Слайд 1

Здравствуйте, уважаемая комиссия! Я, Пивень Вадим Николаевич, представляю вашему вниманию выпускную квалификационную работу на тему: «Быстрая проверка многочленов над конечными полями на неприводимость и примитивность».

Целью моей работы являлся поиск наиболее эффективных методов проверки многочленов над конечными полями на неприводимость и примитивность, с последующей их реализацией в виде полноценного программного комплекса.

Первым этапом работы был поиск существующих алгоритмов и выбор наиболее эффективных из них. В результате к реализации были приняты 4 алгоритма.

Слайд 2

Первый алгоритм – алгоритм Берлекэмпа. Он является одним из первых появившихся алгоритмов проверки многочленов на неприводимость.

Псевдокод алгоритма Берлекэмпа приведён на слайде. Первые 10 строк предназначены для быстрого ответа в простых случаях, алгоритм может работать и без них. Аналогичные проверки реализованы во всех алгоритмах.

Наиболее вычислительно сложной задачей в алгоритме Берлекэмпа является построение матрицы Берлекэмпа и вычисление её ранга. Матрица Берлекэмпа представляет из себя коэффициенты многочленов вида, приведённого на слайде, записанных слева направо от младших к старшим.

Слайд 3

Следующий алгоритм – алгоритм Рабина. Он также предназначен для проверки многочленов на неприводимость. Его преимуществом перед алгоритмом Берлекэмпа является теоретически меньшая асимптотическая сложность.

Существенным недостатком данного алгоритма является большое потребление памяти. Так, для проверки неприводимого многочлена над полем $GF[P]$ степени n алгоритму Рабина потребуется хранить многочлен из P^n коэффициентов, в то время как алгоритму Берлекэмпа понадобится хранить не более P^2 коэффициентов.

Слайд 4

Третий алгоритм – алгоритм Бен-Ора, также предназначенный для проверки многочленов на неприводимость. Данный алгоритм является одним из наиболее новых, и в то же время наиболее эффективных. Асимптотическая сложность алгоритма Бен-Ора выше, чем для алгоритма Рабина. Однако, исследования

показывают, что фактически алгоритм Бен-Ора значительно опережает алгоритм Рабина по эффективности.

Слайд 5

Следующий алгоритм – алгоритм проверки многочленов на примитивность. Примитивные многочлены одновременно являются неприводимыми, кроме того, в некоторых задачах примитивные многочлены являются более предпочтительными.

Асимптотическая сложность данного алгоритма несколько выше, чем у алгоритма Рабина, в то время как его требования к памяти на порядок ниже.

Слайд 6

В ходе работы был проведён поиск существующих решений. Среди платных систем необходимым функционалом обладают Wolfram Mathematica и MATLAB. В существующем бесплатном аналоге MATLAB – Octave – соответствующая функция проверки на данный момент не реализована. Среди бесплатных решений, проверку многочленов на неприводимость умеют выполнять системы PARI/GP и FLINT. Обе эти системы распространяются под лицензией GNU GPL, что существенно ограничивает их применение в рамках несвободных проектов. Отдельно стоит упомянуть, что среди приведённых решений только библиотека FLINT умеет работать с составными полями.

В ходе выполнения представленной работы была реализована библиотека, выполняющая проверки многочленов над простыми конечными полями как на неприводимость, так и на примитивность. Код библиотеки распространяется под лицензией MIT и находится в свободном доступе на GitHub вместе с документацией, что позволяет практически без ограничений использовать её в составе других программных продуктов, в том числе коммерческих и несвободных. Библиотека реализована на языке C++.

Слайд 7

В данной работе класс многочлена над полем Галуа опирается на модульную арифметику, реализованную в классе чисел над полем Галуа, хотя в системах PARI/GP и FLINT вычисления по модулю являются честью функциональности типа-многочлена. Выбранный подход увеличивает объём используемой памяти, а также частоту пересоздания объектов, однако облегчает процессы тестирования и отладки, а значит повышает скорость разработки в целом.

Слайд 8

Для работы алгоритмов проверки потребовалось реализовать некоторые вспомогательные методы. Наиболее интересным из них является метод x_pow_mod, позволяющий уменьшить потребление памяти алгоритмами проверки, путём оптимизации операции деления многочленов с остатком.

Отдельно стоит упомянуть, что для одновременного выполнения нескольких проверок в разных потоках был реализован многопоточный конвейер.

Слайд 9

Финальным этапом работы стало тестирование полученных алгоритмов, а также оценка их производительности. Код проекта был покрыт unit-тестами и проверен двумя статическими анализаторами.

Для верификации правильности работы методов проверки, их результаты вместе с информацией о поле и проверяемом многочлене сохранялись, а затем вручную перепроверялись в системе Wolfram Mathematica.

Для сравнения скорости работы разных методов был выполнен бенчмаркинг. Генерировалась последовательность приведённых многочленов, содержащая заданное число примитивных, а значит и неприводимых многочленов. Многочлены из этой последовательности один за другим подавались на вход проверяющим функциям, до тех пор, пока среди них не было найдено необходимое число многочленов требуемой характеристики, время фиксировалось. Для каждого метода и каждого проверяемого поля выполнялось 100 последовательных запусков, время усреднялось.

Из полученных данных видно, что наиболее эффективным алгоритмом проверки многочленов на неприводимость является алгоритм Бэн-Ора, для всех полей, за исключением поля $GF[2]$, где наибольшую эффективность показал алгоритм Берлекэмпа. Кроме того, при поиске примитивных многочленов целесообразней вначале выполнять проверку на неприводимость, и лишь затем – на примитивность. С учётом данных рекомендаций были реализованы новые методы проверки, а также их многопоточные аналоги.

Слайд 10

Для оценки эффективности новых методов бенчмаркинг был выполнен повторно. Рекомендованные методы действительно оказались наиболее эффективными, однако выяснилось, что многопоточный конвейер не даёт ожидаемого прироста производительности. Это связано с тем, что для большинства многочленов методы проверки обеспечивают быстрый отрицательный ответ, а

получившие его потоки приостанавливаются в ожидании новых входных данных, расходуя дополнительное время. Это может быть исправлено путём изменения архитектуры класса-конвейера, однако тестирование было проведено слишком поздно, и автор не успел внести необходимые исправления.

Слайд 11

Было выполнено измерение времени работы рекомендованных алгоритмов проверки в зависимости от степени проверяемых многочленов. Для этого генерировались все приведённые многочлены заданной степени над полем $GF[3]$, а затем выполнялась их проверка.

Слайд 12

В заключение было выполнено сравнение времени работы реализованной библиотеки с найденными ранее аналогами. Платные системы Wolfram Mathematica и MATLAB показали худшие результаты, что может быть связано с их тяжеловесностью, а также применением интерпретаторов кода. Библиотека PARI/GP, также использующая интерпретатор, показала при этом превосходный результат, однако в процессе работы с ней было выяснено, что её функционал жёстко ограничен, а возможность генерации кода на C из кода оболочки GP не приспособлена для использования получаемых исполняемых файлов в качестве независимых динамических библиотек. Библиотека FLINT показала лучший результат при проверке многочленов на неприводимость, что обусловлено использованием её в качестве статической библиотеки, а также очень эффективным оперированием памятью. Реализованная в данной работе библиотека показала лучшее время для задачи проверки многочленов на примитивность.

Слайд 13

На этом моё выступление закончено, благодарю за внимание.