Check for updates

REMARK ON ALGORITHM 246

Graycode[Z] [J. Boothroyd, *Commun. ACM 7*, 12 (Dec. 1964), 701]

M. C. Er [Received October 1984; revised March 1985; accepted October 1985]
Department of Computer Science, University of Western Australia, Nedlands, WA 6009, Australia

After the publication of Algorithm 246, Misra [1] suggested an improvement to it. He gave no detailed coding; but from his descriptions, we obtain the following algorithm for generating the binary Gray code of N bits, assuming that $A[1 . . N]$ is an array of bits, which is initialized to all 0's.

```
procedure Misra1(N: integer);
var s, j, i: integer;
begin
  s := 1;
  j := 0;
  PrintCode;
  while j <> 1 do begin
    if odd(s) then begin
      if A[N] = 1 then begin
        j := pop;
        j := pop;
        push(j);
        end
      else begin {A[N] = 0}
        push(N);
        j := N;
        end;
      A[N] := 1 - A[N];
      end
    else begin {even(s)}
      j := pop;
      if A[j - 1] = 1 then
        i := pop
      else {A[j - 1] = 0}
        push(j - 1);
      push(j);
      A[j - 1] := 1 - A[j - 1];
      end;
    PrintCode;
    s := s + 1;
    end;
end {Misra 1};
```

Here, the procedure PrintCode simply prints the contents of $A[1 . . N]$. The invariant of the loop is

R1: (j = MAX(i: $1 \le i \le N$: $A[i] = 1$) \/ $j = 0$) /\
the stack contains the positions of all 1 bits in sorted order.

We see that the position of the rightmost 1 bit is frequently pushed onto and popped off from the stack. The efficiency of the algorithm can be improved by

not storing the position of the rightmost 1 bit in the stack but remembering it in j. Hence we obtain the following improved algorithm

```
procedure Misra2(N: integer);
var s, j, i: integer;
begin
  s := 1;
  j := 0;
  PrintCode;
  while j <> 1 do begin
    if odd(s) then begin
      if A[N] = 1 then
        j := pop
      else begin {A[N] = 0}
        push(j);
        j := N;
        end;
      A[N] := 1 - A[N];
      end
    else begin {even(s)}
      if A[j - 1] = 1 then
        i := pop
      else {A[j - 1] = 0}
        push(j - 1);
      A[j - 1] := 1 - A[j - 1];
      end;
    PrintCode;
    s := s + 1;
    end;
end {Misra2};
```

The invariant of the loop is now:

R2: (j = MAX(i:1 ≤ i ≤ N:$A[i]$ = 1) \/ j = 0) /\
the stack contains the positions of all 1 bits, except the rightmost 1 bit, and position 0 in sorted order.

The position 0 stored in the stack is simply a sentinel value, and is not referred to at all.

The procedure Misra2 is more efficient than the procedure Misra1, as it performs only a push or a pop per iteration. Despite the improvement, the overhead for maintaining the stack is still too high.

If recursion is allowed, a recursive algorithm can generate the binary Gray code more efficiently than the above iterative algorithms. Let $A[1..N]$ be an array of bits, initially containing all zeros. The following recursive procedure, when called as GrayCode1(1), will generate the binary Gray code of N bits.

```
procedure GrayCode1(n:natural);
begin
  if n <= N then begin
    GrayCode1(n + 1);
    A[n] := 1 - A[n];
    GrayCode1(n + 1);
    end
  else PrintCode;
end {GrayCode1};
```

Table I. The running times (in seconds) of the algorithms Misra1, Misra2, GrayCode1 and GrayCode2 on a Vax 11/750

N	Misra1	Misra2	GrayCode1	GrayCode2
6	0.80	0.70	0.67	0.60
7	1.73	1.50	1.43	1.37
8	3.63	3.30	3.07	3.00
9	7.83	7.20	6.87	6.70
10	17.13	15.60	14.97	14.17
11	36.40	34.17	32.10	31.10
12	77.17	70.97	68.03	67.23
13	164.27	154.27	147.33	143.80

Three comments are in order. First, the procedure GrayCode1 is considerably shorter than the procedures Misra1 and Misra2. Second, the procedure GrayCode1 manipulates no auxiliary data structures at all. Third, the total number of procedure calls to GrayCode1 to generate $2^N - 1$ codewords is $2^{N+1} - 1$; hence each codeword is generated in $O(2)$ units of time, on average. The average-time complexity can be improved to $O(1)$ by removing redundant procedure calls as shown below:

```
procedure GrayCode2(i:natural);
begin
  if i < N then begin
    GrayCode2(i + 1);
    A[i] := 1 - A[i];
    PrintCode;
    GrayCode2(i + 1);
    end
  else begin {i = N}
    A[i] := 1 - A[i];
    PrintCode;
    end;
end {GrayCode2};
```

The efficiencies of these four procedures implemented in PASCAL have been timed on a Vax 11/750 computer running under UNIX, and are summarized in Table I. The actual running times (measured in seconds of CPU time) support the above observations.

Thus we conclude that GrayCode2 is the most efficient algorithm to generate the binary Gray Code, whereas Misra1 is the least efficient one among these four algorithms.

REFERENCE

1. MISRA, J. Remark on Algorithm 246: Graycode[Z]. *ACM Trans. Math. Softw. 1*, 3 (Sept. 1975), 285.