МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

кафедра «Вычислительная техника».

**Лабораторная работа №1**

по дисциплине: «Автоматизация проектирования технических средств»

Выполнил:

студент 4 курса, гр. ИВТВМбд-41

Захарычев Никита Алексеевич.

Проверил:

кандидат технических наук,

доцент кафедры ВТ

Войт Николай Николаевич.

г. Ульяновск, 2018

**Задание**

Реализовать программу, производящую анализ алгоритма выполнения заданной операции по логической схеме алгоритма (ЛСА).

Входные данные: ЛСА

Выходные данные: результат проверки ЛСА на корректность.

**Теоретические сведения**

Логическая схема алгоритма (ЛСА) является одним из основных способов задания закона функционирования автоматов. Большим плюсом логической схемы алгоритма является компактность и возможность легкого преобразования в другую из форм задания автоматов.

Логические схемы алгоритмов путем равносильных преобразований позволяют осуществлять минимизацию микропрограммы и в сочетании с матричными схемами алгоритмов дают возможность формализовать и упростить этот процесс.

Запись ЛСА должна удовлетворять следующим условиям:

* строка начинается оператором A0, обозначающим вход в ЛСА, и заканчивается конечным оператором Aк;
* перед оператором A0 и после оператора Aк стрелок не должно быть;
* после каждого логического условия обязательно стоит верхняя\uparrow стрелка;
* для каждой верхней \uparrowстрелки с номером i должна быть только одна нижняя \downarrowстрелка с тем же номером;
* для каждой нижней \downarrowстрелки с номером i должна быть хотя бы одна верхняя \uparrowстрелка с тем же номером.

Иногда верхние и нижние стрелки заменяют на правые и левые полускобки.

ЛСА определяет порядок выполнения операторов в зависимости от значения входящих в нее логических условий.

Пример строки ЛСА:

A0 A1 *P*1 ↑1*P*2 ↑2*P*3 ↑1 A4 ω ↑3 ↓1 A2 ω ↑3↓2 A3 ↓3 AK

Работа алгоритма начинается с того, что выполняется первый оператор, стоящий слева. После того, как некоторый член ЛСА выполнится, определятся, какой член ЛСА должен выполняться за ним. Если это был оператор, то следом за ним должен выполняться тот член ЛСА, который стоит непосредственно справа от него. Если последний выполняющий член ЛСА был логическим условием, то возможны два случая. Если проверявшееся условие не выполнено, то должен выполняться член ЛСА, находящийся справа. Если оно выполнено, должен выполняться тот член ЛСА, к которому ведет стрелка, начинающейся после данного условия. Работа алгоритма заканчивается либо тогда, когда последний из выполняющихся операторов содержит указанные о прекращении работы алгоритма, либо тогда, когда на некотором этапе не оказывается такого члена ЛСА, который должен бы выполниться. Таким образом, распределение значений логических условий в ЛСА определяет порядок выполнения операторов, входящих в эту ЛСА.

**Реализация**

Считываем входную строку (ЛСА) с консоли:

Console.WriteLine("Введите входную строку (ЛСА):");

input = Console.ReadLine();

После необходимо «отделить» переменные, что относится к X, Y и стрелки переходов (в данном случаем будем использовать полу скобки). Для это будем использовать следующую функцию:

public static void findXY()

Также необходимо получить позицию каждого элемента в строке. Для этого используем функцию public static int getNumber.

Для хранения состояния и условия перехода в него используется класс StateLSA.

Для поиска стрелок(полу скобок) используется функция: findRight.

Функция public static void addCond\_if(List<StateLSA> list, string cond\_if) используется для инициализации условии переходов в состояние.

Внутреннее представление ЛСА было реализовано в виде списка смежности.

public static List<List<StateLSA>> graph = new List<List<StateLSA>>();

В этом списке для каждой вершины определен список состояний, в которые можно попасть из нее и условия перехода.

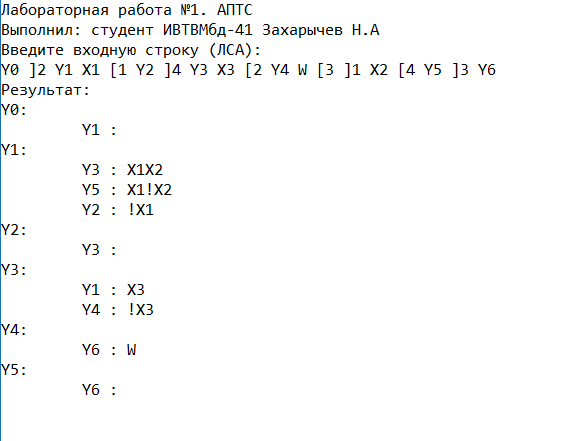
Разбор строки ЛСА выполняется рекурсивно с помощью функции public static void getLSA() которая возвращает все доступные из заданной позиции состояния.

**Краткое описание алгоритма разбора ЛСА**

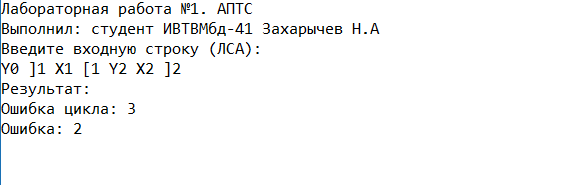
Начиная с указанной позиции в строке ЛСА запускается цикл по всем следующим символам строки. Среди входной строки ищется состояние (Y), для него создается экземпляр класса StateLSA, который определяет состояние и условие перехода. При встрече условия перехода (X), выполняется поиск стрелок (полускобок), имеющих одинаковые индексы (номера). В функцию getState передается позиция вершины, для которой мы хотим найти все доступные из нее другие вершины, функция запускается рекурсивно с позициями в строке, после верхней стрелки – если условие ложное и после нижней стрелки – если условие истинно. При встрече символа “W” выполняется переход только к нижней стрелке. Также реализован поиск ошибок с непосредственным выводом в месте ошибки.

**Тестирование**

1. **Входные данные**



1. **Входные данные с ошибкой**



**Вывод:** В результате выполнения данной лабораторной работы были получены навыки в разборе логической схемы алгоритмов(ЛСА) и построении матрицы переходов на ее основе, а также написания программ на языке C#.

**Исходный код**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace apts\_lab1

{

// класс для реализации списка смежностей, хранит состоянии и условие перехода

class StateLSA

{

public string state;

public string cond\_if;

}

class Program

{

// Входная строка (ЛСА)

public static string input;

// Список смежности

public static List<List<StateLSA>> graph = new List<List<StateLSA>>();

// Счетчик цикла

public static int cycle = 0;

// Функция добавления условия перехода

public static void addCond\_if(List<StateLSA> list, string cond\_if)

{

foreach (StateLSA item in list)

item.cond\_if = cond\_if + item.cond\_if;

}

// Функция поиска правой полускобки

public static int findRight(int number)

{

int index = input.IndexOf("]" + number);

if (index < 0)

{

Console.WriteLine("Ошибка: невозможно найти ] " + number);

throw new Exception();

}

return index;

}

// Поиск макс. состояния из Y

public static int maxState()

{

int maxItem = 0;

for (int i = 0; i < input.Length; i++)

{

if (input[i] == 'Y')

{

i++;

int num = getNumber(ref i);

if (num > maxItem)

maxItem = num;

}

}

return maxItem;

}

// Функция получения номеров в строке

public static int getNumber(ref int index)

{

int number = 0;

try

{

int space\_ind = input.IndexOf(' ', index);

if (space\_ind < 0)

space\_ind = input.Length;

string sub\_str = input.Substring(index, space\_ind - index);

number = int.Parse(sub\_str);

index = space\_ind + 1;

}

catch (Exception)

{

Console.WriteLine("Ошибка: " + index);

throw new Exception();

}

return number;

}

// Функция, реализующая получения состояния

public static List<StateLSA> getState(int index)

{

if (cycle > 100)

{

Console.WriteLine("Ошибка цикла: " + index);

throw new Exception();

}

int x\_number = 0;

string cond\_if = "";

cycle++;

// Создаем список всех состояний

List<StateLSA> states = new List<StateLSA>();

while (index < input.Length)

{

// Если встречааем Y то создаем его состояние и получаем его номер

if (input[index] == 'Y')

{

index++;

int state\_number = getNumber(ref index);

states.Add(new StateLSA { state = state\_number.ToString(), cond\_if = "" });

cycle--;

return states;

}

// Если встречааем Х

else if (input[index] == 'X')

{

// Получаем его индекс и номер в строке

index++;

x\_number = getNumber(ref index);

cond\_if = "X" + x\_number;

if (input[index] == '[') // Стрелка вверх

{

// если условие перехода выполняется то (true)

index++;

int arrow\_number = getNumber(ref index);

int new\_ind = findRight(arrow\_number);

List<StateLSA> true\_states = getState(new\_ind);

addCond\_if(true\_states, cond\_if);

states.AddRange(true\_states);

// если условие перехода не выполняется (false)

List<StateLSA> false\_states = getState(index);

addCond\_if(false\_states, "!" + cond\_if);

states.AddRange(false\_states);

cycle--;

// Возвращаем состояние

return states;

}

}

// Если встречаем W

else if (input[index] == 'W')

{

index += 2;

cond\_if = "W";

// Ищем левые полускобки

if (input[index] == '[')

{

index++;

int arrow\_number = getNumber(ref index);

int new\_ind = findRight(arrow\_number);

List<StateLSA> true\_states = getState(new\_ind);

addCond\_if(true\_states, cond\_if);

states.AddRange(true\_states);

cycle--;

return states;

}

}

// Если пробел, то ничего не делаем и просто увел. индекс

else if (input[index] == ' ')

{

index++;

}

// Если ], то ищем индекс ]

else if (input[index] == ']')

{

index = input.IndexOf(' ', index);

}

else

{

// Вывод если ошибки

Console.WriteLine("Ошибка: " + index);

throw new Exception("Ошибка: " + index);

}

}

cycle--;

return null;

}

// Функция реализующая итоговый вывод всех состояний и переходов

public static void printGraph()

{

for (int i = 0; i < graph.Count - 1; i++)

{

Console.WriteLine("Y{0}: ", i);

for (int j = 0; j < graph[i].Count; j++)

{

Console.WriteLine("\t Y{0} : {1}", graph[i][j].state, graph[i][j].cond\_if);

}

}

}

// Функция для разбития ХY во входной строке

public static void findXY()

{

List<int> initX = new List<int>();

List<int> initY = new List<int>();

for (int i = 0; i < input.Length; i++)

{

// Если встречаем У то получаем номер, иначе выдаем ошибку

if (input[i] == 'Y')

{

i++;

int y\_num = getNumber(ref i);

if (initY.Contains(y\_num))

{

Console.WriteLine("Ошибка: " + i);

throw new Exception();

}

initY.Add(y\_num);

i--;

}

// Если встречаем Х то получаем номер, иначе выдаем ошибку

else if (input[i] == 'X')

{

i++;

int x\_num = getNumber(ref i);

if (initX.Contains(x\_num))

{

Console.WriteLine("Ошибка: " + i);

throw new Exception();

}

initX.Add(x\_num);

i--;

}

}

return;

}

// Функция получения самого ЛСА

public static void getLSA()

{

int i = 0;

try

{

// Инициализация XY

findXY();

// Получаем все состояния

int max\_state = maxState();

// Создаем список смежности

graph = new List<List<StateLSA>>(new List<StateLSA>[max\_state + 1]);

for (i = 0; i < input.Length; i++)

{

if (input[i] == 'Y')

{

i++;

int state\_number = getNumber(ref i);

i--;

var state = getState(i);

graph[state\_number] = state;

}

}

printGraph();

}

catch (Exception e)

{

Console.WriteLine("Ошибка: " + i);

}

}

static void Main(string[] args)

{

Console.WriteLine("Лабораторная работа №1. АПТС");

Console.WriteLine("Выполнил: студент ИВТВМбд-41 Захарычев Н.А");

Console.WriteLine("Введите входную строку (ЛСА):");

input = Console.ReadLine();

Console.WriteLine("Результат:");

getLSA();

Console.ReadKey();

}

}

}