TP2

IFT2015

Data structures

Fall 2023

In this TP, you will implement two different `Map` classes to serve basic natural language processing (NLP) functions. Particularly, you will constitute datasets of numerous documents. You will use the first `Map` to store the words of these documents (Word `Map`). The keys for the Word `Map` will be the words themselves, and the values will be references to the second type of `Map`s (File `Map`s), which keep the word occurrences in the files. The keys for the File `Map`s will be the file names, and the values will be a `List` of the positions of the associated words in the corresponding files.

Consider the following `dataset` which contains three documents (PS. a real `dataset` will contain many more files and longer texts). The files of a dataset are stored in a directory.

`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.
```

Figure 1 shows a fraction of the data structure for this example.

- Before building the data structure, you will need to preprocess the text by replacing all punctuation marks by spaces, replacing multiple spaces by one, and do some NLP text processing (tokenize, pos, and lemmatize the text) as all are described in the instruction.
- In order to create your Word `Map` and File `Map`s you have to implement the `Map` interface by overriding its necessary functions. You are allowed to define any additional functions of your own.
- You can use the `hashcode()` method of the `Object` class, or override it.

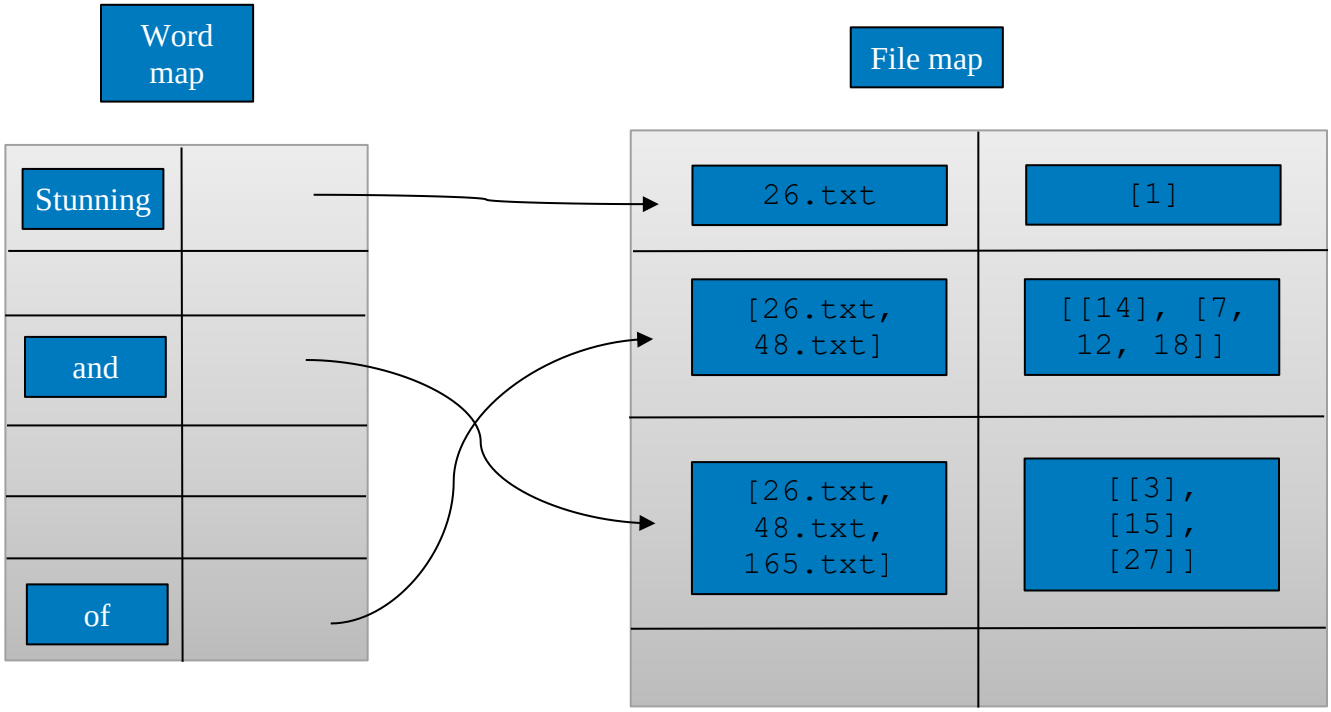


Figure1

- You must implement your load factor management so that if the insertion of a word causes the load factor to go above 0.75, then you should resize the capacity of the word `Map` by 2 × the previous size + 1.

Once your data structure is completed, you will implement two essential operations used in NLP:

1. suggesting the next most probable word like an auto-complete function which is related to the concept named **bi-grams** in NLP;
2. finding the most relevant document for a query (it can be more than one word) search which is related to the concept named **TFIDF** (term frequency–inverse document frequency) in NLP.

Bigrams

A bigram is a segment of text consisting of two consecutive words that occur in a provided text at least once. Bi-grams are very informative means to demonstrate the semantic associations between words. As an example, the bigrams in the following text: "*Photographer Phil Borges shows rarely-seen images of people from the mountains of Dharamsala, India*" are:

Photographer Phil

Phil Borges

Borges shows

shows rarely

rarely seen

seen images

…

Dharamsala, India

Bigrams can be used to suggest the next most probable word that can appear after a typed word in a search engine, and to improve the prediction of an auto-completion system.

Consider the following seven texts:

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.

Suppose after training, our model learns the occurrences of every two-words to determine the most probable one, W_2, after another, W_1. For example, from the sentences 2, 3, 6, and 7, after the word "**really**" the words" appreciate", "like", or "sorry" occur.

Let's calculate the probability of observing the second word, W_2, occurring after word W_1. We apply the relation of the conditional probability:

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

The probability of word W_2 given the preceding word W_1, $P(W_2 | W_1)$, equals the count of their bigram, or the co-occurrence of the two words, $C(W_1, W_2)$ divided by the count of W_1, $C(W_1)$.

From our example, let's find the word that has the highest probability to appear after the word "**really**". We need C("**really**") = 4, and C("**really**", "appreciate") = 1, C("**really**", "like") = 2, and C("**really**", "sorry") = 1. The most probable word to come after "**really**" is "like", with P("like" | "**really**") = 0.5, C("**really**", "like") / C("**really**") = 2/4 = 0.5. The probabilities P("appreciate" | "**really**") = P("sorry" | "**really**") = 0.25.

To the query "*the most probable bigram of really*" your program, given the above dataset, should suggest the next most probable word that can occur after the word "**really**", or the most probable bigram of "**really**", which is the word "like". *If there are two words or more with the same probability, return the smallest word based on the lexicographic order.*

TFIDF (Term frequency-inverse document frequency)

This is a score that reflects the importance of a word in a document. In NLP, a word is effective for a file to be categorized if it occurs frequently in that file but has very few occurrences in other texts in the dataset. To calculate TFIDF, we first calculate the word (term) frequency:

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

where `count(w)` is the number of times `w` appears in a document, and `totalW` is the total number of words in the document (length in terms of words). Then the inverse document frequency is calculated to weigh down the words that are frequent also in other files while scale up the rare ones:

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

where `totalD` is the total number of documents considered, and `count(d,w)` is the number of documents with `w` in it. And, finally, the TFID is calculated by:

$$\texttt{TFIDF}_d\texttt{(w) = TF}_d\texttt{(w)} \times \texttt{IDF(w)}$$

Now for each word in the query you have to sum the calculated $TFIDF_d(w_i)$ and do this for all documents, which give you a score for each document.

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

Imagine a search engine that uses the above scoring system to rank documents based on a provided query. You will be given a query and you should propose the most relevant document in the dataset that ranks first based on it. *If there are two or more documents with the same `score` return the one with the smallest document name based on the lexicographic order*.

Consider the following dataset of four 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.

Documents after being processed are like below:

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

Imagine a browser asked to find the most relevant document given word "**planet**": `search planet`. You will have to compute the TFIDF of the word "**planet**" in each document of the dataset:

```
TFIDF("planet") in document 900 = TF("planet")xIDF("planet") = 0.045
TF("planet") = 3/80
IDF("planet") = 1+ln((1+4)/(1+3))

TFIDF("planet") in document 901 = TF("planet")xIDF("planet") = 0.026
TF("planet") = 1/47
IDF("planet") = 1+ln((1+4)/(1+3))

TFIDF("planet") in document 902 = TF("planet")xIDF("planet") = 0.046
TF("planet") = 3/79
IDF("planet") = 1+ln((1+4)/(1+3))

TFIDF("planet") in document 903 = TF("planet")xIDF("planet") = 0.0
TF("planet") = 0
IDF("planet") = ln(4/3)
```

Therefore, the most relevant document about the word "planet" is document `902` with the optimal TFIDF score of 0.046.

EDIT Query:

There is also a problem of correcting spelling errors in the queries. For instance, we may wish to retrieve document containing the term carrot when the user types the query carot.

So before performing any of the two above queries you should first perform spell correction on each word of the query. For this query: ***the most probable bigram of <word>*** you must perform the query on the corrected word after spelling correction. And for this query **search *<word1><space><word2><space><word3>…*** you must perform spell correction first on each of the <word> and replace them with their corrected one and then perform the given query.

In order to correct spelling errors, you have to implement the following function.
The **Levenshtein distance,** also called the **Edit distance**, between two words is the minimum number single-character operations required to transform one string to another.

Typically, three types of operations are performed (one at a time):

- Replace a character.
- Delete a character.
- Insert a character.

Example:

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

str1 is converted to str2 by replacing 'g' with 'o', deleting the second
'o', and inserting 'f' at the beginning. There is no way to do it with fewer
than three edits.
```

So for each word in the given query, you have to calculate its distance among all words of all the documents that you have already pre-processed and picked the one with minimum distance (If there are more than one words with minimum distance pick the one that is alphabetically smaller)

What your program should do

Your program will read an input file that consists of multiple queries (see below). You don't need to pre-process words of the query as you did for documents. You will write your answers, line by line on a file called `solution.txt`. There are two types of queries. One is for suggesting the next most probable word that appear after a given word:

the most probable bigram of <word>

The second is for retrieving the most relevant document of a given query:

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

Input (two file names)

1. The directory name of the dataset, which contains the documents (all in English). This directory will be stored at the level of your `Java` project;
2. The query file name, where each line is asking for one of the two query types above. Note that the query file will also be stored at the level of your `Java` project.

It is not required to use command line on the terminal to run your code. you need to just manually give the "`query.txt`" and "`dataset`" as the argument to your program instead of using terminal to give these arguments.

Output (full example)

You should perform the queries and write the answers, line by line, on `solution.txt`. Given the directory `dataset` and query file `query.txt`:

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

```
search graveety
the most probable bigram of dwarf
```

your program will output on `solution.txt`:

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

Java code and submission

- You may form teams of two or less programmers
- You must follow the rules of OOP; use `CamelCase` for your identifiers; and comment your code
- All `Java` code files and input files must be in the same directory: no packages. The class that contains the main function must be called `Main.java`
- Submit your directory as an archive, only TP2_f1_f2.tgz orTP2_f1_f2.zip formats are acceptable, where f1 and f2 are the family name of the first and second member of the team, or if you are alone use TP2_f.tgz or TP2_f.zip), where f is your family name.
- You have until Saturday December 16 at 23h59 to submit your solution on `StudiUM`
- A penalty of 20% per day starting on December 10 at oohoo will be applied if you submit after the submission deadline.
- You must follow the exact output format of one answer per line on the `solution.txt`, file corresponding to each line in the query file.

Evaluation

- **Compilation** (10%): The code must compile successfully to produce executable code; failure to do so will result in a zero score for this criterion.
- **Execution on Provided Datasets** (10%):
  Does not crash: 5%
  Produces correct results: 5%
  This criterion assesses the program's stability and accuracy on known datasets.
- **Execution on Unknown Datasets** (60%):
  - Does not crash: 10%
  - Produces correct results: 50%

    - This criterion assesses the program's robustness and accuracy in handling unforeseen or new datasets.
- **Code Quality** (20%):
    - Readability and Comments: 10%
    - Object-Oriented Structure: 10%
    - This criterion evaluates the overall cleanliness, structure, and documentation of the code.

NB. Don't forget to include your names and numbers in all your Java classes.

Questions. If you have questions, use TP2's forum on StudiUM, or contact one fo the TAs or the Prof.

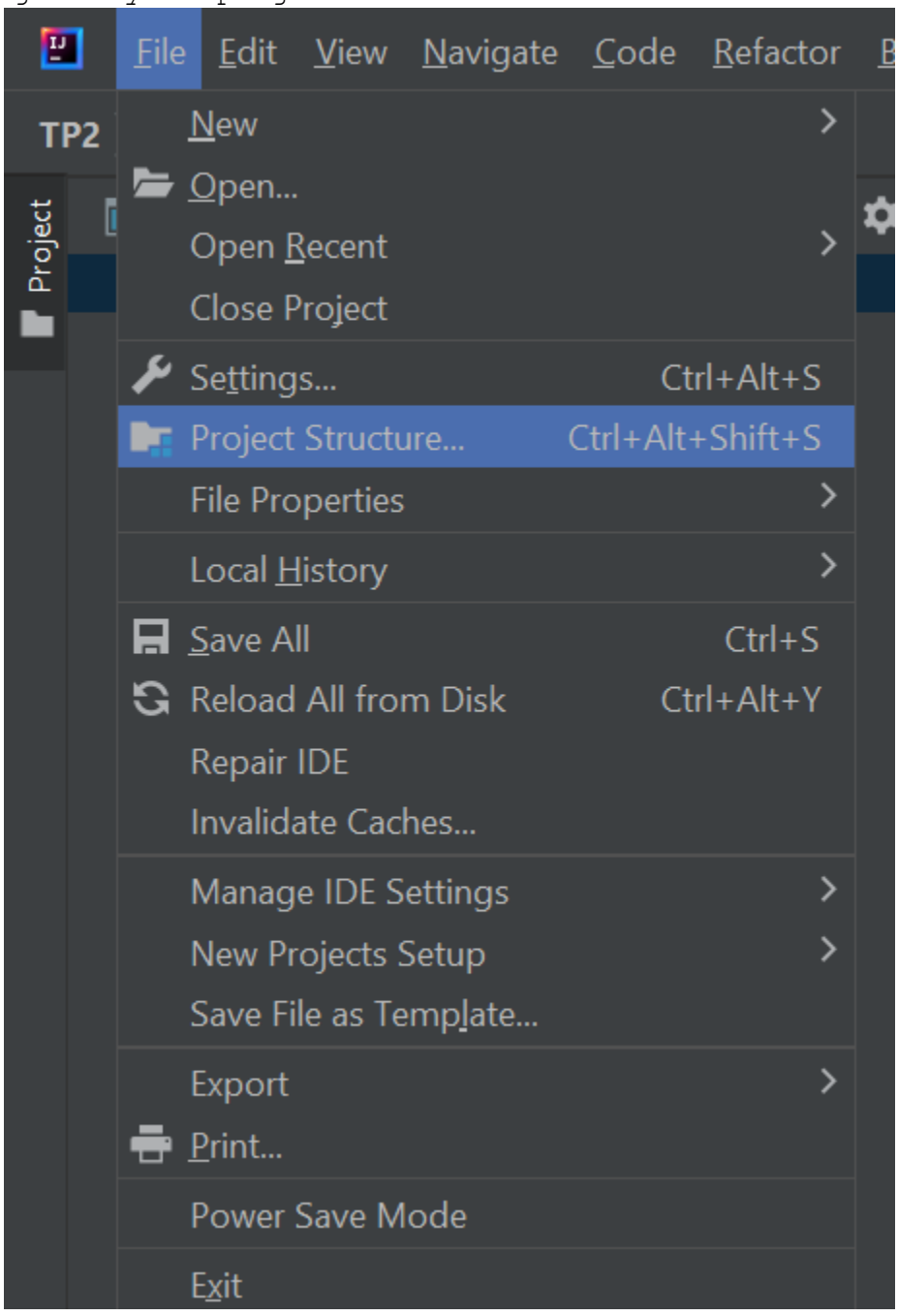
HAVE FUN & GOOD LUCK!

Instructions on adding and using CORENLP

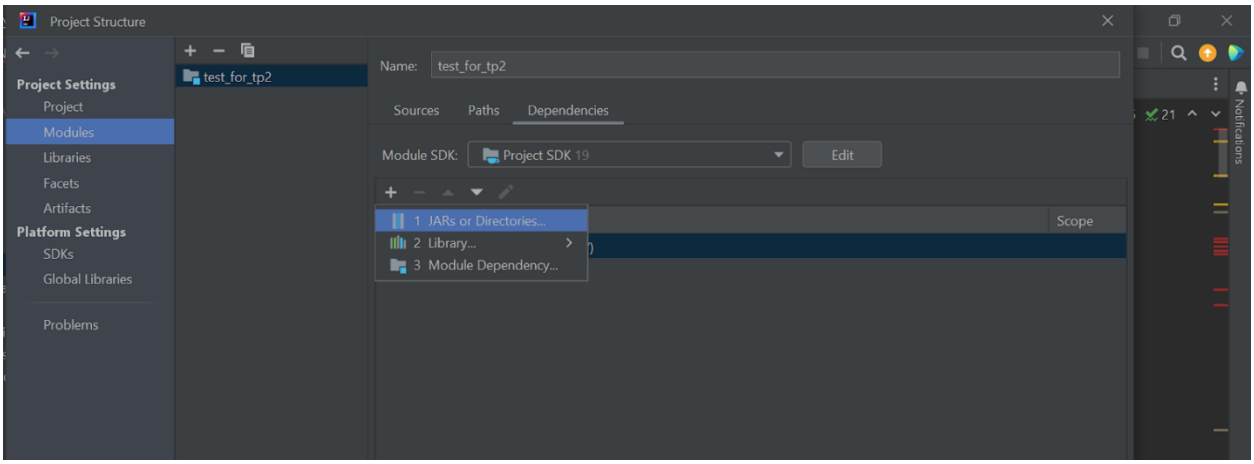
First download this zip file and then extract it.
https://downloads.cs.stanford.edu/nlp/software/stanford-corenlp-4.5.1.zip

Follow the steps below to use this module as the text processing part of your code:

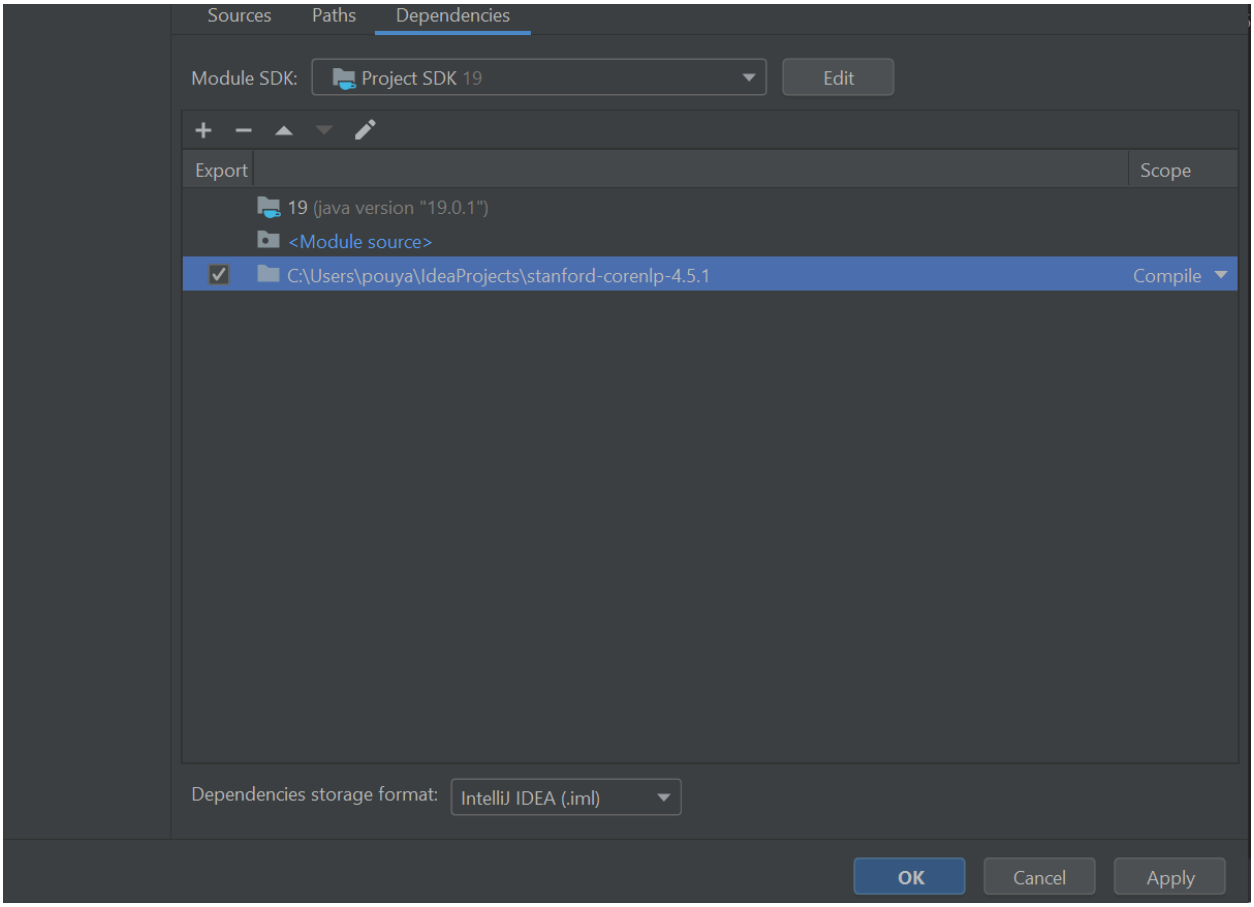
1. First go to your project structure:



2. In the module part click the + icon to add JARS or Directories:



3. Then choose the stanford-corenlp-4.5.1 Folder in which you unzip in the beginning and then click apply and ok at the bottom.



Now you are ready to use the built functions and methods of this module on your java project to do text processing.

How?

Add these three lines in the java file you are going to use from this module to import the necessary files:

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

Below is a simple example of how you are supposed to use this module on your documents which at the end prints how each word is processed. So you should do the same process for each line of each of your documents to do the text processing before calculating anything.

```
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()));
        }
    }
}
```

Output:

```
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

. .

Below is how you should replace all punctuation marks by spaces, and replace multiple spaces by a single one and how to use the module to process each file in the 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
            }
        }
```