МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ПРИКЛАДНОЙ МАТЕМАТИКИ

КАФЕДРА МАТЕМАТИЧЕСКОЙ КИБЕРНЕТИКИ

КУРСОВАЯ РАБОТА

Построение таблицы Кэли группы, заданной образующими и определяющими соотношениями

Студент: Чечина Л.А.

Группа: 8О-104Б

Преподаватель: Яшина Н.П.

Оценка:

Дата: 05.05.2024

Основные понятия и определения по теме работы

Таблица Кэли — это квадратная таблица, которая описывает бинарную операцию в конечной группе. В таблице Кэли строки и столбцы представляют элементы группы, а каждая ячейка на пересечении строки и столбца указывает результат операции между соответствующими элементами.
Построение таблицы Кэли может помочь визуально представить операции группы и выявить некоторые закономерности или свойства, которые могут быть полезны при её анализе.

Группа — это алгебраическая структура, состоящая из множества элементов и операции, удовлетворяющей определенным свойствам.

Группа — это моноид, в котором каждый элемент обратим. G =< M,$*$,e >.

Образующие элементы группы есть подмножество множества элементов, которые могут быть использованы для порождения всех других элементов группы при помощи операции группы.

Определяющие соотношения являются условиями, которые определяют, какие комбинации образующих элементов эквивалентны нейтральному элементу или другим элементам группы.

S — это набор элементов (образующих), из которых можно получить все элементы группы G с помощью операций группы.
Любой элемент группы G можно представить как последовательность (слово) образующих и их обратных.
Последовательное соединение двух слов (строк) из элементов S и их обратных аналогично операции умножения в группе.
Соотношения — это уравнения, которые связывают разные слова, представляющие один и тот же элемент группы. Например, если ab=ba , то слова ab и ba представляют один и тот же элемент группы. Некоторые соотношения можно вывести из других, например, если ab=ba и b=c, то ac=ca.
Для полного описания группы достаточно указать набор ключевых соотношений между образующими.

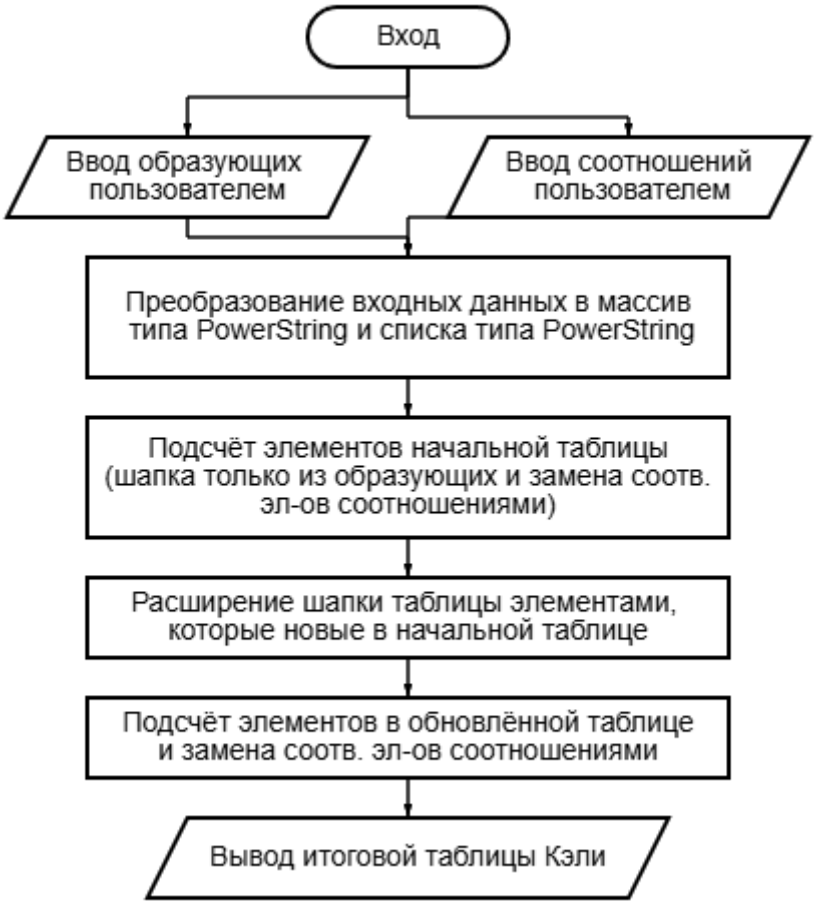
Описание алгоритма

1) Создаём пустую таблицу Кэли размером n x n, где n - количество элементов в группе. Заполняем первую строку и первый столбец таблицы элементами группы.

2) Для каждой ячейки (i, j) таблицы, где i и j - индексы элементов группы: выбираем образующий элемент g и применяем его к элементу, соответствующему индексу i, чтобы получить новый элемент h: h = g * element[i]. Записываем элемент в ячейку (i, j)

3) Преобразовываем полученные значения в соответствии с отношениями. Ищем в таблице Кэли элементы, не встречающиеся раннее и записываем в обновлённую шапку таблицы.

Логическая блок-схема



Описание программы и инструкции по работе с ней

Программа написана на языке Python. Для создания графического интерфейса была использована библиотека PyQt5. Стандартная библиотека sys используется для того, чтобы запускался цикл событий в приложении. Для реализации GUI была выбрана именно библиотека PyQt5, так как при работе с таблицами она аккуратно выводит их и при увеличении количества элементов, таблица сохраняет свою аккуратность и читаемость.

В остальном же построение таблицы реализовано с помощью функций, реализованных самостоятельно.

class PowerString — это класс, который хранит строку и int число: степень, а также имеет несколько методов:

1) get_result(self): Этот метод возвращает строку, которая представляет собой string возведенную в степень power в формате "string^power". Например, если string равно "a" и power равно 2, то get_result() вернет "a^2", т.е возвращает эл-т в эталонном виде.
2) __eq__(self, other): Это специальный метод, который определяет поведение оператора == для экземпляров класса. Он возвращает True, если string и power текущего экземпляра равны string и power другого экземпляра.
3) __hash__(self): Этот метод возвращает хеш-значение экземпляра класса, которое может быть использовано для быстрого поиска в множествах и словарях.
4) __repr__(self): Этот специальный метод возвращает строку, которая представляет собой "официальное" строковое представление экземпляра класса

Функция add_powers может принимать бесконечное количество аргументов. В ней создаём пустой словарь, где ключ — это строка, а значение — степень. Проходимся последовательно по всем аргументам add_powers(). Если строка от этого аргумента есть в ключах созданного нами раннее словаря, то к ключу словаря данного элемента прибавляем степень. Иначе: если строки аргумента нет, то она добавляется вместе со степенью.Создаём массив result. Проходимся по каждому элементу словаря и создаём объект PowerString для каждого элемента словаря. Возвращаем массив типа PowerString.

Функция puller принимает на вход массив типа PowerString и выводит элемент клетки таблицы в эталонном виде.

Работает Функция puller следующим образом:

1) Если массив состоит из 1-го эл-та, то объект типа PowerString возвращается в эталонном виде с помощью метода класса get_result().

2) Если массив состоит более, чем из одного элемента, то создаём пустую строку и поочерёдно прибавляем объекты типа PowerString, не забывая о

знаке умножения. А в конце делаем срез строки, чтобы она не заканчивалась знаком '*'.

Функция puller2() принимает на вход массив, где могут быть одновременно объект типа PowerString и list. Puller2() возвращает строку в эталонном виде, используя функцию puller().

Функция result_add_powers принимает 2 аргумента: объект 1 и объект 2. Она рассматривает 4 случая: 1) оба объекта являются массивами 2) первый объект явл. массивом 3) второй объект явл. массивом 4) оба объекта не явл. массивом. Данная функция объединяет массивы и применяет функцию add_powers перед этим, распаковав массив. Данная функция нужна для перемножения 2 элементов таблицы Кэли.

Функция for_input берёт в качестве аргумента введённые пользователем через пробел данные, каждый введённый элемент преобразует в объект типа PowerString и помещает в массив, который потом и возвращает.

Функция remove_duplicates() принимает на вход строку и удаляет повторяющиеся элементы.

Функция replace_elements() заменяет эл-ты массива arr, т.е массива значений таблицы Кэли на соответствующие им отношения.

После реализации всех нужных для работы внутренней части программы функций приступаем к реализации графического интерфейса.

Класс MainWidget наследуется от класса QWidget в PyQt5. QWidget - это базовый класс для всех объектов пользовательского интерфейса в PyQt5. MainWidget представляет собой виджет, который функционирует как калькулятор таблиц Кэли.

В методе __init__ создаются все необходимые виджеты: метки (QLabel), поля для ввода текста (QLineEdit), кнопка (QPushButton) и таблица (QTableWidget). Также в этом методе устанавливаются вызовы событий (например, кнопка вызывает метод calculate при нажатии) и создаются макеты для размещения виджетов.

В функции calculate() мы считываем входные данные, обрабатываем их с помощью уже написанной раннее функции for_input(). Затем создаём начальную таблицу и заполняем её. Потом усовершенствуем список relation для удобства дальнейшей работы. Делаем замены с помощью функции replace_elements(). Затем ищем уникальные элементы для

обновлённой шапки таблицы. Считаем обновлённую таблицу, делаем замены и выводим.

Вывод таблицы:

1) Устанавливаются количество строк и столбцов для таблицы в соответствии с длиной списков obr2_new и obr1_new.
2) В цикле for создаём элементы QTableWidgetItem, которые затем добавляются в таблицу методом setItem.
3) Создаются заголовки для столбцов и строк таблицы.
4) Устанавливаем заголовки для столбцов и строк таблицы.
5) В последнем цикле for изменяем размеры столбцов в соответствии с содержимым.

Запуск приложения:

1) Создаём объект QApplication, который управляет ресурсами системы и явл. основой для PyQt-приложения
2) Создание экземпляра класса, который явл. основным виджет приложения
3) Запуск команды для отображения окна приложения на экране.
4) Запуск главного цикл дествий приложения и завершение Python-скрипта

Инструкция по работе с программой:

1) При запуске программы открывается главное меню.
2) В главном меню следует ввести образующие и отношения группы в соответствующие окна через пробел с использованием знака “^” и “*”.
3) При нажатии кнопки “Рассчитать” выводится заданная пользователем таблица Кэли с учётом отношений.

Вычисление сложности алгоритма

Основные сложные операции: создание и обновление таблицы, замены элементов и поиск уникальных элементов.

Создание начальной таблицы:

Вложенные проходы по obr1 и obr2: O(n^2), где n - количество элементов в obr1 (и obr2, так как они копируются).

Создание и обновление новой таблицы:

- Вложенные проходы: O(n^2).
- Замены в таблице: O(n^3).

Итоговая сложность для создания новой таблицы ~ O(n^3).

Функция replace_elements:

Вложенные проходы по массиву arr и списку relation: O(r * s * t), где r - количество строк в arr, s - количество элементов в строках arr, t - количество элементов в relation.

Сложность ~ O(r * s * t).

Поиск уникальных элементов: O(n^2)

Общая временная сложность алгоритма ~ O(n^3), где n - количество элементов в списке образующих и соотношений.

Тестовые примеры. Скриншоты программы

Образующие : e, a, b^2
Соотношения: $a \cdot b^2$

Шаг 1: Заполняем таблицу, шапку таблицы составляем из образующих

	e	a	b^2
e	e	a	b^2
a	a	a^2	ab^2
b^2	b^2	b^2a	b^4

Шаг 2: Заменяем эл-ты таблицы соотв. соотношениями

	e	a	b^2
e	e	a	b^2
a	a	a^2	e
b^2	b^2	b^2a	b^4

Шаг 3: Эл-ты, что не встречались раннее добавляем в шапку таблицы и вычисляем

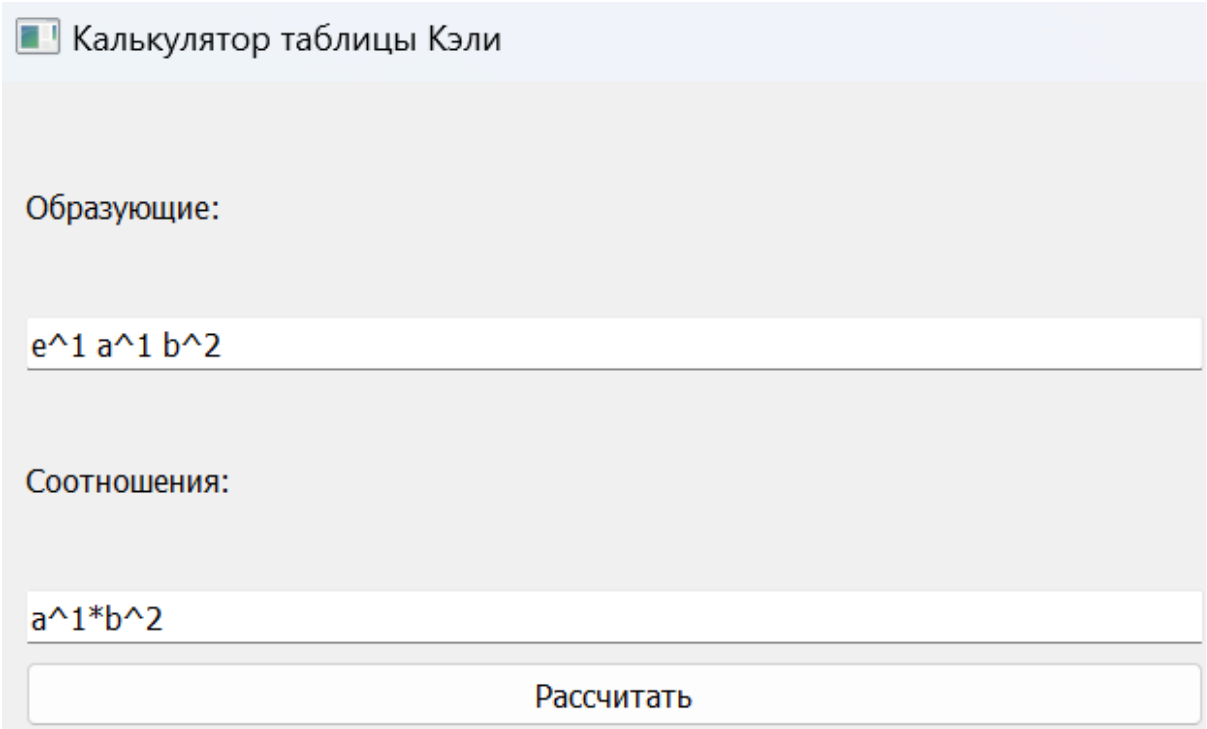
Обновлённую таблицу

	e	a	b^2	a^2	b^2a	b^4
e	e	a	b^2	a^2	b^2a	b^4

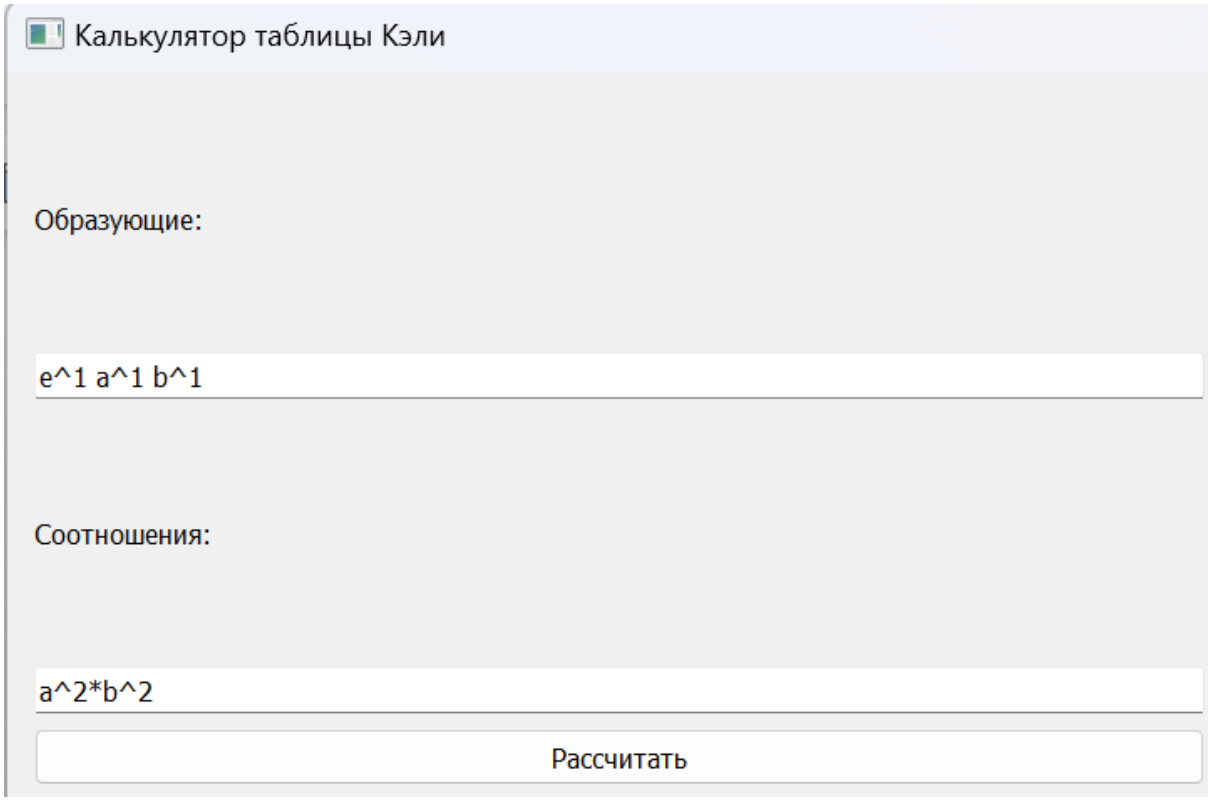
a	a	a^2	ab^2	a^3	a^2b^2	ab^4
b^2	b^2	b^2a	b^4	b^2a^2	b^4a	b^6
a^2	a^2	a^3	a^2b^2	a^4	a^3b^2	a^2b^4
b^2a	b^2a	b^2a^2	b^4a	b^2a^3	b^4a^2	b^6a
b^4	b^4	b^4a	b^6	b^4a^2	b^6a	b^8

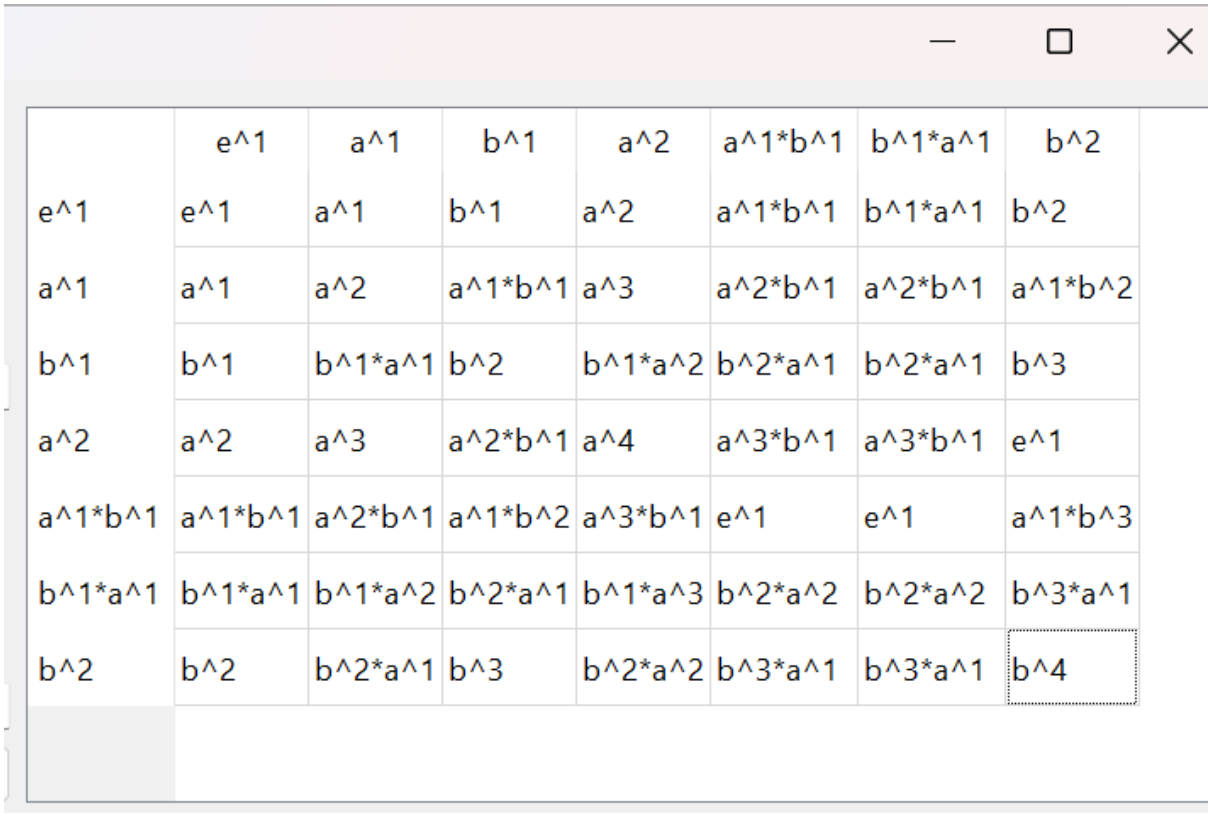
Шаг 4: Заменяем эл-ты таблицы соотв. соотношениями

	e	a	b^2	a^2	b^2a	b^4
e	e	a	b^2	a^2	b^2a	b^4
a	a	a^2	e	a^3	a^2b^2	ab^4
b^2	b^2	b^2a	b^4	b^2a^2	b^4a	b^6
a^2	a^2	a^3	a^2b^2	a^4	a^3b^2	a^2b^4
b^2a	b^2a	b^2a^2	b^4a	b^2a^3	b^4a^2	b^6a
b^4	b^4	b^4a	b^6	b^4a^2	b^6a	b^8



	e^1	a^1	b^2	a^2	b^2*a^1	b^4
e^1	e^1	a^1	b^2	a^2	b^2*a^1	b^4
a^1	a^1	a^2	e^1	a^3	a^2*b^2	a^1*b^4
b^2	b^2	b^2*a^1	b^4	b^2*a^2	b^4*a^1	b^6
a^2	a^2	a^3	a^2*b^2	a^4	a^3*b^2	a^2*b^4
b^2*a^1	b^2*a^1	b^2*a^2	b^4*a^1	b^2*a^3	b^4*a^2	b^6*a^1
b^4	b^4	b^4*a^1	b^6	b^4*a^2	b^6*a^1	b^8





	e^1	a^1	b^1	a^2	a^1*b^1	b^1*a^1	b^2
e^1	e^1	a^1	b^1	a^2	a^1*b^1	b^1*a^1	b^2
a^1	a^1	a^2	a^1*b^1	a^3	a^2*b^1	a^2*b^1	a^1*b^2
b^1	b^1	b^1*a^1	b^2	b^1*a^2	b^2*a^1	b^2*a^1	b^3
a^2	a^2	a^3	a^2*b^1	a^4	a^3*b^1	a^3*b^1	e^1
a^1*b^1	a^1*b^1	a^2*b^1	a^1*b^2	a^3*b^1	e^1	e^1	a^1*b^3
b^1*a^1	b^1*a^1	b^1*a^2	b^2*a^1	b^1*a^3	b^2*a^2	b^2*a^2	b^3*a^1
b^2	b^2	b^2*a^1	b^3	b^2*a^2	b^3*a^1	b^3*a^1	b^4

Прикладная задача

Одной из прикладных задач, которую можно решить с помощью таблицы Кэли, заданной образующими и определяющими соотношениями — это

моделирование возможных переходов между состояниями молекулы в ходе реакции. Допустим, что есть система из нескольких состояний, которая описывает поведение молекулы в химической реакции. Образующие могут представлять различные типы химических воздействий или взаимодействий между молекулами.

Соотношения могут описывать законы сохранения, энергетические барьеры или другие физические ограничения.

Эту таблицу можно использовать для анализа, какие состояния молекулы могут быть достигнуты при различных комбинациях воздействий, и предсказания исхода химической реакции в зависимости от начальных условий.

Таким образом, таблица Кэли, заданная образующими и соотношениями, может быть полезным инструментом для моделирования и анализа поведения в химии.

Код программы алгоритма:

```
Import sys
from PyQt5.QtWidgets import (QApplication, QWidget, QLabel, QLineEdit, QPushButton,
                             QVBoxLayout, QHBoxLayout, QTableWidget, QTableWidgetItem)
from PyQt5.QtCore import Qt


class PowerString:  # класс, который хранит строку и int число
    def __init__(self, string, power):
        self.string = string  # значение образующей
        self.power = power  # степень

    def get_result(self):
        return f"{self.string}^{self.power}"
        # так запустить эталонный вывод:
        # ps1 = PowerString("a", 2)
        # print(ps1.get_result())  Получится вот так! a^2

    def __eq__(self, other):
        return self.string == other.string and self.power == other.power

    def __hash__(self):
        return hash((self.string, self.power))
```

```
    def __repr__(self):
        return f"PowerString('{self.string}', {self.power})"


# это всё для 1 клетки таблицы
def add_powers(*powers):  # функция принимает несколько аргументов, поэтому перед аргументом *
    result_powers = {}  # словарь, где ключи - это строка, а значения - степень
    for power in powers:  # проходка по всем аргументам add_powers()
        if power.string in result_powers:
            result_powers[power.string] += power.power
            # Если строка аргумента уже есть в словаре result_powers, то к соотв. степени добавляется степень аргумента.
        else:
            result_powers[power.string] = power.power  # Если строки аргумента нет,то она добавляется вместе со степенью
    if 'e' in result_powers:
        if len(result_powers) == 1:
            result_powers['e'] = 1
        else:
            del result_powers['e']
    result = []  # Здесь будут храниться объекты PowerString
    for string, power in result_powers.items():  # проходимся по каждому элементу словаря
        result.append(PowerString(string, power))  # создаём объект Powerstring для каждого объекта словаря
    return result
    # Не забудь, что возвращается список!!!


def puller(pow_string):  # принимает на вход массив типа PowerString и красиво выводит клеточки
    if len(pow_string) == 1:
        return pow_string[0].get_result()
    elif len(pow_string) > 1:
        str = ''
        for i in pow_string:
            str += i.get_result() + '*'
        return str[0:-1]
```

```
def puller2(pow_string):  # для второго вывода образующих когда PowString и массивы перемешаны
    if type(pow_string) is PowerString:
        return pow_string.get_result()
    else:
        return puller(pow_string)


def result_add_powers(obj1, obj2):
    obj = []
    if (type(obj1) is list) and (type(obj2) is list):
        obj = obj1 + obj2
    elif type(obj1) is list:
        obj = obj1.copy()
        obj.append(obj2)
    elif type(obj2) is list:
        obj.append(obj1)
        obj = obj + obj2
    else:
        obj.append(obj1)
        obj.append(obj2)
    return add_powers(*obj)  # распаковка массива


def for_input(massiv_prev):  # для приведения введённых данных к нормальному формату для дальнейшей работы
    norm_massiv = []
    for i in massiv_prev:  # a^2  a a^2*a^5
        if '^' in i and len(i) == 3:
            string, power = i.split('^')
            power = int(power)
            norm_massiv.append(PowerString(string, power))  # записали значение в класс
        elif len(i) > 3:
            two_values = []  # мини-список, т.к отнош. 2
            norm_massiv.append(two_values)
            first_part, second_part = i.split('*')  # a^2 и a^5
            string, power = first_part.split('^')
            power = int(power)
            two_values.append(PowerString(string, power))
            string, power = second_part.split('^')
            power = int(power)
```

```
            two_values.append(PowerString(string, power))
    return norm_massiv


def remove_duplicates(input_string):  # принимает строку и удаляет
повторяющиеся эл-ты
    input_list = input_string.split(" ")
    output_list = []
    for i in input_list:
        if i not in output_list:
            output_list.append(i)
    return " ".join(output_list)


def replace_elements(arr, relation):  # f-ия для замены элементов
    for i in range(len(arr)):
        for j in range(len(arr[i])):
            for rel in relation:
                if arr[i][j] == rel:
                    arr[i][j] = [PowerString("e", 1)]
                    break


class MainWidget(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Калькулятор таблицы Кэли")

        # Входные данные
        self.generators_label = QLabel("Образующие:")
        self.generators_input = QLineEdit()
        self.relations_label = QLabel("Соотношения:")
        self.relations_input = QLineEdit()

        # Кнопки
        self.calculate_button = QPushButton("Рассчитать")
        self.calculate_button.clicked.connect(self.calculate)  # Вызов функции подсчёта

        # Таблица
        self.table = QTableWidget()
        self.table.setColumnCount(0)
        self.table.setRowCount(0)
```

```
        # Макет
        input_layout = QVBoxLayout()
        input_layout.addWidget(self.generators_label)
        input_layout.addWidget(self.generators_input)
        input_layout.addWidget(self.relations_label)
        input_layout.addWidget(self.relations_input)
        input_layout.addWidget(self.calculate_button)

        main_layout = QHBoxLayout()
        main_layout.addLayout(input_layout)
        main_layout.addWidget(self.table)

        self.setLayout(main_layout)

    def calculate(self):
        # Получение входных данных
        obr_prev = self.generators_input.text().split()
        relation_prev = self.relations_input.text().split()

        # Обработка входных данных
        obr1 = for_input(obr_prev)  # преобразование в удобный вид
        obr2 = obr1.copy()  # 2 независимые списка, просто "=" сделало бы
созависимые
        relation = for_input(relation_prev)  # Список отношений типа класса
PowerString, но могут быть и части вложенные

        # 1 шаг: начальная таблица
        arr = []
        for j in range(len(obr2)):  # obr2[j] - ведущий эл-т
            arr_ = []
            for i in range(len(obr1)):
                arr_.append(result_add_powers(obr2[j], obr1[i]))
            arr.append(arr_)

        # Усовершенствовали список relation для удобства дальнейшей работы
        for i in range(len(relation)):  # Теперь внутри списка relation лежат списки
            if not isinstance(relation[i], list):
                relation[i] = [relation[i]]

        # Шаг 2: делаем замены в таблице
        replace_elements(arr, relation)
```

```
# Поиск уникальных элементов для будущей обновлённой шапки:
unique_elements = []
for arr_ in arr:
    for j in arr_:
        if j not in unique_elements:
            unique_elements.append(j)
# Добавление уникальных элементов в шапку таблицы
for el in unique_elements:
    obr1.append(el)

# Обновлённая шапка таблицы
str_obr1 = ' '.join(puller2(i) for i in obr1)
for_input(remove_duplicates(str_obr1).split())
obr1_new = for_input(remove_duplicates(str_obr1).split())
obr2_new = obr1_new.copy()
# Промежуточная таблица (до замен)
arr = []
for j in range(len(obr2_new)):  # obr2[j] - ведущий эл-т
    arr_ = []
    for i in range(len(obr1_new)):
        arr_.append(result_add_powers(obr2_new[j], obr1_new[i]))
    arr.append(arr_)

# Итоговая таблица для вывода (без столбца obr2_new слева):
replace_elements(arr, relation)  # Отношения применили
# Построение таблицы
self.table.setRowCount(len(obr2_new))
self.table.setColumnCount(len(obr1_new))

# Заполнение таблицы
for j in range(len(obr2_new)):
    for i in range(len(obr1_new)):
        item = QTableWidgetItem(puller2(arr[j][i]))
        self.table.setItem(j, i, item)

# Шапка таблицы
header_labels_width = [puller2(i) for i in obr1_new]
self.table.setHorizontalHeaderLabels(header_labels_width)
header_labels_height = [puller2(i) for i in obr1_new]
self.table.setVerticalHeaderLabels(header_labels_height)
```

```
        # Изменение размеров столбцов
        for i in range(self.table.columnCount()):
            self.table.resizeColumnToContents(i)


app = QApplication(sys.argv)  # Создание объекта QApplication, который управляет ресурсами системы и явл. основой
# для PyQt-приложения
window = MainWidget()  # основной виджет приложения
window.show()  # для отображения окна GUI
sys.exit(app.exec_())  # для завершения Python-скрипта
```