Mathematical Notes
2018-06-30

Testing for perfect powers

Sebastià Xambó-Descamps

Abstract

In this note we study D. Bernstein's algorithm [1] for testing whether an odd integer $n > 1$ is a perfect power and explain its PYECC implementation in detail.

1 Basic routines

1.1 (The arithmetical routines $\mathsf{mult}(n, k, b)$, $\mathsf{quot}(n, k, b)$ and $\mathsf{pow}(n.k, b)$)**.** If n, k, b are positive integers, these functions evaluate nk, n/k (assuming k odd), and n^k modulo 2^b. Their PYECCdefinition is as follows:

```
def mult(n,k,b):
    Z = Zn(2**b)
    n = n>>Z; k = k>>Z
    return lift(n*k)

def div(n,k,b):
    Z = Zn(2**b)
    n = n>>Z; k = k>>Z
    return lift(n/k)

def pow(n,k,b):
    return power(n,k,2**b)
```

The latter function is just a special case of $\mathsf{power}(n, k, m)$, which is defined as $(n \bmod m)^k$. For example, $73^5 \bmod 10 = 3^5 \bmod 10 = 3(3^2)^2 \bmod 10 = 3 \bmod 10$.

1.2 (Main lemma for nroot)**.** *Let n, k, b be positive integers, n and k odd, $b > 1$. Let $b' = \lceil b/2 \rceil$ and assume that we have constructed an integer r' such that ${r'}^k n \equiv 1 \bmod 2^{b'}$. Let $r_0 = \mathsf{mult}(r', k+1, b)$, $r_1 = \mathsf{mult}(n, \mathsf{pow}(r', k+1, b), b)$, $r = \mathsf{quot}(r_0 - r_1, k, b)$. Then $r^k n \equiv 1 \bmod 2^b$.*

Proof. We have ${r'}^k n = 1 \bmod 2^{b'} j$, for some integer j. Since $2b' \geqslant b$, we also have $2^{2b'} \equiv 0 \bmod 2^b$. Now the binomial theorem allows us to write $(k - 2^{b'} j)^k \equiv k^k - k 2^{b'} j k^{k-1} = k^k(1 - 2^{b' j}) \bmod 2^b$. On the other hand, $r_0 \equiv (k+1) r' \bmod 2^b$, $r_1 \equiv n {r'}^{k+1} \bmod 2^b$ and $rk \equiv r_0 - r_1 \equiv r'(k + 1 - n{r'}^k) \equiv r'(k - 2^{b'} j) \bmod 2^b$. It follows that $k^k r^k n \equiv {r'}^k n (k - 2^{b'} j)^k \equiv (1 + 2^{b'} j) k^k (1 - 2^{b'} j) \equiv k^k (1 - 2^{2b'} j^2) \equiv k^k \bmod 2^b$. Sin k^k is odd, we get $r^k n \equiv 1 \bmod 2^b$, as claimed. □

1.3 (The function nroot)**.** For odd positive integers n and k, and a positive integer b, the function $\mathsf{nroot}(n, k, b)$ computes an integer $r < 2^b$ such that $r^k n \equiv 1 \bmod 2^b$. By the lemma above, this function can be defined as follows:

```
def nroot(n,k,b):
    # Assumes that n and k are odd
    if b==1: return 1
    B = []
    while b>1:
        B = [b]+B
        if b%2: b+=1
        b = b//2
    r = 1
    for b in B:
        r0 = mult(r,k+1,b)
        r1 = mult(n,pow(r,k+1,b),b)
        r  = quot(r0-r1,k,b)
    return r
```

1.4 (The function sqroot)**.** If n is an odd integer and b a positive integer, $r = \mathsf{sqroot}(n, b)$ is 0 if there is no odd integer i such that $i^2 n \equiv 1 \bmod 2^{b+1}$ and otherwise it satisfies $r^2 n \equiv 1 \bmod 2^{b+1}$.

The way to write code for such a function is similar to the one followed in the case of $\mathsf{nroot}(n, k, b)$, but with some additional subtleties. After listing the resulting code, we will study the mathematics that justifies it.

```
def sqroot(n,b):
    # Assumes that n is odd
    if b==1:
        if (n%4)==1: return 1
        else: return 0
    if b==2:
        if (n%8)==1: return 1
        else: return 0
    B = []
    while b>2:
        B = [b]+B
        if b%2: b = (b+1)//2
        else: b = 1+b//2
    r = 1
    for b in B:
        r0 = mult(r,3,b+1)
        r1 = mult(n,pow(r,3,b+1),b+1)
        r = ((r0-r1)//2)%(2**b)
        if r==0: return 0
    return r
```

To prove that this code yields what has been declared in the statement, let us first deal with the cases $b = 1$ and $b = 2$ in turn, which will justify the lines preceding the assignment B = [ ] and hence that it can be assumed, from there on, that $b > 2$.

If $b = 1$, then $2^{b+1} = 4$ and the congruence $r^2 n \equiv 1 \bmod 4$ has the solution $r = 1$ if $n \equiv 1 \bmod 4$ and no solution if $n \equiv 3 \bmod 4$.

Similarly, if $b = 2$, then $2^{b+1} = 8$ and the congruence $r^2 n \equiv 1 \bmod 8$ has the solution $r = 1$ if $n \equiv 1 \bmod 4$ and no solution if $n \equiv 3, 5, 7 \bmod 8$, for $r^2 \equiv 1 \bmod 8$ for $r = 1, 3, 5, 7$.

Now the way to proceed in the case $b > 2$ will turn out to be a straightforward application of next result.

1.5 (Main lemma for sqroot)**.** *Let n and b be positive integers, n odd and $b > 2$, and define $b' = \lceil (b+1)/2 \rceil$. Assume that we have constructed an integer r' such that*

$$r'^2 n \equiv 1 \bmod 2^{b'+1},$$

or $r' = 0$ if the there is no integer j such that $j^2 n \equiv 1 \bmod 2^{b'+1}$. Let $r_0 = \mathsf{mult}(r', 3, b+1)$, $r_1 = \mathsf{mult}(n, \mathsf{pow}(r', 3, b+1), b+1)$, $r = (r_0 - r_1)/2 \bmod 2^b)$. Then $r = 0$ if there is no integer j such that $j^2 n \equiv 1 \bmod 2^{b+1}$ and otherwise $r^2 n \equiv 1 \bmod 2^{b+1}$.

Proof. If $r' = 0$, then $r = 0$ and we claim that there is no integer j such that $j^2 n \equiv 1 \bmod 2^{b+1}$, for this congruence would imply, using that $b' < b$, that $j^2 n \equiv 1 \bmod 2^{b'+1}$ that cannot happen if $r' = 0$.

So suppose $r' \neq 0$ and hence that $r'^2 n \equiv 1 \bmod 2^{b'+1}$. The proof will be complete if we show that then we have $r^2 n \equiv 1 \bmod 2^{b+1}$. Indeed, the assumption implies that $r'^2 n = 1 + 2^{b'+1} j$ for some integer j. Note also that $(1 - 2^{b'} j)^2 \equiv 1 - 2^{b'+1} j \bmod 2^{b+1}$, as $(1 - 2^{b'} j)^2 = 1 - 2^{b'+1} j + 2^{2b'} j^2$ and $2b' \geqslant b+1$. Now we have (letting $\equiv$ mean $\equiv \bmod 2^{b+1}$)

$$r_0 \equiv 3r', \quad r_1 \equiv r'^3 n, \quad 2r \equiv r_0 - r_1 \equiv r'(3 - r'^2 n) \equiv r'(2 - 2^{b'+1} j) \equiv 2r'(1 - 2^{b'} j).$$

Since $2i \equiv 2j$ implies that $i^2 \equiv j^2$ (see the Remark below), we also have

$$r^2 \equiv r'^2 ((1 - 2^{b'} j))^2 \equiv r'^2 (1 - 2^{b'+1} j).$$

Therefore

$$r^2 n \equiv r'^2 n (1 - 2^{b'+1} j) \equiv (1 + 2^{b'+1} j)(1 - 2^{b'+1} j) \equiv 1 - 2^{2b'+2} j \equiv 1.$$

Remark. Indeed, $i - j$ is divisible by 2^b, so $i^2 - j^2 = (i+j)(i-j)$ is divisible by 2^b. But i and j have the same parity, because $b \geqslant 1$, so $i + j$ is even and hence $i^2 - j^2$ is divisible by 2^{b+1}. □

1.6 (To test whether an odd integer n is a k-th power for a given $k \geqslant 2$)**.** The following listing displays the code of the function is_power(n,k) that tests whether the odd integer n is a kth power, where $k \geqslant 2$ is odd or 2.

```
def is_power(n,k):
    # Assumes that n is odd and either k=2 or k>2 and odd.
    f = blen(2*n)
    (q,r)=(f//k,f%k)
    if r==0: b=q
    else: b=q+1
    y = inverse(n,2**(b+1))
    if k==2:
        r = sqroot(y,b)
        if r==0: return 0
    else:
        r = nroot(y,k,b)
```

```
    if power_check(n,r,k):
        return r
    if k==2 and power_check(n,2**b-r,k):
        return 2**b-r
    return 0
```

1.7 (To test whether an odd integer $n > 1$ is a perfect power)**.** For an odd integer $n > 1$, the function is_perfect_power(n) produces a pair of integers (x, p). If n is not a perfect power, this pair is equal to $(n, 1)$. Otherwise, p is prime and $n = x^p$.

```
def is_perfect_power(n):
    # assume n>1 and odd
    f = blen(2*n)
    b = qceiling(f,2)
    y = nroot(n,1,b+1)
    P = primes_less_than(f)
    for p in P:
        x = is_power(n,p)
        if x>0: return (x,p)
    return (n,1)
```

References

[1] D. J. Bernstein, "Detecting perfect powers in essentially linear time," *Mathematics of Computation*, vol. 67, no. 223, pp. 1253–1283, 1998.