Pattern Matching in Trees

Justus Schwartz

Motivation

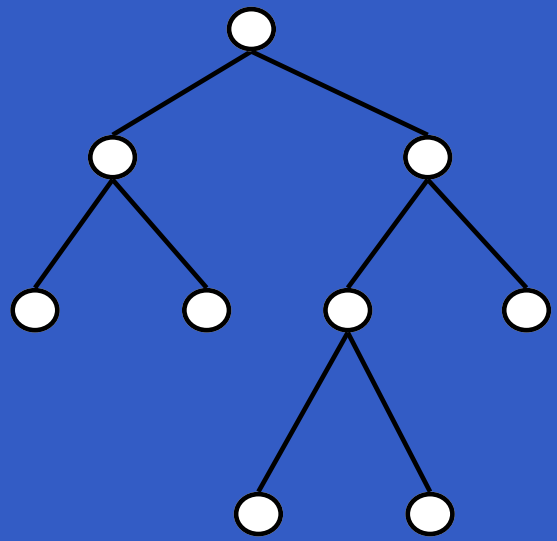
- Nonprocedural programming languages
- Code-Optimizations
- Automatic theorem proving
- Symbolic computation on algebraic terms
- Every application with tree-replacement

Problem Definition

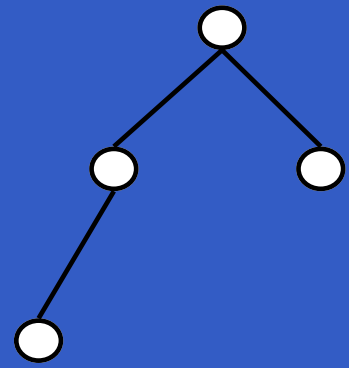
- Given a pattern tree P and a subject tree T.
- Children of all nodes are ordered.
- Nodes are labelled.
- P matches at node a if there is a
  - one-to-one mapping from nodes of P to T
  - root of P maps to a
  - if x maps to y, children map to corresponding children of y
- find all matches of P

Example

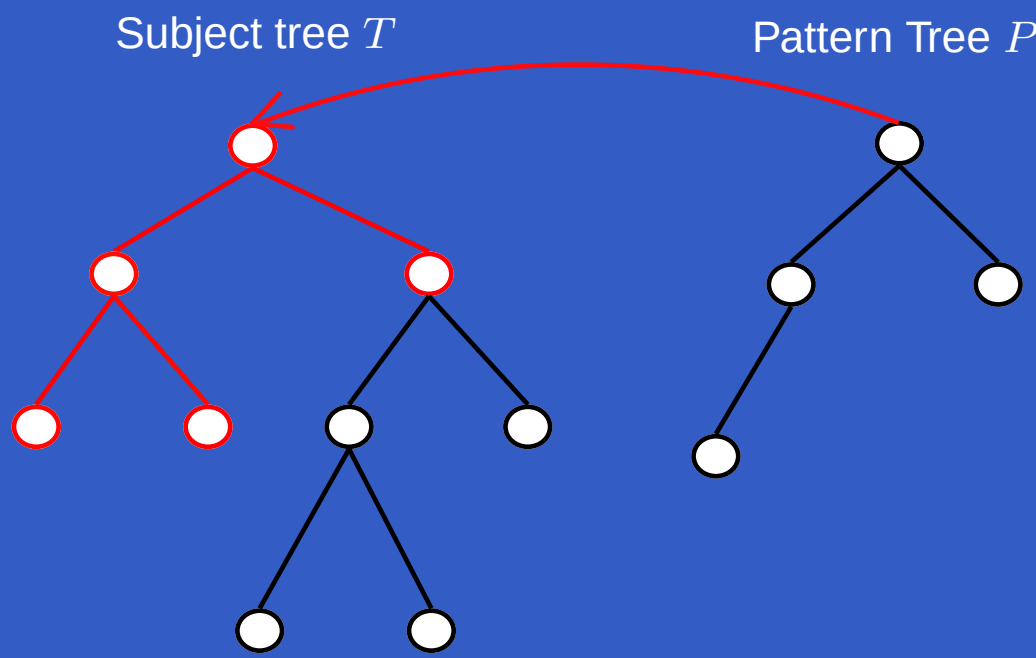
Subject tree T



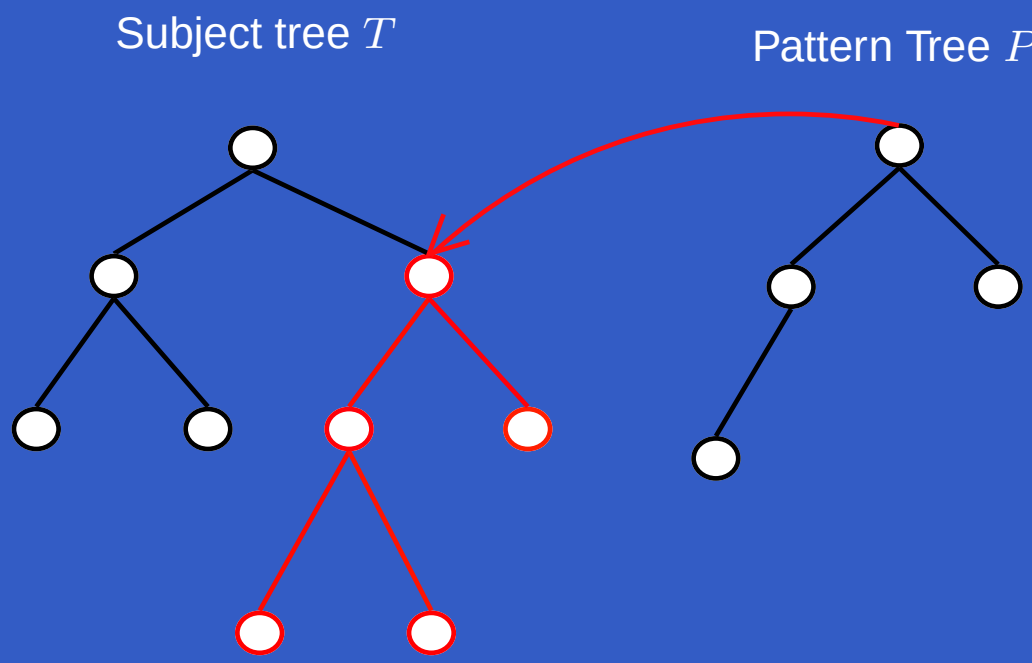
Pattern Tree P



Example



Example



Overview

- Naive Algorithm
- Algorithm by Dubiner,Galil,Magen ($\tilde{O}(n\sqrt{m})$)
- Sketch of faster Algorithms by Cole, Hariharan and Indyk

Definitions

- Number of nodes in subject tree: n
- Number of nodes in pattern tree: m
- $\tilde{O}(f(n,m)) = O(f(n,m) \cdot \mathrm{polylog}(m))$
- Order information of children is coded into node label. So labels of Children are distinct. We could say we consider unlabelled trees with order information.

Naive Algorithm

The naive Algorithm is straightforward.

- Traverse over all nodes of T
- At each node try to match the pattern tree P.

This seems to take O(nm) time. We will see, it's not that bad.

Algorithm by Dubiner,Galil,Magen

- The algorithm takes $\tilde{O}(n\sqrt{m})$ time.
- The naive algorithm will be revisited.
- Matching strings in trees
- Periodicities in the pattern will be exploited.
- Node labels will be from alphabet $\{0, 1\}$. If not we may encode them, which leads to a logarithmic blowup, which will be hidden by the $\tilde{O}$ notation.

Matching Paths

For a node v of P denote σ_v as the path from the root of P to v.
v matches T at node w if σ_v as a tree matches T at w.

Remark: Iff for every leaf v of P v matches T at w then P matches T at w.

Naive algorithm revisited

- The naive algorithm takes time $O(nh)$, with h height of P

A match of a node in P at depth i is only tested if the path of the ancestors have matched. As labels of children are distinct the path of ancestors could not lead to another node.
$\Rightarrow$ every node is compared at most with one node of every depth.

Knuth-Morris-Pratt in trees

Define failure function. Given string $s = s_1, \ldots, s_m$.

$$f(i) = \max\{j | s_1, \ldots, s_j \text{ is proper suffi x of } s_1, \ldots, s_i\}$$

While traversing T by DFS keep at each node v a pointer to largest node w such that path from w to v is prefix of s.

KMP in trees (2)

- Assume node v, parent nodes pointer to w, path length i
- Set pointer of v to w.
- If the path is prefix continue (compare label)
- Else set pointer to $f(i-1)$th ancestor and compare again until prefix is found.

KMP in trees (3)

Since the Pointers advance only with the DFS, the total time is linear in size of T

The failure function is calculate recursively in a similar way.

Analogous we can match a set of strings in linear time. (Aho-Corasick)

Periods in strings

A string α is a Period of string β if for some $k > 0$ β prefix of α^k. Some facts on periods:

1. α is a period of $\beta \Leftrightarrow \beta = \alpha\gamma = \gamma\delta$
2. β has period of length $k \Leftrightarrow \beta_i = \beta_{i+k}$ with $1 \leq i \leq |\beta| - k$
3. if $\alpha\beta$ and $\beta\gamma$ have a period of length k and $|\beta| \geq k$ then $\alpha\beta\gamma$ have period k

Example: $\alpha\beta = abcabcab\underbrace{cabc}_{\beta}$, $\beta\gamma = cabc\underbrace{abcabcab}_{\gamma}$

$\alpha\beta\gamma = abcabcabcabcabcabcab$

k-truncated suffix tree

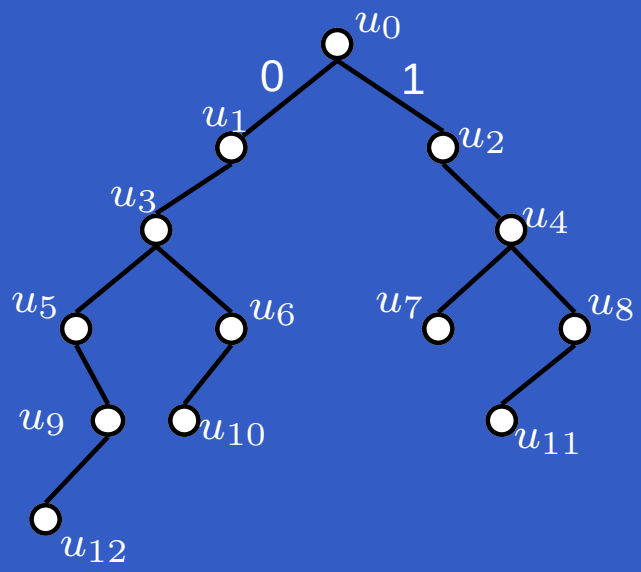
Define for node v $\sigma_{v,k}$ as the string of the last k characters of σ_v (if existent).

The k-truncated suffix tree of P, $\Sigma_{P,k}$, is defined as the trie for all strings $\sigma_{v,k}^R$ for all nodes v of P, where R means reversed.

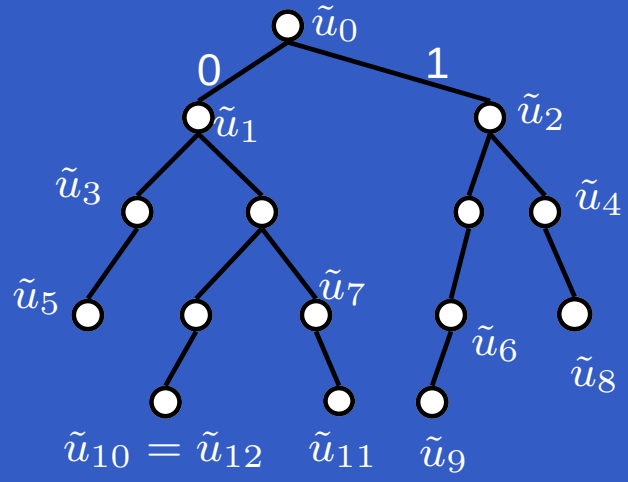
For every node v in P there is a corresponding node $\tilde{v}$ in $\Sigma_{P,k}$. And the path from the root of $\Sigma_{P,k}$ to $\tilde{v}$ is $\sigma_{v,k}$ reversed.

$\Sigma_{P,k}$ can be computed in $O(km)$

k-truncated suffix tree



Tree P



$\Sigma_{P,4}$

k-truncated suffix tree

A useful fact about k-truncated suffix trees that we will use is:
$\tilde{u}$ is an ancestor of $\tilde{v}$ in $\Sigma_{P,k}$ iff $\sigma_{u,k}$ is a suffix of $\sigma_{v,k}$. ($\sigma_{\tilde{u}}$ prefix of $\sigma_{\tilde{v}}$ reverse both and the statement follows)

So for leaves $\tilde{v}$, $\tilde{w}$ the strings σ_w and σ_v cannot be suffix of one another.

We will use suffix trees with $k = 3l = 3 \cdot \lceil \sqrt{m} \rceil$.

Number of leaves of Σ

After computing the $3l$-truncated suffix-tree of P $\Sigma = \Sigma_{P,3l}$ in time $O(lm) = O(m\sqrt{m})$, we will distinguish between two cases.

1. Σ has at least l leaves.
2. Σ has at most l leaves.

At least l leaves

Chose set S of l nodes from P corresponding to leaves of Σ.
In time $O(n)$ mark the nodes of T for each node v in S where σ_v matches.
A node of T can be the end of only one path matching σ_v for all v in S because of the fact stated for leaves of Σ.
Therefor at most n marks are made.

At least l leaves

Only $\frac{n}{l}$ of the marked nodes may be marked l times and be considered possible matchings of P.
Every possible root is tested in $O(m)$ time. This yields altogether a time of $O(nl + \frac{n}{l}m) = O(n\sqrt{m})$

At most l leaves

For this case periodicities in the pattern tree will be discovered.
Then the problem is reduced to string matching with don't care.
The subtree of P consisting of the paths to leaves whose depth is at most $3l$. With the naive Algorithm we can match this subtree in $O(nl) = O(n\sqrt{m})$

So we assume that all leaves of P have at least a depth of $3l$.

At most l leaves

So for every leaf v $|\sigma_{v,3l}| = 3l$ and $\tilde{v}$ is a leaf of Σ.

Lemma: For $v_1, \ldots, v_{l+1}$ distinct nodes of P there are $i \neq j$ such that $\sigma_{v_i,3l}$ is suffix of $\sigma_{v_j,3l}$

Because there are at most $3l$ leaves there are i, j such that $\tilde{v}_i$ is an ancestor of $\tilde{v}_j$. $\square$

Periodicity of paths

Lemma: For every leaf v of P the string of all but at most the last l nodes of σ_v has period of length at most l.

For a node v_0 with depth at least l let v_i be its ith ancestor. By the previous lemma there are $0 \leq i < j \leq l$ such that $\sigma_{v_j,3l}$ is a suffix of $\sigma_{v_i,3l}$.

For the node u which is the beginning of $\sigma_{v_i,3l}$ the path τ from u to v_j is a prefix and a suffix of $\sigma_{v_i,3l}$.

By fact (α period $\Leftrightarrow \alpha\gamma = \gamma\delta$) follows that $\sigma_{v_i,3l}$ has a period of length $0 < j - i \leq l$.

Periodicity of paths

Considering a leaf v there is an ith ancestor v_i of v with $0 \leq i \leq l$ such that $\sigma_{v_i,3l}$ has period of length at most l.
Further up the path we get by the same considerations periodicities of length at most l.
Intersection is at least l long $\Rightarrow$ they have the same period $\Rightarrow$ σ_{v_i} has same periodicity as $\sigma_{v_i,3l}$.

Periodicity of paths

Conclusion: For every leaf v σ_v has period of length at most l, except for a tail of at most l nodes.
To each leaf v the pair of minimal period p and the tail will be attached. (Period ending exactly before tail.)
There are at most l distinct pairs. (Because of at most l leaves in $\Sigma_{P,3l}$

Tail-Period Pairs

We can compute the Tail-Period Pairs in $\tilde{O}(ml)$.

For a leaf v we apply a linear string-matching algorithm to σ_v and the first l characters of σ_v.
The first non-trivial occurrence gives the period (possibly permutated)
The tail is now easily determined in $O(l)$.
Sorting these O(m) pairs returns the at most l pairs.
$\Rightarrow$ time $\tilde{O}(ml)$
$\square$

Maximal-Periodic-Paths

For matching we consider period-tail pair with the period p.

A path in a tree is called a maximal-periodic-path if it

1. has period p (may start any place in p but has to end with p)
2. p is repeated at least twice
3. is maximal (cannot be extended fulfilling 1.)

Maximal-Periodic-Paths

If two maximal-periodic-paths intersect at node v for one path it is among the first $|p|$ nodes. Otherwise the intersection is at least of length $p \Rightarrow$ the paths are equal.

total length of maximal-periodic-paths $O(n)$

Finding the maximal-periodic-paths in $O(n)$ time works in a similar way as string matching.

Reduction to Strings

Now with the given period-tail pair. We can find all corresponding maximal-periodic-paths and occurrences of the tail in T in time $O(n)$

Define for every maximal-periodic-paths in T:

$$b_i = \begin{cases} 1 & \text{if after } i \text{ periods the tail occurs} \\ 0 & \text{otherwise} \end{cases}$$

Reduction to Strings

Similar for the pattern tree P we define for maximal-periodic-paths starting at the root:

$$a_i = \begin{cases} 1 & \text{if after } i \text{ periods the tail occurs and ends at a leaf} \\ 0 & \text{otherwise} \end{cases}$$

This can be done in time $O(m)$ and there are at most $|p|$ paths.
The length of the a/b-sequences is at most the length of the length of the path/$|p|$

String matching with don’t cares

To match the set of leaves with a given period-tail reduces to matching the a-sequences with any of the b-sequences with don’t care. (0 in a as don’t care)

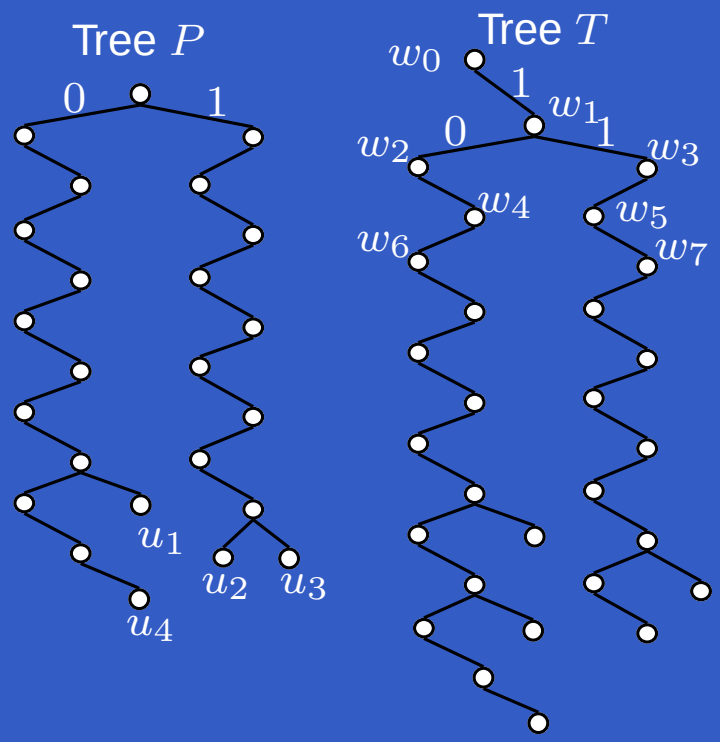
We have at most $|p|$ paths for every period-tail pair with convolution we can match them in $\tilde{O}(\frac{\text{pathlength}}{|p|})$.

String matching with don't cares

We have time $\tilde{O}(n)$ for a period-tail pair. Now translate matches back to nodes considering position in period.

Finally it follows that if Σ has at most l leaves, we can match P and T in time $\tilde{O}(n\sqrt{m})$

Example



pair (“01”, “1”):

a-seq left: 000011.

a-seq right: 00001.

b-seq: from w_0 “1000111”

and from w_1 “000010”

Example

The a-sequence 000011 matches 1000111 at position 0 and 1 and does not match 000010. So for leaves u_1 and u_4 we have two matches at w_1 and w_4.
The a-sequence 00001 matches 1000111 at 0,1 and 2 and matches 000010 at 0. So for leaf u_3 we have matches at $w_{0,2,6}$ and w_1.
Same procedure for the other period-tail pair.
In a final step (traversing T) the only match at w_1 is found.
The only place where all leaves match.

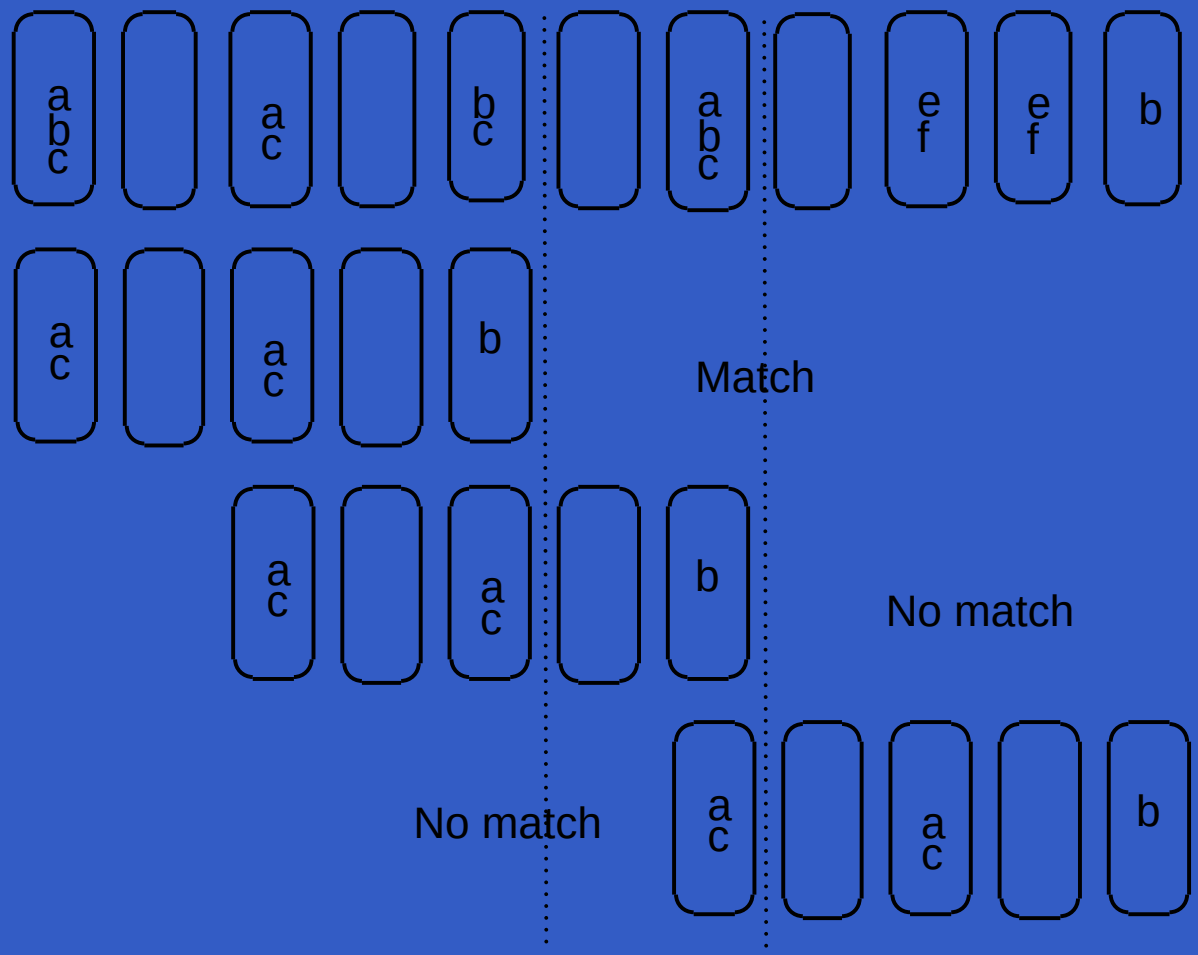
Sketch of faster algorithms

Cole and Hariharan reduce the tree pattern matching to subset matching. They can do this in linear time.

Subset matching: the pattern p = $p_1, \ldots, p_m$ and the text $t = t_1, \ldots, t_n$, where p_i, t_i are subsets of some alphabet Σ match at position k iff $p_i \subset t_{k+i-1} \forall 1 \leq i \leq m$

For subset matching they give an $O(n \log^3 m + m)$ deterministic algorithm and also an $O(n \frac{\log^3 m}{\log \log m} + m)$ probabilistic algorithm.

Subset Matching



Reduction to spine pattern matching

First Cole and Hariharan reduce the tree pattern matching to spine pattern matching.

Each the pattern and the text have one designated path starting at the root, the spine.
Both paths will have the same period.
The matches where the pattern spine lies in the text spine are sought.

Reduction to Spine-Pattern Matching

This reduction is done via choosing a certain spine in the pattern tree.

According to the period of this spine maximal-periodic-paths are found in the subject tree.

Certain paths are then taken as an instance of the spine pattern matching problem.

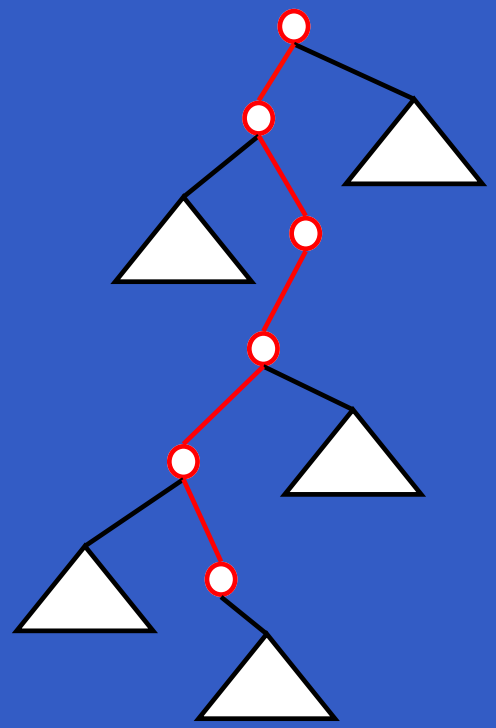
Reduction from spine to subset matchin

The nodes in the subtrees which are not on the spine are enumerated, such that nodes in subtrees with same paths to subtree root get the same label.

All labels in a subtree are put into a set. Obviously subset matching is now equivalent to spine pattern matching

From these reductions follows a time bound of $O(n \log^3 m)$

Tree with spine



A tree with spine. Each spine node has only one non-spine children.

The numbered nodes in the off-spine subtrees for each spine node form the subsets for matching

Resulting size of subset matching problem is $O(n+m)$

Concluding remarks

- For interesting applications tree pattern matching respective tree replacements are essential.
- Fast algorithms exist
- Non-trivial lower bound would be interesting. (at least as hard as string matching)