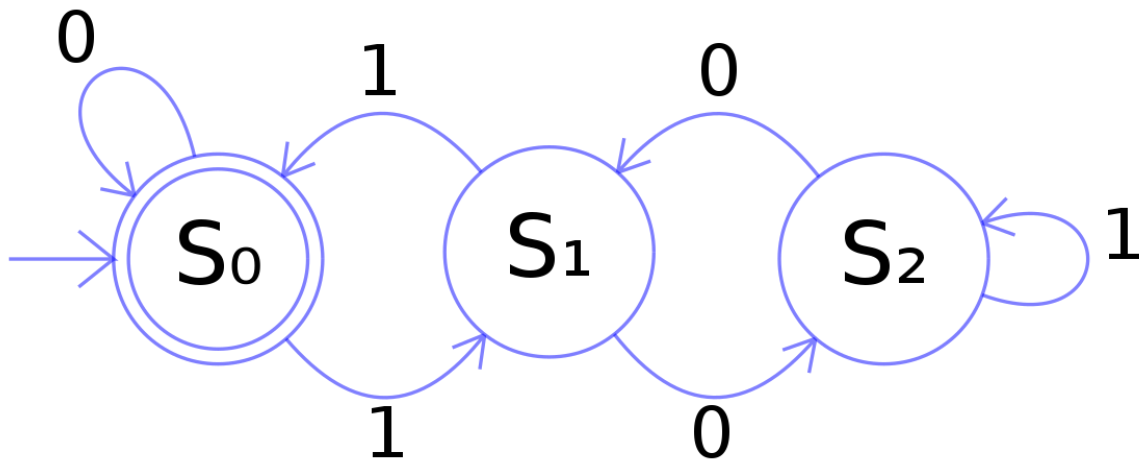


1/30/2020



UNIWA

ΘΕΩΡΙΑ ΥΠΟΛΟΓΙΣΜΟΥ – ΕΡΓΑΣΙΑ ΕΞΑΜΗΝΟΥ

Ασημακόπουλος Χαράλαμπος | cs141098

Email: cs141098@uniwa.gr | xarhsasi@gmail.com

GitHub: <https://github.com/HarrysAsi/DFA-UNIWA>

Πίνακας περιεχομένων

Περιγραφή εργασίας	2
Σενάριο εκτέλεσης	3
Περιγραφή κώδικα	6
Κλάση DFA.....	7
Κλάση DFANode.....	9
Κλάση DFATable.....	10
Κλάση DFATableRow.....	11
Κλάση CLIColor.....	12
Κλάση DFAFileReader	13
Κλάση DFAExcel (Προαιρετικό).....	15
Κλάση DFAGraph (Προαιρετικό).....	16
Κλάση Runnable.....	17
Βασικό main.py	19
Παράδειγμα	19
Οδηγίες εγκατάστασης	22

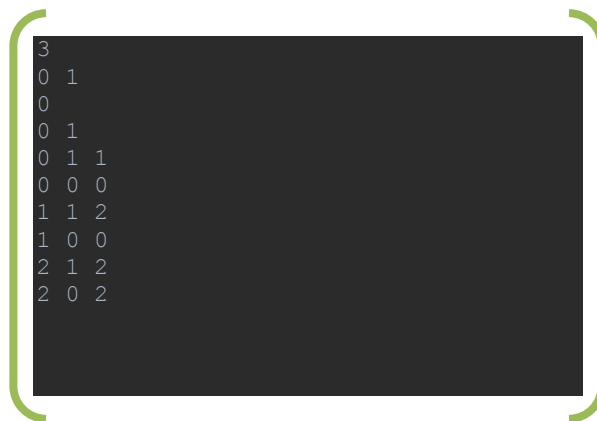
Περιγραφή εργασίας

Η εργασία υλοποιήθηκε σε γλώσσα προγραμματισμού Python έκδοσης 3.7. Δεν υπάρχει γραφικό περιβάλλον και ο χρήστης έχει τη δυνατότητα να τη χειριστεί μέσω τερματικού στο οποίο έχει δωθεί έμφαση έτσι ώστε να προσφέρει όσο το δυνατόν πιο εύκολη χρήση της εφαρμογής.

Το πρόγραμμα που υλοποιήθηκε έχει ως είσοδο ένα αρχείο το οποίο περιγράφει εξ ολοκλήρου το ντετερμινιστικό πεπερασμένο αυτόματο.

Σημειώνεται ότι το πρόγραμμα προσαρμόζεται σε οποιαδήποτε περιγραφή δωθεί στο αρχείο αυτό.

Η μορφή του αρχείου έχει προκαθοριστεί να είναι όπως βλέπουμε στο παρακάτω παράδειγμα:



```
3
0 1
0
0 1
0 1 1
0 0 0
1 1 2
1 0 0
2 1 2
2 0 2
```

όπου οι γραμμές περιγράφουν:

- Το πλήθος των καταστάσεων του αυτόματου
- Τα σύμβολα από τα οποία αποτελείται
- Την αρχική κατάσταση
- Τις τελικές καταστάσεις

Και την περιγραφή των καταστάσεων όπου:

- (0 1 1) → Αν ο κόμβος 0 δεχθεί στην είσοδο του 1, τότε μεταβαίνει στον κόμβο 1
- (0 0 0) → Αν ο κόμβος 0 δεχθεί στην είσοδο του 0, τότε παραμένει στον κόμβο 0
- (1 1 2) → Αν ο κόμβος 1 δεχθεί στην είσοδο του 1, τότε μεταβαίνει στον κόμβο 2
- (1 0 0) → Αν ο κόμβος 1 δεχθεί στην είσοδο του 0, τότε μεταβαίνει στον κόμβο 0
- (2 1 2) → Αν ο κόμβος 2 δεχθεί στην είσοδο του 1, τότε μεταβαίνει στον κόμβο 2
- (2 0 2) → Αν ο κόμβος 2 δεχθεί στην είσοδο του 0, τότε παραμένει στον κόμβο 2

Εφόσον η μορφή του αρχείου είναι ορθή και δεν υπάρξουν προβλήματα κατά το διάβασμα του, τότε εμφανίζεται το κατάλληλο μήνυμα στον χρήστη όπου τον προτρέπει να δώσει μια συμβολοσειρά η οποία θα ελεγχθεί αν είναι έγκυρη ή όχι από το αυτόματο που περιγράφεται στο αρχείο. Εφόσον ολοκληρωθεί η εισαγωγή της συμβολοσειράς από τον χρήστη και πατήσει “enter” τότε το πρόγραμμα ελέγχει αν η συμβολοσειρά είναι αποδεκτή από το ντετερμινιστικό αυτόματο και ενημερώνει τον χρήστη.

Παράλληλα, εφόσον γίνει ο έλεγχος της συμβολοσειράς το πρόγραμμα «σχεδιάζει» το αυτόματο σε ένα αρχείο τύπου PDF και εξάγει και τον πίνακα μεταβάσεων του ντετερμινιστικού αυτόματος που περιγράφηκε.

Σενάριο εκτέλεσης

Ο χρήστης εκτελεί το πρόγραμμα και του εμφανίζονται οι σχετικές πληροφορίες:

```
*****
HELLO DEAR USER, WELCOME TO DFA PYTHON CLI PROGRAM
*****
Author: Charalampos Asimakopoulos
AM: 141098
Email: csl41098@uniwa.gr - xarhsasi@gmail.com
Github: https://github.com/HarrysAsi/DFA-UNIWA.git
Please enter a phrase: 
```

Όπου περιγράφεται ο ιδιοκτήτης του προγράμματος και κάποιες πληροφορίες σχετικά με αυτόν. Έπειρα προτρέπει τον χρήστη να εισάγει μία λέξη έτσι ώστε να ελεγχθεί από το πρόγραμμα αν είναι αποδεκτή από το αυτόματο ή όχι.

```
*****
HELLO DEAR USER, WELCOME TO DFA PYTHON CLI PROGRAM
*****
Author: Charalampos Asimakopoulos
AM: 141098
Email: csl41098@uniwa.gr - xarhsasi@gmail.com
Github: https://github.com/HarrysAsi/DFA-UNIWA.git
Please enter a phrase: 1010
```

Έστω ότι ο χρήστης εισάγει τη λέξη που επιθυμεί να ελέγξει τότε το πρόγραμμα εμφανίζει τις πληροφορίες του αυτόματου και επίσης αν η συμβολοσειρά είναι αποδεκτή ή όχι.

```
*****
HELLO DEAR USER, WELCOME TO DFA PYTHON CLI PROGRAM
*****
Author: Charalampos Asimakopoulos
AM: 141098
Email: csl41098@uniwa.gr - xarhsasi@gmail.com
Github: https://github.com/HarrysAsi/DFA-UNIWA.git
Please enter a phrase: 1010
You entered: 1010
*****
Total States: 3
Symbols: 0 1
First state: 0
Final states: 0 1

*****
Given word: ['1', '0', '1', '0']
Accepted word, good job!
Press Enter to Exit...|
```

Σε περίπτωση που η συμβολοσειρά δεν είναι αποδεκτή τότε εμφανίζεται το κατάλληλο μήνυμα.

```
*****
HELLO DEAR USER, WELCOME TO DFA PYTHON CLI PROGRAM
*****
Author: Charalampos Asimakopoulos
AM: 141098
Email: csl41098@uniwa.gr - xarhsasi@gmail.com
Github: https://github.com/HarrysAsi/DFA-UNIWA.git
Please enter a phrase: 1101
You entered: 1101
*****
Total States: 3
Symbols: 0 1
First state: 0
Final states: 0 1

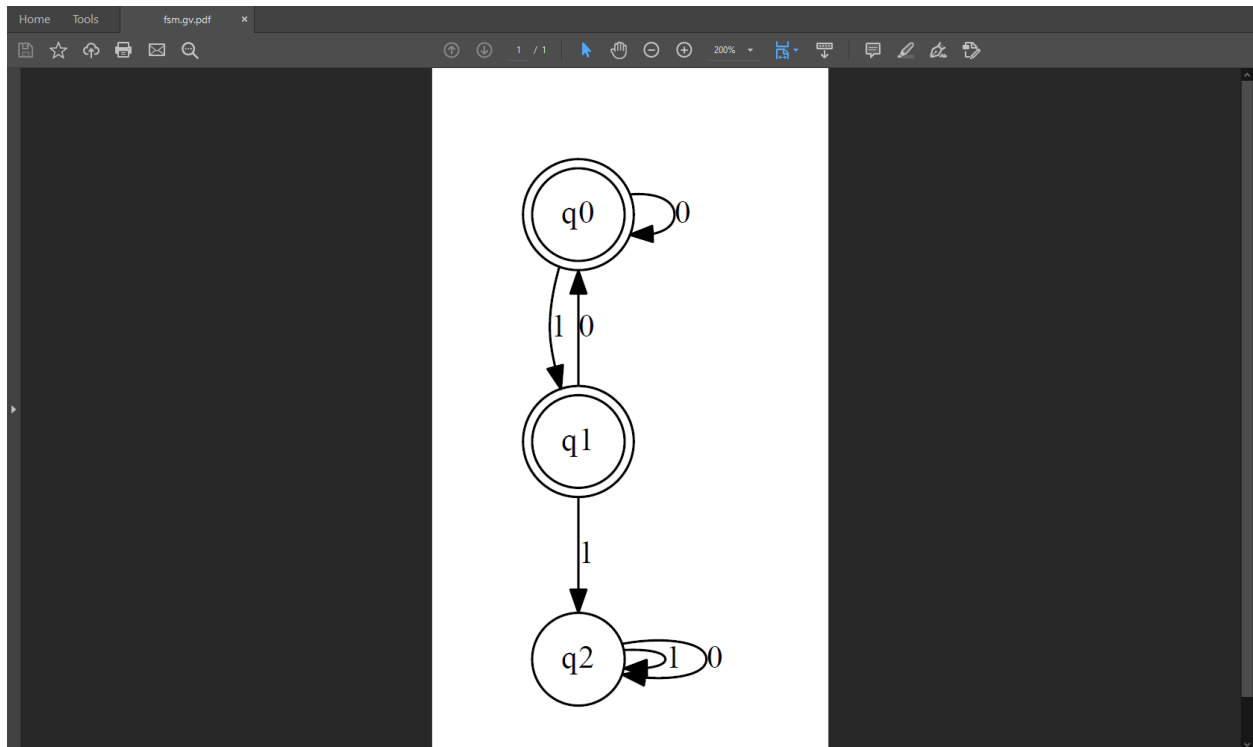
*****
Given word: ['1', '1', '0', '1']
Non accepted word, please try again!
Press Enter to Exit...
```

Το πρόγραμμα εκτός από τη δημιουργία του αυτόματου και τον έλεγχο των συμβολοσειρών:

- Δημιουργεί .pdf αρχείο με το αυτόματο σχεδιασμένο
- Δημιουργεί .xlsx αρχείο με τον πίνακα μετάβασης του αυτομάτου

Οι παραπάνω λειτουργίες υλοποιήθηκαν με σκοπό την καλύτερη κατανόηση των ντετερμινιστικών πεπερασμένων αυτόματων.

Παράδειγμα .pdf αρχείου με το σχεδιασμένο αυτόματο:



Παράδειγμα .xlsx αρχείου με τον πίνακα μετάβασης του αυτόματου:

	A	B	C	D	E	F
1	node	state	next node	is first state	is final state	
2	q0	1	q1	TRUE	TRUE	
3	q0	0	q0	TRUE	TRUE	
4	q1	1	q2	FALSE	TRUE	
5	q1	0	q0	FALSE	TRUE	
6	q2	1	q2	FALSE	FALSE	
7	q2	0	q2	FALSE	FALSE	
8						
9						
10						

Περιγραφή κώδικα

Γλώσσα Προγραμματισμού: Python 3.7

Περιβάλλον Ανάπτυξης: PyCharm

Enviroment: virtualenv

Το πρόγραμμα συνολικά αποτελείται απο 10 αρχεία, εκ των οποίων τα 9 είναι κλάσεις που υλοποιούν τις απαιτήσεις του αυτόματου και ένα αρχείο με όνομα "dfa.txt" στο οποίο περιγράφεται το αυτόματο.

Κλάση DFA

Η κλάση DFA είναι υπεύθυνη για την όλη διαχείριση της λογικής του προγράμματος. Αρχικά αρχικοποιούνται οι μεταβλητές οι οποίες πρόκειται να χρησιμοποιηθούν

Execute_dfa: Η συνάρτηση αυτή, αρχικά δέχεται ως παράμετρο τη λέξη που εισήγαγε ο χρήστης στο πρόγραμμα και στη συνέχεια ελέγχει την ορθότητα της σύμφωνα με το λεξικό που έχει περιγραφεί στο αρχείο. Εφόσον είναι έγκυρη, τότε ελέγχεται αν είναι αποδεκτή από τη περιγραφή τοπυ αυτόματου που δώθηκε μέσα στο αρχείο και εμφανίζει αν είναι αποδεκτή ή όχι.

```
from helpers.CLIColor import CLIColor as color

class DFA:
    """
    The main class which stores and handles the basic logic for the dfa system
    """

    def __init__(self, nodes, table, symbols):
        self.nodes = nodes
        self.table = table
        self.symbols = symbols
        self.current_state = None
        self.final_states = []

    def initialize_state(self):
        """
        initialize the first node and the final nodes
        :return:
        :return:
        """
        for node in self.nodes:
            if node.is_first:
                self.current_state = node
            if node.is_final:
                self.final_states.append(node)

    def execute_dfa(self, word):
        """
        handle the logic for the dfa system
        :param word:
        :return:
        """
        word = self.split_word_to_array(word)
        print(color.UNDERLINE(f"Given word: {word}"))
        if not self.word_validity(word):
            print(color.RED("The given word contains unacceptable symbols for this DFA"))
            return False
        else:
            for char in word:
                node_found = False
                for node in self.nodes:
                    if node is self.current_state and not node_found:
                        node_found = True
                        for w in node.associated_nodes:
                            if str(w.state) == str(char):
                                self.current_state = self.get_node_by_input_state(w.next_state)

            if self.current_state in self.final_states:
                print(color.GREEN("Accepted word, good job!"))
            else:
                print(color.RED("Non accepted word, please try again!"))
            return True

    def word_validity(self, word):
        """
        checks the given word if is contained in the given symbols
        :param word:
        :return: Boolean
        """
        for w in word:
            if w not in self.symbols:
```



```

        return False
    return True

def get_node_by_input_state(self, input_state):
    """
    Returns the node instance based on the given state
    :param input_state:
    :return: DFANode
    """
    for node in self.nodes:
        if str(node.input_state) == str(input_state):
            return node

    return None

def split_word_to_array(self, word):
    """
    split the given word to an array of chars
    :param word:
    :return: Array of chars
    """
    splitted_word = []
    splitted_word[:0] = word
    return splitted_word

def print_nodes(self):
    """
    print all the nodes (debug)
    :return:
    """
    for node in self.nodes:
        print(node)
        for n in node.associated_nodes:
            print(n)

```

Κλάση DFANode

Είναι η κλάση η οποία αναπαριστά κάθε κόμβο του αυτόματου η οποία περιέχει τις καταστάσεις του, τις καταστάσεις που θα επιλεγούν για κάθε πιθανή επόμενη έξοδο καθώς και το αν είναι αρχικός ή τελικός κόμβος.

```
class DFANode:
    """
    DFA Node class which represents each DFA Node
    """

    def __init__(self, input_state, is_first, is_final):
        """
        :param input_state:
        :param is_first:
        :param is_final:
        """
        self.input_state = input_state
        self.is_first = is_first
        self.is_final = is_final
        self.associated_nodes = []

    def add_node(self, node):
        self.associated_nodes.append(node)

    def is_first_or_final_node(self):
        return self.is_final or self.is_final

    def __str__(self):
        return "<DFANode %s %s %s %s>" % (self.input_state, self.is_first, self.is_final,
self.associated_nodes)
```

Κλάση DFATable

Κλάση η οποία αναπαριστά αποκλειστικά τον πίνακα μεταβάσεων του αυτόματου όπως αυτός έχει περιγραφεί από το αρχείο. Ο πίνακας αποτελείται απο γραμμές όπου κάθε γραμμή περιέχει τα χαρακτηριστικά τα οποία περιγράφονται παρακάτω.

```
class DFATable:
    """
    DFA Table class represents the whole table with its rows
    """

    def __init__(self, rows=[]):
        self.rows = rows

    def __del__(self):
        del self.rows
        return self

    def add_row(self, row):
        """
        Adds a row in the table
        :param row:
        :return:
        """
        self.rows.append(row)

    def __str__(self):
        for row in self.rows:
            attrs = vars(row)
            print(' ', '.join("%s: %s" % item for item in attrs.items()))
        return "<DFATable %s>\n" % (self.rows,)
```

Κλάση DFATableRow

Κλάση η οποία αναπαριστά τις γραμμές ενός πίνακα μεταβάσεων οι οποίες προστίθενται στην κλάση DFATable.

```
class DFATableRow:
    """
    DFA Table row class which represents each row of a DFA table
    """

    def __init__(self, current_state, state, next_state, is_first, is_final):
        """
        initialize the RowTable object with its arguments
        :param current_state:
        :param state:
        :param next_state:
        """
        self.current_state = current_state
        self.state = state
        self.next_state = next_state
        self.is_first = is_first
        self.is_final = is_final

    def is_first_or_final(self):
        return self.is_final or self.is_first

    def __str__(self):
        return "<DFATableRow %s %s %s %s %s>" % (
            self.current_state, self.state, self.next_state, self.is_first, self.is_final)
```

Κλάση CLIColor

Κλάση η οποία χειρίζεται τα χρώματα τα οποία εμφανίζονται στο τερματικό για εμφανισιακούς αποκλειστικά λόγους.

```
import os, sys

if sys.platform.lower() == "win32":
    os.system('color')

class CLIColor:
    """
    Available colors for the CLI program used inside print(color.RED(''))
    """
    BLACK = lambda x: '\033[30m' + str(x)
    RED = lambda x: '\033[31m' + str(x)
    GREEN = lambda x: '\033[32m' + str(x)
    YELLOW = lambda x: '\033[33m' + str(x)
    BLUE = lambda x: '\033[34m' + str(x)
    MAGENTA = lambda x: '\033[35m' + str(x)
    CYAN = lambda x: '\033[36m' + str(x)
    WHITE = lambda x: '\033[37m' + str(x)
    UNDERLINE = lambda x: '\033[4m' + str(x)
    RESET = lambda x: '\033[0m' + str(x)
```

Κλάση DFAFileReader

Η κλάση DFAFileReader είναι υπεύθυνη για το διάβασμα του αρχείου και τη εισαγωγή των δεδομένων στις αντίστοιχες κλάσεις για τον χειρισμό και την αναπαράσταση των δεδομένων. Αποτελείται απο 2 βασικές συναρτήσεις οι οποίες είναι:

Symbols_validity: Έλεγχος των χρησιμοποιημένων συμβόλων με αυτών που έχουν περιγραφεί στο αρχείο, σε περίπτωση που έχουν χρησιμοποιηθεί σύμβολα που δεν ανήκουν στο αλφάβητο τότε εμφανίζεται κατάλληλο μήνυμα.

Read_file: Διάβασμα του αρχείου και αποθήκευση κόμβων σε μορφή DFANode(input_state, is_first, is_last, associated_nodes) όπου αναπαριστά έναν κόμβο όπως αυτός έχει περιγραφεί παραπάνω. Αποθήκευση του πίνακα καταστάσεων όπου DFATableRow(input_node, state, next_node, is_first, is_last) είναι οι γραμμές του πίνακα οι οποίες εισάγονται στη κλάση DFATable() όπου είναι ο DFA Πίνακας. Η συνάρτηση αυτή επιστρέφει τους δημιουργημένους κόμβους και τον πίνακα όπου πρόκειται να χρησιμοποιηθούν για τον έλεγχο των λέξεων που θα εισαχθούν απο τον χρήστη.

Symbols_validity: Έλεγχος των συμβόλων, που χρησιμοποιούνται στην περιγραφή του αυτόματου, για το αν είναι αποδεκτά απο το λεξικό το οποίο περιγράφεται στο αρχείο περιγραφής του αυτόματου.

```
from helpers.DFATableRow import DFATableRow
from helpers.DFATable import DFATable
from helpers.DFANode import DFANode
from helpers.CLIColor import CLIColor as color

class FileReader:
    """
    DFA File Reader which handles the object structure based on file
    """

    def __init__(self):
        """
        initialize the file reader variables
        """
        self.total_states = 0
        self.symbols = None
        self.first_state = None
        self.final_states = None

        self.nodes = []
        self.table = DFATable()
        self.used_symbols = []

    def read_file(self, filename):
        """
        Opens and reads the given file path and creates the instances which are described in file
        :param filename:
        :return: [self.nodes, self.table]
        """
        try:
            i = 0
            with open(filename, "r") as f:
                for line in f:
                    if i == 0:
                        self.total_states = line
                    elif i == 1:
                        self.symbols = line
                    elif i == 2:
                        self.first_state = line
                    elif i == 3:
                        self.final_states = line
                    else:
                        row = line.replace("\n", "").split(" ")
                        is_first_state = True if int(self.first_state) == int(row[0]) else False
                        is_final_state = True if row[0] in self.final_states else False
                        if i == 4:
```

```

        node_tmp = DFANode(row[0], is_first_state, is_final_state)
        self.nodes.append(node_tmp)
    else:
        exists = False
        for n in self.nodes:
            if row[0] == n.input_state:
                exists = True
        if not exists:
            node_tmp = DFANode(row[0], is_first_state, is_final_state)
            self.nodes.append(node_tmp)
        self.used_symbols.append(row[1])
        row_tmp = DFATableRow(row[0], row[1], row[2], is_first_state, is_final_state)
        self.table.add_row(row_tmp)
    i = i + 1
    # Add rows in nodes for a better representation and data handling
    for row in self.table.rows:
        for node in self.nodes:
            if row.current_state == node.input_state:
                node.add_node(row)

    self.print_info()
    f.close()
    return self.nodes, self.table, self.symbols
except FileNotFoundError:
    f.close()
    print(color.RED("ERROR: File not found"))

def symbols_validity(self):
    """
    Checks if the dfa symbols described in the file are contained in the dfa dictionary
    which is also described in the file
    :return: Exits if error with the appropriate message, otherwise returns true (is valid)
    """
    for used_symbol in self.used_symbols:
        if used_symbol not in self.symbols:
            print(color.RED("The described symbols are not in the described dictionary, please try
again!"))
            return False
    return True

def print_info(self):
    print(color.YELLOW("*****"))
    print(color.MAGENTA(f"Total States: {self.total_states}"
                        f"Symbols: {self.symbols}"
                        f"First state: {self.first_state}"
                        f"Final states: {self.final_states}"))
    print(color.YELLOW("*****"))

```

Κλάση DFAExcel (Προαιρετικό)

Κλάση η οποία διαχειρίζεται την αποθήκευση του πίνακα μεταβάσεων σε αρχείο Excel για τη καλύτερη κατανόηση της λειτουργίας του αυτόματου.

```
from openpyxl import Workbook

class DFAExcel:
    """
    DFAExcel class which creates an excel file with the dfa table for the automaton
    """

    def __init__(self, filename="dfa_sample.xlsx"):
        """
        Initialize the DFAExcel instance
        :param filename:
        """
        self.wb = Workbook()
        self.ws = None
        self.filename = filename
        self.activate_wb()

    def activate_wb(self):
        """
        Activate the workbook
        :return: void
        """
        self.ws = self.wb.active
        self.initialize_wb()

    def initialize_wb(self):
        """
        Initialize the workbook
        :return: void
        """
        self.ws.title = "DFA Workbook"

    def write_to_excel(self, table, prefix):
        """
        File logic and format
        :param table:
        :param prefix:
        :return:
        """
        self.ws['A1'] = "node"
        self.ws['B1'] = "state"
        self.ws['C1'] = "next node"
        self.ws['D1'] = "is first state"
        self.ws['E1'] = "is final state"

        row_i = 2
        for row in table.rows:
            self.ws.cell(column=1, row=row_i, value=prefix + row.current_state)
            self.ws.cell(column=2, row=row_i, value=row.state)
            self.ws.cell(column=3, row=row_i, value=prefix + row.next_state)
            self.ws.cell(column=4, row=row_i, value=row.is_first)
            self.ws.cell(column=5, row=row_i, value=row.is_final)
            row_i = row_i + 1
        self.save_wb()

    def save_wb(self):
        """
        Save the workbook
        :return:
        """
        self.wb.save(self.filename)
```


Κλάση DFAGraph (Προαιρετικό)

Κλάση η οποία χρησιμοποιεί τη βιβλιοθήκη **graphviz** για τη δημιουργία αρχείου PDF στο οποίο σχεδιάζεται η μορφή του αυτομάτου που έχει περιγραφεί, σε σχηματική μορφή.

```
from graphviz import Digraph
from helpers.CLIColor import CLIColor as color

class DFAGraph:
    """
    class which initializes and creates a pdf file with the dfa representation
    """

    def __init__(self, nodes=None, prefix="q"):
        """
        Initializes the variables
        :param: table
        """
        # self.table = table
        self.nodes = nodes
        self.prefix = prefix
        self.f = None

    def initialize_graph(self):
        """
        Function which initializes and calculates all the nodes based on the created table
        :return: Shows the pdf with the dfa
        """
        try:
            self.f = Digraph('finite_state machine', filename='fsm.gv')
            self.f.attr(rankdir='q', size='8,5')

            for n in self.nodes:
                self.f.attr('node', shape='doublecircle')
                if n.is_first_or_final_node():
                    self.f.node(self.prefix + n.input_state)

            for n in self.nodes:
                self.f.attr('node', shape='circle')
                if not n.is_first_or_final_node():
                    self.f.node(self.prefix + n.input_state)
                for r in n.associated_nodes:
                    self.f.edge(self.prefix + r.current_state, self.prefix + r.next_state, label=r.state)
        except:
            print(color.RED("Please close the pdf file and execute the program!"))

    def show_graph(self):
        """
        Shows the initialized graph
        :return: void
        """
        self.f.view()
```

Κλάση Runnable

Κλάση η οποία διαχειρίζεται την εκτέλεση και τη αρχικοποίηση των παραπάνω κλάσεων με δομημένη μορφή και με τη σειρά που επιβάλλεται.

```
import os
from helpers.FileReader import FileReader
from helpers.DFA import DFA
from helpers.DFAGraph import DFAGraph
from helpers.CLIColor import CLIColor as color
from helpers.DFAExcel import DFAExcel
from helpers.DFATable import DFATable

class Runnable:
    """
    DFA Runnable class which handles the whole program execution
    """

    def __init__(self, prefix="q", input_word="010101010", rel_path="dfa.txt"):
        """
        Initialize the runnable class with its variables
        """
        self.prefix = prefix
        self.input_word = input_word
        self.rel_path = rel_path

        self.print_welcome()
        self.user_input()

    def user_input(self):
        """
        Function which request user to insert a phrase from the keyboard
        """

        input_word = input("Please enter a phrase: ")
        print("You entered: " + input_word)
        self.input_word = input_word

    def calc_path_file(self):
        """
        calculate the relative path of the given file
        :return: File's path
        """
        script_dir = os.path.dirname(__file__)
        abs_file_path = os.path.join(script_dir, self.rel_path)
        return abs_file_path

    def run(self):
        """
        Execute the program logic
        :return: [nodes, table, symbols]
        """
        nodes, table, symbols = FileReader().read_file(self.calc_path_file())
        initialized = self.dfa_init(nodes, table, symbols)
        if not initialized:
            return False
        else:
            self.graph_init(nodes)
            self.to_excel(table)
            return True

    def to_excel(self, table):
        """
        Creates the excel file based on the table
        :param table:
        :return: void
        """
        excel_file = DFAExcel()
        excel_file.write_to_excel(table, self.prefix)

    def dfa_init(self, nodes, table, symbols):
        """
        initialize the DFA instance with the needed parameters
        :param nodes:
        :param table:
        :param symbols:
        :return: void
        """
```

```

        dfa = DFA(nodes, table, symbols)
        dfa.initialize_state()
        executed = dfa.execute_dfa(word=self.input_word)
        if not executed:
            return False
        return True

def graph_init(self, nodes):
    """
    Initialize the graph logic
    :param nodes:
    :return: void
    """
    graph = DFAGraph(nodes=nodes, prefix=self.prefix)
    graph.initialize_graph()
    graph.show_graph()

def print_welcome(self):
    """
    Simple welcome message for the representation
    :return: void
    """
    print(color.YELLOW("*****"))
    print(color.MAGENTA("HELLO DEAR USER, WELCOME TO DFA PYTHON CLI PROGRAM"))
    print(color.YELLOW("*****"))
    print(color.UNDERLINE("Author: Charalampos Asimakopoulos"))
    print(color.UNDERLINE("AM: 141098"))
    print(color.UNDERLINE("Email: cs141098@uniwa.gr - xarhsasi@gmail.com"))
    print(color.UNDERLINE("Github: https://github.com/HarrysAsi/DFA-UNIWA.git"))

```

Βασικό main.py

Το βασικό αρχείο εκτέλεσης το οποίο αρχικοποιεί τη κλάση runnable με prefix="q" για τη καλύτερη αναπαράσταση των κόμβων μέσα στο αρχείο και στο γράφημα.

Έπειτα εκτελεί τη συνάρτηση run() στην οποία εκτελείται ολόκληρο το πρόγραμμα.

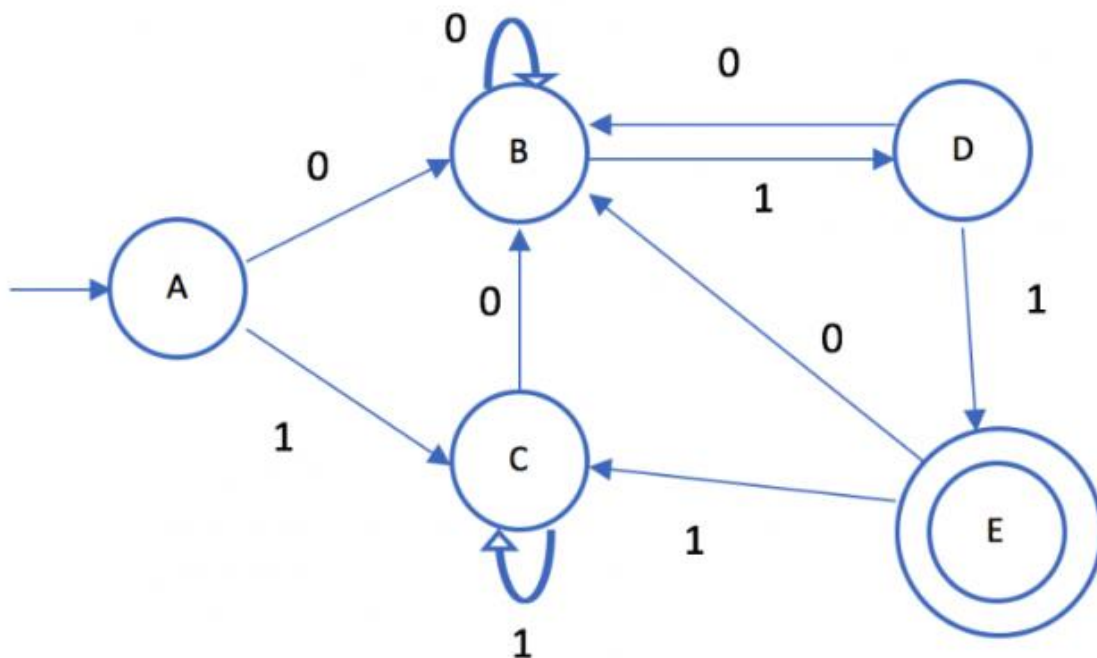
```
from Runnable import Runnable

def main():
    program = Runnable(prefix="q")
    executed = program.run()
    input('Press Enter to Exit...')

if __name__ == '__main__':
    main()
```

Παράδειγμα

Έστω ότι θέλουμε να υλοποιήσουμε το παρακάτω ντετερμινιστικό πεπερασμένο αυτόματο το οποίο περιγράφεται από την εικόνα:



Μερικές απο τις αποδεκτές φράσεις του παραπάνω αυτόματου είναι:

[0 1 1] , [1 1 0 1 1], [1 0 1 1]

Και μερικές απο τις μή αποδεκτές φράσεις είναι:

[0 0 0], [0 1 1 0]

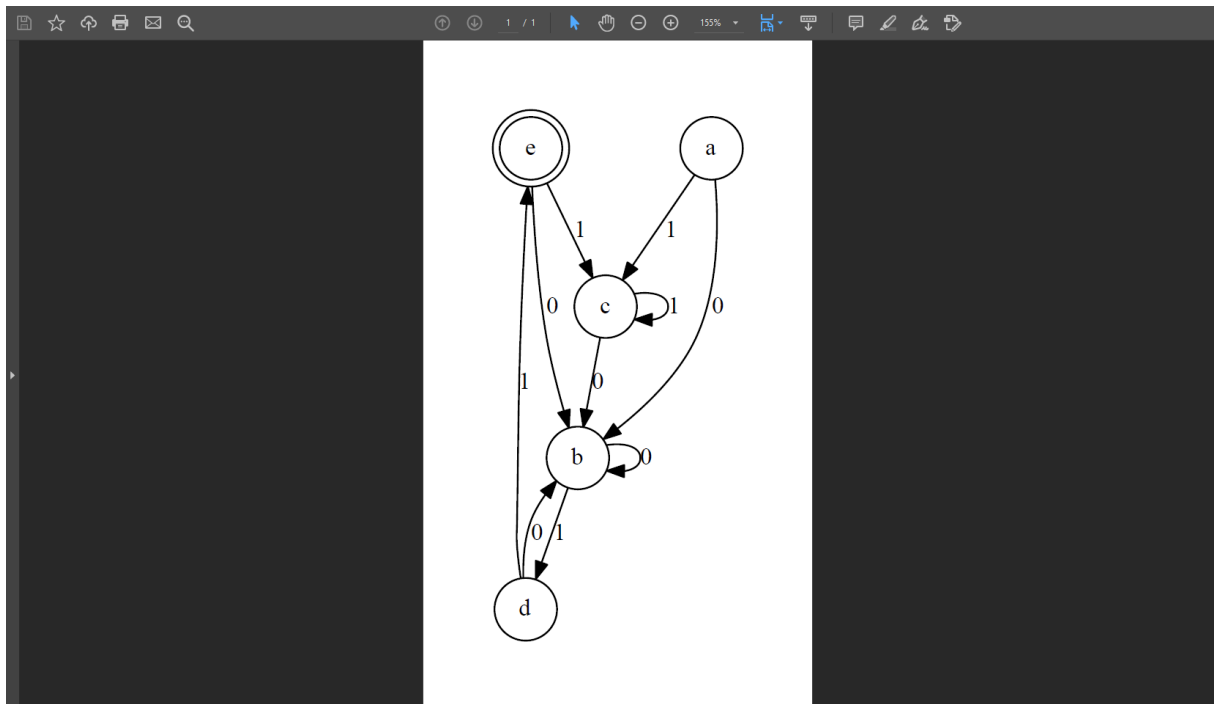
Θέλουμε να εξετάσουν αν το πρόγραμμα μας μπορεί να ανταπεξέλθει στις απαιτήσεις του παραπάνω αυτομάτου, οπότε περιγράφουμε το αυτόματο στο αρχείο μας με τη παρακάτω μορφή:

```
5
0 1
a
e
a 0 b
a 1 c
b 0 b
b 1 d
c 1 c
c 0 b
d 0 b
d 1 e
e 1 c
e 0 b
```

και εισαγουμε τη φράση: « 0 1 1 » η οποία είναι αποδεκτή:

```
*****
HELLO DEAR USER, WELCOME TO DFA PYTHON CLI PROGRAM
*****
Author: Charalampos Asimakopoulos
AM: 141098
Email: csl41098@uniwa.gr - xarhsasi@gmail.com
Github: https://github.com/HarrysAsi/DFA-UNIWA.git
Please enter a phrase: 011
You entered: 011
*****
Total States: 5Symbols: 0 lFirst state: aFinal states: e
*****
Given word: ['0', '1', '1']
Accepted word, good job!
Press Enter to Exit...
```

Και εμφανίζεται σχεδιασμένο το αυτόματο σε αρχείο .pdf όπως φαίνεται παρακάτω:



Το οποίο όπως βλέπουμε είναι το επιθυμητό αυτόματο και στη συνέχεια δημιουργεί το αρχείο με τον πίνακα μετάβασης όπως φαίνεται παρακάτω:

A1		fx node					
	A	B	C	D	E	F	G
1	node	state	next node	is first state	is final state		
2	a	0	b	TRUE	FALSE		
3	a	1	c	TRUE	FALSE		
4	b	0	b	FALSE	FALSE		
5	b	1	d	FALSE	FALSE		
6	c	1	c	FALSE	FALSE		
7	c	0	b	FALSE	FALSE		
8	d	0	b	FALSE	FALSE		
9	d	1	e	FALSE	FALSE		
10	e	1	c	FALSE	TRUE		
11	e	0	b	FALSE	TRUE		
12							

Έπειτα εισάγουμε την μή αποδεκτή φράση: « 0 1 1 0 »

```
C:\Users\Harrys\Anaconda3\envs\DFA\python.exe C:/Users/Harrys/PythonProjects/DFA/main.py
*****
HELLO DEAR USER, WELCOME TO DFA PYTHON CLI PROGRAM
*****
Author: Charalampos Asimakopoulos
AM: 141098
Email: csl41098@uniwa.gr - xarhsasi@gmail.com
Github: https://github.com/HarrysAsi/DFA-UNIWA.git
Please enter a phrase: 0110
You entered: 0110
*****
Total States: 5 Symbols: 0 | First state: a Final states: e
*****
Given word: ['0', '1', '1', '0']
Non accepted word, please try again!
Press Enter to Exit...
```

Την οποία σωστά δε δέχεται ως αποδεκτή.

Οδηγίες εγκατάστασης

- 1) Εγκατάσταση Python3.7 (<https://www.python.org/downloads/release/python-370/>)
- 2) Download Virtual Enviroment (<https://docs.python.org/3/library/venv.html>)
- 3) Extract Zip Folder ή clone the project from the git repository
(<https://www.python.org/downloads/release/python-370/>) εκτελώντας:
`git clone https://www.python.org/downloads/release/python-370/`
- 4) Μέσα στον φάκελο του project εκτελέστε όλες τις παρακάτω εντολές και δημιουργήστε το Virtual Enviroment Folder εκτελώντας:
`python3 -m venv /venv`
- 5) Ενεργοποιήστε το Virtual Enviroment εκτελώντας:
`/venv/Scripts/activate`
- 6) Εγκαταστήστε τα πακέτα στο περιβάλλον που δημιουργήσαμε.
`pip install -r requirements.txt`
- 7) Τρέξτε το main.py αρχείο:
`python3.7 main.py`