Содержание

[Введение 2](#_Toc485248676)

[1 Обзор научной литературы, связанной с проблематикой работы 3](#_Toc485248677)

[1.1 Основные определения 3](#_Toc485248678)

[1.2 Теория представлений симметрических групп 4](#_Toc485248679)

[1.3 YJM-элементы 5](#_Toc485248680)

[1.4 Симметрические многочлены от YJM-элементов 7](#_Toc485248681)

[2 Актуальность темы выпускной квалификационной работы. Постановка задачи 8](#_Toc485248682)

[3 Разработка программного обеспечения 9](#_Toc485248683)

[3.1 Обоснование выбора средств реализации 9](#_Toc485248684)

[3.2 Описание функций и схемы классов 9](#_Toc485248685)

[3.3 Описание алгоритма работы программы 18](#_Toc485248686)

[3.4 Организация параллельных вычислений 20](#_Toc485248687)

[3.5 Анализ полученных результатов 21](#_Toc485248688)

[Заключение 24](#_Toc485248689)

Введение

Теория представлений конечных групп — технически довольно сложный раздел алгебры, зачастую требующий для полного понимания трудоёмких вычислений.

Как ни странно, эти вычисления, как правило, производятся вручную или вообще обсуждаются косвенным образом. Исследователи предпочитают "проявить здравый смысл" и доказывать требуемые свойства, опираясь не на прямые вычисления, а на взаимодействие свойств формул и конструкций. Это усложняет и без того непростые тексты работ, поэтому некоторые из них славятся своей нечитаемостью.

Данная работа является попыткой применить компьютер к явному вычислению одного простого, но важного семейства формул. Кроме практического интереса — дополнительно убедиться в том, что теорема, доказанная при помощи (относительно) косвенных соображений всё-таки верна (хотя бы для небольших представлений), работа также представляет теоретически-эвристический интерес: необходимо понять, почему компьютеры так мало применяются в исследованиях теории представлений и существуют ли каких-нибудь препятствий принципиального вида для их применения.

1 Обзор научной литературы, связанной с проблематикой работы

1.1 Основные определения

Рассмотрим основные определения, используемые в данной работе.

Перестановкой множества называется биекция множества на себя. В этом случае называется порядком перестановки. В теории групп под перестановкой произвольного множества подразумевается биекция этого множества на себя. Тривиальная перестановка – это перестановка, которая отображает каждый элемент множества в самого себя.

Пусть – перестановка конечного множества . Для каждого рассмотрим последовательность . Так как – биекция и конечно рано или поздно в этой последовательности появится . Таким образом, для некоторого единственного имеем, что и все элементы различны. Тогда последовательность называется циклом длины . Цикл длины 2 называется транспозицией.

Группой называется множество, с определенной на нем бинарной операцией, обладающей свойством ассоциативности, причем для этой операции имеется нейтральный элемент, а также для каждого элемента существует обратный .

Симметрический многочлен – многочлен от переменных, который не изменяется при всех перестановках входящих в него переменных. Элементарные симметрические многочлены – это симметрические многочлены вида

|  |  |
| --- | --- |
|  | ((1) |

Алгеброй над полем называется кольцо с единицей, являющееся одновременно векторным пространством над полем .

Рассмотрим всевозможные формальные суммы

|  |  |
| --- | --- |
|  | ((2) |

Две суммы считаются равными тогда и только тогда, когда у них совпадают коэффициенты для всех (формальную сумму можно рассматривать как функцию на группе со значениями в поле , причем коэффициент дает значение этой функции на элементе ). Определим операции над формальными суммами следующим образом

|  |  |
| --- | --- |
|  | ((3) |
|  | ((4) |
|  | ((5) |

Можно проверить, что относительно определенных операций множество формальных сумм образует алгебру , которая называется групповой алгеброй группы над полем .

1.2 Теория представлений симметрических групп

Теория представлений симметрических групп является, по-видимому, одним из старейших приложений аппарата общей теории представлений конечных групп и ассоциативных алгебр. Она восходит к работам А. Юнга (см. напр. [3]) и характерна большой сложностью рассматриваемых конструкций. Например, Г. Джеймс, автор прекрасного изложения этой теории [4], пишет, что работы Юнга очень трудночитаемы. Сложность рассматриваемых конструкций была серьёзным препятствием к разработке самой теории и её обобщений на серии групп, близких к симметрическим (или иначе группам Кокстера серии А): на группы Кокстера серий ВС, D и некоторые другие похожие серии групп и алгебр.

Ситуация существенно изменилась в начале 90-х годов XX века, когда А. М. Вершику удалось в соавторстве с А. Ю. Окуньковым рассмотреть групповые алгебры симметрических групп как локальные стационарные алгебры (см. [5]). Теория оказалась достаточно проста и элегантна, поэтому в работе [6] последовало ее обобщение на сплетения симметрических групп с произвольными конечными группами, а в [7] – на сплетения симметрических групп с произвольными конечномерными полупростыми алгебрами

1.3 YJM-элементы

Рассмотрим возрастающее семейство конечных групп . Эти группы имеют семейство образующих , так, что

|  |  |
| --- | --- |
|  | ((6) |

Пусть – такой элемент группы , что никакой элемент, сопряжённый с ним в группе не содержится ни в какой группе с меньшим номером. Символом (при ) обозначим сумму всех элементов в групповой алгебре группы , которые сопряжёны с и лежат в . Эти суммы будем называть элементами Юнга-Юциса-Мерфи или YJM-элементами .

Заметим, что элемент является разностью центрального элемента групповой алгебры группы и центрального элемента групповой алгебры предыдущей группы. Пусть элементы образуют полную группу представителей классов сопряжённости в группе элементов этой группы, сопряжённых в группе с элементом : . Эти элементы, даже с разными номерами и соответствующие разным элементам, коммутируют между собой . Действительно:

|  |  |
| --- | --- |
|  | ((7) |

В каждом произведении какой-нибудь номер не меньше другого, так что соответствующий элемент лежит в центре соответствующей групповой алгебры и коммутирует с другим сомножителем.

Следовательно, всевозможные элементы вида попарно коммутируют между собой и, тем самым, порождают последовательность коммутативных подалгебр групповых алгебр серий групп. Фактически, теория основана на том факте, что эта подалгебра оказывается максимальной коммутативной подалгеброй групповой алгебры и позволяет построить базис пространства групповой алгебры, который состоит из общих собственных векторов всей подалгебры. Этот базис называется (как и сама подалгебра) базисом (подалгеброй) Гельфанда-Цетлина .

В основном примере групповых алгебр симметрических групп – кокстеровские образующие симметрической группы. Тогда

|  |  |
| --- | --- |
|  | ((8) |

классические элементы Юнга-Юциса-Мерфи .

В остальных случаях применения рассматриваемого метода соответствующие формулы становятся, как правило, немного сложнее.

1.4 Симметрические многочлены от YJM-элементов

Рассмотрим последовательность коммутативных алгебр симметрических многочленов с целыми коэффициентами от формальных переменных . Подстановка в переменные элементов (8) индуцирует гомоморфизм алгебры в центр групповой алгебры -ой симметрической группы . Отсюда можно сформулировать теорему, которая описана и доказана в .

Теорема. Симметрические многочлены от YJM-элементов порождают центр и выражаются как линейные комбинации сумм классов сопряжённости, совпадающих с суммами перестановок одного циклического типа.

2 Актуальность темы выпускной квалификационной работы. Постановка задачи

Тема выпускной квалификационной работы актуальна, так как рассмотрение даже простейшей усложнённой ситуации (группы серии ВС) на этом этапе приводит уже к рассмотрению многочленов от двух семейств переменных, симметрических отдельно по семействам – то есть к многократному усложнению формальной стороны и конкретных вычислений. Это делает «ручные» вычисления малоэффективными, а использование компьютера ещё более необходимым.

Для изучения симметрических многочленов от YJM-элементов необходимо разработать программное обеспечение, которое будет включать в себя следующие функции:

* проведение вычислений в групповой алгебре;
* вычисление образа конкретного симметрического многочлена под действием рассматриваемого гомоморфизма;
* проведение «обратной процедуры» построения по конкретному стандартному элементу центра одного из многочленов прообраза.

Также необходимо детально изучить гомоморфизм из множества симметрических многочленов в центр групповой алгебры с выяснением следующих фактов: какой именно симметрический многочлен является прообразом некоторого стандартного элемента центра, есть ли у этого гомоморфизма ядро и как оно устроено и т.д.

3 Разработка программного обеспечения

3.1 Обоснование выбора средств реализации

Для реализации приложения был выбран язык C#. Такой выбор языка обусловлен несколькими причинами:

* данный язык является объектно-ориентированным, что позволяет удобнее работать с данными и организовывать их в структуры [11];
* наличие специального языка запросов LINQ, который значительно упрощает работу с большими объемами данных [12];
* встроенные реализации списков, словарей объектов, поддерживающие поиск, выполнение сортировки, выборку данных и другие операции ;
* кроссплатформенность языка обеспечивает возможность разработки и использования программы независимо от окружения – операционной системы, IDE, компилятора ;
* присутствие широкого спектра встроенных средств разработки параллельных программ, таких как параллельные циклы (Parallel.For и Parallel.ForEach), класс Task, представляющий асинхронную операцию, класс Thread, представляющий реализацию потока, средства синхронизации, например классы Mutex и Semaphore .

3.2 Описание функций и схемы классов

Схемы алгоритмов основных реализованных функций приведены в приложении А, а исходные коды – в приложении Б.

Для выполнения поставленной задачи было разработано несколько классов: основные, реализующие функции по вычислениям в групповой алгебре, а также по работе с симметрическими многочленами и YJM-элементами, и вспомогательные, которые необходимы для работы основных.

Сначала рассмотрим вспомогательные классы. Для вычисления факториала числа используется статический класс Factorial. Он реализует метод public static int Get (int), который принимает в качестве параметра число, факториал которого необходимо вычислить. Также в классе есть поле private static readonly List<int> factorials, которое используется для хранения уже вычисленных факториалов. Схема класса представлена на рисунке 1.

Статический класс NumberSplits используется для генерации разбиений числа на слагаемые. Схема класса приведена на рисунке 2.

Класс реализует следующие методы:

* public static List<List<int>> GenerateSplits(int) - возвращает список всех разбиений числа в лексикографическом порядке;
* public static bool CompareSplits(List<int> l1, List<int> l2) служит для поэлементного сравнения двух разложений дерева. Если два разложения не отличаются, то он возвращает true, иначе false.

Класс Combination необходим для генерации неупорядоченных выборок множества. Поля класса:

* private List<int> setList – хранит исходное множество;
* private List<List<int>> сombinations – содержит все уже полученные выборки;
* private int setLength – хранит количество элементов множества;
* private int combinatonLength – хранит количество элементов в выборке.

В классе есть следующие свойства:

* public List<int> CurrentCombination { get; } – используется для доступа к текущей выборке;
* public List<List<int>> Combinations { get; } – возвращает все уже сгенерированные выборки;
* public int CurrentNumber { get; } – возвращает номер текущей выборки по порядку;
* public string Text { get; } – возвращает текстовое представление выборки.

Класс реализует конструктор public Combination(int ,int): в качестве первого параметра передается количество элементов множества, в качестве второго параметра – количество элементов, которое должно оказаться в выборке. Схема класса приведена на рисунке 3.

Класс реализует следующие методы:

* public bool GetNextCombination() – генерирует следующую выборку и возвращает true, если это удалось сделать, а иначе возвращает false;
* public void Print() – печатает выборку.

Далее рассмотрим основные классы. Класс Cycle реализует цикл перестановки и содержит поле private int[] cycle, которое используется для хранения элементов цикла. Схема класса приведена на рисунке 4.

Класс реализует несколько конструкторов:

* public Cycle () – конструктор по умолчанию, создает пустой цикл длины 0;
* public Cycle (params int[]) – принимает переменное число параметров в виде массива целых чисел, инициализирует цикл полученным массивом;
* public Cycle (Cycle) – конструктор копирования, инициализирует цикл элементами переданного в качестве параметра цикла.

Класс содержит следующие свойства:

* public int Length { get; } – возвращает длину цикла;
* public int[] Elements { get; } – возвращает массив элементов цикла;
* public int First { get; } – возвращает первый элемент цикла;
* public string Text { get; } – возвращает текстовое представление цикла в стандартной записи – цикл начинается со своего максимального элемента, например (4, 1, 2, 3).

Методы класса:

* private void InitCycle(int[]) - выполняет функцию инициализации цикла и принимает в качестве параметра целочисленный массив;
* public int Apply(int) – реализует функцию применения цикла к числу, которое передается в качестве параметра – в цикле ищется число, равное переданному параметру. Если такое число находится, то возвращается следующее число из цикла, иначе метод возвращает значение параметра;
* public bool Contains(int) – метод возвращает true, если цикл содержит число, иначе возвращает false;
* public void Print() – печатает цикл;
* public override bool Equals(object) – метод, сравнивающий два экземпляра класса Cycle и возвращающий true, если объекты равны;
* public override int GetHashCode() – возвращает хэш текущего экземпляра объекта;
* public static bool operator ==(Cycle, Cycle) – оператор равенства, который производит сравнение двух экземпляров и возвращает true, если экземпляры равны; сравнение циклов производится поэлементно;
* public static bool operator !=(Cycle, Cycle) – оператор неравенства, который производит сравнение двух экземпляров и возвращает true, если экземпляры неравны.

Следующий класс – Permutation, который реализует перестановку множества. Схема класса представлена на рисунке 5.

Поля класса:

* private static Comparison<Cycle> PermutationComparer – компаратор, который применяется при сортировке циклов перестановки;
* private int order – хранит порядок перестановки;
* private List<Cycle> cycles – используется для хранения циклов перестановки.

В классе есть следующие свойства:

* public List<Cycle> NotTrivialCycles { get; } – возвращает только нетривиальные циклы перестановки, т.е. циклы, длина которых больше 1;
* public int Order { get; } – возвращает порядок перестановки;
* public string Text { get; } – возвращает текстовое представление перестановки в стандартной циклической записи – каждый цикл начинается со своего максимального элемента, циклы отсортированы по возрастанию первых элементов, например (4, 1, 2, 3)(7,5)(10,9).

Класс реализует следующие конструкторы:

* public Permutation (int) – принимает в качестве параметра порядок перестановки и создает пустую перестановку;
* public Permutation (int, IEnumerable<Cycle>) – принимает в качестве параметров порядок перестановки и коллекцию циклов и инициализирует ими перестановку;
* public Permutation (int, params Cycle[]) – принимает в качестве параметров порядок перестановки и переменное число циклов, которыми инициализирует перестановку;
* public Permutation(int, params int[][]) - принимает в качестве параметров порядок перестановки и переменное число целочисленных массивов, представляющих собой циклы, которыми инициализирует перестановку.

Методы класса:

* private void InitPermuatation(IEnumerable<Cycle>) – инициализирует перестановку коллекцией циклов;
* private void Normalize() – преобразует перестановку к стандартной циклической записи;
* private void AddCycle(Cycle, bool), private void AddCycle(int[], bool) – добавляет цикл в перестановку, второй параметр показывает, нужно ли вызывать нормализацию после добавления цикла;
* public int Apply(int) – применяет перестановку к числу, переданному в качестве параметра;
* public void SetOrder(int) – устанавливает порядок перестановки и добавляет недостающие тривиальные циклы;
* public void Print() – печатает перестановку;
* public static bool Compare(Permutation, Permutation) – сравнивает перестановки с точки зрения циклической структуры – возвращает true, если перестановки содержат одинаковое количество циклов и наборы длин циклов в обеих перестановках совпадают;
* public static int GetPermutationsCount(Permutation) – возвращает количество перестановок циклического типа перестановки, переданной в качестве параметра;
* public override bool Equals(object) – метод, сравнивающий два экземпляра класса Permutation и возвращающий true, если объекты равны;
* public static Permutation operator \*(Permutation, Permutation) – оператор умножения перестановок;
* public static bool operator ==(Permutation, Permutation) – оператор равенства, возвращает true, если перестановки равны; сравнение производится посредством сравнения циклов перестановок;
* public static bool operator !=(Permutation, Permutation) – оператор неравенства, возвращает true, если перестановки неравны.

Следующий класс ElementarySymmetricPolynomial, который описывает элементарный симметрический многочлен. Схема класса представлена на рисунке 6.

Поля класса:

* private int variablesCount – хранит количество переменных многочлена;
* private List<List<int>> terms – хранит наборы чисел, описывающие слагаемые многочлена: каждое слагаемое описывается номерами входящих в него переменных.

Класс содержит свойство public string Text { get; }, которое возвращает текстовое представление элементарного симметрического многочлена.

Класс реализует конструктор public ElementarySymmetricPolynomial (int, int), который в качестве параметров принимает количество переменных и номер элементарного симметрического многочлена.

Методы класса:

* private void GenerateElementarySymmetricPolynomial(int) – генерирует элементарный симметрический многочлен с заданным номером.
* public void Print() – печатает многочлен;
* public PermutationDictionary Substitution(List<YJMElement>) – осуществляет подстановку аргументов, переданных в качестве параметра, в многочлен.

Класс YJMElement описывает YJM-элемент. Схема класса приведена на рисунке 7.

Поля класса:

* private int order – хранит порядок входящих в YJM-элемент перестановок;
* private List<Permutation> permutations – хранит список перестановок, входящих в YJM-элемент;

Свойства класса:

* public int Order { get; } – возвращает порядок перестановок, входящих в YJM-элемент;
* public  List<Permutation> Permutations – возвращает список входящих в YJM-элемент перестановок;
* public string Text { get; } - возвращает текстовое представление YJM-элемента.

Класс реализует два конструктора:

* public YJMElement(int) – инициализирует стандартный классический YJM-элемент по формуле (8), в качестве параметра передается;
* public YJMElement(List<Permutation>) – инициализирует YJM-элемент списком перестановок.

Методы класса:

* public void Print() – печатает YJM-элемент;
* public void SetOrder(int) – устанавливает порядок входящих в YJM-элемент перестановок;
* public static List<YJMElement> Generate(int) – генерирует набор YJM-элементов по формуле (8);
* public static void Print(List<YJMElement>) – печатает набор YJM-элементов, переданный в качестве параметра;
* public override bool Equals(object) – метод, сравнивающий два экземпляра класса YJMElement и возвращающий true, если объекты равны;
* public override int GetHashCode() – возвращает хэш текущего экземпляра объекта;
* public static YJMElement operator \*(YJMElement, YJMElement) – оператор умножения, перемножает между собой два YJM-элемента;
* public static bool operator ==(YJMElement, YJMElement) – оператор равенства, возвращает true, если YJM-элементы равны; сравнение производится посредством сравнения перестановок;
* public static bool operator !=(YJMElement, YJMElement) – оператор неравенства, возвращает true, если YJM-элементы неравны.

Следующий класс – public class PermutationDictionary, который наследуется от класса Dictionary<Permutation, int>. Схема класса представлена на рисунке 8.

В качестве ключа словаря используется перестановка, а в качестве значения – коэффициент количества вхождений данной перестановки. Он используется для хранения результатов подстановки YJM-элементов в элементарный симметрический многочлен, а также результатов перемножения объектов PermutationDictionary. Класс содержит поле public List<int> Split, которое хранит разбиение числа, соответствующее комбинации симметрических многочленов, использовавшихся для получения данного экземпляра класса. Свойство класса public string Text { get; } возвращает текстовое представление текущего объекта.

Методы класса:

* public new void Add(Permutation, int) – добавляет перестановку в словарь с соответствующим коэффициентом, если такой перестановки еще нет в словаре, иначе добавляет к соответствующему перестановке коэффициенту это число;
* public void Add(IEnumerable<Permutation>) – добавляет несколько перестановок в словарь с единицами в качестве коэффициентов;
* public void Print(Output output = Output.Console, string path = "", string text = "") – печатает текстовое представление словаря в консоль или в файл по переданному в качестве второго параметра пути. В качестве третьего параметра передается дополнительный текст, который нужно вывести;
* public void SimplyPrint(Output output = Output.Console, string path = "", string addText = "") – печатает сгруппированные по циклическим типам перестановки, входящие в словарь
* public statiс PermutationDictionary Copy(PermutationDictionary) – создает копию переданного в качестве параметра экземпляра класса;
* public void ClearEmptyEntries() – удаляет из словаря тривиальные перестановки;
* public void SetOrder(int) – устанавливает порядок перестановкам, входящим в словарь;
* public int GetMaxOrder() – возвращает максимальный порядок среди входящих в словарь перестановок;
* public static PermutationDictionary operator \*(PermutationDictionary, PermutationDictionary) – перемножает два объекта класса.

3.3 Описание алгоритма работы программы

Первый этап работы программы – это задание исходных параметров, а именно количества элементов групповой алгебры (или количества переменных элементарных симметрических многочленов), а также максимальной степени многочлена . Исходя из этих параметров, генерируются первые элементарных симметрических многочленов и YJM-элементов. Многочлены генерируются следующим образом: для каждого из чисел от 1 до генерируются неупорядоченные выборки соответствующего количества элементов из множества чисел . Эти выборки определяют слагаемые симметрического многочлена, -е слагаемое представляет собой произведение переменных с номерами из выборки . YJM-элементы генерируются по формуле (8) для каждого из чисел от 2 до включительно.

После генерации производится подстановка YJM-элементов вместо переменных с соответствующими номерами в элементарные симметрические многочлены, перемножение и приведение подобных. В результате получается сумма перестановок с некоторыми целыми коэффициентами.

Далее для каждого из чисел от 2 до генерируются всевозможные разбиения этого числа. Алгоритм генерации всех разбиений в обратном лексикографическом порядке описан в [17] и проиллюстрирован на рисунке 9.

Каждому из разбиений соответствует некоторое произведение элементарных симметрических многочленов. Множителями произведения являются многочлены с номерами, соответствующими элементам разбиения, например, разбиению числа 4 соответствует произведение . После перемножения элементарных многочленов и приведения подобных получаем набор сумм перестановок с целыми коэффициентами, каждая из которых соответствует какому-либо произведению элементарных симметрических многочленов.

Далее слагаемые в этих суммах группируются по циклическому типу – в сумме остается только одна перестановка каждого циклического типа с коэффициентом, равным сумме коэффициентов перед всеми перестановками этого циклического типа. После этого необходимо, в некотором смысле, «нормировать» полученные выражения. Для этого необходимо разделить коэффициенты перед каждой из перестановок на количество перестановок этого циклического типа. В приведена следующая формула для вычисления количества перестановок данного циклического типа:

|  |  |
| --- | --- |
|  | ((11) |

где – количество элементов множества,

– количество циклов длины в перестановке.

Полученные выражения приравняем к соответствующим им произведениям элементарных симметрических многочленов. В результате получаем системы уравнений следующего вида:

|  |  |
| --- | --- |
|  | ((12) |

где – перестановка циклического типа (2,1),

– перестановка циклического типа (3,2,1),

– перестановка циклического типа (3,2)(4,1),

– -ый элементарный симметрический многочлен.

Система (12) приведена для случая . Иными словами, полученные системы позволяют выразить элементы центра групповой алгебры через элементарные симметрические многочлены и их комбинации.

Для решения полученных систем была использована система компьютерной математики Maple 2017, которая позволяет выполнять символьные вычисления. Для решения использовалась функция solve, которая позволяет решать системы уравнений в символьном виде. Исходные коды скриптов для решения систем представлены в приложении Б.

3.4 Организация параллельных вычислений

Для организации параллельных вычислений использовалась встроенная в C# реализация параллельного цикла, а именно Parallel.ForEach.

Parallel.ForEach – статический метод класса Parallel, который позволяет параллельно выполнять итерации цикла foreach над коллекцией объектов, реализующей интерфейс IEnumerable, а также контролировать и управлять состоянием цикла. В качестве параметров он принимает коллекцию, по которой должен выполняться цикл, и делегат Action, который представляет реализацию одной итерации цикла. Для управления состоянием цикла делегату также может передаваться экземпляр класса ParallelLoopState, который используется, например, для преждевременной остановки цикла.

Большую часть времени работы программа тратит на перемножение элементарных симметрических многочленов, поэтому именно эту часть кода стоит распараллеливать. Здесь есть два основных цикла – первый пробегает по числам от 2 до и генерирует их разбиения, а второй, который вложен в первый, пробегает по разбиениям числа и вычисляет соответствующие этим разбиениям произведения многочленов. Реализуем два параллельных варианта работы алгоритма, с использованием Parallel.ForEach вместо первого цикла и второго и сравним их между собой, а также с последовательным алгоритмом. Полученные результаты приведены в следующем пункте.

3.5 Анализ полученных результатов

Первым из результатов работы программы является построение систем уравнений, которые позволяют определить параметры отображения множества симметрических многочленов от YJM-элементов в центр групповой алгебры. Первая такая система приведена в формуле (12). Вторая система для случая выглядит следующим образом:

|  |  |
| --- | --- |
|  | ((13) |

где – перестановка циклического типа (2,1),

– перестановка циклического типа (3,2,1),

– перестановка циклического типа (3,2)(4,1),

– перестановка циклического типа (4,3,2,1),

– перестановка циклического типа (3,2,1)(5,4),

– перестановка циклического типа (2,1)(4,3)(6,5),

– -ый элементарный симметрический многочлен.

Остальные построенные системы приведены в приложении В.

Определители матриц систем образуют следующую последовательность.

Так как все определители вычисленных матриц не равны 0, значит, решение системы существует и единственное. Следовательно, у рассматриваемого отображения нет ядра, по крайней мере, для рассмотренных групповых алгебр. Решения систем (11) и (12) можно выписать в следующем виде:

|  |  |
| --- | --- |
|  | ((14) |
|  | ((15) |

Решения остальных систем также приведены в приложении В.

Для сравнения быстродействия последовательной и параллельных версий алгоритма использовался ноутбук со следующими характеристиками:

* процессор Intel Core i5-6200U CPU @ 2.86GHz 2 ядра 4 потоков;
* оперативная память 6 Гб 1333Mhz;
* среда исполнения Microsoft .NET 4.0.30319.42000;
* среда разработки Xamarin Studio Community v.6.3 (build 863).

На каждом из тестов программа запускалась по 5 раз, в качестве результата было взято среднее время по этим запускам, чтобы минимизировать влияние внешних факторов.

Из таблицы сравнения быстродействия видно, что распараллеливать внешний цикл не имеет смысла. При малых размерах групповой алгебры этот вариант работает даже медленнее, чем последовательный (что можно объяснить достаточно большими накладными расходами на распараллеливание), а при больших показывает сравнимое быстродействие.

Распараллеливание внутреннего цикла на больших степенях многочленов дает практически двукратный прирост в быстродействии при очень малых затратах на реализацию параллельного алгоритма. Даже учитывая плохую приспособленность языка к разработке высокопроизводительных программ для решения математических проблем ввиду его объектно-ориентированной природы, это говорит о высокой эффективности встроенных в C# методов распараллеливания кода.

Заключение

В результате выполнения выпускной квалификационной работы были получены следующие результаты:

* разработана программа для изучения симметрических многочленов от YJM-элементов;
* построены и решены системы уравнений, определяющие параметры отображения из множества симметрических многочленов в центр групповой алгебры;
* для рассмотренных групповых алгебр показано, что у этого отображения нет ядра;
* стандартные элементы центра выражены через симметрические многочлены, тем самым подтверждена справедливость теоремы о том, что элементы центра являются симметрическими многочленами от YJM-элементов.

Работа показала, что единственным осложнением для изучения является не очень хорошая приспособленность языков программирования для решения алгебраических задач такой сложности. Просто описываемые и интуитивно хорошо понятные алгебраические структуры, как ни странно, оказываются на практике слишком гибкими и изменчивыми для описания при помощи стандартных алгоритмических структур данных.

Дальнейшим развитием темы данной работы может послужить совершенствование алгоритмов работы, так как с увеличением количества элементов в групповой алгебре количество слагаемых в симметрических многочленах растет очень быстро, а значит, увеличивается и количество необходимой памяти, и время, необходимое на выполнение вычислений.