Московский государственный университет
имени М.В.Ломоносова

Механико-математический факультет

Кафедра Математической теории интеллектуальных систем

КУРСОВАЯ РАБОТА

Порождение, проверка аффинности и простоты конечных квазигрупп.
Generation, verification of the affinity and simplicity of finite quasigroups.

Выполнил студент 3 курса:
Р.А.Жигляев
Научный руководитель:
к.ф.-м.н., с.н.с. А.В.Галатенко

Москва, 2020

Аннотация

В работе приводится программная реализация алгоритмов порождения конечных квазигрупп, проверки простоты и аффинности порожденного объекта. Показана зависимость времени работы алгоритмов от порядка квазигруппы.

The paper provides a software implementation of algorithms for generating finite quasigroups, checking the simplicity and affinity of the generated object. The dependence of the running time of the algorithms is shown. from the order of a quasigroup.

1. Введение

Для решения определенных криптографических задач [1] возникает необходимость в использовании некоммутативных и неассоциативных алгебраических объектов, важное место среди которых занимают квазигруппы.

Особенно интересными оказываются алгебраические структуры, обладающие свойством полиномиальной полноты. Этот интерес обусловлен тем, что распознавание разрешимости системы уравнений в функционально полной алгебре является NP-полной задачей [2]. Исходя из того, что почти все квазигруппы являются полиномиально полными, можно сделать вывод, что квазигруппа, взятая случайным образом, почти всегда окажется полезной для решения задач криптографии. Известным является факт, что полиномиальная полнота квазигруппы эквивалентна неафинности и простоте [3].

Задачами данной работы являются разработка программной реализации на языке C++ известных алгоритмов: порождения квазигрупп конечного порядка [4], проверки квазигруппы на аффинность и простоту [5], а также исследование зависимости времени работы программы от порядка квазигруппы. Все эксперименты проводились на компьютере с процессором Intel(R) Core(TM) i5-5200U 2.20GHz, 4 ГБ ОЗУ.

В результате будет представлено описание программы и получено подтверждение теоретических оценок сложности работы алгоритмов.

Дальнейшая часть работы устроена следующим образом: в разделе 2 даны основные определения; разделы 3, 4, 5 посвящены алгоритмам генерации, проверки аффинности и простоты квазигрупп соответственно, а так же исследованию работы этих алгоритмов при разных порядках квазигрупп; в разделе 6 подводятся итоги работы.

2. Основные определения

Определение 1. Множество Q, на котором задана бинарная операция умножения f_Q, такая, что для любых элементов $a, b \in Q$ уравнения $ax = b$ и $ya = b$ однозначно разрешимы в Q, называется *квазигруппой.*

Операцию f_Q бывает удобно задавать в виде таблицы:

	$q_1 \ \ldots q_n$
q_1	$a_{11} \ \ldots a_{1n}$
$\ldots$	$\ldots\ldots\ldots$
q_n	$a_{n1} \ \ldots a_{nn}$

Здесь $\{q_1, \ldots, q_n\}$ - это множество элементов квазигруппы Q, а $a_{ij} = f_Q(q_i, q_j)$. Таблица умножения квазигруппы, как не сложно догадаться, является латинским квадратом.

Определение 2. Квазигруппа (Q, f_Q) называется аффинной, если на множестве Q можно ввести структуру абелевой группы $(Q, +)$, такую, что существуют автоморфизмы α, β группы $(Q, +)$ и элемент $c \in Q$, для которых выполнено тождество

$$f_Q(x, y) = \alpha(x) + \beta(y) + c.$$

Рассмотрим множество элементов квазигруппы $Q = \{q_1, \ldots, q_n\}$, $n \geq 2$ и некоторое разбиение α множества Q в объединение непересекающихся подмножеств $Q = A_1 \sqcup \ldots \sqcup A_m$. Будем называть разбиение α нетривиальным, если $m > 1$, $A_i \neq \emptyset$, $i = 1, \ldots, m$, и существует индекс j, $1 \geq j \geq m$, такой, что $|A_j| > 1$. В случае если $|A_1| = \ldots = |A_m|$, то такое нетривиальное разбиение будем называть равномерным. Элементы a, b, которые принадлежат одному множеству A_i далее назовем эквивалентными и будем использовать запись $a \sim b$.

Будем говорить, что f_Q сохраняет разбиение α, если для любой пары наборов (a_1, b_1), $(a_2, b_2) \in Q^2$, таких, что $a_i \sim b_i$, $i = 1, 2$, выполнено $f(a_1, b_1) \sim f(a_2, b_2)$. Как можно заметить, квазигрупповые операции могут сохранять только равномерные разбиения.

Определение 3. Квазигруппа (Q, f_Q) называется простой, если операция f_Q не сохраняет никакое нетривиальное разбиение Q.

3. Порождение случайной квазигруппы

3.1 Описание алгоритма

Для генерации квазигрупп воспользуемся алгоритмом, предложенным в [4]. Пусть $n \in \mathbb{N}$ порядок квазигруппы, которую необходимо получить. Рассмотрим матрицу инцидентности M размера $n * n * n$. Зададим M таким образом, что $M_{ijk} = 1$ при $k \equiv i + j \ (mod\ n)$, $1 \leq i, j, k \leq n$ и $M_{ijk} = 0$ иначе. Такой матрице соответствует квазигруппа (Q, f_Q), такая, что $Q = \{q_1, \ldots, q_n\}$ - множество элементов квазигруппы и $f_Q(q_i, q_j) = q_k$. Назовем матрицу инцидентности неправильной, если в одной из её клеток присутствует значение -1.

Рассмотрим саму процедуру алгоритма:

1) Если на текущем шаге матрица инцидентности является правильной, то выберем в ней случайную клетку, значение в которой равно нулю. Пусть координаты этой клетки $\{x, y, z\}$. В строках $M_{iyz}, M_{xjz}, M_{xyk}$ $1 \leq i, j, k \leq n$ выберем значения i_1, j_1, k_1 таким образом, что $M_{i_1yz} = M_{xj_1z} = M_{xyk_1} = 1$.

2) Если на текущем шаге матрица инцидентности неправильная, то выберем в ней клетку со значением равным минус единице и обозначим координаты этой клетки $\{x, y, z\}$. В строках $M_{iyz}, M_{xjz}, M_{xyk}$ $1 \leq i, j, k \leq n$ теперь присутствуют по две клетки со значением равным единице. Для каждой из этих строк выбираем по одной случайной единице из двух. Обозначим их $M_{i_1yz}, M_{xj_1z}, M_{xyk_1}$.

После выбора необходимых значений вычтем единицы из $M_{i_1yz}, M_{xj_1z}, M_{xyk_1}$, $M_{i_1j_1k_1}$ и добавим единицы к $M_{xyz}, M_{xj_1k_1}, M_{i_1yk_1}$, $M_{i_1j_1z}$. Если в результате $M_{ijk} = -1$ объявим матрицу неправильной. В противном случае, объявим матрицу правильной. Далее переходим к следующему шагу. В результате, если выполнено необходимое число итераций и матрица инцидентности на текущем шаге правильная, будет получена случайная квазигруппа, соответствующая текущей матрице инцидентности.

3.2 Описание программной реализации

В программной реализации в качестве необходимого числа итераций было выбрано n^3. На каждом шаге выполняется обращение к ячейкам памяти, а также добавление, вычитание, и поиск единицы. Обращение к ячейкам памяти и арифметические операции выполняются с константной сложностью. Для того, чтобы поиск единиц также выполнялся с констатной сложностью, матрица инцидентности хранится в памяти в виде трёх двумерных массивов $unitXY$, $unitXZ$, $unitYZ$, элементы которых соответствуют строкам матрицы и численно равны координатам единиц в этих строках. Например, при необходимости найти единицу в строке M_{ijk}, где i, j фиксированы, искомая координата $k = unitXY[i][j]$. Таким образом, задача поиска единицы также сводится к обращению к ячейке памяти и выполняется с констатной сложностью. Элементы массива $unitXY$ с индексами i, j по факту являются элементами генерируемой квазигруппы, находящимися в i-й строке и j-м стобце соответствующего латинского квадрата. Добавление и удаление единиц сводится к переопределению значений в массивах.

Такой подход, однако, не позволяет учитывать наличие в строке двух единиц и минус единицы. Для этого, в случае перехода матрицы M в неправильный вид, координаты ячеек содержащих вторые единицы и минус единицу сохраняются в отдельных переменных. В тот момент, когда происходит случайный выбор между единицами, одну из которых нужно обнулить, возможны два варианта:

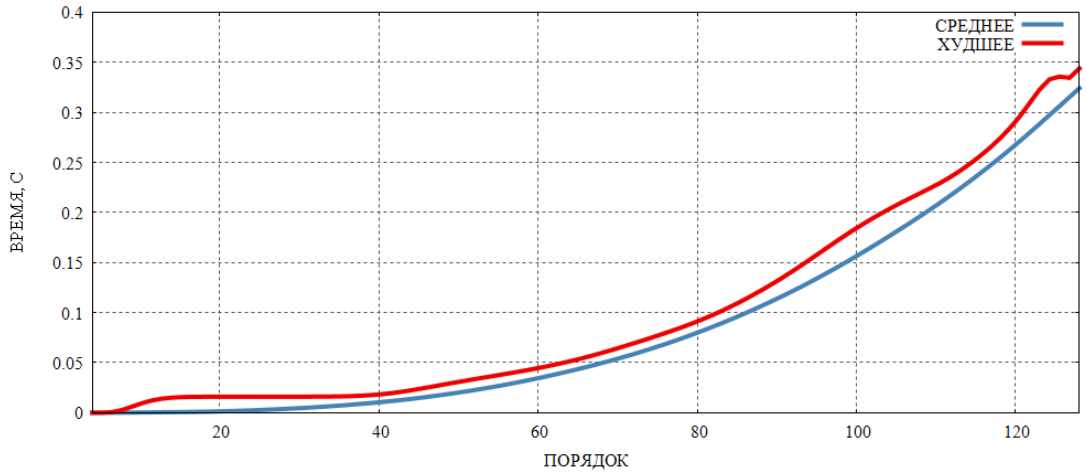
1) Если обнулить необходимо ту единицу, которая хранится в отдельной переменной, то в матрице хранятся координаты той единицы, которая не будет обнулена после выполнения шага, и эти координаты изменять не нужно.

2) Если обнулить необходимо единицу, которая хранится в матрице, то нужно поменять местами эти две единицы. Ту, что была в матрице, сохранить

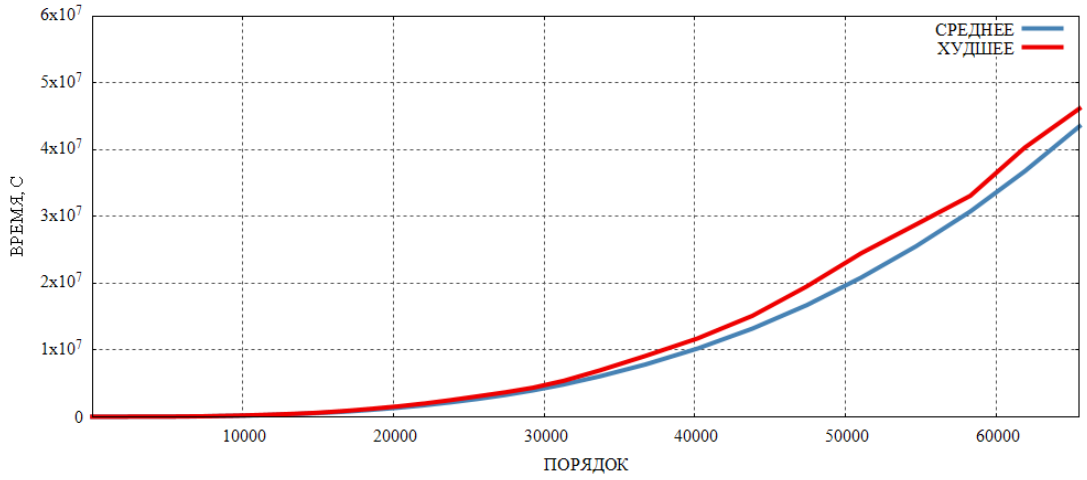
в отдельной переменной, а ту, что была в отдельной переменной, сохранить в матрице. Тем самым, задача свелась к первому пункту.

3.3 Описание эксперимента

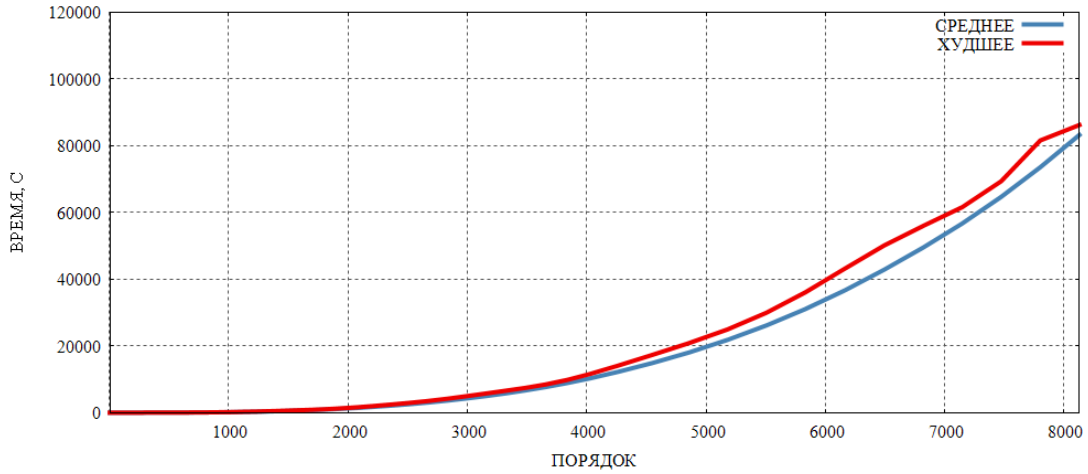
В рамках эксперимента проводилась генерация квазигрупп всех порядков от 4 до 128. Каждая генерация производилась по 100 раз. Замерялось среднее время и худшее время генерации за все 100 попыток. Результаты представлены на следующем графике:



Анализ графика показывает, что время генерации, как и ожидалось, имеет кубический рост. Это позволяет экстраполировать график на большие порядки. Считая, что в память компьютера можно поместить квазигруппы порядков до 2^{16}, экстраполяция проводилась именно до этого порядка. Результат:



Приведем также часть экстраполированного графика, соответствующую работе программы длительностью в одни сутки:



Таким образом, за сутки работы программы реально получить квазигруппы порядков около 8000.

4. Проверка аффинности

4.1 Описание алгоритма

Рассмотрим процедуру проверки квазигрупп на аффинность, подробное описание которой можно найти в [5]. Пусть L - латинский квадрат, являющийся таблицей умножения квазигруппы (Q, f_Q), $Q = \{q_1, ...q_n\}$. Будем обозначать σ_i перестановку, соответствующую i-й строке L.

Построим матрицу L' таким образом, что её i-я строка соответствует перестановке $\sigma_i * \sigma^{-1}$.

Переставим строки L' так, чтобы первый столбец совпадал с первой строкой. Полученную матрицу обозначим L'', а порожденную ей бинарную операцию обозначим $f_{L''}$.

В случае если матрица L'' не симметрична объявим квазигруппу не аффинной.

Проверим, что равенство

$$f_{L''}(f_{L''}(q_r, q_s), q_t) = f_{L''}(q_r, f_{L''}(q_s, q_t));$$

выполнено для любой тройки $1 \leq r, s, t \leq n$. В случае, если хотя бы для одной тройки равенство не выполнено объявим квазигруппу не аффинной.

Найдем строку и столбец матрицы L, первые элементы которых совпадают с левым верхним элементом матрицы L''. Обозначим соответствующие им перестановки за α и β соответственно.

Проверим, что равенства

$$\alpha(f_{L''}(q_i, q_j)) = f_{L''}(\alpha(q_i), \alpha(q_j));$$
$$\beta(f_{L''}(q_i, q_j)) = f_{L''}(\beta(q_i), \beta(q_j));$$

выполнены для любой пары $1 \leq i, j \leq n$. Если хотя бы для одной пары будет получено неравенство, объявим квазигруппу не аффиной.

Проверим, что для любой пары $1 \leq i, j \leq n$ выполнено равенство

$$f_Q(q_i, q_j) = f_{L''}(f_{L''}(\alpha(q_i), \beta(q_j)), c);$$

где c - левый верхний элемент матрицы L. Если хотя бы для одной пары равенство не выполнено, объявим квазигруппу не аффинной. В противном случае, объявим квазигруппу аффинной.

4.2 Описание программной реализации

Будем считать, что на старте работы алгоритма мы уже храним в памяти матрицу L в виде двумерного массива. Создадим матрицу L' так же как двумерный массив. Поскольку строки L' являются произведениями перестановок σ_i и σ^{-1}, то элемент $L'_{ij} = L_{i\sigma_j^{-1}}$, $1 \leq i, j \leq n$. Здесь σ^{-1} хранится в виде одномерного массива, элементы которого могут быть получены из соображений, что $\sigma^{-1}_{L_{1j}} = j$. В результате построения L', первая строка матрицы будет соответствовать тождественной перестановке.

Таким образом, для построения матрицы L'' достаточно переставить строки $2, ..., n$ матрицы L' так, чтобы первый столбец также представлял из себя тождественную перестановку. Это осуществляется путем сортировки строк по первому элементу столбца. После осуществления процедуры на месте матрицы L' в памяти будет храниться матрица L''.

Проверка L'' на симметричность осуществляется предельно просто. Достаточно пройти в цикле по левому верхнему треугольнику матрицы и проверить на равенство элементы L''_{ij} и L''_{ji}, $1 \leq i, j \leq n$, $i \neq j$.

Поиск перестановок α и β также осуществляется путем проходов по первым столбцу и строке матрицы L и поиску в них элемента q_1, который и является левым верхним элементом L''.

Проверка выполнения всех равенств в алгоритме выполняется путем перебора всех требуемых комбинаций и сравнению нужных элементов, что сводится к обращению к ячейкам памяти.

4.3 Описание эксперимента

Эксперимент проводился на всех квазигруппах, полученых в пункте 3.3. Измерялось среднее время работы программы при определенных значениях порядка квазигруппы. Результат представлен на графике:

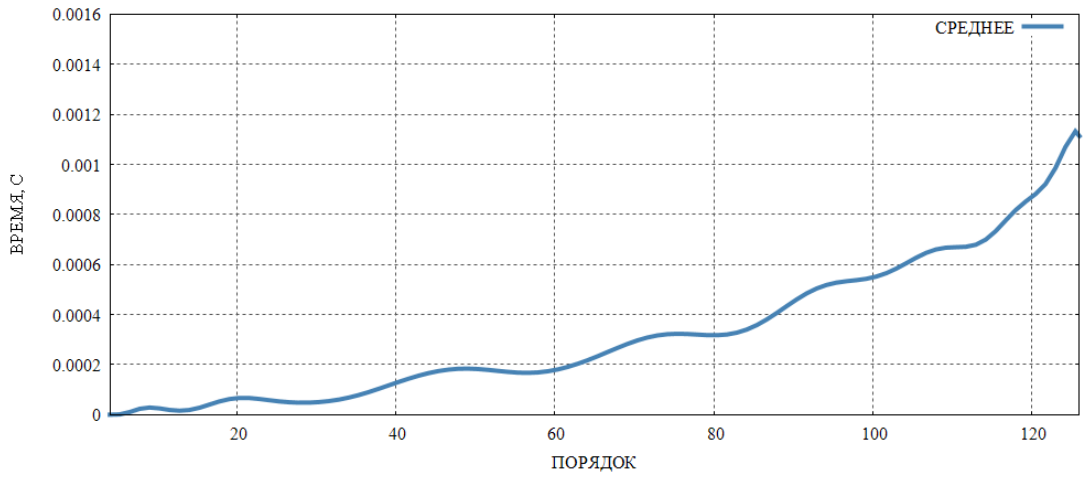
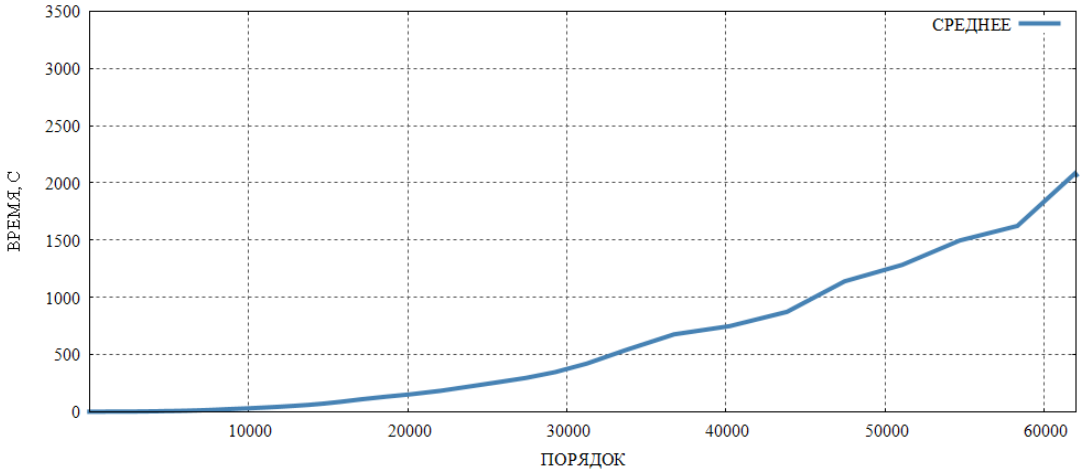


График имеет рост около $n^{2.35}$. Такое значение можно объяснить тем, что почти все квазигруппы неаффинны. Часто алгоритм завершает работу еще во время процедуры проверки коммутативности, которая имеет квадратичную сложность. Приведем экстраполированный график на порядки до 2^{16}:



Можно заметить, что даже при самых больших порядках, алгоритм выполнит работу не более чем за час, что значительно быстрее, чем время получения такой квазигруппы.

5. Проверка простоты

5.1 Описание алгоритма

Пусть (Q, f_Q) - квазигруппа, $Q = \{q_1, ..., q_n\}$, L - латинский квадрат задающий f_Q. Для проверки простоты квазигруппы будем использовать алгоритм, предложенный в [5], и его последующую оптимизацию из [6].

Зафиксируем индекс i_0 и для всех пар, вида $\{q_{i_0}, q_j\}$, $1 \leq j \leq n$, $j \neq i_0$ рассмотрим следующую процедуру:

1. Каждой неупорядоченной паре $1 \leq t_1, t_2 \leq n$, $t_1 \neq t_2$, сопоставим две метки - рассмотренность и эквивалентность. Изначально все пары считаем нерассмотренными и неэквивалентными. Отмечаем пару $\{i_0, j\}$ как эквивалентную.

2. Возьмем произвольную нерассмотренную эквивалентную пару $\{a, b\}$. Для всех перестановок σ, соответствующих строкам и столбцам L будем рассматривать элементы $\sigma(a)$ и $\sigma(b)$. Если их эквивалентность еще не была установлена, объединим соответствующие им классы эквивалентности и отметим новые эквивалентные пары. После рассмотрения всех перестановок помечаем пару $\{a, b\}$ как рассмотренную.

3. Если множество нерассмотренных эквивалентных пар пусто или все элементы попарно эквивалентны, заканчиваем работу, в противном случае переходим к шагу 2.

Если для всех $\{i_0, j\}$ процедура завершилась после того, как все элементы оказались попарно эквивалентными, объявим квазигруппу простой. В противном случае, квазигруппа простой не является.

В оптимизированном варианте алгоритма считается, что нетривиального разбиения не существует, если есть хотя бы один класс эквивалентности, состоящий более чем из $\frac{n}{2}$ элементов. Так же утверждается, что достаточно добавлять в очередь на проверку только одну пару из нового класса эквивалентности.

5.2 Описание программной реализации

Для простоты в качестве q_{i_0} будем брать q_1. Тогда необходимо выполнять указанную процедуру для всех пар вида q_1, q_j, $2 \leq j \leq n$.

Неупорядоченную пару t_1, t_2 будем задавать в виде структуры с двумя полями: два целых числа t_1 и t_2. Будем хранить классы эквивалентности в виде массива из n контейнеров $std :: vector$. Также сохраним массив из n целых чисел, i-й элемент которого будет содержать номер класса эквивалентности, в котором находится элемент i. Это позволит определять, в каком классе находится нужный элемент за константное время и избавит от нужды помечать новые эквивалентные пары.

Так как существует необходимость обращаться к нерассмотренным эквивалентным парам, было бы удобно хранить их в памяти. В качестве контейнера для них будем использовать структуру данных "очередь". При этом, пары, не находящиеся в очереди на проверку, можно не сохранять. После объединения двух классов будет происходит добавление одной пары в очередь, а после её рассмотрения - удаление её из очереди.

Рассмотрение перестановок выполняется перебором строк и столбцов матрицы, хранящей латинский квадрат L, которая является двумерным массивом. Для того, чтобы узнать, является ли пара эквивалентной, достаточно узнать принадлежат ли элементы одному классу, что делается путем обращения к соответствующему массиву. Когда возникает необходимость соединить два класса эквивалентности - элементы вектора, соответствующего одному из классов добавляются в вектор второго класса и удаляются из первого, а номера классов эквивалентности переброшенных элементов изменяются на новое значение.

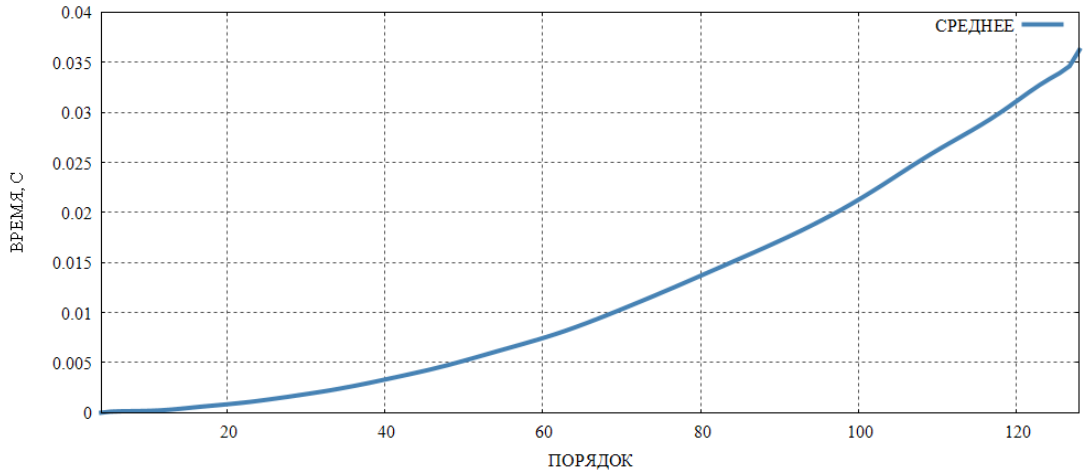
Для того, чтобы не выполнять проверку на то, являются ли все элементы попарно эквивалентными, введем переменную - счетчик. При объединении двух классов, состоящих из m и k элементов соответственно, число эквивалентных пар увеличивается на $m \cdot k$. На то же число увеличивается переменная счетчик.

Аналогично введем переменную, хранящую размер наибольшего из классов. При объединении двух классов, состоящих из m и k элементов соответственно получится класс размером $m + k$. Если это значение превышает текущий максимальный размер класса, то переменной присваивается новое значение. Если это значение окажется больше чем $\frac{n}{2}$, то, как было сказано в описании алгоритма, нетривиального разбиения не будет существовать.

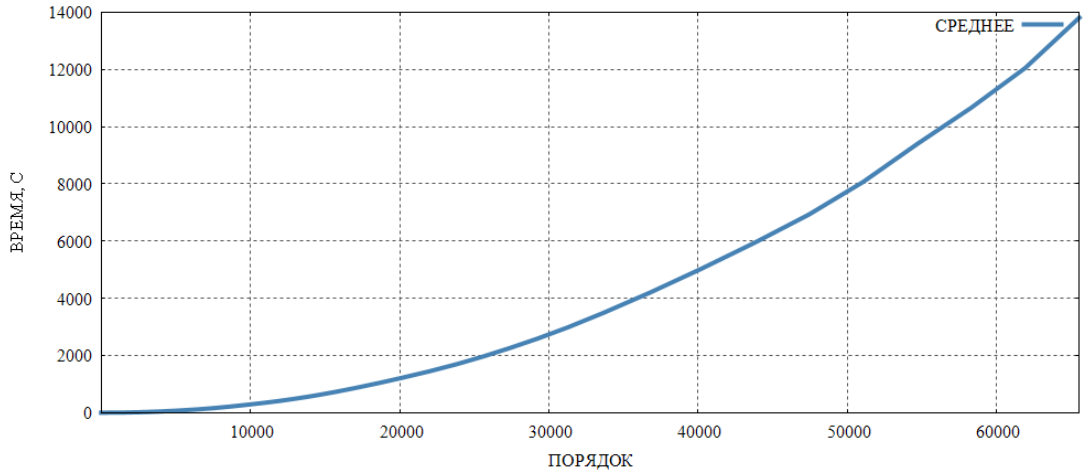
Таким образом, если число эквивалентных пар не равно числу неупорядоченных пар, то есть $\frac{n \cdot (n-1)}{2}$, и при этом, размер максимального класса не превосходит $\frac{n}{2}$, а нерассмотренных эквивалентных пар не осталось, то квазигруппа не является простой.

5.3 Описание эксперимента

Эксперимент проводился на квазигруппах, полученных в пункте 3.3. Аналогичным образом замерялось время работы программы в зависимости от порядка сгенерированной квазигруппы. Приведем среднее время работы:



Полученный график имеет рост около $n^{2.05}$. Приближенность к квадратичному росту связана с тем, что зачастую в алгоритме не требуется проверять все пары, находящиеся в очереди. Почти все квазигруппы простые, и, в большинстве случаев, классы, содержащие больше половины элементов, появляются довольно быстро, что позволяет осуществлять переход к следующей итерации внешнего цикла. На основе анализа роста можно экстраполировать график на порядки до 2^{16}:



Таким образом, проверка на простоту даже при самых больших рассматриваемых порядках не занимает больше 4 часов.

6. Заключение

Как видно из результатов работы, проверка аффинности и простоты сгенерированной квазигруппы занимает значительно меньше времени, чем сама генерация. Алгоритм генерации в среднем имеет кубический порядок роста, а алгоритмы проверки аффинности и простоты зачастую выполняются быстрее заявленной максимальной сложности.

Список литературы

[1] М. М. Глухов, "О применениях квазигрупп в криптографии" , Прикладная дискретная математика, 2008, № 2, 28–32.

[2] G. Horváth, Gh. L. Nehaniv, Cs. Szabó, "An assertion concerning functionally complete algebras and NP-completeness" , Acta Sci. Math. (Szeged), 76 (2010), 35-48.

[3] J. Hagemann, C. Herrmann, "Arithmetical locally equational classes and representation of partial functions" , Universal Algebra, Esztergom (Hungary), 29 (1982), 345-360.

[4] M.T. Jacobson, P. Matthews, "Generating uniformly distributed random Latin squares" , J.Combin. Des., 4(6) (1996), 405–437.

[5] А. В. Галатенко, А. Е. Панкратьев, "О сложности проверки полиномиальной полноты конечных квазигрупп" , Дискрет. матем., 30:4 (2018), 3–11

[6] A. V. Galatenko, A. E. Pankratiev, V. M. Staroverov, "Efficient verifiation of polynomial completeness of quasigroups" , Lobachevskii Journal of Mathematics