A Simple Left-to-Right Algorithm for Minimal Weight Signed Radix-r Representations

James A. Muir*
School of Computer Science
Carleton University, Ottawa, Canada
http://www.scs.carleton.ca/~jamuir

23 October 2006 12:57:11 EDT

Abstract

We present a simple algorithm for computing the *arithmetic weight* of an integer with respect to a given radix $r \geq 2$. The arithmetic weight of n is the minimum number of nonzero digits in any signed radix-r representation of n. This algorithm leads to a new family of *minimal weight* signed radix-r representations which can be constructed using a *left-to-right* on-line algorithm. These representations are different from the ones previously discovered by Joye and Yen [3]. The idea behind our algorithm is that of choosing *closest* elements which was introduced by Muir and Stinson [5]. Our results have applications in coding theory and in the efficient implementation of public-key cryptography.

Index Terms computer arithmetic, signed radix-r representations, redundant representations, minimal weight representations, left-to-right recoding, elliptic curve cryptography.

1 Introduction

In this paper, we are concerned with *radix-r representations* where $r \geq 2$.

Definition 1. A *radix-r representation* is a *finite* sum of the form $\sum_{i \geq 0} a_i r^i$.

Since the sum above is finite, this implies that all but a finite number of the a_i are equal to zero. If n is an integer and $n = \sum_{i \geq 0} a_i r^i$, we say that $\sum_{i \geq 0} a_i r^i$ is a *radix-r representation of n*. To denote radix-r representations, the following notation is commonly used:

$$(\ldots a_3 a_2 a_1 a_0)_r = \cdots + a_3 r^3 + a_2 r^2 + a_1 r^1 + a_0.$$

Each a_i is called a *digit*.

It is well-known that every non-negative integer has a *unique* radix-r representation with digits from the set $\{0, 1, 2, \ldots, r-1\}$. Usually, when one speaks of "radix-r representations", what is meant is radix-r representations which use these digits. However, we do not need to restrict ourselves to only these digits. *Signed* radix-r representations use the digits $\{0, \pm 1, \pm 2 \ldots, \pm(r-1)\}$. We will often write $\overline{1}, \overline{2}, \ldots$ to stand for $-1, -2, \ldots$. Every integer, except for zero, has several different signed radix-r representations. In fact, every nonzero integer has an *infinite* number of such representations (e.g. $42 = (222)_4 = (1\overline{2}22)_4 =$

*J.A. Muir is supported by a Natural Sciences and Engineering Research Council of Canada Postdoctoral Fellowship.

$(1\overline{3}22)_4 = (1\overline{33}22)_4 = \cdots$). Of these *redundant* representations, it is possible to select one which has *as few nonzero digits as possible*. We say that such representations have *minimal weight*.

Signed radix-r representations of minimal weight have been studied by various subgroups of computer scientists; two, in particular, are coding theorists and cryptographers.

AN codes are a family of error detecting and correcting codes ([6] and Chapter 12 of [4] provide good overviews of AN codes). The messages for these codes are integers. Message encoding and decoding has the advantage that it is done using regular integer multiplication and division. Integers n_0, n_1 are encoded as An_0, An_1, where A is a fixed integer. Note that

$$An_0 + An_1 = A(n_0 + n_1);$$

so the sum of two codewords is a codeword. Because of this property, AN codes are useful for detecting errors in computer arithmetic. If S is the result of a computation and $S \not\equiv 0 \pmod{A}$, then an error has occurred. The distance function used in AN codes is based on signed radix-r representations.

Definition 2. The *arithmetic weight* of an integer n, with respect to a given radix r, is the number of nonzero digits in a minimal weight signed radix-r representation of n.

The distance between x and y is the arithmetic weight of $x - y$. Finding AN codes that correct more than one error requires a way of computing an integer's arithmetic weight.

A method of constructing minimal weight representations, for a general radix $r \geq 2$, was provided by Clark and Liang [1]. This construction reveals an integer's arithmetic weight. Clark and Liang call their representations *generalized nonadjacent forms* (GNAFs) since they generalize Reitwiesner's *nonadjacent forms* [7] which are defined in the case $r = 2$. Clark and Liang's algorithm builds a GNAF from least significant digit to most significant digit.

Cryptographers became interested in minimal weight representations because of their utility in making the algebraic operations used in public-key cryptography more efficient. As a concrete example, consider the Elliptic Curve Digital Signature Algorithm (ECDSA) [2]. Signing a message using ECDSA requires a computation of the form

$$nP := \underbrace{P + P + \cdots + P}_{n},$$

where n is an integer and P is an elliptic curve point. Two well-known algorithms for computing nP are presented in Figure 1; both are based on replacing n with a signed radix-r representation. The main difference in the two algorithms is that one processes the digits of $(a_{\ell-1} \ldots a_1 a_0)_r = n$ from *left to right* and the other does so from *right to left*. Note that *any* signed radix-r representation of n may be used in these algorithms. However, some work can be saved if a minimal weight representation is used. This is because, for every nonzero digit in $(a_{\ell-1} \ldots a_1 a_0)_r$, an elliptic curve addition or subtraction is performed.

The left-to-right algorithm in Figure 1 is generally preferred since the value of the variables P_i, which are set in the first "for" loop, can be precomputed off-line and stored. This is advantageous if nP must be computed for various values of n (e.g. when a number of different messages must be signed using ECDSA). Joye and Yen [3] considered the following problem. Suppose we already have a radix-r representation of n at the start of the left-to-right algorithm (this might be provided as input). They asked: if that representation does not have minimal weight, is there a way to compute the digits of a minimal weight representation of n from left to right? If so, then the digits of this minimal weight representation do not need to be stored since they can be computed inside the main "for" loop as they are needed. This would reduce the amount of memory necessary to compute nP.

In [3], Joye and Yen present an algorithm which builds a minimal weight representation from left to right. The genesis of their algorithm is in identifying and replacing "elementary blocks" in an integer's GNAF. This approach initially seems somewhat ad hoc, but their final algorithm speaks for itself.

Left-to-Right

$Q \leftarrow \infty$
for $i = 1 \ldots r-1$
  do $\begin{cases} Q \leftarrow Q + P \\ P_i \leftarrow Q \end{cases}$
$Q \leftarrow \infty$
compute $(a_{\ell-1} \ldots a_1 a_0)_r = n$
for $i = \ell - 1 \ldots 0$
  do $\begin{cases} Q \leftarrow rQ \\ \textbf{if } a_i \neq 0 \\ \quad \textbf{then} \begin{cases} \textbf{if } a_i > 0 \\ \quad \textbf{then } Q \leftarrow Q + P_{a_i} \\ \quad \textbf{else } Q \leftarrow Q - P_{-a_i} \end{cases} \end{cases}$
return Q

Right-to-Left

for $i = 1 \ldots r-1$
  do $P_i \leftarrow \infty$
$Q \leftarrow P$
compute $(a_{\ell-1} \ldots a_1 a_0)_r = n$
for $i = 0 \ldots \ell - 1$
  do $\begin{cases} \textbf{if } a_i \neq 0 \\ \quad \textbf{then} \begin{cases} \textbf{if } a_i > 0 \\ \quad \textbf{then } P_{a_i} \leftarrow P_{a_i} + Q \\ \quad \textbf{else } P_{-a_i} \leftarrow P_{-a_i} - Q \end{cases} \\ Q \leftarrow rQ \end{cases}$
$Q \leftarrow \infty, R \leftarrow \infty$
for $i = 1 \ldots r-1$
  do $\begin{cases} Q \leftarrow Q + P_i \\ R \leftarrow R + Q \end{cases}$
return R

FIGURE 1: Left-to-right and right-to-left versions of the radix-r method for computing nP.

It turns out that there is a more natural and simplistic solution to Joye and Yen's problem. It is based on a technique introduced by Muir and Stinson [5]. To construct a minimal weight representation of n from left to right, we choose an integer c, with arithmetic weight equal to one, that is *closest* to n. The purpose of the current paper is to explain this technique.

Outline The main part of the paper begins with §2 where we review some basic facts about modular arithmetic and present some notation. In §3, we explain what we mean by "choosing closest elements" and demonstrate how this can be efficiently done. Our algorithm for computing an integer's arithmetic weight is presented in §4 with the necessary proofs. In §5, we give our left-to-right algorithm for building minimal weight representations and compare it with Joye and Yen's. We end with some concluding remarks in §6.

2 Preliminaries

The quotient-remainder theorem tells us that, for any integers n and $m > 0$, there exist *unique* integers q and p such that

$$n = q \cdot m + p \quad \text{where} \quad 0 \leq p < m.$$

The value p is commonly denoted by "$n \mod m$". It follows that there also exist *unique* integers $\widehat{q}$ and $\widehat{p}$ such that

$$n = \widehat{q} \cdot m + \widehat{p} \quad \text{where} \quad -m/2 \leq \widehat{p} < m/2.$$

We will denote the value $\widehat{p}$ by "$n \bmod m$". Note that $\widehat{p}$ is equal to either p or $p - m$.

Let D_r be the set consisting of the signed radix-r digits:

$$D_r := \{0, \pm 1, \pm 2, \ldots, \pm(r-1)\}.$$

If α is a string of digits, we denote the number of nonzero digits in α by $\mathsf{wt}(\alpha)$. We refer to $\mathsf{wt}(\alpha)$ as the *weight* of the string α. The set of all strings composed of digits in D_r is denoted by ${D_r}^*$.

For an integer n, we define

$$\mathsf{wt}^*(n) := \min\{\mathsf{wt}(\alpha) : \alpha \in {D_r}^*, (\alpha)_r = n\}.$$

So, $\mathsf{wt}^*(n)$ is the minimum number of nonzero digits required to represent n using a D_r-radix-r representation. Thus, $\mathsf{wt}^*(n)$ is the arithmetic weight of n. If $\alpha \in {D_r}^*$ and $(\alpha)_r = n$ then it must be that $\mathsf{wt}(\alpha) \geq \mathsf{wt}^*(n)$; if $\mathsf{wt}(\alpha) = \mathsf{wt}^*(n)$ we say that α has *minimal weight*.

3 Closest Elements

Let $\mathcal{C}$ be the set of integers which have arithmetic weight equal to one:

$$\mathcal{C} := \left\{dr^i : i \in \mathbb{Z}, i \geq 0, d \in D_r \setminus \{0\}\right\}. \tag{1}$$

Given a nonzero integer, n, we are interested in finding an element in $\mathcal{C}$ that is *closest* to n, with respect to the standard Euclidean distance (i.e. $|x - y|$). The following lemma helps us recognize when $c \in \mathcal{C}$ is closest to n.

Lemma 3. *Let n be a nonzero integer, $i = \lfloor \log_r |n| \rfloor$ and $c \in \mathcal{C}$. If $r^i \leq |c| \leq r^{i+1}$, then c is closest to n if and only if*

$$|n - c| \leq \frac{r^i}{2}.$$

Proof. We will assume n is positive; the argument for $n < 0$ is the same except for some sign changes. We have

$$r^i \leq n < r^{i+1}.$$

Consider the elements of $\mathcal{C}$ in the interval $[r^i, r^{i+1}]$; there are exactly r of them:

$$r^i, 2r^i, 3r^i, \ldots, (r-1)r^i, r^{i+1}.$$

The distance between each pair of consecutive elements is r^i. Thus, if $c \in \mathcal{C}$ and $c \in [r^i, r^{i+1}]$, then c is closest to n if and only if $|n - c| \leq r^i/2$. □

For $n \neq 0$, consider the following function

$$\mathsf{closest}(n) := n - (n \mod r^{\lfloor \log_r |n| \rfloor}).$$

By applying *uniqueness* from the definition of "mods", we can deduce that $\mathsf{closest}(n)$ *equals a multiple of* $r^{\lfloor \log_r |n| \rfloor}$ *that is closest to* n. However, not all multiples of $r^{\lfloor \log_r |n| \rfloor}$ are in $\mathcal{C}$, and there are elements of $\mathcal{C}$ which are not multiples of $r^{\lfloor \log_r |n| \rfloor}$. Despite this, our next result shows that $\mathsf{closest}(n)$ does what we want.

Theorem 4. $\mathsf{closest}(n)$ *is an element of $\mathcal{C}$ that is closest to n.*

Proof. Let $i = \lfloor \log_r |n| \rfloor$ and $\widehat{p} = n \mod r^i$. Note that $\mathsf{closest}(n) = n - \widehat{p}$. We will assume $n > 0$; the proof for $n < 0$ is similar.

We have

$$n = \widehat{q} r^i + \widehat{p},$$

for some integer $\widehat{q}$. Since $r^i \leq n < r^{i+1}$ and $-r^i/2 \leq \widehat{p} < r^i/2$, it must be that

$$\widehat{q} \in \{1, 2, \ldots, r\}.$$

So we see that $n - \widehat{p} = \widehat{q}r^i \in \mathcal{C}$. As well, we get the bound

$$r^i + \widehat{p} \leq n \leq r^{i+1} + \widehat{p},$$

and thus $n - \widehat{p} \in [r^i, r^{i+1}]$.

To finish the argument, observe

$$\begin{aligned} & |\widehat{p}| \leq r^i/2 \\ \Longrightarrow\ & |n - (n - \widehat{p})| \leq r^i/2. \end{aligned}$$

So, by Lemma 3, $n - \widehat{p} = \mathsf{closest}(n)$ is an element of $\mathcal{C}$ closest to n. □

Note that, depending on the choice of r, some values of n can have *two* closest elements in $\mathcal{C}$. For such integers, $\mathsf{closest}(n)$ is equal to the *largest* closest element in $\mathcal{C}$.

4 Minimality

Consider the following procedure:

Algorithm W

INPUT: integers n, r where $r \geq 2$.
OUTPUT: an integer w.

$w \leftarrow 0$
while $n \neq 0$
do $\begin{cases} w \leftarrow w + 1 \\ c \leftarrow \mathsf{closest}(n) \\ n \leftarrow n - c \end{cases}$
return w

Our main result is that Algorithm W computes the arithmetic weight of n. However, before we get to that, we first need to prove that Algorithm W terminates for all inputs (i.e. we need to prove that "Algorithm" W really is an algorithm).

Fix some $r \geq 2$ and take any $n \in \mathbb{Z}$. If $n = 0$ then Algorithm W clearly terminates, so we need only consider $n \neq 0$. It suffices to show that $|n| > |n - c|$ where $c = \mathsf{closest}(n)$. Suppose to the contrary that $|n| \leq |n - c|$. Then,

$$\begin{aligned} & |n| \leq |n - c| \\ \Longrightarrow\ & |n| \leq r^{\lfloor \log_r |n| \rfloor}/2 \qquad \text{(by Lemma 3)} \\ \Longrightarrow\ & r^{\lfloor \log_r |n| \rfloor} \leq r^{\lfloor \log_r |n| \rfloor}/2 \\ \Longrightarrow\ & 2 \leq 1, \end{aligned}$$

which is a contradiction. Thus, we see that the value of the variable n in the execution of Algorithm W must go to 0, hence Algorithm W will always terminate. So, "Algorithm" W is aptly named.

Example 5. We build on an example from [3]. Here is the sequence of values that the variable c assumes during the execution of Algorithm W on input $n = 208063846$ and $r = 4$:

$$201326592,\ 8388608,\ -2097152,\ 524288,\ -65536,\ -12288,\ -768,\ 128,\ -32,\ 8,\ -2.$$

The value returned by Algorithm W is 11 which is equal to the number of integers in this sequence. ◊

If we let w be the value returned by Algorithm W on input n and r, then we claim that $w = \mathsf{wt}^*(n)$. We will prove this by showing that $w \geq \mathsf{wt}^*(n)$ and $w \leq \mathsf{wt}^*(n)$. We start with $w \geq \mathsf{wt}^*(n)$.

Let $c_1, c_2, \ldots c_w$ be the sequence of values that the variable c assumes in the execution of Algorithm W. We have

$$\begin{aligned} &c_1 \text{ is closest to } n \\ &c_2 \text{ is closest to } n - c_1 \\ &c_3 \text{ is closest to } n - c_1 - c_2 \\ &\quad\vdots \\ &c_w \text{ is closest to } n - c_1 - c_2 - \ldots - c_{w-1}. \end{aligned}$$

Because the algorithm terminates, we have

$$\begin{aligned} & n - c_1 - c_2 - \ldots - c_{w-1} - c_w = 0 \\ \Longrightarrow\ & n = c_1 + c_2 + \cdots + c_w. \end{aligned}$$

We show that the c_i form a radix-r representation of n. For each c_i, write

$$c_i = d_i r^{e_i} \quad \text{where} \quad d_i \in D_r \setminus \{0\}.$$

Lemma 6. $e_1 > e_2 > \cdots > e_w$.

Proof. Since we can replace the input n with $n - c_1$, it suffices to prove that $e_1 > e_2$. From the definition of $\mathsf{closest}(n)$ we have

$$\begin{aligned} e_1 &\in \left\{ \left\lfloor \log_r |n| \right\rfloor, \left\lfloor \log_r |n| \right\rfloor + 1 \right\}, \\ e_2 &\in \left\{ \left\lfloor \log_r |n - c_1| \right\rfloor, \left\lfloor \log_r |n - c_1| \right\rfloor + 1 \right\}. \end{aligned}$$

By Lemma 3, we have

$$\begin{aligned} & |n - c_1| \leq r^{\left\lfloor \log_r |n| \right\rfloor}/2 \\ \Longrightarrow\ & |n - c_1| < r^{\left\lfloor \log_r |n| \right\rfloor} \\ \Longrightarrow\ & r^{\left\lfloor \log_r |n - c_1| \right\rfloor} < r^{\left\lfloor \log_r |n| \right\rfloor} \\ \Longrightarrow\ & \left\lfloor \log_r |n - c_1| \right\rfloor + 1 \leq \left\lfloor \log_r |n| \right\rfloor \end{aligned}$$

Thus, the smallest value of e_1 is $\geq$ the largest value of e_2. Hence, $e_1 \geq e_2$.

Suppose $e_1 = e_2$. From the argument we gave to prove that Algorithm W terminates, we have $|n| > |n - c_1| > |(n - c_1) - c_2|$. Thus,

$$|n - c_1| > |n - (c_1 + c_2)|.$$

So the integer $c_1 + c_2$ is closer to n than c_1. However, $e_1 \geq \left\lfloor \log_r |n| \right\rfloor$ so $r^{\left\lfloor \log_r |n| \right\rfloor}$ divides $c_1 = d_1 r^{e_1}$. And, because $e_2 = e_1$, $r^{\left\lfloor \log_r |n| \right\rfloor}$ also divides c_2. This implies that $c_1 + c_2$ is a multiple of $r^{\left\lfloor \log_r |n| \right\rfloor}$. But this is a contradiction because c_1 is a multiple of $r^{\left\lfloor \log_r |n| \right\rfloor}$ that is closest to n (recall that $c_1 = n - (n \mod r^{\left\lfloor \log_r |n| \right\rfloor})$). Hence, it must be that $e_1 \neq e_2$, and thus $e_1 > e_2$, as required. □

Example 7. Continuing from our previous example with $n = 208063846$ and $r = 4$, the sequence of values that the variable c assumes during the execution of Algorithm W gives the following representation:

$$208063846 = (30\overline{2}\overline{2}2\overline{1}0\overline{3}0\overline{3}2\overline{2}\overline{2}2\overline{2})_4.$$

Notice that this representation has 11 nonzero digits. Thus $11 \geq \mathsf{wt}^*(208063846)$. ◊

Lemma 8. *Let w be the value returned by Algorithm W on input n and $r \geq 2$. Then $w \geq \mathsf{wt}^*(n)$.*

Proof. By Lemma 6, the sequence of w values that the variable c assumes during the execution of Algorithm W gives us a string $\alpha \in {D_r}^*$ with $(\alpha)_r = n$ and $w = \mathsf{wt}(\alpha)$. Thus, $w = \mathsf{wt}(\alpha) \geq \mathsf{wt}^*(n)$. □

Next, we need to prove that $w \leq \mathsf{wt}^*(n)$. The following two short lemmas will be of help.

Lemma 9. *Let $(a_{\ell-1} \ldots a_1 a_0)_r$ be any representation of n with each digit in D_r and $a_{\ell-1} \neq 0$. Let $i = \lfloor \log_r |n| \rfloor$. Then, $\ell - 1 \geq i$.*

Proof. Note that $a_{\ell-1} \neq 0$ implies $n \neq 0$ and so both $\log_r |n|$ and i are defined. Since $n = (a_{\ell-1} \ldots a_1 a_0)_r$, we have $|n| < r^\ell$. From the definition of i, we have $r^i \leq |n|$. Thus, $r^i \leq |n| < r^\ell$, which gives $r^i < r^\ell$ and the result follows. □

Lemma 10. *Let $(a_{\ell-1} \ldots a_1 a_0)_r$ be any representation of n with each digit in D_r and $a_{\ell-1} \neq 0$. Let $i = \lfloor \log_r |n| \rfloor$. Then, there are at most two possible values for $(a_{\ell-1} \ldots a_i)_r \cdot r^i$; moreover, both of these values are in $\mathcal{C}$, and one is equal to* $\mathsf{closest}(n)$.

Proof. Consider the constrained Diophantine equation

$$n = xr^i + y, \quad -r^i < y < r^i. \tag{2}$$

It is easily seen that (2) has at most two solutions. More precisely, when r^i divides n there is exactly one solution, and when r^i does not divide n there are exactly two solutions. In either case, setting $y = n \mod r^i$ always gives a solution, and for the resulting x value we have $xr^i = n - (n \mod r^i) = \mathsf{closest}(n)$.

By the definition of i we have that $-r^{i+1} < n < r^{i+1}$. Combining this bound with the fact that $-r^i < -y < r^i$ we get

$$\begin{aligned} & -r^{i+1} - r^i < n - y < r^{i+1} + r^i \\ \Longrightarrow\ & -r - 1 < (n - y)/r^i < r + 1 \\ \Longrightarrow\ & -r \leq x \leq r. \end{aligned}$$

If $x = 0$, then (2) implies that $-r^i < n < r^i$, contrary to the definition of i. Thus, $x \neq 0$ and hence $xr^i \in \mathcal{C}$.

Now, to complete the proof, we simply observe that the representation $(a_{\ell-1} \ldots a_1 a_0)_r$ gives us a solution to (2) because

$$n = (a_{\ell-1} \ldots a_i)_r \cdot r^i + (a_{i-1} \ldots a_0)_r,$$

and $-r^i < (a_{i-1} \ldots a_0)_r < r^i$. Note that, by Lemma 9, the range of digits $a_{\ell-1} \ldots a_i$ is well defined because $\ell - 1 \geq i$. □

We can now present our final lemma.

Lemma 11. *Let w be the value returned by Algorithm W on input n and $r \geq 2$. Then $w \leq \mathsf{wt}^*(n)$.*

Proof. If $n = 0$, then $w = 0 = \mathsf{wt}^*(0)$, and so the Lemma holds in this case. Suppose $n \neq 0$. Let $c_1, c_2, \ldots, c_w$ be the sequence of values that the variable c assumes during the execution of Algorithm W. Let $(a_{\ell-1} \ldots a_1 a_0)_r$ be an arbitrary representation of n with digits in D_r and $a_{\ell-1} \neq 0$. Let $i = \lfloor \log_r |n| \rfloor$. Then

$$n = (a_{\ell-1} \ldots a_i)_r r^i + (a_{i-1} \ldots a_0)_r.$$

We want to replace $(a_{\ell-1}\dots a_i)_r r^i$ with c_1 in the equation above. If $(a_{\ell-1}\dots a_i)_r r^i$ is equal to c_1, then this is easily done, but it is not always true that $(a_{\ell-1}\dots a_i)_r r^i = c_1$. If $(a_{\ell-1}\dots a_i)_r r^i \neq c_1$, we can still carry out the replacement and maintain equality if we also replace a_{i-1} with a new digit $\widehat{a}_{i-1}$ defined as follows:

$$\widehat{a}_{i-1} := \begin{cases} a_{i-1} & \text{if } c_1 = (a_{\ell-1}\dots a_i)_r r^i, \\ a_{i-1} - r & \text{if } c_1 \neq (a_{\ell-1}\dots a_i)_r r^i \text{ and } a_{i-1} > 0, \\ a_{i-1} + r & \text{if } c_1 \neq (a_{\ell-1}\dots a_i)_r r^i \text{ and } a_{i-1} < 0. \end{cases} \tag{3}$$

Observe that $\widehat{a}_{i-1} \in D_r$. We will show that

$$n = c_1 + (\widehat{a}_{i-1}\dots a_0)_r. \tag{4}$$

Before we prove (4), we first make a note about the definition of $\widehat{a}_{i-1}$. It may seem as though the case $(a_{\ell-1}\dots a_i)_r r^i \neq c_1$ and $a_{i-1} = 0$ is missing from (3), but this is not so. Consider the difference $n - (a_{\ell-1}\dots a_i)_r r^i = (a_{i-1}\dots a_0)_r$. When $a_{i-1} = 0$ we have

$$\left|n - (a_{\ell-1}\dots a_i)_r r^i\right| = |(0a_{i-2}\dots a_0)_r| < r^{i-1} \leq \frac{r^i}{2}.$$

By Lemma 3, this bound implies that $(a_{\ell-1}\dots a_i)_r r^i = \mathsf{closest}(n) = c_1$. Thus when $(a_{\ell-1}\dots a_i)_r r^i \neq c_1$, a_{i-1} must be nonzero.

Now we return to establishing (4) for each case listed in (3). In the first case there is nothing to prove. Consider the second case. Suppose that $(a_{\ell-1}\dots a_i)_r r^i \neq c_1$ and $a_{i-1} > 0$. Then it is easily checked that

$$x = (a_{\ell-1}\dots a_i)_r,\ y = (a_{i-1}\dots a_0)_r$$

and

$$x = (a_{\ell-1}\dots a_i)_r + 1,\ y = (a_{i-1}\dots a_0)_r - r^i$$

are the two solutions to the constrained Diophantine equation $n = xr^i + y, -r^i < y < r^i$. Since $(a_{\ell-1}\dots a_i)_r r^i \neq c_1$, it must be that $((a_{\ell-1}\dots a_i)_r + 1)r^i = c_1$ (this follows from Lemma 10). Now,

$$\begin{aligned} n &= ((a_{\ell-1}\dots a_i)_r + 1)\, r^i + \left((a_{i-1}\dots a_0)_r - r^i\right) \\ &= c_1 + (\widehat{a}_{i-1}\dots a_0)_r, \end{aligned}$$

which is what we wanted to show. The third case, $(a_{\ell-1}\dots a_i)_r r^i \neq c_1$ and $a_{i-1} < 0$, is established in the same way (take $x = (a_{\ell-1}\dots a_i)_r - 1$, $y = (a_{i-1}\dots a_0)_r + r^i$). Thus, no matter what representation of n we begin with, (4) always holds.

We have

$$\begin{aligned} & n = c_1 + (\widehat{a}_{i-1}\dots a_0)_r \\ \implies\ & n - c_1 = (\widehat{a}_{i-1}\dots a_0)_r. \end{aligned}$$

Note that $\widehat{a}_{i-1}$ is nonzero if and only if a_{i-1} is nonzero. Thus, $(a_{i-1}\dots a_0)_r$ and $(\widehat{a}_{i-1}\dots a_0)_r$ *contain the same number of nonzero digits.* If $n - c_1 \neq 0$, then $(\widehat{a}_{i-1}\dots a_0)_r$ must contain another nonzero digit. In this case, we can do another replacement and incorporate the value c_2 by repeating the above process with $n - c_1$ in place of n. Thus, we see that we can carry out exactly w of these replacements. After we bring c_w into our equation, there can be no more nonzero digits in what remains of the representation $(\dots a_1 a_0)_r$; this is because $n - c_1 - \dots - c_w = 0$.

Each time we carry out a replacement, we eliminate *at least* one digit that was nonzero in $\alpha = a_{\ell-1}\dots a_0$ and we do not create any new ones. Thus, an upper bound on the number of replacements that can be applied is $\mathsf{wt}(\alpha)$. So, $w \leq \mathsf{wt}(\alpha)$.

This process works for any $\alpha \in {D_r}^*$ with $(\alpha)_r = n$. By taking α with $\mathsf{wt}(\alpha) = \mathsf{wt}^*(n)$ (i.e. by taking a minimal weight representation of n) we get $w \leq \mathsf{wt}^*(n)$, as desired. □

Example 12. We illustrate the transformation used in the proof of Lemma 11. The sequence of values that the variable c assumes during the execution of Algorithm W on input $n = 43$ and $r = 3$ is 54, -9, -2. We can transform any signed radix-3 representation of 43 into the sum $54 + (-9) + (-2)$ using the rules listed in (3). Consider the representation $(1121)_3 = 43$. Since $\lfloor \log_3 43 \rfloor = 3$ we write

$$43 = (1)_3 \cdot 3^3 + (121)_3.$$

Now, $(1)_3 \cdot 3^3 \neq 54$ and $a_2 = 1$ is positive, thus we have

$$43 = 54 + (\overline{2}21)_3.$$

$(\overline{2}21)_3$ is a representation of -11. Since $\lfloor \log_3 |-11| \rfloor = 2$ we write

$$43 = 54 + (\overline{2})_3 \cdot 3^2 + (21)_3.$$

-9 is closest to -11, and $(\overline{2})_3 \cdot 3^2 \neq -9$ with $a_1 = 2$ positive. Thus, we have

$$43 = 54 + (-9) + (\overline{1}1)_3.$$

Finally, $(\overline{1}1)_3$ is a representation of -2. Since $\lfloor \log_3 |-2| \rfloor = 0$ we write

$$43 = 54 + (-9) + (\overline{1}1)_3 \cdot 3^0.$$

-2 is closest to -2, and $(\overline{1}1)_3 \cdot 3^0 = -2$ so we have

$$43 = 54 + (-9) + (-2).$$

Notice that the final substitution eliminated two nonzero digits while the first two substitutions each eliminated one. If we had instead started with the representation $(2\overline{1}0\overline{2})_2 = 43$, then our substitutions would proceed as follows:

$$\begin{aligned} 43 &= (2\overline{1}0\overline{2})_2 \\ &= (2)_3 \cdot 3^3 + (\overline{1}0\overline{2})_3 \\ &= 54 + (\overline{1}0\overline{2})_3 \\ &= 54 + (\overline{1})_3 \cdot 3^2 + (0\overline{2})_3 \\ &= 54 + (-9) + (0\overline{2})_3 \\ &= 54 + (-9) + (-2). \end{aligned}$$

For this representation, each substitution eliminates one nonzero digit. ◊

After all those lemmas, we can now state a theorem.

Theorem 13. *Let w be the value returned by Algorithm W on input n and $r \geq 2$. Then $w = \mathsf{wt}^*(n)$.*

Proof. Apply Lemma 8 and Lemma 11. □

5 Constructing Representations

We now know that by using the function $\mathsf{closest}(n)$ we can build a minimal weight signed radix-r representation of an integer. This representation is implicitly constructed in Algorithm W. Here we present an *on-line* algorithm which outputs the digits of a minimal weight representation from left to right. As we will see, when the radix r is even, the representation built by the on-line algorithm matches that of Algorithm W; when r is odd, however, the two representations can differ.

The input to our algorithm is a representation $(b_{\ell-1} \dots b_1 b_0)_r$ where each $b_i \in \{0, 1, 2, \dots, r-1\}$. It is possible to describe a more general algorithm which takes arbitrary signed radix-r representations as input. However, restricting the input to representations with non-negative digits simplifies our discussion. As well, it allows us to compare our algorithm more directly with Joye and Yen's [3].

Our algorithm uses a "for" loop to construct its output representation $(a_\ell \dots a_1 a_0)_r$. At the beginning of each iteration of the "for" loop, the algorithm has the following information:

$$n_i = (\widehat{b_i} b_{i-1} \dots b_0)_r \text{ where } \widehat{b_i} = b_i + \Delta \text{ and } \Delta \in \{0, -r\}. \tag{5}$$

Initially, $i = \ell - 1$ and $\Delta = 0$. The value of a_i is determined by $\widehat{b_i}$ and b_{i-1}. Often, $\widehat{b_i}$ and b_{i-1} reveal the value of $\mathsf{closest}(n_i)$, but this is not always the case. Observe,

$$\begin{aligned} & n_i = (\widehat{b_i} b_{i-1} \dots b_0)_r \\ \Longrightarrow\ & \widehat{b_i} r^i \le n_i < (\widehat{b_i} + 1) r^i. \end{aligned} \tag{6}$$

Now consider the possible values of $\widehat{b_i}$. Since $\Delta \in \{0, -r\}$ and $b_i \in \{0, 1, 2, \dots, r-1\}$, we have

$$\widehat{b_i} \in \{-r, -(r-1), \dots, -1, 0, 1, \dots, r-1\}.$$

When $\widehat{b_i} \notin \{-1, 0\}$ then from (6) we see that $\widehat{b_i} r^i$ and $(\widehat{b_i} + 1) r^i$ are two consecutive, nonzero elements of $\mathcal{C}$ that bound n_i. Thus, $\mathsf{closest}(n_i)$ equals either $\widehat{b_i} r^i$ or $(\widehat{b_i} + 1) r^i$. When r is even, the value of b_{i-1} always reveals which of the two possibilities is correct: if $b_{i-1} \ge r/2$ then $\mathsf{closest}(n_i) = (\widehat{b_i} + 1) r^i$, and $\mathsf{closest}(n_i) = \widehat{b_i} r^i$ otherwise. When r is odd, this is not necessarily true.

Example 14. For the radix $r = 3$, suppose we have $n_i = (21 b_{i-2} b_{i-3} \dots)_3$. The most significant digit of this representation, $b_i = 2$, tells us that $2r^i \le n_i < 3r^i$. Unfortunately, the next digit, $b_{i-1} = 1$, does not reveal which of these two values is closer. Observe $(210 b_{i-3} \dots)_3$ is closer to $2r^i$, but $(212 b_{i-3} \dots)_3$ is closer to $3r^i$; similarly for $(2110 \dots)_3$ and $(2112 \dots)_3$. By putting a long run of 1's to the right of b_i, we see that reading any finite number of digits immediately to the right of b_i will not suffice to determine the closest choice; thus, no on-line algorithm is able to make this determination. ◊

When r is odd, we can deduce the following: if $b_{i-1} > \lfloor r/2 \rfloor$, then $\mathsf{closest}(n_i) = (\widehat{b_i} + 1) r^i$; if $b_{i-1} < \lfloor r/2 \rfloor$, then $\mathsf{closest}(n_i) = \widehat{b_i} r^i$; but, if $b_{i-1} = \lfloor r/2 \rfloor$, the value of $\mathsf{closest}(n_i)$ cannot be decided. Fortunately, when $b_{i-1} = \lfloor r/2 \rfloor$, it does not matter whether we choose to use $(\widehat{b_i} + 1) r^i$ or $\widehat{b_i} r^i$ in our representation of n_i; both choices yield minimal weight representations.

Lemma 15. *Let $r \ge 2$ be odd. If $n_i = (\widehat{b_i} b_{i-1} \dots b_0)_r$ with $\widehat{b_i} \notin \{-1, 0\}$ and $b_{i-1} = \lfloor r/2 \rfloor$, then*

$$\mathsf{wt}^*(n_i - \mathsf{closest}(n_i)) = \mathsf{wt}^*\left(n_i - \widehat{b_i} r^i\right).$$

Proof. If $\mathsf{closest}(n_i) = \widehat{b_i} r^i$, then there is nothing to prove. Assume that $\mathsf{closest}(n_i) = (\widehat{b_i} + 1) r^i$. From the closest choice representation of n_i, we have

$$n_i = (\widehat{b_i} + 1) r^i + c_2 + c_3 + \dots + c_w.$$

Because $n_i < (\widehat{b_i}+1)r^i$ it must be that $c_2 < 0$. We claim that $c_2 = dr^{i-1}$. To see this, observe

$$\begin{aligned} &\widehat{b_i}r^i + \lfloor r/2 \rfloor\, r^{i-1} \le n_i < \widehat{b_i}r^i + (\lfloor r/2 \rfloor + 1)r^{i-1} \\ \Longrightarrow\ &-r^i + \lfloor r/2 \rfloor\, r^{i-1} \le n_i - (\widehat{b_i}+1)r^i < -r^i + (\lfloor r/2 \rfloor + 1)r^{i-1} \\ \Longrightarrow\ &(\lfloor r/2 \rfloor - r)r^{i-1} \le n_i - (\widehat{b_i}+1)r^i < (\lfloor r/2 \rfloor - r + 1)r^{i-1}. \end{aligned}$$

By writing $r = 2x+1$ where $x \ge 1$, it can be shown that $\lfloor r/2 \rfloor - r \le -2$. Thus, we have two consecutive nonzero elements of $\mathcal{C}$ that bound $n_i - (\widehat{b_i}+1)r^i$; so $\mathsf{closest}\big(n_i - (\widehat{b_i}+1)r^i\big) = dr^{i-1}$ where d is a negative digit in $\{\lfloor r/2 \rfloor - r, \lfloor r/2 \rfloor - r + 1\}$. Now,

$$\begin{aligned} n_i &= (\widehat{b_i}+1)r^i + dr^{i-1} + c_3 + \cdots + c_w \\ &= \widehat{b_i}r^i + (r+d)r^{i-1} + c_3 + \cdots + c_w. \end{aligned}$$

Note that $r+d$ is a valid (positive) digit. Because the first representation has minimal weight, so too does the second. Hence, the result follows. □

In light of Lemma 15, when we cannot decide on the correct value of $\mathsf{closest}(n_i)$ because $b_{i-1} = \lfloor r/2 \rfloor$, we can safely take $\widehat{b_i}r^i$ without sacrificing minimality. However, a consequence of this strategy is that when r is odd, the minimal weight representation built by our on-line algorithm can be different from the closest choice representation.

Here is how our algorithm works. When $\widehat{b_i} \notin \{-1, 0\}$, it sets a_i to equal either $\widehat{b_i}$ or $\widehat{b_i}+1$, and it does so in the following way. At first, it takes $a_i = \widehat{b_i}$, but then, if $b_{i-1} \ge r/2$, it takes $a_i = \widehat{b_i}+1$. After a_i is determined, Δ is updated; if $a_i = \widehat{b_i}+1$ then $\Delta = -r$, otherwise $\Delta = 0$. It is possible that $\widehat{b_i}+1 = r$, in which case we cannot take $a_i = \widehat{b_i}+1$ because r is not a valid digit for our output representation. However, this possibility is easily detected and can be dealt with by setting $a_{i+1} = 1$ and $a_i = 0$ (we will see that the previous value of a_{i+1} is necessarily zero so overwriting it does not cause a problem). It is also possible that $\widehat{b_i} = -r$. This is dealt with similarly; we set $a_{i+1} = -1$ and $a_i = 0$ (again, this causes no problem). In the subsequent loop iteration, we have $n_{i-1} = (\widehat{b_{i-1}} b_{i-2} \ldots b_0)_r = n_i - a_i r^i$.

Two cases which we have not yet addressed are when $\widehat{b_i} \in \{-1, 0\}$. For these values, (6) becomes

$$0 \le n_i < r^i, \quad \text{and} \quad -r^i \le n_i < 0.$$

With respect to determining $\mathsf{closest}(n_i)$, these bounds are not of much use. Regardless, the algorithm must set a_i to some value. In these cases, we set $a_i = 0$. More precisely, when $\widehat{b_i} = 0$, we set $a_i = 0$ and update Δ to equal 0; and when $\widehat{b_i} = -1$, we set $a_i = 0$ and update Δ to equal $-r$. In either situation, it can be shown that updating the value of Δ is unnecessary. For instance, since $0 \le b_i \le r-1$ and $\Delta \in \{0, -r\}$, if $\widehat{b_i} = b_i + \Delta = 0$, then it must be that $b_i = 0$ and $\Delta = 0$; since Δ is already 0, we do not need to update it.

This strategy for dealing with $\widehat{b_i} \in \{-1, 0\}$ works well except when it happens that $\widehat{b_0} = -1$. In this case, we would set $a_0 = 0$ and pass $\Delta = -r$ into the subsequent loop iteration. However, there *is* no subsequent loop iteration. We deal with this problem by checking the value of Δ after the "for" loop completes. If $\Delta = -r$, then we set $a_0 = -1$.

We still need to justify that setting $a_{i+1} = 1$ and $a_i = 0$ instead of $a_i = r$ does not overwrite a nonzero digit. This could only happen if $\widehat{b_i}+1 = r$. From (5), we see that this implies $b_i = r-1$ and $\Delta = 0$. The value of Δ was determined in the previous loop iteration when the initial value of a_{i+1} was set. If a_{i+1} was nonzero, then the fact that $b_i = r-1 \ge r/2$ would imply $\Delta = -r$, contrary to the fact that $\Delta = 0$. So, the initial value of a_{i+1} must have been zero. The case where we set $a_{i+1} = -1$ and $a_i = 0$ instead of $a_i = -r$ can be justified similarly. This happens only if $\widehat{b_i} = -r$. From (5), we can deduce that $b_i = 0$ and $\Delta = -r$. The value of Δ was determined in the previous loop iteration. If the initial value of a_{i+1} was nonzero then

we would have $\Delta = 0$ because $b_i = 0 < r/2$, contrary to the fact that $\Delta = -r$. So, again, the initial value of a_{i+1} must have been zero.

This method of constructing minimal weight signed radix-r representations is implemented in Algorithm R.

Algorithm R

INPUT: $(b_{\ell-1} \dots b_1 b_0)_r$ where each $b_i \in \{0, 1, \dots, r-1\}$.
OUTPUT: $(a_\ell \dots a_1 a_0)_r$

$b_{-1} \leftarrow 0,\ a_\ell \leftarrow 0,\ \Delta \leftarrow 0$
for $i = \ell - 1 \dots 0$
do
- $\widehat{b} \leftarrow b_i + \Delta$
- **if** $\widehat{b} \in \{-1, 0\}$ **then** $\widehat{b} \leftarrow 0$
- **else**
  - **if** $b_{i-1} \geq r/2$
    - **then** $\Delta \leftarrow -r,\ \widehat{b} \leftarrow \widehat{b} + 1$
    - **else** $\Delta \leftarrow 0$
- **if** $\widehat{b} = r$ **then** $a_{i+1} \leftarrow 1,\ a_i \leftarrow 0$
  - **else if** $\widehat{b} = -r$ **then** $a_{i+1} \leftarrow -1,\ a_i \leftarrow 0$
  - **else** $a_i \leftarrow \widehat{b}$

if $\Delta = -r$ **then** $a_0 \leftarrow -1$
return $(a_\ell \dots a_1 a_0)_r$

Algorithm R can be adapted so that it outputs an integer's arithmetic weight. This is done in Appendix A. Computing the arithmetic weight in this way does not require the computation of logarithms or divisions; these operations are used in Algorithm W.

Algorithm R can be combined with the left-to-right radix-r method of computing nP in Figure 1. However, the fact that digit a_{i+1} can be overwritten after the value of a_i is determined requires some consideration. One easy way to deal with this is to simply allow digits to take the values $\pm r$. This can be accomplished by replacing the "if-else-else" statement at the bottom of the "for" loop in Algorithm R with the statement $a_i \leftarrow \widehat{b}$. Note that this change requires the point rP to be precomputed in addition to the points $1P, 2P, \dots, (r-1)P$.

An initial question that we might ask about Algorithm R is whether or not it is equivalent to Joye and Yen's (see Figure 1, page 380, of [3]). It is not, and this fact can be easily demonstrated.

Example 16. Let $r = 3$. The $\{0, 1, 2\}$-radix-3 representation of 41 is $(1112)_3$. Here are three minimal weight $\{0, \pm 1, \pm 2\}$-radix-3 representations constructed by three algorithms:

Algorithm R	$(11\overline{2}\overline{1})_3$
Joye & Yen	$(1\overline{2}\overline{11})_3$
GNAF	$(\overline{2111})_3$.

The GNAF of 41 was constructed using Theorem 3 from [1]. Notice that each output is different from the others. Thus, we can conclude that the algorithms which created these outputs are not equivalent. Note also that the closest choice representation of 41 is $54 + (-9) + (-3) + (-1) = (\overline{2111})_3$ which is different from the output of Algorithm R. ◊

Example 17. The output of Algorithm R on input $(30121230311212)_4 = 208063846$ was listed in Example 6 (note that r is even). Joye and Yen give the output of their algorithm in [3]:

Algorithm R	$(30\overline{2}\overline{2}2\overline{1}0\overline{3}0\overline{3}\overline{2}\overline{2}\overline{2}2)_4$
Joye & Yen	$(30122\overline{1}0\overline{3}0\overline{3}1212)_4$
GNAF	$(30\overline{2121}0\overline{3}0\overline{3}1212)_4$.

Again, these outputs demonstrate that the algorithms are not equivalent. ◊

More generally, it can be shown that, for any $r > 2$, the smallest positive integer for which the the two algorithms produce different representations is $n = r + \lfloor r/2 \rfloor$.

Algorithm R does not always produce a different representation than Joye and Yen's algorithm. In the case when $r = 2$, it appears that the outputs of the two algorithms coincide. If we take $r = 4$ and consider the representations constructed by the two algorithms for the integers $1 \ldots 63$, then all but eleven of them are equal; different representations are output for the integers 6, 13, 22, 24, 25, 38, 39, 52, 53, 54 and 55.

One difference between the outputs produced by Algorithm R and those produced by Joye and Yen's algorithm is that Algorithm R's outputs sometimes have shorter length. For $r = 4$, when we compare the outputs of the two algorithms for the integers $1 \ldots 2047$, we find that 85 representations computed by Algorithm R are shorter (by one digit in each case) and 1962 representations have the same length. This difference indicates that the algorithms for computing nP in Figure 1 perform slightly better with the representations constructed by Algorithm R.

Algorithm R also differs from Joye and Yen's in that it only examines *two* input digits, b_i, b_{i-1}, before it sets the value of a_i; Joye and Yen's algorithm examines *three* digits, b_i, b_{i-1}, b_{i-2}.[1] As well, Algorithm R requires ℓ loop iterations while Joye and Yen's algorithm requires $\ell + 1$; here ℓ is the length of the input representation. However, deciding which of the two algorithms is more efficient is a subjective task; this is dependent on what resources (e.g. memory) are available to an implementor. In software, there seems to be little reason to choose one over the other. In hardware, the fact that Algorithm R requires fewer temporary variables may be of some benefit. A non-academic reason to choose Algorithm R is that it is not encumbered by patents. For $r = 2$, Joye and Yen's technique has been patented in France (patent no. 2811168) and in the USA (patent no. 6903663); for $r \geq 2$, patents are pending in France and the USA.

6 Concluding Remarks

In this paper, we have presented a new algorithm for computing the arithmetic weight of an integer. This algorithm leads to a new family of minimal weight signed radix-r representations which can be constructed using a left-to-right on-line algorithm. The idea behind our algorithm is simply that of choosing closest elements. This is a very general approach and it may be useful in constructing other families of minimal weight integer representations. Use of our algorithm is not encumbered by patents.

Acknowledgements

The author is indebted to Harry Reimann for pointing out an error in Algorithm R in a previous version of this paper.

References

[1] W. CLARK AND J. LIANG. On arithmetic weight for a general radix representation of integers. *IEEE Transactions on Information Theory* **19** (1973), 823–826.

[2] FIPS 186-2. *Digital Signature Standard (DSS)*. Federal Information Processing Standards Publication 186-2, U.S. Department Of Commerce / National Institute of Standards and Technology, 2000. Available from `http://www.csrc.nist.gov/publications/fips/`

[1] More precisely, in each loop iteration, Joye and Yen's algorithm executes one of three cases. In the second case, three input digits must be examined to determine an output digit; in the first and third cases, only two input digits need to be examined.

[3] M. JOYE AND S. YEN. New minimal modified radix-r representation with applications to smart cards. Public Key Cryptography 2002, *Lecture Notes in Computer Science* **2274** (2002), 375–383.

[4] J. VAN LINT. *Introduction to Coding Theory*, 3rd edition, Springer, 1999.

[5] J. MUIR AND D. STINSON. New minimal weight representations for left-to-right window methods. Cryptographers' Track at the RSA Conference 2005, *Lecture Notes in Computer Science* **3376** (2005), 366–383.

[6] T. RAO AND O. GARCIA Cyclic and multiresidue codes for arithmetic operations. *IEEE Transactions on Information Theory* **17** (1971), 85–91.

[7] G. REITWIESNER. Binary arithmetic. In *Advances in Computers, Vol. 1*, Academic Press, 1960, pp. 231–308.

A An Alternate Implementation of Algorithm W

When Algorithm W executes, $\mathsf{closest}(n)$ is evaluated a number of times. Evaluating $\mathsf{closest}(n)$ requires computation of a logarithm and a division; these are necessary to determine $\lfloor \log_r |n| \rfloor$ and $n \bmod r^{\lfloor \log_r |n| \rfloor}$. If we already have the radix-r representation of the input n, then these computations can be avoided.

Below is an alternate implementation of our algorithm for computing an integer's arithmetic weight. Algorithm W' takes the radix-r representation of n as input and computes $\mathsf{wt}^*(n)$ without using logarithms or divisions.

Algorithm W'

INPUT: $(b_{\ell-1} \ldots b_1 b_0)_r$ where each $b_i \in \{0, 1, \ldots, r-1\}$.
OUTPUT: the arithmetic weight of $n = (b_{\ell-1} \ldots b_1 b_0)_r$

$b_{-1} \leftarrow 0,\ w \leftarrow 0,\ \Delta \leftarrow 0$
for $i = \ell - 1 \ldots 0$
do
- $\widehat{b} \leftarrow b_i + \Delta$
- **if** $\widehat{b} \in \{-1, 0\}$ **then** *do nothing*
- **else**
  - $w \leftarrow w + 1$
  - **if** $b_{i-1} \geq r/2$
    - **then** $\Delta \leftarrow -r$
    - **else** $\Delta \leftarrow 0$

if $\Delta = -r$ **then** $w \leftarrow w + 1$
return w