Binary vectors

We denote by $\mathbb{F}_2 = \{0, 1\}$ the field with two elements. The (commutative) operations in this field are as follows ($\forall x \in \mathbb{F}_2$):

- $0 * x = 0$
- $1 * 1 = 1$
- $x + x = 0$ (in particular $1 + 1 = 0$)
- $0 + x = x$

For any positive integer $m \in \mathbb{N}$, $\mathbb{F}_2^m$ is a vector-space of dimension m over $\mathbb{F}_2$. Let $u = (u_0, u_1, \ldots, u_{m-1}) \in \mathbb{F}_2^m$ be a binary vector. For $0 \leq i < m$, u_i is the i-th coordinate of u, which can be seen as an array of bits (or binary string). The vector u can be uniquely identified to a positive integer less than 2^m via the binary representation of integers:

$$u = (u_0, u_1, \ldots, u_{m-1}) \longleftrightarrow u_0 \times 2^0 + u_1 \times 2^1 + \cdots + u_{m-1} \times 2^{m-1}$$

So we can use native integers provided by programming languages to implement binary vectors. In C or C++, we use in general the type `unsigned long`, which can represent binary vectors of dimension typically equal to 32 or 64 (if we need greater dimensions, we can use arrays of ints, or use a big-int library, like gmp). The operation on integers corresponding to addition in $\mathbb{F}_2^m$ (which consists in the addition in $\mathbb{F}_2$ coordinate by coordinate, i.e. $u + v = (u_0 + v_0, \ldots, u_{m-1} + v_{m-1})$) is called the *xor* operation (and is mapped to a CPU instruction in C for native integer types). In C and Python, it is denoted by the `^` operator:

```
unsigned a=3, b=5;
// a and b bits are (1,1,0,0,...) and (1,0,1,0,...)
assert(a^b == 6); // a^b bits are (0,1,1,0,...)
```

Two other operators in these languages are very useful : the mask (&) and right-shift (>>) operators. These operations are defined as follows, for $u, v \in \mathbb{F}_2^m$ and $i \in \mathbb{N}$:

- $u\&v = (u_0 * v_0, \ldots, u_{m-1} * v_{m-1})$
- $u >> i = (u_i, u_{i+1}, \ldots, u_{m-1}, 0, \ldots)$

For example, to get the first bit (u_0) of an integer u, we can write $u\&1$. Similarly, $(u >> i)\&1$ gives the i-th bit (u_i) of u.

Question 1. *Write a function taking as inputs an integer u (viewed as a binary vector) and an integer i, which returns the i-th bit of u.*

Question 2. *Write a function returning the indice of the first non-zero bit of a binary vector.*

The *Hamming weight* of a binary vector u (of dimension m), denoted by $\mathrm{wt}(u)$, is its number of coordinates equal to 1 :

$$\mathrm{wt}(u) = |\{i \in [0, \ldots, m-1] \ : \ u_i = 1\}|$$

Remark : the coordinates u_i for $i \geq m$ are implicitly equal to 0. So $\mathrm{wt}(u) = \sum_{i \in \mathbb{N}} u_i$, the sum being done in $\mathbb{N}$ (not in $\mathbb{F}_2$).

Question 3. *Write a function that implements the Hamming weight of a binary vector (given as an integer).*

Boolean functions

See `http://www.math.univ-paris13.fr/~carlet/chap-fcts-Bool-corr.pdf` for in-depth study.

A *Boolean function* in m variables is a mapping from $\mathbb{F}_2^m$ into $\mathbb{F}_2$. For example, for $m = 2$, the following mapping defines a Boolean function f :

$$(0,0) \to 0$$
$$(1,0) \to 1$$
$$(0,1) \to 0$$
$$(1,1) \to 0$$

Via the one-to-one correspondance between $\mathbb{F}_2^2$ and the set $\{0,1,2,3\}$, we can also define f by writing $f(0) = 0, f(1) = 1, f(2) = 0, f(3) = 0$. The most straightforwad way to represent a Boolean function f is by means of its *truthtable*, i.e. the list of all of its values:

$$f = (f(0), f(1), \ldots, f(2^m - 1)) \in \mathbb{F}_2^{2^m}$$

For the example above, the truthtable of f is $f = (0,1,0,0)$. Note that the truthtable of f is a binary vector of dimension 2^m. The Hamming weight of f is the Hamming weight of its truthtable. Except for small values of m, 2^m will be greater than 64, and we won't be able to represent it via the type `unsigned long`. However, it is very easy to represent it via an array of ints

Here is a sample code in C and in Python:

```
// C code
int table_f[4];
table_f[0] = 0;
table_f[1] = 1;
table_f[2] = 0;
table_f[3] = 0;

# Python code
table_f = [0,1,0,0]
print ("The value of f at position u=(1,0) is %d\n" % table_f[1])
```

Question 4. *Write a function taking as input a truthtable (as an array of ints), and returning its Hamming weight.*

The *Hamming distance* between two Boolean functions is the Hamming weight of their sum (note that the truthtable of the sum of two Boolean functions is the sum of the truthtables of these functions). In other words, it is equal to the cardinal of the set

$$\{u \in \mathbb{F}_2^m \mid f_1(u) \neq f_2(u)\}$$

Question 5. *Implement a function* `hamming_distance(f1,f2)` *returning an int (the prototype in C would/could be* `int hamming_distance(int f1[], int f2[], int m)`

Question 6. *A function in m variables is said to be balanced if it takes values* 0 *and* 1 *the same number of times. What is the Hamming weight of such a Boolean function? Implement a function* `is_balanced(f)` *which checks the balancedness of a Boolean function f.*

Walsh transform

The Walsh transform $\hat{f}$ of a Boolean function f is the mapping which associates to a vector $a \in \mathbb{F}_2^m$ the quantity

$$\hat{f}(a) = \sum_{x \in \mathbb{F}_2^m} (-1)^{f(x)+a \cdot x}$$

where $a \cdot x$ is the scalar product between a and x :

$$a \cdot x = a_0 x_0 + a_1 x_1 + \cdots + a_{m-1} x_{m-1}$$

the sum being done in $\mathbb{F}_2$.

Question 7. *Write a function which computes the scalar product between two vectors of $\mathbb{F}_2^m$ given as integers.*

Question 8. *For $e \in \mathbb{F}_2$, $(-1)^e$ is the* sign *function, with $(-1)^0 = 1$ and $(-1)^1 = -1$. Implement this function (*`int sign(int e)` *in C).*

Question 9. *Write a function which computes the Walsh transform $\hat{f}(a)$ of a Boolean function f (given as a truthtable) at point $a \in \mathbb{F}_2^m$. Write a function which computes $\hat{f}(a)$ for all $a \in \mathbb{F}_2^m$ (it can return an array of size 2^m containing all the values). What is its complexity?*

Fast Walsh transform

There is an algorithm which computes the Walsh transform (all the values of $\hat{f}$) more efficiently than the obvious way. Given an array T of ints containing the values of f of length 2^m (the value $f(u)$ at position u in T is denoted $T[u]$) :

- replace all of its values by their sign.
- apply the following recursive algorithm :
  - if $m > 0$, split the table into two halves. The left (resp. right) part is replaced by the sum (resp. difference) of the two initial parts : for each $0 \leq u < 2^{m-1}$, the new value of $T[u]$ (resp. $T[u+2^{m-1}]$) will be $T[u]+T[u+2^{m-1}]$ (resp. $T[u]-T[u+2^{m-1}]$). Call recursively this algorithm onto each halves (which are of length 2^{m-1}).
  - if $m = 0$, the algorithm ends.

For example, here are the different steps for the function $f = (0,0,1,0,1,1,0,0)$:

m	0	0	1	0	1	1	0	0
3	1	1	−1	1	−1	−1	1	1
2	0	0	0	2	2	2	−2	0
1	0	2	0	−2	0	2	4	2
0	2	−2	−2	2	2	−2	6	2

Question 10. *Apply this algorithm by hand on the following truthtable :* $(1,0,1,1,0,0,0,0)$.

Question 11. *Write a function which implements this algorithm.*

Question 12. *Try to understand why this algorithm works. What is its complexity?*

Question 13. *The* nonlinearity *of a Boolean function f is equal to $\max_{a \in \mathbb{F}_2^m} |\hat{f}(a)|$. Implement a function which computes the nonlinearity.*

Algebraic Normal Form (ANF)

A Boolean function can be uniquely represented as a polynomial in $\mathbb{F}_2[x_0, \ldots, x_{m-1}]$, called its ANF. To be more exact, the representation is unique as long as the relative degree in each variable is at most 1 (see below).

Let's first define coordinate functions, denoted (temporarily) by X_i :

$$\begin{array}{rcl} X_i : & \mathbb{F}_2^m & \longrightarrow \mathbb{F}_2 \\ & (x_0, \ldots, x_i, \ldots, x_{m-1}) & \longmapsto x_i \end{array}$$

In this notation, x_i represents a bit in $\mathbb{F}_2$, while X_i is a Boolean function. Boolean functions can be multiplied : the set $\mathbb{F}_2^{(\mathbb{F}_2^m)}$ of all Boolean functions in m variables is a group for the multiplication, which is obviously defined as $(f * g)(x) = f(x) * g(x)$. So we can multiply coordinate functions, e.g. $X_1 * X_3 * X_4$. According to the laws of multiplication in $\mathbb{F}_2$, for each i, we have $X_i * X_i = X_i$ (check it). We generally omit the $*$ operator, and write X_i in lower case, so that the function $X_1 * X_3 * X_4$ will be written as $x_1x_3x_4$, which can be also be seen as a formal polynomial. A boolean function which is the product of coordinate functions is called a monomial. The sum of monomials (which is again a Boolean function, the addition being defined the same way as the multiplication, i.e. $(f+g)(x) = f(x)+g(x)$) is called a polynomial. The representation of a Boolean function as a polynomial is called its ANF : each Boolean function admits exactly one ANF representation (the proof is not too difficult). Here is an example of how to compute the truthtable of a polynomial, $x_0x_1 + x_1$:

(x_0, x_1)	x_0x_1	$x_0x_1 + x_1$
$(0,0)$	0	0
$(1,0)$	0	0
$(0,1)$	0	1
$(1,1)$	1	0

A monomial M can be written in the following way :

$$M(x_0, \ldots, x_{m-1}) = x_0^{u_0} x_1^{u_1} \cdots x_{m-1}^{u_{m-1}},$$

where u_i is in $\mathbb{F}_2$, and $x_i^{u_i} = x_i$ if $u_i = 1$ and $x_i^{u_i} = 1$ if $u_i = 0$. So M is uniquely determined by the coefficients $u_0, \ldots, u_{m-1}$, and hence by the integer $u = u_0 + 2u_1 + 2^2u_2 + \cdots + 2^{m-1}u_{m-1}$. For example, the monomial $x_1x_3x_4$ is represented by the integer $2 + 2^3 + 2^4 = 26$. The monomial 1 (equal to 1 for each input) corresponds to the integer 0. Its *degree* (the number of terms (different from 1) which appear in the product) is 3.

Question 14. *What is the link between the degree of a monomial and the integer which represents it? Implement a function* `monomial_degree(M)` *which computes the degree of a monomial, given as an integer.*

Question 15. *Write a function* `monomial_eval(M, x)` *which evaluates a monomial M at point $x \in \mathbb{F}_2^m$ (both M and x being given as integers).*

The ANF of a given Boolean function can be represented as a list of monomials (those whose sum is equal to the function). There is also another way (both representations are used, we choose depending on the requirements) : a function f is represented as an array A of size 2^m containing only 0s or 1s, the value $A[u]$ of A at position u ($0 \leq u < 2^m$) corresponding to the monomial whose integer representation is u, that is $A[u] = 1$ if and only if the function f has the monomial u in its ANF. For example, for the ANF $x_0 + x_1x_2$, we have $A[1] = A[6] = 1$, and 0 at other positions, because x_0 is encoded as $1 = 2^0$ and x_1x_2 as $6 = 2^1 + 2^2$.

Question 16. *Implement a function taking an ANF (in the form of an array) as input, which computes its degree, i.e. the maximum degree of its monomials (for example, the degree of $x_0 + x_1x_2$ is 2).*

Question 17. *Write a function* `ANF_eval(A, x)`.

Question 18.

1. *Write a function taking as input an ANF and outputting the corresponding truthtable.*
2. *What is the complexity? Can you improve it?*
3. *What is the memory complexity? Can you modify in place the input table (so the memory complexity would be $O(2^m)$, the size of the input table)?*
4. *Can you think of an algorithm (and implement it) to do the inverse operation (compute the ANF from the truthtable)?*