РЕФЕРАТ

Выпускная квалификационная работа содержит: 26 страниц, 11 рисунков, 5 источников.

МАТРИЦА АДАМАРА, КЛАССЫ ЭКВИВАЛЕНТНОСТИ, ЭКВИВАЛЕНТНОСТЬ МАТРИЦ ПО ХОЛЛУ, АЛГОРИТМ ПОИСКА МИНИМАЛЬНОЙ МАТРИЦЫ, КРИПТОГРАФИЯ.

В выпускной квалификационной работе рассмотрен и реализован алгоритм нахождения минимальной матрицы и проверки эквивалентности двух матриц Адамара, изложены математическое описание и реализация алгоритма.

В теоретической части работы дано математическое описание исследуемых сущностей, необходимых для реализации алгоритма.

В практической части приведены детали реализации, описаны тестовые данные, на которых запускался алгоритм, проведено сравнение результатов работы наивного и эффективного алгоритмов.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Выпускная квалификационная работа посвящена исследованию матриц Адамара. Матрицы Адамара находят множество применений в разных областях, таких как:

- комбинаторика,
- теория кодирования,
- численный анализ,
- цифровая обработка сигналов,
- другие.

Рассмотрим некоторые приложения матриц, чтобы обозначить актуальность данной работы.

Одним из применений это преобразование Уолша-Адамара, которое широко используется в качестве быстрого дискретного преобразования. Коды с исправлением ошибок (коды Рида-Мюллера), используемые при передаче сигналов спутникам, основаны на матрицах Адамара. Каналы передачи данных с множественным доступом путем разделения по кодам (CDMA) используют матрицы Адамара для модуляции передачи по восходящей линии связи и минимизации помех от других передач. Повсюду нас окружают новые приложения, например, в области распознавания образов, оптической связи и сокрытия информации. Несмотря на это, до сих пор нет единого метода построения всех известных матриц Адамара.

Матрицы Адамара были расширены и обобщены для недвоичных алфавитов и для более высокой размерности, а их отличительные свойства адаптированы для многоуровневых и многофазных приложений в обработке сигналов, кодировании и криптографии.

Матрицы Адамара безусловно являются важным объектом исследований и могут быть использованы для решения многих математических задач. Вышеперечисленные области применений лишь подтверждают этот факт.

Целью данной работы является создание алгоритма нахождения минимальной матрицы данного класса эквивалентности и проверки матриц

Адамара на эквивалентность, с помощью которого можно будет находить переход между классами эквивалентности.

ОСНОВНАЯ ЧАСТЬ

1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1 Матрица Адамара

Матрица Адамара – квадратная матрица порядка n, состоящая из элементов 1 и -1, строки (столбцы) которой ортогональны друг другу. То есть выполняется условие

$$H \cdot H^T = n \cdot I, \tag{1.1}$$

где I– единичная матрица порядка n, $n = 1,2,4k$ при натуральном k.

Пример Матрицы Адамара 4-го порядка:

$$H_4 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}$$

1.2 Классы эквивалентности

Две матрицы Адамара одинакового порядка называются эквивалентными по Холлу, если одна может быть получена из другой с помощью любой комбинации следующих операций:

1) перестановка двух строк

2) перестановка двух столбцов

3) умножение на -1 любой строки

4) умножение на -1 любого столбца

Нетрудно посчитать, что для матрицы размером $m \times n$ существует $n!$ перестановок столбцов, $m!$ перестановок строк, 2^n способов умножения столбцов на -1 и 2^m способов умножения строк на -1. Поэтому существует $m! \times n! \times 2^{(m+n)}$ различных матриц, которые являются эквивалентными. Любой класс эквивалентности имеет такое количество элементов.

Матрица Адамара называется нормализованной, если первые строка и столбец состоят из 1. Любой класс эквивалентности будет содержать хотя бы одну нормализованную матрицу.

В общем, можно легко получить много разных матриц, которые эквивалентны исходной, применяя любую комбинацию вышеупомянутых операций, которые сохраняют эквивалентность. Но в то же время довольно сложно проверить эквивалентны или нет две исходных матрицы Адамара одинакового порядка.

Порядок матрицы	Число классов эквивалентности
1, 2, 4, 8, 12	1
16	5
20	3
24	60
28	487
32	13710027
>=36	неизвестно

Рис. 1.1 Число классов эквивалентности матриц Адамара

В таблице мы можем увидеть количество классов эквивалентности для соответствующих порядков матриц. Нетрудно заметить, что при увеличении порядка, увеличивается и число классов, причем рост числа классов заметно быстрее роста порядка. Поэтому для матриц порядка выше 32 число классов эквивалентности еще не посчитано. Сложность подсчета классов связана с тем, что надо уметь находить неэквивалентные друг другу матрицы за приемлемое время, что является пока еще нерешенной задачей.

1.3 Операции переключения

Как уже было замечено, при увеличении порядка число классов растет очень быстро, до сих пор не существует общего метода для получения количества классов для больших порядков. В связи с чем возникают две проблемы – высокая вычислительная сложность перечисления всех матриц

одного класса и нахождение всех неэквивалентных классы (нет понимания как осуществляется переход от одного класса к другому). Как можно решить эти проблему? Можно ввести новые операции, которые сохраняют свойства матрицы Адамара. Эти операции, как правило, порождают много неэквивалентных матриц Адамара. Кроме того, добавление новых операций дает новое понятие эквивалентности. Эта более «слабая» эквивалентность может быть полезна при классификации матриц Адамара, поскольку она разбивает множество матриц на гораздо меньшее число классов эквивалентности, чем эквивалентность по Холлу, но в то же время обеспечивает более эффективное перечисление всех элементов новых классов эквивалентности.

Получив новое определение эквивалентности, так называемую Q-эквивалентность, которая уменьшит количество классов, мы сможем вычислять их быстрее. Так, для порядков 4, 8, 12, 16, 20, 24, 28 известны количества «классических» классов эквивалентности – 1, 1, 1, 5, 3, 60, 487, тогда как для Q-эквивалентности число таких классов равно 1, 1, 1, 1, 1, 2, 2.

1.4 Минимальная матрица

Заметим, что если в матрице Адамара все элементы 1 заменить на 0, а элементы -1 заменить на 1, то исходная матрица превратится в двоичную матрицу.

Определим функцию ρ, которая отображает двоичную матрицу в целое число. Наш алгоритм будет минимизировать это значение в данном классе эквивалентности. Элемент класса, который имеет минимальное ρ, является уникальным, назовём его минимальной матрицей.

Пусть $A = (a_{ij})$ – двоичная матрица размера $m \times n$, где $i = 1,2, \dots, m$ и $j = 1,2, \dots, n$. Тогда,

$$\rho(A) = \sum_{i=1}^{m} \sum_{j=1}^{n} \left[a_{ij} 2^{n(m-i)+(n-j)}\right] \tag{1.2}$$

Пример преобразования:

$$\rho\begin{pmatrix}1 & 1 & 1 & 1\\ 1 & 0 & 1 & 0\\ 1 & 1 & 0 & 0\\ 1 & 0 & 0 & 1\end{pmatrix} = 1111101011001001_2 = 64201$$

Получим отображение множества бинарных матриц размера $m \times n$ в целые числа в диапазоне от 0 до $2^{mn} - 1$. Это составное отображение: первая часть отображение из множества двоичных матриц $m \times n$ в множество двоичных векторов длины mn путем последовательного объединения всех строк матриц; а вторая часть – из двоичного вектора в целое число естественным образом.

Пусть S – множество бинарных матриц размера $m \times n$ и ρ отображает S в множество целых чисел в диапазоне от 0 до $2^{mn} - 1$. Заметим следующие свойства функции ρ:

1) отношение порядка: ρ порождает линейно упорядоченное множество. То есть для любых матриц A и B из S выполняется либо $\rho(A) < \rho(B)$, либо $\rho(A) > \rho(B)$, либо $\rho(A) = \rho(B)$.

2) ρ – взаимно однозначное отображение, то есть для любых A и B из S $\rho(A) = \rho(B)$ тогда и только тогда, когда $A = B$.

3) существует минимальный элемент: любое подмножество S_0 в S должно содержать элемент M_0, который называется минимальной матрицей в S_0, такой что $\rho(M_0)$ имеет минимальное значение ρ в S_0. Кроме того, поскольку ρ взаимно однозначно, такая минимальная матрица определяется однозначно.

Минимальная матрица L двоичной матрицы A размера $m \times n$ является минимальной матрицей класса эквивалентности, содержащего A.

Из всего вышесказанного следует, что для определения эквивалентности двух матриц нам достаточно найти минимальные матрицы каждой из матриц и сравнить их. Если матрицы равны, то исходные матрицы принадлежат одному классу, если не равны, то разным.

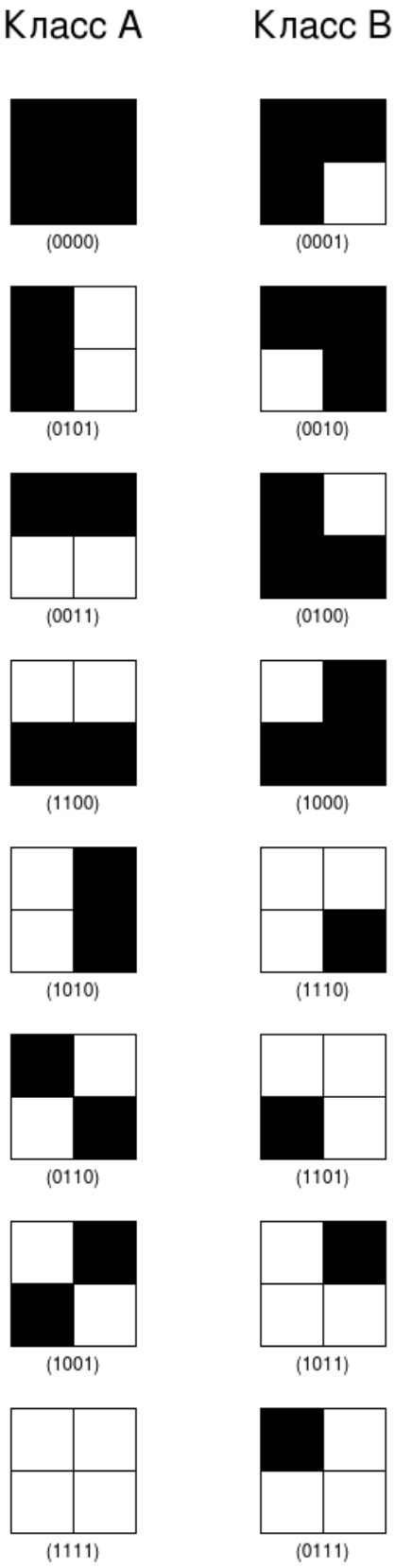

Рис. 1.2 Все бинарные матрицы порядка 2 в двух классах эквивалентности

На рис. 1.2 показаны все бинарные матрицы порядка 2. Матрицы класса A не являются матрицами Адамара. Класс B содержит все эквивалентные матрицы Адамара порядка 2. Минимальная матрица этого класса имеет значение $\rho = 0001_2$.

1.5 Алгоритм нахождения минимальной матрицы Адамара

Введем некоторые обозначения:

1. Если H– двоичная матрица, имеющая n столбцов, то $H(r)$ обозначает вектор ее r-й строки. То есть $H(r) = [h_{r1,}h_{r2,} \dots, h_{rn}]$.
2. Будем использовать $C_C(C_R)$ для обозначения множества операций, которые инвертируют столбцы (строки). Для матриц размера $m \times n$ $|C_C| = 2^n$ и $|C_R| = 2^m$.
3. Будем использовать $P_C(P_R)$для обозначения множества перестановок столбцов (строк). Для матриц размера $m \times n$ $|P_C| = n!$ и $|P_R| = m!$.
4. Column-Sort (или Row-Sort) – это сортировка столбцов (строк), при которой значение ρ достигает минимума.
5. Нормализация – это процесс преобразования заданной двоичной матрицы H в форму, в которой первая строка и первый столбец состоят только из нулей. $B = N(A)$, когда B является нормализованной формой матрицы A.

Ключевые моменты предлагаемого алгоритма:

1. Минимальная матрица должна быть в нормализованной форме.
2. Для заданной двоичной матрицы H и ее минимальной матрицы L существуют матрицы перестановок P_r и P_c, такие что $L = N(P_r H P_c)$.
3. Когда P_r и первый столбец P_c известны, остальные столбцы P_c могут быть определены из пункта 2.

Основная идея поиска минимальной матрицы для исходной двоичной матрицы состоит в том, чтобы использовать эти сведения. Однако их нельзя использовать так, как они сформулированы, поскольку матрица L неизвестна изначально.

Основной алгоритм находит минимальную матрицу путём проверки всех возможных перестановок строк P_r для исходной двоичной матрицы. Он находит P_r и первый столбец P_c, удовлетворяющие равенству $L = N(P_r H P_c)$. Чтобы осуществить нормализацию, алгоритм определяет первый столбец P_c и первую строку P_r. После нормализации алгоритм вызывает функцию $core$, которая определяет r-ую строку P_r. Описание функции $core(r, flag)$ следующее:

1. Найти строки среди строк с индексами от r до m в H, которые минимизировали бы значение $\rho\big(H(r)\big)$, если поменять их местами с r-тыми строками и выполнить все возможные перестановки столбцов, которые не изменяют элементы в индексах строк от 1 до $r-1$. Пусть M – вектор строки $H(r)$ с указанным выше свойством.

2. Сравнить $\rho\big(A(r)\big)$ и $\rho(M)$, найденные в шаге 1. Если $\rho(M) < \rho\big(A(r)\big)$ или $flag = True$, то $A(r)$ присвоить M.

3. Если $\rho(M) \leq \rho\big(A(r)\big)$, выполните следующие действия для всех строк из шага 1:

3.1. Поменять местами выбранную строку с r-й строкой H.

3.2. Найти перестановку столбцов, которая дает наименьшее значение $\rho(H)$. То есть, выполнить сортировку столбцов H.

3.3. Если $r < m$, вызвать $core(r+1, flag)$ с $flag$, определенным в результате шага 2.

4. Завершить работу функции

1.6 Оценка вычислительной сложности алгоритма

В наивном алгоритме нам необходимо сгенерировать все возможные матрицы данного класса эквивалентности, найти ρ для каждой такой матрицы и обновлять текущую минимальную матрицу при нахождении новой минимальной матрицы. Для этого нам потребуется выполнить $m!$ операций перестановки строк, $n!$ операций перестановки столбцов и $2^{(m+n)}$ операций инвертирования

каждых строк и столбцов. Оценка сложности наивного алгоритма $O\left(m!\,n!\,2^{(m+n)}\right)$.

Рассмотрим сложность эффективного алгоритма. Поскольку минимальная L имеет наименьшее значение ρ в классе эквивалентности H, оно также должно иметь наименьшее значение ρ в любом подмножестве класса, содержащего L Поэтому, зная H, P_r и первый столбец P_C мы можем определить оставшиеся столбцы P_C так, чтобы $N(P_r H P_c)$ имело наименьшее значение ρ. Заметим, что нахождение P_C может быть выполнено быстрее чем за $(n-1)!$. Например, за $nlog(n)$, если мы выполним сортировку матрицы по столбцам с использованием любого алгоритма сортировки сравнением за $O\left(nlog(n)\right)$. Общая сложность становится $O\left(m!\,n^2 log(n)\right)$

Для исходной двоичной матрицы H размером $m \times n$ можно найти ее минимальную матрицу L с временной сложностью $O\left(m!\,n^2 log(n)\right)$.

1.7 Оптимизация базового алгоритма

Можно заметить, что для матриц большого порядка, время работы алгоритма увеличивается, но в то же время, минимальная матрица почти всех исходных матриц находится достаточно рано. Поэтому если две матрицы эквивалентны, то минимальная матрица может быть найдена в достаточно короткие сроки. Новый алгоритм проверки эквивалентности матриц работает по следующей схеме:

1. Установим ограничение по времени работы основного алгоритма и по количеству итераций.

2. Запустим основной алгоритм на исходной матрице A.

3. Когда пройдет установленное время или количество итераций, остановим выполнение алгоритма и получим вычисленную к этому моменту матрицу.

4. Запустим основной алгоритм на исходной матрице B.

5. Когда пройдет установленное время или количество итераций, остановим выполнение алгоритма и получим вычисленную к этому моменту матрицу.

6. Сравним полученные матрицы. Если они разные, то продолжить выполнение вычислений в пунктах 2 и 4. Если они одинаковые, то мы можем предположить, что A и B эквивалентны.

7. Сохранить полученные матрицы.

8. Выполнить случайное преобразование матриц A и B с помощью операций, сохраняющих эквивалентность.

9. Повторить пункты 2-8 достаточное количество раз.

10. Если все сохраненные матрицы равны, то можно считать исходные матрицы эквивалентными. Если они отличаются, то повторить вычисления еще раз с увеличенными ограничениями по времени или по количеству итераций.

С помощью этого алгоритма, можно узнать, эквивалентны ли матрицы. Но мы не можем с уверенностью утверждать, что матрицы неэквивалентны. Проверка неэквивалентности может быть осуществлена только по основному алгоритму. Алгоритм можно модифицировать так, чтобы в результате его работы, мы давали вероятностный ответ, являются ли матрицы эквивалентными. Например, сравнить наибольшее (наименьшее) ρ полученных из A матриц и наименьшее (наибольшее) ρ полученных из B матриц. Часто наименьшее ρ одного набора матриц больше, чем наибольшее ρ другого набора матриц, если исходные матрицы неэквивалентны.

2 ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1 Инструментарий разработки

Описанный алгоритм был реализован на языке C. Выбор языка обусловлен в первую очередь скоростью работы. Язык позволяет при необходимости осуществлять низкоуровневые оптимизации.

Для сборки проекта используется система сборки Cmake. Для абстрагирования запуска тестов от аппаратной части рабочей машины была

использована система виртуализации Docker. Эти инструменты ускоряют скорость разработки, позволяют другим разработчикам начать использовать существующую кодовую базу как можно быстрее, не тратя время на конфигурирование среды разработки.

Реализация разбита на три логических компонента – реализацию матрицы, реализацию наивного алгоритма и реализацию эффективного алгоритма. Также было автоматизировано тестирование. Непосредственно для запуска тестов была использована библиотека Check. При каждом запуске тестов формируется отчёт в формате xml, который содержит информацию о длительности работы каждого тестового случая, что освобождает нас от необходимости писать бенчмарки с измерением времени работы. Для визуализации результатов, а конкретно для получения графиков зависимости времени работы алгоритмов от порядка матрицы и визуализации бинарной матрицы, был использован язык Python и модуль matplotlib. Нам остается только распарсить отчёт тестирования и построить нужные графики.

Для удобства разработки были использованы система контроля версий git и веб-сервис хостинга проектов GitHub, которые хранят всю историю изменений кода. На момент написания этой работы существует три хранимые версии разработанного алгоритма, быстрый доступ к которым позволил ускорить аналитическую работу.

Ссылка на проект: https://github.com/tutkarma/hadamard-matrix

Характеристики машины, использованной для вычислений, приведены ниже.

- Процессор Intel Core i7-6500U CPU 2.50GHz x 4
- Оперативная память: 8Gb
- Операционная система: Ubuntu 16.04 LTS
- Компилятор: gcc 9.3

2.2 Реализация наивного алгоритма

В реализации наивного алгоритма нет ничего примечательного. Псевдокод алгоритма представлен ниже. Сама реализация в точности соответствует тому, что представлено на рисунке.

```
   input  : матрица H размера m × n
   output: минимальная матрица A
 1 A = H
 2 while существует следующая перестановка строк do
 3 |  Получить следующую перестановку строк и переставить строки в матрице
   |   H_tmp, полученной из H
 4 |  while существует следующая перестановка столбцов do
 5 |  |  Получить следующую перестановку столбцов и переставить столбцы в
   |  |   матрице H_tmp
 6 |  |  for i = 1 to n do
 7 |  |  |  Получить все сочетания строк C_n^i
 8 |  |  |  for j = 0 to размер C_n^i do
 9 |  |  |  |  Инвертировать все строки в j-ом сочетании в матрице H_tmp
10 |  |  |  |  for k = 1 to m do
11 |  |  |  |  |  Получить все сочетания столбцов C_m^k
12 |  |  |  |  |  for l = 0 to размер C_m^k do
13 |  |  |  |  |  |  Инвертировать все столбцы в l-ом сочетании в матрице H_tmp
14 |  |  |  |  |  |  if ρ(H_tmp) = ρ(A) then
15 |  |  |  |  |  |  |  A = H_tmp
16 |  |  |  |  |  |  end
17 |  |  |  |  |  end
18 |  |  |  |  end
19 |  |  |  end
20 |  |  end
21 |  end
22 end
```

Рис. 2.1 Псевдокод наивного алгоритма

2.3 Реализация основного алгоритма

Основной алгоритм, ожидаемо, состоит из двух функций – min_matrix и core, псевдокод который представлен ниже.

```
   input  : матрица H_0 размера m × n
   output: минимальная матрица A
 1 A = N(H_0)
 2 H = H_0
 3 for j = 1 to n do
 4 |  Поменять местами 1-ый и j-ый столбцы матрицы H
 5 |  for i = 1 to m do
 6 |  |  Поменять местами 1-ую и j-ую строки матрицы H
 7 |  |  H = N(H)
 8 |  |  Вызов функци core(2, False)
 9 |  |  Поменять местами 1-ую и j-ую строки матрицы H
10 |  end
11 |  H = H_0
12 end
```

Рис. 2.2 Псевдокод основной функции – min_matrix

```
   input  : число r, flag
   output: модифицированная глобальная переменная A
   Data:  глобальные переменные – матрицы H и A
 1 if r = m then
 2 |  Сортировка столбцов матрицы H
 3 |  if flag = True или ρ(H(r)) < ρ(A(r)) then
 4 |  |  A(r) = H(r)
 5 |  end
 6 |  Выход из функции
 7 end
 8 M = [11...1]
 9 k = −1
10 for i = r to m do
11 |  Поменять местами r-ую и i-ую строки матрицы H
12 |  Сортировка столбцов матрицы H
13 |  if ρ(H(r)) = ρ(M) then
14 |  |  k = k + 1
15 |  |  RC[k] = i
16 |  end
17 |  if ρ(H(r)) < ρ(M) then
18 |  |  k = 0
19 |  |  RC[k] = i
20 |  |  M = H(r)
21 |  end
22 |  Поменять местами r-ую и i-ую строки матрицы H
23 end
24 if flag = True или ρ(M) < ρ(A(r)) then
25 |  A(r) = M
26 |  Поменять местами r-ую и RC[0]-ую строки матрицы H
27 |  Сортировка столбцов матрицы H
28 |  Вызов функции core(r + 1, True)
29 |  Поменять местами r-ую и RC[0]-ую строки матрицы H
30 |  for i = 1 to k do
31 |  |  Поменять местами r-ую и RC[i]-ую строки матрицы H
32 |  |  Сортировка столбцов матрицы H
33 |  |  Вызов функции core(r + 1, False)
34 |  |  Поменять местами r-ую и RC[i]-ую строки матрицы H
35 |  end
36 end
37 if flag = False M = A(r) then
38 |  for i = 0 to k do
39 |  |  Поменять местами r-ую и RC[i]-ую строки матрицы H
40 |  |  Сортировка столбцов матрицы H
41 |  |  Вызов функции core(r + 1, False)2
42 |  |  Поменять местами r-ую и RC[i]-ую строки матрицы H
43 |  end
44 end
```

Рис. 2.3 Псевдокод вычислительного ядра алгоритма – core

matrix.c	
Matrix matrix_create(TUint m, TUint n);	Создание матрицы
Vector vector_create(TUint order);	Создание вектора
void matrix_destroy(Matrix matrix);	Удаление матрицы
void vector_destroy(Vector vec);	Удаление вектора
TUint matrix_size(Matrix matrix);	Получить размер матрицы
void matriscopy(Matrix destmat, Matrix srcmat);	Копировать матрицу
bool matrisequal(Matrix mat1, Matrix mat2);	Сравнить матрицы поэлементно
bool rowequal(Matrix mat1, TUint row1, Matrix mat2, TUint row2);	Сравнить строки матриц поэлементно
Matrix matrix_xor(Matrix mat1, Matrix mat2);	Найти хог двух матриц
Matrix matrix_from_file(const char *file_path);	Создать матрицу из файла
equivalence.c	
int find_min_matrix(Matrix H0, uint64_t time_limit);	Обертка над функцией нахождения минимальной матрицы
void *min_matrix(void *arg);	Найти минимальную матрицу
Matrix get_result();	Получить минимальную матрицу
void reset();	Обнулить значения глобальных переменных
uint64_t ro(Matrix matrix);	Получить ρ
void negation_row(Matrix matrix, TUint row);	Инвертировать строку
void negation_column(Matrix matrix, TUint column);	Инвертировать столбец
void core(TUint order, TUint r, bool flag);	Вычислительное ядро основного алгоритма
void sort(Matrix matrix, TUint l, TUint r);	Обертка над функцией сортировки
void column_sort(Matrix matrix);	Сортировка столбцов матрицы
void merge(Matrix matrix, TUint l, TUint m, TUint r);	Объединение двух наборов столбцов
TInt column_comp(Matrix matrix, TUint col1, TUint col2);	Сравнить два столбца
void normalize(Matrix matrix);	Нормализация
void swap_rows(Matrix matrix, TUint row1, TUint row2);	Поменять местами две строки
void swap_columns(Matrix matrix, TUint col1, TUint col2);	Поменять местами два столбца

Рис. 2.4 Реализованная функциональность программы

2.4 Модификация реализации

Для реализации оптимизации создадим мьютекс и условную переменную. Запускаем вычисление минимальной матрицы в отдельном потоке, при выходе из функции, до истечения лимитов на время и количество итераций, меняем значение условной переменной. В запускающей функции проверяем значение условной переменной, чтобы понять, вышли мы из функции по достижению ограничений или потому что алгоритм завершил свою работу. Мьютекс нужен, чтобы блокировать обращения к глобальным переменным, используемым в алгоритме. В нашей реализации отсутствуют повторные запуски нахождения промежуточных матриц. То есть основной алгоритм будет запущен ровно 2 раза – для каждой из исходных матриц. Установленное ограничение по времени – 60 секунд, по количеству итераций – 10000.

```
input  : матрицы H_0 и H_1 размера m × n, число tl
output: булево значение, эквивалентны ли матрицы
Установить лимит времени tl на выполнение потока
A_0 = getMinMatrix(H_0) // Запуск основного алгоритма для матрицы H_0
A_1 = getMinMatrix(H_1) // Запуск основного алгоритма для матрицы H_1
while (tl не исчерпан или A_0 != A_1) и (tl исчерпан или A_0 = A_1) do
    A_0 = getMinMatrix(H_0) // возобновление работы потока
    A_1 = getMinMatrix(H_1) // возобновление работы потока
end
if A_0 = A_1 then
    return True
else
    return False
end
```

Рис. 2.5 Псевдокод модифицированной основной функции

2.5 Демонстрация работы программы

На рисунке можно увидеть запуск программы для тестовых данных, состоящих из двух матриц порядка 8, принадлежащих одному классу эквивалентности. Как и ожидалось, минимальные матрицы совпали, из этого можно сделать вывод, что исходные матрицы эквивалентны. Также мы можем увидеть и найденную минимальную матрицу.

```
→  hadamard-matrix git:(master) ✗ ./equivalence/main
First matrix:
1 1 1 1 1 1 1 1
1 0 1 0 1 0 1 0
1 1 0 0 1 1 0 0
1 0 0 1 1 0 0 1
1 1 1 1 0 0 0 0
1 0 1 0 0 1 0 1
1 1 0 0 0 0 1 1
1 0 0 1 0 1 1 0
Min first matrix:
0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1
0 0 1 1 0 0 1 1
0 0 1 1 1 1 0 0
0 1 0 1 0 1 0 1
0 1 0 1 1 0 1 0
0 1 1 0 0 1 1 0
0 1 1 0 1 0 0 1
Second matrix:
1 1 1 1 1 1 1 1
1 1 0 0 1 1 0 0
1 0 1 0 1 0 1 0
1 0 0 1 1 0 0 1
1 1 1 1 0 0 0 0
1 0 1 0 0 1 0 1
1 1 0 0 0 0 1 1
1 0 0 1 0 1 1 0
Min second matrix:
0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1
0 0 1 1 0 0 1 1
0 0 1 1 1 1 0 0
0 1 0 1 0 1 0 1
0 1 0 1 1 0 1 0
0 1 1 0 0 1 1 0
0 1 1 0 1 0 0 1
Min matrices are equal.
```

Рис. 2.6 Демонстрация результата работы программы

2.6 Тест производительности

Тест производительности представляет собой следующее: мы берем заранее заготовленные матрицы порядка 2, 4, 8, 12, 16, 20, 24, 28, для которых известно эквивалентны они или нет, то есть три матрицы каждого порядка, две из которых эквивалентны друг другу, а третья неэквивалентна им, и запускаем на них наивный алгоритм и эффективный алгоритм.

Для наивного алгоритма мы ограничились матрицами порядка 12, так как вычисление даже такого небольшого порядка заняло значительное количество

времени. Как видно из графика, наивный алгоритм требует огромных временных и вычислительных затрат.

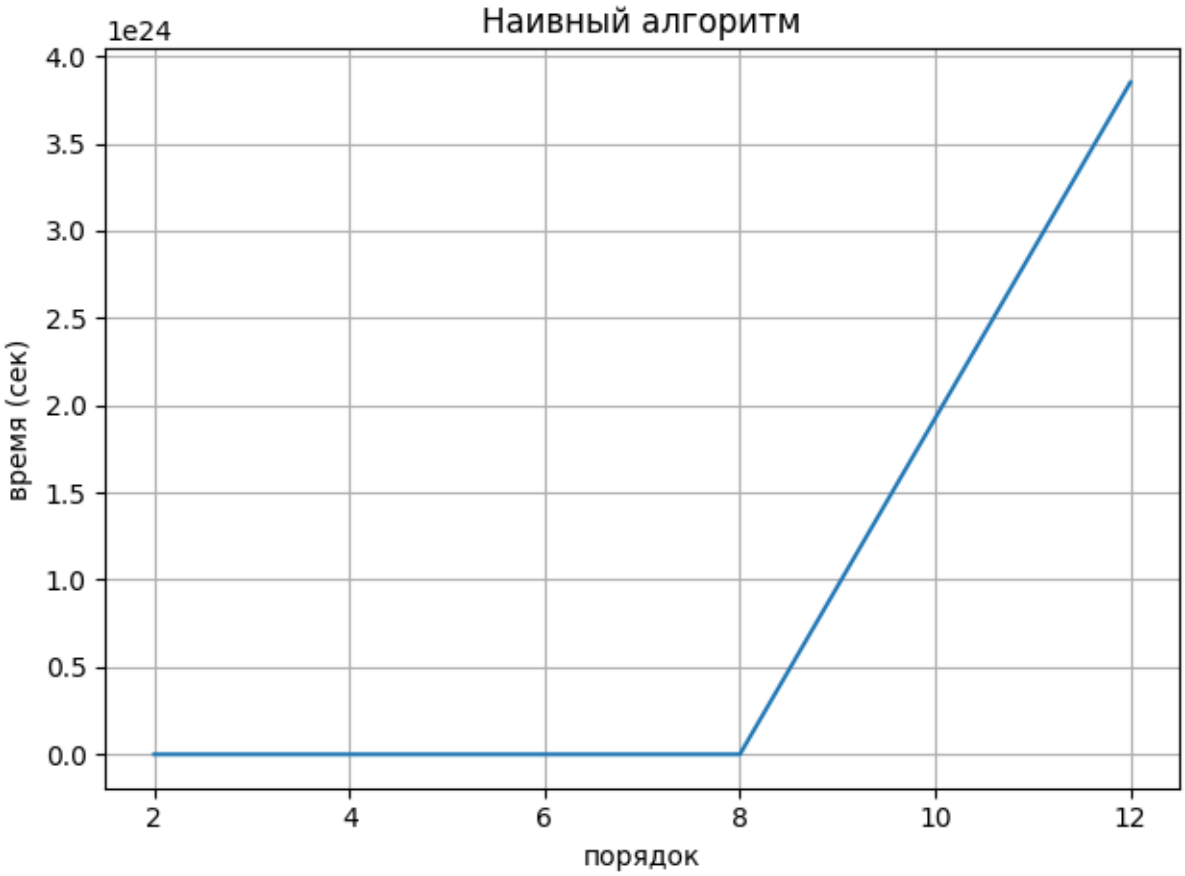

Рис. 2.7 График производительности наивного алгоритма

Далее приведены два графика работы эффективного алгоритма для эквивалентной и неэквивалентной матрицы. Можно заметить, что асимптотика для одинаковых и разных классов не отличается. Также стоит отметить, что полученные на практике результаты плохо соотносятся с заявленной теоретической вычислительной сложностью для матрицы порядка до 32. В результате тестирования мы получили более оптимистичные оценки производительности.

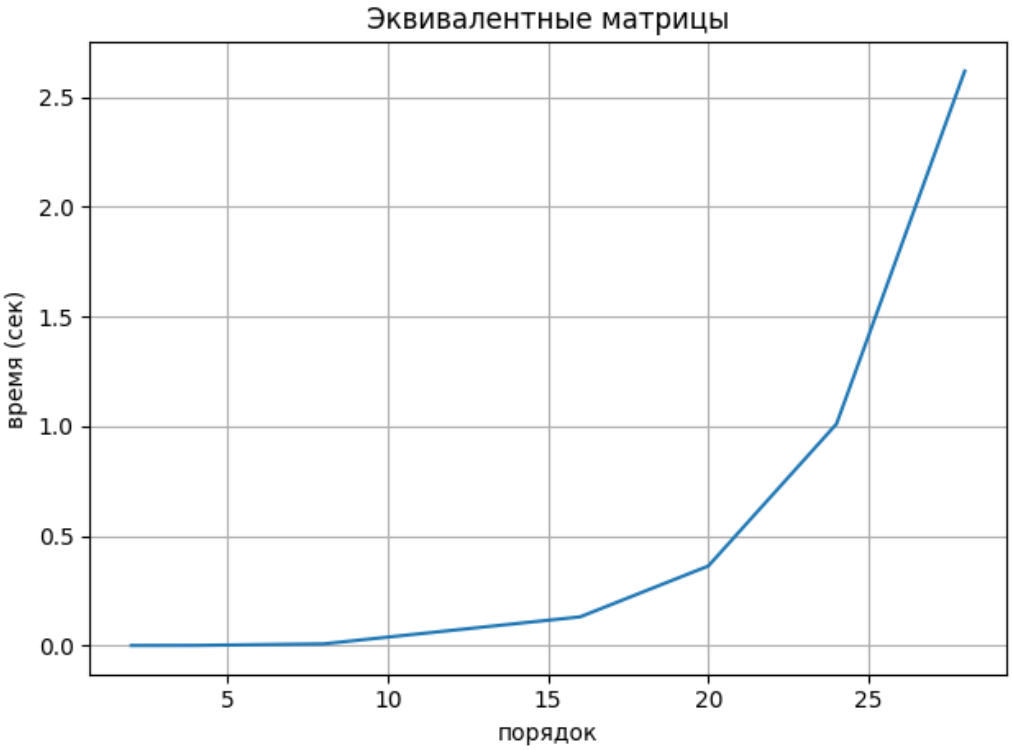

Рис. 2.8 График производительности эффективного алгоритма для эквивалентных матриц

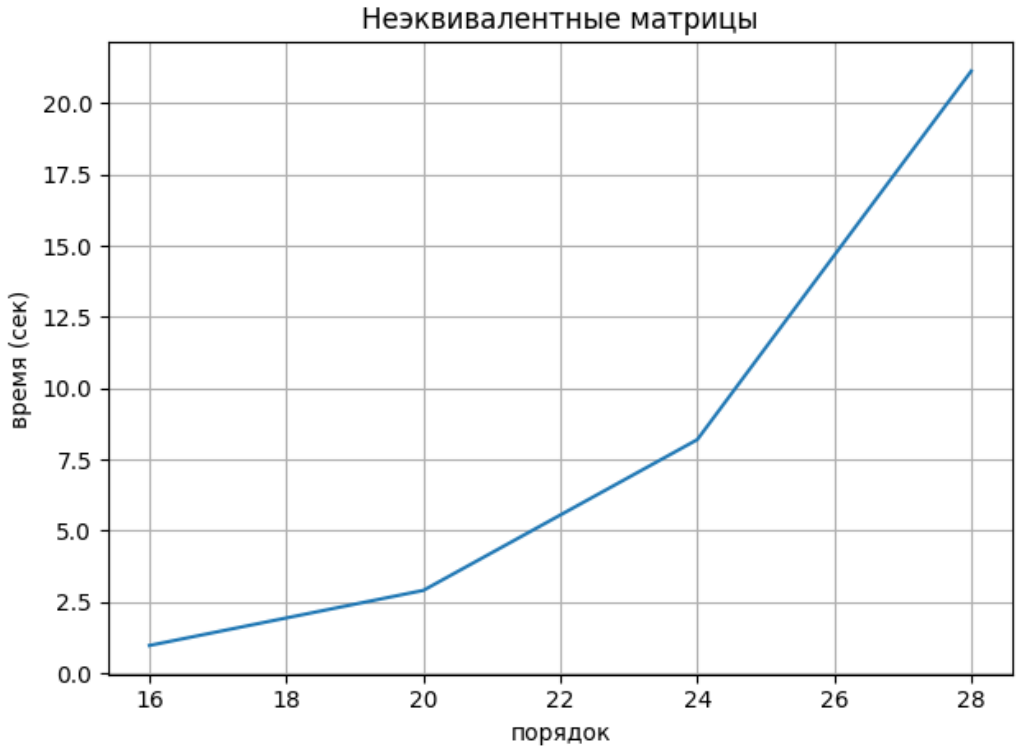

Рис. 2.9 График производительности эффективного алгоритма для неэквивалентных матриц

Также был выполнен один тест для матрицы порядка 32 двумя версиями алгоритма – без оптимизации с оптимизацией. Без оптимизации алгоритм проработал достаточно длительное время, и минимальная матрица так и не была подсчитана, поэтому этот тест не отображен на соответствующем графике. С оптимизацией подсчет минимальной матрицы занял несколько секунд, достигнув ограничения на количество итераций.

ЗАКЛЮЧЕНИЕ

В выпускной квалификационной работе выполнены все поставленные задачи в полном объеме.

- Реализован наивный и эффективный алгоритмы проверки эквивалентности двух матриц Адамара
- Оптимизирован эффективный алгоритм для матриц большого порядка
- Проведено сравнение производительности реализованных алгоритмов и анализ полученных результатов

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. K. -H. Park and H. -Y. Song, «Hadamard Equivalence of Bimary Matrices», 2009
2. W. P. Orrick, «Switching Operations For Hadamard Matrices», 2007
3. K. J. Horadam, «Hadamard Matrices and Their Applications», 2007
4. N. J. A. Sloane, «A Library of Hadamard Matrices» [Электронный ресурс]. - Режим доступа: URL: http://neilsloane.com/hadamard/ (21.03.2020)
5. The on-line encyclopedia of integer sequences [Электронный ресурс]. - Режим доступа: URL: https://oeis.org/A007299 (21.03.2020)