

**МИНОБРНАУКИ РОССИИ**

**Федеральное государственное бюджетное образовательное учреждение  
высшего образования**

**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ  
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»**

Кафедра теоретических основ  
компьютерной безопасности и  
криптографии

**Отношение эквивалентности и отношение порядка**

**ОТЧЁТ**

**ПО ДИСЦИПЛИНЕ**

**«ПРИКЛАДНАЯ УНИВЕРСАЛЬНАЯ АЛГЕБРА»**

Студентки 3 курса 331 группы  
специальности 10.05.01 Компьютерная безопасность  
факультета компьютерных наук и информационных технологий  
Шуликиной Анастасии Александровны

Преподаватель

профессор, д.ф.-м.н.

\_\_\_\_\_  
подпись, дата

В. А. Молчанов

Саратов 2022

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ .....	3
1 Цель работы и порядок её выполнения .....	4
2 Теория .....	5
2.1 Понятия полугруппы, подполугруппы и порождающего множества	5
2.1.1 Алгоритм поиска номера элемента .....	6
2.1.2 Алгоритм проверки свойства ассоциативности .....	7
2.1.3 Алгоритм построения подполугруппы по таблице Кэли ...	7
2.1.4 Алгоритм произведения бинарных отношений .....	7
2.1.5 Алгоритм построения полугруппы бинарных отношений по заданному порождающему множеству .....	8
2.2 Понятия подгруппы, порождающего множества и определяю- щих соотношений .....	8
2.2.1 Алгоритм построения полугруппы по порождающему мно- жеству и определяющим соотношениям .....	10
2.3 Код программы, на основе рассмотренных алгоритмов, на язы- ке C++ .....	12

## ВВЕДЕНИЕ

В данной лабораторной работе поставлена задача рассмотрения понятия теории полугруппы, подполугруппы, порождающего множества и определяющих соотношений, разобрать и реализовать алгоритмы построения подполугрупп по таблице Кэли, полугруппы бинарных отношений по заданному порожденному множеству и подгруппы по порождающему множеству и определяющим соотношениям.

## 1 Цель работы и порядок её выполнения

Цель работы – изучение основных понятий теории полугрупп.

Порядок выполнения работы:

1. Рассмотреть понятия полугруппы, подполугруппы и порождающего множества. Разработать алгоритм построения подполугрупп по таблице Кэли.
2. Разработать алгоритм построения полугруппы бинарных отношений по заданному порождающему множеству.
3. Рассмотреть понятия подгруппы, порождающего множества и определяющих соотношений. Разработать алгоритм построения полугруппы по порождающему множеству и определяющим соотношениям.

## 2 Теория

### 2.1 Понятия полугруппы, подполугруппы и порождающего множества

Полугруппа – это алгебра  $S = (S, \cdot)$  с одной ассоциативной бинарной операцией  $\cdot$ , т.е. выполняется  $(x \cdot y) \cdot z = x \cdot (y \cdot z)$  для любых  $x, y, z \in S$ .

Если полугрупповая операция называется умножением (соответственно, сложением), то полугруппу называют мультипликативной (соответственно, аддитивной).

Лемма. Для любого непустого множества  $X$  множество всех бинарных отношений на множестве  $X$  с операцией композиции является полугруппой. Такая полугруппа называется симметрической полугруппой бинарных отношений на множестве  $X$  и обозначается символом  $\beta(X)$ .

Следствие. Множество всех (частичных) преобразований непустого множества  $X$  с операцией композиции является полугруппой. Такая полугруппа называется симметрической полугруппой (частичных) преобразований множества  $X$  и обозначается символом  $T(X)$  (соответственно,  $PT(X)$ ).

В случае конечного  $n$ -элементного множества  $S = \{s_1, s_2, \dots, s_n\}$  операция умножения на  $S$  задаётся таблицей Кэли размерности  $n \times n$ , строки и столбцы которой помечены элементами множества  $S$  и в которой на пересечении  $i$ -ой строки и  $j$ -го столбца стоит произведение  $s_i \cdot s_j$  элементов  $s_i, s_j$ .

Таблица 1.

$\cdot$	$s_1$	$s_2$	$\dots$	$s_n$
$s_1$	$s_1 \cdot s_1$	$s_1 \cdot s_2$	$\dots$	$s_1 \cdot s_n$
$s_2$	$s_2 \cdot s_1$	$s_2 \cdot s_2$	$\dots$	$s_2 \cdot s_n$
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$
$s_n$	$s_n \cdot s_1$	$s_n \cdot s_2$	$\dots$	$s_n \cdot s_n$

Классификация элементов полугруппы. Элемент  $s$  полугруппы  $S$  называется:

1. нулевым, если  $s \cdot x = x \cdot s = s$  для всех  $x \in S$  (мультипликативной записи операции полугруппы такой элемент обозначается символом  $0$  и называется нулем);
2. нейтральным (единичным), если  $s \cdot x = x \cdot s = x$  для всех  $x \in S$  при мультипликативной записи операции полугруппы такой элемент обозначается символом  $1$  и называется единицей);

3. обратимым, если для некоторого  $x \in S$  выполняется свойство:  $x \cdot s = s \cdot x = 1$  (такой элемент  $x$  называется обратным для элемента  $s$  и обозначается символом  $s^1$ );
4. идемпотентом, если  $s \cdot s = s$ , т.е.  $s^2 = s$ ;

Если в пункте 1 выполняется лишь равенство  $x \cdot s = s$  для всех  $x \in S$  то  $s$  называется правым нулем. Аналогично определяется левый нуль и односторонние единицы.

Подмножество  $X$  полугруппы  $S$  называется подполугруппой, если  $X$  устойчиво относительно операции умножения, т.е. для любых  $x, y \in X$  выполняется свойство:  $x \cdot y \in X$ .

В этом случае множество  $X$  с ограничением на нем операции умножения исходной полугруппы  $S$  образует полугруппу.

В силу общего свойства подалгебр пересечение любого семейства  $X_i$  ( $i \in I$ ) подполугрупп полугруппы  $S$  является подполугруппой  $S$  и, значит, множество  $Sub(S)$  всех подполугрупп полугруппы  $S$  является системой замыканий. Следовательно, для любого подмножества  $X$  полугруппы  $S$  существует наименьшая подполугруппа  $S$ , содержащая множество  $X$ . Такая полугруппа обозначается символом  $\langle X \rangle$  и называется подполугруппой  $S$ , порождённой множеством  $X$ . При этом множество  $X$  называется также порождающим множеством подполугруппы  $\langle X \rangle$ .

В частности, если  $\langle X \rangle = S$ , то  $X$  называется порождающим множеством полугруппы  $S$  и говорят, что множество  $X$  порождает полугруппу  $S$ .

Легко видеть, что полугруппа  $\langle X \rangle$  состоит из всевозможных конечных произведений  $x_1 \cdot \dots \cdot x_n$  элементов  $x_1, \dots, x_n \in X$ , т.е. выполняется равенство:  $\langle X \rangle = \{x_1 \cdot \dots \cdot x_n : n \in N \text{ и } x_1, \dots, x_n \in X\}$ .

### 2.1.1 Алгоритм поиска номера элемента

Вход. Элемент  $a$ , список элементов  $x$  размерностью  $N$ .

Выход. Номер элемента  $a$  в списке  $x$ .

Шаг 1. Цикл  $i$  от 1 до  $N$ .

Шаг 1.2. Когда  $x[i] = a$ .

Шаг 2. Выводим номер элемента  $i$ .

Трудоёмкость алгоритма  $O(N)$

### 2.1.2 Алгоритм проверки свойства ассоциативности

Вход. Таблица Кэли  $A$  размерностью  $N$  и список элементов  $x$ .

Выход. «Операция обладает свойством ассоциативности» или «Операция не обладает свойством ассоциативности»

Шаг 1. Создаем булеву переменную  $chek = true$ .

Шаг 2. Цикл  $i$  от 1 до  $N$ .

Шаг 2.2. Для каждого  $i$  цикл  $j$  от 1 до  $N$ .

Шаг 2.3. Для каждого  $j$  цикл  $k$  от 1 до  $N$ .

Шаг 2.4. Для каждого  $k$  применяем алгоритм поиска номера элемента 2.1.1 для каждого  $A_{(ij)}$  и  $A_{(jk)}$  и сохраняем результаты в  $i2$  и  $j2$ .

Шаг 3. Если  $A_{i2,k} \neq A_{i,j2}$ , то  $chek = false$ .

Шаг 4. Иначе, если  $chek = true$ , то выводим «Операция обладает свойством ассоциативности» .

Трудоемкость алгоритма  $O(N^3)$

### 2.1.3 Алгоритм построения подполугруппы по таблице Кэли

Вход. Элементы полугруппы  $bunch$ , таблица Кэли  $Kel$  размерностью  $N$  и элементы подмножества  $subset$ .

Выход. Подполугруппа  $res$ .

Шаг 1.  $res = subset$ .

Шаг 2. Пока при объединении при проверке по алгоритму 2.1.2 выполняется свойство ассоциативности выполняются следующие шаги.

Шаг 2.2. Заводится дополнительная переменная  $resSt$ . Цикл  $i$  от 1 до  $(subset.size - 1)$ .

Шаг 2.2. Для каждого  $i$  цикл  $j$  от 1 до  $(res.size - 1)$ .

Шаг 2.3. Создаём переменную  $k = Kel[subset[i].second][res[j].second]$ .

Шаг 2.4. В  $resSt$  помещаем пару  $(k, \text{номер элемента, который находим по алгоритму 2.1.1})$ .

Шаг 3.  $return res = \text{объединение элементов } res \text{ и } resSt$ .

Трудоемкость алгоритма  $O(N^2 * M)$

### 2.1.4 Алгоритм произведения бинарных отношений

Вход. Две матрицы бинарного отношения  $a_{(ij)}$  размерностью  $N$ ,  $b_{(ij)}$  размерностью  $N$ .

Выход. Матрица произведения бинарного отношения  $res$ .

Шаг 1. Создаем переменную  $res = a_{(ij)}$ .

Шаг 2. Цикл  $i$  от 1 до  $N$ .

Шаг 2.2. Для каждого  $i$  цикл  $j$  от 1 до  $N$  и создает переменную  $count = 0$ .

Шаг 2.3. Для каждого  $j$  цикл  $k$  от 1 до  $N$ .

Шаг 2.4. Для каждого  $k$   $count + = a[i][k] * b[k][j]$ , если  $count > 0$ , то  $res[i][j] = 1$  иначе  $res[i][j] = 0$ .

Шаг 3.  $return res_{(ij)}$ .

Трудоемкость алгоритма  $O(N^3)$

2.1.5 Алгоритм построения полугруппы бинарных отношений по заданному порождающему множеству

Вход. Несколько матриц бинарного отношения порождающего множества  $matrix$  размерностью  $N$ .

Выход. Полугруппа бинарных отношений  $res$ .

Шаг 1. Создаем переменную  $sets$  в которую добавляем все матрицы порождающего множества.

Шаг 2. Создаем переменную  $group$ . Пока  $group \neq sets$  выполняются следующие шаги.

Шаг 2.2.  $group = sets$ .

Шаг 2.3.  $sets = sets.insert(\text{композиции поданных матриц})$ . Композиция вычисляется по алгоритму 2.1.4.

Шаг 3.  $return sets$ .

Трудоемкость алгоритма  $O(N^2 * M)$

2.2 Понятия подгруппы, порождающего множества и определяющих соотношений

Полугруппа с единичным элементом называется моноидом. Другими словами, моноид  $M = (M, \cdot, 1)$  – это алгебра с ассоциативной бинарной операцией и выделенным единичным элементом 1. При этом полугруппа  $(M, \cdot)$  называется полугруппой моноида  $M = (M, \cdot, 1)$  и гомоморфизмом моноидов называется гомоморфизм их полугрупп, сохраняющий выделенные единичные элементы.

Для любой полугруппы  $S = (S, \cdot)$  канонически определяется моноид  $M(S)$  по следующему правилу:  $M(S) = S \cup \{1\}$  для некоторого элемента



$1 \notin S$  и умножение в  $M(S)$  на новый элемент 1 определяется по формуле:  $x \cdot 1 = 1 \cdot x = x$  (а умножение элементов из  $S$  совпадает с умножением этих элементов в полугруппе  $S$ ). Полугруппа моноида  $M(S)$  обозначается символом  $S^1$  и называется полугруппой с внешне присоединенной единицей.

Моноид называется группой, если в нем все элементы обратимы.

Для любого моноида  $M$  множество всех обратимых элементов  $M^*$  с операцией умножения моноида является группой, т.е. для любых  $a, b \in M^*$  выполняется условие  $a \cdot b \in M^*$ .

Пусть  $A$  – произвольное множество, называемое алфавитом. Элементы  $a \in A$  называются буквами. Словом над алфавитом  $A$  называется конечная последовательность букв  $a_1 \dots a_n$  алфавита  $A$ . Слово без букв называется пустым словом и обозначается символом  $\Lambda$ . Для слова  $w = a_1 \dots a_n$  число  $n$  букв в определяющей его последовательности называется длиной этого слова и обозначается символом  $l(w)$ .

Обозначим символом  $A^+$  множество всех непустых слов над алфавитом и символом  $A^*$  – множество слов  $A^* = A^+ \cup \{\Lambda\}$ . На этих множествах слов определена операция умножения, которая называется операцией конкатенации слов и определяется по правилу: любым словам  $w_1 = a_1 \dots a_n$  и  $w_2 = b_1 \dots b_m$  операция конкатенации ставит в соответствие слово  $w_1 \cdot w_2 = a_1 \dots a_n b_1 \dots b_m$ . В результате множество слов  $A^+$  с операцией конкатенации образует полугруппу, которая называется полугруппой слов над алфавитом  $A$ , и множество слов  $A^*$  с операцией конкатенации образует полугруппу с единичным элементом  $\Lambda$ , которая называется моноидом слов над алфавитом  $A$ .

Теорема. (О представлении полугрупп словами). Любая полугруппа  $S$  является фактор-полугруппой некоторой полугруппы слов  $A^+$ , т.е.  $S \cong A^+/\varepsilon$  для некоторой конгруэнции  $\varepsilon$  полугруппы  $A^+$ .

Для любой конечной полугруппы  $S$  найдется такой конечный алфавит  $A$ , что для некоторого отображения  $\varphi : A \rightarrow S$  выполняется равенство  $\langle \varphi(A) \rangle = S$  и, значит,  $S \cong A^+/\ker \varphi$ . В этом случае множество  $A$  называется множеством порождающих символов полугруппы  $S$  (относительно отображения  $\varphi : A \rightarrow S$ ). Если при этом для слов  $w_1, w_2 \in A^+$  выполняется равенство  $\varphi(w_1) = \varphi(w_2)$ , т.е.  $w_1 \equiv w_2 (\ker \varphi)$ , то говорят, что на  $S$  выполняется соотношение  $w_1 = w_2$  (относительно отображения  $\varphi : A \rightarrow S$ ).

Очевидно, что в общем случае множество таких соотношений  $w_1 = w_2$

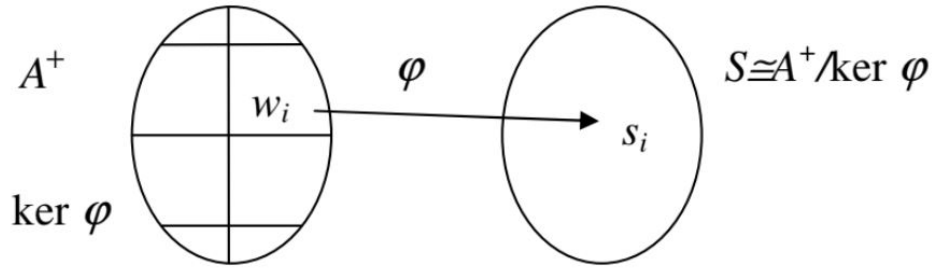


Рисунок 1

для всех пар  $(w_1, w_2) \in \ker \varphi$  будет бесконечным и не представляется возможности эффективно описать полугруппу  $S$  в виде полугруппы классов конгруэнции  $\ker \varphi$ . Однако в некоторых случаях можно выбрать такое сравнительно простое подмножество  $\rho \subset \ker \varphi$ , которое однозначно определяет конгруэнцию  $\ker \varphi$  как наименьшую конгруэнцию полугруппы  $A^+$ , содержащую отношение  $\rho$ , т.е.  $\ker \varphi = f_{con}(\rho) = f_{eq}(f_{reg}(\rho))$ .

Так как в случае  $(w_1, w_2) \in \rho$  по-прежнему выполняется равенство  $\varphi(w_1) = \varphi(w_2)$ , то будем писать  $w_1 = w_2$  и называть такие выражения определяющими соотношениями.

**2.2.1 Алгоритм построения полугруппы по порождающему множеству и определяющим соотношениям**

**Вход.** Алфавит  $alf$  и определяющие отношение  $word$ .

**Выход.** Полугруппа  $res$ .

**Шаг 1.** Создается сортирующий контейнер, в котором рассматриваются слова  $word$  и элементы  $alf$ . Если  $word! = alf.end()$ , то создается новая переменная  $word_3.push\_back(word)$ . Если  $word! = alf.end()$ , то отправляется в  $res$ .

**Шаг 2.** Создаём переменную  $len = 2$ . Цикл, который выполняется пока  $len \leq$  длине максимального соотношения определяющего отношения.

**Шаг 2.2.** Цикл по  $i$  от  $word_3.size()$  до 1.

**Шаг 2.3.** Для каждого  $i$ :  $len > word.size() || len > i + 1$ .

**Шаг 2.4.** Цикл по  $j$  от  $i$  до  $i - len$ .

**Шаг 2.5.** Создаем строку  $probe = word_3[j] + probe$ .

**Шаг 3.** Создаётся новая строка  $word_2$ .

**Шаг 4.** Если  $probe! = alf.end()$ , то создается новая переменная

*probe.pushback(word)*. Цикл по  $k$  от 0 до  $j$ .

Шаг 4.2. Для каждого  $k$ :  $word_2 = word_2 + word_3[k]$ .

Шаг 4.3  $word_2 = word_2 + alf[probe]$ .

Шаг 4.4. Цикл по  $k$  от  $i + 1$  до  $word_3.size()$ .

Шаг 4.5.1. Для каждого  $k$ :  $word_2 = word_2 + word_3[k]$ .

Шаг 4.6. Если  $word_2.size() < word.size()$ , то  $alf.insert(word)$   
 $alf.insert(word_2)$ . Иначе если  $word_2! = alf.end()$ , то отправляется в  $res$ .

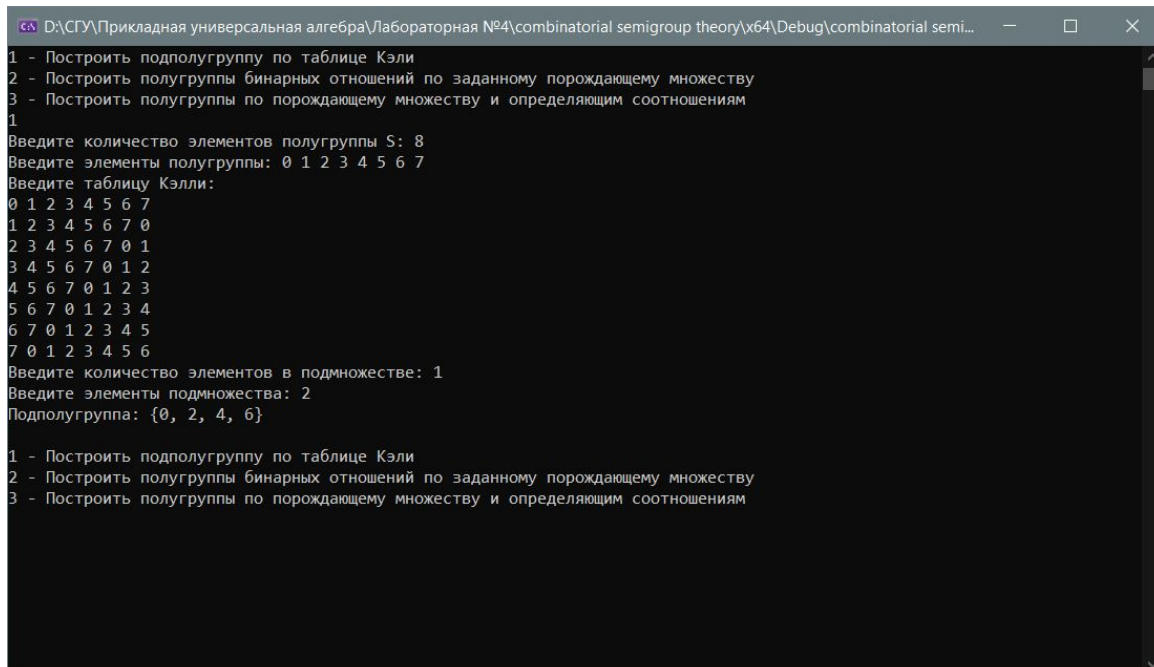
Шаг 5. Если  $word! = alf.end()$ , то в  $res$ .

Шаг 6. *return res*.

Трудоемкость алгоритма  $O(N * M * k^3)$

## 2.3 Код программы, на основе рассмотренных алгоритмов, на языке C++

На рисунках 2-5 можно увидеть работу, реализуемой программы, по рассмотренным алгоритмам.



```
D:\СГУ\Прикладная универсальная алгебра\Лабораторная №4\combinatorial semigroup theory\x64\Debug\combinatorial semi...
1 - Построить подполугруппу по таблице Кэли
2 - Построить полугруппы бинарных отношений по заданному порождающему множеству
3 - Построить полугруппы по порождающему множеству и определяющим соотношениям
1
Введите количество элементов полугруппы S: 8
Введите элементы полугруппы: 0 1 2 3 4 5 6 7
Введите таблицу Кэли:
0 1 2 3 4 5 6 7
1 2 3 4 5 6 7 0
2 3 4 5 6 7 0 1
3 4 5 6 7 0 1 2
4 5 6 7 0 1 2 3
5 6 7 0 1 2 3 4
6 7 0 1 2 3 4 5
7 0 1 2 3 4 5 6
Введите количество элементов в подмножестве: 1
Введите элементы подмножества: 2
Подполугруппа: {0, 2, 4, 6}

1 - Построить подполугруппу по таблице Кэли
2 - Построить полугруппы бинарных отношений по заданному порождающему множеству
3 - Построить полугруппы по порождающему множеству и определяющим соотношениям
```

Рисунок 2

```
D:\СГУ\Прикладная универсальная алгебра\Лабораторная №4\combinatorial semigroup theory\х64\Debug\combinatorial semi...
1 - Построить подполугруппу по таблице Кэли
2 - Построить полугруппы бинарных отношений по заданному порождающему множеству
3 - Построить полугруппы по порождающему множеству и определяющим соотношениям
2
Введите количество элементов на множестве: 3
Введите количество матриц в порождающем множестве: 3
Введите матрицу A
1 0 1
0 1 0
0 0 1
Введите матрицу B
1 1 0
0 0 1
1 0 1
Введите матрицу C
0 0 0
0 0 0
0 0 0

Полученная полугруппа:

C
0 0 0
0 0 0
0 0 0

A
1 0 1
0 1 0
0 0 1

B
1 1 0
0 0 1
1 0 1

D
1 1 1
0 0 1
1 0 1

E
1 1 1
1 0 1
1 1 1
```

Рисунок 3

### Листинг программы

```
#include <iostream>
#include <vector>
#include <set>
#include <map>
#include <string>
#include <algorithm>
#include <iomanip>
```

```

Выбрать D:\СГУ\Прикладная универсальная алгебра\Лабораторная №4\combinatorial semigroup theory\х64\Debug\combinat...
В
1 1 0
0 0 1
1 0 1
D
1 1 1
0 0 1
1 0 1
E
1 1 1
1 0 1
1 1 1
F
1 1 1
1 1 1
1 1 1
Таблица Кэли:
  A B C D E F
A A D C D E F
B D E C E F F
C C C C C C C
D D E C E F F
E E F C F F F
F F F C F F F
1 - Построить подполугруппу по таблице Кэли
2 - Построить полугруппы бинарных отношений по заданному порождающему множеству
3 - Построить полугруппы по порождающему множеству и определяющим соотношениям

```

Рисунок 4

```

D:\СГУ\Прикладная универсальная алгебра\Лабораторная №4\combinatorial semigroup theory\х64\Debug\combinatorial semi...
1 - Построить подполугруппу по таблице Кэли
2 - Построить полугруппы бинарных отношений по заданному порождающему множеству
3 - Построить полугруппы по порождающему множеству и определяющим соотношениям
3
Введите количество элементов алфавита: 2
Введите алфавит: a b
Введите количество определяющих соотношений: 3
Введите определяющие соотношения (через пробел):
ab ba
aaa aa
bb b
Полученная полугруппа: {a aa aab ab b }
Таблица Кэли:
  a aa aab ab b
a aa aa aab aab ab
aa aa aa aab aab aab
aab aab aab ab ab ab
ab ab ab b b b
b ab ab b b b
1 - Построить подполугруппу по таблице Кэли
2 - Построить полугруппы бинарных отношений по заданному порождающему множеству
3 - Построить полугруппы по порождающему множеству и определяющим соотношениям

```

Рисунок 5

```
using namespace std;
```

```

int Find1(int k, vector<int> bunch,
int n) {
    for (int j = 0; j < n; j++) {
        if (k == bunch[j])
            return j;
    }
}

```

```

bool Chek_ass(vector<pair<int, int>> res,
vector<pair<int, int>> prov) {
    if (res.size() == prov.size()) {
        int c = 0;
        for (int i = 0; i < res.size(); i++) {
            for (int j = 0; j < prov.size(); j++) {
                if (res[i].first == prov[j].first)
                    c++;
            }
        }
        if (res.size() == c)
            return false;
        else return true;
    }
    return true;
}

```

```

vector<pair<int, int>> Obed(vector<pair<int, int>> res,
vector<pair<int, int>> resSt, int el1, int el2) {
    vector<pair<int, int>> res1;
    for (int i = 0; i < el1; i++)
        res1.push_back(res[i]);
    for (int i = 0; i < el2; i++) {
        int c = 0;
        for (int j = 0; j < res1.size(); j++) {

```

```

        if (resSt[i].first == res1[j].first &&
            resSt[i].second == res1[j].second)
            c++;
    }
    if (c == 0)
        res1.push_back(resSt[i]);
    }
    return res1;
}

```

```

vector<pair<int, int>> Podpul(vector<vector<int>> Kel,
vector<pair<int, int>> subset, vector<int> bunch,
int n) {
    vector<pair<int, int>> res = subset;
    vector<pair<int, int>> chek;
    do {
        vector<pair<int, int>> resSt;
        for (int i = 0; i < subset.size(); i++) {
            for (int j = 0; j < res.size(); j++) {
                int k = Kel[subset[i].second][res[j].second];
                resSt.push_back(make_pair(k,
                    Find1(k, bunch, n)));
            }
        }
        chek = res;
        res = Obed(res, resSt, res.size(),
            resSt.size());
    } while (Chek_ass(res, chek));
    return res;
}

```

```

void Zad1() {
    int n, m, el;
    cout << "Enter the number of semigroup elements S:";
}

```



```

cin >> n;
vector <int> bunch;
cout << "Enter semigroup elements: ";
for (int i = 0; i < n; i++) {
    cin >> el;
    bunch.push_back(el);
}
vector <vector <int>> Kel;
Kel.resize(n);
cout << "Enter Callie Table:\n";
for (int i = 0; i < n; i++) {
    Kel[i].resize(n);
    for (int j = 0; j < n; j++)
        cin >> Kel[i][j];
}

cout << "Enter the number of elements in the subset:";
cin >> m;
vector <pair <int, int>> subset;
cout << "Enter Subset Elements: ";
for (int i = 0; i < m; i++) {
    cin >> el;
    int e = Find1(el, bunch, n);
    subset.push_back(make_pair(el, e));
}

vector<pair<int, int>> res;
res = Podpul(Kel, subset, bunch, n);
sort(res.begin(), res.end());
cout << "Subsemigroup: {";
    for (int i = 0; i < res.size(); i++)
        if (i != res.size() - 1)
            cout << res[i].first << ", ";
        else cout << res[i].first << "}\n";

```

```

}

vector <vector <int>> MultyMatr(vector <vector <int>> sets ,
vector <vector <int>> res , int n) {
    vector <vector <int>> res1;
    res1.resize(n);
    for (int i = 0; i < n; i++)
        res1[i].resize(n);

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            for (int k = 0; k < n; k++) {
                if (sets[i][k] == 1 &&
                    res[k][j] == 1) {
                    res1[i][j] = 1;
                    break;
                }
                else res1[i][j] = 0;
            }
        }
    }
    return res1;
}

set <vector <vector <int>>> InsMat
(set <vector <vector <int>>> sets , int n) {
    for (vector <vector <int>> sets1 : sets)
        for (vector <vector <int>> sets2 : sets) {
            sets.insert(MultyMatr(sets1 , sets2 , n));
        }
    return sets;
}

void Zad2() {

```

```

int n, m;
cout << "Enter the number of elements in the set: ";
cin >> n;
cout << "Enter the number of matrices in
the generating set: ";
cin >> m;

vector <vector <int>>> matrix;
map <char, vector <vector <int>>> in_m;
map <vector <vector <int>>, char> out_m;
matrix.resize(n);
set <vector <vector <int>>> sets;
int k = 0;
for (int i = 1; i <= m; i++) {
    cout << "Enter matrix " << char(65 + k) << "\n";
    for (int e = 0; e < n; e++) {
        matrix[e].resize(n);
        for (int j = 0; j < n; j++)
            cin >> matrix[e][j];
    }
    sets.insert(matrix);
    in_m.insert(make_pair(char(65 + k), matrix));
    out_m.insert(make_pair(matrix,
        char(65 + k++)));
}

set <vector <vector <int>>> group;
group = sets;
int kout = 0;
while (true) {
    for (auto i = sets.begin(); i !=
        sets.end(); i++) {
        for (auto j = sets.begin(); j !=
            sets.end(); j++) {

```

```

vector <vector <int>> newMatrix =
MultyMatr(*i , *j , n);

if (group.find(newMatrix) == group.end()) {
in_m.insert(make_pair(char(65 + k + kout),
newMatrix));
out_m.insert(make_pair(newMatrix,
char(65 + k + kout++)));
group.insert(newMatrix);
}
}
}

if (group == sets) {
    group = sets;
    sets = InsMat(sets , n);
    cout << endl;
    cout << "The resulting semigroup:\n";

for (vector <vector <int>> res1 : sets) {
    cout << "\n" << out_m[res1] << endl;
    for (int i = 0; i < res1.size(); i++) {
        cout << "\n";
        for (int j = 0; j < res1.size(); j++)
            cout << res1[i][j] << " ";
        }
        cout << endl;
    }

    cout << endl << "Cayley table: \n";
    cout << " ";
    for (int i = 0; i < out_m.size(); i++)
        cout << setw(4) << char(65 + i);
    cout << endl;
}

```

```

        for (int i = 0; i<out_m.size(); i++) {
            cout << char(65 + i) << setw(4);
            for (int j = 0; j < out_m.size(); j++)
                cout << setw(4)
                    << out_m[MultyMatr(in_m[char(65 + i)],
in_m[char(65 + j)], n)];
            cout << endl;
        }
        return;
    }
    else sets = group;
}
}

```

```

set<string> s, sss, s_1, s_2;
int maksimal = 0;
map<string, string> slowar;
int counter;
string word_2;
vector <vector <string>> kali;

```

```

vector<string> Sl(string word) {
    vector<string> new_word;
    for (int i = 0; i < word.size(); ++i)
        new_word.push_back(word.substr(i, 1));
    return(new_word);
}

```

```

void New(string word, bool flag, int row, int size) {
    if (sss.find(word) != sss.end() && !flag)
        return;
    vector<string> word_3 = Sl(word);
}

```

```

int len = 2;
while (len <= maksimal) {
    for (int i = word_3.size() - 1; i > -1; —i) {
        string probe = "";
        if (len > word.size() || len > i + 1)
            break;
        int j;
        for (j = i; j > i - len; —j)
            probe = word_3[j] + probe;

        if (slowar.find(probe) !=
            slowar.end()) {
            word_2 = "";
            vector<string> part_ =
                Sl(slowar[probe]);
            for (int k = 0; k < j; ++k)
                word_2 = word_2 + word_3[k];
            word_2 = word_2 + slowar[probe];
            for (int k = i + 1; k <
                word_3.size(); ++k)
                word_2 = word_2 + word_3[k];

            if (word_2.size() < word.size()
                && !flag) {
                sss.insert(word);
                sss.insert(word_2);
                return;
            }
            else if (sss.find(word_2) != sss
                && !flag)
                return;
            else if (flag && s.find(word_2)
                != s.end()) {
                if (kali[row].size() < size)

```

```

        kali[row].push_back(word_2);
        return;
    }
    else {
        int check = kali[row].size();
        New(word_2, flag, row, size);
        if (flag &&
            kali[row].size() > check)
            return;
    }
}
if (sss.find(word) !=
    sss.end() && !flag)
    return;

    if (flag &&
        s.find(word) != s.end()) {
        if (kali[row].size() <= size)
            kali[row].push_back(word);
        return;
    }
}
++len;
}

if (word_2.size() == word.size() &&
    word_2 < word && sss.find(word) == sss.end() &&
    sss.find(word_2) == sss.end() &&
    !flag) {
    s.insert(word_2);
    s_2.insert(word_2);
    sss.insert(word);
    sss.insert(word_2);
}

```

```

else if (!flag) {
    s.insert(word);
    s_2.insert(word_2);
    sss.insert(word);
    sss.insert(word_2);
}
}

void Zad3() {
    maksimal = 0;
    slowar.clear();
    s.clear();
    sss.clear();

    int n, m;
    cout << "Enter the number of elements of the alphabet:";
    cin >> n;
    cout << "Enter the alphabet: ";
    set<string> alf;
    for (int i = 0; i < n; ++i) {
        string a;
        cin >> a;
        alf.insert(a);
        s.insert(a);
        sss.insert(a);
    }
    cout << "Enter the number of constitutive relations:";
    cin >> m;
    cout << "Enter defining ratios:\n";
    for (int i = 0; i < m; ++i) {
        string l, r;
        cin >> l;
        cin >> r;
        if (l.size() > maksimal)

```



```

maksimal = l.size();
if (r.size() > maksimal)
maksimal = r.size();
if (l.size() < r.size())
slowar[r] = l;
else if (l.size() > r.size())
slowar[l] = r;
else {
    if (l < r)
        slowar[r] = l;
    else
        slowar[l] = r;
}
}

int count = 1;
s_1 = alf;
while (!s_1.empty()) {
    ++count;
    s_2.clear();
    for (string sit : s_1) {
        for (string buc : alf) {
            string sit_1 = sit + buc;
            word_2 = "";
            New(sit_1, false, 0, -1);
        }
    }
    s_1.clear();
    for (auto str : s)
        if (str.size() == count)
            s_1.insert(str);
}
cout << endl;
cout << "The resulting semigroup: {";

```

```

        for (string res : s)
            cout << res << " ";
        cout << "}\n";
    cout << endl;

    vector <string> gr;
    for (string str : s)
        gr.push_back(str);
    sort(gr.begin(), gr.end());
    int sizes = gr.size();

    kali.resize(sizes);
    for (int i = 0; i < sizes; i++)
        for (int j = 0; j < sizes; j++)
            New(gr[i] + gr[j], true, i, sizes);

    cout << endl << "Cayley table: \n";
    cout << "      ";
    for (int i = 0; i < sizes; i++)
        cout << setw(4) << gr[i];
    cout << endl;
    for (int i = 0; i < sizes; i++) {
        cout << setw(4) << gr[i] << setw(4);
        for (int j = 0; j < sizes; j++)
            cout << kali[i][j] << setw(4);
        cout << endl;
    }
    cout << endl;

}

int main() {
    setlocale(LC_ALL, "Russian");
    for (;;) {

```

```

    cout << "1 – Construct a subsemigroup
by the Cayley table
\n2 – Construct semigroups of binary
relations by given generating set\n";
    cout << "3 – Construct semigroups by generator
set and defining relations\n";
    int x;
    cin >> x;
    switch (x) {
        case 1:
            Zad1();
            cout << endl;
            break;
        case 2:
            Zad2();
            cout << endl;
            break;
        case 3:
            Zad3();
            cout << endl;
            break;
        case 4:
            break;
    }
}
}

```

## ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы были рассмотрены понятия полугруппы, подполугруппы, порождающего множества и подгруппы, порождающего множества и определяющих соотношения, и реализованы алгоритм построения полугруппы по таблице Кэли, алгоритм построения полугрупп бинарных отношений по заданному порождающему множеству, алгоритм построения полугруппы по порожденному множеству и определяющим соотношениям