Check if a given polynomial is primitive

Asked 7 years, 6 months ago Modified 5 years ago Viewed 2k times

6

I try to estimate error detection capabilities of arbitrary CRC polynomials. One important criteria is if a given polynomial is primitive. So I need an algorithm to check that. My goal is to write a C or C++ routine.

Unfortunately I only found analytical solutions for the problem on the web.

Is there some numerical algorithm for testing a given polynomial for primitivity?

Please consider that my mathematical knowledge wasted away during the last two decades. Any algorithm descriptions, pseudo code or code in a common programming language would be very helpful.

numerical-algorithms polynomials crc

Share Cite

Improve this question Follow

edited Feb 27, 2019 at 23:39

asked Aug 20, 2016 at 20:08

Silicomancer

202 1 9

1 Answer

Sorted by: Highest score (default)

6

In order to check that a degree n polynomial P over $GF(2)$ is primitive, you first need to know the factorization of $2^n - 1$ (you can look it up in tables, or use a CAS). Then, you test that $x^{2^n-1} \equiv 1 \pmod{P(x)}$ (using repeated squaring to do this efficiently), and that for every prime factor p of $2^n - 1$, $x^{(2^n-1)/p} \not\equiv 1 \pmod{P(x)}$.

You can also just use a CAS (computer algebra software). For example, using the free software Sage you can do

```
F.<x> = GF(2)[]
(x^8+x^6+x^5+x+1).is_primitive()
```

For more on the relevant mathematics, see the Wikipedia article.

Share Cite Improve this answer Follow

answered Aug 20, 2016 at 20:44

Yuval Filmus
276k 27 309 502

1 Do you know if there is a simplified algorithm considering that CRC polynomials are pretty simple, having only terms with coefficients that are either 0 or 1? – Silicomancer Aug 20, 2016 at 22:39

1 @Silicomancer Polynomials over $GF(2)$ always have zero-one coefficients. – Yuval Filmus Aug 20, 2016 at 22:54

Oh, I see. Let me read more about that. Totally forgot the notations :(– Silicomancer Aug 20, 2016 at 23:04

I am not sure if I understand the description correctly. `x^(2^(n-1)) ≡ 1 (modP(x))` means I devide `x^(2^(n-1))` by `P(x)`, then I devide `x^0` by `P(x)` using polynomial division. Than I take both remainders and check if they are equal. Is this correct? – Silicomancer Aug 21, 2016 at 20:40

@Silicomancer That's right. – Yuval Filmus Aug 21, 2016 at 21:08

My CRC polynomials can be 64 bit wide. I.e. highest term `x^64` and `n=64`. `x^((2^n)-1)` would grow up to `x^((2^64)-1)` which is an insanly huge polynomial term. Trying to do polynomial division to such a term would probably take forever. I think I am misunderstanding something. – Silicomancer Aug 27, 2016 at 13:07

@Silicomancer What you're missing is the algorithmic technique of repeated squaring. – Yuval Filmus Aug 27, 2016 at 14:13

1 @Silicomancer The repeated squaring algorithm is described on Wikipedia. – Yuval Filmus Aug 27, 2016 at 14:41

I think I understand that repeated squaring method. At least for calculating concrete numbers (like `x^((2^64)-1)` with `x=42`) it seems pretty easy. However I was not able to wrap my mind around how to apply it to the above problem. I do not have a concrete x. I could probably convert `x^((2^n)-1)` to some nested exponential expression (being faster to calculate) for a given `n`. But in which way would that help me to do the desired polynomial division `x^((2^n)-1) / P(x)`? – Silicomancer Aug 27, 2016 at 18:46

1 @Silicomancer You just repeatedly square x modulo $P(x)$, and then multiply things out (modulo $P(x)$) until you get $x^{2^n-1} \pmod{P(x)}$. If you want to know more, you are welcome to ask another question. – Yuval Filmus Aug 27, 2016 at 23:50