Advanced Data Structures

Labwork 2: Data structures for operations on strings

1. Consider the pattern $P = \texttt{aabab}$. Compute the prefix function

   $$\pi : \{1, 2, 3, 4, 5\} \to \{0, 1, 2, 3, 4\}$$

   for this pattern:

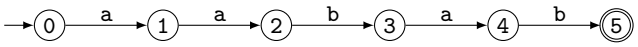
q	$\pi[q]$
1	
2	
3	
4	
5	

reverse of a palindrom pattern

2. Construct the string-matching automaton for the pattern $P = \texttt{aabab}$ and illustrate how it runs on the text string $T = \texttt{aaababaabaababaab}$.

δ	a	b
$\to 0$	1	0
1	1	2
2	3	0
3	4	0
4	3	5
$\leftarrow 5$		

this is like a state thing



i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
$T[i]$		a	a	a	b	a	b	a	a	b	a	a	b	a	b	a	a	b
$\delta^*(T[1..i])$																		

3. Construct the keyword tree and its failure links for the set of patterns

   $$\mathcal{P} = \{\texttt{The}, \texttt{hand}, \texttt{and}, \texttt{pork}, \texttt{port}, \texttt{pot}\}.$$

4. Construct the keyword tree and its failure links for the set of patterns

   $$\mathcal{P} = \{\texttt{woman}, \texttt{man}, \texttt{meat}, \texttt{animal}\}.$$

5. Draw the suffix tree and it suffix links for the text `banana$`.

6. Draw the suffix tree and its suffix links for the text `mamaia$`.

7. Draw the generalized suffix tree and its suffix links for the set of texts $\{\texttt{tatar}, \texttt{tabac}\}$.

Programming exercises

1. Write in C++ or Java a program which reads from the standard input a pattern $P[1..m]$, computes its prefix function, and prints it to the standard output.

   For example, if the input is the pattern `capcana` then the output could be:

   ```
   pi[1]=0
   pi[2]=0
   pi[3]=0
   pi[4]=1
   pi[5]=2
   pi[6]=0
   pi[7]=0
   ```

2. Write in C++ or Java a program which solves the following problem:

   (a) It reads a text T from a text file specified by the user

   (b) It reads from the terminal the number z of strings (patterns) $P_1, P_2, \ldots, P_z$

   (c) It reports all positions from T where there is an occurrence of a patterns P_i $(1 \leq i \leq z)$

   The interaction of the user with the program should be as follows:

   `Enter the source file for the text:` *file-name*
   `Enter the number of patterns:` z
   `Enter pattern 1:` P_1
   ...
   `Enter pattern` z`:` P_z

   Afterwards, the program displays the occurrences of every pattern in text the T which was read from the text file *file-name*:

   `Pattern 1 occurs at positions` $p_{1,1}$ ... p_{1,n_1}
   ...
   `Pattern` z `occurs at positions` $p_{z,1}$ ... p_{z,n_z}

   The program should implement the Aho-Corasick algorithm which builds the keyword tree of the set of templates $\mathcal{P} = \{P_1, P_2, \ldots, P_z\}$ together with its failure links.

   Illustrated example

   Suppose that the file `source.txt` contains the text

   ```
   Tim a mers la Timisoara sa-si cumpere o casa.
   ```

   If we specify

```
Enter the source file for the text: source.txt
Enter the number of patterns: 4
Enter pattern 1: Tim
Enter pattern 2: Timis
Enter pattern 3: sa
Enter pattern 4: casa
```

then the program must display

```
Pattern 1 occurs at positions 1 15
Pattern 2 occurs at positions 15
Pattern 3 occurs at positions 25 43
Pattern 4 occurs at positions 41
```