DOI 10.1007/s10958-018-3921-y
Journal of Mathematical Sciences, Vol. 233, No. 1, August, 2018

INDEXING AND QUERYING CHARACTER SETS IN ONE- AND TWO-DIMENSIONAL WORDS

D. Belazzougui, R. Kolpakov, and M. Raffinot UDC 519.712.43

Abstract. We give a detailed review of results obtained for a relatively new problem of finding, indexing, and querying character sets, which are called fingerprints in fragments of one- and two-dimensional words, and explain basic ideas used for obtaining these results.

1. One-Dimensional Fingerprints

Let Σ be a finite ordered alphabet of size σ, and $S = s_1 \dots s_n$ be a string of n letters from Σ. The length n of the string S is denoted by $|S|$. The set of all strings over Σ (including the empty string) is denoted by Σ^*. For convenience, we range all letters from Σ by their ranks between 0 and $\sigma - 1$. The rank of a letter a in Σ is denoted by $r_\Sigma(a)$. A substring $s_i s_{i+1} \dots s_j$ is called a *factor* of S and denoted by $S\langle i; j\rangle$. The *fingerprint* $f(S)$ of a string S is the set of all distinct letters in S. By extension of the fingerprint notion, the fingerprint $f_S(i, j)$ is the set $f(s_i s_{i+1} \dots s_j)$ of all distinct letters in the factor $S\langle i; j\rangle$. The factor $S\langle i; j\rangle$ is called a *location* of the fingerprint $f_S(i, j)$. A location $S\langle i; j\rangle$ of a fingerprint f is called *maximal* if

(1) $s_{i-1} \notin f$ for $i > 1$;
(2) $s_{j+1} \notin f$ for $j < n$.

Note that any location of a fingerprint is uniquely extended to some maximal location of this fingerprint. We denote by $\mathcal{F}(S)$ the set

$$\{f \subseteq \Sigma \mid \exists\, i, j \colon f = f_S\langle i, j\rangle\}$$

of all distinct fingerprints for factors of S, and by $\mathcal{L}(S)$ the set of all maximal locations for all fingerprints from $\mathcal{F}(S)$. It is not hard to show that $|\mathcal{F}(S)| \le |\mathcal{L}(S)| \le n|\Sigma|$. In this paper, given a string S, we consider the following three algorithmic problems:

(1) compute the set $\mathcal{F}(S)$;
(2) for a given $f \subseteq \Sigma$, check if $f \in \mathcal{F}(S)$;
(3) for a given $f \subseteq \Sigma$, find all maximal locations of f in S if $f \in \mathcal{F}(S)$.

Efficient solution of these problems has many applications in information retrieval, computational biology, and natural language processing. Further we naturally assume that the alphabet Σ is the alphabet of the string S, i.e., $|\Sigma| \le n$.

The first question arising in relation to the considered problems is an effective representation of fingerprints. Note that a fingerprint f can be represented by a binary table of F of size σ, called the *fingerprint array*: if f contains a character α, then $F[r_\Sigma(\alpha)] = 1$, otherwise $F[r_\Sigma(\alpha)] = 0$. However, in the most interesting case for an alphabet Σ of big size, this representation can be too memory expensive. In [2], the naming technique for fingerprints representation was introduced. The naming technique is used to give a unique name to each fingerprint of a substring of S. For describing this technique, we assume for simplicity, but without loss of generality, that σ is a power of two. To name a given fingerprint, we consider a stack of $\log \sigma + 1$ arrays $F_0, F_1, \dots, F_{\log \sigma}$ on top of each other (by logarithm we assume the logarithm to base 2). The lowest array F_0 is the fingerprint array containing only the values [0] or [1].

Translated from Fundamentalnaya i Prikladnaya Matematika, Vol. 20, No. 6, pp. 3–16, 2015.

Each other array F_i contains half the number of names that the array F_{i-1} immediately below it, i.e., each name in F_i corresponds to a pair of consecutive names in F_{i-1}. Thus, the array F_i contains the names of subarrays of length 2^i in the fingerprint array, and these names are computed proceeding from pairs of names of the fingerprint subarrays of length 2^{i-1} that are contained in F_{i-1}. The highest array $F_{\log \sigma}$ contains a single name that is assigned to the fingerprint and is called the *fingerprint name*. Figure 1 shows a simple example with $|\Sigma| = 8$.

<table>
<tr><td colspan="8">[7]</td></tr>
<tr><td colspan="4">[5]</td><td colspan="4">[6]</td></tr>
<tr><td colspan="2">[2]</td><td colspan="2">[2]</td><td colspan="2">[3]</td><td colspan="2">[4]</td></tr>
<tr><td>[1]</td><td>[0]</td><td>[1]</td><td>[0]</td><td>[1]</td><td>[1]</td><td>[0]</td><td>[0]</td></tr>
</table>

Fig. 1. Naming example.

It is easy to note that the naming technique can be used for naming all fingerprints in S so that all identical subarrays of fingerprints are named by the same name. In this case, if we have two fingerprints f and f' that differ by only one letter, then for computing the name of f' from the name of f it is enough to recompute no more than $\log \sigma$ names of subarrays of f'. Using this observation, in [2] an $O(n\sigma \log \sigma \log n)$ time algorithm for solving problem (1) was proposed. Problems (2) and (3) are solved, respectively, in [2] in $O(\sigma \log n)$ time and $O(\sigma \log n + K)$ time, where K is the size of output. Further the naming was improved in [6]. The improvement consists in synchronous recomputations of names for all fingerprint subarrays of the same length required for computation of names of all fingerprints in S. By this improvement the time complexity of solving problem (1) was decreased to $O(\min\{n\sigma \log \sigma, n^2\})$.

Further improvements in solving of problems (1)–(3) were obtained in [7]. One of motivations for these improvements is that the complexity bounds for solving problem (1) obtained in [2,6] are independent of the sizes of $\mathcal{F}(S)$ and $\mathcal{L}(S)$, although many string families have few fingerprints or few maximal locations. As an example, we can consider a family of strings $\{W_1, W_2, \ldots\}$, where W_k is the string over the alphabet $\Sigma_k = \{a_1, a_2, \ldots, a_k\}$ defined by recursive relations $W_1 = a_1$ and $W_k = W_{k-1}a_kW_{k-1}$ for $k > 1$. It can be checked that $|W_k| \cdot |\Sigma_k| = k \cdot (2^k - 1)$ and $|\mathcal{L}(W_k)| = 2^{k+1} - (k + 2)$, so $|\mathcal{L}(W_k)| = o(|W_k| \cdot |\Sigma_k|)$. To describe the algorithm proposed in [7] for solving problem (1), we have to give some auxiliary definitions.

Without loss of generality, we assume that the input string S does not contain two consecutive occurrences of the same character. For the sake of convenience, we add to the string S a last character $s_{n+1} = \#$ that does not appear in the string S, i.e., $S = s_1 \ldots s_n \#_{n+1}$. Let i and j be positions in S such that $1 \le i \le j \le n+1$. We define $\mathrm{fo}_S(i, j)$ as the string formed by concatenating the leftmost occurrences of all distinct characters in the factor $S\langle i; j\rangle$ considered from left to the right. For instance, if

$$S = a_1b_2a_3c_4e_5a_6b_7a_8c_9d_{10}\#,$$

then $\mathrm{fo}_S(3, 9) = aceb$ and $\mathrm{fo}_S(5, 10) = eabcd$. Let, for some position i in S, j be the minimum position greater than i such that $s_j = s_i$ if it exists, and $j = n+2$ otherwise. Then we define $\mathrm{lfo}_S(i) = \mathrm{fo}_S(i, j-1)$. For instance, if

$$S = a_1b_2c_3a_4d_5a_6b_7a_8c_9b_{10}e_{11}\#_{12},$$

then $\mathrm{lfo}_S(1) = abc$ and $\mathrm{lfo}_S(5) = dabce\#$. By $\mathrm{lfo}'_S(i)$ we denote the string $\mathrm{lfo}_S(i)$ without the last symbol. It can be shown that there exists an one-to-one correspondence between all maximal locations from $\mathcal{L}(S)$ and prefixes of all strings $\mathrm{lfo}'_S(i)$ such that the set of symbols of the prefix corresponding to a maximal location is the fingerprint of this location. Thus, for finding all fingerprints from $\mathcal{F}(S)$ we can compute all strings $\mathrm{lfo}_S(i)$ for S. It can be done in $O(|\mathcal{L}(S)| + n)$ by traversing symbols of S from left to the right (see Fig. 2). Then we compute the names of all fingerprints in $\mathcal{F}(S)$, using the naming technique from [6], which can be done in $O\big((|\mathcal{L}(S)| + n) \log \sigma\big)$ time. Thus, the total time for solving of problem (1) is $O\big((|\mathcal{L}(S)| + n) \log \sigma\big)$. A slight improvement $O(|\mathcal{L}(S)| \log \sigma + n)$ for this time bound was obtained in [8].

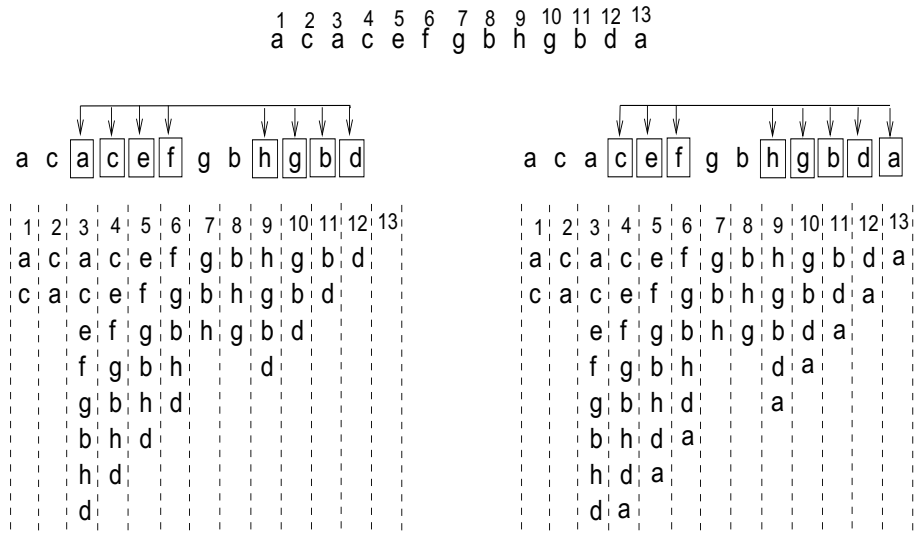

Fig. 2. A (schematic) step of traversing the last symbol a in the procedure of computing $\mathrm{lfo}_S(i)$ for $S = acacefgbhgbda$. We add the character a in the table containing the strings $\mathrm{lfo}_S(i)$.

In [7], the time for solving problems (2) and (3) was also improved, using a new data structure, which is called a *fingerprint tree*. The fingerprint tree of S (denoted by $\mathrm{FT}(S)$) is a binary tree data structure formed by all fingerprint arrays from $\mathcal{F}(S)$, where common initial segments of fingerprint arrays are merged. Thus, each edge of $\mathrm{FT}(S)$ corresponds to some subarray of a named fingerprint array. Note that this subarray can be represented as the concatenation of some suffix U of a named subarray F_l and some prefix V of a named subarray F_r, where the subarrays F_l and F_r can be chosen as minimal as possible, i.e., $|U| > |F_l|$ and $|V| > |F_r|$. Thus, the subarray corresponding to the edge of $\mathrm{FT}(S)$ is uniquely identified by the tuple (n_l, n_r, l, r), where n_l (n_r) is the name of F_l (F_r) and l (r) is the length of U (V) (see Fig. 3) and, moreover, can be computed from this tuple in $O(l + r)$ time, so we label the edge by this tuple.

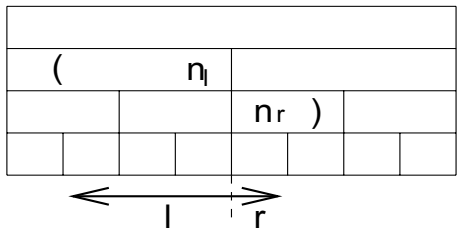

Fig. 3. Label of an edge. (n_l, n_r) is the pair of names of the shortest consecutive fingerprint subarrays covering the segments U of length l and V of length r.

It is shown in [7] that $\mathrm{FT}(S)$ can be constructed in $O(|\mathcal{F}(S)| \log \sigma)$ time and, for a given $f \subseteq \Sigma$, one can check in $O(\sigma)$ time if f is contained in $\mathrm{FT}(S)$. Thus, problems (2) and (3) can be solved, respectively, in time $O(\sigma)$ and $O(\sigma + K)$, where K is the size of output.

The time complexity of solving problem (1) was further improved in [9]. We introduce the following notions for describing this improvement. Two maximal locations $S\langle i; j\rangle$ and $S\langle k; l\rangle$ in S are called *copies* if $s_i \dots s_j = s_k \dots s_l$. For example, in the string

$$S' = ab\underline{acadc}b\underline{acadc}eacdc$$

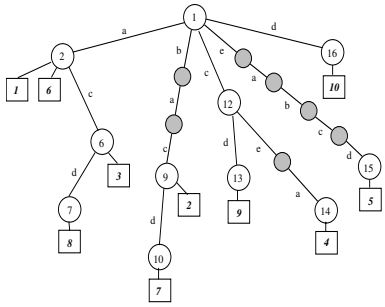

Fig. 4. The participation tree of $S'' = abaceabacd\#$.

the underlined maximal locations are copies. Note that the "copy" relation is obviously an equivalence relation. We denote by $\mathcal{L}_C(S)$ the set of equivalence classes for this relation in $\mathcal{L}(S)$. Note that $|\mathcal{L}_C(S)|$ can be significantly less than $|\mathcal{L}(S)|$. As an example, we can consider a family of strings $\{W'_1, W'_2, \ldots\}$ over the alphabet $\{a_1, a_2, \ldots\}$, where $W'_1 = a_1$ and $W'_k = W'_{k-1}(a_1 a_2 \ldots a_k)^k$ for $k > 1$. We can check that

$$|W'_k| = \frac{1}{6}k(k+1)(2k+1),$$

$$|\mathcal{L}(W'_k)| = \frac{1}{12}k(3k^3 + 2k^2 - 9k + 16) = \Theta(|W'_k|^{4/3}), \quad |\mathcal{L}_C(W'_k)| = \frac{1}{6}k(k^2+5) = \Theta(|W'_k|),$$

so $|\mathcal{L}_C(W'_k)| = o(|\mathcal{L}(W'_k)|)$.

The *participation tree* of the string S (denoted by $\mathrm{PT}(S)$) is a tree data structure formed by all strings $\mathrm{lfo}'_S(i)$, $i = 1, 2, \ldots, n$, where common initial segments the strings are merged (for convenience each edge of $\mathrm{PT}(S)$ is labeled by a single symbol). At Fig. 4 we give an example of the participation tree of the string $S'' = abaceabacd\#$.

Since all fingerprints from $\mathcal{F}(S)$ correspond to prefixes of strings $\mathrm{lfo}'_S(i)$, the tree $\mathrm{PT}(S)$ is actually a compacted representation of all fingerprints from $\mathcal{F}(S)$, i.e., for each fingerprint f from $\mathcal{F}(S)$ there exists at least one corresponding node of $\mathrm{PT}(S)$ such that f is the set of all symbols labeling edges contained in the path from the root of $\mathrm{PT}(S)$ to this node and, moreover, each node of $\mathrm{PT}(S)$ corresponds to some fingerprint from $\mathcal{F}(S)$. Thus, $\mathcal{F}(S)$ can be computed by construction of $\mathrm{PT}(S)$.

To construct $\mathrm{PT}(S)$, first we construct the suffix tree of S. The suffix tree of S (denoted by $\mathrm{ST}(S)$) is a classical tree data structure formed by all suffixes of S, where common initial segments the suffixes are merged. It is easy to see that edges of $\mathrm{ST}(S)$ correspond to some factors of S, which are uniquely identified by their starting and final positions in S. So each edge of $\mathrm{ST}(S)$ is labeled by the starting and final positions of some factor in S corresponding to this edge. Thus, $\mathrm{ST}(S)$ has size $O(n)$ and, moreover, can be constructed in $O(n \log \sigma)$ time (see, e.g., [10,13,14]). In Fig. 5 an example of the suffix tree of the string S'' is presented.

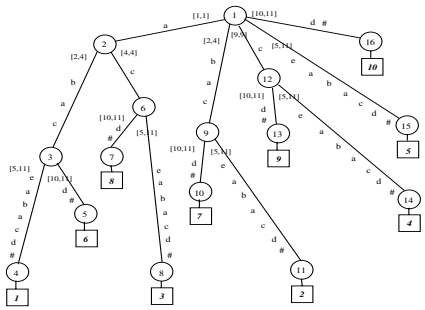

Fig. 5. The suffix tree of $S'' = abaceabacd\#$.

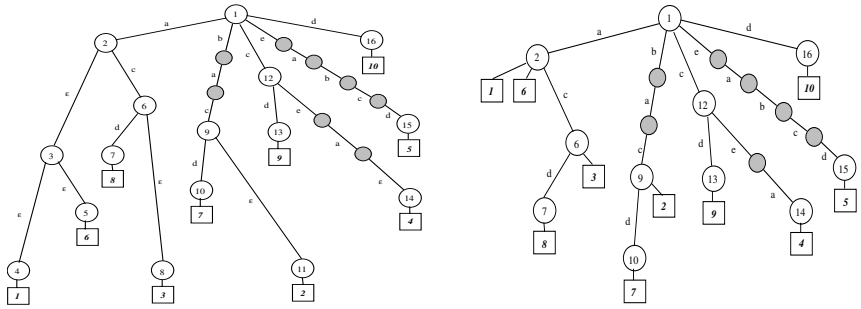

Fig. 6. From suffix tree to the participation tree (right picture) of $S'' = a_1b_2a_3c_4e_5a_6b_7a_8c_9d_{10}\#_{11}$. New nodes are in gray. Attached suffixes are shown in square boxes.

For the sake of convenience, by the label of an edge of $\mathrm{ST}(S)$ we will mean the factor corresponding to the edge and considering as string by itself. By an ancestor of an edge e in $\mathrm{ST}(S)$ we mean any edge in the path from the root of $\mathrm{ST}(S)$ to the edge e (including the edge e itself), and by the main ancestor of e we mean the first edge on this path. Let a' and a'' be two (possibly identical) symbols contained, respectively, in labels of edges e' and e'' in $\mathrm{ST}(S)$. We say that a' is an ancestor of a'' if either e' and e'' are distinct edges and e' is an ancestor of e'' or e' and e'' are the same edge and a' is before a'' in the label of this edge. In this case, a' is called the main ancestor of a'' if e' is the main ancestor of e'' and a' is the first symbol in the label of e'. To construct $\mathrm{PT}(S)$ from $\mathrm{ST}(S)$, we remove from labels of edges in $\mathrm{ST}(S)$ all symbols that have the same symbols as ancestors or have as ancestors symbols that are not main ancestors but identical to main ancestors. Using AVL-trees, this can be done by a bottom-up procedure from leaves to the root of $\mathrm{ST}(S)$ in $O(n\log\sigma + |\mathcal{L}_C(S)|)$ time. Then, in the obtained tree we remove the last symbols of all terminal edge labels (in this case, by a terminal edge we mean an edge that is labeled by a nonempty string and is an ancestor for only edges labeled by empty strings besides the edge itself) and all edges with empty labels by merging their endpoints, which can obviously be done in $O(n)$ time and by inserting new nodes replace edges labeled by strings of several symbols with chains of edges labeled by single symbols, which can be done in $O(|\mathcal{L}_C(S)|)$ time. For final computation of $\mathrm{PT}(S)$, we merge consecutively in the obtained tree all edges that have common endpoints and are labeled by the same symbols. This can be done in $O(|\mathcal{L}_C(S)|\log\sigma)$ time. In Fig. 6 we illustrate the computation of $\mathrm{PT}(S'')$ from $\mathrm{ST}(S'')$ for the string S''. Thus, $\mathrm{PT}(S)$ can be constructed from $\mathrm{ST}(S)$ in total $O\big((n + |\mathcal{L}_C(S)|)\log\sigma\big)$ time, so $\mathrm{PT}(S)$ can be constructed in $O\big((n + |\mathcal{L}_C(S)|)\log\sigma\big)$ time. Then in $O(|\mathcal{L}_C(S)|\log\sigma)$ time we can name all fingerprints presented in $\mathrm{PT}(S)$ using the naming technique from [6]. Thus, we can solve problem (1) in $O\big((n + |\mathcal{L}_C(S)|)\log\sigma\big)$ time.

We would like to note that the suffix tree can be simulated in $O(n)$ time by using the suffix array [1], so the time of construction of $\mathrm{PT}(S)$ can be reduced to $O(n + |\mathcal{L}_C(S)|\log\sigma)$. In this way, the time complexity of solving problem (1) was improved to $O(n + |\mathcal{L}_C(S)|\log\sigma)$ in [3].

Further improvements for solving problems (2) and (3) were made in [5]. In this paper, the authors use an additional data structure, which is called a *backtracking tree*. The construction of this tree is based on the simple observation that for any fingerprint $f \in \mathcal{F}(S)$ one can find in $\mathcal{F}(S)$ a (possibly empty) fingerprint f' obtained from f by removing only one symbol. Thus, any fingerprint f from $\mathcal{F}(S)$ can be provided by an arc from f to f' labeled by the removed symbol. A backtracking tree for S is a directed tree constructed in this way (the root of this tree is the empty fingerprint). This tree can be constructed in $O(|\mathcal{L}(S)|)$ time. In [5], it is shown that, by using for S a simplified version of the fingerprint tree (lexi-string trie), an additional backtracking tree, problem (2) (problem (3)) can be solved in $O\big(|f|\log(\sigma/|f|)\big)$ time and $O(|\mathcal{F}(S)|)$ space ($O(|f|\log(\sigma/|f|) + K)$ time and $O(|\mathcal{L}(S)|)$ space, where K is the size of output).

In [3], we investigate the application of the hashing technique for solving the considered problems. Note that, for some properly chosen prime P, subsets f of Σ can be identified by hash functions

$$h_X(f) = \sum_{a \in f} X^{r_\Sigma(a)} \pmod{P},$$

where $X \in \{1, 2, \ldots, P-1\}$. It can be shown that, for any natural c, $h_X(f)$ can be computed in $O(ct)$ time using $O(c\sigma^{1/c})$ space for a precomputed table. Let

$$H_P = \{h_X \mid X \in [0; P-1]\}.$$

Then the following fact can be proved.

Lemma 1. *Given a collection M of m subsets of Σ, a randomly chosen hash function $h_X \in H_P$ for $P \geq m^2\sigma$ maps injectively the collection M to the interval $[0; P-1]$ with probability at least $1/2$.*

We use also the following result described in [12].

Lemma 2. *Given a cardinal tree of N nodes over an alphabet of size σ, we can build a representation that uses $N\big(\log\sigma + \log e + o(1)\big)$ bits of space and supports the following operation in constant time: given a node p and a character a, check whether p has a child labeled with the character a and return this child if it exists.*

For effective solving of problems (2) and (3), we use a tree data structure obtained from the participation tree by merging all nodes corresponding to the same fingerprint. This data structure, which is actually equivalent to a backtracking tree from [5], is called a *fingerprint trie*. After the procedure of naming all fingerprints in $\mathcal{F}(S)$, a fingerprint trie of S can be obtained from $\mathrm{PT}(S)$ in $O(|\mathcal{L}_C(S)|)$ time. Note that we have a one-to-one correspondence between fingerprints of $\mathcal{F}(S)$ and nodes of the fingerprint trie. We use actually two different representations of the fingerprint trie. The first representation is a *backtracking function* that associates to each fingerprint f from $\mathcal{F}(S)$ the character labeling the last edge of the path in the fingerprint trie from the root to the node corresponding to f. Using Lemma 1, in $O(|\mathcal{F}(S)|$ time we can find a hash function h_X such that the values $h_X(f)$ for all fingerprints f from $\mathcal{F}(S)$ are different and implement the backtracking function as a function defined on hash values $h_X(f)$ of fingerprints f. It follows from the results of [11] that this implementation of backtracking function can be represented using just $O\Big(|\mathcal{F}(S)| \log\sigma\big(1 + o(1)\big)\Big)$ bits of space such that for any value $h_X(f)$ where $f \in \mathcal{F}(S)$ the backtracking function value at f can be retrieved in constant time (if $f \notin \mathcal{F}(S)$, the retrieved value can be arbitrary). The second representation of the fingerprint trie, which is called a *top-down trie representation*, is the cardinal tree representation proposed by Lemma 2. By Lemma 2, this fingerprint trie representation requires only $|\mathcal{F}(S)|\big(\log\sigma + \log e + o(1)\big)$ bits of space and allows for any string U to find in $O(|U|)$ time the fingerprint trie node such that U is the string of symbols labeling the edges of the path from the fingerprint trie root to this node if such node exists.

Problem (2) is solved in [3] by the following way. Let f be a given subset of Σ. First, we compute in $O(|f|)$ time the hash value $h_X(f)$ of f. At the second stage, proceeding from the value $h_X(f)$ using the backtracking function representation, we compute in $O(|f|)$ time the only possible string U that can be a string of symbols labeling the edges of the path in the fingerprint trie from the root to the node corresponding to a single fingerprint f' from $\mathcal{F}(S)$ such that $h_X(f') = h_X(f)$ if f' exists. At the third stage we compare f with the set of all symbols of U. This can be done in $O(k|f|)$ time using $O(\sigma^{1/k}\log\sigma)$ bits of space, where k is any natural number, or in expected $O(|f|)$ time with high probability using only $O(|f|\log\sigma)$ bits of space. If f is not the set of all symbols of U, then we conclude that $f \notin \mathcal{F}(S)$. Otherwise, at the last stage we search in the fingerprint trie a node such that U is the string of symbols labeling the edges of the path from the root to this node (using the top-down trie representation, this can be done in $O(|f|)$ time). If such node exists then we conclude that $f \in \mathcal{F}(S)$ (note that in this case f is the fingerprint corresponding to this node), otherwise $f \notin \mathcal{F}(S)$. The total time complexity of all the stages of solving problem (2) is $O(|f|)$.

To solve problem (3), we attach additionally to each node the top-down trie representation a list of all maximal locations for the fingerprint corresponding to this node. Then after finding in the top-down trie representation the node corresponding to the fingerprint f we can retrieve all maximal locations for f in $O(K)$ time, where K is the number of the maximal locations. Taking into account other various improvements proposed in [3], the results of [3] can be summarized as follows.

- Problem 1 can be solved
  - in $O(n + |\mathcal{L}_C(S)| \log \sigma)$ time and $O(|\mathcal{L}(S)| + |\mathcal{F}(S)| \log \sigma \log n)$ bits of space;
  - in $O(|\mathcal{L}(S)| \log \sigma)$ time and $O(|\mathcal{F}(S)| \log \sigma \log n)$ bits of space;
  - in $O(|\mathcal{L}(S)|)$ expected time and $O(|\mathcal{F}(S)| \log n)$ bits of space with an extremely small probability of error.
- Using $|\mathcal{F}(S)|\big(2 \log \sigma + \log e)(1 + o(1)\big)$ bits of additional memory, problem (2) can be solved
  - in worst-case $O(|f|)$ time and $O(\sigma^{1/\epsilon} \log n)$ bits of working space for any positive integer ϵ;
  - in $O(|f|)$ expected time and $O(|f| \log n)$ bits of working space.

The time complexity bounds for problem (3) are the same as for problem (2) with adding the size K of output.

2. Two-Dimensional Fingerprints

Let M be a two-dimensional array over the alphabet Σ

$$\begin{array}{ccccc} a_{m1} & a_{m2} & \dots & a_{m(n-1)} & a_{mn} \\ a_{(m-1)1} & a_{(m-1)2} & \dots & a_{(m-1)(n-1)} & a_{(m-1)n} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{21} & a_{22} & \dots & a_{2(n-1)} & a_{2n} \\ a_{11} & a_{12} & \dots & a_{1(n-1)} & a_{1n} \end{array},$$

where, without loss of generality, $m \le n$. The fingerprint $f(M)$ of M is the set

$$\{a \in \Sigma \mid \exists\, i, j \colon a_{i,j} = a\} \subseteq \Sigma$$

of all distinct letters in M.

Let $1 \le i' \le i'' \le m$ and $1 \le j' \le j'' \le n$. Then by $\langle i', i''; j', j'' \rangle$ we denote the two-dimensional array

$$\begin{array}{cccc} a_{i''j'} & a_{i''(j'+1)} & \dots & a_{i''j''} \\ \vdots & \vdots & \ddots & \vdots \\ a_{(i'+1)j'} & a_{(i'+1)(j'+1)} & \dots & a_{(i'+1)j''} \\ a_{i'j'} & a_{i'(j'+1)} & \dots & a_{i'j''}, \end{array}$$

which is called a *rectangle* in M. The fingerprint $f_M\langle i', i''; j', j'' \rangle$ is the set $f(\langle i', i''; j', j'' \rangle)$ of all distinct letters in the rectangle $\langle i', i''; j', j'' \rangle$. The rectangle $\langle i', i''; j', j'' \rangle$ is called a *location* of the fingerprint $f_M\langle i', i''; j', j'' \rangle$. By $\mathcal{F}_{\mathrm{R}}(M)$ we denote the set

$$\{f \subseteq \Sigma \mid \exists\, i', i'', j', j'' \colon f = f_M\langle i', i''; j', j'' \rangle\}$$

of all distinct fingerprints for rectangles in M.

A rectangle $\langle i', i''; j', j'' \rangle$ in M is called a *maximal location* of the fingerprint $f = f_M\langle i', i''; j', j'' \rangle$ if this rectangle is not contained in a greater rectangle with the same fingerprint. Examples of maximal locations are shown at Fig. 7.

The set of all maximal locations in M is denoted by $\mathcal{L}_{\mathrm{R}}(M)$. It can be shown that $|\mathcal{F}_{\mathrm{R}}(M)| \le |\mathcal{L}_{\mathrm{R}}(M)| \le nm^2\sigma$.

A rectangle $\langle i', i''; j', j'' \rangle$ is called a *square* if $i'' - i' = j'' - j'$. By $\mathcal{F}_{\mathrm{S}}(M)$ we denote the set

$$\{f \subseteq \Sigma \mid \exists \text{ square } \langle i', i''; j', j'' \rangle \colon f = f_M\langle i', i''; j', j'' \rangle\}$$

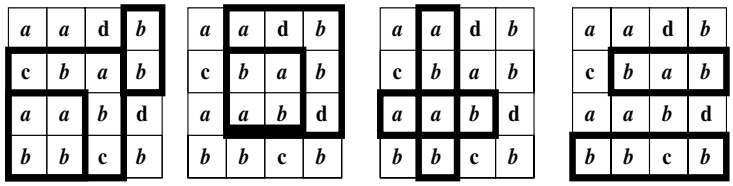

Fig. 7. Examples of maximal locations (outlined by bold contours) in two-dimensional arrays.

of all distinct fingerprints for squares in M. A square in M is called a *square maximal location* if it is not contained in a greater square with the same fingerprint. By $\mathcal{L}_{\mathrm{S}}(M)$ we denote the set of all square maximal locations in M. It can be shown that $|\mathcal{F}_{\mathrm{S}}(M)| \le |\mathcal{L}_{\mathrm{S}}(M)| \le nm\sigma$.

Note that problems (1)–(3) can be naturally reformulated for rectangles and squares in two-dimensional arrays. The considered problems for rectangles and squares were recently studied in [4], where the following results were obtained.

- $\mathcal{F}_{\mathrm{R}}(M)$ can be computed in $O\big(nm^2\sigma \log(|\mathcal{L}_{\mathrm{R}}(M)|/(nm^2)+2)\big)$ time or in $O(nm^2\sigma)$ expected time with polynomially small probability of error;
- $\mathcal{F}_{\mathrm{S}}(M)$ can be computed in $O\big(nm\sigma \log(|\mathcal{L}_{\mathrm{S}}(M)|/(nm)+2)\big)$ time or in $O(nm\sigma)$ expected time with polynomially small probability of error;
- problem (2) (problem (3)) for rectangles can be solved in $O(|f| + \log\log n)$ time ($O(|f| + \log\log n + K)$ time, where K is the size of output) using $O(nm\log n + |\mathcal{F}_{\mathrm{R}}(M)|)$ ($O(nm\log n + |\mathcal{L}_{\mathrm{R}}(M)|)$) additional space;
- problem (2) (problem (3)) for squares can be solved in $O(|f|+\log\log n)$ time ($O(|f|+\log\log n+K)$ time, where K is the size of output) using $O(nm\log n + |\mathcal{F}_{\mathrm{S}}(M)|)$ ($O(nm\log n + |\mathcal{L}_{\mathrm{S}}(M)|)$) additional space.

This work was partially supported by Russian Foundation for Basic Research (grant 14-01-00598).

REFERENCES

1. M. I. Abouelhoda, S. Kurtz, and E. Ohlenbusch, "Replacing suffix trees with enhanced suffix arrays," *J. Discrete Algorithms*, **2**, No. 1, 53–86 (2004).
2. A. Amir, A. Apostolico, G. M. Landau, and G. Satta, "Efficient text fingerprinting via Parikh mapping," *J. Discrete Algorithms*, **1**, No. 5-6, 409–421 (2003).
3. D. Belazzougui, R. Kolpakov, and M. Raffinot, "Various improvements to text fingerprinting," *J. Discrete Algorithms*, **22**, 1–18 (2013).
4. D. Belazzougui, R. Kolpakov, and M. Raffinot, "Indexing and querying color sets of images," *Theor. Comput. Sci.*, **647**, 74–84 (2016).
5. C.-Y. Chan, H.-I. Yu, W.-K. Hon, and B.-F. Wang, "Faster query algorithms for the text fingerprinting problem," *Inf. Comput.*, **209**, No. 7, 1057–1069 (2011).
6. G. Didier, T. Schmidt, J. Stoye, and D. Tsur, "Character sets of strings," *J. Discrete Algorithms*, **5**, No. 2 330–340 (2007).
7. R. Kolpakov and M. Raffinot, "New algorithms for text fingerprinting," in: *Combinatorial Pattern Matching. 17th Annual Symp., CPM 2006, Barcelona, Spain, July 5–7, 2006. Proceedings*, Lect. Notes Comput. Sci., Vol. 4009, Springer, Berlin (2006), pp. 342–353.
8. R. Kolpakov and M. Raffinot, "New algorithms for text fingerprinting," *J. Discrete Algorithms*, **6**, No. 2 243–255 (2008).
9. R. Kolpakov and M. Raffinot, "Faster text fingerprinting," in: *Proc. 15th Int. Symp. on String Processing and Information Retrieval*, Lect. Notes Comput. Sci., Vol. 5280, Springer, Berlin (2009), pp. 15–26.
10. E. M. McCreight, "A space-economical suffix tree construction algorithm," *J. Algorithms*, **23**, No. 2, 262–272 (1976).

11. E. Porat, "An optimal bloom filter replacement based on matrix solving," in: *CSR '09. Proc. Fourth Int. Comput. Sci. Symp. in Russia on Comput. Sci. — Theory and Applications*, Lect. Notes Comput. Sci., Vol. 5675, Springer, Berlin (2009), pp. 263–273.
12. R. Raman, V. Raman, and S. Rao Satti, "Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets," *ACM Trans. Algorithms*, **3**, No. 4, Art. No. 43 (2007).
13. E. Ukkonen, "Constructing suffix trees on-line in linear time," in: *Proc. IFIP 12th World Comput. Congr. on Algorithms, Software, Architecture — Information Processing '92*, Vol. 1, North-Holland, Amsterdam (1992), pp. 484–492.
14. P. Weiner, "Linear pattern matching algorithm," in: *SWAT '73 Proc. 14th Annual Symp. on Switching and Automata Theory*, IEEE Comput. Soc., Washington (1973), pp. 1–11.

Djamal Belazzougui
Department of Computer Science, University of Helsinki, Helsinki, Finland
E-mail: Djamal.Belazzougui@cs.helsinki.fi

Roman Kolpakov
Department of Mechanics and Mathematics, Moscow State University, Moscow, Russia
E-mail: foroman@mail.ru

Mathieu Raffinot
LIAFA, Université Paris Diderot–Paris 7, Paris, France
E-mail: raffinot@liafa.univ-paris-diderot.fr