Words Distinguished by their Subwords

Imre Simon
Universidade de São Paulo
São Paulo, Brasil

<is@ime.usp.br>
http://www.ime.usp.br/~is/

September, 2003 (Words @ Turku)

Words distinguished by their subwords

Word y distinguishes words x_1 and x_2 if y is a subword of exactly one of x_1 and x_2.

y is a subword of x if there exist words $y_1, y_2, \ldots, y_n, x_0, x_1, \ldots, x_n$ in A^*, for some $n \geq 1$, such that

$$y = y_1 y_2 \cdots y_n \quad \text{and} \quad x = x_0 y_1 x_1 y_2 x_2 \cdots y_n x_n.$$

Sometimes y is called "a subsequence" of x.

There exist a word distinguishing x_1 and x_2 iff x_1 and x_2 are distinct.

We are interested in finding a shortest word distinguishing x_1 and x_2 (if there is one).

Result to be presented here: an $O(|A|(|x_1|+|x_2|))$ time complexity algorithm.

Main result: an $O(|A| + |x_1| + |x_2|)$ time complexity algorithm.

Some examples

```
x_1= b c a c a c a d c a b a b a b d a c b a

x_2= b a c c a d c a b b c b c a b a d a b c

y_1= d a c b a c

y_2= d c c c
```

y_1 and y_2 both distinguish x_1 and x_2

y_2 is a shortest word that distinguishes x_1 and x_2:
x_1 and x_2 have the same subwords up to length 3

Some theory

x_1 and x_2 are m-equivalent if
they have the same subwords of length up to m: $x_1 \equiv x_2\ [J_m]$.

Let y be a shortest word distinguishing $x_1 \neq x_2$. We define:

$$\delta(x_1, x_2) = \begin{cases} \infty & \text{if } x_1 = x_2, \\ |y| - 1 & \text{otherwise.} \end{cases}$$

$\delta(x_1, x_2)$ is the greatest m for which $x_1 \equiv x_2\ [J_m]$.

Two one-sided particular cases: $\delta(ua, u)$ and $\delta(av, v)$ lead to
right and left distinguishers for $u, v \in A^*$ and $a \in A$

A fundamental property:

$$\delta(uav, uv) = \delta(ua, u) + \delta(av, v).$$

An illustration of the fundamental property

```
                         *

x_1= b c a c a c a d c a b a b a b d a c b a

     ---------------------------------------
-->  0 0 0 1 1 2 2 0 1 1 1 2 2 3 3 1 2 2 2 3
<--  2 4 4 3 3 2 2 1 1 3 3 2 2 1 1 0 1 0 0 0
     ---------------------------------------

  m= 2 4 4 4 4 4 4 1 2 4 4 4 4 4 4 1 3 2 2 3

                   +     + +   +   +
```

a shortest distinguisher:

$$y = d.b.aad, \quad |y| = 5$$

$$\delta(bcacacadca \overset{b}{\uparrow} ababdacba, bcacacadcaababdacba) = 1 + 3$$

The main theorem

The algorithm is (partially) based on carefully following the proof of the following main theorem:

If words x_1 and x_2 have the same subwords of length at most m, for some m, then x_1 and x_2 can be merged into a new word z, without introducing any new subwords of length up to m

In other words, taking the largest possible m for given x_1 and x_2, we have:

every x_1 and x_2 have a common superword z such that

$$\delta(x_1, x_2) = \min\{\, \delta(x_1, z), \delta(x_2, z) \,\}.$$

Or, still in other words, every x_1 and x_2 can be merged into a word z without introducing any new subwords of length up to $\delta(x_1, x_2)$.

The merging of the example words

```
x_1= b c a c   a c a d c a b         a b a b d a c b   a
     | | | |   | | | | | | |         | | | | | | | |   |
  z= b c a c c a c a d c a b b c b c a b a b d a c b c a
     |   | | | |       | | | | | | | | | | |   | |   | |
x_2= b   a c c a       d c a b b c b c a b a   d a   b c


  y=                   d c       c   c
```

Computing a shortest distinguisher for x_1 and x_2

Distinguisher(x_1, x_2)

$\triangleright$ Preparing the leftist data structures
$\text{Ld}_1 \leftarrow \text{LeftDistinguisher}(x_1, x_2)$
$\text{Ld}_2 \leftarrow \text{LeftDistinguisher}(x_2, x_1)$
$\triangleright$ Merging x_1 and x_2 into z
$(z, \text{Rd}, m, jm, psm, (im, am)) \leftarrow \text{Merge}(x_1, x_2)$
if $m = \infty$ **then** $\triangleright$ $x_1 = x_2$, they can not be distinguished
else $yr \leftarrow \text{Collect}(\text{Rd}, (jm - 1, z[jm]))$
$\quad yl \leftarrow \text{Collect}(\text{Ld}_{psm}, (im, am))$
$\quad y \leftarrow \mathrm{reverse}(yr).z[jm].yl$
$\quad \triangleright$ We just collected a shortest distinguisher in y, $|y| = m + 1$
$\quad |y| \leftarrow m + 1$
return (m, y)

Left distinguisher Ld of x with embeddings in y

```
LeftDistinguisher(x, y)
for a ∈ A do                              ▷ Subword automaton of the reverse of y
        Auto[0, a] ← DeadState
        Auto[DeadState, a] ← DeadState
for i from 1 to |y| do
        for a ∈ A do
                if y[i] = a then
                        Auto[i, a] ← i − 1
                else    Auto[i, a] ← Auto[i − 1, a]
for a ∈ A do                              ▷ Left Distinguisher matrix Ld of x
        Ld[|x|, a] ← (0, nil, |y|)
for i from |x| to 1 do
        for a ∈ A do
                (l, n, s) ← Ld[i, x[i]]
                if a = x[i] or length(Ld[i, a]) > 1 + l then
                        Ld[i − 1, a] ← (1 + l, (i, x[i]), Auto[s, x[i]])
                else    Ld[i − 1, a] ← Ld[i, a]
return Ld
```

Merging $x_1 + x_2 \Rightarrow z$ without new short subwords

Merge(x_1, x_2)

$(k_1, k_2, j, m) \leftarrow (0, 0, 0, \infty)$
for $a \in A$ **do**
    $\mathsf{Rd}[0, a] \leftarrow (0, \mathsf{nil}, \mathsf{nil})$
while $k_1 < |x_1|$ **or** $k_2 < |x_2|$ **do**
    $j \leftarrow j + 1$
    $(\mathrm{Case}, pz, ps, (i', a')) \leftarrow \mathsf{MergeStep}()$
    $(k_{pz}, z[j]) \leftarrow (k_{pz} + 1, x_{pz}[k_{pz}])$
    for $a \in A$ **do**
        $(l, n, s) \leftarrow \mathsf{Rd}[j - 1, z[j]]$
        if $a = z[j]$ **or** $\mathrm{length}(\mathsf{Rd}[j - 1, a]) > 1 + l$ **then**
            $\mathsf{Rd}[j, a] \leftarrow (1 + l, (j - 1, z[j]), \mathsf{nil})$
        else $\mathsf{Rd}[j, a] \leftarrow \mathsf{Rd}[j - 1, a]$
    if $\mathrm{Case} \neq \mathrm{Match}$ **then**
        $m' \leftarrow \mathrm{length}(\mathsf{Rd}[j - 1, z[j]]) + \mathrm{length}(\mathsf{Ld}_{ps}[i', a'])$
        if $m = \infty$ **or** $m' < m$ **then**
            $(m, jm, psm, (im, am)) \leftarrow (m', j, ps, (i', a'))$
$|z| \leftarrow j$
return $(z, \mathsf{Rd}, m, jm, psm, (im, am))$

Auxiliary procedure MergeStep

MergeStep()

if $k_1 < |x_1|$ **and** $k_2 < |x_2|$ **then**
    $(b_1, b_2) \leftarrow (x_1[k_1+1], x_2[k_2+1])$
    if $b_1 = b_2$ **then**
        $(pz, k_2, \text{Case}) \leftarrow (1, k_2+1, \text{Match})$
    else $\text{Case} \leftarrow \text{Mismatch}$
        $(pz, ps, (i', a')) \leftarrow$ Race(k_1, b_2, k_2, b_1)
else $\text{Case} \leftarrow \text{Singleton}$
    if $k_1 = |x_1|$ **then**
        $pz \leftarrow 2$
    else $pz \leftarrow 1$
    ▷ (i', a') is a pointer in Ld$_{ps}$
    $(ps, (i', a')) \leftarrow (pz, (|x_{ps}|, z[j]))$
return $(\text{Case}, pz, ps, (i', a'))$
▷ If $\text{Case} =$ Merge then ps and (i', a') are undefined

Confronting x_1 and x_2 in the case of a mismatch

Race(k_1, b_2, k_2, b_1)

if $\mathrm{length}(\mathsf{Ld}_1[k_1, b_2]) \leq \mathrm{length}(\mathsf{Ld}_2[k_2, b_1])$ **then**

  $(ps, pl) \leftarrow (1, 2)$

else $(ps, pl) \leftarrow (2, 1)$

$\triangleright$ $\mathsf{Ld}_{ps}[k_{ps}, b_{pl}]$ is the left distinguisher which won the race

$(i', a') \leftarrow (k_{ps}, b_{pl})$

if $\mathrm{next}(\mathsf{Ld}_{ps}[k_{ps}, b_{pl}]) =$ nil **or** $\mathrm{suffix}(\mathsf{Ld}_{ps}[k_{ps}, b_{pl}]) \geq 1 + k_{pl}$ **then**

  $pz \leftarrow pl$

else $pz \leftarrow ps$

return $(pz, ps, (i', a'))$ $\triangleright$ (i', a') is a pointer in Ld_{ps}

Collecting the word encoded in a (one sided) distinguisher

Collect(D, (i, a))

$\triangleright$ We collect the word y encoded in D$[i, a]$; with $|y| = \text{length}(\mathsf{D}[i, a])$

$(j, k) \leftarrow (1, \text{length}(\mathsf{D}[i, a]))$

while $k > 0$ **do**

  $(l, (i, a), s) \leftarrow \mathsf{D}[i, a]$ $\triangleright$ We must have $l = k$

  $y[j] \leftarrow a$

  $(j, k) \leftarrow (j + 1, k - 1)$

$|y| \leftarrow j - 1$

return y

The merging of the example words

```
x_1= b c a c   a c a d c a b           a b a b d a c b   a
     | | | |   | | | | | | |           | | | | | | | |   |
  z= b c a c c a c a d c a b b c b c a b a b d a c b c a
     |   | | | |     | | | | | | | | | | |   | |   | |
x_2= b   a c c a     d c a b b c b c a b a   d a   b c


  y=                 d c       c   c
```

Part of the $|A| \times |x_1|$ table of left distinguishers of x_1

A	$\cdots$	11	12	13	14	15
a	$\cdots$	$(3,(12,a),13)$	$=$	$(2,(14,a),15)$	$=$	$(1,(16,d),16)$
b	$\cdots$	$=$	$(3,(13,b),11)$	$=$	$(2,(15,b),14)$	$=$
c	$\cdots$	$=$	$=$	$=$	$=$	$=$
d	$\cdots$	$=$	$=$	$=$	$=$	$(1,(16,d),16)$

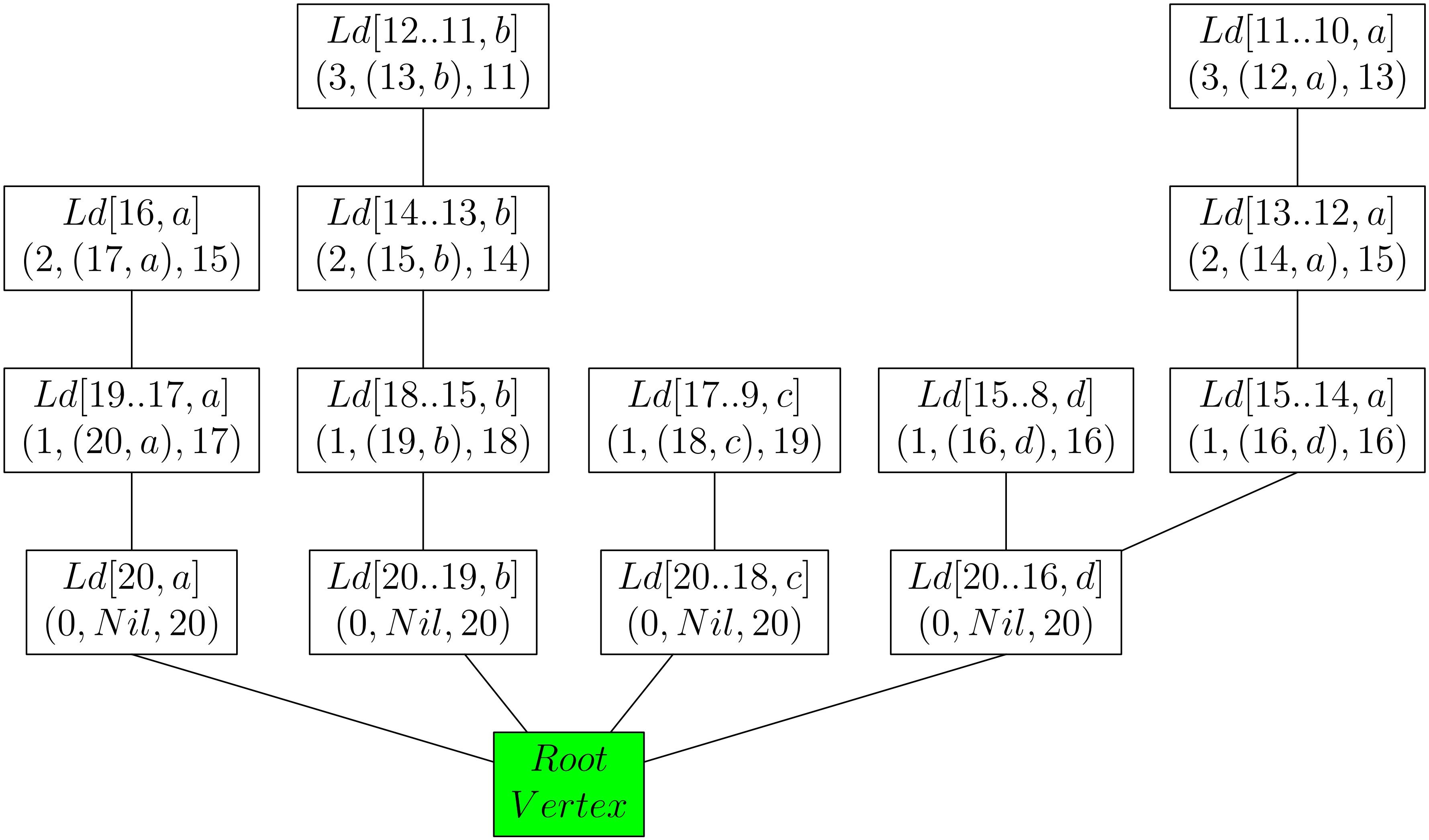
A	16	17	18	19	20
a	$(2,(17,a),15)$	$=$	$=$	$(1,(20,a),17)$	$(0,\mathsf{nil},20)$
b	$=$	$=$	$(1,(19,b),18)$	$=$	$(0,\mathsf{nil},20)$
c	$=$	$(1,(18,c),19)$	$=$	$=$	$(0,\mathsf{nil},20)$
d	$=$	$=$	$=$	$=$	$(0,\mathsf{nil},20)$

$$\mathsf{Ld}_1[13,a] = (2,(14,a),15)$$

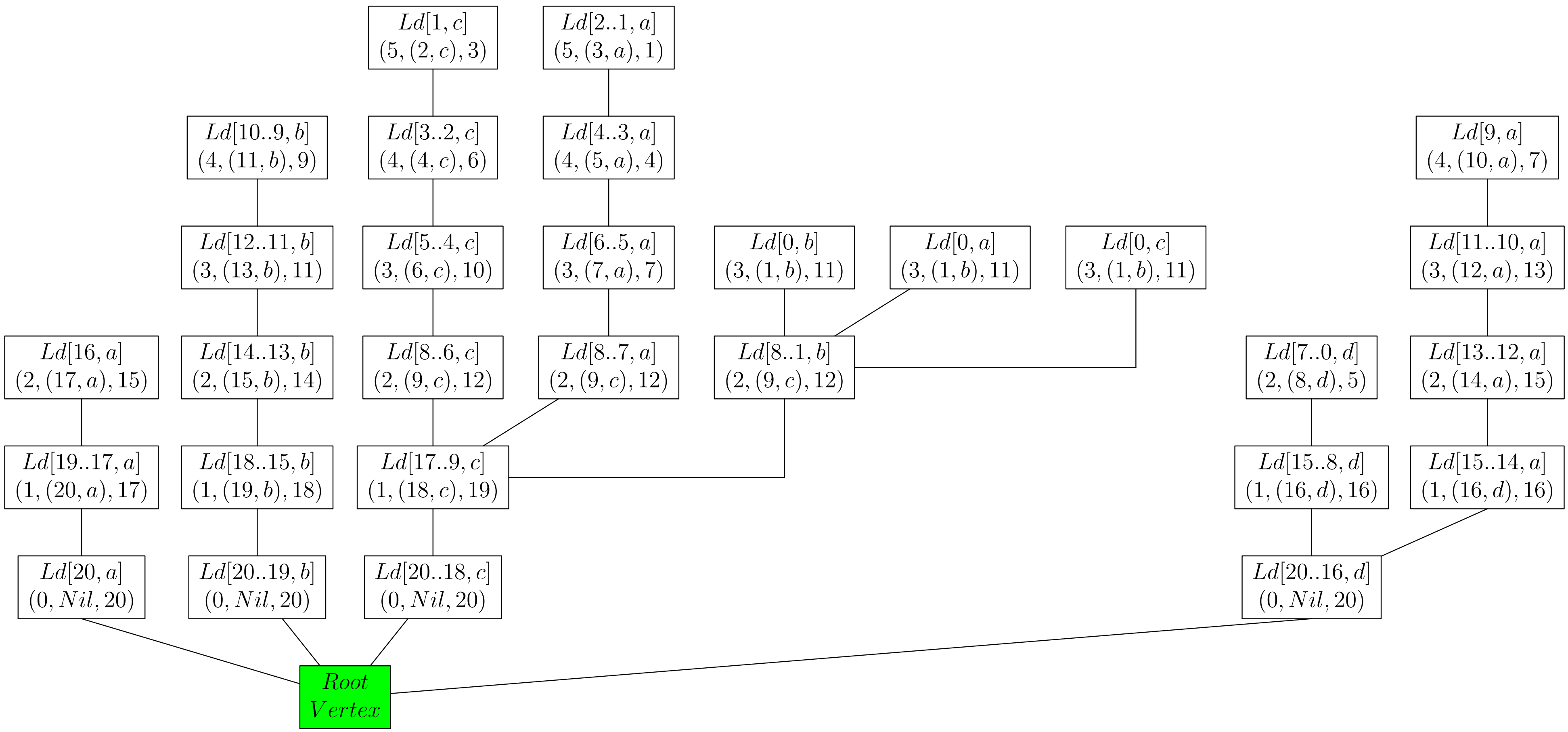
$\delta(a \cdot x_1[13..20], x_1[13..20]) = 2$ where $x_1[13..20] = abdacba$

word stored in $\mathsf{Ld}_1[13,a]$ is ad which is a subword of $x_2[15..20] = adabc$

Part of the left distinguisher tree of x_1



The left distinguisher tree of x_1



From $O(|A|(|x_1| + |x_2|))$ to $O(|A| + |x_1| + |x_2|)$

The $|A| \times |x|$ matrices are sparse: they have $O(|A| + |x_1| + |x_2|)$ different elements

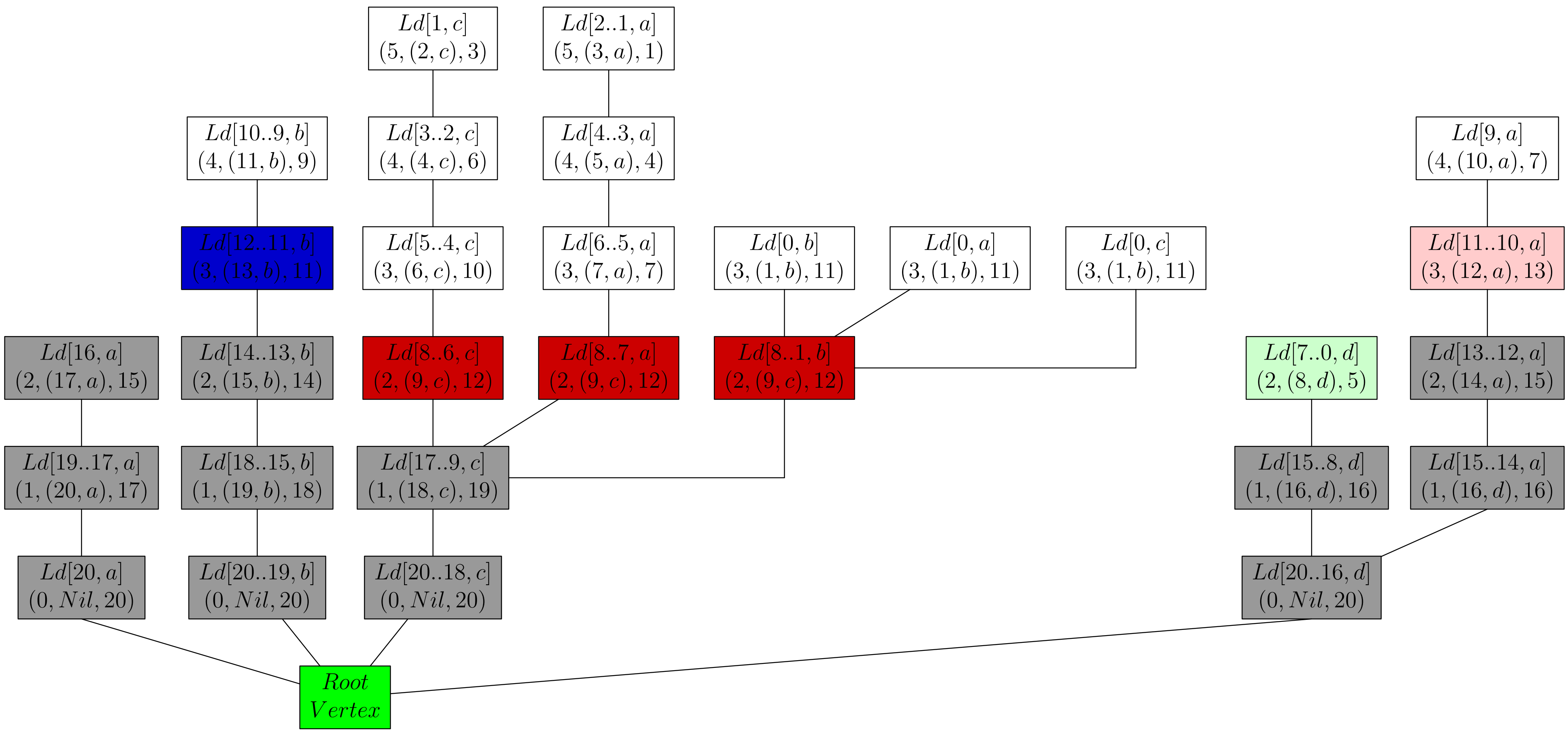
We use inversion techniques to obtain the left distinguisher trees

It would be nice to have mechanisms to recover $\mathsf{Ld}[i, a]$ in $O(1)$ time (does such a mechanism exist?)

We can survive with an $O(|A| + |x|)$ sum of access times throughout the algorithm (the right data in the right place at the right time)

Computing the s's in $\mathsf{Ld}[i, a] = (l, (i', a'), s)$ is the trickiest part

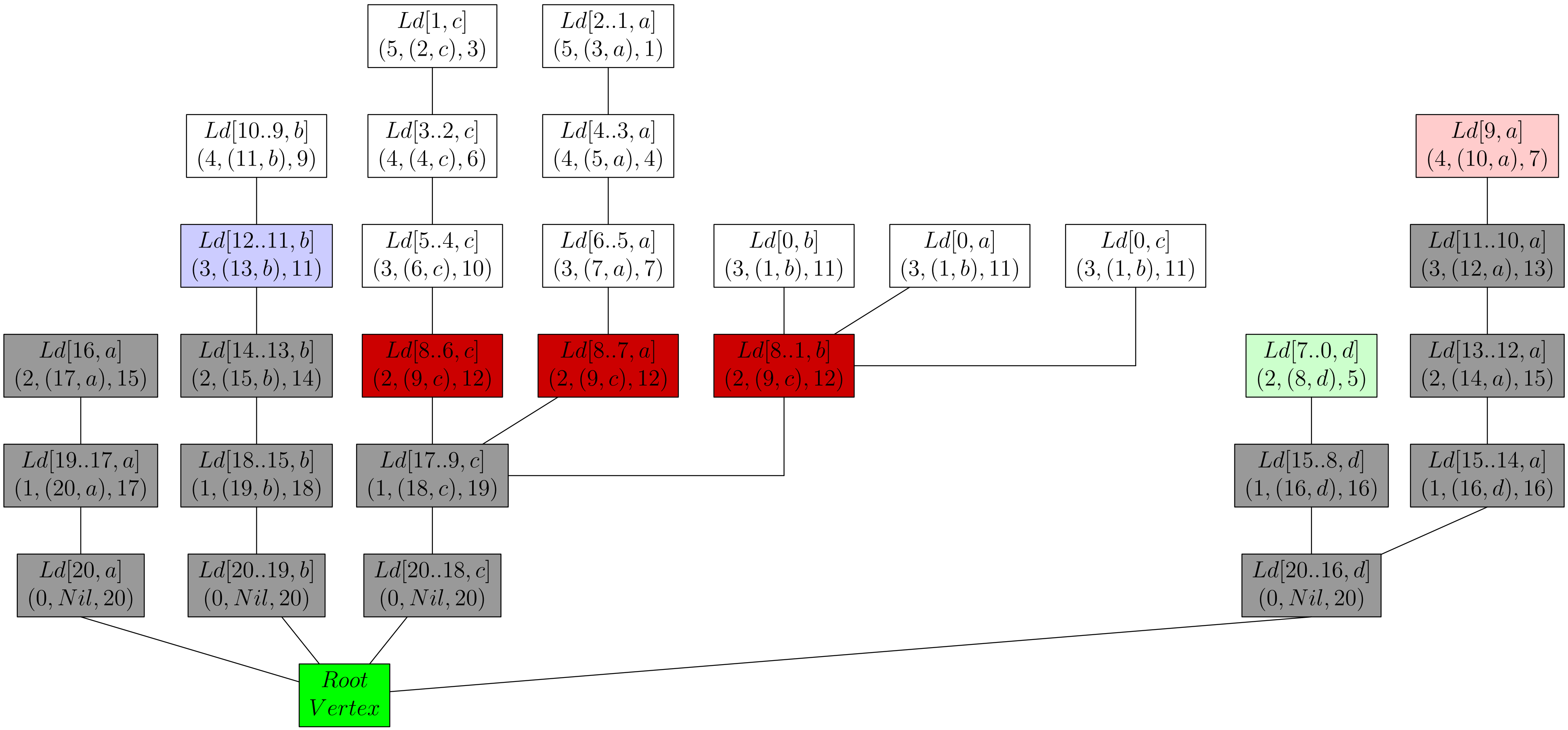
Just entered $x_2[15] = b$, $s \geq 14$'s are known



x_2= b a c c a d c a b b c b c a b a d a b c

*

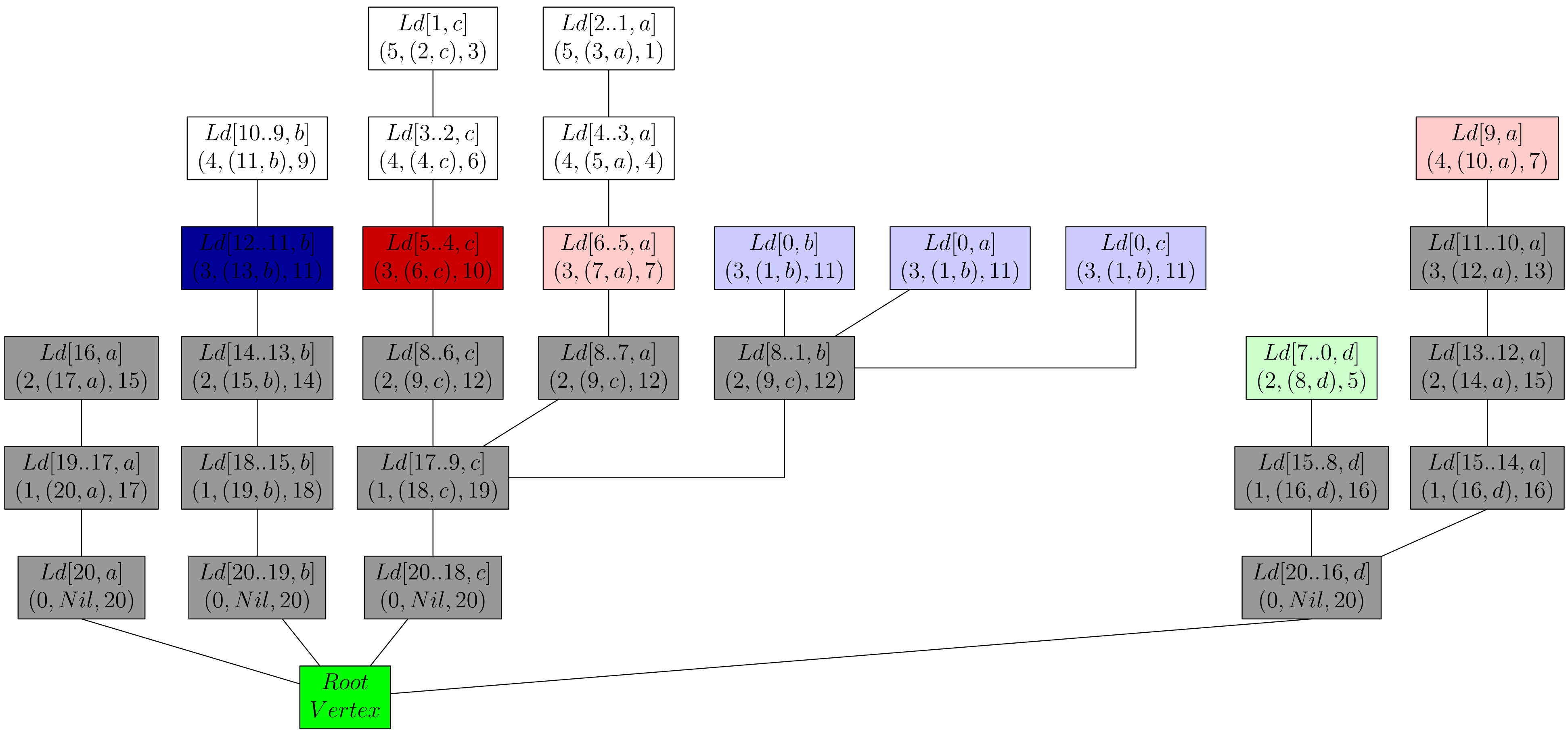
Just entered $x_2[14] = a$, $s \geq 13$'s are known



x_2= b a c c a d c a b b c b c a b a d a b c

*

Just entered $x_2[13] = c$, $s \geq 12$'s are known



x_2= b a c c a d c a b b c b c a b a d a b c

*

Ours is a least common ancestor (LCA) problem

$$[x_1]_0 = [x_2]_0 = [\lambda]_0 = A^*$$

$$[x_1]_1 = [x_2]_1 = [abcd]_1$$

$$[x_1]_2 = [x_2]_2 = [abcdabcd]_2$$

$$[x_1]_3 = [x_2]_3 = [abcdabcdabc]_3$$

$$[x_1]_4 = [bacacdcababdacba]_4 \qquad [x_2]_4 = [bacacdabcabcdabc]_4$$

$$[x_1]_5 = [bacacacdcabababdacba]_5 \qquad [x_2]_5 = [baccadcabbccaabdabc]_5$$

$$[x_1]_6 = \{\, x_1 \,\} \qquad [x_2]_6 = [baccadcabbcbcabadabc]_6$$

$$[x_2]_7 = \{\, x_2 \,\}$$

A family of $O(|A| + |x|)$ algorithms: "the subword calculus"

For every x there exists a least t, $t \leq |x| + 1$, such that $[x]_t$ is a singleton:

$$A^* = [x]_0 \supset [x]_1 \supset [x]_2 \supset \cdots \supset [x]_m \supset \cdots \supset [x]_t = \{\, x \,\} = [x]_{t+1} = \cdots$$

For every $x \neq 1$ there exists a largest s, $s < t$, such that $[x]_s$ is idempotent

For every m, x contains a minimal word which is m-equivalent to it, these subwords can be represented by the following structure:

```
x_1= b c a c a c a d c a b a b a b d a c b a
     3 5 5 4 4 3 3 2 2 5 5 4 4 2 2 1 4 1 1 1
m=3  b         c a d c         a b d   c b a
```

Can this vector of m's be computed in $O(|A| + |x|)$ time?

A message to take home

Please self-archive

your scientific papers

Free Online Scholarship (FOS)
www.earlham.edu/~peters/fos

Nature Debates: e-access
www.nature.com/nature/debates/e-access