---

1. **(Jumbled pattern matching)** Develop an algorithm that finds all substrings of a text T which can be obtained by shuffling the letters of a pattern P.
2. **(Longest repeated substring problem)** Develop a linear-time algorithm that receives a string T and outputs its longest substring that has at least two occurrences in T.
3. **(Generalised suffix tree)** A generalised suffix tree for two strings S_1, S_2 contains suffixes of S_1 and of S_2. Assuming you know a linear-time algorithm that builds a suffix tree, develop an linear-time algorithm that constructs the generalised suffix tree.
4. **(Longest common substring)** Given two strings S_1, S_2, develop an algorithm that finds a longest string X that is a substring of both S_1 and S_2.
5. **(Longest palindrome)** Develop an algorithm that finds a longest palindrome occurring in a string T. (A palindrome is a string equal to its reverse, for example, "anana").
6. **(From suffix tree to suffix array and back)** Let T be a string of length n. The suffix array SA of T stores a permutation of $1, 2, \dots, n$. $SA[i] = j$ iff $T[j, n]$ is the i-th suffix of T in the lexicographic order (the lexicographic order is the usual dictionary order, for example, $a < ab < b$). The longest common prefix array stores longest common prefixes of suffixes of T that are consecutive in the lexicographic order.
   (a) Show how to construct the suffix array and the longest common prefix array given the suffix tree.
   (b) Show how to construct the suffix tree given the suffix array and the longest common prefix array.
7. **(Pattern search in a suffix array)**
   (a) Show that all suffixes starting with P form an interval of a suffix array.
   (b) Show how that the suffix array can be used to find all occurrences of a pattern P in a text T in $O(|P| \log |T|)$ time.
8. **(2D pattern matching)** Assume that both P and T are 2D tables. Develop an algorithm that finds all occurrences of P in T.
9. **(Karp-Rabin pattern matching algorithm)** Given a string $S = S[1]S[2]\dots S[n]$, its Karp-Rabin fingerprint is defined as $S[1] + S[2] \cdot r + S[n] \cdot r^{n-1} \mod p$, where p is a prime number and r is an integer in $[0, p-1]$.
   (a) Show how to compute the Karp-Rabin fingerprints of all n-length substrings of a string T in linear time.
   (b) Hence, develop a linear-time pattern matching algorithm. Can this algorithm have false positives or false negatives ?

(c) Let p be a prime number $> n^3$ and r be a random integer in $[0, p-1]$. Show that the probability of a collision for Karp-Rabin fingerprints is $< 1/n^2$. Hence estimate the error probability of the pattern matching algorithm.

10. **(Lempel-Ziv factorization)** The Lempel-Ziv factorization of a string T is defined as $T = f_1 f_2 \dots f_z$, where f_i is the longest string occurring in $f_1 f_2 \dots f_i$ at least twice or a single letter that does not occur in $f_1 f_2 \dots f_{i-1}$. For example, the Lempel-Ziv factorization of $T = ababbab$ is $a\ b\ ab\ bab$. The Lempel-Ziv factorization can be used for data compression : instead of storing the string $T = ababbab$ we can store its encoding $(a, 1)(b, 1)(1, 2)(2, 3)$ (the first integer is a position of the first occurrence of a factor, the second is its length). Develop an algorithm that computes the Lempel-Ziv factorization of a string T.