Integer-Size Mod-2 Polynomial Degree

```
intmod2polydegree( p ):
[0] Initialize degree = -1
[1] While p>0, do [2] to [3]
[2]    Bitshift right p >>= 1
[3]    Increment degree += 1
[4] Return degree
```

The division theorem for ordinary polynomials applies to mod-2 polynomials:

Theorem 6.14 *For any mod-2 polynomials p_1 and $p_2 \neq 0$, there are unique mod-2 polynomials q, r such that $p_1(t) = p_2(t)q(t) + r(t)$ and $0 \leq \deg r < \deg p_2$.*

Proof: For existence, we construct the quotient q and remainder r using synthetic division. The following implementation produces arrays of 0s and 1s representing the mod-2 polynomials q, r, given input arrays `p1[0]`,...,`p1[d1]` and `p2[0]`,...,`p2[d2]`:[8]

Mod-2 Polynomial Quotient and Remainder

```
mod2polydivision( p1[], d1, p2[], d2 ):
[0] If d1<d2, then let dq = -1 and let dr = d1
[1] Else do [2] to [5]
[2]    Let dq = d1-d2 and let dr = d2-1
[3]    For d=d1 down to d2, do [4] to [5]
[4]       If p1[d]==1, then do [5]
[5]          For n=0 to d2-1, replace p1[n+d-d2] ^= p2[n]
[6] Let dr = mod2polydegree(p1[], dr)
[7] Return p1[], dq, dr
```

Upon termination, the returned array p_1 will contain the coefficients of the remainder and quotient mod-2 polynomials as subarrays: `p1[0]`, ..., `p1[dr]` will be r, while `p1[d2]`, ..., `p1[d2+dq]` will contain q. If the returned value $dr = -1$, then the remainder r is zero and its part of the array is not present. Likewise, if $dq = -1$, then the quotient q is zero and it is not present in the returned array p_1.

For uniqueness, suppose that $p_1(t) = p_2(t)q_1(t) + r_1(t) = p_2(t)q_2(t) + r_2(t)$ with $0 \leq \deg r_1 < \deg p_2$ and $0 \leq \deg r_2 < \deg p_2$. Then $p_2(t)[q_2(t) - q_1(t)] = r_1(t) - r_2(t)$. Suppose toward contradiction that $q_1 \neq q_2$. Then $p_2[q_2 - q_1] \neq 0$, so we may compute mod-2 degrees: $\deg(r_1 - r_2) = \deg(p_2[q_2 - q_1]) = \deg p_2 + \deg(q_2 - q_1) \geq \deg p_2$, but this contradicts $\deg(r_1 - r_2) \leq \max\{\deg r_1, \deg r_2\} < \deg p_2$. Hence we must have $q_1 = q_2$, so $r_1 - r_2 = p_2[q_2 - q_1] = 0$, so $r_1 = r_2$. □

We will reuse symbols and denote the quotient and remainder by $p(t)/q(t)$ and $p(t) \,\%\, q(t)$.

[8] This function assumes that the denominator polynomial p_2 is nonzero, or else the results will be nonsense. How could this assumption be tested in practice?

/, %	1	2	3	4	5	6	7	8
1	1,0	0,1	0,1	0,1	0,1	0,1	0,1	0,1
2	2,0	1,0	1,1	0,2	0,2	0,2	0,2	0,2
3	3,0	1,1	1,0	0,3	0,3	0,3	0,3	0,3
4	4,0	2,0	3,1	1,0	1,1	1,2	1,3	0,4
5	5,0	2,1	3,0	1,1	1,0	1,3	1,2	0,5
6	6,0	3,0	2,0	1,2	1,3	1,0	1,1	0,6
7	7,0	3,1	2,1	1,3	1,2	1,1	1,0	0,7
8	8,0	4,0	7,1	2,0	2,2	3,2	3,1	1,0

Table 6.4: Quotients q and remainders r from division $y(t) = q(t)x(t) + r(t)$ of mod-2 polynomials x, y expressed as base 10 versions of their coefficient bit strings. Here x is at the top of the column and y is at the left of the row.

If both the numerator and denominator polynomials fit into an integer type, then there is a more efficient mod-2 division implementation using bit-shifts:

Integer-Size Mod-2 Polynomial Quotient and Remainder

```
intmod2polydivision( p1, p2 ):
[0] Initialize q=0 and  r=p1
[1] Let sh = intmod2polydegree(r)-intmod2polydegree(p2)
[2] If sh >= 0, then do [3] to [4]
[3]    Replace r ^= (p2<<sh) and replace q ^= (1<<sh)
[4]    Go to [1]
[5] Return q, r
```

Table 6.4 shows the quotients and remainders for the first eight nonzero mod-2 polynomials.

If the remainder is zero, so that $p_1(t) = p_2(t)q(t)$, then we say that p_2 *divides* p_1. Having Theorem 6.14, we can implement Euclid's algorithm for mod-2 polynomials, so each pair $x = x(t), y = y(t)$, not both zero, will have a unique greatest common divisor not equal to zero. In the following implementation, derived from the one on page 4, we assume without loss that x is not the zero polynomial:

Euclid's Algorithm for Mod-2 Polynomials

```
mod2polygcd( x[], dx, y[], dy ):
[0-] Let dz = dx and allocate z[0],...,z[dz]
[0]  For d=0 to dx, copy z[d] = x[d]
[1]  Compute dr,dq,y[] with mod2polydivision(y[],dy,x[],dx)
[1+] Let dx = dr, and for d=0 to dr, copy x[d] = y[d]
[2]  Let dy = dz, and for d=0 to dz, copy y[d] = z[d]
[3]  If dr>=0, then go to [0]
[4]  Return y[], dy
```