TP 5 : Automates, dictionnaires et expressions régulières

L'objectif de ce travail est de se familiariser avec les automates d'états finis en utilisant dictionnaires et expressions régulières, objets de base des langages de script, et donc du langage Python.

Le fichier à traiter correspond à des données au format Gedcom, format de stockage de données généalogiques.

Ces données ont été réduites à plusieurs types de lignes, l'identification d'un individu par la balise `INDI`, ses noms et prénoms par la balise `NAME` et sa profession par la balise `OCCU`…

Les lignes suivantes présentent un extrait du fichier à traiter :

```
0 @4923I@ INDI
1 NAME Julien Hyacinthe/DOUANE/
1 SEX M
1 BIRT
2 DATE 14 DEC 1724
2 PLAC Bazouges-la-Pérouse,35,Ille-et-Vilaine,Bretagne,FRANCE,
1 DEAT
2 DATE @#DFRENCH R@ 26 FLOR an V
2 PLAC Bazouges-la-Pérouse,35,Ille-et-Vilaine,Bretagne,FRANCE,
1 OCCU sabotier (an V)
0 @5539I@ INDI
1 NAME Marie Joseph/GUIBERT/
1 SEX F
1 BIRT
2 DATE 16 JUN 1833
2 PLAC Bazouges-la-Pérouse,35,Ille-et-Vilaine,Bretagne,FRANCE,
1 OCCU sabotière (1856)
```

L'ensemble des fonctions écrites dans cette réalisation pratique seront testées par un appel de la fonction appliquée au fichier Gedcom fourni.

1. Automate d'états finis

Lors de ce TP, on doit construire une fonction `listePatronymes` qui calcule le nombre de fois où une profession apparait pour un patronyme (nom de famille) donné. Il y a donc une extraction de données cohérentes sur plusieurs lignes pour obtenir le résultat final.

Le résultat correspond à un tableau associatif à deux dimensions avec le nombre d'occurrences comme valeur :

```
dict[nom][profession]=nombre d'occurrences
```

La contrainte de structuration du fichier d'entrée (INDI, NAME, OCCU) est une contrainte forte (ordre, existence) rarement complètement réalisée dans de vraies données. Un type de ligne peut ainsi manquer d'un individu à l'autre.

La solution présentée s'appuie sur le modèle des **automates d'états finis** qui propose un type d'**algorithme générique** pour un type de problème, celui de la **détection d'événements ordonnés dans une suite** en s'appuyant sur la **mémorisation d'un état courant** représentée le plus couramment par une valeur numérique entière.

En terme d'implémentation, il existe deux façons de procéder, la première plus simple met l'accent sur la partie codage, la seconde plus sophistiquée déporte une partie de l'automate dans une structure de données.

2. Implémentation d'un automate de type utilisation de conditionnelles

L'automate à programmer est le suivant. On commence dans un état initial égal à `0` ; la découverte d'une ligne `INDI` permet de passer dans l'état `1`…

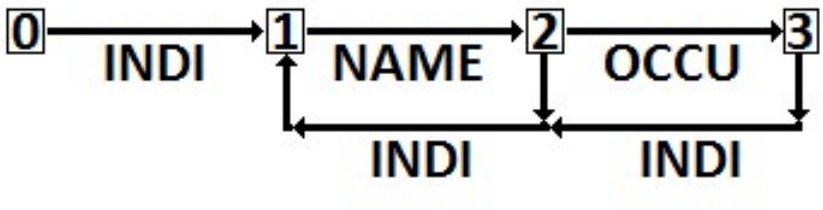

Fig. 1 : Automate d'extraction des patronymes et professions associées

```
import re
# Fonction de recherche des sabotiers
def listePatronymes(nom, dct):
    f = open(nom,'r')
    etat=0
    reg1=re.compile(r"\bINDI\b")
    # …
    for ligne in f:
        if (etat==0):     # Etat initial
            res=reg1.search(ligne)
            if res!=None: # Etat individu détecté
                etat=1
        elif (etat==1):
            pass
        elif (etat==2):
            pass
        elif (etat==3):
                res=reg1.search(ligne)
                if res!=None: # Etat individu détecté
                    etat=1
    f.close()
# Affichage d'un dictionnaire
def affichageDict(dct):
    for c in dct.keys():
        print(c," : ",dct[c])

# Programme principal
if __name__ == "__main__":
    dct={}
    listePatronymes("resultat.txt",dct)
    affichageDict(dct)
```

Q1 : Compléter la fonction `listePatronymes` qui calcule la liste des patronymes dont la profession contient le mot **sabot (sabotier**, **sabotière**, **marchand de sabots**…) en s'appuyant sur une détection des lignes intéressantes par expression régulière.

Aide :

- La fonction possèdera deux paramètres :
  - le nom du fichier Gedcom,
  - le dictionnaire résultat à une dimension qui contient en clé le patronyme et comme valeur le nombre d'occurrences du patronyme.
- Les lignes suivantes donnent un exemple d'utilisation d'un dictionnaire à une dimension :

```
# Gestion du cas où le nom n'est pas présent
if (not nom in dict):
      dict[nom]=1;
```

3. Implémentation d'un automate à l'aide d'une structure de données de description

```
import re

# Automate générique
class Automate:
    # Constructeur
    def __init__(self):
        self.etat=0
        self.struct={}
        # Description de l'automate
        self.automateDon={0: [["^(.*?)\s*$",0,self.traitDefaut]]}

    # Traitements liés à l'automate
    def traitDefaut(self,m1):
        print(m1)
        return

    # Boucle d'analyse du fichier d'entrée à l'aide d'un automate
    def analyser(self, nom):
        f = open(nom,'r')
        for ligne in f:
            for i in range(len(self.automateDon[self.etat])):
                # Recherche des expressions régulières disponibles
                # pour l'état courant
                regexp=re.compile(self.automateDon[self.etat][i][0])
                result=regexp.search(ligne)
                if result!=None:
                    # Les expressions régulières capture TOUTE une zone
                    m1=result.group(1)
                    # Appel de la fonction prévue par l'automate
                    self.automateDon[self.etat][i][2](m1)
                    # Changement de l'état de l'automate
                    self.etat=self.automateDon[self.etat][i][1]
                    # Sortir de la boucle i, le changement d'état ayant
                    # une influence sur le résultat du range
                    break
        f.close()
        return self.struct

    # Impression de la structure (un niveau pas défaut)
    def afficher(self):
       for cle1 in self.struct.keys():
           print(cle1+" : "+str(self.struct[cle1]))
```

La classe fournie ci-dessus représente un **automate générique** qui par défaut reste toujours dans l'état 0, et se contente d'afficher un fichier ligne par ligne.

Le tableau associatif `automateDon` représente la partie centrale du point de vue des données : elle indique ainsi que si la ligne courante respecte l'expression régulière fournie, on appellera la fonction `traitDefaut` et que l'on restera dans l'état initial `0`.

La méthode `analyser` correspond au cœur logiciel en explorant l'automate décrit par `automateDon`, en gérant la concordance avec l'expression régulière et l'extraction d'une sous-zone capturée, l'appel de la fonction de traitement liée et la gestion de l'évolution de la variable `etat`. Un tableau associatif `struct` est proposé par défaut comme résultat lors du fonctionnement de l'automate avec une méthode d'affichage associée.

Q2 : Identifier dans la classe fournie la boucle de balayage de lecture des lignes du fichier, la boucle de balayage des expressions régulières associées à l'état courant, le test de correspondance de la ligne avec l'expression régulière, l'extraction de la zone capturée, l'appel de la fonction de traitement associée, ainsi que la ligne permettant de faire évoluer l'état courant.

Q3 : Construire un programme principal de quelques lignes permettant d'observer le résultat obtenu par la classe `Automate` sur un fichier quelconque.

```
class AutomateComptage(Automate):
    # Constructeur
    def __init__(self):
        super().__init__()
        self.struct["Nombre occurences"]=0
        # Description de l'automate
        self.automateDon={0: [[…,0,…]]}

    # Traitement spécifique pour une ligne donnée
    def trait(self,m1):
        …
```

Q4 : Compléter la classe dérivée précédente `AutomateComptage` pour compter le nombre de lignes correspondant à la balise `INDI`. Là encore, nous ne sommes pas véritablement sur un automate, l'état restant à une valeur constante égale à `0`.

Q5 : Construire un programme principal de quelques lignes permettant d'observer le résultat obtenu par la classe `AutomateComptage` sur un fichier généalogique.

La dernière partie correspond à une réécriture de la question 5 sous forme d'un véritable automate à changement d'état en bénéficiant de l'héritage entre classes. On désire calculer pour chaque couple (patronyme, profession) le nombre d'occurrences de ce couple. Pour mémoriser un tel résultat, on utilisera un tableau associatif à deux dimensions : `dict[nom][profession]=nombre d'occurrences.`

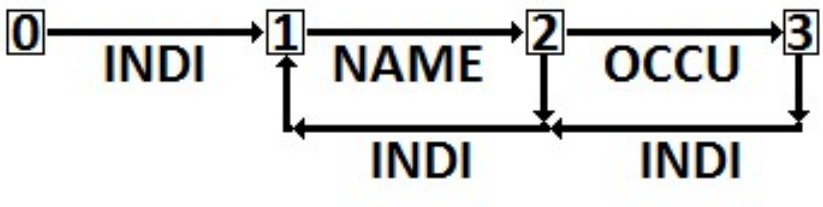

Fig. 2 : Automate d'extraction des patronymes et professions associées

```
class AutomateNomProf(Automate):
    # Constructeur
    def __init__(self):
        super().__init__()

        # Description de l'automate
        self.reIndi=["0…INDI…$",1, self.traitIndi]
        self.reName=["1…NAME…$",2, self.traitName]
        self.reOccu=["1…OCCU…$",3, self.traitOccu]
        self.automateDon={0: [self.reIndi],
                          1: [self.reName, self.reIndi],
                          2: [self.reOccu, self.reIndi],
                          3: [self.reIndi]}
```

```
    # Traitement spécifique pour une ligne donnée
    def traitIndi(self,m1):
        return
    def traitName(self,m1):
        …
    def traitOccu(self,m1):
        # Gestion du cas où le nom n'est pas présent
        …
        # Séparation des professions
        …
        # Balayage des professions séparées
        for p in prof:
            # Cas d'une profession existante
            if (…):
                …
            else:
                # Cas où la profession qui n'existe pas
                …

    # Impression de la structure
    def afficher(self):
        for cle1 in self.struct.keys():
            print(cle1," : ",end="")
            for cle2 in self.struct[cle1].keys():
                print(cle2+" ("+str(self.struct[cle1][cle2])+")",end=", ")
```

Q6 : Compléter la description de l'automate pour procéder à l'extraction des patronymes (balise `NAME`) et l'extraction des professions (balise `OCCU`).

Q7 : Compléter les méthodes `traitName` et `traitOccu`. La première mémorise uniquement dans une variable d'instance le patronyme, la seconde met à jour la structure complète avec les patronymes et les professions pour les compter.

Q8 : Construire un programme principal de quelques lignes permettant d'observer le résultat obtenu par la classe `AutomateNomProf` sur un fichier généalogique.