Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Теоретическая информатика и компьютерные технологии»

ОТЧЁТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ

Студент _____
(Фамилия, Имя, Отчество)

Группа _____

Тип практики _____

Название предприятия Институт Программных Систем им. А.К. Айламазяна РАН

Студент _____ _____ _____
(Группа) (Подпись, дата) (И.О. Фамилия)

Рекомендуемая оценка _____

Руководитель практики
от предприятия _____ _____
(Подпись, дата) (И.О. Фамилия)

Руководитель практики _____ _____
(Подпись, дата) (И.О. Фамилия)

Оценка _____

2025 г.

СОДЕРЖАНИЕ

1 Характеристика предприятия 3

2 Индивидуальное задание 4

3 Теоретическая часть 5

4 Реализация 10

5 Отладка и тестирование 13

ЗАКЛЮЧЕНИЕ 15

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 16

ПРИЛОЖЕНИЕ А 17

1 Характеристика предприятия

Производственная практика проходила в Институте программных систем Российской академии наук (кратко — ИПС им. А.К. Айламазяна РАН). Институт Программных Систем был создан в апреле 1984 года как Филиал Института проблем кибернетики АН СССР по решению Правительства СССР, направленному на развитие вычислительной техники и информатики в стране. Руководителем ФИПК АН СССР был назначен д.т.н., профессор Альфред Карлович Айламазян. В 1986 году Филиал Института проблем кибернетики был преобразован в Институт программных систем АН СССР, а в 2008 году институту было присвоено имя его первого директора профессора А.К. Айламазяна.

С момента создания основными научными направлениями деятельности института являлись:

— высокопроизводительные вычисления,
— программные системы для параллельных архитектур,
— автоматизация программирования,
— телекоммуникационные системы и медицинская информатика.

Сегодня Институт программных систем имени А.К. Айламазяна РАН объединяет пять исследовательских центров:

— Исследовательский центр мультипроцессорных систем (ИЦМС),
— Исследовательский центр медицинской информатики (ИЦМИ Интерин),
— Исследовательский центр искусственного интеллекта (ИЦИИ),
— Исследовательский центр процессов управления (ИЦПУ),
— Исследовательский центр системного анализа (ИЦСА).

Практика проходила в исследовательском центре мультипроцессорных систем, в лаборатории автоматизации программирования.

2 Индивидуальное задание

Конечный моноид (моноид трансформаций) над алфавитом Σ определяется конечным множеством слов M и таблицей трансформаций, соответствующей действиям букв на элементы M.

Множество элементов моноида I является двухсторонним (левым, правым) идеалом, если $MIM = I (MI = I, IM = I)$.

По моноиду трансформаций построить множества его идеалов.

3 Теоретическая часть

Определение 1 (Конечный моноид). $(M, \cdot, 1)$ — *конечный моноид, если:*

1. M *- конечное множество;*
2. $\cdot$ *- замкнутая, ассоциативная, бинарная операция над* M*;*
3. $\forall x \in M : 1 \cdot x = x \cdot 1 = x$.

Определение 2 (Трансформация). *Трансформация множества* X — *отображение* $X \mapsto X$.

Определение 3 (Моноид трансформаций). *Пусть дано множество* M. *Множество трансформаций над* M, *замкнутое относительно операции* $\circ$ *(композиция), называется моноидом трансформаций над* M. *При этом единицей моноида трансформаций является эквивалентная трансформация.*

Определение 4 (Идеал). $I \subset M$ — *идеал моноида* M, *если* $MIM = I$. *Т.е.* $\forall x, y \in M, k \in I : xky \in I$.

Определение 5 (Левый идеал). $I \subset M$ — *левый идеал моноида* M, *если* $MI = I$. *Т.е.* $\forall x \in M, k \in I : xk \in I$.

Определение 6 (Правый идеал). $I \subset M$ — *правый идеал моноида* M, *если* $IM = I$. *Т.е.* $\forall x \in M, k \in I : kx \in I$.

Определение 7 (Алфавит). *Алфавит — конечное множество букв.*

Пример 1 (Алфавит). $\Sigma = \{a, b, c\}$

Определение 8 (Отношения Грина).

$$s \leq_{\mathcal{R}} t \Leftrightarrow \exists u \in S^1 : s = tu \tag{1}$$

$$s \leq_{\mathcal{L}} t \Leftrightarrow \exists u \in S^1 : s = ut \tag{2}$$

$$s \leq_{\mathcal{J}} t \Leftrightarrow \exists u, v \in S^1 : s = utv \tag{3}$$

$$s \leq_{\mathcal{H}} t \Leftrightarrow s \leq_{\mathcal{R}} t \quad \textit{and} \quad s \leq_{\mathcal{L}} t \tag{4}$$

Через идеалы:

$$s \leq_{\mathcal{R}} t \Leftrightarrow sS^1 \subseteq tS^1 \tag{5}$$

$$s \leq_{\mathcal{L}} t \Leftrightarrow S^1 s \subseteq S^1 t \tag{6}$$

$$s \leq_{\mathcal{J}} t \Leftrightarrow S^1 s S^1 \subseteq S^1 t S^1 \tag{7}$$

$$s \leq_{\mathcal{H}} t \Leftrightarrow s \leq_{\mathcal{R}} t \quad \textit{and} \quad s \leq_{\mathcal{L}} t \tag{8}$$

где sS^1 — правый идеал, порожденный элементом s (т.е. минимальный идеал, содержащий s).

Таблица трансформаций задает действие буквы на элементы множества M (слова). То есть эта таблица задает соответствие между буквами и трансформациями множества M.

Множество элементов моноида трансформаций можно построить с помощью замыкания по операции композиции и представить в виде графа Кэли, где вершины соответствуют трансформациям, а ребра соединяют вершины, если конечная вершина ребра может быть получена с помощью композиции транформаций соответствующих начальной вершины ребра и букве, которой помечают ребро.

При построении левого графа Кэли производится композиция транформации соответствущей букве и трансформации соответствующей вершине, а в правом графе Кэли наоборот: вершина к букве.

Пример 2. *Для $M = \{x, y, z\}$, и таблицы трансформаций:*

	a	b	c
x	x	z	z
y	y	y	z
z	x	x	z

Правый граф Кэли представлен на рисунке 1, левый граф Кэли procedure представлен на рисунке 2.

Так как в моноиде все элементы образуются с помощью композиции образующих, из определения идеала следует, что применение буквенной трансформации к элементу идеала должно давать в результате также элемент идеала. В терминологии графов Кэли это означает, что из вершин левого идеала в левом графе Кэли не должно быть ребер в вершины не принадлежащие иско-

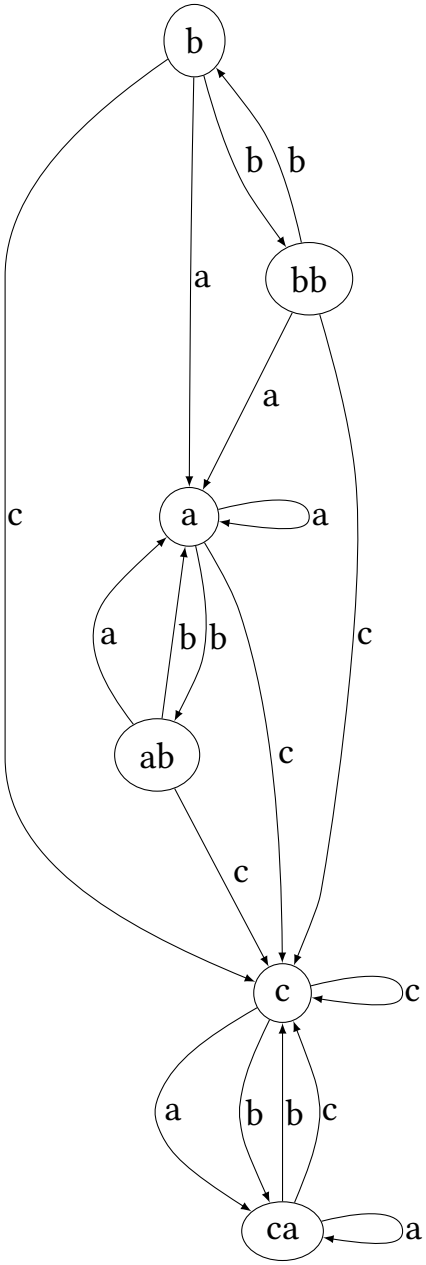

Рис. 1 — Правый граф Кэли для примера 2.

мому идеалу. Для вершин из правого идеала должно выполняться аналогичное условие в правом графе Кэли.

Таким образом, идеалами являются сочетания компонент связности графа Кэли, из вершин которых нет исходящих ребер. То есть для любой компоненты связности из идеала верно, что все достижимые из нее компоненты также принадлежат идеалу.

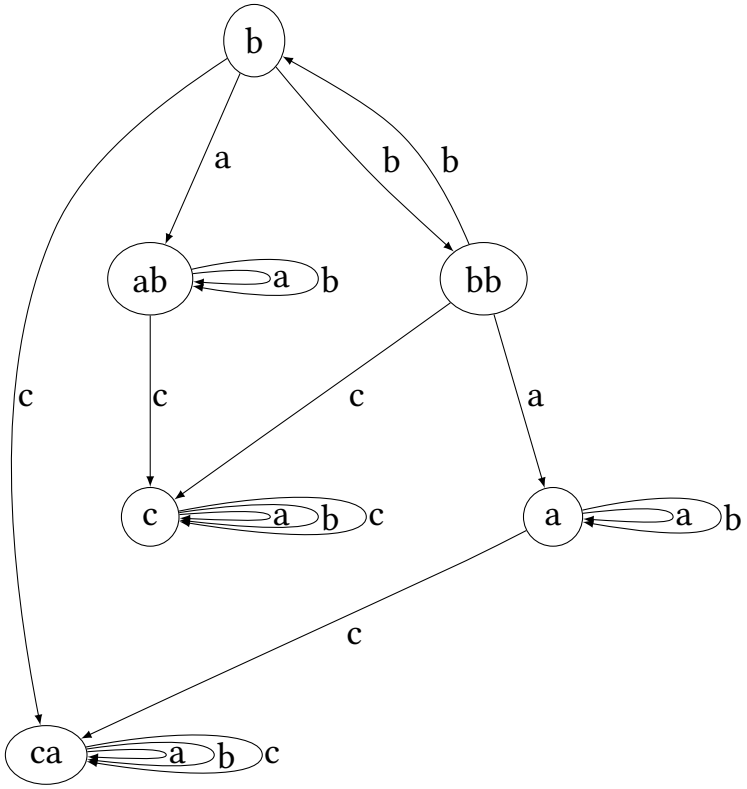

Рис. 2 — Левый граф Кэли для примера 2.

Интересно, что отношения Грина связаны с графами Кэли: $\mathcal{R}$-классы соответствуют компонентам связности в правом графе Кэли, а $\mathcal{L}$-классы — в левом [1].

Для построения множества идеалов необходимо найти конденсацию графа. Это делается с помощью двух запусков обхода в глубину (листинг 7) [2].

Построенный граф конденсации является направленным и ациклличным (рисунок 3). В таком графе нужно найти все сочетания компонент связности, для которых выполняется условие, что для каждой компоненты сильной связности в сочетание входят все достижимые из нее компоненты. Множества вершин для каждого сочетания образуют множество искомых идеалов соответствующего моноида трансформаций.

Такие сочетания можно найти пронумеровав вершины графа конденсации и рекурсивно сгенерировав все сочетания этих вершин от одного до $n = |G|$ элементов [3]. Причем, нужно исключить сочетания, в которых есть пара вершин, такая, что одна достижима из другой. После этого для каждого

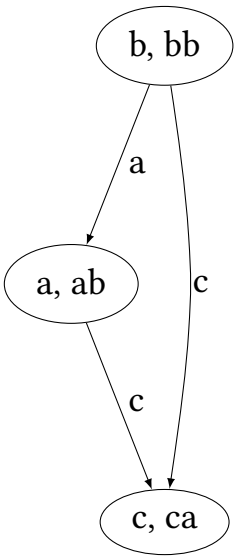

Рис. 3 — Граф конденсации для примера 2.

сочетания находим все достижимые из его элементов вершины, в резльтате получая идеалы. Алгоритм нахождения левых (правых) идеалов по графу конденсации представлен на листинге 1.

Листинг 1: Алгоритм нахождения идеалов по графу конденсации.

```
1:  procedure IDEALS(Condensation(V, E))
2:      prefix ← {}
3:      ideals ← {}
4:      Generate(prefix, ideals)
5:      return ideals
6:  function GENERATE(prefix, ideals)
7:      components ← (descendants of vertices in prefix)
8:      ideal ← {vertices of component  |  for component ∈ components}
9:      ideals ← ideal
10:     if |prefix| = |V| then
11:         return
12:     from ← |prefix|
13:     for i ∈ [from, |V|] do
14:         prefix.append(i)
15:         Generate(prefix, ideals)
16:         prefix.pop()
```

После получения множеств левых и правых идеалов, двусторонние идеалы находятся тривиальным пересечением множеств.

4 Реализация

Для реализации был выбран язык программирования C++, ввиду удобства его стандартной библиотеки для написания алгоритмов. Использован современный стандарт C++23, что позволило пользоваться библиотекой ranges для более декларативного кода.

Вершина графа хранится как структура, состоящая из слова, трансформации и списка смежности (листинг 2). Трансформация хранится как индексы элементов множества M.

Листинг 2: Объявление типов для хранения графа Кэли.

```
using Word = std::string;
using Transformation = std::vector<size_t>;
using LetterToTransformation = std::map<char,
 ↪  Transformation>;
using ElementIndex = size_t;
struct MonoidElement {
  std::string word;
  Transformation transformation;
  std::vector<ElementIndex> transitions;

  bool operator<=>(const MonoidElement& other) const =
   ↪  default;
};
using CayleyGraph = std::vector<MonoidElement>;
```

За построение графов Кэли отвечает класс `CayleyGraphBuilder` (листинг 3). Данный класс строит вершины графа с помощью замыкания по операции композиции. Для построения левого или правого графа Кэли предусмотрена передача в конструктор стратегий композиции и именования вершин, реализованная с помощью `std::function`. Использование такого паттерна является примером использования семантики значений в языке C++, что делает код проще и снижает вероятность ошибок.

Из реализации построения графа конденсации в отдельный класс выделен алгоритм топологической сортировки. За нахождение множеств идеалов ответственнен класс `IdealsBuilder` (листинг 4). Он строит левые или правые идеалы в зависимости от того, какой граф Кэли был передан в конструктор.

Листинг 3: Объявление класса для построения графов Кэли.

```
using CompositionStrategy
    = std::function<Transformation(const Transformation&
     ↪  lhs, const Transformation& rhs)>;
using WordCompositionStrategy
    = std::function<std::string(const MonoidElement&
     ↪  lhs, const MonoidElement& rhs)>;

class CayleyGraphBuilder {
public:
  CayleyGraphBuilder(const LetterToTransformation&
   ↪  letter_transformations,
                     CompositionStrategy
                      ↪  composition_strategy =
                      ↪  RightComposition,
                     WordCompositionStrategy
                      ↪  word_composition_strategy =
                      ↪  RightWordComposition);
  CayleyGraph Build();
private:
  void AddComposition(size_t element_index, size_t
   ↪  letter_index);
private:
  CayleyGraph monoid_elements_;
  std::map<Transformation, size_t>
   ↪  monoid_transformations_;
  CompositionStrategy composition_strategy_;
  WordCompositionStrategy word_composition_strategy_;
};
```

Перебор комбинаторных объектов реализован с помощью рекурсии, а нахождение множества достижимых из вектора вершин с помощью поиска в ширину.

Ввод производится из стандартного потока, но легко поддержать ввод из файла или строки. Вывод также производится в стандартный поток, а также функцией `VisualizeDot` записываются файлы с кодом на `dot`, которые содержат визуализацию левого и правого графов Кэли, что удобно для проверки.

Листинг 4: Объявление класса для построения левых или правых идеалов.

```
class IdealsBuilder {
public:
  IdealsBuilder(CayleyGraph monoid, CondensationGraph
   ↪  graph);
  std::vector<std::vector<ElementIndex>> Build();
private:
  std::vector<size_t>
   ↪  GetDescentants(std::vector<size_t> vertices);
  void Generate(std::vector<size_t>& prefix,
   ↪  std::vector<std::vector<ElementIndex>>& ideals);
private:
  CayleyGraph monoid_;
  CondensationGraph graph_;
  StronglyConnectedComponents scc_;
};
```

5 Отладка и тестирование

Модульные тесты реализованы с использованием библиотеки Google Tests.

Покрытие включает визуализацию, все алгоритмы и ввод.

Пример теста, который проверяет построение правого графа Кэли приведен на листинге 5. В данном тесте определяется таблица, задающая трансформации соответствующие буквам. Затем строится правый граф Кэли. После проверяется количество вершин в графе и трансформации, соответствующие добавленным вершинам.

Листинг 5: Тест, проверяющий построение правого графа Кэли.

```
TEST(MonoidBuilderTest,
 ↪  FindsClosureByTransformationComposition) {
  LetterToTransformation letter_transformations{
      {'a', {0, 1, 0}},
      {'b', {2, 1, 0}},
      {'c', {2, 2, 2}},
  };
  CayleyGraphBuilder
   ↪  builder(std::move(letter_transformations));

  auto got = builder.Build();

  ASSERT_THAT(got.size(), Eq(6));
  ASSERT_THAT(got[3].transformation,
   ↪  Eq(Transformation{2, 1, 2}));
  ASSERT_THAT(got[4].transformation,
   ↪  Eq(Transformation{0, 1, 2}));
  ASSERT_THAT(got[5].transformation,
   ↪  Eq(Transformation{0, 0, 0}));
}
```

Пример теста, который проверяет построение множество правых идеалов приведен на листинге 6. В этом тесте также сначала определяется таблица трансформаций. Затем строится правый граф Кэли и производится его конденсация. Затем по графу Кэли и графу конденсации находится множество правых идеалов и сравнивается с ожидаемым.

Листинг 6: Тест, проверяющий построение множества правых идеалов.

```
TEST(StrongConnectivityTest, FindRightIdeals) {
  LetterToTransformation letter_transformations{
      {'a', {0, 1, 0}},
      {'b', {2, 1, 0}},
      {'c', {2, 2, 2}},
  };
  auto monoid =
   ↪  CayleyGraphBuilder(letter_transformations).Build();
  auto condensation_graph =
   ↪  CondensationGraphBuilder(monoid).Build();

  auto right_ideals = IdealsBuilder(monoid,
   ↪  condensation_graph).Build();

  ASSERT_THAT(IndicesToWords(monoid, right_ideals),
              Eq(std::vector<std::vector<std::string>>{
                {"a", "b", "c", "ab", "bb", "ca"}, {"a",
                   ↪  "c", "ab", "ca"}, {"c", "ca"}}));
}
```

ЗАКЛЮЧЕНИЕ

В результате работы получена реализация алгоритмов на стыке алгебры и дискретной математики. Цели практики достигнуты: программа протестирована модульными тестами и соответствует заданию.

Выполняя данную работу, я подтянул знания алгебры и узнал о ее связи с дискретной математикой, используя книгу «Mathematical foundations of Automata Theory» [1]. Также мне выпала приятная возможность применить на практике фундаментальные алгоритмы из области генерации комбинаторных объектов и теории графов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Pin J.-É. Mathematical foundations of automata theory. — 2025. — (дата обращения: 15.06.2025).
2. Конспект ИТМО по нахождению компонент сильной связности. — URL: https://neerc.ifmo.ru/wiki/index.php?title=Использование_обхода_в_глубину_для_поиска_компонент_сильной_связности ; (дата обращения: 30.09.2025).
3. Конспект algocode по перебору комбинаторных объектов. — URL: https://wiki.algocode.ru/index.php?title=Перебор_комбинаторных_объектов ; (дата обращения: 30.09.2025).

ПРИЛОЖЕНИЕ А

Листинг 7: Построение графа конденсации.

```
1:  procedure CONDENSATION(G(V, E))
2:      used ← false   ∀v ∈ V
3:      order ← {}
4:      for v ∈ V do
5:          if used[v] = false then
6:              DFS1(v)
7:      G^T ← Transposed(G)
8:      used ← false   ∀v ∈ V
9:      components ← {}
10:     for v ∈ Reversed(order) do
11:         if used[v] = false then
12:             comp ← {}
13:             DFS2(v, comp)
14:             components ← comp
15:     C ← ({}, {})
16:     for c_i ∈ components do
17:         C.V ← c_i
18:     for (u, v) ∈ E do
19:         c_u ← Component with u
20:         c_v ← Component with v
21:         if c_u ≠ c_v then
22:             C.E ← (c_u, c_v)
23:     return C
24: function DFS1(v)
25:     used[v] ← true
26:     for u ∈ G.adj[v] do
27:         if used[u] = false then
28:             DFS1(u)
29:     order ← v
30: function DFS2(v, comp)
31:     used[v] ← true
32:     comp ← v
33:     for u ∈ G^T.adj[v] do
34:         if used[u] = false then
35:             DFS2(u, comp)
```