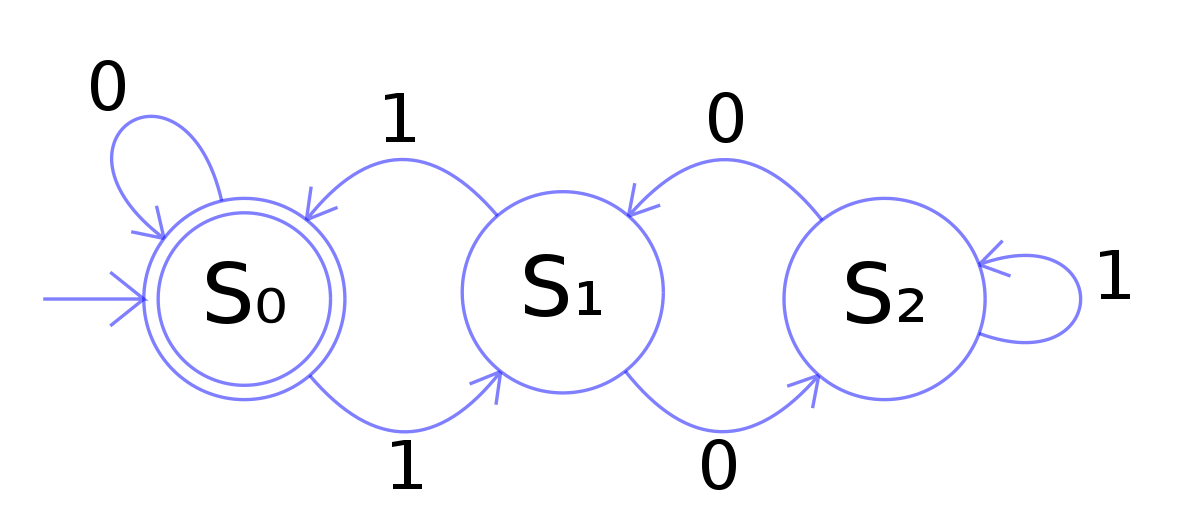
|  |
| --- |
| Ασημακόπουλος Χαράλαμπος | cs141098 |

1/30/2020

|  |
| --- |
| Email: [cs141098@uniwa.gr](mailto:cs141098@uniwa.gr) | [xarhsasi@gmail.com](mailto:xarhsasi@gmail.com)  GitHub: <https://github.com/HarrysAsi/DFA-UNIWA> |

|  |  |
| --- | --- |
| UNIWA | Θεωρία Υπολογισμού – Εργασια Εξαμήνου |



Πίνακας περιεχομένων

[Περιγραφή εργασίας 2](#_Toc31247927)

[Σενάριο εκτέλεσης 3](#_Toc31247928)

[Περιγραφή κώδικα 6](#_Toc31247929)

[Κλάση DFA 7](#_Toc31247930)

[Κλάση DFANode 9](#_Toc31247931)

[Κλάση DFATable 10](#_Toc31247932)

[Κλάση DFATableRow 11](#_Toc31247933)

[Κλάση CLIColor 12](#_Toc31247934)

[Κλάση DFAFileReader 13](#_Toc31247935)

[Κλάση DFAExcel (Προαιρετικό) 15](#_Toc31247936)

[Κλάση DFAGraph (Προαιρετικό) 16](#_Toc31247937)

[Κλάση Runnable 17](#_Toc31247938)

[Βασικό main.py 19](#_Toc31247939)

[Παράδειγμα 19](#_Toc31247940)

[Οδηγίες εγκατάστασης 22](#_Toc31247941)

# 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 οτι 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 απο τα οποία αποτελείται
* Την αρχική κατάσταση
* Τις τελικές καταστάσεις

Και την περιγραφή των καταστάσεων όπου:

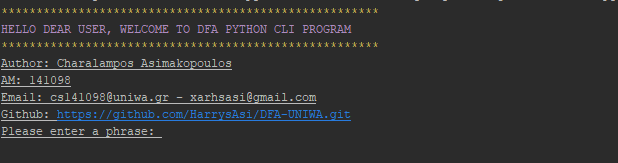
* (0 1 1) 🡪 Αν ο κόμβος 0 δεχθεί στην είσοδο του 1, τότε μεταβαίνει στον κόμβο 1
* (0 0 0) 🡪 Αν ο κόμβος 0 δεχθεί στην είσοδο του 0, τότε παραμένει στον κόμβο 0
* (1 1 2) 🡪 Αν ο κόμβος 1 δεχθεί στην είσοδο του 1, τότε μεταβαίνει στον κόμβο 2
* (1 0 0) 🡪 Αν ο κόμβος 1 δεχθεί στην είσοδο του 0, τότε μεταβαίνει 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 οχι απο το αυτόματο που περιγράφεται στο αρχείο. Εφόσον ολοκληρωθεί η εισαγωγή της συμβολοσειράς απο τον χρήστη και πατήσει “enter” τότε το πρόγραμμα ελέγχει αν η συμβολοσειρά είναι αποδεκτή απο το ντετερμινιστικό αυτόματο και ενημερώνει τον χρήστη.

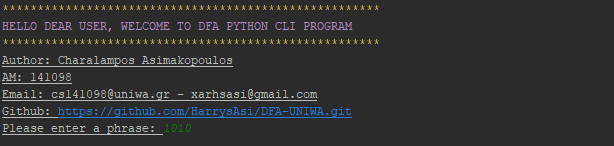
Παράλληλα, εφόσον γίνει ο έλεγχος της συμβολοσειράς το πρόγραμμα «σχεδιάζει» το αυτόματο σε ένα αρχείο τύπου PDF και εξάγει και τον πίνακα μεταβάσεων του ντετερμινιστικού αυτόματου που περιγράφηκε.

# Σενάριο εκτέλεσης

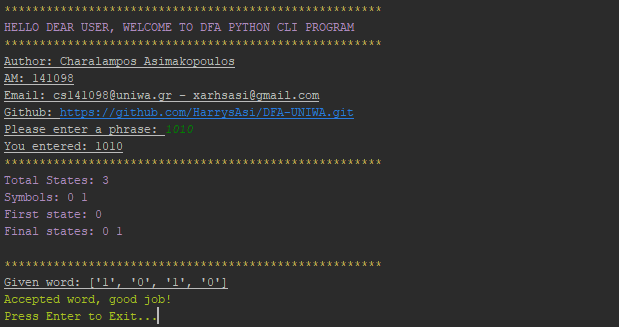
Ο χρήστης εκτελεί το πρόγραμμα και του εμφανίζονται οι σχετικές πληροφορίες:



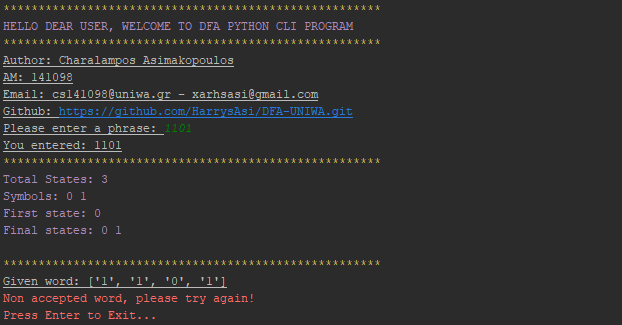
Όπου περιγράφεται ο ιδιοκτήτης του προγράμματος και κάποιες πληροφορίες σχετικά με αυτόν. Έπειρα προτρέπει τον χρήστη να εισάγει μία λέξη έτσι ώστε να ελεγχθεί απο το πρόγραμμα άν είναι αποδεκτή απο το αυτόματο ή όχι.



Έστω οτι 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.



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

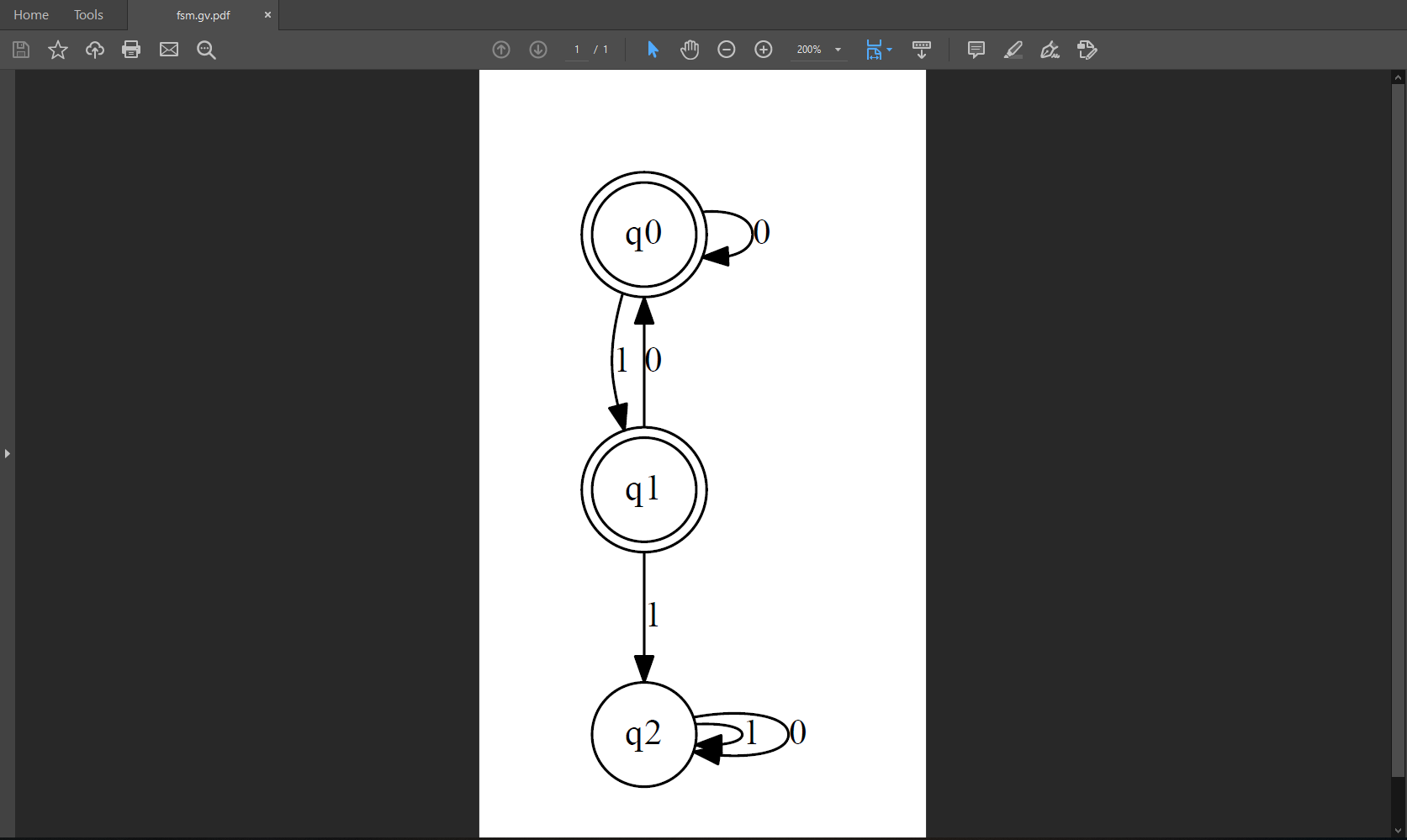


Το πρόγραμμα εκτός απο τη δημιουργία του αυτόματου και τον έλεγχο των συμβολοσειρών:

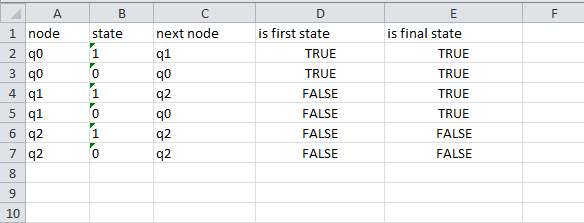
* Δημιουργεί .pdf αρχείο με το αυτόματο σχεδιασμένο
* Δημιουργεί .xlsx αρχείο με τον πίνακα μετάβασης του αυτομάτου

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:



# 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 προγραμμάτος. 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 άν 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 προστίθονται 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 περιεγραφεί 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 χρησιμοποιοηθούν 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

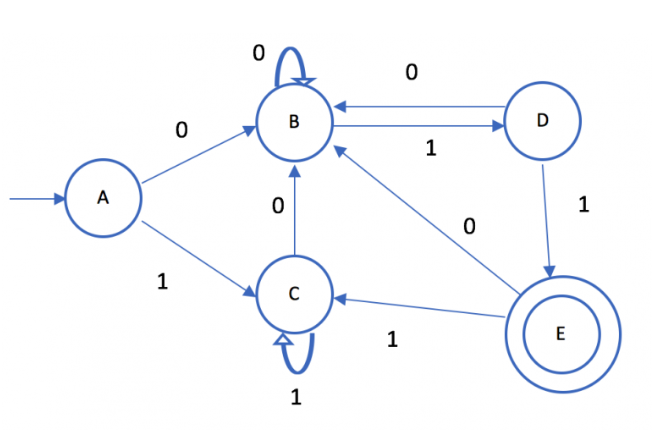
Το βασικό αρχείο εκτέλεσης το οποίο αρχικοποιεί τη κλάση runnable με prefix=”q” για τη καλύτερη αναπαράσταση των κόμβων μέσα στο αρχείο και στο γράφημα.

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 οτι 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]

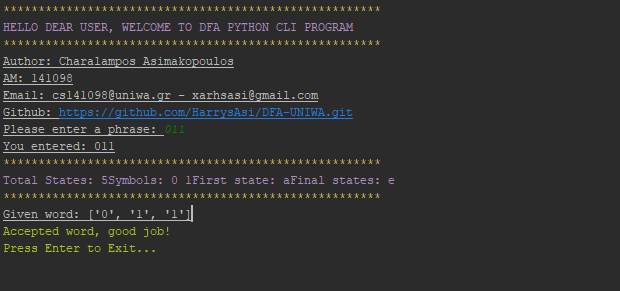
Και μερικές απο τις μή αποδεκτές φράσεις είναι:

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

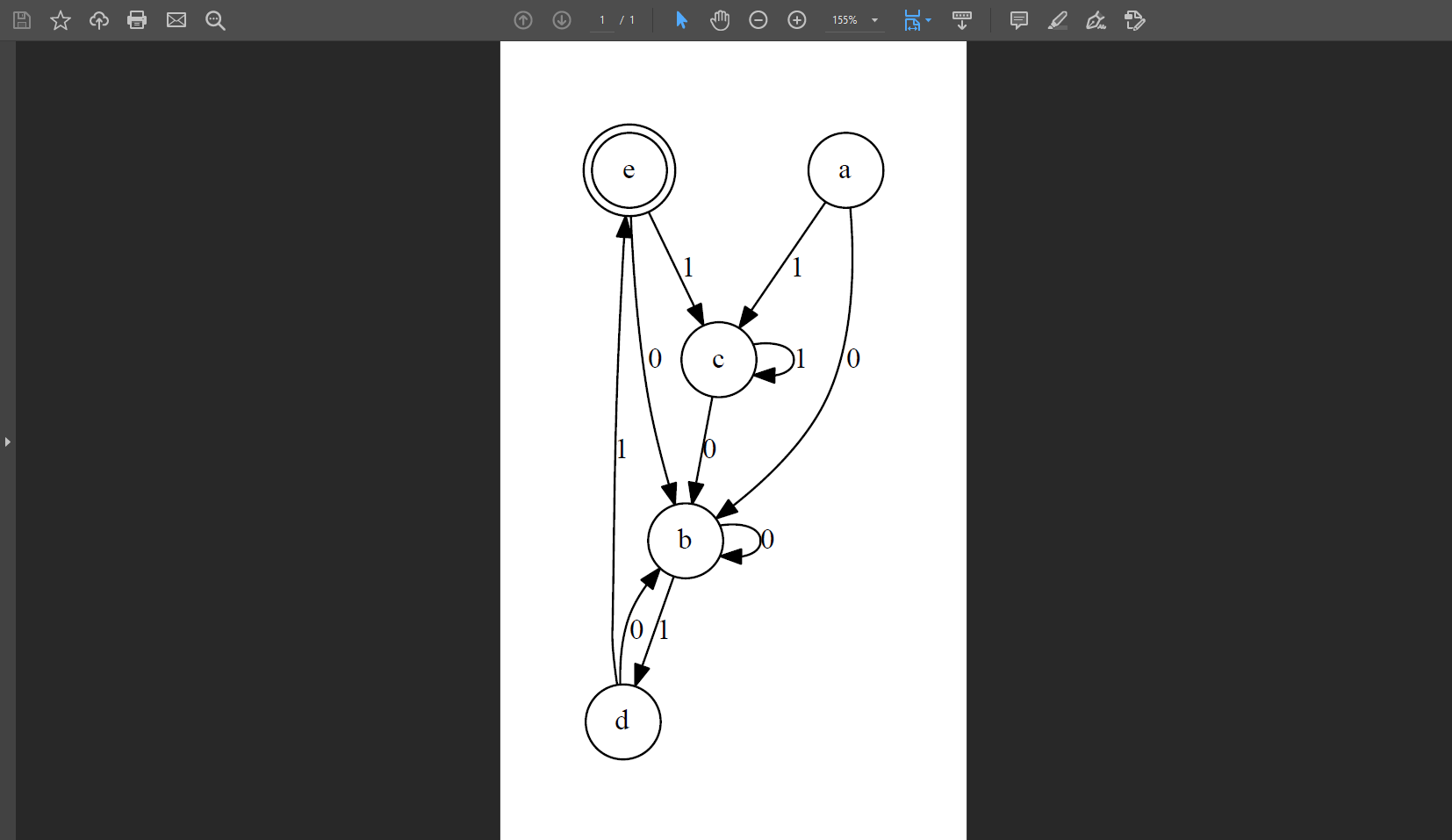
Θέλουμε να εξετάσουν αν το πρόγραμμα μας μπορεί να ανταπεξέλθει στις απαιτήσεις του παραπάνω αυτομάτου, οπότε περιγράφουμε το αυτόματο στο αρχείο μας με τη παρακάτω μορφή:

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

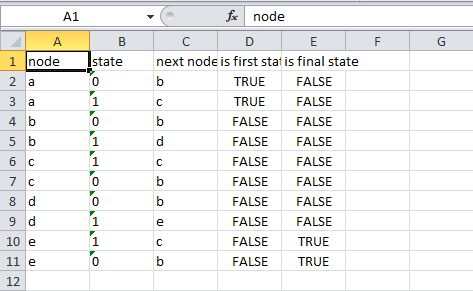
5  
0 1  
a  
e  
a 0 b  
a 1 c  
b 0 b  
b 1 d  
c 1 c  
c 0 b  
d 0 b  
d 1 e  
e 1 c  
e 0 b



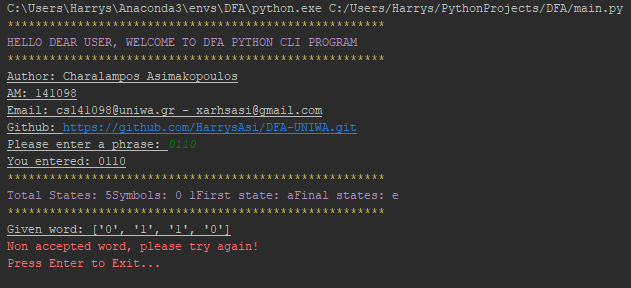
Και εμφανίζεται σχεδιασμένο το αυτόματο σε αρχείο .pdf όπως φαίνεται παρακάτω:



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



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



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/

1. Μέσα στον φάκελο του project εκτελέστε όλες τις παρακάτω εντολές και δημιουργήστε το Virtual Enviroment Folder εκτελώντας:

python3 -m venv /venv

1. Ενεργοποιήστε το Virtual Enviroment εκτελώντας:

/venv/Scripts/activate

1. Εγκαταστήστε τα πακέτα στο περιβάλλον που δημιουργήσαμε.

pip install –r requirements.txt

1. Τρέξτε το main.py αρχείο:

python3.7 main.py