|  |  |
| --- | --- |
| **CSE 206A: Lattice Algorithms and Applications** | Spring 2014 |
| Базовые алгоритмы  Instructor: *Daniele Micciancio* | UCSD CSE |

Мы уже видели алгоритм, который вычисляет ортогонализацию Грама-Шмидта базиса решетки. Мы продолжим изучение алгоритмов на решетках, исследуя два других важных алгоритма:

* Вычисление Эрмитовой нормальной формы, что можно использовать для решения многих проблем теории множеств, например проверка включения в решетку, включение или равенство между двумя решетками, вычисление объединения и пересечения решеток и т.д.
* Различные типы алгоритмов перечисления, основанных на стандартных алгоритмических методах, таких как жадный алгоритм или метод ветвей и границ. Они используются для решения более геометрических проблем, таких как нахождение кратчайших векторов решетки или векторов решетки, близких к заданной цели.

# Эрмитова нормальная форма

Мы уже описали метод для вычисления Эрмитовой нормальной формы невырожденной квадратной матрицы. Но этот метод имеет два недостатка:

* Метод не может быть применен к решеткам, которые неполного ранга, или к линейно зависимому множеству векторов решетки,
* Он может не работать за полиномиальное время из-за резкого увеличения размера чисел

Тут мы приводим эффективный алгоритм вычисления Эрмитовой нормальной формы для произвольных целочисленных матриц, и потом используем его для решения различных проблем решеток. Сперва определение Эрмитовой нормальной Формы (ЭНФ) для квадратной матрицы расширим для произвольной матрицы.

**Определение 1** *Невырожденная матрица* **B** = [**b**1*, ...,* **b***n*] ∈ *m*×*n* является Эрмитовой нормальной формой (ЭНФ), если

* Существует 1 ≤ *i*1 *< ... < ih* ≤ *m такое, что bi,j* ≠ 0 ⇒ (*j < h*) ∧ (*i* ≥ *ij*) *(строго убывающая высота столбца).*
* *Для всех , т.е. все элементы в строках ij приведены по модулю .*

Индекс h это количество ненулевых столбцов в матрице, а индекс *ij* номер строки верхнего ненулевого элемента столбца j. Поскольку эти индексы строго возрастают, каждый столбец содержит строку, которой нет ни в одном из предыдущих столбцов. Таким образом, ненулевые столбцы матрицы в ЭНФ (также, как и строки с индексами *ij*) гарантированно линейно независимы. Мы также знаем (но не покажем тут) что ЭНФ уникальна, т.е., если две матрицы **B** и **B**’ являются ЭНФ, и они производят одинаковую решетку ((**B**) = (**B**’)), тогда **B** = **B**’ (кроме числа нулевых столбцов в конце). Мы говорим, что **H** является ЭНФ от **B,** если (**H**) = (**B**), **H** в ЭНФ, и **H** не содержит нулевых столбцов.

Несложно сделать алгоритм, который вычисляет ЭНФ матрицы, выполняя только полиномиальное количество операций. Но, наивное решение этой проблемы может обернуться суперполиномиальным временем исполнения, потому что число промежуточных вычислений легко может стать очень большим. Чтобы достичь полиномиального времени работы требуется осторожность.

Заметим, что задача вычисление ЭНФ рациональной матрицы **B** ∈ легко сводится к задаче вычисления ЭНФ целочисленной матрицы следующим образом:

1. Пусть D – наименьшее общее кратное всех знаменателей, входящих в **B**,
2. Вычислить ЭНФ **H** целочисленной матрицы
3. Вернуть *D*−1 · **H**

Легко подтвердить, что **H** принадлежит ЭНФ тогда и только тогда, когда принадлежит . Кроме того, если **H** и порождают одну и ту же решетку, то также порождают одну и ту же решетку. Следовательно, *D*−1 · **H** является ЭНФ **B**. Такое сокращение имеет полиномиальное время потому, что log2 *D* не больше размера входной матрицы. В оставшейся части этого раздела мы покажем, как вычислить ЭНФ целочисленной матрицы.

Итак, мы можем предположить, что входная матрица имеет целочисленные элементы. Сперва мы дадим алгоритм для вычисления ЭНФ матриц с полным рангом строки, а затем покажем, как адаптировать его к произвольным матрицам.

## Матрицы с полным рангом строки

Мы дадим алгоритм, который ищет ЭНФ любой матрицы , у которой полный ранг строки. Мы знаем, что в этом случае ЭНФ **B** это квадратная невырожденная матрица **H**. Идея состоит в том, чтобы сперва найти ЭНФ базис **H** подрешетки **,** и потом обновить **H,** включая столбцы из **B** один за другим. Предположим, что у нас есть процедура AddColumn с полиномиальным времен, которая на вход принимает квадратную невырожденную ЭНФ матрицы и вектор **b**, возвращает ЭНФ матрицы [**H|b**]. ЭНФ от **B** может быть вычислена следующим образом:

1. Применить алгоритм Грама-Шмидта к столбцам **B**, чтобы найти *m* линейно независимых столбцов. Пусть **B’** – матрица размера , заданная этими столбцами.
2. Вычислить *d*=det(**B’**), используя алгоритм Грама-Шмидта или любую другую процедуру с полиномиальным временем. Пусть будет диагональной матрицей с *d* на диагонали.
3. Для *i*=1,…,*n* пусть результат применения AddColumn ко входу .
4. Вернуть .

Корректность алгоритма основана на инварианте, что для всех *i* Инвариант определен для *i*=0. Кроме того, он сохраняется по определению на каждой итерации AddColumn. Таким образом, при завершении алгоритм возвращает ЭНФ . Итого, поскольку , мы имеем и алгоритм возвращает ЭНФ **B.**

Заметим, что в течение всего процесса все входы были зависимы от *d*. В частности, все числа полиномиальны на входе **B**. Итак, если AddColumn имеет полиномиальное время, то весь ЭНФ алгоритм имеет полиномиальное время. Остается дать алгоритм AddColumn процедуры. На входе квадратная невырожденная ЭНФ матрицы и вектор , AddColumn работает следующим образом. Если m = 0, то тут ничего не надо делать, и мы можем сразу выйти с возвращением **H**. В другом случае, пусть и и дальше:

1. Вычислить и целые x,y такие, что , используя расширенный НОД алгоритм.
2. Применить унимодулярное преобразование к первому столбцу из **H** и **b** чтобы получить
3. Добавить соответствующий вектор из к **b**’’, чтобы сократить его элементы по модулю диагональных элементов из **H’**.
4. Рекурсивно вызвать AddColumn на вход **H**’ и **b**’’ чтобы получить матрицу **H’’.**
5. Добавить соответствующий вектор из к **h**’ чтобы сократить его элементы по модулю диагональных элементов из **H’’**.
6. Вернуть

## Общий случай

Мы хотим привести общий случай к полноразмерному случаю. Мы начнем, используя операцию проецирования П, чтобы выбрать **B’**, подматрицу **B**, состоящую только из линейно независимых строк из **B**. Потом мы используем алгоритм для полноразмерного случая. Наконец, мы используем обратную к проецированию операцию, чтобы получить финальный результат.

1. Запустить процесс ортогонализации Грамма-Шмидта к строкам из **B,** и пусть – это множество индексов, такое, что . Определим операцию проецирования . Заметим, что строки линейно независимы и любая другая строка может быть выражена как линейная комбинация предыдущих строк Следовательно, однозначна, когда ограничена к , и ее инверсия может быть легко вычислена, используя коэффициенты Грама-Шмидта .
2. Определить новую матрицу **B’**=, которая полноранговая, и запустить алгоритм, данный в предыдущем пункте, чтобы найти ЭНФ **B**’’ от **B’.**
3. Применить функцию обратную операции проецирования, , к ЭНФ, определенной в предыдущем шаге(**B’’**), к данной матрице **H**. Легко заметить, что входят в ЭНФ. Следовательно, **H** является ЭНФ **B**.

Это дает нам алгоритм для нахождения ЭНФ, который работает за полиномиальное время в *n, m* и *log*(*d*). Чтобы завершить доказательство, нам нужно показать, что log(*d*) полиномиально при матрице исходного размера. Т.к. *d* это определитель подматрицы *B*, то достаточно показать, что для любой квадратной матрицы, размер(det(A)) полиномиален размеру(*A*). Используя неравенство Адамара , мы можем написать

Если все имеют битовый размер не более , мы получим , доказывая, что размер определителя полиномиален размеру матрицы.

## Применение

Мы используем ЭНФ алгоритм и двойственные решетки, чтобы эффективно решать различные основные задачи на решетках.

**Основная проблема** Дано множество рациональных векторов **B**, мы хотим вычислить базис для решетки .

Эта проблема мгновенно решается (за полиномиальное время) путем вычисления ЭНФ(**B**).

**Проблема эквивалентности** Даны для базиса **B** и **B’**, мы хотим определить, образуют ли они одинаковую решетку

Эта проблема может решена за полиномиальное время, путем вычисления **H** = ЭНФ(**B**) и **H**’ = ЭНФ(**B**’), и проверки равенства **H** и **H**’.

**Объединение решеток** Даны два базиса **B** и **B**’, мы хотим определить базис для наименьшей решетки, включающей и , и .

Легко увидеть, что эта решетка сгенерирована от [**B** | **B**’], значит базис для решетки может быть легко вычислен, как ЭНФ([**B** | **B**’]).

**Проблема содержания** Даны два базиса **B** и **B**’, мы хотим определить верно ли, что подрешетка , т.е., .

Эта проблема легко сводится к проблеме объединения и равенства: тогда и только тогда, когда . Итого, чтобы проверить включение нам нужно только вычислить ЭНФ([**B | B**’]) и ЭНФ(**B**) и проверить эквивалентность базисов ЭНФ.

**Проблема включения** Дана решетка **B** и вектор **v**, мы хотим определить верно ли, что .

Это немедленно сводится к проблеме содержания путем проверки верно ли, что . Если нам нужно проверить включение для множества векторов тогда удобно сперва вычислить **H** = ЭНФ(**B**), и потом для каждого *i* проверять **H** = ЭНФ([**H |** ]). Заметим, что ЭНФ([**H |** ]) может быть легко вычислена намного быстрее, чем ЭНФ для матричного вычисления, потому что большинство матриц уже являются Эрмитовой Нормальной Формой.

**Решение линейных систем** Пусть **Ax = b** будет системой линейных уравнений. Мы хотим найти решение **x**, или сказать, что решений нет. (Мы знаем, что это может быть выполнено за арифметических операций. Проблема в том, чтобы показать, что числа остаются малыми.)

Мы можем легко сказать, имеет ли система решения путем запуска Грама-Шмидта на [**A | b**], и проверки, что последний столбец равен **b**\* = **0.** Также, мы можем остановить наше внимание на решениях, которые используют переменные, для которых . Итак, предположим, что столбцы из **A** линейно независимы, и **Ax = b** для произвольного **x**. Мы также можем избавиться от избыточных уравнений путем запуска Грама-Шмидта на строки [**A** | **b**] и убрав уравнения, для которых соответствующий ортогонализованный вектор равен **0.**

Итак, мы свели проблему решения произвольной системы линейных уравнений к решению системы **Ax = b**, где **A** это невырожденная квадратная матрица. Потом, система может быть легко решена путем вычисления ЭНФ (или, что эквивалентно, ортогонализацией Грама-Шмидта) , чтобы получить , где **C** это треугольная матрица, и затем решить эквивалентную треугольную систему **Cx = d** обратной подстановкой. Легко увидеть, что в этом случае все задействованные числа гарантированно не будут слишком большими, и система может быть решена за полиномиальное время. (Примечание: это далеко не самый быстрый способ решать систему линейных уравнений. Известны намного более быстрые методы.)

В частном случае, это показывает, что матрица, обратная к невырожденной квадратной матрице **A** может быть вычислена за полиномиальное время путем решение уравнений . Обратная матрица дана как [].

**Двойственная решетка** Дан решеточный базис **B,** вычислить двойственный базис **D**, т.е., базис **D** такой, что и span(**B**) = span(**D**).

Эта проблема легко решается путем вычисления D как D=B〖(B^T B)〗^(-1). Заметим, что это вычисление включает в себя три матричных умножения и одно получение обратной матрицы.

**Пересечение двух решеток** Даны два базиса **B** и **B**’, мы хотим определить базис для пересечения .

Легко показать, что если и двойственные решетки от и , тогда двойственная от это **.** Итак, базис для пересечения найден путем вычисления **D**, **D**’ и **H =** ЭНФ([**D | D**’]), и потом вычисление двойственной от **H.**

**Цикличная решетка** Пусть r(**x**) циклическое вращение вектора **x**, т.е., . Дано множество векторов **B**, найти циклическую решетку, полученную от **B**, т.е. наименьшую циклическую решетку, содержащую **B**.

Эта проблема легко решается, учитывая вектора для всех и , и вычислением решетки, полученной от этих векторов.

Похожая проблема состоит в решение, является ли данная решетка циклической, т.е., верно ли, что (в таком случае, . Задача решение циклической решетки легко решается путем вычисления ЭНФ(**B**) и ЭНФ(*r*(**B**)) и проверкой равенства этих двух матриц.

# Алгоритмы перечисления

Рассмотрим Проблему Ближайшего Вектора(ПБВ): Дан решеточный базис и целевой вектор , найти точку решетки **Bx** такую, что минимизировано. Это задача оптимизации (минимизации) с допустимыми решениями, заданными всеми целочисленными векторами , и целевой функцией f(**x**) = .

Пусть **B = [B**’, **b**] и **x** = (**x**’, *x*), где , **,** и .

Заметим, что если вы зафиксируете значение *x*, то задача ПБВ (**B, t**) потребует найти значение такое, что минимизировано. Это также экземпляр ПБВ (**B’, t’**) с модифицированным , и решеткой меньшего размера . В частности, пространство решений сейчас состоит из (n – 1) целочисленных переменных **x**’. Это говорит о том, что можно решить ПБВ путем установки значения **x** по одной координате за раз.

Есть несколько способов превратить этот подход к уменьшению размерности в алгоритм, используя некоторые стандартные методы алгоритмического программирования. Простейшие техники такие:

* жадный метод, который выдает приближенные значения, но работает за полиномиальное время, и
* метод ветвей и границ, который выдает точные решение за суперэкспоненциальное время.

Оба метода основаны на очень простой нижней оценке целевой функции:

Расстояние от цели **t** к решетке не менее , т.е., расстояние от цели до линейной оболочки решетки. Эта тривиальная нижняя граница кажется неинтересной, но, как мы увидим, она дает очень полезную информацию, когда используется в сочетании с подходом уменьшения размерности, описанным ранее.

**t**

**Bx**

## Жадный метод: Алгоритм ближайшей плоскости Бабая

Жадный метод состоит из выбора переменных по одной, определяющих пространство решений, каждый раз выбирая значение, которые выглядит более многообещающим. В нашем случае, выберем значение *x*, которое дает наименьшее возможное значение для нижней границы . Напомним, что и , и что для любого фиксированного значения *x*, ПБВ (**B, t**) сводится к ПБВ (**B’, t’**), где . Используя для нижней границы, мы хотим выбрать значение *x* такое, что

Как можно меньшее. Это очень простая 1-размерная ПБВ проблема (с решеткой и целью ), которая может быть сразу решена установкой

где компонента вектора **b**, ортогональная другим базисным векторам. Вот и все! Полный алгоритм приведен ниже:

Greedy([],**t**) = **0**

Greedy([**B***,***b**],**t**) = *c* · **b** + Greedy(**B**, **t** − *c* · **b**)

где **b**∗ = **b**⊥**B**

*x* =

*c* =

Легко проверить, что алгоритм всегда возвращает решеточный вектор, и что это выполняется за полиномиальное число арифметических операций. Размер всех использованных чисел также легко оценить, используя тот факт, что все ортогонализованные по Граму-Шмидту векторы **b**\* являются вычислимыми за полиномиальное время. Интересная часть, это оценить качество вывода.

**Упражнение 1** Доказать, что для любого решеточного базиса **B** и целевого вектора , жадный алгоритм возвращает решеточный вектор *такой, что* , где – ортогонализация по Граму-Шмидту базисной матрицы **B**. В частности, .

Приведенное выше упражнение дает абсолютную зависимость расстояния между целью и точкой решетки, возвращенной алгоритмом. Но, это не решает : если **B** произвольная базисная решетка, жадный алгоритм может вернуть решения, которые произвольно хуже, чем наилучшие из возможных.

**Упражнение 2** Доказать, что для любого (возможно функции от n) существует вход (**B, t**) такой, что жадный алгоритм возвращает точку решетки на расстоянии от **t**, *где это расстояние от* **t** *до ближайшей точки решетки*. [Подсказка: вы можете получить плохие входы в размерности меньше 2.]

Иногда жадный алгоритм используют чтобы решить другой вариант ПБВ, названный проблемой Декодирования с Ограниченным Расстоянием (ДОР). Это проблема нахождения точки решетки, ближайшей к цели **t**, при условии, что **t** очень близка к решетке. (Обычно ближе, чем ) Следующее упражнение показывает, что жадный алгоритм решает ДОР, но опять же в пределах радиуса, который зависит от входного радиуса.

**Упражнение 3** Показать, что если , то Greedy(**B, t**) *возвращает (обязательно уникальную) точку решетки, ближайшую к* **t**.

Соединяя два упражнения вместе можно показать, что жадный алгоритм приближенно решает в пределах коэффициента приближения .

**Теорема 2** *Пусть* **B** *базис решетки и пусть* **B**\* *его ортогонализация по Граму-Шмидту. На входе* **B** *и любом целевом векторе* **t** *жадный алгоритм найдет решеточный вектор* *такой, что , где это расстояние от* **t** *до ближайшей точки решетки, и .*

Оценка качества приближения ПБВ полученная от жадного алгоритма зависит от базиса **B**, и более конкретно, от длины ортогонализованных векторов . Мы уже видели, что коэффициент приближения может быть сколь угодно плохим, если базис **B** выбран неправильно. Оценка в теореме предполагает, что хороший базис это такой, что все примерно одинаковые. Заметим, что произведение константно, потому что оно равно определителю решетки. Значит, если некоторые из них меньше, чем , то остальные будут больше, что приведет к плохому коэффициенту приближения. В следующей лекции мы будем изучать алгоритмы “сокращение базиса”, которые позволят вычислять решеточные базисы, которые, используемые вместе с жадным алгоритмом, позволят решать с коэффициентом , который зависит только от размерности решетки. Мы ожидаем, что, используя LLL алгоритм сокращения базиса, жадный алгоритм будет решать за полиномиальное время для .

**Упражнение 4** Показать, что большой интервал не обязательно означает, что жадный алгоритм дает плохие ПБВ приближения. *Более конкретно, покажите, что для любого (произвольно большого) есть решеточный базис* **B** *такой, что , и все же nounGreedy(***B, t***) решает ПБВ точно для цели* **t***.*

Для исторической справки, приведенный выше жадный алгоритм для приближения ПБВ использующий LLL сокращенные базы, был изначально предложен Бабаем, и поэтому его часто называют алгоритмом Бабая. Другое распространённое название алгоритма – *алгоритм ближайшей плоскости* из-за следующей естественной геометрической интерпретации: на каждом шаге, алгоритм выбирает значение *c*, соответствующее гиперплощадке ближайшей к цели.

## Метод Ветвей и Границ: Алгоритм Перечисления

Жадный алгоритм работает за полиномиальное время, но предоставляет только приближенные решения для ПБВ. Иногда нужно точно решить ПБВ, то есть найти точку решетку, ближайшую к цели. Это может быть сделано, используя ту же общую структуру, как и в жадном алгоритме, но используя несколько возможных значений для каждой переменной. Более конкретно, вместо жадной установки на наиболее подходящее значение (то есть на то, для которого нижняя граница расстояния 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 (расстояние от цели до начала координат) в качестве верхней границы. Обычно лучше использовать жадный алгоритм, чтобы найти приближенное решение **v** = Greedy(**B, t**), и использовать в качестве верхней границы. Как только верхняя граница *u* установлена, можно ограничить переменную *x* такими значениями, что .

Окончательный алгоритм похож на жадный и описан ниже:

Branch&Bound([],**t**) = **0**

Branch&Bound([**B***,***b**],**t**) = closest(*V* ,**t**)

где **b**∗ = **b**⊥**B**

**v** = Greedy(**B**,**t**)

*X* = {}

*V* = {*x* · **b** + Branch&Bound(**B***,***t** − *x* · **b**):*x* ∈ *X*}

где closest(*V, t*) выбирает вектор в ближайший к цели **t.**

Как и для жадного алгоритма, производительность (в данном случае время работы) алгоритма Ветвей и Границ может быть произвольно плохой, если мы сперва не сократим базисы.

**Теорема 3** *Число рекурсивных вызовов, совершенных алгоритмом Ветвей и Границ, принимающим на вход базис* **B** *и целевой вектор* **t**, не больше, чем .

Варианты алгоритма Ветвей и Границ были предложены несколькими авторами, включая Финке и Поста, Кэннона, Шнорра и Эйхнерра. Различные варианты часто часто относятся к разным именам, например сферический декодер или алгоритм перечисления, все основанные на естественной геометрической интерпретации алгоритма, который перечисляет все точки решетки в сфере вокруг целевого вектора. Разница между различными вариантами алгоритма обычно состоит в предварительной обработке базисной решетки (например, LLL уменьшен ли перед запуском основного алгоритма), разные выборы верхней границы функции *u*(**B, t**), порядок, в котором элементы выбираются, или детали низкоуровневой реализации. (Например, алгоритм чаще описывается с использованием вложенных циклов, чем с рекурсией, это может ускорить вычисление Грама-Шмидта путем использования чисел с плавающей запятой и т.д.) Два интересных варианта, которые заслуживают упоминания, это алгоритм перечисления Шнорра и Эйхнерра, который наиболее часто используется в криптоанализе и, как сообщается, превосходит остальные методы на практике, и алгоритм Кэннона, который в настоящее время лучший алгоритм полиномиального пространства для точного решения ПБВ

Основная идея алгоритма Шнорра и Эйхнерра состоит в том, чтобы обрабатывать возможные значения *x* в порядке не убывания значения , и динамически обновлять верхнюю границу *u* до расстояния от **t** к ближайшему найденному вектору решетки. В некотором смысле это гибрид между жадным алгоритмом и методом ветвей и границ, где наиболее многообещающее значение *x* выбрано первым, а затем рассматриваются остальные значения *x* (в порядке, показывающем их перспективность). Заметим, что первый вектор решетки, найденный стратегией перечисления, является в точности вектором, возвращаемым жадным алгоритмом для вычисления верхней границы расстояния. Алгоритм рассматривает все возможные значения *x* в порядке возрастания и отслеживает ближайший вектор решетки **v**, найденные после каждой итерации. Цикл заканчивается, когда .

Главная идея алгоритма Кэннона, это потратить намного больше времени на предобработку базиса **B** для того, чтобы ускорить процедуру перебора ветвей и границ. Более конкретно, Кэннон сперва вычисляет базис такой, что это кратчайший ненулевой вектор в решетке. Это выполняется сперва рекурсивной предобработкой (проекции, ортогональной к ) остатка базиса , и затем нахождением кратчайшего вектора, используя вариант алгоритма перечисления. Хотя рекурсивная предобработка, предложенная Кэнноном, предполагает большие накладные расходы (делая ее неконкурентоспособной на практике по сравнению с другими вариантами алгоритма), она имеет теоретическое преимущество, состоящее в значительном сокращении асимптотического времени выполнения всей процедуры перебора с до . К сожалению, накладные расходы такие, что алгоритм Кэннона на практике уступает другим асимптотически худшим стратегиям перечисления с временем выполнения в худшем случае . Существуют также другие алгоритмы (основанные не на перечислении) которые асимптотически даже быстрее чем Кэннон, со временем работы , но опять же неконкурентоспособны на практике.