МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»**

Кафедра теоретических основ компьютерной безопасности и криптографии

**Алгоритм Ленстры-Ленстры-Ловаша**

ОТЧЁТ

ПО ДИСЦИПЛИНЕ

«ТЕОРЕТИКО-ЧИСЛОВЫЕ МЕТОДЫ В КРИПТОГРАФИИ»

студента 5 курса 531 группы

специальности 10.05.01 Компьютерная безопасность

факультета компьютерных наук и информационных технологий

Сенокосова Владислава Владимировича

|  |  |  |
| --- | --- | --- |
| Преподаватель, профессор | \_\_\_\_\_\_\_\_\_\_ | В.А. Молчанов |
|  | подпись, дата |  |

Саратов 2024

**Содержание**

[1 Цель работы и порядок выполнения 3](#_Toc180454131)

[2 Алгоритм Гаусса редукции решеток размерности 2 4](#_Toc180454132)

[3 Алгоритм Ленстры-Ленстры-Ловаша (LLL-алгоритм) 8](#_Toc180454133)

[4 Алгоритм решения задачи целочисленного программирования с помощью LLL-алгоритма 12](#_Toc180454134)

[5 Тестирование реализованных алгоритмов 18](#_Toc180454135)

[ЗАКЛЮЧЕНИЕ 21](#_Toc180454136)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 22](#_Toc180454137)

[ПРИЛОЖЕНИЕ А 23](#_Toc180454138)

# **1 Цель работы и порядок выполнения**

**Цель работы** — изучение алгоритма минимизации базиса решетки и его программная реализация.

Порядок выполнения работы

1. Рассмотреть алгоритм Гаусса редукции решеток размерности 2 и привести его программную реализацию.

2. Рассмотреть Алгоритм Ленстры-Ленстры-Ловаша (LLL-алгоритм) и привести его программную реализацию.

3. Реализовать алгоритм решения задачи целочисленного программирования с помощью LLL-алгоритма.

# **2 Алгоритм Гаусса редукции решеток размерности 2**

**Определение:** Решеткой размерности в пространстве называется любое его подмножество вида:

где – линейно независимая система векторов из , называемая базисом решетки. Если , то решетка называется полной. В частности, подгруппа по сложению пространства Также очевидно, что любая решетка размерности является подмножеством некоторой полной решетки.

**Определение:** Пусть – евклидово пространство размерности Для любой пары векторов определено скалярное произведение а также длина вектора . Векторы называются ортогональными, если . Иногда скалярное произведение двух векторов обозначается .

**Определение:** Нормой вектора называется мера его длины или размера, определяемая по формуле:

При применении алгоритмов редукции решеток, таких как алгоритм Гаусса, используется информация о нормах векторов для упрощения базиса решетки.

Рассмотрим решетку размерности и множество всех ее базисов. На множестве сначала введем отношение эквивалентности

() () для всех

Далее на множестве классов эквивалентности введем отношение порядка

Очевидно, что введенное отношение определено корректно. В силу дискретности решетки можно утверждать, что множество счетно, и в нем найдется наименьший элемент относительно отношения

Обозначим наименьший элемент множества через []~. Нетрудно заметить, что любой базис () обладает следующим свойством:

1) — кратчайший ненулевой вектор ;

2) для любого вектор имеет наименьшую длину среди всех таких векторов , что , можно дополнить до базиса .

**Определение:** Для решетки размерности базис

называется редуцированным (приведенным) по Минковскому.

Из приведенных выше рассуждений следует, что для любой решетки приведенный по Минковскому базис существует. При этом он может быть не единственным.

**Лемма:** Если базис решетки приведен по Минковскому, то:

1) — кратчайший ненулевой вектор ;

2)

3) .

**Определение:** -м последовательным минимумом решетки L называется такое наименьшее положительное число = , что найдутся линейно независимых над векторами решетки , длина которых не превосходит .

Алгоритм Гаусса редукции решеток размерности 2:

**ВХОД:** базис ​, решетки размерности 2, упорядоченный таким образом, что .

**ВЫХОД:** редуцированный по Минковскому базис решетки .

**Шаг 1:** Вычислить **.** Положить

**Шаг 2:**Проверить выполнение неравенства . Если это неравенство выполнено, то заменить , и перейти к шагу 1. В противном случае заменить . Алгоритм заканчивает работу.

Оценим сложность алгоритма:

**Лемма:** Пусть линейно независимые векторы пространства такие что . Пусть – кратчайший ненулевой вектор решетки, порожденный Тогда число шагов алгоритма Гаусса, примененного к , оценивается величиной:

**Лемма:** Пусть линейно независимые векторы из , такие что для некоторого числа Тогда сложность применения алгоритма Гаусса к паре векторов равна двоичных операций при и ограниченном .

В случае работы с машинными словами можно оценить сложность алгоритма как

Псевдокод реализованного алгоритма:

Начало алгоритма

ВХОД:

: Вектор размерности 2 (например, )

: Вектор размерности 2 (например, )

ВЫХОД: Редуцированный базис решетки

ШАГ 1:

Если , то:

Меняем местами и

ШАГ 2:

Вычисляем скалярное произведение и

Вычисляем квадрат нормы

Вычисляем значение

Обновляем вектор

ШАГ 3:

Если , то:

Переходим к Шагу 2

Иначе:

Вернуть

Конец алгоритма

Сложность алгоритма:

# **3 Алгоритм Ленстры-Ленстры-Ловаша (LLL-алгоритм)**

**Определение:** Пусть  — решетка размерности  в , . Пусть имеется некоторая константа ​, которая зависит только от и не зависит от самой решетки. Назовем базис ​ решетки приведенным (редуцированным), если имеет место неравенство .

**Определение:** Базис решетки называется редуцированным, если выполнены следующие условия:

1) при всех ;

2) при .

Свойства LLL редуцированного базиса описывает следующая теорема. Из этой теоремы, в частности, следует, что LLL редуцированный базис является приведенным в смысле первого определения для константы .

**Теорема:** Пусть редуцированный базис решетки Тогда:

1. при всех
2. Для любого ненулевого вектора

Алгоритм Ленстры-Ленстры-Ловаша:

ВХОД: базис решетки .

ВЫХОД: - редуцированный базис решетки .

**Шаг 1:** Вычислить ортогонализацию Грамма–Шмидта для системы . Для этого положить и для всех вычислить

где

Положить Положить

**Шаг 2:** (Уменьшить длины векторов базиса). Выполнить шаг 4 при Если и , то перейти к шагу 3. В противном случае выполнить шаг 4 последовательно при Если , то алгоритм заканчивает работу. В противном случае положить . Перейти к шагу 2.

**Шаг 3:**(Поменять местами , изменить соответствующие значения ). Выполнить следующее:

а также

при

и

при

Положить Перейти к шагу 2.

**Шаг 4:** Если , то вычислить положить и при и

Для обоснования алгоритма надо сделать следующее:

1. проверить, что при перестановке значения изменяются указанным в шаге 3 способом;
2. проверить, что при замене значения чисел изменяются указанным в шаге 4 образом, а все не изменяются;
3. алгоритм заканчивается через конечное число шагов

Заметим, что при текущем значении векторы редуцированы. Отсюда следует, что алгоритм *–* редуцированный базис решетки

Оценка сложности строится на следующей теореме.

**Теорема:** Пусть решетка размерности c базисом где для всех Тогда сложность применения   
алгоритма к этому базису не больше арифметических операций с целыми числами, двоичная длина которых не больше

Псевдокод реализованного алгоритма

Начало алгоритма

ВХОД:

Базис решетки

Параметр, часто

ВЫХОД:

-редуцированный базис решетки

ШАГ 1:

Установить # Начинаем с первого индекса

Пока :

Для каждого от до 0 с шагом -1:

Вычисляем скалярное произведение

Если :

Округлить μ до ближайшего целого

Обновить вектор

Обновить значения и после изменения

ШАГ 2:

Вычисляем квадрат нормы ортогонального базиса:

Вычисляем значение левой части

Вычисляем значение правой части

Если :

Увеличиваем на 1

Иначе:

Меняем местами и

Обновляем ортогональный базис и

Устанавливаем

ШАГ 3:

Вернуть редуцированный базис

Конец алгоритма

Сложность реализованного алгоритма:

# **4 Алгоритм решения задачи целочисленного программирования с помощью LLL-алгоритма**

Рассмотрим следующую задачу. Найти все такие целочисленные векторы , что

… (17)

при заданных , , таких что ранг матрицы размера равен . Заметим, что задача легко сводится к случаю, когда . Действительно, так как ранг матрицы равен , то матрица содержит ненулевой минор порядка . Рассмотрим только те неравенства из (17), которые соответствуют строкам, вошедшим в этот минор. Вычислив все целочисленные решения новой системы неравенств, возьмем те из них, которые удовлетворяют неравенствам (17). Они составят все решения (17).

ВХОД: базис решетки размерности в , множество , определенное системой неравенств

ВЫХОД: элементы множества

**Шаг 1.** Вычислить приведенный базис решетки L. Пусть — приведенный базис , т. е. существует постоянная , которая зависит лишь от , что

Этот шаг может быть реализован применением алгоритма или других алгоритмов построения приведенного базиса.

Далее вместо (17) решим систему неравенств

… (19)

Где

(20)

и целочисленная обратимая над матрица размера . Это матрица перехода от исходного базиса решетки к приведенному базису.

**Шаг 2.** Вычислить ортогонализацию Грамма–Шмидта базиса . Положить .

**Шаг 3.** Положить

Представить p через базис :

, где

Положить

Для каждого выполнить шаг 4.

**Шаг 4.** От системы (19) перейти к системе

… (21)

Пусть составляют столбцы матрицы размера . Найти ненулевой минор порядка матрицы . Пусть — номера строк матрицы , которые содержат этот минор. Рассмотрим задачу вычисления всех решений системы

…

Эта система составлена из неравенств системы (21) с номерами . Для решения этой системы от переменных применим рекурсивно алгоритм.

**Шаг 5.** Из соотношений (20) найдем . Проверкой выполнения неравенств (17) определим, какие из них являются решениями этой системы. Алгоритм заканчивает работу. Докажем, что алгоритм действительно вычисляет все решения системы (17). Заметим, что множество содержится в шаре радиуса с центром в точке . Действительно, этот шар описан вокруг мерного куба со стороной, равной и с центром в точке . Пусть — гиперплоскость в , которая порождается векторами и есть решетка размерности с базисом . Таким образом, , и решетка содержится в счетном объединении гиперплоскостей

Множество содержится в объединении только тех гиперплоскостей , которые имеют непустое пересечение с , так как . Расстояние между гиперплоскостями равно . Поэтому не более последовательных гиперплоскостей пересекают шар . Все эти гиперплоскости содержатся среди таких, что . Отсюда легко следует корректность алгоритма. Заметим, что . Поэтому по свойству приведенного базиса имеем

Отсюда . Значит, число вариантов , которые перебираются на шаге 3, ограничено величиной . В конкретных вычислениях (в случае неприведенного базиса), длина вектора велика, а величина может быть мала. Это может привести к значительному увеличению трудоемкости вычислений, если пользоваться неприведенным базисом.

Сложность алгоритма:

Псевдокод алгоритма

Начало алгоритма

ВХОД:

: Вектор нижних границ

Вектор верхних границ

Матрица базиса решетки

Размерность пространства

ВЫХОД:

Список возможных решений на решетке

ШАГ 1:

= Грам-Шмидт(B) # Ортогонализируем базис B методом Грама-Шмидта

Вычисляем произведение норм ортогонализированных векторов:

= произведение(норм для от до )

ШАГ 2:

# Создаем список для хранения границ возможных решений

Для от до :

Вычисляем скалярное произведение ортогонализированных векторов:

Вычисляем нижнюю границу для i-го измерения:

d = округление\_вверх()

Вычисляем верхнюю границу для i-го измерения:

= округление\_вниз()

Если — это скалярное значение:

Иначе:

первый элемент

Если — это скаляр:

Иначе:

Добавить в список solutions пару ()

ШАГ 3:

# Находим среднюю точку между границами

максимум(абсолютных значений 2 для от # Вычисляем радиус

ШАГ 4:

Функция

Если

Вернуть список с нулевым вектором длины depth

Для каждого :

Для каждого z в диапазоне от до

= добавить в конец

Добавить в список

Вернуть список

# Находим все возможные решения

ШАГ 5:

Для каждого в

Если произведение матрицы на вектор удовлетворяет условиям

Добавить в список

Вернуть

Конец алгоритма

Сложность алгоритма:

# **5 Тестирование реализованных алгоритмов**

Все алгоритмы были реализованы на языке Python. Результаты работы программы продемонстрированы на следующих рисунках.

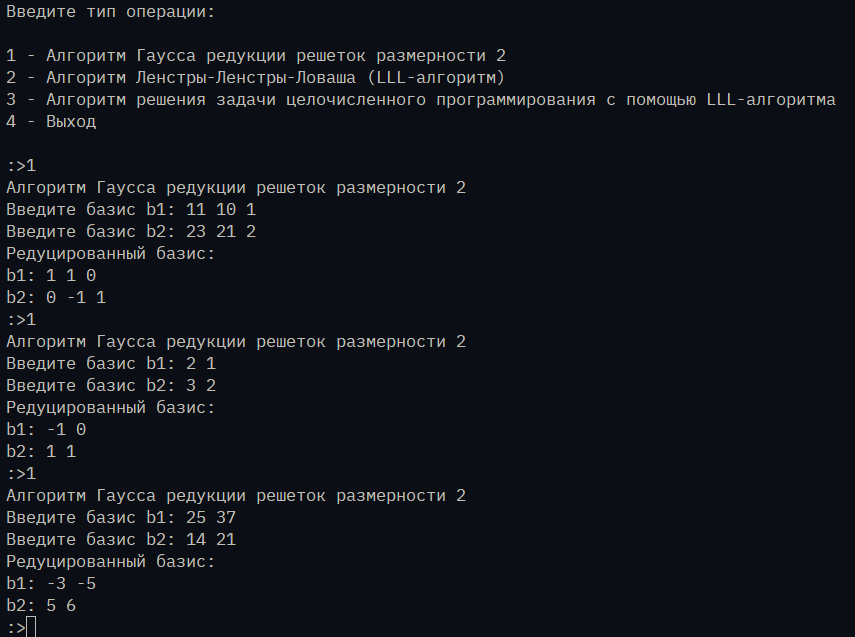


Рисунок 1 – Тестирование алгоритма Гаусса редукции решеток размерности 2

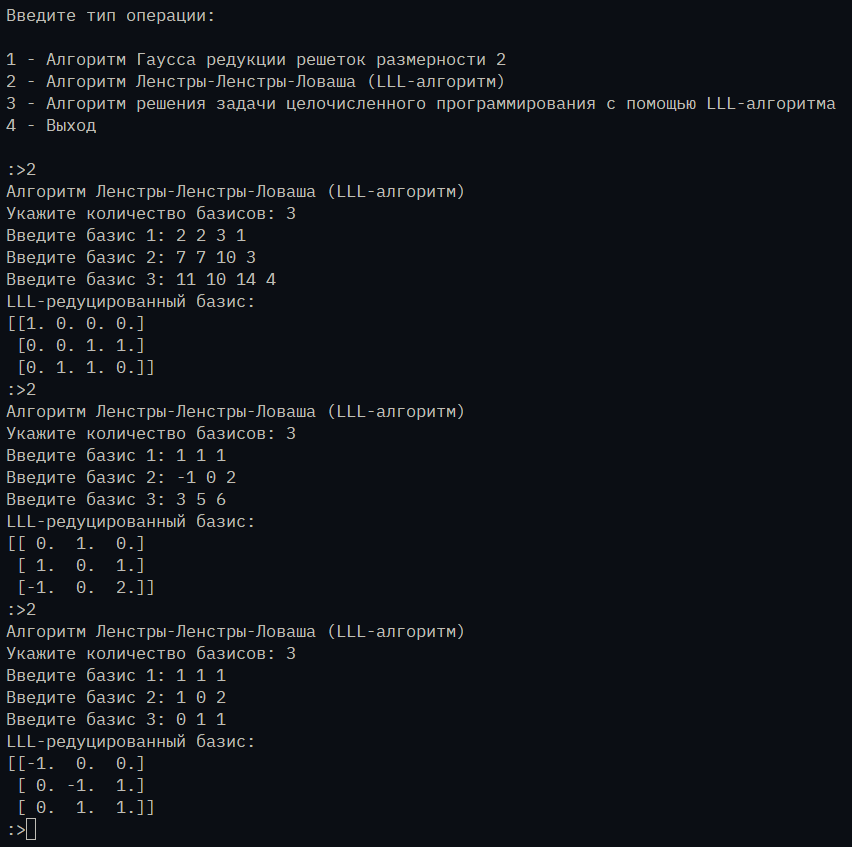


Рисунок 2 – Тестирование алгоритма Ленстры-Ленстры-Ловаша (LLL-алгоритм)

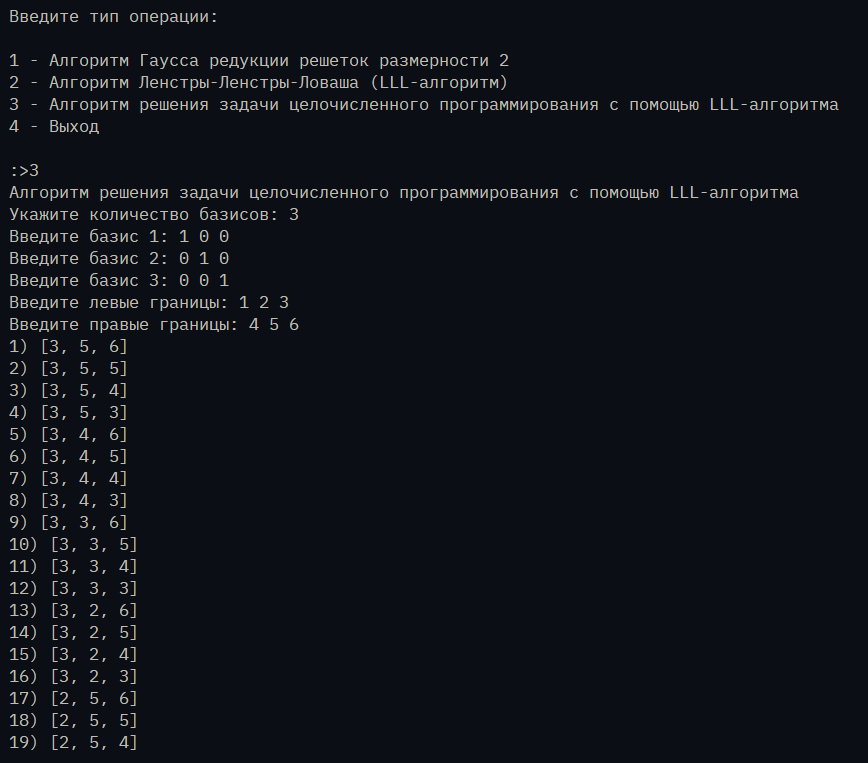


Рисунок 3 – Тестирование алгоритма решения задачи целочисленного программирования с помощью LLL-алгоритма

# **ЗАКЛЮЧЕНИЕ**

В ходе выполнения работы была достигнута основная цель — изучение алгоритмов минимизации базиса решетки и их программная реализация на языке Python. Работа включала три этапа:

1. Алгоритм Гаусса для редукции решеток размерности 2: На первом этапе был изучен и реализован алгоритм Гаусса, предназначенный для минимизации базиса двумерной решетки. Алгоритм был подробно описан и реализован в виде Python-кода, что позволило понять основные принципы редукции решеток в двумерном пространстве.
2. Алгоритм Ленстры-Ленстры-Ловаша (LLL-алгоритм): На втором этапе был рассмотрен LLL-алгоритм, который представляет собой более общий подход к минимизации базиса решеток любой размерности. Этот алгоритм был также реализован на Python, с использованием необходимых математических вычислений для проверки условий редукции и выполнения шагов алгоритма.
3. Реализация задачи целочисленного программирования с использованием LLL-алгоритма: На третьем этапе был разработан алгоритм для решения задачи целочисленного программирования, основанный на LLL-алгоритме. В процессе работы был выполнен перебор решений на решетке с использованием ортогонализации базиса и проверкой найденных решений на соответствие заданным границам. Для этого были применены как итеративные методы, так и дополнительные проверки правильности решений.

Таким образом, все задачи, поставленные в рамках данной работы, были успешно решены. Разработанная программа корректно выполняет редукцию базиса решетки и решает задачи целочисленного программирования с использованием LLL-алгоритма.

# **СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ**

1. Глухов М. М. и др. Введение в теоретико-числовые методы криптографии: учеб. пособие - Москва : Лань, 2011.

2. Маховенко Е.Б. Теоретико-числовые методы в криптографии. М.: Гелиос APB, 2006.

3. Черемушкин, А. В. Лекции по арифметическим алгоритмам в криптографии. - Москва : МЦНМО, 2002.

4. Панкратова И.А. Теоретико-числовые методы в криптографии. Томск, 2009.

5. Василенко О.Н. Теоретико-числовые алгоритмы в криптографии. М.:МЦНМО, 2003.

6. Венбо Мао. Современная криптография: теория и практика. М.:Вильямс, 2005.

# **ПРИЛОЖЕНИЕ А**

**Реализованные программы для лабораторной работы**

import numpy as np

*# Вывод матрицы в терминал*

def print\_matrix(matrix):

    for row in matrix:

        print(\*row)

    print()

def gauss\_reduce\_lattice\_2d(b1, b2):

    b1 = np.array(b1)

    b2 = np.array(b2)

    if np.linalg.norm(b1) > np.linalg.norm(b2):

        b1, b2 = b2, b1

    while True:

        dot\_product = np.dot(b1, b2)

        norm\_b1\_squared = np.linalg.norm(b1) \*\* 2

        r = dot\_product / norm\_b1\_squared + 0.5

        r\_rounded = int(r + 0.1)

        a = b2 - r\_rounded \* b1

        if np.linalg.norm(a) \*\* 2 < np.linalg.norm(b1) \*\* 2:

            b1, b2 = a, b1

        else:

            return b1, a

def gram\_schmidt(basis):

    d = len(basis)

    orthogonal\_basis = np.zeros\_like(basis)

    mu = np.zeros((d, d))

    for i in range(d):

        orthogonal\_basis[i] = basis[i]

        for j in range(i):

            mu[i, j] = np.dot(basis[i], orthogonal\_basis[j]) / \

                np.dot(orthogonal\_basis[j], orthogonal\_basis[j])

            orthogonal\_basis[i] -= mu[i, j] \* orthogonal\_basis[j]

    return orthogonal\_basis, mu

def update\_mu(mu, basis, orthogonal\_basis, k, d):

    for j in range(k):

        mu[k, j] = np.dot(basis[k], orthogonal\_basis[j]) / \

            np.dot(orthogonal\_basis[j], orthogonal\_basis[j])

def lll\_reduce(basis, delta=3/4):

    d = len(basis)

    basis = np.array(basis, dtype=float)

    orthogonal\_basis, mu = gram\_schmidt(basis)

    k = 1

    while k < d:

        for j in range(k - 1, -1, -1):

            if abs(mu[k, j]) > 0.5:

                r = round(mu[k, j])

                basis[k] -= r \* basis[j]

                update\_mu(mu, basis, orthogonal\_basis, k, d)

        if np.dot(orthogonal\_basis[k], orthogonal\_basis[k]) >= (delta - mu[k, k - 1]\*\*2) \* \

                    np.dot(orthogonal\_basis[k - 1], orthogonal\_basis[k - 1]):

            k += 1

        else:

            basis[k], basis[k - 1] = basis[k - 1].copy(), basis[k].copy()

            orthogonal\_basis, mu = gram\_schmidt(basis)

            k = max(k - 1, 1)

    return basis

def gram\_schmidt\_1(B):

    """Ортогонализация базиса методом Грама-Шмидта."""

    B = np.array(B)

    N = B.shape[0]

    B\_reduced = np.zeros\_like(B, dtype=float)

    for i in range(N):

        B\_reduced[i] = B[i]

        for j in range(i):

            B\_reduced[i] -= np.dot(B[i], B\_reduced[j]) / \

                np.dot(B\_reduced[j], B\_reduced[j]) \* B\_reduced[j]

    return B\_reduced

def iterative\_solve(solutions):

    """Итеративный перебор всех возможных решений в заданных границах."""

    N = len(solutions)

    stack = [[]]

    results = []

    while stack:

        current\_sol = stack.pop()

*# Проверяем, если текущее решение содержит все координаты*

        if len(current\_sol) == N:

            results.append(current\_sol)

        else:

            depth = len(current\_sol)

            lower\_bound, upper\_bound = solutions[depth]

*# Добавляем новые возможные решения на стеке*

            for z in range(lower\_bound, upper\_bound + 1):

                stack.append(current\_sol + [z])

    return results

def find\_solutions(C1, C2, B):

    """Нахождение всех решений на решетке в пределах границ C1 и C2."""

    N = len(B)

*# Шаг 1: Ортогонализация Грама-Шмидта*

    B\_reduced = gram\_schmidt\_1(B)

*# Шаг 2: Вычисляем границы для каждого измерения*

    solutions = []

    for i in range(N):

        dot\_product = np.dot(B\_reduced[i], B\_reduced.T)

        lower\_bound = np.ceil((C1[i] - dot\_product) / B\_reduced[i][i]).astype(int)

        upper\_bound = np.floor((C2[i] - dot\_product) / B\_reduced[i][i]).astype(int)

        lower\_bound\_scalar = lower\_bound if np.isscalar(lower\_bound) else lower\_bound[0]

        upper\_bound\_scalar = upper\_bound if np.isscalar(upper\_bound) else upper\_bound[0]

        solutions.append((lower\_bound\_scalar, upper\_bound\_scalar))

*# Шаг 3: Итерируем через возможные решения*

    all\_solutions = iterative\_solve(solutions)

*# Шаг 4: Фильтруем решения по условиям C1 <= np.dot(B, sol) <= C2*

    final\_solutions = []

    for sol in all\_solutions:

        sol\_vector = np.dot(B, sol)

        if np.all(sol\_vector >= C1) and np.all(sol\_vector <= C2):

            final\_solutions.append(sol)

    return final\_solutions

if \_\_name\_\_ == "\_\_main\_\_":

    type\_ = """Введите тип операции: \n

1 - Алгоритм Гаусса редукции решеток размерности 2

2 - Алгоритм Ленстры-Ленстры-Ловаша (LLL-алгоритм)

3 - Алгоритм решения задачи целочисленного программирования с помощью LLL-алгоритма

4 - Выход\n"""

    print(type\_)

    param = None

    while param not in ["1", "2", "3", "4"]:

        param = input(":>")

        match param:

            case "1":

                print("Алгоритм Гаусса редукции решеток размерности 2")

                b1 = list(map(lambda x: int(x),

input("Введите базис b1: ").split()))

                b2 = list(map(lambda x: int(x),

input("Введите базис b2: ").split()))

                b1\_reduced, b2\_reduced = gauss\_reduce\_lattice\_2d(b1, b2)

                print("Редуцированный базис:")

                print("b1:", \*b1\_reduced)

                print("b2:", \*b2\_reduced)

                param = None

            case "2":

                print("Алгоритм Ленстры-Ленстры-Ловаша (LLL-алгоритм)")

                k = int(input("Укажите количество базисов: "))

                lst = []

                for i in range(k):

                    b = list(map(lambda x: int(x),

input(f"Введите базис {i + 1}: ").split()))

                    lst.append(b)

                reduced\_basis = lll\_reduce(lst)

                print("LLL-редуцированный базис:")

                print(reduced\_basis)

                param = None

            case "3":

                print("Алгоритм решения задачи целочисленного

программирования с помощью LLL-алгоритма")

                k = int(input("Укажите количество базисов: "))

                lst = []

                for i in range(k):

                    b = list(map(lambda x: int(x),

input(f"Введите базис {i + 1}: ").split()))

                    lst.append(b)

                c1 = list(map(lambda x: int(x),

input("Введите левые границы: ").split()))

                c2 = list(map(lambda x: int(x),

input("Введите правые границы: ").split()))

                B = np.array(lst)

                C1 = np.array(c1)

                C2 = np.array(c2)

                solutions = find\_solutions(C1, C2, B)

                for i, el in enumerate(solutions):

                    print(f"{i + 1})", el)

                param = None

            case "4":

                param = "4"