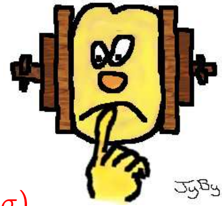
$$n\left(\lg\sigma + \mathcal{O}(\sigma\lg\lg\lg n/\lg\lg n)\right) \text{ bits, for constant time.}$$

- Ferragina *et al.*'s encoding uses

$$2n\lg\sigma + \mathcal{O}(n) \text{ bits, for partial constant time.}$$

- Information theory suggests a lower bound of

$$n\left(2+\lg\sigma\right) \text{ bits.}$$



- Our encoding, in this particular case, uses:

$$n\left(\lg\sigma + o(\lg\sigma)\right) \text{ bits, for time } O(\lg\lg\sigma).$$

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).

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

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:

ROWS	=	1	3		2	3		1	2	3		1	3	
NEWCOLUMN	=	0	0	1	0	0	1	0	0	0	1	0	0	1

represents the binary relation $R = \begin{matrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{matrix}$

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:

$$\begin{array}{lcccccccccccccc} ROWS & = & 1 & 3 & & 2 & 3 & & 1 & 2 & 3 & & 1 & 3 & \\ NEWCOLUMN & = & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \end{array}$$

represents the binary relation $R = \begin{array}{cccc} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{array}$

This uses $\left(t \lg \sigma + n + t\right)$ 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:

$$\begin{array}{lccccccccccccccc} \mathit{ROWS} & = & 1 & 3 & & 2 & 3 & & 1 & 2 & 3 & & 1 & 3 & \\ \mathit{NEWCOLUMN} & = & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \end{array}$$

represents the binary relation $R = \begin{array}{cccc} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{array}$

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:

$$
\begin{array}{lcccccccccccccc}
\mathit{ROWS} & = & 1 & 3 & & 2 & 3 & & 1 & 2 & 3 & & 1 & 3 & \\
\mathit{NEWCOLUMN} & = & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1
\end{array}
$$

represents the binary relation $R = \begin{array}{cccc} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{array}$

This uses $\left(t \lg \sigma + n + t\right)$ 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:

$$\begin{array}{lccccccccccccccc} ROWS & = & 1 & 3 & & 2 & 3 & & 1 & 2 & 3 & & 1 & 3 & \\ NEWCOLUMN & = & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \end{array}$$

represents the binary relation $R = \begin{array}{cccc} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{array}$

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

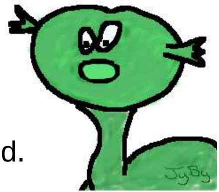
String Operations

`rank`, `select` and `access` on *ROWS* and *NEWCOLUMN*
`rank`, `select` and `access` on the rows of R.

For instance:

$$\begin{array}{lccccccccccccccc} ROWS & = & 1 & 3 & & 2 & 3 & & 1 & 2 & 3 & & 1 & 3 & \\ NEWCOLUMN & = & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \end{array}$$

$$R = \begin{array}{cccc} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{array}$$



`rank` and `select` on columns are more complicated.

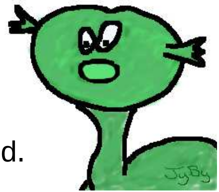
String Operations

`rank`, `select` and `access` on *ROWS* and *NEWCOLUMN*
`rank`, `select` and `access` on the rows of *R*.

For instance:

$$\begin{array}{lcccccccccccccc} ROWS & = & 1 & 3 & & 2 & 3 & & 1 & 2 & 3 & & 1 & 3 \\ NEWCOLUMN & = & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \end{array}$$

$$R = \begin{array}{cccc} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{array}$$



`rank` and `select` on columns are more complicated.

Operators on Binary Relations:

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

Access	$O(\lg \lg \sigma)$	$O(\lg \lg \sigma)$;
`rank` on rows	$O(\lg \lg \sigma)$	$O(\lg \lg \sigma \lg \lg \lg \sigma)$;
`select` on rows	$O(1)$	$O(\lg \lg \sigma)$;
`rank` on columns	$O\left((\lg \lg \sigma)^2\right)$	$O(\lg \lg \sigma)$;
`select` on columns	$O(\lg \lg \sigma)$	$O(1)$.

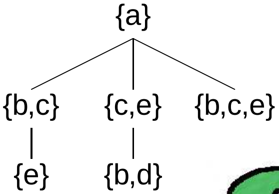
This is much better than $O(\lg n)$, using sorted arrays!

Outline

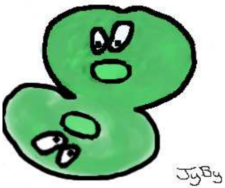
What's a Multi-Labeled Tree?

A Multi-Labeled Tree is defined by:

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



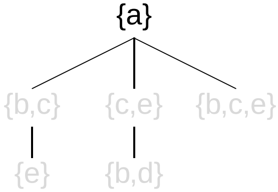
It's like a labeled tree, except that several labels can be associated to each node.



Separate Labels and Structure.

We encode it as

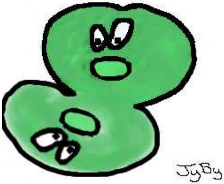
- one string LABELS on alphabet $[\sigma]$;
- one binary string NODES of length t;
- the tree structure in $2n$ bits.



For instance, our tree corresponds to:

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

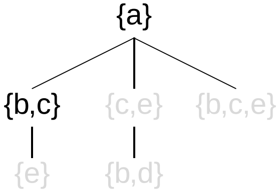
This uses $(t \lg \sigma + t + 2n)$ bits.



Separate Labels and Structure.

We encode it as

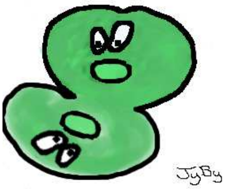
- one string LABELS on alphabet $[\sigma]$;
- one binary string NODES of length t;
- the tree structure in $2n$ bits.



For instance, our tree corresponds to:

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

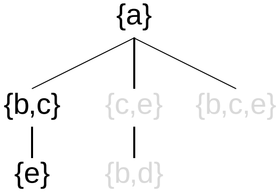
This uses $(t \lg \sigma + t + 2n)$ bits.



Separate Labels and Structure.

We encode it as

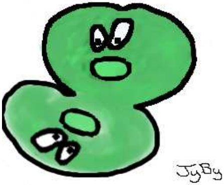
- one string LABELS on alphabet $[\sigma]$;
- one binary string NODES of length t;
- the tree structure in $2n$ bits.



For instance, our tree corresponds to:

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

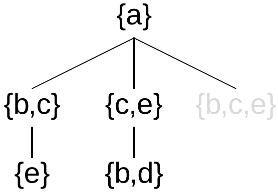
This uses $\left(t \lg \sigma + t + 2n\right)$ bits.



Separate Labels and Structure.

We encode it as

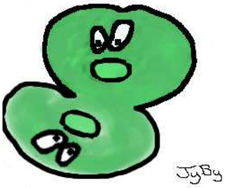
- one string LABELS on alphabet $[\sigma]$;
- one binary string NODES of length t;
- the tree structure in $2n$ bits.



For instance, our tree corresponds to:

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

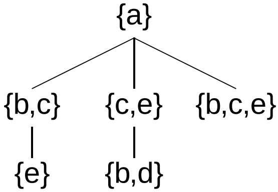
This uses $(t \lg \sigma + t + 2n)$ bits.



Separate Labels and Structure.

We encode it as

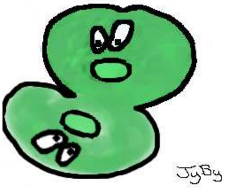
- one string LABELS on alphabet $[\sigma]$;
- one binary string NODES of length t;
- the tree structure in $2n$ bits.



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 $\left(t \lg \sigma + t + 2n\right)$ bits.

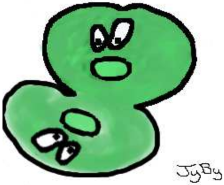


Operators on Multi-Labeled Trees:

We propose an encoding using $(t \times o(\lg \sigma))$ additional bits, which support

- Operator `select` on preorder list and navigation operators in constant time;
- Operators `rank` and `access` on preorder list, and `labeltree_desc` and `labeltree_anc`, in time $O(\lg \lg \sigma)$.

For simple labeled trees, space is much better than [Geary] and [Ferragina].



Outline

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	10	11	12	13
1	2	3	4	9	11	13

$R = \{$

Intersection of k sets computed in $O(\delta k)$ searches, where δ is the non-deterministic complexity.

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	10	11	12	13
1	2	3	4	9	11	13

$R = \{$

Intersection of k sets computed in $O(\delta k)$ searches, where δ is the non-deterministic complexity.

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	10	11	12	13
1	2	3	4	9	11	13

$R = \{$

Intersection of k sets computed in $O(\delta k)$ searches, where δ is the non-deterministic complexity.

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	10	11	12	13
1	2	3	4	9	11	13

$R = \{$

Intersection of k sets computed in $O(\delta k)$ searches, where δ is the non-deterministic complexity.

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	10	11	12	13
1	2	3	4	9	11	13

$R = \{3$

Intersection of k sets computed in $O(\delta k)$ searches, where δ is the non-deterministic complexity.

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	10	11	12	13
1	2	3	4	9	11	13

$R = \{3$

Intersection of k sets computed in $O(\delta k)$ searches, where δ is the non-deterministic complexity.

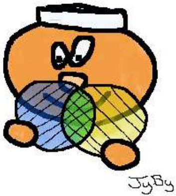
Algorithm for Intersection

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	10	11	12	13
1	2	3	4	9	11	13

$R = \{3$



Intersection of k sets computed in $O(\delta k)$ searches, where δ is the non-deterministic complexity.

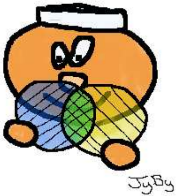
Algorithm for Intersection

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	10	11	12	13
1	2	3	4	9	11	13

$R = \{3$



Intersection of k sets computed in $O(\delta k)$ searches, where δ is the non-deterministic complexity.

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	10	11	12	13
1	2	3	4	9	11	13

$R = \{3\}$

Intersection of k sets computed in $O(\delta k)$ searches,
where δ is the non-deterministic complexity.

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	10	11	12	13
1	2	3	4	9	11	13

$R = \{3\}$

Intersection of k sets computed in $O(\delta k)$ searches, where δ is the non-deterministic complexity.

Time issues

- Using Binary Search, the algorithms performs

$$O(\delta k \lg n) \text{ comparisons}$$

- Using Doubling Search, the algorithms performs

$$O(\delta k \lg(n/\delta k)) \text{ comparisons}$$

- We propose another search operator.

$$O(\delta k \lg \lg \sigma) \text{ comparisons}$$

(Where n is the sum of the sizes of the arrays,
δ the non-deterministic complexity.)

Time issues

- Using Binary Search, the algorithms performs

$$O(\delta k \lg n) \text{ comparisons}$$

- Using Doubling Search, the algorithms performs

$$O(\delta k \lg(n/\delta k)) \text{ comparisons}$$

- We propose another search operator.

$$O(\delta k \lg \lg \sigma) \text{ comparisons}$$

(Where n is the sum of the sizes of the arrays,
δ the non-deterministic complexity.)

Time issues

- Using Binary Search, the algorithms performs

$$O(\delta k \lg n) \text{ comparisons}$$

- Using Doubling Search, the algorithms performs

$$O(\delta k \lg(n/\delta k)) \text{ comparisons}$$

- We propose another search operator.

$$O(\delta k \lg \lg \sigma) \text{ comparisons}$$

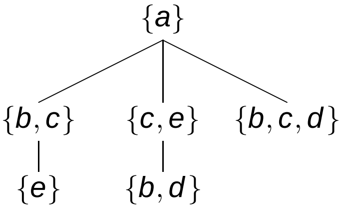
(Where n is the sum of the sizes of the arrays,
δ the non-deterministic complexity.)

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.

$Q(a, d, e)$

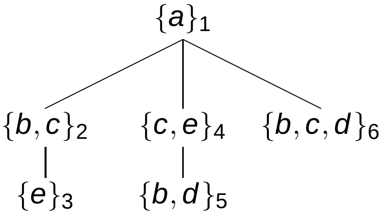


$\Rightarrow$ File System Search.

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.

$Q(a, d, e)$

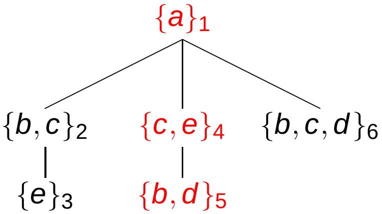


$\Rightarrow$ File System Search.

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.

$Q(a, d, e)$

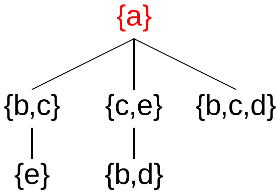


$\Rightarrow$ File System Search.

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 $O(\delta k \lg\lg \sigma)$.

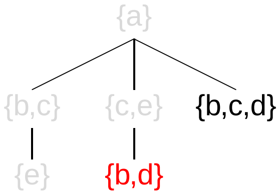


$Q(a, d, e) = \{$

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 $O(\delta k \lg\lg\sigma)$.

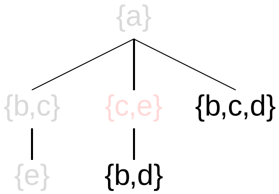


$Q(a, d, e) = \{$

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 $O(\delta k \lg \lg \sigma)$.

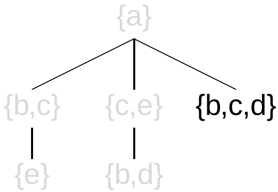


$Q(a, d, e) = \{$

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 $O(\delta k \lg \lg \sigma)$.

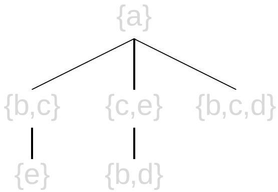


$Q(a, d, e) = \{5$

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 $O(\delta k \lg\lg \sigma)$.



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

Outline

Summary

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

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

Appendix

Main References

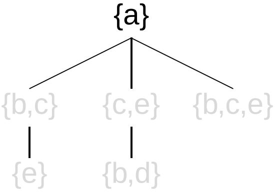
- Barbay, J. and Kenyon, C. (*SODA'02*).
  Adaptive intersection and t-threshold problems.
- Geary, R. F., Raman, R., and Raman, V. (*SODA '04*).
  Succinct ordinal trees with level-ancestor queries.
- Golynski, A., Munro, J. I., and Rao, S. S. (*SODA'06*).
  Rank/select operations on large alphabets: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

Encoding for 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:

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

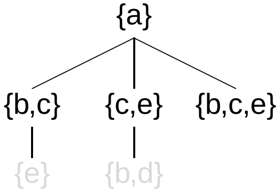
This uses $(t \lg \sigma + t + 2n)$ bits.

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:

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

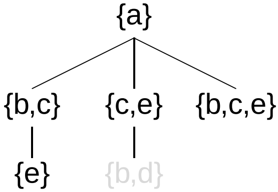
This uses $(t \lg \sigma + t + 2n)$ bits.

Encoding for 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:

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

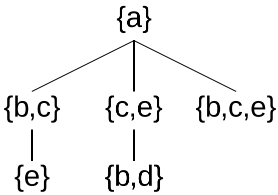
This uses $\left(t \lg \sigma + t + 2n\right)$ bits.

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.

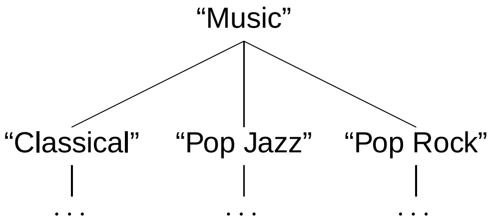


For instance, the previous tree corresponds to:

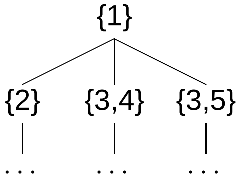
$$\texttt{LABELS} = a, b, c, c, e, b, c, e, e, b, d$$
$$\texttt{NODES} = 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1$$

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

From a File System to a Multi-Labeled Tree

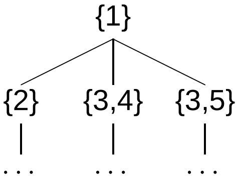


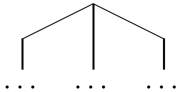
1 Music
2 Classical
3 Pop
4 Jazz
5 Rock



From a Multi-Labeled Tree to its Succinct Encoding

1	Music
2	Classical
3	Pop
4	Jazz
5	Rock





1		2		···	3	4		···	3	5		··
0	1	0	1	···	0	0	1	···	0	0	1	··

Entropy and Compression

- By replacing strings by label numbers, we reduce the space usage.
- 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?

Information Retrieval

Possible Extension:

- In real applications, labels are strings.
- For sub-string match, each query label corresponds to
  - a subtree in the suffix tree S of all lables;
  - i.e. an interval in the pre-order traversal of S.

Can we extend rank and select to intervals of labels?