МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ И ФИЗИКО-МАТЕМАТИЧЕСКИХ НАУК
КАФЕДРА ФУНДАМЕНТАЛЬНОЙ ИНФОРМАТИКИ И ПРИКЛАДНОЙ МАТЕМАТИКИ

Допускаю к защите
Заведующий кафедрой ФИиПМ
_____/ Котельников Е.В.
(подпись) (Ф.И.О.)

# *ВЫРАЖЕНИЕ ЦЕНТРАЛЬНЫХ ИДЕМПОТЕНТОВ НЕПРИВОДИМЫХ ПРЕДСТАВЛЕНИЙ ГРУПП КОКСТЕРА ЧЕРЕЗ YJM-ЭЛЕМЕНТЫ*

Пояснительная записка выпускной квалификационной работы

Разработал студент гр. ПМИм-2301-01-00 _____/ Стерлягов А.А. / _____

Руководитель к.ф.-м.н., доцент каф. ФИиПМ_____/ Пушкарев И.А. / _____

Киров 2017

**Содержание**

# Введение

# 1 Обзор научной литературы, связанной с проблематикой работы

## 1.1 Основные определения

Рассмотрим основные определения, используемые в данной работе.

Перестановкой множества $M$ называется биекция множества $\{1,2,\dots,n\}$ на себя. $n$ 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 себя. Циклом называется перестановка, которая тривиальна на всем множестве, кроме подмножества $\{m_1, m_2, \dots, m_k\}$. $k$ в таком случае называется длиной цикла. Транспозицией называется цикл длины 2.

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 [1].

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

$$\sigma_k(x_1, x_2, \dots, x_n) = \sum_{1 \le j_1 \le j_2 \le \dots \le j_k \le n} x_{j_1} \dots x_{j_k} \qquad (1)$$

Алгеброй над полем $K$ называется кольцо $A$ с единицей, являющееся одновременно векторным пространством над полем $K$ [2].

Рассмотрим всевозможные формальные суммы

$$\sum a_g \cdot g, \qquad a_g \in K \tag{2}$$

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

$$\sum a_g \cdot g + \sum b_g \cdot g = \sum (a_g + b_g) \cdot g = \tag{3}$$

$$\left(\sum_{g \in G} a_g \cdot g\right)\left(\sum_{h \in G} b_h \cdot h\right) = \sum_{g,h \in G} a_g b_h gh = \sum_{f \in G} c_f \cdot f \tag{4}$$

$$t\left(\sum_{g \in G} a_g \cdot g\right) = \sum_{g \in G} t a_g g\,, \quad t \in K \tag{5}$$

Можно проверить, что относительно определенных операций множество формальных сумм образует алгебру $K[G]$, которая называется групповой алгеброй группы $G$ над полем $K$ [2].

### 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 A): на группы Кокстера серий BC, 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 [5]. Теория оказалась достаточно проста и элегантна, поэтому последовало на сплетения симметрических групп с произвольными конечными группами [6], а также на сплетения симметрических групп с произвольными конечномерными полупростыми алгебрами [7].

### 1.3 YJM-элементы

Рассмотрим возрастающее семейство конечных групп $G_1 \subseteq G_2 \subseteq \cdots \subseteq G_n \subseteq \cdots$. Эти группы имеют семейство образующих $s_1, s_2, \ldots, s_n, \ldots$, так, что

$$G_n =< s_1, s_2, \ldots, s_n > \qquad (6)$$

Пусть $\sigma$ – такой элемент группы $G_m$, что никакой элемент, сопряжённый с ним в группе $G_m$ не содержится ни в какой группе с меньшим номером. Символом $\Xi_n(\sigma)$ (при $n \geq m$) обозначим сумму всех элементов в групповой алгебре группы $G_n$, которые сопряжёны в этой группе с элементом $\sigma$. Эти суммы будем называть элементами Юнга-Юциса-Мерфи или YJM-элементами [8].

Заметим, что элемент $\Xi_n(\sigma)$ является разностью центрального элемента $C_n(\sigma)$ групповой алгебры группы $G_n$ и центрального элемента $C_{n-1}(\sigma_j)$ групповой алгебры предыдущей группы. Элементы $\{\sigma_j\}$ образуют полную группу представителей классов сопряжённости в группе $G_{n-1}$ элементов этой группы, сопряжённых в группе $G_n$ с элементом $\sigma$: $\Xi_n(\sigma) = C_n(\sigma) - C_{n-1}(\sigma_j)$. Эти элементы, даже с разными номерами и соответствующие разным элементам, коммутируют между собой [9]. Действительно:

$$\begin{aligned} \Xi_n(\sigma)\cdot\Xi_m(\eta) &= \Big(C_n(\sigma) - C_{n-1}\big(\sigma_j\big)\Big)\big(C_m(\eta) - C_{m-1}(\eta_k)\big) = \\ &= C_n(\sigma)\cdot C_m(\eta) - C_{n-1}\big(\sigma_j\big)\cdot C_m(\eta) + C_n(\sigma)\cdot C_m(\eta) \\ &\quad - C_{n-1}\big(\sigma_j\big)\cdot C_{m-1}(\eta_k) \end{aligned} \tag{7}$$

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 $\Xi_n(\sigma)$ 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 [8].

В основном примере групповых алгебр симметрических групп $s_i = (i, i+1)$ – кокстеровские образующие симметрической группы. Тогда

$$\Xi_n(\sigma) = \sum_{i=1}^{n-1} (i, n) \qquad (8)$$

классические элементы Юнга-Юциса-Мерфи [9].

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

### 1.4 Симметрические многочлены от YJM-элементов

Рассмотрим последовательность коммутативных алгебр $Q_n(y_1, \dots, y_n)$ симметрических многочленов с целыми коэффициентами от формальных переменных $y_1, \dots, y_n$. Подстановка в переменные $y_k$ элементов (8) индуцирует гомоморфизм алгебры $Q_n$ в центр $Z_n$ групповой алгебры $C[S_n]$ $n$-ой симметрической группы $S_n$.

Теорема. Симметрические многочлены от YJM-элементов порождают центр и выражаются как линейные комбинации сумм классов сопряжённости, совпадающих с суммами перестановок одного циклического типа [10].

## 2 Актуальность темы выпускной квалификационной работы.

### Постановка задачи

Тема выпускной квалификационной работы актуальна, так как рассмотрение даже простейшей усложнённой ситуации (группы серии BC) 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 $Z_n$ 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;
- наличие специального языка запросов LINQ, который значительно упрощает работу с большими объемами данных;
- встроенные реализации списков, словарей объектов, поддерживающие поиск, выполнение сортировки, выборку данных и другие операции;
- кроссплатформенность языка позволяет разработку и использование программы независимо от используемого окружения – операционной системы, IDE, компилятора.

## 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.

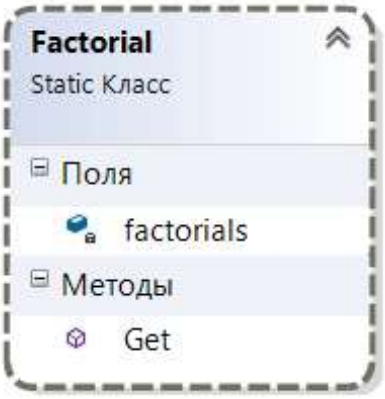


Рисунок 1 – Схема класса Factorial

Статический класс NumberSplits используется для генерации разбиений числа на слагаемые.

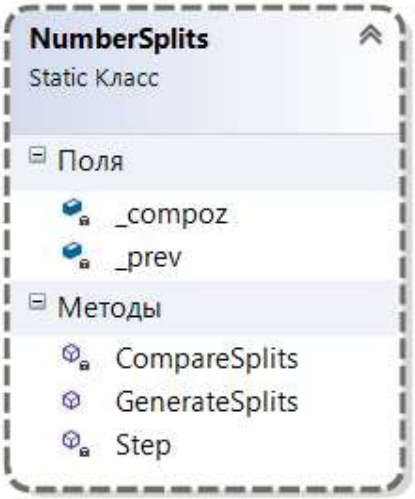


Рисунок 2 – Схема класса NumberSplits

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>> combinations – 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.

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.

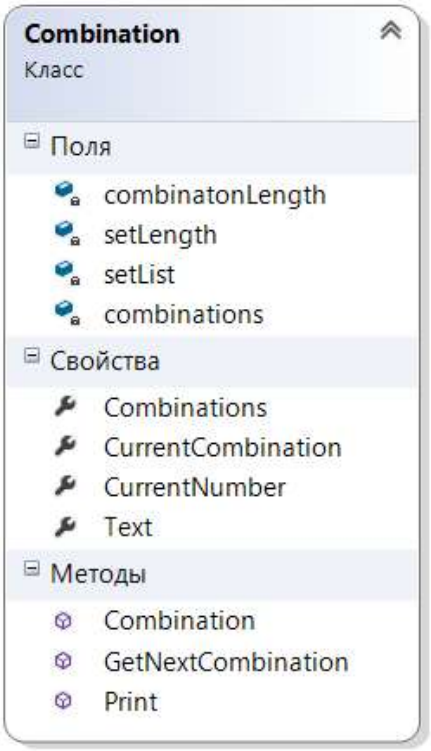


Рисунок 3 – Схема класса Combination

Далее рассмотрим основные классы. Класс Cycle реализует цикл перестановки. Класс содержит поле private int[] cycle, которое используется для хранения элементов цикла.

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.

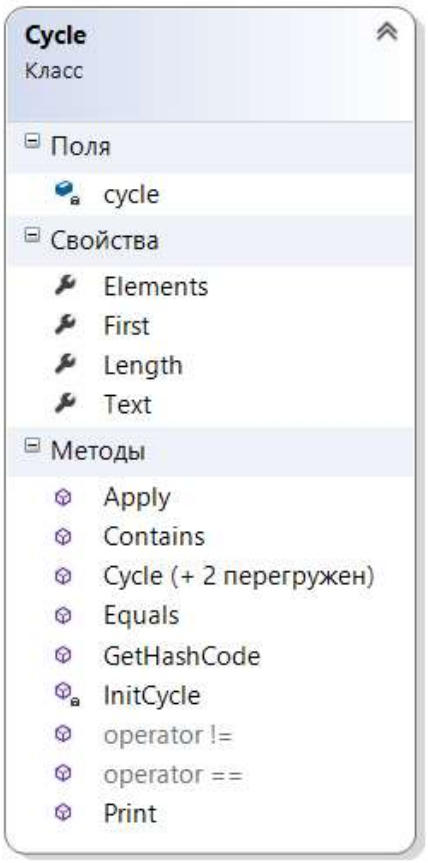


Рисунок 4 – Схема класса Cycle

Следующий класс – Permutation, который реализует перестановку множества.

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.

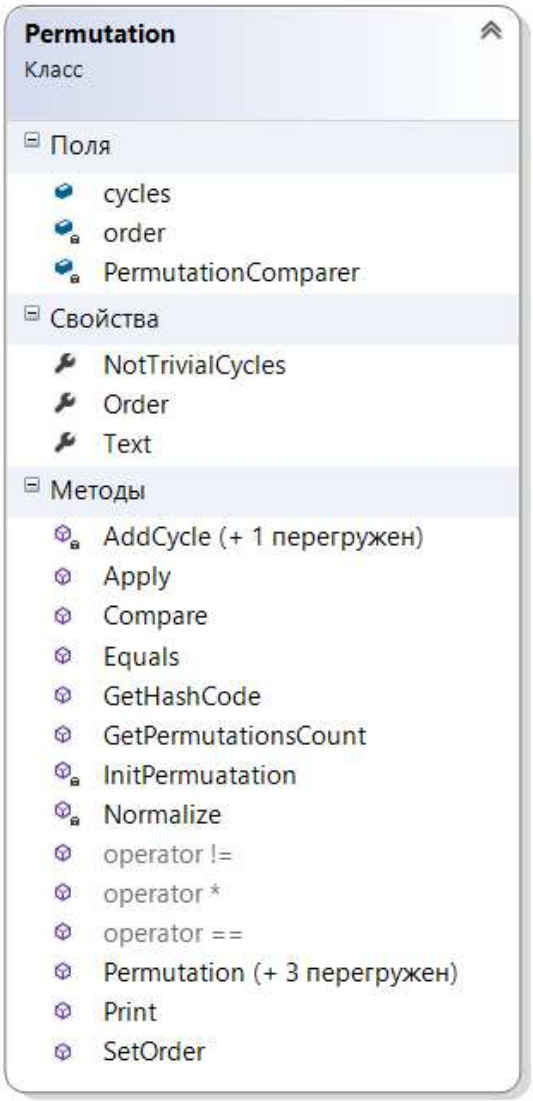


Рисунок 5 – Схема класса Permutation

Следующий класс ElementarySymmetricPolynomial, который описывает элементарный симметрический многочлен.

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.

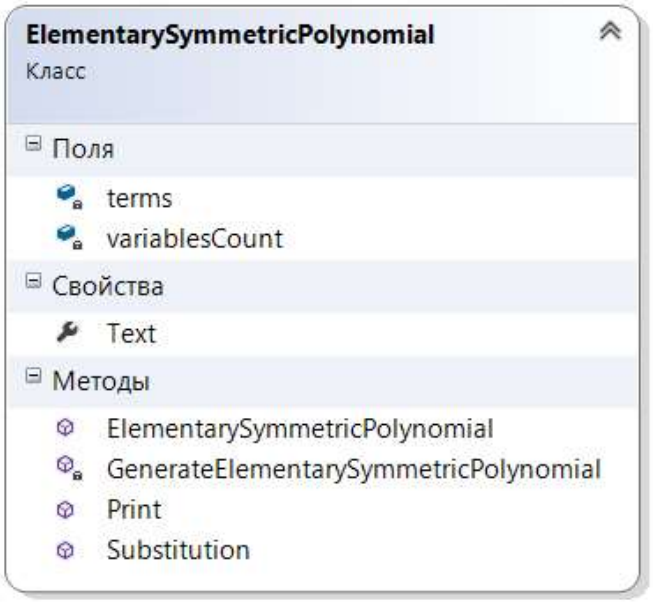


Рисунок 6 – Схема класса ElementarySymmetricPolynomial

Класс YJMElement описывает YJM-элемент.

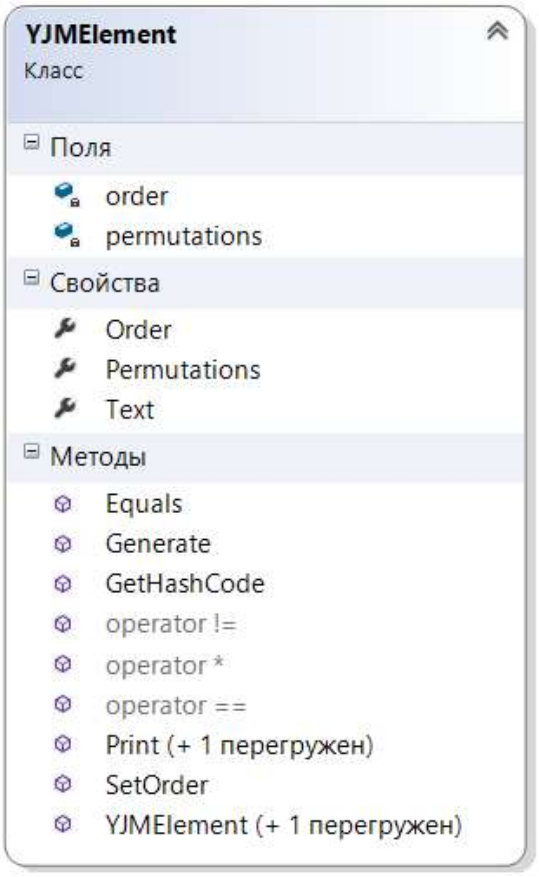


Рисунок 7 – Схема класса YJMElement

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 наследуются от класса Dictionary<Permutation, int>.

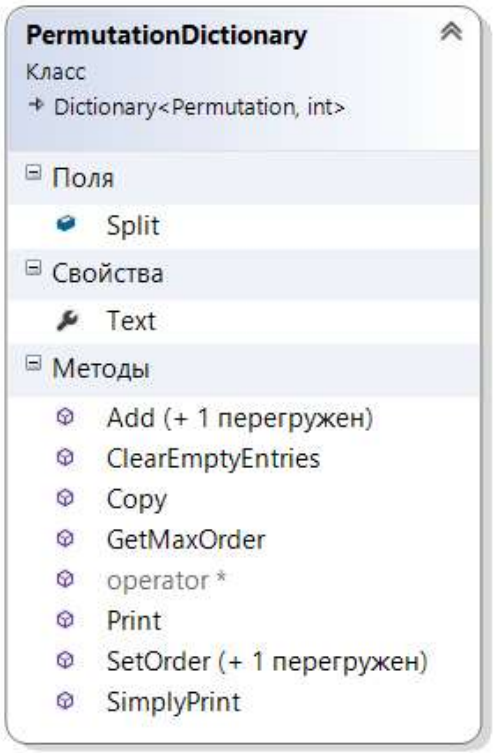


Рисунок 8 – Схема класса PermutationDictionary

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 static 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 в работе программы – это задание исходных параметров – количества элементов групповой алгебры $k$ (или количества переменных в элементарных симметрических многочленах), а также максимальной степени многочлена $n$. Исходя из этих параметров, генерируются первые $n$ элементарных симметрических многочленов и $k$ YJM-элементов. Многочлены генерируются следующим образом: для каждого из чисел от 1 до $n$ генерируются неупорядоченные выборки $m_i$ соответствующего

количества элементов из множества чисел $(1, 2, \dots, k)$. Эти выборки определяют слагаемые симметрического многочлена, $i$-е слагаемое представляет собой произведение переменных с номерами из выборки $m_i$. YJM-элементы генерируются по формуле (8) для каждого из чисел от 2 до $k$ включительно.

После генерации производится подстановка YJM-элементов вместо переменных с соответствующими номерами в элементарные симметрические многочлены. После перемножения YJM-элементов получается сумма перестановок с некоторыми коэффициентами.

Далее для каждого из чисел от 2 до $n$ генерируются всевозможные разбиения этого числа. Процесс разбиения проиллюстрирован на рисунке 9. Каждому из разбиений соответствует некоторое произведение элементарных симметрических многочленов. Множителями произведения являются многочлены с номерами, соответствующими элементам произведения. После перемножения элементарных многочленов получаем набор сумм перестановок с коэффициентами, каждая из которых соответствует какому-либо произведению элементарных симметрических многочленов.

Далее эти суммы группируются по циклическому типу – в сумме остается только одна перестановка каждого циклического типа с коэффициентом, равным сумм коэффициентов перед всеми перестановками этого циклического типа. В результате получаем системы уравнений следующего вида:

$$\begin{cases} x_1 = \sigma_1 \\ x_2 + x_3 + 0 = \sigma_2 \\ 3x_2 + 2x_3 + 10 = \sigma_1^2 \end{cases} \quad (11)$$

где $x_1$ – перестановка циклического типа (2,1),

$x_2$ – перестановка циклического типа (3,2,1),

$x_3$ – перестановка циклического типа (3,2)(4,1),

$\sigma_i$ – $i$-ый элементарный симметрический многочлен.

Система (11) приведена для случая $k = 4, n = 2$. 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, которая позволяет выполнять символьные вычисления.

### 3.4 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 из систем приведена в формуле (11). Вторая система для случая $k = 6, n = 3$ выглядит следующим образом:

$$\begin{cases} x_1 = \sigma_1 \\ x_2 + x_3 = \sigma_2 \\ 3x_2 + 2x_3 + 21 = \sigma_1^2 \\ x_4 + x_5 + x_6 = \sigma_3 \\ 20x_1 + 6x_4 + 4x_5 + 3x_6 = \sigma_2\sigma_1 \\ 71x_1 + 16x_4 + 9x_5 + 6x_6 = \sigma_1^3 \end{cases} \quad (12)$$

где $x_1$ – перестановка циклического типа (2,1),

$x_2$ – перестановка циклического типа (3,2,1),

$x_3$ – перестановка циклического типа (3,2)(4,1),

$x_4$ – перестановка циклического типа (4,3,2,1),

$x_5$ – перестановка циклического типа (3,2,1)(5,4),

$x_6$ – перестановка циклического типа (2,1)(4,3)(6,5),

$\sigma_i$ – $i$-ый элементарный симметрический многочлен.

Остальные рассчитанные системы приведены в приложении В.

Определители матриц систем образуют следующую последовательность.

Таблица 1. Определители матриц систем

| $k=2,n=1$ | $k=4,n=2$ | $k=6,n=3$ | $k=8,n=4$ | $k=10,n=5$ |
|---|---|---|---|---|
| 1 | -1 | 1 | 1 | -1 |

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

$$\begin{cases} (2,1) = \sigma_1 \\ (2,1)(4,3) = 10 + 3\sigma_2 - \sigma_1^2 \\ (3,2,1) = -10 - 2\sigma_2 + \sigma_1^2 \end{cases} \quad (13)$$

$$\begin{cases} (2,1) = \sigma_1 \\ (2,1)(4,3) = 21 + 3\sigma_2 - \sigma_1^2 \\ (2,1)(4,3)(6,5) = -2\sigma_1 + 10\sigma_3 - 7\sigma_2\sigma_1 + 2\sigma_1^3 \\ (3,2,1)(5,4) = 13\sigma_1 - 12\sigma_3 + 10\sigma_2\sigma_1 - 3\sigma_1^3 \\ (4,3,2,1) = -11\sigma_1 + 3\sigma_3 - 3\sigma_2\sigma_1 + \sigma_1^3 \\ (3,2,1) = -21 - 2\sigma_2 + \sigma_1^2 \end{cases} \quad (14)$$

Решения остальных систем также приведены в приложении В.

## 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;
- показано, что у этого отображения нет ядра;
- стандартные элементы центра выражены через симметрические многочлены.

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.

**Приложение А**
**(справочное).**
**Схемы алгоритмов основных функций**

# Приложение Б
# (справочное).
# Часть листинга программы

```
/// <summary>
/// Класс для вычисления факториалов
/// </summary>
public static class Factorial
{
        /// <summary>
        /// Вычисленные факториалы
        /// </summary>
        private static readonly List<int> factorials = new List<int> { 1, 1 };
        /// <summary>
        /// Вычислить факториал числа
        /// </summary>
        /// <param name="num">Число</param>
                public static int Get(int num)
                {
                        if (num >= factorials.Count)
                        {
                                for (int i = factorials.Count; i <= num; i++)
                                {
                                        factorials.Add(factorials[i - 1] * i);
                                }
                        }
                        return factorials[num];
                }
        }
public static class NumberSplits
{
        private static List<int> _prev = new List<int>();
        private static List<List<int>> _compoz = new List<List<int>>();
        /// <summary>
        /// генерирует разбиения числа
        /// </summary>
        /// <param name="n">Число</param>
        public static List<List<int>> GenerateSplits(int n)
        {
                n += 1;
                Step(n - 1, n - 1);
                var fCompoz = new List<List<int>>();
                foreach (var item in _compoz)
                {
                        fCompoz.Add(new List<int>());
                        item.Sort((delegate (int x, int y)
                        {
                                if (x > y)
                                {
                                        return -1;
                                }
                                else if (x < y)
                                {
                                        return 1;
                                }
                                else
                                {
                                        return 0;
                                }
                        }));
                        fCompoz.Last().AddRange(item);
                        item.Clear();
                }
                _compoz.Clear();
                for (int i = fCompoz.Count - 1; i >= 0; i--)
                {
                        for (int j = i - 1; j >= 0; j--)
                        {
                                if (CompareSplits(fCompoz[i], fCompoz[j]))
                                {
                                        fCompoz.RemoveAt(i);
```

```
                                break;
                        }
                }
        }
        fCompoz.Sort(delegate (List<int> x, List<int> y)
        {
                if (x.Count > y.Count)
                {
                        return 1;
                }
                else if (x.Count < y.Count)
                {
                        return -1;
                }
                else
                {
                        for (int i = 0; i < x.Count; i++)
                        {
                                if (x[i] > y[i])
                                {
                                        return -1;
                                }
                                else if (x[i] < y[i])
                                {
                                        return 1;
                                }
                        }
                        return 0;
                }
        });
        return fCompoz;
}
/// <summary>
/// шаг рекурсии, считающей композиции числа
/// </summary>
/// <param name="n">N.</param>
/// <param name="beg">Beg.</param>
private static void Step(int n, int beg)
{
        for (int i = 1; i <= n; i++)
        {
                _prev.Add(i);
                Step(n - i, beg);
        }
        if (_prev.Sum() == beg)
        {
                if (_prev.Count == 0)
                {
                        _prev.Add(0);
                }
                _compoz.Add(new List<int>(_prev));
        }
        if (_prev.Count > 0)
        {
                _prev.RemoveAt(_prev.Count - 1);
        }
}
        /// <summary>
        /// Compares the splits2.
        /// </summary>
        /// <returns><c>true</c>, if splits2 was compared, <c>false</c> otherwise.</returns>
        /// <param name="l1">L1.</param>
        /// <param name="l2">L2.</param>
        private static bool CompareSplits(List<int> l1, List<int> l2)
        {
                if (l1.Count != l2.Count)
                {
                        return false;
                }
                for (int i = 0; i < l1.Count; i++)
                {
                        if (l1 [i] != l2 [i])
                        {
```

```
                                        return false;
                                }
                        }
                        return true;
                }
        }
/// <summary>
        /// Класс для полчения неупорядоченных выборок множества
        /// </summary>
        public class Combination
        {
                /// <summary>
                /// Множество
                /// </summary>
                private List<int> setList = new List<int>();
                /// <summary>
                /// Все уже полученные выборки
                /// </summary>
                private List<List<int>> combinations = new List<List<int>> ();
                /// <summary>
                /// Количество элементов множства
                /// </summary>
                private int setLength;
                /// <summary>
                /// Количество элементов выборки
                /// </summary>
                private int combinatonLength;
                /// <summary>
                /// Выборка
                /// </summary>
                public List<int> CurrentCombination
                {
                        get
                        {
                                var res = new List<int> ();
                                for (int i = 0; i < combinatonLength; i++)
                                {
                                        res.Add (setList [i]);
                                }
                                return res;
                        }
                }
                /// <summary>
                /// Все уже полученные выборки
                /// </summary>
                public List<List<int>> Combinations
                {
                        get { return combinations; }
                }
                /// <summary>
                /// Текущий номер выборки
                /// </summary>
                public int CurrentNumber
                {
                        get { return combinations.Count; }
                }
                /// <summary>
                /// Текстовое представление
                /// </summary>
                public string Text
                {
                        get { return string.Join (" ", CurrentCombination); }
                }
                /// <summary>
                /// Инициализирует экземпляр класса <see cref="Combination"/>
                /// </summary>
                /// <param name="_setLength">Количество элементов множества</param>
                /// <param name="_combinationLength">Количетсов элементов выборки</param>
                public Combination(int _setLength, int _combinationLength)
                {
                        setLength = _setLength;
                        combinatonLength = _combinationLength;
                        for (int i = 0; i < setLength; i++)
```

```
                {
                        setList.Add (i + 1);
                }
                combinations.Add (CurrentCombination);
        }
        /// <summary>
        /// Вычисляет следующую выборку
        /// </summary>
        /// <returns><c>true</c>, если выборка получена, иначе - <c>false</c></returns>
        public bool GetNextCombination ()
        {
                for (int i = combinatonLength - 1; i >= 0; i--)
                {
                        if (setList [i] < setLength - combinatonLength + i + 1)
                        {
                                setList [i]++;
                                for (int j = i + 1; j < combinatonLength; j++)
                                {
                                        setList [j] = setList [j - 1] + 1;
                                }
                                combinations.Add (CurrentCombination);
                                return true;
                        }
                }
                return false;
        }
}
/// <summary>
/// Класс, описывающий цикл перестанови
/// </summary>
public class Cycle
{
        /// <summary>
        /// Сам цикл
        /// </summary>
        private int[] cycle;
        /// <summary>
        /// Длина цикла
        /// </summary>
        public int Length
        {
                get { return cycle.Length; }
        }
        /// <summary>
        /// Элементы цикла
        /// </summary>
        public int[] Elements
        {
                get { return cycle; }
        }
        /// <summary>
        /// Первый элемент цикла
        /// </summary>
        public int First
        {
                get { return cycle [0]; }
        }
        /// <summary>
        /// Текстовое представление
        /// </summary>
        public string Text
        {
                get { return "(" + string.Join (",", cycle) + ")"; }
        }
        /// <summary>
        /// Инициализирует экземпляр класса <see cref="Cycle"/>
        /// </summary>
        public Cycle ()
        {
                cycle = new int[0];
        }
        /// <summary>
        /// Инициализирует экземпляр класса <see cref="Cycle"/>
```

```
/// </summary>
/// <param name="newCycle">Массив элементов цикла</param>
public Cycle (params int[] newCycle)
{
	InitCycle (newCycle);
}
/// <summary>
/// Конструктор копирования класса <see cref="Cycle"/>
/// </summary>
/// <param name="newCycle">Другой цикл</param>
public Cycle (Cycle newCycle)
{
	InitCycle (newCycle.Elements);
}
/// <summary>
/// Инициализирует цикл массивом элементов
/// </summary>
/// <param name="newCycle">Массив элементов</param>
private void InitCycle(int[] newCycle)
{
	cycle = new int[newCycle.Length];
	var maxIndex = -1;
	var maxValue = -1;
	for (int i = 0; i < newCycle.Length; i++)
	{
		if (newCycle [i] > maxValue)
		{
			maxValue = newCycle [i];
			maxIndex = i;
		}
	}
	for (int i = maxIndex, j = 0; j < cycle.Length; i++, j++)
	{
		if (i == newCycle.Length)
		{
			i = 0;
		}
		cycle [j] = newCycle [i];
	}
}
/// <summary>
/// Применяет цикл к элементу
/// </summary>
/// <returns>The cycle.</returns>
/// <param name="number">Элемент</param>
public int Apply(int number)
{
	var res = number;
	for (int i = 0; i < cycle.Length; i++)
	{
		if (cycle [i] == number)
		{
			return cycle [(i + 1) % cycle.Length];
		}
	}
	return res;
}
/// <summary>
/// Содержит ли цикл заданный элемент
/// </summary>
/// <param name="number">Элемент</param>
public bool Contains(int number)
{
	return cycle.Contains (number);
}
/// <summary>
/// Определяет равен ли объект <see cref="object"/> данному объекту <see cref="Cycle"/>.
/// </summary>
/// <param name="obj">Объект, который нужно сравнить</param>
public override bool Equals(object obj)
{
	return obj is Cycle && this == obj as Cycle;
}
```

```
        /// <summary>
        /// Возвращает хэш-код объекта <see cref="Cycle"/>
        /// </summary>
        public override int GetHashCode()
        {
                return Text.GetHashCode ();
        }
        /// <param name="c1">Цикл 1</param>
        /// <param name="c2">Цикл 2</param>
        public static bool operator ==(Cycle c1, Cycle c2)
        {
                if (c1.Length == c2.Length)
                {
                        for (int i = 0; i < c1.Length; i++)
                        {
                                if (c1.Elements [i] != c2.Elements [i])
                                {
                                        return false;
                                }
                        }
                }
                else
                {
                        return false;
                }
                return true;
        }
        /// <param name="c1">Цикл 1</param>
        /// <param name="c2">Цикл 2</param>
        public static bool operator !=(Cycle c1, Cycle c2)
        {
                return !(c1 == c2);
        }
}
/// <summary>
/// Класс, описывающий перестановку
/// </summary>
public class Permutation
{
        /// <summary>
        /// Компаратор для сортировки циклов перестановки
        /// </summary>
        private static Comparison<Cycle> PermutationComparer = new Comparison<Cycle> ((x, y) =>
        {
                if (x.First == y.First)
                {
                        return 0;
                }
                else
                {
                        return x.First < y.First ? -1 : 1;
                }
        });
        /// <summary>
        /// Порядок перестановки
        /// </summary>
        private int order;
        /// <summary>
        /// Циклы
        /// </summary>
        public List<Cycle> cycles;
        /// <summary>
        /// Gets the not trivial cycles.
        /// </summary>
        /// <value>The not trivial cycles.</value>
        public List<Cycle> NotTrivialCycles
        {
                get
                {
                        return cycles.Where (c => c.Elements.Length > 1).ToList ();
                }
        }
        /// <summary>
```

```
		/// Порядок перестановки
		/// </summary>
		public int Order
		{
			get { return order; }
		}
		/// <summary>
		/// Текствое представление
		/// </summary>
		/// <value>The text.</value>
		public string Text
		{
			get
			{
				if (NotTrivialCycles.Count > 0)
				{
					return string.Join ("", NotTrivialCycles.Select (cycle =>
cycle.Text));
				}
				else
				{
					return string.Join ("", cycles.Select (cycle => cycle.Text));
				}
			}
		}
		/// <summary>
		/// Инициализирует экземпляр класса <see cref="Permutation"/>
		/// </summary>
		public Permutation (int o)
		{
			order = o;
			cycles = new List<Cycle> ();
		}
		/// <summary>
		/// Инициализирует экземпляр класса <see cref="Permutation"/>
		/// </summary>
		/// <param name="_cycles">Коллекция циклов</param>
		public Permutation (int o, IEnumerable<Cycle> _cycles)
		{
			order = o;
			InitPermuatation(_cycles);
		}
		/// <summary>
		/// Инициализирует экземпляр класса <see cref="Permutation"/>
		/// </summary>
		/// <param name="_cycles">Циклы</param>
		public Permutation (int o, params Cycle[] _cycles)
		{
			order = o;
			InitPermuatation(_cycles.ToList());
		}
		/// <summary>
		/// Инициализирует экземпляр класса <see cref="Permutation"/>
		/// </summary>
		/// <param name="cycles">Массивы элементов циклов</param>
		public Permutation(int o, params int[][] cycles)
		{
			order = o;
			var lsCycles = new List<Cycle> ();
			foreach (var cycle in cycles)
			{
				lsCycles.Add (new Cycle (cycle));
			}
			InitPermuatation(lsCycles);
		}
		/// <summary>
		/// Инициализирует перестановку
		/// </summary>
		/// <param name="_cycles">Коллекция циклов</param>
		private void InitPermuatation(IEnumerable<Cycle> _cycles)
		{
			cycles = new List<Cycle> (_cycles);
			Normalize ();
```

```
}
/// <summary>
/// Приводит перестановку к стандартному виду
/// </summary>
private void Normalize()
{
	for (int i = 1; i <= order; i++)
	{
		if (cycles.Count (c => c.Contains (i)) == 0)
		{
			cycles.Add (new Cycle (new [] { i }));
		}
	}
	cycles.Sort (PermutationComparer);
}
/// <summary>
/// Добавляет в перестановку цикл
/// </summary>
/// <param name="newCycle">Новый цикл</param>
private void AddCycle(Cycle newCycle, bool needNormalize = false)
{
	cycles.Add (newCycle);
	if (needNormalize)
	{
		Normalize ();
	}
}
/// <summary>
/// Добавляет в перестановку цикл
/// </summary>
/// <param name="newCycle">Новый цикл</param>
private void AddCycle(int[] newCycle, bool needNormalize = false)
{
	cycles.Add (new Cycle(newCycle));
	if (needNormalize)
	{
		Normalize ();
	}
}
/// <summary>
/// Применяет перестановку к заданному числу
/// </summary>
/// <param name="number">Число</param>
public int Apply(int number)
{
	var res = number;
	foreach (var cycle in cycles)
	{
		if (cycle.Contains (number))
		{
			res = cycle.Apply (number);
			break;
		}
	}
	return res;
}
/// <summary>
/// Устанавливает порядок перестановки
/// </summary>
/// <param name="newOrder">Новый порядок</param>
public void SetOrder(int newOrder)
{
	if (newOrder > order)
	{
		order = newOrder;
		for (int i = order + 1; i <= newOrder; i++)
		{
			AddCycle (new [] { i });
		}
		Normalize ();
	}
}
/// <summary>
```

```
		/// <summary>
		/// "Сравнение двух перестановок" с точки зрения циклической структуры
		/// </summary>
		/// <param name="p1">P1.</param>
		/// <param name="p2">P2.</param>
		public static bool Compare(Permutation p1, Permutation p2)
		{
			if (p1.NotTrivialCycles.Count != p2.NotTrivialCycles.Count)
			{
				return false;
			}
			var fLen = p1.NotTrivialCycles.Select(cycle => cycle.Length).ToList();
			var sLen = p2.NotTrivialCycles.Select(cycle => cycle.Length).ToList();
			fLen.Sort();
			sLen.Sort();
			for (int i = 0; i < fLen.Count; i++)
			{
				if (fLen[i] != sLen[i])
				{
					return false;
				}
			}
			return true;
		}
		/// <summary>
		/// Количество перестановок определенного циклического типа
		/// </summary>
		/// <param name="p">Перестановка</param>
		public static int GetPermutationsCount(Permutation p)
		{
			var dict = new Dictionary<int, int>();
			foreach (var cycle in p.cycles)
			{
				if (!dict.ContainsKey(cycle.Length))
				{
					dict.Add(cycle.Length, 0);
				}
				dict[cycle.Length]++;
			}
			var nnn = 1;
			foreach (var kvp in dict)
			{
				nnn *= (int)Math.Pow(kvp.Key, kvp.Value) * Factorial.Get(kvp.Value);
			}
			return Factorial.Get(p.Order) / nnn;
		}
		/// <summary>
		/// Определяет равен ли объект <see cref="object"/> данному объекту <see
cref="Permutation"/>.
		/// </summary>
		/// <param name="obj">Объект, который нужно сравнить</param>
		public override bool Equals(object obj)
		{
			return obj is Permutation && this == obj as Permutation;
		}
		/// <summary>
		/// Возвращает хэш-код объекта <see cref="Permutation"/>
		/// </summary>
		public override int GetHashCode()
		{
			return Text.GetHashCode ();
		}
		/// <param name="p1">Первая перестановка</param>
		/// <param name="p2">Вторая перестановка</param>
		public static Permutation operator *(Permutation p1, Permutation p2)
		{
			if (p1.order != p2.order)
			{
				if (p1.order > p2.order)
				{
					p2.SetOrder(p1.order);
				}
				else if (p1.order < p2.order)
```

```
                        {
                                p1.SetOrder(p2.order);
                        }
                }
                var result = new Permutation (p1.order);
                foreach (var cycle in p1.cycles)
                {
                        var ls = new List<int>();
                        foreach (var elem in cycle.Elements)
                        {
                                if (result.cycles.Count(c => c.Contains(elem)) == 0)
                                {
                                        ls.Add(elem);
                                        var i = 0;
                                        while (i < ls.Count)
                                        {
                                                var s = p2.Apply(p1.Apply(ls[i]));
                                                if (!ls.Contains(s))
                                                {
                                                        ls.Add(s);
                                                }
                                                i++;
                                        }
                                        if (ls.Count > 0)
                                        {
                                                result.AddCycle(ls.ToArray());
                                                ls = new List<int>();
                                        }
                                }
                        }
                }
                result.Normalize ();
                return result;
        }
        /// <param name="p1">Первая перестановка</param>
        /// <param name="p2">Вторая перестановка</param>
        public static bool operator ==(Permutation p1, Permutation p2)
        {
                if (p1.order == p2.order)
                {
                        for (int i = 0; i < p1.cycles.Count; i++)
                        {
                                if (p1.cycles [i] != p2.cycles [i])
                                {
                                        return false;
                                }
                        }
                }
                else
                {
                        return false;
                }
                return true;
        }
        /// <param name="p1">Первая перестановка</param>
        /// <param name="p2">Вторая перестановка</param>
        public static bool operator !=(Permutation p1, Permutation p2)
        {
                return !(p1 == p2);
        }
    }
/// <summary>
        /// Класс, описывающий элементарный симметрический многочлен
        /// </summary>
        public class ElementarySymmetricPolynomial
        {
                /// <summary>
                /// Количество переменных
                /// </summary>
                private int variablesCount;
                /// <summary>
                /// Переменные
                /// </summary>
```

```
		private List<List<int>> terms = new List<List<int>>();
		/// <summary>
		/// Текстовое представление
		/// </summary>
		/// <value>The text.</value>
		public string Text
		{
			get
			{
				return string.Join (" + ", terms.Select (t => string.Join ("*", t.Select
(tt => "X" + tt))));
			}
		}
		/// <summary>
		/// Инициализирует экземпляр класса <see cref="ElementarySymmetricPolynomial"/>
		/// </summary>
		/// <param name="_varCount">Количество переменных</param>
		/// <param name="_num">Номер элементарного симметрического многочлена</param>
		public ElementarySymmetricPolynomial (int _varCount, int _num)
		{
			if (_num > _varCount)
			{
				throw new ArgumentException ("Номер многочлена не может быть больше
количества переменных.");
			}
			variablesCount = _varCount;
			GenerateElementarySymmetricPolynomial (_num);
		}
		/// <summary>
		/// Генерирует элементарный симметрический многочлен
		/// </summary>
		private void GenerateElementarySymmetricPolynomial(int num)
		{
			var combination = new Combination (variablesCount, num);
			do
			{
				terms.Add(combination.CurrentCombination);
			}
			while (combination.GetNextCombination ());
		}
		/// <summary>
		/// Подставить аргументы
		/// </summary>
		/// <param name="args">Список аргументов</param>
		public PermutationDictionary Substitution(List<YJMElement> args)
		{
			var res = new PermutationDictionary ();
			foreach (var term in terms)
			{
				var tmp = args[term[term.Count - 1] - 1];
				for (int i = term.Count - 2; i >= 0; i--)
				{
					tmp = tmp * args [term [i] - 1];
				}
				res.Add (tmp.Permutations);
			}
			return res;
		}
	}
/// <summary>
	/// Класс, представляющий YJM элемент
	/// </summary>
	public class YJMElement
	{
		/// <summary>
		/// Порядок входящих в YJM-элемент перестановок
		/// </summary>
		private int order;
		/// <summary>
		/// Список перестановок
		/// </summary>
		private List<Permutation> permutations;
		/// <summary>
```

```
		/// Порядок перестановки
		/// </summary>
		public int Order
		{
			get { return order; }
		}
		/// <summary>
		/// Список перестановок
		/// </summary>
		public List<Permutation> Permutations
		{
			get { return permutations; }
		}
		/// <summary>
		/// Текствое представление
		/// </summary>
		/// <value>The text.</value>
		public string Text
		{
			get
			{
				return string.Format("S({0}) = {1}", order, string.Join("+",
permutations.Select(p => p.Text)));
			}
		}
		/// <summary>
		/// Инициализирует экземпляр класса <see cref="YJMElement"/>
		/// </summary>
		/// <param name="ord">Порядок</param>
		public YJMElement(int ord)
		{
			order = ord;
			permutations = new List<Permutation>();
			for (int i = 1; i < ord; i++)
			{
				permutations.Add(new Permutation(ord, new int[] { i, ord }));
			}
		}
		/// <summary>
		/// Инициализирует экземпляр класса <see cref="YJMElement"/>
		/// </summary>
		/// <param name="perm">Список перестановок</param>
		public YJMElement(List<Permutation> perm)
		{
			permutations = new List<Permutation>();
			permutations.AddRange(perm);
		}
		/// <summary>
		/// Устанавливает порядок YJM-элемента
		/// </summary>
		/// <param name="newOrder">Новый порядок</param>
		public void SetOrder(int newOrder)
		{
			if (newOrder > order)
			{
				order = newOrder;
				foreach (var perm in permutations)
				{
					perm.SetOrder(newOrder);
				}
			}
		}
		/// <summary>
		/// Генерирует набор последовательных YJM-элементов
		/// </summary>
		/// <param name="order">Порядок группы</param>
		public static List<YJMElement> Generate(int order)
		{
			var result = new List<YJMElement>();
			for (int i = 2; i <= order; i++)
			{
				result.Add(new YJMElement(i));
			}
```

```
                return result;
        }
        /// <summary>
        /// Печатает набор YJM-элементов
        /// </summary>
        /// <param name="elements">Набор YJM-элементов</param>
        public static void Print(List<YJMElement> elements)
        {
                foreach (var elem in elements)
                {
                        elem.Print();
                }
        }
        /// <summary>
        /// Определяет равен ли объект <see cref="object"/> данному объекту <see
cref="YJMElement"/>.
        /// </summary>
        /// <param name="obj">Объект, который нужно сравнить</param>
        public override bool Equals(object obj)
        {
                return obj is YJMElement && this == obj as YJMElement;
        }
        /// <summary>
        /// Возвращает хэш-код объекта <see cref="YJMElement"/>
        /// </summary>
        public override int GetHashCode()
        {
                return permutations.GetHashCode();
        }
        /// <param name="y1">Первый YJM-элемент</param>
        /// <param name="y2">Второй YJM-элемент</param>
        public static YJMElement operator *(YJMElement y1, YJMElement y2)
        {
                if (y1.order > y2.order)
                {
                        y2.SetOrder(y1.order);
                }
                else if (y2.order > y1.order)
                {
                        y1.SetOrder(y2.order);
                }
                var result = new List<Permutation>();
                foreach (var perm1 in y1.permutations)
                {
                        foreach (var perm2 in y2.permutations)
                        {
                                result.Add(perm1 * perm2);
                        }
                }
                return new YJMElement(result);
        }
        /// <param name="y1">Первый YJM-элемент</param>
        /// <param name="y2">Второй YJM-элемент</param>
        public static bool operator ==(YJMElement y1, YJMElement y2)
        {
                if (y1.order == y2.order)
                {
                        for (int i = 0; i < y1.order; i++)
                        {
                                if (y1.permutations[i] != y2.permutations[i])
                                {
                                        return false;
                                }
                        }
                }
                else
                {
                        return false;
                }
                return true;
        }
        /// <param name="y1">Первый YJM-элемент</param>
        /// <param name="y2">Второй YJM-элемент</param>
```

```
        public static bool operator !=(YJMElement y1, YJMElement y2)
        {
                return !(y1 == y2);
        }
}
/// <summary>
/// Словарь "перестановка - коэффициент"
/// </summary>
public class PermutationDictionary : Dictionary<Permutation, int>
{
        /// <summary>
        /// Разбиение, соответствующее объекту
        /// </summary>
        public List<int> Split = new List<int>();
        /// <summary>
        /// Текстовое представление
        /// </summary>
        public string Text
        {
                get
                {
                        return string.Join(" + ", this.Select(kvp =>
                        {
                                if (kvp.Value < 1)
                                {
                                        return "";
                                }
                                else if (kvp.Value == 1)
                                {
                                        return kvp.Key.Text;
                                }
                                else
                                {
                                        return kvp.Value + "*" + kvp.Key.Text;
                                }
                        }));
                }
        }
        /// <summary>
        /// Добавляет перестановку в словарь
        /// </summary>
        /// <param name="p">Перестановка</param>
        public new void Add(Permutation p, int count = 1)
        {
                if (ContainsKey(p))
                {
                        this[p] += count;
                }
                else
                {
                        base.Add(p, count);
                }
        }
        /// <summary>
        /// Добавляет несколько перестановок в словарь
        /// </summary>
        /// <param name="p">Коллекция перестановок</param>
        public void Add(IEnumerable<Permutation> p)
        {
                foreach (var i in p)
                {
                        Add(i);
                }
        }
        /// <summary>
        /// Печатает словарь перестановок
        /// </summary>
        public void Print(Output output = Output.Console, string path = "", string text = "")
        {
                if (text == "")
                {
                        text = Text;
                }
```

```
                        if (output == Output.Console)
                        {
                                Console.WriteLine(text);
                        }
                        else if (output == Output.File)
                        {
                                using (var fs = File.AppendText(path))
                                {
                                        fs.Write(text);
                                        fs.WriteLine();
                                }
                        }
                }
                /// <summary>
                /// Печатаем только тип перестановки и его количество
                /// </summary>
                public void SimplyPrint(Output output = Output.Console, string path = "", string addText =
""")
                {
                        var dictCounts = new Dictionary<Permutation, int>();
                        foreach (var kvp in this)
                        {
                                var flagInc = false;
                                foreach (var kvp2 in dictCounts)
                                {
                                        if (Permutation.Compare(kvp2.Key, kvp.Key))
                                        {
                                                dictCounts[kvp2.Key] += kvp.Value;
                                                flagInc = true;
                                                break;
                                        }
                                }
                                if (!flagInc)
                                {
                                        dictCounts.Add(kvp.Key, kvp.Value);
                                }
                        }
                        var txt = dictCounts.Select(kvp =>
                        {
                                var cnt = Permutation.GetPermutationsCount(kvp.Key);
                                if (kvp.Value % cnt > 0)
                                {
                                        Console.WriteLine("Error. Order = " + kvp.Key.Order + "; Perm = "
+ kvp.Key.Text + "; Count = " + kvp.Value + "; PermCount = " + cnt + "; Reminder = " + (kvp.Value % cnt));
                                        MainClass.ErrorBeep();
                                        Console.ReadKey();
                                }
                                return ((kvp.Value / cnt) != 1 ? (kvp.Value / cnt).ToString() : "") +
(kvp.Key.NotTrivialCycles.Count > 0 ? ("(" + kvp.Key.Text + ")") : "");
                        });
                        Print(output, path, string.Join(" + ", txt) + addText + "\r\n");
                }
                /// <summary>
                /// Copy the specified PermutationDictionary.
                /// </summary>
                /// <param name="p1">P1.</param>
                public static PermutationDictionary Copy(PermutationDictionary p1)
                {
                        var res = new PermutationDictionary();
                        foreach (var kvp in p1)
                        {
                                res.Add(kvp.Key, kvp.Value);
                        }
                        res.Split.AddRange(p1.Split);
                        return res;
                }
                /// <summary>
                /// Clears the empty entries.
                /// </summary>
                public void ClearEmptyEntries()
                {
                        var keys = new List<Permutation>();
                        foreach (var kvp in this)
```

```
                        {
                                if (kvp.Key.NotTrivialCycles.Count == 0)
                                {
                                        keys.Add(kvp.Key);
                                }
                        }
                        foreach (var key in keys)
                        {
                                Remove(key);
                        }
                }
                /// <summary>
                /// Sets the order.
                /// </summary>
                /// <param name="newOrder">New order.</param>
                public void SetOrder(int newOrder)
                {
                        foreach (var kvp in this)
                        {
                                kvp.Key.SetOrder(newOrder);
                        }
                }
                /// <summary>
                /// Sets the order.
                /// </summary>
                public void SetOrder()
                {
                        SetOrder(GetMaxOrder());
                }
                /// <summary>
                /// Gets the max order.
                /// </summary>
                /// <returns>The max order.</returns>
                public int GetMaxOrder()
                {
                        return Keys.Max(t => t.Order);
                }
                /// <param name="p1">P1.</param>
                /// <param name="p2">P2.</param>
                public static PermutationDictionary operator *(PermutationDictionary p1,
PermutationDictionary p2)
                {
                        var res = new PermutationDictionary();
                        if (p1.Count == 0 && p2.Count > 0)
                        {
                                return Copy(p2);
                        }
                        else if (p2.Count == 0 && p1.Count > 0)
                        {
                                return Copy(p1);
                        }
                        else if (p1.Count == 0 && p2.Count == 0)
                        {
                                throw new Exception("Две нулевых штуки.");
                        }
                        foreach (var kvp1 in p1)
                        {
                                foreach (var kvp2 in p2)
                                {
                                        res.Add(kvp1.Key * kvp2.Key, kvp1.Value * kvp2.Value);
                                }
                        }
                        return res;
                }
        }
/// <summary>
        /// Словарь "перестановка - коэффициент"
        /// </summary>
        public class PermutationDictionary : Dictionary<Permutation, int>
        {
                /// <summary>
                /// Разбиение, соответствующее объекту
                /// </summary>
```

```
public List<int> Split = new List<int>();
/// <summary>
/// Текстовое представление
/// </summary>
public string Text
{
	get
	{
		return string.Join(" + ", this.Select(kvp =>
		{
			if (kvp.Value < 1)
			{
				return "";
			}
			else if (kvp.Value == 1)
			{
				return kvp.Key.Text;
			}
			else
			{
				return kvp.Value + "*" + kvp.Key.Text;
			}
		}));
	}
}
/// <summary>
/// Добавляет перестановку в словарь
/// </summary>
/// <param name="p">Перестановка</param>
public new void Add(Permutation p, int count = 1)
{
	if (ContainsKey(p))
	{
		this[p] += count;
	}
	else
	{
		base.Add(p, count);
	}
}
/// <summary>
/// Добавляет несколько перестановок в словарь
/// </summary>
/// <param name="p">Коллекция перестановок</param>
public void Add(IEnumerable<Permutation> p)
{
	foreach (var i in p)
	{
		Add(i);
	}
}
/// <summary>
/// Печатает словарь перестановок
/// </summary>
public void Print(Output output = Output.Console, string path = "", string text = "")
{
	if (text == "")
	{
		text = Text;
	}
	if (output == Output.Console)
	{
		Console.WriteLine(text);
	}
	else if (output == Output.File)
	{
		using (var fs = File.AppendText(path))
		{
			fs.Write(text);
			fs.WriteLine();
		}
	}
}
```

```
		/// <summary>
		/// Печатаем только тип перестановки и его количество
		/// </summary>
		public void SimplyPrint(Output output = Output.Console, string path = "", string addText =
"")
		{
			var dictCounts = new Dictionary<Permutation, int>();
			foreach (var kvp in this)
			{
				var flagInc = false;
				foreach (var kvp2 in dictCounts)
				{
					if (Permutation.Compare(kvp2.Key, kvp.Key))
					{
						dictCounts[kvp2.Key] += kvp.Value;
						flagInc = true;
						break;
					}
				}
				if (!flagInc)
				{
					dictCounts.Add(kvp.Key, kvp.Value);
				}
			}
			var txt = dictCounts.Select(kvp =>
			{
				var cnt = Permutation.GetPermutationsCount(kvp.Key);
				if (kvp.Value % cnt > 0)
				{
					Console.WriteLine("Error. Order = " + kvp.Key.Order + "; Perm = "
+ kvp.Key.Text + "; Count = " + kvp.Value + "; PermCount = " + cnt + "; Reminder = " + (kvp.Value % cnt));
					MainClass.ErrorBeep();
					Console.ReadKey();
				}
				return ((kvp.Value / cnt) != 1 ? (kvp.Value / cnt).ToString() : "") +
(kvp.Key.NotTrivialCycles.Count > 0 ? ("(" + kvp.Key.Text + ")") : "");
			});
			Print(output, path, string.Join(" + ", txt) + addText + "\r\n");
		}
		/// <summary>
		/// Copy the specified PermutationDictionary.
		/// </summary>
		/// <param name="p1">P1.</param>
		public static PermutationDictionary Copy(PermutationDictionary p1)
		{
			var res = new PermutationDictionary();
			foreach (var kvp in p1)
			{
				res.Add(kvp.Key, kvp.Value);
			}
			res.Split.AddRange(p1.Split);
			return res;
		}
		/// <summary>
		/// Clears the empty entries.
		/// </summary>
		public void ClearEmptyEntries()
		{
			var keys = new List<Permutation>();
			foreach (var kvp in this)
			{
				if (kvp.Key.NotTrivialCycles.Count == 0)
				{
					keys.Add(kvp.Key);
				}
			}
			foreach (var key in keys)
			{
				Remove(key);
			}
		}
		/// <summary>
		/// Sets the order.
```

```
		/// </summary>
		/// <param name="newOrder">New order.</param>
		public void SetOrder(int newOrder)
		{
			foreach (var kvp in this)
			{
				kvp.Key.SetOrder(newOrder);
			}
		}
		/// <summary>
		/// Sets the order.
		/// </summary>
		public void SetOrder()
		{
			SetOrder(GetMaxOrder());
		}
		/// <summary>
		/// Gets the max order.
		/// </summary>
		/// <returns>The max order.</returns>
		public int GetMaxOrder()
		{
			return Keys.Max(t => t.Order);
		}
		/// <param name="p1">P1.</param>
		/// <param name="p2">P2.</param>
		public static PermutationDictionary operator *(PermutationDictionary p1,
PermutationDictionary p2)
		{
			var res = new PermutationDictionary();
			if (p1.Count == 0 && p2.Count > 0)
			{
				return Copy(p2);
			}
			else if (p2.Count == 0 && p1.Count > 0)
			{
				return Copy(p1);
			}
			else if (p1.Count == 0 && p2.Count == 0)
			{
				throw new Exception("Две нулевых штуки.");
			}
			foreach (var kvp1 in p1)
			{
				foreach (var kvp2 in p2)
				{
					res.Add(kvp1.Key * kvp2.Key, kvp1.Value * kvp2.Value);
				}
			}
			return res;
		}
	}
/// <summary>
		/// Генерация
		/// </summary>
		/// <param name="degree">Степень</param>
		/// <param name="variablesCount">Количество переменных</param>
		public static void Generate(int degree, int variablesCount, bool newVariant = true)
		{
			Stopwatch sw = new Stopwatch(), sw2 = new Stopwatch(), sw3 = new Stopwatch();
			sw.Start();
			var sigmas = new List<List<PermutationDictionary>>();
			//first variant
			for (int i = 1; i <= degree; i++)
			{
				var y = YJMElement.Generate(variablesCount + 1);
				var e = new ElementarySymmetricPolynomial(variablesCount, i);
				var pd = e.Substitution(y);
				pd.Split.Add(i);
				sigmas.Add(new List<PermutationDictionary> { pd });
			}
			var maxOrder = sigmas.Max(kvp => kvp.Max(kvp2 => kvp2.GetMaxOrder()));
			foreach (var item in sigmas)
```

```
                        {
                                foreach (var item2 in item)
                                {
                                        item2.SetOrder(maxOrder);
                                }
                        }
                        sw.Stop();
#if sharp6
                        Console.WriteLine($"First stage ready! {sw.Elapsed} elapsed.");
#else
                        Console.WriteLine("First stage ready! " + sw.Elapsed + " elapsed.");
#endif
                        sw.Restart();
                        sigmas[0][0].SimplyPrint(Output.File, "temp.txt", " = {" + string.Join(", ",
sigmas[0][0].Split) + "}");
#if sharp6
                        Console.WriteLine("{" + string.Join(", ", sigmas[0][0].Split) + "} printed! " +
$"{sw.Elapsed} elapsed.");
#else
                        Console.WriteLine("{" + string.Join(", ", sigmas[0][0].Split) + "} printed! " +
sw.Elapsed + " elapsed.");
#endif
                        for (int i = 2; i <= degree; i++)
                        {
                                var ls = NumberSplits.GenerateSplits(i);
                                foreach (var split in ls)
                                {
                                        if (split.Count > 1)
                                        {
                                                var res = new PermutationDictionary();
                                                //split.Reverse ();
                                                int i1 = 0, i2 = 0, i3 = -1, prevCnt = 0;
                                                var c1 = 0;
                                                sw2.Start();
                                                if (newVariant)
                                                {
                                                        foreach (var sigma in sigmas)
                                                        {
                                                                var c2 = 0;
                                                                foreach (var perm in sigma)
                                                                {
                                                                if (perm.Split.Count <= split.Count)
                                                                        {
                                                                                var cnt = 0;
                                                                                int j = 0;
                                                                        for (; j < perm.Split.Count; j++)
                                                                                {
                                                                        if (split[j] != perm.Split[j])
                                                                                {
                                                                                        break;
                                                                                }
                                                                                cnt++;
                                                                        }
                                                                        if (cnt > prevCnt)
                                                                        {
                                                                                prevCnt = cnt;
                                                                                i1 = c1;
                                                                                i2 = c2;
                                                                                i3 = j;
                                                                        }
                                                                }
                                                                c2++;
                                                        }
                                                        c1++;
                                                }
                                        }
                                        if (i3 >= 0)
                                        {
                                                res = sigmas[i1][i2];
                                        }
                                        if (!newVariant)
                                        {
                                                i3 = 0;
```

```
                                                }
                                                sw2.Stop();
                                                sw3.Start();
                                                for (int j = i3; j < split.Count; j++)
                                                {
                                                        res = res * sigmas[split[j] - 1][0];
                                                }
                                                res.Split.AddRange(split);
                                                sigmas[i - 1].Add(res);
                res.SimplyPrint(Output.File, "temp.txt", " = {" + string.Join(", ", res.Split) + "}");
#if sharp6
                                                Console.WriteLine("{" + string.Join(", ", res.Split) + "}
printed! " + $"{sw.Elapsed} elapsed.");
#else
                                                Console.WriteLine("{" + string.Join(", ", res.Split) + "}
printed! " + sw.Elapsed + " elapsed.");
#endif
                                                sw3.Stop();
                                        }
                                        else
                                        {
                                                sigmas[i - 1][0].SimplyPrint(Output.File, "temp.txt", " =
{" + string.Join(", ", sigmas[i - 1][0].Split) + "}");
#if sharp6
                                                Console.WriteLine("{" + string.Join(", ", sigmas[i -
1][0].Split) + "} printed! " + $"{sw.Elapsed} elapsed.");
#else
                                                Console.WriteLine("{" + string.Join(", ", sigmas[i -
1][0].Split) + "} printed! " + sw.Elapsed + " elapsed.");
#endif
                                        }
                                }
                        }
                        sw.Stop();
#if sharp6
                        Console.WriteLine($"Finding stage: {sw2.Elapsed} elapsed.");
                        Console.WriteLine($"Multiplying stage: {sw3.Elapsed} elapsed.");
                        Console.WriteLine($"Second stage ready! {sw.Elapsed} elapsed.");
#else
                        Console.WriteLine($"Finding stage: " + sw2.Elapsed + " elapsed.");
                        Console.WriteLine($"Multiplying stage: " + sw3.Elapsed + " elapsed.");
                        Console.WriteLine($"Second stage ready! " + sw.Elapsed + " elapsed.");
#endif

                        sw.Restart();
                        sw.Stop();
#if sharp6
                        Console.WriteLine($"Third stage ready! {sw.Elapsed} elapsed.");

#else
                        Console.WriteLine("Third stage ready! " + sw.Elapsed + " elapsed.");
#endif
                }
```

## Приложение В
## (справочное).
## Формулы систем уравнений

$k = 8, n = 4$

$$\begin{cases} x_1 = \sigma_1 \\ x_2 + x_3 = \sigma_2 \\ 3x_2 + 2x_3 + 36 = \sigma_1^2 \\ x_4 + x_5 + x_6 = \sigma_3 \\ 35x_1 + 6x_4 + 4x_5 + 3x_6 = \sigma_2\sigma_1 \\ 120x_1 + 16x_4 + 9x_5 + 6x_6 = \sigma_1^3 \\ x_7 + x_8 + x_9 + x_{10} + x_{11} = \sigma_4 \\ 33x_2 + 34x_3 + 10x_7 + 7x_8 + 6x_9 + \\ +5x_{10} + 4x_{11} = \sigma_3\sigma_1 \\ 100x_2 + 70x_3 + 20x_7 + 12x_8 + 11x_9 + \\ +8 * x10 + 6x_{11} + 546 = \sigma_2^2 \\ 273x_2 + 204x_3 + 50x_7 + 28x_8 + 24x_9 + \\ +17x_{10} + 12x_{11} + 1260 = \sigma_2\sigma_1^2 \\ 783x_2 + 544x_3 + 125x_7 + 64x_8 + 54x_9 + \\ +36x_{10} + 24x_{11} + 4320 = \sigma_1^4 \end{cases} \quad \text{(B.1)}$$

где $x_1$ – перестановка циклического типа (2,1),

$x_2$ – перестановка циклического типа (3,2,1),

$x_3$ – перестановка циклического типа (3,2)(4,1),

$x_4$ – перестановка циклического типа (4,3,2,1),

$x_5$ – перестановка циклического типа (3,2,1)(5,4),

$x_6$ – перестановка циклического типа (2,1)(4,3)(6,5),

$x_7$ – перестановка циклического типа (5,4,3,2,1),

$x_8$ – перестановка циклического типа (4,3,2,1)(6,5),

$x_9$ – перестановка циклического типа (3,2,1)(6,5,4),

$x_{10}$ – перестановка циклического типа (3,2,1)(5,4)(7,6),

$x_{11}$ – перестановка циклического типа (2,1)(4,3)(6,5)(8,7),

$\sigma_i$ – $i$-ый элементарный симметрический многочлен.

$$\{"(2,1)" = \sigma 1,\ "(2,1)(4,3)" = 36 + 3\,\sigma 2 - \sigma 1^2,$$
$$"(2,1)(4,3)(6,5)" = 5\,\sigma 1 + 10\,\sigma 3 - 7\,\sigma 2\,\sigma 1 + 2\,\sigma 1^3,$$
$$"(3,2,1)(6,5,4)" = -18 - 6\,\sigma 2 + 6\,\sigma 1^2 + 8\,\sigma 2\,\sigma 1^2 + 10\,\sigma 4 - 10\,\sigma 3\,\sigma 1 - 3\,\sigma 2^2 - 2\,\sigma 1^4,$$
$$"(3,2,1)(5,4)(7,6)" =$$
$$1296 + 210\,\sigma 2 - 95\,\sigma 1^2 - 43\,\sigma 2\,\sigma 1^2 - 60\,\sigma 4 + 48\,\sigma 3\,\sigma 1 + 24\,\sigma 2^2 + 10\,\sigma 1^4,$$
$$"(2,1)(4,3)(6,5)(8,7)" =$$
$$-534 - 92\,\sigma 2 + 42\,\sigma 1^2 + 22\,\sigma 2\,\sigma 1^2 + 35\,\sigma 4 - 25\,\sigma 3\,\sigma 1 - 13\,\sigma 2^2 - 5\,\sigma 1^4,$$
$$"(4,3,2,1)(6,5)" =$$
$$-1056 + 17\,\sigma 2\,\sigma 1^2 + 66\,\sigma 1^2 + 20\,\sigma 4 - 17\,\sigma 3\,\sigma 1 - 10\,\sigma 2^2 - 4\,\sigma 1^4 - 146\,\sigma 2,$$
$$"(3,2,1)(5,4)" = 10\,\sigma 1 - 12\,\sigma 3 + 10\,\sigma 2\,\sigma 1 - 3\,\sigma 1^3,$$
$$"(5,4,3,2,1)" = 312 - 4\,\sigma 2\,\sigma 1^2 - 19\,\sigma 1^2 - 4\,\sigma 4 + 4\,\sigma 3\,\sigma 1 + 2\,\sigma 2^2 + \sigma 1^4 + 34\,\sigma 2,$$
$$"(4,3,2,1)" = -15\,\sigma 1 + 3\,\sigma 3 - 3\,\sigma 2\,\sigma 1 + \sigma 1^3,\ "(3,2,1)" = -36 - 2\,\sigma 2 + \sigma 1^2\}$$

Рисунок В.1 – Решение системы (В.1)

$k = 10, n = 5$

$$
\begin{cases}
x_1 = \sigma_1 \\
x_2 + x_3 = \sigma_2 \\
3x_2 + 2x_3 + 55 = \sigma_1^2 \\
x_4 + x_5 + x_6 = \sigma_3 \\
54x_1 + 6x_4 + 4x_5 + 3x_6 = \sigma_2\sigma_1 \\
181x_1 + 16x_4 + 9x_5 + 6x_6 = \sigma_1^3 \\
x_7 + x_8 + x_9 + x_{10} + x_{11} = \sigma_4 \\
52x_2 + 53x_3 + 10x_7 + 7x_8 + 6x_9 + \\
+5x_{10} + 4x_{11} = \sigma_3\sigma_1 \\
157x_2 + 108x_3 + 20x_7 + 12x_8 + 11x_9 + \\
+8x_{10} + 6x_{11} = \sigma_2^2 \\
418x_2 + 307x_3 + 50x_7 + 28x_8 + 24x_9 + \\
+17x_{10} + 12x_{11} = \sigma_2\sigma_1^2 \\
1179x_2 + 804x_3 + 125x_7 + 64x_8 + 54x_9 + \\
+36x_{10} + 24x_{11} + 9955 = \sigma_1^4 \\
x_{12} + x_{13} + x_{14} + x_{15} + x_{16} + \\
+x_{17} + x_{18} = \sigma_5 \\
49x_4 + 51x_5 + 52x_{10} + 15x_{12} + 11x_{13} + \\
+9x_{14} + 8x_{15} + 7x_{16} + 6x_{17} + 5x_{18} = \sigma_4\sigma_1 \\
1266\mathrm{x}_1 + 299\mathrm{x}_4 + 211\mathrm{x}_5 + 162\mathrm{x}_6 + 50\mathrm{x}_{12} + \\
+30\mathrm{x}_{13} + 25\mathrm{x}_{14} + 19\mathrm{x}_{15} + 17\mathrm{x}_{16} + 13\mathrm{x}_{17} + \\
+10 * \mathrm{x}_{18} = \sigma_3\sigma_2 \\
2844x_1 + 741x_4 + 544x_5 + 433x_6 + 120 * x_{12} + \\
+70x_{13} + 55x_{14} + 42x_{15} + 36x_{16} + 27x_{17} + \\
+20x_{18} = \sigma_3\sigma_1^2 \\
8034x_1 + 1656x_4 + 1069x_5 + 768x_6 + 225x_{12} + \\
+120x_{13} + 96x_{14} + 68x_{15} + 59x_{16} + 42x_{17} \\
+30x_{18} = \sigma_2^2\sigma_1 \\
21546x_1 + 4274x_4 + 2686x_5 + 1887x_6 + 540x_{12} + \\
+275x_{13} + 214x_{14} + 148x_{15} + 126x_{16} + 87x_{17} + \\
+60x_{18} = \sigma_2\sigma_1^3 \\
60121x_1 + 11168x_4 + 6681x_5 + 4500x_6 + 1296x_{12} + \\
+625x_{13} + 480x_{14} + 320x_{15} + 270x_{16} + 180x_{17} + \\
+120x_{18} = \sigma_1^5
\end{cases}
\tag{B.2}
$$

где $x_1$ – перестановка циклического типа (2,1),

$x_2$ – перестановка циклического типа (3,2,1),

$x_3$ – перестановка циклического типа (3,2)(4,1),

$x_4$ – перестановка циклического типа (4,3,2,1),

$x_5$ – перестановка циклического типа (3,2,1)(5,4),

$x_6$ – перестановка циклического типа (2,1)(4,3)(6,5),

$x_7$ – перестановка циклического типа (5,4,3,2,1),

$x_8$ – перестановка циклического типа (4,3,2,1)(6,5),

$x_9$ – перестановка циклического типа (3,2,1)(6,5,4),

$x_{10}$ – перестановка циклического типа (3,2,1)(5,4)(7,6),

$x_{11}$ – перестановка циклического типа (2,1)(4,3)(6,5)(8,7),

$x_{12}$ – перестановка циклического типа (5,4,3,2,1),

$x_{13}$ – перестановка циклического типа (5,4,3,2,1)(7,6)

$x_{14}$ – перестановка циклического типа (4,3,2,1)(7,6,5)

$x_{15}$ – перестановка циклического типа (4,3,2,1)(6,5)(8,7)

$x_{16}$ – перестановка циклического типа (3,2,1)(6,5,4)(8,7)

$x_{17}$ – перестановка циклического типа (3,2,1)(5,4)(7,6)(9,8)

$x_{18}$ – перестановка циклического типа (2,1)(4,3)(6,5)(8,7)(10,9)

$\sigma_i$ – $i$-ый элементарный симметрический многочлен.

{"(2,1)(4,3)(6,5)"= 16 σ1 + 10 σ3 – 7 σ2 σ1 + 2 σ1^3, "(2,1)(4,3)"= 55 + 3 σ2 – σ1^2,
"(2,1)"= σ1, "(3,2,1)"= –55 – 2 σ2 + σ1^2, "(2,1)(4,3)(6,5)(8,7)(10,9)"= 113568 σ2^2
– 496860 σ1^2 – 75116 σ1 + 1112020 σ2 – 47988 σ3 + 14 σ1^5 – 9654 σ1^3
+ 33724 σ2 σ1 – 283920 σ4 + 227136 σ3 σ1 – 203476 σ2 σ1^2 + 47320 σ1^4 + 126 σ5
– 91 σ4 σ1 – 96 σ3 σ2 + 81 σ3 σ1^2 + 83 σ2^2 σ1 – 75 σ2 σ1^3 + 1067066Q,
"(3,2,1)(6,5,4)(8,7)"= 118560 σ2^2 – 518700 σ1^2 – 77495 σ1 + 1160900 σ2 – 49985 σ3
– 10090 σ1^3 + 35201 σ2 σ1 – 296400 σ4 + 237120 σ3 σ1 – 212420 σ2 σ1^2
+ 49400 σ1^4 + 105 σ5 – 95 σ4 σ1 – 81 σ3 σ2 + 83 σ3 σ1^2 + 78 σ2^2 σ1 – 77 σ2 σ1^3
+ 15 σ1^5 + 11139700, "(3,2,1)(5,4)(7,6)(9,8)"= –274560 σ2^2 + 1201200 σ1^2
+ 180940 σ1 – 2688400 σ2 + 116130 σ3 + 23395 σ1^3 – 81683 σ2 σ1 + 686400 σ4
– 549120 σ3 σ1 + 491920 σ2 σ1^2 – 114400 σ1^4 – 280 σ5 + 220 σ4 σ1 + 224 σ3 σ2
– 198 σ3 σ1^2 – 200 σ2^2 σ1 + 185 σ2 σ1^3 – 35 σ1^5 – 25797200, "(4,3,2,1)(6,5)(8,7)"=

$$106080\sigma 2^2 - 464100\sigma 1^2 - 69580\sigma 1 + 1038700\sigma 2 - 45175\sigma 3 - 9125\sigma 1^3$$
$$+ 31832\sigma 2\,\sigma 1 - 265200\sigma 4 + 212160\sigma 3\,\sigma 1 - 190060\sigma 2\,\sigma 1^2 + 44200\sigma 1^4 + 105\sigma 5$$
$$- 85\sigma 4\,\sigma 1 - 96\sigma 3\,\sigma 2 + 82\sigma 3\,\sigma 1^2 + 86\sigma 2^2\,\sigma 1 - 79\sigma 2\,\sigma 1^3 + 15\sigma 1^5 + 9967100,$$
$$"(4,3,2,1)(7,6,5)" = -3517800 - 37440\sigma 2^2 + 163800\sigma 1^2 + 24345\sigma 1 - 366600\sigma 2$$
$$+ 15825\sigma 3 + 3200\sigma 1^3 - 11155\sigma 2\,\sigma 1 + 93600\sigma 4 - 74880\sigma 3\,\sigma 1 + 67080\sigma 2\,\sigma 1^2$$
$$- 15600\sigma 1^4 - 30\sigma 5 + 30\sigma 4\,\sigma 1 + 24\sigma 3\,\sigma 2 - 27\sigma 3\,\sigma 1^2 - 24\sigma 2^2\,\sigma 1 + 25\sigma 2\,\sigma 1^3$$
$$- 5\sigma 1^5,\ "(5,4,3,2,1)(7,6)" = -3048760 - 32448\sigma 2^2 + 141960\sigma 1^2 + 20718\sigma 1$$
$$- 317720\sigma 2 + 13850\sigma 3 + 2819\sigma 1^3 - 9808\sigma 2\,\sigma 1 + 81120\sigma 4 - 64896\sigma 3\,\sigma 1$$
$$+ 58136\sigma 2\,\sigma 1^2 - 13520\sigma 1^4 - 30\sigma 5 + 26\sigma 4\,\sigma 1 + 30\sigma 3\,\sigma 2 - 26\sigma 3\,\sigma 1^2 - 28\sigma 2^2\,\sigma 1$$
$$+ 26\sigma 2\,\sigma 1^3 - 5\sigma 1^5,\ "(3,2,1)(5,4)(7,6)" =$$
$$2255 + 235\sigma 2 - 105\sigma 1^2 - 43\sigma 2\,\sigma 1^2 - 60\sigma 4 + 48\sigma 3\,\sigma 1 + 24\sigma 2^2 + 10\sigma 1^4,$$
$$"(2,1)(4,3)(6,5)(8,7)" =$$
$$-880 - 95\sigma 2 + 45\sigma 1^2 + 22\sigma 2\,\sigma 1^2 + 35\sigma 4 - 25\sigma 3\,\sigma 1 - 13\sigma 2^2 - 5\sigma 1^4,$$
$$"(4,3,2,1)(6,5)" =$$
$$-2200 - 10\sigma 2^2 + 86\sigma 1^2 - 194\sigma 2 + 20\sigma 4 - 17\sigma 3\,\sigma 1 + 17\sigma 2\,\sigma 1^2 - 4\sigma 1^4,$$
$$"(6,5,4,3,2,1)" = 586300 + 6240\sigma 2^2 - 27300\sigma 1^2 - 3812\sigma 1 + 61100\sigma 2 - 2657\sigma 3$$
$$- 545\sigma 1^3 + 1889\sigma 2\,\sigma 1 - 15600\sigma 4 + 12480\sigma 3\,\sigma 1 - 11180\sigma 2\,\sigma 1^2 + 2600\sigma 1^4$$
$$+ 5\sigma 5 - 5\sigma 4\,\sigma 1 - 5\sigma 3\,\sigma 2 + 5\sigma 3\,\sigma 1^2 + 5\sigma 2^2\,\sigma 1 - 5\sigma 2\,\sigma 1^3 + \sigma 1^5,$$
$$"(3,2,1)(5,4)" = 3\sigma 1 - 12\sigma 3 + 10\sigma 2\,\sigma 1 - 3\sigma 1^3,$$
$$"(5,4,3,2,1)" = 660 + 2\sigma 2^2 - 25\sigma 1^2 + 46\sigma 2 - 4\sigma 4 + 4\sigma 3\,\sigma 1 - 4\sigma 2\,\sigma 1^2 + \sigma 1^4,$$
$$"(4,3,2,1)" = -19\sigma 1 + 3\sigma 3 - 3\sigma 2\,\sigma 1 + \sigma 1^3,$$
$$"(3,2,1)(6,5,4)" = 165 + 8\sigma 2 - \sigma 1^2 + 8\sigma 2\,\sigma 1^2 + 10\sigma 4 - 10\sigma 3\,\sigma 1 - 3\sigma 2^2 - 2\sigma 1^4\}$$

Рисунок В.2 – Решение системы (В.2)

**Приложение Г**
**(обязательное).**
**Графическая часть**

## Приложение Д
## (обязательное).
## Авторская справка

Я, Стерлягов Андрей Александрович, автор выпускной квалификационной работы, сообщаю, что мне известно о персональной ответственности автора за разглашение сведений, подлежащих защите законами РФ о защите объектов интеллектуальной собственности. Одновременно сообщаю, что:

1) при подготовке к защите дипломной работы не использованы источники (документы, отчеты, диссертации, литература и т.п.), имеющие гриф секретности или «Для служебного пользования» ВятГУ или другой организации;

2) данная работа не связана с незавершенными исследованиями или уже с завершенными, но еще официально не разрешенными к опубликованию ВятГУ или другими организациями;

3) данная работа не содержит коммерческую информацию, способную нанести ущерб интеллектуальной собственности ВятГУ или другой организации;

4) данная работа не является результатом НИР или ОКР, выполняемой по договору с организацией;

5) в предлагаемом к опубликованию тексте нет данных по незащищенным объектам интеллектуальной собственности других авторов;

6) согласен на использование результатов своей работы ВятГУ для учебного процесса;

7) использование моей дипломной работы в научных исследованиях оформляется в соответствии с законодательством РФ о защите интеллектуальной собственности.

«__»_____ 2017 г. Подпись автора _____

Сведения по авторской справке подтверждаю

«__»_____ 2017 г. Зав. кафедрой _____

## Приложение Е
## (обязательное).
## Библиографический список

1. Каргаполов М.И., Мерзляков Ю.И. Основы теории групп. 3-е издание-е изд. Москва: Наука, 1982. 288 с.

2. Кэртис Ч., Райнер И. Теория представлений конечных групп и ассоциативных алгебр. Москва: Наука, 1969. 668 с.

3. Young A. The Collected Papers of Alfred Young (1873–1940). Toronto, Ont., Buffalo, NY.: University of Toronto Press, 1977. 714 pp.

4. Джеймс Г. Теория представлений симметрических групп: Пер. с англ. Москва: Мир, 1982. 216 с.

5. Vershik A., Okunkov A. A new approach to representation theory of symmetric groups // Selecta Math, No. 4, 1996. pp. 581–605.

6. Пушкарев И.А. К теории представлений сплетений конечных групп с симметрическими группами // Зап. научн. сем. ПОМИ, № 240, 1997. С. 229-244.

7. Пушкарев И.А. Дистрибутивные решётки в теории представлений локальных алгебр и комбинаторике фибоначчиевых рабиений. Диссертация на соискание учёной степени кандидата физико-математических наук. СПбГУ, Санкт-Петербург. 1997. 158 с.

8. Вершик А.М., Окуньков А.Ю. Новый подход к теории представлений симметрических групп // Зап. науч. сем. ПОМИ, Т. 307, 2004. С. 57-98.

9. Пушкарев И.А., Стерлягов А.А. Программное обеспечение исследования свойств YJM-элементов групповых алгебр 2017. С. 6.

10. ЫCeccherini-Silberstein T., Scarabotti F., and Tolli F. Representation Theory of the Symmetric Groups. New York: Cambridge University Press, 2010.

412 pp.