Tight bounds for searching a sorted array of strings

Arne Andersson[*] Torben Hagerup[†] Johan Håstad[‡] Ola Petersson[§]

August 1, 2000

Abstract

Given a k-character query string and an array of n strings arranged in lexicographical order, computing the rank of the query string among the n strings or deciding whether it occurs in the array requires the inspection of

$$\Theta\left(\frac{k\log\log n}{\log\log\left(4+\frac{k\log\log n}{\log n}\right)}+k+\log n\right)$$

characters in the worst case.

1 Introduction

Given n strings, each of k characters, arranged in lexicographical order (i.e., a string precedes a string from which it differs if it has the smaller character in the first position in which the two strings differ), how many characters must we inspect to determine whether a k-character query string is present? We assume that the n strings are given in a $k \times n$ array and that no extra information is available. If k is a constant, we can solve the problem with $\Theta(\log n)$ character inspections by means of binary search, and this is optimal; but what happens for larger values of k? In the presence of preprocessing and extra storage, there are efficient methods, such as using a trie, each node of which can be implemented as a weighted tree as suggested by Mehlhorn [7, Sect. III.6.3], or the suffix array of Manber and Myers [6]; but what if we are given just the sorted strings?

The question is a basic one; we are simply asking for the complexity of searching a dictionary for a string, where the common assumption that entire strings can be compared in constant time is replaced by the assumption that only single characters can be compared in constant time. For

[*] Computing Science Department, Information Technology, Uppsala University, Box 311, SE–751 05 Uppsala, Sweden. arnea@csd.uu.se. Part of the work was done while at Lund University.

[†] Fachbereich Informatik, Johann Wolfgang Goethe-Universität Frankfurt, D–60054 Frankfurt am Main, Germany. hagerup@informatik.uni-frankfurt.de. Part of the work was done while this author was with the Max-Planck-Institut für Informatik in Saarbrücken, Germany and supported by the ESPRIT Basic Research Actions Program of the EU under contract No. 7141 (project ALCOM II).

[‡] Department of Numerical Analysis and Computing Science, Royal Institute of Technology, SE–100 44 Stockholm, Sweden. johanh@nada.kth.se. Part of the work was done while visiting MIT.

[§] Department of Mathematics, Statistics, and Computer Science, Växjö University, SE–351 95 Växjö, Sweden; and Department of Computer Science, Lund University, Box 118, SE–221 00 Lund, Sweden. sithtry@yahoo.com. Part of the work was done while visiting Columbia University.

sufficiently long strings, the latter assumption seems more realistic. At first glance the problem may appear easy—some kind of generalized binary search should do the trick. However, closer acquaintance with the problem reveals an unexpected intricacy.

Given the relevance of this problem, surprisingly few results have been reported. Hirschberg [4] indicated a lower bound of $\Omega(k + \log n)$ and upper bounds of $O(k \log n)$ and $O(k + n)$ and combined the upper bounds to derive a bound of $O(k \log(2 + n/k))$, all of which is straightforward. A later publication by the same author [3] mentions a first nontrivial upper bound of $O(k \log n / \log k)$. Kosaraju [5] gave an algorithm with a running time of $O(k\sqrt{\log n} + \log n)$. The only nontrivial lower bound deals with constant factors: Kosaraju [5] showed that at least roughly $\log n + \frac{1}{2}\sqrt{k \log n} = O(k + \log n)$ characters need to be inspected. We determine the exact complexity of the problem, up to a constant factor.

Before formulating our result, we describe two relevant computational problems more formally. For all integers $k, n \geq 1$ and all ordered sets Σ, an instance of the *string-ranking problem* of size $k \times n$ over the alphabet Σ is given by a list $s, s_1, \ldots, s_n$ of $n + 1$ strings, each consisting of k characters drawn from Σ, such that $s_1 \preceq \cdots \preceq s_n$, where $\preceq$ denotes the lexicographical order derived from the order on Σ. The task is to compute $|\{j : 1 \leq j \leq n \text{ and } s_j \preceq s\}|$, i.e., the rank of s in the multiset $\{s_1, \ldots, s_n\}$. An instance of the *string-membership problem* of size $k \times n$ over Σ is given by a list of the same form, and the task is to output "yes" if $s = s_j$ for some $j \in \{1, \ldots, n\}$, and "no" otherwise.

The string-membership problem clearly is no harder than the string-ranking problem in the sense that after solving the latter, we can solve the former after inspecting at most k additional characters. An algorithm for the string-ranking problem also allows us to determine the indices of the first and last occurrences, if any, of the query string. As implied by our result, stated below, these problems in fact all have the same deterministic complexity, up to a constant factor. The logarithm function to base 2 is denoted by "log".

Theorem 1.1 *For all integers $k \geq 1$ and $n \geq 4$ and all ordered sets Σ of at least two elements, the solution of instances of size $k \times n$ of the string-ranking problem and of the string-membership problem over the alphabet Σ requires the inspection of*

$$\Theta\left(\frac{k \log\log n}{\log\log\left(4 + \frac{k \log\log n}{\log n}\right)} + k + \log n\right)$$

characters in the worst case.

As a curiosity we note that for the special case $k = \Theta(\log n)$, natural in view of the lower bound of $\Omega(k + \log n)$, we get a tight bound of

$$\Theta\left(\frac{\log n \log\log n}{\log\log\log\log n}\right),$$

which would have been hard to guess in advance.

This paper is based on the two conference presentations [1] and [2]. The proofs given here are significantly shorter and simpler due to the use of potential functions. All four authors contributed equally to both upper-bound and lower-bound parts of the paper.

After introducing notation and terminology in Section 2, we provide intuition in Section 3 and prove the upper bound in Section 4 and the lower bound in Section 5. Sections 3, 4 and 5 can be read independently of each other.

2 Preliminaries

2.1 The leftmost-all-1 problem

To simplify the presentation, we introduce a simplified searching problem called the *leftmost-all-1* problem and demonstrate the upper bound first for this problem. For all integers $k, n \geq 1$, the leftmost-all-1 problem of size $k \times n$ is the special case of the string-ranking problem of size $k \times n$ obtained by fixing the alphabet to be $\{0, 1\}$ and the query string to be $1^{k-1}0$ (i.e., $k-1$ 1's followed by one 0). We assume an instance of the leftmost-all-1 problem of size $k \times n$ to be given in a $k \times n$ matrix I in the following way: For $i = 1, \ldots, k$ and $j = 1, \ldots, n$, $I[i, j]$ is the ith character of the jth string; i.e., each string is written vertically from top to bottom, and the strings are ordered from left to right. The rows and columns of I are numbered from top to bottom and from left to right, respectively, the number of a row or column also being called its *index*. The task is to determine the number of columns in I that contain at least one 0. An alternative formulation, which gives the problem its name, is that the task is to compute one less than the index of the leftmost column in I containing the string 1^k, or n if there is no such column. An algorithm for the leftmost-all-1 problem is said to perform a *probe* when it examines an entry in I, and we charge the algorithm according to how many probes it performs.

2.2 Surface and fence algorithms

In this subsection we introduce terminology convenient for discussing solutions to the leftmost-all-1 problem. The terminology is illustrated in Figs. 1 and 2.

Consider an algorithm for the leftmost-all-1 problem that inspects entries in a $k \times n$ input matrix I one by one. Once the algorithm has established that certain positions in I contain 1's, it may be able to deduce from the sortedness of I and without actual probes that certain other positions in I must also contain 1's. Specifically, let $1 \leq r \leq k$ and $1 \leq c \leq n$ and suppose that the algorithm has already established that $I[i, c] = 1$ for $i = 1, \ldots, r$. Then, clearly, we also have $I[i, j] = 1$ for $i = 1, \ldots, r$ and $j = c+1, \ldots, n$. We say that these additional occurrences of 1 in I are deduced by *1-extension*.

If a column in I is known to contain at least one 0 because a 0 was found in the column itself or in a column to its right, the column and all positions in the column are said to be *rejected*; such a column cannot be the leftmost column containing the string 1^k. The rightmost rejected column is called the *0-barrier*; initially, before any columns have been rejected, we take the 0-barrier to be an imaginary column of index 0. The remaining positions are classified as follows: If a nonrejected position is known to contain a 1, either because it was explicitly probed or by way of 1-extension, it belongs to the *matching area*—the entries in this area are known to match those of the string 1^k. A nonrejected position outside of the matching area is a *surface position* if all positions above it belong to the matching area, and a *buried position* otherwise. Of course, each column contains at most one surface position.

For $r = 1, \ldots, k$, row r is said to be *excluded* if none of the rows $1, \ldots, r$ contains a surface position. The part of an excluded row outside of the rejected columns is known to contain only 1's. If and when when an algorithm for the leftmost-all-1 problem manages to exclude the last row, it can therefore output the number of rejected columns as its result. As long as at least one row is not excluded, we define the *top row* to be the topmost nonexcluded row. Initially,

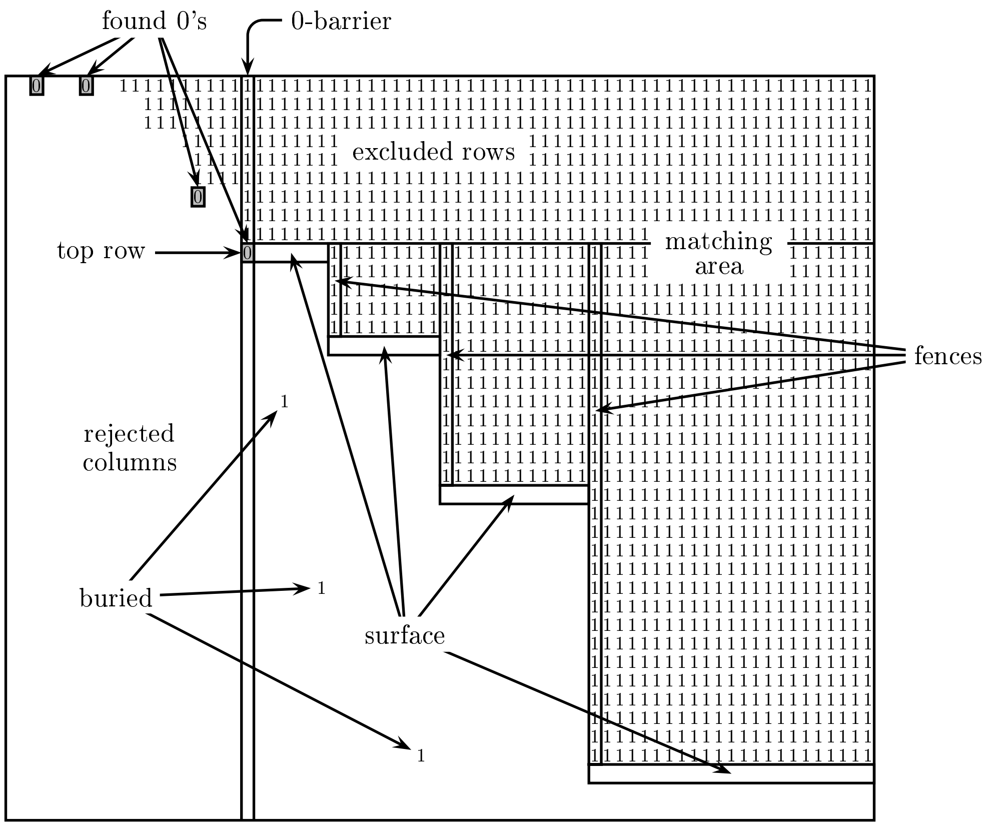

Figure 1: Terminology related to surface algorithms.

no columns are rejected and no rows are excluded, row 1 is the top row, all positions in the top row are surface positions, and all other positions are buried positions.

A *surface probe* is a probe that inspects the entry in a surface position, and a *surface algorithm* is an algorithm all of whose probes are surface probes. We call a surface probe *successful* if it returns a 1 (the string probed still matches the string 1^k) and *unsuccessful* if it returns a 0. The following observations are helpful.

Whenever a surface algorithm finds a 1 in some position, that position and all positions above it and to its right subsequently are part of the matching area. If the position containing the 1 is in the column next to the 0-barrier, the row containing the 1 is excluded and the top row moves down by one position.

Whenever a surface algorithm finds a 0 in a particular column, the 0-barrier moves to that column, and that column and all nonrejected columns to its left are rejected. Since the newly rejected columns may have contained all surface positions of some rows, this may also cause the top row to move downwards. The new top row will be either the row in which the 0 was found or a row below it; the latter happens when the position immediately to the right of the 0 belongs to the matching area.

It can be seen that at all times during the execution of a surface algorithm, a position above or to the right of a position in the matching area also belongs to the matching area—we express this by saying that the matching area is *monotonic*—so that the boundary between the matching area and the remaining positions forms a "staircase" going down and to the right. The part of a column contained in the matching area but outside of the excluded rows is called a *fence* if at least one position immediately to its left is not part of the matching area (i.e., a fence resides in a column where the matching area becomes "deeper"). A *fence algorithm* is a surface algorithm each of whose probes is a *top-row probe*, i.e., a probe in the (current) top row, or an *extending probe*, i.e., a probe of an entry immediately below an existing fence.

The height of a fence F, denoted $|F|$, is defined as the number of positions contained in F. It is obvious that every fence is strictly higher than every fence to its left. When a fence algorithm executes a probe below a fence F, we will say that it attempts to *extend* F. If the extension is successful, the height of F usually increases by 1, i.e., the position probed and the positions belonging to F before the probe form a fence that we identify with F; two exceptions are noted below. If the extension is unsuccessful, F and all fences to the left of F (equivalently, shorter than F) are *excluded*, since they are now completely contained in the excluded rows (and in the rejected columns), while the height of every other fence decreases by the number of rows excluded.

An exceptional case of a successful extension of a fence F or of a successful top-row probe that creates F occurs if before the probe, some fence F' to the right of F had the same height as F after the probe; i.e., a "stair" of the "staircase" vanishes. In this case we will say that F and F' *merge* to create a new fence. We shall frequently need to distinguish between new fences that result from merges and new fences that result from successful top-row probes (without a merge or before a merge triggered by the probe); we shall say that the latter fences are created *from scratch.*

Another exceptional situation happens when a fence algorithm finds a 1 in the column next to the 0-barrier or a 0 in the column immediately to the left of some fence. In either case, no new fence is created, no merge takes place, one or more rows are excluded, and one or more fences may be excluded. We call a probe *excluding* if it causes one or more rows to be excluded.

The *gap* of a fence F is defined as its distance from the 0-barrier, i.e., as $c_F - c_Z$, where c_F and c_Z are the indices of the column containing F and of the 0-barrier, respectively. Observe that the gap of every fence is at least 2. The *index* of a fence is one more than the number of fences strictly to its right; in other words, the fences are numbered from right to left. The proofs of both the upper bound and the lower bound associate with each fence F an integer *weight*, denoted $\|F\|$. We define the *cumulative weight* of a fence as the sum of its own weight and the weights of all fences of strictly smaller index; i.e., if the weights are summed from right to left, the partial sum obtained for each fence is its cumulative weight.

We will denote by $\mathcal{F}$ the (current) list of fences, ordered from left to right. E.g., if $\mathcal{F} = (F_N, \ldots, F_1)$, then F_N and F_1 are the leftmost and rightmost fences, respectively, and $1 \leq |F_N| < \cdots < |F_1|$. For $i = 2, \ldots, N$, we call F_i the *left neighbor* of F_{i-1} and F_{i-1} the *right neighbor* of F_i.

Figure 2 illustrates the various possibilities for a surface probe. Rejected columns and excluded rows are separated from the remaining positions by double vertical and horizontal lines, respectively, the matching area is filled with 1's, and unknown entries that are still of interest are represented by dark squares. Consider the situation in (a), in which we have two fences: F_1 of height 3 in column 10 and F_2 of height 2 in column 7. The six situations in (b)–(g) show possible results of performing one probe, starting from (a). The contents of the probed position are circled. (b) shows a successful top-row probe, creating a new fence of height 1 from scratch in column 5. (c) shows an unsuccessful top-row probe in column 5. (d) shows a successful extension of F_2, leading to a merge of F_1 and F_2. (e) shows an unsuccessful attempt to extend F_2. (f) shows a successful top-row probe in the column next to the 0-barrier; no new fence is created, but one row is excluded. (g) shows an unsuccessful top-row probe in the column next to the leftmost fence F_2; F_2 and the rows that it spans are excluded. Finally, starting from the situation in (b), (h) shows a successful top-row probe performed when the leftmost fence is of height 1; the new fence immediately merges with its right neighbor to form a new leftmost fence.

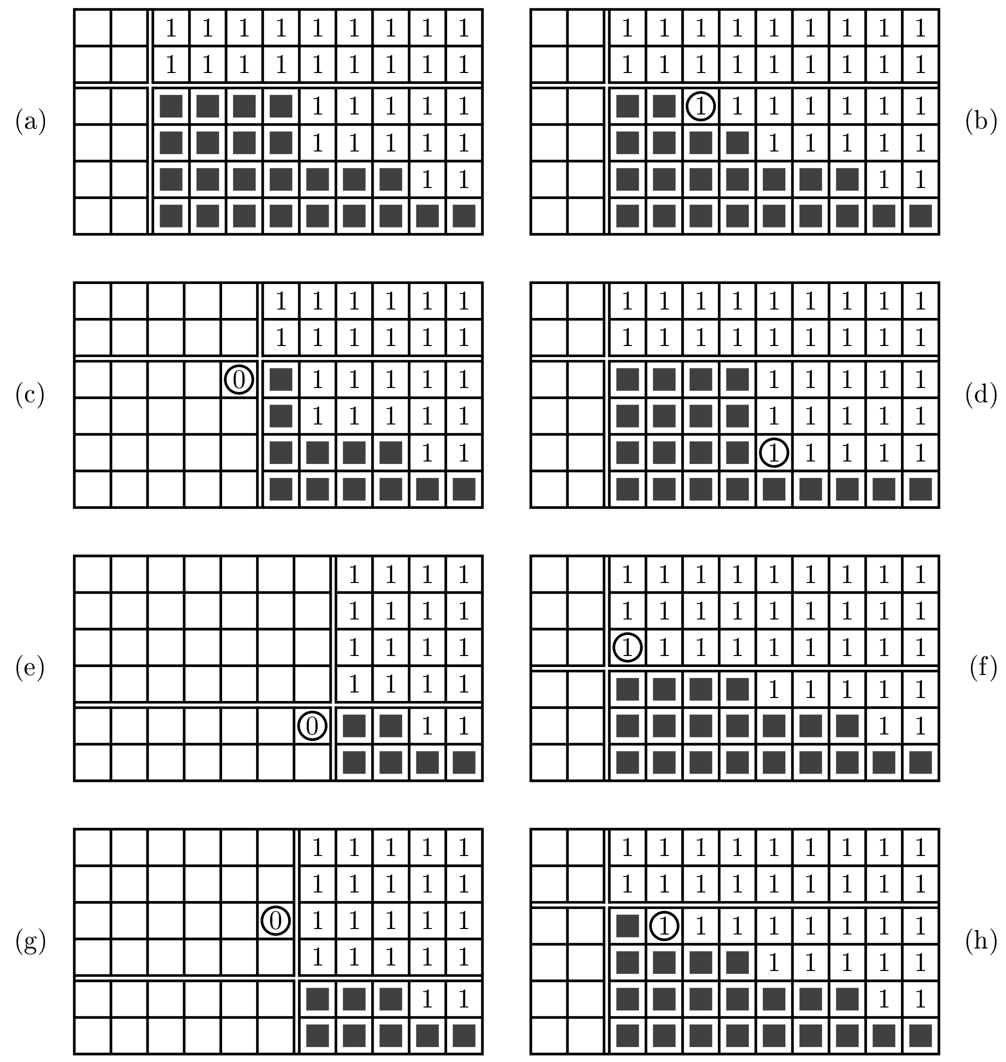

Figure 2: Possible outcomes of a surface probe.

3 Intuitive sketch

Before delving into the formal arguments, let us try to provide intuition behind some important parameters used in the proofs of the upper and lower bounds. The reader should keep in mind that we do not claim to *prove* anything in this section, but merely to show how one may arrive at the parameter values of the proofs through plausible reasoning. Exact values of the parameters and rigorous proofs are given in the remainder of the paper. This section can be skipped without loss of continuity.

First, fence algorithms do come naturally—and one conclusion of our work is that they are optimal, up to a constant factor—so let us concentrate on fence algorithms. Since the goal of a fence algorithm can be viewed as that of excluding all k rows, a natural measure of its progress is the number of rows that it has managed to exclude so far. Excluding a row comes at a certain price, however, since, in some sense, all probes of positions in a row become worthless when the row is excluded. Our main concern therefore will be not to perform too many probes per row, at least in an amortized sense. A symmetric argument could be made concerning rejected columns instead of excluded rows, but this appears to offer less useful insights.

Important properties of a fence algorithm are the number of fences, their spatial distribution, and their heights. Suppose that we aim for a bound of the form kt, for some value of t. That is, we wish to spend an average of t probes per row. We must analyze what happens when we fail to extend a fence. One the one hand, we gain one or more rows, i.e., they are excluded, but on the other hand we lose probes. We identify two kinds of losses:

1. All fences contained in the excluded rows disappear and thus all probes spent to create these short fences are now useless.

2. If N fences remain after the row exclusions, N probes per row used in building these fences are now useless. This suggests that we should keep the number of fences bounded by t.

Let us define some quantities needed to analyze losses of the first kind. When creating a new fence from scratch, it is natural to probe the middle entry of the unknown part of the top row, that is, to make one step of a binary search for locating the rightmost 0 in that row. Suppose that we perform more such probes, which all find 1's. In effect, the new fence moves leftwards and away from its neighboring fence, while still being of height 1. It is intuitively clear that as the new fence F moves to the left, it becomes more and more valuable to the fence algorithm. In order to quantify this, we count the number of binary-search steps used to move F to its current position and call this number the *weight* of F, denoted $\|F\|$. When two fences merge, we define the weight of the resulting fence to be the sum of the weights of the two fences from which it is formed. It can be seen that with this rule, the total weight of all fences can grow to around $\log n$, but not beyond that.

A second important quantity associated with a fence F is the total number of probes spent to construct F. We call this the *investment* in F and denote it by $\mathit{Invest}(F)$. Investments also add up when fences merge. Investments differ from weights in that vertical extensions of fences are counted in the former, but not in the latter. As a consequence, $\mathit{Invest}(F) \geq \|F\|$ for every fence F.

As observed above, we want at most t fences, and a natural way to spread the fences is to make sure that, going from left to right, they are of exponentially increasing weights. If weights increase by a factor of a from each fence to the next, then we should have $a^t = \log n$ or, equivalently,

$$\log a = \frac{\log \log n}{t}, \tag{1}$$

since the maximum weight of any fence is around $\log n$.

Consider a loss of the first kind. As stated above, we choose the weights of the existing fences to be exponentially increasing from left to right, and it turns out that the same will be true for the investments in the fences. Hence a constant fraction of the loss is due to the disappearance of the tallest excluded fence; for simplicity we assume that this is the only loss.

We want the height of a fence to be a function simply of its weight and denote the relevant function by T. Now consider the situation when we fail to extend a fence F. We gain $T(\|F\|)$ rows and lose the investment $\mathit{Invest}(F)$. To keep the cost of t probes per row, this means that we need

$$T(\|F\|) \geq \mathit{Invest}(F)/t \geq \|F\|/t. \tag{2}$$

It is natural to assume that it is optimal for the first inequality to hold with equality. Under this assumption, let us analyze the key operation of merging two fences.

Consider two fences F and F' that merge, with F' to the left of F. The merge is caused by a number of probes that extend F'. Assume that before these probes are made, F and F' are of heights $T(\|F\|)$ and $T(\|F'\|)$, respectively, and that $tT(\|F\|)$ and $tT(\|F'\|)$ probes, respectively, have been invested in them. Furthermore, since the weight of the new fence is the sum of the weights of the old fences, assume that after its creation it is extended to height $T(\|F\| + \|F'\|)$. The extension needs $T(\|F\| + \|F'\|) - T(\|F\|)$ probes, so that afterwards the investment in the new fence will be

$$t(T(\|F\|) + T(\|F'\|)) + T(\|F\| + \|F'\|) - T(\|F'\|).$$

If we disregard the last term (which turns out to be insignificant) and observe that T grows at least linearly, by Equation (2), we see that this expression is at least $(t+1)(T(\|F\|) + T(\|F'\|))$. As the investment in the new fence should be at most $tT(\|F\| + \|F'\|)$, we obtain the relation

$$(t+1)(T(\|F\|) + T(\|F'\|)) \leq tT(\|F\| + \|F'\|).$$

Combining this with the relation

$$\|F\| + \|F'\| = (1 + 1/a)\|F\|,$$

which expresses the intended meaning of the parameter a, yields

$$(t+1)T(\|F\|) \leq tT((1 + 1/a)\|F\|)$$

or

$$(1 + 1/t)T(\|F\|) \leq T((1 + 1/a)\|F\|).$$

Setting $T(x) = dx^p$ for arbitrary constants $d > 0$ and $p \geq 1 + a/t$ is sufficient to satisfy this requirement, and choosing

$$T(x) = \frac{x^{1+a/t}}{t}$$

also takes care of the condition of Equation (2).

Finally, the biggest possible weight of a fence (namely $\log n$) should correspond to the biggest possible height of a fence (namely k), i.e.,

$$T(\log n) = \frac{(\log n)^{1+a/t}}{t} = k.$$

Together with Equation (1), this implies $a \log a = \log(kt/\log n)$. After some simplification, this and Equation (1) yield suitable values for a and t.

Essentially, the algorithm can be derived from this discussion by defining everything precisely and adjusting a few constants. This is done in the next section.

The lower bound is proved by means of an adversary that keeps track of the investments made by an algorithm. Whenever the algorithm has not protected its investment by erecting tall enough fences, the adversary reveals information that makes the algorithm lose part of its investment at a too high cost. The adversary's actions basically force the algorithm to behave as the algorithm in the proof of the upper bound, or it will do worse.

4 The upper bound

In this section we first describe a fence algorithm that solves the leftmost-all-1 problem using a number of probes that is within the upper bound of Theorem 1.1. Later, in Section 4.4, we extend the methods to solve the original string-ranking problem using at most twice as many probes, where a probe, in the case of the string-ranking and string-membership problems, is the inspection of a character in the sorted sequence of input strings.

Consider an input I of size $k \times n$, where $k \geq 1$ and $n \geq 4$ (the condition $n \geq 4$ simply ensures that $\log\log n$ is well-defined and at least 1). We begin by defining a number of parameters in terms of k and n. First,

$$a = \sqrt{\max\left\{\log\left(\frac{k \log\log n}{\log n}\right), 4\right\}} \qquad \text{and} \qquad t = \left\lceil \frac{\log\log n}{\log a} \right\rceil + 2.$$

Second, for all real $x \geq 0$, take

$$T(x) = \frac{x^{1+ea/t}}{t},$$

where $e = 2.718\ldots$ is the base of the natural logarithm function. T maps the set of nonnegative real numbers to itself, and its derivative T' satisfies $T'(x) \geq 1/t$ for all $x \geq 1$. The only other properties of T of relevance to us are expressed in the two lemmas below.

Lemma 4.1 $T(2\log n) = O(k + \log n/\log\log n)$.

Proof.

$$\begin{aligned} T(2\log n) &= \frac{(2\log n)^{1+ea/t}}{t} \leq \frac{(2\log n)^{1+ea\log a/\log\log n} \cdot \log a}{\log\log n} \\ &\leq 2^{1+2ea\log a} \cdot \log a \cdot \frac{\log n}{\log\log n} = O\left(2^{a^2} \cdot \frac{\log n}{\log\log n}\right). \end{aligned}$$

Since $2^{a^2} \cdot \log n/\log\log n = k$ if $a > 2$, the lemma follows. □

Lemma 4.2 *Let x, y and ϵ be nonnegative real numbers with $0 \le \epsilon \le 1$ and $y \ge x \ge \epsilon y$. Then*

$$T(x+y) \ge (1 + \epsilon a/t)(T(x) + T(y)).$$

Proof. By the mean-value theorem, $e^{\epsilon/e} \le 1 + \epsilon$, and therefore

$$x + y \ge (1+\epsilon)y \ge e^{\epsilon/e} \cdot y$$

and

$$\begin{aligned} tT(x+y) &= (x+y)^{1+ea/t} \\ &\ge (e^{\epsilon/e} \cdot y)^{ea/t}(x+y) \\ &= e^{\epsilon a/t} \cdot y^{ea/t}(x+y) \\ &\ge (1+\epsilon a/t) y^{ea/t}(x+y) \\ &\ge (1+\epsilon a/t)\left(x^{1+ea/t} + y^{1+ea/t}\right) \\ &= (1+\epsilon a/t)t(T(x)+T(y)). \end{aligned}$$

□

Using the lemma with $x = y$ and $\epsilon = 0$, we obtain the following.

Corollary 4.3 *For all $x \ge 0$, $T(2x) \ge 2T(x)$.*

4.1 The algorithm

Whenever the fence algorithm to be described performs a top-row probe, it probes in the middle of the surface part of the top row. More precisely, if the leftmost and rightmost surface positions in the top row are in columns c_{L} and c_{R}, respectively, the top-row probe is performed in column $\lfloor (c_{\mathrm{L}} + c_{\mathrm{R}})/2 \rfloor$ or in column $\lceil (c_{\mathrm{L}} + c_{\mathrm{R}})/2 \rceil$ (if $c_{\mathrm{L}} + c_{\mathrm{R}}$ is odd, either choice is acceptable). It will be convenient to allow probes below the actual input, i.e., in "row r" for $r > k$. By convention, such probes always return 1.

As mentioned earlier, the algorithm associates with each fence F a positive integer, called the *weight* of F and denoted by $\|F\|$. The *target height* of F is defined as $H(F) = \lceil T(\|F\|) \rceil$. We say that F is *of target height* if $|F| = H(F)$ and *below target height* if $|F| < H(F)$.

The algorithm repeatedly performs one probe using the procedure Probe specified below until all k rows have been excluded and then outputs the number of rejected columns, which is easily seen to be the correct answer even if fictitious probes below row k were performed. Upon entry to the procedure, the notation is assumed to be fixed so that $\mathcal{F} = (F_N, \ldots, F_1)$ (recall that $\mathcal{F}$ is the list of all fences in the order from left to right).

```
Probe:
  if some fence is below target height
  then perform an extending probe below the rightmost such fence
  else
    if $N \geq 2$ and $\|F_N\| > \|F_{N-1}\|/a$
    then perform an extending probe below $F_N$
    else perform a top-row probe.
```

In the interest of succinctness, the description given above does not specify the manipulation of fence weights. When a new fence is created from scratch, it is given weight 1. When two fences F and F' merge, the resulting fence acquires $\|F\| + \|F'\|$ as its weight. Finally, when a nonexcluding top-row probe encounters a 0 and $N \geq 1$, the weight of the leftmost fence is increased by 1. No other weight changes take place. In particular, only the leftmost fence ever changes its weight.

It may be helpful to visualize how a second fence is created by the algorithm. As long as there is only one fence and this fence is of height 1, the new fence created by a top-row probe that encounters a 1 immediately merges with the old fence. Informally, the net effect can be viewed as the old fence moving to the left and increasing its weight by 1. When the weight of the single fence has increased sufficiently for its target height to reach 2, the fence may be extended beyond height 1, after which a second fence can be created.

4.2 Properties of the algorithm

Lemma 4.4 *Suppose that $\mathcal{F} = (F_N, \ldots, F_1)$ before a probe that extends a fence F_i such that $2 \leq i \leq N$ and $\|F_i\| \leq \|F_{i-1}\|/a$ before the probe. Then the probe will not cause F_i and F_{i-1} to merge.*

Proof. Consider the situation before the probe. F_i is below target height, whereas F_{i-1} is not, since otherwise F_i would not be extended. In particular, $T(\|F_i\|) \geq 1$. Since $a \geq 2$ and $T(x) \geq 2T(x/2)$ for all $x \geq 0$ (Corollary 4.3), $T(\|F_{i-1}\|) \geq 2T(\|F_i\|) \geq T(\|F_i\|)+1$ and therefore $H(F_{i-1}) > H(F_i)$. The lemma follows. □

The following two lemmas are proved together by induction on the number of probes executed.

Lemma 4.5 *At all times, with $\mathcal{F} = (F_N, \ldots, F_1)$, $\|F_i\| \leq \|F_{i-1}\|/a \leq \|F_{i-1}\|/2$ for $2 \leq i < N$.*

Lemma 4.6 *Every merge combines the two leftmost fences.*

Proof. Lemma 4.4 states that if $2 \leq i < N$ and $\|F_i\| \leq \|F_{i-1}\|/a$ before an extension of F_i, then the extension will not cause a merge. Conversely, if the condition of Lemma 4.5 holds before a merge that involves the leftmost fence, it will clearly hold afterwards. Since the condition of Lemma 4.5 is vacuously satisfied initially, while no fence is ever created from scratch unless $N \leq 1$ or the claim holds even for $i = N$, it can be seen that the condition must hold at all times. □

Lemma 4.7 *At all times, if $\mathcal{F} = (F_N, \ldots, F_1)$ and $1 \le i \le j < N$, then $\|F_j\| \le a^{i-j}\|F_i\|$ and $T(\|F_j\|) \le 2^{i-j}T(\|F_i\|)$.*

Proof. The first part of the lemma is obtained by $j - i$ applications of Lemma 4.5. Since $x \le y/2$ implies $T(x) \le T(y)/2$ for all $x, y \ge 0$ (Corollary 4.3), the second part can be proved in a similar way. □

Lemma 4.8 *At all times, if $\mathcal{F} = (F_N, \ldots, F_1)$ and $N \ge 2$, then $\|F_N\| \le \|F_{N-1}\|$.*

Proof. Initially, the claim is vacuously satisfied. We show that if it holds immediately before a probe, then it holds immediately after the probe. Let $\mathcal{F} = (F_N, \ldots, F_1)$ before the probe.

When a fence is created from scratch, it is given weight 1, and the claim is clearly satisfied. An unsuccessful top-row probe may increase $\|F_N\|$ by 1, but is not performed unless $N \le 1$ or $\|F_{N-1}\| \ge a\|F_N\| \ge \|F_N\| + 1$, so that the claim also holds after the probe. Lemma 4.6 states that every merge combines the two leftmost fences. By induction and Lemma 4.5, their combined weight is bounded by the weight of the right neighbor, if any, of the resulting fence. In symbols: $\|F_N\| + \|F_{N-1}\| \le 2\|F_{N-1}\| \le \|F_{N-2}\|$. Finally, and again using Lemma 4.5, the claim is easily seen to hold after the exclusion of one or more fences. □

Lemma 4.9 *At all times, with $\mathcal{F} = (F_N, \ldots, F_1)$, $\sum_{j=i}^{N} T(\|F_j\|) \le 2T(\|F_i\|)$ for $i = 1, \ldots, N$.*

Proof. Assume that $i < N$, since otherwise the claim is trivial. By Lemmas 4.7 and 4.8, $T(\|F_j\|) \le 2^{i-j}T(\|F_i\|)$ for $j = i, \ldots, N-1$ and $T(\|F_N\|) \le T(\|F_{N-1}\|)$. Thus

$$\sum_{j=i}^{N} T(\|F_j\|) \le \left(\sum_{j=i}^{N-1} 2^{i-j} + 2^{i-N+1}\right) \cdot T(\|F_i\|) = 2T(\|F_i\|).$$

□

Lemma 4.10 *At all times, with $\mathcal{F} = (F_N, \ldots, F_1)$, $H(F_i)/2 \le |F_i| \le H(F_i)$ for $i = 1, \ldots, N-1$.*

Proof. Initially, the claim is vacuously satisfied. We show that if it holds immediately before a probe, then it holds immediately after the probe. Let $\mathcal{F} = (F_N, \ldots, F_1)$ before the probe.

Consider first the upper bound of the lemma, i.e., the claim that only the leftmost fence can have a height exceeding its target height. By induction and since the weight of a fence never decreases, when a fence becomes the leftmost fence, because of row exclusions or a merge or because the fence was just created from scratch, its height will not exceed its target height. Moreover, once the algorithm starts extending a fence beyond its target height, the fence must be the leftmost fence, and the algorithm will continue to extend it until an extension is unsuccessful, a merge takes place, or the algorithm terminates. In all cases, the offending fence disappears before it can acquire a left neighbor.

The other inequality of the lemma states that only the leftmost fence can have a height below half its target height. Since no other fence is below target height when a new fence is created from scratch, while only the leftmost fence can increase its weight, this could be invalidated only by row exclusions, which decrease fence heights. Before a top-row probe is performed, no fence is below target height, so that even if the probe excludes one row, the height of every surviving fence after the probe will still be at least half its target height. Consider therefore a probe that excludes fences $F_N, \ldots, F_i$ and let j be arbitrary with $1 \le j < i-1$. Before the probe, F_{i-1} and F_j are both of target height, by the upper bound established above, so that at this time, by Corollary 4.3 and Lemma 4.5,

$$|F_j| = H(F_j) \ge T(\|F_j\|) \ge T(2\|F_{i-1}\|) \ge 2T(\|F_{i-1}\|) \ge 2(H(F_{i-1}) - 1) = 2(|F_{i-1}| - 1).$$

Hence, before the probe, $|F_i| \le |F_{i-1}| - 1 \le |F_j|/2$, so that after the probe we still have $|F_j| \ge H(F_j)/2$. This holds for all j with $1 \le j < i-1$; i.e., the lower bound of the lemma is satisfied. □

The following consequence of the previous lemma and its proof will be useful later.

Corollary 4.11 *Immediately before the algorithm performs a probe, every fence strictly to the right of the position probed is of target height.*

Lemma 4.12 *At all times, if $\mathcal{F} = (F_N, \ldots, F_1)$, then for $i = 1, \ldots, N$, the gap of F_i is at most $2^{\lceil \log n \rceil + 1 - w_i}$, where $w_i = \sum_{j=1}^{i} \|F_j\|$ is the cumulative weight of F_i.*

Proof. By induction on the number of probes performed. Since the gap of a fence never increases, the proof amounts to a simple inspection of the cases in which a fence is created or its cumulative weight increases.

Consider first the case in which a fence F is created from scratch. If F is created without a right neighbor (i.e., it is the only fence), it has a cumulative weight of 1, and its gap is bounded by n, so the claim is satisfied. Immediately after the creation from scratch of a fence F with a right neighbor F', the cumulative weight w of F is one more than the cumulative weight w' of F', and by the policy of placing new fences created from scratch in the middle of the surface part of the top row, the gap of F is at most $1/2$ plus half the gap of F'. By induction, therefore, the gap of F is at most $\lfloor 2^{\lceil \log n \rceil + 1 - w'}/2 + 1/2 \rfloor = 2^{\lceil \log n \rceil + 1 - w}$ (recall that the gap of F' is at least 2), and the claim continues to hold.

When two fences merge, the cumulative weight of every fence after the merge is the same as the cumulative weight of the fence residing in the same column before the merge. Finally, an unsuccessful top-row probe that increases the weight of the leftmost fence by 1 also reduces its gap to at most $1/2$ plus half the old gap, and the claim continues to hold as above. □

Corollary 4.13 *No weight of a fence ever exceeds $\lceil \log n \rceil + 1 < 2 \log n$.*

Lemma 4.14 *The number of fences never exceeds t.*

Proof. If at some point more than t fences exist, we can apply Lemma 4.7 with $i = 1$ and $j = t$ and derive the following contradiction from the previous corollary:

$$1 \le \|F_t\| \le a^{1-t} \cdot \|F_1\| < a^{-\log \log n / \log a - 1} \cdot 2 \log n = 2/a.$$

□

4.3 Analysis of the number of probes

In this subsection we complete the analysis of the algorithm by showing the number of probes performed to be $O(kt + \log n)$. The approach is simple: We define a potential function Φ of the state of the algorithm, initially zero, show that every probe increases Φ by at least one, and bound the maximum value of Φ.

Consider a point in time during the execution of the algorithm at which $\mathcal{F} = (F_N, \ldots, F_1)$ and let X_{C} and X_{R} be the numbers of rejected columns and of excluded rows, respectively. The potential function Φ has the form $\Phi = \Phi_1 + \Phi_2 + \Phi_3 + \Phi_4$, where $\Phi_1 = \log(n/(n - X_{\mathrm{C}}))$ measures the ratio of original columns to remaining columns, $\Phi_2 = 13tX_{\mathrm{R}}$ is proportional to the number of excluded rows, $\Phi_3 = \sum_{i=1}^{N} |F_i|$ is the total height of all fences, and

$$\Phi_4 = 2t \cdot \left(\min\{T(\|F_N\|), 3|F_N|\} + \sum_{i=1}^{N-1} T(\|F_i\|) \right),$$

taken to be zero if no fences exist. Φ_3 and Φ_4 decompose naturally into contributions by the individual fences. The contribution of a fence F_i to Φ_3 is its height $|F_i|$, and the contribution of F_i to Φ_4 is essentially $2t$ times $T(\|F_i\|)$. An exception concerns the leftmost fence F_N. If the height of F_N is less than one third of $T(\|F_N\|)$, the contribution of F_N to Φ_4 instead is $6t$ times its height. If and when column n is rejected, Φ_1 becomes undefined; since this also causes the algorithm to terminate without additional probes, it is of no concern to the proof.

Lemma 4.15 *Every nonexcluding successful top-row probe increases Φ by at least* 1.

Proof. The probe leaves Φ_1 and Φ_2 unchanged. If it does not cause a merge, it increases Φ_3 by 1 and does not decrease Φ_4. If the probe causes a merge, it leaves Φ_3 unchanged and increases Φ_4 by replacing the leftmost fence by a fence of the same height whose weight is greater by 1. Since the leftmost fence was of target height before the probe (Corollary 4.11) and $T'(x) \geq 1/t$ for all $x \geq 1$, the increase in Φ_4 is at least 2. In either case, the net increase in Φ is at least 1. □

Lemma 4.16 *Every successful extending probe increases Φ by at least* 1.

Proof. Even if the probe causes a merge, consider an imagined intermediate situation in which the fence in question has been extended, but no merge has yet taken place. Until this point, the probe leaves Φ_1 and Φ_2 unchanged, increases Φ_3 by 1, and does not decrease Φ_4. Overall, Φ increases by at least 1, and we are done if no merge happens.

In the rest of the proof we assume that a merge happens and prove that it does not decrease Φ. Φ_1 and Φ_2 are not affected by the merge. Let $\mathcal{F} = (F_N, \ldots, F_1)$ immediately before the merge. By Lemma 4.6, the fences that merge are the leftmost ones, i.e., F_N and F_{N-1}. By Corollary 4.11 and Lemma 4.8, the fences that merge as well as the resulting fence are all of height $|F_N| = H(F_{N-1}) \geq T(\|F_{N-1}\|) \geq T(\|F_N\|)$.

The contribution to $\Phi_3 + \Phi_4$ of F_N and F_{N-1} before the merge is at most

$$2|F_N| + 2t(T(\|F_N\|) + T(\|F_{N-1}\|)) \leq 6t|F_N|,$$

the contribution to $\Phi_3 + \Phi_4$ of the fence resulting from the merge is

$$|F_N| + 2t \cdot \min\{T(\|F_N\| + \|F_{N-1}\|), 3|F_N|\},$$

and no other fence changes its contribution.

If $3|F_N| < T(\|F_N\| + \|F_{N-1}\|)$, the merge clearly increases $\Phi_3 + \Phi_4$. Otherwise the increase in Φ caused by the merge is at least

$$2t[T(\|F_N\| + \|F_{N-1}\|) - (T(\|F_N\|) + T(\|F_{N-1}\|))] - |F_N|.$$

By Lemmas 4.4 and 4.8, we must have $\|F_{N-1}\| \geq \|F_N\| > \|F_{N-1}\|/a$. We can therefore apply Lemma 4.2 with $\epsilon = 1/a$, which shows that Φ increases by at least

$$\frac{2t}{t}[T(\|F_N\|) + T(\|F_{N-1}\|)] - |F_N|.$$

To see that this is nonnegative, observe that $T(\|F_{N-1}\|) > 1$ (since $|F_{N-1}| \geq 2$) and therefore $|F_N| = \lceil T(\|F_{N-1}\|) \rceil \leq 2T(\|F_{N-1}\|)$. □

Lemma 4.17 *Every nonexcluding unsuccessful probe increases* Φ *by at least* 1.

Proof. The probe is a top-row probe and leaves $\Phi_2 + \Phi_3$ unchanged. If no fences are present when the probe is performed, at least half of the remaining columns are rejected, causing Φ_1 to increase by at least 1 while Φ_4 remains zero. If one or more fences are present at the time of the probe, the weight of the leftmost fence increases by 1. Since the fence was of target height before the probe, by Corollary 4.11, an argument as in the proof of Lemma 4.15 shows that this increases Φ_4 by at least 2, while Φ_1 does not decrease. □

Lemma 4.18 *Every excluding probe causes all nonrejected columns to be rejected or increases* Φ *by at least* 1.

Proof. Let $\mathcal{F} = (F_N, \ldots, F_1)$ immediately before the probe under consideration. Suppose that the probe leads to the exclusion of m rows, but not to the rejection of all nonrejected columns. Then the probe does not decrease Φ_1, increases Φ_2 by $13tm$ and, since there are at most t fences (Lemma 4.14), decreases Φ_3 by at most tm.

If no fences are excluded and $N \geq 1$, $|F_N|$ decreases by m and $\|F_i\|$ is unchanged for $i = 1, \ldots, N$, so that Φ_4 decreases by at most $6tm$.

If at least one fence is excluded, let the excluded fences be $F_N, \ldots, F_i$. By Lemmas 4.9 and 4.10, the contribution to Φ_4 of the excluded fences $F_N, \ldots, F_i$ before the probe was at most $4tT(\|F_i\|) \leq 8t|F_i|$ if $i < N$ and at most $6t|F_i|$ if $i = N$, so that their exclusion decreases Φ_4 by at most $8t|F_i| \leq 8tm$. If $i > 1$, the contribution of F_{i-1} to Φ_4 may also decrease because F_{i-1} becomes the new leftmost fence. Since F_{i-1} was of target height before the probe under consideration, however (Corollary 4.11), this happens only if F_{i-1} loses more than two thirds of its height, in which case its contribution to Φ_4 before the probe was $2tT(\|F_{i-1}\|) \leq 2t|F_{i-1}| \leq 3tm$. Altogether, therefore, Φ_4 decreases by at most $8tm + 3tm = 11tm$.

In either case Φ_4 decreases by at most $11tm$, yielding a net increase in Φ of at least $13tm - tm - 11tm = tm \geq 1$. □

Lemma 4.19 *As long as at least one column has not been rejected,* $\Phi = O(kt + \log n)$.

Proof. As long as at least one column has not been rejected, $\Phi_1 \le \log n$, and $\Phi_2 \le 13kt$. By Corollary 4.13, the weight of every fence remains bounded by $2\log n$. Coupled with the facts that $\sum_{i=1}^{N} T(\|F_i\|) \le 2T(\|F_1\|)$ if $N \ge 1$ (Lemma 4.9), that $T(2\log n) = O(k + \log n/\log\log n)$ (Lemma 4.1), and that $t = O(\log\log n)$, this shows that $\Phi_4 = O(kt + \log n)$. In order to complete the proof by demonstrating that $\Phi_3 = O(kt + \log n)$, it suffices, since there are at most t fences (Lemma 4.14), to show that no fence is ever of height more than $\lceil T(2\log n)\rceil$. Since only the leftmost fence can have a height exceeding its target height (Lemma 4.10), this follows immediately from Corollary 4.13 for all other fences. As for the leftmost fence F, its height is bounded by 1 or by the height of a nonleftmost fence (that may disappear at the creation of F) when F is created and whenever it is not the only fence. We finally observe that the algorithm never extends a fence whose height is no smaller than its target height if it is the only fence and conclude that even the leftmost fence can never acquire a height of more than $\lceil T(2\log n)\rceil$. $\square$

Lemma 4.20 *The total number of probes performed by the algorithm is* $O(kt + \log n)$.

Proof. Since $\Phi = 0$ initially, the claim follows immediately from Lemmas 4.15–4.19. $\square$

We can conclude

Theorem 4.21 *For all integers $k \ge 1$ and $n \ge 4$, leftmost-all-1 problems of size $k \times n$ can be solved by a fence algorithm using*

$$O\left(\frac{k\log\log n}{\log\log\left(4 + \frac{k\log\log n}{\log n}\right)} + k + \log n\right)$$

probes in the worst case.

4.4 String ranking

In this subsection we extend the algorithm for the leftmost-all-1 problem to solve the original string-ranking problem using at most twice as many probes. The upper bound of our main result, Theorem 1.1, follows from Theorem 4.21 and Lemma 4.22 below.

Lemma 4.22 *For all integers $k, n, m \ge 1$, if there is a surface algorithm that solves instances of size $k \times n$ of the leftmost-all-1 problem using at most m probes, then there is an algorithm that solves instances of size $k \times n$ of the string-ranking problem using at most $2m$ probes.*

Proof. Let I and s be the input matrix and the query string, respectively, and denote by s_i the ith character of s, for $i = 1, \dots, k$. We derive a $k \times n$ matrix I' from I as follows: First, for each column of I that coincides with s, the corresponding column of I' contains 1's in every position. Now assume that $1 \le c \le n$ and that column c of I differs from s in at least one position and let r be the smallest row index with $I[r,c] \ne s_r$. Then we set $I'[r,c] = 0$ if $I[r,c] < s_r$ and $I'[r,c] = 2$ if $I[r,c] > s_r$, and all other entries in column c of I' are set to 1. It can be seen that

I' is sorted, and that the task is to compute the number of columns in I' that do not contain a 2.

With the understanding that each occurrence of 2 is to be considered equivalent to a 1, a convention that again preserves sortedness, we can run a process $\mathcal{A}_{\mathrm{L}}$ that executes the given surface algorithm for the leftmost-all-1 problem on the input I'. This computes the number of columns containing a 0, which is not what we want. On the other hand, we can instead run a "mirrored" process $\mathcal{A}_{\mathrm{R}}$ that also executes the given surface algorithm, but interchanging the roles of left and right, $<$ and $>$, and 0 and 2 (in particular, $\mathcal{A}_{\mathrm{R}}$ considers 0 to be equivalent to 1). This process will indeed compute the number of columns that do not contain a 2 or, rather, n minus that number.

There is a catch, however, namely that it is not clear how to produce the input I' before starting $\mathcal{A}_{\mathrm{R}}$ without using too many probes. Instead, $\mathcal{A}_{\mathrm{R}}$ must be able to convert the outcome of each of its probes in I to the corresponding entry in I' without performing additional probes. This takes a little care and requires us to execute both $\mathcal{A}_{\mathrm{L}}$ and $\mathcal{A}_{\mathrm{R}}$ in an interleaved fashion.

Let us consider the situation from the perspective of $\mathcal{A}_{\mathrm{L}}$. Since $\mathcal{A}_{\mathrm{L}}$ is a surface algorithm, initially it has no difficulty converting the entries read in I to the corresponding entries in I', the reason being that in each column, it probes from the top towards the bottom. When about to exclude one or more rows, however, $\mathcal{A}_{\mathrm{L}}$ runs into a problem. Outside of the rejected columns, the rows of I' to be excluded are known to contain only characters that $\mathcal{A}_{\mathrm{L}}$ considers to be equivalent to 1, namely 1's and 2's, and $\mathcal{A}_{\mathrm{L}}$ will assume that all such characters are in fact 1's. However, any occurrence of a 2 changes the interpretation of a 0 that may later be discovered further down in the same column, and therefore $\mathcal{A}_{\mathrm{L}}$ may later convert entries of I incorrectly to ones of I' if it excludes a row containing 2's. Whenever $\mathcal{A}_{\mathrm{L}}$ is about to exclude a row, it therefore needs help. In complete symmetry, $\mathcal{A}_{\mathrm{R}}$ can run until it is about to exclude one or more rows, at which point, since the rows of I' to be excluded might contain 0's, $\mathcal{A}_{\mathrm{R}}$ needs help.

Now consider a situation in which both $\mathcal{A}_{\mathrm{L}}$ and $\mathcal{A}_{\mathrm{R}}$ are blocked and waiting for help and let r_{L} and r_{R} be the indices of the topmost rows about to be excluded by $\mathcal{A}_{\mathrm{L}}$ and $\mathcal{A}_{\mathrm{R}}$, respectively. Also let z_{L} and z_{R} be the indices of the 0-barriers of $\mathcal{A}_{\mathrm{L}}$ and $\mathcal{A}_{\mathrm{R}}$, respectively. Assume inductively that up to the present point, none of the two processes has ever excluded a row of I' with a position containing an entry different from 1 and rejected by neither $\mathcal{A}_{\mathrm{L}}$ nor $\mathcal{A}_{\mathrm{R}}$. Also assume that $\mathcal{A}_{\mathrm{L}}$ and $\mathcal{A}_{\mathrm{R}}$ are modified so that whenever one of the processes wants to query a position that has been rejected by the other process, it receives a 1 as the answer to its query without consulting I. By the inductive hypothesis, the column of I' of index z_{L} is known to contain a string smaller than 1^k and therefore must be to the left of the column of I' of index z_{R}, which is known to contain a string larger than 1^k. In other words, $z_{\mathrm{L}} < z_{\mathrm{R}}$. Moreover, for all c with $z_{\mathrm{L}} < c < z_{\mathrm{R}}$, every position of the form $I[\min\{r_{\mathrm{L}}, r_{\mathrm{R}}\}, c]$ is known to contain an entry that is both ≥ 1 and ≤ 1 and thus equals 1. It follows that if $r_{\mathrm{L}} \leq r_{\mathrm{R}}$, then $\mathcal{A}_{\mathrm{L}}$ can proceed and exclude row r_{L} without falsifying the inductive property, while if $r_{\mathrm{R}} \leq r_{\mathrm{L}}$, $\mathcal{A}_{\mathrm{R}}$ can resume operation.

Thus $\mathcal{A}_{\mathrm{L}}$ and $\mathcal{A}_{\mathrm{R}}$ are never simultaneously blocked. Moreover, once one of the processes terminates, the other process can finish without being suspended again. Since the two processes $\mathcal{A}_{\mathrm{L}}$ and $\mathcal{A}_{\mathrm{R}}$ are copies of the given surface algorithm, except that sometimes they wait and that some answers are given to them for free, the total number of probes performed is bounded by $2m$. □

Although the upper bounds of Theorems 1.1 and 4.21 specify only the number of probes performed, we note that the algorithms realizing the upper bounds can be executed in a total time that is within the bound on the number of probes, each probe being followed by exactly one (three-way) comparison between two characters. The only nontrivial observation needed is that during an execution of the algorithm described in Section 4.1, whenever a fence that is not the leftmost fence is of target height, it remains of target height until the next row exclusion, at which point we can afford to step through a list of all fences.

5 The lower bound

The aim of this section is to prove the following theorem, which implies the lower-bound part of Theorem 1.1.

Theorem 5.1 *For all integers $k \geq 1$ and $n \geq 4$, every deterministic algorithm for the string-membership problem or the leftmost-all-1 problem performs*

$$\Omega\left(\frac{k \log\log n}{\log\log\left(4 + \frac{k \log\log n}{\log n}\right)} + k + \log n\right)$$

probes on some input of size $k \times n$.

We prove Theorem 5.1 by exhibiting an adversary that forces every deterministic algorithm $\mathcal{A}$ for either problem to spend as many probes as stated in the theorem before announcing its answer. In the case of the string-membership problem, the lower bound is proved for a special case: The alphabet Σ and the string s whose presence is to be tested are fixed to be $\{0,1\}$ and $1^{k-1}0$, respectively.

The adversary fixes entries of a legal input I online in response to the queries made by $\mathcal{A}$. Whenever $\mathcal{A}$ poses a query (r,c), i.e., asks for the value of $I[r,c]$, the adversary executes a call $\mathsf{Process}(r,c)$, where $\mathsf{Process}$ is described in the next subsection, that fixes zero or more entries of I. Subsequently $I[r,c]$ will have been fixed to either 0 or 1, and the value to which it was fixed is returned to $\mathcal{A}$ as the answer to its query.

We formally define a *position* to be an element of $\{1,\ldots,k\} \times \{1,\ldots,n\}$. The adversary maintains information about the part of I already fixed in a $k \times n$ array J with entries drawn from $\{0, 1, \text{‘tentative-1’}, \text{‘?’}\}$ and a set P of *pending positions*. For $r = 1,\ldots,k$ and $c = 1,\ldots,n$, by definition, if $(r,c) \in P$, then $I[r,c]$ has been fixed to 1; if $(r,c) \notin P$, then $I[r,c]$ has been fixed to the value $b \in \{0,1\}$ exactly if $J[r,c] = b$.

Although the adversary is not a fence algorithm probing the input I, it will be very convenient for the proof to reuse the terminology introduced for fence algorithms in Section 2.2. In order to make this possible, it suffices to define the rejected positions and the matching area, since all other relevant concepts (fences, surface positions, the 0-barrier, etc.) were derived from these basic notions. But this is easy: A position (r,c) is rejected exactly if $J[i,j] = 0$ for some position (i,j) with $j \geq c$, and a nonrejected position (r,c) belongs to the matching area exactly if $J[r,c] \in \{1, \text{‘tentative-1’}\}$. The adversary will carry out explicit 1-extension (or, rather, “tentative-1-extension”) to ensure that the matching area remains monotonic. Thus entries of 0 in J correspond to probes answered 0, and entries of 1 or ‘tentative-1’ correspond to probes answered 1. The difference between 1 and 'tentative-1' is that a 1 is permanent, as is a 0, while a 'tentative-1' may later be changed to 0 or 1. If $J[r,c] =$ '?' for some position (r,c), the adversary has not yet decided upon the value of $I[r,c]$ (unless $(r,c) \in P$, in which case $I[r,c] = 1$).

To a first approximation, the adversary fixes only those entries of I that were queried by $\mathcal{A}$ or whose values can be deduced from such entries by 1-extension. In order to simplify the bookkeeping, however, we let the adversary sometimes fix additional entries of I. Informally, this allows us to assume that $\mathcal{A}$ operates largely as the algorithm analyzed in the previous section. The lower bound holds even if the additional information about I volunteered by the adversary in this manner is made known to $\mathcal{A}$. It is therefore not necessary to distinguish between the information available to $\mathcal{A}$ and that available to the adversary—this is obvious anyway, since the adversary operates according to a fixed, deterministic strategy.

5.1 The adversary's strategy

The adversary's strategy is formulated in terms of a number of parameters that we introduce next. It will be convenient to use the natural logarithm function "ln" to base e instead of the logarithm function to base 2 employed in the previous section. First, let

$$a = \ln\left(\frac{k \ln\ln n}{\ln n}\right) \qquad \text{and} \qquad v = 3a + 1.$$

For $k = O(\log n/\log\log n)$, the bound of Theorem 5.1 reduces to a trivial bound of $\Omega(k + \log n)$. This allows us to assume a to be larger than any convenient constant. In particular, we will assume that $a \geq 4$ and hence

$$v \leq a^2. \tag{3}$$

Next, we take

$$t = \left\lfloor \frac{\ln\ln n}{8 \ln a} \right\rfloor.$$

Similarly as before, the bound of Theorem 5.1 reduces to the trivial bound of $\Omega(k + \log n)$ for $t = O(1)$ and hence for $a = (\log n)^{\Omega(1)}$, for which reason we will assume that $t \geq 2$ and that the following relations hold:

$$e^{1/(3t)} \leq 1 + 1/(2t) \tag{4}$$

$$4at \leq \sqrt{\ln n} \tag{5}$$

$$kt + 1 \leq n^{1/4}. \tag{6}$$

The parameters a and t have essentially the same meaning as in the proof of the upper bound. In particular, the goal of the adversary is to force $\mathcal{A}$ to spend $\Omega(t)$ probes per row. We associate with each fence F (as implied by J) a weight, $\|F\|$, which is maintained, with one exception, as in the proof of the upper bound. Every new fence created from scratch has weight 1, and when two fences F and F' merge to form a new fence, the new fence is given weight $\|F\| + \|F'\|$. The difference to the proof of the upper bound is that these two rules are the only ones that govern the weights of fences. In particular, the rejection of a number of columns does not change the weight of any surviving fence.

For each integer i and all $x \geq 0$, take

$$\phi_i(x) = \frac{x}{t}\left(\frac{x}{v^{t-i}}\right)^{a/t} \ln\left(\frac{x}{v^{t-i}} + e\right) \geq 0$$

and note for later use that the derivative

$$\phi_i'(x) = \frac{1}{t}\left(\frac{x}{v^{t-i}}\right)^{a/t}\left[\left(1+\frac{a}{t}\right)\ln\left(\frac{x}{v^{t-i}}+e\right) + \frac{1}{1+e\cdot v^{t-i}/x}\right]$$

of ϕ_i is a strictly increasing function. Recall that the *index* of a fence F is one more than the number of fences to the right of F. We define the *value* (to $\mathcal{A}$) of a fence F of index i as

$$\Phi(F) = |F| + \|F\| + \phi_i(\|F\|).$$

The fence will be called *dense* if $\Phi(F) \geq t|F|$. A fence is *critical* if it is dense or its index is t. The following technical lemma is needed later.

Lemma 5.2 *If a fence F of index i is not dense, then*

$$|F| \geq \frac{\|F\|}{at} \cdot z^{a/t},$$

where $z = \|F\|/v^{t-i}$.

Proof. Since F is not dense,

$$|F| \geq \frac{\Phi(F)}{t} \geq \frac{\|F\|}{t} + \frac{\|F\|}{t^2} \cdot z^{a/t} \ln(z+e).$$

If $z \leq a^{t/a}$, then

$$|F| \geq \frac{\|F\|}{t} \geq \frac{\|F\|}{t} \cdot \frac{z^{a/t}}{a},$$

as desired. If $z > a^{t/a} \geq e^{t/a}$, on the other hand, then

$$|F| \geq \frac{\|F\|}{t^2} \cdot z^{a/t}(t/a) = \frac{\|F\|}{at} \cdot z^{a/t}.$$

□

The adversary exercises tight control over the horizontal placement of fences. As an aid in describing this mechanism, we introduce a set L of "legal fence-column indices" and a corresponding set of "legal fence columns". Suppose that $\mathcal{F} = (F_N, \ldots, F_1)$, that F_i is in column c_i, for $i = 1, \ldots, N$, and that the 0-barrier is in column c_Z. Then $L = \{c^*, c_N, c_{N-1}, \ldots, c_1\}$, where $c^* = c_Z + \max\{\lfloor e^{-a} \cdot (c_N - c_Z)\rfloor, 1\}$, with c_N taken to be $n+1$ if $N = 0$. A legal fence column is a column whose index belongs to L. Thus every column that already contains a fence is a legal fence column, and there is one more legal fence column to the left of the leftmost fence, about e^{-a} of the way from the 0-barrier to that fence. The adversary rejects the column of every probe to the left of the leftmost legal fence column and translates every other probe to the nearest legal fence column to the left of or in the column of the position probed and in this way allows fences to grow only in legal fence columns. Note that since $e^{-a} \le 1/2$, this implies that no two fences will ever reside in adjacent columns.

We now describe the strategy of the adversary precisely by giving the procedure Process and two subroutines PutZero and OneProbe that it employs. Before the first call of Process, every entry of J is initialized to '?', and the set P of pending positions is set to $\emptyset$. For the sake of a succinct description of PutZero, we take $\min\emptyset$ to be ∞, i.e., distinct from every integer.

```
PutZero($c$):
  for $(i,j) \in \{1,\ldots,k\} \times \{1,\ldots,c\}$ do
    $J[i,j] := \begin{cases} 0, & \text{if } i = \min\{\ell \mid 1 \le \ell \le k \text{ and } J[\ell,j] \ne 1\} \\ 1, & \text{otherwise.} \end{cases}$
OneProbe($r,c$):
  if $c < \min L$ then PutZero($c$) else
    $c' := \max\{j \in L \mid j \le c\}$;
    for $j \in \{c',\ldots,n\}$ do if $J[r,j] =$ '?' then $J[r,j] :=$ 'tentative-1';
    if column $c'$ contains a critical fence then PutZero($c'$).
Process($r,c$):
  if $(r,c)$ is buried then insert $(r,c)$ in $P$ and return 1 else
    if $J[r,c] =$ '?' then
      OneProbe($r,c$);
      while $P$ contains a surface position do
        Let $(r',c')$ be a surface position in $P$;
        OneProbe($r',c'$);
        Remove from $P$ all positions $(i,j) \in P$ with $J[i,j] \ne$ '?';
      for $(i,j) \in \{1,\ldots,k\} \times \{1,\ldots,n\}$ do if $J[i,j] =$ 'tentative-1' then $J[i,j] := 1$;
    Return $J[r,c]$.
```

We illustrate the adversary's strategy through an extensive example worked in Fig. 3. The symbols '0', '1' and 'T' denote positions in J containing the values 0, 1 and 'tentative-1', respectively. Each 'P' denotes a position $(i,j) \in P$ with $J[i,j] =$ '?', while other occurrences of '?' in J are not shown explicitly. The matching area is separated from the remaining positions by a staircase line, and surface positions are shown shaded.

Assume that Process$(6,7)$ is called when the situation is as shown in (a). Since $(6,7)$ is a surface position, a call OneProbe$(6,7)$ is executed. The argument $(6,7)$ of OneProbe is indicated by a circle in (a), and the effect of the call OneProbe$(6,7)$ is shown in (b). The position $(6,7)$ is "moved" left until it hits a legal fence column, the fence F in that column is extended by one position containing the value 'tentative-1', and "tentative-1-extension" is carried out from the new fence position. The latter causes several formerly buried positions in P to become part of the surface, and the call of Process proceeds to execute OneProbe(r',c') for one such position (r',c'). This second extension of F, the result of which is shown in (c), causes F to merge with its right neighbor. The transition from (c) to (d) gives rise to yet another merge. We assume that the fence in column 6 resulting from the merge is dense, so that PutZero(6) is executed. The outcome is shown in (e): Four columns were rejected, and six rows were excluded. Each excluded column contains a 0 in the position that belonged to the surface in situation (a) and 1's in all other positions. In situation (e), the surface still contains elements of P, so another call of

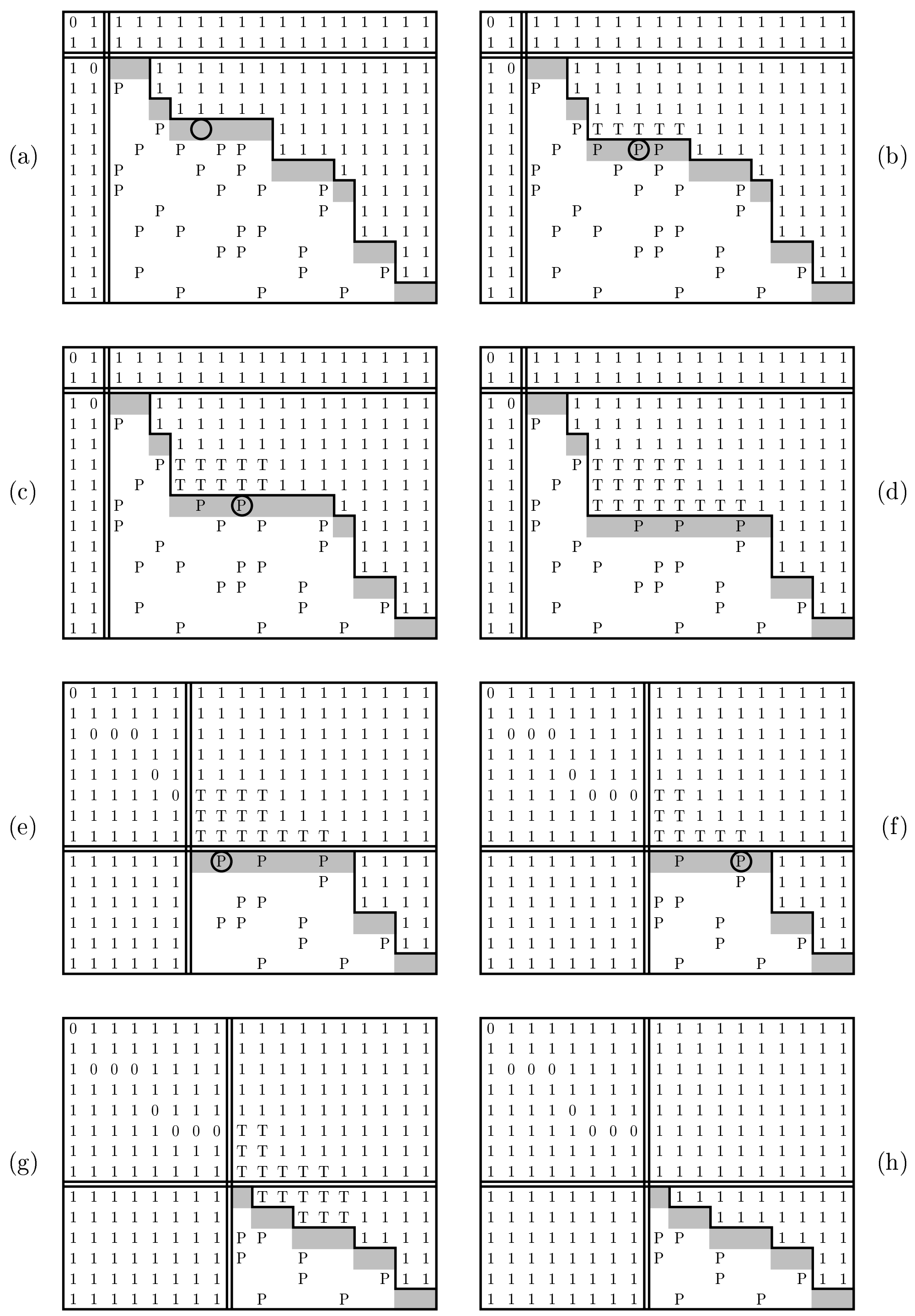

Figure 3: An example execution of Process.

OneProbe is executed. Let us assume that the argument of this call is the leftmost eligible surface position, $(9, 8)$, and that the condition $c < \min L$ is satisfied. Then the call PutZero(8) leads to the situation in (f). The effect of three more calls of OneProbe, none of which is assumed to call PutZero, is shown in (g). Since the surface in (g) contains no elements of P, no further calls of OneProbe are initiated. The call Process$(6, 7)$ finally converts all occurrences of 'tentative-1' to 1—we call this step, shown in the transition from (g) to (h), the *consolidation*—and returns the value of $J[6, 7]$, which is 0.

5.2 Properties of the strategy

We first show in a series of lemmas that the answers provided by the adversary are consistent with a fixed input I, by which we mean that each query (r, c) is answered by $I[r, c]$. We also argue (Lemma 5.8) that I can be chosen to be sorted, i.e., as a legal input to the string-membership and leftmost-all-1 problems.

Lemma 5.3 *For all $(r, c) \in \{1, \dots, k\} \times \{1, \dots, n\}$, the value of $J[r, c]$ can change only according to the transitions indicated in Fig. 4. Moreover, no occurrences of 'tentative-1' are present in J at the start or end of a call of* Process, *and no call of* Process *returns the value 'tentative-1'.*

$$\begin{tikzcd}
 & & 0 \\
\text{`?'} \arrow[urr] \arrow[r] \arrow[drr] & \text{`tentative-1'} \arrow[ur] \arrow[dr] & \\
 & & 1
\end{tikzcd}$$

Figure 4: The possible transitions of an entry in J.

Proof. The claim concerning 'tentative-1' is a consequence of the consolidation. It is clear from a simple inspection of the code that following the initialization, no '?' is stored in J, and that no occurrence of 1 is replaced by a different value. Moreover, an occurrence of 0 could be changed only in a call of PutZero. What remains, therefore, is to show that no 0 in a column rejected in a call of PutZero is changed by a subsequent call of PutZero. But this is obvious. □

Lemma 5.4 *Every call* Process(r, c) *returns a value in* $\{0, 1\}$.

Proof. By Lemma 5.3, the claim is obvious if (r, c) is buried or $J[r, c] \neq$ '?' at the start of the call. But otherwise a call OneProbe(r, c) is executed, which can easily be seen to store a value different from '?' in $J[r, c]$. The lemma now follows by a second appeal to Lemma 5.3. □

Lemma 5.5 *All positions whose entries are set to 0 during a call of* Process *were surface positions at the start of the call of* Process.

Proof. The only occasions on which a 1 is stored in a position in J are when the column containing the position is rejected and during the final consolidation. By this observation and

Lemma 5.3, immediately before a call of Process causes a column to be rejected, the topmost position in the column that contains a value different from 1 is the same as it was at the start of the call of Process, at which time it was a surface position. □

Lemma 5.6 *The matching area at all times is monotonic.*

Proof. The matching area initially is empty, hence monotonic. It is changed through only two types of operations, the removal of a number of leftmost (rejected) columns and the inclusion of a surface position and all positions to its right that do not already belong to the matching area. Both operations preserve monotonicity. □

Lemma 5.7 *No column of J is ever rejected while it contains the string 1^k.*

Proof. Suppose that $1 \le c \le n$ and that column c of J is rejected while containing 1^k. Then column c must contain 1^k already at the start of the call of Process that rejects it. At that time, by Lemmas 5.3 and 5.6, all columns to the right of column c also contain 1^k, in which case column c cannot be rejected. □

Lemma 5.8 *The answers provided by the adversary are consistent with a sorted input.*

Proof. Let I be the input obtained from the final value of J by changing to 1 all entries that are not 0. We prove that each query (r, c) is answered by $I[r, c]$ and that I is sorted.

A query (r, c) is answered by 0 only if $J[r, c] = 0$ at the time of the answer and thus, by Lemma 5.3, only if $I[r, c] = 0$. The same argument applies to answers of 1, except that, because of the probes of buried positions, we must additionally show that no entry in J of a position that belongs to P at some time is ever set to 0. Assume, to the contrary, that such an entry is set to 0 in some call of Process. By Lemma 5.3, the value of the entry must have been '?' at the start of the call of Process; i.e., the corresponding position still belonged to P at that time. But given that P contained no surface positions at that time, due to the termination condition of the while loop in Process, this contradicts Lemma 5.5.

We now show that I is sorted and begin by observing that no column of I contains more than one 0. Let $1 \le c < c' \le n$ and $1 \le r' \le k$ and suppose that $I[r', c'] = 0$. By Lemmas 5.3 and 5.7, we must have $I[r, c] = 0$ for some $r \in \{1, \ldots, k\}$. In order to complete the demonstration that I is sorted, it suffices to show that $r \le r'$.

By Lemma 5.5, (r, c) belongs to the surface at some time τ, and (r', c') belongs to the surface at the same or some later time (since column c' is rejected simultaneously with or later than column c). If $(\bar{r}, c')$ is the surface position in column c' at time τ, we have $r \le \bar{r}$ by the monotonicity of the matching area at time τ (Lemma 5.6) and $\bar{r} \le r'$ because, as long as a column contains a surface position, the row index of its surface position never decreases (this is a consequence of Lemma 5.3). Thus indeed $r \le r'$. □

Lemma 5.9 *When $\mathcal{A}$ terminates, either all rows except at most one have been excluded, or all entries in the last row outside of the rejected columns have been fixed to* 1.

Proof. Assume that $\mathcal{A}$ terminates while the entry of some position (k, c) in the last row is still unfixed. We complete the tableau J to two inputs I_0 and I_1: I_1 is obtained simply by fixing all remaining unfixed entries of I to 1. I_0 is obtained as follows: If $c > 1$, first $\mathsf{PutZero}(c-1)$ is executed. Then the entry in position (k, c) is fixed to 0 and all remaining unfixed entries of I are fixed to 1.

The inputs I_0 and I_1 are both consistent with all answers obtained by $\mathcal{A}$. It was already argued in the preceding proof that I_1 is sorted, and using essentially the same argument, it can be seen that I_0 is sorted as well, so that I_0 and I_1 are both legal inputs. Moreover, column c of I_0 contains the string $1^{k-1}0$, column c of I_1 contains 1^k, and if two or more rows had not been excluded when $\mathcal{A}$ terminated, no column of I_1 contains $1^{k-1}0$. But in that case, whether $\mathcal{A}$ is an algorithm for the string-membership problem with query string $1^{k-1}0$ or for the leftmost-all-1 problem, I_0 and I_1 are not associated with a common correct output. Thus $\mathcal{A}$ cannot produce its answer, a contradiction. □

Recall that the *gap* of a fence is its distance from the 0-barrier and that its *cumulative weight* is the sum of its own weight and the weights of all fences to its right. We define the *bias* of a fence F of gap g and cumulative weight w as the quantity

$$Bias(F) = \frac{\ln(n+1) - \ln g - 2aw}{-\ln(1-e^{-a})}.$$

We apply this definition even to an imaginary fence F_0 in column $n+1$ and of cumulative weight 0 and define the *maximum bias* B as $\max_{i=0}^{N} Bias(F_i)$, where $\mathcal{F} = (F_N, \dots, F_1)$.

Lemma 5.10 *$B = 0$ initially, $B \geq 0$ always, and a call of* $\mathsf{PutZero}$ *or* $\mathsf{OneProbe}$ *increases B by at most 1.*

Proof. The first two claims are obvious, the second one because $Bias(F_0) \geq 0$. It is not difficult to see from the definition of the set of legal fence-column indices that after the rejection of one or more columns, the gap of every surviving fence is at least $1 - e^{-a}$ times what it was before the rejection—every potential new 0-barrier is at most a fraction of e^{-a} of the way from the current 0-barrier to the fence. A call of $\mathsf{PutZero}$ or $\mathsf{OneProbe}$ that causes columns to be rejected therefore increases B by at most 1.

If a new fence F is created from scratch at a time when the previous leftmost fence F' (F_0 if there are no fences) has gap g, we have $e^{-a}g \geq 2$, so that the gap of F is at least $e^{-a}g - 1 \geq e^{-2a}g$. Since the cumulative weight of F is one more than that of F', we will have $Bias(F) \leq Bias(F')$. Thus a call of $\mathsf{OneProbe}$ that does not cause columns to be rejected also does not increase B. □

Note that since a critical fence is excluded immediately after its creation, the number of fences never exceeds t.

5.3 Analysis of the number of probes

The number of probes needed is bounded from below using the potential function

$$\Phi = \Phi_1 + \Phi_2 + \Phi_3 + \Phi_4,$$

where $\Phi_1 = \sum_{F \in \mathcal{F}} \Phi(F)$ is the total value of all fences, $\Phi_2 = tX_{\mathrm{R}}$ is t times the number of excluded rows, $\Phi_3 = 4|P|$ is four times the number of pending positions, and $\Phi_4 = B$ is the maximum bias.

Lemma 5.11 *If $\Phi \le kt$, then the gap of every fence, including the imaginary fence F_0, is greater than $n^{1/4}$.*

Proof. Assume that $\Phi \le kt$ and that the gap g of some fence F (possibly F_0) is at most $n^{1/4}$ and let w be the cumulative weight of F. By the mean-value theorem, $\ln(1-x) \ge -2x$ for $0 \le x \le 1/2$. Using this with $x = e^{-a} = \ln n/(k \ln\ln n)$ and assuming that $\mathit{Bias}(F) \ge 0$, we obtain

$$\mathit{Bias}(F) = \frac{\ln(n+1) - \ln g - 2aw}{-\ln(1-e^{-a})} \ge (\ln(n+1) - \ln g - 2aw) \cdot \frac{k \ln\ln n}{2\ln n} \ge \left(\frac{3}{4} - \frac{2aw}{\ln n}\right) \cdot 4kt.$$

Since $\mathit{Bias}(F) \le kt$, we must have $2aw/\ln n \ge 1/2$. This relation holds also if $\mathit{Bias}(F) < 0$. In either case, Equation (5) therefore implies that $w \ge t\sqrt{\ln n}$. In particular, $F \ne F_0$. Let $\mathcal{F} = (F_N, \ldots, F_1)$. Then $1 \le N \le t$ and therefore, by Equation (3),

$$\frac{w/N}{v^{t-1}} \ge \frac{w/N}{a^{2t}} \ge e^{\ln(w/t) - 2t\ln a} \ge e^{\ln(\sqrt{\ln n}) - (\ln\ln n)/4} = e^{(\ln\ln n)/4}.$$

Now, by the convexity of ϕ_1,

$$\begin{aligned}
\Phi &\ge \sum_{i=1}^{N} \phi_i(\|F_i\|) \ge \sum_{i=1}^{N} \phi_1(\|F_i\|) \ge N\phi_1\left(\frac{1}{N}\sum_{i=1}^{N}\|F_i\|\right) \\
&\ge N\phi_1(w/N) \ge \frac{w}{t}\left(\frac{w/N}{v^{t-1}}\right)^{a/t} \ln\left(\frac{w/N}{v^{t-1}}\right) \\
&\ge \frac{\ln n}{4at} \cdot e^{(\ln\ln n)/4 \cdot a/t} \cdot \frac{\ln\ln n}{4} \ge \frac{\ln n}{2a} \cdot e^{2a\ln a} \cdot \ln a \ge e^a \ln n = k \ln\ln n > kt,
\end{aligned}$$

a contradiction. □

Lemma 5.12 *During the execution of $\mathcal{A}$, Φ increases by at least $(k-1)t$.*

Proof. $\Phi = 0$ initially, so let us consider the situation when $\mathcal{A}$ terminates and show that $\Phi \ge (k-1)t$. This is obvious if all except at most one row have been excluded, so assume that this is not the case and that $\Phi \le kt$. Let g be the gap of an arbitrary fence, or of F_0 if no fence exists. By Lemma 5.9, we have $|P| \ge g - 1$ and hence, by Equation (6), $g \le kt + 1 \le n^{1/4}$. Lemma 5.11 now shows that $\Phi > kt$, a contradiction. □

Lemma 5.13 *Every call of* PutZero *increases Φ by at most* 1.

Proof. The call increases neither Φ_1 nor Φ_3. By Lemma 5.10, Φ_4 increases by at most 1. If the call causes the exclusion of $m > 0$ rows, some fence of height m was critical—we here use the fact that no two fences are ever in adjacent columns—so the corresponding increase in Φ_2 of tm is compensated by a decrease in Φ_1 of at least tm caused by the exclusion of a dense fence of height m or by a reduction by m in the height of each of t fences. □

Lemma 5.14 *If $\Phi \le kt$ before a call of* OneProbe, *the call increases Φ by at most* 4.

Proof. Unless a call of OneProbe simply executes a call of PutZero, which increases Φ by at most 1 according to the previous lemma, it begins by extending a fence F' by one position or creating a new fence F' from scratch. Extending an existing fence by one position increases its value by 1, and the value of a fence of height and weight 1 is bounded by 3; we here use the fact that there are never more than t fences, so that we never employ ϕ_i for $i > t$. Until this point, therefore, Φ has increased by at most 3.

Subsequently to the operation on the fence F', it may disappear through exclusion because it is in the column next to the 0-barrier, as part of the execution of a call of PutZero, or through merging with its right neighbor. The first case is ruled out by Lemma 5.11 in conjunction with Equation (5), which shows that $e^{-a} \cdot n^{1/4} \ge n^{1/4-1/(4t)} \ge n^{1/8} \ge 2$, where the last inequality follows from the assumption $t \ge 2$. The second case increases Φ by another at most 1, according to Lemma 5.13, for a total increase in Φ of at most 4. What remains, therefore is to assume that F' merges with its right neighbor F and show that this does not increase Φ.

Let the indices of F and F' be i and $i+1$, respectively. The new fence resulting from the merge has index i, and the remaining fences are affected by the merge only insofar as some of them decrease their index by 1; since this decreases Φ, we need not account for it here. What is to be shown, hence, is that the value of the new fence resulting from the merge is no larger than the combined value of the two fences from which it is formed.

All three fences of interest have the same height $|F|$. Let us write $\|F'\| = u\|F\|$, where $u > 0$ is a suitable real number. Then the weight of the new fence is $(1+u)\|F\|$, so that the relation to be shown is

$$(|F| + \|F\| + \phi_i(\|F\|)) + (|F| + u\|F\| + \phi_{i+1}(u\|F\|)) - (|F| + (1+u)\|F\| + \phi_i((1+u)\|F\|)) \ge 0$$

or, equivalently,

$$|F| + \phi_i(\|F\|) + \frac{\phi_i(uv\|F\|)}{v} - \phi_i((1+u)\|F\|) \ge 0.$$

The derivative of the left-hand side above with respect to u is

$$\|F\|\phi_i'(uv\|F\|) - \|F\|\phi_i'((1+u)\|F\|).$$

Recall that ϕ_i' is a strictly increasing function. This implies that the original left-hand side has a unique minimum that occurs for the value of u satisfying $uv = 1+u$, i.e., $u = 1/(v-1)$. It therefore suffices to prove the original claim for this value of u, i.e., to show that

$$|F| + \phi_i(\|F\|) + \frac{1}{v}\phi_i\left(\frac{v}{v-1}\|F\|\right) - \phi_i\left(\frac{v}{v-1}\|F\|\right) \ge 0$$

or, equivalently, that

$$\frac{1}{q}\phi_i(q\|F\|) - \phi_i(\|F\|) \le |F|,$$

where we introduced the abbreviation $q = v/(v-1)$. Note that $q = 1 + 1/(3a)$ and hence

$$q^{a/t} = \left(1 + \frac{1}{3a}\right)^{a/t} \le e^{1/(3t)}.$$

Take $z = \|F_i\|/v^{t-i}$. Then

$$\begin{aligned}
\frac{1}{q}\phi_i(q\|F\|) - \phi_i(\|F\|) &= \frac{\|F\|}{t}(qz)^{a/t}\ln(qz+e) - \frac{\|F\|}{t}z^{a/t}\ln(z+e) \\
&\le \frac{\|F\|}{t}z^{a/t}\left[q^{a/t}(\ln q + \ln(z+e)) - \ln(z+e)\right] \\
&= \frac{\|F\|}{t}z^{a/t}\left[(q^{a/t}-1)\ln(z+e) + q^{a/t}\ln q\right] \\
&\le \frac{\|F\|}{t}z^{a/t}\left[(e^{1/(3t)}-1)\ln(z+e) + e^{1/(3t)}\frac{1}{3a}\right].
\end{aligned}$$

By Equation (4), we can bound the right-hand side above by

$$\frac{\|F\|}{2t^2}\cdot z^{a/t}\ln(z+e) + \frac{\|F\|}{2at}\cdot z^{a/t} = \frac{\phi_i(\|F\|)}{2t} + \frac{\|F\|}{2at}\cdot z^{a/t}.$$

Since critical fences are excluded as soon as they arise, F is not dense at the time of the merge. Thus the first term of the right-hand side above is bounded by $|F|/2$, and Lemma 5.2 shows that the same is true of the second term. This completes the proof of the lemma. □

Lemma 5.15 *If $\Phi \le kt - 4$ before a call of* Process, *the call increases Φ by at most* 4.

Proof. Consider a call Process(r, c). If (r, c) is buried, the call increases $|P|$ by at most 1 and Φ by at most 4. Otherwise the call executes a call of OneProbe and subsequently, zero or more times, executes a call of OneProbe and decreases $|P|$ by at least 1 and hence Φ by at least 4. By a simple induction based on Lemma 5.14, $\Phi \le kt$ at the start of each call of OneProbe, and Φ altogether increases by at most 4. The final consolidation does not affect Φ. □

Theorem 5.1 follows from Lemmas 5.8, 5.12 and 5.15.

6 Conclusions

We have given tight bounds for a fundamental searching problem. The problem is natural and easy to formulate, yet the solution—the bound achieved as well as its proof—is surprisingly complicated.

As mentioned in the introduction, the problem becomes much easier if preprocessing and extra space is allowed. It should be noted that our lower bound imposes no restrictions on the model of computation other than the absence of preprocessing; a search algorithm is allowed to use extra memory and arbitrary data structures during its execution.

Acknowledgements

We thank Jyrki Katajainen for bringing the papers by Hirschberg [3, 4] to our attention.

References

[1] A. Andersson, T. Hagerup, J. Håstad, and O. Petersson, *The complexity of searching a sorted array of strings*, in Proc. 26th Annual ACM Symposium on the Theory of Computing, 1994, pp. 317–325.

[2] A. Andersson, J. Håstad, and O. Petersson, *A tight lower bound for searching a sorted array*, in Proc. 27th Annual ACM Symposium on the Theory of Computing, 1995, pp. 417–426.

[3] D. S. Hirschberg, *A lower worst-case complexity for searching a dictionary*, in Proc. 16th Annual Allerton Conference on Communication, Control, and Computing, 1978, pp. 50–53.

[4] D. S. Hirschberg, *On the complexity of searching a set of vectors*, SIAM J. Comput., 9 (1980), pp. 126–129.

[5] S. R. Kosaraju, *On a multidimensional search problem*, in Proc. 11th Annual ACM Symposium on Theory of Computing, 1979, pp. 67–73.

[6] U. Manber and G. Myers, *Suffix arrays: A new method for on-line string searches*, SIAM J. Comput., 22 (1993), pp. 935–948.

[7] K. Mehlhorn, *Data Structures and Algorithms, Vol. 1: Sorting and Searching*, Springer-Verlag, Berlin, 1984.