ENUMERATIVE CODING LIBRARY

9 September 2013

TÜBİTAK-BİLGEM

National Research Institute of Electronics and Cryptology

M. Oğuzhan Külekci

Edanur Demir

Contents

1. INTRODUCTION

You can read the paper about enumerative coding and algorithms of the class members by M. Oğuzhan Külekci (http://arxiv.org/pdf/1211.2926v1.pdf)

Please careful that for some computations, GNU Multiple Precision Library is used in the Enumerative Coding Library. We have implemented almost all functions two times. One of them takes an unsigned integer parameter and other one takes a GNU Multiple Precision Library parameter. But, all of them compute with GNU Multiple Precision Library inside. Thus, you should write your running command like :

```
$ g++ -o EnumCode main.cpp EnumCode.cpp EnumCode.h -lgmpxx -lgmp
```

You should these "`-lgmpxx -lgmp`" commands for GNU Multiple Precision Library.

2. ENUMCODE BASICS AND TYPES

EnumCode class has four private data members. Two of them are unsigned integer vectors, *alpVector* and *sequence*. Other members are *alphabetSize* (unsigned integer) and *alphabet* (unsigned char*).

`vector<unsigned int> alpVector`

The *alpVector* is a parikh vector. The number of counters is also the size of *alpVector* and the sum of counters is inner sum of *alpVector*. For example, when the number of counters is 3 and the sum is 4, *alpVector* can be <1, 1, 2>. In this example, there are 15 different parikh vectors and the *alpVector* can be any of them.

`vector<unsigned int> sequence`

The *sequence* vector is a sequence of the *alpVector*. Its size is the inner sum of the *alpVector* and the number of symbols is the same with size of *alpVector*. For example, when the *alpVector* is <1, 1, 2>, sequence can be <3 1 2 3>. It means that there are three symbols and they occur 1, 1, 2 times respectively. In this example, there are 12 different sequences and the *sequence* vector can be any of them.

`unsigned int alphabetSize`

The *alphabetSize* is easily the size of *alpVector*.

`unsigned char* alphabet`

The *alphabet* is a character array which includes the alphabet symbols. Its size is the same with the size of the *alpVector*.

3. FUNCTIONS

This chapter describes functions of the EnumCode class.

3.1 Initialization Functions

EnumCode (void)

EnumCode x; //initialize x

EnumCode (unsigned alphabetSize)

EnumCode x (3); // initialize x and set size of all vectors (alphabet and sequence vectors) to 3.

EnumCode (vector<unsigned int> alpVector)

vector <unsigned int> alpVector; …

EnumCode x (alpVector); // initialize *x* and set its alphabet vector to *alpVector* by call-by-value method. Also compute the size of distinct alphabet characters and reset the size of *sequence* vector.

EnumCode (vector<unsigned int> alpVector, vector<unsigned int> sequence)

vector <unsigned int> alpVector, seq; …

EnumCode x (alpVector, seq); // initialize *x* and set its alphabet vector to *alpVector* and its *sequence* vector to *seq* by call-by-value method.

3.2 Set Functions

void SetAlphaSize (unsigned alphabetSize)

Object.SetAlphaSize (size); // reset the size of its alphabet vector to *size* and reset all elements of the alpVector to 0.

void SetVector (vector<unsigned int> alpVector)

vector<unsigned int> alpVector; ...

Object.SetVector (alpVector); // set its alphabet vector to *alpVector* by call-by-value method. Also compute the size of distinct alphabet characters and reset the size of *sequence* vector.

void SetSequence (vector<unsigned int> sequence)

vector<unsigned int> seq; ...

Object.SetSequence (seq); // set its *sequence* vector to *seq* by call-by-value method. Also reset its alphabet vector to proper one which is composed of new sequence vector.

void SetAlphabet (unsigned char alphabet)*

Object.SetAlphabet (alp); // set its alphabet string to *alp* by call-by-value method.

3.3 Get Functions

unsigned int GetAlphaSize (void)

Object.GetAlphaSize (); // return the size of its alphabet vector.

vector<unsigned int> GetVector (void)

Object.GetVector (); // return its alphabet vector.

vector<unsigned int> GetSequence (void)

Object.GetSequence (); // return its sequence vector.

unsigned char GetAlphabet (void)*

Object.GetAlphabet (); // return its alphabet string.

3.4 Main Functions

void Number_to_BinarySeq (unsigned number)

The function takes an unsigned integer number *x* and set sequence vector to *x*'s proper binary representation. Because the *sequence* vector is unsigned vector, 1 represents to 0 and 2 represents 1. For example,

```
…
object.Number_to_BinarySeq (5);
for (int i; i<object.GetSequence().size(); i++)
      cout << object.GetSequence()[i];
cout << endl;
…
```

Output is :

```
12
```

All binary representation of positive numbers begins with number 1. Because the enumerative numbers are positive, the function doesn't set the first 1.

void Number_to_BinarySeq (mpz_t number)

Like *void Number_to_BinarySeq (unsigned number)*, take a GNU mutable-precision-library variable and set *sequence* vector to its proper binary representation.

unsigned BinarySeq_to_Number ( )

This function computes 10-ary representation of the binary number stored in the *sequence* vector and returns the 10-ary value. Careful that if the *sequence* vector is empty, it returns 0.

unsigned BinarySeq_to_Number (mpz_t value)

This function computes 10-ary representation of the binary number stored in the *sequence* vector and sets *value* (mpz_t variable) to this 10-ary number. Careful that if the *sequence* vector is empty, it returns 0.

int Index_to_Vector_ui (unsigned sum, unsigned counter, unsigned index)

The function takes an unsigned integer *index* number and generates corresponding parikh vector on *alpVector*. Block number of the vector is *counter* parameter and the inner sum of blocks is *sum* parameter. If there is an error, it returns -1. Otherwise it returns 0. For example, the index number of the vector *<0, 4, 1>* is 4.

```
…
cout << object.Index_to_Vector_ui(5,3,4) << endl;
cout << "Corresponding vector is " << '\t';
for(int i=0; i<3; i++)
      cout << object.GetVector()[i] << "  ";
cout << endl << endl;
…
```

Output is:

```
0
Corresponding vector is 0  4  1
```

int Index_to_Vector_ui (unsigned index)

The function works like the *Index_to_Vector_ui* function. It takes an unsigned integer *index* number and generates corresponding parikh vector on *alpVector*. Block number of the vector equals the size of the previous alphabet vector and the inner sum of blocks is also computed from the previous alphabet vector. If there is an error, it returns -1. Otherwise it returns 0.

int Index_to_Vector (unsigned sum, unsigned counter, mpz_t index)

The function takes a GNU mutable-precision-library variable (index number) and generates corresponding parikh vector on *alpVector*. Block number of the vector is *counter* parameter and the inner sum of the blocks is *sum* parameter. If there is an error, it returns -1. Otherwise it returns 0.

int Index_to_Vector (mpz_t index)

The function works like the *Index_to_Vector_ui* function. It takes a GNU mutable-precision-library variable (index number) and generates corresponding parikh vector on *alpVector*. Block number of the vector equals the size of the previous alphabet vector (*alpVector*) and the inner sum of blocks is also computed from the previous alphabet vector. If there is an error, it returns -1. Otherwise it returns 0.

unsigned Vector_to_Index (void)

The function returns the index number of its alphabet vector (*alpVector*). Careful that if the alphabet vector is empty, it returns 0. For example,

```
…
cout << object.Index_to_Vector_ui(5,3,4) << endl;
cout << "Corresponding vector is " << '\t';
for(int i=0; i<3; i++)
      cout << object.GetVector()[i] << "  ";
cout << endl;
cout << object.Vector_to_Index( ) << endl;
…
```

Output is:

```
0

Corresponding vector is 0  4  1

4
```

unsigned Vector_to_Index (mpz_t index)

The function takes a GNU mutable-precision-library variable (index number) by call-by-references and reset it to the index number of its alphabet vector (*alpVector*). Careful that if the alphabet vector is empty, it returns 0.

unsigned PermIndex_to_Seq (mpz_t permID)

The function takes permutation rank of the sequence and generates the corresponding *sequence* vector by using alphabet vector. Careful that if the alphabet vector is empty, it returns 0. For example, alphabet vector is *<3 2 1>* which means the first character appears 3 times, the second character appears 2 times etc. when the index number is 0, corresponding *sequence* vector is *<1 1 1 2 2 3>*. When the index number is 1, corresponding *sequence* vector is *<1 1 1 2 3 2>*.

```
…

mpz_set_ui(index,1);

object.SetVector(vec); // vec = <3,2,1>

object.PermIndex_to_Seq(index);

cout << "Corresponding sequence is " << '\t';

for(int i=0; i<object.GetSequence().size(); i++)

      cout << object.GetSequence()[i] << "  ";

cout << endl;

…
```

Output is:

> *Corresponding sequence is 1 1 1 2 3 2*

unsigned Seq_to_PermIndex (void)

The function returns permutation rank of its *sequence* vector. Careful that if the sequence vector is empty, it returns 0. For example, the *sequence* vector is <*1 1 1 2 3 2*>, its rank is 1.

unsigned Seq_to_PermIndex (mpz_t permID)

The function reset GNU mutable-precision-library variable *permID* to the permutation rank of its *sequence* vector. Careful that if the *sequence* vector is empty, it returns 0. For example, the *sequence* vector is <*1 1 1 2 3 2*>, its rank is 1.

unsigned Count_ParVectors (unsigned sum, unsigned counter)

The function takes inner sum of the alphabet vector (*unsigned sum*) and the block number (u*nsigned counter*) and returns the number of parikh vectors. If there is an error, it returns 0. For example, when the *sum* is 3 and the *counter* is 2, the number of parikh vectors is 4. (<0,3>, <1,2>, <2,1>, <3,0>)

unsigned Count_ParVectors (unsigned sum, unsigned counter, mpz_t value)

The function takes inner sum of the alphabet vector (*unsigned sum*), the block number (u*nsigned counter*), a GNU mutable-precision-library variable (*mpz_t value*) and resets *value* to the number of parikh vectors. If there is an error, it returns 0. For example, when the *sum* is 3 and the *counter* is 2, the number of parikh vectors is 4. (<0,3>, <1,2>, <2,1>, <3,0>)

unsigned Number_to_BinaryIndex (unsigned num)

The function takes an unsigned number, and computes corresponding binary number, returns its binary index. If there is an error, it returns 0. For example, 54 is represented <*110110*>, and its permutation rank is 6.

unsigned Number_to_BinaryIndex (unsigned num, mpz_t value)

The function takes an unsigned number, and computes corresponding binary number, resets value to its binary index. If there is an error, it returns 0. For example, 54 is represented <*110110*>, and its permutation rank is 6.

Notice, because the all positive numbers begin with *1*, these two functions compute binary representation of the 54 like <10110>.

unsigned BinaryIndex_to_Number (unsigned index, pair<int, int> p)

The function takes an index number and a pair whose first number is the size of the corresponding binary sequence except the first number (for 54 = <110110> is 5) and the second number is the number of zeros (for 54 = <110110> is 2). The function returns the corresponding unsigned *number*. For example,

```
…
  pair <int, int> p = make_pair(5,2);
  cout << enumcode.BinaryIndex_to_Number(6,p) << endl;
…
```

Output is :

```
54
```

void BinaryIndex_to_Number (unsigned index, pair<int, int> p, mpz_t value)

The function takes an index number and a pair whose first number is the size of the corresponding binary sequence except the first number (for 54 = <110110> is 5) and the second number is the number of zeros (for 54 = <110110> is 2). The function resets *value* to the corresponding unsigned *number*.

void test()

The *test* function is implemented in order to show how to use these functions.