1 Introduction and notation

Define the "size" of a number as the number of bits, digits or words needed to store the number: $\#x = 1 + \lfloor \log^W |x| \rfloor$, where $W = 2$ for "bitsizes", or $W = 2^L$ if we measure size as the number of words of L bits each.

2 Euclid's algorithm

Given two numbers a and b, $a > b$, define the remainder and quotient sequences r_k

$$
\begin{aligned}
r_0 &= a & r_1 &= b && (1)\\
q_k &= \lfloor r_{k-1}/r_k \rfloor &&&& (2)\\
r_{k+1} &= r_{k-1} - q_k r_k &&&& (3)\\
&&&&& (4)
\end{aligned}
$$

The sequence terminates for some τ with $r_\tau | r_{\tau-1}, r_{\tau+1} = 0$. We can also define two "co-sequences" u_k, v_k, such that $r_k = u_k a + v_k b$:

$$
\begin{aligned}
u_0 &= 1 & v_0 &= 0 && (5)\\
u_1 &= 0 & v_1 &= 1 && (6)\\
u_{k+1} &= u_{k-1} - q_k u_k & |u_{k+1}| &= |u_{k-1}| + q_k |u_k| && (7)\\
v_{k+1} &= v_{k-1} - q_k v_k & |v_{k+1}| &= |v_{k-1}| + q_k |v_k| && (8)\\
&&&&& (9)
\end{aligned}
$$

The sizes of $|u_k|, |v_k|$ are limited: $|u_k| \leq b/r_{k-1}$, $|v_k| \leq a/r_{k-1}$. We also have $|v_{k+1}| > |v_k|$ for all k. We can arrange the quotients q_k and the u_k, v_k values into matrices

$$E_k = \begin{pmatrix} 0 & 1 \\ 1 & -q \end{pmatrix} \qquad T_k = \begin{pmatrix} u_k & v_k \\ u_{k+1} & v_{k+1} \end{pmatrix} \tag{10}$$

so that $T_k = \prod_{i=0}^{k} E_i$.

For even k, the diagonal elements of T_k are non-negative and the off-diagonal elements are non-positive, and for odd k the signs are reversed. We have $\det T_k = (-1)^k$, which implies that $\gcd(u_k, v_k) = 1$.

3 Jebelean's criterion

Let $a > b > 0$, and let r_i, q_i be the quotient and remainder sequences as defined by Euclid's algorithm.

Theorem 1 *Assume that the following relations are satisfied for $0 \leq i \leq k$:*

$$r_{i+1} \geq \max(-u_{i+1}, -v_{i+1}) \tag{11}$$

$$r_i - r_{i+1} \geq \max(u_{i+1} - u_i, v_{i+1} - v_i) \tag{12}$$

Then for any $j \geq 0$, any base $W \geq 2$ and any $A', B', 0 \leq A', B' < W^j$, the sequence $q_0, q_1, \ldots, q_k$ is the quotient sequence for the numbers $A = aW^j + A', B = aW^j + B'$. In particular, if we let

$$\begin{pmatrix} C \\ D \end{pmatrix} = T_k \begin{pmatrix} A \\ B \end{pmatrix} \tag{13}$$

then $\gcd(A, B) = \gcd(C, D)$.

Proof: Let $R_i = u_i A + v_i B$. From the construction of u_i, v_i we have $R_{i+1} = R_{i-1} - q_i R_i$ for $i > 1$. We will show that $0 < R_{i+1} < R_i$, it then follows that $R_{i+1} = R_{i-1} \bmod R_i$ and therefore $q_i = \lfloor R_{i-1}/R_i \rfloor$ as desired. Substituting the definition of A, B we get

$$R_{i+1} = u_{i+1}(aW^j + a') + v_{i+1}(bW^j + B') = r_{i+1}W^j + u_{i+1}A' + v_{i+1}B' \tag{14}$$

At most one of u_{i+1}, v_{i+1} is negative. Assume that $u_{i+1} \geq 0, v_{i+1} \leq 0$ (which happens if i is odd, the even i case is similar). Then $\max(-u_{i+1}, -v_{i+1}) = |v_{i+1}|$ and

$$\begin{aligned} R_{i+1} &\geq r_{i+1}W^j - |v_{i+1}|B' \geq r_{i+1}W^j - |v_{i+1}|(W^j - 1) \\ &\geq (r_{i+1} - |v_{i+1}|)W^j + |v_{i+1}| \geq |v_{i+1}| > 0 \end{aligned} \tag{15}$$

Furthermore

$$R_i - R_{i+1} = (r_i - r_{i+1})W^j + (u_i - u_{i+1})A' + (v_i - v_{i+1})B' \tag{16}$$

Again, consider the case of odd i, so that $u_i, v_{i+1} \leq 0, u_{i+1}, v_i \geq 0$ and $\max(u_{i+1} - u_i, v_{i+1} - v_i) = u_{i+1} - u_i = |u_i| + |u_{i+1}|$. We get

$$\begin{aligned} R_i - R_{i+1} &\geq (r_i - r_{i+1})W^j - (|u_i| + |u_{i+1}|)A' \\ &\geq [(r_i - r_{i+1}) - (|u_i| + |u_{i+1}|)]\, W^j + (|u_i| + |u_{i+1}|) > 0 \end{aligned} \tag{17}$$

We have shown that $0 < R_{i+1} < R_i$, as required. $\square$.

There is also a simplified version of the criterion.

Theorem 2 *Let $n = \#a, m = \lceil n/2 \rceil$. Assume that $\#r_i > m$, or $r_i \geq W^m$, for $0 \leq i \leq k+2$. Then Jebelean's criterion is satified.*

Proof: First half of the criterion follows from $|v_{i+1}| \leq a/r_i < W^n/W^m = W^{\lfloor n/2 \rfloor} \leq W^m \leq r_{i+1}$, and similarly for $|u_{i+1}| \leq b/r_i < a/r_i$. The second half follows from

$$\begin{aligned} |v_i| + |v_{i+1}| &\leq |v_i| + q_{i+1}|v_{i+1}| = |v_{i+2}| \\ &\leq a/r_{i+1} < W^m \leq r_{i+2} = r_i - q_i r_{i+1} \leq r_i - r_{i+1} \end{aligned} \tag{18}$$

4 The hgcd function

We define a function $\text{hgcd}(a, b)$ as follows. Let $a > b > 0$, and let $n = \#a, m = \lceil n/2 \rceil$. If $\#(a \bmod b) \leq m$ then hgcd fails. Otherwise, $\text{hgcd}(a, b)$ returns four numbers $r_0 > r_1 > r_2 > r_3$ in the remainder sequence for a, b which satisfies Jebelean's simplified criterion, $\#r_0 \geq \#r_1 \geq \#r_2 > m \geq \#r_3$, together with the complete quotient sequence leading from a, b to r_3, and the corresponding coefficients so that $r_i = u_i a + v_i b$.

Then the sizes of the coefficients u_i, v_i is bounded, $\#u_i, \#v_i \leq \#\lfloor a/r_2 \rfloor \leq \#a - \#r_2 + 1 \leq n - m = \lfloor n/2 \rfloor$. We also have $r_0 = q_1 r_1 + r_2 \geq r_1 + r_2 > 2r_2 \geq 2W^m$, or equivalently, $\#\lfloor r_0/2 \rfloor > m$.

We can compute hgcd using Euclid's or Lehmer's algorithm, in time proportional to $O(n^2)$. Schönhage's algorithm is a way of computing hgcd recursively, in time $O(np(\log n))$, where p is a polynomial.

4.1 Computing hgcd on the most significant words

The central operation of the algorithm is to recursively compute hgcd on the most significant words of two given numbers. Assume we have two numbers $A > B$, with $N = \#A$. Let

$$A = aW^k + A' \qquad\qquad B = bW^k + B' \tag{19}$$

where $A', B' < W^k$ are the k least significant words of A, B, and a, b are the most significant words. Let $n = \#a = N - k$. Recursively compute $\text{hgcd}(a, b)$, and denote the returned values with r_i, u_i, v_i (Computation of hgcd can fail, if $a \bmod b$ is too small, that has to be handled specially). Construct

$$R_i = u_i A + v_i B = r_i W^k + u_i A' + v_i B' \tag{20}$$

From Jebelean's simplified criterion, and the definition of hgcd, we know that R_0, R_1 occur in the remainder sequence for A, B (often, also R_2, R_3 are correct, as will be tested below). First we consider the sizes of the R_i. Let $m = \lceil n/2 \rceil$, so that $r_0 > r_1 > r_2 \geq W^m > r_3, r_0 \geq 2W^m$, and $|u_i|, |v_i| < W^{n-m}$. We get

$$\begin{aligned}
R_0 &\geq 2W^{m+k} - (W^{n-m} - 1)(W^k - 1) && (21)\\
&= W^k(2W^m - W^{n-m}) + W^{n-m} + W^k - 1 && (22)\\
&\geq W^{k+m} && (23)\\
R_1, R_2 &\geq W^{m+k} - (W^{n-m} - 1)(W^k - 1) && (24)\\
&= W^k(W^m - W^{n-m}) + W^{n-m} + W^k - 1 && (25)\\
&\geq W^k && (26)\\
R_3 &< W^{m+k} + (W^{n-m} - 1)(W^k - 1) && (27)\\
&= W^k(W^m + W^{n-m}) - W^{n-m} - W^k + 1 && (28)\\
&< 2W^{k+m} - W^{n-m} - W^k + 1 < W^{k+m+1} && (29)
\end{aligned}$$

which can also be expressed as

$$\#R_0 > k + m \tag{30}$$

$$\#R_1, \#R_2 > k \tag{31}$$

$$\#R_3 \leq k + m + 1 \text{ if } R_3 \geq 0 \tag{32}$$

5 Lehmer's algorithm

In this subsection, we count sizes in bits. Let $\#A = N$. Let our word size be m bits, $A = a2^{N-2m} + A', B = b2^{N-2m} + B'$, with $\#a = 2m$ so that $k = N - 2m$ in the above notation. Then compute $\mathrm{hgcd}(a, b)$ and construct $R_i = u_i A + v_i B$. We find that

$$\#R_0 > k + m = N - m \tag{33}$$

$$\#R_1, \#R_2 > k = N - 2m \tag{34}$$

$$\#R_3 \leq k + m + 1 = N - m + 1 \text{ if } R_3 \geq 0 \tag{35}$$

We need to consider two different cases.

First assume that $N = mL$, i.e. A consists of L words, with the most significant bit of the most significant word set. Then

$$\#R_0 > (L - 1)m \tag{36}$$

$$\#R_1, \#R_2 > (L - 2)m \tag{37}$$

$$\#R_3 \leq (L - 1)m + 1 \tag{38}$$

$$\tag{39}$$

This means that R_0 is also L words, R_1, R_2 are L or $L - 1$ words, and that R_3 is either an L-word number where the most significant word equals one, or smaller than L words.

Next, assume the most singificant word of A has ℓ leading zero bits, $0 < \ell < m$. Then $N = mL - \ell$. We get

$$\#R_0 > (L - 1)m - \ell \tag{40}$$

$$\#R_1, \#R_2 > (L - 2)m - \ell \tag{41}$$

$$\#R_3 \leq (L - 1)m + 1 - \ell \leq (L - 1)m \tag{42}$$

$$\tag{43}$$

Now, this means that R_0 consists of L or $L - 1$ words, R_1, R_2 of $L, L - 1$ or $L - 2$ words, and that R_3 is $L - 1$ words or smaller.

5.1 Lehmer Algorithm Details

Lehmer's algorithm gives us a way of computing $\mathrm{hgcd}(A, B)$ by repeatedly computing hgcd on two word numbers. The size is reduced by approximately one word, m bits, per step. The input to the algorithm is two numbers $A > B$.

For each of the Lehmer steps below, the hgcd call can fail, and in that case we have to perform a full division and try again.

5.1.1 Initialization

Let $N = \#A, M = \lceil N/2 \rceil$ (where # denotes size in words). Let $X = A, Y = B$.

5.1.2 Main loop

We loop while either $\#X > M + 2$ or $\#X = M + 2$ and the most significant bit of the most significant word of X is set. We want to make as large a reduction step as possible.

Compute new R_k by a Lehmer step. We always get $\#R_0, \#R_1, \#R_2 > M$. Set (X, Y) to the smallest pair of R_k which are correct (by applying Jebeleans criterion).

If $\#Y \leq M$, we have to backup two steps and finish.

5.1.3 Single-word steps

Loop while we still have $\#X = M + 2$ (in which case the most significant word must have some leading zeros), or $\#X = M + 1$ and the most significant bit of the most significant word of X is set. Perform another Lehmer steps, computing new R_k. Now, we always get $\#R_0 > M$, but we can have $\#R_k \leq M$ for any of $k \geq 1$.

So we first check if $\#R_1 \leq M$. If it is, we make two back up steps and finish. Otherwise, we compute the next two remainders R_2, R_3, as long as they are correct and large. We are guaranteed that if R_3 is correct, $\#R_3 \leq M + 1$.

5.1.4 Partial words

After the full Lehmer steps, we may end up with $\#X = \#Y = M + 1$, and leading zero bits in the most significant word of X. Then a full Lehmer step could result in $\#R_0 \leq M$, which is no good.

Instead of extracting top bits corresonding to two full limbs, we let $x = \lfloor X/W^{M-1} \rfloor$ and similarly for y, and reduce until $r_0, r_1, r_2 \geq W, r_3 < W$. Then Jebelean's criterion can be applied and used exactly as in the single word case.

6 Schönhage's algorithm

Schönhage's algorithm is an efficient way of computing hgcd. It will typically consist of one recursive hgcd call, followed by one division, and then a second recursive call of hgcd. This is exactly how it works for polynomials, which have the property that $\deg(fg) = \deg f + \deg g$. For integers, we get some more complications.

6.1 The first recursive call

We now describe the algorithm. The input to the is two numbers A, B. Let $N = \#A, M = \lceil N/2 \rceil$. We will assume that $N \geq 8$ (for small N, hgcd should be computed non-recursively using Lehmer's algorithm).

Choose $k = M, n = N - k = \lfloor N/2 \rfloor$. Recursing as described above results in

$$R_i = u_i A + v_i B = r_i W^M + u_i A' + v_i B' \tag{44}$$
$$\#R_0 > M + m \tag{45}$$
$$\#R_1, R_2 > M \tag{46}$$
$$\#R_3 \leq M + m + 1 \text{ if } R_3 \geq 0 \tag{47}$$
$$\tag{48}$$

The next goal is to either terminate the algorithm, or construct two values C, D in the remainder sequence, with $\#D \leq M + m + 1 \approx 3N/4$. There are several possibilities.

1. The recursive hgcd computation failed: That means that $a \bmod b < W^m$. Then we perform one euclid step, $D = A \bmod B$. It may happen that $\#D \leq M + m + 1$, in which case we can set $C = B$ and be done with the first phase. But it's possible that D is large, in that case we have to set $(A, B) \leftarrow (B, D)$ and call hgcd again.

2. $R_1 > R_2 > R_3 \geq 0$: Then all the R_i are correct elements of the remainder sequence for A, B. If $\#R_3 \leq M$, return, otherwise set $C = R_2, D = R_3$.

3. $R_2 > R_1$: Now R_2 is too large. If $\delta = \lfloor R_2/R_1 \rfloor$, then $R_2' = R_2 - \delta R_1 = R_0 \bmod R_1$. Further more, $\delta \in \{1, 2\}$. Consider the case $u_1 \leq 0, u_2 \geq 0$, then $R_2 < 3R_1$ because

$$\begin{aligned} 3R_1 - R_2 &= (3r_1 - r_2)W^M + (3u_1 - u_2)A' + (3v_1 - v_2)B' \\ &\geq (3r_1 - q_2 r_2)W^M - (3|u_1| + q_2|u_2|)(W^M - 1) \\ &= (2r_1 + r_3)W^M - (2|u_1| + |u_3|)(W^M - 1) \\ &\geq 2W^{M+m} - (W^m + W^m)W^M + (2|u_1| + |u_3|) > 0 \end{aligned} \tag{49}$$

   Furthermore, $\#R_2' \leq M + m + 1$, because

$$\begin{aligned} 0 \leq R_2' = R_2 - \delta R_1 &\leq R_2 - R_1 \\ &= (r_2 - r_1)W^M + (u_3 - u_2)A' + (v_3 - v_2)B' \\ &< W^{M+m} + 2W^{n+M} \leq 3W^{M+m} < W^{M+m+1} \end{aligned} \tag{50}$$

   So we can set $C = R_1, D = R_2'$.

4. $R_1 > R_2, R_3 > R_2$: Now R_3 is too large. Compute $R_3' = R_1 \bmod R_2 = R_3 - \delta R_2$, where we have $\delta > 0$ but no simple upper bound for δ. Since $\#R_3' \leq \#R_3 \leq M + m + 1$, we set $C = R_2, D = R_3'$.

5. $R_1 > R_2, R_3 < 0$: For this case, we have no guaranteed bounds. If $\#R_2 \leq M + m + 1$, we can set $C = R_1, D = R_2$ and go on. Otherwise, we have to set $(A, B) \leftarrow (R1, R_2)$ call hgcd again.

For all cases above, note that we can use Jebelean's criterion as a sufficient test for each of the inequalities $R_1 > R_2 > R_3 \geq 0$. We must also keep track of values u, v, u', v' such that $C = uA + vB, D = uA + vB$.

6.2 Single Euclid step

Now we have a small D, $\#D \leq M + n + 1$ and a potentially much larger C. If $\#C > M + m + 1$, we perform a single Euclid step, as described below.

Let $E = C \bmod D$. If $\#E \leq M$, we terminate the algorithm, returning R^*, C, D, E, where R_* is the remainder preceding C in the remainder sequence.

On the other hand, if E was large enough, set $C = D, D = E$, update the u, v, u', v' coefficients accordingly, and go on to the next recursive call.

6.3 Second recursive call

By now we have computed two new numbers C, D of size approximately $3/4$ of the numbers we started with. Let $N' = \#C$ be the actual size. We have $M < N' \leq M + m + 1$. The condition that $N \geq 8$ implies that $N' < N$.

Choose $k' = 2M - N'$, let $n' = N' - k, m' = \lceil n'/2 \rceil$. With our choice of k' we get $k' + m' = M$. The bounds for N' implies $M - m - 1 < k \leq M$. We also get $n' = N' - k = 2(N' - M) \leq 2(m + 1) = 2(\lceil n/2 \rceil + 1) \leq n + 3$.

Recursively compute $\operatorname{hgcd}(c, d)$, and reuse the notation r_i, u_i, v_i for the result, and let $R_i = u_i C + v_i D$. We now get

$$\#R_0 > k' + m' = M \tag{51}$$

$$\#R_1, \#R_2 > k' \tag{52}$$

$$\#R_3 \leq M + 1 \text{ if } r_3 \geq 0 \tag{53}$$

$$\tag{54}$$

Note that unlike the first recursive call, now the only number that is guaranteed to be of size larger than M is R_0. Therefore we may need up to two backup steps to get numbers of the desired sizes. Again, there are several possibilities.

- The recursive hgcd computation failed: That means that $c \bmod d < W^{m'}$. Compute $E = C \bmod D$. If $\#E \leq M + 1$, we can return after performing one backup step or a few Euclid steps. If $\#E > M + 1$, we have to set $(C, D) \leftarrow (D, E)$ and call hgcd again.
- $R_1 < W^M$: Backup two steps and return.
- $R_1 \geq W^M, R_2 < W^M$: Backup one step and return.
- $R_1 \geq W^M, R_2 > R_1$: Compute $R_2' = R_0 \bmod R_1$. We get $\#R_2' \leq M + 1$, and may need one backup step or a few Euclid steps before returning.
- $R_1 > R_2 \geq W^M, 0 \leq R_3 < R_2$: If $R_3 \geq W^M$, we need a few additional Euclid steps. Return.
- $R_1 > R_2 \geq W^M, R_3 > R_2$: Compute $R_3' = R_1 \bmod R_2$. We get $\#R_3' \leq M + 1$, and may need a few Euclid steps before returning.
- $R_1 > R_2 \geq W^M, R_3 < 0$: We can't use R_3. If $\#R_2 \leq M + 1$, we can return after one backup step or a few Euclid steps. Otherwise, we have to set $(C, D) \leftarrow (R_1, R_2)$ and call hgcd again.

6.4 Backing up

The *backup* operation is the inverse of the Euclid step. Given two elements in the remainder sequence, say R_1, R_2, it computes the previus element $R_0 = R_2 + q_1 R_1$. The need for this backup makes the Schönhage algorithm more complicated, as we need to store the quotients. Luckily, it is possible to store the entire quotient sequence in space $O(n)$, unlike the complete remainder sequence which would need $O(n^2)$ storage. We arrange the quotient list as a stack, where new quotients are pushed as they are computed, and occasionally popped again by the backup operation.

An examination of the above algorithm shows that we need at most two backup operation per recursive call; when the quotient stack depth grows larger than that older values (corresponding to larger remainders) can be safely discarded.

7 Size analysis

For the implementation it is important to keep track of the storage needed for the recursive hgcd calls. Let N be the size of the input, $S(N)$ be the storage needed for storing the result. Assume we have a threshold L such that for $N < L$ we compute hgcd without recursion. We assume that S has the property that $S(x) + S(y) \leq S(x + y) + s_0$. This is the case, for example, if S is of the form $S(N) = c_1 N + c_2 \lfloor (c_3 N + c_4)/c_5 \rfloor$.

When $N > L$, we make two recursive calls, with sizes $n = \lfloor N/2 \rfloor, n' \leq n + 3$. Let $U(N)$ be the storage needed for the results of the recursive calls, then we have

$$U(N) \leq S(n+3) + U(n+3) \text{ where } n = \lfloor N/2 \rfloor \tag{55}$$

Let $N_0 = N, N_{j+1} = \lfloor N_j/2 \rfloor + 3$, and let k be large enough that $N_k \leq L$. Then we get

$$\begin{aligned} U(N) \leq S(N_1) + U(N_1) &\leq S(N_1) + S(N_2) + U(N_2) \\ &\leq S(N_1) + \cdots + S(N_k) \\ &\leq S(N_1 + N_2 + \cdots + N_k) + (k-1)s_0 \end{aligned} \tag{56}$$

We need to bound the N_j sequence. By induction, we get $N_j \leq N/2^j + 6$, and we choose k such that $2^k \geq \frac{N}{L-6}$. It follows that

$$N_1 + N_2 + \cdots + N_k \leq N + 6k \tag{57}$$

$$U(N) \leq S(N + 6k) + (k-1)s_0 \tag{58}$$

We also need storage for quotients, to be able to perform backup steps. When the input is of size N_j, we need to store the $2j$ most recently constructed quotients, and any new quotients we add are bounded in size by N_j. Let Q_j be a bound for the storage used in the quotient stack just before the recursive call of size N_j. Then $Q_0 = 2N_0, Q_{j+1} = Q_k + 2N_{j+1}$, so that the total size id bounded by

$$Q_k \leq 2(N_0 + N_1 + \cdots + N_k) \leq 2N + 12(k+1) \tag{59}$$

Another way to express this is to say that at an arbitrary time in the algorithm, we need to store at most 2 quotients of size larger than N_1, at most 4 of size larger than N_2, and at most $2j$ quotients of size larger than N_j.