Counting Basic-Irreducible Factors Mod p^k in Deterministic Poly-Time and p-Adic Applications

Ashish Dwivedi
IIT Kanpur, India

Joint work with
Rajat Mittal (IIT Kanpur, India) and Nitin Saxena (IIT Kanpur, India)

34TH COMPUTATIONAL COMPLEXITY CONFERENCE (CCC) 2019
Rutgers University, New Brunswick, NJ, USA

Overview

Introduction

Root finding of univariates modulo a prime is a well studied problem!

Introduction

Root finding of univariates modulo a prime is a well studied problem!

Many efficient randomized algorithms are known.

Introduction

Root finding of univariates modulo a prime is a well studied problem!

Many efficient randomized algorithms are known.

Open: A deterministic poly-time algorithm?

Introduction

Root finding of univariates modulo a prime is a well studied problem!

Many efficient randomized algorithms are known.

Open: A deterministic poly-time algorithm?

Known:

Introduction

Root finding of univariates modulo a prime is a well studied problem!

Many efficient randomized algorithms are known.

Open: A deterministic poly-time algorithm?

Known: Deterministic poly-time root counting.

Introduction

Root finding of univariates modulo a prime is a well studied problem!

Many efficient randomized algorithms are known.

Open: A deterministic poly-time algorithm?

Known: Deterministic poly-time root counting.

What about factoring modulo a composite n?

Introduction

Root finding of univariates modulo a prime is a well studied problem!

Many efficient randomized algorithms are known.

Open: A deterministic poly-time algorithm?

Known: Deterministic poly-time root counting.

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

Introduction

Root finding of univariates modulo a prime is a well studied problem!

Many efficient randomized algorithms are known.

Open: A deterministic poly-time algorithm?

Known: Deterministic poly-time root counting.

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

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

Introduction

Root finding of univariates modulo a prime is a well studied problem!

Many efficient randomized algorithms are known.

Open: A deterministic poly-time algorithm?

Known: Deterministic poly-time root counting.

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

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

Getting roots mod p^k

Introduction

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 $f \bmod p$, Hensel's lifting guarantees unique lift mod p^k.

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

$\Rightarrow f \equiv x(x-1) \bmod 3$.

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 $f(x) = x^2 - 10x + 21$ and $p = 3$.

$\Rightarrow f \equiv x(x-1) \bmod 3$. Roots: $0, 1$!

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 $f(x) = x^2 - 10x + 21$ and $p = 3$.

$\Rightarrow f \equiv x(x-1) \bmod 3$. Roots: $0, 1$!

Let $f \equiv (x - 3a)(x - 1 - 3b) \bmod 9$

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 $f(x) = x^2 - 10x + 21$ and $p = 3$.

$\Rightarrow f \equiv x(x-1) \bmod 3$. Roots: $0, 1$!

Let $f \equiv (x - 3a)(x - 1 - 3b) \bmod 9$

$\Rightarrow a \equiv 1 \bmod 3$ and $b \equiv 2 \bmod 3$

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 $f(x) = x^2 - 10x + 21$ and $p = 3$.

$\Rightarrow f \equiv x(x-1) \bmod 3$. Roots: $0, 1$!

Let $f \equiv (x - 3a)(x - 1 - 3b) \bmod 9$

$\Rightarrow a \equiv 1 \bmod 3$ and $b \equiv 2 \bmod 3$ Unique solution!

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 $f(x) = x^2 - 10x + 21$ and $p = 3$.

$\Rightarrow f \equiv x(x-1) \bmod 3$. Roots: $0, 1$!

Let $f \equiv (x - 3a)(x - 1 - 3b) \bmod 9$

$\Rightarrow a \equiv 1 \bmod 3$ and $b \equiv 2 \bmod 3$ Unique solution!

So $f(x) \equiv (x-3)(x-7) \bmod 3^2$.

Getting roots mod p^k

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

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

$\Rightarrow f \equiv x(x-1) \bmod 3$. Roots: $0, 1$!

Let $f \equiv (x - 3a)(x - 1 - 3b) \bmod 9$

$\Rightarrow a \equiv 1 \bmod 3$ and $b \equiv 2 \bmod 3$ Unique solution!

So $f(x) \equiv (x-3)(x-7) \bmod 3^2$. Roots after lift: $3, 7$!

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 $f(x) = x^2 - 10x + 21$ and $p = 3$.

$\Rightarrow f \equiv x(x-1) \bmod 3$. Roots: $0, 1$!

Let $f \equiv (x - 3a)(x - 1 - 3b) \bmod 9$

$\Rightarrow a \equiv 1 \bmod 3$ and $b \equiv 2 \bmod 3$ Unique solution!

So $f(x) \equiv (x-3)(x-7) \bmod 3^2$. Roots after lift: $3, 7$!

The lifting goes on same way 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!

It requires co-prime factors,

Introduction

First issue:

Multiplicity > 1? Hensel lifting fails!

It requires co-prime factors, otherwise non-unique lift or no lift at all.

Introduction

First issue:

Multiplicity > 1? Hensel lifting fails!

It requires co-prime factors, otherwise non-unique lift or no lift at all.

Eg. $f = x^2 + p$

Introduction

First issue:

Multiplicity > 1? Hensel lifting fails!

It requires co-prime factors, otherwise non-unique lift or no lift at all.

Eg. $f = x^2 + p$ and so $f \equiv x^2 \bmod p$.

Introduction

First issue:

Multiplicity > 1? Hensel lifting fails!

It requires co-prime factors, otherwise non-unique lift or no lift at all.

Eg. $f = x^2 + p$ and so $f \equiv x^2 \bmod p$. Root 0 doesn't lift mod p^2.

First issue:

Multiplicity > 1? Hensel lifting fails!

It requires co-prime factors, otherwise non-unique lift or no lift at all.

Eg. $f = x^2 + p$ and so $f \equiv x^2 \bmod p$. Root 0 doesn't lift mod p^2.

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

First issue:

Multiplicity > 1? Hensel lifting fails!

It requires co-prime factors, otherwise non-unique lift or no lift at all.

Eg. $f = x^2 + p$ and so $f \equiv x^2 \bmod p$. Root 0 doesn't lift mod p^2.

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

Second issue:

Introduction

First issue:

Multiplicity > 1? Hensel lifting fails!

It requires co-prime factors, otherwise non-unique lift or no lift at all.

Eg. $f = x^2 + p$ and so $f \equiv x^2 \bmod p$. Root 0 doesn't lift mod p^2.

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!

It requires co-prime factors, otherwise non-unique lift or no lift at all.

Eg. $f = x^2 + p$ and so $f \equiv x^2 \bmod p$. Root 0 doesn't lift mod p^2.

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!

It requires co-prime factors, otherwise non-unique lift or no lift at all.

Eg. $f = x^2 + p$ and so $f \equiv x^2 \bmod p$. Root 0 doesn't lift mod p^2.

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!

It requires co-prime factors, otherwise non-unique lift or no lift at all.

Eg. $f = x^2 + p$ and so $f \equiv x^2 \bmod p$. Root 0 doesn't lift mod p^2.

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!

It requires co-prime factors, otherwise non-unique lift or no lift at all.

Eg. $f = x^2 + p$ and so $f \equiv x^2 \bmod p$. Root 0 doesn't lift mod p^2.

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 $\alpha \in \mathbb{F}_p$.

Due to this, the search space could be exponential at every stage of lifting!

Introduction

First issue:

Multiplicity > 1? Hensel lifting fails!

It requires co-prime factors, otherwise non-unique lift or no lift at all.

Eg. $f = x^2 + p$ and so $f \equiv x^2 \bmod p$. Root 0 doesn't lift mod p^2.

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 $\alpha \in \mathbb{F}_p$.

Due to this, the search space could be exponential at every stage of lifting!

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

Introduction

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

Introduction

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

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

Eg. Let $f = x^4 + 249x^2 + 1458$ and $p^k = 3^6$.

Introduction

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

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

Eg. Let $f = x^4 + 249x^2 + 1458$ and $p^k = 3^6$.

So $f \equiv x^4 \bmod 3$ Hard Case!

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

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

Eg. Let $f = x^4 + 249x^2 + 1458$ and $p^k = 3^6$.

So $f \equiv x^4 \bmod 3$ Hard Case!

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

Introduction

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

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

Eg. Let $f = x^4 + 249x^2 + 1458$ and $p^k = 3^6$.

So $f \equiv x^4 \bmod 3$ Hard Case!

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

A completely unrelated irreducible factorization:

Introduction

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

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

Eg. Let $f = x^4 + 249x^2 + 1458$ and $p^k = 3^6$.

So $f \equiv x^4 \bmod 3$ Hard Case!

$f = (x^2 + 243)\,(x^2 + 6) \bmod 3^6$ 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 $f \bmod p^k$; 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 $f \bmod p^k$; exponential in input size.

Berthomieu, Lecerf and Quintin [BLQ 2013] gave a randomized poly-time algorithm to 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 $f \bmod p^k$; exponential in input size.

Berthomieu, Lecerf and Quintin [BLQ 2013] gave a randomized poly-time algorithm to find and count exactly the roots of $f \bmod p^k$.

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 $f \bmod p^k$; exponential in input size.

Berthomieu, Lecerf and Quintin [BLQ 2013] gave a randomized poly-time algorithm to find and count exactly the roots of $f \bmod p^k$.

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

Counting roots 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 $f \bmod p^k$; exponential in input size.

Berthomieu, Lecerf and Quintin [BLQ 2013] gave a randomized poly-time algorithm to find and count exactly the roots of $f \bmod p^k$.

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

Counting roots is stronger than just showing the existence of a root.

Extension to count irreducible factors will give an irreducibility criteria.

Our Results

Derandomization is a holy-grail in computational complexity.

Our Results

Derandomization is a holy-grail in computational complexity.

It is interesting to know how we can search deterministically in an exponential space.

Our Results

Derandomization is a holy-grail in computational complexity.

It is interesting to know how we can search deterministically in an exponential space.

We give a deterministic poly-time algorithm to exactly count roots .

Our Results

Derandomization is a holy-grail in computational complexity.

It is interesting to know how we can search deterministically in an exponential space.

We give a deterministic poly-time algorithm to exactly count roots .

We will do more-

Our Results

Derandomization is a holy-grail in computational complexity.

It is interesting to know how we can search deterministically in an exponential space.

We give a deterministic poly-time algorithm to exactly count roots .

We will do more- A Structural Result.

Our Results

Derandomization is a holy-grail in computational complexity.

It is interesting to know how we can search deterministically in an exponential space.

We give a deterministic poly-time algorithm to exactly count roots .

We will do more- A Structural Result.

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

Our Results

Derandomization is a holy-grail in computational complexity.

It is interesting to know how we can search deterministically in an exponential space.

We give a deterministic poly-time algorithm to exactly count roots .

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.

Our Results

Derandomization is a holy-grail in computational complexity.

It is interesting to know how we can search deterministically in an exponential space.

We give a deterministic poly-time algorithm to exactly count roots .

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.

Our result extends to count exactly the basic-irreducible factors of $f \bmod p^k$ as well.

Efficiently Partitioning the Root Set

To get exponentially many roots efficiently, the real challenge is to first find a compact representation of the root set of $f \bmod p^k$.

Efficiently Partitioning the Root Set

To get exponentially many roots efficiently, the real challenge is to first find a compact representation of the root set of $f \bmod p^k$.

This was first achieved by Berthomieu, Lecerf and Quintin (2013) in randomized setting.

Efficiently Partitioning the Root Set

To get exponentially many roots efficiently, the real challenge is to first find a compact representation of the root set of $f \bmod p^k$.

This was first achieved by Berthomieu, Lecerf and Quintin (2013) in randomized setting.

By efficiently partitioning the root set of $f \bmod p^k$, [BLQ 13] gave the first randomized poly-time algorithm to find (& count) exactly the roots of $f \bmod p^k$.

Efficiently Partitioning the Root Set

To get exponentially many roots efficiently, the real challenge is to first find a compact representation of the root set of $f \bmod p^k$.

This was first achieved by Berthomieu, Lecerf and Quintin (2013) in randomized setting.

By efficiently partitioning the root set of $f \bmod p^k$, [BLQ 13] gave the first randomized poly-time algorithm to find (& count) exactly the roots of $f \bmod p^k$.

We give a simple exposition of [BLQ 13] which helps understand our deterministic algorithm.

Overview

Randomized Algorithm: Framework

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

Randomized Algorithm: Framework

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

Fact: any root mod p^k is a lift of some root mod p^ℓ for all $\ell \leq k$.

Randomized Algorithm: Framework

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

Fact: any root mod p^k is a lift of some root mod p^ℓ for all $\ell \leq k$.

$$r = r_0 + pr_1 + \ldots + p^{k-1} r_{k-1}$$

Randomized Algorithm: Framework

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

Fact: any root mod p^k is a lift of some root mod p^ℓ for all $\ell \leq k$.

$$r = r_0 + pr_1 + \ldots + p^{k-1} r_{k-1}$$

r is a lift of r_0 mod p,

Randomized Algorithm: Framework

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

Fact: any root mod p^k is a lift of some root mod p^ℓ for all $\ell \leq k$.

$$r = r_0 + pr_1 + \ldots + p^{k-1} r_{k-1}$$

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 CZ).

Fact: any root mod p^k is a lift of some root mod p^ℓ for all $\ell \leq k$.

$$r = r_0 + pr_1 + \ldots + p^{k-1} r_{k-1}$$

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

Idea: 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 $r_{k-\alpha}$ and beyond.

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 $r_{k-\alpha}$ and beyond.

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 $r_{k-\alpha}$ and beyond.

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 $\mathbb{F}_p$.

In short, we write

$$r = r_0 + pr_1 + \ldots + p^{k-\alpha} *$$

where r is called a **representative root** representing p^α 'distinct' roots of $f \bmod p^k$.

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 $r_{k-\alpha}$ and beyond.

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 $\mathbb{F}_p$.

In short, we write

$$r = r_0 + pr_1 + \ldots + p^{k-\alpha} *$$

where r is called a **representative root** representing p^α 'distinct' roots of f mod p^k.

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

Randomized Algorithm

Recall: Incrementally build up r by finding co-ordinates r_i one by one.

Randomized Algorithm

Recall: Incrementally build up r by finding co-ordinates r_i one by one.

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

Randomized Algorithm

Recall: Incrementally build up r by finding co-ordinates r_i one by one.

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

For every r_0 obtained do the following:
{

Randomized Algorithm

Recall: Incrementally build up r by finding co-ordinates r_i one by one.

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

For every r_0 obtained do the following:
{

Shift: $f(x) \mapsto f(r_0 + px)$,

Randomized Algorithm

Recall: Incrementally build up r by finding co-ordinates r_i one by one.

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

For every r_0 obtained do the following:
{

Shift: $f(x) \mapsto f(r_0 + px)$,

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

Randomized Algorithm

Recall: Incrementally build up r by finding co-ordinates r_i one by one.

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

For every r_0 obtained do the following:
$\{$

Shift: $f(x) \mapsto f(r_0 + px)$,

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

Repeat the Shift-Divide cycle on $g(x) \bmod p^{k-\alpha}$ to get corresponding r_1s and so on.
$\}$

Randomized Algorithm: Correctness

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

Randomized Algorithm: Correctness

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

Essentially every iteration reduces finding roots of $f(x) \bmod p^k$, which are lifts of r_0, to roots of $g(x) \bmod p^{k-\alpha}$.

Randomized Algorithm: Correctness

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

Essentially every iteration reduces finding roots of $f(x) \bmod p^k$, which are lifts of r_0, to roots of $g(x) \bmod p^{k-\alpha}$.

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

Essentially every iteration reduces finding roots of $f(x) \bmod p^k$, which are lifts of r_0, to roots of $g(x) \bmod p^{k-\alpha}$.

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

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

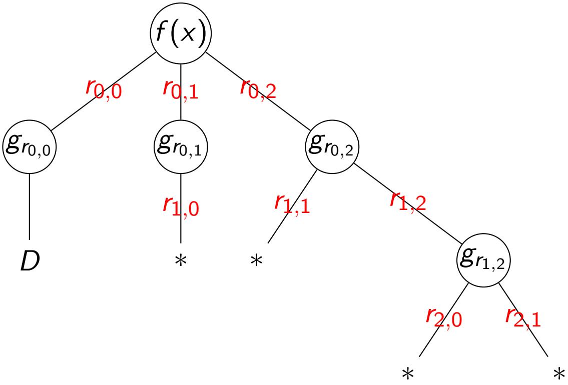
The time taken could be very high? $\deg(f)^k$ many roots in the end?

The algorithm forms a virtual tree of roots:

Randomized Algorithm: Time Complexity

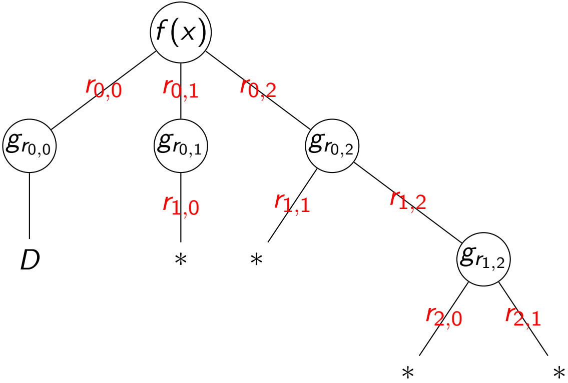
The time taken could be very high? $\deg(f)^k$ many roots in the end?

The algorithm forms a virtual tree of roots:



Randomized Algorithm: Time Complexity

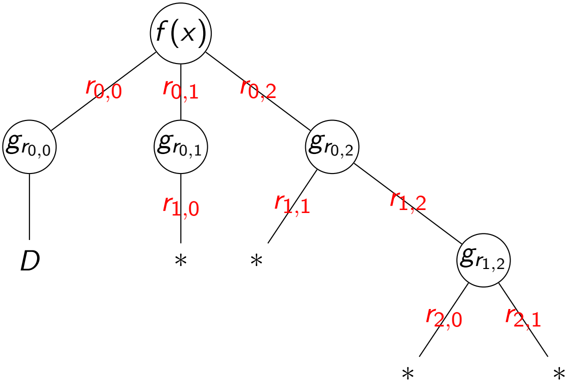
The time taken could be very high? $\deg(f)^k$ many roots in the end?
The algorithm forms a virtual tree of roots:



Roots are: $r_{0,1} + pr_{1,0} + p^2*$,

Randomized Algorithm: Time Complexity

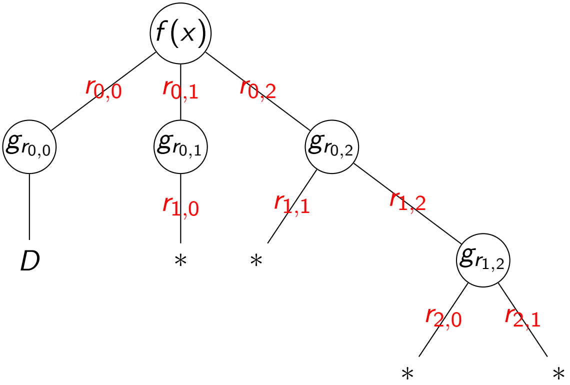
The time taken could be very high? $\deg(f)^k$ many roots in the end?
The algorithm forms a virtual tree of roots:



Roots are: $r_{0,1} + pr_{1,0} + p^2*$, $r_{0,2} + pr_{1,1} + p^2*$,

Randomized Algorithm: Time Complexity

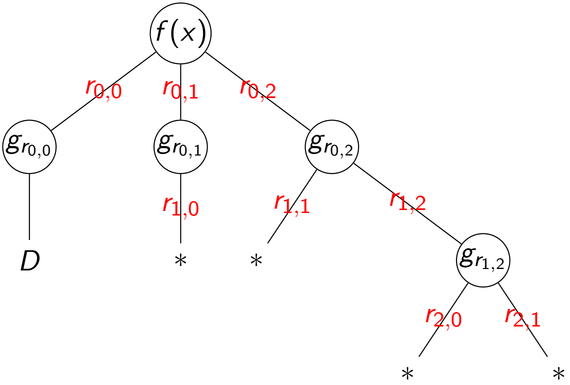
The time taken could be very high? $\deg(f)^k$ many roots in the end?
The algorithm forms a virtual tree of roots:



Roots are: $r_{0,1} + pr_{1,0} + p^2*$, $r_{0,2} + pr_{1,1} + p^2*$,
$r_{0,2} + pr_{1,2} + p^2 r_{2,0} + p^3*$,

Randomized Algorithm: Time Complexity

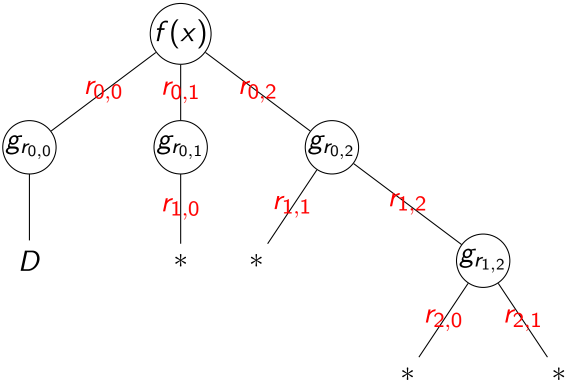
The time taken could be very high? $\deg(f)^k$ many roots in the end?
The algorithm forms a virtual tree of roots:



Roots are: $r_{0,1} + pr_{1,0} + p^2 *$, $r_{0,2} + pr_{1,1} + p^2 *$,
$r_{0,2} + pr_{1,2} + p^2 r_{2,0} + p^3 *$, $r_{0,2} + pr_{1,2} + p^2 r_{2,1} + p^3 *$

Randomized Algorithm: Time Complexity

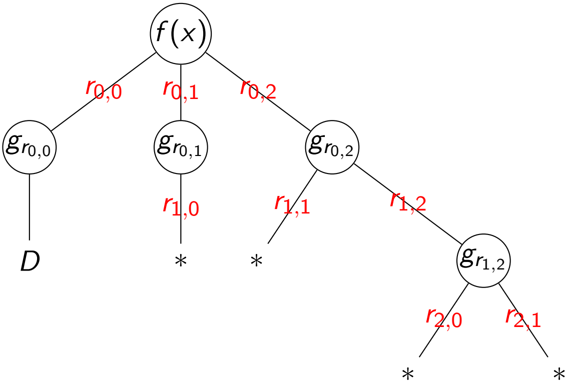
The time taken could be very high? $\deg(f)^k$ many roots in the end?
The algorithm forms a virtual tree of roots:



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

Randomized Algorithm: Time Complexity

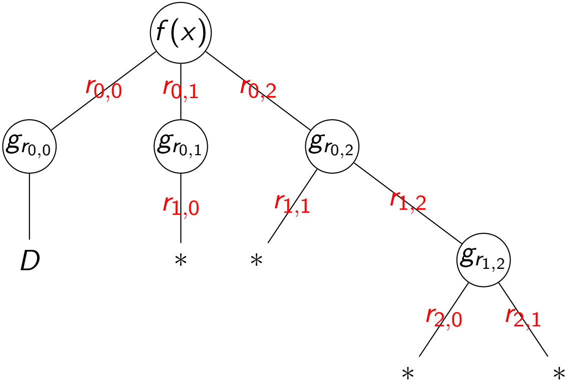
The time taken could be very high? $\deg(f)^k$ many roots in the end?
The algorithm forms a virtual tree of roots:



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

Randomized Algorithm: Time Complexity

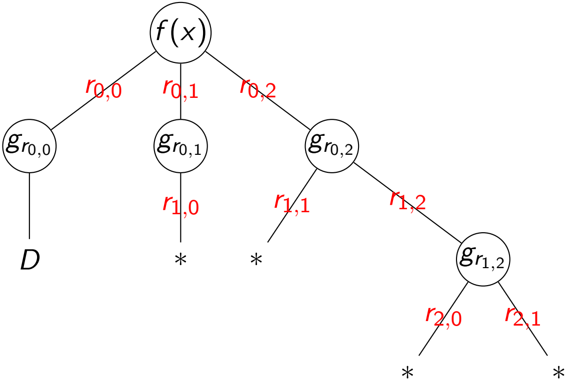
The time taken could be very high? $\deg(f)^k$ many roots in the end?
The algorithm forms a virtual tree of roots:



Multiplicity Property:

Randomized Algorithm: Time Complexity

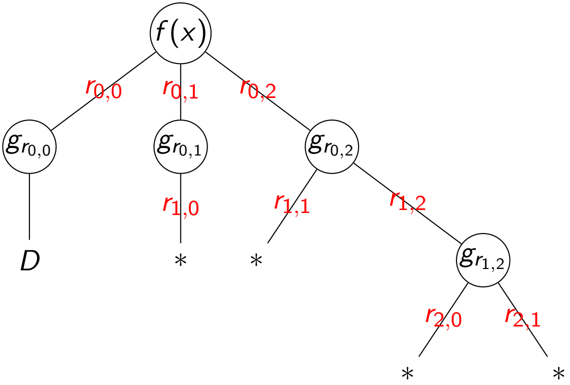
The time taken could be very high? $\deg(f)^k$ many roots in the end?
The algorithm forms a virtual tree of roots:



Multiplicity Property: Let r_0 be a root of multiplicity m of $f(x) \bmod p$ then the degree of children corresponding to r_0 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:



So, the size of tree is polynomial in input size and the algorithm runs in randomized poly($\deg(f)$, $k \log p$) time.

Overview

Challenges in Derandomization

Challenge: Extend the properties, of the randomized algorithms, to the deterministic (poly-time) context.

Challenges in Derandomization

Challenge: Extend the properties, of the randomized algorithms, to the deterministic (poly-time) context.

Can we still cluster (may be implicitly) the roots of $f \bmod p^k$ into $\deg(f)$ many clusters, deterministically? (CZ is not available)

Challenges in Derandomization

Challenge: Extend the properties, of the randomized algorithms, to the deterministic (poly-time) context.

Can we still cluster (may be implicitly) the roots of $f \bmod p^k$ into $\deg(f)$ many clusters, deterministically? (CZ is not available)

Can we generalize the multiplicity argument of [BLQ'13] in the deterministic context?

Challenges in Derandomization

Challenge: Extend the properties, of the randomized algorithms, to the deterministic (poly-time) context.

Can we still cluster (may be implicitly) the roots of $f \bmod p^k$ into $\deg(f)$ many clusters, deterministically? (CZ is not available)

Can we generalize the multiplicity argument of [BLQ'13] in the deterministic context?

Can we extend the techniques to count basic-irreducible factors $f \bmod p^k$?

Challenges in Derandomization

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

Challenges in Derandomization

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.

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 $g(x_0 + px) \bmod \langle x_0 - a \rangle$.

Similarly,

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

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

Deterministic Algorithm: Tool 2

Given $g(x) \bmod p$, 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 $g(x) \bmod p$, 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

Initialization:

Deterministic Algorithm

Initialization:
$h_0(x) := (f(x), x^p - x) \bmod p$.

Deterministic Algorithm

Initialization:
$h_0(x) := (f(x), x^p - x) \bmod p$.
$I_0 := \langle h_0(x_0) \rangle$.

Deterministic Algorithm

Initialization:

$h_0(x) := (f(x), x^p - x) \bmod p$.

$I_0 := \langle h_0(x_0) \rangle$. $\deg(I_0) = \deg(h_0)$= count on roots mod p.

Deterministic Algorithm

Initialization:

$h_0(x) := (f(x), x^p - x) \bmod p$.

$I_0 := \langle h_0(x_0) \rangle$. $\deg(I_0) = \deg(h_0)$= count on roots mod p.

Intermediate Step:

Deterministic Algorithm

Initialization:

$h_0(x) := (f(x), x^p - x) \bmod p$.

$I_0 := \langle h_0(x_0) \rangle$. $\deg(I_0) = \deg(h_0)$= count on roots mod p.

Intermediate Step:

Let $I_\ell = \langle h_0(x_0), h_1(x_0, x_1), \ldots, h_\ell(x_0, \ldots, x_\ell) \rangle$.

Deterministic Algorithm

Initialization:

$h_0(x) := (f(x), x^p - x) \bmod p$.

$I_0 := \langle h_0(x_0) \rangle$. $\deg(I_0) = \deg(h_0)$= count on roots mod p.

Intermediate Step:

Let $I_\ell = \langle h_0(x_0), h_1(x_0, x_1), \ldots, h_\ell(x_0, \ldots, x_\ell) \rangle$. $\deg(I_\ell) := \prod \deg_{x_i} h_i$

Deterministic Algorithm

Initialization:

$h_0(x) := (f(x), x^p - x) \bmod p$.

$I_0 := \langle h_0(x_0) \rangle$. $\deg(I_0) = \deg(h_0) =$ count on roots mod p.

Intermediate Step:

Let $I_\ell = \langle h_0(x_0), h_1(x_0, x_1), \ldots, h_\ell(x_0, \ldots, x_\ell) \rangle$. $\deg(I_\ell) := \prod \deg_{x_i} h_i$

The split ideal I_ℓ splits as $I_\ell = \bigcap \langle x_0 - a_0, x_1 - a_1, \ldots, x_\ell - a_\ell \rangle$.

Deterministic Algorithm

Initialization:

$h_0(x) := (f(x), x^p - x) \bmod p$.

$I_0 := \langle h_0(x_0) \rangle$. $\deg(I_0) = \deg(h_0)$= count on roots mod p.

Intermediate Step:

Let $I_\ell = \langle h_0(x_0), h_1(x_0, x_1), \ldots, h_\ell(x_0, \ldots, x_\ell) \rangle$. $\deg(I_\ell) := \prod \deg_{x_i} h_i$

The split ideal I_ℓ splits as $I_\ell = \bigcap \langle x_0 - a_0, x_1 - a_1, \ldots, x_\ell - a_\ell \rangle$.

Shift: $g(x) := f(x_0 + px_1 + \ldots + p^\ell x_\ell + p^{\ell+1} x) \bmod I_\ell$

Deterministic Algorithm

Initialization:

$h_0(x) := (f(x), x^p - x) \bmod p$.

$I_0 := \langle h_0(x_0) \rangle$. $\deg(I_0) = \deg(h_0)$= count on roots mod p.

Intermediate Step:

Let $I_\ell = \langle h_0(x_0), h_1(x_0, x_1), \ldots, h_\ell(x_0, \ldots, x_\ell) \rangle$. $\deg(I_\ell) := \prod \deg_{x_i} h_i$

The split ideal I_ℓ splits as $I_\ell = \bigcap \langle x_0 - a_0, x_1 - a_1, \ldots, x_\ell - a_\ell \rangle$.

Shift: $g(x) := f(x_0 + px_1 + \ldots + p^\ell x_\ell + p^{\ell+1} x) \bmod I_\ell$

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

Deterministic Algorithm

Initialization:

$h_0(x) := (f(x), x^p - x) \bmod p$.

$I_0 := \langle h_0(x_0) \rangle$. $\deg(I_0) = \deg(h_0)$= count on roots mod p.

Intermediate Step:

Let $I_\ell = \langle h_0(x_0), h_1(x_0, x_1), \ldots, h_\ell(x_0, \ldots, x_\ell) \rangle$. $\deg(I_\ell) := \prod \deg_{x_i} h_i$

The split ideal I_ℓ splits as $I_\ell = \bigcap \langle x_0 - a_0, x_1 - a_1, \ldots, x_\ell - a_\ell \rangle$.

Shift: $g(x) := f(x_0 + px_1 + \ldots + p^\ell x_\ell + p^{\ell+1} x) \bmod I_\ell$

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

GCD: $h_{\ell+1}(x) = (g(x), x^p - x) \bmod I_\ell$.

Deterministic Algorithm

Initialization:

$h_0(x) := (f(x), x^p - x) \bmod p$.

$I_0 := \langle h_0(x_0) \rangle$. $\deg(I_0) = \deg(h_0) =$ count on roots mod p.

Intermediate Step:

Let $I_\ell = \langle h_0(x_0), h_1(x_0, x_1), \ldots, h_\ell(x_0, \ldots, x_\ell) \rangle$. $\deg(I_\ell) := \prod \deg_{x_i} h_i$

The split ideal I_ℓ splits as $I_\ell = \bigcap \langle x_0 - a_0, x_1 - a_1, \ldots, x_\ell - a_\ell \rangle$.

Shift: $g(x) := f(x_0 + px_1 + \ldots + p^\ell x_\ell + p^{\ell+1} x) \bmod I_\ell$

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

GCD: $h_{\ell+1}(x) = (g(x), x^p - x) \bmod I_\ell$.

Update: $I_{\ell+1} := I_\ell + \langle h_{\ell+1} \rangle$.

Deterministic Algorithm

Initialization:

$h_0(x) := (f(x), x^p - x) \bmod p$.

$I_0 := \langle h_0(x_0) \rangle$. $\deg(I_0) = \deg(h_0)$= count on roots mod p.

Intermediate Step:

Let $I_\ell = \langle h_0(x_0), h_1(x_0, x_1), \ldots, h_\ell(x_0, \ldots, x_\ell) \rangle$. $\deg(I_\ell) := \prod \deg_{x_i} h_i$

The split ideal I_ℓ splits as $I_\ell = \bigcap \langle x_0 - a_0, x_1 - a_1, \ldots, x_\ell - a_\ell \rangle$.

Shift: $g(x) := f(x_0 + px_1 + \ldots + p^\ell x_\ell + p^{\ell+1} x) \bmod I_\ell$

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

GCD: $h_{\ell+1}(x) = (g(x), x^p - x) \bmod I_\ell$.

Update: $I_{\ell+1} := I_\ell + \langle h_{\ell+1} \rangle$.

In the end, all ideals implicitly store all the roots of $f \bmod p^k$.

Time Complexity

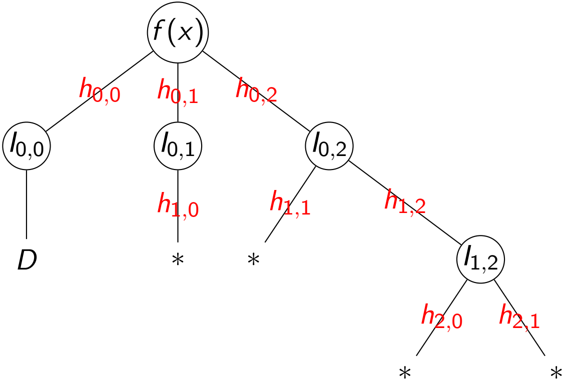
Why the deterministic algorithm is efficient?

Time Complexity

Why the deterministic algorithm is efficient?
The deterministic algorithm virtually creates a Roots-Tree:

Time Complexity

Why the deterministic algorithm is efficient?
The deterministic algorithm virtually creates a Roots-Tree:



Time Complexity

Consider a Node N labelled by split ideal I.

Time Complexity

Consider a Node N labelled by split 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

Consider a Node N labelled by split 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.

Degree of a node distributes to degree of its children.

Time Complexity

Consider a Node N labelled by split 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.

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

Consider a Node N labelled by split 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.

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.

Similar to the randomized root tree, the size of the deterministic root tree is polynomial in input size.

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 $f \bmod p^k$.

Recently, D., Mittal, Saxena [ISSAC' 19] gave a randomized poly-time algorithm to factor $f \bmod p^k$ for $k \leq 4$.

Conclusion

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

Recently, D., Mittal, Saxena [ISSAC' 19] gave a randomized poly-time algorithm to factor $f \bmod p^k$ for $k \leq 4$.

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 $f \bmod p^k$.

Recently, D., Mittal, Saxena [ISSAC' 19] gave a randomized poly-time algorithm to factor $f \bmod p^k$ for $k \leq 4$.

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

Questions?

Thank You for your attention!