МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Национальный исследовательский
Нижегородский государственный университет им. Н.И. Лобачевского»
(ННГУ)**

Институт информационных технологий, математики и механники

Кафедра: алгебры, геометрии и дискретной математики

Направление подготовки: «Программная инженерия»
Профиль подготовки: «Разработка программно-информационных систем»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

на тему:
«Нахождение Эрмитовой нормальной формы и решение проблемы ближайшего вектора решетки»

Выполнил(а): студент(ка) группы ____
_____ Д.В. Огнев

Подпись

Научный руководитель:
Доцент, кандидат физико-математических наук

_____ С.И. Весёлов

Подпись

Нижний Новгород
2022

Аннотация (ДОПИСАТЬ)

Тема выпускной квалификационной работы бакалавра – «Нахождение Эрмитовой нормальной формы и решение проблемы ближайшего вектора решетки».

Ключевые слова: решетки, Эрмитова нормальная форма, проблема ближайшего вектора, криптография.

Данная работа посвящена изучению задач теории решеток и методов их решения. В работе изложены основные понятия, связанные с решетками, исследованы алгоритмы для нахождения Эрмитовой нормальной формы и решения проблемы ближайшего вектора решетки, разработана программная реализация разобранных алгоритмов и дан краткий анализ временной сложности алгоритмов.

Целью работы является рассмотрение и программная реализация алгоритмов для решения задач теории решеток. Для успешного достижения цели поставленной цели необходимо разобрать теоретические основы и описание алгоритмов, определить необходимые программные инструменты, научиться эффективно их использовать и получить программную реализацию.

Объем работы - ...

Содержание

1. Список условных обозначений и сокращений (TODO)

ПБВ (CVP) – проблема ближайшего вектора (Closest vector problem)

ЭНФ (HNF) – Эрмитова нормальная форма (Hermite normal form)

B&B – Branch and bound

GMP – GNU Multiprecision Library

G++ – GNU C++

2. Введение (TODO)

Криптография занимается разработкой методов преобразования (шифрования) информации с целью ее зашиты от незаконных пользователей. Самыми известными вычислительно трудными задачами считаются проблема вычисления дискретного логарифма и факторизация (разложение на множители) целых чисел. Для этих задач неизвестны эффективные (работающие за полиномиальное время) алгоритмы. С развитием квантовых компьютеров было показано существование полиномиальных алгоритмов решения задач дискретного логарифмирования и разложения числа на множители на квантовых вычислителях, что заставляет искать задачи, для которых неизвестны эффективные квантовые алгоритмы. В области постквантовой криптографии фаворитом считается криптография на решетках. Считается, что такая криптография устойчива к квантовым компьютерам.

Объектом исследования данной работы являются алгоритмы для нахождения Эрмитовой нормальной формы и решения проблемы ближайшего вектора. Целью работы является получение программной реализации алгоритмов для нахождения ЭНФ за полиномиальное время, приблизительного решения ПБВ за полиномиальное время и точного решения ПБВ за суперполиномиальное время. Необходимо будет показать, как можно использовать данные алгоритмы на практике. В качестве основы, откуда взяты теоретические основы и описание алгоритмов для программирования, будем использовать серию лекций по основам алгоритмов на решетках и их применении.

3. Основные определения (TODO)

Матрица – прямоугольная таблица чисел, состоящая из n столбцов и m строк. Обозначается полужирной заглавной буквой, а ее элементы - строчными с двумя индексами (строка и столбец). При программировании использовалась стандартная структура хранения матриц:

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

Квадратная матрица – матрица, у которой число строк равно числу столбцов $m = n$.

Единичная матрица – матрица, у которой диагональные элементы $(i = j)$ равны единице.

Невырожденная матрица – квадратная матрица, определитель которой отличен от нуля.

Вектор – если матрица состоит из одного столбца $(n = 1)$, то она называется вектором-столбцом. Если матрица состоит из одной строки $(m = 1)$, то она называется вектором-строкой. Матрицы можно обозначать через вектора-столбцы и через вектора-строки: $\mathbf{A} = \begin{bmatrix} \mathbf{a}_1 & \dots & \mathbf{a}_n \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1^{\mathbf{T}} \\ \vdots \\ \mathbf{a}_m^{\mathbf{T}} \end{bmatrix}$.

Линейная зависимость и независимость – пусть имеется несколько векторов одной размерности $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_k$ и столько же чисел $\alpha_1, \alpha_2, \cdots, \alpha_k$. Вектор $\mathbf{y} = \alpha_1\mathbf{x}_1 + \alpha_2\mathbf{x}_2 + \ldots + \alpha_k\mathbf{x}_k$ называется линейной комбинацией векторов $\mathbf{x}_k$. Если существуют такие числа $\alpha_i, i = 1, \dots, k$, не все равные нулю, такие, что $\mathbf{y} = \mathbf{0}$, то такой набор векторов называется линейно зависимым. В противном случае векторы называются линейно независимыми.

Ранг матрицы – максимальное число линейно независимых векторов. Матрица называется матрицей с полным рангом строки, когда все строки матрицы линейно независимы. Матрица называется матрицей с полным рангом столбца, когда все столбцы матрицы линейно независимы.

Решетка - пусть $\mathbf{B} = [\mathbf{b}_1, \dots, \mathbf{b}_n] \in \mathbb{R}^{d \times n}$ - линейно независимые вектора из $\mathbb{R}^d$. Решетка, генерируемая от $\mathbf{B}$ есть множество

$$\mathcal{L}(\mathbf{B}) = \{\mathbf{Bx} : \mathbf{x} \in \mathbb{Z}^n\} = \left\{ \sum_{i=1}^{n} x_i \cdot \mathbf{b}_i : \forall i \; x_i \in \mathbb{Z} \right\}$$

всех целочисленных линейных комбинаций столбцов матрицы $\mathbf{B}$. Матрица $\mathbf{B}$ называется базисом для решетки $\mathcal{L}(\mathbf{B})$. Число n называется рангом решетки. Если $n = d$, то решетка $\mathcal{L}(\mathbf{B})$ называется решеткой полного ранга или полноразмерной решеткой в $\mathbb{R}^d$.

Эрмитова нормальная форма - невырожденная матрица $\mathbf{B} = [\mathbf{b}_1, \dots, \mathbf{b}_n] \in \mathbb{R}^{m \times n}$ является Эрмитовой нормальной формой, если

- Существует $1 \leq i_1 < \ldots < i_h \leq m$ такое, что $b_{i,j} \neq 0 \Rightarrow (j < h) \wedge (i \geq i_j)$ (строго убывающая высота столбца).

- Для всех $k > j, 0 \leq b_{i_j,k} < b_{i_j,j}$, т.е. все элементы в строках i_j приведены по модулю $b_{i_j,j}$.

4. Постановка задачи (TODO)

Цель работы - реализовать алгоритмы для нахождения ЭНФ и решения ПБВ за полиномиальное и суперполиномиальное время. Для достижения этой цели необходимо решить следующие задачи:

- Изучить теоретические основы для программирования алгоритмов.
- Найти необходимые инструменты для программной реализации, научиться их эффективно использовать.
- Написать программу, в которой будут реализованы разобранные алгоритмы. Полученная программа должна быть использована как подключаемая библиотека.
- Полученную библиотеку использовать для решения задач теории решеток и найти практическое применение.

5. Обзор литературных источников (TODO)

В результате работы был получен перевод статьи, описывающей необходимые алгоритмы и их применение.

6. Обзор инструментов (TODO)

Для программной реализации был выбран язык C++. Приоритет этому языку был отдан из-за его скорости, статической типизации, большому количеству написанных библиотек и богатой стандартной библиотеке. Сборка проекта осуществляется с помощью системы сборки CMake, при сборке можно указать флаги `BUILD_DOCS` – для сборки документа выпускной квалификационной работы, написанной в формате LaTeX, `BUILD_PARALLEL_BB` – для сборки параллельной реализации алгоритма Branch and Bound и `BUILD_GMP` – для использования GMP. Для работы с матрицами была выбрана библиотека Eigen, для работы с большими числами используется часть библиотеки Boost – Boost.Multiprecision, которая подключается в режиме Standalone. Используется встроенная в Boost реализация больших чисел и реализация от GMP.

Используется система контроля версий Git и сервис Github, все исходные файлы проекта доступны в онлайн репозитории. Для подключения Boost.Multiprecision используются модули Git.

6.1. Обзор библиотеки Eigen

Eigen - библиотека для работы с линейной алгеброй. Предоставляет шаблонные классы и методы для работы с матрицами, векторами и связанными алгоритмами. Является header-only библиотекой и не требует отдельной компиляции. Для работы не требует других библиотек, кроме стандратной.

Все необходимые классы находятся в заголовочном файле `Eigen/Dense` и подключаются командой `#include <Eigen/Dense>`. Для их использования необходимо указывать пространство имен `Eigen`, например `Eigen::Matrix2d`.

Используемые классы:

`Matrix<typename Scalar, int RowsAtCompileTime, int ColsAtCompileTime>` – шаблонный класс матрицы. Первый параметр шаблона отвечает за тип элементов матрицы, второй параметр за количество строк, третий за количество столбцов. Если количество строк/столбцов неизвестно на этапе компиляции, а будет найдено в процессе выполнения программы, то необходимо ставить количество строк/столбцов равным `Eigen::Dynamic`, либо `-1`. Имеет псевдонимы для различных встроенных типов (int, double, float) и размеров матриц (2, 3, 4), например `Matrix3d` – матрица элементов `double` размера `3x3`.

`Vector` и `RowVector` – вектор-столбец и вектора-строка, являются псевдонимами класса матриц, в которых количество строк или столбцов равно единице соответственно. Используются псевдонимы для различных встроенных типов (int, float, double) и размеров векторов (2, 3, 4), например `Vector2f` – вектор, состоящий из элементов `float` размера `3`.

С матрицами и векторами можно производить различные арифметические действия, например складывать и вычитать между собой, умножать и делить между собой и на скаляр. Все

действия должны осуществляться по правилам линейной алгебры.

Используемые методы:

`matrix.rows()` – получение количества строк.

`matrix.cols()` – получение количества столбцов.

`vector.norm()` – длина вектора.

`vector.squaredNorm()` – квадрат длины вектора.

`matrix << elems` – comma-инициализация матрицы, можно вставлять скалярные типы, матрицы, вектора.

`Eigen::MatrixXd::Identity(m, m)` – получение единичной матрицы размера $m \times m$.

`Eigen::VectorXd::Zero(m)` – получение нулевого вектора размера m.

`matrix.row(index)` – получение строки матрицы по индексу.

`matrix.col(index)` – получение столбца матрицы по индексу.

`matrix.row(index) = vector` – установить строку матрицы значениями вектора.

`matrix.col(index) = vector` – установить столбец матрицы значениями вектора.

`matrix.block(startRow, startCol, endRow, endCol)` – получение подматрицы по индексам.

`matrix.block(startRow, startCol, endRow, endCol) = elem` – установка блока матрицы по индексам значением elem.

`matrix.cast<type>()` – привести матрицу к типу type.

`vector1.dot(vector2)` – скалярное произведение двух векторов.

`vector.tail(size)` – получить с конца вектора size элементов.

`matrix(i, j)` – получение элемента матрицы по индексам.

`vector(i)` – получение элемента вектора по индексу.

`matrix(i, j) = elem` – установка элемента матрицы по индексам значением elem.

`vector(i) = elem` – установка элемента вектора по индексу значением elem.

`for (const Eigen::VectorXd &vector : matrix.colwise())` – цикл по столбцам матрицы.

`for (const Eigen::VectorXd &vector : matrix.rowwise())` – цикл по строкам матрицы.

6.2. Обзор библиотеки Boost.Multiprecision

Boost.Multiprecision – часть библиотеки Boost, подключается в режиме Standalone и не требует подключения основной библиотеки, что позволяет не использовать модули, которые не требуются и уменьшить итоговый размер. Все классы находятся в пространстве имен `boost::multiprecision`. Для подключения используется директива препоцессора `#include <boost/multiprecision/cpp_тип.hpp>`. Если при сборке CMake будет указан флаг `BUILD_GMP=ON`, то будет использована обертка от Boost над библиотекой GMP. Классы, связанные с GMP, под-

ключаются с помощью `#include <boost/multiprecision/gmp.hpp>`. В документации Boost указано, что реализация GMP работает быстрее.

Библиотека предоставляет классы для работы с целыми, рациональными числами и числами с плавающей запятой неограниченной точности. Размер этих чисел ограничен только количеством оперативной памяти.

Используемые классы:

`cpp_int` – класс целых чисел.

`cpp_rational` – класс рациональных чисел.

`cpp_bin_float_double` – класс чисел с плавающей запятой с увеличенной точностью.

`mpz_int` – класс целых чисел, использующий реализацию GMP.

`mpq_rational` – класс рациональных чисел, использующий реализацию GMP.

`mpf_float_50` – класс чисел с плавающей запятой, использующий реализацию GMP.

Используемые методы:

`sqrt(int)` – квадратный корень из целого числа.

`numerator(rational)` – числитель рационального числа.

`denominator(rational)` – знаменатель рационального числа.

7. Нахождение ЭНФ (TODO)

Для нахождения ЭНФ будет разобрано два алгоритма - для матриц с полным рангом строки и общий (для любых матриц), который сводится к использованию первого алгоритма. Оба алгоритма предполагают использование ортогонализации Грама-Шмидта, поэтому предварительно будет дано его описание.

7.1. Ортогонализация Грама-Шмидта

Любой базис $\mathbf{B}$ может быть преобразован в ортогональный базис для того же векторного пространства используя алгоритм ортогонализации Грама-Шмидта. Предположим у нас есть набор векторов $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_n]$, $\mathbf{B} \in \mathbb{R}^{m \times n}$. Этот набор необзятельно ортогонален или даже линейно независим. Ортогонализацией этого набора векторов является набор векторов $\mathbf{B}^* = [\mathbf{b}_1^*, \cdots, \mathbf{b}_n^*] \in \mathbb{R}^{m \times n}$, где

$$\mathbf{b}_i^* = \mathbf{b}_i - \sum_{i<j} \mu_{i,j} \mathbf{b}_j^*, \text{ где } \mu_{i,j} = \frac{\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle}{\langle \mathbf{b}_j^*, \mathbf{b}_j^* \rangle}, i = 1, \ldots, n, j = 1, \ldots, i$$

Полученный набор векторов может не являться базисом для решетки, сгенерированной от исходного набора векторов, т.к. точки этой решетки могут не входить в решетку от ортогонализованного базиса. Этот набор также обладает важным свойством, которое мы будем использовать: если вектор $\mathbf{b}_i^* = \mathbf{0}$, то этот вектор линейно зависим от других векторов в наборе и может быть представлен линейной комбинацией этих векторов.

Для процесса ортогонализации Грама-Шмидта нельзя сделать параллельную реализацию, так как каждая следующая итерация требует данные, найденные на предыдущем шаге. Но можно ускорить ее нахождение, путем параллельного нахождения суммы $\sum_{i<j} \mu_{i,j} \mathbf{b}_j^*$. Конечный алгоритм выглядит следующим образом:

Input: $\mathbf{B}$
Output: $\mathbf{GS}$
  $\mathbf{GS} \leftarrow [\,]$
  $n \leftarrow \mathbf{B}.columns$
  for $i \leftarrow 0$ to n **do**
    $\mathbf{b}_i \leftarrow \mathbf{B}.column(i)$
    projections $\leftarrow \mathbf{0}$
    for $j \leftarrow 0$ to i **do**
      $\mathbf{b}_j \leftarrow \mathbf{GS}.column(j)$
      projections $\leftarrow$ **projections** $+ \mathbf{b}_j \cdot \frac{\langle \mathbf{b}_i, \mathbf{b}_j \rangle}{\langle \mathbf{b}_j, \mathbf{b}_j \rangle}$
    end for
    $\mathbf{GS}.push\_back(\mathbf{b}_i -$ **projections**$)$
  end for

7.2. Алгоритм для матриц с полным рангом строки

Дана матрица $\mathbf{B} \in \mathbb{Z}^{m \times n}$. Основная идея состоит в том, чтобы найти ЭНФ $\mathbf{H}$ подрешетки от $\mathcal{L}(\mathbf{B})$, и затем обновлять $\mathbf{H}$, включая столбцы $\mathbf{B}$ один за другим. Предположим, что у нас есть процедура AddColumn, которая работает за полиномиальное время и принимает на вход квадратную невырожденную ЭНФ матрицу $\mathbf{H} \in \mathbb{Z}^{m \times m}$ и вектор $\mathbf{b} \in \mathbb{Z}^m$, а возвращает ЭНФ матрицы $[\mathbf{H}|\mathbf{b}]$. Такая процедура должна следить, чтоб выходная матрица подоходила под определение ЭНФ, что будет показано в описании этой процедуры. ЭНФ от $\mathbf{B}$ может быть вычислено следующим образом:

1. Применить алгоритм Грама-Шмидта к столбцам $\mathbf{B}$, чтобы найти m линейно независимых столбцов. Пусть $\mathbf{B}'$ - матрица размера $m \times m$, заданная этими столбцами.

2. Вычислить $d = \det(\mathbf{B}')$, используя алгоритм Грама-Шмидта или любую другую процедуру с полиномиальным временем. Пусть $\mathbf{H}_0 = d \cdot \mathbf{I}$ будет диагональной матрицей с d на диагонали.

3. Для $i = 1, \ldots, n$ пусть $\mathbf{H}_i$ – результат применения AddColumn к входным $\mathbf{H}_{i-1}$ и $\mathbf{b}_i$.

4. Вернуть $\mathbf{H}_n$.

Разберем подпункты:

1. Необходимо найти линейно независимые столбцы матрицы. Их количество всегда будет равно m, т.к. наша матрица полного ранга строки и ранг матрицы равен m, а значит матрица, состоящая из этих столбцов, будет размера $m \times m$. Для нахождения этих строк можно использовать алгоритм ортогонализации Грама-Шмитда: если $\mathbf{b}_i^* = \mathbf{0}$, то i-ая строка является линейной комбинацией других строк, и ее необходимо удалить. Реализация данного алгоритма находится в пространстве имен Utils в функции `get_linearly_independent_columns_by_gram_schmidt`. Полученная матрица будет названа $\mathbf{B}'$.

2. Необходимо вычислить d, будем вычислять его по следующей форумле: $d = \sqrt{\prod_i \|\mathbf{b}_i^*\|^2}$ - сумма произведений квадратов длин всех столбцов, полученных после применения ортогонализации Грама-Шмидта. Матрица $\mathbf{H_0}$ будет единичной матрицей размера $m \times m$, умноженной на определитель. В результате все диагональные элементы будут равны d.

3. Применяем AddColumn (реализация находится в функции `add_column`) к $\mathbf{H}_0$ и первому столбцу матрицы $\mathbf{B} - \mathbf{b}_0$, получаем $\mathbf{H}_1$; повторяем для всех оставшихся столбцов, получаем $\mathbf{H}_n$.

4. $\mathbf{H}_n$ является ЭНФ($\mathbf{B}$).

Алгоритм AddColumn на вход принимает квадратную невырожденную ЭНФ матрицы $\mathbf{H} \in \mathbb{Z}^{m \times m}$ и вектор $\mathbf{b} \in \mathbb{Z}^m$ и работает следующим образом. Если $m = 0$, то возвращаем $\mathbf{H}$. В противном случае, пусть $\mathbf{H} = \begin{bmatrix} a & \mathbf{0}^T \\ \mathbf{h} & \mathbf{H}' \end{bmatrix}$ и $\mathbf{b} = \begin{bmatrix} b \\ \mathbf{b}' \end{bmatrix}$ и дальше:

1. Вычислить $g = \text{НОД}\,(a, b)$ и целые x, y такие, что $xa + yb = g$, используя расширенный НОД алгоритм.
2. Применить унимодулярное преобразование $\mathbf{U} = \begin{bmatrix} x & (-b/g) \\ y & (a/g) \end{bmatrix}$ к первому столбцу из $\mathbf{H}$ и $\mathbf{b}$ чтобы получить $\begin{bmatrix} a & b \\ \mathbf{h} & \mathbf{b}' \end{bmatrix} \mathbf{U} = \begin{bmatrix} g & 0 \\ \mathbf{h}' & \mathbf{b}'' \end{bmatrix}$
3. Добавить соответствующий вектор из $\mathcal{L}(\mathbf{H}')$ к $\mathbf{b}''$, чтобы сократить его элементы по модулю диагональных элементов из $\mathbf{H}'$.
4. Рекурсивно вызвать AddColumn на вход $\mathbf{H}'$ и $\mathbf{b}''$ чтобы получить матрицу $\mathbf{H}''$.
5. Добавить соответствующий вектор из $\mathcal{L}(\mathbf{H}'')$ к $\mathbf{h}'$ чтобы сократить его элементы по модулю диагональных элементов из $\mathbf{H}''$.
6. Вернуть $\begin{bmatrix} g & \mathbf{0}^T \\ \mathbf{h}' & \mathbf{H}'' \end{bmatrix}$

Разберем подпункты:

1. Функция extended_gcd принимает a, b , вычисляет наибольший общий делитель и целые x, y такие, что $xa + yb = g$
2. Составляем матрицу $\mathbf{U} = \begin{bmatrix} x & (-b/g) \\ y & (a/g) \end{bmatrix}$ и умножаем ее на матрицу, составленную из первого столбца $\mathbf{H}$ и столбца $\mathbf{b}$, чтобы получить
$$\begin{bmatrix} a & b \\ \mathbf{h} & \mathbf{b}' \end{bmatrix} \mathbf{U} = \begin{bmatrix} g & 0 \\ \mathbf{h}' & \mathbf{b}'' \end{bmatrix}$$
3. Функция `reduce` принимает на вход матрицу и вектор, получает необходимый вектор из решетки от матрицы на входе, чтобы сократить элементы вектора по модулю диагональных элементов из матрицы. Применяем функцию `reduce` к $\mathbf{H}'$ и $\mathbf{b}$
4. Рекурсивно вызываем AddColumn, на вход отправляем $\mathbf{H}'$ и $\mathbf{b}''$ получаем матрицу $\mathbf{H}''$.
5. Вызываем функцию `reduce` к $\mathbf{H}''$ и $\mathbf{h}'$
6. Составляем необходимую матрицу и возвращаем $\begin{bmatrix} g & \mathbf{0}^T \\ \mathbf{h}' & \mathbf{H}'' \end{bmatrix}$

7.3. Общий алгоритм для любых матриц

1. Запустить процесс ортогонализации Грама-Шмидта к строкам $\mathbf{r}_1, \ldots, \mathbf{r}_m$ из $\mathbf{B}$, и пусть $\mathrm{K} = \{\mathrm{k}_1, \ldots, \mathrm{k}_l\}\,(\mathrm{k}_1 < \ldots < \mathrm{k}_l)$ – это множество индексов, такое, что $\mathbf{r}^*_{\mathrm{k}_i} \neq \mathbf{0}$. Определим операцию проецирования $\prod_{\mathrm{K}} : \mathbb{R}^m \to \mathbb{R}^l$ при $[\prod_{\mathrm{K}}(\mathbf{x})]_i = \mathrm{x}_{\mathrm{k}_i}$. Заметим, что строки $\mathbf{r}_{\mathrm{k}}$ $(\mathrm{k} \in \mathrm{K})$ линейно независимы и любая строка $\mathbf{r}_i$ $(i \in \mathrm{K})$ может быть выражена как линейная комбинация предыдущих строк $\mathbf{r}_j$ $(\{j \in \mathrm{K} : j < i\})$. Следовательно, операция проецирования $\prod_{\mathrm{K}}$ однозначно определена, когда ограничена к $\mathcal{L}(\mathbf{B})$, и ее инверсия может быть легко вычислена, используя коэффициенты Грама-Шмидта $\mu_{i,j}$.

2. Введем матрицу $\mathbf{B}' = \prod_{\mathrm{K}}(\mathbf{B})$, которая полного ранга (т.к. все строки линейно независимы), и запустим алгоритм для матриц полного ранга строки, чтобы найти ЭНФ $\mathbf{B}''$ от $\mathbf{B}'$.

3. Применить функцию, обратную операции проецирования, $\prod_{\mathrm{K}}^{-1}$ к ЭНФ $\mathbf{B}''$, чтобы получить матрицу $\mathbf{H}$, которая является ЭНФ матрицы $\mathbf{B}$.

Алгоритм прост, но нужно обратить внимание на операцию проецирования и обратную к ней. Для того, чтобы находить результат проецирования напишем функцию `get_linearly_independent_rows_by_gram_schmidt`, которая будет возвращать матрицу $\mathbf{B}'$, состоящую из линейно независимых строк, а также массив индексов этих строк из исходного массива. К матрице $\mathbf{B}'$ применяется алгоритм нахождения ЭНФ для матриц с полным рангом, разобранный в прошлом разделе. Далее необходимо восстановить удаленные строки. Т.к. они являются линейной комбинацией линейно независимых строк, то мы можем найти коэффициенты, на которые нужно умножить строки из матрицы $\mathbf{B}'$ и после чего сложить их, чтобы получить нужную строку, которую необходимо добавить к $\mathbf{B}'$.

7.4. Пример нахождения ЭНФ

Рассмотрим нахождение ЭНФ на примере небольшой матрицы размера 2×2. Получим случайную матрицу $\mathbf{B} = \begin{bmatrix} \mathbf{b}_1^{\mathbf{T}} \\ \vdots \\ \mathbf{b}_m^{\mathbf{T}} \end{bmatrix} = \begin{bmatrix} 2 & 4 \\ 1 & 4 \end{bmatrix}$. Т.к. мы получаем случайную матрицу, то не можем заранее знать, матрица с полного ранга строки или нет, поэтому будем использовать общий алгоритм. Первый шаг алгоритма требует от нас найти l линейно независимых строк матрицы $\mathbf{B}$, используя алгоритм ортогонализации Грама-Шмидта. Обозначим искомую ортогонализацию строк за $\mathbf{B}^* = \begin{bmatrix} \mathbf{b}_1^{\mathbf{T}*} \\ \vdots \\ \mathbf{b}_m^{\mathbf{T}*} \end{bmatrix}$ и найдем их:

1. $\mathbf{b}_1^{\mathbf{T}*} = \mathbf{b}_1^{\mathbf{T}} + \sum\limits_{j<1} \mu_{1,j} \mathbf{b}_j^{\mathbf{T}*} = \mathbf{b}_1^{\mathbf{T}} = \begin{bmatrix} 2 & 4 \end{bmatrix}$

2. $\mathbf{b}_2^{\mathbf{T}*} = \mathbf{b}_2^{\mathbf{T}} + \sum\limits_{j<2} \mu_{2,j}\mathbf{b}_j^{\mathbf{T}*} = \mathbf{b}_2^{\mathbf{T}} + \frac{\left\langle \mathbf{b}_2^{\mathbf{T}}, \mathbf{b}_1^{\mathbf{T}*} \right\rangle}{\left\langle \mathbf{b}_1^{\mathbf{T}*}, \mathbf{b}_1^{\mathbf{T}*} \right\rangle}\mathbf{b}_1^{\mathbf{T}*} = \begin{bmatrix} -\frac{4}{5} & \frac{2}{5} \end{bmatrix}$

Нулевых строк нет, значит матрица $\mathbf{B}$ полностью состоит из линейно независимых строк, и матрица $\mathbf{B}'$ будет содержать в себе все строки из $\mathbf{B}$. Далее алгоритм требует от нас найти ЭНФ от матрицы $\mathbf{B}'$, используя алгоритм для полного ранга строки.

Рассмотрим алгоритм для полного ранга строки. Алгоритм принимает на вход матрицу $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_n] = \begin{bmatrix} 2 & 4 \\ 1 & 4 \end{bmatrix}$. Требуется найти m линейно независимых строк, используя алгоритм Грама-Шмидта. Используем алгоритм Грама-Шмидта на строки $\mathbf{B}$:

1. $\mathbf{b}_1^* = \mathbf{b}_1 + \sum\limits_{j<1} \mu_{1,j}\mathbf{b}_j^* = \mathbf{b}_1 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$

2. $\mathbf{b}_2^* = \mathbf{b}_2^{\mathbf{T}} + \sum\limits_{j<2} \mu_{2,j}\mathbf{b}_j^* = \mathbf{b}_2 + \frac{\left\langle \mathbf{b}_2, \mathbf{b}_1^* \right\rangle}{\left\langle \mathbf{b}_1^*, \mathbf{b}_1^* \right\rangle}\mathbf{b}_1^* = \begin{bmatrix} -\frac{4}{5} \\ \frac{8}{5} \end{bmatrix}$

Т.к. матрица полного ранга строки, ее ранг меньше либо равен количеству столбцов и равен количеству строк m. Используя алгоритм Грама-Шмидта на столбцы матрицы мы удаляем линейно зависимые столбцы, и т.к. количество столбцов больше либо равно количества строк, то количество столбцов становится равно количеству строк. Получаем матрицу $\mathbf{B}' = \begin{bmatrix} 2 & 4 \\ 1 & 4 \end{bmatrix}$ размера $m \times m$, состоящую из линейно независимых столбцов матрицы $\mathbf{B}$.

Далее необходимо составить матрицу $\mathbf{H}_0$. Для этого необходимо найти определитель решетки $d = \sqrt{(5 \cdot \frac{16}{5})} = 4$ и умножить единичную матрицу размера $m \times m$ на d.

Для $i = 1, \ldots, n$ используем AddColumn для каждого $\mathbf{H}_{i-1}$ и $\mathbf{b}_i$:

1. $\mathbf{H} = \begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix}$, $\mathrm{a} = 4$, $\mathbf{h} = \begin{bmatrix} 0 \end{bmatrix}$, $\mathbf{H}' = \begin{bmatrix} 4 \end{bmatrix}$, $\mathbf{b} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$, $\mathrm{b} = 2$, $\mathbf{b}' = \begin{bmatrix} 1 \end{bmatrix}$

   Используем расширенный НОД алгоритм, находим $\mathrm{g} = 2$, $\mathrm{x} = 0$, $\mathrm{y} = 1$. Составляем матрицу $\mathbf{U} = \begin{bmatrix} 0 & -1 \\ 1 & 2 \end{bmatrix}$, умножаем матрицу, составленную из первого столбца $\mathbf{H}$ и столбца $\mathbf{b}$ на матрицу $\mathbf{U}$: $\begin{bmatrix} \mathrm{a} & \mathrm{b} \\ \mathbf{h} & \mathbf{b}' \end{bmatrix}\mathbf{U} = \begin{bmatrix} \mathrm{g} & 0 \\ \mathbf{h}' & \mathbf{b}'' \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 1 & 2 \end{bmatrix}$, $\mathbf{h}' = \begin{bmatrix} 1 \end{bmatrix}$, $\mathbf{b}'' = \begin{bmatrix} 2 \end{bmatrix}$

   Сокращаем $\mathbf{b}''$ по модулю диагональных элементов из $\mathbf{H}'$, вычисляя и добавляя соответствующий вектор из $\mathcal{L}(\mathbf{H}')$: $\mathbf{b}'' = \begin{bmatrix} 2 \end{bmatrix}$.

   Рекурсивно вызываем AddColumn со входом $\mathbf{H}'$ и $\mathbf{b}''$, получаем матрицу $\mathbf{H}'' = \begin{bmatrix} 2 \end{bmatrix}$:

   - $\mathbf{H} = \begin{bmatrix} 4 \end{bmatrix}$, $\mathrm{a} = 4$, $\mathbf{h} = []$, $\mathbf{H}' = []$, $\mathbf{b} = \begin{bmatrix} 2 \end{bmatrix}$, $\mathrm{b} = 2$, $\mathbf{b}' = []$

     Находим $\mathrm{g} = 2$, $\mathrm{x} = 0$, $\mathrm{y} = 1$. Составляем матрицу $\mathbf{U} = \begin{bmatrix} 0 & -1 \\ 1 & 2 \end{bmatrix}$, умножаем: $\begin{bmatrix} \mathrm{a} & \mathrm{b} \\ \mathbf{h} & \mathbf{b}' \end{bmatrix}\mathbf{U} = \begin{bmatrix} \mathrm{g} & 0 \\ \mathbf{h}' & \mathbf{b}'' \end{bmatrix} = \begin{bmatrix} 2 & 0 \end{bmatrix}$, $\mathbf{h}' = []$, $\mathbf{b}'' = []$

Сокращаем $\mathbf{b}''$ по модулю диагональных элементов из $\mathbf{H}'$, вычисляя и добавляя соответствующий вектор из $\mathcal{L}(\mathbf{H}')$: $\mathbf{b}'' = []$.

Рекурсивно вызываем AddColumn со входом $\mathbf{H}'$ и $\mathbf{b}''$: произойдет выход из рекурсии по условию и вернется пустая матрица $\mathbf{H}''$.

Сокращаем $\mathbf{h}'$ по модулю диагональных элементов из $\mathbf{H}''$, вычисляя и добавляя соответствующий вектор из $\mathcal{L}(\mathbf{H}'')$: $\mathbf{h}' = []$.

Возвращаем $\begin{bmatrix} \mathrm{g} & \mathbf{0^T} \\ \mathbf{h}' & \mathbf{H}'' \end{bmatrix} = \begin{bmatrix} 2 \end{bmatrix}$

Сокращаем $\mathbf{h}'$ по модулю диагональных элементов из $\mathbf{H}''$, вычисляя и добавляя соответствующий вектор из $\mathcal{L}(\mathbf{H}'')$: $\mathbf{h}' = \begin{bmatrix} 2 \end{bmatrix}$.

Возвращаем $\begin{bmatrix} \mathrm{g} & \mathbf{0^T} \\ \mathbf{h}' & \mathbf{H}'' \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 1 & 2 \end{bmatrix}$

2. $\mathbf{H} = \begin{bmatrix} 2 & 0 \\ 1 & 2 \end{bmatrix}$, $\mathrm{a} = 2$, $\mathbf{h} = \begin{bmatrix} 1 \end{bmatrix}$, $\mathbf{H}' = \begin{bmatrix} 2 \end{bmatrix}$, $\mathbf{b} = \begin{bmatrix} 4 \\ 4 \end{bmatrix}$, $\mathrm{b} = 4$, $\mathbf{b}' = \begin{bmatrix} 4 \end{bmatrix}$

Находим $\mathrm{g} = 2, \mathrm{x} = 0, \mathrm{y} = 1$. Составляем матрицу $\mathbf{U} = \begin{bmatrix} 1 & -2 \\ 0 & 1 \end{bmatrix}$, умножаем: $\begin{bmatrix} \mathrm{a} & \mathrm{b} \\ \mathbf{h} & \mathbf{b}' \end{bmatrix} \mathbf{U} = \begin{bmatrix} \mathrm{g} & 0 \\ \mathbf{h}' & \mathbf{b}'' \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 1 & 2 \end{bmatrix}$, $\mathbf{h}' = \begin{bmatrix} 1 \end{bmatrix}$, $\mathbf{b}'' = \begin{bmatrix} 2 \end{bmatrix}$

Сокращаем $\mathbf{b}''$ по модулю диагональных элементов из $\mathbf{H}'$, вычисляя и добавляя соответствующий вектор из $\mathcal{L}(\mathbf{H}')$: $\mathbf{b}'' = \begin{bmatrix} 0 \end{bmatrix}$.

Рекурсивно вызываем AddColumn со входом $\mathbf{H}'$ и $\mathbf{b}''$, получаем матрицу $\mathbf{H}'' = \begin{bmatrix} 2 \end{bmatrix}$:

- $\mathbf{H} = \begin{bmatrix} 2 \end{bmatrix}$, $\mathrm{a} = 2$, $\mathbf{h} = []$, $\mathbf{H}' = []$, $\mathbf{b} = \begin{bmatrix} 0 \end{bmatrix}$, $\mathrm{b} = 0$, $\mathbf{b}' = []$

  Находим $\mathrm{g} = 2$, $\mathrm{x} = 1$, $\mathrm{y} = 0$. Составляем матрицу $\mathbf{U} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, умножаем: $\begin{bmatrix} \mathrm{a} & \mathrm{b} \\ \mathbf{h} & \mathbf{b}' \end{bmatrix} \mathbf{U} = \begin{bmatrix} \mathrm{g} & 0 \\ \mathbf{h}' & \mathbf{b}'' \end{bmatrix} = \begin{bmatrix} 2 & 0 \end{bmatrix}$, $\mathbf{h}' = []$, $\mathbf{b}'' = []$

  Сокращаем $\mathbf{b}''$ по модулю диагональных элементов из $\mathbf{H}'$, вычисляя и добавляя соответствующий вектор из $\mathcal{L}(\mathbf{H}')$: $\mathbf{b}'' = []$.

  Рекурсивно вызываем AddColumn со входом $\mathbf{H}'$ и $\mathbf{b}''$: произойдет выход из рекурсии по условию и вернется пустая матрица $\mathbf{H}''$.

  Сокращаем $\mathbf{h}'$ по модулю диагональных элементов из $\mathbf{H}''$, вычисляя и добавляя соответствующий вектор из $\mathcal{L}(\mathbf{H}'')$: $\mathbf{h}' = []$.

  Возвращаем $\begin{bmatrix} \mathrm{g} & \mathbf{0^T} \\ \mathbf{h}' & \mathbf{H}'' \end{bmatrix} = \begin{bmatrix} 2 \end{bmatrix}$

Сокращаем $\mathbf{h}'$ по модулю диагональных элементов из $\mathbf{H}''$, вычисляя и добавляя соответствующий вектор из $\mathcal{L}(\mathbf{H}'')$: $\mathbf{h}' = \begin{bmatrix} 2 \end{bmatrix}$.

Возвращаем $\begin{bmatrix} \mathrm{g} & \mathbf{0^T} \\ \mathbf{h}' & \mathbf{H}'' \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 1 & 2 \end{bmatrix}$

ЭНФ($\mathbf{B}$) = $\begin{bmatrix} 2 & 0 \\ 1 & 2 \end{bmatrix}$

7.5. Сложность алгоритма

7.6. Обзор программной реализации

В ходе работы была получена реализация с использованием библиотеки Boost.Multiprecision. Реализация находится в пространстве имен `Algorithms::HNF` и состоит из 4 функций:

1. `HNF_full_row_rank(matrix)` $\rightarrow$ `result_HNF` – принимает на вход матрицу с полным рангом строки и возвращает ее ЭНФ. Использует встроенную реализацию больших чисел Boost.Multiprecision.

2. `HNF(matrix)` $\rightarrow$ `result_HNF` – принимает на вход матрицу и возвращает ее ЭНФ. Использует встроенную реализацию больших чисел Boost.Multiprecision.

3. `HNF_full_row_rank_GMP(matrix)` $\rightarrow$ `result_HNF` – принимает на вход матрицу с полным рангом строки и возвращает ее ЭНФ. Использует реализацию больших чисел от GMP.

4. `HNF_GMP(matrix)` $\rightarrow$ `result_HNF` – принимает на вход матрицу и возвращает ее ЭНФ. Использует реализацию больших чисел от GMP.

Программная реализация тестировалась с использованием компилятора G++ версии 6.3.0 в режиме сборки Release на ПК со следующими характеристиками: CPU: Intel(R) Core (TM) i5-9600KF CPU @ 3.70GHz, ОЗУ: DDR4, 16 ГБ (двухканальных режим 8x2), 2666 МГц. Тестирование проводилось на одинаковых данных.

Таблица 1: Время работы ЭНФ

m	n	Время, сек
5	5	0.001
10	10	0.005
17	17	0.05
25	25	0.24
35	35	1.03
50	50	4.27
75	75	23.2
100	100	78.3
100	125	117.1
125	100	104.7

Таблица 2: Время работы ЭНФ с использованием GMP

m	n	Время, сек
5	5	0.002
10	10	0.01
17	17	0.06
25	25	0.22
35	35	0.85
50	50	3.35
75	75	17.9
100	100	59.6
100	125	84
125	100	71.23

7.7. Применение

Будут рассмотрены некоторые проблемы и задачи теории решеток и их решение с помощью ЭНФ.

Нахождение базиса. Дан набор рациональных векторов $\mathbf{B}$, необходимо вычислить базис для $\mathcal{L}(\mathbf{B})$. Проблема решается за полиномиальное время путем вычисления ЭНФ($\mathbf{B}$):

$$\mathbf{B} = \begin{bmatrix} 2 & 2 \\ 1 & 0 \end{bmatrix}, \text{ЭНФ}(\mathbf{B}) = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}.$$

Проблема эквивалентности. Дано два базиса $\mathbf{B}$ и $\mathbf{B}'$. Необходимо узнать, образуют ли они однинаковую решетку $\mathcal{L}(\mathbf{B}) = \mathcal{L}(\mathbf{B}')$. Проблема решается путем вычисления ЭНФ($\mathbf{B}$) и ЭНФ($\mathbf{B}'$) и сравнения их равенства:

$\mathbf{B} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, $\mathbf{B}' = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$, ЭНФ$(\mathbf{B}) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, ЭНФ$(\mathbf{B}') = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ – образуют одинаковую решетку.

$\mathbf{B} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, $\mathbf{B}' = \begin{bmatrix} 2 & 2 \\ 1 & 0 \end{bmatrix}$, ЭНФ$(\mathbf{B}) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, ЭНФ$(\mathbf{B}') = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$ – не образуют одинаковой решетки.

Объединение решеток. Дано два базиса $\mathbf{B}$ и $\mathbf{B}'$. Необходимо найти базис для наименьшей решетки, содержащей обе решетки $\mathcal{L}(\mathbf{B})$ и $\mathcal{L}(\mathbf{B}')$. Такая решетка будет сгенерирована от $[\mathbf{B}|\mathbf{B}']$, и можно легко найти ее базис через ЭНФ:

$$\mathbf{B} = \begin{bmatrix} 2 & 2 \\ 1 & 0 \end{bmatrix}, \mathbf{B}' = \begin{bmatrix} 0 & 1 \\ 2 & 2 \end{bmatrix}, [\mathbf{B}|\mathbf{B}'] = \begin{bmatrix} 2 & 2 & 0 & 1 \\ 1 & 0 & 2 & 2 \end{bmatrix}, \text{ЭНФ}([\mathbf{B}|\mathbf{B}']) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}.$$

Проблема включения. Дано два базиса $\mathbf{B}$ и $\mathbf{B}'$. Необходимо узнать, является ли $\mathcal{L}(\mathbf{B}')$ подрешеткой $\mathcal{L}(\mathbf{B})$, т.е. $\mathcal{L}(\mathbf{B}') \subseteq \mathcal{L}(\mathbf{B})$. Эта проблема сводится к проблемам объединения и эквивалентности: $\mathcal{L}(\mathbf{B}') \subseteq \mathcal{L}(\mathbf{B})$ тогда и только тогда, когда $\mathcal{L}([\mathbf{B}|\mathbf{B}']) = \mathcal{L}(\mathbf{B})$. Для этого необходимо вычислить ЭНФ$([\mathbf{B}|\mathbf{B}'])$ и ЭНФ$(\mathbf{B})$ и сравнения их равенства:

$\mathbf{B} = \begin{bmatrix} 2 & 2 \\ 1 & 0 \end{bmatrix}$, $\mathbf{B}' = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, $[\mathbf{B}|\mathbf{B}'] = \begin{bmatrix} 2 & 2 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$, ЭНФ$([\mathbf{B}|\mathbf{B}']) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$, ЭНФ$(\mathbf{B}) = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$ – $\mathcal{L}(\mathbf{B}')$ не является подрешеткой $\mathcal{L}(\mathbf{B})$.

$\mathbf{B} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, $\mathbf{B}' = \begin{bmatrix} 2 & 2 \\ 1 & 0 \end{bmatrix}$, $[\mathbf{B}|\mathbf{B}'] = \begin{bmatrix} 1 & 0 & 2 & 2 \\ 0 & 1 & 1 & 0 \end{bmatrix}$, ЭНФ$([\mathbf{B}|\mathbf{B}']) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$, ЭНФ$(\mathbf{B}) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ – $\mathcal{L}(\mathbf{B}')$ является подрешеткой $\mathcal{L}(\mathbf{B})$.

Проблема содержания. Дана решетка $\mathbf{B}$ и вектор $\mathbf{v}$, необходимо узнать, принадлежит ли вектор решетке ($\mathbf{v} \subseteq \mathcal{L}(\mathbf{B}')$). Эта проблема сводится к проблеме включения путем проверки $\mathcal{L}([\mathbf{v}]) \subseteq \mathcal{L}(\mathbf{B})$. Если необходимо проверить содержание нескольких векторов $\mathbf{v}_1, \ldots, \mathbf{v}_n$, тогда следует сначала вычислить $\mathbf{H} = \text{ЭНФ}(\mathbf{B})$, и затем проверять, равно ли $\mathbf{H}$ ЭНФ$([\mathbf{H}|\mathbf{v}_i])$ для каждого вектора:

$\mathbf{B} = \begin{bmatrix} 2 & 2 \\ 1 & 0 \end{bmatrix}$, $\mathbf{v} = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$, $[\mathbf{B}|\mathbf{B}'] = \begin{bmatrix} 2 & 2 & 2 \\ 1 & 0 & 0 \end{bmatrix}$, ЭНФ$([\mathbf{B}|\mathbf{v}]) = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$, ЭНФ$(\mathbf{B}) = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$ – вектор $\mathbf{v} \subseteq \mathcal{L}(\mathbf{B})$.

$\mathbf{B} = \begin{bmatrix} 2 & 2 \\ 1 & 0 \end{bmatrix}$, $\mathbf{v} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, $[\mathbf{B}|\mathbf{B}'] = \begin{bmatrix} 2 & 2 & 1 \\ 1 & 0 & 0 \end{bmatrix}$, ЭНФ$([\mathbf{B}|\mathbf{v}]) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$, ЭНФ$(\mathbf{B}) = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$ – вектор $\mathbf{v} \not\subseteq \mathcal{L}(\mathbf{B})$.

Решение систем линейных уравнений.

8. Решение ПБВ (TODO)

Будет разобрано два алгоритма - жадный метод, работающий за полиномиальное время, но дающий приближенное решение, и метод ветвей и границ, работающий за суперполиномиальное время, но точно решающий проблему ближайшего вектора.

8.1. Определение проблемы

Рассмотрим проблему ближайшего вектора (ПБВ): Дан базис решетки $\mathbf{B} \in \mathbb{R}^{d \times n}$ и вектор $\mathbf{t} \in \mathbb{R}^d$, найти точку решетки $\mathbf{Bx}$ ($\mathbf{x} \in \mathbb{Z}^n$) такую, что $||\mathbf{t} - \mathbf{Bx}||$ (расстояние от точки до решетки) минимально. Это задача оптимизации (минимизации) с допустимыми решениями, заданными всеми целочисленными векторами $\mathbf{x} \in \mathbb{Z}^n$, и целевой функцией $f(\mathbf{x}) = ||\mathbf{t} - \mathbf{Bx}||$.

Пусть $\mathbf{B} = [\mathbf{B}', \mathrm{b}]$ и $\mathbf{x} = (\mathbf{x}', x)$, где $\mathbf{B}' \in \mathbb{R}^{d \times (n-1)}$, $\mathbf{b} \in \mathbb{R}^d$, $\mathbf{x}' \in \mathbb{Z}^{n-1}$ и $x \in \mathbb{Z}$. Заметим, что если зафиксировать значение x, то задача ПБВ$(\mathbf{B}, \mathbf{t})$ потребует найти значение $\mathbf{x}' \in \mathbb{Z}^{n-1}$ такое, что

$$||\mathbf{t} - (\mathbf{B}'\mathbf{x}' + \mathbf{b}x)|| = ||(\mathbf{t} - \mathbf{b}x) - \mathbf{B}'\mathbf{x}'||$$

минимально. Это также экземпляр ПБВ $(\mathbf{B}', \mathbf{t}')$ с измененным вектором $\mathbf{t}' = \mathbf{t} - \mathbf{b}x$, и решеткой меньшего размера $\mathcal{L}(\mathbf{B}')$. В частности, пространство решений сейчас состоит из $(n–1)$ целочисленных переменных $\mathbf{x}'$. Это говорит о том, что можно решить ПБВ путем установки значения $\mathbf{x}$ по одной координате за раз. Есть несколько способов превратить этот подход к уменьшению размерности в алгоритм, используя некоторые стандартные методы алгоритмического программирования. Простейшие методы:

1. Жадный метод, который выдает приближенные значения, но работает за полиномиальное время
2. Метод ветвей и границ, который выдает точное решение за суперэкспоненциальное время.

Оба метода основаны на очень простой нижней оценке целевой функции:

$$\min_x f(\mathbf{x}) = dist\left(\mathbf{t}, \mathcal{L}\left(\mathbf{B}\right)\right) \geq dist\left(\mathbf{t}, span\left(\mathbf{B}\right)\right) = ||\mathbf{t} \perp \mathbf{B}||$$

8.2. Жадный метод: алгоритм ближайшей плоскости Бабая

Суть жадного метода состоит в выборе переменных, определяющих пространство решений, по одной, каждый раз выбирая значение, которые выглядит наиболее многообещающим. В нашем случае, выберем значение x, которое дает наименьшее возможное значение для нижней границы $||\mathbf{t}' \perp \mathbf{B}'||$. Напомним, что $\mathbf{B} = [\mathbf{B}', \mathbf{b}]$ и $\mathbf{x} = (\mathbf{x}', x)$, и что для любого фиксированного

значения x, ПБВ $(\mathbf{B}, \mathbf{t})$ сводится к ПБВ $(\mathbf{B}', \mathbf{t}')$, где $\mathbf{t}' = \mathbf{t} - \mathbf{b}x$. Используя $||\mathbf{t}' \perp \mathbf{B}'||$ для нижней границы, мы хотим выбрать значение x такое, что

$$||\mathbf{t}' \perp \mathbf{B}'|| = ||\mathbf{t} - \mathbf{b}x \perp \mathbf{B}'|| = ||(\mathbf{t} \perp \mathbf{B}') - (\mathbf{b} \perp \mathbf{B}')x||$$

как можно меньше. Это очень простая 1-размерная ПБВ проблема (с решеткой $\mathcal{L}(\mathbf{b} \perp \mathbf{B}')$ и целью $\mathbf{t} \perp \mathbf{B}'$), которая может быть сразу решена установкой

$$x = \left\lfloor \frac{\langle \mathbf{t}, \mathbf{b}^* \rangle}{||\mathbf{b}^*||^2} \right\rceil$$

где $\mathbf{b}^* = \mathbf{b} \perp \mathbf{B}'$ компонента вектора $\mathbf{b}$, ортогональная другим базисным векторам. Полный алгоритм приведен ниже:

Input: $[\mathbf{B}, \mathbf{b}], \mathbf{t}$

Output: $\begin{cases} \mathbf{0} & \textbf{Input} = [\,], \mathbf{t} \\ c \cdot \mathbf{b} + Greedy(\mathbf{B}, \mathbf{t} - c \cdot \mathbf{b}) & \textbf{Input} = [\mathbf{B}, \mathbf{b}], \mathbf{t} \end{cases}$

$\mathbf{b}^* \leftarrow \mathbf{b} \perp \mathbf{B}$

$x \leftarrow \langle \mathbf{t}, \mathbf{b}^* \rangle \,/\, \langle \mathbf{b}^*, \mathbf{b}^* \rangle$

$c \leftarrow \lfloor x \rceil$

8.3. Нерекурсивная реализация

Рекурсивный алгоритм, описанный в прошлом пункте, можно преобразовать в нерекурсивный. Для этого необходимо избавиться от рекурсии путем простой замены на цикл:

Input: $\mathbf{B}, \mathbf{t}$

Output: **result**

$\mathbf{GS} \leftarrow GramSchmidt(\mathbf{B})$

$n \leftarrow \mathbf{B}.columns$

$result \leftarrow \mathbf{0}$

for $i \leftarrow 0$ to n **do**

$index \leftarrow n - i - 1$

$\mathbf{b} \leftarrow \mathbf{B}.column(index)$

$\mathbf{b}^* \leftarrow \mathbf{GS}.column(index)$

$x \leftarrow \langle \mathbf{t}, \mathbf{b}^* \rangle \,/\, \langle \mathbf{b}^*, \mathbf{b}^* \rangle$

$c \leftarrow \lfloor x \rceil$

$\mathbf{t} \leftarrow \mathbf{t} - c \cdot \mathbf{b}$

result $\leftarrow$ **result** $+ c \cdot \mathbf{b}$

end for

8.4. Пример жадного метода

Рассмотрим пример на простой решетке $\mathbf{B} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ и целевым вектором $\mathbf{t} = \begin{bmatrix} 1.6 \\ 1.6 \end{bmatrix}$.

Представим входную матрицу в виде $[\mathbf{B}, \mathbf{b}]$. На каждом шаге нам необходимо вычислять вектор $\mathbf{b}^* = \mathbf{b} \perp \mathbf{B}$. Эти вектора можно заранее вычислить через алгоритм Грама-Шмидта. В нашем случае вектора уже перпендикулярны друг другу. Смысл алгоритма заключается в установлении одной координаты за раз, для этого мы берем крайний вектор базиса, находим коэффициент, на который его надо умножить, и скадываем с результатом рекурсии текущего алгоритма со входом уменьшенной матрицы и отредактированной целью. Таким образом мы найдем коэффициенты для каждого вектора базиса, и ответ будет суммой умножения коэффициентов на соответствующий вектор базиса:

1. $[\mathbf{B}, \mathbf{b}] = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \mathbf{t} = \begin{bmatrix} 1.6 \\ 1.6 \end{bmatrix}, \mathbf{b}^* = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, x = 1.6, c = 2, c \cdot \mathbf{b} = \begin{bmatrix} 0 \\ 2 \end{bmatrix}.$

   Рекурсивно вызываем метод, на вход отправляем $[\mathbf{B}, \mathbf{b}] = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \mathbf{t} = \mathbf{t} - c \cdot \mathbf{b} = \begin{bmatrix} 0 \\ -0.4 \end{bmatrix}$

2. $[\mathbf{B}, \mathbf{b}] = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \mathbf{t} = \begin{bmatrix} 0 \\ -0.4 \end{bmatrix}, \mathbf{b}^* = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, x = 1.6, c = 2, c \cdot \mathbf{b} = \begin{bmatrix} 2 \\ 0 \end{bmatrix}.$

   Рекурсивно вызываем метод, на вход отправляем $[\mathbf{B}, \mathbf{b}] = []$, $\mathbf{t} = \mathbf{t} - c \cdot \mathbf{b} = \begin{bmatrix} -2 \\ -0.4 \end{bmatrix}$

3. Т.к. $[\mathbf{B}, \mathbf{b}] = []$, то возвращаем пустой вектор.

   В итоге сумма векторов будет равна $\begin{bmatrix} 2 \\ 2 \end{bmatrix}$ – искомый вектор.

8.5. Метод ветвей и границ

Алгоритм похож на жадный метод, но вместо установки x_n на наиболее подходящее значение (то есть на то, для которого нижняя граница расстояния $\mathbf{t}' \perp \mathbf{B}'$ минимальна), мы ограничиваем множество всех возможных значений для x , и затем мы переходим на каждую из них для решения каждой соответствующей подзадачи независимо. В заключении, мы выбираем наилучшее возможное решение среди возвращенных всеми ветками.

Чтобы ограничить значения, которые может принимать x, нам также нужна верхняя граница расстояния от цели до решетки. Ее можно получить несколькими способами. Например, можно просто использовать $||\mathbf{t}||$ (расстояние от цели до начала координат) в качестве верхней границы. Но лучше использовать жадный алгоритм, чтобы найти приближенное решение $\mathbf{v} = \text{Greedy}(\mathbf{B}, \mathbf{t})$, и использовать $||\mathbf{t} - \mathbf{v}||$ в качестве верхней границы. Как только верхняя граница u установлена, можно ограничить переменную x такими значениями, что $(\mathbf{t} - x\mathbf{b}) \perp \mathbf{B}'|| \leq u$.

Окончательный алгоритм похож на жадный метод и описан ниже:

Input: $[\mathbf{B}, \mathbf{b}], \mathbf{t}$

Output: $\begin{cases} \mathbf{0} & \textbf{Input} = [\,], \mathbf{t} \\ closest(V, \mathbf{t}) & \textbf{Input} = [\mathbf{B}, \mathbf{b}], \mathbf{t} \end{cases}$

$\mathbf{b}^* \leftarrow \mathbf{b} \perp \mathbf{B}$

$\mathbf{v} \leftarrow Greedy([\mathbf{B}, \mathbf{b}], \mathbf{t})$

$X \leftarrow \{x : ||(\mathbf{t} - x\mathbf{b}) \perp \mathbf{B}|| \leq ||\mathbf{t} - \mathbf{v}||\}$

$V \leftarrow \{x \cdot \mathbf{b} + \text{Branch\&Bound}(\mathbf{B}, \mathbf{t} - x \cdot \mathbf{b}) : x \in X\}$

где closest$(V, \mathbf{t})$ выбирает вектор в $V \subset \mathcal{L}(\mathbf{B})$ ближайший к цели $\mathbf{t}$.

Как и для жадного алгоритма, производительность (в данном случае время выполнения) метода Ветвей и Границ может быть очень низкой, если мы сперва не сократим базис входной решетки (например используя LLL-алгоритм).

Сложность алгоритма заключается в нахождении множества X. Его можно найти, используя выражение, выведенное в прошлом алгоритме: $x = \frac{\langle \mathbf{t}, \mathbf{b}^* \rangle}{||\mathbf{b}^*||^2}$. С помощью него мы найдем x, который точно удовлетворяет множеству, а затем будем увеличивать/уменьшать до тех пор, пока выполняется условие $||(\mathbf{t} - x\mathbf{b}) \perp \mathbf{B}|| \leq ||\mathbf{t} - \mathbf{v}||$.

8.6. Пример метода ветвей и границ

Рассмотрим пример на простой решетке $\mathbf{B} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ и целевым вектором $\mathbf{t} = \begin{bmatrix} 1.6 \\ 1.6 \end{bmatrix}$.

Представим входную матрицу в виде $[\mathbf{B}, \mathbf{b}]$. На каждом шаге нам необходимо вычислять вектор $\mathbf{b}^* = \mathbf{b} \perp \mathbf{B}$. Заранее вычислим их с помощью алгоритма Грама-Шмидта. В нашем случае вектора уже перпендикулярны друг другу. Смысл алгоритма также заключается в установлении одной координаты за раз, но вместо самого перспективного варианта мы будем строить множество X, подходящее под условие $|(\mathbf{t} - x\mathbf{b}) \perp \mathbf{B}| \leq |\mathbf{t} - \mathbf{v}|$. Вектор $\mathbf{v}$ найдем с помощью жадного метода. Далее также, как и в жадном методе ищем необходимую сумму векторов, получим множество V, из которого необходимо будет выбрать ближайший к цели $\mathbf{t}$.

$[\mathbf{B}, \mathbf{b}] = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \mathbf{t} = \begin{bmatrix} 1.6 \\ 1.6 \end{bmatrix}, \mathbf{b}^* = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \mathbf{v} = \begin{bmatrix} 2 \\ 0 \end{bmatrix}, X = \{2, 3, 1, 0\}.$

Рекурсивно вызываем метод для каждого $x \in X$, на вход отправляем $[\mathbf{B}, \mathbf{b}] = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, $\mathbf{t} = \mathbf{t} - x \cdot \mathbf{b}$.

Получаем множество $V = \left\{ \begin{bmatrix} 2 \\ 2 \end{bmatrix}, \begin{bmatrix} 2 \\ 3 \end{bmatrix}, \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \begin{bmatrix} 2 \\ 0 \end{bmatrix} \right\}$.

Ближайший вектор будет равен $\begin{bmatrix} 2 \\ 2 \end{bmatrix}$ – искомый вектор.

8.7. Параллельная реализация метода ветвей и границ

Для получения параллельной реализации будем использовать задачи (task) из библиотеки OpenMP. После получения множества X нахождение множества подходящих векторов V можно получить параллельным образом, для каждого значения $x \in X$ создавая свою задачу. Задачи помещаются в специальный пул задач, после чего свободные потоки берут задачи и выполняют работу параллельно. В качестве синхронизации используется директива `#pragma omp taskwait`, она указывается перед вызовом `closest(V, t)`

8.8. Сложность алгоритмов

8.9. Обзор программной реализации

Реализация находится в пространстве имен `Algorithms::CVP` и состоит из 4 функций:

1. `greedy_recursive(matrix, vector)` $\rightarrow$ `vector` – рекурсивный Greedy алгоритм, принимает на вход базис решетки и целевой вектор, возвращает вектор решетки, примерно ближайший к целевому.

2. `greedy(matrix, vector)` $\rightarrow$ `vector` – последовательный Greedy алгоритм, принимает на вход базис решетки и целевой вектор, возвращает вектор решетки, примерно ближайший к целевому.

3. `branch_and_bound(matrix, vector)` $\rightarrow$ `vector` – рекурсивный Branch and Bound алгоритм, принимает на вход базис решетки и целевой вектор, возвращает вектор решетки, ближайший к целевому.

4. `greedy_recursive(matrix, vector)` $\rightarrow$ `vector` – параллельный рекурсивный Branch and Bound алгоритм, принимает на вход базис решетки и целевой вектор, возвращает вектор решетки, ближайший к целевому.

Программная реализация тестировалась с использованием компилятора G++ версии 6.3.0 в режиме сборки Release на ПК со следующими характеристиками: CPU: Intel(R) Core (TM) i5-9600KF CPU @ 3.70GHz, ОЗУ: DDR4, 16 ГБ (двухканальных режим 8x2), 2666 МГц. Тестирование проводилось на одинаковых данных. Производительность данных алгоритмов (особенно метода Ветвей и Границ) может быть невысокой из-за несокращенных базисов.

Таблица 3: Время работы рекурсивного Greedy

m	n	Время, сек
12	12	0.002
20	20	0.003
50	50	0.004
100	100	0.006
150	150	0.1
250	250	0.027
500	500	0.2
1000	1000	0.9
1500	1500	2.9
2500	2500	13.4
3500	3500	29.2
5000	5000	78.8

Таблица 4: Время работы последовательного Greedy

m	n	Время, сек
12	12	0.002
20	20	0.003
50	50	0.004
100	100	0.007
150	150	0.01
250	250	0.027
500	500	0.2
1000	1000	0.9
1500	1500	2.9
2500	2500	13.2
3500	3500	29
5000	5000	78.6

Таблица 5: Время работы Branch and Bound

m	n	Время, сек
3	3	0.002
7	7	0.061
11	11	9.4

Таблица 6: Время работы параллельного Branch and Bound

m	n	Время, сек
3	3	0.001
7	7	0.01
11	11	1.6
12	12	16.1
13	13	91.2

8.10. Применение

9. Обзор программной реализации (TODO)

10. Заключение (TODO)

В ходе выполнения выпускной квалификационной работы бакалавра была написана библиотека, в которой реализованы алгоритмы для нахождения ЭНФ и решения ПБВ на языке C++. Полученную библиотеку можно подключать и использовать в других проектах.

Был создан Github репозиторий, который содержит в себе все исходные файлы программы, подключенные библиотеки и .tex файлы выпускной квалификационной работы. Программная реализация использует CMake для автоматической сборки исходного кода и .pdf документа.

Был получен опыт работы с языком C++, библиотеками для работы с линейной алгеброй и числами высокой точности, системой контроля версий Git, системой сборки CMake и написанием отчетов в формате .tex.

Список литературы

1. Daniele Micciancio. Point Lattices. [Электронный ресурс]. — URL: https://cseweb.ucsd.edu/classes/sp14/cse206A-a/lec1.pdf (Дата обращения: 16.05.2022).
2. Daniele Micciancio. Basic Algorithms. [Электронный ресурс]. — URL: https://cseweb.ucsd.edu/classes/sp14/cse206A-a/lec4.pdf (Дата обращения: 16.05.2022).
3. Документация библиотеки Eigen. [Электронный ресурс]. — URL: https://eigen.tuxfamily.org/dox/index.html (Дата обращения: 16.05.2022).
4. Документация библиотеки Boost.Multiprecision. [Электронный ресурс]. — URL: https://www.boost.org/doc/libs/1_79_0/libs/multiprecision/doc/html/index.html (Дата обращения: 16.05.2022).