**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**МЕХАНИКО-МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ**

**Кафедра геометрии, топологии и методики преподавания математики**

Паньков

Игорь Петрович

**ГРУППОВЫЕ МЕТОДЫ В ТЕОРИИ ПОЛИМИНО**

Дипломная работа

Научный руководитель:

доктор физ.-мат. наук

профессор С. М. Агеев

Допущен к защите

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2017 г.

Заведующий кафедрой геометрии, топологии и методики преподавания математики

Доктор физ.-мат. наук, профессор В. И. Янчевский

Минск, 2017

Оглавление

[Аннотация 3](#_Toc452029091)

[Перечень условных обозначений 5](#_Toc452029092)

[Введение 6](#_Toc452029093)

[Глава 1. Групповые методы в теории полимино](#_Toc452029094)

[1.1. Основные сведения из теории групп 9](#_Toc452029095)

[1.2. Свободная группа от двух переменных и её связь с множеством путей](#_Toc452029096)

[на клетчатой бумаге 1](#_Toc452029096)1

[1.3. Регулярные полимино 1](#_Toc452029097)5

[Глава 2. Программная реализация](#_Toc452029099)

[2.1. Постановка задачи 19](#_Toc452029100)

[2.2. Разработка программы 19](#_Toc452029101)

[2.3. Краткое описание методов 2](#_Toc452029102)0

[2.4. Детальное описание методов 21](#_Toc452029103)

[2.4. Программная реализация и изображение классов регулярных](#_Toc452029103)

[полимино 23](#_Toc452029103)

[Заключение](#_Toc452029104) 37

[Список использованной литературы](#_Toc452029105) 38

# **Реферат**

Дипломная работа содержит:

- 38 страниц,

- 6 иллюстраций,

- 6 использованных источников,

Ключевые слова: ПОЛИМИНО, РЕГУЛЯРНОЕ ПОЛИМИНО, ТЕОРИЯ ГРУПП, СВОБОДНАЯ ГРУППА, ГРУППА ПУТЕЙ, ГРУППА СЛОВ, ПРЕДСТАВЛЕНИЕ ГРУПП ПУТЕЙ.

В данной дипломной работе рассмотрены групповые методы в теории полимино. В первой главе рассматриваются основные термины теории групп. Далее вводятся свободные группы от двух переменных и устанавливается их связь с группами путей. Затем на основании свободных групп вводится понятие регулярного полимино.

Во второй главе ставится задача разработки программы и представлена программная реализация на языке java. Далее получены различные изображения регулярных полимино.

**Abstract**

Thesis includes:

- 38 pages,

- 6 illustrations,

- 6 sources,

Keywords: POLYOMINO, REGULAR POLYOMINO, GROUP THEORY, FREE GROUP, GROUPS OF PATHS, GROUPS OF WAYS, GROUP OF PATHS REPRESENTATIONS.

In this thesis work group methods of polyomino theory are considered. The first section represents basic definitions of group theory. Further, free groups for two variables are introduced and relations with groups of paths are established. Then, it’s defined the definition of regular polyomino by the virtue of free group.

In the second section program development task is set and program implementation is represented. Further, different pictures of regular polyomino are represented.

**Перечень условных обозначений**

— множество слов в алфавите.

— симметричная группа степени n.

— отношение эквивалентности.

— знак окончания доказательства.

[n] — ссылка на используемую литературу под номером n.

**Введение**

Полимино (-мино) — это плоские геометрические фигуры, образованные путем соединения нескольких одноклеточных квадратов по их сторонам. В зависимости от того, из какого количества квадратов полимино состоит, оно может называться по-разному: мономино ( = 1), домино ( = 2), тримино ( = 3), тетрамино ( = 4) и так далее (рис. 1).

Рис. 1. Все возможные полимино, состоящие не больше чем из четырех квадратов

Рис. 1

Классическая задача в теории полимино связана с так называемым *замощением* плоскости. Замощение плоскости (или фигуры) — представление плоскости в виде множества непересекающихся полимино. Пусть — конечный набор полимино. Область мы будем называть замощаемой набором , если, выбрав какое-то число экземпляров каждого из входящих в набор полимино, сможем составить из этих экземпляров рассматриваемую область . При этом подразумевается, что все используемые полимино имеют один и тот же масштаб, то есть составлены из квадратов одного размера, однако их можно как угодно поворачивать, параллельно переносить, переворачивать (зеркально отображать). Число используемых экземпляров каждого из входящих в набор полимино может быть различным; для отдельных полимино оно может равняться нулю, что означает отказ от использования именно этого полимино; если область бесконечна, то и число экземпляров хоть одного из полимино должно быть бесконечным.

В качестве простого (по формулировке) примера можно рассмотреть задачу об уголках, которая имеет следующую формулировку “При каком можно сложить прямоугольник из уголков и оставшегося числа плиток?” (и уголки, и плитки содержат ровно три клетки).

Если привлечь к решению методы “красочной” комбинаторики [1], то можно прийти к выводу, что при укладка невозможна, а при все три угла должны быть одинаково ориентированными. После этого задача об уголках превращается в такую задачу о трёх уголках: “Показать, что нельзя сложить из прямоугольник из трёх уголков одного вида, скажем вида , и оставшегося числа плиток”.

Эта задача обсуждалась в одном из выпусков журнала “Квант” [2], где использовался метод перебора, и показывалось, что она имеет решение для прямоугольника, имеющего ширину . С увеличением сложность существенно возрастает (поэтому даже компьютер мало чем может помочь) и может показаться, что задача превращается в одну из не берущихся задач комбинаторики. Но, тем не менее, задача о трёх уголках имеет красивое и изящное решение [3], где идея состоит в изучении направленных путей, соединяющих узлы сетки. Это так называемый метод регулярных полимино, который рассмотрим подробнее. Ориентированные пути проходят по рёбрам клетчатой бумаги. На множестве путей имеется несколько алгебраических операций. Для любого пути можно перейти к обратному , получающегося заменой направления пути на обратный. Существует единичный элемент, который состоит из одного узла. Если конец пути совпадает с началом пути , тогда можно определить операцию *соединения,* при которой получится путь , состоящий в последовательном прохождении путей и . Назовём это множество с введёнными выше операциями *группой путей*. Операция соединения определена не для всех путей, поэтому группа путей не является группой. Попробуем перейти от неё к *группе слов*, кодируя направленные пути словами. Пути длиной 1 с направлением вправо, вверх, влево и вниз отождествим с соответственно. Тогда любой направленный путь будет некоторым словом в алфавите из 4 букв. Если фиксировать начало пути, то можно совершить взаимно однозначное соответствие между путём и словом. Обратное слово получается прочтением слова слева направо, при этом меняя показатели на противоположные числа. В качестве единичного слова берётся слово без букв, то есть пустое слово. А операция умножения во множестве слов — это простое приписывание одного слова другому. Множество с такими операциями соответствует группе. Придавая образующим различные значения из другой группы (), мы будем получать для каждого слова некоторый элемент свободной группы, порождённой образующими (). Будем называть это отображение представлением группы путей в Полимино — регулярное при данном представлении, если слово, соответствующее , отображается в .Те полимино, которые получают в соответствие единицу — регулярные. Тем самым получаются представления групп путей в конкретной группе. При конкретном представлении получается список регулярных полимино, то есть регулярные полимино зависят от выбора представления.

Будем рассматривать всевозможные ориентированные, односвязные полимино. Это множество можно задавать в зависимости от количества одноклеточных квадратов в одном полимино. Оказывается, что уже при количество полимино равно 31967, при — 117486, а при — уже более 11 миллионов. Поэтому более целесообразно изучать более узкие классы, такие как регулярные полимино.

Таким образом, цель в данной дипломной работе состоит в нахождении классов регулярных полимино в зависимости от выбора представлений групп путей. Для реализации этого разработаны алгоритмы генерации полимино в зависимости от их длины, выделения списка регулярных полимино и их графического изображения. Предполагалось нахождения закономерности геометрических форм регулярных полимино в зависимости от представления.

**Глава 1. Групповые методы в теории полимино**

* 1. **Основные сведения из теории групп.**

**Определение 1.1.** *Множество вместе и бинарной операцией называют полугруппой, если эта операция ассоциативна, то есть:*

, ;

**Определение 1.2.** *Полугруппа называется группой, если:*

1. *Операция гарантирует единицу, то есть в существует такой элемент , называемый единицей, такой, что ;*
2. *Операция гарантирует обратные элементы, то есть существует так называемый обратный элемент , такой, что: ;*

Будем считать определения *гомоморфизма групп*, *подгрупп*, *сопряжённых элементов* известными.

**Определение 1.3.** *Пусть задана некоторая группа . Определим на ней отношение , полагая , если элементы сопряжены в . Легко проверяется, что это отношение эквивалентности, то есть оно:*

1. *рефлексивно ();*
2. *симметрично ();*
3. *транзитивно ().*

Поэтому разбивается на непересекающиеся классы сопряжённых элементов.

**Определение 1.4.** *Множество всех тех элементов , для которых образуют центр группы .*

Будем пользоваться обозначениями для любых подмножеств .

В частности, могут возникнуть произведения вида:

, где

**Определение 1.5.** *Пусть — множество, — совокупность всех взаимно-однозначных отображений на себя. Если в качестве умножения на взята последовательное выполнение отображений, то — группа. При эта группа становится группой всех подстановок -й степени. В этом случае она называется симметричной группой -й степени и обозначается .*

Если , то для любое взаимно-однозначного отображение на себя можно рассматривать как элемент , если считать, что на оно действует тождественно. То есть существует естественное вложение . Поэтому, в частности, существует инъекция .

Известно, что существует гомоморфизм :

так что , если — чётная подстановка, и , если —нечётная подстановка.

При этом выполняются два свойства:

1. ;
2. Если , то ;

Рассмотрим цепочку естественных вложений:

;

В силу свойства 1 на группе корректно определён гомоморфизм:

;

формулой: если , то ;

Для гомоморфизма будет выполняться усиленное свойство 2.

**Предложение 1. 1.**

1. *Для любых элементов , результат сопряжения посредством лежит в , то есть:;*
2. *Для любых элементов , найдётся такой элемент , такой что: и ;*

**Доказательство.**

1. Пусть . Выберем натуральное число из условия, что ; Тогда для и, следовательно, . Окончательно имеем , то есть .
2. В качестве выберем следующий элемент :

; Тогда в случае имеем . Так как , , то . Если , то , . Так как , то

* 1. **Свободная группа от двух переменных и её связь**

**с множеством путей на клетчатой бумаге.**

Пусть — множество слов в алфавите . Во множестве естественным образом определено умножение: произведение двух слов образуется через простое приписывание одного слова другому. Конечно, введённое умножение ассоциативно и пустое слово является и левой, и правой единицей; следовательно, множество является полугруппой.

Однако полугруппа не является группой, так как только один элемент из имеет обратный и этот элемент .

Для того, чтобы образовать группу, собираем слова в классы эквивалентности. Будем считать слово вида () элементарно эквивалентным слову (), где ; ;

Кратко это будем записывать так:

;

**Определение 1.5.** *Слова и называются эквивалентными (), если существует конечная последовательность такая, что*

;

Легко проверить, что отношение является отношением эквивалентности. Таким образом, множество разбивается на классы эквивалентности.

Обозначим через класс эквивалентности, который представляет слово . Следовательно, равенство обозначает то же самое, что и . Множество классов эквивалентности обозначается .

Легко вывести, что если , то .

Другими словами, множество получает умножение, имеющееся в :

;

Ассоциативность полученного умножения в следует непосредственно из ассоциативности умножения в . Класс эквивалентности единицы является сразу же и левой, и правой единицей. Следовательно, получает так же структуру полугруппы. Однако, каждый элемент в имеет также и обратный.

Обратный элемент для класса порождается словом , которое получается из слова написанием всех букв в обратном порядке и изменением знака показателя у каждого из них на противоположный.

Например

,

то .

Это показывает, что полугруппа на самом деле является группой, которая называется *свободной*.

Нужно отметить наиболее общее свойство группы :

*“Для любой группы и для любых её двух элементов существует единственных гомоморфизм”:*

*, такой что .*

Теперь нужно указать связь между группой и множеством всех путей на клетчатой бумаге.

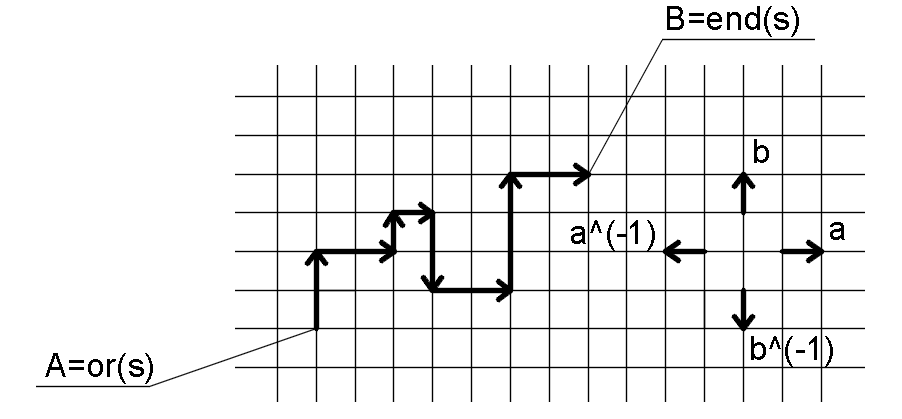
 Рассмотрим клетчатую бумагу. Будем считать геометрически ясным понятие направленного пути на клетчатой бумаге. Любой направленный путь имеет начало , конец , направление обхода на клетчатой бумаге.

Рис. 2

На рисунке 2 изображены 4 элементарных направленных пути. С их помощью можно любому пути сопоставить слово в алфавите .

Для того нужно в процессе обхода пути s от начала к концу последовательно выписывать встречающиеся элементарные пути.

Например

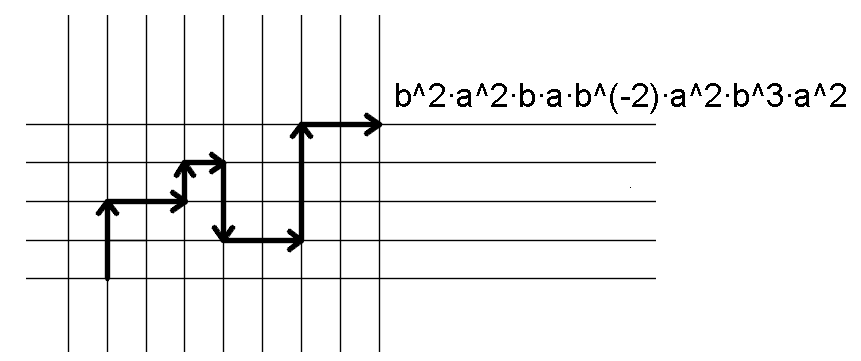


Рис. 3

Соответствующее отображение множества всех направленных путей на клетчатой бумаге в полугруппу обозначим через :

*.*

Верно и обратное: если имеется слово и имеется начало на клетчатой бумаге, то существует единственный направленный путь с началом в точке такой, что :

Полученное отображение обозначим через :

:

Из определения и ясно, что справедливо следующее:

**Предложение 1.2.**

1. *eсть такой результат параллельного переноса пути , что ;*
2. *.*

Пусть заданы и из . Тогда есть путь, параллельный пути с началом в точке . Последовательное прохождение пути и порождает новый путь, обозначаемый через

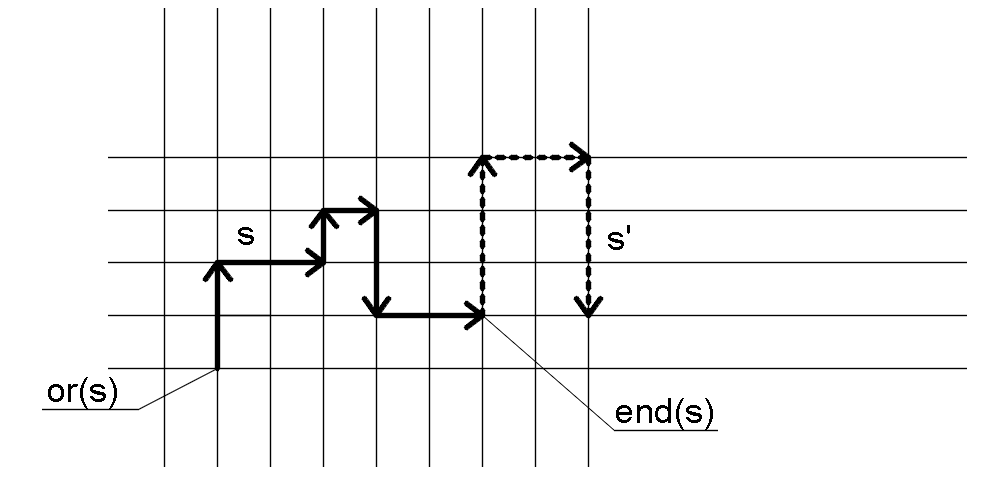


Рис. 4

Путь, полученный из обращением направления обхода обозначим через .

**Предложение 1.3.**

На множестве введём следующее обозначение эквивалентности.

**Определение 1.6.** *, если . Множество классов эквивалентности обозначим через .*

**Определение 1.7.** *Если и — классы эквивалентности путей, то положим: , .*

Доказательство очевидно.

**Предложение 1.4.**  *является группой относительно введённых операций, а отображение , заданное формулой — изоморфизм групп.*

* 1. **Регулярные полимино.**

Мы будем рассматривать полимино — фигуры, составленные из одноклеточных квадратов так, чтобы каждый квадрат примыкал хотя бы к одному соседнему, делящему с ним общую сторону (шахматная ладья может пройти все “поля” такого полимино).

**Определение 1.8.** *Полимино — связный и односвязный многоугольник на клетчатой бумаге.*

Пусть имеется полимино и выбран узел , лежащий на контуре полимино .

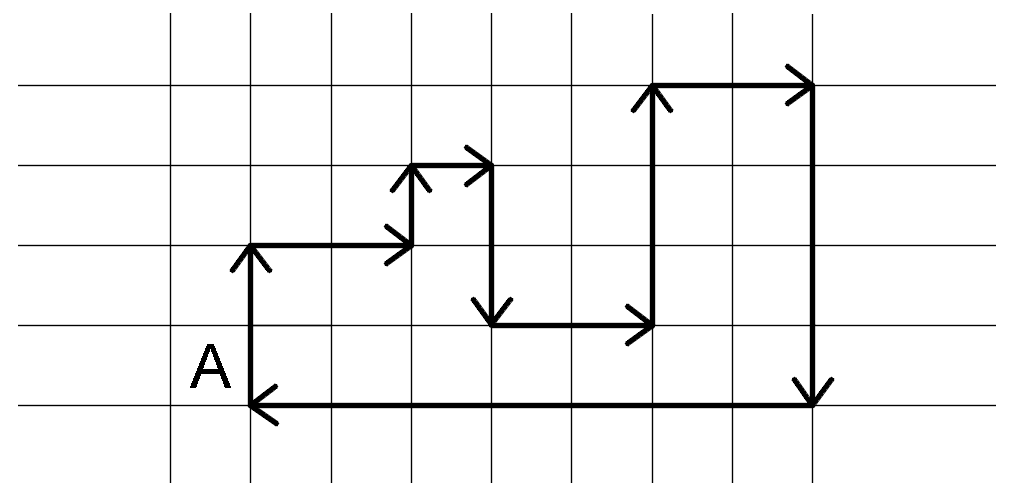


Рис. 5

Двигаясь из начальной точки вдоль контура по часовой стрелке, мы получим замкнутый направленный путь .

Пусть также задана группа , в которой выбраны два элемента . Так как группа свободна, то существует единственный гомоморфизм , такой что . Сопоставим полимино с выбранным начальным узлом элемент группы , обозначаемый через .

Такое сопоставление обладает рядом замечательных свойств, делающих его удобным в различных приложениях.

**Предложение 1.5*.*** *Пусть — другой узел на контуре полимино . — часть контура , идущего по часовой стрелке от к . Тогда:*

**Доказательство.** Основывается на формуле:

Применяя гомоморфизм к обеим её частям, получаем требуемое.

Предложение 5 показывает, что и лежат в одном классе сопряжённых элементов группы . Поэтому каждое полимино определяет единственный класс элементов, сопряжённых с . Его мы обозначим через .

**Определение 1.9.** *Полимино является результатом сложения двух полимино и , если пересечение P с Q не содержит клеток бумаги, а .*

Кратко это записывается так: .

**Предложение 1.6.** *Если , то*

**Доказательство.** Пусть и — два узла на контуре , которые одновременно лежат в и .

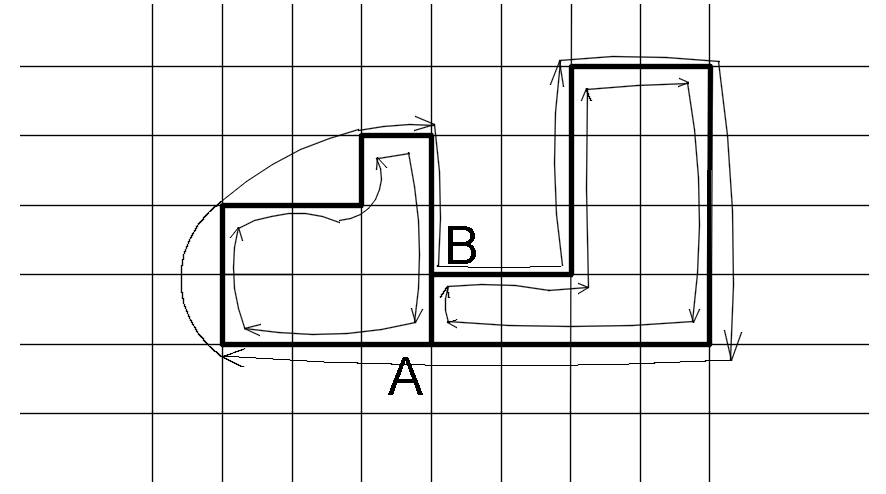


Рис. 6

Тогда и, следовательно

Так как — гомоморфизм, то:

и

Для любого . Поэтому:

Последнее предложение мотивирует такое.

**Определение 2.0.** *Полимино называют:*

1. *Квазирегулярным, если для некоторого узла на контуре*
2. *Регулярным, если для некоторого узла на контуре .*

Всякое регулярное полимино является квазирегулярным. Из предложения 5 следует, что квазирегулярного полимино , как только и лежат на его контуре. Поэтому, можно писать кратко: .

**Теорема 1.1. (теорема суммы**). *Если , а — квазирегулярные (регулярные) полимино, то — также квазирегулярное (регулярное) полимино.*

**Теорема 1.2.** (**теорема разности).** *Если , а — квазирегулярные (регулярные) полимино, то — также квазирегулярное (регулярное) полимино.*

**Доказательство.** Применяя формулу предложения 6, так же замечая, что — подгруппа, получаем требуемое.

Из вышеизложенной теоремы следует такой наглядно-геометрический факт: из любого числа регулярных полимино нельзя сложить нерегулярное.

**Глава 2. Программная реализация.**

**2.1. Постановка задачи.**

Среди всего множества полимино будет рассматривать те, которые обладают следующими свойствами:

1. *Связности:* каждая клетка фигуры соседствует хотя бы с одной другой клеткой (соседствует — значит имеет с соседней клеткой общую сторону; одной лишь общей вершины недостаточно для соседства).
2. *Односвязности:* Не допускаются пустые области, ограниченные со всех сторон закрашенными клетками

Если , фигура называют мономино, при  —домино, при  — тримино, при  — тетрамино, при  — пентамино, при  — гексамино, и т. д. В случае -клеточной фигуры мы будем говорить о -полимино.

Главная цель — получение графических изображений различных классов регулярных полимино в зависимости от представлений групп путей и слов, которые описывают путь соответствующего полимино ()

**2.2. Разработка программы.**

Программа реализована на объектно-ориентированном языке java. Определим несколько терминов из данной области. В объектно-ориентированном программировании *класс —* это абстрактная именованная сущность, имеющая программную реализацию, которая задаёт множество *объектов* с одинаковыми свойствами и атрибутами, но которые могут принимать различные значения. *Объект* — конкретный экземпляр класса с заданными значениями свойств и атрибутов. В моей программе центральным классом является класс *Polyomino,* главным атрибутом которого является строковый тип данных *sequence,* задающий образующие пути в виде последовательности символов для направлений вверх, вниз, вправо и влево соответственно. *Метод/функция/процедура* — именованный программный алгоритм внутри класса, принимающий некоторые значения, который реализует конкретное свойство общее для всех объектов данного класса. *Конструктор* — метод, название которого совпадает с названием класса, предназначенный для создания конкретных экземпляров класса (то есть объектов) на основании принимаемых значений атрибутов. Таким образом, в программе предусмотрены методы для создания новых фигур, их канонизации, вырастания клетки на стороне с заданным номером, для создания различных внешних текстовых представлений фигуры (в виде кодирующей строки, SVG-документа).

* 1. **Краткое описание методов.**

Конструктор **Polyomino(string sequence) —** создаёт новый объект класса на основе кодирующей строки sequence.

Метод **canonize(string sequence) —** канонизирует фигуру, заменяя её кодирующую строку на канонический представитель класса, к которому эта строка принадлежит.

Метод **offshut(string sequence, int n)** — создаёт новое -e полимино, полученное добавлением клетки на -ой стороне объекта Polyomino. Если клетки добавить невозможно (контур нового -го полимино оказывается не простым), возвращает значение пустое значение (null).

Метод **toString()** — возвращает кодирующую строку фигуры.

Метод **boundingBox(string sequence)** — возвращает четырёхэлементный массив с координатами левого нижнего и правого верхнего углов фигуры (начальная точка располагается в начале координат).

Метод **toSVG(string sequence)** — возвращает строку, содержащую SVG-документ с изображением фигуры.

Метод **subpath(int k, int l)** — Возвращает строку, кодирующую часть границы фигуры, начинающуюся с k-й вершины и длиной l.

Метод **perimeter(string sequence)** — возвращает периметр фигуры, то есть длину её кодирующей строки.

Метод **isSimple(string sequence)** — возвращает истинное значение, если фигура правильная (её контур прост), и ложное в противном случае.

* 1. **Детальное описание методов.**

Метод **canonize(string sequence)**.

При программировании метода canonize в переменную str отправляется кодирующая строка sequence. Такое же первоначальное значение получит и переменная minStr в которую в конце концов попадёт канонический представитель класса эквивалентности кодирующей строки. Затем будем в цикле подвергать строку преобразованиям, отражающим всевозможные повороты и всевозможные способы выбора начальной точки контура. Среди этих строк найдём минимальную (лексикографически) и отправим её в переменную minStr, которую сразу после выхода из цикла сделаем кодирующей строкой полимино. Будем искать минимальную строку по индукции, по мере появления новых строк из класса эквивалентности.

Получать эти строки будем в двойном цикле. Внешний цикл будет перебирать разные способы выбора начальной точки контура, переставляя один символ из начала кодирующей строки в её конец.

Во внутреннем цикле кодирующая строка будет подвергаться поворотам на 90° против часовой стрелки, то есть замене символов ‘R’,’U’, ‘L’, ‘D’ на ‘U’, ‘L’, ‘D’, ‘R’ соответственно.

Получившаяся строка str сравнивается со строкой minStr, и, если окажется меньше, становится на место последней.

Метод **offshut(int n)**.

Параметр  метода offshut содержит номер стороны фигуры. Сначала создаётся новый объект Polyomino с той же кодирующей строкой, что и данный объект. Новый объект отправляется в переменную polyomino.

Затем извлекается -й символ из кодирующей строки этого объекта и заменяется на соответствующую трёхсимвольную строку.

Но на новой фигуре могут появиться “разрезы” глубиной один или два. Их устранение проводится в один или два этапа по одному и тому же принципу: устранение сочетаний ‘’, ‘’, ‘’, ‘’ в кодирующей строке.

После устранения “разрезов” может появиться ещё одна проблема: наложение новой клетки на другие клетки полимино, или касание с другими клетками — потеря односвязности. Это может быть справедливо для фигур, которые содержат 7 или больше клеток. Признаком всего этого является наличие самопересечений контура, то есть наличие двух или более совпадающих вершин. Проверку простоты контура (отсутствия самопересечений) поручим методу isSimple(string str). Самопересечения после проверки устраняются, а если контроль качества пройден, новая фигура канонизируется и возвращается из конструктора.

Метод **subPath(int k, int l)**.

Извлечение части контура с заданной вершины с заданной длиной, будет осуществлять метод subpath, а проверку всевозможных частей — на метод isSimple. При обнаружении первой же петли последний возвращает ложное значение, а поиск других петель больше не производится. Если все отрезки контура перебраны, а петель среди них не обнаружено, метод isSimple возвращает истинное значение.

Метод subpath принимает два параметра — номер вершины k, с которой начинается отрезок контура, и длину отрезка l. Cоответствующий этому отрезку кусок кодирующей строки возвращается выражением нахождения подстроки substr(str, k, l).

Метод **isSimple(string sequence)**.

Теперь нужно перебрать все отрезки контура (с началами во всех вершинах и всеми допустимыми длинами) и проверять, не является ли хотя бы один из них петлёй.

В соответствии с проблемой утраты односвязности сделаем перебор участков границы, которые могут быть петлями. Первая найденная петля делает ненужным дальнейший перебор.

Метод **perimeter(string sequence)**.

Возвращает периметр заданного полимино.

* 1. **Программная реализация и изображения классов регулярных полимино**

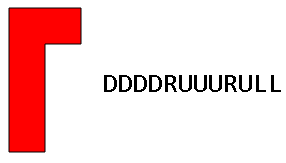
**package** by.bsu;  
  
**import** org.apache.commons.lang.text.StrBuilder;  
**import** org.jfree.graphics2d.svg.SVGGraphics2D;  
  
**import** java.awt.geom.Path2D;**public class** Polyomino {  
  
 **private** String **sequence**;  
  
 **public** Polyomino(String sequence) {  
 **this**.**sequence** = sequence;  
 }  
  
 **public** String canonize(String str\_1) {  
 String str\_2 = str\_1;  
 String minStr = str\_1;  
 **for** (**int** i = 0; i < **sequence**.length(); i++) {  
 str\_2 = str\_2.substring(1).concat(str\_2.substring(0, 1));  
 **for** (**int** j = 0; j < 4; j++) {  
 *replaceChars*(str\_2, **"RULD"**, **"ULDR"**);  
 **if** (minStr.compareTo(str\_2) > 0) {  
 minStr = str\_2;  
 }  
 }  
 }  
 **return** minStr;  
 }  
  
 **public** String offShut(**int** n) {  
 Polyomino poly = **new** Polyomino(**sequence**);  
 StringBuilder str\_1 = **new** StringBuilder(**sequence**);  
 **char** temp = poly.getSequence().substring(n, n + 1).charAt(0);  
 **switch**(temp) {  
 **case 'R'**: str\_1.replace(n, n + 1, **"DRU"**); **break**;  
 **case 'L'**: str\_1.replace(n, n + 1, **"ULD"**); **break**;  
 **case 'U'**: str\_1.replace(n, n + 1, **"RUL"**); **break**;  
 **case 'D'**: str\_1.replace(n, n + 1, **"LDR"**); **break**;  
 }  
 String str\_2 = str\_1.toString();  
 **for** (**int** i = 0; i < 2; i++) {  
 str\_2 = str\_2.replaceAll(**"RL|LR|DU|UD"**, **""**);  
 }  
 **for** (**int** i = 0; i < 2; i++) {  
 str\_2 = str\_2.replaceAll(**"^R(.\*)L$/$1"**, **"$1"**);  
 }  
 **for** (**int** i = 0; i < 2; i++) {  
 str\_2 = str\_2.replaceAll(**"^L(.\*)R$/$1"**, **"$1"**);  
 }  
 **for** (**int** i = 0; i < 2; i++) {  
 str\_2 = str\_2.replaceAll(**"^U(.\*)D$/$1"**, **"$1"**);  
 }  
 **for** (**int** i = 0; i < 2; i++) {  
 str\_2 = str\_2.replaceAll(**"^D(.\*)U$/$1"**, **"$1"**);  
 }  
 **if** (!(isSimple(str\_2)))  
 str\_2 = **null**;  
 **if** (str\_2 != **null**)  
 canonize(str\_2);  
 **return** str\_2;  
 }  
  
 **public** String subPath(String str\_1, **int** vertex, **int** length) {  
 String str\_2 = str\_1 + str\_1;  
 **return** str\_2.substring(vertex, length);  
 }  
  
 **public boolean** isSimple(String str) {  
 **int** perimeter = str.length();  
 String subPath;  
 **for** (**int** i = 1; i < perimeter; i++) {  
 **for** (**int** j = 4; j < perimeter - 12; j += 2) {  
 subPath = subPath(str, i, j);  
 **if** ((subPath = *replaceChars*(subPath, **"R"**, **""**)).equals(  
 subPath = *replaceChars*(subPath, **"L"**, **""**)) &&  
 (subPath = *replaceChars*(subPath, **"U"**, **""**)).equals(  
 (subPath = *replaceChars*(subPath, **"D"**, **""**))))  
 **return false**;  
 }  
 }  
 **return true**;  
 }  
  
 **public int**[] boundingBox(String str) {  
 **int**[] terms = {0, 0, 0, 0};  
 **int**[] coordinates = {0, 0};  
 **for** (Character ch : str.toCharArray()) {  
 **if** (ch.equals(**'R'**)) {  
 coordinates[0]++;  
 }  
 **else if** (ch.equals(**'U'**)) {  
 coordinates[1]++;  
 }  
 **else if** (ch.equals(**'L'**)) {  
 coordinates[0]--;  
 }  
 **else if** (ch.equals(**'D'**)) {  
 coordinates[1]--;  
 }  
 **if** (terms[0] > coordinates[0])  
 terms[0] = coordinates[0];  
 **if** (terms[1] > coordinates[1])  
 terms[1] = coordinates[1];  
 **if** (terms[2] < coordinates[0])  
 terms[2] = coordinates[0];  
 **if** (terms[3] < coordinates[1])  
 terms[3] = coordinates[1];  
 }  
 **return** terms;  
 }  
  
 **public** String toSVG(String str) {  
 **int**[] terms = boundingBox(str);  
 **double**[] viewBox = {terms[0] - .5, -terms[3] - .5, terms[2] - terms[0] + 1, terms[3] - terms[2] + 1};  
 **int** cellSize = 36;  
 **for** (**int** i = 0; i < viewBox.**length**; i++) {  
 viewBox[i] \*= cellSize;  
 }  
 SVGGraphics2D svg = **new** SVGGraphics2D((**int**) viewBox[1], (**int**) viewBox[2]);  
 Path2D path = **new** Path2D.Float();  
 path.reset();  
 path.moveTo(0, 0);  
 **double**[] coordinates = {0, 0};  
 **for** (Character ch : str.toCharArray()) {  
 **if** (ch.equals(**'R'**)) {  
 coordinates[0] += cellSize;  
 path.lineTo(coordinates[0], coordinates[1]);  
 } **else if** (ch.equals(**'U'**)) {  
 coordinates[1] -= cellSize;  
 path.lineTo(coordinates[0], coordinates[1]);  
 } **else if** (ch.equals(**'L'**)) {  
 coordinates[0] -= cellSize;  
 path.lineTo(coordinates[0], coordinates[1]);  
 } **else if** (ch.equals(**'D'**)) {  
 coordinates[1] += cellSize;  
 path.lineTo(coordinates[0], coordinates[1]);  
 }  
 }  
 svg.draw(path);  
 String doc;  
 **return** doc = svg.getSVGDocument();  
 }  
  
 **public** String getSequence() {  
 **return sequence**;  
 }  
  
 **public static** String replaceChars(String str, String searchChars, String replaceChars) {  
 **if**(!*isEmpty*(str) && !*isEmpty*(searchChars)) {  
 **if**(replaceChars == **null**) {  
 replaceChars = **""**;  
 }  
 **boolean** modified = **false**;  
 **int** replaceCharsLength = replaceChars.length();  
 **int** strLength = str.length();  
 StrBuilder buf = **new** StrBuilder(strLength);  
 **for**(**int** i = 0; i < strLength; ++i) {  
 **char** ch = str.charAt(i);  
 **int** index = searchChars.indexOf(ch);  
 **if**(index >= 0) {  
 modified = **true**;  
 **if**(index < replaceCharsLength) {  
 buf.append(replaceChars.charAt(index));  
 }  
 } **else** {  
 buf.append(ch);  
 }  
 }  
 **if**(modified) {  
 **return** buf.toString();  
 } **else** {  
 **return** str;  
 }  
 } **else** {  
 **return** str;  
 }  
 }  
  
 **public static boolean** isEmpty(String str) {  
 **return** str == **null** || str.length() == 0;  
 }  
}

**package** by.bsu;  
  
**import** java.util.ArrayList;  
**import** java.util.Arrays;  
**import** java.util.HashMap;  
**public class** Permutations {  
 **private int n** = 4;  
 Integer[] **a** = {2, 0, 1, 3};  
 Integer[] **b** = {3, 0, 1, 2};  
 Integer[] **c** = reverse(**a**);  
 Integer[] **d** = reverse(**b**);  
  
 **public** Integer[] initialize(Integer a[]) {  
 **for** (**int** j = 0; j < **n**; j++)  
 a[j] = j;  
 **return** a;  
 }  
  
 **public** Integer[] swap(Integer a[]) {  
 **for** (**int** i = 0; i < **n**; i++) {  
 **int** r = (**int**) (Math.*random*() \* (i + 1));  
 **int** swap = a[r];  
 a[r] = a[i];  
 a[i] = swap;  
 }  
 **return** a;  
 }  
  
 **public int** getN() {  
 **return n**;  
 }  
  
 **public** Integer[] getC() {  
 **return c**;  
 }  
  
 **public** Integer[] getD() {  
 **return d**;  
 }  
  
 **public** Integer[] reverse(Integer a[]) {  
 Integer[] rev = **new** Integer[**n**];  
 **for**(**int** i = 0; i < **n**; i++) {  
 **for**(**int** j = 0; j < **n**; j++) {  
 **if**(a[j] == i) {  
 rev[i] = j;  
 }  
 }  
 }  
 **return** rev;  
 }  
  
 **public** Integer[][] group(String str) {  
 **char**[] ch = str.toCharArray();  
 ArrayList<Integer[]> arr\_1 = **new** ArrayList<>();  
 arr\_1.add(0, **a**);  
 arr\_1.add(1, **b**);  
 arr\_1.add(2, **c**);  
 arr\_1.add(3, **d**);  
 Integer[][] arr\_2 = **new** Integer[ch.**length**][**n**];  
 **for** (**int** i = 0; i < ch.**length**; i++) {  
 **if**(ch[i] == **'D'**) {  
 arr\_2[i] = arr\_1.get(0);  
 } **else if** (ch[i] == **'R'**) {  
 arr\_2[i] = arr\_1.get(1);  
 } **else if** (ch[i] == **'U'**) {  
 arr\_2[i] = arr\_1.get(2);  
 } **else if** (ch[i] == **'L'**) {  
 arr\_2[i] = arr\_1.get(3);  
 }  
 }  
 **return** arr\_2;  
 }  
  
 **public** String checkMultiply(String str) {  
 **char**[] ch = str.toCharArray();  
 Integer[][] arr = group(str);  
 Integer[] e = initialize(**new** Integer[**n**]);  
 **for** (**int** i = 0; i < ch.**length** - 1; i++) {  
 **int** j = i + 1;  
 Integer[] res = **new** Integer[**n**];  
 **for** (**int** k = 0; k < **n**; k++) {  
 res[k] = arr[j][arr[i][k]];  
 }  
 **for** (**int** l = 0; l < **n**; l++) {  
 arr[j][l] = res[l];  
 }  
 }  
 **for** (**int** m = 0; m < **n**; m++) {  
 **if** (arr[ch.**length** - 1][m] != e[m]) {  
 **return null**;  
 }  
 }  
 **return** str;  
 }

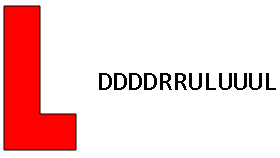
}

# 





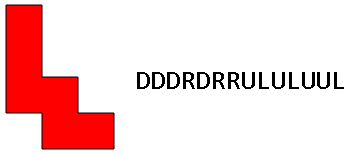
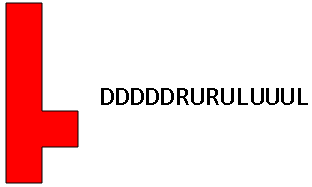


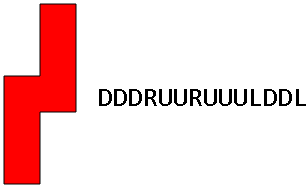


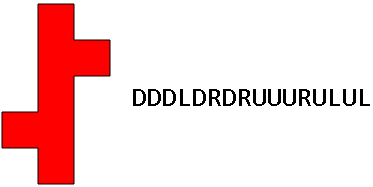
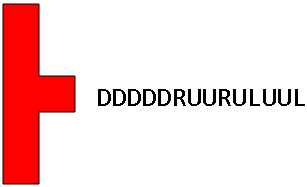








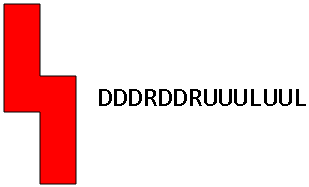




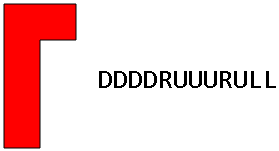












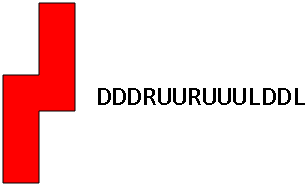






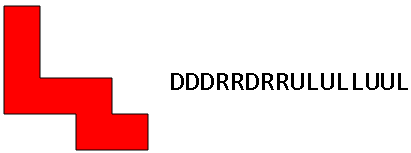


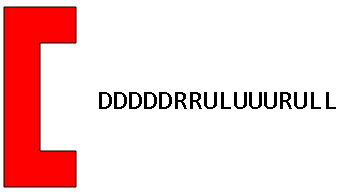


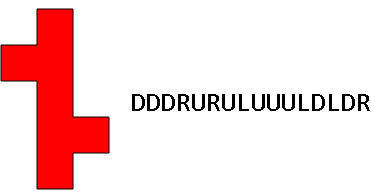












**Заключение**

Присваивая различные значения элементам свободной группы из , получаются различные представления групп путей и, соответственно, классы регулярных полимино. Получая графические изображения элементов этих классов, устанавливается некоторая взаимосвязь между алгеброй и геометрией.

**Список использованной литературы**

1. Кордемский Б.А. Красочная комбинаторика — М.: Наука, 1973 (журнал “Квант” в. №9).
2. Бартенев Ф., Савин А. Метод перебора — М.: Наука, 1982 (журнал “Квант” в. №2).

3. Агеев С.М. Регулярные полимино — М.: Наука, 1979 (журнал “Квант” в. №11).

4. Голомб С. Полимино — M.: Мир, 1975.

5. Александров П.С. Введение в теорию групп — M.: Наука, 1980.

6. Магнус В., Каррас А., Солитэр Д. Комбинаторная теория групп — М.: Наука, 1974.