Master Informatique, parcours MALIA-MIASHS

Carnets de note Python pour le cours de Network Analysis for Information Retrieval

Julien Velcin, laboratoire ERIC, Université Lyon 2

Prétraitements (partie 1)

Quelques prétraitements à connaître

- expressions régulières et nettoyages simples
- segmentation en mots (*tokenization*)
- mots-outils
- stemming et lemmatisation
- n-grammes et collocations

Quelques prétraitements à connaître

- **expressions réguières et nettoyages simples**
- segmentation en mots (*tokenization*)
- mots-outils
- stemming et lemmatisation
- n-grammes et collocations

expressions régulières

In [1]:
```
import os

texte = "" # chaîne vide
with open("datasets/Frank Herbert - Dune.txt", "r", encoding='utf8') as f:
    texte = texte.join(line.rstrip("\n") + " " for line in f.readlines())
```

In [2]:
```
import re
pattern = re.compile("spice")

# cherche toutes les occurrences
res = pattern.finditer(texte)

start_pattern = [m.start() for m in res]
```

```
#print(start_pattern)
print(len(start_pattern))
```

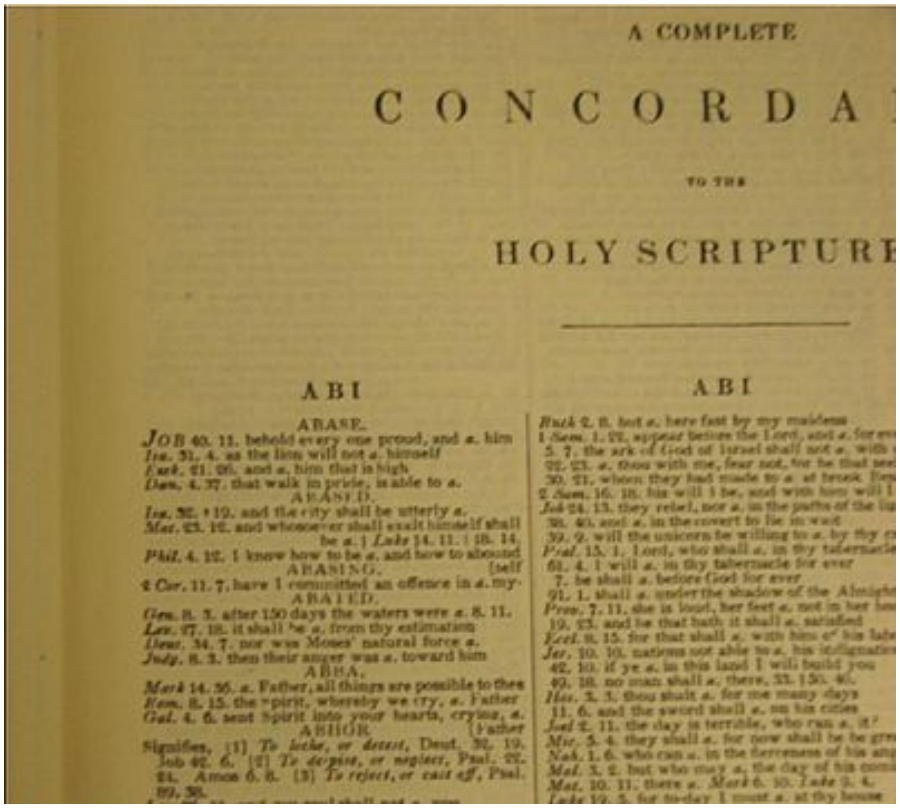
226

In [3]:
```
import re # notez qu'importer à nouveau la librairie n'est pas nécessaire
pattern = re.compile("spice", re.IGNORECASE)

# cherche la première occurrence seulement
#res = pattern.search(texte)

# cherche toutes les occurrences
res = pattern.finditer(texte)

start_pattern = [m.start() for m in res]

print(len(start_pattern))
#sp = [m.span() for m in res]
#print(sp)
#print(start_pattern[100])
#print(texte[start_pattern[100]-10 : start_pattern[100]+20])
```

236

Si on souhaite trouver toutes les occurrences, on peut afficher ce qu'on appelle un concordancier.

Depuis le XIIIème siècle, un **concordancier** est une liste triée alphabétiquement des principaux mots employés dans un corpus, précisant **chaque instance** des mots accompagnée de leur **contexte immédiat**.


A COMPLETE

CONCORDA

TO THE

HOLY SCRIPTURE

ABI

ABASE.

Job 40. 11. beholde every one proud, and *a.* him
Isa. 31. 4. as the lion will not *a.* himself
Ezek. 21. 26. and *a.* him that is high
Dan. 4. 37. that walk in pride, is able to *a.*

ABASED.

Isa. 32. 19. and the city shall be utterly *a.*
Mat. 23. 12. and whosoever shall exalt himself shall be *a.* | *Luke* 14. 11. | 18. 14.
Phil. 4. 12. I know how to be *a.* and how to abound

ABASING. [self

2 *Cor.* 11. 7. have I committed an offence in *a.* my-

ABATED.

Gen. 8. 3. after 150 days the waters were *a.* 8. 11.
Lev. 27. 18. it shall be *a.* from thy estimation
Deut. 34. 7. nor was Moses' natural force *a.*
Judg. 8. 3. then their anger was *a.* toward him

ABBA.

Mark 14. 36. *a.* Father, all things are possible to thee
Rom. 8. 15. the spirit, whereby we cry, *a.* Father
Gal. 4. 6. sent spirit into your hearts, crying, *a.*

ABHOR. [Father

Signifies, (1) *To loathe, or detest*, Deut. 32. 19. Job 42. 6. (2) *To despise, or neglect*, Psal. 22. 24. Amos 6. 8. (3) *To reject, or cast off*, Psal. 89. 38.

ABI

Ruth 2. 8. but *a.* here fast by my maidens
1 *Sam.* 1. 22. appear before the Lord, and *a.* for ev
5. 7. the ark of God of Israel shall not *a.* with
22. 23. *a.* thou with me, fear not, for he that see
30. 21. whom they had made to *a.* at brook Bes
2 *Sam.* 16. 18. his will I be, and with him will I
Job 24. 13. they rebel, nor *a.* in the paths of the li
38. 40. and *a.* in the covert to lie in wait
39. 9. will the unicorn be willing to *a.* by thy c
Psal. 15. 1. Lord, who shall *a.* in thy tabernacle
61. 4. I will *a.* in thy tabernacle for ever
7. he shall *a.* before God for ever
91. 1. shall *a.* under the shadow of the Almigh
Prov. 7. 11. she is loud, her feet *a.* not in her ho
19. 23. and he that hath it shall *a.* satisfied
Eccl. 8. 15. for that shall *a.* with him of his labo
Jer. 10. 10. nations not able to *a.* his indignatio
42. 10. if ye *a.* in this land I will build you
49. 18. no man shall *a.* there, 33. | 50. 40.
Hos. 3. 3. thou shalt *a.* for me many days
11. 6. and the sword shall *a.* on his cities
Joel 2. 11. the day is terrible, who can *a.* it?
Mic. 5. 4. they shall *a.* for now shall he be gre
Nah. 1. 6. who can *a.* in the fierceness of his an
Mal. 3. 2. but who may *a.* the day of his com
Mat. 10. 11. there *a.* *Mark* 6. 10. *Luke* 9. 4.
Luke 19. 5. for to-day I must *a.* at thy house

Cruden's Concordance (concordance of the King James Bible that was single-handedly created by Alexander Cruden)

In [4]:
```
import pandas as pd

window = 50

def concord(texte, pat):
    pattern = re.compile(pat)
    res = pattern.finditer(texte)
    pos_pattern = [m.span() for m in res]
    context_left = pd.DataFrame([texte[i-window:i-1] for (i, j) in pos_pattern])
    center = pd.DataFrame([texte[i: j] for (i, j) in pos_pattern])
    context_right = pd.DataFrame([texte[j+1:j+window] for (i, j) in pos_pattern])
    return (pd.concat([context_left, center, context_right], axis=1))
```

Quelques exemples d'expressions régulières :

In [5]:
```
# suite exacte :
concord(texte, "CHOAM Company")

# présence optionnelle d'un caractère :
#concord(texte, "dune?")

# chiffre :
#concord(texte, "[0-9]")

# suite de chiffres :
#concord(texte, "[0-9]+")   # le + indique qu'il doit au moins y avoir un chiffre
#concord(texte, "\d+")   # \d est plus général pour les suites de chiffre

# recherche de deux motifs :
#concord(texte, "Paul|Atreid")    # le | (ou "pipe") indique un OU logique
```

Out[5]:

	0	0	0
0	y years, holding the planet in quasi-fief under a	CHOAM Company	contract to mine the geriatric spice, melange. No
1	in, we'll have an irrevocable directorship in the	CHOAM Company	" Feyd-Rautha nodded. Wealth was the thing.
2	e genetic strains without plan. The Imperium, the	CHOAM Company	all the Great Houses, they are but bits of flots
3	Combine Honnete Ober Advancer Mercantiles -- the	CHOAM Company	By giving me Arrakis, His Majesty is forced to g
4	ugh, I'll have my own report in his hands through	CHOAM Company	channels. I will explain that I luckily discovere
5	. . . ah . . . have no objections?" "None. My	CHOAM Company	directorship will bear the closest scrutiny." And
6	, to the Harkonnen frigates lined up there with a	CHOAM Company	banner waving gently from its staff on the ground
7	re's a subtle piece of business," Paul said. "The	CHOAM Company	flag." "It's the same as the flag at the othe
8	this Arrakis nonsense has taken from me? Nor the	CHOAM Company	profits pouring down this rat hole? Nor the court
9	structions?" she asked. "The Emperor's entire	CHOAM Company	holdings as dowry," he said. "Entire?" She wa

L'action la plus élémentaire consiste à nettoyer le texte s'il contient des symboles non souhaités (par ex. des caractères unicodes).

In [6]:
```
import re

txt_a_nettoyer = "ÅVoilà un texte qui pose problÞme  –voyons ce que l ơn peut y fai

txt_propre = re.sub("Å", " ", txt_a_nettoyer)
txt_propre = re.sub("Þ", "è", txt_propre)
txt_propre = re.sub(" –", "-", txt_propre)
txt_propre = re.sub(" ‴", "'", txt_propre)

print(txt_propre)
```

```
Voilà un texte qui pose problème - voyons ce que l'on peut y faire
```

In [7]:
```
clean_unicode = {
  "Å": " ",
  "Þ": "è",
  " –" : "-",
  " ‴" : "'"
}

txt_propre_2 = txt_a_nettoyer
for c in clean_unicode:
```

```
    txt_propre_2 = re.sub(c, clean_unicode[c], txt_propre_2)

print(txt_propre_2)
```

```
Voilà un texte qui pose problème - voyons ce que l'on peut y faire
```

Il est également simple de remplacer une expression par une autre, par exemple les occurrences de "chaumas" par "poison".

In [8]:
```
# la fonction re.sub permet de faire un "remplacer tout" pour les occurrences d'
# attention à penser à sauvegarder le résultats, c'est-à-dire la nouvelle chaîne de

texte_comp = re.sub("chaumas", "poison", texte)
```

In [9]:
```
# on vérifie que le remplacement a bien été réalisé

num_it = 0 # numéro de l'instance qu'on souhaite vérifier

liste_it_texteini = [m.start() for m in re.finditer("chaumas", texte)]
print(texte[liste_it_texteini[num_it]-10: liste_it_texteini[num_it]+50])

liste_it_textetransf = [m.start() for m in re.finditer("poison", texte_comp)]
print(texte_comp[liste_it_textetransf[num_it]-10: liste_it_textetransf[num_it]+50])
#print(sp_pgpp_comp)
```

```
ill it be chaumas--poison in the food?     He shook his head
a drop of poison on its tip. Ah-ah! Don't pull away or you'l
```

On peut utiliser des expressions régulières plus complexes.

Par exemple pour chercher tous les nombres qui peuvent comporter des espaces, des virgules ou des points, dans leur écriture :

In [10]:
```
int_pat = "[0-9]+((\s|,|\.)|[0-9])+"   #\s indique n'importe quel espace

concord(texte, int_pat)
```

```
<>:1: SyntaxWarning: invalid escape sequence '\s'
<>:1: SyntaxWarning: invalid escape sequence '\s'
/var/folders/44/_q8kssp12vb3ks1rlb59jm6m0000gp/T/ipykernel_15709/3533720138.py:1: Sy
ntaxWarning: invalid escape sequence '\s'
  int_pat = "[0-9]+((\s|,|\.)|[0-9])+"   #\s indique n'importe quel espace
```

Out[10]:

	0	0	0
0		1965	ook 1 DUNE = = = = = = A beginning is the time
1		1	UNE = = = = = = A beginning is the time for ta
2	that you first place him in his time: born in the	57	h year of the Padishah Emperor, Shaddam IV. And t
3	= = YUEH (yu'e), Wellington (weling-tun), Stdrd	10,082	10,191; medical doctor of the Suk School (grd Std
4	UEH (yu'e), Wellington (weling-tun), Stdrd 10,082	10,191	medical doctor of the Suk School (grd Stdrd 10,1
...	...	...	...
81	rice on the Imperial market has ranged as high as	620,000	olaris the decagram. MENTAT: that class of Imperi
82	quartz. Noted for extreme tensile strength (about	450,000	ilos per square centimeter at two centimeters' th
83	orms grow to enormous size (specimens longer than	400	eters have been seen in the deep desert) and live
84	e-dimensional image from a solido projector using	360	degree reference signals imprinted on a shigawire
85	of Maometh (the so-called "Third Muhammed") about	1381	.G. The Zensunni religion is noted chiefly for it

86 rows × 3 columns

Un nettoyage standard pour les analyses ultérieures consiste à passer tout le corpus en minuscule. C'est une forme de normalisation permettant de rapprocher des termes comme "Unité" et "unité" par exemple.

In [11]:
```
print("avant : ", texte[1500:1600])
print("après : ", texte[1500:1600].lower())

texte_minuscules = texte.lower()
```

```
avant :  e glittering jewels.     "Is he not small for his age, Jessica?" the old
man asked. Her voice whee
après :  e glittering jewels.     "is he not small for his age, jessica?" the old wo
man asked. her voice whee
```

Quelques prétraitements à connaître

- expressions réguières et nettoyages simples
- **segmentation en mots (*tokenization*)**

- mots-outils
- stemming et lemmatisation
- n-grammes et collocations

segmentation en mots (*tokenization*)

Split a string into basic (lexical) **units**, such as words.

Different rules for different languages:

我已经等我的包裹三星期了!
(I've been waiting three weeks for my parcel to arrive!)

En français,c'est assez simple mais
ilfaut faire attention
notamment sur plusieurs lignes.

Even in English or French, you can have difficulties splitting strings.

In [48]:
```
chaine = "Here is, /for sure, a small paragraph I'd like to parse\non multiple l
re.sub("[\\.'/,:!?\n]", "", chaine)
```

Out[48]:
```
'Here is for sure a small paragraph Id like to parseon multiple linesIs this tha
simpleI guess not'
```

En général, avec la plupart des langues européennes, c'est une tâche assez facile si on utilise les séparateurs suivants :

, . \n \t - : ; ( ) ! ? [ ] _ ' " (etc.)

Mais... :

> "Harry Potter" => "Harry", "Potter"
> "rez-de-chaussée" => "chaussée", "de", "rez"
> "idiot?" => "idiot"
> "C.E.O" => "C", "E", "O"

Nous allons voir dans les carnets qui suivant et que les librairies Python comme *nltk*, *scikit-learn* ou *spacy* embarquent des tokeniseurs puissants.

Cependant, essayons de réaliser cette opération manuellement afin de mesurer les mécanismes sous-jacents avec le texte suivant :

"The challenge of exploiting the large proportion of enterprise information that originates in "unstructured" form has been recognized for decades.[7] It is recognized in the earliest definition of business intelligence (BI), in an October 1958 IBM Journal article by H.P. Luhn, A

Business Intelligence System, which describes a system that will:"
(excerpt of https://en.wikipedia.org/wiki/Text_mining)

In [49]:
```
ch = "The challenge of exploiting the large proportion of enterprise information

re.split(r'\W+', ch)
```

Out[49]:
```
['The',
 'challenge',
 'of',
 'exploiting',
 'the',
 'large',
 'proportion',
 'of',
 'enterprise',
 'information',
 'that',
 'originates',
 'in',
 'unstructured',
 'form',
 'has',
 'been',
 'recognized',
 'for',
 'decades',
 '7',
 'It',
 'is',
 'recognized',
 'in',
 'the',
 'earliest',
 'definition',
 'of',
 'business',
 'intelligence',
 'BI',
 'in',
 'an',
 'October',
 '1958',
 'IBM',
 'Journal',
 'article',
 'by',
 'H',
 'P',
 'Luhn',
 'A',
 'Business',
 'Intelligence',
 'System',
 'which',
 'describes',
 'a',
 'system',
 'that',
 'will',
 '']
```

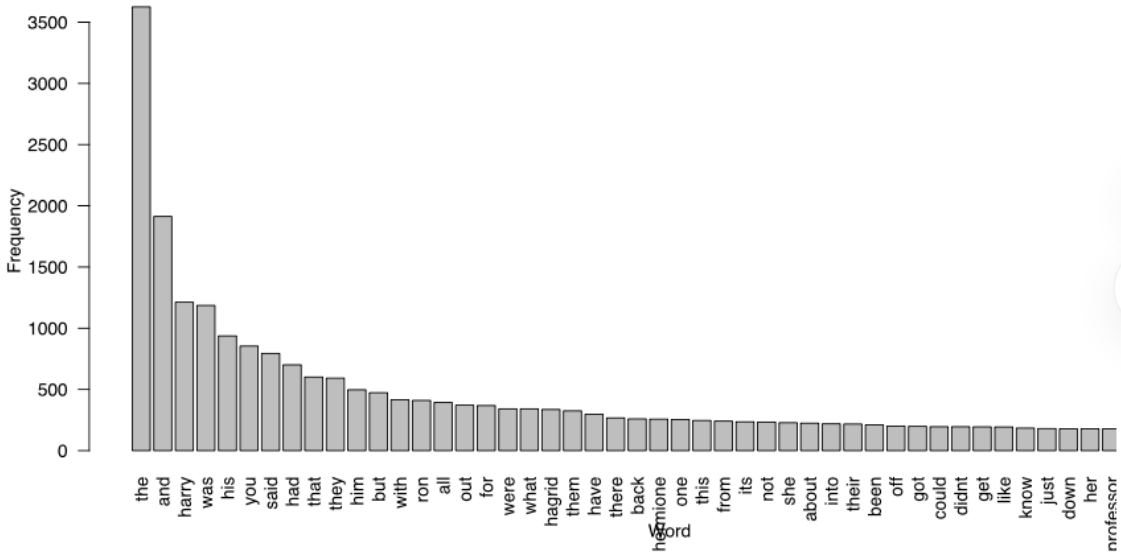
Quelques prétraitements à connaître

- expressions réguières et nettoyages simples
- segmentation en mots (*tokenization*)
- **mots-outils**
- stemming et lemmatisation
- n-grammes et collocations

mots-outils

Commençons par quelques observations générales sur des corpus bien connus.

Mots les plus fréquents observés dans Harry Potter :



Cette forme particulière est appelée "loi de Zipf". Elle a été observée pour la première fois par le linguiste George K. Zipf (1902-1950).

La loi de Zipf indique que si l'on observe un corpus suffisamment grand, la fréquence d'apparition d'un mot est **inversement proportionnelle** à son rang dans la table des fréquences.

Les mots-outils sont des mots qui n'apportent pas beaucoup d'information et sont surtout utilisés pour construire les phrases. Ils jouent donc un rôle *fonctionnel*.

Il faut faire un peu attention mais, la plupart du temps, on les enlève.

La librairie *scikit-learn* fournit ses propres listes de mots-outils, mais *nltk* a une bibliothèque plus fournie de langues. Il est bien entendu possible de charger votre propre liste de mots-outils.

In [14]:
```
# il ne faut pas oublier de télécharger les ressources nécessaires, en particuli
#import nltk
#nltk.download()
```

In [50]:
```
from nltk.corpus import stopwords
print("Mots-outils en anglais : {}".format(stopwords.words('english')[0:20]))
print("Mots-outils en français : {}".format(stopwords.words('french')[0:20]))
print("Mots-outils en arabe : {}".format(stopwords.words('arabic')[0:20]))
```

```
Mots-outils en anglais : ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselv
s', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'your
selves', 'he', 'him', 'his']
Mots-outils en français : ['au', 'aux', 'avec', 'ce', 'ces', 'dans', 'de', 'des', 'd
u', 'elle', 'en', 'et', 'eux', 'il', 'ils', 'je', 'la', 'le', 'les', 'leur']
Mots-outils en arabe : ['إذ', 'إذا', 'إذما', 'إذن', 'أف', 'أقل', 'أكثر', 'ألا', 'إلا', 'التي', 'الذ
ي', 'اللاتي', 'اللائي', 'اللتان', 'اللتيا', 'اللتين', 'اللذان', 'اللذين', 'اللواتي']
```

On peut tester ces listes à la main en passant par un *tokenizer* maison sous nltk.

In [51]:
```
from nltk.tokenize import word_tokenize

example_sent = "Here is a simple example of a sentence with multiple stopwords"

stop_words = set(stopwords.words('english'))
word_tokens = word_tokenize(example_sent)

#print(word_tokens)
filtered_sentence = [w for w in word_tokens if not w in stop_words]

print(filtered_sentence)
```

```
['Here', 'simple', 'example', 'sentence', 'multiple', 'stopwords']
```

Suivant l'hypothèse du "petit monde", on peut développer un algorithme simple mais efficace d'identification de la langue si la longueur d'un texte est assez grande.

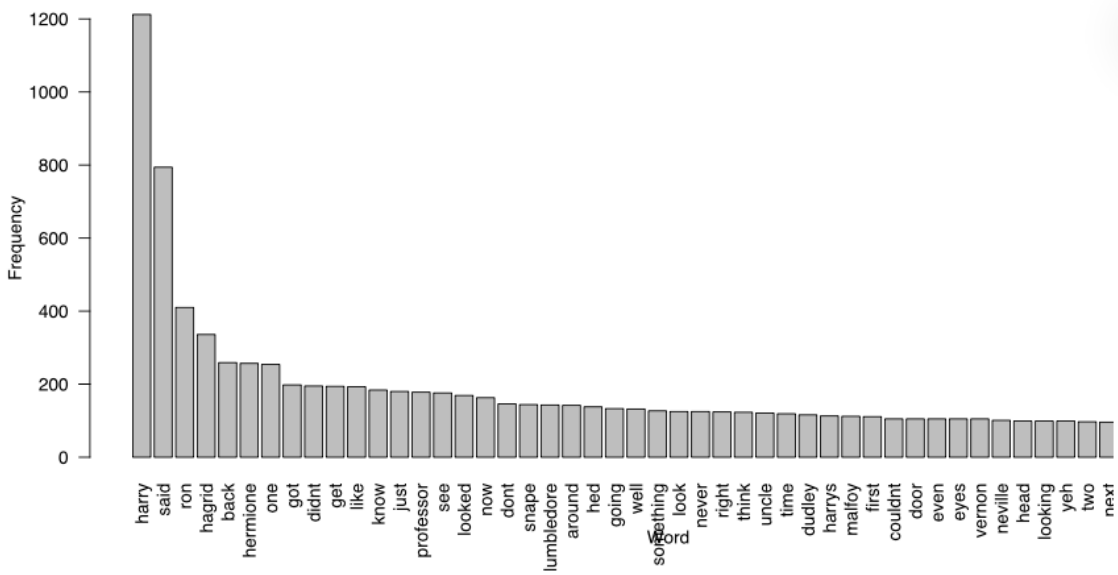
Si $|T| > 30$, alors on obtient 99,5% de réussite (Grefenstette 1995).

In [52]:
```
import numpy as np

languages = (
        "arabic",
        "danish",
        "dutch",
        "english",
        "finnish",
        "french",
        "german",
        "hungarian",
        "italian",
```

```
        "norwegian",
        "portuguese",
        "romanian",
        "russian",
        "spanish",
        "swedish",
    )

ch = "Las Antillas, o islas del Caribe, estan situadas junto al tropico de Cancer.
#ch = "Voilà un test en français un peu plus long et on continue"
#ch = "hello there, how are you guys?"

# compte le nombre de mots-outils pour une langue donnée
def nb_stopwords(s, langue):
    stop_words = set(stopwords.words(langue))
    word_tokens = set(word_tokenize(s))
    return len(stop_words.intersection(word_tokens))

list_val = [nb_stopwords(ch, l) for l in languages]

print(list_val)
print("My guess is: {}".format(languages[np.argmax(list_val)]))
```

```
[0, 1, 2, 1, 0, 1, 0, 2, 3, 1, 2, 4, 0, 6, 1]
My guess is: spanish
```

Même graphique mais en supprimant les mots-outils anglais :



Quelques prétraitements à connaître

- expressions réguières et nettoyages simples
- segmentation en mots (*tokenization*)
- mots-outils
- **stemming et lemmatisation**
- n-grammes et collocations

racinisation et lemmatisation

D'autres prétraitements sont également disponibles, tels que :

- racinisation (*stemming*) : trouver la racine des mots, comme dans :

> learn: learns, learned, learning...
> march: marcher, marchera, marcherai...

- lemmatisation (*lemmatization*) : trouver le lemme, comme dans :

> to be: am, are
> être : suis, sont...

La racinisation (*stemming*) nous aide à :

- rapprocher deux documents sémantiquement liés mais n'utilisant pas exactement les mêmes termes
- réduire la taille du vocabulaire (cf. malédiction de la dimension)

Ok pour :

```
adventur: [adventure,adventurer,adventurers,adventures,adventurou
```

mais...

```
anim: [animal, animals, animation]
```

In [53]:
```python
from nltk.stem.snowball import SnowballStemmer

stemmer_fr = SnowballStemmer("french")
stemmer_en = SnowballStemmer("english") # à priori, l'algorithme de Porter
```

In [54]:
```python
#stemmer.stem("maisons")
print("En français :")
print([stemmer_fr.stem(w) for w in ["marcher", "marcherons", "marcherait", "marché"
print([stemmer_fr.stem(w) for w in ["aventure", "aventures", "aventuriers", "aventu
```

```
print("En anglais :")
print([stemmer_en.stem(w) for w in ["adventures", "adventurers", "adventurous", "ad
```

```
En français :
['march', 'march', 'march', 'march']
['aventur', 'aventur', 'aventuri', 'aventur']
En anglais :
['adventur', 'adventur', 'adventur', 'adventur', 'adventur']
```

In [55]:
```
# Pour l'utilisation de CountVectorizer avec la librairie scikit-learn, voir le
from sklearn.feature_extraction.text import CountVectorizer

analyzer = CountVectorizer().build_analyzer()

def stemmed_words_en(doc):
    return (stemmer_en.stem(w) for w in analyzer(doc))

stem_vectorizer_en = CountVectorizer(analyzer=stemmed_words_en)

X = ["Here is the adventures of an adventurous adventurer"]
stem_vectorizer_en.fit(X)
stem_vectorizer_en.vocabulary_
```

Out[55]:
```
{'here': 2, 'is': 3, 'the': 5, 'adventur': 0, 'of': 4, 'an': 1}
```

La lemmatisation (*lemmatization*) transforme les mots en leur lexème sous-jacent (*lemma*).

> running --> to run (verb) / running (noun)
> are --> to be (verb)

Pour cela, il est nécessaire de résoudre le problème de détection des catégories grammaticales (*POS tagging*) et de recourir à une base de connaissances lexicales, voire des données annotées.

La lemmatisation est plus précise que la racinisation, mais elle nécessite plus de ressources. Cela revient à un choix entre + de précision et - de rappel (lemmatisation) vs. - de précision et + de rappel (racinisation).

On peut utiliser la library stanza (https://stanfordnlp.github.io/stanza/) qui fonctionne en général bien dans beaucoup de langues ou la librairie spacy (https://spacy.io) :

In [66]:
```
# pour Stanza :

#import stanza
#stanza.download('fr') # modèle pour le français
#   (attention, c'est assez lourd ~500Mo, il vaut mieux avoir une bonne connexion)
#nlp = stanza.Pipeline('fr') # pipeline pour le français
#doc = nlp("Barack Obama est né à Hawaii.") # premier test d'annotation

# avec Spacy :

import spacy
```

```
# Il faut avoir installé la ressource en local, par ex. :

#python -m spacy download en_core_news_sm
# ou :
#python -m spacy download en
# (attention, à le faire en console)

#python -m spacy download fr_core_news_sm

#nlp = spacy.load('fr')
#nlp = spacy.load('en')
nlp = spacy.load('fr_core_news_sm')
#nlp = spacy.load('en_core_web_sm')
```

In [68]:
```
corpus = [
    "La recherche d'information est le domaine qui étudie la manière de retrouver d
    "Avec l'apparition des premiers ordinateurs est née l'idée d'utiliser des machi
]

docs = [nlp(d) for d in corpus]
```

In [76]:
```
docs[0].to_json()
```

Out[76]:

```
{'text': "La recherche d'information est le domaine qui étudie la manière de retrouver des informations dans un corpus.",
 'ents': [],
 'sents': [{'start': 0, 'end': 109}],
 'tokens': [{'id': 0,
   'start': 0,
   'end': 2,
   'tag': 'DET',
   'pos': 'DET',
   'morph': 'Definite=Def|Gender=Fem|Number=Sing|PronType=Art',
   'lemma': 'le',
   'dep': 'det',
   'head': 1},
  {'id': 1,
   'start': 3,
   'end': 12,
   'tag': 'NOUN',
   'pos': 'NOUN',
   'morph': 'Gender=Fem|Number=Sing',
   'lemma': 'recherche',
   'dep': 'nsubj',
   'head': 6},
  {'id': 2,
   'start': 13,
   'end': 15,
   'tag': 'ADP',
   'pos': 'ADP',
   'morph': '',
   'lemma': 'de',
   'dep': 'case',
   'head': 3},
  {'id': 3,
   'start': 15,
   'end': 26,
   'tag': 'NOUN',
   'pos': 'NOUN',
   'morph': 'Gender=Fem|Number=Sing',
   'lemma': 'information',
   'dep': 'nmod',
   'head': 1},
  {'id': 4,
   'start': 27,
   'end': 30,
   'tag': 'AUX',
   'pos': 'AUX',
   'morph': 'Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin',
   'lemma': 'être',
   'dep': 'cop',
   'head': 6},
  {'id': 5,
   'start': 31,
   'end': 33,
   'tag': 'DET',
   'pos': 'DET',
   'morph': 'Definite=Def|Gender=Masc|Number=Sing|PronType=Art',
   'lemma': 'le',
```

```
 'dep': 'det',
 'head': 6},
{'id': 6,
 'start': 34,
 'end': 41,
 'tag': 'NOUN',
 'pos': 'NOUN',
 'morph': 'Gender=Masc|Number=Sing',
 'lemma': 'domaine',
 'dep': 'ROOT',
 'head': 6},
{'id': 7,
 'start': 42,
 'end': 45,
 'tag': 'PRON',
 'pos': 'PRON',
 'morph': 'PronType=Rel',
 'lemma': 'qui',
 'dep': 'nsubj',
 'head': 8},
{'id': 8,
 'start': 46,
 'end': 52,
 'tag': 'VERB',
 'pos': 'VERB',
 'morph': 'Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin',
 'lemma': 'étudier',
 'dep': 'acl:relcl',
 'head': 6},
{'id': 9,
 'start': 53,
 'end': 55,
 'tag': 'DET',
 'pos': 'DET',
 'morph': 'Definite=Def|Gender=Fem|Number=Sing|PronType=Art',
 'lemma': 'le',
 'dep': 'det',
 'head': 10},
{'id': 10,
 'start': 56,
 'end': 63,
 'tag': 'NOUN',
 'pos': 'NOUN',
 'morph': 'Gender=Fem|Number=Sing',
 'lemma': 'manière',
 'dep': 'obj',
 'head': 8},
{'id': 11,
 'start': 64,
 'end': 66,
 'tag': 'ADP',
 'pos': 'ADP',
 'morph': '',
 'lemma': 'de',
 'dep': 'mark',
 'head': 12},
```

```
{'id': 12,
 'start': 67,
 'end': 76,
 'tag': 'VERB',
 'pos': 'VERB',
 'morph': 'VerbForm=Inf',
 'lemma': 'retrouver',
 'dep': 'acl',
 'head': 10},
{'id': 13,
 'start': 77,
 'end': 80,
 'tag': 'DET',
 'pos': 'DET',
 'morph': 'Definite=Ind|Number=Plur|PronType=Art',
 'lemma': 'un',
 'dep': 'det',
 'head': 14},
{'id': 14,
 'start': 81,
 'end': 93,
 'tag': 'NOUN',
 'pos': 'NOUN',
 'morph': 'Gender=Fem|Number=Plur',
 'lemma': 'information',
 'dep': 'obj',
 'head': 12},
{'id': 15,
 'start': 94,
 'end': 98,
 'tag': 'ADP',
 'pos': 'ADP',
 'morph': '',
 'lemma': 'dans',
 'dep': 'case',
 'head': 17},
{'id': 16,
 'start': 99,
 'end': 101,
 'tag': 'DET',
 'pos': 'DET',
 'morph': 'Definite=Ind|Gender=Masc|Number=Sing|PronType=Art',
 'lemma': 'un',
 'dep': 'det',
 'head': 17},
{'id': 17,
 'start': 102,
 'end': 108,
 'tag': 'NOUN',
 'pos': 'NOUN',
 'morph': 'Gender=Masc|Number=Sing',
 'lemma': 'corpus',
 'dep': 'nmod',
 'head': 14},
{'id': 18,
 'start': 108,
```

```
    'end': 109,
    'tag': 'PUNCT',
    'pos': 'PUNCT',
    'morph': '',
    'lemma': '.',
    'dep': 'punct',
    'head': 6}]}
```

Réécriture du texte avec les lemmes :

In [81]:
```
[w.lemma_ for w in docs[0]]
```

Out[81]:
```
['le',
 'recherche',
 'de',
 'information',
 'être',
 'le',
 'domaine',
 'qui',
 'étudier',
 'le',
 'manière',
 'de',
 'retrouver',
 'un',
 'information',
 'dans',
 'un',
 'corpus',
 '.']
```

On peut aller plus loin en identifiant les entités nommées, par exemple.

In [82]:
```
import os
texte_rois = ""
with open(os.path.join("datasets", "rois.txt"), "r", encoding='utf8') as f:
    texte_rois = texte_rois.join(line.rstrip("\n") + " " for line in f.readlines())

# la ligne suivante permet de retirer les espaces redondantes à l'intérieur de la c
#texte_rois = " ".join(texte_rois.split())
```

In [83]:
```
doc_rois = nlp(texte_rois)
```

In [85]:
```
#Find named entities, phrases and concepts
c = 0
for entity in doc_rois.ents:
    print(entity.text, entity.label_)
    if c>50:
        break;
    c +=1
```

```
MAURICE MISC
Académie française ORG
ROIS ORG
Le Roi de fer : MISC
Maurice Druon PER
Pion et MISC
Éditions Del Duca ORG
ISBN ORG
Edmond PER
Jules de Concourt PER
PROLOGUE ORG
Philippe IV PER
Flamands LOC
Anglais LOC
Aquitaine LOC
Papauté qu’ ORG
Avignon LOC
Parlements MISC
Edouard II PER
Angleterre LOC
jusqu’ LOC
Russie LOC
n’ PER
Église MISC
Juifs MISC
Trésor LOC
s’ LOC
État LOC
la France LOC
Français MISC
Ordre souverain des chevaliers ORG
Temple MISC
Templiers MISC
Philippe le Bel PER
Histoire PER
C’ MISC
PREMIÈRE PARTIE    LA MALÉDICTION      I     MISC
Assise LOC
Angleterre LOC
Isabelle PER
d’amphore PER
Guillaume d’Aquitaine PER
D’ PER
Car je n’ MISC
n’ PER
j’ LOC
n’ PER
qu’ MISC
Ah ! MISC
n’ PER
Guillaume PER
qu’ MISC
```

In [87]:
```
list_loc = [ent.text for ent in doc_rois.ents if ent.label_=="LOC"]
list_loc[0:50]
```

Out[87]:
```
['Flamands',
 'Anglais',
 'Aquitaine',
 'Avignon',
 'Angleterre',
 'jusqu’',
 'Russie',
 'Trésor',
 's’',
 'État',
 'la France',
 'Assise',
 'Angleterre',
 'j’',
 'qu’',
 'qu’',
 'de France',
 'Angleterre',
 'qu’',
 'qu’',
 'Mortimer',
 'de France',
 'Angleterre',
 'qu’',
 's’',
 's’',
 'j’',
 'j’',
 'qu’',
 'qu’',
 'qu’',
 's’',
 's’',
 'Angleterre',
 'Artois',
 'Blanche',
 'Blanche',
 'j’',
 'j’',
 'de France',
 'croyez-le',
 'Avez',
 'Artois',
 'France',
 'Damas',
 'j’',
 'Conches',
 's’',
 'hôtel de Nesle',
 's’']
```

In [88]:
```
ensemble_lieux = set(sorted(list_loc))
print(list(ensemble_lieux))
```

['Trésor s’', 'Zaccaria', 'York', 'château de Pontoise', 'Es-tu', 'Auxerre', 'Exco
unié', 'Dourdan', 'Longwy', 'j’', 'Capétiens', 'Vannes', 'comté d’Artois', 'Albizz
i', 'Champagne', 'Turcs', 'Porto', 'croyez-le', 'archidiocèse de Sens', 'comté de Fl
andre', 'Roquemaure', 'Flandre', 'Prévôt', 'Saint-Maurice de l’an 1307', 'Chrétient
é', 'Clarisses', 'Mégarée', 'd’Aunay-lès-Bondy', 'Bohême', 'Temple', 'Roberto', 'Gât
inais', 'Que Blanche', 'Neauphle-le-Château', 'Archevêque', 'Eau', 'Avez', 'Saint-Po
l', 'Courtrai', 'Assise', 'Laon', 'mourrais', 'Ansel', 'Arabie', 'Épousa', 'Chanoin
e', 'Tarente', 'Pompadour', 'd’Occident', 'd’Orléans', 'Palais même[22].', 'Rhône',
'Magicienne', 'Pucci', 'Allemagne', 'Oderisi', 'château de Conches', 'comtesse de Ha
inaut', 'Entendez', 'Salins', 'Douvres', 'Lincoln', 'grand-peine', 'Venise', 'châtea
u de Douvres', 'Barbette', 'Frédol', 'Buonsignori', 'Santa Maria del Carmine', 'd’Is
abelle', 'Évêque', 'Templiers', 'États', 'Mouche', 'Lieux', 'Monte-Falcone', 'Carpen
tras', 'Orsini', 'Boiteux', 'diocèse de Paris', 'Nogaret', 'sur-le-champ', 'Tors-Co
l', 'Milan', 'Debout', 'Metz', 'Terre sainte', 'Irlande', 'Prudence', 'Westminster',
'Hamelin', 'Haute-Saône', 'Lorris', 'rue Mazarine', 'Tunis', 'forêt de Pont-Sainte-M
axence', 'Guigues', 'Levant', 'chambellan', 'évêché de Pamiers', 'Maisons', 'Franc
e', 'Lyon', 'Lombards', 'Arles', 'Russie', 'Moyen', 'Bourgognes', 'Saint-Grégoire',
'Essonne', 'Orléans', 'Plainville-en-Vexin', 'Courtille', 'forêt des Ardennes', 'pré
vôt de Paris', 'Italiens', 'Château-Gaillard', 'Bretonnerie', 'Angoulême', 'Châteaun
euf', 'Jérusalem', 'Guyenne', 'Hongrie', 'Principal', 'Le Portier', 'comté n’', 'rue
des', 'Gautier d’Aunay', 'Bourdonnais', 'Inquisition', 'Italie', 'État', 'Brie', 'Ch
ambre des Communes', 'Pontoise', 'Reims', 'Duèze', 'jusqu’', 'selle', 'Espagne', 'Ch
er', 'voglio', 'Gardes', 'Colonna', 'Sainte-Marie Majeure', 'Constantinople', 'Mauri
enne', 'Guccio', 'dentelles de Malines', 'Monnaie', 'notre Sainte-Mère l’Église', 'C
lichy', 'Rome', 'Béarn', 'Berkeley', 'Pampelune', 'Mâtin', 'collège Mazarin', 'ROYA
ME ', 'Got', 'Holà', 'Grèce', 'Viennois', 'Béarnais', 'Les', 'Éclairez', 'Clign
court', 'Aude', 'Derrière', 'Reynolds', 'Suisse', 'Italien', 'la manche', 'Cordoue',
'Anglais', 'Portier de Marigny', 'Grez', 'Herbes', 'quartier Saint-Eustache', 'Co
enay', 'pont Notre-Dame', 'd’Harcourt', 'Bourgogne', 'Nesle', 'répondez', 'Fontai
leau', 'Mauldre', 'Stephen', 'port de', 'Curieuse', 'château de Maubuisson', 'Portug
al', 'Rassurez Marguerite', 'porte de Marguerite', 'Grand-Moulin', 'Montpellier', 'C
ahors', 'Blanche', 'comte de Beaumont-le-Roger', 'Louvre', 'ammazzalo[24', 'Grand Ne
sle', 'Baglioni', 'marchand Albizzi', 'Vélines', 'Langres', 'Violante', 'Vexé', 'Bri
ançon', 'Narbonne', 'île de la Cité', 'Arras', 'château de Clermont', 'Montfort-l’Am
aury', 'Perche', 'comte de Flandre', 'Inhumé', 'Paris', 'de France', 'Sussex', 'Hire
çon', 'Parloir', 'collège de Presles', 'Neauphle-le-Vieux', 'Vénétie', 'Londoniens',
'j’ai accepté', 'Pareilles', 'Devinez', 'Bretagne', 'Sainte-Chapelle', 'Hardi', 'Giv
ry', 'Béziers', 'Mont-Martre', 'Moucy-le-Neuf', 'Philippine de Luxembourg', 'Artoi
s', 'Quels', 'Égypte ancienne', 'Hélas', 'm’', 'Écoutez', 'Normandie', 'Pergame', 'E
vrard', 'Pontife romain', 'Ursins', 'Rouen', 'Santa Maria della Scala', 'Bourbourg',
'Vermandois', 'Windsor', 'Boulogne', 'Bel', 'Chartres', 'roi de France', 'Études',
'Petit-Pré-aux-Clercs', 'Peruzzi', 'Bourdenai', 'Université', 'Trésor', 'Portefrui
t', 'Castille', 'plaine blanche', 'Gênes', 'Luxembourg', 'Chirk', 'Prato', 'Cherchem
ont', 'Loos', 'Poitiers', 'Franche-Comté', 'Oxford', 'commission de l’Église', 'Bocc
anegra', 'Sens', 'pays de Liège', 'tour de Nesle', 'Fontaines', 'Saint-Marcel', 'cha
peron blanc', 'Limoges', 'd’Inquisition', 'Gironde', 'Comté-Franche', 'Saint-Miche
l', 'château de Hertford', 'Sienne', 'Recueillit', 'Romain', 'Londres', 'Fiennes',
'Dauphiné', 'Parisiens', 'Chypre', 'Toulouse', 'Lorraine', 'Butors', 'Latium', 'Roya
ume du Maroc', 'roi de Navarre', 'Jusque', 'Hertfordshire', 'Regardez', 'Sicile', 'N
otre', 'Vieux', 'Évêque de Béziers', 'Gautier d’', 'Tribunal', 'pont de Londres', 'P
oitou', 'Biche', 'Albi', 'd’Aunay', 'Campanie', 'prévôt', 'Chambre des Comptes', 'Ac
ceptez', 'Rundingen', 'Khan de Perse', 'Flamands', 'Clermont', 'hôtel de Marigny',
'Gand', 'Auch', 'Douai', 'Comminges', 'Ployebouche', 'Damas', 'Lyons-la-Forêt', 'Vém
ars', 'Morte', 'Bigorre', 'Appelle', 'Combien', 'Beauvais', 'pont Saint-Michel', 'Pé
rigord', 'Vaumain', 'Monnaie de Paris', 'Cité', 'Blancs-Manteaux', 'Valence', 'Piace
nza', 'Sénéchal', 'Diable', 'Corbeil', 'Etienne', 'Pair', 'Toscane', 'Tour', 'Romagn

```
e', 'd’Aragon', 'Boccacio da Chellino', 'la Manche', 'Picardie', 'Blois', 'Bordeau
x', 'château qu’', 'Forgerons', 'Galerie', 'Couchés', 'Nivernais', 'Norfolk', 'la Se
ine', 'Noyon', 'Souffrez', 'Palais', 'royaume de Sicile', 'Galerie marchande[8].  ',
'Winchester', 'hôtel de Nesle n’', 'royaume de France', 'Tolomei', 'Shrewsbury', 'Lo
uve de France', 'royaume de Naples', 'Blême', 'Participa', 'roi de Sicile', 'Longuev
ille', 'Longchamp', 'Voilà', 'Aquitaine', 'Archevêque de Sens', 'Saint-Sauveur-le-Vi
comte', 'Santa Maria dei Servi', 'palais de la Cité les bourgeois de Paris', 'Archev
êque de Rouen', 'Ajoutez', 'Nulle', 'Tiens', 's’étira', 'Avesnes', 'Dubois', 'Chaude
s', 'Rigny', 'vienne', 'Blessé', 'Tamise', 'roi de Naples', 'Hôtel de Ville de Pari
s', 'Oise', 'Angevins', 'cathédrale n’', 'Dames', 'Notre-Seigneur', 'Ouais', 'Écossa
is', 'Châlons', 'Harcourt', 'Caumont', 'Notre-Dame de Paris', 'la Cerise', 'Alips de
Mons', 'None', 'dauphin de Viennois', 'Aquitains', 'Jura', 'charrois', 'Boulogne-sur
-Mer', 'nacre', 'Flandres', 'Dijon', 'Rendez', 'Renversé', 'Clare', 'Saint-Germain-d
es-Prés', 'jardin du Palais', 'Saint-Front de Périgueux', 'Outre-mer', 'Grand', 'Pet
it Nesle', 'la Bretagne', 'Saint-Siège', 'Savez', 'Anseau', 'Notre-Dame s’', 'Cource
lles-la-Garenne', 'Tyburn', 'Sois', 'comte de Hainaut', 'Quelle', 'Arno', 'Westmouti
ers', 'Orleton', 'Anjou', 'Burghersh', 'Dauphin', 'Dreux', 'Pré-aux-Clercs', 'châtea
u de Langley', 'Beaune', 'Messer', 'Mirepoix', 'haut-de-chausses', 'Maillard', 'Comp
agnonnage', 'Signor', 'château de Gournay', 'Jeune', 'Arbois', 'Cornouailles', 'port
e de Buci', 'Villandraut', 'Aunay', 'Institut', 'Emporte', 'bru Blanche', 'A', 'Sarl
at', 'suppôts du Diable', 'Couvin', 'Worcester', 'rue Mauconseil', 'Provence', 'Reto
urnez', 'Montdidier', 's’impatienter', 'Boccacio', 'Martroy', 'Santa Maria delle Nev
i', 'Noyers', 'Gaucelin', 'Broderies', 'Montfaucon', 'd’Espagne', 'Nièce', 'Saint-De
nis', 'Porcien', 'Gibelins', 'Beaugency', 'Wigmore', 'Français', 'Mesnil', 'Plantage
nets', 'Chester', 'Vénitiens', 'Sire', 'diocèse de Toulouse', 'Asie', 'Angleterre',
'Saint-Eustache', 'Hereford', 'Insultes', 'Guines', 'Voyez', 'Ponte Vecchio', 'com
de Bourgogne', 'Languedoc', 'Todi', 'château de Westminster', 'L’', 'Oui', 'Mornay',
'Seine', 'Boiteuse', 'Cressay', 'Mauvais', 'Bénédictin', 'Perpignan', 'Roncevill
'Outre-manche', 'Europe', 'Finissons', 'place de Grève', 'Déramé', 'Châtelet', 'L
rches', 'd’', 'Lizines', 'Pharaon', 'Bouvines', 'Fanatique', 'abbaye de Maubuisson',
'rue Saint-Merri', 'Spolète', 'Saint-Merry', 'tour de Londres', 'Byzance', 'comtesse
de Nevers', 'Saint-André', 'le Salut', 'Saint-Jean-d’Acre', 'Sceaux', 'Maine', 'Font
froide', 'palais Caëtani', 'Allerani', 'Naples', 'Siennois', 'In', 'Chartreux', 'd’É
vreux', 'Nevers', 'Périgueux', 'Sully', 'Aragon', 'Maubuisson', 'Nogent-le-Roi', 'Gr
ands Conseils', 's’', 'Le soleil', 'Hollande', 'Majorque', 'Marigny', 'comté de Valo
is', 'la France', 'manche', 'royaume latin de', 'Montrons', 'Poissy', 'Enchaîné', 'I
le-de-France', 'ROYAUME  Pendant', 'la Franche-Comté', 's’éteignit', 'Comtesse-pai
r', 'Souastre', 'Palais n’', 'Vincennes', 'Avignon', 'Alençon', 'Bristol', 'Toscan',
'Bourgogne-Comté', 'qu’', 'Hainaut', 'Dole', 'Saintonge', 'Namur', 'Asnières', 'comt
esse de', 'Calais', 'Bruges', 'Vois-tu', 'Bardi', 'Palais de la Cité', 'Salimbene',
'Marche', 'Florence', 'l’', 'Morts', 'Brésil', 'Coutances', 'Notre-Dame de Boulogn
e', 'Occident', 'Leicester', 'Lyonnais', 'Notre-Dame', 'Palestine', 'prévôté de Mont
fort-l’Amaury', 'Navarre', 'Pays de Galles', 'Amiens', 'Prayères', 'Savoie', 'Béthun
e', 'Brienne', 'Mortimer', 'l’Italie', 'Beaumont', 'Senlis', 'Vandouvre', 'Bar', 'Es
-tu bien', 'Toscans', 'Conflans', 'Marguerite', 'répondrez', 'abbé de Chancelade',
'Épouse', 'Bourgogne-Duché', 'Limousin', 'Zélande', 'Prends', 'Neauphle', 'comté de
Beaumont-le-Roger', 'Beaucaire', 'la Saint-Hugues', 'Afrique', 'couenne', 'Mons-en-P
évèle', 'Mettez', 'Orthez', 'hargne', 'vilenie', 'Bouville', 'Anagni', 'hôtel de Nes
le', 'tour du Louvre', 'Coglione', 'Tabernacle', 'Conches', 'Francophonie', 'Gascogn
e', 'Saint Empire', 'archevêque', 'Génois', 'mailles', 'comté de Guines', 'Molay',
'Hirson', 'Charnay', 'Augustins', 'château de Berkeley']
```

Une alternative est constituée par la librairie spacy, cf.: https://spacy.io/

Quelques autre prétraitements

Autre prétraitement utile : remplacer certains motifs trouvés dans les textes.

In [63]:
```
print(X)
X_rep = [x.replace("is", "are") for x in X]
print(X_rep)
```

```
['Here is the adventures of an adventurous adventurer']
['Here are the adventures of an adventurous adventurer']
```

In [64]:
```
re.sub("[s|S]+\w*", "truc", X_rep[0])
```

Out[64]:
```
'Here are the adventuretruc of an adventuroutruc adventurer'
```

Corriger automatiquement l'orthographe des mots grâce à la distance d'édition (ou distance Levenshtein, proposée en 1965). Il s'agit de calculer le plus petit nombre d'opérations (insertion, suppression, remplacement) pour passer d'une chaîne à une autre.

In [65]:
```
# one possible implementation from https://en.wikibooks.org/wiki/Algorithm_Imple
def levenshtein(s1, s2):
    if len(s1) < len(s2):
        return levenshtein(s2, s1)

    # len(s1) >= len(s2)
    if len(s2) == 0:
        return len(s1)

    previous_row = range(len(s2) + 1)
    for i, c1 in enumerate(s1):
        current_row = [i + 1]
        for j, c2 in enumerate(s2):
            insertions = previous_row[j + 1] + 1 # j+1 instead of j since previous_
            deletions = current_row[j] + 1       # than s2
            substitutions = previous_row[j] + (c1 != c2)
            current_row.append(min(insertions, deletions, substitutions))
        previous_row = current_row

    return previous_row[-1]
```

In [66]:
```
#levenshtein("chaîne", "chaine")
#levenshtein("chaîne", "chiane")
levenshtein("remerciement", "remerciements")
```

Out[66]:
```
1
```

In [ ]:
```

```