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

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 k в пространстве $\mathrm{R}^n, n \geq k,$ называется любое его подмножество вида:

$$L = \{z_1 b_1 + z_2 b_2 + \cdots + z_k b_k | z_i \in Z, i = 1, \ldots k\} \in \mathrm{R}^n$$

где $b_1 \ldots . b_k$ – линейно независимая система векторов из R^n, называемая базисом решетки. Если $n = k$, то решетка называется полной. В частности, $L -$ подгруппа по сложению пространства R^n. Также очевидно, что любая решетка размерности $k < n$ является подмножеством некоторой полной решетки.

Определение: Пусть R^n – евклидово пространство размерности n. Для любой пары векторов $b = (b_1, b_2, \ldots, b_n), c = (c_1, c_2, \ldots, c_n)$ определено скалярное произведение $(b; c) = \sum_{i=1}^{n} b_i c_i$, а также длина вектора $\|b\| = \sqrt{(b;\ b)}$. Векторы b, c называются ортогональными, если $(b; c) = 0$. Иногда скалярное произведение двух векторов b_1, b_2 обозначается $\langle b_1, b_2 \rangle$.

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

$$\|x\| = \sqrt{x_1^2 + \cdots + x_n^2}$$

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

Рассмотрим решетку $L \subseteq R^n$ размерности $k \ \leq \ n$ и множество G_L всех ее базисов. На множестве G_L сначала введем отношение эквивалентности

$(b_1,\ b_2,\ \ldots,\ b_k) \ \sim \ (c_1,\ c_2,\ \ldots,\ c_k) \ \Leftrightarrow$ для всех $i \in \{1, \ldots, n\}$: $||b_i|| = ||c_i||$.

Далее на множестве классов эквивалентности $G_L/\sim$ введем отношение порядка $<$:

$[b_1,\ b_2,\ \ldots,\ b_k]_\sim < \ [c_1,\ c_2,\ \ldots,\ c_k]_\sim \ \Leftrightarrow \ j \ \in \ \{1, \ldots, n\}$: $||b_1|| = ||c_1||, \ldots, ||b_{j-1}|| \ = \ ||c_{j-1}||,\ ||b_j|| \ < \ ||c_j||$.

Очевидно, что введенное отношение определено корректно. В силу дискретности решетки L можно утверждать, что множество $G_L/\sim$ счетно, и в нем найдется наименьший элемент относительно отношения $<$.

Обозначим наименьший элемент множества $G_L/\sim$ через $M_0 = [b_1, b_2, \ldots, b_k]_\sim$. Нетрудно заметить, что любой базис $(b_1, b_2, \ldots, b_k) \in M_0$ обладает следующим свойством:

1) b_1 — кратчайший ненулевой вектор L;

2) для любого $j \in \{2, \ldots, k\}$ вектор b_j имеет наименьшую длину среди всех таких векторов $b \in L$, что $b_1, b_2, \ldots, b_{j-1}, b$ можно дополнить до базиса L.

Определение: Для решетки $L \subseteq R^n$ размерности $k \leq n$ базис $b_1, b_2, \ldots, b_k \in M_0$ 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 $b_1, b_2, \ldots, b_k$ решетки L приведен по Минковскому, то:

1) b_1 — кратчайший ненулевой вектор L;

2) $||b_1|| \leq ||b_2|| \ldots \leq ||b_k||$;

3) $2|(b_i ; b_j)| \leq \left||b_i|\right|^2, 1 \leq i \leq j \leq k$.

Определение: j-м последовательным минимумом решетки L называется такое наименьшее положительное число $\lambda_j = \lambda_j\ (L), 1 \leq j \leq k$, что найдутся j линейно независимых над векторами решетки L, длина которых не превосходит λ_j.

Алгоритм Гаусса редукции решеток размерности 2:

ВХОД: базис b_1, b_2 решетки $L \in R^n$ размерности 2, упорядоченный таким образом, что $\left||b_1|\right| \leq ||b_2||$.

ВЫХОД: редуцированный по Минковскому базис решетки L.

Шаг 1: Вычислить $r = \left[\frac{(b_1;b_2)}{||b_1||^2} + \frac{1}{2}\right]$**.** Положить $a = b_2 - rb_1$

Шаг 2: Проверить выполнение неравенства $||\text{a}|| < ||b_1||$. Если это неравенство выполнено, то заменить $b_2 = b_1$, $b_1 = \text{a}$ и перейти к шагу 1. В противном случае заменить $b_2 = \text{a}$. Алгоритм заканчивает работу.

Оценим сложность алгоритма:

Лемма: Пусть b_1, b_2 – линейно независимые векторы пространства R^n, такие что $||b_1|| < ||b_2||$. Пусть b – кратчайший ненулевой вектор решетки, порожденный b_1, b_2. Тогда число шагов алгоритма Гаусса, примененного к b_1, b_2, оценивается величиной:

$$O(1 + \log\left(\frac{||b_2||}{||b||}\right))$$

Лемма: Пусть b_1, b_2 – линейно независимые векторы из Z^n, такие что $||b_1|| \leq ||b_2|| \leq M$ для некоторого числа M. Тогда сложность применения алгоритма Гаусса к паре векторов b_1, b_2 равна $O(\log^2 M)$ двоичных операций при $M \to \infty$ и ограниченном n.

В случае работы с машинными словами можно оценить сложность алгоритма как $O(n^2 \log^2 M)$.

Псевдокод реализованного алгоритма:

Начало алгоритма $function\ gauss\_reduce\_lattice\_2d(B1, B2)$

ВХОД:

$B1$: Вектор размерности 2 (например, $[x1, y1]$)

$B2$: Вектор размерности 2 (например, $[x2, y2]$)

ВЫХОД: Редуцированный базис решетки

ШАГ 1:

Если $||B1|| > ||B2||$, то:

Меняем местами $B1$ и $B2$

ШАГ 2:

Вычисляем скалярное произведение $B1$ и $B2 \rightarrow dot\_product$

Вычисляем квадрат нормы $B1 \rightarrow norm\_b1\_squared$

Вычисляем значение $r = dot\_product / norm\_b1\_squared + 0.5$

Обновляем вектор $A = B2 - r * B1$

ШАГ 3:

Если $||A|| < ||B1||$, то:

$(B1, B2) = (A, B1)$

Переходим к Шагу 2

Иначе:

Вернуть $(B1, A)$

Конец алгоритма

Сложность алгоритма: $O(n^2 \log^2 M)$

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

Определение: Пусть L — решетка размерности k в R^n, $n \geq k$. Пусть имеется некоторая константа c_k, которая зависит только от k и не зависит от самой решетки. Назовем базис $b_1, b_2, \ldots, b_k$ решетки L приведенным (редуцированным), если имеет место неравенство $\prod_{i=1}^{k} \|b_i\|^2 \leq c_k \Delta^2(L)$.

Определение: Базис $b_1, b_2, \ldots, b_k$ решетки L называется LLL редуцированным, если выполнены следующие условия:

1) $|\mu_{ij}| \leq 0.5$ при всех $1 \leq j \leq i \leq k$;

2) $\left\|b_i^* + \mu_{i,i-1} \cdot b_{i-1}^*\right\|^2 \geq \frac{3}{4} \|b_{i-1}^*\|^2$ при $1 < i \leq k$.

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 $c_k = 2^{\frac{k(k-1)}{2}}$.

Теорема: Пусть $b_1, b_2, \ldots, b_k - LLL -$ редуцированный базис решетки L. Тогда:

1. $\left||b_j|\right|^2 \leq 2^{i-1} \left||b_i^*|\right|^2$ при всех $1 \leq j < i \leq k$
2. $\Delta(L) \leq \prod_{i=1}^{k} \left||b_i|\right| \leq 2^{\frac{k(k-1)}{4}} \Delta(L)$
3. $\left||b_1|\right| \leq 2^{\frac{k-1}{4}} \Delta^{\frac{1}{k}}(L)$
4. Для любого ненулевого вектора $b \in L$: $\left||b_1|\right|^2 \leq 2^{k-1} \left||b|\right|^2$.

Алгоритм Ленстры-Ленстры-Ловаша:

ВХОД: базис $b_1, b_2, \ldots, b_k$ решетки L.

ВЫХОД: LLL - редуцированный базис решетки L.

Шаг 1: Вычислить ортогонализацию Грамма–Шмидта для системы $b_1, b_2, \ldots, b_k$. Для этого положить $b_1^* = b_1$ и для всех $i \in \{2, \ldots, k\}$ вычислить

$$b_i^* = b_i - \sum_{j=1}^{i-1} \mu_{ij} b_j^*,$$

где

$$\mu_{ij} = \frac{(b_i; b_j^*)}{(b_j^*; b_j^*)}.$$

Положить $B_i = ||b_i^*||^2$. Положить $t = 2$.

Шаг 2: (Уменьшить длины векторов базиса). Выполнить шаг 4 при $l = t - 1$. Если $B_t < \left(\frac{3}{4} - \mu_{t,t-1}^2\right) B_{t-1}$ и $t \geq 2$, то перейти к шагу 3. В противном случае выполнить шаг 4 последовательно при $l = t - 2, t - 3, \ldots, 1$. Если $t = k + 1$, то алгоритм заканчивает работу. В противном случае положить $t = t + 1$. Перейти к шагу 2.

Шаг 3: (Поменять местами b_{t-1}, b_t, изменить соответствующие значения μ_{ij}, B_i). Выполнить следующее:

$$b_{t-1} \leftrightarrow b_t, \mu \leftarrow \mu_{t,t-1}, B \leftarrow B_t + \mu^2 B_{t-1},$$

а также

$$\mu_{t,t-1} \leftarrow \frac{\mu B_{t-1}}{B}, B_t \leftarrow \frac{B_{t-1} B_t}{B}, B_{t-1} \leftarrow B;$$

$$\mu_{t-1,j} \leftrightarrow \mu_{t,j} \text{ при } j = 1,2, \ldots, t-2$$

и

$$(\mu_{i,t-1}, \mu_{t,j}) \leftarrow (\mu_{i,t-1}, \mu_{t,j}) \begin{pmatrix} 0 & 1 \\ 1 & -\mu \end{pmatrix} \begin{pmatrix} 1 & 0 \\ \mu_{t,t-1} & 1 \end{pmatrix}$$

при $i = t + 1, t + 2, \ldots, k$.

Положить $t \leftarrow t - 1$. Перейти к шагу 2.

Шаг 4: Если $|\mu_{tl}| > \frac{1}{2}$, то вычислить $r = [\mu_{tl}]$, положить $b_t = b_t - rb_l$ и $\mu_{tj} = \mu_{tj} - r\mu_{lj}$ при $j = 1,2, \ldots, l-1$ и $\mu_{tl} = \mu_{tl} - r$.

Для обоснования алгоритма надо сделать следующее:

1) проверить, что при перестановке b_{t-1}, b_t значения μ_{ij}, B_i изменяются указанным в шаге 3 способом;

2) проверить, что при замене $b_t = b_t - rb_l$ значения чисел μ_{ij} изменяются указанным в шаге 4 образом, а все B_i не изменяются;

3) алгоритм заканчивается через конечное число шагов

Заметим, что при текущем значении t векторы $b_1, b_2, \dots, b_{t-1}$ $LLL -$ редуцированы. Отсюда следует, что алгоритм LLL – редуцированный базис решетки L.

Оценка сложности строится на следующей теореме.

Теорема: Пусть $L \in Z^n$ решетка размерности k с базисом $b_1, b_2, \dots, b_k$, где $||b_i|| \leq M$ для всех $i \in \{1, \dots, k\}$. Тогда сложность применения LLL алгоритма к этому базису не больше $O(n^4 logM)$ арифметических операций с целыми числами, двоичная длина которых не больше $O(nlogM)$.

Псевдокод реализованного алгоритма

Начало алгоритма

ВХОД:

Базис решетки $(B1, B2, \dots, Bd)$

Параметр Δ $(0.25 < \Delta < 1)$, часто $\Delta = 3/4$

ВЫХОД:

LLL-редуцированный базис решетки (L)

ШАГ 1:

Установить $K = 1$ # Начинаем с первого индекса

Пока $K < D$:

Для каждого J от $K - 1$ до 0 с шагом -1:

Вычисляем скалярное произведение $\mu = (Bk, Bj)$

Если $|\mu| > 0.5$:

Округлить μ до ближайшего целого R

Обновить вектор Bk: $Bk = Bk - R * Bj$

Обновить значения μ и Bk после изменения

ШАГ 2:

Вычисляем квадрат нормы ортогонального базиса: $||Bk||$^2

Вычисляем значение левой части $(LHS) = ||Bk||$^2

Вычисляем значение правой части $(RHS) = (\Delta - \mu^2) * ||B(k - 1)||$^2

Если $LHS \geq \quad RHS$:

Увеличиваем K на 1

Иначе:

Меняем местами Bk и $B(k-1)$

Обновляем ортогональный базис и μ

Устанавливаем $K = max(K - 1, 1)$

ШАГ 3:

Вернуть редуцированный базис $(B1, B2, \ldots, Bn)$

Конец алгоритма

Сложность реализованного алгоритма: $O(n^4)$

4 Алгоритм решения задачи целочисленного программирования с помощью LLL-алгоритма

Рассмотрим следующую задачу. Найти все такие целочисленные векторы $(x_1, \dots, x_k)$, что

$$\begin{gathered} c_{12} \le b_{21}x_1 + b_{22}x_2 + \dots + b_{2k}x_k \le c_{22} \\ \dots \\ c_{1n} \le b_{n1}x_1 + b_{n2}x_2 + \dots + b_{nk}x_k \le c_{2n} \end{gathered} \tag{17}$$

при заданных c_{1i}, c_{2i}, b_{ij}, $1 \le I \le n, 1 \le j \le k \le n$, таких что ранг матрицы $B = (b_{ij})$ размера $n * k$ равен k. Заметим, что задача легко сводится к случаю, когда $k = n$. Действительно, так как ранг матрицы B равен k, то матрица содержит ненулевой минор порядка k. 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 $b_1, b_2, \dots, b_n$ решетки L размерности n в R^n, множество K, определенное системой неравенств $K = (y_1, y_2, \dots, y_n) \in R^n \mid c_{1_i} \le y_i \le c_{2_i}$, $1 \le i \le n$

ВЫХОД: элементы множества $K \cap L$

Шаг 1. Вычислить приведенный базис решетки L. Пусть $b'_1, b'_2, \dots, b'_n$— приведенный базис L, т. е. существует постоянная c_n, которая зависит лишь от n, что

$$\prod_{i=1}^{n} | b'_i | \le c_n \Delta(L)$$

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

Далее вместо (17) решим систему неравенств

$$c_{11} \le b'_{11}z_1 + b'_{12}z_2 + \dots + b'_{1n}z_n \le c_{21};$$
$$c_{12} \le b'_{21}z_1 + b'_{22}z_2 + \dots + b'_{2n}z_n \le c_{22};$$

$$\ldots \qquad (19)$$

$$c_{1n} \le b'_{n1}z_1 + b'_{n2}z_2 + \cdots + b'_{nn}z_n \le c_{2n},$$

Где

$$\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = U \begin{pmatrix} z_1 \\ \vdots \\ z_n \end{pmatrix} \qquad (20)$$

и U целочисленная обратимая над Z матрица размера $n * n$. Это матрица перехода от исходного базиса решетки к приведенному базису.

Шаг 2. Вычислить ортогонализацию Грамма–Шмидта $b_1^{*}{}', \ldots, b_n^{*}{}'$ базиса $b_1{}', \ldots, b_n{}'$. Положить $h = \|b_n'^{*}\|$.

Шаг 3. Положить

$$P = \left(\frac{c_{11} + c_{21}}{2}, \ldots, \frac{c_{1n} + c_{2n}}{2}\right)$$

Представить p через базис $b_1{}', \ldots, b_n{}'$:

$p = t' b_1' + \ldots + t_n b_n'$, где $t_i \in R$

Положить

$$R = \max\left\{\frac{c_{2i} - c_{1i}}{2}\sqrt{n}\right\} \text{ и } s = \left\lfloor\frac{R}{n}\right\rfloor$$

Для каждого $z_n \in \{\lfloor t\rceil - s, \ldots, \lfloor t\rceil\ -\ s + 1\}$ выполнить шаг 4.

Шаг 4. От системы (19) перейти к системе

$$c_{11} - b'_{1n}z_n \le b'_{11}z_1 + b'_{12}z_2 + \cdots + b'_{1n-1}z_{n-1} \le c_{21} - b'_{1n}z_n;$$

$$c_{12} - b'_{2n}z_n \le b'_{21}z_1 + b'_{22}z_2 + \cdots + b'_{2n-1}z_{n-1} \le c_{22} - b'_{2n}z_n;$$

$$\ldots \qquad (21)$$

$$c_{1n} - b'_{2n}z_n \le b'_{n1}z_1 + b'_{n2}z_2 + \cdots + b'_{nn-1}z_{n-1} \le c_{2n} - b'_{2n}z_n,$$

Пусть составляют столбцы матрицы B' размера $n * n - 1$. Найти ненулевой минор порядка $n - 1$ матрицы B. Пусть $i_1, i_2, \ldots, i_{n-1}$ — номера строк матрицы B', которые содержат этот минор. Рассмотрим задачу вычисления всех решений системы

$$c_{i_1 1} - b'_{i_1 n}z_n \le b'_{i_1 1}z_1 + b'_{i_1 2}z_2 + \cdots + b'_{i_1 n-1}z_{n-1} \le c_{2i_1} - b'_{i_1 n}z_n;$$

$$c_{i_2 2} - b'_{i_2 n}z_n \le b'_{i_2 1}z_1 + b'_{i_2 2}z_2 + \cdots + b'_{i_2 n-1}z_{n-1} \le c_{i_2 2} - b'_{i_2 n}z_n;$$

$$\ldots$$

$$c_{i_{n-1}n} - b'_{i_{n-1}n} z_n \le b'_{i_{n-1}1} z_1 + b'_{i_{n-1}2} z_2 + \dots + b'_{i_{n-1}} z_{n-1} \le c_{i_{n-1}n} - b'_{i_{n-1}n} z_n,$$

Эта система составлена из неравенств системы (21) с номерами $i_1, i_2, \dots, i_{n-1}$. Для решения этой системы от $n-1$ переменных $z_1, z_2, \dots, z_{n-1}$ применим рекурсивно алгоритм.

Шаг 5. Из соотношений (20) найдем $x_1, x_2, \dots, x_n$. 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 K содержится в шаре $V(p, R)$ радиуса R с центром в точке p. Действительно, этот шар описан вокруг n мерного куба со стороной, равной $\max\limits_{1 \le i \le n}\{c_{2i} - c_{1i}\}$ и с центром в точке p. Пусть H — гиперплоскость в R^n, которая порождается векторами $b_1', \dots, b_{n-1}'$ и L' есть решетка размерности $n-1$ с базисом $b_1', \dots, b_{n-1}'$. Таким образом, $L' \subseteq H$, и решетка L содержится в счетном объединении гиперплоскостей

$$L = \bigcup_{z_n \in \mathbb{Z}} (L' + z_n b_n) \subseteq \bigcup_{z_n \in \mathbb{Z}} (H + z_n b_n')$$

Множество $K \cap L$ содержится в объединении только тех гиперплоскостей $H + z_n b_n'$, которые имеют непустое пересечение с $V(p, R)$, так как $K \subseteq V(P, R)$. Расстояние между гиперплоскостями равно $h = \|b_n'^*\|$. Поэтому не более $\frac{2R}{h} + 1$ последовательных гиперплоскостей пересекают шар $V(p, R)$. Все эти гиперплоскости содержатся среди таких, $H + z_n b_n'$ что $z_n \in \{\lfloor t_n \rceil - s, \dots, \lfloor t_n \rceil - s + 1\}$. Отсюда легко следует корректность алгоритма. Заметим, что $\Delta(L) = h\Delta(L')$. Поэтому по свойству приведенного базиса имеем

$$\prod_{1=1}^{n} \|b_i'\| \le c_n \Delta(L) = c_n h \Delta(L') \le c_n h \prod_{1=1}^{n} \|b_i'\|$$

Отсюда $h \ge c_n^{-1} \|b_n'\|$. Значит, число вариантов z_n, которые перебираются на шаге 3, ограничено величиной $\frac{2R}{n} + 1 \le \frac{2c_n R}{\|b_n'\|} + 1$. В конкретных вычислениях (в случае неприведенного базиса), длина вектора b_n

велика, а величина h может быть мала. Это может привести к значительному увеличению трудоемкости вычислений, если пользоваться неприведенным базисом.

Сложность алгоритма: $O(n^3)$

Псевдокод алгоритма

Начало алгоритма

ВХОД:

$C1$: Вектор нижних границ

$C2$: Вектор верхних границ

B: Матрица базиса решетки

N: Размерность пространства

ВЫХОД:

Список возможных решений на решетке

ШАГ 1:

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

Вычисляем произведение норм ортогонализированных векторов:

CN = произведение(норм $B\_reduced[i]$ для i от 0 до N)

ШАГ 2:

$solutions\ =\ []$ # Создаем список для хранения границ возможных решений

Для i от 0 до $N-1$:

Вычисляем скалярное произведение ортогонализированных векторов:

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

Вычисляем нижнюю границу для i-го измерения:

$lower\_bound$ = округление_вверх($(C1[i] - dot\_product)$ / $B\_reduced[i][i]$)

Вычисляем верхнюю границу для i-го измерения:

$upper\_bound$ = округление_вниз($(C2[i] - dot\_product)$ / $B\_reduced[i][i]$)

Если $lower\_bound$ — это скалярное значение:

$lower\_bound\_scalar = lower\_bound$

Иначе:

$lower\_bound\_scalar$ = первый элемент $lower\_bound$

Если $upper\_bound$ — это скаляр:

$upper\_bound\_scalar = upper\_bound$

Иначе:

$upper\_bound\_scalar$ = первый элемент $upper\_bound$

Добавить в список solutions пару $(lower\_bound\_scalar, upper\_bound\_scalar)$

ШАГ 3:

$P = (C1 + C2) / 2$ # Находим среднюю точку между границами

R = максимум(абсолютных значений $(C2[i] - C1[i]) / 2$ для i от 0 до N) # Вычисляем радиус

ШАГ 4:

Функция $recursive\_solve(solutions, depth)$:

Если $depth == 0$:

Вернуть список с нулевым вектором длины depth

$sub\_solutions = recursive\_solve(solutions$ до $depth - 1, depth - 1)$

$results = []$

Для каждого sol в $sub\_solutions$:

Для каждого z в диапазоне от $solutions[depth - 1][0]$ до $solutions[depth - 1][1]$:

new_sol = добавить z в конец sol

Добавить $new\_sol[: depth]$ в список $results$

Вернуть список $results$

$all\_solutions = recursive\_solve(solutions, N)$ # Находим все возможные решения

ШАГ 5:

$final\_solutions = []$

Для каждого sol в $all\_solutions$:

Если произведение матрицы B на вектор sol удовлетворяет условиям $C1 <= np.dot(B, sol) <= C2$:

Добавить sol в список $final\_solutions$

Вернуть $final\_solutions$

Конец алгоритма

Сложность алгоритма: $O(n^3)$

5 Тестирование реализованных алгоритмов

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

```
Введите тип операции:

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

:>1
Алгоритм Гаусса редукции решеток размерности 2
Введите базис b1: 11 10 1
Введите базис b2: 23 21 2
Редуцированный базис:
b1: 1 1 0
b2: 0 -1 1
:>1
Алгоритм Гаусса редукции решеток размерности 2
Введите базис b1: 2 1
Введите базис b2: 3 2
Редуцированный базис:
b1: -1 0
b2: 1 1
:>1
Алгоритм Гаусса редукции решеток размерности 2
Введите базис b1: 25 37
Введите базис b2: 14 21
Редуцированный базис:
b1: -3 -5
b2: 5 6
:>
```

Рисунок 1 – Тестирование алгоритма Гаусса редукции решеток размерности 2

```
Введите тип операции:

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

:>2
Алгоритм Ленстры-Ленстры-Ловаша (LLL-алгоритм)
Укажите количество базисов: 3
Введите базис 1: 2 2 3 1
Введите базис 2: 7 7 10 3
Введите базис 3: 11 10 14 4
LLL-редуцированный базис:
[[1. 0. 0. 0.]
 [0. 0. 1. 1.]
 [0. 1. 1. 0.]]
:>2
Алгоритм Ленстры-Ленстры-Ловаша (LLL-алгоритм)
Укажите количество базисов: 3
Введите базис 1: 1 1 1
Введите базис 2: -1 0 2
Введите базис 3: 3 5 6
LLL-редуцированный базис:
[[ 0.  1.  0.]
 [ 1.  0.  1.]
 [-1.  0.  2.]]
:>2
Алгоритм Ленстры-Ленстры-Ловаша (LLL-алгоритм)
Укажите количество базисов: 3
Введите базис 1: 1 1 1
Введите базис 2: 1 0 2
Введите базис 3: 0 1 1
LLL-редуцированный базис:
[[-1.  0.  0.]
 [ 0. -1.  1.]
 [ 0.  1.  1.]]
:>
```

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

```
Введите тип операции:

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

:>3
Алгоритм решения задачи целочисленного программирования с помощью LLL-алгоритма
Укажите количество базисов: 3
Введите базис 1: 1 0 0
Введите базис 2: 0 1 0
Введите базис 3: 0 0 1
Введите левые границы: 1 2 3
Введите правые границы: 4 5 6
1) [3, 5, 6]
2) [3, 5, 5]
3) [3, 5, 4]
4) [3, 5, 3]
5) [3, 4, 6]
6) [3, 4, 5]
7) [3, 4, 4]
8) [3, 4, 3]
9) [3, 3, 6]
10) [3, 3, 5]
11) [3, 3, 4]
12) [3, 3, 3]
13) [3, 2, 6]
14) [3, 2, 5]
15) [3, 2, 4]
16) [3, 2, 3]
17) [2, 5, 6]
18) [2, 5, 5]
19) [2, 5, 4]
```

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 АРВ, 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"
```