Matching Subsequences in Trees

Philip Bille and Inge Li Gørtz
IT University of Copenhagen

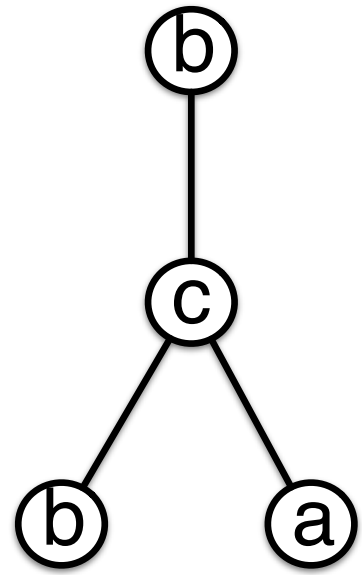
Basic Setup

Trees are rooted and labeled.

- **Rooted**: A distinguished node is designated as a *root*.
- **Labeled**: Each node is assigned a *label* from an $\sum$.

We are interested in the sequences of labels formed by the *root-to-leaf* paths in trees.

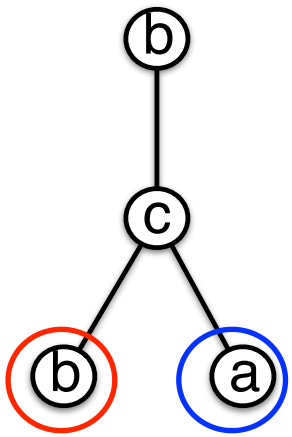
Example

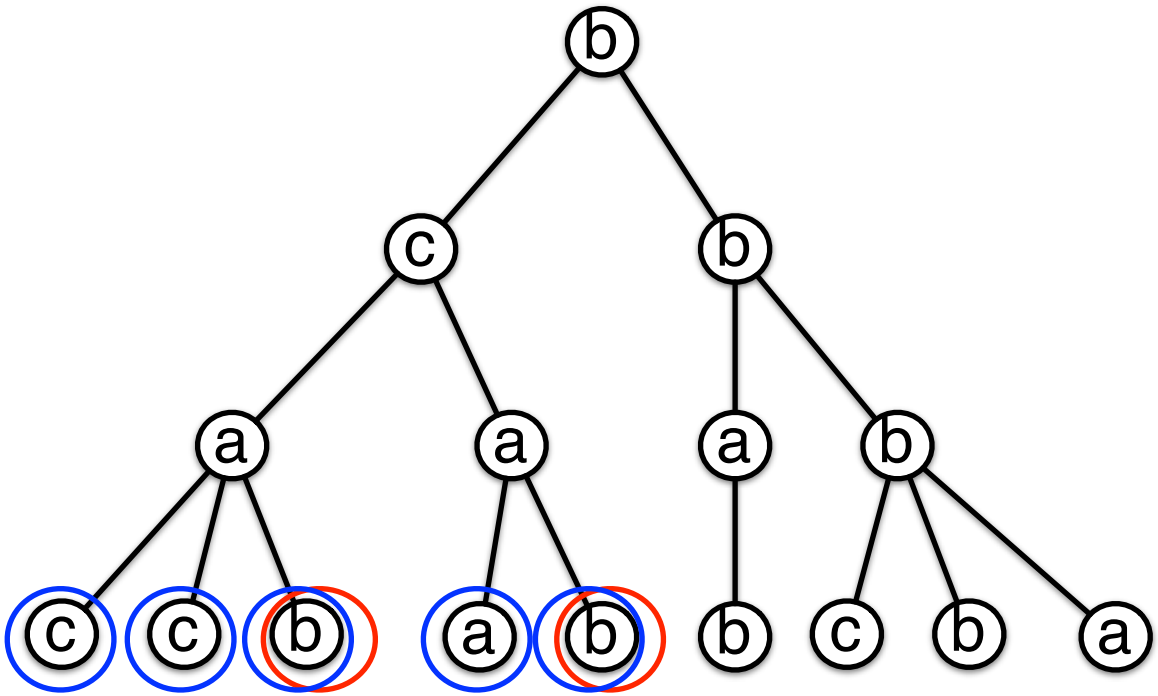


$\{bcb, bca\}$

Tree Path Subsequence

- All paths are root-to-leaf unless otherwise stated.
- A path p is a *subsequence* of a path t, denoted $p \sqsubseteq t$, if p can be obtained by removing (0 or more) nodes from t.
- The *tree path subsequence problem* (TPS) is to report for each path p in P all paths t in T such that $p \sqsubseteq t$.


b
c
b
a


b
c
b
a
a
a
b
c
c
b
a
b
b
c
b
a

Trivial Solutions

- Let p and t be paths in $P$and T, respectively.
- We can determine in $O(|p| + |t|)$ time if $p \sqsubseteq t$.
- Applying this to all $O(n_P n_T)$ pair of paths gives an algorithm with running time

$$O(n_P n_T (n_P + n_T))$$

Trivial Solutions

- We can preprocess t in $O(|t| \log |t|)$ time such that queries $p \sqsubseteq t$ can be done in time $O(|p| \log |t|)$ [BY91].
- Applying this to each of the $O(n_T)$ paths in T gives an algorithm with running time

$$O(n_T^2 \log n_T + n_P^2 \log n_T)$$

Results

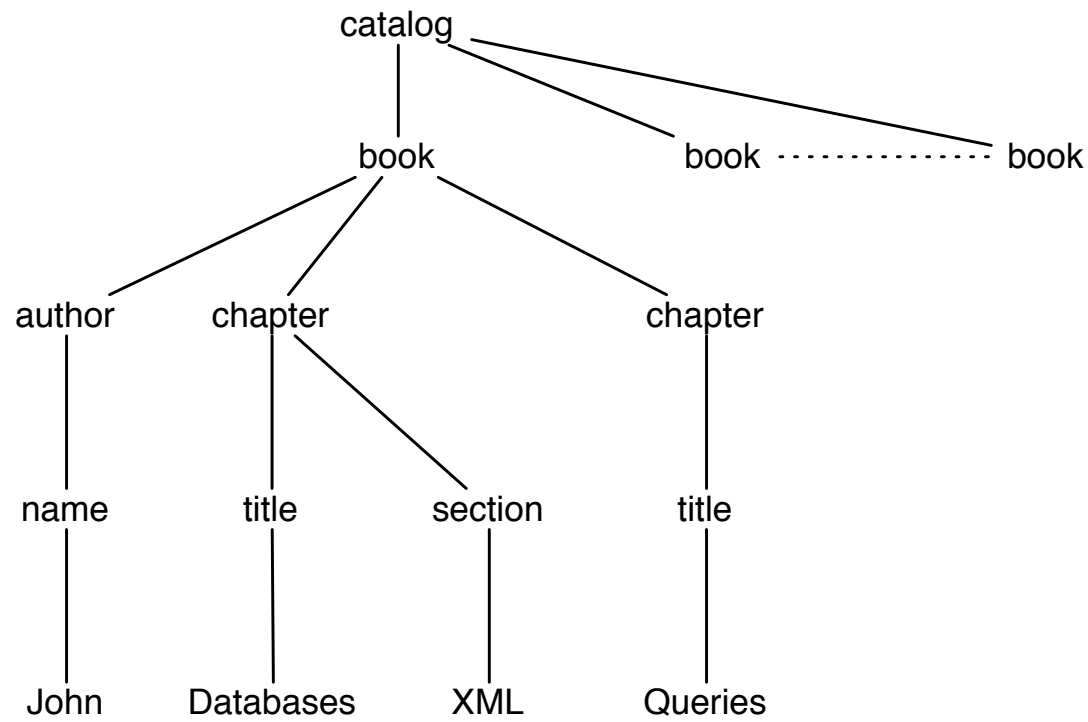
Time	Space	Reference
$O(l_P n_T + n_P)$	$O(d_P l_T + n_P + n_T)$	[Chen00]
$O(n_P l_T + n_T)$		
$O(l_P n_T + n_P)$	$O(n_T + n_P)$	This paper
$O(n_P l_T + n_T)$		
$O\left(\frac{n_T n_P}{\log n_T} + n_T + n_P \log n_P\right)$		

Applications

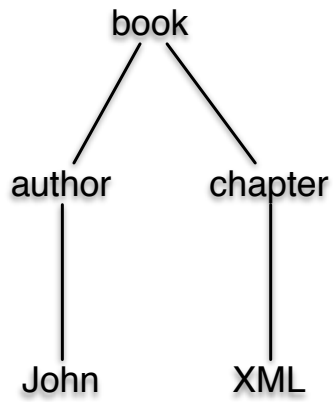
- Batched *path-queries*.
- Filter for *tree inclusion* queries.

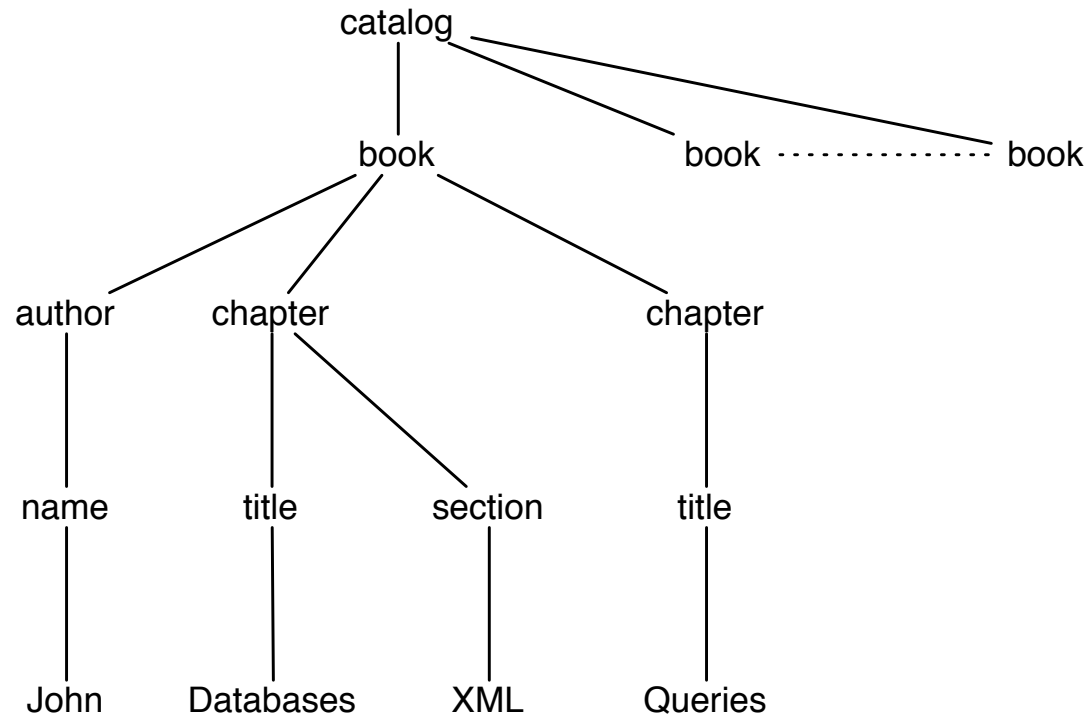
Path queries on XML

- Find all books written by John.
- Find all books with a chapter with some XML.



Path queries on XML





Complexity

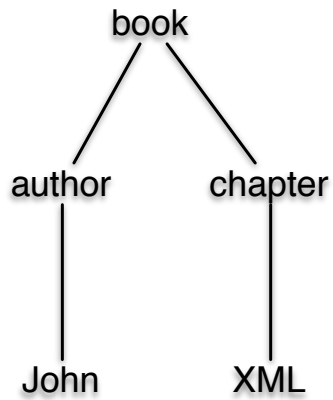
- A path query for a single path p can be solved in $O(|p| + n_T)$ time.
- Hence, we can answer k path queries for paths $p_1, \ldots, p_k$ in time.

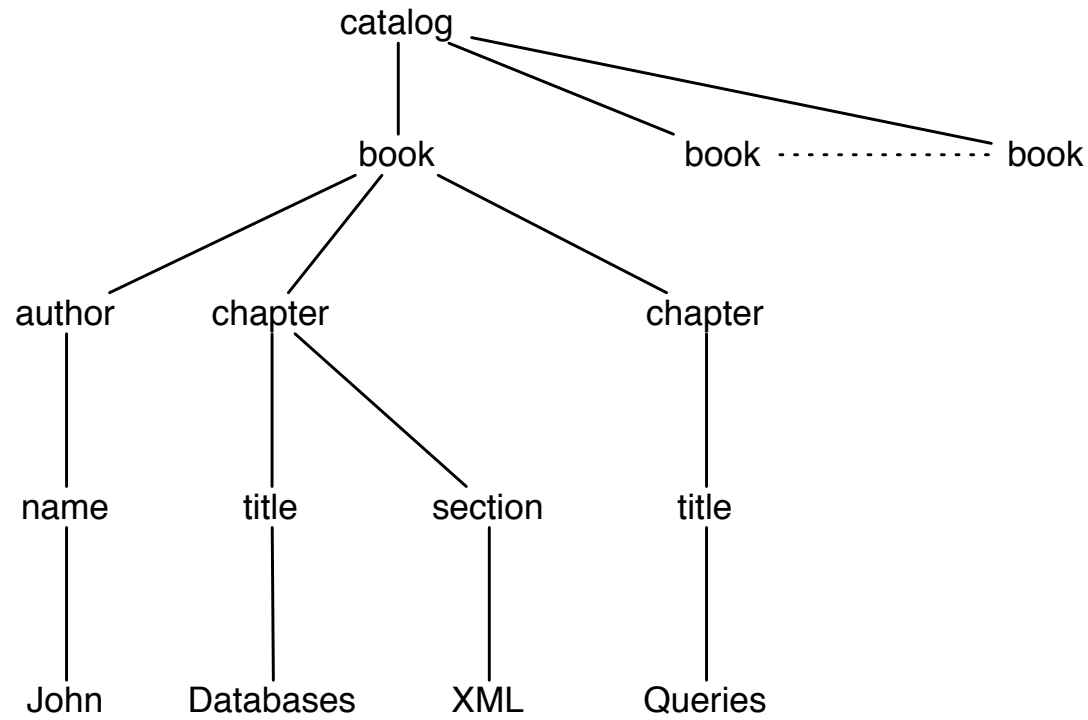
$$O(|p_1| + \cdots + |p_k| + kn_T)$$

Complexity

- Alternatively, construct the *trie* P of the paths and solve TPS on P and T.
- Depending on the overlap n_P can be much smaller than $|p_1| + \cdots + |p_k|$.
- This solution is *at least* as good as the naive algorithm and faster in most cases.

Tree Inclusion



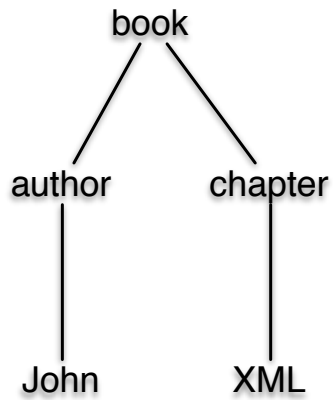


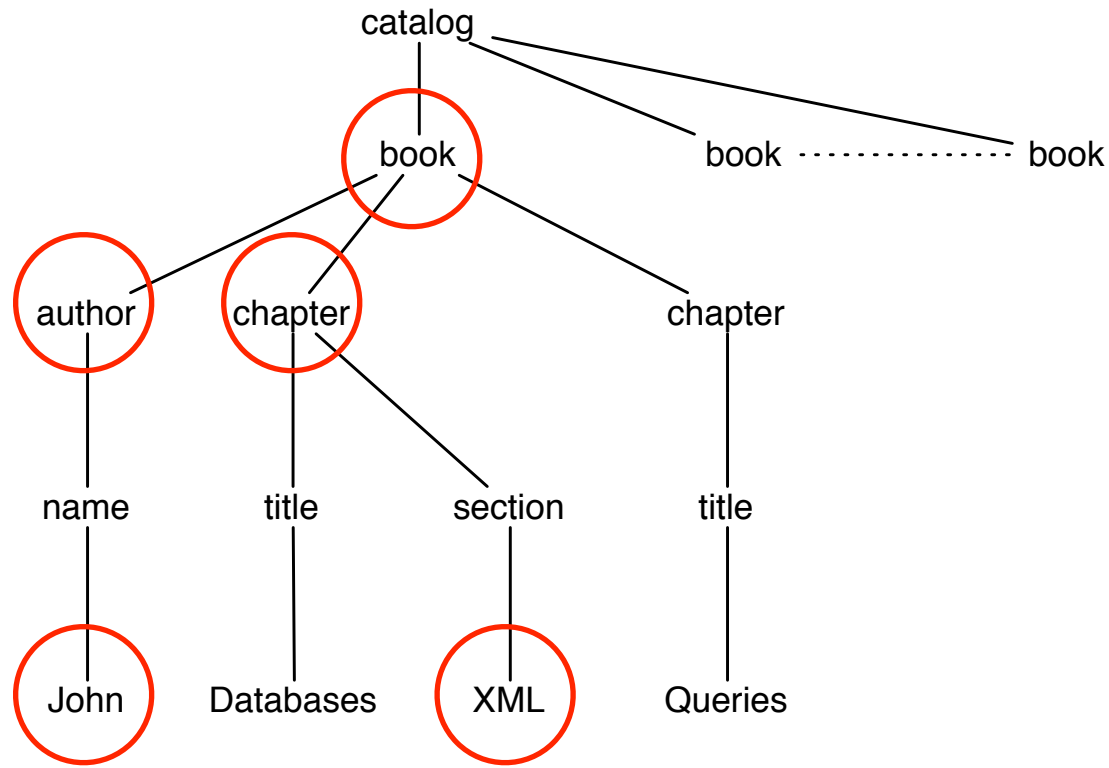
Find all books written by John with a chapter with some XML.

Tree Inclusion

- The query is an instance of the *tree inclusion problem* (TI): Decide if P can be obtained from T by *deleting* nodes of T.
- Deleting node y means making the children of y children of the parent of y and removing y.

Tree Inclusion





Find all books written by John with a chapter with some XML.

Tree Inclusion

- Tree inclusion is NP-complete.
- A necessary condition for P to be included in T is that all paths in P are subsequences of paths in T.
- We can use TPS to quickly rule out trees or parts of a tree that might be included in T.

Algorithms

- First, we give a simple framework for solving the problem.
- Secondly, we describe the worst-case efficient algorithm.

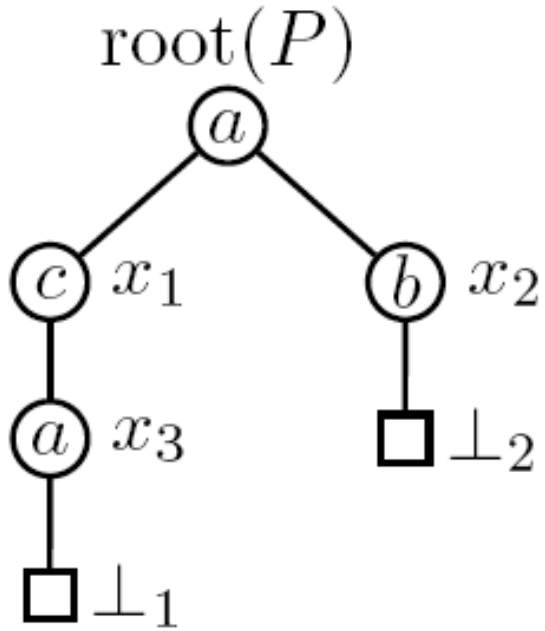
Framework

- Add a numbered *pseudo-leaf* to each leaf in P identifying paths in P.
- Compute at each node y in T a *state*

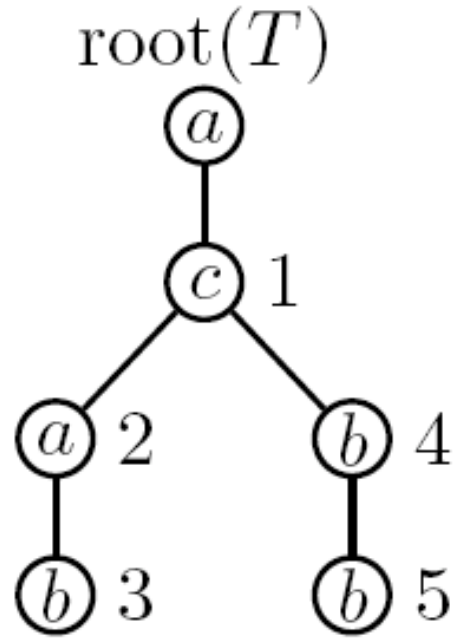
$$X_y \subseteq V(P)$$

satisfying that if $x \in X_y$ then

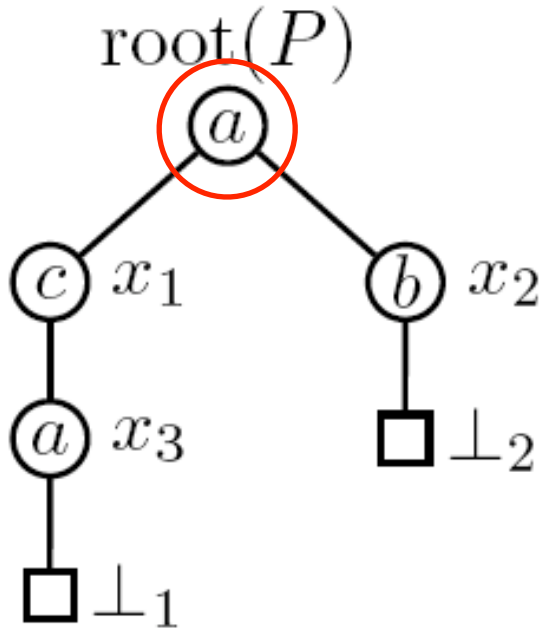
$$\mathrm{path}(\mathrm{parent}(x)) \sqsubseteq \mathrm{path}(y)$$



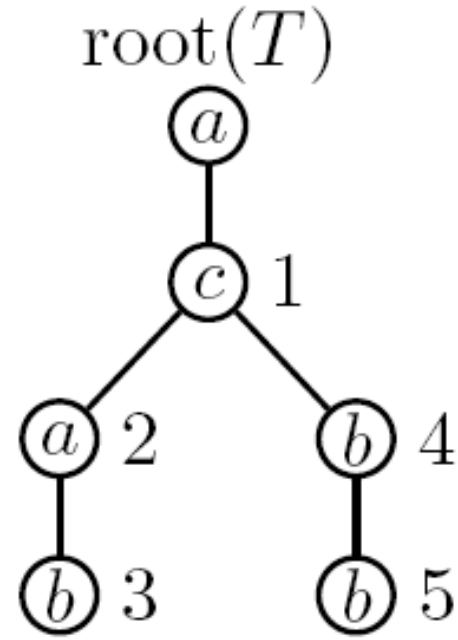
P



T

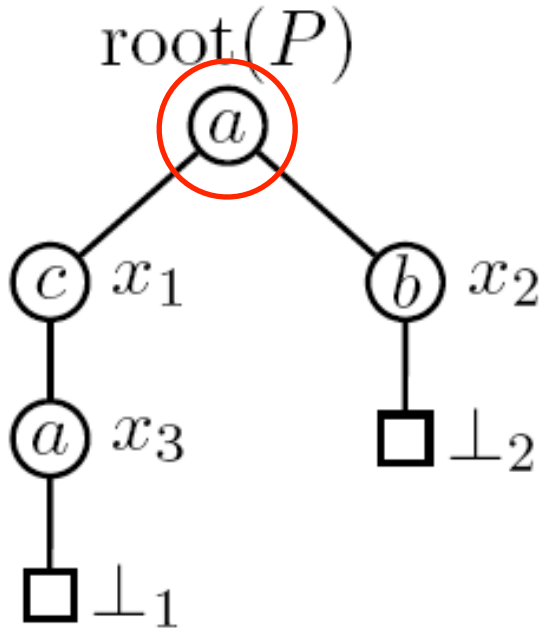


P

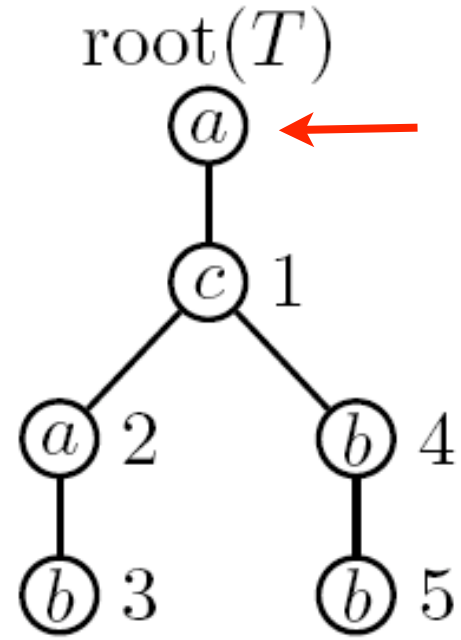


T

Initially, set $X = \{\text{root}(P)\}$

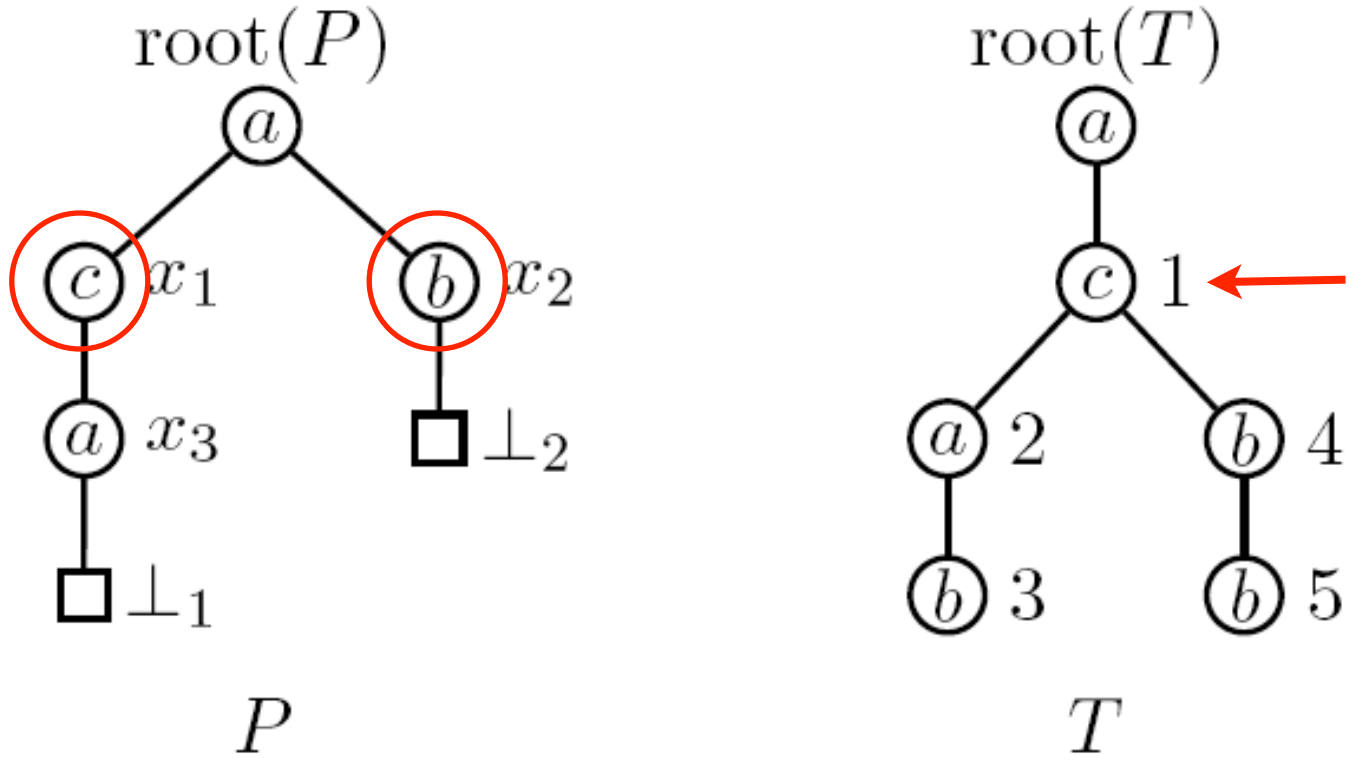


P

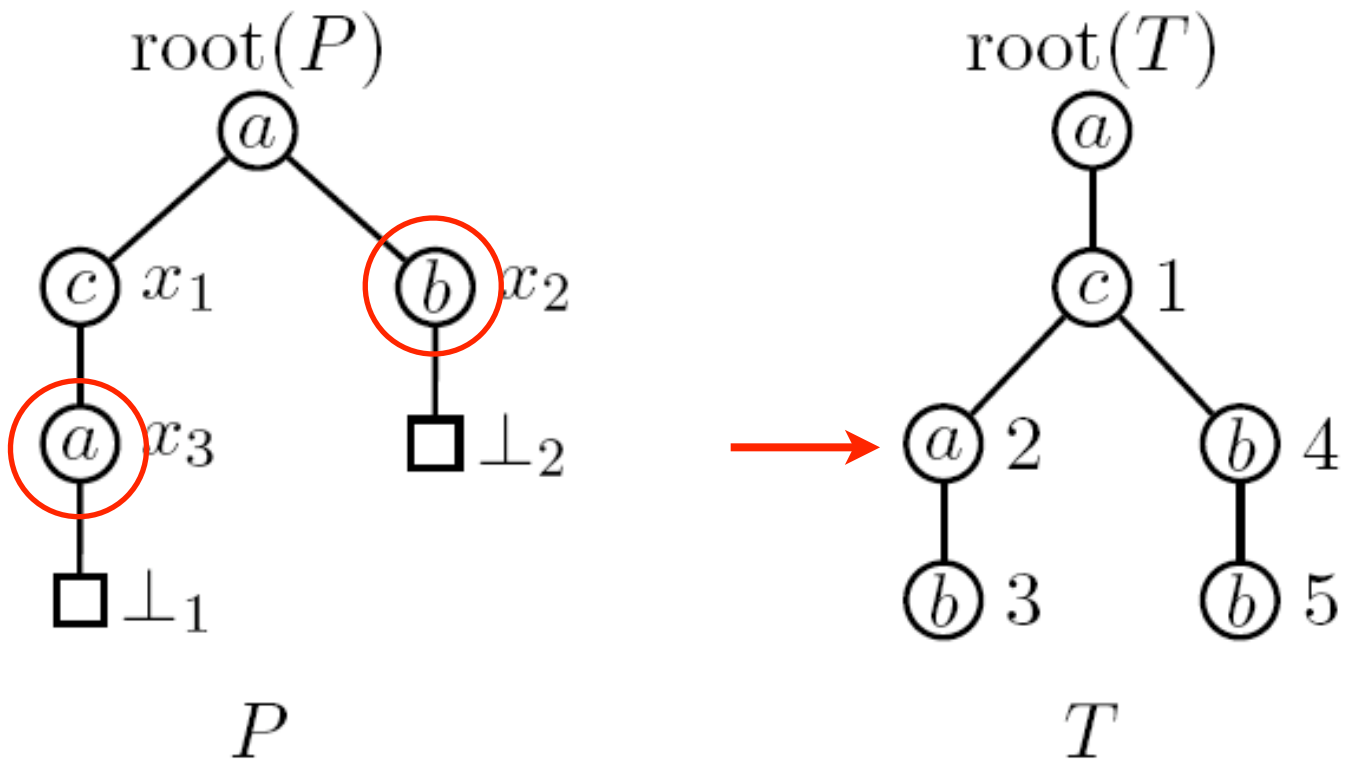


T

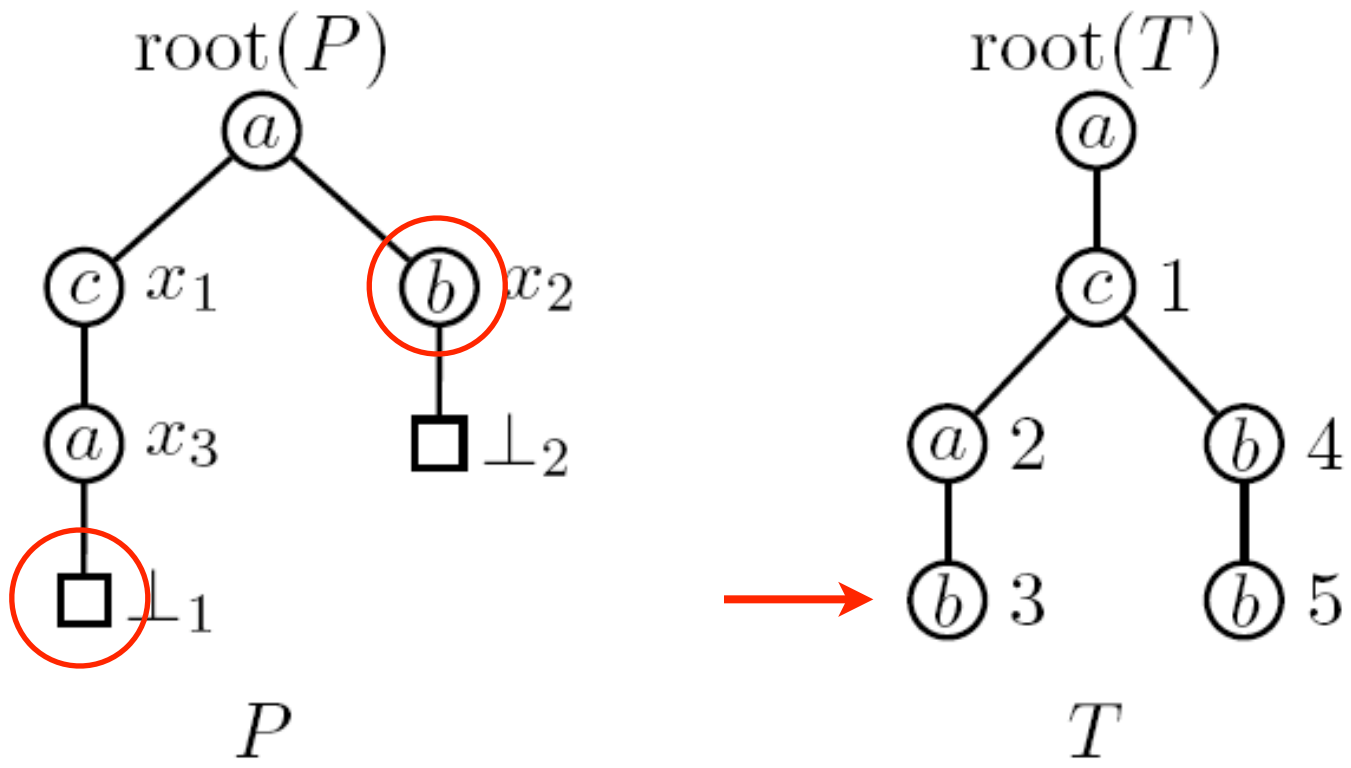
$X_{\mathrm{root}(P)} =$ {Nodes in X not labeled a} $\cup$
children{Nodes in X labeled a} $= \{x_1, x_2\}$



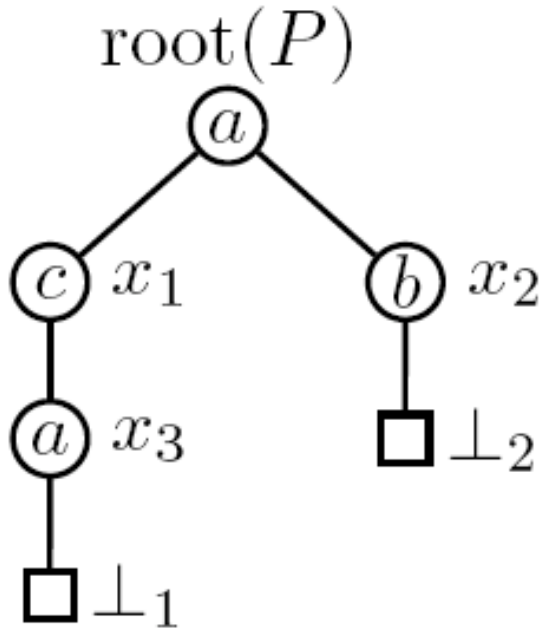
$X_1 =$ {Nodes in $X_{\mathrm{root}(P)}$ not labeled c} $\cup$
children{Nodes in $X_{\mathrm{root}(P)}$ labeled c } $= \{x_3, x_2\}$



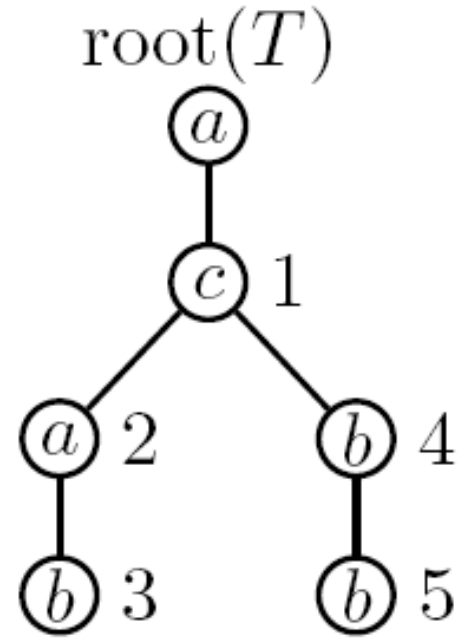
$$X_2 = \{\perp_1, x_2\}$$



$$X_2 = \{\perp_1, x_2\}$$
$$X_3 = \{\perp_1, \perp_2\}$$

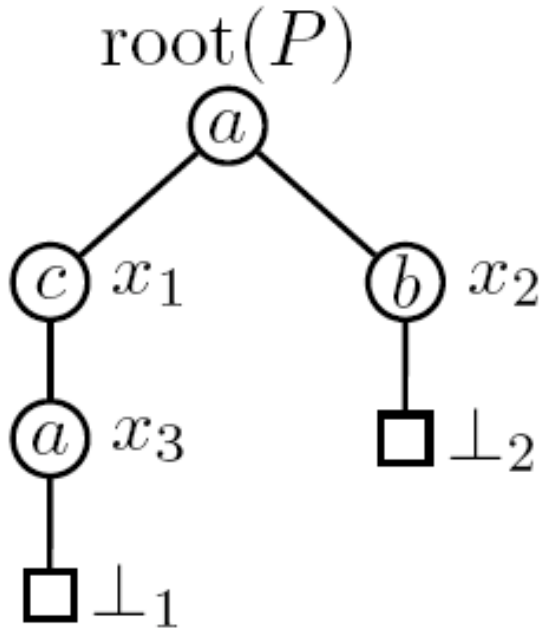


P

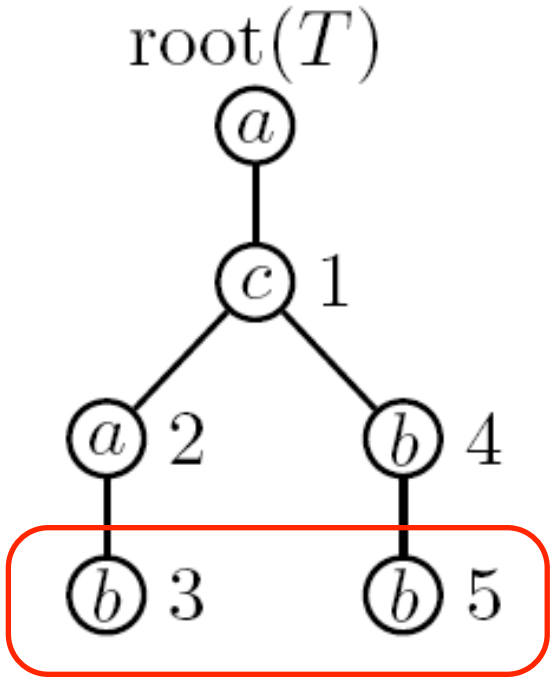


T

$$X_2 = \{\bot_1, x_2\} \qquad X_4 = \{x_3, \bot_2\}$$
$$X_3 = \{\bot_1, \bot_2\} \qquad X_5 = \{x_3, \bot_2\}$$



P



T

$$X_2 = \{\bot_1, x_2\} \qquad X_4 = \{x_3, \bot_2\}$$
$$X_3 = \{\bot_1, \bot_2\} \qquad X_5 = \{x_3, \bot_2\}$$

Simple Algorithm

- Immediate algorithm: Depth-first traversal of T maintaining states as linked-lists.
- For each node y, $|X_y| \leq l_P$.
- Each step uses $O(l_P)$ time.
- $O(l_P n_T + n_P)$ time and space.

Improved algorithm

- Exploiting the "overlap" among states using more sophisticated data structures we reduce space and get another time bound:
- **Thm.** For trees P and T, TPS can be solved in $O(\min(l_P n_T + n_P, n_P l_T + n_T))$ time and $O(n_P + n_T)$ space.

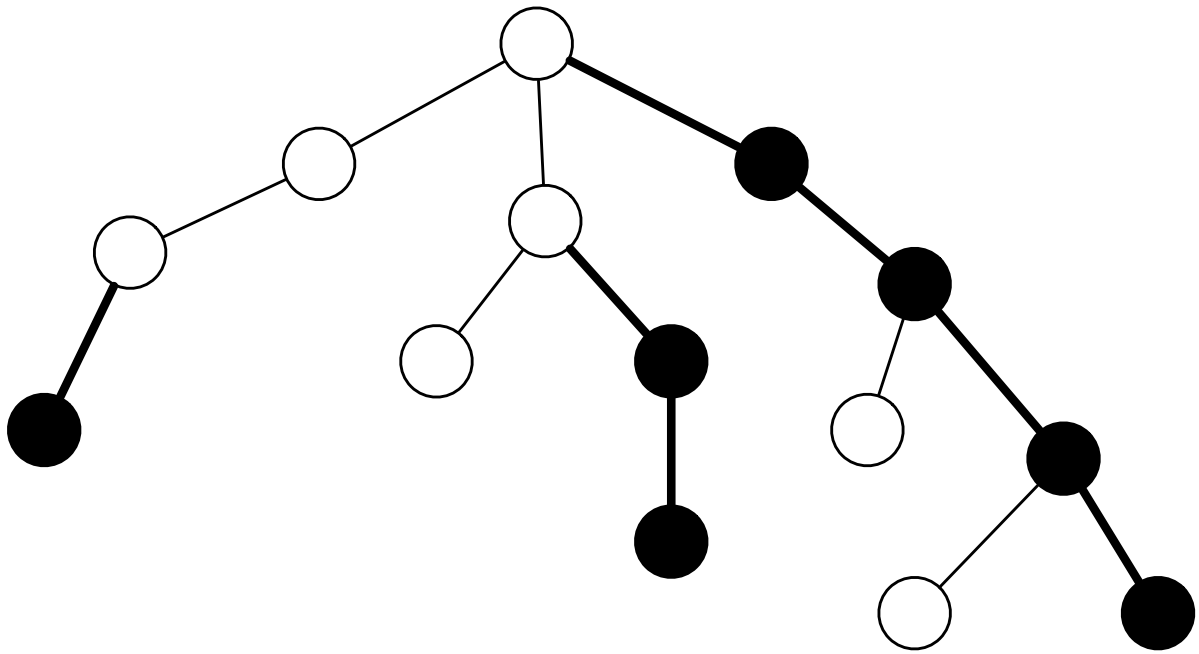
A Worst-Case Efficient Solution

- Consider the simple $O(n_P n_T)$ time and space solution (assume many leaves).
- First, we reduce space by modifying the traversal of T.
- Secondly, we compactly represent states.

Path decomposition

- Classify each node of T as either *heavy* or *light*.
- The root is light.
- At node y pick a child z of y of maximum size among y's children and classify z as heavy. The remaining children are light.
- Edges to light and heavy children are *light edges* and *heavy edges*, respectively.

Example



Light-depth

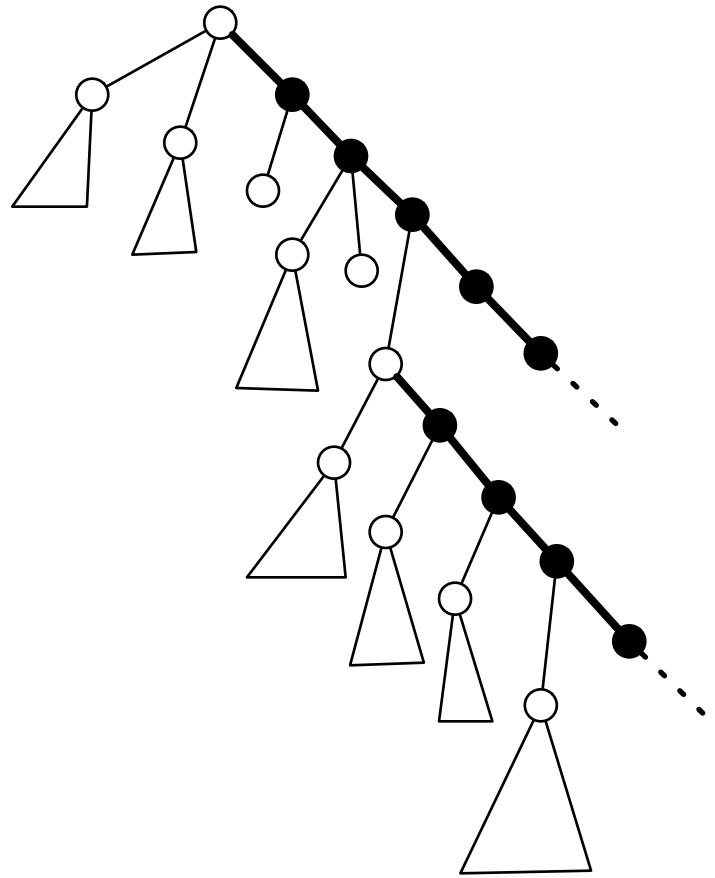
- Removing light edges partitions T into *heavy-paths*.
- $\mathrm{lightdepth}(y)$ is the number of light edges on $\mathrm{path}(y)$.
- [HT84] For any node y in T

$$\mathrm{lightdepth}(y) \leq \log n_T + O(1)$$

Heavy-path traversal

- At node y with heavy child z:
- Recursively visit the light children of y.
- Compute X_z and *discard* X_y.
- Recursively visit z.

Example



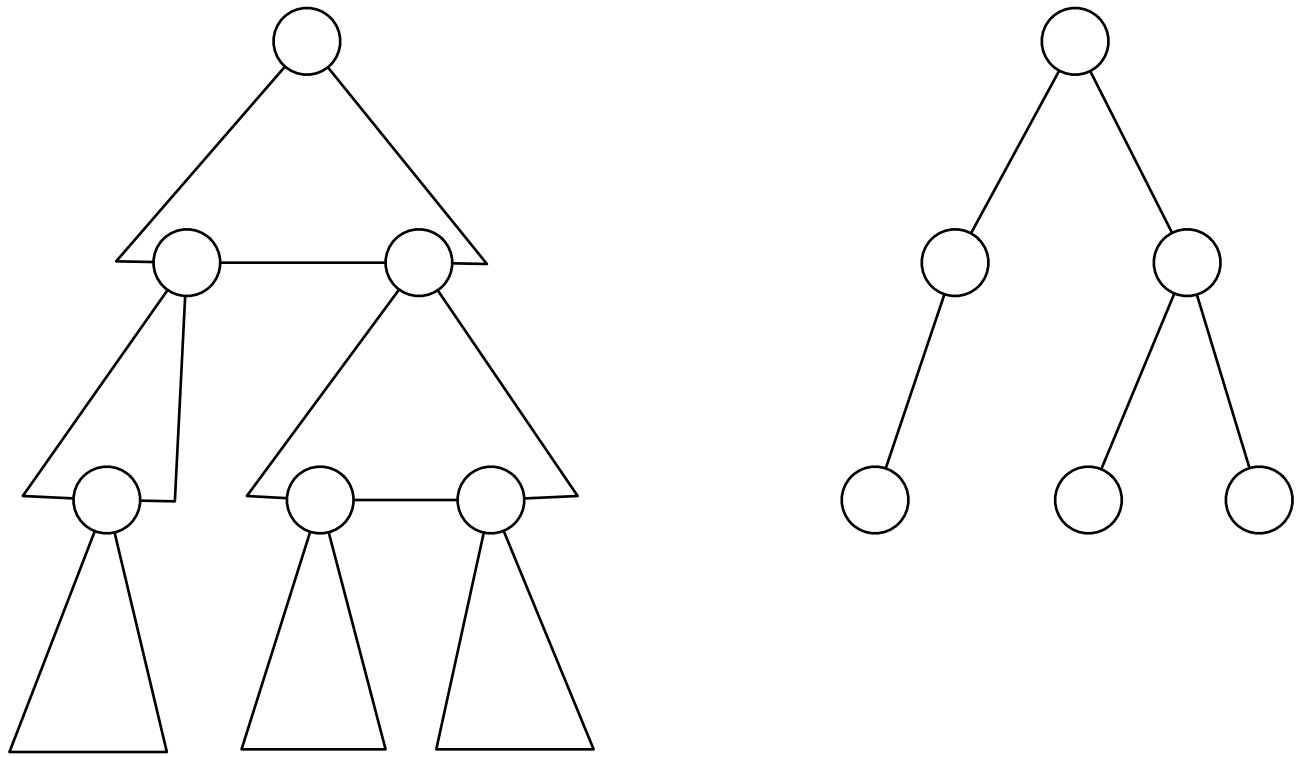
Heavy-path traversal

- At node y we store a single state for each of the heavy paths in $\mathrm{path}(y)$.
- At any node y we store $\mathrm{lightdepth}(y) + 1$ states.
- $O(n_P \log n_T)$ space and $O(n_P n_T)$ time.

Compact State Representation

- Recall that states are subsets of nodes in P.
- Divide P into a set of *micro-trees* each containing at most $s = \Theta(\log n_T)$ nodes.
- Two micro-trees share at most one node, which must be the root in at least one of the trees.
- The roots of the micro-trees induce a macro-tree containing $O(\lceil n_P/s \rceil)$ nodes.

Example



Compact State Representation

- Represent the state in each micro-tree using a bit vector of length s. Since $s = \Theta(\log n_T)$ the vector fits in $O(1)$ words of memory.
- The total space for a state is $O(\lceil n_P/s \rceil)$.

Updating the state

- With this encoding and a number of extra ideas ("Four Russian Technique") it is possible to simulate a step in the simple algorithm on a micro-tree in constant time.
- Using this encoding and the macro tree we can simulate a step in the simple algorithm on P in $O(\lceil n_P/s \rceil)$ time.

Complexity

- Total space: $O(\lceil n_P/s \rceil \log n_T + n_P + n_T)$
- Total time: $O(\lceil n_P/s \rceil n_T + n_P \log n_P)$
- Plugging in $s = \Theta(\log n_T)$ gives $O(n_P + n_T)$ space and running time:

$$O\left(\frac{n_T n_P}{\log n_T} + n_T + n_P \log n_P\right)$$

Thank you!