МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ   
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ   
«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт математики и информационных систем

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ И ФИЗИКО-МАТЕМАТИЧЕСКИХ НАУК

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Допущена к защите

Заведующий кафедрой ПМИ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.В. Разова

**ЧИСЛА ГУРВИЦА**

Курсовая работа по дисциплине  
«Дискретная математика»

Выполнил студент группы ПМИб-2301-52-00     / А. О. Хохолков /

Руководитель к.ф.-м.н., доцент кафедры ПМИ     / В. А. Бызов/

Работа защищена с оценкой     \_\_\_\_.\_\_\_\_.2023 г.

Члены комиссии:     /     /

    /     /

Киров 2023

Содержание

[Введение 3](#_Toc125234187)

[1. Предварительные теоретические сведения 4](#_Toc125234188)

[1.1. Код Прюфера 4](#_Toc125234189)

[1.2. Теорема Кэли 6](#_Toc125234190)

[1.3. Графы и перестановки 7](#_Toc125234191)

[1.4. Теорема о биекции между деревьями и длинными циклами 8](#_Toc125234192)

[1.5. Минимальное число транспозиций в получении длинного цикла и теорема о количестве транспозиций в 10](#_Toc125234193)

[1.6. Следствие из теоремы Кэли 10](#_Toc125234194)

[1.7. Вывод по 1 разделу 11](#_Toc125234195)

[2. Числа Гурвица 12](#_Toc125234196)

[2.1. Понятие числа Гурвица 12](#_Toc125234197)

[2.2. Определение числа Гурвица 12](#_Toc125234198)

[2.3. Методы вычисления чисел Гурвица 14](#_Toc125234199)

[2.4. Решение упражнений 17](#_Toc125234200)

[2.5. Вывод по разделу 2 18](#_Toc125234201)

[3. Программная реализация 19](#_Toc125234202)

[3.1. Реализация программы 19](#_Toc125234203)

[3.2. Результат работы программы 25](#_Toc125234204)

[3.3. Вывод по разделу 3 26](#_Toc125234205)

[Заключение 27](#_Toc125234206)

[Библиографический список 28](#_Toc125234207)

[Приложения 29](#_Toc125234208)

[Приложение А. Листинг программы 29](#_Toc125234209)

# Введение

Числа Гурвица были введены А. Гурвицем в конце 19 века.

Числа Гурвица применяют для перечисления классов графов, они являются коэффициентами связи в симметрических группах, представляют собой инварианты Громова- Виттена и т. д. Таким образом, числа Гурвица являются актуальными. (см в [8]).

Если по-простому, то число Гурвица - это число способов разложить перестановку размера в произведение транспозиций, деленное на , при больших вычисление чисел Гурвица становиться задачей, которую трудно решить вручную. Таким образом актуальной задачей является разработка программы для вычисления чисел Гурвица.

Целью данной работы является программная реализация методов вычисления чисел Гурвица, и их оценка.

Для достижения этой цели необходимо решить следующие задачи:

1. Изучение теории по числам Гурвица,
2. Изучение методов нахождения чисел Гурвица,
3. Программная реализация этих методов и их оценка.

# Предварительные теоретические сведения

Теорема Кэли необходима для рассмотрения последующей теории по теме чисел Гурвица. Рассмотрим Код Прюфера, из которого получим формулу Кэли.

## Код Прюфера

**Определение1.1.** Код Прюфера - это сопоставление дереву на n вершинах кода длиной из символов алфавита с возможными повторами символов. (см [6]).

Например, код: , который соответствует дереву



Рисунок 1.1 – Дерево на 5 вершинах

Метод построения кода:

Пусть Т - дерево на n номерованных вершинах, – алфавит.

Код Прюфера строится путем последовательного удаления висячей вершины с наименьшим номером из дерева Т и приписывания в конец слова номера связанной с ней вершины. Эти операции производятся пока не останется две вершины.

Рассмотрим пример построения кода Прюфера по дереву на 6 вершинах.

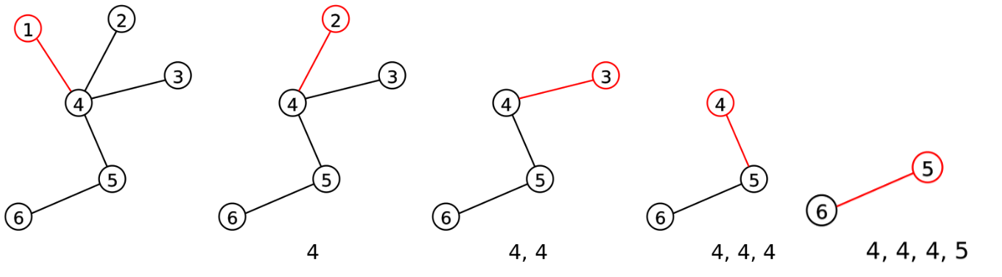


Рисунок 1.2- Построение кода Прюфера

На рисунке 1.2 видно, что сперва удаляется концевая вершина № (1), затем в конец слова заносится символ, соответствующий вершине № (4), связанной с вершиной № (1).

Остается еще 5 вершин: алгоритм продолжается.

Все то же самое повторяется со следующими вершинами № (2), (3).

Когда мы доходим до вершины № (4), мы удаляем вершину № (4) и записываем в конец слова символ, соответствующий вершине № (5), связанной с вершиной № (4). На данном этапе остается 2 вершины – алгоритм завершен, код Прюфера для дерева с рисунка 1.2 завершен. Мы получили код: .

Построение дерева по коду Прюфера:

**Определение1.2.**

Пусть - код Прюфера, а – список номеров вершин, – список выбранных для построения вершин из .

Строится дерево последовательным выбором вершины с наименьшим номером из ( – это некоторый элемент из , где – это не номер элемента из , а – обозначает номер из выбранных вершин) не встречающейся в коде Прюфера, построением ребра и удалением и из и соответственно. Это повторяется пока не закончится код Прюфера.

В конце код Прюфера заканчивается и остается добавить ребро   
.

Рассмотрим пример построения дерева по коду Прюфера из 4 символов.

Код: .

Номера вершин:

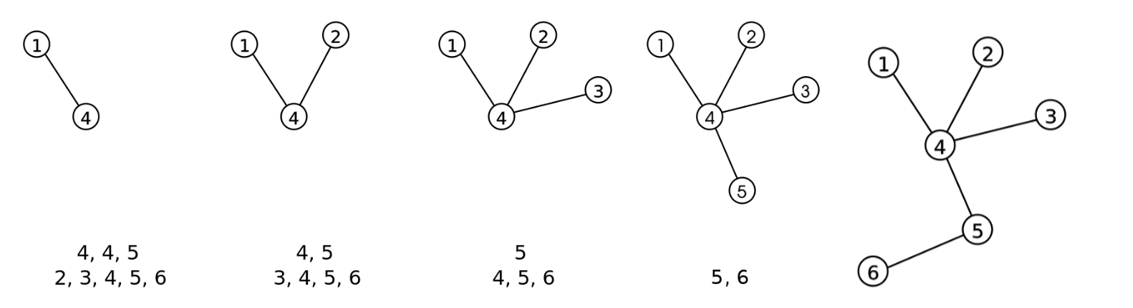


Рисунок 1.3 – Построение дерева.

На рисунке 1.3 видно, что сначала берется вершина № (1) не встречающаяся в коде, и вершина № (4) стоящая на первом месте в коде, строится ребро (1,4), после чего берется следующая минимальная вершина, не встречающаяся в коде, это № (2), и строится ребро , потом соответственно получаем ребро и . Остались не использованы вершины из списка вершин, соответственно строим последнее ребро .

Получение формулы Кэли:

Вопрос: сколько существует слов(деревьев) из символов алфавита из n символов (на n номерованных вершинах) и длиной ?

Так как на любое место мы можем поставить любую вершину, количество слов(деревьев.

Таким образом, мы получили формулу Кэли, что позволяет нам плавно перейти к ее рассмотрению, рассматривая код Прюфера как ее доказательство.

## Теорема Кэли

**Определение1.3**. Число различных деревьев на нумерованных n вершинах равно . (см [1], [6])

**Пример1.1.**

Случай с 1 и 2 вершинами проиллюстрирован рисунком 1.4:



Рисунок 1.4 – Дерево на 1 и 2 вершинах.

Для 3 вершин получаем 3 дерева:



Рисунок1.5 – Дерево на 3 вершинах.

Для 4 вершин по формуле 16:

Как видно для деревьев из 3 вершин, количество деревьев вида “цепочки” зависит от числа не крайних вершин, а их соответственно и деревьев

тогда для 4 вершин таких деревьев существует 12.

Остаются деревья вида:



Рисунок 1.5 – Дерево на 4 вершинах 2 типа.

Их количество зависит от центральной вершины, соответственно их 4.  
Итого, деревьев на 4 вершинах – 16, что соответствует формуле Кэли.

## Графы и перестановки

Графы на n помеченных вершинах и группа перестановок на n элементов тесно связаны. Это связано с тем, что каждое ребро определяет транспозицию, которая меняет местами вершины ребра. Так произведение набора транспозиций в определенном порядке определяет перестановку, которая несет в себе информацию о графе.

Так рассмотрим граф на нумерованных вершинах и произвольно занумерованными ребрами. Каждому ребру такого графа можно поставить в соответствие транспозицию - элемент из группы перестановок .Поскольку произведение транспозиций не коммутативно, нам важно чтобы ребра были пронумерованы, при этом порядок нумерации вершин не влияет на результат.

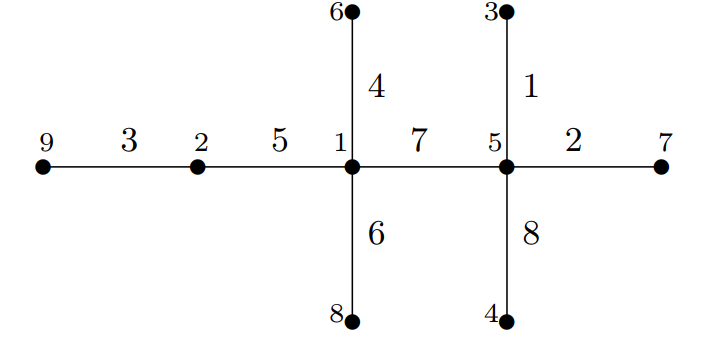


Рисунок 1.6 – Дерево с занумерованными вершинами и ребрами

Рассмотрим рисунок 1.6, на этом рисунке представлен граф на 9 вершинах с занумерованными вершинами и ребрами.

Произведение транспозиций данного графа в порядке нумерации ребер:

Данное произведение образует перестановку с длинным длины 9 в группе перестановок :

Перемножение перестановок производится справа налево.

## Теорема о биекции между деревьями и длинными циклами

**Теорема.**   
При описанном выше соответствии, дереву на n вершинах с произвольной нумерацией ребер и вершин соответствует длинный цикл в группе перестановок , который получается путем перемножения транспозиций, соответствующих ребрам дерева , в порядке их нумерации.

Замечание: Так как перенумерация элементов множества действует на любой перестановке сопряжением, не меняя циклический тип перестановки(см [4]), то данное утверждение не зависит от порядка нумерации вершин дерева.

Докажем данную теорему:

Пускай σ произвольная перестановка в , – произвольная транспозиция в , где , принадлежат набору вершин .  
Рассмотрим разложение в произведение независимых циклов, их количество зависит от того входят ли , в один цикл или в разные:

Если , принадлежат одному циклу, то при расщепляет на два цикла.

Если же , принадлежат разным циклам, то при склеивает их в один цикл.  
Рассмотрим подграф дерева , состоящий из всех вершин и первых ребер дерева.

При представлении перестановки , соответствующей данному графу, в виде произведения независимых циклов, циклы в данном разложении взаимно однозначно соответствуют компонентам связности графа . Добавление последующих ребер с номерами (), приводит к склеиванию данных циклов (соответствующих компонентам связности, которые соединяются этими ребрами) в один длинный цикл, объединению компонент связности в одну.

**Пример1.2.** Рассмотрим процесс соединения циклов для дерева на рисунке 1.6. Сначала вершины дерева образуют циклы длины 1(тождественную перестановку). При умножении на две вершины 3 и 5 образуют цикл длины 2: (3,5). При умножении на к этим вершинам добавляется вершина 7, образуя цикл длины 3. При умножении на появляется новый цикл длины два: (2,9). При умножении на появляется еще один цикл: (1,6). При умножении на циклы и склеиваются образуя один цикл длины 4. При умножении на к последнему циклу добавляется вершина 8, образуя цикл длины 5. Умножение на склеивает циклы длин 3 и 5, образуя цикл длины 8. А умножение на добавляет вершину 4, образуя длинный цикл длины 9 в .

## Минимальное число транспозиций в получении длинного цикла и теорема о количестве транспозиций в

Так как каждая транспозиция соответствует ребру графа, то если у графа на n вершинах меньше ребра то он несвязный. Поэтому - это минимальное число транспозиций для разложения цикла длины .

**Теорема о количестве транспозиций в .**

Мы хотим понять сколько есть транспозиций для группы перестановок на n вершинах.

Перестановку можно представить в виде таблицы с 2 строками или с одной, имея ввиду что первая строка не пишется для экономии места, но можно записать и в циклическом виде.

Рассмотрим количество транспозиций в . Так как транспозиция меняет два элемента местами, то нам нужно всего лишь выбрать эти два элемента

Очевидно, что на первое место претендуют n элементов, а на второе остается элементов.

Тогда число транспозиций будет равно .

Но мы рассматривали транспозиции в циклической форме записи, а они могут записываться как , а могут как , при этом обозначая одну и ту же транспозицию.

Тогда получаем, что число транспозиций будет в два раза меньше.

Количество транспозиций в группе перестановок равно выражению:

.

## Следствие из теоремы Кэли

Согласно теореме Кэли число деревьев на n нумерованных вершинах равно . Занумеровать ребра можно способами, поэтому это количество способов перемножить транспозиции в так, чтобы они давали длинный цикл. В свою очередь это количество длинных циклов в .

Поэтому каждый длинный цикл в можно представить через произведение транспозиции.

## Вывод по 1 разделу

В данном разделе мы рассмотрели теоретический материал, который нам понадобиться в последующих разделах.

# Числа Гурвица

## Понятие числа Гурвица

Пусть некоторая перестановка в . Ее можно разложить в произведение транспозиций. Пусть это количество транспозиций, через произведение которых мы хотим представить перестановку.  
Тогда количество различных комбинаций этих транспозиций, в произведении дающих перестановку , деленное на , будет числом Гурвица.

Очевидно:

Число таких комбинаций зависит от циклического типа перестановок и одинаково для всех перестановок данного циклического типа.

Есть минимальное значение m для которого такое разложение существует, оно определяется выражением , где это количество независимых циклов в . (по аналогии минимальное число транспозиций для разложения цикла длиной равно )

Все значения m, для которых возможно такое разложение, имеют совпадающую с четностью перестановки четность.

## Определение числа Гурвица

Разбиение можно обозначить как последовательность невозрастающих частей, , где , в которой конечное число частей отлично от нуля, либо в мультипликативной форме записи , где соответствует кратности части в разбиении, соответственно конечное число частей имеют кратность отличную от 0.  
Справедливо равенство:

Пример записи разбиений числа .

Теперь можно сформулировать определение простого числа Гурвица.

**Определение2.1.** Обозначим – число транспозиций в наборе, - разбиение числа .

Тут - множество всех транспозиций в , а –множество всех перестановок с циклическим типом в , например, .

**Определение2.2.** Связное число Гурвица определяется похожим способом, но рассматривается только наборы, порождающие подгруппу , которая действует транзитивно на множестве .

**Определение2.3.** Транзитивность - после применения некоторого набора транспозиций элемент переходит в любой другой, заданный наперед, элемент , где и подгруппа действует на множестве транзитивно, тогда

Важно: Простые несвязные числа Гурвица перечисляют графы с m ребрами и заданным циклическим типом произведения транспозиций, связные простые числа Гурвица перечисляют связные графы.

Выполненное в прошлом разделе рассуждение приводит нас к выводу о количестве деревьев на n вершинах, оно равно , подставляя в формулу получаем:

.

Теперь рассмотрим определение общих чисел Гурвица.

**Определение2.4.** Общее число Гурвица определяется как число наборов из m перестановок данных циклических типов, дающих в произведении тождественную перестановку, деленное на . Связные общие числа Гурвица определяется схожим образом, но, как и с простыми, с требованием транзитивности действия подгруппы перестановок , порожденной перестановки . (см [2], [5])

## Методы вычисления чисел Гурвица

Первым (очевидным) способом вычисления числа Гурвица является перебор.

Определение метода:

1) Нахождение всех возможных транспозиций.

2) Подсчет комбинаций транспозиций, дающих в произведении подходящую перестановку.

3) Подстановка полученного значения в формулу

**Пример2.1.**

Возьмем . Количество транспозиций в равно 3: .

Случай первый: Если мы возьмем произведение транспозиции на себя, мы получим тождественную перестановку. Данный набор нам не подходит.

Случай второй: Если мы возьмем произведение двух различных транспозиций, то, так как в любой паре транспозиций в есть одинаковые элементы, они образуют длинный цикл длины 3, поэтому можно сделать вывод, что подгруппа транспозиций действует транзитивно.

Таким образом, число подходящих наборов равно 6, подставляем в формулу:

Вторым способом является модификация первого:

Первый способ предполагал счет всех возможных произведений транспозиций.

Зная количество различных транспозиций в , мы можем без перебора и перемножения подсчитать количество способов представить длинный цикл в произведение транспозиций.

Возьмем . В любые две различные транспозиции дают в произведении цикл длины 3, получаемая подгруппа совпадает с , поэтому действует транзитивно на .

В существует 3 различных транспозиции, поэтому существует   
 упорядоченных наборов различных транспозиций. Это частный случай для помеченных деревьев .

Стоит отметить то, что если перестановка представляет собой длинный цикл, то получаемая подгруппа, порождаемая транспозициями , действует транзитивно. Поэтому для таких случаев простое и связное числа Гурвица равны.

Теперь попробуем изменить циклический тип этот циклический тип описывает транспозицию. Естественно, что данный тип можно представить в виде одной транспозиции взятой три раза, так же, как и через три разные транспозиции, поэтому количество наборов будет равно   
. Соответственно:

Так же первый способ представления через умножение транспозиции самой на себя не дает транзитивно действующей подгруппы, поэтому:

Теперь возьмем :

Видно, что числа Гурвица могут быть дробными.

Вторым способом подсчета чисел Гурвица является способ с применением ориентированных графов.

Он применяется для подсчета количества комбинаций транспозиций. Вспомним, что может происходить при умножении на транспозицию:

Если элементы транспозиции включены в разные циклы, она их объединяют.

Если элементы транспозиции включены в один цикл, то она его разделяет.

Если элементы транспозиции не включены ни в один цикл, то добавляется цикл длины 2.

Таким образом, мы будем использовать ориентированный граф, у которого в каждую вершину может приходить до 3 ребер, и уходить из этой вершины до 3 ребер.

Рассмотрим пример вычисления числа Гурвица с применением ориентированный граф:

Зададим начальные условия:

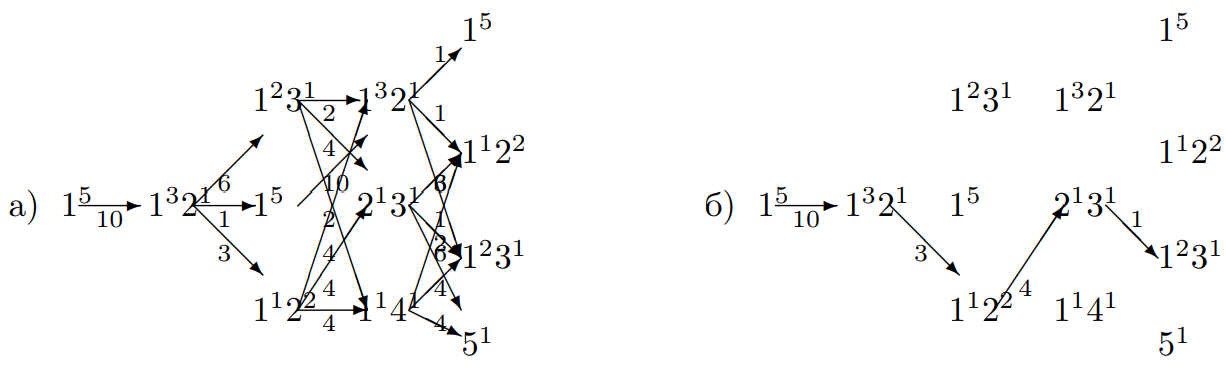


Рисунок 2.1 – ориентированный граф.

Рассмотрим рисунок 2.1, на нем изображен ориентированный граф, позволяющий вычислить данное число Гурвица. Стрелки в этом графе соответствуют умножению на транспозицию, вершины обозначают циклический тип перестановки (разбиение числа ), число над стрелкой – ее кратность, она показывает число транспозиций, умножение на которые приводит к результату на правом конце стрелки.

Важно: из вершины не может выходить стрелки с общей кратностью больше количества различных транспозиций в группе .

Например, стрелка идущая из вершины в вершину будет иметь кратность 6, так как перестановка должна содержать один из “свободных” элементов (их 2) перестановки и элемент входящий в цикл длины 3 (их 3), то есть .

Но нас интересует количество наборов транспозиций. Оно получается путем суммирования кратностей путей от левой вершины до нужной правой. А кратность пути в свою очередь это произведение кратностей ребер этого пути.

Например, кратность выбранного для примера пути на рисунке 2.1 под буквой б равняется:   
.

(см [2], [5]).

## Решение упражнений

Ниже будет приведено решение нескольких задач из [3].

**Задача1.1.** Пусть Γ — связный граф на n вершинах с единственным простым циклом. Докажите, что произведение в всех транспозиций, соответствующих ребрам графа Γ, раскладывается в произведение двух независимых циклов вне зависимости от порядка умножения транспозиций.

**Доказательство:**

Возьмем для доказательства дерево, при произведении транспозиций, соответствующих ребрам дерева, всегда образуется один длинный цикл, при добавлении одного ребра образуется простой цикл; так как наш длинный цикл имеет все элементы из , то умножение на любую транспозицию из , содержащую любые элементы из , приводит к разделению данного цикла на два независимых цикла.

Утверждение доказано.

**Задача1.2.** Приведите пример связного графа с единственным простым циклом, в котором длины двух циклов в произведении всех транспозиций, отвечающих его ребрам, зависят от выбранного порядка умножения.

**Решение.**

В качестве такого графа возьмем граф на 4 вершинах.

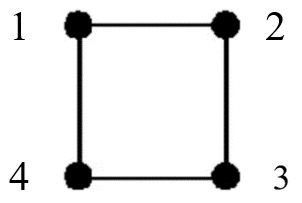


Рисунок 2.2 – граф на 4 вершинах

При умножении транспозиций в порядке: получаем один цикл длины 1 и один длины 3.

При умножении транспозиций в порядке: получаем два цикла длинны 2.

Как видно из выше написанного образовалось 2 цикла разной длины, значит пример имеет место.

## Вывод по разделу 2

В данном разделе мы рассмотрели понятие числа Гурвица, рассмотрели примеры нахождения чисел Гурвица, разные методы вычисления чисел Гурвица.

# Программная реализация

Реализовывать будем алгоритм вычисления простого числа Гурвица с полным перебором. Для того, чтобы программная реализация представляла большую ценность, реализуем деление на потоки (см [7]). Язык реализации c++ (см [3]).

## Реализация программы

**На вход** алгоритма подается – число транспозиций в наборе, - разбиение числа .

Ввод данных производиться в функции input:

cout << "m, M + 0\n\"0\" = end of enter\n";//вывод формы ввода

cin >> m;//ввод количества транспозиций

short n = 1;//просто временная переменная для ввода циклического типа

while (n > 0)//пока не встретили отрицательное число

{

cin >> n;//ввод основания

if (n > 0)//обрботка нечетного ввода

{

M[0].push\_back(n);//добавление в конец вектора основания

cin >> n;//ввод степени

M[1].push\_back(n);//добавление степени в конец вектора

M[2].push\_back(0);//инициализация счетчика

}

}

Далее, после того как произвели ввод, создаем переменную и записываем в нее время начала работы алгоритма (используем для этого библиотеку chrono). Далее необходимо проверить, что данные введены корректно, этим занимается функция chek.

В случае если данные введены не корректно программа завершит свою работу с кодом 1 или 2.

short ki = 0;//счетчик количества всех циклов

short chet = 0;//переменная для определения четности

for (int i = 0; i < k; i++)//цикл по вектору циклического типа

{

n += M[0][i] \* M[1][i];//считаем n

ki += M[1][i];//считаем количество циклов

chet += (1 - M[0][i] % 2) \* M[1][i];//определяем четность

}

if (m < n - ki)//условие выхода первое(m<минимального значения)

{

cout << "it`s wrong\n";//вывод ошибки

return 1;//возврат кода ошибки

}

else if (chet % 2 != m % 2)//условие выхода второе (проверка четности)

{

cout << "it`s wrong\n";//вывод ошибки

return 2;//возврат кода ошибки

}

Далее вычисляем количество транспозиций в .

Далее создаем и заполняем вектор, хранящий транспозиции, этим занимается функция intr.

short x = 0;//счетчик транспозиции

for (int i = 0; i < n - 1; i++)//цикл по первым элементам транспозиции

{

for (int j = i + 1; j < n; j++)//цикл по вторым элементам транспозиции

{

for (int k = 0; k < n; k++)//цикл для заполнения тождественной перестановки

{

tr[x][k] = k;//запись биекции i->i

}

tr[x][i] = j;//запись первого элемента транспозиции

tr[x][j] = i;//запись второго элемента транспозиции

x++;//увеличение счетчика транспозиции

}

}

Теперь мы закончили подготовительные вычисления и готовы приступить непосредственно к вычислению числа Гурвица, функция - hurvic.

int x = 0;//счетчик хороших комбинаций транспозиций

int y = multischet(tr, n, m1, x, m, M);//вызов функции деления на потоки и получение числа всех хороших

return (double(y) / fact(n)) ;//вычисление числа гурвица и его вывод

Теперь мы пробуем разделить выполнение алгоритма на потоки, так как дальше начинается большой перебор, для этого мы используем функцию multischet. Processor\_count – число виртуальных потоков процессора, глобальная переменная.

const auto processor\_count = std::thread::hardware\_concurrency(); // получаю число виртуальных потоков системы

short my = m1;//задание верхней границы

short delt = m1 / processor\_count;//высчитывание шага

short mx = delt \* (processor\_count - 1);//задание нижней границы

vector<thread> fx(processor\_count);//создание вектора потоков

vector<short>ii(processor\_count);//создание вектора счетчиков для потоков

vector<vector<short> > key(processor\_count, vector<short>(m));//создание вектора векторов индексов транспозиций

vector<int> xx(processor\_count);//создание вектора счетчиков хороших наборов транспозиций

for (short i1 = 0; i1 < processor\_count; i1++) //цикл в котором мы инициализируем потоки

{

for (int i = 0; i < m; i++)//обнуление вектора разрядов

key[i1][i] = -1;

ii[i1] = 0;//инициализация счетчика разрядов

xx[i1] = 0;//инициализация счетчика хороших наборов

//инициализация потока и передача в него лямбда функции с вызовом функции schet

fx[i1] = thread([&tr, &n, &ii, i1, m1, mx, my, &xx, &m, M, &key]() {schet(tr, n, ii, i1, m1, mx, my, xx, m, M, key); });

my = mx; //смена верхней границы

mx -= delt;// смена нижней границы

}

for (int i = 0; i < processor\_count; i++)

{

fx[i].join();//ждем потоки

}

for (int i = 0; i < processor\_count; i++)

{

x += xx[i];//суммирование всех хороших комбинаций транспозиций

}

return x;//возврат числа всех хороших транспозиций

В этой функции мы выделяем свою копию данных для каждого потока, так как синхронизировать обращение потоков к одним и тем же данным не целесообразно, потому что данных немного, и они почти не занимают места. Так же в конце мы суммируем полученные результаты. В алгоритме используется int счетчики, потому что short переполняется в одном из потоков, так же в счете результата тоже используется int, потому что его хватает для хранения необходимых чисел.

Самой сложной оказалась реализация функции schet, потому что нужно было корректно перебирать m1- мерные числа m разрядов в заданных диапазонах.

bool bl = 0;//флаг для определения критериев верхней границы

for (int j = 0; j < i[i1]; j++)//проверка более высоких разрядов на приближение к границе

if (key[i1][j] < my)bl = 1; //если есть хотя бы один разряд не превышающий границы

short my1 = ((bl) ? m1 - 1 : (i[i1] == m - 1) ? my - 1 : my);//если есть маленькие разряды,

//то граница = размерности сс числа, иначе если крайний разряд, то число должно быть на еденицу меньше

my1 = (my1 == m1) ? my1 - 1 : my1;//так как разряд не может быть равен разрядности

for (int j = (key[i1][i[i1]] > -1) ? 0 : mx; j <= my1; j++) //сам цикл перебора

{

key[i1][i[i1]] = j; i[i1]++; //записываем число в разряд и увеличиваем счетчик разряда

if (i[i1] == m)//органичение перебора, если все разряды подобраны

{

vector<short> per(n);//создаем вектор под перестановку

per = multi(tr, n, key[i1], m);//перемножаем транспозиции

x[i1] = x[i1] + our(per, n, M);//проверяем наша ли транспозиция и прибавляем к счетчику число

}

else schet(tr, n, i, i1, m1, mx, my, x, m, M, key);//рекурсивный вызов если не подобраны все разряды

i[i1]--;//уменьшаем счетчик разрядов

}

Осталось произвести перемножение полученных наборов транспозиций и определить их циклический тип, сравнив его с данным, и изменить счетчик.

Для перемножения транспозиций используется функция multi.

vector<short> per(n); //вектор под конечную перестановку

//for (int i = 0; i < n; i++) per[i] = i; // делаю перестановку тождественной

for (int i = 0; i < n; i++)//иду по перестановке

{

int z = i;//переменная для перемножения транспозиций

for (int j = 0; j < m; j++) //цикл в котором происходит перемножение

{

z = tr[key[j]][z];//перемножение транспозиций

}

per[i] = z;//записывание результата в перестановку

}

return per;//возврат перестановки

Для определения типа полученной перестановки используется функция our, на ее выходе имеем либо 0, либо 1, их и прибавляем к счетчику в функции schet.

vector<short> key(n);//массив флагов

for (size\_t i = 0; i < n; i++) key[i] = 0;//обнуляю флаги

short ki = 0;//счетчик обхода перестановки

short z;//переменная для индекса в перестановке

while (ki != n)//проверка того, что что мы еще не проверили всю перестановку

{

z = 0;//обнуление счетчика

while (key[z] != 0) z++;//находим не использованный элемент

key[z] = 1;//отмечаю флаг первого на этот раз элемента

short k = 1; //счетчик длинны цикла

short z1 = per[z];//переменная для обхода цикла

while (z != z1)//пока не обошли цикл

{

key[z1] = 1;//отмечаем пройденный элемент

k++;//увеличиваем счетчик длинны цикла

z1 = per[z1];//переходим у следующему элементу

}

ki += k;//к счетчику перестановки прибавляем длину пройденного цикла

z = 0;// обнуляем счетчик для того чтобы пройтись по массиву циклического типа

while (z < M[0].size() && M[0][z] != k)//ищем есть ли такой циклический тип

z++;

if (z == M[0].size()) break;//если не нашли данный тип, выходим из анализа перестановки

if (M[2][z] < M[1][z]) M[2][z]++; //если не превышено количество таких циклов увеличиваем счетчик количества таких циклов

else break;//иначе выходим из анализа

}

short bl = 1;//создаем флаг

if (ki != n) bl = 0;//если вышли из анализа раньше то bad per

for (z = 0; z < M[0].size(); z++)//просто цикл обнуления счетчиков количества циклов

{

M[2][z] = 0;//обнуление элемента

}

return bl;//возврат флага

Для счета факториала используем функцию fact.

int x = 1;//задание переменной для счета

for (int i = 1; i <= n; i++)

{

x \*= i;//счет факториала

}

return x;//возврат полученного значения

В самом конце мы получаем время конца работы программы и считаем разницу.

На вывод программы поступают два значения число Гурвица и время работы программы.

**Листинг** программы приведен в приложении А.

## Результат работы программы

Для замеров времени работы программы используем библиотеку chrono.

Критерии оценки работы программы: входные данные, примерное число комбинаций в переборе, время работы, так как памяти программа почти не занимает, то ее в оценке учитывать не будем.

Тестирования проводились с помощью ноутбука Microsoft surface book 2 на базе intel i7 8650U(1.9 ГГц, до 4.2 ГГц, 16 ГБ ).

Таблица 3.1 – Работа программы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Входные данные | Число комбинаций в переборе | Занимаемый объем памяти | Полученное число | Время работы программы |
| m=3 µ=4^1 | 216 | 880 КБ | 4 | 0.0134163 с |
| m=4 µ=5^1 | 10 000 | 888 КБ | 25 | 0.0191984 с |
| m=5 µ=6^1 | 759 375 | 892 КБ | 216 | 0.267389 с |
| m=6 µ=7^1 | 85 766 121 | 1 МБ | 2401 | 18.2741 с |
| m=7 µ=8^1 | 13 492 928 512 | 1 МБ | 32768 | 2581.69 с |

К минусам программы можно отнести не совсем корректное распределение диапазонов перебора по потокам, так как при делении на потоки остается неохваченный диапазон, он прибавляется к первому потоку. В моем случае один поток занимает 12,5% ресурсов процессора. При тестировании на последнем диапазоне после 30 из 40 минут остается всего один поток. Если этот диапазон равномерно распределять по всем потокам, то можно получить прибавку в скорости работы приблизительно в раза. Так же к минусам программы можно отнести не реализованность некоторых эвристик, которые могли бы ускорить работу программы на подходящих для них входных данных, позволяя пропустить перебор.

## Вывод по разделу 3

Таким образом, в данном разделе мы рассмотрели программную реализацию полного перебора и провели тестирование программы на разных диапазонах входных данных.

# Заключение

Числа Гурвица представляют собой число способов разложить перестановку размера n в произведение транспозиций, деленное на n. Они применяются для перечисления классов графов, являются коэффициентами связи в симметрических группах и т. д.

При работе над курсовой работой были поставлены следующие задачи:

1. Изучение теории по числам Гурвица,
2. Изучение методов нахождения чисел Гурвица,
3. Программная реализация этих методов и их оценка.

Так, в первом разделе были рассмотрены предварительные теоретические сведения, во втором были разобраны сами числа Гурвица и методы их вычисления, в третьем он был программно реализован и протестирован, в ходе тестирования стало понятно, что программная реализация имеет несколько недочетов.

В результате исследования можно сделать вывод, что качество программной реализации во многом зависит от правильности деления на потоки и количества и качества эвристик, используемых в реализации.

Таким образом, все поставленные задачи выполнены, а цель достигнута.

# Библиографический список

1. Айцгнер М.,Циглер Г., Доказательства из книги. Лучшие доказательства со времен Евклида до наших дней: Пер. с англ. // М.:Мир, 2006. 256 с.
2. Бычков Б.С., Ямбург Н.Я., Курс лекций. числа Гурвица // hse.ru : НИУ ВШЭ, 2021. URL: [https://math.hse.ru](https://math.hse.ru/Hurwitz_numbers_by4_spring2021)
3. Документация по языку C++. // microsoft.com : Microsoft. URL: [https://learn.microsoft.com](https://learn.microsoft.com/ru-ru/cpp/cpp/?view=msvc-170)
4. Калужнин Л.А., Сущанский В.И., Преобразования и перестановки. Пер. с укр – 2-е изд., перераб и доп. // М.: Наука. Главная редакция физико-математической литературы, 1985. 160 с.
5. Ландо С.К. Введение в дискретную математику // hse.ru : НИУ ВШЭ. 2012. 121-131 c. URL: [https://math.hse.ru](https://math.hse.ru/data/2012/02/24/1266138841/LandoBook.pdf)
6. Омельченко А.В. Теория графов // М.: МЦНМО, 2018. 416 с.
7. Уильямс Э. Параллельное программирование на C++ в действии. Практика разработки многопоточных программ. Пер. с англ. Слинкин А.А. // М.:ДМК Пресс, 2012. 672 с.
8. Lando S.K., Hurwitz numbers: on the edge between combinatorics and geometry // Proceedings of the International Congress of Mathematicians. Hyderabad, India, 2010. 1-25 с.

# Приложения

## Приложение А. Листинг программы

#include <iostream> // подключаю библиотеку iostream

#include <clocale> // подключаю библиотеку clocale

#include <windows.h> // подключаю библиотеку windwos.h

#include <vector> // подключаю библиотеку vector

#include <ctime> // подключаю библиотеку ctime

#include <thread> // подключаю библиотеку thread

const auto processor\_count = std::thread::hardware\_concurrency(); // получаю число виртуальных потоков системы

using namespace std; //полe имен std

vector<short> multi(vector<vector<short>>& tr, short& n, vector<short>& key, short& m) //функция произведения транспозиций

{

vector<short> per(n); //вектор под конечную перестановку

//for (int i = 0; i < n; i++) per[i] = i; // делаю перестановку тождественной

for (int i = 0; i < n; i++)//иду по перестановке

{

int z = i;//переменная для перемножения транспозиций

for (int j = 0; j < m; j++) //цикл в котором происходит перемножение

{

z = tr[key[j]][z];//перемножение транспозиций

}

per[i] = z;//записывание результата в перестановку

}

return per;//возврат перестановки

}

short our(vector<short>& per, short& n, vector<vector<short>>& M)//функция проверки циклического типа перестановки

{

vector<short> key(n);//массив флагов

for (size\_t i = 0; i < n; i++) key[i] = 0;//обнуляю флаги

short ki = 0;//счетчик обхода перестановки

short z;//переменная для индекса в перестановке

while (ki != n)//проверка того, что что мы еще не проверили всю перестановку

{

z = 0;//обнуление счетчика

while (key[z] != 0) z++;//находим не использованный элемент

key[z] = 1;//отмечаю флаг первого на этот раз элемента

short k = 1; //счетчик длинны цикла

short z1 = per[z];//переменная для обхода цикла

while (z != z1)//пока не обошли цикл

{

key[z1] = 1;//отмечаем пройденный элемент

k++;//увеличиваем счетчик длинны цикла

z1 = per[z1];//переходим у следующему элементу

}

ki += k;//к счетчику перестановки прибавляем длину пройденного цикла

z = 0;// обнуляем счетчик для того чтобы пройтись по массиву циклического типа

while (z < M[0].size() && M[0][z] != k)//ищем есть ли такой циклический тип

z++;

if (z == M[0].size()) break;//если не нашли данный тип, выходим из анализа перестановки

if (M[2][z] < M[1][z]) M[2][z]++; //если не превышено количество таких циклов увеличиваем счетчик количества таких циклов

else break;//иначе выходим из анализа

}

short bl = 1;//создаем флаг

if (ki != n) bl = 0;//если вышли из анализа раньше то bad per

for (z = 0; z < M[0].size(); z++)//просто цикл обнуления счетчиков количества циклов

{

M[2][z] = 0;//обнуление элемента

}

return bl;//возврат флага

}

void schet(vector<vector<short>>& tr, short& n, vector<short>& i, short i1, short m1, short mx, short my, vector<int>& x, short& m, vector<vector<short>> M, vector<vector<short>>& key)

//рекурсивная процедура для перебора комбинаций транспозиций, их перемножения и подсчета верных

{

bool bl = 0;//флаг для определения критериев верхней границы

for (int j = 0; j < i[i1]; j++)//проверка более высоких разрядов на приближение к границе

if (key[i1][j] < my)bl = 1; //если есть хотя бы один разряд не превышающий границы

short my1 = ((bl) ? m1 - 1 : (i[i1] == m - 1) ? my - 1 : my);//если есть маленькие разряды,

//то граница = размерности сс числа, иначе если крайний разряд, то число должно быть на еденицу меньше

my1 = (my1 == m1) ? my1 - 1 : my1;//так как разряд не может быть равен разрядности

for (int j = (key[i1][i[i1]] > -1) ? 0 : mx; j <= my1; j++) //сам цикл перебора

{

key[i1][i[i1]] = j; i[i1]++; //записываем число в разряд и увеличиваем счетчик разряда

if (i[i1] == m)//органичение перебора, если все разряды подобраны

{

vector<short> per(n);//создаем вектор под перестановку

per = multi(tr, n, key[i1], m);//перемножаем транспозиции

x[i1] = x[i1] + our(per, n, M);//проверяем наша ли транспозиция и прибавляем к счетчику число

}

else schet(tr, n, i, i1, m1, mx, my, x, m, M, key);//рекурсивный вызов если не подобраны все разряды

i[i1]--;//уменьшаем счетчик разрядов

}

}

int multischet(vector<vector<short>>& tr, short& n, short& m1, int& x, short& m, vector<vector<short>>& M)

//функция для деления перебора на потоки и суммирование полученных в потоках значений

{

short my = m1;//задание верхней границы

short delt = m1 / processor\_count;//высчитывание шага

short mx = delt \* (processor\_count - 1);//задание нижней границы

vector<thread> fx(processor\_count);//создание вектора потоков

vector<short>ii(processor\_count);//создание вектора счетчиков для потоков

vector<vector<short> > key(processor\_count, vector<short>(m));//создание вектора векторов индексов транспозиций

vector<int> xx(processor\_count);//создание вектора счетчиков хороших наборов транспозиций

for (short i1 = 0; i1 < processor\_count; i1++) //цикл в котором мы инициализируем потоки

{

for (int i = 0; i < m; i++)//обнуление вектора разрядов

key[i1][i] = -1;

ii[i1] = 0;//инициализация счетчика разрядов

xx[i1] = 0;//инициализация счетчика хороших наборов

//инициализация потока и передача в него лямбда функции с вызовом функции schet

fx[i1] = thread([&tr, &n, &ii, i1, m1, mx, my, &xx, &m, M, &key]() {schet(tr, n, ii, i1, m1, mx, my, xx, m, M, key); });

my = mx; //смена верхней границы

mx -= delt;// смена нижней границы

}

for (int i = 0; i < processor\_count; i++)

{

fx[i].join();//ждем потоки

}

for (int i = 0; i < processor\_count; i++)

{

x += xx[i];//суммирование всех хороших комбинаций транспозиций

}

return x;//возврат числа всех хороших транспозиций

}

int fact(short& n)

//функция счета факториала

{

int x = 1;//задание переменной для счета

for (int i = 1; i <= n; i++)

{

x \*= i;//счет факториала

}

return x;//возврат полученного значения

}

void input(short &m,vector<vector<short>> &M)

{

cout << "m, M + 0\n\"0\" = end of enter\n";//вывод формы ввода

cin >> m;//ввод количества транспозиций

short n = 1;//просто временная переменная для ввода циклического типа

while (n > 0)//пока не встретили отрицательное число

{

cin >> n;//ввод основания

if (n > 0)//обрботка нечетного ввода

{

M[0].push\_back(n);//добавление в конец вектора основания

cin >> n;//ввод степени

M[1].push\_back(n);//добавление степени в конец вектора

M[2].push\_back(0);//инициализация счетчика

}

}

}

short chek(short& m, vector<vector<short>>& M,short&n,short k)

{

short ki = 0;//счетчик количества всех циклов

short chet = 0;//переменная для определения четности

for (int i = 0; i < k; i++)//цикл по вектору циклического типа

{

n += M[0][i] \* M[1][i];//считаем n

ki += M[1][i];//считаем количество циклов

chet += (1 - M[0][i] % 2) \* M[1][i];//определяем четность

}

if (m < n - ki)//условие выхода первое(m<минимального значения)

{

cout << "it`s wrong\n";//вывод ошибки

return 1;//возврат кода ошибки

}

else if (chet % 2 != m % 2)//условие выхода второе (проверка четности)

{

cout << "it`s wrong\n";//вывод ошибки

return 2;//возврат кода ошибки

}

}

void intr(vector<vector<short>>& tr, short n)

{

short x = 0;//счетчик транспозиции

for (int i = 0; i < n - 1; i++)//цикл по первым элементам транспозиции

{

for (int j = i + 1; j < n; j++)//цикл по вторым элементам транспозиции

{

for (int k = 0; k < n; k++)//цикл для заполнения тождественной перестановки

{

tr[x][k] = k;//запись биекции i->i

}

tr[x][i] = j;//запись первого элемента транспозиции

tr[x][j] = i;//запись второго элемента транспозиции

x++;//увеличение счетчика транспозиции

}

}

}

double hurvic(vector<vector<short>>& tr, short& n, short& m1, short& m, vector<vector<short>>& M)

{

int x = 0;//счетчик хороших комбинаций транспозиций

int y = multischet(tr, n, m1, x, m, M);//вызов функции деления на потоки и получение числа всех хороших

return (double(y) / fact(n)) ;//вычисление числа гурвица и его вывод

}

int main()

{

short m;//создание переменной хранящей количество транспозиций в произведении

vector<vector<short>> M(3);//создание вектора для хранения циклической структуры

input(m, M);

auto start = chrono::high\_resolution\_clock::now();//получаем время начала

short k = short(size(M[0]));//получение числа разных циклов

short n = 0;

short ch = chek(m, M, n, k);

if (ch) return ch;

short m1 = n \* (n - 1) / 2;//вычисление количества всех транспозиций для данного n

vector<vector<short>> tr(m1, vector<short>(n));//создание вектора для транспозиций

intr(tr,n);

double h= hurvic(tr, n, m1, m, M);

auto end = chrono::high\_resolution\_clock::now();//получаем время конца

chrono::duration<float> duration = end - start;//считаем разницу во времени

cout << h << endl;//вывод числа Гурвица

cout << "time: " << duration.count()<<" s\n";//выводим время в секундах

return 0;//возврат кода

}