PU.M.A. Vol. **17** (2006), No. 1–2, pp. 135–146

Gray code and loopless algorithm for the reflection group D_n

James Korsh
Department of Computer Science
Temple University

and

Seymour Lipschutz
Department of Computer Science
Temple University

(Received: October 31, 2006)

Abstract. Conway, Sloane and Wilks [2] prove the existence of a Hamiltonian circuit (Gray code) for the Cayley graphs of the finite reflection groups $A_n[\cong S_{n+1}]$, B_n, and D_n. Here, we give a loopless algorithm which generates a specific Gray code for D_n. A loopless algorithm for generating $S_{n+1} \cong A_n$ was first given by Ehrlich [3] and, recently, a loopless algorithm generating B_n was given by Korsh, LaFollette and Lipschutz [6].

Mathematics Subject Classifications (2000). 94B60, 68R05, 20F65, 05C38

Keywords: algorithms; combinatorial problems; loopless; Gray code; reflection groups.

1 Introduction

The original idea of a Gray code was to list the "codewords" (n-bit strings) in the n-cube Q_n so that successive codewords differ in only one bit. [In other words, a Gray code for Q_n is a Hamiltonian circuit through the 2^n vertices of Q_n.] The fact that this can be done for any Q_n follows from the construction of the Binary Reflected Gray Code (BRGC) for Q_n (see [4]) 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 only a "small change" takes place from one object to the next in the list. The definition of "small change" depends on the particular family, its context, and its applications.

A Gray code for the permutation group S_n may be defined as a list of the permutations in S_n such that successive permutations differ by a transposition. One such famous Gray code for S_n was given by Johnson [5] and Trotter [8], apparently independently. In fact, their Gray code uses only adjacent interchanges. We discuss this Gray code in detail later.

Now consider any finite group G with a set of generators. A Gray code for G is usually defined to be a Hamiltonian circuit in the Cayley diagram of G, that is, a list of the objects of G such that each object is obtained from the previous one by applying one of the generators. We note that an algorithm which generates the objects of a combinatorial family is said to be loopless if it takes no more than a constant amount of time between successive objects. This notion of a loopless algorithm was first formulated by Gidean Ehrlich [3].

Conway, Sloane and Wilks (CSW) [2] proved the existence of a Gray code for all the finite reflection groups, which includes the three infinite families, denoted by A_n ($\cong S_{n+1}$), B_n, and D_n. A loopless algorithm implementing the Johnson–Trotter Gray code for $S_{n+1} \cong A_n$ was first given by Ehrlich [3] and, recently, a loopless algorithm generating a specific Gray code for B_n was given by Korsh, LaFollette and Lipschutz (KLL) [6].

This paper gives a loopless algorithm which generates a specific Gray code for D_n. This algorithm uses a loopless algorithm by Bitner, Ehrlich and Reingold (BER) [1] for the BRGC for Q_n, and our [7] 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 listing for S_n and our loopless algorithm for generating it. Section 4 discusses the KLL Gray code for B_n and their loopless algorithm implementing it. Section 5 discusses the reflection group D_n and our algorithm for its generation and Section 6 gives a loopless version of the algorithm.

2 Binary Reflected Gray Code

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

There are many Gray codes for Q_n. We are only interested in the Binary Reflected Gray Code (BRGC) for Q_n. Figure 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. The BRGC for Q_n is obtained recursively in this way beginning with the matrix $[0, 1]$ for $Q = Q_1$. [Note this construction gives a Hamiltonian circuit, not just a Hamiltonian path, for Q_n.]

Discussing the BRGC, Herbert Wilf [9] 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."

Observe that the transition sequence $T_4 = [4, 3, 4, \ldots, 3, 4]$ for the BRGC for Q_4 also appears in Figure 1 where we view the codewords from the bottom (fourth row) to the top (first 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 [5] gave a very clever loopless algorithm which gen-

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

Figure 1: Binary Reflected Gray Code for Q_4.

erates the transition sequence T_n using an array $[t_0, t_1, t_2, \ldots, t_n]$. Our loopless algorithm generating a Gray code for D_n essentially uses the BER algorithm to go both forward and backward in the BRGC for Q_n.

3 Johnson–Trotter list for S_n

The Johnson–Trotter permutation list for S_n is also defined recursively. The list for $n = 4$ is shown in Figure 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 in boldface and on top of the block. We note that the boldface labels form the Johnson–Trotter list for S_3.

In the first block, the largest item 4 sweeps from right to left, in the second block it sweeps 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. Thus the 4 does not interfere with any transposition involving 1, 2 and 3. Accordingly, the Johnson–Trotter list JT(n) is a Gray code for S_n. [In fact, successive permutations in 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 [7]. We have to move both forward and backward in the JT list, so our algorithm, although similar to the Ehrlich algorithm, will be more involved. We describe one part of our algorithm now.

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 Figure 2, changes its direction from LEFT to RIGHT after the fourth permutation 4123 where 4 has reached

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

Figure 2: Johnson–Trotter List for S_4.

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 one 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 will use arrays D and E, analogous to the arrays d and e, when moving backward in the JT list.

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 $1, 2, \ldots, n$ where each number has a sign, $+$ or $-$, attached to it. [We will let bold-faced underlined numbers indicate negative numbers.] Thus B_n has order $2^n \cdot n!$. In particular, B_2 has $|B_2| = 2^2 \cdot 2! = 8$ elements which follow:

$$B_2 = \{12, 1\underline{\mathbf{2}}, \underline{\mathbf{2}}1, \underline{\mathbf{21}}, \underline{\mathbf{12}}, \underline{\mathbf{1}}2, 2\underline{\mathbf{1}}, 21\}.$$

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 negates 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 $z = (p, g)$ where:

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

(ii) g (for Gray code) belongs to Q_n: g denotes the numbers in z which are negative. So, if the i-th entry of g is 1 then integer i in the permutation is signed.

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

Next we discuss the KLL [6] algorithm which outputs the objects of B_n as the pairs (p, g). We will illustrate their algorithm using B_4 as an example.

First of all, we assume the objects of B_n are arranged in a table whose columns are labeled by the BRGC for Q_n and whose rows are labeled by the Johnson–Trotter list for S_n Figure 3 pictures B_4 where the $|B_4| = 2^4 \cdot 4! = 384$ objects of B_4 are arranged in such a table.

In general, the first column H under $00\ldots0$ consists of all permutations where the signs are all $+$; hence it is the subgroup H of B_n which is isomorphic to S_n. Each of the other columns is a coset of H. Since the first column H is a Johnson–Trotter list for S_n, 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 only by using the generator R_n which negates the last item in a permutation. That is, to move from permutation p in a column to permutation q in an adjacent column, p must differ from q by a negation of its last item. [In Figure 3, we use a star $*$ to denote an edge between columns, and an underlined star, $\underline{*}$, to denote the first edge between the columns (which plays an important part in the Hamiltonian path for B_4).]

We note that there will always be an edge 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 up to and including (the last two) columns $2^n - 1$ and 2^n. These edges will be called *special edges*, and they will be in the Hamiltonian path for B_n.

The KLL Hamiltonian path for B_n will have two parts, A and B, where A moves forward through the table and down the last column, and B moves backward through the table and up the first column. Specifically:

(a) Part A begins at $x = 123\ldots n$ (in the first row, first column). Then (i) it will move from one column to the next column using a top special edge or the first edge between the columns, or (ii) it will move up or down within a column. It will arrive at the last column using a top special edge and then move all the way down the last column to the last entry $y = 2\underline{\mathbf{1}}3\ldots n$ in the column.

(b) Part B begins at the element y (in the last row, last column). Then (i) it will move from one column to the preceding column using either a bottom special edge or the second edge (just below the first edge) between the columns, or (ii) it will move up or down within a column. It will arrive at the first column using a bottom special edge and then move all the way up the first column to the second entry $123\dots n(n-1)$ in the column (the entry below x).

It would be instructive if the reader used our algorithm to follow the Gray code for B_4 in Figure 3. 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.

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.

5 Reflection roup D_n

The reflection group D_n is similar to B_n but now there is only an even number of negative numbers. Thus each element z in D_n may be represented by a pair $z = (p, g)$ where:

(i) p belongs to S_n

(ii) g belongs to Q_n but with an even number of 1's.

[We note that the codewords in Q_n with an even number of 1's (negative positions) can be listed by taking every other codeword in Q_n.] Observe that $|D_n| = 2^{n-1} \cdot n!$.

Again the generators $R_1, R_2, \dots, R_{n-1}$ of D_n correspond to the adjacent transpositions and hence generate a subgroup H of D_n which is isomorphic to S_n. The generator R_n in D_n interchanges and negates the last two coordinates. For example,

$R_n(\underline{\mathbf{3}}4\underline{\mathbf{18}}62\underline{\mathbf{7}}5) = \underline{\mathbf{3}}4\underline{\mathbf{18}}62\underline{\mathbf{5}}7$; $R_n(6\underline{\mathbf{1}}5\underline{\mathbf{43}}78\underline{\mathbf{2}}) = 6\underline{\mathbf{1}}5\underline{\mathbf{43}}72\underline{\mathbf{8}}$; $R_n(3\underline{\mathbf{41}}6\underline{\mathbf{25}}) = 3\underline{\mathbf{41}}652$.

The elements of D_n may also be listed in a table. Again the rows are labeled by the Johnson–Trotter list for S_n, but now the columns are labeled by the codewords with only an even number of 1's. Figure 4 shows the reflection group D_4 arranged in such a table. As noted above, the generator R_n interchanges and negates the last two coordinates. Accordingly, if x is an edge between an entry in column i and row j and an entry in column $i+1$, then x connects to the entry in row $j+1$ in column $i+1$. This will be a forward edge. Furthermore, there is another corresponding edge between the entry in column $i+1$ and row j, and the entry in column i and row $j+1$. This will be a backward edge. Figure 4 indicates the first such corresponding pairs of forward and backward edges between adjacent columns.

Our algorithm for a Gray code for D_n is a simple generalization of the algorithm for D_4. Thus we mainly describe our algorithm for D_4 using Figure 4. There will always be an edge from the first row to the second row between each odd column and the even column to its right, that is, between columns: 1 and 2, 3 and 4, 5 and 6, ..., $2^{n-1} - 1$ and 2^{n-1}.

	0000	0001	0011	0010	0110	0111	0101	0100	1100	1101	1111	1110	1010	1011	1001	1000
1234	1234 *	123**4**	12**34** *	12**3**4	1**23**4 *	1**234**	1**2**3**4** *	1**2**34	**12**34 *	**12**3**4**	**1234** *	**123**4	**1**2**3**4 *	**1**2**34**	**1**23**4** *	**1**234
1243	1243	12**4**3 *	12**43**	124**3**	1**2**4**3**	1**243** *	1**24**3	1**2**43	**12**43	**124**3 *	**1243**	**12**4**3**	**1**24**3**	**1**2**43** *	**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 *	41**32**	**4**1**32**	**4**13**2**	413**2**	4**1**3**2**	**41**3**2**	**4132**	4**132** *	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 *	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**	2**3**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

Figure 3: Reflection Group B_4.

Again, these edges will be called special edges, and they, along with their corresponding edges, will be in our Gray code for D_n.

Our Gray code for D_n will have two parts, A and B, where A moves forward through the table and B moves backward through the table. Also, our Gray code will only move down a column, but when we reach the last row in the column we can continue to the first row of the column since the Johnson–Trotter list is circular (a circuit). Specifically:

(a) A begins at $x = 1234$ (first row, first column) and follows the special edge

	0000	0011	0110	0101	1100	1111	1010	1001
1234	1234	12**34**	1**23**4	1**2**3**4**	**12**34	**1234**	**1**2**3**4	**1**23**4**
	\ /		\ /		\ /		\ /	
1243	1243 / \	12**43**	1**2**4**3** / \	1**24**3	**12**43 / \	**1243**	**1**24**3** / \	**1**2**4**3
1423	1423	1**4**2**3**	14**23**	1**42**3	**1**4**2**3	**1423**	**1**42**3**	**14**23
4123	4123	**4**12**3**	41**23**	**4**1**2**3	4**12**3	**4123**	4**1**2**3**	**41**23
4132	4132	**4**1**3**2	41**32**	**4**13**2**	4**1**3**2**	**4132**	4**13**2	**41**32
1432	1432	1**43**2	14**32**	1**4**3**2**	**1**43**2**	**1432**	**1**4**3**2	**14**32
1342	1342	1**34**2	1**3**4**2**	13**42**	**1**34**2**	**1342**	**13**42	**1**3**4**2
		\ /				\ /		
1324	1324	1**3**2**4** / \	1**32**4	13**24**	**1**3**2**4	**1324** / \	**13**24	**1**32**4**
3124	3124	**3**12**4**	**3**1**2**4	31**24**	3**12**4	**3124**	**31**24	3**1**2**4**
3142	3142	**3**1**4**2	**3**14**2**	31**42**	3**1**4**2**	**3142**	**31**42	3**14**2
3412	3412	**34**12	**3**41**2**	3**4**1**2**	34**12**	**3412**	**3**4**1**2	3**41**2
4312	4312	**43**12	4**3**1**2**	**4**31**2**	43**12**	**4312**	4**31**2	**4**3**1**2
4321	4321	**43**21	4**32**1	**4**3**2**1	43**21**	**4321**	4**3**2**1**	**4**32**1**
3421	3421	**34**21	**3**4**2**1	3**42**1	34**21**	**3421**	**3**42**1**	3**4**2**1**
3241	3241	**3**2**4**1	**32**41	3**24**1	3**2**4**1**	**3241**	**3**24**1**	32**41**
				\ /				
3214	3214	**3**21**4**	**32**14	3**2**1**4** / \	3**21**4	**3214**	**3**2**1**4	32**14**
2314	2314	2**3**1**4**	**23**14	**2**31**4**	**2**3**1**4	**2314**	2**31**4	23**14**
2341	2341	2**34**1	**23**41	**2**3**4**1	**2**34**1**	**2341**	2**3**4**1**	23**41**
2431	2431	2**43**1	**2**4**3**1	**24**31	**2**43**1**	**2431**	24**31**	2**4**3**1**
4231	4231	**4**2**3**1	4**23**1	**42**31	4**2**3**1**	**4231**	42**31**	**4**23**1**
4213	4213	**4**21**3**	4**2**1**3**	**42**13	4**21**3	**4213**	42**13**	**4**2**1**3
2413	2413	2**4**1**3**	**2**41**3**	**24**13	**2**4**1**3	**2413**	24**13**	2**41**3
2143	2143	21**43**	**2**14**3**	**2**1**4**3	**21**43	**2143**	2**1**4**3**	2**14**3
2134	2134	21**34**	**2**1**3**4	**2**13**4**	**21**34	**2134**	2**13**4	2**1**3**4**

Figure 4: Reflection Group D_4.

to 12**43**. Then A moves down column two to 1**34**2 where there is the first edge crossing to column three. Then A follows the edge to 1**32**4, continues all the way down column three to the last row, and then up to the fist row in the column to 1**23**4.

Next A follows the special edge to 1$\underline{\mathbf{24}}$3. Then A moves down column four to 3$\underline{\mathbf{24}}$1, where it follows the first edge crossing to column five to 3$\underline{\mathbf{21}}$4. Now A moves all the way down column five to the last row and then up to the fist row in the column to $\underline{\mathbf{12}}$34.

And so on, until A moves all the way down the last column and then up to its first row to $\underline{\mathbf{1}}$23$\underline{\mathbf{4}}$. Here A ends.

(b) B begins at $\underline{\mathbf{1}}$23$\underline{\mathbf{4}}$ (first row, last column) and follows the special edge backward to $\underline{\mathbf{1}}$24$\underline{\mathbf{3}}$. Then B moves down column seven to $\underline{\mathbf{13}}$42, where there is the first edge crossing backward to column six. B follows the edge to $\underline{\mathbf{1324}}$. Now B moves all the way down column six to the last row and then up to the fist row in the column to $\underline{\mathbf{1234}}$.

Next B follows the special edge backward to $\underline{\mathbf{12}}$43. Then B moves down column five to 3$\underline{\mathbf{2}}$4$\underline{\mathbf{1}}$, where there is the first edge crossing backward to column four. B follows the edge to 3$\underline{\mathbf{2}}$1$\underline{\mathbf{4}}$. Now B moves all the way down column four to the last row and then up to the fist row in the column to 1$\underline{\mathbf{2}}$3$\underline{\mathbf{4}}$.

And so on, until B moves all the way down the first column to 2134. The Gray code is complete.

Observe that here, contrary to the Gray code for B_n, we are always moving down a column, never up, except when we go directly from the last row to the first row in a column.

6 Loopless algorithm for D_n

The loopless algorithm of KLL [6] generating the Gray code for B_n uses two functions, *nextperm* and *nextgray*. *Nextperm* could start at an arbitrary p in a column and looplessly generate successive permutations in the column by going either up or down the column. *Nextgray* looplessly generated the next codeword g when going forward to the next column or backward to the previous column and returned the entry in g that will change from g to g's successor. Here we can use a simplified version of *nextperm* since we only go down in a column except when going from the last row to the first row in a column.

Implementing our algorithm for D_n 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 *nextgray*. (2) Looplessly move down a column or, equivalently, looplessly move between permutations of the symmetric group S_n. This we do using a simplified *nextperm*. This version uses only the arrays d and e described above. (3) Determine, in constant time, whether or not we are at a crossover row. This can be done by storing the last two items, the $(n-1)$-th and the n-th, of the crossover permutation corresponding to the two positions of g that change when moving from one column to the next. For example, in Figure 4 when crossing over from 2 to column 3 it is positions 2 and 4 of g that change (g goes from 0011 to 0110). In fact position n will always change.

Let $N1[s]$ and $N2[s]$, respectively, be the $(n-1)$-th and the n-th items of the crossover permutation when positions s and n of g change. Then,

if n is even, $N1[s] = n$ and $N2[s] = s$ for $1 \le s \le n-2$ else $N1[s] = s$ and $N2[s] = n$ for $1 \le s \le n-3$ and $N1[n-2] = n$ and $N2[n-2] = n-2$.

For crossovers from row 1, s is always $n-1$, and these will be dealt with as special cases. So, when crossing from column 2 to column 3, s is 2 and the crossover permutation is 1342 whose last two items are $N1[2]$ and $N2[2]$, namely, 4 and 2.

Consequently, we can use arrays $N1$ and $N2$ of length n to see if we are at a crossover permutation for any row but row 1.

Our loopless algorithm, written in C++, appears below. We assume $n \ge 4$.

```
//This program looplesly generates the Dn reflections for n>4.
#include <iostream.h>
int n, i, j, I1, I2, p[20], pl[20], d[21],
e[21],  done, LEFT=1, num=0, s, g[20], t[21], last,  N1[20], N2[20];

int nextgray() {s=t[n+1]; g[s]=!g[s]; t[n+1]=n; t[s+1]=t[s];
t[s]=s-1; return t[n+1];}

void nextperm(int e[], int d[]) {
   e[n+1]=n;
   if (d[j]==LEFT)
   {
      I1=pl[j]; I2=pl[j]-1; i=p[I2]; p[I1]=i; p[I2]=j;
      pl[i]=I1; pl[j]=I2; done=((I2==1)||(p[I2-1]>j));
   }
   else
   {
      I1=pl[j]; I2=pl[j]+1; i=p[I2]; p[I1]=i; p[I2]=j;
      pl[i]=I1; pl[j]=I2; done=((I2==n)||(p[I2+1]>j));
   }
   if (done) {d[j]=!d[j]; e[j+1]=e[j]; e[j]=j-1;}
}

void visit() {
   num++; cout<<"num "<<num<<endl;
   for (i=1; i<=n; i++) cout<<g[i]<<" "; cout<<endl;
   for (i=1; i<=n; i++) cout<<p[i]<<" "; cout<<endl;;
   if (num%50==0) cin>>i;
}

int main(void) {
   //Initialize
   cout<<"Enter n"<<endl; cin>>n;
   for (i=1; i<=n; i++) p[i]=pl[i]=i;
   for (i=1; i<=n; i++) {d[i]=1; e[i]=i-1; g[i]=0; t[i]=i-1;} t[n+1]=n;
```

```
if (n%2==0) {for (s=1; s<=n-2; s++) {N1[s]=n; N2[s]=s;}}
else
{for (s=1; s<=n-3; s++) {N1[s]=s; N2[s]=n;} N1[n-2]=n; N2[n-2]=n-2;}

//Generate the first item of the first column
//and the second item of the next column
visit(); j=n; nextperm(e, d); last=j; j=e[n+1];
nextgray(); nextgray(); visit(); s=t[t[n+1]];

//Generate items to be listed as the algorithm moves to the
//right up to and including the second item of the last column
while (s>0)
{
   //Generate the rest of the second column
   //down to and including the crossover item
   while ((p[n-1]!=N1[s])||(p[n]!=N2[s]))
   {nextperm(e, d); last=j; j=e[n+1]; visit();}

   //crossover to the next column
   nextgray(); nextgray();

   //Generate the rest of the next column
   while (j>1) {nextperm(e, d); j=e[n+1]; visit();}

   //Generate the first item of that column
   //and the second item of the next column
   p[1]=1; p[2]=2; pl[1]=1; pl[2]=2; d[2]=1; e[n+1]=n;
   visit(); j=n; nextperm(e, d); last=j; j=e[n+1];
   nextgray(); nextgray(); visit(); s=t[t[n+1]];
 }

 //Generate the rest of the last column
 while (j>1) {nextperm(e, d); j=e[n+1]; visit();}

 //Generate the first item of the last column
 //and the second item of the previous column
 p[1]=1; p[2]=2; pl[1]=1; pl[2]=2; d[2]=1; e[n+1]=n;
 visit(); j=n; nextperm(e, d); last=j; j=e[n+1];
 nextgray(0; nextgray(); nextgray(); nextgray(); s=nextgray(); visit();

 //Generate items to be listed as the algorithm moves to the
 //left up to and including the second item of the first column
 while (s>0)
 {
    //Generate the rest of the previous column
    //down to and including the crossover item
    while ((p[n-1]!=N1[s])||(p[n]!=N2[s]))
    {nextperm(e, d); last=j; j=e[n+1]; visit();}

    //crossover to the previous column
```

```
        nextgray(); nextgray();

        //Generate the rest of the previous column
        while (j>1) {nextperm(e, d); j=e[n+1]; visit();}

        //Generate the first item of that column
        //and the second item of the previous column
        p[1]=1; p[2]=2; pl[1]=1; pl[2]=2; d[2]=1; e[n+1]=n;
        visit(); j=n; nextperm(e, d); last=j; j=e[n+1];
        nextgray(); s=nextgray(); visit();
    }

    //Generate the rest of the first column
    while (j>1) {nextperm(e, d); j=e[n+1]; visit();}
    cout<<"num is "<<num<<endl;
}
```

References

[1] J.R. Bitner, G. Ehrlich and E.M. Reingold, *Efficient generation of the binary reflected Gray code and its applications*, Comm. ACM, **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*, ACM, **20** (1973), 500–513.

[4] E.N. Gilbert, *Gray codes and paths on the n-cube*, Bell Syst. Tech. J., **37** (1958), 815–826.

[5] S.M. Johnson, *Generation of permutations by adjacent transposition*, Math. Comput., **17** (1963), 282–285.

[6] J. Korsh, P. LaFollette and S. Lipschutz, *Gray code and loopless algorithm for the reflection group B_n*, Submitted for review.

[7] J. Korsh and S. Lipschutz, *Generating multiset permutations in constant time*, J. Algorithms, **25** (1997), 321–335.

[8] H.F. Trotter, *Algorithm 115, Permutations*, Comm. ACM, **5** (1962), 434–435.

[9] H. Wilf, *Combinatorial algorithms - an update*, SIAM, Philadelphia, 1989.