3rd tutorial

AC Automaton. Construct the AC search automaton for words `dar`, `radar`, `adam`, `a`.

Too many occurences. Find an example input of patterns and text which has asymptitically more than linear number of occurences. Specifically, for every n, show that there is input to the Text Search problem such that $|T| + \sum_i |P_i|$ is $\Theta(n)$, and the number of occurences is not $\mathcal{O}(n)$.

Naive jumps. Consider the simplified AC algorithm which doesn't use shortcut edges and always walks along back edges all the way to the root. Show an example of an input where this algorithm is asymptotically slower than the actual AC algorithm.

Frequencies of Occurences. Describe an algorithm which, in time $\mathcal{O}(|T| + \sum_i |P_i|)$, outputs an array of frequencies, i.e. f_i is how many times P_i appears in T. (Notice that the complexity should not depend on the number of occurences!)

2D Search. Given an $n \times n$ matrix A, decide whether it contains a given $m \times m$ submatrix B as a contiguous submatrix.

Censorship. A censor receives a set of forbidden substrings and a text. He always looks for the left-most occurence of a forbidden substring in the text (this is an occurence with the left-most end; if there are multiple, choose the longest one), cuts it out of the text, and repeats the process. Show how to produce a censored text in linear time.

Dynamic Search. Design a data structure for dynamic searching. The pattern P is fixed, but the characters in the text T may change, and the data structure should always quickly answer whether the current text contains the pattern.

Shortest Word without Occurence. Given a set of patterns $P_1, \ldots, P_n$ over the alphabet $\{\mathtt{a}, \ldots, \mathtt{z}\}$ and a number $l \in \mathbb{N}$, compute the lexicographically smallest word of length l which does not contain any of the patterns.

Shortest Universal Word. Given are again patterns $P_1, \ldots, P_n$ over the alphabet $\{\mathtt{a}, \ldots, \mathtt{z}\}$. Find the shortest word which contains all patterns. Aim for an algorithm with complexity $\mathcal{O}(2^n \sum_{i=1}^{n} J_i)$.

d-dimensional search. Same as the task *2D search*, but generally in d dimensions.

Fibonacci words. Let us define Fibonacci words as follows: $F_0 = \mathtt{a}$, $F_1 = \mathtt{b}$, $F_{n+2} = F_n F_{n+1}$. Design an algorithm which, in a given string over the alphabet $\{\mathtt{a}, \mathtt{b}\}$, finds the longest Fibonacci subword.