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.

Данная работа посвящена изучению задач теории решеток и методов их решения. В работе изложены основные понятия, связанные с решетками, и разбор алгоритмов для нахождения Эрмитовой нормальной формы и решения ближайшего вектора решетки.

Целью работы является программная реализация алгоритмов для решения задач.

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

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 неравен нулю.

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. Будем обозначать матрицы через вектора-столбцы, то есть $\mathbf{A} = [\mathbf{a}_1, \cdots, \mathbf{a}_n]$. Вектора-строки будем обознать $\mathbf{a}^{\mathbf{T}}$.

Линейная зависимость и независимость – пусть имеется несколько векторов одной размерности $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k$ и столько же чисел $\alpha_1, \alpha_2, \cdots, \alpha_k$. Вектор $\mathbf{y} = \alpha_1 x_1 + \alpha_2 x_2 + \ldots + \alpha_k x_k$ называется линейной комбинацией векторов $\mathbf{x}_k$. Если существуют такие числа $\alpha_i, i = 1, \ldots, k$, не все равные нулю, такие, что $\mathbf{y} = 0$, то такой набор векторов называется линейно зависимым. В противном случае векторы называются линейно независимыми.

Ранг матрицы – максимальное число линейно независимых векторов. Матрица называется матрицей полного ранга строки, когда все строки матрицы линейно независимы. Матрица называется матрицей полного ранга столбца, когда все столбцы матрицы линейно независимы.

Решетка - пусть $\mathbf{B} = [\mathbf{b}_1, ..., \mathbf{b}_n] \in \mathbb{R}^{d \times n}$ - 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 $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_n] \in \mathbb{R}^{m \times n}$ 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 яызку отдается из-за его скорости, статической типизации и большому количеству написанных библиотек. Сборка проекта осуществляется с помощью системы сборки CMake, при сборке она автоматически собирает документ выпускной квалификационной работы, написанный в формате LaTeX. 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 система контроля версий Git и сервис Github, все исходные файлы проекта доступны в онлайн репозитории. Для Boost.Multiprecision используются модули Git.

6.1. Обзор библиотеки Eigen

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

Все необходимые классы находятся в заголовочном файле `Eigen/Dense` и подключается командой `#include <Eigen/Dense>`. Все используемые классы находятся в пространстве имен Eigen.

Используемые классы:

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

`Vector` и `RowVector` – псевдонимы класса матриц, в которых количество строк/столбцов равно единице. Используются псевдонимы для различных типов и размеров векторов, например `Vector2f` – вектор, состоящий из элементов `float` размера `3`.

Матрицы и вектора можно складывать и вычитать между собой, умножать и делить между собой и на скаляр.

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())` – перебор матрицы по столбцам.

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

6.2. Обзор библиотеки Boost.Multiprecision

Multiprecision – часть библиотеки Boost. Подключается в режиме Standalone и не требует подключения основной библиотеки. Все классы находятся в пространстве имен `boost::multiprecision`. Для подключения используется директива `#include <boost/multiprecision/cpp_тип.hpp>`.

Библиотека предоставляет классы для работы с целыми, рациональными числами и числами с плавающей запятой, которые имеют большую точность, чем встроенные в C++ типы данных. Точность и размер чисел ограничен 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.

Используемые методы:

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

`numerator(rational)` – числитель рационального числа.

`denominator(rational)` – знаменатель рационального числа.

7. Нахождение ЭНФ (TODO)

Будет разобрано два алгоритма - общий и алгоритм для матриц полного ранга строки, который используется в общем алгоритме.

7.1. Алгоритм для матриц полного ранга строки

Дана матрица $\mathbf{B} \in \mathbb{Z}^{m \times n}$. Предположим, что у нас есть процедура AddColumn, которая работает за полиномиальное время и принимает на вход квадратную невырожденную ЭНФ матрицы $\mathbf{H} \in \mathbb{Z}^{m \times m}$ и вектор $\mathbf{b}$, а возвращает ЭНФ матрицы $[\mathbf{H}|\mathbf{b}]$. ЭНФ от $\mathbf{B}$ может быть вычислена 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 $\mathbf{H}_i$ это результат применения AddColumn ко входу $\mathbf{H}_{i-1}$ 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, а значит матрица, состоящая из этих столбцов, будет размера $m \times m$. Для нахождения этих строк можно использовать алгоритм ортогонализации Грама-Шмитда: если $\mathbf{b}_i^* = 0$, 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. Для вычисления det напишем функцию det_by_gram_schmidt, которая принимает на вход матрицу и вычисляет det по формуле $d = \prod_i \|\mathbf{b}_i^*\|$ - сумма произведений длин всех элементов, 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 функцию 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 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 тут ничего не надо делать, и мы можем сразу вернуть $\mathbf{H}$. 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 $g = НОД\,(a, b)$ 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 $\begin{bmatrix} g & \mathbf{0}^T \\ \mathbf{h}' & \mathbf{H}'' \end{bmatrix}$

7.2. Общий алгоритм для любых матриц

1. Запустить процесс ортогонализации Грамма-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 $\mathbf{r}^*_{\mathrm{k}_i} \neq 0$. 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 другая строка может быть выражена как линейная комбинация предыдущих строк $\mathbf{r}_j\,(\{j \in \mathrm{K} : j < i\})$. Следовательно, $\prod_{\mathrm{K}}$ однозначна, когда ограничена к $\mathcal{L}(\mathbf{B})$, и ее инверсия может быть легко вычислена, используя коэффициенты Грама-Шмидта $\mu_{i,j}$.

2. Определить новую матрицу $\mathbf{B}' = \prod_{\mathrm{K}}(\mathbf{B})$, которая полного ранга, и запустить алгоритм, данный в предыдущем пункте, чтобы найти ЭНФ $\mathbf{B}''$ от $\mathbf{B}'$.

3. Применить функцию обратную операции проецирования, $\prod_{\mathrm{K}}^{-1}$, к ЭНФ, определенной в предыдущем шаге $(\mathbf{B}'')$, к данной матрице $\mathbf{H}$. Легко заметить, что $\mathcal{L}\,(\mathbf{H})\,\mathcal{L}\,(\mathbf{B})$ и $\mathbf{H}$ входят в ЭНФ. Следовательно, $\mathbf{H}$ является ЭНФ $\mathbf{B}$.

Алгоритм прост, но вызывает вопрос операция проецирования и обратная 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, данный 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 $\mathbf{B}'$.

7.3. Пример нахождения ЭНФ

7.4. Сложность алгоритма

7.5. Обзор программной реализации

7.6. Применение

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

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

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

Greedy$([], \mathbf{t}) = 0$

Greedy$([\mathbf{B}, \mathbf{b}], \mathbf{t}) = c \cdot \mathbf{b} +$ Greedy$(\mathbf{B}, \mathbf{t} - c \cdot \mathbf{b})$

где $\mathbf{b}^* = \mathbf{b} \perp \mathbf{B}$

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

$c = \lfloor x \rceil$.

8.3. Пример жадного метода

8.4. Метод ветвей и границ

Структура похожа на жадный алгоритм, но вместо жадной 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 $\mathbf{v} =$ 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:

Branch&Bound$([], \mathbf{t}) = 0$

Branch&Bound$([\mathbf{B}, \mathbf{b}], \mathbf{t}) =$ closest$(V, \mathbf{t})$

где $\mathbf{b}^* = \mathbf{b} \perp \mathbf{B}$

$\mathbf{v} =$ Greedy$(\mathbf{B}, \mathbf{t})$

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

$V = x \cdot \mathbf{b} +$ 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 метода Ветвей и Границ может быть произвольно плохой, если мы сперва не сократим базисы.

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

8.5. Пример метода ветвей и границ

8.6. Сложность алгоритмов

8.7. Обзор программной реализации

8.8. 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).

Приложения (TODO)