О свободных и несвободных модулях, порождённых крестовыми матрицами

А. В. Решетников

Пусть R – ассоциативное кольцо с единицей. Зафиксируем два натуральных числа m и n. Множество всех матриц размера $m \times n$, составленных из элементов множества R, будем обозначать через $\mathrm{M}_{m\times n}(R)$. Введём на этом множестве покомпонентные операции сложения матриц и умножения их слева на элементы кольца R:

$$\mathbf{Q}_1 + \mathbf{Q}_2 \in \mathrm{M}_{m\times n}(R) \quad \text{для всех } \mathbf{Q}_1, \mathbf{Q}_2 \in \mathrm{M}_{m\times n}(R);$$

$$\lambda\mathbf{Q} \in \mathrm{M}_{m\times n}(R) \quad \text{для всех } \lambda \in R,\ \mathbf{Q} \in \mathrm{M}_{m\times n}(R).$$

Относительно этих операций множество $\mathrm{M}_{m\times n}(R)$ является *левым модулем* над R.

Модуль $\mathrm{M}_{m\times n}(R)$ свободный: его базис состоит из всех таких матриц $\mathbf{E}_{ij}$ $(1 \leq i \leq m;\ 1 \leq j \leq n)$, у которых каждая компонента, за исключением (i,j)-ой, равна 0, а компонента на (i,j)-ой позиции равна единичному элементу кольца R (мы будем обозначать его обычным символом 1). Матрицы вида $\mathbf{E}_{ij}$ называются *матричными единицами* [1, п. 3.6].

Наряду с матричными единицами $\mathbf{E}_{ij}$ рассмотрим также матрицы вида $\mathbf{T}_{ij}$, определённые следующим образом:

$$\mathbf{T}_{ij} = \sum_{k=1}^{m} \mathbf{E}_{kj} + \sum_{l=1}^{n} \mathbf{E}_{il} - \mathbf{E}_{ij}. \tag{1}$$

Каждая компонента матрицы $\mathbf{T}_{ij}$, находящаяся в её i-ой строке, а также все компоненты этой матрицы, находящиеся в её j-ом столбце, равны элементу $1 \in R$; остальные элементы матрицы $\mathbf{T}_{ij}$ равны 0. Условно:

$$\mathbf{E}_{ij} = \left(\begin{array}{ccccccc} \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \hline & & 0 & & & & \\ \hline \cdots & 0 & 1 & 0 & 0 & \cdots & \\ \hline & & 0 & & & & \\ & & 0 & & & & \\ \hline \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{array}\right); \quad \mathbf{T}_{ij} = \left(\begin{array}{ccccccc} \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \hline & & 1 & & & & \\ \hline \cdots & 1 & 1 & 1 & 1 & \cdots & \\ \hline & & 1 & & & & \\ & & 1 & & & & \\ \hline \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{array}\right).$$

Назовём $\mathbf{T}_{ij}$ *крестовой матрицей с центром* (i,j), или просто *крестовой матрицей*.

Крестовые матрицы интересны хотя бы тем, что *в некоторых* случаях (не во всех!) они образуют ещё один базис модуля $\mathrm{M}_{m\times n}(R)$. Нетрудно, например, доказать следующее утверждение.

Положим по определению

$$\mathbf{Q} \;=\; \sum_{x,y} Q[x,y]\cdot \mathbf{E}_{xy}; \quad \mathbf{Q}^{\#} \;=\; \sum_{x,y} Q[x,y]\cdot \mathbf{T}_{xy} \tag{2}$$

для любой матрицы $\mathbf{Q}\in \mathrm{M}_{m\times n}(R)$. Таким образом, через $Q[x,y]$ мы обозначаем (x,y)-ую компоненту матрицы $\mathbf{Q}$, а через $\mathbf{Q}^{\#}$ обозначаем матрицу, определённую в (2) второй формулой. Итак, всегда верно

$$\mathbf{Q}^{\#} \;=\; \sum_{x=1}^{m}\sum_{y=1}^{n} Q^{\#}[x,y]\cdot \mathbf{E}_{xy}.$$

Теорема 1. *Пусть F – какое угодно поле, имеющее характеристику* $\operatorname{char} F = 2$. *Если оба числа m, n чётные, то*[1]

$$\mathbf{Q} \;=\; \sum_{x=1}^{m}\sum_{y=1}^{n} Q^{\#}[x,y]\cdot \mathbf{T}_{xy} \tag{3}$$

для любой матрицы $\mathbf{Q}\in \mathrm{M}_{m\times n}(F)$.

Доказательство. Непосредственно из определения (2) следует, что $\mathbf{E}_{ij}[x,y]=1$ тогда и только тогда, когда $i=x$, $j=y$; во всех остальных случаях $\mathbf{E}_{ij}[x,y]=0$. Поэтому $\mathbf{E}_{ij}^{\#}=\mathbf{T}_{ij}$; а значит, отображение $\mathbf{Q}\mapsto\mathbf{Q}^{\#}$ линейно в пространстве $\mathrm{M}_{m\times n}(F)$. Отсюда:

$$\mathbf{Q}^{\#\#} \;=\; \Bigl(\sum_{x,y} Q[x,y]\cdot \mathbf{T}_{xy}\Bigr)^{\#} \;=\; \sum_{x,y} Q[x,y]\cdot \mathbf{T}_{xy}^{\#}. \tag{4}$$

Но по условию $\operatorname{char} F = 2$ и $m=n=0$ в поле F. Легко видеть, что в этом случае выполняется $\mathbf{T}_{xy}^{\#}=\mathbf{E}_{xy}$. В самом деле:

$$\begin{aligned}\mathbf{T}_{xy}^{\#} \;&=\; \sum_{k=1}^{m}\mathbf{E}_{ky}^{\#} \;+\; \sum_{l=1}^{n}\mathbf{E}_{xl}^{\#} \;-\; \mathbf{E}_{xy}^{\#} \;=\; \sum_{u=1}^{m}\mathbf{T}_{uy} \;+\; \sum_{v=1}^{n}\mathbf{T}_{xv} \;-\; \mathbf{T}_{xy} \;= \\ &=\; \sum_{k}\left(m\mathbf{E}_{ky}-2\mathbf{E}_{ky}\right) \;+\; \sum_{l}\left(n\mathbf{E}_{xl}-2\mathbf{E}_{xl}\right) \;+\; 2\sum_{k,l}\mathbf{E}_{kl} \;+\; \mathbf{E}_{xy},\end{aligned}$$

согласно определению (1). Так что действительно, $\mathbf{T}_{xy}^{\#}=\mathbf{E}_{xy}$ в условиях теоремы. Стало быть, $\mathbf{Q}^{\#\#}=\mathbf{Q}$ ввиду (4) и (2). То есть $\mathbf{Q}=(\mathbf{Q}^{\#})^{\#}$ – именно это и записано в (3). Тем самым теорема доказана полностью. □

[1] Теорема 1 обобщает теорему 9 из статьи [2].

Из теоремы 1 следует, что *в том случае, когда* $\operatorname{char} F = 2$*, а числа* m *и* n *чётные, крестовые матрицы образуют базис линейного пространства* $\mathrm{M}_{m\times n}(F)$ (поскольку $\dim \mathrm{M}_{m\times n}(F) = mn$, и количество различных крестовых матриц в этом пространстве тоже равно числу mn).

В частности, *если оба числа* m *и* n *являются чётными, то множество всех крестовых* $m \times n$*-матриц над полем* $\mathbb{Z}_2$ *(поле остатков от деления на* 2*) – один из базисов пространства* $\mathrm{M}_{m\times n}(\mathbb{Z}_2)$, а значит, и базис левого модуля $\mathrm{M}_{m\times n}(R)$ над кольцом $R = \mathbb{Z}_2$.

Справедливо более общее утверждение, чем теорема 1. Прежде, чем его сформулировать, примем по определению, что

$$Q[x,\bullet] \,=\, \sum_y Q[x,y]; \quad Q[\bullet,y] \,=\, \sum_x Q[x,y];$$

$$Q[\bullet,\bullet] \,=\, \sum_{x,y} Q[x,y]$$

для любой матрицы $\mathbf{Q} \in \mathrm{M}_{m\times n}(F)$. Кроме того, введём следующие обозначения:

$$\mathbf{E}_{i\bullet} \,=\, \sum_j \mathbf{E}_{ij}; \quad \mathbf{E}_{\bullet j} \,=\, \sum_i \mathbf{E}_{ij}; \quad \mathbf{E}_{\bullet\bullet} \,=\, \sum_{i,j} \mathbf{E}_{ij}.$$

Таким образом:

$$\mathbf{T}_{ij} \,=\, \mathbf{E}_{i\bullet} + \mathbf{E}_{\bullet j} - \mathbf{E}_{ij}. \tag{5}$$

Непосредственно проверяется следующее соотношение:

$$\begin{pmatrix} \mathbf{E}^{\#}_{ij} \\ \mathbf{E}^{\#}_{\bullet j} \\ \mathbf{E}^{\#}_{i\bullet} \\ \mathbf{E}^{\#}_{\bullet\bullet} \end{pmatrix} = \begin{pmatrix} -1 & 1 & 1 & 0 \\ 0 & m-1 & 0 & 1 \\ 0 & 0 & n-1 & 1 \\ 0 & 0 & 0 & m+n-1 \end{pmatrix} \begin{pmatrix} \mathbf{E}_{ij} \\ \mathbf{E}_{\bullet j} \\ \mathbf{E}_{i\bullet} \\ \mathbf{E}_{\bullet\bullet} \end{pmatrix}. \tag{6}$$

Например, в последней строке имеем

$$\mathbf{E}^{\#}_{\bullet\bullet} \,=\, \sum_{i,j} \mathbf{E}^{\#}_{ij} \,=\, \sum_{i,j} \mathbf{T}_{ij} \,=\, m \sum_{k,j} \mathbf{E}_{kj} \,+\, n \sum_{i,l} \mathbf{E}_{il} \,-\, \sum_{i,j} \mathbf{E}_{ij}.$$

Во второй строке:

$$\mathbf{E}^{\#}_{\bullet j} \,=\, \sum_i \mathbf{E}^{\#}_{ij} \,=\, m \sum_k \mathbf{E}_{kj} \,+\, \sum_{i,l} \mathbf{E}_{il} \,-\, \sum_i \mathbf{E}_{ij}.$$

Третья строка проверяется аналогично, а в первой записано не что иное, как полученное ранее утверждение (5).

Теорема 2. *Пусть F – произвольное поле, а числа m и n таковы, что*

$$(m-1)(n-1)(m+n-1) \neq 0 \tag{7}$$

в поле F. Тогда для любой матрицы $\mathbf{Q} \in \mathrm{M}_{m\times n}(F)$ справедливо

$$\mathbf{Q} + \mathbf{Q}^{\#} = \frac{1}{m-1}\sum_{j=1}^{n} Q[\bullet, j]\cdot \mathbf{E}^{\#}_{\bullet j} + \frac{1}{n-1}\sum_{i=1}^{m} Q[i,\bullet]\cdot \mathbf{E}^{\#}_{i\bullet} - \\ - \frac{1}{m+n-1}\left(\frac{1}{m-1}+\frac{1}{n-1}\right) Q[\bullet,\bullet]\cdot \mathbf{E}^{\#}_{\bullet\bullet}$$

Доказательство. Положим $\lambda = m+n-1$. Из (6) легко находим, что

$$\mathbf{E}_{\bullet\bullet} = \frac{\mathbf{E}^{\#}_{\bullet\bullet}}{\lambda}; \quad \mathbf{E}_{\bullet j} = \frac{\mathbf{E}^{\#}_{\bullet j} - \mathbf{E}^{\#}_{\bullet\bullet}/\lambda}{m-1}; \quad \mathbf{E}_{i\bullet} = \frac{\mathbf{E}^{\#}_{i\bullet} - \mathbf{E}^{\#}_{\bullet\bullet}/\lambda}{n-1};$$

$$\mathbf{E}_{ij} + \mathbf{E}^{\#}_{ij} = \mathbf{E}_{\bullet j} + \mathbf{E}_{i\bullet} = \frac{\mathbf{E}^{\#}_{\bullet j}}{m-1} + \frac{\mathbf{E}^{\#}_{i\bullet}}{n-1} - \frac{\mathbf{E}^{\#}_{\bullet\bullet}}{\lambda}\left(\frac{1}{m-1}+\frac{1}{n-1}\right).$$

Домножая последнее равенство на $Q[i,j]$ и суммируя обе его части по всем индексам i и j, мы получаем искомое соотношение, поскольку

$$\mathbf{Q}^{\#} = \sum_{i,j} Q[i,j]\cdot \mathbf{E}^{\#}_{ij}; \qquad \sum_{i,j} Q[i,j]\cdot \mathbf{E}^{\#}_{\bullet j} = \sum_{j=1}^{n} Q[\bullet, j]\cdot \mathbf{E}^{\#}_{\bullet j}$$

и т.д. В поле F можно делить на λ, $m-1$ и $n-1$ ввиду условия (7). $\square$

Таким образом, матрицы $\mathbf{T}_{ij}$ размера $m\times n$ над любым полем, в котором выполняется (7), образуют базис линейного пространства $\mathrm{M}_{m\times n}(F)$.

Один из вопросов, связанных с крестовыми матрицами, состоит в следующем: *сколько различных элементов имеет линейная оболочка всех крестовых матриц $\mathbf{T}_{ij} \in \mathrm{M}_{m\times n}(\mathbb{Z}/d\mathbb{Z})$ для заданных значений m, n и d?* Здесь $\mathbb{Z}/d\mathbb{Z}$ – кольцо вычетов по модулю d.

Иначе говоря, ставится вопрос о нахождении мощности множества

$$\mathrm{H}_{m\times n}(R) = \left\{ \sum_{i,j} \xi_{ij}\mathbf{T}_{ij} \;\middle|\; \xi_{ij} \in R \right\} \tag{8}$$

в том случае, когда кольцо $R = \mathbb{Z}/d\mathbb{Z}$. Элементами этого множества (как видно из его определения) служат всевозможные линейные комбинации крестовых матриц, принадлежащих $\mathrm{H}_{m\times n}(R)$, притом, что $R = \mathbb{Z}/d\mathbb{Z}$.

При небольших m, n и d мощность множества $\mathrm{H}_{m\times n}(\mathbb{Z}/d\mathbb{Z})$ может быть вычислена алгоритмически. Идея состоит в том, чтобы всякой матрице $\mathbf{x} \in \mathrm{M}_{m\times n}(\mathbb{Z}/d\mathbb{Z})$ поставить в соответствие какое-либо уникальное число $f(\mathbf{x}) \in \mathbb{Z}$. То есть *все $m \times n$-матрицы над $\mathbb{Z}/d\mathbb{Z}$ предлагается закодировать теми или иными целыми числами с соблюдением условия*

$$f(\mathbf{x}_1) = f(\mathbf{x}_2) \;\Rightarrow\; \mathbf{x}_1 = \mathbf{x}_2 \quad \text{для всех } \mathbf{x}_1, \mathbf{x}_2 \in \mathrm{M}_{m\times n}(R). \tag{9}$$

Тогда, если диапазон возможных значений для $f(\mathbf{x})$ будет известен, то, перебирая все числа из этого диапазона, мы сможем осуществить полный перебор элементов множества $\mathrm{M}_{m\times n}(\mathbb{Z}/d\mathbb{Z})$ и всякой матрице $\mathbf{x}$ из этого множества поставить в соответствие матрицу $\mathbf{x}^{\#}$, заведомо принадлежащую $\mathrm{H}_{m\times n}(R)$. В процессе такого перебора гарантированно будут получены все элементы множества $\mathrm{H}_{m\times n}(R)$ в виде матриц $\mathbf{x}^{\#}$.

А далее, чтобы алгоритм не просто строил матрицы, принадлежащие множеству $\mathrm{H}_{m\times n}(\mathbb{Z}/d\mathbb{Z})$, а находил *мощность* этого множества – то есть подсчитывал количество *различных* матриц вида $\mathbf{x}^{\#}$, входящих в данное множество – в программу должен быть добавлен механизм, который бы отсекал повторяющиеся матрицы $\mathbf{x}^{\#}$, полученные в процессе работы алгоритма. То есть, если при выполнении перебора вдруг получилась такая матрица $\mathbf{x}^{\#}$, которая уже была получена на более ранних стадиях работы программы, то для данной матрицы должен сработать *механизм отсечения*: алгоритм должен узнать о том, что $\mathbf{x}^{\#}$ совпала с одной из ранее полученных матриц, и перейти к следующей итерации перебора.

При наличии такого механизма, выполняющего отсечения повторяющихся матриц, мощность множества $\mathrm{H}_{m\times n}(R)$ будет, очевидно, совпадать с количеством тех матриц $\mathbf{x}^{\#}$, которые не пришлось отсекать. Условие (9) позволяет достаточно просто реализовать данный механизм: достаточно, чтобы алгоритм помнил коды $f(\mathbf{x}^{\#})$ всех ранее полученных матриц $\mathbf{x}^{\#}$. И тогда на каждой итерации после получения очередной новой матрицы её код сравнивается с кодами всех ранее полученных матриц: если такой код уже был получен хотя бы один раз, то это означает, что новая матрица совпала с какой-то из уже полученных матриц, и её необходимо отсечь. В противном случае матрица ещё не встречалась в процессе работы алгоритма – тогда необходимо запомнить её код (т.е. значение f для этой матрицы; саму матрицу запоминать при этом не обязательно), и можно будет переходить к следующей итерации перебора матриц $\mathbf{x}$.

В конечном итоге для вычисления мощности множества $\mathrm{H}_{m\times n}(R)$ в программу можно добавить счётчик, начальное значение которого рав-

но 0: на каждой итерации значение этого счётчика увеличивается на 1, кроме тех случаев, когда срабатывает механизм отсечения матриц. После окончания работы алгоритма счётчик будет содержать значение, равное количеству элементов множества $\mathrm{H}_{m\times n}(\mathbb{Z}/d\mathbb{Z})$. Так в общих чертах выглядит алгоритм, способный для заданных значений d, m и n находить количество всевозможных $m\times n$-матриц, представимых в виде линейной комбинации крестовых $m\times n$-матриц над кольцом $\mathbb{Z}/d\mathbb{Z}$.

Поскольку мы собираемся далее представить результат работы описанного выше алгоритма, то нам бы хотелось сперва дать более формальное его описание – такое, которое сводило бы к минимуму риск возникновения ошибок при его реализации в виде компьютерной программы.

Для любого подмножества $X\subseteq \mathrm{M}_{m\times n}(R)$ положим по определению, что

$$X^{\#} \;=\; \left\{\mathbf{Q}^{\#} \,\middle|\, \mathbf{Q}\in X\right\}.$$

Тогда понятно, что

$$\mathrm{H}_{m\times n}(R) \;=\; X^{\#} \quad \text{при } X=\mathrm{M}_{m\times n}(R),$$

каким бы ни было кольцо R. Следовательно, для мощностей справедливо

$$\left|\mathrm{H}_{m\times n}(R)\right| \;=\; \left|X^{\#}\right| \;=\; \left|f\left(X^{\#}\right)\right| \quad \text{при } X=\mathrm{M}_{m\times n}(R) \tag{10}$$

ввиду условия (9).

Зафиксируем теперь значения d, m, n и рассмотрим конкретное кольцо $R=\mathbb{Z}/d\mathbb{Z}$. Пусть $Y\subset\mathbb{Z}$ – какое-то конечное множество целых чисел, о котором достоверно известно, что при наших выбранных параметрах d, m и n соотношение $f(\mathbf{x})\in Y$ выполняется для всех матриц $\mathbf{x}\in\mathrm{M}_{m\times n}(R)$. Другими словами: Y – некое множество, из которого выбираются коды для $m\times n$-матриц над R; это не обязательно множество всех кодов матриц. Тогда мы можем утверждать, что $X=f^{-1}(Y)$ в формуле (10). Здесь f^{-1} – частичное отображение[2], которое восстанавливает матрицу $\mathbf{x}$ по её коду $f(\mathbf{x})$:

$$f^{-1}:\quad \mathrm{dom}\left(f^{-1}\right) \;\to\; \mathrm{M}_{m\times n}(R):$$
$$f^{-1}(y)=\mathbf{x},\ \text{если } y=f(\mathbf{x});\ \text{иначе } f^{-1}(y) \text{ не определено.}$$

При этом важно, что область определения $\mathrm{dom}\left(f^{-1}\right)\subseteq Y$.

[2]Напомним, что *частичным отображением* множества A в множество B называется [3] какое-либо отображение $\varphi: A'\to B$ при условии, что $A'\subseteq A$. Множество A' называется при этом *областью определения* частичного отображения φ; оно обозначается следующим образом: $A'=\mathrm{dom}\varphi$. Полагаем $\varphi(Y)=\{\varphi(y)\,|\,y\in Y\cap\mathrm{dom}\varphi\}$.

Предложение 3. *Пусть R – ассоциативное кольцо с единицей, f – произвольное вложение $\mathrm{M}_{m\times n}(R)\to Y$, а множество $Y\subseteq\mathbb{Z}$. Тогда*[3]

$$|\mathrm{H}_{m\times n}(R)| \;=\; \mathrm{Card}\left\{ \left(yf^{-1}\right)^{\#} f \;\middle|\; y\in\left[Y\cap\mathrm{dom}\left(f^{-1}\right)\right]\right\}. \qquad (11)$$

Доказательство. Полагая $X=Yf^{-1}$, находим, что $X^{\#}f=\{\mathbf{x}^{\#}f\,|\,\mathbf{x}\in Yf^{-1}\}$. Но любая матрица $\mathbf{x}\in Yf^{-1}$ представима в виде $\mathbf{x}=yf^{-1}$ для некоторого $y\in\mathrm{dom}\,(f^{-1})$. Причём $y\in Y$ ввиду того, что $\mathrm{dom}\,(f^{-1})\subseteq Y$ по условию. Но в таком случае из (10) немедленно следует (11). $\square$

Предложение 3 довольно детально отражает работу алгоритма, пригодного для вычисления значения $|\mathrm{H}_{m\times n}(R)|$ в случае $R=\mathbb{Z}/d\mathbb{Z}$: *метод состоит в том, чтобы прежде всего выбрать подходящим образом вложение f, кодирующее матрицы, а затем подсчитать количество элементов, принадлежащих множеству, указанному в правой части равенства* (11). Это делается непосредственным образом[4] – путём полного перебора элементов множества Y, зависящего конечно же от f.

В качестве f вполне естественно взять функцию, которая позволила бы осуществить хранение компонент любой матрицы $\mathbf{x}\in\mathrm{M}_{m\times n}(\mathbb{Z}/d\mathbb{Z})$ в отдельных битах целочисленных переменных, расположенных в памяти вычислительного устройства. Чтобы получить аналитическое выражение для такой функции, рассмотрим компоненты матрицы $\mathbf{x}$ как цифры, участвующие в записи целых чисел в k-ичной системе исчисления:

$$f(\mathbf{x}) \;=\; \sum_{i=1}^{m}\sum_{j=1}^{n} x[i,j]\cdot k^{(i-1)n+(j-1)} \quad \textit{при условии, что } x[i,j]\in\mathbb{Z}. \qquad (12)$$

При этом основание k может быть выбрано произвольно с соблюдением единственного требования: должно быть выполнено условие $d\le k$.

[3]Здесь запись $\mathrm{Card}\,A$ обозначает то же, что $|A|$ для любого множества A. Для большей наглядности вместо $f(a)$ и $f^{-1}(a)$ мы пишем, соответственно, af и af^{-1}. Квадратными скобками в (11) и далее в (15) мы пользуемся наравне с круглыми скобками, не вкладывая в данное обозначение какого-либо дополнительного смысла.

[4]Составляется битовая карта, в которой каждый бит взаимно однозначно соответствует тому или иному числу из множества Y. Все биты изначально имеют нулевые значения. Как только в процессе перебора встречается число $\left(yf^{-1}\right)^{\#}f$, оно отмечается в битовой карте. Но перед этим алгоритм смотрит, не было ли данное число уже отмечено ранее. Если было, то выполняется обычный переход на следующую итерацию, а если не было, то счётчик (начальное значение которого устанавливается в 0) увеличивается на 1. После окончания перебора данный счётчик будет содержать количество элементов того множества, для которого была составлена битовая карта.

Подчеркнём лишний раз, что вычисления по формуле (12) производятся в кольце целых чисел, а не в кольце вычетов. При этом, поскольку мы условились, что $d \le k$, то $x[i,j] < k$ для всех индексов i, j; следовательно, отображение (12) осуществляет вложение

$$\mathrm{M}_{m\times n}(\mathbb{Z}/d\mathbb{Z}) \to Y, \quad \text{где } Y = \{0, 1, ..., k^{mn} - 1\}$$

и тем самым подпадает под условия предложения 3. Однако, для построения окончательного алгоритма нам необходимо ещё получить аналитическое выражение для f^{-1} – обратного отображения к f из (12).

Интуитивно понятно, что в качестве k удобно выбирать какую-либо степень числа 2: потому что в этом случае не только упрощаются формулы, но и вычисления по ним становятся более эффективными. Следующее утверждение подтверждает данную догадку строгим обоснованием:

Предложение 4. *Пусть $k = 2^c$, где c – целое положительное число. Каким бы ни было целое положительное $d \le k$, отображение*

$$f: \quad \mathrm{M}_{m\times n}(\mathbb{Z}/d\mathbb{Z}) \;\to\; \{0, 1, ..., 2^{c\,m\,n} - 1\},$$

определённое соотношением (12) для всякой матрицы $\mathbf{x} \in \mathrm{M}_{m\times n}(\mathbb{Z}/d\mathbb{Z})$, имеет обратное ч а с т и ч н о е о т о б р а ж е н и е f^{-1}, удовлетворяющее следующему условию[5] при любом выборе $y \in \mathrm{dom}(f^{-1})$:

$$\left(yf^{-1}\right)^{\#} f \;=\; \sum_{i=1}^{m}\sum_{j=1}^{n} x_{ij}^{\#} \ll (h_{ij}\cdot c), \tag{13}$$

$$\text{где} \quad h_{ij} \;=\; (i-1)n + (j-1), \quad x_{ij}^{\#} \;=\; x^{\#}[i,j], \tag{14}$$

$$x[i,j] \;=\; [y \,\&\, ((2^c - 1) \ll (h_{ij}\cdot c))] \gg [h_{ij}\cdot c]. \tag{15}$$

Здесь $a \,\&\, b$ – обозначение для операции побитового умножения беззнаковых a и b, а через $a \ll b$ и $a \gg b$ обозначены стандартные операции логического сдвига числа a влево и, соответственно, вправо на b бит.

Замечание. Величины $x[i,j]$ в формулах (13) – (15) – это не что иное, как h_{ij}-ые цифры числа y, записанного в k-ичной системе исчисления.

Доказательство. Учитывая замечание, легко разглядеть, что $\mathbf{x} = yf^{-1}$ при $y \in \mathrm{dom}(f^{-1})$. После чего ясно, что в правой части равенства (13) находится не что иное, как число $(\mathbf{x}^{\#})f$, согласно определению (12). $\square$

[5]Мы вновь здесь пишем af и af^{-1} вместо соответственно $f(a)$ и $f^{-1}(a)$, как в формуле (11), чтобы не загромождать выражения излишним количеством скобок.

Итак, формула (13) даёт эффективный метод расчёта величин вида $(yf^{-1})^{\#}\, f$, находящихся внутри формулы (11). О том, как вычислять, используя (11), мощность линейной оболочки множества крестовых матриц, мы рассказали весьма подробно[4] несколькими абзацами ранее.

Настала пора представить *результат работы* алгоритма[6], основанного на предложениях 3 и 4. При ограничении объёма используемой памяти примерным значением 512 Мбайт были получены мощности множеств $\mathrm{H}_{m\times n}(\mathbb{Z}/d\mathbb{Z})$ для следующих значений параметров d, m и n:

$d = 4$	$d = 8$
2×2: *свободный модуль*	2×2: *свободный модуль*
2×3: $\lvert H\rvert = 4^5$	2×3: $\lvert H\rvert = 8^5$
2×4: *свободный модуль*	2×4: *свободный модуль*
2×5: $\lvert H\rvert = 4^9$	2×5: $\lvert H\rvert = 8^9$
2×6: *свободный модуль*	3×3: $\lvert H\rvert = 4\cdot 8^7$
2×7: $\lvert H\rvert = 4^{13}$	
3×3: $\lvert H\rvert = 4^7$	$d = 9$
3×4: $\lvert H\rvert = 4^{10}$	2×2: $\lvert H\rvert = 3\cdot 9^3$
3×5: $\lvert H\rvert = 4^{11}$	2×3: *свободный модуль*
4×4: *свободный модуль*	2×4: $\lvert H\rvert = 3\cdot 9^7$
$d = 6$	$d = 10$
2×2: $\lvert H\rvert = 2\cdot 6^3$	2×2: *свободный модуль*
2×3: $\lvert H\rvert = 3\cdot 6^5$	2×3: $\lvert H\rvert = 5\cdot 10^5$
2×4: $\lvert H\rvert = 2\cdot 6^7$	2×4: $\lvert H\rvert = 2\cdot 10^7$
2×5: $\lvert H\rvert = 6^9$	
3×3: $\lvert H\rvert = 3^4\cdot 6^5$	

Если мощности множеств $\mathrm{H}_{m\times n}(\mathbb{Z}/d\mathbb{Z})$ и $\mathrm{M}_{m\times n}(\mathbb{Z}/d\mathbb{Z})$ совпадают, то

[6]Автором была выполнена реализация данного алгоритма в виде компьютерной программы, составленной строго в соответствии со стандартом ISO/IEC 9899:1999 «Programming languages — C». В настоящей работе после списка литературы содержится полный программный код, способный воспроизвести все результаты, представленные в таблице. Для корректной работы программы платформа, на которой осуществляется её запуск, должна быть в состоянии предоставить программе достаточный объём памяти для хранения битовых карт, используемых в процессе её работы.

любая $m \times n-$матрица над $\mathbb{Z}/d\mathbb{Z}$ представима в виде линейной комбинации крестовых матриц, и в таком случае модуль $\mathrm{H}_{m\times n}(\mathbb{Z}/d\mathbb{Z})$ свободный.

Модуль $\mathrm{H}_{m\times n}(\mathbb{Z}/d\mathbb{Z})$ заведомо *несвободен*, если количество элементов в нём не является степенью числа d; из таблицы отчётливо видно, что *над кольцами вычетов существуют несвободные модули, порождённые крестовыми матрицами.* Самый маленький из них – это модуль $\mathrm{H}_{2\times 2}(\mathbb{Z}/6\mathbb{Z})$, содержащий 432 элемента.

В таблице нет информации о модулях над кольцами вида $\mathbb{Z}/p\mathbb{Z}$, где p – простое число: потому что каждое такое кольцо является полем и, следовательно, модули над ними представляют собой линейные пространства; то есть эти модули обязательно являются свободными.

Было бы интересно для произвольного (ассоциативного) кольца R выяснить, всегда ли его модуль $\mathrm{H}_{m\times n}(R)$ является свободным при условии, что характеристика этого кольца $\mathrm{char}\, R$ – простое число. Но данный вопрос находится за рамками настоящей работы.

Список литературы

[1] *Туганбаев А. А.* Теория колец. Арифметические модули и кольца. М.: МЦНМО, 2009. 472 с.

[2] *Решетников А. В., Манилов Д. Ю.* О крестовых матрицах над полем $\mathbb{Z}_2$. [2026, электронный ресурс: PREPRINTS.RU] `https://doi.org/10.24108/preprints-3114494`

[3] *Ляпин Е.С., Евсеев А. Е.* Частичные алгебраические действия. – С.-Петербург, Росс. гос. пед. ун-т им. А. И. Герцена: Образование, 1991. – 163 с.

```c
// The following program calculates the power of H[m x n](Z/dZ).
// This code is distributed under the terms of MIT license.

// ---------------------
// General Configuration
// ---------------------

// The limits for m, n, and d:
#define CONFIG_D_MIN 4
#define CONFIG_D_MAX 10
#define CONFIG_M_MIN 2
#define CONFIG_M_MAX 10
#define CONFIG_N_MAX 10

// Memory limit for usage (16 = 64kB, 20 = 1Mb etc.):
#define CONFIG_MEMLIM_POWER 29

// Do not change the following compiler variable unless you know
// what you are doing:
#define DEBUG 0

// ============================================================================

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <limits.h>

// ----------------------------------------------------------------------------

// The following function prints the components of a given matrix
// to the standard output stream without tracking any input/output
// errors. The function is intended for debugging purposes only:

void print_matrix(unsigned* a, unsigned m, unsigned n, unsigned d)
{   unsigned i, j;
    for (i = m; i-- > 0; )
    {   for (j = n; j-- > 0; ) printf("%u", a[i*n+j] % d);
        printf(" ");
    }
}

// ----------------------------------------------------------------------------

unsigned long power(unsigned m, unsigned n, unsigned d)
{   const size_t memlim = 1ul << CONFIG_MEMLIM_POWER;

    static size_t bufsize = 0;
    static char* buffer = NULL;
    static unsigned* a = NULL;
    static unsigned* coeff = NULL;

    unsigned c, pow;
    unsigned long k, cnt;

    if (m >= 8 || n >= 8 || d >= 256) return 0;
```

```
 57     cnt = 0;
 58
 59     // Allocate a memory buffer:
 60     for (c = 0; d > (1u<<c); c++);
 61     fprintf(stderr, "m=%u, n=%u, d=%u, c=%u: ", m, n, d, c);
 62     bufsize = (1ul << m*n*c) / CHAR_BIT + 1;
 63     fprintf(stderr, "%lu %s ",
 64      bufsize < 1024 ? bufsize : (unsigned long)bufsize >> 10,
 65      bufsize < 1024 ? "bytes" : "kb");
 66     if (bufsize > memlim || bufsize == 0) goto error_limit;
 67     if ((buffer = malloc(bufsize)) == NULL) goto error_memory;
 68     if ((a = malloc(sizeof(a[0]) * m*n)) == NULL) goto error_memory;
 69     if ((coeff = malloc(sizeof(coeff[0]) * m*n)) == NULL)
 70      goto error_memory;
 71     memset(buffer, 0, bufsize);
 72
 73     pow = m*n*c;
 74     if (pow >= sizeof(k) * CHAR_BIT) goto error_special;
 75     for (k = 0; k < (1ul << pow); k++)
 76     {   unsigned i, j, mask;
 77         unsigned long index;
 78         size_t offset;
 79         char mark;
 80
 81 // ----------------------------------------
 82 if (k % 10000000 == 0) fprintf(stderr, ".");
 83 // ----------------------------------------
 84
 85         // To set the coefficients of cross matrices:
 86         for (i = 0; i < m; i++)
 87         {   for (j = 0; j < n; j++)
 88             {   unsigned long mask = (1u<<c) - 1;
 89                 unsigned pos = (i*n+j)*c;
 90                 coeff[i*n+j] = (k & (mask << pos)) >> pos;
 91                 if (coeff[i*n+j] >= d) break;
 92                 a[i*n+j] = 0;
 93             }
 94             if (j < n) break;
 95         }
 96         if (i < m) continue;
 97
 98         // To calculate the sum of (coeff[i,j] * T[i,j]):
 99         for (i = 0; i < m; i++) for (j = 0; j < n; j++)
100         {   for (unsigned x = 0; x < m; x++) a[x*n+j] += coeff[i*n+j];
101             for (unsigned y = 0; y < n; y++) a[i*n+y] += coeff[i*n+j];
102             a[i*n+j] -= coeff[i*n+j];
103         }
104
105         // To mark the element which corresponds to the matrix "a":
106         index = 0;
107         for (i = 0; i < m; i++) for (j = 0; j < n; j++)
108          index += ((unsigned long)(a[i*n+j] % d) << (i*n+j)*c);
109         offset = index / CHAR_BIT;
110         if (offset >= bufsize) abort();
111         mask = 1u << (index % CHAR_BIT);
112         mark = buffer[offset] & mask;
```

```
113 // ------------------------------------------------
114 #if DEBUG
115 printf("k=%lu: ", k); print_matrix(coeff,m,n,d);
116 printf("  ->   "); print_matrix(a,m,n,d);
117 printf("index=%lu%s\n", index, mark ? " (v)" : "");
118 #endif
119 // ------------------------------------------------
120         if (!mark)
121         {   cnt++;
122             buffer[offset] |= mask;
123         }
124     }
125
126 // --------------------
127 fprintf(stderr, "\n");
128 // --------------------
129
130 exit:
131     free(buffer); buffer = NULL; bufsize = 0;
132     free(a); a = NULL;
133     free(coeff); coeff = NULL;
134     return cnt;
135
136 error_limit:
137     fprintf(stderr, "\nMemory limit exceeded: %zu kb\n", memlim >> 10);
138     goto exit;
139
140 error_memory:
141     fprintf(stderr, "\nNot enough memory\n");
142     goto exit;
143
144 error_special:
145     fprintf(stderr, "\nm*n*c = %u: too much...\n", pow);
146     goto exit;
147 }
148
149 // ---------------------------------------------------------------------------
150
151 // ********************
152 // * The Entry Point *
153 // ********************
154
155 int main()
156 {   unsigned m, n, d, n_min, m_max, m_min, d_max, pow;
157     unsigned long cnt, a;
158     d_max = CONFIG_D_MAX;
159     for (d = CONFIG_D_MIN; d <= d_max; d++)
160     {   if (d == 2 || d == 3 || d == 5 || d == 7) continue;
161         m_min = CONFIG_M_MIN;
162         m_max = CONFIG_M_MAX;
163         for (m = m_min; m <= m_max; m++)
164         {   //if (m % d == 1) continue;
165             n_min = m;
166             for (n = n_min; n <= CONFIG_N_MAX; n++)
167             {   //if (n % d == 1 || (m+n) % d == 1) continue;
168                 if (m % d == 1 && n % d == 1) continue;
```

```
169                 cnt = power(m,n,d);
170                 for (pow = 0, a = 1; a*d <= cnt; pow++) a *= d;
171                 if (cnt > 0)
172                 {   printf("%2u x %-2u, mod %-2u: ", m, n, d);
173                     if (a == cnt)
174                     {   if (pow == m*n) printf("(free)");
175                         else printf("  |H| = %u^%u", d, pow);
176                         printf("\n");
177                     }
178                     else
179                     {   printf("  |H| = ");
180                         if (cnt % a == 0)
181                          printf("%lu * %u^%u", cnt/a, d, pow);
182                         else printf("%lu", cnt);
183                         printf(":  non-free\n");
184                     }
185                     //printf("\n");
186                 }
187                 else
188                 {   if (n == n_min)
189                     {   if (m == m_min) d = d_max;
190                         m = m_max;
191                     }
192                     break;
193                 }
194                 fflush(stdout);
195             }
196         }
197         printf("\n");
198     }
199     return 0;
200 }
201
202 // ========================================================================
203
204 // by Reshetnikov Artyom <a_reshetnikov@hush.com>
205 // October 10, 2025
206
```