Филиал Московского Государственного Университета имени М.В. Ломоносова в г. Ташкенте

Факультет прикладной математики и информатики

Кафедра прикладной математики и информатики

Зияев Азизбек Бекзодович

КУРСОВАЯ РАБОТА

на тему: «Компьютерный симулятор автоматов в лабиринтах. Редактор автомата.»

по направлению 01.03.02 «Прикладная математика и информатика»

Курсовая работа рассмотрена и рекомендована к защите

руководитель филиала, доцент _____ Строгалов А. С.

Научный руководитель:

к.ф.-м.н. _____ Волков Н. Ю.

«____»_____ 2020 г.

Ташкент 2020

Аннотация

В данной работе рассматривается симулятор инициального конечного автомата в шахматном лабиринте. Симулируется поведение автомата на плоскости, задаваемого таблицей переходов.

Реализован редактор автомата, а также удобный пользовательский интерфейс. Реализация осуществлялась возможностями языка C#.

Содержание

1 Введение

В последнее время большое внимание уделяется изучению проблемы поведения автоматов в лабиринтах. Наблюдение за движением автоматов в лабиринтах даже в случае, когда входной алфавит, выходной алфавит и алфавит состояний небольшие по объему, является сложным, поэтому естественным образом возникает потребность моделирования поведения автоматов в лабиринтах при помощи компьютера. Для этой цели разработана программа, в которой создание лабиринта, автомата и симулирование его поведения на заданном лабиринте становятся более чем реальными. Эта программа позволяет создавать лабиринты, которые в дальнейшем будут использоваться для обхода заданным автоматом. Работая в автономном режиме, программа позволяет рисовать лабиринт любой сложности, находить кратчайший путь между двумя точками, задавать автомат, который в дальнейшем сможет обойти данный лабиринт. Для произвольно заданного лабиринта обеспечивается визуальное наблюдение за поведением автомата в нем. Программа может быть использована в качестве удобного "вспомогательного инструмента"для решения проблем, связанных с поведением автоматов в лабиринтах. Так, например, программа в дальнейшем может быть очень полезна в процессе поиска универсальных автоматов для заданного класса лабиринтов.

В настоящей работе рассматривается реализация инициального конечного автомата и алгоритм поиска кратчайшего пути.

2 Основные определения

Под *автоматом* будем понимать инициальный конечный автомат вида $\mathcal{A} = (A, Q, B, \varphi, \psi, q_0)$, где A — входной, B — выходной, Q — внутренний алфавит автомата $\mathcal{A}$, $\varphi : Q \times A \to Q$ и $\psi : Q \times A \to B$ — функции переходов и выходов $\mathcal{A}$, соответственно, $q_0 \in Q$ — его начальное состояние. Алфавит A определяет возможности $\mathcal{A}$ «видеть» происходящее вокруг, а алфавит B — его возможности перемещаться. Алфавит Q и функции φ и ψ задают внутреннюю логику автомата $\mathcal{A}$. Выходным алфавитом A является множество $B = D_{(0,0),V}$, где параметр $V \in N$ называется *скоростью автомата* $\mathcal{A}$. Входной алфавит $\mathcal{A}$ зависит от параметра $R \in N(RV)$, называемого *обзором автомата*

Способы задания автомата

1. Таблица.

2. Диаграмма Мура.

3. Каноническое уравнение.

4. Схема.

1. Автомат заданный таблицей.

$$A = \{0, 1\}, B = \{0, 1\}, Q = \{q_1, q_2, q_3\}$$

A	Q	φ	ψ
0	q_1	q_2	1
1	q_1	q_2	0
0	q_2	q_3	0
1	q_2	q_2	1
0	q_3	q_1	0
1	q_3	q_1	1

Под *шахматным лабиринтом* L назовем множество, вершины которого составляют произвольное связное подмножество клеток из $\mathbb{Z}^2$. Это означает, что для любых клеток лабиринта $(x_1, y_1), (x_k, y_k)$ существует путь: $(x_1, y_1), (x_2, y_2)...(x_k, y_k)$ по клеткам подмножества L, такой что любые подряд идущие клетки этого пути являются соседними, то есть $p((x_i, y_i), (x_{i+1}, y_{i+1})) = 1$. Будем изображать шахматный лабиринт на плоскости $\mathbb{Z}^2$ следующим образом: клетки, лежащие в лабиринте, будут белыми, а клетки, не лежащие в лабиринте, черными.

Манхэттенской метрикой на плоскости назовем расстояние

$$\rho((x_1, y_1), (x_2, y_2)) = |x_1 - x_2| + |y_1 - y_2|.$$

Направления движения автомата в шахматном лабиринте соответствуют сторонам света — w, n, e, s.Поставим в соответствие каждой стороне единичный вектор: (-1, 0), (0, 1), (1, 0), (0, -1).

Определим перемещение автомата в шахматном лабиринте.[2]

Автомат $\mathcal{A}$ начинает перемещение в лабиринте L. Фиксируется начальное (в нулевой момент времени) расположение автомата в белой клетке лабиринта. В каждый момент времени $\mathcal{A}$ воспринимает в качестве входного символа допустимые ходы $a(t) \in A$ в формате — w, n, e, s. Автомат $\mathcal{A}$, зная свое текущее состояние $q(t) \in Q$, в соответствии со своими функциями переходов, выходов, определяет свое следующее состояние $q(t+1) \in Q$, выходной символ $b(t) \in B$ и перемещается на вектор $\overline{b}$, то есть

$$\begin{cases} x(t+1) = x(t) + b_x(t) \\ y(t+1) = y(t) + b_y(t) \\ q(t+1) = \varphi(q(t),\ a(t)) \\ b(t) = (b_x(t),\ b_y(t)) = \psi(q(t),\ a(t)) \end{cases}$$

Поведение автомата $\mathcal{A}$ в лабиринте может явно задаваться таблицей вида:

A	Q	φ	ψ
$a(t_i)$	$q(t_i)$	$q(t_i+1)$	$b(t_i)$
$a(t_i+1)$	$q(t_i+1)$	$q(t_i+2)$	$b(t_i+1)$
$\ldots$	$\ldots$	$\ldots$	$\ldots$
$a(t_j)$	$q(t_j)$	$q(t_j+1)$	$b(t_j)$

3 Описание автомата в программе

Внутреннее представление автомата в программе осуществляется с помощью словаря. Словарь в свою очередь хранит ключ и значение. Ключ представляет собой входной символ $a(t) \in A$, а значение задается кортежем из выходного символа $b(t) \in A$ и следующего состояния $q(t+1) \in Q$.

Условно программа разделена на блоки: блок с логикой — назовем его *model* и блок с объединением визуальной части — *view model*.

3.1 Model

Инициализация автомата происходив в *model*, блок также защищен от некорректных данных. Для взаимодействия с автоматом имеется удобный программный интерфейс.

Блок состоит из пяти классов:

* AIO

* State

* Range

* MooreDiagram

* Automaton

Класс *AIO* служит для записи входных данных автомата и подачи в автомат или автомат записывает в этот класс выходные данные и возвращает их.

Класс *State* определяет состояние автомата. Из него исходят «стрелки» в другие состояния. Эти «стрелки» или переходы записаны в словарь.

Класс *Range* служит для описания диапазона значений для элемента во входном символе.

Например $Range[]$ задает алфавит только для одного элемента. $alphabet = [Range[0, 15], Range[-1, 1], Range[0, 4]]$. Для такого алфавита удовлетворяют только следующие $AIO\ symbol = [a_1, a_2, a_3]$, где $0 \leq a_1 \leq 15$, $-1 \leq a_2 \leq 1$, $0 \leq a_3 \leq 4$.

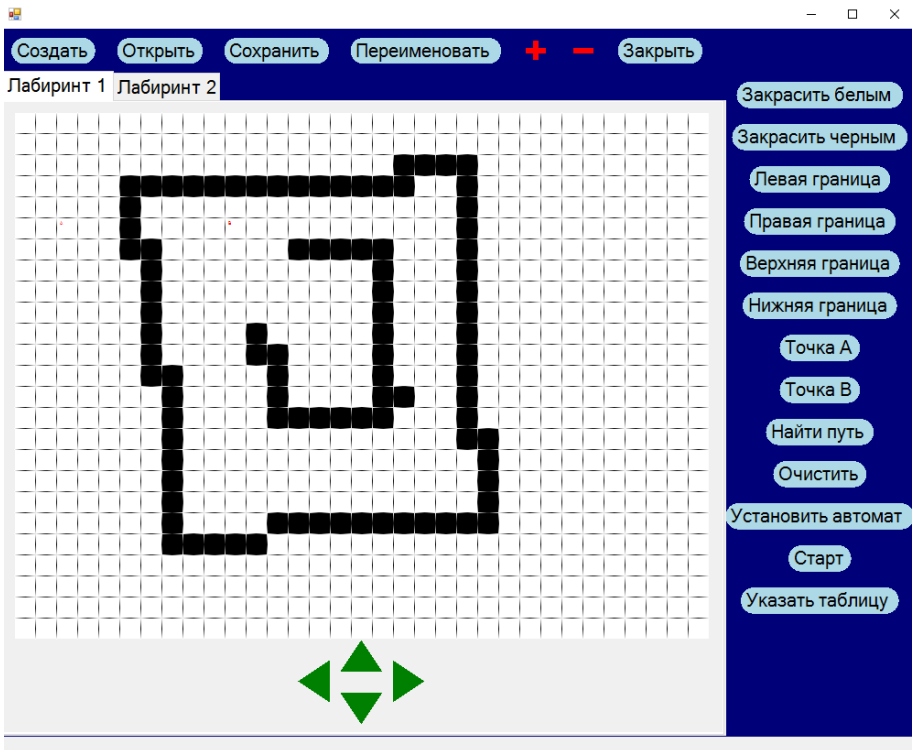
Класс $MooreDiagram$ инициализирует автомат, хранит список состояний автомата Q, а также использует остальные классы и их методы для редактирования автомата.

Класс $Automaton$ хранит в себе текущее местоположение на плоскости, а так же используется для связи с $view\ model$ блоком.

3.2 View Model

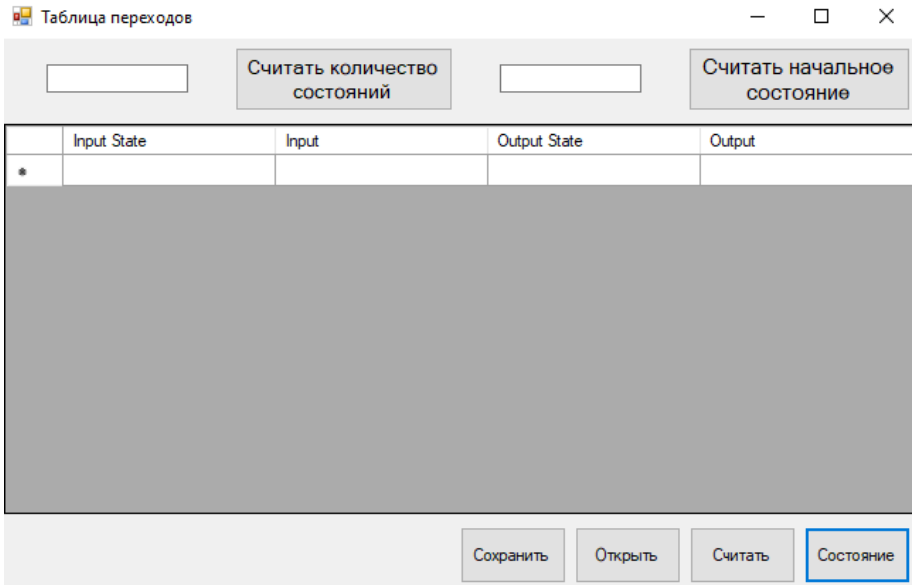
Данный блок был написан и внедрен в уже имеющийся пользовательский интерфейс, а именно редактор лабиринтов[1].

Был добавлен новый функционал для пользователя в виде кнопок «Установить автомат», «Старт», «Указать таблицу».



Установка автомата происходит на белой клетке шахматного лабиринта. В это время автомат инициализируется. Для того, чтобы задать таблицу была дополнительно создана форма «Таблица переходов».

Таблица переходов выполняет функцию считывания таблицы в инструмент *dataGridView*. Есть возможность открыть таблицу из файла, а также можно всегда сохранить свою таблицу в файл. Это предоставляет пользователю легко загружать таблицы и делиться ими с другими пользователями.



4 Инструкция к пользованию

1. **Установка автомата**

   В начале работы с автоматом необходимо задать стартовое положение для автомата. Это делается кнопкой «Установить автомат». Следует нажать на кнопку и курсором мыши щелкнуть на клетку, на поле лабиринта.

2. **Таблица**

   После установки автомата на поле, его необходимо инициализировать. Это делается с помощью кнопки «Указать таблицу». Нажав на нее мы видим новое окно, которое имеет свой функционал. Снизу мы видим несколько кнопок: кнопки «Открыть» и «Сохранить» позволяют соответственно открыть таблицу из файла и сохранить в текстовый файл.

   В окне таблицы следует вначале установить количество состояний и начальное состояние. Это можно сделать в верхней части программы. Указав количество и начальное состояние необходимо нажать на соответствующие кнопки. Теперь можно приступать к заполнению таблицы. Для того, чтобы задать таблицу необходимо заполнить её в формате:

   $$q_i(t)|a(t)|q_i(t+1)|b(t)$$

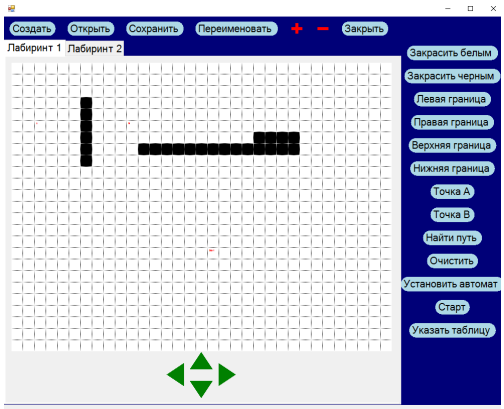
   После заполнения таблицы нужно нажать на кнопку «Считать».

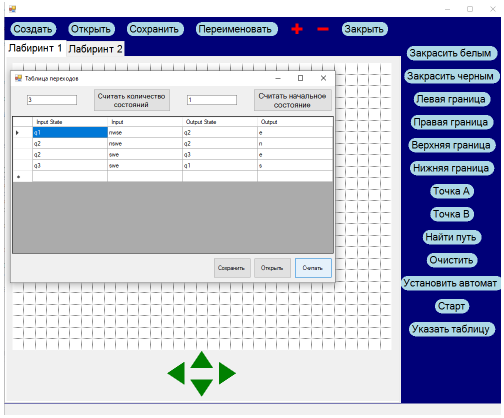
3. **Старт**

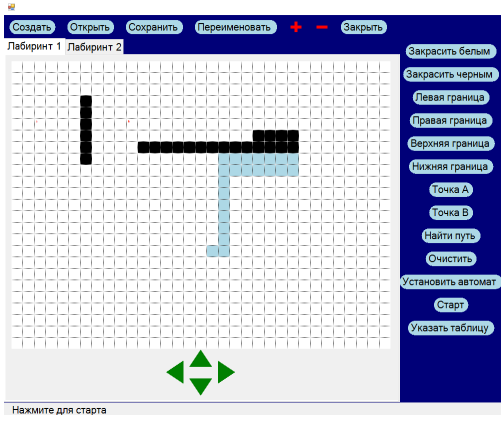
   Когда мы считали таблицу, можно переходить в основное окно для дальнейшей работы.

   Нажимая на кнопку «Старт» автомат будет распознавать входные данные и двигаться в лабиринте согласно своей таблице переходов.

Пример использования:







5 Алгоритм Дейкстры

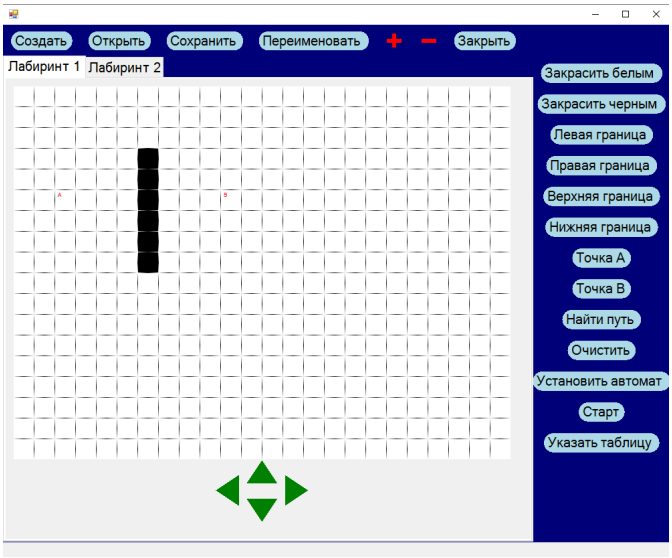
Алгоритм на графах был написан и внедрен в программу редактора лабиринта ещё на ранних этапах создания редактора. Данный алгоритм позволял протестировать лабиринт на корректность.

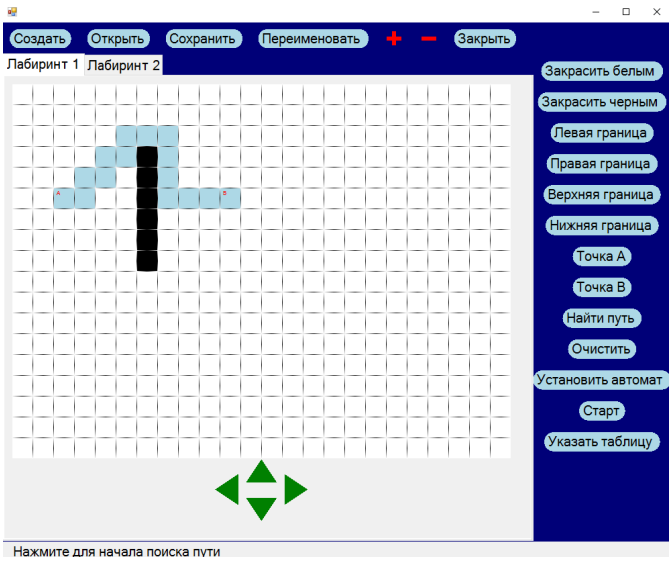
Инструкция к пользованию:

Для запуска поиска пути установите в лабиринте начальную точку (A) и конечную точку (B) при помощи кнопок из правой панели. После этого нажмите кнопку «Поиск пути».

Результаты поиска можно очистить кнопкой «Очистить».

Пример использования:





6 Заключение

В будущем планируется увеличить функционал программы, а также перестройка архитектуры программы для четкого разделения блоков программы (использование паттерна $MVVM$).

Будет произведено добавление таких функций как:

1. Моделирование нескольких автоматов. Например моделирование преследования автомата волка автомата зайца. Моделирование независимой системы автоматов и коллектива.

2. Композиция автоматов. Программа будет уметь моделировать автомат полученный из композиции других автоматов заданных таблицей.

3. Фиксация автоматом факт обхода лабиринта.

4. Мультиплатформенность. Возможность запускать программу на нескольких платформах является современной потребностью пользователей.

5. Язык предикатов для коллектива автоматов.

Также дальнейшее улучшение и оптимизация работы программы, для более удобного пользования.

Автор выражает благодарность Н.Ю. Волкову за научное руководство.

7 Приложение

Здесь показывается блок *model* с логикой

1. Класс *AIO*

```
34 references
public class AIO
{
    11 references
    public int[] Data { get; private set; }
    2 references
    public int DataLength { get => Data.Length; }

    4 references
    public AIO(params int[] data)
    {
        Data = data;
    }

    2 references
    public override bool Equals(object obj)
    {
        if (obj == null)
            return false;
        if (ReferenceEquals(this, obj))
            return true;
        if (!obj.GetType().Equals(typeof(AIO)))
            return false;

        return (obj as AIO).Data.SequenceEqual(Data);
    }

    2 references
    public override int GetHashCode()
    {
        return Hash.GetHashCode(Data);
    }

    4 references
    public override string ToString()
    {
        return "[" + string.Join(", ", Data) + "]";
    }
}
```

2. Класс *State*

```
22 references
internal class State
{
    17 references
    public string Id { get; }
    7 references
    public Dictionary<AIO, (AIO, State)> Table { get; }

    2 references
    public State(string id, Dictionary<AIO, (AIO, State)> table)
    {
        Id = id;
        Table = table;
    }



    1 reference
    public (AIO, State) Process(AIO input)
    {
        if (!Table.ContainsKey(input))
            throw new ArgumentException($"Состояние с Id={Id} не содержит выходящей из него стрелки с входом={input}");

        return Table[input];
    }

    2 references
    public override bool Equals(object obj)
    {
        if (obj == null)
            return false;
        if (ReferenceEquals(this, obj))
            return true;
        if (!obj.GetType().Equals(typeof(State)))
            return false;

        return (obj as State).Id == Id;
    }

    4 references
    public override string ToString()
```

3. Класс *Range*

```
12 references
public class Range
{
    5 references
    public int Begin { get; }
    5 references
    public int End { get; }

    2 references
    public Range(int begin, int end)
    {
        if (begin > end)
            throw new ArgumentException("begin > end", "end");
        Begin = begin;
        End = end;
    }

    1 reference
    public bool Contains(int value)
    {
        return Begin <= value && value <= End;
    }

    4 references
    public override string ToString()
    {
        return $"[{Begin}, {End}]";
    }

    2 references
    public override int GetHashCode()
    {
        return this.ToString().GetHashCode();
    }

    2 references
    public override bool Equals(object obj)...
}
```

4. Класс *MooreDiagram*

```
public class MooreDiagram
{
    private State currentState;
    4 references
    public Range[] InputAlphabet { get; }
    3 references
    public Range[] OutputAlphabet { get; }

    private List<State> allStates;

    1 reference
    public MooreDiagram(Range[] inputAlphabet, Range[] outputAlphabet)
    {
        InputAlphabet = inputAlphabet;
        OutputAlphabet = outputAlphabet;
        currentState = null;
        allStates = new List<State>();
    }

    0 references
    public int InputLength { get => InputAlphabet.Length; }
    0 references
    public int OutputLength { get => OutputAlphabet.Length; }
    0 references
    public string CurrentState { get => currentState.Id; }

    1 reference
    public void SetInitState(string id)
    {
        if (currentState == null)
        {
            currentState = new State(id, new Dictionary<AIO, (AIO, State)>());
            allStates.Add(currentState);
        }
        else
        {
            State curState = allStates.Find(x => x.Id == id);
            if (curState == null) throw new ArgumentException($"Состояние с Id={Id} не найдено!");
            currentState = curState;
        }
    }
```

```
    1 reference
    public void AddNewState(string id)
    {
        if (allStates.Find(x => x.Id == id) != null)
            throw new ArgumentException($"Состояние с Id={id} уже существует", id);
        State s = new State(id, new Dictionary<AIO, (AIO, State)>());
        allStates.Add(s);
    }

    1 reference
    public bool Check_transition(string state, AIO input)
    {
        State curstate = allStates.Find(x => x.Id == state);
        if (curstate == null) return false;
        else return curstate.Table.ContainsKey(input);
    }

    1 reference
    public void AddNewTransition(string curStateId, AIO input, string nextStateId, AIO output)
    {
        if (currentState == null)
            throw new InvalidOperationException("Автомат ещё не инициализирован.");
        if (allStates.Find(x => x.Id == curStateId) == null)
            throw new ArgumentException($"Состояния с Id={curStateId} не существует.", "curStateId");
        if (allStates.Find(x => x.Id == nextStateId) == null)
            throw new ArgumentException($"Состояния с Id={nextStateId} не существует.", "nextStateId");
        if (!doesAIOSatisfyAlphabet(input, InputAlphabet))
            throw new ArgumentException("Не все входные данные находятся в требуемом алфавите.", "input");
        if (!doesAIOSatisfyAlphabet(output, OutputAlphabet))
            throw new ArgumentException("Не все выходные данные находятся в требуемом алфавите.", "output");
        if (!check(input, output))
            throw new ArgumentException("Невозможно попасть из текущего расположения");

        State curState = allStates.Find(x => x.Id == curStateId);
        State nextState = allStates.Find(x => x.Id == nextStateId);

        curState.Table.Add(input, (output, nextState));

    }
```

```
}

1 reference
public AIO Process(AIO input)
{
    if (currentState == null)
        throw new InvalidOperationException("Автомат ещё не инициализирован.");
    if (!doesAIOSatisfyAlphabet(input, InputAlphabet))
        throw new ArgumentException("Автомат не распознаёт эти входные данные.");

    var output = currentState.Process(input);
    currentState = output.Item2;
    return output.Item1;
}

3 references
private bool doesAIOSatisfyAlphabet(AIO aio, Range[] alphabet)
{
    if (aio.DataLength != alphabet.Length)
        return false;

    for (int i = 0; i < aio.DataLength; i++)
    {
        if (!alphabet[i].Contains(aio.Data[i]))
            return false;
    }

    return true;
}
```

8 Список использованной литературы

[1] В. Б. Кудрявцев, С. В. Алешин, А. С. Подколзин «Введение в теорию автоматов», Москва, Наука, 1985.

[2] Г. Килибарда, В. Б. Кудрявцев, Ш. М. Ушчумлич, «Независимые системы автоматов в лабиринтах», Дискрет. матем.

[3] Г. Килибарда, В. Б. Кудрявцев, Ш. М. Ушчумлич, «Коллективы автоматов в лабиринтах», Дискрет. матем.

[4] Н. Ю. Волков «Об автоматной модели преследования», Дискретная математика, т.19, вып.2. 2007 г.

[5] Wikipedia. Алгоритм Дейкстры.