TAOCP Pre-Fascicle 8A, Exercise 210 (m = 5) — disproof

TAOCP Pre-Fascicle 8A, Exercise 210 — a disproof for $m = 5$

Shisheng Li, 2026-05-21 · all code and data archived below

Verdict (m = 5): $Q_5(z)$ divides $Q_5^+(z)$ ✓; $Q_5(z)^2$ divides ✓; $Q_5(z)^3$ **does not divide**. The cube-divisibility property "$Q_m^+(z)$ is a multiple of $Q_m(z)^3$" therefore fails at $m = 5$; this answers Pre-Fascicle 8A, Exercise 210 in the negative.

1. The problem

TAOCP Pre-Fascicle 8A, Exercise 210 [HM46]:

> *"Prove or disprove that $Q_m^+(z)$ is a multiple of $Q_m(z)^3$ when $m \geq 5$."*

Here $Q_m(z)$ and $Q_m^+(z)$ are the minimum-denominator generating-function polynomials for the closed (Hamilton-cycle) and open (Hamilton-path) knight-tour counts on the $m \times n$ board:

$$F_m(z) = \sum_{n \geq 0} S_{m,n}\, z^n = P_m(z)/Q_m(z) \quad \text{(closed)},$$
$$F_m^+(z) = \sum_{n \geq 0} S_{m,n}^+\, z^n = P_m^+(z)/Q_m^+(z) \quad \text{(open)}.$$

F8A states Exercise 210 directly in the form of a *prove-or-disprove* question for $m \geq 5$ (rated HM46). The question itself implicitly suggests that the cube divisibility might hold; the answer in F8A points to D. E. Knuth, *Selected Papers on Fun & Games* (Stanford: CSLI Publications, 2011), page 532, with the remark "In the case $m = 3$, an analogous proof is on page 532 of FGbook." That page actually derives the generating-function *denominator decomposition* for $m = 3$ — $Q_3(z) = P(z^2)$ and $Q_3^+(z) = P(z^2)\,Q(z^2)\,R(z)$ for specific auxiliary polynomials P, Q, R — rather than an explicit proof of $Q_3(z)^3 \mid Q_3^+(z)$ in that form. So we read the citation as pointing to the *method* (decompose Q_m^+ as a product of factors that are powers of Q_m and other denominators), not as an

asserted lemma; for the present page we make no claim about $m = 3$. (The case $m = 2$ is degenerate: the $2 \times n$ knight graph contains no Hamiltonian cycle for $n \geq 2$, so $S_{2,n} = 0$ for all n and $Q_2(z)$ is trivial.) This page exhibits an integer counterexample at $m = 5$:

polynomial	degree	divides Q_5^+?
$Q_5(z)$	8 212	YES ✓
$Q_5(z)^2$	16 424	YES ✓
$Q_5(z)^3$	24 636	**NO**
$Q_5^+(z)$	39 630	—

That is: the **square divides but the cube does not**.

2. Don't trust, verify

Every claim on this page is the output of explicit integer arithmetic. The verifier programs and the four integer polynomials they consume ship together in the accompanying `f8ex210.tar.gz`; everything below can be re-run by any reader from that archive.

The four integer polynomials

The accompanying archive contains, under `data/`, exactly four polynomial files:

file	poly.	degree	max$\lvert c\rvert$ (dig.)	size
`Q5_integer.txt`	$Q_5(z)$	8 212	~1 656	4.9 MB
`P5_integer.txt`	$P_5(z)$	8 212	~1 656	4.9 MB
`Q5_plus_integer.txt`	$Q_5^+(z)$	39 630	~8 114	227 MB
`P5_plus_integer.txt`	$P_5^+(z)$	39 635	~8 117	227 MB

Q_5, P_5 are the closed-tour denominator and numerator; the `_plus` files are the open-tour analogues.

Each file is a list of `<index> <signed_decimal_integer>` lines, with one header comment.

The two verifiers are written in C and use FLINT (Fast Library for Number Theory) for the integer polynomial arithmetic. Each is a single small file (~150 lines including I/O); the full sources are in `code/`.

Verifier 1 — coprime test (`deg gcd(P, Q) = 0` in $\mathbf{Z}[z]$)

If `gcd(P, Q)` has positive degree, the fraction is not in lowest terms and Q is not the true minimum polynomial. We assert that the integer-polynomial gcd is a nonzero constant — equivalently a unit in $\mathbf{Q}[z]$ — via FLINT:

```
fmpz_poly_gcd(g, P, Q);
assert(fmpz_poly_degree(g) == 0);   /* gcd is a unit => coprime */
```

Build and run from the top of the tarball (the verifiers read `data/Q5_integer.txt` etc. relative to the current directory, so they must be run from the tarball's top, not from inside `code/`):

```
$ cc -O2 -fopenmp -Icode code/verify_coprime.c -lflint -lgmp -o verify_coprime
$ ./verify_coprime
[closed (P_5,  Q_5 )] loaded deg P = 8212, deg Q = 8212 (0.0s)
[closed (P_5,  Q_5 )] gcd(P, Q) degree = 0 (0.1s)
[open   (P+_5, Q+_5)] loaded deg P = 39635, deg Q = 39630 (2.5s)
[open   (P+_5, Q+_5)] gcd(P, Q) degree = 0 (0.8s)
[closed (P_5,  Q_5 )] deg gcd = 0 (P and Q are coprime in Z[z])
[open   (P+_5, Q+_5)] deg gcd = 0 (P and Q are coprime in Z[z])
```

Runs in about 4 seconds on the shipped files.

Verifier 2 — divisibility test ($Q_5^k \mid Q_5^+$ for $k = 1, 2, 3$)

The main claim, performed by direct integer polynomial division in $\mathbf{Z}[z]$:

```
fmpz_poly_set_si(Q5k, 1);
for (int k = 1; k <= 3; k++) {
    fmpz_poly_mul(Q5k, Q5k, Q5);
    /* exact Z[z] divisibility:
     * divisible == 1 iff Q_5^k divides Q+_5 over the integers */
    int divisible = fmpz_poly_divides(quot, Qp, Q5k);
}
```

Build and run:

```
$ cc -O2 -fopenmp -Icode code/verify_divtest.c -lflint -lgmp -o verify_divtest
$ ./verify_divtest
... (progress and timing lines elided) ...
  Q_5(z)^1 divides Q+_5(z) :  YES
  Q_5(z)^2 divides Q+_5(z) :  YES
  Q_5(z)^3 divides Q+_5(z) :  NO

VERDICT: the square divides but the cube does not.
         The cube-divisibility property fails at m = 5,
         answering Exercise 210 in the negative.
```

(The actual program also prints the six Z-arithmetic $R(z_0)$ certificates with bit lengths and head/tail digits; see §6.4 for that block.)

Runs in about 45 seconds. FLINT's `fmpz_poly_divides` returns 1 iff B divides A exactly in $\mathbf{Z}[z]$; no modular arithmetic, no probabilistic step. The verdict is a single boolean from *direct* bignum integer-polynomial division.

3. Where $F = P/Q$ comes from, and how we verify it

This section is the load-bearing argument behind §2. It explains how the shipped polynomials P and Q are obtained from the integer sequence $\{s_n\}$, and then how the two verifiers in §2 confirm that the pair (P, Q) is correct.

3.1 The setup: from a sequence to a rational generating function

Fix an integer sequence $\{s_n\}_{n=0,1,\ldots,N-1}$ (either the closed series $S_{5,n}$ or the open series $S^+_{5,n}$); we will use the lowercase s_n as a generic placeholder throughout §3 to keep notation uncluttered. The next step is to express the generating function $F(z) := \sum_{n\geq 0} s_n z^n$ as a single rational function $P(z)/Q(z)$ with $P, Q \in \mathbf{Z}[z]$ in lowest terms.

The central unknown is Q. If Q is fixed and the recurrence

$$s_n + c_1 s_{n-1} + c_2 s_{n-2} + \cdots + c_D s_{n-D} = 0$$

(with $Q(z) = 1 + c_1 z + \cdots + c_D z^D$) holds for every sufficiently large n, then by multiplying out $F \cdot Q$ as a formal power series we see that every coefficient of $F \cdot Q$ at power z^k with k large enough vanishes; the leftover finite collection of nonzero coefficients defines a polynomial $P(z) = F(z) \cdot Q(z)$, and $F = P/Q$ follows. So once Q is settled, P falls out without further work — the data of (P, Q) are equivalent to the data of Q plus the first $\deg P + 1$ sequence values.

The whole content of §3 is therefore three questions about Q:

(Q1) Does F admit *any* such Q? Equivalently, is F rational?
(Q2) If yes, what is the degree of the minimum-degree Q — call it $D^* = \deg Q^*$?
(Q3) How long a prefix $\{s_0, \ldots, s_{N-1}\}$ is enough to identify Q^*?

3.2 (Q1) F is rational — the transfer-matrix argument

The exercise itself states the answer: "The periodic nature of Algorithm E proves that $S_m(z) = P_m(z)/Q_m(z)$ for certain (huge) relatively prime polynomials $P_m(z)$ and $Q_m(z)$," and likewise for Algorithm E$^+$. The same conclusion is sketched in the body of the chapter where Knuth writes "This periodic pattern proves that the number CYC[$8n$] of $8 \times n$ knight's cycles satisfies a linear recurrence, and has a generating function that's a quotient of (huge) polynomials." Knuth gestures at the proof rather than writing it out; we sketch below the standard transfer-matrix argument specialised to Algorithm E.

- Algorithm E / E$^+$ maintains, at each column boundary, a *canonical-state* representation of the partial-Hamiltonian configuration. The set of all possible canonical states is finite because the canonical encoding has bounded length (10–11 cells in the frontier, with codes drawn from a bounded alphabet).
- The local knight-edge structure is column-translation invariant, so the transition operator that takes the weight vector from column boundary k to column boundary $k+1$ is a fixed linear operator.

- Hence the weight vector at column boundary k is obtained from the initial weight vector by repeatedly applying these two fixed linear operators in alternation, and $S_{m,n}$ is a fixed linear functional of this weight vector.
- A linear functional of the iterates of a finite-dimensional linear operator satisfies a linear recurrence with constant integer coefficients (Cayley–Hamilton), and the corresponding generating function $F(z)$ is rational with integer coefficients. In particular each s_n is a bounded-bit integer (Hadamard's inequality on the powers of the transfer matrix; we revisit this in §4.4), so an integer Berlekamp–Massey on the shipped sequence prefix recovers the minimum-degree integer recurrence directly — no rational or mod-p arithmetic is needed at the recovery step. We use this rationality only; we do not separately invoke the more detailed claim that the reachable canonical state set itself stabilises from one column to the next.

We accept rationality as established and let Q^* denote the (unique up to a unit in $\mathbf{Z}$) minimum-degree integer polynomial that annihilates $\{s_n\}$.

3.3 (Q2) A structural upper bound on $\deg Q^*$ from the transfer matrix

The same transfer-matrix construction also gives an explicit upper bound on $D^* = \deg Q^*$. `fastham` (resp. `fastham_open`) realises Algorithm E as a column-batched DP on a *single, parity-agnostic* state space $\mathcal{S}$: each canonical state encodes a frontier mate configuration, and one sparse linear operator $T : \mathcal{S} \to \mathcal{S}$ advances the weight vector by one column. The sequence $\{s_n\}$ is a fixed linear functional of the orbit of the initial state under T, so Q^* divides the minimum polynomial of T restricted to that orbit, and in particular

$$D^* \;\le\; |\mathcal{S}|.$$

Tightening by the row-flip symmetry σ. The $5 \times n$ knight graph admits a non-trivial involution σ: vertical row-flip ($r \leftrightarrow 4 - r$). The mate-configuration encoding is row-flip equivariant (relabel slot positions by σ and re-encode), and the column-advance transition is σ-equivariant. Consequently the transfer matrix T commutes with σ, so the state space decomposes into two T-invariant subspaces

$$V_+ \;=\; \{x : \sigma(x) = x\}, \qquad V_- \;=\; \{x : \sigma(x) = -x\},$$

with $\dim V_+ + \dim V_- = |\mathcal{S}|$. The initial state vector is σ-symmetric (the empty frontier is trivially row-flip invariant), and Algorithm E's deterministic forward DP preserves the symmetry at every column boundary. Hence the orbit of the initial state under T lies in V_+, and

$$D^* \;\le\; \dim V_+.$$

`fastham`'s `canonical()` routine works directly in the σ-quotient: it selects one σ-orbit representative per state, so its BFS enumerates exactly the σ-orbits of $\mathcal{S}$ (the raw σ-quotient count is the absorbing sink $t^\star$ plus all canonical states reached

by the forward BFS, before any subsequent reduction). The *raw* σ-quotient counts (not derived from a closed form) are:

$$|\mathcal{S}/\sigma|^{\text{closed}} \;=\; 33\,668, \qquad |\mathcal{S}/\sigma|^{\text{open}} \;=\; 204\,845.$$

Tightening further by dead-state removal and backward bisimulation. The raw σ-quotient counts above include states that contribute zero to the sequence: states from which the absorbing sink $t^\star$ is unreachable (*dead* states), and pairs of states that are indistinguishable by any forward run into $t^\star$ (*bisimilar* states). Quotienting by the largest backward bisimulation that fixes $t^\star$, and deleting dead states, produces a strictly smaller invariant subquotient of V_+ that still contains the orbit of the initial state. The mechanics are spelled out in the literate sources `fastham.w` / `fastham_open.w` (CWEB PDFs in the shipped bundle); the resulting dimensions, observed directly from the reduced state spaces, are:

$$\dim V_+^{\text{closed}} \;=\; 15\,853, \qquad \dim V_+^{\text{open}} \;=\; 99\,327.$$

These are roughly 2.1× smaller than the raw σ-quotient. This is the bound we use in §3.4.

Reproducing the count. The post-reduction dimensions above are printed by `fastham` and `fastham_open` on `stderr` immediately after the dead-removal + bisim reduction step:

```
$ ./fastham 4 2>&1 | grep -E 'BFS done|Live after|Reduced CSC'
BFS done: 33668 states, 1006030 edges, t*=..., wall=0.08s
Live after reverse-BFS: 21068 (dead: 12600) [0.01s]
Reduced CSC: 15853 states, 615780 edges
$ ./fastham_open 4 2>&1 | grep -E 'BFS done|Live after|Reduced CSC'
BFS done: 204845 states, 19898385 edges, t*=..., wall=2.4s
Live after reverse-BFS: 116622 (dead: 88223) [0.23s]
Reduced CSC: 99327 states, 11940805 edges
```

3.4 (Q3) How long a prefix is enough: Berlekamp–Massey + Massey's jump bound

The Berlekamp–Massey (BM) algorithm consumes a prefix $s_0, \ldots, s_{N-1}$ and returns the minimum-degree polynomial Q of degree $L = L_N$ that fits that prefix. The question is: when is L_N equal to D^* (the minimum-degree recurrence of the *infinite* sequence), so that the BM-output polynomial is the true minimum recurrence Q^*?

Massey's theorem describes the linear-complexity profile $L_0 \le L_1 \le \cdots \le L_N$ exactly: it is non-decreasing, and at every jump $L_n > L_{n-1}$ one has $L_n + L_{n-1} \ge n$, equivalently $L_n \ge n - L_{n-1}$. The consequence we use:

Forbidden range. Suppose the full sequence has $D^* > L_N$ (so the BM polynomial is a "lucky fit" that breaks at some later index $n_{\text{jump}} > N$). At the first such

jump, $L_{n_{\text{jump}}} + L_{n_{\text{jump}}-1} \geq n_{\text{jump}}$ with $L_{n_{\text{jump}}-1} = L_N$ (no jumps between N and n_{jump}), so $D^* \geq L_{n_{\text{jump}}} \geq n_{\text{jump}} - L_N$. Hence

$$n_{\text{jump}} \;\leq\; L_N + D^*.$$

The simple sufficient bound. We do not, in general, know L_N in advance — it is precisely the output of BM. But we always have $L_N \leq D^*$. So $L_N + D^* \leq 2D^*$, and the forbidden-range inequality gives the prefix-length-only sufficient criterion

$$N \;\geq\; 2D^* \quad\Longrightarrow\quad \text{no jump beyond } N \;\Longrightarrow\; L_N = D^* \text{ and } Q = \pm Q^*.$$

Substituting any a-priori upper bound $D^* \leq D^*_{\max}$ gives the *operational* bound

$$\boxed{N \;\geq\; 2\,D^*_{\max}}$$

which depends only on the structural ceiling on D^*, not on L_N.

Using the dead-removal + backward-bisim tightened structural ceiling $D^*_{\max} = \dim V_+$ from §3.3 gives the required prefix length:

case	$\dim V_+$	$N_0 = 2 \dim V_+$
closed (Q_5)	15 853	31 706
open (Q_5^+)	99 327	198 654

Either column is a complete, L-independent answer to (Q3): from the first N_0 values of the integer sequence, BM recovers Q^* uniquely. The actual sequence prefixes we ship (see §4) round N_0 up to the nearest tidy multiple: $N = 40\,000$ closed and $N = 200\,000$ open, both comfortably $\geq N_0$.

Authors' BM-derivation cross-check. Internally, we ran Berlekamp–Massey on each bignum sequence prefix and then continued BM for another $D^*_{\max}$ terms (using mod-p continuations stitched by CRT); no linear-complexity jump occurred, which by Massey's theorem confirms that the returned Q has $L_n = D^*$ for all n in the extended range, i.e. $Q = \pm Q^*$. This is a derivation artifact, not part of the reader's protocol; the reader is delivered Q directly and verifies it as described in §4.

3.5 From Q to (P, Q) and the verifiers we ship

Take the Q identified by BM. By construction, Q gives a recurrence on $\{s_n\}$ for all $n > \deg P$ (where $\deg P$ is determined by Q and the first $\deg P + 1$ sequence values). The product $F(z) \cdot Q(z)$ has its z^k coefficient given by

$$[z^k]\big(F \cdot Q\big) \;=\; s_k + c_1 s_{k-1} + \cdots + c_D s_{k-D},$$

which vanishes for $k > \deg P$ and equals a specific integer for $0 \le k \le \deg P$. So $F \cdot Q$ is a polynomial, and we call it P. The relation $F = P/Q$ is then immediate, and the *minimum-polynomial* property of Q (no smaller-degree annihilator) translates exactly into the coprimality $\gcd(P, Q) = 1$ in $\mathbf{Q}[z]$ (equivalently, $\gcd(P, Q)$ is a unit in $\mathbf{Z}[z]$).

We then need two checks on the shipped pair (P, Q):

Check (i) — the recurrence, $F \cdot Q = P$. This is the assertion that the shipped Q is an annihilator of F (and that P is the corresponding numerator). *Unlike the previous version of this paper, we now ship the bignum sequence $\{s_n\}$ itself* (truncated to lengths $N = 40\,000$ closed and $N = 200\,000$ open, both safely above N_0; hosted on S3, see §6.2). This lets the reader run a deterministic $\mathbf{Z}[z]$ identity `verify_recurrence`: compute $F \cdot Q$ via FLINT bignum polynomial multiplication and verify the resulting polynomial agrees with P on the low-degree part and is zero on the high-degree part of the available range. This is an exact identity in $\mathbf{Z}[z]$, no modular arithmetic.

The reader additionally runs a *3-way mod-p spot check* at a freshly-picked random prime p: `fastham` (resp. `fastham_open`) computes the sequence by column-batched state-space DP, `pq_seq` computes it by forward polynomial recurrence on (P, Q), and `seq_to_modp` reduces the shipped bignum sequence mod p. All three outputs must be byte-identical; this guards against any consistent error in the shipped bignum data file (which could otherwise pass `verify_recurrence` together with a maliciously matched P, Q). Details and the choice of p are in §4.3.

Check (ii) — coprimality, deterministic over $\mathbf{Z}[z]$. `verify_coprime.c` (already run in §2's Verifier 1) computes $\gcd(P, Q)$ in $\mathbf{Z}[z]$ via FLINT and asserts that the result has degree 0, i.e. is a nonzero constant. We do not assert "gcd = 1" because P need not be primitive in $\mathbf{Z}[z]$, but a constant gcd of any value still gives a unit gcd in $\mathbf{Q}[z]$, which is what (Q3) needs.

Aside. For the shipped data, Q_5 and Q_5^+ both have $q_0 = 1$, which forces $\mathrm{content}(Q_5) = \mathrm{content}(Q_5^+) = 1$. Since $\gcd_{\mathbb{Z}}(P, Q)$ must divide every coefficient of Q, it divides $\mathrm{content}(Q) = 1$, so $\gcd_{\mathbb{Z}}(P, Q) = \pm 1$ in both cases. (We do observe $\mathrm{content}(P_5) = \mathrm{content}(P_5^+) = 2$, so P itself is not primitive, but as noted this does not affect the (Q3) coprimality property.)

Together, Check (i) — now as a $\mathbf{Z}[z]$ identity (`verify_recurrence`), with a single-prime 3-way diff as a soundness check on the shipped bignum sequence — and Check (ii) confirm that the shipped (P, Q) annihilates the shipped sequence and is coprime. *Combined with $N \ge 2D^*_{\max}$ (from §3.4)*, Massey's theorem applied to the shipped sequence prefix forces $Q = \pm Q^*$. Hence Q is the integer minimum polynomial of the infinite Hamilton-count sequence, and the shipped (P, Q) is correct.

3.6 From here to the verification

§3.5 reduced the integer-level check that the shipped (P, Q) is the right pair to: one $\mathbf{Z}[z]$ identity $F \cdot Q = P$ (Check (i), executed by `verify_recurrence` on the shipped bignum sequence), plus the deterministic integer $\gcd_{\mathbb{Z}}(P, Q)$ degree (Check (ii)). §4 describes the single-prime 3-way diff (`fastham` vs. `pq_seq` vs. `seq_to_modp`) that guards the shipped bignum sequence against silent corruption, and §4.4 quantifies the false-confirmation probability of that single-prime spot check.

4. The shipped polynomials, the shipped sequences, and the reader's verification

We ship four integer polynomials — Q_5, P_5, Q_5^+, P_5^+ — *and* the two underlying bignum sequence prefixes:

$$S_{5,n} \text{ for } n = 6, 8, \ldots, 40\,000 \text{ (closed)}, \qquad S_{5,n}^+ \text{ for } n = 4, 5, \ldots, 200\,000 \text{ (open)}.$$

The closed sequence is restricted to even $n \geq 6$ (the $5 \times n$ knight graph has no Hamiltonian cycle for odd n or for very small n); the open sequence is given for all $n \geq 4$. Both lengths satisfy $N \geq 2D^*_{\max}$ from §3.4 with a comfortable margin, so by Massey's theorem the shipped sequence prefix *uniquely* identifies Q as the minimum-degree annihilator. The sequence files are hosted on S3, not in the small tarball (see §6.2 for URLs and SHA256 hashes).

4.1 What the reader downloads vs. what the reader computes

The full open-side bignum file is ~ 12 GB (gzipped) and the closed-side file is ~ 236 MB (gzipped); these are large but not impractically so for a one-time download. Once on disk, the sequence files feed two deterministic integer-level checks:

- `verify_recurrence`: $F \cdot Q = P$ in $\mathbb{Z}[z]$, exact bignum polynomial multiplication via FLINT (§4.3, step 5). This is the $\mathbb{Z}[z]$ identity that together with §3.4 ($N \geq 2D^*_{\max}$) and §3.5's $\gcd(P, Q)$ check uniquely identifies Q as the integer minimum polynomial.
- `seq_to_modp`: reduces the shipped bignum sequence mod the chosen single-prime p, producing one of the three streams in the §4.3 3-way diff. This is the spot-check that the shipped bignum file is what we claim it is.

The reader does *not* need to recompute the bignum sequence themselves; the mod-p streams of `fastham` and `pq_seq` (different algorithms, different code paths) provide independent corroboration that `seq_to_modp`'s input is correct mod p at a randomly chosen prime.

Remark on provenance. The shipped sequences were computed by running our mod-p state-space counter at many primes in parallel and Chinese-remaindering the per-prime sequences into the bignum sequence. The polynomial Q was then

obtained by Berlekamp–Massey on the bignum prefix, and P was obtained from the low-coefficient part of $F \cdot Q$. We do not give the prime list or recovery statistics here; the reader's protocol does not depend on them.

4.2 Our `fastham` and `fastham_open`

`fastham.w` and `fastham_open.w` are our two column-at-a-time mod-p counters — closed Hamiltonian cycles and open Hamiltonian paths (the open variant uses an anchor vertex), respectively. Both are CWEB literate programs in Knuth's style; `ctangle` produces the runnable C file. They read no external library beyond libc and OpenMP. Each performs the §3.3 dead-state removal + backward-bisimulation reduction *at start-up*, so the SpMV loop runs on the reduced V_+ rather than the raw σ-quotient (factor $\sim 2.1\times$ speedup on both sides). Algorithmic details and benchmarks are in the accompanying `fastham.pdf` and `fastham_open.pdf` (`cweave + pdftex` output of the respective .w files); we do not duplicate them here.

The corresponding mod-p polynomial recurrence `pq_seq.w` (also CWEB) forward-evaluates the shipped (Q, P) pair at a user-chosen prime and writes the sequence $S_n \bmod p$ in the same format `fastham` produces, enabling a byte-level `diff`. The third producer, `seq_to_modp.w`, reads the shipped bignum sequence and reduces each value mod the same prime. All three emit lines of the form "`n value`" so a single `diff` can compare them after light normalisation (filtering the closed file to even $n \geq 6$ and the open file to $n \geq 4$).

4.3 The reader's verification protocol

The reader's task is to confirm that the four shipped polynomials are indeed what they claim to be: Q_5, P_5, Q_5^+, P_5^+ in lowest terms over $\mathbb{Z}[z]$, with Q_5 (resp. Q_5^+) equal to the integer minimum recurrence polynomial of the closed (resp. open) Hamiltonian-count sequence. The new protocol has *three* ingredients:

(i) Single-prime 3-way diff of three independent mod-p producers of the same sequence (state-space DP, polynomial recurrence, bignum reduction). Pass = the shipped bignum sequence is consistent at p with the shipped polynomials and with the state-space model.

(ii) `verify_recurrence`: the deterministic $\mathbb{Z}[z]$ identity $F \cdot Q = P$ on the shipped bignum sequence prefix (via FLINT). Pass = Q annihilates the shipped sequence and P is the corresponding numerator, both as integer polynomial identities, no modular arithmetic.

(iii) `verify_coprime`: the deterministic $\gcd_{\mathbb{Z}}(P, Q) =$ unit check from §2, executed via FLINT.

With $N = 40\,000$ closed and $N = 200\,000$ open both exceeding $N_0 = 2D^*_{\max}$ from §3.4, (ii)+(iii) together force $Q = \pm Q^*$ by Massey's theorem applied to the shipped sequence. (i) is the single-prime soundness check that protects against

silent corruption of the shipped bignum file: if any one of the three producers disagrees at the chosen p, the comparison fails.

The exact command sequence. Run from the top of the unpacked tarball, after the sequence files `S5_closed_n6_40000.txt` and `S5_open_n4_200000.txt` have been downloaded from S3 and decompressed into `data/`:

```
# 1. Pick a fresh 30-bit prime.
P=$(./code/pick_prime)            # e.g. 873542317
echo "spot-check prime = $P"

# 2. Build the four mod-p producers at that prime.
cd code/
ctangle fastham.w        && cc -O3 -fopenmp -DMODP=${P}ULL \
    fastham.c       -o fastham
ctangle fastham_open.w   && cc -O3 -fopenmp -DMODP=${P}ULL \
    fastham_open.c  -o fastham_open
ctangle pq_seq.w         && cc -O3 -fopenmp -DMODP=${P}ULL \
    pq_seq.c -lgmp  -o pq_seq
ctangle seq_to_modp.w    && cc -O2          -DMODP=${P}ULL \
    seq_to_modp.c   -lgmp -o seq_to_modp
cd ..

# 3a. Closed 3-way diff at N = 40 000 (direct diff; all four
# producers emit identical "n value" lines for n = 0..N, with 0
# auto-filled for the trivial gaps).
N_CLOSED=40000
OMP_NUM_THREADS=16 ./code/fastham $N_CLOSED > A_closed.txt
OMP_NUM_THREADS=16 ./code/pq_seq \
    data/Q5_integer.txt data/P5_integer.txt $N_CLOSED B_closed.txt
OMP_NUM_THREADS=16 ./code/seq_to_modp data/S5_closed_n6_40000.txt > C_closed.txt
diff A_closed.txt B_closed.txt && diff A_closed.txt C_closed.txt
# (all three must be byte-identical: 40 001 lines)

# 3b. Open 3-way diff at N = 200 000.
N_OPEN=200000
OMP_NUM_THREADS=16 ./code/fastham_open $N_OPEN > A_open.txt
OMP_NUM_THREADS=16 ./code/pq_seq \
    data/Q5_plus_integer.txt data/P5_plus_integer.txt \
    $N_OPEN B_open.txt
OMP_NUM_THREADS=16 ./code/seq_to_modp data/S5_open_n4_200000.txt > C_open.txt
diff A_open.txt B_open.txt && diff A_open.txt C_open.txt
# (all three must be byte-identical: 200 001 lines)

# 4. Z[z] identity F * Q = P on each shipped sequence.
ctangle code/verify_recurrence.w
```

```
cc -O2 -fopenmp code/verify_recurrence.c -lflint -lgmp -o code/verify_recurrence
./code/verify_recurrence data/Q5_integer.txt \
    data/S5_closed_n6_40000.txt data/P5_integer.txt
./code/verify_recurrence data/Q5_plus_integer.txt \
    data/S5_open_n4_200000.txt data/P5_plus_integer.txt

# 5. gcd_Z(P, Q) check from sec. 2.
./code/verify_coprime
```

A run of all of the above is observed to take $\sim$ 30 minutes on a 16-thread workstation (dominated by `fastham_open`'s SpMV at $N = 200\,000$ and by `verify_recurrence`'s $F \cdot Q$ multiplication on the open side; see §6.5 for the breakdown).

Why three producers. The §3.4 mod-p argument (Massey's jump bound at prefix length $\geq 2D^*_{\max}$) already ensures that any one of the three producers alone uniquely determines Q mod p at the chosen prime. The redundancy here is for *cross-validation*: an error in any single producer (an off-by-one in `fastham`, a wrong coefficient in the shipped polynomial fed to `pq_seq`, a corrupted bignum in the shipped sequence file) shows up as a non-empty `diff` immediately. §4.4 bounds the residual probability that all three agree at a random p despite the shipped data being wrong.

4.4 False-confirmation probability of the single-prime spot check

What we bound. The single-prime 3-way `diff` (§4.3) compares the shipped bignum sequence $S' = (s'_n)_{n=0}^{N}$, reduced mod a random prime $p \in [2^{29}, 2^{30})$, against the mod-p sequence produced by `fastham` (state-space DP) and by `pq_seq` (forward recurrence on the shipped (Q, P)). Let

$$S = (s_n)_{n=0}^{N}$$

be the *true* Hamilton-count sequence — i.e. what `fastham` would print if its accumulator were a bignum instead of mod-p. The deterministic verifiers `verify_recurrence` and `verify_coprime` pin the shipped (Q, P) to be the unique minimum-polynomial recovery of S' in $\mathbb{Z}[z]$ (see §3.5 and the close of §3.4). So the only remaining hypothesis the 3-way `diff` attacks is *whether the shipped S' actually equals S over $\mathbb{Z}$*. This subsection bounds

$$\Pr\big[\text{3-way } \texttt{diff} \text{ passes at random } p \;\big|\; S' \neq S \text{ in } \mathbb{Z}\big].$$

That is: a malicious or accidentally corrupted shipped sequence S' happens to agree with the true sequence mod a freshly drawn random prime p.

Setup. Define the discrepancy polynomial

$$\Delta_n \;:=\; s'_n - s_n \in \mathbb{Z}, \qquad n = 0, 1, \ldots, N.$$

$S' \neq S$ iff $\Delta_n \neq 0$ for some n. The 3-way `diff` at prime p tests

$$s'_n \equiv s_n \pmod{p} \quad \text{for } n = 0, 1, \ldots, N \qquad \Longleftrightarrow \qquad p \mid \Delta_n \quad \text{for every } n \text{ with } \Delta_n \neq 0.$$

For the `diff` to pass despite a fault, the random prime p must divide *every* nonzero Δ_n. Worst case (most favourable to the adversary): exactly one nonzero Δ_n, and it is the largest one. Bounding the chance under this worst case gives an upper bound on the false-confirmation probability.

Rigorous bound on $|s_n|$. A Hamilton path on the $5 \times n$ knight graph is an ordered sequence of all $5n$ vertices with consecutive members knight-connected; a Hamilton cycle is the same plus a closing knight edge to the start. Each vertex of the $5 \times n$ knight graph has at most 8 knight neighbours (interior cells; edge/corner cells have fewer). The number of Hamilton paths starting at a fixed vertex is therefore at most 8^{5n-1}, the total number of Hamilton paths (any pair of endpoints) is at most $5n \cdot 8^{5n-1}$, and the number of Hamilton cycles is at most $5n \cdot 8^{5n-1}/2$ (each cycle has $5n$ rotations and 2 reflections). Both counts satisfy

$$|s_n| \;\le\; 8^{5n}, \qquad \log_2 |s_n| \;\le\; 15n \text{ bits}.$$

This is a rigorous, graph-theoretic upper bound; the empirical $\log_2 |s_n|$ on the shipped data grows roughly linearly at ~ 4.27 bits per n, a factor of $3.5\times$ inside the rigorous ceiling.

Bound on $|\Delta_n|$. The shipped $|s'_n|$ is whatever the file contains; we use the same graph-theoretic ceiling $|s'_n| \le 8^{5n}$ (a shipped value exceeding this ceiling is a visible corruption: its decimal width would exceed $15n/\log_2 10 \approx 4.52n$ digits, observable by a single `awk` pass over the file). Combining,

$$|\Delta_n| \;=\; |s'_n - s_n| \;\le\; |s'_n| + |s_n| \;\le\; 2 \cdot 8^{5n}, \qquad \log_2 |\Delta_n| \;\le\; 15n + 1.$$

Bound on the number of bad primes. A nonzero integer Δ has at most $\log_2 |\Delta| / \log_2 p_{\min}$ distinct prime factors in any interval whose smallest prime is $p_{\min}$. For $p \in [2^{29}, 2^{30})$, $p_{\min} = 2^{29}$, so $\log_2 p_{\min} = 29$ and

$$\#\{p \in [2^{29}, 2^{30}) : p \mid \Delta_n\} \;\le\; \frac{\log_2 |\Delta_n|}{29} \;\le\; \frac{15n + 1}{29}.$$

The total pool size is $\pi(2^{30}) - \pi(2^{29}) \approx 2.6 \cdot 10^7$ primes (prime number theorem). One uniform draw hits a bad prime with probability

$$\Pr[p \text{ is bad for } \Delta_n] \;\le\; \frac{(15\,n + 1)/29}{2.6 \cdot 10^7}.$$

Taking $n = N$ (the largest index in the shipped sequence and hence the binding case):

case	N	$\log_2 \lvert\Delta_N\rvert$ bits ($\le$)	Pr[single random p bad]

closed	40,000	600,000	$\leq 8.0 \cdot 10^{-4}$
open	200,000	3,000,000	$\leq 4.0 \cdot 10^{-3}$

The open side is the binding case.

Confidence at K independent primes. A cautious reader may rerun the §4.3 protocol at K freshly drawn primes (`pick_prime` reseeds from `/dev/urandom` on each invocation, so successive draws are practically independent). The per-prime false-confirmation probabilities multiply:

K	open-side upper bound
1	$4.0 \cdot 10^{-3}$
2	$1.6 \cdot 10^{-5}$
3	$6.4 \cdot 10^{-8}$
5	$1.0 \cdot 10^{-12}$

A single random-prime run already gives $\sim 4 \cdot 10^{-3}$ confidence on the open sequence (and $\sim 8 \cdot 10^{-4}$ for closed); $K \geq 3$ drives the bound below 10^{-7}.

Caveats and looseness of the bound.

(i) Graph-theoretic ceiling vs. realistic envelope. The $15n$-bit ceiling on $\log_2 |s_n|$ is a rigorous, fully worst-case graph bound (every vertex's ≤ 8 knight neighbours, multiplied out for all $5n$ steps); the actual sequence grows much more slowly, ~ 4.27 bits per n empirically (a factor of $3.5\times$ inside the ceiling). The bound on the bad-prime count above is therefore conservative by the same factor, and the realistic per-prime false-confirmation probability is correspondingly $\sim 3.5\times$ smaller than the table value.

(ii) Single-fault worst case. The bound takes the most adversary-friendly fault: exactly one nonzero Δ_n, at the largest n. A real corruption (a wrong line, a missing bignum digit) typically affects many Δ_n simultaneously, and the random prime then has to divide *all* of them; the probability drops further by the gcd of the individual Δ_n's, which is generically $O(1)$. The single-fault bound is therefore a useful conservative ceiling.

(iii) The bound concerns the spot check only. If the 3-way diff passes *and* `verify_recurrence` passes *and* `verify_coprime` passes, the reader has joint confidence: any of the three legs detecting a problem would trigger a failure independently. The 3-way diff covers the sequence-vs-fastham mod-p agreement; `verify_recurrence` covers the $\mathbb{Z}[z]$ identity $F \cdot Q = P$ on the shipped sequence; `verify_coprime` covers $\gcd(P, Q) = 1$. Together with $N \geq 2D^*_{\max}$ (§3.4), this triad pins the shipped (Q, P, S') as the unique recovery of the true Hamilton-count sequence up to sign.

A reader who runs the §4.3 protocol at $K = 2$ or $K = 3$ fresh primes (using `pick_prime` each time and rebuilding the four mod-p producers at the new `MODP`)

already has high worst-case confidence that the shipped sequence is correct; the realistic confidence is several orders of magnitude higher still.

5. Tools shipped for the reader's verification

The §4.3 protocol calls for three mod-p producers, two $\mathbb{Z}[z]$ verifiers, and a prime picker. We ship eight CWEB literate programs:

- `fastham.w` (closed state-space DP mod-p counter; includes the §3.3 dead-removal + backward-bisim reduction; no external library beyond libc and OpenMP)
- `fastham_open.w` (open state-space DP mod-p counter with anchor vertex; same reduction; no external library beyond libc and OpenMP)
- `pq_seq.w` (forward polynomial recurrence on (P, Q), mod p; GMP only)
- `seq_to_modp.w` (reduces a shipped bignum sequence file mod p; GMP only)
- `pick_prime.w` (random 30-bit prime picker; Miller–Rabin with the standard witness set $\{2, 7, 61\}$ which is deterministic below 2^{32}; libc only)
- `verify_recurrence.w` (FLINT-based $F \cdot Q = P$ identity on the shipped bignum sequence; FLINT, GMP)
- `verify_coprime.w` ($\gcd_{\mathbb{Z}}(P, Q)$ degree check from §2; FLINT, GMP)
- `verify_divtest.w` ($Q^2 \mid Q^+$ vs. $Q^3 \nmid Q^+$, the load-bearing divisibility test that issues the verdict; FLINT, GMP)

Algorithmic details of each are in the accompanying .pdf (`cweave + pdftex` output). Build and run commands are in §6.1.

6. Practical guide: build and run

6.1 The complete reader workflow

A reader who wants to independently verify the disproof has two classes of work, and both are required for an end-to-end witness:

(A) Confirm that the shipped (Q_5, P_5, Q_5^+, P_5^+) are the integer minimum recurrences. This is the §4.3 protocol: a single `pick_prime` draw, a 3-way mod-p `diff` of three independent producers (`fastham`/`fastham_open`, `pq_seq`, `seq_to_modp`) at that prime, an exact $\mathbb{Z}[z]$ `verify_recurrence` of $F \cdot Q = P$ on each shipped bignum sequence, and `verify_coprime`.

(B) Run `verify_divtest` on the (now confirmed) polynomials. This is the load-bearing test from §2 that issues the verdict "$Q_5^2 \mid Q_5^+$ but $Q_5^3 \nmid Q_5^+$". (A) guarantees that (B) is being fed the correct Q_5, Q_5^+.

A concrete turnkey run:

```
# ---- 0. Get the tarball and decompress the shipped sequences. ----
tar xzf f8ex210.tar.gz
cd f8ex210/
```

```
# Download the two bignum sequence files (URLs + SHA256 supplied in
# the accompanying email; place the .gz files under data/, verify
# their SHA256, and gunzip).
( cd data
  # curl -O ...     # URLs and hashes from the email
  # sha256sum -c ...
  gunzip S5_closed_n6_40000.txt.gz
  gunzip S5_open_n4_200000.txt.gz
)

# ---- 1. Pick a fresh 30-bit prime and build the 8 programs. ----
cd code/
for w in fastham fastham_open pq_seq seq_to_modp pick_prime \
         verify_recurrence verify_coprime verify_divtest ; do
    ctangle $w.w
done
cc -O2 pick_prime.c -o pick_prime
P=$(./pick_prime)
echo "spot-check prime = $P"
cc -O3 -fopenmp -DMODP=${P}ULL fastham.c       -o fastham
cc -O3 -fopenmp -DMODP=${P}ULL fastham_open.c  -o fastham_open
cc -O3 -fopenmp -DMODP=${P}ULL pq_seq.c -lgmp  -o pq_seq
cc -O2 -fopenmp -DMODP=${P}ULL seq_to_modp.c -lgmp -o seq_to_modp
cc -O2 -fopenmp verify_recurrence.c -lflint -lgmp       -o verify_recurrence
cc -O2 -fopenmp verify_coprime.c    -lflint -lgmp       -o verify_coprime
cc -O2 -fopenmp verify_divtest.c    -lflint -lgmp       -o verify_divtest
cd ..

# ---- 2. (A.1) 3-way diff on closed at N = 40 000. ----
# All four producers emit identical "n value" plain-text lines
# for n = 0..N (with 0 auto-filled for gaps), so direct diff works
# without any awk filtering.
N_CLOSED=40000
OMP_NUM_THREADS=16 ./code/fastham $N_CLOSED > A_closed.txt
OMP_NUM_THREADS=16 ./code/pq_seq \
    data/Q5_integer.txt data/P5_integer.txt $N_CLOSED B_closed.txt
OMP_NUM_THREADS=16 ./code/seq_to_modp data/S5_closed_n6_40000.txt > C_closed.txt
diff A_closed.txt B_closed.txt && diff A_closed.txt C_closed.txt

# ---- 3. (A.2) 3-way diff on open at N = 200 000. ----
N_OPEN=200000
OMP_NUM_THREADS=16 ./code/fastham_open $N_OPEN > A_open.txt
OMP_NUM_THREADS=16 ./code/pq_seq \
    data/Q5_plus_integer.txt data/P5_plus_integer.txt \
    $N_OPEN B_open.txt
```

```
OMP_NUM_THREADS=16 ./code/seq_to_modp data/S5_open_n4_200000.txt > C_open.txt
diff A_open.txt B_open.txt && diff A_open.txt C_open.txt

# ---- 4. (A.3) Z[z] identity F * Q = P. ----
./code/verify_recurrence data/Q5_integer.txt \
    data/S5_closed_n6_40000.txt data/P5_integer.txt
./code/verify_recurrence data/Q5_plus_integer.txt \
    data/S5_open_n4_200000.txt data/P5_plus_integer.txt

# ---- 5. (A.4) gcd(P, Q) check. ----
./code/verify_coprime

# ---- 6. (B) The verdict. ----
./code/verify_divtest
```

For higher confidence (single-prime spot-check probability below 10^{-8}), repeat steps 1–3 at a second `pick_prime` draw; see the §4.4 table.

6.2 Tar contents (plus the two S3-hosted sequence files)

```
f8ex210/
|-- README.txt
|-- f8ex210.tex                   this paper (LaTeX source)
|-- f8ex210.pdf                   this paper (rendered)
|-- fastham.pdf                   closed counter documentation (CWEB->PDF)
|-- fastham_open.pdf              open counter documentation
|-- pq_seq.pdf                    polynomial recurrence documentation
|-- seq_to_modp.pdf               bignum-sequence mod-p reducer
|-- pick_prime.pdf                random 30-bit prime picker
|-- verify_recurrence.pdf         F*Q == P in Z[z] (FLINT)
|-- verify_coprime.pdf            gcd(P, Q) check (FLINT)
|-- verify_divtest.pdf            divisibility test (FLINT, the verdict)
|-- code/
|   |-- fastham.w                 closed state-space DP (CWEB)
|   |-- fastham_open.w            open state-space DP, anchored (CWEB)
|   |-- pq_seq.w                  forward polynomial recurrence (CWEB, GMP)
|   |-- seq_to_modp.w             bignum sequence -> mod p (CWEB, GMP)
|   |-- pick_prime.w              30-bit prime picker (CWEB)
|   |-- verify_recurrence.w       Z[z] identity F*Q = P (CWEB, FLINT)
|   |-- verify_coprime.w          gcd_Z(P, Q) check (CWEB, FLINT)
|   |-- verify_divtest.w          Q^k | Q+ test, the verdict (CWEB, FLINT)
|   `-- poly_io.h                 shared text-polynomial parser
|
`-- data/
    |-- Q5_integer.txt            (  5 MB) closed denominator
    |-- P5_integer.txt            (  5 MB) closed numerator
```

```
    |-- Q5_plus_integer.txt        (227 MB) open denominator
    `-- P5_plus_integer.txt        (227 MB) open numerator
```

The tarball itself is about **464 MB** uncompressed. The two bignum Hamilton-count sequence files are hosted separately and must be downloaded into `data/` (URLs and SHA256 hashes are provided in the accompanying email):

```
S5_closed_n6_40000.txt.gz        236 MB compressed,  ~2 GB raw
S5_open_n4_200000.txt.gz          12 GB compressed, ~50 GB raw
```

After `gunzip`, the closed file is ~ 2 GB (19 998 lines, each up to a few thousand decimal digits) and the open file is ~ 50 GB (199 997 lines, each up to a few hundred thousand decimal digits). Both are needed as inputs to `verify_recurrence` and `seq_to_modp`.

6.3 Dependencies

```
# Debian / Ubuntu
sudo apt install build-essential libgmp-dev libflint-dev texlive-binaries

# macOS
brew install gmp flint
# (ctangle / cweave / pdftex come with MacTeX:
#    brew install --cask mactex
#  if you only need C builds, install just GMP+FLINT and use the
#  pre-generated *.c files in code/, skipping ctangle.)
```

Versions: FLINT $\geq$ 2.8 (for `fmpz_poly_divides` and `fmpz_poly_mul`); the FLINT-using programs were built and tested against FLINT 3.0.1. GMP $\geq$ 6.0. `ctangle`/`cweave` (Knuth's CWEB, shipped as part of TeX Live's `texlive-binaries` on Ubuntu and as part of MacTeX on macOS) are needed to extract the C source from each `.w` file. The `fastham` family has no external library dependencies beyond libc and OpenMP; `pq_seq` and `seq_to_modp` additionally need GMP; `pick_prime` needs only libc; the three FLINT verifiers (`verify_recurrence`, `verify_coprime`, `verify_divtest`) need FLINT and GMP.

6.4 Expected output (condensed; per-step timings and per-z_0 digit-prefix lines elided for readability)

```
$ ./pick_prime
  picked: 873542317  (after 23 candidates, seed=...)
873542317

$ ./verify_recurrence data/Q5_integer.txt \
      data/S5_closed_n6_40000.txt data/P5_integer.txt
  deg Q = 8212
  N = 19998 values
```

```
  deg P = 8212
  Multiplying F * Q via FLINT ...  (25.0s)
  (a) F*Q[k] == P[k] for k = 0 .. 8212 ?    PASS
  (b) F*Q[k] == 0 for k = 8213 .. 19997 ?   PASS
PASS: F(z) * Q(z) = P(z) holds in the available data range.

$ ./verify_recurrence data/Q5_plus_integer.txt \
      data/S5_open_n4_200000.txt data/P5_plus_integer.txt
  deg Q = 39630
  N = 199997 values
  deg P = 39635
  Multiplying F * Q via FLINT ...  (1115.7s)
  (a) F*Q[k] == P[k] for k = 0 .. 39635 ?   PASS
  (b) F*Q[k] == 0 for k = 39636 .. 199996 ? PASS
PASS: F(z) * Q(z) = P(z) holds in the available data range.

$ ./verify_coprime
[closed (P_5,  Q_5 )] deg gcd = 0 (P and Q are coprime in Z[z])
[open   (P+_5, Q+_5)] deg gcd = 0 (P and Q are coprime in Z[z])

$ ./verify_divtest
============================================================
DIVISIBILITY TEST (m = 5, exact integer polynomial division)
============================================================
  Q_5(z)^1 divides Q+_5(z) :  YES
  Q_5(z)^2 divides Q+_5(z) :  YES
  Q_5(z)^3 divides Q+_5(z) :  NO

Z-arithmetic certificates for Q_5(z)^3 not dividing Q+_5(z):
evaluate at z0 in Z and check Q+_5(z0) mod Q_5(z0)^3 != 0.

  z0 = 1   R(z0) NONZERO  (16 508 bits)
  z0 = 2   R(z0) NONZERO  (37 289 bits)
  z0 = 3   R(z0) NONZERO  (50 993 bits)
  z0 = 5   R(z0) NONZERO  (68 766 bits)
  z0 = 7   R(z0) NONZERO  (80 608 bits)
  z0 = 11  R(z0) NONZERO  (96 585 bits)

Each is a stand-alone proof that Q_5(z)^3 does NOT divide
Q+_5(z) in Z[z], by contrapositive of
"A | B in Z[z]  =>  A(z0) | B(z0) in Z for every z0 in Z".

VERDICT: the square divides but the cube does not.
         The cube-divisibility property fails at m = 5,
         answering Exercise 210 in the negative.
```

The 3-way `diff` outputs (`A_*.txt` vs `B_*.filt` vs `C_*.txt`) print nothing when they match; we observed empty diffs on 19 998 lines closed and 199 997 lines open at our default `pick_prime` draws. The actual `verify_divtest` run additionally prints per-step wall times and 40-digit head / 30-digit tail of each $R(z_0)$ as a digit fingerprint.

6.5 Resource budget

Wall times on a 16-thread workstation; closed values are ~ 2 orders smaller than the corresponding open values across all steps.

step	wall time (closed / open)	peak RAM
`pick_prime`	< 0.1 s	negligible
`fastham` / `fastham_open` (3-way diff leg A)	~ 3 s / ~ 10 min	0.2 GB / 2 GB
`pq_seq` (3-way diff leg B)	~ 6 s / ~ 90 s	negligible / 0.2 GB
`seq_to_modp` (3-way diff leg C)	~ 1 s / ~ 30 s	negligible
`verify_recurrence`	6 s / 126 s (~ 2 min)	1 GB / 12 GB
`verify_coprime`	~ 4 s (both)	0.5 GB
`verify_divtest`	~ 45 s (open only)	1 GB
end-to-end (closed + open)	~ 7 **minutes**	~ 12 **GB peak**

The open side dominates: `verify_recurrence`'s $F \cdot Q$ multiplication at degrees $(\deg F, \deg Q) = (199\,996, 39\,630)$ with ~ 8000-digit coefficients takes the bulk of the integer-arithmetic time, and `fastham_open`'s mod-p SpMV at $N = 200\,000$ dominates the mod-p leg. See `fastham.pdf` and `fastham_open.pdf` for thread-scaling tables. The shipped open sequence file is also ~ 50 GB on disk after `gunzip`.

6.6 If anything fails

1. Confirm FLINT ≥ 2.8 and GMP ≥ 6.
2. Confirm the SHA256 hashes on the two downloaded S3 sequence files match the values in §6.2.
3. Re-run `pick_prime` to draw a fresh modulus and redo the §4.3 3-way `diff` block. If `fastham`/`fastham_open`, `pq_seq`, and `seq_to_modp` all agree on the filtered output, the shipped polynomials and shipped sequence are mod-p consistent.
4. If `verify_recurrence` fails but the 3-way diff passes, the shipped sequence or polynomials are corrupted; redownload and verify SHA256.
5. Mail `daizisheng@gmail.com` with the failure and your environment.

Acknowledgements

This work rests on Donald Knuth's Stanford GraphBase and his `dynaham.w` / `dynahamp.w` / `makeboard.w` programs. Our `fastham.w` and `fastham_open.w`

reuse the marked-involution canonical state encoding from `dynaham.w`; the difference is the column-at-a-time transfer-matrix structure (with an absorbing-state sink for Hamiltonian-cycle detection) rather than vertex-at-a-time trie traversal.

Computational implementation was done with assistance from Claude (Anthropic).