# Algorithms for Massive Data Sets

**Student name and id:** Alexander R. Johansen (s145706)

**Collaborator name(s) and id(s):** Jonas (s142957)

**Hand-in for week:** 10

## 1 Compressed subsequence recognition revisited

The $Compressed\_Subsequence\_Recognition\_Revisited(SLP, S')$ Problem, which I will refer to as the $CompSub(SLP, S')$ can be solved in $O(nd)$ space, where $d$ is the amount of unique characters in $P$, $d <= M$, and $O(hM)$ query time, where $h$ is the height of the SLP parse-tree, $h <= n$. The space bound is given by using dynamic programming to store 1: the size of every rule and 2: the first occurrence of every unique letter in $P$ in every rule. The query time bound is given by doing traversals of the parse tree, where at most $O(M)$ traversals along the $O(h)$ height tree is being done.

### 1.1 Analysis of data structure

The length of each SLP can be constructed in $O(n)$ space(Mandatory assignment 09).
The $O(nd)$ table is a table with rules on one axis and unique characters from the pattern in the other. Using a leaf-to-root approach it is possible to construct the first occurrence of every unique character in every rule in $O(nd)$ time. The approach follows the following dynamic programming logic:
Start with the leafs, as the leafs contains at max one letter compute it's sparse representation in the $O(nd)$ table, this should take $O(d)$ time.
When combining lower hanging rules into a parent rules we would like to manage the childs respective placements in the parent rule, such that left childs placements starts from 0, where as right childs starts from leftChild.length. Thus if a character is represented in the left child, it must be the first occurrence of a character since all occurences of character's in the right child does not occur until after the leftChild. With this left-right child invariance, it can be concluded that the parent can copy the entire left child, and only needs to query the right child for the places in which the left child had missing values, the right child will then have its indexes padded with leftChild.length. Thus combining child notes to a parent would take $O(d)$.
Since both the leaf-case and parent-building case took $O(d)$ and a total of $O(n)$ rules exist, it can be done in $O(nd)$ time and space.

### 1.2 Analysis of query

**The simple algorithm:**
$1(P_i, X_j)$: From a given rule(when $i = 1$, rule is root) use the $RandomAccess(k, X_j)$ to traverse from a given rule to it's $k'th$ index, where k is decided by accessing the $O(nd)$ table at the $X_j$'th column and $P_i.letter$ row.

2: From a found leaf, traverse upwards in the SLP tree and look for right hanging children whom contains the $P_{i+1}.letter$ by looking in the $O(nd)$ table. If a right-hanging rule with the character exists, go back to step 1 and repeat using the right-hanging rule as $X_j$.
**Cost**
1: The $RandomAccess(k, X_j)$ can be done in $O(h)$(Mandatory Exercise 09)
2: Traversing upwards can take at max $O(h)$
2: Each evaluation of right hanging trees are a constant look-up in a table, thus $O(1)$
For each $P_i$ in $P$, it takes $O(h + h * 1) = O(h)$. In total $O(hM)$ for all $P$
**A more complex algorithm.**
1. Bille et al. 2013 argues that the $RandomAccess$ problem can be solved in $O(logN)$[1]
1 2: $O(h)$ on both down and uptraversal could be tighter as it is only possible to uptraverse the entire path to the root once(as it only has one right-hanging child).

## 1.3 Analysis of invariant

The invariant in this algorithm is that it is able to locate the first occurence in $S$ of a given letter in $P$ from a specific index(why we used findnext(i,c) in the exercises). This algorithm can do that using using the uptraversal from a given i, the $O(nd)$ table for evaluating right-childs and $RandomAccess(k, X_j)$ in step 2.
During the uptraversal; looking at every right child from bottom upwards will give the next chunk of charracters in S. The $O(nd)$ table will then allow evaluation of whether the next letter in $P$ exists in the given chunk, and the first index if it does, thus the "first occurence in $S$ of a given letter in $P$ from a specific index" can be validly solved.

[1] http://arxiv.org/pdf/1001.1565.pdf