MISE EN CORRESPONDANCE DE VOCABULAIRES DU DOMAINE DE LA MUSIQUE

Travail de recherche sur l'alignement de vocabulaires musicaux de deux grandes institutions culturelles Françaises

Parcours : MASTER 1 INFORMATIQUE POUR LES SCIENCES IPS

Encadrant : Mr. Konstantin TODOROV
BENDJOUDI Nassim, DJERROUD Ilyes, KEBLOUTI Wissam Loubna

2017/2018

Remerciement :

Nous remercions **Mr Konstantin TODOROV**, encadreur et initiateur du projet, malgré sa charge de travail importante, il nous a encadré tout au long de notre projet. Nous tenons également à le remercier pour sa confiance, son aide et ses encouragements.

Nos sincères gratitudes à **l'équipe de laboratoire de LIRMM** Montpellier, qui nous ont donné l'opportunité de collaborer avec eux afin d'acquérir une expérience dans le travail en multitâches et en équipe.

Nous remercions également **Mr Pierre POMPIDOR** pour l'aide qu'il a pu nous apporter avec ses remarques constructives au début du projet.

Table des matières :

INTRODUCTION :

Le projet DOREMUS (Doing Reusable Musical Data) auquel nous apportons contribution, a pour but de permettre aux communautés et institutions culturelles, de disposer de modèles de connaissances communs, de les lier, de les mettre en correspondance, pour créer ce qu'on appelle des ontologies c'est-à-dire, une représentation partagée et consensuelle d'un ensemble de connaissances (voir *Figure 1*) ; en créant des catalogues d'œuvres et d'événements musicaux, partagés et connectés et en les mettant à disposition de tous.
Le projet s'appuie sur les bibliothèques de plusieurs institutions culturelles et musicales, telle que *La Bibliothèque Nationale de France* et *Radio France*, dans le but de créer un répertoire commun de connaissances d'une œuvre ou d'un opus. Permettant ainsi par la suite, la mise en place d'outils pédagogiques qui intègrent ces données référencées.

Contexte du travail de recherche :

Ce projet rentre dans le cadre du master 1 Informatique pour les sciences (IPS), plus précisément dans L'unité d'enseignement intitulée Travail d'étude et recherche (TER), qui permet de former les étudiants, par la pratique, la recherche et la conception ainsi que l'élaboration et la coopération.

Ce travail est un grand privilège, qui nous permet d'apporter notre pierre à l'édifice qu'est ce grand projet DOREMUS et de savoir que par la suite nos travaux de recherche vont être réutilisés, afin de permettre à d'autre gens d'avancer sur ce projet là ou un autre.

Cette participation très enrichissante d'un point de vue relationnel avec notre encadrant, mais aussi entre nous, le travail de groupe. Ainsi que sur le fait d'accroître nos connaissances et d'en acquérir de nouvelles pour nous former au monde professionnel par la suite.

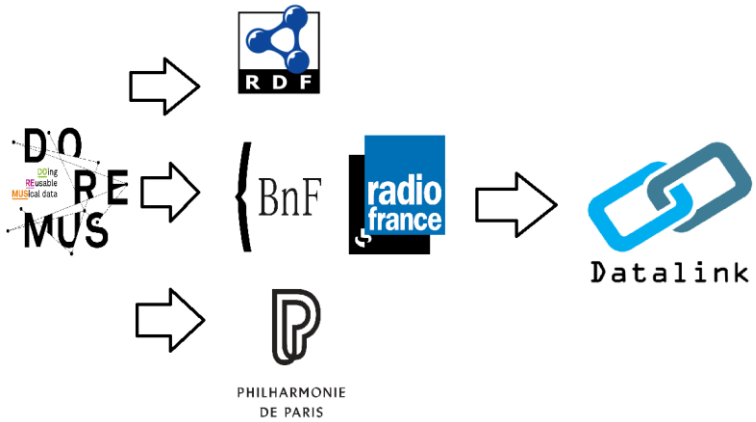

Figure 1 : Schéma global des tâches effectuées

Tâche Préliminaire :
Dresser l’état de l’art des mesures de similarités

Ce travail de documentation approfondis était nécessaire au travail d’alignement qui nous attends pour la poursuite du projet.

Lorsque l’on désire comparer deux entités, on peut s’intéresser au sens mais également à la syntaxe ressemblante entre deux termes. Et c’est partant de ce postulat là que nous nous sommes basés sur la recherche de mesures d’alignements syntaxiques, en essayant de trouver le moyen de faire concorder deux chaînes de caractères.

Différentes mesures peuvent être employées pour comparer deux ensembles de chaînes de caractères S et T selon qu’on les traite comme des chaînes de caractères, des ensembles d’éléments ou réellement comme des ensembles de chaînes de caractères. Si on les assimile à deux chaînes de caractères, alors, on peut avoir recours à « La similarité Levenshtein » (Levenshtein (1966)). Elle opère sur deux chaînes données, et retourne un nombre qui est le nombre d'insertions, suppression ou substitutions de caractère nécessaires pour transformer l'une des chaînes en l'autre.

On a eu recours également à « La Similarité de Jaro-Winkler » car elle mesure également la similarité entre deux chaînes de caractères. Il s'agit d'une variante qui est principalement utilisée dans la détection de doublons. Le résultat est normalisé de façon à avoir une mesure entre 0 et 1, donc zéro représente l'absence de similarité et 1, l'égalité des chaînes comparées. Cette mesure est particulièrement adaptée au traitement de chaînes courtes comme des noms ou des mots de passe.

Tâche N°1 : Traitement de données sous Format Rdf

Le fichier contenant le programme est nommé: *TER2.0.py*
La première étape consiste au traitement de données *.rdf afin d’extraire des informations spécifiques décrite dans ce qui suit, nous avons donc décidé de travailler avec un script PYTHON, en se basant sur les acquis du semestre premier.

Ce programme, va parcourir le dossier courant contenant les fichiers .rdf à traiter, un par un et en boucle et en tirer tous les blocs de balises <map></map> contenant des relations de type <relation>.*exactmatch</relation> et dont les identifiants <entity1> et /ou <entity2> sont présents au minimum deux fois et dans deux blocs différents *(voir figure 2)*. Ces blocs de balises identifiés seront ensuite écrits sur des fichiers retour sous format rdf, situés dans un dossier * _DB, chaque nouveau fichier est marqué par *_DB (pour doublure).

```
<map>
  <Cell>
    <entity1 rdf:resource='http://www.mimo-db.eu/InstrumentsKeywords/3730'/>
    <entity2 rdf:resource='http://data.doremus.org/vocabulary/diabolo/mop/aerophone'/>
    <relation>http://www.w3.org/2004/02/skos/core#exactMatch</relation>
    <measure rdf:datatype='http://www.w3.org/2001/XMLSchema#float'>1.0</measure>
  </Cell>
</map>
<map>
  <Cell>
    <entity1 rdf:resource='http://www.mimo-db.eu/InstrumentsKeywords/4310'/>
    <entity2 rdf:resource='http://data.doremus.org/vocabulary/diabolo/mop/aerophone'/>
    <relation>http://www.w3.org/2004/02/skos/core#exactMatch</relation>
    <measure rdf:datatype='http://www.w3.org/2001/XMLSchema#float'>1.0</measure>
  </Cell>
</map>
```

Figure 2 : Exemple de structure rdf

Nous avons utilisé différents modules re, os, sys, basename, splitext..et avons utilisé des fichiers temporaires durant le processus, qui contiennent des informations de plus en plus spécifiques, pertinentes et nécessaire à l'écriture du fichier final. Toutes les étapes sont commentées et expliquées. *Voir Annexe Tâche 1 - TER.2.0*

Durant l'exécution du programme, nous avons un affichage des opérations effectuées en temps-réel (voir Figure 3) afin de suivre l'évolution des étapes de création des résultats.

```
nbendjoudi@x2go3:~/TER/Projet TER/valid 2018 copie/valid 2018 copie/otherFiles$ ./TER2.0.py

Dossier courant :/auto_home/nbendjoudi/TER/Projet TER/valid 2018 copie/valid 2018 copie/otherFiles

liste des fichiers à traiter :
mop_mimo_diabolo_valide.rdf
mop_mimo_itema3_valide.rdf


Liste des fichiers retour créés :

Traitement de :TER2.0.py
fichier d'une autre extension repéré

Traitement de :mop_mimo_diabolo_valide.rdf
mop_mimo_diabolo_valide_DB.rdf : créé dans /auto_home/nbendjoudi/TER/Projet TER/valid 2018 copie/valid 2018 copie/otherFiles_DB  <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< :D

lien présent : http://data.doremus.org/vocabulary/diabolo/mop/aerophone : 2  fois sauvegardé.
lien présent : http://data.doremus.org/vocabulary/diabolo/mop/aerophone : 2  fois sauvegardé.
Veuillez patienter SVP!
ecriture de mop_mimo_diabolo_valide_DB.rdf en cours

Liens entity1 présents chacun une seule fois

Liens entity2 présents au min 2 fois
```

Figure 3 : Affichage console durant l'exécution du programme

À la fin de l'exécution du programme, nous avons testé les fichiers retour sur http://yamplusplus.lirmm.fr/validator et avons obtenu des résultats corrects à 100% et sans bug. (*Voir figure 4*)

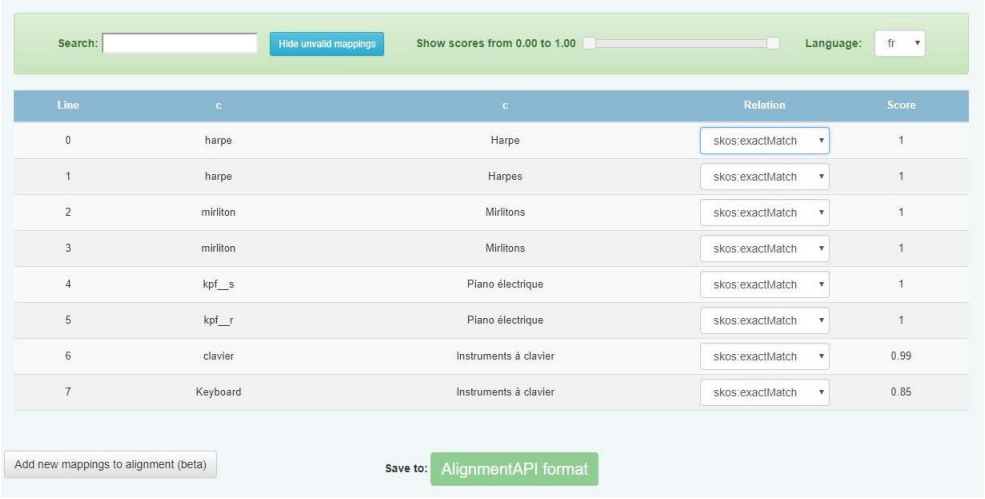

Figure 4 : Test des fichiers retour sur http://yamplusplus.lirmm.fr/validator

Tâche N°2 : Alignements des Termes Communs de la BNF et à Radio France

2.1 Préparation de la structure des fichiers BNF et RAMEAU

L'étape de Normalisation des données est indispensable pour gérer les caractères spéciaux et les mots vides :

Le fichier contenant le programme PYTHON dédié au traitements des données d'entrée (BNF-RAMEAU) est appelé: *prepaFichiers.py*

Nous devons avant toutes choses convertir le contenu du fichier RAMEAU vers le format utf-8 afin de pouvoir exploiter les données nécessaires à cette tâche, pour cela nous avons utilisé la ligne de code suivante, à exécuter directement dans le terminal et dans le dossier contenant les données.

```
iconv -f macintosh -t utf-8 RAMEAU-Groupes_ethniques-2.txt > RAMEAU_utf-8.txt
```

En premier lieu, nous allons importer les modules nécessaires pour effectuer la tâche, à savoir le module re, sys et os.

Ensuite nous allons extraire les données (en jaune) à aligner à l'aide des expressions régulières (regex), nos regex vont extraire dans un premier lieu le contenu en string situé après le $e(.*) et $m(.*)$e(.*) avec leur iD respectifs pour les bases BNF (*voir Figure 5*), et le contenu situé juste après le $a(.*) seulement et leurs iD du fichier RAMEAU *(voir Figure 6).*

```
485  43838434
486  143   $aTraditions$mAsie$eSufi
487  42398379
488  143   $aTraditions$mAfrique du Nord$mMaroc
489  42439908
490  143   $aTraditions$mAfrique du Nord$mMaroc
491  38562298
492  143   $aTraditions$mEurope$mRoumanie$eTziganes
493  43828169
494  143   $aTraditions$eJuifs
```

Figure 5 : Fichier de départ BNF

```
25  11934317   $aVikings
26  11934325   $aSaxons
27  11934371   $aPieds-noirs
28  11934398   $aKhazars
29  11934407   $aMaghrébins
30  11934421   $aSuédois
31  11934445   $aIndo-Européens
32  11934458   $aFrisons
33  11935298   $aKru$gpeuple d'Afrique
34  11935469   $aPeuls$gpeuple d'Afrique
```

Figure 6 : Fichier de départ RAMEAU

On va ensuite appliquer une série d'opérations de traitement des contenus récupérés, qui visent à transformer les majuscules en minuscules, supprimer les accents et les stopwords, un fichier french ajouté dans notre dossier de traitement contient la liste des stopwords en français, auquel nous avons ajouté quelques mots spécifiques à notre cas *(voir Figure 7)*

```
13  #liste des caractères accentués à désaccentuer
14  accent = ['é', 'è', 'ê', 'ë', 'à','á','É', 'ù', 'û', 'ú', 'ü','ç','č', 'ô','ó','ö', 'î','í','Î', 'ï'
15  sans_accent = ['e', 'e', 'e', 'e', 'a','a','E', 'u', 'u', 'u','u', 'c', 'c','o','o','o', 'i','i', 'I
16
17  #liste stop words
18  stopwords = open('french', 'r').read().split()
19  stopWordFr=re.compile('d\'|l\'|democratique|republique|peuples|!|origine|civilisation|\(|\)|_|,')
```

Figure 7 : Les caractères spéciaux à normaliser

Et écrire les résultats dans deux fichiers temporaires (newBNF_utf_8.txt et newRAMEAU_utf_8.txt), ces fichiers temporaires sont présentés comme sur la figure BNF *(voire Figure7)* et RAMEAU (voir Figure8). :

```
41347680~musulmans
38421203~turquie\tsiganes
38427666~amerique nord\bresil
42155205~yemen\juifs
43760455~sichuan\nuosu yi
40137206~russie\kalmoukes
```

Figure 7 : fichier temporaire newBNF_utf-8.txt

Structure BNF

(iD~ethnie) ou bien (iD~région\ethnie)

```
11933715~tzeltal
11933727~ukrainiens
11933789~vietnamiens
11934034~allemands
11934070~azteques
11934071~aymara
```

Figure 8 : Fichier temporaire RAMEAU

Structure RAMEAU

(iD~peuples)

2.2 Implémentation des mesures de similarité

L'étape d'implémentation des mesures de similarité vient après le traitement des fichiers BNF et RAMEAU.

Le fichier contenant le programme PYTHON d'alignement est nommé :
Alignement_BNF_RAMEAU_jaro.py

Dans le contexte de notre projet, nous nous sommes orientés vers la recherche des différentes méthodes à implémenter afin de réussir à obtenir un résultat d'alignement efficace. Nous avons choisi de réaliser une combinaison entre deux mesures de similarités « *Levenshtein » et « Jaro-winkler* » qui traitent différemment l'alignement de deux chaînes de caractères, qui avec des seuils minimum de similarité prédéfinis, nous permettent d'obtenir un résultat concret et pertinent (nous avons testé plusieurs seuils avant de définir les seuils actuels, qui nous offre l'alignement souhaité).

2.2.1 Similarité Normalisée Jaro-Winkler

La distance de Jaro-Winkler mesure la similarité entre deux chaînes de caractères, on va l'utiliser afin de repérer les similarités exacte et dépassant un seuil prédéfini par nous-même (après plusieurs tests).

Le résultat est normalisé de façon à avoir une mesure entre 0 et 1, dont le 0 représente l'absence de similarité et 1, l'égalité des chaînes comparées.

2.2.1.1 Distance de Jaro

La distance de Jaro entre chaînes mo11 et mot2 est définie par : *result*

```
97      result= ((matches / mot1_len) + (matches / mot2_len) + ((matches - transpositions/2) / matches)) / 3
```

Ou:

- moti_len est la longueur de la chaîne de caractères Si
- matches est le nombre de caractères correspondants
- transpositions est le nombre de transpositions

Deux caractères identiques de mot1 et de mot2 sont considérés comme *correspondants* si leur éloignement (i.e. la différence entre leurs positions dans leurs chaînes respectives) ne dépasse pas :

```
60      match_distance = (max(mot1_len, mot2_len) // 2) - 1
```

Le nombre de transpositions est obtenu en comparant le i-ème caractère *correspondant* de mot1 avec le i-ème caractère *correspondant* de mot2 . Le nombre de fois où ces caractères sont différents, divisé par deux, donne le nombre de *transpositions*.

2.2.2.2 Distance de Jaro-Winkler

La distance de Jaro entre chaînes mot1 et mot2 est définie par : *resultW*

```
107     resultW = result + (l * SCALING_FACTOR * (1 - result));
108
109     return (round(resultW,3))
```

Ou:

- l est la longueur du préfixe commun (maximum 4 caractères)
- SCALING_FACTOR est coefficient qui permet de favoriser les chaînes avec un préfixe commun, Winkler propose pour valeur SCALING_FACTOR = 0.1

2.2.2 Similarité Normalisée Levenshtein

Définition[1]: Prenons comme exemple les deux chaînes de caractère Cinéphile (S1) et Cynophile (S2), On appelle distance de Levenshtein entre deux mots (S1) et (S2) le coût minimal pour transformer (S1) en (S2) en effectuant les seules opérations élémentaires suivantes :

- substitution d'un caractère de (S1) par un caractère différent de (S2) ;
- insertion (ou ajout) dans (S1) d'un caractère de (S2) ;
- suppression (ou effacement) d'un caractère de (S1).

La distance Levenshtein Normalisée est donnée comme suit:

$$\delta_{LevN} = \frac{\delta_{Lev}(s_1, s_2)}{max(|s_1|, |s_2|)},$$

avec |S1|=len(S1)

On associe à chacune de ces opérations un coût. Généralement, le coût est égal à 1 pour les trois opérations.

Et on retourne un résultat de similarité et non de distance de telle sorte que le résultat entre 0 et 1 *(voir Figure 9)* (avec 0 pas de similarité et 1 similarité parfaite)

(s1=mot1 & s2=mot2 dans notre code) $\sigma_{LevN}(s_1, s_2) = 1 - \delta_{LevN}(s_1, s_2).$

```
114 #////////////////////////////////////////////////////////////////////////////
115 #////////////////// MESURE LEVENSTEIN ///////////////////////////////////////
116 #////////////////////////////////////////////////////////////////////////////
117
118 def levenshteinN(mot1,mot2):
119     ligne_i = [ k for k in range(len(mot1)+1) ]
120     for i in range(1, len(mot2) + 1):
121         ligne_prec = ligne_i
122         ligne_i = [i]*(len(mot1)+1)
123         for k in range(1,len(ligne_i)):
124             cout = int(mot1[k-1] != mot2[i-1])
125             ligne_i[k] = min(ligne_i[k-1] + 1, ligne_prec[k] + 1, ligne_prec[k-1] + cout)
126     result = 1-(ligne_i[len(mot1)]/(max(len(mot1),len(mot2))))
127     return(round(result,3))
```

Figure 9 : Mesure de similarité Levenshtein Normalisée

On arrondit majoritairement les résultats obtenus suite aux mesures LevenshteinN et JaroW à 3 chiffres après la virgule pour faciliter la lecture.

2.3 Alignement des termes BNF vers Radio France

Après avoir effectué la préparation de la structure des fichiers BNF et RAMEAU à aligner (voir *4.1 Préparation de la structure des fichiers BNF et RAMEAU*). Nous allons maintenant récupérer les termes à aligner grâce à une boucle qui parcours les fichiers newBNF et newRAMEAU avec la méthode **.readlines() et ensuite mesurer en cartésien la similarité entre les termes, dans code en annexe explique dans le détail les étapes et la démarche suivie pour les mesures, nous citons les seuils suivant :*

```
if levenshteinN(ethnie,mot2) >= 0.9 and jaro(ethnie, mot2) > 0.9 :  #<<<<<<< ici vous mettez le seuil de similarité Jaro min
    if iD1 not in m:
        for cle,valeur in BNF.items():
            if cle==iD1:
                ethnieMatch.write(cle+' '+valeur+'\n')#ecriture ligne BNF
        for cle,valeur in RAMEAU.items():
            if cle==iD2:
                ethnieMatch.write(cle+' '+valeur+'\n')#ecriture ligne Rameau
        ethnieMatch.write('JaroW:'+str(jaro(ethnie,mot2))+' Levenstein: '+str(levenshteinN(ethnie,mot2))+'\n\n')#ecriture mesures
        m.append(iD1)
```

Figure 10 : identification "ethnieMatch"

- *Seuil 1 : Lev = 0.7 & Jaro = 0.9 : Il s'agit de similarité minimale pour un ethnieMatch $e: levenshteinN(**mot1,mot2**) >= 0.7 and jaro(mot1, mot2) > 0.9 : (voir figure 10)*

  On identifie chaque terme ethnique de la BNF et Rameau, qui sont présents **sans aucune information géographique**, on passe après à l'alignement de ces ethnies.
  A chaque fois que les mesures sont respectées pendant l'alignement de chaque terme, on écrit ces deux derniers avec le calcul des deux mesures de similarités sur leurs chaînes de caractères, et on écrits chaque terme et son identifiant dans un nouveau fichier « ethnieMatch » : (*figure 11*)

```
177  42155205    $aTraditions$mAsie$mYémen$eJuifs
178  11932167    $aJuifs
179  JaroW:1.0 Levenstein: 1.0
180
181  43795806    $aTraditions$mAsie$mChine$mSichuan$eQiang
182  12460130    $aQiang$gpeuple de Chine
183  JaroW:1.0 Levenstein: 1.0
184
185  42695285    $aTraditions$mAfrique$mAfrique du Sud$eZoulous
186  11948763    $aZoulous$gpeuple d'Afrique
187  JaroW:1.0 Levenstein: 1.0
```

Figure 11 : Fichier "ethnieMatch"

- Seuil 2 : *Lev = 0.6 & Jaro = 0.9 :* On réitère la dernière opération avec ces seuils là et on obtient des alignement *"ambigus"* qui serviront à un seconde opération d'alignement plus tard, la différence ici consiste à mesurer la distance entre deux termes constitués d'une listes de mots avec *Tokenisation* (mot1S) et (mot2S) avec **.split() (voir Figure 12)* afin d'identifier les alignements avec au moins un terme en commun (Juif > Juifs Croates)
  levenshteinN(**mot1S,mot2S**) >= 0.5 and jaro(mot1, mot2) > 0.7:

```
3425  43828228    $aTraditions$eJuifs
3426  16934663    $aJuifs latino-américains
3427  JaroW:0.843 Levenstein: 0.217
3428
3429  43828228    $aTraditions$eJuifs
3430  16934866    $aJuifs sud-africains
3431  JaroW:0.853 Levenstein: 0.263
3432
3433  41347724    $aTraditions$eMusulmans
3434  13623718    $aMusulmans indiens$gde l'Inde
3435  JaroW:0.906 Levenstein: 0.529
```

Figure 12 : Fichier "ambiguMatch"

- Seuil 3 : *Il s'agit de similarité minimale pour un ethnieMatch me: levenshteinN(**ethnie**,mot2) >= 0.9 and jaro(**ethnie**, mot2) > 0.9 :*
  *levenshteinN(**region**,mot2) >= 0.6 and jaro(**region**, mot2) > 0.9 :*

On identifie maintenant et on traite le contre cas, qui est une information ethnique qui précède avec une information sur une zone géographique. Par exemple : ($mYemen$eJuifs)
On récupère la région et l'ethnie (Figureen utilisant une expression régulière python, ainsi on récupère dans chaque ligne du nouveau fichier RAMEAU l'identifiant avec son terme. Ensuite, on aligne l'ethnie avec le terme de Rameau, ainsi que la région géographique avec le terme de RAMEAU, et on stockera les résultats en ordre dans les fichiers (**ethnieMatch**) et le nouveau fichier (**regionMatch** *voir Figure 13*).

```
17  40941897    $aTraditions$mAsie centrale$mRussie$eTadjikes
18  11942615    $aRusses
19  JaroW:0.528 Levenstein: 0.25
20
21  42664693    $aTraditions$mEurope$mEspagne$eGitans
22  11936374    $aEspagnols
23  JaroW:0.611 Levenstein: 0.333
24
25  38625368    $aTraditions$mAmérique du Sud$mBrésil$eBahia
26  11976818    $aBrésiliens
27  JaroW:0.58 Levenstein: 0.2
```

Figure 13 : Fichier "regionMatch"

Termes BNF non alignés
A chaque fois que nous avons un alignement correcte entre la BNF et RAMEAU, on ajoute l'ID des lignes alignées de la BNF dans une liste, afin de récupérer à la fin les différents ID avec leurs lignes qui n'étaient pas alignées et on stocke tout ça dans un fichier (bnfNotMatch).
Le script complet (voir *annex TER2.0.py*)

Tâche N°3 : Associer des Concerts de la base de données Du Philharmonie de Paris

Nous disposons pour cette tâche de deux dossiers contenant respectivement des fichiers de données euterpe et pp concerts,

Notre mission dans cette tâche consiste à aligner deux entités. Or cette fois-ci nous avons affaire à une seule bibliothèque celle de la Philharmonie de Paris.

Notre tâche consiste à mettre en correspondance des listes de concerts et d'évènements organisés par cette institution Française. Nous avons deux types d'éléments à faire « Matcher », une liste d'événements prévus (appelées EUTERPE) avec une liste d'évènements ayant eu lieu (appelées PP). Nous comparerons les titres et dates des concerts.

3.1 Parcourir le dossier euterpe

Au cours de cette étape nous allons récupérer les titres des concerts prévus et leurs dates respectives, qu'on peut trouver grâce à la classe:M26 qu'on doit récupérer dans le script de chaque fichier, le bloc contenant cette mention contiendra les informations :

- ce bloc aura comme début (M26) et comme fin (>.) (*voir Figure 14)*
- titre: rattaché directement à **M26** par la propriété **ecrm:P102_has_title**

- date: rattachée à **M26** par la propriété **mus:U8_foresees_time_span** dont la valeur est un identifiant **décrit plus loin dans le même fichier**.

```
<http://data.doremus.org/performance/e8ae72ab-9474-3809-95ba-7b999ddb190e>
        a                               mus:M26_Foreseen_Performance , prov:Entity ;
        rdfs:comment                    "Le plus jeune des fils de Bach, Johann Christian, connaissait à Londres un succès certain
        lorsqu'il composa son opéra Zanaïda en 1763. À propos de cette partition, ressuscitée pour la première fois depuis sa
        création, Charles Burney, célèbre critique musical de l'époque, déclarait : « Bach a donné à Amicis (la soprano Anna Lucia De
        Amicis, qui chantait le rôle-titre) les airs les plus magistraux qu'un homme puisse écrire. »\n L'histoire se déroule à la
        cour de Perse. La princesse turque Zanaïda, fille de Soliman, doit s'y unir au sophi persan Tamasse, qui lui préfère toutefois
        Osira. Craignant les représailles de Soliman, il retient Zanaïda prisonnière et l'accuse d'un amour illicite. Elle échappera
        néanmoins à son exécution et pardonnera Tamasse."@fr ;
        rdfs:label                      "Zanaïda" ;
        mus:U67_has_subtitle            "Opera Fuoco" ;
        mus:U77_foresees_performing_plan
                <http://data.doremus.org/expression/a7b4cd5d-dcfb-3525-ba99-5455a743bc7a> ;
        mus:U7_foresees_place_at        <http://data.doremus.org/place/a59b933e-7715-398c-a075-ed4191c62bba> , <http://data.doremus.org/
        place/c4f1ead7-d8a3-3f3d-8d82-6d6923313966> ;
        mus:U8_foresees_time_span       <http://data.doremus.org/performance/e8ae72ab-9474-3809-95ba-7b999ddb190e/interval/0> ;
        ecrm:P102_has_title             "Zanaïda" ;
        ecrm:P2_has_type                "concert"@fr ;
        ecrm:P3_has_note                "Le plus jeune des fils de Bach, Johann Christian, connaissait à Londres un succès certain
        lorsqu'il composa son opéra Zanaïda en 1763. À propos de cette partition, ressuscitée pour la première fois depuis sa
        création, Charles Burney, célèbre critique musical de l'époque, déclarait : « Bach a donné à Amicis (la soprano Anna Lucia De
        Amicis, qui chantait le rôle-titre) les airs les plus magistraux qu'un homme puisse écrire. »\n L'histoire se déroule à la
        cour de Perse. La princesse turque Zanaïda, fille de Soliman, doit s'y unir au sophi persan Tamasse, qui lui préfère toutefois
        Osira. Craignant les représailles de Soliman, il retient Zanaïda prisonnière et l'accuse d'un amour illicite. Elle échappera
        néanmoins à son exécution et pardonnera Tamasse."@fr ;
        ecrm:P69_has_association_with   <http://data.doremus.org/performance/e8ae72ab-9474-3809-95ba-7b999ddb190e/1> ;
        dcterms:identifier              "11435" ;
        prov:wasAttributedTo            <http://data.doremus.org/organization/DOREMUS> ;
        prov:wasDerivedFrom             <http://data.doremus.org/source/euterpe/11435> ;
        prov:wasGeneratedBy             <http://data.doremus.org/activity/45dcd571-340d-3e3f-a330-90daca761715> ;
        foaf:isPrimaryTopicOf           <http://www.citedelamusique.fr/francais/activite/concert/11435> .
```

Figure 14 : bloc contenant la classe M26, le titre et l'URI de la date

Dans un second temps, Une fois l'URI récupéré, On relis encore le fichier courant ligne par ligne (f.readlines()) mais cette fois pour récupérer l'URI de la date qui sera un lien indice pour récupérer la date de l'événement, on vise avec une *regex* ce qui vient après et on définit un nouveau bloc dans lequel on peut tirer maintenant la date au format conventionnel AAAA-MM-JJ (*voir figure 15*)

```
<http://data.doremus.org/performance/e8ae72ab-9474-3809-95ba-7b999ddb190e/interval/0>
        a                   ecrm:E52_Time-Span , time:Interval ;
        rdfs:label          "2011-09-15" ;
        time:hasBeginning   [ a                 time:Instant ;
                              time:inXSDDate    "2011-09-15T20:00"^^xsd:dateTime
```

Figure 15 : bloc contenant le titre et l'URI de la date

Après la récupération des titres et des dates, on stocke tout ça dans un nouveau fichier (**euterpe.txt**) qui va être structuré de plusieurs lignes de la manière (NomFICHIER TITRE_traité date :DATE TITRE_original) prêt à être aligné.

3.2 Parcourir le dossier PP

Au cours de cette étape nous allons récupérer les titres des concerts prévus et leurs dates respectives, qu'on peut trouver grâce à la classe:M26 qu'on doit récupérer dans le script de chaque fichier, le bloc contenant cette mention contiendra les informations :

- ce bloc aura comme début (**F31**) et comme fin (>.) (*voir Figure 16*)
- titre: rattaché directement à **F31** par la propriété **ecrm:P102_has_title**
- date: rattachée à **F31** par la propriété **ecrm:P4_has_time-span** dont la valeur est un identifiant **décrit plus loin dans le même fichier**.

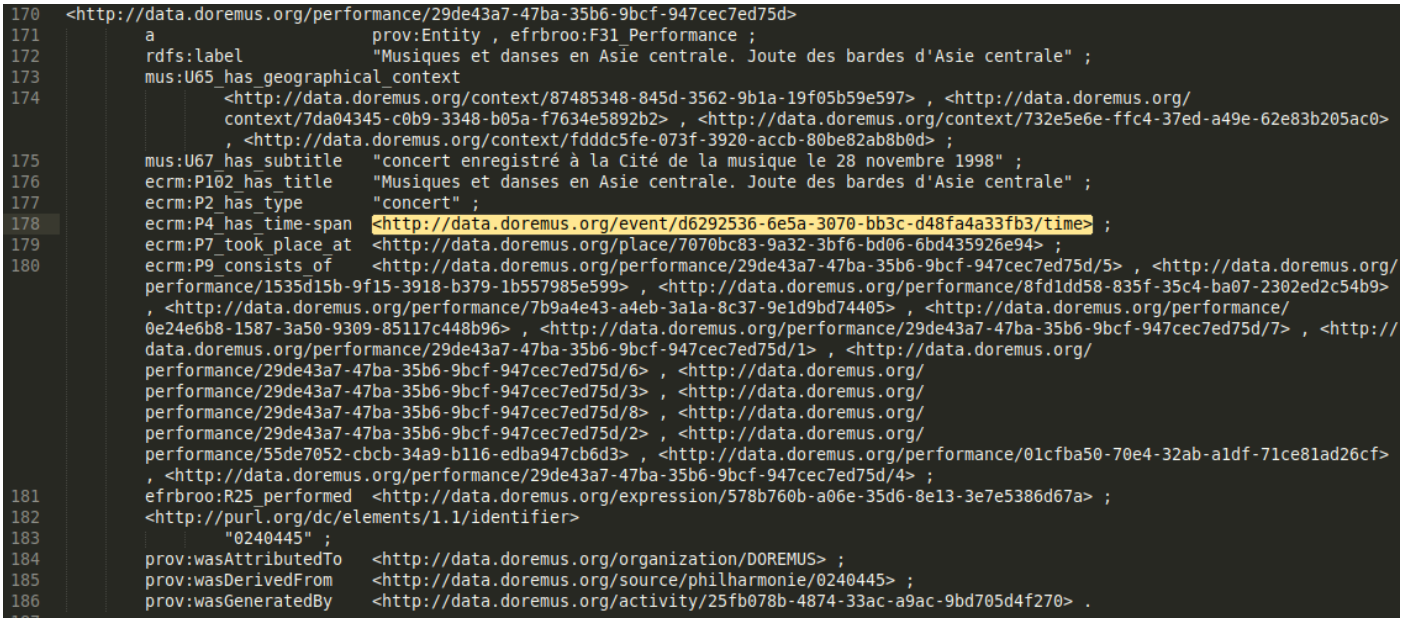

Figure 16 : bloc contenant la classe F31, le titre et l'URI de la date

On exécute le même processus de la seconde étape du parcours du dossier EUTERPE sur les fichier PP, on obtient du coup et avec les mêmes procédures les dates associées à chaque concert effectué.

Après la récupération des titres et des dates, on stocke tout ça dans un nouveau fichier (**pp.txt**) qui va être structuré de plusieurs lignes de la manière (NomFICHIER TITRE_traité date : DATE TITRE_original) prêt à être aligné.

3.3 Alignement des évènements

On a quatre figures d'alignement :

- Événements de même titre et de même date. (**EMTD**)
- Événements ayant un même date. (**EMD**)
- Événements avec titres ressemblant mais pas similaires et avec une même date. (**EMTrD**)
- Evènements seulement avec des titres qui se ressemblent. (**EMTr**)

On génère un fichiers pour chaque type d'alignement (EMTD, EMD, EMTrD, EMT) et on les ouvres en lecture et écriture.

Ensuite, on ouvre les deux fichiers à traiter (**euterpe.txt, pp.txt**) en lecture ligne par ligne (readlines()), on procède ainsi à la récupération des différentes parties « NomFICHIER , TITRE_traité , date : DATE , TITRE_original » des différentes lignes de chaque fichier et on stocke ça dans des variables .

L'alignement est prêt, on commence à définir nos seuils de *Levenshtein et Jaro* afin d'avoir nos différents cas comme dans la Tâche 2 *(voir Figure 17)*:

N.B : On aligne les titres traités (Normalisés :sans accent, sans espace, sans stopwords...) mais on affiche les titres originaux :

- Seuil 1 : *if levenshteinN(date1,date2) ==1.0 and jaro(date1,date2) == 1.0:* **dates exactes**
  if levenshteinN(titre1,titre2) >= 0.9 and jaro(titre1,titre2) > 0.9: **titres exactes**

Il s'agit de similarité minimale pour un un alignement exacte entre les titres et les dates respectives.
Nous n'avons pas choisi la valeur 1 pour la présence de légères différences de caractères
Ces résultats seront inscrits dans le fichier **EMTD.txt**

- Seuil 2 : *if levenshteinN(date1,date2) ==1.0 and jaro(date1,date2) == 1.0:* **dates exactes**
  elif levenshteinN(titre1,titre2) >= 0.5 and jaro(titre1,titre2) > 0.7: **titres légèrement différentes**

  Il s'agit de similarité minimale pour un un alignement exacte entre les dates. mais avec des titres légèrement différents.
  Ces résultats seront inscrits dans le fichier **EMTrD.txt**

- Seuil 3 : if levenshteinN(titre1,titre2) >= 0.5 and jaro(titre1,titre2) > 0.7: **titres différents**
  if levenshteinN(date1,date2) ==1.0 and jaro(date1,date2) == 1.0: **dates exactes**
  Il s'agit de similarité minimale pour un un alignement exacte entre les dates. mais avec des titres légèrement différents.
  Ces résultats seront inscrits dans le fichier **EMD.txt**

- Seuil 4 : *if not levenshteinN(date1,date2) ==1.0 and jaro(date1,date2) == 1.0:* **dates différentes** *if levenshteinN(titre1,titre2) >= 0.9 and jaro(titre1,titre2) > 0.9:* **titres exactes**

Il s'agit de similarité minimale pour un un alignement exacte entre les titres. mais avec des dates différentes.
Ces résultats seront inscrits dans le fichier **EMT.txt**

```
348                 if levenshteinN(date1,date2) ==1.0 and jaro(date1,date2) == 1.0:#dates exactes
349                     if levenshteinN(titre1,titre2) >= 0.9 and jaro(titre1,titre2) > 0.9:  #titres exactes
350
351                         #ecriture des alignements répondants aux conditions
352                         EMTD.write(titrePrime1+':'+date1+'\n'+titrePrime2+':'+date2+'\n'+'JaroW:'+str(jaro
353                         if titre1 not in s:
354                             s.append(titre1)
355                     elif levenshteinN(titre1,titre2) >= 0.5 and jaro(titre1,titre2) > 0.7: #titres légèrem
356                         EMTrD.write(titrePrime1+':'+date1+'\n'+titrePrime2+':'+date2+'\n'+'JaroW:'+str(jar
357                         if titre1 not in s:
358                             s.append(titre1)
359                     else: #titres complétement différents
360                         EMD.write(titrePrime1+':'+date1+'\n'+titrePrime2+':'+date2+'\n'+'JaroW:'+str(jaro(
361
362                 elif levenshteinN(titre1,titre2) >= 0.9 and jaro(titre1,titre2) > 0.9:  #titres exactes
363                         #ecriture des alignements répondants aux conditions
364                         EMT.write(titrePrime1+':'+date1+'\n'+titrePrime2+':'+date2+'\n'+'JaroW:'+str(jaro(
365                         if titre1 not in s:
366                             s.append(titre1)
367
```

Figure 17 : opérations de mesures.txt.

On procède maintenant au troisième cas qui est la même date mais pas exactement le même titre, pour cela il faut garder un seuil de similarité égale à 1 pour les dates et diminuer le seuil de similarité pour les titres. (Figure 10)

Analyse et Conduite du projet

Diagramme de Gantt

Sitographie

- Les expressions régulières en python [en ligne]. Consulté le 04 Décembre 2018. Disponible sur : http://apprendre-python.com/page-expressions-regulieres-regular-python
- Regular expression opérations, syntax [en ligne]. Consulté le 04 Décembre 2018. Disponible sur : https://docs.python.org/2/library/re.html
- Gael Pasgrimaud, Remplacement de chaîne en python [en ligne]. Consulté le 12 Décembre 2018. Disponible sur : http://www.gawel.org/python-re-sub/
- Konstantin Todorov, Ontology Matching and Data Linking, Février 2017. Consulté le 26 Janvier 2018 : P.22-50
- Tokenizer for Python source [en ligne]. Consulté le 26 Janvier 2018. Disponible sur : https://docs.python.org/2/library/tokenize.html
- Jonathan Mugan, How can i tokenize a sentence with python [en ligne], publié le 18 Avril 2017. Consulté le 7 Février 2018. Disponible sur : https://www.oreilly.com/learning/how-can-i-tokenize-a-sentence-with-python
- Jaro distance [en ligne]. Consulté le 03 Mars 2018. Disponible sur : https://rosettacode.org/wiki/Jaro_distance
- Frank Hofmaan, Levenshtein Distance and Text Similarity in Python [en ligne], publié le 17 Janvier 2018. Consulté le 06 Mars 2018. Disponible sur : http://stackabuse.com/levenshtein-distance-and-text-similarity-in-python/
- Algorithm Implementation/Strings/Levenshtein Distance [en ligne], édité le 08 Mars 2018. Consulté le 15 Mars 2018. Disponible sur : https://en.wikibooks.org/wiki/Algorithm_Implementation/Strings/Levenshtein_distance
- Find the Jaro Winkler Distance which indicates the similarity score between two Strings [en ligne], édité le 05 Mars 2017. Consulté le 08 Mars 2018. Disponible sur : https://github.com/nap/jaro-winkler-distance
- Salvador Dali, Edit Distance in Python [en ligne], édité le 27 Avril 2016. Consulté le 10 Mars 2018. Disponible sur : https://stackoverflow.com/questions/2460177/edit-distance-in-python

7 ANNEXES :

TER2.0.py :

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


import re, os, sys, cgi,os.path
from os.path import basename, splitext

#////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#/////////////////////////////////////////////////////// TER 2.0 ////////////////////////////////////////////////////////
#////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////


#////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#///////////////////// PROGRAMME QUI GENERE UN  FICHIER  RETOUR.RDF  CONTENANT  QUE  LES RELATIONS //////////////////////
#///////////////////// VALIDES  DONT  L'ARGUMENT ENTITY1  OU  ENTITY2  EST PRESENT AU MINIMUM DEUX //////////////////////
#///////////////////// FOIS, EXPORTATION AUTOMATIQUE DES FICHIERS RETOUR AVEC LE NOM D'ORIGINE+_DB //////////////////////
#////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#//////////// UTILISATION CODE : VEUILLEZ COPIER COLLER LE CODE TER.PY DANS LE DOSSIER CONTENANT LES FCHIERS .RDF //////
#////////////////// ET L'EXECUTER A L'AIDE DE L'INVITÉ DE COMMANDE AVEC DROITS D'ECRITURE ET DE MODIFICATION ///////////
#////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////


#///////////////////////Lecture et création des fichiers (fichier de base + _DB)://///////////////////////////////////////

rep=os.getcwd() #récupérer le dossier courant

print('\n'+'Dossier courant :'+rep+'\n') #Console :suivi
newRep=rep.split('\\')[-1]+'_DB'

if not os.path.exists(newRep):    #Vérification des fichiers et dossiers et création du dossier contenant les retour_DB
    os.mkdir(newRep)

liFichiers = os.listdir(rep) #Listing du contenu du répertoire courant (ONLY .RDF FILES) que le code va traiter
```

```python
print('liste des fichiers à traiter :')
for fichier in liFichiers:
    if fichier.endswith('.rdf'):
        print(fichier)
print('\n')


#explorer tout le contenu du dossier courant et appliquer le traitement pour chaque fichier.rdf dedans
print('Liste des fichiers retour créés :')
listeNomsFichiers={} #Utilisation d'un dictionnaire
for fichier in liFichiers:
    print('\n'+'Traitement de :'+fichier)

    fDispo = re.search("(.*)\.rdf", fichier)
    if fDispo:
        filepath=(fDispo.group(1)+'.rdf')
        if filepath not in listeNomsFichiers:
            listeNomsFichiers[filepath] = fichier

            fichierRdf = open(filepath) #chemin
            lecture = fichierRdf.read() #lecture fichier 1
            filename = splitext(basename(filepath))[0]
            fichieRetour = open(newRep+'/'+filename+'_DB.rdf','x') #création fichier de retour_DB.rdf


            print(filename+'_DB.rdf : créé dans '+newRep+'  <<<<<<<<<<<<<<<<<<<<<<<<<<<<<< :D'+'\n') #Console: Validation création fichier

            newF = open('newF','x') #nouveau fichier temporaire 1
```

```
#//////////////////////  Rajouter le début du code XML   ////////////////////////////////////////////////////////////////////////////

        debut = re.compile(r".*(\<\?xml.*\s+\<rdf.*\s+.*\s+.*\s+.*\s+.*\s+.*\s+.*\s+\<Alignment.*\s+\<xml.*\s+\<level.*\s+\<type.*\s+.*\
        # début code XML

        res1 = re.search(debut,lecture)

        if res1:
            a = res1.group(1)

            fichieRetour.write(a)   #ecriture du début sur le fichier retour
        else:
            print('Pas de correspondance lors de la récupération de l\'entête, verifier le code source SVP!')

#//////////////////////  Recupérer les relations valides   ////////////////////////////////////////////////////////////////////////////


        corps = re.compile(r".*(\s+\<map\>\s+\<Cell\>\s+.*\s+.*\s+.*exactMatch.*\s+.*\s+\</Cell\>\s+\</map\>).*") #les relations valides

        res = corps.findall(lecture)

        if res:
            for ligne in res:
                newF.write(ligne) #ecriture des lignes valides dans un fichier tomporaire (newF)
        else:
            print('Pas de relations valides trouvées')



        newF.close() #femeture fichier temporaire 1
```

```
//////////////////////  Rajouter aux liens présents >1 fois l'indicateur CIBLA   ////////////////////////////////////////////////////////////////////////////


        newF = open('newF','r') # exploiter temporaire 1 en lecture

        lecture2 = newF.read() #lecture fichier 1

        semiFinals = open('newF2','x') # création fichier tomporaire 2

        res2 = re.findall(".*rdf:resource='(.*)'/\>",lecture2) #identifier les liens

        m = list() # On crée une liste pour éviter d'afficher les mêmes lignes plusieurs fois (distinct)

        if res2:
            for line in res2:
                d = res2.count(line)
                if d > 1: # ce lien est présent au moins 2 fois
                    avecCibla = lecture2.replace(line,line+'CIBLA') #identifier les liens doublons avec le mot CIBLA
                    semiFinals.write(avecCibla) # ecriture des lignes avec modifications sur le fichier temporaire 2
                    if line not in m:
                        print('lien présent :',line,':',d,' fois sauvegardé.') #Console : suivi
                        m.append(line) # si la ligne n'existe pas dans m() alors l'ajouter.



        semiFinals.close() #femeture fichier temporaire 2




        print('Veuillez patienter SVP!'+'\n'+'ecriture de '+filename+'_DB.rdf en cours'+'\n')
```

```
//////////////  Rajouter aux liens présents >1 fois l'indicateur <<CIBLA>>   ////////////////////////////////////////


semiFinals = open('newF2','r') # exploiter temporaire 2 en lecture

lecture3 = semiFinals.read() #lecture fichier 2

semiFinals2 = open('newF3','x') # création fichier temporaire 3

#identifier les blocs <map>....</map> contenant 'CIBLA' >>>> ça nous permet de ne repérer que les blocs avec des liens pré
res3 = re.findall(".*(\s+\<map\>\s+\<Cell\>\s+.*CIBLA'/\>\s+.*\s+.*\s+.*\s+\</Cell\>\s+\</map\>).*", lecture3) # resultat
res4 = re.findall(".*(\s+\<map\>\s+\<Cell\>\s+.*\s+.*CIBLA'/\>\s+.*\s+.*\s+\</Cell\>\s+\</map\>).*", lecture3) # resultat

if res3 :
    print('Liens entity1 présents au min 2 fois'+'\n')
    for ligne in res3:
        semiFinals2.write(ligne) # ecriture des lignes avec (entity1 > 2) sur le fichier temporaire 3
else:
    print('Liens entity1 présents chacun une seule fois'+'\n')

if res4 :
    print('Liens entity2 présents au min 2 fois'+'\n')
    for ligne in res4:
        semiFinals2.write(ligne) # ecriture des lignes avec (entity2 > 2) sur le fichier temporaire 3
else:
    print('Liens entity2 présents chacun une seule fois'+'\n')


semiFinals2.close() #femeture fichier temporaire 3

print('\n')
```

```
//////////////////////  Suppression de l'indicateur CIBLA  ////////////////////////////////////////////////////////////


    semiFinals2 = open('newF3','r') # exploiter temporaire 3 en lecture

    lignes = semiFinals2.readlines() #lecture fichier 3

    finals = open('newF4.rdf','x') # création fichier temporaire 4

    for ligne in lignes:
        ligneFinale = ligne.replace('CIBLA','') # parcourir les lignes du fichier temporaire 3 et remplacer CIBLA par un vide''
        finals.write(ligneFinale) # ecriture des lignes propres sur le fichier temporaire 4

    semiFinals2.close() #femeture fichier temporaire 3
    finals.close() #femeture fichier temporaire 4
```

```
//////////////////// Suppression des blocs doublons en respectant l'ordre des lignes du fichier ////////////////////////////

      s = list() #création d'une liste s vide

      finals = open('newF4.rdf','r') # exploiter temporaire 4 en lecture

      corps = re.compile(r".*(\s+\<map\>\s+\<Cell\>\s+.*\s+.*\s+.*exactMatch.*\s+.*\s+\</Cell\>\s+\</map\>).*")
      fichierF = finals.read()

      res = corps.findall(fichierF)
      if res:
          for line in res:
              if line not in s:
                  s.append(line) # si la ligne existe dans s() alors l'ignorer.
                  fichieRetour.write(line) # ecriture des lignes finales sur le fichier de retour_DB.rdf

      finals.close() #femeture fichier temporaire 4
```

```
////////////////////  Rajouter le bas du code XML dans le fichier final et suppression des fichiers tmp ////

        res3 = re.search("(\s+</Alignment>\s+</rdf:RDF>)", lecture)

        if res3:
            c = res3.group(1)
            fichieRetour.write(c) #ecriture des lignes du bas XML



        os.remove("newF")  #suppression fichier temporaire 1
        os.remove("newF2") #suppression fichier temporaire 2
        os.remove("newF3") #suppression fichier temporaire 3
        os.remove("newF4.rdf") #suppression fichier temporaire 4
        fichieRetour.close() #femeture fichier retour

////////////////////  Continuer à éxecuter le programme tant qu'on a tjrs des fichiers non traités  ///////
    else:
        listeNomsFichiers[filepath] += fichier
else:
    print('fichier d\'une autre extension repéré, aucune action n\'est exécutée')
```

prepaFichiers.py :

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
from __future__ import division

import re, sys, os, math


#////////////////// création de deux nouveaux fichiers en écrtiture
nFichierBNF = open('newBNF_utf-8.txt','x+')
nFichierRM = open('newRAMEAU_utf-8.txt','x+')


#liste des caractères accentués a désaccentuer
accent = ['é', 'è', 'ê', 'ë', 'à','á','É', 'ù', 'ú', 'ú', 'ü','ç','č', 'ô','ó','ö', 'î','í','Î', 'ï', 'â','ã','ä','ţ','(\'','\', \'','\', "','\')','")','$e']
sans_accent = ['e', 'e', 'e', 'e', 'a','a','E', 'u', 'u', 'u','u', 'c', 'c','o','o','o', 'i','i', 'I', 'i','a', 'a', 'a','t','','~','~','','','\\']

#liste stop words
stopwords = open('french', 'r').read().split()
stopWordFr=re.compile('d\'|l\'|democratique|republique|peuples|!|origine|civilisation|\(|\)|_|,')
```

```
#///////////////// on parcour le fichier BNF , on récupère dans toutes les lignes que la partie après le dernier $
#/////////////// on écrit ces lignes là dans le nouveau fichier > BNF
#////////////////////// on ouvre le fichier BNF , lecture ligne

file1 =open('BNF_traditions_field.txt','r')
read1= file1.read()

#////////////////// on récupère tous les $e solitaires

#////////////////////////////////////////////////////////////////////////////////
#////////////////////////////        $e        ////////////////////////////////
#////////////////////////////////////////////////////////////////////////////////

res1b = re.findall(r"(.*)\n\d{3}\s+\$a[A-Z][a-z]+\$e(.*)",read1)
if res1b:
    for ligne in res1b:
        sentence=str(ligne)

        #Transformer les majuscules en minuscules
        sentence=sentence.lower()

        #Supprimer les accents
        for i in range(len(accent)):
            sentence = sentence.replace(accent[i], sans_accent[i])

        sentence = sentence.split()

        filteredtext = [t for t in sentence if t.lower() not in stopwords]

        s = " ";
        # on recole la phrase
        sentence = s.join( filteredtext )

        #Supprimer les stop-words manuellement et sans appel à ntlk
        sentence=re.sub(stopWordFr,'',sentence)

        #Ecriture du nouveau fichier à alligner RAMEAU
        nFichierBNF.write(sentence+'\n')
```

```
#////////////////////// On récupère les $m.*$e (si $e ne suffit pas à determiner l'alignement alors on utilise $m pour nous
#raprocher du résultat

res1 = re.findall(r"(.*)\n\d{3}\s+\$a.*\$m(.*\$e.*)",read1)
if res1:
    for ligne in res1:
        sentence=str(ligne)

        #Transformer les majuscules en minuscules
        sentence=sentence.lower()

        #Supprimer les accents
        for i in range(len(accent)):
            sentence = sentence.replace(accent[i], sans_accent[i])

        sentence = sentence.split()

        filteredtext = [t for t in sentence if t.lower() not in stopwords]

        s = " ";
        # on recole la phrase
        sentence = s.join( filteredtext )

        #Supprimer les stop-words manuellement et sans appel à ntlk
        sentence=re.sub(stopWordFr,'',sentence)

        #Ecriture du nouveau fichier à alligner RAMEAU
        nFichierBNF.write(sentence+'\n')
```

```
#/////////////// Lignes simplifiées avec le contenu après le premier $a seulement > RAMEAU
#////////////////// on ouvre le fichier RAMEAU , lecture ligne

#////////////////////////////////////////////////////////////////////////////
#//////////////////////////////        $a        ///////////////////////////////
#////////////////////////////////////////////////////////////////////////////

file2 = open('RAMEAU_utf-8.txt','r')
read2 = file2.read()

res2 = re.findall(r"(\d{8})\s+\$a(.+)\n",read2)




if res2:
    for line in res2:
        sentence=str(line)

        #Transformer les majuscules en minuscules
        sentence=sentence.lower()

        #Supprimer les accents
        for i in range(len(accent)):
            sentence = sentence.replace(accent[i], sans_accent[i])

        sentence = sentence.split()

        #Tokenisation & stop-words suppression
        filteredtext = [t for t in sentence if t.lower() not in stopwords]

        s = " ";

        # on recole la phrase
        sentence = s.join( filteredtext )
        #Garder que ce qui a après $a.*$|$a.*\n
        res3 = re.search(r".*\$(g.*)",sentence)
        if res3:
            aSupprimer = res3.group(1)
```

```
                sentence2=str(aSupprimer)

                #Supprimer le contenu de $g....
                sentence=re.sub('\$'+sentence2, '',sentence)

            #Supprimer les stop-words spécifiques
            sentence=re.sub(stopWordFr,'',sentence)

            #Ecriture du nouveau fichier à alligner RAMEAU
            nFichierRM.write(sentence+'\n')

    file1.close()
    file2.close()
    nFichierBNF.close()
    nFichierRM.close()
```

Alignement_BNF_RAMEAU_jaro.py :

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
from __future__ import division
from time import sleep

import time

import re, sys, os, math


#////////////////////// Création fichiers de retour ////////////////////////////
ethnieMatch = open('ethnieMatch.txt','x+') #Pour les BNF alignés grace à leur éthnie
regionMatch = open('regionMatch.txt','x+') #Pour les BNF alignés grace à la position géographique
ambigueMatch = open('ambigueMatch.txt','x+') #Pour les BNF alignés avec ambiguité
bnfNotMatch = open('bnfNotMatch.txt','x+') #Pour les BNF non alignés

#////////////////////// Conversion vers UTF 8 ////////////////////////////////
#////////////////////// lancer ces commandes dans le terminal/////////////////////

#iconv -f macintosh -t utf-8 RAMEAU-Groupes_ethniques-2.txt > RAMEAU_utf-8.txt
#iconv -f macintosh -t utf-8 BNF_traditions_field.txt > BNF_utf-8.txt ////// `A ne pas utiliser car le format initial du fichier est en UTF-8



#/////////////////////////////////////////////////////////////////////////
#//////////////////    MESURE JARO    ////////////////////////////////////
#/////////////////////////////////////////////////////////////////////////
def _get_diff_index(first, second):
    if first == second:
        return -1

    if not first or not second:
        return 0

    max_len = min(len(first), len(second))
    for i in range(0, max_len):
        if not first[i] == second[i]:
            return i

    return max_len
```

```python
def _get_prefix(first, second):
    if not first or not second:
        return ""

    index = _get_diff_index(first, second)
    if index == -1:
        return first

    elif index == 0:
        return ""

    else:
        return first[0:index]

def jaro(mot1, mot2): #s et t étant les chaines de caractère à aligner
    mot1_len = len(mot1) #s_len est le nombre de caractères dans s
    mot2_len = len(mot2) #t_len est le nombre de caractères dans t

    if mot1_len == 0 and mot2_len == 0: # si il n y a pas de lettres dans les deux mots, alors retourner la valeur 1 comme distance JARO
        return 1

    match_distance = (max(mot1_len, mot2_len) // 2) - 1 # on calcul la moitié de la distance du max des longueurs de s ou t et on soustrait cette valeur à 1

    #print('\n','Match_distance :',match_distance) #Console

    mot1_matches = [False] * mot1_len
    mot2_matches = [False] * mot2_len

    matches = 0
    transpositions = 0

    for i in range(mot1_len): #on parcours s
        start = max(0, i-match_distance) #start est le max entre 0 et la valeur de i-match_distance
        end = min(i+match_distance+1, mot2_len) #end est le min entre i+match_distance et la longueur de t

        for j in range(start, end): #
            if mot2_matches[j]:
                continue
            if mot1[i] != mot2[j]:
                continue
            mot1_matches[i] = True
            mot2_matches[j] = True
            matches += 1
            break
```

```
#////////////////////////////////////////////////////////////////////////////
#//////////////////////////////    TRAITEMENT    ///////////////////////////////
#////////////////////////////////////////////////////////////////////////////

#///////// Récupération lignes BNF grace à l'iD dans un dictionnaire BNF//////////

#On commence par créer nos listes d'iD et texte BNF
listeId = list()
listeTexte = list()

file1 =open('BNF_traditions_field.txt','r')

read1= file1.readlines()
for ligne in read1:

    #On remplie la liste des iD
    ligneIdBNF = re.search(r"([0-9]{8}.*).*",ligne)
    if ligneIdBNF:
        idBNF= ligneIdBNF.group(1)
        listeId.append(idBNF)#rajouter cet iD à la liste des iDs
    #On remplie la liste des Contenus reliés à chaque iD
    ligneTxtBNF = re.search(r"\d{3}\s+(\$a[A-Z][a-z]+.*)",ligne)
    if ligneTxtBNF:
        txtBNF= ligneTxtBNF.group(1)
        listeTexte.append(txtBNF)#rajouter ce cotenu à la liste des textes

#On fusionne et on crée un dictionnaire qui enregistre les iDs comme des clés et le texte comme valeur
BNF = dict(zip(listeId, listeTexte))


#///////// Récupération lignes RAMEAU grace à l'iD dans un dictionnaire RAMEAU//////////

#On commence par créer nos listes d'iD et texte RAMEAU
listeId = list()
listeTexte = list()
```

```
#On va traiter dans un premier lieu les $e présents sans information geographique ($eMalgaches)
resT = re.search(r"(.*)\\(.*)",mot1)
if not resT:
    #Tokenization BNF
    mot1S = mot1.split('\\')

    for line in read_mots2:
        res_mots2 = re.search(r"([0-9].*)~(.*)",line)
        if res_mots2:
            iD2 = res_mots2.group(1)
            mot2 = res_mots2.group(2)
            #Tokenization RAMEAU
            mot2S = mot2.split()

            #On définie les seuils minimums Jaro et Levenstein
            #Seuil 1 : same people (même peuple)
            if levenshteinN(mot1,mot2) >= 0.7 and jaro(mot1, mot2) > 0.9 : #and iD1 not in m
                for cle,valeur in BNF.items():
                    if cle==iD1:
                        ethnieMatch.write(cle+' '+valeur+'\n')#ecriture ligne BNF
                for cle,valeur in RAMEAU.items():
                    if cle==iD2:
                        ethnieMatch.write(cle+' '+valeur+'\n')#ecriture ligne Rameau
                ethnieMatch.write('JaroW:'+str(jaro(mot1,mot2))+' Levenstein: '+str(levenshteinN(mot1,mot2))+'\n\n')#ecriture mesures
                #iD aligné > à rajouter à la liste S,M
                m.append(iD1)
            else:
                # Si au minimum on a un mot match et jaro avec condition
                if levenshteinN(mot1S,mot2S) >= 0.5 and jaro(mot1, mot2) > 0.7:  #<<<<<<< ici vous mettez le seuil de similarité Jaro min
                    for cle,valeur in BNF.items():
                        if cle==iD1:
                            ambigueMatch.write(cle+'    '+valeur+'\n')#ecriture ligne BNF
                    for cle,valeur in RAMEAU.items():
                        if cle==iD2:
                            ambigueMatch.write(cle+'    '+valeur+'\n')#ecriture ligne BNF
                    ambigueMatch.write('JaroW:'+str(jaro(mot1,mot2))+' Levenstein: '+str(levenshteinN(mot1,mot2))+'\n\n')#ecriture mesures
```

```
#Dans un second lieu les $e qui dispose d'une information geographique ($mYemen$eJuifs)
else:
    #on identifie les ethnies et leur régions respectives
    region = resT.group(1)
    ethnie = resT.group(2)

    #on récupère les informations RAMEAU à aligner
    for line in read_mots2:
        #on identifie chaque terme avec son identifiant BNF
        res_mots2 = re.search(r"([0-9].*)~(.*)",line)
        if res_mots2:

            iD2 = res_mots2.group(1)
            mot2 = res_mots2.group(2)
            #On définieles seuils minimums Jaro et Levenstein
            #Seuil 1 : same people (même peuple)

            if levenshteinN(ethnie,mot2) >= 0.9 and jaro(ethnie, mot2) > 0.9 :  #<<<<<<< ici vous mettez le seuil de similarité Jaro min
                if iD1 not in m:
                    for cle,valeur in BNF.items():
                        if cle==iD1:
                            ethnieMatch.write(cle+' '+valeur+'\n')#ecriture ligne BNF
                    for cle,valeur in RAMEAU.items():
                        if cle==iD2:
                            ethnieMatch.write(cle+' '+valeur+'\n')#ecriture ligne Rameau
                    ethnieMatch.write('JaroW:'+str(jaro(ethnie,mot2))+' Levenstein: '+str(levenshteinN(ethnie,mot2))+'\n\n')#ecriture mesures
                    m.append(iD1)

            elif iD1 not in m: #PROBLEME AVEC CETTE CONDITION ///// CA NE FONCTIONNE PAS ////// POURQUOI?

                if levenshteinN(region,mot2) >= 0.6 and jaro(region, mot2) > 0.9 :  #Seuil 2 : same region (même région géographique)
                    #ecriture des alignements répondants aux conditions

                    for cle,valeur in BNF.items():
                        if cle==iD1:
                            regionMatch.write(cle+' '+valeur+'\n')#ecriture ligne BNF
                    for cle,valeur in RAMEAU.items():
                        if cle==iD2:
                            regionMatch.write(cle+' '+valeur+'\n')#ecriture ligne Rameau
                    regionMatch.write('JaroW:'+str(jaro(ethnie,mot2))+' Levenstein: '+str(levenshteinN(ethnie,mot2))+'\n\n')#ecriture mesures
                    m.append(iD1)
```

```
        # Si mot1 et mot2 sont composés de plusieurs mots, on mesure jaro et lev entre chaque mot des deux cotés
        if iD1 not in m:
            for cle,valeur in BNF.items():
                if cle==iD1:
                    bnfNotMatch.write(cle+' '+valeur+'\n')#ecriture ligne BNF



"""
bnfNotMatch.close()
nFichierBNF.close()
nFichierRM.close()
ethnieMatch.close()
regionMatch.close()
ambigueMatch.close()
os.remove('newBNF_utf-8.txt')
os.remove('newRAMEAU_utf-8.txt')
"""
```

tache3.py :

```python
#!/usr/bin/python3
# -*- coding: utf-8 -*

from itertools import dropwhile
import os, re



#////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#Traitement des fichiers et récupération titres & dates : ////////////////////////////////////////////////////////////////////////////////////////////////////
#////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////


#/////////////////////////////////////////////////////////////////////////
#////////////////////    MESURE JARO    //////////////////////////////////
#/////////////////////////////////////////////////////////////////////////

def _get_diff_index(first, second):
    if first == second:
        return -1

    if not first or not second:
        return 0

    max_len = min(len(first), len(second))
    for i in range(0, max_len):
        if not first[i] == second[i]:
            return i

    return max_len

def _get_prefix(first, second):
    if not first or not second:
        return ""

    index = _get_diff_index(first, second)
    if index == -1:
        return first

    elif index == 0:
        return ""

    else:
        return first[0:index]
```

```python
def jaro(mot1, mot2): #s et t étant les chaines de caractère à aligner
    mot1_len = len(mot1) #s_len est le nombre de caractères dans s
    mot2_len = len(mot2) #t_len est le nombre de caractères dans t

    if mot1_len == 0 and mot2_len == 0: # si il n y a pas de lettres dans les deux mots, alors retourner la valeur 1 comme distance JARO
        return 1

    match_distance = (max(mot1_len, mot2_len) // 2) - 1 # on calcul la moitié de la distance du max des longueurs de s ou t et on soustrait cette valeur à 1

    #print('\n','Match_distance :',match_distance) #Console

    mot1_matches = [False] * mot1_len
    mot2_matches = [False] * mot2_len

    matches = 0
    transpositions = 0

    for i in range(mot1_len): #on parcours s
        start = max(0, i-match_distance) #start est le max entre 0 et la valeur de i-match_distance
        end = min(i+match_distance+1, mot2_len) #end est le min entre i+match_distance et la longueur de t

        for j in range(start, end): #
            if mot2_matches[j]:
                continue
            if mot1[i] != mot2[j]:
                continue
            mot1_matches[i] = True
            mot2_matches[j] = True
            matches += 1
            break

    if matches == 0:
        return 0
    #print ('Matches :',matches) #Console
    k = 0
    for i in range(mot1_len):
        if not mot1_matches[i]:
            continue
        while not mot2_matches[k]:
            k += 1
        if mot1[i] != mot2[k]:
            transpositions += 1
        k += 1
    #print ('Transpositions :',transpositions) #Console
    result= ((matches / mot1_len) + (matches / mot2_len) + ((matches - transpositions/2) / matches)) / 3
```

```python
#define SCALING_FACTOR 0.1
    SCALING_FACTOR = 0.1
#calculer les caractères en commun prefix supérieur à 4 chars */

    l = min(len(_get_prefix(mot1, mot2)), 4)
    #Jaro-Winkler distance */
    resultW = result + (l * SCALING_FACTOR * (1 - result));

    return (round(resultW,3))
#////////////////////////////////////////////////////////////////////////////
#////////////////// MESURE LEVENSTEIN ///////////////////////////////////////
#////////////////////////////////////////////////////////////////////////////

def levenshteinN(mot1,mot2):
    ligne_i = [ k for k in range(len(mot1)+1) ]
    for i in range(1, len(mot2) + 1):
        ligne_prec = ligne_i
        ligne_i = [i]*(len(mot1)+1)
        for k in range(1,len(ligne_i)):
            cout = int(mot1[k-1] != mot2[i-1])
            ligne_i[k] = min(ligne_i[k-1] + 1, ligne_prec[k] + 1, ligne_prec[k-1] + cout)
    result = 1-(ligne_i[len(mot1)]/(max(len(mot1),len(mot2))))
    return(round(result,3))

#////////////////////////////////////////////////////////////////////////////
#Fonction qui récupère un bloc de texte qui commence par begin\d et se termine par end\d
#////////////////////////////////////////////////////////////////////////////
def get_block_lines(f, begin, end):
    for line in dropwhile(lambda x: begin not in x, f):
        yield line
        if end in line: return


#////////////////////////////////////////////////////////////////////////////
#Traitement de texte //////////////////////////////////////////////////////////
#////////////////////////////////////////////////////////////////////////////

#liste des caractères accentués a désaccentuer
accent = ['é', 'è', 'ê', 'ë', 'à','á','É', 'ù', 'û', 'ú', 'ü','ç','č', 'ô','ó','õ', 'í','ĺ','Î', 'ï', 'â','ã','ā','ţ','(','\',')','\', "','\')',")",'\\','/','ø']
sans_accent = ['e', 'e', 'e', 'e', 'a','a','E', 'u', 'u', 'u','u', 'c', 'c','o','o','o', 'i','i', 'I', 'i','a', 'a', 'a','t','','~','~','','','','o']

#liste stop words
stopwords = open('french', 'r').read().split()
stopWordFr=re.compile('d\'|l\'|d’|l’|\-|:|"|!|\(|\)|«|-|_|,')
```

```python
#Récupérer les titres + dates de chaque évenement depuis EUTERPE : ///////////////////////////////////////////////////////////////////////
#///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////


#Fonction pour parcourir le contenu des dossiers EUTERPE
repertoire="/auto_home/nbendjoudi/TER/TER2/Etape 2/FINAL/TACHE 3/euterpe"

def parcours(repertoire) :
    print("EUTERPE > ", repertoire)
    liste = os.listdir(repertoire)
    for fichier in liste :
        if os.path.isdir(repertoire+"/"+fichier) :
            parcours(repertoire+"/"+fichier)
            #print(repertoire+"/"+fichier)
        else :
            #REGEX qui vérifie l'extension des fichiers (*.ttl)
            resultat = re.search(".\.ttl", fichier)   # Il y a au moins un caractère avant l'extension
            if resultat:
                #On parcours fichier par fichier
                f=open(repertoire+"/"+fichier,'r')
                #On définit le block1 qui contient le titre et l'iD de l'événement
                begin = 'M26'
                end = '> .'
                block1 = ''.join(get_block_lines(f, begin, end))

                #Récupérer le titre
                reTitre=re.search(r'\s+ecrm:P102_has_title\s+"(.*)"',block1)
                if reTitre:
                    titre=reTitre.group(1) #titre mémorisé
                    titre=str(titre)
                    titrePrime = titre
                    #Transformer les majuscules en minuscules
                    titre=titre.lower()

                    #Supprimer les accents
                    for i in range(len(accent)):
                        titre = titre.replace(accent[i], sans_accent[i])


                    titre = titre.split()
                    #print(titre)
                    filteredtext = [t for t in titre if t.lower() not in stopwords]

                    s = "";
                    # on recole la phrase
                    titre = s.join(filteredtext)
                    #Supprimer les stop-words manuellement et sans appel à ntlk
                    titre=re.sub(stopWordFr,'',titre)

                #On referme le fichier f pour pouvoir le relire à nouveau pour extraire les informations de date, car si on ne fait pas ça, le programme récupérera que les dates situées ap
                f.close()

                #Récupérer la date
                #On relis le fichier à nouveau
                f=open(repertoire+"/"+fichier,'r')
                fR=f.readlines()
                #REGEX iD de l'événement date
                reDateUri=re.search(r'\s+mus:U8_foresees_time_span\s+(.*)\s+;',block1)
                if reDateUri:
                    #On définit le block2 qui contient la date en utilisant l'iD récupéré dans la précédente opérantion
                    dateUri=reDateUri.group(1) #iD mémorisé
                    begin2 = dateUri #iD utilisé comme début du block2
                    end2='dateTime'
                    block2 = ''.join(get_block_lines(fR, begin2, end2))
                    #REGEX date
                    reDate= re.search(r'\s+"(\d{4}-\d{2}-\d{2}T\d{2}:\d{2}).*".*dateTime\s+',block2)
                    if reDate:
                        date=reDate.group(1)
                        euterpe.write(fichier+' '+titre+' $date:'+date+'    '+titrePrime+'\n')
                f.close()
#création fichier contenant les résultats EUTERPE
euterpe = open('euterpe.txt','x+')
parcours(repertoire)
euterpe.close()
```

```
#Récupérer les titres + dates de chaque évenement depuis PP : ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

repertoire="/auto_home/nbendjoudi/TER/TER2/Etape 2/FINAL/TACHE 3/pp"

def parcours(repertoire) :
    print("PP > ", repertoire)
    liste = os.listdir(repertoire)
    for fichier in liste :
        if os.path.isdir(repertoire+"/"+fichier) :
            parcours(repertoire+"/"+fichier)
            #print(repertoire+"/"+fichier)
        else :
            #REGEX qui vérifie l'extension des fichiers (*.ttl)
            resultat = re.search(".\.ttl", fichier)   # Il y a au moins un caractère avant l'extension
            if resultat:
                #On parcours fichier par fichier
                f=open(repertoire+"/"+fichier,'r')
                #On définit le block1 qui contient le titre et l'iD de l'événement
                begin = 'F31'
                end = '> .'
                block = ''.join(get_block_lines(f, begin, end))

                #Récupérer le titre
                reTitre=re.search(r'\s+ecrm:P102_has_title\s+"(.*)"',block)
                if reTitre:
                    titre=reTitre.group(1) #titre mémorisé
                    titre=str(titre)
                    titrePrime = titre
                    #Transformer les majuscules en minuscules
                    titre=titre.lower()

                    #Supprimer les accents
                    for i in range(len(accent)):
                        titre = titre.replace(accent[i], sans_accent[i])


                    titre = titre.split()

                    filteredtext = [t for t in titre if t.lower() not in stopwords]

                    s = "";
                    # on recole la phrase
                    titre = s.join( filteredtext )

                    #Supprimer les stop-words manuellement et sans appel à ntlk
                    titre=re.sub(stopWordFr,'',titre)
                #On referme le fichier f pour pouvoir le relire à nouveau pour extraire les informations de date, car si on ne fait pas ça, le programme récupérera que les dates
                f.close()

                #Récupérer la date
                #On relis le fichier à nouveau
                f=open(repertoire+"/"+fichier,'r')
                fR=f.readlines()
                #REGEX iD de l'événement
                reDateUri=re.search(r'\s+ecrm:P4_has_time-span\s+(.*)\s+;',block)
                if reDateUri:
                    #On définit le block2 qui contient la date en utilisant l'iD récupéré dans la précédente opérantion

                    dateUri=reDateUri.group(1) #iD mémorisé
                    begin2 = dateUri #iD utilisé comme début du block2
                    end2='date\s'
                    block2 = ''.join(get_block_lines(fR, begin2, end2))
                    #REGEX date
                    reDate= re.search(r'\s+"(\d{4}-\d{2}-\d{2}T\d{2}:\d{2}).*".*dateTime\s+',block2)
                    if reDate:
                        date=reDate.group(1)
                        pp.write(fichier+'  '+titre+' $date:'+date+'    '+titrePrime+'\n')
                f.close()
#création fichier contenant les résultats PP
pp = open('pp.txt','x+')
parcours(repertoire)
pp.close()
```

```
#Aligner les titres, ensuite les dates depuis EUTERPE vers PP : ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
EMTD=open('EMTD.txt','x+')
EMT=open('EMT.txt','x+')
EMTrD=open('EMTrD.txt','x+')
EMTr=open('EMTr.txt','x+')

#/////////////// on collecte les termes à mesurer ////////////////////////////

mots1 = open('euterpe.txt','r')
read_mots1= mots1.readlines()

mots2 = open('pp.txt','r')
read_mots2= mots2.readlines()
#print(read_mots1)
s=list()
#on récupère les informations EUTERPE à aligner
for ligne in read_mots1:
    #on identifie les titres et dates
    res_mots1 = re.search(r'(.*)\t(.*)\$date:(.*)\t(.*)',ligne)
    if res_mots1:
        nFchierEuterpe = res_mots1.group(1)
        titre1 = res_mots1.group(2)
        date1 = res_mots1.group(3)
        titrePrime1= res_mots1.group(4)
        #print(nFchierEuterpe,titre1,date1)


        for ligne in read_mots2:
            #on identifie les titres et dates
            res_mots2 = re.search(r'(.*)\t(.*)\$date:(.*)\t(.*)',ligne)
            if res_mots2:
                nFchierPP = res_mots2.group(1)
                titre2 = res_mots2.group(2)
                date2 = res_mots2.group(3)
                titrePrime2= res_mots2.group(4)

                #print(titre2,date2)
                #On définie les seuils minimums Jaro et Levenstein
                #Seuil 1 : same people (même peuple)
```

```
348                     if levenshteinN(date1,date2) ==1.0 and jaro(date1,date2) == 1.0:#dates exactes
349                         if levenshteinN(titre1,titre2) >= 0.9 and jaro(titre1,titre2) > 0.9:  #titres exactes
350
351                             #ecriture des alignements répondants aux conditions
352                             EMTD.write(titrePrime1+':'+date1+'\n'+titrePrime2+':'+date2+'\n'+'JaroW:'+str(jaro
353                             if titre1 not in s:
354                                 s.append(titre1)
355                         elif levenshteinN(titre1,titre2) >= 0.5 and jaro(titre1,titre2) > 0.7: #titres légèrem
356                             EMTrD.write(titrePrime1+':'+date1+'\n'+titrePrime2+':'+date2+'\n'+'JaroW:'+str(jar
357                             if titre1 not in s:
358                                 s.append(titre1)
359                         else: #titres complètement différents
360                             EMD.write(titrePrime1+':'+date1+'\n'+titrePrime2+':'+date2+'\n'+'JaroW:'+str(jaro(
361
362                     elif levenshteinN(titre1,titre2) >= 0.9 and jaro(titre1,titre2) > 0.9:  #titres exactes
363                             #ecriture des alignements répondants aux conditions
364                             EMT.write(titrePrime1+':'+date1+'\n'+titrePrime2+':'+date2+'\n'+'JaroW:'+str(jaro(
365                             if titre1 not in s:
366                                 s.append(titre1)
367
```