PUBLICATIONS DE L'INSTITUT MATHÉMATIQUE
Nouvelle série, tome 89(103) (2011), 37–47 DOI: 10.2298/PIM1103037K

A LOOPLESS IMPLEMENTATION OF A GRAY CODE FOR SIGNED PERMUTATIONS

James Korsh, Paul LaFollette, and Seymour Lipschutz

Communicated by Slobodan Simić

Abstract. Conway, Sloane and Wilks [1989] proved the existence of a Gray code for the reflection group B_n. The elements of this group are the signed permutations of the set $1, 2, \dots, n$. Here we give a loopless algorithm which generates a specific Gray code for B_n.

1. Introduction

The original idea of a Gray code was to list all n-bit strings, sequences of 0's and 1's of length n, called *codewords*, so that successive codewords differ in only one bit. Q_n is used to denote all codewords of length n. The fact that this can be done for any n follows from the construction of the Binary Reflected Gray Code (BRGC) for Q_n (see [5]) which we describe below.

The above idea of a Gray code has been generalized as follows. A Gray code for any combinatorial family of objects is a listing of the objects such that successive objects differ in some prescribed, usually "small", way. The definition of "small" depends on the particular family, its context, and its applications. Gray codes are useful in studying the properties of the objects being generated because the less successively generated objects differ from each other, the faster the properties being studied can be updated in passing from one object to the next. Carla Savage [10] has an excellent survey of Gray codes with 154 references.

Consider, for example, S_n, the set of all permutations of the integers 1 to n. A Gray code for S_n may be defined to be a list L of the permutations in S_n such that successive permutations differ by a *transposition*, the interchange of two integers. One such famous Gray code for S_n was given by Johnson [7] and Trotter [12], apparently independently. In fact, their Gray code uses only adjacent interchanges. We discuss this Gray code in detail later.

2010 *Mathematics Subject Classification:* Primary 68R05; Secondary 05A05.
Key words and phrases: algorithms; combinatorial; loopless; gray code; reflection groups; signed permutations.

Now consider any finite group G with a set of generators. A Gray code for G is a list L of the elements of G such that each element is obtained from the previous one by applying one of the generators. This is equivalent to a Hamiltonian path in the Caley graph of the group.

Conway, Sloane and Wilks (CSW) [2] proved the existence of a Gray code for all the finite reflection groups; the term reflection comes from the fact that each generator of the group has order two. This includes the infinite family of reflection groups B_n ($n \geqslant 2$) (which are defined below). The elements of these groups are in fact the signed permutations of the set $\{1, 2, \ldots, n\}$. Signed permutations are used, for example, in bioinformatics [4, 11, 14]. Rankin [9] showed that the alternating group $A(6)$ with generators (2,4,6,5,3) and (1,6,3)(2,4,5) does not have a Gray code.

We note that an algorithm which generates the elements of a combinatorial family is said to be *loopless* if it takes no more than a constant amount of time between the elements. Gidean Ehrlich [3] first formulated this notion of a loopless algorithm. This paper gives a loopless algorithm which generates a specific Gray code for B_n. The algorithm uses a loopless algorithm by Bitner, Ehrlich and Reingold (BER) [1] for the BRGC for Q_n, and our loopless version of the Johnson–Trotter listing for S_n.

Our paper is organized as follows. Section 2 discusses the Binary Reflected Gray Code for Q_n, and the loopless algorithm generating it by BER [1]. Section 3 discusses the Johnson–Trotter Gray code listing for S_n and our loopless algorithm for generating it. Section 4 discusses the reflection group B_n, Section 5 discusses our algorithm for its generation, and Section 6 gives two loopless versions of the algorithm. The second version uses integers no larger than n.

2. Binary reflected gray code

Since a Gray code for Q_n is a list of all codewords of Q_n such that successive codewords differ by only one bit, the Gray code is completely determined by its *transition sequence*, the list of the bit positions which change as we go from one codeword to the next in the sequence.

There are many Gray codes for Q_n. We are only interested in the Binary Reflected Gray Code (BRGC) for Q_n. There is a simple recursive construction of this Gray code. Specifically, Table 1 shows how the BRGC for Q_4 can be obtained from the BRGC for Q_3. That is, first we list the Gray code for Q_3 which appears as the upper left 3×8 matrix (where the codewords are the columns). Then next to it we list the Gray code for Q_3 but in reverse order. This yields a 3×16 matrix. Finally, we add a fourth row consisting of eight 0's followed by eight 1's. This gives the BRGC for Q_4.

We show that this is in fact a Gray code for Q_4. Observe that only one bit changes within the first eight columns since the bottom bit is always 0, and only one bit changes within the last eight columns since the bottom bit is always 1. Furthermore, the two middle columns (and also the first and last columns) are identical except for the added last bits, and so again only one bit changes. The BRGC for Q_n is the one obtained recursively in this way beginning with the matrix $[0, 1]$ for Q_1.

Frank Gray, for whom Gray codes are named, first specified this recursive construction in [6]. Discussing the BRGC, Herbert Wilf [14] comments: "This method of copying a list and its reversal . . . seems to be a good thing to think of when trying to construct some new kind of Gray code."

	Q_3	Q_3 reversed
	0 1 1 0 0 1 1 0	0 1 1 0 0 1 1 0
	0 0 1 1 1 1 0 0	0 0 1 1 1 1 0 0
	0 0 0 0 1 1 1 1	1 1 1 1 0 0 0 0
	0 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1
T_4:	4 3 4 2 4 3 4 1 4 3 4 2 4 3 4	

TABLE 1. Binary reflected gray code for Q_4

Observe that the transition sequence $T_4 = [4, 3, 4, \ldots, 3, 4]$ for the BRGC for Q_4 also appears in Table 1, where we view the codewords from the bottom ("first" row) to the top ("fourth" row). That is, first the fourth bit changes, then the third bit changes, then the fourth bit changes again, and then the second bit changes, and so on. BER [1] gave a very clever loopless algorithm which generates the transition sequence T_n using an array $[t_0, t_1, t_2, \ldots, t_n]$. We essentially use this algorithm (in our loopless algorithm generating a Gray code for B_n) to go both forward and backward in the BRGC for Q_n.

3. Johnson–Trotter list

The Johnson–Trotter permutation list is also defined recursively. The list for $n = 4$ is shown in Table 2. Note that the list is partitioned into six "blocks", each with four successive permutations. Each block corresponds to, and is labeled by, a permutation in S_3 which is in boldface and is on top of the block. We note that the boldface labels form the Johnson–Trotter list for S_3.

123	**132**	**312**	**321**	**231**	**213**
1234	4132	3124	4321	2314	4213
1243	1432	3142	3421	2341	2413
1423	1342	3412	3241	2431	2143
4123	1324	4312	3214	4231	2134

TABLE 2. Johnson–Trotter list for S_4

Observe that in the first block, the largest item 4 sweeps from right to left, in the second block from left to right, in the third block from right to left, and so on. Also, the relative positions of the remaining items $1, 2, 3$ do not change in each block and correspond to the label of the block. Moreover, the recursive changes of the relative positions of $1, 2$, and 3 occur only from block to block when the largest item 4 is in an end position so the item 4 does not interfere with any transposition involving the items $1, 2$, and 3. Thus the Johnson–Trotter list $\mathrm{JT}(n)$ is a Gray code for S_n. (In fact, successive permutations in $\mathrm{JT}(n)$ differ by only an adjacent transposition.)

Ehrlich [3] gave the first loopless algorithm to generate the Johnson–Trotter list. An alternate loopless algorithm for the JT list appears in [8]. We have to move both forward and backward in the JT list, so our algorithm, although similar to the Ehrlich [3] algorithm, will be more involved. We describe one part of our algorithm as follows.

Suppose we first consider moving forward in the JT list. We use two arrays d and e which we describe below.

(a) Each item (number) in any permutation has a direction, LEFT or RIGHT. Whenever the item reaches an end position in its respective subpermutation it changes its direction. All items begin with the direction (moving) LEFT. (For example, the item 4, in Table 2, changes its direction from LEFT to RIGHT after the fourth permutation 4123 where 4 has reached an end position.) Array d will contain the directions of the items in the permutations.

(b) Our algorithm implicitly uses two lists, a “mobile" list and a “finished" list, and the lists are ordered with the largest element first. Each item is on exactly one of the two lists. The item that moves will be the first element on the mobile list and, when it moves, all larger items on the finished list are inserted at the beginning of the mobile list, retaining their relative order. An item on the mobile list is moved to the finished list when it reaches an end position in its respective subpermutation. The algorithm ends when the mobile list is empty or, equivalently, when all items are on the finished list. Array e keeps track of the items on the mobile lists and on the finished lists. Similarly, we use arrays D and E, analogous to d and e, when moving backward in the JT list.

To explain how these arrays can be updated in constant time consider array e. The item at the front of the mobile list is always in $e[n+1]$. If this item is j, then its successor, $j1$, will be in $e[j]$. Similarly, $j1$'s successor will be in $e[j1]$ and so on. As an example, if n is 6 and $e[1]\,e[2]\,e[3]\,e[4]\,e[5]\,e[6]\,e[7]$ are $0\,1\,1\,3\,4\,5\,4$, then the list is $1, 3, 4$ where 4 is $e[7]$, 3 is $e[4]$, and 1 is $e[3]$. The finished list contains items not on the mobile list $(2, 5$, and 6 here).

If the moving item j finishes, it must be removed from the list. It is removed by setting $e[j+1]$ to $e[j]$. This works because $e[j+1]$ must hold the successor of $j+1$ in the updated list and this successor is the current successor of j, which is in $e[j]$.

In addition, $e[j]$ is set to $j-1$. Since j can move only when $j+1$ to n are all finished, as $j+1$ to n finish, this will have set $e[j+1]$ to j, $e[j+2]$ to $j+2, \ldots$, and $e[n]$ to $n-1$. Consequently, when $e[n+1]$ is set to n, the list becomes $j+1, j+2, \ldots, n$.

If the moving item does not finish, then n should become the item at the front of the list and setting $e[n+1]$ to n accomplishes this.

E must also be updated. Remember that E controls the reverse generation of S_n. When j is finished, this entails simply setting $E[n+1]$ to j. Suppose j is not finished. If $E[j+1]$ is j, then j is on the mobile list of E so all that is required is again setting $E[n+1]$ to j. However, if $E[j+1]$ is not j, then j must have finished (in the reverse generation) and so was taken off E's list by first setting $E[j+1]$ to $E[j]$ and then setting $E[j]$ to $j-1$. Restoring it requires first setting $E[j]$ to $E[j+1]$ then setting $E[j+1]$ to j and again setting $E[n+1]$ to j.

These changes and determining when j is finished take at most constant time. Updating d and D is straightforward and also takes at most constant time.

4. Reflection group B_n

The reflection group B_n with generators $R_1, R_2, \ldots, R_n$ may be represented by the permutations of $1, 2, \ldots, n$ with each integer having a $+$ or $-$ sign attached to it, so each integer can be positive or negative – the signed permutations. (We will let boldface numbers indicate integers with $-$ signs.) Thus B_n has order $2^n n!$. In particular, B_2 has $|B_2| = 2^2 2! = 8$ elements which follow:

$$B_2\colon \{12, 1\mathbf{2}, \mathbf{2}1, \mathbf{21}, \mathbf{12}, \mathbf{1}2, 2\mathbf{1}, 21\}$$

The generators $R_1, R_2, \ldots, R_{n-1}$ of B_n correspond to the adjacent transpositions $(12), (23), (34), \ldots, (n-1, n)$ and hence $R_1, R_2, \ldots, R_{n-1}$ generate a subgroup H of B_n which is isomorphic to S_n. The generator R_n changes the sign of the last coordinate. Observe that the above list for B_2 is in fact a Gray code for B_2.

Using 0 to denote $+$ and 1 to denote $-$, an element z in B_n may be represented by a pair (p, g) where:

(i) p (for permutation) belongs to S_n: p is the list of the numbers in z without any signs;

and

(ii) g (for Gray code) belongs to Q_n: g denotes the numbers (not positions) in z which are negative.

For example:

$$z = 3\mathbf{6}4\mathbf{25}1 \text{ in } B_6 \text{ corresponds to } (364251,\ 010011) \text{ and}$$
$$z = \mathbf{4}1\mathbf{8}56\mathbf{3}27 \text{ in } B_8 \text{ corresponds to } (41856327,\ 00110001)$$

Our algorithm will output the elements of B_n as pairs (p, g). We will illustrate our algorithm using B_3 and then B_4 as examples.

Table 3 pictures B_3, where the $|B_3| = 2^3 3! = 48$ elements of B_3 are arranged in an array. Note that the columns of the array are labeled by the BRGC for Q_3 and the rows are labeled by the Johnson–Trotter list for S_3.

The first column H under 000, which consists of all permutations where the signs are all $+$, is the subgroup H of B_3, which is isomorphic to S_3. Each of the other columns is a coset of H. Since the first column H is a Johnson–Trotter list for S_3, we can move up or down any column using the generators $R_1, R_2, \ldots, R_{n-1}$. On the other hand, we can move between adjacent columns using *only* the generator R_n which negates the last item in the permutation. The double arrow in Table 3 indicates an edge between the adjacent cosets (columns). Observe there are exactly two edges between any two adjacent columns.

Our algorithm will generate the following Hamiltonian path (Gray code) for B_3:

	000	001	011	010	110	111	101	100
123	$123 \leftrightarrow$	$12\mathbf{3}$	$1\mathbf{23} \leftrightarrow$	$1\mathbf{2}3$	$\mathbf{12}3 \leftrightarrow$	$\mathbf{123}$	$\mathbf{1}2\mathbf{3} \leftrightarrow$	$\mathbf{1}23$
132	132	$1\mathbf{3}2 \leftrightarrow$	$1\mathbf{32}$	$13\mathbf{2}$	$\mathbf{1}3\mathbf{2}$	$\mathbf{132} \leftrightarrow$	$\mathbf{13}2$	$\mathbf{1}32$
312	312	$\mathbf{3}12 \leftrightarrow$	$\mathbf{3}1\mathbf{2}$	$31\mathbf{2}$	$3\mathbf{12}$	$\mathbf{312} \leftrightarrow$	$\mathbf{31}2$	$3\mathbf{1}2$
321	321	$\mathbf{3}21$	$\mathbf{32}1$	$3\mathbf{2}1 \leftrightarrow$	$3\mathbf{21}$	$\mathbf{321}$	$\mathbf{3}2\mathbf{1}$	$32\mathbf{1}$
231	231	$2\mathbf{3}1$	$\mathbf{23}1$	$\mathbf{2}31 \leftrightarrow$	$\mathbf{2}3\mathbf{1}$	$\mathbf{231}$	$2\mathbf{31}$	$23\mathbf{1}$
213	$213 \leftrightarrow$	$21\mathbf{3}$	$\mathbf{2}1\mathbf{3} \leftrightarrow$	$\mathbf{2}13$	$\mathbf{21}3 \leftrightarrow$	$\mathbf{213}$	$2\mathbf{13} \leftrightarrow$	$2\mathbf{1}3$

TABLE 3. Reflection group B_3

$123 \to 12\mathbf{3}, 1\mathbf{3}2 \to 1\mathbf{32}, 1\mathbf{23} \to 1\mathbf{2}3, 13\mathbf{2}, 31\mathbf{2}, 3\mathbf{2}1 \to 3\mathbf{21}, 3\mathbf{12}, \mathbf{1}3\mathbf{2}, \mathbf{12}3 \to \mathbf{123}, \mathbf{132} \to \mathbf{13}2, \mathbf{123} \to \mathbf{1}23$ and down the column to $2\mathbf{1}3$
$\mapsto 2\mathbf{13}, 2\mathbf{31}, \mathbf{3}2\mathbf{1}, \mathbf{31}2 \mapsto \mathbf{312}, \mathbf{321}, \mathbf{231}, \mathbf{213} \mapsto \mathbf{21}3, \mathbf{2}3\mathbf{1} \mapsto \mathbf{2}31, \mathbf{2}13 \mapsto \mathbf{2}1\mathbf{3}, \mathbf{23}1, \mathbf{32}1,$
$\mathbf{3}1\mathbf{2} \mapsto \mathbf{3}12, \mathbf{3}21, 2\mathbf{3}1, 21\mathbf{3} \mapsto 213$ and up the column to 132.
That is, the path moves from the first column to the last and then back again, and:

(i) Each of the seven $\to$ denotes a move from a column to the next column.

(ii) Each of the seven $\mapsto$ denotes a move from a column to the preceding column.

(iii) Each comma "," denotes a move up or a move down within a column.

It would be very instructive if the reader followed the path through Table 3.

Table 4 pictures B_4 where the $|B_4| = 2^4 4! = 384$ elements of B_4 are again arranged in an array. The columns are labeled by the BRGC for Q_4 and the rows are labeled by the Johnson–Trotter list for S_4. Here a star $*$ is used instead of $\leftrightarrow$ to denote an edge between columns. Observe that here there are six edges between adjacent columns. We will also use the underlined star $\underline{*}$ to denote the first edge between the columns.

5. Algorithm generating a gray code for B_n

This section presents our algorithm generating a Gray code for B_n which is similar to the one for B_3. First of all we assume the elements of B_n are arranged in an array where the columns of the array are labeled by the BRGC for Q_n and the rows are labeled by the Johnson–Trotter list for S_n. (This was done in Table 3 for B_3 and in Table 4 for B_4).

The first column H under $00\ldots0$ is the subgroup of B_n which is isomorphic to S_n, and each of the other columns is a coset of H. There will always be an edge in our path for B_n in the first row and in the last row between each odd column and the next even column, that is, between columns 1 and 2, between columns 3 and 4, and so on until between (the last two) columns $2^n - 1$ and 2^n. These edges will be called *special edges*.

Our path will have two parts, A and B, where A moves forward through the array and down the last column and B moves backward through the array and up the first column. Specifically:

(A) Part A begins at the element $x = 123\ldots n$ (in the first row, first column). Then either:

	0000	0001	0011	0010	0110	0111	0101	0100	1100	1101	1111	1110	1010	1011	1001	1000
1234	1234 $\underline{*}$	123**4**	12**34** $\underline{*}$	12**3**4	1**23**4 $\underline{*}$	1**234**	1**2**3**4** $\underline{*}$	1**2**34	**12**34 $\underline{*}$	**12**3**4**	**1234** $\underline{*}$	**123**4	**1**2**3**4 $\underline{*}$	**1**2**34**	**1**23**4** $\underline{*}$	**1**234
1243	1243	12**4**3 $\underline{*}$	12**43**	124**3**	1**2**4**3**	1**243** $\underline{*}$	1**24**3	1**2**43	**12**43	**124**3 $\underline{*}$	**1243**	**12**4**3**	**1**24**3**	**1**2**43** $\underline{*}$	**1**2**4**3	**1**243
1423	1423	1**4**23 $*$	1**4**2**3**	142**3**	14**23**	1**423** $*$	1**42**3	14**2**3	**1**4**2**3	**142**3 $*$	**1423**	**1**4**23**	**1**42**3**	**14**2**3** $*$	**14**23	**1**423
4123	4123	**4**123 $*$	**4**12**3**	412**3**	41**23**	**4**1**23** $*$	**4**1**2**3	41**2**3	4**12**3	**412**3 $*$	**4123**	4**123**	4**1**2**3**	**41**2**3** $*$	**41**23	4**1**23
4132	4132	**4**132	**4**1**3**2	41**3**2 $\underline{*}$	41**32**	**4**1**32**	**4**13**2**	413**2**	4**1**3**2**	**41**3**2**	**4132**	4**132** $\underline{*}$	4**13**2	**413**2	**41**32	4**1**32
1432	1432	1**4**32	1**43**2	14**3**2 $*$	14**32**	1**432**	1**4**3**2**	143**2**	**1**43**2**	**14**3**2**	**1432**	**1**4**32** $*$	**1**4**3**2	**143**2	**14**32	**1**432
1342	1342	13**4**2	1**34**2	1**3**42 $*$	1**3**4**2**	1**342**	13**42**	134**2**	**1**34**2**	**1**3**42**	**1342**	**13**4**2** $*$	**13**42	**134**2	**1**3**4**2	**1**342
1324	1324 $*$	132**4**	1**3**2**4** $*$	1**3**24	1**32**4 $*$	1**324**	13**24** $*$	13**2**4	**1**3**2**4 $*$	**1**3**24**	**1324** $*$	**132**4	**13**24 $*$	**13**2**4**	**1**32**4** $*$	**1**324
3124	3124 $*$	312**4**	**3**12**4** $*$	**3**124	**3**1**2**4 $*$	**3**1**24**	31**24** $*$	31**2**4	3**12**4 $*$	3**124**	**3124** $*$	**312**4	**31**24 $*$	**31**2**4**	3**1**2**4** $*$	3**1**24
3142	3142	31**4**2	**3**1**4**2	**3**142 $*$	**3**14**2**	**3**1**42**	31**42**	314**2**	3**1**4**2**	3**142**	**3142**	**31**4**2** $*$	**31**42	**314**2	3**14**2	3**1**42
3412	3412	3**4**12	**34**12	**3**412 $*$	**3**41**2**	**34**1**2**	3**4**1**2**	341**2**	34**12**	3**412**	**3412**	**3**4**12** $*$	**3**4**1**2	**341**2	3**41**2	34**1**2
4312	4312	**4**312	**43**12	4**3**12 $*$	4**3**1**2**	**43**1**2**	**4**31**2**	431**2**	43**12**	**4**3**12**	**4312**	4**312** $*$	4**31**2	**431**2	**4**3**1**2	43**1**2
4321	4321	**4**321	**43**21	4**3**21	4**32**1	**432**1	**4**3**2**1	43**2**1 $\underline{*}$	43**21**	**4**3**21**	**4321**	4**321**	4**3**2**1**	**43**2**1**	**4**32**1**	432**1**
3421	3421	3**4**21	**34**21	**3**421	**3**4**2**1	**342**1	3**42**1	34**2**1 $*$	34**21**	3**421**	**3421**	**3**4**21**	**3**42**1**	**34**2**1**	3**4**2**1**	342**1**
3241	3241	32**4**1	**3**2**4**1	**3**241	**32**41	**324**1	3**24**1	3**2**41 $*$	3**2**4**1**	3**241**	**3241**	**32**4**1**	**3**24**1**	**3**2**41**	32**41**	324**1**
3214	3214 $*$	321**4**	**3**21**4** $*$	**3**214	**32**14 $*$	**32**1**4**	3**2**1**4** $*$	3**2**14	3**21**4 $*$	3**214**	**3214** $*$	**321**4	**3**2**1**4 $*$	**3**2**14**	32**14** $*$	32**1**4
2314	2314 $*$	231**4**	2**3**1**4** $*$	2**3**14	**23**14 $*$	**23**1**4**	**2**31**4** $*$	**2**314	**2**3**1**4 $*$	**2**3**14**	**2314** $*$	**231**4	2**31**4 $*$	2**314**	23**14** $*$	23**1**4
2341	2341	23**4**1	2**34**1	2**3**41	**23**41	**234**1	**2**3**4**1	**2**341 $*$	**2**34**1**	**2**3**41**	**2341**	**23**4**1**	2**3**4**1**	2**341**	23**41**	234**1**
2431	2431	2**4**31	2**43**1	24**3**1	**2**4**3**1	**243**1	**24**31	**2**431 $*$	**2**43**1**	**24**3**1**	**2431**	**2**4**31**	24**31**	2**431**	2**4**3**1**	243**1**
4231	4231	**4**231	**4**2**3**1	42**3**1	4**23**1	**423**1	**42**31	4**2**31 $*$	4**2**3**1**	**42**3**1**	**4231**	4**231**	42**31**	**4**2**31**	**4**23**1**	423**1**
4213	4213	**4**213 $*$	**4**21**3**	421**3**	4**2**1**3**	**42**1**3** $*$	**42**13	4**2**13	4**21**3	**421**3 $*$	**4213**	4**213**	42**13**	**4**2**13** $*$	**4**2**1**3	42**1**3
2413	2413	2**4**13 $*$	2**4**1**3**	241**3**	**2**41**3**	**24**1**3** $*$	**24**13	**2**413	**2**4**1**3	**241**3 $*$	**2413**	**2**4**13**	24**13**	2**413** $*$	2**41**3	24**1**3
2143	2143	21**4**3 $*$	21**43**	214**3**	**2**14**3**	**2**1**43** $*$	**2**1**4**3	**2**143	**21**43	**214**3 $*$	**2143**	**21**4**3**	2**1**4**3**	2**143** $*$	2**14**3	2**1**43
2134	2134 $*$	213**4**	21**34** $*$	21**3**4	**2**1**3**4 $*$	**2**1**34**	**2**13**4** $*$	**2**134	**21**34 $*$	**21**3**4**	**2134** $*$	**213**4	2**13**4 $*$	2**134**	2**1**3**4** $*$	2**1**34

TABLE 4. Reflection group B_4

(i) it will move from one column to the next column using the topmost edge between successive columns. This edge is alternately the top special edge and another topmost edge, starting and ending with the top special edge or

(ii) it will move up or down within a column. After arriving at the last column using the top special edge it will move all the way down the last column to the last entry $y = \mathbf{213} \ldots n$ in the column.

(b) Part B begins at the element y (in the last row, last column). Then either:

(i) it will move from one column to the preceding column using alternately the bottom special edge and the edge just below the topmost edge between successive columns (which is not a top special edge), starting and ending with the bottom special edge or

(ii) it will move up or down within a column. After arriving at the first column using the bottom special edge it will move all the way up the first column to its second entry, $123 \ldots (n-2)n(n-1)$.

It would be instructive if the reader used our algorithm to follow the Gray code for B_4 in Table 4. We note that we move from Column 4 to Column 5 in Row 5, and hence on the way back we move from Column 5 to Column 4 in Row 6. Similarly, we move from Column 8 to Column 9 in Row 13, and hence on the way back we move from Column 9 to Column 8 in Row 14.

The path this algorithm traces is Hamiltonian because each topmost edge that is not the top special edge has a second edge immediately below it in the next row. In Section 6 we will see that this second edge always exists.

Note that this Gray code is circular since the last element differs from the first by an adjacent interchange, so the Hamiltonian path we generate can be extended to a Hamiltonian cycle by adding one more edge.

We emphasize that there are many Gray codes for B_n. We use the first edge when moving forward; but other edges could also have been used to yield a Gray code for B_n. It is important to note that we cannot simply generate entire columns, one after another, because the generator R_n *must* be used to go between columns.

6. Loopless algorithm for our gray code for B_n

Implementing our algorithm looplessly, we need to do the following three things:

(1) Looplessly move from one column to an adjacent column or, equivalently, looplessly move between elements of the BRGC for Q_n. This we do using a modification of the algorithm by BER [1].

(2) Looplessly move up and down a column or, equivalently, looplessly move between elements of the symmetric group S_n. This we do using a loopless version of the JT list. This version uses the arrays d, e, D and E described above. When moving down, our algorithm uses d and e and looplessly updates D and E; and when moving up it uses D and E and looplessly updates d and e.

(3) Keep track of each row where we move from one column to the next column so on the way back we know the row we must use to go backward. This can be done in two different ways:

(a) We can use an array to keep track of the row position when we move to the next column. Unfortunately the row position may be a very large number, that is, of order $n!$.

(b) We can keep in memory a two-dimensional $n \times n$ array r whose rows are the permutations corresponding to those edges where we move forward from one column to the next.

For example, for B_4:

$$r = \begin{bmatrix} 4 & 3 & 2 & 1 \\ 4 & 1 & 3 & 2 \\ 1 & 2 & 4 & 3 \\ 1 & 2 & 3 & 4 \end{bmatrix}.$$

The numbers in the last column are all different since we move from column to column depending on the last number in the permutation. Row s of array r is always the first permutation in the JT list, where s occurs at the right end of the permutation. Also, s is the position of the bit that changes in the codeword as we move to the next column.

When $s < n$, row s is where our algorithm moves forward to the next column using a topmost edge that is not the top special edge. Since s will have just occurred at the right end, s must remain at the right end in the next row with its sign unchanged and the element to its right must differ from it only in the sign of s. The next element in the path traced by our algorithm is in the same row as row s but the next column and differs from row s only in the sign of s. Hence there is always an edge in the row just below row s.

In general, row s has the form

$$[\text{evens} > (s+2)], (s+2), 1, 2, \ldots, (s-1), (s+1), [\text{odds} > (s+2)], s$$

where the evens decrease from left to right and the odds increase from left to right. For example, for $n = 16$ and $s = 7$, we have the row

$$16, 14, 12, 10, 9, 1, 2, 3, 4, 5, 6, 8, 11, 13, 15, 7$$

This second version uses only integers that are no larger than n.

We also point out that this version must *looplessly* determine when, as it moves *up a column on the way back*, it has encountered the permutation corresponding to the (first) edge where it earlier *moved forward to the current column.* When that permutation is encountered, we backtrack to the edge below it and then move left to the preceding column. Since the last permutation in each column is always $2134\ldots n$, it is not necessary to check for that permutation until $p[n]$ is s in the current permutation. After that has occurred, we compare exactly one item in the current permutation with the corresponding item in row s of the array r. There are only two cases that can occur for this comparison, when the first item of row $r, r[s][1]$ is n, and when it is not n. These are illustrated below for $n = 7$. When s is 5, row s is 7 1 2 3 4 6 5 so $r[s][1]$ is 7 and when s is 4, row s is 6 1 2 3 5 7 4 so $r[s][1]$ is not 7:

$s = 5$	$s = 4$
7 1 2 3 4 6 5	6 1 2 3 **5** 7 4
1 7 **2** 3 4 6 5	6 1 2 **3** 7 5 4
1 2 7 **3** 4 6 5	6 1 **2** 7 3 5 4
1 2 3 7 **4** 6 5	6 **1** 7 2 3 5 4
1 2 3 4 7 **6** 5	**6** 7 1 2 3 5 4
1 2 3 4 6 7 5	7 6 1 2 3 5 4

These are the permutations that are encountered when moving up in a column looking for row s in the two cases. In the first case, $n = 7$ is moving left and in the second case, $n = 7$ is moving right. We can looplessly determine when row s has been reached by checking that the corresponding item in the current permutation matches the boldface item in row s. When all these required matches have occurred, we are at the permutation we must backtrack from.

Our two loopless algorithms for $n > 2$ written in C^{++}, are available at:
http://www.cis.temple.edu/~korsh/reflectionswithcounts.c++ and
http://www.cis.temple.edu/~korsh/reflectionswithnocounts.c++

Both programs represent the elements using an array g and an array p as described in Section 4. Array p contains positive integers from 1 to n.

Two loopless programs which represent the elements more directly as signed permutations using an array p of positive and negative integers from 1 to n are available at:
http://www.cis.temple.edu/~korsh/signedpermutationswithcounts.c++ and
http://www.cis.temple.edu/~korsh/signedpermutationswithnocounts.c++

Acknowledgement. The authors would like to thank a referee for suggestions which significantly improved the paper.

References

1. J. R. Bitner, G. Ehrlich, and E. M. Reingold, *Efficient generation of the binary reflected Gray code and its applications*, Comm. Assoc. Comput. Mach. **19** (1976), 517–521.
2. J. H. Conway, N. J. A. Sloane and A. R. Wilks, *Gray codes for reflection groups*, Graphs Combin. **5** (1989) 315–325.
3. G. Ehrlich, *Loopless algorithms for generating permutations, combinations, and other combinatorial configurations* J. Assoc. Comput. Mach. **20** (1973), 500–513.
4. M. Figeac and J.-S. Varre, *Sorting by Reversals with Common Intervals*, in: *Algorithms in Bioinformatics: 4th International Workshop, WABI 2004, Bergen, Norway, September 17–21, 2004, Proceedings* (Lect. Notes Comput. Sci. and Lect. Notes Bioinformatics), 26–37.
5. E. N. Gilbert, *Gray codes and paths on the n-cube*, Bell Syst. Tech. J. **37** (1958), 815–826.
6. F. Gray, *Pulse code communication*, March 17, 1953 (filed Nov. 1947). U.S. Patent 2,632,058.
7. S. M. Johnson, *Generation of permutations by adjacent transposition*, Math. Comput. **17** (1963), 282–285.
8. J. Korsh and S. Lipschutz, *Generating multiset permutations in constant time*, J. Algorithms **25** (1997), 321–335.
9. R. A. Rankin, *A campanological problem in group theory*, Proc. Camb. Philos. Soc. **44** (1948), 17–25.
10. C. Savage, *A survey of combinatorial Gray codes*, SIAM Rev. **39**(4) (1997), 605–629.

11. K. M. Swenson, V. Rajan, Yu Lin, and B. M. Moret, *Sorting Signed Permutations by Inversions in $O(n \log n)$ Time*, RECOMB 2'09, Proc. 13th Annual Internat. Conf. on Research in Computational Molecular Biology, 386–399.
12. H. F. Trotter, *Algorithm* 115*, Permutations*, Comm. Assoc. Comput. Mach. **5** (1962), 434–435.
13. H. Wilf, *Combinatorial Algorithms – an Update*, SIAM, Philadelphia, 1989.
14. S. Yancopoulos, O. Attie, and R. Friedberg, *Efficient sorting of genomic permutations by translocation, inversion and block interchange*, Bioinformatics **21**(16) (2005), 3340–3346.

Department of CIS (Korsh and LaFollette)
Department of Mathematics (Lipschutz)
Temple University
Philadelphia, PA 19122
U.S.A.
korsh@temple.edu
paul.lafollette@temple.edu
seymour@temple.edu

(Received 06 01 2010)
(Revised 19 01 2011 and 08 02 2011)