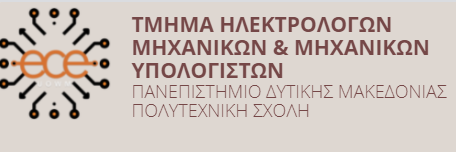
****

**2η ΕΡΓΑΣΙΑ ΣΤΟ ΜΑΘΗΜΑ: ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ**

**ΟΜΑΔΑ ΣΠΟΥΔΑΣΤΩΝ:**

**ΠΑΛΕΓΚΑΣ ΚΩΝΣΤΑΝΤΙΝΟΣ-ΒΑΣΙΛΕΙΟΣ 1530**

**ΚΥΡΙΝΑΣ ΧΑΡΙΣΙΟΣ 1495**

**ΚΑΡΑΓΚΙΩΤΗΣ ΑΘΑΝΑΣΙΟΣ 1481**

**ΓΕΩΡΓΙΤΣΑΡΟΣ ΚΩΝΣΤΑΝΤΙΝΟΣ 1451**

**ΕΞΑΜΗΝΟ: 7Ο**

Στο πρώτο στάδιο του προγράμματος δημιουργούμε την βάση γνώσης συγκεκριμένα στον φάκελο **createKnowledgeBase.**Η βάση γνώσης έχει 3 στοιχεία το **p** το οποίο είναι ο αριθμός των λεκτικών, το **l,** το οποίο είναι το μέγιστο μήκος τον προτάσεων και το  **c**, το οποίο είναι το πλήθος των προτάσεων.

Αρχικά έχουμε **returnRandomCharacters(p)** με είσοδο το p.

Αποθηκεύουμε σε μία μεταβλητή **characters** όλους τους διαθέσιμους χαρακτήρες και με μια προσωρινή μεταβλητή επιλέγουμε από τον πίνακα χαρακτήρων characters μία τυχαία μεταβλητή και την αποθηκεύουμε στην λίστα **retString** η οποία αποθηκεύει τις p τυχαίες μεταβλητές και τις επιστρέφει.

def returnRandomCharacters(p):  
 *"""Επιστρέφει ενα string μεγεθους p με τυχαιους μοναδικους χαρακητρες"""* characters = 'abcdefghijklmnopqrstunwxyz' # για αρχη το μεγιστο που μπορει να παρει το p ειναι 26  
 retString = ''  
 for i in range(p): # Επιλέγει τυχαία p λεκτικα  
 tmpChar = ''  
 while tmpChar in retString: # Συγουρεύεται οτι δεν θα επιλέξει το ίδιο λεκτικό 2 φορες  
 tmpChar = characters[random.randint(0, len(characters)1)]  
 retString += tmpChar  
 return retString

Η επόμενη συνάρτηση **retSentance(l,p,availableCharacters)** μας δημιουργεί τυχαίες προτάσεις με τα διαθέσιμα λεκτικά.

def retSentance(l, p,availableCharacters):

Η συνάρτηση δέχεται σαν ορίσματα το l ,ρ και το availablecharacters που είναι οι χαρακτήρες(λεκτικά) που επέστρεψε η παραπάνω συνάρτηση retRandomCharacters.Η λογική της συνάρτησης είναι να επιστρέψουμε μία πρόταση μήκους από 1 - l γιαύτο και αρχικοποιούμε κάθε φορά το l με τυχαίο τρόπο με μήκος 0 – l. Η μεταβλητή sentence θα αποθηκεύει τις μεταβλητές από τις οποίες θα αποτελείται η οποιαδήποτε πρόταση.

l = random.randint(1, l) # Οι προτάσεις μπορει να έχουν μήκος απο 1 εως l (πλήθος λεκτικων)  
sentance = ''

Στο παρακάτω κομμάτι ο κώδικας επιλέγει ένα τυχαίο χαρακτήρα από τους διαθέσιμους και η μεταβλητή **flipChance** επιλέγει αν το γράμμα θα είναι μικρό η κεφαλαίο.

for i in range(l): # επιλέγουμε l λεκτικά (μέγεθος πρότασης)  
 tmpChar = availableCharacters[random.randint(0, p - 1)]  
 flipChance = random.randint(0, 1)  
 if (flipChance == 1):  
 tmpChar = tmpChar.swapcase()

Στη συνέχεια έχουμε τον έλεγχο που βλέπουμε αν το λεκτικό υπάρχει ήδη στην πρόταση,σε αυτή την περίπτωση ξαναεπιλέγεται ένα καινούριο λεκτικό στην τύχη και όταν τελείωσει με την δημιουργία της πρότασης την επιστρέφει.

while (tmpChar in sentance or tmpChar.swapcase() in sentance): # αν το λεκτικό που επιλεχθηκε ήδη υπάρχει επιλέγουμε αλλο  
 tmpChar = availableCharacters[random.randint(0, p - 1)]  
 flipChance = random.randint(0, 1)  
 if (flipChance == 1):  
 tmpChar = tmpChar.swapcase() # Αναστρέφουμε τον χαρακτήρα  
 sentance += tmpChar # κάθε νεο επιλεγμένο προσθέτεται στην πρόταση  
  
return sentance

Στην συνέχεια έχουμε την συνάρτηση

def readKnowledgeBase():

η οποία διαβάζει το αρχείο και το αρχικοποιεί σε μορφή πίνακα τον οποίο επιστρέφει και ταυτόχρονα επιστρέφει τα λεκτικά από τα οποία αποτελούνται όλες οι προτάσεις .Σημαντικό εδώ να τονίσουμε ότι τις 2 πρώτες γραμμές τις προσπερνάμε διότι η πρώτη είναι τα στοιχεία του αρχείου και η δεύτερη τα λεκτικά.

3,10,3  
lwq Διαθέσιμα λεκτικά  
LQw  
wlq  
q  
Qw  
wLq  
L  
W  
LQW  
wQL  
qWL

for line in kbfile:  
 if (i==0 or i==1): #αγνοούμε τις 2 πρώτες γραμμές για να κρατήσουμε μόνο τη Βάση γνώσης  
 if (i == 1):  
 characters = line  
 i+=1  
 continue

Μία πολύ χρήσιμη συνάρτηση είναι η:

def retKBcharacteristics():

η οποία διαβάζει την πρώτη γραμμή του κώδικα και επιστρέφει τα στοιχεία p,c και l.

kbfile = open('knowledgeBase.txt', 'r')  
characteristics = kbfile.readline()  
kbfile.close()  
p, c, l = characteristics.split(',')  
  
return int(p), int(c), int(l)

Η κύρια συνάρτηση που μας απασχολεί είναι η **createKnowledgeBase(p,c,l)**

def createKnowledgeBase(p, c, l):

Αρχικά επιλέγει τυχαίους χαρακτήρες.

availableCharacters =returnRandomCharacters(p)

Στην συνέχεια δημιουργέι c προτάσεις (βάζουμε c+1 διότι c σκέτο δημιουργεί c-1 προτάσεις) και τις αποθηκεύουμε στην λίστα μας kb

for i in range(c+1):  
 sentance = ''  
 while sentance in kb:  
 sentance = retSentance(l, p,availableCharacters)  
 kb.append(sentance)

Στην συνέχεια δημιουργούμε ένα αρχείο **KnowledgeBase.txt** στο οποίο γράφουμε στη πρώτη γραμμή τα στοιχεια p,c,l και στην δεύτερη γράφουμε τους διαθέσιμους χαρακτήρες,στην 3η γραμμή και έπειτα γράφουμε τις c προτάσεις μας.

**Τώρα θα ασχοληθούμε με τον Gsat κώδικα.**

Αρχικά έχουμε την συνάρτηση της δομής δεδομένων η οποία ελέγχει ποιες προτάσεις είναι F και ποιες είναι Τ.Οι είσοδοι της είναι οι διαθέσιμοι χαρακτήρες (λεκτικά),η βάση γνώσης και οι Boolean τιμές του κάθε χαρακτήρα.

def data\_structure(chars,kb,binaryvalues):

Ελέγχουμε αν το κάθε γράμμα του πίνακα chars, υπάρχει στην πρόταση της βάσης γνώσης κεφαλαίο η μικρό,εάν είναι μικρό και είναι true αυτομάτος όλη η πρόταση θα βγαίνει true και θα αποθηκεύεται στην λίστα **lst,** αλλιώς αν είναι false συνεχίζουμε στο επόμενο γράμμα ,εάν είναι κεφαλαίο τότε, αν το μικρο του είναι false αυτό θα είναι true και η πρόταση θα βγει αυτομάτος true και θα αποθηκεύεται, αλλιώς θα προχωρήσουμε στο επόμενο γράμμα.

for i in range(c):  
 k = 0  
 while (k <= p): # ελέγχουμε αν το κάθε στοιχείο του chars είναι ίσο με το  
 if chars[k] in kb[i]:

if chars[k] in kb[i]: # εαν το στοιχείο j του chars είναι ίσο με το kb πήγαινε  
 if binaryvalues[k] == True: # εάν είναι true τότε όλη η πρόταση θα είναι true οπότε δεν χρειάζεται περεταίρο αναζήτηση προχωράμε στην επόμενη  
 lst.append(bool(1))  
 break  
elif chars[k].swapcase() in kb[i]: # εαν το κεφαλαίο στοιχείο του chars είναι ίσο με το kb  
 if binaryvalues[k] == False:  
 lst.append(bool(1))  
 break

Αν όλα βγαίνουν false τότε ολόκληρη η πρόταση βγαίνει false και στην **lst** αποθηκεύουμε false bool τιμή και προχωράμε στο επόμενο i δηλαδή στην επόμενη πρόταση.

if (k == p):  
 lst.append(bool(0))

Στο τέλος μετράμε πόσες προτάσεις βγαίνουν false και τις αποθηκεύουμε στην μεταβλητή counter η οποία προστίθεται στην λίστα lst σαν τελευταία στήλη ,τέλος επιστρέφουμε την λίστα.

Η κύρια συνάρτηση gsat έχει σαν δεδομένο τις προτάσεις maxsentences( c ) maxtries και maxflips

def gsat(maxsentences,maxTries,maxFlips):

το c το ορίζουμε ως max sentences διότι αν ο χρήστης δώσει μία πρόταση θα έχουμε c+1,αν δεν γίνει entailed τότε το c θα παραμείνει ως έχει και δεν θα αυξηθεί.Στην ουσία το maxsentences λειτουργεί ως μία προσωρινή μεταβλητή c.

Για αρχή έχουμε μία επανάληψη που ορίζει τις μέγιστες προσπάθειες που θα γίνουν για να βρεθεί λύση,σε κάθε προσπάθεια θα αρχικοποιούνται οι μεταβλητές binaryvalues με τυχαίες τιμές.

for i in range(p):#αναθέτουμε random bool τιμές  
 binaryvalues.append(bool(random.getrandbits(1)))  
print(binaryvalues)

Αμέσως μετά συνεχίζουμε με τις maxflips που είναι οι συνολικές αλλαγές των τιμών.Αρχικοποιούμε τον πίνακα data ο οποίος θα κρατάει τα αποτελέσματα των προτάσεων και τα κόστη της κάθε αλλαγής, την λίστα **tmpointer** που κρατάει τις κινήσεις που επιφέρουν το μικρότερο κόστος(θα γίνει καλύτερη επεξήγηση στην συνέχεια) και την λίστα tmplist που κρατάει το κόστος της κάθε αλλαγής μεταβλητής.

data=[]#πίνακας που κρατάει το αποτέλεσμα της πρότασης  
  
for k in range(maxFlips):  
 tmpointer=[] #μεταβλητή που έχει δείκτη/ες από για τις αλλαγές με το μικρότερο κόστος  
 tmplist=[] #λίστα για να κρατάει τα κόστη των κινήσεων  
 data.append(data\_structure(characters,kb,binaryvalues))

Στην παρακάτω επανάληψη βρίσκουμε το κόστος της αλλαγής του κάθε χαρακτήρα και αποθηκεύουμε τα κόστη αυτά στην λίστα tmplist.

for i in range(p): #συνάρτηση για να βρει τα ελάχιστο κόστος  
 values=binaryvalues.copy() # το copy χρειάζεται γιατί χωρίς αυτό ο πίνακας values παίρνει τις τιμές του binaryvalues και λειτουργεί σαν δείκτης για τον binary  
 flip(values,i)  
 tmplist.append(data\_structure(characters,kb,values)[-1])

Η συνάρτηση flip αντιστρέφει τις τιμές των λεκτικών, δηλαδή αν είναι true γίνεται false και το αντίθετο.

def flip(values,i):  
 binaryvalues=values  
 if binaryvalues[i] == True:  
 binaryvalues[i] = False  
 elif binaryvalues[i] == False:  
 binaryvalues[i] = True

Στην συνέχεια βρίσκουμε το ελάχιστο κόστος, δηλαδή ποια αλλαγή χαρακτήρα θα μετατρέψει τις περισσότερες προτάσεις σε T και το αποθηκεύουμε στην μεταβλητή mincost.

for i in tmplist: #συνάρτηση για να βρίσκει το μικρότερο κόστος  
 if(i < mincost):  
 mincost = i

Στην συνέχεια συγκρίνουμε την κάθε τιμή κόστους της λίστας tmplist με την μικρότερη και αν ισχύει η ισότητα τότε αποθηκεύουμε την θέση της tmplist στον πίνακα **tmpointer** .

for i in range(len(tmplist)): #λίστα για να κρατάει τους δείκτες των τιμών όπου η αλλαγή τους έχει το μικρότερο κόστος  
 if(tmplist[i] == mincost):  
 tmpointer.append(i)  
print("δείκτες",tmpointer)

**Θα συνεχίσω με ένα παράδειγμα για την καλύτερη κατανόηση.**

Αν έχουμε τον πίνακα χαρακτήρων **characters** ‘abc’ θέση 0🡪a , θέση 1->b , θέση 2->c

Ο πίνακας **binaryvalues** έχει τιμές [T,F,F] θέση 0->Τ δηλαδή το λεκτικό a έχει τιμή T και ούτω καθεξής.

Ο πίνακας **tmplist** έχει τιμές **[3,2,2]** δηλαδή αν αλλάξουμε την τιμή του a από Τ σε F το κόστος θα είναι 3 , αν αλλάξουμε την τιμή του b από F σε Τ τότε το κόστος είναι 2, το ίδιο με την αλλαγή του c.

Τέλος έχουμε τον πίνακα **tmpointer** που θα έχει τους δείκτες του πίνακα tmplist με τα μικρότερα κόστη δηλαδή tmpointer=[1,2]

Εάν έχουμε ένα mincost=2 και οι αλλαγή της κάθε μεταβλητής μας επιφέρει μεγαλύτερο κόστος από 2,τότε έχουμε πέσει σε τοπικό μέγιστο και αρχικοποιούμε ξανά τιμές για τα λεκτικά

if len(tmpointer) == 0:  
 break

Στην συνέχεια αλλάζουμε το λεκτικό όπου η αλλαγή του επιφέρει το μικρότερο κόστος ,αν είναι παραπάνω από ένα επιλέγουμε τυχαία

if(len(tmpointer)>1):  
 flip(binaryvalues,tmpointer[random.randint(0,len(tmpointer)-1)])# επιλέγουμε τυχαία ποια μεταβλητή θα αλλάξει  
else:  
 flip(binaryvalues,tmpointer[0])

Τέλος εάν το μικρότερο κόστος είναι ίσο με το 0 τότε επιστρέφεται Τ δηλαδή βρέθηκε λύση άρα όλες οι προτάσεις είναι Τ και η πρόταση που δώσαμε ΔΕΝ γίνεται entailed

if mincost == 0:  
 data.append(data\_structure(characters, kb, binaryvalues))  
 print(data)  
 print("Congrats every sentence in the KB is True ")  
 return True

Έφόσων είδαμε την πρώτη φάση (τον Gsat) προχοράμε τώρα στον αλγόριθμο ανάλυσης, οποίος καλείτε αφού ο gsat δεν καταφέρει να εξάγει κάποιο αποτέλεσμα για την λογική κάλυψη.

**Ο αλγόριθμος ανάλυσης** βασίζεται κυρίως στην συνάρτηση πλήρης ανάλυσης sentenceFullResolusion():

Η συνάρτηση αυτή παίρνει σαν είσοδο δύο προτάσεις, ξεκινά αφαιρώντας τα κενά τους και μετά τις ενώνει σε μία πρόταση.

Αφού τις ενώσει ελέγχει για κάθε χαρακτήρα (λεκτικό) της πρότασης αν υπάρχει ο ανάστροφός του (αν p, ελέγχουμε για P, αν P ελέγχουμε για p).

Αν υπάρχει αντικαθιστόυμε τον χαρακτήρα, αλλά και τον ανάστροφό του με έναν άδειο χαρακτήρα, ουσιαστικά τους αφαιρούμε απο την πρόταση.  
Επίσης θέτουμε το flag validCase True, για να δείξουμε οτι γινόταν πλήρης ανάλυση μεταξύ των δύο προτάσεων και υπάρχει αποτέλεσμα να επιστρέψουμε.

Εξηγόντας την συνάρτηση στην οποία βασίζεται ο αλγόριθμος της ανάλυσης πάμε τώρα να εξηγίσουμε τον ίδιο.

Σαν είσοδος έρχεται η βάση γνώσης και το λεκτικό για το οποίο θέλουμε ελέγξουμε αν καλύπτετα λογικά απο την βάση γνώσης.

Αρχικά φτιάχνουμε ένα αντίγραφο της βάσης γνώσης και δουλεύουμε πάνω σε αυτό.

Τοποθετούμε προσορινά το ανάστροφο του λεκτικού που ήρθε σαν είσοδος στην αρχή της βάσης γνώσης έτσι ώστε να του δώσουμε προτεραιότητα στην ανάλυση και για να ελέγξουμε αν η πρόταση ΚΓ^-α είναι μη ικανοποίησημη. (α = λεκτικό εισόδου)

Το κύριο κομμάτι του αλγορίθμου είναι η διπλή δομή λούπας, μια δομή επανάληψης εμφολευμένη μέσα σε μία άλλη.

Σε αυτήν την δομή κάνουμε πλήρης ανάλυση σε κάθε πρόταση της ΒΓ με όλες τις άλλες προτάσεις.

Στην περίπτωση που το αποτέλεσμα είναι αποδεκτό το προσθέτουμε σε έναν προσορινό πίνακα result.

Αν το αποτέλεσμα της ανάλυσης που προσθέθηκε στον προσορινό πίνακα είναι άδειος χαρακτήρας (άτοπο δηλαδή) και άρα η πρόταση ΚΓ ^ -α είναι μη ικανοποιήσημη, αφαιρούμε τον προσορινό ανάστροφο του λεκτικού που εισάγαμε στην αρχή και επιστρέφουμε στο κύριο πρόγραμμα True για να δείξουμε ότι γίνεται λογική κάλυψη για το λεκτικό που ήρθε σαν είσοδο και την νέα βάση γνώσης (με όσες νέες αναλύσεις προέκυψαν).

Αλλίως προσθέτουμε τα προσορινά αποτελέσματα του result στο αντίγραφο της ΒΓ και συνέχιζουμε σε νέα επανάληψη με άλλη πρόταση.

Όταν και άμα τελειώσει αυτή η εμφολευμένη επανάληψη σημαίνει οτι δεν καταφέραμε να βρούμε κάποιο άτοπο και άρα δεν ισχυεί η λογική κάλυψη και έτσι αφαιρούμε τον προσορινό ανάστροφο του λεκτικού, επιστρέφουμε False και None(null) αντί για την ΒΓ.