String Processing Algorithms 2015 - Week 2 Exercises

Rodion Efremov

November 1, 2015

Exercise 1

Outline algorithms that find the most frequent symbol in a give string.

(a) for ordered alphabet, and

(b) for integer alphabet.

The algorithms should be as fast as possible. What are their (worst case) time complexities? Consider also the case where $\sigma \gg n$.

Solution

Algorithm 1: MostFrequentSymbol(S)

```
let f be an empty map f: Σ → ℕ
μ = nil
L_μ = 0
for i = 1 to |S| do
    if S[i] is not mapped in f then
        f(S[i]) = 1
        if L_μ = 0 then
            L_μ = 1
            μ = S[i]
    else
        f(S[i]) = f(S[i]) + 1
        if L_μ < f(S[i]) then
            L_μ = f(S[i])
            μ = S[i]
return μ
```

Exercise 2

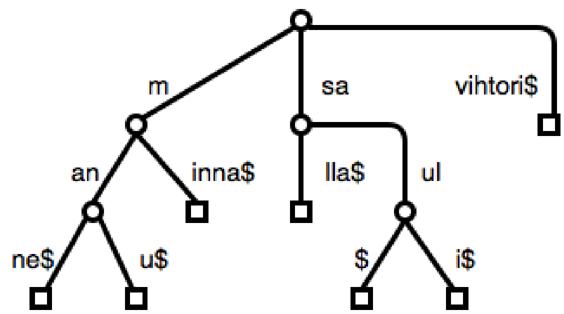
Let $\mathcal{R} = \{\texttt{manne}, \texttt{manu}, \texttt{minna}, \texttt{salla}, \texttt{saul}, \texttt{sauli}, \texttt{vihtori}\}$.

(a) Give the compact trie of $\mathcal{R}$.

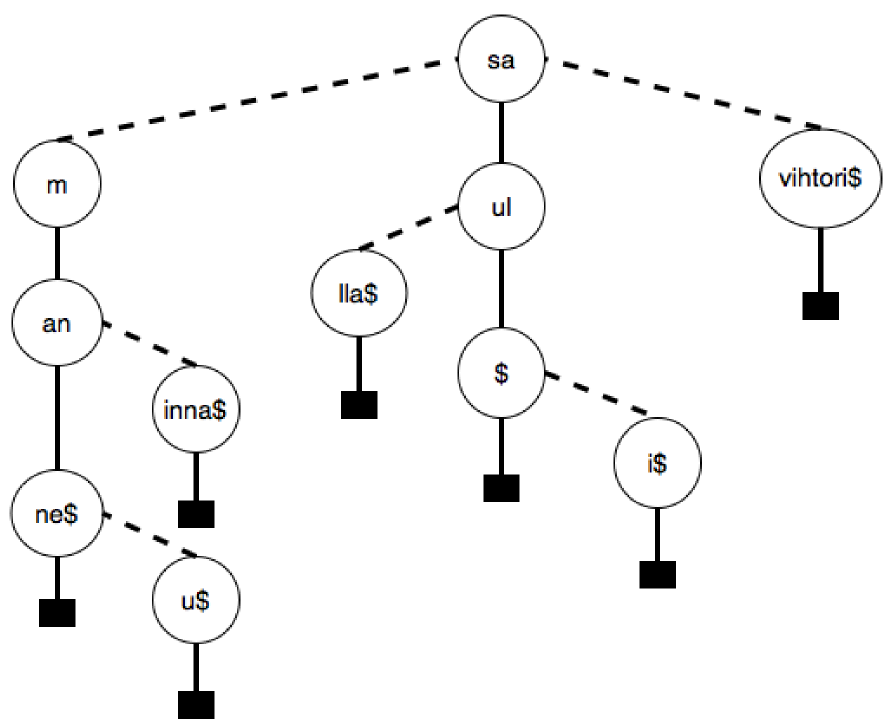
(b) Give the balanced compact ternary trie of $\mathcal{R}$.

Solution

(a)



(b)



Exercise 3

Exercise 4

Prove

(a) Lemma 1.14: For $i \in [2..n]$, $LCP_{\mathcal{R}}[i] = lcp(S_i, \{S_1, \dots, S_{i-1}\})$.

(b) Lemma 1.15: $\Sigma LCP(\mathcal{R}) \leq \Sigma lcp(\mathcal{R}) \leq 2 \cdot \Sigma LCP(\mathcal{R})$.

Solution

(a)

We need an auxiliary lemma first:

Lemma 1 (Neighborhood lemma)**.** *If $S_1 < S_2 < S_3$, $lcp(S_1, S_3) \leq lcp(S_2, S_3)$.*

Proof. The proof is by contradiction: Let $l_{13} = lcp(S_1, S_3)$ and $l_{23} = lcp(S_2, S_3)$. Assume the opposite that $l_{13} > l_{23}$. Now

$$S_1 = a_1 a_2 \dots a_{l_{23}} \dots a_{l_{13}} b_1 b_2 \dots,$$
$$S_2 = a_1 a_2 \dots a_{l_{23}} c_1 c_2 \dots,$$
$$S_3 = a_1 a_2 \dots a_{l_{23}} \dots a_{l_{13}} d_1 d_2 \dots.$$

Since $S_1 < S_2$, $c_1 > a_{l_{23}+1}$ and we must also have that $S_2 > S_3$, which is a contradiction. Analogous proof can be used to deduce that $lcp(S_1, S_3) \leq lcp(S_1, S_2)$. $\square$

Example:

$$S_1 : aaaab$$
$$S_2 : aaaba$$
$$S_3 : aabba$$

Now assume that $i \in [2...n]$. We have that

$$lcp(S_i, \{S_1, \dots, S_{i-1}\}) = \max\{lcp(S_i, S_{i-1}),\ lcp(S_i, \{S_1, \dots, S_{i-2}\})\}.$$

By neighborhood lemma, for any $j = 1, 2, \dots, i-2$, $lcp(S_i, S_j)$ cannot exceed $lcp(S_i, S_{i-1})$ and we must have that

$$lcp(S_i, \{S_1, \dots, S_{i-1}\}) = lcp(S_i, S_{i-1}) = LCP_{\mathcal{R}}[i],$$

as expected.

(b)

$$\begin{aligned}
\Sigma lcp(\mathcal{R}) &= \sum_{S \in \mathcal{R}} lcp(S, \mathcal{R} \setminus \{S\}) \\
&\leq \sum_{i \in [1..n-1]} lcp(S_i, S_{i+1}) + \sum_{i \in [2..n]} lcp(S_{i-1}, S_i) && \text{(by neighborhood lemma)} \\
&= \sum_{i \in [2..n]} lcp(S_{i-1}, S_i) + \sum_{i \in [2..n]} lcp(S_{i-1}, S_i) \\
&= 2 \sum_{i \in [2..n]} lcp(S_{i-1}, S_i) \\
&= 2 \sum_{i \in [2..n]} LCP_{\mathcal{R}}[i] \\
&= 2 \sum_{i \in [1..n]} LCP_{\mathcal{R}}[i] && \text{(since } LCP_{\mathcal{R}}[1] = 0\text{)} \\
&= 2 \cdot \Sigma LCP(\mathcal{R}).
\end{aligned}$$

What comes to the lower bound of $\Sigma lcp(\mathcal{R})$, we have

$$\begin{aligned}
\Sigma lcp(\mathcal{R}) &= lcp(S_1, S_2) + lcp(S_{n-1}, S_n) \\
&\quad + \sum_{i \in [2..n-1]} \max\{lcp(S_i, S_{i-1}), lcp(S_i, S_{i+1})\} && \text{(by neighborhood lemma)} \\
&\geq \sum_{i \in [2..n]} lcp(S_{i-1}, S_i) \\
&= \sum_{i \in [2..n]} LCP_{\mathcal{R}}[i] \\
&= \sum_{i \in [1..n]} LCP_{\mathcal{R}}[i] && \text{(since } LCP_{\mathcal{R}}[1] = 0\text{)} \\
&= \Sigma LCP(\mathcal{R}),
\end{aligned}$$

which concludes the proof.

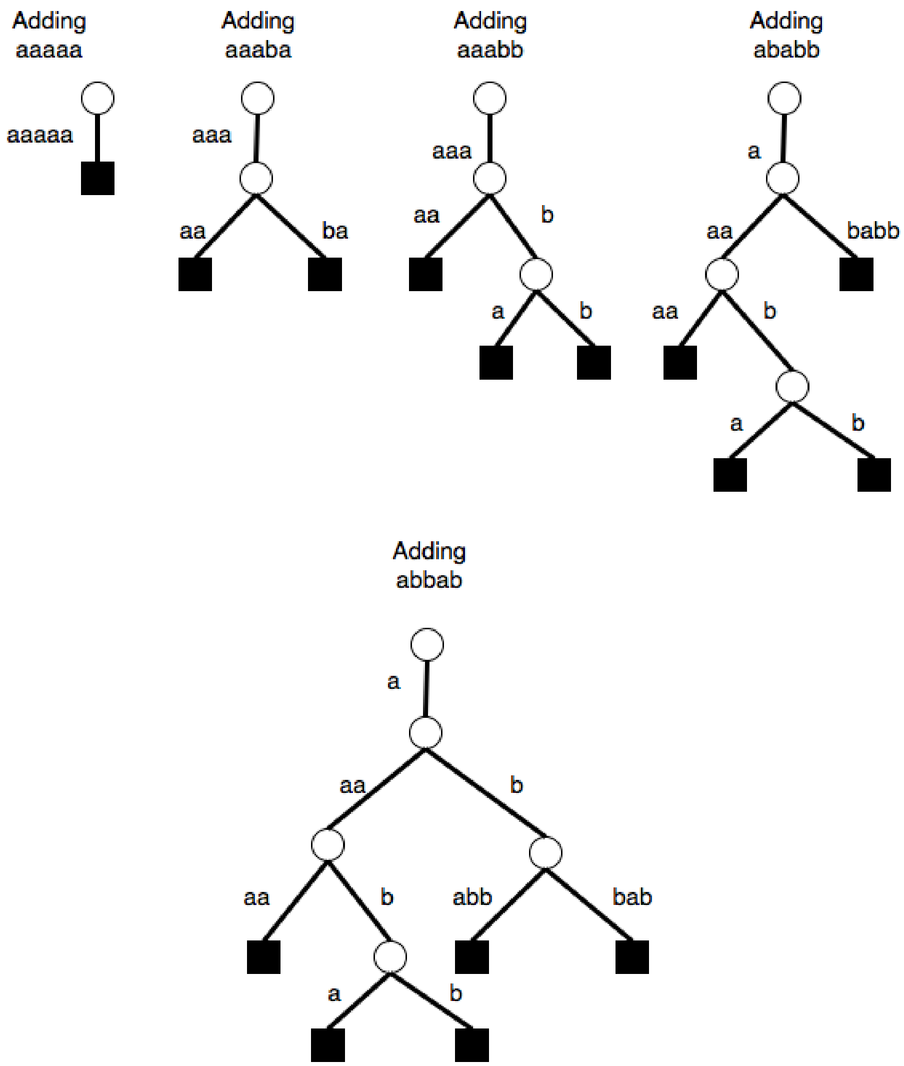
Example:

$\mathcal{R}$	$LCP_{\mathcal{R}}$	$lcp(S, \mathcal{R})$
aaaa	0	3
aaab	3	3
abba	1	1
baab	0	0
$\sum$	4	7

Exercise 5

Show how to construct the compact trie for a set $\mathcal{R}$ in $\mathcal{O}(|\mathcal{R}|)$ time (rather than $\mathcal{O}(||\mathcal{R}||)$ time) given the string set $\mathcal{R}$ in lexicographic order and the LCP array $LCP_{\mathcal{R}}$.

Solution



The LCP array is

i	S_i	$LCP_{\mathcal{R}}[i]$
1	$aaaaa$	0
2	$aaaba$	3
3	$aaabb$	4
4	$ababb$	1
5	$abbab$	2

It would seem that the algorithm must keep track of the last added edge/node. If the current LCP value ($l_i = LCP_{\mathcal{R}}[i]$) is no less than the previous one ($l_{i-1} = LCP_{\mathcal{R}}[i-1]$), take the last edge and split it after $l_i - l_{i-1}$ characters into two edges: the left one completing the previously added string, and the right one completing current string. If, however, $l_i < l_{i-1}$, we must restart from the root of the trie.