58093 String Processing Algorithms (Autumn 2011)
Exercises 4 (22–23 November)

1. Two string x and y are *rotations* of each other if there exists strings u and v such that $x = uv$ and $y = vu$. For example `abcde` and `deabc` are rotations of each other. Describe a *linear time* algorithm for determining whether given two strings are rotations of each other. (Hint: use a linear time exact string matching algorithm.)

2. The Knuth–Morris–Pratt algorithm differs from the Morris–Pratt algorithm only in the failure function, which can be defined as

   > $fail_{\text{KMP}}[i] = k$, where k is the length of the longest proper border of $P[0..i)$ *such that* $P[k] \neq P[i]$, or -1 if there is no such border.

   (a) Compute both failure functions for the pattern `ananassana`.

   (b) Give an example of a text, where some text character is compared three times by the MP algorithm but only once by the KMP algorithm when searching for `ananassana`.

3. Modify Algorithm 3.6 on the lecture notes to compute $fail_{\text{KMP}}$ instead of $fail_{\text{MP}}$.

4. Let us analyze the average case time complexity of the Horspool algorithm, where the average is taken over all possible patterns of length m and all possible texts of length n for the integer alphabet $\Sigma = \{0, 1, \ldots, \sigma - 1\}$ where $\sigma > 1$. This is the same as the expected time complexity when each pattern and text character is chosen independently and randomly from the uniform distribution over Σ.

   (a) Show that the average time spent in the loop on line 7 is $\mathcal{O}(1)$.

   (b) Show that the probability that the shift is shorter than $\min(m, \sigma/2)$ is at most $1/2$.

   (c) Combine the above results to show that the average time complexity is $\mathcal{O}(n/\min(m, \sigma))$.

5. The multiple exact string matching problem is to find the occurrences of multiple patterns $P_1, P_2, \ldots, P_k$ in a text T. The trivial solution is to find each pattern separately. Show how the following algorithms can be modified to solve the problem more efficiently:

   (a) Shift-And

   (b) Karp-Rabin

6. A don't care character # is a special character that matches any single character. For example, the pattern `#oke#i` matches `sokeri`, `pokeri` and `tokeni`.

   (a) Modify the Shift-And algorithm to handle don't care characters.

   (b) It may appear that the Morris–Pratt algorithm can handle don't care characters almost without change: Just make sure that the character comparisons are performed correctly when don't care characters are involved. However, such an algorithm would be incorrect. Give an example demonstrating this.