from sympy import Matrix  
from collections import namedtuple  
from itertools import chain  
import numpy as np  
import sympy as sym  
  
  
\_UNSUPPORTED = 'QDZ'  
  
# Ορισμός Στοιχείων Κυκλώματος  
\_Element = namedtuple('\_Element', 'Name Node1 Node2 Value')  
\_Vsource = namedtuple('\_Vsource', 'Name Node1 Node2 Value')  
\_Isource = namedtuple('\_Isource', 'Name Node1 Node2 Value')  
\_Opamp = namedtuple('\_Opamp', 'Name Node1 Node2 Node3')  
  
sym.init\_printing(use\_unicode=False)  
  
def mnaSolve(fname):  
  
 with open(fname, 'r') as file:  
 d = np.loadtxt(file, delimiter=' ',  
 dtype={'names': ('Name', 'N1', 'N2', 'ARG3'),  
 'formats': ('U15', 'i4', 'i4', 'U15')})  
  
 # Αρχικοποίηση Λιστών Πινάκων  
 numNode = 0 # Αριθμός Κόμβων, ο κόμβος #0 είναι στη γείωση.  
 Elements = []  
 Vsources = []  
 Opamps = []  
 Isources = []  
  
 # Διάβασμα Αρχείου Κυκλώματος  
 for name, n1, n2, arg3 in d:  
 desig = name[0]  
 if desig in 'RLC': # Αντιστάσεις Πηνία Πυκνωτές  
 try:  
 val = float(arg3)  
 except ValueError:  
 val = np.nan  
 Elements.append(\_Element(Name=name, Node1=n1, Node2=n2, Value=val))  
 elif desig == 'V': # Πηγές Τάσης  
 try:  
 val = float(arg3)  
 except ValueError:  
 val = np.nan  
 Vsources.append(\_Vsource(Name=name, Node1=n1, Node2=n2, Value=val))  
 elif desig == 'I': # Πηγές Ρεύματος  
 try:  
 val = float(arg3)  
 except ValueError:  
 val = np.nan  
 Isources.append(\_Isource(Name=name, Node1=n1, Node2=n2, Value=val))  
 elif desig == 'O': # Ενισχυτές  
 Opamps.append(\_Opamp(Name=name, Node1=n1, Node2=n2, Node3=arg3))  
 elif desig in \_UNSUPPORTED:  
 raise ValueError('Unsupported component type {!s}'.format(name))  
 else:  
 raise ValueError('Unknown component type {!s}'.format(name))  
 numNode = max(n1, n2, numNode)  
  
 numV = len(Vsources)  
 numO = len(Opamps)  
 nodes = list(range(numNode))  
  
 # Προκαθορισμός Χώρου για τους Πίνακες της MNA  
 G = sym.zeros(numNode, numNode)  
 V = sym.zeros(numNode, 1)  
 I = sym.zeros(numNode, 1)  
 if (numV + numO) != 0:  
 B = sym.zeros(numNode, numV + numO)  
 C = sym.zeros(numV + numO, numNode)  
 D = sym.zeros(numV + numO, numV + numO)  
 E = sym.zeros(numV + numO, 1)  
 J = sym.zeros(numV + numO, 1)  
  
 # Πίνακας G  
 # Αρχικοποίηση του πίνακα με μηδενικά.  
 # Γέμισμα του πίνακα με τις τιμές αγωγιμότητας του αρχείου κυκλώματος.  
 for element in Elements:  
 n1 = element.Node1 - 1  
 n2 = element.Node2 - 1  
 desig = element.Name[0]  
 # Δημιουργία χαρακτήρα με την αγωγιμότητα του τρέχοντος στοιχείου.  
 if desig == 'R':  
 g = '1/{}'.format(element.Name)  
 elif desig == 'L':  
 g = '1/s/{}'.format(element.Name)  
 elif desig == 'C':  
 g = 's\*{}'.format(element.Name)  
 else:  
 raise ValueError('Unknown component type {!s}'.format(element.Name))  
 # Εάν το στοιχείο δε συνδέεται με τη γείωση, το αφαιρούμε από τον πίνακα  
 if n1 != -1 and n2 != -1:  
 G[n1, n2] = '{}-{}'.format(G[n1, n2], g)  
 G[n2, n1] = '{}-{}'.format(G[n2, n1], g)  
 # Εάν ο κόμβοι 1,2 συνδέονται με τη γείωση, προσθέτουμε το στοιχείο στη διαγώνιο.  
 if n1 != -1:  
 G[n1, n1] = '{}+{}'.format(G[n1, n1], g)  
 if n2 != -1:  
 G[n2, n2] = '{}+{}'.format(G[n2, n2], g)  
  
 # Πίνακας I  
 for j in nodes:  
 for isource in Isources:  
 if isource.Node1 == j + 1:  
 I[j] = '{}-{}'.format(I[j], isource.Name)  
 elif isource.Node2 == j + 1:  
 I[j] = '{}+{}'.format(I[j], isource.Name)  
  
 # Πίνακας V  
 for i in nodes:  
 V[i] = 'v\_{}'.format(i + 1)  
  
 # Πίνακας B  
 if (numV + numO) != 0:  
 for i, vsource in enumerate(Vsources): # Ανατρέχουμε για όλες τις ανεξάρτητες πηγές τάσης.  
 for j in nodes: # Για κάθε κόμβο, αν ο κόμβος είναι ο 1ος, βάζουμε 1, διαφορετικά -1.  
 if vsource.Node1 == j + 1:  
 B[j, i] = '1'  
 C[i, j] = '1'  
 elif vsource.Node2 == j + 1:  
 B[j, i] = '-1'  
 C[i, j] = '-1'  
  
 # Ενισχυτές στον Πίνακα Β  
 for i, opamp in enumerate(Opamps):  
 for j in nodes:  
 if int(opamp.Node3) == j + 1:  
 print("\n opamp.node3 = ", int(opamp.Node3))  
 B[j, i + numV] = '1'  
 else:  
 B[j, i + numV] = '0'  
  
 # Ενισχυτές C  
 for i, opamp in enumerate(Opamps):  
 for j in nodes:  
 if opamp.Node1 == j + 1:  
 C[i + numV, j] = '1'  
 elif opamp.Node2 == j + 1:  
 C[i + numV, j] = '-1'  
 else:  
 C[i + numV, j] = '0'  
  
 # Πίνακας E  
  
 for i, vsource in enumerate(Vsources):  
 E[i] = vsource.Name  
  
 # Πίνακας J  
 for i, vsource in enumerate(Vsources):  
 J[i] = 'I\_{}'.format(vsource.Name)  
 for i, opamp in enumerate(Opamps):  
 J[i + numV] = 'I\_{}'.format(opamp.Name)  
  
 # Δημιουργία των 3 πινάκων της εξίωσης Ax=Z  
 A = G.row\_join(B).col\_join(C.row\_join(D))  
 X = V.col\_join(J)  
 Z = I.col\_join(E)  
 else:  
 A = G  
 X = V  
 Z = I  
  
 # Αποθήκευση της λύσης στον πίνακα V που θα τυπώσουμε  
 # Printing the Solution Matrix  
 print('\nMNA Finished - Here are your results')  
 print('\nThe A matrix is : \n', np.matrix(A))  
 print('\nThe z matrix is : \n', np.matrix(Z))  
 print('\nThe B matrix is : \n', np.matrix(B))  
 print('\nThe C matrix is : \n', np.matrix(C))  
 print('\nThe D matrix is : \n', np.matrix(D))  
 print('\nThe G matrix is : \n', np.matrix(G))  
 print('\n The v matrix is \n',np.matrix(V))  
 print('\n The j matrix is\n', np.matrix(J))  
 print('\n The i matrix is\n', np.matrix(I))  
 print('\n The e matrix is\n', np.matrix(E))  
  
 V = sym.factor(A.inv() \* Z)  
  
 # Επιστροφή Τιμών  
 Sol = sym.cancel(  
 V.subs([(elem.Name, elem.Value) for elem in chain(Elements, Vsources, Isources) if elem.Value is not np.nan]))  
  
  
 # Εκτύπωση του κυκλώματος  
 print('\n The Circuit:')  
 for row in d:  
 print('{Name} {N1} {N2} {arg3}'.format(Name=row['Name'], N1=row['N1'],  
 N2=row['N2'], arg3=row['ARG3']))  
  
 print('\n The x Matrix:')  
 for entry in X[:]:  
 print(entry)  
 print('')  
  
 return Sol, V, A, X, Z, nodes, Elements, Vsources, Isources  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 import sys  
  
 rets = ['Sol', 'V', 'A', 'X', 'Z', 'nodes', 'Elements', 'Vsources', 'Isources']  
 fname = sys.argv[1]  
 Sol, V, A, X, Z, nodes, Elements, Vsources, Isources = mnaSolve(fname)  
  
 print('\nThe Solution Matrix is :\n', np.matrix(V))  
  
 sys.exit()