УДК 51.681.3

С.Л. Кривий, В.Т. Антонюк

Реалізація алгоритму розв'язання системи лінійних діофантових рівнянь в кільці лишків

Проведен анализ алгоритмов построения базиса множества решений системы линейных однородных и неоднородных диофантовых уравнений над кольцами и полями вычетов по модулю составного и простого числа. Рассмотрены алгоритмы, в основу которых положен *TSS*-метод. Разработаны и реализованы алгоритмы в виде кросс-платформенного программного приложения, которые позволяют эффективно решать описанную задачу.

Ключевые слова: кольцо вычетов, поле вычетов, линейные диофантовые уравнения, базис множества решений, программная реализация.

Проведено аналіз алгоритмів побудови базису множини розв'язків системи лінійних однорідних та неоднорідних діофантових рівнянь над кільцями та полями лишків за модулем складеного і простого числа. Розглянуто алгоритми, в основу яких покладено *TSS*-метод. Розроблено та реалізовано алгоритми у вигляді крос-платформного програмного застосування, які дозволяють ефективно розв'язувати описану задачу.

Ключові слова: кільце лишків, поле лишків, лінійні діофантові рівняння, базис множини розв'язків, програмна реалізація.

Вступ. Пошук розв'язків системи лінійних діофантових рівнянь є однією з актуальних та необхідних задач алгебри, теорії чисел, криптографії, теорії математичних ігор та ін. Прикладами задач з перерахованих галузей, розв'язання яких зводиться до розв'язання системи лінійних діофантових рівнянь в кільці або полі лишків за модулем деякого складеного чи простого числа, є задачі про математичний сейф [1], розпізнавання зображень (про мозаїку) [2], розподілу пам'яті в паралельних обчисленнях.

Аналіз властивостей математичних моделей таких задач зводиться до пошуку базису множини всіх розв'язань системи лінійних однорідних або неоднорідних діофантових рівнянь в кільці або в полі лишків за модулем складеного або простого числа. В даній статті запропоновано алгоритм побудови базису множини всіх розв'язань таких систем і його програмна реалізація на мові *C*++. Подано спочатку необхідні відомості про основні поняття та визначення з коротким оглядом існуючих методів розв'язання. Здійснено аналіз алгоритму, опис його реалізації та обмеження даної реалізації.

Нагадаємо, що кільце лишків Z_m називається *примарним*, якщо модуль m є степенем простого числа p, тобто $m = p^t$, де $t > 1, t \in N$. Оскільки m не є простим числом, то порівняння $ax \equiv b \pmod m$ в кільці Z_m при $a \neq 0$ не завжди матиме розв'язок. Воно матиме розв'язок, якщо НСД(a, m)=1 або НСД(a, m)=d і d – дільник числа b.

Відомо, що коли модуль є простим числом, то кільце лишків за даним модулем буде полем, яке позначається F_p. В полі операція взяття оберненого елемента до даного повністю визначена.

Постановка задачі

Системою лінійних діофантових рівнянь (СЛДР) над кільцем Z_m називається система

$$S = \begin{cases} L_1(x) = a_{11}x_1 + \ldots + a_{1q}x_q = b_1 \\ L_2(x) = a_{21}x_1 + \ldots + a_{2q}x_q = b_2 \\ \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \\ L_s(x) = a_{s1}x_1 + \ldots + a_{sq}x_q = b_s \end{cases} (\mathrm{mod}\, m),$$

де $a_{ij}, b_i, x_j \in Z_m$, $i = 1, \ldots, s$, $j = 1, \ldots, q$.

Нехай $x = (x_1, x_2, \ldots, x_q)$. Розв'язком СЛДР S називається такий вектор $c = (c_1, c_2, \ldots, c_q)$, при

підстановці якого замість x в $L_i(x)$ отримуємо рівність $L_i(c) = b_i \pmod m$ для всіх $i = 1, 2, \ldots, s$. СЛДР називається однорідною (СЛОДР), якщо всі b_i дорівнюють нулю, в іншому випадку СЛДР називається неоднорідною (СЛНДР).

Базисом множини всіх розв'язків СЛДР S називається така множина векторів $B = \{e_1, e_2, \ldots, e_k\}$, для якої будь-який розв'язок S можна подати у вигляді невід'ємної лінійної комбінації векторів із B.

Задачею, що розв'язується в даній статті, є пошук базису множини всіх розв'язків СЛДР S в кільці лишків за модулем m.

Аналіз алгоритмів

У випадку розв'язання задачі в полі F_p використовується відомий *метод Гауса* [3] (метод послідовного виключення невідомих). В даному методі шляхом елементарних перетворень над рядками необхідно привести матрицю до одиничного вигляду. Для виконання цієї операції необхідно вміти знаходити значення оберненого елемента. Саме тому цей метод можна застосовувати для розв'язання задачі в полі F_p (оскільки в полі завжди існує обернений елемент). Проте метод Гауса стає незастосовним у випадку кільця за модулем складеного числа.

Другий метод ґрунтується на алгоритмі [4], в якому розширена матриця системи $(A|b)$ зводиться до ступеневого вигляду $(A'|b')$ і відшукується права матриця переходу R така, що $(A|b) \times R = (A'|b')$. На основі матриці R і можна отримати розв'язок системи. Проте, при розв'язанні системи лінійних рівнянь в кільці лишків спостерігається експоненційний ріст норм коефіцієнтів. Ще одним вагомим недоліком даного методу є те, що він дозволяє знайти лише один розв'язок системи, коли цікавить пошук базису множини всіх розв'язків.

В результаті було обрано алгоритм, в основу якого покладено *TSS*-метод [5] з характеристиками, що найкраще підходять для розв'язання поставленої задачі. В основі алгоритму – ідея розбиття заданої системи на підсистеми в полях та примарних кільцях лишків за модулями, які є числами в розкладі модуля на прості множники. В процесі роботи алгоритму не виникає проблем з експоненційним ростом норм коефіцієнтів, оскільки всі операції відбуваються у відповідному полі або кільці. Окрім цього, даний алгоритм дозволяє за поліноміальний час знаходити саме базис всіх розв'язків заданої СЛДР.

***TSS*-метод**

Наведемо опис *TSS*-методу без доведення всіх поданих тверджень [5–7].

Нехай модуль m має такий розклад на прості множники $m = p_1^{k_1} p_2^{k_2} \ldots p_r^{k_r}$, де $p_1 < p_2 < \ldots < p_r$. Тоді за системою S побудуємо систему з $r \cdot s$ рівняннями наступного вигляду:

$$
\begin{cases}
\begin{cases}
L_1(x) = a_{11}x_1 + \ldots + a_{1q}x_q - b_1x_0 = 0 \\
L_2(x) = a_{21}x_1 + \ldots + a_{2q}x_q - b_1x_0 = 0 \\
\ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \\
L_s(x) = a_{s1}x_1 + \ldots + a_{sq}x_q - b_sx_0 = 0
\end{cases} \pmod{p_1^{k_1}} \\
\begin{cases}
L_1(x) = a_{11}x_1 + \ldots + a_{1q}x_q - b_1x_0 = 0 \\
L_2(x) = a_{21}x_1 + \ldots + a_{2q}x_q - b_1x_0 = 0 \\
\ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \\
L_s(x) = a_{s1}x_1 + \ldots + a_{sq}x_q - b_sx_0 = 0
\end{cases} \pmod{p_2^{k_2}} \\
\ldots \\
\begin{cases}
L_1(x) = a_{11}x_1 + \ldots + a_{1q}x_q - b_1x_0 = 0 \\
L_2(x) = a_{21}x_1 + \ldots + a_{2q}x_q - b_1x_0 = 0 \\
\ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \\
L_s(x) = a_{s1}x_1 + \ldots + a_{sq}x_q - b_sx_0 = 0
\end{cases} \pmod{p_r^{k_r}}
\end{cases}.
$$

Теорема. Системи S та S' еквівалентні, тобто кожен розв'язок системи S є розв'язком системи S' та навпаки.

З результатів роботи [5] випливає така процедура побудови базису множини розв'язків СЛОДР S в кільці Z_m: початкова СЛОДР S розбивається на r підсистем $S_1, S_2, \ldots, S_r$ за модулями $p_1^{k_1}, p_2^{k_2}, \ldots, p_r^{k_r}$ відповідно, кожна з

цих підсистем розв'язується окремо. В результаті знаходимо базиси $B_1, B_2, \ldots, B_r$, відповідно для $S_1, S_2, \ldots, S_r$, а потім будується базис $B = m/p_1^{k_1} B_1 \bigcup \ldots \bigcup m/p_r^{k_r} B_r$, де $m/p_i^{k_i} B_i$ означає множення кожного вектору із B_i на число $m/p_i^{k_i}$.

Теорема. Множина B, побудована *TSS*-методом, є базисом множини розв'язків СЛОДР *S*. Арифметична складність побудови B пропорційна величині $O\left(l^6\right)$, де $l = \max\left(n, s, q, r\right)$, n – максимальна розрядність чисел [5, 6].

Так побудова базису множини розв'язків СЛНДР зводиться до побудови базису множини розв'язків розширених СЛОДР в полі лишків або примарному кільці лишків. Будемо називати розширену СЛОДР S' такою, яка відповідає СЛНДР S і яка будується шляхом введення додаткової невідомої величини x_0 і стовпця коефіцієнтів при цьому невідомому, який складається з вільних членів СЛНДР. Якщо базис множини розв'язків розширеної СЛОДР знайдено, то виділимо в ньому розв'язки $x_1, x_2, \ldots, x_t$, у яких остання координата відмінна від нуля. Нехай $d_1, d_2, \ldots, d_t$ – значення цих координат. Складемо рівняння

$$c_1 d_1 + c_2 d_2 + \ldots + c_t d_t + \equiv 1 \left(\mathrm{mod}\, m\right). \qquad (1)$$

Якщо це порівняння має розв'язок $(u_1, u_2, \ldots, u_t)$, то СЛНДР сумісна і лінійна комбінація

$$x^1 = u_1 x_1 + u_2 x_2 + \ldots + u_t x_t$$

є окремим розв'язком СЛНДР, а решта векторів із базису СЛОДР будуть розв'язками СЛОДР відповідно даній СЛНДР.

Зауважимо, що порівняння (1) матиме розв'язок лише у випадку $\mathrm{НСД}\left(d_1, d_2, \ldots d_t\right) = 1$.

Детальніше ознайомитися з описаними твердженнями, а також з методами розв'язання СЛОДР в полях та примарних кільцях можна у [5–7].

TSS-алгоритм

Для опису цього алгоритму та його програмної реалізації скористаємося його псевдокодом.

Головною функцією програми реалізації *TSS*-алгоритму є *SOLVE*, яка отримує дані про початкову СЛНДР та повертає результат пошуку базису множини всіх розв'язків цієї початкової СЛНДР. Точніше, ця функція перетворює отриману систему в СЛОДР, використовуючи допоміжну функцію розв'язання однорідних систем, отримує її розв'язок та перетворює його у розв'язок початкової СЛНДР.

```
   function SOLVE(coefficients, modulo)
   ввід: coefficients – двомірний масив коефіцієнтів системи рівнянь (останній елемент кожного рядка задає вільний член кожного рівняння), modulo – модуль.
   вивід: базис множини всіх розв'язків заданої системи.

   rows ← len(coefficients) // кількість рівнянь в заданій системі
   columns ← len(coefficients[1]) - 1 // кількість змінних в заданій системі
   is_homogeneous ← True

   for i = 1..rows do
       if coefficients[i][columns + 1] != 0 // вільний член i-го рівняння
           is_homogeneous ← False
       coefficients[i][columns + 1] = modulo - coefficients[i][columns]

   result ← SOLVE_HOMOGENEOUS_SYSTEM (coefficients, modulo)
   // result – двовимірний масив, де кожен рядок задає один із
   // векторів базису множини всіх розв'язків

   if is_homogeneous
       for i = 1..len(result) do
           remove_last_element(result[i]) // видалити останній елемент i-го вектора
       return result // для однорідної системи повертаємо тільки базис

   // у випадку неоднорідної системи, необхідно повернути
   // частковий розв'язок та базис
   coeffs ← {} // присвоїти порожній масив
   temp ← {}
   for vector in result do
       if vector[columns + 1] != 0 then
   // додати останній елемент з vector в масив coeffs
           coeffs add vector[columns + 1]
             // додати vector в масив temp
           temp add vector

   if len(temp) == 0 then // масив temp порожній
```

```
        return NO_SOLUTION

    result ← temp // присвоїти масиву result масив temp
    u, solved ← MOD_SOLVER(coeffs, modulo)

    if not solved then
        return NO_SOLUTION

    for i = 1..len(result) do
        remove_last_element(result[i])    // видалити останній елемент з result[i]

    x1 ← ∑ u[k] × result[k] for all k = 1..len(u) // поелементна сума векторів

    return x1, result
```

Функція *SOLVE_HOMOGENEOUS_SYSTEM* знаходить базис множини розв'язків СЛОДР. Вхідна СЛОДР розбивається на множину підсистем, кожна з яких розглядається над кільцем або полем лишків за модулем відповідного множника факторизації вхідного модуля m. Далі використовуються допоміжні функції розв'язання СЛОДР у полях та кільцях та комбінування їх результатів у один спільний базис, який і повертається як результат.

```
function SOLVE_HOMOGENEOUS_SYSTEM(coefficients, modulo)
```
ввід: *coefficients* – двомірний масив коефіцієнтів розширеної системи рівнянь, *modulo* – модуль.
вивід: базис множини всіх розв'язків заданої системи.

```
    result ← {}
    // функція factorization(m) повертає масив трійок (div, prime, pow),
    // де кожна трійка задає один з дільників числа m – div = prime ^ pow
    for each (divider, prime, power) in factorization(modulo) do
        if power = 1 then
            curr_basis ← SOLVE_PRIME_MODULO(coefficients, prime)
        else
            curr_basis ← SOLVE_PRIME_POWER_MODULO(coefficients, divider)

        for each vector in curr_basis do
            new_vector ← (modulo / divider) × vector // добуток числа на вектор
            result add new_vector // додати new_vector в масив result

    return result
```

Функція *SOLVE_PRIME_MODULO* знаходить базис множини розв'язків СЛОДР у полі за модулем простого числа. Дана функція використовує допоміжну функцію пошуку базису множини розв'язків одного рівняння, а далі за допомогою знайденого базису знаходимо базис множини розв'язків вхідної системи.

```
function SOLVE_PRIME_MODULO (coefficients, modulo)
```
ввід: *coefficients* – двомірний масив коефіцієнтів розширеної системи рівнянь, *modulo* – модуль, що є простим числом.
вивід: базис множини всіх розв'язків заданої системи.

```
    basis ← TSS_BASIS_PRIME_MOD(coefficients[1], modulo)
    for i = 2 .. len(coefficients) do
        temp ← пустий вектор
        for j = 1 .. len(basis) do
            // coefficients[i] .* basis[j] – поелементний добуток векторів
            temp add sum(coefficients[i] .* basis[j])

        new_basis ← {}
        for v in TSS_BASIS_PRIME_MOD(temp, modulo) do
            temp ← ∑ v[k] * basis[k] for all k = 1..len(v) // сума векторів
            new_basis add temp

        basis ← new_basis

    return result
```

Функція *TSS_BASIS_PRIME_MOD* знаходить базис множини розв'язків ЛОДР (*TSS*-множину) у полі за модулем простого числа.

```
function TSS_BASIS_PRIME_MOD (coefficients, modulo)
```
ввід: *coefficients* – вектор коефіцієнтів, що задає одне рівняння, *modulo* – модуль, що є простим числом.
вивід: базис множини всіх розв'язків заданого рівняння.

```
    size ← len(coefficients)
    if size = 1
        if coefficients[1] = 0 then
            return {{1}}
        else
            return {{0}}

    // індекс першого ненульового елемента
    main_element ← FIRST_NON_ZERO_INDEX (coefficients)
    if main_element = -1 then
```

```
    return eye(size) //одинична матриця розмірності size на size

  result ← {}
  for i = 1..size do
  if i != main_element then
      temp ← zero_vector(size)
      temp[main_element] = coefficients[i]
      temp[i] = modulo - coefficients[i]
      result add temp

  return result, main_element
```

Наступні дві функції *SOLVE_PRIME_POWER_MODULO* та *TSS_BASIS_PRIME_POWER_MOD* є повністю аналогічними до попередніх двох, за винятком того, що розв'язують задачу у примарних кільцях.

```
  function SOLVE_PRIME_POWER_MODULO (coefficients, modulo)
  ввід: coefficients – двомірний масив коефіцієнтів розширеної системи рівнянь, modulo – модуль, що є степенем простого числа.
  вивід: базис множини всіх розв'язків заданої системи.

  basis ← TSS_BASIS_PRIME_POWER_MOD (coefficients[1], modulo)
  for i = 2 .. len(coefficients) do
      temp ← {} // пустий масив
      for j = 1 .. len(basis) do
          // coefficients[i] .* basis[j] – поелементний добуток векторів
          temp add sum(coefficients[i] .* basis[j])

      new_basis ← {}
      for v in TSS_BASIS_PRIME_POWER_MOD (temp, modulo) do
          temp ← ∑ v[k] * basis[k] for all k = 1..len(v) // сума векторів
          new_basis add temp

      basis ← new_basis

  return result

  function TSS_BASIS_PRIME_POWER_MOD (coefficients, modulo)
  ввід: coefficients – вектор коефіцієнтів, що задає одне рівняння, modulo – модуль, що є степенем простого числа.
  вивід: базис множини всіх розв'язків заданого рівняння.

  g ← GCD(modulo, coefficients) // НСД модуля та всіх коефіцієнтів
  coefficients ← coefficients ./ g // поелементне ділення вектора на число
  modulo ← modulo / g

  result, main_element ← TSS_BASIS_PRIME_MOD(coefficients, modulo)
  if g != 1 then
      temp ← zeros(len(result[0])) // нульовий вектор такої довжини, як result[0]
      temp[main_element] ← modulo
      result add temp

  return result
```

Ключові характеристики алгоритму

Таким чином було отримано алгоритм, здатний розв'язувати задачу за поліноміальний час за умови відомого розкладу модуля на прості множники. Враховуючи асимптотичну оцінку складності алгоритму, в реалізації було накладено наступні обмеження на вхідні дані:

- кількість рівнянь та невідомих не перевищує 500, що не є критичним, оскільки в разі необхідності цю границю можна збільшити;
- коефіцієнти та модуль не перевищують 10^9.

Зазначимо, що для розкладу модуля на прості множники було реалізовано один з най простіших алгоритмів, відомий як решето Ератосфена. Саме це і стало причиною встановлення такого обмеження на максимальну величину модуля.

У пропонованій таблиці подано результати експериментальних статистичних залежностей тривалості роботи програми від розмірності вхідних даних. Також зауважимо, що дані наведено з урахуванням тривалості роботи алгоритму розкладу модуля на прості множники.

Кількість рівнянь	Кількість невідомих	Модуль	Час роботи
25	26	10	0,2 с
90	99	600	3,4 с
100	150	463050	18,6 с
300	310	500	3,6 хв.
350	400	36	6,6 хв.
450	480	600	22,1 хв.

Висновки. Описаний алгоритм, в основу якого покладено *TSS*-метод, реалізовано у вигляді програмного застосування. Його тестування на реальних задачах підтверджує ефективне розв'язання поставлених задач.

1. *Донец Г.А.*, *Агаи Аг Гамиш Якуб.* Задача о математическом сейфе на матрицах // Зб. наук. пр. Ин-т кибернетики им. В. М. Глушкова НАНУ. – 2013. – № 12. –С. 124–131.

2. *Донец Г.А.* Решение задачи о построении линейной мозаики. – Теория оптимальных решений // Там же. – 2005. – № 4. – С. 15–24.
3. *Гантмахер Ф.Р.* Теория матриц. – М.: Гостехиздат, 1953. – 575 с.
4. *Василенко О.Н.* Теоретико-числовые алгоритмы в криптографии. – М.: МЦНМО, 2003. – 328 с.
5. *Кривий С.Л.* Лінійні діофантові обмеження та їх застосування. – Чернівці – Київ: Букрек. – 2015. – 222 с.
6. *Крывый С.Л.* Алгоритмы решения систем линейных диофантовых уравнений в кольцах вычетов. // Кибернетика и системный анализ. – 2016. – № 5. – С. 149–160.
7. *Крывый С.Л.* Алгоритмы решения систем линейных диофантовых уравнений в полях вычетов. // Там же. – 2007. – № 2. – С. 15–23.

Поступила 20.10.2017
Тел. для справок: +38 044 521-3554 (Киев)
E-mail: *sl.krivoi@gmail.com*, *vasia.antoniuk@gmail.com*

С.Л. Крывый, В.Т. Антонюк

Реализация алгоритма решения систем линейных диофантовых уравнений в кольце вычетов

Введение. Поиск множества решений системы линейных диофантовых уравнений – одна из актуальных задач алгебры, теории чисел, криптографии, теории математических игр и др. Примерами задач из перечисленных областей, решение которых необходимо находить в кольце или поле вычетов по модулю некоторого составного или простого числа, – это задачи о математическом сейфе [1], о распознавании изображений (о мозаике) [2], о распределении памяти при параллельных вычислениях.

Анализ свойств математических моделей этих задач сводится к поиску базиса множества всех решений системы линейных однородных или неоднородных диофантовых уравнений в кольце или в поле вычетов. В данной статье предложен алгоритм построения базиса множества всех решений таких систем и его программная реализация в языке C^{++}. Вначале вводятся необходимые сведения об основных понятиях и определениях с кратким обзором существующих методов решения. Далее приводятся детальный анализ алгоритма, описание реализации алгоритма и ограничения этой реализации.

Напомним, что кольцо вычетов Z_m называется *примарным*, если модуль m является степенью простого числа p, т.е. $m = p^t$, где $t > 1$, $t \in N$. Поскольку m не является простым числом, то сравнение $ax \equiv 4\ b(\mathrm{mod}\, m)$ в кольце Z_m при $a \neq 0$ не всегда будет иметь решение. Оно будет иметь решение, если НСД$(a, m) = 1$ или НСД$(a, m) = d$ и d – делитель числа b.

Известно, что если модуль – простое число, то кольцо вычетов по этому модулю будет полем, которое обозначается F_p. В поле операция взятия обратного элемента к данному элементу полностью определена

Постановка задачи

Системой линейных диофантовых уравнений (СЛДУ) над кольцом Z_m называется система

$$S = \begin{cases} L_1(x) = a_{11}x_1 + \ldots + a_{1q}x_q = b_1 \\ L_2(x) = a_{21}x_1 + \ldots + a_{2q}x_q = b_2 \\ \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots \\ L_s(x) = a_{s1}x_1 + \ldots + a_{sq}x_q = b_s \end{cases} (\mathrm{mod}\, m),$$

где $a_{ij}, b_i, x_j \in Z_m$, $i = 1, \ldots, s$, $j = 1, \ldots, q$.

Пусть $x = (x_1, x_2, \ldots, x_q)$. Решением СЛДУ S называется такой вектор $c = (c_1, c_2, \ldots, c_q)$, при подстановке которого вместо x в $L_i(x)$ получим равенство $L_i(c) = b_i (\mathrm{mod}\, m)$ для всех $i = 1, 2, \ldots, s$. СЛДУ называется однородной (СЛОДУ), если все b_i равны нулю, в противном случае СЛДУ называется неоднородной (СЛНДУ).

Базисом множества всех решений СЛДУ S называется такое множество векторов $B = \{e_1, e_2, \ldots, e_k\}$, для которого любое решение системы S можно единственным образом представить в виде линейной комбинации векторов из B.

Задачей, которая решается в данной статье, является поиск базиса множества всех решений СЛДУ S в кольце вычетов по модулю m.

Анализ алгоритмов

В случае решения задачи в поле F_p используется известный *метод Гаусса* [3] (метод последовательного исключения неизвестных). В этом методе путем элементарных преобразований над строками необходимо привести матрицу к единичному виду. Для выполнения этой операции необходимо уметь находить значение обратного элемента. Именно поэтому этот метод применим для решения задачи в поле F_p (поскольку в поле всегда существует обратный элемент). Однако метод Гаусса становится неприменимым в случае кольца по модулю составного числа.

Второй метод основан на алгоритме [4], в котором расширенная матрица системы $(A|b)$ приводится к ступенчатому виду $(A'|b')$, где находится правая матрица перехода R такая, что $(A|b) \times R = (A'|b')$. На основе матрицы R и можно получить решение системы. Однако при решении системы линейных уравнений в кольце вычетов наблюдается экспоненциальный рост норм коэффициентов. Еще один весомый недостаток данного метода – то, что он позволяет найти только одно решение системы, а не базис множества всех решений.

В итоге был выбран алгоритм, в основу которого положен *TSS*-метод [5] с характеристиками, которые лучше всего подходят для решения поставленной задачи. В основе алгоритма – идея разбиения заданной системы на подсистемы в полях и примарных кольцах вычетов по модулям, которые являются степенями простых чисел в разложении модуля на простые делители. В процессе работы алгоритма не возникает экспоненциального роста норм коэффициентов, поскольку все операции совершаются в соответствующем поле или кольце. Кроме того, этот алгоритм позволяет за полиномиальное время находить именно базис всех решений заданной СЛДУ.

***TSS*-метод**

Приведем описание *TSS*-метода без доказательства сопутствующих утверждений [5–7].

Пусть модуль m имеет такое разложение на простые делители $m = p_1^{k_1} p_2^{k_2} \dots p_r^{k_r}$, где $p_1 < p_2 < \dots < p_r$. Тогда по системе S построим систему с $r \cdot s$ уравнениями следующего вида:

$$\begin{cases} \begin{cases} L_1(x) = a_{11}x_1 + \dots + a_{1q}x_q - b_1x_0 = 0 \\ L_2(x) = a_{21}x_1 + \dots + a_{2q}x_q - b_1x_0 = 0 \\ \dots \quad \dots \quad \dots \quad \dots \quad \dots \\ L_s(x) = a_{s1}x_1 + \dots + a_{sq}x_q - b_sx_0 = 0 \end{cases} (\mathrm{mod}\ p_1^{k_1}) \\ \begin{cases} L_1(x) = a_{11}x_1 + \dots + a_{1q}x_q - b_1x_0 = 0 \\ L_2(x) = a_{21}x_1 + \dots + a_{2q}x_q - b_1x_0 = 0 \\ \dots \quad \dots \quad \dots \quad \dots \quad \dots \\ L_s(x) = a_{s1}x_1 + \dots + a_{sq}x_q - b_sx_0 = 0 \end{cases} (\mathrm{mod}\ p_2^{k_2}) \\ \dots \\ \begin{cases} L_1(x) = a_{11}x_1 + \dots + a_{1q}x_q - b_1x_0 = 0 \\ L_2(x) = a_{21}x_1 + \dots + a_{2q}x_q - b_1x_0 = 0 \\ \dots \quad \dots \quad \dots \quad \dots \quad \dots \\ L_s(x) = a_{s1}x_1 + \dots + a_{sq}x_q - b_sx_0 = 0 \end{cases} (\mathrm{mod}\ p_r^{k_r}) \end{cases}.$$

Теорема. Система S и S' эквивалентны, т.е. каждое решение системы S является решением системы S' и наоборот.

Из [5] следует такая процедура построения базиса множества решений СЛОДУ S в кольце Z_m: исходная СЛОДУ S разбивается на r подсистем $S_1, S_2, \dots, S_r$ по модулям $p_1^{k_1}, p_2^{k_2}, \dots, p_r^{k_r}$ соответственно, каждая из этих подсистем решается отдельно. В результате находим базисы $B_1, B_2, \dots, B_r$, соответственно для $S_1, S_2, \dots, S_r$, а затем строим базис $B = m/p_1^{k_1} B_1 \cup \dots \cup m/p_r^{k_r} B_r$, где $m/p_i^{k_i} B_i$ означает умножение каждого вектора из B_i на число $m/p_i^{k_i}$.

Теорема. Множество B, построенное *TSS*-методом, является базисом множества решений СЛОДУ S. Арифметическая сложность построения B пропорциональна величине $O(l^6)$, где $l = \max(n, s, q, r)$, n – максимальная разрядность чисел [5, 6].

Таким образом задача построения базиса множества решений СЛНДУ сводится к построению базиса множества решений расширенных СЛОДУ в поле вычетов или примарном кольце вычетов. Будем называть СЛОДУ S' соответствующей СЛНДУ S, которая строится путем введения дополнительной неизвестной x_0 и столбца коэффициентов при этом неизвестном, состоящем из свободных членов СЛНДУ. Если базис множества решений расширенной СЛОДУ найден, то выделим в нем решения $x_1, x_2, \dots, x_t$, в которых последняя координата не нулевая. Пусть $d_1, d_2, \dots, d_t$ – значение этих координат. Составим уравнение

$$c_1d_1 + c_2d_2 + \dots + c_td_t \equiv 1 (\mathrm{mod}\, m). \qquad (1)$$

Если это сравнение имеет решение $(u_1, u_2, \dots, u_t)$, то СЛНДУ совместна и линейная комбинация

$$x^1 = u_1x_1 + u_2x_2 + \dots + u_tx_t$$

является частным решением СЛНДУ, а остальные векторы из базиса множества решений будут решениями СЛОДУ, которая соответствует данной СЛНДУ.

Заметим, что сравнение (1) будет иметь решение только в случае $\mathrm{НОД}(d_1, d_2, \dots d_t) = 1$.

Более подробно ознакомиться с описанными утверждениями, а также с методами решения СЛОДУ в полях и примарных кольцах можно в [5–7].

***TSS*-алгоритм**

Для описания этого алгоритма и его программной реализации воспользуемся псевдокодом.

Главной функцией программы реализации *TSS*-алгоритма является *SOLVE*, которая получает данные о начальной СЛНДУ и возвращает результат построения базиса множества всех решений этой СЛНДУ. Точнее, эта функция превращает полученную систему в СЛОДУ, используя вспомогательную функцию решения однородных систем, получает ее решение и превращает его в решение начальной СЛНДУ.

```
function SOLVE(coefficients, modulo)
ввод: coefficients – двухмерный массив коэффициентов системы уравнений (последний элемент каждой строки задает свободный член каждого уравнения), modulo – модуль.
вывод: базис множества всех решений заданной системы.

    rows ← len(coefficients) // количество уравнений в заданной системе
    columns ← len(coefficients[1]) - 1 // количество переменных в заданной системе
    is_homogeneous ← True

    for i = 1..rows do
        if coefficients[i][columns + 1] != 0 // свободный член i-го уравнения
            is_homogeneous ← False
```

```
        coefficients[i][columns + 1] = modulo - coefficients[i][columns]

    result ← SOLVE_HOMOGENEOUS_SYSTEM(coefficients, modulo)
    // result – двумерный массив, где каждая строка задает один из
    // векторов базиса множества всех решений

    if is_homogeneous
        for i = 1..len(result) do
            remove_last_element(result[i]) // удалить последний элемент i-го вектора
        return result // для однородной системы возвращаем только базис

    // в случае неоднородной системы, необходимо вернуть
    // частичное решение и базис
    coeffs ← {} // присвоить пустой массив
    temp ← {}
    for vector in result do
        if vector[columns + 1] != 0 then
            // добавить последний элемент из vector в массив coeffs
            coeffs add vector[columns + 1]
            // добавить vector в массив temp
            temp add vector

    if len(temp) == 0 then // массив temp пустой
        return NO_SOLUTION

    result ← temp // присвоить массиву result массив temp
    u, solved ← MOD_SOLVER(coeffs, modulo)

    if not solved then
        return NO_SOLUTION

    for i = 1..len(result) do
        remove_last_element(result[i]) // удалить последний элемент i-го вектора

    x1 ← ∑ u[k] * result[k] for all k = 1..len(u) // поэлементная сумма векторов

    return x1, result
```

Функция *SOLVE_HOMOGENEOUS_SYSTEM* находит базис множества решений СЛОДУ. Входная СЛОДУ разбивается на множество подсистем, каждая из которых рассматривается над кольцом или полем вычетов по модулю соответствующего множителя факторизации входного модуля m. Затем используются вспомогательные функции решения СЛОДУ в полях и кольцах и комбинирование их результатов в один общий базис, который и возвращается в качестве результата.

```
    function SOLVE_HOMOGENEOUS_SYSTEM(coefficients, modulo)
    ввод: coefficients – двухмерный массив коэффициентов расширенной системы уравнений, modulo - модуль.
    вывод: базис множества всех решений заданной системы.

    result ← {}
    // функция factorization (m) возвращает массив троек (div, prime, pow),
    // где каждая тройка задает один из делителей числа m - div = prime ^ pow
    for each (divider, prime, power) in factorization(modulo) do
        if power = 1 then
            curr_basis ← SOLVE_PRIME_MODULO (coefficients, prime)
        else
            curr_basis ← SOLVE_PRIME_POWER_MODULO (coefficients, divider)

        for each vector in curr_basis do
            new_vector ← (modulo / divider) × vector // произведение числа на вектор
            result add new_vector // добавить new_vector в массив result

    return result
```

Функция *SOLVE_PRIME_MODULO* находит базис множества решений СЛОДУ в поле по модулю простого числа. Эта функция использует вспомогательную функцию нахождения базиса множества решений одного уравнения, а далее с помощью найденного базиса ищем базис множества решений исходной системы.

```
    function SOLVE_PRIME_MODULO(coefficients, modulo)
    ввод: coefficients – двухмерный массив коэффициентов расширенной системы уравнений, modulo - модуль, является простым числом.
    вывод: базис множества всех решений заданной системы.

    basis ← TSS_BASIS_PRIME_MOD(coefficients[1], modulo)
    for i = 2 .. len(coefficients) do
        temp ← пустой вектор
        for j = 1 .. len(basis) do
            // coefficients[i] .* basis[j] – поэлементное произведение векторов
            temp add sum(coefficients[i] .* basis[j])

        new_basis ← {}
        for v in TSS_BASIS_PRIME_MOD(temp, modulo) do
            temp ← ∑ v[k] * basis[k] for all k = 1..len(v) // сумма векторов
            new_basis add temp

        basis ← new_basis

    return result
```

Функция *TSS_BASIS_PRIME_MOD* находит базис множества решений ЛОДУ (*TSS*-множество) в поле по модулю простого числа.

```
    function TSS_BASIS_PRIME_MOD(coefficients, modulo)
    ввод: coefficients – вектор коэффициентов, что задает одно уравнение, modulo - модуль, является простым числом.
    вывод: базис множества всех решений заданного уравнения.

    size ← len(coefficients)
    if size = 1
        if coefficients[1] = 0 then
            return {{1}}
        else
            return {{0}}
```

```
    // индекс первого ненулевого элемента
    main_element ← FIRST_NON_ZERO_INDEX (coeffi-
cients)
    if main_element = -1 then
        return eye(size) // единичная матрица разме-
рности size на size

    result ← {}
    for i = 1..size do
    if i != main_element then
        temp ← zero_vector(size)
        temp[main_element] = coefficients[i]
        temp[i] = modulo - coefficients[i]
        result add temp

    return result, main_element
```

Следующие две функции *SOLVE_PRIME_POWER_MODULO* и *TSS_BASIS_PRIME_POWER_MOD* полностью аналогичны предыдущим двум, за исключением того, что решают задачу в примарных кольцах.

```
    function SOLVE_PRIME_POWER_MODULO(coefficients,
modulo)
    ввод: coefficients – двухмерный массив коэффи-
циентов расширенной системы уравнений, modulo -
модуль, является степенью простого числа.
    вывод: базис множества всех решений заданной
системы.

    basis ← TSS_BASIS_PRIME_POWER_MOD (coefficients
[1], modulo)
    for i = 2 .. len(coefficients) do
        temp ← {} // пустой массив
        for j = 1 .. len(basis) do
    // coefficients[i] .* basis[j] - поэлементное
произведение векторов
            temp add sum(coefficients[i] .*
basis[j])

        new_basis ← {}
        for v in TSS_BASIS_PRIME_POWER_MOD(temp,
modulo) do
                temp ← Σ v[k] * basis[k] for all k =
1..len(v) // сумма векторов
                new_basis add temp

            basis ← new_basis

    return result

    function TSS_BASIS_PRIME_POWER_MOD (coeffi-
cients, modulo)
    ввод: coefficients – вектор коэффициентов, за-
дает одно уравнение, modulo - модуль, является
степенью простого числа.
    вывод: базис множества всех решений заданного
уравнения.

    g ← GCD(modulo, coefficients) // НОД модуля и
всех коефіцієнтов

    coefficients ← coefficients ./ g // поэлемент-
ное деление вектора на число
    modulo ← modulo / g

    result, main_element ← TSS_BASIS_PRIME_MOD (co-
efficients, modulo)
    if g != 1 then
        temp ← zeros(len(result[0])) // нулевой ве-
ктор такой длины, как result[0]
        temp[main_element] ← modulo
        result add temp

    return result
```

Ключевые характеристики алгоритма

Приведенный алгоритм решает поставленную задачу за полиномиальное время при условии известного разложения модуля на простые делители. Учитывая описанную асимптотическую оценку сложности алгоритма, в реализации были наложены следующие ограничения на входные данные:

- количество уравнений и неизвестных не превышает 500, что не критично, поскольку при необходимости эту границу можно увеличить;
- коэффициенты и модуль не превышают величины 10^9.

Отметим, что для разложения модуля на простые множители реализован один из самых простых алгоритмов факторизации, известный как решето Эратосфена. Это обстоятельство и стало причиной такого ограничения на максимальную величину модуля.

В предлагаемой таблице приведены результаты экспериментальных статистических зависимостей продолжительности работы программы от размерности входных данных. Также заметим, что данные приведены с учетом продолжительности работы алгоритма разложения модуля на простые множители.

Количество уравнений	Количество неизвестных	Модуль	Время работы
25	26	10	0,2 с
90	99	600	3,4 с
100	150	463050	18,6 с
300	310	500	3,6 мин.
350	400	35	6,6 мин.
450	480	600	22,1 мин.

Заключение. Описанный алгоритм, в основе которого – *TSS*-метод, реализован в виде программного приложения. Его тестирование на реальных задачах подтверждает эффективное решение поставленных задач.

UDC 51.681.3

S.L. Kryvyi[1], V.T. Antoniuk[2]

[1] Doctor of Physical and Mathematical Science, Professor, Professor of the Informational Systems Department of the Taras Shevchenko National University of Kyiv, E-mail: sl.krivoi@gmail.com

[2] Student of the Informational Systems Department of the Taras Shevchenko National University of Kyiv, E-mail vasia.antoniuk@gmail.com

The Implementation of the Algorithm for Solving Systems of Linear Diophantine Equations Over Finite Residue Rings

Keywords: residue ring, linear Diophantine constraints, set of basis solutions, software implementation.

Introduction. Finding the basis solutions set of the system of the linear Diophantine equations is one of the important and necessary problems of algebra, number theory, cryptography, the theory of mathematical games etc. When constructing a mathematical model of a large number of problems, it becomes clear that one of the methods of their solution is to find the basis of the set of all solutions of the system of the linear homogeneous or inhomogeneous Diophantine equations in a residue ring or a residue field modulo a composite or a prime number.

Purpose. The purpose of this work is to develop a programme that will be able to find a set of basis solutions of a system of linear Diophantine in a residue ring or a residue field modulo a composite or a prime number.

Methods. Achievement of the purpose is associated with solving the following problems: analysis of existing algorithms; detailed analysis and description of the chosen algorithm; its software implementation.

Results. Algorithms based on the TSS-method were chosen to solve the problem. The chosen algorithms were described in detail and the corresponding software solution was constructed.

Conclusion. As a result of the work, effective algorithms for solving the problem are described and implemented. The resulting application can be used in solving the relevant practical problems. Also, a detailed description of the algorithms will allow other researchers to build a wider system (for example, for greater constraints).

1. *Donets G.A*., *Aghaei Agh Ghamish Yaghoub* The mathematical safe problem on matrices, in: Prepr. V.M. Glushkov Institute of Cybernetics of NANU, 2013, P. 124–131. (In Russian).
2. *Donets G.A*., *Samer I.M*. Alshalame, «Solution of the problem of construction of a linear mosaic», in: Theory of Optimal Solutions, V.M. Glushkov Cybernetic Institute of NASU, Kiev, 2005, P. 15–24. (In Russian).
3. *Gantmacher F.R*., Matrix theory, Chelsea Publishing, New York, 1959.
4. *Vasilenko O.N*., Number-Theoretic Algorithms in Cryptography (MTsNMO, Moscow, 2003). (In Russian).
5. *Kryvyi S.L*., Linear Diophantine Equations and Their Applications (Bukrek, Kiev, 2015). (In Ukrainian).
6. *Kryvyi S.L.* Algorithms for solving systems of linear Diophantine equations in residue rings, Cybernetics and Systems Analysis, 2007, 43, **6**, P. 787–798. (In Russian).
7. *Kryvyi S.L*, "Algorithms for solution of systems of linear Diophantine equations in residue fields", Cybernetics and Systems Analysis, 43, 2007, **2**, P. 171–178.(In Russian).

●

Для соответствия научно-метрическим базам при подаче статей к рассмотрению, авторы должны подать метаданные на английском языке:

– ФИО

– место и адрес работы каждого автора

– расширенную аннотацию (до 2000 знаков с пробелами и рубриками: *Introduction*, *Purpose*, *Methods*, *Results*, *Conclusion*)

– список пристатейной литературы в переводе или транслитерации.

При оформлении списков литературы к расширенной аннотации на английском языке, можно пользоваться сайтом *http://translit.net* для русских ссылок *http://ukrlit.org/transliteratsiia* для украинских.