String Covers of a Tree

Jakub Radoszewski, Wojciech Rytter, Juliusz Straszyński, **Tomasz Waleń**, and Wiktor Zuba

University of Warsaw, Poland

Overview

Introduction and definitions

Cover of a string

Definition

String W is a **Cover** of a text T if every character of T is covered by some occ. of W in T.

Example

$$T: \quad \underbrace{a\ b\ a}\ \underbrace{a\ b}\ \underbrace{a\ b\ a}$$

$$W: \quad a\ b\ a$$

Cover of a string

Definition

String W is a **Cover** of a text T if every character of T is covered by some occ. of W in T.

Example

$$T: \quad a \quad b \quad a \quad a \quad b \quad a \quad b \quad a$$

$$W: \quad a \quad b \quad a$$

Observation

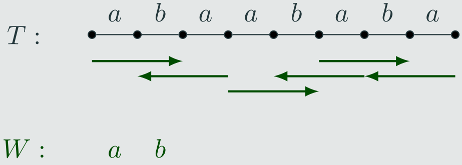
If W is a cover of T, then W is both a prefix and a suffix of T.

Cover of an undirected path

Definition

String W is a **cover** of an edge labelled undirected simple path T if every edge of T can be covered by some simple path with label W.

Example

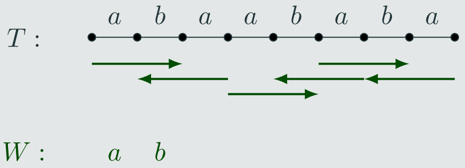


Cover of an undirected path

Definition

String W is a **cover** of an edge labelled undirected simple path T if every edge of T can be covered by some simple path with label W.

Example



Observation: We can cover path T with both W and W^R, so observation about prefix and suffix does not hold anymore.

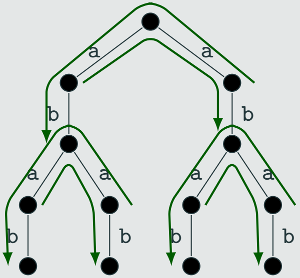
Cover of an undirected tree

Definition

String W is a **cover** of an edge labelled undirected tree T if every edge of T can be covered by some simple path with label W.

Example

$W = \texttt{aab}$



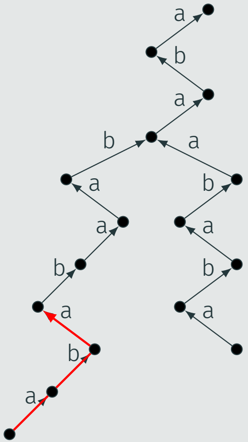
Cover of a directed tree

Definition

String W is a **cover** of an edge labelled directed tree T if every edge of T can be covered by some simple path **(with all edges directed towards the root of T)** with label W.

Example

$W = \texttt{aba}$



Motivation

- many results for covers in non-standard settings, for example:
    - 2-dimensional,
    - Abelian,
    - parameterized,
    - order-preserving,
    - on indeterminate and weighted strings,
- continuation of work on algorithmic and combinatorial properties of:
    - palindromes,
    - powers
    - runs

  in labeled trees.

Algorithm for String Covers of an Undirected Tree

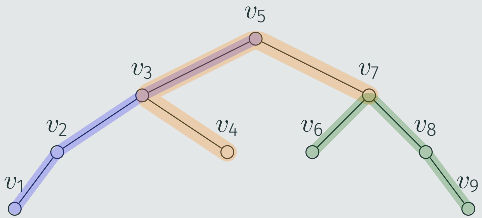
Covering Tree by Paths

Problem definition

Input A set M of simple paths in an undirected tree T (given by their endpoints).

Output YES if M covers T, NO otherwise.

Example



$$M = \{(v_1, v_5), (v_4, v_7), (v_6, v_9)\}$$

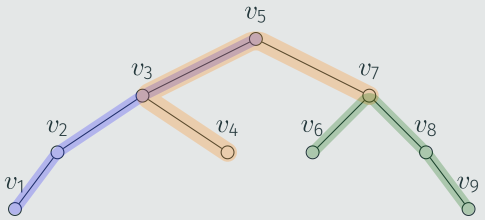
Covering Tree by Paths

Problem definition

Input A set M of simple paths in an undirected tree T (given by their endpoints).

Output YES if M covers T, NO otherwise.

Example



$$M = \{(v_1, v_5), (v_4, v_7), (v_6, v_9)\}$$

Lemma

Covering Tree by Paths can be solved in $O(|T| + |M|)$ time.

Enumerating all paths in tree T

- there are $\Theta(n^2)$ distinct paths in tree T,
- in $\Theta(n^2)$-time we can assign to each path (u, v) its identifier $id(u, v)$, such that two paths share the same identifier iff they have the same label,
- each identifier $id(u, v)$ can be an integer from range $\{1, \ldots, n^2\}$,
- we group paths with the same identifier using lists $L_i = \{(u_j, v_j) : id(u_j, v_j) = i\}$

$O(n^2)$ time and space algorithm

Algorithm 1: ReportAllCovers

Input: undirected labelled tree T

Output: all unique covers: $\{(v_i, v_j) : \; label(v_i, v_j)$ is a cover of $T\}$

```
enumerate all paths in T
C = { all identifiers from T }
foreach i ∈ C do
    solve problem of Covering Tree by Paths for T and paths L_i
    if answer is YES then
        report cover corresponding to the identifier i
    end
end
```

$O(n^2)$ time and space algorithm

Algorithm 2: ReportAllCovers

Input: undirected labelled tree T
Output: all unique covers: $\{(v_i, v_j) : \ label(v_i, v_j)$ is a cover of $T\}$
enumerate all paths in T
C = { all identifiers from T }
foreach $i \in C$ **do**
  solve problem of Covering Tree by Paths for T and paths L_i
  if *answer is YES* **then**
    report cover corresponding to the identifier i
  end
end

Unfortunately we can not afford to verify $O(n^2)$ candidates,
since this would require $O(n^3)$ time.

Candidates for covers

Lemma

An undirected labeled tree with n nodes has at most $2n - 2$ covers.

Proof

If we select any arbitrary leaf node w, then edge outgoing from w needs to be matched by some occurrence of cover that starts (or ends) in w.

This gives us set of $2(n-1)$ candidates for cover:

$$\{label(w, u) : u \in T\} \cup \{label(u, w) : u \in T\}$$

$O(n^2)$ time and space algorithm

Algorithm 3: ReportAllCovers

Input: undirected labelled tree T

Output: all unique covers: $\{(v_i, v_j) : \ label(v_i, v_j)$ is a cover of $T\}$

enumerate all paths in T

select arbitrary leaf $w \in T$

$C = \{id(u, w) : u \in T\} \cup \{id(w, u) : u \in T\}$

foreach $i \in C$ **do**

    solve problem of Covering Tree by Paths for T and paths L_i

    if *answer is YES* **then**

        report cover corresponding to the identifier i

    end

end

Results

Theorem

All covers of an undirected tree with n nodes can be computed in $O(n^2)$ *time and space.*

Results

Theorem

All covers of an undirected tree with n nodes can be computed in $O(n^2)$ time and space.

Theorem

All covers of an undirected tree can be computed in $O(n^2 \log n)$ time and $O(n)$ space.

Proof outline

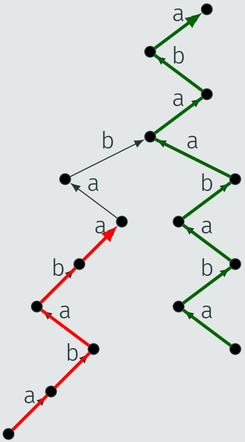
We introduce a new problem Anchored Covering Problem (covers need to pass through root node) and use Centroid Decomposition of a tree.

Algorithm for String Covers of a Directed Tree

Observations

If w is a cover of directed tree T:

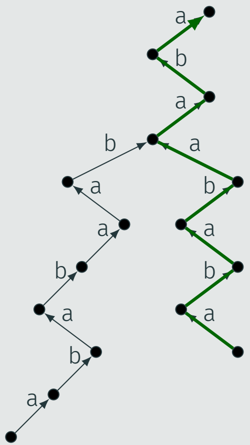
- w is a cover of **at least one** leaf-to-root label (i.e. ababa is a cover of ababaababa),
- it might happen that w is **not** a cover of some leaf-to-root labels (i.e. ababa is not a cover of abababa).



Observations

If w is a cover of directed tree T:

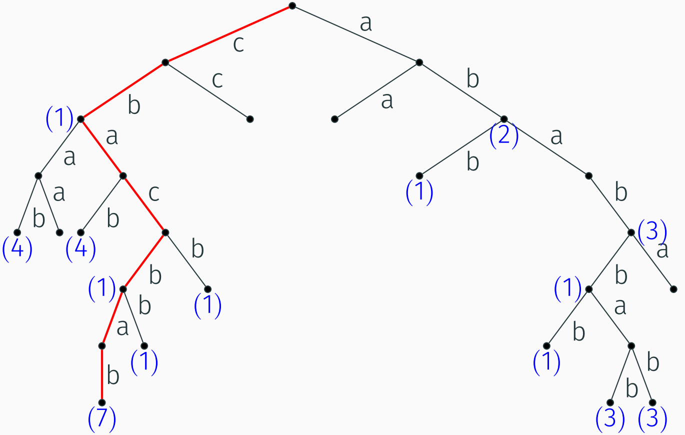
- w is a prefix of **all** leaf-to-root labels,
- w is a prefix of longest common prefix of all leaf-to-root paths ($S[1..m]$),
- here $S[1..m] = ababaaba$.



TreePref array

$\text{TreePref}_S[v]$ = length of longest common prefix of S and $label(v, r)$

Example of TreePref_S for $S = babcabc$ (marked with red color):



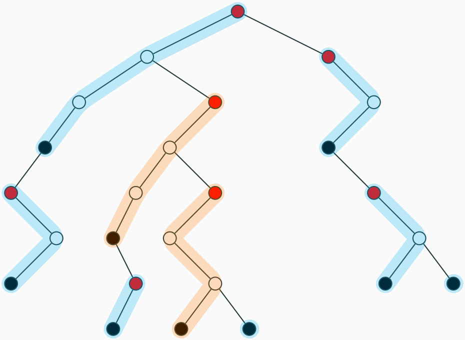
Chain decomposition

We decompose all nodes of T into chains.

Black (marked) nodes represent start of an occurrence of $S[1..d]$,
Red nodes represent top-node of the chain.

Length of a chain is a number of its non-root nodes (orange chains have length 4).

Objective: organize chains to minimize the maximal length of chains (each node is assigned to closest marked node).



Covers in Directed Trees - Algorithm outline

Algorithm 4: ReportAllCovers

Input: directed labelled tree T

Output: all unique covers of T

$S \leftarrow$ label of $label(leaf, r)$ (for any arbitrary leaf)

calculate $\textsc{TreePref}_S$

$m := \min\{TreePref_S[v] : v$ is leaf $\in T\}$

for $d \leftarrow m$ **downto** 1 **do**

  maintain partition of nodes of T into chains M,
    each chain corresponds to the occurrence of $S[1..d]$

  if *maximal length of a chain from* $M \leq d$ **then**

    report $S[1..d]$ as a cover of T

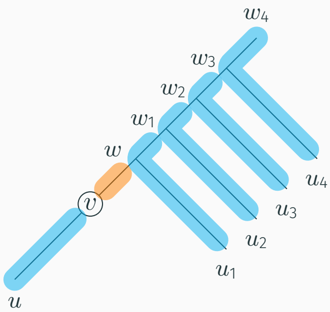
  end

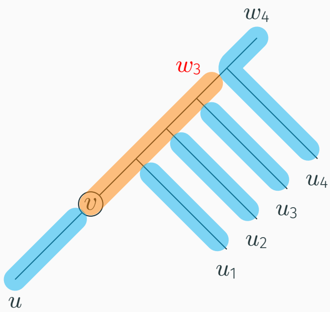
end

Chain updates

What happens, when we observe new occurrence of $S[1..d]$ starting in node v?

- node v is marked as black node,
- existing chain is split creating new chain starting in v,
- some nodes are re-assigned to new chain.





Some details

- to calculate $\textsc{TreePref}_S$ array we use Suffix Tree of a Tree data structure (+LCA queries),
- we need to prove that chain updates amortize to $O(n)$ time,
- to keep track of top-nodes of chains, we use **Dynamic Marked Ancestor Problem**, each such query requires $O(\log n / \log \log n)$-time.

Summary

Summary of Algorithms for String Covers of a Tree

Variant	Time	Space
undirected	$O(n^2)$	$O(n^2)$
undirected	$O(n^2 \log n)$	$O(n)$
directed	$O(n \log n / \log \log n)$	$O(n)$

Thank you for your attention!