Deterministic Root Counting Modulo Prime Powers

Ashish Dwivedi
IITK

Joint work with
Rajat Mittal (IITK) and Nitin Saxena (IITK)

ICTS, WACT' 19

Overview

Overview

Introduction

Factoring or root finding of a univariate modulo a prime is well studied problem!

Introduction

Factoring or root finding of a univariate modulo a prime is well studied problem!

Many efficient randomized algorithms are known.

Introduction

Factoring or root finding of a univariate modulo a prime is well studied problem!

Many efficient randomized algorithms are known.

Open: A deterministic poly-time algorithm?

Introduction

Factoring or root finding of a univariate modulo a prime is well studied problem!

Many efficient randomized algorithms are known.

Open: A deterministic poly-time algorithm?

Deterministic algorithm is known only for irreducibility testing or more strongly root counting.

Introduction

Factoring or root finding of a univariate modulo a prime is well studied problem!

Many efficient randomized algorithms are known.

Open: A deterministic poly-time algorithm?

Deterministic algorithm is known only for irreducibility testing or more strongly root counting.

What about factoring modulo a composite n?
(given prime factorization of n)

Factoring or root finding of a univariate modulo a prime is well studied problem!

Many efficient randomized algorithms are known.

Open: A deterministic poly-time algorithm?

Deterministic algorithm is known only for irreducibility testing or more strongly root counting.

What about factoring modulo a composite n?
(given prime factorization of n)

It reduces to factoring modulo a prime power p^k. (CRT)

Introduction

Getting roots mod p^k

Getting roots mod p^k

For roots of multiplicity 1 of f mod p, Hensel's lifting guarantees unique lift mod p^k.

Getting roots mod p^k

For roots of multiplicity 1 of f mod p, Hensel's lifting guarantees unique lift mod p^k.

Eg. Given $f(x) = x^2 + 10x + 21 = (x+3)(x+7)$ and $p = 3$.

Getting roots mod p^k

For roots of multiplicity 1 of f mod p, Hensel's lifting guarantees unique lift mod p^k.

Eg. Given $f(x) = x^2 + 10x + 21 = (x+3)(x+7)$ and $p = 3$.

$f \equiv x(x+1) \bmod 3$.

Introduction

Getting roots mod p^k

For roots of multiplicity 1 of f mod p, Hensel's lifting guarantees unique lift mod p^k.

Eg. Given $f(x) = x^2 + 10x + 21 = (x+3)(x+7)$ and $p = 3$.

$f \equiv x(x+1) \bmod 3$.

Using Hensel's lemma,

$f(x) \equiv (x+3)(x+1+6) \bmod 3^2 \Rightarrow f \equiv (x+3)(x+7) \bmod 3^2$.

Getting roots mod p^k

For roots of multiplicity 1 of f mod p, Hensel's lifting guarantees unique lift mod p^k.

Eg. Given $f(x) = x^2 + 10x + 21 = (x+3)(x+7)$ and $p = 3$.

$f \equiv x(x+1) \bmod 3$.

Using Hensel's lemma,

$f(x) \equiv (x+3)(x+1+6) \bmod 3^2 \Rightarrow f \equiv (x+3)(x+7) \bmod 3^2$.

This keeps on lifting for any power 3^k.

Introduction

First issue:

Introduction

First issue:

Multiplicity > 1?

Introduction

First issue:

Multiplicity > 1? Hensel lifting fails!

Introduction

First issue:

Multiplicity > 1? Hensel lifting fails!

Hensel's lifting requires co-prime factors, otherwise lifting could be non-unique or no lift at all.

Introduction

First issue:

Multiplicity > 1? Hensel lifting fails!

Hensel's lifting requires co-prime factors, otherwise lifting could be non-unique or no lift at all.

The hard case is- $f(x) \equiv (x - a)^e \bmod p$!

Introduction

First issue:

Multiplicity > 1? Hensel lifting fails!

Hensel's lifting requires co-prime factors, otherwise lifting could be non-unique or no lift at all.

The hard case is- $f(x) \equiv (x-a)^e \bmod p$!

Second issue:

First issue:

Multiplicity > 1? Hensel lifting fails!

Hensel's lifting requires co-prime factors, otherwise lifting could be non-unique or no lift at all.

The hard case is- $f(x) \equiv (x - a)^e \bmod p$!

Second issue:

The coefficient ring $\mathbb{Z}/\langle p^k \rangle$ is not a unique factorization domain!

Introduction

First issue:

Multiplicity > 1? Hensel lifting fails!

Hensel's lifting requires co-prime factors, otherwise lifting could be non-unique or no lift at all.

The hard case is- $f(x) \equiv (x-a)^e \bmod p$!

Second issue:

The coefficient ring $\mathbb{Z}/\langle p^k \rangle$ is not a unique factorization domain!

Exponentially many factors.

Introduction

First issue:

Multiplicity > 1? Hensel lifting fails!

Hensel's lifting requires co-prime factors, otherwise lifting could be non-unique or no lift at all.

The hard case is- $f(x) \equiv (x - a)^e \bmod p$!

Second issue:

The coefficient ring $\mathbb{Z}/\langle p^k \rangle$ is not a unique factorization domain!

Exponentially many factors.

Eg. $x^2 + px \bmod p^2$.

Introduction

First issue:

Multiplicity > 1?   Hensel lifting fails!

Hensel's lifting requires co-prime factors, otherwise lifting could be non-unique or no lift at all.

The hard case is- $f(x) \equiv (x - a)^e \bmod p$!

Second issue:

The coefficient ring $\mathbb{Z}/\langle p^k \rangle$ is not a unique factorization domain!

Exponentially many factors.

Eg. $x^2 + px \bmod p^2$. $(x + p\alpha)$ is a factor for all $\alpha \in \mathbb{F}_p$.

Introduction

First issue:

Multiplicity > 1? Hensel lifting fails!

Hensel's lifting requires co-prime factors, otherwise lifting could be non-unique or no lift at all.

The hard case is- $f(x) \equiv (x - a)^e \bmod p$!

Second issue:

The coefficient ring $\mathbb{Z}/\langle p^k \rangle$ is not a unique factorization domain!

Exponentially many factors.

Eg. $x^2 + px \bmod p^2$. $(x + p\alpha)$ is a factor for all $\alpha \in \mathbb{F}_p$.

It becomes non-trivial to find or even count all the factors.

Introduction

Let us see an example of Hensel lifting failure.

Introduction

Let us see an example of Hensel lifting failure.

Eg. $f(x) = x^3 + 12x^2 + 3x + 36$ and $p^k = 3^3$.

Let us see an example of Hensel lifting failure.

Eg. $f(x) = x^3 + 12x^2 + 3x + 36$ and $p^k = 3^3$.

$f(x) \equiv x^3 \mod 3$. Take the factorization $f \equiv x.x^2 \mod 3$.

Introduction

Let us see an example of Hensel lifting failure.

Eg. $f(x) = x^3 + 12x^2 + 3x + 36$ and $p^k = 3^3$.

$f(x) \equiv x^3 \mod 3$. Take the factorization $f \equiv x.x^2 \mod 3$.

It lifts to mod 3^2 as $x(x^2 + 3x + 3)$, $(x + 6)(x^2 + 6x + 3)$ and $(x + 3)(x^2 + 3)$.

Introduction

Let us see an example of Hensel lifting failure.

Eg. $f(x) = x^3 + 12x^2 + 3x + 36$ and $p^k = 3^3$.

$f(x) \equiv x^3 \mod 3$. Take the factorization $f \equiv x.x^2 \mod 3$.

It lifts to mod 3^2 as $x(x^2 + 3x + 3)$, $(x + 6)(x^2 + 6x + 3)$ and $(x + 3)(x^2 + 3)$.

Only the last factorization lifts to mod 3^3 as

Introduction

Let us see an example of Hensel lifting failure.

Eg. $f(x) = x^3 + 12x^2 + 3x + 36$ and $p^k = 3^3$.

$f(x) \equiv x^3 \mod 3$. Take the factorization $f \equiv x.x^2 \mod 3$.

It lifts to mod 3^2 as $x(x^2 + 3x + 3)$, $(x + 6)(x^2 + 6x + 3)$ and $(x + 3)(x^2 + 3)$.

Only the last factorization lifts to mod 3^3 as

$(x + 3)(x^2 + 9x + 3)$, $(x + 12)(x^2 + 3)$ and $(x + 21)(x^2 + 18x + 3)$.

Introduction

Let us see an example of Hensel lifting failure.

Eg. $f(x) = x^3 + 12x^2 + 3x + 36$ and $p^k = 3^3$.

$f(x) \equiv x^3 \mod 3$. Take the factorization $f \equiv x.x^2 \mod 3$.

It lifts to mod 3^2 as $x(x^2 + 3x + 3)$, $(x + 6)(x^2 + 6x + 3)$ and $(x + 3)(x^2 + 3)$.

Only the last factorization lifts to mod 3^3 as

$(x + 3)(x^2 + 9x + 3)$, $(x + 12)(x^2 + 3)$ and $(x + 21)(x^2 + 18x + 3)$.

Not every factorization lifts.

Introduction

Let us see an example of Hensel lifting failure.

Eg. $f(x) = x^3 + 12x^2 + 3x + 36$ and $p^k = 3^3$.

$f(x) \equiv x^3 \mod 3$. Take the factorization $f \equiv x.x^2 \mod 3$.

It lifts to mod 3^2 as $x(x^2 + 3x + 3)$, $(x + 6)(x^2 + 6x + 3)$ and $(x + 3)(x^2 + 3)$.

Only the last factorization lifts to mod 3^3 as

$(x + 3)(x^2 + 9x + 3)$, $(x + 12)(x^2 + 3)$ and $(x + 21)(x^2 + 18x + 3)$.

Not every factorization lifts.

Which one will lift?

Introduction

Let us see an example of Hensel lifting failure.

Eg. $f(x) = x^3 + 12x^2 + 3x + 36$ and $p^k = 3^3$.

$f(x) \equiv x^3 \mod 3$. Take the factorization $f \equiv x.x^2 \mod 3$.

It lifts to mod 3^2 as $x(x^2 + 3x + 3)$, $(x + 6)(x^2 + 6x + 3)$ and $(x + 3)(x^2 + 3)$.

Only the last factorization lifts to mod 3^3 as

$(x + 3)(x^2 + 9x + 3)$, $(x + 12)(x^2 + 3)$ and $(x + 21)(x^2 + 18x + 3)$.

Not every factorization lifts.

Which one will lift? Exponential time by direct search.

Introduction

Gathen and Hartlieb [1996] showed that when p^k is large, factorizations are nicely connected with unique p-adic factorization.

Introduction

Gathen and Hartlieb [1996] showed that when p^k is large, factorizations are nicely connected with unique p-adic factorization.

They also gave example that factors are not always nicely connected.

Introduction

Gathen and Hartlieb [1996] showed that when p^k is large, factorizations are nicely connected with unique p-adic factorization.

They also gave example that factors are not always nicely connected.

Eg. $f = (x^2 + 243)(x^2 + 6) \bmod 3^6$ an irreducible factorization.

Introduction

Gathen and Hartlieb [1996] showed that when p^k is large, factorizations are nicely connected with unique p-adic factorization.

They also gave example that factors are not always nicely connected.

Eg. $f = (x^2 + 243)(x^2 + 6) \bmod 3^6$ an irreducible factorization.

A completely unrelated irreducible factorization:

$f = (x + 351)(x + 135)(x^2 + 243x + 249) \bmod 3^6$.

Overview

The Problem

Input: a univariate $f(x) \in \mathbb{Z}[x]$ and a prime power p^k (in bits).

The Problem

Input: a univariate $f(x) \in \mathbb{Z}[x]$ and a prime power p^k (in bits).

Output: Find and count exactly the roots of $f \bmod p^k$.

The Problem

Input: a univariate $f(x) \in \mathbb{Z}[x]$ and a prime power p^k (in bits).

Output: Find and count exactly the roots of $f \bmod p^k$.

There could be p^k many roots of $f \bmod p^k$, which is exponential in input size.

The Problem

Input: a univariate $f(x) \in \mathbb{Z}[x]$ and a prime power p^k (in bits).

Output: Find and count exactly the roots of $f \bmod p^k$.

There could be p^k many roots of $f \bmod p^k$, which is exponential in input size.

Open: A deterministic polynomial time algorithm to exactly count the roots of $f \bmod p^k$?

The Problem

Input: a univariate $f(x) \in \mathbb{Z}[x]$ and a prime power p^k (in bits).

Output: Find and count exactly the roots of $f \bmod p^k$.

There could be p^k many roots of $f \bmod p^k$, which is exponential in input size.

Open: A deterministic polynomial time algorithm to exactly count the roots of $f \bmod p^k$?

The root counting problem is stronger than just showing the existence of a root.

The Problem

Input: a univariate $f(x) \in \mathbb{Z}[x]$ and a prime power p^k (in bits).

Output: Find and count exactly the roots of $f \bmod p^k$.

There could be p^k many roots of $f \bmod p^k$, which is exponential in input size.

Open: A deterministic polynomial time algorithm to exactly count the roots of $f \bmod p^k$?

The root counting problem is stronger than just showing the existence of a root.

Extension to count irreducible factors will give irreducibility criteria.

Overview

Our Results

We give an algorithm to count roots that runs in deterministic poly-time.

Our Results

We give an algorithm to count roots that runs in deterministic poly-time.

We will do more-

Our Results

We give an algorithm to count roots that runs in deterministic poly-time.

We will do more- A Structural Result.

Our Results

We give an algorithm to count roots that runs in deterministic poly-time.

We will do more- A Structural Result.

The root set partitions into at most $\deg(f)$ many subsets of easily computable size.

Our Results

We give an algorithm to count roots that runs in deterministic poly-time.

We will do more- A Structural Result.

The root set partitions into at most $\deg(f)$ many subsets of easily computable size.

It is similar to the property shown by a univariate over fields.

Overview

Randomized Algorithm

Derandomization question for $\mathbb{F}_p$ is open.

Randomized Algorithm

Derandomization question for $\mathbb{F}_p$ is open.

Even for $\mathbb{Z}/\langle p^k \rangle$, for a long time, even a randomized poly-time algorithm was not known.

Randomized Algorithm

Derandomization question for $\mathbb{F}_p$ is open.

Even for $\mathbb{Z}/\langle p^k \rangle$, for a long time, even a randomized poly-time algorithm was not known.

Finally Berthomieu, Lecerf and Quintin (2013) gave the first randomized poly-time algorithm to find (& count) all the roots of f mod p^k.

Randomized Algorithm: Framework

[BLQ' 13] uses randomized algorithm mod p repeatedly as a black-box (eg. Cantor-Zassenhaus).

Randomized Algorithm: Framework

[BLQ' 13] uses randomized algorithm mod p repeatedly as a black-box (eg. Cantor-Zassenhaus).

Based on the fact: any root mod p^k is a lift of some root mod p^l for all $l \leq k$.

Randomized Algorithm: Framework

[BLQ' 13] uses randomized algorithm mod p repeatedly as a black-box (eg. Cantor-Zassenhaus).

Based on the fact: any root mod p^k is a lift of some root mod p^l for all $l \leq k$.

Visualize roots in series form- $\boxed{r = r_0 + pr_1 + \ldots + p^{k-1} r_{k-1}}$

Randomized Algorithm: Framework

[BLQ' 13] uses randomized algorithm mod p repeatedly as a black-box (eg. Cantor-Zassenhaus).

Based on the fact: any root mod p^k is a lift of some root mod p^l for all $l \leq k$.

Visualize roots in series form- $\boxed{r = r_0 + pr_1 + \ldots + p^{k-1} r_{k-1}}$

So r is a lift of $r_0 \bmod p$, $r_0 + pr_1 \bmod p^2$ and so on.

Randomized Algorithm: Framework

[BLQ' 13] uses randomized algorithm mod p repeatedly as a black-box (eg. Cantor-Zassenhaus).

Based on the fact: any root mod p^k is a lift of some root mod p^l for all $l \leq k$.

Visualize roots in series form- $\boxed{r = r_0 + pr_1 + \ldots + p^{k-1} r_{k-1}}$

So r is a lift of $r_0 \bmod p$, $r_0 + pr_1 \bmod p^2$ and so on.

Idea

So the approach is to find each r_i one by one using the CZ algorithm to incrementally build up the lifts of r_0 with higher and higher precision leading up to r.

Randomized Algorithm: Notation

If $p^\alpha | f(x) \bmod p^k$ then any root $r = r_0 + pr_1 + \ldots + p^{k-1} r_{k-1}$ is independent of $r_{k-\alpha}, \ldots, r_{k-1}$.

Randomized Algorithm: Notation

If $p^{\alpha}|f(x) \bmod p^k$ then any root $r = r_0 + pr_1 + \ldots + p^{k-1}r_{k-1}$ is independent of $r_{k-\alpha}, \ldots, r_{k-1}$.

In other words $r = r_0 + pr_1 + \ldots + p^{k-\alpha-1}r_{k-\alpha-1} + p^{k-\alpha} * + \ldots + p^{k-1}*$, where $*$ denotes everything in $\mathbb{F}_p$.

Randomized Algorithm: Notation

If $p^\alpha | f(x) \bmod p^k$ then any root $r = r_0 + pr_1 + \ldots + p^{k-1} r_{k-1}$ is independent of $r_{k-\alpha}, \ldots, r_{k-1}$.

In other words $r = r_0 + pr_1 + \ldots + p^{k-\alpha-1} r_{k-\alpha-1} + p^{k-\alpha} * + \ldots + p^{k-1} *$, where $*$ denotes everything in $\mathbb{F}_p$.

Representative roots

In short we write $r = r_0 + pr_1 + \ldots + p^{k-\alpha} *$, where r is called a **representative root** representing $p^{k-\alpha}$ 'distinct' roots of f mod p^k.

Randomized Algorithm: Notation

If $p^\alpha | f(x) \bmod p^k$ then any root $r = r_0 + pr_1 + \ldots + p^{k-1} r_{k-1}$ is independent of $r_{k-\alpha}, \ldots, r_{k-1}$.

In other words $r = r_0 + pr_1 + \ldots + p^{k-\alpha-1} r_{k-\alpha-1} + p^{k-\alpha} * + \ldots + p^{k-1} *$, where $*$ denotes everything in $\mathbb{F}_p$.

Representative roots

In short we write $r = r_0 + pr_1 + \ldots + p^{k-\alpha} *$, where r is called a **representative root** representing $p^{k-\alpha}$ 'distinct' roots of $f \bmod p^k$.

The randomized algorithm will return **all** the roots in representative form- at most $\deg(f)$ many!

Randomized Algorithm

Find co-ordinates r_i one by one by applying CZ mod p in every cycle of shift-divide operations.

Randomized Algorithm

Find co-ordinates r_i one by one by applying CZ mod p in every cycle of shift-divide operations.

To get candidates for r_0 apply CZ on $f(x) \bmod p$.

Randomized Algorithm

Find co-ordinates r_i one by one by applying CZ mod p in every cycle of shift-divide operations.

To get candidates for r_0 apply CZ on $f(x) \bmod p$.

For every r_0 obtained do the following:

Randomized Algorithm

Find co-ordinates r_i one by one by applying CZ mod p in every cycle of shift-divide operations.

To get candidates for r_0 apply CZ on $f(x) \bmod p$.

For every r_0 obtained do the following:

Shift: $f(x) \mapsto f_{r_0}(x)$, $f_{r_0}(x) := f(r_0 + px)$

Randomized Algorithm

Find co-ordinates r_i one by one by applying CZ mod p in every cycle of shift-divide operations.

To get candidates for r_0 apply CZ on $f(x) \bmod p$.

For every r_0 obtained do the following:

Shift: $f(x) \mapsto f_{r_0}(x)$, $f_{r_0}(x) := f(r_0 + px)$

Divide: Get $g(x) = f(r_0 + px)/p^{\alpha} \bmod p^{k-\alpha}$ where $p^{\alpha} || f(r_0 + px)$.

Randomized Algorithm

Find co-ordinates r_i one by one by applying CZ mod p in every cycle of shift-divide operations.

To get candidates for r_0 apply CZ on $f(x) \bmod p$.

For every r_0 obtained do the following:

Shift: $f(x) \mapsto f_{r_0}(x)$, $f_{r_0}(x) := f(r_0 + px)$

Divide: Get $g(x) = f(r_0 + px)/p^{\alpha} \bmod p^{k-\alpha}$ where $p^{\alpha} || f(r_0 + px)$.

Repeat the process on $g(x) \bmod p^{k-\alpha}$.

Randomized Algorithm

Find co-ordinates r_i one by one by applying CZ mod p in every cycle of shift-divide operations.

To get candidates for r_0 apply CZ on $f(x) \bmod p$.

For every r_0 obtained do the following:

Shift: $f(x) \mapsto f_{r_0}(x)$, $f_{r_0}(x) := f(r_0 + px)$

Divide: Get $g(x) = f(r_0 + px)/p^{\alpha} \bmod p^{k-\alpha}$ where $p^{\alpha} || f(r_0 + px)$.

Repeat the process on $g(x) \bmod p^{k-\alpha}$.

Essentially every iteration reduces finding roots of $f(x) \bmod p^k$ to roots of $g(x) \bmod p^{k-\alpha}$.

Randomized Algorithm: Correctness

Recall $f_{r_0}(x) := f(r_0 + px)$ and $g(x) = f(r_0 + px)/p^\alpha \bmod p^{k-\alpha}$.

Randomized Algorithm: Correctness

Recall $f_{r_0}(x) := f(r_0 + px)$ and $g(x) = f(r_0 + px)/p^{\alpha} \bmod p^{k-\alpha}$.

Lemma

For any root r' of $g \bmod p^{k-\alpha}$ the corresponding roots of $f \bmod p^k$ are: $r_0 + p(r' + p^{k-\alpha} *)$

Randomized Algorithm: Correctness

Recall $f_{r_0}(x) := f(r_0 + px)$ and $g(x) = f(r_0 + px)/p^{\alpha} \bmod p^{k-\alpha}$.

Lemma

For any root r' of $g \bmod p^{k-\alpha}$ the corresponding roots of $f \bmod p^k$ are: $r_0 + p(r' + p^{k-\alpha}*)$

The proof follows by following two claims:

Randomized Algorithm: Correctness

Recall $f_{r_0}(x) := f(r_0 + px)$ and $g(x) = f(r_0 + px)/p^{\alpha} \bmod p^{k-\alpha}$.

Lemma

For any root r' of $g \bmod p^{k-\alpha}$ the corresponding roots of $f \bmod p^k$ are: $r_0 + p(r' + p^{k-\alpha} *)$

The proof follows by following two claims:

Claim: $r = r_0 + pr'$ is a root of $f \bmod p^k$ iff r' is a root of $f_{r_0} \bmod p^k$.

Claim: r' is a root of $f_{r_0} \bmod p^k$ iff r' is a root of $g(x) \bmod p^{k-\alpha}$.

Randomized Algorithm: Correctness

Recall $f_{r_0}(x) := f(r_0 + px)$ and $g(x) = f(r_0 + px)/p^{\alpha} \bmod p^{k-\alpha}$.

Lemma

For any root r' of $g \bmod p^{k-\alpha}$ the corresponding roots of $f \bmod p^k$ are: $r_0 + p(r' + p^{k-\alpha} *)$

The proof follows by following two claims:

Claim: $r = r_0 + pr'$ is a root of $f \bmod p^k$ iff r' is a root of $f_{r_0} \bmod p^k$.

Claim: r' is a root of $f_{r_0} \bmod p^k$ iff r' is a root of $g(x) \bmod p^{k-\alpha}$.

Always $\alpha \geq 1$, so the process stops in at most k iterations.

Randomized Algorithm: Time Complexity

The time taken could be very high?

Randomized Algorithm: Time Complexity

The time taken could be very high? $\deg(f)^k$ many roots in the end?

Randomized Algorithm: Time Complexity

The time taken could be very high? $\deg(f)^k$ many roots in the end?

The algorithm forms a virtual tree of roots:

- An edge at i-th level represents i-th co-ordinate of some root.
- $g(x)$ defined as before denote a node in the tree.

Randomized Algorithm: Time Complexity

The time taken could be very high? $\deg(f)^k$ many roots in the end?

The algorithm forms a virtual tree of roots:

- An edge at i-th level represents i-th co-ordinate of some root.
- $g(x)$ defined as before denote a node in the tree.

Lemma: A path from root to a leaf denotes a representative-root of f.
The tree has at most d leaves.

Randomized Algorithm: Time Complexity

The time taken could be very high? $\deg(f)^k$ many roots in the end?

The algorithm forms a virtual tree of roots:

- An edge at i-th level represents i-th co-ordinate of some root.
- $g(x)$ defined as before denote a node in the tree.

Lemma: A path from root to a leaf denotes a representative-root of f. The tree has at most d leaves.

Claim: The degree of a node distributes to its children.

Randomized Algorithm: Time Complexity

The time taken could be very high? $\deg(f)^k$ many roots in the end?

The algorithm forms a virtual tree of roots:

- An edge at i-th level represents i-th co-ordinate of some root.
- $g(x)$ defined as before denote a node in the tree.

Lemma: A path from root to a leaf denotes a representative-root of f. The tree has at most d leaves.

Claim: The degree of a node distributes to its children.

Proof: let a be a root of multiplicity m of $g(x)$ mod p then the degree of children corresponding to a is at most m.

Randomized Algorithm: Time Complexity

The time taken could be very high? $\deg(f)^k$ many roots in the end?

The algorithm forms a virtual tree of roots:

- An edge at i-th level represents i-th co-ordinate of some root.
- $g(x)$ defined as before denote a node in the tree.

Lemma: A path from root to a leaf denotes a representative-root of f. The tree has at most d leaves.

Claim: The degree of a node distributes to its children.

Proof: let a be a root of multiplicity m of $g(x) \bmod p$ then the degree of children corresponding to a is at most m.

Hence, the degree of $f(x)$ (root) inductively distributes to leaves which have degree at least 1.

Overview

How to derandomize the root counting problem?

Derandomization

How to derandomize the root counting problem?

Last year Cheng, Gao, Rojas, Wan [ANTS' 18] partially derandomized in time exponential in the parameter k.

Derandomization

How to derandomize the root counting problem?

Last year Cheng, Gao, Rojas, Wan [ANTS' 18] partially derandomized in time exponential in the parameter k.

We give the first deterministic poly($d, k \log p$) time algorithm to count the roots. A complete derandomization.

Derandomization

In fact, we consider the general question of root finding and efficiently construct a list data structure $\mathcal{L}$.

Derandomization

In fact, we consider the general question of root finding and efficiently construct a list data structure $\mathcal{L}$.

$\mathcal{L}$ is a list of Split Ideals, having two explicit parameters length and degree.

Derandomization

In fact, we consider the general question of root finding and efficiently construct a list data structure $\mathcal{L}$.

$\mathcal{L}$ is a list of Split Ideals, having two explicit parameters length and degree.

Ideals in $\mathcal{L}$ collectively store exactly the roots of $f \bmod p^k$ partitioning roots into at most d sets.

Derandomization

In fact, we consider the general question of root finding and efficiently construct a list data structure $\mathcal{L}$.

$\mathcal{L}$ is a list of Split Ideals, having two explicit parameters length and degree.

Ideals in $\mathcal{L}$ collectively store exactly the roots of $f \bmod p^k$ partitioning roots into at most d sets.

Each split ideal $I(n, D)$ implicitly stores a subset of roots- $D \times p^{k-n}$.

Derandomization

In fact, we consider the general question of root finding and efficiently construct a list data structure $\mathcal{L}$.

$\mathcal{L}$ is a list of Split Ideals, having two explicit parameters length and degree.

Ideals in $\mathcal{L}$ collectively store exactly the roots of $f \bmod p^k$ partitioning roots into at most d sets.

Each split ideal $I(n, D)$ implicitly stores a subset of roots- $D \times p^{k-n}$.

Our result can be seen as a deterministic poly-time reduction to root finding mod p.

Overview

Deterministic Algorithm

Can not apply Cantor-Zassenhaus!

Deterministic Algorithm

Can not apply Cantor-Zassenhaus!

Intermediate roots are not available!

Deterministic Algorithm

Can not apply Cantor-Zassenhaus!

Intermediate roots are not available!

Shifting same way is not possible!

Deterministic Algorithm

Can not apply Cantor-Zassenhaus!

Intermediate roots are not available!

Shifting same way is not possible!

Needs a different perspective.

Deterministic Algorithm: Tool 1

A shift $g(x) \mapsto g(a + px)$ is equivalent to $g(x_0 + px) \bmod \langle x_0 - a \rangle$.

Deterministic Algorithm: Tool 1

A shift $g(x) \mapsto g(a + px)$ is equivalent to $g(x_0 + px) \bmod \langle x_0 - a \rangle$.

Similarly,

$g(a + px) \mapsto g(a + pb + p^2 x) \Leftrightarrow g(x_0 + px_1 + p^2 x) \bmod \langle x_0 - a, x_1 - b \rangle$.

Deterministic Algorithm: Tool 1

A shift $g(x) \mapsto g(a+px)$ is equivalent to $g(x_0+px)$ mod $\langle x_0 - a \rangle$.

Similarly,

$g(a+px) \mapsto g(a+pb+p^2x) \Leftrightarrow g(x_0+px_1+p^2x)$ mod $\langle x_0 - a, x_1 - b \rangle$.

So we consider the representation- $x \to x_0 + px_1 + \ldots + p^{k-1}x_{k-1}$.

Deterministic Algorithm: Tool 2

Given $g(x)$ mod p, how can we count the roots of g?

Deterministic Algorithm: Tool 2

Given $g(x)$ mod p, how can we count the roots of g?

Apply Polynomial Method:

$$h(x) := (g(x), x^p - x) \bmod p$$

Deterministic Algorithm: Tool 2

Given $g(x)$ mod p, how can we count the roots of g?

Apply Polynomial Method:

$$h(x) := (g(x), x^p - x) \bmod p$$

$h(x)$ implicitly stores all the roots of g. The degree of h gives count!

Deterministic Algorithm

Let I be a split ideal as $I = \bigcap \langle x_0 - a_0, x_1 - a_1, \ldots, x_l - a_l \rangle$.

Deterministic Algorithm

Let I be a split ideal as $I = \bigcap \langle x_0 - a_0, x_1 - a_1, \ldots, x_l - a_l \rangle$.

The reduction $f(x_0 + px_1 + \ldots + p^l x_l + p^{l+1}x) \bmod I$ can be seen as performing shift by all the roots $\bar{a}$ of I simultaneously (CRT).

Deterministic Algorithm

Let I be a split ideal as $I = \bigcap \langle x_0 - a_0, x_1 - a_1, \ldots, x_l - a_l \rangle$.

The reduction $f(x_0 + px_1 + \ldots + p^l x_l + p^{l+1} x) \bmod I$ can be seen as performing shift by all the roots $\bar{a}$ of I simultaneously (CRT).

We will not have access to individual roots but we can construct such an ideal.

Deterministic Algorithm

Let I be a split ideal as $I = \bigcap \langle x_0 - a_0, x_1 - a_1, \ldots, x_l - a_l \rangle$.

The reduction $f(x_0 + px_1 + \ldots + p^l x_l + p^{l+1}x) \bmod I$ can be seen as performing shift by all the roots $\bar{a}$ of I simultaneously (CRT).

We will not have access to individual roots but we can construct such an ideal.

And perform GCD modulo such an ideal.

Deterministic Algorithm

Let I be a split ideal as $I = \bigcap \langle x_0 - a_0, x_1 - a_1, \ldots, x_l - a_l \rangle$.

The reduction $f(x_0 + px_1 + \ldots + p^l x_l + p^{l+1} x) \bmod I$ can be seen as performing shift by all the roots $\bar{a}$ of I simultaneously (CRT).

We will not have access to individual roots but we can construct such an ideal.

And perform GCD modulo such an ideal.

Using the tools described before we will construct split ideals of triangular form- $I = \langle h_0(x_0), h_1(x_1), \ldots, h_n(x_n) \rangle$.

Deterministic Algorithm

Let I be a split ideal as $I = \bigcap \langle x_0 - a_0, x_1 - a_1, \ldots, x_l - a_l \rangle$.

The reduction $f(x_0 + px_1 + \ldots + p^l x_l + p^{l+1}x) \bmod I$ can be seen as performing shift by all the roots $\bar{a}$ of I simultaneously (CRT).

We will not have access to individual roots but we can construct such an ideal.

And perform GCD modulo such an ideal.

Using the tools described before we will construct split ideals of triangular form- $I = \langle h_0(x_0), h_1(x_1), \ldots, h_n(x_n) \rangle$.

They implicitly store all the roots of $f \bmod p^k$.

Time Complexity

Why is the algorithm efficient?

Time Complexity

Why is the algorithm efficient?

The process as forms a virtual Root tree:

- Edge at level i is labelled by some $h_i(\bar{x}_i)$.
- A node denote the split ideal generated by edge labels on the path going to root.
- Leaves either denote maximal split ideal or a dead end.

Time Complexity

Why is the algorithm efficient?

The process as forms a virtual Root tree:

- Edge at level i is labelled by some $h_i(\bar{x}_i)$.
- A node denote the split ideal generated by edge labels on the path going to root.
- Leaves either denote maximal split ideal or a dead end.

Consider a Node N labelled by spit ideal I.

For all $\bar{a} \in \mathcal{Z}(I)$, $[N] := \deg(I) \times$ degree of the node $N_{\bar{a}}$ in [BLQ' 13] tree.

Time Complexity

Degree of a node distributes to degree of its children.

Time Complexity

Degree of a node distributes to degree of its children.

Inductively, it yields that degree of root $\deg(f)$ is at least sum of the degrees of the leaves.

Time Complexity

Degree of a node distributes to degree of its children.

Inductively, it yields that degree of root $\deg(f)$ is at least sum of the degrees of the leaves.

Degree of any intermediate split ideal is at most $\deg(f)$.

Time Complexity

Degree of a node distributes to degree of its children.

Inductively, it yields that degree of root $\deg(f)$ is at least sum of the degrees of the leaves.

Degree of any intermediate split ideal is at most $\deg(f)$.

Ring operations modulo an ideal I are bounded by $\text{poly}(k \log p, \deg(I)) = \text{poly}(k \log p, d)$.

Time Complexity

Degree of a node distributes to degree of its children.

Inductively, it yields that degree of root $\deg(f)$ is at least sum of the degrees of the leaves.

Degree of any intermediate split ideal is at most $\deg(f)$.

Ring operations modulo an ideal I are bounded by $\mathrm{poly}(k \log p, \deg(I)) = \mathrm{poly}(k \log p, d)$.

Size of tree captures the number of iterations- $O(kd)$.

Overview

Conclusion

Our algorithm extends to exactly count basic irreducible factors of $f \bmod p^k$.

Conclusion

Our algorithm extends to exactly count basic irreducible factors of $f \bmod p^k$.

Open: Testing irreducibility of $f \bmod p^k$ in deterministic (even randomized) poly-time?

Conclusion

Our algorithm extends to exactly count basic irreducible factors of $f \bmod p^k$.

Open: Testing irreducibility of $f \bmod p^k$ in deterministic (even randomized) poly-time?

Questions?

Thank You for your attention!