Άσκηση 3

Για την τρίτη άσκηση ξεκινήσαμε δημιουργώντας ένα txt αρχείο το οποίο έχει μέσα τα ονόματα των πρακτόρων και τις συνδέσεις μεταξύ τους όπως αναγράφονται στην άσκηση:

```
file_path = 'agents.txt'
start = encryptAgentNames(file_path)
my_graph = start.getGraph()
agent_numbers = start.assignNumbersToAgents()
print(f'my agent numbers are: {agent_numbers}')
```

Στην συνέχεια προχωράμε στην κωδικοποίηση των ονομάτων των πρακτόρων αυτών και στην δημιουργία ενός δέντρου με τις συνδέσεις αυτές

```
class encryptAgentNames:
    def __init__(self,data):
        self.graph = {}
        self.agent_numbers = {}
        with open(data, 'r') as file:
            for line in file:
                sender, receiver= line.strip().split(',')
                if sender not in self.graph:
                    self.graph[sender] = []
                if receiver not in self.graph:
                    self.graph[receiver] = []
                self.graph[sender].append(receiver)
                self.graph[receiver].append(sender)

    def assignNumbersToAgents(self):
        self.agent_number = 1
        for agent in self.graph:
            self.agent_numbers[agent] = self.agent_number
            self.agent_number +=1
        return self.agent_numbers

    def CreateTree(self):
        treeConnections = []
        for agent in self.graph:
            agent_num = self.agent_numbers[agent]
            agent_conn = self.graph[agent]
            for i in range(len(agent_conn)):
                connection = [agent_num,self.agent_numbers[agent_conn[i]]]
                if [self.agent_numbers[agent_conn[i]],agent_num] not in treeConnections:
                    treeConnections.append(connection)
        return treeConnections
```

Τα αποτελέσματα για την κωδικοποίηση των ονομάτων και την δημιουργία του δέντρου:

```
ατ??/erg5/erg5.py"
my agent numbers are: {'Alpha': 1, 'Charlie': 2, 'Golf': 3, 'Bravo': 4, 'Delta': 5, 'Echo': 6, 'Fox': 7, 'Hotel': 8}
my tree´s connections are: [[1, 2], [1, 3], [1, 5], [4, 5], [5, 6], [6, 7], [6, 8]]
```

Φτιάξαμε μια κλάση για την κωδικοποίηση των αριθμών των πρακτόρων χρησιμοποιώντας την κωδικοποίηση του Prufer
Χρησιμοποιώντας μια λίστα που έχει τον αριθμό των συνδέσεων του κάθε πράκτορα (πρακτικά ο βαθμός κάθε κορυφής του δέντρου) έτσι ώστε να πετύχει η κωδικοποίηση του Prufer

```python
class myPrufer:
    def __init__(self,graph,agent_numbers,treeConnections):
        self.agent_numbers = agent_numbers
        self.num_of_con = {}
        self.treeConnections = treeConnections
        for agent in graph:
            values =  graph[agent]
            self.num_of_con[agent_numbers.get(agent)] = len(values)
```

```python
def pruferEncoding(self):
    s = []
    temp_connections = self.treeConnections
    temp_num_of_con = self.num_of_con
    for i in range(len(self.agent_numbers)-2):
        min = self.find_min_label(temp_num_of_con)
        flag = True
        j=0
        while flag:
            if temp_connections[j][0]== min  :
                s.append(temp_connections[j][1])
                temp_num_of_con[temp_connections[j][1]] -= 1
                temp_connections.remove(temp_connections[j])
                del temp_num_of_con[min]
                flag = False
            elif temp_connections[j][1]== min  :
                s.append(temp_connections[j][0])
                temp_num_of_con[temp_connections[j][0]] -= 1
                temp_connections.remove(temp_connections[j])
                del temp_num_of_con[min]
                flag = False
            j+=1
    return s
```

Αλγόριθμος που χρησιμοποιήσαμε

- **Prufer Encoding**

Input: A tree T with numerical labeling on its vertices a_i $(1 \leq i \leq n)$

Output: A Prufer sequence of length $n-2$

1) For $i = 1:1:n-2$
2) Let v the vertex with the minimum label
3) Let b_i the label of the only neighbor of vertex v
4) $T \leftarrow T - v$
5) Return $S(b_1, b_2, \dots\ b_{n-2})$

Αποτελέσματα κωδικοποίησης

```
Prufers encoding of the graph: [1, 1, 5, 5, 6, 6]
```

Για την κωδικοποίηση του μηνύματος χρησιμοποιήσαμε ρουτίνα κρυπτογράφησης τύπου ολίσθησης και στο συγκεκριμένο παράδειγμα με μυστικό κλειδί K = 2

```
coder = myCryptography(coded,2)
encrypted = coder.encryption()
print(f'Coded message sent: {encrypted}')
decrypted = coder.decryption()
print(f'Decoded message sent: {decrypted}')
```

Αποτελέσματα κωδικοποίησης και αποκωδικοποίησης

```
Coded message sent: [3, 3, 7, 7, 8, 8]
Decoded message sent: [1, 1, 5, 5, 6, 6]
```

```
class myCryptography:
    def __init__(self,s,k):
        self.s =s
        self.k=k

    def encryption(self):
        for i in range(len(self.s)):
            self.s[i] += self.k
        return self.s

    def decryption(self):
        for i in range(len(self.s)):
            self.s[i] -= self.k
        return self.s
```

Prufer Decoding

Input: A Prufer sequence of length $n-2$

Output: A tree T with numerical labeling on its vertices a_i $(1 \le i \le n)$

1) Initialize empty list P (Prufer sequence)
2) Initialize list $L = 1,2,\dots,n$
3) $F \leftarrow$ forest from n independent vertices enumerated from 1 to n
4) For $i = 1:1:n-2$
   a. Let k the minimum number in $L : \notin P$
   b. Let j the first number in P
   c. Connect by an edge the vertices k and j
   d. Delete k from L
   e. Delete the first appearance of j from P
5) Connect by an edge the vertices with the remaining numbers of P
6) Return the forest (tree) F

Και τέλος χρησιμοποιήσαμε την αποκωδικοποίηση του Prufer για να φέρουμε πάλι το δέντρο στην αρχική του κατάσταση το οποίο το πετύχαμε δημιουργώντας μια λίστα connections και υλοποιώντας τον αλγόριθμο αποκωδικοποίησης που φαίνεται δίπλα. Έτσι και τελειώνουμε την διαδικασία που μας αναφέρεται στην άσκηση:

Encoding >> Encryption >> Decryption >> Decoding

```
def pruferDecoding(self,s):
    l=[]
    connections = []
    for agent in self.agent_numbers:
        l.append(self.agent_numbers[agent])
    while s!=[]:
        flag = True
        j=0
        while flag:
            if l[j] not in s:
                connection = [s[0],l[j]]
                connections.append(connection)
                flag = False
                s.remove(s[0])
                l.remove(l[j])
            j+=1
    connection = [l[0],l[1]]
    connections.append(connection)
    return connections
```

Αποτελέσματα:

```
Prufers decoding of the graph: [[1, 2], [1, 3], [5, 1], [5, 4], [6, 5], [6, 7], [6, 8]]
```