TP2

IFT2015

Structures de données

Automne 2022

Dans ce TP, vous implémenterez deux classes `Map` différentes pour servir des fonctions de base en traitement du langage naturel (TLN). En particulier, vous constituerez des jeux de données avec de nombreux documents. Vous utiliserez la première `Map` pour stocker les mots de ces documents (`WordMap`). Les clés de la `WordMap` seront les mots eux-mêmes et les valeurs seront des références au deuxième type de `Map` (`FileMap`), qui conservent les occurrences de chaque mots dans les fichiers. Les clés des `FileMap` seront les noms de fichiers des documents et les valeurs seront une liste de positions des mots associés dans les fichiers correspondants (voir **Figure 1**).

Considérez l'ensemble de documents suivants, trois textes (PS. les vrais `dataset` contiendront beaucoup plus de fichiers et des textes plus longs). Les fichiers d'un `dataset` sont stockés dans un répertoire.

`0000026.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.
```

`0000048.txt:`

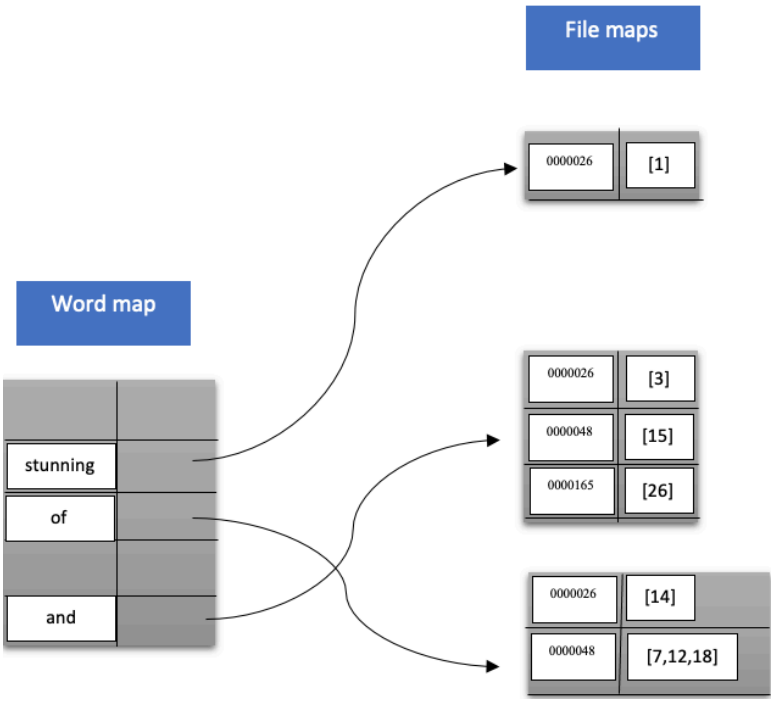
```
Photographer Phil Borges shows rarely-seen images of people from
the mountains of Dharamsala, India, and the jungles of the
Ecuadorean Amazon. In documenting these endangered cultures, he
intends to help preserve them.
```

`0000165.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 montre une fraction de la structure de données pour cet exemple.

- Avant de construire la structure de données, vous devrez pré-traiter le texte en remplaçant tous les signes de ponctuation par des espaces, en remplaçant plusieurs espaces par un seul et en convertissant tous les caractères en minuscules.
- Pour implémenter les `Map`, vous remplacerez les fonctions dans les fichiers fournis `WordMap.java` et `FileMap.java`, qui implémentent toutes les deux l'interface `Map`. Vous êtes autorisé à définir vos propres fonctions supplémentaires.



- Vous pouvez utiliser la méthode `hashcode()` de la classe `Object` ou la réécrire.
- Vous devez implémenter votre gestion du facteur de charge de sorte que si l'insertion d'un mot le fait passer au-dessus de 0,75, alors vous devez redimensionner la capacité de la `WordMap` de 2 × sa capacité précédente + 1.

Une fois votre structure de données terminée, vous mettrez en place deux opérations essentielles utilisées en TLN :

1. suggérer le prochain mot le plus probable comme une fonction d'auto-complétion, liée au concept qu'on appelle **bi-grammes** en TLN
2. trouver le document le plus pertinent pour une recherche par mot-clé, lié au concept **TFIDF** (fréquence de terme – fréquence de document inverse) en TLN.

Bi-grammes

Un bi-gramme est un segment de texte composé de deux mots consécutifs apparaissant au moins une fois dans un des textes fouris. Les bi-grammes sont des moyens très informatifs pour démontrer les associations sémantiques entre les mots. Par exemple, les bi-grammes 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 bi-grammes peuvent être utilisés pour suggérer le prochain mot 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 un entraînement, notre modèle apprend les occurrences de tous les deux mots pour déterminer le mot le plus probable, W_2, après un 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, survenant après le mot W_1. On applique la relation de la probabilité conditionnelle :

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

La probabilité du mot W_2 compte tenu du mot précédent W_1, $P(W_2 | W_1)$, est égale au nombre de leur bi-gramme, ou la co-occurrence des deux mots, $C(W_1, W_2)$ divisé par le nombre de W_1, $C(W_1)$.

À partir de notre exemple, trouvons le mot qui a la plus grande probabilité d'apparaître après le mot "**really**". Nous avons besoin de C("**really**") = 4, et C("**really**", "appreciate") = 1, C("**really**", "like") = 2, et C("**really**", "sorry") = 1. Le mot le plus probable après "**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, étant donné l'ensemble des textes ci-dessus, devra suggérer le prochain mot le plus probable qui peut apparaître après le mot "**really**", ou le bi-gramme 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 en fonction de l'ordre lexicographique.

TFIDF (fréquence de terme – fréquence de document inverse)

C'est un score qui reflète l'importance d'un mot dans un document. En TLN, un mot est efficace pour qu'un fichier soit catégorisé s'il apparaît fréquemment dans ce fichier, mais a très peu d'occurrences dans d'autres textes de l'ensemble de données. Pour calculer la TFIDF, nous calculons d'abord la fréquence du mot (ou terme) :

```
TF(w) = count(w)/totalW
```

où `count(w)` est le nombre de fois que `w` apparaît dans un document, et `totalW` est le nombre total de mots dans le document (longueur du document en mots). Ensuite, la fréquence inverse des documents est calculée pour peser les mots qui sont également fréquents dans d'autres fichiers tout en augmentant celui des mots rares :

```
IDF(w) = ln(totalD/count(d,w))
```

où `totalD` est le nombre total de documents considérés, et `count(d,w)` est le nombre de documents contenant `w`. Et, enfin, le TFID est calculé par :

```
TFIDF(w) = TF(w)×IDF(w)
```

Imaginez un moteur de recherche qui utilise la TFIDF pour classer les documents en fonction d'un mot fourni. Vous recevrez un mot et vous devrez proposer le document le plus pertinent dans l'ensemble des textes qui se classera premier en fonction de celui-ci. PS. Lorsqu'un mot n'est présent dans aucun document de l'ensemble de données, `TF(w) = 0` et `IDF(w) = infinity`, alors considérez `TDIDF(w) = 0`. *S'il y a deux documents ou plus avec la même* `TFIDF(w)`*, retournez celui avec le plus petit nom de fichier, basé sur l'ordre lexicographique.*

Considérez l'ensemble suivant de quatre documents :

`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.
```

`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.

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.

Imaginez un navigateur cherchant le document le plus pertinent étant donné le mot "**planet**" : `search planet`. Pour le trouver, vous devrez calculer le TFIDF du mot "**planet**" dans chaque document du jeu de données :

TFIDF("**planet**") dans document 902 = TF("**planet**")xIDF("**planet**") = 0.0109

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

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

TFIDF("**planet**") dans document 900 = TF("**planet**")xIDF("**planet**") = 0.0109

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

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

TFIDF("**planet**") dans document 901 = TF("**planet**")xIDF("**planet**") = 0.0061

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

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

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

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

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

Par conséquent, le document le plus pertinent sur le mot "**planet**" est le document `900` avec le score TFIDF optimal de 0,0109. Ce score est le même pour le document `902`, mais dans l'ordre lexicographique `900` vient avant `902`.

Qu'est-ce que votre programme doit faire

Votre programme lira un fichier d'entrée composé de plusieurs requêtes (voir ci-dessous). Vous écrirez vos réponses, une par requête et par ligne sur `stdout`. Il existe deux types de requêtes. L'une consiste à suggérer le prochain mot le plus probable qui apparaît après un mot donné :

the most probable bigram of <word>

La seconde permet de récupérer le document le plus pertinent d'un mot donné :

search <word>

Input (2 noms de fichiers)

1. Le nom du répertoire de l'ensemble de documents. Ce répertoire sera stocké au niveau de votre projet `Java` ;
2. Le nom du fichier de requête, où chaque ligne demande l'un des deux types de requête ci-dessus. Notez que le fichier de requête sera également stocké au niveau de votre projet `Java`.

Output (exemple complet)

Vous devez effectuer les requêtes et écrire les réponses, une par ligne, sur `stdout`. Étant donné le répertoire `dataset2` et le fichier de requête `query.txt` :

```
search planet
the most probable bigram of new
search species
```

la ligne de commande : `java Main dataset2 query.txt`, votre programme affichera sur `stdout` :

```
900.txt
new york
63.txt
```

Code Java et soumission

- Vous pouvez former des équipes de deux programmeurs ou moins
- Vous devez suivre les règles de la POO ; utilisez `CamelCase` pour vos identificateurs ; et commentez votre code
- Tous les fichiers de code `Java` doivent se trouver dans le même répertoire : pas de packages. La classe qui contient la fonction `main` doit s'appeler `Main.java`

- Pour une soumission sur StudiUM, archivez votre répertoire (seuls les noms `TP2.tar.gz` ou `TP2.zip` seront autorisés)
- Vous avez jusqu'au dimanche 9 décembre à 23h59 pour soumettre votre solution sur StudiUM
- Une pénalité de 20% par jour à partir du 10 décembre à 00h00 sera appliquée si vous soumettez après la date limite.
- Vous devez suivre le format de sortie exact d'une réponse par ligne sur la sortie standard, correspondant à chaque ligne du fichier de requête.

Evaluation

- 10% si votre code se compile correctement sans aucun argument
- 10 % si votre code s'exécute et trouve les résultats corrects sur les exemples résolus fournis
- 40 % si votre code s'exécute et trouve les résultats corrects sur des exemples non vus
- 20 % si votre code est efficace (comment cela sera fait reste à déterminer)
- 20 % si votre code suit les principes de la POO, est lisible et contient des commentaires appropriés.

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

Questions. Si vous avez des questions, utilisez le forum du TP2 sur StudiUM, ou contactez un de vos démonstrateurs ou le Prof.

AMUSEZ-VOUS ET BONNE CHANCE !