TestVector

Outline

We will proceed as follows. First, we will write a program to test if a given test *vector* can be produced by a seed vector combination, according to the function given. Then, we shall write a program that checks the compatibility of test *cubes* with test *vectors*. Finally, we may write a program that reads a sequence of test cubes and checks if it matches with one or more test vectors.

I.**Testing if a given test *vector* can be produced from some *a*, *b*.**

The title of this section is somewhat misleading- in addition to checking if a certain test vector can be compressed directly, we will also produce the *a* and *b* which decompress to the given vector. To understand the general plan of action, we will first take note of some theory.
The partial products of the multipliers correspond to the given equations: (taken from the code in MATLAB)

```
t11= v11;
t12= v12;
t13= v13;
t14= v14;
t21= xor(v12,v21);
t22= xor(v12,v23);
t23= xor(v13,v24);
t24= v24;
t31= xor(v12,xor(v23,v31));
t32= xor(v13,xor(v24,v32));
t33= xor(v23,v34);
t34= v34;
t41= xor(xor(v14,v22),xor(v33,v41));
t42= xor(v23,xor(v34,v42));
t43= xor(v34,v43);
t44= v44;
```

where the variables v_{ij} are obtained from a, b as:

$v_{ij} := a_i \wedge b_j$

with the indices on a and b signifying the *bit number* under current consideration. The equivalence of the *XOR* operation and *multiplication modulo 2* means that all the operations involved in the evaluation of t_{ij} from a,b can be regarded as operations in a *finite field*, more specifically, the field F_2. This means that we may ignore, for now, the operation by which we obtain v_{ij} from a and b, and treat the mapping of the matrix $[v_{ij}]$ to $[t_{ij}]$ as that of a linear operator acting on ${F_2}^{16}$, in which we replace $[v_{ij}]$ and $[t_{ij}]$ by the vectors V and T respectively, with the definition of their components taken in the *row-major* convention:

$t_{ij} = T_{4i + j - 4}$

and similarly for v and V. In order to test, now, whether a given test vector can be compressed directly, we must first analyse the *form* of this linear operator.
Writing it out in matrix form we get:

$$\begin{matrix}
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&1&0&1&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&1&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&1&0&1&0&0&0&0&1&0&1&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&1&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1
\end{matrix}$$

We'll call this matrix A. The rest of the analysis is very simple and apparent immediately:
1) Check if the given 16-bit data (T) is in the column space of A.
2) Find the V which corresponds to T, if the first step allows for it.
3) Find a and b from V according to the multiplier equations.

To perform 1), we analyse A a little. While performing computation in F_2, the operator is non-invertible if it has an *odd* determinant. The above matrix has determinant 0. It can be seen that its rank is 14- by simply *looking* at each column from the rightmost (i.e. column #16), we find that the linear independence of the columns is broken by the fact that: $C_6 = C_{11} = C_{13}$. That is why, on elimination of these columns, row reduction gives us a very simple formula for the partial inversion of the operation. The formulae we get by reducing the block matrix $[\ A \mid T\ ]$, is as below.

$V_1 = T_1$
$V_2 = T_2$
$V_3 = T_3$
$V_4 = T_4$
$V_5 = T_5 - T_2$
$V_6 = T_6 - T_2$
$V_7 = T_7 - T_3$
$V_8 = T_9 - T_6$
$V_9 = T_{10} - T_7$
$V_{10} = T_{11} - T_6 + T_2$
$V_{11} = T_{13} - T_4$

$V_{12} = T_{14} - T_{11}$
$V_{13} = T_{15} - T_{11} + T_6 - T_2$
$V_{14} = T_{16}$
$V_{15} = T_8 - T_7 + T_3$
$V_{16} = T_{12} - T_{11} + T_6 - T_2$

Further, row reduction on the 14-column version of A gives the last two rows zero, as expected. This means that a given T *cannot* lie in the column space of A if the last two expressions above (for V_{15} and V_{16}) evaluate to zero. This gives us step number one in the operation to get V from T:

```
⟨ 1. Check if the given $T \in$ column-space of $A$. ⟩:-
if(
    ( ( $T$[8] - $T$[7] + $T$[3] ) mod 2 $\neq$ 0 )
                    $\vee$
    ( ( $T$[12] - $T$[11] + $T$[6] - $T$[2] ) mod 2 $\neq$ 0 )
  )
{
    printf( "\nCannot be compressed normally- not in column space." );
    goto end;
}
```

The above equations also allow us to complete step (2) of our analysis easily.

```
⟨ 2. Find $V$ from $T$. ⟩:-
$V$[1] $\leftarrow$ ( $T$[1] ) mod 2;
$V$[2] $\leftarrow$ ( $T$[2] ) mod 2;
$V$[3] $\leftarrow$ ( $T$[3] ) mod 2;
$V$[4] $\leftarrow$ ( $T$[4] ) mod 2;
$V$[5] $\leftarrow$ ( $T$[5] - $T$[2] ) mod 2;
$V$[6] $\leftarrow$ ( $T$[6] - $T$[2] ) mod 2;
$V$[7] $\leftarrow$ ( $T$[7] - $T$[3] ) mod 2;
$V$[8] $\leftarrow$ ( $T$[9] - $T$[6] ) mod 2;
$V$[9] $\leftarrow$ ( $T$[10] - $T$[7] ) mod 2;
$V$[10] $\leftarrow$ ( $T$[11] - $T$[6] + $T$[2] ) mod 2;
$V$[11] $\leftarrow$ ( $T$[13] - $T$[4] ) mod 2;
$V$[12] $\leftarrow$ ( $T$[14] - $T$[11] ) mod 2;
$V$[13] $\leftarrow$ ( $T$[15] - $T$[11] + $T$[6] - $T$[2] ) mod 2;
$V$[14] $\leftarrow$ ( $T$[16] ) mod 2;
$V$[15] $\leftarrow$ ( $T$[8] - $T$[7] + $T$[3] ) mod 2;
$V$[16] $\leftarrow$ ( $T$[12] - $T$[11] + $T$[6] - $T$[2] ) mod 2;
```

Step (3) is also simple, and hinges on one basic insight. Let us look at the rectangular array representation of v_{ij}, and use its definition in therms of a and b.

$$\begin{array}{cccc} a_1b_1 & a_1b_2 & a_1b_3 & a_1b_4 \\ a_2b_1 & a_2b_2 & a_2b_3 & a_2b_4 \\ a_3b_1 & a_3b_2 & a_3b_3 & a_3b_4 \\ a_4b_1 & a_4b_2 & a_4b_3 & a_4b_4 \end{array}$$

It is clear that if any entry is zero, one of three cases holds:
1) Every other entry in its row is zero
2) Every other entry in its column is zero
3) Both 1) and 2) holds
That is because of the repitition of one of the (bit-) multiplicands along each row and column. Given the interpretation of V as being the row-major reading of v, we can implement this easily. We simply have to search for a zero, in V, then check if every entry along either its row or column is also zero. If not, the given vector *cannot* be compressed. If it can, then identifying that row, or column will immediately identify a bit as being zero. If all the a_i's and b_i's were intitialised to '1', repeating this will identify *all* the bits of the multiplicand.
Here, we shall:
1) Get v from V.
2) Search for a zero in the result.
3) Check if the entire row vanishes.
4) Check if the entire column vanishes.
5) Error handling- if *neither* row nor column vanishes.

```
⟨ 3. Find $a$, $b$ from $V$. ⟩:-
for( $i \leftarrow 1$; $i \leq 16$; $i \leftarrow i + 1$ )
{
    $I \leftarrow$ (int)$i/4 + 1$;
    $J \leftarrow$ ( $i$ mod 4 ) $+ 1$;
    $v[I][J] \leftarrow V[i]$;
}

for( $I \leftarrow 1$; $I \leq 4$; $I \leftarrow I + 1$ )
{
    for( $J \leftarrow 1$; $J \leq 4$; $J \leftarrow J + 1$ )
    {

        if( $v[I][J] = 0$ )
        {

            for( $k \leftarrow 1$; $k \leq 4$; $k \leftarrow k + 1$ )
                $row\_sum \leftarrow row\_sum + v[I][k]$;
            if( $row\_sum = 0$ ) $a[I] \leftarrow 0$;

            for( $k \leftarrow 1$; $k \leq 4$; $k \leftarrow k + 1$ )
                $column\_sum \leftarrow column\_sum + v[k][J]$;
            if( $column\_sum = 0$ ) $b[J] \leftarrow 0$;
            if( ( $row\_sum \neq 0$ ) $\wedge$ ( $column\_sum \neq 0$ ) )
            {
               printf( "\nImage not compatible with an \"a\", \"b\"." );
               goto end;
            }
         }

      }
   }

   for( $k \leftarrow 1$; $k \leq 4$; $k \leftarrow k + 1$ )
      printf( "\na%d= %d, b%d= %d", k, a [k], k, b [k] );
```

Finally, there is a lot of work in the initialisation. Immediately, we note that we must:
1) The *seed vectors* (i.e. a, b) must be initialised to 1111,1111.
2) Resolve the symbol "mod" according to the language ("%" in C.)
3) Read T from the command line arguments.

```
   ⟨ Initialise. ⟩:-
   int $I \leftarrow 1$, $J \leftarrow 1$, $i \leftarrow 1$, $k \leftarrow 1$;
   int $row\_sum \leftarrow 0$, $column\_sum \leftarrow 0$;
   int V [16], T [16], v [4][4];
   int a [5] $\leftarrow$ { 1, 1, 1, 1, 1}, b [5] $\leftarrow$ { 1, 1, 1, 1, 1};

   for( $i \leftarrow 1$; $i \leq 16$; $i \leftarrow i + 1$ )
   { T [ i ] $\leftarrow$ argv [1][ i - 1 ] - '0' ; }
```

In C, this requires further resolution to head it off,

```
   ⟨ Begin. ⟩:-
   #include <stdio.h>

   int compress( char *argv [] )
   {
```

and, at the end,

```
   ⟨ Finish. ⟩:-
   end:

      return 0;
   }
```

That takes care of determining a, b from T. The procedure is, therefore, as:

$\langle$ Begin. $\rangle$

$\langle$ Initialise. $\rangle$
$\langle$ 1. Check if the given $T \in$ column-space of A. $\rangle$
$\langle$ 2. Find V from T. $\rangle$
$\langle$ 3. Find a, b from V. $\rangle$
$\langle$ Finish. $\rangle$