In Proc. 26th ACM Symposium on Theory of Computing, pages 317--325. ACM Press, 1994.

The complexity of searching a sorted array of strings

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

Abstract

We present an algorithm for finding a given k-character string in an array of n strings, arranged in alphabetical order, using

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

character comparisons. This improves significantly upon previous bounds.

1 Introduction

Given n strings arranged in alphabetical order, how many characters must we probe to determine whether a k-character query string is present? If k is a constant, we can solve the problem with $\Theta(\log n)$ probes by means of binary search, and this is optimal, but what happens for larger values of k?

The question is a fundamental 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 more conservative assumption that only single characters can be compared in constant time. For 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 a surprising intricacy.

Being slightly more precise, we consider finite strings of characters drawn from an arbitrary, ordered alphabet. Strings are ordered linearly by the induced lexicographical ordering, and we study the following string searching problem: Given a sorted array A of n strings of k characters each and a query string X of k characters, determine the rank of X in A, i.e., the number of strings in A no larger than X.

The string searching problem was introduced by Hirschberg [2], who indicated a lower bound of $\Omega(k+\log n)$ and upper bounds of $O(k \log n)$ and $O(k+n)$, all of which are straightforward. A later publication by the same author [1] mentions a first nontrivial upper bound of $O(k\log n/\log k)$. Kosaraju [3] gave an algorithm with a running time of $O(k\sqrt{\log n}+\log n)$ and a lower bound of roughly $\log n + \frac{1}{2}\sqrt{k\log n} = O(k+\log n)$.

In this paper we establish a new upper bound of

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

In view of the lower bound of $\Omega(k+\log n)$, this is optimal for small and for large values of k; specifically, for $k = O(\log n/\log\log n)$ and for $k=\Omega\left(2^{(\log n)^{\epsilon}}\right)$, for any $\epsilon > 0$. Furthermore, the running time of our algorithm never exceeds the known lower bound by more than a factor of $\Theta(\log\log n)$. Ongoing work, reported briefly in Section 5, aims at proving that our algorithm in fact is optimal for all combinations of n and k.

By way of comparison, Kosaraju's algorithm is optimal for $k = O(\sqrt{\log n})$, and Hirschberg's algorithm is optimal for $k=\Omega(n^{\epsilon})$, for any $\epsilon > 0$. We further note that for k asymptotically larger than $\log n$, our bound contradicts the claim that $\log n + k\log\log n$ is not achievable [3].

2 The one-sided problem

Consider the following one-sided version of our problem: A contains only 0's and 1's and the query string X consists of $k-1$ 1's followed by one 0. We will view A as a matrix in which we seek the position of the leftmost column containing only 1's. At first, this problem might seem easier than the one that we really want to solve; however, its solution will essentially be sufficient to solve the original problem.

We start by describing a search algorithm for the one-sided problem in generic terms. To find X the algorithm explores A; during the process the knowledge gained divides the matrix A into three disjoint areas: columns to the left of the rightmost column (inclusive) in which a 0 has been encountered belong to the *rejected area*; the part of A defined by the set of known matching prefixes outside of the rejected

*Department of Computer Science, Lund University, Box 118, 221 00 Lund, Sweden. arne@dna.lth.se

†Max-Planck-Institut für Informatik, Im Stadtwald, D–66123 Saarbrücken, Germany. Supported by the ESPRIT Basic Research Actions Program of the EU under contract No. 7141 (project ALCOM II). torben@mpi-sb.mpg.de

‡Department of Computer Science, Royal Institute of Technology, 100 44 Stockholm, Sweden. johanh@nada.kth.se

§Department of Computer Science, Lund University, Box 118, 221 00 Lund, Sweden; and Department of Mathematics, Statistics, and Computer Science, Växjö University, 351 95 Växjö, Sweden. ola@dna.lth.se

area make up the *matching area*; the remaining part of A is called the *uncertainty area.*

Figure 1(a) illustrates the above notions: the rejected area consists of empty squares and is separated from the rest by double vertical lines; the matching area is filled with 1's; and the uncertainty area is filled with black squares.

Initially the uncertainty area constitutes the entire matrix and the search terminates when it is empty. The algorithm to be described only probes entries of the uncertainty area. Furthermore, the entries probed are on the first row of A or located immediately below the matching area. This implies that whenever a 1 is encountered, the entry probed and all entries to the right of it must also be 1's, and become part of the matching area. If a 0 is found, all columns to its left (inclusive) belong to the rejected area. Hence, going from left to right, every row in A consists of three parts (each of which can be empty): a rejected part; an uncertainty part; and a matching part. Rows with no uncertainty part are *excluded* from the search. In Figure 1(a) excluded rows are separated from the rest by double horizontal lines.

The search makes two different kinds of probes, both aiming to decrease the uncertainty part of some row:

Horizontal probe: Probe at the middle of the uncertainty part of the top row (i.e., the topmost row which has not been excluded). More precisely, if the uncertainty part of the top row contains l entries, probe the $\lceil l/2 \rceil$th of these, counting from the left. Regardless of whether we find 0 or 1, the uncertainty part of the row will shrink by a factor of at least 2; if we find 0 the uncertainty parts of all other rows will also shrink. The two cases are illustrated in Figure 1(b) and (c).

Vertical probe: Consider two adjacent rows where the upper one has a smaller uncertainty part. Try to make the two parts the same width by probing immediately below the leftmost entry of the upper row's matching part. If we find a 1 we succeed. If we find a 0, the areas will change differently; the column in which the probe was made will be rejected, the uncertainty parts of all rows above the 0 encountered will disappear, and hence those rows are excluded. (See Figure 1(d) and (e).)

We now describe how an algorithm for the one-sided problem can be applied to the original problem. In every algorithm that we consider, a character probed in A will only be compared with the character of X in the same position. Therefore, the only relevant property of a character in A is whether it is less than, equal to, or greater than the corresponding character in X. Hence, we may assume that A contains only 0's, 1's, and 2's and that X contains only 1's.

The problem will be solved by making two one-sided searches, S_L and S_R, in an alternating fashion, to be made more precise below. Basically, the searches will work their way towards X from opposite directions. S_L rejects columns smaller than X and S_R rejects columns larger than X. We concentrate on one of the searches, say S_L. S_L will regard some columns as *passive*; namely, those containing strings that are known to be larger than X. That is, the leftmost column outside of the rejected area in which a 2 has been found and all columns to its right. Any probe made in a passive column is interpreted as a 1 by S_L. With this modification all horizontal probes and the vertical probes that give 1's have the same effect as in the one-sided case.

The only situation in which interaction with S_R is required is when we find a 0 at some row p in A at a vertical probe. Then we aim to exclude all rows above the entry probed. We note that, in order to exclude a row, neither S_L nor S_R may have an uncertainty part on that row. To ensure this, S_L is interrupted; we say that it is *waiting* at row p. Observe that when S_L is waiting at row p, its uncertainty parts on all rows above the p'th are empty. This is the only occasion when a search is interrupted. At this point S_R will be waiting at some row q. (Both searches will wait at row 0 at the beginning of the search.) Hence, both searches are waiting and we have to decide which one to restart. If $p < q$ we restart S_L; otherwise we restart S_R. W.l.o.g. we can assume that $p < q$. Then, since S_R is waiting below row p, we know that S_R has no uncertainty part at row p. Hence, S_L can exclude all rows above row p, and then continue.

In this way, the two searches will be run as coroutines, waiting for each other to catch up. Ties are easily broken, and so no deadlock occurs. The search is completed as soon as either one of S_L and S_R terminates.

The informal discussion in this section is summarized in the following lemma, stated without proof.

Lemma 2.1 *Given any algorithm for solving the one-sided problem using at most T probes, there is a corresponding algorithm that solves the string searching problem using at most $2T$ probes.*

3 The algorithm

This section describes an algorithm that solves the one-sided problem. Without loss of generality we assume that $n \geq 4$. The algorithm centers around the notion of a *fence*. A fence resides in a certain column and consists of the part of the column which is contained in the matching area. (From the description in the previous section it follows that every fence occupies a contiguous upper portion of its column.) A fence is *extended* by one or more vertical probes. An extension is *successful* if all probes return 1's; otherwise it *fails.* Every fence F has a *height*, $|F|$, defined as the number of rows spanned by F; excluded rows are not counted. A fence is *excluded* if, as a result of row exclusions, its height is reduced to zero.

The algorithm maintains an ordered collection $\mathcal{F} = \{F_t, F_{t-1}, \ldots\}$ of fences in selected columns, where t is a parameter to be fixed later. For $i = t, t-1, \ldots$, we denote by $pos(F_i)$ the index of the column containing F_i. We will always have $pos(F_t) > pos(F_{t-1}) > \cdots$, i.e., the fences in $\mathcal{F}$ are numbered consecutively from left to right, starting at an index that varies during the execution of the algorithm as a function of the number of fences in $\mathcal{F}$. Besides these fences, there is an imaginary fence F_{t+1} of infinite height at position $n+1$.

The uncertainty part of the top row will play an important role, and we let m denote the logarithm of its width; this expresses how many more horizontal probes are sufficient to reduce it to zero. That is, m is the logarithm of the distance from the rightmost *known* 0 on the top row to the leftmost fence of $\mathcal{F}$. (Henceforth, unless stated otherwise, the term 'rightmost 0' refers to the rightmost known 0 on the topmost nonexcluded row.) If we ever fail to extend a fence, the rejected area will expand to the right, and so the rightmost 0 will 'move' to the right. We thus get a new leftmost fence, which might cause m to change. Therefore, with each fence $F_i \in \mathcal{F}$ we keep a number m_i, which should be thought of as the new value of m if an extension of F_i were to fail. A natural choice for m_i would be

$$m_i = \log(pos(F_{i+1}) - pos(F_i)).$$

However, during the course of the algorithm F_i will move to the left a number of times, and this would cause a number of increases in m_i. Thus, for technical reasons we choose to define

$$m_i = \log(pos(F_{i+1}) - \text{the position of the rightmost 0 when } F_i \text{ is created}).$$

Since F_i will never move left of the rightmost 0 this definition will always give a larger number than the former one. Every fence F_i will be created by a horizontal probe, that is, in the middle between F_{i+1} and the rightmost 0. Moreover, a fence will only move left, and consequently, the current definition of m_i is always within an additive 1 of the more intuitive definition. The advantage of our definition is that m_i will not change during the lifetime of F_i. This follows since whenever F_{i+1} moves in the algorithm, F_i is deleted.

The number of probes performed by the algorithm will depend on the number of fences. To control this quantity the algorithm maintains an invariant that puts a restriction on how close neighboring fences are allowed to be. First, for all $F_i \in \mathcal{F}$, define

$$\Delta_i = m_i - m_{i-1},$$

where if F_i is the leftmost fence, $m_{i-1} = m$. Δ_i captures how much closer F_i is to the rightmost 0 than is F_{i+1}. Next, define

$$c = \sqrt{\max\left\{16, \log\left(\frac{k \log\log n}{\log n}\right)\right\}} \geq 4.$$

The invariant says that, going from left to right, the Δ_i's increase exponentially with ratio at least c:

Invariant 1 *$\Delta_{i+1} \geq c\Delta_i$ for all $F_i \in \mathcal{F} - \{F_t\}$.*

As demonstrated in Lemma 4.1 below, Invariant 1 ensures that the number of fences is at most $t+1$.

The height of a fence is determined by a function H, defined as

$$H(x) = \left\lceil \frac{x^{1+ec/t}}{t} \right\rceil,$$

where e is the basis of the natural logarithm and

$$t = \left\lceil \frac{\log\log n}{\log c} \right\rceil.$$

Define the *target height* of a fence F_i to be $H(\Delta_i)$. The algorithm will ensure that fences attain their target height:

Invariant 2 *$|F_i| = H(\Delta_i)$ for all $F_i \in \mathcal{F}$.*

One consequence of the two invariants, which could be kept in mind when studying the algorithm, is that as H is superlinear and since the Δ_i's increase exponentially, the heights of the fences will increase exponentially, going from left to right. (See Lemma 4.3.)

Initially, we only have the imaginary fence F_{t+1} and $m = \log n$. The algorithm repeatedly executes a *round*, which consists of a sequence of three steps, the last two of which are repeated as long as necessary: (1) Make a horizontal probe; (2) Restore Invariant 2; (3) Restore Invariant 1.

In order to avoid unnecessary repetition when describing the three steps in more detail, let us first clarify a few things. If the leftmost fence F_d is extended to the same height as its neighbor F_{d+1} then F_d is removed from $\mathcal{F}$, and the position of F_{d+1} is changed to that of F_d. We say that F_{d+1} has *moved* to (the position of) F_d.

If we encounter a 0 when extending a fence F_i of height h then the top h rows are excluded, and as a result of this F_i and all fences to its left are excluded and removed from $\mathcal{F}$.

We are now ready for a detailed description of a round:

(1) *Make a horizontal probe:* If we find a 1, this results in either a new leftmost fence (if there was no fence of height one), which is put in $\mathcal{F}$; or (if there already was a fence of height one) the moving of the leftmost fence to the left.

Step (1) might cause either or both of the invariants to be violated, and the remainder of the round reestablishes these (this is not immediate, but will be proved below). We repeatedly execute Steps (2) and (3) until the precondition of neither step holds. Whenever a 0 is encountered, Step (2) is restarted.

(2) *Restore Invariant 2:* If Invariant 2 is violated at any fence then for each fence $F_i \in \mathcal{F}$, in the order of appearance from right to left, attempt to extend F_i to its target height.

(3) *Restore Invariant 1:* If Invariant 1 is violated at the leftmost fence F_d then attempt to move F_{d+1} to F_d.

Some special cases have not been taken into account in the general description. If a probe executed during some round ends the search, of course, the rest of that round is aborted. Furthermore, the description may specify the extension of a fence that already spans all rows. In this case, although in actual fact the probe is skipped, we will pretend that it takes place, extending the fence into fictitious rows below the real rows; we assume that fictitious positions contain 1's only.

The analysis of the algorithm will of course rely upon the invariants, and we need to verify that they are maintained. As for Invariant 2 this follows trivially from Step (2). To see that Invariant 1 is satisfied is not that obvious: We only restore it at the leftmost fence—is it always satisfied at all remaining fences? We next show that this is indeed the case. (For the purpose of subsequent use, the following lemma is stronger than needed at this point.)

Lemma 3.1 *Consider a fence F_i at some point during a round. Let Δ_i' denote its Δ-value before the round started, and let Δ_i denote its current Δ-value. Furthermore let F_d be the leftmost fence at the start of the round. If F_i is the leftmost fence, then*

$$\Delta_i' \leq \Delta_i \leq 1 + \sum_{j=d}^{i} \Delta_j';$$

otherwise, $\Delta_i = \Delta_i'$. Here Δ_j' should be interpreted as zero if F_j did not exist at the beginning of the round.

Proof. Similarly as in the statement of the lemma, in the following m denotes the current uncertainty on the first row and m' denotes its value at the start of the round. The same convention applies to m_j and m_j'.

Consider first how Step (1) affects the Δ-values. Since a horizontal probe is made in the middle of the uncertainty part of the top row, after the probe $m = m' - 1$, independently of the outcome.

First, suppose we find a 0 or a 1 that does not give rise to a new leftmost fence. Then, at the leftmost fence we have

$$\Delta_d = m_d - m = m_d' - (m' - 1) = \Delta_d' + 1,$$

and at any fence F_i with $i \geq d + 1$ we have

$$\Delta_i = m_i - m_{i-1} = m_i' - m_{i-1}' = \Delta_i'.$$

If we find a 1 that gives a new leftmost fence F_{d-1}, at this fence

$$\Delta_{d-1} = m_{d-1} - m = m' - m = 1,$$

while at any fence F_i with $i \geq d$ we have

$$\Delta_i = m_i - m_{i-1} = m_i' - m_{i-1}' = \Delta_i'.$$

We conclude that after Step (1) the Δ-value of the leftmost fence has increased by one, and the Δ-values of all remaining fences are unaltered.

Inductively assume that the lemma is true when Step (2) is started. As long as Step (2) is successful extending fences, no Δ-values change. Whenever we encounter a 0, we get a new leftmost fence F_i, the Δ-value of which, prior to finding the 0, was equal to its original value, that is, Δ_i'. After finding the 0, we have

$$\Delta_i = m_i - m = m_i' - m \leq m_i' - (m_{i-1}' - 1) = \Delta_i' + 1.$$

To verify the inequality note that the m expresses the distance between the two fences before we found the 0. We know that F_{i-1} was created in the middle between F_i and the, at that time, rightmost 0, so at this time we have equality. Since then F_i cannot have moved at all (because if it did it would have become the leftmost fence), which leaves the right-hand side of the inequality unchanged; and F_{i-1} can only have moved left, which increases m and decreases the left-hand side. We conclude that the lemma holds after Step (2) by induction.

Consider now Step (3). If we ever encounter a 0, the same argument as in Step (2) applies. Suppose that we succeed in moving F_i to the leftmost fence F_{i-1}. By induction, the Δ-value of F_{i-1} just before the move was at most

$$1 + \sum_{j=d}^{i-1} \Delta_j' = 1 + \sum_{j=d}^{i-1} (m_j' - m_{j-1}')$$

$$= 1 + m_{i-1}' - m' = 1 + m_{i-1} - m',$$

where the first equality is by definition, and the third follows since F_i cannot have moved up to now (because when a fence moves it becomes the leftmost fence). On the other hand, the exact Δ-value of F_{i-1} just before the move is $m_{i-1} - m$. Hence,

$$m_{i-1} - m \leq 1 + m_{i-1} - m',$$

which implies that $-m \leq 1 - m'$. After the move we thus have

$$\begin{aligned} \Delta_i &= m_i - m = m_i' - m \leq 1 + m_i' - m' \\ &= 1 + \sum_{j=d}^{i} (m_j' - m_{j-1}') = 1 + \sum_{j=d}^{i} \Delta_j', \end{aligned}$$

which completes the induction.■

We note that the invariants are satisfied initially, and so, by the above lemma and the discussion preceding it, they hold after and thus prior to any round.

4 Analysis

For the purpose of the analysis, the probes performed during a round are divided into three different classes, analyzed in three subsections.

Horizontal probes: All probes of Step (1).

Repairing probes: All vertical probes of Step (2) that extend fences which existed prior to this round to the heights they had prior to this round.

Constructing probes: All probes of Step (3) plus those probes of Step (2) that are not repairing, that is, those that extend new fences and existing fences beyond the heights they had prior to this round.

Throughout the following three subsections, we assume that fences can attain the heights obtained by dropping the rounding in the definition of H. That is, we assume that

$$H(x) = \frac{x^{1+ec/t}}{t}.$$

In Section 4.4 we justify this assumption.

4.1 Repairing probes

The key observation when bounding the number of repairing probes is that we only make such probes after excluding a number of rows, and the number of probes performed repairing a fence to the height it had before the round started equals the number of rows excluded in the round. Hence, the maximum number of repairing probes is given by k times the maximum number of fences that can coexist, that is, the maximum cardinality of $\mathcal{F}$ at the beginning of a round.

Lemma 4.1 *After each round, $|\mathcal{F}| \leq t+1$.*

Proof. Let F_d be the leftmost fence in $\mathcal{F}$ after a round. Then Δ_d exists and is at least one. By Invariant 1, we have $\Delta_t \geq c\Delta_{t-1}$, and thus by iteration, $\Delta_t \geq c^{t-d}\Delta_d$. Taking logarithms yields

$$t-d \leq \frac{\log(\Delta_t/\Delta_d)}{\log c} \leq t,$$

where the second inequality follows from the inequalities $\Delta_t \leq \log n$ and $\Delta_d \geq 1$, coupled with the definition of t. Hence, $d \geq 0$, and the lemma follows.∎

Corollary 4.2 *The number of repairing probes is $O(tk)$.*

4.2 Constructing probes

The number of constructing probes will be bounded in terms of the sum of the heights of the fences present in $\mathcal{F}$ plus the sum of the heights of all fences that have been excluded, for which we count the height as the one they had upon exclusion. We denote this quantity by $\mathcal{H}$. Note that the only fences which do not contribute to $\mathcal{H}$ are those that are deleted as a result of successful moves.

Our first goal is to show that $\mathcal{H} = O(k+\log n/\log\log n)$. We start our development by proving that the heights of the fences in $\mathcal{F}$ decrease exponentially from right to left:

Lemma 4.3 *At any moment, $|F_{i+1}| \geq 4|F_i|$ for all $F_i \in \mathcal{F} - \{F_t, F_d\}$, where F_d is the leftmost fence in $\mathcal{F}$.*

Proof. Before each round we have, by Invariant 1 and Invariant 2,

$$|F_{i+1}| = \frac{\Delta_{i+1}^{1+ec/t}}{t} \geq \frac{(c\Delta_i)^{1+ec/t}}{t} \geq \frac{4\Delta_i^{1+ec/t}}{t} = 4|F_i|,$$

where the last inequality follows by the choice of c.

Step (1) does not affect the height of any existing fence but can introduce a new leftmost fence, which might be too high compared to its right neighbor. Since the inequality held everywhere before the step it holds everywhere except possibly at the new leftmost fence afterwards.

Assume inductively that the claim holds whenever Step (2) is started. As long as Step (2) is successful extending fences the claim will continue to hold, by the order in which the fences are processed. If we encounter a 0 the claim will also hold since the ratio of the heights between any two fences F_{i+1} and F_i that are not excluded increases if both fences lose the same number of rows.

If the extension in Step (3) is successful the leftmost (possibly violating) fence is deleted from $\mathcal{F}$, and so the inequality holds everywhere. If we find a 0 it also continues to hold by the same argument as that used for Step (2).∎

Corollary 4.4 *For any $F_i \in \mathcal{F}$, $\sum_{\{j \leq i | F_j \in \mathcal{F}\}} |F_j| \leq 3|F_i|$.*

Proof. Let F_d be the leftmost fence in $\mathcal{F}$ and assume that $i \geq d+3$, since otherwise the claim is obvious. In the light of Lemma 4.3, we have

$$\sum_{j=d}^{i} |F_j| \leq |F_d| + \sum_{j=d+1}^{i} |F_j| \leq |F_i| + 2|F_i| = 3|F_i|.$$

∎

Lemma 4.5 *$\mathcal{H} = O(k + \log n/\log\log n)$.*

Proof. We bound each of the two sets of fences that contribute to $\mathcal{H}$, starting by fences present in $\mathcal{F}$. Consider the tallest fence F_t. As $\Delta_t \leq \log n$,

$$|F_t| \leq H(\Delta_t) \leq \frac{(\log n)^{1+ec/t}}{t} \leq \frac{\log n}{\log\log n} 2^{ec\log c}\log c,$$

by the definition of t. Consider now the definition of c in terms of k. If $c > 4$, then $k = 2^{c^2}\log n/\log\log n$, in which case $|F_t| = o(k)$. If $c = 4$, then $|F_t| = O(\log n/\log\log n)$. Hence, $|F_t| = O(k+\log n/\log\log n)$. Applying Corollary 4.4 gives that the sum of the heights of all fences in $\mathcal{F}$ is $O(k + \log n/\log\log n)$.

The sum of the heights of excluded fences increases only when rows are excluded. If in some step of the algorithm we find a 0 when extending fence F_j then $|F_j|$ rows are excluded, and the sum of the heights of the excluded fences is $O(|F_j|)$ by Corollary 4.4. Hence the increase in $\mathcal{H}$ is linear in the number of excluded rows. We conclude that the contribution to $\mathcal{H}$ by excluded fences is proportional to the number of excluded rows, which is $O(k)$.∎

Lemma 4.6 *The number of constructing probes is $O(tk + \log n)$.*

Proof. Note first that each 0 found by a constructing probe results in the exclusion of at least one row. Consequently, there can be at most k such probes, and we can thus restrict ourselves to bounding the number of constructing probes that return 1's. We prove the lemma by attributing all such probes to fences that contribute to $\mathcal{H}$ in such a way that a fence of height h gets $O(th)$ probes. The lemma then follows from Lemma 4.5 and the choice of t.

We now describe how the constructing probes are attributed to fences. When extended to its target height for the first time, a fence of height h gets h probes. When a fence F_{i+1} is moved to F_i the probes so far attributed to F_{i+1} are gradually transferred to F_i as follows. Suppose F_{i+1} has been attributed $a|F_{i+1}|$ probes. Then before starting the move we transfer $a|F_i|$ probes from F_{i+1} to F_i. For each subsequent successful vertical probe made at F_i, we attribute that probe to F_i and transfer another a probes from F_{i+1}. We note that at any time during the move, the number of remaining probes attributed to F_{i+1} is $a(|F_{i+1}| - |F_i|)$.

We claim that, during the course of the algorithm, a fence is never attributed more constructing probes than t times its target height, and, moreover, no more than $(12/5)t$ times its actual height. These claims are certainly true initially, and we inductively assume that they hold prior to any probe.

Consider first how a horizontal probe affects the claims. The probe itself is not attributed to any fence, since it is not a constructing probe. By the description of the algorithm, all fences are of target height at this point. A horizontal probe can only increase target heights, and does not change the actual heights, of remaining fences. Hence, by induction, after a horizontal probe no fence is attributed more constructing probes than t times its actual height, which proves both claims.

We next turn to vertical probes. Suppose we make a vertical probe at a fence F_i of height h. We distinguish between two main cases depending on the outcome of the probe.

We find a 0: By induction, our claims hold for all fences that are excluded. Finding a 0 does not decrease the target height of any remaining fence, so by the inductive assumption no such fence will have more probes attributed to it than t times its target height. To prove the second claim, concerning the actual heights, we consider two subcases depending on the step in which the algorithm found the 0.

Case (1): The 0 was found when extending F_i to its target height in Step (2). We establish an upper bound on the decrease in actual heights of all remaining fences by bounding h from above, which in turn is accomplished by bounding the target height of F_i from above. Consider the Δ-value of F_i, Δ_i, just before we hit the 0. In the following, Δ'_j, m'_j and m' refer to the values prior to the *round* in which we encounter the 0. Further, let F_d be the leftmost fence at the start of the round. By Lemma 3.1,

$$\begin{aligned}\Delta_i &\leq 1 + \sum_{j=d}^{i} \Delta'_j \leq 1 + c\Delta'_i/(c-1) \\ &\leq 1 + 4\Delta'_i/3,\end{aligned}$$

where the second inequality follows from Invariant 1 being satisfied at all fences when the round started, and the last inequality follows from the choice of c. The target height of F_i when we find the 0 is thus

$$\begin{aligned}H(\Delta_i) &\leq H(1 + 4\Delta'_i/3) \leq H(7\Delta'_i/3) \\ &\leq H(7\Delta'_{i+1}/12) \leq 7\,H(\Delta'_{i+1})/12,\end{aligned}$$

where the third inequality again follows by Invariant 1, and the last inequality follows from the superlinearity of H. Hence, $h \leq 7\,H(\Delta'_{i+1})/12$.

Using Lemma 3.1 again, before we find the 0, $\Delta_{i+1} = \Delta'_{i+1}$, and so the target height of F_{i+1} is $H(\Delta'_{i+1})$ at this moment. By the description of the algorithm, all fences that remain to the right of F_i, and in particular F_{i+1}, were of target height at the time we hit the 0. Consequently, F_{i+1} and thus any other remaining fence loses at most 7/12 of its height. By induction, for these fences the ratios of the number of probes attributed to the height were at most t before the 0 was found, so they will be at most $(12/5)t$ afterwards.

Case (2): The 0 was found when moving F_{i+1} to F_i in Step (3). (Then, *during* the move the height of F_i exceeds its target height, which is why we need to handle this case separately). Again, by induction, the number of probes attributed to F_{i+1} before starting the move was at most t times its height. Given how the probes previously attributed to F_{i+1} are transferred to F_i during a move, it follows that the number of probes attributed to the remainder of F_{i+1} is at most $t \leq (12/5)t$ times its height. It remains to consider fences to the right of F_{i+1}. The same argument as above yields that when we find the 0, the target heights of F_{i+1} and F_{i+2} are $H(\Delta'_{i+1})$ and $H(\Delta'_{i+2})$, respectively. Moreover,

$$H(\Delta'_{i+1}) \leq H(\Delta'_{i+2}/4) \leq H(\Delta'_{i+2})/4.$$

Hence, since F_{i+1} is not excluded, F_{i+2} and any other fence to its right loses at most one fourth of its height. The claim thus follows by induction in this case as well.

We find a 1: We distinguish between three subcases depending on the nature of the probe: (1) it aimed to move F_{i+1} to F_i or to extend the already moved F_{i+1} to its target height; (2) it was a repairing probe; (3) it aimed to extend F_i to its target height for the first time.

We start with the last two subcases, which are straightforward. A repairing probe does not change the number of probes attributed to any fence but only increases the height of some fence by one, so in case (2) the claims hold by induction. In case (3) the number of probes attributed to F_i equals its height, and so again both bounds hold trivially, by induction.

Case (1): By induction, the number of probes attributed to the extended fence is at most

$$(12/5)th + t + 1 \leq (12/5)t(h+1),$$

where the additive t reflects the fact that, while moving F_{i+1}, in addition to the probe made we might transfer up to t probes from F_{i+1}. It remains to verify that the number of probes attributed to the fence being extended is at most t times its target height. Let Δ denote the Δ-value of F_{i+1} after the move. The target height of F_{i+1} is thus $H(\Delta)$, and, by induction, F_{i+1} gets attributed at most

$$t\,(H(\Delta_{i+1})+H(\Delta_i))+H(\Delta)-H(\Delta_i)$$
$$\leq t\,(H(\Delta_{i+1})+H(\Delta_i))+H(\Delta)$$

probes. Let m_j denote the value when the move starts. Then

$$\Delta=m_{i+1}-m_{i-1}=(m_{i+1}-m_i)+(m_i-m_{i-1})=\Delta_{i+1}+\Delta_i.$$

Therefore, by the superlinearity of H, the number of attributed probes is bounded from above by

$$(t+1)(H(\Delta_{i+1})+H(\Delta_i)).$$

It thus suffices to show that

$$t\,H(\Delta)\geq(t+1)(H(\Delta_{i+1})+H(\Delta_i)),$$

or, equivalently, that

$$\Delta^{1+ec/t}\geq\left(1+\frac{1}{t}\right)\left(\Delta_{i+1}^{1+ec/t}+\Delta_i^{1+ec/t}\right).$$

We know that

$$\Delta=\Delta_{i+1}+\Delta_i\geq\left(1+\frac{1}{c}\right)\Delta_{i+1}\geq e^{1/ec}\cdot\Delta_{i+1},$$

where the first inequality expresses that Invariant 1 is violated (otherwise we would not have moved F_{i+1} in the first place), and the second inequality follows from the mean value theorem. Hence,

$$\begin{aligned}\Delta^{1+ec/t} &\geq (e^{1/ec}\cdot\Delta_{i+1})^{ec/t}\Delta=e^{1/t}\Delta_{i+1}^{ec/t}\Delta\\ &\geq \left(1+\frac{1}{t}\right)\Delta_{i+1}^{ec/t}(\Delta_{i+1}+\Delta_i)\\ &\geq \left(1+\frac{1}{t}\right)\left(\Delta_{i+1}^{1+ec/t}+\Delta_i^{1+ec/t}\right)\end{aligned}$$

and we are done.■

4.3 Horizontal probes

Lemma 4.7 *The number of horizontal probes is $O(tk+\log n)$.*

Proof. At the beginning of the search, we have $m=\log n$ and at the end we have $m\geq 0$. Each horizontal probe decreases m by one. Whenever we find a 0 when trying to extend a fence, m increases; however, we will show that the total increase in m over the entire search is $O(tk)$. We do this by demonstrating that the increase in each round is linear in t times the number of excluded rows, after which the lemma follows by summing over all rounds.

Let F_d be the leftmost fence when our round starts. We consider two different cases depending on the outcome of Step (1). In what follows the Δ_i''s and m_i''s refer to the values prior to this round.

Suppose that Step (1) of our round encounters a 0, or a 1 which does not add any new fence. Let F_i be the fence at which we find the *last* 0 during this round. Then this round increases m' to at most m_i, which is m_i', because F_{i+1} cannot have moved. The increase in m' is thus at most

$$m_i'-m'=\sum_{j=d}^{i}(m_j'-m_{j-1}')=\sum_{j=d}^{i}\Delta_j'\leq 4\Delta_i'/3$$

where the last inequality follows from Invariant 1 and the choice of c.

Since F_i existed (and was of target height) before this round, the number of rows excluded during the round must be at least $H(\Delta_i')>\Delta_i'/t$. (The number of rows will exceed $H(\Delta_i')$ if F_i was repaired during the round.) In this case the increase in m' over the round is thus at most linear in t times the number of excluded rows.

Suppose now that Step (1) of our round adds a new fence F_{d-1} to $\mathcal{F}$, and pick i as above. Then, if F_i existed previously, the same argument as above applies. Otherwise, $i=d-1$ and F_i is the new leftmost fence. But then the leftmost fence after the round will be F_d, which is the same as when the round started. F_d cannot have moved during the round since then F_i would have been deleted. Thus, m' *decreases* by one over the round in this case.■

4.4 Main theorem

Before summing up let us return to the rounding issue. To simplify the analysis we have this far assumed that fences can attain noninteger heights. This is of course not possible in reality, but all heights are rounded upwards, as stated in the definition of H. Hence, the algorithm performs more probes than accounted for in the analysis. We claim, however, that the actual number of probes is at most twice that indicated by the idealized analysis. To see this, note that fences of ideal height at most 1 are already paid for by the horizontal probes, while the actual height of each larger fence is at most twice its ideal height.

If we add the contributions by the different kinds of probes, bounds on which are given in Corollary 4.2 and Lemmas 4.6 and 4.7, and apply Lemma 2.1, we can conclude:

Theorem 4.8 *The string searching problem can be solved with*

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

probes.

5 The lower bound

In this section we outline a proof of a lower bound for the string searching problem that matches our upper bound for many combinations of n and k. More precisely, we study a lower bound on the number of probes needed to solve the string searching problem, restricted in the following way: The input alphabet is $\{0, 1\}$, no input string contains more than one occurrence of 0, and the task is to determine the position of the leftmost string consisting of k 1's.

We start by showing that we can restrict ourselves to algorithms that build fences. After that, we give a brief sketch of an adversary that forces any such algorithm to spend many probes.

5.1 Restricting attention to fence algorithms

A position in the input matrix is *exposed* if all positions above it are contained in the matching area; otherwise it is *buried*. A probe at an exposed position is called a *surface probe*, while a probe at a buried position is called a *buried probe*.

A *fence probe* is a probe either in the topmost row or immediately below the leftmost position in the matching part of some row; clearly, every fence probe is a surface probe. A *fence algorithm* is an algorithm that only makes fence probes. Our goal in this subsection is to show that fence algorithms are as good as general algorithms in the context of the restricted searching problem.

Lemma 5.1 *If the restricted searching problem can be solved using at most T probes, it can be solved using at most T probes by an algorithm that only makes surface probes.*

Proof. Let $\mathcal{A}^+$ be an algorithm that solves the restricted problem in no more than T probes. We show how to use $\mathcal{A}^+$ to solve the restricted problem in no more than T probes using surface probes only.

In order to determine which probes to make, we run $\mathcal{A}^+$ on an *imaginary* input, created on-line. Let I be the real input and I^+ the imaginary input.

The contents of I^+ is fixed on-line, depending on the outcome of the probes made in I. By placing a 0 in some column we mean that we fix the topmost unfixed position in the column to 0 and the rest of the column to 1's. We use $\mathcal{A}^+$ in the following way:

If $\mathcal{A}^+$ probes an already fixed position in I^+ we just return the value fixed for that position; no probe is made in I.

If $\mathcal{A}^+$ makes a buried probe, we fix this position to contain 1. No probe is made in I, but the probe is saved and will be carried out later (as soon as it can be executed as a surface probe).

If $\mathcal{A}^+$ makes a surface probe, a number of saved probes may become connected to the surface. Before fixing the contents of $\mathcal{A}^+$'s probe, we perform these saved probes as surface probes, i.e., from top to bottom (other buried probes may remain in the same or other columns), until they are all done or we find a 0. In the first case, when only 1's are found, we fix $\mathcal{A}^+$'s probe to 1; all positions in I that become part of the matching area are fixed in I^+, filled by 1's.

In the second case, one of the saved probes return a 0, say in column p. We then fix a part of I^+ by placing a 0 in column p and all columns to its left. In this way, the position where $\mathcal{A}^+$ made its probe will contain a 0.

It is obvious that the number of probes made in I does not exceed the number of probes made in I^+. It can further be verified in a relatively straightforward manner that I and I^+ contain 0's in the same columns, and that the columns in I^+ are lexicographically sorted. Hence, we do indeed find the correct rank.■

Lemma 5.2 *If the restricted searching problem can be solved using at most T probes, it can be solved by a fence algorithm using at most $2T$ probes.*

Proof. By the preceding lemma, it suffices to show that every surface probe can be simulated by at most two fence probes.

Suppose that we aim to make a surface probe in row $r \geq 2$. Instead, we begin by carrying out the unique fence probe in row r. If this yields a 1, we know that the probe that we aimed to perform also yields a 1. If it yields a 0, on the other hand, row r becomes the topmost row, and we can execute the original probe as a fence probe.■

5.2 Lower bounds for fence algorithms

In the light of Lemma 5.2, we can discuss our lower bounds using basically the same terminology as for the upper bound. That is, we can phrase the discussion in terms of the number of fences, the heights of fences, vertical probes, etc.

In our upper bound, the parameter t plays an important role, in that the term kt in the complexity can intuitively be thought of as if we spend, on average, $O(t)$ probes per row. The adversary designed to achieve a lower bound will use a constant t in essentially the same way, that is, it will force the algorithm to use at least t probes per row. Below, we sketch some of the rules according to which the adversary acts. (Number the fences in the same way as in the algorithm, that is, from left to right, where F_t is the rightmost one.)

Suppose that at some time F_0 exists. Then the algorithm has made at least t probes on each of the $|F_0|$ topmost rows (one probe per fence and row). Knowing this, the adversary can safely put a 0 immediately below F_0, which results in the exclusion of $|F_0|$ rows, each one of which has costed the algorithm at least t probes. This adversary rule thus puts a restriction on how many fences there can be.

We will also attribute probes to fences in a way that is almost the same as in the proof of Lemma 4.6. Upon creation and when not being moved a fence is attributed the number of probes made at it. When a fence F_{i+1} is moved to F_i, the probes attributed to the moved fence will be the sum of the probes attributed to F_{i+1} and F_i before the move plus the actual number of probes performed. Suppose now that a fence F_i is attributed at least $t|F_i|$ probes. Then, again, the adversary can put a 0 immediately below F_i, which results in the exclusion of $|F_i|$ rows, each one of which has costed the algorithm at least t probes. This second rule of the adversary says that the algorithm will not be allowed to move a fence to the left 'too many' times.

The third main rule of the adversary is concerned with how to handle probes on the first row, and is slightly more technical. Basically, what it says is that unless the probe made is too far to the left (relative to the actual width of the uncertainty part of the top row), return a 1; otherwise, return a 0.

All vertical probes not covered by the rules above are answered 1 by the adversary.

In order to understand how the above rules interact, first note that, intuitively, the goal of any fence algorithm can be thought of as either constructing a tall fence 'far' to the left or gaining rows 'cheaply.' In one way or another, all three rules guiding the adversary prevents the algorithm from advancing to the left too easily. The first rule makes it impossible to repeatedly erect new fences in order to move to the left; the second rule prevents the algorithm from repeatedly moving the leftmost fence to the left; and the third rule ensures that any new leftmost fence does not advance the position of the leftmost fence by too much.

Using the adversary described above, we believe that we can prove:

Conjecture 5.3 *The string searching problem requires*

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

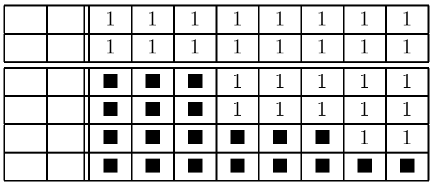
probes.

Recently, we have extended the adversary by two additional rules, very similar to the invariants maintained by our algorithm. Using the extended adversary we hope to prove a lower bound matching the upper bound presented in this paper:

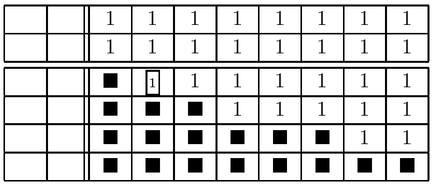
Conjecture 5.4 *The string searching problem requires*

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

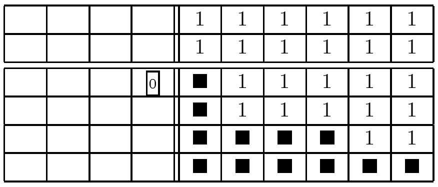
probes.



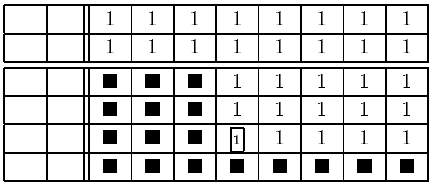
(a)



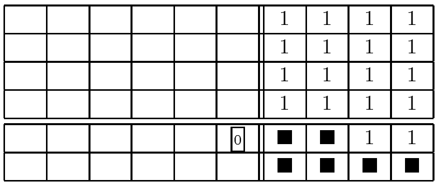
(b)



(c)



(d)



(e)

Figure 1: The one-sided problem.

References

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

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

[3] S. R. Kosaraju. On a multidimensional search problem. In *Proc. 11th Ann. ACM Symp. on Theory of Computing*, pp. 67–73, 1979.