Palindrome Recognition in the Streaming Model

P. Berenbrink, F. Ergun, F. Mallmann-Tren, E. Sadeqi-Azer

Palindrome

A string which reads the same forwards and backwards.

Palindrome

A string which reads the same forwards and backwards.

- Kayak
- Level

Palindrome

A string which reads the same forwards and backwards.

- Kayak
- Level
- Stressed? No tips? Spit on desserts.

(tip your waiter!)

Goals/Model

- Find long(est) palindrome in a given sequence
- Ideally, with errors. Currently, exact match, approximate length.
- Streaming model

Properties of Palindromes

P is a palindrome (eg AABBACCDDCCABBAA)

-- P consists of a string and its reverse on two sides of a midpoint.

$P = P_1 P_1^R$ AABBACCD DCCABBAA

-- P contains nested, smaller palindromes inside it with the same midpoint.

AABBACCDDCCABBAA

Finding Palindromes

Find palindromes in given text:

A A B A C A D D A C A B D A B A C

Palindromes

Find palindromes in given text:

A A B A C A D D A C A B D A B A C

What about these?

Find long palindromes. (longest?)

Outline

- Preliminaries, observations
- An additive-error approximation
- An exact algorithm
- A multiplicative error approximation
- A quick look at a lower bound

Observations and Definitions

P[m] denotes the maximal length palindrome centered at index m; its length is l(m).

The first half of the palindrome is the inverse of the second half.

Odd Length Palindromes

What about palindromes such as

A B C D C B A?

Where is the midpoint?

Simple fix: double all letters.

A A B B C C D D C C B B A A

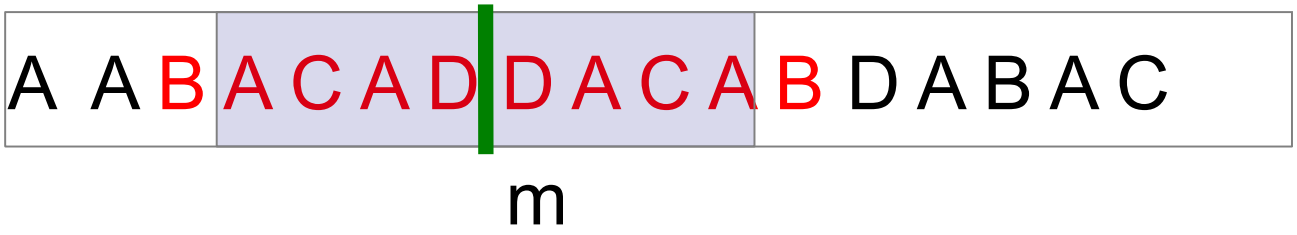
Length is doubled, palindromes are doubled.

Complexity not changed.

Observations and Definitions

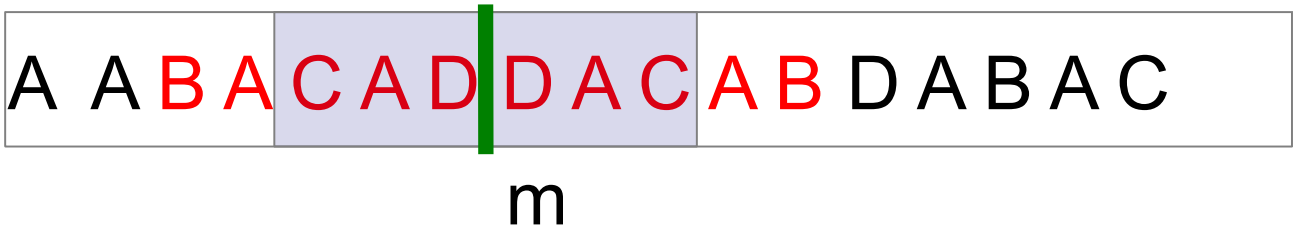
Palindromes show a “nested” structure.

Observations and Definitions



Palindromes show a “nested” structure.

Observations and Definitions



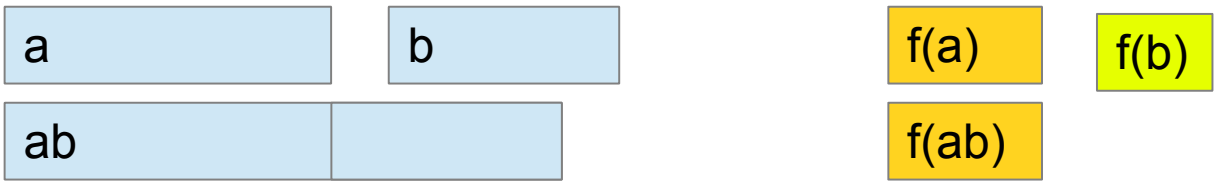
Palindromes show a “nested” structure.

Our goal is to find the maximal palindromes.

Compromise: find close to maximal

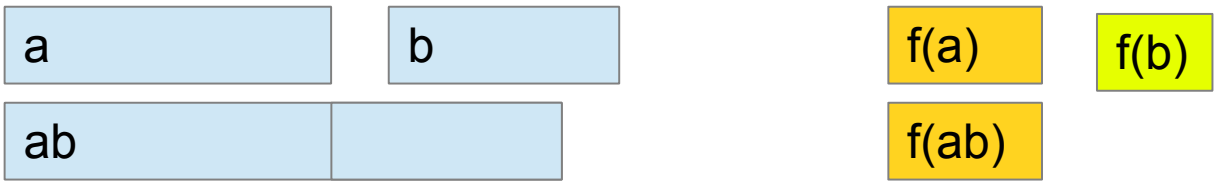
Rabin-Karp Fingerprints

- Commonly used in streaming string algorithms
- Can be combined/separated efficiently



Rabin-Karp Fingerprints

- Commonly used in streaming string algorithms
- Can be combined/separated efficiently



- Trivial to keep running fingerprints of substrings and their reversals

Computation of KR-Fingerprints

S is a string of length k.

Given prime p $\{n^4, n^4\}$ and a random

r $\{1,\ldots,p\}$,

$$\Phi_{r,p}(S) = \sum_{i=1..k} \left(S[i] \cdot r^i \right) \bmod r$$

Manipulating KR-Fingerprints

S = a b c d e f S' = b c d e f g

Given Φ(S), how do we get Φ(S')?

Outline

- Preliminaries, observations
- An additive-error approximation
- An exact algorithm
- A multiplicative error approximation
- A quick look at a lower bound

First Algorithm

Given S of length n:

- Finds all palindromes in S.
- Lengths of these palindromes are approximated to an additive $\varepsilon\sqrt{n}$ factor.
- Space usage is $O(\sqrt{n}/\varepsilon)$.

How?

Let X' denote the reverse of a string X.

Any palindrome P is of the form XX'.

Guess the midpoint, compare fingerprints of substrings before the midpoint to those after.

Short Palindromes, Long Palindromes

We run a sliding window of length $\sqrt{n}$ along S.

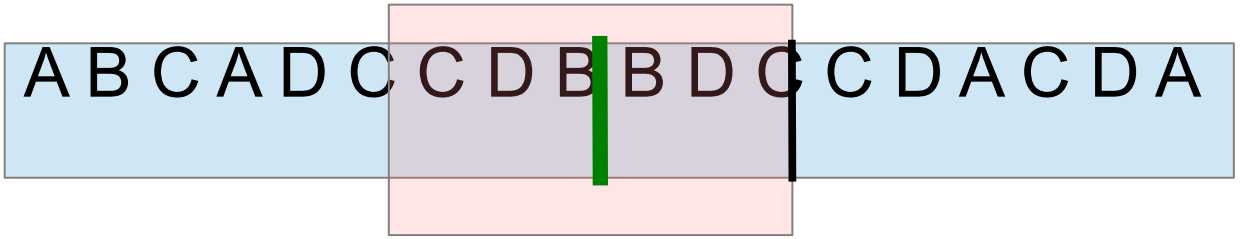
A short palindrome is length at most $\sqrt{n}$.

Easily found since it fits within the window.

A long palindrome must contain a short palindrome, so we can generate "candidates."

Long Palindrome Candidates

Window size = 6

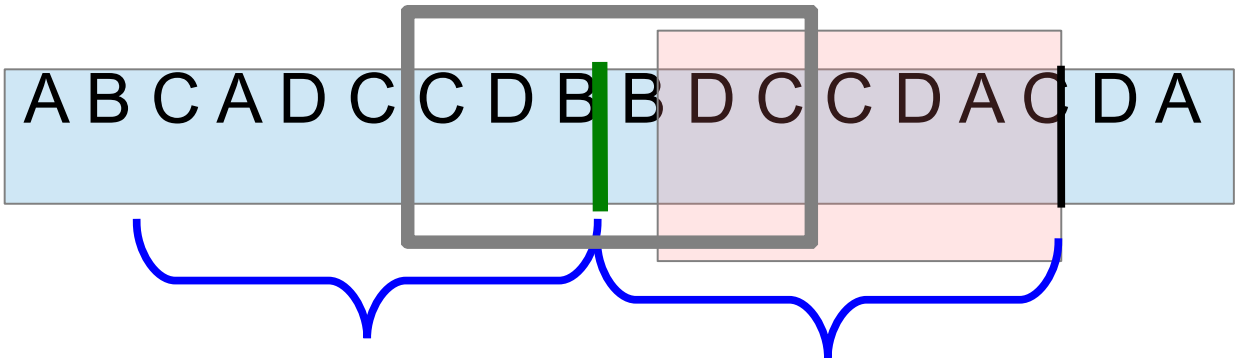


Palindrome detected in window.

Size = 6, thus potential to be long.

Growing A Palindrome

4 characters later...



If we could compare the contents of the braces, we would note that there is a long palindrome.

But we cannot keep track of every substring.

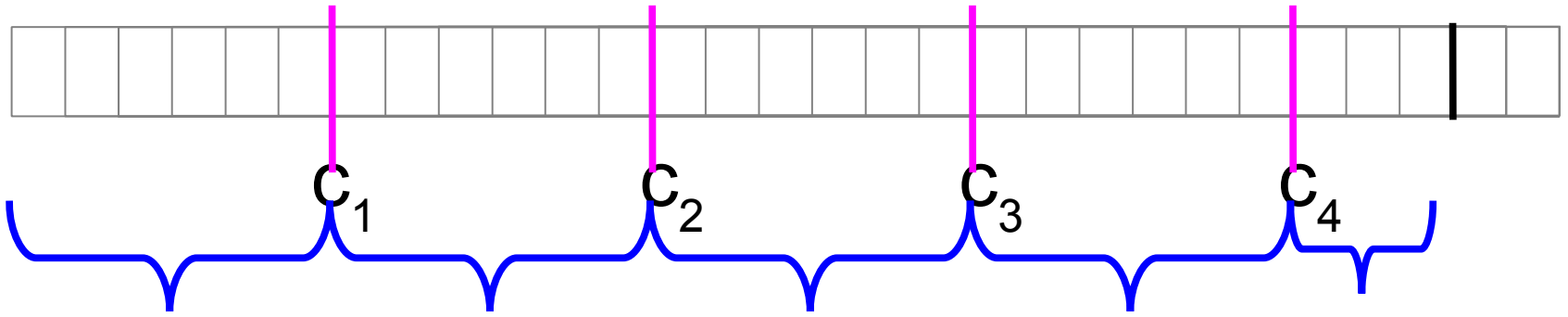
Keeping Track

Insert a checkpoint every $\varepsilon\sqrt{n}$ locations.

Remember the substring between every adjacent pair of checkpoints.

Altogether, $O(\sqrt{n}/\varepsilon)$ space.

Can reconstruct any substring of S with some loss



Keeping Track

Insert a checkpoint every $\varepsilon\sqrt{n}$ locations.

Remember the substring between every adjacent pair of checkpoints.

Altogether, $O(\sqrt{n}/\varepsilon)$ space.

Can reconstruct any substring of S with some loss

How much loss?

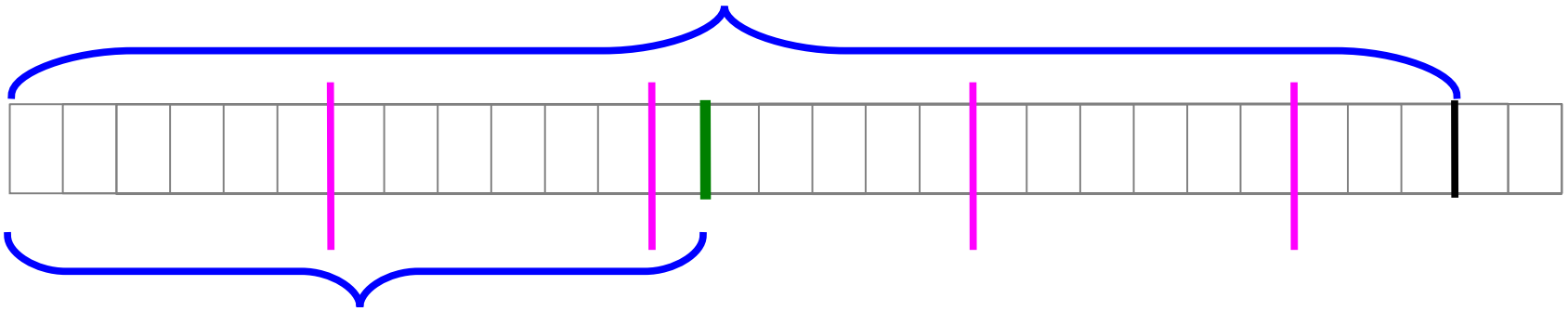
$O(\sqrt{n}/\varepsilon)$ characters lost at each end of S if endpoints of S fall between checkpoints.

More Keeping Track

For every candidate midpoint m, store the substring up to m.

Store prefix up to current character

Store everything forward and backward (not really necessary)



Space Needs

Checkpointing not too bad.

Storing midpoints is potentially a problem.

We could be storing up to linear midpoints.

More on midpoint storage later.

“Growing” A Palindrome

When we first notice a long palindrome, its size is exactly the same as the sliding window.

As we go, we “grow” our palindrome.

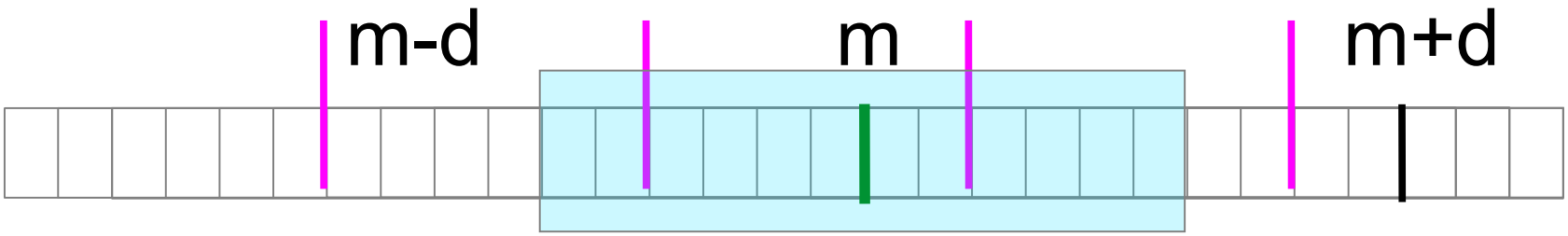
Once we can't grow anymore, we report it in terms of its

- Midpoint (we need to be exact)
- Length (we can underestimate a bit)

“Growing” A Palindrome

The blue box is a known candidate palindrome centered at point m. We would like to grow this palindrome if possible.

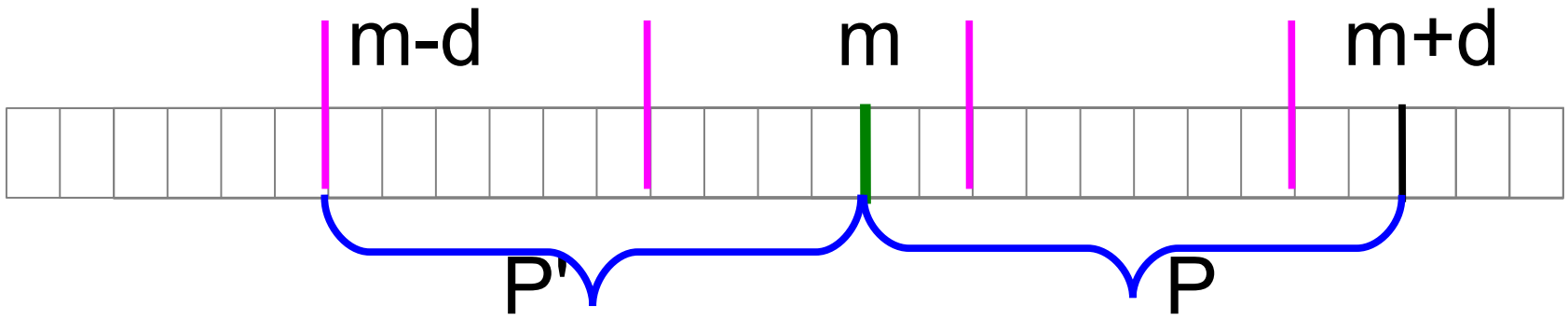
This can only be done when we are at point m+d, where m-d is a checkpoint.



“Growing” A Palindrome

We can reconstruct the string from a midpoint m to any checkpoint that comes before it (say at distance d).

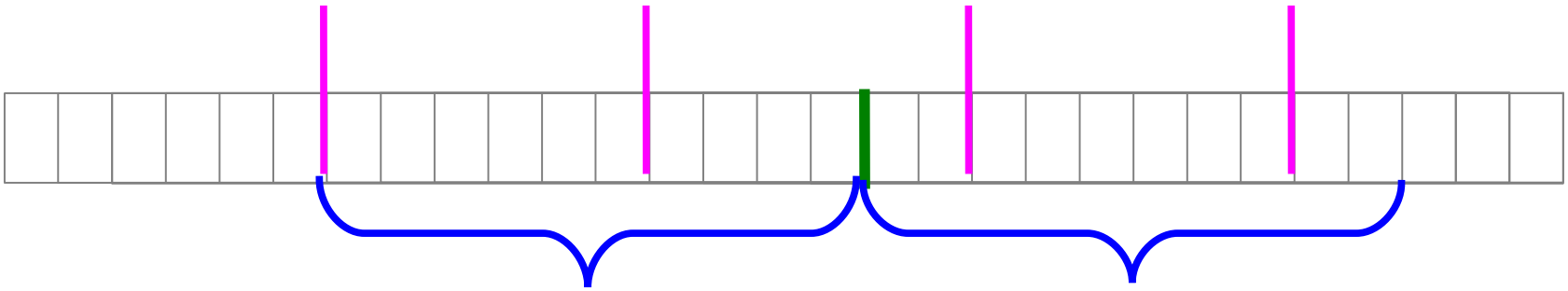
At location m+d, we check the d spots before and after m: is P' reverse of P?



Approximation

Any palindrome we detect necessarily starts at a checkpoint.

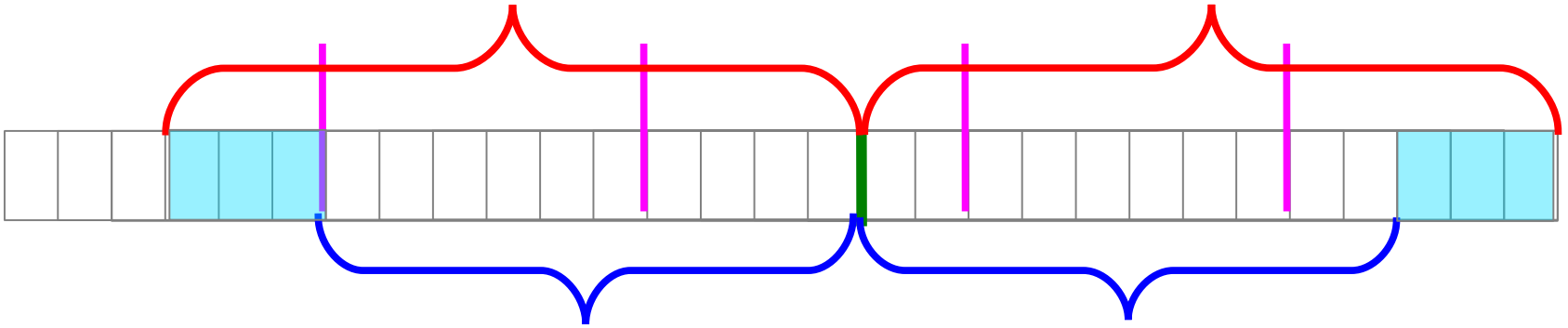
If the left endpoint of a palindrome falls between two checkpoints, the portion until the first checkpoint is missed.



Approximation

Any palindrome we detect necessarily starts at a checkpoint.

If the left endpoint of a palindrome falls between two checkpoints, the portion until the first checkpoint is missed.



Approximation Analysis

The distance between successive checkpoints is $\varepsilon\sqrt{n}$, which is an upper bound on the additive error.

How Much Space?

We store $O(\sqrt{n}/\varepsilon)$ fingerprints for everything except the midpoints.

Midpoints need to be stored as long as the palindrome around them is growing. There could be a linear number.

If any point is covered by a constant number of palindromes, we can process them separately.

It is possible that each character is in a linear number of palindromes. BAD!

How Much Space?

If a point lies under many palindromes, these palindromes must be overlapping in a major way.

In pattern matching, overlaps are well understood -- they can occur in very specific ways and overlapping patterns can be compressed due to periodicity.

Overlapping palindromes show a very distinct pattern as well.

Overlapping Palindromes

ABBAABBAABBAABBAABBAABBAABBA

Palindromes cannot overlap arbitrarily.

If they do, they show a periodic pattern.

Simple Example on Patterns

On general strings, studied in [PP,EJ,GB]

A B C D B C D B C D B B C D B C D B C K L

Simplified view: a pattern can overlap with another copy of itself either at periodic intervals, or at most once.

More on Overlaps

A B C D B C D B C D B B C D B C D B C K L

Closely spaced (n/2 or closer) overlaps are indicators of a periodic string.

A periodic string can be summarized as:

Length, period/length of period, #repeats, suffix

(10, BCD, 3, B).

Then we can reconstruct substrings of the string on the fly.

What about Overlapping Palindromes ?

A run is a contiguous sequence of midpoints with "small" distance between them.

A run must have the following form:

w w' w w' w w' w w'...

Where w is a short substring and w' is its reverse.

A run can be remembered and reconstructed on the fly using constant space.

So, Ultimately...

- Short palindromes: they fit in the sliding window,
- Long palindromes, one of two cases:

  They are far apart and we don't have too many; we can process them in O(1) space

  They are close together, then they form a run; we can store in O(1) space again.

In either case we do not need to keep too many midpoints.

Final Theorem

We can return all palindromes in stream S of length n in one pass using $\sqrt{n}/\varepsilon$ space, where the lengths of the palindromes are underestimated by at most $\varepsilon\sqrt{n}$.

Outline

- Preliminaries, observations
- An additive-error approximation
- An exact algorithm
- A multiplicative error approximation
- A quick look at a lower bound

What Can We Obtain from One More Pass?

Adding one more pass to our algorithm yields an exact algorithm that returns the following in $O(\sqrt{n})$ space:

A value ℓ_{max} which is the length of the longest palindrome in S

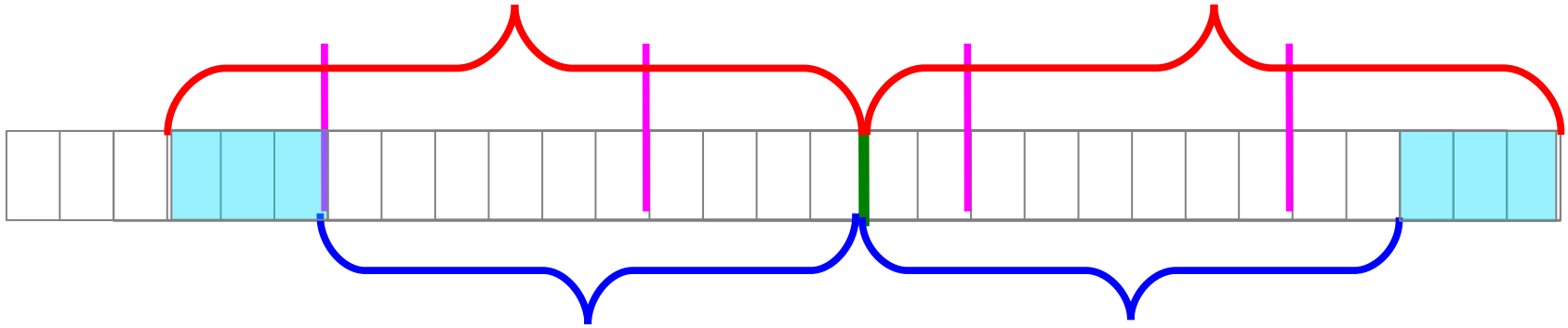
All palindromes of length ℓ_{max} in S.

An Exact Algorithm

Recall: our error due to left endpoints falling between checkpoints.

To fix this error, we need to remember this section that we never track.

Remember first turquoise, compare with second turquoise.



Outline of Exact Algorithm

During the first pass, make note of palindromes that span the largest number of checkpoints.

During the second pass, pay more attention to each such palindrome: remember the intervals at both their ends (for runs, it suffices to remember the palindrome in the middle only).

Calculating the lengths exactly this way will not increase asymptotic space.

Outline

- Preliminaries, observations
- An additive-error approximation
- An exact algorithm
- A multiplicative error approximation
- A quick look at a lower bound

A Logspace Algorithm

A slightly different question:

Find the (approximately) longest palindrome in stream S of length n.

Accuracy: of length at least $\ell_{max}/(1+\varepsilon)$

in space $O(\log n/\varepsilon\log(1+\varepsilon))$.

We may not return the longest palindrome, but one which is very close.

Intuition

Why is the previous algorithm space inefficient?
It keeps track of many checkpoints.

If we reduce the number of checkpoints to logarithmic, additive error will jump to n/log n.

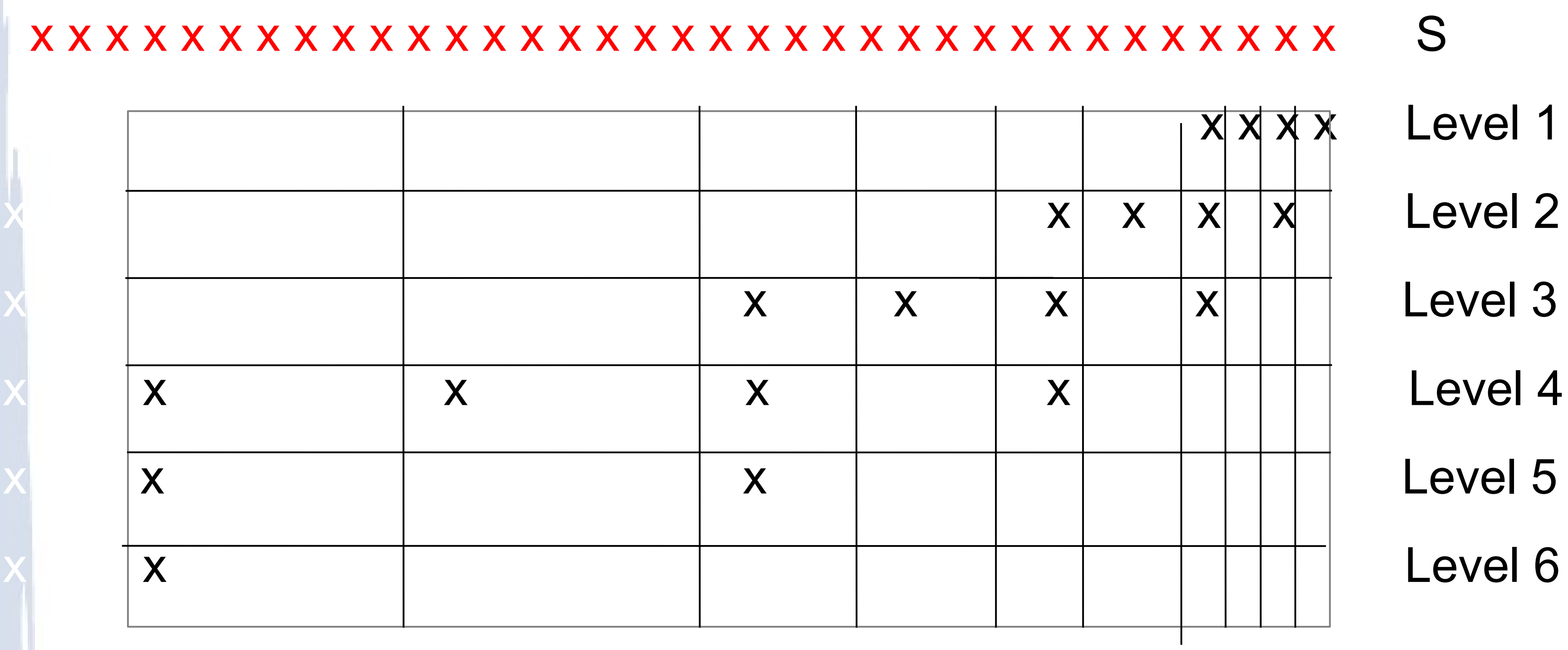
We will space checkpoints at exponentially dropping intervals as distance from current location grows.

Exponentially Increasing Intervals

Technique used in many disguises in approximation algorithms:

As the potential length of a palindrome grows, so does the size of allowed multiplicative error. We will thus use sparser checkpoints for longer palindromes.

Checkpoint Layers:



Space Use

In each layer we have four checkpoints.

Palindromes in each layer can be compressed as before (checkpointing is the main reason for space usage)

We have a logarithmic number of layers.

Space required is O(log n).

Outline

- Preliminaries, observations
- An additive-error approximation
- An exact algorithm
- A multiplicative error approximation
- A quick look at a lower bound

Lower Bound

Additive $\sqrt{n}$-approximation to the longest palindrome requires $\sqrt{n}$ space.

How?

For deterministic, generate a distribution whose inputs need to be separated.

Use Yao's technique to convert into randomized lower bound.

Wish List/Future Work

A simpler technique exists which does not change complexity.

Palindromes with mismatches: what we would like for real applications. Hamming, and finally, Edit Distance.

Other structures than palindromes/generalized palindromes.

Thank youoy knahT