Optimal pattern matching in LZW compressed strings

Paweł Gawrychowski

Institute of Computer Science, University of Wrocław

February 1, 2011

Motivation?

We produce more and more information. This motivates developing efficient compression methods. But are we really interested in just storing the data? NO! We would like to process it efficiently on demand.
Is it really necessary to uncompress the data in order to process it? Depend what ”to process” means. Here we focus on the simplest yet still useful queries concerning compressed text:

Compressed pattern matching

Given a compressed representation of a text t and a pattern p, does p occur in t? If yes, where is the leftmost occurrence?

Compressed pattern matching is a fairly well-studied problem. Let $|t| = N$, $|p| = m$, and n be the compressed size of t. Depending on the exact compression method used we have the following results:

LZW	$\mathcal{O}(n + m^2)$	Amir, Benson, and Farach SODA 1994
	$\mathcal{O}(n \log m)$	
	$\mathcal{O}(n + m^{1+\epsilon})$	Kosaraju FSTTCS 1995
LZ	$\mathcal{O}(n \log^2 \frac{N}{n})$	Farach and Thorup STOC 95

In this paper we focus on LZW compression, and get the following result:

This paper

Compressed pattern matching can be solved in optimal linear time for LZW compressed texts.

If the alphabet is of non-constant (yet polynomial in both n and m) size, to get this linear time bound we need the word RAM model.

What Lempel-Ziv-Welsch compression really is?
Text is encoded as a sequence of **blocks** $b_1 b_2 \ldots b_n$. Each block is either a single letter, or a previously defined block concatenated with a single letter.

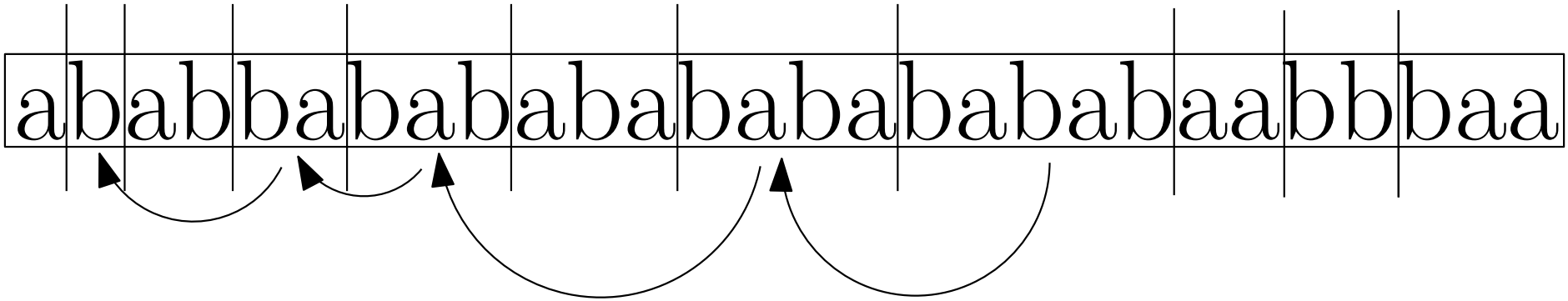
ababbababababababababaabbbaa

$n \in \Omega(\sqrt{N})$ but with very quick compression/decompression possible!

What Lempel-Ziv-Welsch compression really is?
Text is encoded as a sequence of **blocks** $b_1 b_2 \ldots b_n$. Each block is either a single letter, or a previously defined block concatenated with a single letter.

$n \in \Omega(\sqrt{N})$ but with very quick compression/decompression possible!

What Lempel-Ziv-Welsch compression really is?
Text is encoded as a sequence of **blocks** $b_1 b_2 \ldots b_n$. Each block is either a single letter, or a previously defined block concatenated with a single letter.



$n \in \Omega(\sqrt{N})$ but with very quick compression/decompression possible!

LZW parse is not very convenient to work with. The first step of the algorithm is to convert into to a more uniform form.

Pattern matching in a sequence of snippets

Given a string s and a collection of its substrings $s[i_1..j_1] \ldots s[i_n..j_n]$, does s occur in their concatenation?

If the alphabet is of constant size, the conversion is rather simple to perform in linear time. If it is not constant (but still polynomial in n and m), achieving such complexity requires a few tricks: we need to store a description of the suffix automaton built for s which allows constant time navigation yet requires just linear space to store.

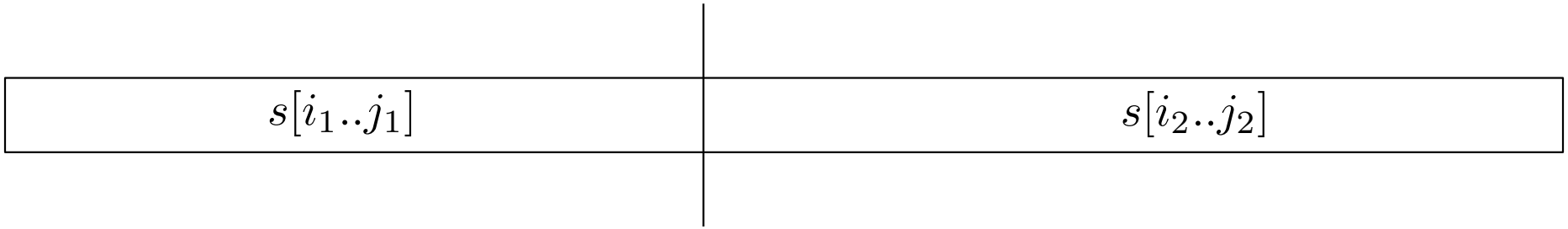
Given a sequence of snippets, simulate the KMP algorithm on the corresponding text. Instead of processing it letter-by-letter, try to process all letters from a single snippet at once.

Algorithm 1 NAIVE-PATTERN-MATCHING$(s_1, s_2, \ldots, s_n)$

1: $\ell \leftarrow 0$
2: **for** $k = 1, 2, 3, \ldots, n$ **do**
3: check if s occurs inside $s[1..\ell]s[i_k..j_k]$
4: $\ell \leftarrow$ longest prefix of s ending $s[1..\ell]s[i_k..j_k]$
5: **end for**

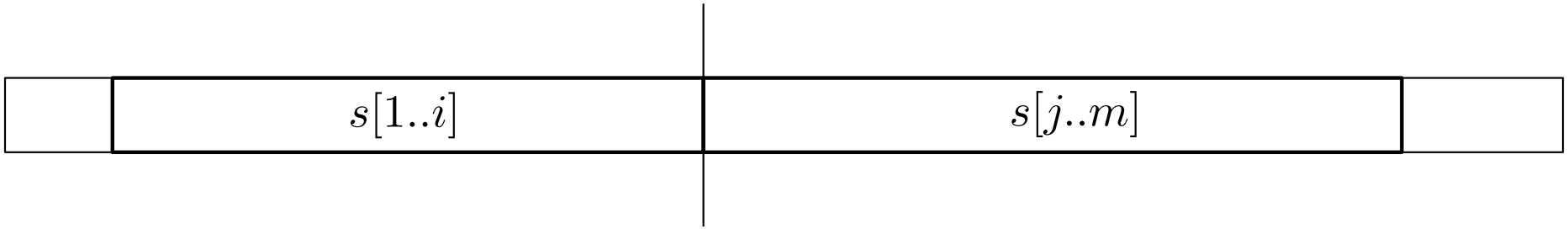
Lemma 3.1, also known as take the bigger half lemma

Given two snippets we can detect an occurence of the pattern in their concatenation in constant time.



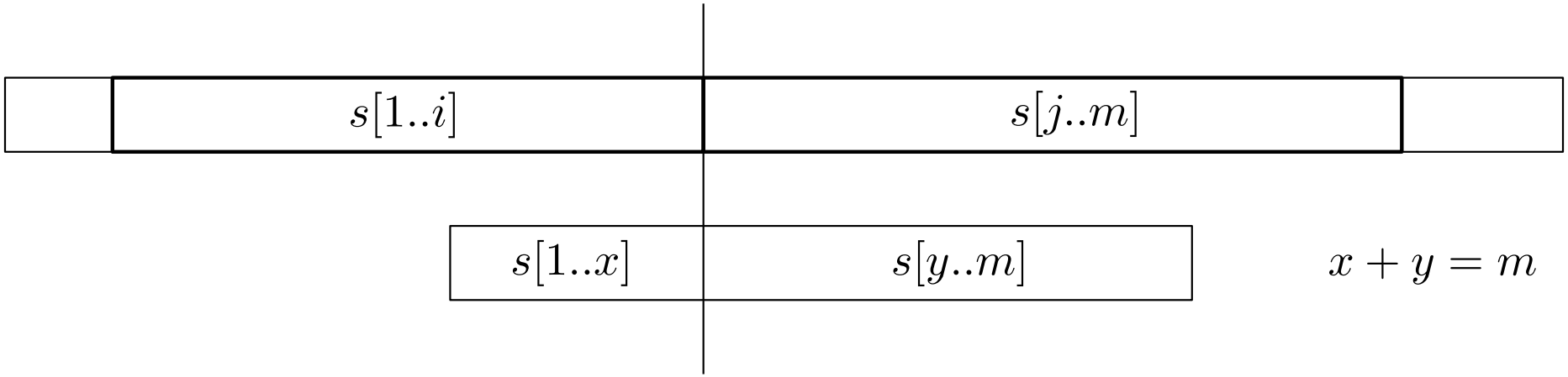
Lemma 3.1, also known as take the bigger half lemma

Given two snippets we can detect an occurence of the pattern in their concatenation in constant time.



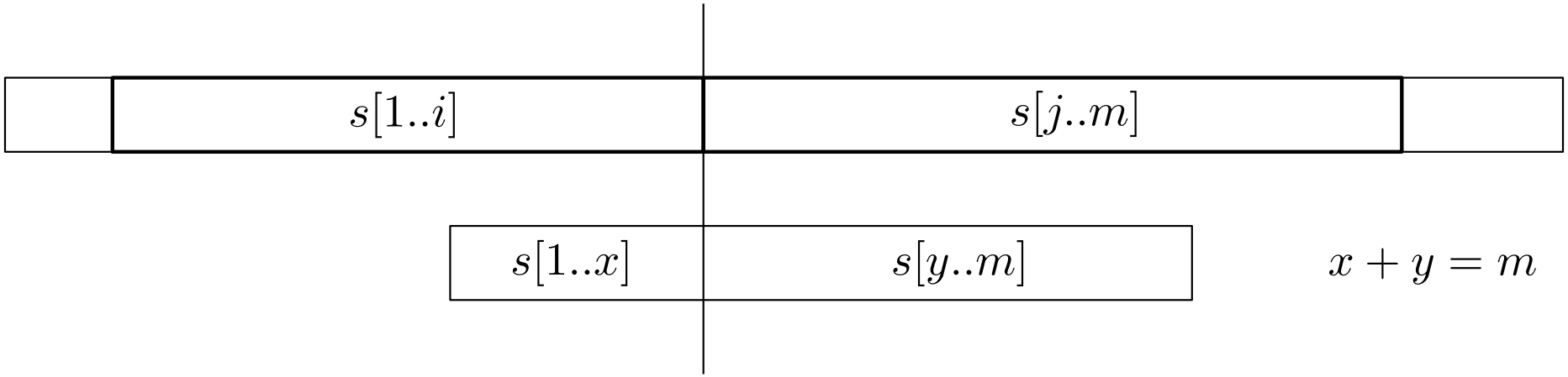
Lemma 3.1, also known as take the bigger half lemma

Given two snippets we can detect an occurence of the pattern in their concatenation in constant time.



Lemma 3.1, also known as take the bigger half lemma

Given two snippets we can detect an occurence of the pattern in their concatenation in constant time.



b is a border of a word w iff $w[1..b] = w[|w| - b + 1..|w|]$

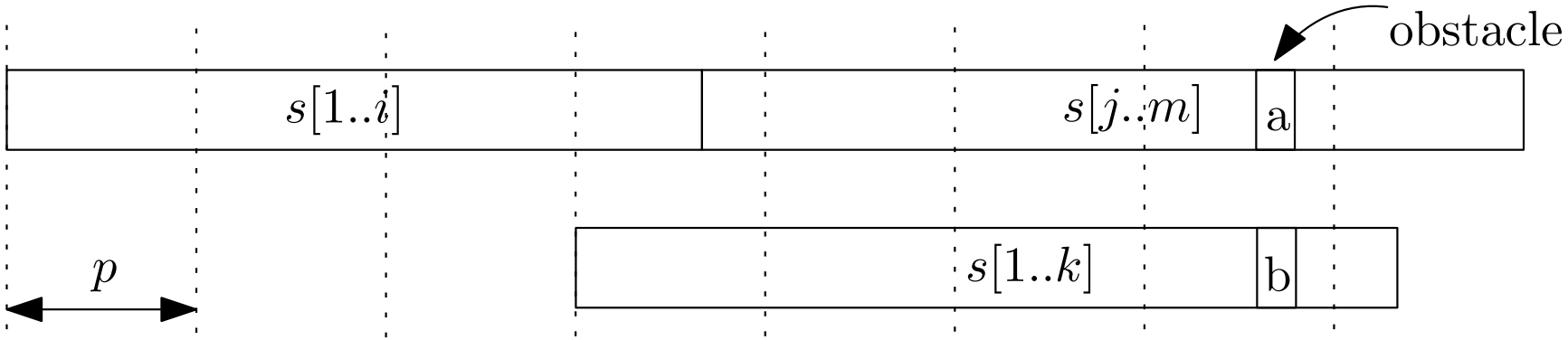
We are looking for $x \in \text{border}(s[1..i])$ and $y \in \text{border}(s[j..m])$ such that $x + y = m$.

border(w) potentially has a lot of completely different elements. But if we consider

$$\text{border}(w) \cap \{\left\lceil \frac{|w|}{2} \right\rceil, \ldots, |w| - 1, |w|\}$$

its elements create one arithmetic progression. More precisely, if p is the period of w, they are of the form $|w| - \alpha p$. We call such borders **long**.
Now either x or y is a long border. Both possibilities are symmetric so it is enough to consider the former.

Let p be the period of $s[1..i]$. Compute how far this period extends in the whole s and call the corresponding prefix $s[1..k]$. Take its rightmost shift of the form αp starting in $s[1..i]$ and compute the first mismatch.



It turns out that the position of this obstacle gives a simple arithmetic condition which allows us to eliminate all but one possible shift, which can be then verified in $\mathcal{O}(1)$ time using a few LCA queries.

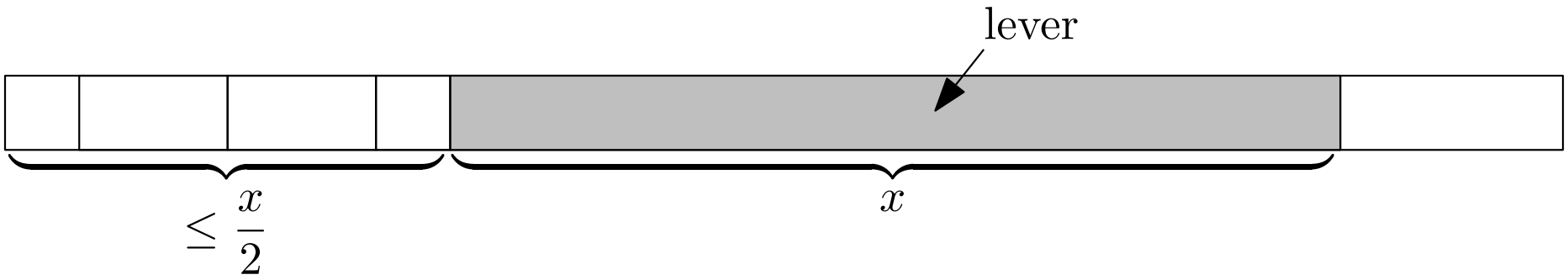
A similar approach can be used to show:

(an extension of) Lemma 3.2

Given two snippets we can find the longest suffix of their concatenation which is a prefix of the pattern in $\mathcal{O}(\log m)$ time.

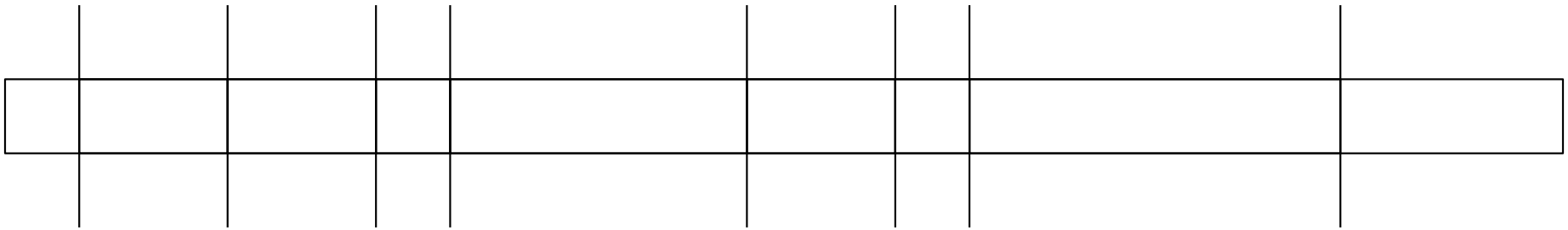
(this requires storing the corresponding vertex in the suffix tree for each snippet) which results in $\mathcal{O}(n \log m)$ total complexity. Too slow to improve the already known bounds.

How to accelerate the computation? Try to identify the ”bad case” and deal with it separately.
We use the concept of **levers**. A snippet is a lever if it occurs not very far to the right and is relatively long.



Turns out that thanks to such lever we can process all snippets on its left more efficiently. But what if there is no lever?
Switch to the previous method.

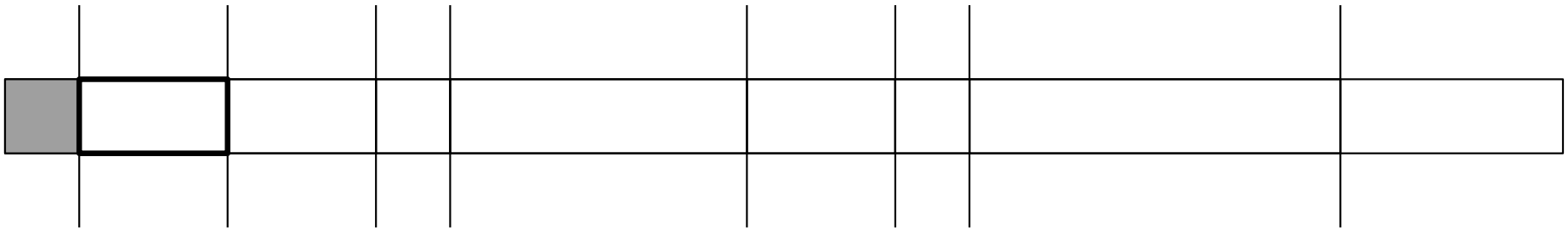
The intuition is that either there exists a lever, or we will create one after a few steps of the slow algorithm. More formally, with any sequence of snippets $s_1, s_2, \ldots, s_n$ we associate its potential $\Phi(|s_1|, |s_2|, \ldots, |s_n|)$. This potential does not depend on the exact structure of the words, just on their lengths.



Each snippet marks a segment on its left. Then we mark whole suffix of length m (because of some technical reasons).

$$\Phi(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} 2 + \log \frac{x_i}{y_i}$$

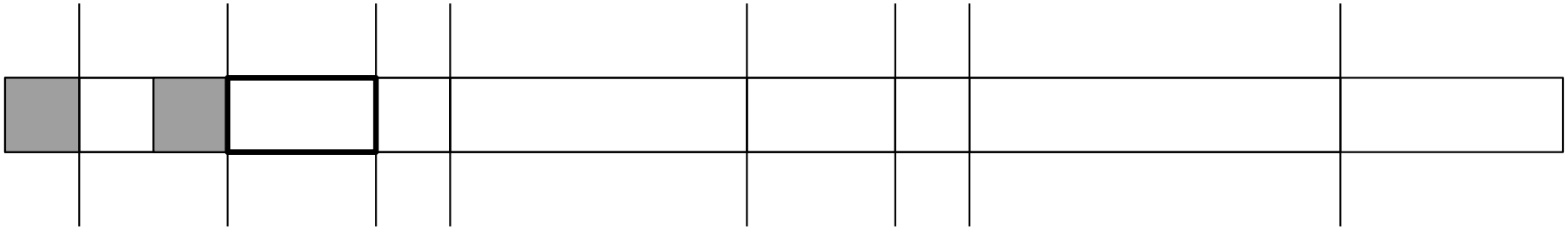
The intuition is that either there exists a lever, or we will create one after a few steps of the slow algorithm. More formally, with any sequence of snippets $s_1, s_2, \ldots, s_n$ we associate its potential $\Phi(|s_1|, |s_2|, \ldots, |s_n|)$. This potential does not depend on the exact structure of the words, just on their lengths.



Each snippet marks a segment on its left. Then we mark whole suffix of length m (because of some technical reasons).

$$\Phi(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} 2 + \log \frac{x_i}{y_i}$$

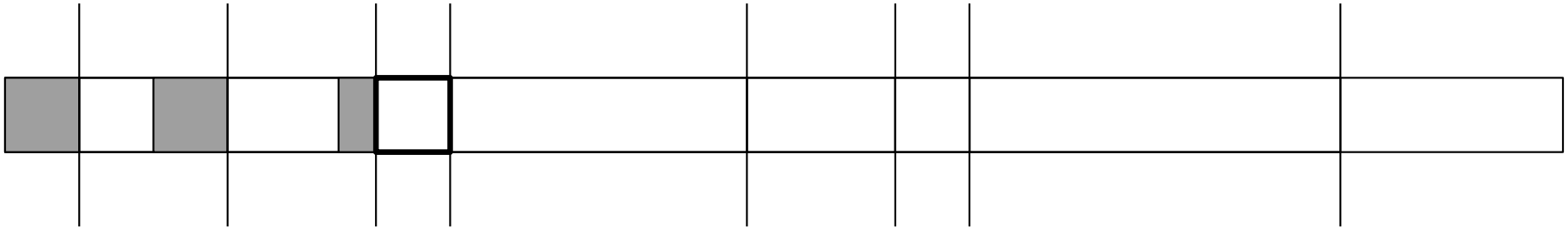
The intuition is that either there exists a lever, or we will create one after a few steps of the slow algorithm. More formally, with any sequence of snippets $s_1, s_2, \ldots, s_n$ we associate its potential $\Phi(|s_1|, |s_2|, \ldots, |s_n|)$. This potential does not depend on the exact structure of the words, just on their lengths.



Each snippet marks a segment on its left. Then we mark whole suffix of length m (because of some technical reasons).

$$\Phi(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} 2 + \log \frac{x_i}{y_i}$$

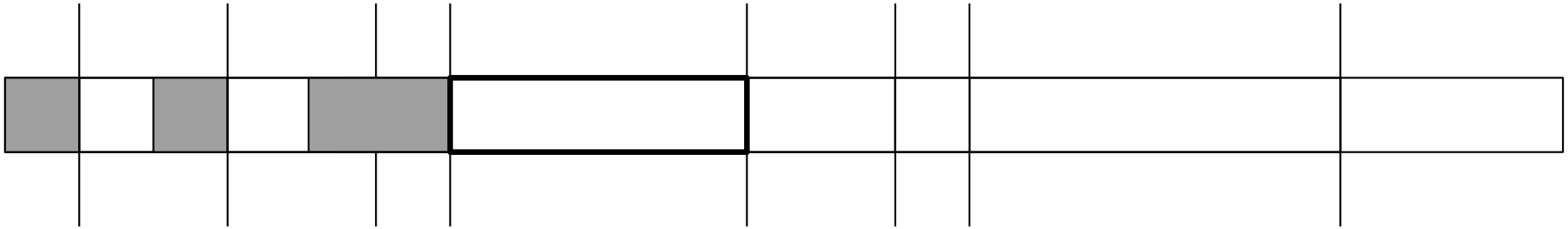
The intuition is that either there exists a lever, or we will create one after a few steps of the slow algorithm. More formally, with any sequence of snippets $s_1, s_2, \ldots, s_n$ we associate its potential $\Phi(|s_1|, |s_2|, \ldots, |s_n|)$. This potential does not depend on the exact structure of the words, just on their lengths.



Each snippet marks a segment on its left. Then we mark whole suffix of length m (because of some technical reasons).

$$\Phi(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} 2 + \log \frac{x_i}{y_i}$$

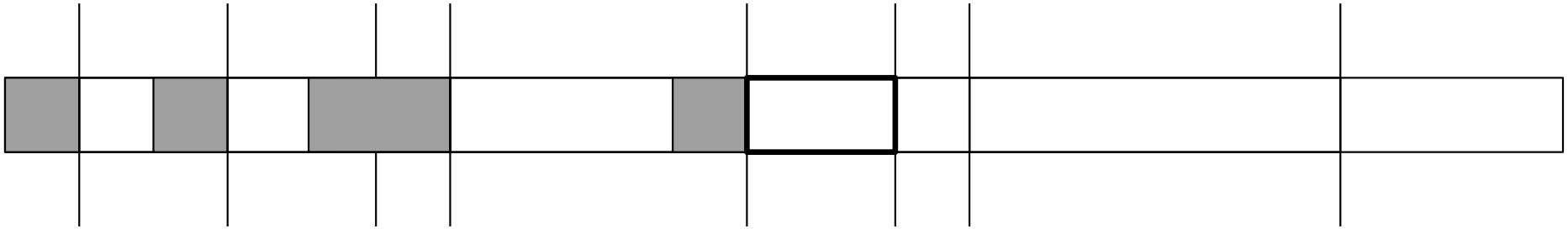
The intuition is that either there exists a lever, or we will create one after a few steps of the slow algorithm. More formally, with any sequence of snippets $s_1, s_2, \ldots, s_n$ we associate its potential $\Phi(|s_1|, |s_2|, \ldots, |s_n|)$. This potential does not depend on the exact structure of the words, just on their lengths.



Each snippet marks a segment on its left. Then we mark whole suffix of length m (because of some technical reasons).

$$\Phi(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} 2 + \log \frac{x_i}{y_i}$$

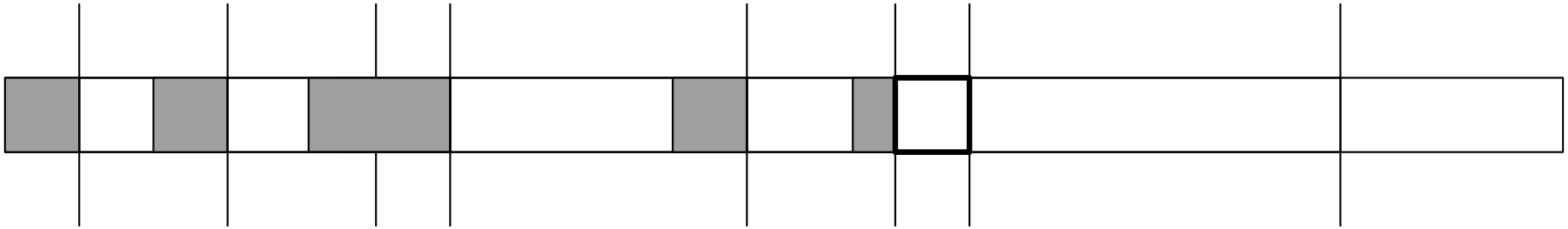
The intuition is that either there exists a lever, or we will create one after a few steps of the slow algorithm. More formally, with any sequence of snippets $s_1, s_2, \ldots, s_n$ we associate its potential $\Phi(|s_1|, |s_2|, \ldots, |s_n|)$. This potential does not depend on the exact structure of the words, just on their lengths.



Each snippet marks a segment on its left. Then we mark whole suffix of length m (because of some technical reasons).

$$\Phi(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} 2 + \log \frac{x_i}{y_i}$$

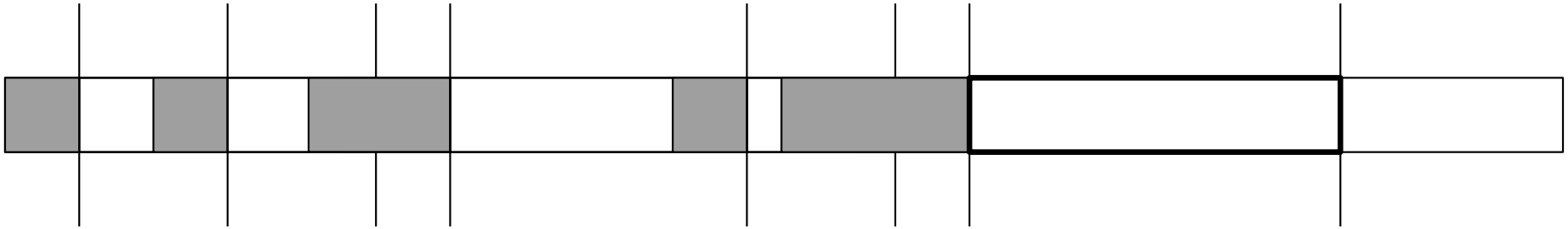
The intuition is that either there exists a lever, or we will create one after a few steps of the slow algorithm. More formally, with any sequence of snippets $s_1, s_2, \ldots, s_n$ we associate its potential $\Phi(|s_1|, |s_2|, \ldots, |s_n|)$. This potential does not depend on the exact structure of the words, just on their lengths.



Each snippet marks a segment on its left. Then we mark whole suffix of length m (because of some technical reasons).

$$\Phi(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} 2 + \log \frac{x_i}{y_i}$$

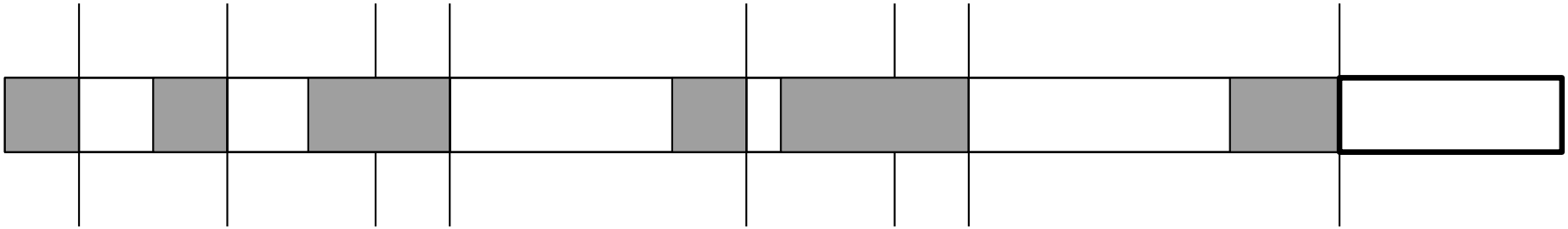
The intuition is that either there exists a lever, or we will create one after a few steps of the slow algorithm. More formally, with any sequence of snippets $s_1, s_2, \ldots, s_n$ we associate its potential $\Phi(|s_1|, |s_2|, \ldots, |s_n|)$. This potential does not depend on the exact structure of the words, just on their lengths.



Each snippet marks a segment on its left. Then we mark whole suffix of length m (because of some technical reasons).

$$\Phi(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} 2 + \log \frac{x_i}{y_i}$$

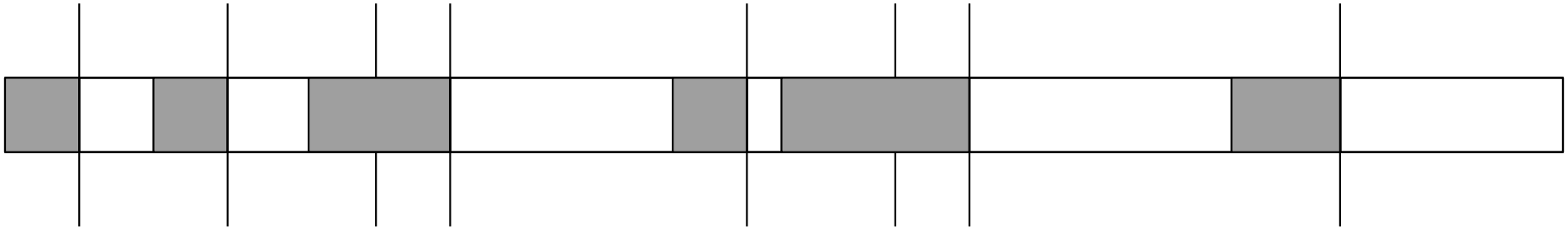
The intuition is that either there exists a lever, or we will create one after a few steps of the slow algorithm. More formally, with any sequence of snippets $s_1, s_2, \ldots, s_n$ we associate its potential $\Phi(|s_1|, |s_2|, \ldots, |s_n|)$. This potential does not depend on the exact structure of the words, just on their lengths.



Each snippet marks a segment on its left. Then we mark whole suffix of length m (because of some technical reasons).

$$\Phi(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} 2 + \log \frac{x_i}{y_i}$$

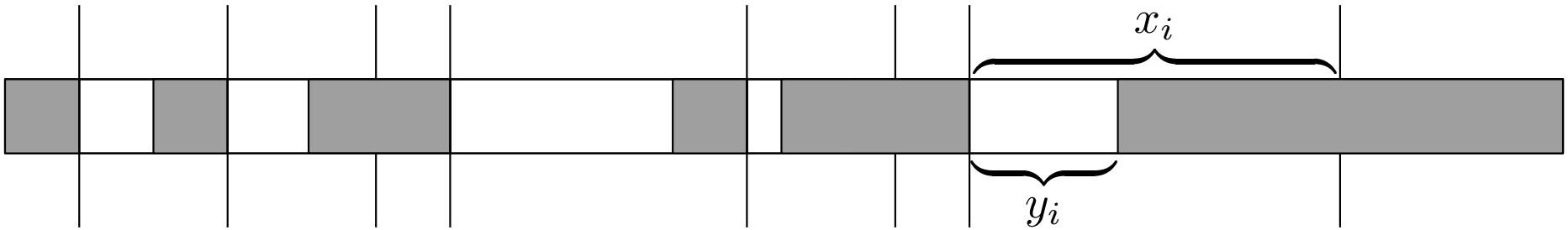
The intuition is that either there exists a lever, or we will create one after a few steps of the slow algorithm. More formally, with any sequence of snippets $s_1, s_2, \ldots, s_n$ we associate its potential $\Phi(|s_1|, |s_2|, \ldots, |s_n|)$. This potential does not depend on the exact structure of the words, just on their lengths.



Each snippet marks a segment on its left. Then we mark whole suffix of length m (because of some technical reasons).

$$\Phi(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} 2 + \log \frac{x_i}{y_i}$$

The intuition is that either there exists a lever, or we will create one after a few steps of the slow algorithm. More formally, with any sequence of snippets $s_1, s_2, \ldots, s_n$ we associate its potential $\Phi(|s_1|, |s_2|, \ldots, |s_n|)$. This potential does not depend on the exact structure of the words, just on their lengths.



Each snippet marks a segment on its left. Then we mark whole suffix of length m (because of some technical reasons).

$$\Phi(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} 2 + \log \frac{x_i}{y_i}$$

We bound Φ using a sequence of lemmata, and then show that the running time cannot exceed Φ.

$$\Phi(x_1, x_2, \ldots, x_k) \leq 7k$$

$$\Phi(x_1, x_2, \ldots, x_k) \geq 1 + \Phi(x_1 + x_2, \ldots, x_k).$$

$$\Phi(x_1, x_2, \ldots x_k) \geq \Phi(x_1', x_2, \ldots, x_k) \text{ if } x_1 \geq x_1'.$$

Hence:

$$\text{running time } \leq \Phi(x_1, x_2, \ldots, x_n) \leq \mathcal{O}(n)$$

which gives the claimed result.

Thank you for your attention!