Министерство образования и науки Российской Федерации  
Федеральное‌ ‌государственное‌ ‌бюджетное‌ ‌образовательное‌ ‌учреждение‌

высшего‌ ‌образования‌

**«Пермский национальный исследовательский  
политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЁТ**

**по лабораторной работе №7**

Дисциплина: «Дискретная математика и математическая

логика»

Тема: Теория автоматов

Выполнил работу

студент группы ИВТ-22-2б

Казанцев А.В.

Проверила

Старший преподаватель кафедры ИТАС

Рустамханова Г.И.

Пермь 2024

**Цель работы**

1. Создание консольного приложения «Конечный автомат».

2. Изучение теории дискретной математики об свойствах автоматов.

3. Изучение и отработка навыков работы с детерминированными конечными автоматами.

**Постановка задачи**

Программа должна позволять пользователям вводить слова в конечный автомат с клавиатуры.

Программа должна выполнять следующие функции:

1. Построение автомата: Создать конечный автомат на основе заданного регулярного выражения или таблицы переходов.
2. Распознавание языка: Определите, принимается ли данная строка конечным автоматом.

**Анализ задачи**

Формальный язык задан в алфавите и содержит слова любой длины.  


|  |  |
| --- | --- |
| 9 | в которых НЕТ подслова «*abc*» |

Проведем анализ и определим количество необходимых состояний конечного автомата.

Q0 – это первое начальное состояние в него можно попасть с самого начала и из 2 состояний ниже и самого себя при входных параметрах . Может быть конечным состоянием.

Q1 – состояние в которое можно попасть из начального Q0 и самого Q1 при входном параметре . Может быть конечным состоянием.

Q2 – состояние в которое можно попасть только из Q1 при входном параметре . Может быть конечным состоянием.

QE – состояние ошибка. В это состояние можно попасть из любого состояния если на вход попадает символ не из алфавита. Также в это состояние можно попасть из Q2 если на вход поступает . Из этого состояния нет выхода. Не является конечным состоянием.

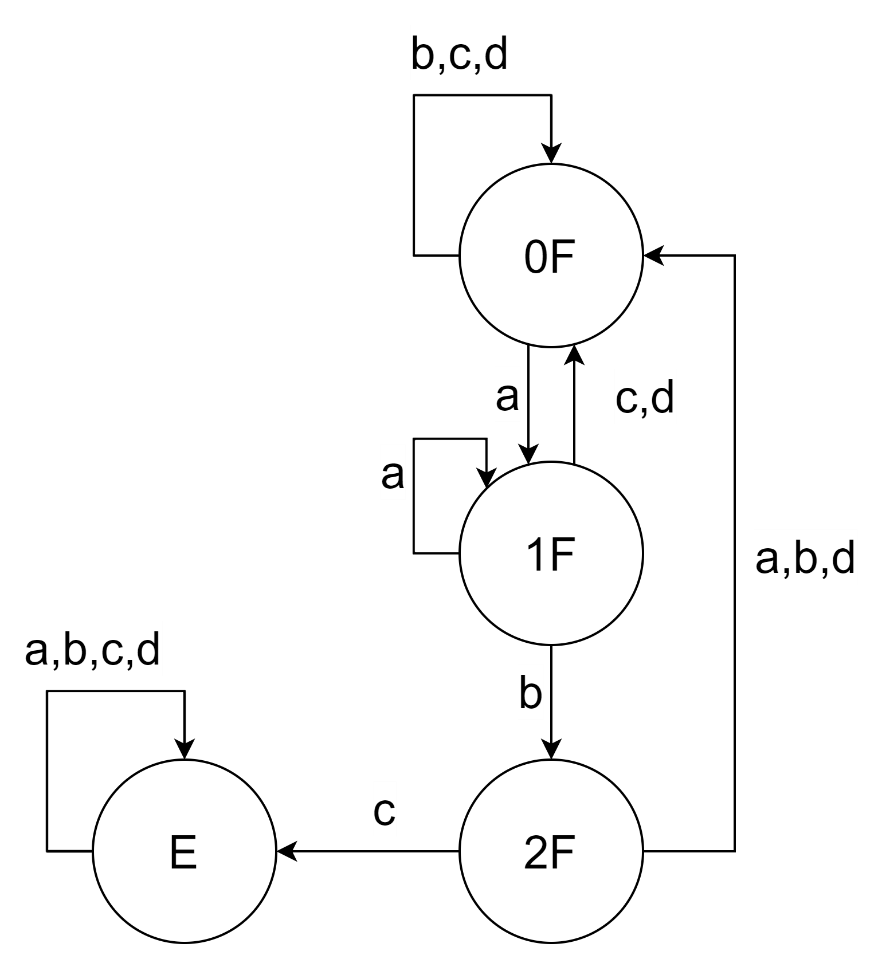
88

Рисунок 1 — Частичный автомат

Таблица 1 - Таблица переходов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ***q0*** | ***q1*** | ***q2*** | ***qE*** |
| ***a*** | *q1* | *q1* | *q0* | *qE* |
| ***b*** | *q0* | *q2* | *q0* | *qE* |
| ***c*** | *q0* | *q0* | *qE* | *qE* |
| ***d*** | *q0* | *q0* | *q0* | *qE* |

**Ход работы**

Программа имеет возможность создать конечный автомат на основе таблицы переходов или схемы Мура. Она принимает алфавит и переходы в состояния

Класс FiniteAutomaton реализовывает модель конечного автомата. Он состоит из начального состояния, хеш-множества конечных состояний для быстрого поиска и переходной таблицы.  
Словарь хранит входное состояние и словарь хранящий символ и конечное состояние. Класс инициализируется алфавитом языка массивом с символов.

public void AddTransition(int state, char input, int nextState) – функция добавляющая переход в таблицу переходов.

public bool Accepts(string word) – функция возвращает значение если после выполнения работы автомата он оказался в одном из финальных состоянии.

public static void Check(FiniteAutomaton fa, string word) – функция выводит результат прохождения слова в автомате в консоль

Тесты

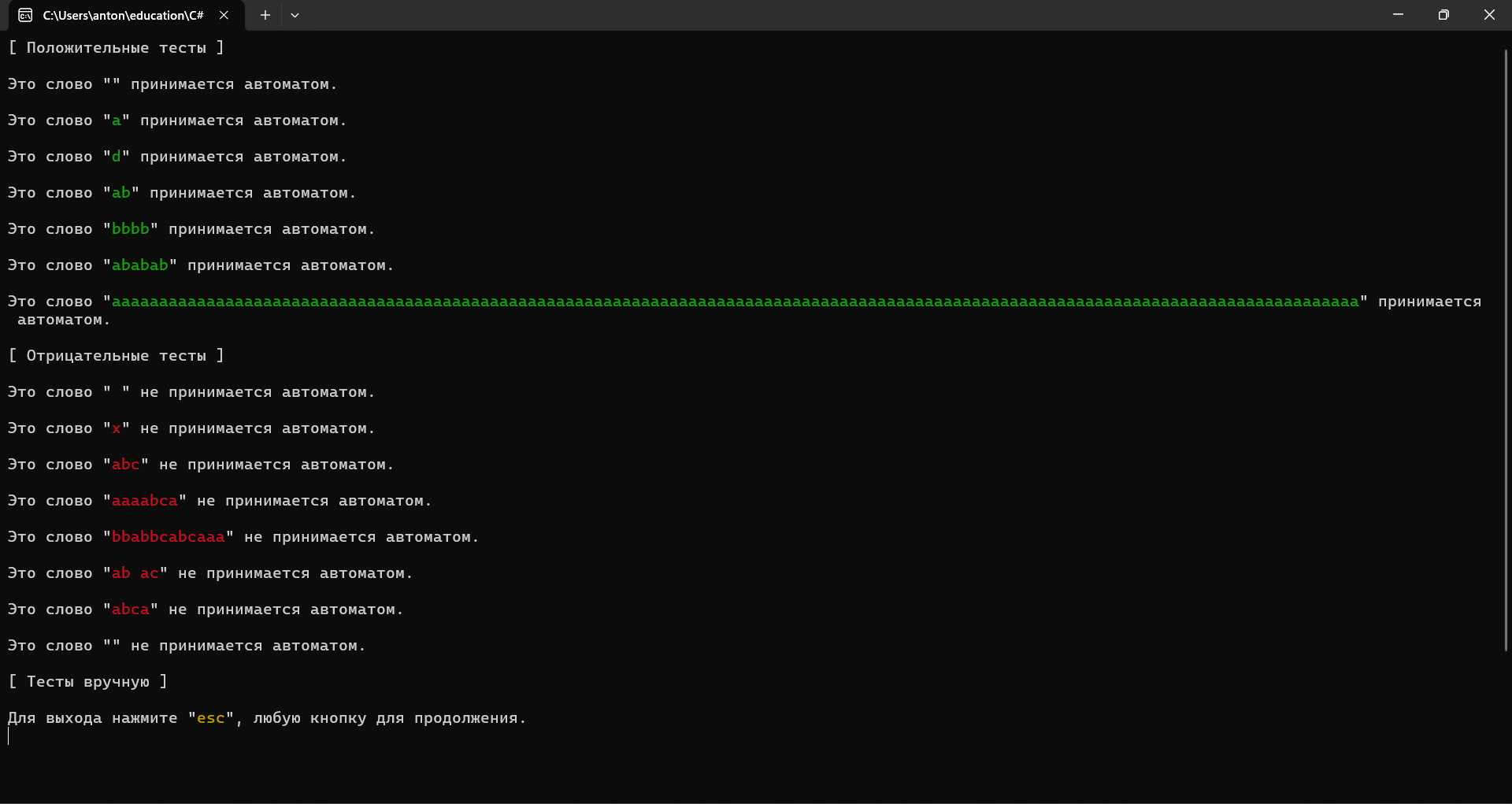


Рисунок 2 — Тесты

**Код программы**

namespace FiniteAutomaton

{

class Program

{

static void Main(string[] args)

{

#region Насторойка автомата

// Алфавит

char[] alphabet = { 'a', 'b', 'c', 'd' };

FiniteAutomaton fa = new FiniteAutomaton(alphabet);

// Определите переход в состояние

fa.AddTransition(0, 'a', 1);

fa.AddTransition(0, 'b', 0);

fa.AddTransition(0, 'c', 0);

fa.AddTransition(0, 'd', 0);

fa.AddTransition(1, 'a', 1);

fa.AddTransition(1, 'b', 2);

fa.AddTransition(1, 'c', 0);

fa.AddTransition(1, 'd', 0);

fa.AddTransition(2, 'a', 0);

fa.AddTransition(2, 'b', 0);

fa.AddTransition(2, 'd', 0);

// Установите начальное и конечное состояния

fa.InitialState = 0;

fa.FinalStates.Add(0);

fa.FinalStates.Add(1);

fa.FinalStates.Add(2);

#endregion

#region Демонстрационные тесты

Console.WriteLine("[ Положительные тесты ]\n");

Check(fa, "");

Check(fa, "a");

Check(fa, "d");

Check(fa, "ab");

Check(fa, "bbbb");

Check(fa, "ababab");

Check(fa, "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa");

Console.WriteLine("[ Отрицательные тесты ]\n");

Check(fa, " ");

Check(fa, "x");

Check(fa, "abc");

Check(fa, "aaaabca");

Check(fa, "bbabbcabcaaa");

Check(fa, "ab ac");

Check(fa, "abca");

Check(fa, null!);

#endregion

Console.WriteLine("[ Тесты вручную ]\n");

Console.WriteLine("Для выхода нажмите \"\x1B[33mesc\x1B[0m\", любую кнопку для продолжения.");

while (Console.ReadKey(true).Key !=ConsoleKey.Escape)

{

Console.WriteLine("Введите слово:");

Check(fa, Console.ReadLine()!);

Console.WriteLine("Для выхода нажмите \"\x1B[33mesc\x1B[0m\"");

}

}

public static void Check(FiniteAutomaton fa, string word)

{

bool accepted = fa.Accepts(word);

Console.WriteLine($"Это слово {(accepted ? $"\"\x1B[32m{word}\x1B[0m\"" : $"\"\x1B[31m{word}\x1B[0m\"")} {(accepted ? "" : "не ")}принимается автоматом.\n");

}

}

class FiniteAutomaton

{

private int initialState;

private HashSet<int> finalStates;

private Dictionary<int, Dictionary<char, int>> transitionFunction;

public FiniteAutomaton(char[] alphabet)

{

transitionFunction = new Dictionary<int, Dictionary<char, int>>();

finalStates=new HashSet<int>();

foreach (char c in alphabet)

{

transitionFunction.Add(c, new Dictionary<char, int>());

}

}

public int InitialState

{

get { return initialState; }

set { initialState = value; }

}

public HashSet<int> FinalStates

{

get { return finalStates; }

set { finalStates = value; }

}

public void AddTransition(int state, char input, int nextState)

{

if (!transitionFunction.ContainsKey(state))

{

transitionFunction.Add(state, new Dictionary<char, int>());

}

transitionFunction[state].Add(input, nextState);

}

public bool Accepts(string word)

{

if (word is null) return false;

int currentState = initialState;

foreach (char c in word)

{

if (!transitionFunction[currentState].TryGetValue(c, out int nextState))

{

return false;

}

currentState = nextState;

}

return finalStates.Contains(currentState);

}

}

}