SIAM J. DISCRETE MATH.
Vol. 21, No. 1, pp. 16–25

A SIMPLE GRAY CODE TO LIST ALL MINIMAL SIGNED BINARY REPRESENTATIONS[*]

J. SAWADA[†]

Abstract. A signed binary representation (SBR) of an integer N is a string $a_b \cdots a_2 a_1 a_0$ over the alphabet $\{-1, 0, 1\}$ such that $N = \sum_{i=0}^{b} a_i 2^i$. An SBR of an integer N is said to be *minimal* if the number of nonzero digits is minimum. In this paper, we describe a simple 3-close Gray code for listing all minimal SBRs of an integer N. The algorithm is implemented to run in constant amortized time. In addition, we identify the values for N that have the maximum number of minimal SBRs given the length of the binary representation of N.

Key words. Signed binary representations, Gray code

AMS subject classification. 05A

DOI. 10.1137/050641405

1. Introduction.

A *signed binary representation* (SBR) of an integer N is a string $a_b \cdots a_2 a_1 a_0$ over the alphabet $\{-1, 0, 1\}$ such that $N = \sum_{i=0}^{b} a_i 2^i$. An example of an SBR for $N = 51$ is $10\bar{1}010\bar{1}$ (where for convenience we use $\bar{1}$ for -1), which corresponds to $2^6 - 2^4 + 2^2 - 2^0$.

An SBR of an integer N is said to be *minimal* if the number of nonzero digits is minimum. A minimal SBR for an integer N is not necessarily unique; in fact, we will show that there may be an exponential number of such strings with respect to the length of the binary representation of N. As an example, there are 5 minimal SBRs for $N = 51$ each requiring 4 nonzero bits:

$$0110011, 011010\bar{1}, 100\bar{1}\bar{1}0\bar{1}, 10\bar{1}0011, 10\bar{1}010\bar{1}.$$

Booth [2] first applied the notion of SBRs to a signed binary multiplication technique. A decade later, Reitwiesner [10] gave the first linear time algorithm to find a minimal SBR for a given integer N. Since then, several other researchers have provided similar linear time algorithms, including the following one-line algorithm (based on work by Güntzer and Paul [4]) given by Prodinger [9]: "writing $3N/2$ in binary and subtracting (bitwise) the binary representation of $N/2$." For a more thorough history of SBRs and how they apply to fast exponentiation and cryptography, consult [6, 8, 12, 14].

As there are potentially an exponential number of minimal SBRs for an integer N (with respect to the length of the binary representation of N), it is natural to ask how efficiently we can produce an exhaustive listing of these objects. Ideally, a listing algorithm will run in time proportional to the number of objects (strings) generated. Such algorithms are said to be CAT for constant amortized time. Also, it is often useful for a listing of objects to have the *Gray code property*: successive objects in the listing differ by a constant amount.

In [6], Ganesan and Manku show how minimal SBRs can be used to find optimal routes in a network derived from Chord [5], a peer-to-peer network topology. They

*Received by the editors September 28, 2005; accepted for publication (in revised form) July 23, 2006; published electronically January 8, 2007.
http://www.siam.org/journals/sidma/21-1/64140.html

†Computing and Information Science, University of Guelph, Guelph N1G 2W1, ON, Canada (sawada@cis.uoguelph.ca).

also present the only previously known algorithm for exhaustively listing all minimal SBRs. Unfortunately, no analysis of the algorithm was provided and the resulting listing does not have the Gray code property. To remedy this situation, we modify their algorithm into one that is a 3-close Gray code listing (successive strings differ in 3 consecutive positions) and provide steps to make the algorithm CAT. This is discussed in section 2. Then in section 3, as a secondary result, we identify precisely the values for N that have the maximum number of minimal SBRs given the length of the binary representation for N. In section 4 we identify two interesting sequences with respect to SBRs and conclude with final remarks in section 5.

For the remainder of this paper we will let SBR(N) denote the set of all minimal signed binary representations of an integer N. It also assumed that N is represented in binary as $a_b \cdots a_2 a_1 a_0$, and as mentioned earlier, we will use $\bar{1}$ to represent -1 for convenience.

2. Listing minimal SBRs.

The following is a recursive description of Ganesan and Manku's algorithm [6] where $\mathbf{B}(N)$ denotes a listing of all the strings in SBR(N). The notation $\mathbf{B}(N) \cdot 1$ denotes the listing $\mathbf{B}(N)$ with an additional 1 appended to each string. The notation $\mathbf{B}(N), \mathbf{B}(M)$ indicates the list of strings $\mathbf{B}(N)$ followed by the list of strings $\mathbf{B}(M)$:

$$\mathbf{B}(N) = \begin{cases} 0 & \text{if } N = 0, \\ \mathbf{B}(\frac{N}{2}) \cdot 0 & \text{if } \mathbf{suffix}(N, 0) \text{ and } N > 0, \\ \mathbf{B}(\frac{N-1}{2}) \cdot 1 & \text{if } \mathbf{suffix}(N, 0(01)^*01), \\ \mathbf{B}(\frac{N+1}{2}) \cdot \bar{1} & \text{if } \mathbf{suffix}(N, 1(10)^*11)\ , \\ \mathbf{B}(\frac{N+1}{2}) \cdot \bar{1},\ \mathbf{B}(\frac{N-1}{2}) \cdot 1 & \text{if } \mathbf{suffix}(N, 11(01)^*01) \text{ or} \\ & \text{if } \mathbf{suffix}(N, 00(10)^*11). \end{cases}$$

The predicate **suffix**(N, *expr*) returns true if a suffix of N, represented in binary (and padded with 0's on the left), matches the regular expression *expr*. An example computation tree for $\mathbf{B}(51)$ is given in Figure 1. The nodes in the tree represent the input strings, and the labels on the edges represent the character to be prepended to the output string as specified by $\mathbf{B}(N)$. Thus, each minimal SBR can be found by tracing a path from a leaf back to the root while recording the labels on the edges. Observe that since 0011 is a suffix of 110011, it satisfies the last case in the recursive description. Thus, the root node in Figure 1 has 2 children producing strings that end with $\bar{1}$ and 1, respectively.

2.1. A Gray code.

In general the listing $\mathbf{B}(N)$ is not a Gray code since successive strings in the listing may differ by up to a linear amount $\Omega(b)$. However, by studying the listings for a variety of input values and focusing on the parities of the repeated terms in the regular expressions, we discover a 3-close Gray code description for SBR(N). This new listing is obtained by reversing the order of particular subtrees within the computation tree of $\mathbf{B}(N)$. The result is a listing that will produce the same strings but in a different order. The overline in the description of this new listing $\mathbf{L}(N)$ indicates that the listing of strings is reversed:

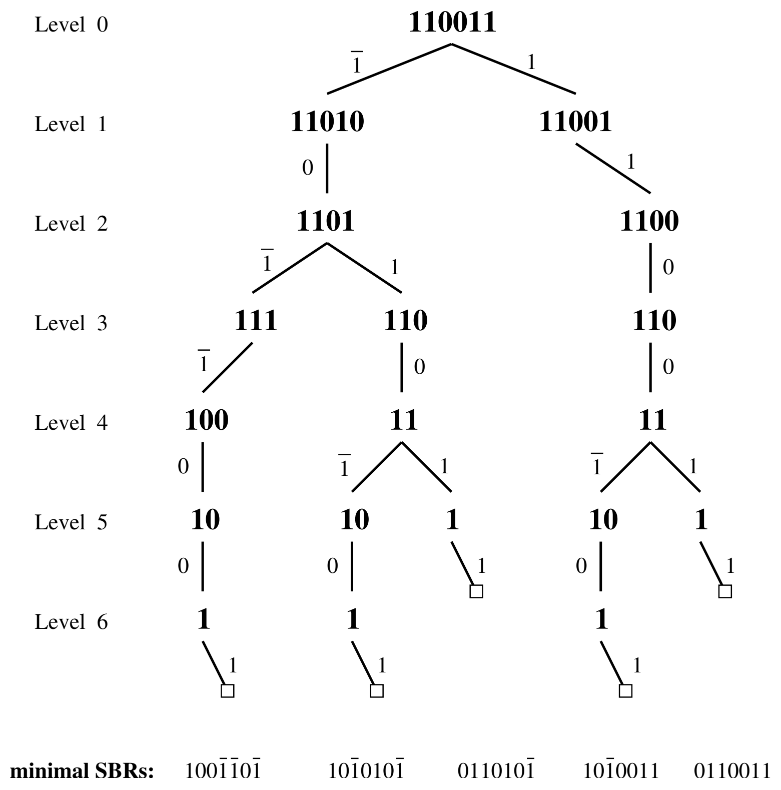

FIG. 1. *Computation tree for* $\mathbf{B}((110011)_2) = \mathbf{B}(51)$.

$$\mathbf{L}(N) = \begin{cases} 0 & \text{if } N = 0, \\ \mathbf{L}(\frac{N}{2}) \cdot 0 & \text{if } \mathbf{suffix}(N, 0) \text{ and } N > 0, \\ \mathbf{L}(\frac{N-1}{2}) \cdot 1 & \text{if } \mathbf{suffix}(N, 0(01)^*01), \\ \mathbf{L}(\frac{N+1}{2}) \cdot \bar{1} & \text{if } \mathbf{suffix}(N, 1(10)^*11), \\ \overline{\mathbf{L}(\frac{N+1}{2})} \cdot \bar{1},\ \mathbf{L}(\frac{N-1}{2}) \cdot 1 & \text{if } \mathbf{suffix}(N, 11(01)^t01) \text{ and } t \text{ even} \qquad (1), \\ \mathbf{L}(\frac{N+1}{2}) \cdot \bar{1},\ \overline{\mathbf{L}(\frac{N-1}{2})} \cdot 1 & \text{if } \mathbf{suffix}(N, 00(10)^t11) \text{ and } t \text{ even} \qquad (2), \\ \mathbf{L}(\frac{N+1}{2}) \cdot \bar{1},\ \mathbf{L}(\frac{N-1}{2}) \cdot 1 & \text{if } \mathbf{suffix}(N, 11(01)^t01) \text{ and } t \text{ odd or} \qquad (3), \\ & \text{if } \mathbf{suffix}(N, 00(10)^t11) \text{ and } t \text{ odd} \qquad (4). \end{cases}$$

THEOREM 1. *The listing* $\mathbf{L}(N)$ *of all strings in SBR(N) where* $N > 0$ *is a* 3*-close Gray code.*

Proof. Assume that N is represented in binary. Let $\mathbf{first}(N)$ and $\mathbf{last}(N)$ denote the first and last strings in the listing of $\mathbf{L}(N)$. To prove that the listing $\mathbf{L}(N)$ is a 3-close Gray code we show that the interface strings for Cases (1), (2), (3), and (4) differ in *exactly the last three positions*. Applying induction completes the proof.

Case (1). N is of the form $x11(01)^t01$, where x is some binary string and t is even. Here we must compare the last string in $\overline{\mathbf{L}(\frac{N+1}{2})} \cdot \bar{1} = \mathbf{first}(x11(01)^t1) \cdot \bar{1}$ and the first string in $\mathbf{L}(\frac{N-1}{2}) \cdot 1 = \mathbf{first}(x11(01)^t0) \cdot 1$. First consider $t > 0$:

$$\begin{aligned}\mathbf{first}(x11(01)^t1) \cdot \bar{1} &= \mathbf{first}(x1(10)^t11) \cdot \bar{1}\\ &= \mathbf{first}(x1(10)^{t-1}110) \cdot \bar{1}\bar{1}\\ &= \mathbf{first}(x1(10)^{t-1}11) \cdot 0\bar{1}\bar{1},\\ \mathbf{first}(x11(01)^t0) \cdot 1 &= \mathbf{first}(x11(01)^t) \cdot 01\\ &= \mathbf{first}(x11(01)^{t-1}01) \cdot 01\\ &= \mathbf{first}(x11(01)^{t-1}1) \cdot \bar{1}01\\ &= \mathbf{first}(x1(10)^{t-1}11) \cdot \bar{1}01.\end{aligned}$$

If $t = 0$, let $x = y01^r$:

$$\begin{aligned}\mathbf{first}(y01^r111) \cdot \bar{1} &= \mathbf{first}(y10^r00) \cdot \bar{1}\bar{1}\\ &= \mathbf{first}(y10^r0) \cdot 0\bar{1}\bar{1},\\ \mathbf{first}(y01^r110) \cdot 1 &= \mathbf{first}(y01^r11) \cdot 01\\ &= \mathbf{first}(y10^r0) \cdot \bar{1}01.\end{aligned}$$

Case (2). N is of the form $x00(10)^t11$, where x is some binary string and t is even. Again we consider two subcases depending on the value for t. If $t > 0$, then we must compare the last string in $\mathbf{L}(\frac{N+1}{2}) \cdot \bar{1} = \mathbf{last}(x00(10)^{t-1}110) \cdot \bar{1}$ and the first string in $\overline{\mathbf{L}(\frac{N-1}{2})} \cdot 1 = \mathbf{last}(x00(10)^t1) \cdot 1$:

$$\begin{aligned}\mathbf{last}(x00(10)^{t-1}110) \cdot \bar{1} &= \mathbf{last}(x00(10)^{t-1}11) \cdot 0\bar{1}\\ &= \mathbf{last}(x00(10)^{t-1}1) \cdot 10\bar{1}\\ &= \mathbf{last}(x0(01)^t) \cdot 10\bar{1},\\ \mathbf{last}(x00(10)^t1) \cdot 1 &= \mathbf{last}(x0(01)^t01) \cdot 1\\ &= \mathbf{last}(x0(01)^t0) \cdot 11\\ &= \mathbf{last}(x0(01)^t) \cdot 011.\end{aligned}$$

If $t = 0$, then the two interface strings are $\mathbf{last}(x010)\cdot\bar{1}$ and $\mathbf{last}(x001)\cdot 1$, respectively:

$$\begin{aligned}\mathbf{last}(x010) \cdot \bar{1} &= \mathbf{last}(x01) \cdot 0\bar{1}\\ &= \mathbf{last}(x0) \cdot 10\bar{1},\\ \mathbf{last}(x001) \cdot 1 &= \mathbf{last}(x001) \cdot 1\\ &= \mathbf{last}(x0) \cdot 011.\end{aligned}$$

Case (3). N is of the form $x11(01)^t01$, where x is some binary string and t is odd. Here we must compare the last string in $\mathbf{L}(\frac{N+1}{2}) \cdot \bar{1} = \mathbf{last}(x11(01)^t1) \cdot \bar{1}$ and the first string in $\mathbf{L}(\frac{N-1}{2}) \cdot 1 = \mathbf{first}(x11(01)^t0) \cdot 1$:

$$\begin{aligned}\mathbf{last}(x11(01)^t1) \cdot \bar{1} &= \mathbf{last}(x1(10)^t11) \cdot \bar{1}\\ &= \mathbf{last}(x1(10)^{t-1}110) \cdot \bar{1}\bar{1}\\ &= \mathbf{last}(x1(10)^{t-1}11) \cdot 0\bar{1}\bar{1},\\ \mathbf{first}(x11(01)^t0) \cdot 1 &= \mathbf{first}(x11(01)^{t-1}01) \cdot 01\\ &= \mathbf{last}(x11(01)^{t-1}1) \cdot \bar{1}01\\ &= \mathbf{last}(x1(10)^{t-1}11) \cdot \bar{1}01.\end{aligned}$$

Case (4). N is of the form $x00(10)^t11$, where x is some binary string and t is odd. Here we must compare the last string in $\mathbf{L}(\frac{N+1}{2}) \cdot \bar{1} = \mathbf{last}(x00(10)^{t-1}110) \cdot \bar{1}$ and the first string in $\mathbf{L}(\frac{N-1}{2}) \cdot 1 = \mathbf{first}(x00(10)^t1) \cdot 1$:

$$\begin{aligned}
\mathbf{last}(x00(10)^{t-1}110) \cdot \bar{1} &= \mathbf{last}(x00(10)^{t-1}11) \cdot 0\bar{1} \\
&= \mathbf{first}(x00(10)^{t-1}1) \cdot 10\bar{1} \\
&= \mathbf{first}(x0(01)^t) \cdot 10\bar{1}, \\
\mathbf{first}(x00(10)^t1) \cdot 1 &= \mathbf{first}(x0(01)^t01) \cdot 1 \\
&= \mathbf{first}(x0(01)^t0) \cdot 11 \\
&= \mathbf{first}(x0(01)^t) \cdot 011.
\end{aligned}$$

In all cases we have shown that the interface strings differ in exactly the last 3 positions. By applying induction, this proves that $\mathbf{L}(N)$ is a 3-close Gray code for the strings in SBR(N). □

As an example, Figure 2 displays the computation tree for $\mathbf{L}(110011)$. As before, the nodes represent the input strings, but now if a subtree is to be reversed, this information is additionally passed down via the edges and represented by R. Naturally, a reversal of a reversed subtree produces the regular ordering (see the 11 node furthest to the right in Figure 2).

The following is the final listing generated for $\mathbf{L}(110011)$:

$$\begin{array}{cccccccccc}
0 & 1 & 0 & 0 & \bar{1} & \bar{1} & 0 & \bar{1} & & \\
0 & 1 & 0 & \bar{1} & 0 & 1 & 0 & \bar{1} & & (2) \\
0 & 0 & 1 & 1 & 0 & 1 & 0 & \bar{1} & & (4) \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & & (0) \\
0 & 1 & 0 & \bar{1} & 0 & 0 & 1 & 1 & & (4)
\end{array}$$

Observe that each successive string differs in exactly 3 *consecutive* positions by either the transformation $011 \leftrightarrow 10\bar{1}$ or $0\bar{1}\bar{1} \leftrightarrow \bar{1}01$. Also observe that the rightmost of these positions (indicated in parentheses) corresponds to the levels of the degree 2 nodes in the computation tree when visited in order. These properties can also be inferred from Theorem 1 and its proof. Therefore given the in-order sequence of levels of the degree 2 nodes along with the first output string in the listing, we can generate the Gray code listing $\mathbf{L}(N)$ in constant amortized time. In the next subsection, we describe how we can efficiently generate this sequence.

It is also interesting to note from this example that a 2-close listing is impossible in general. This is because the first string is the only one that contains $0\bar{1}\bar{1}$ in positions 4, 5, and 6.

Also of interest is the underlying graph with vertices corresponding to the minimal SBRs (for a given integer N) and edges between two vertices if and only if their corresponding SBRs differ in exactly 3 adjacent positions. Clark and Liang [3] showed that this graph is connected. The result in this paper shows that this graph contains a Hamilton path. In general there is no Hamilton cycle since the underlying graph from our example has a vertex with degree 1: $0100\bar{1}\bar{1}0\bar{1}$.

2.2. Efficiency considerations. If we apply the algorithm for $\mathbf{L}(N)$ or $\mathbf{B}(N)$ directly, we may require a linear amount of work to process each node: computing the suffix and modifying the input string for the next recursive call. This amount of computation is not desirable; however, because there are repeated subtrees, we can precompute the parent child relationships for each node. In fact, given that

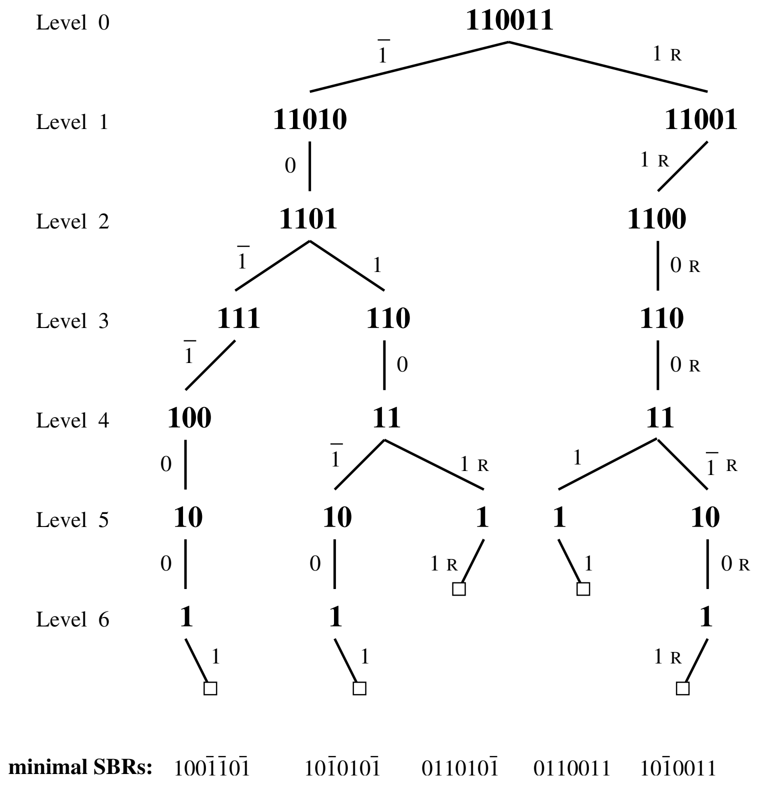
Level 0
Level 1
Level 2
Level 3
Level 4
Level 5
Level 6
110011
11010
11001
1101
1100
111
110
110
100
11
11
minimal SBRs: $100\bar{1}\bar{1}0\bar{1}$ $10\bar{1}010\bar{1}$ $011010\bar{1}$ 0110011 $10\bar{1}0011$

FIG. 2. *Computation tree for* $\mathbf{L}(110011)$.

$N = a_b \cdots a_2a_1a_0$ and that α_i denotes the prefix $a_b \cdots a_i$ and $\beta_i = \alpha_i+1$, the following lemma shows that the number of different possible nodes in the computation tree for $\mathbf{L}(N)$ or $\mathbf{B}(N)$ is at most $2(b+1)$.

LEMMA 2. *At level i in the computation tree of $\mathbf{L}(N)$ or $\mathbf{B}(N)$ the input string is either α_i or β_i.*

Proof. The proof is by induction on i. When $i = 0$, we are at the root of the computation tree and the input string is α_0. For the inductive hypothesis, suppose that the input string for a node at level $i \geq 0$ is either α_i or β_i. By using the rules of the listing $\mathbf{L}(N)$ or $\mathbf{B}(N)$, observe that the input string for the child of a node is obtained either by trimming off the least significant bit or by adding one first and then trimming off the final bit. Thus, if the input string of a node at level i is α_i, then the input for its children must be either α_{i+1} or β_{i+1}. In the case where the input string is β_i we consider two subcases depending on the last bit. If β_i ends with 0, then its only child is $(\beta_i)/2 = \beta_{i+1}$. Otherwise, if β_i ends with 1, then trimming off the last bit will result in α_{i+1}. If we add one first and then trim the last bit, we will obtain β_{i+1}. $\square$

Since there at most $2(b+1)$ different nodes in the computation tree for $\mathbf{L}(N)$, we can precompute the parent child relationships for all nodes in $O(b^2)$ time. (In fact, if we reuse the suffix details starting at node α_0, we could perform this precomputation in linear time $O(b)$.) After performing this precomputation, we can determine the children of a node in constant time, allowing us to construct the computation tree for $\mathbf{L}(N)$ in constant time per node. Applying this method, the overall running time of the algorithm will be proportional to the number of nodes in the computation tree. If this number is proportional to the number of strings generated (the leaves in the tree), then the algorithm will be CAT. However, in general, this will not be the case due to the large number of degree 1 nodes.

Fortunately, as discussed at the end of the previous subsection, we need only visit the degree 2 nodes (in order) from the computation tree to obtain the Gray code listing. Again, since there are only a linear number of nodes, we can precompute the nearest (left and right) descendants that have degree 2 for each potential node in the computation tree. This can be done in $O(b)$ time, since there are only $O(b)$ nodes to visit. Now, for each node, we can find the nearest left and/or right descendant that has degree 2 in constant time. All that remains is to compute the initial minimal SBR by following the leftmost path in the computation tree and then traversing the degree 2 nodes in order, outputting the original level of each node. When traversing this tree we must be careful to maintain information about subtree reversal so that we know which child branch to visit first.

The following is a detailed summary of the steps required to produce the listing $\mathbf{L}(N)$ in constant amortized time:

1. For $0 \leq i \leq n$ determine the child or children of each node α_i and β_i. The level of these nodes will be i, and from the recursive description of $\mathbf{L}(N)$ we can determine whether or not the subtrees for each child should be reversed. This will take $O(b^2)$ time.
2. For $0 \leq i \leq n$ determine the nearest left and/or right descendant of α_i and β_i that has degree 2. This can be computed in linear time $O(b)$ by starting with $i = b$ and working back to $i = 0$. Details about whether or not the subtrees are to be reversed must be maintained for each degree 2 node.
3. Determine the initial minimal SBR of the listing $\mathbf{L}(N)$ by tracing a path through the virtual computation tree rooted by α_0. This will take linear time $O(b)$.
4. Visit the degree 2 nodes in order, being careful to consider when subtrees are to be reversed. For each level i that is output, modify the current minimal SBR in positions $i+2, i+1, i$. This is done by scanning these 3 characters and applying the appropriate transformation rule: $011 \leftrightarrow 10\bar{1}$ or $0\bar{1}\bar{1} \leftrightarrow \bar{1}01$. The degree 2 nodes can be traversed in constant amortized time; thus the running time of this step will be proportional to the number of minimal SBRs generated.

Observe that the original computation tree is never actually constructed.

THEOREM 3. *The Gray code listing $\boldsymbol{L}(N)$ can be generated in constant amortized time with $O(b^2)$ initialization.*

3. Maximizing the number of minimal SBRs.

If N is represented by the binary string $a_b \cdots a_2 a_1 a_0$, where $a_b = 1$, then there may be only one string in SBR(N) or there could potentially be an exponential number with respect to b. Thus given b, we are interested in finding a tight upper bound on the number of strings in SBR(N), denoted $Max(b)$, as well as a characterization of the bitstrings that obtain this upper bound. Note that the actual length of the binary representation of N is $b+1$. When $b = 0, 1, 2$, the binary representations that produce the maximum number of minimal SBRs are 1, 11, and 110, respectively. The values $Max(0) = 1$ and $Max(1) = Max(2) = 2$. For $3 \leq b \leq 10$ we apply a generation algorithm to determine which binary representations of N have $Max(b)$ strings in SBR(N):

b	Binary representations of N		$Max(b)$
3	1011	1101	3
4	10110	11010	3
5	101101	110011	5
6	1011010	1100110	5
7	10110011	11001101	8
8	101100110	110011010	8
9	1011001101	1100110011	13
10	10110011010	11001100110	13

THEOREM 4. *$Max(b) = f_{\lceil b/2 \rceil+2}$, the $\lceil b/2 \rceil+2^{nd}$ Fibonacci number. Moreover, the SBRs of the two values of N that have $Max(b)$ minimal SBRs where $b \geq 3$ are*

$10(1100)^t11$	and	$11(0011)^t01$	if $b = 4t{+}3$,
$10(1100)^t110$	and	$11(0011)^t010$	if $b = 4t{+}4$,
$10(1100)^t1101$	and	$11(0011)^t0011$	if $b = 4t{+}5$,
$10(1100)^t11010$	and	$11(0011)^t00110$	if $b = 4t{+}6$.

Proof. Applying a generation algorithm, it is trivial to verify the theorem for $3 \leq b \leq 6$. For $b > 6$ we assume that $Max(i) = f_{\lceil \frac{i}{2} \rceil+2}$ for $3 \leq i < b$ (inductive hypothesis) and consider $a_b \cdots a_2a_1a_0$, the binary representation for an integer N. Using the recursive listing $\mathbf{B}(N)$, we will examine each possible suffix of N to determine restrictions on the strings in SBR(N). In particular, we will show that the strings in SBR(N) must end with particular bit sequences for a given suffix.

suffix$(N, 0)$. All strings end with 0. Thus, the maximum number of strings is bounded by $Max(b-1)$. This result implies that $Max(b)$ does not decrease as b increases.

suffix$(N,\, 0(01)^*01)$. Applying the recursive rules twice, all strings must end with 01. Thus, the maximum number of strings is bounded by $Max(b-2)$.

suffix$(N,\, 1(10)^*11)$. Applying the recursive rules twice, all strings must end with $0\bar{1}$. Thus, the maximum number of strings is bounded by $Max(b-2)$.

For the remaining two suffixes, the strings may end with either 1 or $\bar{1}$.

suffix$(N, 11(01)^t01)$. Applying the recursive rules, all strings ending with 1 will end with 01. Otherwise if a string ends with $\bar{1}$, then if $t = 0$, it must end with $00\bar{1}\bar{1}$; if $t = 1$, it must end with $00\bar{1}0\bar{1}\bar{1}$; if $t > 1$, it must end with $\bar{1}0\bar{1}0\bar{1}\bar{1}$. Thus, when the suffix of N is 1101, an upper bound on the maximum number of minimal SBRs is $Max(b-2) + Max(b-4)$. Otherwise if $t > 1$, the upper bound is $Max(b-2) + Max(b-6)$.

suffix$(N, 00(10)^t11)$. Applying the recursive rules, all strings ending with $\bar{1}$ will end with $0\bar{1}$. Otherwise if a string ends with 1, then if $t = 0$, it must end with 0011; if $t = 1$, it must end with 001011; if $t > 1$, it must end with 101011. Thus, when the suffix of N is 0011, an upper bound on the maximum number of minimal SBRs is $Max(b-2) + Max(b-4)$. Otherwise if $t > 1$, the upper bound is $Max(b-2) + Max(b-6)$.

Observe that from our inductive hypotheses that when b is even, $Max(b-1) = Max(b-2) + Max(b-4) = f_{\lceil \frac{b}{2} \rceil+2}$, and when b is odd, $Max(b-1) = f_{\lceil \frac{b}{2} \rceil+2-1}$. Thus an overall upper bound on $Max(b)$ is $f_{\lceil \frac{b}{2} \rceil+2}$. We complete the proof by showing the two strings that obtain this bound, thus proving $Max(b) = f_{\lceil \frac{b}{2} \rceil+2}$. We examine four cases depending on b. Since $b > 6$, we must have $t \geq 1$.

$b = 4t-1$. In this case b is odd, so if $Max(b)$ is to obtain the upper bound of $Max(b-2) + Max(b-4)$, N must have the suffix 1101 or 0011. If it ends with 1101, then in order for the number of strings in SBR(N) that end with 01 to meet the maximum of $Max(b-2)$, the first $b-2$ bits must be either $10(1100)^{t-1}1101$ or $11(0011)^{t-1}0011$ (by induction). Additionally, in order for the number of strings that end with $00\bar{1}\bar{1}$ to meet the maximum of $Max(b-4)$, the first $b-4$ bits must be either $10(1100)^{t-1}11$ or $11(0011)^{t-1}01$ with 1 subtracted as a result of the recursive definition applied to the final 4 bits. Thus taking the union of these criteria over all binary strings, we are left with $N = 11(0011)^{t}01$. A similar examination will show that when N has a suffix 0011, the only string that will obtain the upper bound of $Max(b-2) + Max(b-4)$ strings is $N = 10(1100)^{t}11$.

$b = 4t$. Using induction, if N ends with 0, then $Max(b) = Max(b-1)$ if and only if $N = 10(1100)^{t}110$ or $N = 11(0011)^{t}010$. Otherwise, in order for the size of SBR(N) to meet the upper bound, it must have the suffix 1101 or 0011. If it ends with 1101, the first $b-2$ bits must be either $10(1100)^{t}11010$ or $11(0011)^{t}00110$. However, since neither of these strings ends with 11, no value for N ending with 1101 will meet the upper bound in this case. If it ends with 0011, the first $b-2$ bits must be either $10(1100)^{t}11010$ or $11(0011)^{t}00110$, but this time with 1 subtracted since the strings must end with $0\bar{1}$ (from the recursive definition). However, since neither of the resulting strings ends with 00, no value for N ending with 0011 will meet the upper bound.

$b = 4t+1$. This is similar to the case $b = 4t-1$.

$b = 4t+2$. This is similar to the case $b = 4t$. $\square$

4. Related sequences. If we consider the number of bits required to represent each string in SBR(N) for each value of N starting from $N = 0$, we obtain the following sequence:

$$A = 0, 1, 1, 2, 1, 2, 2, 2, 1, 2, 2, 3, 2, 3, 2, 2, \ldots .$$

For example, the minimum number of bits required to represent the integer 3 as the difference of 2 binary numbers is 2 (given by the fourth element in the sequence). This sequence corresponds to sequence A007302 in Sloane's *The On-Line Encyclopedia of Integer Sequences* [13]. Interestingly, these values also correspond to the cost of grid communications on the Connection Machine [15]. This sequence is also discussed with respect to k-regular sequences in [1].

Another interesting sequence is the one obtained from the number of strings in SBR(N) for each value of N starting from 0:

$$B = 1, 1, 1, 2, 1, 1, 2, 1, 1, 1, 1, 3, 2, 3, 1, 1, \ldots .$$

For example, the fourth element in this sequence is 2 since there are two strings in SBR(3). This sequence corresponds to sequence A110955 in Sloane's *The On-Line Encyclopedia of Integer Sequences* [13].

5. Final remarks. In this paper we have presented a 3-close Gray code algorithm to generate all minimal SBRs of an integer N. After some initialization, the algorithm can be implemented to run in constant amortized time. A CAT implementation is available from the author upon request or at http://www.cis.uoguelph.ca/~sawada/prog.html. As a secondary result, we have precisely identified the values for N that produce the maximum number of minimal SBRs given the length of the binary representation of N.

A preliminary version of this work appears in the proceedings of GRACO 2005 [11]. Since this manuscript was submitted, a related result by Manku and Sawada appeared in the proceedings of ESA 2005 [7]. In that work, a loopless algorithm to list all minimal SBRs of an integer N is provided. The loopless algorithm is based on the binary reflected Gray code and is significantly more complex than the simple recursive description given in this paper.

Acknowledgments. The author would like to thank the anonymous referees and Frank Ruskey for helpful comments to improve the final version of this paper.

REFERENCES

[1] J. P. Allouche and J. Shallit, *The ring of k-regular sequences*, II, Theoret. Comput. Sci., 307 (2003), pp. 3–29.

[2] A. D. Booth, *A signed binary multiplication technique*, Quart. J. Mech. Appl. Math., 4 (1951), pp. 236–240.

[3] W. E. Clark and J. J. Liang, *On arithmetic weight for a general radix representation of integers*, IEEE Trans. Inform. Theory, 19 (1973), pp. 823–826.

[4] U. Güntzer and M. Paul, *Jump interpolation search trees and symmetric binary numbers*, Inform. Process. Lett., 26 (1987), pp. 193–204.

[5] I. Stoica, R. Morris, D. Liben-Lowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, *Chord: A scalable peer-to-peer lookup protocol for Internet applications*, IEEE/ACM Trans. Networking, 11 (2003), pp. 17–32.

[6] P. Ganesan and G. S. Manku, *Optimal routing in Chord*, in Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), New Orleans, 2004, pp. 169–178.

[7] G. S. Manku and J. Sawada, *A loopless Gray code for minimal signed-binary representations*, in Proceedings of the 13th Annual European Symposium on Algorithms (ESA), Lecture Notes in Comput. Sci. 3669, Springer-Verlag, Berlin, 2005, pp. 438–447.

[8] K. Okeya, K. Schmidt-Samoa, C. Spahn, and T. Takagi, *Signed binary representations revisited*, in Advances in Cryptology—CRYPTO 2004, Lecture Notes in Comput. Sci. 3152, Springer-Verlag, Berlin, 2004, pp. 123–139.

[9] H. Prodinger, *On binary representations of integers with digits* -1, 0, 1, Integers (2000), A8, 14 pp.

[10] G. W. Reitwiesner, *Binary arithmetic*, in Advances in Computers, Vol. 1, Academic Press, New York, 1960, pp. 231–308.

[11] J. Sawada, *A Gray code for binary subtraction*, in Proceedings of GRACO 2005, Electron. Notes Discrete Math., 19 (2005), pp. 125–131.

[12] K. Schmidt-Samoa, O. Semay, and T. Takagi, *Analysis of fractional window recoding methods and their application to elliptic curve cryptosystems*, IEEE Trans. Comput., 55 (2006), pp. 48–57.

[13] N. Sloane, *The on-line encyclopedia of integer sequences*: ID A007302, A110955, http://www.research.att.com/~njas/sequences/index.html (2006).

[14] T. Takagi, D. Reis, Jr., S. Yen, and B. Wu, *Radix-r nonadjacent form and its application to pairing-based cryptosystem*, IEICE Trans. Fund. Elec., Comm., & Comp. Sci., E89-A (2006), pp. 115–123.

[15] A. Weitzman, *Transformation of parallel programs guided by micro-analysis*, in Algorithms Seminar (1992–1993), B. Salvy, ed., INRIA, Rocquencourt, France, 1993, Rapport de Recherche 2130, pp. 155–159.