Efficient Data Structures for the Factor Periodicity Problem

Tomasz Kociumaka Jakub Radoszewski
Wojciech Rytter Tomasz Waleń

University of Warsaw, Poland

SPIRE 2012 Cartagena, October 23, 2012

Factor Periodicity Problem

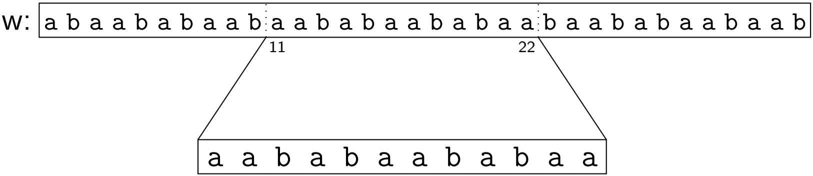
w: `a b a a b a b a a b a a b a b a a b a b a a b a a b a b a a b a a b`

Factor Periodicity Problem

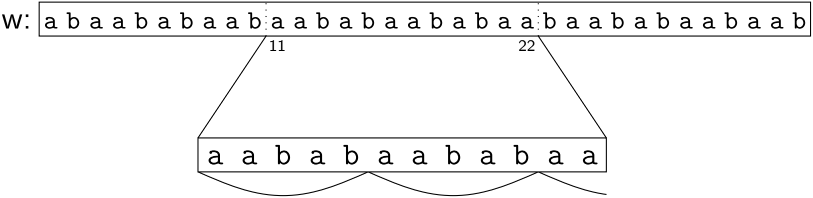
w: `a b a a b a b a a b a a b a b a a b a b a a b a a b a b a a b a a b`

11 22

Factor Periodicity Problem

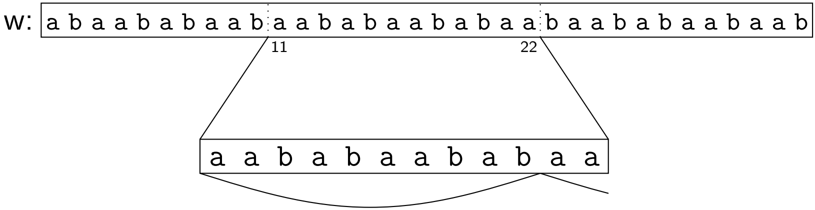


Factor Periodicity Problem



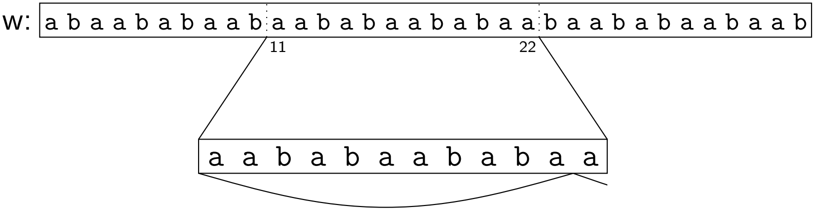
Periods of $w[11..22]$ are 5

Factor Periodicity Problem



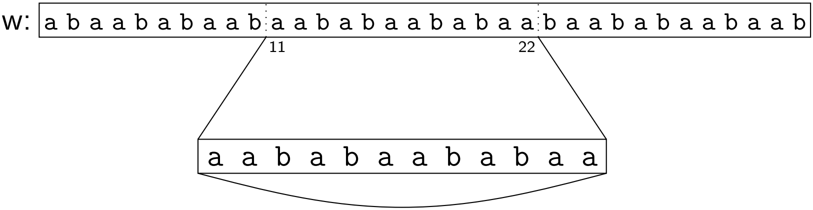
Periods of $w[11..22]$ are 5, 10

Factor Periodicity Problem



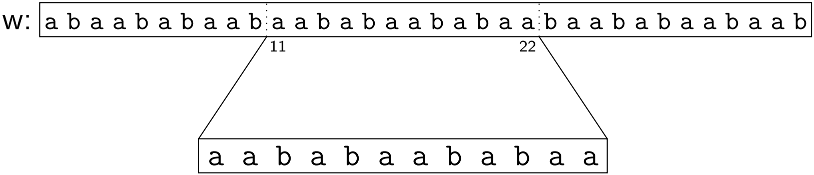
Periods of $w[11..22]$ are 5, 10, 11

Factor Periodicity Problem



Periods of $w[11..22]$ are 5, 10, 11 and 12.

Factor Periodicity Problem



Periods of $w[11..22]$ are 5, 10, 11 and 12.
Notation $Per(w[11..22]) = \{5, 10, 11, 12\}$, $per(w[11..22]) = 5$.

Arithmetic sets

A word of length m might have $\Theta(m)$ periods, e.g. a^m.

Definition

A set $A = \{a, a+d, a+2d, \ldots, a+kd\} \subseteq \mathbb{Z}$ is called *arithmetic*. An integer d is called the *difference* of A.

Observe that an arithmetic set can be represented by three integers: a, d and k.

Arithmetic sets

A word of length m might have $\Theta(m)$ periods, e.g. $\mathtt{a}^m$.

Definition

A set $A = \{a, a+d, a+2d, \ldots, a+kd\} \subseteq \mathbb{Z}$ is called *arithmetic*. An integer d is called the *difference* of A.

Observe that an arithmetic set can be represented by three integers: a, d and k.

Fact

Let v be a word of length m. Then $Per(v)$ is a union of at most $\log m$ disjoint arithmetic sets.

For example
$$Per(w[11..22]) = \{5\} \cup \{10, 11, 12\} = \{5, 10\} \cup \{11, 12\}.$$

Formal problem statement

Problem (Period Queries)

Design a data structure that for a fixed word w *of length* n *answers the following queries. Given integers* i, j *(*$1 \leq i \leq j \leq n$*) compute* $Per(w[i..j])$ *respresented as a union of* $O(\log n)$ *arithmetic sets.*

Formal problem statement

Problem (Period Queries)

Design a data structure that for a fixed word w *of length* n *answers the following queries. Given integers* i, j *(*$1 \leq i \leq j \leq n$*) compute* $Per(w[i..j])$ *respresented as a union of* $O(\log n)$ *arithmetic sets.*

Definition

We say that p is an $(1+\delta)$-period of v if $|v| \geq (1+\delta)p$.

Formal problem statement

Problem (Period Queries)

Design a data structure that for a fixed word w of length n answers the following queries. Given integers i, j ($1 \le i \le j \le n$) compute $Per(w[i..j])$ respresented as a union of $O(\log n)$ arithmetic sets.

Definition

We say that p is an $(1+\delta)$-period of v if $|v| \ge (1+\delta)p$.

Problem ($(1+\delta)$-Period Queries)

Let us fix a real number $\delta > 0$. Design a data structure that for a fixed word w of length n answers the following queries. Given integers i, j ($1 \le i \le j \le n$) compute all $(1+\delta)$-periods of $w[i..j]$ respresented as a union of $O(1)$ arithmetic sets.

Related work

- To the best of our knowledge no previous research on the general case of Period Queries.
- Even for computing the shortest period, only straightforward solutions:
  - memorize all answers — $O(n^2)$ space, $O(1)$ query time
  - compute the answer from scratch for each query — no extra space, $O(n)$ query time
- Efficient data structures for primitivity testing (generalized by 2-Period Queries)
  - Karhumäki, Lifshits & Rytter; CPM 2007 $O(n \log n)$ space, $O(1)$ query time,
  - Crochemore et. al; SPIRE 2010 $O(n \log^{\varepsilon} n)$ space, $O(\log n)$ query time.

Our results

Several results based on the common idea but different tools.

Space	All periods	$(1+\delta)$-periods
$O(n)$	$O(\log^{1+\varepsilon} n)$	$O(\log^{\varepsilon} n)$
$O(n \log\log n)$	$O(\log n (\log\log n)^2)$	$O((\log\log n)^2)$
$O(n \log^{\varepsilon} n)$	$O(\log n \log\log n)$	$O(\log\log n)$
$O(n \log n)$	$O(\log n)$	$O(1)$

Our results

Several results based on the common idea but different tools.

Space	All periods	$(1+\delta)$-periods
$O(n)$	$O(\log^{1+\varepsilon} n)$	$O(\log^{\varepsilon} n)$
$O(n \log\log n)$	$O(\log n (\log\log n)^2)$	$O((\log\log n)^2)$
$O(n \log^{\varepsilon} n)$	$O(\log n \log\log n)$	$O(\log\log n)$
$O(n \log n)$	$O(\log n)$	$O(1)$

Standard assumptions on the model of computation:

- word RAM model with $w = \Omega(\log n)$,
- randomization.

Our approach

Let $Borders(v) = \{|u| : u \text{ is a border of } v\}$.

Fact

$Per(v) = |v| \ominus Borders(v) = \{|v| - b : b \in Borders(v)\}$.

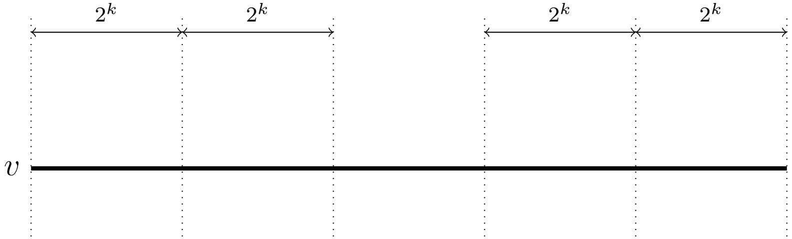
Our approach

Let $Borders(v) = \{|u| : u$ is a border of $v\}$.

Fact

$Per(v) = |v| \ominus Borders(v) = \{|v| - b : b \in Borders(v)\}$.

We compute $Borders(v) \cap \{2^k, \ldots, 2^{k+1}\}$ separately for each $k \in \{0, \ldots, \lceil \log |v| \rceil\}$.



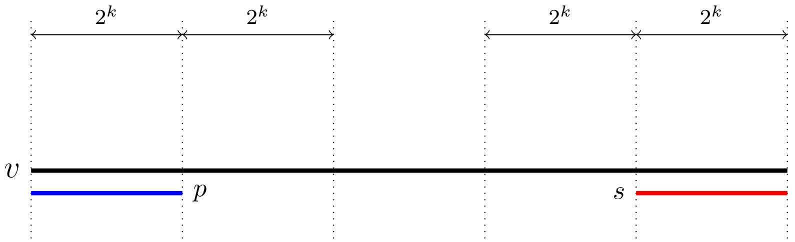
Our approach

Let $Borders(v) = \{|u| : u$ is a border of $v\}$.

Fact

$Per(v) = |v| \ominus Borders(v) = \{|v| - b : b \in Borders(v)\}$.

We compute $Borders(v) \cap \{2^k, \ldots, 2^{k+1}\}$ separately for each $k \in \{0, \ldots, \lceil \log |v| \rceil\}$.



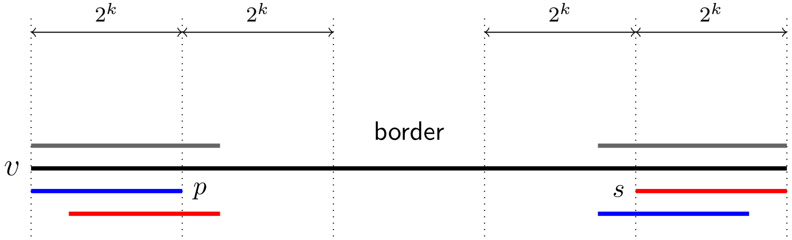
Our approach

Let $Borders(v) = \{|u| : u$ is a border of $v\}$.

Fact

$Per(v) = |v| \ominus Borders(v) = \{|v| - b : b \in Borders(v)\}$.

We compute $Borders(v) \cap \{2^k, \ldots, 2^{k+1}\}$ separately for each $k \in \{0, \ldots, \lceil \log |v| \rceil\}$.



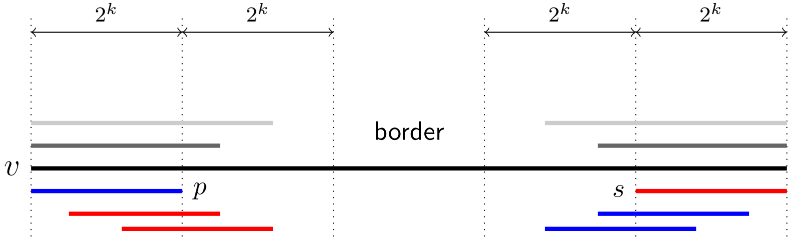
Our approach

Let $Borders(v) = \{|u| : u$ is a border of $v\}$.

Fact

$Per(v) = |v| \ominus Borders(v) = \{|v| - b : b \in Borders(v)\}$.

We compute $Borders(v) \cap \{2^k, \ldots, 2^{k+1}\}$ separately for each $k \in \{0, \ldots, \lceil \log |v| \rceil\}$.

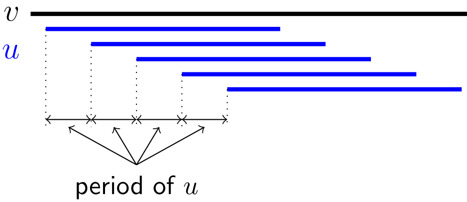


Close occurrences

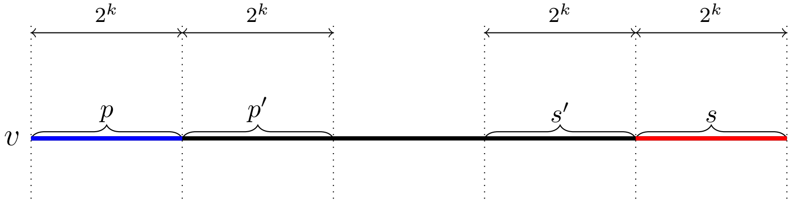
Let $Occ(v, u)$ be the set of positions of v where an occurrence of u starts. Arithmetic sets naturally appear as the Occ sets.

Fact

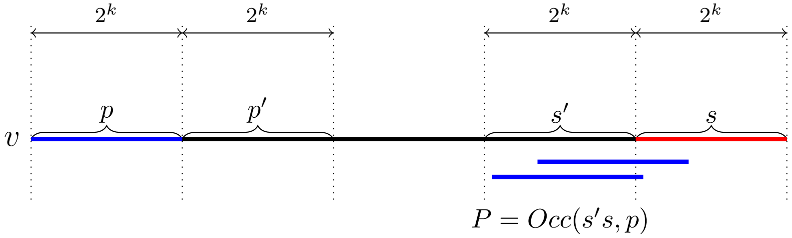
Let $|v| \leq 2|u|$. Then $Occ(v, u)$ is arithmetic. Moreover, if $|Occ(v, u)| \geq 3$ then its difference is equal to $per(u)$.



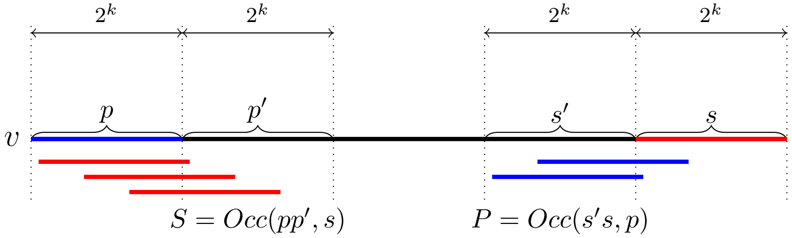
A formula for border lengths



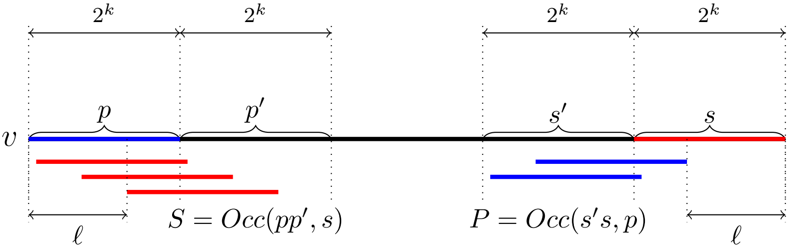
A formula for border lengths



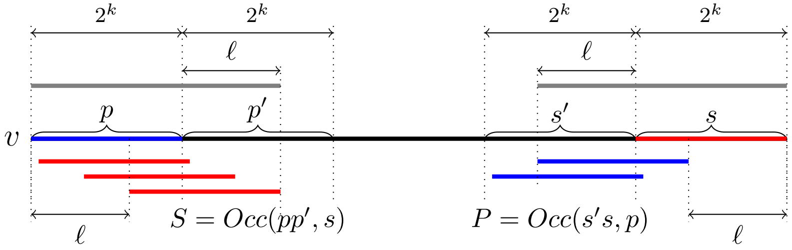
A formula for border lengths



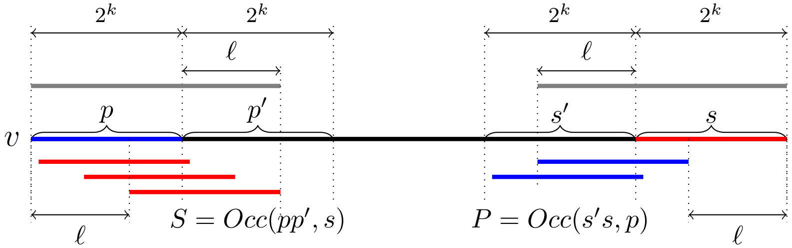
A formula for border lengths



A formula for border lengths



A formula for border lengths



Fact

Let $0 \leq \ell < 2^k$. Then the word v has a border of length $2^k + \ell$ if and only if $\ell + 1 \in S$ and $2^k - \ell \in P$.

Consequently $Borders(v) \cap \{2^k, \ldots, 2^{k+1}\}$ is arithmetic.

Intersecting arithmetic sets

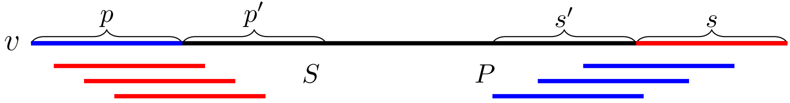
Lemma

If $|P| \geq 3$ *and* $|S| \geq 3$, *then* $per(p) = per(s)$. *Consequently* P *and* S *are arithmetic of common difference.*

Intersecting arithmetic sets

Lemma

If $|P| \geq 3$ *and* $|S| \geq 3$, *then* $per(p) = per(s)$. *Consequently* P *and* S *are arithmetic of common difference.*



Intersecting arithmetic sets

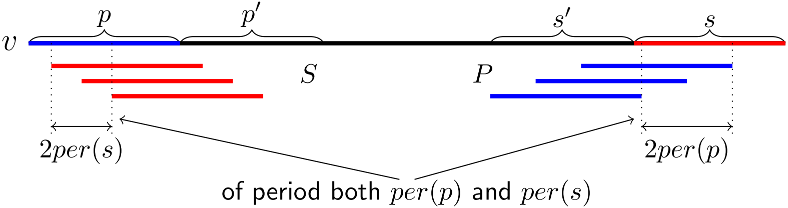
Lemma

If $|P| \geq 3$ *and* $|S| \geq 3$*, then* $per(p) = per(s)$*. Consequently* P *and* S *are arithmetic of common difference.*

Intersecting arithmetic sets

Lemma

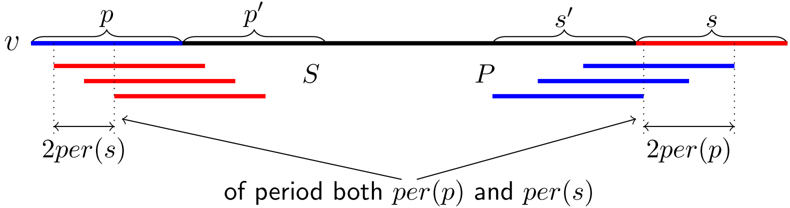
If $|P| \geq 3$ *and* $|S| \geq 3$*, then* $per(p) = per(s)$*. Consequently* P *and* S *are arithmetic of common difference.*



Intersecting arithmetic sets

Lemma

If $|P| \geq 3$ and $|S| \geq 3$, then $per(p) = per(s)$. Consequently P and S are arithmetic of common difference.



Intersecting two arithmetic sets can be performed in $O(1)$ time, when one of them is small or when they share a common difference.

Summary of the combinatorial part

Problem (Occurrence Queries)

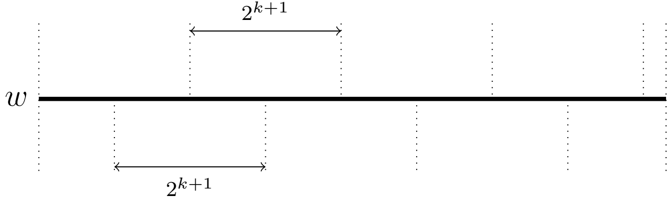
Design a data structure that for a word w can answer the following queries. Given a basic factor u and a factor v of w such that $|v| \le 2|u|$ (both represented by one of their occurrences) compute the arithmetic set $Occ(v, u)$.

Theorem

Assume there is a data structure answering the Occurrence Queries in $O(f(n))$ time. Then this data structure can answer Period Queries in $O(f(n) \log n)$ time and $(1 + \delta)$-Period Queries in $O(f(n))$ time.

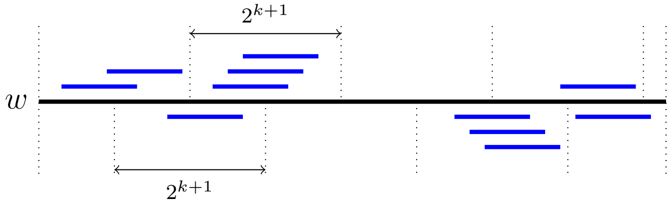
Occurrence Queries in $O(1)$ time

- Fix $2^k \leq n$,
- Split w into parts of length 2^{k+1} with overlaps of size 2^k,



Occurrence Queries in $O(1)$ time

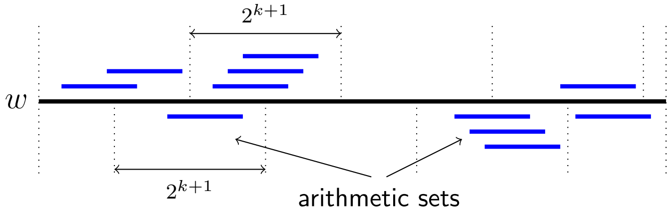
- Fix $2^k \le n$,
- Split w into parts of length 2^{k+1} with overlaps of size 2^k,
- Consider a basic factor u, $|u| = 2^k$.



- Each occurrence of u occurs within a single part.

Occurrence Queries in $O(1)$ time

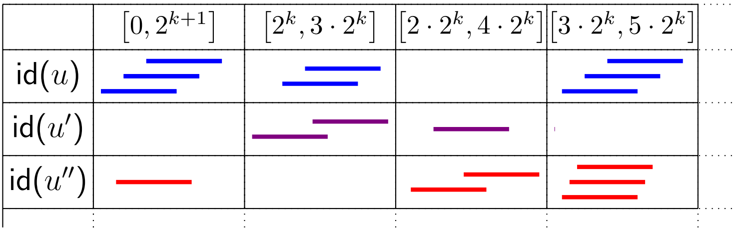
- Fix $2^k \leq n$,
- Split w into parts of length 2^{k+1} with overlaps of size 2^k,
- Consider a basic factor u, $|u| = 2^k$.



- Each occurrence of u occurs within a single part.
- Occurrences in a single part form an arithmetic set.

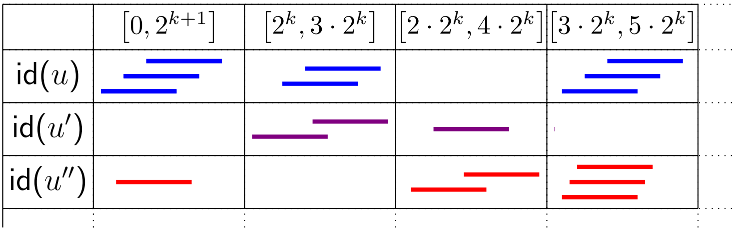
Occurrence Queries in $O(1)$ time

Imagine a (large) array with columns indexed by parts and rows by identifiers of all basic factors of length 2^k. The identifiers are obtained from the DBF (Dictionary of Basic Factors)



Occurrence Queries in $O(1)$ time

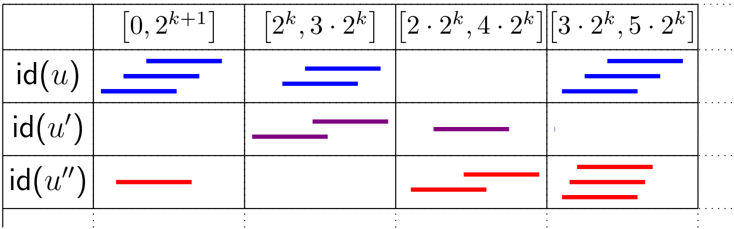
Imagine a (large) array with columns indexed by parts and rows by identifiers of all basic factors of length 2^k. The identifiers are obtained from the DBF (Dictionary of Basic Factors)



- This array has $\Theta\left(\frac{n^2}{2^k}\right)$ cells.

Occurrence Queries in $O(1)$ time

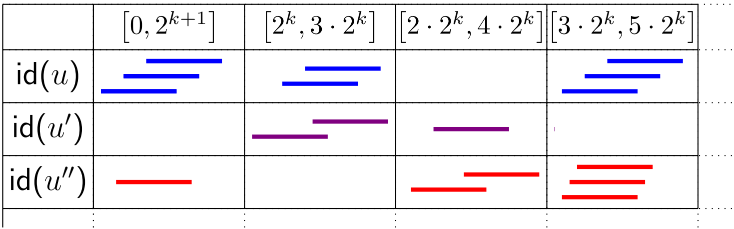
Imagine a (large) array with columns indexed by parts and rows by identifiers of all basic factors of length 2^k. The identifiers are obtained from the DBF (Dictionary of Basic Factors)



- This array has $\Theta\left(\frac{n^2}{2^k}\right)$ cells.
- All factors of length 2^k have $\leq n$ occurrences in total, so $\leq n$ non-empty fields — *perfect hashing* can be used.

Occurrence Queries in $O(1)$ time

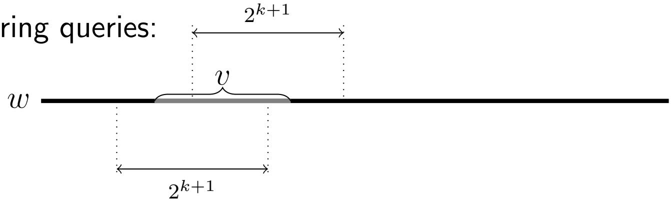
Imagine a (large) array with columns indexed by parts and rows by identifiers of all basic factors of length 2^k. The identifiers are obtained from the DBF (Dictionary of Basic Factors)



- This array has $\Theta\left(\frac{n^2}{2^k}\right)$ cells.
- All factors of length 2^k have $\leq n$ occurrences in total, so $\leq n$ non-empty fields — *perfect hashing* can be used.
- This gives $O(n \log n)$ size in total for all values of k.

Occurrence Queries in $O(1)$ time

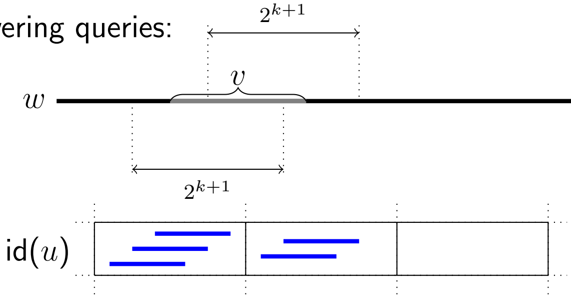
Answering queries:



- v lies within at most two consecutive parts,

Occurrence Queries in $O(1)$ time

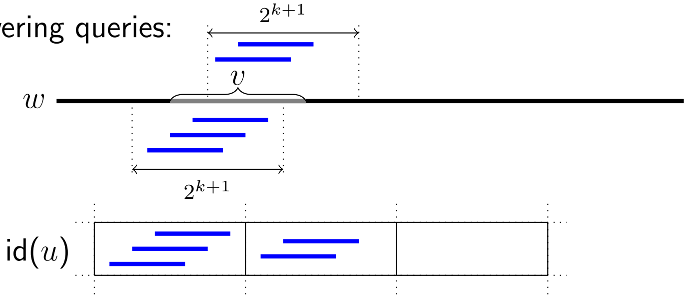
Answering queries:



- v lies within at most two consecutive parts,
- get the occurrences of u from the hash table,

Occurrence Queries in $O(1)$ time

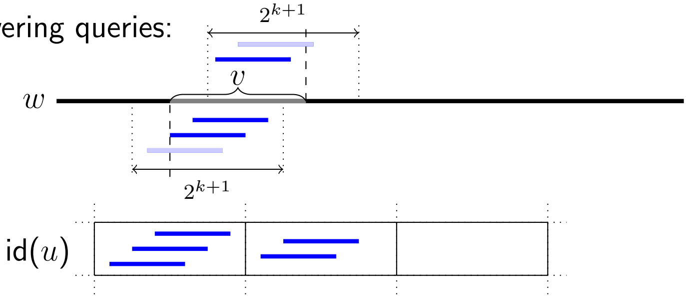
Answering queries:



- v lies within at most two consecutive parts,
- get the occurrences of u from the hash table,

Occurrence Queries in $O(1)$ time

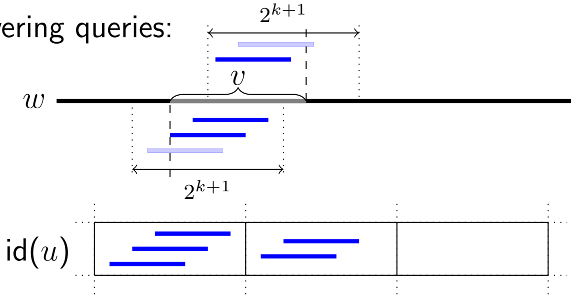
Answering queries:



- v lies within at most two consecutive parts,
- get the occurrences of u from the hash table,
- crop and merge these arithmetic sets to obtain the result.

Occurrence Queries in $O(1)$ time

Answering queries:



- v lies within at most two consecutive parts,
- get the occurrences of u from the hash table,
- crop and merge these arithmetic sets to obtain the result.

Corollary

There exists a data structure of $O(n \log n)$ size that answers the Occurrence Queries in $O(1)$ time.

Range Predecessor/Successor Queries

Problem (Range Predecessor/Successor Queries)

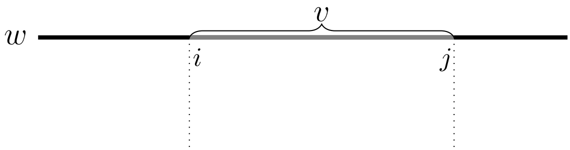
Design a data structure that for a word w can answer the following queries. Given a factor u of w (represented by an occurrence in w) and $i \in \{1 \dots n\}$ find $PRED(u, i)$ — the last occurrence of u ending at a position $\leq i$, $SUCC(u, i)$ — the first occurrence of u starting at a position $\geq i$.

Range Predecessor/Successor Queries

Problem (Range Predecessor/Successor Queries)

Design a data structure that for a word w can answer the following queries. Given a factor u of w (represented by an occurrence in w) and $i \in \{1 \dots n\}$ find $PRED(u, i)$ — the last occurrence of u ending at a position $\leq i$, $SUCC(u, i)$ — the first occurrence of u starting at a position $\geq i$.

The Occurrence Queries can be reduced to three Range Predecessor/Successor Queries, where u is a basic factor of w.

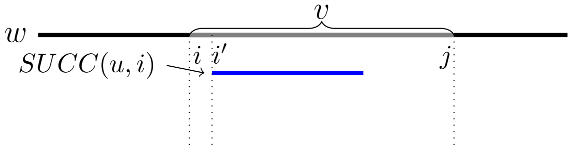


Range Predecessor/Successor Queries

Problem (Range Predecessor/Successor Queries)

Design a data structure that for a word w can answer the following queries. Given a factor u of w (represented by an occurrence in w) and $i \in \{1 \dots n\}$ find $PRED(u, i)$ — the last occurrence of u ending at a position $\leq i$, $SUCC(u, i)$ — the first occurrence of u starting at a position $\geq i$.

The Occurrence Queries can be reduced to three Range Predecessor/Successor Queries, where u is a basic factor of w.

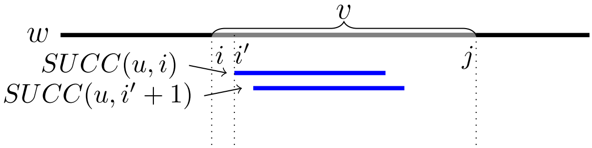


Range Predecessor/Successor Queries

Problem (Range Predecessor/Successor Queries)

Design a data structure that for a word w can answer the following queries. Given a factor u of w (represented by an occurrence in w) and $i \in \{1 \dots n\}$ find $PRED(u,i)$ — the last occurrence of u ending at a position $\leq i$, $SUCC(u,i)$ — the first occurrence of u starting at a position $\geq i$.

The Occurrence Queries can be reduced to three Range Predecessor/Successor Queries, where u is a basic factor of w.

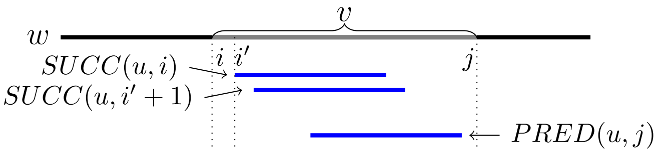


Range Predecessor/Successor Queries

Problem (Range Predecessor/Successor Queries)

Design a data structure that for a word w can answer the following queries. Given a factor u of w (represented by an occurrence in w) and $i \in \{1 \dots n\}$ find $PRED(u, i)$ — the last occurrence of u ending at a position $\leq i$, $SUCC(u, i)$ — the first occurrence of u starting at a position $\geq i$.

The Occurrence Queries can be reduced to three Range Predecessor/Successor Queries, where u is a basic factor of w.

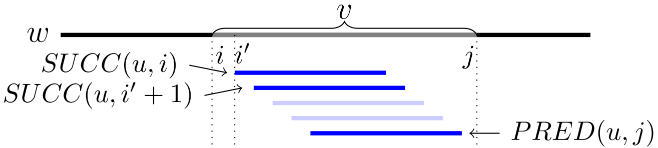


Range Predecessor/Successor Queries

Problem (Range Predecessor/Successor Queries)

Design a data structure that for a word w can answer the following queries. Given a factor u of w (represented by an occurrence in w) and $i \in \{1 \dots n\}$ find $PRED(u, i)$ — the last occurrence of u ending at a position $\le i$, $SUCC(u, i)$ — the first occurrence of u starting at a position $\ge i$.

The Occurrence Queries can be reduced to three Range Predecessor/Successor Queries, where u is a basic factor of w.



Range Predecessor/Successor Queries

Theorem (Nekrich, Navarro; 2012)

There exist data structures that given the locus of u in the suffix tree of w answer the Range Predecessor/Successor queries in and satisfy the following space and time bounds:

Space	*Query time*
$O(n)$	$O(\log^{\varepsilon} n)$
$O(n \log\log n)$	$O((\log\log n)^2)$
$O(n \log^{\varepsilon} n)$	$O(\log\log n)$

Theorem (Weighted LA — Kopelovitz, Lewenstein; 2007)

There exists a data structure of size $O(n)$, which given an interval $[i..j]$ finds the locus of $w[i..j]$ in the suffix tree of w in $O(\log\log n)$ time.

Summary

Theorem (this paper)

There exist data structures that satisfy following time and space bounds for size, Period Queries query time and $(1+\delta)$*-Period Queries query time:*

Space	*Period Queries*	$(1+\delta)$*-Period Q.*
$O(n)$	$O(\log^{1+\varepsilon} n)$	$O(\log^{\varepsilon} n)$
$O(n \log\log n)$	$O(\log n (\log\log n)^2)$	$O((\log\log n)^2)$
$O(n \log^{\varepsilon} n)$	$O(\log n \log\log n)$	$O(\log\log n)$
$O(n \log n)$	$O(\log n)$	$O(1)$

Further research

Space	Period Queries	$(1+\delta)$-Period Q.
$O(n)$	$O(\log^{1+\varepsilon} n)$	$O(\log^{\varepsilon} n)$
$O(n \log\log n)$	$O(\log n (\log\log n)^2)$	$O((\log\log n)^2)$
$O(n \log^{\varepsilon} n)$	$O(\log n \log\log n)$	$O(\log\log n)$
$O(n \log n)$	$O(\log n)$	$O(1)$

Further research

Currently in progress:

Space	Period Queries	2-Period Queries
$O(n)$	$O(\log^{1+\varepsilon} n)$	$O(\log^{\varepsilon} n)$
$O(n \log\log n)$	$O(\log n (\log\log n)^2)$	$O((\log\log n)^2)$
$O(n \log^{\varepsilon} n)$	$O(\log n \log\log n)$	$O(\log\log n)$
$O(n \log n)$	$O(\log n)$	$O(1)$
$O(n)$	—	$O(1)$

Further research

Currently in progress:

Space	Period Queries	2-Period Queries
$O(n)$	$O(\log^{1+\varepsilon} n)$	$O(\log^{\varepsilon} n)$
$O(n \log\log n)$	$O(\log n \log\log n)$	$O(\log\log n)$
$O(n \log n)$	$O(\log n)$	$O(1)$
$O(n)$	—	$O(1)$

Further research

Currently in progress:

Space	Period Queries	2-Period Queries
$O(n)$	$O(\log^{1+\varepsilon} n)$	$O(\log^{\varepsilon} n)$
$O(n \log\log n)$	$O(\log n \log\log n)$	$O(\log\log n)$
$O(n \log n)$	$O(\log n)$	$O(1)$
$O(n)$	—	$O(1)$

Open problems:

- Can the $O(n \log n)$ time preprocessing be improved with $o(n)$ query time?
- Can the shortest period be found faster than $O(\log n)$ with $o(n^2)$ space?

Thank you for your attention

Thank you!