Министерство науки и высшего образования Российской

Федерации

Федеральное государственное бюджетное образовательное

учреждение высшего образования

"Московский авиационный институт

(национальный исследовательский университет)"

Отчет

по лабораторной работе

по дисциплине "Теория сложности алгоритмов"

Выполнил студент
группы М8О-102М-22
Москаленко С.С.

Москва 2023

Вариант № 7.

7. $B = \{w : w$ не содержит 0 в три раза меньше, чем $1\}$;

Диаграмма состояний.

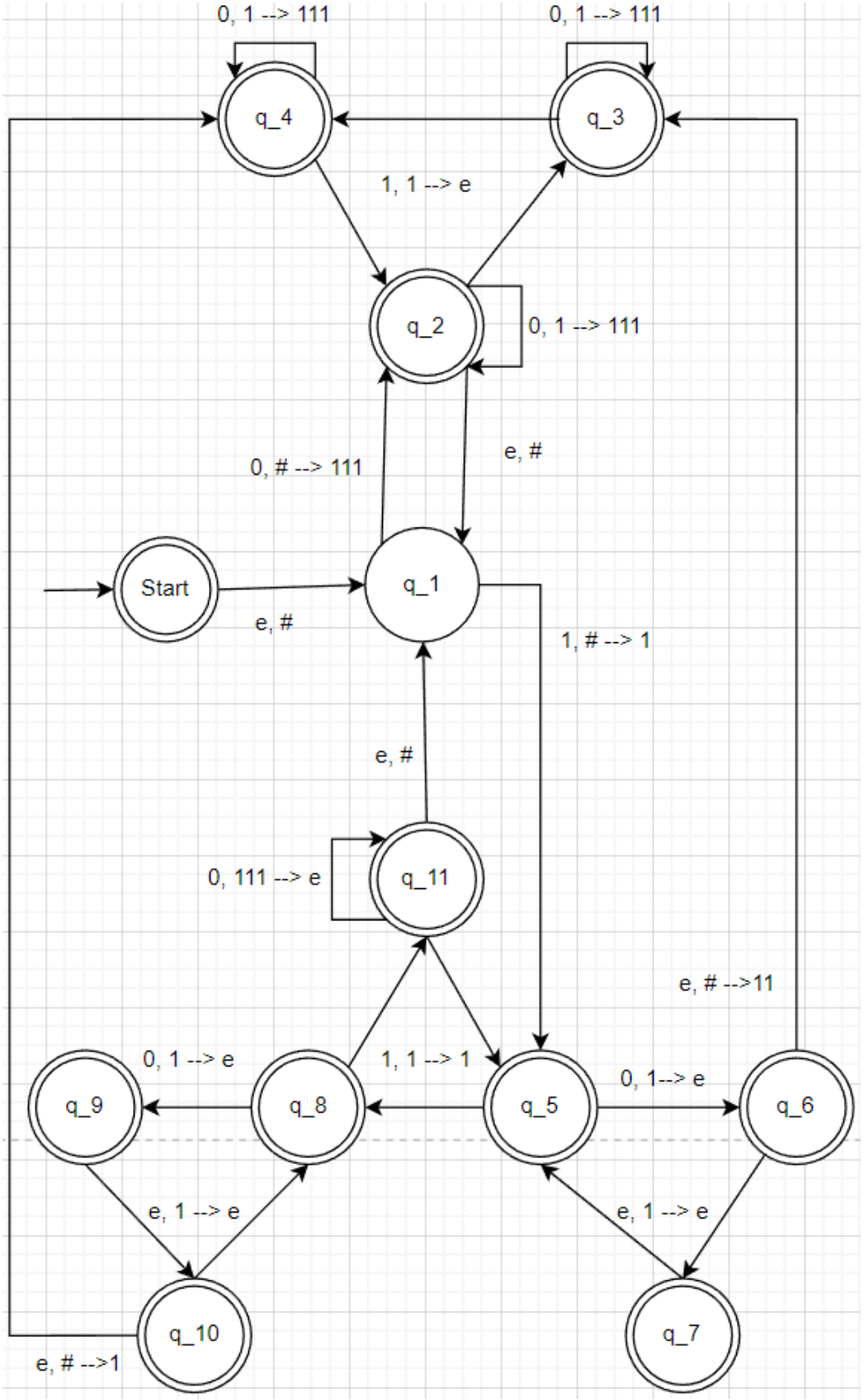

Таблица переходов.

	0			1		e	
	1	#	111	1	#	1	#
Start						q_1, #	
q_1		q_2, 111			q_5, 1		
q_2	q_2, 111			q_3, e			q_1
q_3	q_3, 111			q_4, e			
q_4	q_4, 111			q_2, e			
q_5	q_6, e			q_8, 1			
q_6						q_7, e	q_3, 11
q_7						q_5, e	
q_8	q_9, e			q_11, 1			
q_9						q_10, e	
q_10						q_8, e	q_4, 1
q_11			q_11, e	q_5, 1			q_1

Код программы.

Файл Program.cs

```
using System;

namespace AlgorithmComplexityTheory
{
    internal class Program
    {
        static void Main()
        {
            Console.Write("Введите строку:");
            char[] Alphabet = { '0', '1' };
            //FirstLaboratoryWork firstLaboratoryWork = new FirstLaboratoryWork(Alphabet,
Console.ReadLine());
            //while (Cont()) { firstLaboratoryWork.Start(Console.ReadLine()); }
            SecondLaboratoryWork secondLaboratoryWork = new SecondLaboratoryWork(Alphabet,
Console.ReadLine());
            Console.ReadKey();
        }
        static bool Cont()
        {
            Console.WriteLine("Продолжить? Да или Y");
            string cont = Console.ReadLine();
            if (cont == "Да" || cont == "Y") return true;
```

```
            else return false;
        }
    }
}
```

Файл LaboratoryWorks.cs

```
using System;

namespace AlgorithmComplexityTheory
{
    internal abstract class LaboratoryWorks
    {
        private protected string _w = null;
        private protected char[] Alphabet = null;
        public LaboratoryWorks(char[] alphabet, string w)
        {
            Alphabet = alphabet;
            _w = w;
        }
        private protected bool CheckingAlphabet(string w, bool result = false)// string replace char[]
        {
            for (int i = 0; i < w.Length; i++)
            {
                result = false;
                for (int j = 0; j < Alphabet.Length; j++)
                {
                    if (w[i] == Alphabet[j])
                        result = true;
                }
                if (!result)
                {
                    Console.WriteLine($"В строке обнаружен символ не принадлежащий
алфавиту\n{w[i]}");
                    Console.ReadKey();
                    return result;
                }
            }
            return result;
        }
        public abstract void Start(string w);
        //private protected virtual void FiniteAutomaton() { }
    }
}
```

Файл SecondLaboratoryWork.cs

```
using System;
using System.Collections.Generic;
using System.IO;

namespace AlgorithmComplexityTheory
{
```

```
internal sealed class SecondLaboratoryWork : LaboratoryWorks
{
    //private List<Active> actives = new();
    private Active action = null;
    private sealed class Active
    {
        internal State state = State.Start;
        internal Stack<char> stack = new();
        internal string history = $"State: {State.Start}, Stack: #; ";
        internal bool active = true;
        public Active(State state)
        {
            this.state = state;
            stack.Push('#');
        }
        public Active(State state, string parentHistory)
        {
            this.state = state;
            history = parentHistory;
            stack.Push('#');
        }
    }
    private enum State
    {
        Start,
        q1,
        q2,
        q3,
        q4,
        q5,
        q6,
        q7,
        q8,
        q9,
        q10,
        q11,
    }
    public SecondLaboratoryWork(char[] alphabet, string w) : base(alphabet, w) { Start(w); }

    public sealed override void Start(string w)
    {
        bool result = false;
        action = new Active(State.Start);
        if (w != string.Empty)
        {
            bool check = CheckingAlphabet(w);
            if (!check)
                return;
            else
                _w = w;
            //actives.Add(new Active(State.Start));
            Check(w);
```

```
            foreach (char symbol in _w)
            {
                //for (int i = actives.Count - 1; i >= 0; i--)
                //{
                //    if (actives[i].active)
                //    {
                //        FiniteAutomaton(actives[i], symbol);
                //        Write(actives[i], symbol);
                //    }
                //}
                FiniteAutomaton(action, symbol);
                Write(action, symbol);
            }
        }
        using (StreamWriter writer = new("result.txt", false))
        {
            //foreach (Active action in actives)
            //{
            //    writer.WriteLine(action.history);
            //    if (action.state != State.Exit && action.state != State.q1)
            //        result = true;
            //}
            writer.WriteLine(action.history);
            if (action.state != State.q1)
                result = true;
            writer.WriteLine(result);
        }
    }
    private void FiniteAutomaton(Active active, char symbol)
    {
        //actives.Add(new Active(State.q5, active.history));
        //actives.Last().stack.Push('1');
        bool read = false;
        switch (active.state)
        {
            case State.Start:
                {
                    if (read)
                        break;
                    if (active.stack.Peek() == '#')
                    {
                        active.state = State.q1;
                        goto case State.q1;
                    }
                    else
                        goto default;
                }
            case State.q1:
                {
                    if (read)
                        break;
                    read = true;
```

```
        if (symbol == '0' && active.stack.Peek() == '#')
        {
            active.stack.Push('1');
            active.stack.Push('1');
            active.stack.Push('1');
            active.state = State.q2;
            goto case State.q2;
        }
        else if(symbol == '1' && active.stack.Peek() == '#')
        {
            active.stack.Push('1');
            active.state = State.q5;
            goto case State.q5;
        }
        else
            goto default;
    }
case State.q2:
    {
        if (active.stack.Peek() == '#')
        {
            active.state = State.q1;
            goto case State.q1;
        }
        if (read)
            break;
        read = true;
        if (symbol == '0' && active.stack.Peek() == '1')
        {
            active.stack.Push('1');
            active.stack.Push('1');
            active.stack.Push('1');
            goto case State.q2;
        }
        else if (symbol == '1' && active.stack.Peek() == '1')
        {
            active.stack.Pop();
            active.state = State.q3;
            goto case State.q3;
        }
        else
            goto default;
    }
case State.q3:
    {
        if (read)
            break;
        read = true;
        if (symbol == '0' && active.stack.Peek() == '1')
        {
            active.stack.Push('1');
            active.stack.Push('1');
```

```
            active.stack.Push('1');
            goto case State.q3;
        }
        else if (symbol == '1' && active.stack.Peek() == '1')
        {
            active.stack.Pop();
            active.state = State.q4;
            goto case State.q4;
        }
        else
            goto default;
    }
case State.q4:
    {
        if (read)
            break;
        read = true;
        if (symbol == '1' && active.stack.Peek() == '1')
        {
            active.stack.Pop();
            active.state = State.q2;
            goto case State.q2;
        }
        else if (symbol == '0' && active.stack.Peek() == '1')
        {
            active.stack.Push('1');
            active.stack.Push('1');
            active.stack.Push('1');
            goto case State.q4;
        }
        else
            goto default;
    }
case State.q5:
    {
        if (read)
            break;
        read = true;
        if (symbol == '0' && active.stack.Peek() == '1')
        {
            active.stack.Pop();
            active.state = State.q6;
            goto case State.q6;
        }
        else if (symbol == '1' && active.stack.Peek() == '1')
        {
            active.stack.Push('1');
            active.state = State.q8;
            goto case State.q8;
        }
        else
            goto default;
```

```
    }
case State.q6:
    {
        if (active.stack.Peek() == '#')
        {
            active.stack.Push('1');
            active.stack.Push('1');
            active.state = State.q3;
            goto case State.q3;
        }
        else if (active.stack.Peek() == '1')
        {
            active.stack.Pop();
            active.state = State.q7;
            goto case State.q7;
        }
        else
            goto default;
    }
case State.q7:
    {
        if (active.stack.Peek() == '1')
        {
            active.stack.Pop();
            active.state = State.q5;
            goto case State.q5;
        }
        else
            goto default;
    }
case State.q8:
    {
        if (read)
            break;
        read = true;
        if (symbol == '0' && active.stack.Peek() == '1')
        {
            active.stack.Pop();
            active.state = State.q9;
            goto case State.q9;
        }
        else if (symbol == '1' && active.stack.Peek() == '1')
        {
            active.stack.Push('1');
            active.state = State.q11;
            goto case State.q11;
        }
        else
            goto default;
    }
case State.q9:
    {
```

```
        if (active.stack.Peek() == '1')
        {
            active.stack.Pop();
            active.state = State.q10;
            goto case State.q10;
        }
        else
            goto default;
    }
case State.q10:
    {
        if (active.stack.Peek() == '#')
        {
            active.stack.Push('1');
            active.state = State.q4;
            goto case State.q4;
        }
        else if (active.stack.Peek() == '1')
        {
            active.stack.Pop();
            active.state = State.q8;
            goto case State.q8;
        }
        else
            goto default;
    }
case State.q11:
    {
        if (active.stack.Peek() == '#')
        {
            active.state = State.q1;
            goto case State.q1;
        }
        if (read)
            break;
        read = true;
        if (symbol == '1' && active.stack.Peek() == '1')
        {
            active.stack.Push('1');
            active.state = State.q5;
            goto case State.q5;
        }
        else if (symbol == '0' && active.stack.Peek() == '1')
        {
            active.stack.Pop();
            active.stack.Pop();
            active.stack.Pop();
            goto case State.q11;
        }
        else
            goto default;
    }
```

```
            default:
                {
                    Console.WriteLine("Конец");
                    //Console.WriteLine($"{active.state}, строка: {actives.IndexOf(active)}");
                    Console.WriteLine($"{active.state}");
                    Console.ReadKey();
                    return;
                }
        }
    }
    private void Write(Active active, char symbol) => active.history += $"Input: {symbol}, State:
{active.state}, Stack: {string.Join(",", active.stack.ToArray())}; ";
    private void Check(string w)
    {
        int[] count = new int[Alphabet.Length];
        for (int i = 0; i < w.Length; i++)
            for (int j = 0; j < Alphabet.Length; j++)
                if (w[i] == Alphabet[j])
                    count[j]++;
        Console.WriteLine($"Количество {Alphabet[0]} = {count[0]}, количество {Alphabet[1]} =
{count[1]}.\n{count[1] / count[0]} => Подходит? {count[0] * 3 != count[1]}");
    }
}
}
```