Optimal Tree Node Ordering for Child/Descendant Navigations

Atsuyuki Morishima, University of Tsukuba
Keishi Tajima, Kyoto University
Masateru Tadaishi, University of Tsukuba

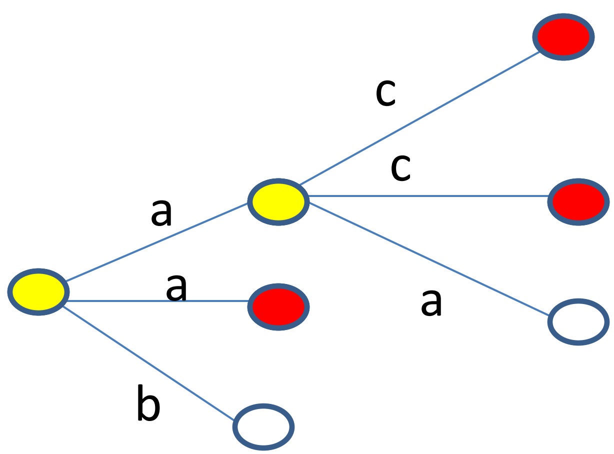
ICDE 2010

Background

- There are many applications to deal with huge tree data:
  - File Directories, XML data, taxonomies, and some of bioinformatics data.
- *Set-based navigations* are fundamental operations for interactively accessing such data.

Set-based Navigations

- A user specifies one node and a type of navigation, then the system retrieves all the nodes reachable from that node via that type of navigation.
- The user browses the retrieved nodes, selects one node, and repeats set-based navigations from it.



Four Basic Navigations

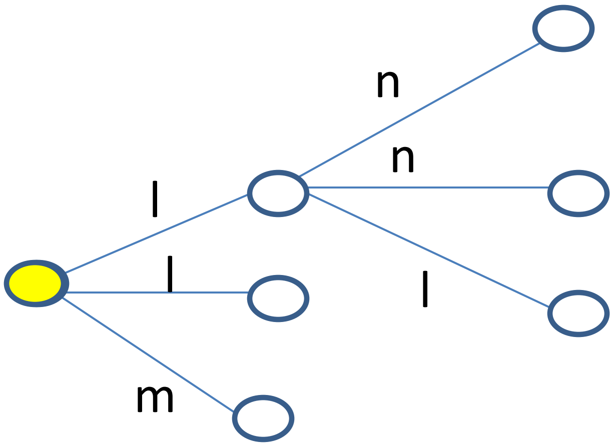
In such an interactive browse-and-traverse style of access, users rarely specify complex traverse conditions. Therefore, we focus on the following most basic tree navigations:

child $$a \longrightarrow X$$

descendant $$a \overset{*}{\longrightarrow} X$$

l-child $$a \overset{l}{\longrightarrow} X$$

l-descendant $$a \overset{l*}{\longrightarrow} X$$



Four Basic Navigations

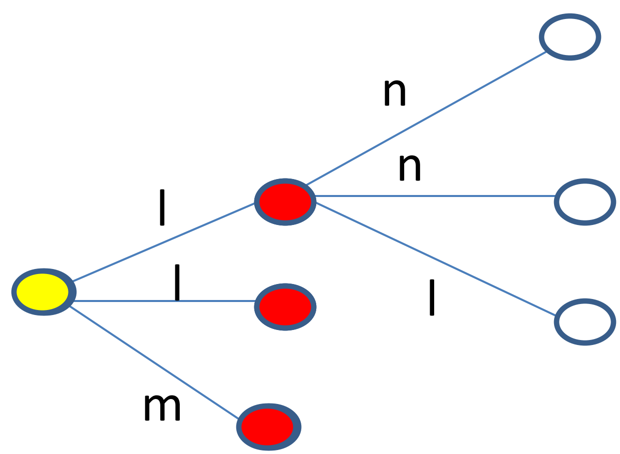
In such an interactive browse-and-traverse style of access, users rarely specify complex traverse conditions. Therefore, we focus on the following most basic tree navigations:

child $a \longrightarrow X$

descendant $a \xrightarrow{*} X$

l-child $a \xrightarrow{l} X$

l-descendant $a \xrightarrow{l*} X$



Four Basic Navigations

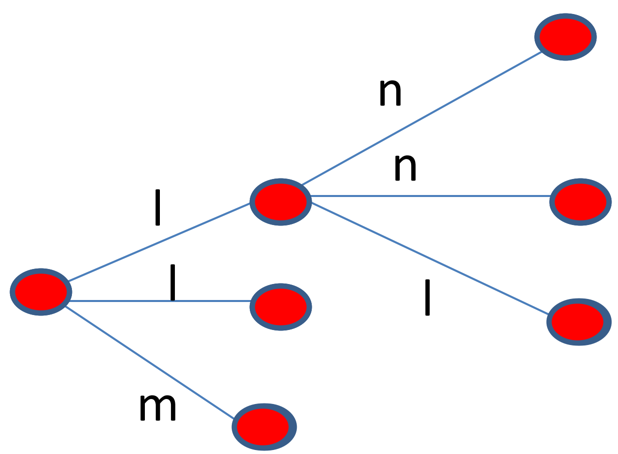
In such an interactive browse-and-traverse style of access, users rarely specify complex traverse conditions. Therefore, we focus on the following most basic tree navigations:

child $a \longrightarrow X$

descendant $a \overset{*}{\longrightarrow} X$

l-child $a \overset{l}{\longrightarrow} X$

l-descendant $a \overset{l*}{\longrightarrow} X$



Four Basic Navigations

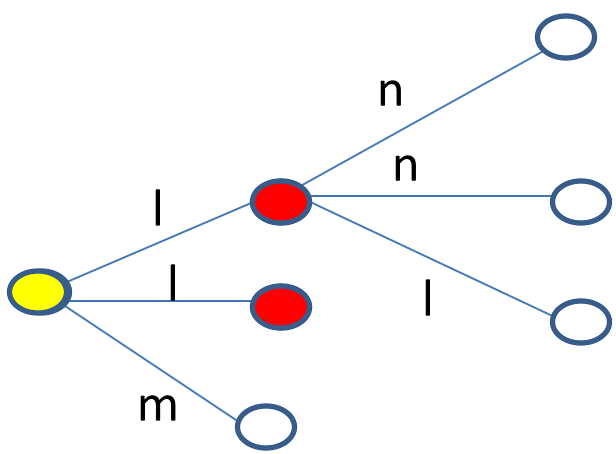
In such an interactive browse-and-traverse style of access, users rarely specify complex traverse conditions. Therefore, we focus on the following most basic tree navigations:

child $a \longrightarrow X$

descendant $a \xrightarrow{*} X$

l-child $a \xrightarrow{l} X$

l-descendant $a \xrightarrow{l*} X$



Four Basic Navigations

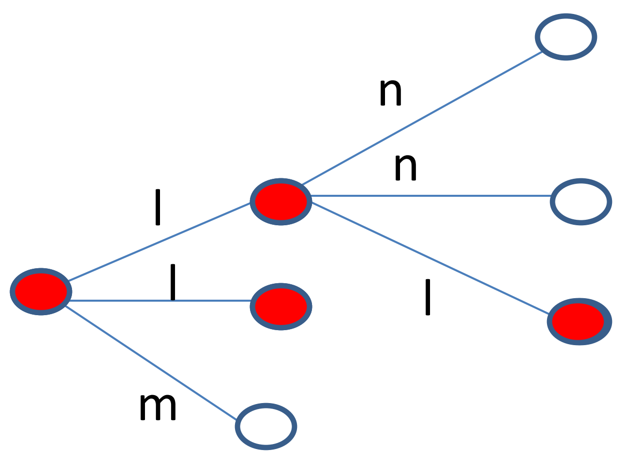
In such an interactive browse-and-traverse style of access, users rarely specify complex traverse conditions. Therefore, we focus on the following most basic tree navigations:

child $a \longrightarrow X$

descendant $a \overset{*}{\longrightarrow} X$

l-child $a \overset{l}{\longrightarrow} X$

l-descendant $a \overset{l*}{\longrightarrow} X$



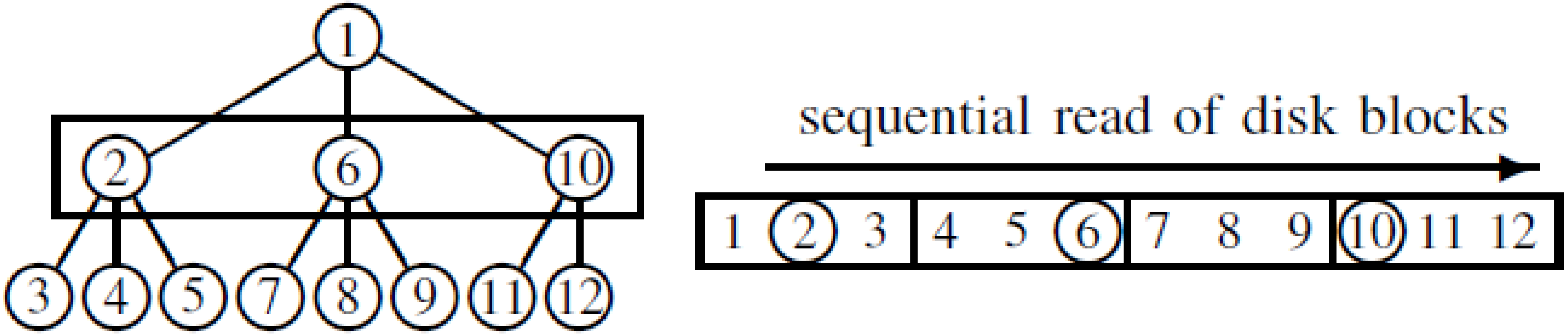
The Problem Addressed

- When the data is huge and stored on the disk, how to store the nodes to achieve efficient evaluation of the navigations is not trivial.
- One approach is to store the nodes in an appropriate order so that the nodes accessed together by these operations are clustered, but how?

What's the Problem?

There is no trivial ordering for set-based navigations which is I/O optimal

Ex) Depth-first Order



Contributions of Our Work

- Show that there is *no* single ordering scheme that is optimal for all the four operations.
- Show three ordering schemes, each of which is optimal only for a subset of them.
- Found that one of the schemes can process all the four operations with disk access to a constant-bounded number of regions on the disks, without accessing irrelevant nodes

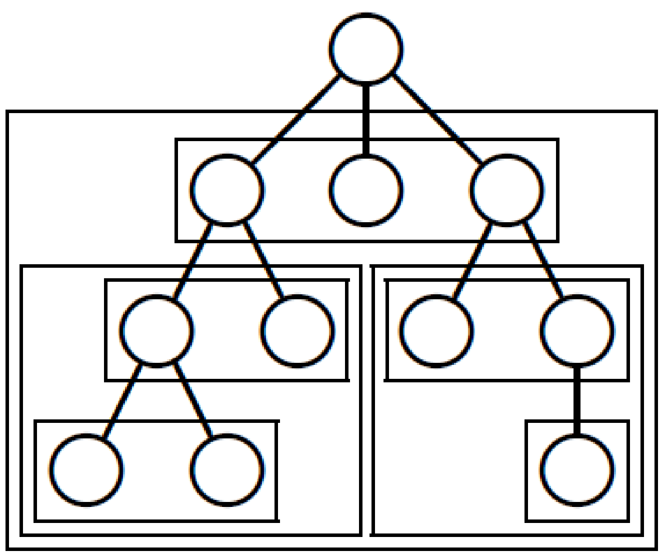
Outline

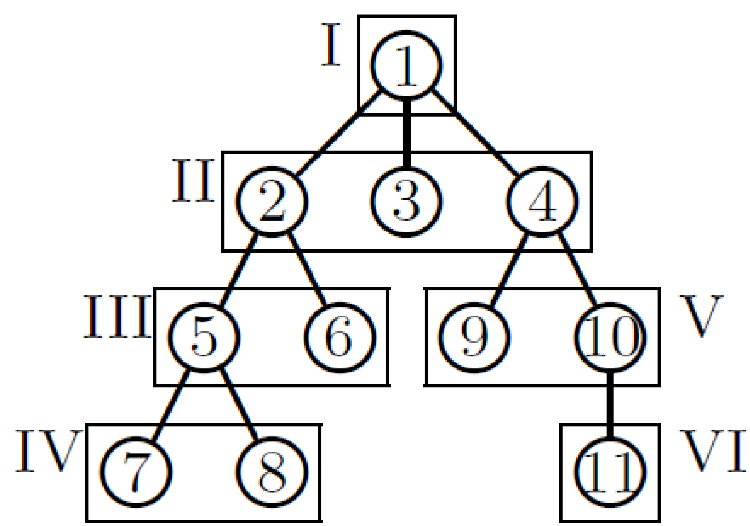
1. Overview and background
2. The first order: $<_t$
3. The second order: $<_t^l$
4. The third order: $<_t^{l*}$
5. Properties and Discussions
6. Summary

$<_t$: Optimal Ordering for $a \rightarrow X$ and $a \overset{*}{\rightarrow} X$

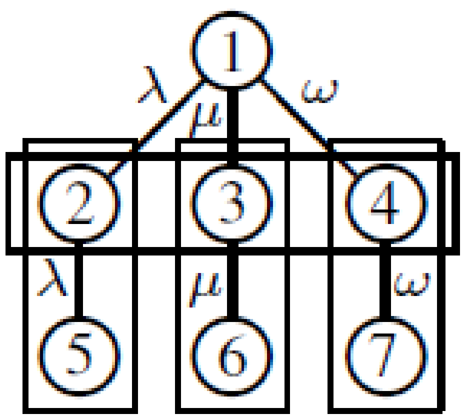
Intuitively,

1. Group siblings who has the same parent,
2. Sort the groups in the depth first order in t, and
3. Sort the siblings within each group in the sibling order

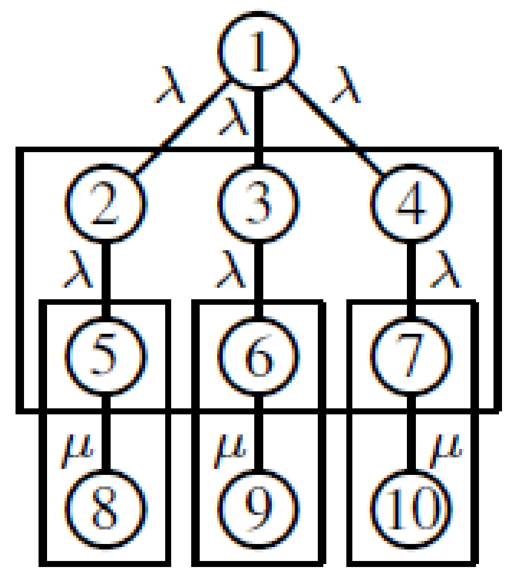




Conflicts caused by $a \xrightarrow{l*} X$



Conflict with $a \rightarrow X$



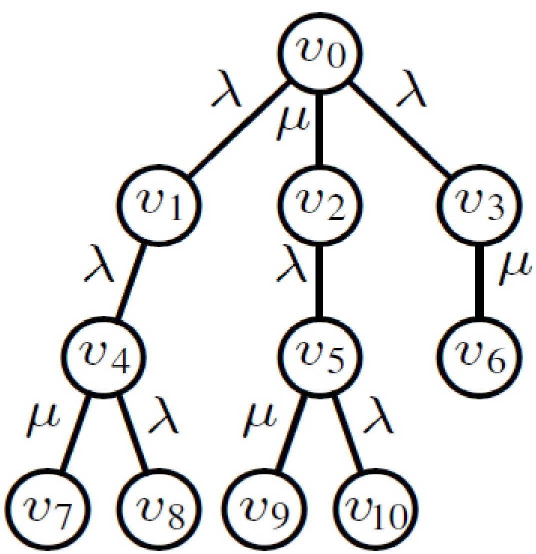
Conflict with $a \xrightarrow{*} X$

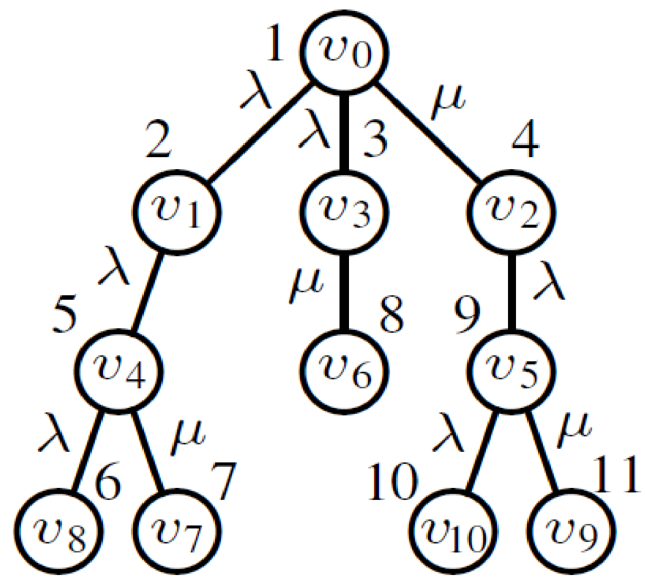
- This proves that there is no single ordering which is optimal for all the four set-based navigations
- We have two choices.

$<_t^l$: Optimal Ordering for $a \rightarrow X$, $a \overset{*}{\rightarrow} X$, and $a \overset{l}{\rightarrow} X$

Intuitively,

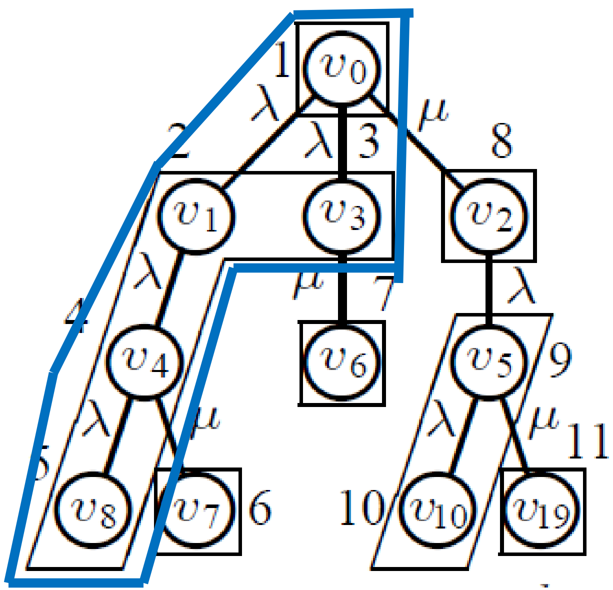
1. Sort the children of each node primarily by the labels of their incoming edges, and secondly by their original sibling order.
2. Sort the nodes in the same way as $<_t$

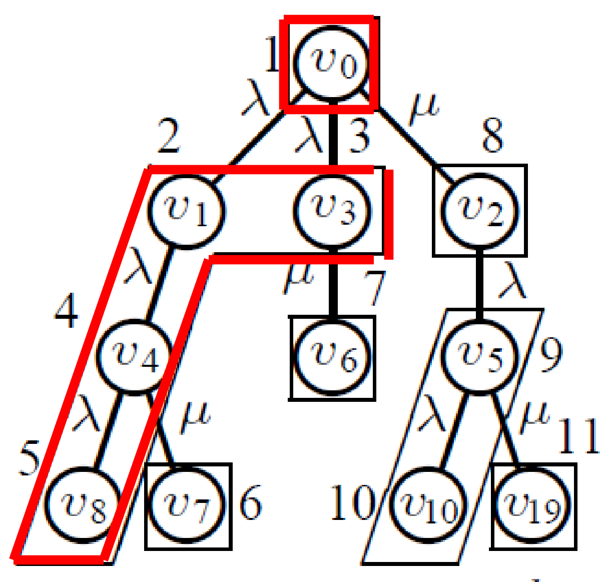




$<_t^{l*}$: Optimal Ordering for $a \xrightarrow{l} X$ and $a \xrightarrow{l*} X$ (1/2)

- A maximal unilabeled connected subgraph: A Maximal connected subgraph that includes only one kind of edge label
- Unilabeled clusters : subgraphs created from maximal unilabeled connected subgraphs by removing their root nodes

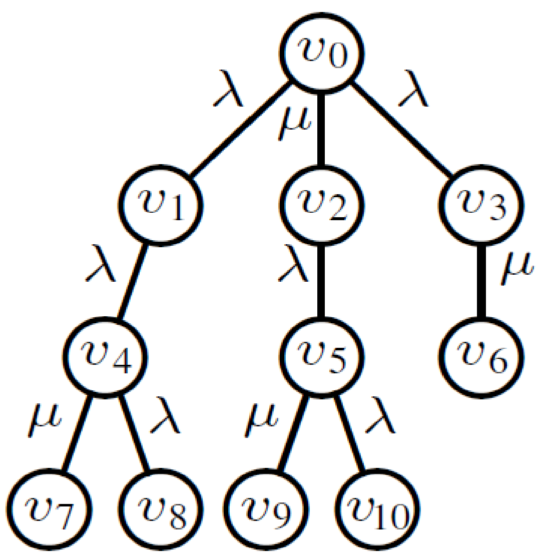


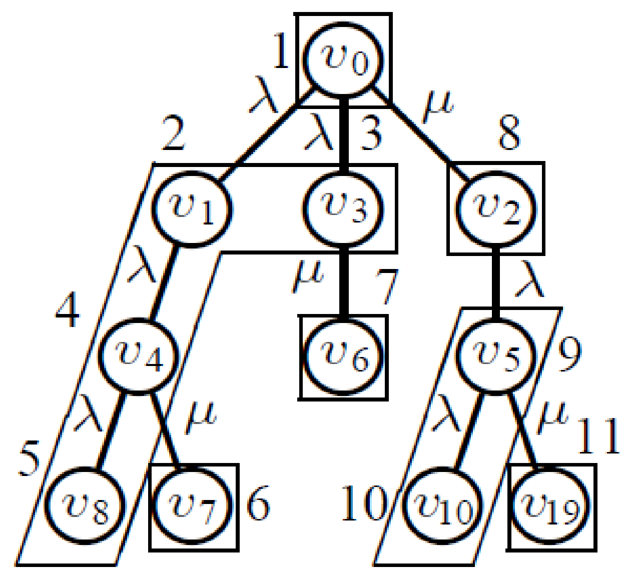


$<_t^{l*}$: Optimal Ordering for $a \xrightarrow{l} X$ and $a \xrightarrow{l*} X$ (2/2)

Intuitively,

1. Sort the unilabeled clusters primarily in the depth-first order of their original roots, and secondly in the dictionary order of their labels
2. Within each unilabeled cluster, we sort nodes in the order of $<_t$





Theorems

- There are procedures for $a \xrightarrow{l} X$ and $a \xrightarrow{l*} X$ with $<_t^{l*}$ that are I/O optimal.
- Similar theorems hold for $<_t$ and $<_t^l$.

The numbers of disk regions to access

	$a \rightarrow X$	$a \xrightarrow{*} X$	$a \xrightarrow{l} X$	$a \xrightarrow{l*} X$
$<_t$	1	1	n	n
$<_t^l$	1	1	1	n
$<_t^{l*}$	l	2	1	1

n : the number of nodes, l : the number of edge labels

Related Work

- Indexing schemes and labeling schemes for path queries on XML data
  - ⇔Completely different problem because of differences in the number of steps, starting nodes, and support of the closure operator for specific edge-labels.
- Storage schemes for trees (Clustering-based, two-dimensional disk-space architecture, …)

  ⇔ Efficiently executed by the current disk access interface. No modification to the disk access interface
- Many processing schemes of path queries that scan nodes in the depth-first order.
  - ⇔ Efficient for the fundamental navigations without additional indices

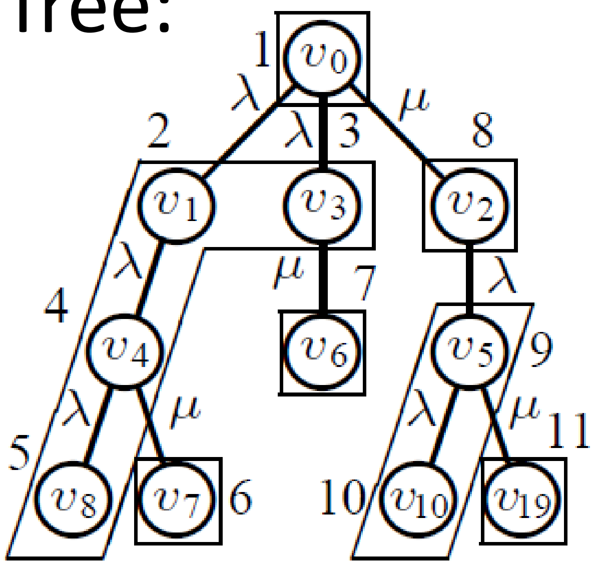
Summary

We showed how we should order nodes of labeled trees on the disk for efficient processing of set-based navigations.

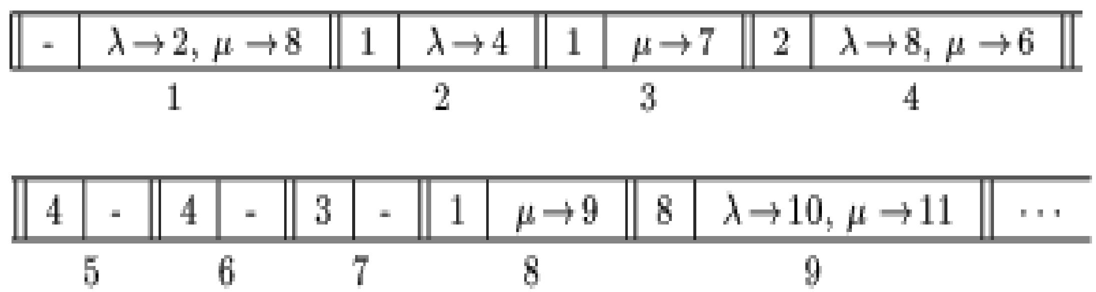
- Showed there is *no* single ordering scheme that is optimal for all the four operations.
- Showed a couple of schemes, each of which is optimal only for a subset of them.
- Found that one of the schemes can process all the four operations with disk access to a constant-bounded number of regions on the disks, without accessing irrelevant nodes

Processing $a \xrightarrow{l*} X$ with $<_t^{l*}$

Tree:



Disk image:



Procedure:

1. Scan the node entries starting at firstChild(a,l)
2. Stop the scan when we reach a node n s.t either:
   1. Parent(n)≠a and parent(n)<firstChild(a,l), or
   2. Label(n) ≠l