The Tree Inclusion Problem: In Optimal Space and Faster

Philip Bille
Inge Li Gørtz

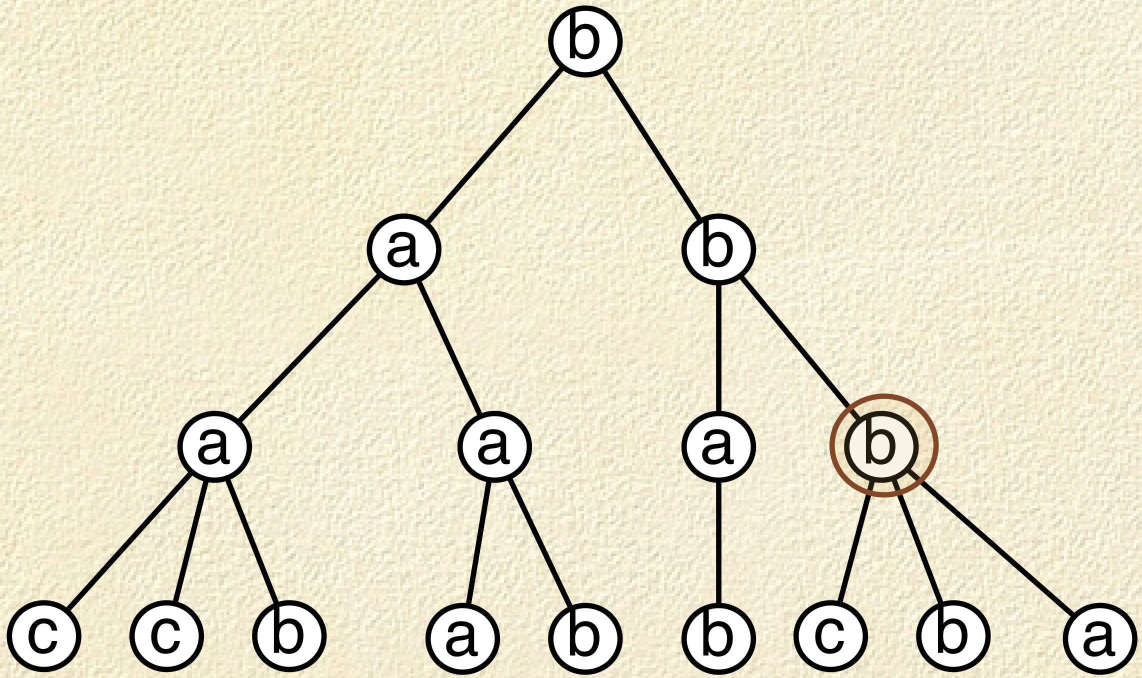
Basic setup

Trees are labeled, rooted, and ordered.

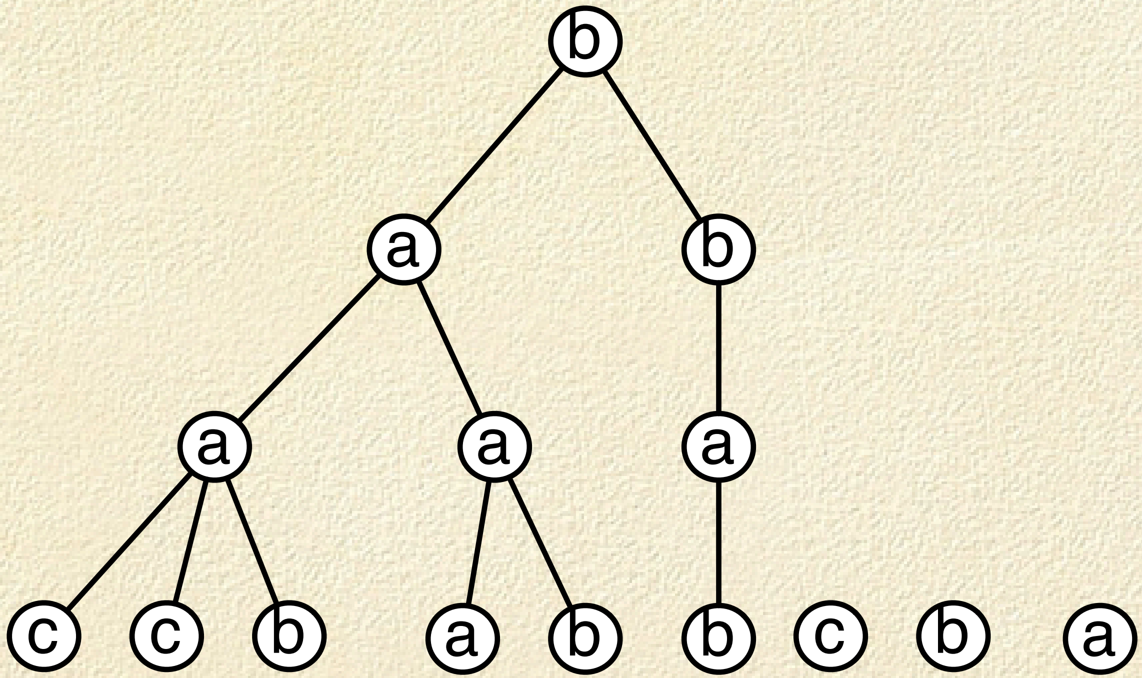
- **Rooted**: A specific node is designated as the root of the tree.
- **Labeled**: Each node is assigned a *label* from some alphabet Σ.
- **Ordered**: There is a left-to-right order among siblings.

We compare trees by *deleting* nodes.

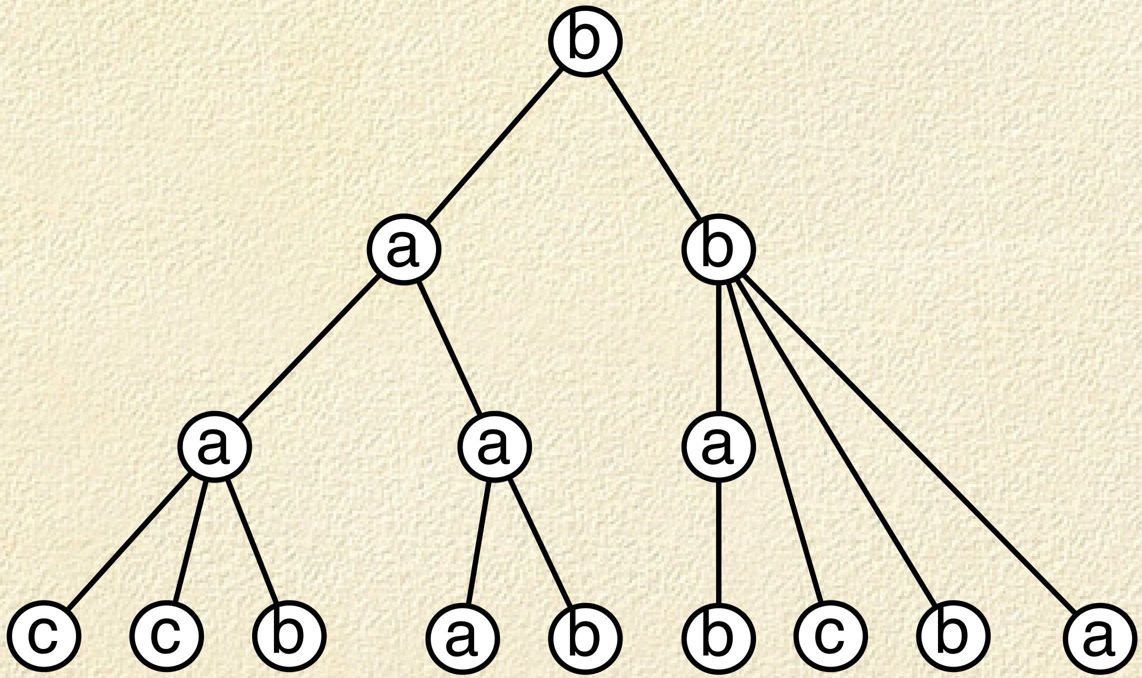
Delete a node



Delete a node



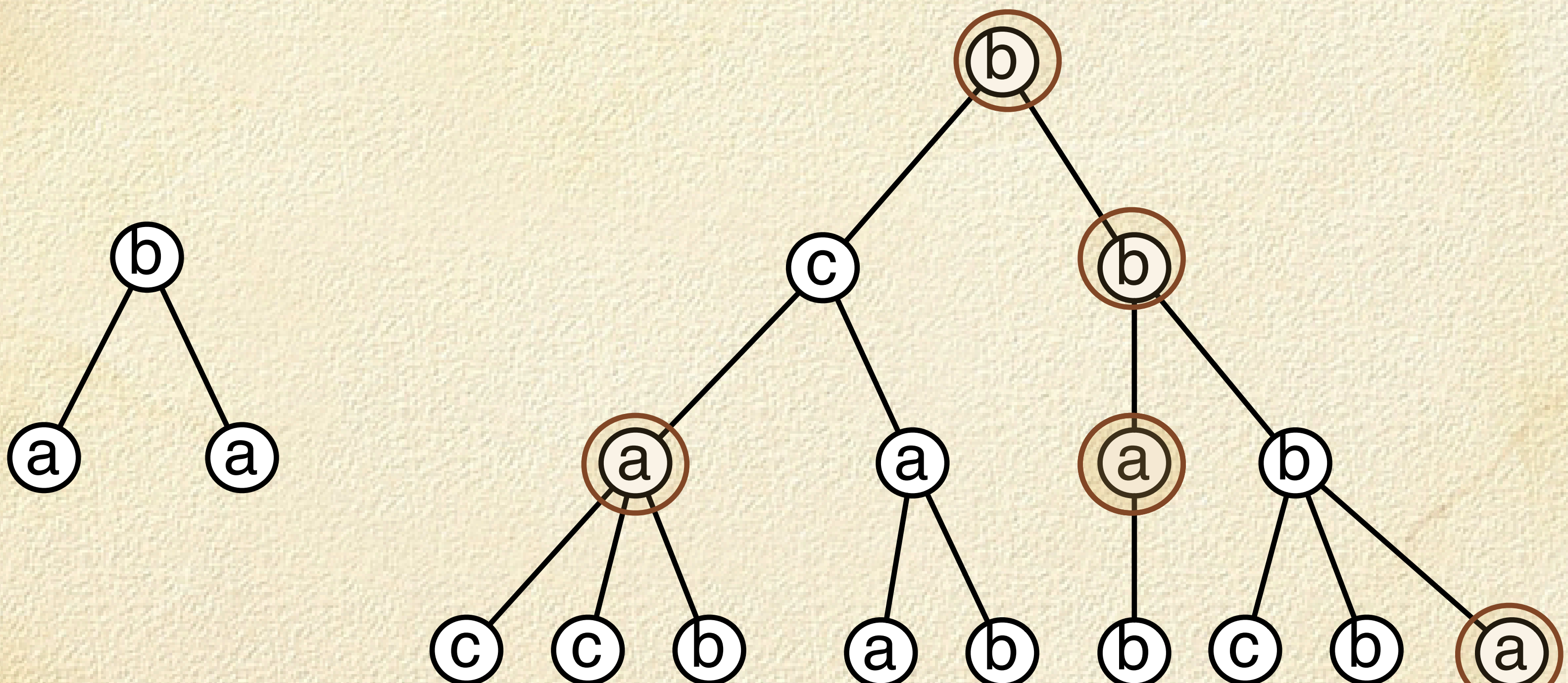
Delete a node



Tree Inclusion

- P is *included* in T if P can be obtained from T by deleting nodes in T.
- P is *minimally included* in T if P is not included in any subtree of T.
- The *tree inclusion problem* is to decide if P is included in T, and if so, compute all subtrees of T which minimally includes P.

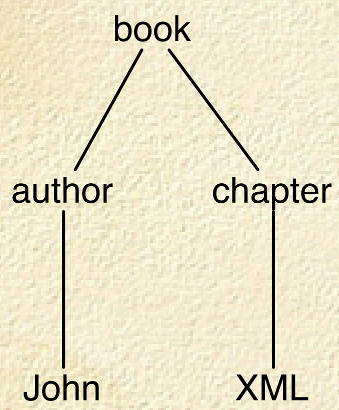
Example

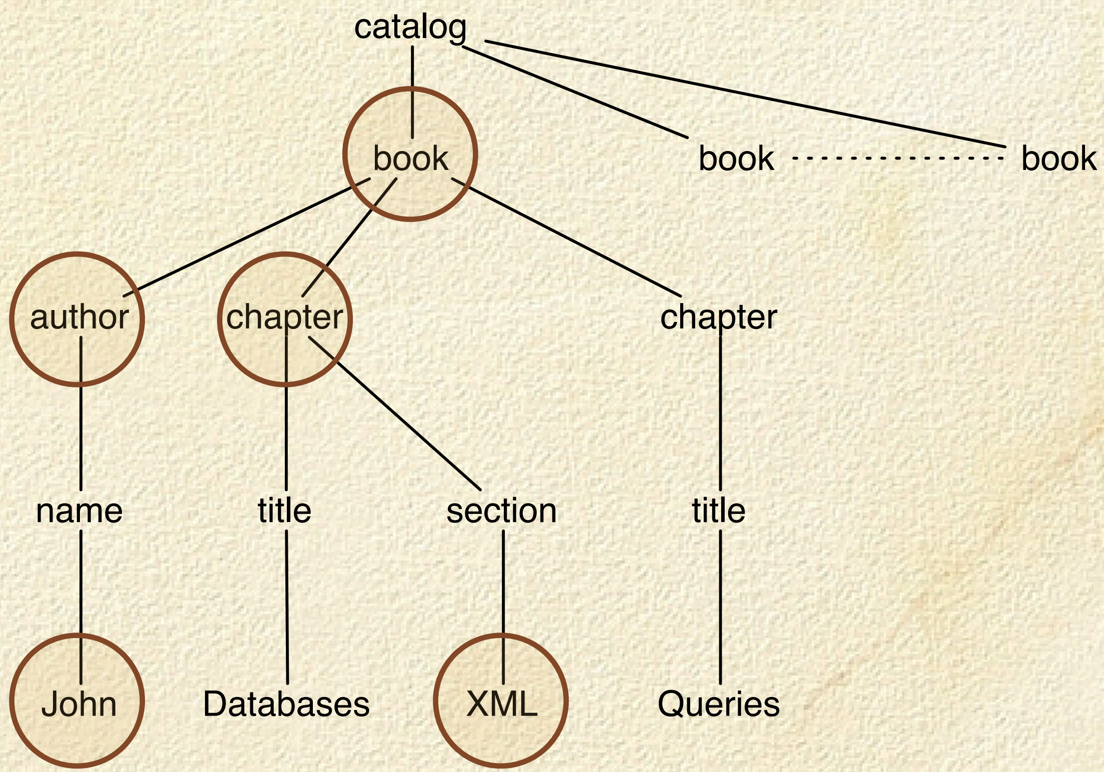


Results

Time	Space	Reference
$O(n_P n_T)$	$O(n_P n_T)$	[KM92]
$O(l_P n_T)$	$O(l_P \min(d_T, l_T))$	[Che98]
$O(l_P n_T)$	$O(n_P + n_T)$	This paper
$O(n_P l_T \log\log n_T)$		
$O(\frac{n_P n_T}{\log n_T})$		

XML example





Query: "Find all books written by John with a chapter that has something to do with XML".

Practical implications

- Space reduction from quadratic to linear:
  - Possible to query significantly larger XML databases.
  - Faster query time since more computation can be kept in main memory.

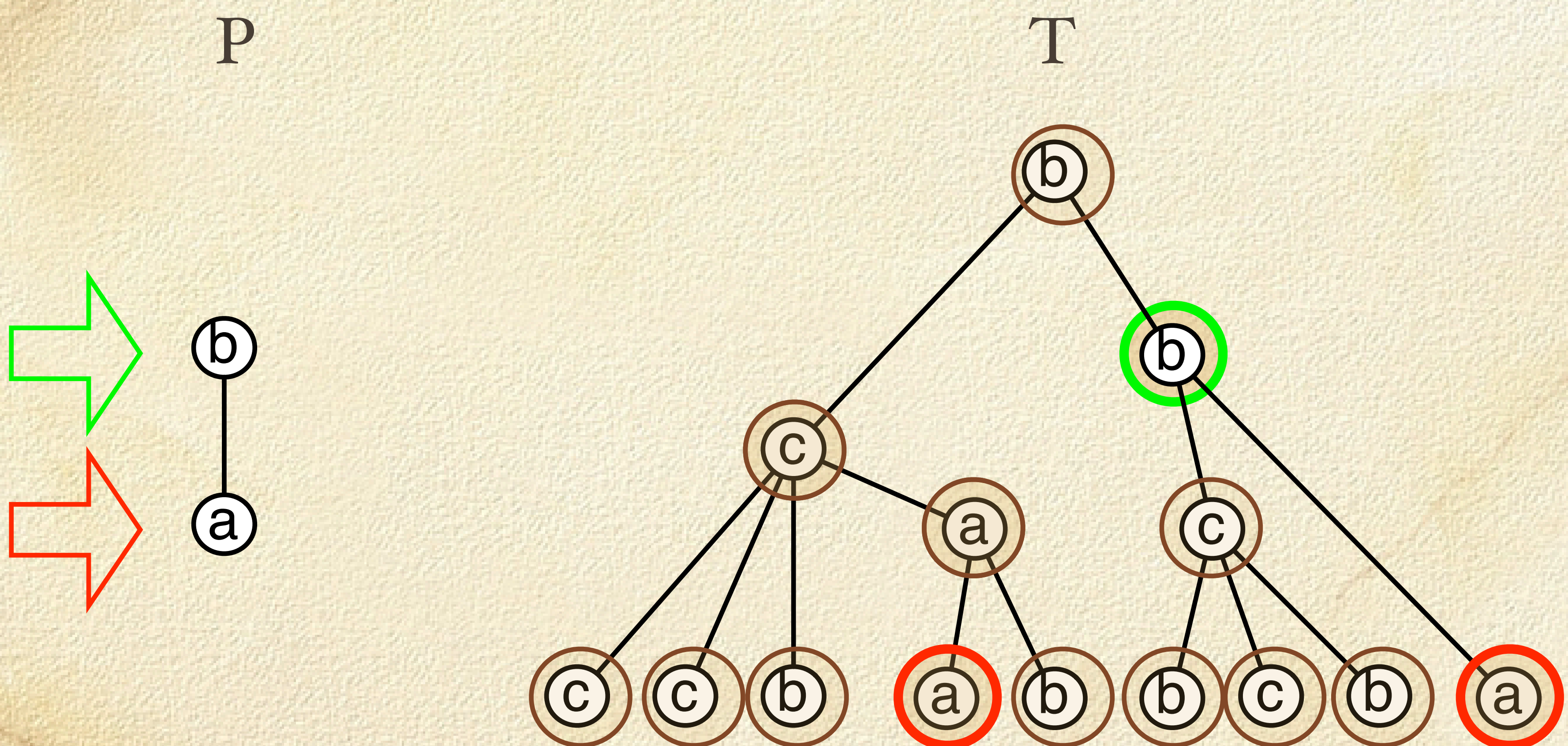
Embeddings

An injective function from the nodes of P to T is an *embedding* if:

- label(v) = label(f(v)),
- v is ancestor of w iff f(v) is an ancestor of f(w),
- v is to the left of w iff f(v) is to the left of f(w).

P is included in T iff there is an embedding from P to T.

A simple case: P is a path

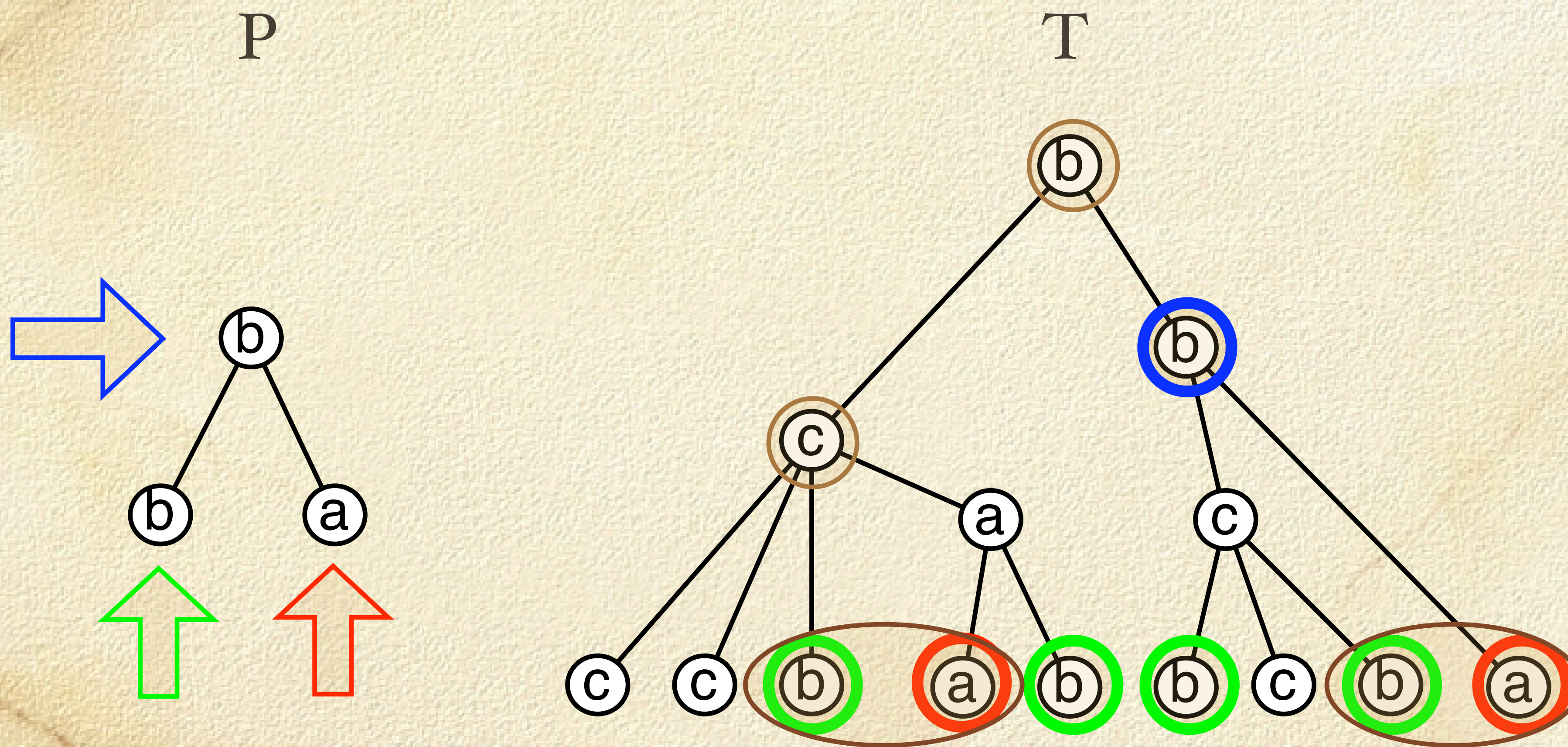


=Active set =Root of min. subtree including

Complexity

- At each step of the algorithm the active set "moves up".
- Each parent pointer in T is traversed a constant number of times.
- Using a simple data structure and exploiting the ordering of the nodes we get a total running time of $O(n_T)$.

When P is not a path:



Complexity

- Let Δ denote the set of all *leaf-to-root* paths in P.

- Running time is by bounded by the time used to solve the tree inclusion problem on each path in Δ. In total:

$$\sum_{\delta \in \Delta} O(n_T) = O(l_P n_T)$$

- Space is $O(n_P + n_T)$.

Alternative algorithm.

- Reconsider the case when P is path:
- Let *firstlabel(v,l)* denote the nearest ancestor of the node *v* in T with label *l*.
- At each step we “essentially” compute *firstlabel(v, l)* for each *v* in the active set.

Alternative algorithm

- Idea: Use a fast data structure supporting *firstlabel* queries. Known as the *tree color problem*.

Lemma [Dietz89] For any tree T there is a data structure using $O(n_T)$ space, $O(n_T)$ expected preprocessing time which supports *firstlabel(v,l)* in $O(\log \log n_T)$ time.

Complexity

- For each node in P there is an active set and for each node in this active set we have to compute a *firstlabel* query.
- Size of active set is at most l_T. Total time:

$$O(n_P l_T \log\log n_T)$$

- Space is still $O(n_P + n_T)$.

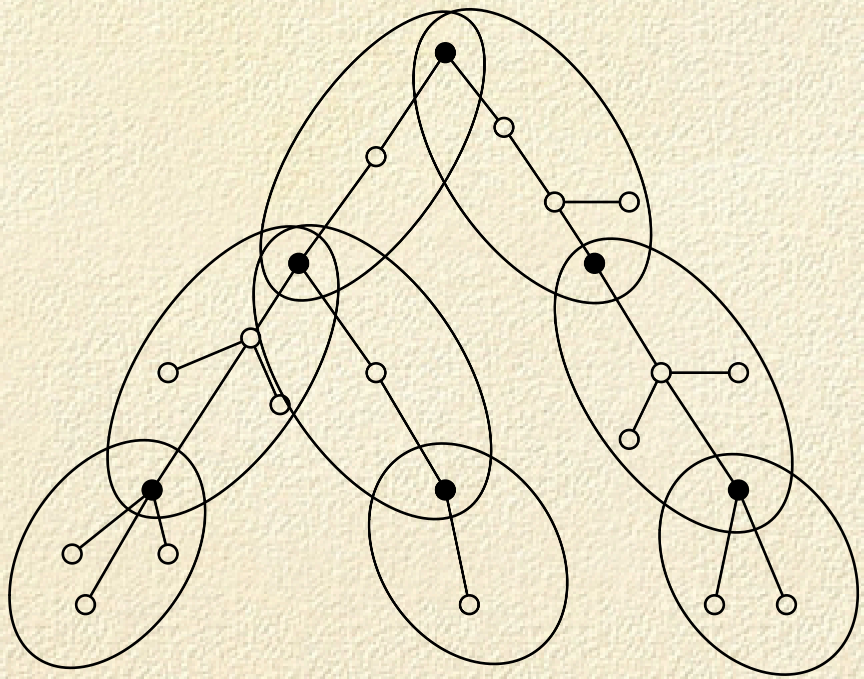
Improving the worst-case

- Divide T into $O(n_T/\log n_T)$ *micro trees* of size $O(\log n_T)$ which overlap in at most 2 nodes using a clustering technique from [AHT97].

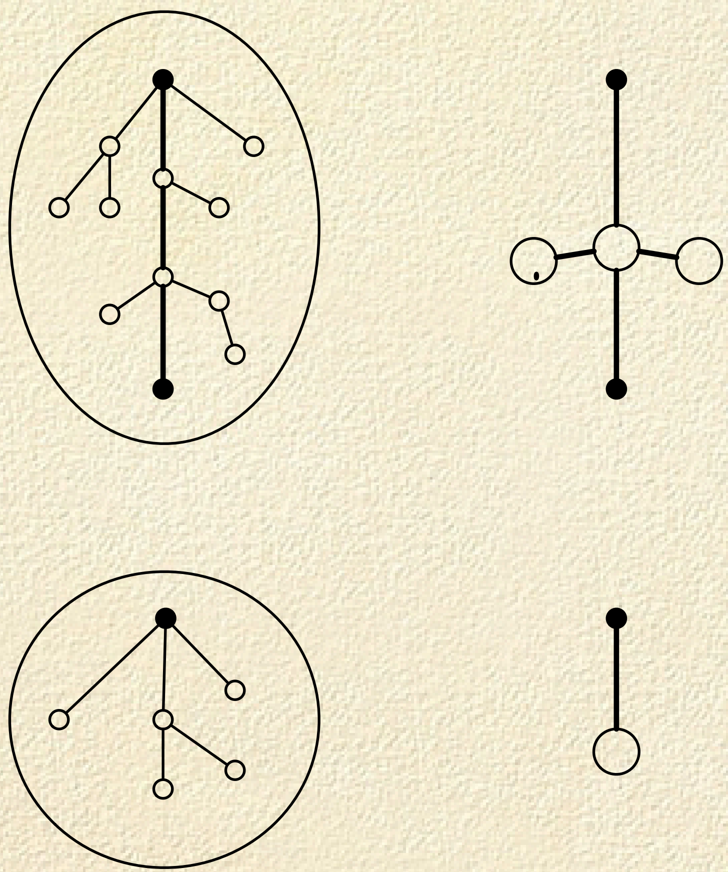
- Each micro tree is represented by a constant number of nodes in a *macro tree* and connected according to the overlap in the micro trees.

- Essentially we do a “four russian” technique to get the speedup.

Clustering example



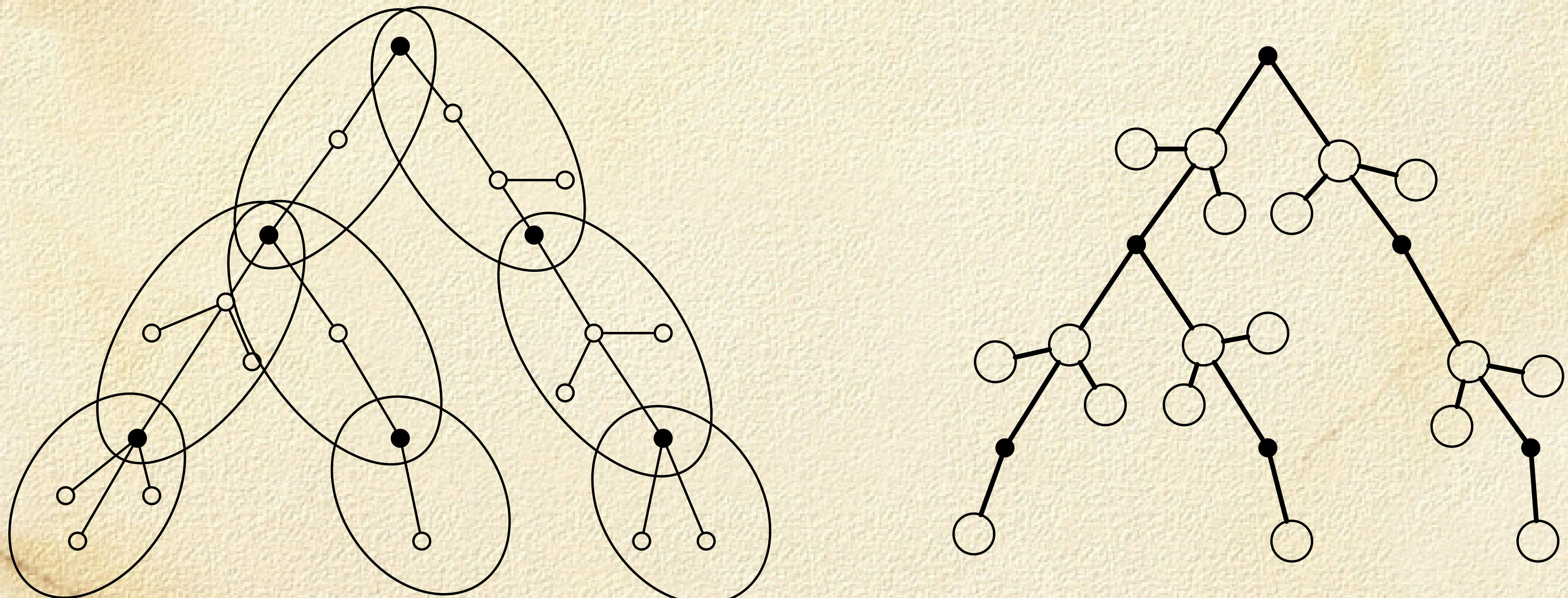
The Macro Tree



Properties of macro trees

- Each node x in the macro tree induces a micro tree (or forest) denoted I(x).
- The label of x is the set of labels in I(x).

Macro Tree



MM-node sets

- To a represent a node set V in T we represent for each node x in the macro tree the subset of V in I(x).
- Each such subset is represent by a bitstring using a constant number of words.
- V is represented in $O(n_T/\log n_T)$ space.

Firstlabel on mm-node set

- How can we compute *firstlabel*(M, l) for a mm-node set M?
- Use preprocessing to compute firstlabel on the micro trees fast.
- Combine the results using the macro tree to get solution.

Handling micro trees fast

- Compute *firstlabel*(S,X, l) for a set of nodes X in a micro tree S. S and X are represented compactly in bitstrings.
- For *all possible* S and X precompute the following:
    - *ancestor*(S, X): All ancestors of X in S.
    - *deep*(S,X): Subset of X obtained by removing nodes that are ancestors of another node in X.

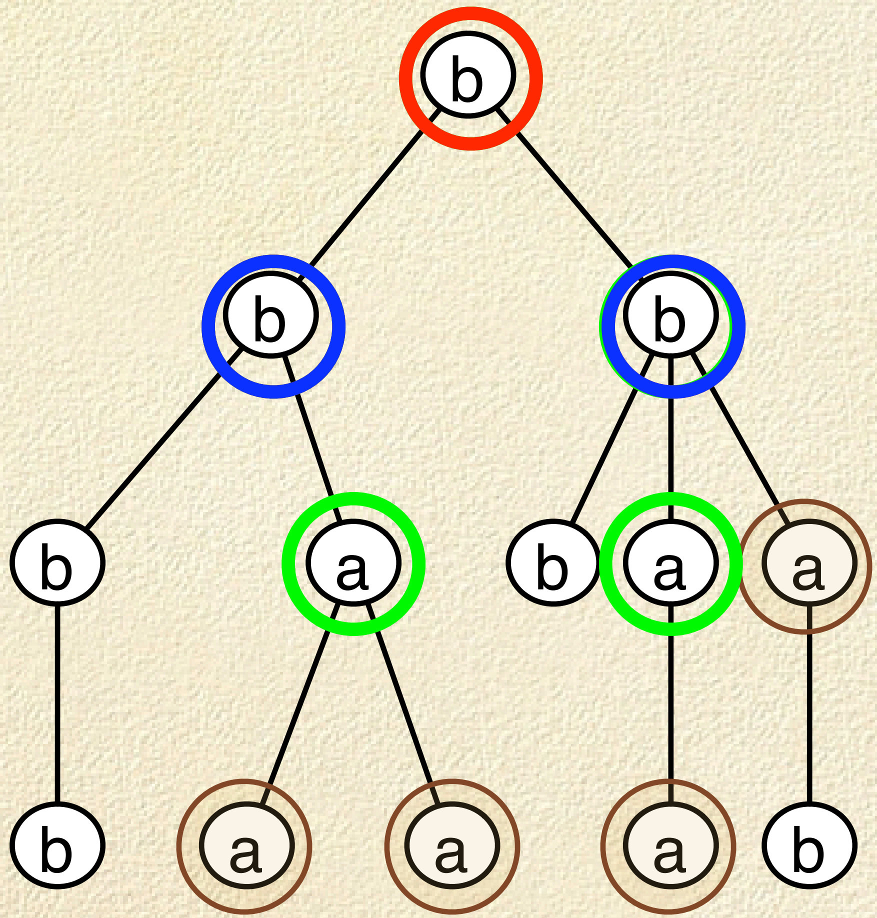
Handling micro trees fast

- Since S and X are of logarithmic size *ancestor* and *deep* can be computed (using dynamic programming) in linear time and space.
- Tabulating all inputs gives linear time lookup.

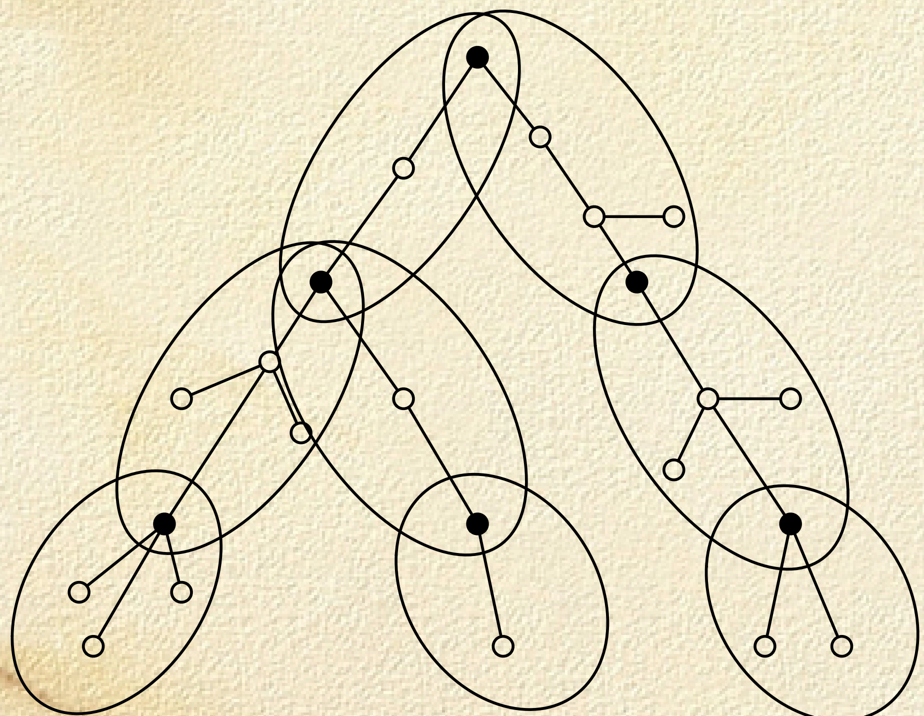
Handling micro trees fast

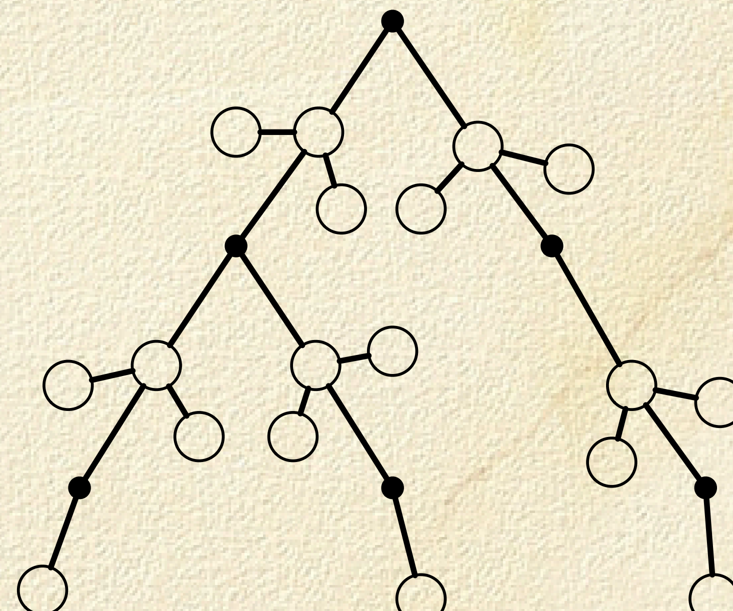
- For each micro tree S in T compute and store a dictionary (indexed by labels) containing:
  - *mask*(l): The set of nodes in S with label l.
- With perfect hashing this gives total linear space, linear expected preprocessing time, and constant lookup time.

firstlabel(S,X,b)



Firstlabel not in S?





General solution

- Compute *firstlabel* on each of the micro trees in M.
- This gives a *firstlabel* query on the macro tree which is solved in linear time on the macro tree.

Complexity

- Time for *firstlabel* becomes $O(n_T / \log n_T)$.
- Similar bound for all other needed manipulation of node sets.
- Total time becomes $O(\frac{n_P n_T}{\log n_T})$.
- Space is still $O(n_P + n_T)$.

Conclusion

Theorem 1 For tree P and T the tree inclusion problem can be solved in time

$$O(\min(l_P n_T, n_P l_T \log\log n_T, \frac{n_P n_T}{\log n_T}))$$

and space $O(n_P + n_T)$.

- International Society for **C**omputer **A**ssisted **O**rthopaedic **S**urgery.
- **C**lub of **A**mateurs in **O**ptical **S**pectroscopy.
- **C**ollective for **A**lternative **O**rganisation **S**tudies.
- **C**oastal **A**laskan **O**bserving **S**ystem.
- **C**ommunity **A**ssembled **O**perating **S**ystem.
- **C**ode for **A**daptive **O**ptics **S**ystems.