Математика и ее приложения : ЖИМО. 2013. Вып. 1 (10). С. 71–76

УДК 512.543

С. И. Хашин[1]

Эффективная реализация систем булевых полиномов

Ключевые слова: булевы полиномы, автоморфизм.

При разработке алгоритмов булева сжатия информации и в некоторых разделах криптографии возникает задача эффективного нахождения множества, порождающего данную систему булевых полиномов. Предлагаемый в настоящей работе метод ее решения эффективен, если количество булевых переменных не превосходит 32.

Keywords: boolean polynomials, automorphism.

In developing algorithms for Boolean data compression and in some sections of the cryptography, the problem of effective finding the generated set for given system of Boolean polynomials. This paper proposes a method for its effective solution if the number of Boolean variables does not exceed 32.

1. Введение

Для некоторых задач криптографии ([4]), сжатия информации ([2, 5, 6, 7]) требуется эффективная реализация работы с булевыми полиномами. Требования таких программ к памяти быстро растут с ростом количества булевых переменных.

В настоящей работе описывается реализация таких алгоритмов и оценивается их эффективность.

Предлагаемая реализация также может использоваться в учебных целях, при изучении дисциплин, использующих булеву алгебру: дискретной математики, защиты информации, распознавании образов.

Работа с булевыми полиномами, описанная ниже собрана в два файла: `booleanf.h` и `booleanf.cpp`. Для ее работы не требуются никакие дополнительные библиотеки. Из стандартных `include`-файлов подключается только `stdlib.h`. Динамическое выделение памяти также не используется. Даже при конвертировании объектов в строку (методы `toString` классов `SC` и `BP`) используется всегда один и тот же массив, выделяемый статически. С одной стороны, это создает определенные неудобства для пользователя, с другой – уменьшает риск возникновения “утечек памяти“, (подробнее – см. далее).

[1]Ивановский государственный университет; E-mail: `khash2@mail.ru`. Работа выполнена при поддержке гранта РФФИ 13-07-00628

Исходные файлы размещены в интернете [8], доступны для скачивания без ограничений и регистрации.

2. Кортеж (corteg)

Определение 1. *Кортежем будем называть цепочку из битов фиксированной длины. Ее длину будем считать фиксированной и обозначать* `nVar`.

Настройка пакета на работу с булевыми многочленами от данного количества переменных осуществляется с помощью константы `nVar`, определенной в файле `booleanf.h`:

```
#define nVar 8   // the number of boolean variables
```

Таким образом, количество булевых переменных задается один раз и до конца работы программы. Это, конечно же, является недостатком реализации но, с другой стороны, заметно упрощает работу с системой. В реальных задачах, где приходилось использовать булевы полиномы, количество исходных переменных всегда оставалось одним и тем же, поэтому указанный недостаток не являлся существенным.

Обычно в наших примерах `nVar` будет равно 8, то есть кортеж является байтом, но не исключаются и другие варианты. Кортеж определяется беззнаковым числом, лежащим в пределах от 0 до $2^{nVar} - 1$. Поэтому при $nVar \leq 8$ кортеж представлен байтом (unsigned char), при $nVar \leq 16$ – беззнаковым коротким числом (unsigned short), при $N \leq 32$ – беззнаковым целым(unsigned). Никаких особых операций над кортежами, помимо обычных арифметических и битовых операций над беззнаковыми числами мы пока не определяем.

3. Множество кортежей, класс SC

Следующий класс представляем множество кортежей. Фактически, это просто массив из 2^N битов. Его же можно рассматривать как функцию на множестве всех кортежей со значениями $\{0, 1\}$. Основные методы класса:

```
void set0();        // this = 0
void ins(corteg a); // добавить к множеству кортеж a
void join (SC &s);  // добавить к this множество s
void inters(SC &s); // this := this пересечение с множеством s
void xor (SC &s);   // this := this XOR s
bool get(corteg a); // входит ли кортеж a в множество
bool isBitSymmetric( int k );  // симметрично ли множество
                    // относительно бита k
int  weight();      // количество элементов в множестве char
*toString();
```

Они достаточно очевидны и не требуют особых комментариев, кроме последнего (toString). Чтобы не сталкиваться с проблемами выделения и освобождения памяти, эта функция строит ответ всегда в одной и той же статически созданной строке и именно еҷ возвращает в качестве результата. Это удобно, так как пользователю не надо заботиться ни о выделении, ни об освобождении памяти. С другой стороны, это приводит к невозможности одновременного использования двух и более объектов в одном выражении, типа

```
printf( "s=%s; t=%s\n", s.toString(), t.toString() );
```

4. Булевы полиномы, класс BP

Булевы полиномы – это суммы некоторого количества мономов. Каждый моном определяется списком переменных, в него входящих, то есть некоторым кортежем. Поэтому как множество, булев полином – то же самое, что и множество кортежей. Различаются они набором операций и методов.

Замечание 1. *Каждому кортежу b длины $nVar$ соответствует булев моном $M(b)$ от переменных $(x_0, \ldots x_{nVar-1})$, например*

$$M(10010011) = x_0 \cdot x_1 \cdot x_4 \cdot x_7$$

Значение монома $M(b)$ на кортеже a равно 1, если `b & a == b`. В противном случае это значение равно 0.

Основные методы класса:

```
void set0();              // обнулить полином
void set1(corteg a);      // коэффициент при мономе a := 1
void add1(corteg a);      //прибавить моном a к полиному
void add(BP &b);          // this += b
void mult(BP &b, BP &c); // this = b*c
bool get(corteg a);       //значение полинома в точке a
void values(SC &r);       //r:= значение полинома во всех точках
void Roots (SC &r);       // r:= множество корней полинома
void OneUnit(corteg a);   // строим полином равный 1 в точке a
                          // и 0 в остальных
void OneRoot(corteg a);   // строим полином равный 0 в точке a
                          // и 1 в остальных
void or(BP &b, BP &c);    // this = b OR c
void givenRoots(SC &r);//строим полином с данным набором корней
void givenUnits(SC &r);//строим полином с данным набором единиц
                          // (т.е. не корней)
char *toString();
```

5. Автоморфизмы кольца булевых полиномов

В работе [3] описаны гомоморфизмы колец булевых полиномов $\mathrm{Hom}(G_N, G_M)$, а также группа автоморфизмов кольца G_N. В описываемом пакете имеется реализация этого механизма. А именно, имеется функция

```
void automorphism( BP *f, corteg *s)
```

которая по данной перестановке кортежей `s` находит автоморфизм `f`. Перестановка задается в виде множества кортежей

$$(s_0, s_1, \ldots, s_{2^{nVar}-1}),$$

то есть в виде массива длиной 2^{nVar}. Искомый авторомфизм возвращается в массиве f булевых полиномов

$$\begin{array}{lcl} x_0 & \to & f_0(x_0, \ldots, x_{nVar-1}), \\ x_1 & \to & f_1(x_1, \ldots, x_{nVar-1}), \\ & \ldots & \\ x_{N-1} & \to & f_{nVar-1}(x_1, \ldots, x_{nVar-1}), \end{array}$$

Например, при $nVar = 8$ входной массив s (подстановка) является массивом из 256 байтов, выходной (f) – массивом из 8 многочленов, каждый размером по $256/8 = 32$ байта.

При $nVar = 16$, s – массив из 2^{16} двухбайтовых кортежей, выходной – массивом из 16 многочленов, каждый размером по $2^{16}/8 = 8192$ байта. То есть общий объем требуемой памяти около 256 Кбайт.

При $nVar = 32$ массив s будет занимать уже 16 Гбайт, а массив f – еще столько же.

6. Использование пакета

Единственная требуемая настройка пакета – выбор количества булевых переменных. Она осуществляется с помощью константы `nVar`, определенной в файле `booleanf.h`:

```
#define nVar 8   // the number of boolean variables
```

Таким образом, количество булевых переменных задается один раз и до конца работы программы.

В зависимости от величины`nVar` тип `corteg` имеет разные определения:

```
#if   nVar<=  8 typedef unsigned char corteg;
#elif nVar<= 16 typedefunsigned short corteg;
#elif nVar<= 32 typedef unsigned corteg;
#elif nVar<= 64 typedef unsigned __int64 corteg;
#else
#error nVar too large
#endif
```

Таким образом, константа `nVar` не должна принимать значения, большие 64. На самом деле, при `nVar` большем 32 объекты типов `SC` и `BP` потребуют слишком много памяти и работа с ними может быть затруднена или вообще невозможна. Даже при `nVar=32` следует очень осторожно подходить созданию объектов из этих классов.

Приведем два простых примера использования описанного пакета:

```
#include "BP.h" ...
    SC s, t;
    s.ins(7);
    s.ins(17);
    s.ins(200);
    printf( "s=%s>\n", s.toString() );
```

Будет выведено на экран множество, состоящее из трех элементов: $\{7, 17, 200\}$.

```
#include "BP.h" ...
    BP f;
    SC r;
    f.add1(2);f.add1(5);
    printf( "f=<%s>\n", f.toString() );
    f.Roots(r);
    printf( "r=<%s>\n", r.toString() );
    f.givenRoots(r);
    printf( "F=<%s>\n", f.toString() );
```

7. Заключение

Описанный в статье способ работы с булевыми полиномами достаточно эффективен при количестве булевых переменных, не превосходящем 32. В этом случае объект типа `SC` (множество кортежей) или `BP` (булевский полином) будет занимать 512 Мбайт памяти. Для современных компьютеров это вполне допустимой значение.

В работах ([6, 7]) сформулирована задача поиска для данного набора булевых многочленов $(f_1, \ldots, f_s)$ из G_N наименьшего количества полиномов $(h_1, \ldots, h_M)$, через которые можно выразить исходные полиномы.

Решение этой задачи с помощью методов, описанных в настоящей работе при 32 булевых переменных требуется оперативная память размером 16 Гбайт. На сегодняшний день это большая, но вполне реальная цифра.

При большем количестве булевых переменных предложенный предложенная реализация неработоспособна. В этом случае придется искать более эффективные алгоритмы.

Список литературы

1. *Владимиров Д. А.* Булевы алгебры. М. : Наука, 1969. 320 с.
2. *Гришко М. Е.* Один из возможных способов разбиения файла на буферы при булевом сжатии файлов // Математика и ее приложения : журн. Иван. мат. о-ва. 2010. Вып. 1(7). С. 25–28.
3. *Логачев О. А., Сальников А. А., Ященко В. В.* Булевы функции в теории кодирования и криптологии. М. : МЦНМО, 2004. 470 с.
4. *Ростовцев А. Г.* Защита от side channel attack на основе случайных изоморфизмов. 2004. URL: http: //www.ssl.stu.neva.ru/ssl/archieve/sidech1.pdf (дата обращения: 4.12.2013)
5. *Толстопятов А. А.* О возможности использования булевых уравнений для сжатия файлов // Вестник Иван. гос. ун-та. 2003. Вып. 3. С. 82–84.
6. *Толстопятов А. А.* Алгоритм разбиения файла на буферы при булевом сжатии // Математика и ее приложения : журн. Иван. мат. о-ва. 2008. Вып. 1(5). С. 77–88.
7. *Толстопятов А. А.* Построение кодирующего уравнения при булевом сжатии файлов // Математика и ее приложения : журн. Иван. мат. о-ва. 2010. Вып. 1(7). С. 69–83.
8. *Хашин С. И.* Булевы полиномы. 2013.
URL: http://math.ivanovo.ac.ru/dalgebra/Khashin/BP/index_r.html (дата обращения: 12.12.2013)
9. *Хашин С. И., Хашина Ю. А.* Свойства b-ранга системы булевых полиномов // Математика и ее приложения : журн. Иван. мат. о-ва. 2013. Вып. 1(10). С. 65–70.

Поступила в редакцию 12.12.2013.