О РЕАЛИЗАЦИИ N-МЕРНОГО BMS-АЛГОРИТМА СРЕДСТВАМИ ОБОБЩЁННОГО ПРОГРАММИРОВАНИЯ

Пеленицын А.М.
Южный федеральный университет

We describe the n-dimensional BMS-algorithm in the way that was used for our implementation of the algorithm. We present the implementation of BMS-algorithm that hugely use generic programming techniques based on C++ templates mechanism. We discuss main units, their interfaces and semantics and the tools involved in implementation.

1. Введение. Алгоритм Берлекэмпа–Месси (BM-алгоритм), созданный в конце шестидесятых годов, нашёл многочисленные применения в различных областях математики и информатики, обзор последних наиболее важных результатов содержится в [1]. В связи с декодированием алгебро-геометрических кодов появилась необходимость обобщения BM-алгоритма на «многомерные последовательности». Впервые такое обобщение предложил С. Саката [2], [3] и теперь многомерный вариант алгоритма принято называть алгоритмом Берлекэмпа—Месси—Сакаты (BMS-алгоритмом). Современное изложение BMS-алгоритма приведено в [4] и [5]. В [6] обсуждается теоретико-операторный подход к BMS-алгоритму, приложения этого алгоритма к актуальным задачам помехоустойчивого кодирования обсуждаются в [7]. В [8] представлена реализация кодека для одного класса алгебро-геометрических кодов с использованием двумерного BMS-алгоритма.

В данной работе рассматривается реализация n-мерного BMS-алгоритма средствами обобщённого программирования [9, гл. 14], [10, п. 7.2].

2. BMS-алгоритм. Пусть $\mathbb{N} = \{0; 1; 2; \ldots\}$. Элементы $\mathbb{N}^n$ будем называть точками и обозначать полужирным шрифтом, например: $\mathbf{m} = (m_1, \ldots, m_n) \in \mathbb{N}^n$.

Определим сумму точек: $\mathbf{m} + \mathbf{k} = (m_1 + k_1, \ldots, m_n + k_n)$. Аналогично будем использовать разность точек тогда, когда она корректно определена. Введём два отношения порядка на $\mathbb{N}^n$:

1.
$$\mathbf{m} < \mathbf{k} \iff \Big((\sum_i m_i < \sum_i k_i) \vee \\ (\sum_i m_i = \sum_i k_i) \wedge \big(\exists j\, \forall i > j \colon (m_i = k_i) \wedge (m_j < k_j)\big)\Big)$$

2. $\mathbf{m} \leqslant_{\mathrm{P}} \mathbf{k} \Leftrightarrow \forall i \colon m_i \leqslant k_i$.

Первое отношение порядка является отношением полного порядка и называется градуированным антилексикографическим порядком [5, с. 8] (заметим, что в качестве отношения $<$ можно взять любой другой мономиальный порядок). Второе отношение порядка является отношением частичного порядка. Полный порядок позволяет для каждой точки $\mathbf{m}$ определить *непосредственно следующую за ней* точку, обозначаемую $\mathbf{m}'$.

Пусть $\mathbb{F}$ — фиксированное конечное поле. Полином от n переменных над $\mathbb{F}$, то есть элемент кольца $\mathbb{F}[x_1, \dots, x_n]$, будем записывать так:

$$f(x) = \sum_{\mathbf{i} \in \Gamma_f} f_{\mathbf{i}} x^{\mathbf{i}},$$

где $\Gamma_f (\subset \mathbb{N}^n)$ — конечное множество, $x^{\mathbf{i}} = x_1^{i_1} \dots x_n^{i_n}$, $f_{\mathbf{i}} \in \mathbb{F}$. Полный порядок $<$ на $\mathbb{N}^n$ позволяет корректно определить (*старшую*) *степень* полинома от n переменных f, которую будем обозначать $\deg(f)$:

$$\deg(f) = \max_{<} \{\mathbf{m} \in \Gamma_f\} \ (\in \mathbb{N}^n),$$

Нормированными будем называть такие полиномы $f(x)$, у которых коэффициент при старшей степени $f_{\deg(f)}$ равен 1.

Любое отображение $u \colon \mathbb{N}^n \to \mathbb{F}$ будем называть n-мерной последовательностью. Здесь и далее будут рассматриваться только такие n-мерные последовательности, количество отличных от нуля членов которых конечно. Для каждой такой последовательности u определена степень:

$$\deg(u) = \max_{<} \{\mathbf{m} \mid u_{\mathbf{m}} \neq 0\} \quad (\in \mathbb{N}^n).$$

Пусть f — полином от n переменных, u — n-мерная последовательность, $\deg(f) \leqslant_{\mathrm{P}} \mathbf{m} < \deg(u)$. Обозначим:

$$f[u]_{\mathbf{m}} = \sum_{\mathbf{i}} f_{\mathbf{i}} u_{\mathbf{i}+\mathbf{m}-\mathbf{s}} \quad (\in \mathbb{F}).$$

Будем писать $f[u] = 0$, если

$$\forall \mathbf{m} \in \{\mathbf{i} \mid \deg(f) \leqslant_{\mathrm{P}} \mathbf{i} < \deg(u)\} \colon f[u]_{\mathbf{m}} = 0.$$

Отметим, что для некоторых f, u множество $\{\mathbf{i} \mid \deg(f) \leqslant_{\mathrm{P}} \mathbf{i} < \deg(u)\}$ пустое.

Пусть дана последовательность u. *Минимальным множеством полиномов* [3, п. 4] для последовательности u называется множество нормированных полиномов $\{f^{(i)}(x)\}_{i=1}^{k}$ такое, что

(1) $\forall i \colon f^{(i)}[u] = 0$;

(2) $\forall g(x) \colon (g[u] = 0) \to (\exists i \colon \deg(f^{(i)}) \leqslant_{\mathrm{P}} \deg(g))$;

(3) $\forall i\ \forall j \neq i\colon \deg(f^{(i)}) \nleqslant_{\mathrm{P}} \deg(f^{(j)})$.

BMS-алгоритм строит по заданной последовательности u её минимальное множество полиномов. Приведём шаги BMS-алгоритма, в соответствии, в основном, с [5].

Шаг 0. Пусть $F, G\colon \mathbb{N}^n \to \mathbb{F}[x_1, \ldots, x_n]$ — два отображения, такие что $F(0, \ldots, 0) = 1$, а в любой другой точке F равно 0, G равно нулю в любой точке. В промежуточных вычислениях будут определены «временные» версии F, G — отображения $F_{\mathrm{new}}, G_{\mathrm{new}}$. $\delta = \varnothing$, $\sigma = \{(0, \ldots, 0)\} \subset \mathbb{N}^n$. Введём также отображение $d\colon \mathbb{N}^n \to \mathbb{F}$. Положим $\mathbf{k} = (0, \ldots, 0)$.

Шаг 1. Пусть $\delta' = \varnothing$. Для каждой точки $\mathbf{s} \in \sigma$: если $\mathbf{s} \leqslant_{\mathrm{P}} \mathbf{k}$, то

(a) положим $d(\mathbf{s}) = (F(\mathbf{s}))[u]_{\mathbf{k}}$;

(b) если $d(\mathbf{s}) \neq 0$ и существует такое $\mathbf{t} \in \sigma$, что $\mathbf{t} \leqslant_{\mathrm{P}} \mathbf{k} - \mathbf{s}$, то добавим в множество δ' точку $\mathbf{k} - \mathbf{s}$.

Шаг 2. Положим $\delta_{\mathrm{new}} = \min_{\leqslant_{\mathrm{P}}}(\delta' \cup \delta)$. $\sigma_{\mathrm{new}} = \min_{\leqslant_{\mathrm{P}}}\{\mathbf{s} \in \mathbb{N}^n \mid \nexists \mathbf{c} \in \delta_{\mathrm{new}}\colon \mathbf{s} \leqslant_{\mathrm{P}} \mathbf{c}\}$.

Шаг 3. Для каждой точки $\mathbf{c} \in \delta_{\mathrm{new}}$: если $\mathbf{c} \in \delta$, то положим $G_{\mathrm{new}}(\mathbf{c}) = G(\mathbf{c})$, иначе $G_{\mathrm{new}}(\mathbf{c}) = (d(\mathbf{k} - \mathbf{c}))^{-1} F(\mathbf{k} - \mathbf{c})$.

Шаг 4. Для каждой точки $\mathbf{t} \in \sigma_{\mathrm{new}}$:

(a) найти $\mathbf{s} \in \sigma$, такое что $\mathbf{s} \leqslant_{\mathrm{P}} \mathbf{t}$; положить $\mathbf{u} = \mathbf{t} - \mathbf{s}$;

(b) если $\mathbf{t} \leqslant_{\mathrm{P}} \mathbf{k}$ и существует $\mathbf{c} \in \delta$, такое что $\mathbf{k} - \mathbf{t} \leqslant_{\mathrm{P}} \mathbf{c}$, то положить

$$F_{\mathrm{new}}(\mathbf{t}) = F(\mathbf{s}) x^{\mathbf{u}} - d(\mathbf{s}) G(\mathbf{c}) x^{\mathbf{c} - (\mathbf{k} - \mathbf{t})},$$

иначе:

$$F_{\mathrm{new}}(\mathbf{t}) = F(\mathbf{s}) x^{\mathbf{u}}.$$

Шаг 5. Положить $F = F_{\mathrm{new}}$, $G = G_{\mathrm{new}}$, $\delta = \delta_{\mathrm{new}}$, $\sigma = \sigma_{\mathrm{new}}$, $\mathbf{k} = \mathbf{k}'$. Если $\mathbf{k} < \deg(u)$ или $\mathbf{k} = \deg(u)$, то перейти на шаг 1, иначе завершить алгоритм, подав на выход множество $F(\sigma)$.

3. Архитектура реализации. Создана библиотека, реализующая работу с полиномами многих переменных в объёме, необходимом для выполнения BMS-алгоритма, а также сам алгоритм. Библиотека в терминологии C++ является «headers-only», то есть вся реализация содержится в заголовочных файлах C++, предназначенных для текстового включения в использующие её проекты и, таким образом, существует, прежде всего, в виде открытых исходных кодов. Она включает в себя три основных шаблонных класса, `Point<N>`, `Polynomial<T>`, `BMSAlgorithm< PolynomialT >` и набор вспомогательных шаблонных классов и свободных функций, занимающих в общей сложности около полутора тысяч строк кода (включая документацию внутри исходного кода).

Наличие свободных функций является отступлением от чистого объектно-ориентированного дизайна и следует рекомендациям [11, п. 44], касающимся использования языка программирования C++ как мультипарадигменного. В соответствии с этими рекомендациям в данной реализации особое внимание уделено применению обобщённого программирования на основе шаблонов C++. Объектно-ориентированный подход используется лишь в малой степени, как дополнительное средство поддержки модульности: наследование применяется лишь в нескольких местах для удобства реализации идиом обобщённого программирования, не используется (динамический, основанный на виртуальных функциях) полиморфизм.

Шаблон класса `Point<N>` моделирует точку в $\mathbb{N}^N$. С точки зрения реализации он представляет собой обёртку над шаблоном `tr1::array<N, int>` из библиотеки TR1 [12], который, в свою очередь, является статическим массивом с STL-совместимым интерфейсом [12, п. 6.2]. К отдельным координатам точки можно обращаться с помощью операции взятия индекса (`operator[]`). Важной частью интерфейса `Point<N>` являются функции, реализующие различные порядки на множестве $\mathbb{N}^N$, такие как `byCoordinateLess` и `totalLess`. Первая из них, свободная функция, обеспечивает покоординатное сравнение точек (частичный порядок), вторая, функция-член, задаёт градуированный антилексикографический порядок. Кроме того, имеется набор свободных функций, реализующих поиск в наборе точек минимумов и максимумов относительно разных порядков, поиск в наборе точек точки, меньшей заданной и т. п. Всё это необходимо выполнять на разных стадиях BMS-алгоритма.

Реализация `totalLess` в форме функции-члена (в отличие от других функций, связанных с упорядочением) связана с желанием параметризовать тип точки мономиальным упорядочением. Такая параметризация доступна с помощью использования второго параметра шаблона `Point<N>`, который имеет значение по умолчанию, соответствующее градуированному антилексикографическому упорядочению.

Шаблон класса `Polynomial<T>` представляет тип полинома, причём типовый параметр `T` обозначает тип коэффициентов соответствующего множества полиномов. Технически `Polynomial<T>` является особого рода контейнером для элементов типа `T`. Полиномы многих переменных получаются из `Polynomial<T>`, так как, к примеру, полином от двух переменных с коэффициентами типа `T` можно отождествить с полиномом от одной переменной с коэффициентами — полиномами от одной переменной и коэффициентами типа `T`; таким объектам соответствует тип `Polynomial<Polynomial<T>>`, который можно получить, используя рассматриваемую библиотеку.

Для удобства использования полиномов многих переменных введён отдельный шаблон класса `MVPolyType<n, T>`, где первый параметр указывает на количество переменных в результирующем типе. Например, следующие две строки кода определяют переменные одинакового типа, представляющего полином от трёх переменных с целочисленными коэффициентами.

```
Polynomial< Polynomial< Polynomial<int> > > p;
MVPolyType<3, int>::ResultT q;
```

Полиномы можно задавать в специальном строковом представлении, перечисляя коэффициенты полинома в квадратных скобках через пробел в порядке возрастания степеней. Например, ‘‘`[2 0 1]`’’ соответствует полиному $x^2 + 2$. Так как полином от нескольких переменных это обычный полином с коэффициентами — полиномами от переменных, число которых меньше на единицу, то строковое представление таких полиномов будет содержать вложенные скобки. Например, ‘‘`[[2 0 2] [1] [0 3]]`’’ соответствует полиному $3x^2y + 2y^2 + x + 2$ (с точностью до возможного переименования переменных).

Для типов полиномов перегружены арифметические операции, которые необходимы для реализации BMS-алгоритма. Это сложение полиномов, умножение полинома на скаляр и умножение полинома на моном (`operator<<`) — таким образом, из основных операций с полиномами не реализовано лишь умножение. Поясним использование операции умножения на моном. Считается, что моном задаётся своей степенью, которая, в свою очередь, представлена типом `Point<N>`. Вот пример умножения полинома от двух переменных на моном x^3y^2:

```
MVPolyType<2, int>::ResultT p("[[1 2] [3]]");
Point<2> mon; mon[0] = 3; mon[1] = 2;
p <<= mon;
```

Шаблон класса `BMSAlgorithm<PolynomialT>` параметризован типом полиномов (в частности, алгоритм может работать с полиномами различного числа переменных) и имеет довольно простой интерфейс, отражающий назначение и использование BMS-алгоритма. Напомним, что BMS-алгоритм по заданной конечной n-мерной последовательности строит конечное множество полиномов от n переменных, называемое минимальным множеством этой последовательности. С технической точки зрения, n-мерная последовательность ничем не отличается от полинома от n переменных, потому тип соответствующего аргумента алгоритма совпадает с типом полиномов, получаемых на выходе алгоритма. Конструктор класса принимает последовательность и точку, обозначающую конец последовательности, алгоритм обрабатывает все элементы в диапазоне от элемента с мультииндексом $(0, \dots, 0)$ до этого маркера конца последовательности в порядке, заданном используемым мономиальным порядком (конкретный пордяок является частью описания типа точки). Метод данного шаблона класса `computeMinimalSet` возвращает коллекцию с STL-совместимым интерфейсом, содержащую выход алгоритма. Тип этой коллекции является частью определения шаблона класса, к нему можно обратиться следующим образом: `BMSAlgorithm<PolynomialT>::PolynomialCollection`.

4. Инструменты реализации. Реализация выполнена на языке программирования C++ [13] с использованием стандартных расширений TR1 [12] (поддержка программирования с STL), библиотеки NTL [14] (арифметика в конечных полях), части коллекции библиотек Boost [15] (поддержка программирования с STL), библиотеки Loki [16] (поддержка обобщённого программирования). Для трансляции использовался компилятор с языка C++ из свободной коллекции компиляторов GNU Compiler Collection (GCC) версии 4.4, в составе которого имеется, в частности, реализация TR1. Проект размещён в сети интернет на площадке Google Code [17]: исходные коды доступны для просмотра онлайн, кроме того, имеется доступ на чтение в Mercurial-репозиторий.

Результаты работы созданной реализации BMS-алгоритма тестировались с помощью примеров, приведённых в статьях [2] и [3]. Кроме того, для отдельных программных модулей был создан набор тестов, использующих свободную легковесную систему автоматизированного модульного тестирования CUTE (C++ Unit Testing Easier) [18].Основные программные сущности документированы в исходном коде в формате Doxygen [20], что позволяет извлекать документацию к проекту запуском соответствующей утилиты (возможные выходные форматы: HTML, LaTeX).

Eclipse IDE предоставляет средства интеграции почти всех перечисленных инструментов: связь с Mercurial-репозиторием (посредством плагина Mercurial-Eclipse), развёрнутым на площадке Google Code, компиляция, линковка и отладка средствами GCC (посредством плагина CDT), визуальное отображение результатов тестов CUTE (посредством соответствующего плагина, предлагающегося на сайте CUTE), генерация тегов при добавлении Doxygen-комментариев к модулям программы.

ЛИТЕРАТУРА

[1] *Куракин В.Л.* Алгоритм Берлекэмпа—Месси над коммутативными артиновыми кольцами главных идеалов // Фундаментальная и прикладная математика. 1999. Т. 5, вып. 4. С. 1061–1101.

[2] *Sakata S.* Finding a minimal set of linear recurring relations capable of generating a given finite two–dimensional array // J. Symb. Comp. 1988. Vol. 5. Pp. 321–337.

[3] *Sakata S.* Extension of the Berlekamp—Massey algorithm to N dimensions // Inform. and Comput. 1990. Vol. 84. No. 2, P. 207–239.

[4] *Sakata S.* The BMS algorithm // In Sala M. et al. (ed.), Gröbner bases, coding, and cryptography. Springer, 2009. P. 143–163.

[5] *Cox D.A., Little J.B., O'Shea D.B.* Using Algebraic Geometry, Second Edition. Springer, 2005. 496 p.

[6] *Деундяк В.М., Пеленицын А.М.* Теоретико-операторный подход к алгоритму Берлекэмпа–Месси–Сакаты // Изв. вузов. Сев.-Кав. регион. Естественные науки. 2011 (в печати).

[7] *Sakata S.* The BMS algorithm and Decoding of AG Codes // In Sala M. et al. (ed.), Gr obner bases, coding, and cryptography. Springer, 2009. P. 143–163.

[8] *Маевский А.Э., Пеленицын А.М.* Реализация программного алгебро-геометрического кодека с применением алгоритма Сакаты // Изв. ЮФУ. Технические науки. 2008. № 8. С. 196–198.

[9] *Вандевурд Д., Джосаттис Н.* Шаблоны C++: справочник разработчика. М.: Вильямс, 2003. 544 с.

[10] *Stroustrup B.* Evolving a language in and for the real world: C++ 1991–2006 // Procs. of the ACM HOPL-III. 2007. Pp. 4-1–4-59.

[11] *Саттер Г., Александреску А.* Стандарты программирования на C++. М.: Вильямс, 2005. 224 с.

[12] *International Standards Organization*: C++ Library Extensions. International Standard ISO/IEC TR 19768
URL: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2005/n1836.pdf (дата обращения: 03.08.2010).

[13] *International Standards Organization*: Programming Languages – C++. International Standard ISO/IEC 14882:1998.

[14] *Shoup V.* NTL: A Library for doing Number Theory // URL: http://shoup.net/ntl/ (дата обращения: 03.08.2010).

[15] Boost C++ Libraries. URL: http://www.boost.org/ (дата обращения 03.08.2010).

[16] Loki C++ library // URL: http://loki-lib.sourceforge.net/ (дата обращения: 25.08.2010).

[17] *Pelenitsyn A.* Multivariate polynomials for C++
URL: http://code.google.com/p/cpp-mv-poly/ (дата обращения: 03.08.2010).

[18] CUTE: C++ Unit Testing Easier // URL: http://r2.ifs.hsr.ch/cute (дата обращения: 03.08.2010).

[19] Официальный сайт Eclipse IDE // URL: http://www.eclipse.org/ (дата обращения: 03.08.2010).

[20] Официальный сайт Doxygen // URL: http://www.stack.nl/ dimitri/doxygen/ (дата обращения: 03.08.2010).