A Faster Randomized Algorithm for Counting Roots in $\mathbb{Z}/(p^k)$

Natalie Randall

Department of Mathematics, Texas A&M University
DMS - 1757872
Austin College

16th July 2018

Applications

- **Exponential Sums:**

$$S(f,n) = \left| \sum_{i=1}^{n} e^{\frac{2\pi\sqrt{-1}g(x)}{n}} \right|$$

In various applications, one needs an estimate for $S(f,n)$, where $g \in \mathbb{Z}[x]$.

Applications

- **Exponential Sums:**

$$S(f,n) = \left| \sum_{i=1}^{n} e^{\frac{2\pi\sqrt{-1}g(x)}{n}} \right|$$

  In various applications, one needs an estimate for $S(f,n)$, where $g \in \mathbb{Z}[x]$.

- Work done by Cochrane and Zheng (2001) shows that estimating $S(f,n)$ is closely related to counting roots of $g \in \mathbb{Z}/(p^k)$.

Applications

- Counting roots in the p-adic integers ($\mathbb{Z}_p$) has numerous applications to Diophantine equations (e.g., Skolem's method [Smart, 1999]).

Applications

- Counting roots in the p-adic integers ($\mathbb{Z}_p$) has numerous applications to Diophantine equations (e.g., Skolem's method [Smart, 1999]).
- Counting roots in $\mathbb{Z}/(p^k)$ is also closely related to counting roots in $\mathbb{Q}_p$.

Main Results

- There is a Las Vegas randomized algorithm that counts the roots of any $f \in \mathbb{Z}[x]$ (such that f is not identically 0 modulo p) of degree d in time:

$$d^{1.5+o(1)}(\log p)^{2+o(1)}1.12^k$$

Main Results

- There is a Las Vegas randomized algorithm that counts the roots of any $f \in \mathbb{Z}[x]$ (such that f is not identically 0 modulo p) of degree d in time:

$$d^{1.5+o(1)}(\log p)^{2+o(1)}1.12^k$$

- "Las Vegas" refers to an algorithm that fails with probability $< \frac{1}{3}$, but correctly announces this failure. So, to get failure probability $< \frac{1}{3^{100}}$, just run the algorithm 100 times.

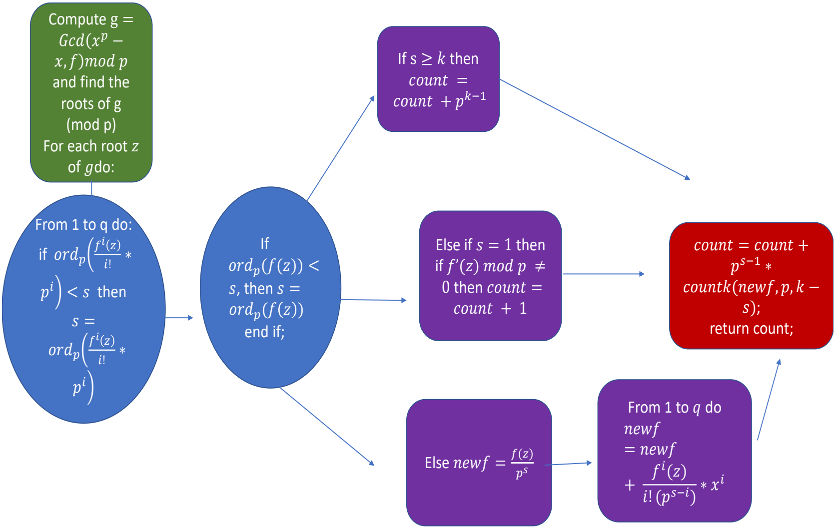
Main Results

- There is a Las Vegas randomized algorithm that counts the roots of any $f \in \mathbb{Z}[x]$ (such that f is not identically 0 modulo p) of degree d in time:

$$d^{1.5+o(1)}(\log p)^{2+o(1)}1.12^k$$

- "Las Vegas" refers to an algorithm that fails with probability $< \frac{1}{3}$, but correctly announces this failure. So, to get failure probability $< \frac{1}{3^{100}}$, just run the algorithm 100 times.
- Las Vegas algorithms are accepted by and occur frequently in algorithmic number theory (e.g., factoring polynomials over finite fields and primality testing).

The Algorithm



The Key Trick

- For a polynomial $f \in \mathbb{Z}/(p^k)$, we compute its roots $\mod p$ by taking the $Gcd(x^p - x, f) \mod p$

The Key Trick

- For a polynomial $f \in \mathbb{Z}/(p^k)$, we compute its roots $\mod p$ by taking the $Gcd(x^p - x, f) \mod p$
- Then, given any root $\zeta \in \mathbb{Z}/(p)$ of f and an $\varepsilon \in \{0, 1, \cdots, p^{k-1} - 1\}$, we have a perturbation as follows (using a Taylor series expansion):

$$f(\zeta + p \cdot \varepsilon) = f(\zeta) + f'(\zeta)p\varepsilon + \frac{1}{2}f''(\zeta)p^2\varepsilon^2 + \ldots$$

$$\ldots + \frac{1}{(k-1)!}f^{(k-1)}(\zeta)p^{k-1}\varepsilon^{k-1} \mod p^k$$

The Key Trick

- By finding a maximum $s \in \{1, ..., k\}$ such that $p^s \mid \frac{f(\zeta)}{p^s}, ..., \frac{1}{(k-1)!} f^{(k-1)}(\zeta) p^{k-1}$ we get:

$$p^s \left(\frac{f(\zeta)}{p^s} + \frac{f'(\zeta)}{p^{s-1}} \cdot \varepsilon + ... + \frac{f^{(m)}(\zeta)}{m! \cdot p^{s-m}} \cdot \varepsilon^m \right) \mod p^k$$

where we let $m = min(degree(f), k - 1)$

The Key Trick

- By finding a maximum $s \in \{1, ..., k\}$ such that $p^s \mid \frac{f(\zeta)}{p^s}, ..., \frac{1}{(k-1)!} f^{(k-1)}(\zeta) p^{k-1}$ we get:

$$p^s \left(\frac{f(\zeta)}{p^s} + \frac{f'(\zeta)}{p^{s-1}} \cdot \varepsilon + ... + \frac{f^{(m)}(\zeta)}{m! \cdot p^{s-m}} \cdot \varepsilon^m \right) \mod p^k$$

  where we let $m = min(degree(f), k-1)$

- We can then take the parenthetical part and write this as a function in terms of ε

$$g(\varepsilon) := \left(\frac{f(\zeta)}{p^s} + \frac{f'(\zeta)}{p^{s-1}} \cdot \varepsilon + ... + \frac{f^{(m)}(\zeta)}{m! \cdot p^{s-m}} \cdot \varepsilon^m \right) \mod p^{k-s}$$

Main Idea

- In previous algorithms, we were able to use this s to keep track of the cluster of lifts that each root produced.

Main Idea

- In previous algorithms, we were able to use this s to keep track of the cluster of lifts that each root produced.

The Case $k = 4$

Depending on the value of s, each root ζ generates 0, 1, p, $2p$, p^2, $2p^2$, $3p^2$, or p^3 lifts

Main Idea

- In previous algorithms, we were able to use this s to keep track of the cluster of lifts that each root produced.

The Case $k = 4$

Depending on the value of s, each root ζ generates 0, 1, p, $2p$, p^2, $2p^2$, $3p^2$, or p^3 lifts

- So, while it's possible to predict the number of roots via multiple cases by evaluating $ord_p(f(\zeta)), ord_p(f'(\zeta)), \ldots$, it is easier to rely on recursion for the cases where $s \in \{2, \ldots, k-1\}$.

Implementing the Algorithm

- Here is where the randomization of the algorithm is obtained- when we compute the roots $\mod p$.
- Compute $g = Gcd(x^p - x, f) \mod p$

Implementing the Algorithm

- Here is where the randomization of the algorithm is obtained- when we compute the roots $\mod p$.
- Compute $g = Gcd(x^p - x, f) \mod p$
- Calculate the roots of $g \mod p$, and for each root ζ implement the following steps:
  - Let $s = k$ and let $q = min(degree(f), k - 1)$;

Implementing the Algorithm

- Here is where the randomization of the algorithm is obtained- when we compute the roots $\mod p$.
- Compute $g = Gcd(x^p - x, f) \mod p$
- Calculate the roots of $g \mod p$, and for each root ζ implement the following steps:
  - Let $s = k$ and let $q = min(degree(f), k - 1)$;
  - For i from 1 to q, if $ord_p(\frac{f^i(\zeta)}{i!} \cdot p^i) < s$ then let $s = ord_p(\frac{f^i(\zeta)}{i!} \cdot p^i)$;

Implementing the Algorithm

- Here is where the randomization of the algorithm is obtained- when we compute the roots $\mod p$.
- Compute $g = Gcd(x^p - x, f) \mod p$
- Calculate the roots of $g \mod p$, and for each root ζ implement the following steps:
  - Let $s = k$ and let $q = min(degree(f), k-1)$;
  - For i from 1 to q, if $ord_p(\frac{f^i(\zeta)}{i!} \cdot p^i) < s$ then let $s = ord_p(\frac{f^i(\zeta)}{i!} \cdot p^i)$;
  - If $ord_p(f(\zeta)) < s$ then let $s = ord_p(f(\zeta))$;

Implementing the Algorithm

- Depending on the value of s, we get three distinct cases:

Implementing the Algorithm

- Depending on the value of s, we get three distinct cases:
- If s $\geq$ k, then let $count = count + p^{k-1}$ (ζ has exactly p^{k-1} lifts)

Implementing the Algorithm

- Depending on the value of s, we get three distinct cases:
- If s $\geq$ k, then let $count = count + p^{k-1}$ (ζ has exactly p^{k-1} lifts)
- Else if $s = 1$ then if $f'(\zeta)$ mod $p \neq 0$ then let $count = count + 1$ (ζ has 1 unique lift);

Implementing the Algorithm

- Depending on the value of s, we get three distinct cases:
- If s $\geq$ k, then let $count = count + p^{k-1}$ (ζ has exactly p^{k-1} lifts)
- Else if $s = 1$ then if $f'(\zeta) \bmod p \neq 0$ then let $count = count + 1$ (ζ has 1 unique lift);
- Else let $newf = \frac{f(\zeta)}{p^s}$; for i from 1 to q :

$$newf = newf + \frac{f^i(\zeta)}{i! \cdot p^{s-i}} \cdot x^i$$

- Let $count = count + p^{s-1} \cdot countk(newf, p, k - s)$

Example

- Consider $f(x) = x^2 - 4x + 226$ in $\mathbb{Z}/(7^4)$

Example

- Consider $f(x) = x^2 - 4x + 226$ in $\mathbb{Z}/(7^4)$
- $f(x) = x^2 - 4x + 226$
  mod $7 = x^2 + 3x + 2 = (x+1)(x+2) \to \zeta = -1, \zeta = -2$

Example

- Consider $f(x) = x^2 - 4x + 226$ in $\mathbb{Z}/(7^4)$
- $f(x) = x^2 - 4x + 226$
  mod $7 = x^2 + 3x + 2 = (x+1)(x+2) \to \zeta = -1, \zeta = -2$
- $f(\zeta + p \cdot \varepsilon) = f(-1 + 7\varepsilon) = 49\varepsilon^2 - 42\varepsilon + 231 \mod 7^4$

Example

- Consider $f(x) = x^2 - 4x + 226$ in $\mathbb{Z}/(7^4)$
- $f(x) = x^2 - 4x + 226$
  mod $7 = x^2 + 3x + 2 = (x+1)(x+2) \to \zeta = -1, \zeta = -2$
- $f(\zeta + p \cdot \varepsilon) = f(-1 + 7\varepsilon) = 49\varepsilon^2 - 42\varepsilon + 231 \mod 7^4$
- Here, $s = 1$; we increment *count* by 1 and continue counting in $\mathbb{Z}/(7^3)$

Advantages

The main benefit of using the randomized algorithm is its efficiency.

Advantages

The main benefit of using the randomized algorithm is its efficiency.

- For a polynomial with degree 84 in $\mathbb{Z}/(211^3)$:
  - Brute Force: 92.19 seconds
  - Randomized Algorithm: 11.00 milliseconds

Advantages

The main benefit of using the randomized algorithm is its efficiency.

- For a polynomial with degree 84 in $\mathbb{Z}/(211^3)$:
  - Brute Force: 92.19 seconds
  - Randomized Algorithm: 11.00 milliseconds
- For a polynomial with degree 99 in $\mathbb{Z}/(1049^3)$:
  - Brute Force: 3.81 hours
  - Randomized Algorithm: 1000.00us

Advantages

We can especially see the advantages of the randomized algorithm when we introduce large primes.

Advantages

We can especially see the advantages of the randomized algorithm when we introduce large primes.

- For a polynomial with degree 76 in $\mathbb{Z}/(8713^3)$, the randomized algorithm takes 2.00 ms.

Advantages

We can especially see the advantages of the randomized algorithm when we introduce large primes.

- For a polynomial with degree 76 in $\mathbb{Z}/(8713^3)$, the randomized algorithm takes 2.00 ms.
- For a polynomial with degree 93 in $\mathbb{Z}/(104729^3)$, the randomized algorithm takes 29.00 ms.

Advantages

We can especially see the advantages of the randomized algorithm when we introduce large primes.

- For a polynomial with degree 76 in $\mathbb{Z}/(8713^3)$, the randomized algorithm takes 2.00 ms.
- For a polynomial with degree 93 in $\mathbb{Z}/(104729^3)$, the randomized algorithm takes 29.00 ms.
- For a polynomial with degree 87 in $\mathbb{Z}/(179424673^3)$, the randomized algorithm takes 92.51 sec.

Analysis

Questions we want to know the answers to:

- What time complexity does this algorithm have?
- Can we can bound the maximum number of roots for any given polynomial?

Acknowledgements

- NSF
- J. Maurice Rojas
- Yuyu Zhu
- Leann Kopp