Implementing the Thull-Yap fast integer GCD algorithm

F. Morain

Laboratoire d'Informatique de l'École polytechnique

CNRS

GRACE GT, 23–30/03/2021

Contents

I. Introduction: GCD is good for you!

Def. $\gcd(A, B) = \max\{d > 0, d \mid A, d \mid B\}$.

- basic arithmetic operation: $\gcd(a, b) \mid a$;
  modular inversion: $ax + by = 1$.
- rational reconstruction: $x \equiv a/b \bmod N$, $a, b = O(\sqrt{N})$;
- continued fractions: $|x - p/q| \leq 1/(2q^2) \rightsquigarrow$ Wiener, etc.;
  Cornacchia's algorithm.
- fundamental tool in asymmetric cryptanalysis (Chaum/De Jonge, etc.; LLL).
- ...

Related algorithm: reduction of binary quadratic forms.

In this talk: *integer* GCD only; many things are similar for polynomials (over a ring), also in Euclidean domains (e.g., $\mathbb{Z}[i]$), etc.

Euclid: classical set up

For $A \in \mathbb{Z}$: $\|A\| = \log_{\mathbb{B}} |A|$, $\mathbb{B} = 2^{64}$
(could work with polynomials, $\mathbb{Z}[i]$, etc.).

Euclid: Start from $\|A\| > \|B\| > 0$
$A = BQ + R$, $0 \leq \|R\| < \|B\|$; then $\gcd(A, B) = \gcd(B, R)$.

$$R_0 = A, \quad R_1 = B, \quad R_{i+2} = R_i \bmod R_{i+1}, \quad R_{n+1} = 0, \quad R_n = \gcd(A, B)$$

Remainder sequence: $(R_0, R_1, \ldots, R_n)$.

Quotient sequence: $(Q_0 = 0, Q_1, \ldots, Q_n)$, $Q_{i+1} = R_i \div R_{i+1}$.

Companion sequences:

$$(U_0, V_0) = (1, 0), (U_1, V_1) = (0, 1),$$

$$(U_{i+2}, V_{i+2}) = (U_i, V_i) - Q_{i+1}(U_{i+1}, V_{i+1}).$$

$$R_{i+2} = R_i - Q_{i+1} R_{i+1},$$

$$R_i = AU_i + BV_i.$$

Classical results (see Knuth)

Lemma 1/2. The number of division performed during the execution of Euclid's algorithm on the pair (a, b) is less than or equal to $2 \log b / \log 2 + 1$. We gain half a bit per iteration.

Thm. Let a and b be two integers uniformly distributed in the interval $[1, N]$. The number of steps of Euclid's algorithm is bounded by

$$\left\lceil \frac{\log(\sqrt{5}N)}{\log((1+\sqrt{5})/2)} \right\rceil - 2 \approx \lceil 2.078 \log N + 1.672 \rceil - 2;$$

Thm. (Gauss-Kuzmin) $p(q) = \text{Prob}(Q_n = q)$ is approximately

$$p(q) = \frac{\log(1 + 1/q) - \log(1 + 1/(q+1))}{\log 2}.$$

q	1	2	3	4
$p(q)$	0.415	0.170	0.093	0.059

Enter matrices

Forward:

$$A = BQ + R \Longleftrightarrow \begin{pmatrix} A \\ B \end{pmatrix} = \begin{pmatrix} Q & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} B \\ R \end{pmatrix}$$

or $(A;B) = M(B;R)$ for short. Note $\det(M) = -1$.

Backwards: $(B;R) = M^{-1}(A;B)$ with

$$\begin{pmatrix} B \\ R \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & -Q \end{pmatrix} \begin{pmatrix} A \\ B \end{pmatrix}$$

Notation: $\langle Q \rangle = [[Q,1],[1,0]]$.

More generally: $(A;B) = \langle Q_1 \rangle \langle Q_2 \rangle \cdots \langle Q_n \rangle \ (\gcd(A,B);0)$.

Necessary elements of complexity

M(n) = time to multiply two n-bit integers; Schönhage: M$(n) \in O(n \log n \log\log n)$; Furer, ..., Harvey and van der Hoeven M$(n) \in O(n \log n)$.

M(n,m) = time to multiply n-bit by m-bit. When $n > m$, we have (using slicing): M$(n,m) \leq (n/m)$M(m).

Traditional hyp.: M$(2n) \geq 2$M(n).

Lemma. [GaGe99, Lemma 8.2] Let $b > 0$, d in $\mathbb{N}$; $S(2n) \geq 2S(n)$ and $S(n) \geq n$ s.t.

$$T(1) = d, T(n) \leq bT(n/2) + S(n) \text{ for } n = 2^i, i \geq 1.$$

Then:

$$T(n) \leq \begin{cases} (2-2/n)S(n) + d \in O(S(n)) & \text{if } b = 1, \\ S(n)\log n + dn \in O(S(n)\log n) & \text{if } b = 2. \end{cases}$$

Algorithms

The development of fast algorithms is slow. A. Schönhage.

Pb	quadratic $O(n^2)$	subquadratic $O(\mathrm{M}(n)\log n)$
gcd	Lehmer (1938) Collins (1980) Jebelean (1995)	Knuth (1970) Schönhage (1971) Thull/Yap (1990)
		Möller (2008)
binary	Stein (1967) Sorenson (1994) Weber (1995)	Stehlé/Zimmermann (2004) Bernstein/Yang (2019)
red. qform	Rickert (1989)	Schönhage (1991)

GMP version	algorithm	year
1.3.2	binary gcd	
$\leq$ 2.0.2	gcd_1, Weber95	1996
4.3.0 – 6.2.1	Lehmer + Möller (thresh. = 400)	

II. Fast slow algorithms

Facts:

- $\gcd(A, B)$ is generally small (quite often $= 1$);
- Q_i is generally small (quite often $Q_i = 1$); suggests finding Q_i by subtraction (less costly than $\div$), or finding Q_i by consideration of $MSB(A)$ and $MSB(B)$ only.

Idea: $a = A \div 2^h$, $b = B \div 2^h$ with a and b single precision.
Compute $(a_0, a_1, \ldots, a_k, a_{k+1})$, $(q_1, \ldots, q_k)$, $(u_0, \ldots, u_{k+1})$, $(v_0, \ldots, v_{k+1})$ for some $k \geq 0$ s.t.

$$q_i = Q_i \text{ for all } i \leq k. \tag{1}$$

When this is the case

$$(u_i, v_i) = (U_i, V_i), \text{for all } i \leq k+1,$$

$$R_k = u_k A + v_k B, \quad R_{k+1} = u_{k+1} A + v_{k+1} B.$$

Pb: testing (1) cannot be done directly!

Lehmer, Collins, Jebelean

Lehmer: considers quotient seq. of $(a, b+1)$ and $(a+1, b)$.

Idea: find easy condition on a_i, q_i, u_i, v_i for (1) to be satisfied.

Collins (1980): $a_{i+1} \geq |v_{i+1}|, a_{i+1} - a_i \geq |v_{i+1} - v_i|$ for all $i \leq k$.

Thm. (Jebelean, 1995) the k-length quotient sequences of (a, b) and $(2^h a + A', 2^h b + B')$ match for any $h > 0$, $A', B' < 2^h$ iff

$$a_{i+1} \geq -u_{i+1} \text{ and } a_i - a_{i+1} \geq v_{i+1} - v_i \text{ and } i \text{ even},$$

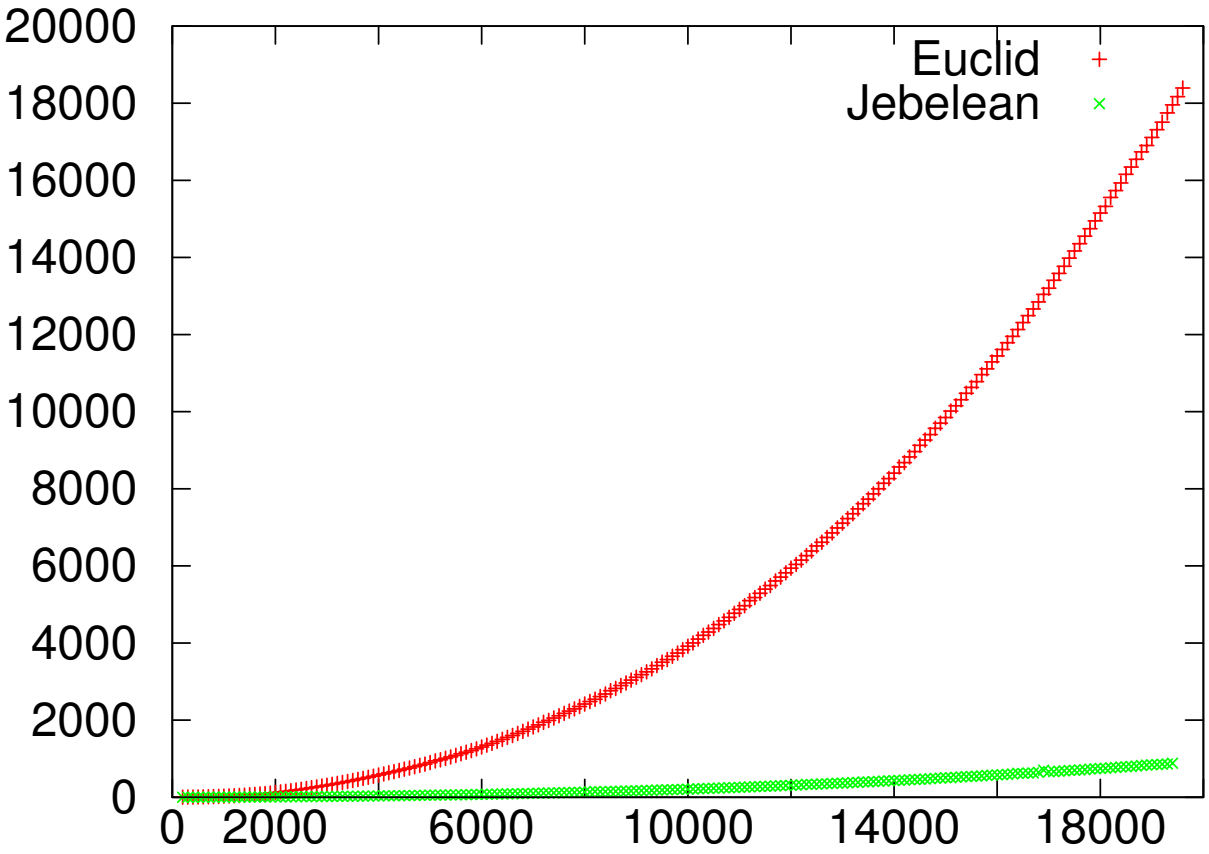
or

$$a_{i+1} \geq -v_{i+1} \text{ and } a_i - a_{i+1} \geq u_{i+1} - u_i \text{ and } i \text{ odd},$$

for all $i \leq k$. Operate on double words ($\Rightarrow$ `longlong`).

Summary: important (but constant) speedup for small operands.

Euclid vs. Jebelean



abs: number of 64-words; ord: time in ms.

III. Fast (subquadratic) algorithms

Z) Regular matrices

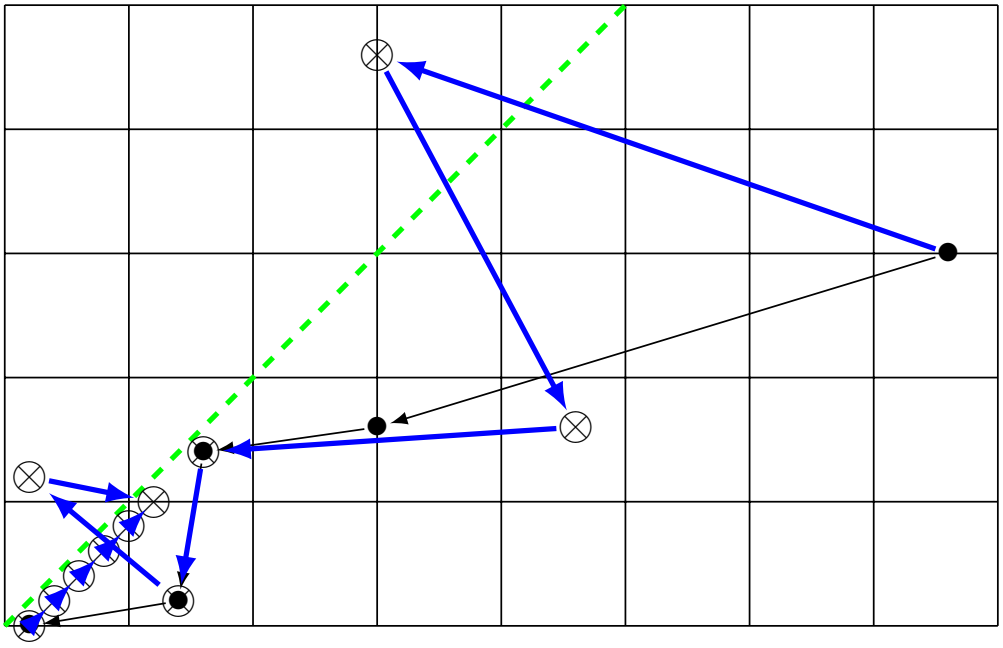
Def. $M = M_1 M_2 \ldots M_k$ with $k \geq 0$, where $M_i = \langle q_i \rangle$ for $q_i > 0$. By extension, the empty product $E = [[1,0],[0,1]]$ is also regular.

Fundamental prop. Let a, b s.t. $\|a\| > \|b\| \geq 0$ and a', b' be integers. The following are equivalent:

(i) a', b' are consecutive elements in the Euclidean remainder sequence of a, b;

(ii) there is a regular matrix M s.t. $(a;b) = M(a';b')$ and $\|a'\| > \|b'\| > 0$.

Example



1: (76, 30); 2: (30, 16); 3: (16, 14); 4: (14, 2); 5: (2, 0)

Rem. $r = \min\{|a - bx|, x \in \mathbb{Z}\}$.

In blue: iterating $\gcd(A, B) = \gcd(A - B, B)$.

A) General setting

$\|A\| = \log_{\mathbb{B}} |A| \approx$ number of words of A;

$\|A, B\| = \max(\|A\|, \|B\|)$;

$\|M\| = \max\{\|M_{i,j}\|\}$.

SPLIT(A, p):
$\|A\| = n$, write $A = a_0 \mathbb{B}^{n-p} + a_1$, $\|a_0\| = p$, $\|a_1\| = n - p$.

a_0	a_1
p words	$n - p$ words

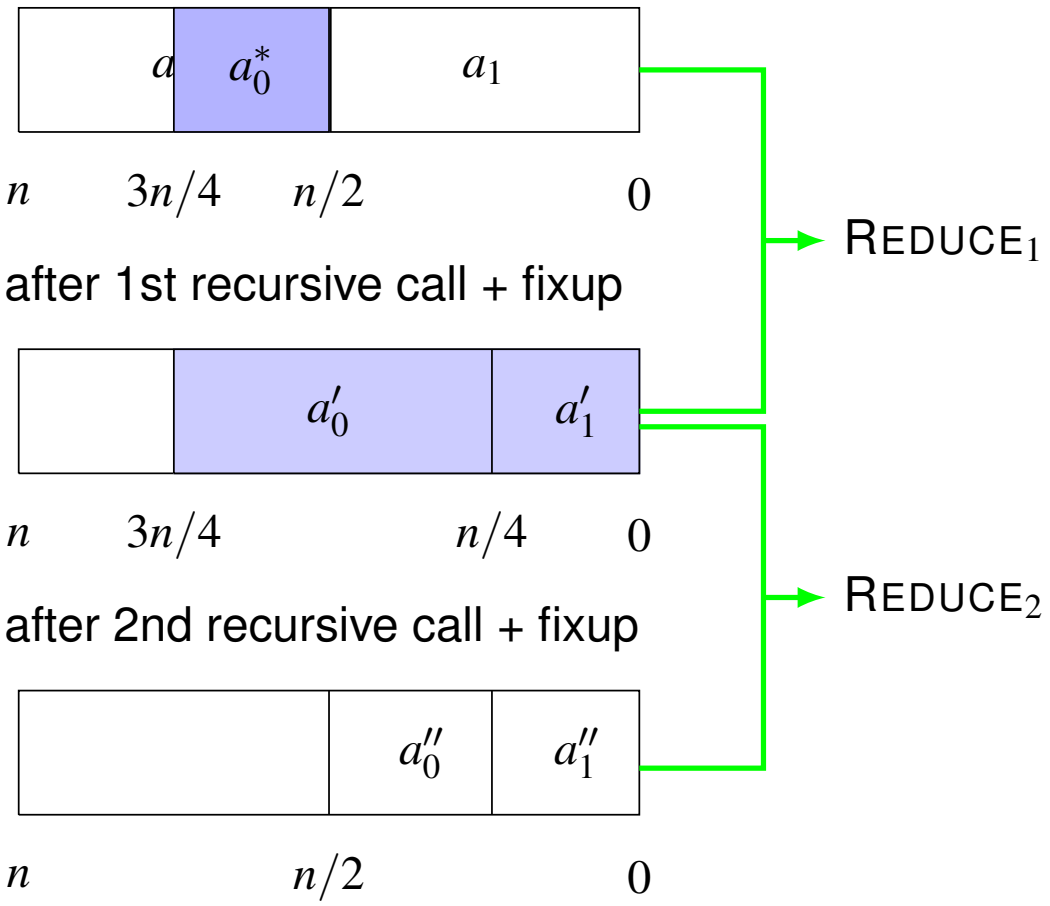
n words

Half-gcd (HGCD): compute gcd recursively on (a_0, b_0).

GCD: use HGCD($n/2$), HGCD($n/4$), etc.

HGCD pictorially

Principle: HGCD(a, b) returns a', b' of size $\approx n/2$.



A fundamental primitive: HGCD

Algorithm 1: HGCD

input : A, B with $\|A, B\| = n$
output: M, A', B' s.t. $(A; B) = M(A'; B')$, M regular, A', B' and M have $\approx n/2$ bits

if $n < n_0$ **then**
  return PartialGCD$(A, B, 2^{n/2})$;
$(M_1, A_1, B_1) \leftarrow \text{REDUCE}_1(A, B)$; // $\|A_1, B_1\| \approx 3n/4$
$(M_2, A', B') \leftarrow \text{REDUCE}_2(A_1, B_1)$; // $\|A', B'\| \approx n/2$
$M \leftarrow M_1 \cdot M_2$; // $\|M\| \approx n/4 + n/4 = n/2$ $(\star)$
return M, A', B'.

(*) not always needed!

HGCD: analysis

Rem. FIXUP contains at least

$$(A';B') \leftarrow M^{-1}(A;B) = \mathbb{B}^{n/2}(a_0^*;b_0^*) + M^{-1}(a_1;b_1).$$

REDUCE$_1$: $\|a_i,b_i\| = n/2 \rightsquigarrow \|a_0^*,b_0^*,M\| = n/4$
FIXUP(a_0^*,b_0^*,M,a_1,b_1): 4 multiplications
$\mathsf{M}(n/2,n/4) \leq 8\,\mathsf{M}(n/4)$;

REDUCE$_2$: $\|a_0',b_0'\| = n/2$, $\|a_1',b_1'\| = n/4 \rightsquigarrow \|a_0'',b_0'',M'\| = n/4$
FIXUP$(a_0'',b_0'',M',a_1',b_1')$: 4 multiplications $\mathsf{M}(n/4)$;

$M = M_1 \cdot M_2$: $\mu\,\mathsf{M}(n/4)$ for $\mu \in \{7,8\}$ (Strassen/Bodrato).

$\Rightarrow$ cost inside HGCD: $O(\mathsf{M}(n))$.

Total cost:

$$H(n) \leq 2H(n/2) + c_1\mathsf{M}(n) \overset{\text{fdal lem.}}{\Longrightarrow} H(n) \leq c_2\mathsf{M}(n)\log n.$$

Fast GCD

Algorithm 1: Fast GCD

input : $\|A\| > \|B\| > 0$; a threshold $n_0 \geq 0$
output: $\gcd(A, B)$

while $\|A, B\| > n_0$ **do**
    // $\|A, B\| = n$
    $(M, A, B) \leftarrow$HGCD(A, B);
    // $\|M, A, B\| \approx n/2$
return GCD-BASE(A, B).

Cost: $n = 2^i$

$$H(n) + H(n/2) + \cdots + H(1)$$

$$\leq c_1 \log n \sum_{j=1}^{i} \mathsf{M}(n/2^j) \leq c_1 \log n \sum_{j=1}^{i} \mathsf{M}(n/2^j) \leq c_1 \mathsf{M}(n) \log n \sum_{j=1}^{i} 1/2^j.$$

Various instantiations

- **Knuth, Schönhage**;
- **Thull-Yap:** intricate, but same formalism on polynomials/integers;
- **Möller:** by-passes the fixup problem using a relaxed notion of quotients $\Rightarrow$ cannot be used for continued fractions. Available in GMP with more tricks.
- **Stehlé/Zimmermann:** new binary reduction.
- **Bernstein/Yang:** constant time subquadratic algorithm!

B) Thull-Yap algorithm

- Fast with some tricks and a lot of technicalities for FIXUP.
- Originally $\mathbb{B} = 2$, but works for any 2^β.

Magical threshold: $m = \mathbb{T}(a) = 1 + \left\lceil \frac{\|a\|}{2} \right\rceil$.

Basic idea: THULLYAPHGCD returns $\mathcal{M}, (a'; b')$ such that $(a; b) = \mathcal{M}(a'; b')$ for which $\|a'\| \geq m > \|b'\|$.

We use $\mathcal{M} = (M, \mathcal{Q})$ for efficiency reasons and we write freely $\mathcal{M}(a; b)$, $\mathcal{M} \cdot \langle q \rangle$; $\mathcal{E} = (E, \langle\rangle)$.

SPLIT(a, b, m): for $a > b > 0$, use

$(a_0, a_1) = (1 + (a \div \mathbb{B}^m), a_0\mathbb{B}^m - a)$;
$(b_0, b_1) = (b \div \mathbb{B}^m, b \bmod \mathbb{B}^m)$;

$b_0 < a_0 < \mathbb{B}^m$, $0 < a_1 \leq \mathbb{B}^m$, $0 \leq b_i < \mathbb{B}^m$.

Algorithm 2: THULLYAPHGCD

Input : $a > b \geq 0$ two integers

Output: $\mathcal{M}$, a', b' such that $(a;b) = \mathcal{M}(a';b')$ for regular $\mathcal{M}$ with $\mathcal{M} = \mathcal{E}$ or $\|a'\| \geq \mathbb{T}(a) > \|b'\|$

1. $m \leftarrow \mathbb{T}(a)$;
if $\|a\| < \mathcal{A}$ *or* $\|b\| < m$ **then**
  return $\mathcal{E}$, a, b;
if $\|a\| < \mathcal{J}$ **then**
  return *Jebelean*(a, b, m);
2. $\mathcal{M}, a', b' \leftarrow$ REDUCE(a, b, m);
3. **if** $\|b'\| < m$ **then**
  return $\mathcal{M}$, a', b';
4. $(q, c, d) \leftarrow (a' \div b', b', a' \bmod b')$; $\mathcal{M} \leftarrow \mathcal{M} \cdot \langle q \rangle$;
4.1 **if** $\|d\| < m$ **then**
  return $\mathcal{M}$, c, d;
4.2 **if** $\|1 + (c \div \mathbb{B}^m)\| < \mathcal{A}$ **then**
  $\mathcal{M}', a', b', \leftarrow$ FIXUP$_0(c, d, m)$; **return** $\mathcal{M} \cdot \mathcal{M}'$, a', b';
5. $\ell \leftarrow \lceil \|c\| \rceil$; $k \leftarrow 2m - \ell - 1$; // $k \approx m/2$
6. $\mathcal{M}', c', d' \leftarrow$ REDUCE(c, d, k);
7. $\mathcal{M}'', a', b' \leftarrow$ FIXUP$_0(c', d', m)$; $\mathcal{M} \leftarrow \mathcal{M} \cdot (\mathcal{M}' \cdot \mathcal{M}'')$;
8. **return** $\mathcal{M}$, a', b';

REDUCE

Algorithm 3: Algorithm REDUCE

Input : $a > b \geq 0$ two integers, m an integer

Output: $\mathcal{M}$, a', b' such that $(a;b) = \mathcal{M}(a';b')$ for regular $\mathcal{M}$ with $\mathcal{M} = \mathcal{E}$ or $\|a'\| \geq m > \|b'\|$

1. $(a_0, a_1; b_0, b_1) \leftarrow$SPLIT$(a, b, m)$;
2. $t \leftarrow \mathbb{T}(a_0)$; // $t \approx m/2$
3. $\mathcal{M}_0^*, a_0^*, b_0^* \leftarrow$THULLYAPHGCD$(a_0, b_0)$;
   // $\|a_0^*\| \geq t > \|b_0^*\|$
4. $(a';b') \leftarrow (a_0^*;b_0^*)\mathbb{B}^m + \mathcal{M}_0^{*-1}(-a_1;b_1)$;
5. **if** $\|a_0\| < \mathcal{A}$ *or* $\|a'\| < m + t$ **then**
   $\mathcal{M}, a', b' \leftarrow$ FIXUP$_0(a, b, \mathcal{M}_0^*, m)$;
   else
   $\mathcal{M}, a', b' \leftarrow$FIXUP$(a', b', \mathcal{M}_0^*, m, t)$;
6. **return** $\mathcal{M}, a', b'$;

FIXUP setup (1/2)

The recursive call yields

$$(a_0^*; b_0^*) = M^{-1}(a_0; b_0), \quad \|a_0^*\| \geq t > \|b_0^*\|$$

$$M = [[p, q], [r, s]] \Longrightarrow M^{-1} = \delta[[s, -q], [-r, p]], \delta = \pm 1$$

$$\begin{aligned}(a'; b') &= M^{-1}(a_0\mathbb{B}^m - a_1; b_0\mathbb{B}^m + b_1) \\ &= \underbrace{M^{-1}(a_0; b_0)}_{\text{already known}}\mathbb{B}^m + \delta[[s, -q], [-r, p]](-a_1; b_1) \\ &= (a_0^*; b_0^*)\mathbb{B}^m + \delta[[s, -q], [-r, p]](-a_1; b_1)\end{aligned}$$

TRICK:

$$[[s, -q], [-r, p]](-a_1; b_1) = (-(sa_1 + qb_1); ra_1 + pb_1)$$

obtainable from $[[s, q], [r, p]](a_1; b_1)$ where all entries are > 0 (did I say trick?).

FIXUP setup (2/2)

From $\mathcal{M} = (M, \mathcal{Q} = \langle q_1, q_2, \ldots, q_k \rangle)$ for (a_0^*, b_0^*), we want the correct sequence $\mathcal{M}^*$ for (a, b) with new starting point $(a^*, b^*) = M^{*-1}(a, b)$ such that

$$\|a^*\| \geq m + t > \|b^*\| \geq 0.$$

We start from

$$a' = a_0^* \mathbb{B}^m - \delta(sa_1 + qb_1),$$

$$b' = b_0^* \mathbb{B}^m + \delta(ra_1 + pb_1).$$

In general: $a' > b' \geq 0$.

We may have two problems:

- Case $\delta = -1$: b' can be negative.
- Case $\delta = +1$: an inversion $b' \geq a'$ may occur.

The two cases

The case $\det(M) = -1$**:**
(-A) If $b' \geq 0$ then $M^* = M$.
(-B) Else if $\|a' + b'\| \geq m + t$, then M^* is the toggle of M.
(-C) Else if $q_k \geq 2$ then $M^* = \langle q_1, \ldots, q_{k-1}, q_k - 1 \rangle$ is the backup of the toggle of M.
(-D) Else $M^* = \langle q_1, \ldots, q_{k-3}, q_{k-2} \rangle$ is the backing up of M by two steps.

The case $\det(M) = +1$**:**
(+A) If $\|a'\| \leq \|b'\|$ then M^* is the advancement of $\langle q_1, q_2, \ldots, q_{k-1} \rangle$ by at most 2β steps.
(+B) Else if $\|a'\| < m + t$ then M^* is the backing up of M by one or two steps.
(+C) Else M^* is the advancement of M by at most 2β steps.

$\Rightarrow$ we need really need both M and $\mathcal{Q} = \langle q_1, q_2, \ldots, q_k \rangle$