Fast Algorithms for Abelian Periods in Words and Greatest Common Divisor Queries

Tomasz Kociumaka, Jakub Radoszewski,
Wojciech Rytter

University of Warsaw

STACS 2013 Kiel, February 28, 2013

Part I

Greatest Common Divisor Queries

Problem

Problem (Greatest Common Divisor)

For a positive integer n build a data structure that given integers $x, y \in \{1, \ldots, n\}$ computes $gcd(x, y)$.

Problem

Problem (Greatest Common Divisor)

For a positive integer n build a data structure that given integers $x, y \in \{1, \ldots, n\}$ computes $gcd(x, y)$.

RAM model with word-size $\Omega(\log n)$, i.e. constant-time arithmetic operations on $O(\log n)$-bit integers.

Problem

Problem (Greatest Common Divisor)

For a positive integer n build a data structure that given integers $x, y \in \{1, \ldots, n\}$ computes $gcd(x, y)$.

RAM model with word-size $\Omega(\log n)$, i.e. constant-time arithmetic operations on $O(\log n)$-bit integers.

	space	construction	query time
Euclid's algorithm	-	-	$O(\log n)$
precompute answers	$O(n^2)$	$O(n^2)$	$O(1)$
use factorization	$O(n)$	$O(n)$	$O(\frac{\log n}{\log \log n})$
this work	$O(n)$	$O(n)$	$O(1)$

Special factorization

Computing $gcd(x, y)$ is sometimes easy:

- we can precompute $gcd[x', y']$ for every $x', y' \leq \sqrt{n}$ and then for $x \leq \sqrt{n}$ we can use the precomputed answer $gcd[x, y \bmod x]$,
- if x is prime it suffices to check whether x divides y.

Special factorization

Computing $gcd(x, y)$ is sometimes easy:

- we can precompute $gcd[x', y']$ for every $x', y' \leq \sqrt{n}$ and then for $x \leq \sqrt{n}$ we can use the precomputed answer $gcd[x, y \bmod x]$,
- if x is prime it suffices to check whether x divides y.

Definition

Let k be a positive integer. Then (k_1, k_2, k_3) is a *special decomposition* of k if $k = k_1 k_2 k_3$ and each k_i is prime or does not exceed $\sqrt{k}$.

Queries

The data structure consists of:

- precomputed answers for any $x, y \leq \sqrt{n}$,
- a special decomposition of each $x \in \{1, \ldots, n\}$.

Queries

The data structure consists of:

- precomputed answers for any $x, y \leq \sqrt{n}$,
- a special decomposition of each $x \in \{1, \ldots, n\}$.

Algorithm *gcd*(x, y)

$(x_1, x_2, x_3) := decomp[x]$;
$g := 1$;
for $i := 1$ **to** 3 **do**
  if $x_i \leq \sqrt{n}$ **then**
    $d := gcd[x_i, y \bmod x_i]$;
  else if $x_i \mid y$ **then** $d := x_i$;
  else $d := 1$;
  $g := g \cdot d$;
  $y := y/d$;
return g;

853
11
3
3
2
2

$y = 337788$

853
7
5
3
2
2
2

$x = 716520$

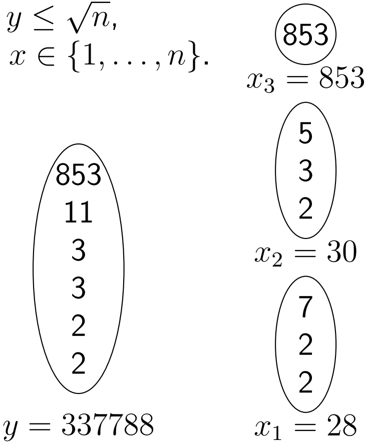
Queries

The data structure consists of:

- precomputed answers for any $x, y \le \sqrt{n}$,
- a special decomposition of each $x \in \{1, \ldots, n\}$.

Algorithm *gcd*(x, y)

$(x_1, x_2, x_3) := decomp[x]$;
$g := 1$;
for $i := 1$ **to** 3 **do**
  if $x_i \le \sqrt{n}$ **then**
    $d := gcd[x_i, y \bmod x_i]$;
  else if $x_i \mid y$ **then** $d := x_i$;
  else $d := 1$;
  $g := g \cdot d$;
  $y := y/d$;
return g;



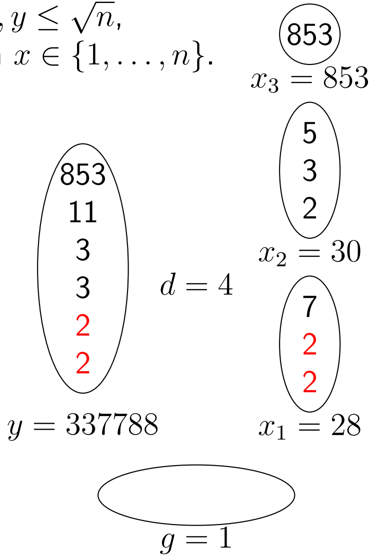
Queries

The data structure consists of:

- precomputed answers for any $x, y \leq \sqrt{n}$,
- a special decomposition of each $x \in \{1, \ldots, n\}$.

Algorithm *gcd*(x, y)

$(x_1, x_2, x_3) := decomp[x]$;
$g := 1$;
for $i := 1$ **to** 3 **do**
  if $x_i \leq \sqrt{n}$ **then**
    $d := gcd[x_i, y \bmod x_i]$;
  else if $x_i \mid y$ **then** $d := x_i$;
  else $d := 1$;
  $g := g \cdot d$;
  $y := y/d$;
return g;



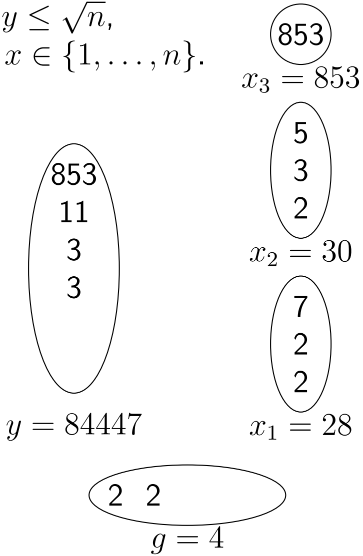
Queries

The data structure consists of:

- precomputed answers for any $x, y \le \sqrt{n}$,
- a special decomposition of each $x \in \{1, \dots, n\}$.

Algorithm *gcd*(x, y)

$(x_1, x_2, x_3) := decomp[x]$;
$g := 1$;
for $i := 1$ **to** 3 **do**
  if $x_i \le \sqrt{n}$ **then**
    $d := gcd[x_i, y \bmod x_i]$;
  else if $x_i \mid y$ **then** $d := x_i$;
  else $d := 1$;
  $g := g \cdot d$;
  $y := y/d$;
return g;



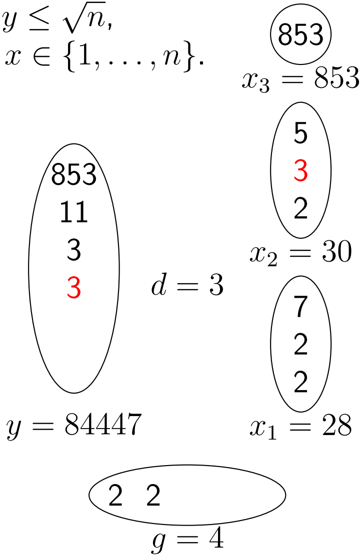
Queries

The data structure consists of:

- precomputed answers for any $x, y \le \sqrt{n}$,
- a special decomposition of each $x \in \{1, \ldots, n\}$.

Algorithm *gcd*(x, y)

$(x_1, x_2, x_3) := decomp[x]$;
$g := 1$;
for $i := 1$ **to** 3 **do**
  if $x_i \le \sqrt{n}$ **then**
    $d := gcd[x_i, y \bmod x_i]$;
  else if $x_i \mid y$ **then** $d := x_i$;
  else $d := 1$;
  $g := g \cdot d$;
  $y := y/d$;
return g;



Queries

The data structure consists of:

- precomputed answers for any $x, y \leq \sqrt{n}$,
- a special decomposition of each $x \in \{1, \ldots, n\}$.

Algorithm *gcd*(x, y)

$(x_1, x_2, x_3) := decomp[x]$;
$g := 1$;
for $i := 1$ **to** 3 **do**
  if $x_i \leq \sqrt{n}$ **then**
    $d := gcd[x_i, y \bmod x_i]$;
  else if $x_i \mid y$ **then** $d := x_i$;
  else $d := 1$;
  $g := g \cdot d$;
  $y := y/d$;
return g;

853
$x_3 = 853$

5
3
2
$x_2 = 30$

7
2
2
$x_1 = 28$

853
11
3
$y = 28149$

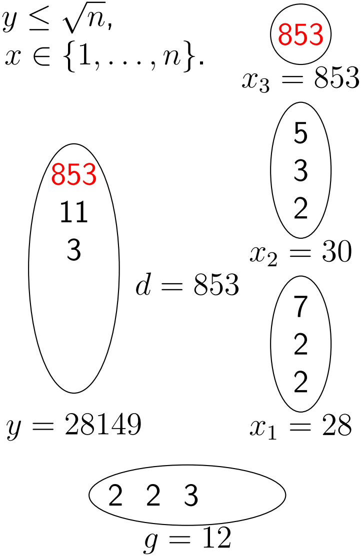
2 2 3
$g = 12$

The data structure consists of:

- precomputed answers for any $x, y \le \sqrt{n}$,
- a special decomposition of each $x \in \{1, \ldots, n\}$.

Algorithm *gcd*(x, y)

  $(x_1, x_2, x_3) := decomp[x]$;
  $g := 1$;
  for $i := 1$ **to** 3 **do**
    if $x_i \le \sqrt{n}$ **then**
      $d := gcd[x_i, y \bmod x_i]$;
    else if $x_i \mid y$ **then** $d := x_i$;
    else $d := 1$;
    $g := g \cdot d$;
    $y := y/d$;
  return g;



Queries

The data structure consists of:

- precomputed answers for any $x, y \leq \sqrt{n}$,
- a special decomposition of each $x \in \{1, \ldots, n\}$.

Algorithm *gcd*(x, y)

$(x_1, x_2, x_3) := decomp[x]$;
$g := 1$;
for $i := 1$ **to** 3 **do**
  if $x_i \leq \sqrt{n}$ **then**
    $d := gcd[x_i, y \bmod x_i]$;
  else if $x_i \mid y$ **then** $d := x_i$;
  else $d := 1$;
  $g := g \cdot d$;
  $y := y/d$;
return g;

11
3

$y = 33$

853

$x_3 = 853$

5
3
2

$x_2 = 30$

7
2
2

$x_1 = 28$

2 2 3 853

$g = 10236$

Construction

Lemma

Let $\ell > 1$ be a positive integer, p be the smallest prime divisor of ℓ and $k = \frac{\ell}{p}$. A decomposition of ℓ can be obtained from a decomposition of k by multiplying the smallest factor by p.

Construction

Lemma

Let $\ell > 1$ be a positive integer, p be the smallest prime divisor of ℓ and $k = \frac{\ell}{p}$. A decomposition of ℓ can be obtained from a decomposition of k by multiplying the smallest factor by p.

Theorem (Gries & Misra, 1978)

The smallest prime divisors for all positive integers up to n can be computed in $O(n)$ time.

Part II

Abelian Periods

Commutative equivalence and Parikh vectors

Definition

Let w be a word over Σ. A Parikh vector $\mathcal{P}(w)$ counts for each letter $a \in \Sigma$ its number of occurrences in w.

$$w = \texttt{a b b a c} \qquad \mathcal{P}(w) = (2, 2, 1)$$

Commutative equivalence and Parikh vectors

Definition

Let w be a word over Σ. A Parikh vector $\mathcal{P}(w)$ counts for each letter $a \in \Sigma$ its number of occurrences in w.

$$w = \mathtt{a\,b\,b\,a\,c} \qquad \mathcal{P}(w) = (2, 2, 1)$$

Commutative equivalence and Parikh vectors

Definition

Let w be a word over Σ. A Parikh vector $\mathcal{P}(w)$ counts for each letter $a \in \Sigma$ its number of occurrences in w.

$$w = \texttt{a}\,\texttt{b}\,\texttt{b}\,\texttt{a}\,\texttt{c} \qquad \mathcal{P}(w) = (2, 2, 1)$$

Commutative equivalence and Parikh vectors

Definition

Let w be a word over Σ. A Parikh vector $\mathcal{P}(w)$ counts for each letter $a \in \Sigma$ its number of occurrences in w.

$$w = \texttt{a b b a c} \qquad \mathcal{P}(w) = (2, 2, 1)$$

Commutative equivalence and Parikh vectors

Definition

Let w be a word over Σ. A Parikh vector $\mathcal{P}(w)$ counts for each letter $a \in \Sigma$ its number of occurrences in w.

$$w = \mathtt{a\,b\,b\,a\,c} \qquad \mathcal{P}(w) = (2,2,1)$$

Definition

Words u, w are *commutatively equivalent* if $\mathcal{P}(u) = \mathcal{P}(w)$.

$$\mathtt{a\,b\,b\,a\,c} \approx \mathtt{a\,c\,b\,a\,b} \qquad \mathtt{b\,a\,b} \not\approx \mathtt{a\,b\,a}$$

Abelian Periods

Definition

Let w be a word. An integer q is:

- a *full* Abelian period of w if w can be partitioned into commutatively equivalent factors of length q,

a	b	a	b	a	c	a	b	a	a	b	c	b	a	a	b

$$q = 8 \quad \mathcal{P} = (4, 3, 1)$$

Abelian Periods

Definition

Let w be a word. An integer q is:

- a *full* Abelian period of w if w can be partitioned into commutatively equivalent factors of length q,
- an Abelian period of w if q is a full Abelian period of some extension *to the right* of w,

a b a b a c	a b a a b c	b a a b a c

$$q = 6 \quad \mathcal{P} = (3, 2, 1)$$

Abelian Periods

Definition

Let w be a word. An integer q is:

- a *full* Abelian period of w if w can be partitioned into commutatively equivalent factors of length q,
- an Abelian period of w if q is a full Abelian period of some extension *to the right* of w,
- a *weak* Abelian period of w if q is a full Abelian period of some extension of w.

b c a b a | b a c a b | a a b c b | a a b b c

$$q = 5 \quad \mathcal{P} = (2, 2, 1)$$

Previous results

Year	Authors	Variant	Time complexity
2011	Fici et al.	weak	$O(n^2\sigma)$
2012	Fici et al.	standard	$O(n^2)$
		full	$O(n \log \log n)$
2013	Crochemore et al.	weak	$O(n^2)$

Previous results

Year	Authors	Variant	Time complexity
2011	Fici et al.	weak	$O(n^2\sigma)$
2012	Fici et al.	standard	$O(n^2)$
		full	$O(n \log\log n)$
2013	Crochemore et al.	weak	$O(n^2)$
2013	this work	standard	$O(n \log\log n)$ randomized
			$O(n \log\log n + n \log\sigma)$ deterministic
		full	$O(n)$

Previous results

Year	Authors	Variant	Time complexity
2011	Fici et al.	weak	$O(n^2\sigma)$
2012	Fici et al.	standard	$O(n^2)$
		full	$O(n \log \log n)$
2013	Crochemore et al.	weak	$O(n^2)$
2013	this work	standard	$O(n \log \log n)$ randomized
			$O(n \log \log n + n \log \sigma)$ deterministic
		full	$O(n)$

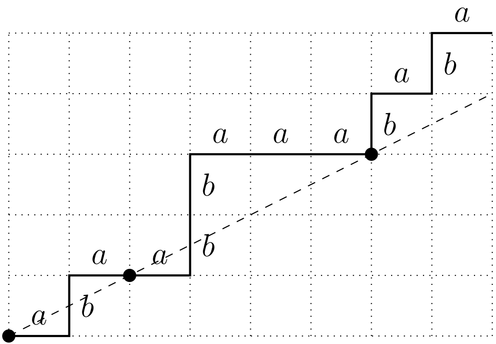
Assumptions:

- $\Sigma = \{1, \ldots, \sigma\}$
- standard RAM model
  (arrays, arithmetic of $O(\log n)$-bit integers)

Proportionality

Definition

Let $\mathcal{P}_i$ be the Parikh vector of $w[1..i]$. We write $i \sim j$ if there exists $c \in \mathbb{R}$ such that $\mathcal{P}_i[s] = c\mathcal{P}_j[s]$ for each $s \in \Sigma$.

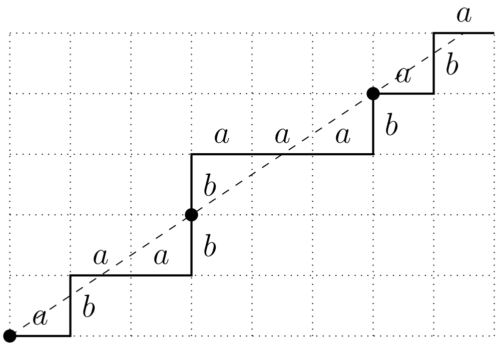


$3 \sim 9$

Proportionality

Definition

Let $\mathcal{P}_i$ be the Parikh vector of $w[1..i]$. We write $i \sim j$ if there exists $c \in \mathbb{R}$ such that $\mathcal{P}_i[s] = c\mathcal{P}_j[s]$ for each $s \in \Sigma$.



$5 \sim 10$

Efficient proportionality testing

Lemma

After $O(n)$ randomized or $O(n \log \sigma)$ deterministic time preprocessing $\sim$ can be tested in constant time.

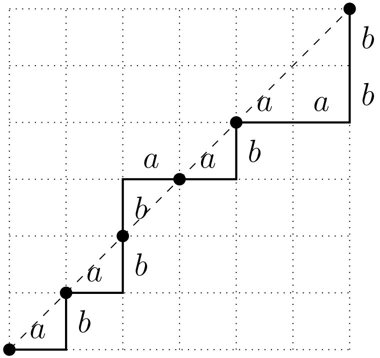
Fact

The set $[n]_\sim = \{k : k \sim n\}$ can be constructed in $O(n)$ time.

Full Abelian Periods

Fact

Let $A = \{k : k \sim n\}$. Then q is a full Abelian period $\iff$ there $q \mid k$ and $k \leq n$ implies $k \in A$.

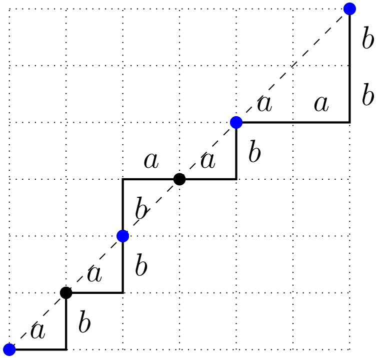


$$A = \{2, 4, 6, 8, 12\}$$

Full Abelian Periods

Fact

Let $A = \{k : k \sim n\}$. Then q is a full Abelian period $\iff$ there $q \mid k$ and $k \leq n$ implies $k \in A$.



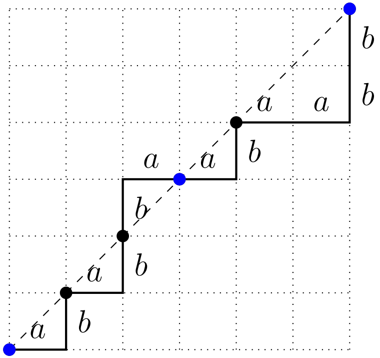
$$A = \{2, 4, 6, 8, 12\}$$

4 is a full Abelian period.

Full Abelian Periods

Fact

Let $A = \{k : k \sim n\}$. Then q is a full Abelian period $\iff$ there $q \mid k$ and $k \leq n$ implies $k \in A$.



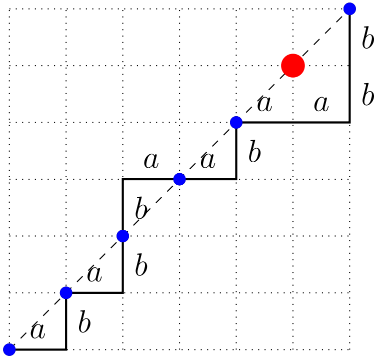
$$A = \{2, 4, 6, 8, 12\}$$

6 is a full Abelian period.

Full Abelian Periods

Fact

Let $A = \{k : k \sim n\}$. *Then* q *is a full Abelian period* $\iff$ *there* $q \mid k$ *and* $k \leq n$ *implies* $k \in A$.



$$A = \{2, 4, 6, 8, 12\}$$

2 is *not* a full Abelian period.

Full Abelian Periods

Fact

Let $A = \{k : k \sim n\}$. *Then* q *is a full Abelian period* $\iff$ *there* $q \mid k$ *and* $k \leq n$ *implies* $k \in A$.

Observation

There is no $k \notin A$ *such that* $q \mid k \iff$ *there is no* q' *such that* $q \mid q'$ *and* $q' = gcd(k, n)$ *for some* $k \notin A$.

Full Abelian Periods

Fact

Let $A = \{k : k \sim n\}$*. Then* q *is a full Abelian period* $\iff$ *there* $q \mid k$ *and* $k \leq n$ *implies* $k \in A$*.*

Observation

There is no $k \notin A$ *such that* $q \mid k \iff$ *there is no* q' *such that* $q \mid q'$ *and* $q' = gcd(k, n)$ *for some* $k \notin A$*.*

1. $A' := \{k : k \not\sim n\}$
2. $X := \{q' : \exists_{k \notin A}\, gcd(k, n) = q'\}$
   (iterating over $k \notin A$ and using fast gcd queries)
3. For each $q \mid n$ check whether there exists $q' \in X$ such that $q \mid q'$

Full Abelian Periods

Fact

Let $A = \{k : k \sim n\}$. Then q is a full Abelian period $\iff$ there $q \mid k$ and $k \leq n$ implies $k \in A$.

Observation

There is no $k \notin A$ such that $q \mid k \iff$ there is no q' such that $q \mid q'$ and $q' = gcd(k, n)$ for some $k \notin A$.

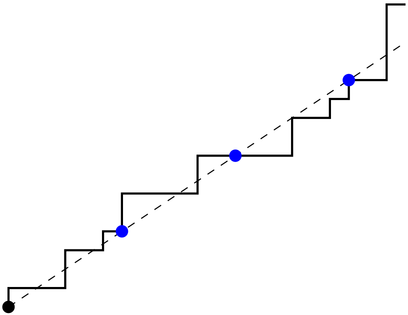
1. $A' := \{k : k \not\sim n\}$
2. $X := \{q' : \exists_{k \notin A}\, gcd(k, n) = q'\}$
   (iterating over $k \notin A$ and using fast gcd queries)
3. For each $q \mid n$ check whether there exists $q' \in X$ such that $q \mid q'$

The number of pairs (q, q') is $o(n)$, since the number of divisors of n is $o(n^{\varepsilon})$.

Standard Abelian Periods

Definition

A positive integer $q \le n$ is a *candidate* if $q \sim kq$ for each $k \in \left\{1, \ldots, \left\lfloor \frac{n}{q} \right\rfloor \right\}$.

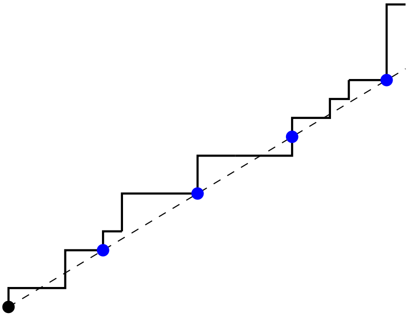


10 is a candidate

Standard Abelian Periods

Definition

A positive integer $q \le n$ is a *candidate* if $q \sim kq$ for each $k \in \left\{1, \dots, \left\lfloor \frac{n}{q} \right\rfloor\right\}$.

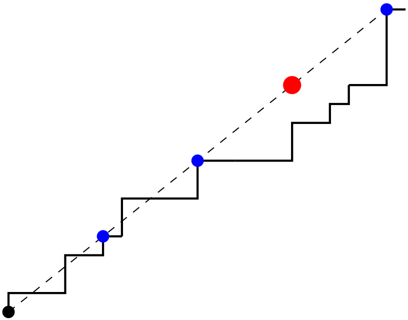


8 is a candidate

Standard Abelian Periods

Definition

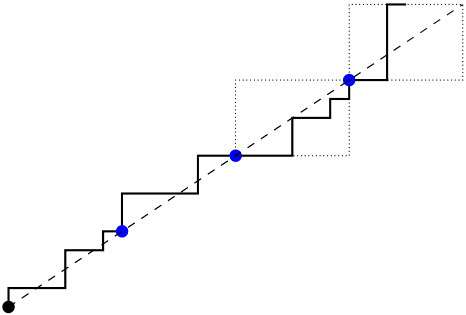
A positive integer $q \le n$ is a *candidate* if $q \sim kq$ for each $k \in \left\{1, \ldots, \left\lfloor \frac{n}{q} \right\rfloor\right\}$.



9 is not a candidate

Standard Abelian Periods

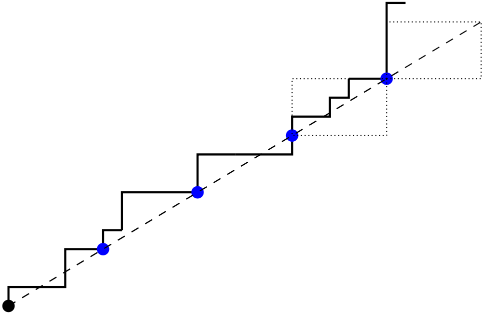
A simple application of the techniques from weak Abelian periods algorithm gives an $O(n)$ time algorithm computing the set of Abelian periods given the set of candidates.



10 is an Abelian period

Standard Abelian Periods

A simple application of the techniques from weak Abelian periods algorithm gives an $O(n)$ time algorithm computing the set of Abelian periods given the set of candidates.



8 is not an Abelian period

Computing candidates

Lemma

The set $\mathcal{C}$ of all candidates can be computed in $O(n \log \log n)$ time provided that $\sim$ can be tested in constant time.

Computing candidates

Lemma

The set $\mathcal{C}$ of all candidates can be computed in $O(n \log\log n)$ time provided that $\sim$ can be tested in constant time.

Observation

$$q \in \mathcal{C} \iff \forall_{k \in \mathbb{Z}_+ \,:\, kq \le n}\; q \sim kq \iff \forall_{\,p \in Primes \,:\, pq \le n}\; (q \sim pq \;\wedge pq \in \mathcal{C}).$$

Recall that primes up to n can be generated in $O(n)$ time.

Computing candidates

Lemma

The set $\mathcal{C}$ of all candidates can be computed in $O(n \log \log n)$ time provided that $\sim$ can be tested in constant time.

Observation

$$q \in \mathcal{C} \iff \forall_{k \in \mathbb{Z}_+ : kq \leq n} \; q \sim kq \iff \forall_{p \in Primes : pq \leq n} \; (q \sim pq \;\wedge pq \in \mathcal{C}).$$

Recall that primes up to n can be generated in $O(n)$ time.
A fixed $p \in Primes$ is processed for at most $\frac{n}{p}$ values of q, so the total number of operations is bounded by

$$\sum_{p \in Primes,\, p \leq n} \frac{n}{p} = O(n \log \log n).$$

Conclusions

Theorem

Let w be a word of length n over the alphabet $\{1, \ldots, \sigma\}$. Full Abelian periods of w can be computed in $O(n)$ time.

Theorem

Let w be a word of length n over the alphabet $\{1, \ldots, \sigma\}$. There exist an $O(n \log \log n + n \log \sigma)$ time deterministic and an $O(n \log \log n)$ time randomized algorithm that compute all Abelian periods of w. Both algorithms require $O(n)$ space.

Thank you

Thank you for your attention!