TP2

IFT2015

Structures de données

automne2023

Dans ce TP, vous implémenterez deux classes de Map différentes pour servir les fonctions de traitement du langage naturel (NLP). En particulier, vous constituerez des ensembles de données de nombreux documents. Vous utiliserez la première `Map` pour stocker les mots de ces documents (Word `Map`). Les clés pour la Map de mots seront les mots eux-mêmes, et les valeurs seront des références au deuxième type de `Map` (File `Maps`), qui conservent les occurrences de mots dans les fichiers. Les clés pour les Map de fichiers seront les noms de fichiers, et les valeurs seront une `List` des positions des mots associés dans les fichiers correspondants.

Considérez l'ensemble de données suivant qui contient trois documents (PS : un ensemble de données réel contiendra beaucoup plus de fichiers et des textes plus longs). Les fichiers d'un ensemble de données sont stockés dans un répertoire.

`26.txt:`

```
With stunning photos and stories, National Geographic Explorer
Wade Davis celebrates the extraordinary diversity of the world's
indigenous cultures, which are disappearing from the planet at
an alarming rate.
```

`48.txt:`

```
Photographer Phil Borges shows rarely seen images of people from
the mountains of Dharamsala, India, and the jungles of the
Ecuadorean Amazon. In documenting these endangered cultures, he
intends to help preserve them.
```

`165.txt:`

```
In this deceptively casual talk, Charles Leadbeater weaves a
tight argument that innovation isn't just for professionals
anymore. Passionate amateurs, using new tools, are creating
products and paradigms that companies can’t.
```

La Figure 1 montre une fraction de la structure de données pour cet exemple.

- Avant de construire la structure de données, vous devrez prétraiter le texte en remplaçant tous les signes de ponctuation par des espaces, en remplaçant plusieurs espaces par un seul, et en effectuant un traitement de texte NLP (tokenization, POS, et lemmatization du texte) comme décrit dans les instructions.
- Afin de créer votre Word `Map` et File `Maps` , vous devez implémenter l'interface `Map` en remplaçant les fonctions nécessaires. Vous êtes autorisé(e) à définir des fonctions supplémentaires selon vos besoins.

- Vous pouvez utiliser la méthode `hashcode` () de la classe Object, ou la redéfinir.

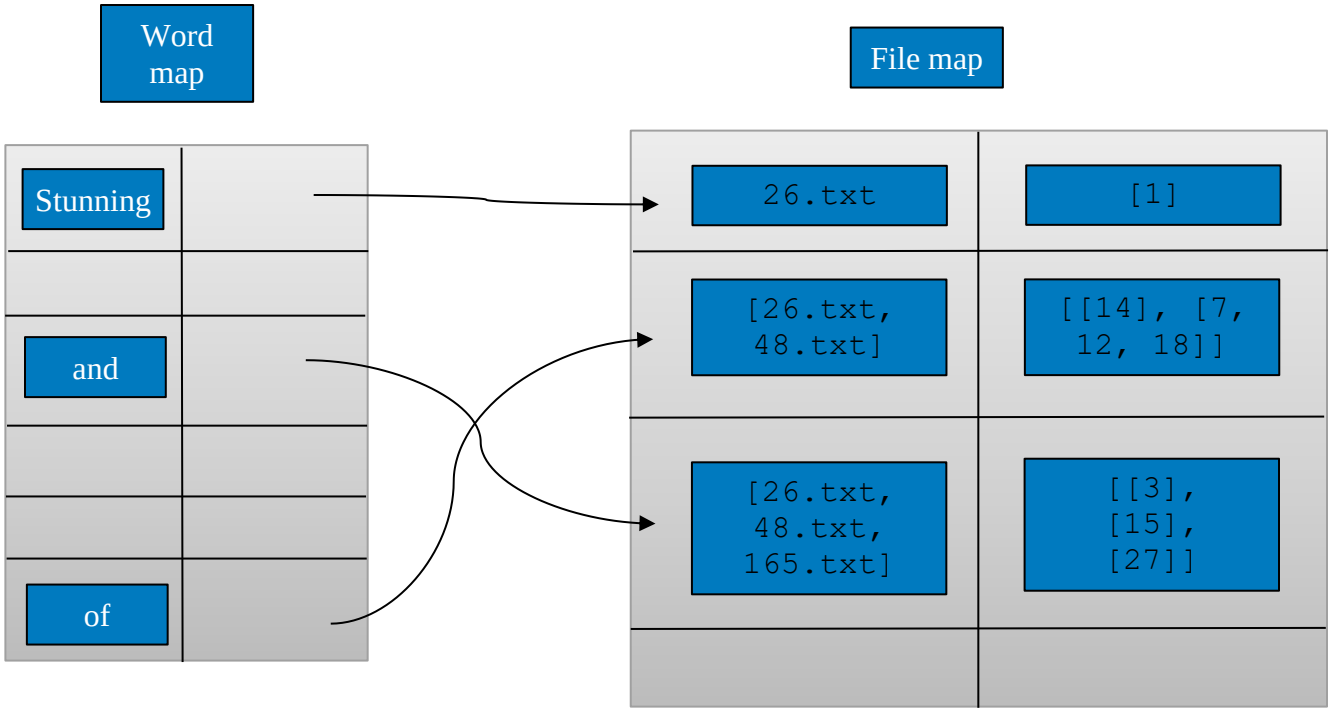


Figure1

- Vous devez implémenter la gestion du facteur de charge de sorte que si l'insertion d'un mot entraîne un facteur de charge supérieur à 0.75, vous devez redimensionner la capacité de la Word Map à 2 fois la taille précédente + 1.

Une fois votre structure de données terminée, vous implémenterez deux opérations essentielles utilisées en NLP :

1. Proposer le mot suivant le plus probable, comme une fonction d'auto-complétion liée au concept de **bi-grammes** en traitement du langage naturel (NLP).
2. Trouver le document le plus pertinent pour une recherche de requête (qui peut être composée de plusieurs mots), en lien avec le concept de **TF-IDF** (fréquence des termes–fréquence inverse des documents) en traitement du langage naturel (NLP).

Bigrammes

Un bigramme est un segment de texte composé de deux mots consécutifs qui se produisent au moins une fois dans un texte donné. Les bigrammes sont des moyens très informatifs pour démontrer les associations sémantiques entre les mots. À titre d'exemple, les bigrammes dans le texte suivant : "*Photographer Phil Borges shows rarely-seen images of people from the mountains of Dharamsala, India*" sont:

Photographer Phil

Phil Borges

Borges shows

shows rarely

rarely seen

seen images

...

Dharamsala, India

Les bigrammes peuvent être utilisés pour suggérer le mot suivant le plus probable qui peut apparaître après un mot tapé dans un moteur de recherche, et pour améliorer la prédiction d'un système d'auto-complétion.

Considérez les sept textes suivants :

1. Thank you so much for your help.
2. I **really** appreciate your help.
3. I **really** like what you said.
4. I apologize for coming over unannounced like this.
5. Sorry, do you know what time it is?
6. I'm **really** sorry for not inviting you.
7. I **really** like your watch.

Supposons qu'après l'entraînement, notre modèle apprend les occurrences de chaque paire de mots pour déterminer le plus probable, W_2, après l'autre, W_1. Par exemple, à partir des phrases 2, 3, 6 et 7, après le mot "**really**", les mots "appreciate", "like" ou "sorry" apparaissent.

Calculons la probabilité d'observer le deuxième mot, W_2, se produisant après le mot W_1. Nous appliquons la relation de la probabilité conditionnelle:

$$P(W_2 | W_1) = C(W_1, W_2) / C(W_1)$$

La probabilité du mot `W2` étant donné le mot précédent `W1`, $P(W_2 | W_1)$, équivaut au nombre de leur bigramme, ou à la co-occurrence des deux mots, $C(W_1, W_2)$, divisé par le nombre de fois où W_1apparaît, $C(W_1)$.

Dans notre exemple, trouvons le mot qui a la probabilité la plus élevée d'apparaître après le mot "**really**". Nous avons besoin de C("**really**") = 4, et de C("**really**", "appreciate") = 1, C("**really**", "like") = 2, et C("**really**", "sorry") = 1. Le mot le plus probable pour suivre "**really**" est "like", avec P("like" | "**really**") = 0.5, C("really", "**like**") / C("**really**") = 2/4 = 0.5. Les probabilités P("appreciate" | "**really**") = P("sorry" | "**really**") = 0.25.

À la requête "*the most probable bigram of really*", votre programme, compte tenu de l'ensemble de données ci-dessus, devrait suggérer le mot suivant le plus probable qui peut apparaître après le mot "**really**", ou le bigramme le plus probable de "**really**", qui est le mot "like". S'il y a

deux mots ou plus avec la même probabilité, retournez le plus petit mot selon l'ordre lexicographique.

TFIDF (fréquence des termes–fréquence inverse des documents)

Il s'agit d'un score qui reflète l'importance d'un mot dans un document. En traitement du langage naturel (NLP), un mot est considéré comme efficace pour la catégorisation d'un fichier s'il apparaît fréquemment dans ce fichier, mais a très peu d'occurrences dans les autres textes de l'ensemble de données. Pour calculer le TF-IDF, nous commençons par calculer la fréquence du mot (terme) :

$$TF_d(w) = count(w)/totalW$$

où `count(w)` est le nombre de fois où le mot w apparaît dans un document, et `totalW` est le nombre total de mots dans le document (longueur en termes de mots). Ensuite, la fréquence inverse des documents est calculée pour diminuer le poids des mots fréquents également présents dans d'autres fichiers, tout en amplifiant ceux qui sont rares.:

$$IDF(w) = 1 + \ln((1+totalD)/(1+count(d,w)))$$

où `totalD` est le nombre total de documents considérés, et `count(d, w)` est le nombre de documents contenant le mot `w`. Enfin, le TF-IDF est calculé par:

$$TFIDF_d(w) = TF_d(w) \times IDF(w)$$

Maintenant, pour chaque mot de la requête, vous devez additionner les $TFIDF_d(w_i)$ calculés pour ce mot dans tous les documents, ce qui vous donne un score pour chaque document.

$$Score_d = \sum_{w \in query} TFIDFd(w)$$

Imaginez un moteur de recherche qui utilise le système de notation décrit ci-dessus pour classer les documents en fonction d'une requête fournie. Vous recevrez une requête et vous devrez proposer le document le plus pertinent dans l'ensemble de données qui se classe en premier en fonction de cette requête. *S'il y a deux documents ou plus avec le même `score`, retournez celui avec le nom de document le plus petit selon l'ordre lexicographique*.

Considérez l'ensemble de données suivant composé de quatre documents :

`900.txt:`

```
The discovery of other solar system wanderers rivaling Pluto in
size suddenly had scientists asking what wasn't a planet. They
put their heads together in 2006 and came up with three
conditions for planethood: A planet must orbit the sun, be large
enough so that its own gravity molds it into a spherical shape,
and it must have an orbit free of other small objects.
Unfortunately for Pluto, our one time ninth planet failed to
meet the third condition.
```

`901.txt:`

```
The largest known small bodies, in the conventional sense, are
several icy Kuiper belt objects found orbiting the Sun beyond
the orbit of Neptune. Ceres which is the largest main belt
```

asteroid and is now considered a dwarf planet is roughly 950 km (590 miles) in diameter.

902.txt:

This article is about the astronomical object. For other uses, see Planet (disambiguation). A planet is a large, rounded astronomical body that is neither a star nor its remnant. The best available theory of planet formation is the nebular hypothesis, which posits that an interstellar cloud collapses out of a nebula to create a young protostar orbited by a protoplanetary disk. Planets grow in this disk by the gradual accumulation of material driven by gravity, a process called accretion.

903.txt:

The collapse of International Coffee Organization, ICO, talks on export quotas yesterday removes the immediate need to reinstate U.S. legislation allowing the customs service to monitor coffee imports, analysts here said. The Reagan administration proposed in trade legislation offered Congress last month that authority to monitor coffee imports be resumed. That authority lapsed in September 1986. A bill also was introduced by Rep. Frank Guarini (DN.J.) However, the failure of the ICO talks in London to reach agreement on export quotas means the U.S. legislation is not immediately needed, one analyst said. Earlier supporters of the coffee bill hoped it could be passed by Congress quickly. "You're going to have a hard time convincing Congress (now) this is an urgent issue," the coffee analyst said.

Les documents après traitement ressemblent à ce qui suit:

900.txt:

the discovery of other solar system wanderer rival Pluto in size suddenly have scientist ask what be not a planet they put they head together in 2006 and come up with three condition for planethood a planet must orbit the sun be large enough so that its own gravity mold it into a spherical shape and it must have a orbit free of other small object unfortunately for Pluto we one time ninth planet fail to meet the third condition

901.txt:

the large know small body in the conventional sense be several icy Kuiper belt object find orbit the Sun beyond the orbit of Neptune Ceres which be the large main belt asteroid and be now consider a dwarf planet be roughly 950 km 590 mile in diameter

902.txt:

this article be about the astronomical object for other use see Planet disambiguation a planet be a large rounded astronomical body that be neither a star nor its remnant the good available theory of planet formation be the nebular hypothesis which posit that a interstellar cloud collapse out of a nebula to create a young protostar orbit by a protoplanetary disk planet grow in this disk by the gradual accumulation of material drive by gravity a process call accretion

903.txt:

```
the collapse of International Coffee
Organization ICO talk on export quota yesterday remove the immediate need to
reinstate U S legislation allow the custom service to monitor coffee import
analyst here say the Reagan administration propose in trade legislation offer
Congress last month that authority to monitor coffee import be resume that
authority lapse in September 1986 a bill also be introduce by Rep Frank
Guarini DN J however the failure of the ICO talk in London to reach agreement
on export quota mean the U S legislation be not immediately need one analyst
say early supporter of the coffee bill hope it could be pass by Congress
quickly you be go to have a hard time convince Congress now this be a urgent
issue the coffee analyst say
```

Imaginez un navigateur demandant de trouver le document le plus pertinent pour le mot "**planet**" : `recherche planet`. Vous devrez calculer le TF-IDF du mot "**planet**" dans chaque document de l'ensemble de données.:

TFIDF("**planet**") dans le document 900 = TF("**planet**")xIDF("**planet**") = 0.045

TF("**planet**") = 3/80

IDF("**planet**") = 1+ln((1+4)/(1+3))

TFIDF("**planet**") dans le document 901 = TF("**planet**")xIDF("**planet**") = 0.026

TF("**planet**") = 1/47

IDF("**planet**") = 1+ln((1+4)/(1+3))

TFIDF("**planet**") dans le document 902 = TF("**planet**")xIDF("**planet**") = 0.046

TF("**planet**") = 3/79

IDF("**planet**") = 1+ln((1+4)/(1+3))

TFIDF("**planet**") dans le document 903 = TF("**planet**")xIDF("**planet**") = 0.0

TF("**planet**") = 0

IDF("**planet**") = ln(4/3)

Par conséquent, le document le plus pertinent concernant le mot "planet" est le document `902` avec le score TF-IDF optimal de 0.046.

Modifier la requête:

Il existe également un problème de correction des erreurs d'orthographe dans les requêtes. Par exemple, nous pouvons souhaiter récupérer des documents contenant le terme "carrot" lorsque l'utilisateur tape la requête "carot".

Donc, avant d'effectuer l'une des deux requêtes ci-dessus, vous devriez d'abord effectuer une correction d'orthographe sur chaque mot de la requête. Pour cette requête : ***the most probable bigram of <word>***, vous devez effectuer la requête sur le mot corrigé après correction d'orthographe. Et pour cette requête **search *<word1><space><word2><space><word3>…***, vous devez d'abord effectuer une correction d'orthographe sur chaque <word> et les remplacer par leur version corrigée, puis effectuer la requête donnée.

Pour corriger les erreurs d'orthographe, vous devez implémenter la fonction suivante. La **distance de Levenshtein**, également appelée **distance d'édition**, entre deux mots est le nombre minimum d'opérations sur un seul caractère nécessaires pour transformer une chaîne en une autre.

Typiquement, trois types d'opérations sont effectués (une à la fois) :

- Remplacer un caractère
- Supprimer un caractère
- Insérer un caractère

Exemple:

```
Input: str1 = "glomax",str2 = "Folmax"
Output: 3

La chaîne str1 est transformée en str2 en remplaçant 'g' par 'o', en
supprimant le deuxième 'o' et en insérant 'f' au début. Il n'y a aucun moyen
de le faire avec moins de trois modifications.
```

Pour chaque mot de la requête donnée, vous devez calculer sa distance parmi tous les mots de tous les documents que vous avez déjà prétraités, puis sélectionner celui avec la distance minimale (s'il y a plusieurs mots avec la distance minimale, choisissez celui qui est alphabétiquement plus petit).

Ce que votre programme devrait faire

Votre programme lira un fichier d'entrée composé de plusieurs requêtes (voir ci-dessous). Vous n'avez pas besoin de prétraiter les mots de la requête comme vous l'avez fait pour les documents. Vous écrirez vos réponses, ligne par ligne, dans un fichier appelé `solution.txt`. Il existe deux types de requêtes. L'une vise à suggérer le mot suivant le plus probable qui apparaît après un mot donné :

the most probable bigram of <word>

Le deuxième type vise à récupérer le document le plus pertinent pour une requête donnée :

search <word1><space><word2><space><word3>…

Entrée (deux noms de fichiers)

1. Le nom du répertoire du jeu de données, qui contient les documents (tous en anglais). Ce répertoire sera stocké au niveau de votre projet `Java`
2. Le nom du fichier de requêtes, où chaque ligne demande l'un des deux types de requêtes mentionnés ci-dessus. Notez que le fichier de requêtes sera également stocké au niveau de votre projet `Java`.

Il n'est pas nécessaire d'utiliser la ligne de commande dans le terminal pour exécuter votre code. Vous devez simplement fournir manuellement les arguments "`query.txt`" et "`dataset`" à votre programme au lieu d'utiliser le terminal pour fournir ces arguments.

Sortie (exemple complet)

Vous devez effectuer les requêtes et écrire les réponses, ligne par ligne, dans le fichier `solution.txt`. En supposant le répertoire dataset et le fichier de requêtes `query.txt`:

```
search plonet
the most probable bigram of cofee
search coffe importe
search graveety
the most probable bigram of dwarf
```

votre programme produira en sortie dans `solution.txt`:

```
902.txt
coffee import
903.txt
902.txt
dwarf planet
```

Code Java et soumission

- Vous pouvez former des équipes de deux programmeurs au maximum.
- Vous devez suivre les règles de la POO ; utilisez `CamelCase` pour vos identificateurs ; et commentez votre code.
- Tous les fichiers de code `Java` et les fichiers d'entrée doivent être dans le même répertoire : pas de packages. La classe qui contient la fonction principale doit s'appeler `Main.java`.
- Soumettez votre répertoire sous forme d'une archive, seuls les formats TP2_f1_f2.tgz ou TP2_f1_f2.zip sont acceptables, où f1 et f2 sont les noms de famille du premier et du deuxième membre de l'équipe, ou si vous êtes seul, utilisez TP2_f.tgz ou TP2_f.zip, où f est votre nom de famille.

- Vous avez jusqu'au samedi 16 décembre à 23h59 pour soumettre votre solution sur `StudiUM`.
- Une pénalité de 20 % par jour, à partir du 10 décembre à oohoo, sera appliquée si vous soumettez après la date limite de soumission.
- Vous devez suivre le format de sortie exact avec une réponse par ligne dans le fichier `solution.txt`, correspondant à chaque ligne dans le fichier de requêtes.

Evaluation

- **Compilation** (10%): Le code doit compiler avec succès pour produire un code exécutable ; l'échec entraînera une note de zéro pour ce critère.
- **Exécution sur les ensembles de données fournis** (10%) :

  Ne plante pas : 5%

  Produit des résultats corrects : 5%

  Ce critère évalue la stabilité du programme et sa précision sur des ensembles de données connus.
  - **Exécution sur des ensembles de données inconnus** (60%) :
    - Ne plante pas : 10%
    - Produit des résultats corrects : 50%
    - Ce critère évalue la robustesse et la précision du programme dans le traitement d'ensembles de données imprévus ou nouveaux.
  - **Qualité du code** (20%):
    - Lisibilité et commentaires : 10%
    - Structure orientée objet : 10%
    - Ce critère évalue la propreté globale, la structure et la documentation du code.

NB. N'oubliez pas d'inclure vos noms et numéros dans toutes vos classes Java.

Questions. Si vous avez des questions, utilisez le forum de TP2 sur StudiUM ou contactez l'un des assistants ou le professeur.

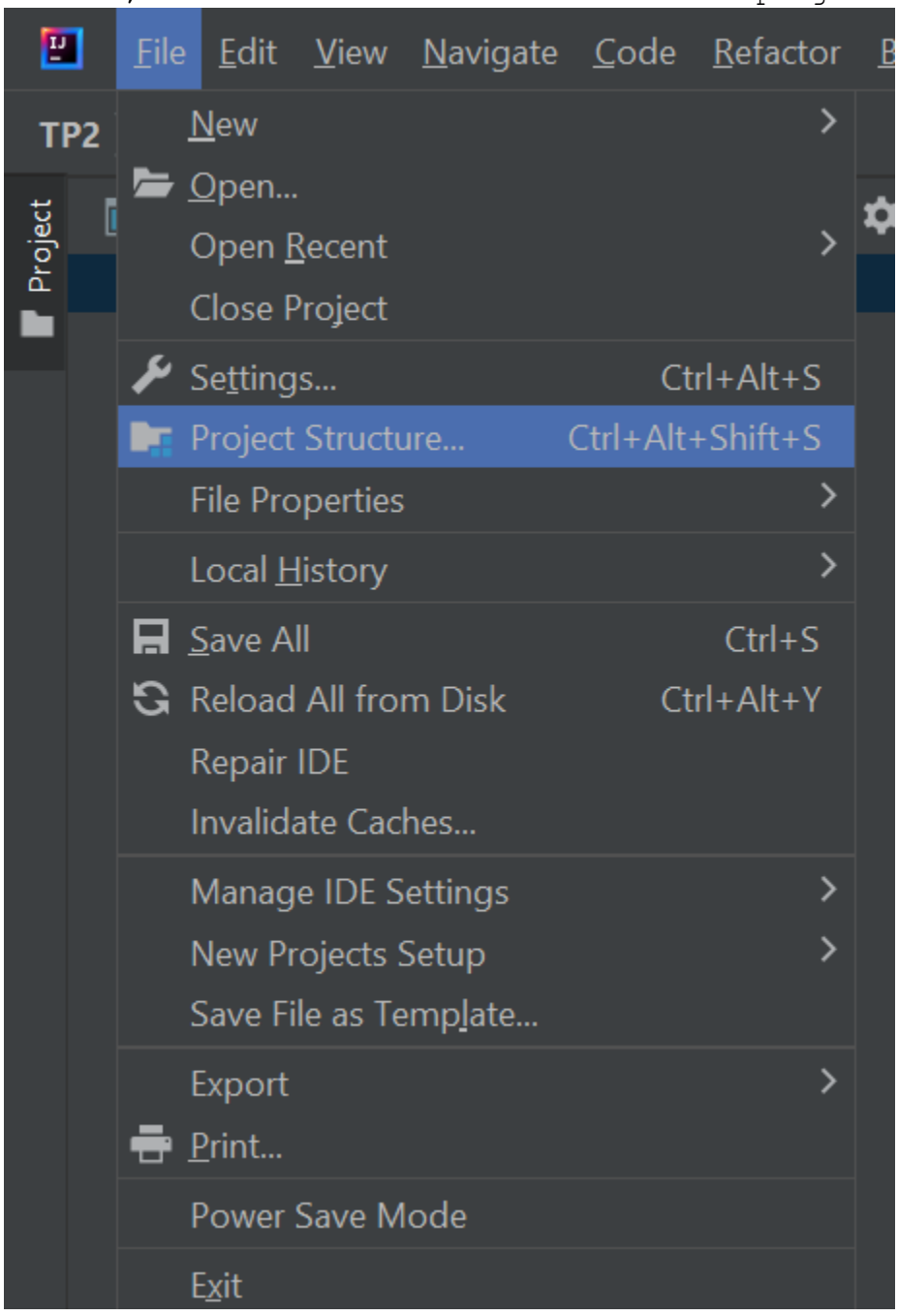
AMUSEZ-VOUS ET BONNE CHANCE !

Instructions pour ajouter et utiliser CORENLP

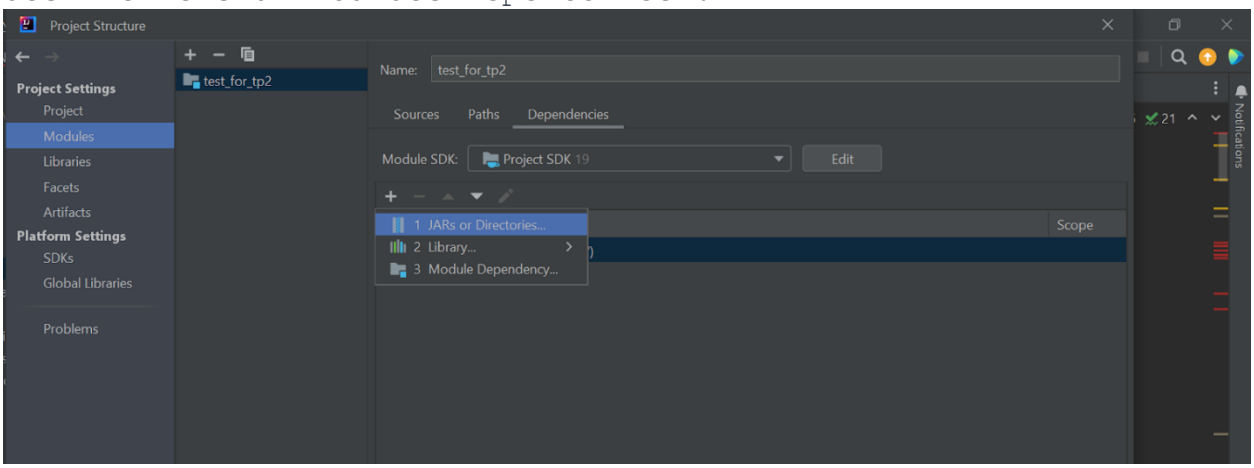
Téléchargez d'abord ce fichier zip, puis extrayez-le https://downloads.cs.stanford.edu/nlp/software/stanford-corenlp-4.5.1.zip

Suivez les étapes ci-dessous pour utiliser ce module en tant que partie du traitement de texte de votre code :

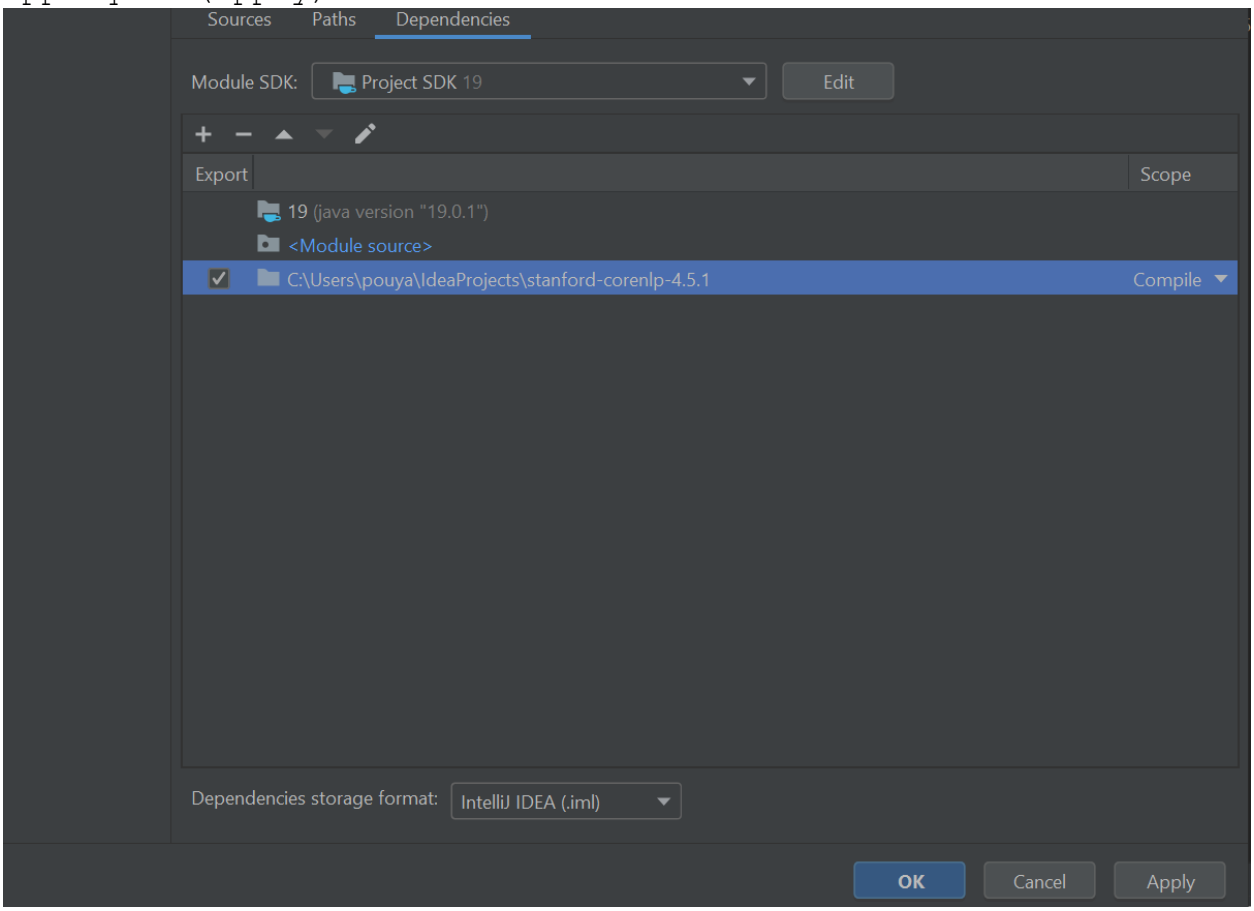
1. Tout d'abord, accédez à la structure de votre projet :



2. Dans la section du module, cliquez sur l'icône + pour ajouter des fichiers JAR ou des répertoires :



3. Ensuite, choisissez le dossier stanford-corenlp-4.5.1 dans lequel vous avez extrait les fichiers au début, puis cliquez sur Appliquer (Apply) et OK en bas.



Maintenant, vous êtes prêt à utiliser les fonctions et méthodes intégrées de ce module dans votre projet Java pour effectuer le traitement de texte.

Comment?

Ajoutez ces trois lignes dans le fichier Java que vous allez utiliser à partir de ce module pour importer les fichiers nécessaires:

```
import edu.stanford.nlp.ling.*;
import edu.stanford.nlp.pipeline.*;
import java.util.Properties;
```

Voici un exemple simple de la manière dont vous êtes censé utiliser ce module sur vos documents, qui imprime à la fin comment chaque mot est traité. Vous devriez suivre le même processus pour chaque ligne de chacun de vos documents avant de calculer quoi que ce soit.

```
import edu.stanford.nlp.ling.*;
import edu.stanford.nlp.pipeline.*;
import java.util.Properties;

public class Main {
   public static String text = "Joe Smith wasn't born in
California. " +
           "In 2017, he went to his car, sister's car Paris,
France in the summer. " +
           "His flight left at 3:00pm on July 10th, 2017. " +
           "After eating some escargot for the first time, Joe
said, \"That was delicious!\" " +
           "He sent a postcard to his sister Jane Smith. " +
           "After hearing about Joe's trip, Jane decided she
might go to France one day.";

   public static void main(String[] args) {
       // set up pipeline properties
       Properties props = new Properties();
       // set the list of annotators to run
       props.setProperty("annotators", "tokenize,pos,lemma");
       // set a property for an annotator, in this case the
coref annotator is being set to use the neural algorithm
       props.setProperty("coref.algorithm", "neural");
       // build pipeline
       StanfordCoreNLP pipeline = new StanfordCoreNLP(props);
       // create a document object
       CoreDocument document = new CoreDocument(text);
       // annnotate the document
```

```
        pipeline.annotate(document);
        //System.out.println(document.tokens());
        for (CoreLabel tok : document.tokens()) {
            System.out.println(String.format("%s\t%s",
tok.word(), tok.lemma()));
        }
    }
}
```

Sortie:

```
Joe  Joe
Smith     Smith
was  be
n't  not
born bear
in   in
California     California
.    .
In   in
2017 2017
,    ,
he   he
went go
to   to
his  he
car  car
,    ,
sister    sister
's   's
car  car
Paris     Paris
,    ,
France    France
in   in
the  the
summer    summer
.    .
His  he
flight    flight
left leave
at   at
3:00 3:00
pm   pm
on   on
July July
```

10th 10th
, ,
2017 2017
. .
After after
eating eat
some some
escargot escargot
for for
the the
first first
time time
, ,
Joe Joe
said say
, ,
" "
That that
was be
delicious delicious
! !
" "
He he
sent send
a a
postcard postcard
to to
his he
sister sister
Jane Jane
Smith Smith
. .
After after
hearing hear
about about
Joe Joe
's 's
trip trip
, ,
Jane Jane
decided decide
she she
might might
go go
to to
France France
one one

day day

. .

Voici comment vous devez remplacer tous les signes de ponctuation par des espaces, et remplacer les espaces multiples par un seul, et comment utiliser le module pour traiter chaque fichier dans le document.

```
File folder = new File(dir);
File[] listOfFiles = folder.listFiles();
for (File file : listOfFiles)
        {
            if(file.isFile())
            {
                BufferedReader br=new BufferedReader(new FileReader(new
File(dir+"/"+file.getName())));
                StringBuffer word=new StringBuffer();
                String line;

                while((line=br.readLine())!=null)
                {
                    String newline=line.replaceAll("[^'a-zA-Z0-9]"," ");
                    String finalline=newline.replaceAll("\\s+"," ").trim();
                    // set up pipeline properties
                    Properties props=new Properties();
                    // set the list of annotators to run
                    props.setProperty("annotators","tokenize,pos,lemma");
                    // set a property for an annotator, in this case the
coref annotator is being set to use the neural algorithm
                    props.setProperty("coref.algorithm","neural");
                    // build pipeline
                    StanfordCoreNLP pipeline=new StanfordCoreNLP(props);
                    // create a document object
                    CoreDocument document=new CoreDocument(finalline);
                    // annnotate the document
                    pipeline.annotate(document);
                    //System.out.println(document.tokens());
                    for(CoreLabel tok:document.tokens()){
                        //System.out.println(String.format("%s\t%s",
tok.word(), tok.lemma()));
                        String str=String.valueOf(tok.lemma());
                        if(!(str.contains("'s")||str.contains("’s"))){
                            word.append(str).append(" ");
                        }
                    }
                }
                String str=String.valueOf(word);
                str=str.replaceAll("[^a-zA-Z0-9]"," ").replaceAll("\\s+","
").trim();
//                  now str is a string which has the content of the read file
but it is processed and their words are space-separated. However there maybe
some details which has not been cleaned very well, just follow these steps to
clean the text.
//                  in the following you can continue your own implementation
            }
        }
```