Exercises

1. Let x be a non-empty word. Let u be the shortest word such that x is a prefix of ux. Show that $|u| = period(x)$.

   Prove that $period(x^2) = |x|$ iff x is primitive (i. e. cannot be written u^k for $k > 1$) iff x^2 contains only two occurrences of x.

2. Show that any non-empty word x can be uniquely written as uk, where u is a primitive word and $k > 0$. The word u is called the root of x and k is called its exponent. Show that the set $\{y \in A^* : xy = yx\}$ is equal to u^* where u is the root of x.

   Two words y and z are said to be conjugate if they can be written $y = uv$ and $z = vu$. Show that this defines an equivalence relation on A^*. Show that two words are conjugate iff their roots are conjugate. Show that the equivalence class of x has cardinality $|x|/k$ where k is the exponent of x.

3. Prove the following properties of Fibonacci words Fib_k (for large enough k):

   - $Fib_k Fib_{k+1} = c(Fib_{k+1} Fib_k)$, where c is the operation of exchanging two last letters of the text,
   - Fib_k is an "almost" palindrome, i.e. after cutting the last two letters it becomes a palindrome.

   Find a morphism h, defined from $\{a, b\}^*$ to itself, such that $Fib_k = h^{k-2}(a)$ for $k > 1$.

4. * Show that if the non-empty word u^2 is factor of a Fibonacci word then $|u|$ is a Fibonacci number. Deduce that Fibonacci words have no non-empty factor of the form u^4.

   Prove that a Fibonacci word of length n contains $O(\log n)$ squares (of primitive word).

   {first statement is by Séébold; for the second, see [Cr 81]}

5. A set $C \subseteq A^*$ is a uniquely decipherable code iff for any two sequences of elements of C, $u = (u_1, u_2, \ldots, u_k)$ and $v = (v_1, v_2, \ldots, v_k)$, the equality $u_1 u_2 \ldots u_k = v_1 v_2 \ldots v_k$ implies $u = v$.

   Let $B \subseteq A^*$. Show that the minimal generator of B^* is $C = (B^* - \{\epsilon\})^2 - (B^* - \{\epsilon\})$. Show that C is a uniquely decipherable code iff, for all $w, y, z \in A^*$, $y, z, yw, wz \in B^*$ implies $w \in B^*$.

   Show that, for any $B \subseteq A^*$, there exists a unique uniquely decipherable code C such that $B^* \subseteq C^*$ and that, for any uniquely decipherable code C' satisfying $B^* \subseteq C'^*$, we have $C^* \subseteq C'^*$ (i. e. C^* is minimal for inclusion). The set C is called the *uniquely decipherable code induced by* B.

   Propose an algorithm to compute the uniquely decipherable code induced by B.

   {see [Lo 83], or [BP 85] for further studies on the subject}

6. * Defect theorem. Let B be a subset of A^* that is not a uniquely decipherable code. Show that the code C induced by B (see previous exercise) satisfies $|C| < |B|$.

   Deduce from the statement:

   - the two-word set $\{x, y\}$ is a uniquely decipherable code iff $xy \neq yx$,
   - two words y and z satisfy $yz = zy$ iff they are powers of a same word,
   - the same conclusion holds iff $y^i = z^j$ for two integers i and j.

7. A word w is called borderfree if its only borders are itself and the empty word, or equivalently *period*$(w) = |w|$. Assume that the word x has a shortest non-empty border u. Show that u is borderfree, and that either $x = u$ or $x = uvu$ for some word v.

   Let $B(x) = \{u \in A^* : u \text{is prefix of} x, \text{and} u \text{is borderfree}\}$. Show that the minimal generator of $B(x)^*$ is a uniquely decipherable code $C(x)$.

   Let $x = u_1 u_2 \ldots u_k$ (with u_i's in $C(x)$) be the unique factorization of x according to $C(x)$. For instance, the decomposition of $abacaba$ is $(abac, ab, a)$. Show that u_k is the shortest borderfree prefix of x, and that u_1 is the longest borderfree prefix of x. Give a linear time algorithm to compute the decomposition $(u_1, u_2, \ldots, u_k)$ of x.

   {Hint: use the failure function of the MP algorithm}

8. Let *Maxsuf*(x) be the maximal suffix of the word x ($\neq \epsilon$) with respect to lexicographic ordering. Let $u =$ *Maxsuf*(x), and $va =$ *Maxsuf*(xa), for some non-empty x and letter a. Prove that v is a border of u.

   Let $(u_1, u_2, \ldots, u_k)$ be the sequence of borders of *Maxsuf*(x) in decreasing order of lengths. Define letters a_i's and words v_i's by

   $$\textit{Maxsuf}(x) = u_1 a_1 v_1 = u_2 a_2 v_2 = \ldots = u_k a_k v_k.$$

   Show that $a_1 \leq a_2 \leq \ldots \leq a_k$.

   For a letter $a \leq a_1$, show that *Maxsuf*(xa) = *Maxsuf*(x)a. Otherwise (when $a > a_1$), show that *Maxsuf*(xa) $= u_g a_g$ for the largest integer g such that $a_g \leq a$. Develop a full proof of algorithm *Maxsuf* of Chapter 3.

9. An overlap is a word w that satisfies *period*(w) $< |w|/2$. A simple overlap is a word of the form $auaua$ for a letter a and a word u. Show that a word x contains an overlap factor iff it contains a simple overlap factor.

   A word that contains no factor that is an overlap (or a simple overlap) is said to be *overlap-free.*

   Let x an overlap-free word on the alphabet $\{a, b\}$. Show that if $|x| \geq 7$, x has a unique factorization as gud with $g, d \in \{\epsilon, a, b, aa, bb\}$ and $u \in \{ab, ba\}^*$.

   Let h be the (Thue-Morse) morphism defined by $h(a) = ab$ and $h(b) = ba$. For the word x, show that, for some integer r and word u, $x = g_1 g_2 \ldots g^r h^{r-1}(u) d_r \ldots d_2 d_1$, with $|u| < 7$ and for all $i = 1, \ldots, r$, $g_i, d_i \in \{\epsilon, hi-1(a), hi-1(b), hi-1(aa), hi-1(bb)\}$.

   * Design a linear-time algorithm to test the existence of an overlap factor in a given text.

   {see [Kf 86] for the algorithm; see also [RS 85]}

10. Let w be a non-empty word. Assume that w has a strict border u, and that u has a strict border v. Then, show that $|w| > |u| + |v| + 1$.

    Prove that $delay(|x|) \leq \log_\Phi(|x| + 1)$ (Lemma 3.4), where $\Phi = (1 + \sqrt{5})/2$. Prove that the delay is $O(\log |x|)$ for Fibonacci patterns.

    {Hint: Fibonacci words satisfy the inequality, for large enough k, $|Fib_k| = F_k \geq F_{k-2}$}

11. Design a linear-time algorithm computing the table *t_Bord* used by MPS algorithm (Chapter 3).

12. * Write real-time versions of KMP and MPS algorithms on a random access machine.

    {see [Ga 81]}

13. * Give a linear-time algorithm (independently of the alphabet) that, for each position i in the word *text* finds the longest suffix ending at x and belonging to the language $\{x, y, z\}^*$, where x, y, and z are given texts. The size of the problem is the total size of x, y, z, and text.

    {see [NC 92]}

14. Assume that $a^k b$ ($a, b \in A$, $a \neq b$, $k > 0$) is a prefix of pattern x. Consider the following variant of KMP algorithm, which uses the same failure table: the scan is done from left to right on the suffix $x_{k+1} \ldots x_m$ of x; if the suffix occurs in text at the current position, we check if it is preceded by a^k and report a possible match.

    Write in full details the above algorithm. Prove that the search makes at most $1.5|text|$ comparisons.

    {idea in [AC 91]}

    * Show that considering other interrupted periodicities in addition to the first one as above leads to a searching algorithm making at most $4|text|/3$.

    For finding all occurrences of the pattern aba in a text of length n, prove that any algorithm has to make at least $4n/3 - c$ comparisons, where c is a constant.

    {see [CGG 90]; other upper bounds are in [CH 92] and [ZP 92]}

15. Let $k \geq 2$. A k-hrp of a string x is a primitive word u such that u^k is prefix of x (Sections 13.2 and 13.3).

    Let $k > 2$. Show that if u and v are k-hrp's of x with $|u| < |v|$, then $(k-1)|u| < |v|$. Deduce that x has at most $\log_{k-1}|x|$ k-hrp's.

    * Let $k = 2$. Show that if u, v, et w are 2-hrp's of x with $|u| < |v| < |w|$, then $|u| + |v| < |w|$. Deduce that x has at most $\log_\Phi |x|$ 2-hrp's ($\Phi = (1 + \sqrt{5}/2)$. What are the words reaching the worst case ? How many 2-hrp's have Fibonacci words and their prefixes ?

    {see [CR 93]}

16. A language $L \subseteq A^* - \{\epsilon\}$ is said to have the left cancellation property if $u, uv \in L$ implies $v \in L$.

    Let $P \subseteq A^* - \{\epsilon\}$ be a language such that P^* has the left cancellation property. Assume that there is algorithm that, given a string x of length n, can find a prefix u of x, $u \in P$, or prove that no such prefix exists, in time $O(n)$. Then, prove that it can be tested if $x \in P^*$ in time $O(n)$.

Show that the set Pal of even palindromes has the left cancellation property. Gives a linear time algorithm for the recognition of Pal^*.

* Write a real-time algorithm for the recognition of even palindromes.

{see [KMP 77]}

17. Design a linear-time version of KMP algorithm for a Turing machine.

    {see [FP 74]}

18. Prove that in the search phase of BM algorithm a symbol at a given position in the text can be inspected at most a logarithmic number of times.

19. Give a worst-case pattern for Turbo_BM algorithm (search phase).

20. Give an example of text for which the uncompacted position tree (each edge is labeled by a single symbol) has quadratic size.

21. Implement the three suffix tree constructions of Chapter 5. Compare them according to time and space actually required.

    Implement the suffix array method of Chapter 5. Compare with above implementations.

22. Design a linear-time algorithm for the computation of suffix trees using only the table $sext$ of Section 6.4 (without tables $link$ and $test$).

23. Let T be a tree whose edges are labelled with symbols. Assume that the edges outgoing the same node are labelled with distinct symbols. Let $Suf(T)$ be the set of all suffixes of the paths of T. Construct an $O(n \log n)$ time algorithm which builds the compressed tree representing all strings in $Suf(T)$.

24. Let T be a tree whose edges are labelled with symbols. Let $Rev(T)$ be the set of all reverses of paths of T. Construct an $O(n \log n)$ time algorithm which builds the compressed tree representing all strings in $Rev(T)$.

25. What is the complexity of the problems of the two preceding exercises if, instead of a tree, we have a directed acyclic graph ? Note that the resulting tree can be very large (investigate how large). Denote its size by s. Can you get $O(s)$ time algorithms ?

26. Consider a directed acyclic graph with one root and one sink. Assume that edges outgoing a same node has distinct labels. The set of patterns consists of all labels of paths from the root to the sink. Assume the total size of the graph is m. Can you construct an $O(m+n)$-time algorithm for pattern-searching in a text of length n ?

27. Let $twice(u)$ be the size of the largest suffix of u which occurs in x at least twice. Design a linear-time algorithm that, for each prefix u of a given word x, finds the value of $twice(u)$ (assume that the size of the alphabet is fixed).

    Compute the function $twice(x)$ for a single word x in linear time, and simultaneously in constant space.

28. Let us define the function $k - times(x)$. Its value is the size of the largest prefix of x which occurs in x at least k times. Compute $k - times(x)$ in linear time. The parameter k can be a function of n (the alphabet is fixed).

29. Let P be a family of disjoint subsets of $\{1, 2, \ldots, m\}$ satisfying the non-overlap property: if two subsets from P have a common element then one of them is included in the other. Prove that P has less than $2m$ elements. How this is related to the size of suffix dawg's ?

    Assume that $|pat| > 2$. Prove that the number of states of the suffix dawg $DAWG(pat)$ is at most $2|pat|-1$, and that the bound is reached only if pat belongs to ab^*, where a, b are distinct letters.

    Assume that $|pat| > 3$. Prove that the number of edges of the suffix dawg $DAWG(pat)$ is at most $3|pat|-4$, and that the bound is reached only if pat belongs to ab^*c, where a, b, c are letters with $a \neq b$ and $b \neq c$.

    {see [BBEHCS 85] and [Cr 86]}

30. Give exact bounds, in term of m, of the size of the minimal deterministic automaton—factor automaton—accepting the set of all factors of a text of length m. Note that this automaton can be smaller than the minimal deterministic automaton accepting suffixes of the same text.

    On a fixed alphabet, design a linear on-line algorithm that builds the factor automaton of a text.

    {see [BBEHCS 85] and [Cr 86]}

31. Give an algorithm to compute a suffix dawg for a finite set of patterns.

    {see [BBEHM 87]}

32. * Let us call a compact dawg a dawg in which edges are labelled by strings and no node has only one outgoing edge, except possibly the initial state. Design a linear-time algorithm for building the compact suffix dawg of a text.

33. Let $G = DAWG(pat)$, and $G' = DAWG(pat^R)$. Prove that there is a one-to-one correspondence between essential nodes of G and G' (internal nodes of outdegreee larger than one).

34. Design efficient algorithms to compute, for a given text w, the number of distinct

    - primitive factors of w,
    - squares of primitive words, factors of w,
    - squares, factors of w,
    - even palindromes, factors of w.

35. Design an efficient algorithm to test if a given word is a "squarestar" (i.e. product of non- empty squares).

    Design an efficient algorithm to test if a given word is a "cubestar" (i.e. product of non- empty cubes).

36. Implement and compare Aho-Corasick and Multi_BDM multi-pattern matching algorithms (Sections 7.1 and 7.2).

37. Show that the smallest deterministic automaton accepting all words over alphabet $\{a, b\}$ whose 7-th letter from the end is a, has 128 states.

    Generalize the example.

38. Design an efficient algorithm to compute the number of all distinct subsequences of the word. What is the maximal value of this number for a text of length 10.

39. On a fixed alphabet, design an algorithm that builds the minimal automaton accepting the set of subsequences of a text. What are the time and space complexities ?

    Design algorithms that use this kind of automaton to find a longest common subsequence of two words, and a shortest word that is subsequence of only one of two words.

    {see [HC 86]; an unpublished algorithm for the last problem is by Simon as an implementation of Theorem 6.2.11 in [Lo 83] Chapter 6}

40. Construct 2dpda's accepting the languages:

- $L_{prefpal} = \{ww^Ru : w \text{is non-empty}\}$,
- $L_{prefpal3} = \{ww^Ruu^Rvv^Rz : w, u, v \text{are non-empty words}\}$,
- $L_{pal2} = \{ww^Ruu^R : w, u \text{are non-empty words}\}$.

41. Construct 2dpda's accepting the languages $\{a^nb^m : m = 2^n\}$, $\{a^nb^m : m = n^4\}$, and $\{a^nb^m : m = \log^* n\}$.

42. * Construct a 2dpda accepting the language

$$L = \{1^n : n \text{is a square of an integer}\}.$$

{see [Mo 85]}

43. Write a version of the Karp-Miller-Rosenberg algorithm for the two-dimensional case (and generally for k dimensions, where k is a constant) working in $O(N \log N)$, where N is the total size of the input.

44. Prove the combinatorial properties required for the correctness of the linear-time constant-space algorithm for *righttest* (Section 8.2). Give a formal proof of algorithm *righttest*.

Prove that $righttest(u,v)$ is computed in time $O(|v|)$ by the algorithm.

{see [ML 79] or [ML 84]}

45. Prove that the f-factorization of texts introduced in Section 8.2 can be computed in a linear time using suffix trees or suffix dawg's.

According to the f-factorization $(v_1, v_2, \ldots, v_m)$ of *text*, prove that this word contains a square iff, for some k, one of the following conditions holds:

- $pos(v_k) + |v_k| \geq |v_1v_2 \ldots v_{k-1}|$ (self-overlapping of v_k),
- or $lefttest(v_{k-1}, v_k)$,
- or $righttest(v_{k-1}, v_k)$,
- or $righttest(v_1v_2 \ldots v_{k-2}, v_{k-1}v_k)$.

{see [Cr 86]}

46. Estimate the number of comparisons made by Manacher algorithm (Section 8.3).

Give a similar algorithm for computing the radii of odd palindromes, and running in linear time.

{see [Ma 75]}

47. Write an efficient algorithm computing the number of all distinct (as strings) palindromes contained in a given text.

48. Give a full proof of the Lemma 8.14:

$$parse1(text) \in \{first1(text), 2first1(text) - 1, 2first1(text) + 1\}.$$

49. Compute tables of all longest palindromes starting or ending at positions $i = 1, 2, \ldots, n$ of a word of length n by a linear-time algorithm.

50. * Design a efficient algorithm checking membership of a given text in PAL_k.

    {linear time for this problem is an open question}

51. Given n subintervals of $[1 \ldots n]$, design a parallel algorithm that computes which of them are maximal (in the sense of inclusion) in $O(\log n)$ time with n processors.

    {see [GR 88]}

52. Given a linked list of n elements stored in an array of size n, design a parallel algorithm that computes for each element its rank in the list in $O(\log n)$ time with n processors. Has the algorithm optimal speed-up ?

    {see [Ma 89]}

53. Prove Lemma 9.2.

    {see [GR 88]}

54. * Implement the Ziv-Lempel compression algorithm in parallel that works in $O(\log n)$ time with $O(n)$ processors of a CRCW PRAM.

    {hint: use suffix trees, see [CR 91a]}

55. Design an efficient (serial) algorithm computing the edit distance when the costs of one-symbol manipulations depend on the symbols involved.

56. Give a complete algorithm for the computation of a longest common subsequence of two texts, and not only its length (Chapter 11).

57. Show how to reduce in linear time the string-matching problem with don't care symbols to the tree-pattern-matching problem.

58. Consider the pair of words (abb, baa). A replacement in a word is a replacement of an occurrence of abb by baa, or a replacement of an occurrence of baa by abb. Two words are equivalent iff one can be obtained from the other by a sequence of such replacements. Construct a linear-time and constant-space algorithm that checks whether two words are equivalent. What happens for the pair of strings (aa, bb) ? Does the complexity change for the pair (a^k, b^j) ?

59. Implement efficiently the operations *CLASS*, *SPLIT*, *UNION* for intervals of integers used by Hunt-Szymanski algorithm (Section 11.2).

    {hint: use B-trees, see [AHU 83]}

60. Estimate the number of symbol comparisons made by the algorithm that checks the cyclic-equivalence of two words in linear time and constant-space (Chapter 15). What is the maximum number of comparisons this algorithm makes for texts u = 111..1201 and w = 1111...120.

61. Prove the Lyndon Factorization Theorem (Theorem 15.8)

    Show that if u and v are Lyndon words with $u < v$, then, for any $k, l > 0$, $u^k v^l$ is a Lyndon word. {Hint: use Lemma 15.9}

    Show that w is a Lyndon word of length greater than 1 iff $w = uv$ for two Lyndon words u and v satisfying $u < v$. Give a linear-time algorithm to compute the factorization uv of w.

    {see [Lo 83]}

62. Prove Lemma 15.12.