PREPRINT. In Proc. 27th ACM Symposium on Theory of Computing, pages 417-426. ACM Press, 1995.

A tight lower bound for searching a sorted array

Arne Andersson* Johan Håstad† Ola Petersson*‡

Abstract

We show that given a k-character query string and an array of n strings arranged in alphabetical order, finding a matching string or report that no such string exists requires

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

character comparisons in the worst case, which is tight.

1 Introduction

The problem of searching a sorted set of strings is indeed fundamental. We assume that the strings are given in (lexicographically) sorted order in an array and that no extra information is available. In general, one cannot assume that two strings can be compared in constant time, but must consider the number of characters, or machine words, that need to be inspected.

Given its significance, the problem has received surprisingly little attention. A first non-trivial upper bound of $O(k \log n/\log k)$ was mentioned by Hirschberg [3]. Next, Kosaraju [4] gave an upper bound of $O(k\sqrt{\log n}+\log n)$, and recently, Andersson et al. [1] presented an upper bound of the same complexity as the lower bound claimed in the abstract. This subsumes both previous results.

Hirschberg [3] pointed out a trivial lower bound of $\Omega(k+\log n)$: if $k = 1$ any algorithm makes $\Omega(\log n)$ comparisons; moreover, it has to inspect all characters of the query string. The only non-trivial lower bound deals with constant factors: Kosaraju [4] showed a lower bound of roughly $\log n + \frac{1}{2}\sqrt{k \log n} = O(k + \log n)$ characters comparisons.

In this article, we show the following lower bound.

Theorem 1.1 *Given a k-character query string and an array of n strings arranged in alphabetical order, to find some matching string or report that no such string exists requires*

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

character comparisons in the worst case.

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

†Department of Computer Science, Royal Institute of Technology, 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, 351 95 Växjö, Sweden. The work was done while visiting Columbia University.

As this bound matches the recently shown upper bound [1], we close the problem, at least regarding its asymptotic complexity.

2 Preliminaries

For the purpose of proving a lower bound, we study the following, somewhat restricted, problem: The input is stored in a matrix of width n and height k, in which the columns are numbered from left to right and the rows from top to bottom. The strings contain only 0's and 1's, no string contains more than one 0, and they are stored in (lexicographically) sorted order in the columns of the matrix. The task is to determine the column of the leftmost string consisting of k 1's, and we charge an algorithm according to how many matrix entries it examines. Lemma 8.1 elaborates on the relation between this seemingly simpler problem and the original one.

To simplify matters further, we concentrate on a certain class of algorithms, called *fence algorithms.* The concept of a fence algorithm was defined in the paper which described the upper bound [1], and it was shown that for *any* algorithm there exists a corresponding fence algorithm of the same asymptotic complexity. We provide a brief sketch of this proof below. First, we recall what constitutes a fence algorithm.

A *fence* is a contiguous portion, starting at the top row, which is known to contain only 1's, of a column of the matrix. The height of a fence F is denoted by $|F|$ and defined as the number of rows spanned by the fence. The way we have stated the problem implies that all entries on the $|F|$ top rows to the right of F (inclusive) contain 1's.

To illustrate some important concepts, suppose an algorithm starts by probing the middle position of the top row. If it finds a 1 then it has erected a fence of height one, and it can conclude that all entries on the top row to the right of the fence are also 1's. Suppose the algorithm next probes and finds a 1 in the middle entry of the second row, i.e., the entry immediately below the just probed one. Suppose the next probe is made one quarter from the left end on the first row, and that it results in a 1. Our algorithm has then erected a second fence. The algorithm might then decide to probe at the first fence again, extending it to height three, etc. It is not difficult to see how the algorithm can create several fences in this way.

A fence algorithm is an algorithm which only makes two different kinds of probes: extension of an existing fence, or creation of a new leftmost fence.

If a fence algorithm encounters a 0 when attempting to extend a fence F_i, it can conclude that all columns to the left of the 0 (inclusive) need no longer be considered, and the algorithm can thus *reject* them. The algorithm can further conclude that all rows above the 0 (exclusive) can be omitted from consideration. This follows from that the $|F_i|$ top rows to the right of the fence F_i contain only 1's. The problem can thus be reduced by *excluding* these rows. In our terminology, the following four events occur: (1) the *rightmost* 0 moves to the right; (2) we get a new top row; (3) fences of height at most $|F_i|$ disappear completely (which makes sense because they no longer contribute any useful information of the whereabouts of the sought column); (4) the heights of all remaining fences are reduced accordingly.

Let F_1 and F_2 be two fences such that F_1 resides to the left of F_2 and $|F_1| < |F_2|$. If F_1 is extended to the same height as F_2 then F_2 ceases to exist. The justification for this is that the information that can be concluded about the matrix from F_2 is strictly less than what can be concluded from F_1. We say that F_1 and F_2 *merge*.

The fences are numbered in descending order from right to left, starting with fence F_t, where t is a parameter to be specified below. Thus, from right to left, at any point in time we have fences F_t, $F_{t-1}, \ldots$ When two fences F_i and F_{i+1} merge, F_{i+1} disappears, and all fences to its left are renumbered such that the indices of two neighboring fences differ by exactly one at any time. It follows that any fence is shorter than its right neighbor. For technical reasons we also have a virtual fence in column $n+1$, which is denoted by F_{t+1}, and which is of height k.

A fence algorithm also allows an adversary to make fence probes at any moment. These probes will cost nothing to the algorithm but have the same effect in terms of creating fences, excluding rows etc.

As already mentioned, in a previous paper [1], the following lemma was proven:

Lemma 2.1 *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.*

Sketch of proof. The lemma is proven by showing that, given an arbitrary algorithm A and an input matrix I, there exists a fence algorithm A_f and a matrix I' such that the cost of running A_f on I is at most twice as high as that of running A on I', and the outcome of A on I' is exactly the same as the outcome of A_f on I. A_f and I' are defined online as A runs. At any time, the two algorithms maintain the same set of fences. A_f performs essentially the same probes as A, though not always in the same order.

If A makes a 'fence probe' at entry p, then A_f probes entry p of I, and its outcome is copied to I'. If A makes a probe which does not extend a fence or create a new leftmost fence, there are two possibilities. First, A may probe an entry whose contents is known, or an entry in an excluded column. In this case, the probed entry in I' is set to 1, and no probe is made by A_f.

Second, A may probe an entry p somewhere in the unexplored area below the fences. p does not extend any fence, but might become part of a fence later on. In this case, p is set to 1 in I'. If entry p in I contains a 0, this is compensated for later on by placing a 0 in I'. This happens when A makes a 'connecting' probe at entry q in I' that—if answered by 1—makes p part of a 'matching prefix.' Then q is set to 0; we also reveal a 0 in the topmost unknown entry in p's column. In this way, the invariant that both algorithms maintain the same fences is preserved.

A_f does not execute the probe at p immediately, but postpones it until a fence in p's column, or in a column to its left, reaches the row immediately above p. Note that this happens when A makes a connecting probe. If the 'arriving' fence is in the same column as p, then A_f performs the probe. Otherwise, instead of probing p, A_f extends the arriving fence by probing on the same row as p. If it finds a 1, p must also contain a 1. Otherwise, the probed row becomes the top row, and A_f can execute the original probe as a fence probe. In the last case, two fence probes simulate the probe at p. □

We show:

Lemma 2.2 *Successful searches in the restricted searching problem require*

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

probes in the worst case.

In Theorem 1.1 we claim that this lower bound applies to a somewhat different problem, in that we include unsuccessful searches as well as only require a successful search to report *some* matching string. Observe that in the restricted problem, there is a trivial $\Theta(k)$-time query algorithm for the problem addressed in Theorem 1.1, which simply inspects the rightmost string. If it contains only 1's the search is successful; if it contains a 0 the search is unsuccessful. Thus, Theorem 1.1 does not follow immediately from Lemma 2.2. We return to this issue in Section 8, where we show that it does indeed follow from *our* lower bound proof.

3 Intuition

Let us try to give an intuitive explanation of some important parameters, used to prove the lower bound. Recall that fence algorithms maintain an ordered collection of fences, $\mathcal{F}$. Important properties are the *number* of fences in $\mathcal{F}$, the *distances* between fences, and their *heights*. Suppose there is a fence algorithm which makes at most tk probes, for some value of t. Below, we make three observations that relate the parameters of an algorithm to t.

We stress that we do not claim to *prove* anything in this section; we merely give the intuition behind the choice of parameters. The arguments given are sometimes imprecise and sloppy.

1. Assume that a new fence F_i is created by probing the *middle* entry of the unknown part of the top row, that is, one step of a binary search for locating the rightmost 0 is made. (This, significantly simplifying, assumption is only made in this section for the sake of intuition.) Suppose more such probes are made which all find 1's. In effect, F_i moves leftwards. An intuitive way to quantify the 'horizontal investment' in F_i is to count the number of binary search probes invested in it. Denote this quantity by Δ_i.

2. If x fences exist at the moment when a row is excluded, the algorithm has spent at least x probes on that row. If the algorithm want to ensure the cost per row to be at most t, the cardinality of $\mathcal{F}$ should not exceed t at any time. This suggests that the algorithm should spread the fences wisely.

3. Let us briefly describe a natural accounting scheme. When a fence is created or extended by one probe the

vertical investment in the fence increases by one, and when two fences merge, the new fence gets the sum of the vertical investments of the participating fences plus the number of actual probes performed to accomplish the merge. Then, the algorithm must ensure that the vertical investment in a fence during its lifetime must not be too large, unless it is tall enough. To see why, suppose more than $t|F|$ probes have been invested in fence F. Then, a failure to extend F results in that $|F|$ rows get excluded at a cost of *more* than t per row. Hence, a fence must be kept sufficiently tall so that a failure to extend it is not too costly for the algorithm.

We proceed by combining the above observations to derive some ideas on how the algorithm should chose its parameters.

The second observation states that we should have at most t fences. A natural way to spread them is at exponentially increasing distances, where the distance is measured according to the first observation. This means that distances of neighboring fences should increase by a factor of c, where

$$c^t = \log n \Rightarrow \log c = \frac{\log\log n}{t}, \qquad (1)$$

since the maximum distance between any two fences is at most $\log n$.

To apply the third observation, consider what happens when an extension of a fence F_i fails. The number of rows gained must compensate for the loss of both horizontal and vertical investments. The algorithm should thus attempt to keep F_i higher than some lower bound, which is a function of the total investment in F_i. Denote this function by T. To determine the function T we have a number of relations to consider.

Consider first the loss in horizontal investment. It must hold that

$$T(\Delta_i) \geq \Delta_i/t, \qquad (2)$$

since otherwise a failure to extend F_i would result in the exclusion of $|F_i| < \Delta_i/t$ rows, which would cost the algorithm more than t probes per row. Hence, the horizontal investment in F_i puts one restriction on T.

Next, the vertical investment in F_i should be at most th_i, since otherwise the loss of *this* investment cannot be compensated for by the number of rows excluded. The vertical investment in a fence depends heavily on how much it has been merged. In order to understand how it grows, note that when merging two fences F_{i-1} and F_i, the Δ-value for the resulting fence F_i (recall that renumbering occurs after the merge) becomes the sum of the previous Δ_{i-1} and Δ_i. Due to the distance ratio c described above, a merge increases the Δ-value by a factor $(1+1/c)$, that is,

$$\Delta_i' = \Delta_{i-1} + \Delta_i = (1+1/c)\Delta_i, \qquad (3)$$

where Δ_i' is the distance after the merge has taken place. Suppose that, prior to the merge, F_{i-1} and F_i are of heights $T(\Delta_{i-1})$ and $T(\Delta_i)$, respectively, and that $t\,T(\Delta_{i-1})$ and $t\,T(\Delta_i)$ probes, respectively, have been invested in them. Then, the vertical investment in the (new) fence F_i after the merge is

$$t[T(\Delta_{i-1}) + T(\Delta_i)] + T(\Delta_i') - T(\Delta_{i-1}).$$

If we disregard the last term (which turns out to be insignificant) and observe that T grows at least linearly, by Equation (2), this expression is at least $(t+1)[T(\Delta_{i-1}) + T(\Delta_i)]$. As the vertical investment in the new F_i should be at most $t\,T(\Delta_i')$, we get the relation:

$$(t+1)[T(\Delta_{i-1}) + T(\Delta_i)] \leq t\,T(\Delta_i').$$

Using Equation (3) and disregarding $T(\Delta_{i-1})$ gives:

$$(t+1)T(\Delta_i) \leq t\,T((1+1/c)\Delta_i),$$

which has a solution of the form $T(\Delta_i) \geq a \cdot \Delta_i^{c/t}$, for some constant a. Setting

$$T(\Delta_i) = \frac{\Delta_i^{1+c/t}}{t}$$

satisfies this requirement as well as Equation (2).

Finally, the fact that the tallest possible fence (which might have Δ-value of $\log n$) should span about all rows gives a relation to the the number of rows, k, namely

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

which implies $c \log c = \log(tk/\log n)$. Together with Equation (1) this gives the values of c and t.

Essentially, the algorithm of Andersson et al. [1], which achieves the optimal upper bound, can be derived from the above discussion by defining everything precisely and adjusting a few constants.

The lower bound is proven by means of an adversary for fence algorithms. This adversary keeps track of the investments made by an algorithm, and whenever the algorithm has not protected its investments by erecting tall enough fences, it reveals information that makes the algorithm lose its investment at too high cost. The adversary's actions basically forces the algorithm to behave as described above, or it will do worse.

One detail which makes the lower bound proof quite involved is that we need to take special care of probes on the first row since we cannot assume that the algorithm always makes its probes in the middle. Dealing with this is nontrivial. It is not intuitively clear that biased probes do not help. Interestingly, the algorithm that achieves the optimal upper bound does not make biased probes [1]. However, an algorithm using biased probes might achieve an improvement in constant factors.

4 Two proofs

For small k's, $k = O(\log n/\log\log n)$, and for large k's, $k = \Omega(2^{(\log n)^\epsilon})$, for any constant $\epsilon > 0$, the bound claimed in Lemma 2.2 reduces to the trivial lower bound of $\Omega(k+\log n)$. It thus suffices to provide a proof for intermediate values of k.

The proof is divided into two similar, but different proofs handling two ranges of k's. The proof for small k's takes care of $\log n/\log\log n < k \leq \log n$, and the proof for large k's assumes that $\log n < k < 2^{(\log n)^{1/4}}$. The two proofs share the same structure; however, the calculations are somewhat different. Due to lack of space, this extended abstract only contains a complete version of the proof for large k's, which is also the easier of the two. In Section 7 we outline the proof for large k's. Until then we can thus assume that $\log n < k < 2^{(\log n)^{1/4}}$.

5 The adversary

Before specifying the adversary, we introduce some parameters and notation. Define

$$\begin{aligned} c &= \log\left(\frac{k(\log\log n)^2}{\log n}\right) \\ t &= \frac{\log\log n}{10\log c} \\ B &= \max\{10c\log c, 1+\log k\}. \end{aligned}$$

Our aim is to prove an $\Omega(kt)$ lower bound on the number of probes required. This yields the desired lower bound when $k \geq \log n$. Observe that if $t = O(1)$ the aimed bound reduces to the trivial lower bound, and so we can assume that t is greater than some sufficiently large constant. The same applies to c, which is super-constant for $k \geq \log n$; and then also to B. Finally, note that the upper bound on k implies that $B = O(\sqrt{\log n})$.

For any fence F_i, h_i denotes the height of F_i, that is, $h_i = h(F_i) = |F_i|$. Let F_d be the leftmost fence. Define

$$m_{d-1} = m = \log(\text{column of } F_d - \text{column of the rightmost } 0).$$

This reflects the uncertainty on the top row, in that by making m 'binary search' probes on the row an algorithm will know its entire contents. For any other fence F_i, define

$$m_i = m(F_i) = \log(\text{column of } F_{i+1} - \text{column in which } F_i \text{ was created}),$$

which approximates the new value of m if F_i finds a 0. Finally, for any fence F_i, define

$$\Delta_i = \Delta(F_i) = \frac{m_i - m_{i-1}}{B}.$$

This quantifies the 'distance' between F_i and F_{i+1}.

Note that h_i, m_i, and Δ_i are attributes of fence F_i—not of index i—so if F_i changes index then its h-value, m-value, and Δ-value remain the same (unless their underlying parameters change). We use the shorter forms for the sake of brevity.

5.1 A game

Our adversary plays a game against a fence algorithm. Each probe made by the algorithm is answered by the adversary, which also sometimes reveals the contents of other entries at no cost to the algorithm.

The adversary will reveal information which is unfavorable to the algorithm. Suppose the algorithm has erected a fence F_i of height h_i and that it has performed $\Omega(h_i t)$ probes on the top h_i rows. Then the adversary *might* reveal a 0 immediately below fence F_i. We call this *putting a 0 on* F_i. This results in the exclusion of the columns to the left of F_i (inclusive) and the top h_i rows. The adversary does not always put a 0 on a fence once the above condition holds; however, whenever it does then the condition holds.

It is not obvious why this would be unfavorable to the algorithm. After all, it learns everything it needs to know about the top h_i rows, and thus need not invest any more probes on these rows. The underlying motivation for the adversary's behavior is that the distance from the leftmost fence to the rightmost 0 increases; a drawback for the algorithm.

For each excluded row, the algorithm has thus made $\Omega(t)$ probes. Since we aim to prove a lower bound of $\Omega(kt)$ it is natural to declare the adversary a winner of the game if it manages to exclude all rows. (Formally, excluding the kth row would require putting a 0 on row $k+1$. However, there is no need to explicitly place this last 0, just the fact that the adversary can allow itself to do so is enough to conclude that the algorithm has used $\Omega(kt)$ probes.)

On the other hand, if the adversary is not able to exclude the remaining rows when the algorithm has terminated the algorithm wins the game. (The adversary is thus allowed one more move after the algorithm has terminated.) The search can terminate in two ways: the algorithm successfully finds the leftmost column containing only ones, or it finds a 0 in the rightmost column, in which case the search was unsuccessful.

5.2 Accounting

In order to be able to carry out its strategy, the adversary keeps track of the probes performed by the algorithm by attributing them to the fences and to a horizontal (probe) counter, as follows:

- If a probe extends an existing fence F_i, attribute eight probes to F_i.
- Let F_i be the leftmost fence. If a probe is made on the top row, erecting F_{i-1}, then eight probes are attributed to F_{i-1}, three probes are attributed to F_i, and four probes are attributed to the horizontal counter.
- When F_i and F_{i-1} merge then the new F_i is attributed the sum of the attributions to the two old fences.
- Whenever a 0 is put on F_i, then delete
  - h_i attributed probes from each remaining fence;
  - $\sum_{j=d}^{i} \Delta_j$ attributed probes from the horizontal counter; and
  - all probes attributed to F_i and the fences to its left.

  Lemmas 6.6 and 6.7 below ensure that there will always be enough probes to delete.

Following these rules, each probe made by the algorithm yields at most 15 attributed probes. Hence, the total number of probes getting attributed is at most linear in the actual number of probes made. It is thus sufficient to derive a lower bound on the former quantity.

In the following, $A_i = A(F_i)$ and C denote the number of probes attributed to fence F_i and the horizontal counter, respectively.

5.3 The adversary's strategy

In order to understand how the adversary acts, first note that, intuitively, the goal of any fence algorithm can be thought of as constructing a fence 'far' to the left. Basically, there are three different ways for the algorithm to pursue this goal, and for each of those there is a counteracting adversary rule. Before presenting the precise rules we give some intuition.

First, when creating a new fence, the algorithm may try to place it far to the left of the leftmost existing fence. This is prevented by rule A, which simply answers 0 if a probe is made too far to the left on the top row. Second, it may try to advance leftwards by erecting many fences. This is handled by rule B, which puts an absolute restriction on the number of fences that an algorithm is allowed to maintain simultaneously. Third, to be allowed to erect a new fence further to the left, the algorithm might start by reducing the number of fences by merging two existing fences F_{i-1} and F_i. This strategy has two consequences: the horizontal distance from the new F_i (that results from the merge) to its right neighbor, F_{i+1}, increases; and, by the accounting scheme following a merge, the number of probes attributed to F_i increases. Adversary rule C ensures that fences are not too far apart, and rule D puts a restriction on how many probes can be attributed to a fence.

The first of the rules specifies how probes on the top row are answered.

Rule A: Let F_i be the leftmost fence, and suppose that the rightmost 0 is in column r, that is, F_i resides in column $r + 2^m$. A probe in column $r + a2^m$, where $0 < a < 1$, on the top row is answered as follows:

1. If $a \le 1/2^{B+1}$, a 0 is answered. The adversary also reveals a 1 in column $r + 2^{m-B}$.
2. If $a > 1/2^{B+1}$, a 1 is answered. If $a > 1/2^B$ the adversary also reveals a 1 in column $r + 2^{m-B}$.

Thus, *one* new fence F_{i-1} is *always* created between columns $r + 2^{m-B-1}$ and $r + 2^{m-B}$.

Any probe that attempts to extend an existing fence is answered by 1. However, after any probe made by the algorithm the adversary checks if any of the following rules apply. If it does, it is executed; otherwise, the algorithm is allowed to make another probe. If several rules are enabled, they are executed in the order in which they are given.

Rule B: Whenever F_0 exists, put a 0 on it.

Rule C: If $\Delta_i > h_i t$, put a 0 on F_i.

Rule D: If $A_i > h_i t$, put a 0 on F_i.

6 Analysis

In Section 6.1 we state and prove a number of lemmas which explain the immediate effect on the Δ-values caused by the various actions by the adversary and the algorithm. Then, in Section 6.2 we provide the main argument of the proof, and in Section 6.3 we prove the main lemma used in the main argument.

In the sequel, for any variable X, let X' denote the new value of the variable after some type of change has taken place.

6.1 Basic lemmas

The proofs of the lemmas below require no nontrivial observations and are purely algebraic. During the first reading the impatient reader might therefore want to skip them in order to reach the 'action' in the next subsection.

The first two lemma investigate how a probe on the top row affects the Δ-values:

Lemma 6.1 *Let F_i be the leftmost fence. Then, after a probe on the top row we have*

1. $m'_j = m_j$, *for any* $j \ge i$;
2. $m + \log(1 - 1/2^B) \le m'_{i-1} \le m + \log(1 - 1/2^{B+1})$;
3. $m - B - 1 \le m' \le m - B$.

Proof. Recall the definition of m_i. The first claim is obvious. Consider the second claim. $2^{m'_{i-1}}$ is the distance (number of columns) between the new fence and F_i. This is maximized (minimized) if F_{i-1} is erected as close to (far away from) the rightmost 0 as possible. Hence,

$$m'_{i-1} \ge \log\left(2^m - 2^{m-B}\right) = m + \log(1 - 1/2^B)$$

and

$$m'_{i-1} \le \log\left(2^m - 2^{m-B-1}\right) = m + \log(1 - 1/2^{B+1}).$$

$2^{m'}$ is the distance from the rightmost 0 to F_{i-1}, and so m' is maximized (minimized) if F_{i-1} is erected as far away from (close to) the rightmost 0 as possible. Hence,

$$m - B - 1 = \log 2^{m-B-1} \le m' \le \log 2^{m-B} = m - B.$$

□

The next lemma follows from plugging in the upper and lower bounds on m'_{i-1}, m'_i, and m', provided by the above lemma, in the definition of Δ_i.

Lemma 6.2 *Let F_i be the leftmost fence. Then, after a probe on the top row we have*

1. $\Delta_i \le \Delta'_i \le \Delta_i + 1/2^B$;
2. $1/2 \le \Delta'_{i-1} \le 2$.

Proof. By definition and Lemma 6.1, we have

$$\Delta'_i = \frac{m'_i - m'_{i-1}}{B} = \frac{m_i - m'_{i-1}}{B}.$$

The first claim then follows by replacing m'_{i-1} by the upper and lower bounds provided by the above lemma:

$$\begin{aligned}\Delta'_i &\le \frac{m_i - (m + \log(1 - 1/2^B))}{B} \\ &= \Delta_i - \frac{\log(1 - 1/2^B)}{B} \le \Delta_i + 1/2^B,\end{aligned}$$

for sufficiently large B, and

$$\begin{aligned}\Delta'_i &\ge \frac{m_i - (m + \log(1 - 1/2^{B+1}))}{B} \\ &= \Delta_i - \frac{\log(1 - 1/2^{B+1})}{B} \ge \Delta_i.\end{aligned}$$

Similarly, since $\Delta'_{i-1} = (m'_{i-1} - m')/B$, we have

$$\begin{aligned}\Delta'_{i-1} &\le \frac{m + \log(1 - 1/2^{B+1}) - (m - B - 1)}{B} \\ &= \frac{\log(2^{B+1} - 1)}{B} \le \frac{B+1}{B} \le 2,\end{aligned}$$

because $B \geq 1$; and

$$\begin{aligned}\Delta'_{i-1} &\geq \frac{m + \log(1 - 1/2^B) - (m - B)}{B} \\ &= \frac{\log(2^B - 1)}{B} \geq \frac{B-1}{B} \geq 1/2.\end{aligned}$$

□

We also show that a Δ-value never declines below its initial value:

Lemma 6.3 *For any fence F_i, $\Delta_i > 1/2$.*

Proof. The claim holds initially, by Lemma 6.2. As long as F_i exists, the only parameter in the definition of Δ_i that can change during the course of the search is m_{i-1}. However, its initial value is an absolute upper bound on its future value. The lemma follows. □

The next lemma shows that putting a 0 on a fence never increases the Δ-value of the new leftmost fence:

Lemma 6.4 *If a 0 is put on F_{i-1}, then $\Delta'_i \leq \Delta_i$.*

Proof. Note that $m'_{i-1} = m'$ and $m'_i = m_i$. If F_{i-1} has not merged with its left neighbor since it was created then $m' = m_{i-1}$, in which case Δ_i remains the same. Otherwise, F_{i-1} has merged to the left, in which case $m' > m_{i-1}$, and so Δ_i decreases (slightly). □

The next lemma shows how a merge affects the Δ-values.

Lemma 6.5 *If F_i and F_{i-1} merge then*

$$\Delta'_j = \begin{cases} \Delta_{j-1} & j \leq i-1, \\ \Delta_i + \Delta_{i-1} & j = i. \end{cases}$$

Proof. Recall that as a result of the merge, the former F_i disappears, and due to the renumbering, after the merge, fence F'_i resides in the same column as F_{i-1} did before the merge. Also, the indices of all fences to the left of F'_i are incremented by one. For the new F'_i, we have

$$\begin{aligned}\Delta'_i &= \frac{m'_i - m'_{i-1}}{B} = \frac{m_i - m_{i-2}}{B} \\ &= \frac{m_i - m_{i-1}}{B} + \frac{m_{i-1} - m_{i-2}}{B} \\ &= \Delta_i + \Delta_{i-1}.\end{aligned}$$

For any fence F'_j, $j \leq i-1$, to the left of the merge,

$$\Delta'_j = \frac{m'_j - m'_{j-1}}{B} = \frac{m_{j-1} - m_{j-2}}{B} = \Delta_{j-1}.$$

□

6.2 Main argument

We first prove that if the adversary wins, i.e., excludes all rows, then the algorithm has indeed made $\Omega(kt)$ probes (Lemma 6.8). We then show that the algorithm cannot win (Lemma 6.11). If combined, these two lemmas lead to Lemma 2.2.

Lemma 6.6 *For any fence F_i, $A_i \geq h_i$.*

Proof. This is easily proven by induction. Whenever h_i increases by one, A_i increases by eight; and whenever h_i decreases by one, A_i decreases by one. □

Lemma 6.7 $C \geq \sum_{j \leq t} \Delta_j$.

Proof. The proof is by induction on the probes. Initially the claim holds trivially. When a new fence F_{i-1} is erected, the sum increases by

$$(\Delta'_i - \Delta_i) + \Delta'_{i-1} \leq 1/2^B + 2 \leq 3,$$

by Lemma 6.2, and so the four probes added to C pay for the increase.

If F_i and F_{i-1} merge then, by Lemma 6.5, the sum is not affected.

When a 0 is put on F_i then, by Lemma 6.4, the sum decreases by

$$\sum_{j \leq i} \Delta_j + (\Delta_{i+1} - \Delta'_{i+1}) \geq \sum_{j \leq i} \Delta_j,$$

while C decreases by exactly $\sum_{j \leq i} \Delta_j$. □

Lemma 6.8 *If the adversary wins then $\Omega(kt)$ probes have been made.*

Proof. When the adversary puts a 0 on fence F_i, h_i rows get excluded, and by the accounting scheme, a number of attributed probes get deleted. We show that this quantity is at least $h_i t$. We distinguish three cases depending on which one of the rules B, C, and D triggered at F_i:

B. In this case $h_i = 1$ and one attributed probe per fence gets deleted, giving a total of $t + 1$ deleted probes. Lemma 6.6 guarantees that no fence will ever run out of attributed probes, but can always pay one.

C. The horizontal counter decreases by $\sum_{j=d}^{i} \Delta_j \geq \Delta_i$, which is at least $h_i t$, by rule C. Lemma 6.7 guarantees that the horizontal counter can be charged.

D. In this case $A_i > h_i t$, so the (at least) $h_i t$ attributed probes deleted from F_i suffice.

Hence, for each excluded row, at least t attributed probes, and thus at least $t/15 = \Omega(t)$ actual probes, get deleted. If all rows get excluded, the algorithm must therefore have spent $\Omega(kt)$ probes altogether. □

To accomplish our second goal, that the algorithm never wins, requires two additional lemmas.

Lemma 6.9 *No fence is ever erected within distance $n/2$ from fence F_{t+1}.*

Proof. Consider the location of fence F_t during the course of the game. According to rule A, the first time F_t is created it resides to the left of column $n/2^B$, and as long as it exists the (at least) $n(1 - 1/2^B)$ columns to its right cannot be excluded. The second time F_t is created it resides to the left of column $n(1 - 1/2^B)/2^B$, and as long as it exists the (at least) $n(1 - 1/2^B)^2$ columns to its right remain. Each creation of F_t is preceded by the exclusion of at least one row, and at most k rows can be excluded. Therefore, the number of columns that remain to the right of F_t after the search is at least

$$n\left(1-\frac{1}{2^B}\right)^k \geq n\left(1-\frac{1}{2^{1+\log k}}\right)^k = n\left(1-\frac{1}{2k}\right)^k \geq \frac{n}{2},$$

for $k \geq 1$. □

The next lemma says that if F_i is 'far' from F_{i+1}, it must have many probes attributed to it:

Lemma 6.10 *For any fence F_i,*

$$A_i \geq h_i + \frac{\Delta_i}{t}\left(\frac{\Delta_i}{(2c+1)^i}\right)^{c/t}.$$

Lemma 6.10 is the core of the entire proof, and its proof is postponed till the next subsection.

Lemma 6.11 *The algorithm never wins.*

Proof. Recall that the adversary never places a 0 to the right of F_t. Consequently, by Lemma 6.9, a search cannot be unsuccessful. The other possibility for the algorithm to win is by erecting a fence which reaches the bottom row in the leftmost non-rejected column. Note that this fence has no other fence to its right since it spans all rows, and so it must be F_t. We show that then Δ_t is large, and therefore, by Lemma 6.10, F_t must have many probes attributed to it. In fact, we show that A_t is so large that rule D will apply, so the adversary can put a 0 on F_t and exclude all rows and win the game.

At the end of the search, $m_{t-1} = m = 0$, so $\Delta_t \geq \log(n/2)/B$, by Lemma 6.9. Since $B = O(\sqrt{\log n})$, this is $\Omega(\sqrt{\log n})$.

Now,

$$\begin{aligned}
\frac{\Delta_t}{(2c+1)^t} &= \frac{\Delta_t}{2^{t\log(2c+1)}} \\
&\geq 2^{\log\Delta_t - 2t\log c} \\
\text{[by def. of } t] &= 2^{\log\Delta_t - (\log\log n)/5} \\
\text{[for suff. large } n] &\geq 2^{(\log\log n)/4},
\end{aligned}$$

where the last inequality holds since $\Delta_t = \Omega(\sqrt{\log n})$.

By Lemma 6.10,

$$\begin{aligned}
A_t &\geq \frac{\log(n/2)}{Bt}\left(\frac{\Delta_t}{(2c+1)^t}\right)^{c/t} \\
&\geq \frac{\log n}{2Bt}\, 2^{(\log\log n)/4\cdot c/t} \\
\text{[by def. of } t] &= \frac{\log n}{2Bt}\, 2^{(\log\log n)/4\cdot c\cdot 10\log c/\log\log n} \\
\text{[by def. of } B \text{ and } t] &\geq \frac{2^{(5c\log c)/2}\log n\, 10\log c}{2(10c\log c + 1 + \log k)\log\log n} \\
\text{[for suff. large } c] &\geq \frac{2^{3c+1}\log n}{c\log k\,\log\log n} \\
&\geq \frac{2^{c+1}\log n}{\log\log n}\,\frac{2^c}{\log k} \\
\text{[by def. of } c] &= \frac{2k(\log\log n)^2\log n}{\log n\,\log\log n}\,\frac{k(\log\log n)^2}{\log n\,\log k} \\
\text{[since } k > \log n] &\geq 2k\log\log n \\
\text{[by def. of } t] &\geq 2kt.
\end{aligned}$$

As $h_t \leq k$ it follows that $A_t \geq 2h_t k$, so rule D triggers at F_t, and the adversary puts a 0 on F_t and wins the game. □

6.3 Proof of Lemma 6.10

Coupled with rule D, Lemma 6.10 immediately leads to the following lemma, which is very useful in the inductive proof of Lemma 6.10:

Lemma 6.12 *For any fence F_i, $h_i \geq \frac{\Delta_i}{t^2}\left(\frac{\Delta_i}{(2c+1)^i}\right)^{c/t}$.*

Proof of Lemma 6.10. For brevity, let $K = 2c+1$. We may assume that $K \geq 6$.

The proof is by induction on the probes. Initially, when F_i is created $C_i = 8$, $h_i = 1$, and $\Delta_i \leq 2$ (by Lemma 6.2). Hence

$$h_i + \frac{\Delta_i}{t}\left(\frac{\Delta_i}{K^i}\right)^{c/t} \leq 1 + \frac{2}{t}\left(\frac{2}{K^i}\right)^{c/t} \leq 2,$$

because $K^i \geq 2$ and $t \geq 2$.

Inductively assume that the lemma, and consequently also Lemma 6.12, holds at any fence F_i prior to a certain probe made by the algorithm. We show show that then it holds at fence F_i' after the probe. We have five cases to consider depending on how the next probe affects the values of i, h_i, A_i, and Δ_i. In each case we need to show that A_i increases by at least as much as the claimed lower bound. For each case, we provide a brief sketch of how it is handled; details are given after the enumeration.

1. F_i is extended but does not merge with F_{i+1}. This case is straightforward.
2. F_j is extended and merges with F_{j+1}, for some $j \geq i$. Then the new F_i is the former F_{i-1}, and so $A_i' = A_{i-1}$. As the claimed bound is decreasing on i, the lemma follows by induction.
3. The adversary puts a 0 on fence F_j, for some $j \leq i-1$. According to Lemma 6.4, Δ_i does not increase in this case, and both A_i and h_i decrease by $|F_j|$. Hence, the bound decreases by at least as much as does A_i.
4. F_{i-1} appears. In this case, Δ_i increases by at most $1/2^B$, by Lemma 6.2. The claim then follows from the mean value theorem and the fact that the derivative of our bound with respect to Δ_i is at most 2^B.
5. F_{i-1} is extended and merges with F_i. The new F_i gets attributed $A_i' = A_i + A_{i-1}$ probes while $\Delta_i' = \Delta_i + \Delta_{i-1}$, by Lemma 6.5. We need to prove that the additional A_{i-1} probes compensates for the increase in Δ_i. This is accomplished by applying the inductive assumption to the two contributing fences, and requires a little bit of elementary calculus.

Case 1 is easy: A_i increases by eight, and since Δ_i does not change, the bound increases by one.

In case 2, F_j and all fences to its left change index by one, so the new F_i is the former F_{i-1}. By induction,

$$A_i' = A_{i-1} \quad\geq\quad h_{i-1} + \frac{\Delta_{i-1}}{t}\left(\frac{\Delta_{i-1}}{K^{i-1}}\right)^{c/t}$$

$$= h_i' + \frac{\Delta_i'}{t}\left(\frac{\Delta_i'}{K^{i-1}}\right)^{c/t}$$
$$\geq h_i' + \frac{\Delta_i'}{t}\left(\frac{\Delta_i'}{K^{i}}\right)^{c/t},$$

as desired.

Consider now case 3. If F_{i-1} disappears then, according to Lemma 6.4, Δ_i *decreases*; otherwise it remains unchanged. Thus, in either case, $\Delta_i' \leq \Delta_i$. Exclusion of h rows decreases both A_i and h_i by h. Hence, by induction

$$A_i' = A_i - h \geq h_i + \frac{\Delta_i}{t}\left(\frac{\Delta_i}{K^i}\right)^{c/t} - h \geq h_i' + \frac{\Delta_i'}{t}\left(\frac{\Delta_i'}{K^i}\right)^{c/t}.$$

In case 4, F_i gets attributed three probes, that is, $A_i' = A_i + 3$. We show that the right side of the claimed bound increases by less than three. When F_i gets a new left neighbor, Δ_i increases; however, the increase is at most $1/2^B$, by Lemma 6.2.

We show that the derivative of our bound with respect to Δ_i in the point $\Delta_i + 1/2^B$ is bounded by 2^B. Since the bound is a convex function on Δ_i, it follows from the mean value theorem that it increases by at most one.

First note that $\Delta_i + 1/2^B \leq 2\Delta_i$, by Lemma 6.3. The derivative of the bound with respect to Δ_i in the point $2\Delta_i$ is

$$\begin{aligned}
\left(\frac{2\Delta_i}{K^i}\right)^{c/t}\left(1+\frac{c}{t}\right)\frac{1}{t} &\leq \left(\frac{2\Delta_i}{K^i}\right)^{c/t}\left(1+\frac{c}{t}\right) \\
&\leq \left(\frac{2\Delta_i e}{K^i}\right)^{c/t} \\
\text{[since } K^i \geq 6\text{]} &\leq \Delta_i^{c/t} \\
&\leq (\log n)^{c/t} \\
\text{[by def. of } t\text{]} &= 2^{10c\log c} \\
\text{[by def. of } B\text{]} &\leq 2^B.
\end{aligned}$$

In case 5 the new F_i gets attributed $A_i' = A_i + A_{i-1}$ probes. By induction, all we have to prove is

$$h_i + \frac{\Delta_i}{t}\left(\frac{\Delta_i}{K^i}\right)^{c/t} + h_{i-1} + \frac{\Delta_{i-1}}{t}\left(\frac{\Delta_{i-1}}{K^{i-1}}\right)^{c/t} - \left[h_i' + \frac{\Delta_i'}{t}\left(\frac{\Delta_i'}{K^i}\right)^{c/t}\right] \geq 0,$$

or equivalently, since $\Delta_i' = \Delta_i + \Delta_{i-1}$ and $h_i' = h_i = h_{i-1}$,

$$h_i + \frac{\Delta_i}{t}\left(\frac{\Delta_i}{K^i}\right)^{c/t} + \frac{\Delta_{i-1}}{t}\left(\frac{\Delta_{i-1}}{K^{i-1}}\right)^{c/t} - \frac{\Delta_i + \Delta_{i-1}}{t}\left(\frac{\Delta_i + \Delta_{i-1}}{K^i}\right)^{c/t} \geq 0.$$

By induction, the lemma held at F_i before the merge, and so, by Lemma 6.12,

$$h_i \geq \frac{\Delta_i}{t^2}\left(\frac{\Delta_i}{K^i}\right)^{c/t}.$$

Replacing h_i and putting $\Delta_{i-1} = a\Delta_i$, we thus need to establish

$$\frac{\Delta_i}{t^2}\left(\frac{\Delta_i}{K^i}\right)^{c/t} + \frac{\Delta_i}{t}\left(\frac{\Delta_i}{K^i}\right)^{c/t} + \frac{a\Delta_i}{t}\left(\frac{a\Delta_i}{K^{i-1}}\right)^{c/t} - \frac{(1+a)\Delta_i}{t}\left(\frac{(1+a)\Delta_i}{K^i}\right)^{c/t} \geq 0.$$

Canceling the common factor $(\Delta_i/t)(\Delta_i/K^i)^{c/t}$ yields

$$\frac{1}{t} + 1 + a^{1+c/t}K^{c/t} - (1+a)^{1+c/t} \geq 0. \tag{4}$$

Differentiating the left side with respect to a we obtain

$$\left(1+\frac{c}{t}\right)\left((aK)^{c/t} - (1+a)^{c/t}\right),$$

and thus we have a minimum at $a_0 = 1/(K-1) = 1/2c$. We show that for $a = a_0$ the negative contribution in inequality (4) is smaller than the positive:

$$\begin{aligned}
(1+a_0)^{1+c/t} &= (1+a_0)^{c/t} + a_0(1+a_0)^{c/t} \\
&\leq e^{a_0 c/t} + a_0\left(\frac{K}{K-1}\right)^{c/t} \\
&= e^{1/2t} + K^{c/t}a_0^{1+c/t} \\
&\leq 1 + \frac{1}{t} + K^{c/t}a_0^{1+c/t},
\end{aligned}$$

since we can assume $t \geq 1$, and we are done. □

7 Proof for small k's

The structure of the proof of Lemma 2.2 for small k's is exactly the same as that for large k's, and most of the lemmas do indeed mirror the previous ones; however, there are some subtle differences. In particular, the difference between Lemmas 6.10 and 7.11 should be noted.

In the next subsection we describe the required changes in parameters and adversary strategy. We then proceed in the same way as in Section 6, first stating some basic lemmas and then providing the main argument.

Recall that we can assume that $\log n/\log\log n \leq k \leq \log n$.

7.1 Changes in parameters and adversary

Let

$$c = \max\left\{D, \log\left(\frac{k\log\log n}{\log n}\right)\right\},$$

where D is some large constant. We can thus assume that c is larger than some sufficiently large constant. t is defined as above. Since $k \leq \log n$ we have $c = O(\log\log\log n)$, and so $t = \Omega(\log\log n/\log\log\log\log n)$. Hence, also t can be assumed to be larger than some sufficiently large constant.

For any fence F_i, define

$$d_i = |F_{i+1}|, \text{ when } F_i \text{ is created.}$$

We stress that, when F_i is created, $d_i = h_{i+1}$; while later on we can have either $d_i > h_{i+1}$ or $d_i \leq h_{i+1}$. The definition of Δ_i is altered:

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

The only change needed in the described adversary's is in rule A, which is modified slightly: Replace 2^{B+1} by $8h_i = 8d_{i-1}$.

The accounting scheme is also changed slightly in that when two fences F_{i-1} and F_i merge, the new fence does not get all the attributed probes, but $\log d_{i-1}$ probes are transferred to the horizontal counter. (This stems from the new definition of Δ_i.)

7.2 Basic lemmas

The first five lemmas mirror Lemmas 6.1–6.5.

Lemma 7.1 *Let F_i be the leftmost fence. Then, after a probe on the top row we have*

1. $m'_j = m_j$, *for any* $j \geq i$;
2. $m + \log(1 - 1/4d_{i-1}) \leq m'_{i-1} \leq m + \log(1 - 1/8d_{i-1})$;
3. $m + \log(1/8d_{i-1}) \leq m' \leq m + \log(1/4d_{i-1})$.

Proof. Replace 2^{B+1} by $8d_{i-1}$ in the proof of Lemma 6.1. □

Lemma 7.2 *Let F_i be the leftmost fence. Then, after a probe on the top row we have*

1. $\Delta_i \leq \Delta'_i \leq \Delta_i + 1/d_{i-1}$;
2. $1 < \Delta'_{i-1} < 3$.

Proof. Plug in the bounds on m'_{i-1}, m'_i, and m' from the preceding lemma into the definition of Δ_j. □

Lemma 7.3 *For any fence $F_i \in \mathcal{F}$, $\Delta_i > 1$.*

Proof. Identical to the proof of Lemma 6.3. □

Lemma 7.4 *If a 0 is put on F_{i-1}, then $\Delta'_i \leq \Delta_i$.*

Proof. Identical to the proof of Lemma 6.4. □

When two fences merge, the situation is a little bit more complicated than before due to the revised definition of Δ_i:

Lemma 7.5 *If F_i and F_{i-1} merge then*

$$\Delta'_j = \begin{cases} \Delta_{j-1} & j \leq i-1, \\ \Delta_i + \Delta_{i-1} + \log d_{i-1} & j = i. \end{cases}$$

Proof. Almost identical to the proof of Lemma 6.5. □

When two fences merge we thus have to transfer some probes to the horizontal counter. For subsequent use we need to bound this quantity in terms of the increase incurred by the merge. Let $H_i = H(F_i)$ be the total number of times that F_i has been extended (during its lifetime).

Lemma 7.6 *When F_{i-1} and F_i merge then $H_{i-1} \geq d_{i-1}$.*

Proof. Omitted. □

7.3 Main argument

We proceed along the same lines as in Section 6.2

Lemma 7.7 *For any fence $F_i \in \mathcal{F}$, $A_i \geq h_i + 7H_i$.*

Sketch of proof. Similar to the proof of Lemma 6.6. The main difference is that, when two fences merge, we need to transfer some probes to the horizontal counter. This quantity is bounded from above using Lemma 7.6. □

Lemma 7.8 $C \geq \sum_{j \leq t} \Delta_j$.

Sketch of proof. Again, the only difference from the corresponding lemma for large k's (Lemma 6.7) is the transfer of probes to the horizontal counter, following a merge. □

Lemma 7.9 *If the adversary wins then $\Omega(kt)$ probes have been made.*

Proof. Identical to the proof of Lemma 6.8. □

Lemma 7.10 *No fence is ever erected within distance n/k from fence F_{t+1}.*

Proof. Consider the location of fence F_t during the course of the game. According to rule A, the first time F_t is created it resides to the left of column $n/4k$, and as long as it exists the (at least) $n(1 - 1/4k)$ columns to its right cannot be rejected. In general, the ith time F_t is created, there are at most $k + 1 - i$ rows left. Hence, the second time F_t is created, the number of columns to its right is at least

$$n\left(1 - \frac{1}{4k}\right)\left(1 - \frac{1}{4(k-1)}\right) = n\frac{4k-1}{4k} \cdot \frac{4(k-1)-1}{4(k-1)}.$$

Suppose F_t is created ℓ times altogether. Then the number of columns remaining when the algorithm is finished is at least

$$\begin{aligned} & n\frac{4k-1}{4k} \cdot \frac{4(k-1)-1}{4(k-1)} \cdot \frac{4(k-2)-1}{4(k-2)} \cdots \frac{4(k-(\ell-1))-1}{4(k-(\ell-1))} \\ \geq \; & n\frac{4(k-1)}{4k} \cdot \frac{4(k-2)}{4(k-1)} \cdot \frac{4(k-3)}{4(k-2)} \cdots \frac{4(k-\ell)}{4(k-(\ell-1))} \\ = \; & n\frac{4(k-\ell)}{4k} \\ \geq \; & \frac{n}{k}. \end{aligned}$$

□

Lemma 7.11 *For any fence $F_i \in \mathcal{F}$,*

$$A_i \geq h_i + H_i + \frac{\Delta_i}{t}\left(\frac{\Delta_i}{(2c+1)^i}\right)^{c/t} \log\left(\frac{\Delta_i}{(2c+1)^i} + 1\right).$$

As in the proof for large k's in Section 6.2, this lemma is the core of the argument, and its proof is omitted in this extended abstract. The proof has the same structure as that of Lemma 6.10; however, it is much more technically involved.

Lemma 7.12 *The algorithm never wins.*

Proof. By the same token as in the proof of Lemma 6.11, it suffices to consider the number of probes attributed to fence F_t at the end of the game.

If the search terminates successfully, we have $m_{t-1} = m = 0$, and so by Lemma 7.10,

$$\Delta_t \geq \log \binom{n}{k} - \log k = \log \binom{n}{k^2} \geq \frac{\log n}{2}$$

for sufficiently large n, because $k = O(\log n)$.

Now, the same argument as in the proof of Lemma 6.11 shows that

$$\left(\frac{\Delta_t}{(2c+1)^t}\right)^{c/t} \log \left(\frac{\Delta_t}{(2c+1)^t} + 1\right) \geq \frac{2^{c \log c} \log \log n}{4}.$$

Hence, by Lemma 7.11,

$$\begin{aligned} A_t &\geq \frac{\log n}{2t} \frac{2^{c \log c} \log \log n}{4} \\ \text{[by def. of } t] &= \frac{\log n}{2 \log \log n} \frac{10 \log c}{} \frac{2^{c \log c} \log \log n}{4} \\ &\geq 2^{c \log c} \log n \\ \text{[for suff. large } c] &\geq 2^{c+1} \log n \\ \text{[by def. of } c \text{ and } t] &\geq 2kt, \end{aligned}$$

Hence, rule D triggers at F_t, so the adversary will eliminate it and win the game. □

8 Extending the lower bound

Suppose we are only interested in finding *some* matching string, and that, if no matching string exists, then we do not care what the answer to the query is. Recall the trivial $\Theta(k)$-time query algorithm in Section 2. To neutralize this simplistic strategy, a slight modification of the restricted problem is required.

We have not found an immediate reduction from successful searches in the restricted problem to that in Theorem 1.1, but we have to modify the original proof of Lemma 2.2 slightly. Thus, this proof relies on the specified adversaries; the lemma would not necessarily hold if we used different adversaries.

Lemma 8.1 *The lower bound in Lemma 2.2 applies to the problem in Theorem 1.1.*

Proof. We create a new problem of n strings of length $k+1$ by appending a 0 to the leftmost string of 1's and a 1 to all other strings. We then search for the string of k 1's followed by a 0. This string is unique, and moreover it coincides with the leftmost string of k 1's in the restricted problem. It follows that any lower bound for successful searches in the restricted problem applies to successful searches in the original problem as well. Furthermore, as the outcome of a search, that is, whether it is going to be successful or not, is not determined until the last probe is made, it follows that the same lower bound applies to unsuccessful searches.

Unfortunately, the above modification gives rise to a subtle complication. Suppose an algorithm probes the last row. If a 1 is found the corresponding string can be excluded as a candidate. Thus, it might make perfect sense for an algorithm to make probes that are not allowed by the definition of fence algorithms. Note that according to Lemma 2.1, this is *not* the case in the restricted problem. In the spirit of Lemma 2.1, we show how to modify our adversaries such that for any non-fence probe there is a fence probe which is at least as profitable. Consequently, we can still assume fence algorithms, and so the lower bound in Lemma 2.2 applies to the modified problem as well.

The adversary answers any probe on the last row by 1. The probed entry is ignored by the adversary until the algorithm has erected a fence which spans all rows immediately to its right. (Note that this fence has to be F_t.) When this occurs, the adversary reveals 1's in the entire column above the probe. In effect, F_t gets moved one step to the left. If the algorithm had probed also the next entry on the last row, F_t gets moved two steps to the left, and so on.

The key observation is that instead of moving F_t one step to the left, the adversary can move the rightmost 0 one step to the *right*. This follows since this is equivalent of moving *all* fences one step to the left, and the adversary is free to give the help of also moving the other fences.

Now, if an algorithm probes the entry immediately to the right of the rightmost 0 on the top row, either one of our two original adversaries will answer 0. Hence, instead of probing the last row, the algorithm is always better off probing the entry immediately to the right of the rightmost 0 on the top row. The lemma follows. □

9 Comments

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

It should be noted that we make no other restrictions in the computational model; the algorithm is allowed to use extra memory during the search, create hash tables etc. Our proof only relies on the fact that the content of an entry can be determined in only two ways: either by the contents of neighboring entries or by an explicit probe at that entry.

Acknowledgements

We would like to thank Torben Hagerup for several helpful comments and suggestions.

References

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

[2] 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.

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

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