Top Tree Compression of Tries

Philip Bille, Pawel Gawrychowski,
Inge Li Gørtz, Gad M. Landau,
Oren Weimann

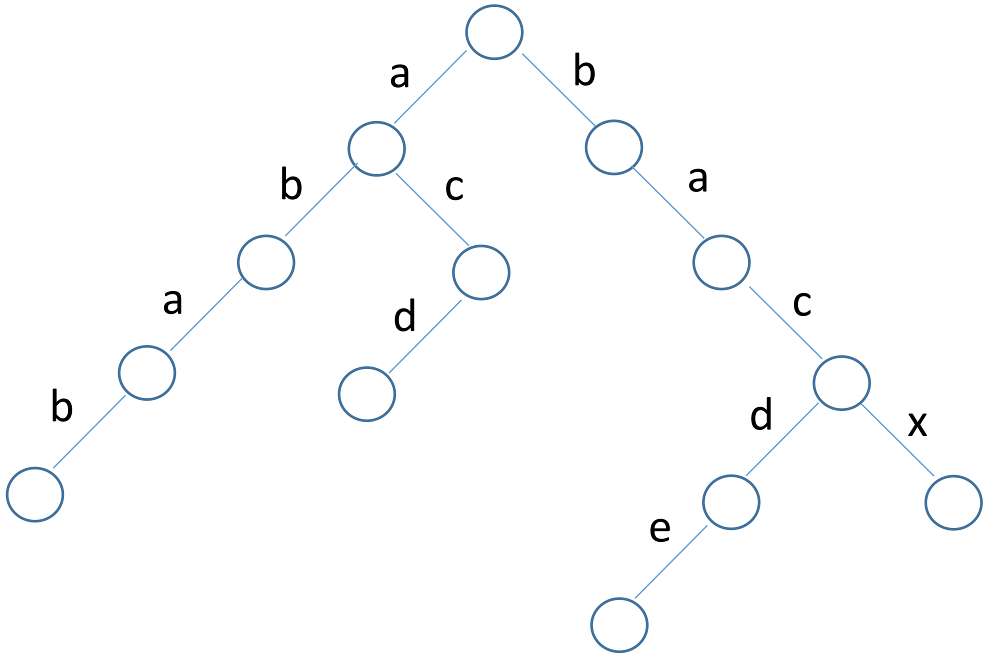

PR24
RADIO

Goals

Compressed representation of tries

A Trie (Fredkin 1960) - *k* strings

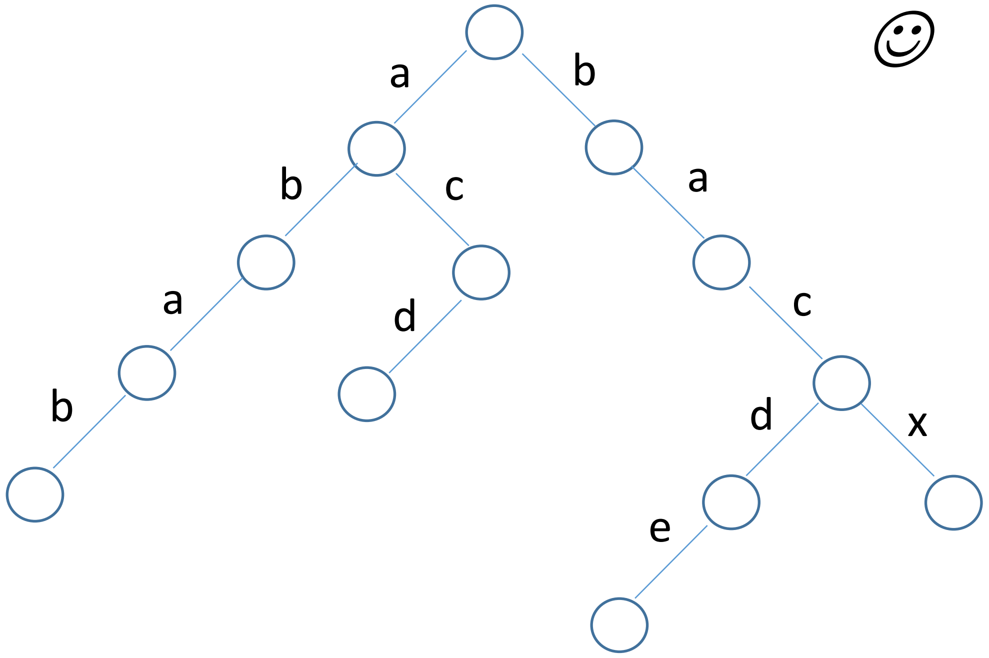
abab acd bacde bacx



Goals

Compressed representation of tries

Given a pattern string *P* of length *m* determines if P is a prefix of one of the strings



aba ☺

acx ☹

Results

Set of strings $S = S_1,\ldots,S_k$ of total length n

Alphabet of size σ

Pointer Machine

Compressed data structure (worst-case optimal)

size $O(n/\log_\sigma n)$

Lempel Ziv

Query time:

$O(\min(m \log \sigma, m + \log n))$ (A tight Lower Bound)

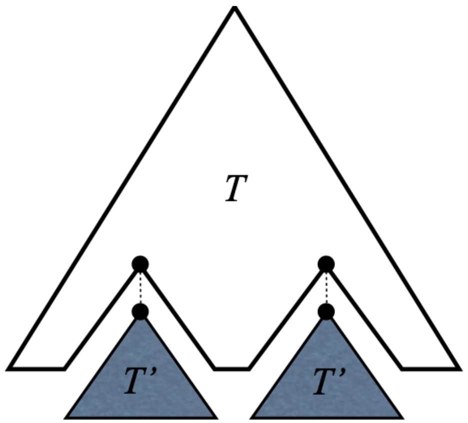
Tools

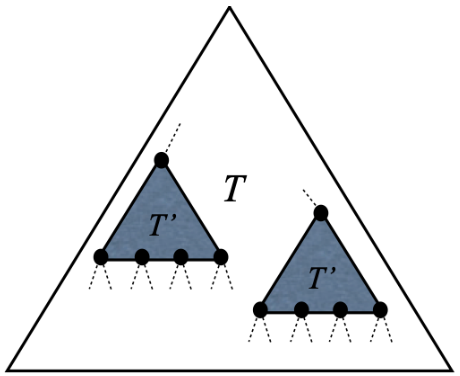
Top Trees (Alstrup, Holm, De Lichtenberg, Thorup 2005)
DAG compression of trees
Karp-Rabin Fingerprints

The Pointer Machine Model

A directed graph with bounded out-degree.
Each node contains a constant number of data fields or pointer to other nodes.
Algorithms must access the data structure by traversing the graph.

Using Repetitions to Compress Trees

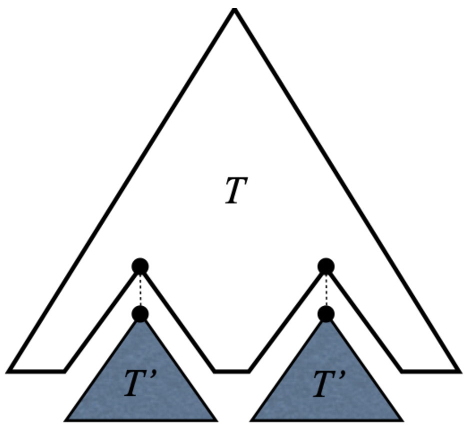




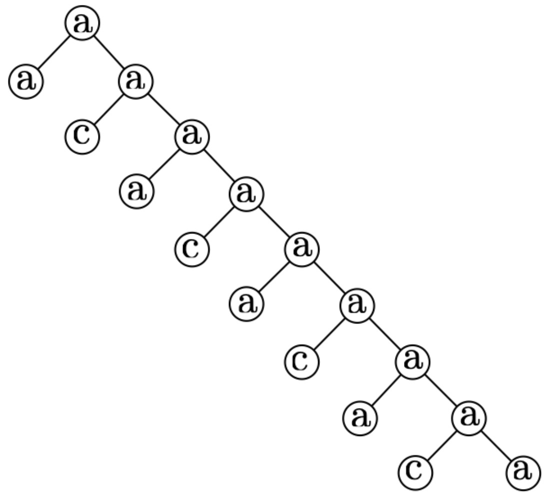
- Input. Labeled, ordered, rooted tree T with N nodes over an alphabet of size σ.
- Goal. Compress T to:
  - Take advantage of repetitions (*tree pattern repeats)*
  - Obtain good guarantees on compression ratio.
  - Support efficient navigation (access, parent, depth, height, size, LCA, ...)

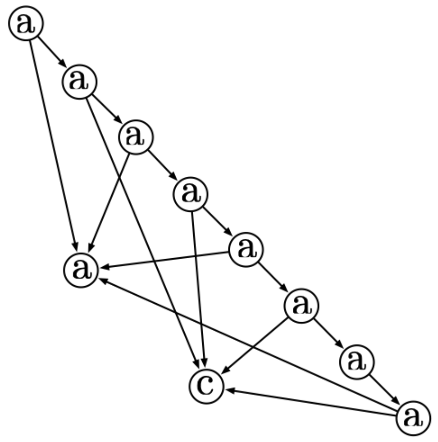
DAG Compression of Trees

- Merge *subtree repeats* into directed acyclic graph (DAG) representing T.
- Takes advantage of subtree repeats but not tree pattern repeats.



DAG Compression of Trees

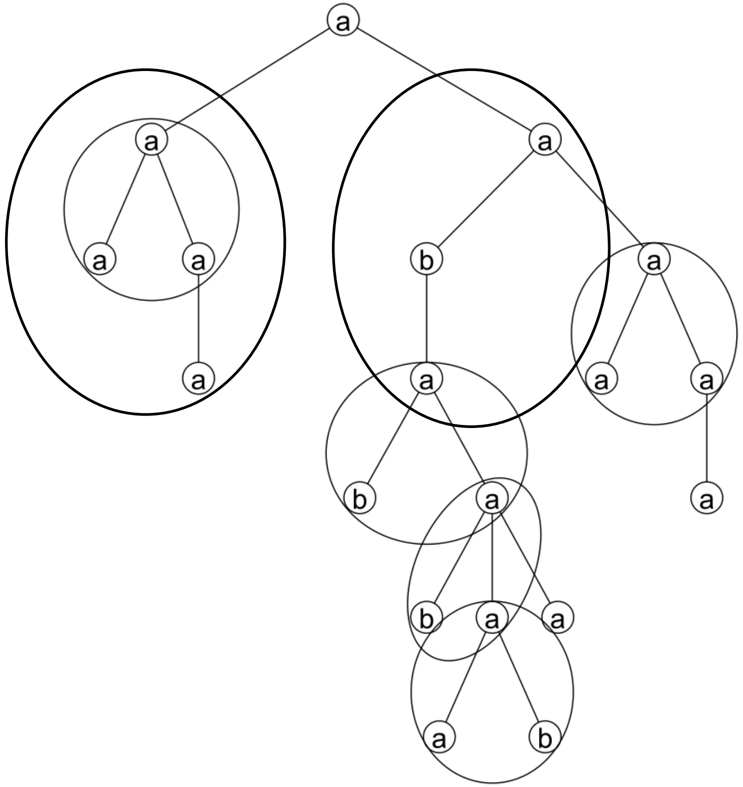


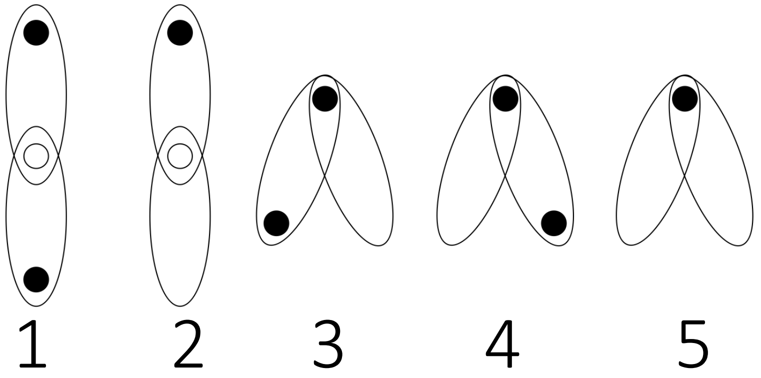


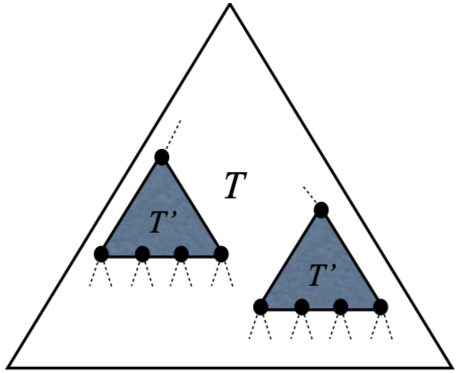
DAG Compression of Trees

- Smallest DAG is unique.
- We can build smallest DAG in O(N) time [Downey, Sethi, Tarjan 1980]
- Smallest DAG can be exponentially smaller than N, but may not compress at all.
- We can support navigational operations in O(log N) time [Bille,L., Raman, Sadakane, Satti, Weimann 2011]
- Popular for XML compression. See e.g. [Buneman, Grohe, Koch 2003] [Frick, Grohe, Koch 2003]

Clustering and Top Trees

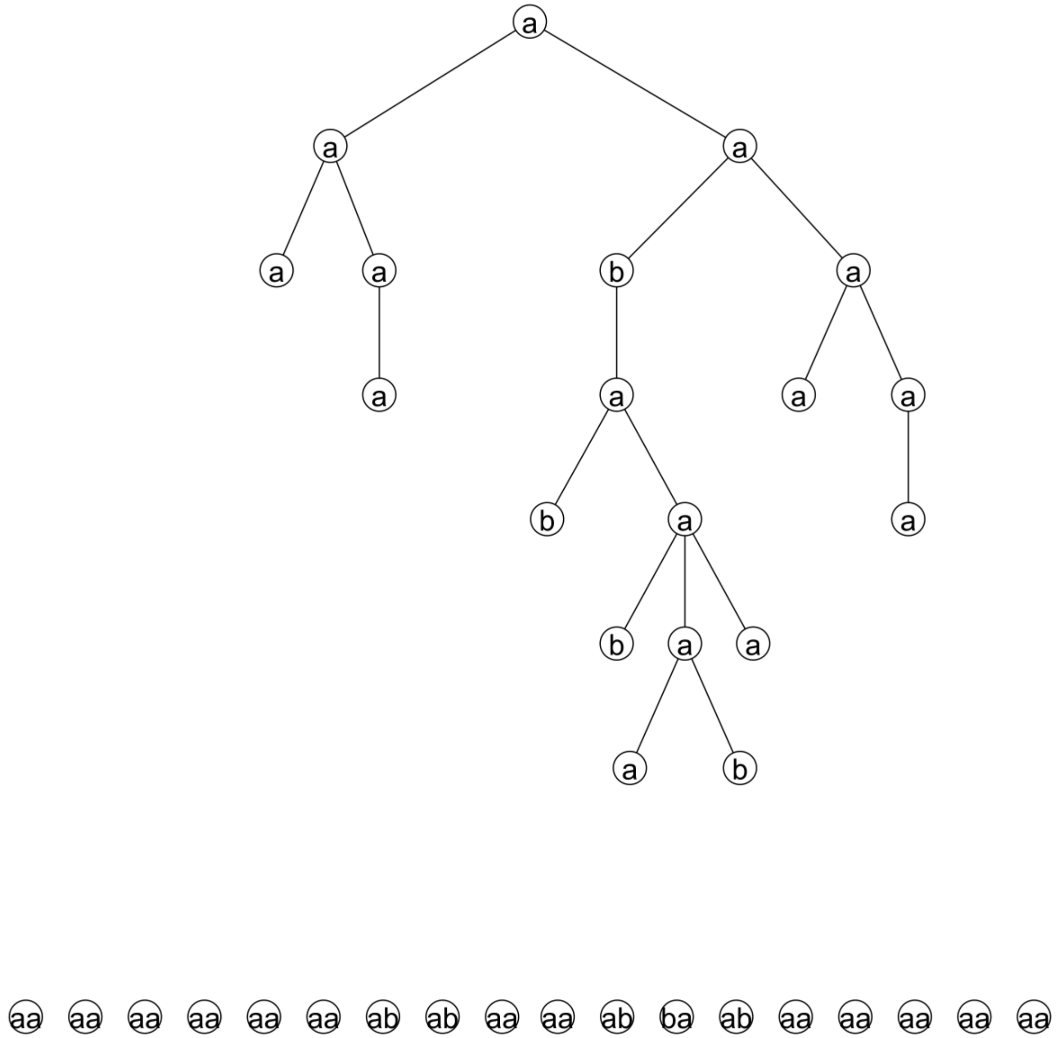




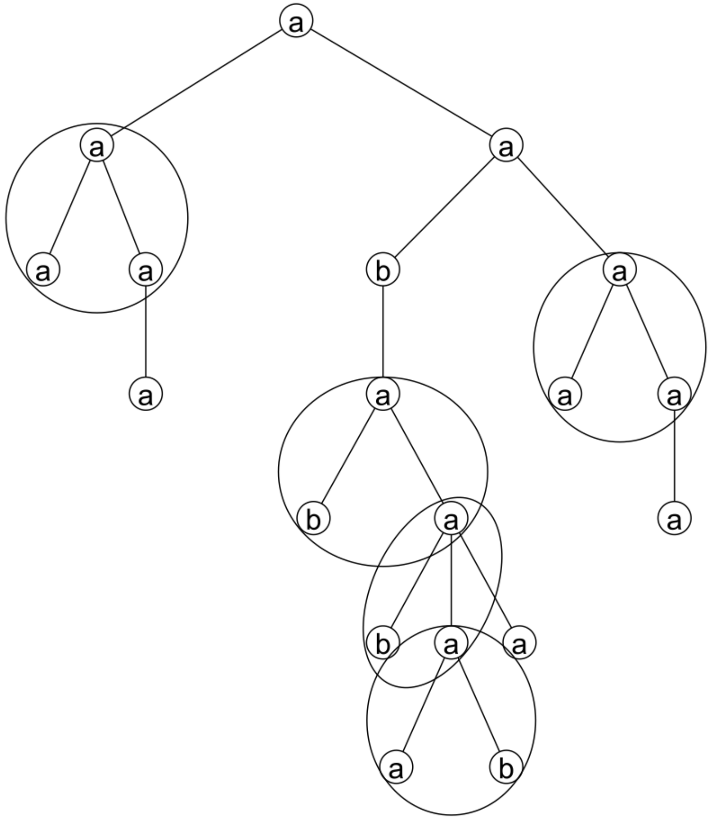


- Cluster is a connected subgraph of T, overlapping in 1 or 2 *boundary nodes.*
- 2 Clusters can be *merged* to form new cluster.
- Top tree = tree of clusters.

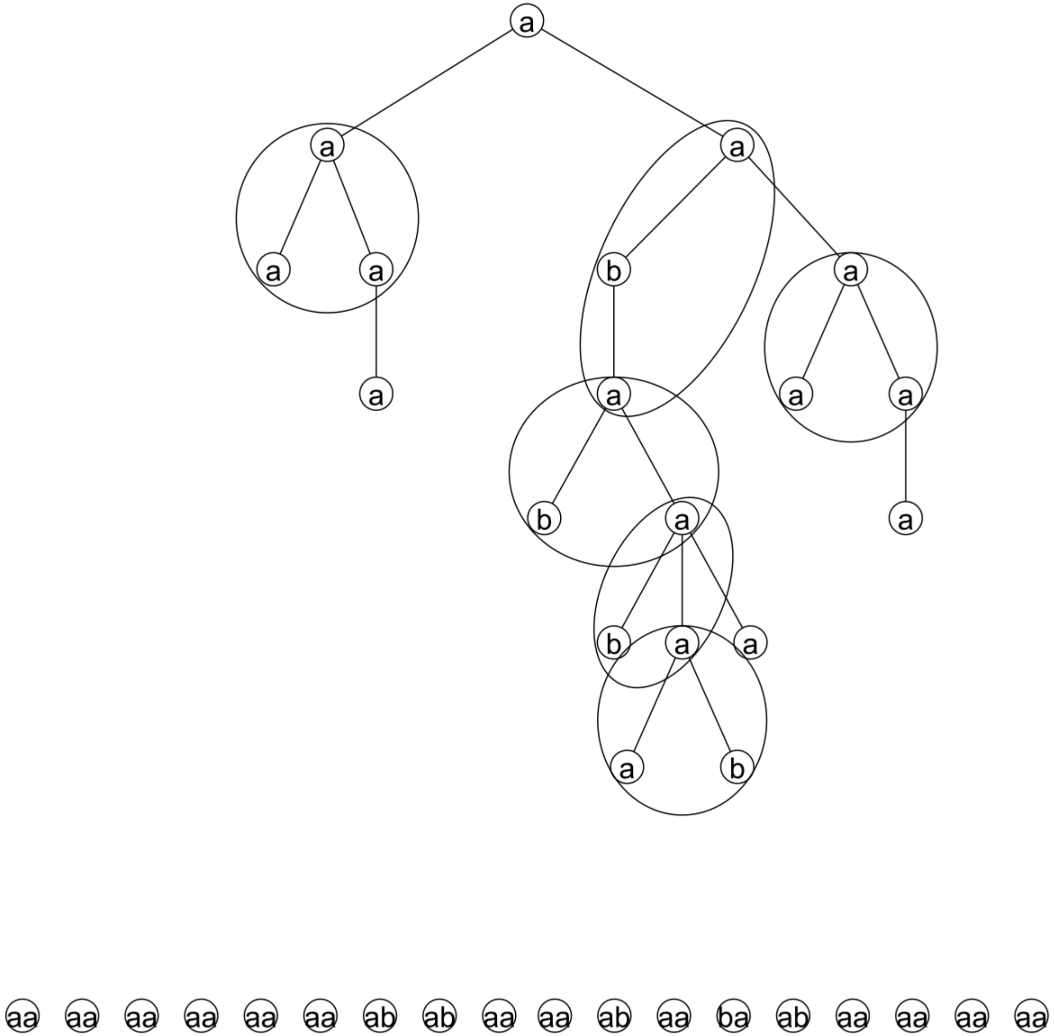
Creating the Top Tree



Iteration 1

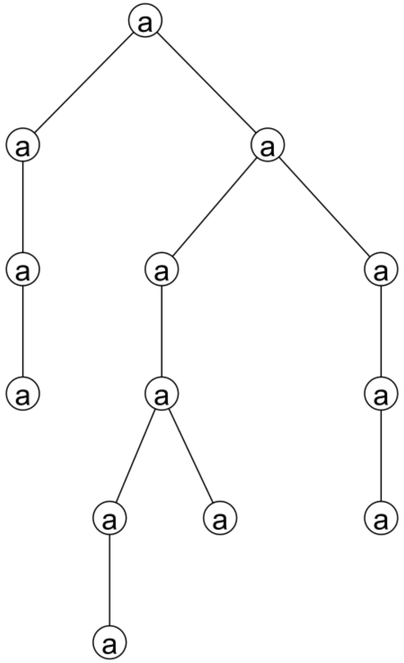


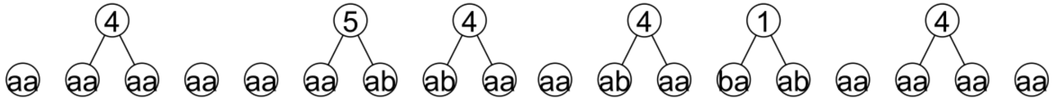
Iteration 1



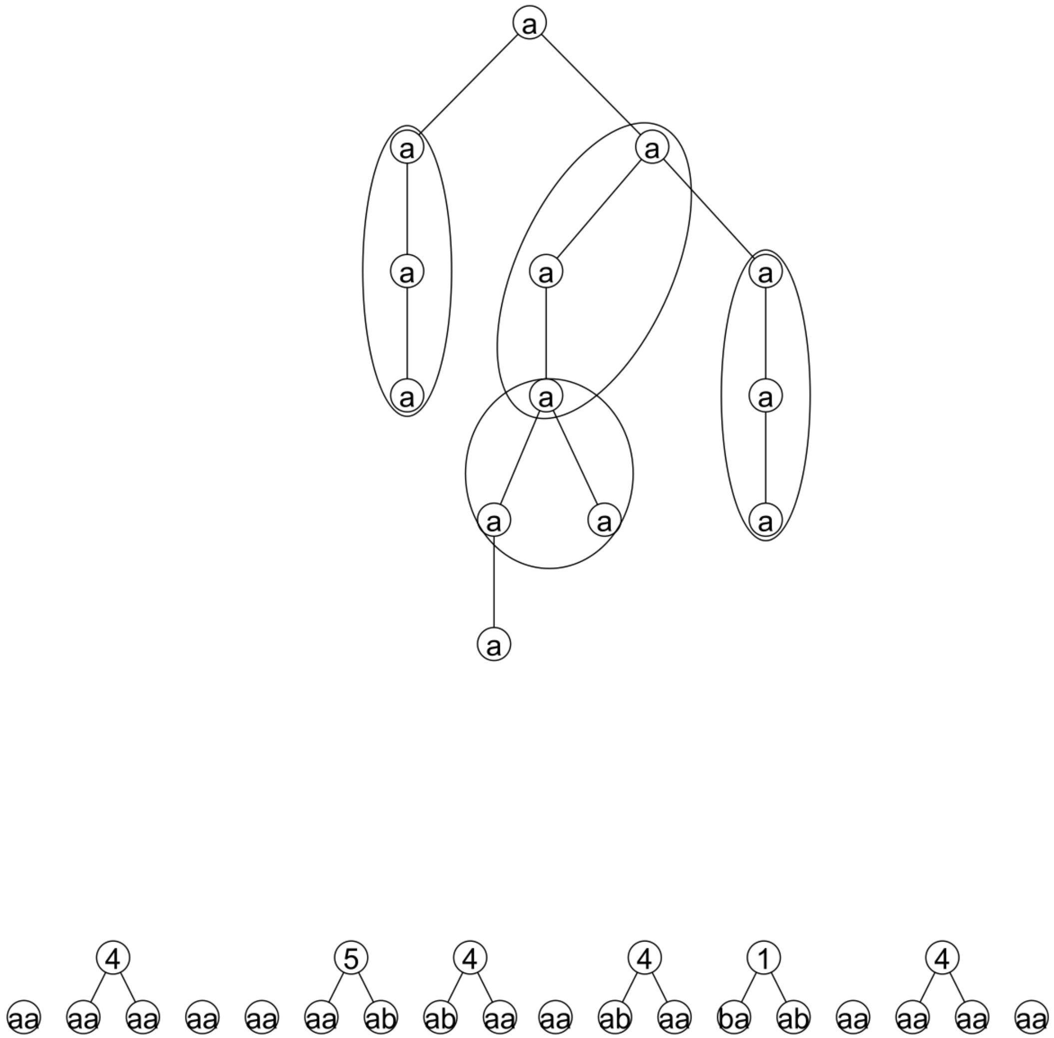
a
a
a
b
a
a
a
a
a
a
b
a
a
b
a
a
a
b
aa aa aa aa aa aa ab ab aa aa ab aa ba ab aa aa aa aa

Iteration 2

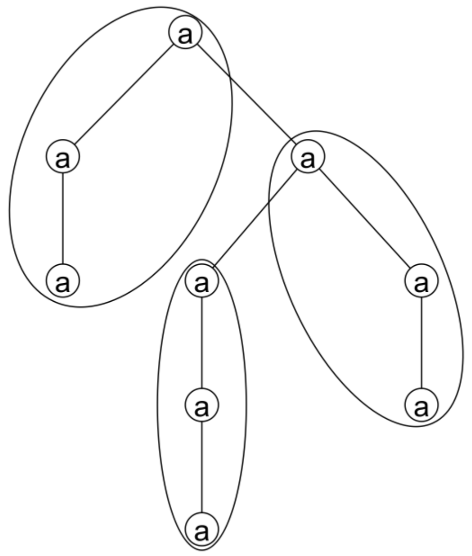

a
a a
a a a
a a a
a a a
a

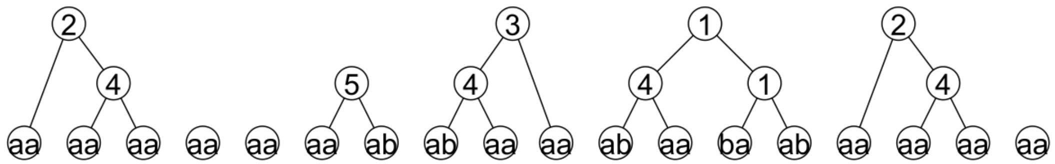

4 5 4 4 1 4
aa aa aa aa aa aa ab ab aa aa ab aa ba ab aa aa aa aa

Iteration 2

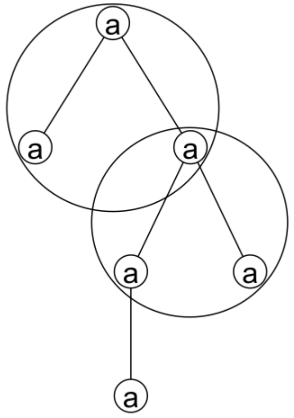

a
a a
a a a
a a a
a a a
a
4 5 4 4 1 4
aa aa aa aa aa aa ab ab aa aa ab aa ba ab aa aa aa aa

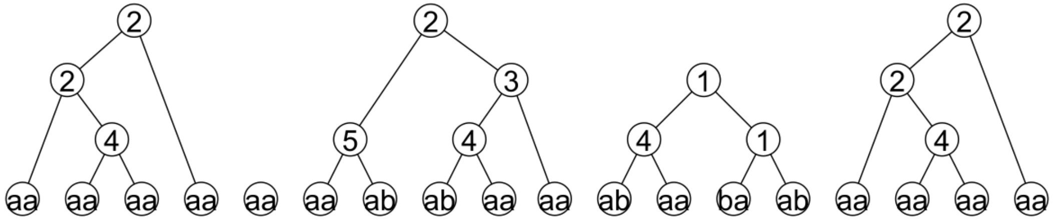
Iteration 3


a
a
a
a
a
a
a
a
a

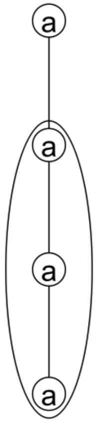

2
4
5
3
4
1
4
1
2
4
aa aa aa aa aa aa ab ab aa aa ab aa ba ab aa aa aa aa

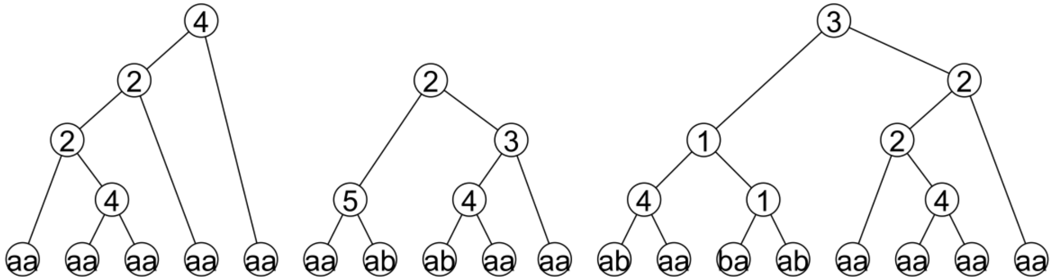
Iteration 4


a
a a
a a
a

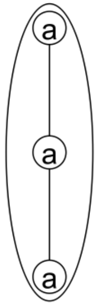

2 2 2
2 3 1 2
4 5 4 4 1 4
aa aa aa aa aa aa ab ab aa aa ab aa ba ab aa aa aa aa

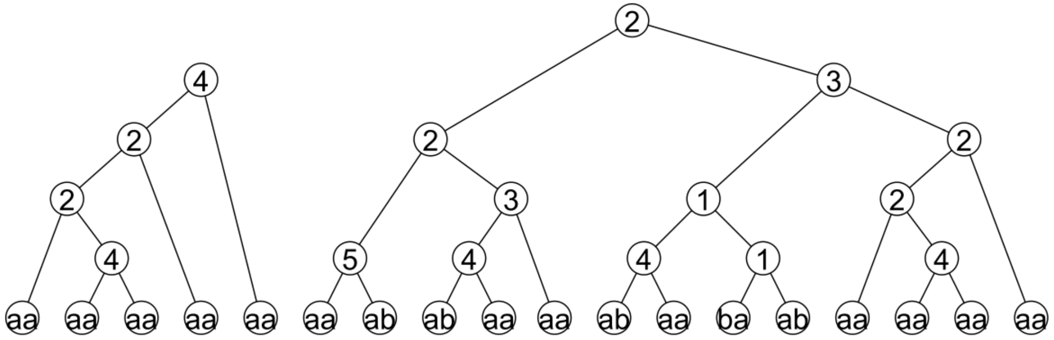
Iteration 5


a
a
a
a


4
2
2
4
aa aa aa aa aa
2
5
3
4
aa ab ab aa aa
3
1
2
4
1
2
4
ab aa ba ab aa aa aa aa

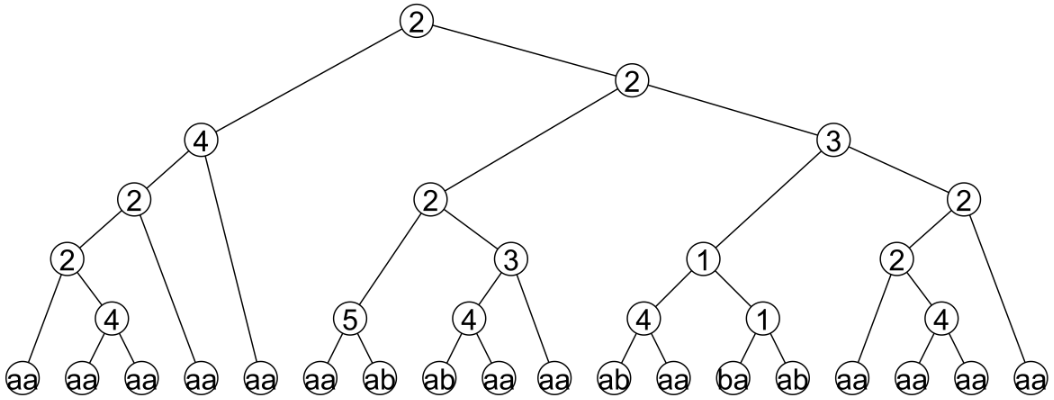
Iteration 6


a
a
a


2
4 3
2 2 2
2 2 3 1 2
4 5 4 4 1 4
aa aa aa aa aa aa ab ab aa aa ab aa ba ab aa aa aa aa

Iteration 7


a
a


2
2
4 3
2 2 2
2 2 3 1 2
4 5 4 4 1 4
aa aa aa aa aa aa ab ab aa aa ab aa ba ab aa aa aa aa

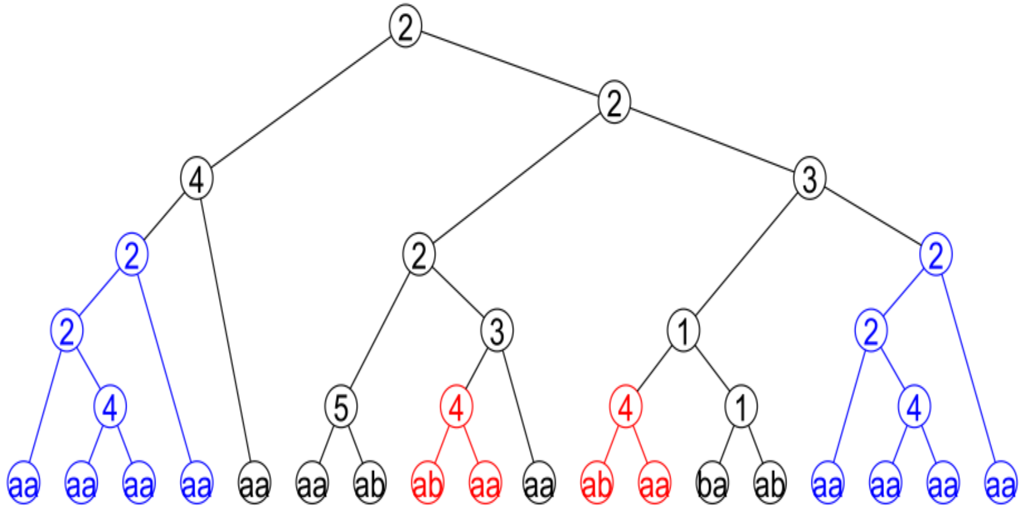
Top Tree Properties

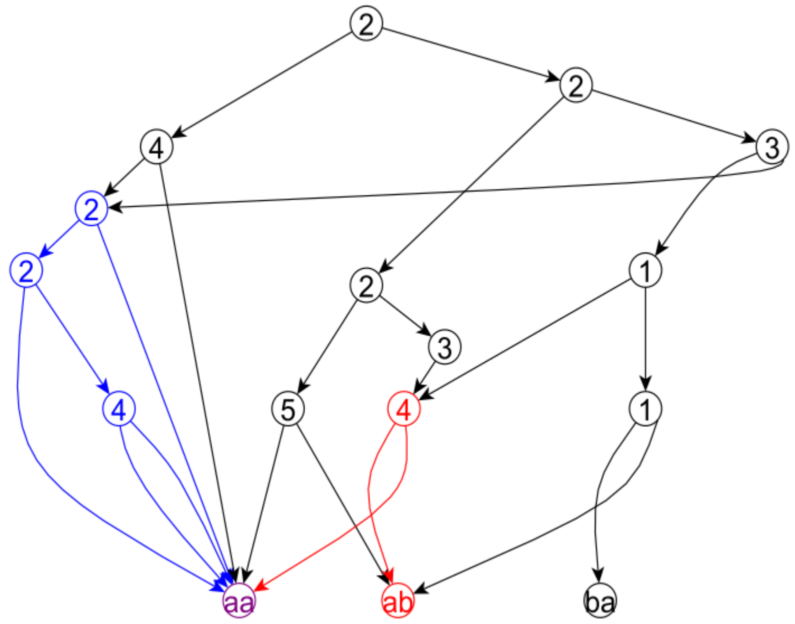
- Top tree is a binary tree.
- Clusters size increase at each level by a factor of at most 2.
- Constructing and size of the top tree is O(N), its height is O(log N) (Alstrup et al.).

Top Tree Compression

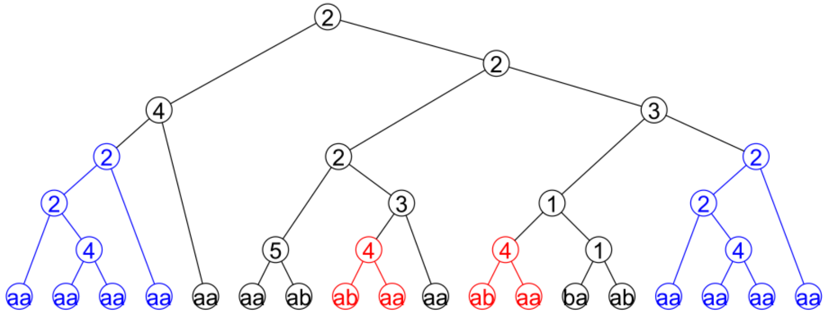
- DAG compress top tree
- Top tree compression may be viewed as *transformation* of the input tree into another tree (which compresses well and supports fast navigation).

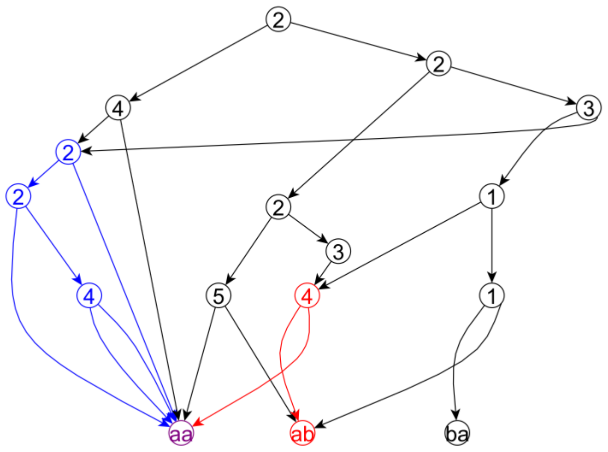
Top DAG





Top DAG





Top DAG has size at most $O(N / \log_\sigma N)$. (Dudek and Gawrychowski)

Intuition.

Identical clusters in top tree are merged in top DAG.

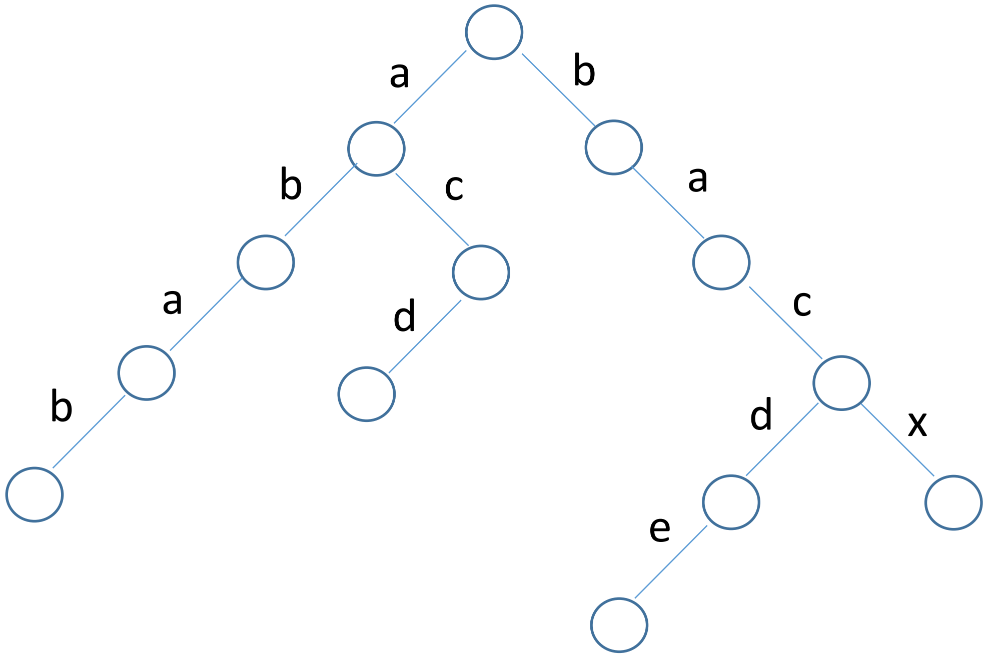
⇒ All clusters encoded in top DAG are unique.

Top Tree Compression Of Tries

Given a pattern string *P* of length *m* determines if P is a prefix of one of the strings

aba ☺

acx ☹



Randomized Monte-Carlo word RAM solution

Karp-Rabin Fingerprints

$\phi(x) = \sum_{i=1}^{|x|} x[i] \cdot c^i \bmod p$

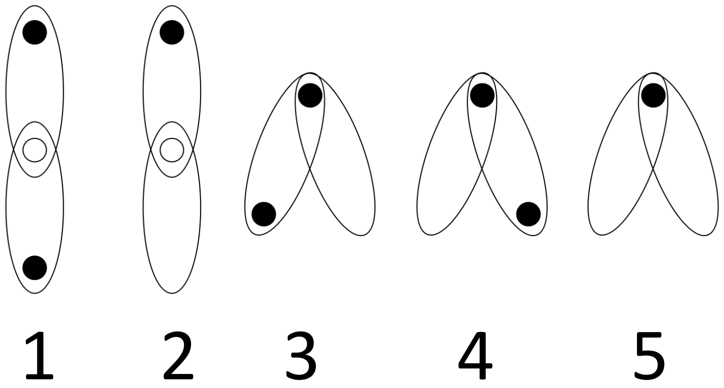
C – is a randomly chosen positive integer

P – prime

Let x = yz

Given any two of ϕ(x), ϕ(y) and ϕ(z) it is possible to calculate the remaining fingerprint in constant time.

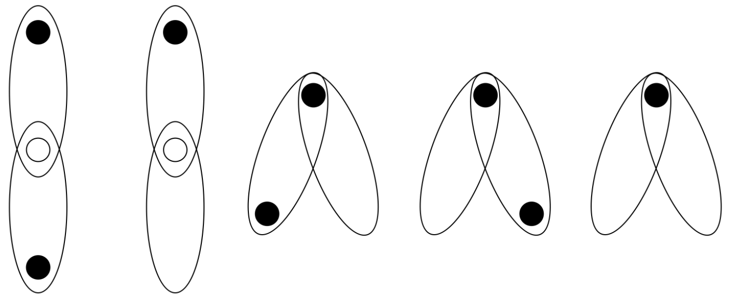
Compressed Pattern Matching



Case 1: A leaf cluster. Let e be the edge stored in C. We compare P[i + 1] with the label of e.

Case 2: 3,4,5. Let A and B be the left and right child of C, respectively. We compare P[i + 1] with the label α of the edge to the rightmost child of A. If P[i + 1] ≤ α, we continue the search in A for P[i+1...m]. Otherwise, we continue the search in B for P[i+1...m].

Compressed Pattern Matching



1 2 3 4 5

Case 3: 1,2. Let A and B be the left and right child of C, respectively. If $|spine(A)| > m - i$ we continue the search in A for P[i + 1...m]. Otherwise, we compare the fingerprint.

Compressed Pattern Matching

Given a pattern string *P* of length *m* determines if P is a prefix of one of the strings

Pointer Machine model,

Deterministic algorithm

Time Complexity - $O(\min(m \log \sigma , m + \log n))$

A Tight Lower Bound

Theorem: any structure storing a set S of strings of total length n over an alphabet of size σ needs to perform Ω(min(m+log n, m log σ)) comparisons to decide if a given pattern of length m belongs to S.

* Note that the bound holds regardless of the size of the structure

Proof: by showing that any comparison-based algorithm that given P checks if $\sum_{i=1}^{m} P[i] = 0 \ (mod\ 2)$ needs to perform Ω(min(m+logn, m log σ)) comparisons in the worst case.

Conclusion

Find new uses of top tree compression to solve problems faster or with less space.

Thanks